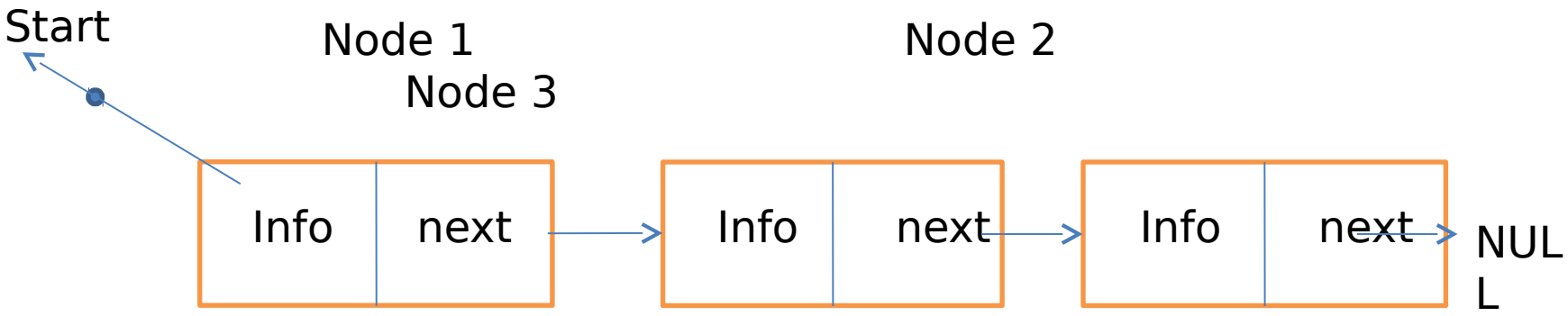
E- Lecture
Data Structures and Algorithms

By
H R Choudhary (Asstt. Professor)
Department of CSE
Engineering College Ajmer

Introduction to linked list

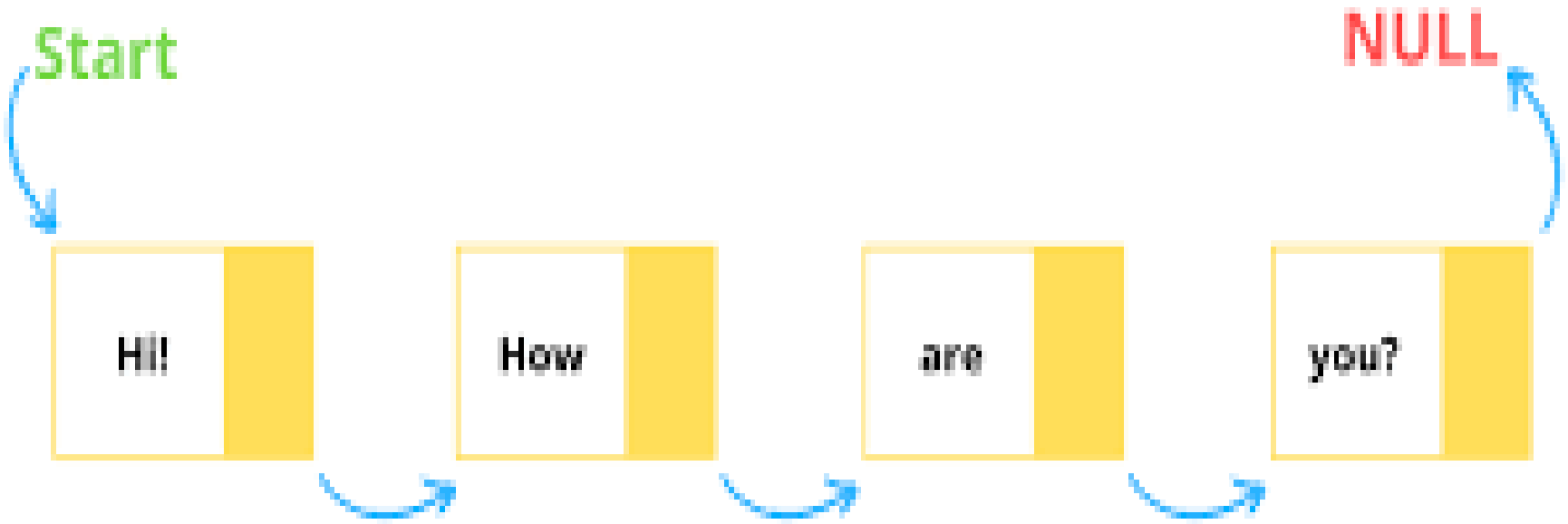
Definition of Singly linked list

- It is a special list of structures(node) which contain two part. **INFO** part stores the information and a **pointer** which points to the next element. It is dynamic in nature. Items may be added to it or deleted from it.
- **Example**



if Start= NULL
Singly linked list
List is empty

Linked list



Introduction to linked list cont..

- A list item has a pointer to the next element, or to Null if the current element is the tail (end of the list).

- This pointer points to a structure of the same type as itself.

- This structure that contains elements and pointers to the next structure is called a Node

Example of linked list:

- Hash tables use linked lists for collision resolution
- Any "File Requester" dialog uses a linked list
- Binary Trees
- Stacks and Queues can be implemented with a doubly linked list

Introduction to linked list cont..

Advantage and disadvantage of linked list:

Advantage

- Dynamic in nature, grow and shrink during execution time

- Efficient memory utilization that is, memory allocation and de-allocation as per requirement

- Insertion and deletions are easier and efficient that is, possible from any specified location

Disadvantage

- More memory space in needed to store large number of node than array
- Access to arbitrary node is time consuming and tedious

Introduction to linked list cont..

Operation on linked list:

Primary operation on linked list

- **Create:** One by one new node is created to create linked list
- **Insertion:** Insert a new node at specified location in the list
- **Deletion:** A specified node from the list
- **Traversing:** Visiting the entire list node and finding specified node in the list
- **Concatenation:** Joining one list at the end of the other list

Introduction to linked list cont..

Types of linked list

Following are the categories of linked list

- Singly- linked list
- Doubly- linked list
- Circular linked list
- Circular doubly linked list

Singly linked list

Representation of singly linked list

Singly linked list cab be represented in memory with following declaration

This declaration define new data type

```
struct node
{ int i;
struct node *next;
}
typedef struct node NODE;
NODE *start;
```

Singly linked list cont..

Creating a node of linked list

- Using Pointers , structures and dynamic memory allocation **malloc** function
- C provides following functions for dynamic memory allocation and de-allocation

1. malloc():

- This function allocates memory block in bytes and returns void pointer to first block
- So type casting is required to convert void pointer in the required type
- E.g. int *ptr;

```
ptr= (int *) malloc(10 *sizeof(int));
```

Singly linked list cont..

- **Memory allocation for structure variable**

```
 struct node
{ int item;
struct node  * next;
};
struct node  * ptr_var;
ptr_var = (struct node *) malloc(sizeof(struct node)
```

- If we write syntax

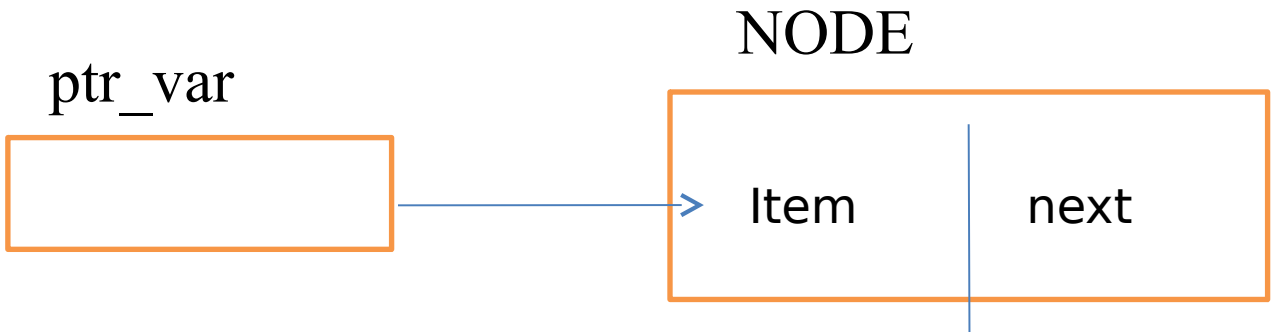
typedef struct node NODE;

Then

```
 ptr_var =(NODE *) malloc(sizeof(NODE));
```

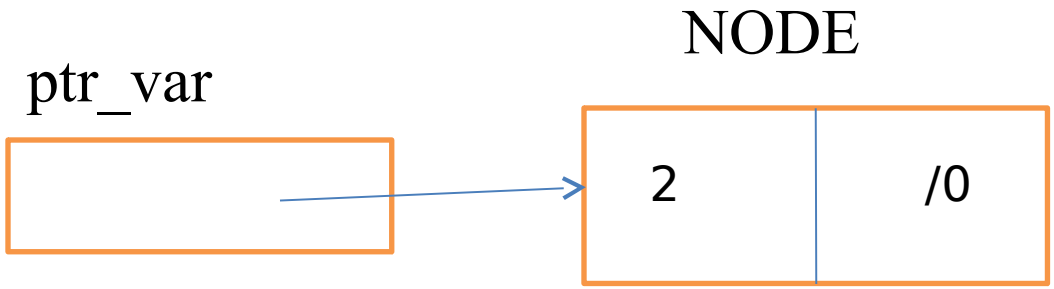
Singly Linked list cont..

- **Memory allocation for structure variable**



Assigning values to the field of NODE

- Ptr_var-->item=2;
- Ptr_var-- >next='\0';



Singly Linked list operations

Inserting the node in the linked list

1. Inserting at the beginning of the list
2. Inserting at the end of the list
3. Insertion at the specified position within the list

Algorithm Notations

- **node(p):** a node pointed to by the pointer **p**
- **item(p):** item of the node pointed by pointer **p**
- **next(p):** next field of the **node(p)**

Singly Linked list operations cont..

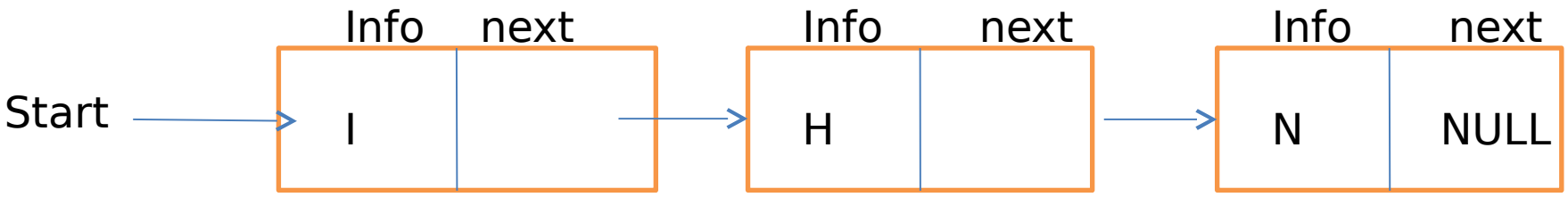
Algorithm steps for inserting a node to the List

Common steps are

- allocate space for a new node using **malloc()**
- copy the item into it,
- make the pointers adjustments

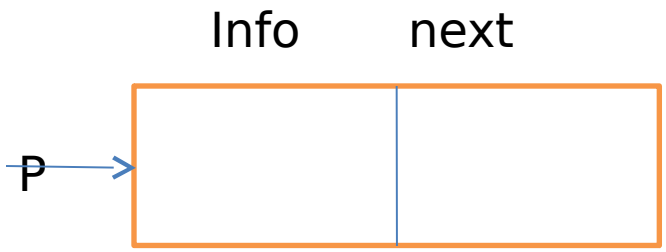
Inserting element at the beginning of the list

Assume the following list



Singly Linked list operations cont..

