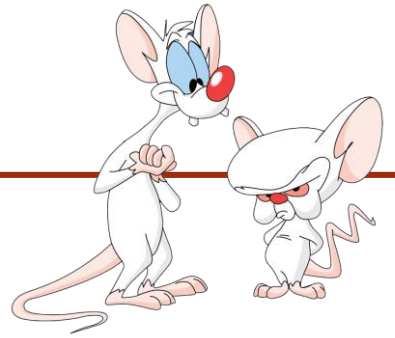
COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 7-2 :

Giulia Alberini, Fall 2020
Slides adapted from Michael Langer's

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Queues

ADT (ABSTRACT DATA TYPE)

- List

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

- Stack

  push(e) , pop(), ..

- Queue

  enqueue(e), dequeue()

QUEUE

dequeue
(remove from front)

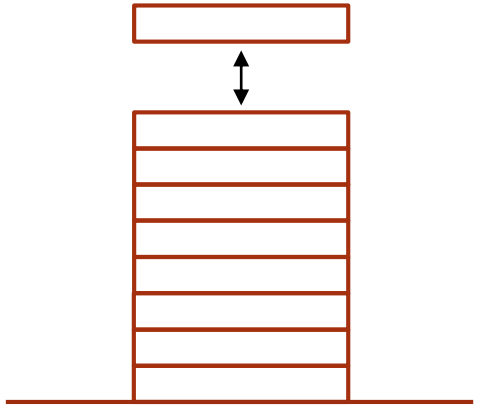
enqueue
(add at back)

e.g. Server

clients

EXAMPLES

- keyboard buffer
- CPU processes (app rallel)
- web server
- students in the zoom waiting room
- …

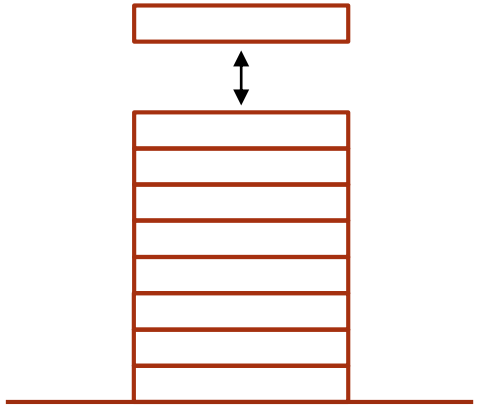


Stack

push(e)

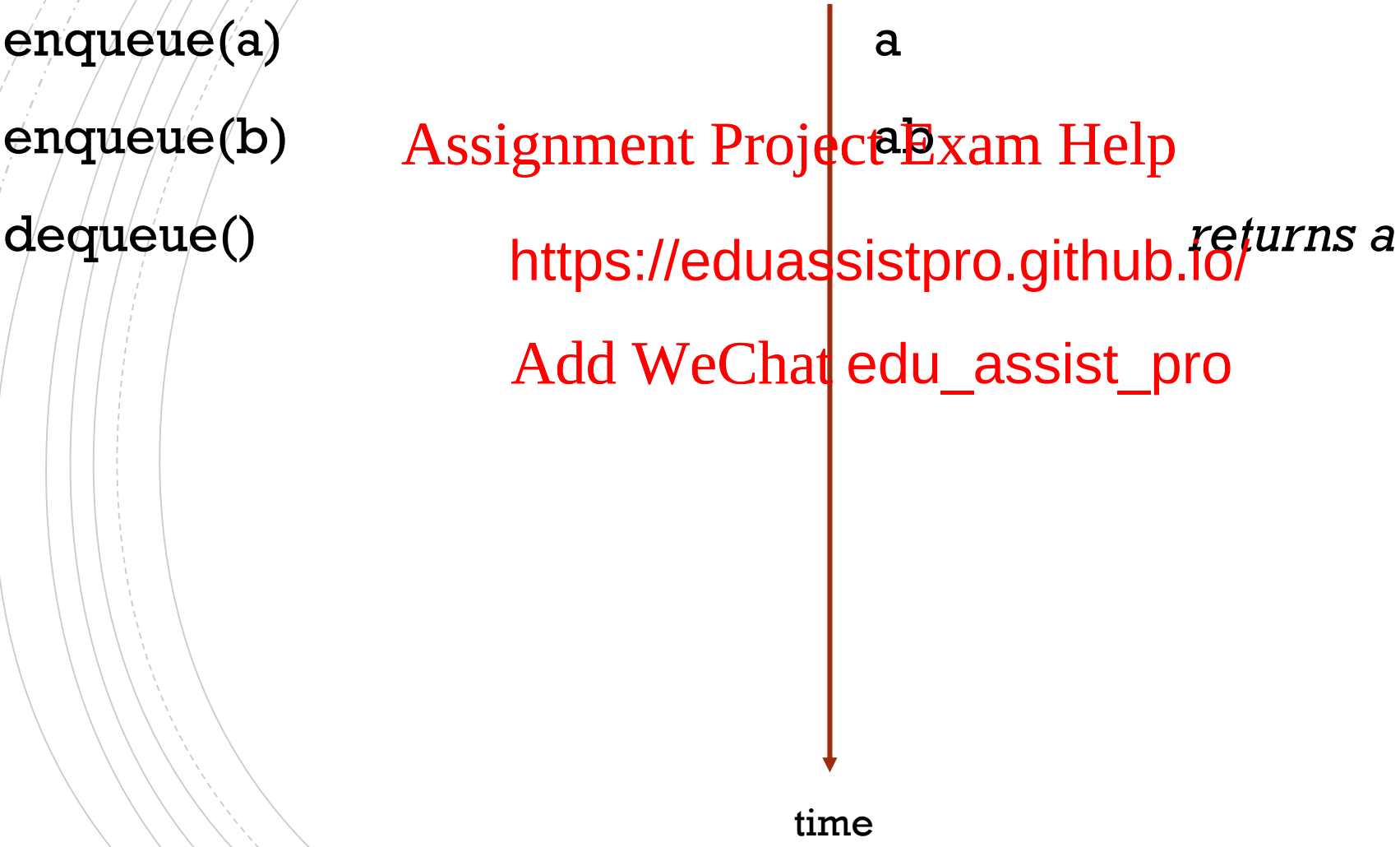
pop()

