Implementation of queue using linked list

Submitted to
Nemi Chandra Rathore
HEAD OF THE DEPARTMENT

Presented by
Balmukund jha
Mehul Mohan

CONTENTS

WHAT IS QUEUE?

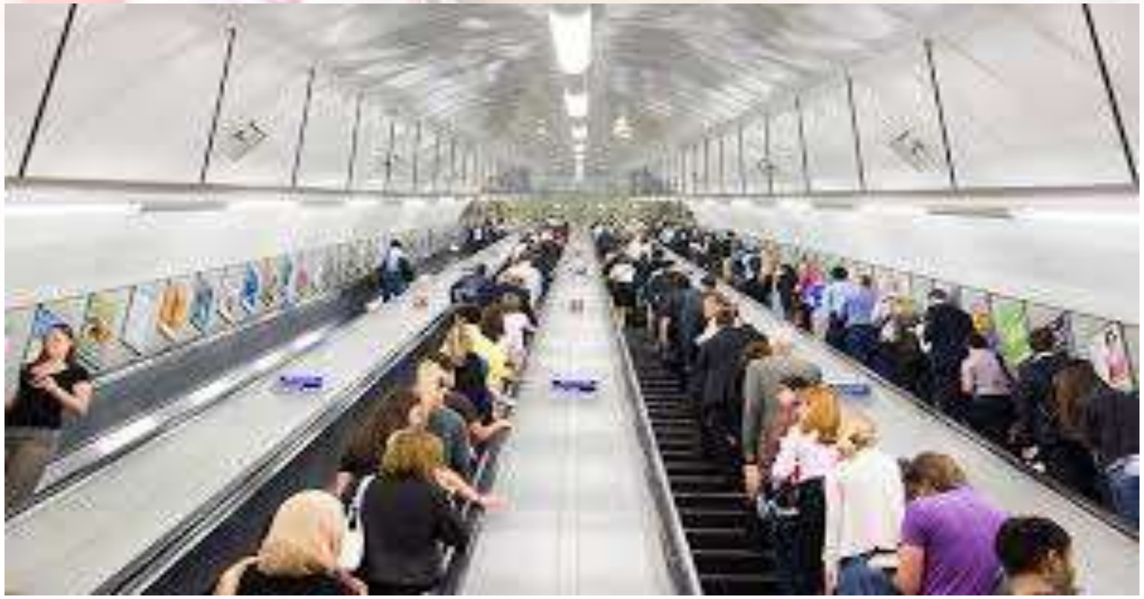
- Queue is a linear data structure.
- It is used for temporary storage of data values.
- A new element is added at one end called rear end .
- The existing elements deleted from the other end called front end.
- First in first out property.

Real world example of queue

- At bank counter.
- People on an escalator.
- At ATM machine.
- In one-way road.

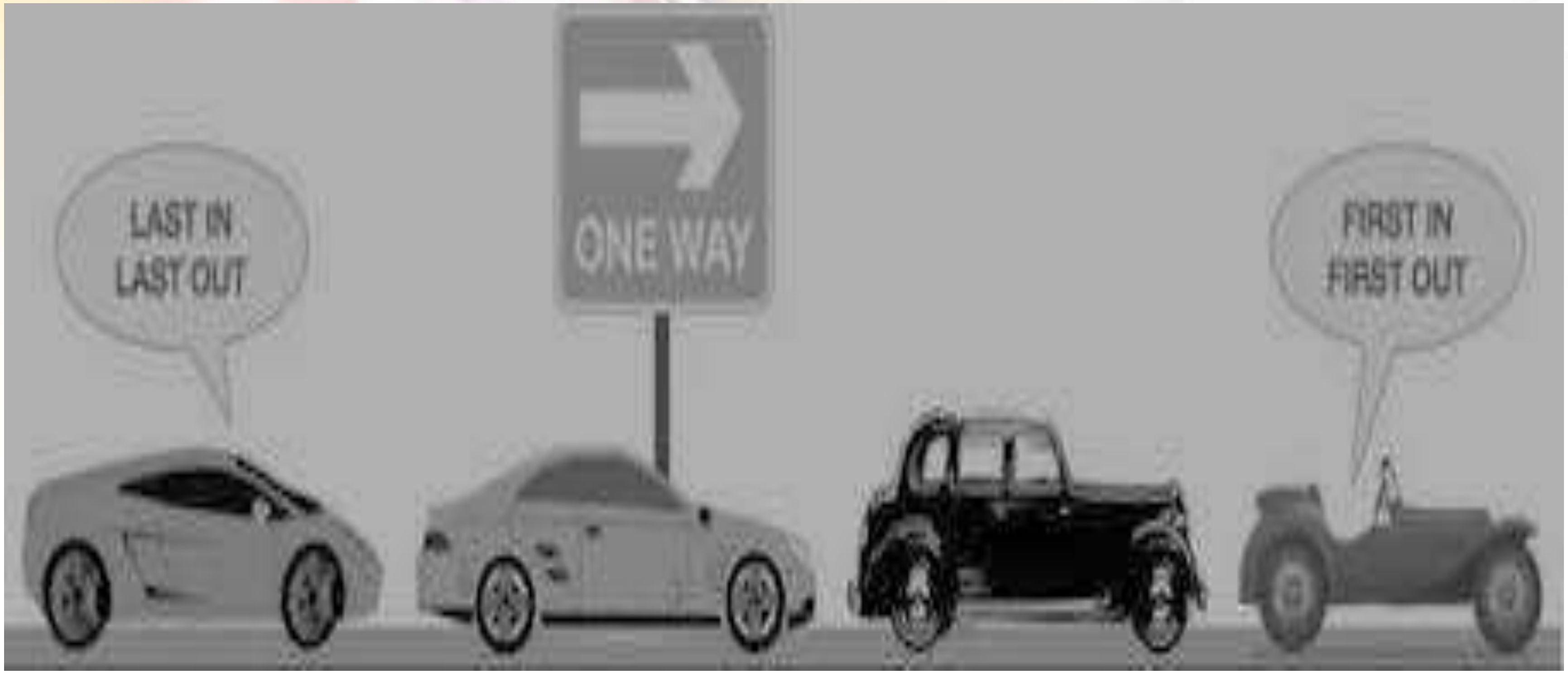
At bank counter

People on an escalator.



