Implementation
ArrayList

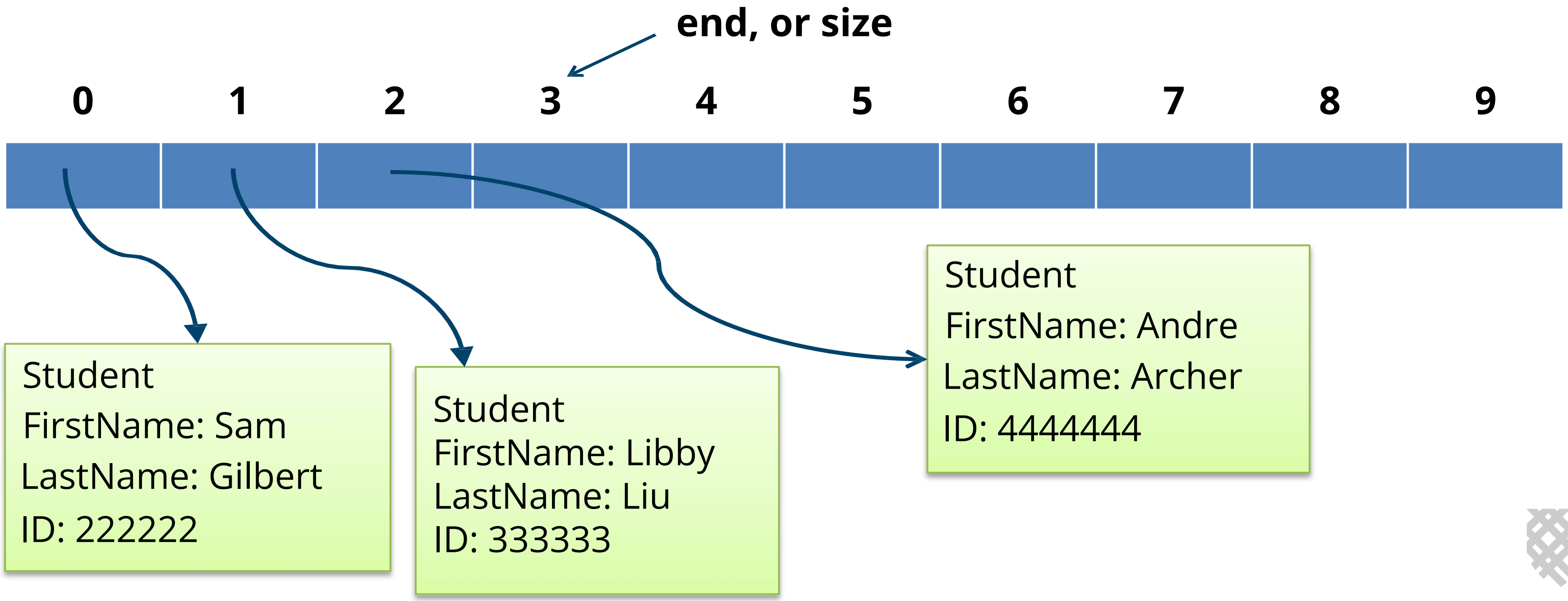
COMP128 Data Structures

The List Interface

`Boolean` **`add`**`(E e)`	Appends the specified element to the end of this list.
`void` **`add`**`(int index, E element)`	Inserts the specified element at the specified position in this list.
`E` **`get`**`(int index)`	Returns the element at the specified position in this list.
`Int` **`indexOf`**`(Object o)`	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`E` **`remove`**`(int index)`	Removes the element at the specified position in this list.
`Boolean` **`remove`**`(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.

ArrayList Implementation

The ArrayList class has:
An array that contains references to Objects, such as instances of type Student
An int that keeps track of the end of the list or the next available slot