LIFO
(last in, first out)

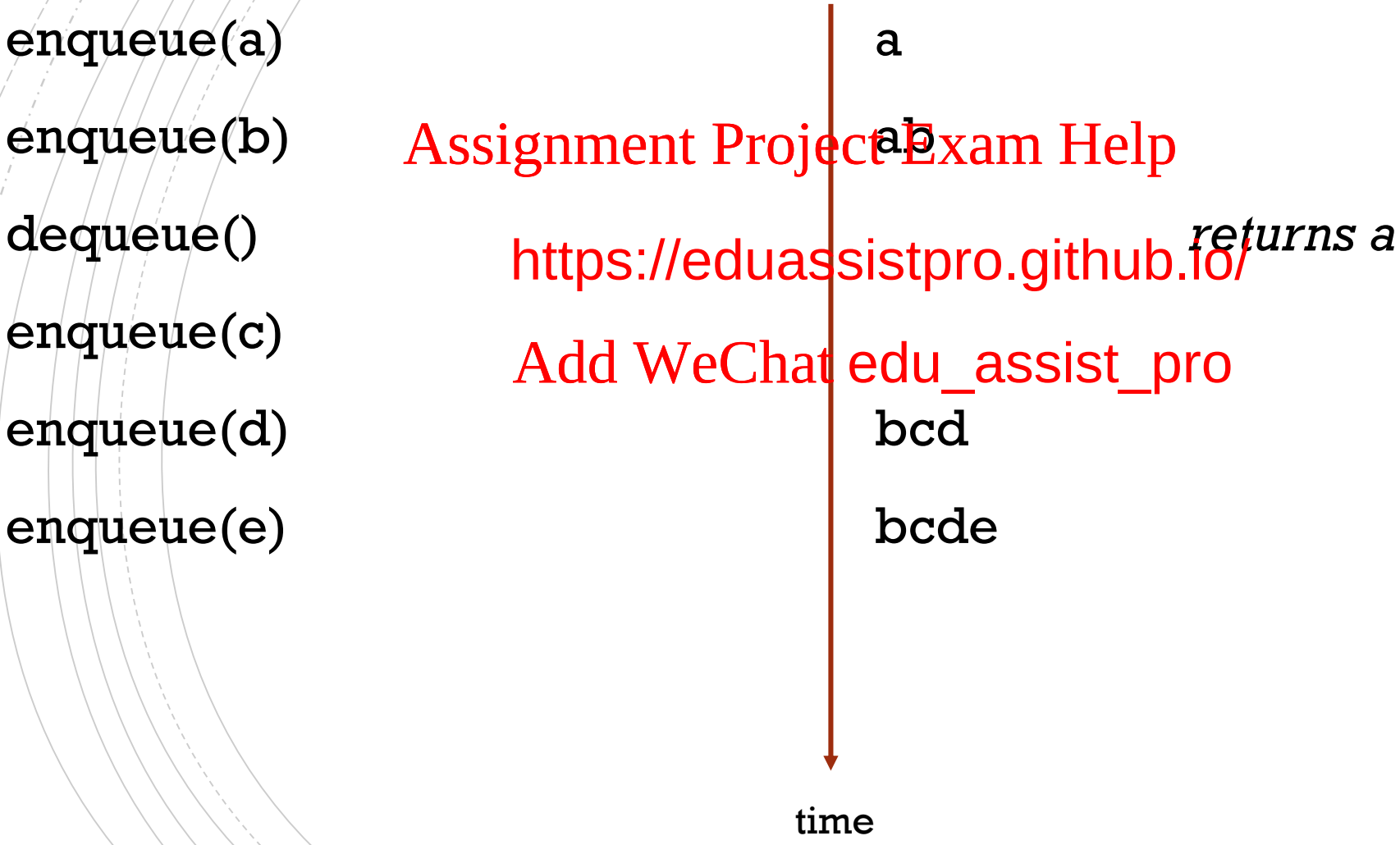


Stack	Queue
push(e)	ue(e)
pop()	dequeue()
LIFO (last in, first out)	FIFO (first in, first out) "first come, first serve"

