
Intro to C++



**OLD
MISSION
CAPITAL**



Introductions

- Who we are
 - proprietary trading (no customers or clients)
 - ~80 people in Chicago, New York, London
- Who are we
 - Andrew Wonnacott — Developer — Princeton COS '19
 - Janine Manolakos — Senior Trader — Brown Math+Econ '01
 - Kyle Gubbins — Head of Recruiting



Introductions

- What we do
 - Securities arbitrage (our trades are observations of mispricings, not term-based predictions of price movement)
 - ETFs, equity, commodity, bonds, fx, futures & options, etc...
 - essentially any electronic market with central clearing
- Who we're looking for
 - Summer 2020 interns and full-time positions
 - Talented software developers who want to build systems
 - Awareness of the forest while building the trees
 - Also hiring traders & quants



Java quiz!

```
int i = 3;
```

```
int j = i;
```

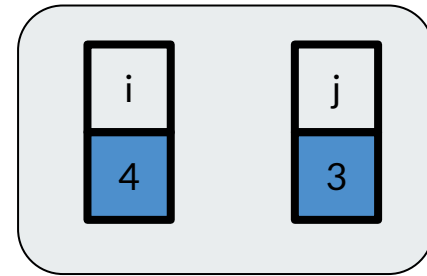
```
i++;
```

```
System.out.println(j);
```



Java quiz!

```
int i = 3;  
int j = i;  
i++;  
  
System.out.println(j); // 3
```





Java quiz!

```
ArrayList<String> l = new ArrayList<String>();
```

```
ArrayList<String> m = l;
```

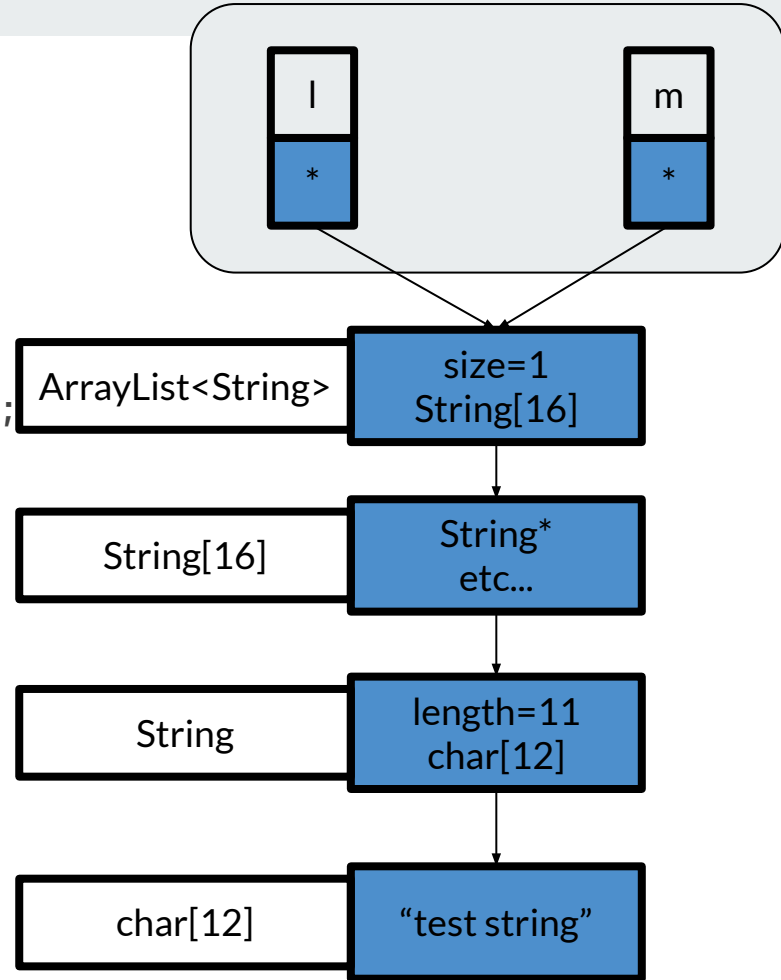
```
l.add("test string");
```

```
System.out.println(m);
```



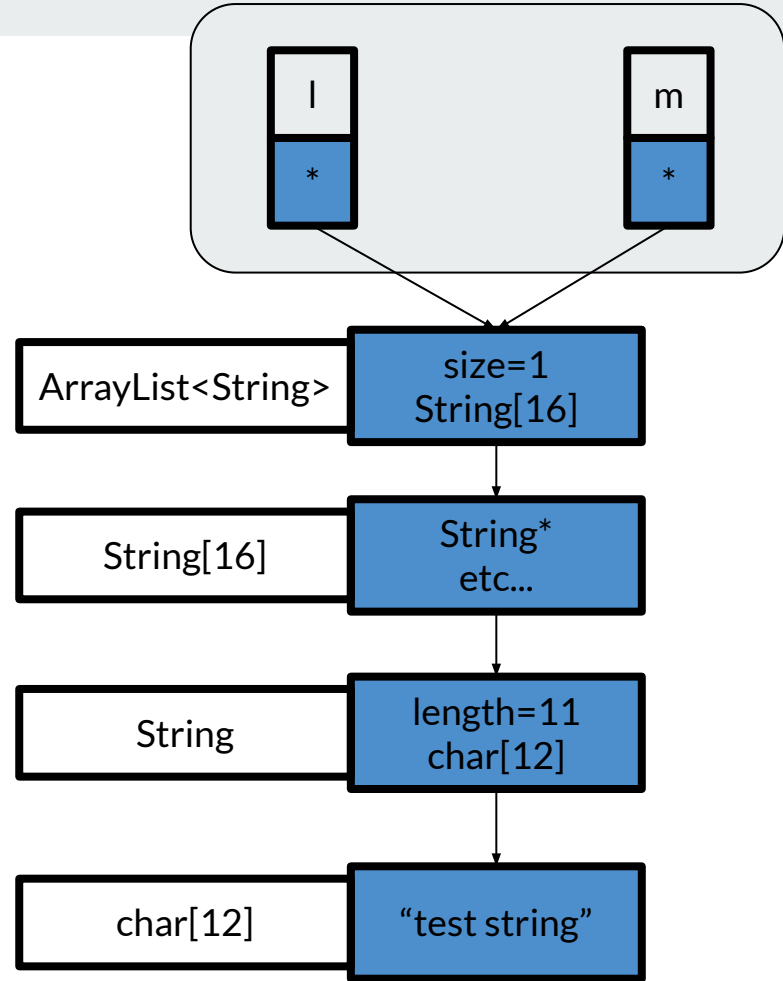
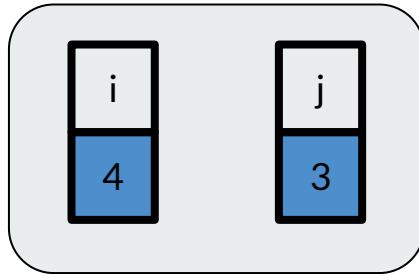
Java quiz!

```
ArrayList<String> l = new ArrayList<String>();  
ArrayList<String> m = l;  
l.add("test string");  
System.out.println(m); // ["test string"]
```





