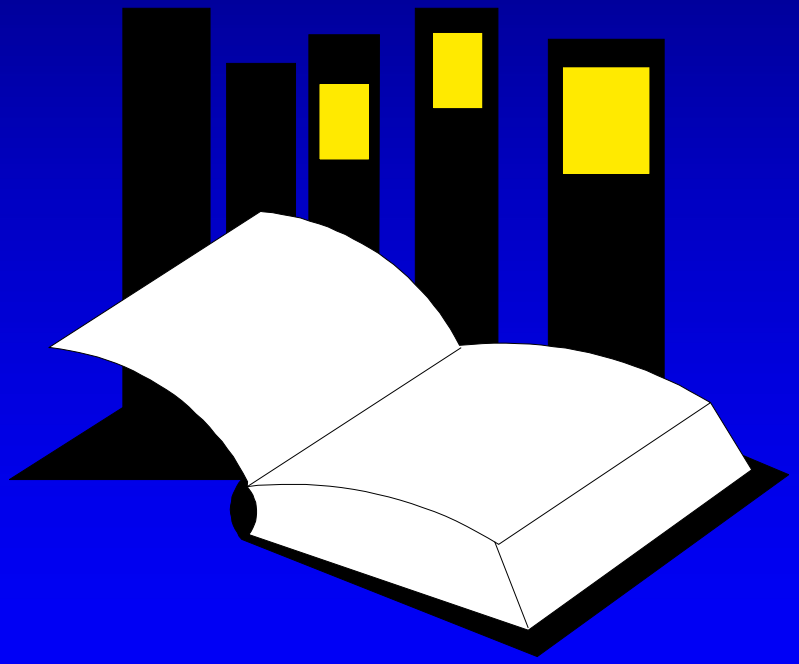
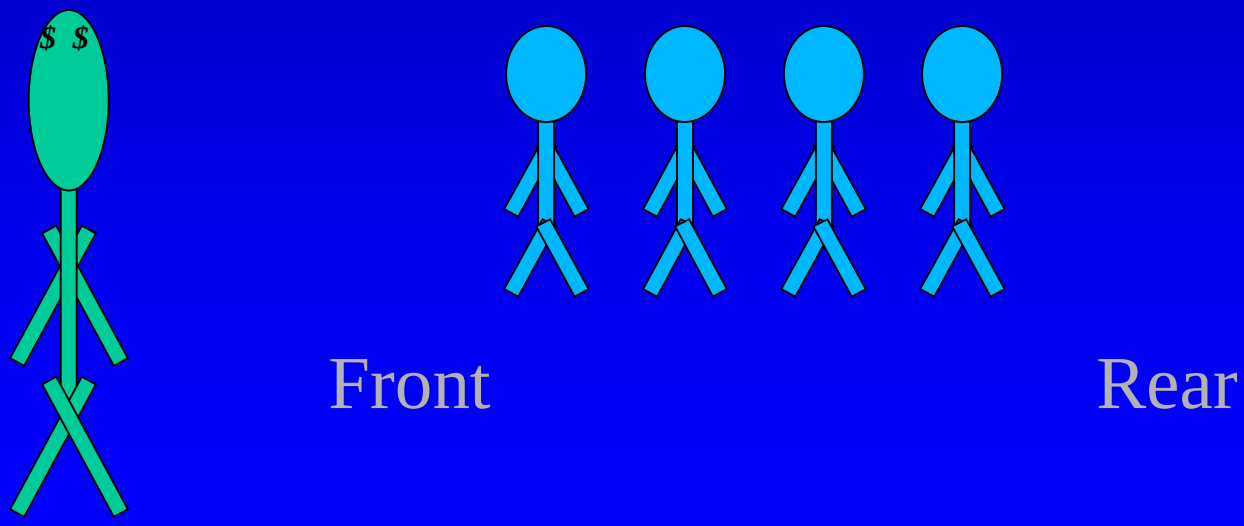
Using a Queue



- Queue is a linear list of elements
- Deletions can take place only one end-Front
- Insertions can take place at other end – Rear
- It is also called as FIFO

The Queue Operations

- A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.



