Databases and Algorithms

Queue

Instructor: Rossano Schifanella

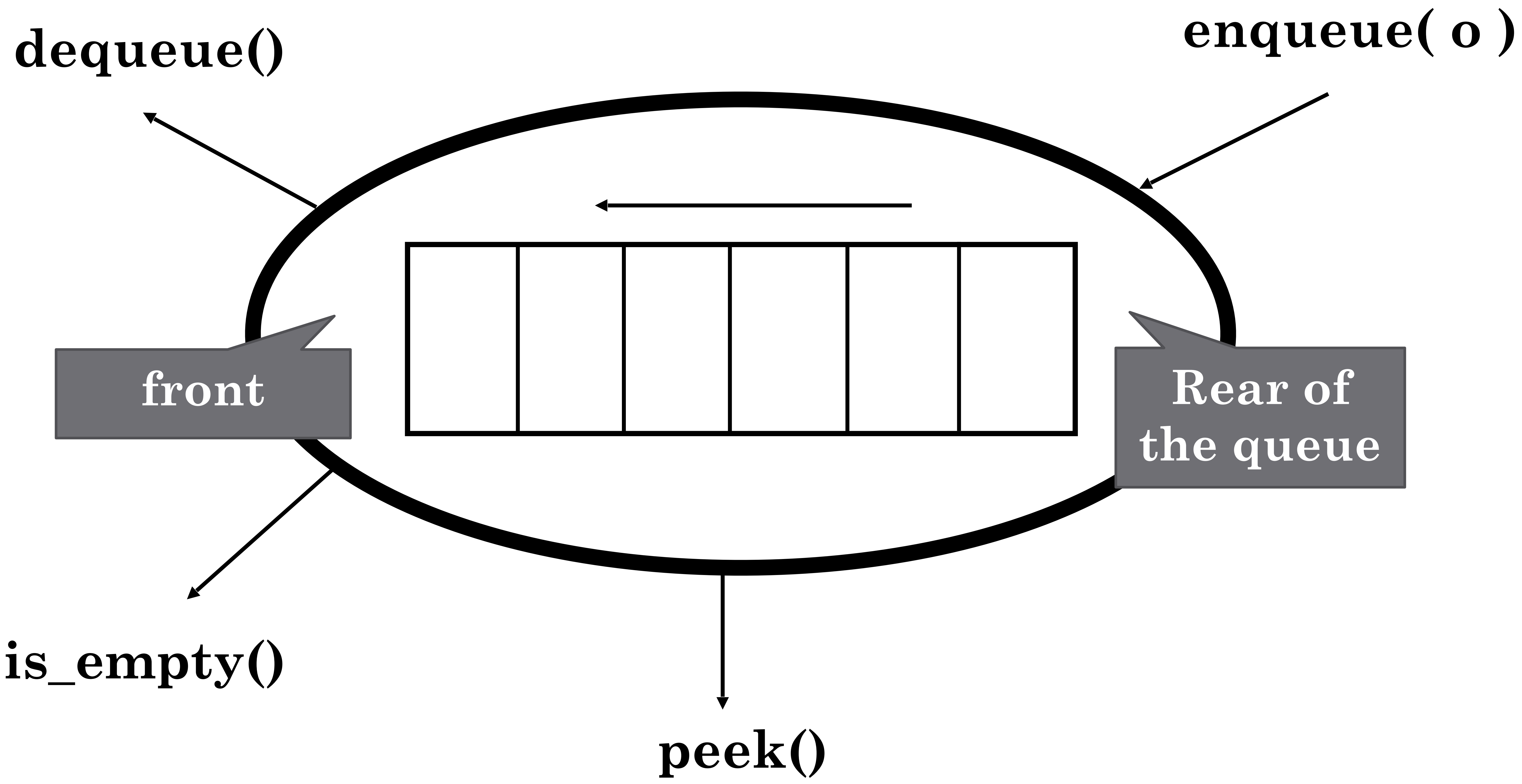
@SDS

Queue

- Queues are appropriate for many real-world situations
  - Example: A line to buy a movie ticket
  - Computer applications, e.g. a request to print a document
- A queue is an ordered collection of items where the addition of new items happens at one end (the rear or back of the queue) and the removal of existing items always takes place at the other end (the front of the queue).
  - New items enter at the back, or rear, of the queue
  - Items leave from the front of the queue
  - First-in, first-out (FIFO) property:
    - The first item inserted into a queue is the first item to leave

More formally

- Queues implement the FIFO (first-in first-out) policy:
  - e.g. the printer / job queue!