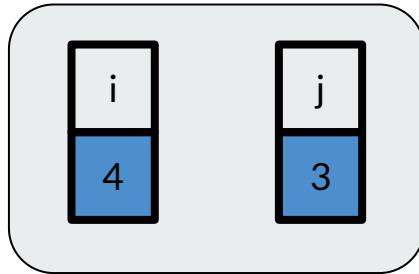
References vs values



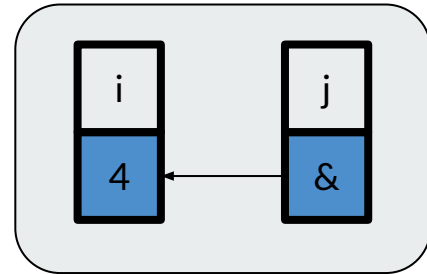


References vs values: primitives, idea

```
int i = 3;  
int j = i;  
i++;  
println(j);
```



```
int i = 3;  
int& j = i;  
i++;  
println(j);
```





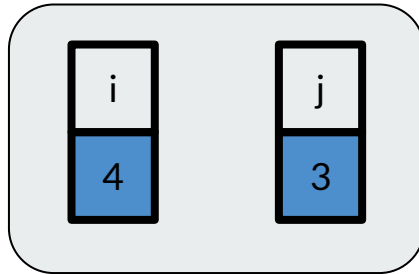
References vs values: primitives, C++

```
int i = 3;
```

```
int j = i;
```

```
i++;
```

```
cout << j << endl; // 3
```

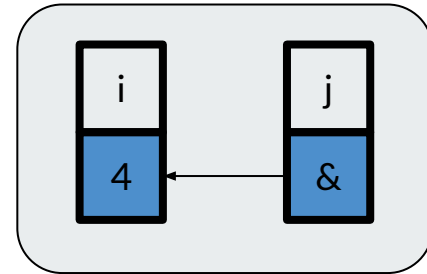


```
int i = 3;
```

```
int& j = i;
```

```
i++;
```

```
cout << j << endl; // 4
```





References vs values: objects

```
array_list<string> l =  
array_list<string>();
```

```
array_list<string> m = l;
```

```
l.add("test string");
```

```
cout << m; // []
```

```
array_list<string> l =  
array_list<string>();
```

```
array_list<string>& m = l;
```

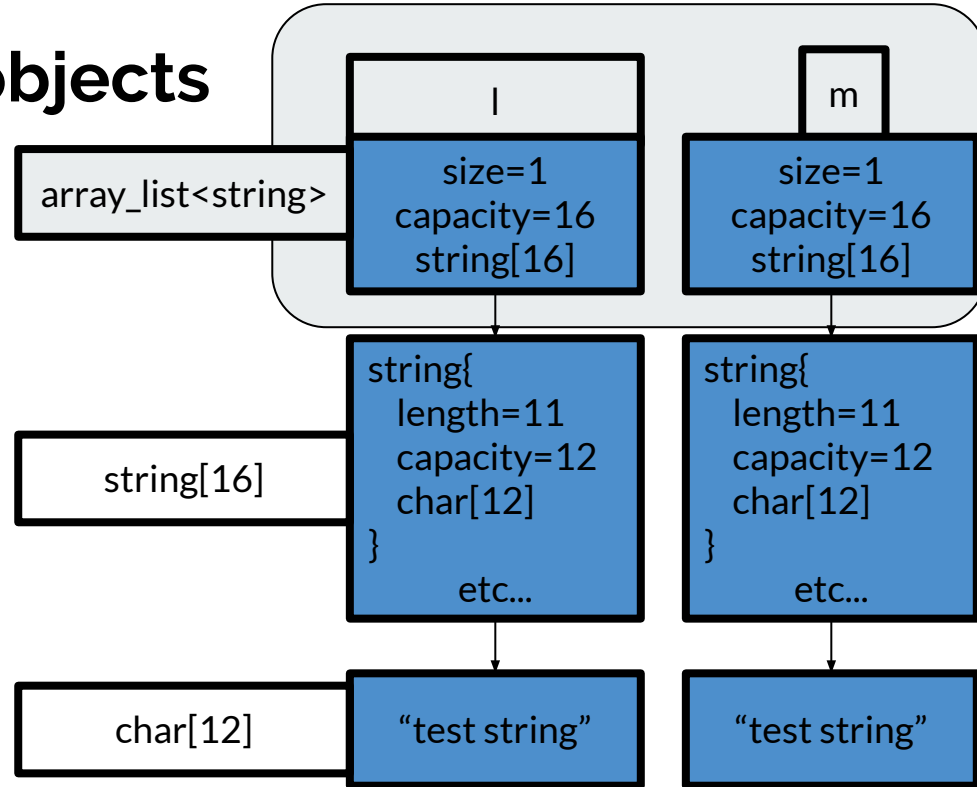
```
l.add("test string");
```

```
cout << m; // ["test string"]
```



