Queues

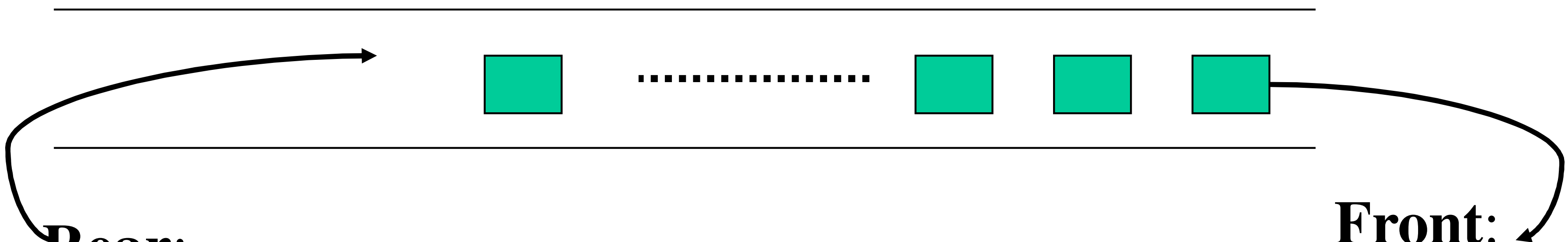
Definition of a Queue

- A queue is a data structure that models/enforces the **first-come first-serve** order, or equivalently the **first-in first-out** (FIFO) order.
- The element that is inserted first into the queue is deleted first
  - The element that is inserted last is deleted last.
- A waiting line is a good real-life example of a queue.

Queues

- Linear list.
- One end is called front (head).
- Other end is called rear (tail).
- Insertions are done at the rear (tail) only.
- Deletions are made from the front(head) only.

A Graphic Model of a Queue



Rear:
All new items
are added on
this end

Front:
All items are
deleted from
this end

Examples of Queues

- An electronic mailbox is a queue
  - The ordering is chronological (by arrival time)
- A waiting line in a store, at a service counter, on a one-lane road
- Equal-priority processes waiting to run on a processor in a computer system

Example

Given the following Queue, how will it change when we apply the given operations?

Rear — Front

			4	1	3

enqueue(1);

		1	4	1	3

enqueue(5);

	5	1	4	1	3

dequeue();

	5	1	4	1	

dequeue();

	5	1	4		

dequeue();

	5	1			

Queue ADT

instances

ordered list of elements; one end is the front; the other is the rear;

operations

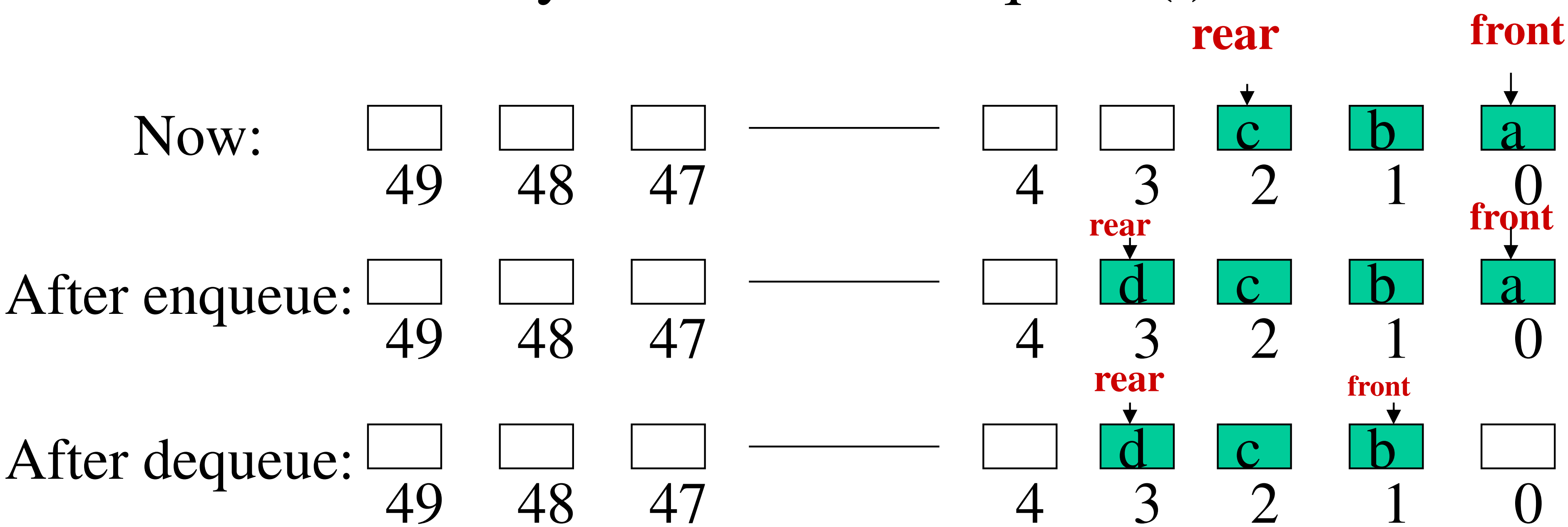
empty():	Return true if queue is empty, return false otherwise
size():	Return the number of elements in the queue
front():	Return the front element of queue
rear():	Return the rear element of queue
dequeue():	Remove an element from the queue
enqueue(x):	Add element x to the queue

