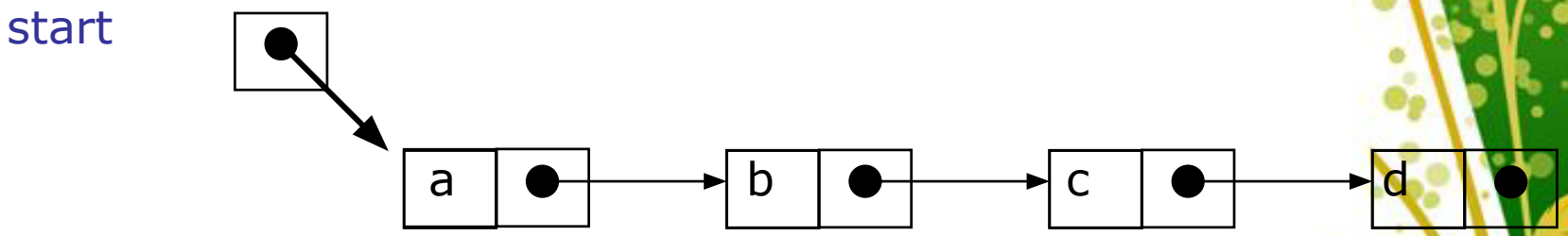
LINKED LIST

Array Vs Linked List

- Array
  - Arrays have fixed size
  - Data must be shifted during insertions and deletions
- Linked list
  - Linked list is able to grow in size as needed
  - Does not require the shifting of items during insertions and deletions

Anatomy of a linked list

- A linked list consists of:
  - A sequence of nodes



  - Each node contains a value
  - and a link (pointer or reference) to some other node
  - The last node contains a null link
  - The list may (or may not) have a header

More terminology

- Null pointer
- External pointer
- Empty List

Operations on Linked Lists

- Creation
- Traversal
- Insert a new item
  - At the head of the list, or
  - At the tail of the list, or
  - Inside the list, in some designated position

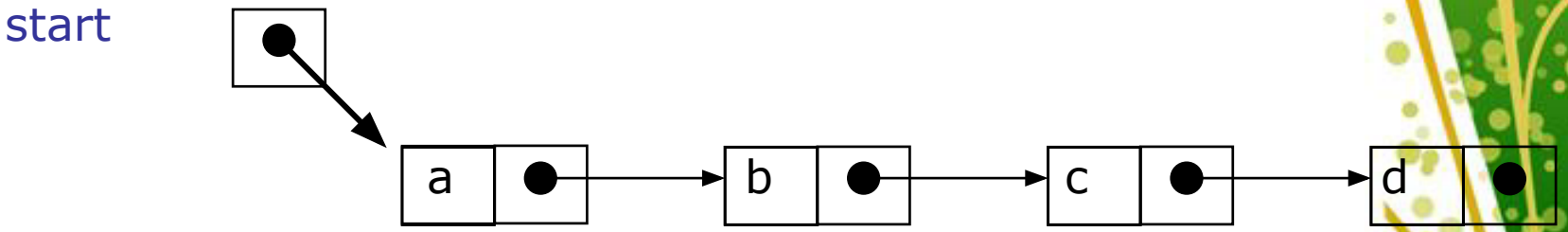
- Delete an item from the list
  - Search for and locate the item, then remove the item, and finally adjust the surrounding pointers
- Search for an item in the list
  - The item can be specified by position, or by some value
- Concatenation

Types of Linked List

- Basically, we can put LL into following four types:
  - Singly-Linked List
  - Doubly-Linked List
  - Circular Linked List
  - Circular Doubly Linked List

