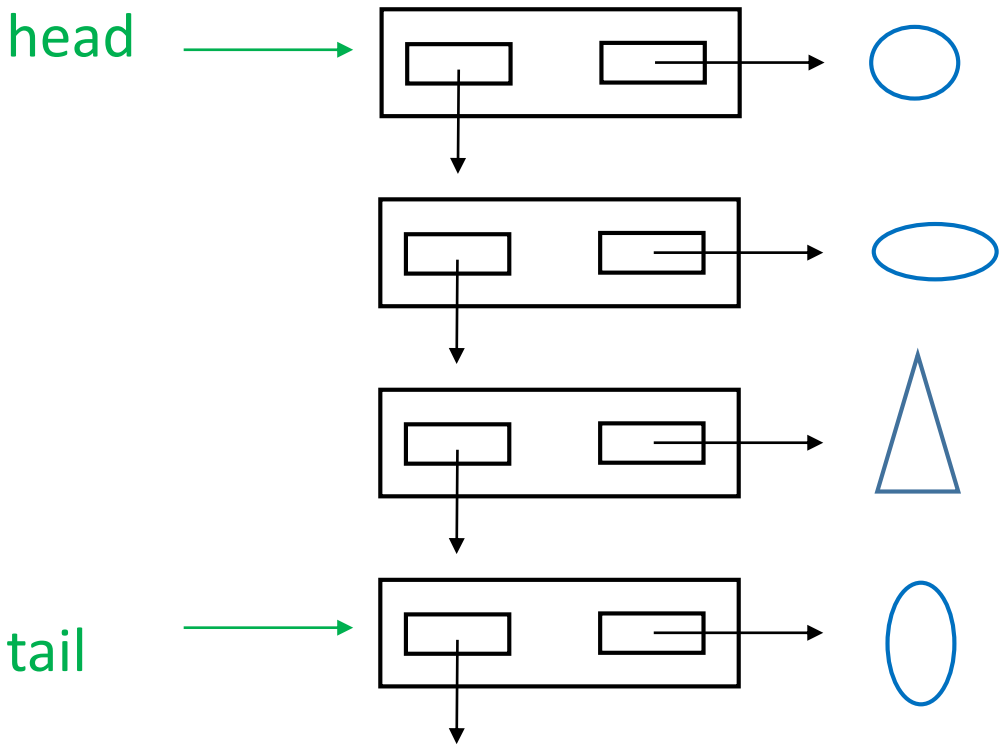
COMP 250

Lecture 6

doubly linked lists

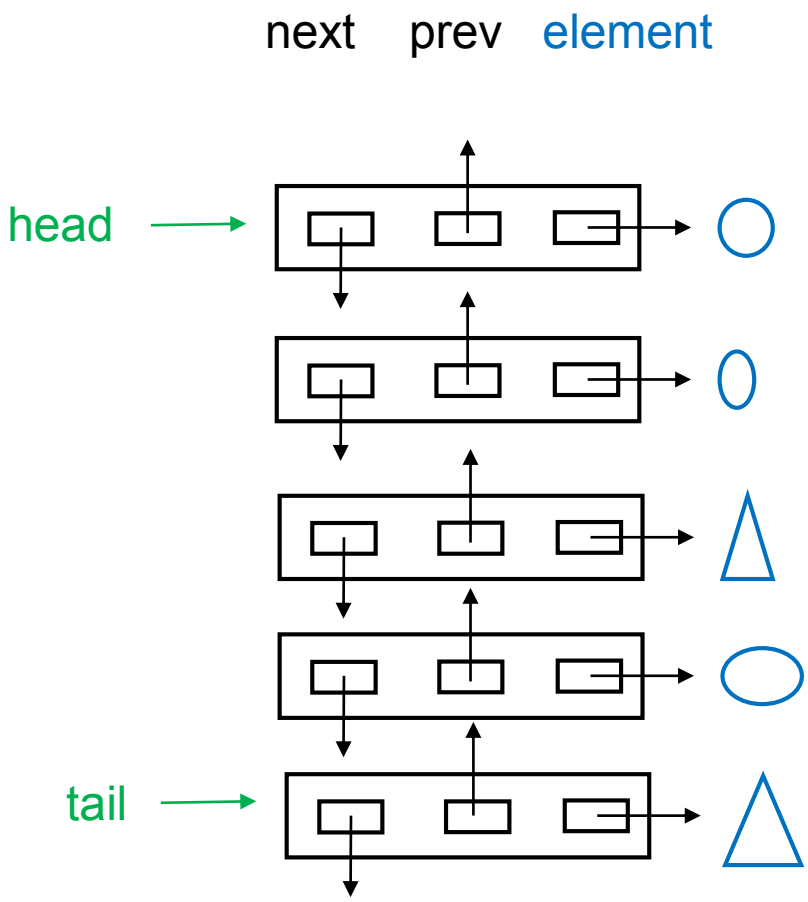
Sept. 20/21, 2017

Singly linked list

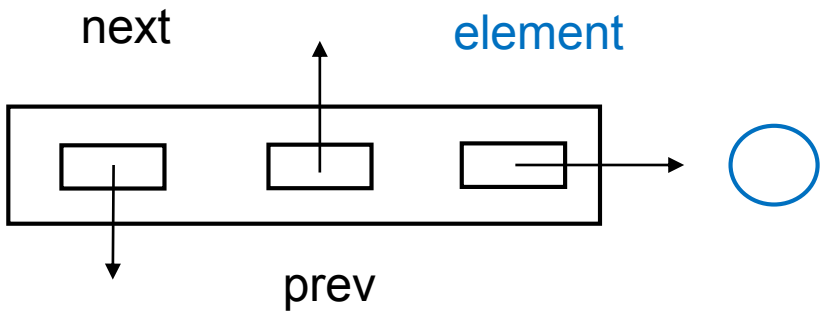


Doubly linked list

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



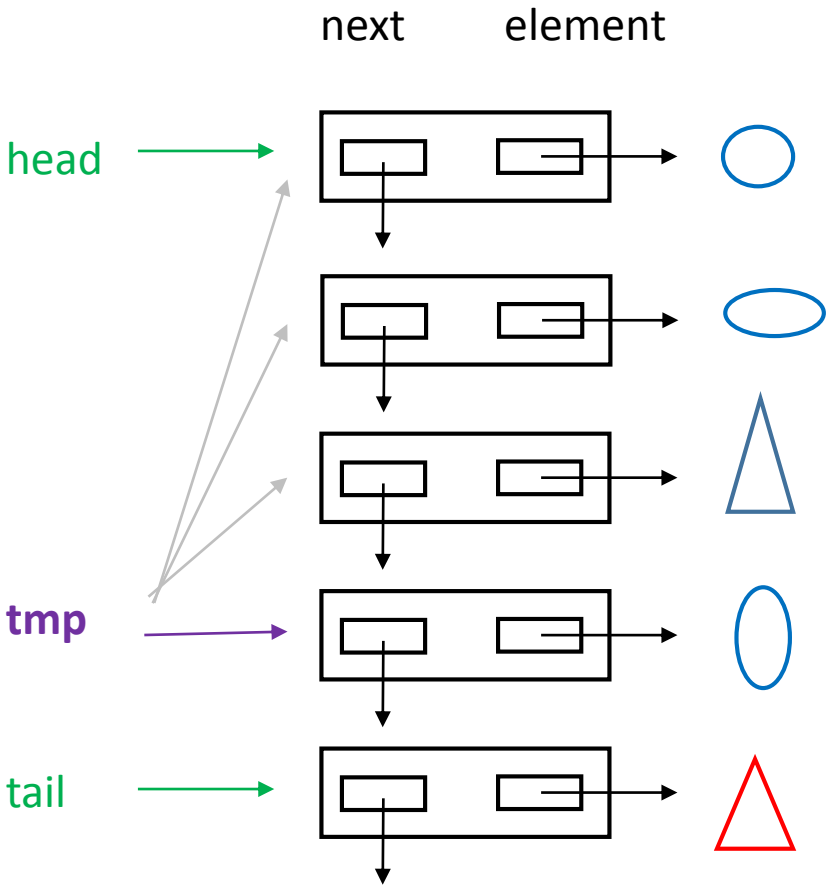
```
class DNode< E > {

