CSE 373

MARCH 29 – TESTING AND PRIORITY QUEUES

ASSORTED MINUTIAE

- **Midterm exam:**
  - April 28th; 2:30 – 3:20
- **Canvas**
  - Site is up – HW1 out after class
- **143 Workshop**
  - Late this week or early next

ASSORTED MINUTIAE

- **HW 1**
  - Two parts
  - Implementation and Testing
  - Part 1 (143 review)
  - Part 2 (Testing)

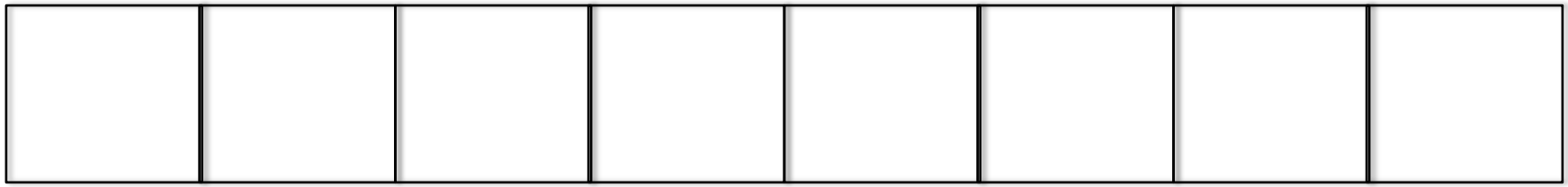
TODAY'S SCHEDULE

- Circular Queues
- Software Testing
- New ADT: Priority Queue

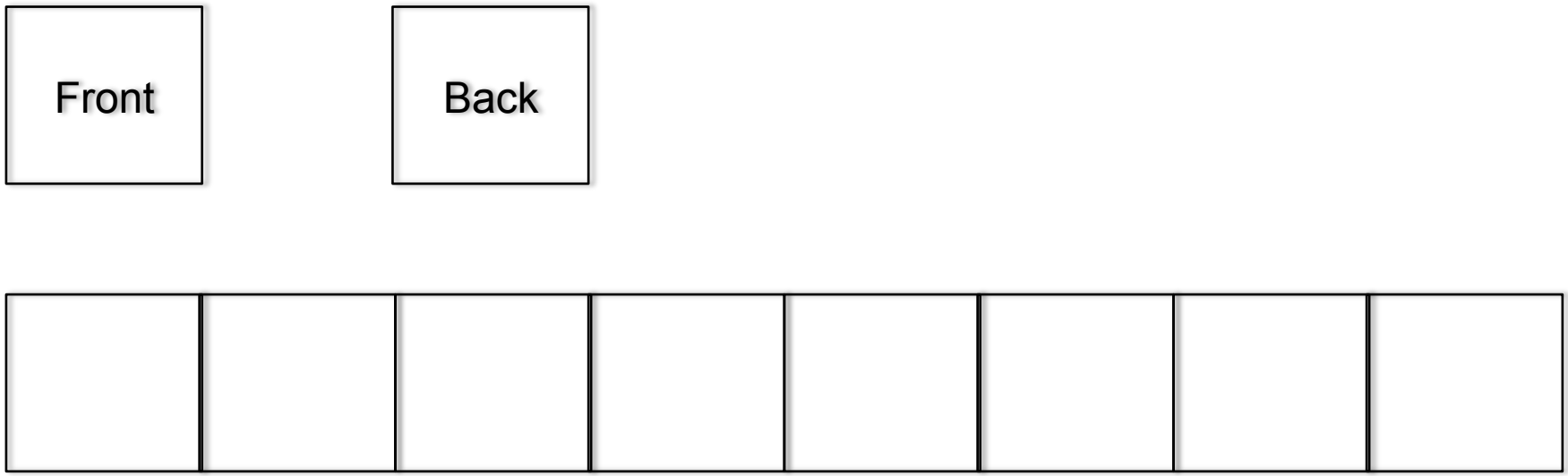
CONTINUING FROM LAST WEEK

- Stacks and Queues
- Linked List v. Array Implementation
- Design decisions and “resizing”
- Alternating push and pop situation

