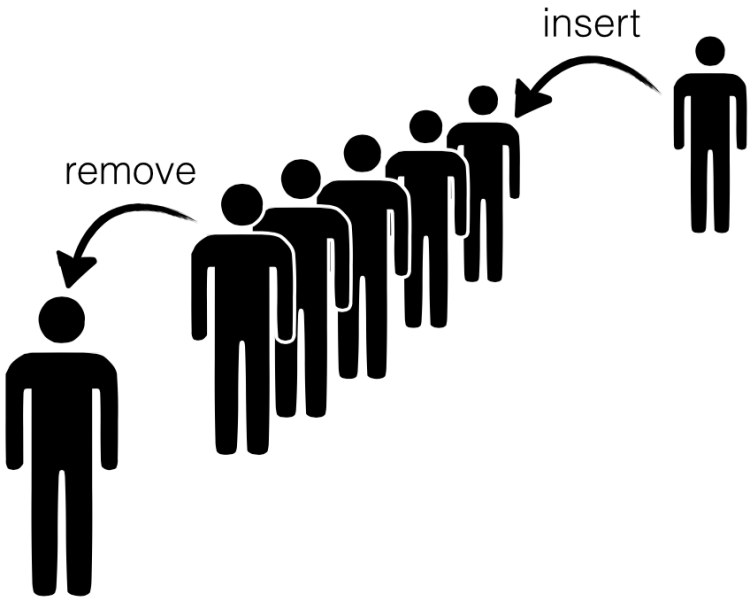
COSC 2306
Data Programming

Queue

Queues



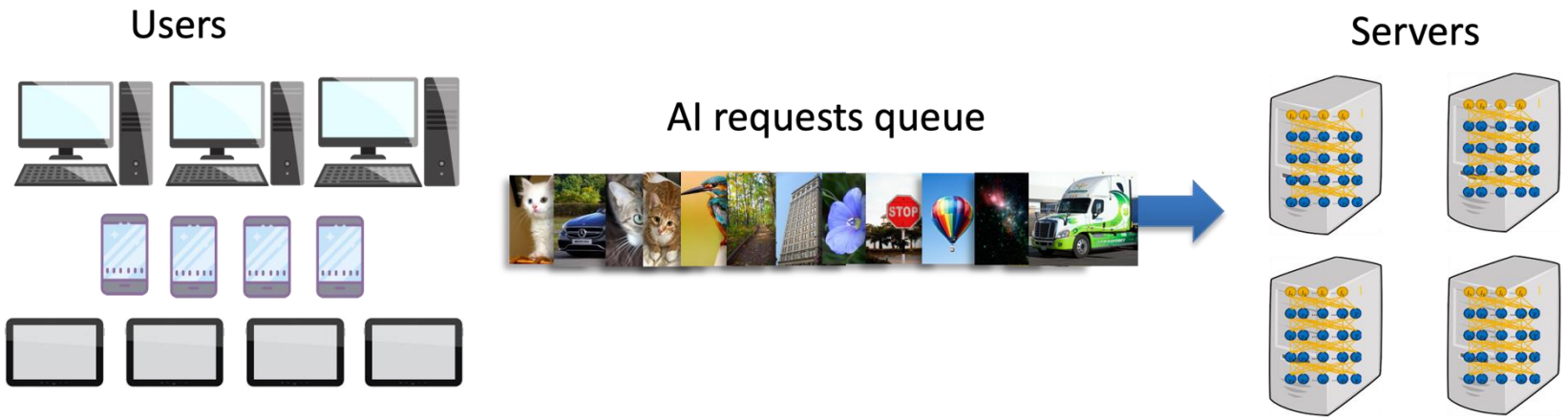
- Stacks were LIFO structures (last in-first out)
- Queues are FIFO (first in-first out)