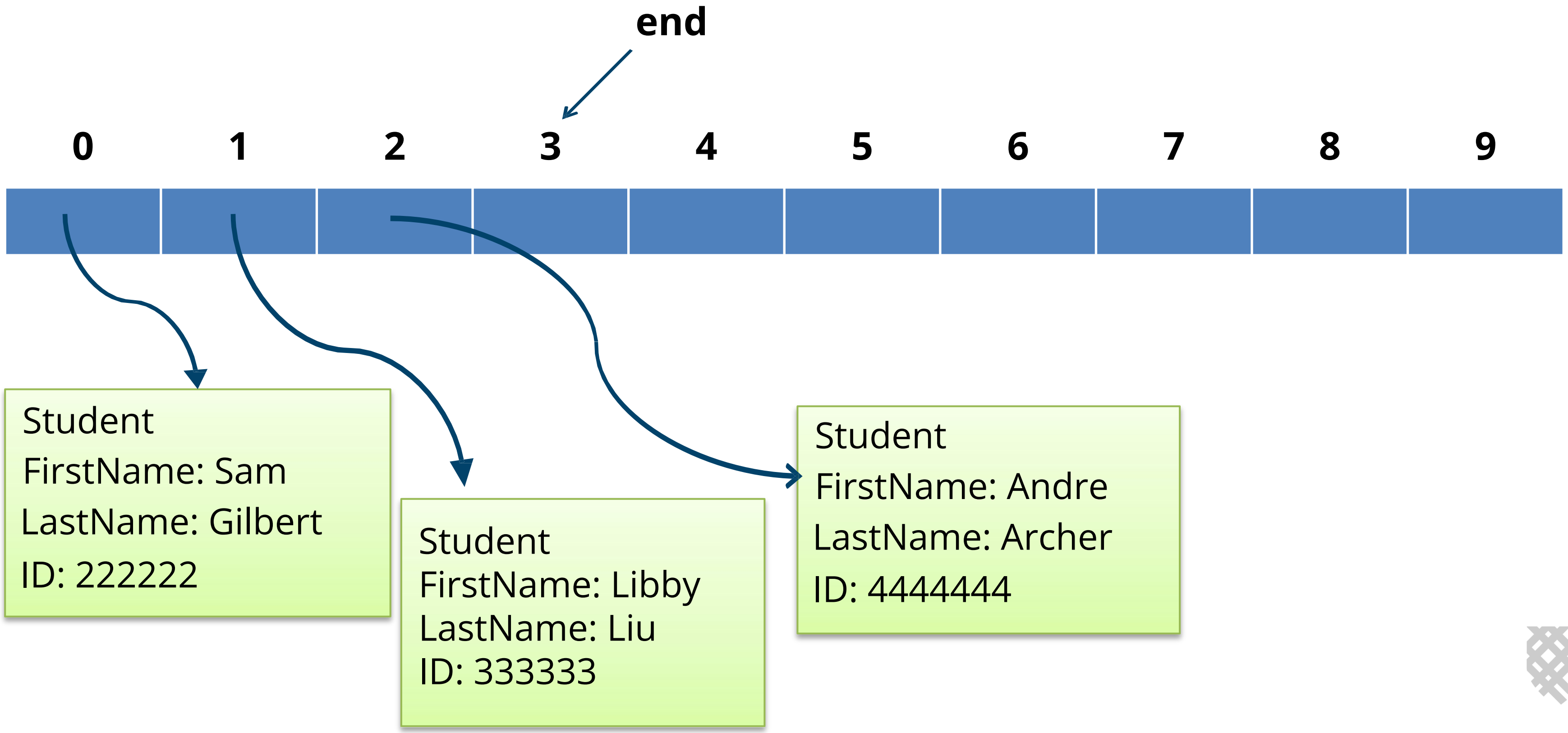
ArrayList Implementation

```
//This class implements some of the methods of the Java ArrayList class.
public class ArrayList<T> implements ListADT<T> {
 private final static int DEFAULT_CAPACITY = 100;
 protected int rear;
 protected T[] list;

// Creates an empty list using the default capacity.
public ArrayList() {
    this(DEFAULT_CAPACITY);
}

//Creates an empty list using the specified capacity.
public ArrayList(int initialCapacity) {
   rear = 0;
   list = (T[])(new Object[initialCapacity]);
  }
```

Operation Runtime

- For lists, these operations are usually either O(1), where we access one element to get what we need done, or O(N), where in the worst case we need to access all N elements to complete the operation.
- **It depends on the implementation.**
- We describe it in terms of the number of elements in the list, thinking about adding up how many times we would have to access an element in the list to complete the operation