ADT Queue Operations

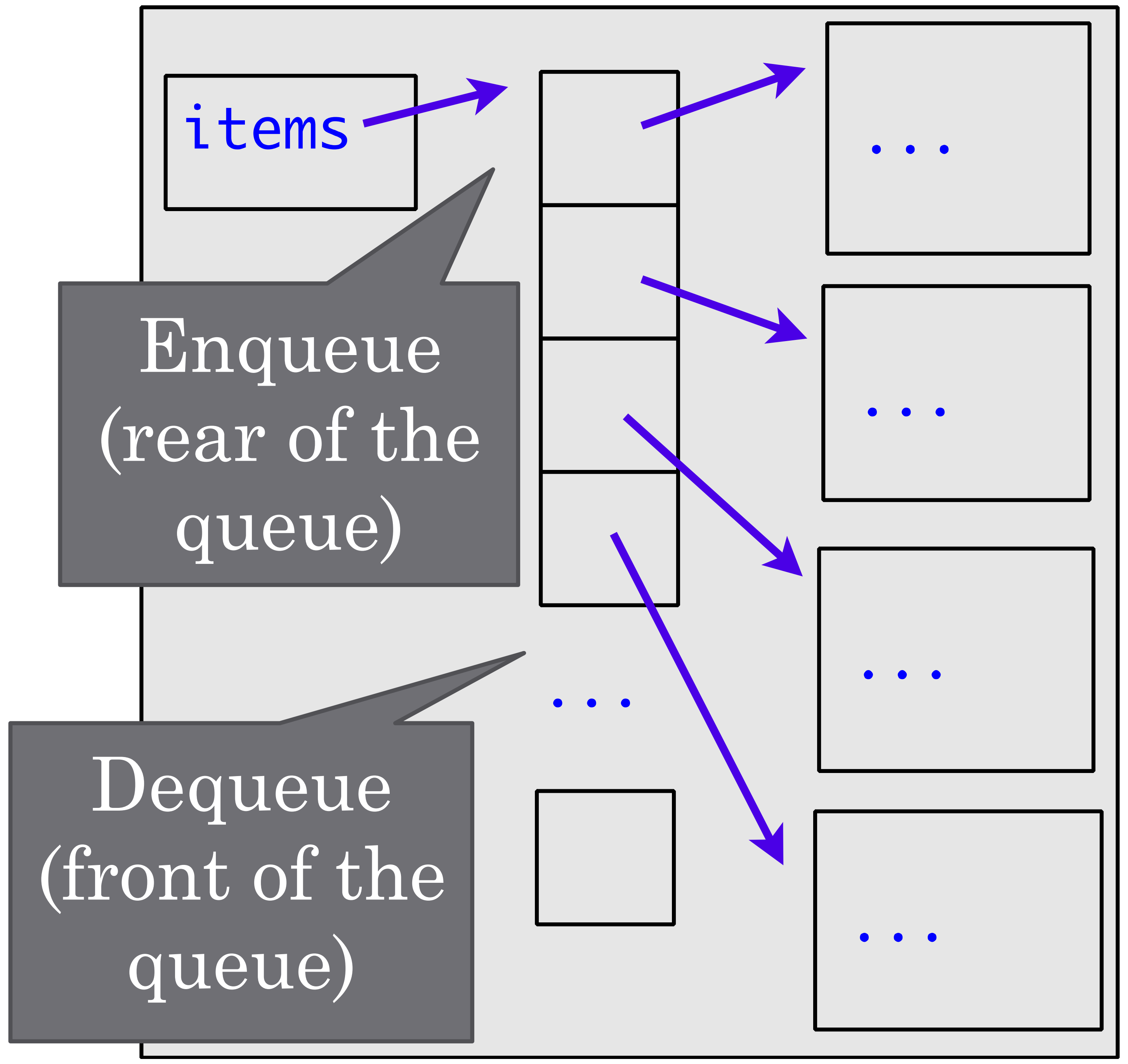
- What are the operations which can be used with a Stack Abstract Data?
    - Create an empty queue:
    - Determine whether a queue is empty:
    - Add a new item to the queue:
        - enqueue
    - Remove from the queue the item that was added earliest:
        - dequeue
    - Retrieve from the queue the item that was added earliest:
        - peek

The Queue Abstract Data Type

- Queue() creates a new queue that is empty.
  - It needs no parameters and returns an empty queue.
- enqueue(item) adds a new item to the rear of the queue.
  - It needs the item and returns nothing.
- dequeue() removes the front item from the queue.
  - It needs no parameters and returns the item. The queue is modified.
- is_empty() tests to see whether the queue is empty.
  - It needs no parameters and returns a boolean value.
- size() returns the number of items in the queue.
  - It needs no parameters and returns an integer.

The Queue Implementation In Python

- We use a python List data structure to implement the queue.
  - The addition of new items takes place at the beginning of the list
  - The removal of existing items takes place at the end of the list



```
class Queue:

   def __init__(self):
        self.items = []
   def is_empty(self):
        return self.items == []
   def size(self):
        return len(self.items)

   def enqueue(self, item):
        self.items.insert(0,item)
   def dequeue(self):
       return self.items.pop()
```

