COMP 250

Lecture 10

singly linked lists

Sept. 28, 2018

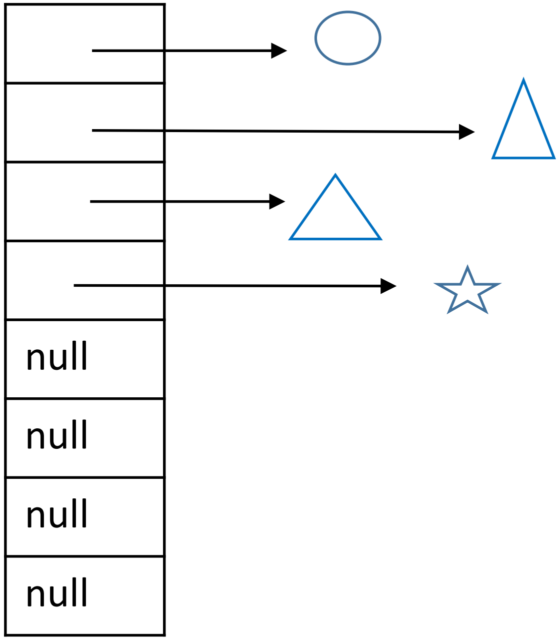
Lists

- array list (last lecture)
- singly linked list (today)
- doubly linked list (next lecture)

⋮

array list

linked list



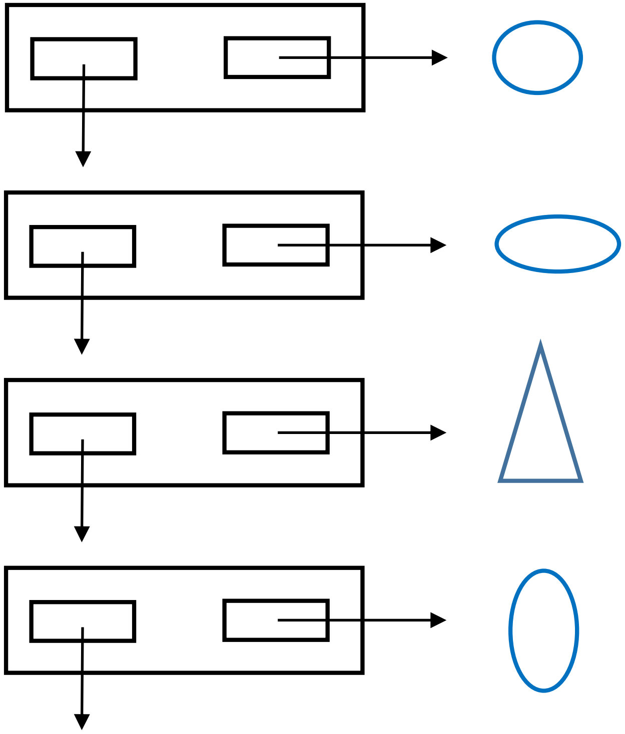
null

null

null

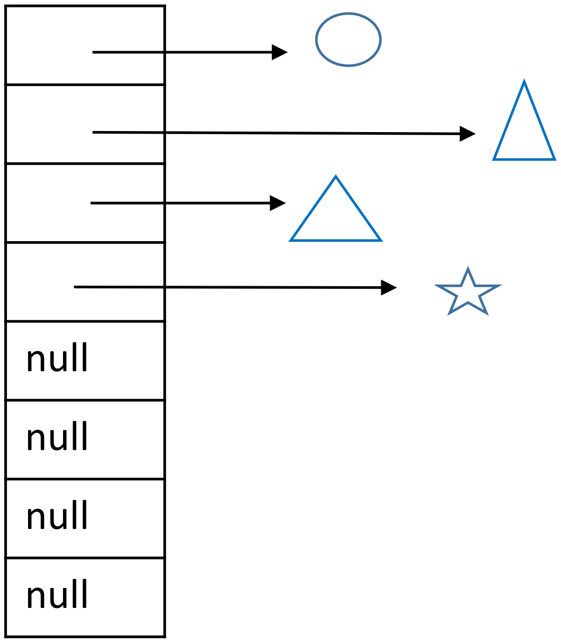
null

“nodes”



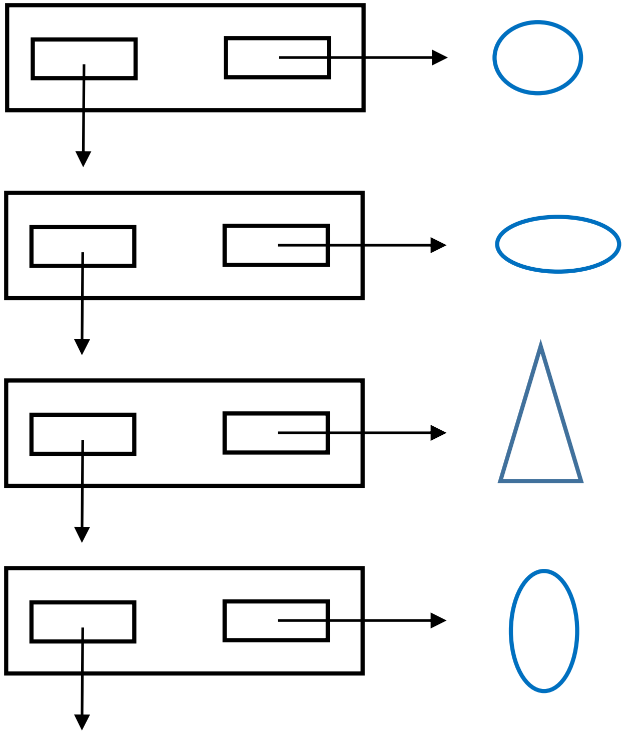
size = 4

array list



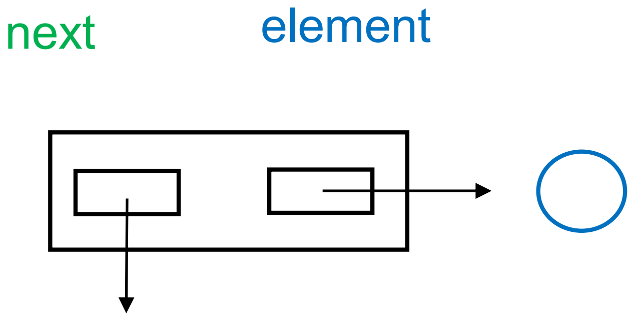
Array slots are in consecutive locations (addresses) in memory, but objects can be anywhere.

linked list



Linked list “nodes” and objects can be anywhere in memory.

