Applications Of Stacks

- Execution of function calls
- Parantheses Maching
- Arithmetic Expressions Evaluation
  - Conversion of infix expressions into Pre and Post fix notations
  - Evaluation of Pre and Post fix expressions

Applications Of Stacks

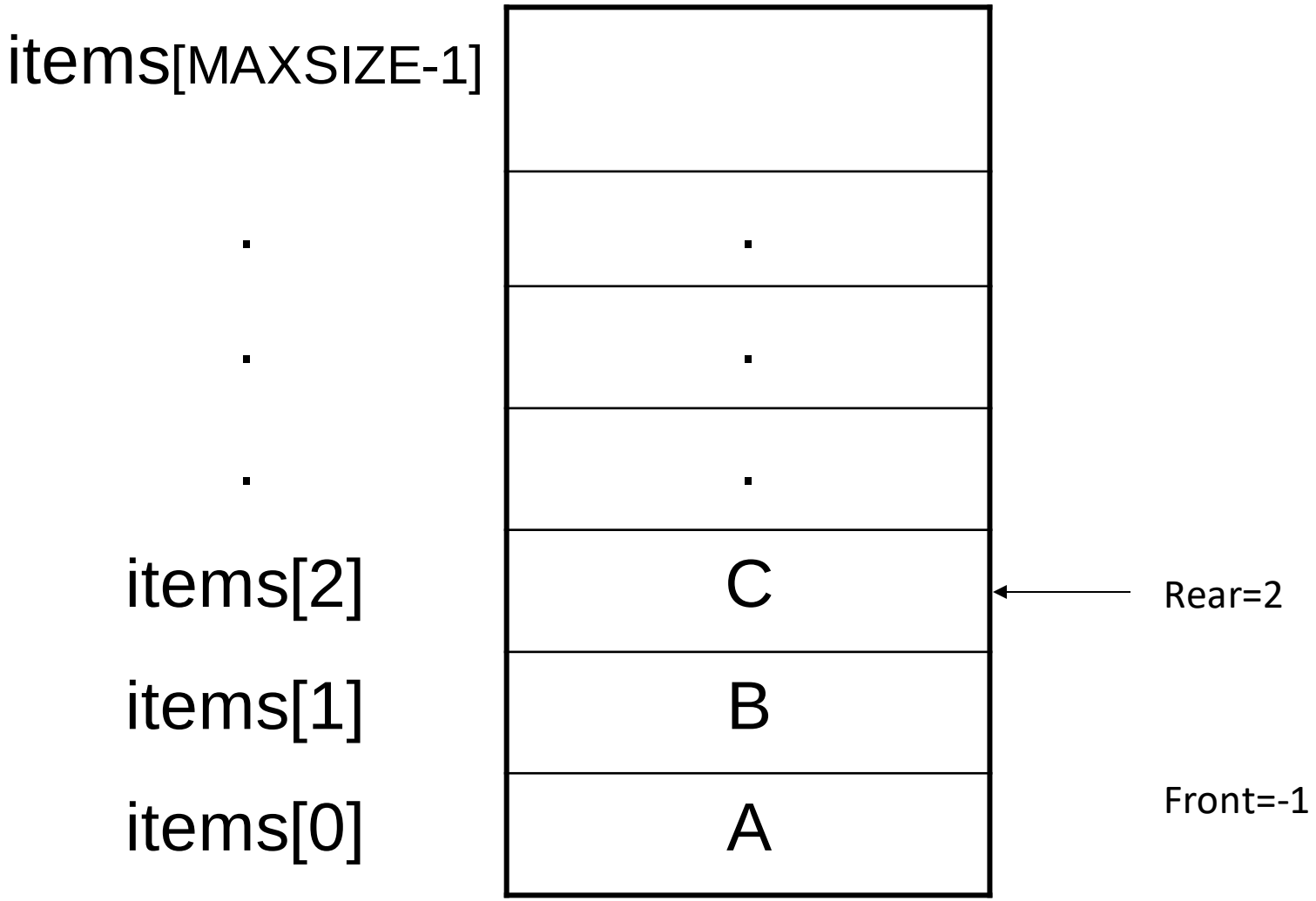
- The stack data structure is used in the evaluation and conversion of arithmetic expressions.
- It is used for parenthesis checking.
- While reversing a string, the stack is used as well.
- Stack is used in memory management.
- It is also used for processing function calls.
- The stack is used to convert expressions from infix to postfix.
- The stack is used to perform undo as well as redo operations in word processors.
- The stack is used in virtual machines like JVM.
- The stack is used in the media players. Useful to play the next and previous song.
- The stack is used in recursion operations.

- Approach:
  - Whenever a left parenthesis is encountered, it is pushed in the stack
  - Whenever a right parenthesis is encountered, pop from stack and check if the parentheses match
  - Works for multiple types of parentheses ( ), { }, [ ]

DEFINITION OF QUEUE

A **Queue** is an ordered collection of items from which items may be deleted at one end (called the ***front*** of the queue) and into which items may be inserted at the other end (the ***rear*** of the queue).
The first element inserted into the queue is the first element to be removed. For this reason a queue is sometimes called a ***FIFO*** (first-in first-out) list as opposed to the stack, which is a ***LIFO*** (last-in first-out).

Queue

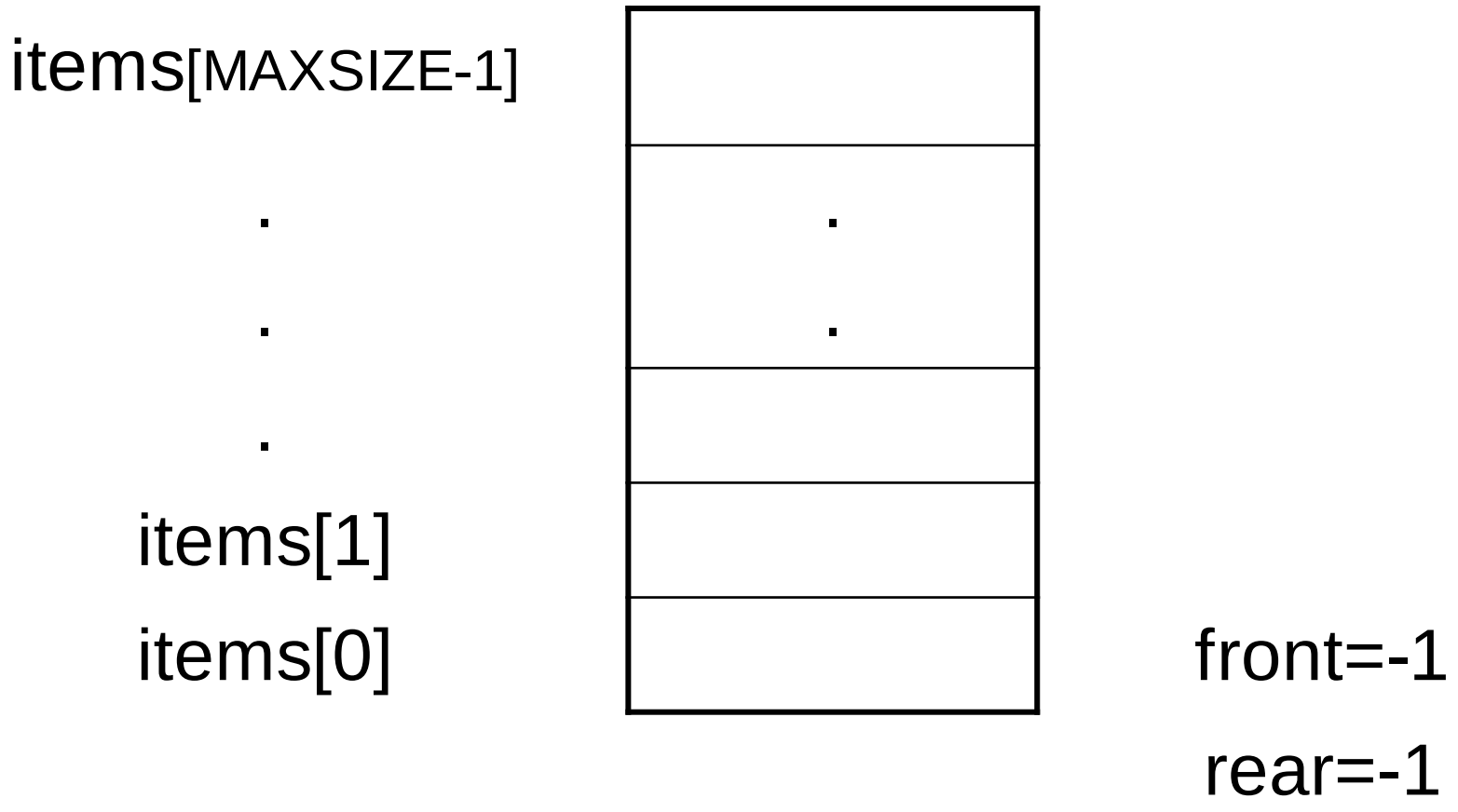


QUEUE OPERATIONS

Initialize **the queue**
Insert **to the rear of the queue**
Remove **(Delete) from the front of the queue**
Is the Queue Empty
Is the Queue Full

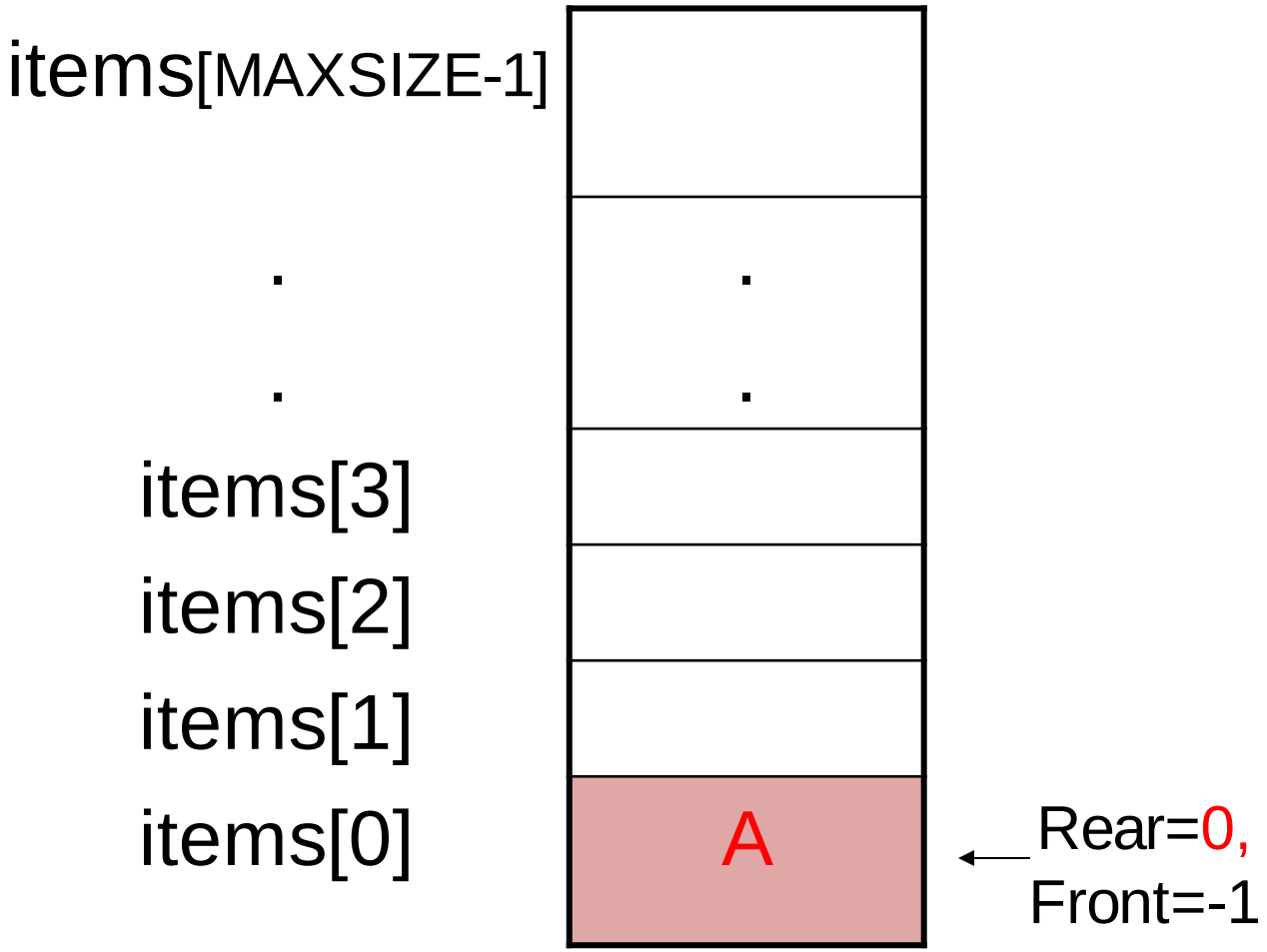
INITIALIZE THE QUEUE

•The queue is initialized by having the front and *rear* set to *-1*. Let us assume that maximum number of the element we have in a queue is MAXSIZE elements as shown below.



