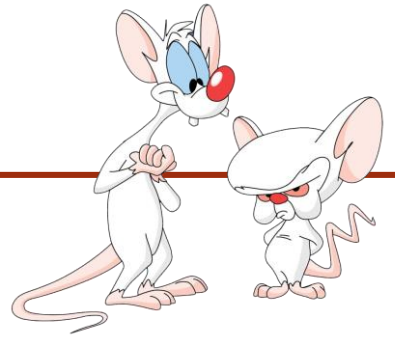
COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 3 - Double

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Doubly Linked Lists

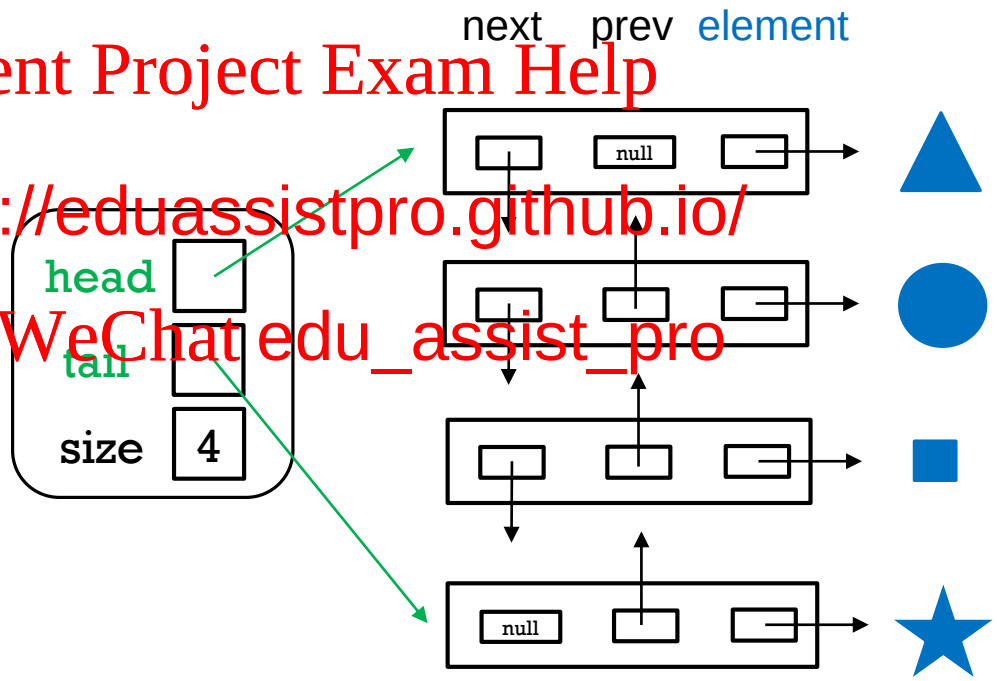
IMPLEMENTATIONS

There are different implementations of a list:

- Array list
- Singly linked list
- Doubly linked list

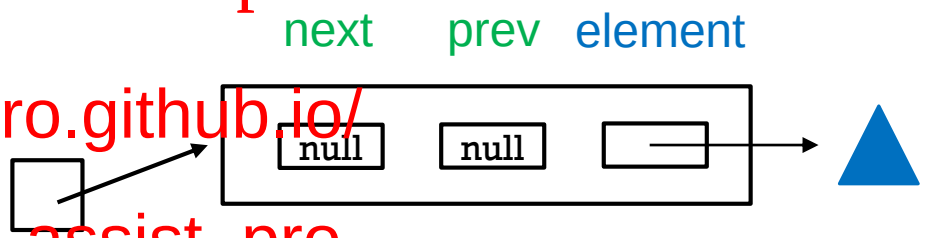
DOUBLY LINKED LIST

Each node has a reference to the next node *and* to the previous node.



