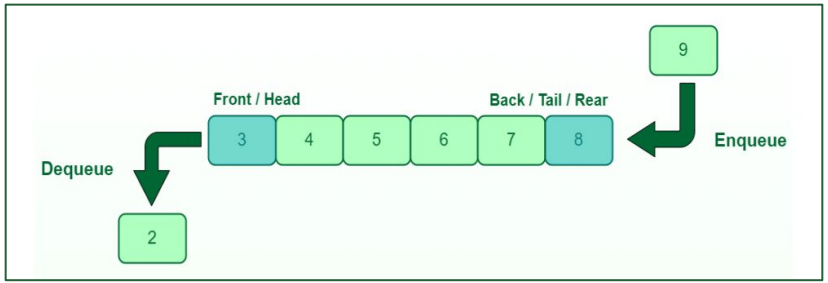
CMPUT 175

UNIVERSITY OF ALBERTA

Queues

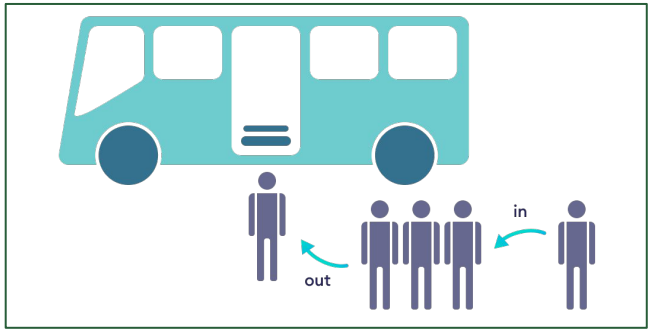
- It’s a linear data structure that stores items in **First In First Out (FIFO)** manner-
  - So, the item added first is removed first
  - **Enqueue** (used to add an item to the tail)
  - **Dequeue** (used to remove an item from the head)



- **Example:** any queue of consumers for a resource where the consumer that came first is served first

Uses

- **Processing orders:** The orders that are added first are dealt with first
- **Project management:** A project usually has a lot of tasks within it, so we can queue these to make sure to handle the most important tasks before the rest
- **CPU scheduling:** Managing requests on a single shared resource where jobs can be divided into categories based on priority and then processed in a FIFO order



Methods

- You can make there methods in your queue class using lists, or arrays
  - **enqueue()**, **dequeue()**, **isEmpty()**, **isFull()**, **capacity()**, **size()**, **clear()**
  - Store the queue items in a list and check the length to implement some of these methods

```
class Queue:
    #Creates a new empty queue
    def __init__(self, capacity):
        self.storage_list = []
        self.cap = capacity

    #check if queue is full
    def isFull(self):
        return len(self.storage_list)==self.cap
```

```
#returns the max capacity of queue
def capacity(self):
    return self.cap

#returns the number of items in queue
def size(self):
    return len(self.storage_list)

#clears the items currently in queue
def clear(self):
    self.storage_list = []
```

Methods

- You can make there methods in your queue class using lists, or arrays
  - **enqueue()**, **dequeue()**, **isEmpty(), isFull()**, **capacity()**, **size()**, **clear()**

```
class Queue:
    def enqueue(self, data):
        #check if queue is full before proceeding
        #else append data to storage list

    def dequeue(self):
        #check if queue is empty first
        #else remove data from storage list
```

Methods

```
#main

q = Queue(3) #making a queue of capacity 3

print("queue is empty: " + str(q.isEmpty()) + "\n")

q.enqueue("ace")
q.enqueue("fox")
print("size: " + str(q.size()) + "\n")

print(q.dequeue())
print("size: " + str(q.size()))
```

Output:

```
True
size: 2
ace
size: 1
```

Types

We will focus on two types of queues-

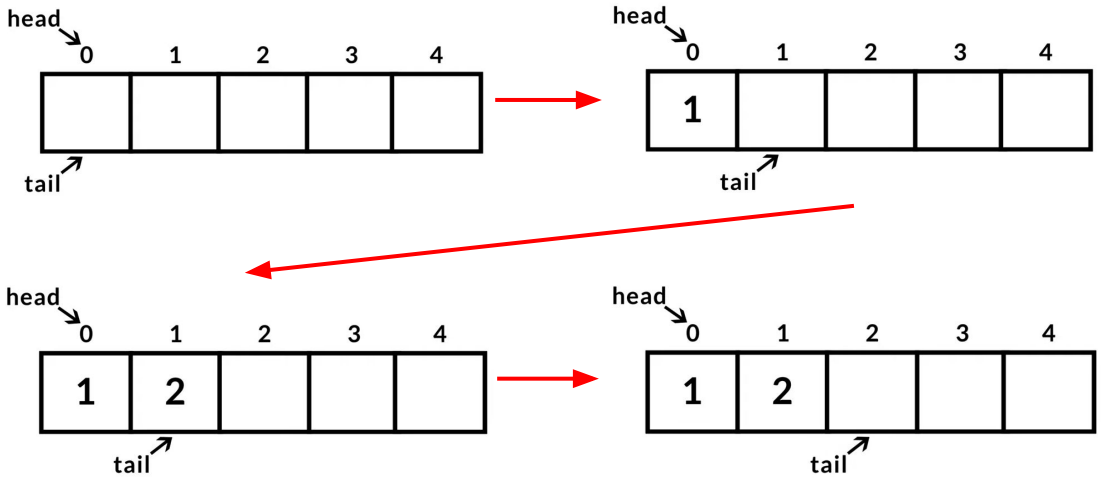
- **Bounded queue**:
  - is limited to a fixed number of items
  - so the capacity is fixed at creation and can't be changed
  - positions of the head and tail are fixed