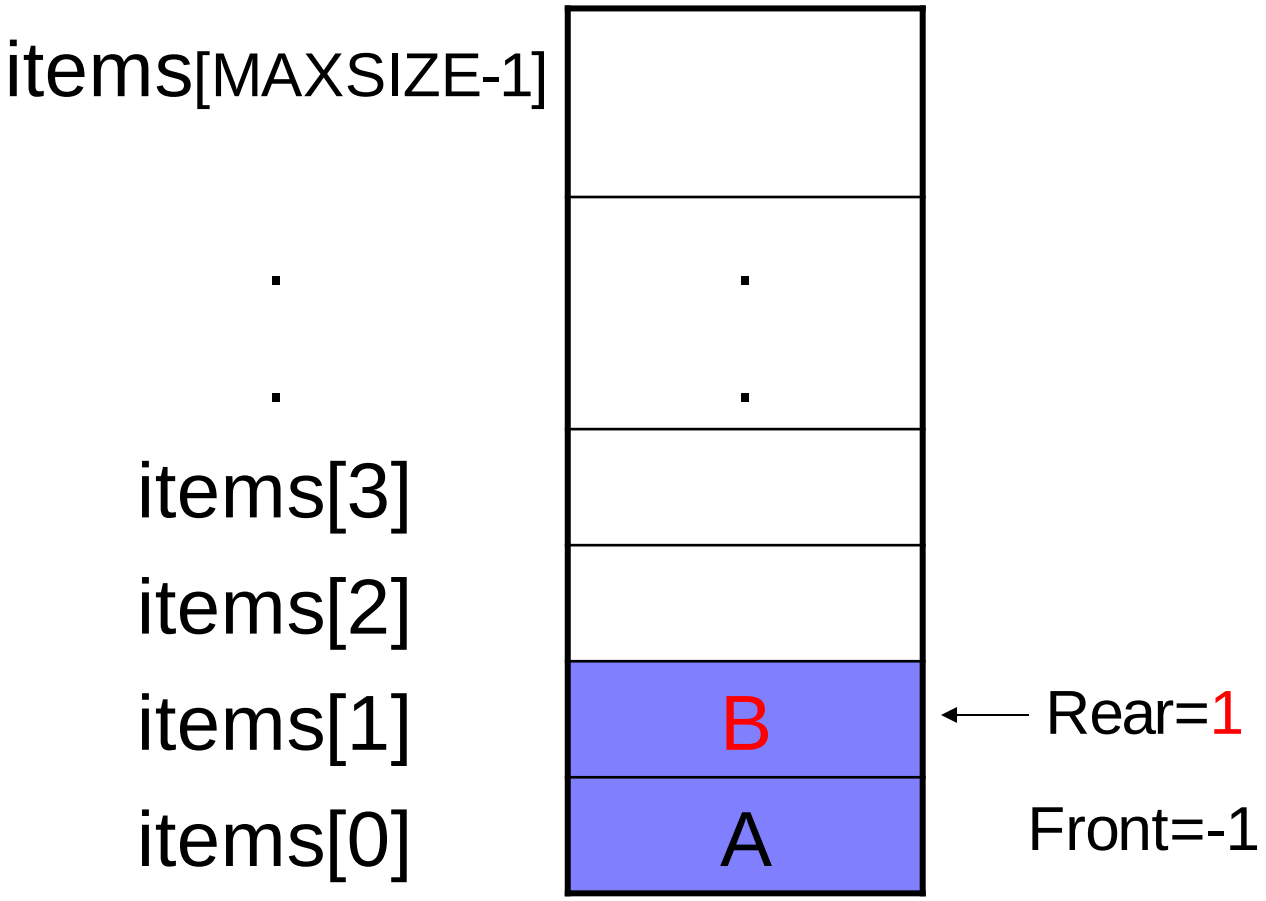
enqueue('A')

an item (*A*) is *inserted* at the *Rear* of the queue



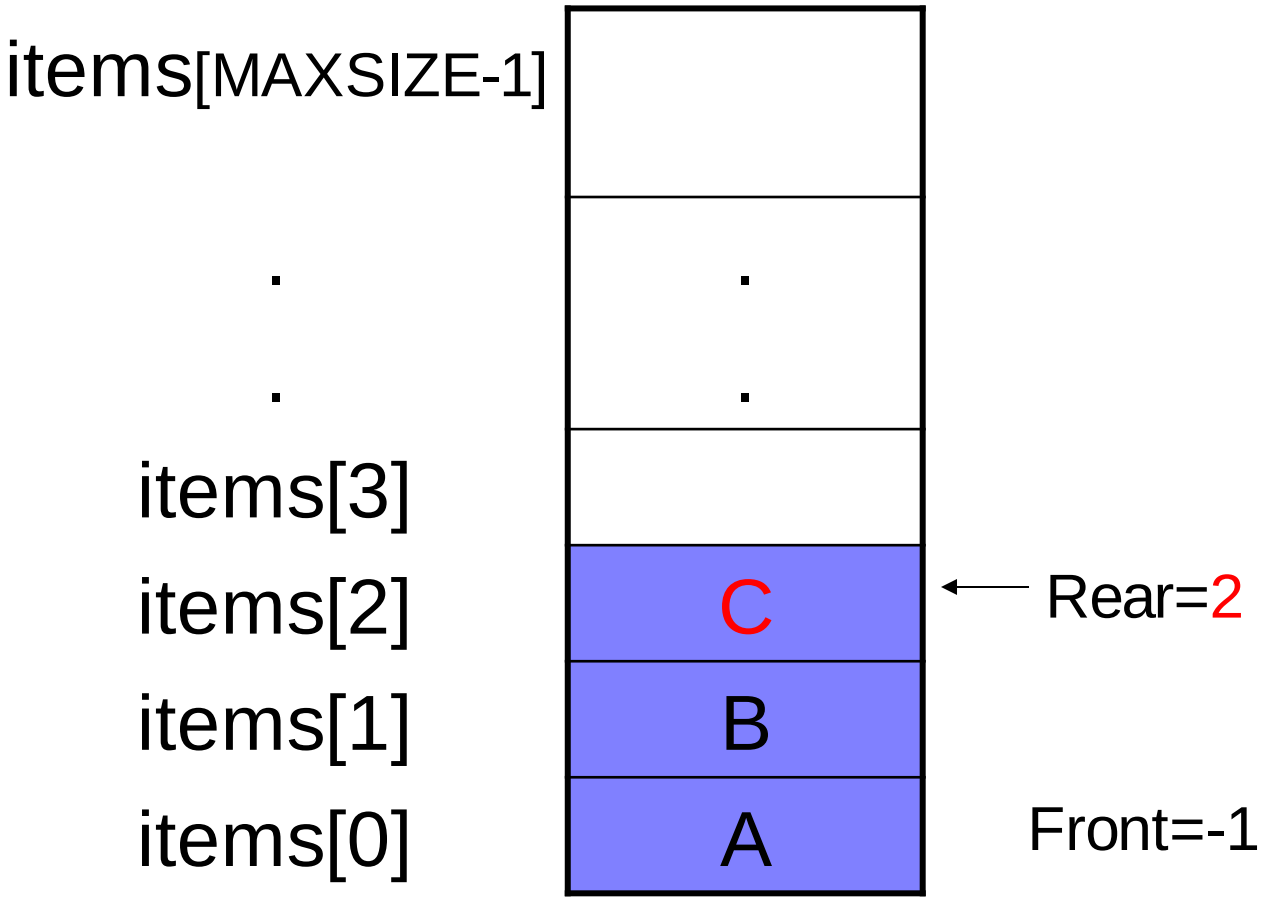
enqueue('B')

A new item (*B*) is *inserted* at the *Rear* of the queue



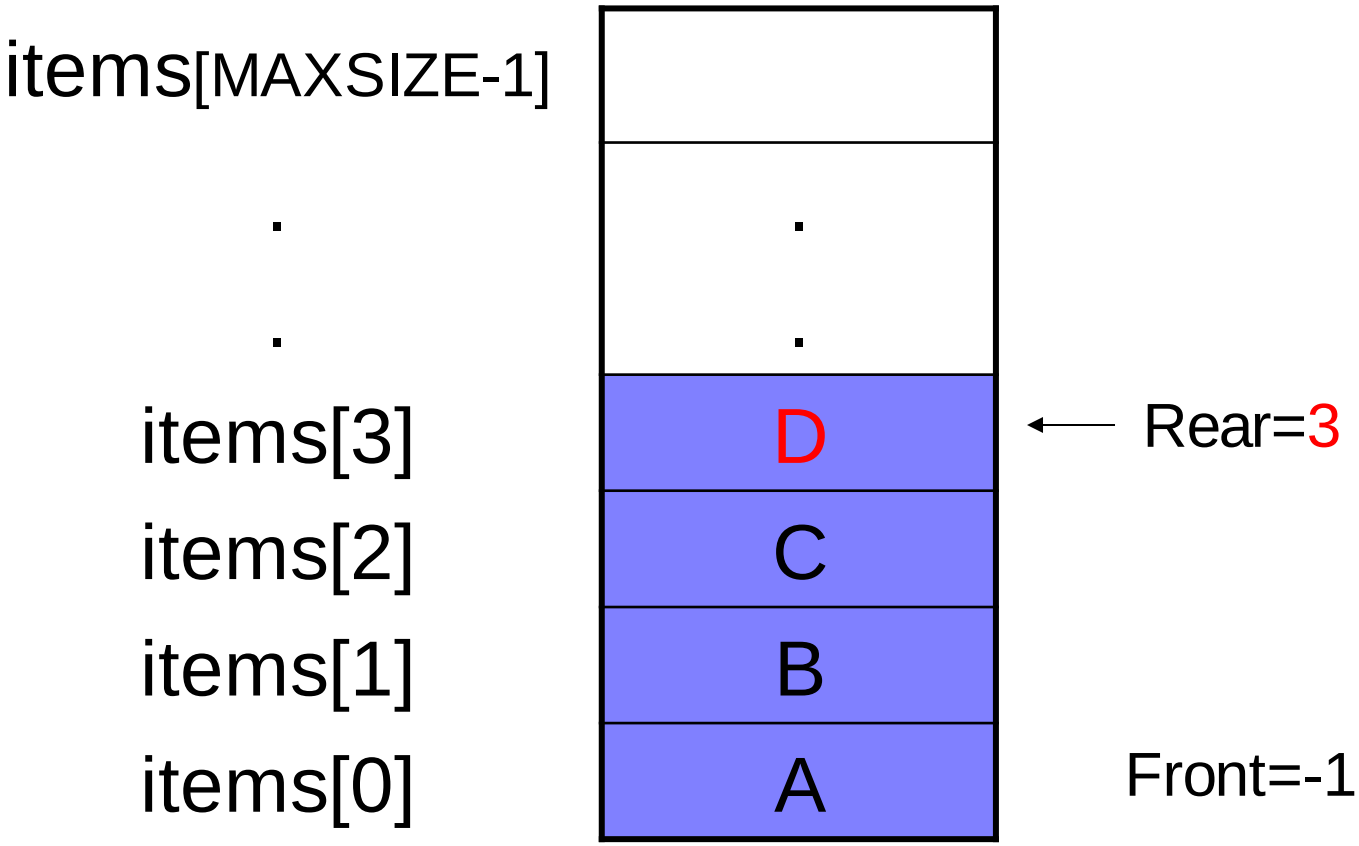
enqueue('C')

