Unit-3
Link List, Stack, Queue

Course: MCA

Subject: Data and File Structure

DEFINITION: STACK

- An ordered collection of data items
- Can be accessed at only one end (the top)

- Stacks are **LIFO** structures, providing
- Add Item (=PUSH) Methods
- Remove Item (=POP) Methods
- They are a simple way to build a collection
- No indexing necessary
- Size of collection must not be predefined
- But: extremely reduced accessibility

Basic Operations on a Stack

- **initializeStack**: Initializes the stack to an empty state
- **destroyStack**: Removes all the elements from the stack, leaving the stack empty
- **isEmptyStack**: Checks whether the stack is empty. If empty, it returns true; otherwise, it returns false

CONT..

- **isFullStack**: Checks whether the stack is full.
- If full, it returns true; otherwise, it returns false
- push:
  - Add new element to the top of the stack
  - The input consists of the stack and the new element.
  - Prior to this operation, the stack must exist and must not be full

CONT..

- **top**: Returns the top element of the stack. Prior to this operation, the stack must exist and must not be empty.

- **pop**: Removes the top element of the stack. Prior to this operation, the stack must exist and must not be empty.

Stacks: Properties

- Possible actions:
- PUSH an object (e.g. a plate) ontodispenser
- POP object out of dispenser

APPLICATIONS USING A STACK

- Examples:
- Finding Palindromes
- Bracket Parsing
- RPN
- RECURSION !

RECURSION

- Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first

- Recursion is a technique that solves a problem by solving a smaller problem of the same type

- A procedure that is defined in terms of itself

Factorial

$$a! = 1 * 2 * 3 * \ldots * (a-1) * a$$

$$a! = a * (a-1)!$$

remember

...splitting up the problem into a smaller problem of the same type...

$$\begin{tikzcd}
& a! \arrow[dl] \arrow[dr] & \\
a & * & (a-1)!
\end{tikzcd}$$

INFIX, POSTFIX AND PREFIX EXPRESSIONS

- **INFIX:** From our schools times we have been familiar with the expressions in which operands surround the operator,
- **e.g. x+y, 6*3** etc this way of writing the Expressions is called infix notation.

- **POSTFIX**: Postfix notation are also Known as Reverse Polish Notation (RPN). They are different from the infix and prefix notations in the sense that in the postfix notation, operator comes after the operands,
- **e.g. xy+, xyz+* etc.**

- **PREFIX**: Prefix notation also Known as Polish notation.In the prefix notation, as the name only suggests, operator comes before the operands,
- **e.g. +xy, *+xyz etc**.

Using a Queue

- ➢Representation Of Queue
- ➢Operations On Queue
- ➢ Circular Queue
- ➢Priority Queue
- ➢Array Representation of Priority Queue
- ➢Double Ended Queue
- ➢Applications of Queue

WHAT IS A QUEUE?

- A queue system is a linear list in which deletions can take place only at one end the “front” of the list, and the insertions can take place only at the other end of the list, the “back” .
- Is called a First-In-First-Out(FIFO)

The Queue Operations

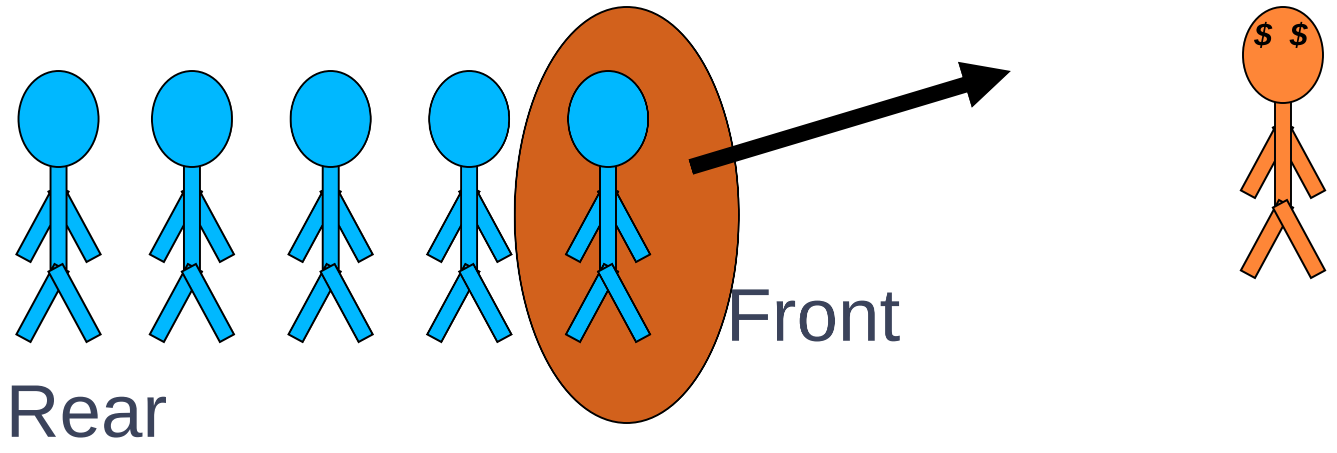
- A queue is like a line of people waiting for a bank teller.
- The queue has a **front** and a **rear**.

