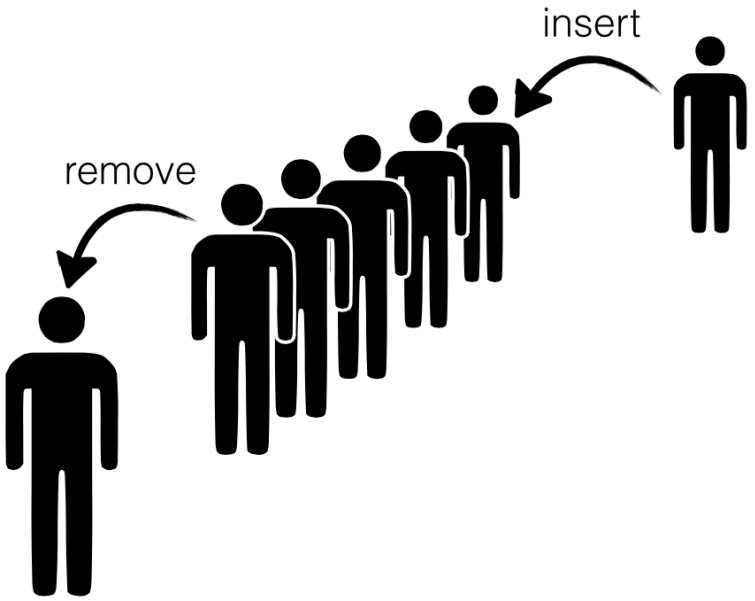
COSC 2306
Data Programming

Queue

Queues



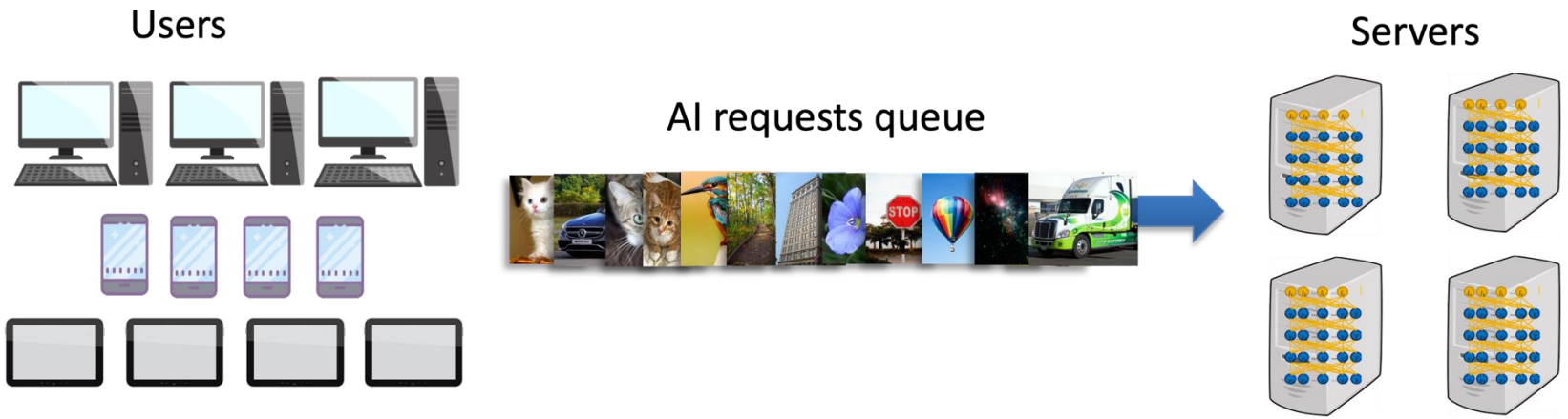
- Stacks were LIFO structures (last in-first out)
- Queues are FIFO (first in-first out)