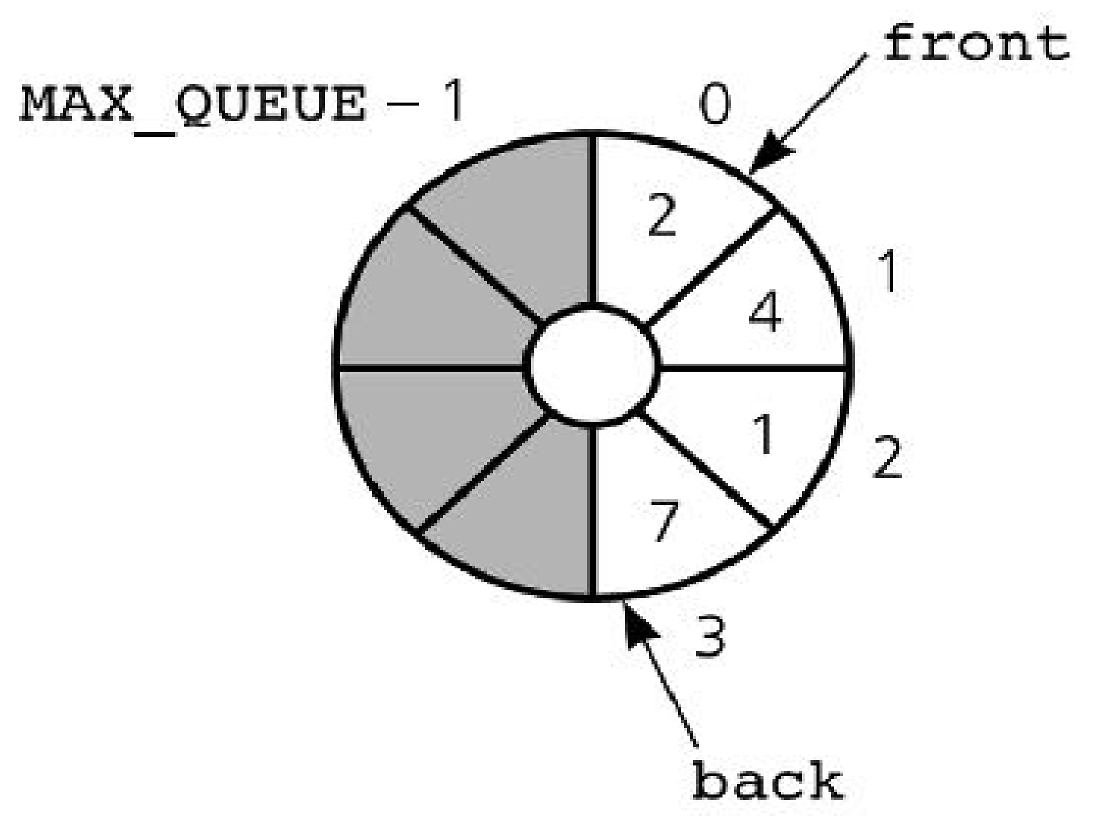
The Queue ADT Code Example – Result

- Queue operations, state of the queue, values returned

q = Queue()		
q.is_empty()	[]	True
q.enqueue(4)	[4]	
q.enqueue('dog')	['dog',4]	
q.enqueue(True)	[True,'dog',4]	
q.size()	[True,'dog',4]	3
q.is_empty()	[True,'dog',4]	False
q.enqueue(8.4)	[8.4,True,'dog',4]	
q.dequeue()	[8.4,True,'dog']	4
q.dequeue()	[8.4,True]	'dog'
q.size()	[8.4,True]	2

Circular Queue

- What is the Big-O performance of enqueue and dequeue of the implementation using Python List?
  - enqueue(…): O(n)
    - Shifting array elements to the right after each addition – too Expensive!
  - dequeue() : O(1)
- Another Implementation: Circular Queue
  - enqueue & dequeue : O(1)
    - Items can be added/removed without shifting the other items in the process



Circular Queue - Set up

- Uses a Python list data structure to store the items in the queue.
- The list has an initial capacity (all elements None)
- Keeps an index of the current front of the queue and of the current back of the queue.
    - set front to 0,
    - set back to MAX_QUEUE – 1,
    - set count to 0
- New items are enqueued at the back index position
- Items are dequeued at the front index position.
- A count of the queue items to detect queue-full and queue-empty conditions

Circular Queue - How To Advance

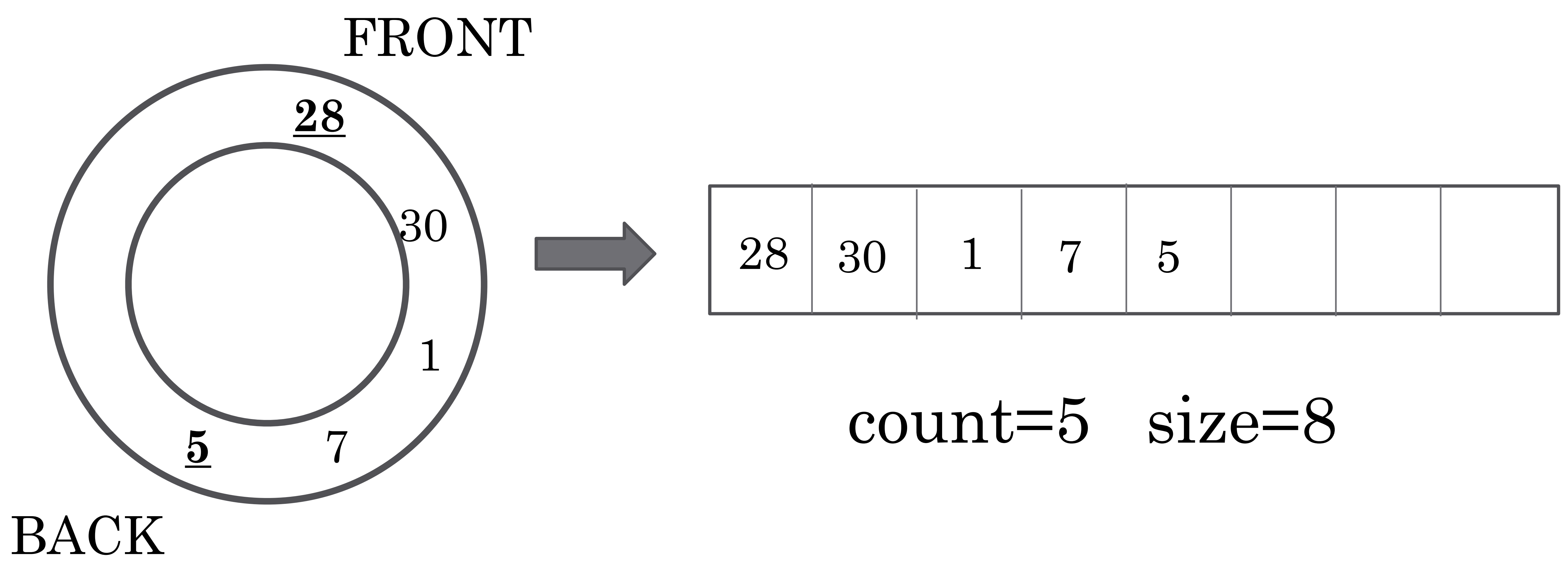
- Queue-empty:
  - front is one slot ahead of back
- When either front or back advances past MAX_QUEUE - 1, it wraps around to 0
  - The wrap-around effect: by using Modulus (%) arithmetic operator
- enqueue
  - If it is not full

```
self.back = (self.back + 1) %
self.MAX_QUEUE
self.items[self.back] = item
self.count += 1
```

