Module 5

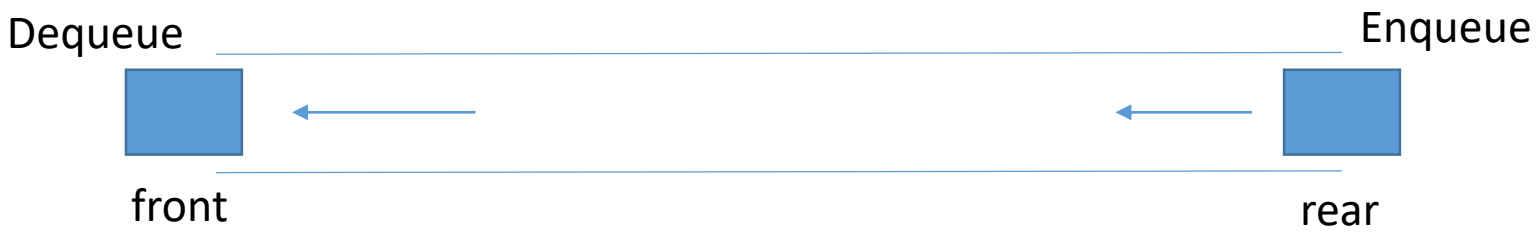
Queue

Queues

- Queue is another linear data structure used for storing data
- Items are inserted and deleted as per the FIFO (first-in-first-out) principle
- The first element inserted in the queue is the first element to be deleted
- Unlike Stack, Queue is open from both the ends
  - Insertion takes place from one end called **front**
  - Deletion takes place from another end called **rear**
- Applications
  - Queue formed for bus/ticket/elevator
  - CPU scheduling
  - Printer scheduling

Queue Operations

- Fundamental ADT operations:
  - Enqueue(x): Inserts an element x at the rear of the queue
  - Dequeue(): Deletes an element from the front of the queue



Queue Implementation using Array

- Use an array to store a queue
- Two indexes are marked as ***front*** and ***rear***
- Queue is empty when *front == rear ==* 0
- A queue is full when *rear == MAX-SIZE*

Working Example: Enqueue Operation

ENQUEUE(A, x)
if *rear == MAX-SIZE*
    Error OVERFLOW
else if *rear == 0*
    front = *rear* = 1
else
    rear = *rear* + 1
A[*rear*] = *x*

Working Example: Dequeue Operation

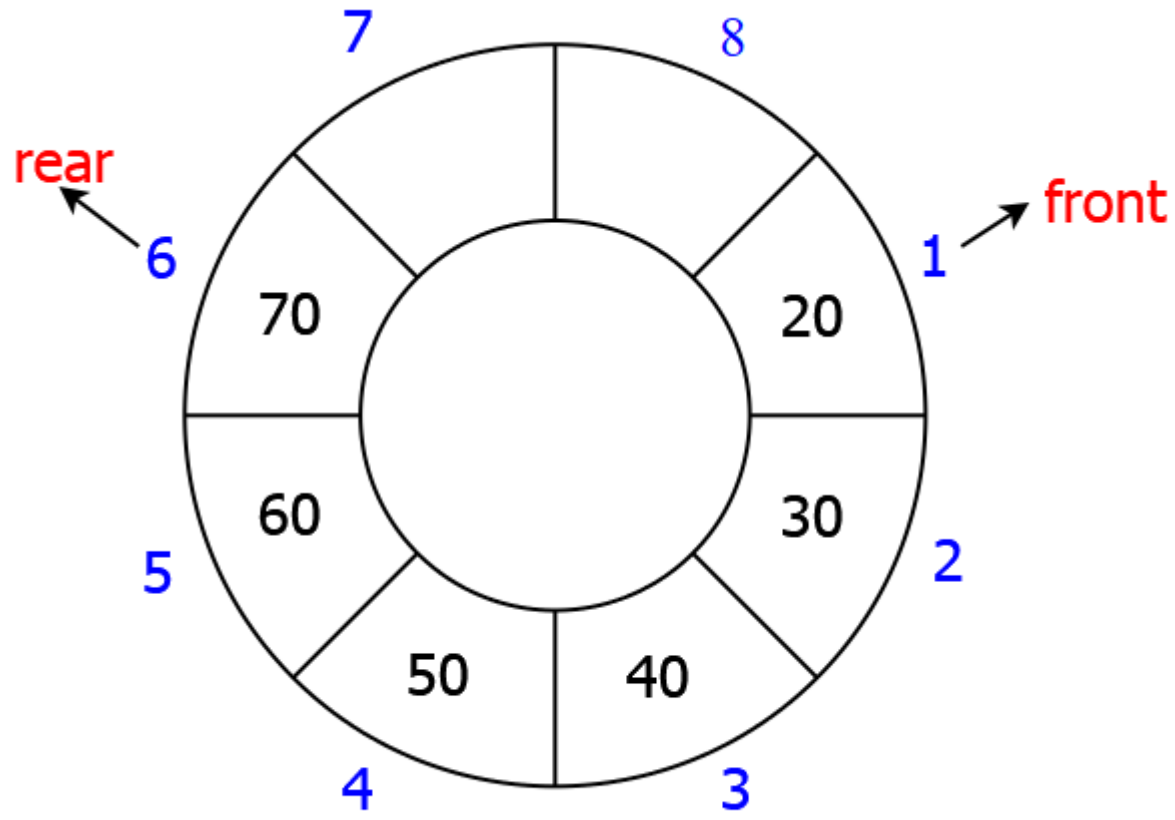
DEQUEUE(*A, k*)
if *front == 0*
    Error UNDERFLOW
k = A[front]
if *front == rear*
    front = rear = 0
else
    front = front + 1