It is also possible to represent Queues using

1. Array-based representation
2. Linked representation

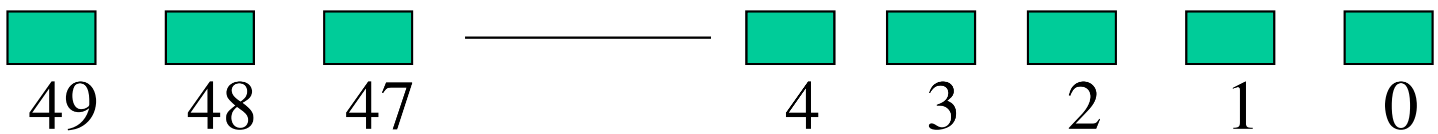
How front and rear Change

- **front** increases by 1 after each dequeue( )
- **rear** increases by 1 after each enqueue( )

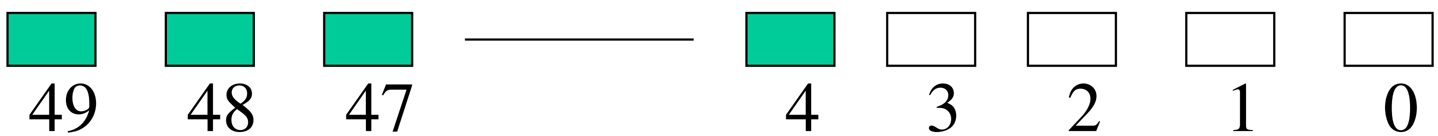


False-Overflow Issue First

- Suppose 50 calls to enqueue have been made, so now the queue array is full



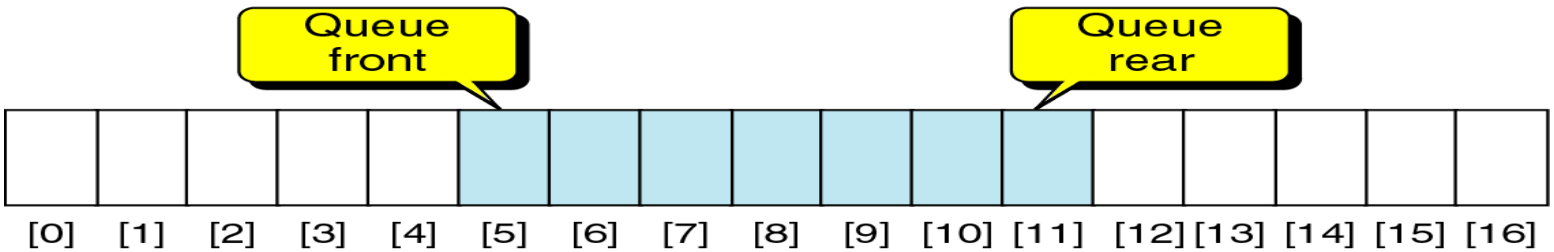
- Assume 4 calls to dequeue( ) are made



- Assume a call to enqueue( ) is made now.
- The rear part have no space, but the front has 4 unused spaces; if never used, they are wasted.

Circular Queue

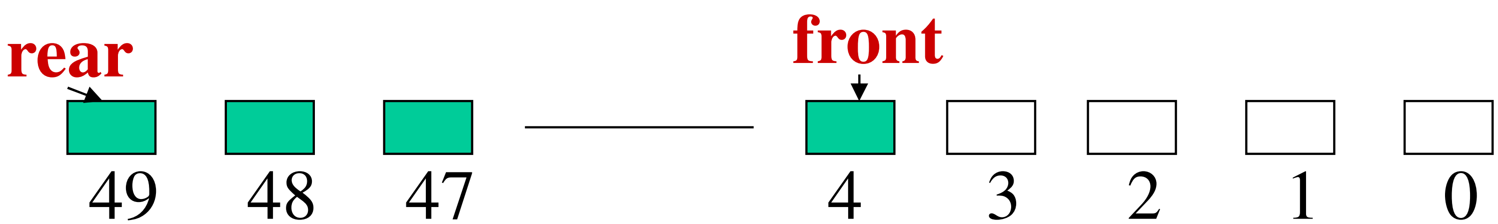
- Use Linear Array to implement a queue.



- Waste of memory: The deleted elements can not be re-used.
- Solution: to use circular queue.
- Two implementations:
  - Using n-1 space.
  - Using n space + full tag

Solution: A Circular Queue

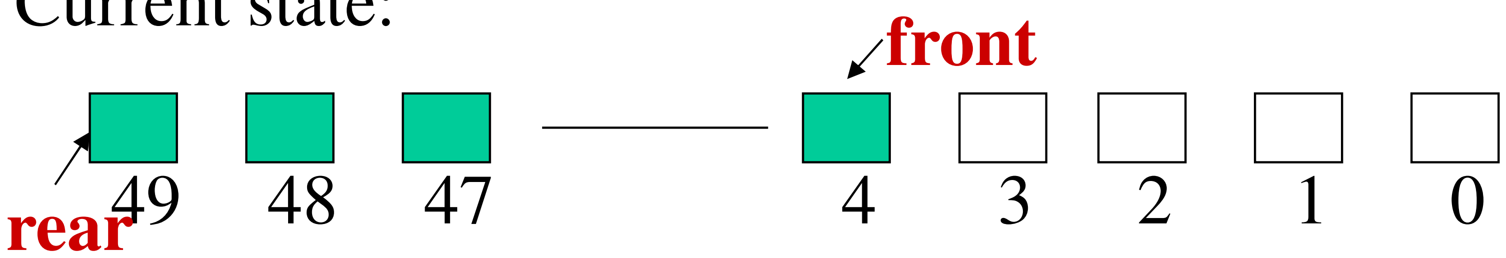
- Allow the front (and the rear) to be moving targets
- When the rear end fills up and front part of the array has empty slots, new insertions should go into the front end