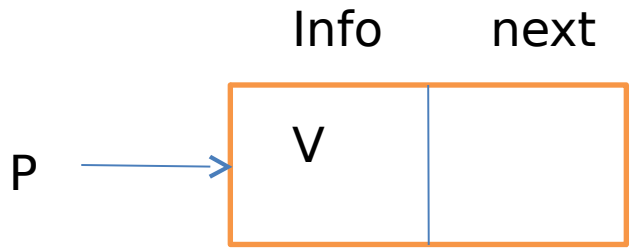
- allocate space for a new node using **malloc()**

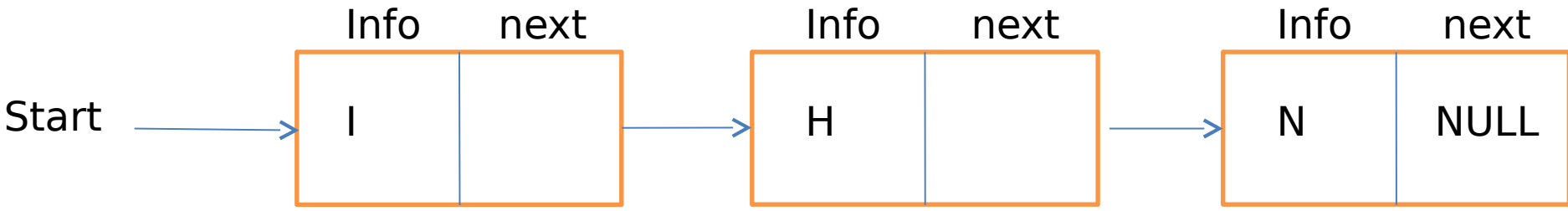


- copy the item into it,

p- ->info = V;

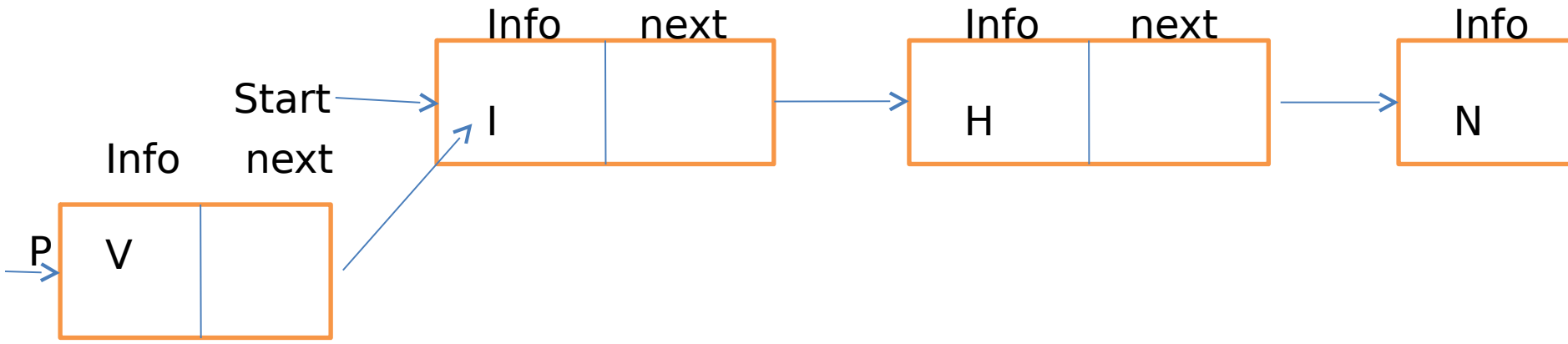
- make the pointers adjustments



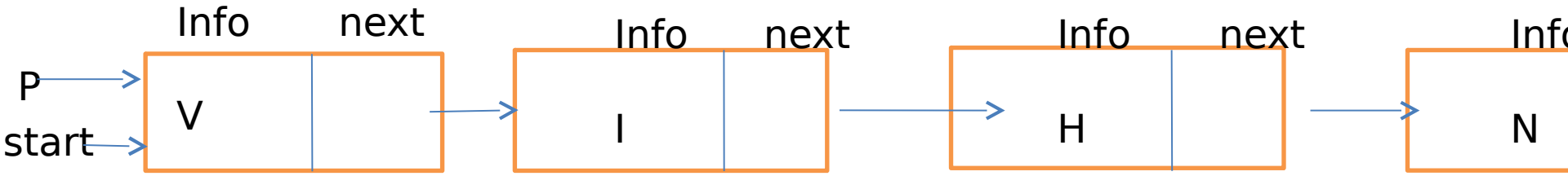


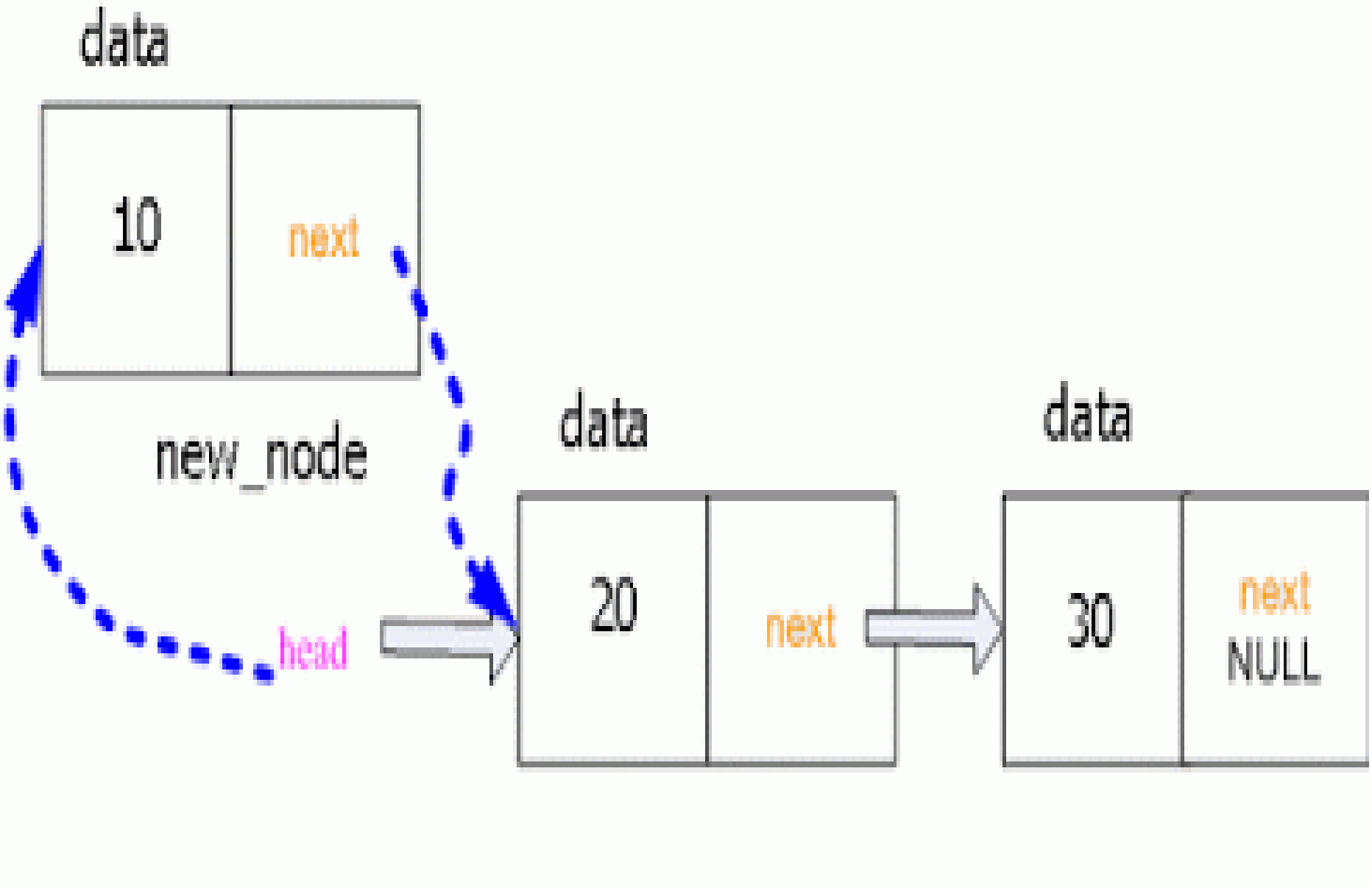
Singly Linked list operations cont..

- next(p) = start;



- start = p;

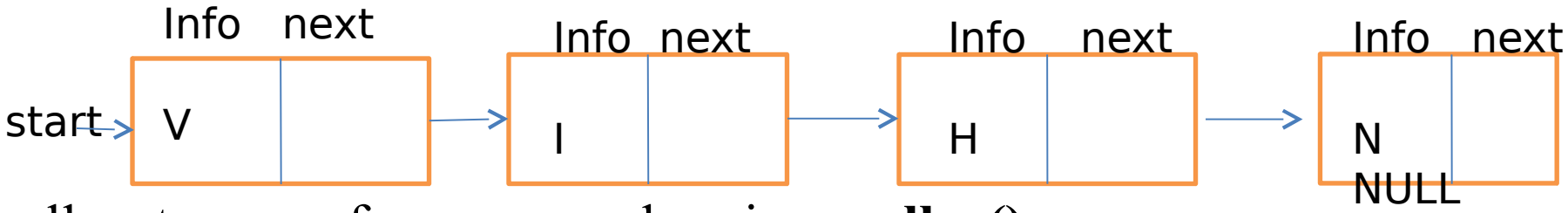



data
10
next
new_node
head
data
20
next
data
30
next
NULL

Singly Linked list operations cont..