References vs values: objects

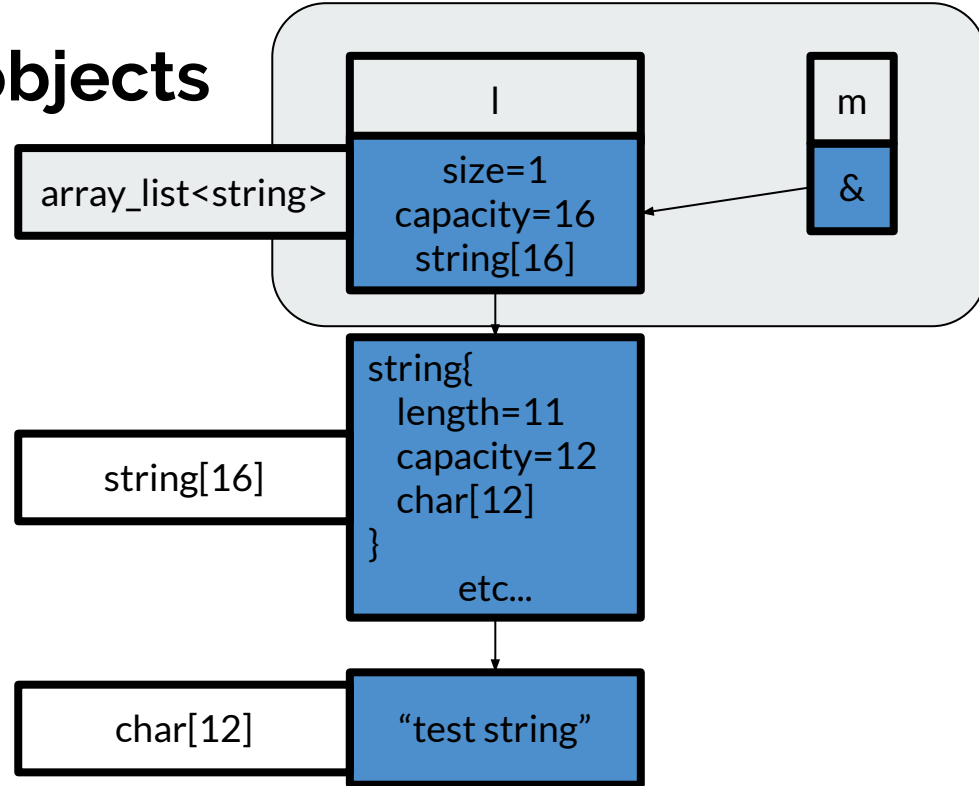
```
array_list<string> l =  
array_list<string>();  
  
array_list<string> m = l;  
  
l.add("test string");  
  
cout << m;
```





References vs values: objects

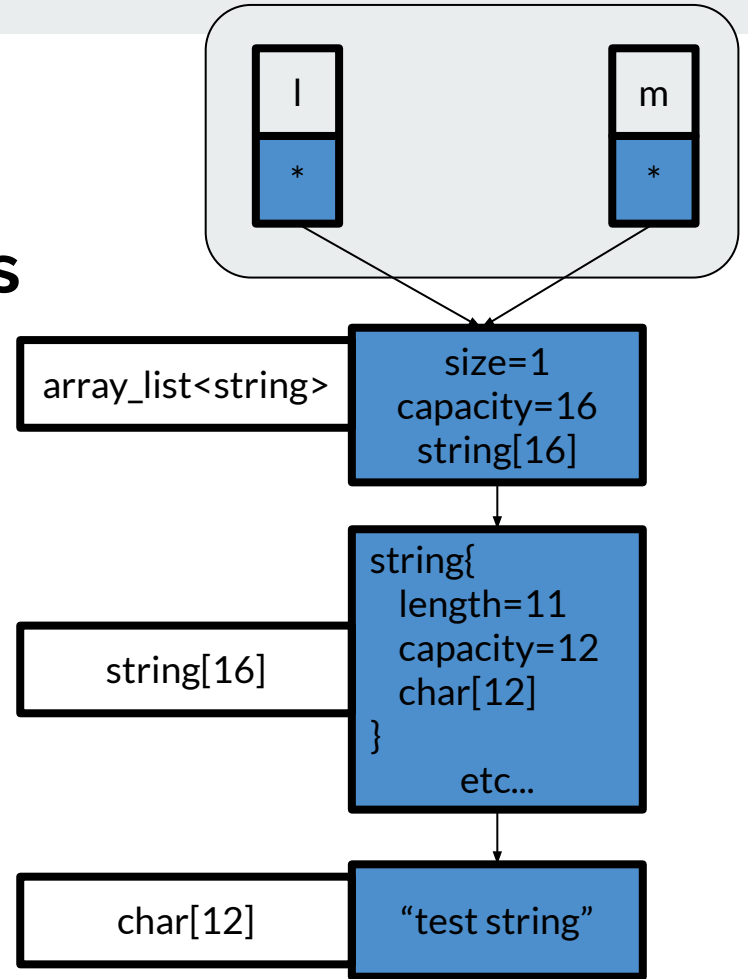
```
array_list<string> l =  
array_list<string>();  
  
array_list<string>& m = l;  
  
l.add("test string");  
  
cout << m;
```





References vs values: objects

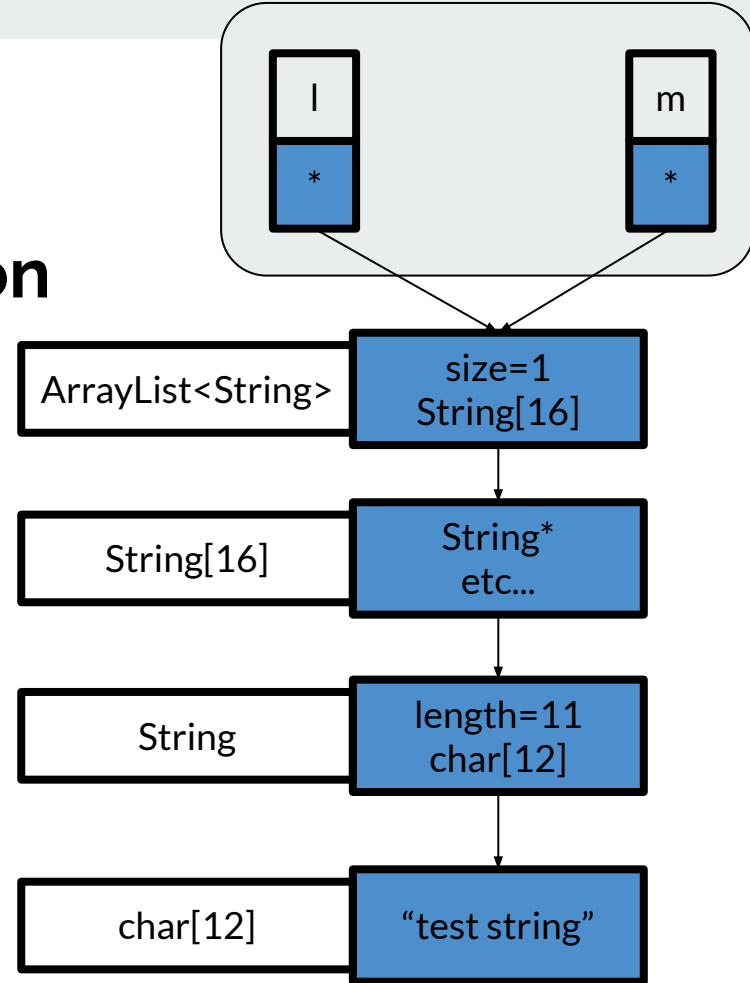
```
array_list<string>* l = new  
array_list<string>();  
  
array_list<string>* m = l;  
  
(*l).add("test string");  
  
cout << *m;
```