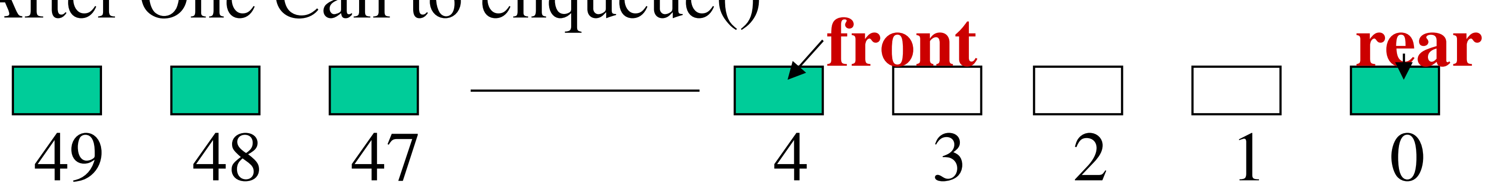
- Next insertion goes into slot 0, and rear tracks it. The insertion after that goes into a lot 1, etc.

Illustration of Circular Queues

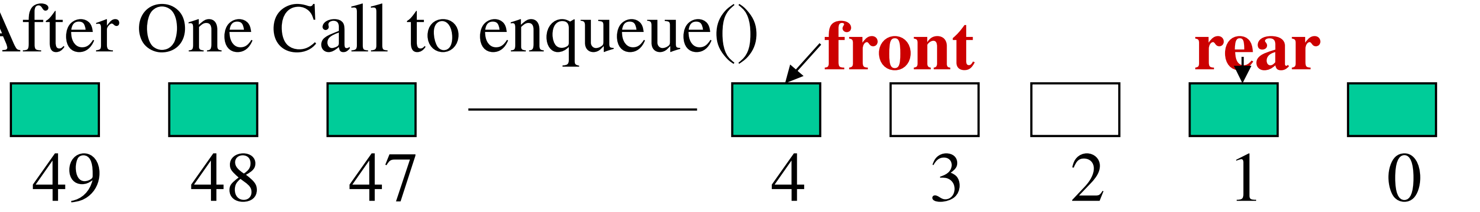
- Current state:



- After One Call to enqueue()



- After One Call to enqueue()



Numeric for Circular Queues

- **front** increases by (1 modulo capacity) after each dequeue( ):

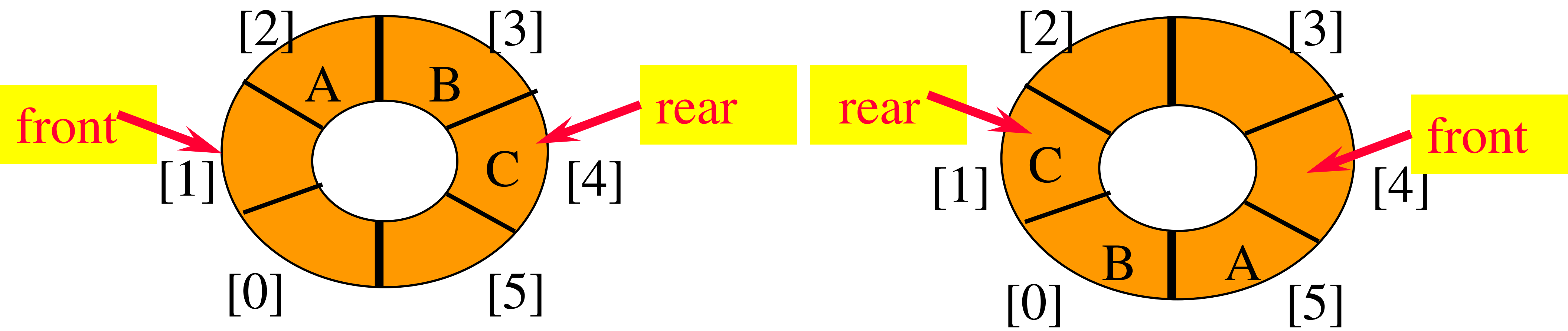
  front = (**front** +1) % capacity;

- **rear** increases by (1 modulo capacity) after each enqueue( ):

  rear = (**rear** +1) % capacity;