Queue with elements

Front

Rear

After AAA deleted at front end

Front

Rear

After ‘EEE’ & ‘FFF’ inserted at rear

Front

Rear

Representation of Queues

Queue represented in two ways:

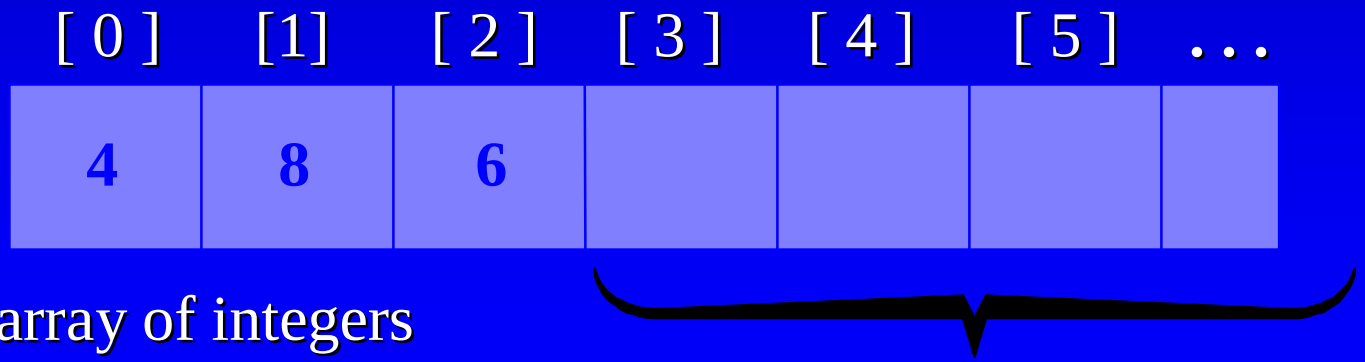
1. One way list (or) Linear Array
2. Linked representation of queues

Array Representation of Queue

- It is maintained by linear array QUEUE
- And two pointer variables FRONT & REAR
- FRONT = NULL and REAR=NULL (Empty)
- FRONT = FRONT + 1 (whenever element is deleted from queue)
- REAR = REAR + 1 (whenever element is added to the queue)
- FRONT=REAR (Only one element)

