Data Structure and Algorithm (CS 102)

Queue & its Applications

Dr. Sambit Bakshi
Dept. of CSE, NIT Rourkela

Queue

- A queue is a linear list of elements in which
  - deletion can take place only at one end called Front, and
  - Insertion takes place at one end called Rear
- Queues are also known as First-In-First-Out (FIFO) list

Queue

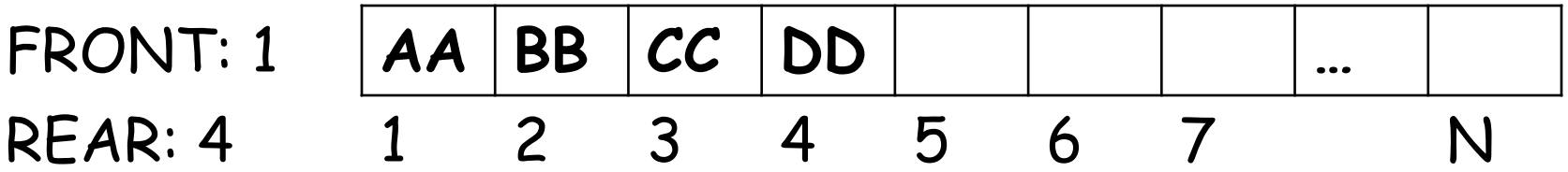
- Queue are represented in two-ways
  - Linear Array
  - One-way Linked List

Array representation of Queue

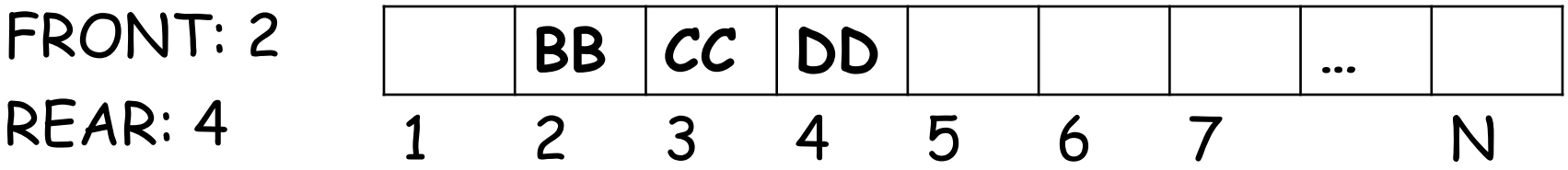
- A queue is maintained by a
  - linear array QUEUE
  - Two pointer variables
    - FRONT : Containing the location of the front element of the queue
    - REAR : Containing the location of the last element of the queue