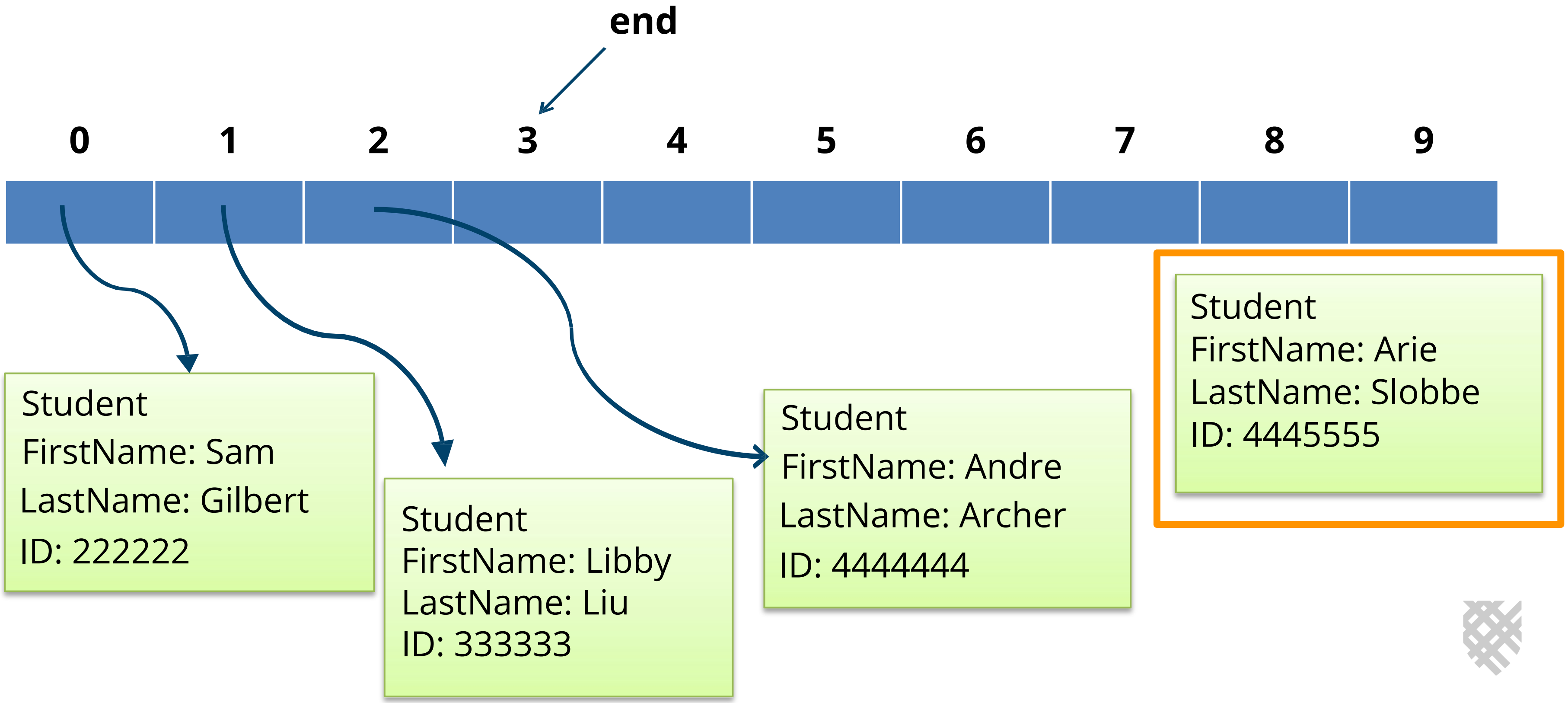
boolean **add**(E e)



boolean **add**(E e)

1. Create new object

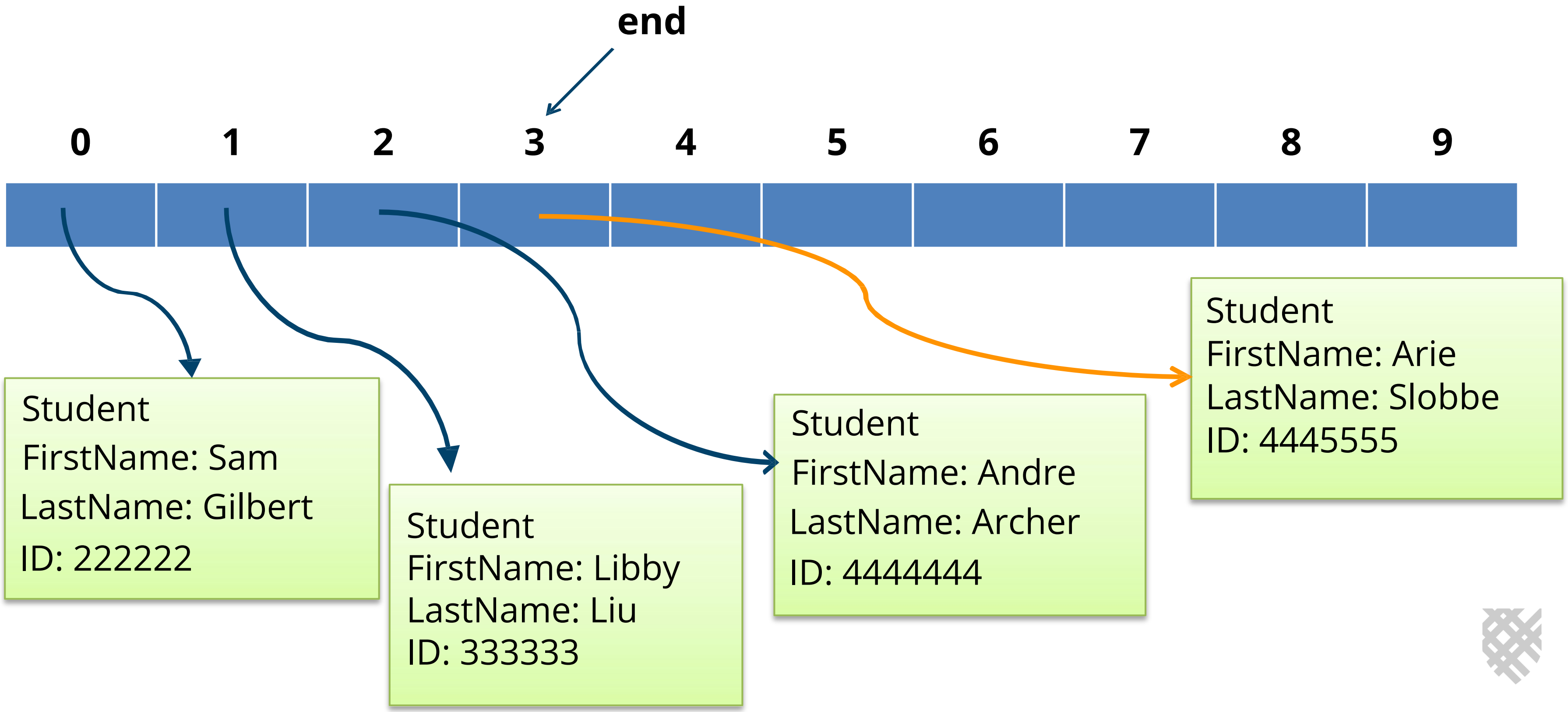
```
Student arie = new Student("Arie",
"Slobbe", 4445555);
```