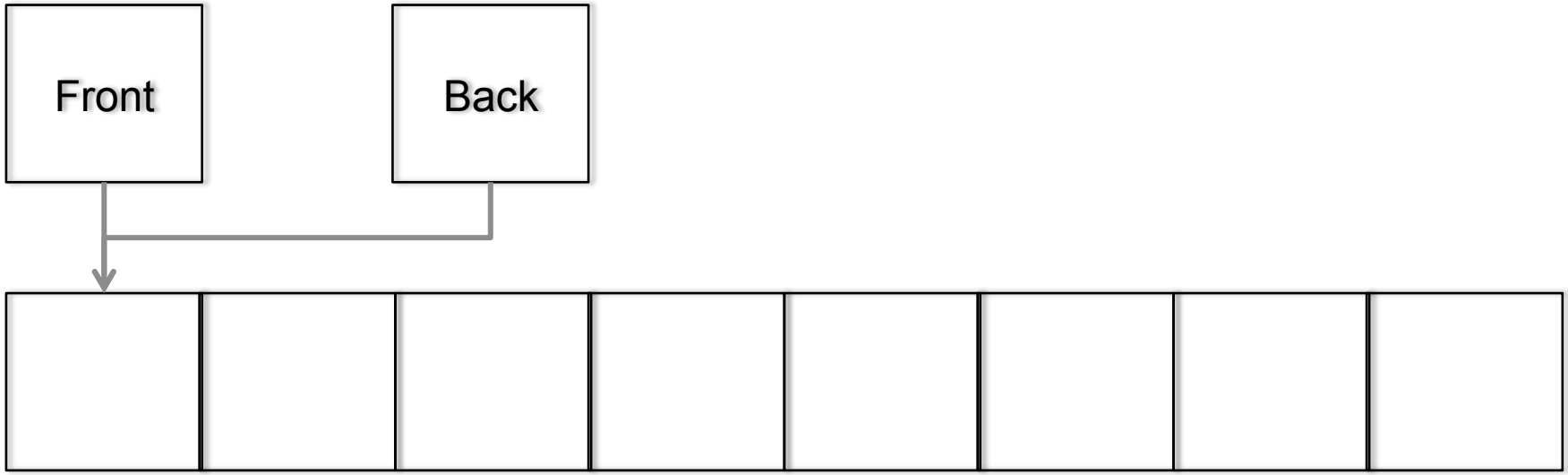
CIRCULAR QUEUES



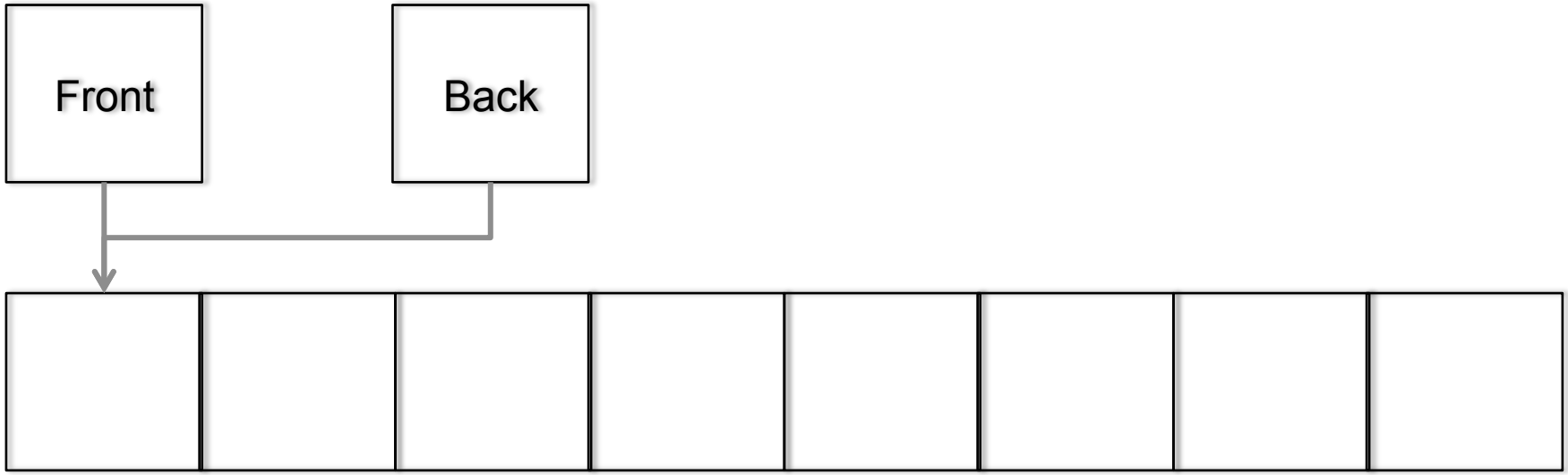
CIRCULAR QUEUES



CIRCULAR QUEUES