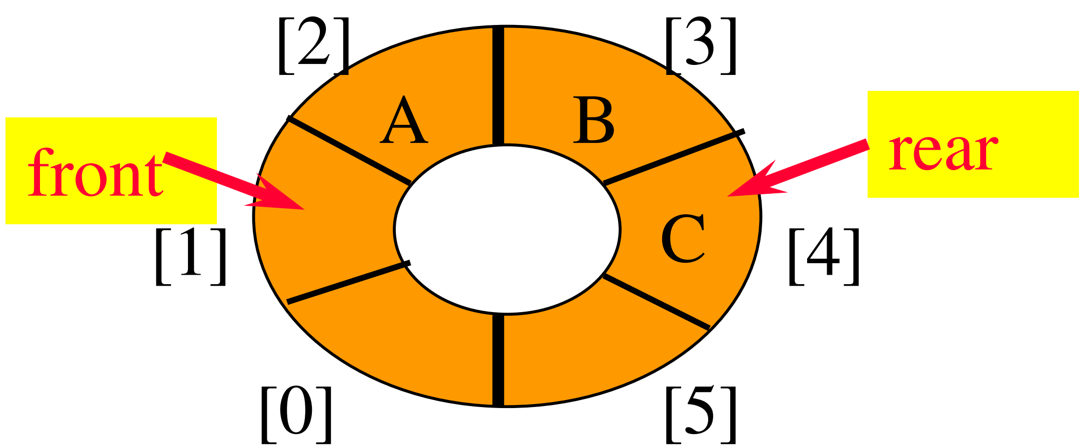
Yet another Illustration of Circular Queues(Using n-1 space)

- Use integer variables front and rear.
  - front is one position counterclockwise from first element
  - rear gives position of last element



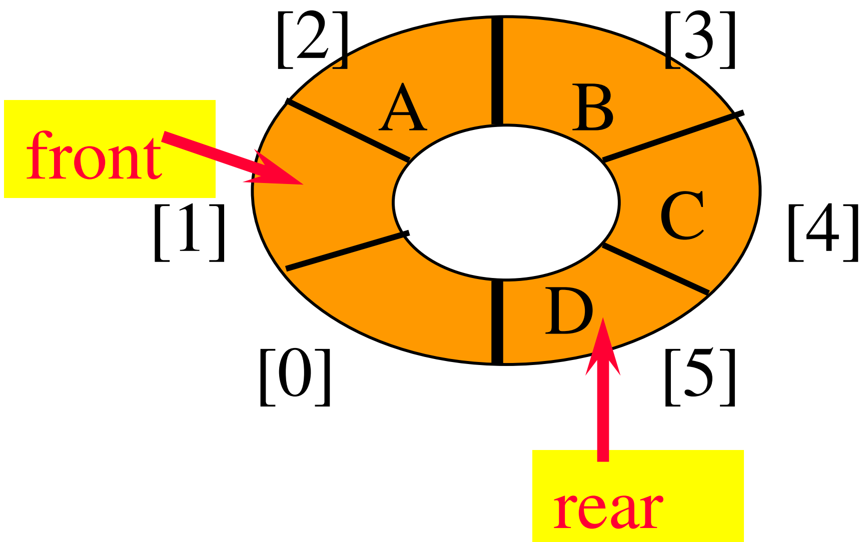
Add An Element

- Move rear one clockwise.



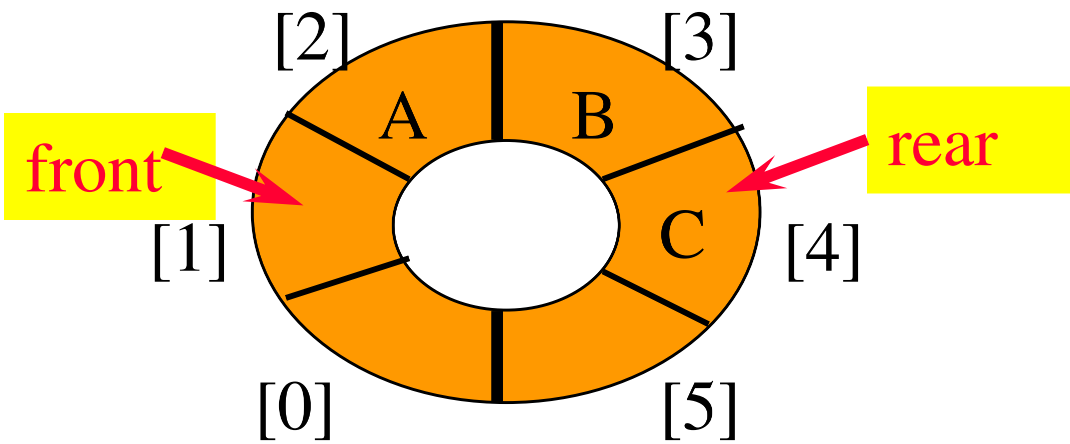
Add An Element

- Move rear one clockwise.
- Then put into queue[rear].



