Data Structures and Algorithms

XMUT-COMP 103 - 2025 T1

Queues

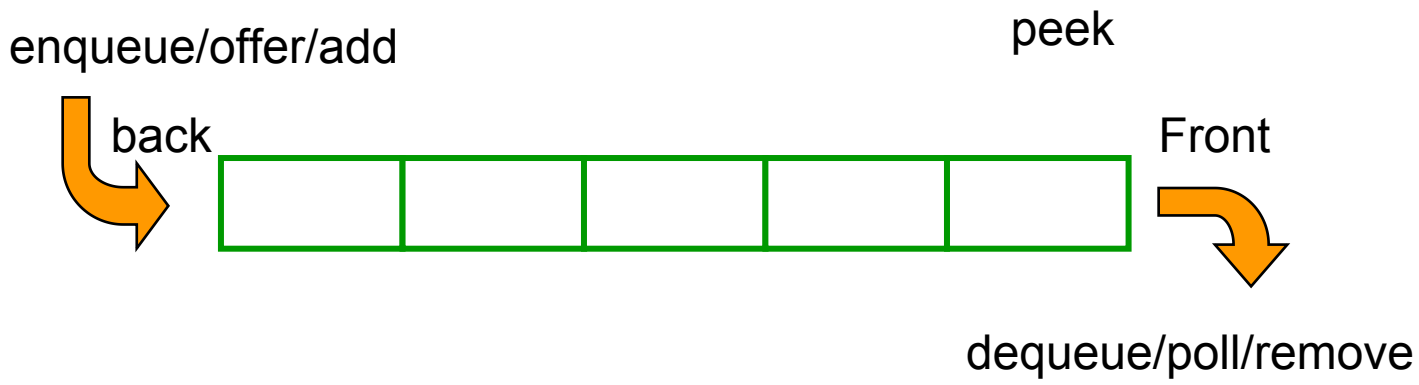
Felix Yan

School of Engineering and Computer Science

Victoria University of Wellington

Queues

- Collection of items in order
  - like Lists and Stacks



- Main operations:
  - enqueue: put item on the queue
  - dequeue: remove item from front of the queue

- These operations should be efficient.
  - Shouldn't get much more expensive if the queue is very large

- A Queue is a Collection:
  - THEREFORE: other operations – contains(…), remove(…), etc – also work

    BUT, they are not efficient.

Queue operations

- isEmpty(),
- size(),
- clear()

- offer(E item) enqueue
- add(E item) enqueue

- poll() → E dequeue (returns null if queue is empty)
- remove() → E dequeue (throws exception if queue is empty)

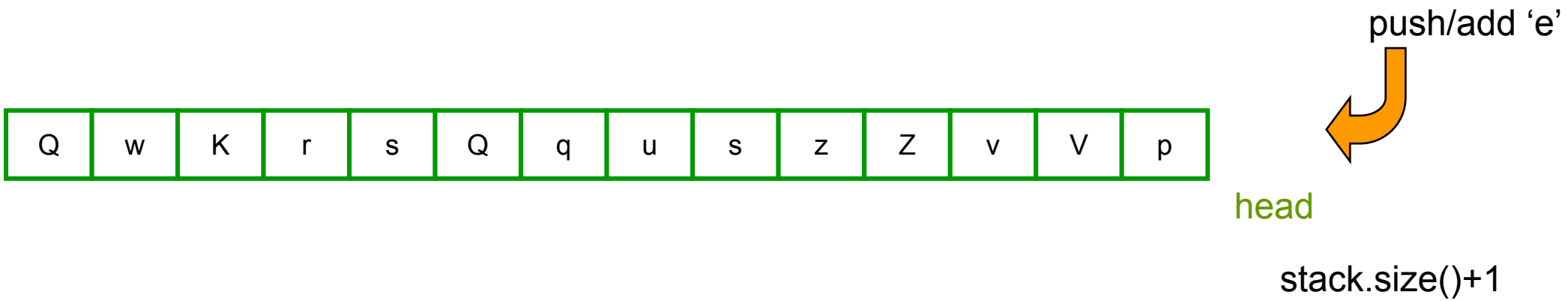
- peek() → E look at front (returns null if queue is empty)
- element() → E look at front (throws exception if queue empty)

Queues and efficiency

- The main operators of queues should be efficient.
  - the time it take to do them should be fast
  - especially important when they grow in size <= a constant speed is needed!
- Let’s investigate how stacks can be implemented efficiently.

