WHERE CAN A QUEUE BE USED?

CUSTOMER SERVICE HOTLINE, CALLS ARE ASSIGNED TO REPRESENTATIVES IN THE ORDER THAT THEY ARE RECEIVED

QUEUEING JOBS TO BE PRINTED

ANY ORDER PROCESSING SYSTEMS LIKE IN E-COMMERCE WEBSITES OR BANK TRANSACTION SYSTEMS

IMPLEMENT A QUEUE USING TWO STACKS

USE A **FORWARD STACK** USED TO PUSH THE ENQUEUED ELEMENTS

ENQUEUE OPERATIONS ARE ALWAYS PERFORMED ON THE FORWARD STACK

THE **REVERSE STACK** HOLDS THE ELEMENTS IN REVERSE ORDER FROM THE FORWARD STACK

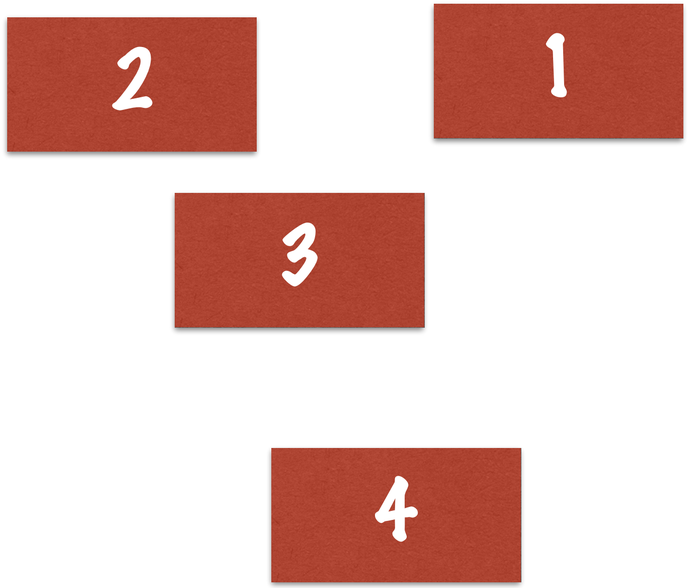
DEQUEUE OPERATIONS ARE ALWAYS PERFORMED ON THE REVERSE STACK

ENQUEUE REQUIRES MOVING ALL ELEMENTS TO THE FORWARD STACK AND PUSHING THE LAST ELEMENT IN I.E. THE TOP

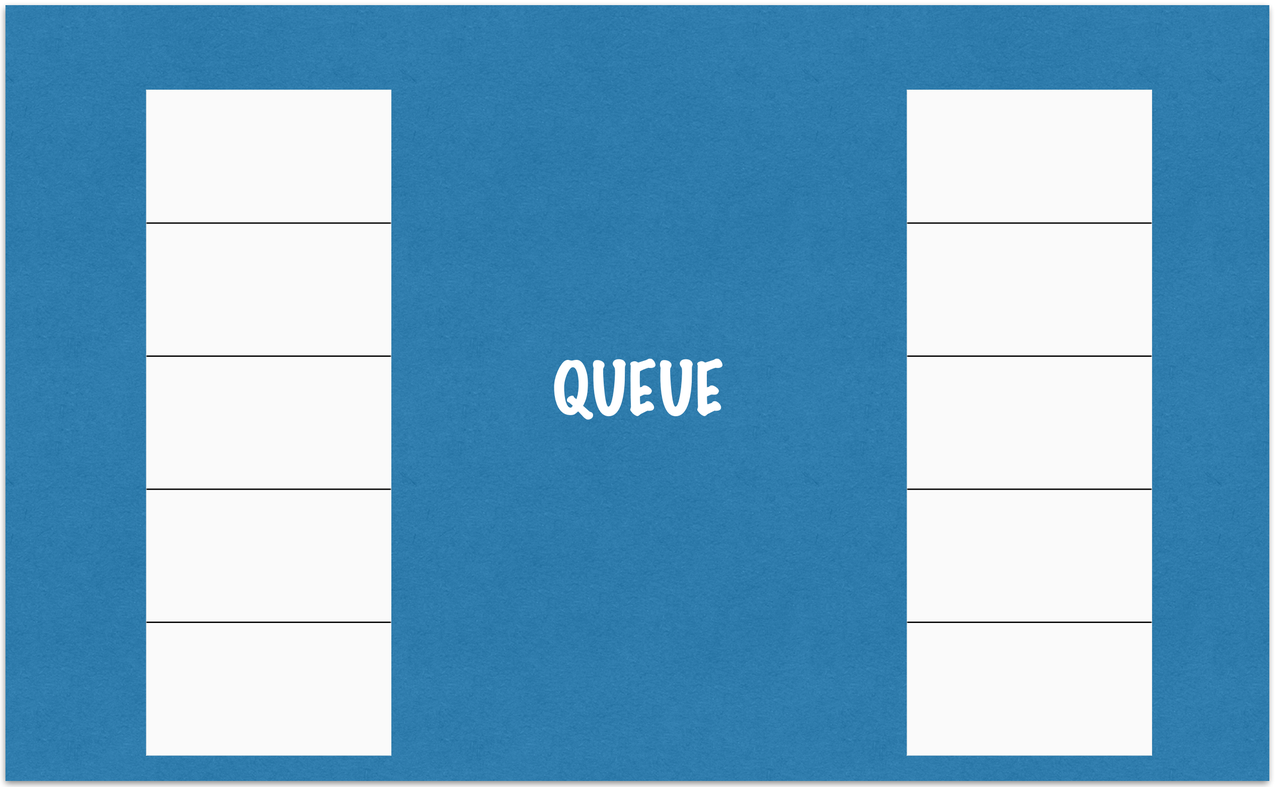
DEQUEUE REQUIRES MOVING ALL ELEMENTS TO THE REVERSE TRACK AND POPPING THE LAST ELEMENT IN I.E. THE TOP

