Linked List

ESC108A Elements of Computer Science and Engineering

B. Tech. 2017

Course Leaders:

Roopa G.

Ami Rai E.

Chaitra S.

Objectives

- At the end of this lecture, student will be able to
  - use the structure and operations of a ***singly linked list*** data structure
  - use the structure and operations of a ***doubly linked list*** data structure

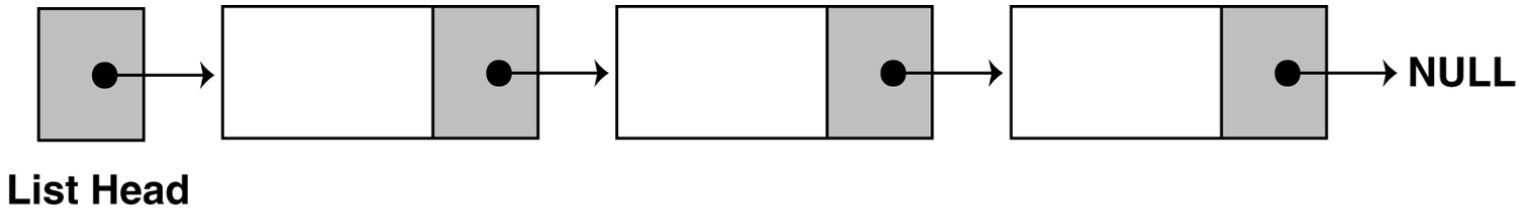
Contents

- Singly Linked List
- Doubly Linked List

Linked List

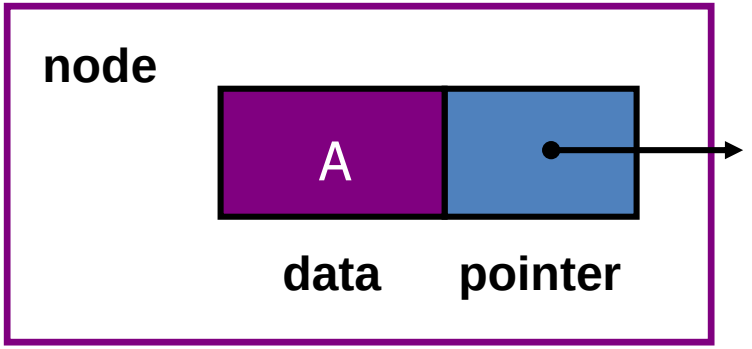
- A linked list can grow or shrink in size as the program runs
- Does not require the shifting of items during insertions and deletions

- A linked list is called "linked" because each node in the series has a pointer that points to the next node in the list

