Data Structures and Algorithms

MSc. KhangVQH

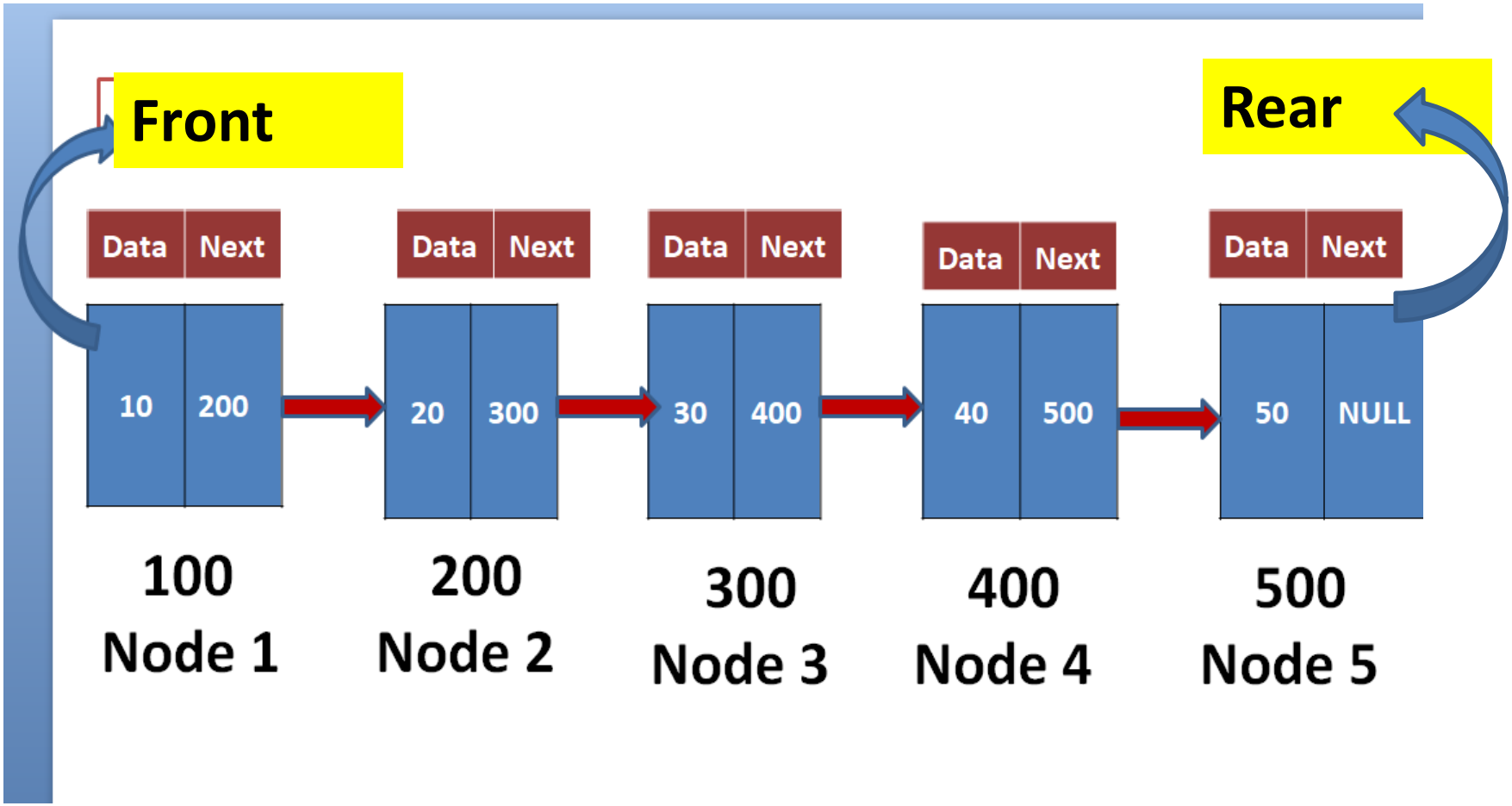
Faculty Of Information Technology

Fall - 2022

Queue Linked List

Terminology: Front, Rear, Node, Data, Next, NULL
Operations: Enqueue( ), Dequeue( ), Display()

Queue using SLL



Content

- What is Queue Single linked list
- Queue working principle
- What is node?
- Node Structure.
- Operations on Queue SLL.

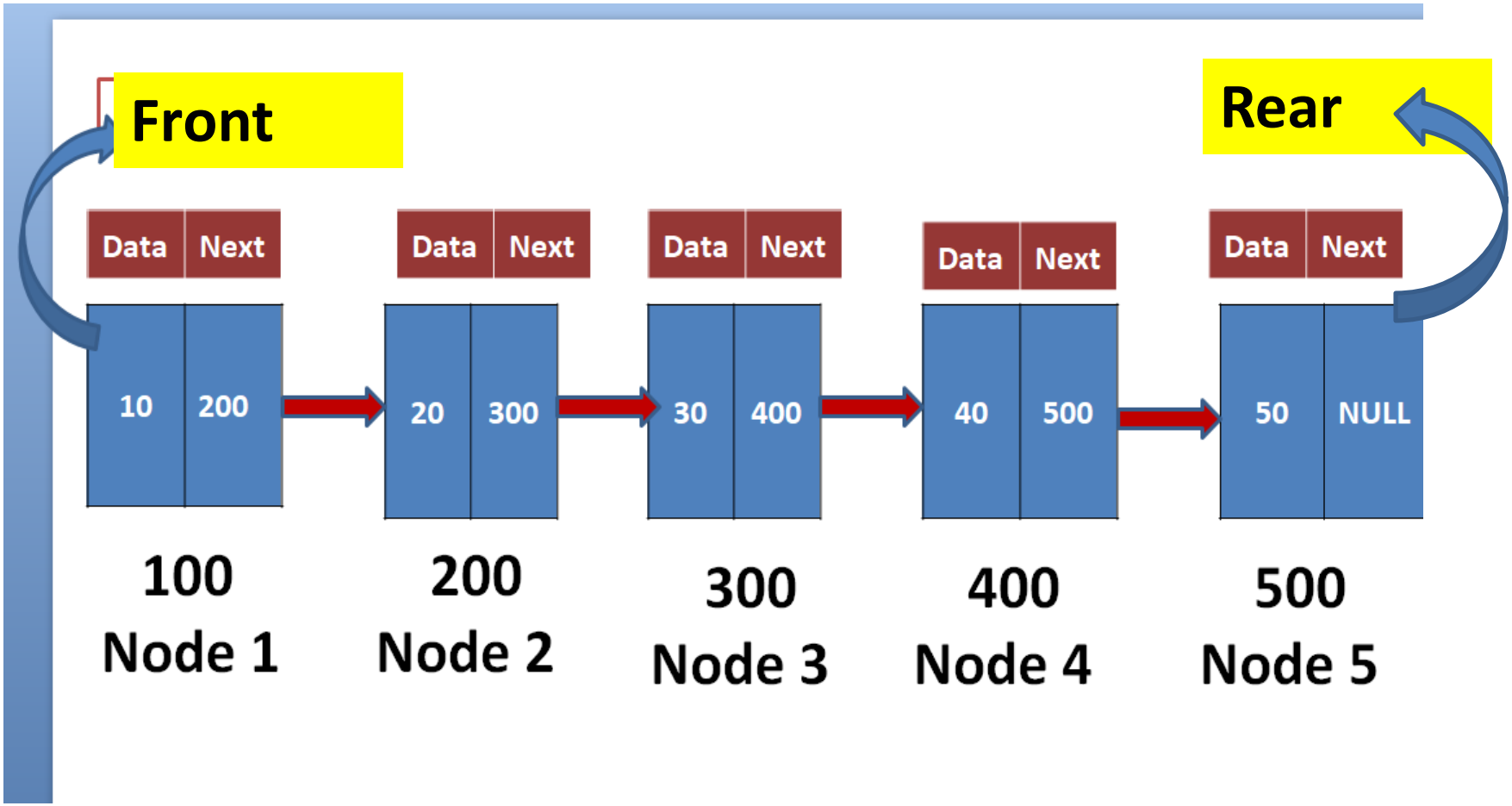
What is Queue Single linked list

- **List:** collection of number of elements
- **SLL**: SLL is linear Data Structure.
- It is also a collection of elements(nodes) but every element is linked with next element(node) by address.
- **Queue SLL**: Implement the queue data structure with single linked list concept.
- The Problem with queue using array is fixed size. But Queue using SLL is unlimited size.