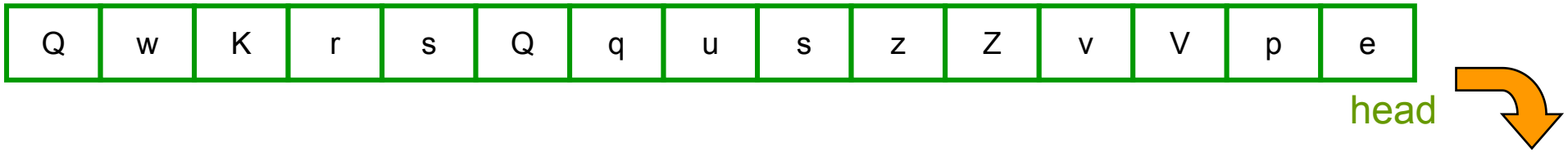
Stacks and efficiency

- You can use an ArrayList to implement a Stack (LIFO) efficiently:



Stacks and efficiency

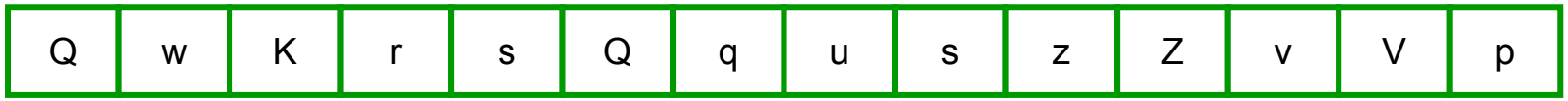
- You can use an ArrayList to implement a Stack (LIFO) efficiently:



- No changes to the stack other than an ’e’ was added at the end.

Stacks and efficiency

- You can use an ArrayList to implement a Stack (LIFO) efficiently:



pop/remove
returns ‘e’
stack size -1

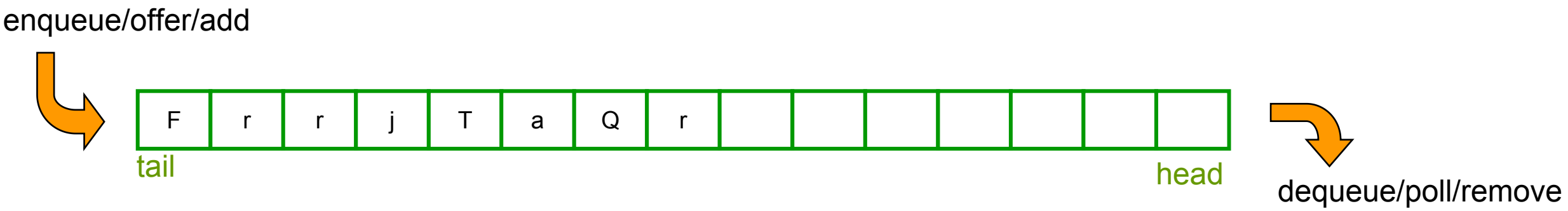
- Again only the end changes, nothing else
- push and pop at the end! O(1)

- Stacks are naturally efficient with an ArrayList!

Queues and efficiency

- What about a Queue (FIFO) ?

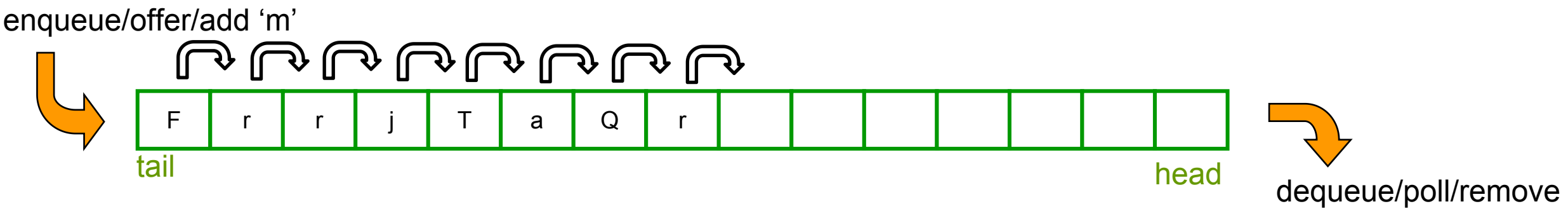
- Dequeue works like a stack, so is fast ! O(1)



Queues and efficiency

- What about a Queue (FIFO) ?