A new item (*C*) is *inserted* at the *Rear* of the queue



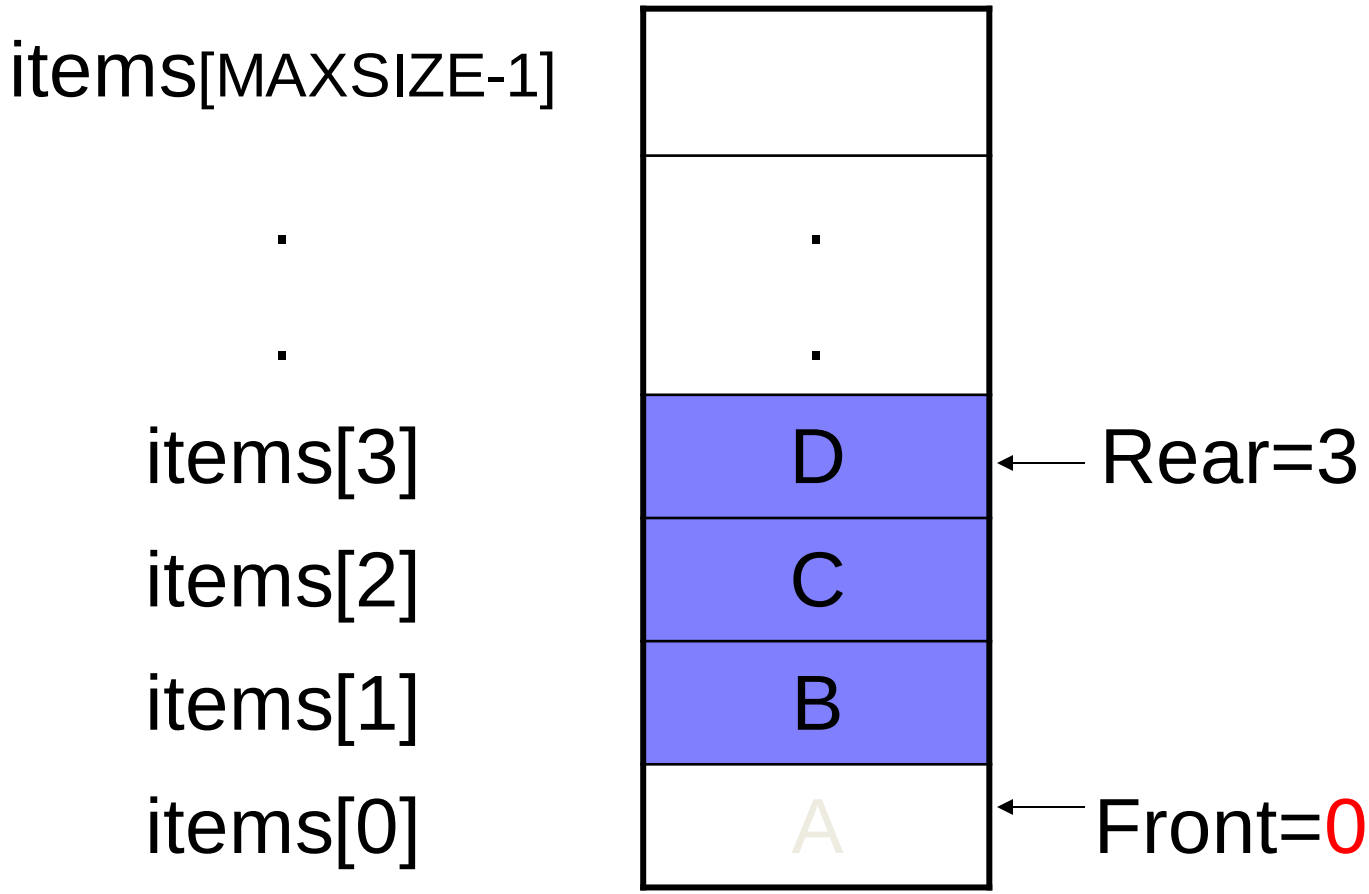
enqueue('D')

A new item (*D*) is *inserted* at the *Rear* of the queue



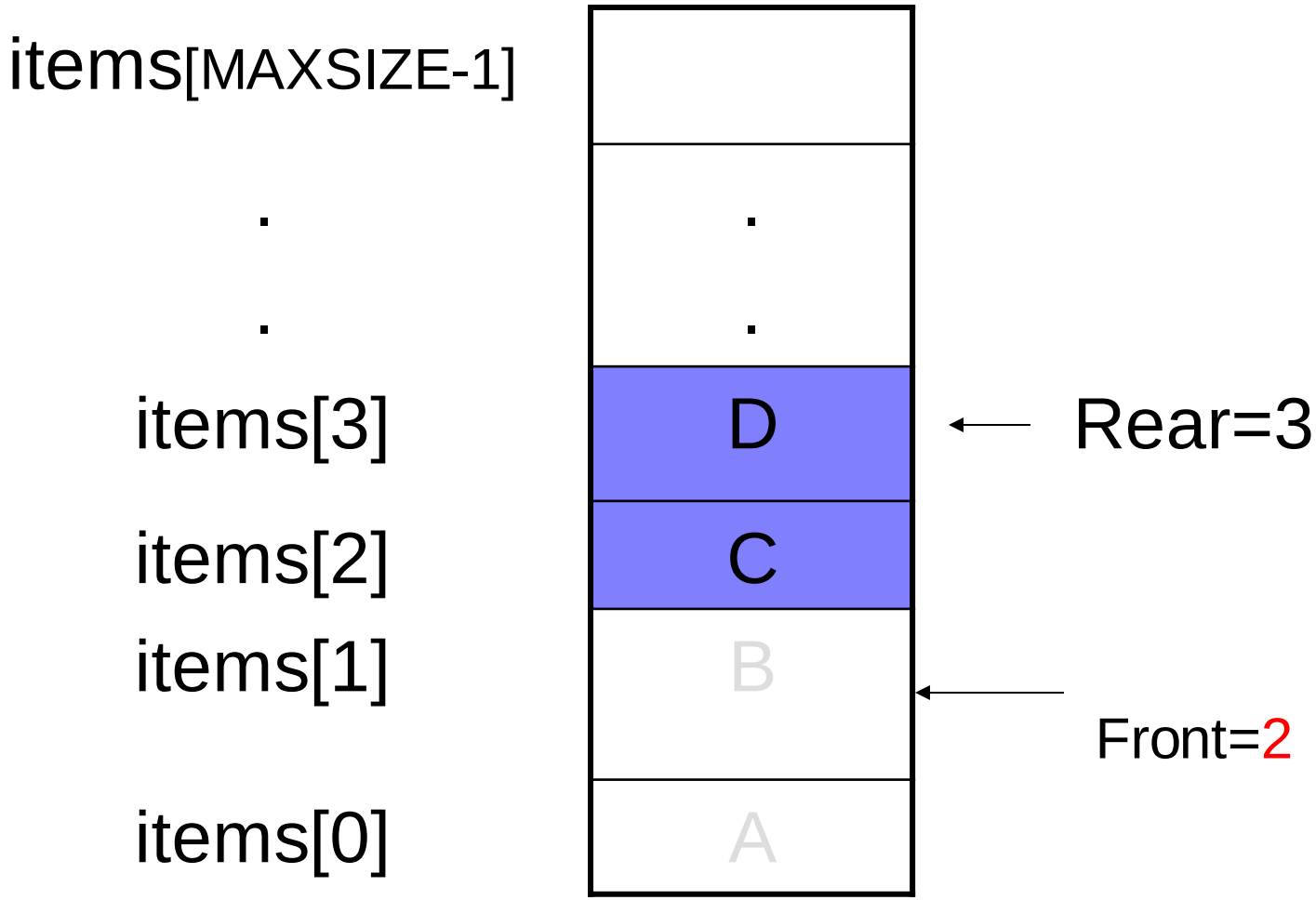
dequeue()

an item (*A*) is removed (deleted) from the *Front* of the queue



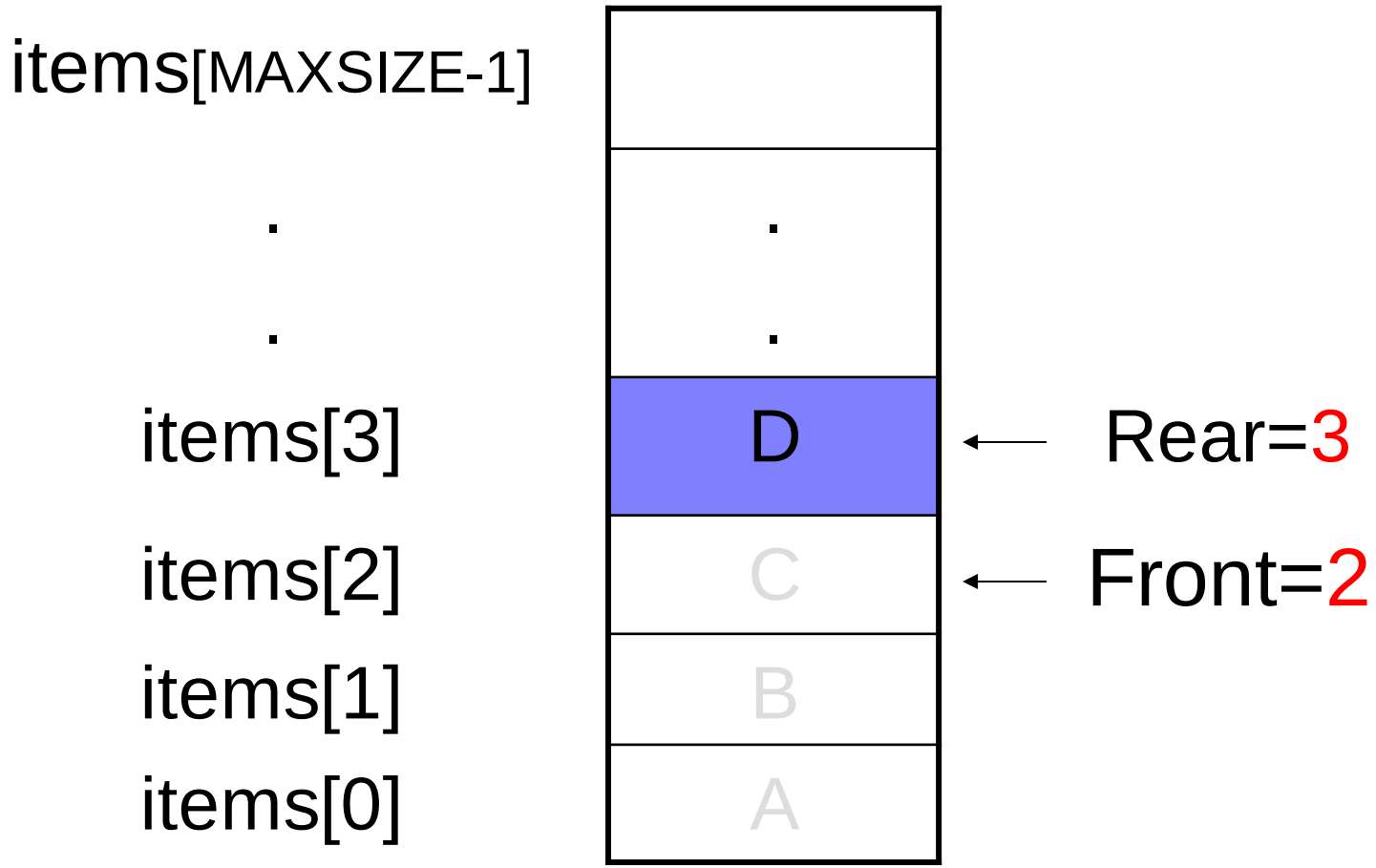
dequeue()

***Remove* two items from the front of the queue.**



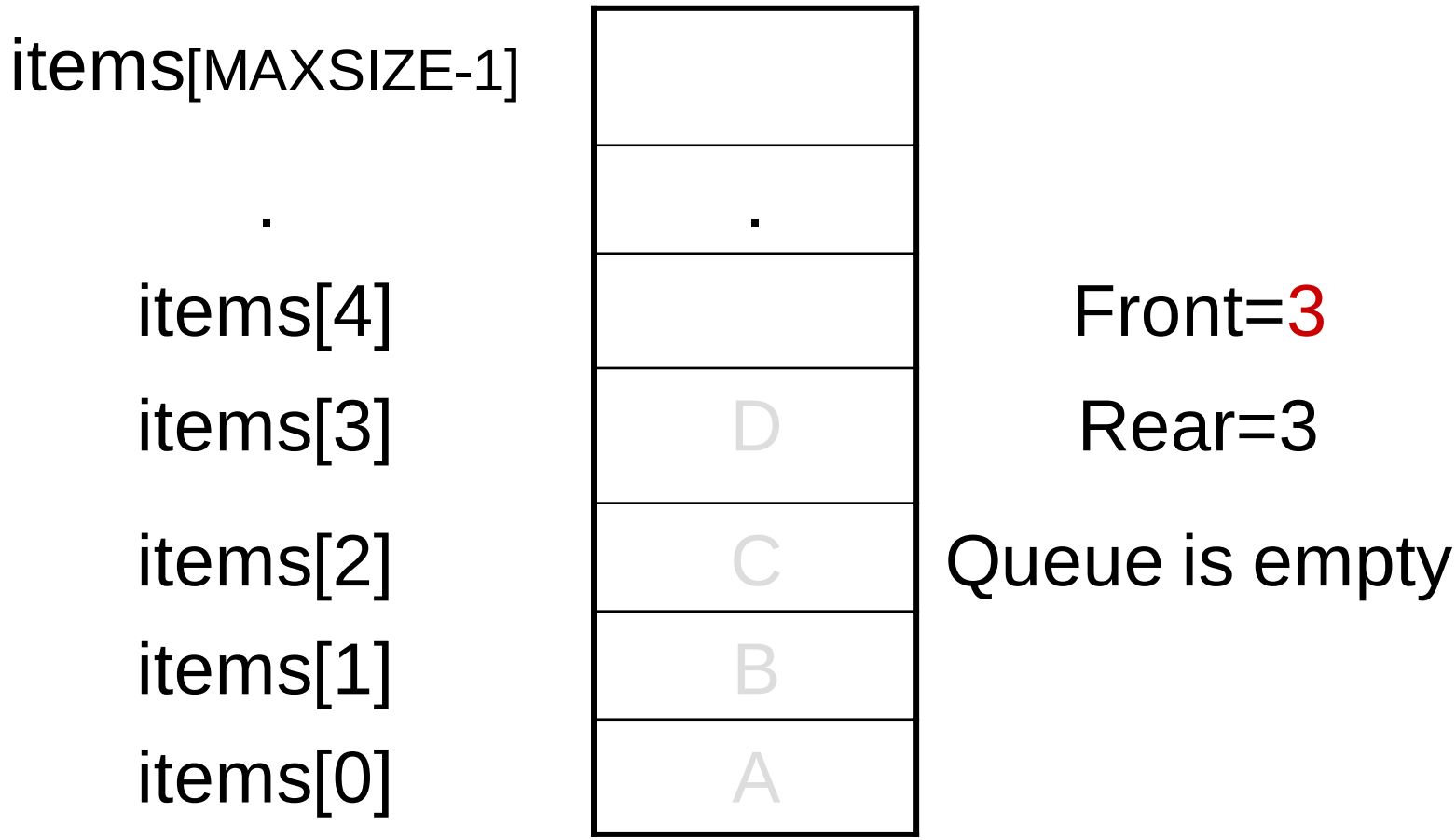
dequeue()

