sit with anyone that looks friendly

CMPS 12B/M
Introduction to Data Structures

- Instructor: Nathan Whitehead

Stacks and Queues

- An Englishman, even if he is alone, forms an orderly queue of one.

  -*George Mikes*

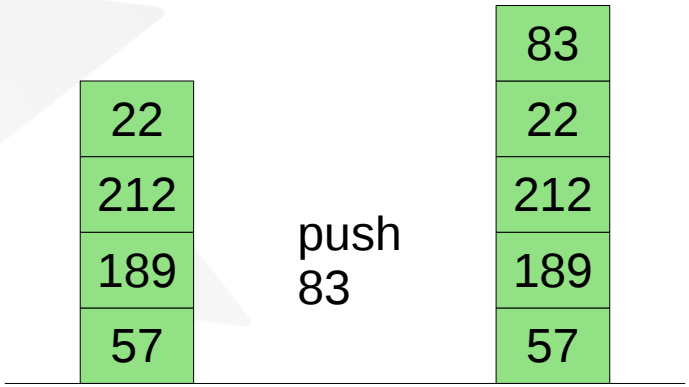
Different Kind of Structure

- Stacks and queues restrict access
  - Less capabilities compared to arrays
  - More suited to programming tasks than data tasks

interface matters

Stacks

- Only have access to top of stack
  - *Push* – add an element to top
  - *Pop* – remove top element
  - *Peek* – look at top without removing