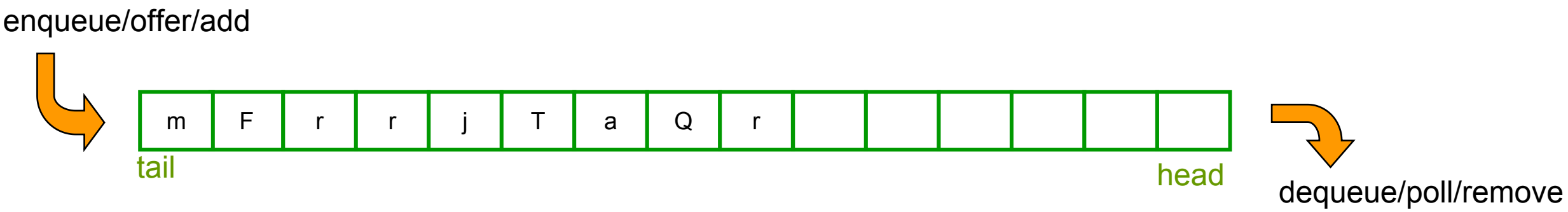
- Dequeue works like a stack, so is fast ! O(1)
- Enqueue requires shifting every item up one place to add
  - It “costs” the current length (n) to move ! O(n)



Queues and efficiency

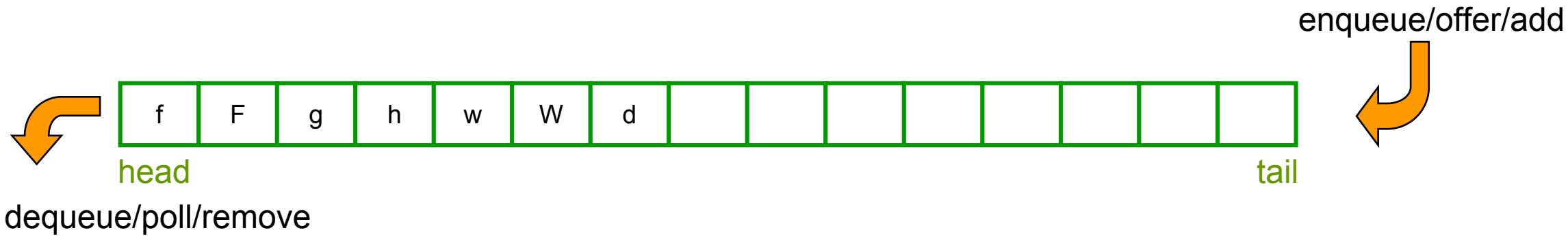
- What about a Queue (FIFO) ?

- Dequeue works like a stack, so is fast ! O(1)
- Enqueue requires shifting every item up one place to add
  - It “costs” the current length (n) to move ! O(n)



Queues and efficiency

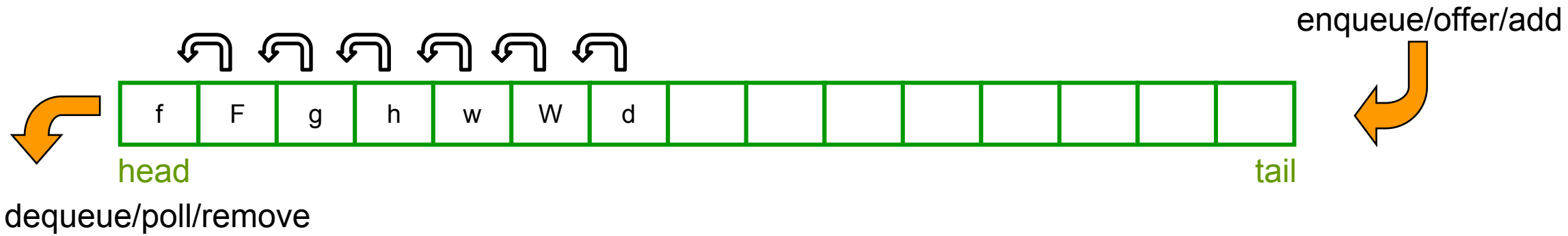
- What about a Queue, The other way round?