Queue at ATM

At one-way road



Application of queue

- Queues are widely used as waiting lists for a single shared resource like printer, disk, cpu.
- Queues are used to transfer data asynchronously between two processes , e.g., file IO, sockets.
- Queue are used as buffers on MP3 players and portable CD players , iPod playlist.
- Queues are used in playlist for jukebox to add songs to the end , play from the front of the list.

Queue implementation

<u>Implementation</u>

*array based (linear or circular)

*pointer based : linked list

ADVANTAGE OF LINKED LIST OVER ARRAY

- Linked list provides two advantages over array

*Dynamic memory allocation.

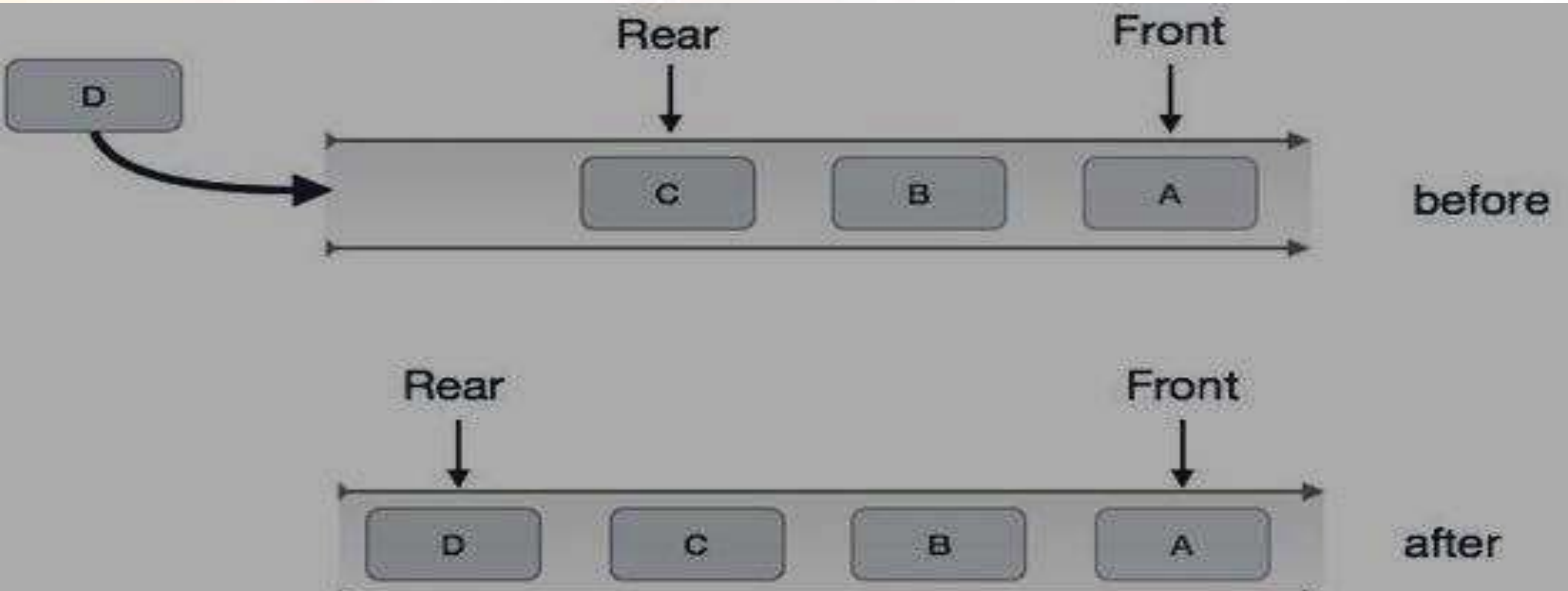
*Ease of insertion and deletion.

THERE ARE TWO OPERATIONS ON QUEUE

- ENQUEUE(INSERTION AT REAR)
- DEQUEUE(DELETION FROM FRONT)

