Book

A Simplified Approach
to

Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

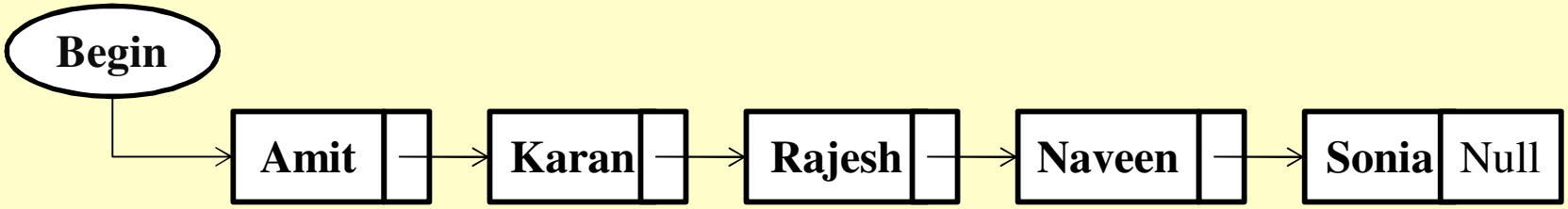
CIRCULAR LINKED LIST AND DOUBLY LIST

Contents for Today's Lecture

- Circular Linked list
- Two-Way Linked List

3.4 Circular Linked List

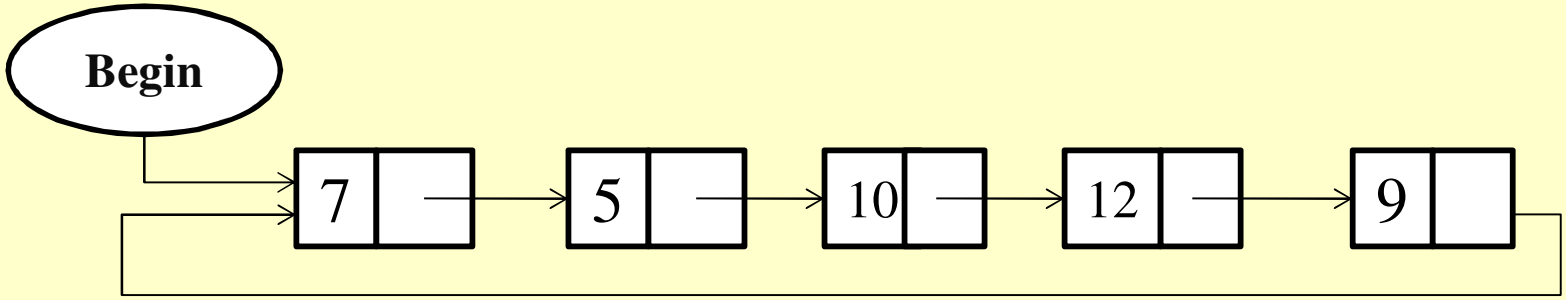
Linked List: A Linked List refers to a linear collection of data elements in which linear order is not given by their physical placement n memory (as in case of array). In linked list, the data elements are managed by collection of nodes, where each node contains link or pointer which points to the next node in the list. The beginning f the linked list is maintained by a special pointer variable which contains the address of the first node in the list. The link part of the last node contains a special value called ***Null*** which shows the end of the list.



Representation of Data using Linked list

3.4 Circular Linked List(continued)

A ***Circular Linked List*** is a list in which last node points back to the first node instead of containing the Null pointer in the next part of the last node. The circular linked list can be shown diagrammatically



A CIRCULAR LINKED LIST

3.4 Circular Linked List(continued)

All the operations which can be performed on ordinary singular linked list can easily be performed on circular linked list with the following changes:

•Looking for the end of the linked list -In the case of one way singular linked list, the next part of the last node will contain ***Null*** address but in the case of circular linked list, the next part of the last node consist of address of the first node i.e. ***Begin.*** Thus for reaching at the end of the circular linked list, we will compare the address of the first node i.e. ***Begin*** with address stored in ***Next*** part of each node. If both the addresses come out to be same, then we have reached at the end of the circular list.

•When a new node is to be inserted at the end of the circular linked list, it's next part will contain the address of the first node instead of null as is in the case of one-way singular linked list.

3.4.1.*Traversal in Circular Linked List*

The traversal of circular linked list having list pointer variable ***Begin*** and a pointer variable ***Pointer*** to traverse the linked list from begin to end.