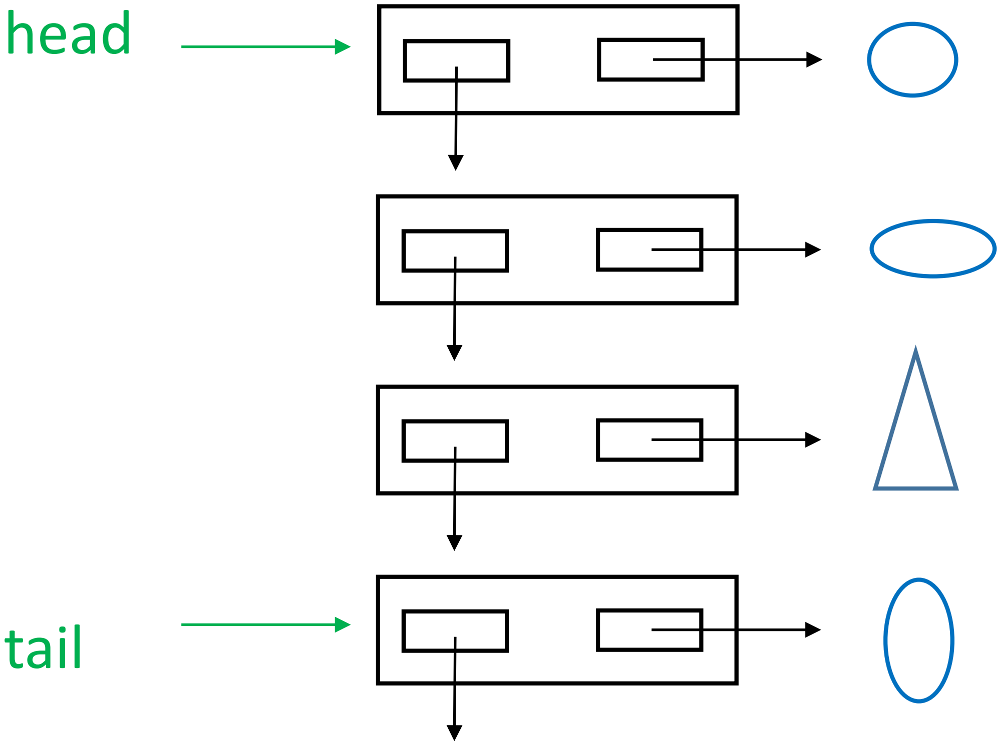
Singly linked list node (“S” for singly)



```
class   SNode<E> {

        SNode<E>   next;
        E          element;
        :
}
```

e.g. E might be Shape

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



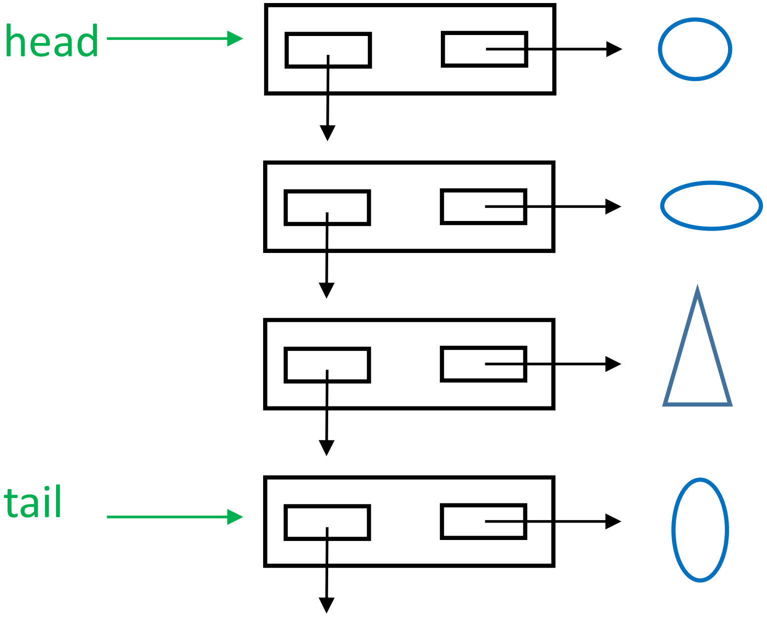
```
class   SLinkedList<E> {

        SNode<E>   head;
        SNode<E>   tail;
        int        size;

         :

         private   class   SNode<E> {        // inner class

                    SNode<E>   next;
                    E          element;
                     :
         }
}
```

Linked list operations

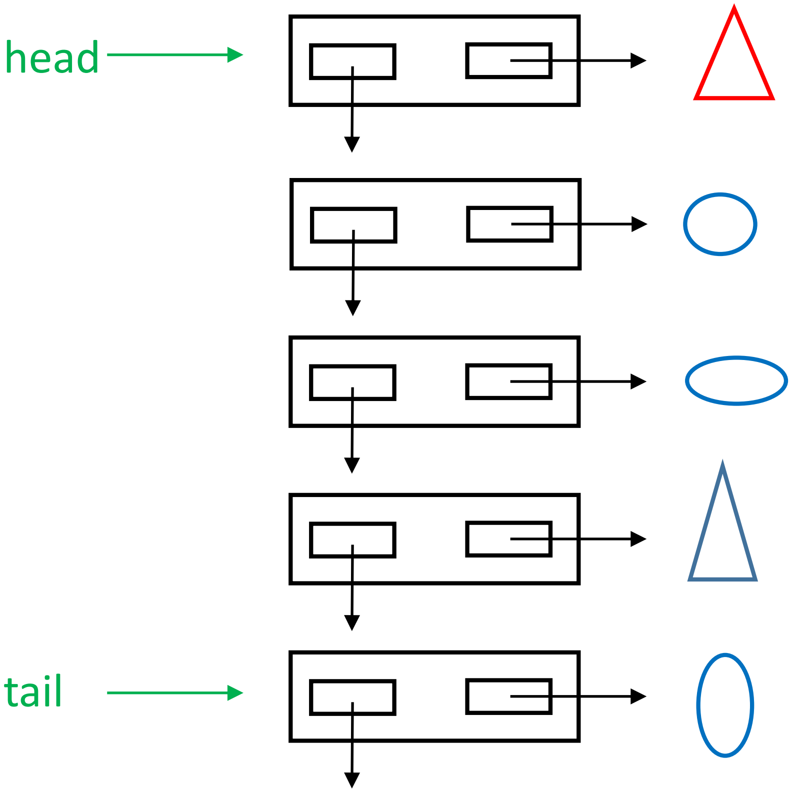
- addFirst (e)
- removeFirst( )
- addLast (e)
- removeLast( )
- ……. many other list operations

addFirst (△)