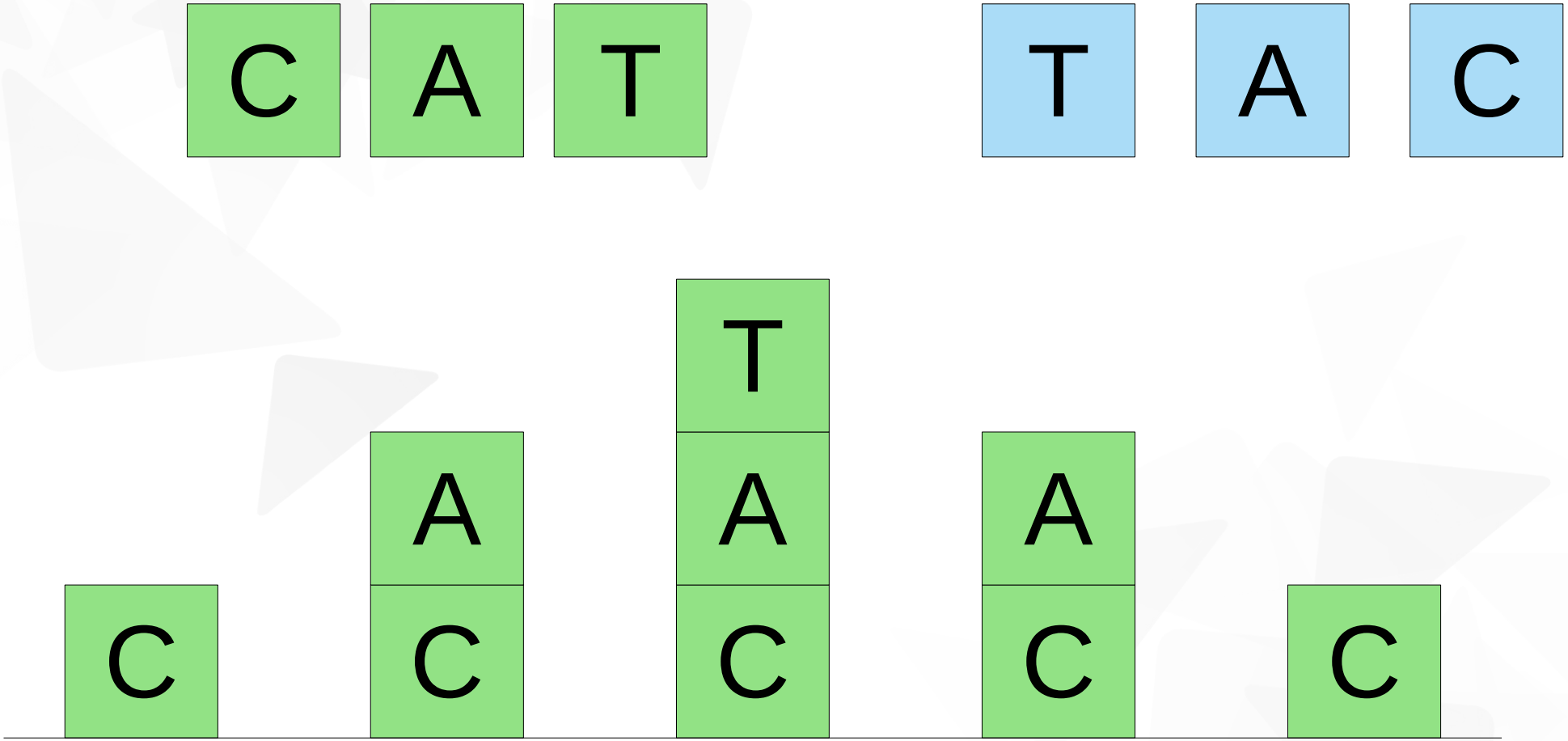
Stack Code

- Examples/Chap04/Stack/stack.java

Reversing Items

- Suppose you want to process items in reverse order
- We've solved this
  - Put all the elements in an array
  - Do a loop, swapping elements to reverse the array
  - Process elements in the array
- Easier
  - Put all elements in an array
  - Loop through array in reverse order

Reversing with a Stack