***Remove* two items from the front of the queue.**



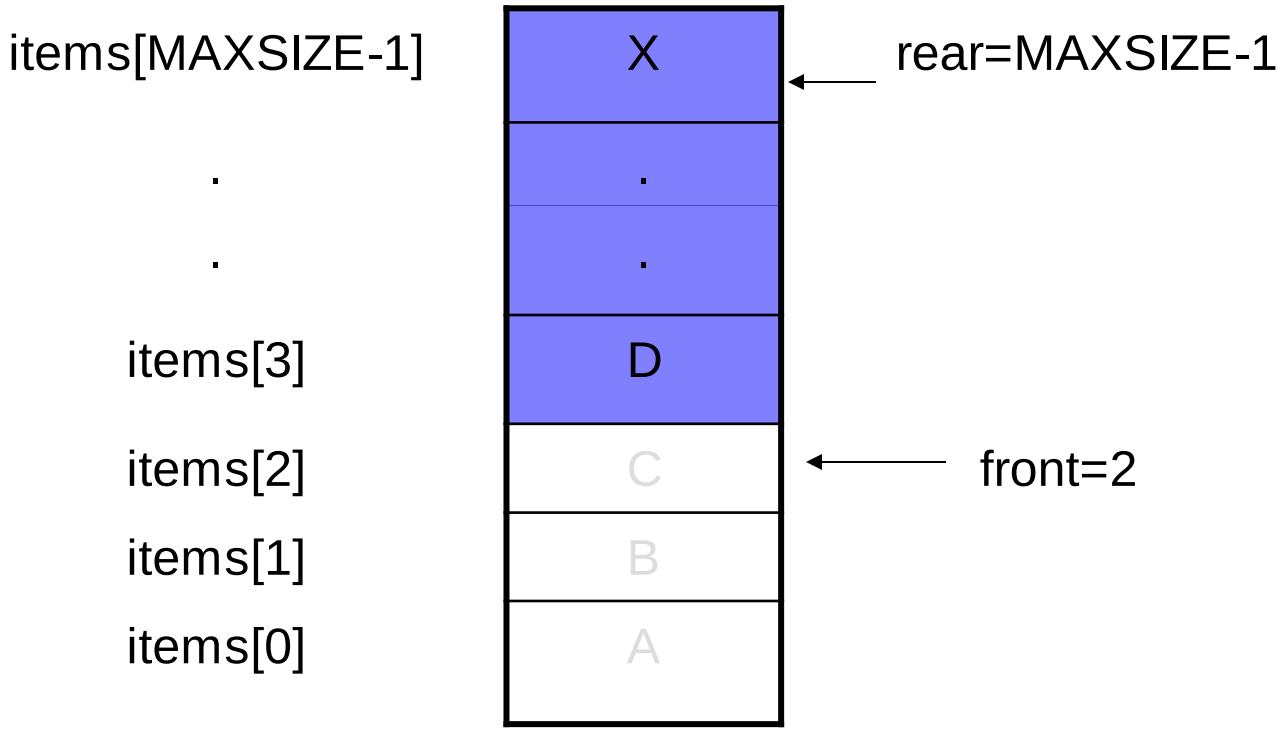
dequeue()

***Remove* one more item from the front of the queue.**



INSERT / REMOVE ITEMS

***Assume that* the *rear*= MAXSIZE-1**



- **What happens if we want to insert a new item into the queue?**

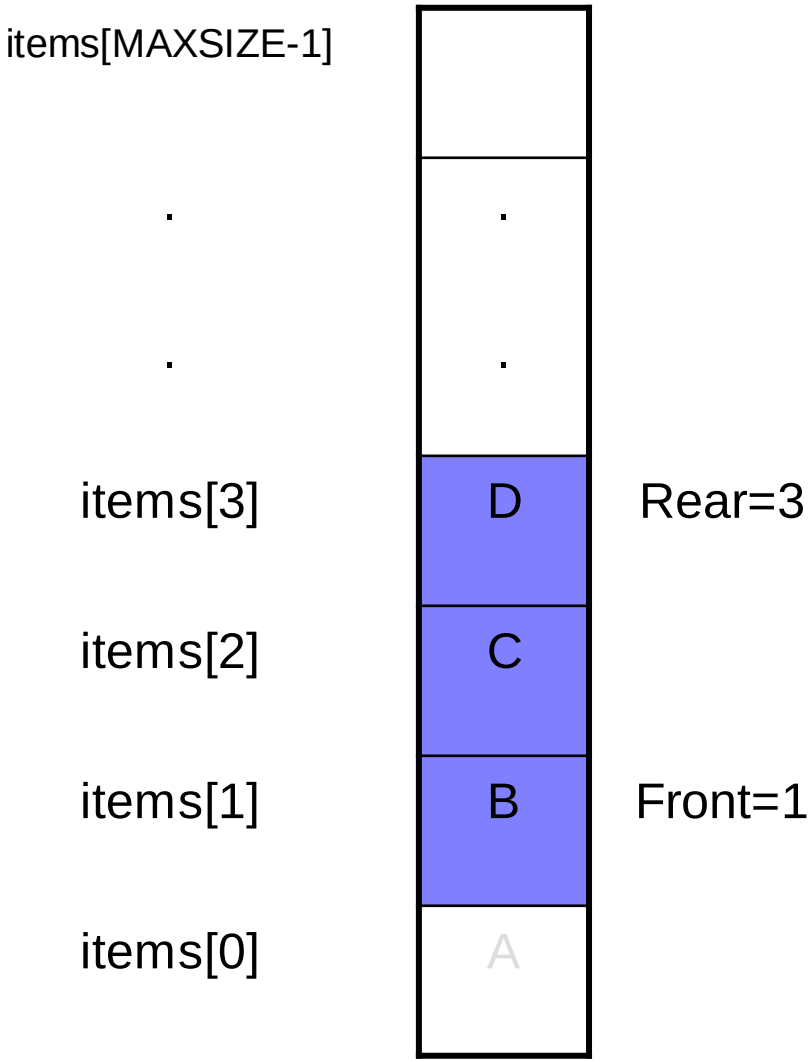
INSERT / REMOVE ITEMS

What happens if we want to insert a new item F into the queue?

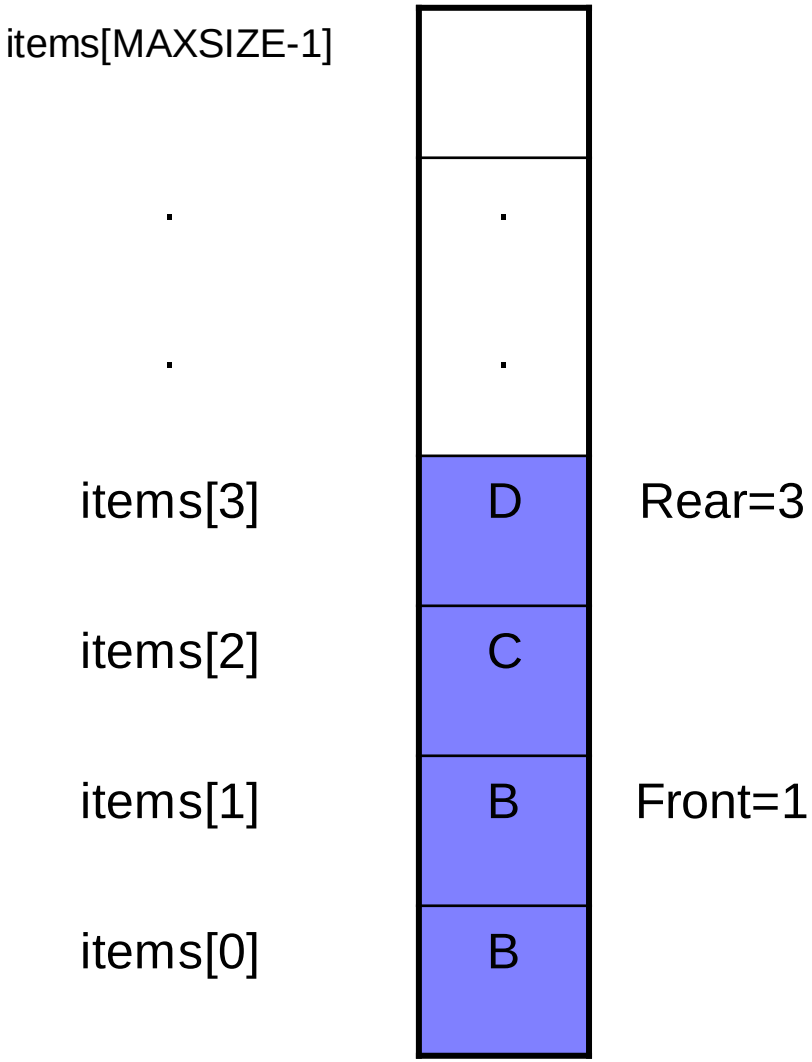
Although there is some empty space, the queue is full,since rear is at MAXSIZE-1

One of the methods to overcome this problem is to shift all the items to occupy the location of deleted item .