Singly-linked lists

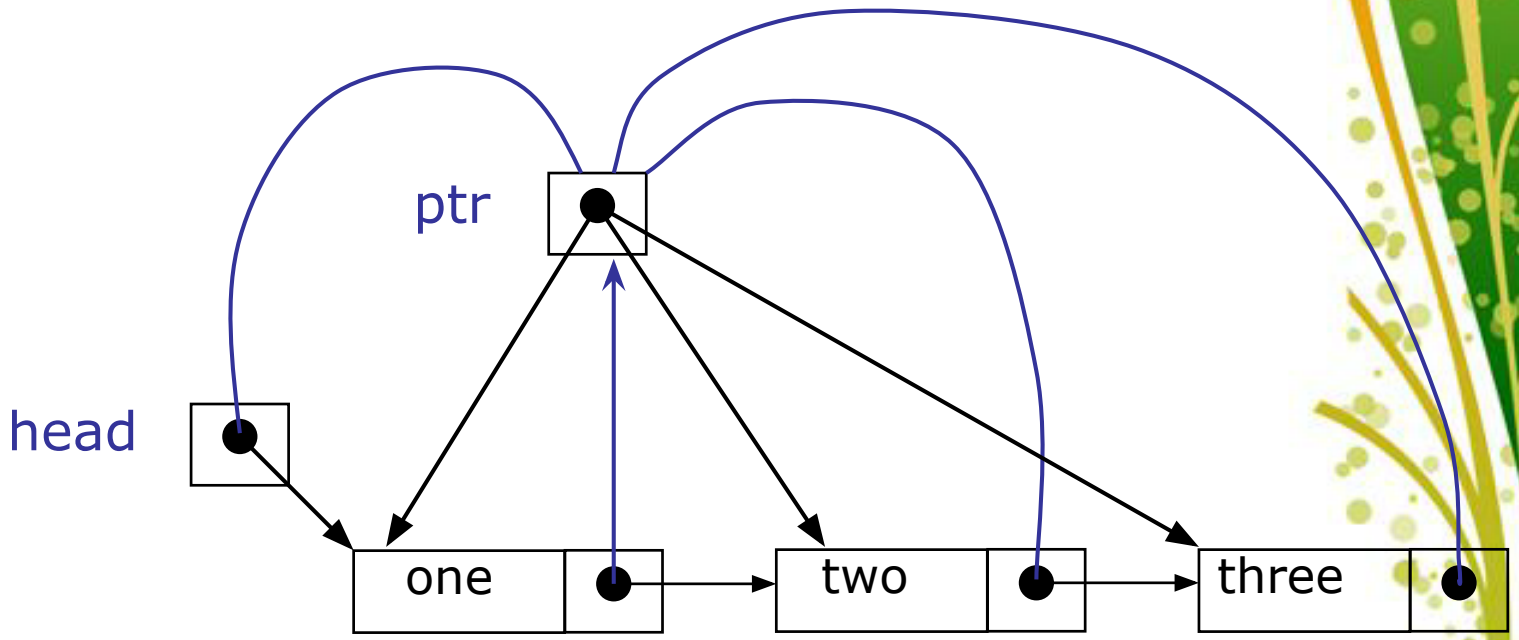
- Here is a singly-linked list (SLL):



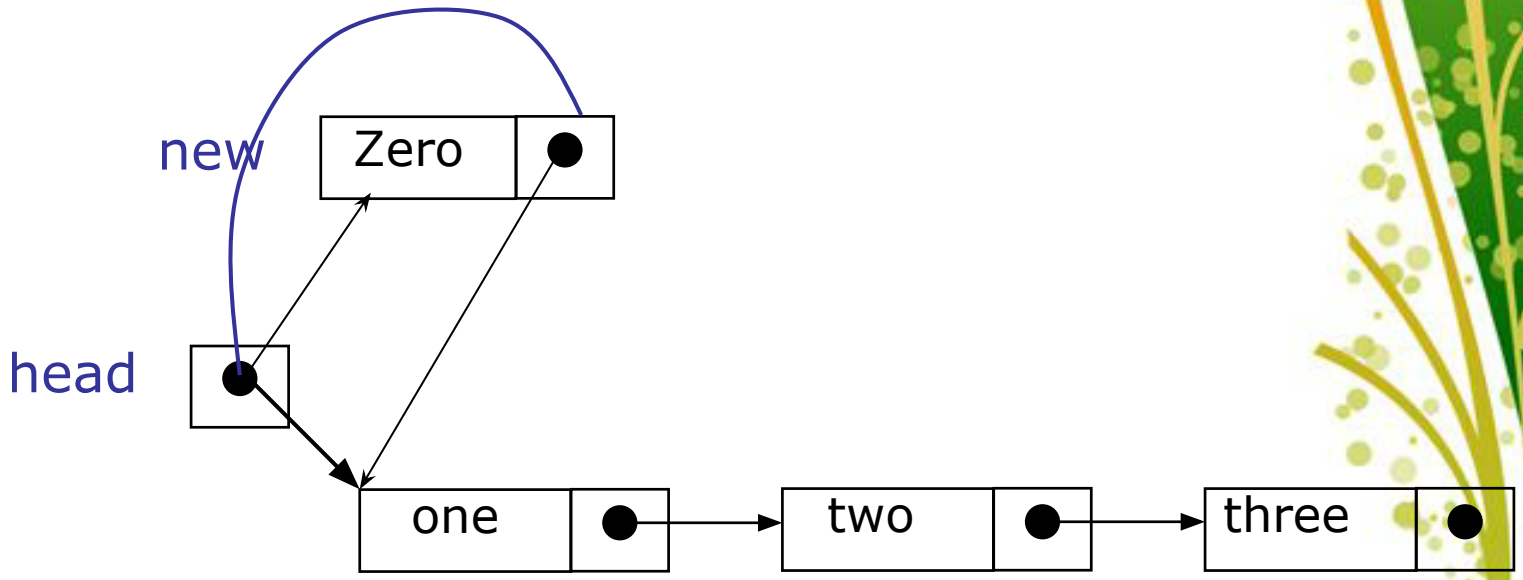
- Each node contains a value and a link to its successor (the last node has no successor)
- The header points to the first node in the list (or contains the null link if the list is empty)