DOUBLY LINKED LIST NODE

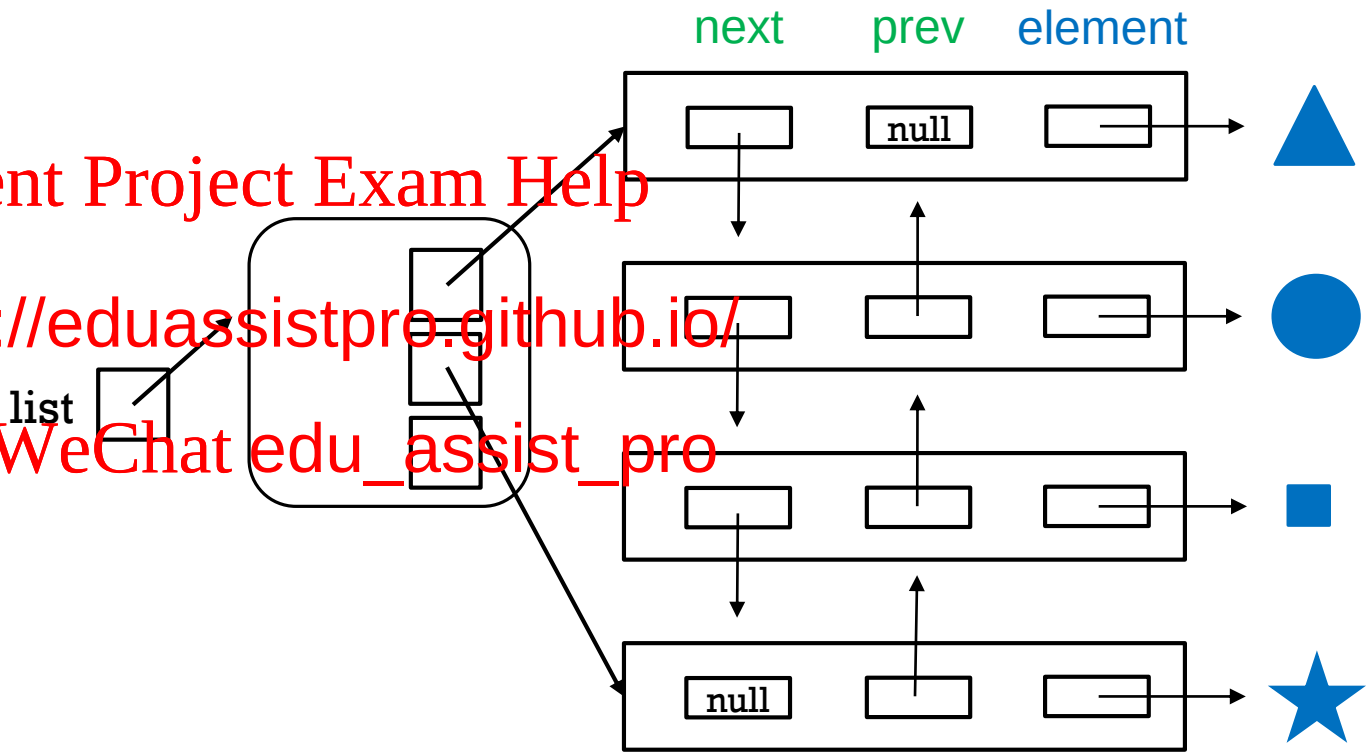
```
class DNode {
    Shape element;
    DNode next;
    DNode prev;
}
```