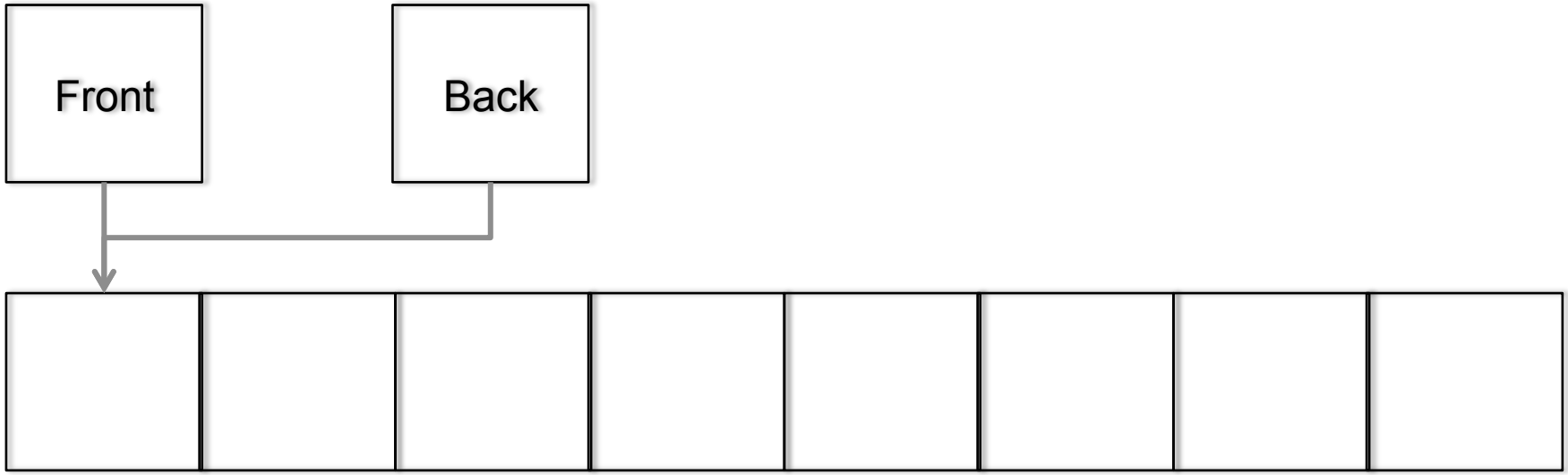
CIRCULAR QUEUES



Why this way?

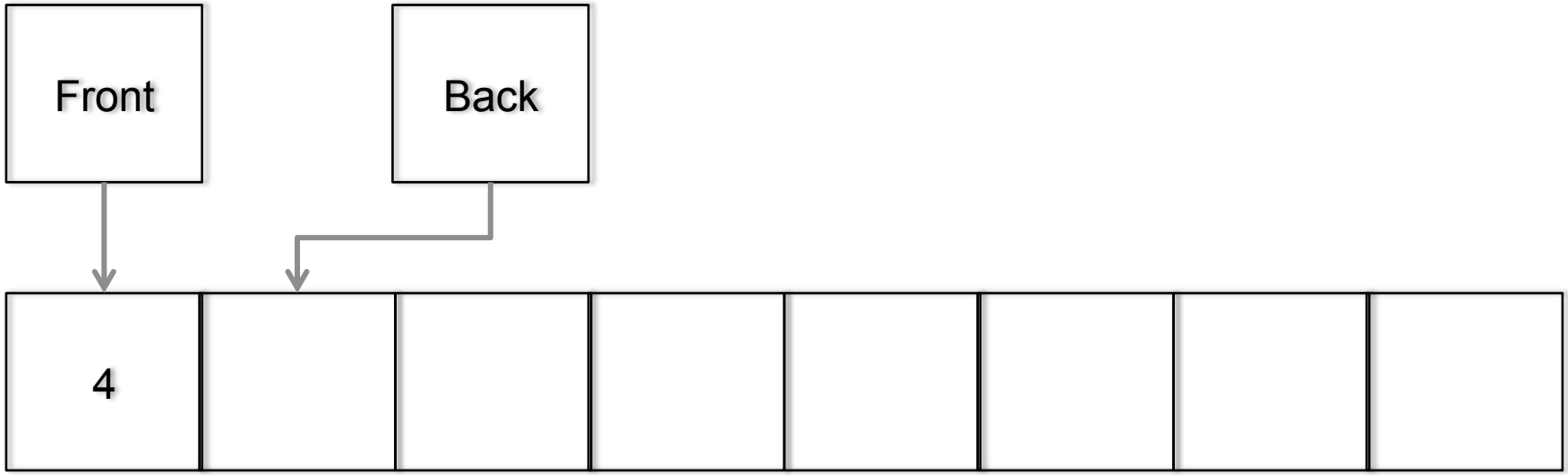
What function to front and back serve?

