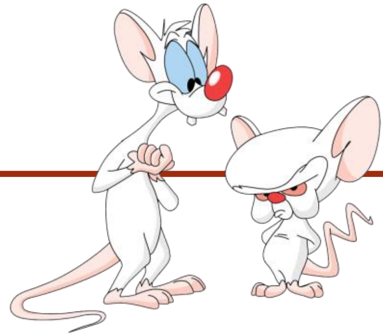
COMP 250

INTRODUC TER SCIENCE

Week 5-2 : Singly

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Singly Linked Lists

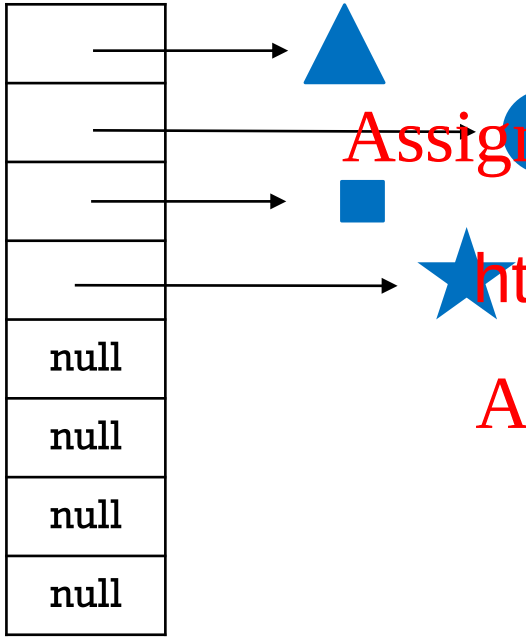
IMPLEMENTATIONS

There are different implementations of a list:

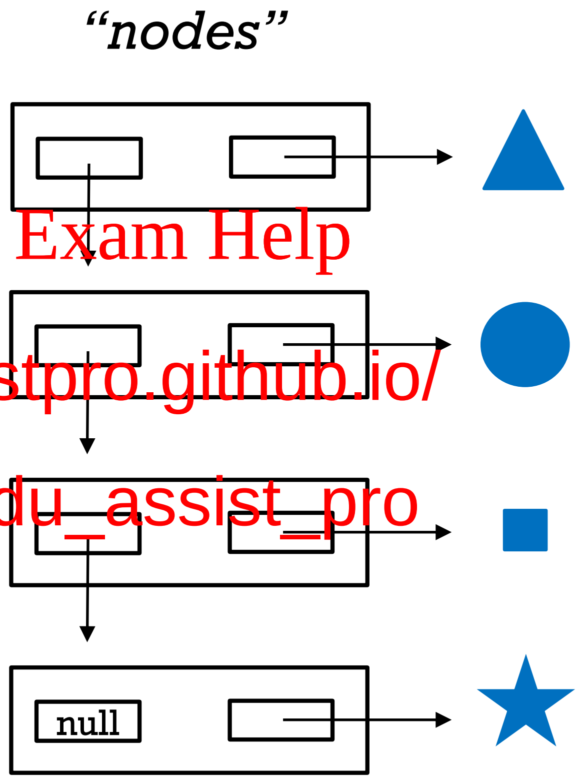
- Array list
- Singly linked list
- Doubly linked list

n the list are linked using poi

Array list

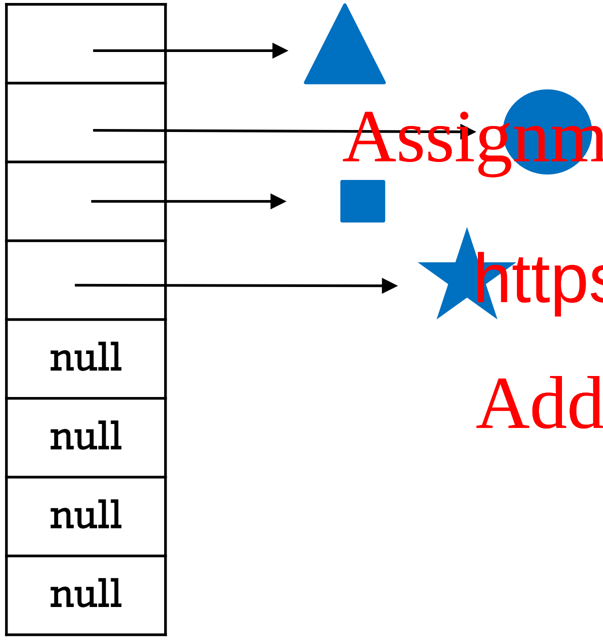


Linked list



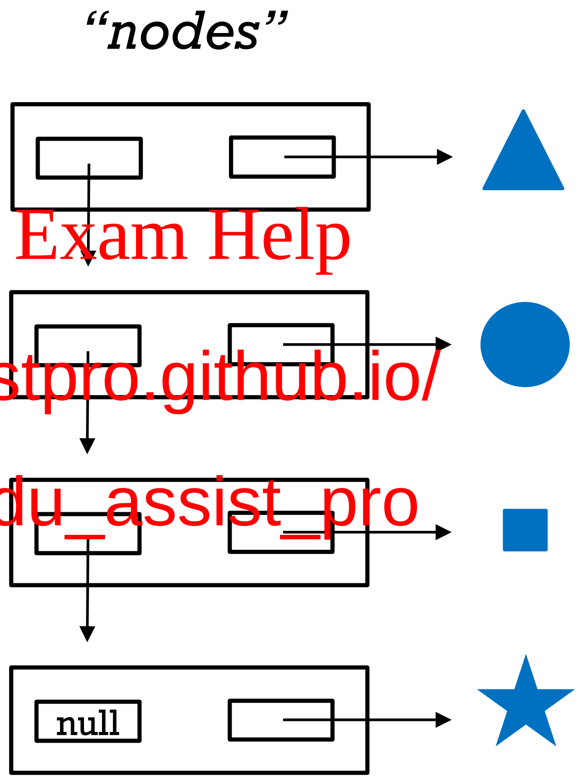
size = 4

Array list



Array slots are in consecutive locations (addresses) in memory, but objects (elements) can be anywhere.

Linked list

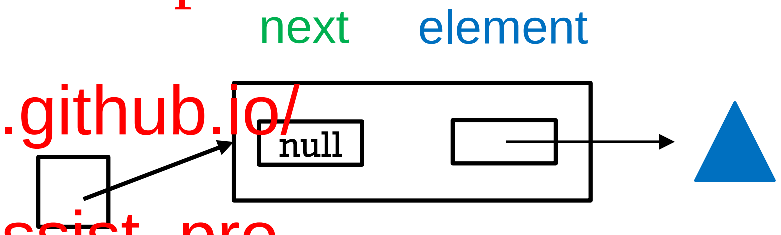


Linked list "nodes" and objects (elements) can be anywhere in memory.

