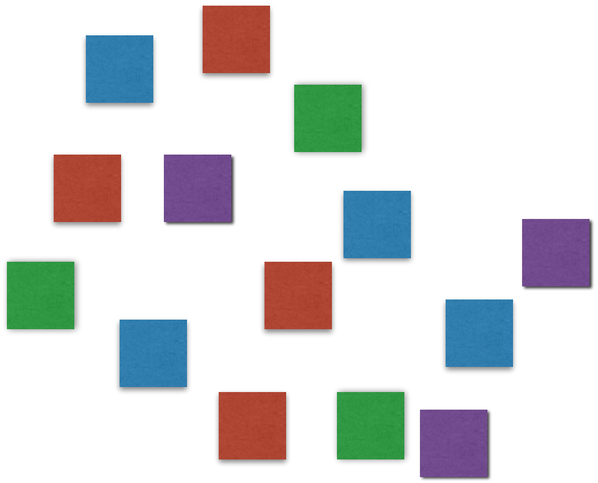
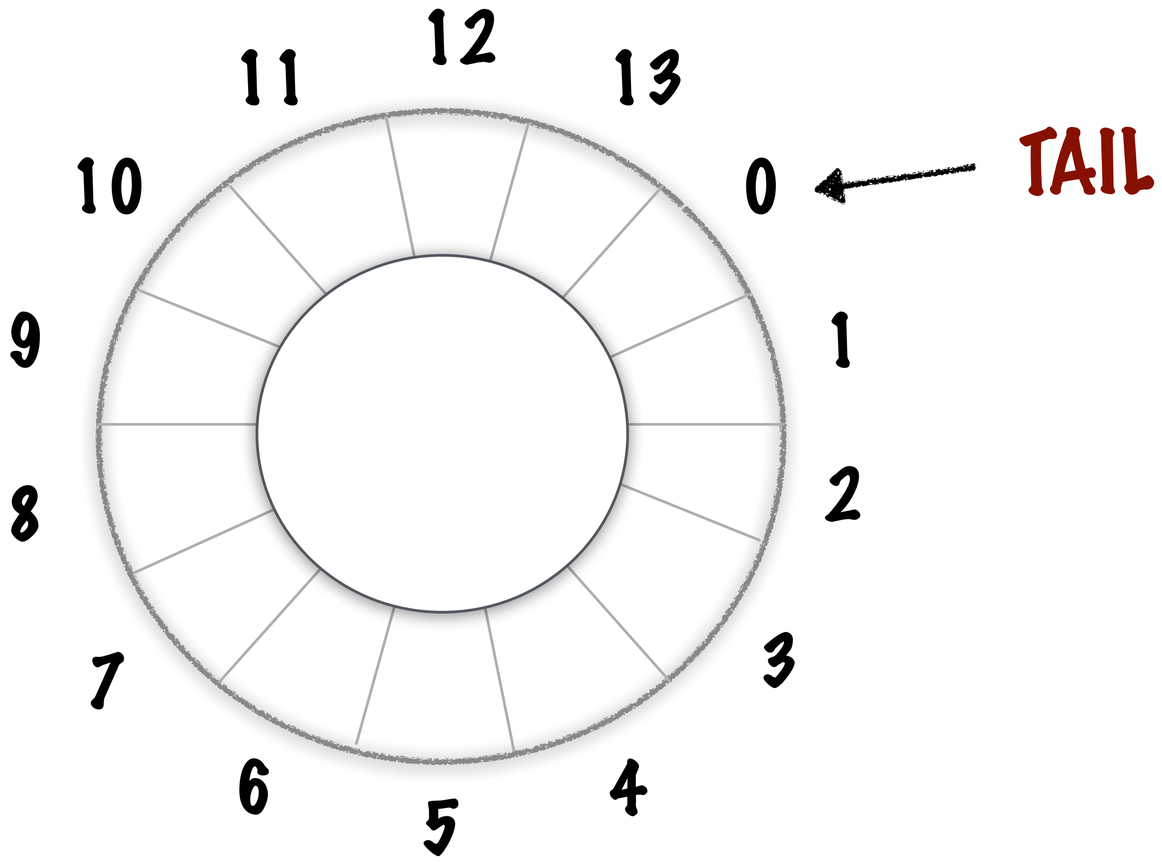
LET'S BUILD A QUEUE