CIRCULAR QUEUES



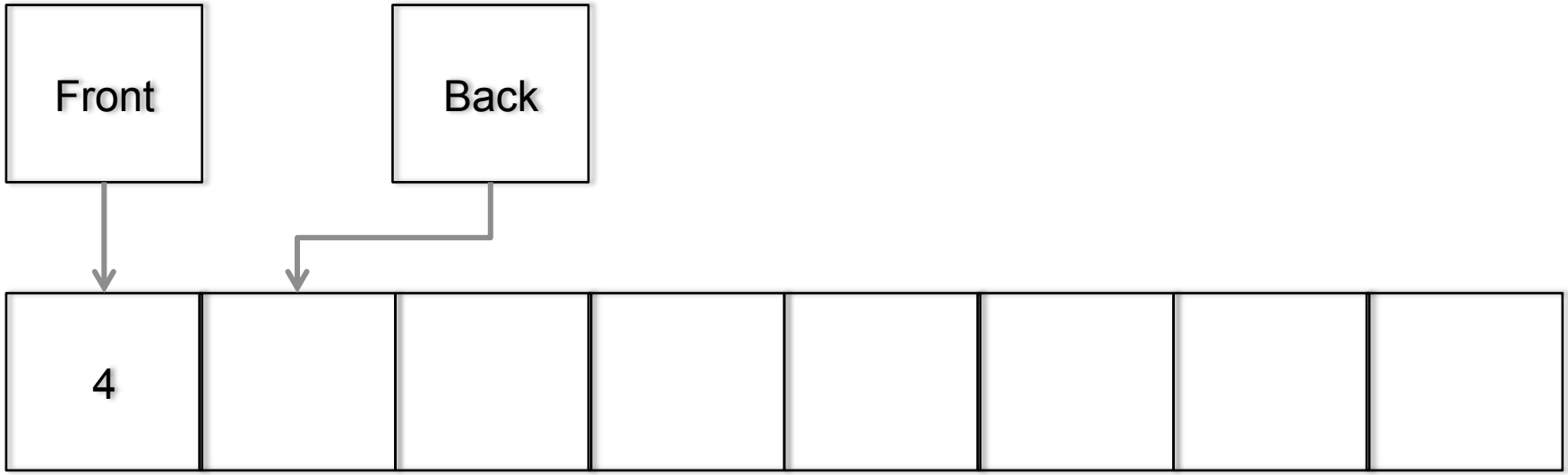
enqueue(4)

CIRCULAR QUEUES



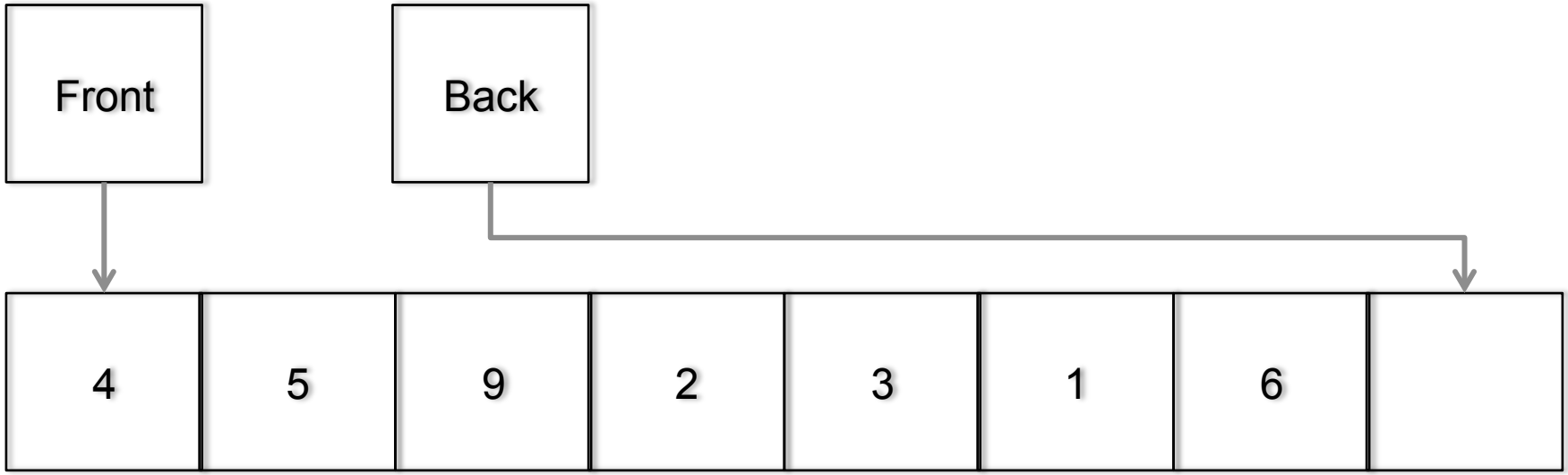
Which operations will move what pointers?