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

DOUBLY LINKED LIST

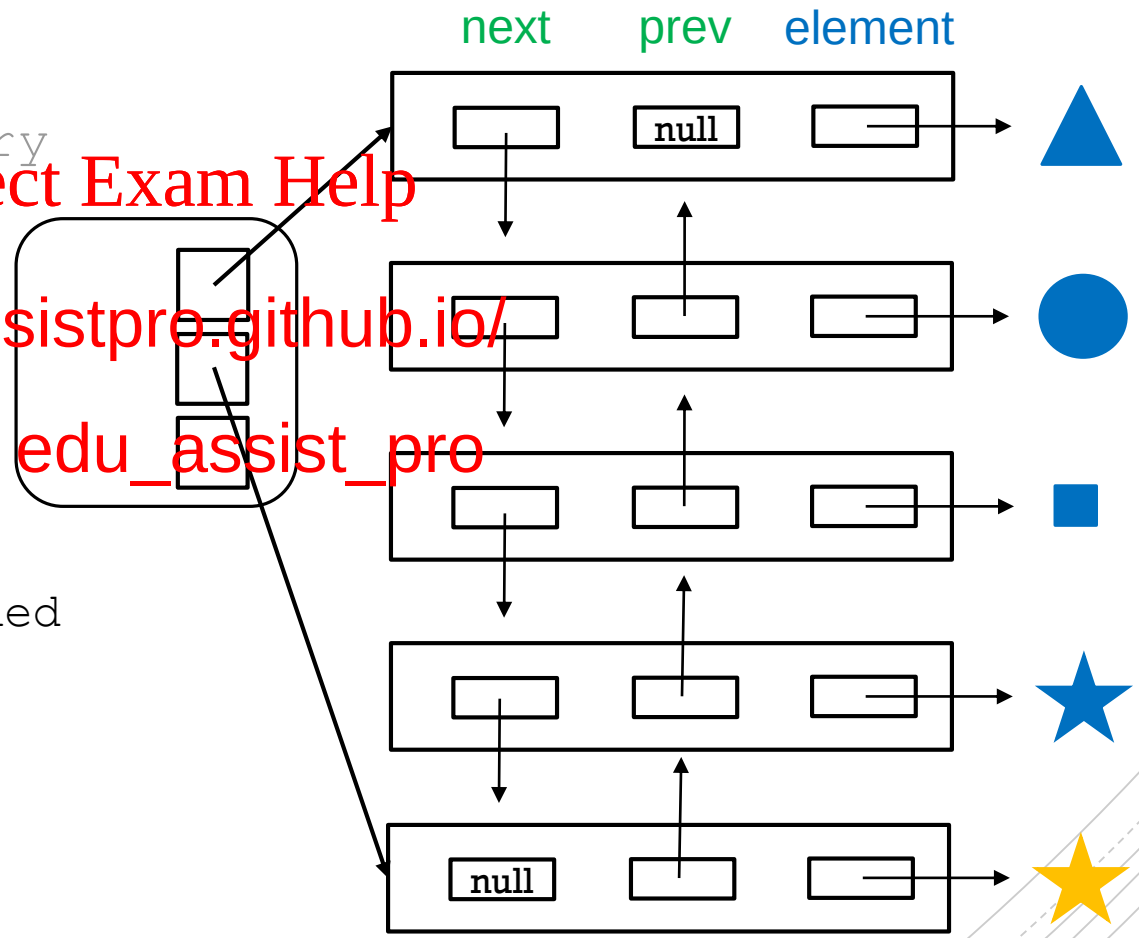
```
public class DLinkedList {
    private DNode head;
    private DNode tail;
    private int size;
    :
    private class DNode {
        Shape element;
        DNode next;
        DNode prev;
    }
}
```


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

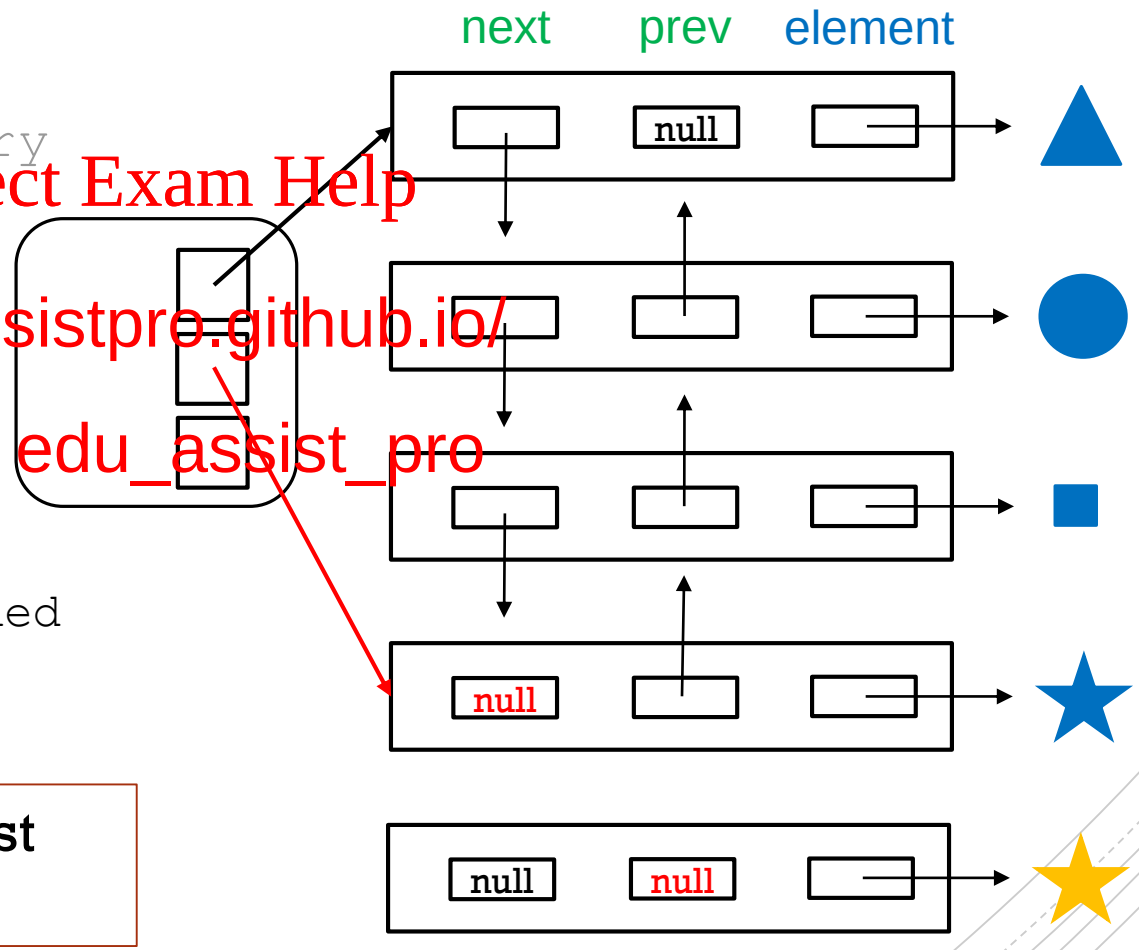
DOUBLY LINKED LIST – removeLast()

```
tail = tail.prev;

tail.next.prev = null; // not necessary

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
```