Limitations

- Once we *dequeue* an element from the array, the associated cell space is wasted
- The above space cannot be reused until the queue is empty (*front* = *rear* =0)
- **Possible way to solve that?**
  - Using a circular array
  - If the current *rear* = *i* then next element to be inserted at *(i % N) +1*
    - *N* = size of the array (MAX-SIZE)

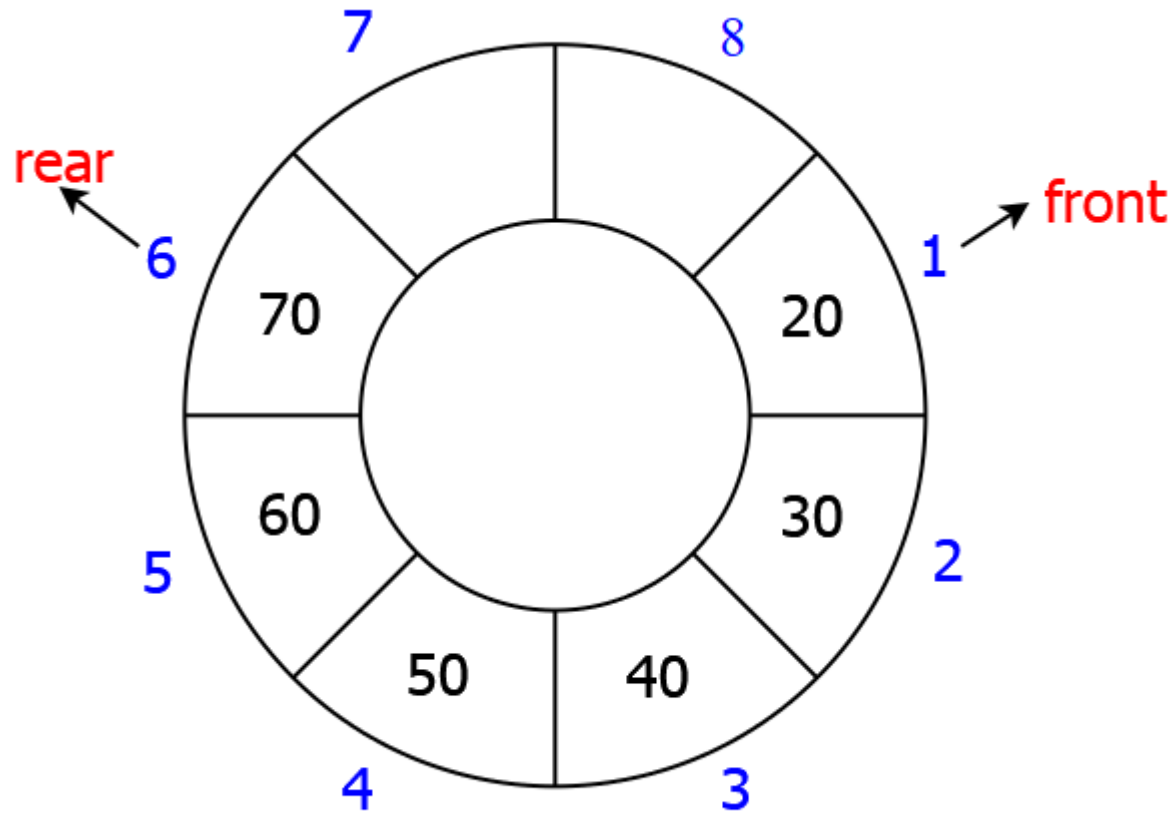
Queue Implementation using Circular Array