Priority queues

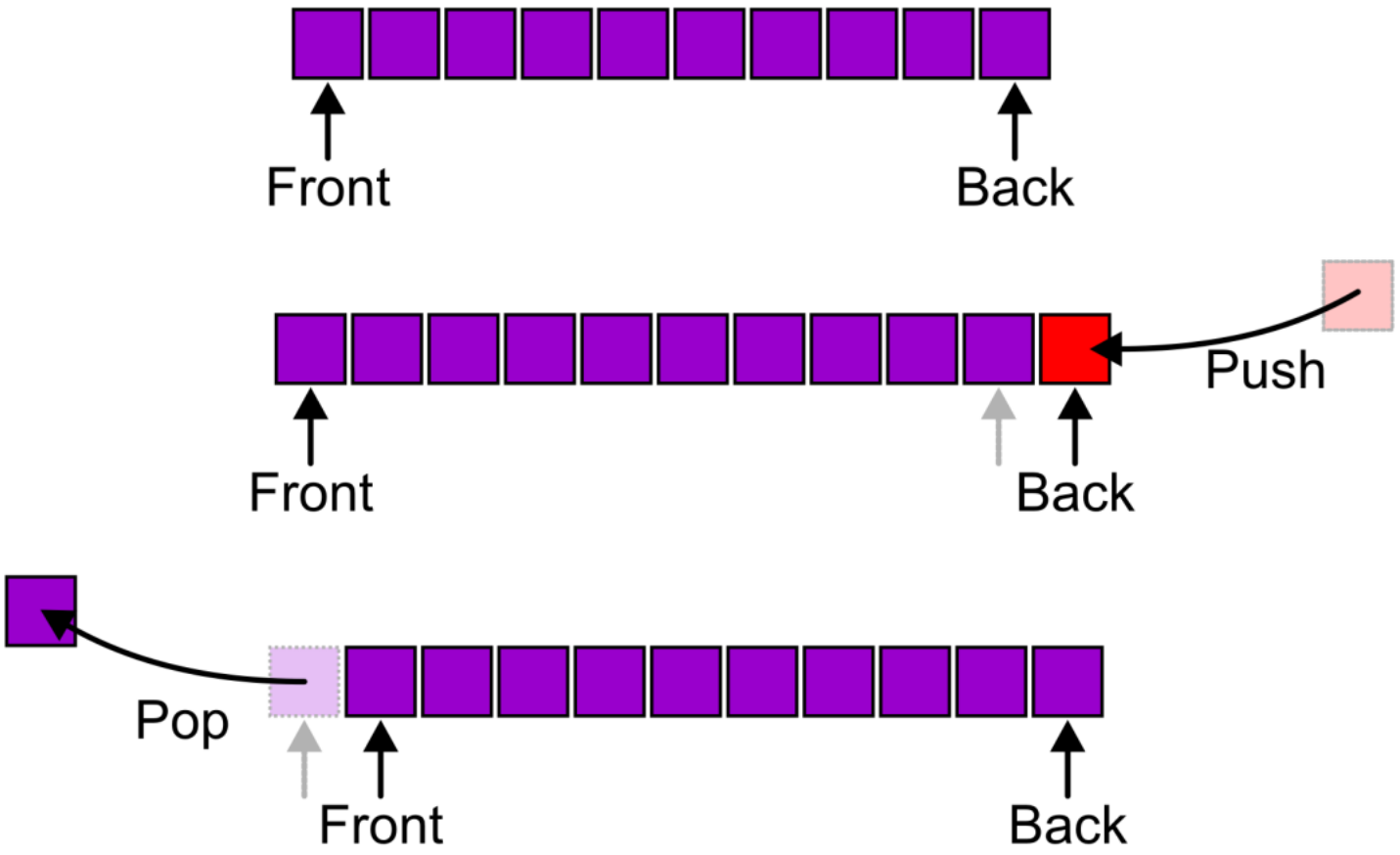
- Queue structure ensures items processed in the order received
- Priority queues
  - Customers (jobs) with higher priority pushed to the front of the queue

Applications of queues

- Everything that requires scheduling of operations/requests
- Queuing systems
  - Computer simulations where queues represent the basic data structure
  - Queues of objects wait to be served by various servers
  - Consist of servers and queues of objects waiting to be served