BEFORE

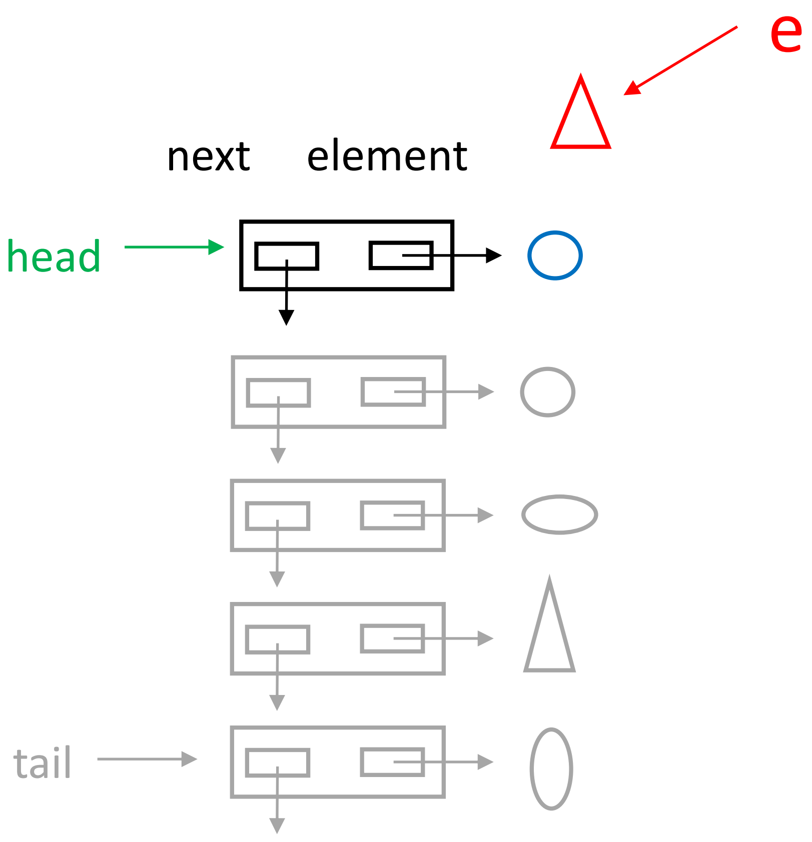


AFTER



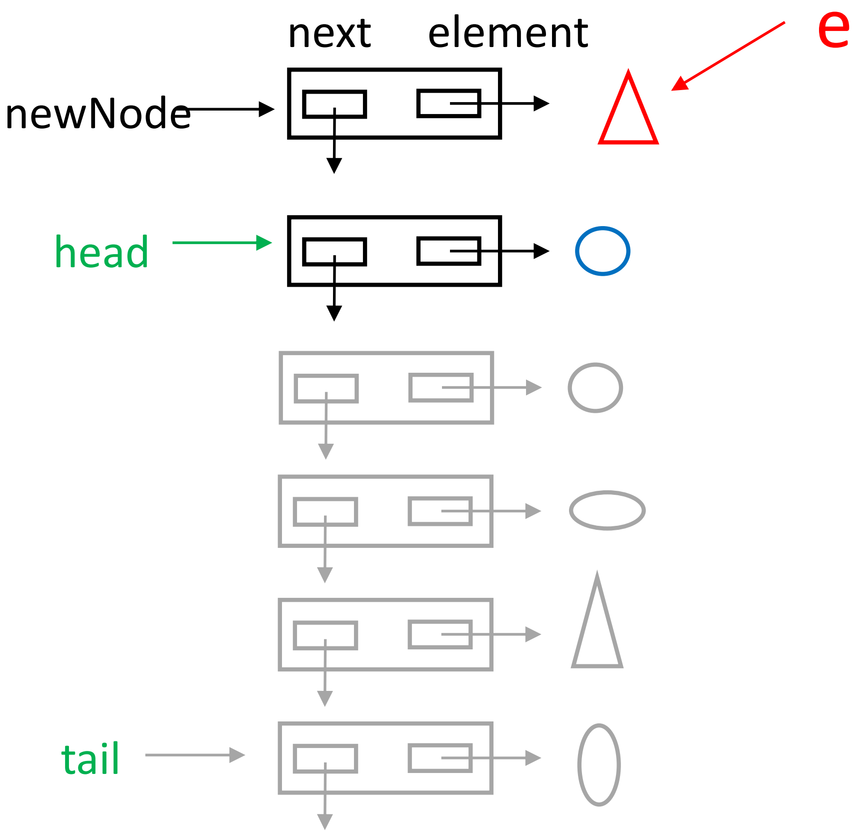
addFirst (e)

BEFORE



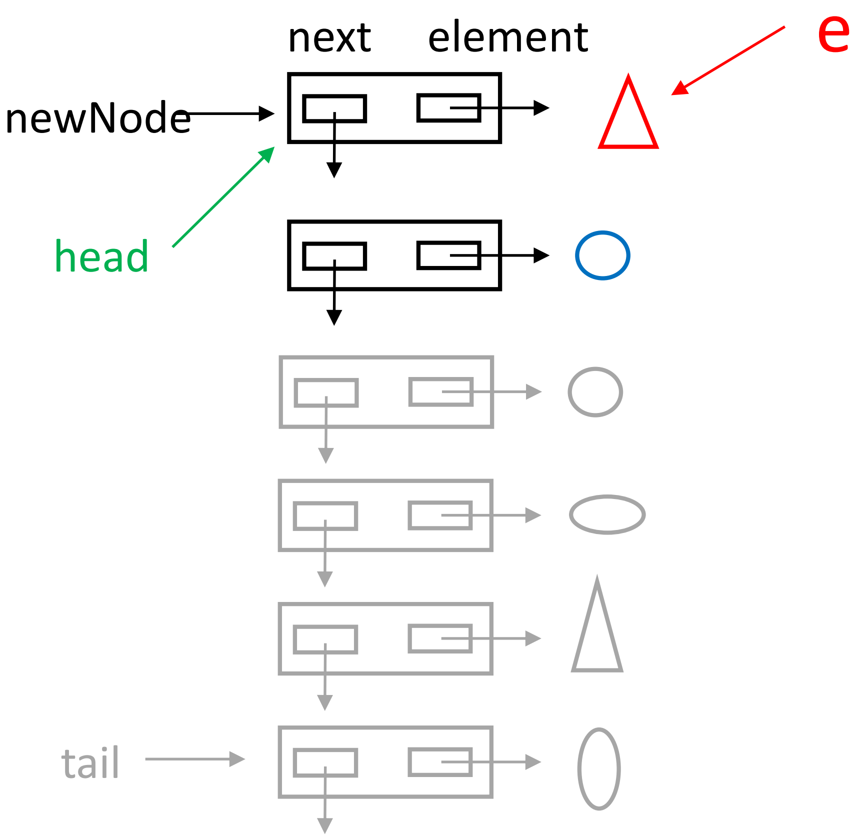
addFirst (e)

```
construct  newNode
newNode.element  =  e
newNode.next     =  head

head =  newNode
size = size+1

//  special case
if size == 1
     tail = head
```



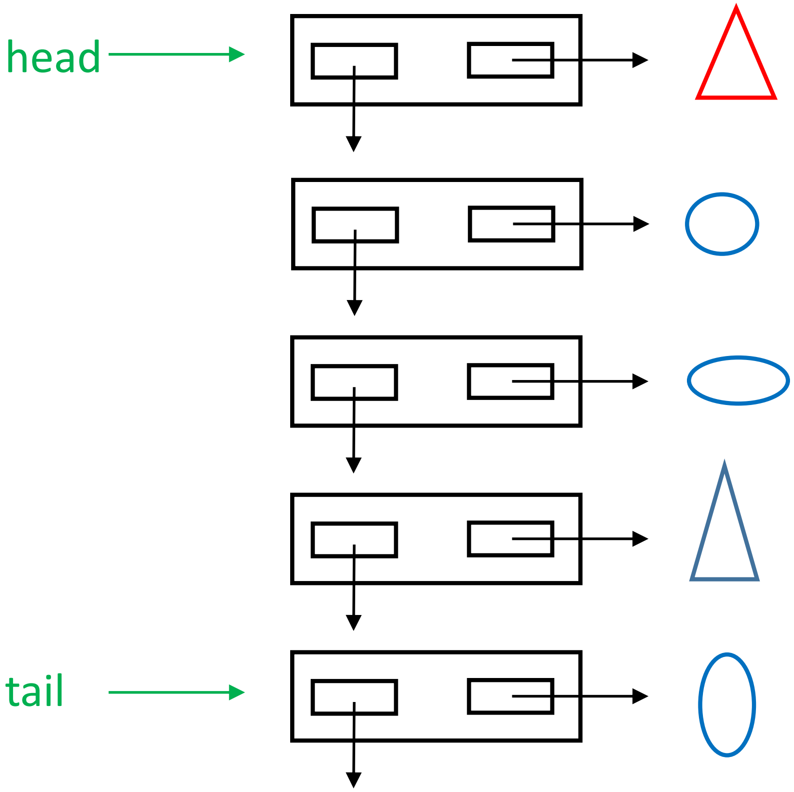
addFirst (e)   pseudocode

```
construct  newNode
newNode.element  =  e
newNode.next     =  head

head =  newNode
size = size+1

//  special case
if size == 1
     tail = head
```

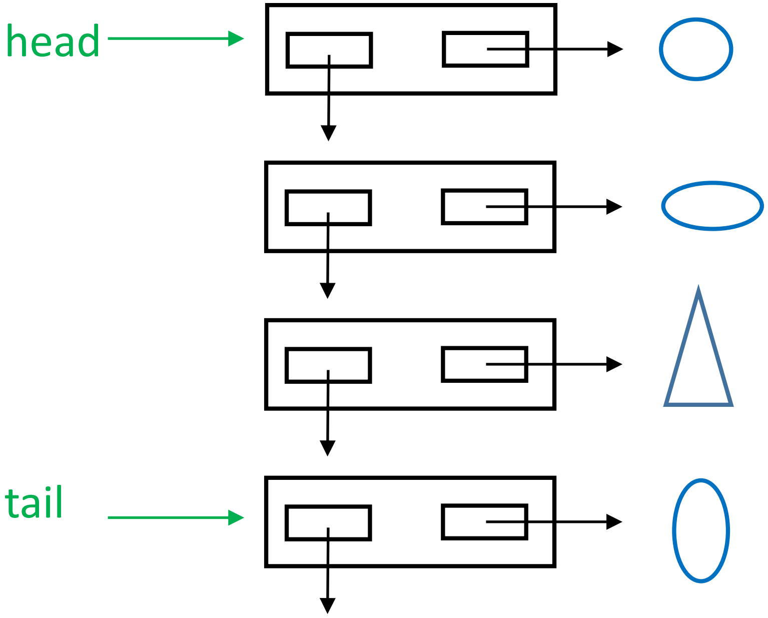


removeFirst ( )