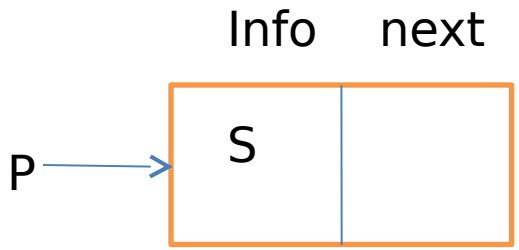
Inserting at the specified position within the list

- **Assume the following list**



- allocate space for a new node using **malloc()**
- copy the item into it,

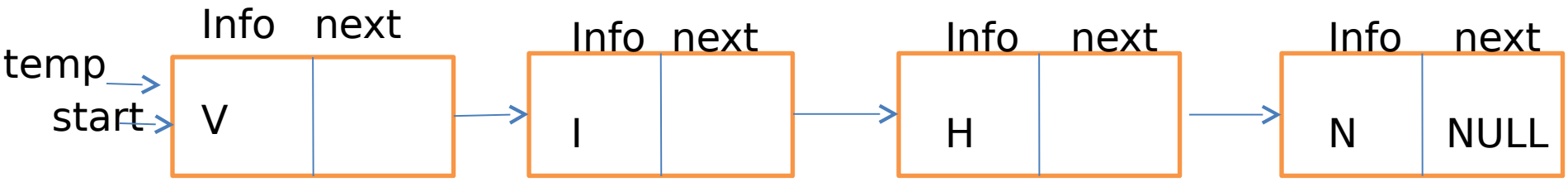
p- ->info = S;



Singly Linked list operations cont..

- **make the pointers adjustments**
- Assume we want to insert a node between Node I and Node H
- So to find that location

Set counter c=0 and pointer Node * temp
Set temp=start

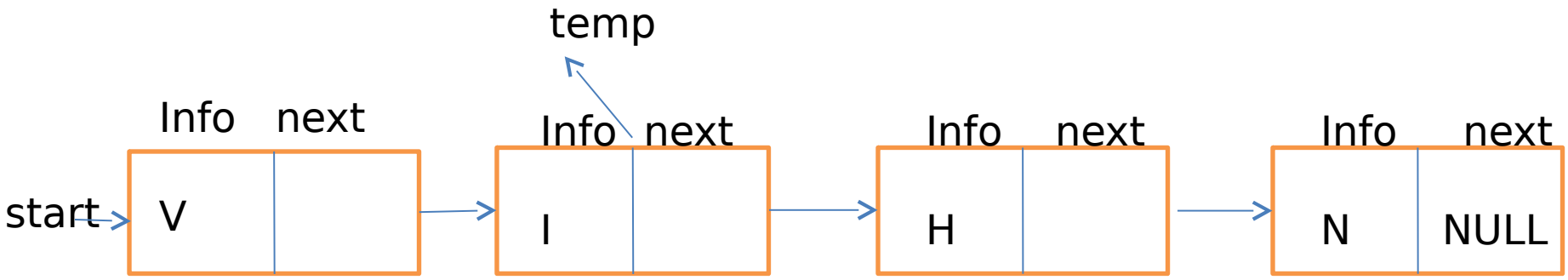


Singly Linked list operations cont..

Now to find the location, repeatedly do

Set temp =temp->next till c<location

Set c=c+1

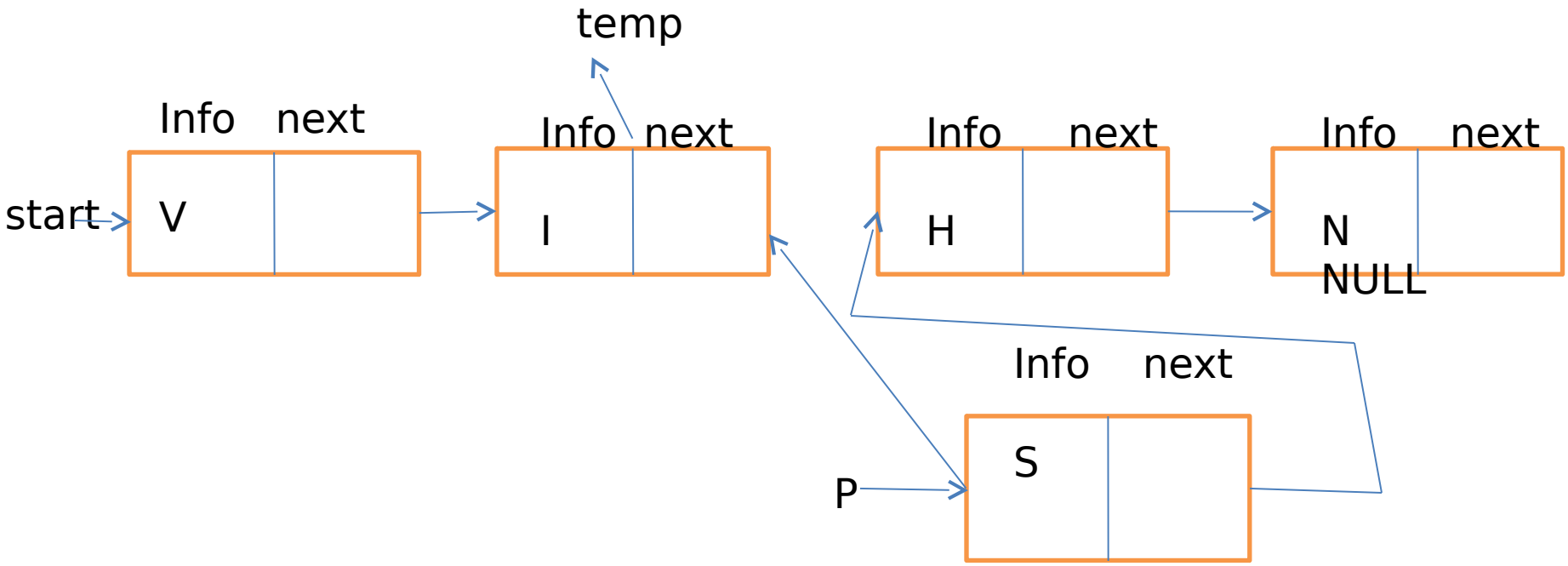


Singly Linked list operations cont..

then

Set p-> next = temp->next

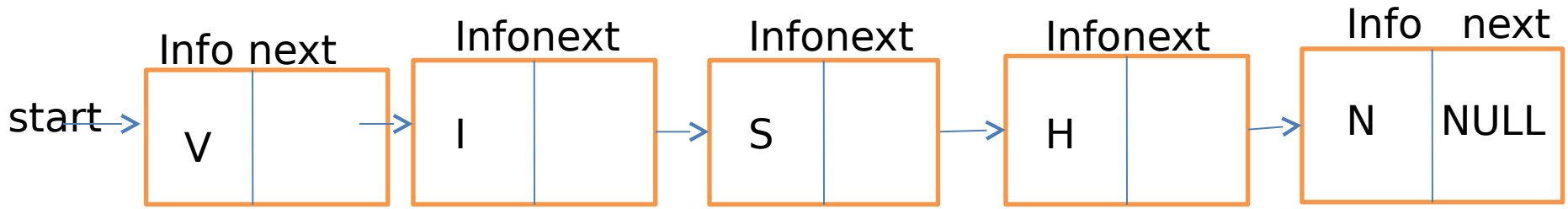
Set temp->next = p



Singly Linked list operations cont..

Inserting at the end of the list

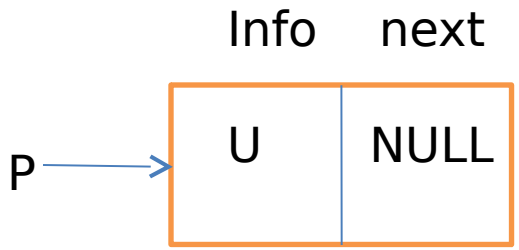
- Again assume the following list
- allocate space for a new node using **malloc()**



- copy the item into it,

p- ->info = U;

P- ->next=NULL;



Singly Linked list operations cont..

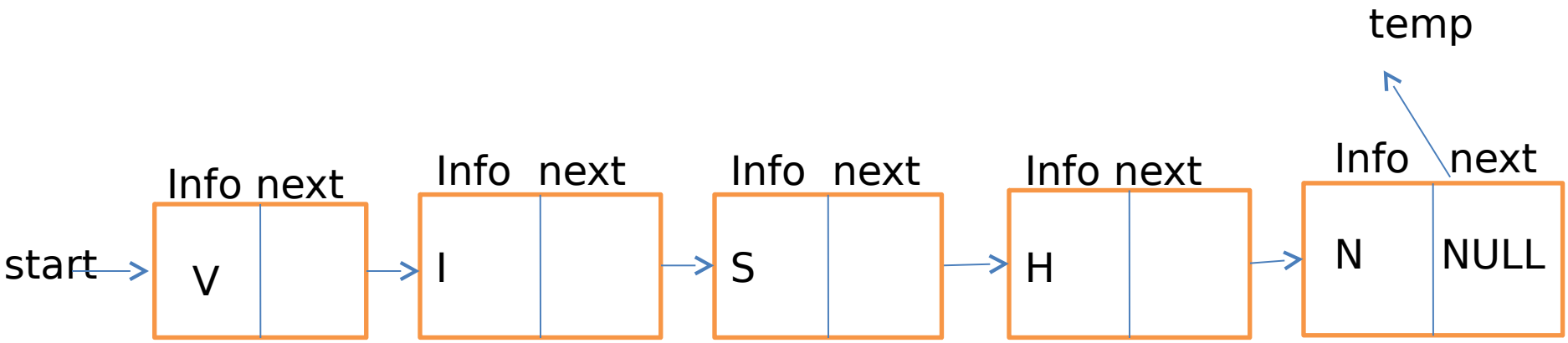
- So to find that location

Set pointer Node * temp

Set temp=start

Now to find the location, repeatedly do

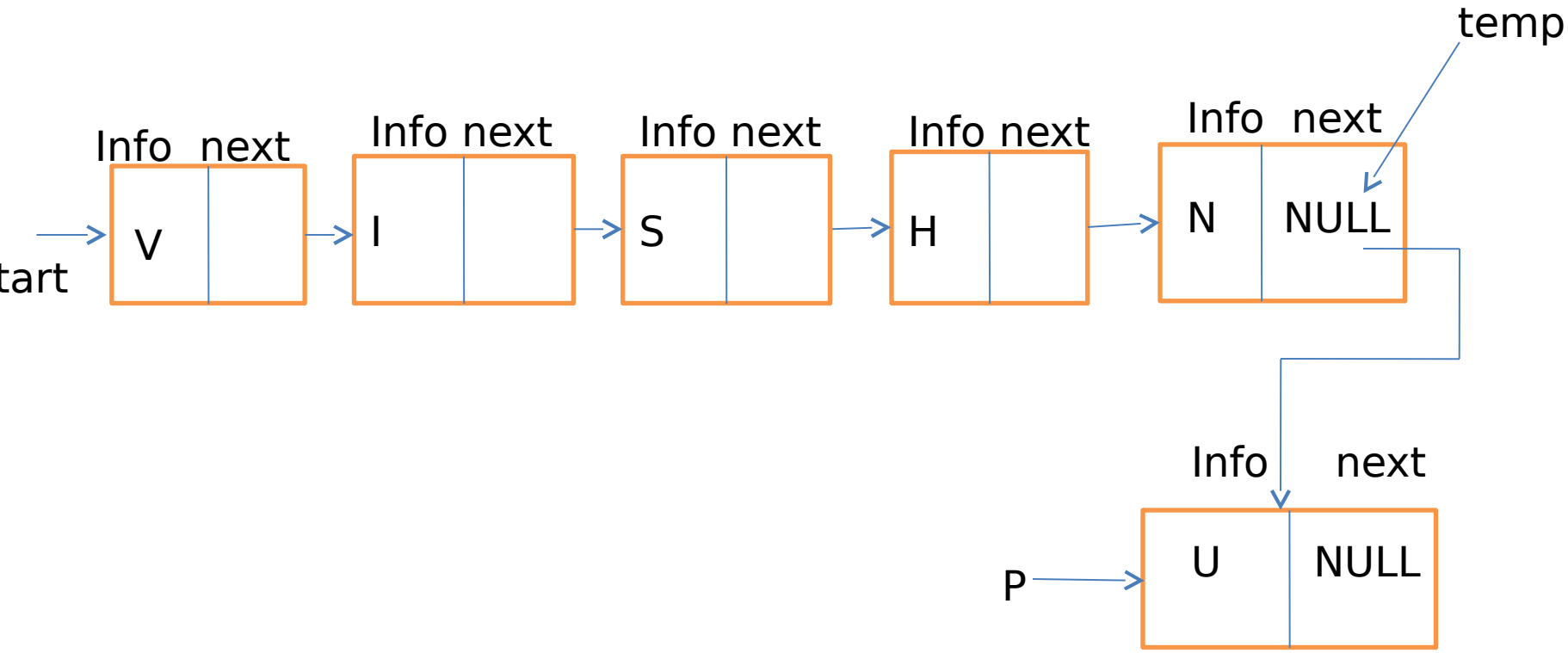
Set temp =temp->next untill temp- ->next!=NULL



Singly Linked list operations cont..