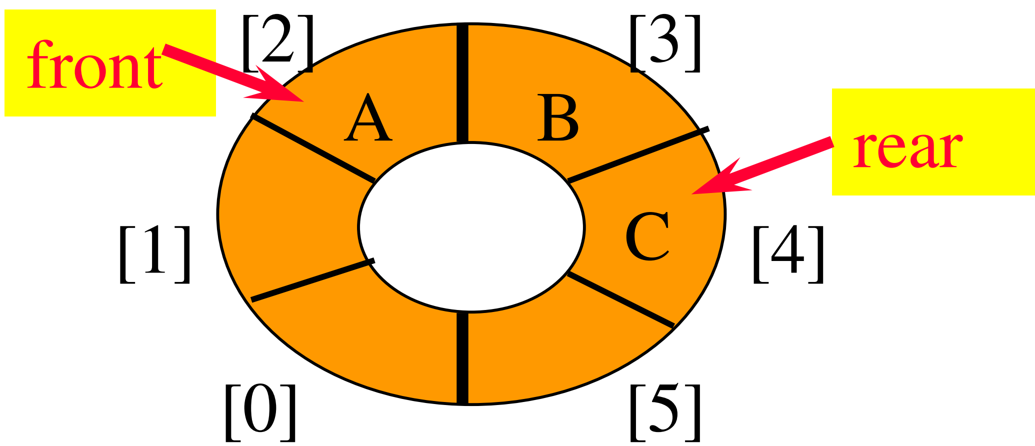
Remove An Element

- Move front one clockwise.



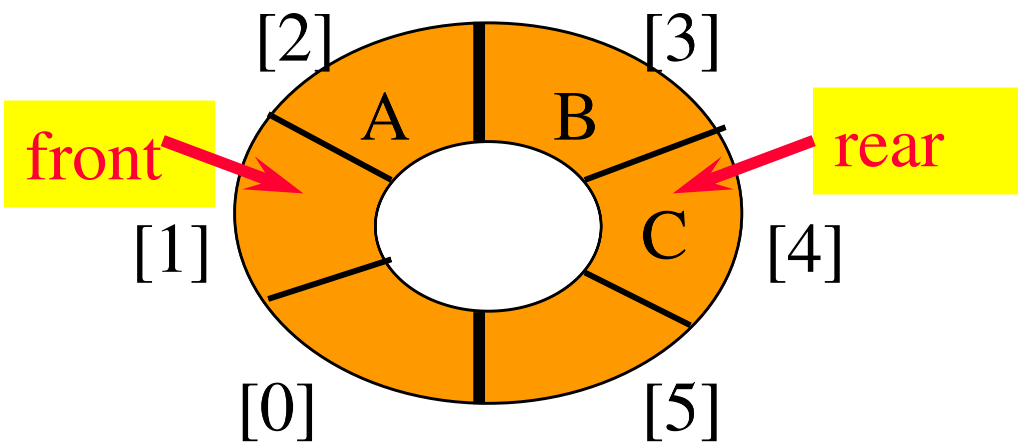
Remove An Element

- Move front one clockwise.
- Then extract from queue[front].



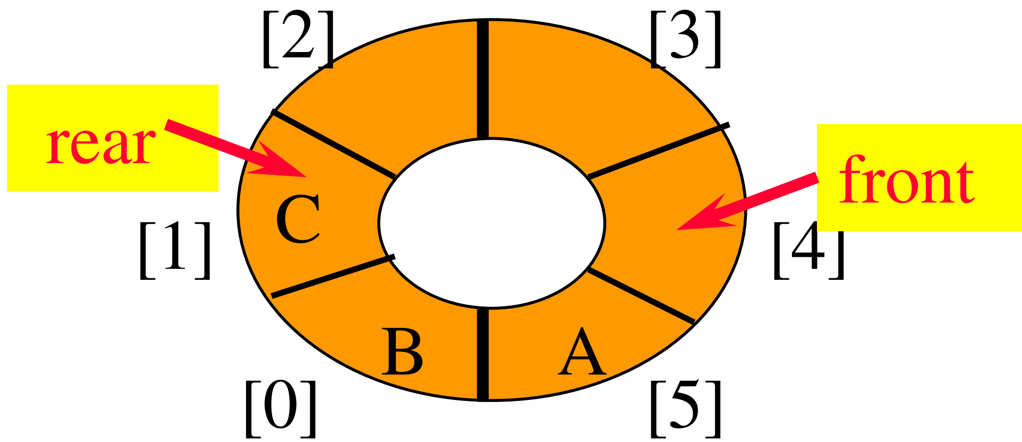
Moving rear Clockwise

- rear++;
  if (rear = = queue.length) rear = 0;

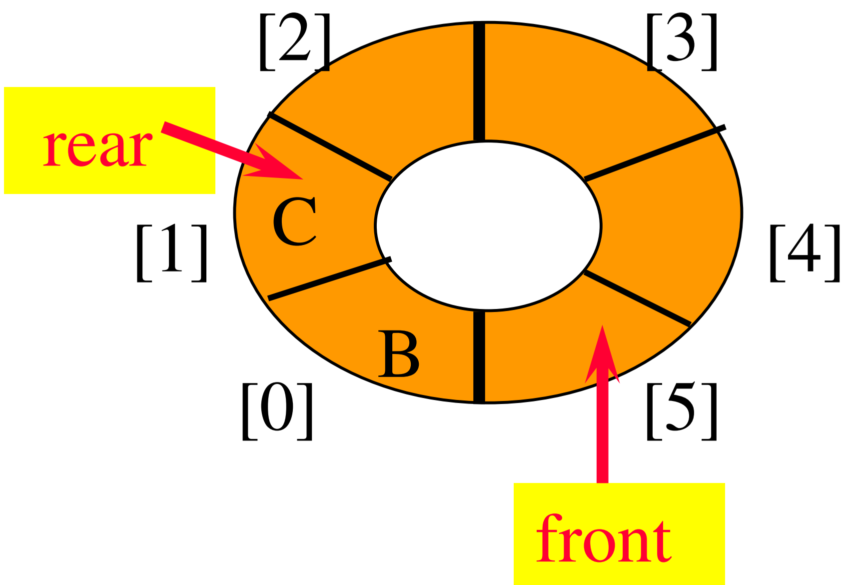


- rear = (rear + 1) % queue.length;