ENQUEUE(*A*, *x*)
if (*rear* % *N*) +1 == *front*
    Error OVERFLOW
else if *rear* == *0*
    front = *rear* = 1
else
    rear = (*rear* % N)+*1*
A[*rear*] = *x*



Queue Implementation using Circular Array

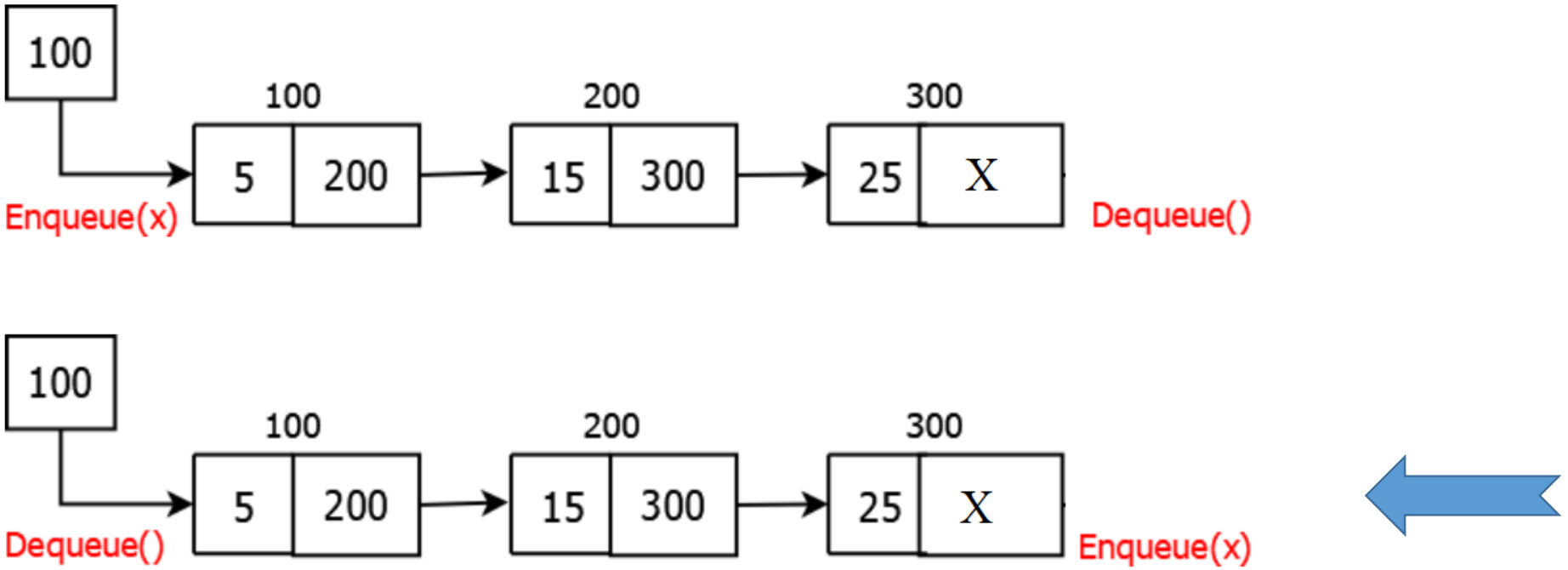
DEQUEUE(*A*)
if *front == 0*
    Error UNDERFLOW
k = A[front]
if *front == rear*
    front = rear = 0
else
    front = (*front* % N)+*1*