A QUEUE USING 2 STACKS

ALWAYS ENQUEUE BY PUSHING INTO THE FORWARD STACK



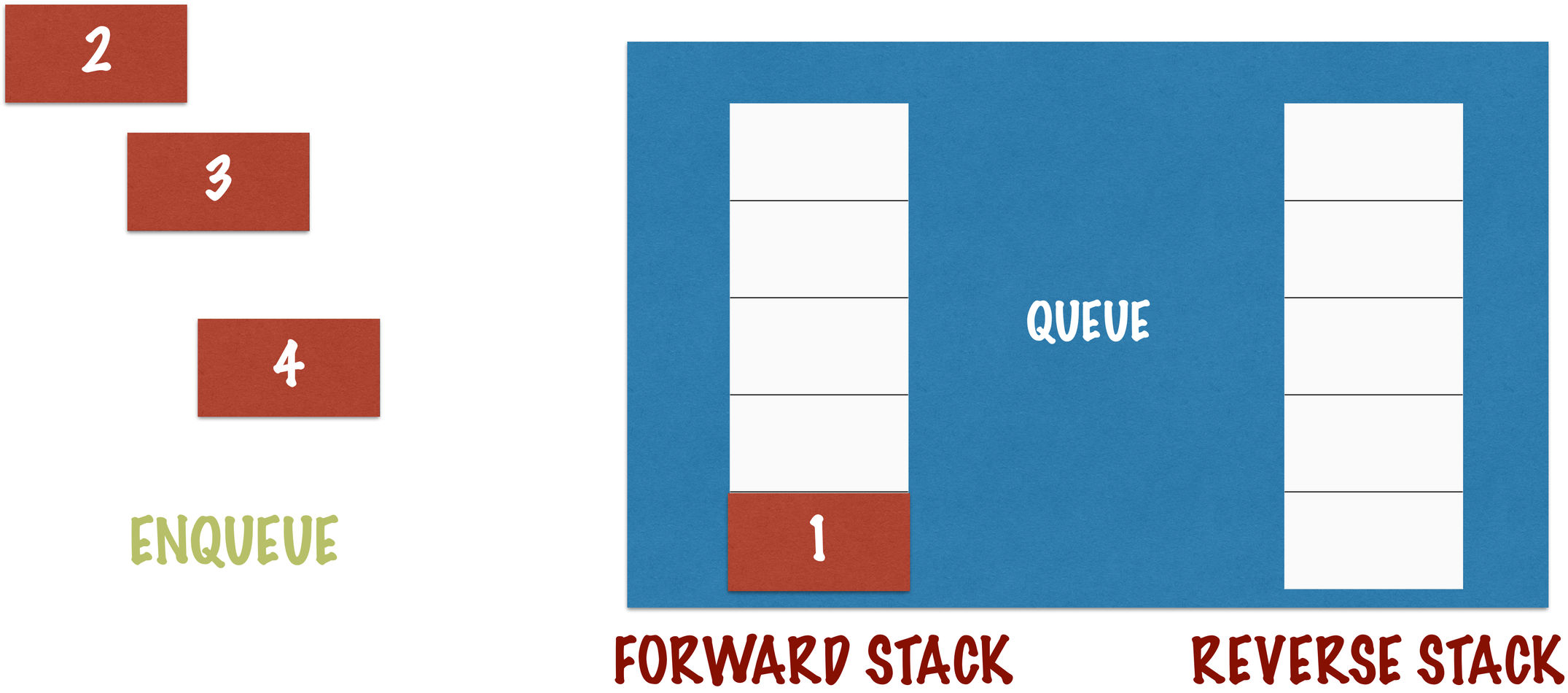
ENQUEUE



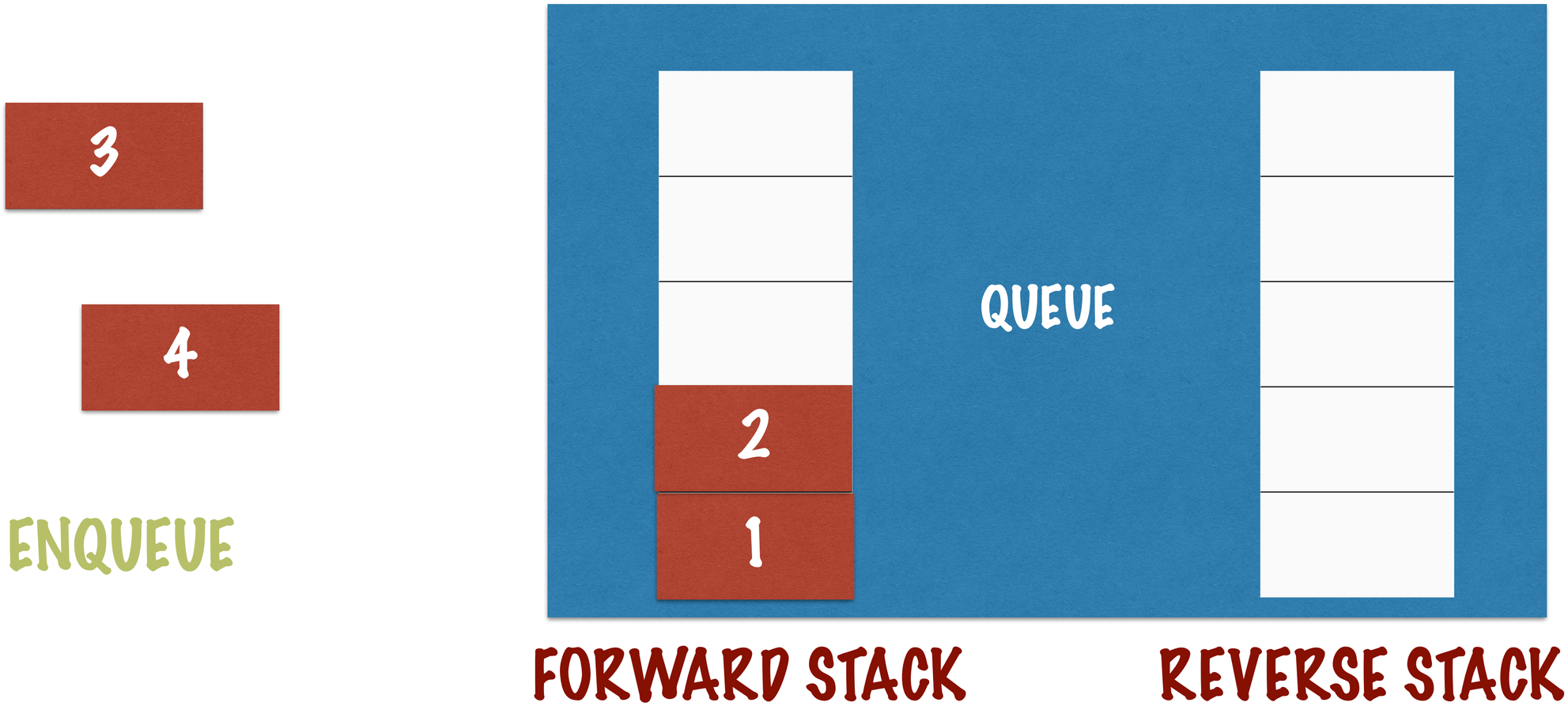
FORWARD STACK

REVERSE STACK

A QUEUE USING 2 STACKS

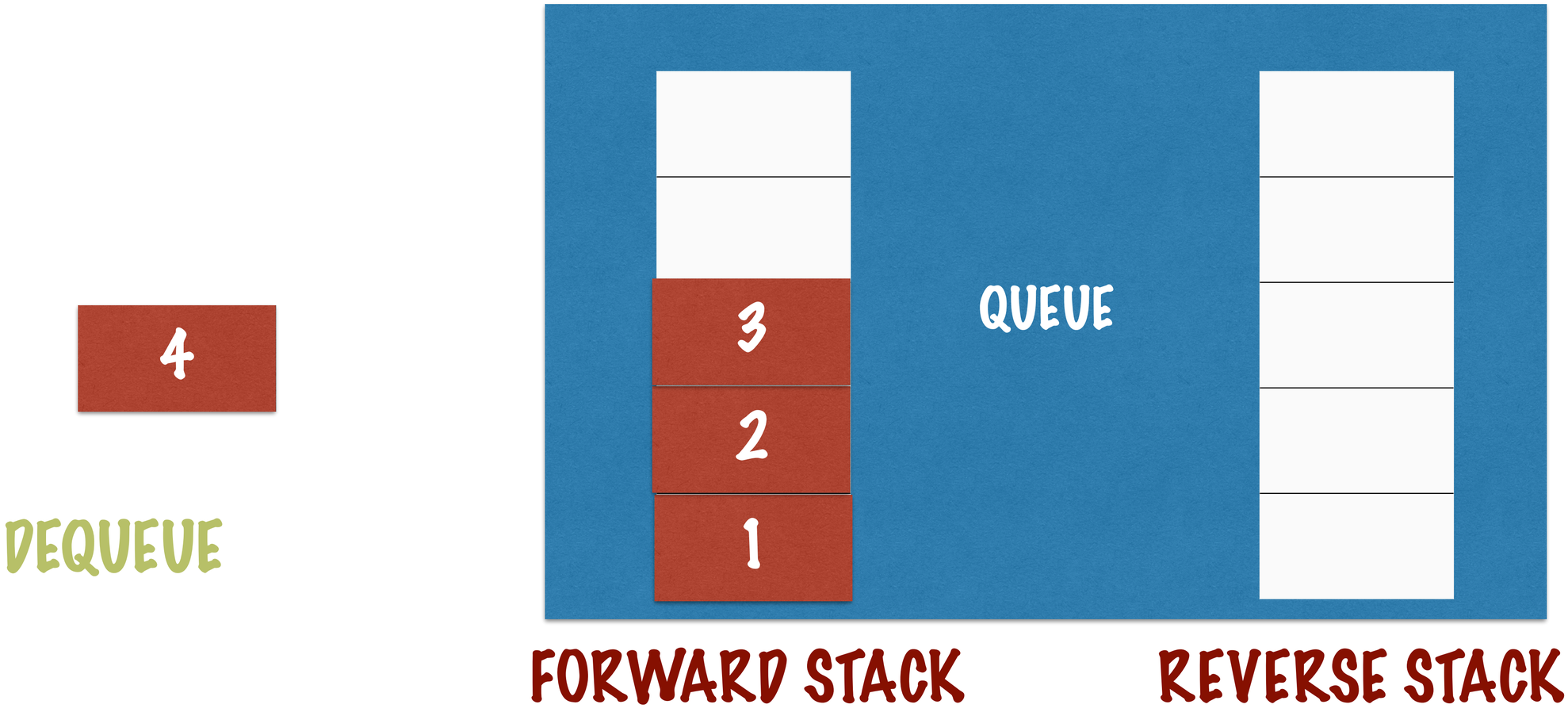


A QUEUE USING 2 STACKS



A QUEUE USING 2 STACKS

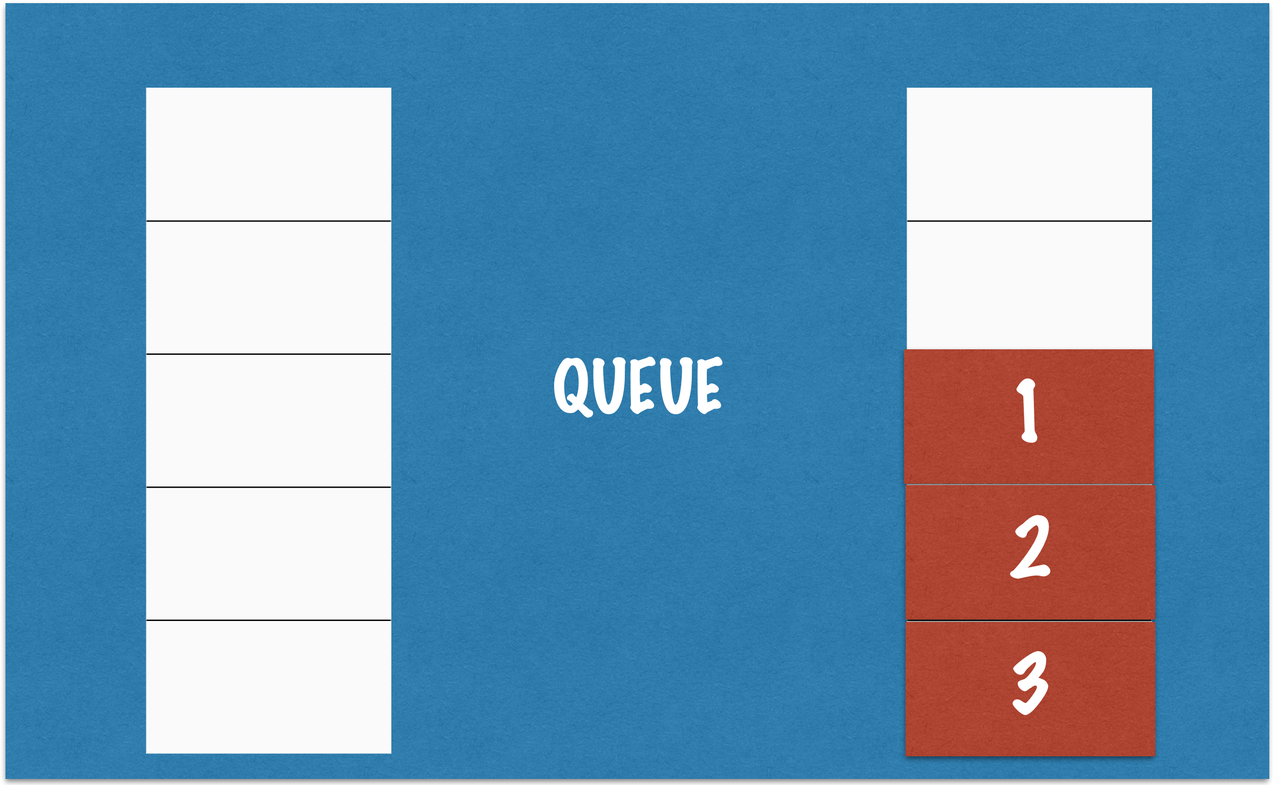
MOVE ALL ELEMENTS
TO THE REVERSE STACK



A QUEUE USING 2 STACKS

NOW DEQUEUE BY POPPING FROM THE REVERSE STACK

DEQUEUE

