COMP250: Queues, deques, and doubly-linked lists

Lecture 18

Jérôme Waldispühl

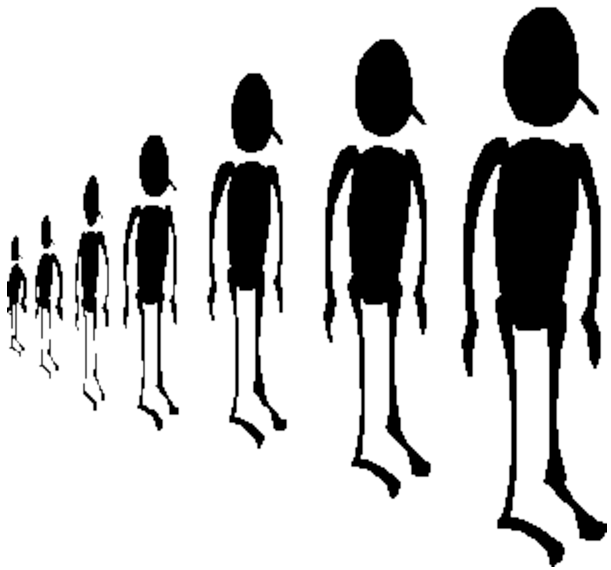
School of Computer Science

McGill University

Slides by Mathieu Blanchette

Queues

Rear of queue



Front of queue

Queue: First-in First-out data structure (FIFO)

Applications: Any first-come first-serve service

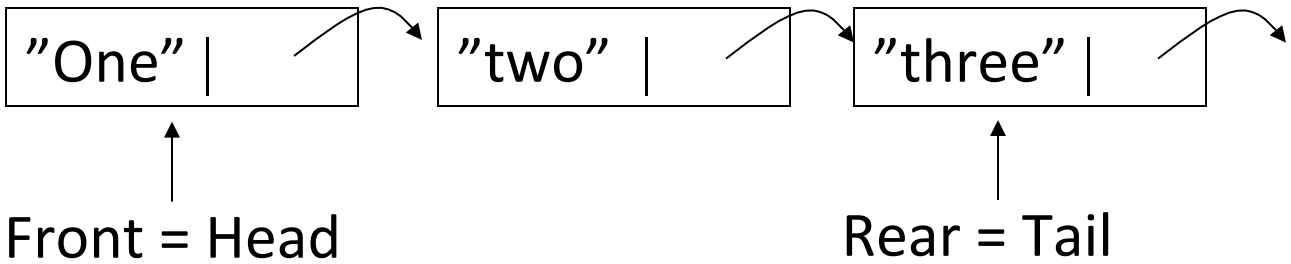
Queues operations

- void **enqueue** (Object o)
  - Add o to the rear of the queue
- Object **dequeue**()
  - Removes object at the front of the queue. Exception thrown if queue is empty (N.B. other implementations also return the object)
- Object **front**()
  - Returns object at the front of the queue but doesn't remove it from the queue. Exception if queue empty.
- int **size**()
  - Returns the number of objects in the queue
- boolean **isEmpty**()
  - returns True is queue is empty

Example

```
Queue q = new Queue()
q.enqueue(”one”)
q.enqueue(“two”)
q.enqueue(”three”)
print q.size()          “3”
print q.front()         “one”
q.dequeue()
print q.front()         “two”
q.dequeue()
print q.front()         “three”
print q.isEmpty()       False
```