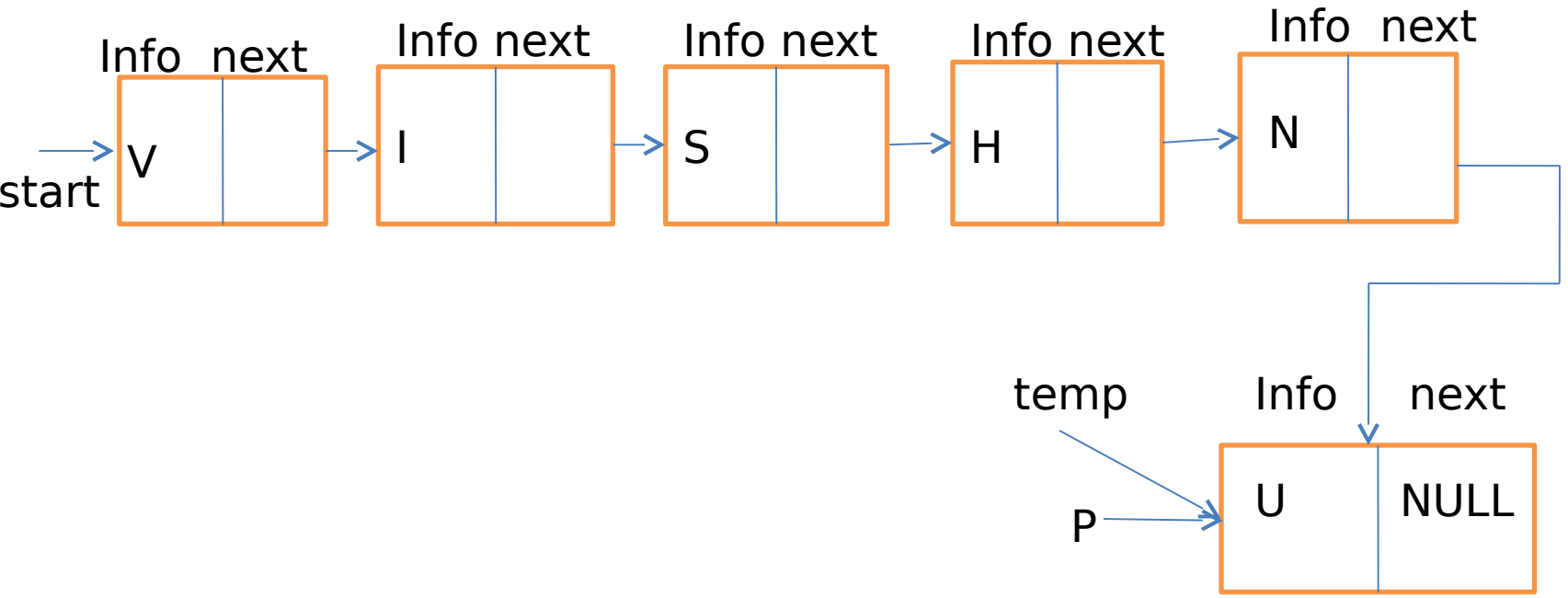
then

Set temp->next = p



Singly Linked list operations cont..

- **Set** temp = temp->next



Singly Linked list operations cont..

Linked list node deletion

1. Deleting the start node of the list
2. Deleting the end of the list
3. Deletion at the specified position within the list

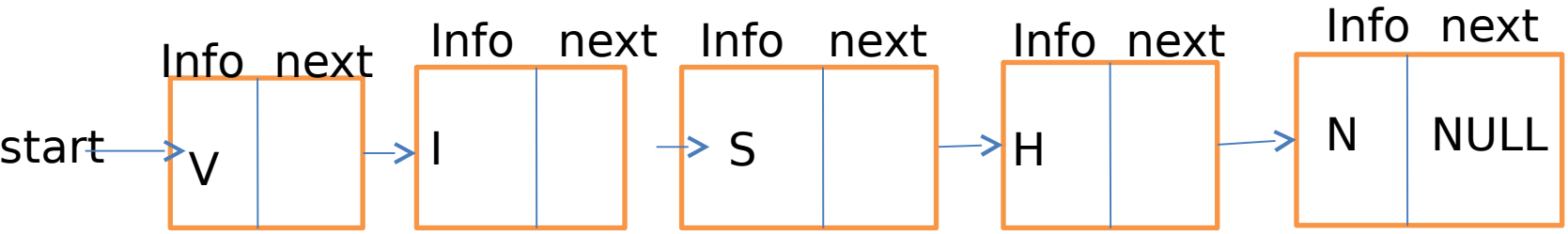
Algorithm steps for deleting a node to the List
Common steps are

- Search the node by setting and moving the pointer
- Take the information of info field of desired node in temporary variable
- Make pointers adjustments

Singly Linked list operations cont..

