COMP 250
INTRODUCTION TO COMPUTER SCIENCE

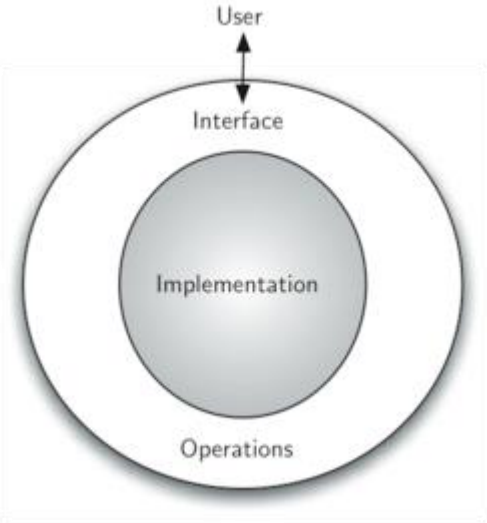
Lecture 16 – Queues

Roman Sarrazin-Gendron, Winter 2020

Slides very much based on Michael Langer and Giulia Alberini

REMINDER: ABSTRACT DATA TYPES

- Type/class of object defined by values and operations from the perspective of the user

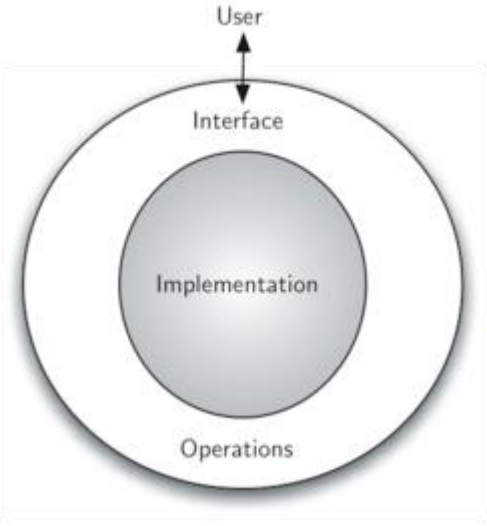


- Completely independent of implementation and language (more abstract than data structures)

- Puts the focus on the essential tasks towards solving a problem.

REMINDER: ABSTRACT DATA TYPES (ADT)

- Type/class of object defined by values and operations from the perspective of the user



- Completely independent of implementation and language (more abstract than data structures)

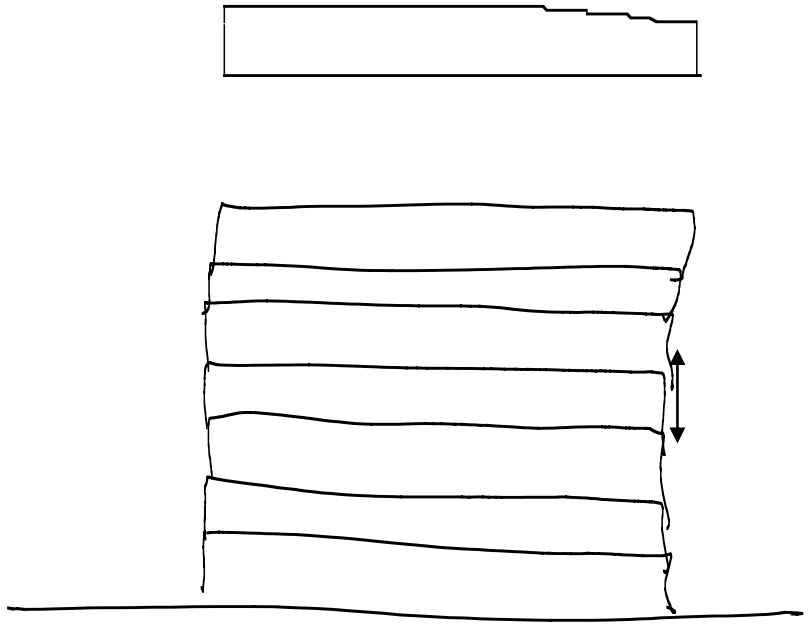
- **Goal :** force the user to only use efficient operations by restricting number of functions available.
  - Many ADTs are specialized to specific problems/data

REMINDER : STACKS

Stack

push(e)
pop()

LIFO
(last in, first out)



- By implementing a Stack with an ArrayList and making push and pop execute addLast and removeLast, we make sure the user never uses inefficient methods like addFirst, and guarantee a good execution time.

ADT (ABSTRACT DATA TYPE)

- List

  add(i,e), remove(i), get(i), set(i), …..