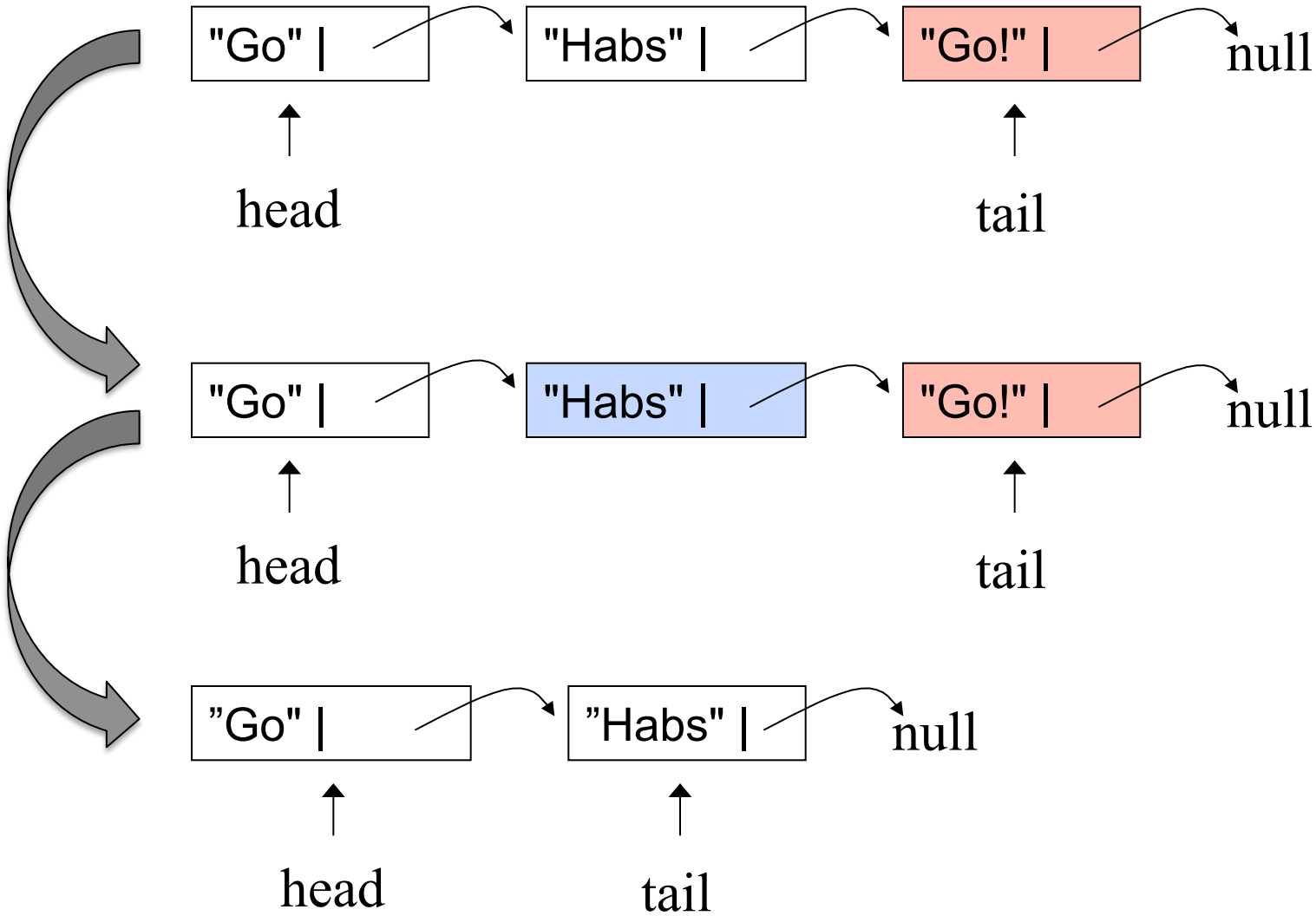
Queues with linked-lists



Queue operation	**Linked-list operation**	**Running time**
enqueue(Object o)	addLast(o)	O(**1**)
dequeue()	removeFirst()	
front()	getFirst()	
empty()	empty()	
size()	size()	O(**n**)

What would happen if we used instead the convention: "Front of queue = tail, Rear of queue = head" ?

removeLast() on singel linked list requires to access the predecessor of the last node in order to update Next.



Double-ended queues

- A double-ended queue (a.k.a. "deque") allows insertions and removal from the front and back
- Deque operations with linked-lists

Why?