- FRONT == 0 indicates that the queue is empty

Queue



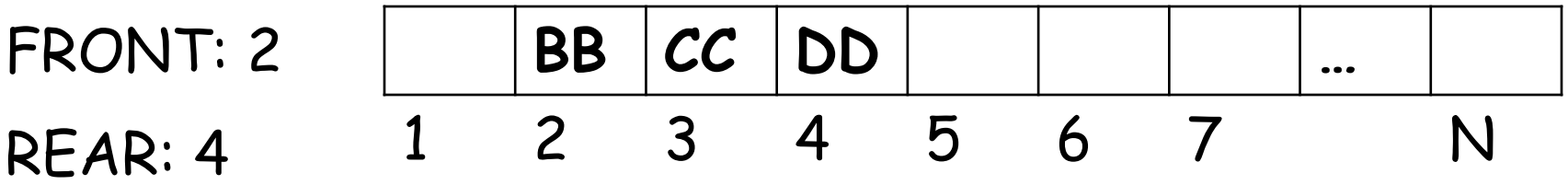
Delete an element



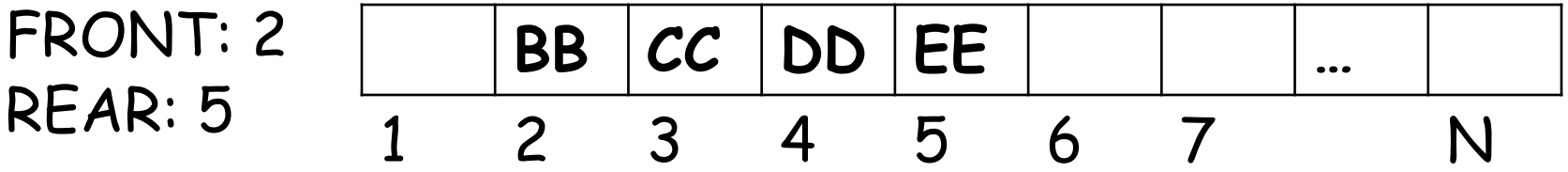
Whenever an element is deleted from the queue, the value of FRONT is increased by 1

FRONT = FRONT + 1

Queue



Insert an element

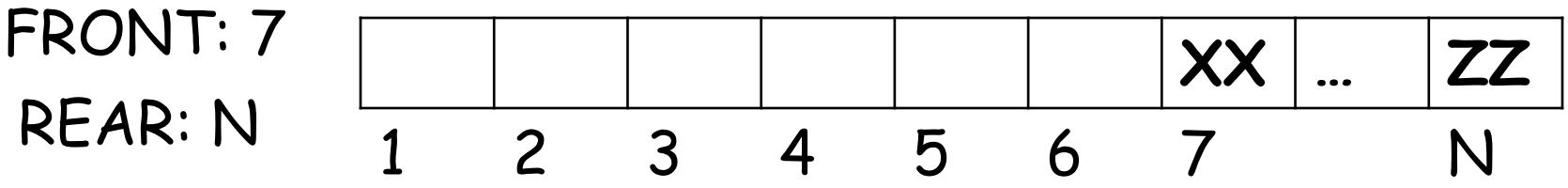


Whenever an element is inserted into the queue, the value of REAR is increased by 1

REAR = REAR + 1

Queue

- REAR = N and Insert an element into queue



Move the entire queue to the beginning of the array
Change the FRONT and REAR accordingly
Insert the element

This procedure is too expensive

Queue

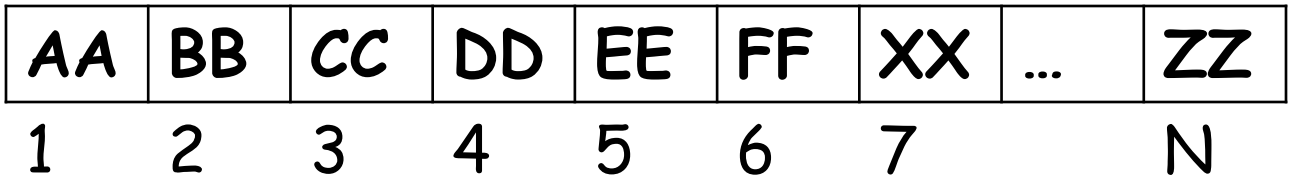
- Queue is logically assumed to be circular
- QUEUE[1] comes after QUEUE[N]
- Instead of increasing REAR to N +1, we reset REAR = 1 and then assign