DOUBLY LINKED LIST – removeLast()

```
tail = tail.prev;

tail.next.prev = null; // not necessary

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
```

For a doubly linked list, removing the last element is much faster.

WORSE CASE TIME COMPLEXITY (N = LIST SIZE)

	array list	SLinkedList	DLinkedList
addFirst()		)	O(1)
removeFirst()	O(N)		O(1)
addLast()	O(1)	O(1)	O(1)
removeLast()	O(1)	O(N)	O(1)

OTHER LIST OPERATIONS

Many list operations require access to a specific node `i`

get(i)

set(i,e)

add(i,e)

remove(i)

:

LINKED LISTS

Suppose we want to access general node `i` in a linked list.

Two issues arise:

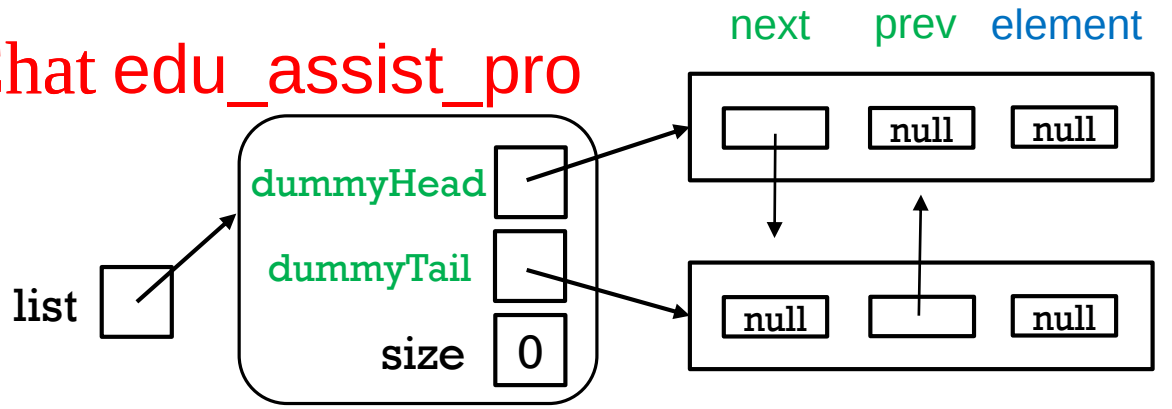
- Edge cases (i = 0, i = size – 1) require extra code.
  This is a pain and can lead to co .

- How long does it take to access node `i`?

AVOID EDGE CASES WITH "DUMMY NODES"

```
public class DLinkedList {
    private DNode dummyHead;
    private DNode dummyTail;
    private int size;
    :
    public DLinkedList(){
        dummyHead = new DNode();
        dummyTail = new DNode();
        dummyHead.next = dummyTail;
        dummyTail.prev = dummyHead;
        size =0;
    }
}
```