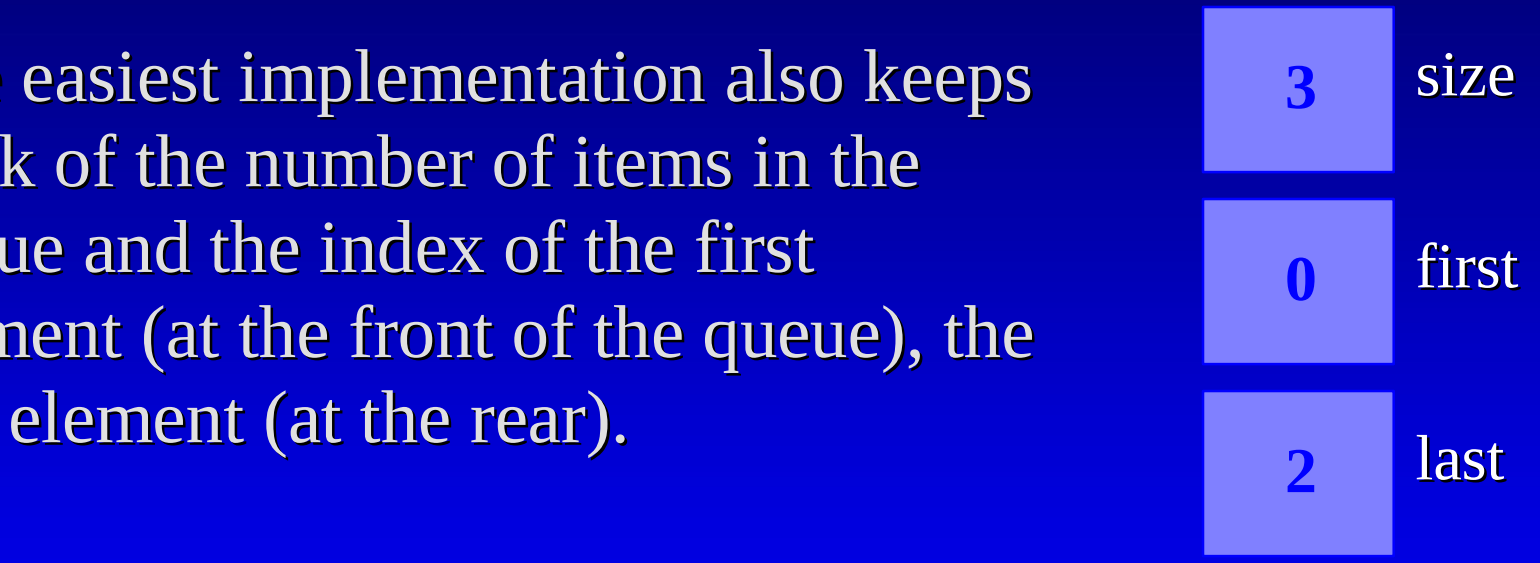
Array Implementation

- A queue can be implemented with an array, as shown here. For example, this queue contains the integers 4 (at the front), 8 and 6 (at the rear).



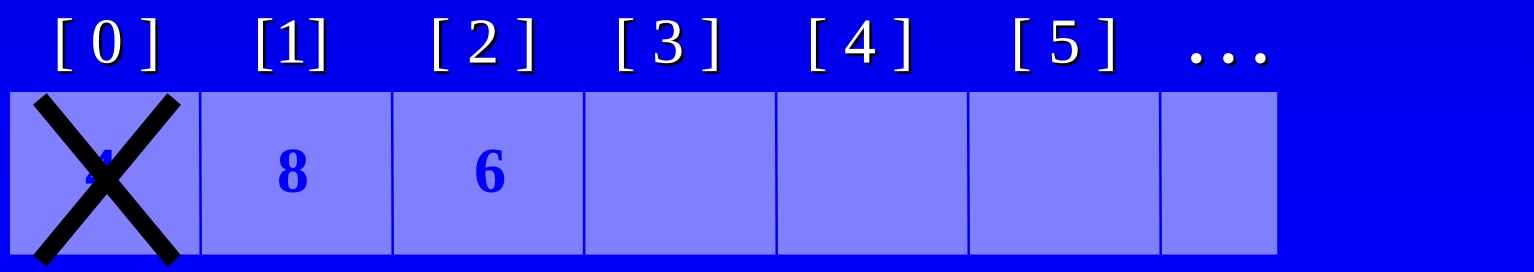
Array Implementation

- The easiest implementation also keeps track of the number of items in the queue and the index of the first element (at the front of the queue), the last element (at the rear).