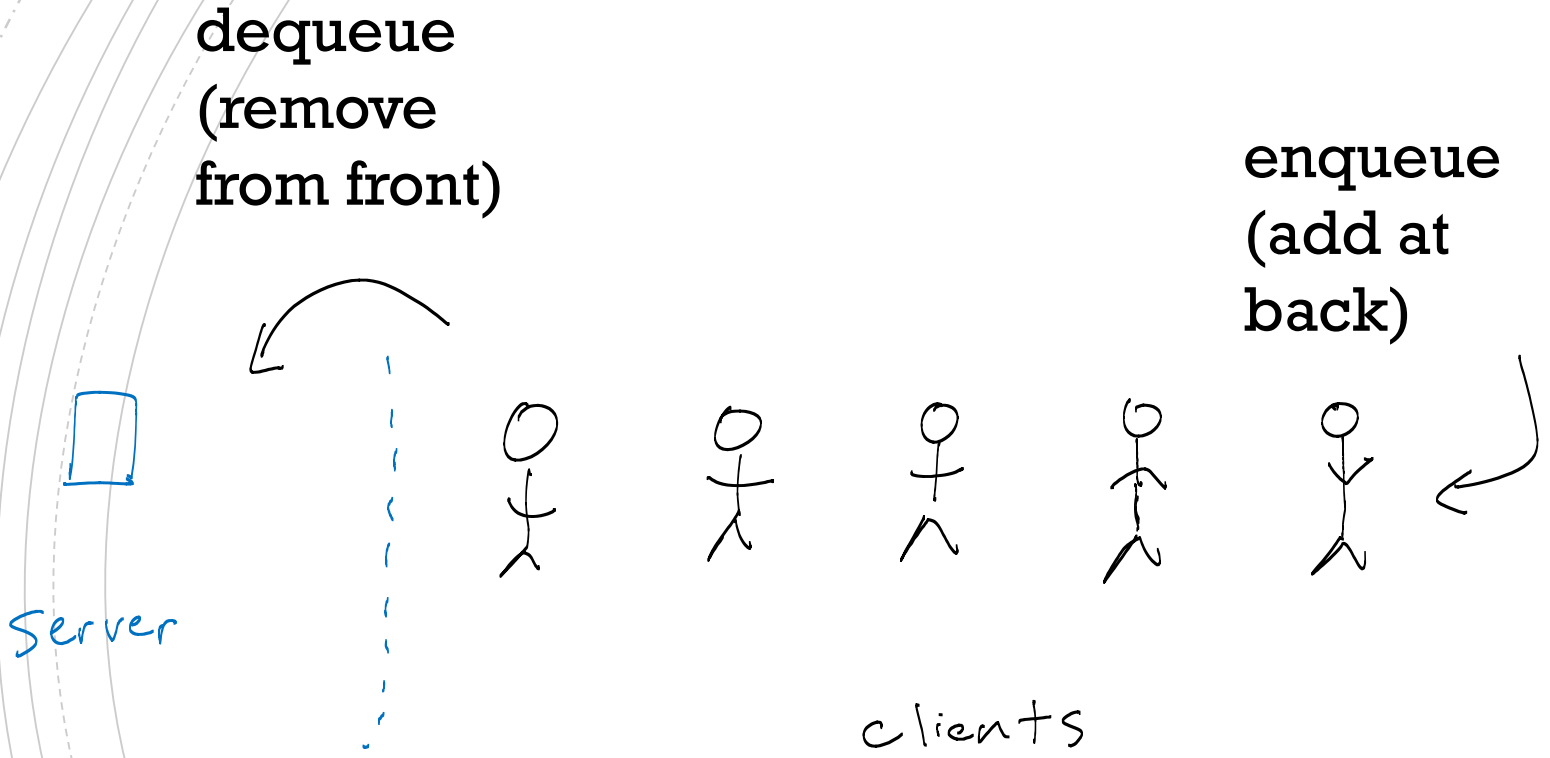
- Stack

  push, pop(), ..

- Queue

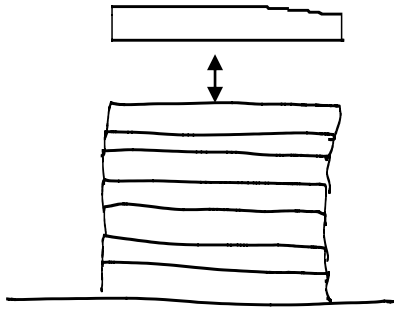
  enqueue(e), dequeue()

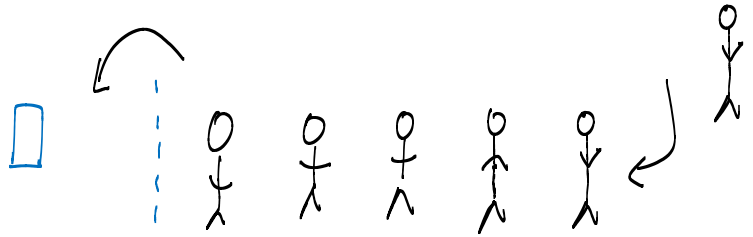
QUEUE


dequeue
(remove
from front)
enqueue
(add at
back)
server
clients

EXAMPLES

- keyboard buffer
- printer jobs
- CPU processes (applications do not run in parallel)
- web server
- .....