Java quiz 2! Garbage collection

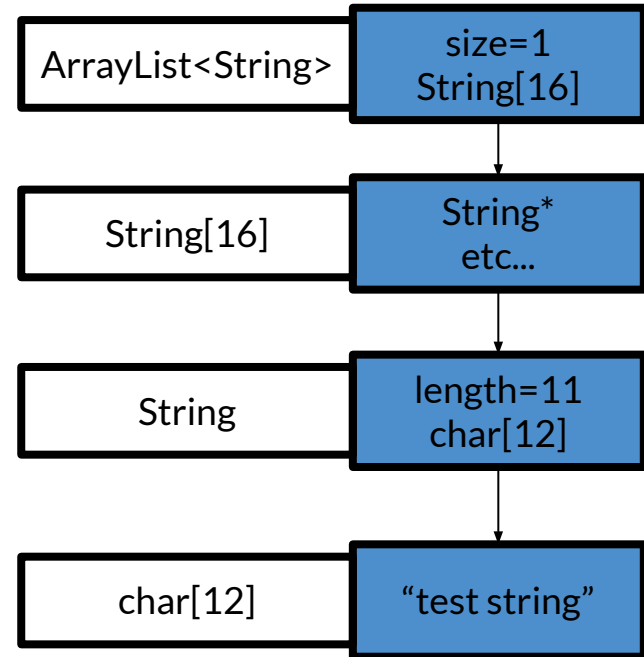
```
ArrayList<String> l = new ArrayList<String>();  
ArrayList<String> m = l;  
l.add("test string");  
System.out.println(m);  
return;
```





Java quiz 2! Garbage collection

```
ArrayList<String> l = new ArrayList<String>();  
ArrayList<String> m = l;  
l.add("test string");  
System.out.println(m);  
return;
```





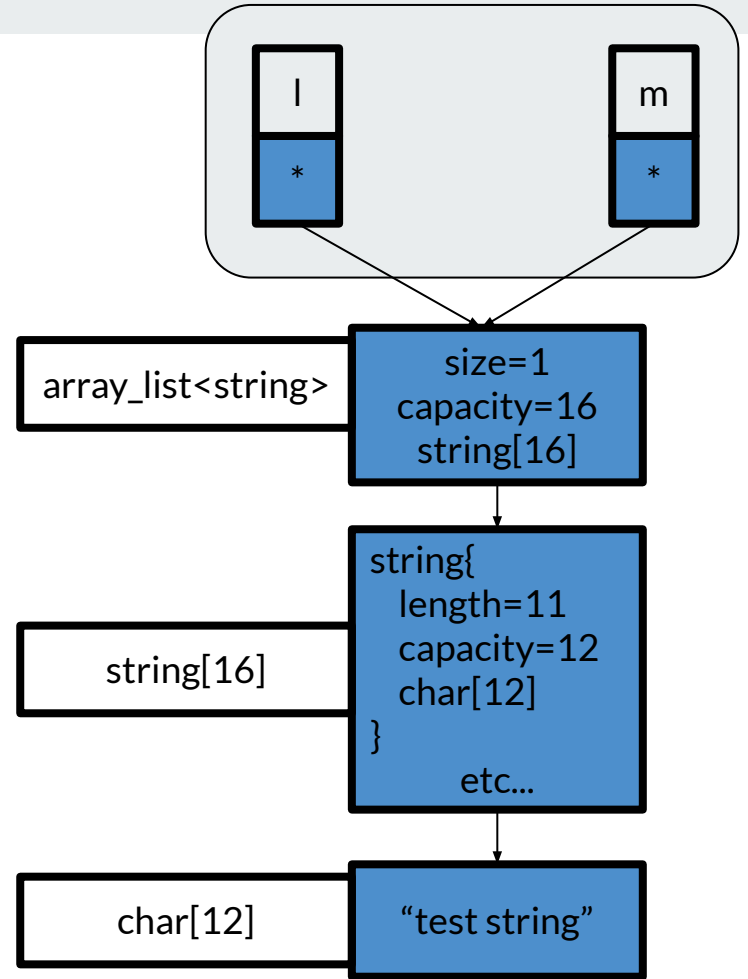
Java quiz 2! Garbage collection

```
ArrayList<String> l = new ArrayList<String>();  
  
ArrayList<String> m = l;  
  
l.add("test string");  
  
System.out.println(m);  
  
return;
```