Priority queues

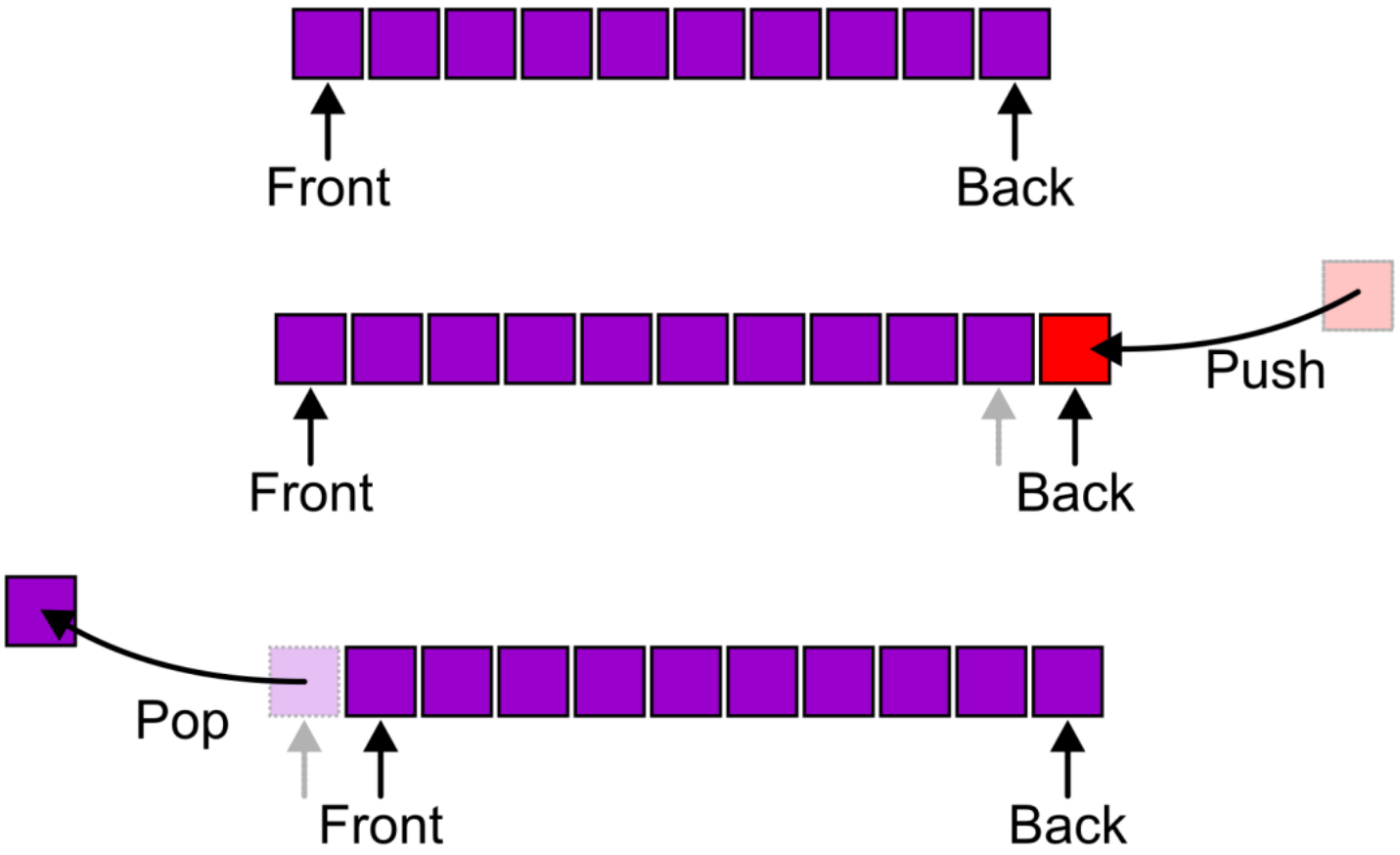
- Queue structure ensures items processed in the order received
- Priority queues
  - Customers (jobs) with higher priority pushed to the front of the queue