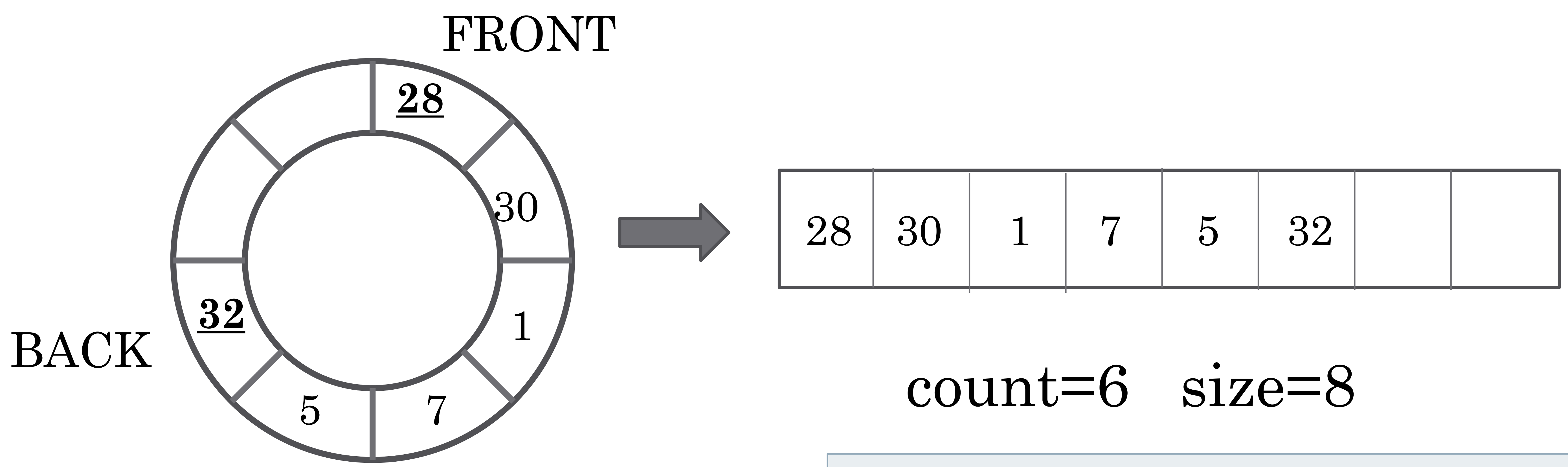
- dequeue
  - If it is not empty

```
item = self.items[self.front]
self.front = (self.front + 1) %
self.MAX_QUEUE
self.count -= 1
return item
```

Circular Queue - enqueue(32)



count=5 size=8

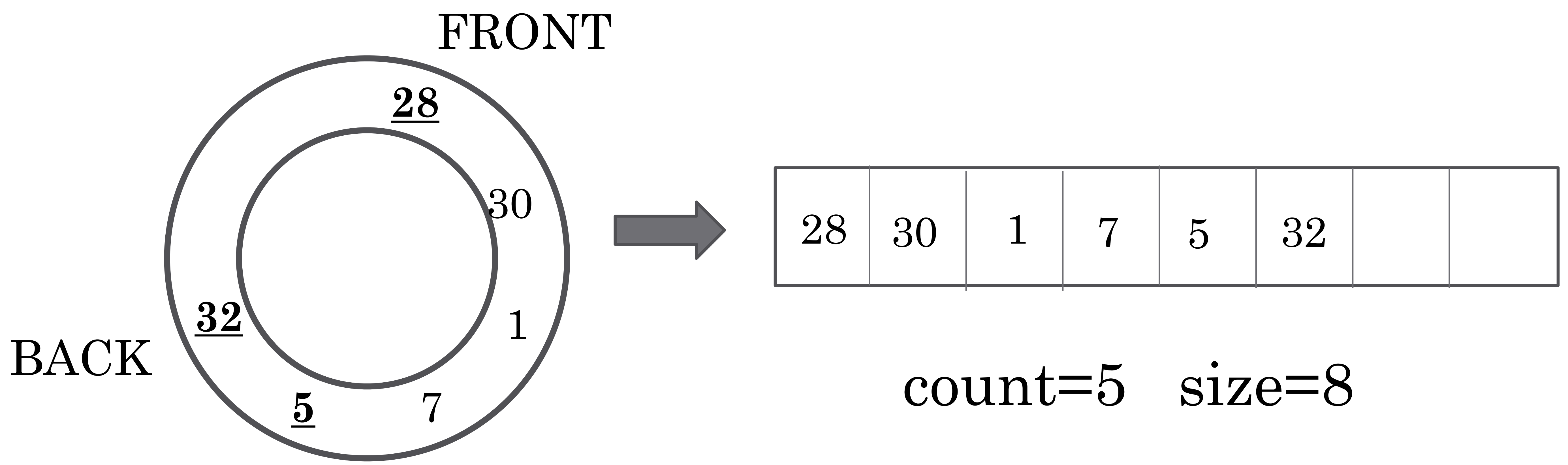


count=6 size=8

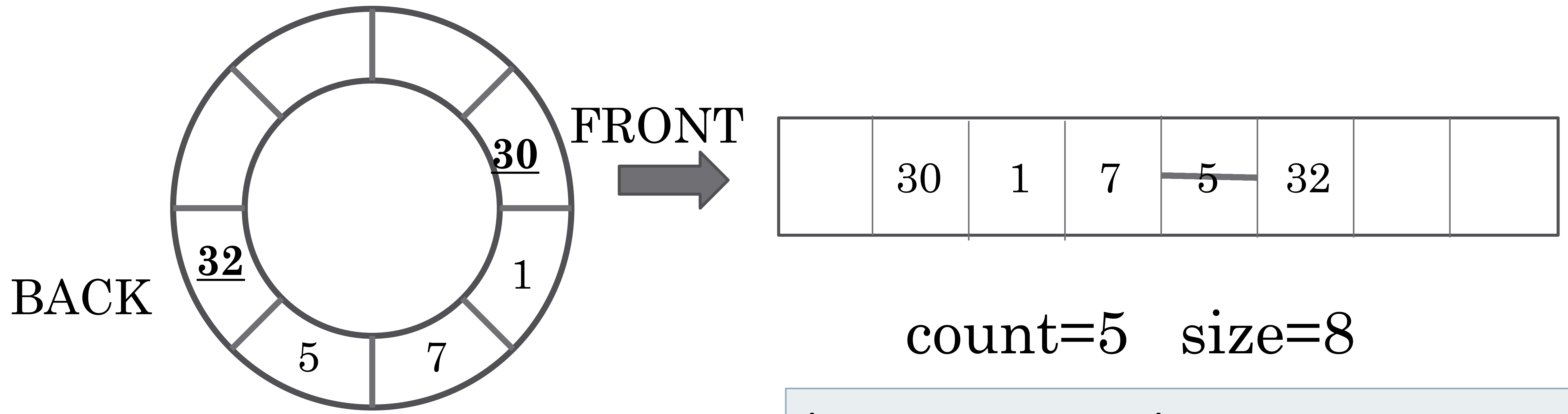
New item is inserted at the position following back
back is advanced by one position
count is incremented by 1

```
self.back = (self.back + 1)
% self.MAX_QUEUE
self.items[self.back] =
item
self.count += 1
```