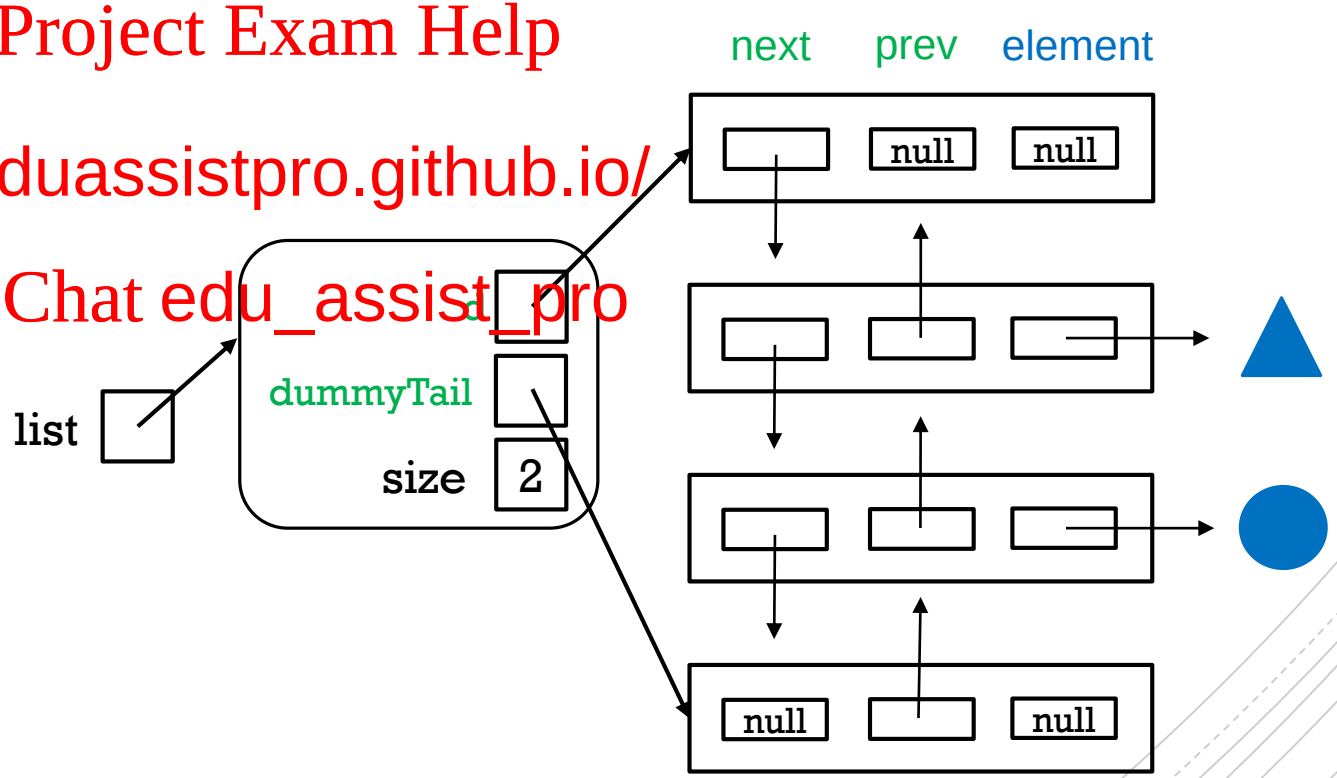
```
// empty list
DLinkedList list = new DLinkedList();
```

AVOID EDGE CASES WITH "DUMMY NODES"

```
public class DLinkedList {
    private DNode dummyHead;
    private DNode dummyTail;
    private int size;
    :
    public DLinkedList(){
        dummyHead = new DNode();
        dummyTail = new DNode();
        dummyHead.next = dummyTail;
        dummyTail.prev = dummyHead;
        size =0;
    }
}
```