What is Queue Single linked list

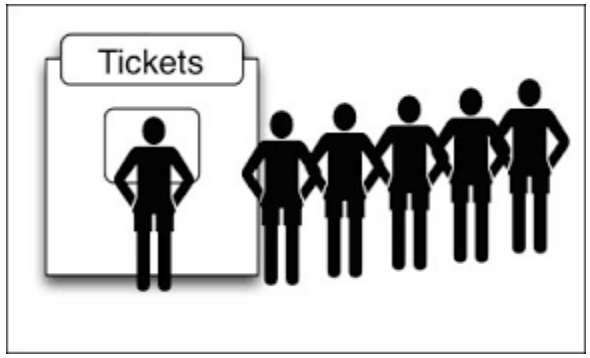
- Queue SLL uses two pointer fields
- 1) Front: it is used for delete the elements from the list
- 2) Rear: it is used to insert the elements into the list

Queue using SLL



Queue working Principle

- In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.



What is node?

- Every single element in a List is called “Node”.
- Node contains two fields

1) Data filed-it holds data(element value)

2) Next field- it holds address of next node

- Every node has it’s own address value in the memory

Node
100

Node Structure

```
Structure Node
{
Int data;
Structure Node *Next;
}*front=NULL,*rear=NULL;
```

Node

100

Operations on SLL.

- **Enqueue():** it is used to insert the node at Rear.
- **Dequeue():** it is used to delete the node at Front.
- **Display():** it is used to display the nodes from the list.

Algorithm for Enqueue()

enqueue(value) : It is used inserting an element into the Queue

Step 1 : Create a **NewNode** with given value and set '**NewNode → next**' to **NULL**.

Step 2 : Check whether queue is **Empty** (**rear** == **NULL && front==NULL**)

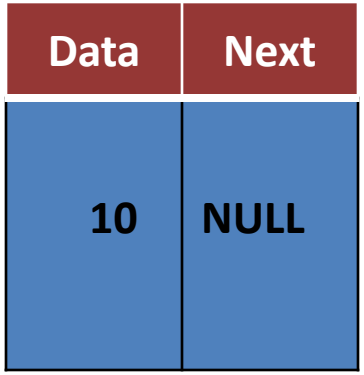
Step 3 : If it is **Empty** then, set **front** = **NewNode** and **rear** = **NewNode**.

Step 4 : If it is **Not Empty** then,

set **rear → next** = **NewNode** and **rear** = **NewNode**.

Connecting nodes by address

1)Before creating first node :: Assign **front=rear=NULL**



100

Node 1

```
Node1=(*struct Node)malloc(sizeof(*struct Node);

Node1->data=10;
Node1->next=NULL;
```

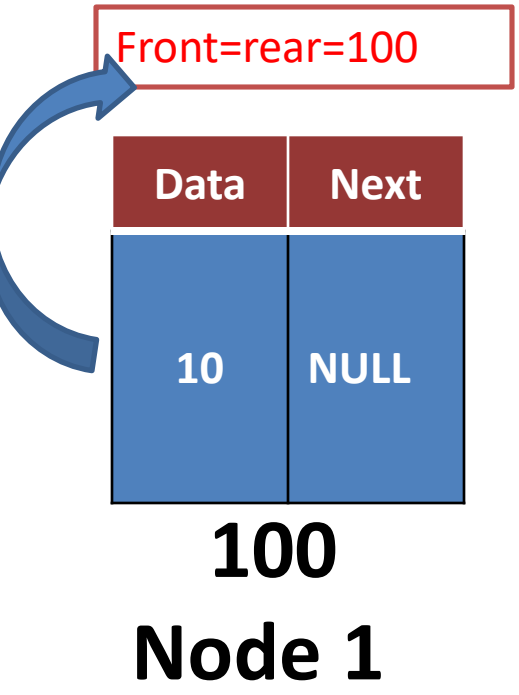
i.e:

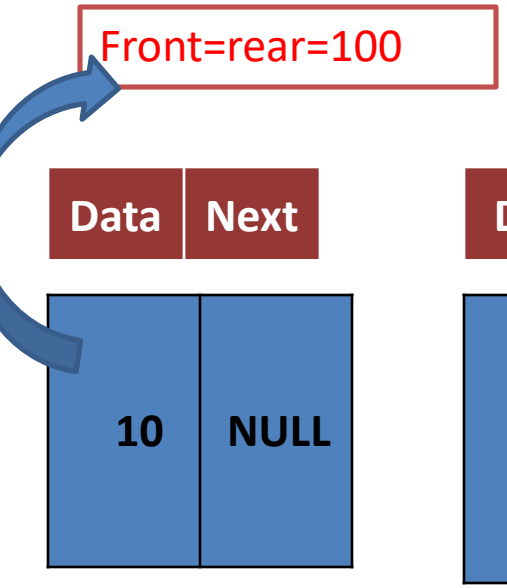
```
100->data=10;
100->next=NULL;
```

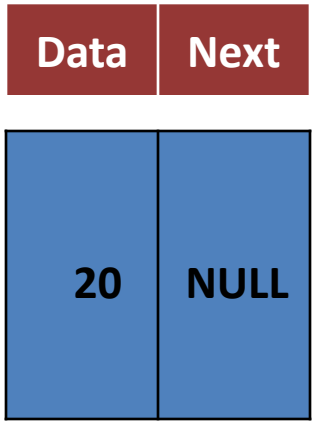
```
If(front==NULL && rear==NULL)
{
front=rear=node1;
}
```

(first node of list is called “Front” and last node is called ”rear ” in QSLL.