CIRCULAR QUEUES



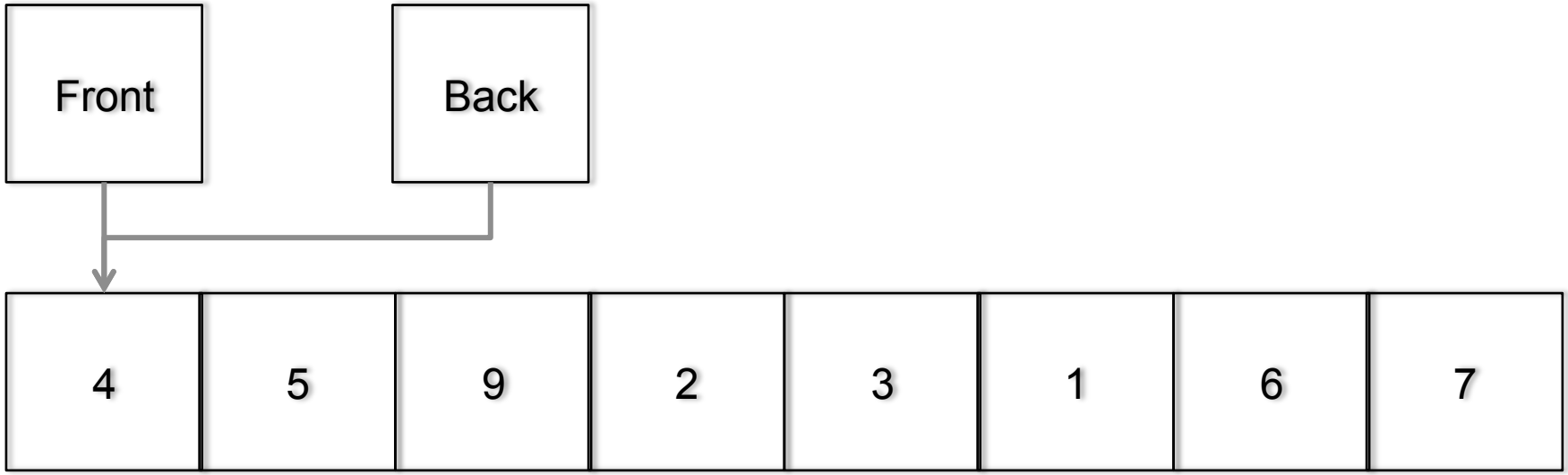
Let’s do several enqueues

CIRCULAR QUEUES



What happens now, on enqueue(7)?

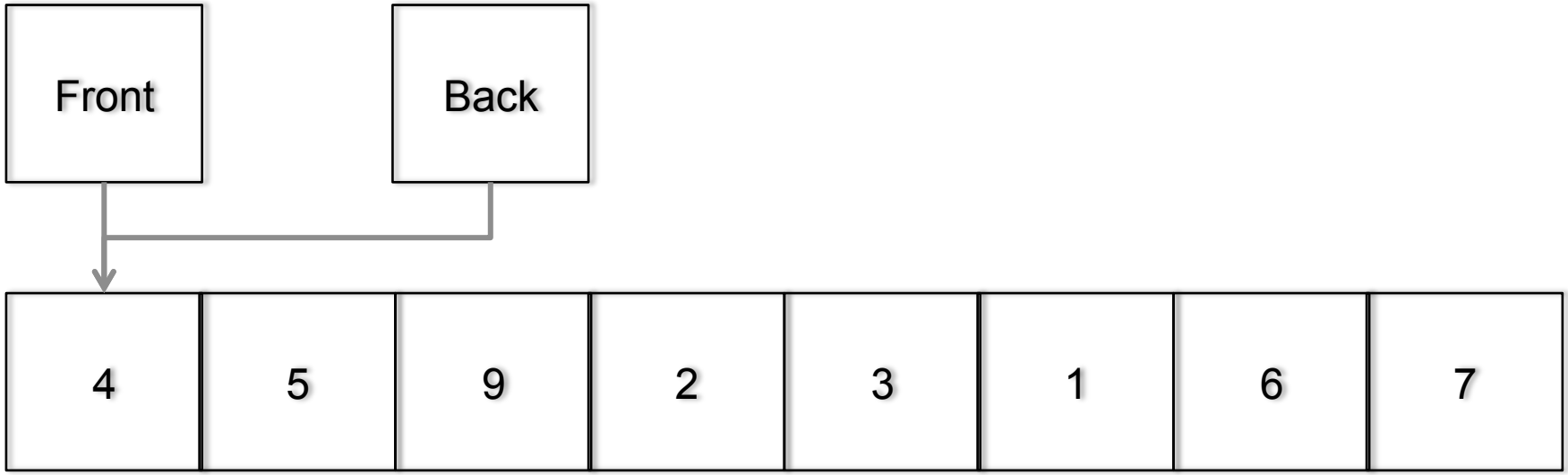
CIRCULAR QUEUES



Problems here?