- **Circular queue:**
  - is a bounded queue but
  - both the head and tail positions can change

Types: Bounded Queue

Bounded queue:

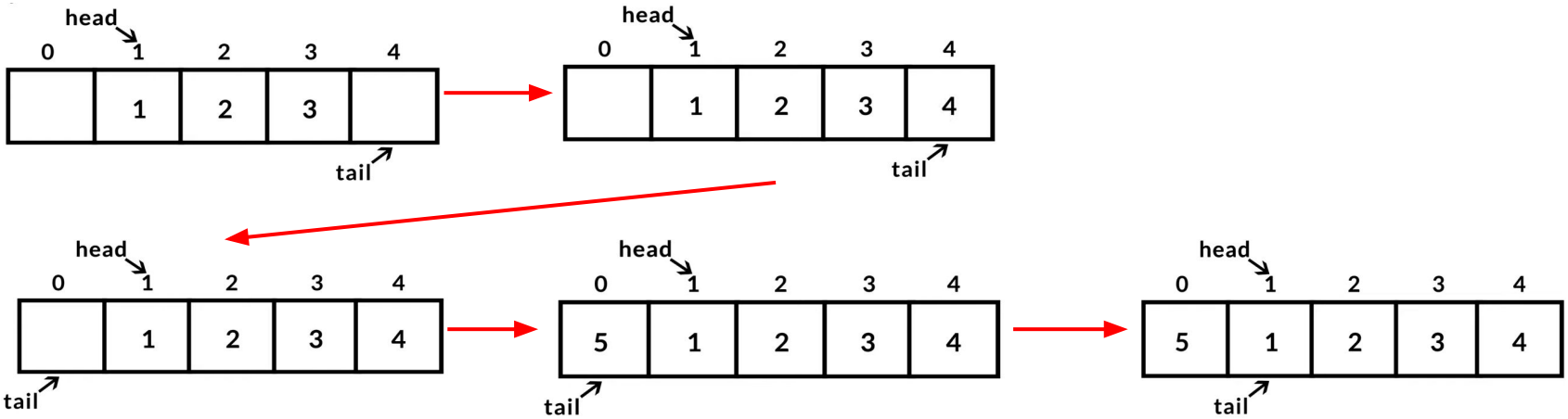
- **Enqueue:** head is always at position 0
- Here we enqueue items [1,2]



Types: Circular Queue

Circular queue:

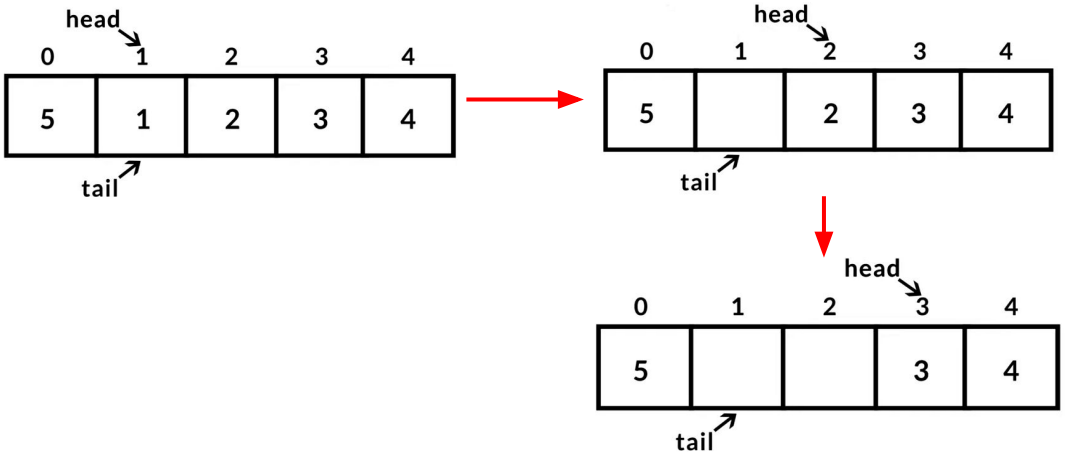
- **Enqueue:** head and tail aren't fixed
- Here we enqueue items [4,5]



Types: Circular Queue

Circular queue:

- **Dequeue:** we change the position of head



Types: Circular Queue

Circular queue:

- Condition for an empty queue: the size is 0
- Condition for a full queue is: the size is equal to the capacity

- **enqueue():** remove from head then increment head
- **dequeue():** add to tail then increment tail

References

1. https://www.geeksforgeeks.org/queue-in-python/
2. https://www.geeksforgeeks.org/queue-data-structure/
3. https://history-computer.com/applications-of-queue-real-world-uses-explained-in-plain-english/
4. https://www.programiz.com/blog/dsa-in-everyday-life/
5. https://youtu.be/VFSUWEAFmy4?list=PLG6bQ2KtCuYOhDZiiVzxiBnrzgSNzLW-y