Algorithm : Traverses a circular linked list with pointer variable 'Begin'

Step1*:** If ***Begin =Null Then
Print: "Circular linked list is empty"
Exit
[End If]

Step2: Process ***Begin → Info***

3.4.1.Traversal in Circular Linked List(continued)

Step3: Set ***Pointer = Begin → Next***

Step4: Repeat steps 5 and 6 while ***Pointer ≠ Begin***

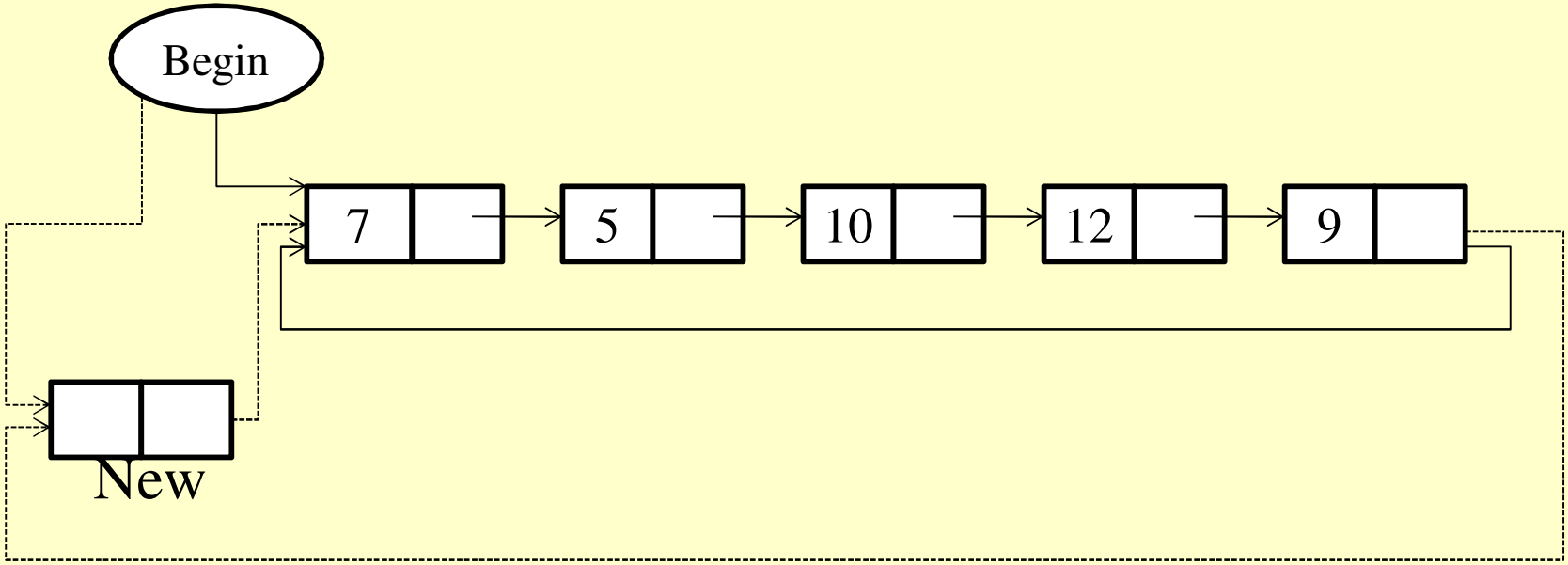
Step5: Process ***Pointer → Info***

Step6**:** Set ***Pointer = Pointer→ Next***
[End Loop]

Step7: Exit

3.4.2.*Insertion at the Beginning of Circular Linked List*

In this case, the ***New*** node is inserted as the first node and the next part of the last node is changed and now it points to the newly inserted node as shown below in figure:



Insertion of a New Node at the Beginning of a Circular Linked List

3.4.2.*Insertion at the Beginning of Circular Linked List(continued)*

Algorithm: Insertion of an element 'Data' at the Beginning of the Circular Linked List

Step1: If ***Free=Null*** Then
Print: "No free space available"
Exit
[End If]

Step2: Set ***New= Free*** and ***Free=Free → Next***

Step3: Set ***New →Info=Data***

3.4.2.*Insertion at the Beginning of Circular Linked List(continued)*

Step4: If ***Begin=Null*** Then
Set ***Begin=New***
Set ***New*** → ***Next=Begin***
Exit
[End If]

Step5: Set ***Pointer=Begin***

Step6: Repeat while ***Pointer*** →***Next*** ≠ ***Begin***
Set ***Pointer=Pointer*** → ***Next***
[End Loop]

3.4.2.Insertion at the Beginning of Circular Linked List(continued)