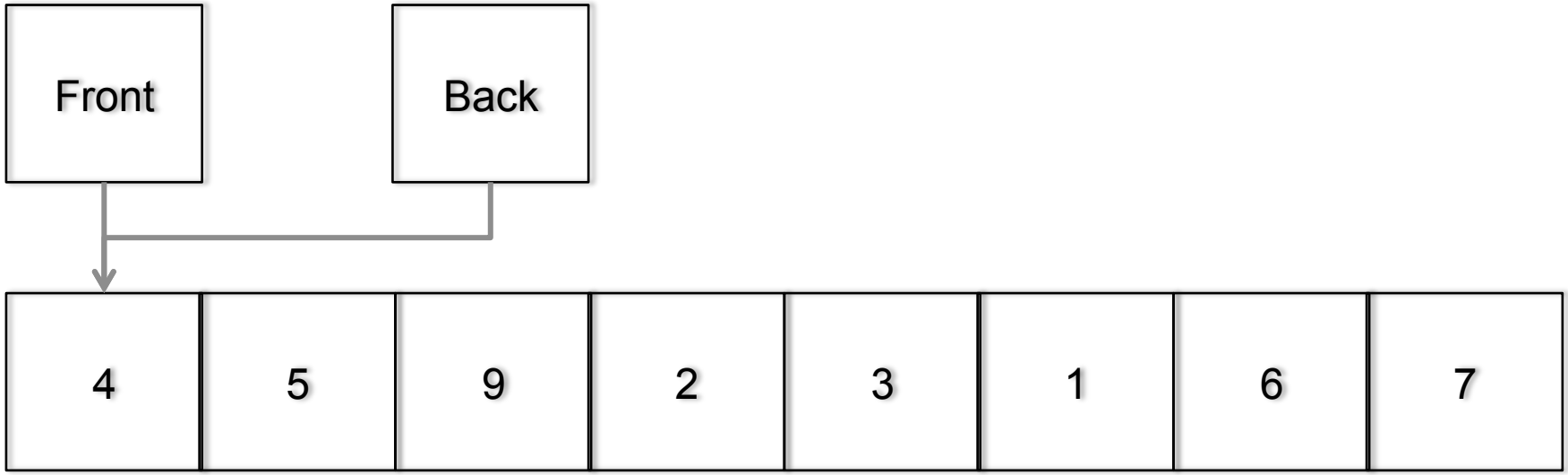
How to implement?

CIRCULAR QUEUES



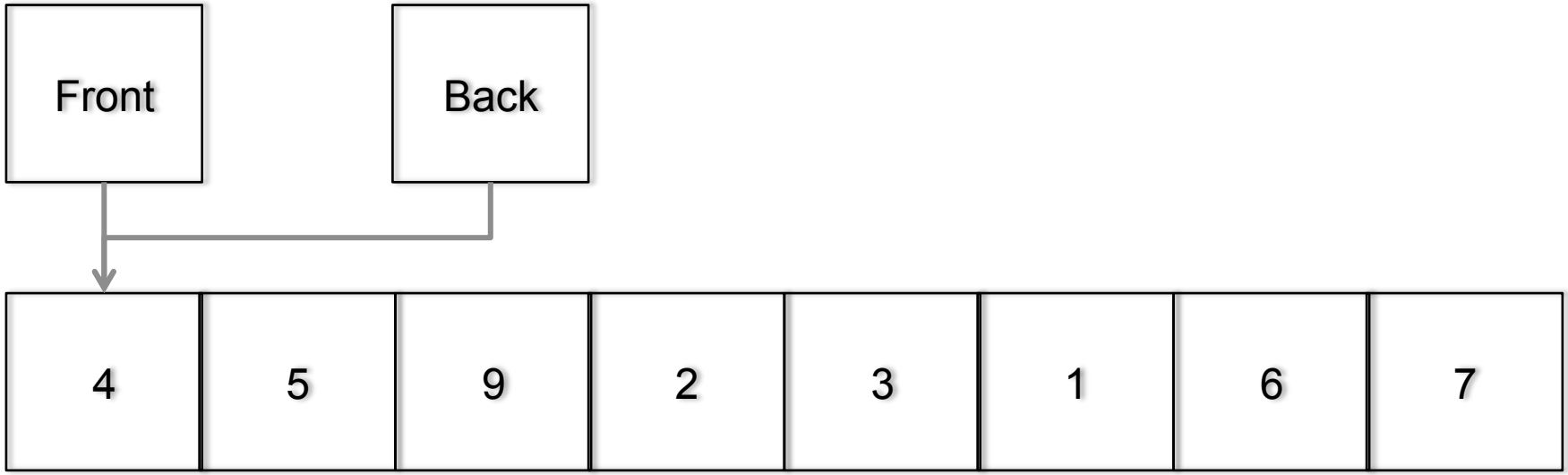
The queue is full, but it is the same situation (front == back) as when the queue is empty. This is a boundary condition.