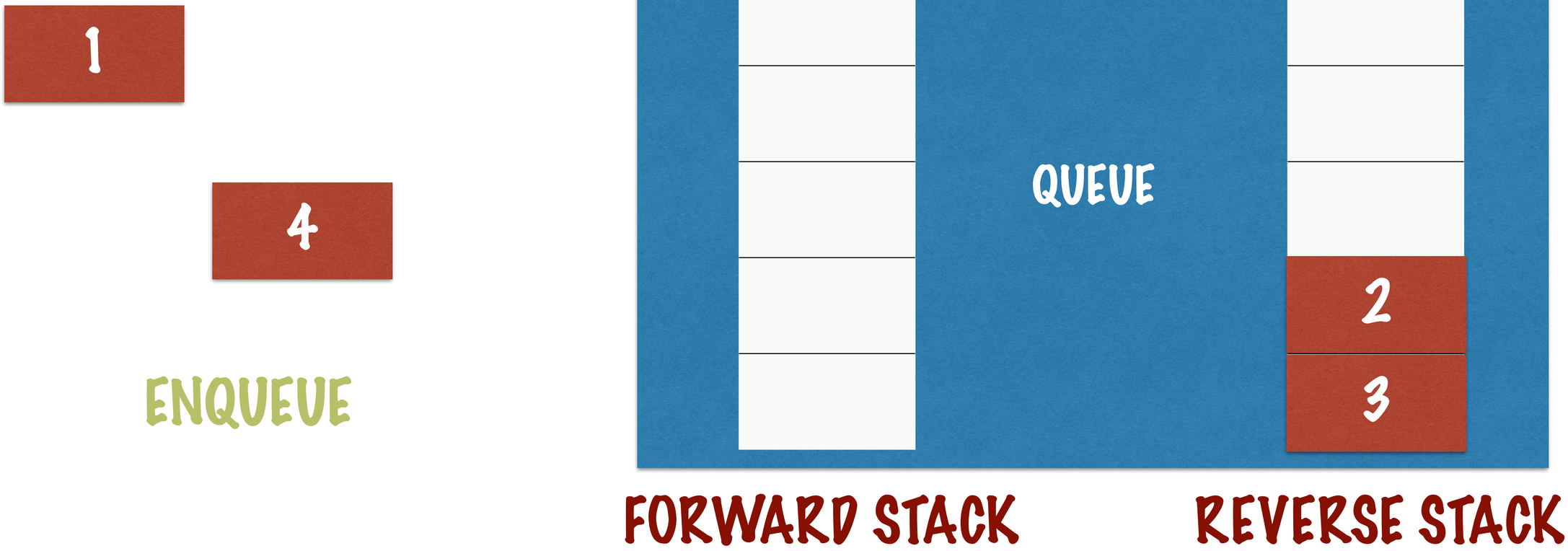


FORWARD STACK

REVERSE STACK

A QUEUE USING 2 STACKS

MOVE ALL ELEMENTS
TO THE FORWARD STACK



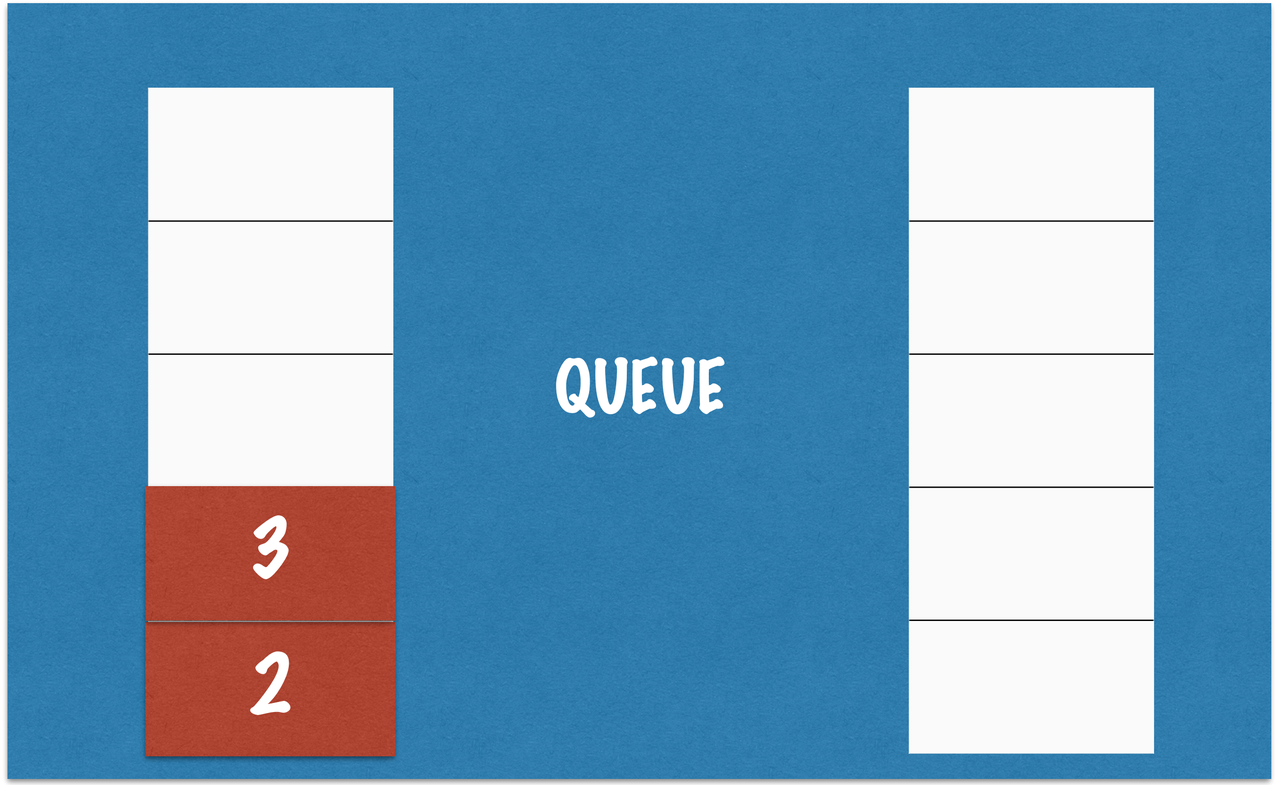
A QUEUE USING 2 STACKS

NOW PUSH INTO THE FORWARD STACK

1

4

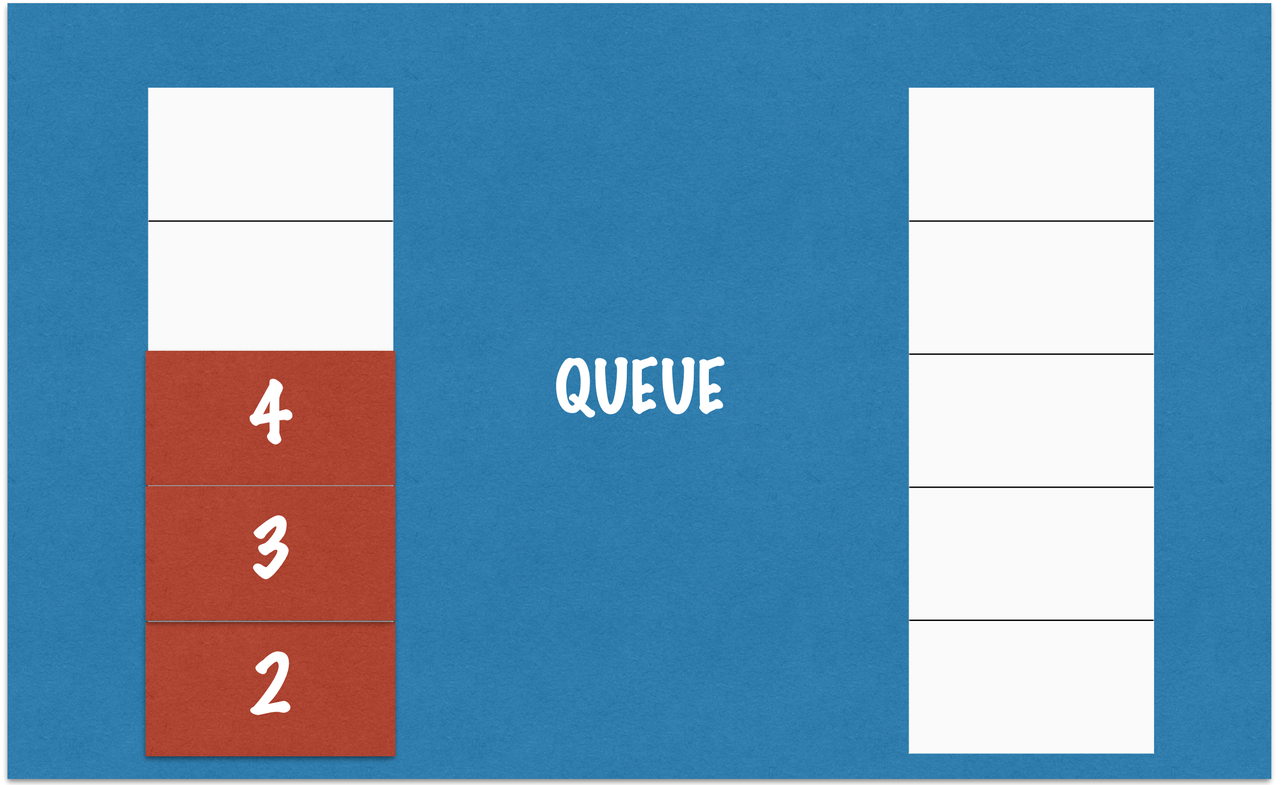
ENQUEUE



A QUEUE USING 2 STACKS

NOW PUSH INTO THE FORWARD STACK

1



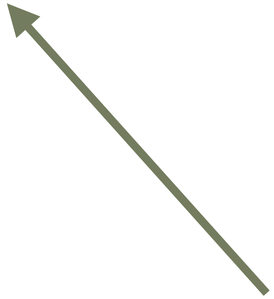
SET UP THE QUEUE IMPLEMENTED WITH 2 STACKS

```
public class QueueBuiltWithTwoStacks<T> {

    private Stack<T> forwardStack = new Stack<>();