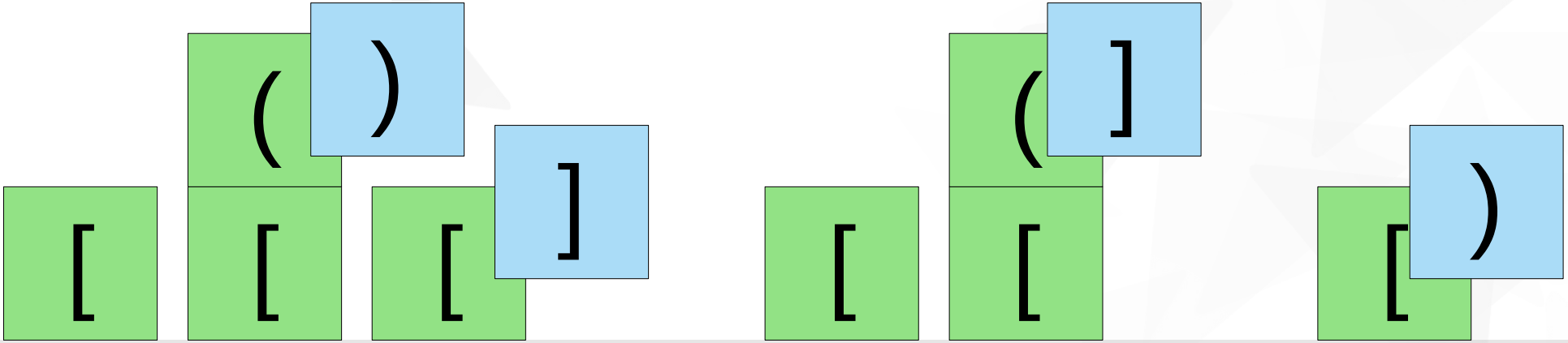


Matching Parentheses and Brackets

a	[	b	(	a	)	]	✓

a	[	b	(	a	]	]	✗

a	[	b	)	a	(	]	✗



Bracket Checker Code

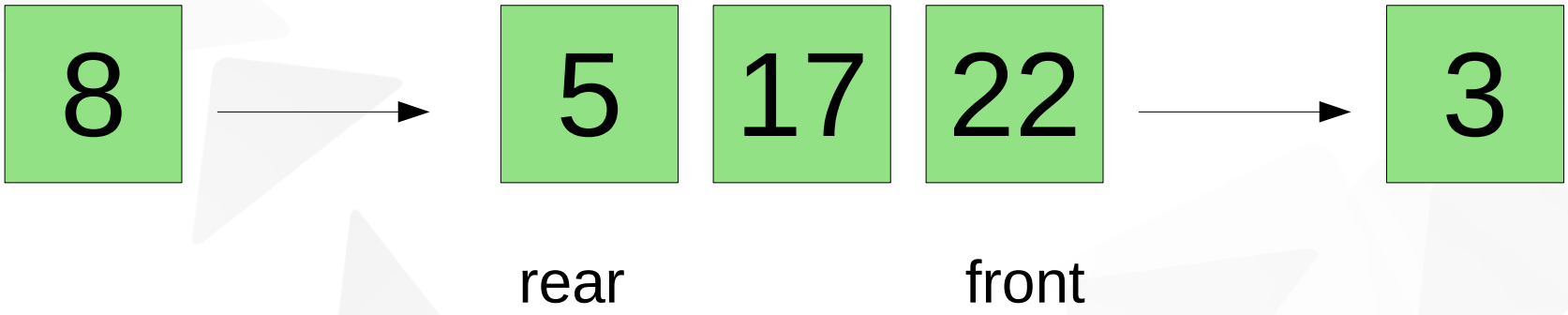
- Examples/Chap04/Brackets/brackets.java

Stack Efficiency

- How efficient are stacks?
  - Push, pop, peek
    - No comparisons needed
    - One array access
    - Doesn't depend on how many elements in stack
    - O(1) time for all operations