CIRCULAR QUEUES



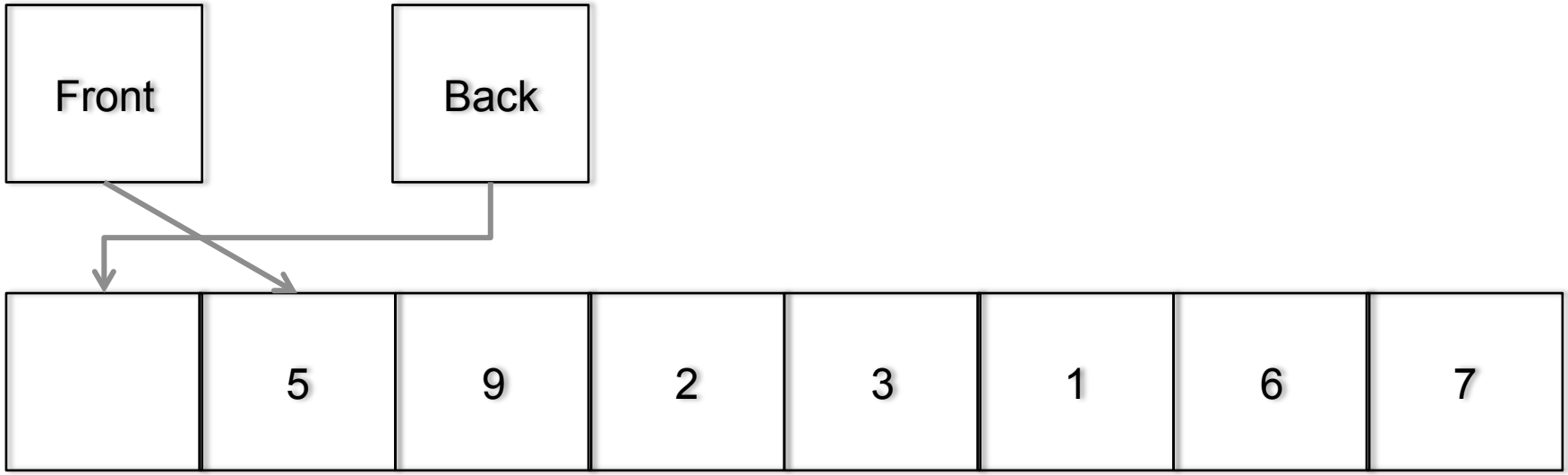
We have to resize the list (or deny the add) if we get another enqueue.

CIRCULAR QUEUES



What if we dequeue some items?