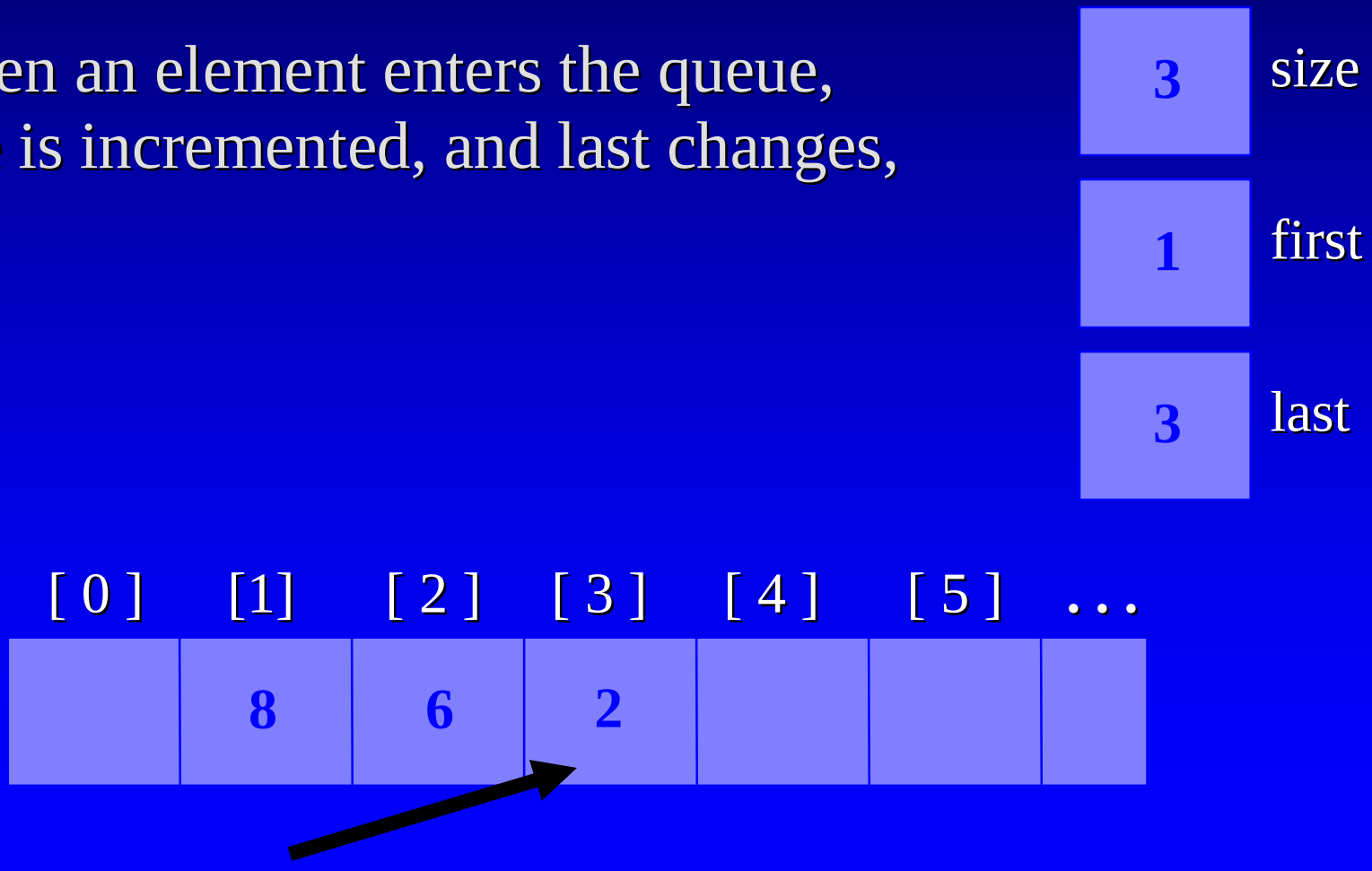
A Dequeue Operation

- When an element leaves the queue, size is decremented, and first changes, too.