Step7: Set ***New → Next=Begin***

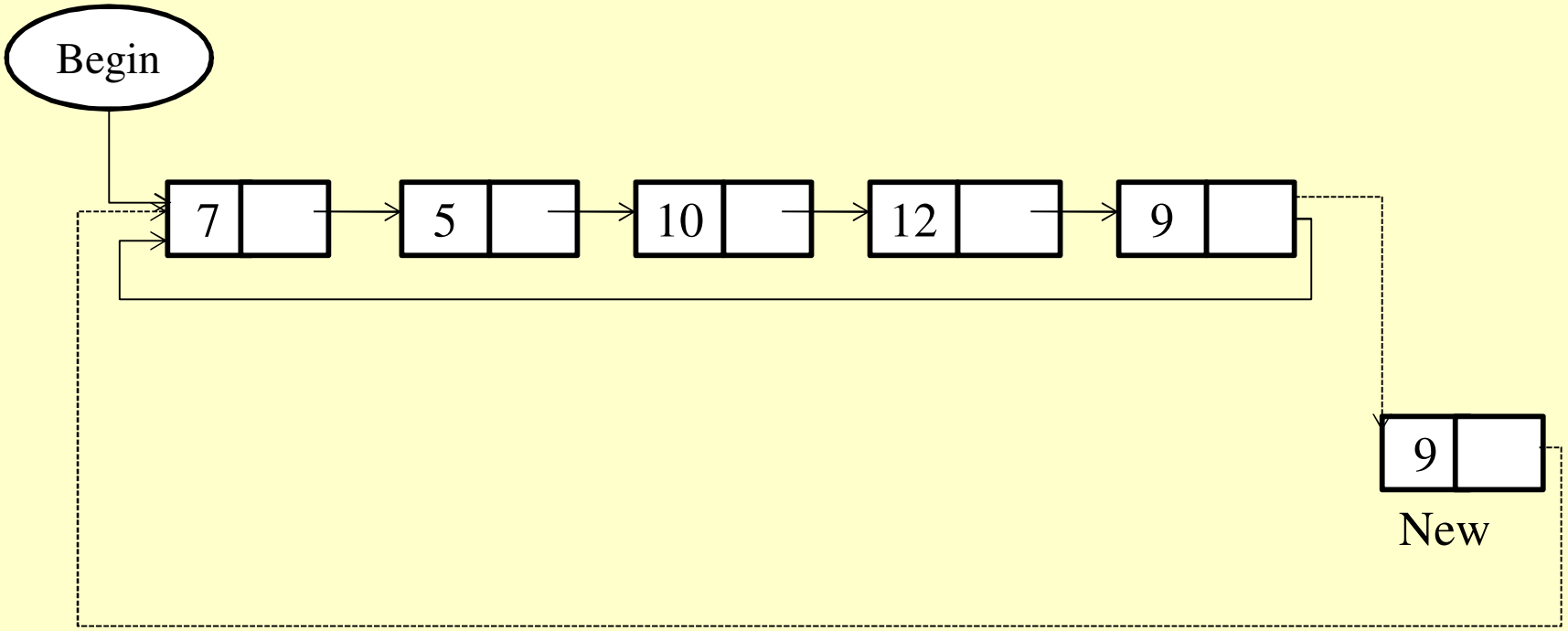
Step8: Set ***Begin=New***

Step9: Set ***Pointer → Next=Begin***

Step11: Exit

3.4.3. *Insertion at the End of the Circular Linked List*

In the process of inserting an element at the end of the circular linked list, the address stored in the ***Next*** part of the last node and next part of ***New*** node need to be changed as shown below:



Insertion of a Node 'New' at the End of the circular linked list

3.4.3. *Insertion at the End of the Circular Linked List(continued)*

Algorithm: Insertion of an element 'Item' at the end of the circular linked list

Step1: If ***Free=Null*** Then
Print: "No Free space available for Insertion"
Exit
[End If]

Step2: Allocate memory to node ***New***
Set ***New =Free*** and ***Free=Free → Next***

Step3: Set ***New→ Info=Item***

3.4.3. *Insertion at the End of the Circular Linked List(continued)*

Step4: If ***Begin=Null*** Then
Set ***Begin=New*** and ***New → Next=Begin***
Exit
[End If]

Step5: Set ***Pointer=Begin***

Step6: Repeat while ***Pointer → Next ≠ Begin***
Set Pointer=Pointer → Next
[End Loop]

Step7: Set ***Pointer → Next=New*** and ***New → Next=Begin***

Step8: Exit

3.4.4. Applications of Circular Linked List

Circular linked list can be used for:

• ***Implementing a time sharing problem of the operating system:***

The operating system must maintain a list of executing processes and must alternately allow each process to use a slice of CPU time, one process at a time and there should be ***no Null*** Pointer unless there is no process requesting CPU time.