Stack

push(e)

pop()

LIFO
(last in, first out)

Queue

enqueue(e)

dequeue()

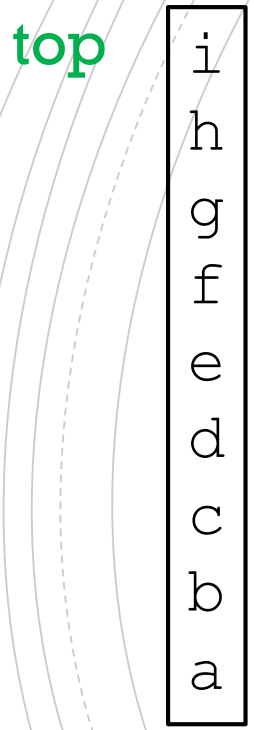
FIFO
(first in, first out)
"first come, first serve"

EXERCISE : IMPLEMENT A QUEUE WITH A STACK

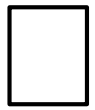
```
enqueue( e ){      // add element
        :
}

dequeue( ) {       // remove 'oldest' element
        :
}
```

Write pseudocode for these two methods that uses a stack, namely use the operations push(e) , pop(), isEmpty() .

HINT FOR EXERCISE

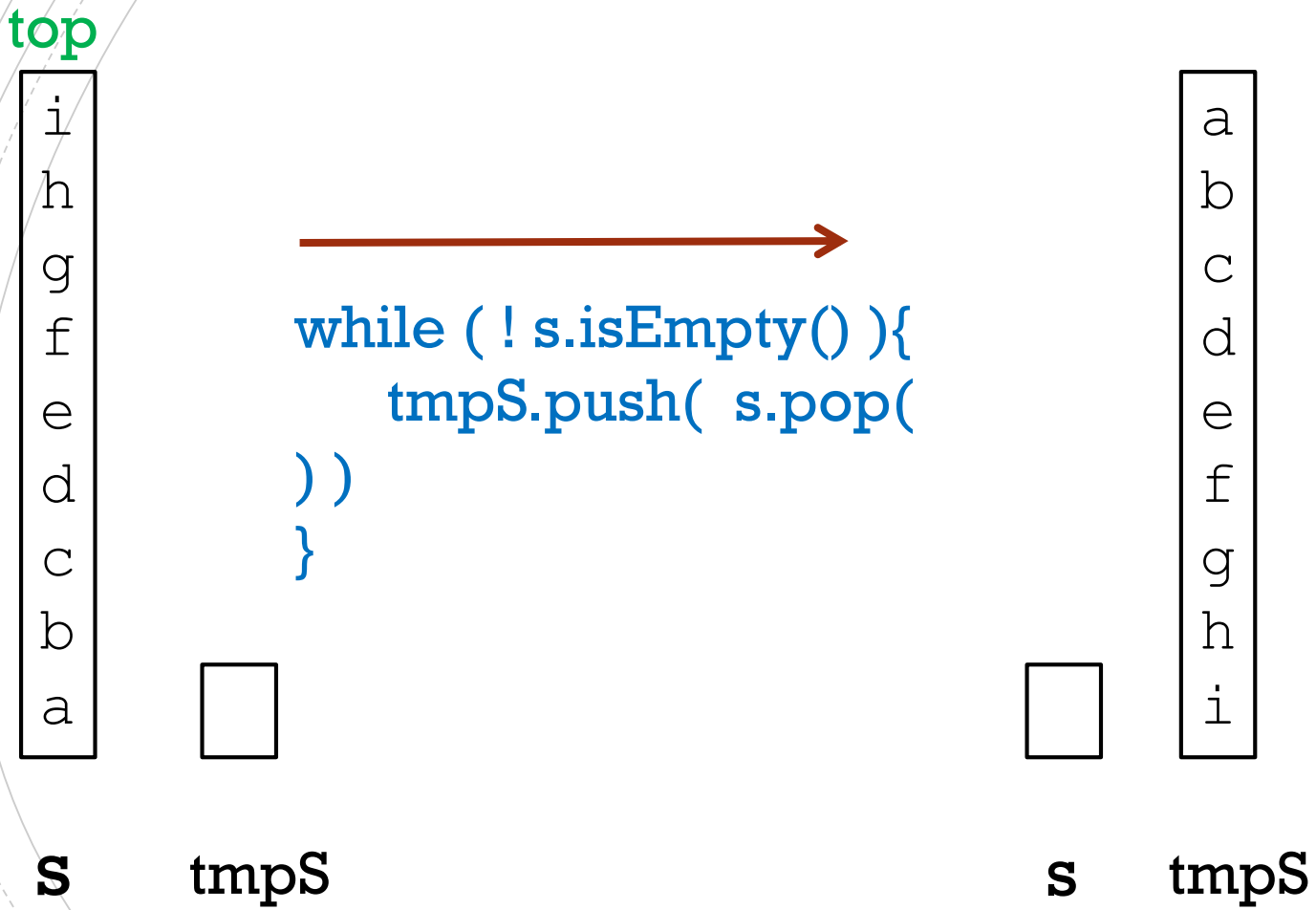


Use a second stack.