C++ quiz! Garbage collection

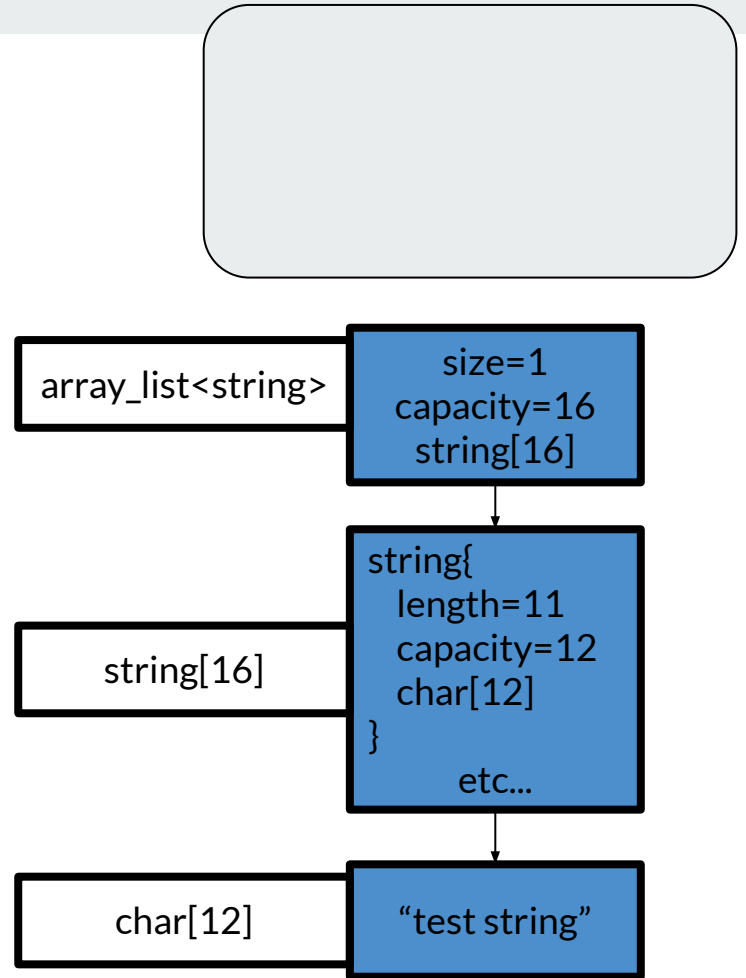
```
array_list<string>* l = new  
array_list<string>();  
  
array_list<string>* m = l;  
  
(*l).add("test string");  
  
cout << *m;  
  
return;
```





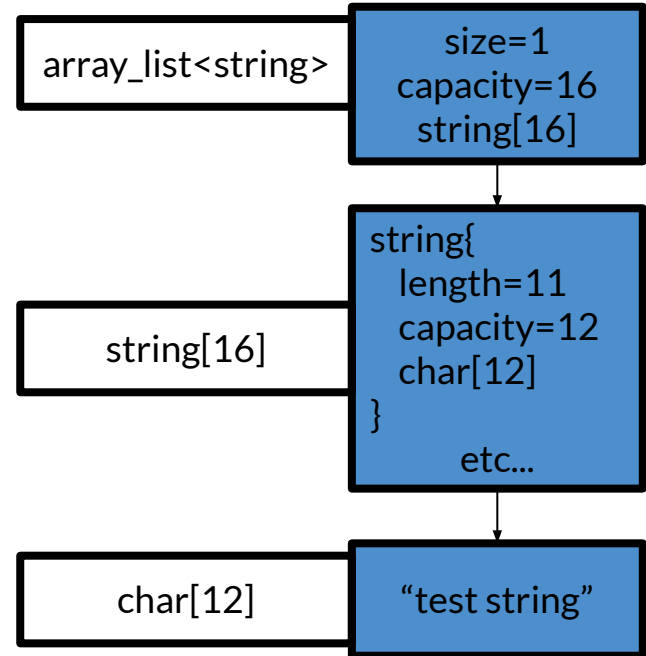
C++ quiz! Garbage collection

```
array_list<string>* l = new  
array_list<string>();  
  
array_list<string>* m = l;  
  
(*l).add("test string");  
  
cout << *m;  
  
return;
```





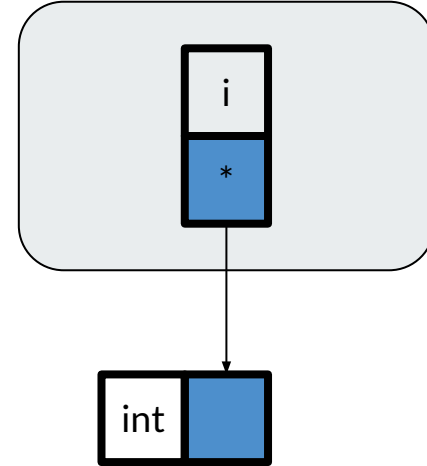
```
array_list<string>* l = new  
array_list<string>();  
  
array_list<string>* m = l;  
  
(*l).add("test string");  
  
cout << *m;  
  
return;
```





A simple class: C++ and Java

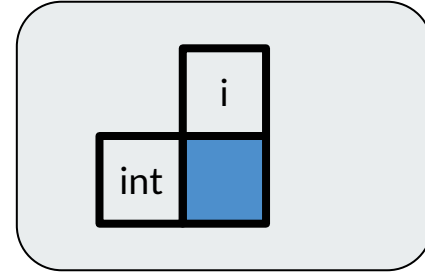
```
class Integer {  
    private int i;  
    public Integer(int i) {  
        this.i = i;  
    }  
    public int get() {  
        return i;  
    }  
}
```





A simple class: C++ and Java

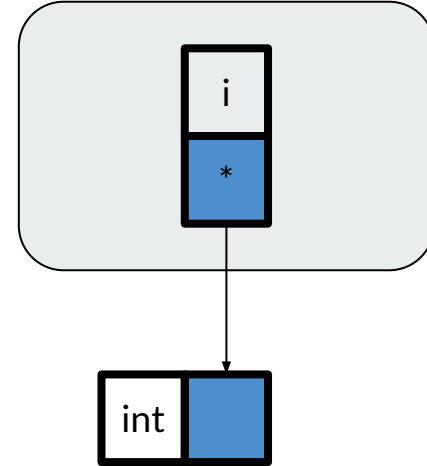
```
class Integer {  
private:  
    int _i;  
public:  
    Integer(int i) {  
        _i = i;  
    }  
    int get() {  
        return _i;  
    }  
}
```