Queues using Linked Lists

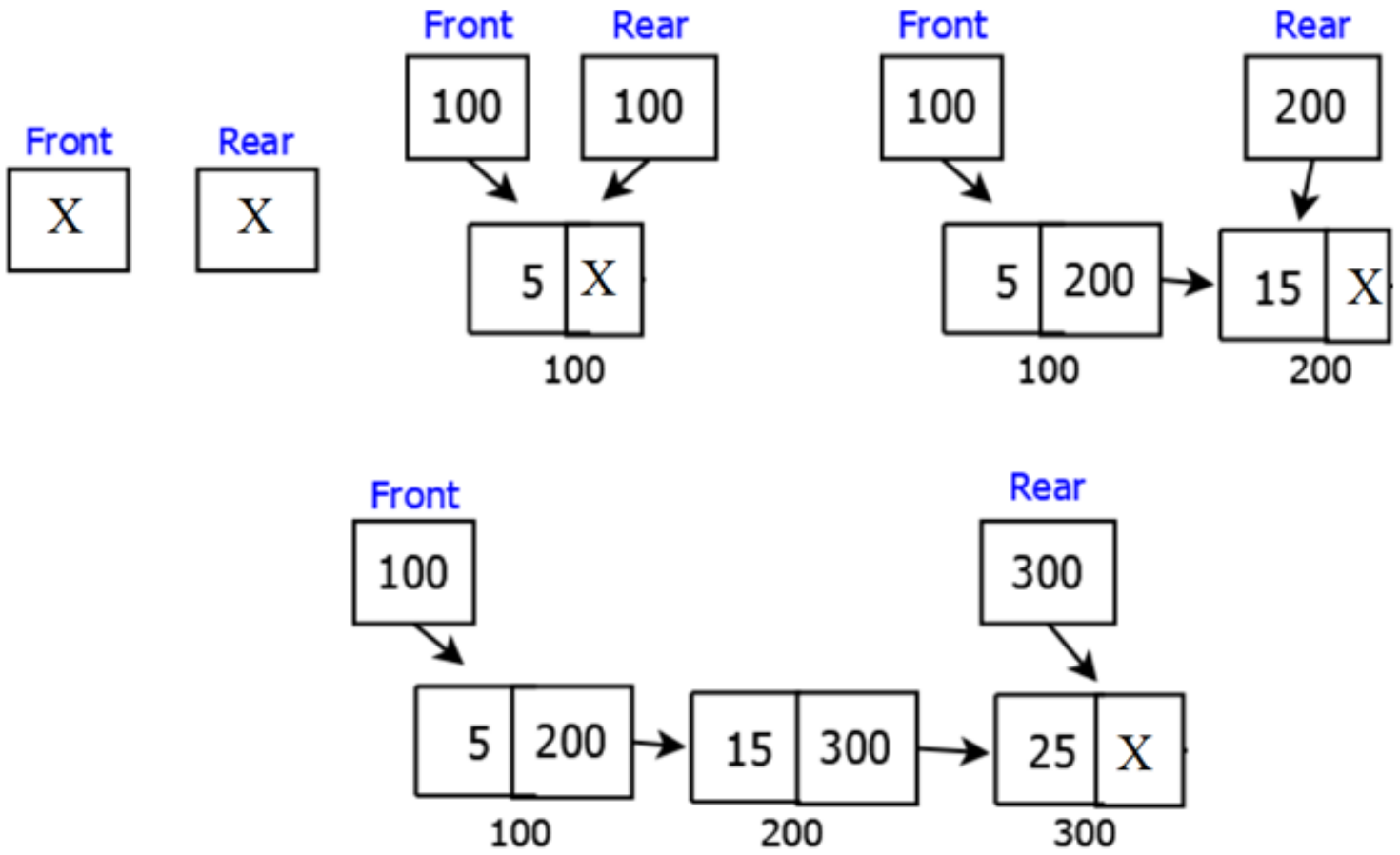
- An array can allow only *N* number of elements to be inserted in the queues at once
  - Likely to get exhausted even for circular arrays
- Possible Solution is to use **Dynamic Arrays**
  - Copying the elements from smaller arrays to larger array costs $O(n)$
- Another feasible solution is to implement the queues using Linked Lists