BEFORE

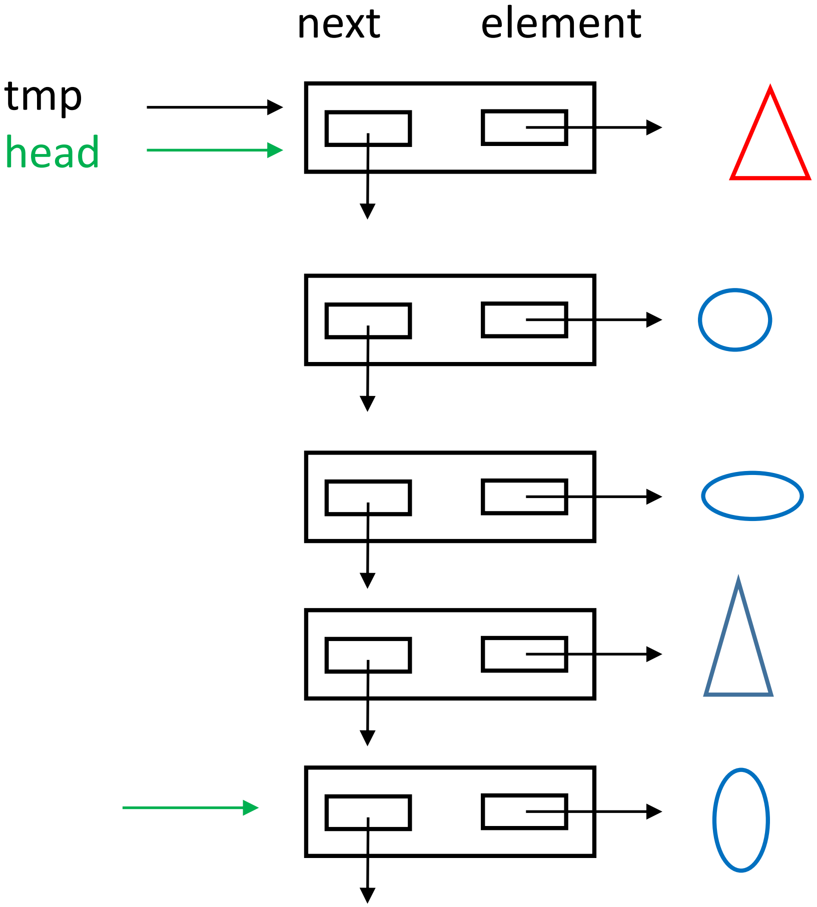


AFTER



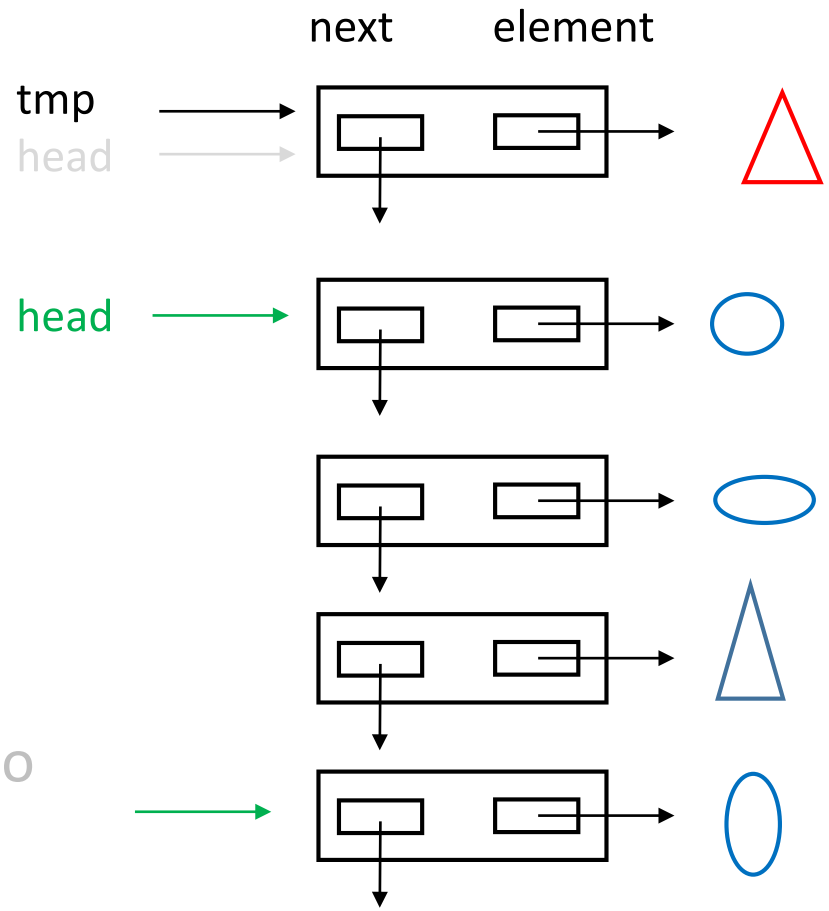
removeFirst ( )

```
tmp  =  head
if (size == 0)
   throw exception

head     =  head.next
tmp.next =  null
size = size – 1

if  (size == 0)
     tail = null
```



removeFirst ( )

```
tmp  =  head
if (size == 0)
   throw exception

head     =  head.next
tmp.next =  null    // unnecessary
size = size – 1

if  (size == 0)
     tail = null      //  head == null  also
```



Comparison of array lists and linked lists

The number of instructions to compute addFirst(e) and removeFirst( ) does not depend on the number of elements, N=size, in the linked list.

This is different from array lists! Recall add(0, e) and remove(0) for array lists required a “for” loop with N=size iterations.

Note:

- addFirst(e) achieves the same thing as add(0, e).
- removeFirst() achieves the same as remove(0).

Worse Case Time Complexity (N = size)

	array list	linked list
addFirst	O(N)	O(1)
removeFirst	O(N)	O(1)

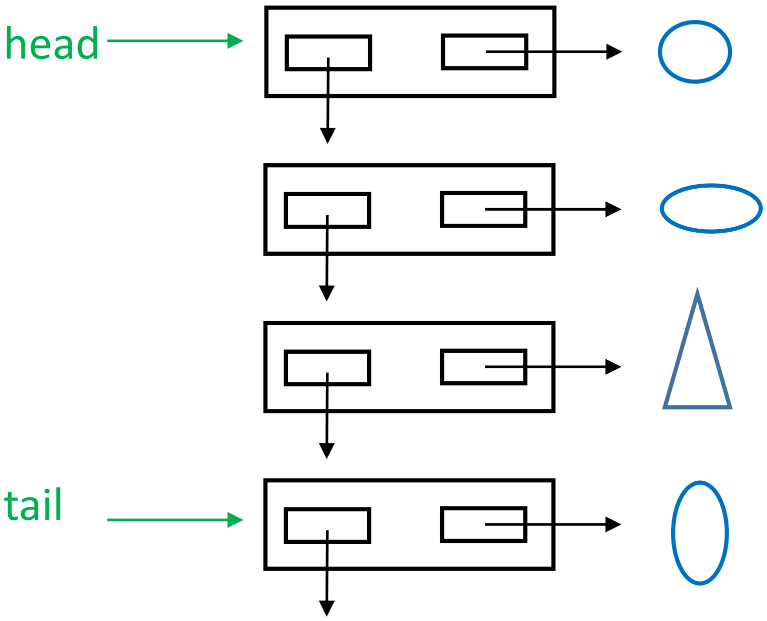
Worse Case Time Complexity (N = size)