A simple class: C++ and Java

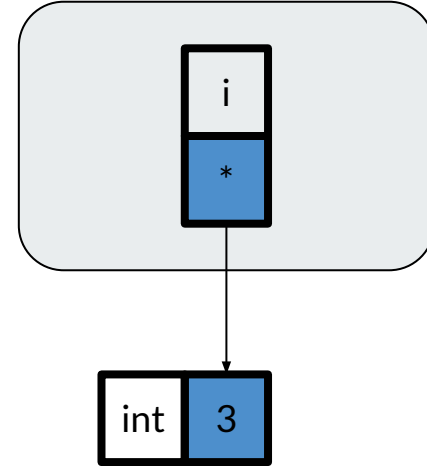
```
class Integer {  
private:  
    int* _i;  
public:  
    Integer(int i) {  
        _i = new int(i);  
    }  
    int& get() {  
        return *_i;  
    }  
}
```





A simple class: C++ and Java

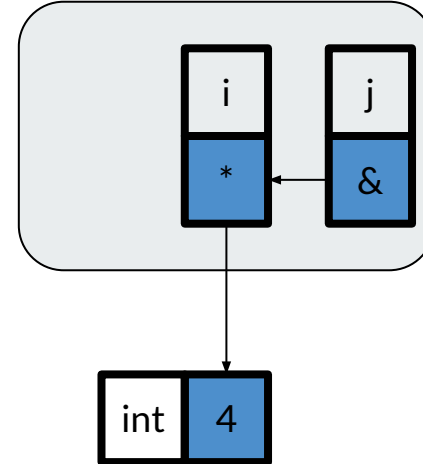
```
Integer i = Integer(3);  
cout << i.get() << endl; // 3
```





A simple class: C++ and Java

```
Integer i = Integer(3);  
Integer& j = i;  
i.get()++;  
cout << j.get() << endl; // 4
```





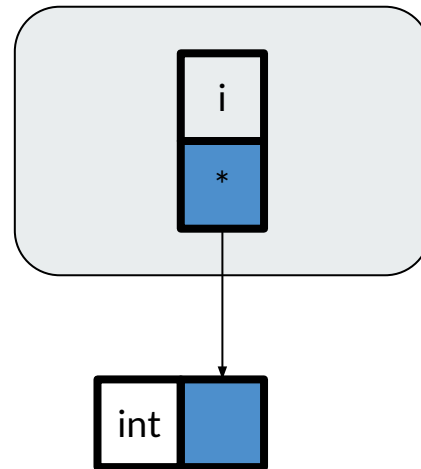
A simple class: C++ and Java

```
Integer i = Integer(3);  
cout << i.get() << endl;  
// 💣 leaking memory
```





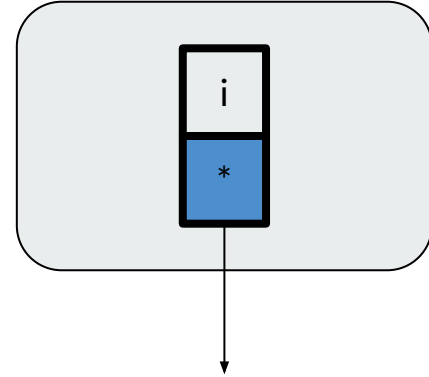
```
class Integer {  
private:  
    int *_i;  
public:  
    Integer(int i) {  
        _i = new int(i);  
    }  
    ~Integer() {  
        delete _i;  
    }  
    int& get() {  
        return *_i;  
    }  
}
```





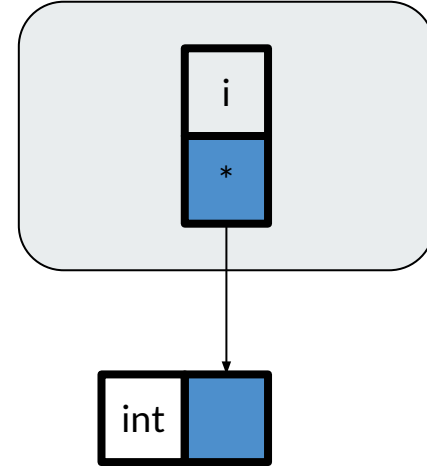
A simple class: C++ and Java

```
Integer i;  
cout << i.get() << endl; // 💣
```





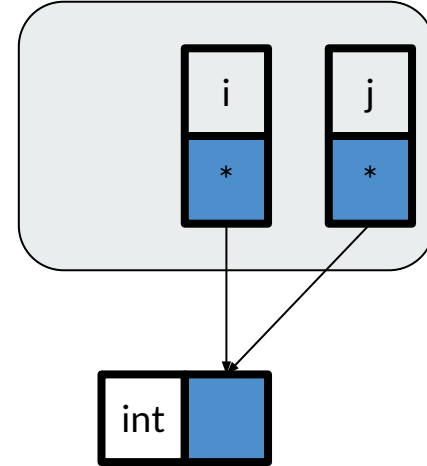
```
class Integer {  
private:  
    int *_i;  
public:  
    Integer() {  
        _i = new int();  
    }  
    Integer(int i) {  
        _i = new int(i);  
    }  
    ~Integer() {  
        delete _i;  
    }  
    int& get() {  
        return *_i;  
    }  
}
```





A simple class: C++ and Java

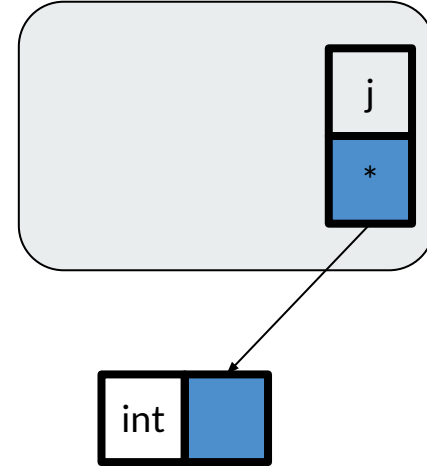
```
Integer i = Integer(3);  
Integer j = i;  
i.get()++;  
cout << j.get() << endl; // 4
```





A simple class: C++ and Java

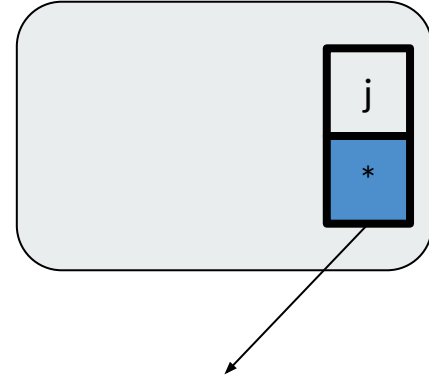
```
Integer i = Integer(3);  
Integer j = i;  
i.get()++;  
cout << j.get() << endl;
```