After creating first node Head=first node address
i.e **front=rear=100**

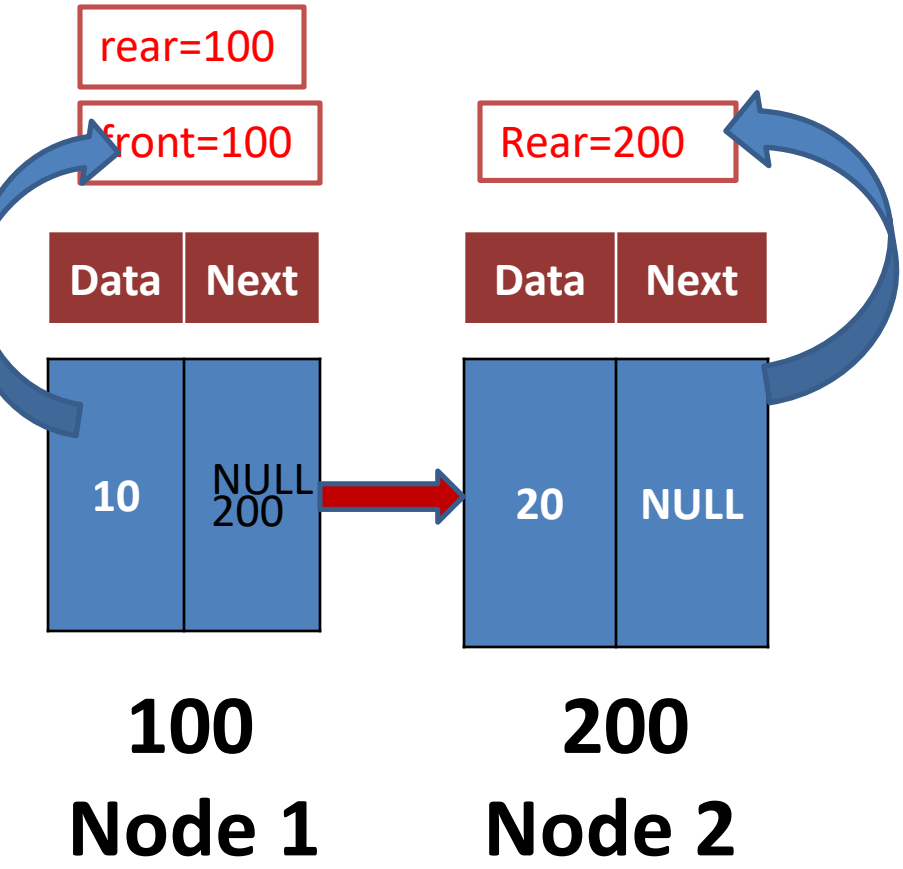






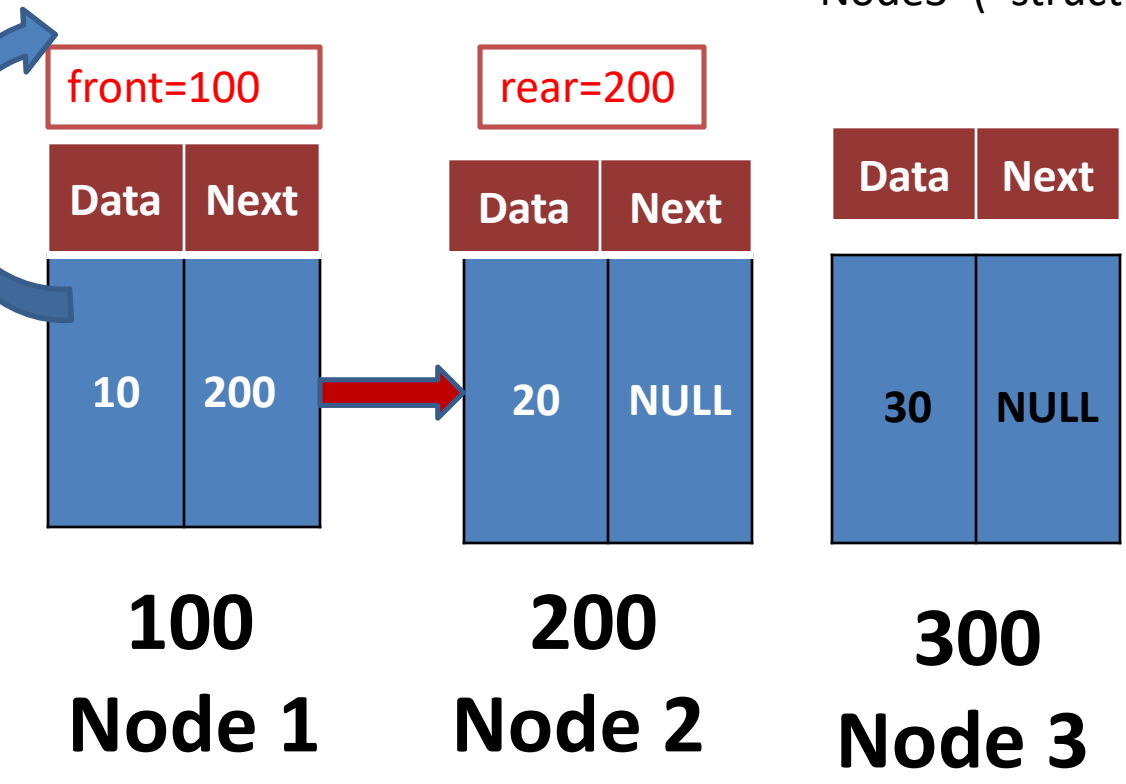

```
Node2=(*struct Node)malloc(sizeof(*struct Node);

Node2->data=20;
Node2->next=NULL;
```

```
If(front==NULL && rear==NULL)
{
front=rear=node2;
}
```



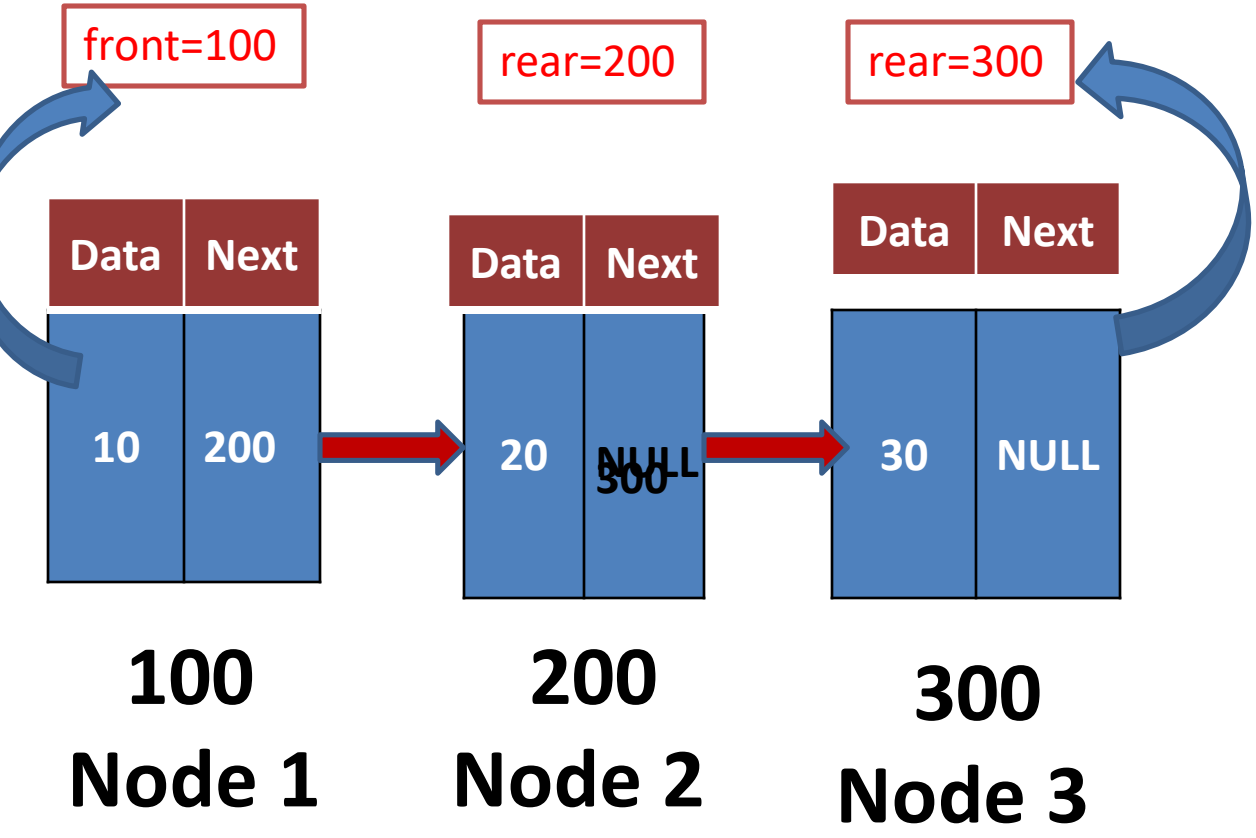
rear->next=node2;
Rear=newNode;

Node3=(*struct Node)malloc(sizeof(*struct Node);