  QUEUE[REAR] = ITEM

Queue

- FRONT = N and an element of QUEUE is Deleted

FRONT: N
REAR: 7

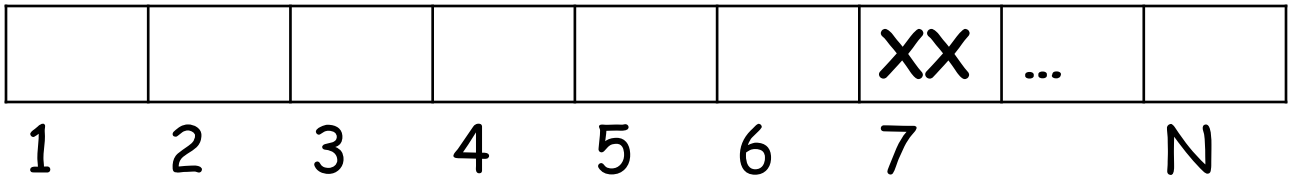


We reset FRONT = 1, instead of increasing FRONT to N + 1

Queue

- QUEUE contain one element
  FRONT = REAR ≠ 0

FRONT: 7
REAR: 7



Algorithm to Insert in Q

[1] If FRONT = REAR % N + 1 then

Print: Overflow and Exit

[2] If REAR = 0 **then** //first time insertion

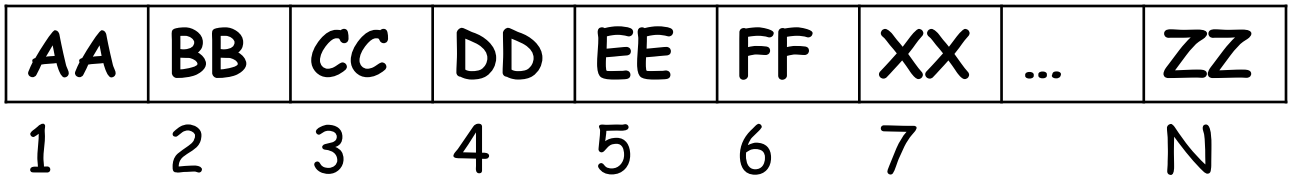
Set FRONT = 1 and REAR = 1

Else Set REAR = REAR % N + 1

[3] Set QUEUE[REAR] = ITEM

[4] Exit

Queue



FRONT = REAR % N + 1 [FULL QUEUE]

Algorithm to Delete from Q

[1] If FRONT = 0 then
Print: Underflow and Exit

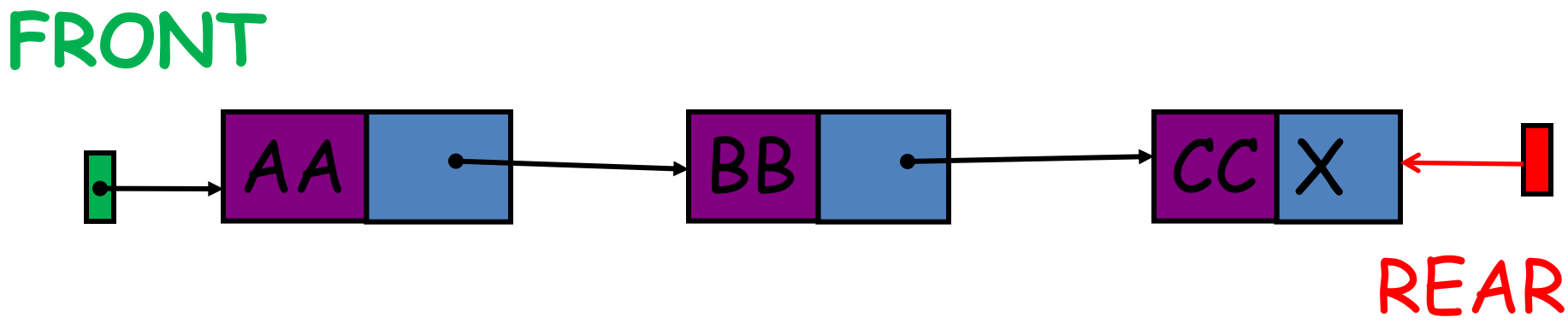
[2] Set ITEM = QUEUE[FRONT]

[3] If FRONT = REAR **then**
Set FRONT = 0 and REAR = 0
Else Set FRONT = FRONT % N + 1
[4] Exit