Circular Queue - dequeue()



count=5   size=8

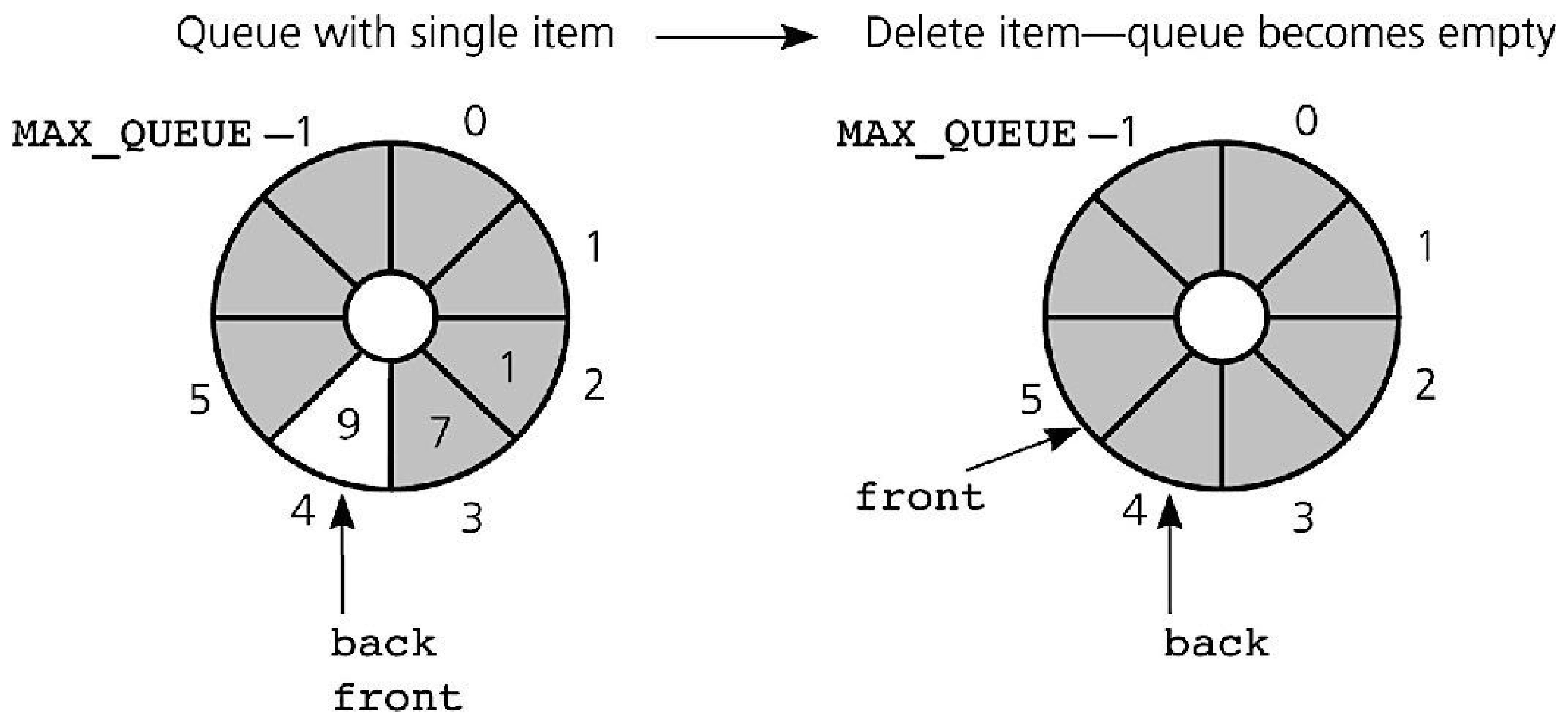


count=5   size=8

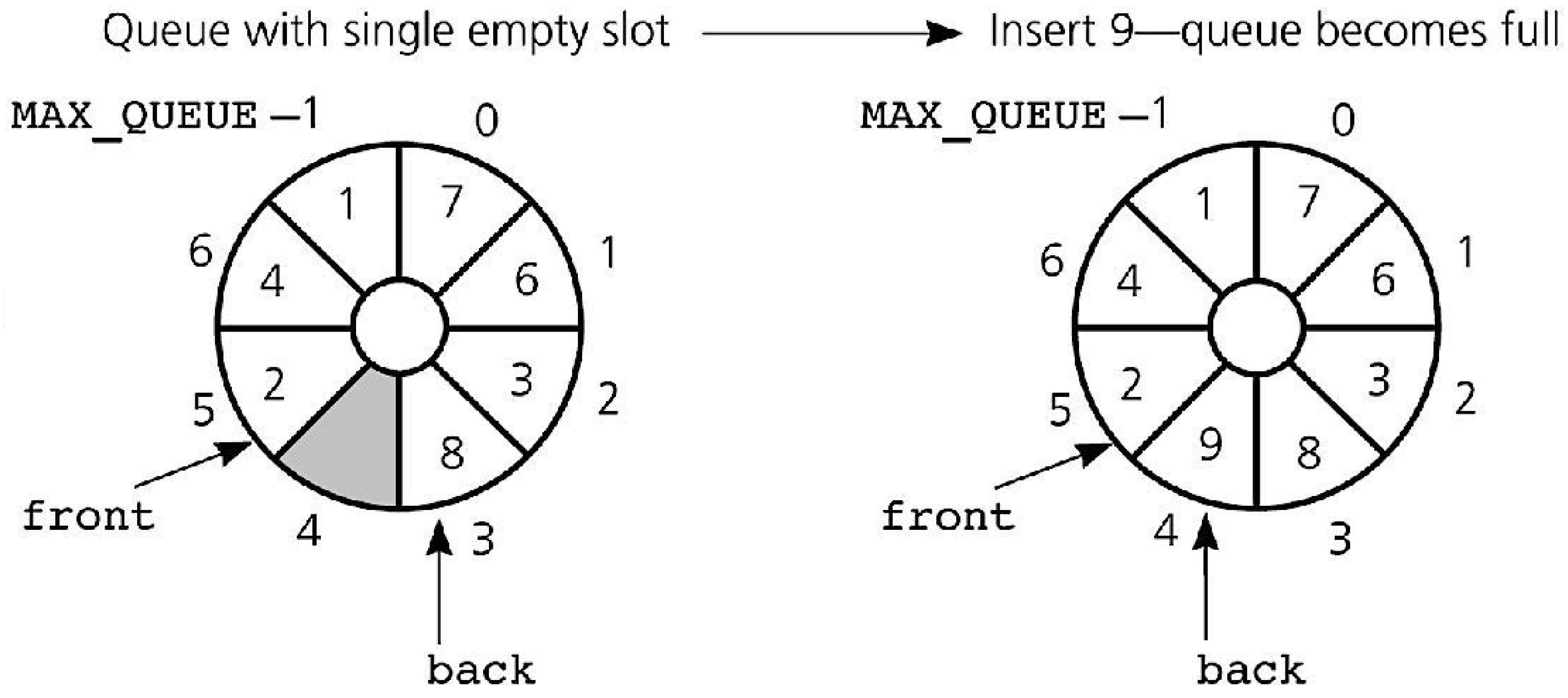
```
item = self.items[self.front]
self.front = (self.front + 1) %
                 self.MAX_QUEUE
self.count -= 1
return item
```

Circular Queue: Empty ? Full?

- front and back cannot be used to distinguish between queue-full and queue-empty conditions for a circular array
- Case 1:



Circular Queue: Empty ? Full?



```
def is_empty():
   return self.count ==
0
```

```
def is_full():
   return self.MAX_QUEUE <=
self.count
```

Deque Abstract Data Type

- Deque - Double Ended Queue
  - A deque is an ordered collection of items where items are added and removed from either end, either front or back.
- The newest item is at one of the ends