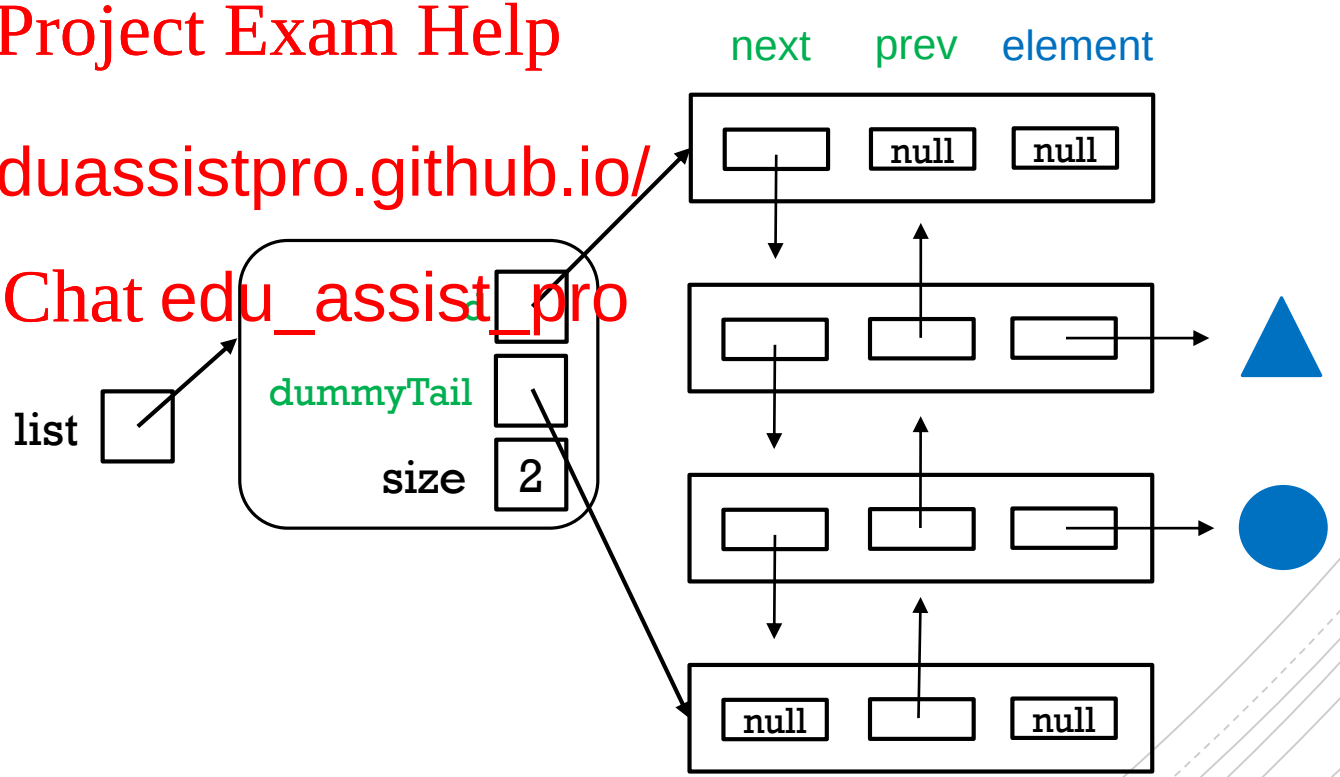
```
DLinkedList list = new DLinkedList();
// add 2 elements…
```

HOW DO WE ACCESS A NODE? – get()

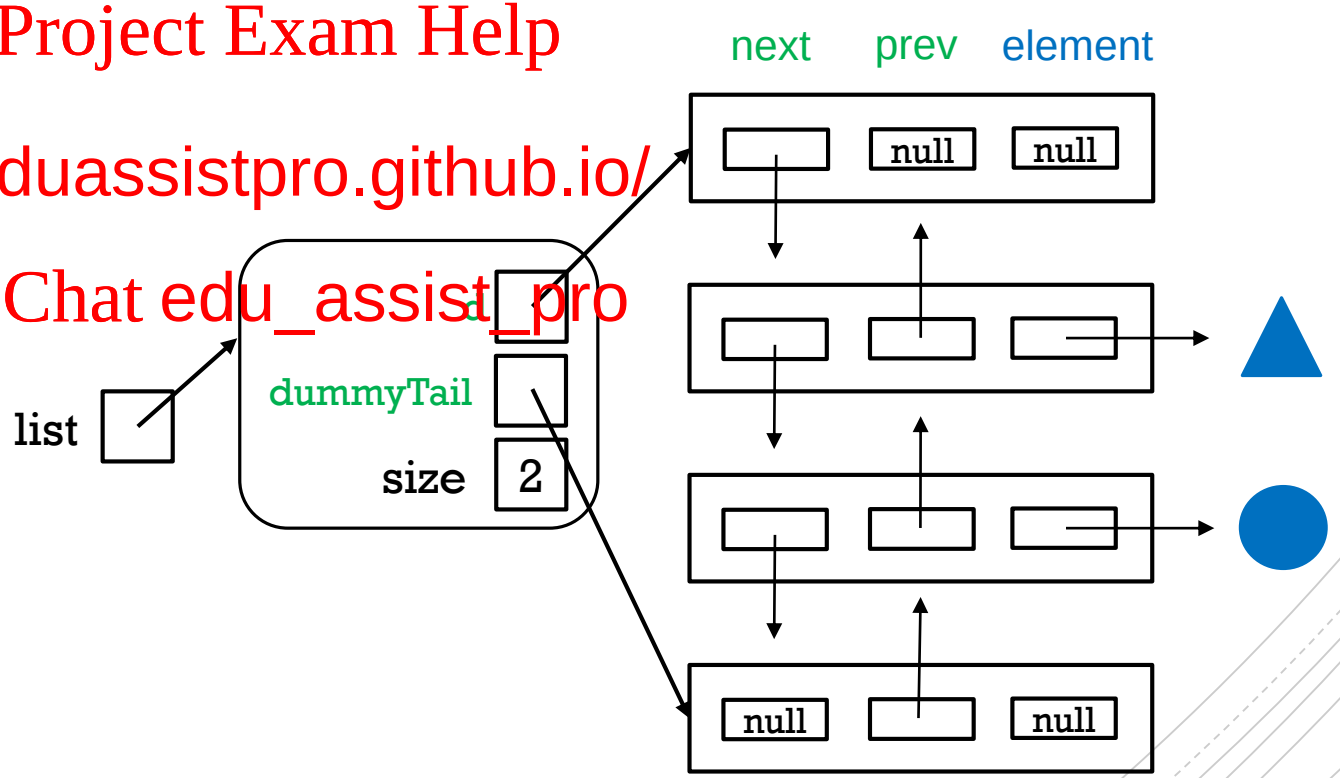
```
public Shape get(int i) {
    DNode node = getNode(i);
    return node.element;
}
```

```
private DNode getNode(int i) {
    // verify that 0<=i<size omitted
    DNode node = dummyHead.next;
    for(int k=0; k<i; k++)
        node = node.next;
    return node;
}
```