Deleting first element of the list

Assume the following list



If start =NULL then

Empty list

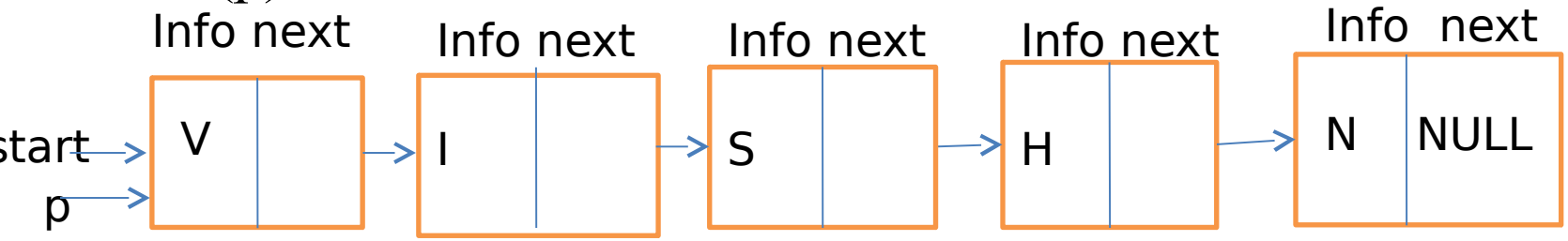
Else

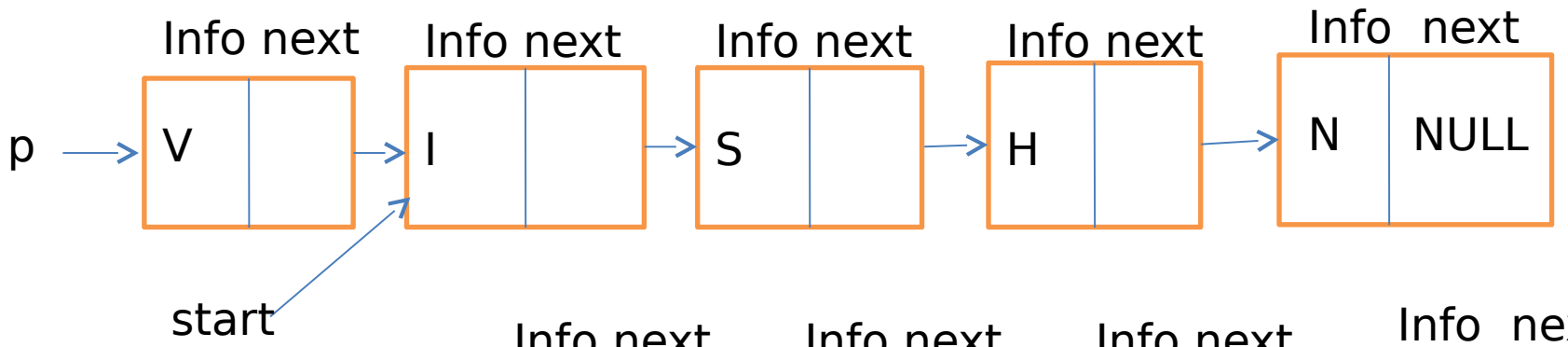
Set p = start

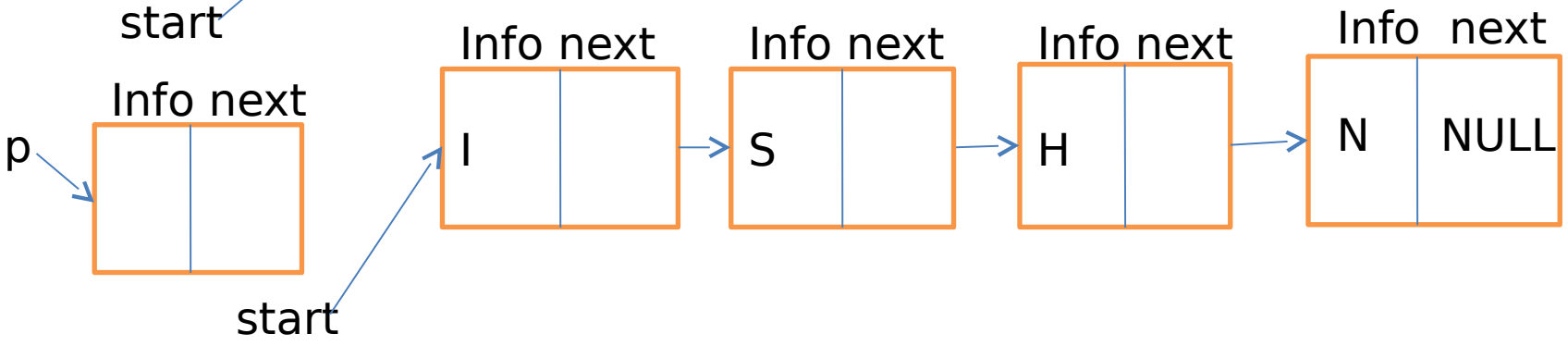
Set start = start- -> next

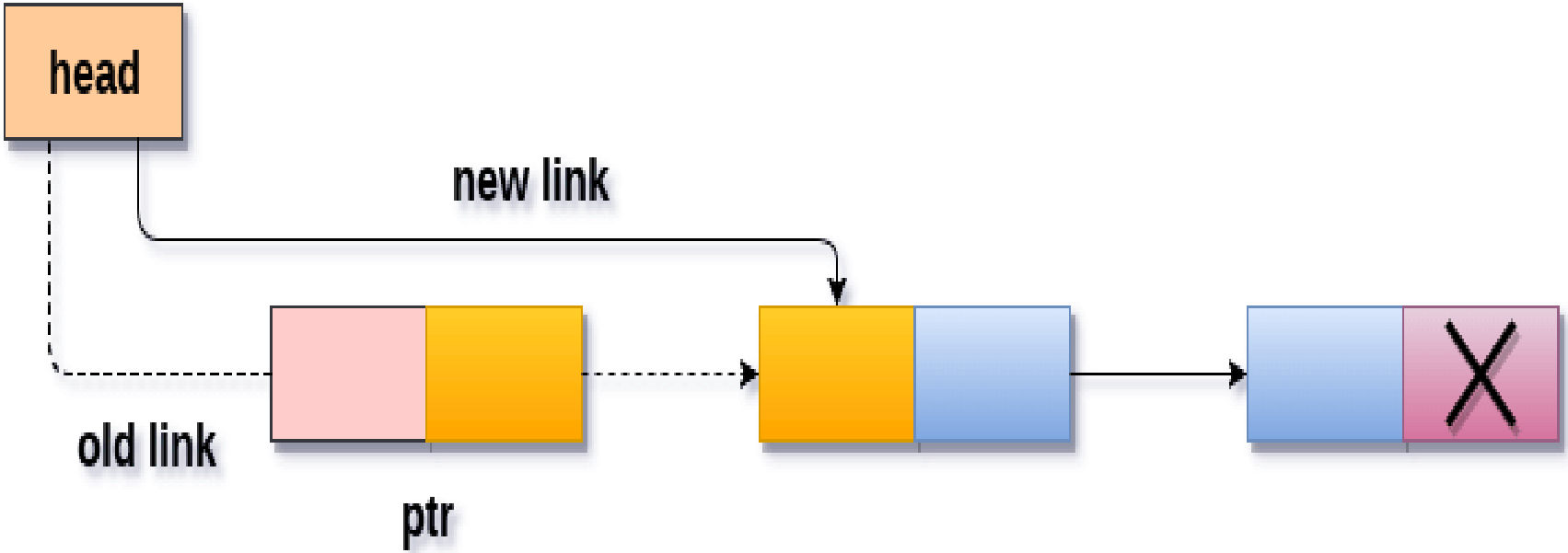
Singly Linked list operations cont..

- Char var = p- ->info
- p->Next=NULL
- free(p)









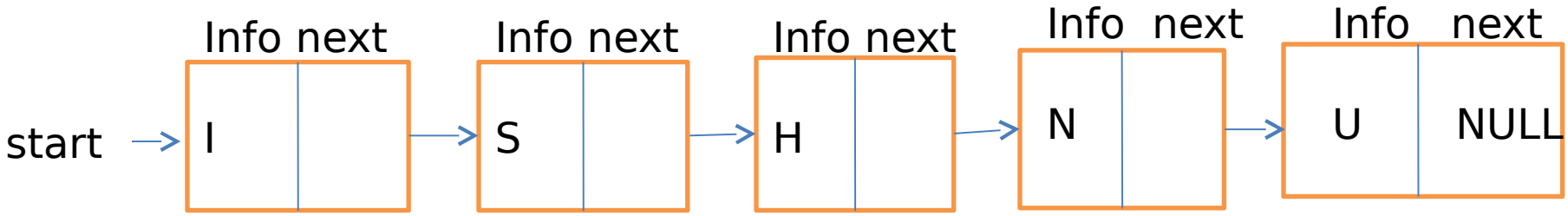

```
ptr = head
head = ptr -> next
free(ptr)
```

Deleting a node from the beginning

Singly Linked list operations cont..

Deleting the last element of the list

Assume the following list



If start = NULL **then**

Empty list

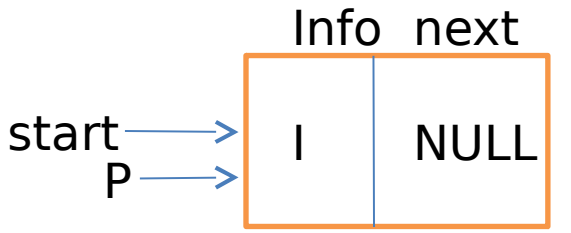
If start - -> next=NULL then

Set p = start

Set start = NULL

Char var = P - -> info

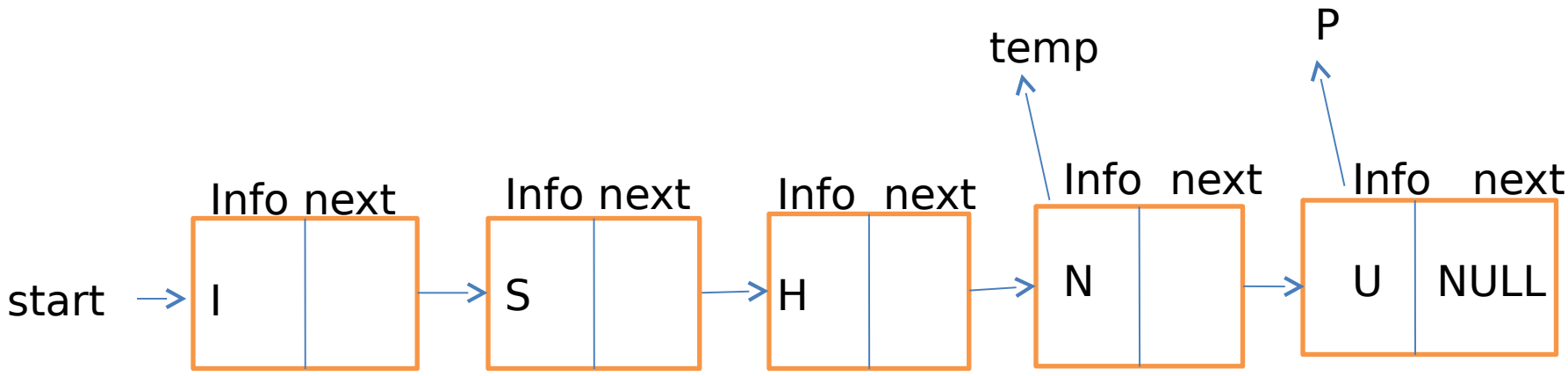
Free(p)



start = NULL

Singly Linked list operations cont..

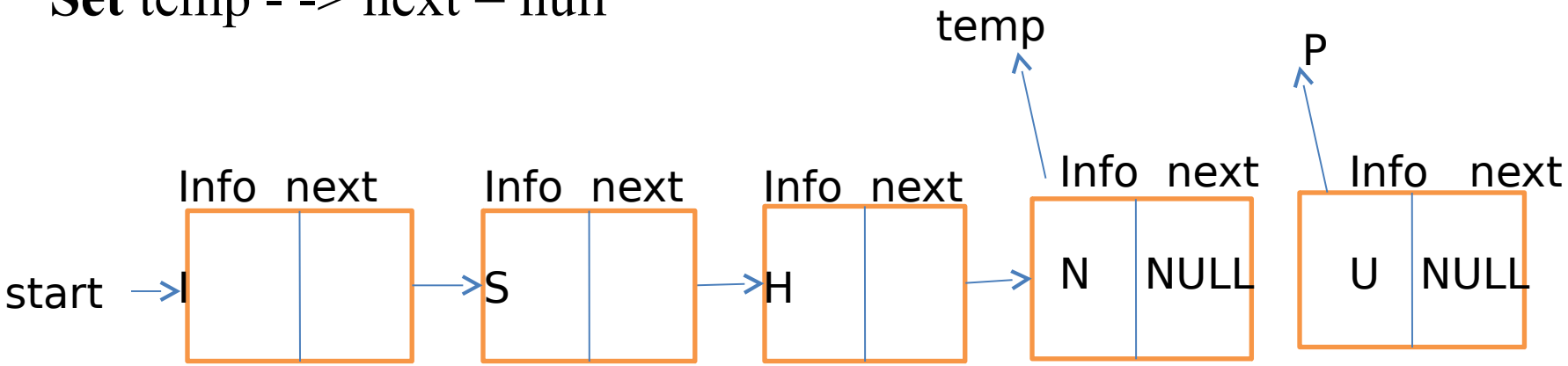
else
set p = start
Now to find the location, repeatedly do
until p- ->next ! = NULL
Set temp = p
Set p = p - ->next



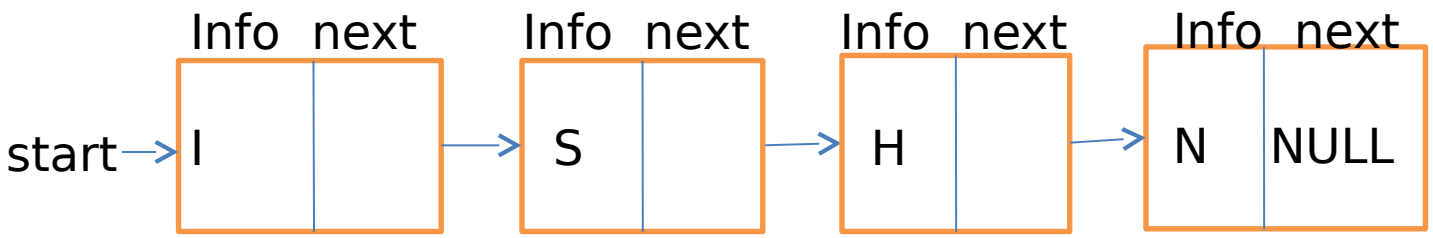
Singly Linked list operations cont..