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

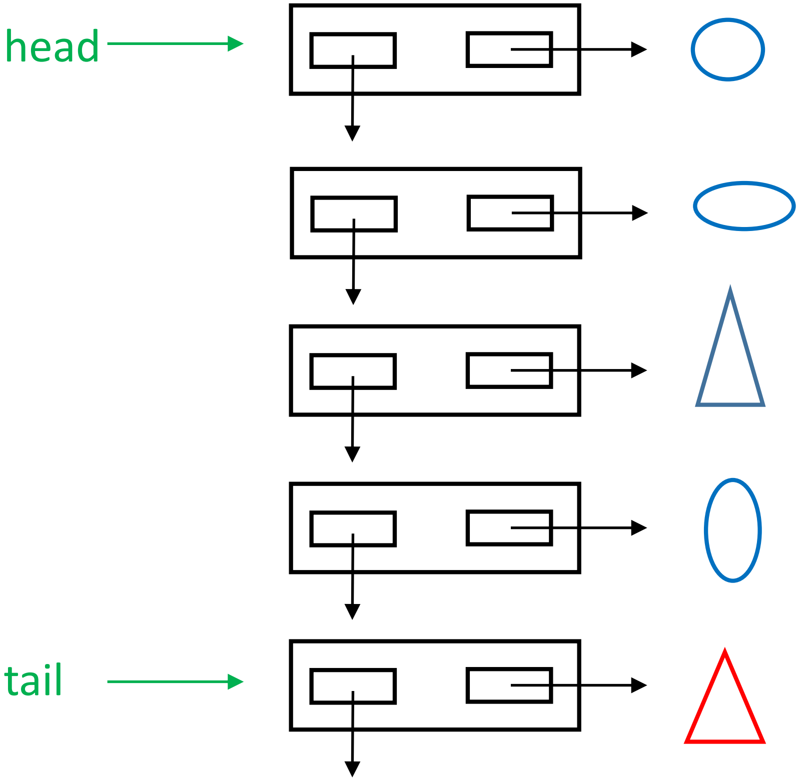
*But it is O(N) if array is full

addLast (△)

BEFORE



AFTER



addLast (e)

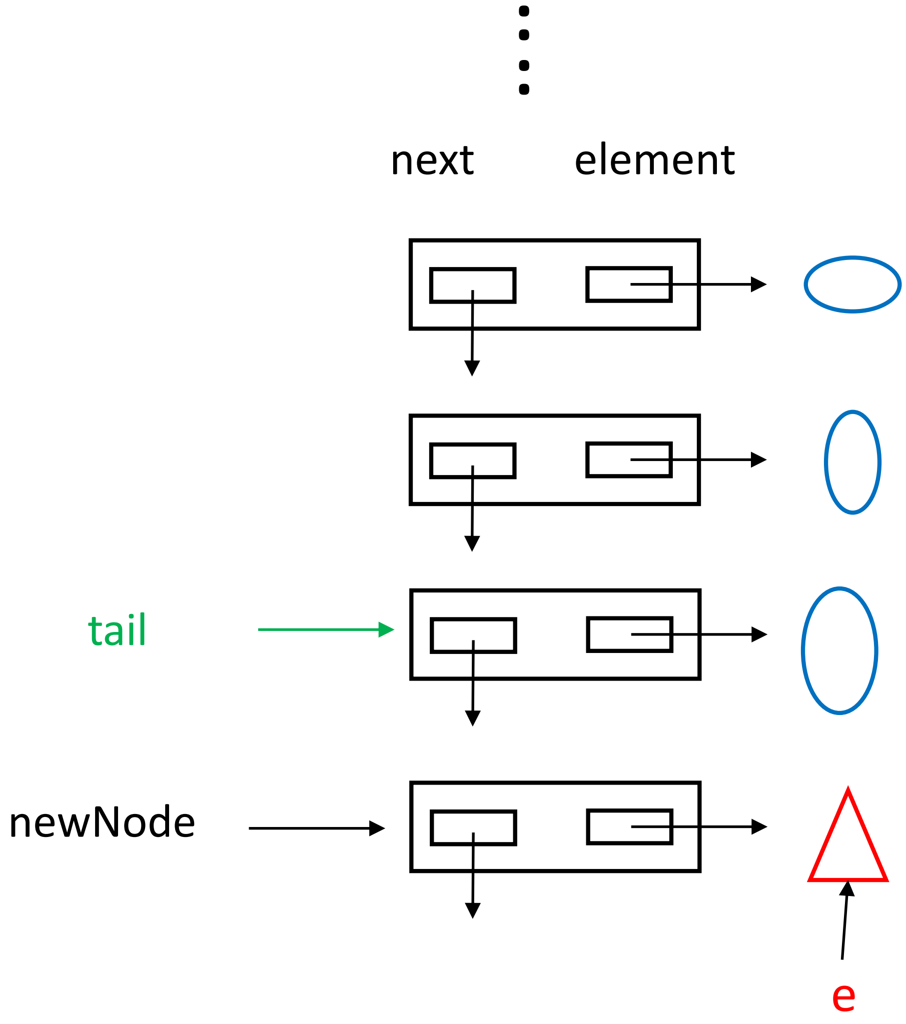
newNode = new Node

newNode.element = e

tail.next = newNode

tail = tail.next

size = size+1



addLast (e)