  - Object getFirst()
  - Object getLast()
  - addFirst(Object o)
  - addLast(Object o)
  - boolean isEmpty()
  - Object removeFirst()

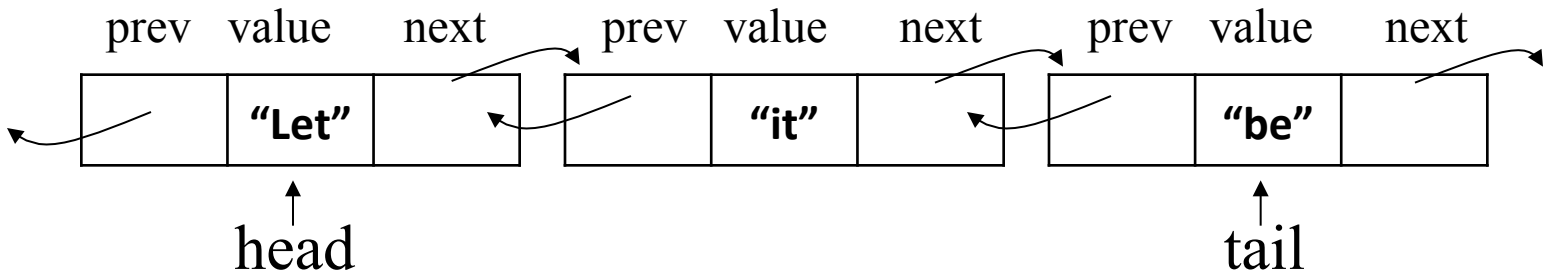
  O(**1**)

  - Object removeLast()
  - int size()

  O(**n**)

Deques and doubly-linked-lists

- Problem: removeLast takes time O(n) with linked lists
- To do it faster, each node has to have a reference to the *previous* node in the list



- class node {
  node prev, next;
  Object value;
  node(Object val, node p, node n);
  node getPrev(); void SetPrev(node n);
  node getNext(); void SetNext(node n);
  Object getValue(); void setValue(Object o); }

Operations on doubly-linked-lists

```
Object removeLast() throws Exception {
  if (tail==null) throw new Exception("Empty deque");
  Object ret = tail.getValue();
  tail = tail.getPrev();
  if (tail==null) { head=null; }
  else { tail.setNext(null); }
  return ret;  // If we return the object removed
}
```

Now in O(1)!

```
void addFirst(Object o) {
  node n = new node(o, null, head);
  if (head != null) { head.setPrev( n ); }
  else { tail = n; }
  head = n;
}
```

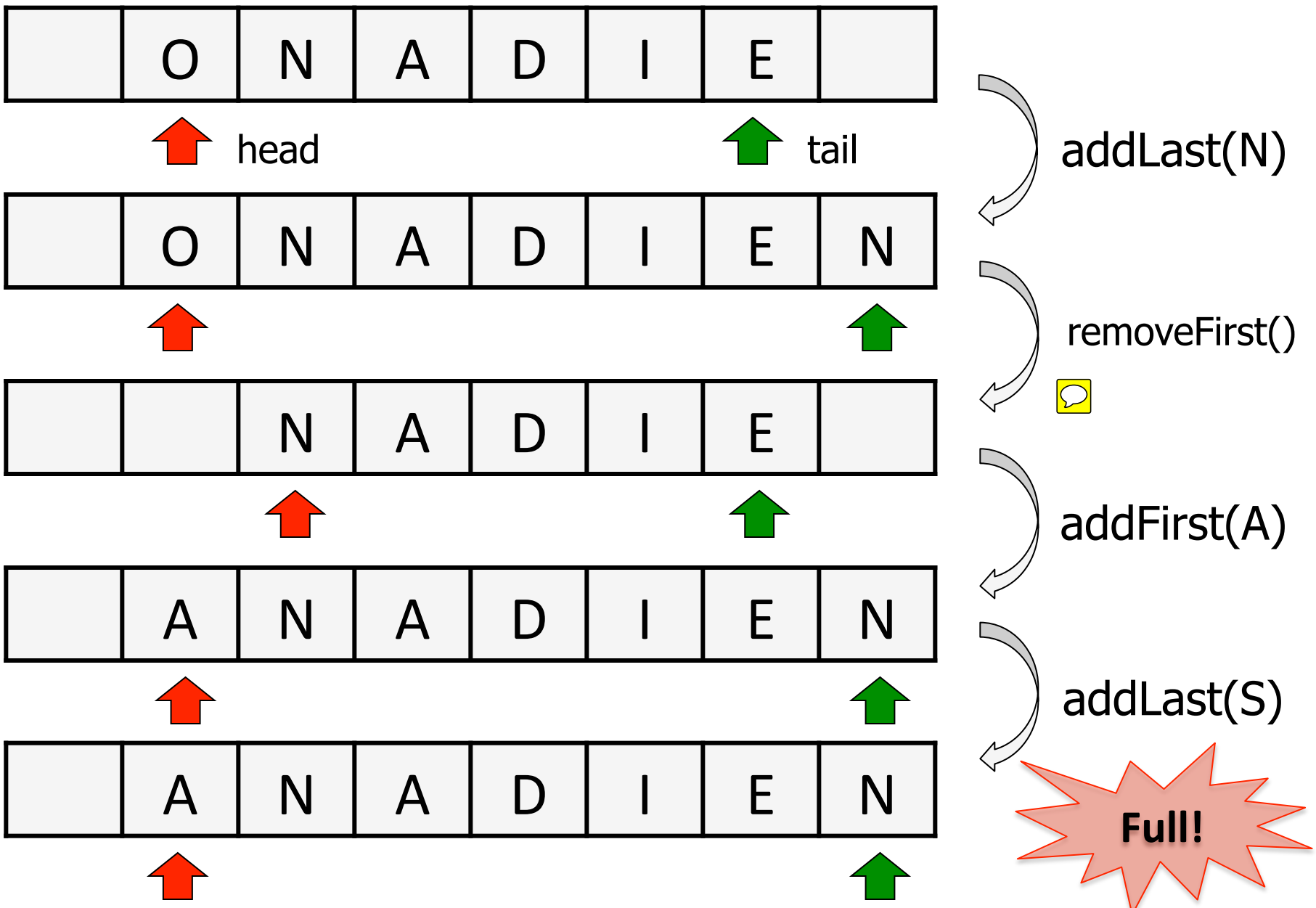
Exercise: Write all other deque methods using a doubly linked-list

