Data Structures – CST 201
Module - 3

Syllabus

- **Linked List and Memory Management**
  - Self Referential Structures
  - Dynamic Memory Allocation
  - Singly Linked List-Operations on Linked List.
  - Doubly Linked List
  - Circular Linked List
  - Stacks using Linked List
  - **Queues using Linked List**
  - Polynomial representation using Linked List
  - Memory allocation and de-allocation
    - First-fit, Best-fit and Worst-fit allocation schemes

Queue Data Structure

- **Definition:** A queue is a linear list of elements in which insertion can take place only at one end, called the **rear**, and deletion can take place only at the other end, called the **front**.
- Queue is a **First-in-First-Out(FIFO)** data structure. Items are removed in the same order as they were inserted. It is also called FIFO list.
- **Basic operations on queue**
  - **ENQUEUE:** Insert an item at the rear end of queue
  - **DEQUEUE:** Delete an item from the front end of queue
- **Queue Representations:**
  - Array Representation
  - Linked List Representation

Queue Using Linked List

- Data structure used is **Singly Linked List**

- **ENQUEUE**: Insert an item at the end of the list
- **DEQUEUE**: Delete an item from the beginning of the list

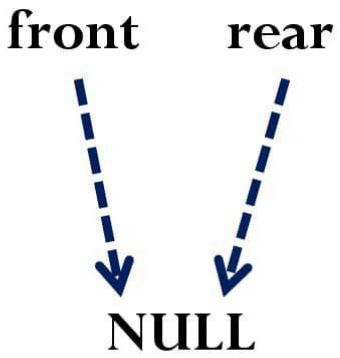
ENQUEUE

2 cases:

1. Queue is empty
2. Queue is not empty

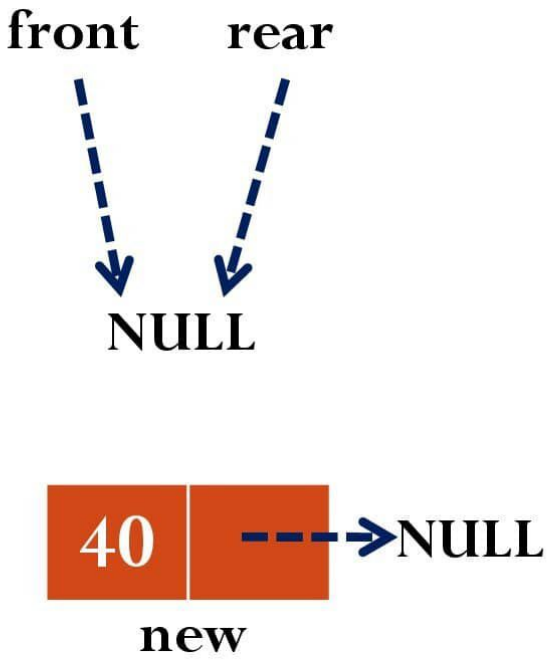
ENQUEUE

Case 1



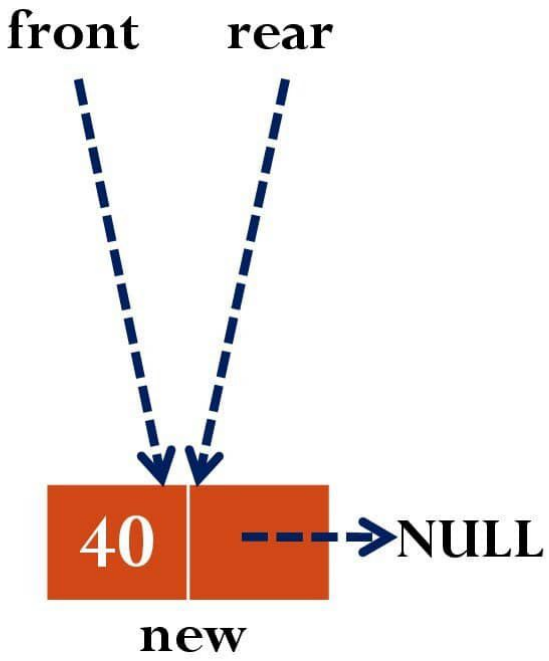
ENQUEUE

Case 1



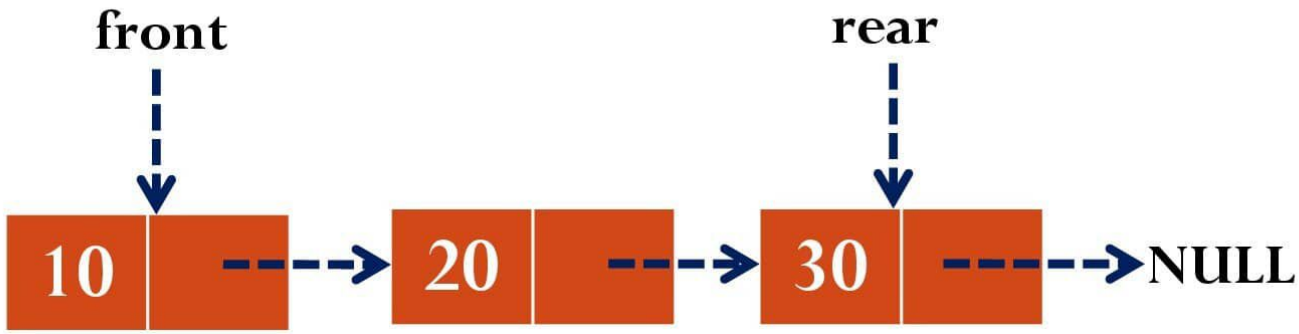
ENQUEUE

Case 1



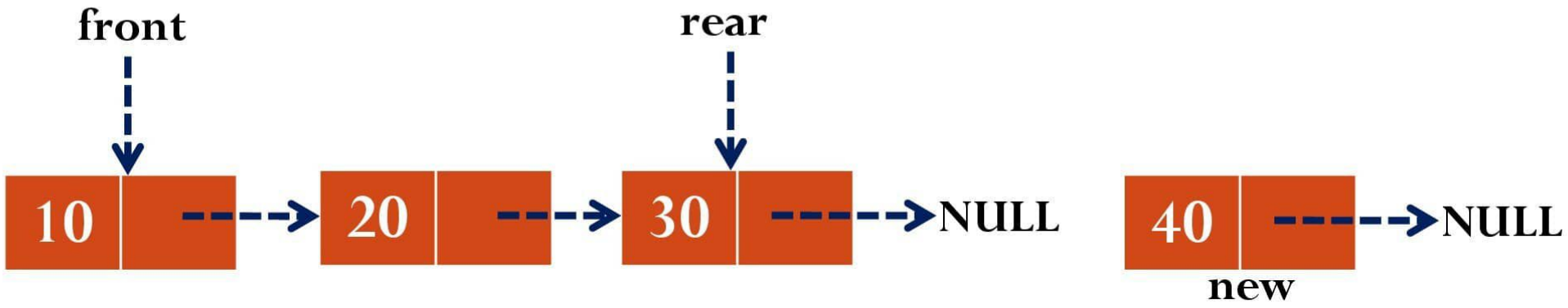
ENQUEUE

Case 2



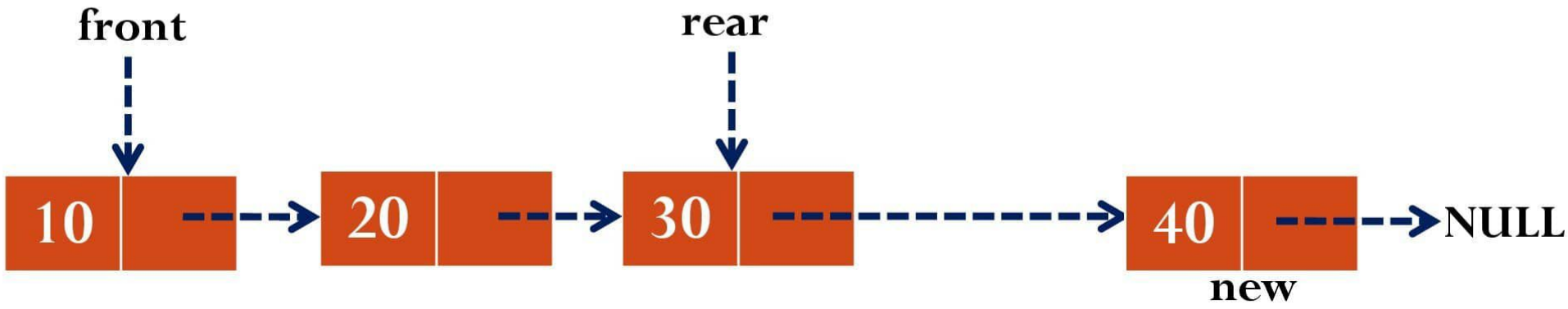
ENQUEUE

Case 2