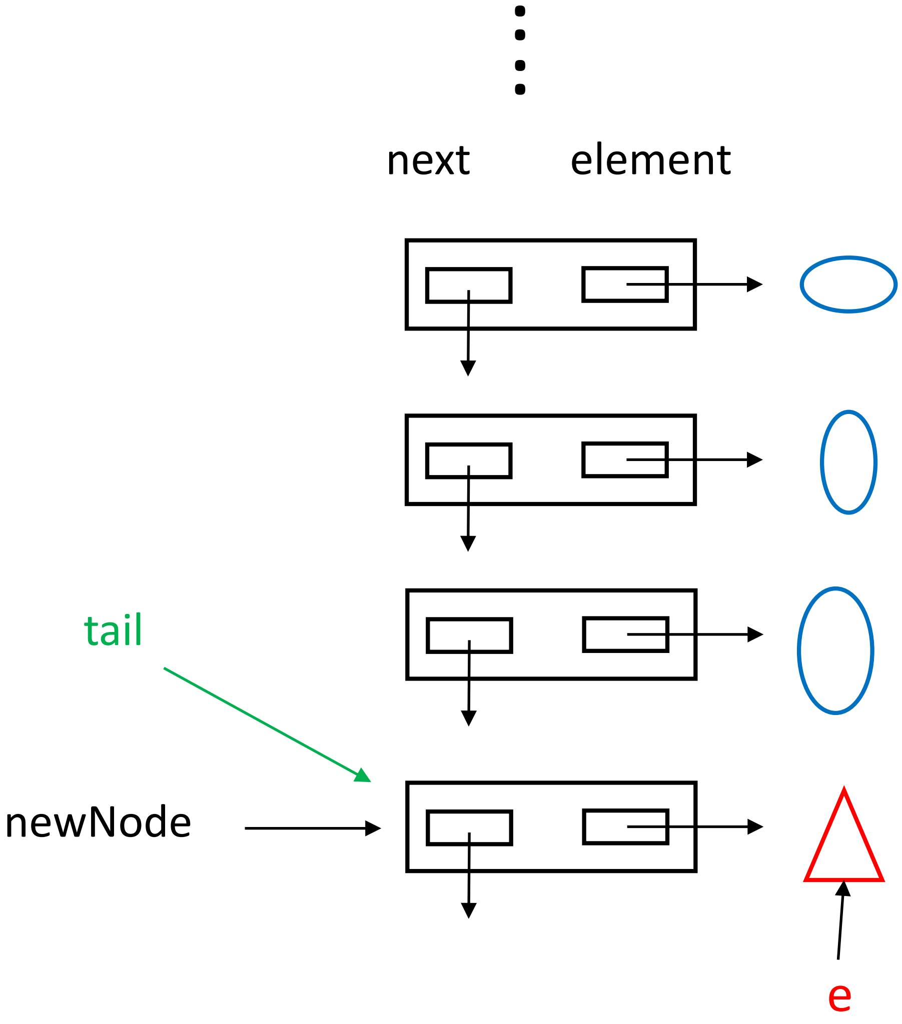
newNode = new Node

newNode.element = e

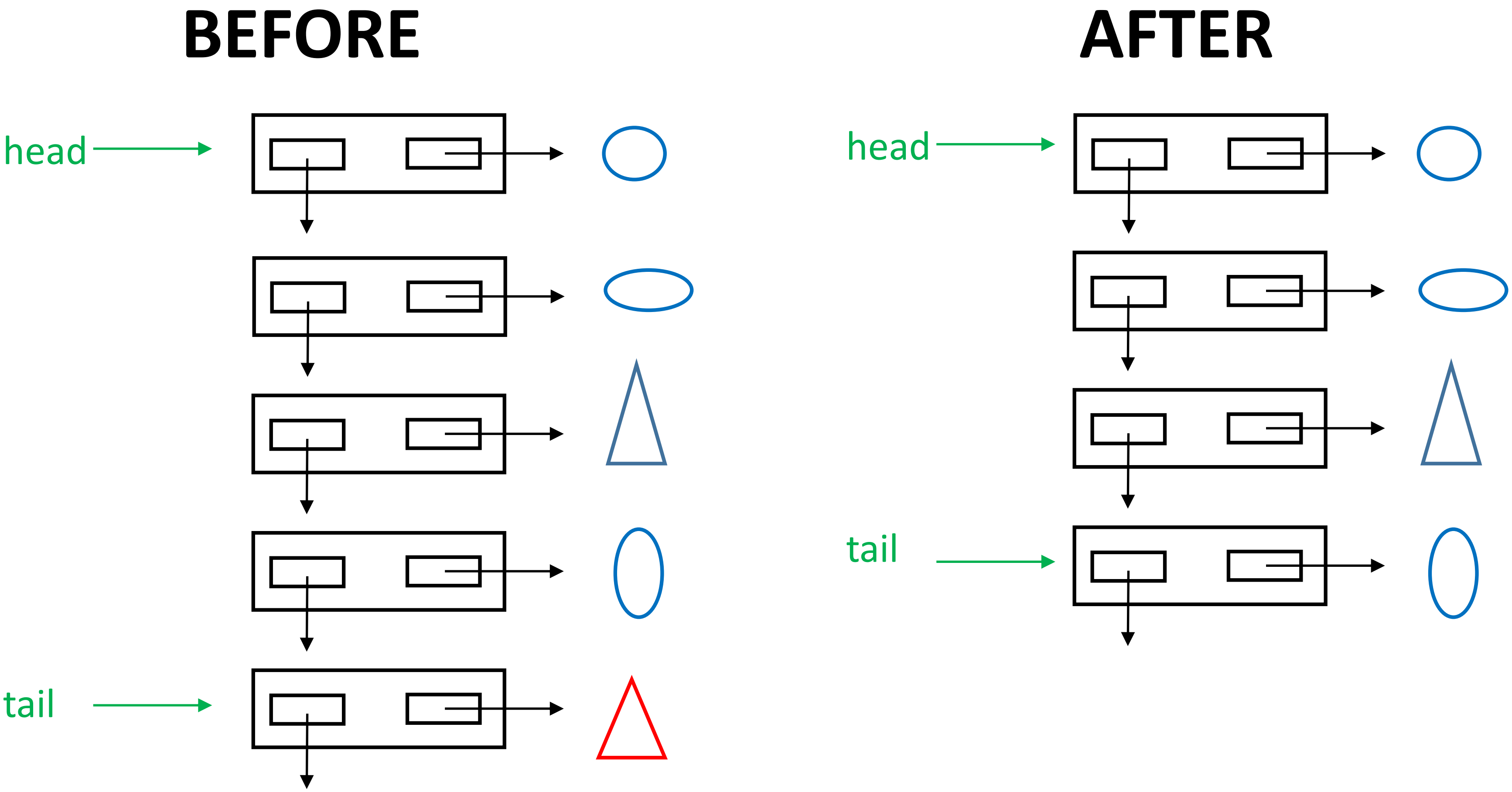
tail.next = newNode

tail = tail.next

size = size+1



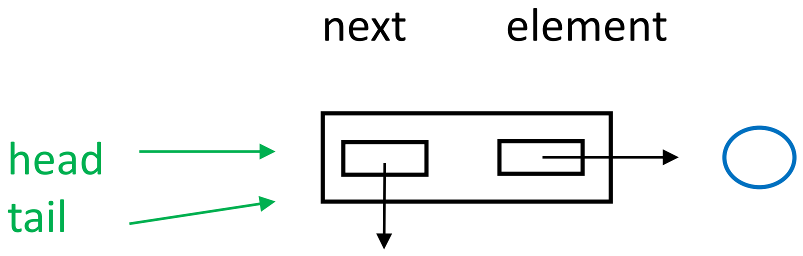
What about removeLast ( ) ?



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

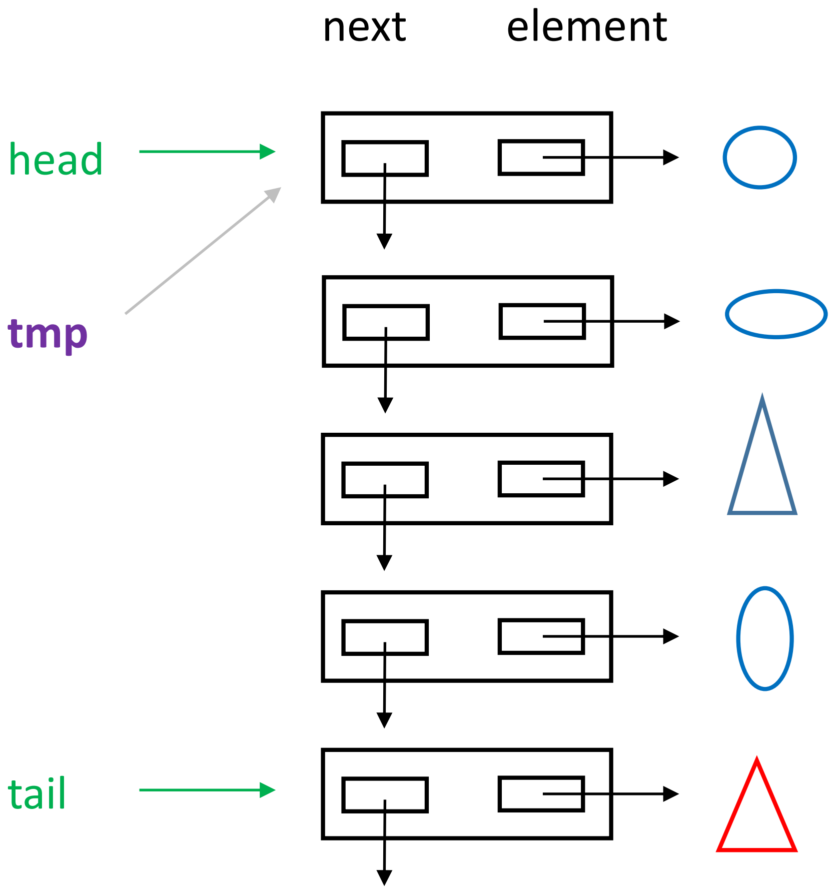
removeLast (   )

```
if  (size == 1){
     head = null
     tail     = null
}
else {

    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
```