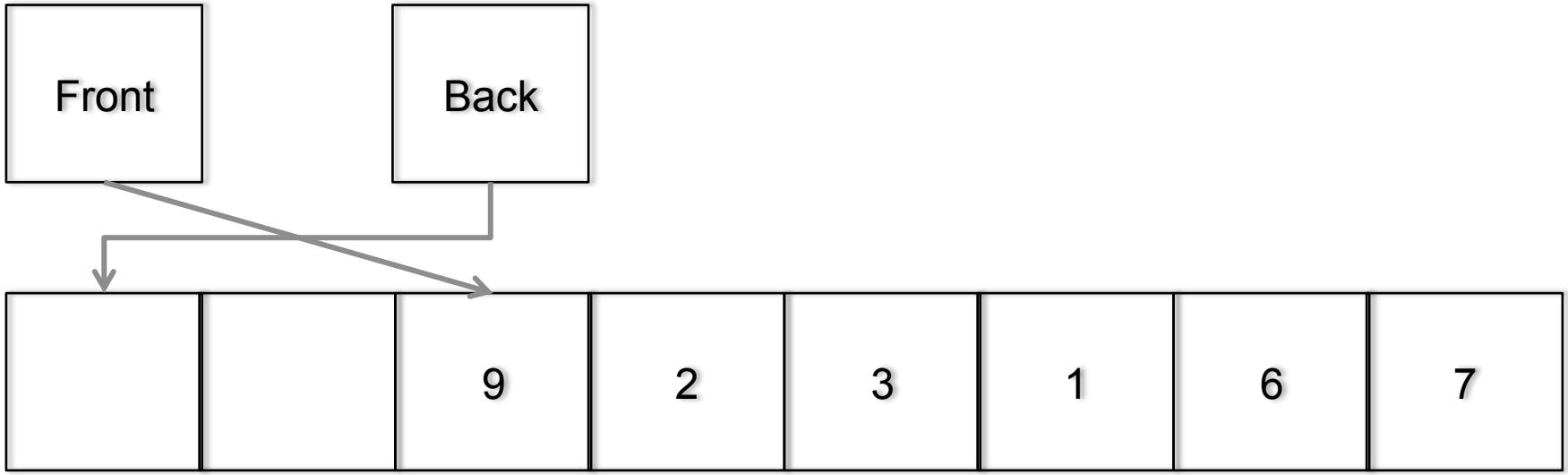
CIRCULAR QUEUES



Dequeue() outputs 4

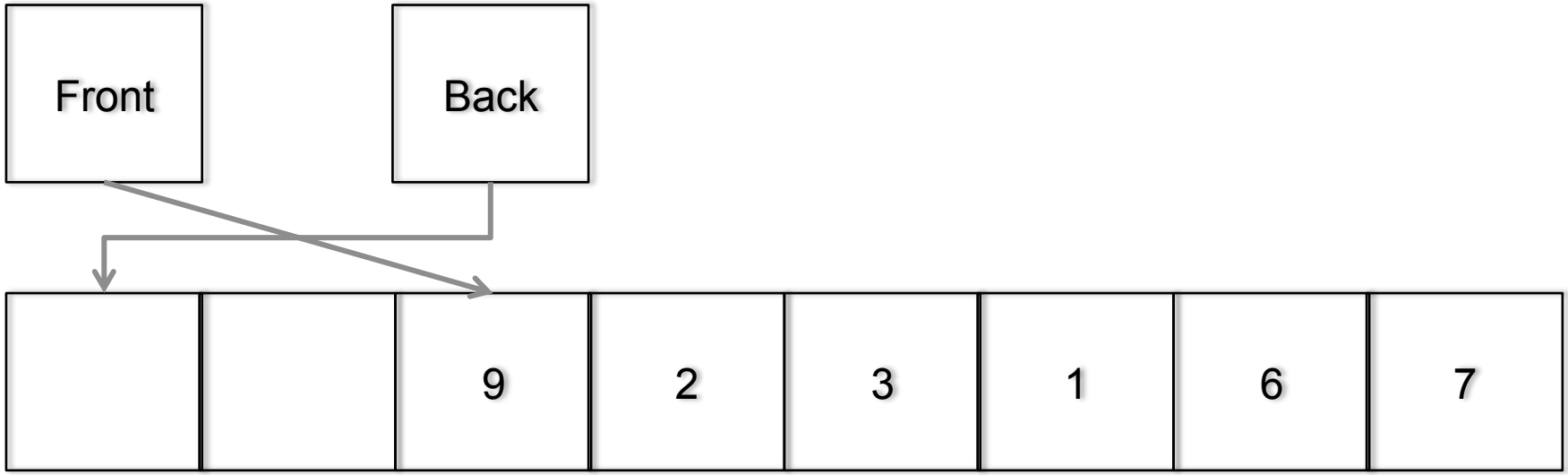
Is the 4 really “deleted”?

CIRCULAR QUEUES



Dequeue outputs 5

CIRCULAR QUEUES



Now we've freed up some space and can enqueue more

CIRCULAR QUEUES

- **By moving the front and back pointers, we can utilize all of the space in the array**
- **Advantages over a linked list?**
  - Fixed number of items
  - Small data (Memory efficiency)
- **BONUS: What is the memory overhead of the linked list?**

TESTING

- Implementation is great if it works on the first try
- In a large implementation, what is causing the problem?
- Object oriented programming allows modularity – good testing can pinpoint bugs to particular modules

TESTING

- **Two primary types of testing**
  - Black box
    - Behavior only, no peeking into the code
  - White box (or clear box)
    - Where there is an understanding of the implementation that can be leveraged for testing

TESTING

- **Part 1 on the homework will involve writing tests for your own implementation. (White box)**
- **Part 2 will involve testing java .class files.**
  - Only the interface (TestQueue) and expected behavior are known

TESTING

- **Isolate the problem**
  - Write specific tests
  - Running the whole program doesn't help narrow down problems
- **What are expected test cases?**
  - In general: [0,1,n] are good starting points
  - White box testing can take advantage of boundary cases (e.g. the resize of an array)

TESTING

- **Many test cases (and large ones)**
  - You can prove that an algorithm is correct, but you cannot necessarily prove an arbitrary implementation is correct
- **More inputs can increase certainty**
  - Adversarial testing
  - The client is not your friend

PRIORITY QUEUE

- **New ADT**
- **Objects in the priority queue have:**
  - Data
  - Priority
- **Conditions**
  - Lower priority items should dequeue first
  - Change priority?

PRIORITY QUEUE

- **Applications?**
  - Hospitals
  - CSE course overloads
  - Etc…

PRIORITY QUEUE

- **How to implement?**
  - Keep data sorted (somehow)
- **Array?**
  - Inserting into the middle is costly (must move other items)
- **Linked list?**
  - Must iterate through entire list to find place
  - Cannot move backward if priority changes

PRIORITY QUEUE

- These implementations will all give us the behavior we want as far as the ADT, but they may be poor design decisions
- Any other data structures to try?

NEXT CLASS

- Improve upon LL v. Array
- ADTs v. Data Structures
- DS properties and their importance
- Implementation impacts