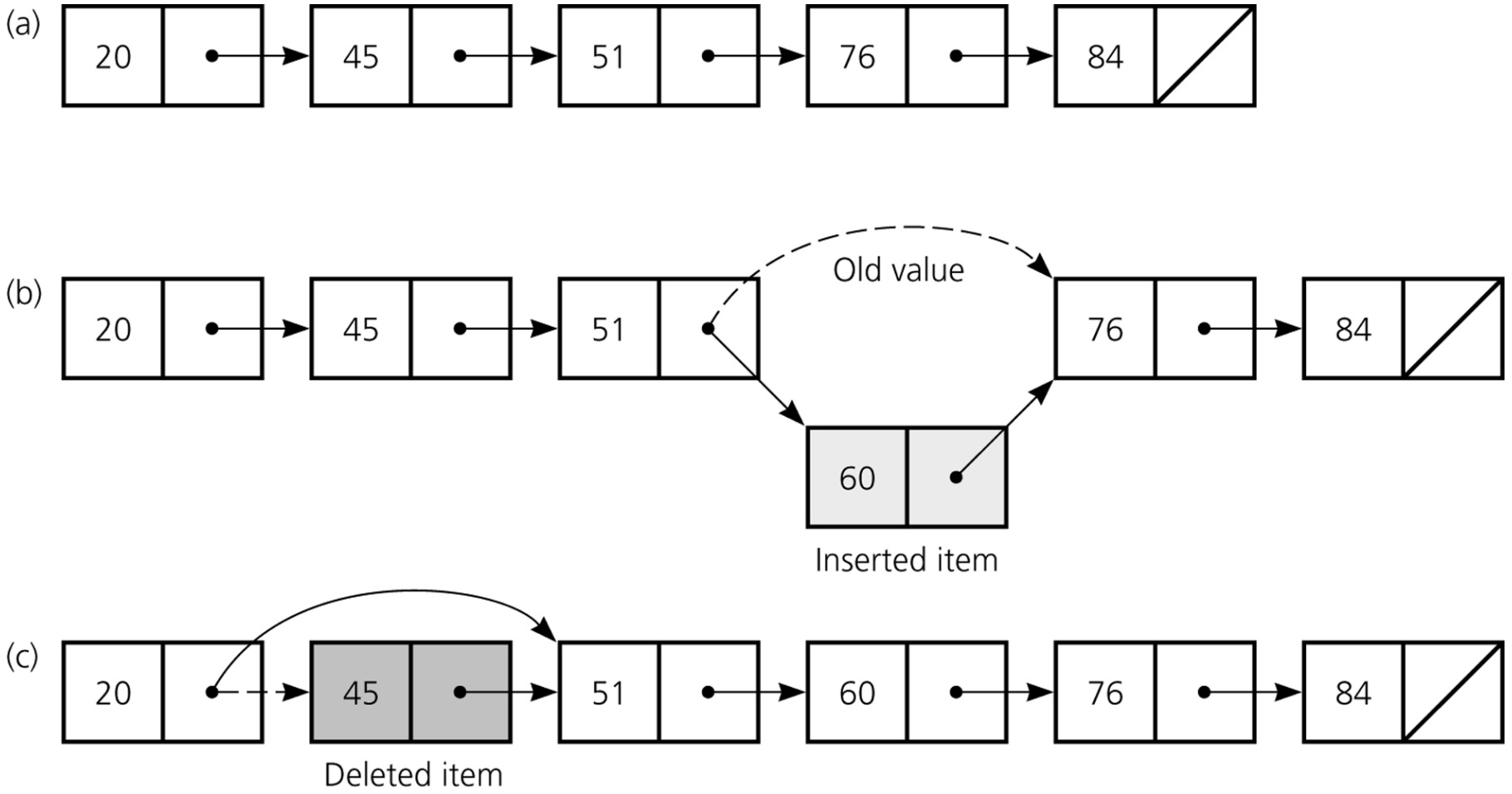


Linked List

- A *linked list* is a series of connected *nodes*
- Each node contains at least
  1. A piece of data (any type)
  2. Pointer to the next node in the list
- *Head*: pointer to the first node
- The last node points to NULL to mark the end of the list

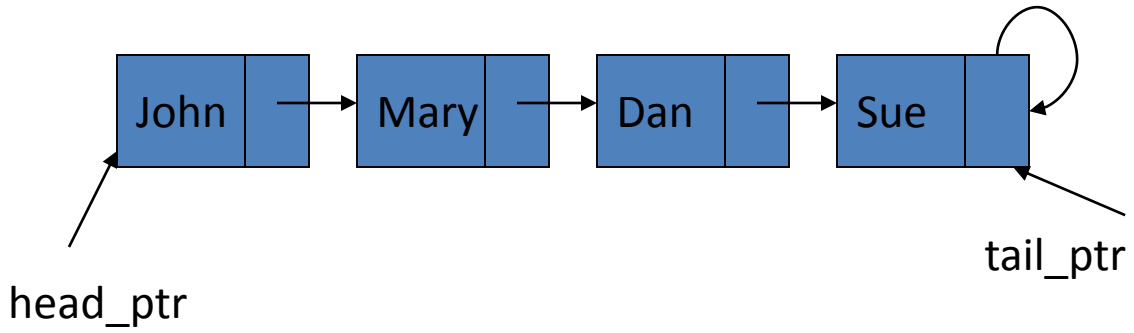


A Linked List of Integers

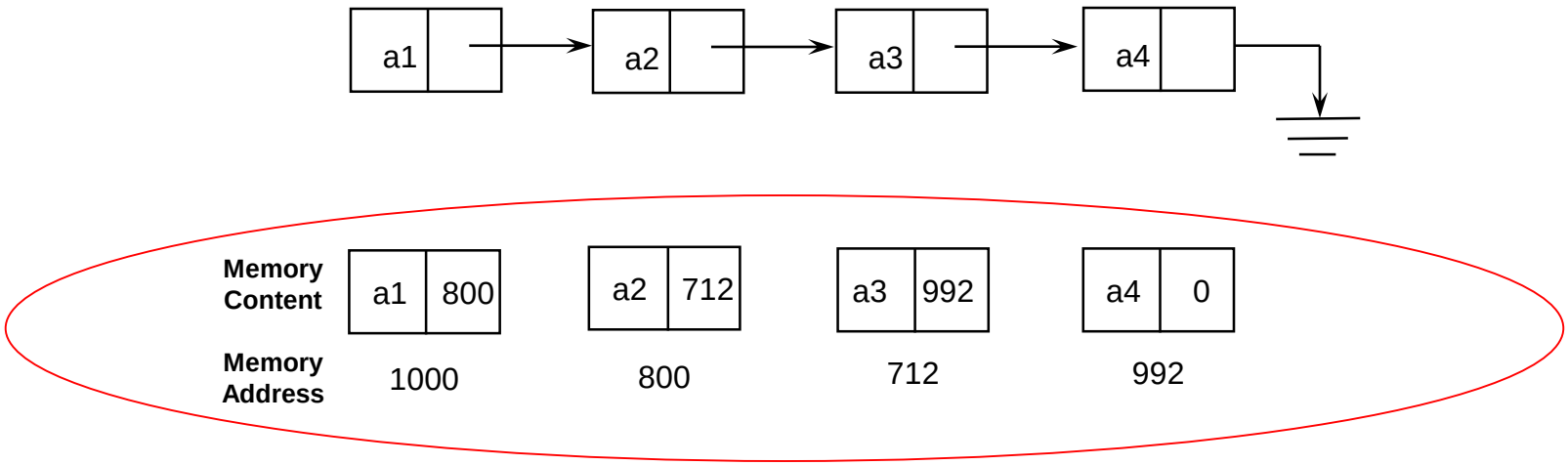


Example of Linked Lists

- A waiting line of customers: John, Mary, Dan, Sue (from the head to the tail of the line)
- A linked list of **strings** can represent this line:



What does the memory look like?



- Linked list nodes are normally not stored contiguously in memory
- Logically, however, the nodes of a linked list appear to be contiguous

Array Vs. Linked List

- Lists of data can be stored in arrays, but linked lists provide several advantages

- A linked list is appropriate when
  - the number of data elements to be represented in the data structure is unpredictable

- Linked lists are dynamic
  - length of a list can increase or decrease as necessary
  - The size of an array, cannot be altered once memory is allocated

Array Vs. Linked List contd.