THE QUEUE OPERATIONS

- New people must enter the queue at the rear.
- The C++ queue class calls this a **push**, although it is usually called an **enqueue** operation.

The Queue Operations

- When an item is taken from the queue, it always comes from the front.
- The C++ queue calls this a **pop**, although it is usually called a **dequeue** operation.



Operations on Queues

- **Insert(**item**)**: (also called enqueue)
  - It adds a new item to the tail of the queue
- **Remove( )**: (also called delete or dequeue)
  - It deletes the head item of the queue, and returns to the caller.
  - If the queue is already empty, this operation returns NULL
- **getHead( ):**
  - Returns the value in the head element of the queue
- **getTail( ):**
  - Returns the value in the tail element of the queue
- **isEmpty( )**
  - Returns **true** if the queue has no items
- **size**( )
  - Returns the number of items in the queue

The Queue Class

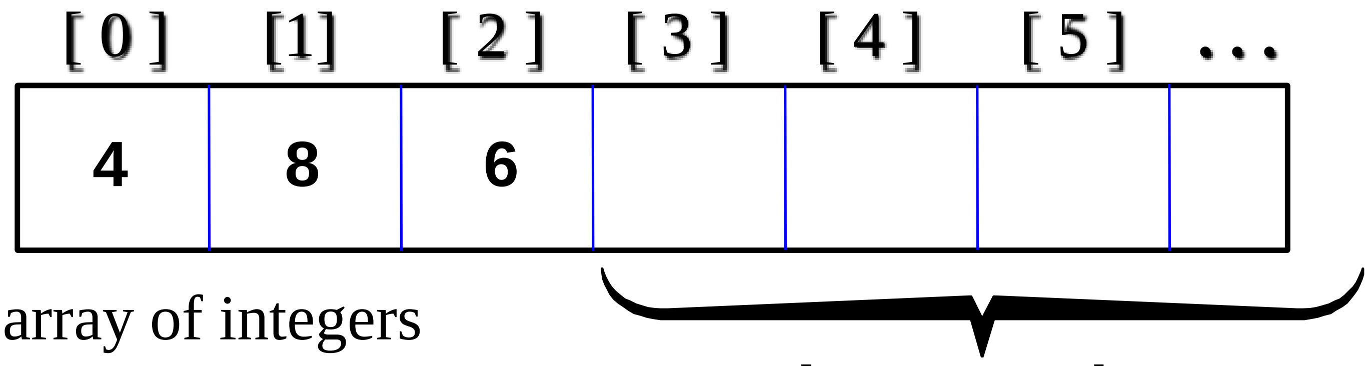
- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

```
template <class Item>
class queue<Item>
{
public:
        queue( );
  void push(const Item& entry);
        void pop( );
        bool empty( ) const;
        Item front( ) const;
        …

};
```

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).