Queue

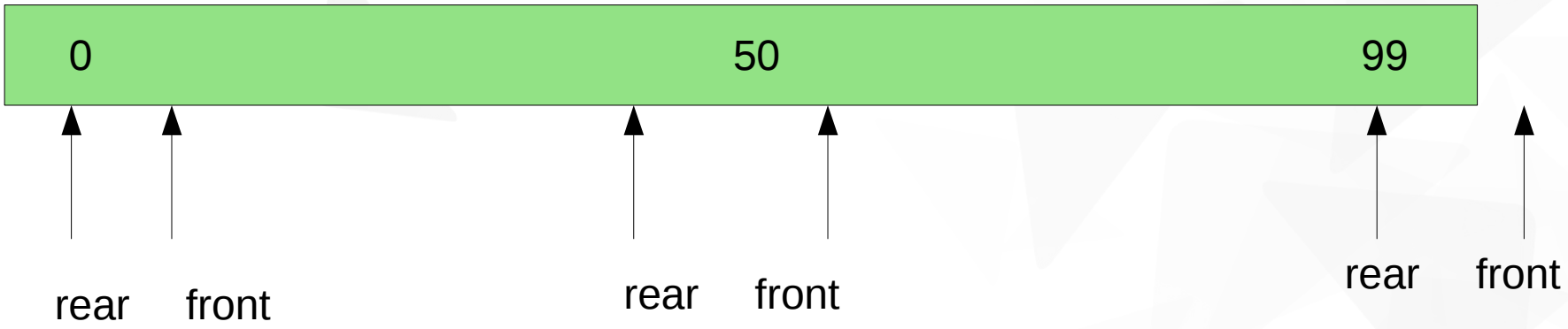
- Waiting in line
  - Arrive at the rear
  - Leave at the front
  - FIFO (first-in-first-out)



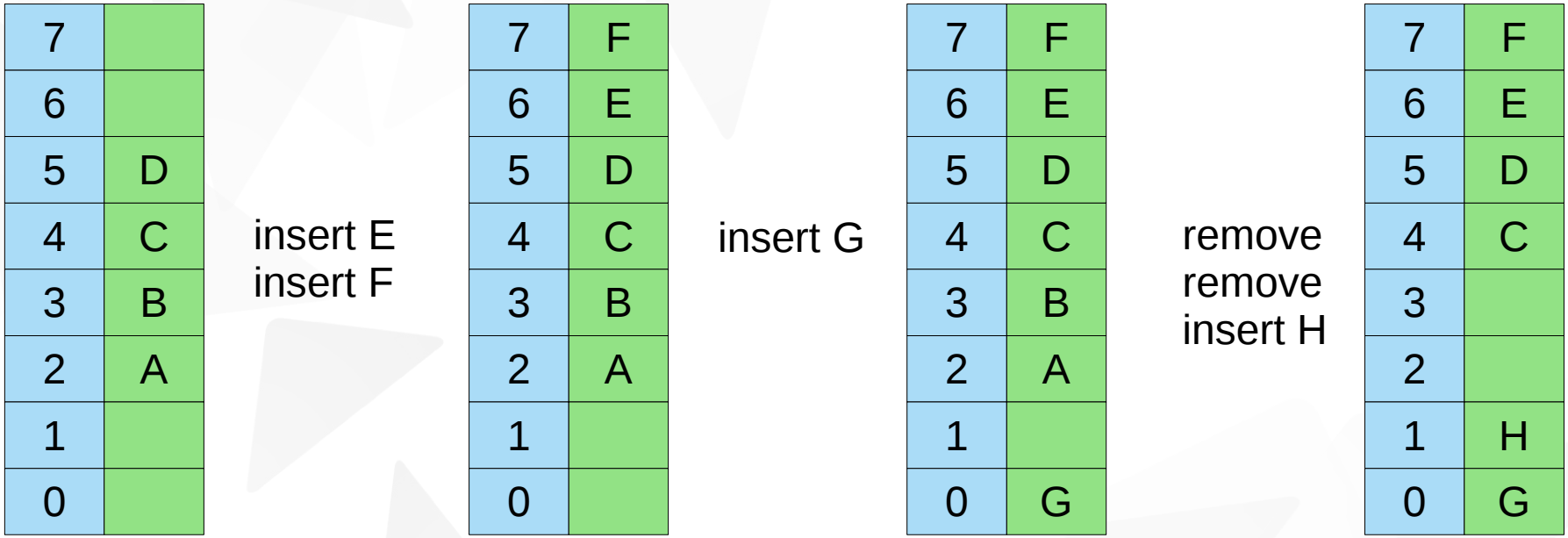
incredibly useful for keeping track of events, work to do, etc.