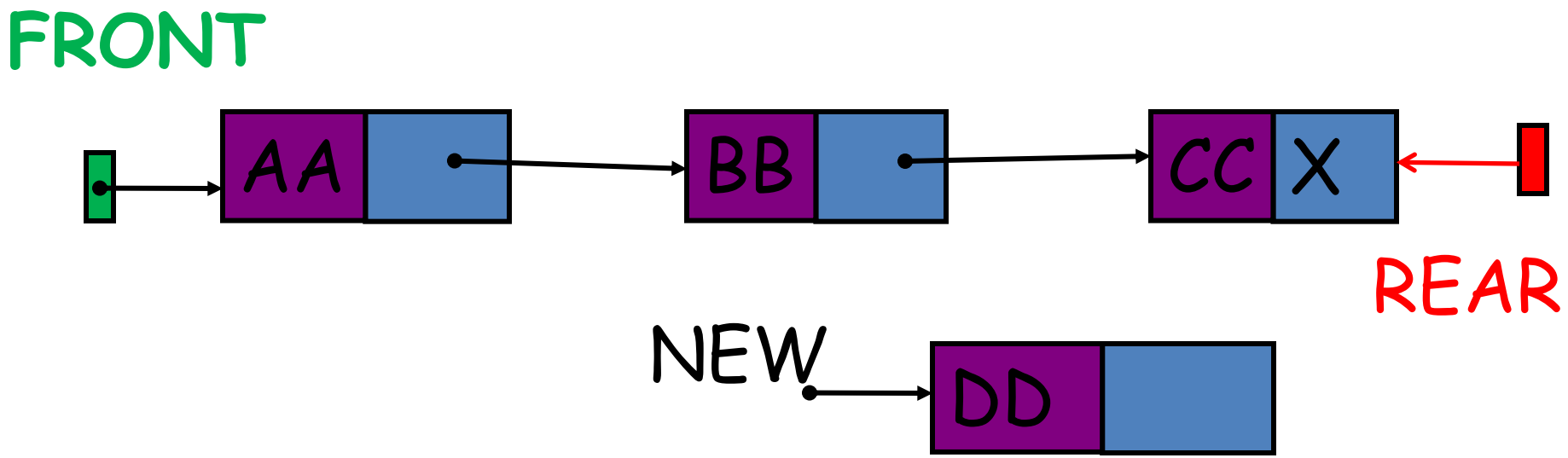
Linked List Representation of Queue

- A linked queue is a queue implemented as linked list with two pointer variable FRONT and REAR pointing to the nodes which is in the FRONT and REAR of the queue