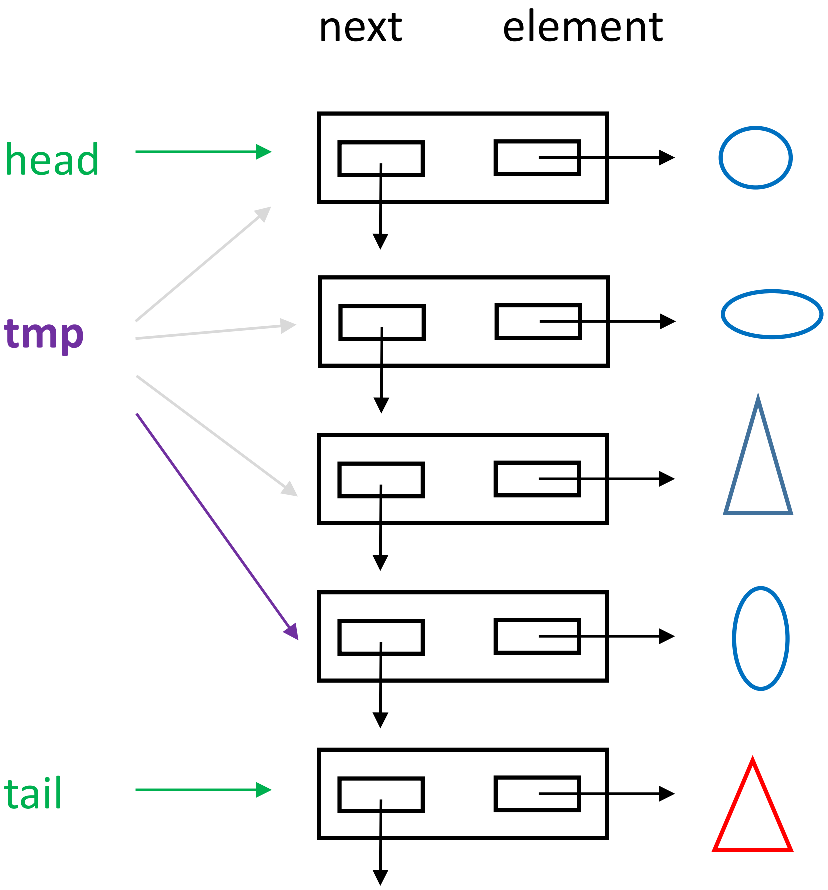
removeLast (   )

```
if  (size == 1){
     head = null
     tail     = null
}
else {

    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
```



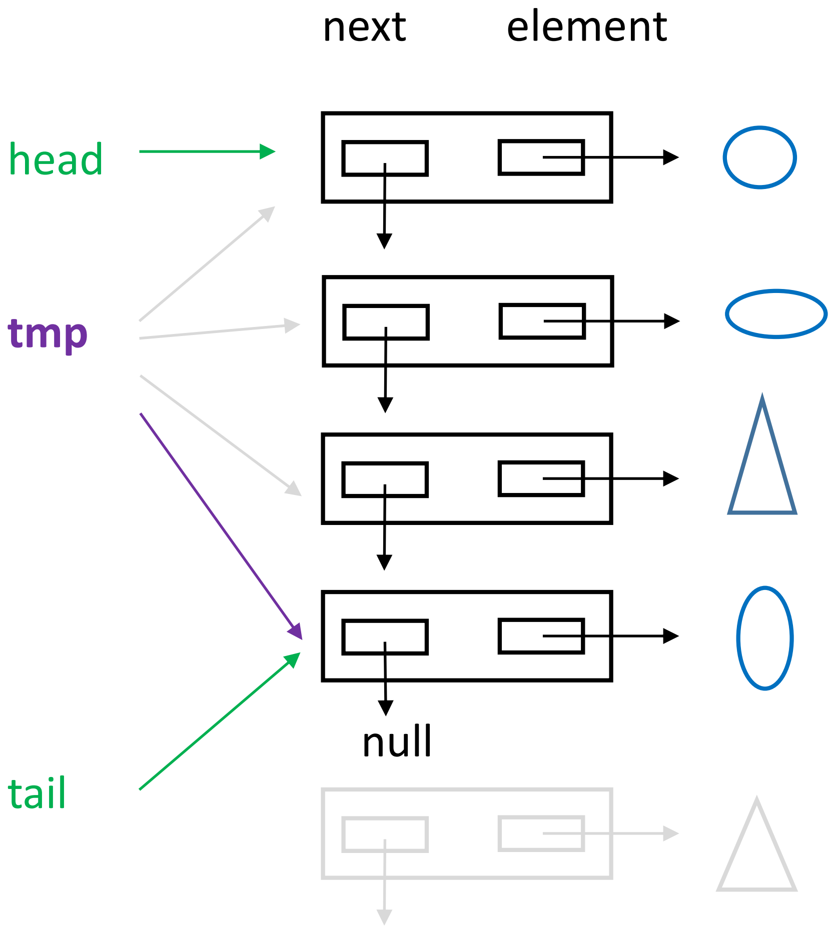
removeLast (   )

```
if  (size == 1){
    head = null
    tail     = null
}
else {

   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
```



removeLast (   )

```
if  (size == 1){
     head = null
     tail     = null
}
else {

    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
```



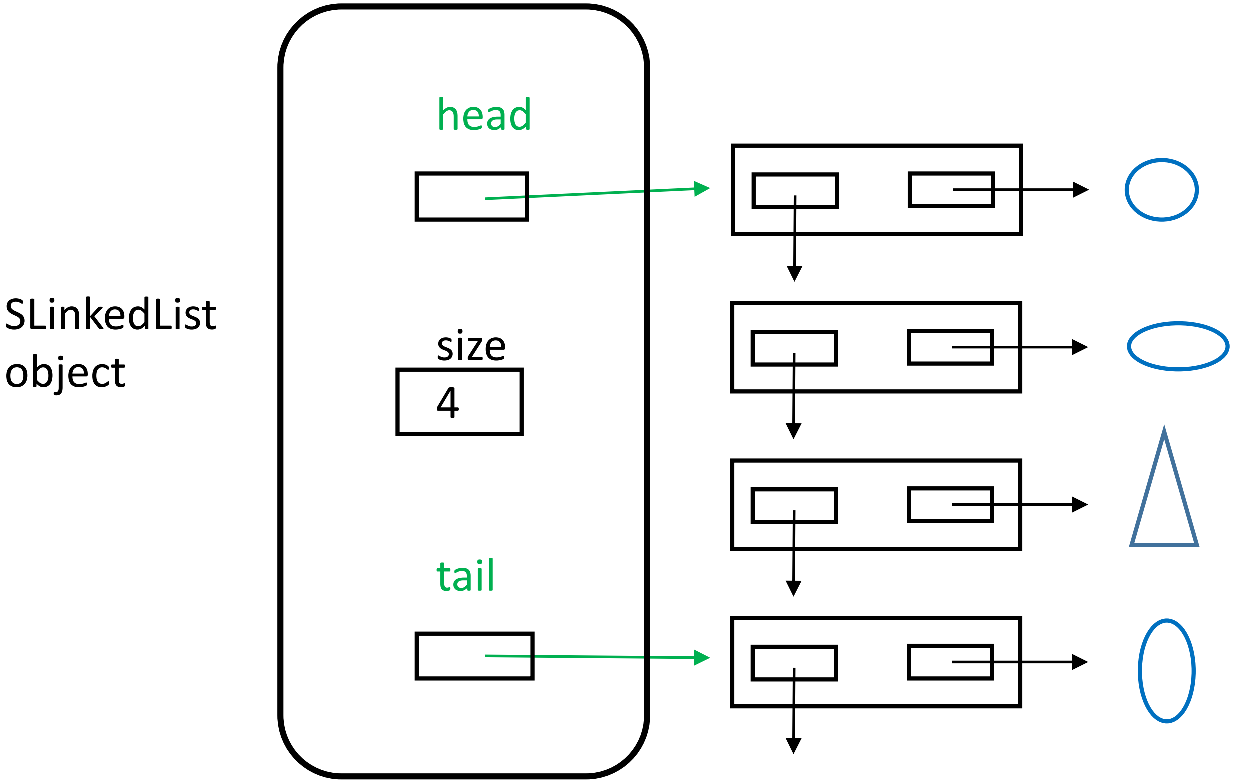
Time Complexity (N = list size)