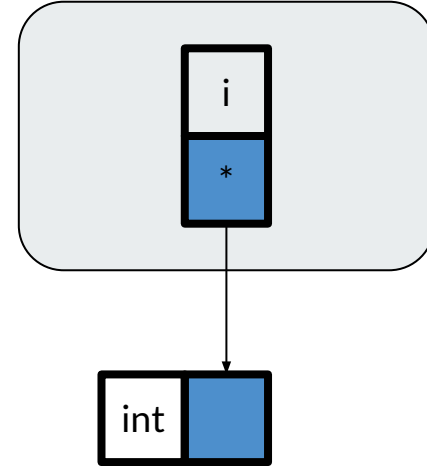
A simple class: C++ and Java

```
Integer i = Integer(3);  
Integer j = i;  
i.get()++;  
cout << j.get() << endl;  
// 💣 "double free" (deleting twice)
```





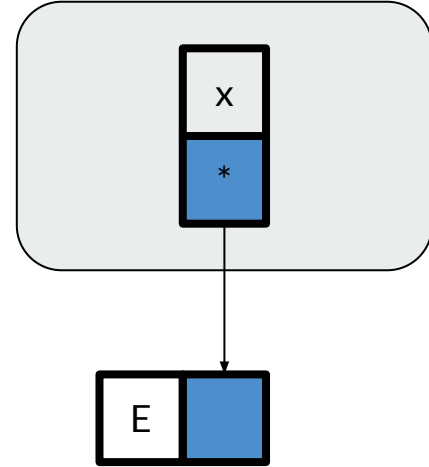
```
class Integer {  
private:  
    int *_i;  
public:  
    Integer() {  
        _i = new int();  
    }  
    Integer(int i) {  
        _i = new int(i);  
    }  
    Integer(Integer& o) {  
        _i = new int(o.get());  
    }  
    ~Integer() {  
        delete _i;  
    }  
    int& get() {  
        return *_i;  
    }  
}
```





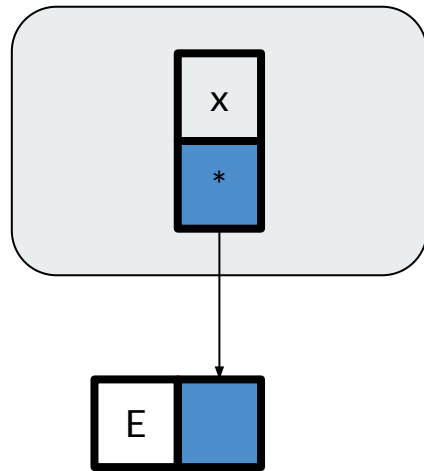
A box

```
class Box<E> {  
    private E e;  
    public E(E e) {  
        this.e = e;  
    }  
    public E get() {  
        return e;  
    }  
}
```





```
template <class E> class Box {
private:
    E *_e;
public:
    Box() {
        _e = new E();
    }
    Box(E& e) {
        _e = new E(e);
    }
    Box(Box& o) {
        _i = new E(o.get());
    }
    ~Box() {
        delete _e;
    }
    E& get() {
        return *_e;
    }
}
```





Why is the box useful? Why is C++ useful?

- Expressive: allows control of low-level behavior like memory layout
 - don't want to store a large object on the stack
- Modular: encapsulates low level operations in a reusable, compact way
 - we avoid 💣 code using the box
- Performant: no garbage collection needed
 - we know exactly what our code does

We need our trading system to be all of these



Okay, but is the box useful?

- We can store a big object on the heap with a Box on the stack
- But we need to copy it (expensive) whenever we copy the Box
- Let's do something different...



“lvalues” and “rvalues”

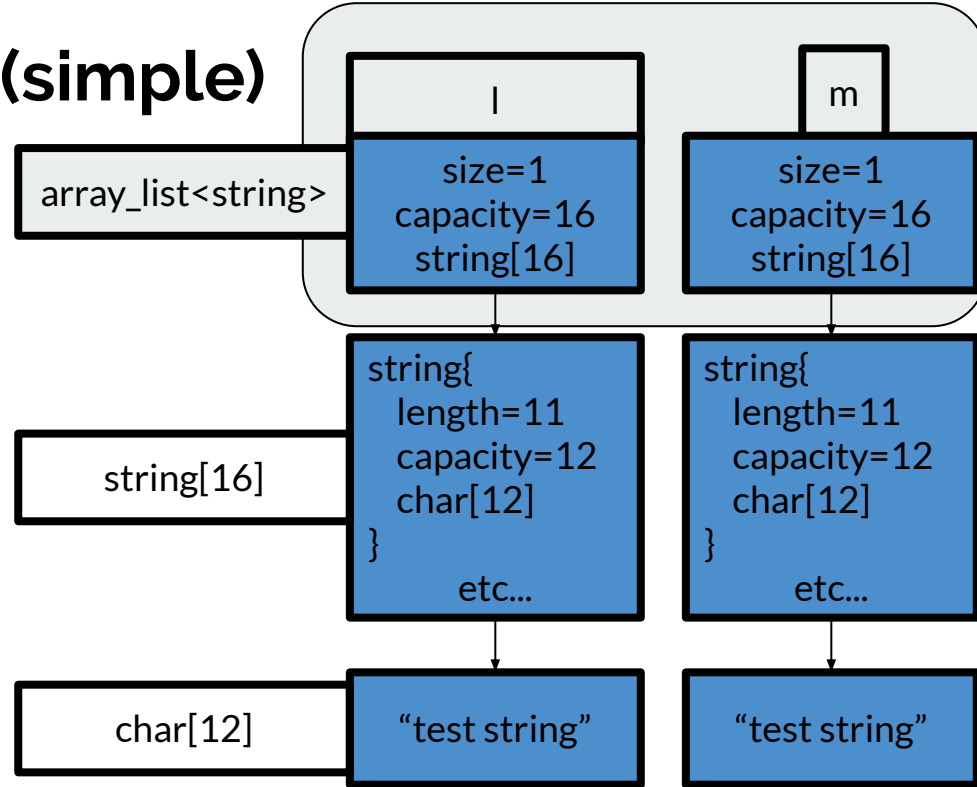
- left-hand side of =
- lvalues outlive the expression
 - have a name
- examples
 - variables
 - function return references
- right-hand side of =
- rvalues are temporary
 - no name
- examples
 - literal expressions
 - function return values

unclear? I'll demonstrate...