    DNode< E >          next;
    Dnode<  E >         prev;
    E                   element;

   // constructor

    DNode( E    e ) {
        element = e;
        prev = null;
        next = null;
    }

}
```



Motivation for doubly linked lists:

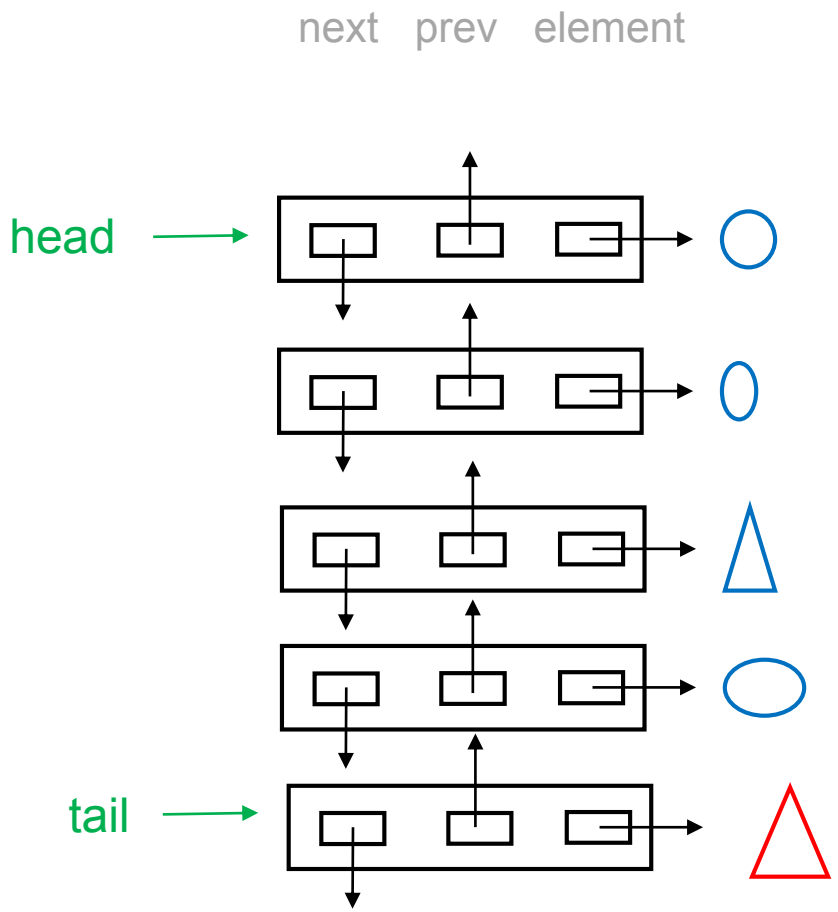
recall removeLast ( ) for singly linked lists