THE QUEUE

HEAD = -1

A SPECIAL VALUE TO DENOTE AN EMPTY LIST

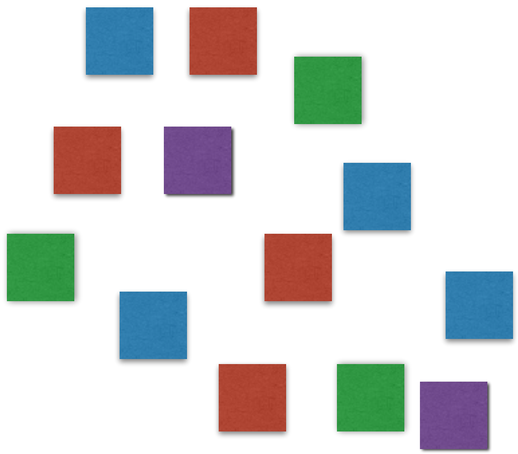


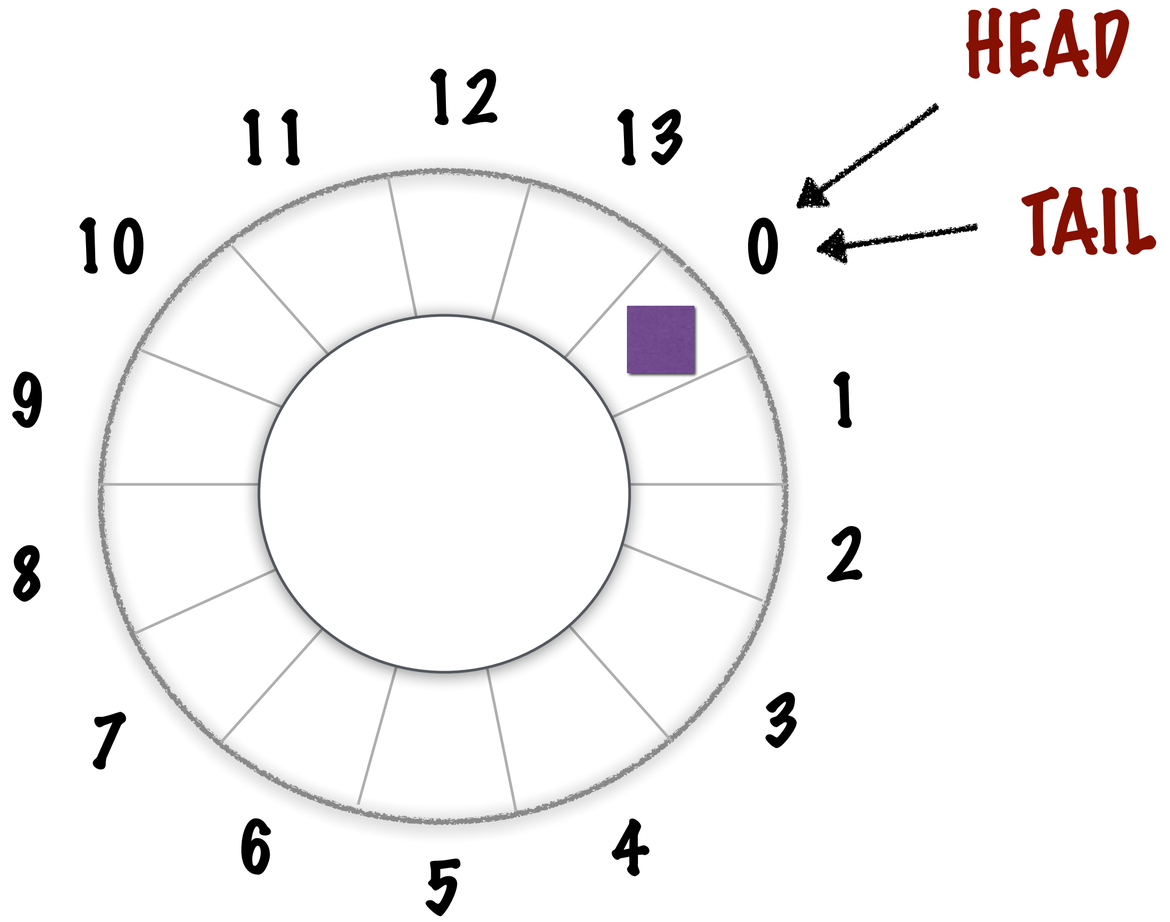
TAIL

ENQUEUE

THIS CAN BE IMPLEMENTED USING AN ARRAY WHERE THE LAST ELEMENT WRAPS AROUND TO THE FIRST ELEMENT

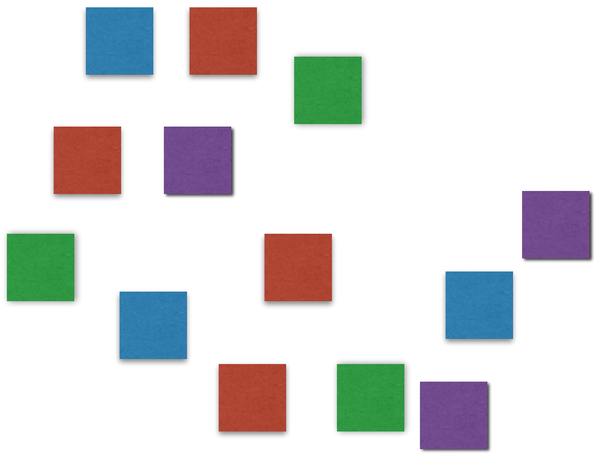
THE QUEUE

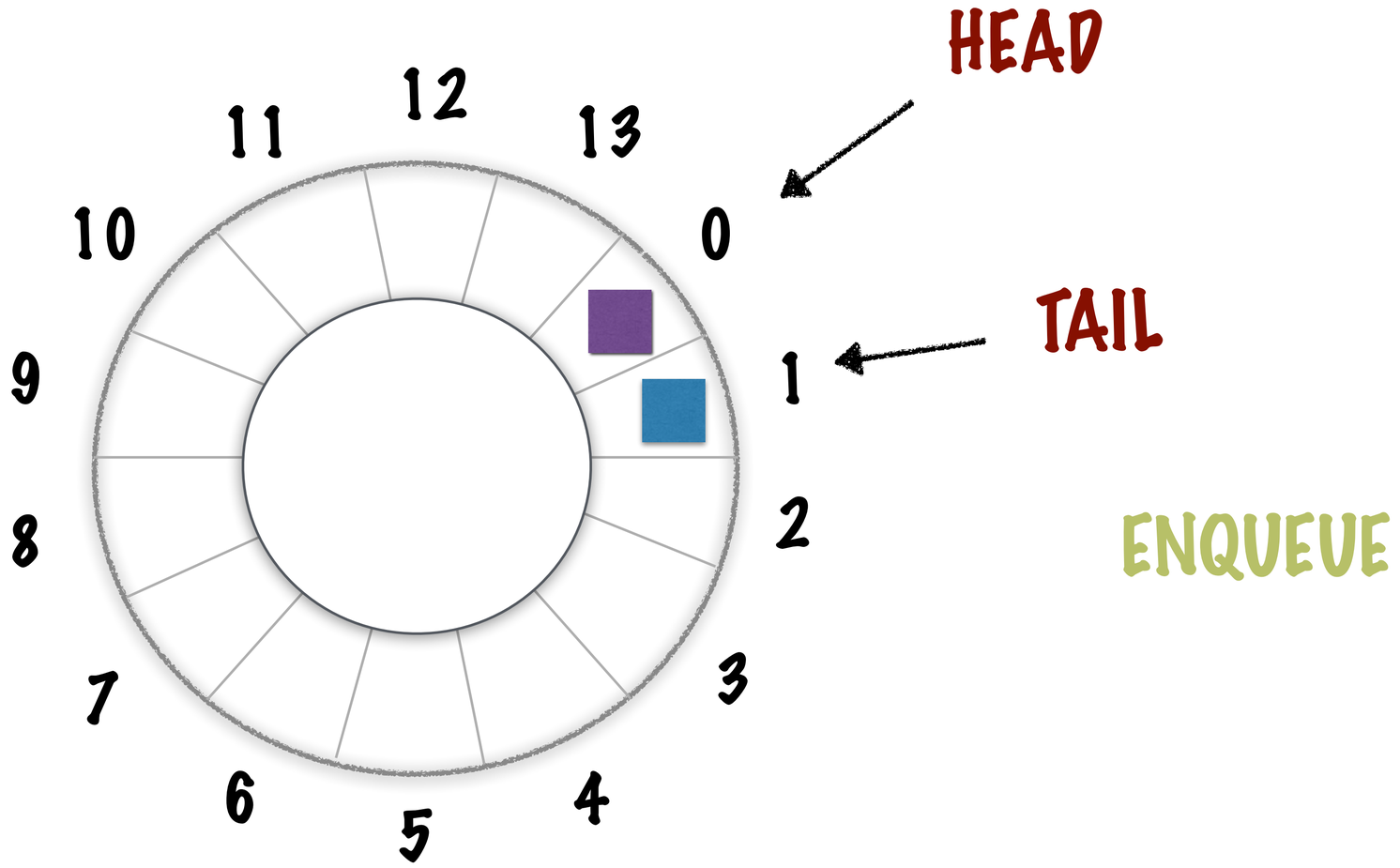




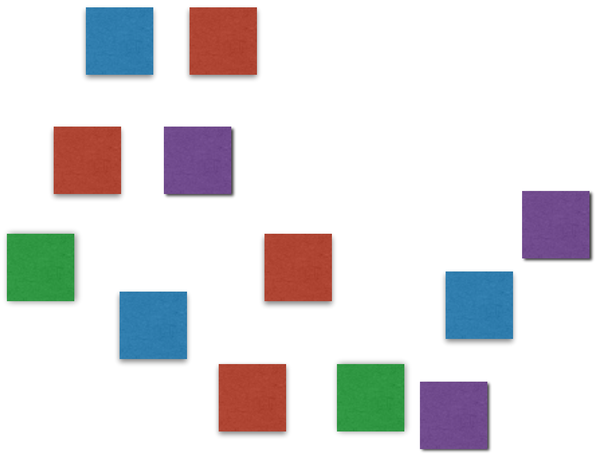
ENQUEUE

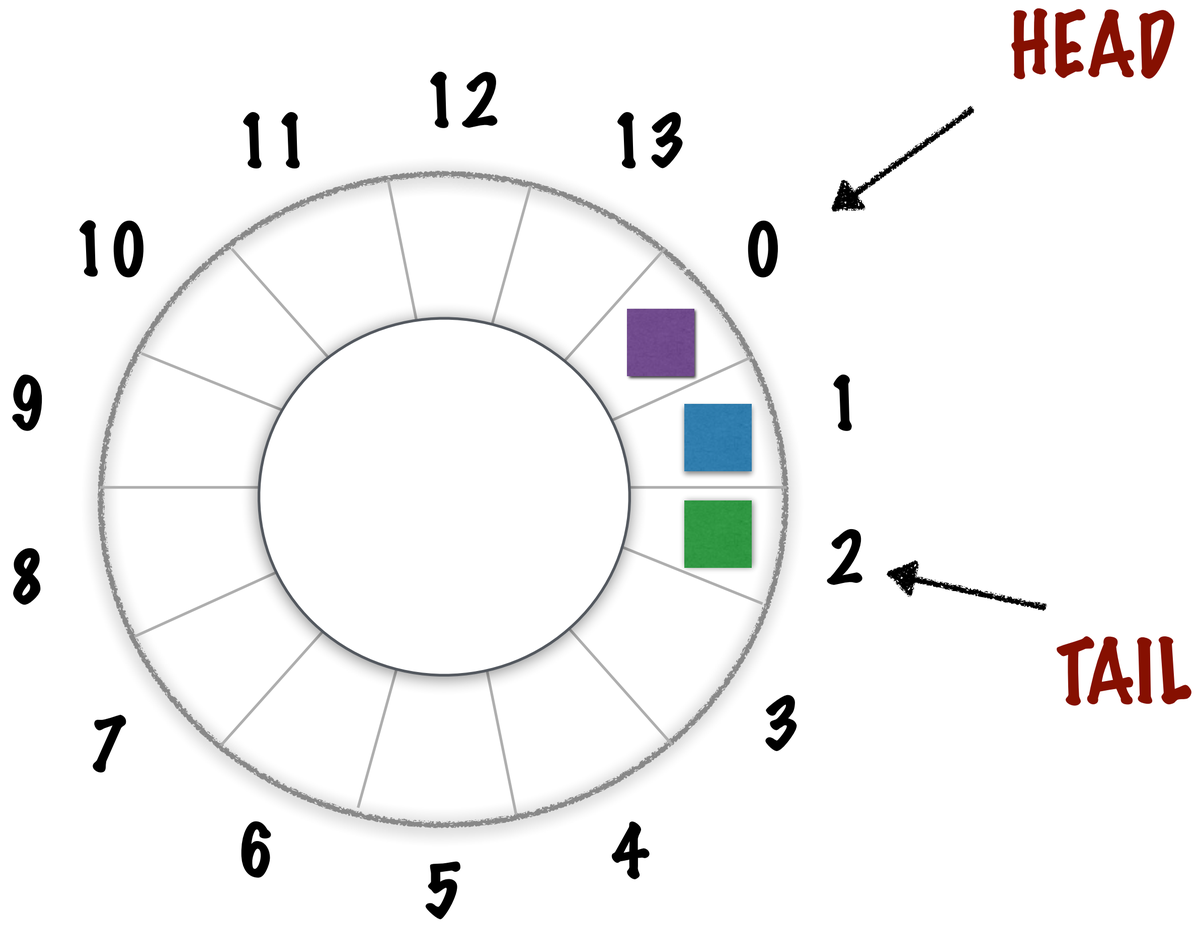
THE QUEUE





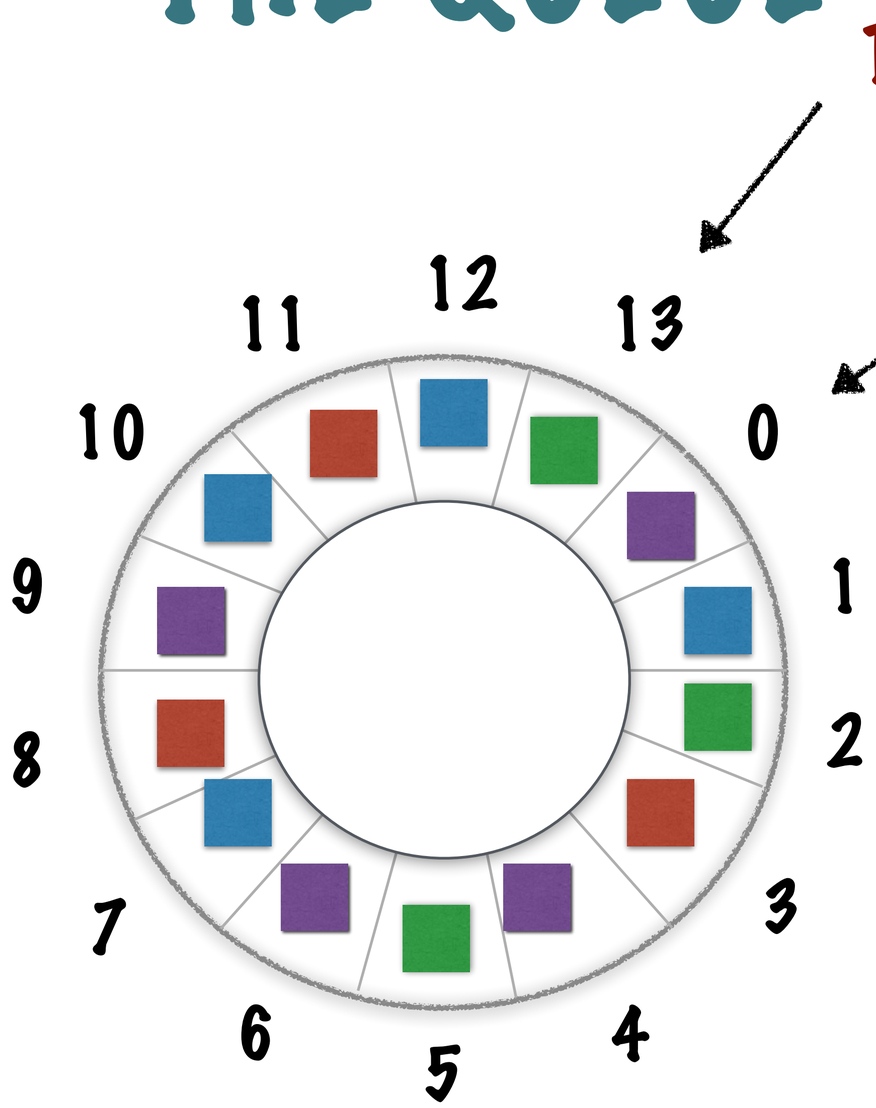
THE QUEUE





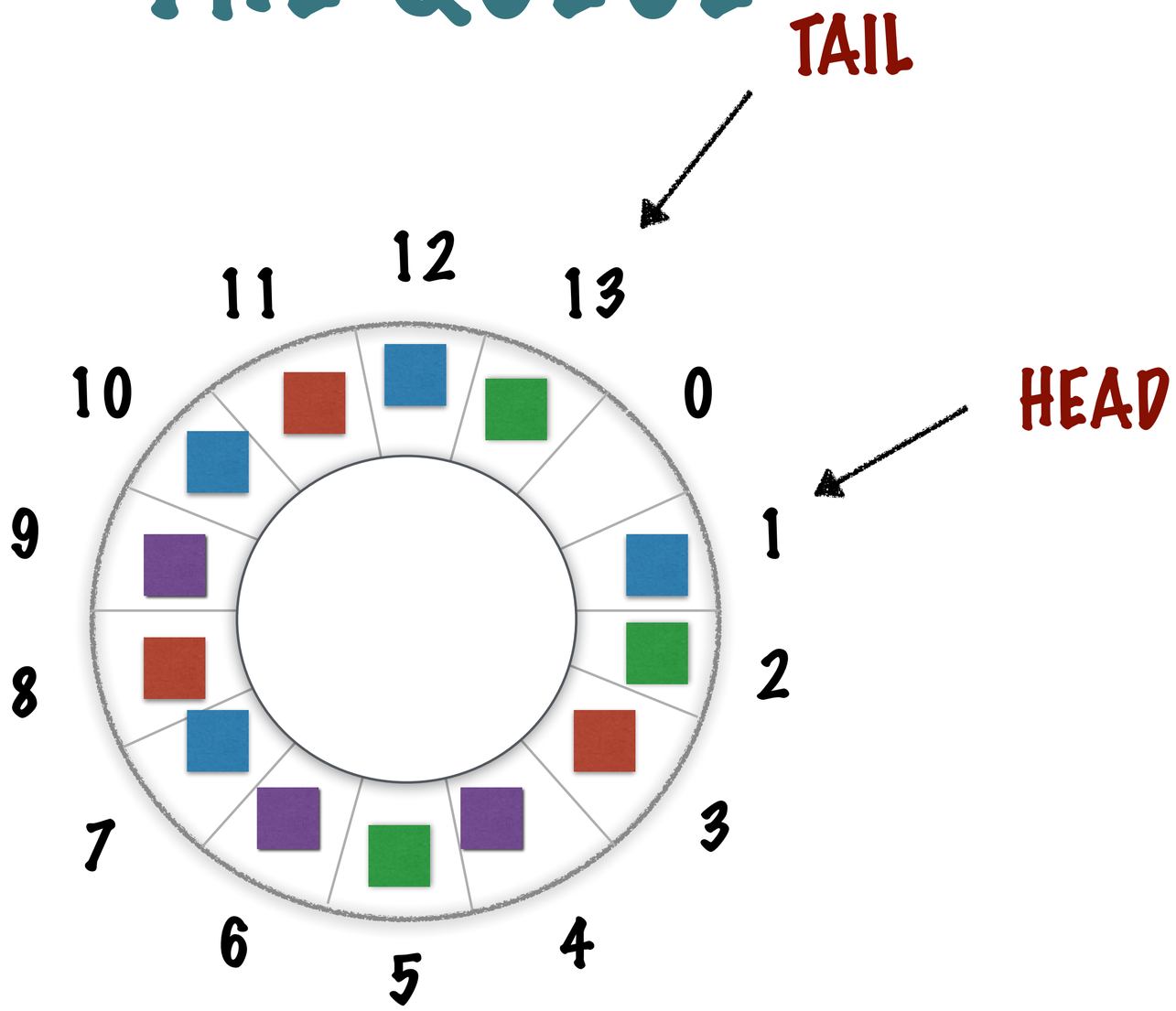
ENQUEUE ELEMENTS TILL