Creation

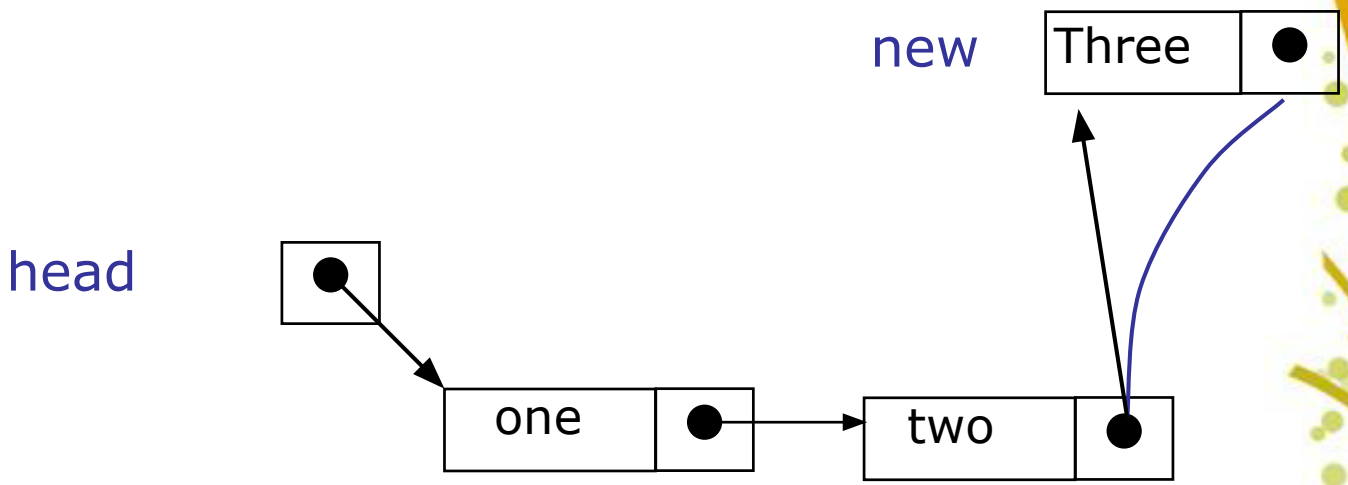
Traversal



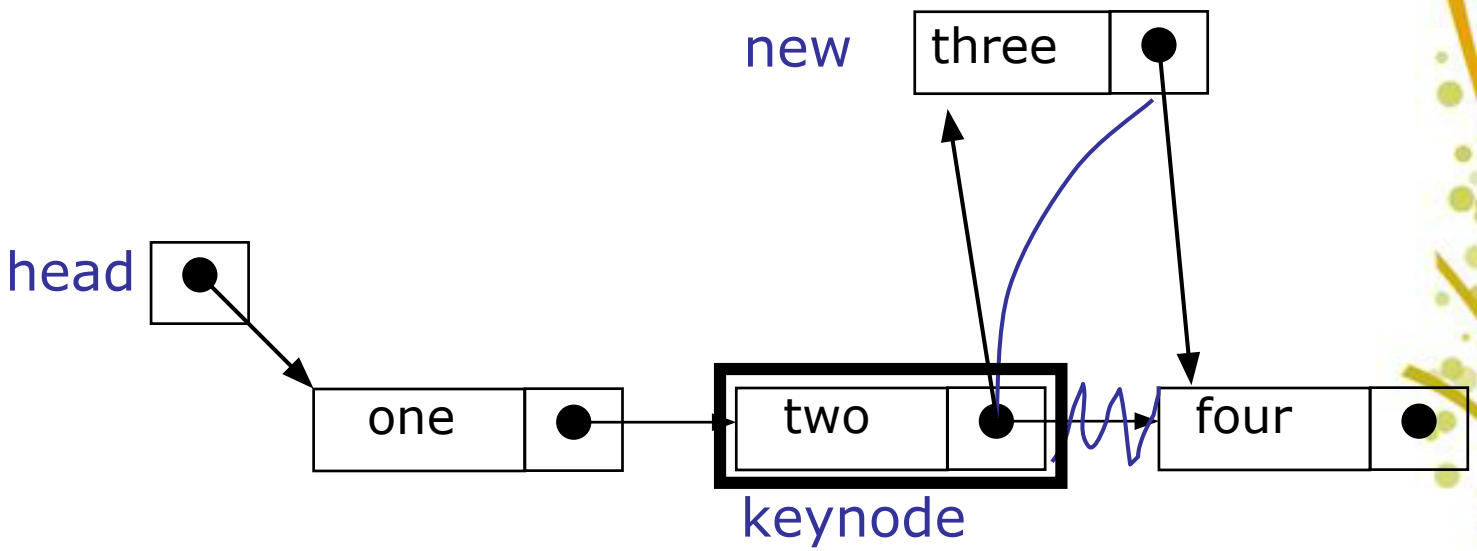
Insertion at the beginning



Insertion at the end



Insertion at a specified position



Deletion from Beginning

- **Algorithm DELETE_SL_FRONT (HEAD)**
- **Input: Head pointer**
- **Output: A list eliminating the node from the front**

Steps:

1. Ptr = head
2. If (ptr = NULL) then
    1. Print “The list is empty”
    2. Exit
3. Else
    1. p = ptr. Link
    2. Head = p
    3. Free (ptr)
4. End if
5. Stop

```
void delete_from_begin()
{
 if(head==NULL)
 {
  printf("LINKED LIST IS EMPTY !!!");
  return;
 }
 else
 {
  temp=head;
  printf("THE DELETED ELEMENT IS %d", temp->da
  head=head->link;
  free(temp);
 }
}
```

Deletion from End

Algorithm DELETE_SL_END (HEAD)

Input: Head pointer

Output: A list eliminating the node from the end.

Steps:

1. Ptr = head
2. If (ptr = NULL) then
    1. Print “The list is empty”
    2. Exit
3. Else
    1. While (ptr. link! = NULL) do
        1. Ptr1 = ptr
        2. Ptr = ptr. Link
    2. End While
    3. Ptr1. link = NULL
    4. Free (ptr)
4. End if
5. Stop

Function

```
void delete_from_end()
{
 if(head==NULL)
 {
  printf("LINKED LIST IS EMPTY !!!");
  return;
 }
 else
 {
  if(head->link==NULL)
  {
   temp = head;
   printf ("THE DELETED ELEMENT IS %d", temp->data
   head=NULL;