Queue data structure

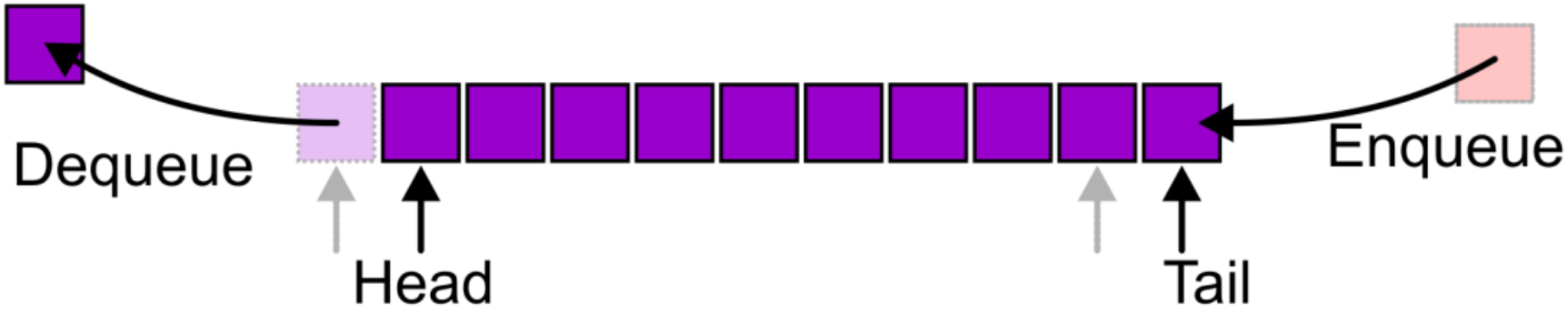


Queue data structure

- Elements added at one end (rear), deleted from the other end (front)
- Middle elements inaccessible

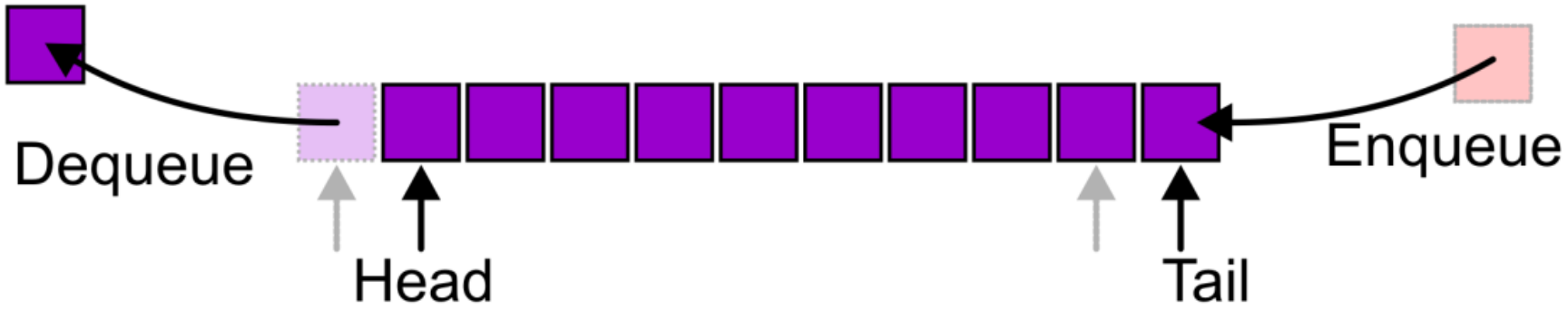
- Two key operations
  - **enqueue**
  - **dequeue**
- Additional operations
  - **isEmpty**, **size**

Queue data structure



Implementing a queue in Python

- Like stack implementation
- Use Python list as a base
- Insert and remove (pop) elements in 2 different sides



Simple list implementation

```
my_queue = []
my_queue.append(1) #enqueue
my_queue.append(2)
first = my_queue.pop(0) #dequeue
print(first)
print(my_queue)


>1
>[2]
```

Implementing a queue in Python

```
class Queue:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        if self.isEmpty():
            return None
        return self.items.pop(0)
    def size(self):
        return len(self.items)
```

Another Implementation

```
def enqueue(self, item):
    self.items.insert(0,item)
def dequeue(self):
    if self.isEmpty():
        return None
    return self.items.pop()
```

Time Complexity

- The current implementations are not good enough
  - insert(0) and pop(0) cost O(n)
  - Need to rearrange all elements in the list
- Better choice: Linked List

Queue Example

```
shop_queue = Queue() # A shop
people = [ "AAA" , "BBB", "CCC" ]
# 3 people want to buy products

for person in people:
    shop_queue.enqueue(person) # they get in queue

While not shop_queue.isEmpty():
    print("Serving ", shop_queue.dequeue())
    # Serve people in the queue 1 by 1
```

Output:

Serving AAA
Serving BBB
Serving CCC

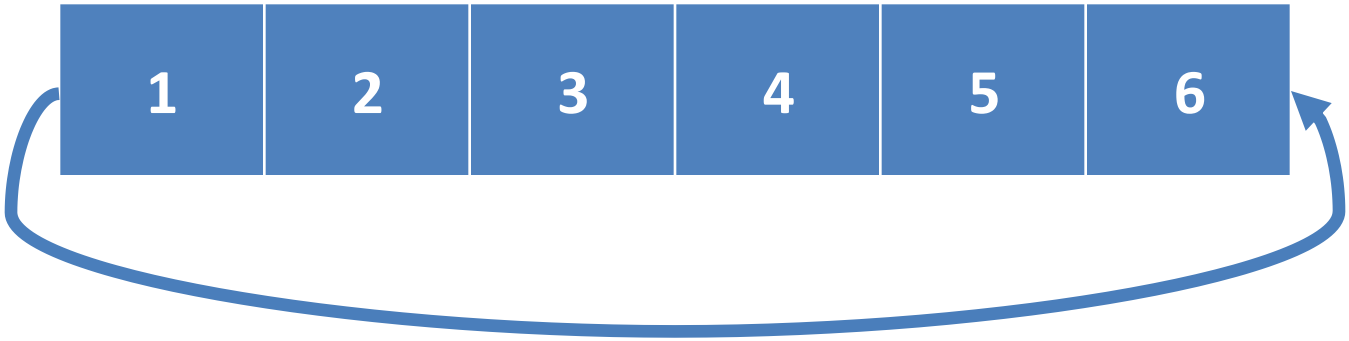
Queue Example

```
task_queue = Queue()
tasks = [ "read textbook" , "do assignment", "write
code" ]
# 3 tasks

for task in tasks:
    tasks_queue.enqueue(task) # Assigned 1 by 1

While not tasks_queue.isEmpty():
    print("Working on ", tasks_queue.dequeue())
    # Do the tasks in the order of request
```

Output:

Working on read textbook
Working on do assignment
Working on write code

Moving Items in a Circle



2	3	4	5	6	1

3	4	5	6	1	2

4	5	6	1	2	3

Moving Items in a Circle

```
def CircularShift(mylist, num):
    myq = Queue()
    for i in mylist:
        myq.enqueue(i)                  #[1, 2, 3, 4, 5, 6]

    ...

                                        #[4, 5, 6, 1, 2, 3]
    return myq.dequeue()

list = [1, 2, 3, 4, 5, 6]
result=CircularShift(list, 3)
print(result)                           #4
```

Moving Items in a Circle

```
def CircularShift(mylist, num):
    myq = Queue()
    for i in mylist:
        myq.enqueue(i)                  #[1, 2, 3, 4, 5, 6]

    if not myq.isEmpty():
        for i in range(num):
            item = myq.dequeue()
            myq.enqueue(item)
                                        #[4, 5, 6, 1, 2, 3]
    return myq.dequeue()


list = [1, 2, 3, 4, 5, 6]
result=CircularShift(list, 3)
print(result)                           #4
```

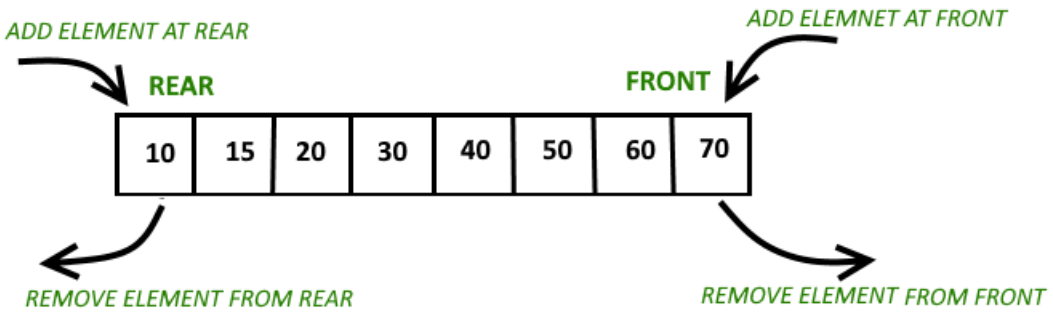
Reverse First K Elements – Stacks/Queues

```
def firstKSQ():

    num_elements = int(input('How many elements? '))
    k = int(input('How many first elements to be reversed? '))

    mystack = Stack()
    myq = Queue()



    …



    all_elements = []
    while not mystack.isEmpty():
        all_elements.append(mystack.pop())
    while not myq.isEmpty():
        all_elements.append(myq.dequeue())
```