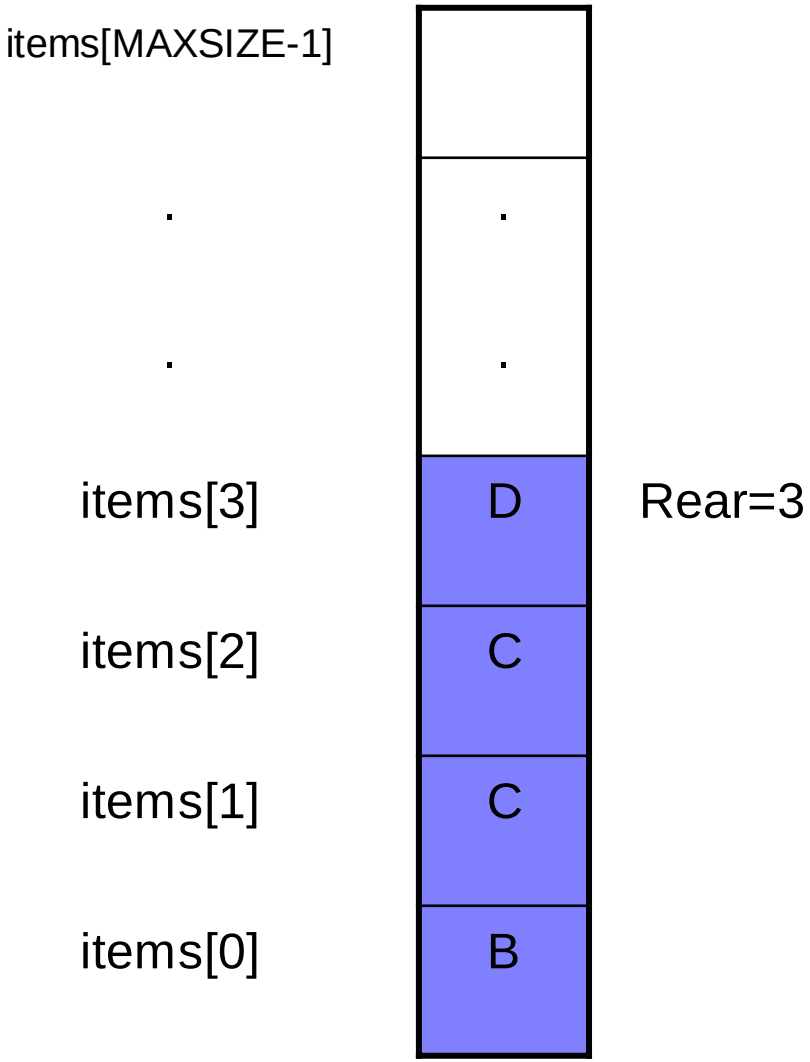
REMOVE ITEM


items[MAXSIZE-1]
items[3]
D
Rear=3
items[2]
C
items[1]
B
Front=1
items[0]
A

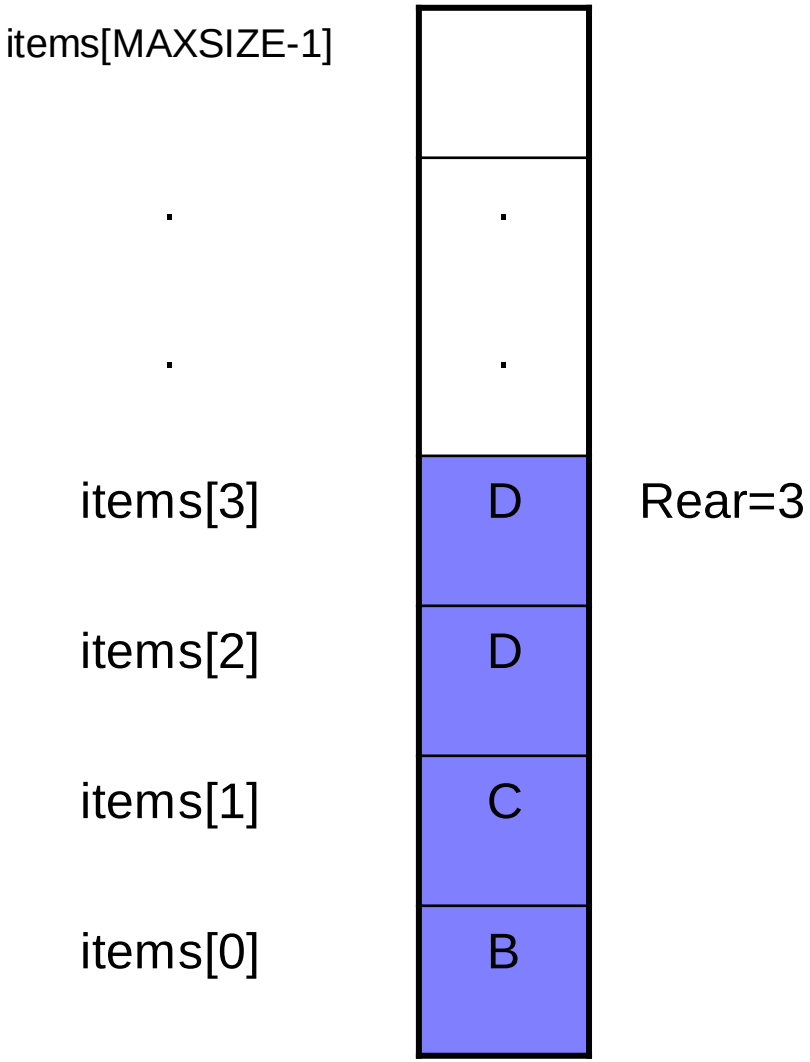
REMOVE ITEM


items[MAXSIZE-1]
items[3]
D
Rear=3
items[2]
C
items[1]
B
Front=1
items[0]
B

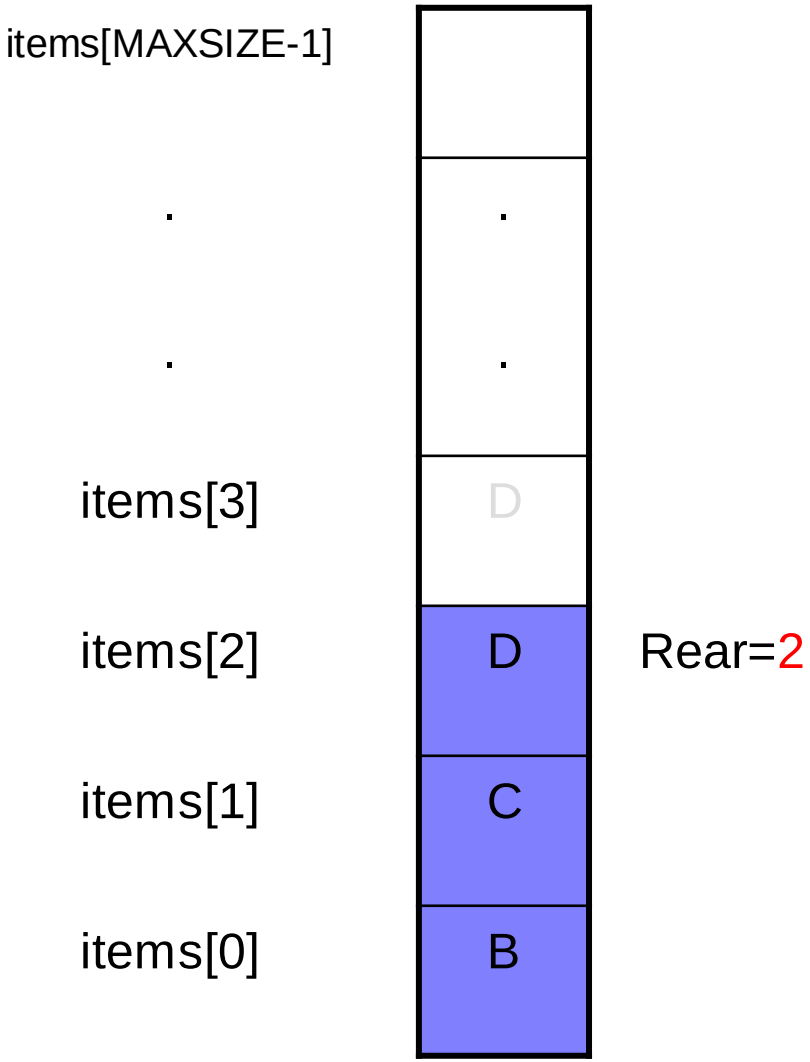
REMOVE ITEM



REMOVE ITEM



REMOVE ITEM



INSERT / REMOVE ITEMS

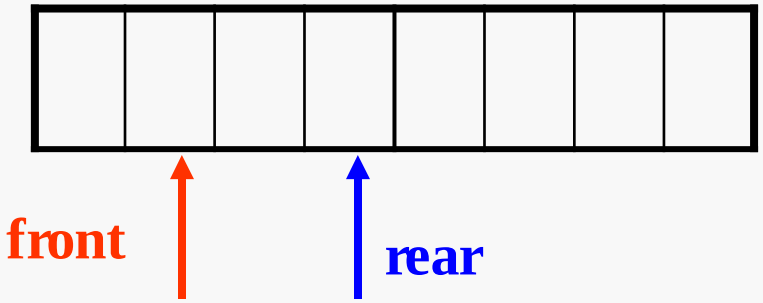
Since all the items in the queue are required to shift when an item is deleted, this method is not preferred as it needs more time.

The other method is ***circular queue.***

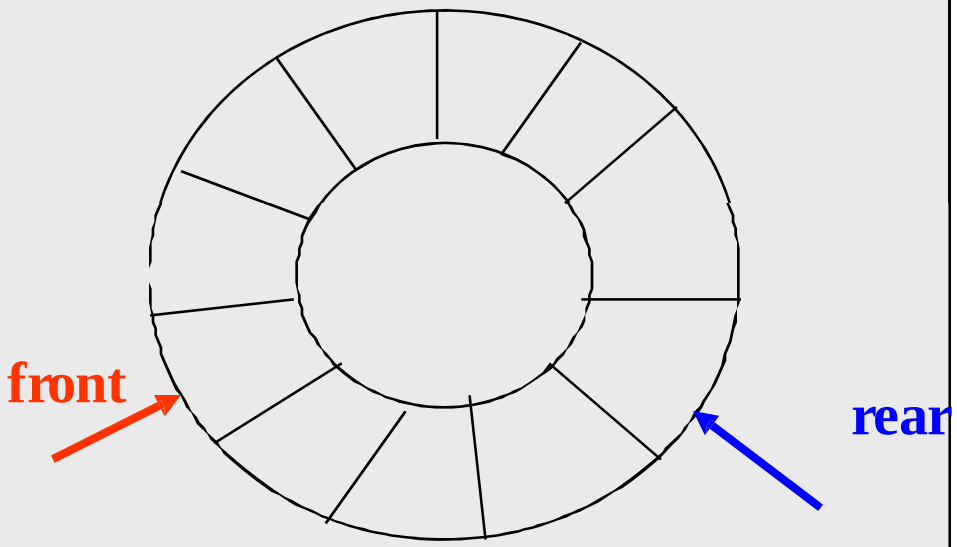
When rear = MAXSIZE-1, the next element is entered at items[0] in case that spot is free.

Possible Implementations

Linear Arrays:



Circular Arrays:

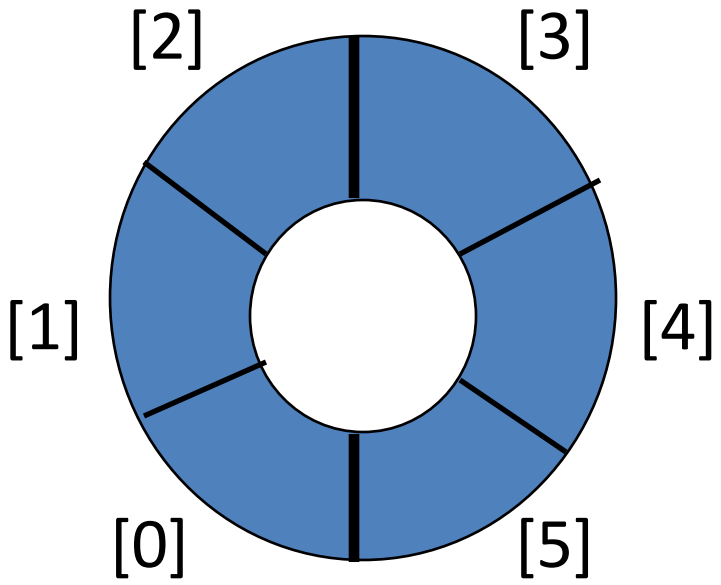


Can be implemented by a 1-d array using modulus operations

Custom Array Queue

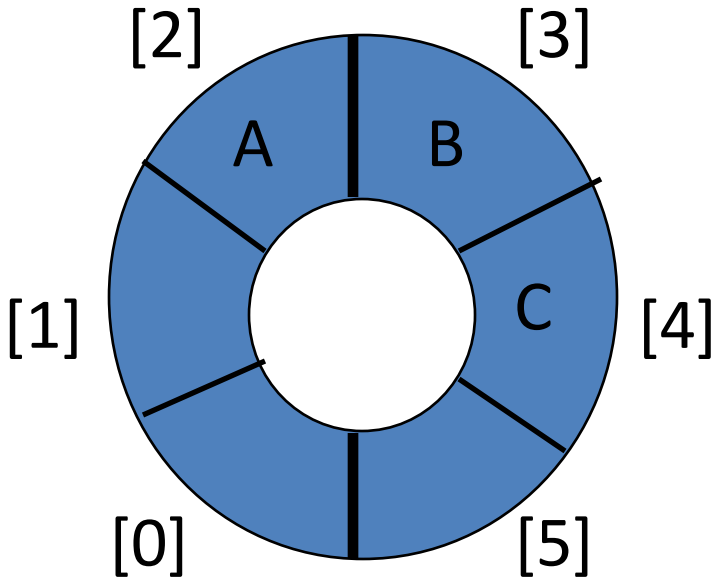
queue[]

- Circular view of array.



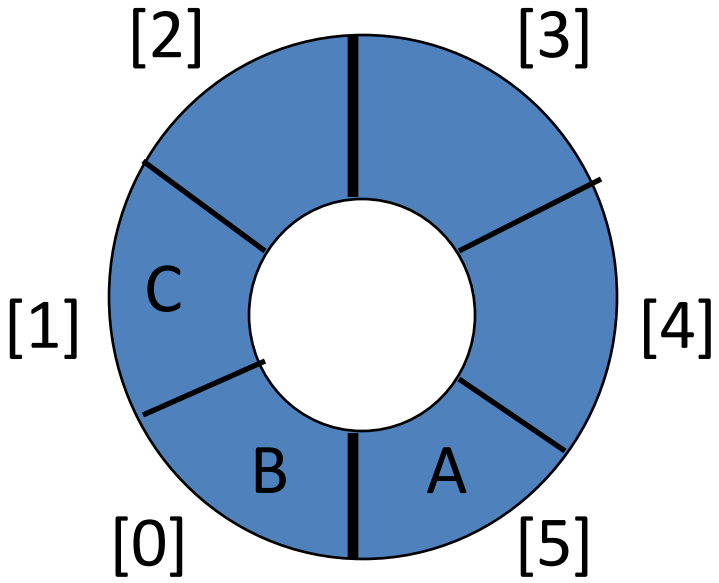
Circular Queue

- Possible configuration with 3 elements.