boolean **add**(E e)

2. Set A[end] to point to object

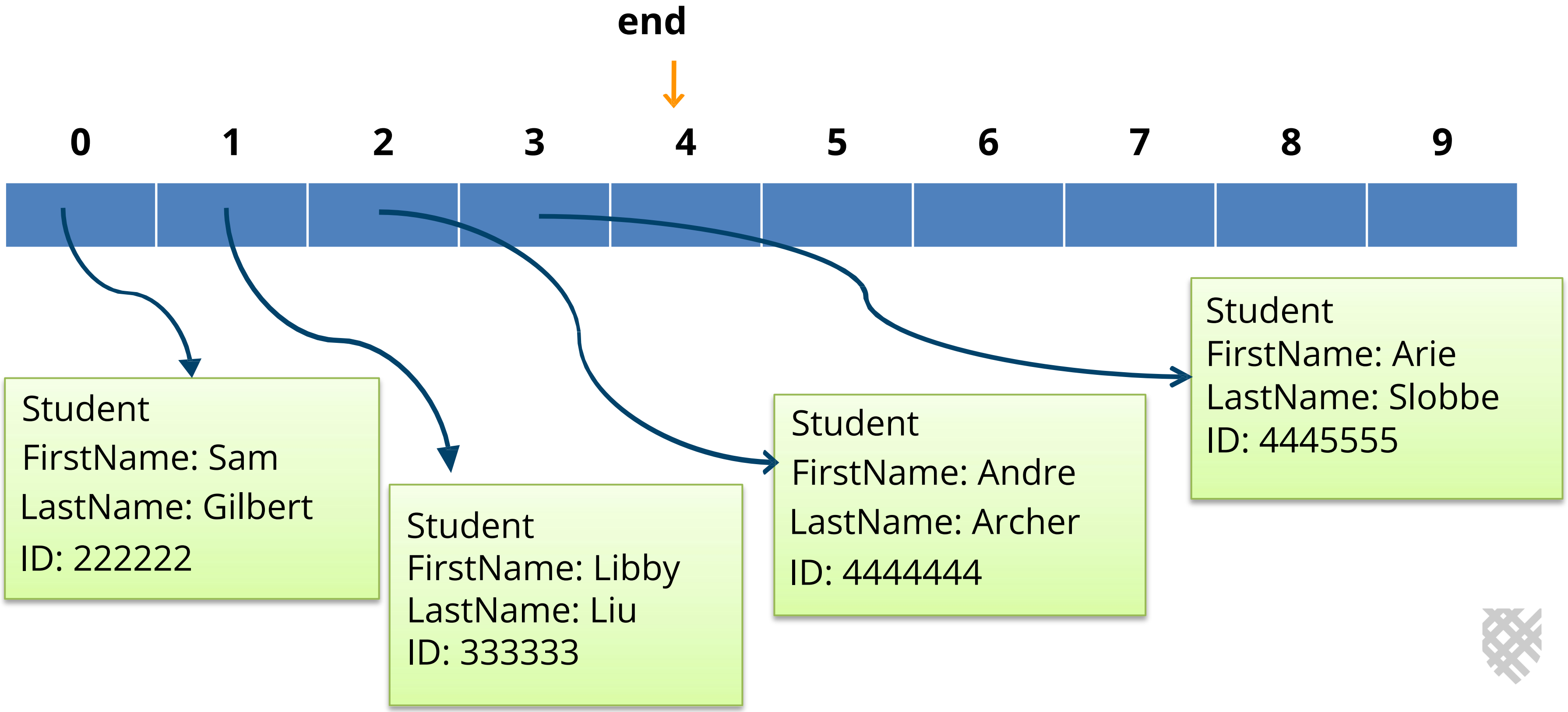
```
A[end] = arie;
```



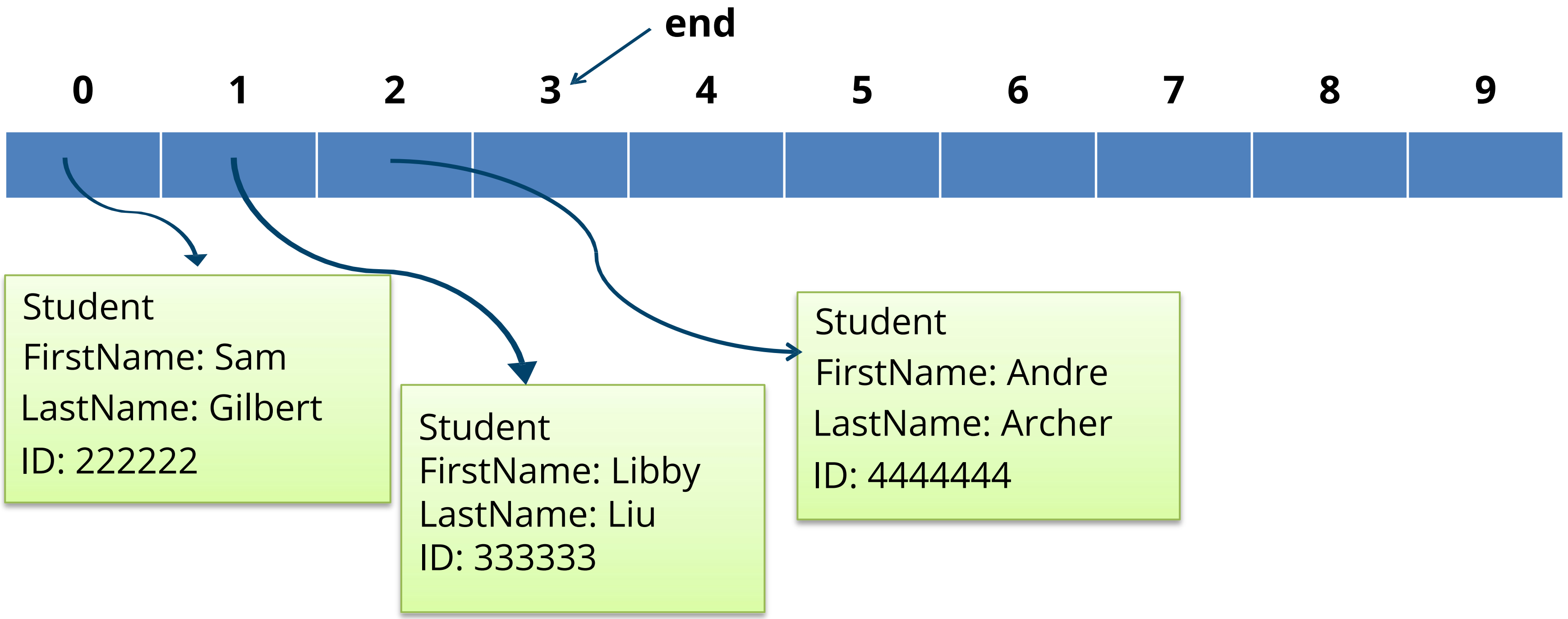
boolean **add**(E e)

3. Increment end

```
end++;
```



void **add**(int index, E element)