SINGLY LINKED LIST NODE

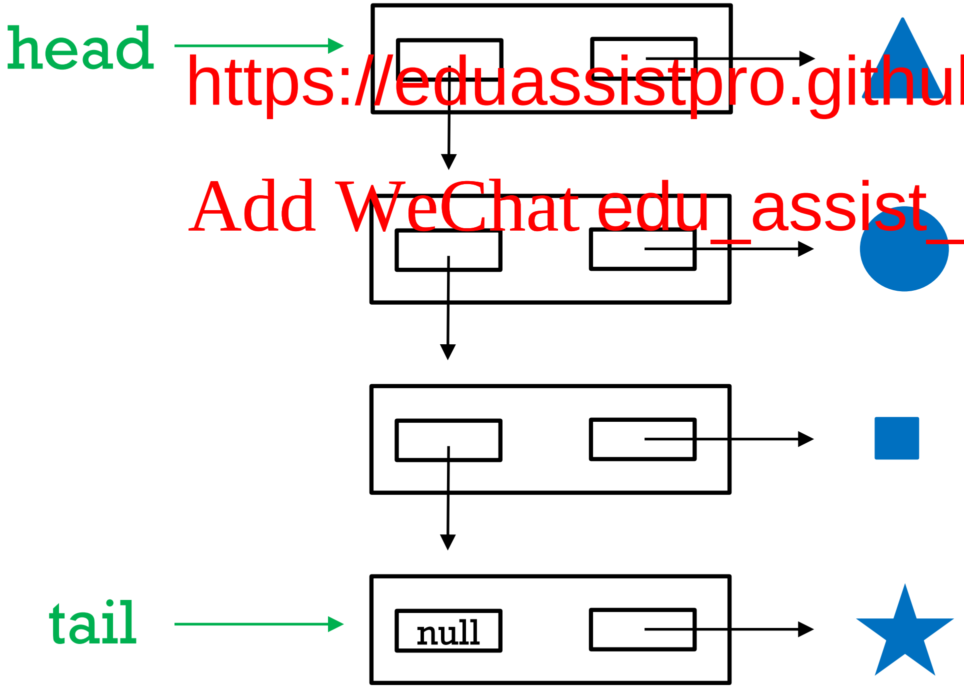
```
class SNode {
   Shape element;
   SNode next;
}
```

```
SNode myNode = new SNode();
n.element = new Shape(▲);
```



SINGLY LINKED LIST

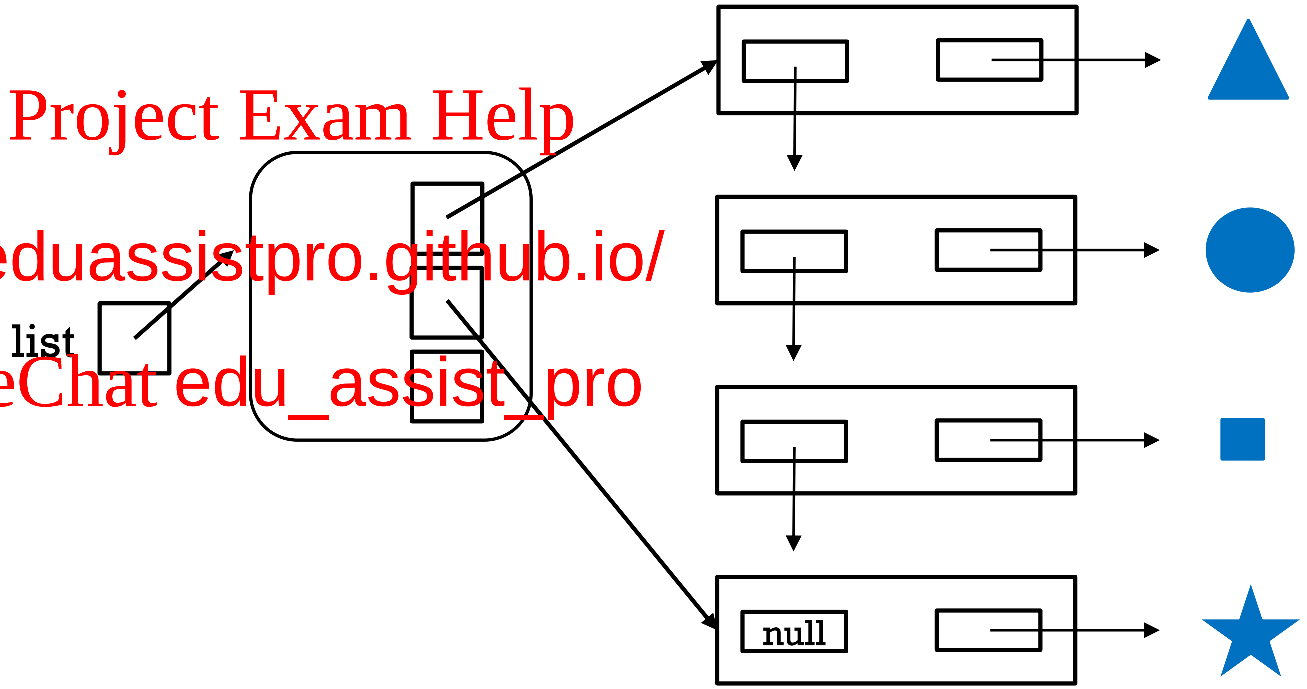
We think of a linked list as a sequence of nodes, along with a reference to the first (head) and last (tail) node.



SINGLY LINKED LIST

```
public class SLinkedList {
    private SNode head;
    private SNode tail;
    private int size;
    :
    private class SNode {
        Shape element;
        SNode next;
    }
}
```

```
SLinkedList list = new SLinkedList();
:
```