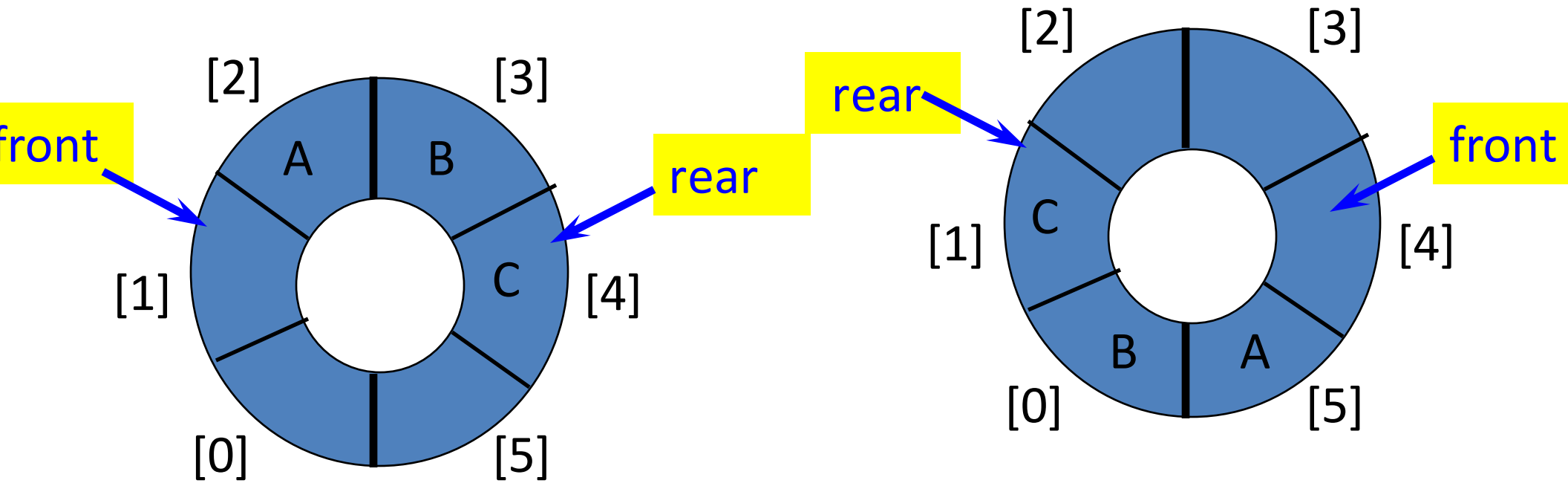
Circular Queue

- Another possible configuration with 3 elements.



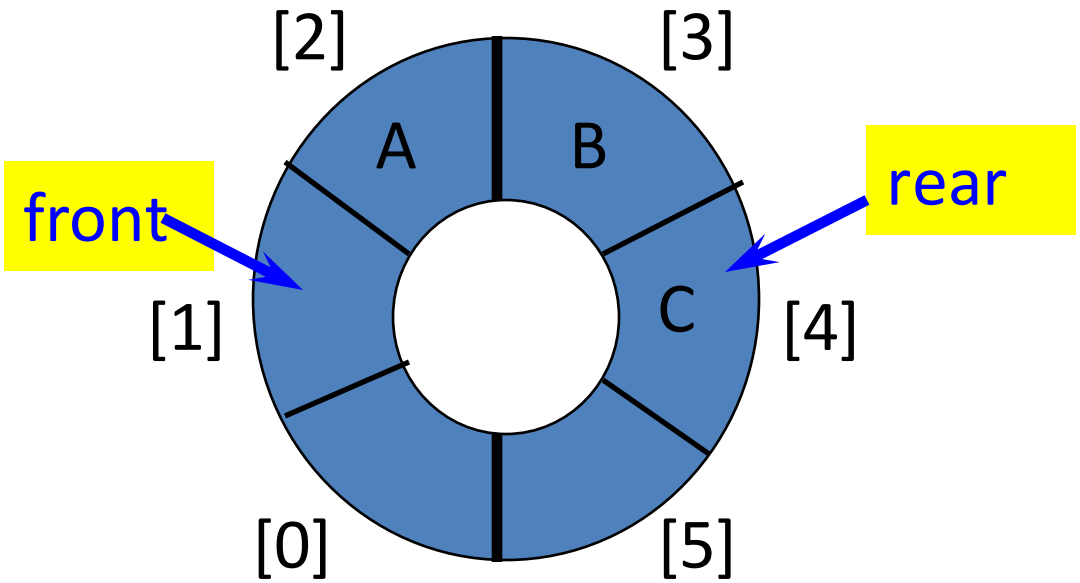
Custom Array Queue

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



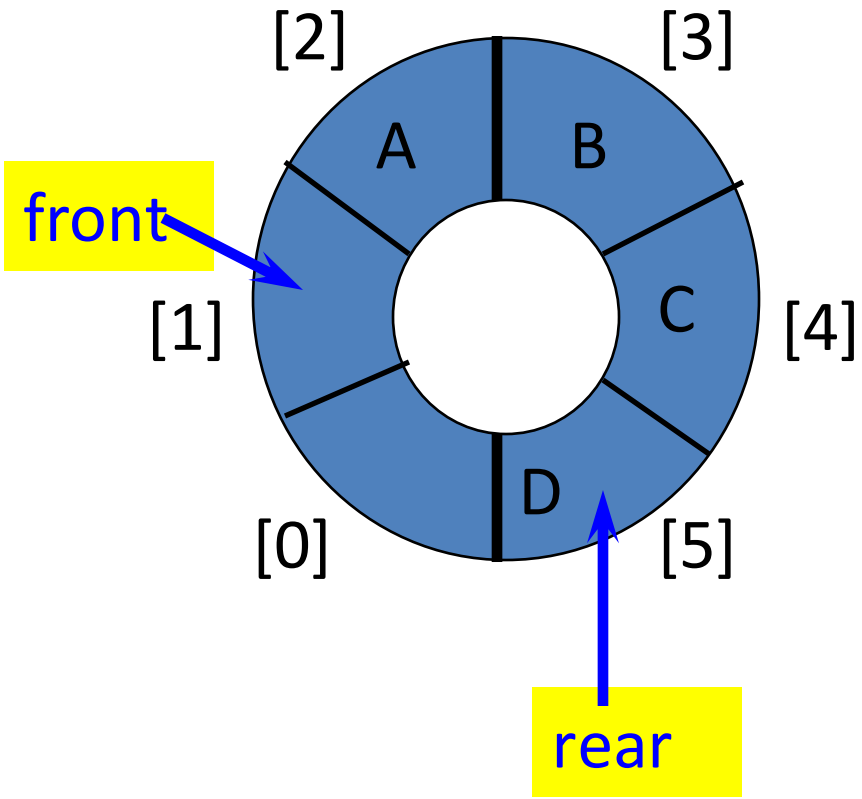
Push An Element

- Move rear one clockwise.



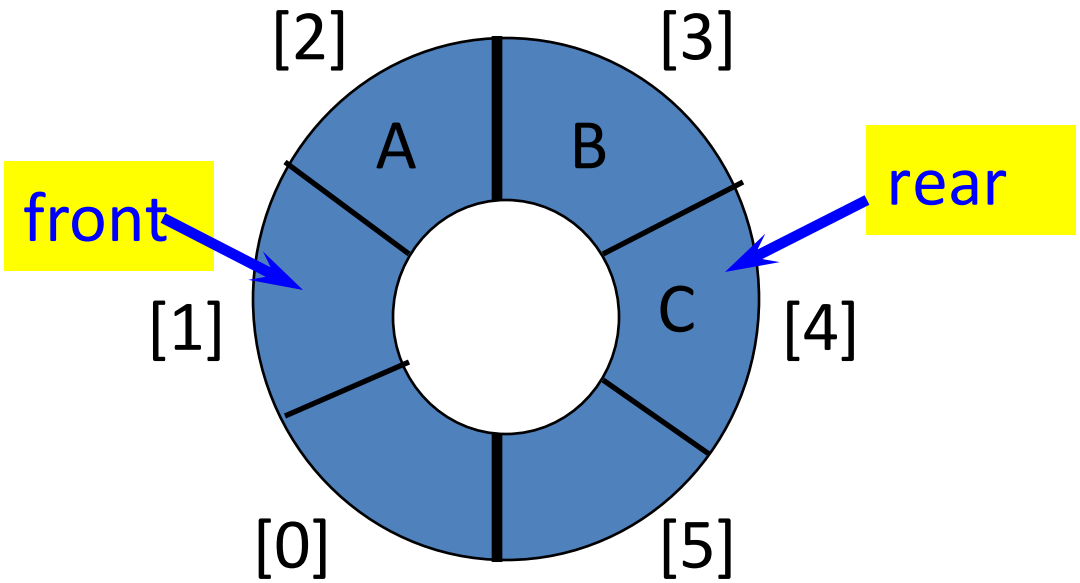
Push An Element

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



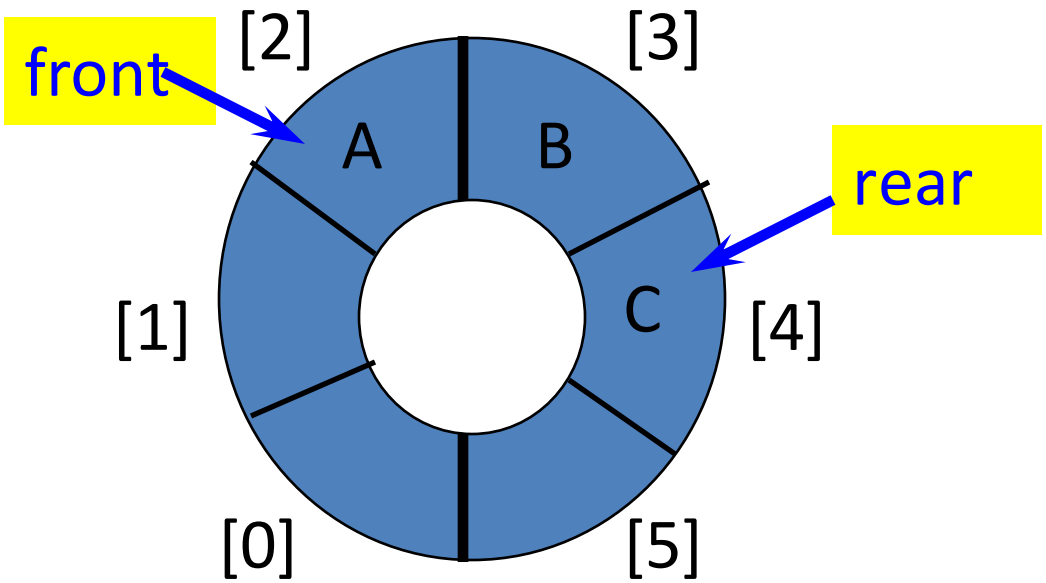
Pop An Element

- Move front one clockwise.



Pop An Element

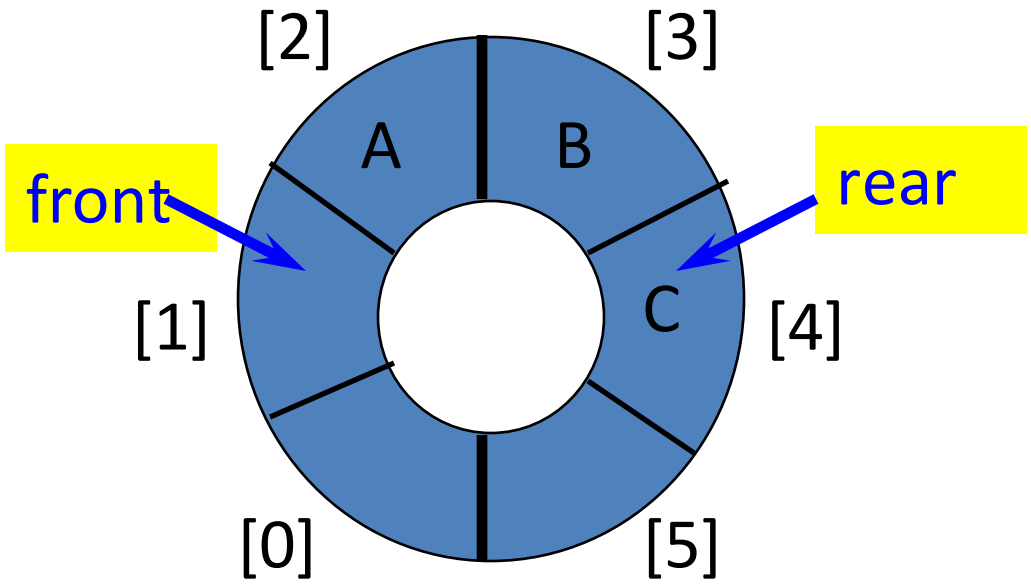
- Move front one clockwise.
- Then extract from queue[front].