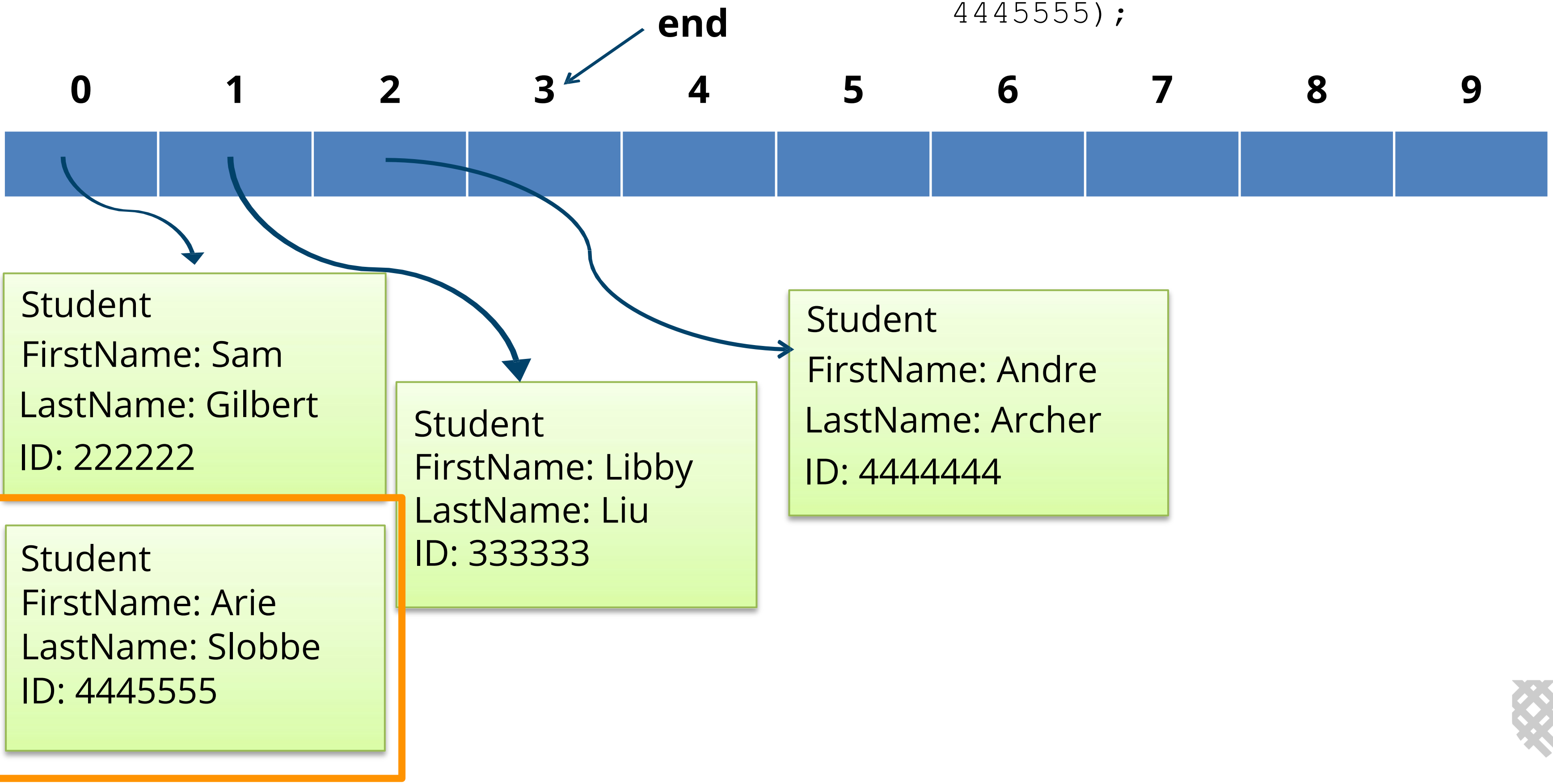


Suppose we are going to add at the front, or index = zero

void **add**(int index, E element)