S tmpS

HINT FOR EXERCISE



QUEUE EXAMPLE

`enqueue( a )`	`a`
`enqueue( b )`	`ab`
`dequeue( )`	`b`

QUEUE EXAMPLE

enqueue(a)	a
enqueue(b)	ab
dequeue( )	b
enqueue(c)	bc
enqueue(d)	bcd
enqueue(e)	bcde
dequeue( )	cde
enqueue(f)	cdef
enqueue(g)	cdefg

HOW TO IMPLEMENT A QUEUE?

enqueue(e) dequeue()

singly linked list

doubly linked list

array list

HOW TO IMPLEMENT A QUEUE?

	enqueue(e)	dequeue()
singly linked list	addLast(e)	removeFirst()
doubly linked list	(unnecessary)	
array list		

HOW TO IMPLEMENT A QUEUE?

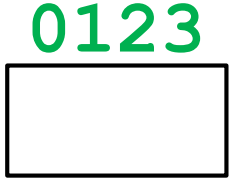
	enqueue(e)	dequeue()
singly linked list	addLast(e)	removeFirst()
doubly linked list	(unnecessary)	
array list	addLast(e)	**removeFirst()**

SLOW

QUEUES AND ARRAYLISTS

Implementing a queue with an array list. **(BAD)**

length = 4



`enqueue( a )`	`a---`
`enqueue( b )`	`ab--`
`dequeue( )`	`b---`

Requires shift

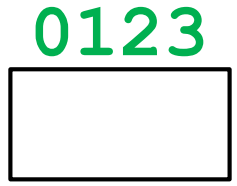
QUEUES AND ARRAYLISTS

length = 4

enqueue(a)	a---	
enqueue(b)	ab--	
dequeue( )	b---	Requires shift
enqueue(c)	bc--	
enqueue(d)	bcd-	
enqueue(e)	bcde	
dequeue( )	cde-	**Requires shift**

Implementing a queue with an array list. (BAD)

length = 4



enqueue(a)	a---
enqueue(b)	ab--
dequeue( )	b---
enqueue(c)	bc--
enqueue(d)	bcd-
enqueue(e)	bcde
dequeue( )	cde-
enqueue(f)	cdef
enqueue(g)	cdefg---

requires expansion

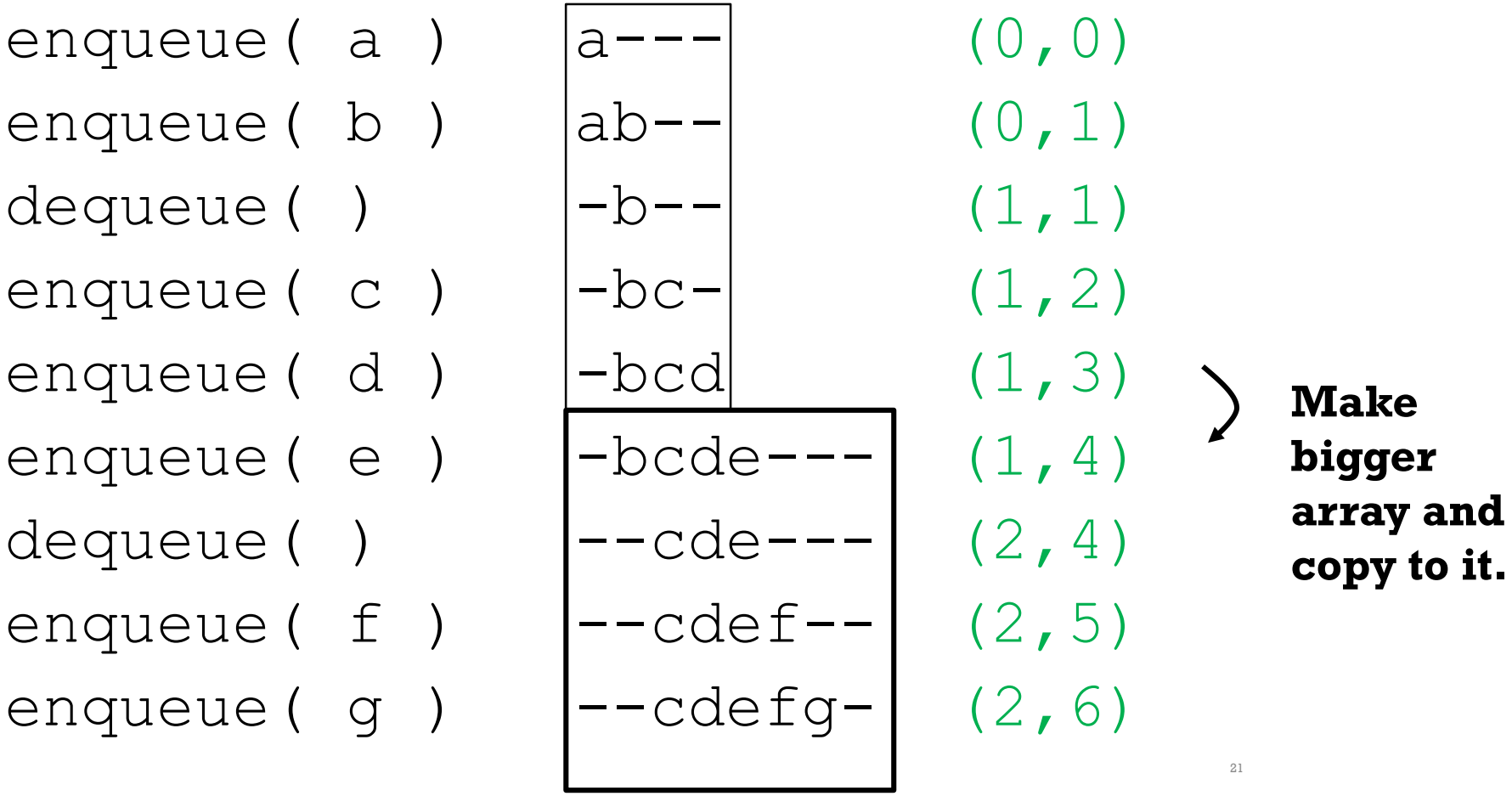
Implementing a queue with an **expanding array.** **(also BAD)**