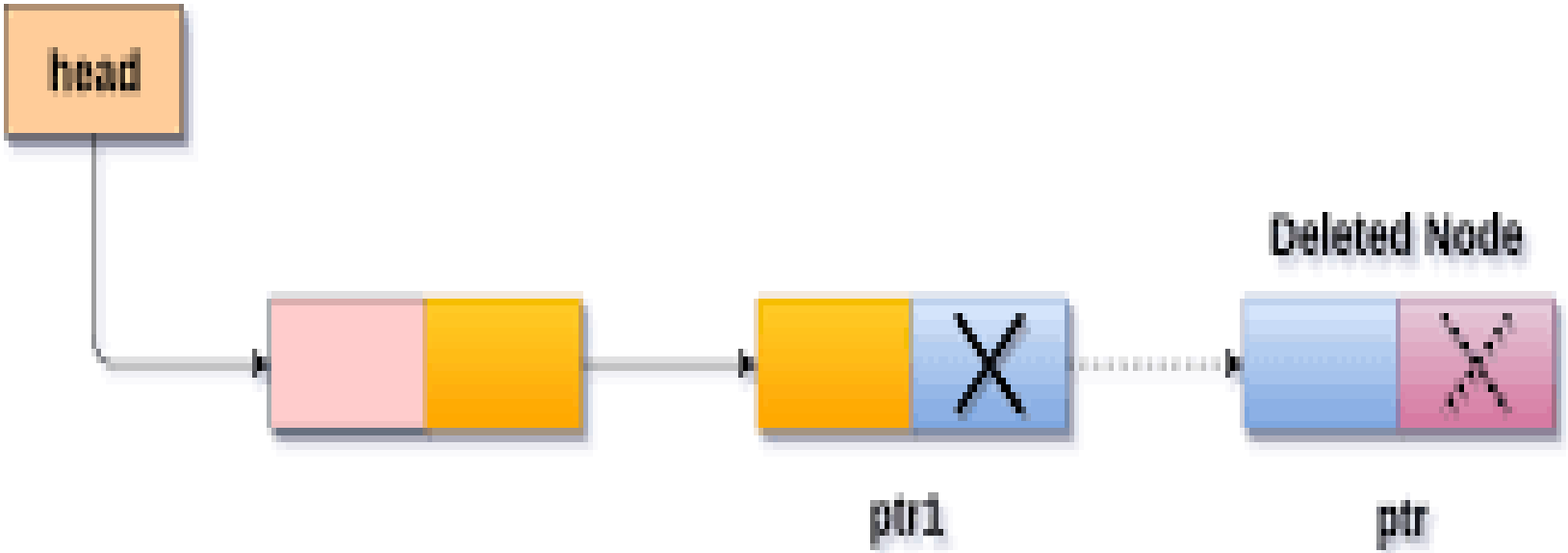
then

Set temp - -> next = null



Free(p)



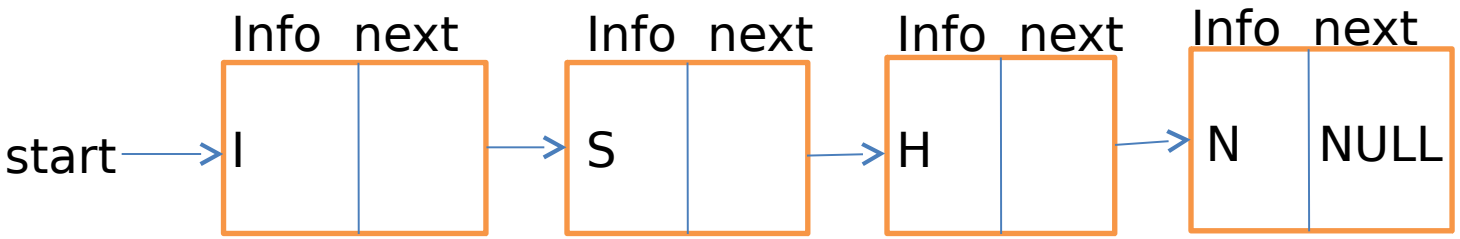


Deleting a node from the last

Singly Linked list operations cont..

Deletion at the specified position within the list

Assume the following list



Suppose we want to remove node at **3rd** position from start that is node H

If start = NULL **then**

Empty list

So to find that location

Set counter and pointer

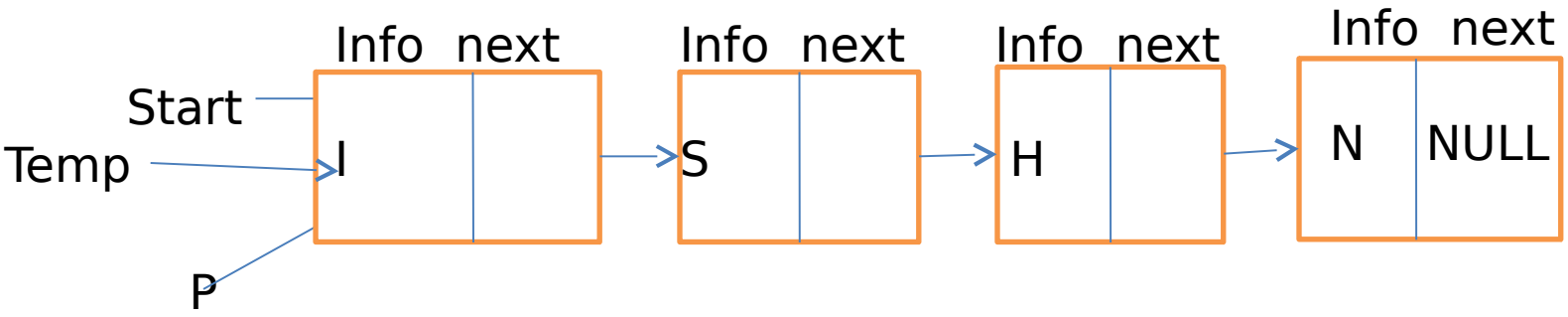
Set c=1

Set Node * temp , p

Set p=start

Singly Linked list operations cont..

Repeat following steps until c < location
Set temp = p
Set p = p --> next
Set c=c+1
Print p - -> info

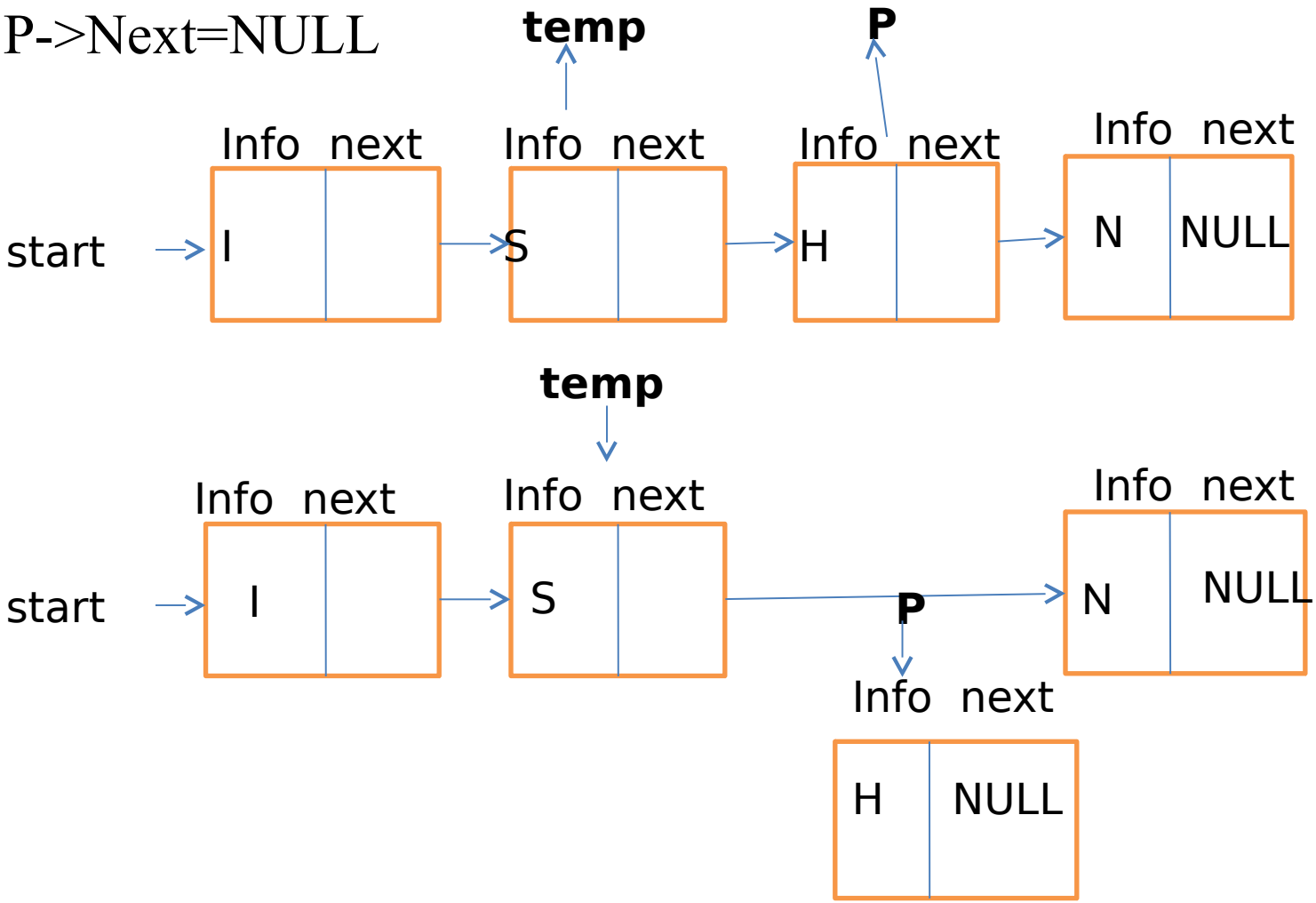


Singly Linked list operations cont..