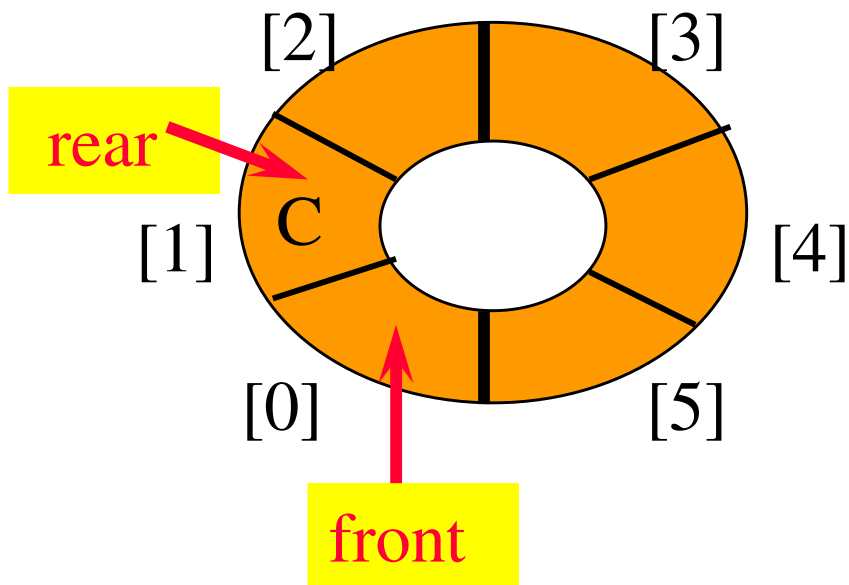
Empty Queue



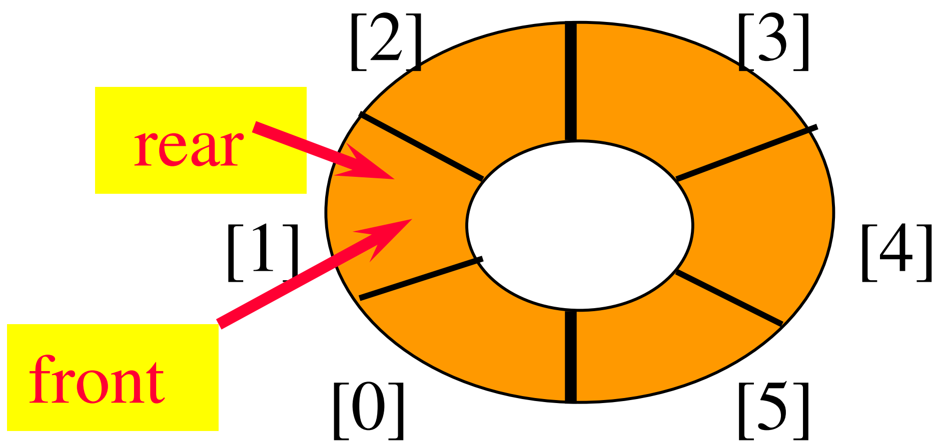
Empty Queue



Empty Queue

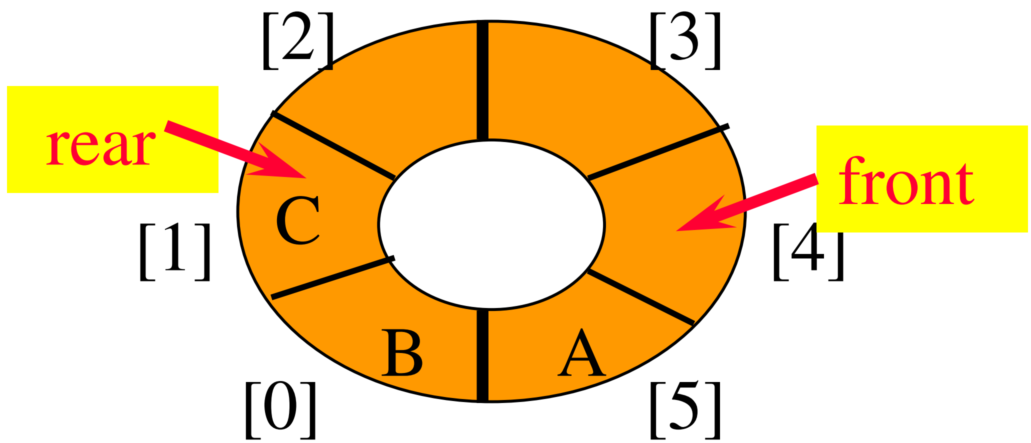


Empty Queue

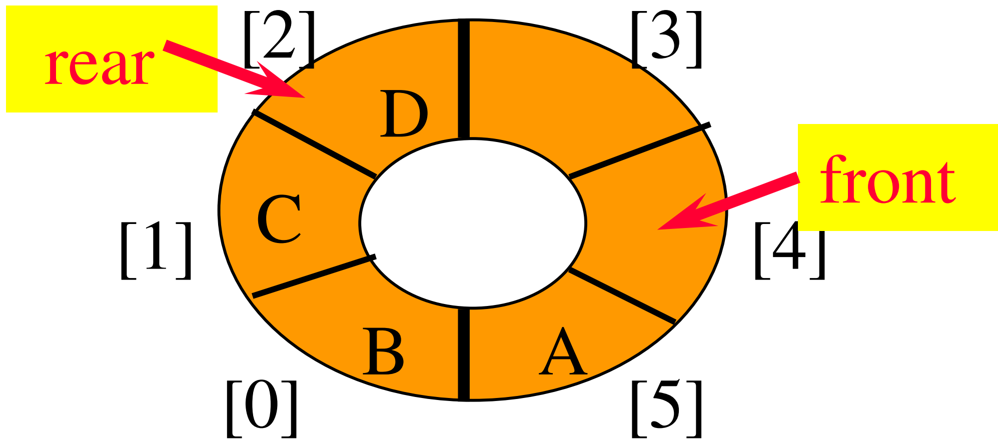


- When a series of removes causes the queue to become empty, front = rear.
- When a queue is constructed, it is empty.
- So initialize front = rear = 0.

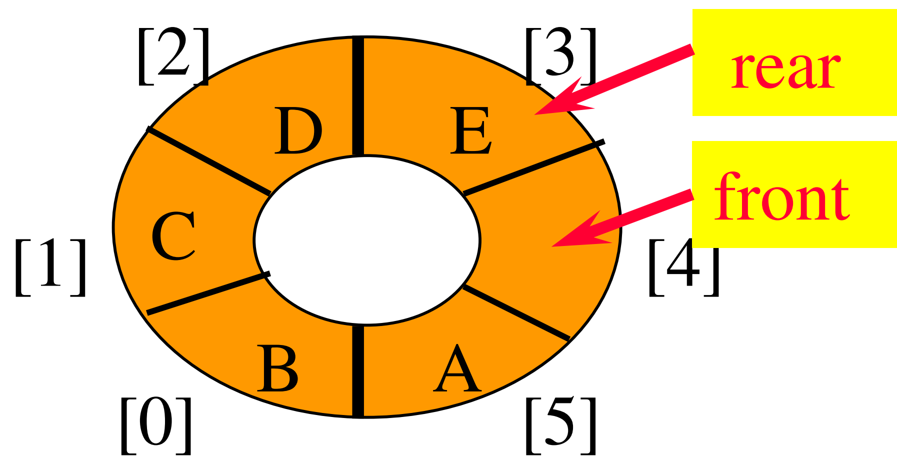
Full Queue



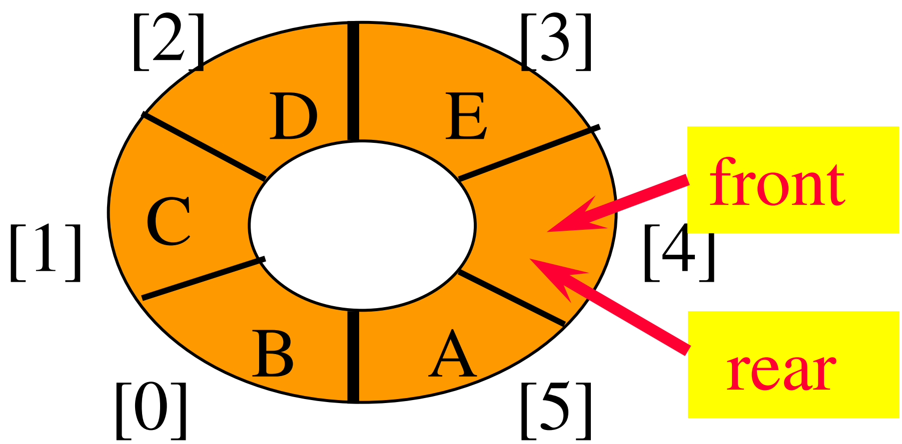
Full Queue



Full Queue



Full Queue