- Linked lists become full only when the system has insufficient memory to satisfy dynamic storage allocation requests
  - Arrays can become full
- An array can be declared to contain more elements than the number of data items expected
  - This can waste memory
  - Linked lists can provide better memory utilization in these situations
- Insertion and deletion in a sorted array can be time consuming
  - In linked list, it is easy

Pointer-Based Linked Lists

- A node in a linked list is usually a struct

```
struct node{
    int item;
    node *next;
};
```

- The next step is to declare a pointer to serve as the list head

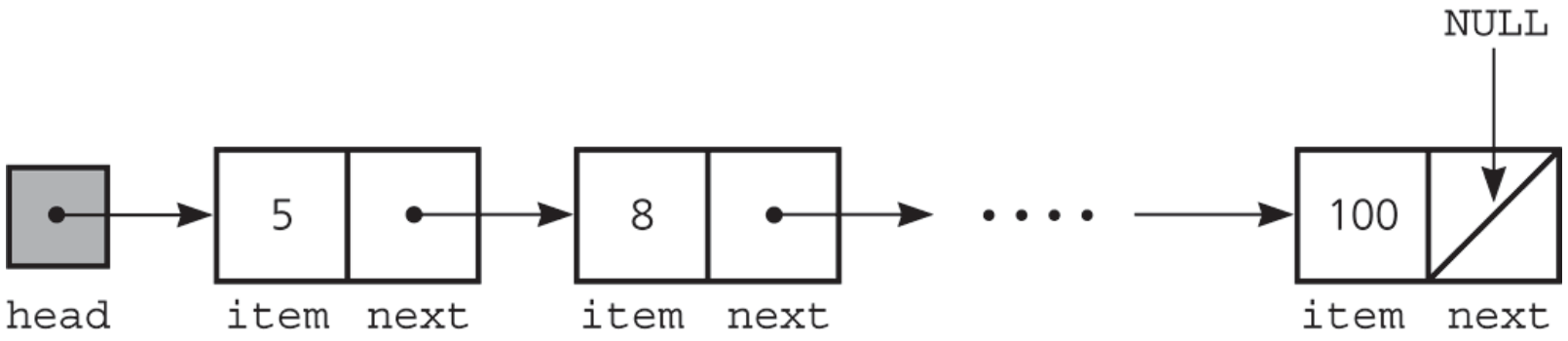
```
struct node *head;
head=NULL;
```

Pointer-Based Linked Lists contd.

- The head pointer points to the first node in a linked list
- If head is *NULL*, the linked list is empty