Moving rear Clockwise

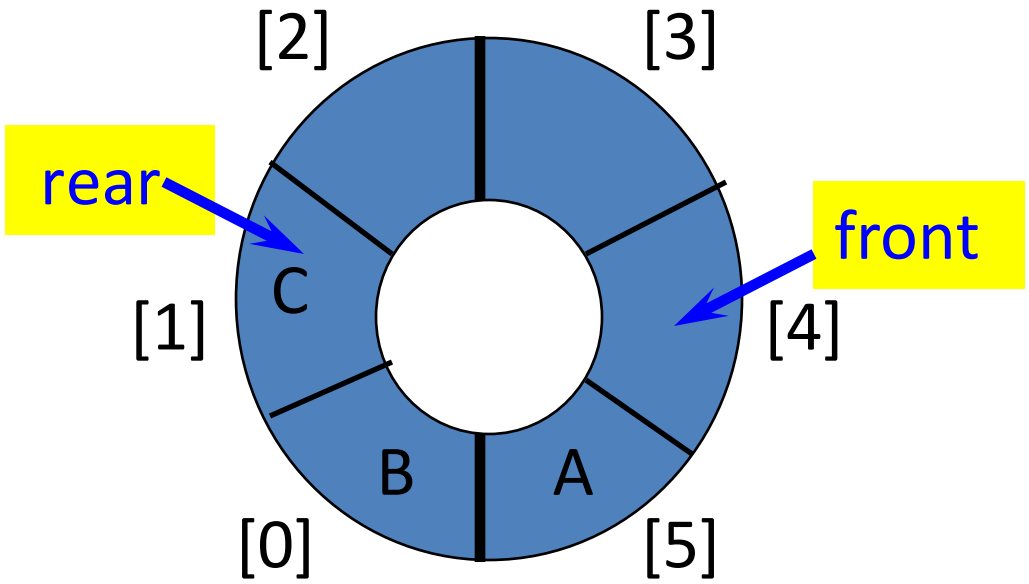
- rear++;

  if (rear = = capacity) rear = 0;

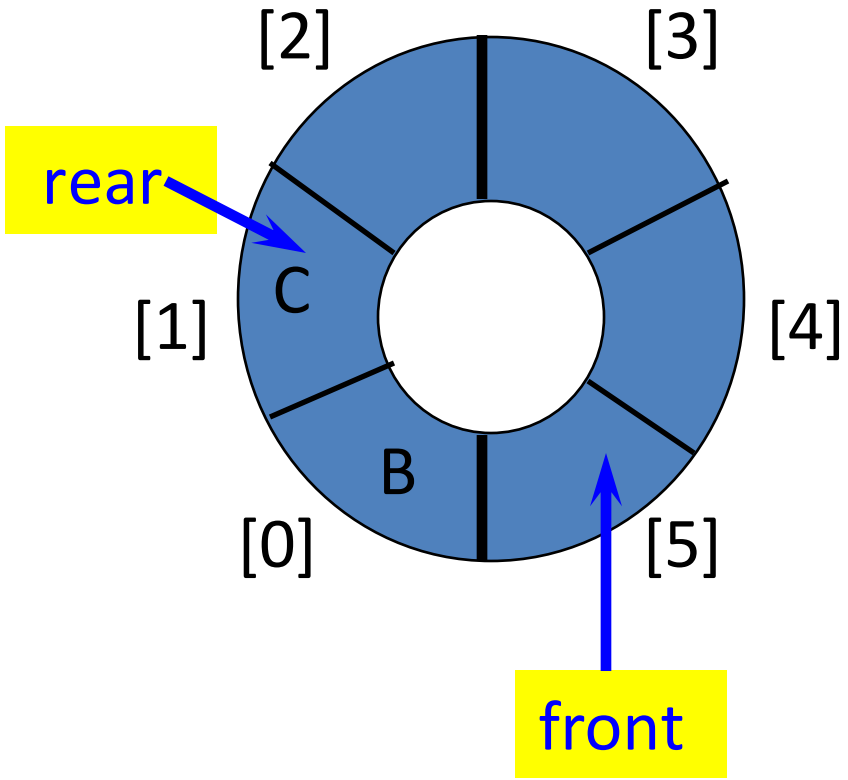


- rear = (rear + 1) % capacity;

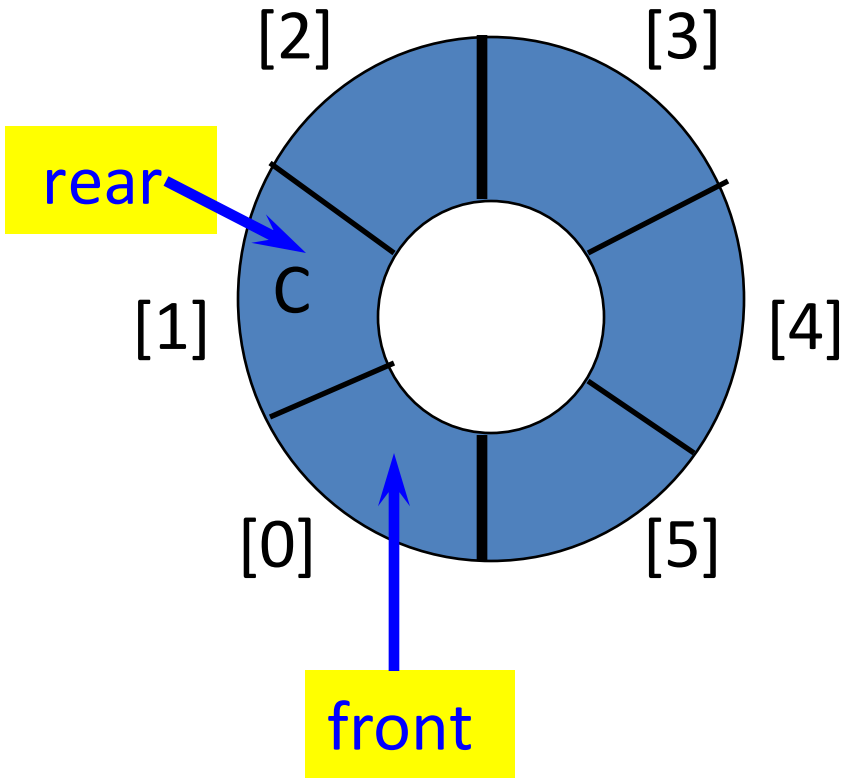
Empty That Queue



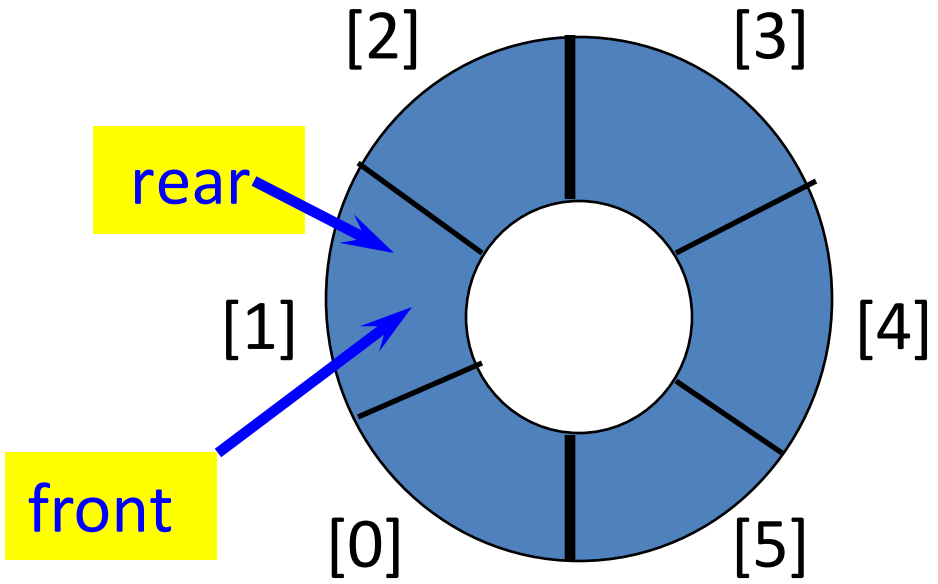
Empty That Queue



Empty That Queue

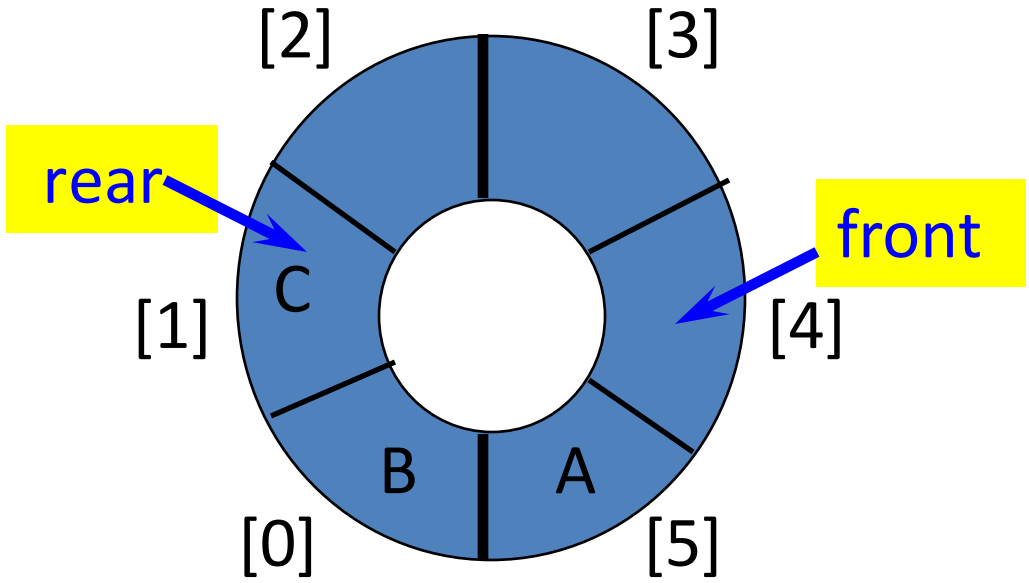


Empty That Queue

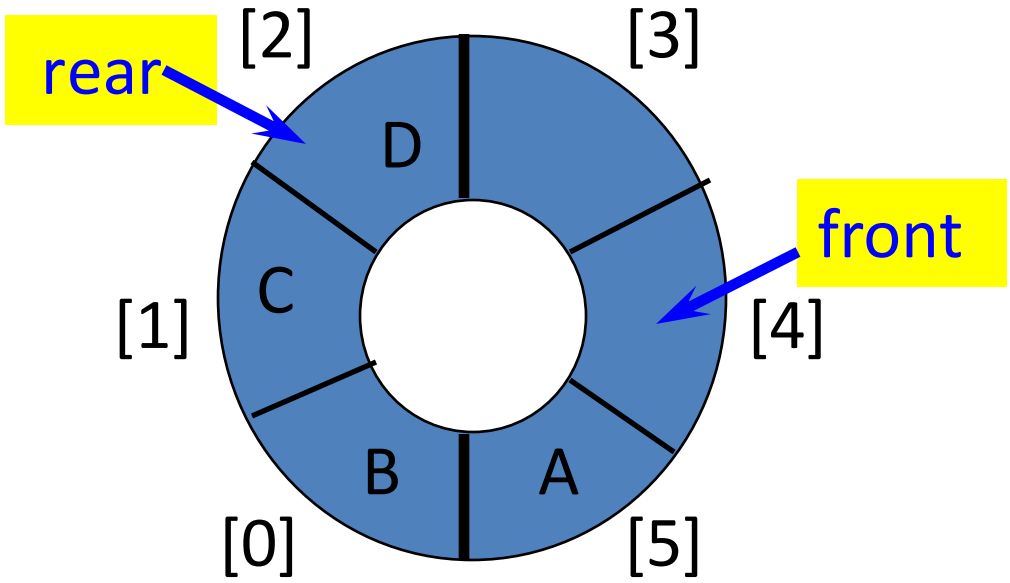


When a series of removes causes the queue to become empty, front = rear.
When a queue is constructed, it is empty.
So initialize front = rear = 0.