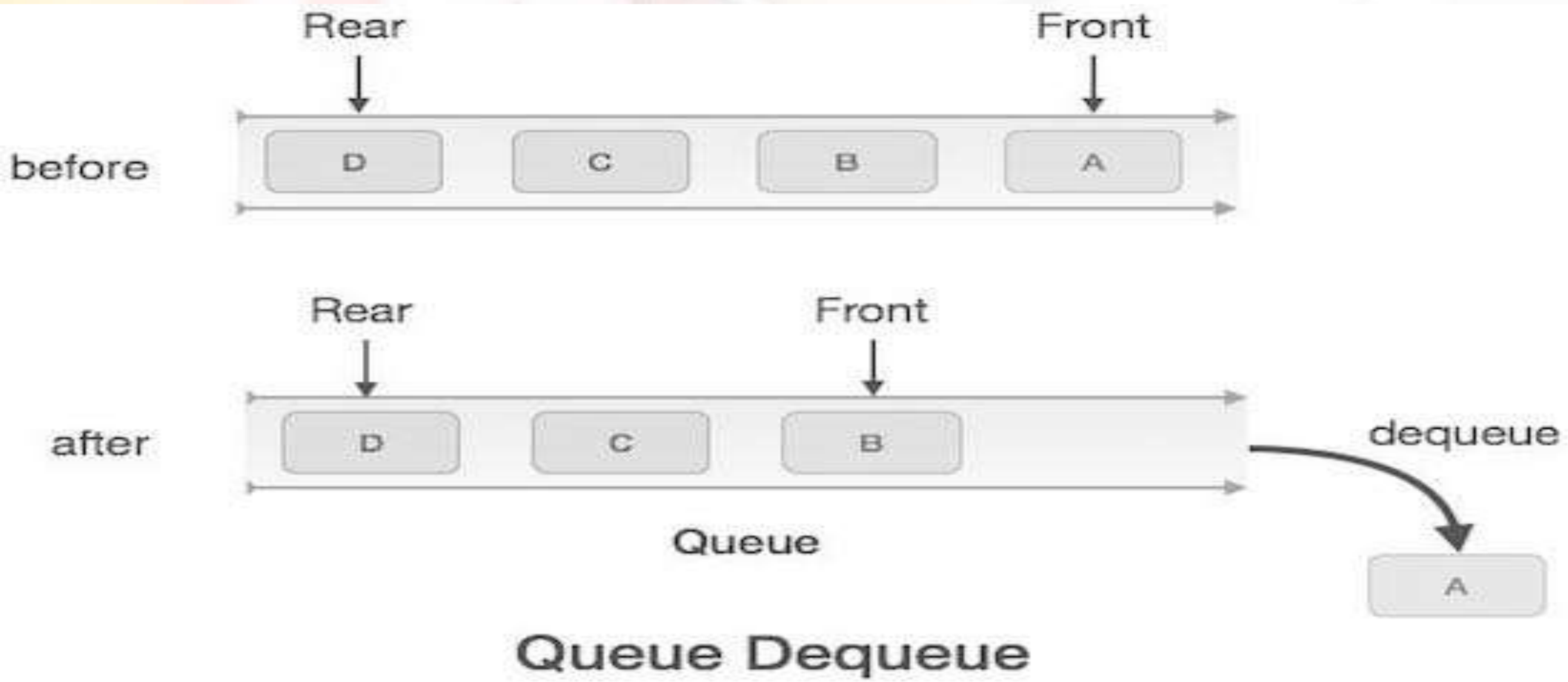
ENQUEUE

Placing an item in a queue is called "insertion or enqueue", which is done at the end of the queue called "rear".



DEQUEUE

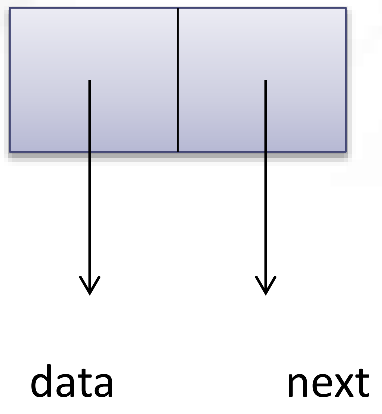
Removing an item from a queue is called “deletion or dequeue”, which is done at the other end of the queue called “front”.



IMPLEMENTATION OF QUEUE USING SINGlY LINKED LIST

- Create queue implementation using linked list

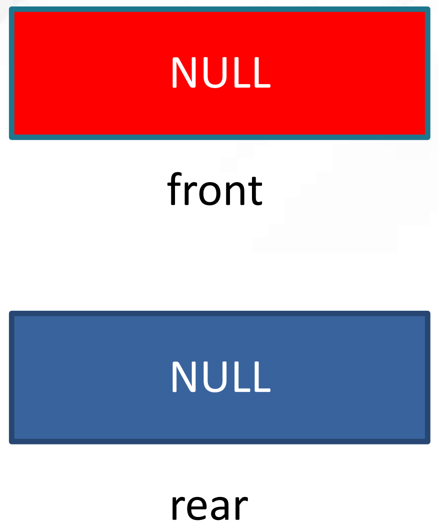
```
struct queue
   {
              int data ;
              struct queue *next;
   }
```



Create queue implementation using singly linked list

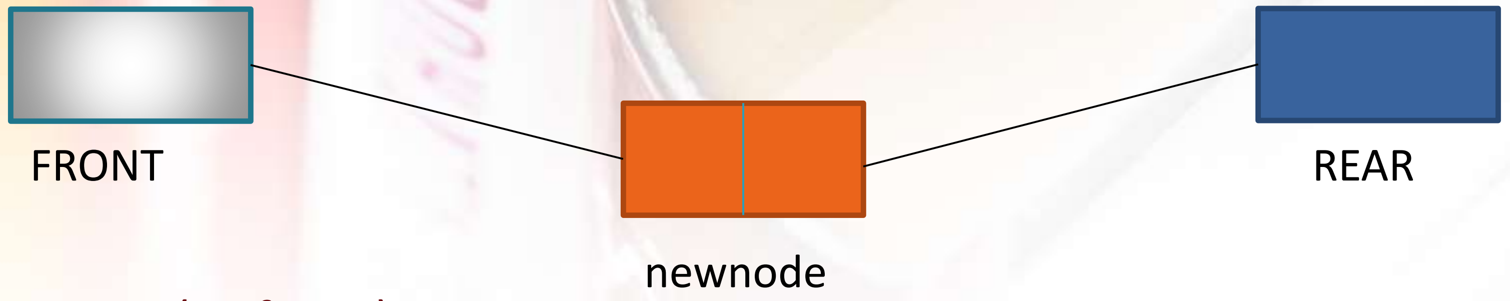
```
struct queue *front = NULL;

struct queue *rear = NULL;
```