3.5 Two-Way Linked List (Doubly Linked List)

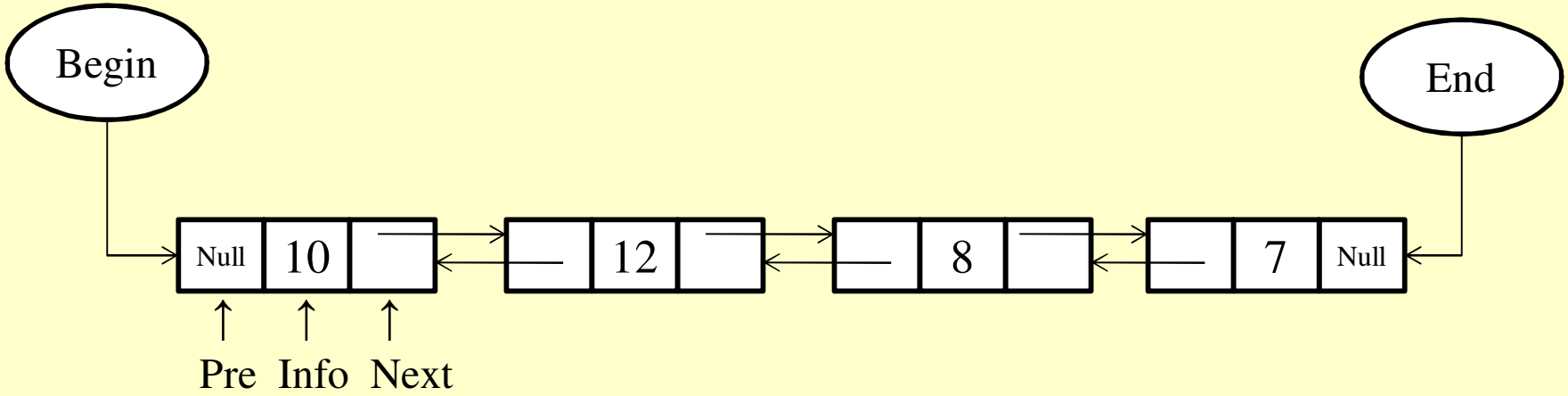
In Two-Way Linked List, we traverse the list in both the directions i.e. forward direction (from beginning to end) and in backward direction (from end to beginning). The Two-Way Linked List is also known as ***Doubly Linked List***. In Two-Way Linked List, each node is divided into three parts: ***Pre, Info, Next.*** The structure of a node used in Two-Way Linked List is as shown below:

Structure of a Node used in a Two-Way Linked List

Pre part contains the address of the preceding node
Info part contains the element
Next part contains the address of the Next node.

3.5 Two-Way Linked List (Doubly Linked List) (continued)

Here in Two-Way Linked List, two list Pointer variables i.e. ***Begin*** and ***End*** are used which contains the address of the first node and last node of the Linked List respectively . Two-Way Linked List can be shown diagrammatically as shown:



A Two-Way Linked List

The ***Pre*** part of the first node of a Two-Way Linked List will contain ***Null*** as there is no node preceding the first node and the ***Next*** part of last node will contain ***Null*** as there is no node following the last node.

3.5 Two-Way Linked List (Doubly Linked List) (continued)

Operations performed on Two-Way Linked List:

- Traversing
- Searching
- Insertion
- Deletion

3.5.1.Traversing a Two-Way Linked List

A Two-Way Linked List can be traversed in both the directions:

•***forward direction***, the Pointer variable will be assigned with the address stored in the ***Begin*** pointer variable and reach at node whose ***Next*** part contains ***Null*** i.e. we reach at the end of list.

• ***backward direction***, the Pointer variable will be assigned with the address stored in the ***End*** pointer variable and reach at node whose ***Pre*** part contains ***Null*** i.e. we reach at the beginning of the list. The variable ***Pointer*** keeps track of the address of the current node.

3.5.1.Traversing a Two-Way Linked List (continued)

Algorithm: Traverses a two-way linked list starting from the end of the list to the beginning

Step1: If ***End=Null*** Then
Print: "Linked List is empty"
Exit
[End If]

Step2: Set ***Pointer=End***

Step3: Repeat while ***Pointer ≠ Null***
Process ***Pointer → Info***
Set ***Pointer=Pointer → Pre***
[End Loop]

Step4: Exit

3.5.2. Searching in a Two-Way Linked List

To find the location of a given linked list:

•traverse the list either from end or beginning

•keep comparing the element stored in each node with the desired item

•if desired item is found then further traversing is stopped and address of the node containing the desired element is returned.

3.5.2. Searching in a Two-Way Linked List(continued)

Algorithm: To find the position of a given element 'data' in a Two-Way Linked List by traversing it from end to beginning.

Step1: If End=Null Then
Print: "Linked List is empty"
Exit
[End If]

Step2: Set ***Pointer=End***

3.5.2. Searching in a Two-Way Linked List(continued)

Step3: Repeat while ***Pointer ≠ Null***

If ***Pointer → Info=Data*** Then

Print: "Element ***Data*** is found at address":Pointer

Exit

Else

Set **Pointer=Pointer → Pre**

[End If]

[End Loop]

Step4: Print: "Element **Data** is not found in the linked list"