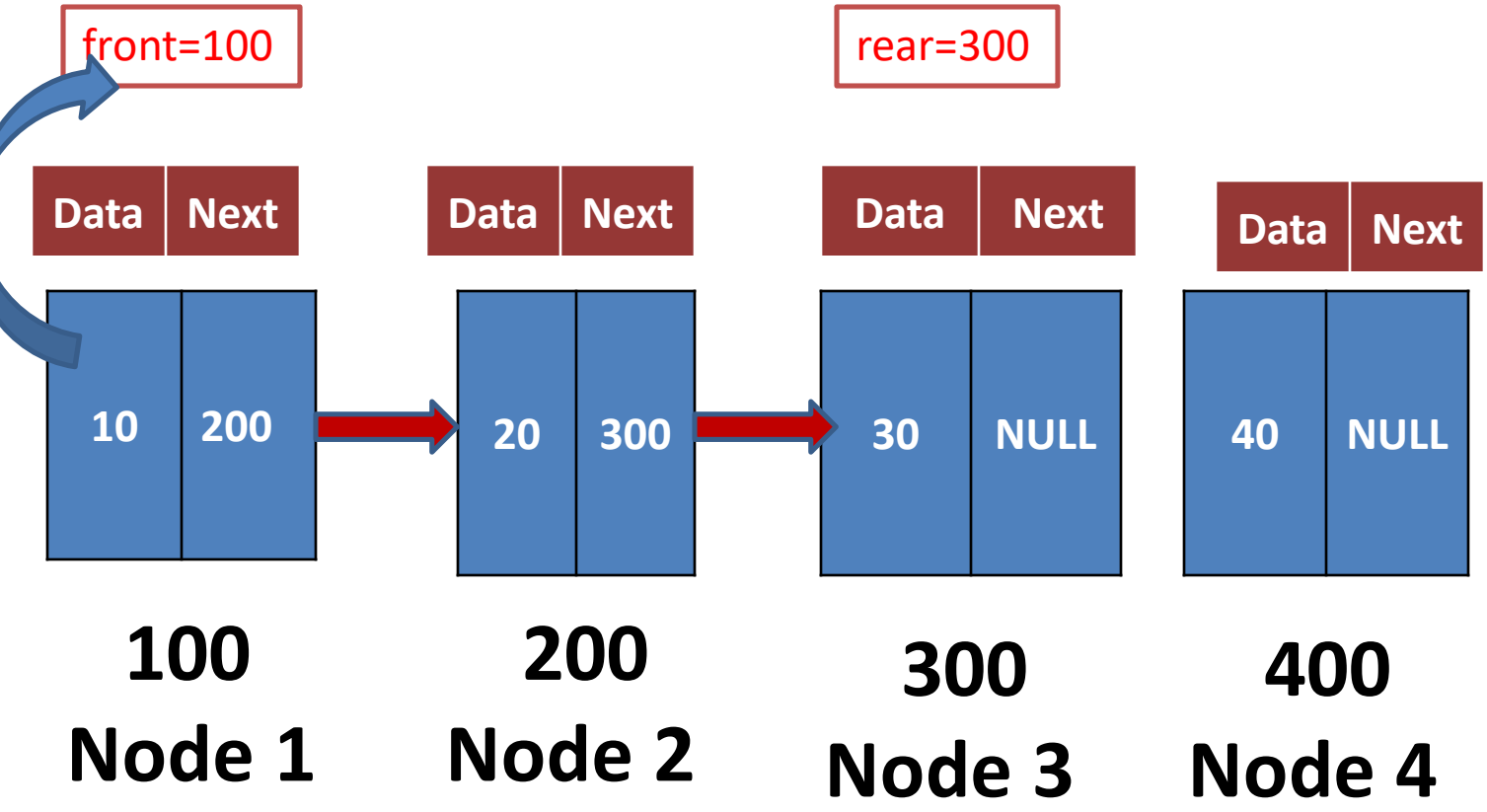


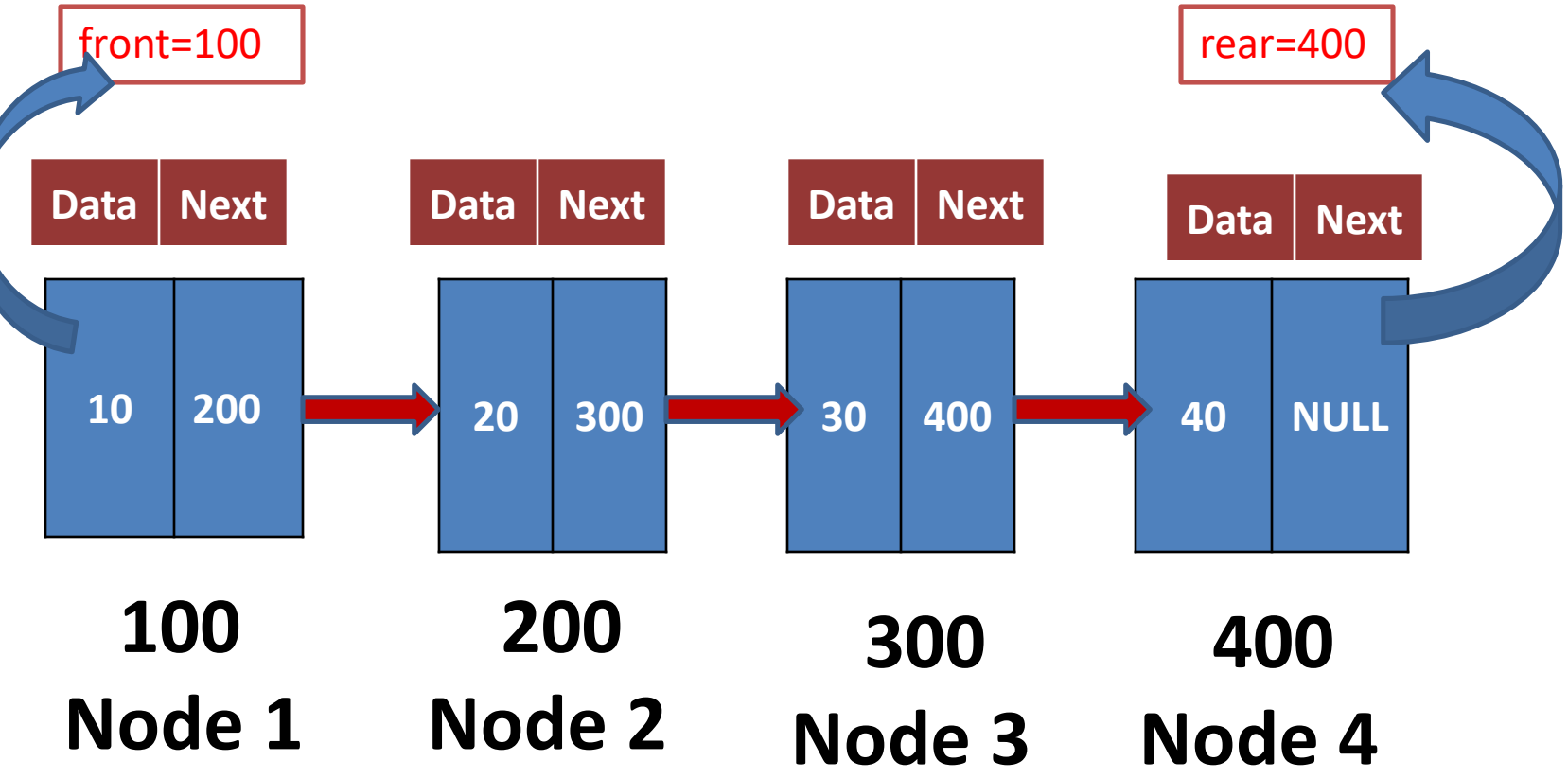

```
Node3->data=30;
Node3->next=NULL;
```

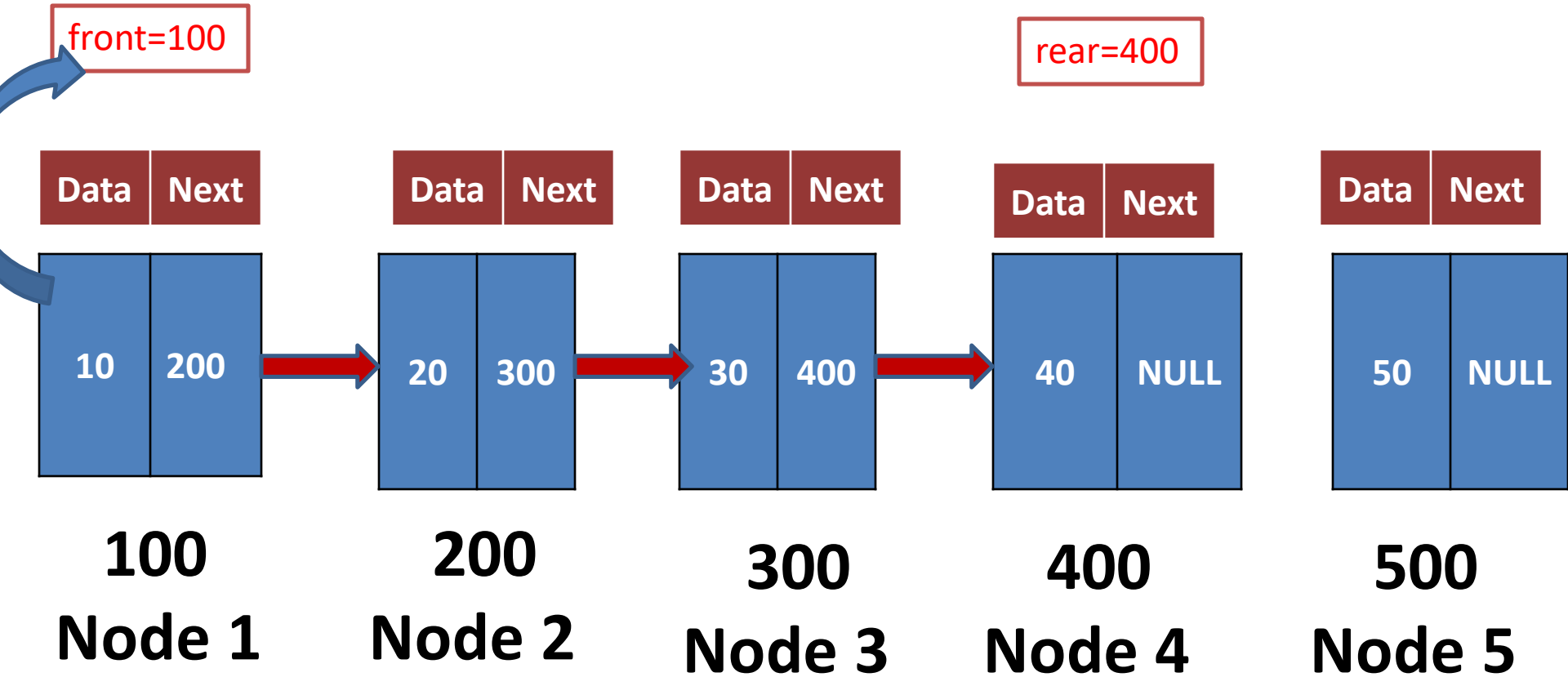
```
If(front==NULL && rear==NULL)
{
front=rear=node3;
}
```