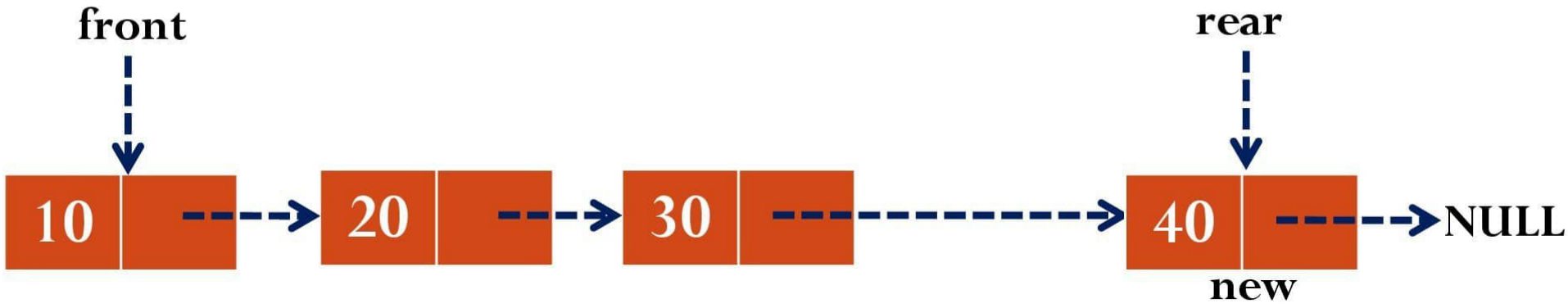
ENQUEUE

Case 2



ENQUEUE

Case 2



ENQUEUE - Algorithm

Algorithm ENQUEUE(front, rear,item)

1. Create a node new
2. new→data=item
3. new→link=NULL
4. If front=NULL then
    1. front=rear=new
5. Else
    1. rear→link=new
    2. rear=new

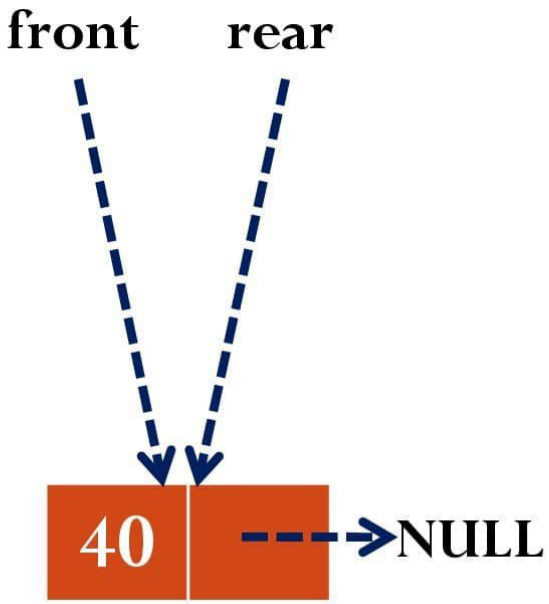
DEQUEUE

2 cases:

1. Queue is empty
2. Queue contains only one node
3. Queue is contains more than one node

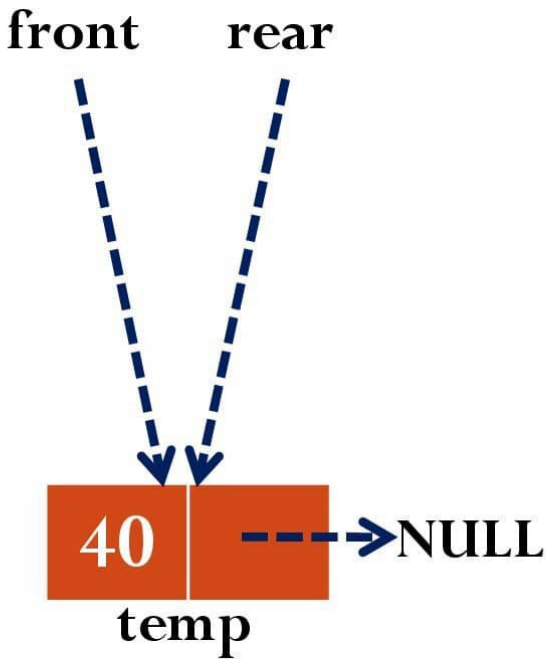
DEQUEUE

Case 2



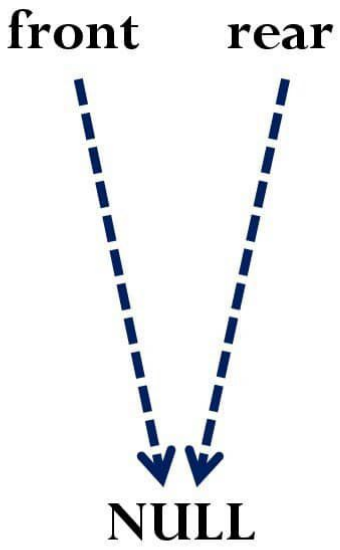
DEQUEUE

Case 2



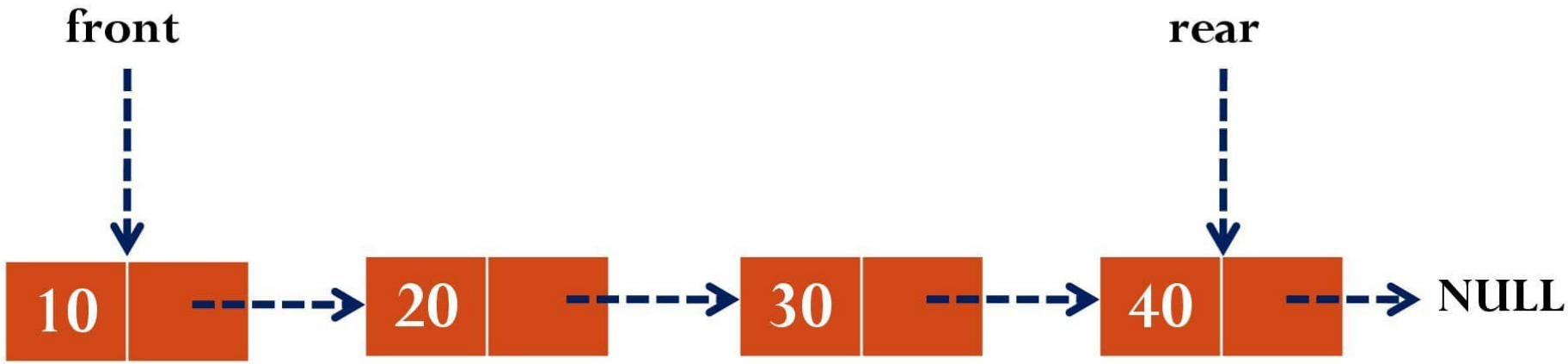
DEQUEUE

Case 2



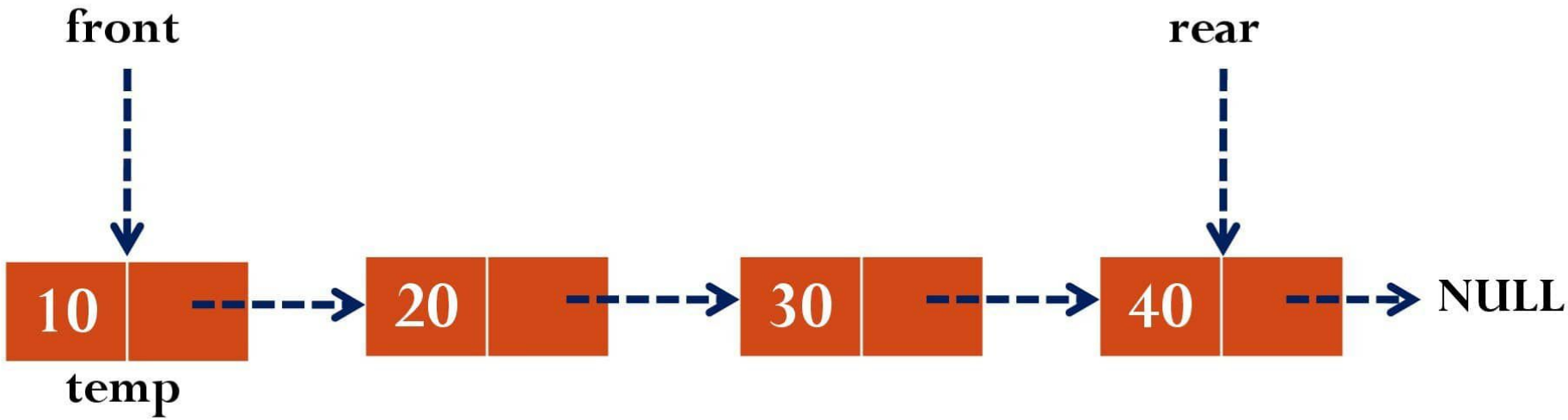
DEQUEUE

Case 3



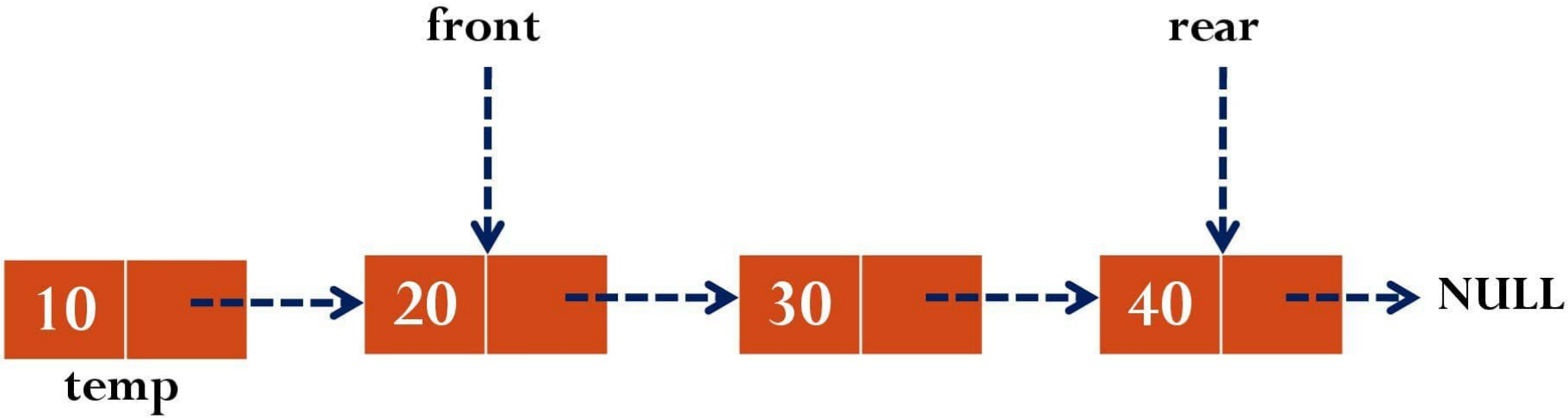