Enqueue

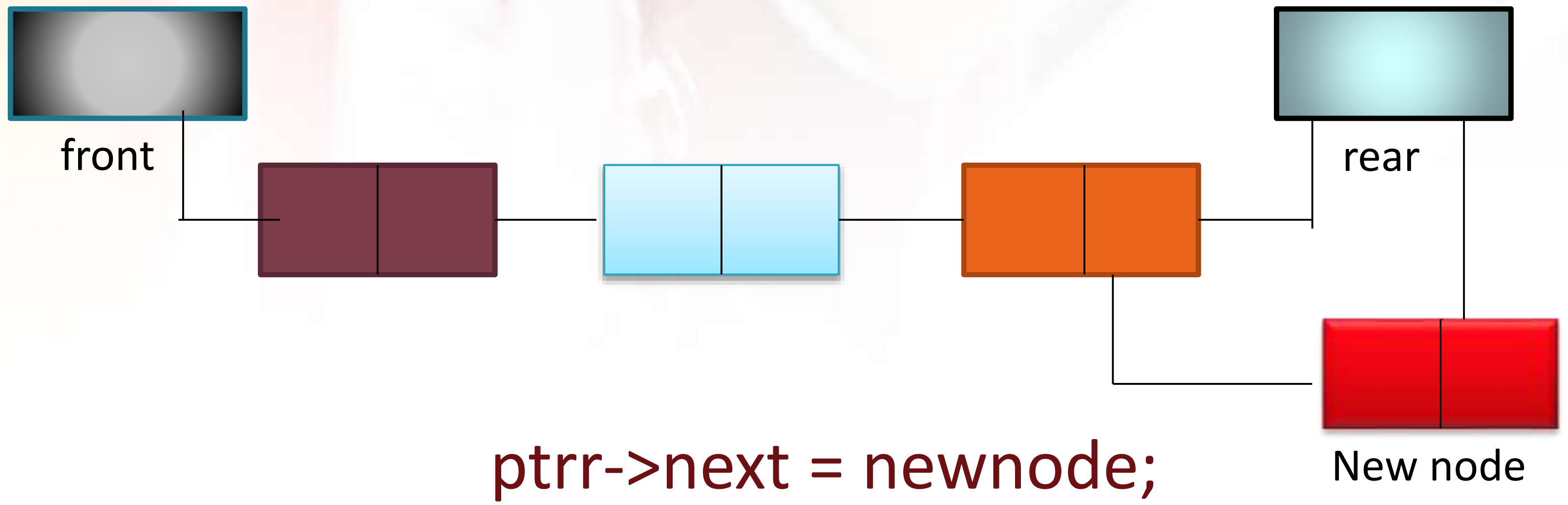
If there is no node present in the queue



```
enqueue(ptrf, ptrr)
{
                if(ptrf==NULL)
        {
                ptrf=newnode;
                ptrr=newnode;
                newnode->next = NULL;
        }
}
```

Enqueue

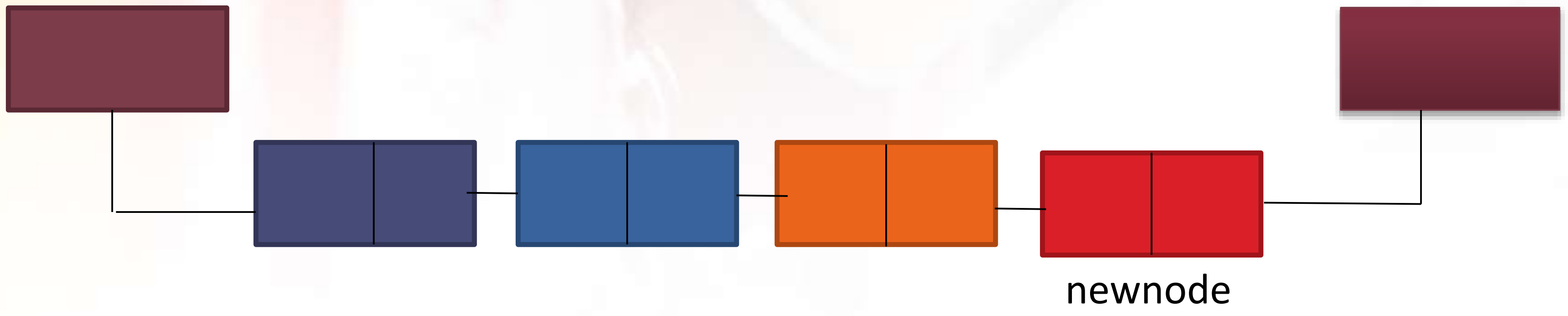
- If there is one node and multiple node in the queue



```
ptrr->next = newnode;
newnode->next = NULL;
ptrr = newnode;
```

After insertion of a node in the queue

- If there is one node and multiple node in the queue



Dequeue

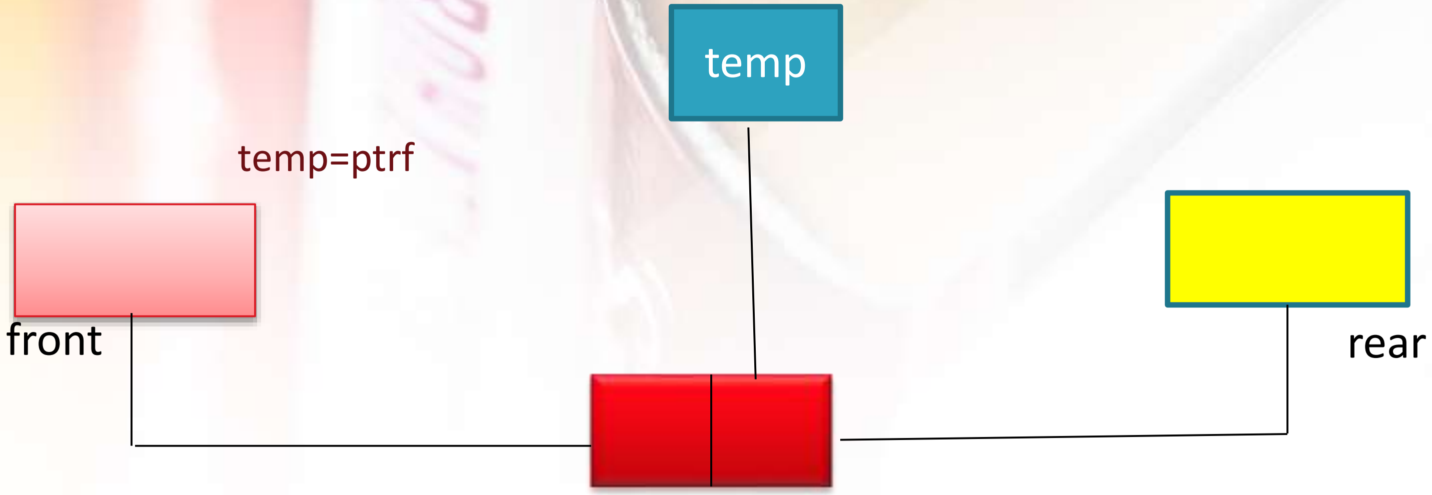
- If there is no node in the queue.
- Message should be prompted to the user.

```
Dequeue(ptrf, ptrr)
{
if(ptrf==NULL)
          {
          printf("\n there is no data present in the queue");
          }
}
```