THE QUEUE IS FULL

THE QUEUE



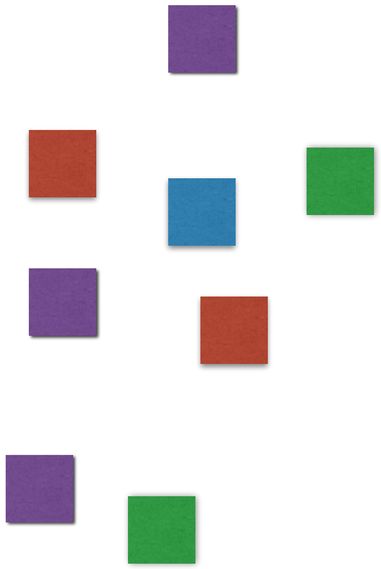
DEQUEUE

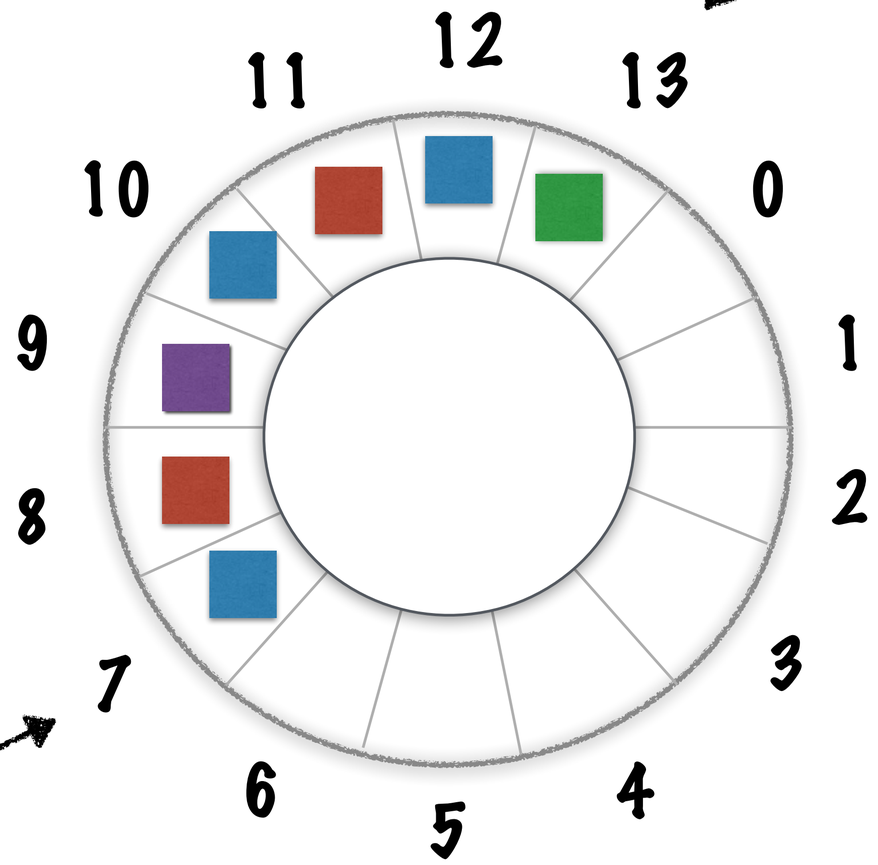
THE QUEUE



DEQUEUE 6 ELEMENTS

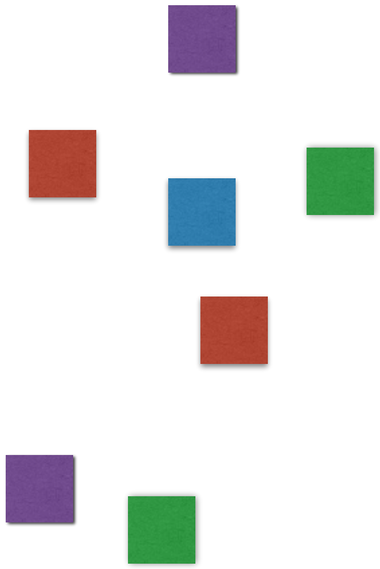
THE QUEUE

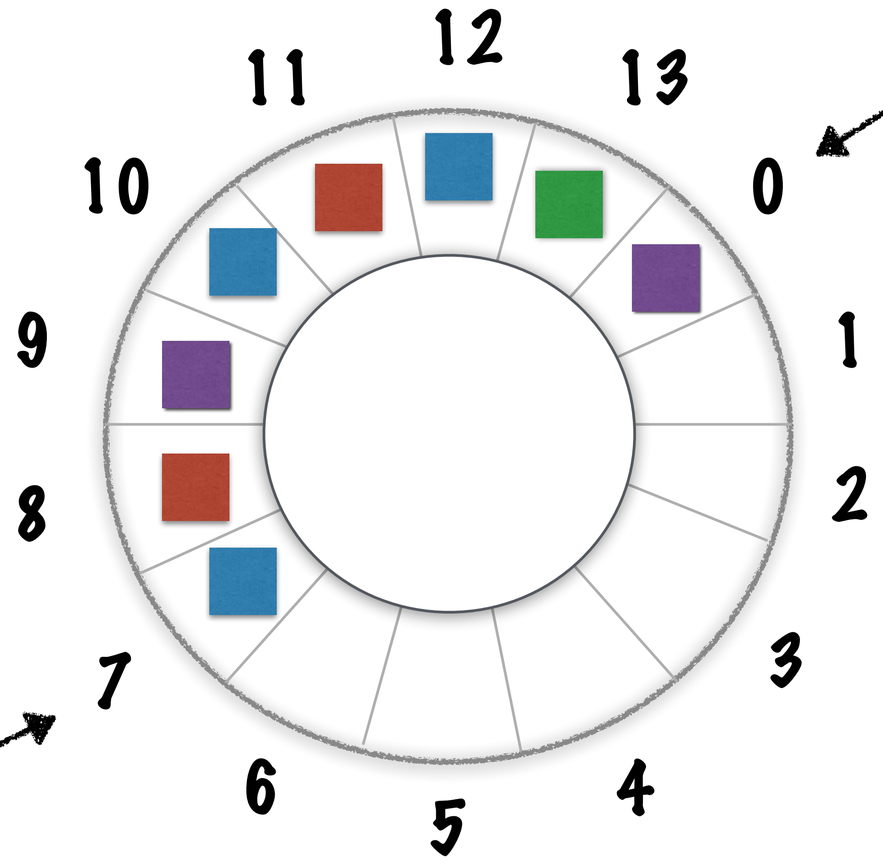




ENQUEUE

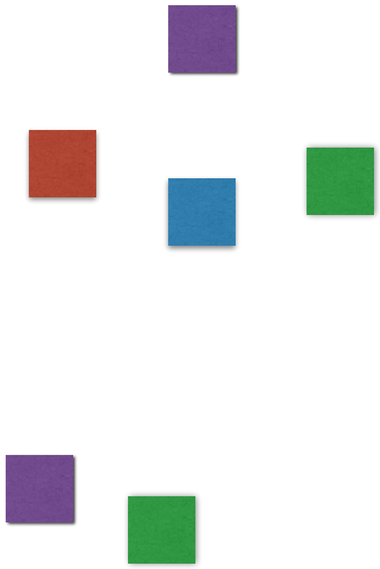
THE QUEUE

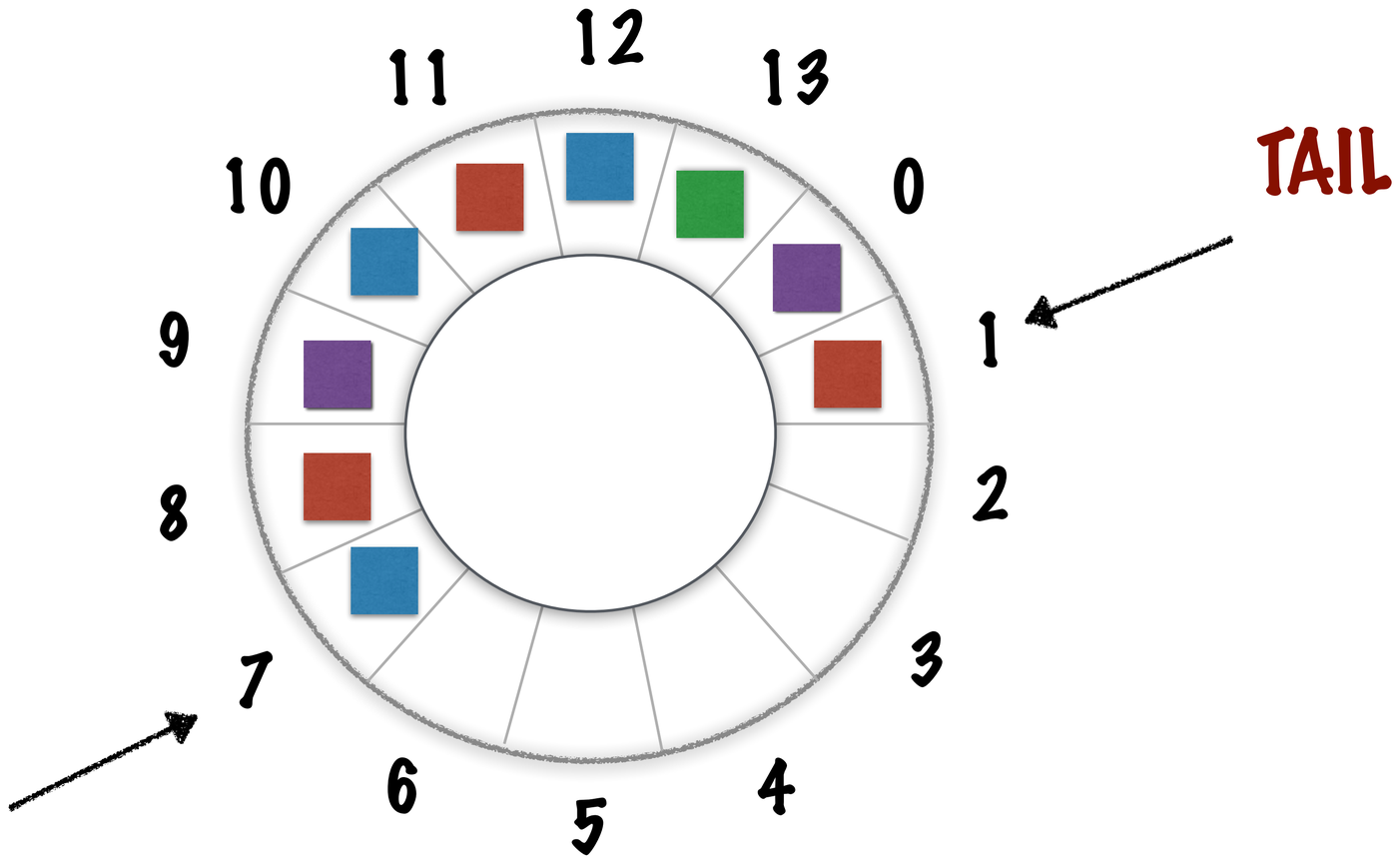




ENQUEUE

THE QUEUE





SET UP THE QUEUE CLASS - VARIABLES AND CONSTRUCTOR

```java
public class Queue<T> {

    private static final int SPECIAL_EMPTY_VALUE = -1;
    private static int MAX_SIZE = 40;
    private T[] elements;