DEQUEUE

Case 3



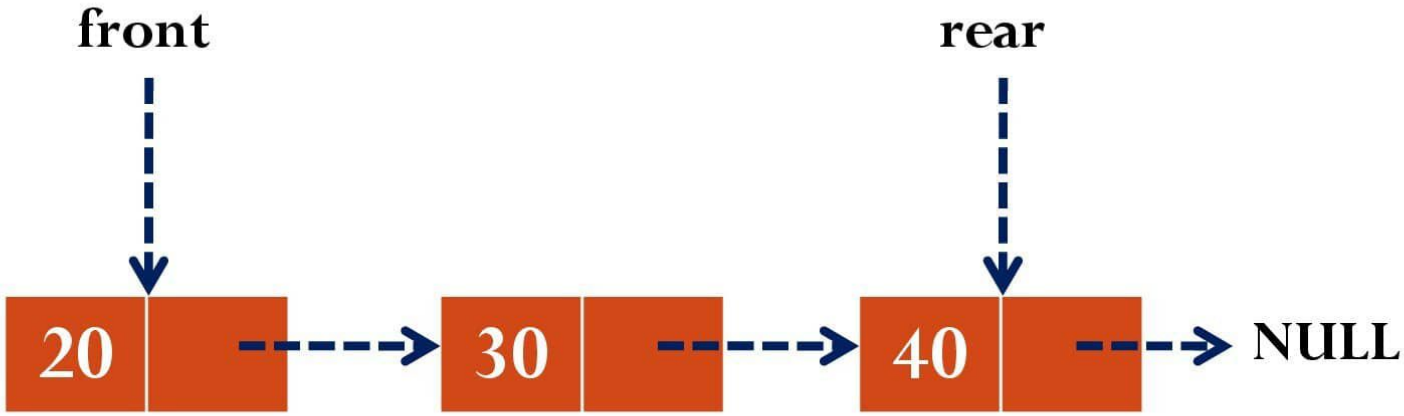
DEQUEUE

Case 3



DEQUEUE

Case 3



DEQUEUE - Algorithm

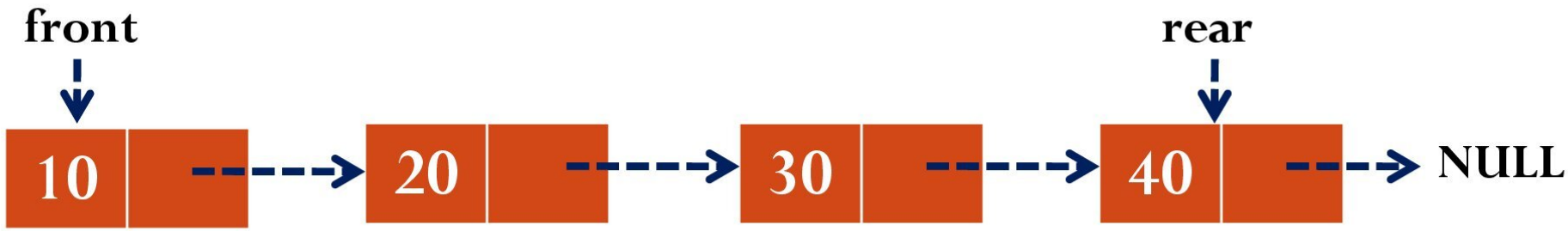
Algorithm DEQUEUE(front, rear)

1. If front=NULL then
    1. Print "Queue is empty."
2. Else if front=rear then
    1. temp=front
    2. front=rear=NULL
    3. Dispose(temp)