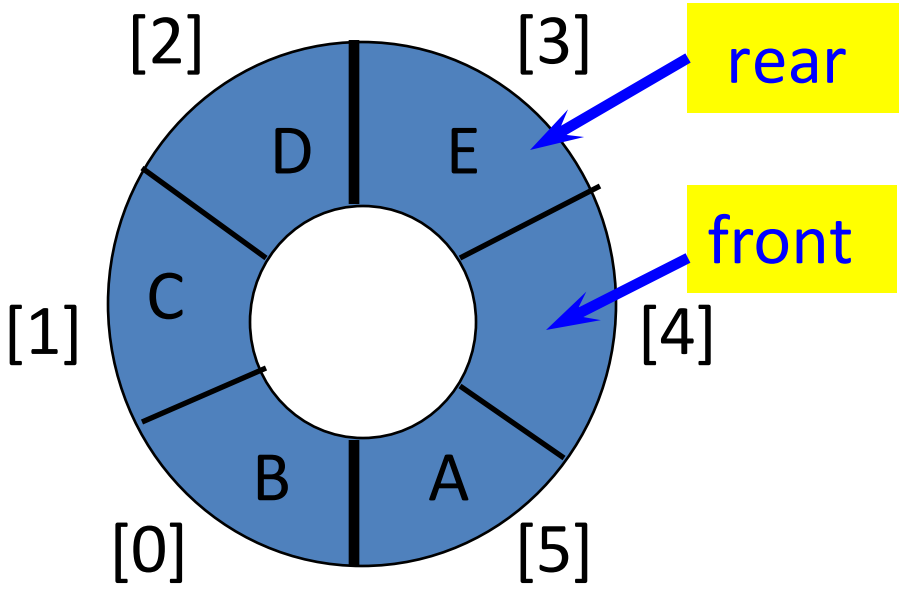
A Full Tank Please



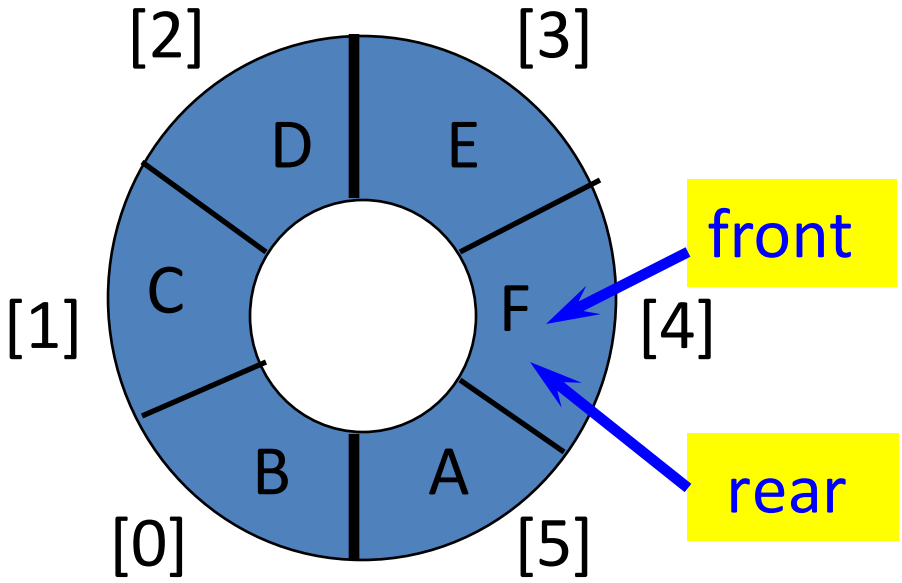
A Full Tank Please



A Full Tank Please

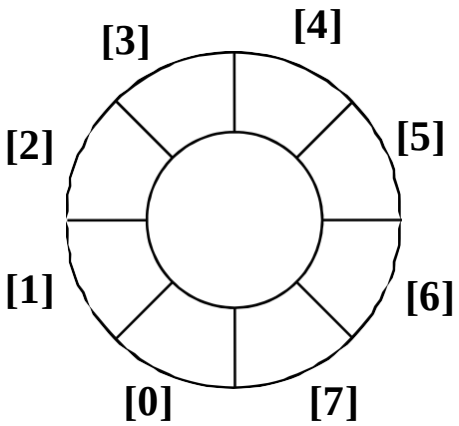


A Full Tank Please



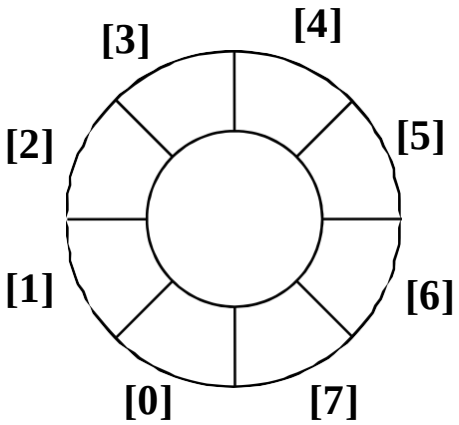
- When a series of adds causes the queue to become full, front = rear.
- So we cannot distinguish between a full queue and an empty queue!

Circular Queue

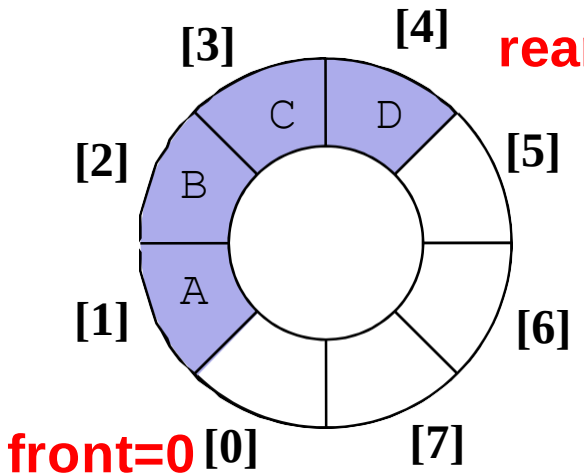


front=0
rear=0

Circular Queue: examples

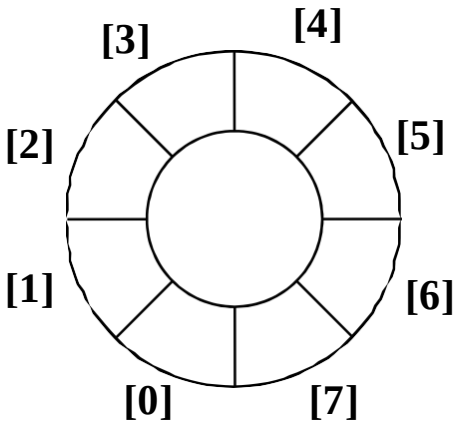


front=0
rear=0

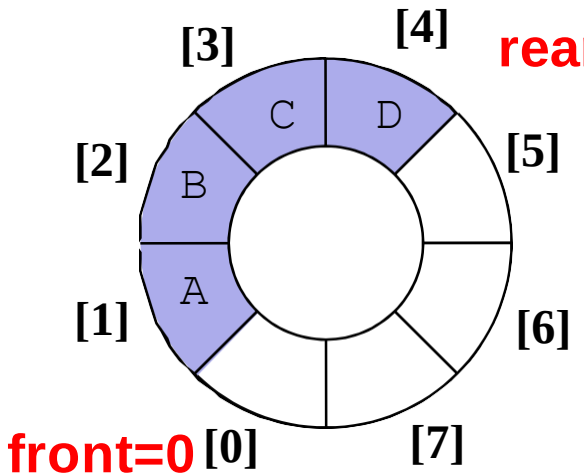


After insertion of A, B, C, D

Circular Queue

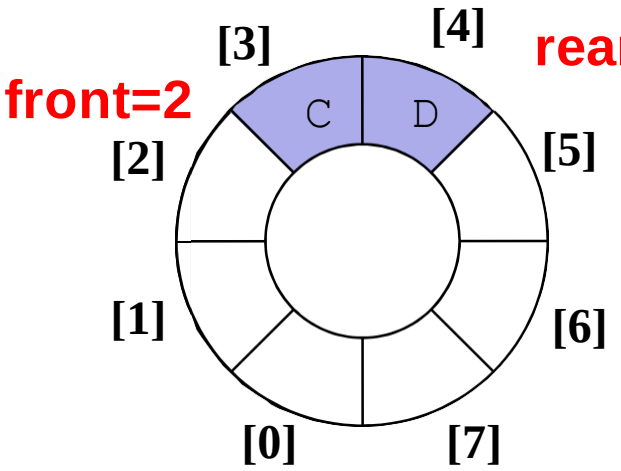


front=0
rear=0



rear = 4

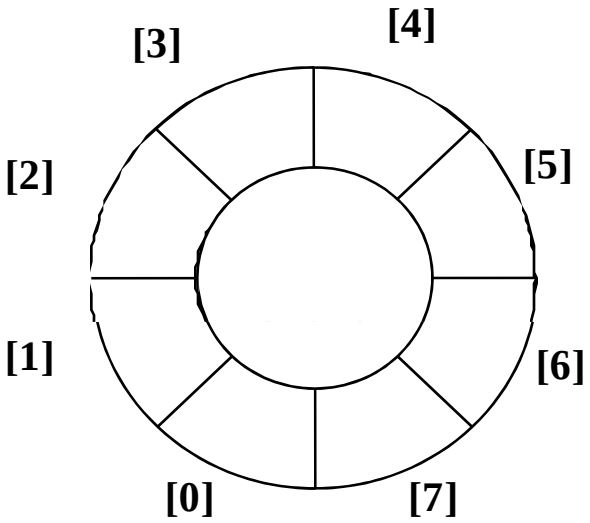
After insertion of A, B, C, D



rear = 4

After deletion of of A, B

front: index of queue-head (always empty – why?)
rear: index of last element, unless rear = front

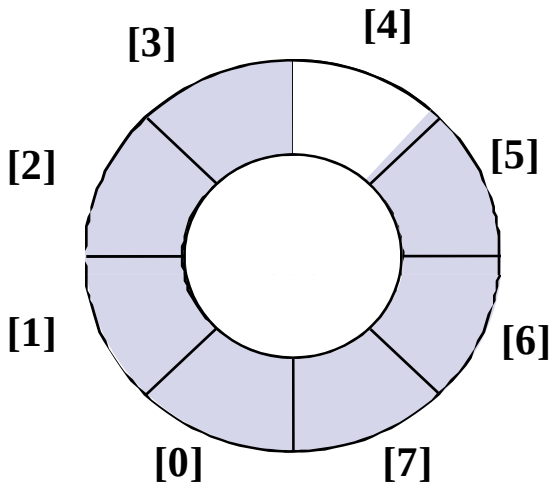


front=0
rear=0

Queue Empty

rear = 3

front=4



Queue Full

Queue Empty Condition: ***front == rear***
Queue Full Condition: ***front == (rear + 1) % MAX_Q_SIZE***

DEQUE(Double Ended Queue)

A deque is one where
Insertion and deletion can be made from both ends , front and rear
Operations we perform on this queue are-

1. insertat_front(x)
2. Insertat_rear(x)
3. deletefrombeg()
4. deletefromend()

Input Restricted DEQUE(Double Ended Queue)

An input-restricted deque is one where deletion can be made from both ends , front and rear
but insertion can be made at one end only i.e. at rear end
Operations we perform on this queue are-

1. insertat_end(x)
2. deletefrombeg()
3. deletefromend()

Output Restricted DEQUE

An output-restricted deque is one where insertions can be made from both ends , front and rear
but deletion can be made at one end only i.e. from front end
Operations we perform on this queue are-

1. insertat_beg(x)
2. insertat_end(x)
3. deletefrombeg()

Applications of queues

Task Scheduling: Queues can be used to schedule tasks based on priority or the order in which they were received.

Resource Allocation: Queues can be used to manage and allocate resources, such as printers or CPU processing time.

Batch Processing: Queues can be used to handle batch processing jobs, such as data analysis or image rendering.

Message Buffering: Queues can be used to buffer messages in communication systems, such as message queues in messaging systems or buffers in computer networks.

Applications of Queues:

Applications of Queue in Operating systems: Semaphores

FCFS (first come first serve) scheduling, example: FIFO queue

Spooling in printers

Buffer for devices like keyboard

CPU Scheduling

Memory management

Applications of Queues:

Applications of Queue in Networks:

Queues in routers/ switches

Mail Queues

Variations: (Deque, Priority Queue, Doubly Ended Priority Queue)

Applications of Queues:

Some other applications of Queue:

Applied as waiting lists for a single shared resource like CPU, Disk, and Printer.

Applied as buffers on MP3 players and portable CD players.

Applied on Operating system to handle the interruption.

Applied to add a song at the end or to play from the front.

Applied on WhatsApp when we send messages to our friends and they don't have an internet connection then these messages are queued on the server of whatsApp.

Traffic software (Each light gets on one by one after every time of interval of time.)