Queue Implementation using Linked Lists

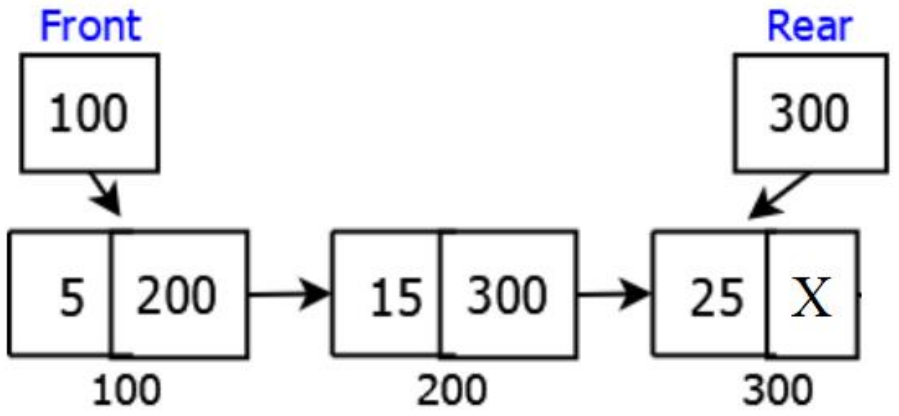


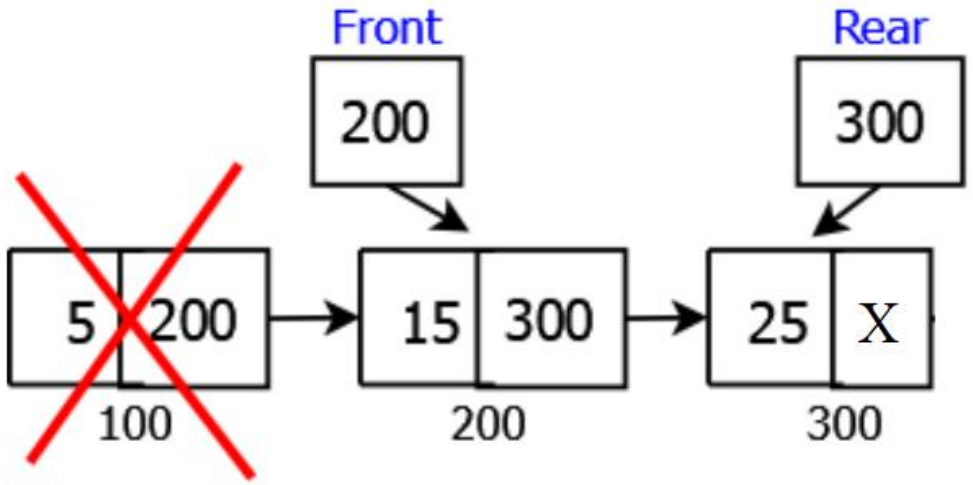
- **Challenge:** Insertion or deletion operation at the end of the queue takes $O(n)$ time
  - **Solution:** Maintain two pointers *front* and *rear*

Example: Enqueue Operation



Example: Dequeue Operation





Doubly Ended Queue (Deque)