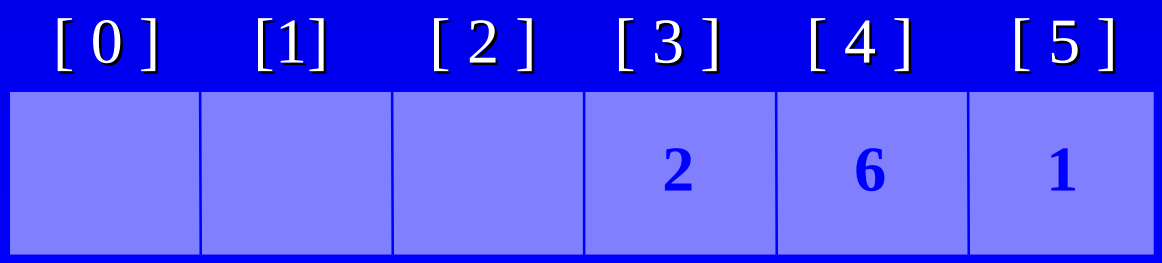
An Enqueue Operation

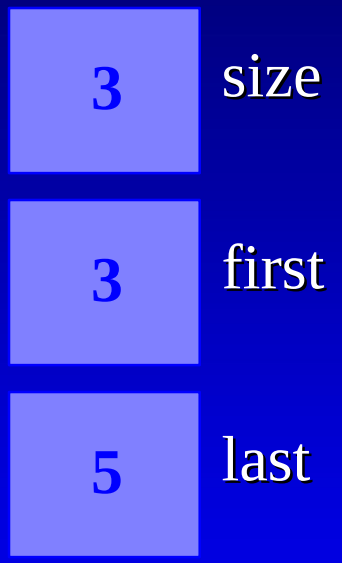
- When an element enters the queue, size is incremented, and last changes, too.



At the End of the Array

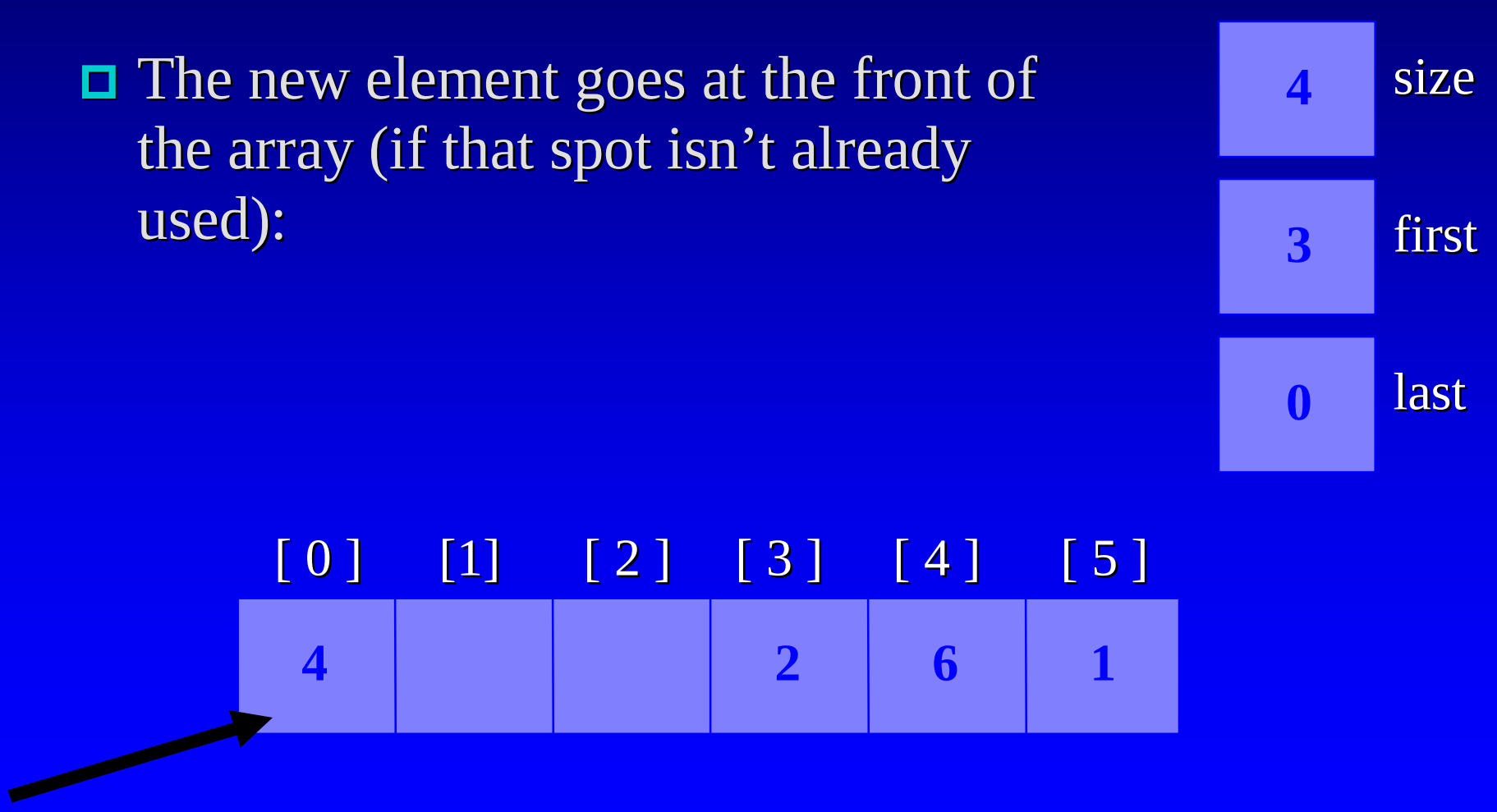
- There is special behaviour at the end of the array. For example, suppose we want to add a new element to this queue, where the last index is [5]:





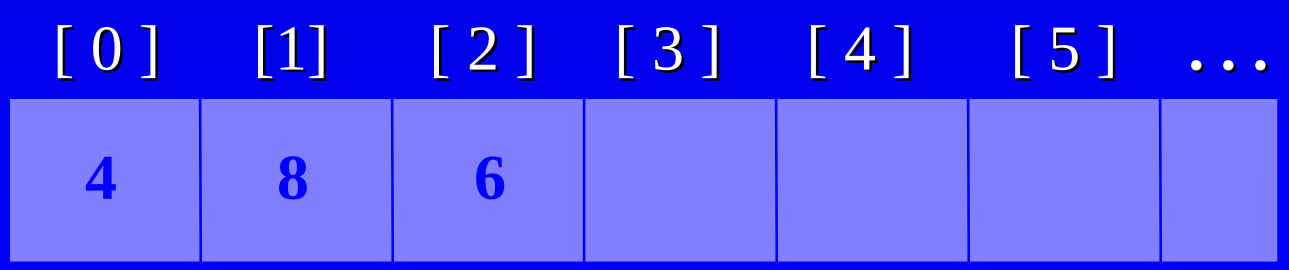
At the End of the Array

- The new element goes at the front of the array (if that spot isn't already used):



Array Implementation

- Easy to implement
- But it has a limited capacity with a fixed array
- Or you must use a dynamic array for an unbounded capacity
- Special behaviour is needed when the rear reaches the end of the array.



Queue Insert

QINSERT(QUEUE, N, FRONT, REAR, ITEM)

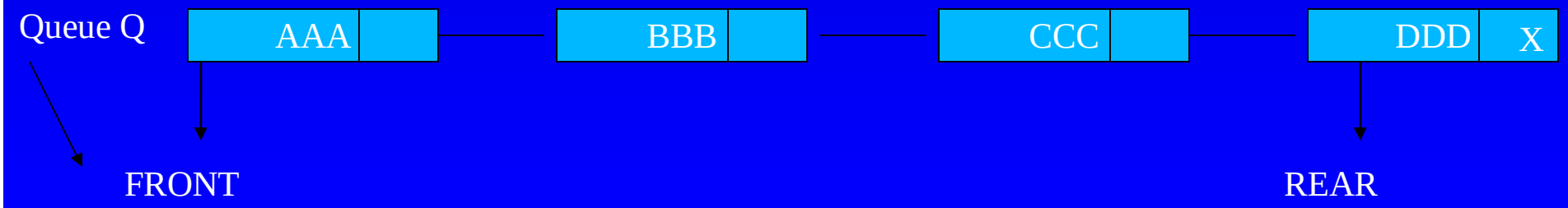
1. If FRONT=1 & REAR=N, or if FRONT = REAR+1, then
   Write: OVERFLOW and Return.
2. If FRONT = null, then (Queue is Empty)
   set FRONT=1 and REAR = 1
   Else if REAR=N then, set REAR = 1
   Else: set REAR= REAR+1
   End if
3. Set QUEUE (REAR)= ITEM (inserts new element)
4. Return.