Step5: Exit

3.5.3. Insertion of an element in a Two-Way Linked List

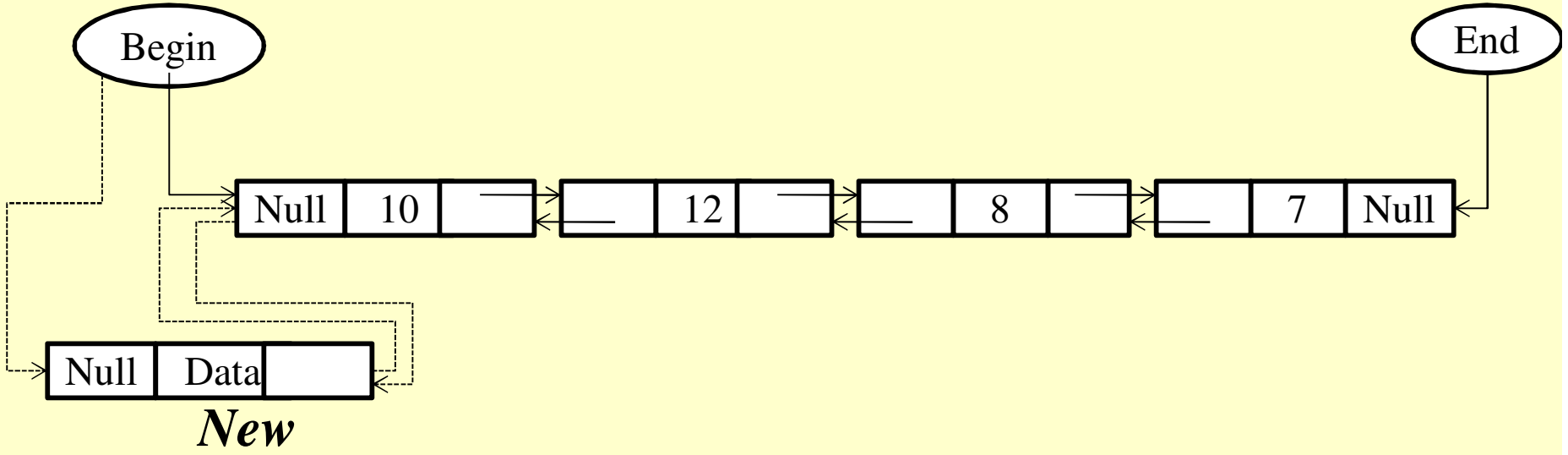
Insertion can take place at various positions in a linked list such as:

•at beginning

•at the end or after any particular node in a linked list: It requires searching the location of the node after which new node is to be inserted.

3.5.3.1. Inserting a New node at the Beginning of a Two-Way Linked List

An element ***Data*** is to be inserted at the beginning of the doubly linked list.



Insertion of a node at the Beginning in a Doubly linked List

3.5.3.1. Inserting a New node at the Beginning of a Two-Way Linked List(continued)

Algorithm: To insert a New node at the Beginning of a Two-Way Linked List

Step1: If ***Free=Null*** Then
Print: "Free space not available"
Exit
[End If]

Step2: Allocate memory to node ***New***
(Set ***New=Free*** and ***Free= Free*** → ***Next***)

Step3: Set ***New*** → ***Pre=Null*** and ***New*** → ***Info=Data***

3.5.3.1. Inserting a New node at the Beginning of a Two-Way Linked List(continued)

Step4: If ***Begin=Null*** Then

Set ***New*** → ***Next=Null*** and ***End=New***

Else

Set ***New*** → ***Next=Begin*** and ***Begin*** → ***Pre=New***

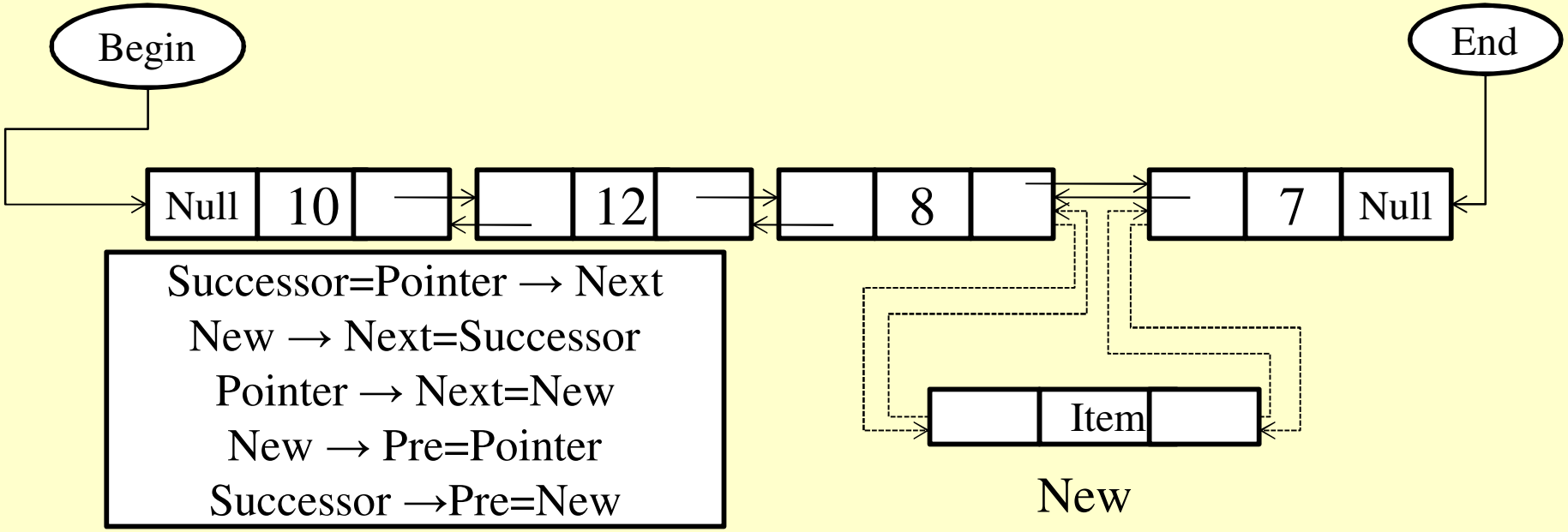
[End If]