CAN WE SPEED THIS UP? – getNode()

```
private DNode getNode(int i) {
    // verify that 0<=i<size omitted
    DNode node;
    if (i < size/2) {
        node = dummyHead.next;
        for(int k=0; k<i; k++)
            node = node.next;
    }
    else {
        node = dummyTail.prev;
        for(int k=size -1; k>i; k--)
            node = node.prev;
    }
    return node;
}
```

JAVA LINKEDLIST CLASS

https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

It uses a *doubly linked list* as the underlying data structure.

It has some methods thag.:

- addFirst()
- removeFirst()
- addLast()
- removeLast()

Why ?

Q: What is the time complexity of the following ?

```
DLinkedList list =   new  DLinkedList( ) ;
for (k = 0; k                          s some constant
          li
```

Q: What is the time complexity of the following ?

```
DLinkedList list =   new  DLinkedList( ) ;
for (k = 0; k                          s some constant
          li
```


A: $\mathbf{1} + \mathbf{1} + \mathbf{1} + \ldots. \ \mathbf{1} = \boldsymbol{N} \quad \Rightarrow \quad \boldsymbol{O}(\boldsymbol{N})$

where ‘1′ means constant.

Q: What is the time complexity of the following ?

```
for (k = 0; k < list.size(); k ++) //  size == N
    list.get(k);
```


Assuming here that `getNode(i)` *always starts at the head.*

Q: What is the time complexity of the following ?

```
for (k = 0; k < list.size(); k ++) //  size == N
    list.get(k);
```