Use head and tail indices
(tail = head + size – 1)

enqueue(a)	a---	(0,0)
enqueue(b)	ab--	(0,1)
dequeue( )	-b--	(1,1)
enqueue(c)	-bc-	(1,2)
enqueue(d)	-bcd	(1,3)
enqueue(e)	?	

Implementing a queue with an **expanding array**. **(also BAD)**

Use head and tail indices
(tail = head + size – 1)



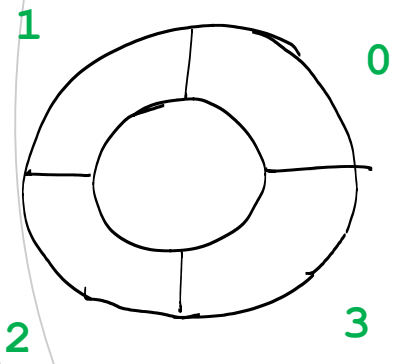
ALTERNATIVES?

- An expanding array is an inefficient usage of space.
- A better idea is….

CIRCULAR ARRAY

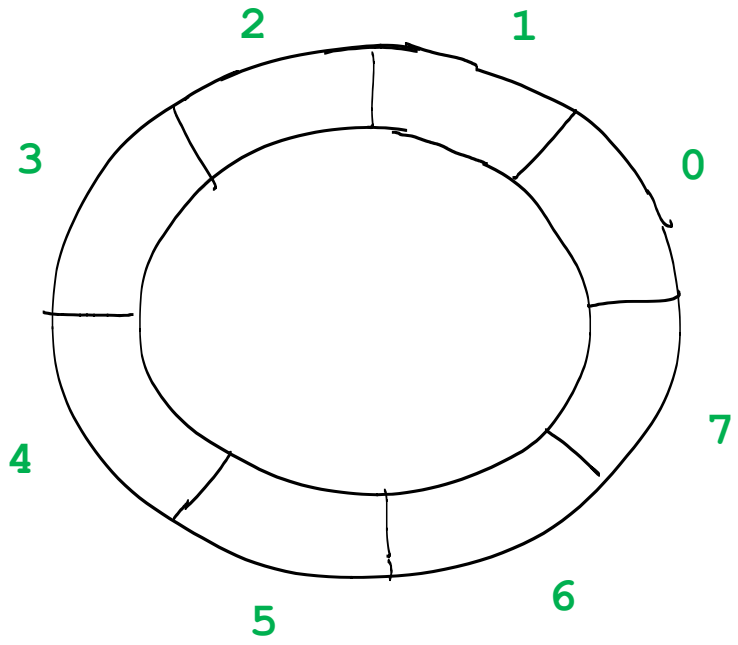
length = 4

0123



length = 8

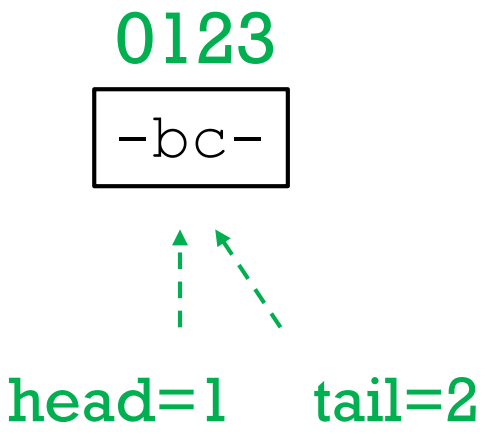
01234567

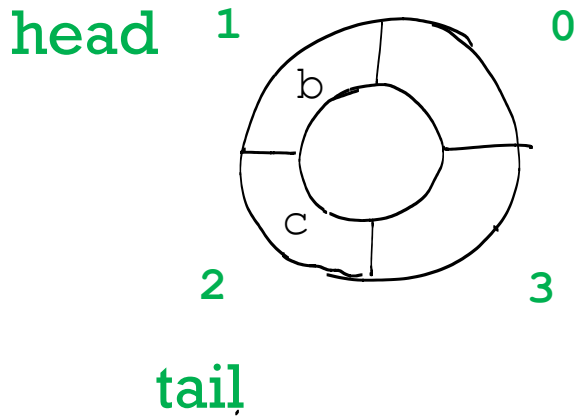


CIRCULAR ARRAY

tail = (head + size – 1) % length

```
enqueue( a
enqueue( b )
dequeue()