Step5: Set ***Begin=New***

Step6: Exit

3.5.3.2. Inserting a New node after a particular node in a Two-Way Linked List

Insertion after a particular node requires finding the location of the node after which new node is to be inserted. After finding the desired node, the New node can be inserted easily by changing few pointers as shown:



Insertion of a node 'New' in the Linked List after a particular element 'Data'

3.5.3.2. Inserting a New node after a particular node in a Two-Way Linked List(continued)

Algorithm: insert a New node 'Item' after a given element 'Data' in the Two-Way Linked List

Step1: If ***Free=Null*** Then
Print: "Free space not available"
Exit
[End If]

Step2: Begin=Null Then
Print: "List is Empty, No insertion will take place"
Exit
[End If]

3.5.3.2. Inserting a New node after a particular node in a Two-Way Linked List(continued)

Step3: Set ***Pointer = Begin***

Step4: Repeat while ***Pointer → Next ≠ Null***
and ***Pointer→ Info ≠ Data***
Pointer=Pointer → Next
[End Loop]

Step5: If ***Pointer → Next=Null*** and ***Pointer → Info ≠ Data***
Print: "Item cannot be inserted as element Data is not present"
Exit
[End If]

3.5.3.2. Inserting a New node after a particular node in a Two-Way Linked List(continued)

Step6: Allocate memory to node New
(Set ***New=Free*** and ***Free=Free → Next***)
Set ***New → Info=Item***

Step7: If ***Pointer → Next ≠ Null*** Then
Successor=Pointer → Next
New → Next=Successor
Pointer → Next=New
New → Pre=Pointer
Successor → Pre=New

3.5.3.2. Inserting a New node after a particular node in a Two-Way Linked List(continued)

Else

New → Next=Null

New → Pre=Pointer

Pointer → Next=New

End=New

[End If]

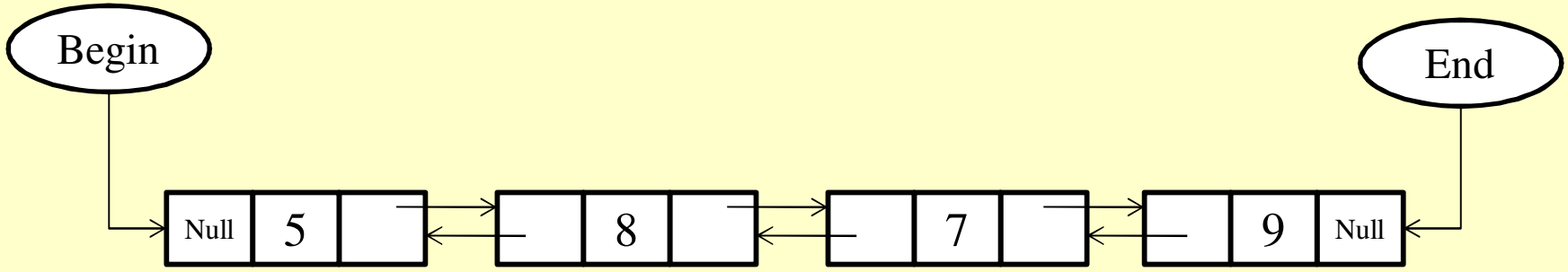
Step8: Exit

3.5.4. Deleting a node with given Item from 2-Way linked List

For deleting a particular node:

•traverse the list either in forward or backward direction to locate the node containing the element to be deleted

•if the desired node is found, it can be removed from the linked list by changing few pointers as shown:

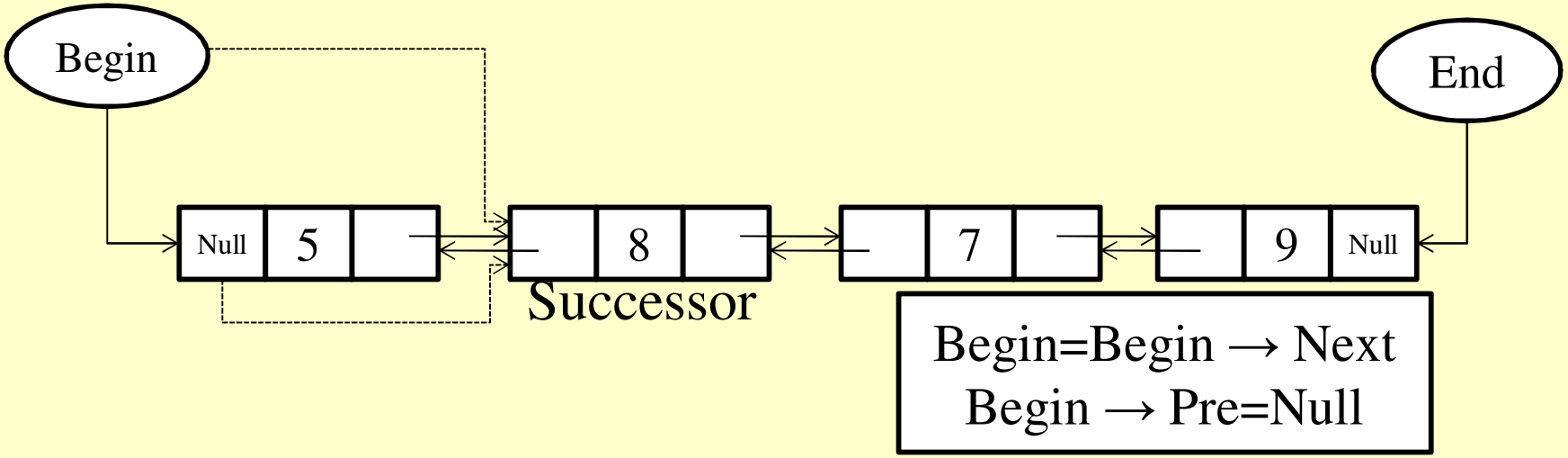


A Two-Way Linked List with 4 nodes

3.5.4. Deleting a node with given Item from 2-Way linked List(continued)

•if the desired node is not found and we reach at the end of a list then an appropriate message is displayed.

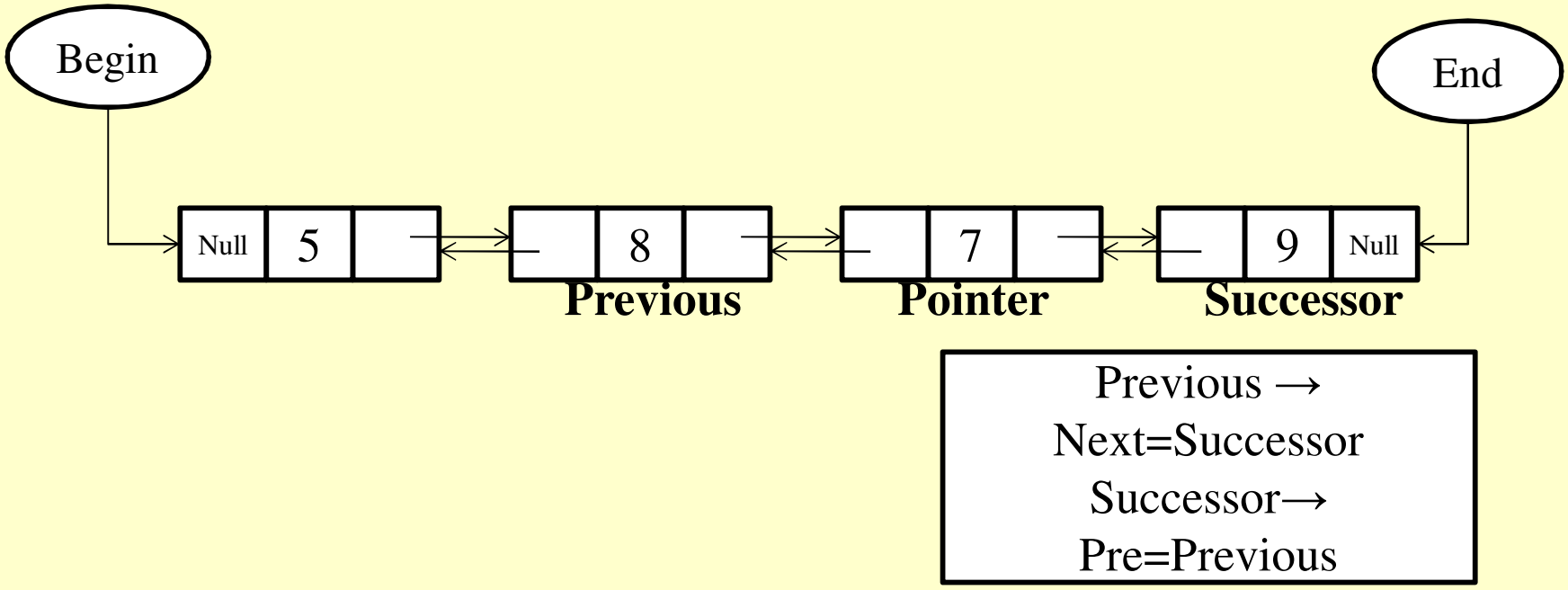
CASE1: Suppose we want to delete an element 5, which is contained in the first node of the list. Deletion will be performed as shown in the figure below:



Deleting the 1st node of a Two-Way linked List

3.5.4. Deleting a node with given Item from 2-Way linked List(continued)

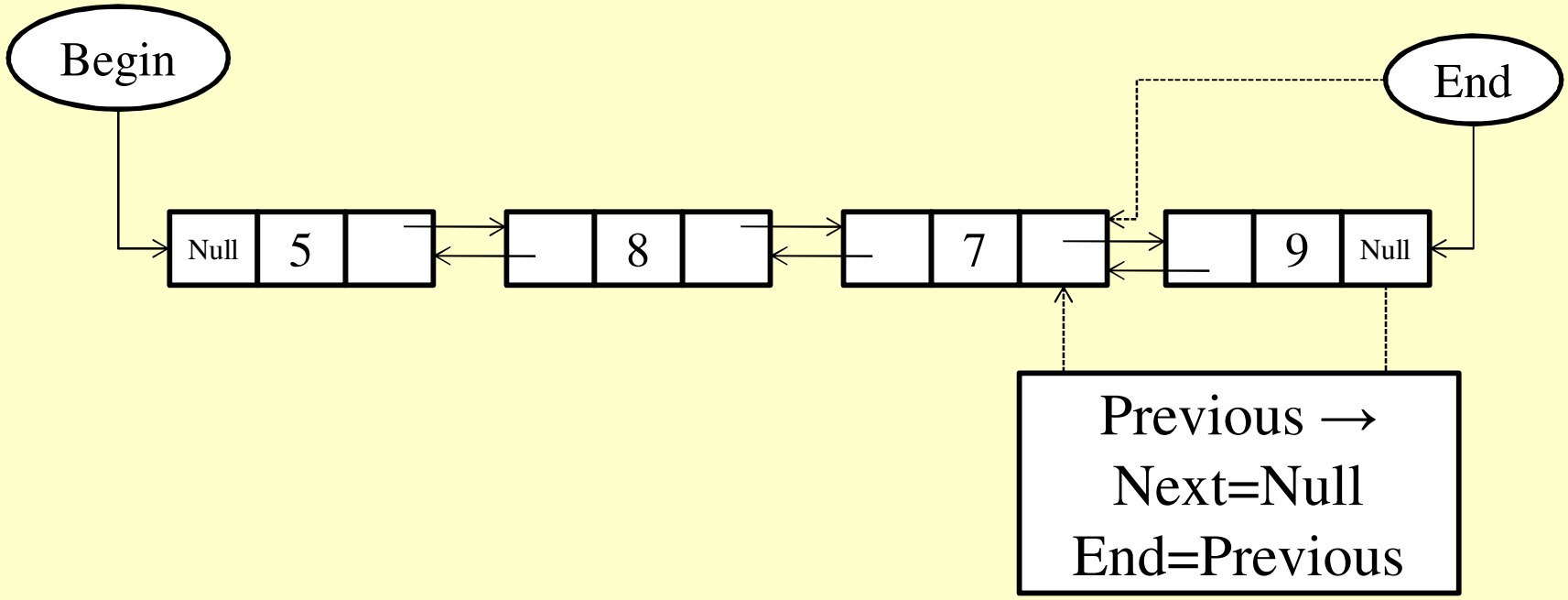
CASE2: Suppose we want to delete an element 7, which is contained in a node that lies between the first and last node of the list then deletion will be performed as shown in the figure below:



Deleting a Node present between the first and the last node of a Two-Way Linked List

3.5.4. Deleting a node with given Item from 2-Way linked List(continued)

CASE3: Now we will delete an element 9 which is encountered in the node which is last node of the list as shown:



Deleting the last node from a Two-Way Linked List

3.5.4. Deleting a node with given Item from 2-Way linked List(continued)

Algorithm: Delete a node containing an element 'Item' from a Two-Way Linked List

Step1: If ***Begin=Null*** Then
Print: "Linked List is already empty"
Exit
[End If]

Step2: If ***Begin → Info=Item*** Then
Pos=Begin
Begin=Begin → Next
Begin → Pre=Null

3.5.4. Deleting a node with given Item from 2-Way linked List(continued)

//Deallocate memory held by Pos

Pos → Next=Free, Free=Pos

Exit

[End If]

Step3: Set ***Pointer=Begin → Next***

Step4: Repeat while ***Pointer → Next ≠ Null***

and ***Pointer → Info ≠ Item***

Set ***Pointer=Pointer → Next***

[End Loop]

3.5.4. Deleting a node with given Item from 2-Way linked List(continued)

Step5: If ***Pointer → Next=Null*** and ***Pointer → Info ≠ Item***
Print: " Item to be deleted not found"
Exit
[End If]

Step6: If ***Pointer → Next=Null*** Then
//last node to be deleted
Set ***Previous=Pointer → Pre***
Set ***Previous → Next=Null***
Set ***End=Previous***