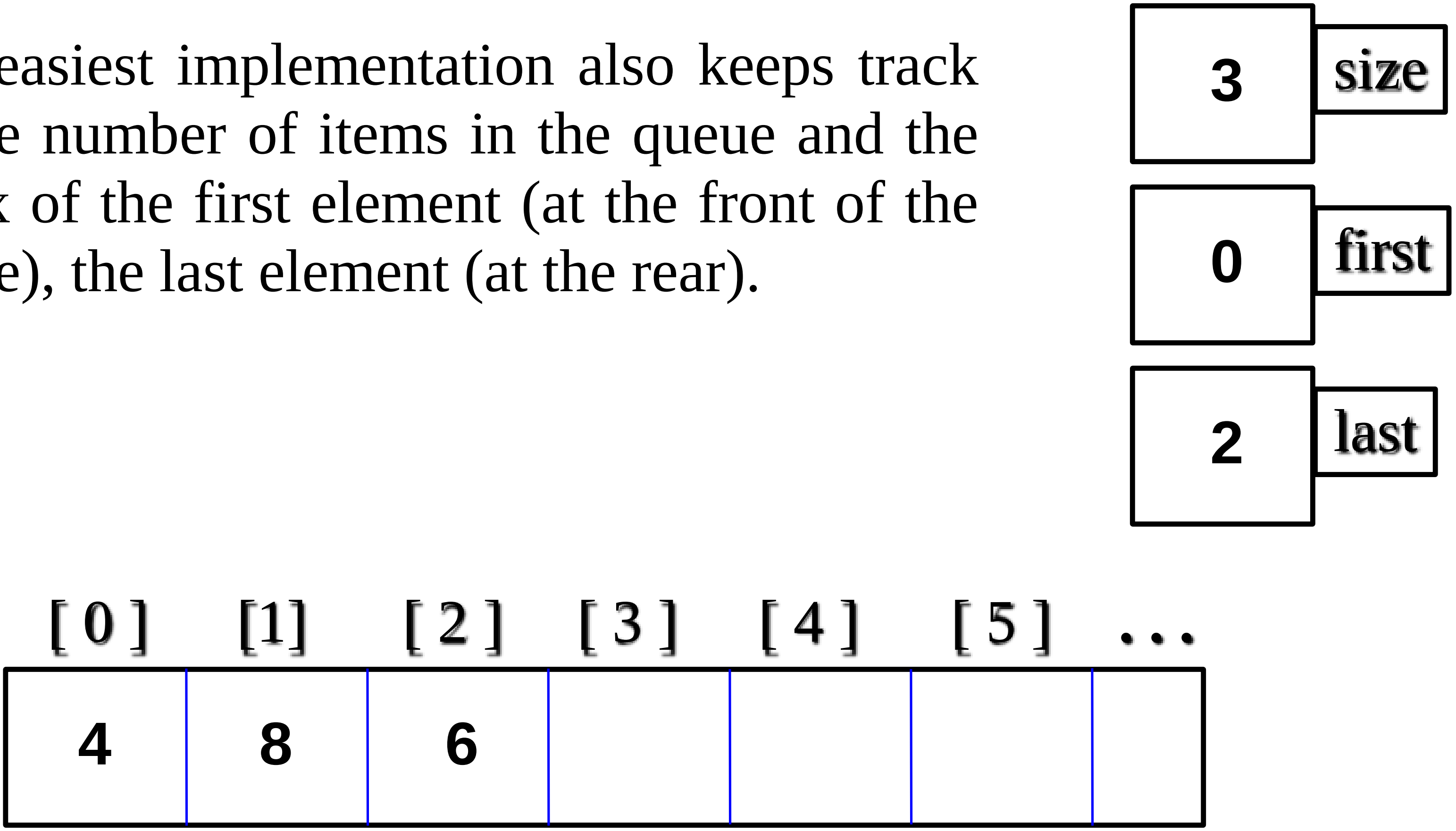


An array of integers to implement a queue of integers

We don't care what's in this part of the array.

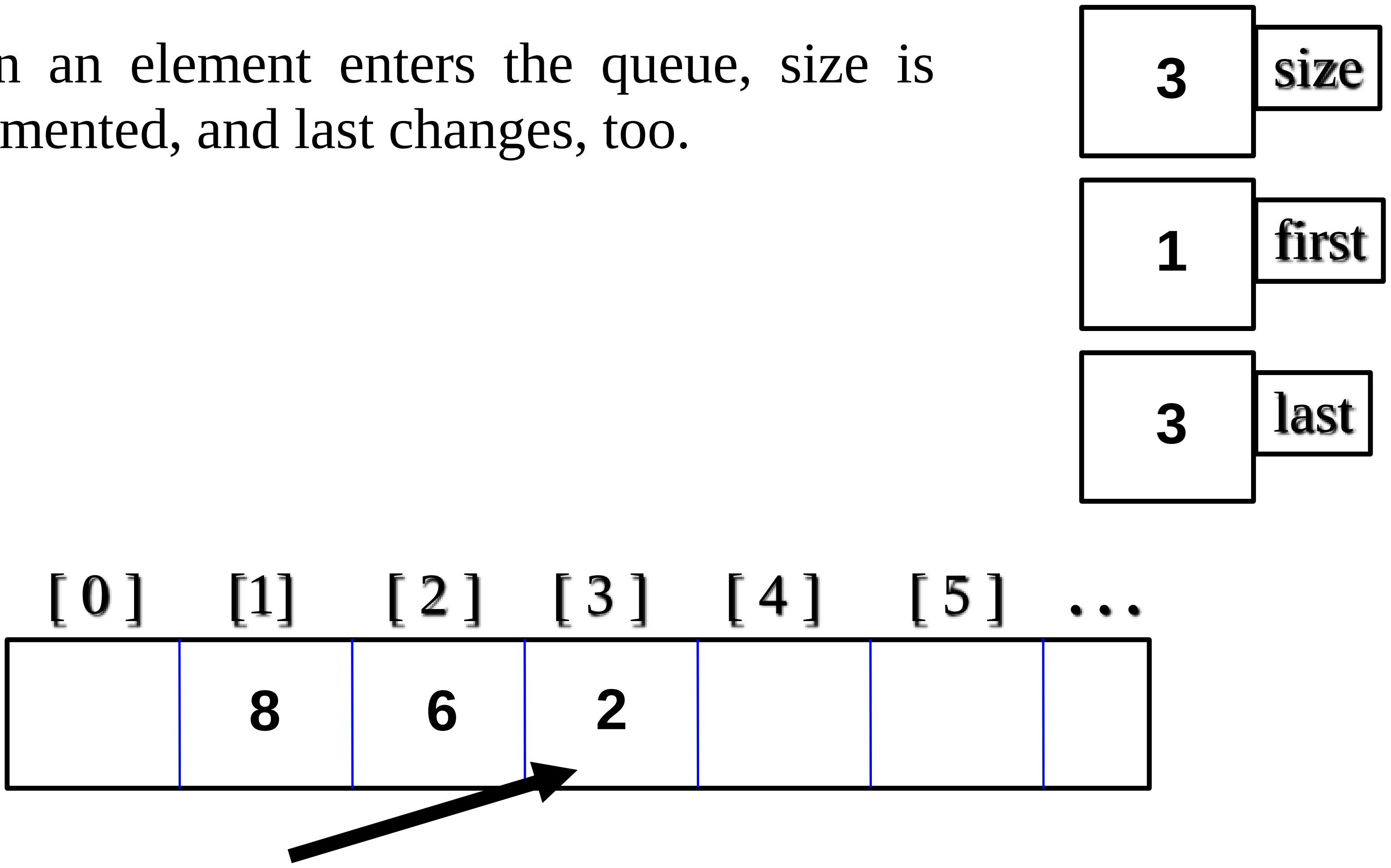
ARRAY IMPLEMENTATION

- 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).