- When a series of enqueue causes the queue to become full, front = rear.
- So, we cannot distinguish between a full queue and an empty queue!

Implementation of Circular queue with (n-1) space used

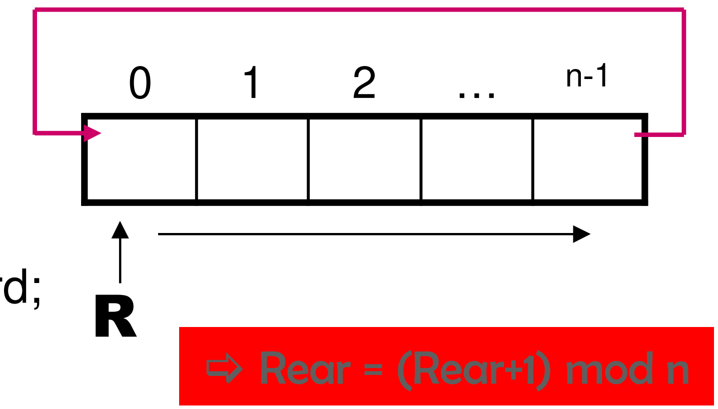
- **Create(Q)**

  Q: Array[0…n-1]

  front = rear = 0 //initialize

- **Enqueue(item, Q) ⇨Queue**

```
{    rear = (rear+1) mod n;  //rear moves forward;
     if( rear = =front){
         QueueFull;      // Queue is full.
         rear = (rear-1) mod n;    //  rear back to the previous position;
     }
     else
         Q[rear]=item;
}
```