LINKED LIST OPERATIONS

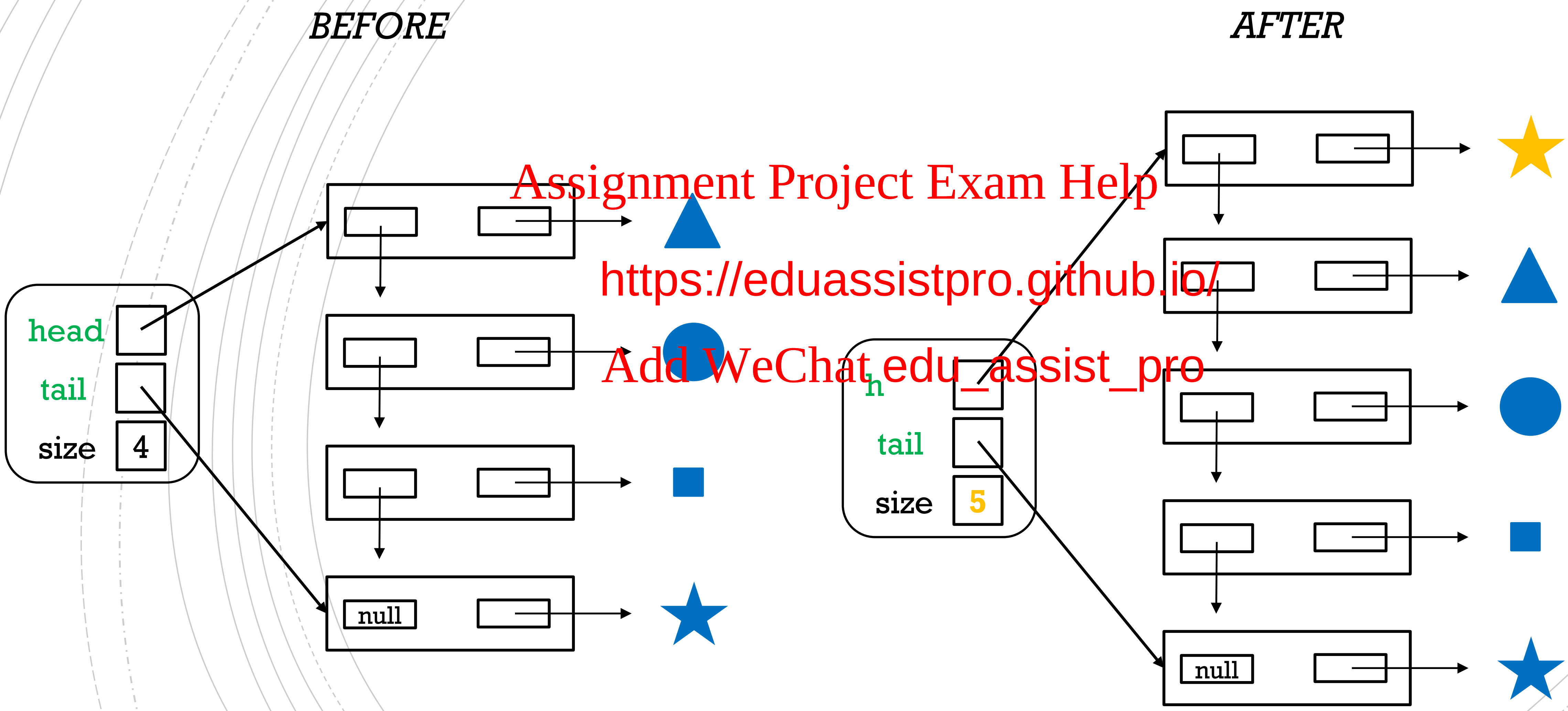
- addFirst (e)

- removeFirst( )

- addLast (e)

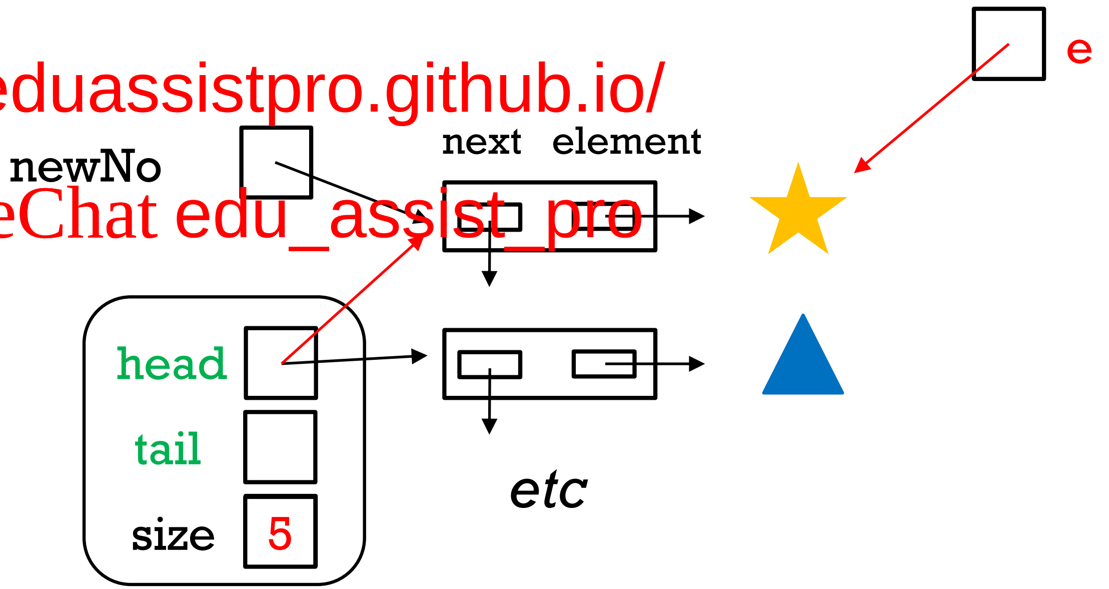
- removeLast( )
- ……. many other list operations

LINKED LIST OPERATIONS – addFirst(★)