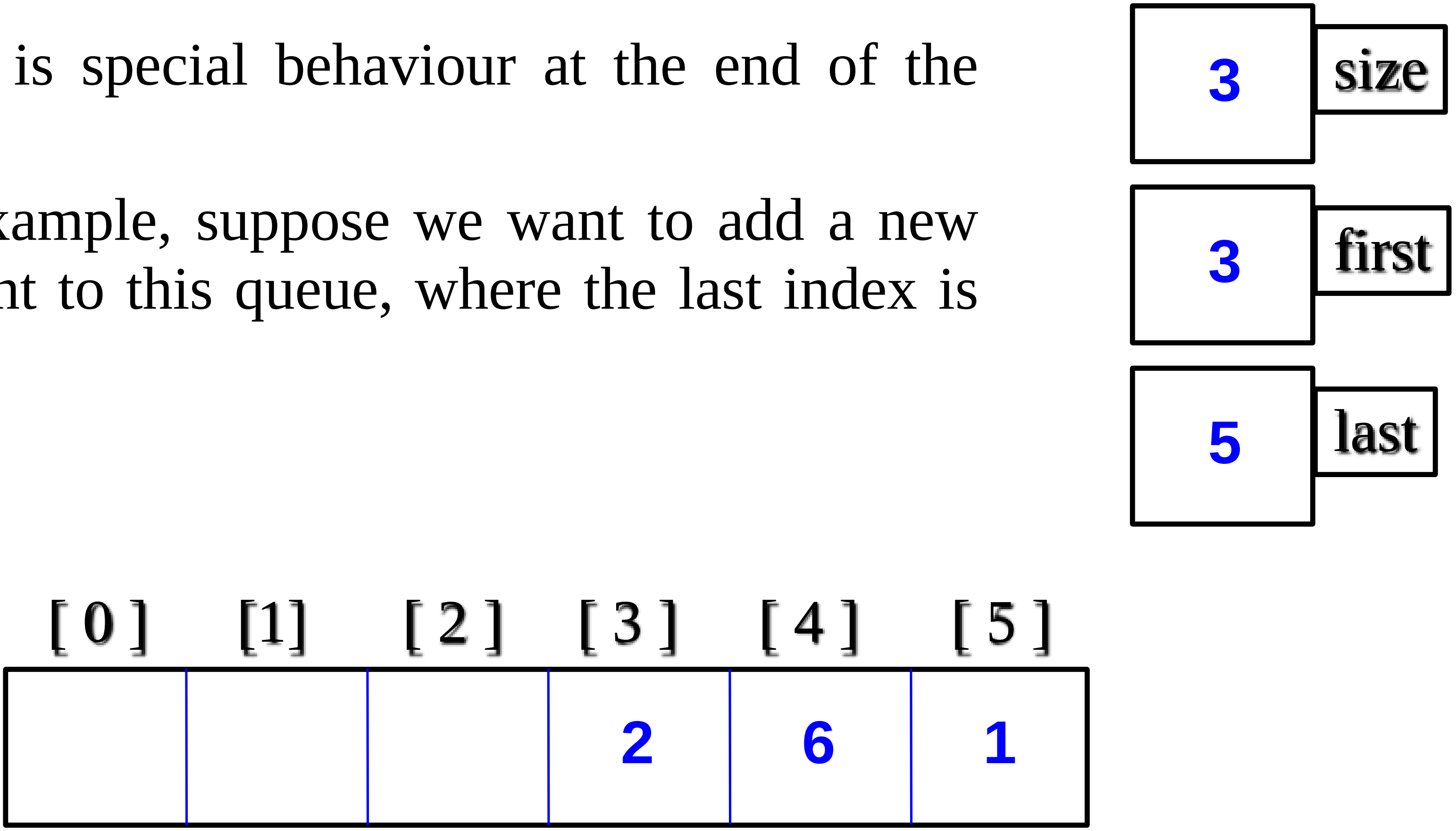
An Enqueue Operation

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



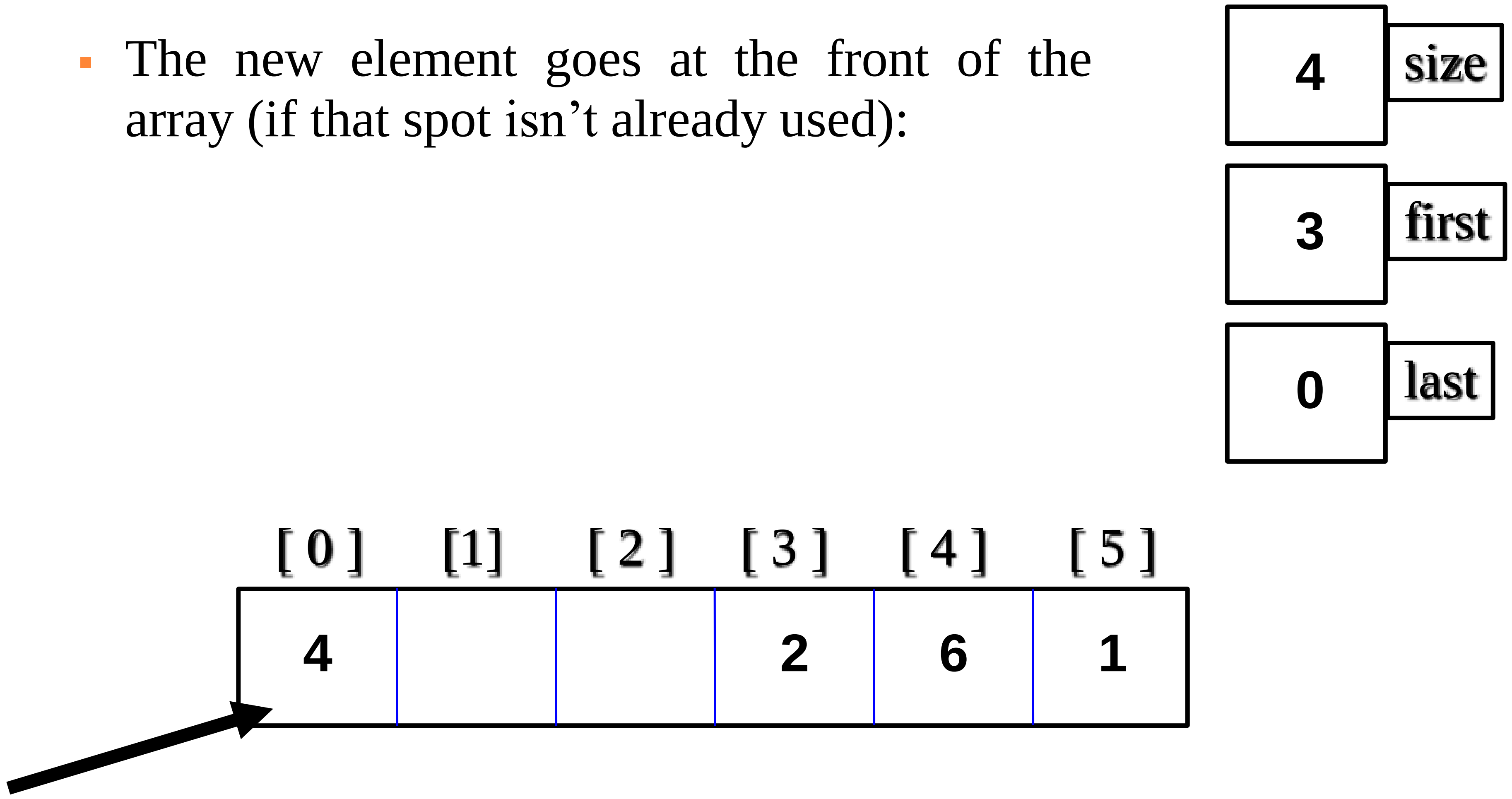
At the End of the Array

- 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]:



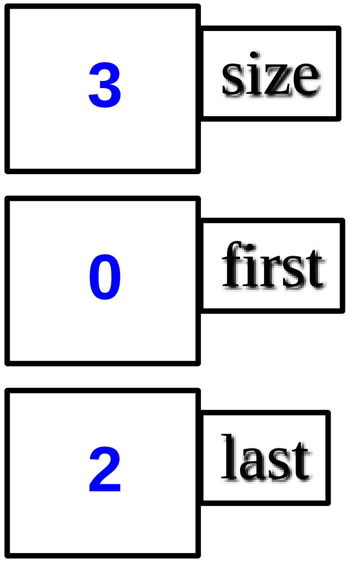