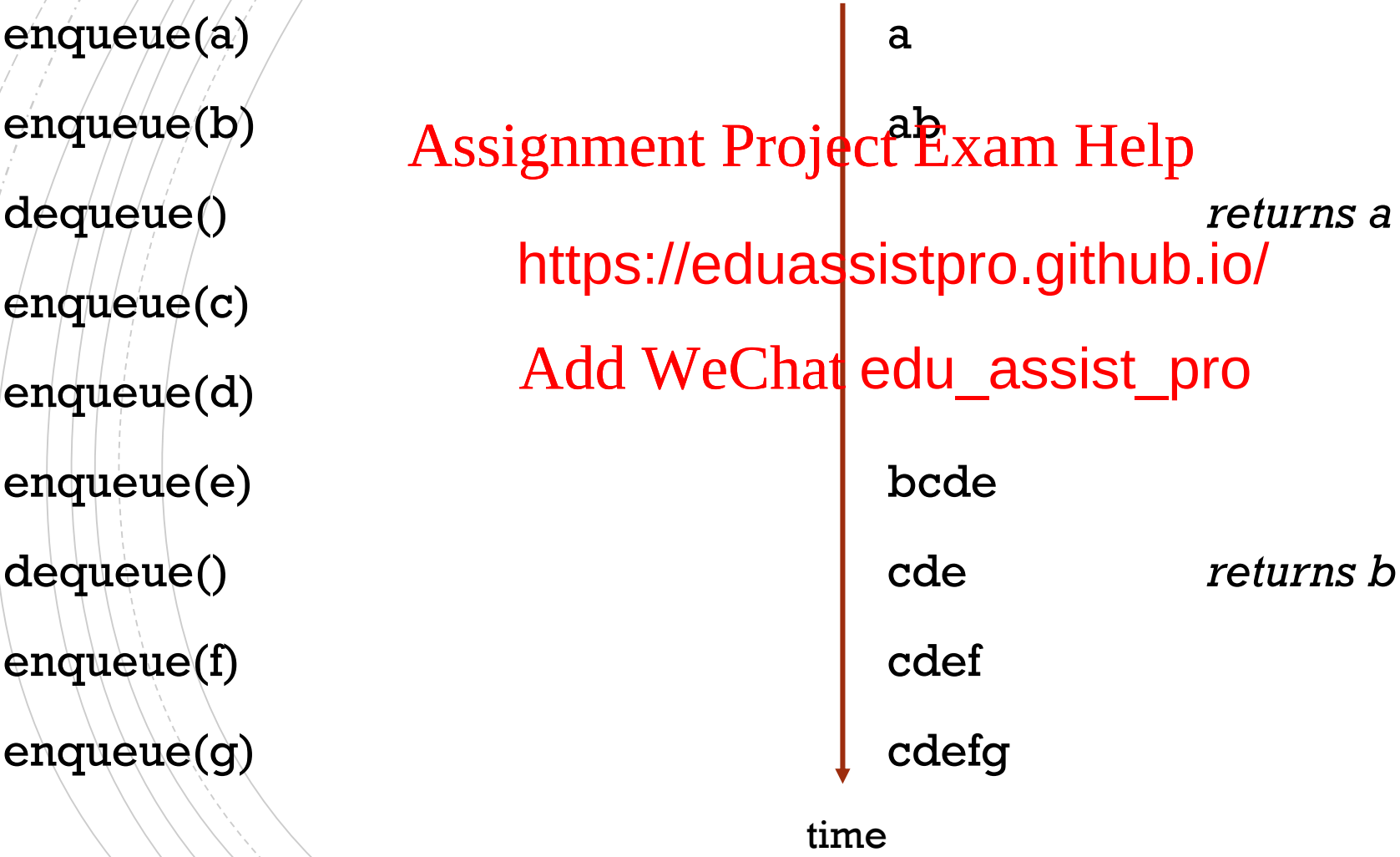
QUEUE EXAMPLE



QUEUE EXAMPLE



QUEUE EXAMPLE



HOW TO IMPLEMENT A QUEUE?

enqueue(e) dequeue()

singly linked li

doubly linked list

array list

HOW TO IMPLEMENT A QUEUE?

enqueue(e) dequeue()

singly linked li removeFirst()

doubly linked list

array list

HOW TO IMPLEMENT A QUEUE?

enqueue(e) dequeue()

singly linked li removeFirst()

doubly linked list (same, or & removeLast())

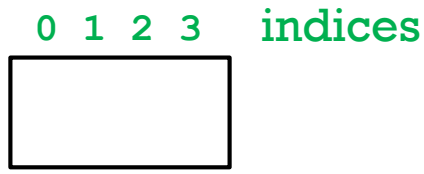
array list

HOW TO IMPLEMENT A QUEUE?

	enqueue(e)	dequeue()
singly linked li		removeFirst()
doubly linked list	(same, or	& removeLast())
array list	addLast(e)	**removeFirst()**

SLOW!

IMPLEMENTING A QUEUE WITH AN ARRAY LIST (BAD)

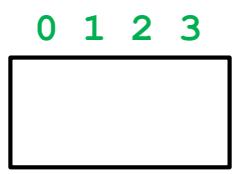


length = 4

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

a _ _ _

IMPLEMENTING A QUEUE WITH AN ARRAY LIST (BAD)