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

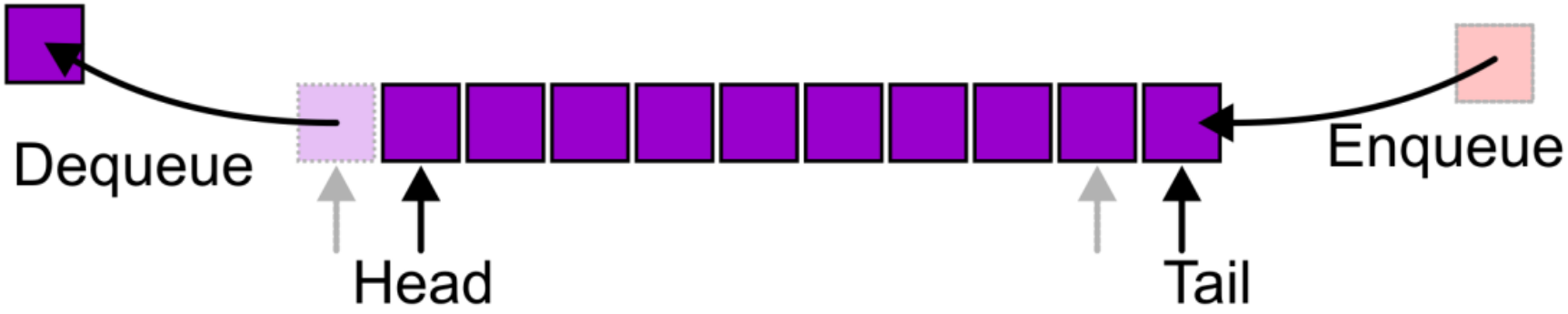


Queue data structure

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

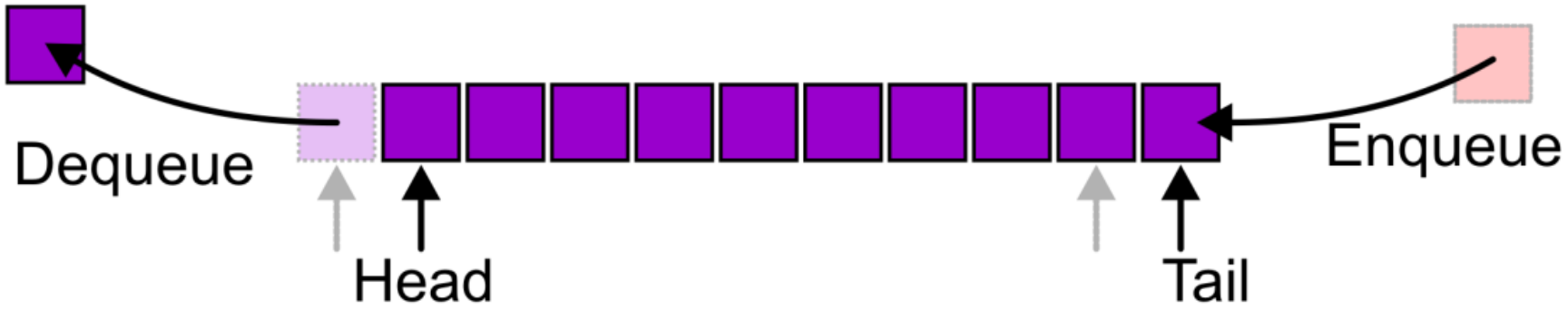
- Two key operations
  - **enqueue**
  - **dequeue**
- Additional operations
  - **isEmpty**, **size**

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

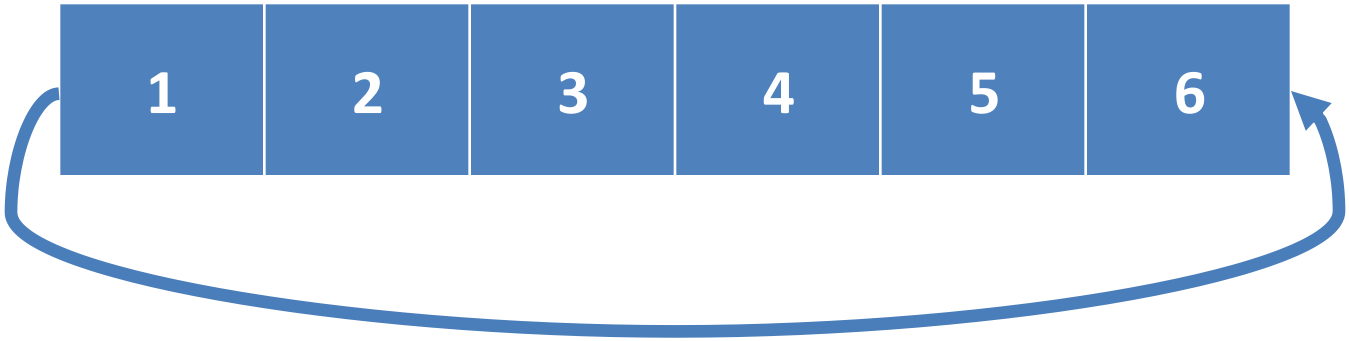
Queue Example

```
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 shop_queue.isEmpty():
    print("Working on ", shop_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
```

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())
```