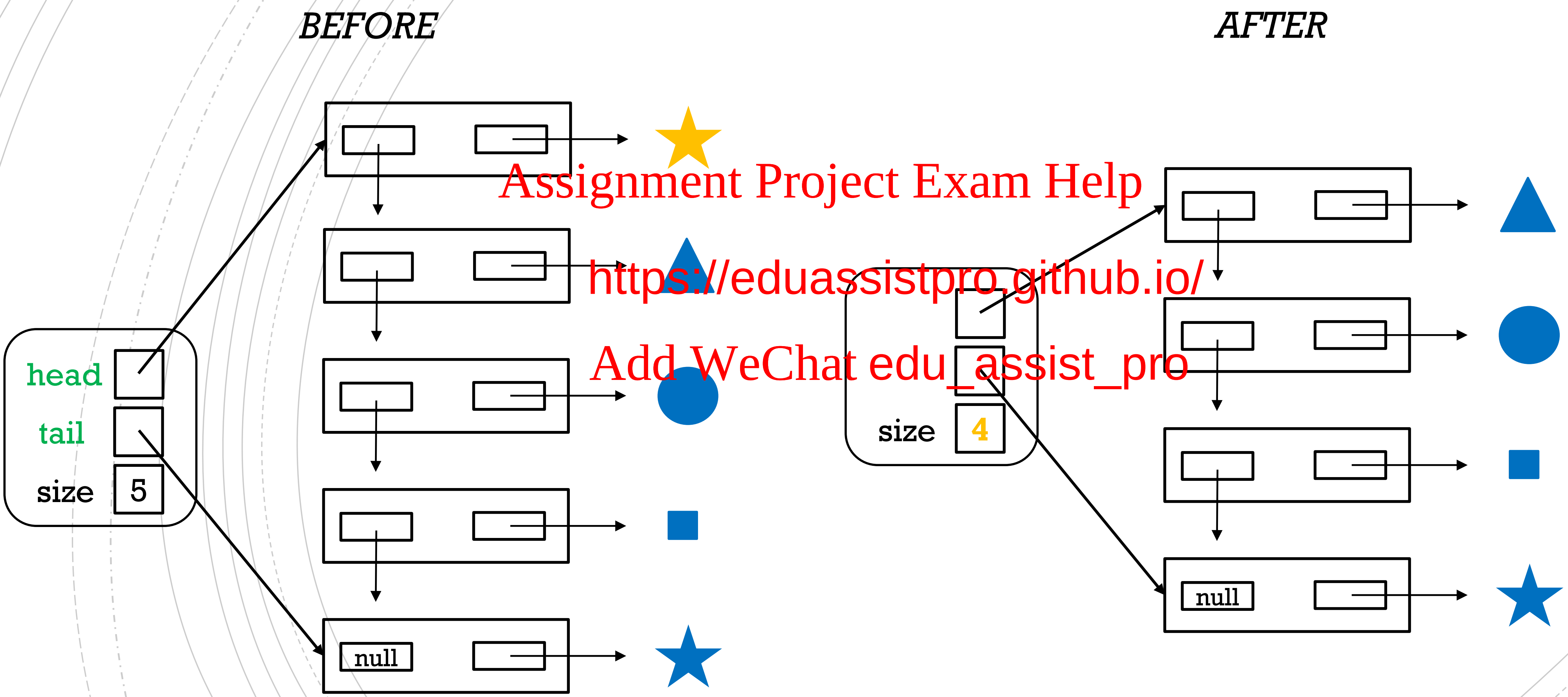


addFirst(e) – PSEUDOCODE

```
SNode newNode = new SNode();
newNode.element = e;
newNode.next = head;
// edge case
if (head == null)
      tail = newNode;
head = newNode;
size = size +1;
```

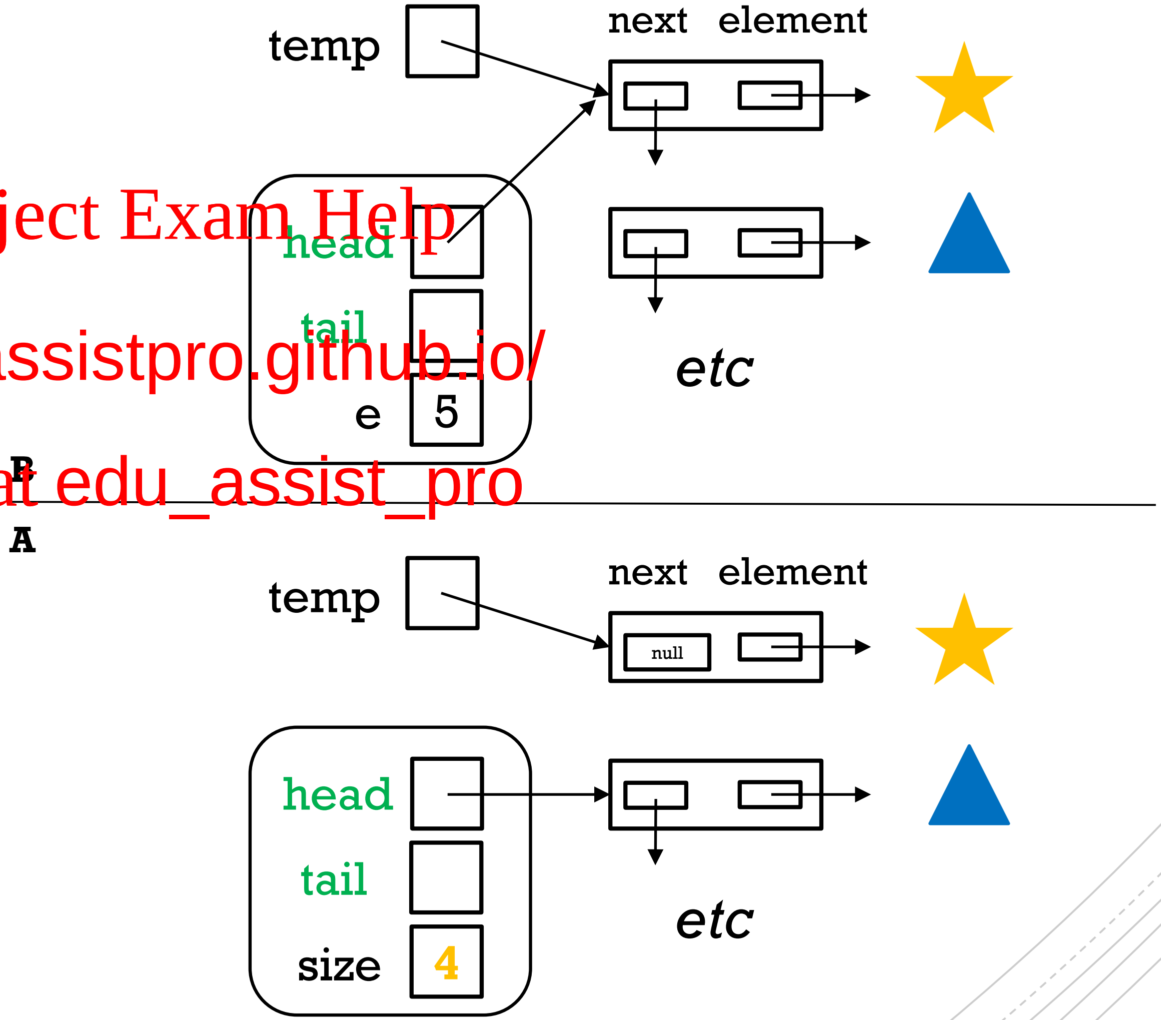


LINKED LIST OPERATIONS – removeFirst( )