Queue

- Methods
  - *Insert* – add to rear of queue
  - *Remove* – remove from front of queue
  - *Peek* – look at front of queue
- Implementation
  - Maybe an array?
  - Front and rear keep increasing over time



Circular Buffer



wraparound

Circular Queue Code

- Examples/Chap04/Queue/queue.java

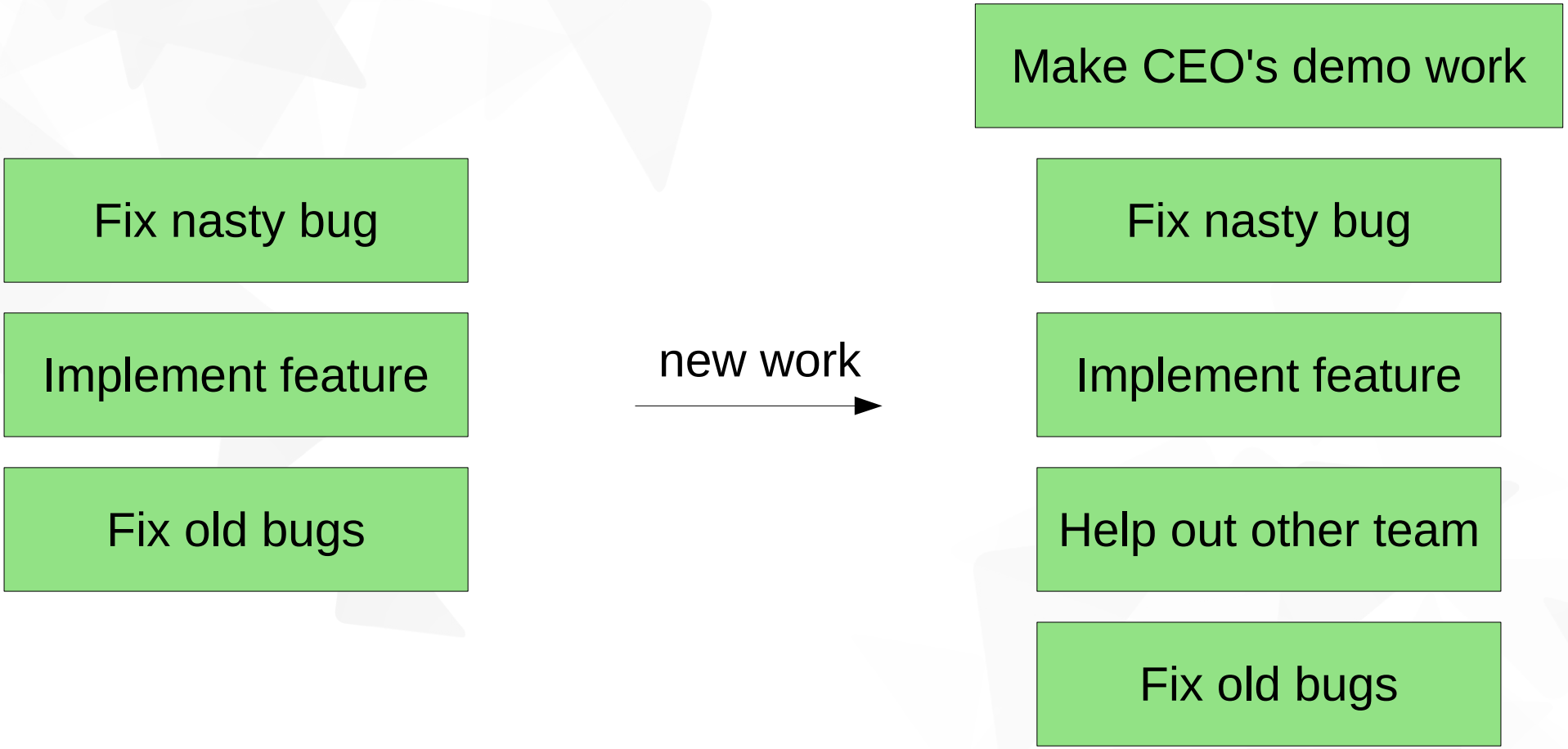
Queue Efficiency

- How efficient are circular queues?
  - Insert
    - No comparisons
    - One array access
    - O(1)
  - Remove
    - No comparisons
    - One array access
    - O(1)

Deque

- With circular techniques we can insert and remove from front or rear of buffer
  - Left and right indexes
  - insertLeft, removeLeft, insertRight, removeRight
  - insertRight and removeRight == stack
  - insertLeft and removeRight == queue
- Deques are generalization of stacks and queues
  - Commonly provided by libraries
  - Usually a stack or queue is what problem needs