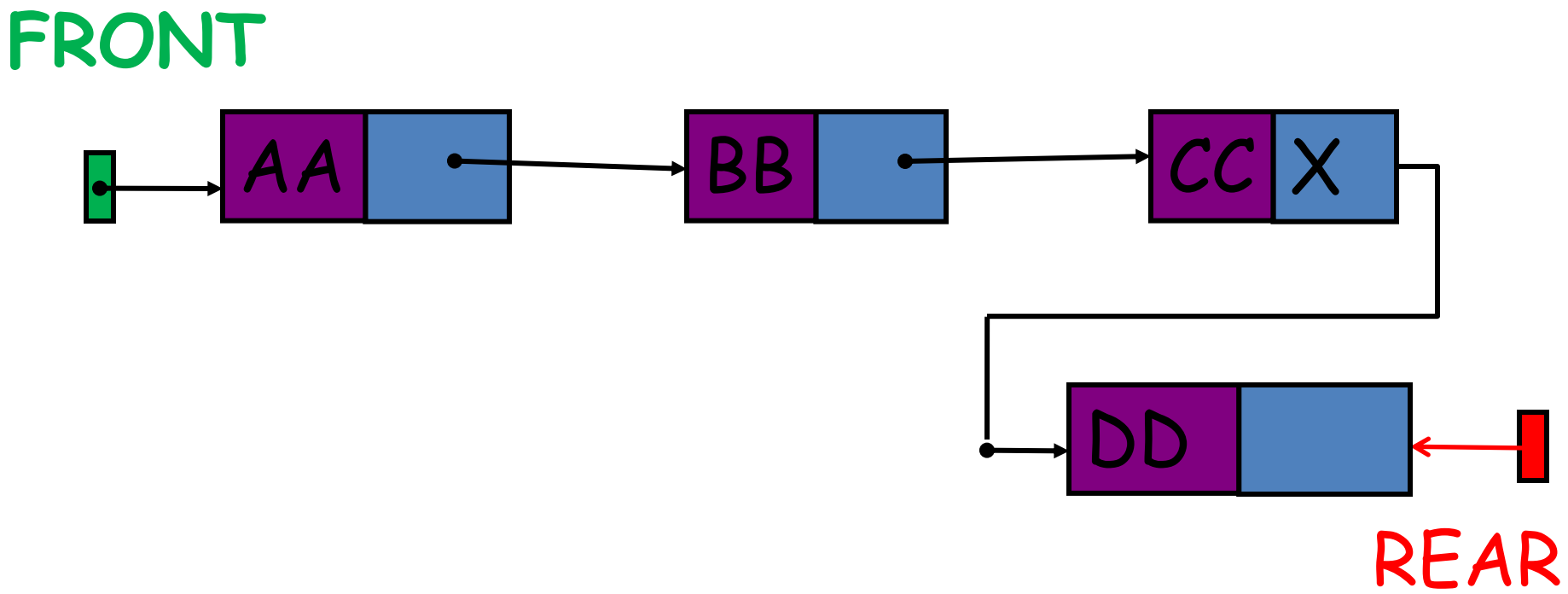
Linked List Representation of Queue



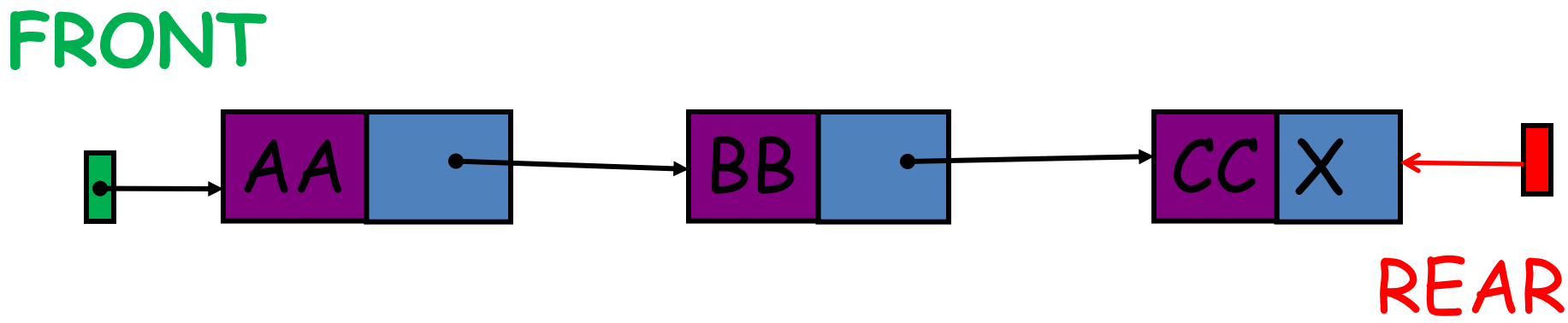
Insertion in a Queue



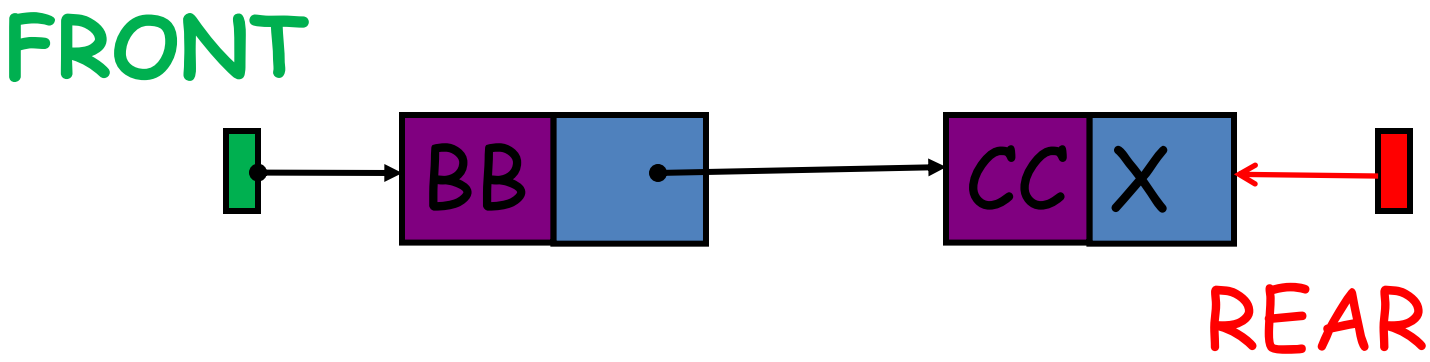
Insertion in a Queue



Delete from a Queue



Delete from a Queue



Linked Queue

- No need to view it as circular for efficient management of space

Insertion

```
[1]  NEW -> INFO = ITEM
     NEW -> LINK  = NULL
[2] If (FRONT = NULL) then
          FRONT = REAR = NEW
     else
          Set REAR -> LINK = NEW
               REAR = NEW
[3] Exit
```

Deletion

[1] If (FRONT = NULL) then
Print: Underflow, and Exit

[2] PTR=FRONT

[3] FRONT = FRONT -> LINK

[4] FREE PTR

[5] Exit

Deque