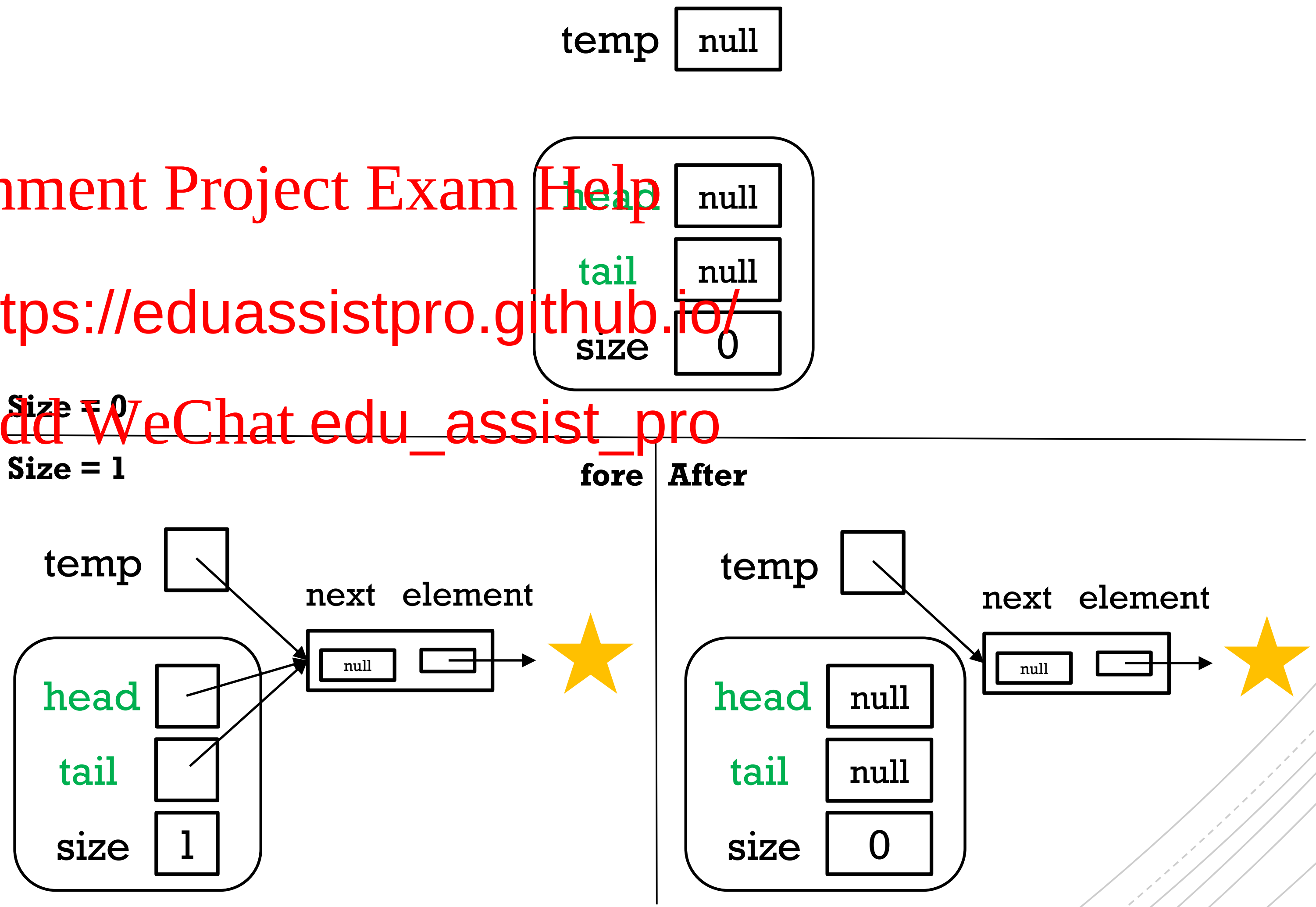
removeFirst() – PSEUDOCODE

```
SNode temp = head;

head = temp.next;

temp.next = null; // not required

size = size - 1;

return temp.element;
```



removeFirst() – EDGE CASES (SIZE IS 0 OR 1)

```
SNode temp = head;
if (size == 0)
      throw exception
head = temp.next;
temp.next = null;
size = size - 1;
if (size == 0)
      tail = null;
return temp.element;
```



WORSE CASE TIME COMPLEXITY (N = LIST SIZE)

	array list	linked list
addFirst()		)
removeFirst()	O(N)	

For arraylist with N elements, recall that add(0, e) and remove(0) required a loop with N iterations

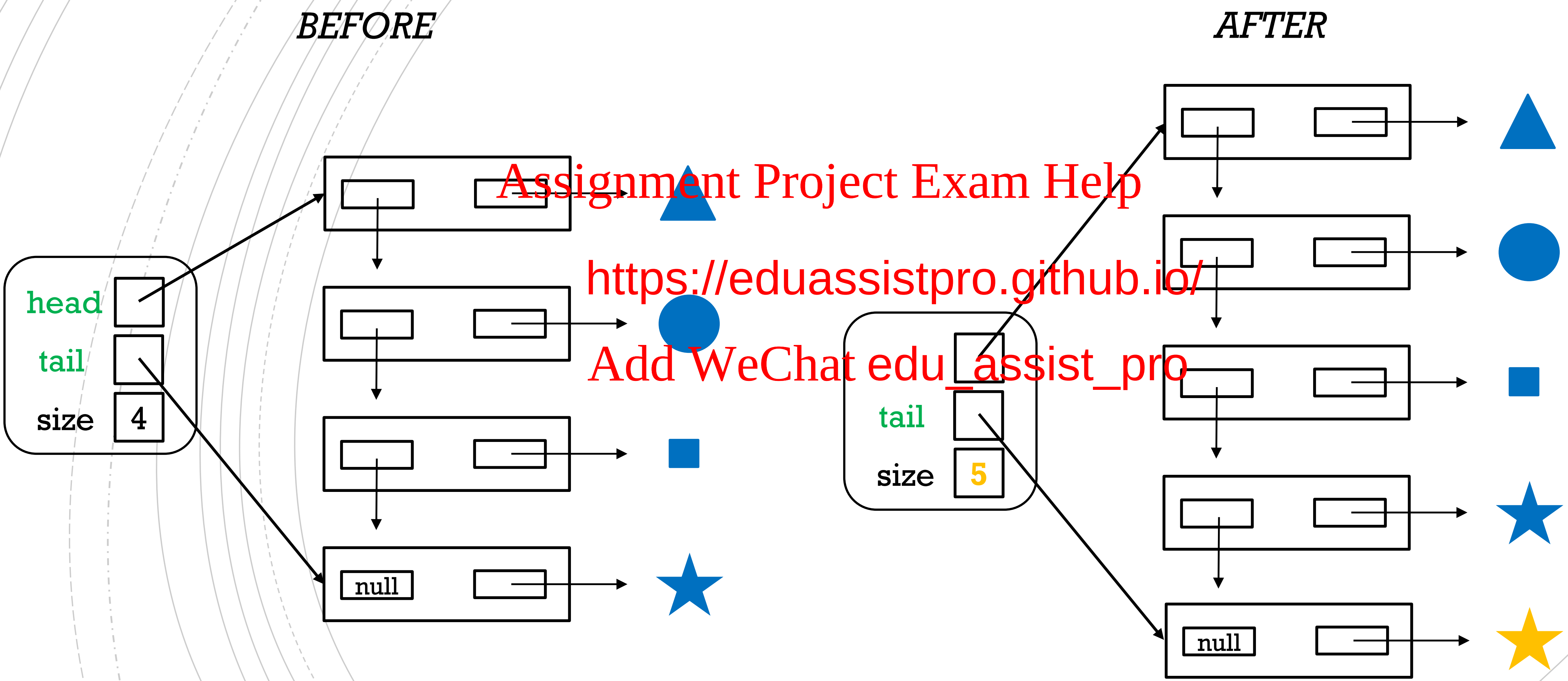
For linked lists the implementation of addFirst() and removeFirst() does not depend on the number of elements in the list

WORSE CASE TIME COMPLEXITY (N = LIST SIZE)

	array list	linked list
addFirst()		O(1)
removeFirst()	O(N	O(1)
addLast()	O(1)*	?
removeLast()	O(1)	?