At this stage C==3 required position

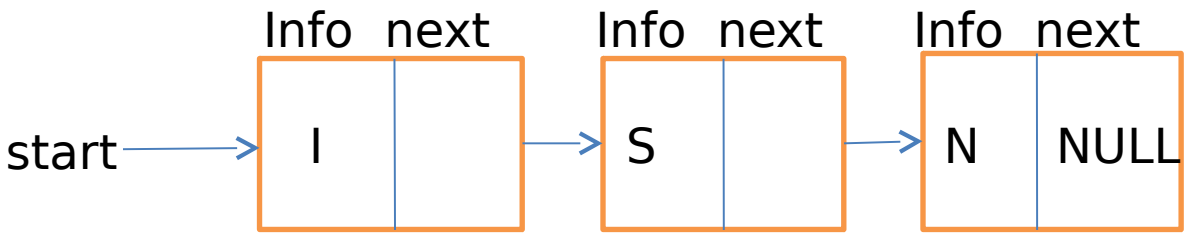
Set temp - ->next = p - -> next

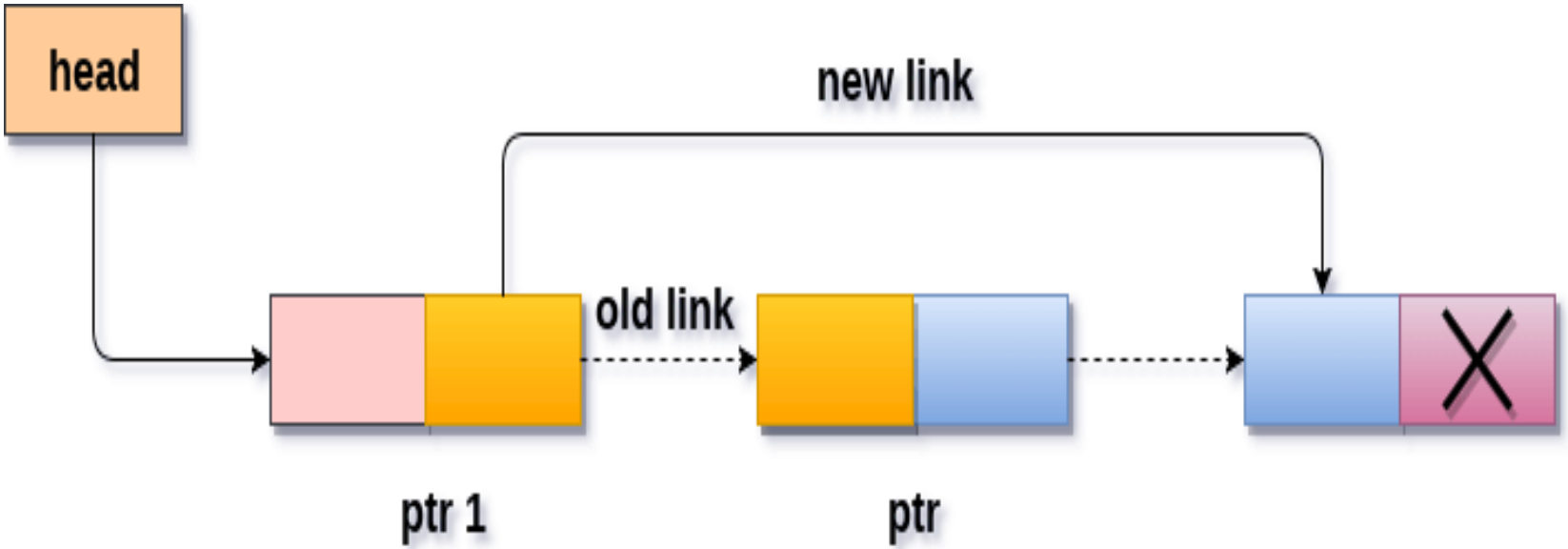
Set P->Next=NULL



Singly Linked list operations cont..

After Free(p)
Finally we have






```
ptr1 -> next = ptr -> next
free(ptr)
```

Deletion a node from specified position

Advantage and disadvantage of single linked list

Advantage

- Forward direction accessing of nodes are easy

- Insertion and deletions are easier and efficient that is, possible from any specified location

Disadvantage

- Forward traversal does not allow accessing the preceding node of the current node

- Access to arbitrary node is time consuming and tedious

Thank You

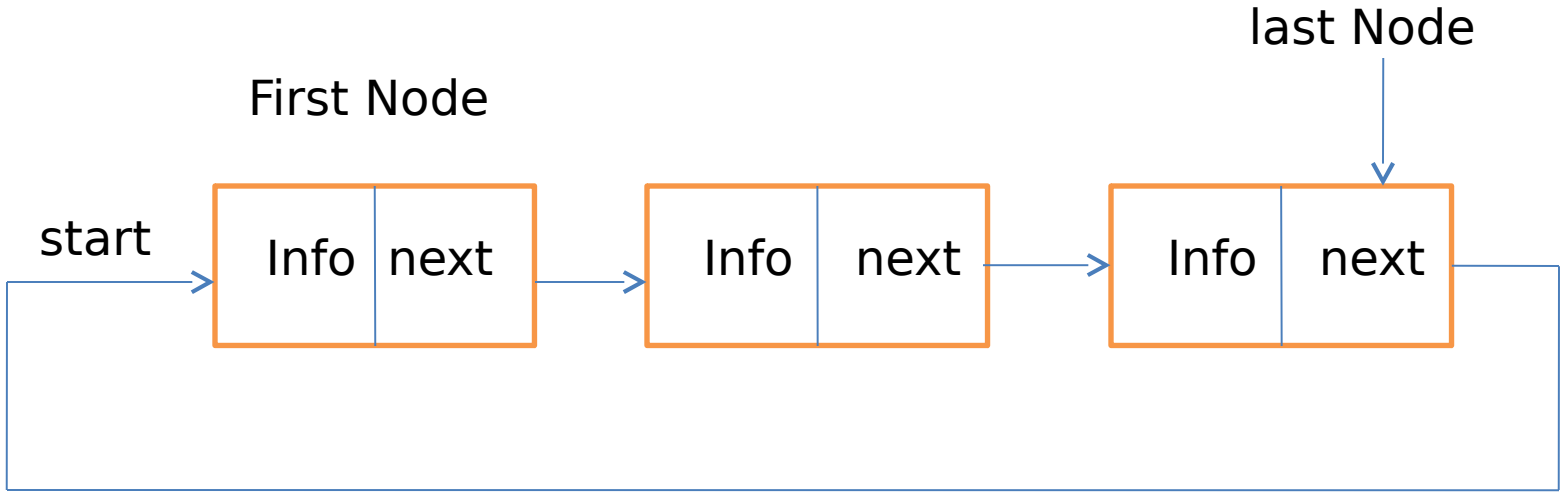
H R Choudhary Asstt Professor Dept. Of CSE, Engineering College Ajmer

Circular linked List

Definition

- Circular lists are like singly linked lists, except that the last node contains a pointer back to the first node rather than the null pointer.

Example



Circular linked List cont..

- Possible to reach from any point in such a list, to any other point
- Does not have a natural "first or "last" node
- Let, external pointer **start** point to the First node, and the following node be the Last node

Representation of circular linked list

```
struct node
{ int info;
struct node *next;
};
typedef struct node NODE;
NODE  *start= NULL;
NODE  *last= NULL
```

Circular linked List operations

Insertion and deletion in circular linked list

Inserting node at beginning

allocate space for a new node using **malloc()**

- copy the item into it,

p- ->info = V;

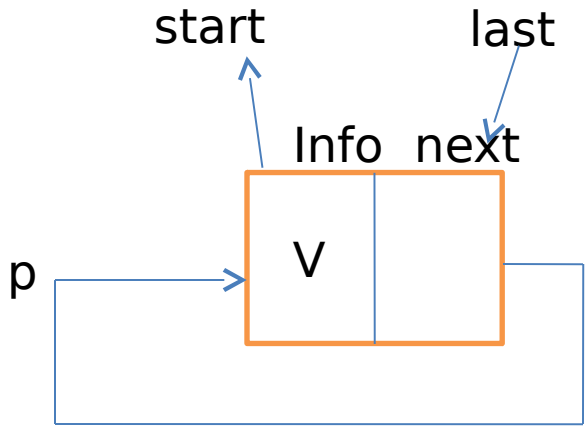
- make the following pointers adjustments

If start= = NULL

P - -> next = p;

start = p;

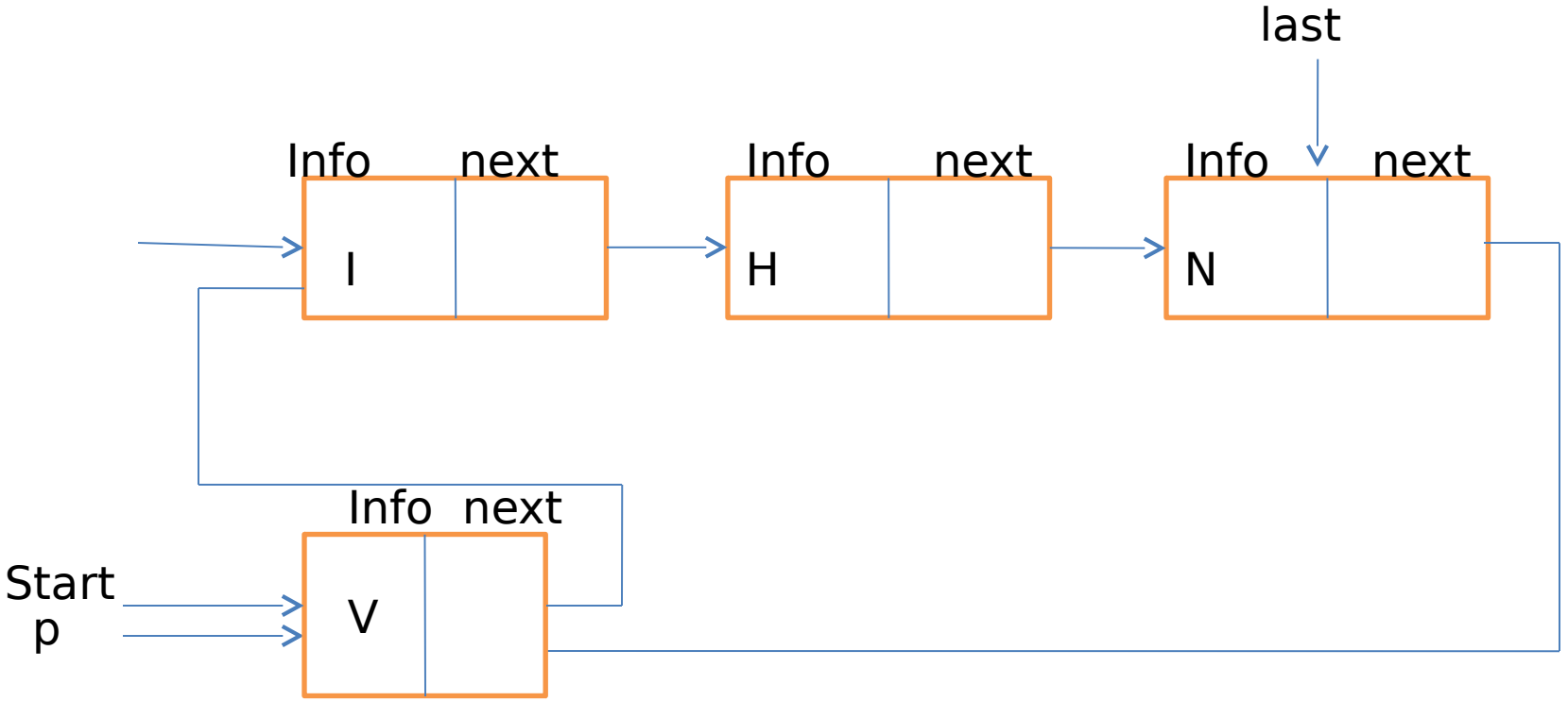
last = p;



Insertion and deletion in circular linked list cont..

else

- p - -> next = start;
- start = p;
- last - - > next = p;