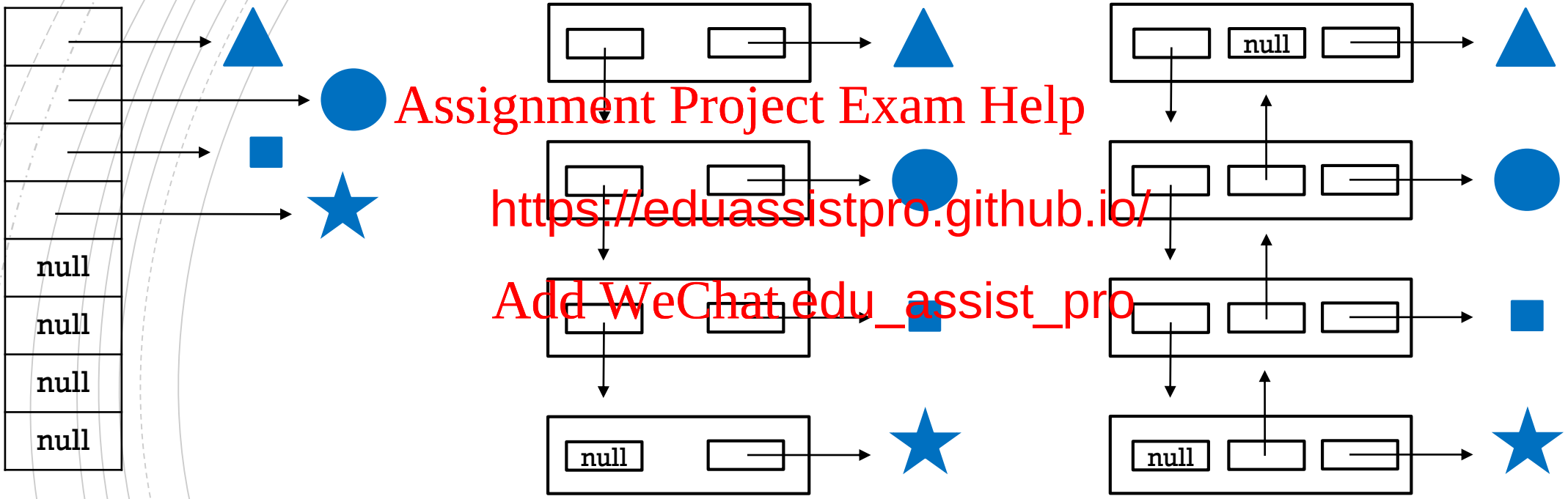

Assuming here that `getNode(i)` *always starts at the head.*

A: $\mathbf{1 + 2 + 3 + \ldots . N}$

$$= \frac{N\,(N+1)}{2} \quad \Rightarrow \quad O(N^2)$$

In 3 weeks we'll talk about a more efficient way to iterate through elements in a (Java) LinkedList!

WHAT ABOUT "SPACE COMPLEXITY" ?



All three data structures use space O(N) for a list of size N.
But linked lists use 2x (single) or 3x (double).

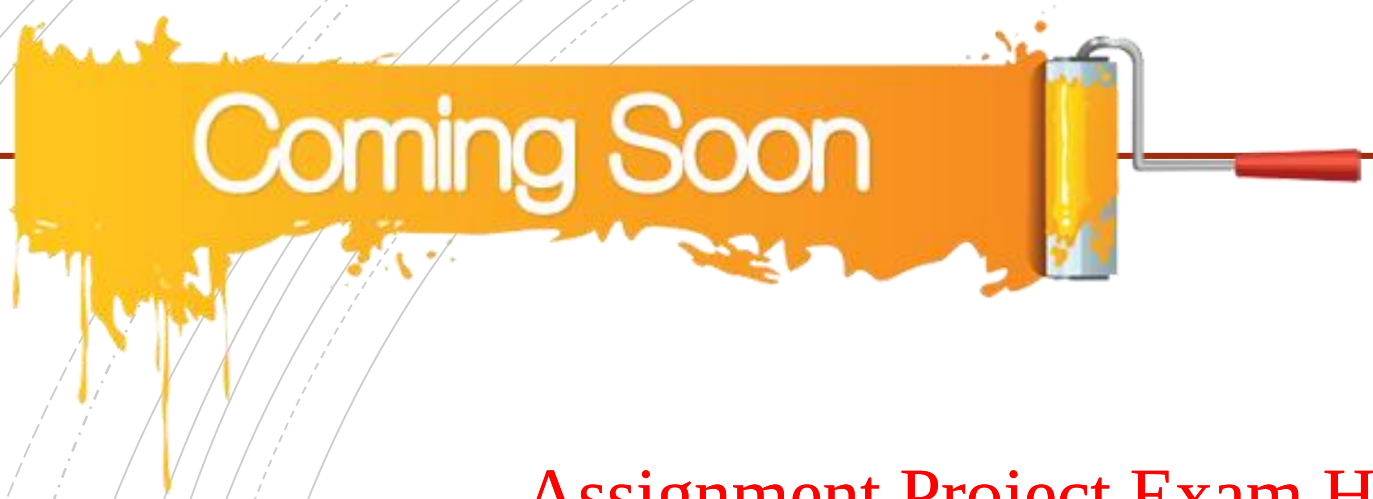
ARRAY LIST VERSUS LINKED LIST ?

Array lists and linked lists both take O(N) time to add or remove from an arbitrary position in the list.

In practice and w lists are faster. But the reasons are subtle and hav h how computer memory works, in particular, exploit contiguous memory allocation. You will learn about that topic in COMP 273.

DO YOU EVER NEED LINKED LISTS ?

Yes. Even if you prefer ArrayLists, you still need to understand Li are special cases of a general a ucture called a *tree* which we will be discus sively.



In the next

- Quadrat

- Asymptotic notations