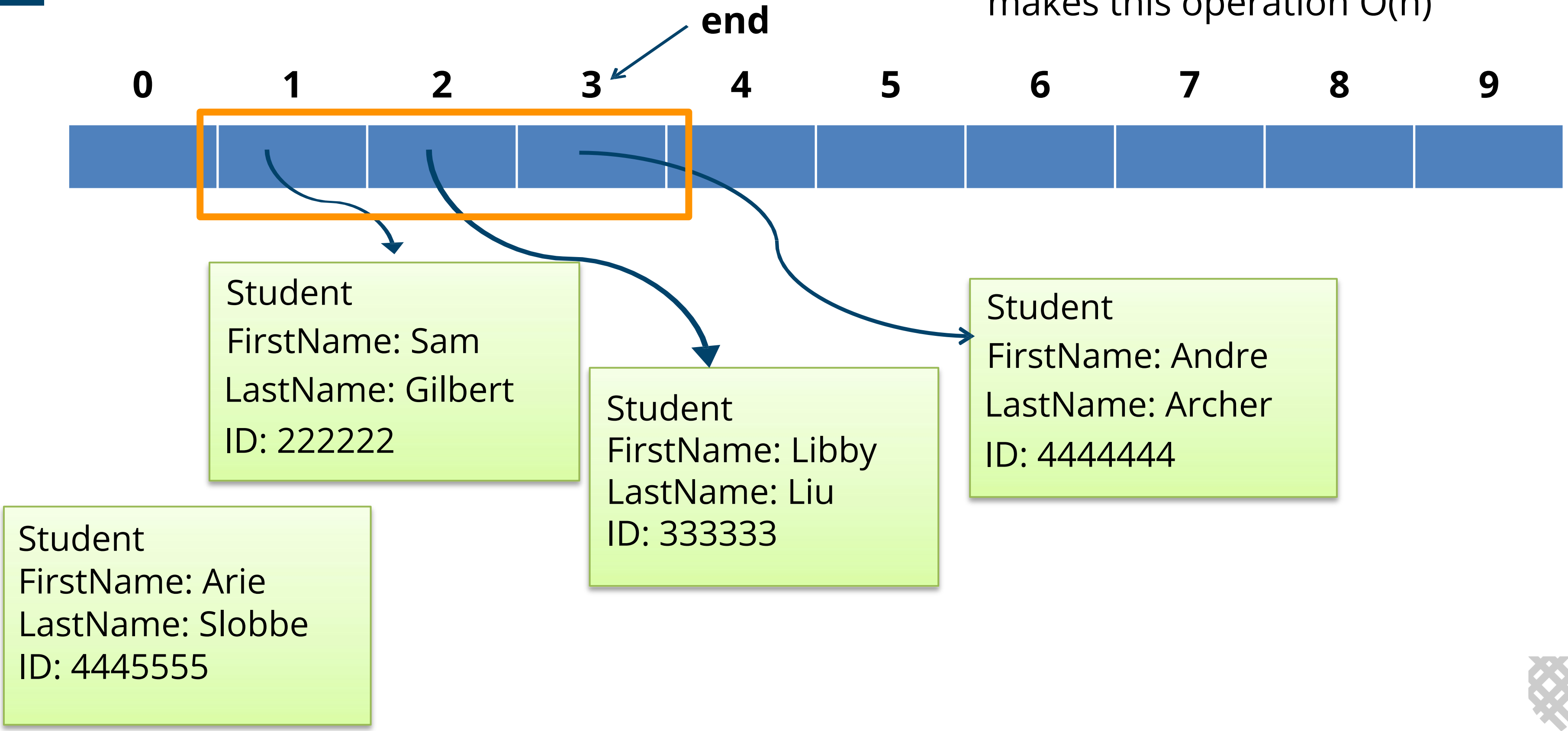
1. Create new object

```
Student arie = new
Student("Arie", "Slobbe",
4445555);
```