The only way to access the element before the tail was to loop through all elements from the head.



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

```
removeLast(){

        tail        = tail.prev
        tail.next.prev = null
        tail.next = null
        size = size – 1
                :
}
```



You need to do more work to return it.

Time Complexity (N = list size)

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

Other List Operations

:

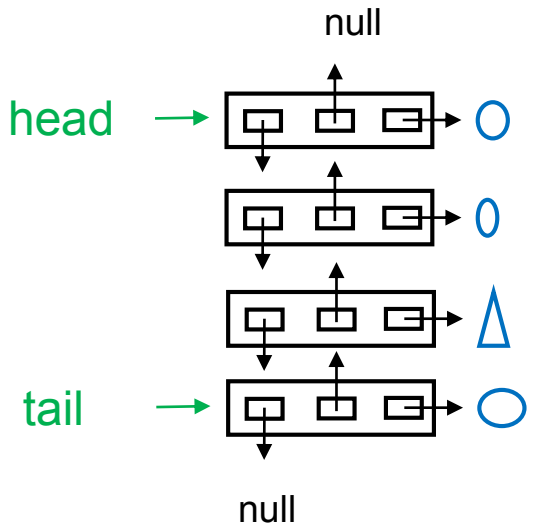
get(i)

set(i,e)

add(i,e)

remove(i)

:



Many list operations require access to node i.

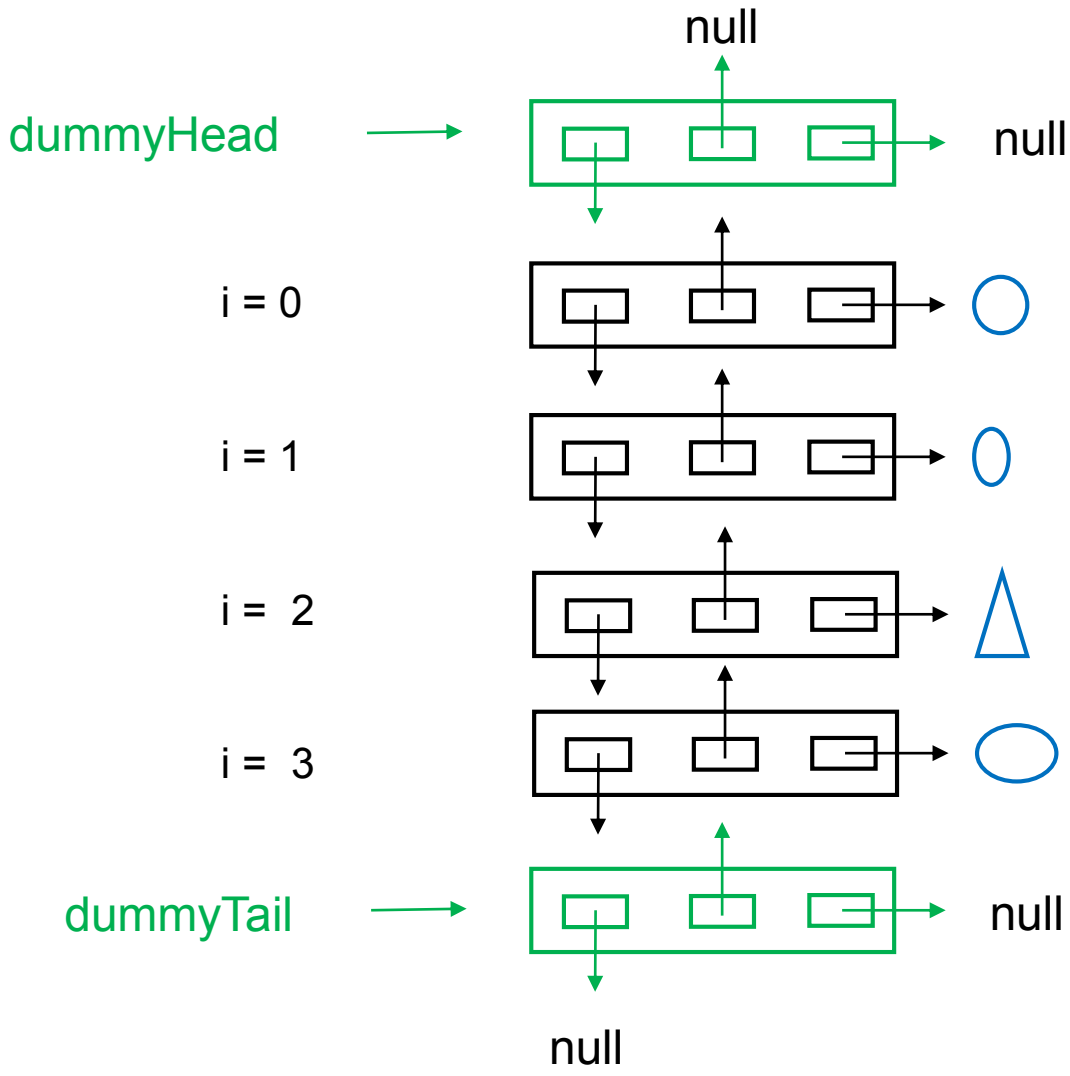
Suppose we want to access general node i in a linked list.