enqueue( c )
```

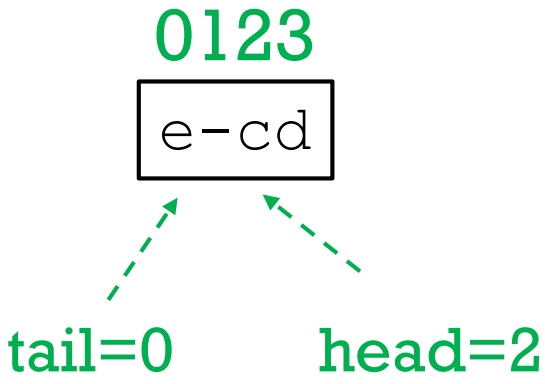


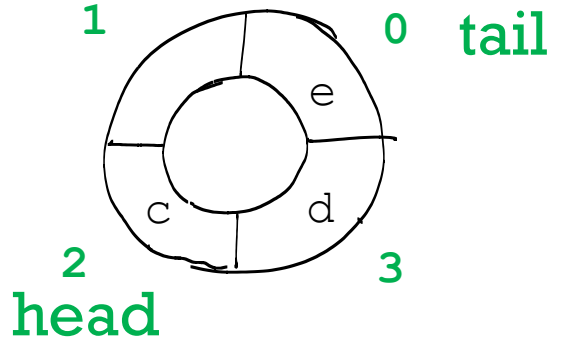


CIRCULAR ARRAY

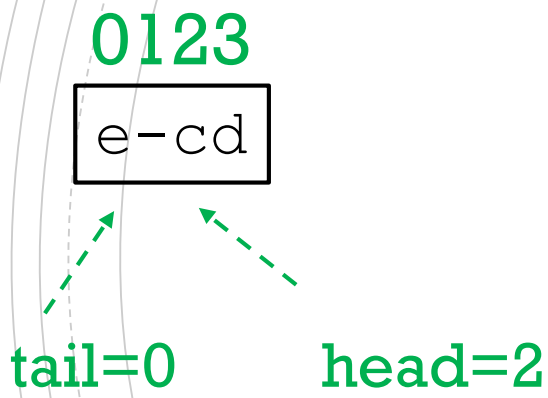
tail = (head + size – 1) % length

```
enqueue( a )
enqueue( b )
dequeue( )
enqueue( c )
enqueue( d )
enqueue( e )
dequeue()
```





CIRCULAR ARRAY



tail = (head + size – 1) % length

```
enqueue( element ){
      if  (size < length)
           queue[ (tail + 1) % length] = element
      else ….  // coming up
      size = size+1
}
```

```
dequeue(){   //  check if empty omitted
      element = queue[head]
      head = (head + 1) % length
      size = size-1
      return element
}
```

Implementing a queue with a circular array (GOOD)