void **add**(int index, E element)

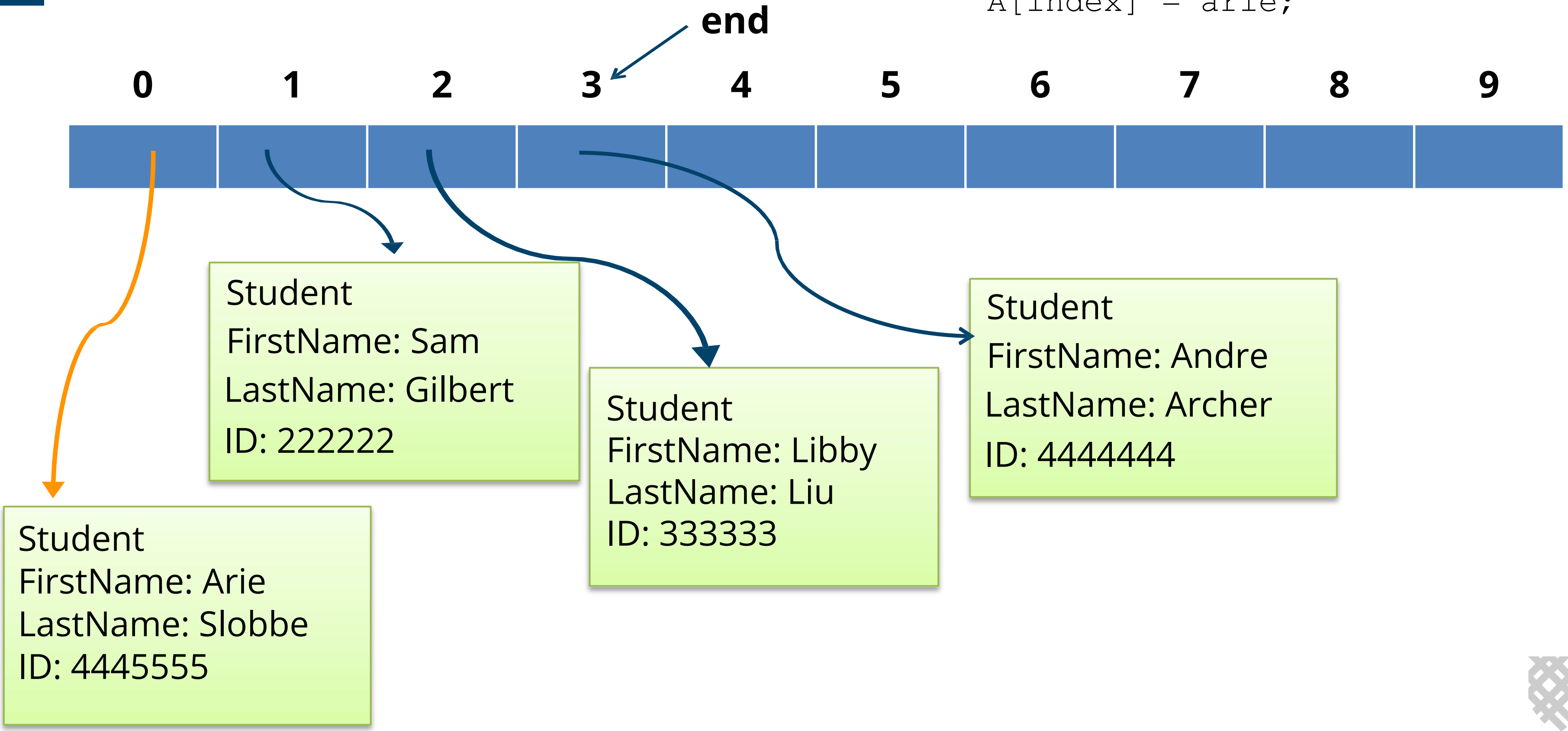
2. Move each element down; This makes this operation O(n)



void **add**(int index, E element)