   free(temp);
  }
```

```
else
   {
    curr=head;
    temp=head->link;
    while(temp->link!=NULL)
    {
     curr=temp;
     temp=temp->link;
    }
    printf("THE DELETED ELEMENT IS %d", temp->d
    curr->link=NULL;
    free(temp);
   }
  }
}
```

Deletion of any particular node

Algorithm DELETE_SL_ANY (HEAD, KEY)

Input: Head pointer, Key is the data content of the node to be deleted.

Output: A list eliminating the node with the data content as Key.

Steps:

1. Ptr1 = head
2. If (ptr1 = NULL) then
    1. Print “The list is empty”
    2. Exit
3. Else
    1. Ptr = ptr1
    2. While (ptr ! = NULL) do
        1. If (ptr. data! = KEY) then
            1. Ptr1 = ptr
            2. Ptr = ptr. Link
        2. Else
            1. Ptr1. link=ptr. Link
            2. Free (ptr)
            3. Exit
        3. End If
    3. End While

4. If (ptr = NULL) then

    1. Print “Node with Key does not exist”

5. End If

4. End if

5. Stop

Function

```
void delete_a_value()
{
 int val;
 if(head==NULL)
 {
  printf("THE LINKED LIST IS EMPTY!!");
  return;
 }
 else
 {
  printf("Enter the value to be deleted:");
  scanf("%d",&val);
  temp=head;
  curr=temp;
```

```
while(temp!=NULL)
   {
    if(temp->data == val)
    {
    curr->link=temp->link;
    free (temp);
    return;
    }
    curr=temp;
    temp=temp->link;
   }
  }
}
```