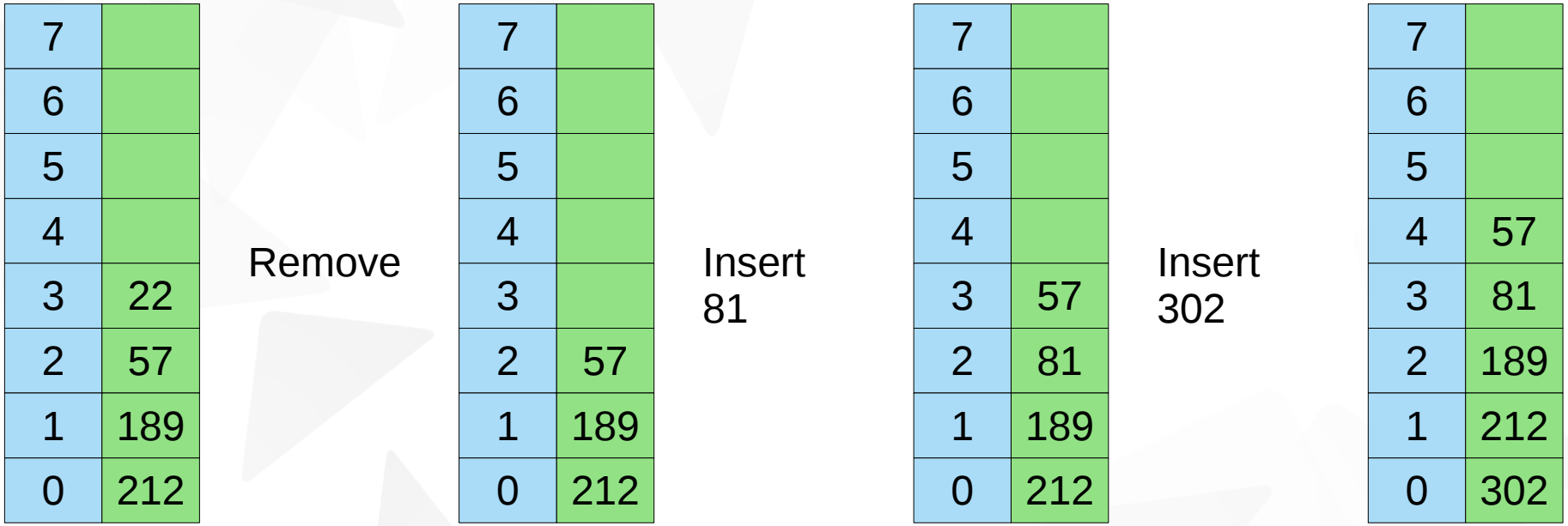
Priority Queue



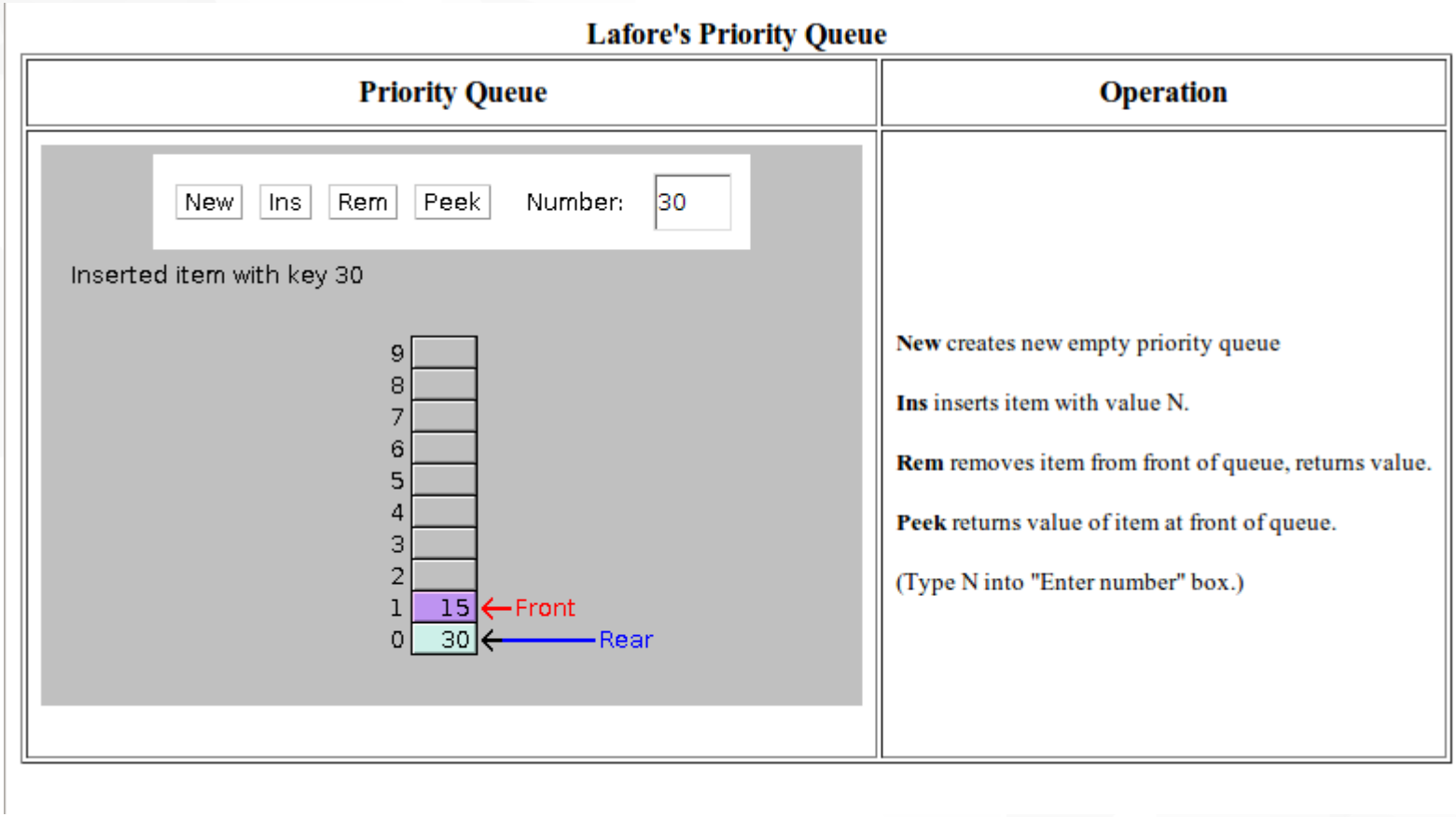
Priority Queue

- Items go into queue
- Items come out of queue lowest first
  - Not FIFO
- One implementation
  - Keep sorted array
  - *Remove* – take out element in front (it is smallest)
  - *Insert* – find proper place to put new item, make room for it