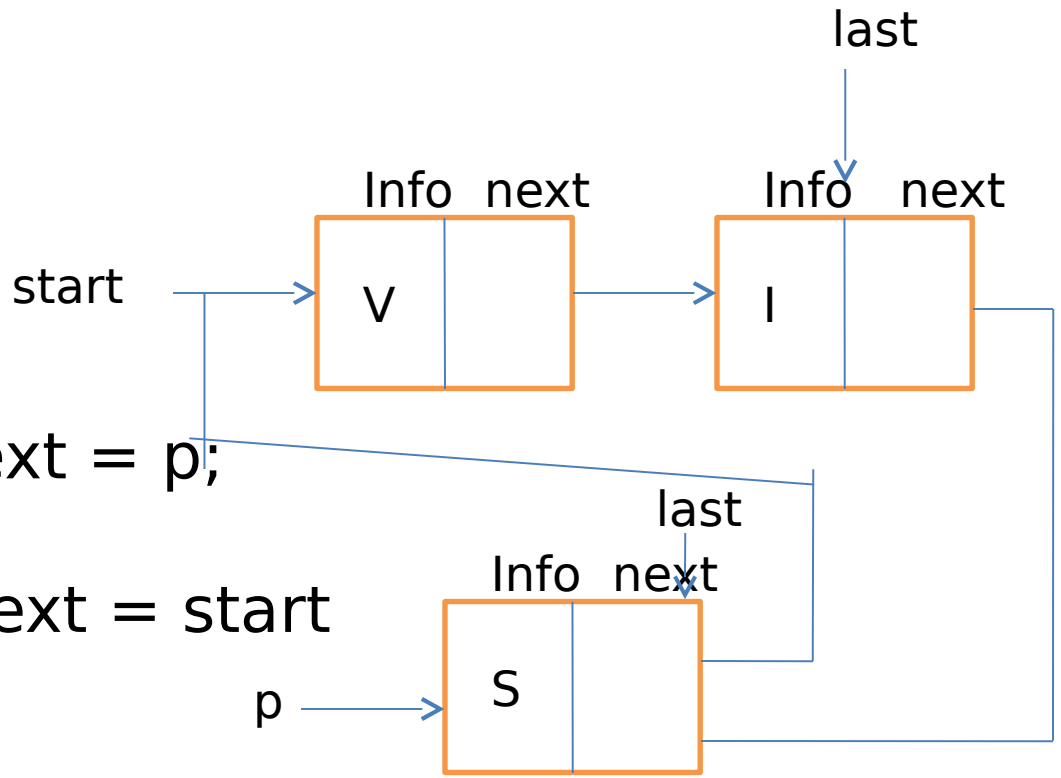


Insertion and deletion in circular linked list cont..

Inserting node at the end of list

If start = NULL **then**
Set P - -> next = p;
set last = p;
set start = p;

else



Set last - -> next = p;
Set last = p;
Set last - - > next = start

Insertion and deletion in circular linked list cont..

Algorithm for deleting a node from the beginning of the list

If start = NULL
list empty
else
Set p = start
Set start = start - - > next
Char var = p - -> info
Set last - -> next = start
Free(p)

Note: In same way other operations can also be performed on circular list

Advantage and disadvantage of circular linked list

Advantage

- Possible to reach from any point in such a list, to any other point
- Able to add or remove an element from either the front or rear of a list

Disadvantage

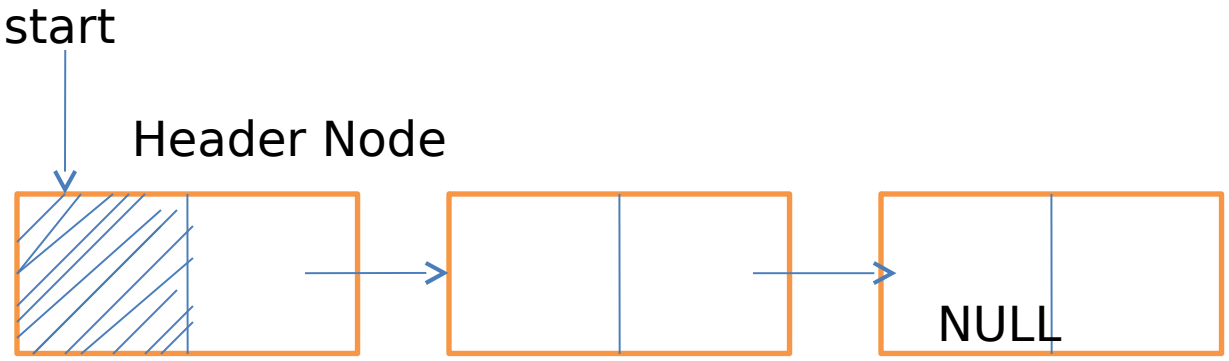
- Inability to delete a node, given only a pointer to that node
- Inability to traverse the list in the backward direction

Header linked list

Introduction

- A header linked list is a linked list which always contains a special node called the *header node* at the beginning of the list.
- It is an extra node kept at the front of a list.
- Such a node does not represent an item in the list.
- The information portion might be unused or contains a flag.

Example



Simple list with a header Node

Header linked list cont..

- The external pointer to the list is to its header node
- The information portion of header node could be used to keep global information about the entire list such as:
- number of nodes (not including the header) in the list
- count in the header node must be adjusted after adding or deleting the item from the list
- pointer to the last node in the list

it simplifies the representation of a queue

pointer to the current node in the list

- eliminates the need of a external pointer during traversal

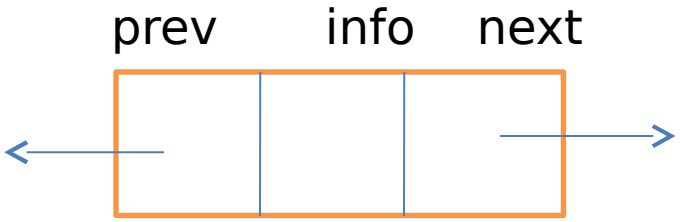
Doubly linked list

Introduction

It is a bidirectional list of structures(node) in which all nodes are linked together and contain three part.

- It can be either linear or circular and may or may not contains header node
- **INFO** part stores the information
- One **pointer** which points to the next element(successor)
- Other **pointer** points to the previous node (predecessor)

Example



A node of doubly linked list

Doubly linked list cont..

Representing doubly linked list nodes

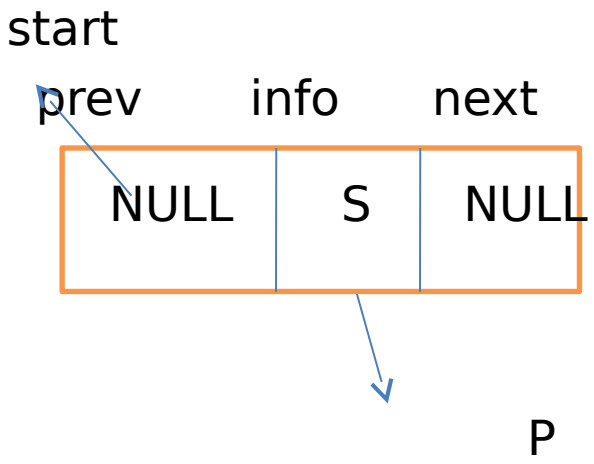
```
Struct node
{
Char info;
Struct node * prev;
Struct node * next;
};
typedef struct node NODE;
```

Doubly linked list cont..

Inserting and deleting in doubly linked list

Algorithm for inserting node at the beginning

allocate space for a new node using **malloc()**
copy the item into it,
p- ->info = S;
NODE * prev, * next;
If start = = NULL;
p - - > prev = p - - > next = NULL;
start = p;



Inserting and deleting in doubly linked list cont..

else

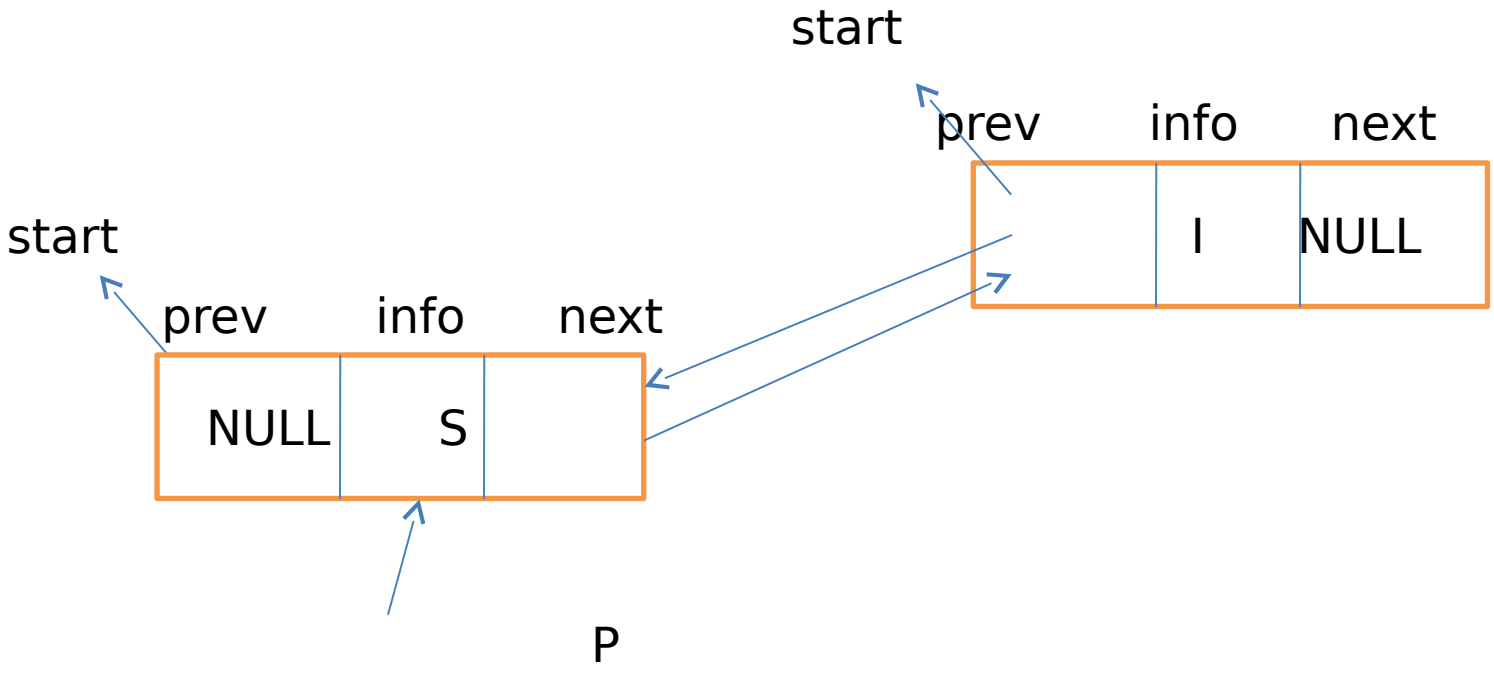
p - - > prev = NULL;

p - - > next = start;

start - - > prev = p;

start = p;

assume following list



Inserting and deleting in doubly linked list cont..

Deleting a node from the beginning of a doubly linked list

```
 If  start = = NULL;
return;
 else if
 start - - > next = = NULL;
 p = start;
start = NULL;
else
 p = start;
start = start - - > next;
start - - > prev = NULL;
free(p);
```

Advantage and disadvantage of circular linked list

Advantage

- Forward and backward accessing of nodes makes easy accessibility of nodes

- Insertion and deletions are easier and efficient than other list

Disadvantage

- It uses two pointers, so increases the memory requirement

Thank You

H R Choudhary Asstt Professor Dept. Of CSE, Engineering College Ajmer