- A **deque** is a linear list in which elements can be added or removed at either end but not in the middle

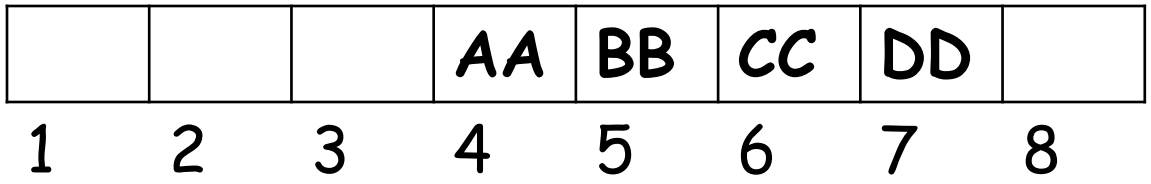
- Deque is implemented by a circular array DEQUE with pointers **LEFT** and **RIGHT** which points to the two end of the deque

Deque

- LEFT = RIGHT = 0 indicate deque is empty

LEFT: 4

RIGHT: 7



Variation of deque

- There are two variation of deque

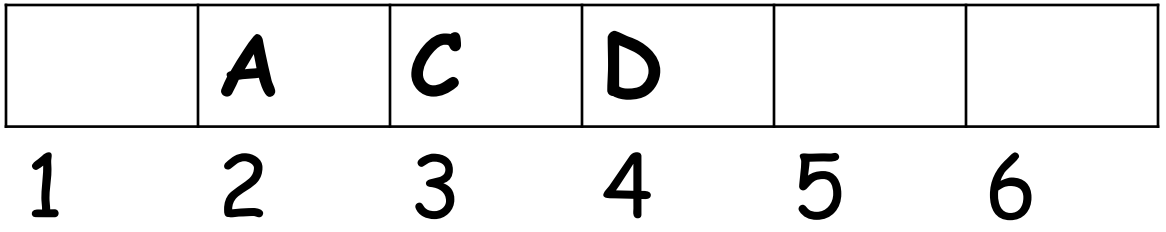
[1] **Input-restricted queue**: Deque which allows insertions at only one end of the list but allows deletion at both ends of the list

[2] **Output-restricted queue**: Deque which allows deletion at only one end of the list but allows insertion at both ends of the list