- Enqueue is like push on a stack, so it is fast ! O(1)

Queues and efficiency

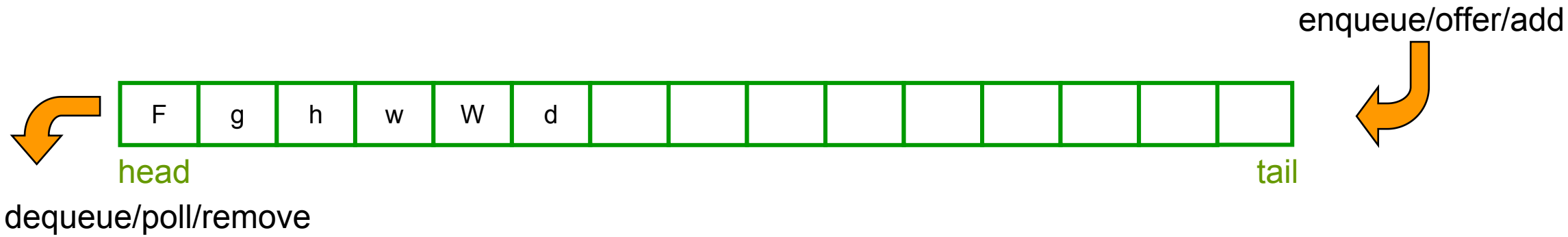
- What about a Queue, The other way round?



- Enqueue is like push on a stack, so it is fast ! O(1)
- Dequeue requires shifting every item down one place ! O(n)

Queues and efficiency

- What about a Queue, The other way round?

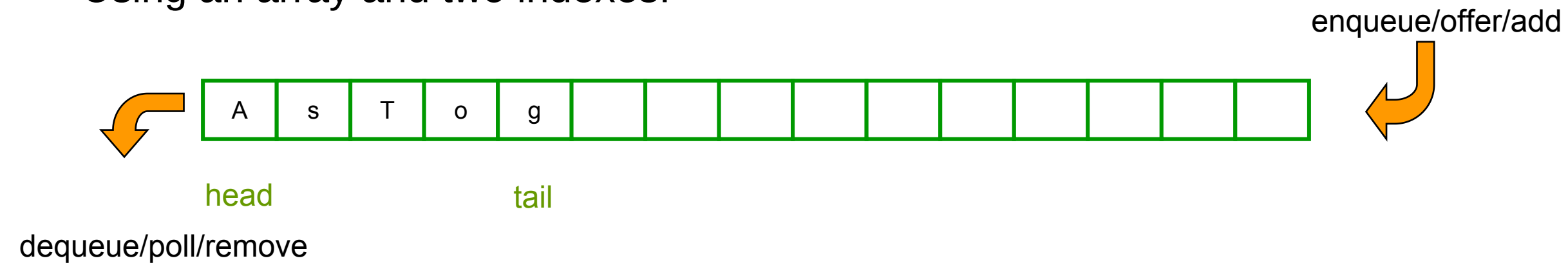


- Enqueue is like push on a stack, so it is fast ! O(1)
- Dequeue requires shifting every item down one place ! O(n)

- Big Oh notation:
  - O(1) : fixed number of steps, regardless of how big the collection is
  - O(n) : number of steps proportional to the size of the collection.

Queues and efficiency

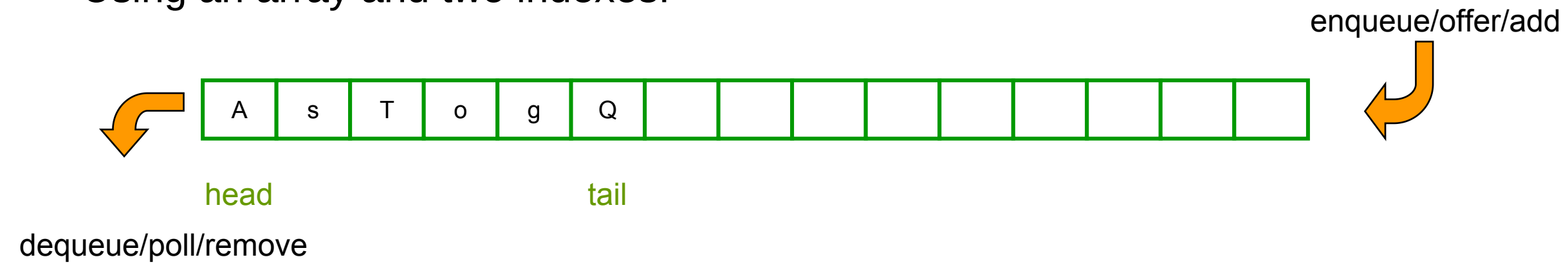
- Using an array and two indexes:



- Enqueue:
  - Get tail
  - tail++
  - Add new value at new tail

Queues and efficiency

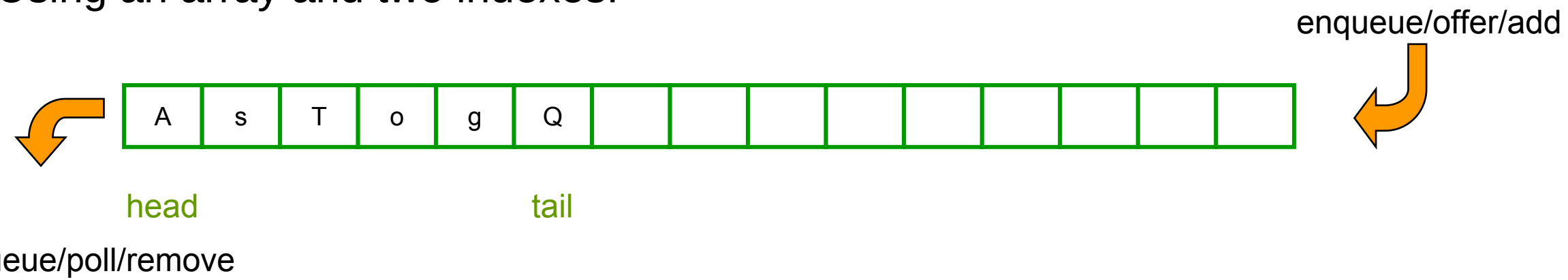
- Using an array and two indexes:



- Enqueue:
  - Get tail
  - tail++
  - Add new value at new tail

Queues and efficiency

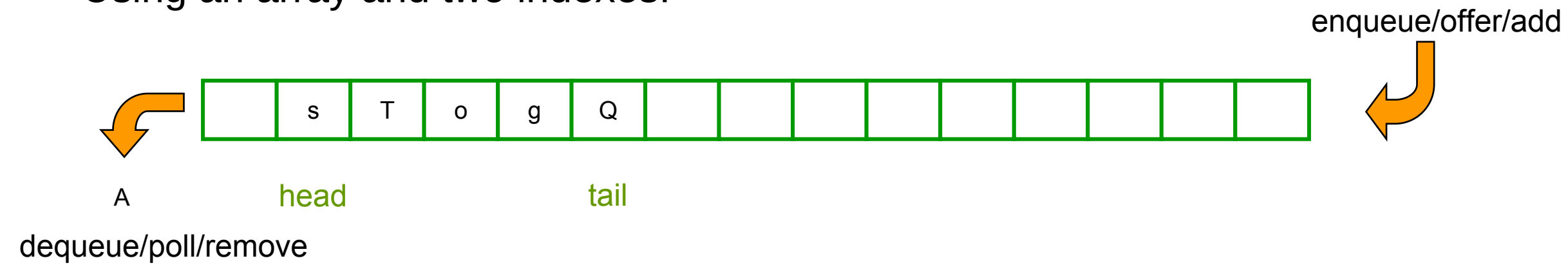
- Using an array and two indexes:



- Enqueue is fast ! O(1)
- Dequeue:
  - Return value at head
  - head++

Queues and efficiency

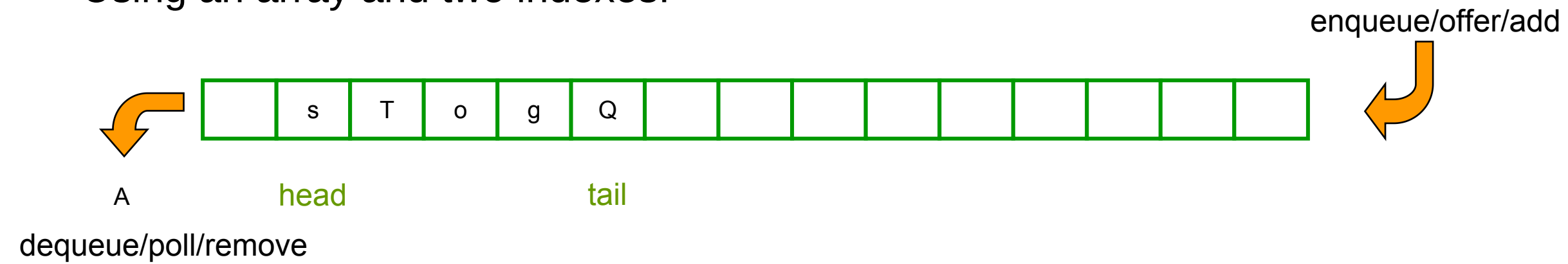
- Using an array and two indexes:



- Enqueue is fast ! O(1)
- Dequeue:
  - Return value at head
  - head++

Queues and efficiency

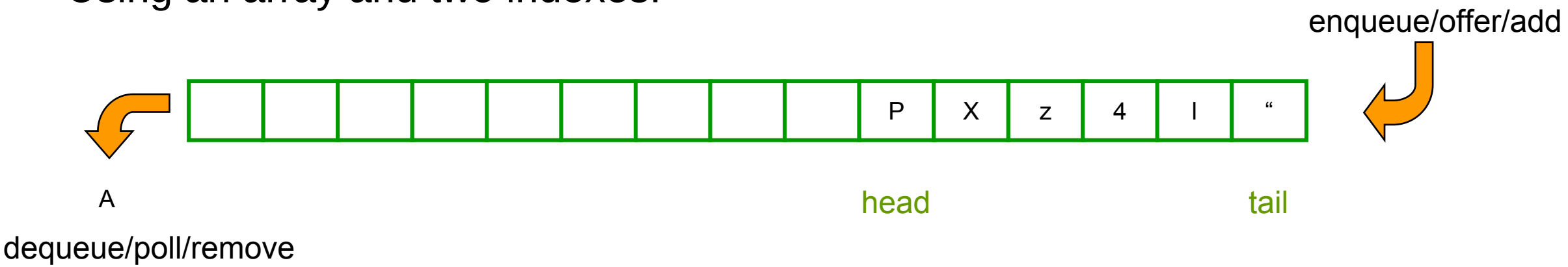
- Using an array and two indexes:



- Enqueue is fast ! O(1)
- Dequeue is fast ! O(1)

Queues and efficiency

- Using an array and two indexes:

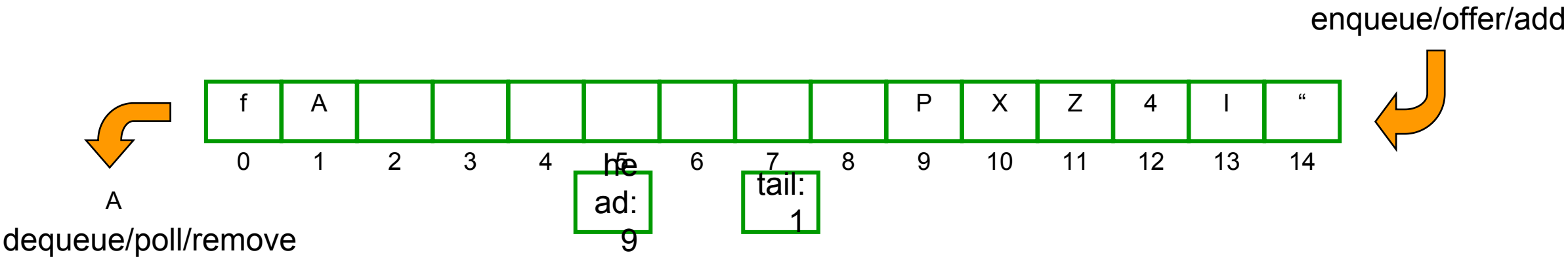


- Enqueue is fast ! O(1)
- Dequeue is fast ! O(1)

- What about space? (memory)

Queues and efficiency

- Using an array and two indexes:



- Enqueue is fast ! O(1)
- Dequeue is fast ! O(1)

- What about space? (memory)
- “Wrap around” at the end;

Java Implementations

- Java classes for Queue:
  - ArrayDeque `Queue<Patient> waitingRoom = new ArrayDeque<Patient>();`
  - LinkedList

- ArrayDeque is actually a kind of Deque – an extension of Queue:
  - Deque = Double Ended Queue
  - Add or remove at either end.
  - Includes Stacks and Queue

  - offer(e) = offerLast(e)
  - push(e) = offerFirst(e)
  - poll() = pop() = pollFirst()
  - - = pollLast()
  - peek() = peekFirst()
  - - = peekLast()