Continue…

- If there is one node available in the queue.

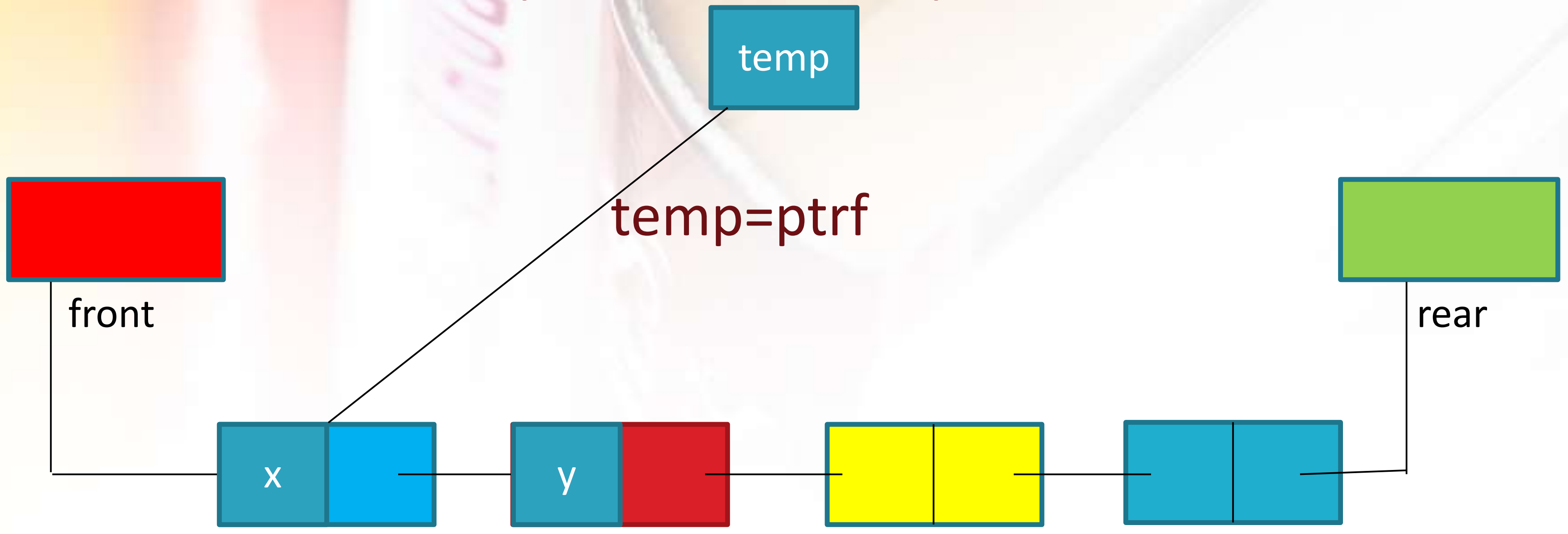
- temp=ptrf



```
if(temp->next ==NULL)
        {
                ptrf = NULL;
                ptrr =NULL;
                free(temp);
        }
```

Continue…

- If there are multiple node in the queue.

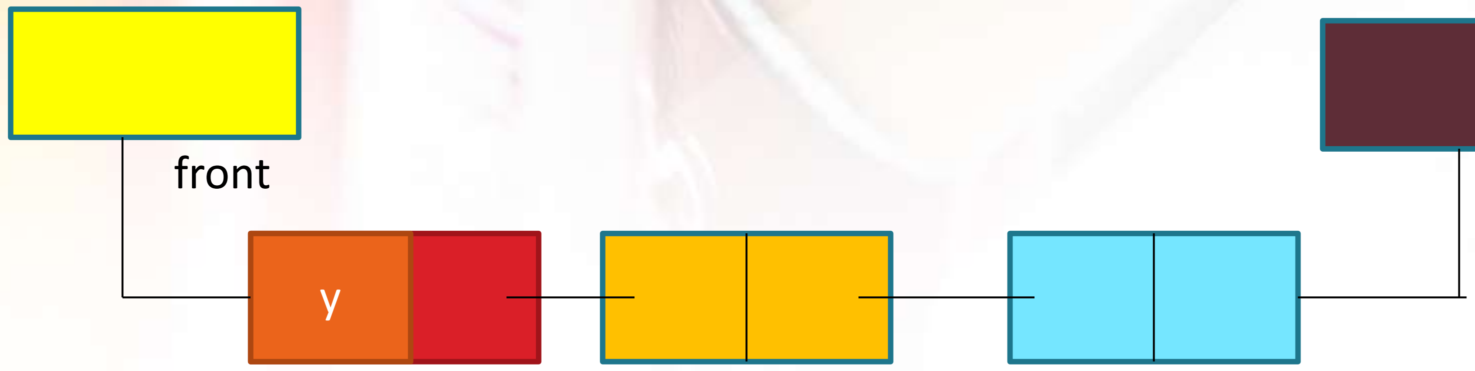


ptrf= temp->next;

free(temp)