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

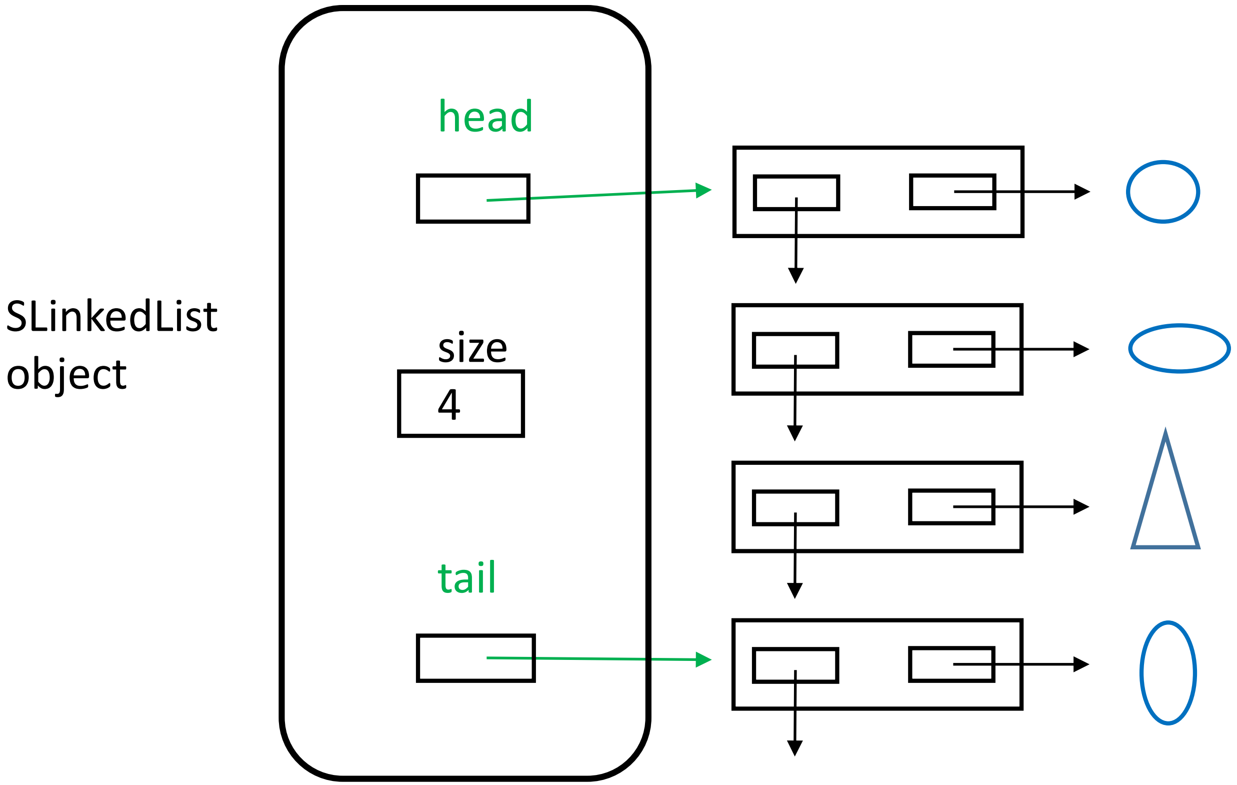
*O(N) if array is full

```
class   SLinkedList<E> {

    SNode<E>   head;
    SNode<E>   tail;
    int        size;

     :    //   various methods

     private   class   SNode<E> {        // inner class

            SNode<E>   next;
            E          element;
             :
     }
}
```

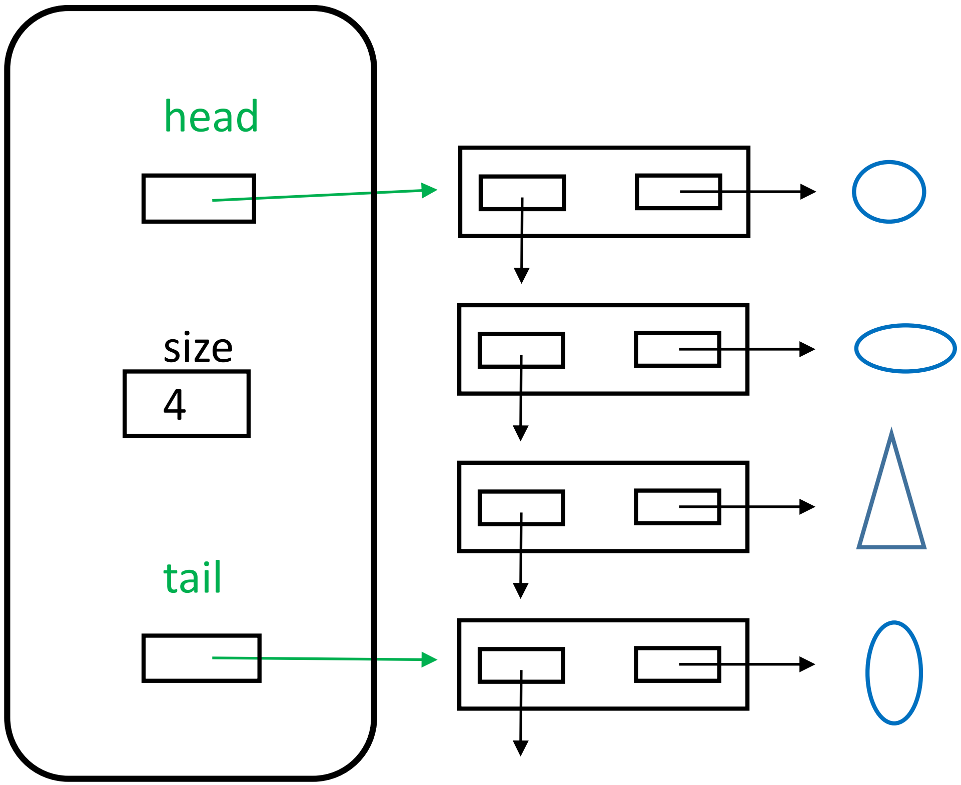
```
class   SLinkedList<E> {

        SNode<E>   head;
        SNode<E>   tail;
        int        size;

         :
}
```



How many objects?



How many objects?



1 + 4 + 4 = 9

SLinkedList SNode Shape

Reminders

- **Quiz 1** today until 8 pm

- **IDE Tutorials (Trottier 3120)**
  - Friday, Sept 28 2pm to 3:30pm (Eclipse)
  - Saturday, Sept 29 12pm to 1:30pm (IntelliJ)
  - Monday, Oct 1 4pm tio 5:30pm (Eclipse)
  - Wednesday, Oct 3 4:30pm to 6pm (IntelliJ)
  - Thursday Oct 4 4:30pm to 6pm (Eclipse)
  - Friday Oct 5 4pm to 5:30pm (IntelliJ)