- Reference a node member with the -> operator

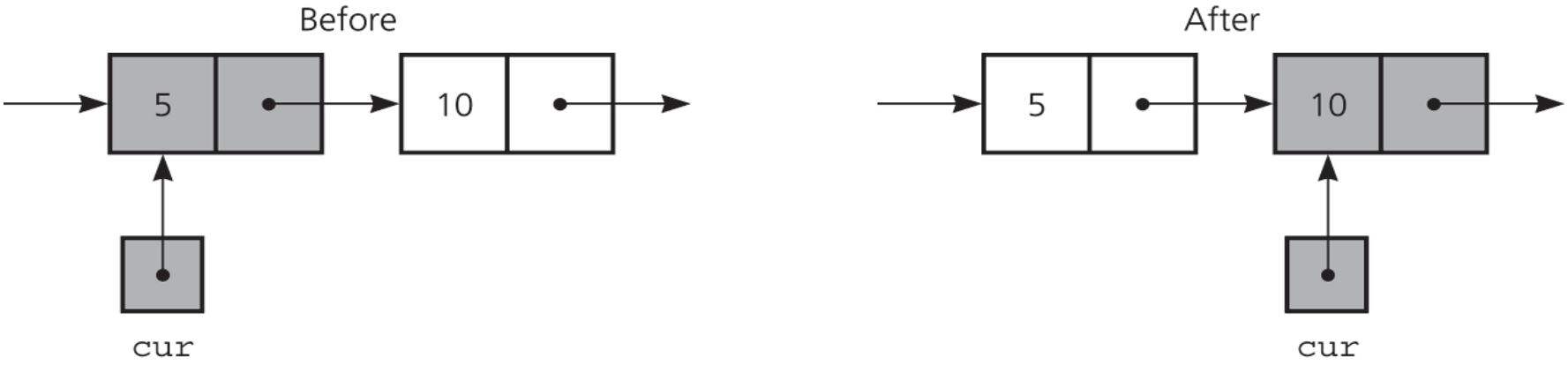
```
p->item;
```



Displaying the Contents of a Linked List

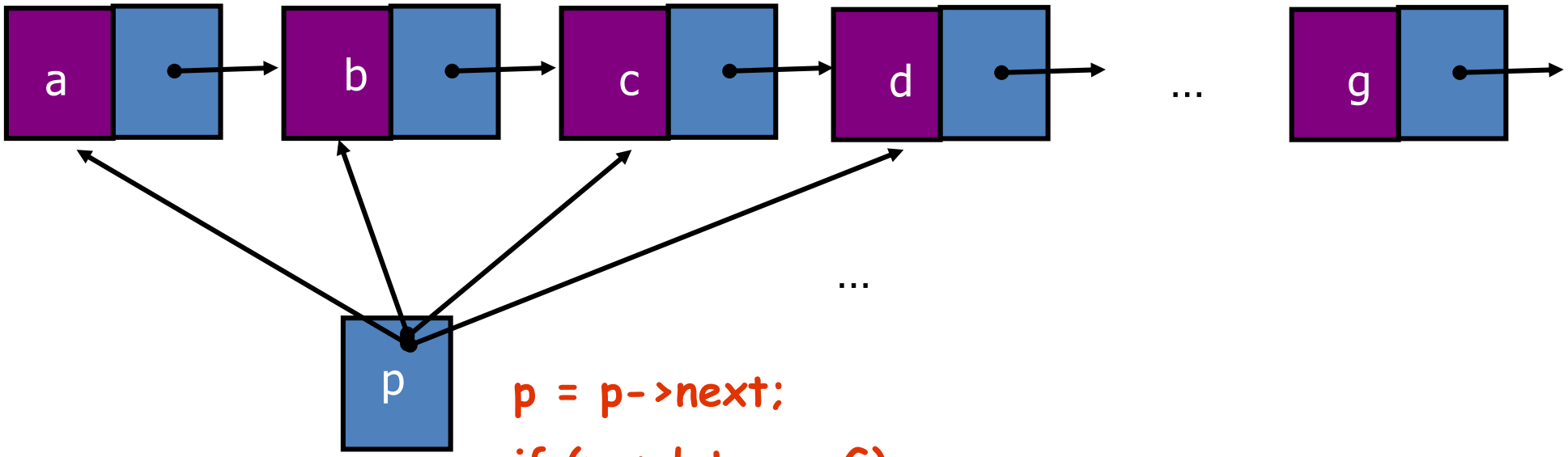
- traverses the list, displaying the value member of each node

Assign head to curr pointer
While curr pointer is not NULL
Display the value member of the node pointed to by curr pointer
Assign curr pointer to its own next member
End While.



The effect of the assignment cur = cur->next

Finding a Node in a Linked List




```
p = p->next;
if (p->data == G)
...
```

Inserting a Node into a a Linked List

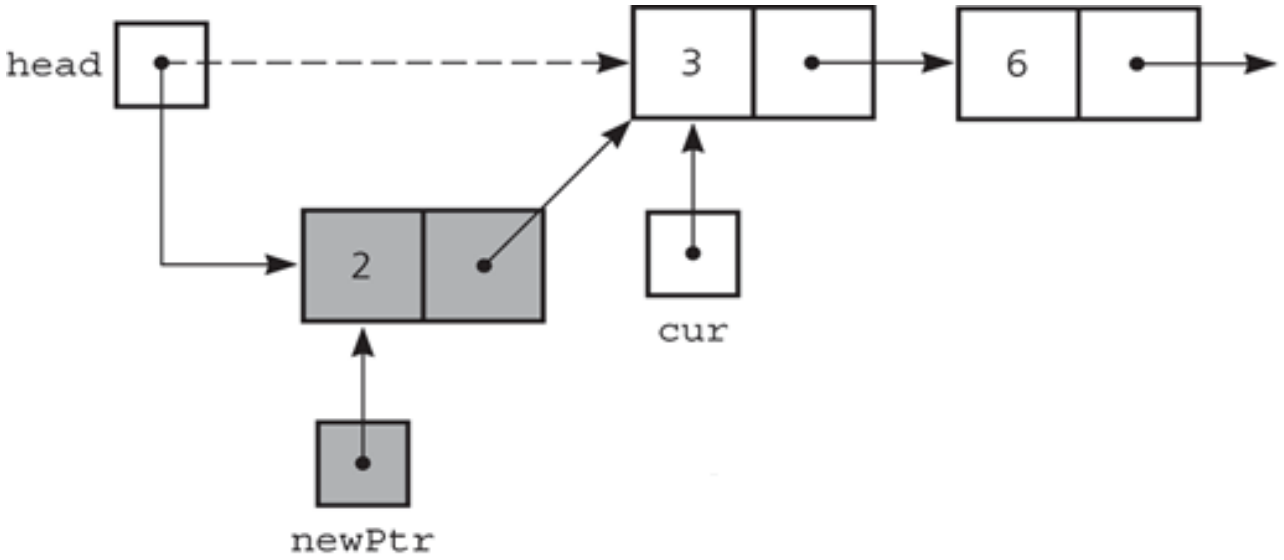
- Four cases can arise:

1. Inserting into an empty list
2. Insertion at the beginning of the list
3. Insertion at the end of the list
4. Insertion in the middle of the list

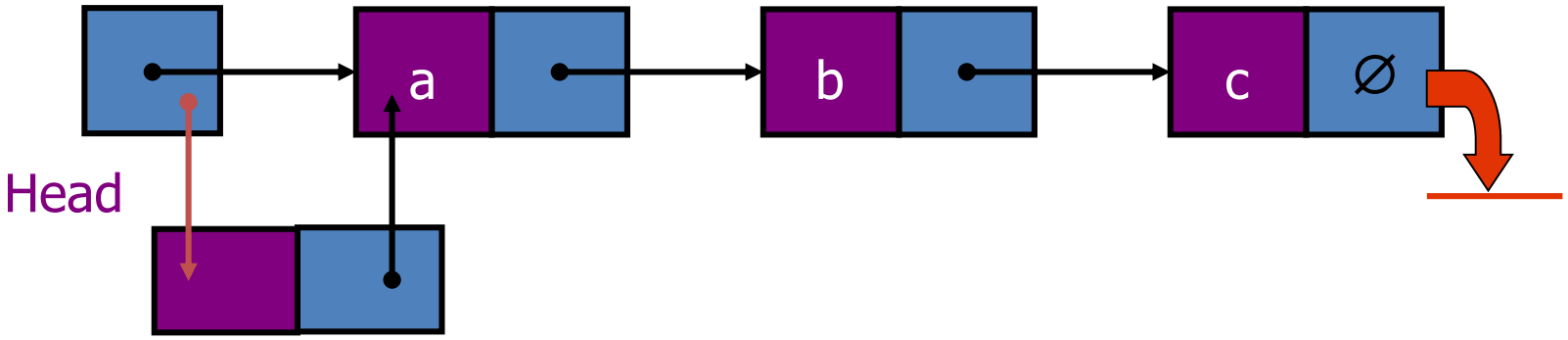
Inserting at the Beginning of a Linked List