After deletion

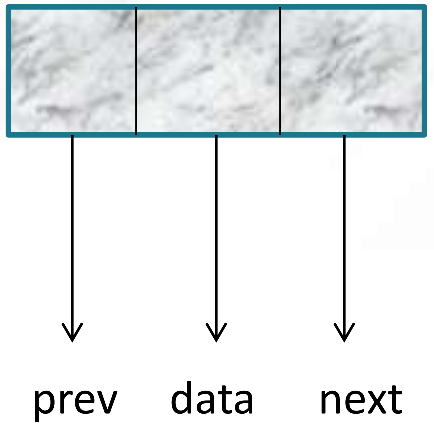
- .



IMPLEMENTATION OF QUEUE USING DOUBLY LINKED LIST

- Create queue implementation using doubly linked list

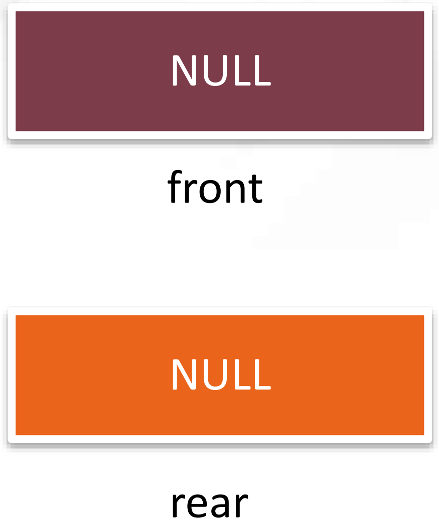
```
Struct queue
{
            int data ;
            struct queue *next;
            struct queue *prev;
 }
```



Create queue implementation using doubly linked list

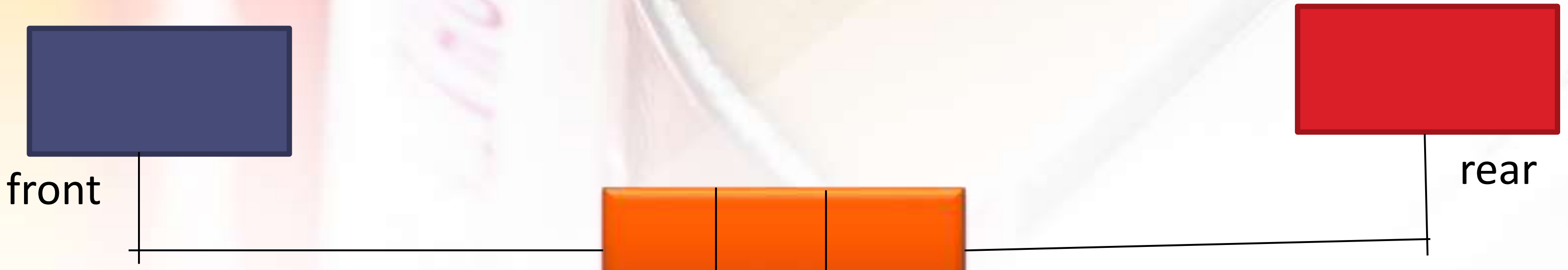
```
struct queue *front = NULL;

struct queue *rear = NULL;
```



Enqueue

If there is no node in the linked list.
We need to change the front and rear pointer.

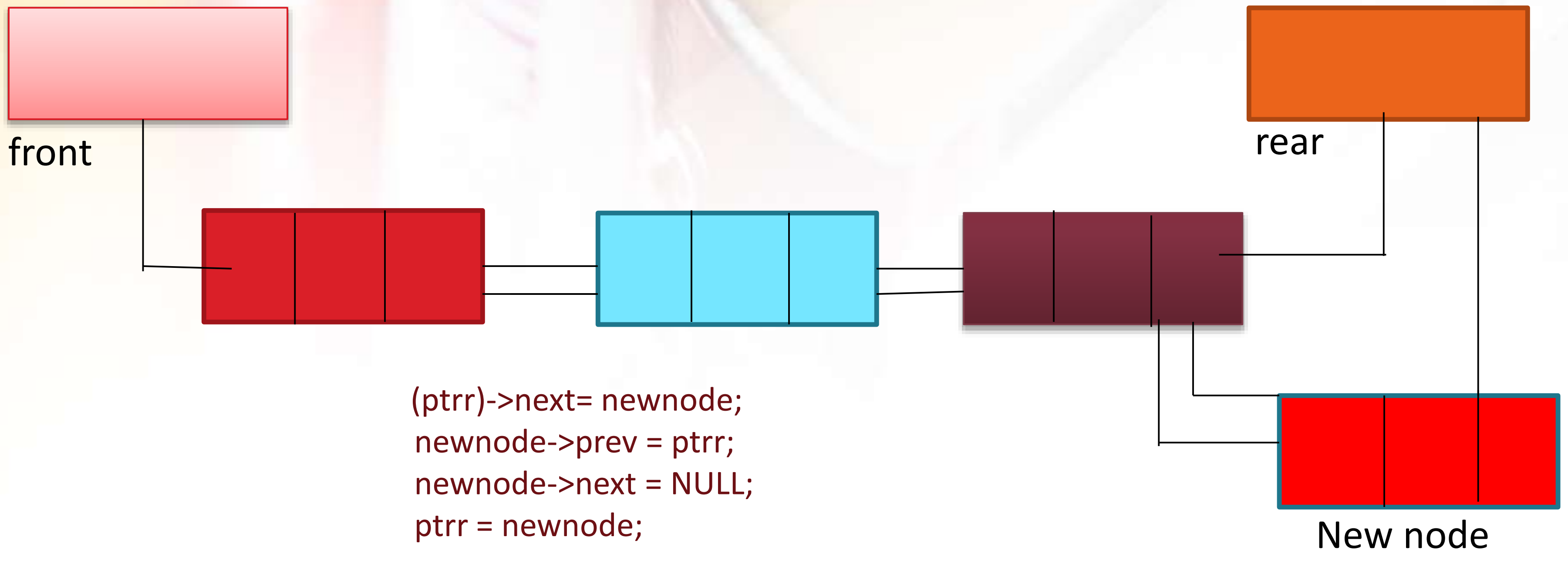


```
if(ptrf==NULL)
        {
                ptrf = newnode;
                ptrr = newnode;
                newnode->next = NULL;
                newnode->prev = NULL;

        }
```