3. Else
    1. temp=front
    2. front=front→link
    3. Dispose(temp)

Display - Algorithm

Algorithm Display(front, rear)

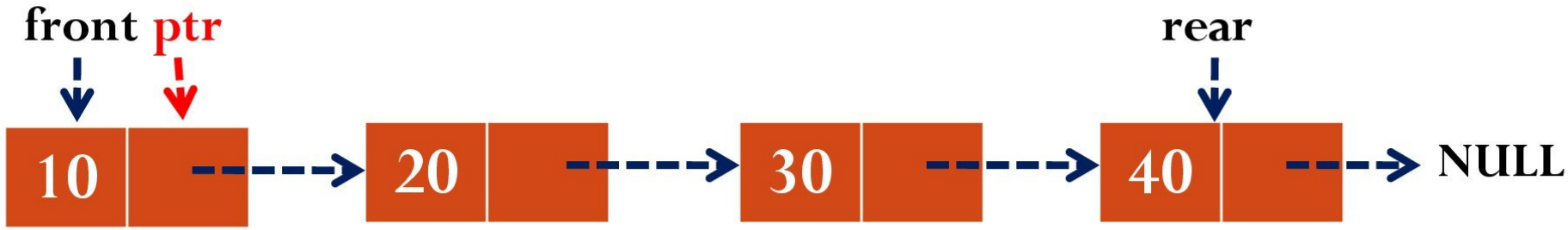
1. If front=NULL then
    1. Print "Queue is Empty"
2. Else
    1. ptr=front
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display - Algorithm

Algorithm Display(front, rear)