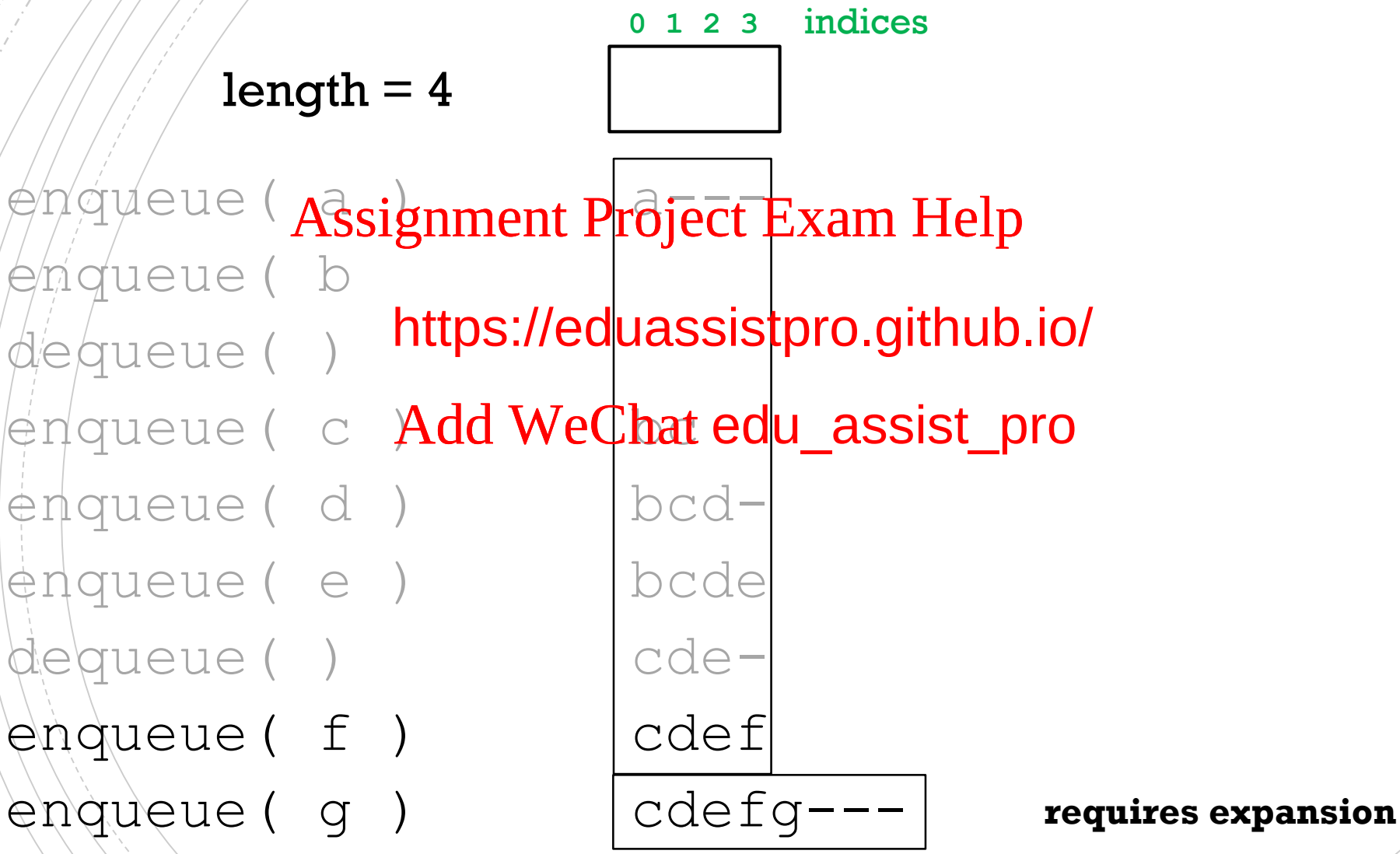
length = 4

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

Requires shift

Requires shift

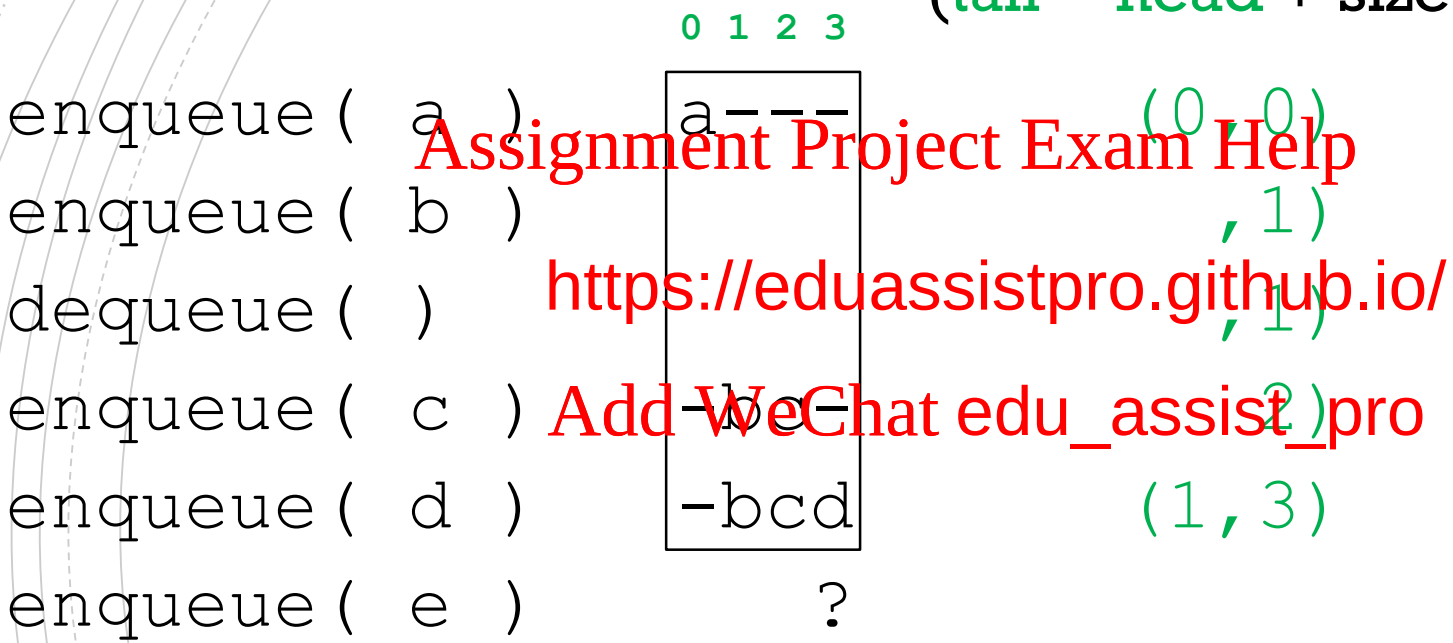
IMPLEMENTING A QUEUE WITH AN ARRAY LIST (BAD)



requires expansion

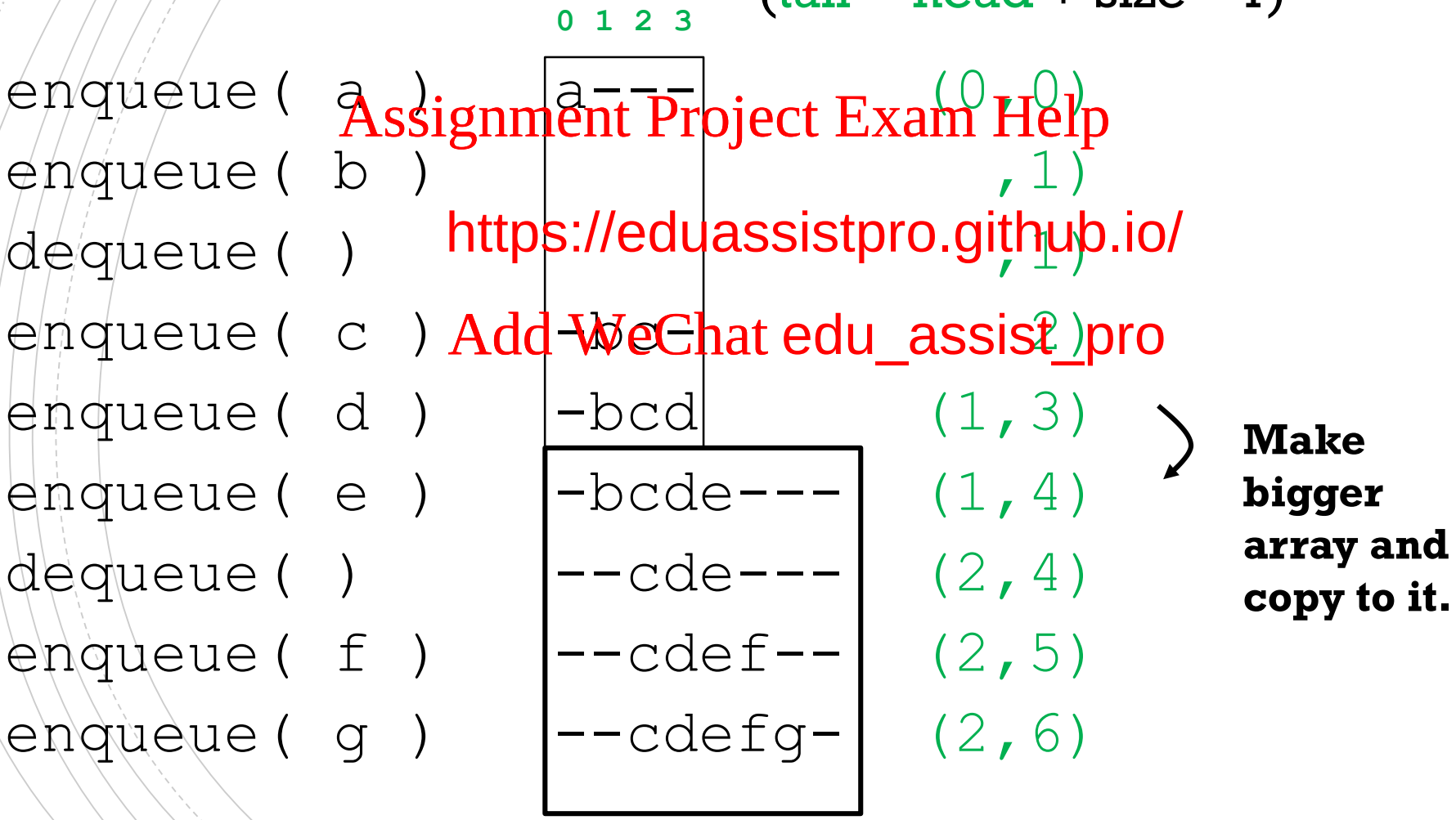
IMPLEMENTING A QUEUE WITH AN EXPANDING ARRAY (ALSO BAD)