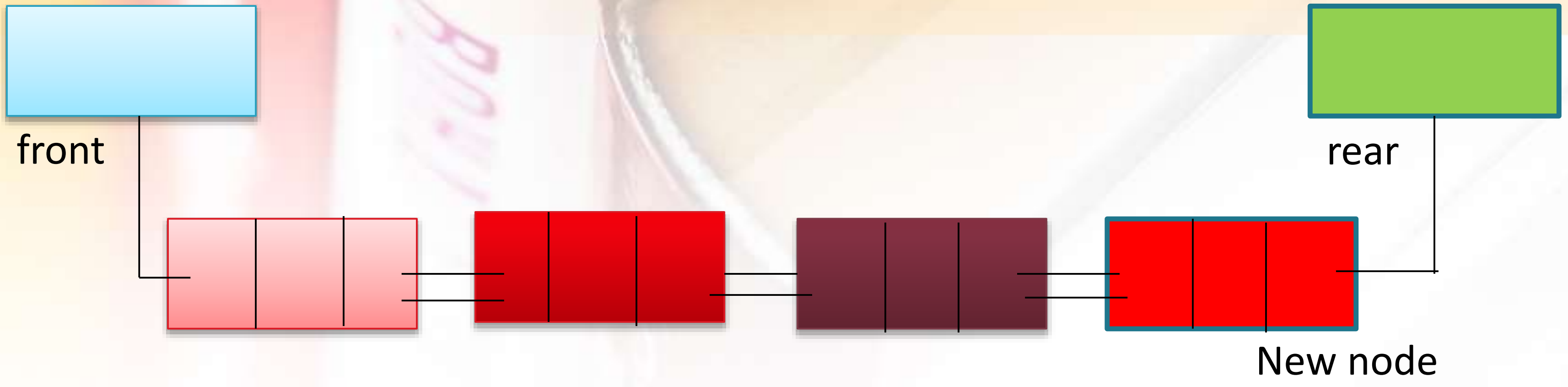
Continue...

- If there is one or multiple node in the queue.



```
(ptrr)->next= newnode;
 newnode->prev = ptrr;
 newnode->next = NULL;
 ptrr = newnode;
```

After insertion…



Dequeue

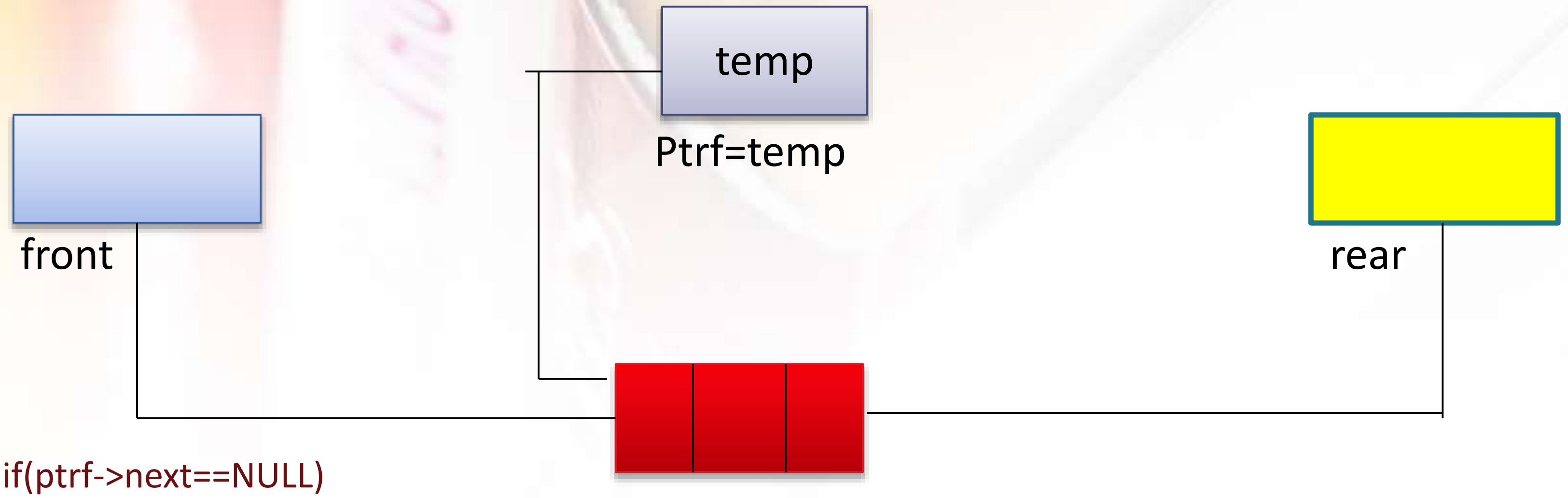
- If there is no node in the queue.
- We should prompt message .

```
        if(ptrf==NULL)
        {
                printf("\n there is no data present in the queue");


        }
```