Two issues arise:

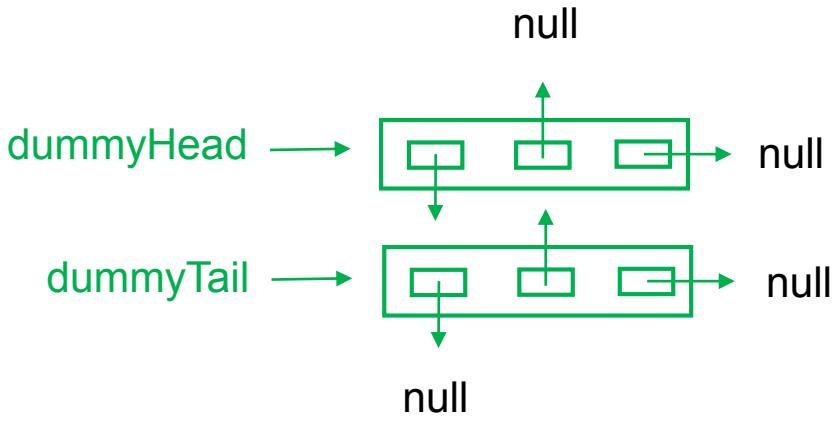
- Edge cases (i = 0, i = size – 1) require extra code.
  This is a pain and can lead to coding errors.

- How long does it take to access node i ?

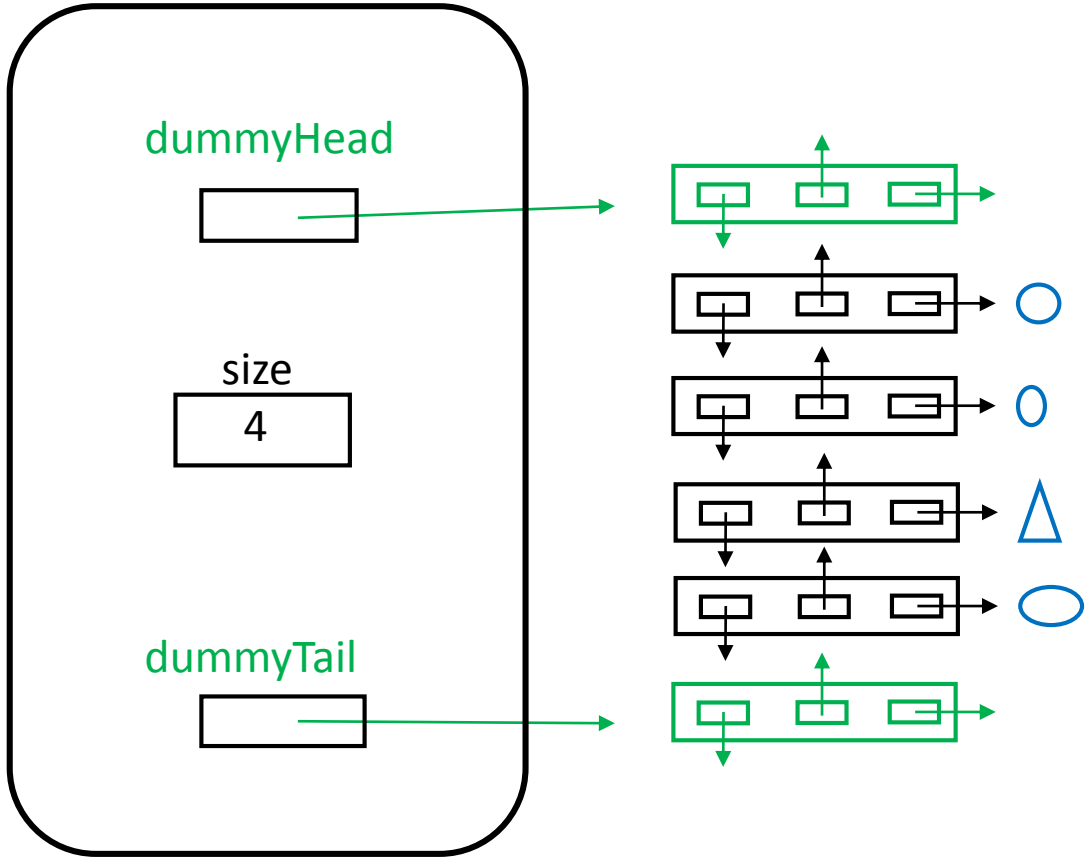
Avoid edge cases with “dummy nodes”



```
class  DLinkedList<E>{   //  Java code

    DNode<E>      dummyHead;
    DNode<E>      dummyTail;
    int           size;
     :

    //  constructor

    DLinkedList<E>(){
        dummyHead =  new  DNode<E>();
        dummyTail  =  new  DNode<E>();
        dummyHead.next  =  dummyTail;
        dummyTail.prev    =  dummyHead;
        size     = 0;
    }

    private class  DNode<E>{ … }
}
```