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



IMPLEMENTING A QUEUE WITH AN EXPANDING ARRAY (ALSO BAD)

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



IMPLEMENTING A QUEUE WITH AN EXPANDING ARRAY

- to dequeue: retrieve the element at head and increase the index head

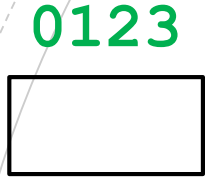
- to enqueue: add the element at tai

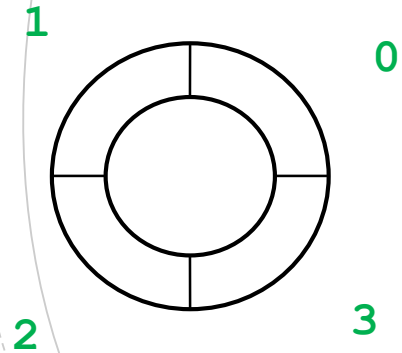
An expanding array is an inefficient usage of space!

A better idea is…

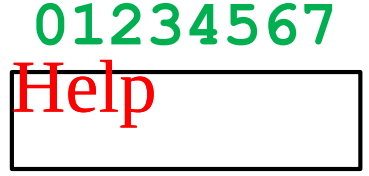
CIRCULAR ARRAY

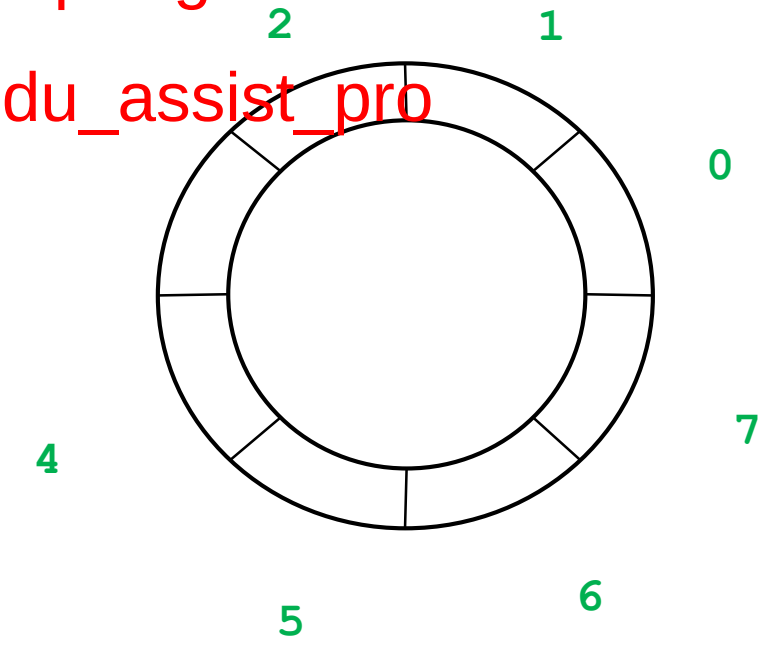
length = 4





length = 8





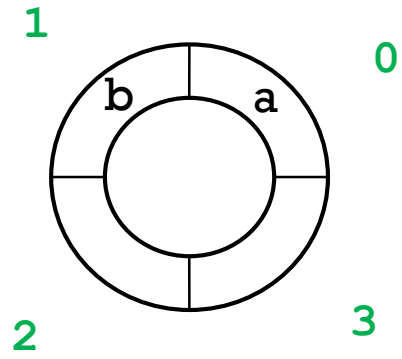
IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

```
enqueue( a )
enqueue( b )
dequeue() ?
```