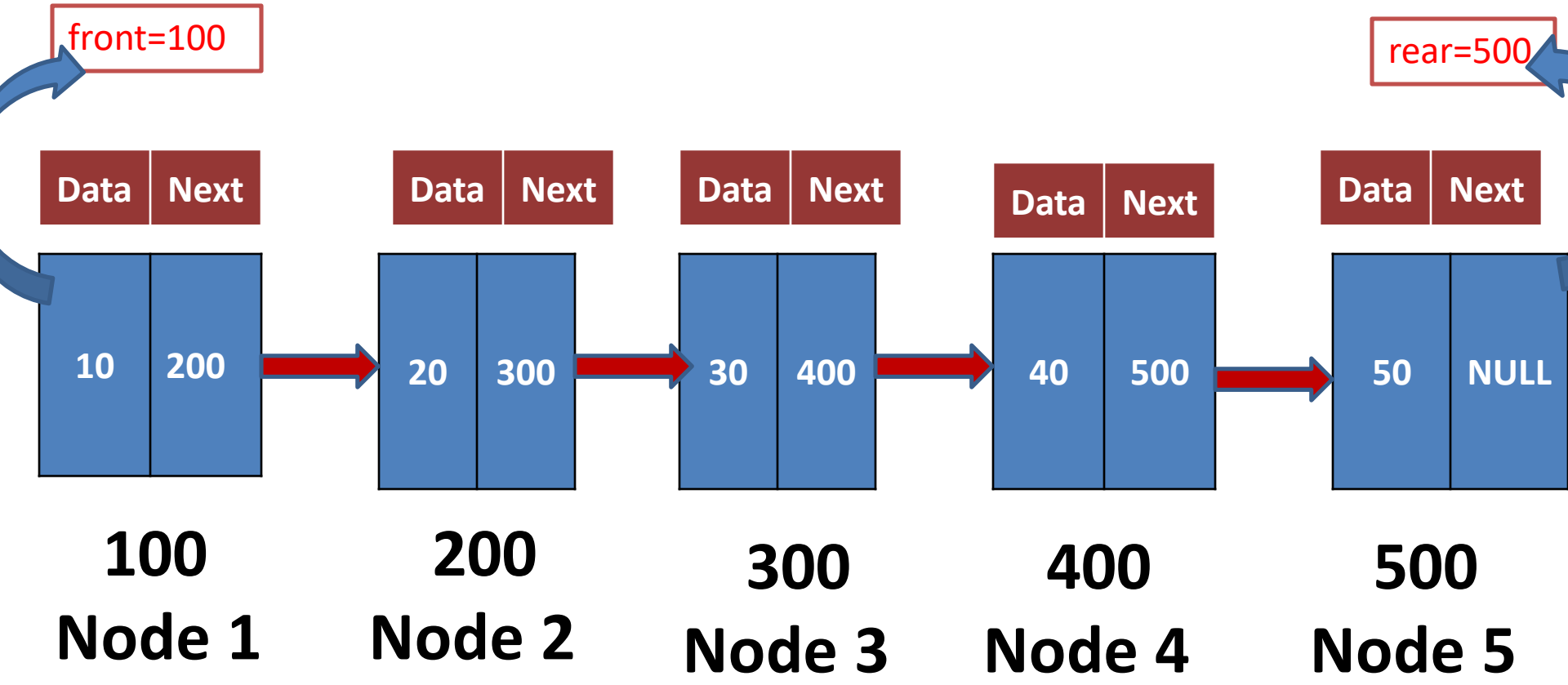


rear->next=node3;
Rear=newNode;


front=100
rear=300
Data
Next
10
200
20
300
30
NULL
40
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4


front=100
rear=400
Data
Next
10
200
20
300
30
400
40
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4


front=100
rear=400
Data
Next
10
200
20
300
30
400
40
NULL
50
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4
500
Node 5


front=100
rear=500
Data
Next
10
200
20
300
30
400
40
500
50
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4
500
Node 5

Algorithm for Dequeue()

dequeue(): Deleting an Element from Queue

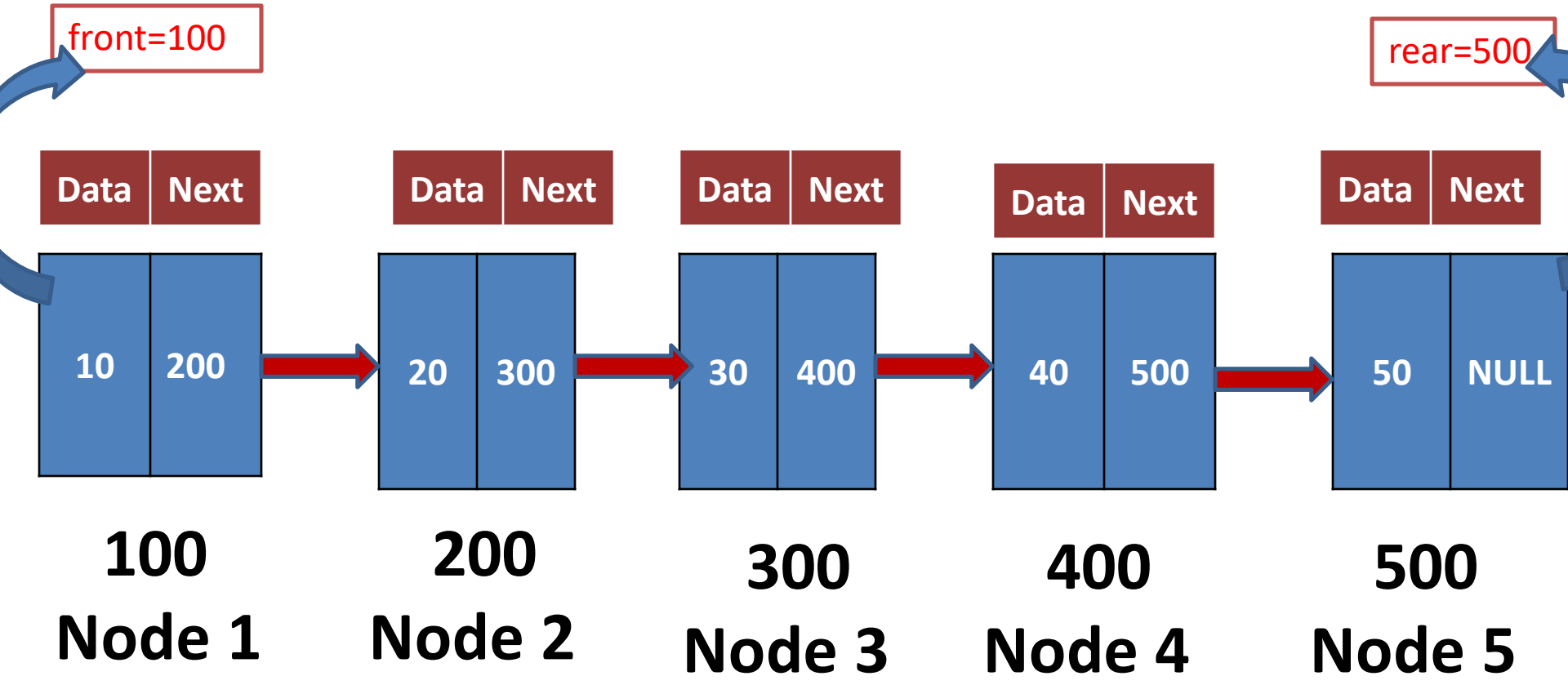
Step 1: Check whether **queue** is **Empty**