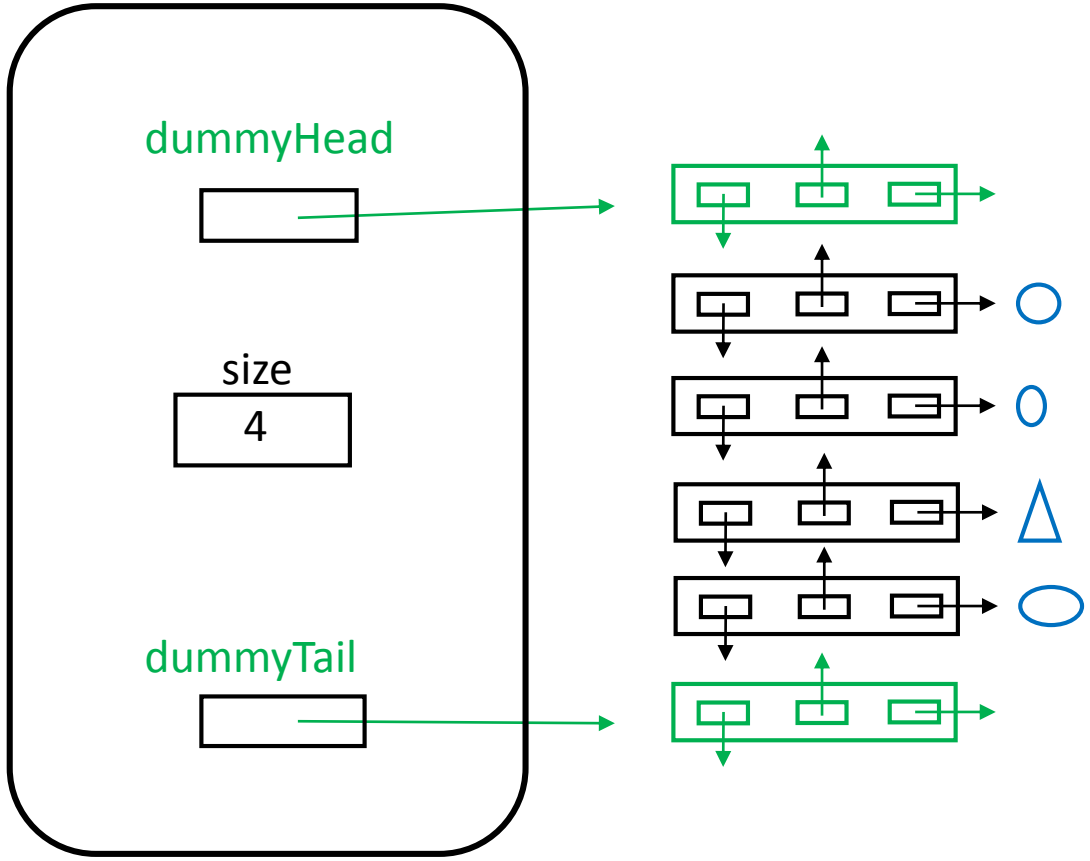
DLinkedList< Shape >
object



Q: How many objects in total in this figure?

A:

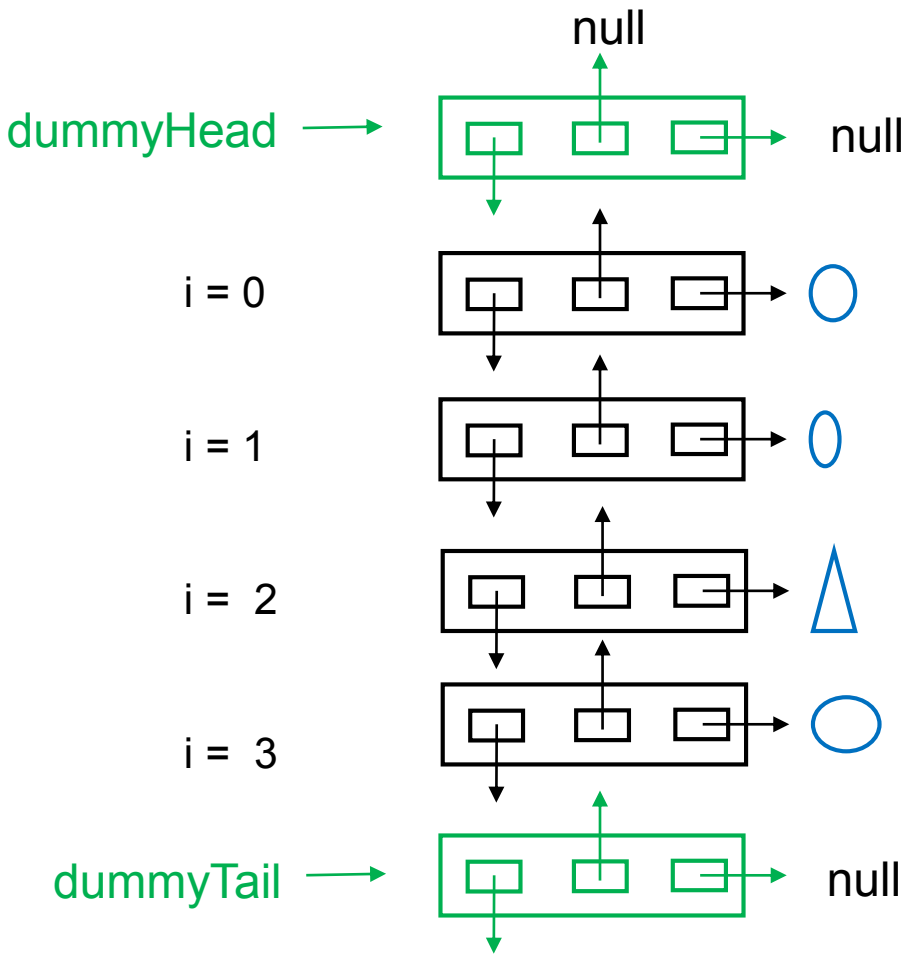
DLinkedList< Shape >
object



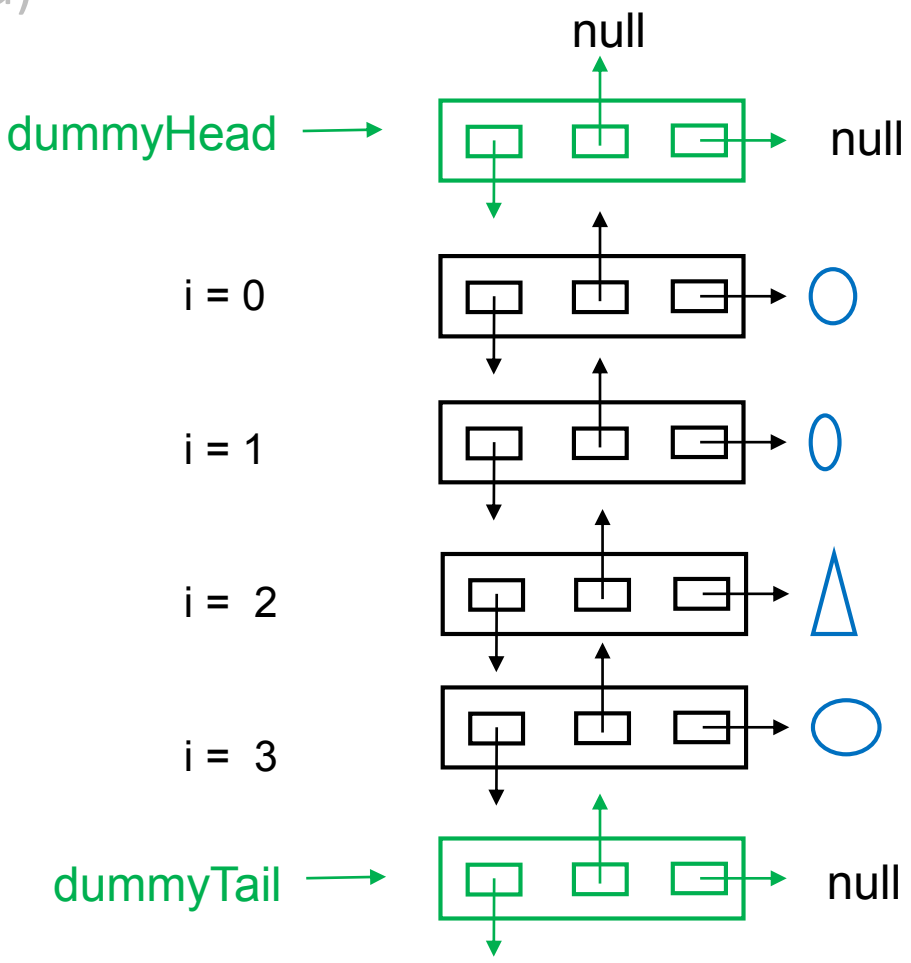
Q: How many objects in total in this figure?