- To insert a node at the beginning of a linked list

```
newPtr->next = head;
head = newPtr;
```



Insertion at the Beginning of a Linked List

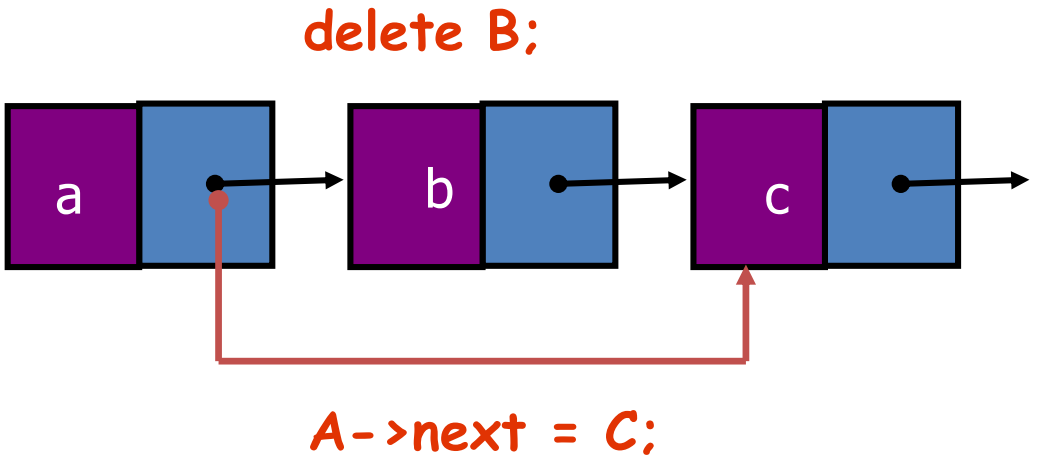


Appending a Node to the Linekd List

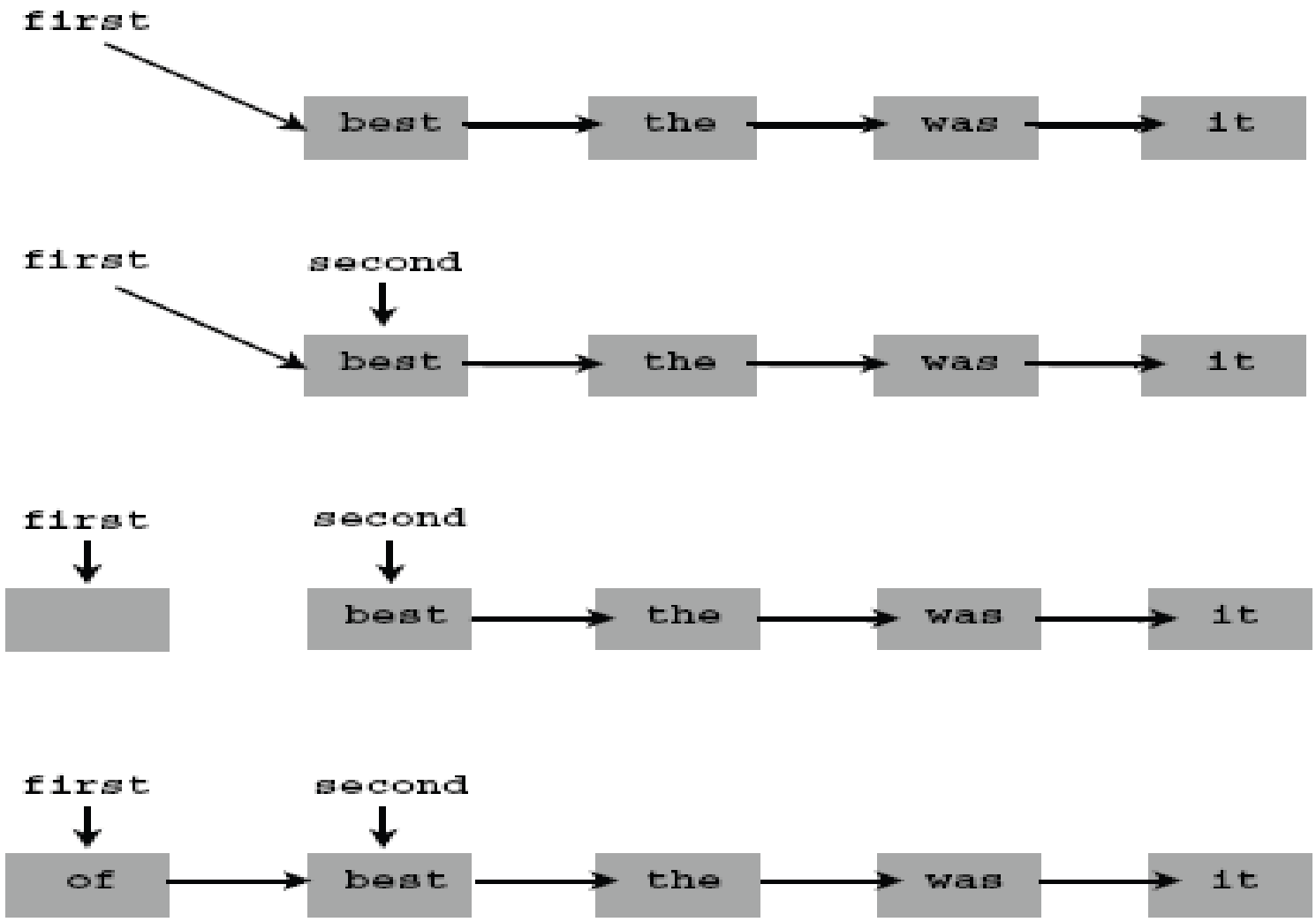
- To append a node to a linked list means to add the node to the end of the list