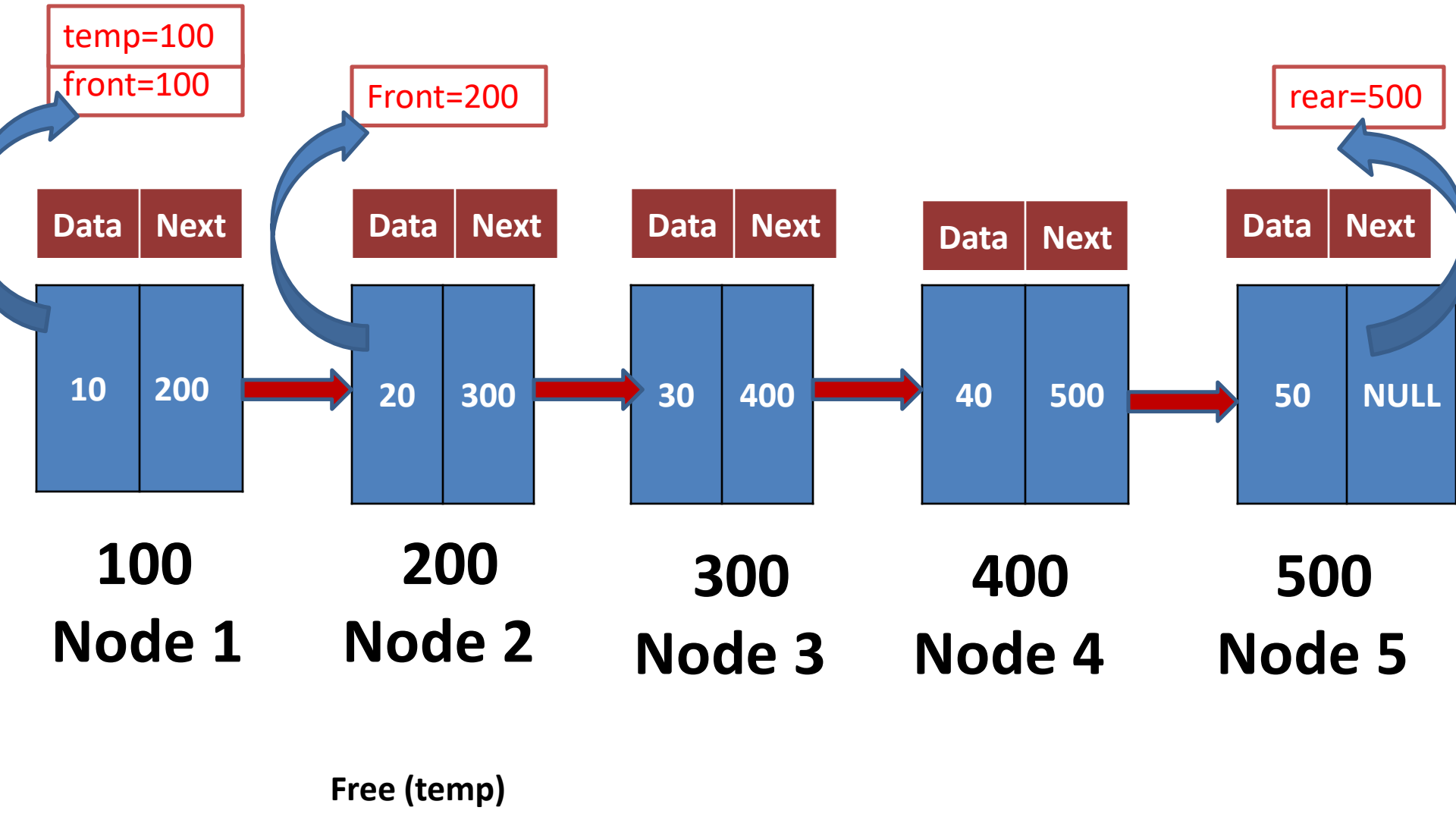
(**front == NULL && rear==NULL**).

Step 2 : If it is **Empty**, then display "**Queue is Empty, Deletion is not possible**" and end from the function

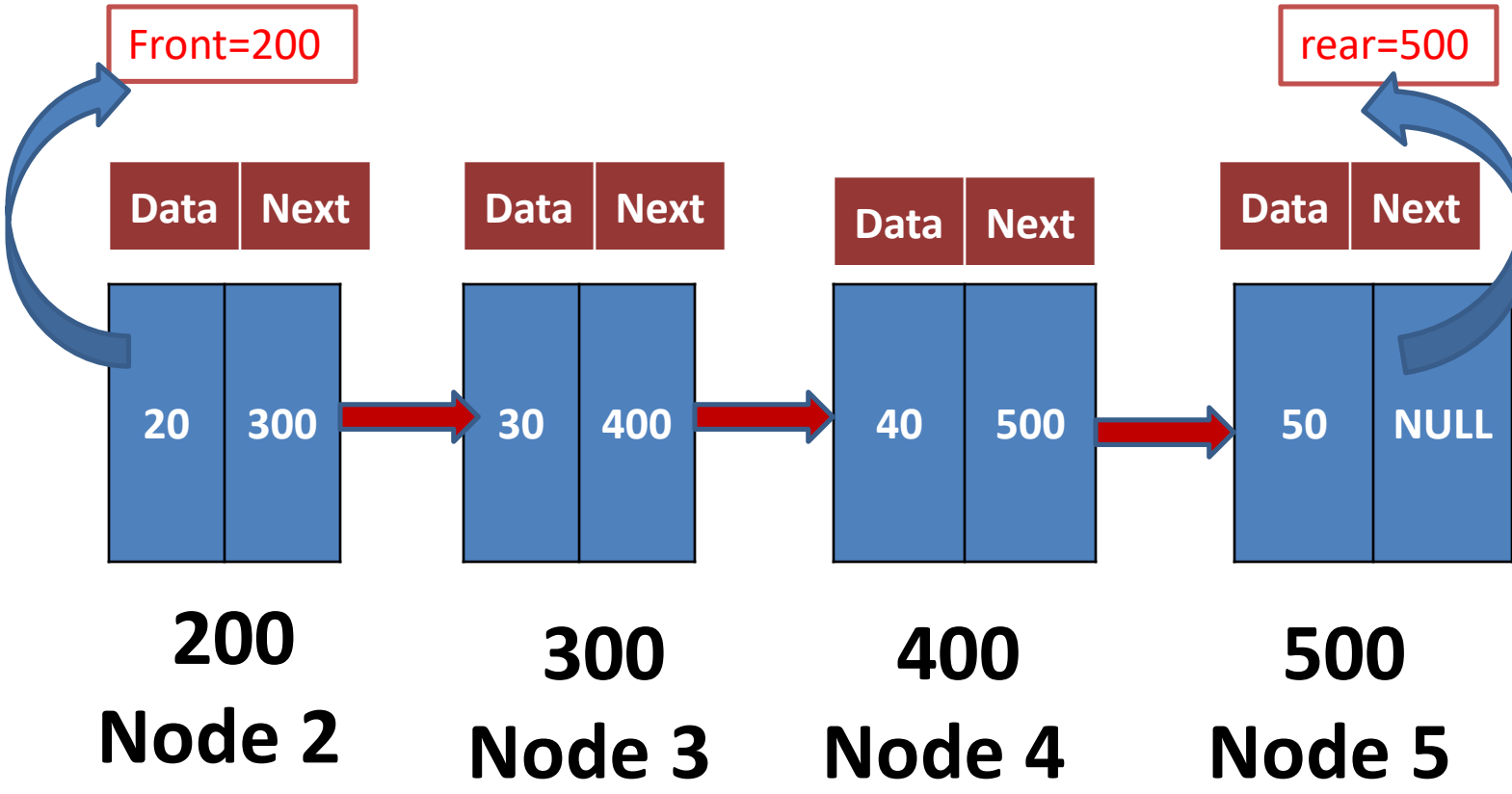
Step 3 : If it is **Not Empty** then, define a Node pointer '**temp**' and set it to '**front**'.