tail = (head + size – 1) % length

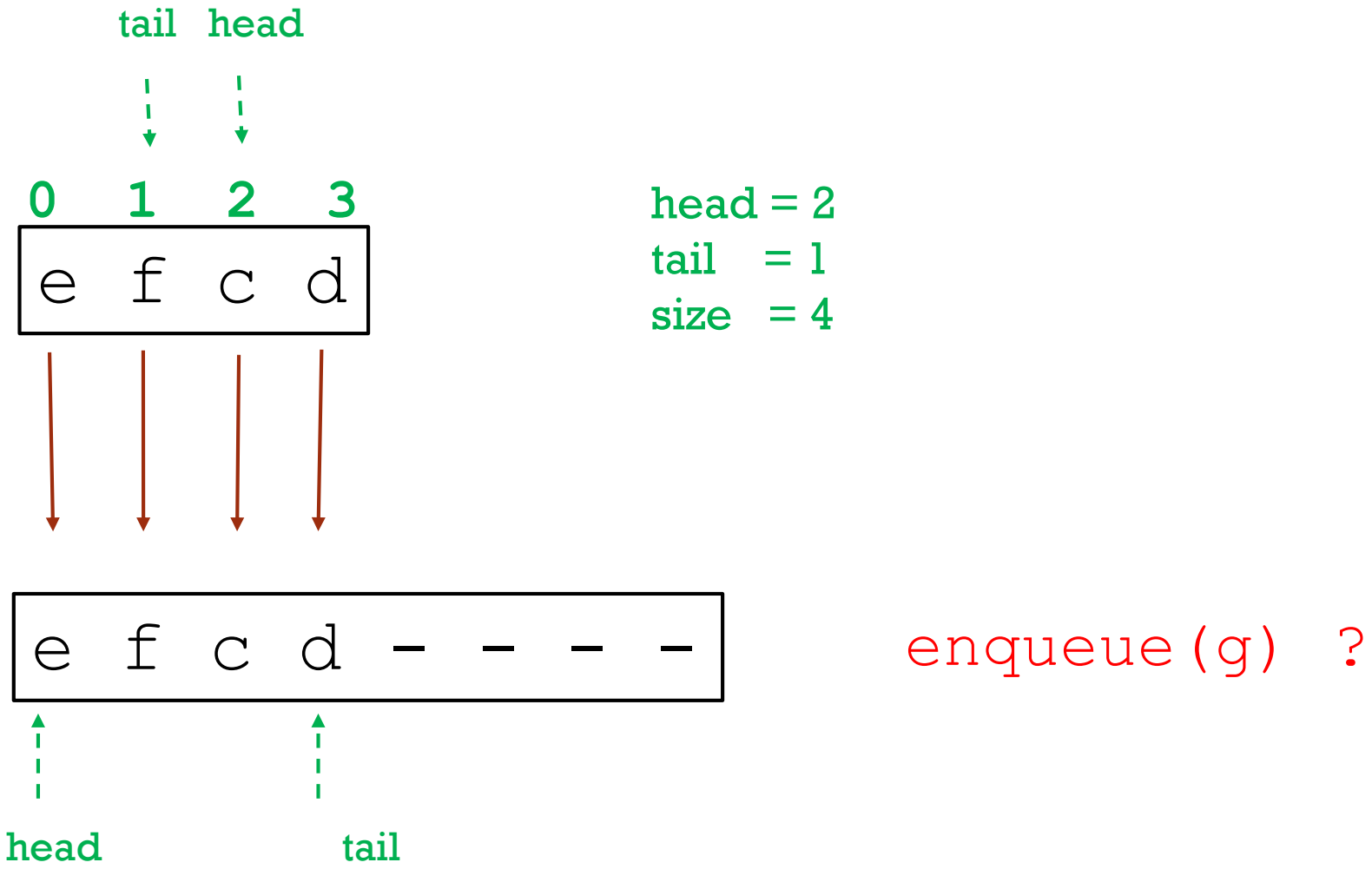
	array	(head, tail, size)
enqueue(a)	a---	(0, 0, 1)
enqueue(b)	ab--	(0, 1, 2)
dequeue()	-b--	(1, 1, 1)
enqueue(c)	-bc-	(1, 2, 2)
enqueue(d)	-bcd	(1, 3, 3)
enqueue(e)	ebcd	(1, 0, 4)
dequeue()	e-cd	(2, 0, 3)
enqueue(f)	efcd	(2, 1, 4)