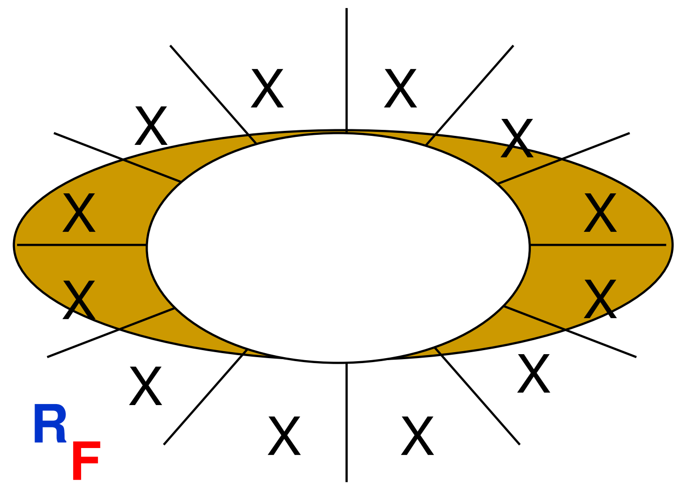


Implementation of Circular queue with (n-1) space used

- Dequeue(Q) ⇨item

```
{
    if( front==rear)
        QueueEmpty;
    else{
            front = (front+1) mod n;
            item = Q[front];
    }
}
```

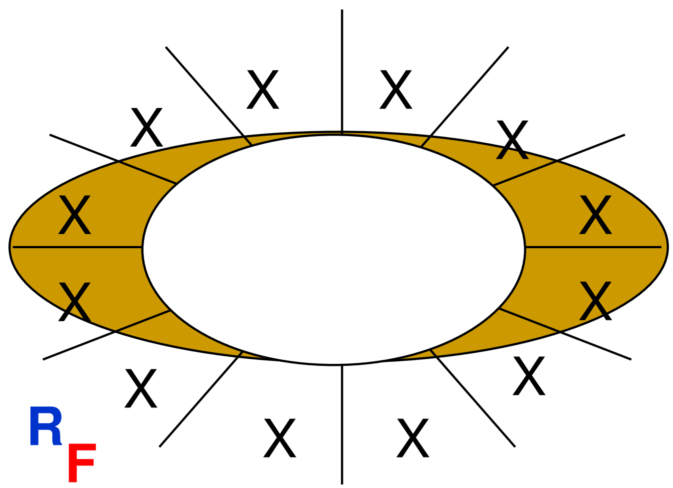
- Note: only **(n-1)** space used;



Implementation of Circular Queue with n space used

- A parameter "Tag" is introduced to help to make sure the queue is Empty or Full:
  - Boolean
  - If Tag = True, combined with other conditions => queue is Full
  - If Tag = False, combined with other conditions => queue is Null
  - **"Tag" can determine the states of the queue solely!**

Implementation of Circular Queue with n space used



Implementation of Circular Queue with n space used

- **Create(Q)**

  Q: Array[0…n-1]

  int front = rear = 0

  Boolean Tag = False (0)

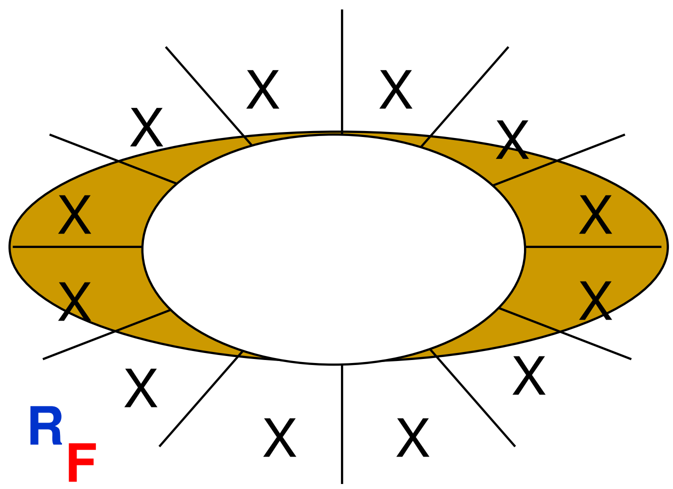
- **Enqueue(item, Q) ⇨Queue**

```
{
    if (rear == front && Tag = =True)
        Overflow message: QueueFull;
    else {
        rear = (rear+1) mod n;    //rear moves forward;
        Q[rear]=item;
        if (rear==front)
                Tag=True;
    }
}
```

Implementation of Circular Queue with n space used

- **Dequeue(Q) ⇨item**

```
{
    if (front==rear && Tag==False)
       Underflow message: QueueEmpty;
    else {
              front = (front+1) mod n;
              item = Q[front];
              if (front==rear)
                     Tag=False;
     }
}
```



Slides and figures have been collected from various publicly available Internet sources for preparing the lecture slides of IT2001 course. I acknowledge and thank all the original authors for their contribution to prepare the content.