    private Stack<T> reverseStack = new Stack<>();


    public QueueBuiltWithTwoStacks() {

    }
```

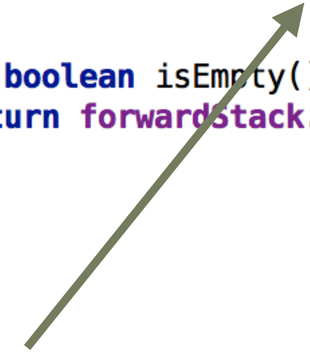


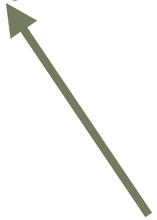
SET UP THE TWO STACKS - THE FORWARD STACK AND THE REVERSE STACK

ISFULL AND ISEMPTY DELEGATES TO THE STACKS

```
public boolean isFull() {
    return forwardStack.isFull() || reverseStack.isFull();
}

public boolean isEmpty() {
    return forwardStack.isEmpty() && reverseStack.isEmpty();
}
```





EITHER STACK COULD HOLD ALL THE ELEMENTS, IF EITHER IS FULL THE QUEUE IS FULL

IF BOTH STACKS ARE EMPTY NO ELEMENTS HAVE BEEN ADDED TO THE QUEUE

ENQUEUE

```
public void enqueue(T data) throws Queue.QueueOverflowException {
    if (isFull()) {
        throw new Queue.QueueOverflowException();
    }

    try {
        if (forwardStack.isEmpty()) {
            while (!reverseStack.isEmpty()) {
                forwardStack.push(reverseStack.pop());
            }
        }