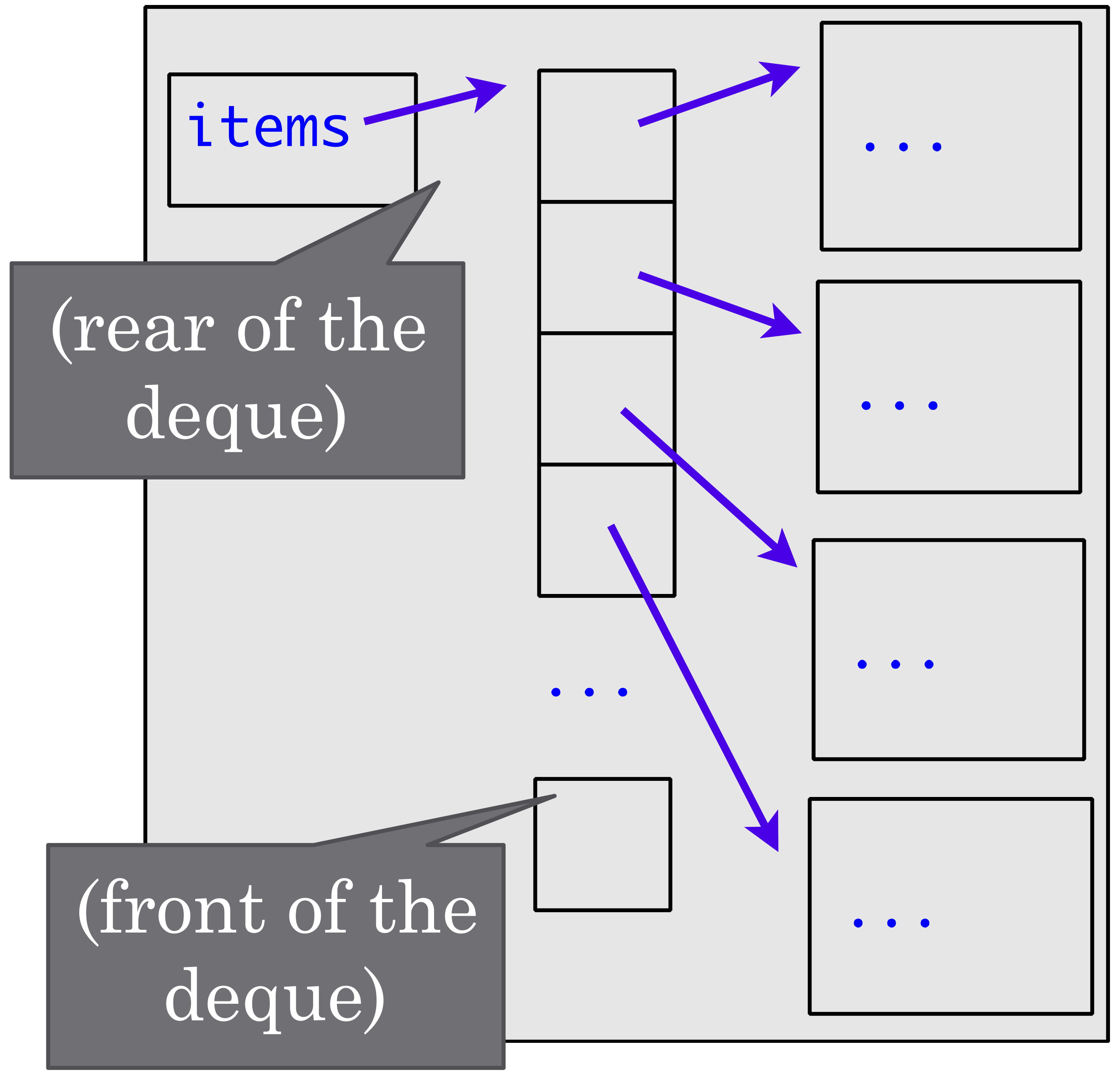
ADT Deque Operations

- What are the operations which can be used with a Deque Abstract Data?
  - Create an empty deque:
  - Determine whether a deque is empty:
  - Add a new item to the deque:
    - add_front()
    - add_rear()
  - Remove from the deque the item that was added earliest:
    - remove_front()
    - remove_rear()

Deque In Python

- We use a python List data structure to implement the deque.

```
class Deque:

   def __init__(self):
     self.items = []
   ...

   def add_front(self, item):
     self.items.append(item)

   def add_rear(self, item):