Deque

LEFT: 2

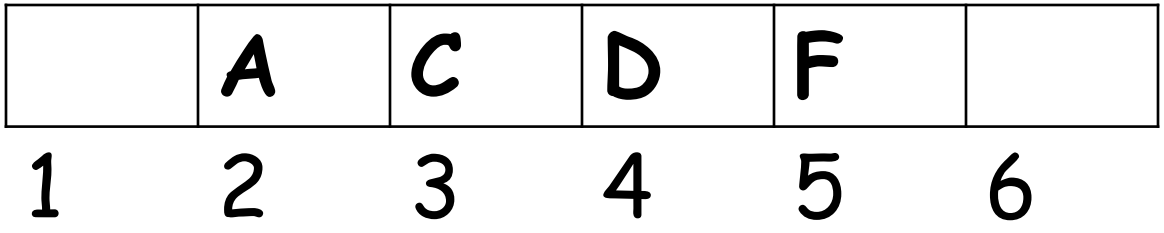
RIGHT: 4



F is added to the right

LEFT: 2

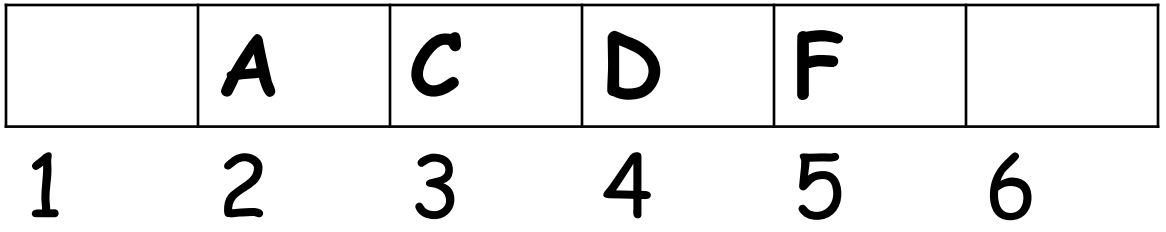
RIGHT: 5



Deque

LEFT: 2

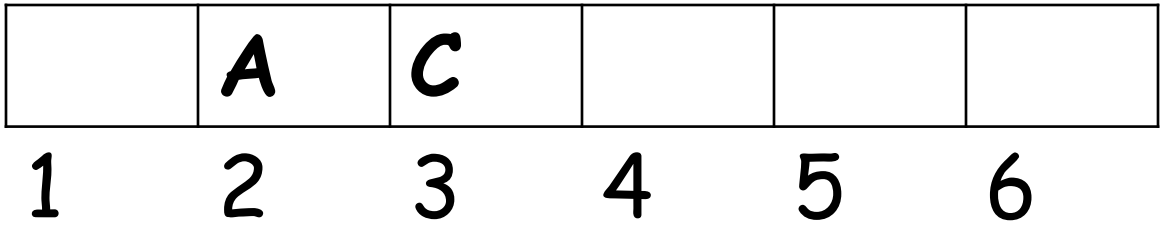
RIGHT: 5



Two Letters on right is deleted

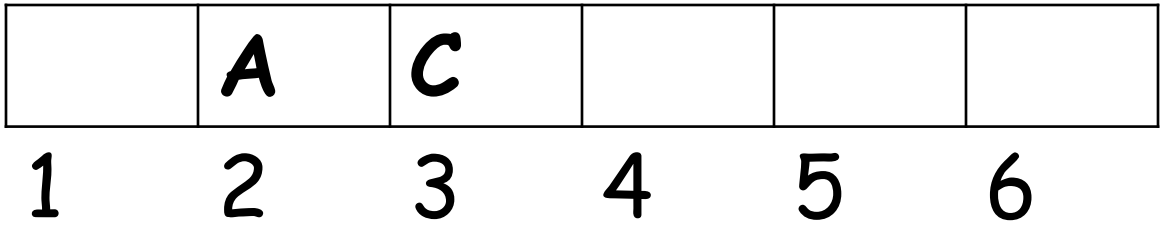
LEFT: 2

RIGHT: 3



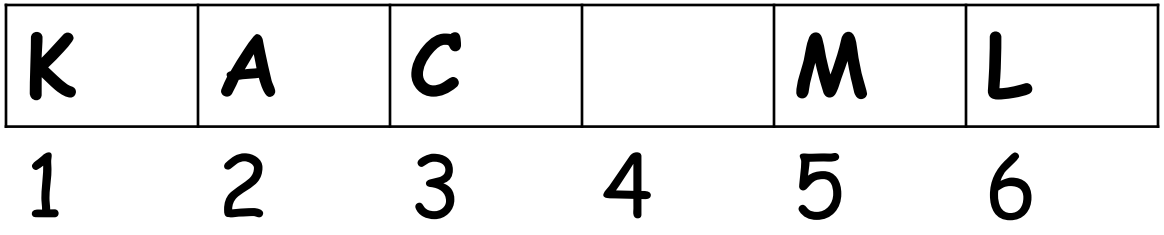
Deque

LEFT: 2
RIGHT: 3



K, L and M are added to the Left

LEFT: 5
RIGHT: 3



- What is initial value of LEFT and RIGHT?
- Special first case condition
- Condition for underflow
- Condition for overflow

Priority Queue

- A priority queue is a collection of elements such that each element has been assigned a priority and that the order in which the elements are deleted and processed comes from the following rules:

[1] Elements of higher priority is processed before any elements of lower priority

[2] Two elements with the same priority are processed according to the order in which they were added to the queue

Priority Queue

- There are different ways a priority queue can be represented such as

  [1] One-way List

  [2] Multiple queue

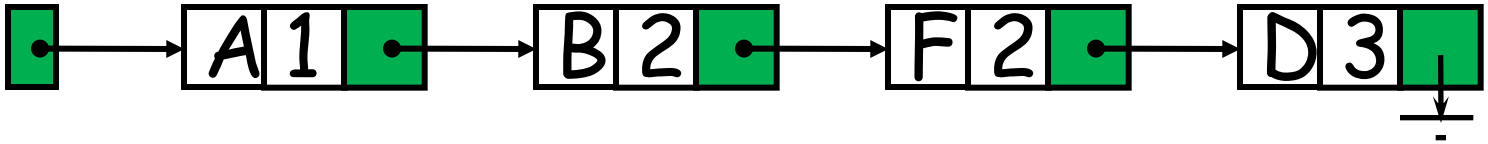
One-Way List Representation of a Priority Queue