Continue…

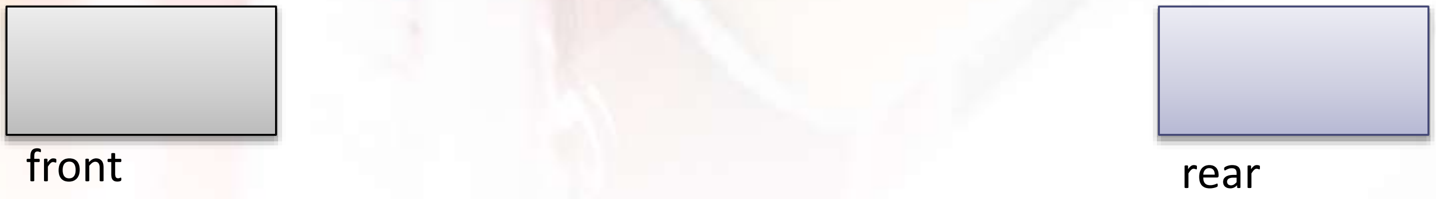
- If there is one node in the queue.



```
if(ptrf->next==NULL)
        {
                        ptrf =NULL
                        ptrr =NULL
                        free(temp)
        }
```

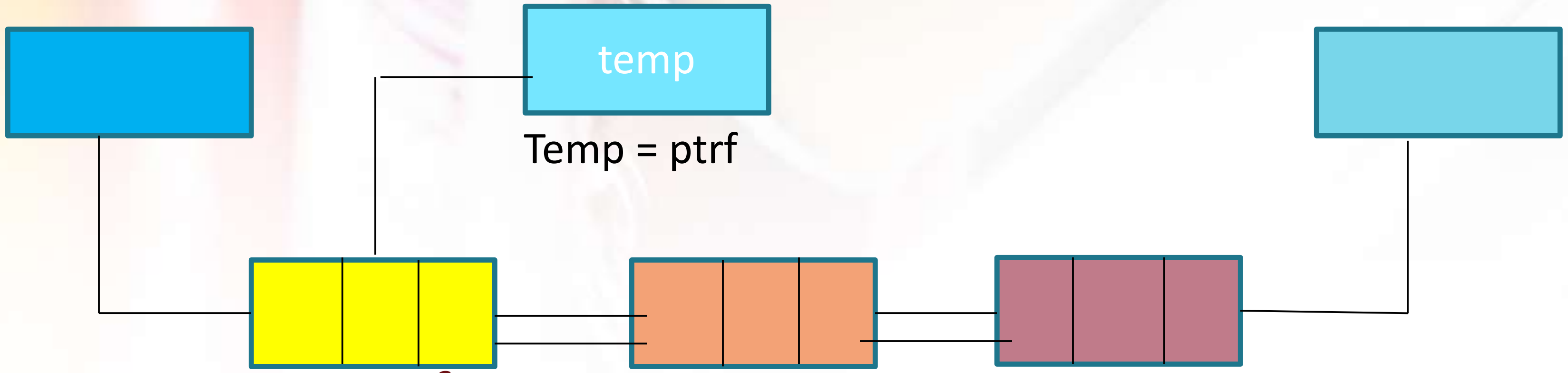
Continue…

- After deleting



Continue…

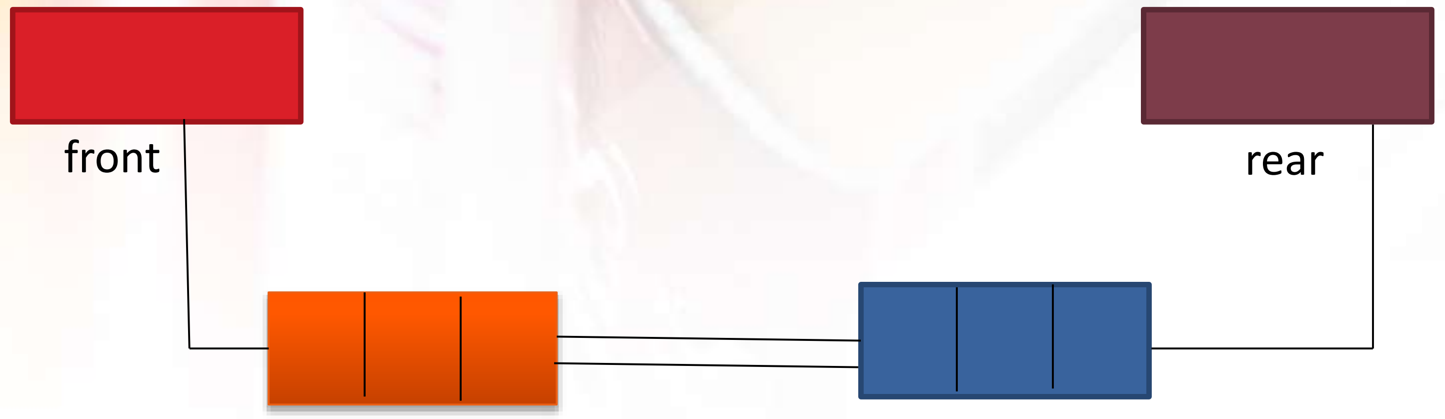
- If there are multiple node in the queue.



```
temp = ptrf;
          (temp->next)->prev = NULL;
          ptrf = temp->next;
          free(temp);
```

After deleting

- ->



Time complexity

Singly linked list	Doubly linked list
Enqueue(O(1))	Enqueue(O(1))
Dequeue(O(1))	Dequeue(O(1))
Traverse(O(n))	Traverse(O(n))
Travresal is not an easy task	Traversal is easy

summary

- Queue is a linear data structure in which insertion is performed from rear and deletion is performed from front end .
- A queue is a FIFO based data structure.
- Doubly linked list is more efficient to implement a queue because of easy traversal.

THANK YOU !