    // The head index is initialized to a special value which
    // indicate that the queue is empty.
    private int headIndex = SPECIAL_EMPTY_VALUE;
    private int tailIndex = SPECIAL_EMPTY_VALUE;


    public Queue(Class<T> clazz) {
        elements = (T[]) Array.newInstance(clazz, MAX_SIZE);
    }
```

SPECIAL EMPTY VALUE FOR THE HEAD OF THE QUEUE TO FLAG WHEN THERE ARE NO ELEMENTS IN A QUEUE

INITIALIZE BOTH THE HEAD AND THE TAIL INDICES TO THE SPECIAL VALUE

THIS IS HOW GENERIC ARRAYS ARE INITIALIZED IN JAVA, IT'S AN IMPLEMENTATION DETAIL - BUT USEFUL

EXCEPTIONS THROWN

ENQUEUING INTO A FULL QUEUE

```
public static class QueueOverflowException extends Exception {
}

public static class QueueUnderflowException extends Exception {
}
```

DEQUEUEING OR PEEKING INTO AN EMPTY QUEUE

ISFULL AND ISEMPTY

```
public boolean isEmpty()  {
    return headIndex == SPECIAL_EMPTY_VALUE;
}

public boolean isFull()  {
    int nextIndex = (tailIndex + 1) % elements.length;

    return nextIndex == headIndex;
}
```

THE HEADINDEX IS ALWAYS SET TO THE SPECIAL MARKER VALUE WHEN THE QUEUE IS EMPTY

WHEN THE QUEUE IS FULL THIS MEANS THAT THE HEAD INDEX AND TAIL INDEX ARE RIGHT NEXT TO ONE ANOTHER