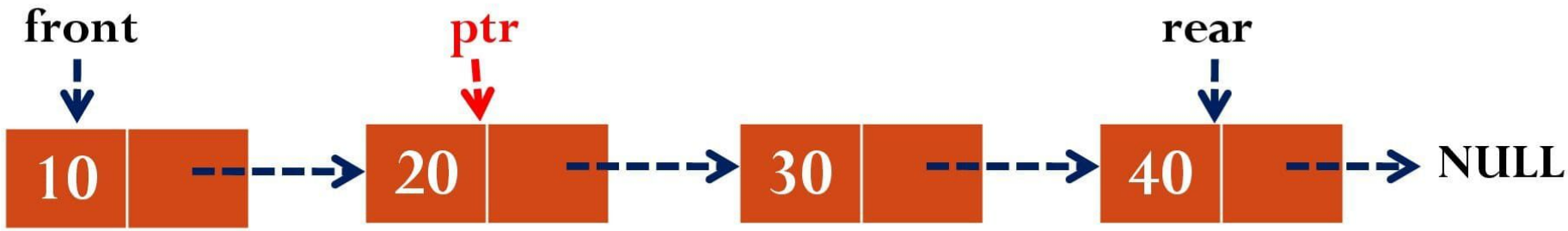
1. If front=NULL then
    1. Print "Queue is Empty"
2. Else
    1. ptr=front
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display - Algorithm

Algorithm Display(front, rear)

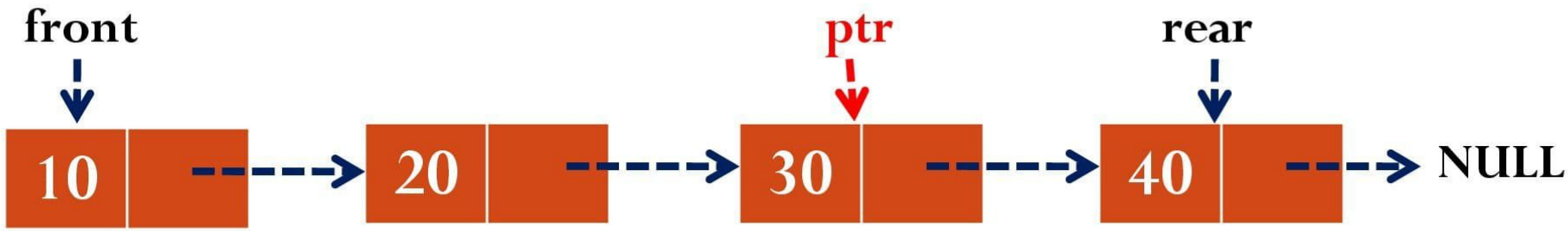
1. If front=NULL then
    1. Print "Queue is Empty"
2. Else
    1. ptr=front
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display - Algorithm

Algorithm Display(front, rear)

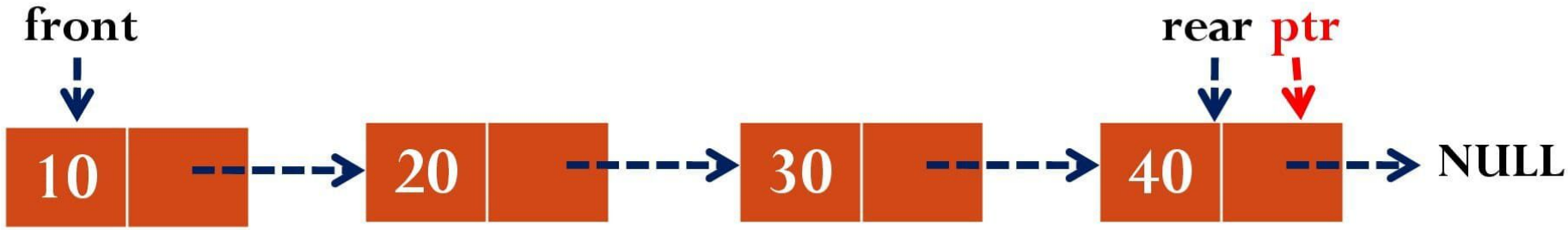
1. If front=NULL then
    1. Print "Queue is Empty"
2. Else
    1. ptr=front
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display - Algorithm

Algorithm Display(front, rear)

1. If front=NULL then
    1. Print "Queue is Empty"
2. Else
    1. ptr=front
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display - Algorithm

Algorithm Display(front, rear)

1. If front=NULL then
    1. Print "Queue is Empty"
2. Else
    1. ptr=front
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link