[1] Each node in the list will contain three items of information: **an information field INFO, a priority number PRN, and a link filed LINK**

[2] A node X precedes a node Y in the list
(a) when X has **higher priority** than Y or
(b) when both have same priority but X was **added to the list before** Y

Queue

Head



Insertion and Deletion

- Deletion : Delete the first node in the list.
- Insertion: Find the location of Insertion

Add an ITEM with priority number N

[a] Traverse the list until finding a node X whose priority is less than N. Insert ITEM in front of node X

[b] If no such node is found, insert ITEM as the last element of the list

Array representation of Priority Queue

- Separate queue for each level of priority
- Each queue will appear in its own circular array and must have its own pair of pointers, FRONT and REAR
- If each queue is given the same amount space then a 2D queue can be used

	FRONT	REAR
1	2	2
2	1	3
3	0	0
4	4	1
5	3	3

QUEUE

	1	2	3	4	5
1		AA			
2	BB	CC	DD		
3					
4	FF			DD	EE
5			GG		

Deletion Algorithm [outline]

[1] Find the smallest K such that FRONT[K] $\neq$ 0

[2] Delete and process the front element in row K of QUEUE

[3] Exit

Insertion Algorithm [outline]

Insert an element with priority M

[1] Insert ITEM as the rear element in row M of QUEUE

[2] Exit