     self.items.insert(0,item)

   def remove_front(self):
     return self.items.pop()

   def remove_rear(self):
     return self.items.pop(0)
```



Deque Code Example – Result

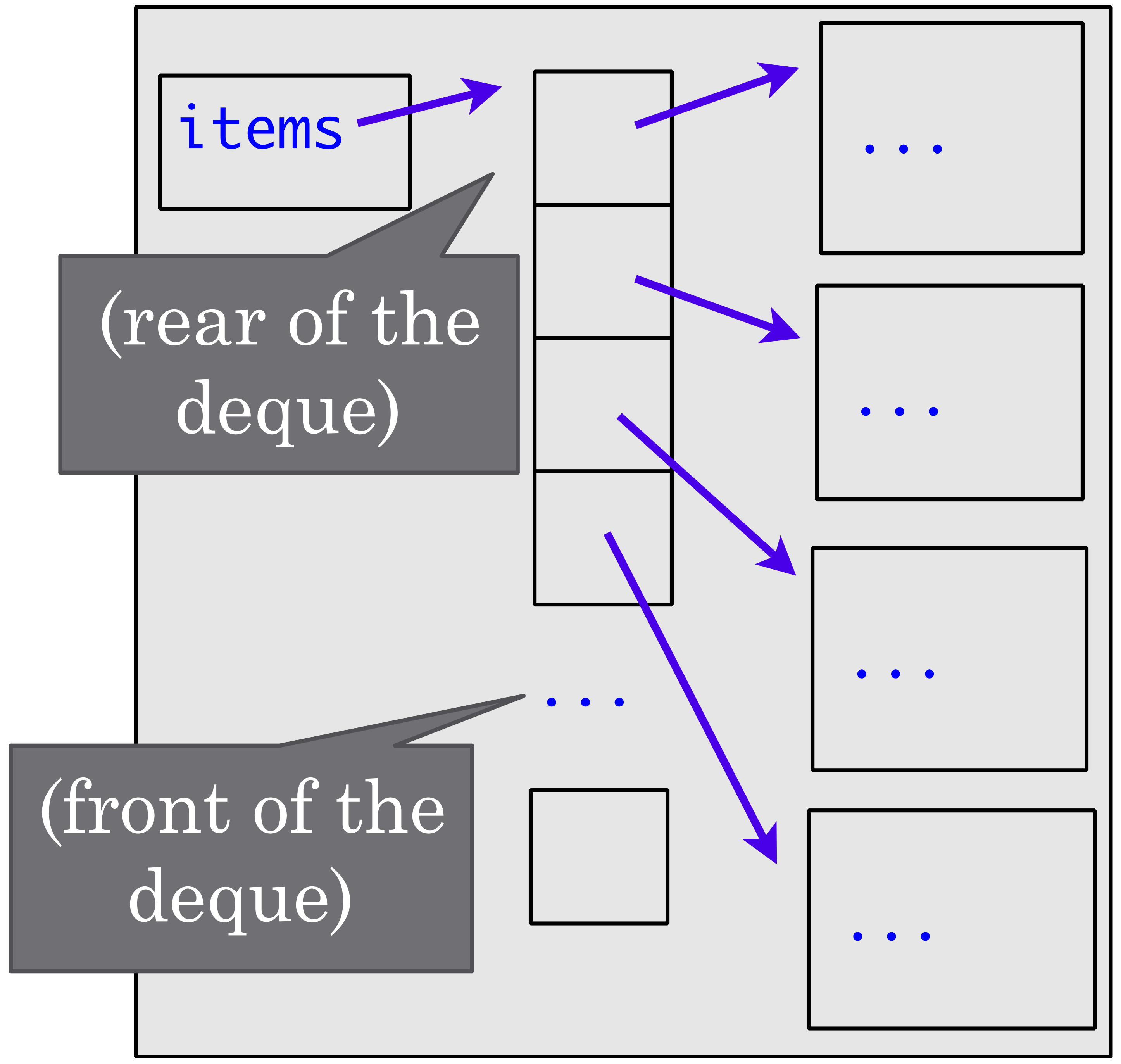
- Deque operations, state of the deque, values returned

d = Deque()		
d.is_empty()	[]	True
d.add_rear(4)	[4]	
d.add_rear('dog')	['dog',4,]	
d.add_front('cat')	['dog',4,'cat']	
d.add_front(True)	['dog',4,'cat',True]	
d.size()	['dog',4,'cat',True]	4
d.is_empty()	['dog',4,'cat',True]	False
d.add_rear(8.4)	[8.4,'dog',4,'cat',True]	
d.remove_rear()	['dog',4,'cat',True]	8.4
d.remove_front()	['dog',4,'cat']	True