A: 1 + 6 + 4 = 11

```
E    get( i ) {

     node =  getNode(i);    //  getNode() to be discussed next slide
     return node.element;

}
```



```
getNode( i ) {    //  returns a DNode

//    verify that 0 <= i < size   (omitted)

      node = dummyHead.next
      for (k = 0; k <  i ;   k ++)
            node = node.next
      return node
}
```



More efficient getNode()… half the time

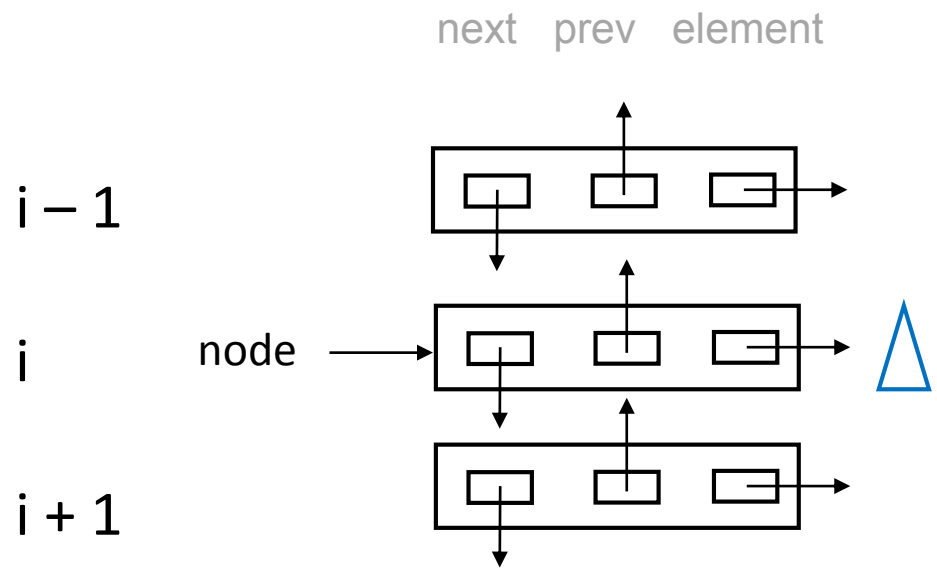
```
getNode( i ) {                                  //  returns a DNode

     if ( i < size/2 ){                         //  iterate from head
          node = dummyHead.next
          for (k = 0; k < i;   k ++)
                node = node.next
           }
      else{                                     //  iterate from  tail
          node = dummyTail.prev
          for ( k = size-1;   k >  i;   k -- )
                node = node.prev
           }
      return node
}
```

```
remove( i ) {
        node  =  getNode( i )
```

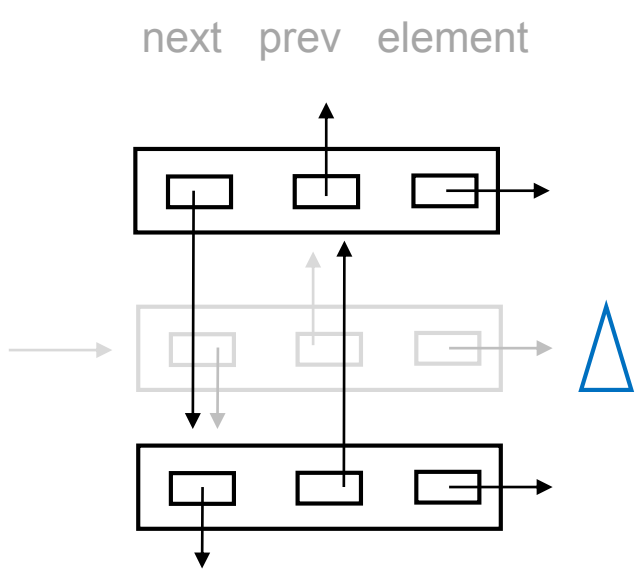
Exercise (see online code;
also reviewed in upcoming tutorial)

```
}
```

BEFORE



AFTER



Time Complexity (N = list size)

	array list	SLinkedList	DLinkedList
addFirst	O(N)	O(1)	O(1)
removeFirst	O(N)	O(1)	O(1)
addLast	O(1)	O(1)	O(1)
removeLast	O(1)	**O(N)**	**O(1)**
remove(i)	?	?	?

Time Complexity *in Worst Case*
(N = list size)

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

As I will discuss that later, "O( )" ignores constant factors.

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 when N is large, array lists are faster. But the reasons are subtle and have to do with how computer memory works, in particular, how caches 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 LinkedLists. Linked lists are special cases of a general and widely used data structure called a *tree* which we will be discussing extensively.

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 that ArrayList doesn't have e.g.:

- addFirst()
- removeFirst()

- addLast()
- removeLast()

Why ?