Implementing deques with arrays

- **Suppose we know in advance the deque will never contain more than N elements.**
- We can use an array to store elements in the deque
- Keep track of indices for head and tail

- addLast(o) { tail = tail + 1; L[tail] = o; }
- addFirst(o) {head = head - 1; L[head] = o }
- removeLast { tail = tail – 1; }
- removeFirst { head = head + 1; }

Implementing deques with

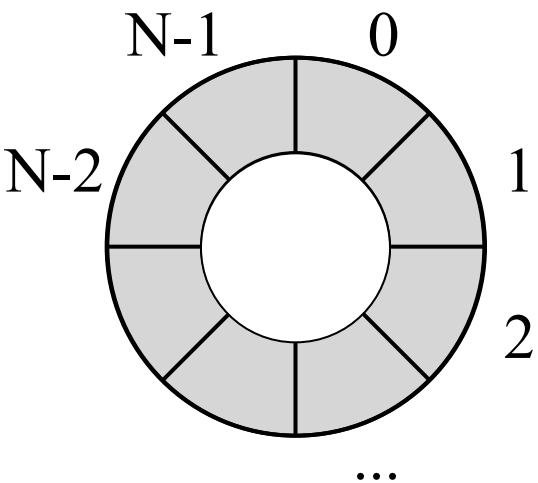


Rotating arrays

- Idea: To avoid outOfBounds exceptions, have indices "wrap around":