Create a new node
Store data in the new node
If there are no nodes in the list
    Make the new node the first node
Else
    Traverse the List to Find the last node
    Add the new node to the end of the list
End If

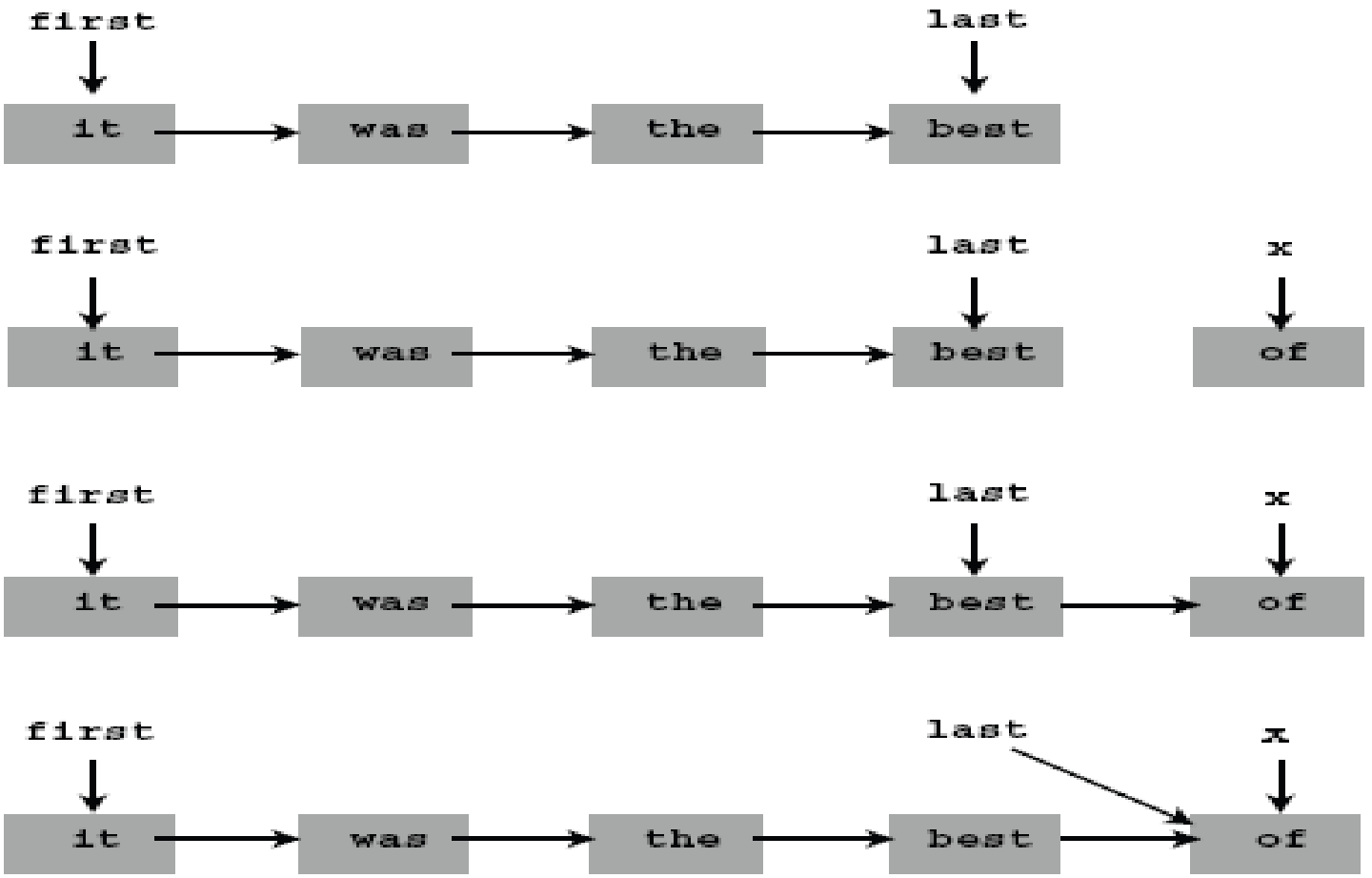
Deleting a Node from the Linked List



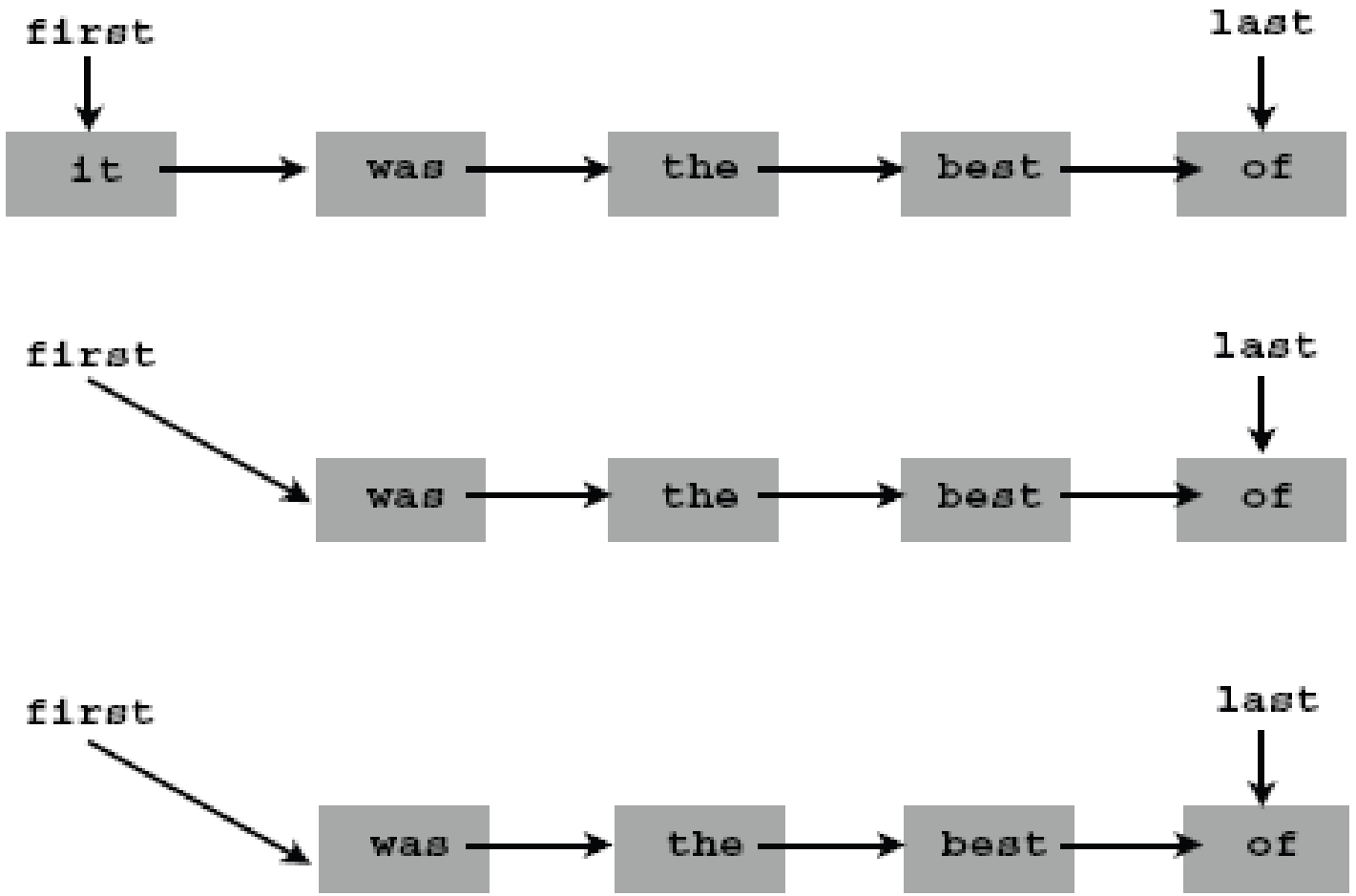
List Based Stack Implementation



List Based Queue Implementation

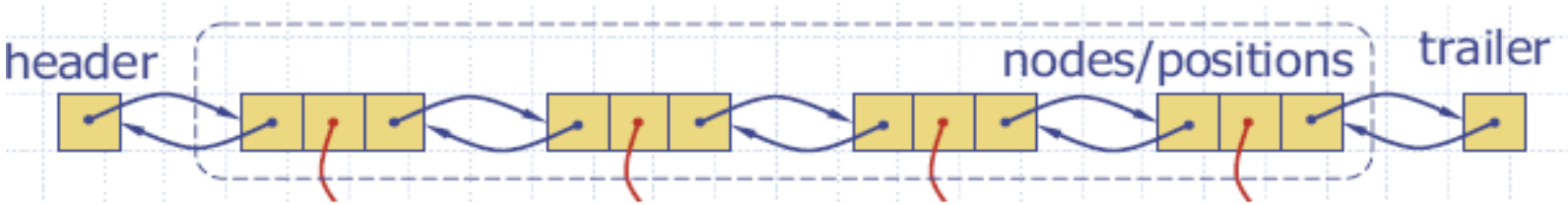