Q: What is the time complexity of the following ?

```
LinkedList< E >    list =   new  LinkedList< E >( ) ;

for (k = 0; k < N;   k ++)                    //  N is some constant
        list.addFirst(  new  E( …. )   );
```

Q: What is the time complexity of the following ?

```
LinkedList< E >   list =   new  LinkedList< E >( ) ;

for (k = 0; k < N;   k ++)                    //  N is some constant
        list.addFirst(  new  E( …. )   );     // or addLast(..)
```

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  );
```

Assume 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  );
```

Assume here that getNode(i) always starts at the head.

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

Q: What is the time complexity of the following ?

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

Assume here that getNode(i) always starts at the head.

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

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

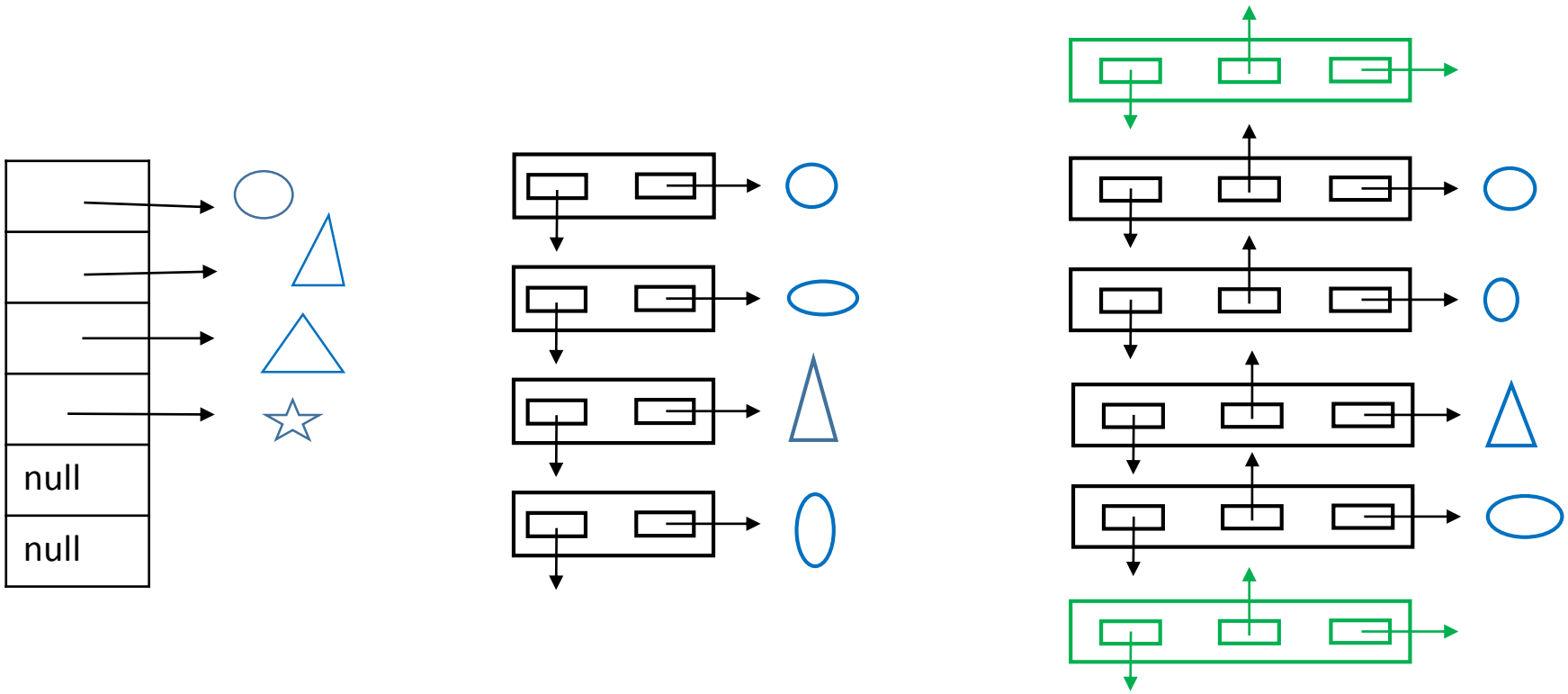
ASIDE: Java 'enhanced for loop'

A more efficient way to iterate through elements in a Java LinkedList is to use:

for (E e : list)

```
// 'list' references the LinkedList< E > object
// Do something with each element e in list

// But this is sometimes awkward. I don't recommend it for
// Assignment 1.
```

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).

Java terminology (time permitting)

- method “overloading”
  - add(int index, E element)
  - add(E element)

  - remove(E element)
  - remove(int i)

- method “signature”
  - name
  - number and type of parameters,
  - ~~return type~~

Java terminology

Method “overriding” vs. “overloading” ?

Classes can “inherit” methods from other classes.

I will cover inheritance formally at the end of the course.

But sometimes you do not want a class to inherit a method. Instead, you “override” the method by writing a more suitable one which has the same signature.

Announcements

- Quiz 0 solutions posted

  (Let me know if you have trouble viewing them)

- Assignment 1 posted (due on Tues. Oct. 3)
  - TA office hours posted and will be updated

- Tutorials for linked lists
  - Assumes you have attended/read up to today
  - You need to sign up.