Deque Big-O Performance

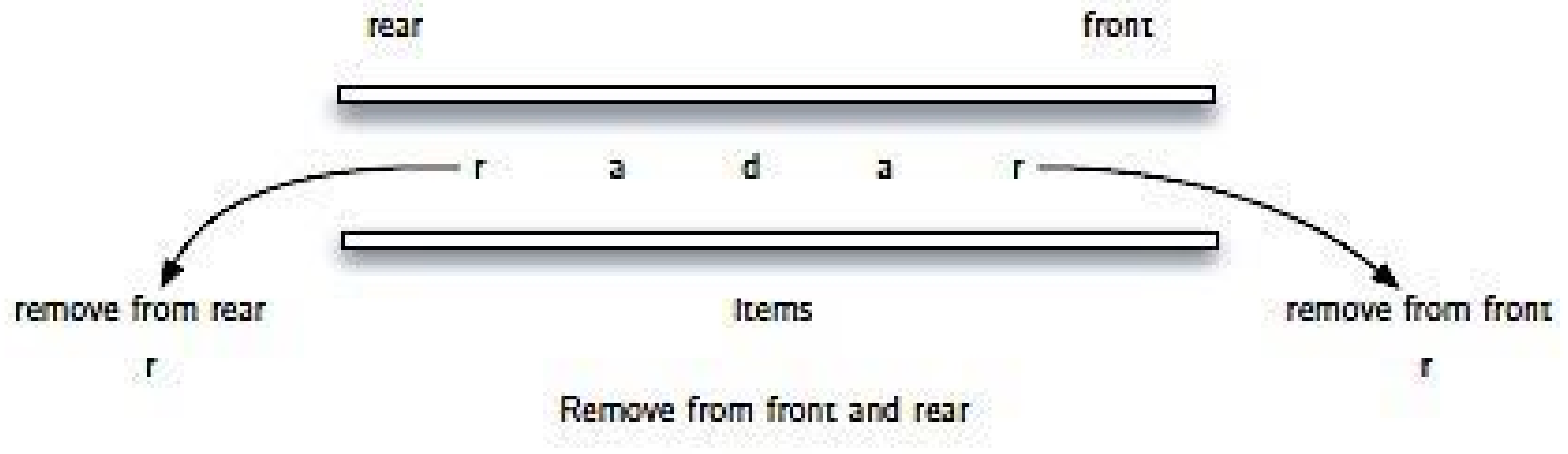
- O(1)
  - add_front()
  - remove_front()
- O(n)
  - add_rear()
  - remove_rear()

```
def add_front(self, item):
    self.items.append(item)

def add_rear(self, item):
    self.items.insert(0,item)

def remove_front(self):
    return self.items.pop()

def remove_rear(self):
    return self.items.pop(0)
```



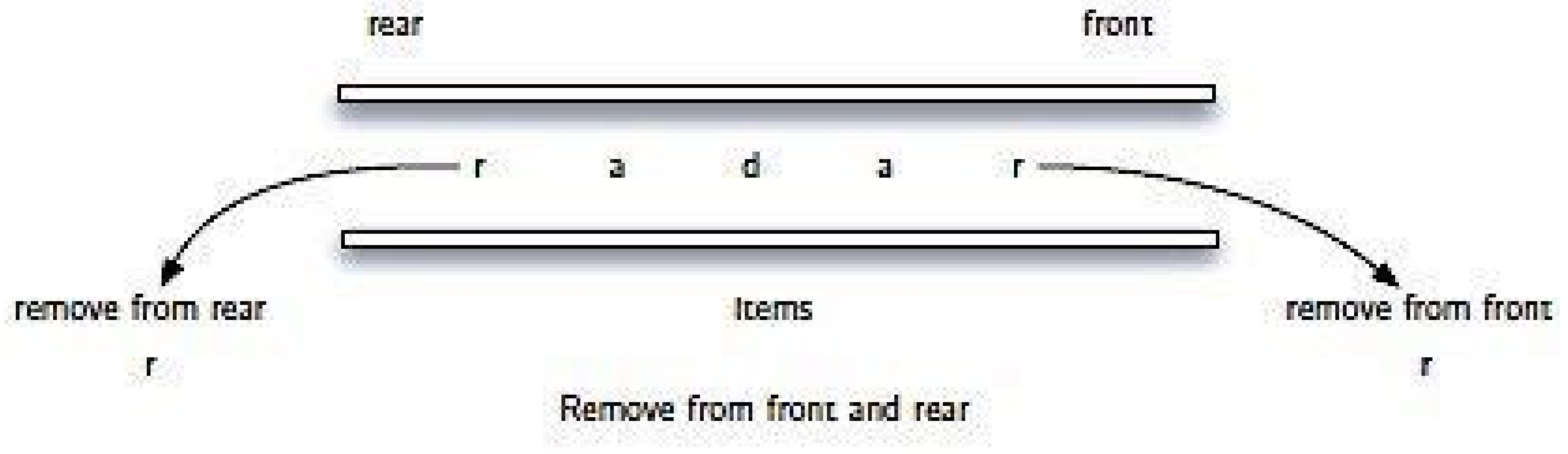
Deque Application: Palindrome Checker

- A string which reads the same either left to right, or right to left is known as a palindrome
  - radar
  - deed
  - A dog, a plan, a canal: pagoda.



Deque
Palindrome Checker - Algorithm

- Create a deque to store the characters of the string
  - The front of the deque will hold the first character of the string and the rear of the deque will hold the last character
- Remove both of them directly, we can compare them and continue only if they match.
  - If we can keep matching first and the last items, we will eventually either run out of characters or be left with a deque of size 1
    - In either case, the string must be a palindrome.



Summary

- The definition of the queue operations gives the ADT queue first-in, first-out (FIFO) behaviour
- To distinguish between the queue-full and queue-empty conditions in a queue implementation that uses a circular array,
  - count the number of items in the queue
- Models of real-world systems often use queues

Questions?

@rschifan

schifane@di.unito.it

http://www.di.unito.it/~schifane