Priority Queue



Priority Queue Demo



http://www2.latech.edu/~box/ds/PriorityQ/PriorityQ.html

PriorityQueue code

- Examples/Chap04/PriorityQ/priorityQ.java

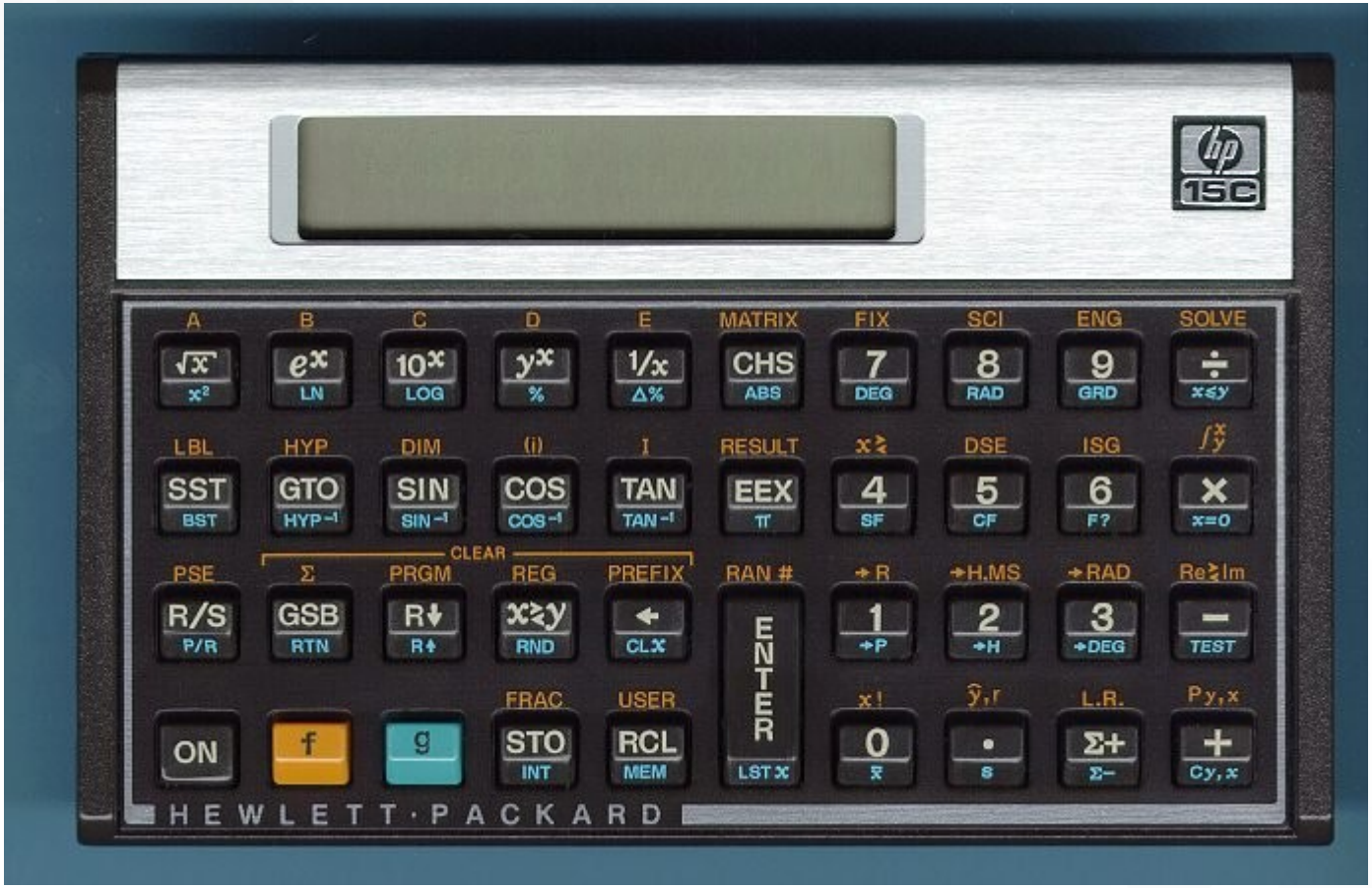
Efficiency of Priority Queue

- Our first implementation
  - Remove
    - No comparisons
    - No copying
    - One array access
    - O(1)
  - Insert
    - Find where to put it
    - Shift elements out of the way
    - Worst case requires shifting *n* elements
    - O(*n*) comparisons and copies

there is another way

we'll see later

Parsing Arithmetic Expressions



http://hp15c.com/

RPN

- Reverse Polish Notation
  - also known as postfix
- Instead of (3 + 4) do 3 4 +

(3 + 11) + 5	→	3 11 + 5 +
1 + 2 * 3	→	1 2 3 * +
5 + (1 + 2) * 4 – 3	→	5 1 2 + 4 * + 3 -

http://en.wikipedia.org/wiki/Reverse_Polish_notation

Stack Operations

5 1 2 + 4 * + 3 -

Input	Operation	Stack	Comment
5	push	5	
1	push	5 1	
2	push	5 1 2	
+	add	5 3	pop two, add, push
4	push	5 3 4	
*	mult	5 12	pop two, mult, push
+	add	17	pop two, add, push
3	push	17 3	
-	sub	14	pop two, sub, push

Challenge

Convert the following expressions from infix to postfix

((2 + 4) * 7) + 3 * (9 – 5)

(1 + 1 + 2 + 1) * 3

2 * (1 + (2 * (1 + 1)))

Challenge Two

Convert the following postfix expressions back to infix

1 3 4 + * 2 *

2 4 + 7 * 3 9 5 - * +

1 1 1 1 1 + + + + 2 + 2 + 2 + 2 +

Parsing Infix

- Computing results of infix expression
  - Bit tricky
  - Requires scanning forward and back
  - Keep track of where we are, current state of parse
- Computing results of postfix expression
  - Just use a stack
  - Natural for a program

- Converting infix expressions to postfix requires parsing
  - Compilers do this all the time

Linked Lists

- "You can either have software quality or you can have pointer arithmetic, but you cannot have both at the same time."

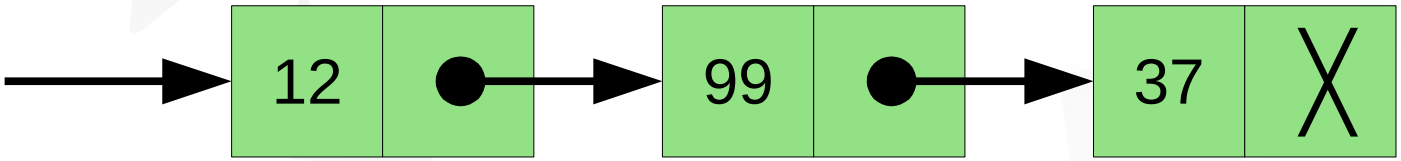
  Bertrand Meyer

What is a linked list?