3. Set A[end] to point to object

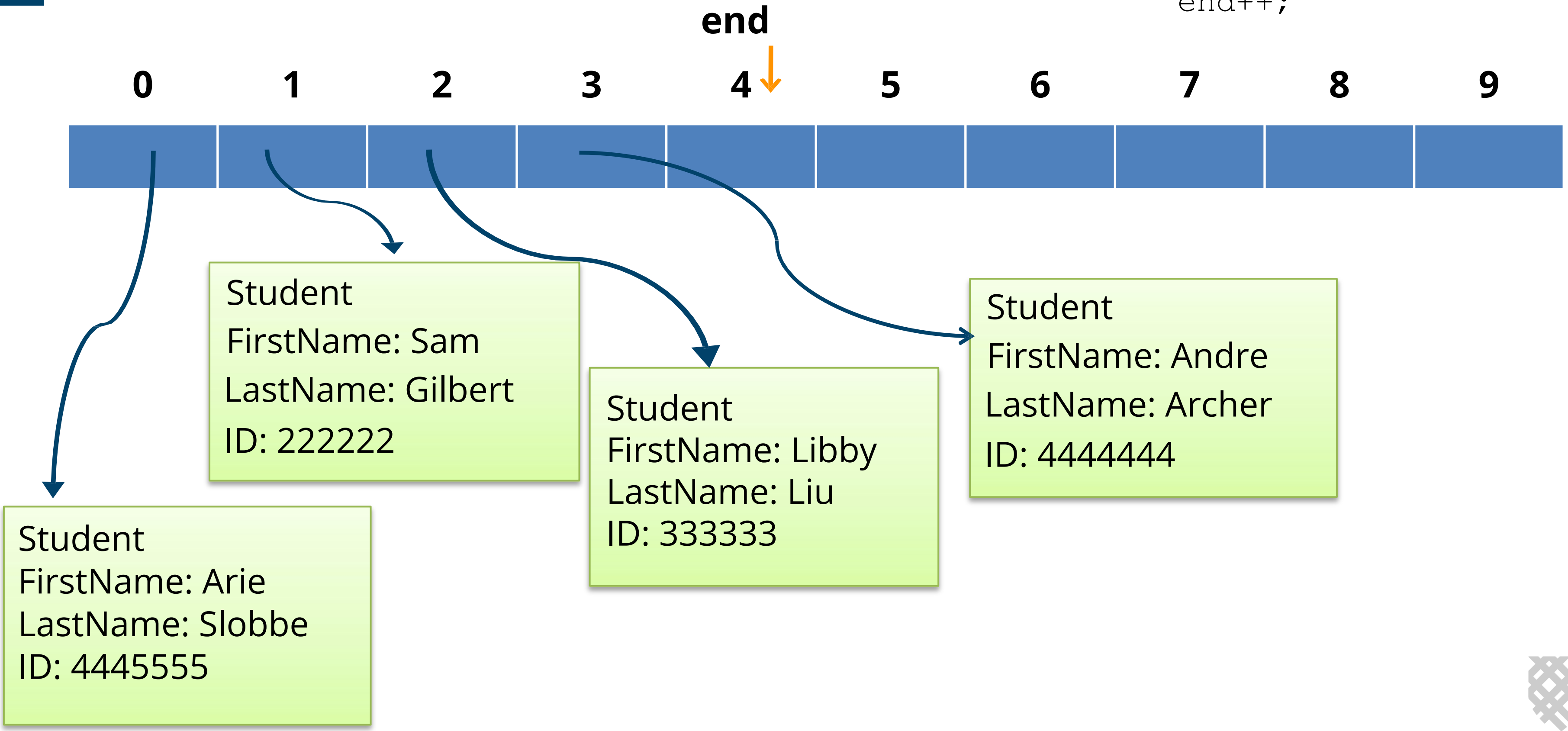
```
A[index] = arie;
```



void **add**(int index, E element)

4. Increment end

```
end++;
```



At worst, how long do these take?

Operation	**Runtime**
`Boolean` **`add`**`(E e)`	O(1)
`void` **`add`**`(int index, E element)`	O(n)
`E` **`get`**`(int index)`	O(1)
`Int` **`indexOf`**`(Object o)`	O(n)
`E` **`remove`**`(int index)`	O(n)
`Boolean` **`remove`**`(Object o)`	O(n)

Using ArrayList

- In an application, what is the ideal situation for using ArrayLists?

Using ArrayList

- In an application, what is the ideal situation for using ArrayLists?
- What happens when we have reached capacity of the underlying array and we wish to add?

In-class Activity

ArrayList Implementation Activity