At the End of the Array

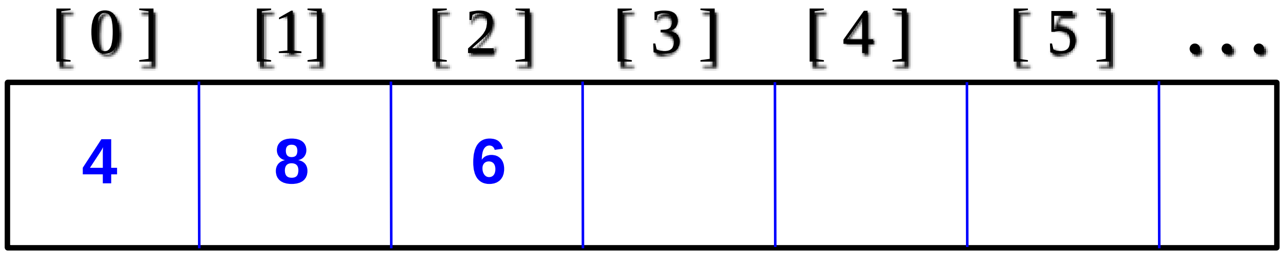
- The new element goes at the front of the array (if that spot isn't already used):



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 behavior is needed when the rear reaches the end of the array.