tail = (head + size – 1) mod length

tail

head



head=0 tail=1

IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

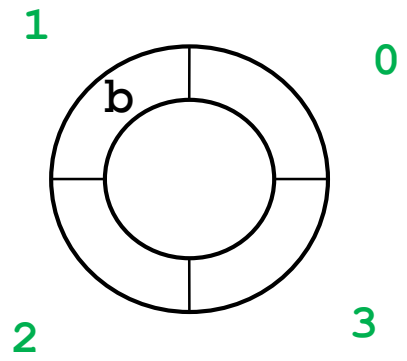
tail = (head + size – 1) mod length

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


head

tail

head=1 tail=1

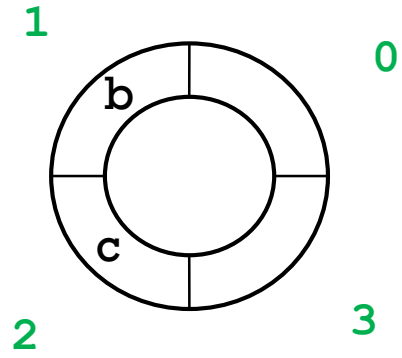


IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

tail = (head + size – 1) mod length

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


head=1 tail=2

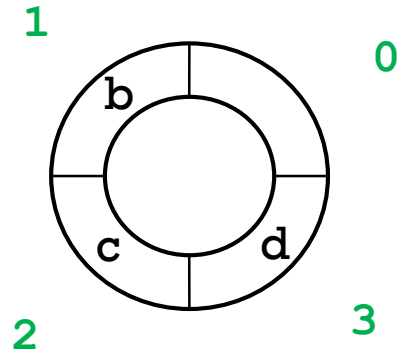


IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

tail = (head + size – 1) mod length

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


head



head=1 tail=3

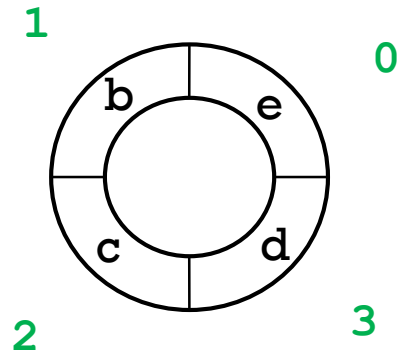
tail

IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

tail = (head + size – 1) mod length

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


head=1 tail=0

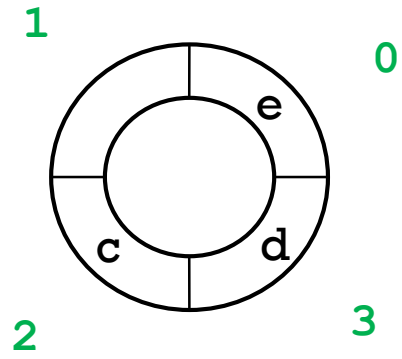


IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

tail = (head + size – 1) mod length

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


head=2 tail=0

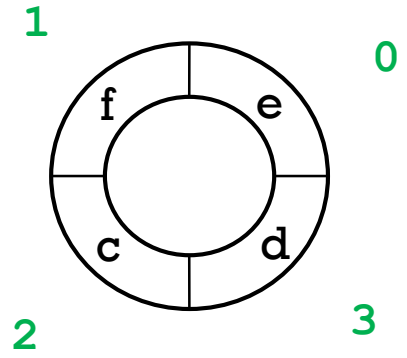


IMPLEMENTING A QUEUE WITH A CIRCULAR ARRAY (GOOD)

tail = (head + size – 1) mod length

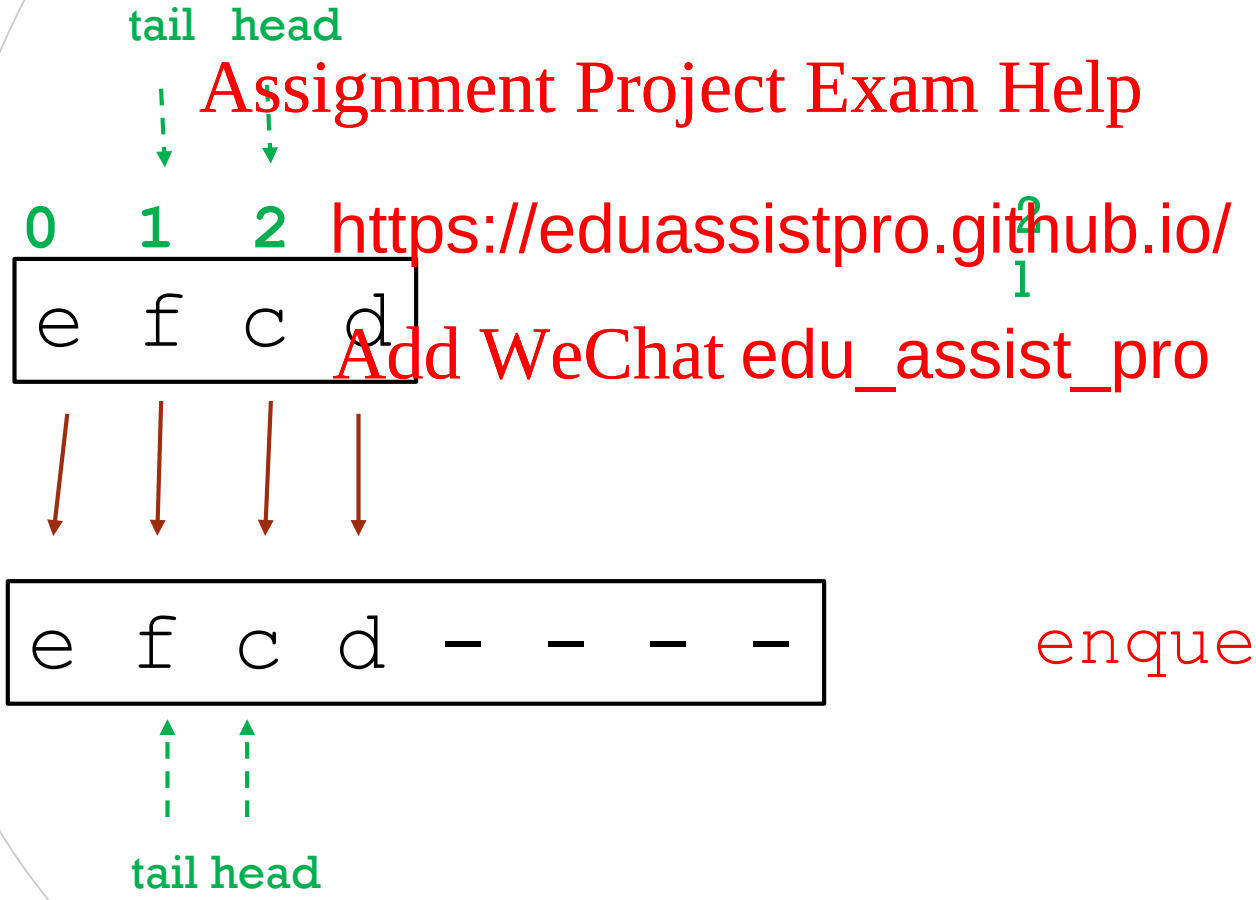
```
enqueue( a )
enqueue( b )
dequeue()
enqueue( c )
enqueue( d )
enqueue( e )
dequeue()
enqueue( f )
enqueue( g ) ?
```