- Linked lists are versatile data structure
  - 2nd most common after arrays/vectors
- Chain elements together with arrows going from one to the next
  - Can rearrange arrows to insert/delete anywhere in list
    - No shifting required 😄
  - Give up the ability to directly index into list ☹

Singly Linked List

- Three element list of integers
- Singly-linked
  - Each node has a value and a next node
  - Or next node is *null*, represented by X



Node

```
class Node {
    public int value;
    public Node next;
}
```

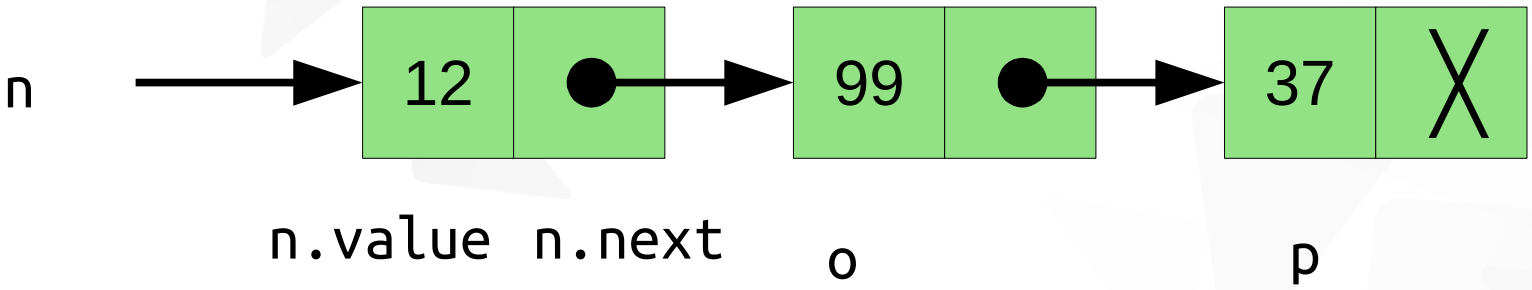
kind of like recursion

defining a class in terms of itself

Building a List

```
class Node {
   public int value;
   public Node next;
}
```

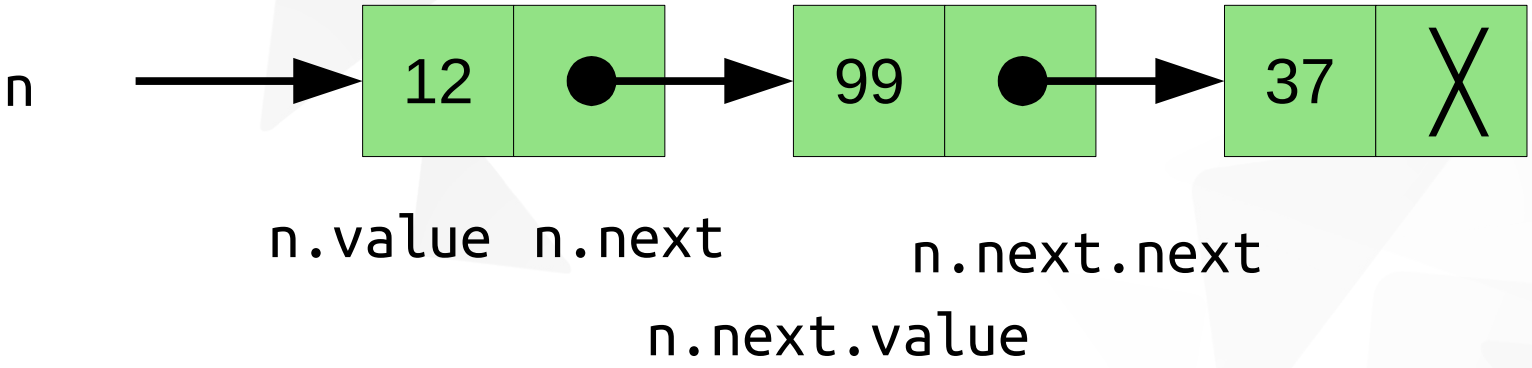
```
Node p = new Node();
p.value = 37;
p.next = null;
Node o = new Node();
o.value = 99;
o.next = p;
Node n = new Node();
n.value = 12;
n.next = o;
```



Building a List

```
class Node {
   public int value;
   public Node next;
}
```

```
Node n = new Node();
n.value = 12;
n.next = new Node();
n.next.value = 99;
n.next.next = new Node();
n.next.next.value = 37;
n.next.next.next = null;
```



Linked List Class

```
class Node {
   public int value;
   public Node next;
}
```

```
class LinkList {
   private Node first;

   public LinkList() {
      first = null;
   }

   public void insertFirst(int value) {
      ...
   }
   .
   .
   .
}
```

Linked List Class

```
class Node {
    public int value;
    public Node next;
}
```

```
class LinkList {
    private Node first;

    public LinkList() {
        first = null;
    }

    public void insertFirst(int value) {
        ...
    }
    .
    .
    .
}
```

implement

insertFirst

test

```
LinkList lst = new LinkList();
lst.insertFirst(37);
lst.insertFirst(99);
lst.insertFirst(12);
```

The End