Reverse First K Elements – Stacks/Queues

```
def firstKSQ():

    num_elements = int(input('How many elements? '))
    k = int(input('How many first elements to be reversed? '))

    mystack = Stack()
    myq = Queue()
    count = 0
    for i in range(num_elements):
        val = int(input('Please enter element: '))
        if count < k:
            mystack.push(val)
        else:
            myq.enqueue(val)
        count += 1

    all_elements = []
    while not mystack.isEmpty():
        all_elements.append(mystack.pop())
    while not myq.isEmpty():
        all_elements.append(myq.dequeue())
```

deque



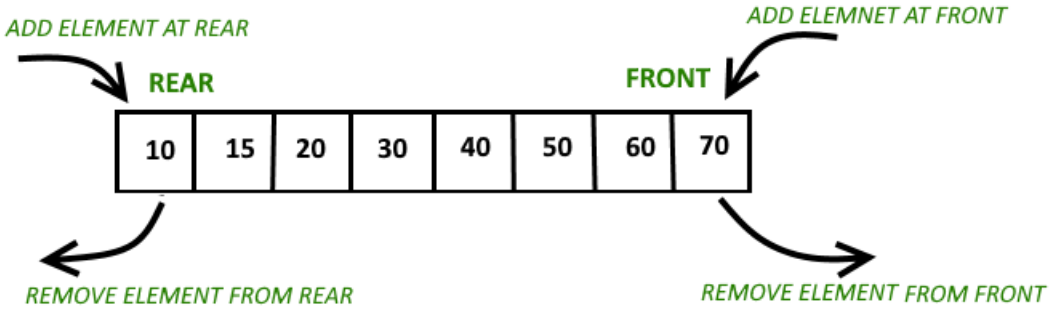
Append and pop on both ends

Faster than list – deque is O(1) vs list is O(n) for adding/removing elements

Operations:
- **append()**: insert argument to the right end of the deque
- **appendleft()**: insert argument to the left end of the deque
- **pop()**: delete an argument from the right end of the deque
- **popleft()**: delete an argument from the left end of the deque

- **index(ele, beg, end)**: return the first index of the value in arguments, starting searching from beg till end index
- **insert(i, a)**: insert the value in arguments(a) at index(i) specified in arguments
- **remove()**: remove the first occurrence of the value in arguments
- **count()**: count the number of occurrences of value in arguments

deque



Example:

```
from collections import deque

# Declaring deque
queue = deque(['name','age','DOB'])

print(queue)
```

> deque(['name', 'age', 'DOB'])

deque

```
import collections
de = collections.deque([1, 2, 3, 3, 4, 2, 4])

# using index() to print the first occurrence of 4
print ("The number 4 first occurs at a position : ")
print (de.index(4,2,5))
> The number 4 first occurs at a position : 4

# using insert() to insert the value 3 at 5th position
de.insert(4,3)
print ("The deque after inserting 3 at 5th position is : ")
print (de)
> The deque after inserting 3 at 5th position is : deque([1, 2, 3, 3, 3, 4, 2, 4])

# using count() to count the occurrences of 3
print ("The count of 3 in deque is : ")
print (de.count(3))
> The count of 3 in deque is : 3

# using remove() to remove the first occurrence of 3
de.remove(3)
print ("The deque after deleting first occurrence of 3 is : ")
print (de)
> The deque after deleting first occurrence of 3 is : deque([1, 2, 3, 3, 4, 2, 4])
```

Stack implementation using deque

```
from collections import deque  #Doubly Ended Queue
stack = deque()
print(stack)
# append() function to push element in the stack
stack.append('a')
stack.append('b')
stack.append('c')

# pop() function to pop element from stack in LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)
```

```
> deque(['a', 'b', 'c'])
> Elements popped from stack:
> c
> b
> a
> Stack after elements are popped: deque([])
```

Queue implementation using deque

```
from collections import deque  #Doubly Ended Queue
queue = deque()
print(queue)
# append() function to enqueue element to a queue
queue.append('a')
queue.append('b')
queue.append('c')

# remove elements from queue in FIFO order
print('\nElements dequeued from a queue:')
print(queue.popleft())
print(queue.popleft())
print(queue.popleft())

print(‘\nQueue after elements are removed:')
print(queue)
```

```
> deque(['a', 'b', 'c'])
> Elements dequeued from a queue:
> c
> b
> a
> Queue after elements are removed: deque([])
```