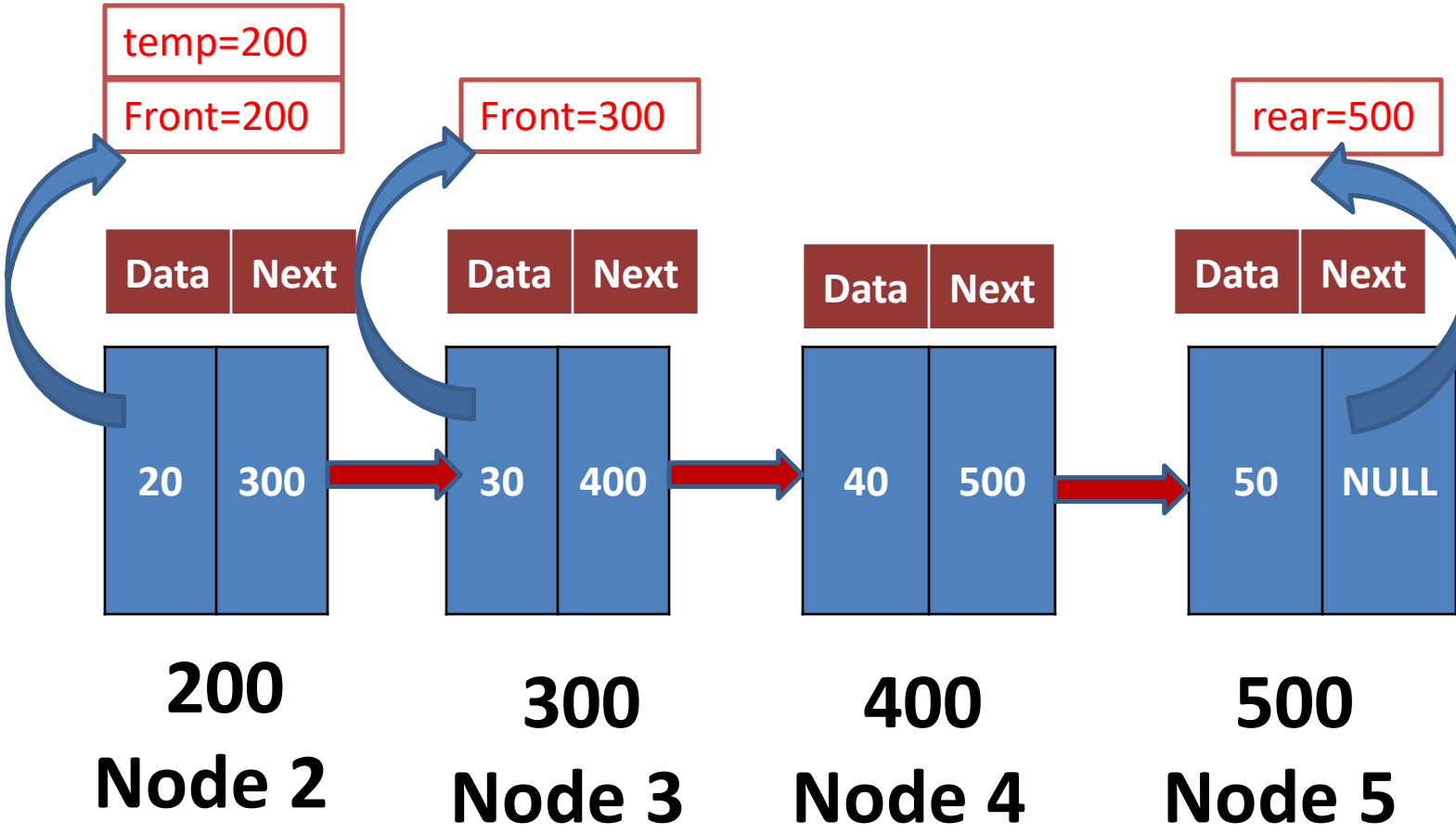
Step 4 : Then set '**front = front → next**' and delete '**temp**' (**free(temp)**).


front=100
rear=500
Data
Next
10
200
20
300
30
400
40
500
50
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4
500
Node 5

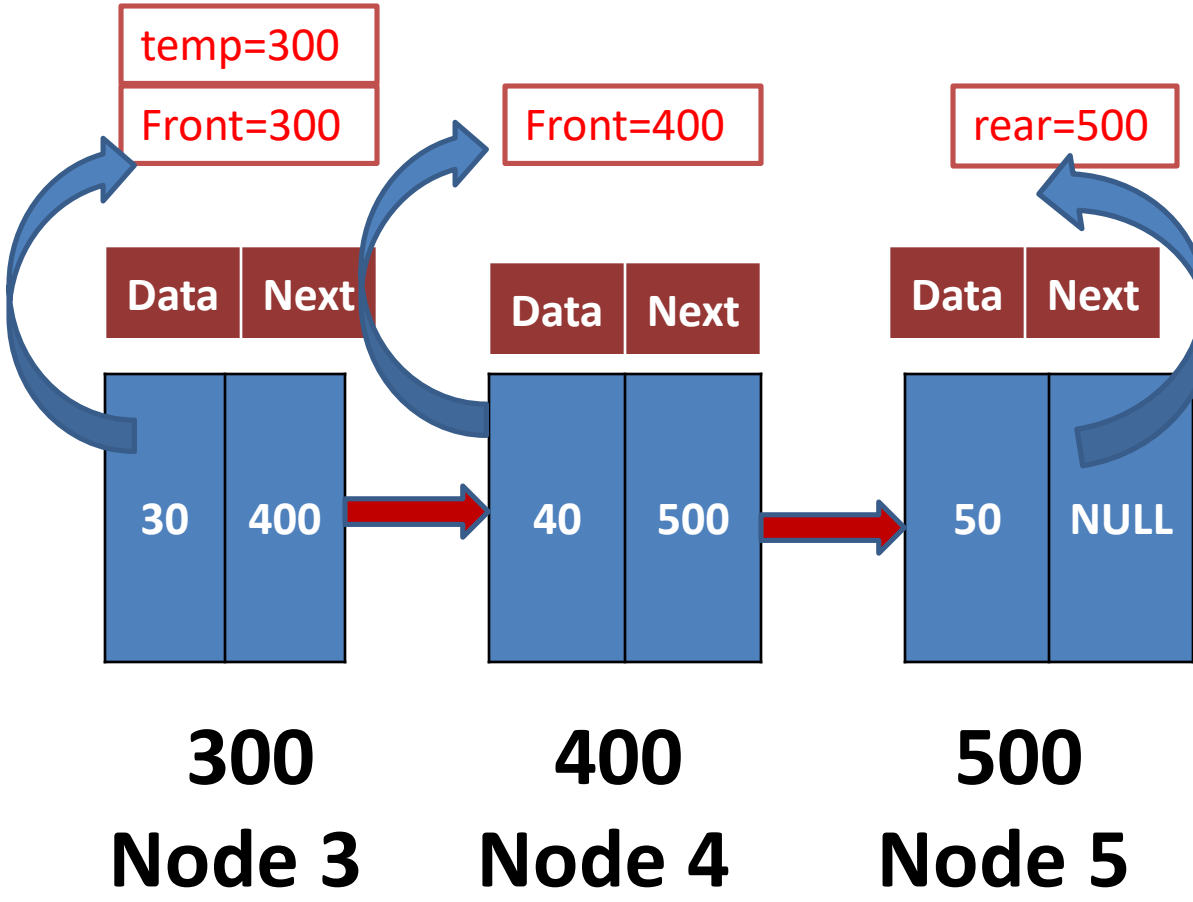

temp=100
front=100
Front=200
rear=500
Data
Next
10
200
20
300
30
400
40
500
50
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4
500
Node 5

Free (temp)