Queue_delete (QUEUE, N, FRONT, REAR, ITEM)

1. If FRONT= null then, “underflow” and return
2. Set ITEM= QUEUE(FRONT)
3. If (FRONT=REAR)then
   Set FRONT=NULL and REAR = NULL
   else if (FRONT =N) then set FRONT=1
   else
   set FRONT= FRONT+1
   endif
4. Return

Linked representation of queues

- Queue implemented as a linked list with two pointer variables (FRONT & REAR)
- INFO field holds the element
- LINK field holds the pointers to the neighboring elements



Algorithm Insertion in LL

LinkQ_insert(info, link, front, rear, avail, item)

1. If AVAIL = NULL, then “overflow” and exit
2. Set NEW = AVAIL and AVAIL=LINK (AVAIL)
3. Set INFO [NEW] = ITEM & LINK (NEW)=NULL
4. If (FRONT = NULL) then FRONT = REAR = NEW

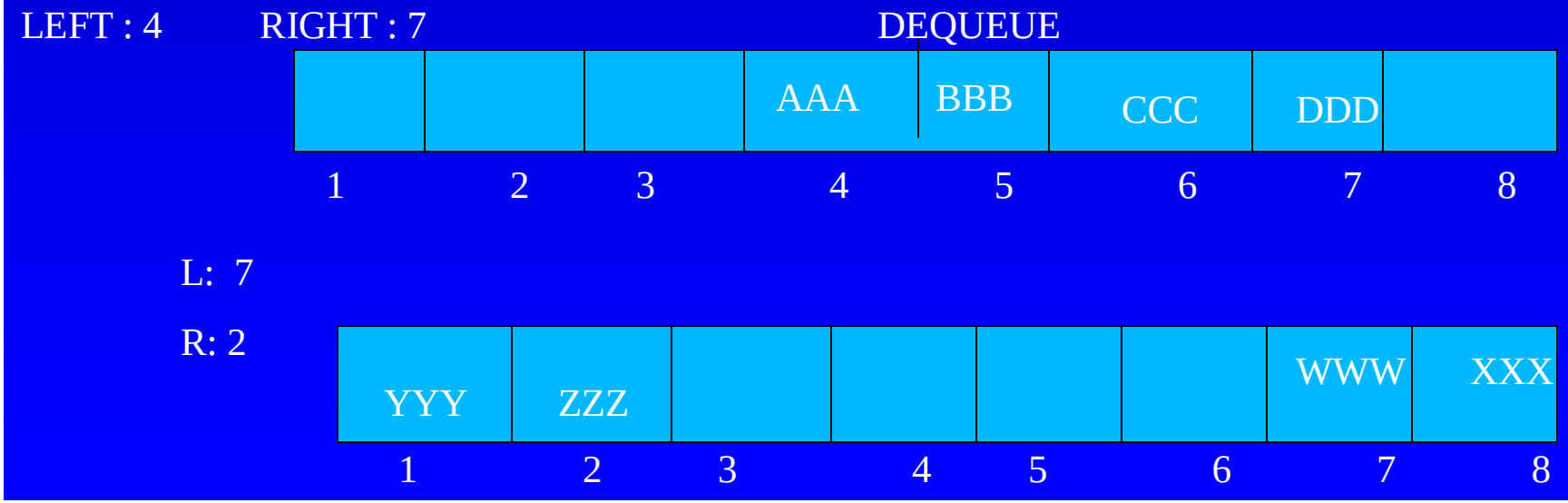
   Else: set LINK (REAR)=NEW & REAR=NEW
5. Exit

Linkq_delete(info,link, front, rear, avail, item)

1. If FRONT=NULL, then write”underflow” and exit
2. Set TEMP=FRONT
3. ITEM =INFO [TEMP]
4. FRONT = LINK [TEMP]
5. LINK (TEMP)=AVAIL & AVAIL=TEMP
6. Exit.

DEQUEUES

- is a linear list in which elements can be added or removed at either end not in middle
- Maintained by circular array
- With two pointers LEFT & RIGHT
- LEFT is NULL (deque is empty)



Priority Queues

- is a collection of elements & each has been assigned priority
- Rules are
  - An element of higher priority is processed before any element of lower priority
  - Two elements with same priority are processed according to the order in which they were added to the queue

Thankyou