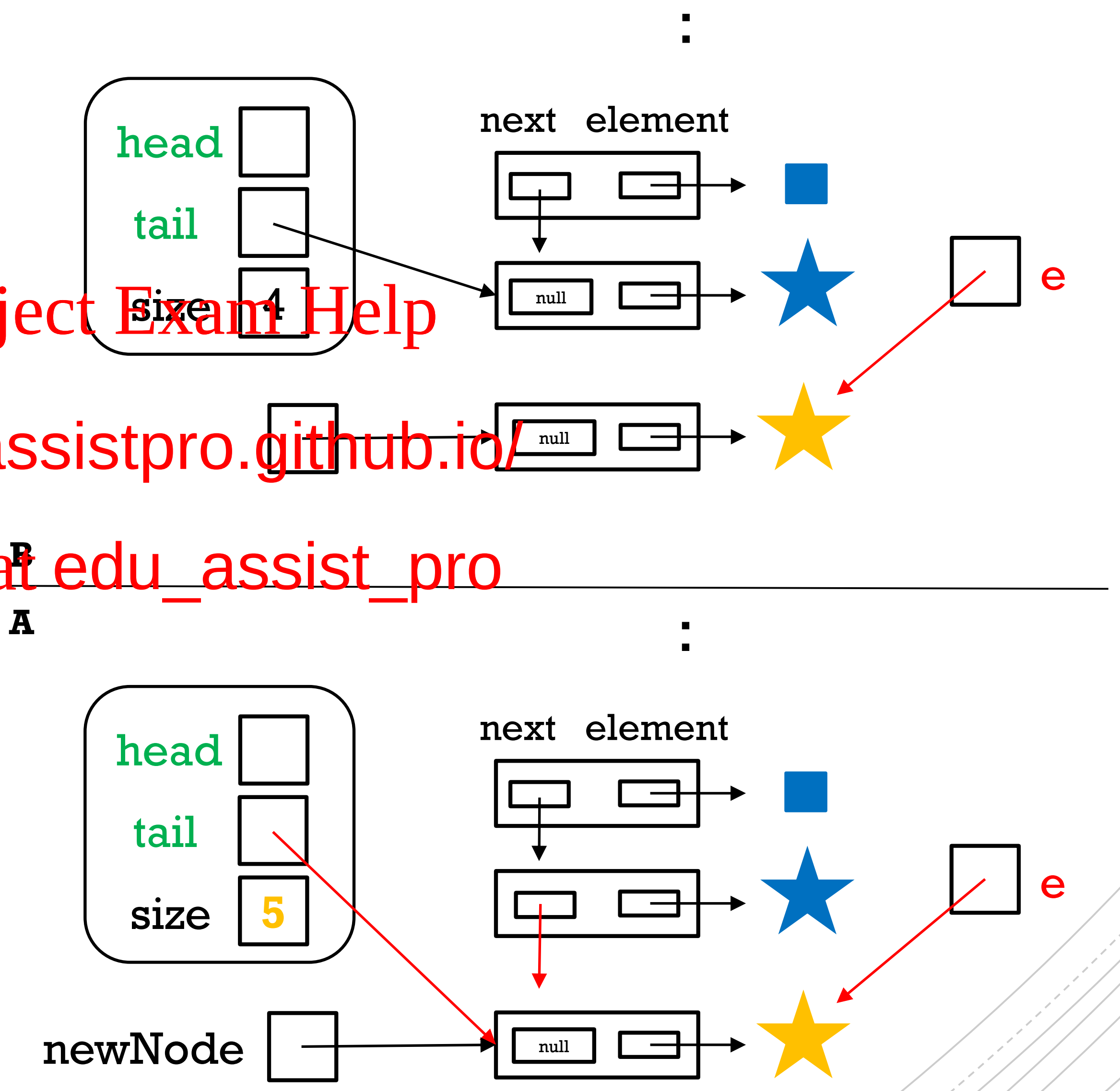
*if array is not full

LINKED LIST OPERATIONS – addLast(★)