Implementing a queue with a circular array

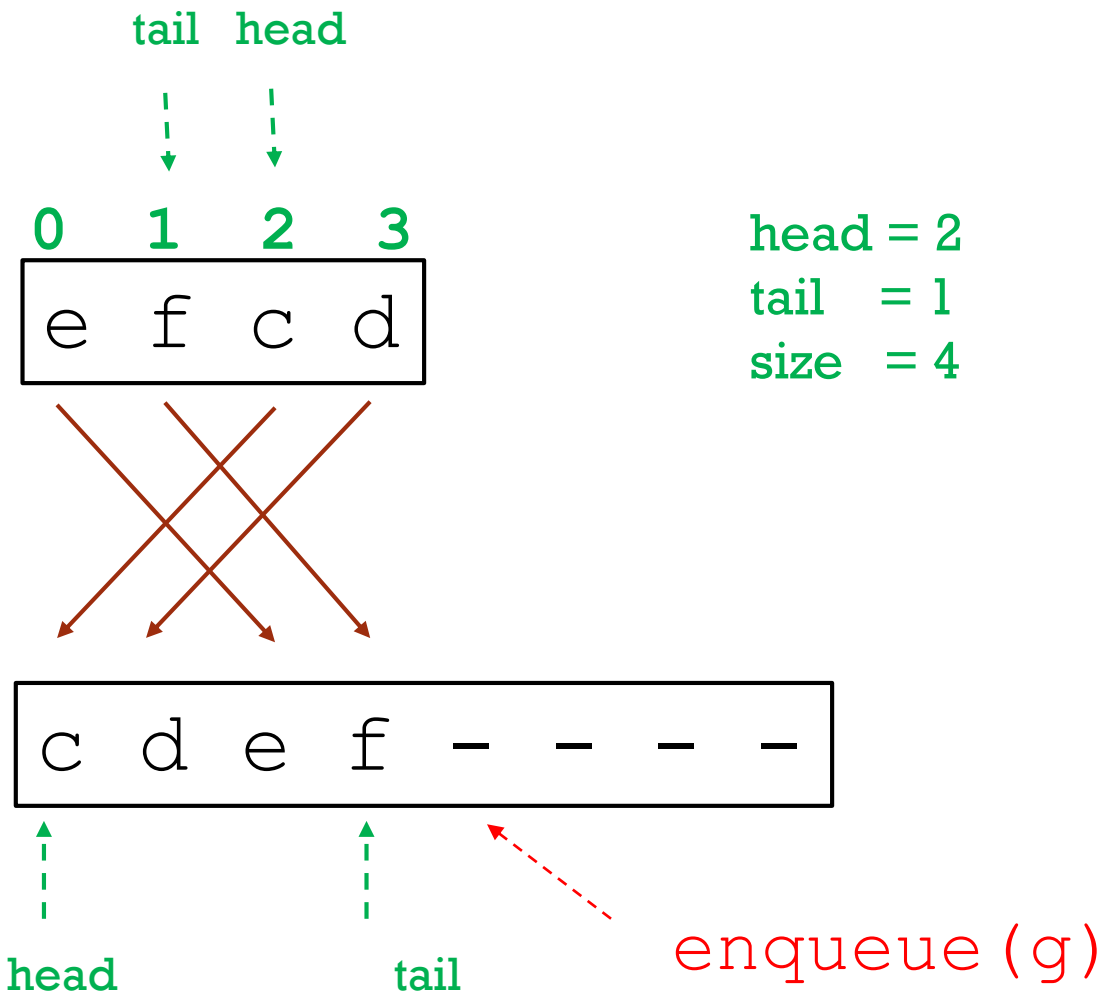
tail = (head + size – 1) % length

	array	(head, tail, size)
enqueue(a)	a---	(0, 0, 1)
enqueue(b)	ab--	(0, 1, 2)
dequeue()	-b--	(1, 1, 1)
enqueue(c)	-bc-	(1, 2, 2)
enqueue(d)	-bcd	(1, 3, 3)
enqueue(e)	ebcd	(1, 0, 4)
dequeue()	e-cd	(2, 0, 3)
enqueue(f)	efcd	(2, 1, 4)
enqueue(g)	**?**	

Increase length of array and copy? BAD



Increase length of array. Copy so that head moves to front. (GOOD)



EXTENDING CIRCULAR ARRAY

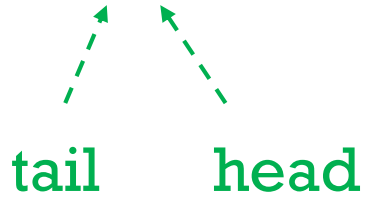
```
enqueue( element ){
    if ( queue.size == queue.length)  {
        //  increase length of array

        create a bigger array tmp[ ]    //  e.g. 2*length
        for i = 0  to queue.length  - 1
              tmp[i] = queue[ (head + i) %  queue.length ]
        head = 0
        queue = tmp
    }
    queue[size] = element
    queue.size = queue.size + 1
}
```

WHAT DO WE DO IF THE QUEUE IS EMPTY?

tail = (head + size – 1) % length

What happens when size == 0 ?

	array	(head, tail, size)
Initial state	----	(0, 3, 0)
enqueue(a)	a---	(0, 0, 1)
enqueue(b)	ab--	(0, 1, 2)
dequeue()	-b--	(1, 1, 1)
dequeue()	----	(2, 1, 0)



STACK AND QUEUES IN JAVA

- Java has a stack class : https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html
  - Push, pop, isEmpty, peek

- Java does NOT have a queue class, but it has a queue **interface**, implemented by different queue classes.
  - You will discuss intefaces on Monday with Giulia.

SCOPE OF THE COURSE

- We will not discuss Java packages/classes for ADTs
  - We will never ask you to write a program using the Stack class in Java.
  - We can ask you to write your own Stack class and manipulate it.
    - This is what we did with Homework 2!

- The objective of the course is to understand how ADTs work and why we use them.

QUIZ 2 TODAY!

- You have until 8PM today to submit the quiz
- **Monday- Wednesday:** two last classes about Java (with Giulia)
- **Next Friday**: new topic! We start discussing induction and recursion.