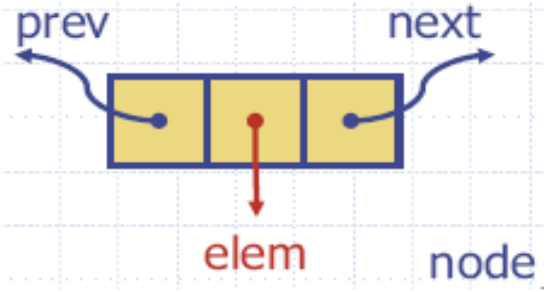
List Based Queue Implementation



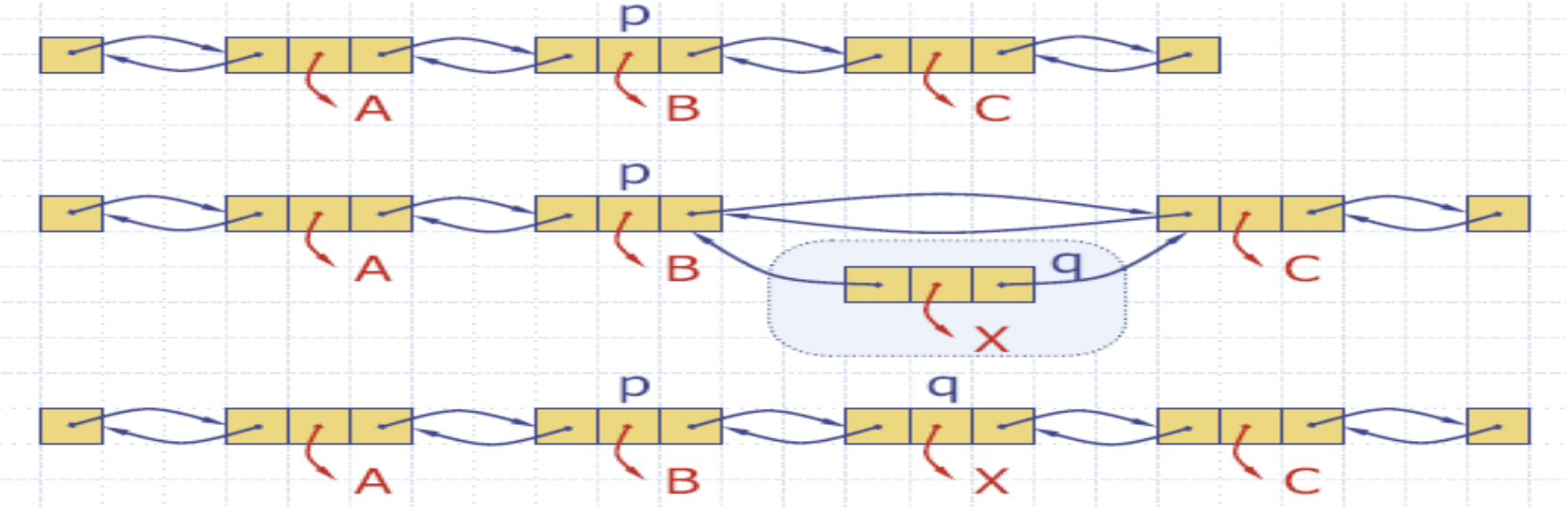
Doubly Linked List

- Doubly Linked List provides a better implementation of the Link ADT (than Singly Linked List)



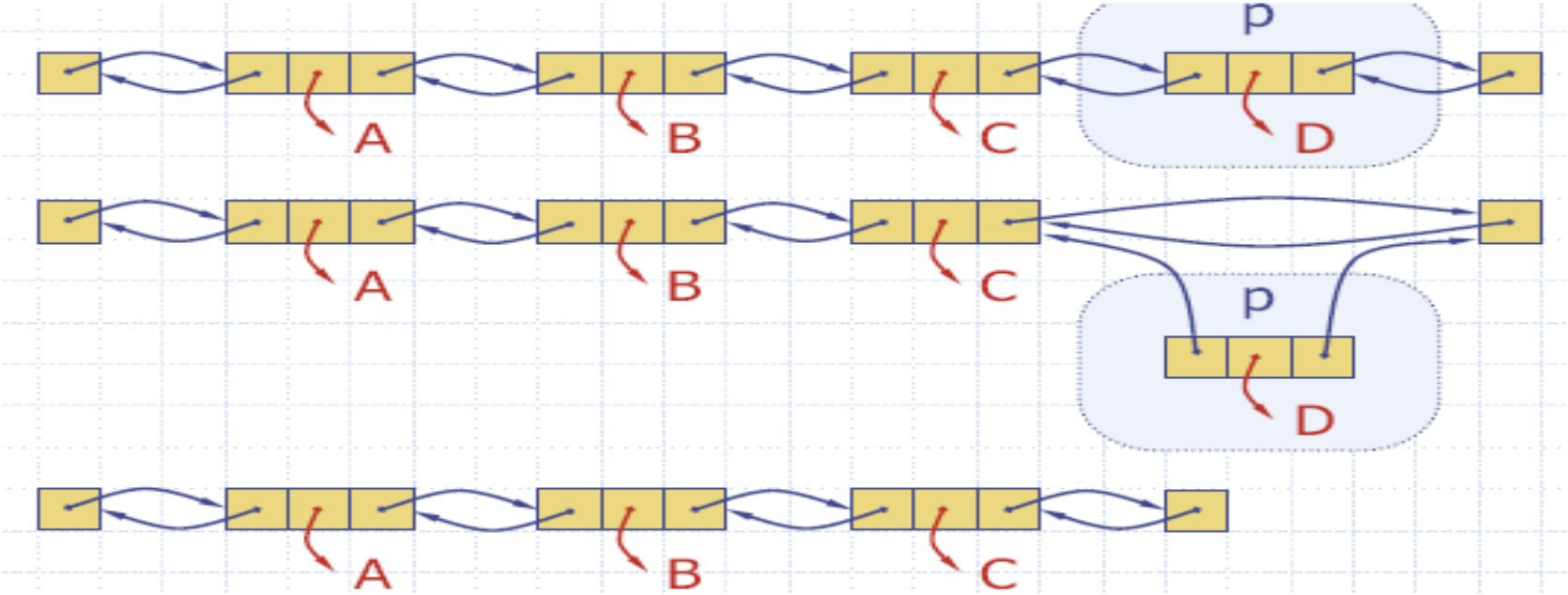


Doubly Linked List - insertAfter




```
function insertAfter(p;e):
q ← new node
q.element ← e
q.previous ← p
q.next ← p.next
(p.next).previous ← q
p.next ← q
end function
```

Doubly Linked List - remove




```
function remove(p):
t ← p.element
(p.previous).next ← p.next
(p.next).previous ← p.previous
p.previous  ← NULL
p.next  ← NULL
return t
end function
```

Summary

- Linked lists optimise memory usage when compared with arrays for dynamically changing data
- There are various categories of linked lists such as singly linked list and doubly linked list