CIRCULAR QUEUE

- When the queue reaches the end of the array, it “wraps around” and the rear of the queue starts from index 0.
- A The figure below demonstrates the situation.

PRIORITY QUEUES

- A priority queue is a collection of zero or more elements → each element has a **priority** or value
- Unlike the FIFO queues, the order of deletion from a priority queue (e.g., who gets served next) is determined by the element priority
- Elements are deleted by increasing or decreasing order of priority rather than by the order in which they arrived in the queue

PRIORITY QUEUES

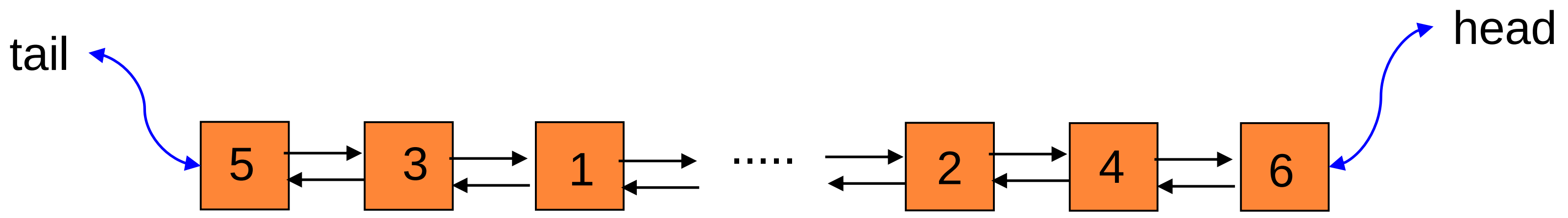
- Operations performed on priority queues
  - 1) Find an element, 2) insert a new element, 3) delete an element, etc.
- Two kinds of (Min, Max) priority queues exist
- In a Min priority queue, find/delete operation finds/deletes the element with minimum priority
- In a Max priority queue, find/delete operation finds/deletes the element with maximum priority
- Two or more elements can have the same priority

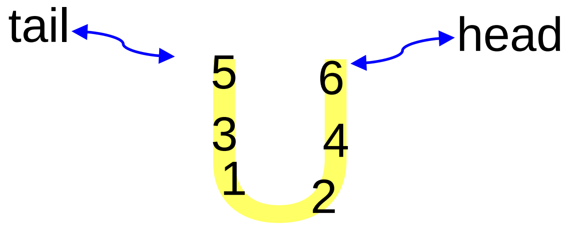
DEQUES

- A deque is a double-ended queue
- Insertions *and* deletions can occur at *either* end
- Implementation is similar to that for queues
- Deques are not heavily used
- You should know what a deque is, but we won’t explore them much further

DEQUE (DOUBLE-ENDED QUEUE)[1]

Implemented by the class Arraydeque in Java 6 seen as a doubly linked list with a head (right) and a tail (left) and insert and remove at either ends.





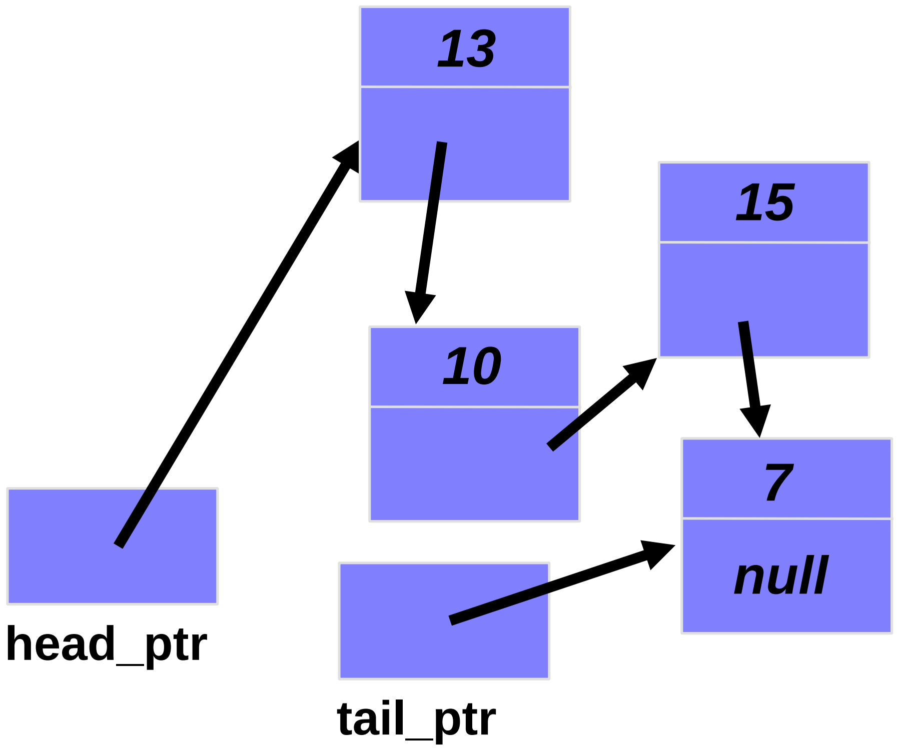
commuting pipes

Queue Applications

- Real life examples
  - Waiting in line
  - Waiting on hold for tech support
- Applications related to Computer Science
  - Threads
  - Job scheduling (e.g. Round-Robin algorithm for CPU allocation)