  (N-1) + 1 = 0

  0 - 1 = N-1



- Equivalent to arithmetic modulo N

  a mod N = rest of integer division a/N

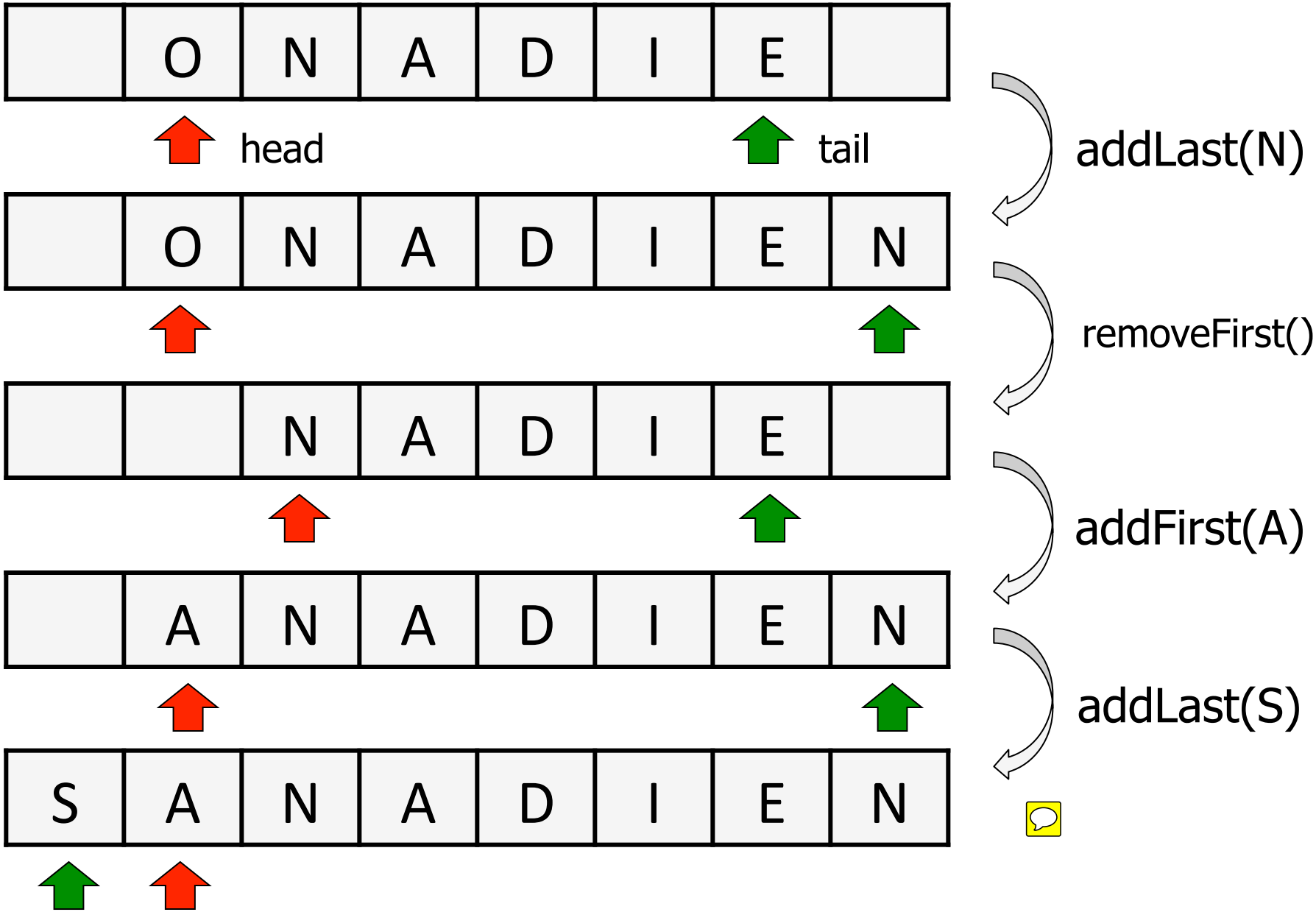
  3 mod 7 = 3

  7 mod 7 = 0

  10 mod 7 = 3

- With a rotating array, the deque will never go out of bounds, but may overwrite itself if we try to put more than N elements into it.
- How can we check if the deque is full (has N elements?)

Implementing deques with



Operations on deques with Array

- ```
  Enqueue(o) throw Exception {
      if ( isFull() ) {
          throw new Exception(“Full”) }
      tail = ( ( tail + 1 ) % N ) ;
      L[tail] = o;
  }
  ```
- ```
  Dequeue() {
      If ( isEmpty() ) {
          throw new Exception(“Empty”) }
      Object o = L[head];
      head = ( ( head + 1 ) % N );
      return o;  // If return object
  }
  ```

Exercise: Write all other deque methods using a rotating array. What are the index of an empty list?

Operations on deques with Array

- Head and Tail index are initialized at -1
- Enqueue and Dequeue must handle specific cases:
  - There is only one object in deque when we remove an element.
  - We insert the first element in the file (head & tail must be updated!)
  - Clean implementation of isEmpty() and isFull().