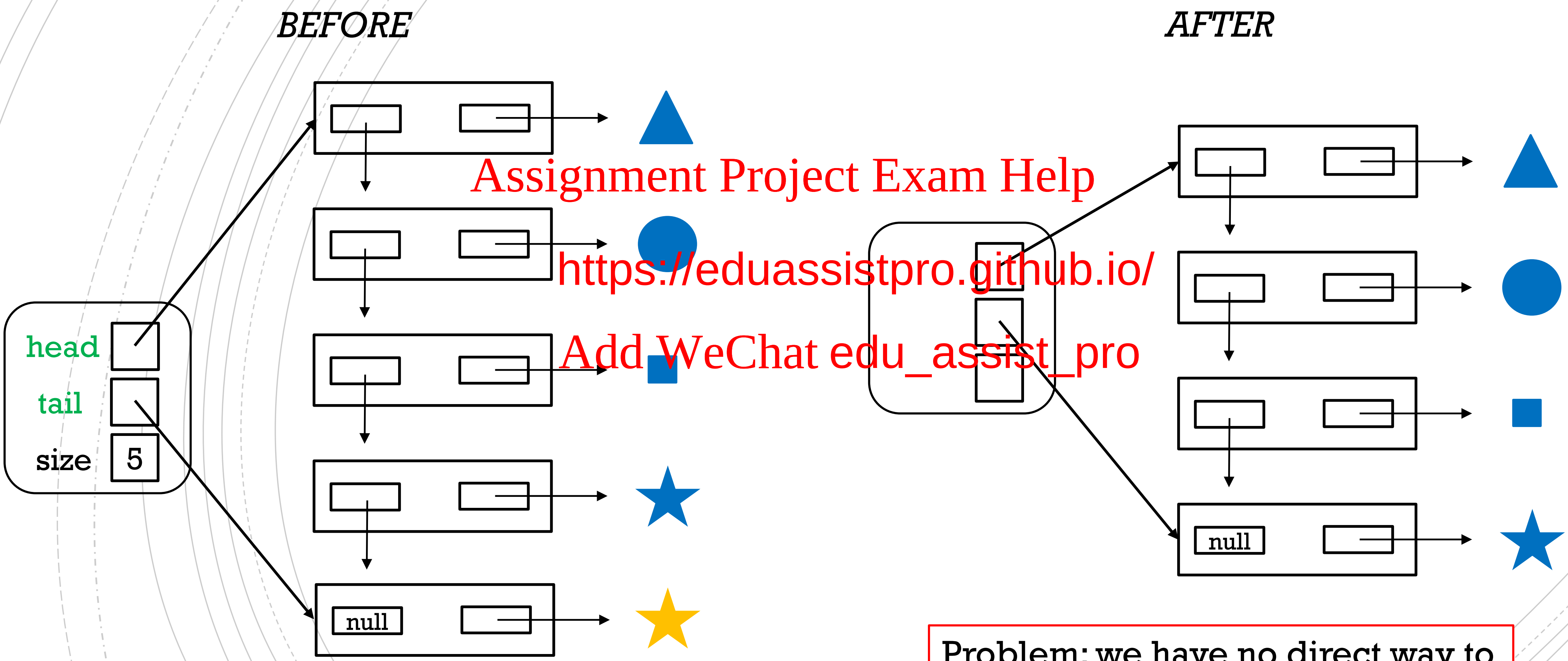


addLast(e) – PSEUDOCODE

```
// create a new node
SNode newNode = new SNode();
newNode.element = e;


// add it at the end
tail.next = newNode;
tail =  tail.next;
size = size + 1;
```



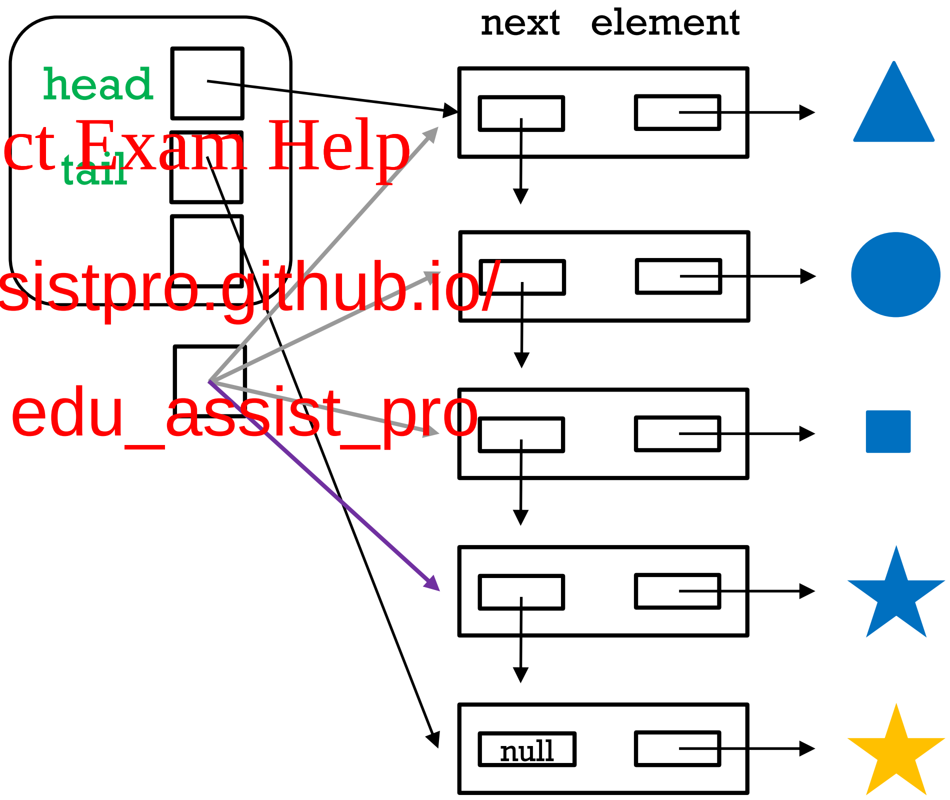
LINKED LIST OPERATIONS – removeLast( )



Problem: we have no direct way to access the node before tail!

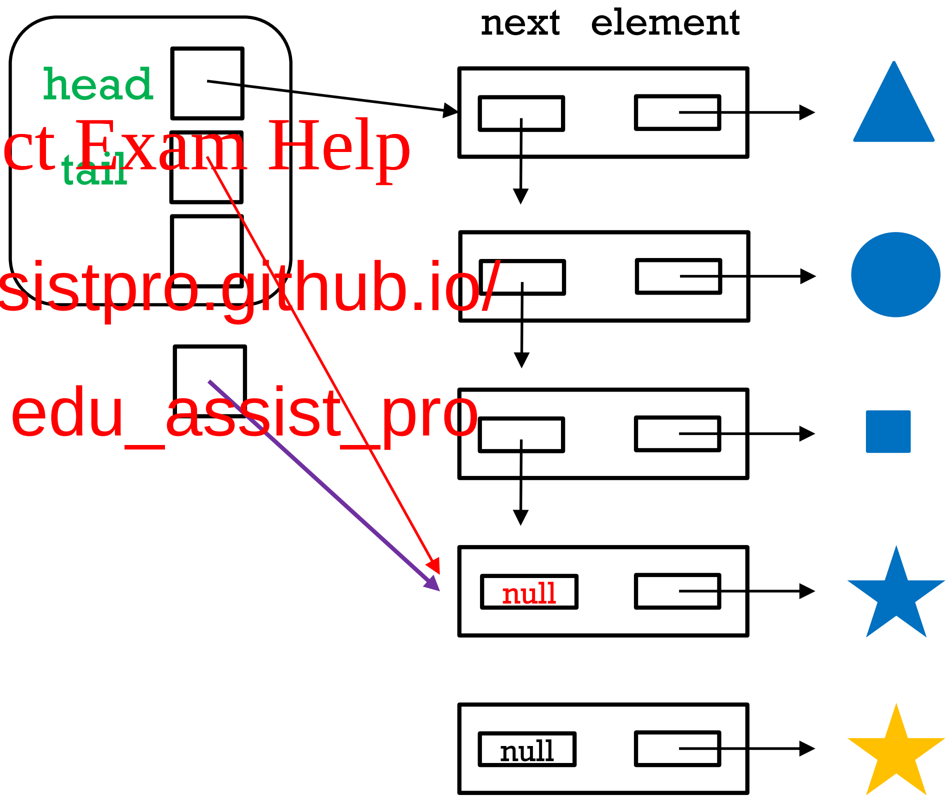
removeLast() – PSEUDOCODE

```
SNode tmp = head;
while (tmp.next != tail)
        tmp = tmp.next;
```