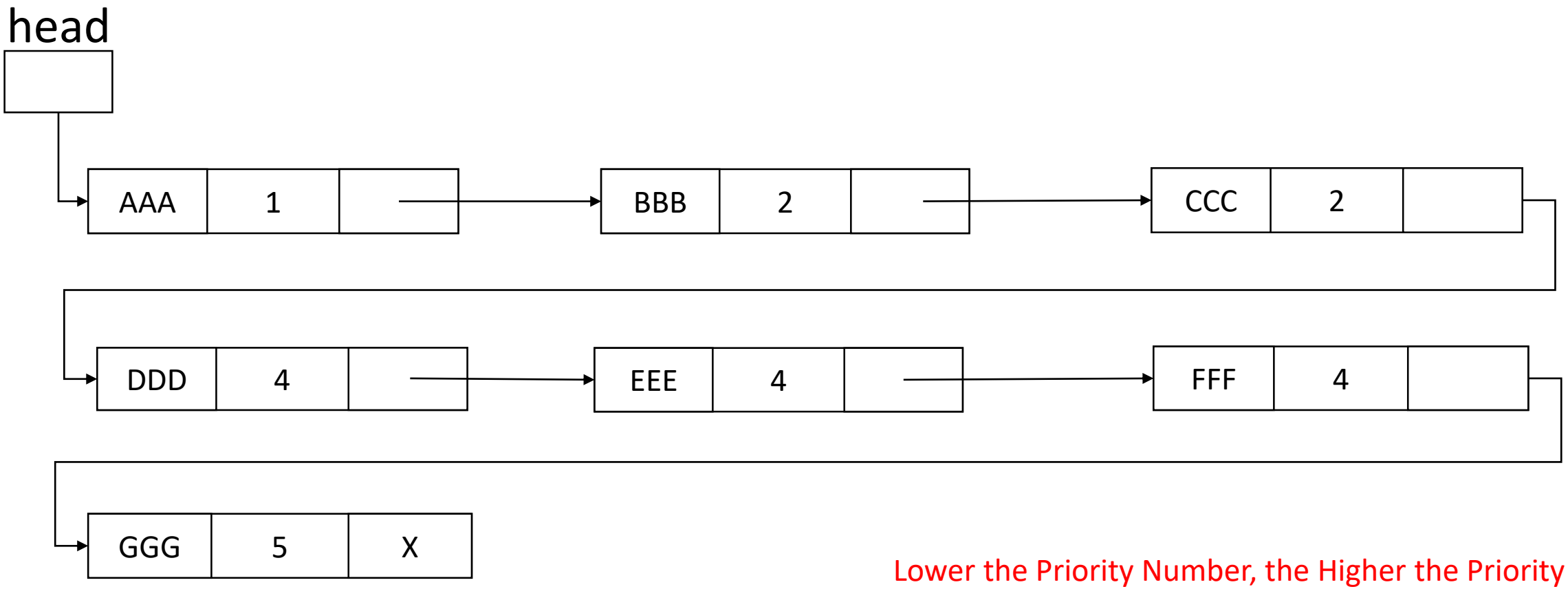
- Linear data structure in which elements can be inserted or deleted at either end
  - However, no element can be added and deleted from the middle
- Operations
  - Insertion at rear end
  - Insertion at front end
  - Deletion at front end
  - Deletion at rear end
- Other variants of a **double-ended queue**
  - **Input restricted deque**: Insertions can be done only at one end while deletions can be done from both ends
  - **Output restricted deque**: Deletions can be done only at one end while insertions can be done on both ends

Priority Queue

- Each element is assigned a priority (value) and the order in which elements are deleted (processed) is based on the rules:
  - An element of higher priority is processed before any element of lower priority
  - Elements with the same priority are processed according to the order in which they were added
- Applications
  - Priority queue at airport check-in counters
  - Priority queue in a time-sharing system

Priority Queue

- Linked List Representation: Node



Lower the Priority Number, the Higher the Priority

References

- Saymour L., **“Data Structures”,** Schaum’s Outline Series, McGraw Hill, Revised First Edition
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, **“Introduction to Algorithms”**, The MIT Press
- Sahni, S., **“Data Structures, Algorithms, and Applications in C++”,** WCB/McGraw-Hill