        forwardStack.push(data);
    } catch (Stack.StackOverflowException | Stack.StackUnderflowException se) {
        throw new Queue.QueueOverflowException();
    }
}
```

SANITY CHECK,
IS THE QUEUE FULL?

PUSH ALL ELEMENTS FROM THE REVERSE STACK TO THE FORWARD STACK, ENQUEUE ALWAYS HAPPENS ON THE FORWARD STACK

FINALLY DO THE ENQUEUE - A PUSH ON THE FORWARD STACK

DEQUEUE

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

    try {
        if (reverseStack.isEmpty()) {
            while (!forwardStack.isEmpty()) {
                reverseStack.push(forwardStack.pop());
            }
        }

        return reverseStack.pop();
    } catch (Stack.StackOverflowException | Stack.StackUnderflowException se) {
        throw new Queue.QueueUnderflowException();
    }
}
```

SANITY CHECK,
IS THE QUEUE EMPTY?

PUSH ALL ELEMENTS FROM THE FORWARD STACK TO THE REVERSE STACK, DEQUEUE ALWAYS HAPPENS ON THE REVERSE STACK

FINALLY DO THE DEQUEUE - A POP ON THE REVERSE STACK

QUEUE USING 2 STACKS - PERFORMANCE AND COMPLEXITY

ALL ENQUEUES AND THEN ALL DEQUEUES ARE O(1) - IF ONLY ONE OF THESE OPERATIONS ARE PERFORMED

NOTICE THAT EACH ELEMENT IS PUSHED NO MORE THAN TWICE

ONCE ONTO THE FORWARD STACK TO ENQUEUE IT AND ONCE ONTO THE REVERSE STACK JUST BEFORE DEQUEUEING

EACH ELEMENT IS POPPED NO MORE THAN TWICE

ONCE TO MOVE TO THE REVERSE FROM THE FORWARD STACK JUST BEFORE DEQUEUEING AND THEN TO ACTUALLY DEQUEUE IT

TIME COMPLEXITY IS O(M) WHERE M IS THE NUMBER OF OPERATIONS WE PERFORM ON THE QUEUE