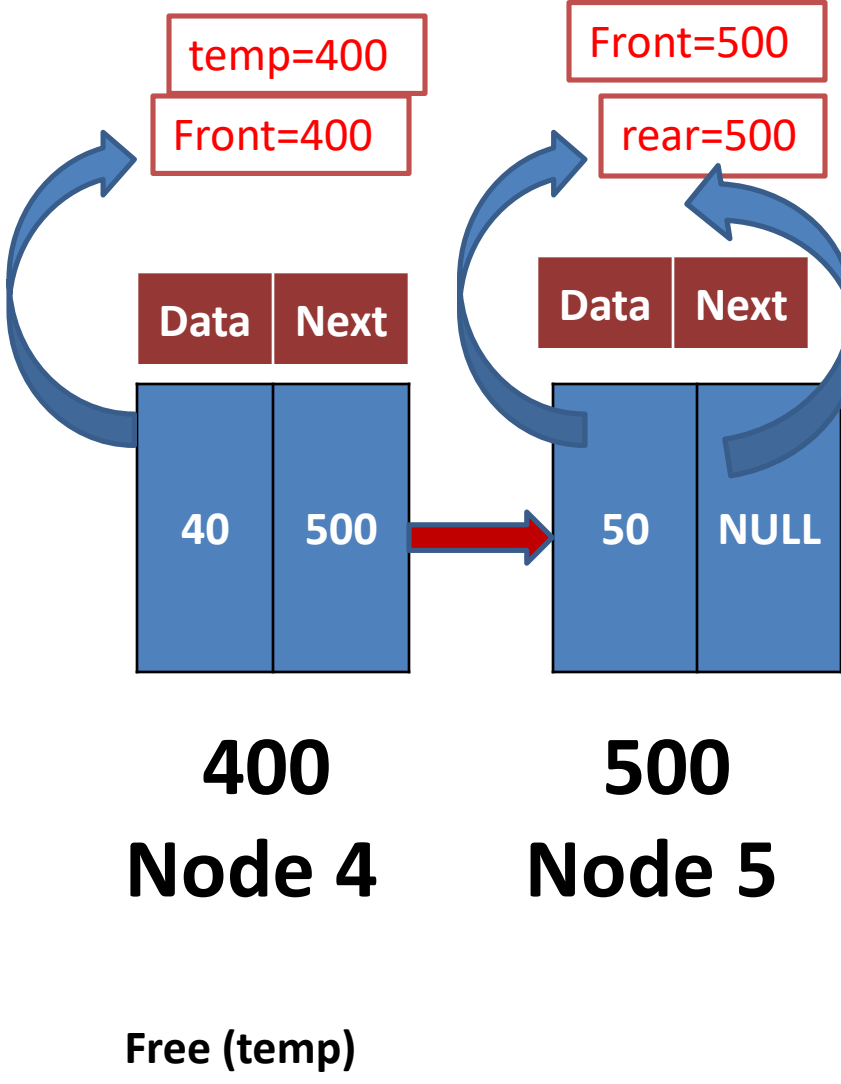

Front=200
rear=500
Data
Next
20
300
30
400
40
500
50
NULL
200
Node 2
300
Node 3
400
Node 4
500
Node 5


temp=200
Front=200
Front=300
rear=500
Data
Next
Data
Next
Data
Next
Data
Next
20
300
30
400
40
500
50
NULL
200
Node 2
300
Node 3
400
Node 4
500
Node 5

Free (temp)

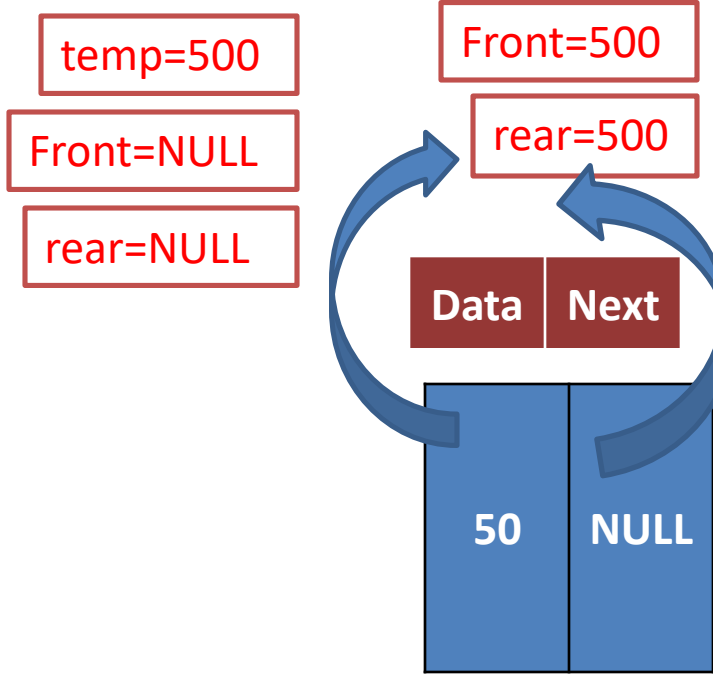

temp=300
Front=300
Front=400
rear=500
Data
Next
Data
Next
Data
Next
30
400
40
500
50
NULL
300
Node 3
400
Node 4
500
Node 5

Free (temp)


temp=400
Front=400
Front=500
rear=500
Data
Next
Data
Next
40
500
50
NULL
400
Node 4
500
Node 5
Free (temp)

front == NULL && rear==NULL).

Queue is Empty, Deletion is not possible



Free (temp)

Algorithm for Display()

display(): Display an Element from Queue

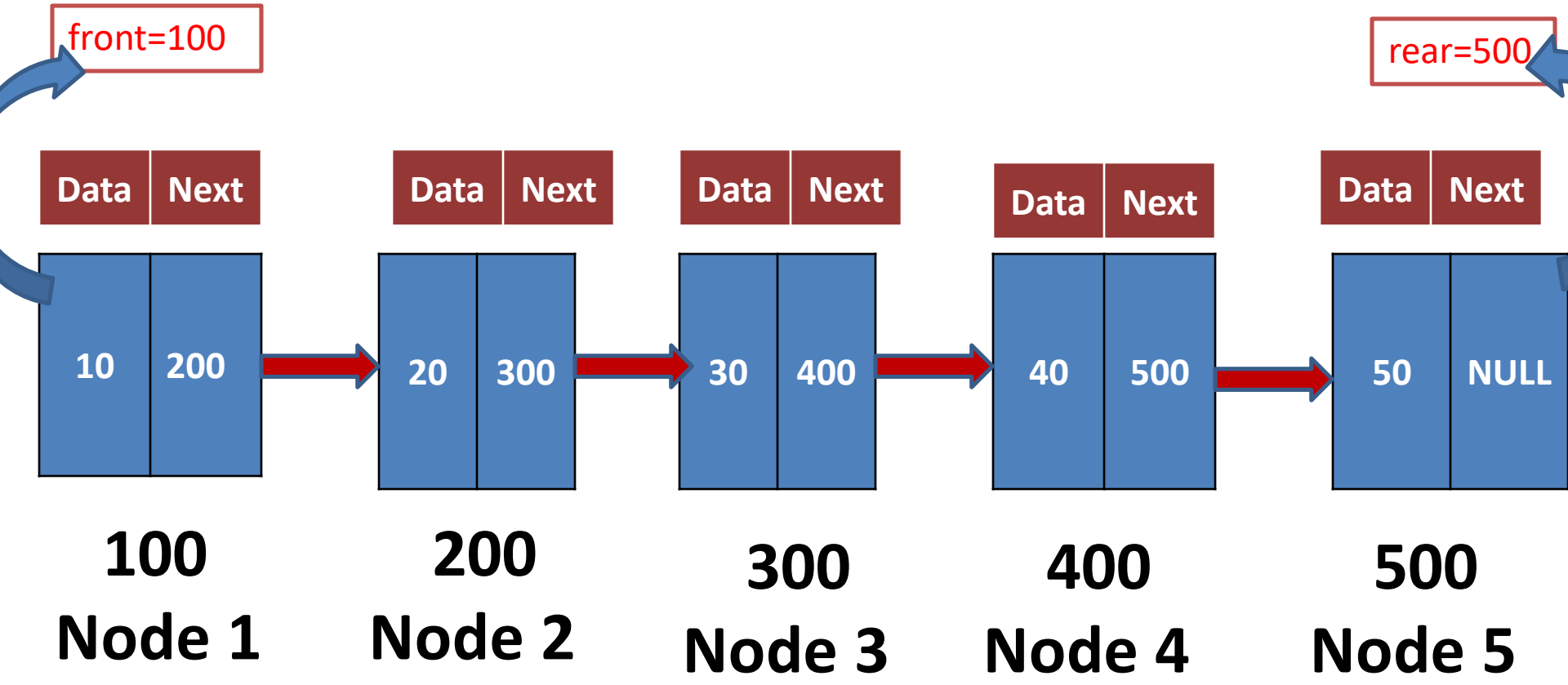
Step 1: Check whether **queue** is **Empty**
(**front == NULL && rear==NULL**).

Step 2 : If it is **Empty**, then display "**Queue is Empty, Display is not possible**" and end from the function

Step 3 : If it is **Not Empty** then, define a Node pointer '**temp**' and initialize with **front**.

Step 4 : Display '**temp → data**' and move it to the next node. Repeat the same until '**temp**' reaches to '**rear**' **(temp → next != NULL).**

Step 5 : Finally! Display '**temp → data ---> NULL**'.


front=100
rear=500
Data
Next
10
200
20
300
30
400
40
500
50
NULL
100
Node 1
200
Node 2
300
Node 3
400
Node 4
500
Node 5

Thank You