CHECK WHETHER THE NEXT POSITION OF THE TAIL IS THE HEAD INDEX

ENQUEUE

```
public void enqueue(T data) throws QueueOverflowException {
    if (isFull()) {
        throw new QueueOverflowException();
    }
    tailIndex = (tailIndex + 1) % elements.length;
    elements[tailIndex] = data;

    // This is the first element enqueued, set the head index
    // to the tail index.
    if (headIndex == SPECIAL_EMPTY_VALUE) {
        headIndex = tailIndex;
    }
}
```

CHECK FOR A FULL QUEUE AND FAIL ACCORDINGLY

GET THE NEXT TAIL INDEX AND INSERT THE NEW ELEMENT THERE

NOTE THAT WE NEED TO WRAP AROUND TO THE FIRST POSITION IF WE'RE AT THE END OF THE CIRCULAR ARRAY

IF THE HEAD IS THE SPECIAL MARKER VALUE, IT MEANS THE QUEUE WAS PREVIOUSLY EMPTY - SET IT TO THE SAME INDEX AS TAIL

DEQUEUE

```
public T dequeue() throws QueueUnderflowException {
    if (isEmpty()) {
        throw new QueueUnderflowException();
    }

    T data = elements[headIndex];

    // This was the last element in the queue.
    if (headIndex == tailIndex) {
        headIndex = SPECIAL_EMPTY_VALUE;
    } else {
        headIndex = (headIndex + 1) % elements.length;
    }

    return data;
}
```

CHECK FOR AN EMPTY QUEUE AND FAIL ACCORDINGLY

HEAD INDEX POINTS TO THE FIRST ELEMENT, STORE THAT VALUE TO RETURN

IF THE HEAD INDEX IS THE SAME AS THE TAIL INDEX, THEN WE'VE JUST DEQUEUED THE VERY LAST ELEMENT - MARK THE HEAD ACCORDINGLY

MOVE THE HEAD TO THE NEXT ELEMENT - REMEMBER TO WRAP AROUND TO THE BEGINNING OF THE ARRAY FOR THE LAST ELEMENT

THE QUEUE - PERFORMANCE AND COMPLEXITY

ENQUEUING AND DEQUEUING
IMPLEMENTED IN THIS WAY IS
O(1), CONSTANT TIME
COMPLEXITY

IS EMPTY AND IS FULL
IS ALSO O(1)

SPACE COMPLEXITY IS O(N)