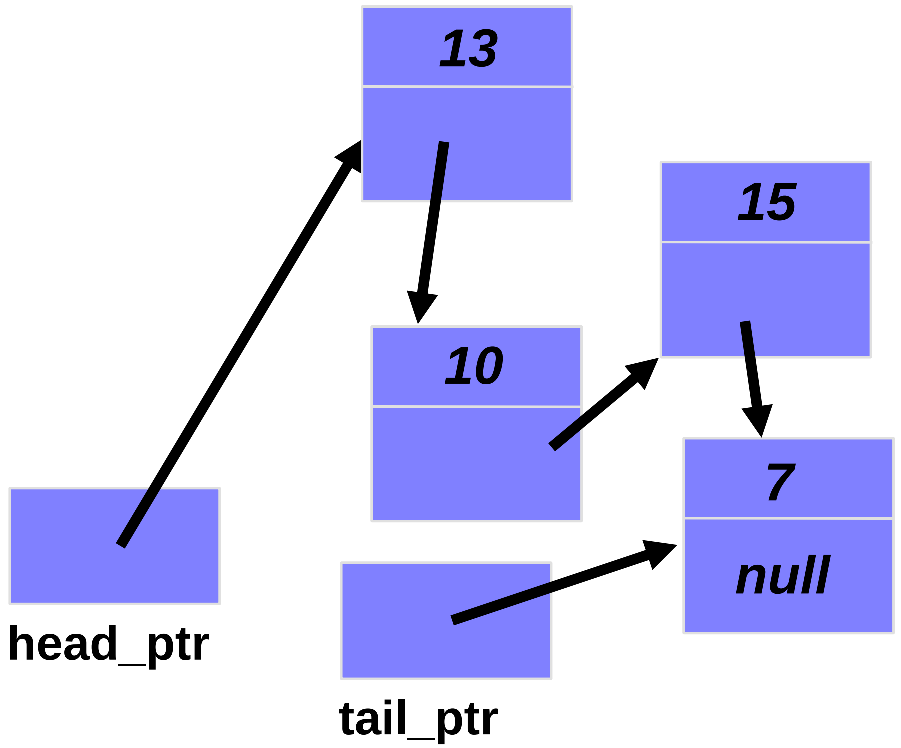
Linked List Implementation[2]

- A queue can also be implemented with a linked list with both a head and a tail pointer.



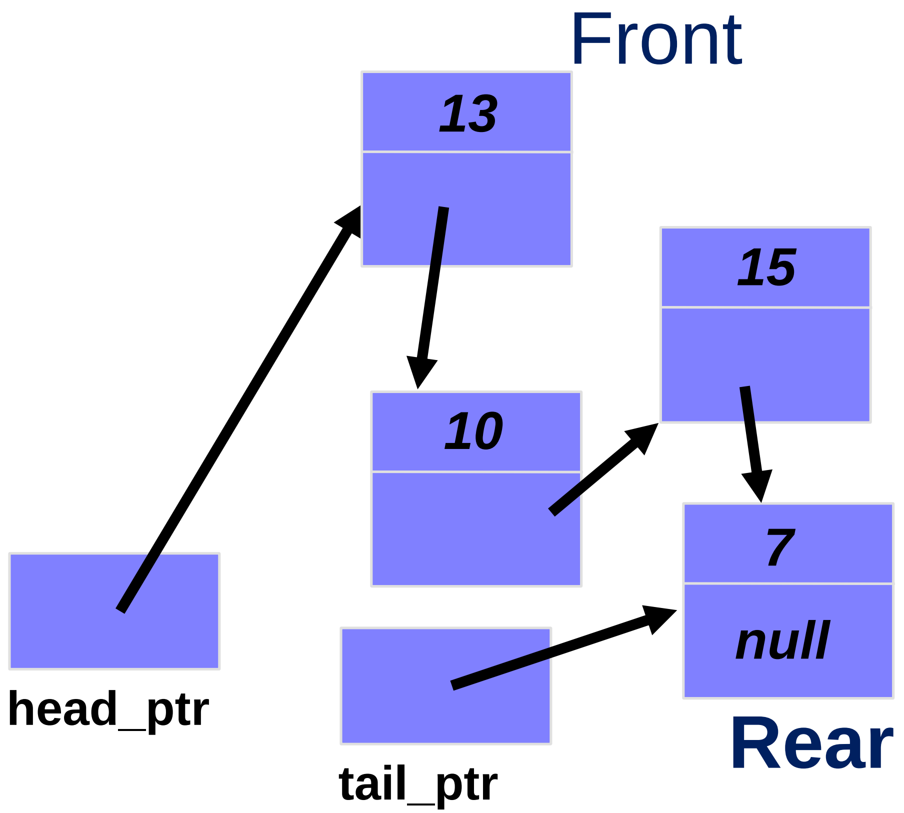
LINKED LIST IMPLEMENTATION[3]

- Which end do you think is the front of the queue? Why?



LINKED LIST IMPLEMENTATION[4]

- The head_ptr points to the front of the list.
- Because it is harder to remove items from the tail of the list.