head=2 tail=1



INCREASE THE LENGTH OF THE ARRAY AND COPY

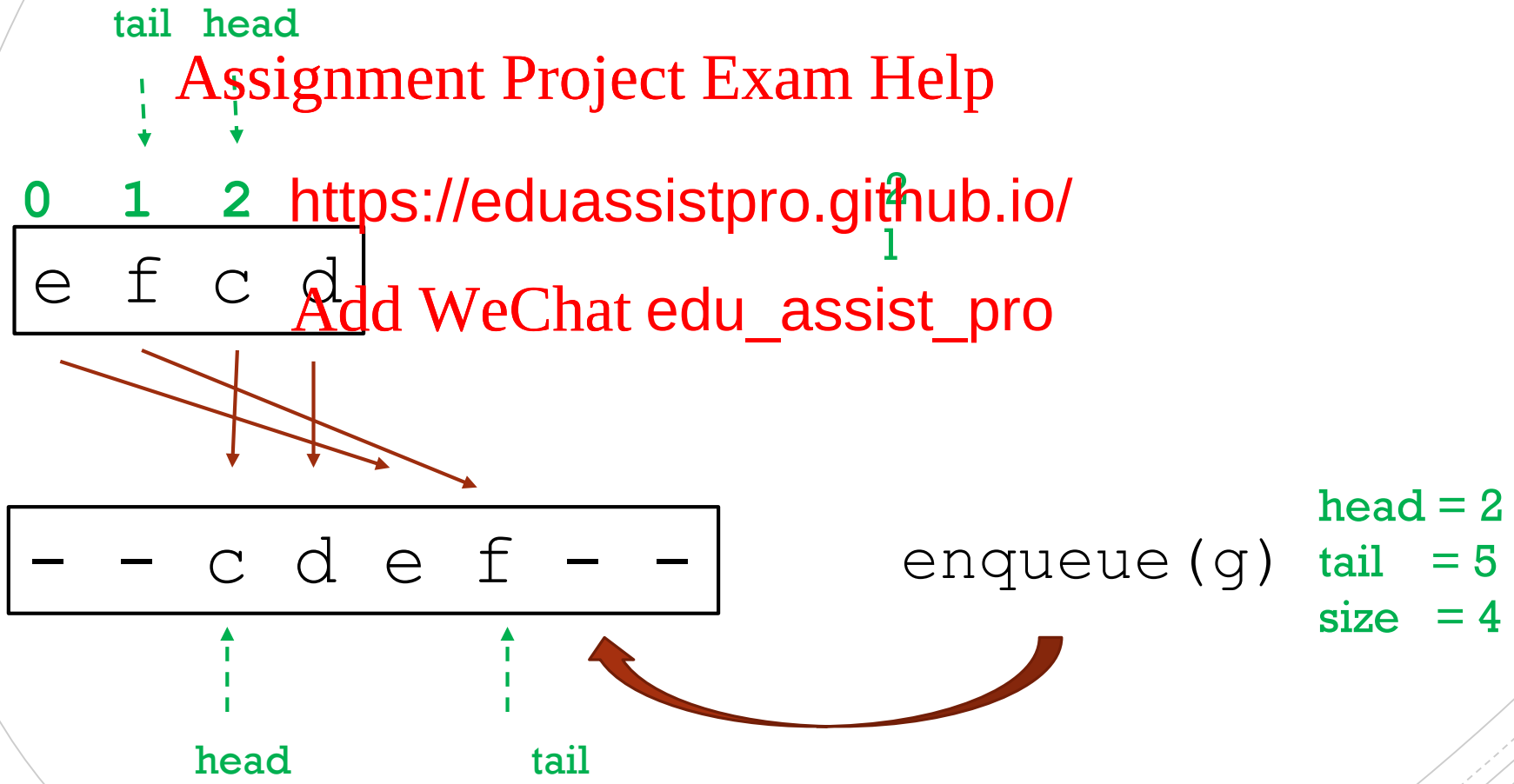
Be careful, on how you copy! The following would not work:



`enqueue(g) ?`

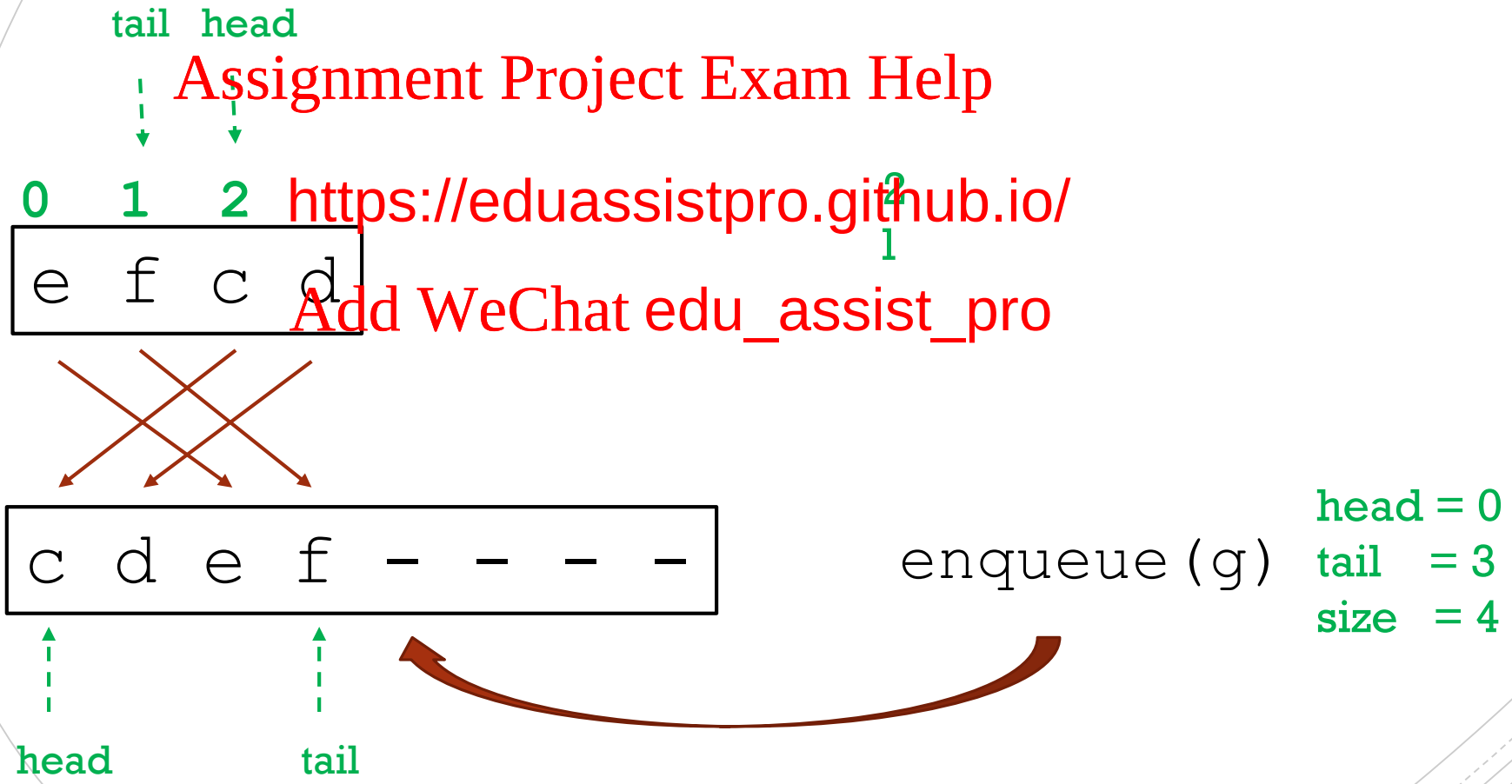
INCREASE THE LENGTH OF THE ARRAY AND COPY

Instead you can copy so that the head remains in the same position.



INCREASE THE LENGTH OF THE ARRAY AND COPY

OR you can copy so that the head moves to position 0.



ENQUEUE(e)

```
enqueue( element ){
   if (size == queue.length)  {
      //  increase length of array
      create a bigg                          e.g. 2*length
      for i = 0  to
         tmp[i] = queue[(head +              ue.length]
      head = 0
      queue = tmp
   }
   queue[(head + size) mod length] = element
   size++
}
```


Note that we don't have a tail variable here. Instead, recall that *tail = (head + size -1) mod length*, and note that the new element is added in position *(tail + 1) mod length*.

DEQUEUE()

```
dequeue() {
    if size <= 0
        ra
    element = queue[he
    size = size - 1
    head = (head + 1) mod length
    return element
}
```

WHAT IF SIZE IS 0?

What is the relation between head and tail when size is equal to 0?

tail = (head + size – 1) mod length

Initial state: add(tail, 3, 0)

WHAT IF SIZE IS 0?

What is the relation between head and tail when size is equal to 0?

tail = (head + size – 1) % length

Initial state	___	(3, 0)
enqueue(a)	a---	(0, 1)
enqueue(b)	ab--	(0, 1, 2)
dequeue()	-b--	(1, 1, 1)
dequeue()	----	(2, 1, 0)

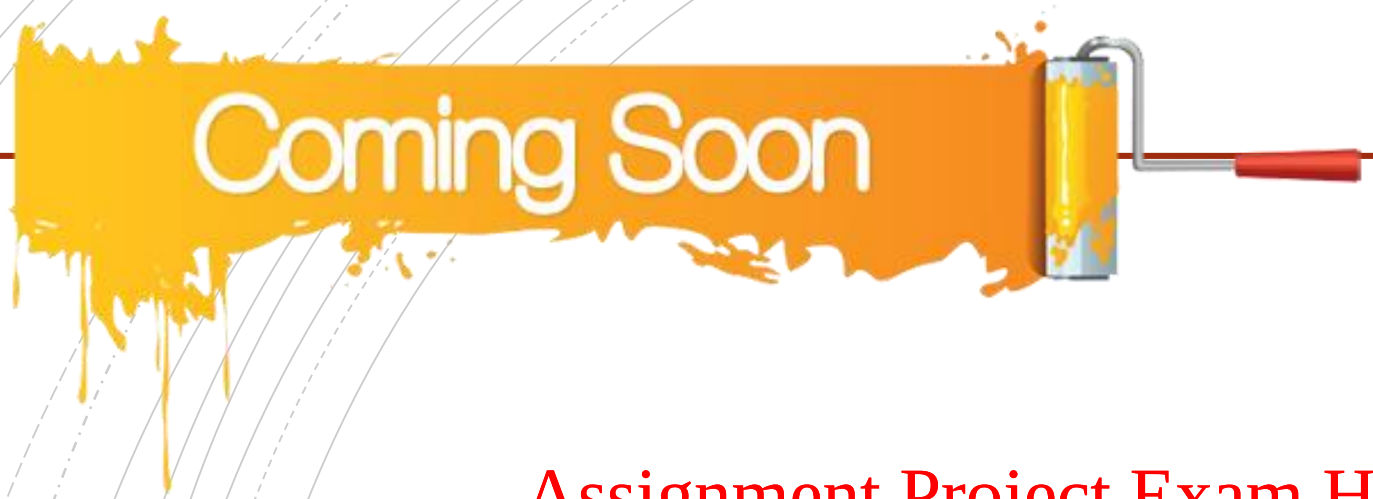
tail head

ADT (ABSTRACT DATA TYPE)

Defines a data type by the values and operations from the user's perspective only. It ignores the details of the implementation.

Examples:

- list
- stack
- queue
- …



In the next

- Back to J