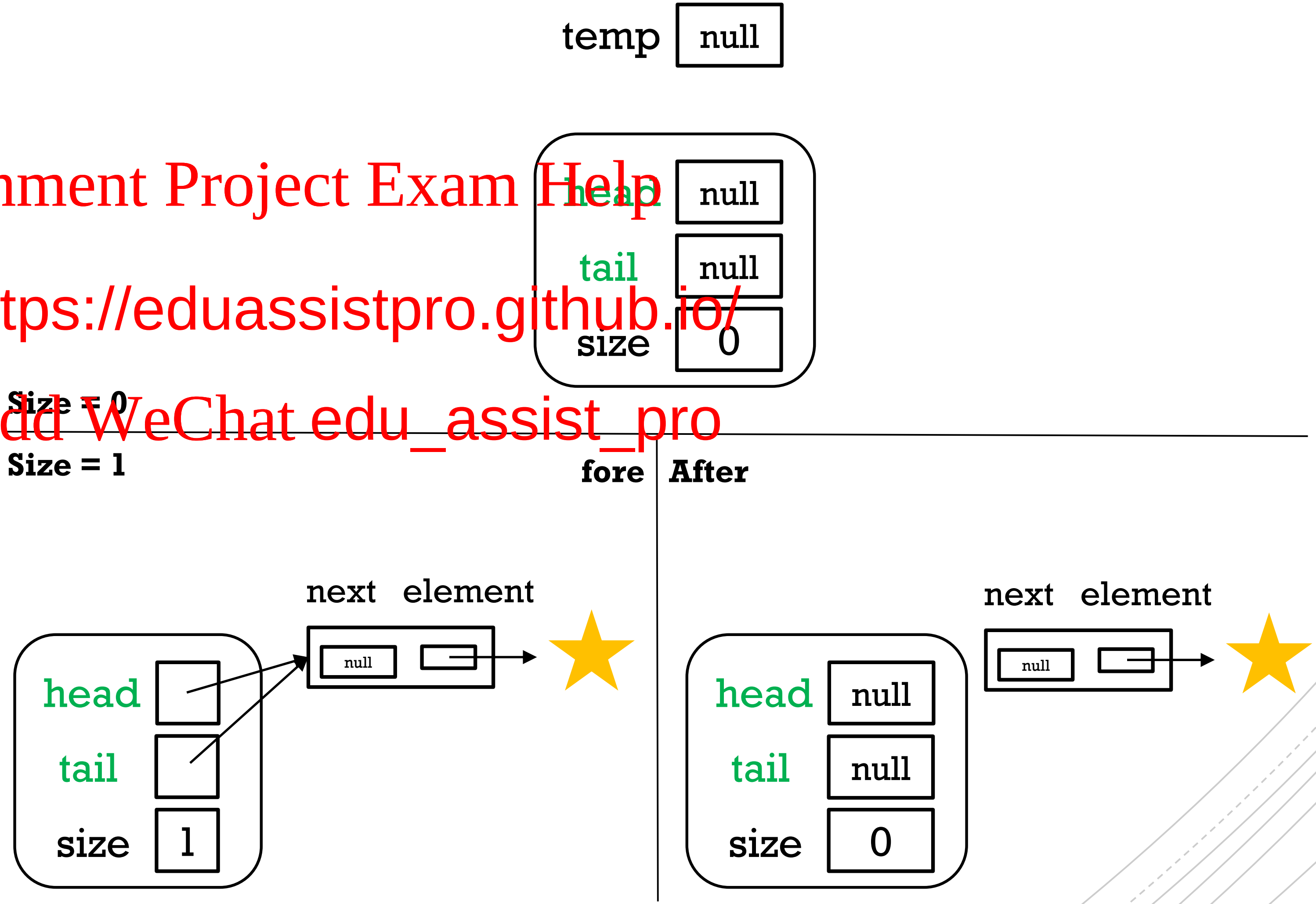
removeLast() – PSEUDOCODE

```
SNode tmp = head;
while (tmp.next != tail)
      tmp = tmp.next;


tail = tmp;
tail.next = null;
size = size - 1;
// to return the element,
// you need to do a bit more work
// edge cases for size = 0 and 1 to be added
```



removeLast() – EDGE CASES (SIZE IS 0 OR 1)

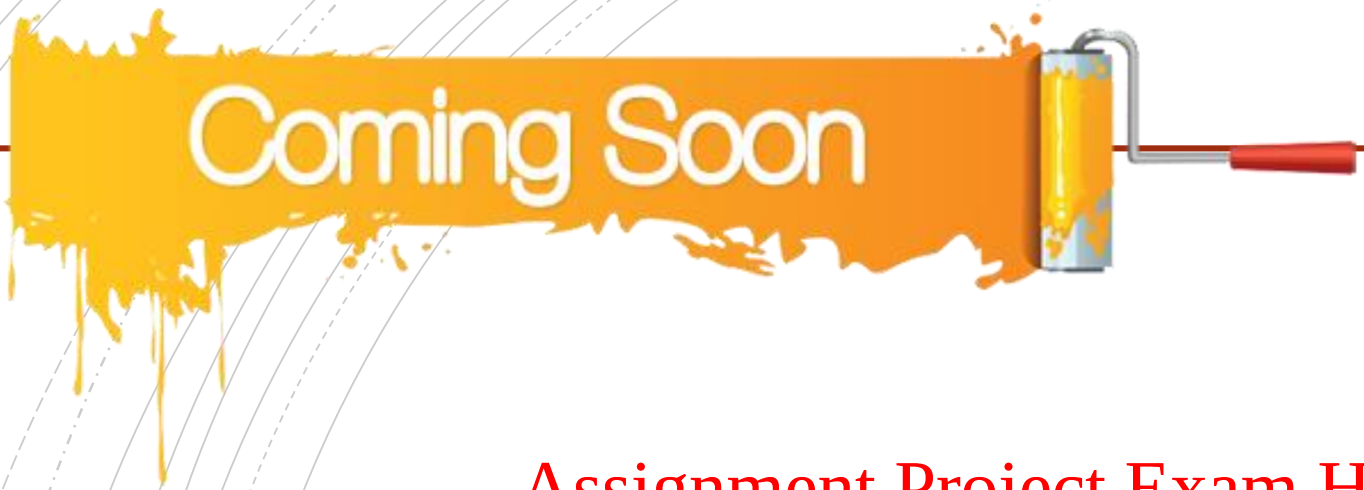
```
if (size == 0)
      throw exception
if (size == 1)
      head = null;
      tail = null;
else {
      ...
}
size = size - 1;
```



WORSE CASE TIME COMPLEXITY (N = LIST SIZE)

	array list	linked list
addFirst()		O(1)
removeFirst()	O(N	O(1)
addLast()	O(1)*	**O(1)**
removeLast()	O(1)	**O(N)**

*if array is not full



In the next

- Doubly