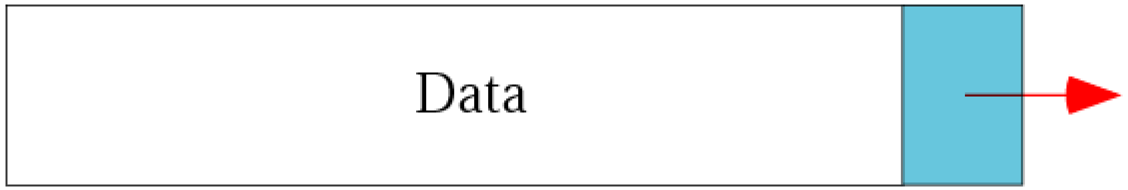


LINKED LIST

- Linkedlist an ordered collection of data in which each element contains the location of the next element.
- Each element contains two parts: data and link.
- The link contains a pointer (an address) that identifies the next element in the list.

- Singly linked list
- The link in the last element contains a null pointer, indicating the end of the list.

Node[5]



- Nodes : the elements in a linked list.

- The nodes in a linked list are called self-referential records.
- Each instance of the record contains a pointer to another instance of the same structural type.

SENTINEL NODES

- To simplify programming, two special nodes have been added at both ends of the doubly-linked list.

- Head and tail are dummy nodes, also called sentinels, do not store any data elements.

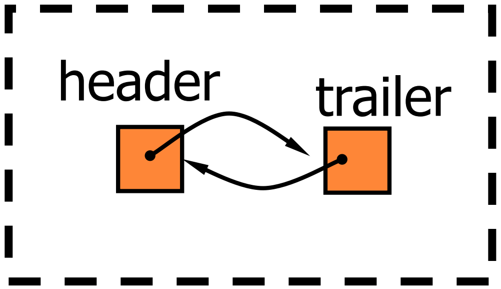
- Head: header sentinel has a ***null-prev*** reference (link).
- Tail: trailer sentinel has a ***null-next*** reference (link).

Empty Doubly-Linked List:[6]

Using sentinels, we have no null-links; instead, we have:

head.next = tail

tail.prev = head



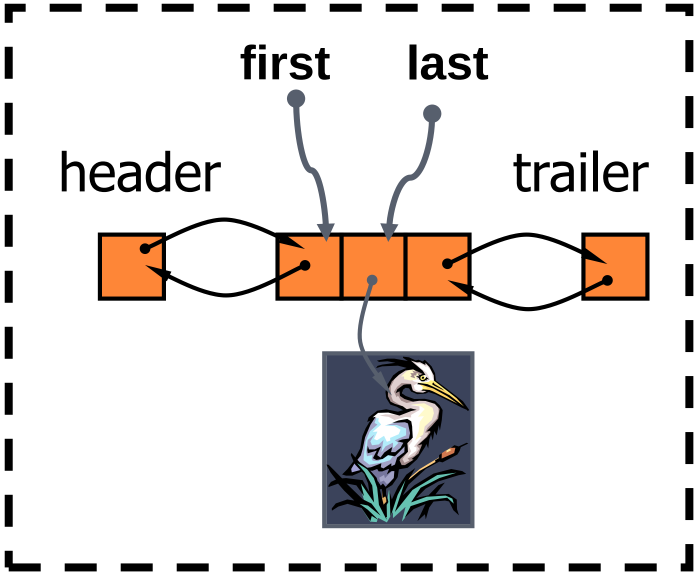
Singl Node List:

Size = 1

This single node is the first node, and also is the last node:

first node is head.next

last node is tail.prev



CIRCULAR LINKED LISTS

- In linear linked lists if a list is traversed (all the elements visited) an external pointer to the list must be preserved in order to be able to reference the list again.

- Circular linked lists can be used to help the traverse the same list again and again if needed.

- A circular list is very similar to the linear list where in the circular list the pointer of the last node points not NULL but the first node.

Circular Linked Lists

- In a circular linked list there are two methods to know if a node is the first node or not.

  - Either a external pointer, ***list***, points the first node or
  - A ***header node*** is placed as the first node of the circular list.

- The header node can be separated from the others by either heaving a ***sentinel value*** as the info part or having a dedicated ***flag*** variable to specify if the node is a header node or not.

References

- An introduction to Datastructure with application by jean Trembley and sorrenson
- Data structures by schaums and series –seymour lipschutz
- http://en.wikipedia.org/wiki/Book:Data_structures
- http://www.amazon.com/Data-Structures-Algorithms
- http://www.amazon.in/Data-Structures-Algorithms-Made-Easy/dp/0615459811/
- http://www.amazon.in/Data-Structures-SIE-Seymour-Lipschutz/dp
- **List of images**

1) http://whttp://www.amazon.in/Data-Structures-SIE-Seymour-Lipschutz/dq
2) ww.amazon.in/Data-Structures-linkedlist
3) ww.amazon.in/Data-Structures-linkedlist
4) ww.amazon.in/Data-Structures-linkedlist
5) ww.amazon.in/Data-Structures-linkedlist
6) ww.amazon.in/Data-Structures-doubly linkedlist