We want code like this (simple)

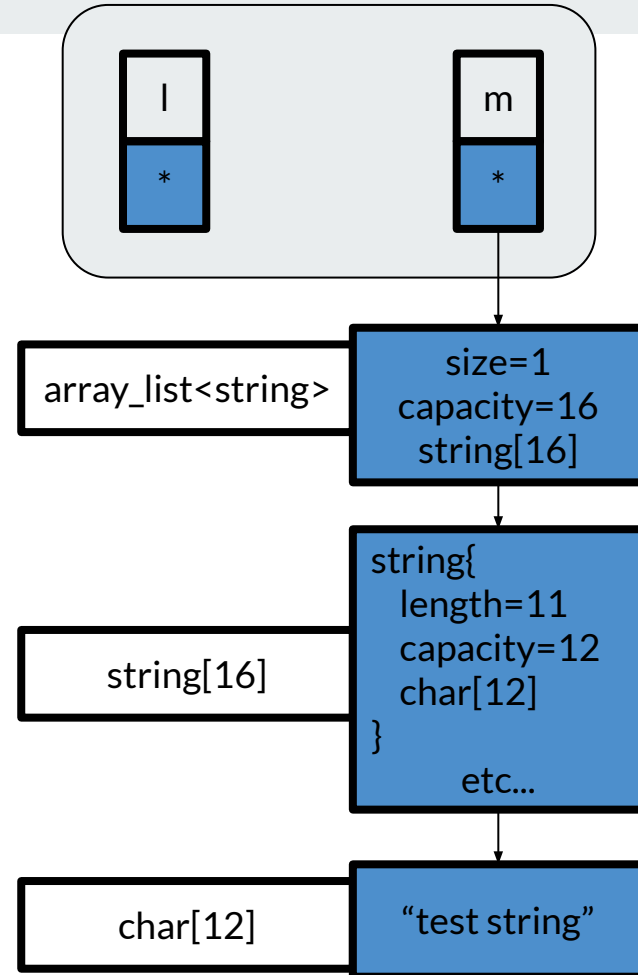
```
array_list<string> l =  
array_list<string>();  
  
array_list<string> m = l;  
  
l.add("test string");  
  
cout << m;
```





to get a result like this (gifting)

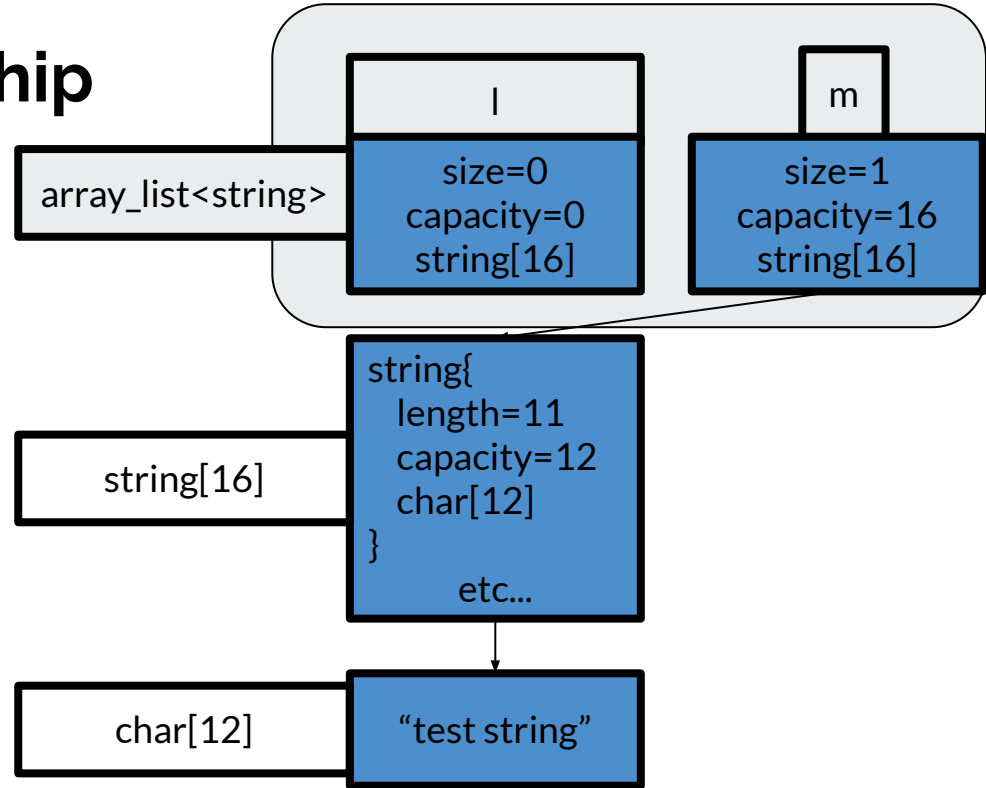
```
array_list<string>* l = new  
array_list<string>();  
  
array_list<string>* m = l;  
  
l = null;  
  
(*m).add("test string");  
  
cout << *m;
```





we can “move” ownership

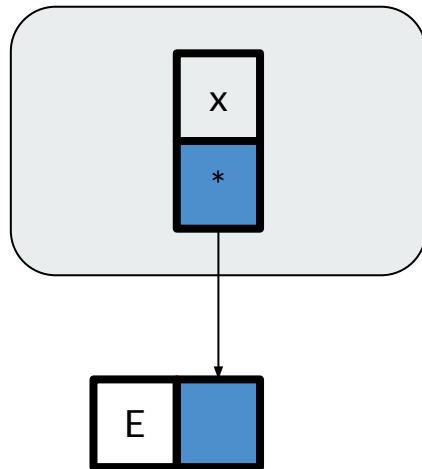
```
array_list<string> l =  
array_list<string>();  
  
array_list<string> m = move(l);  
  
l.add("test string");  
  
cout << m;
```





“move” turns an lvalue into an rvalue

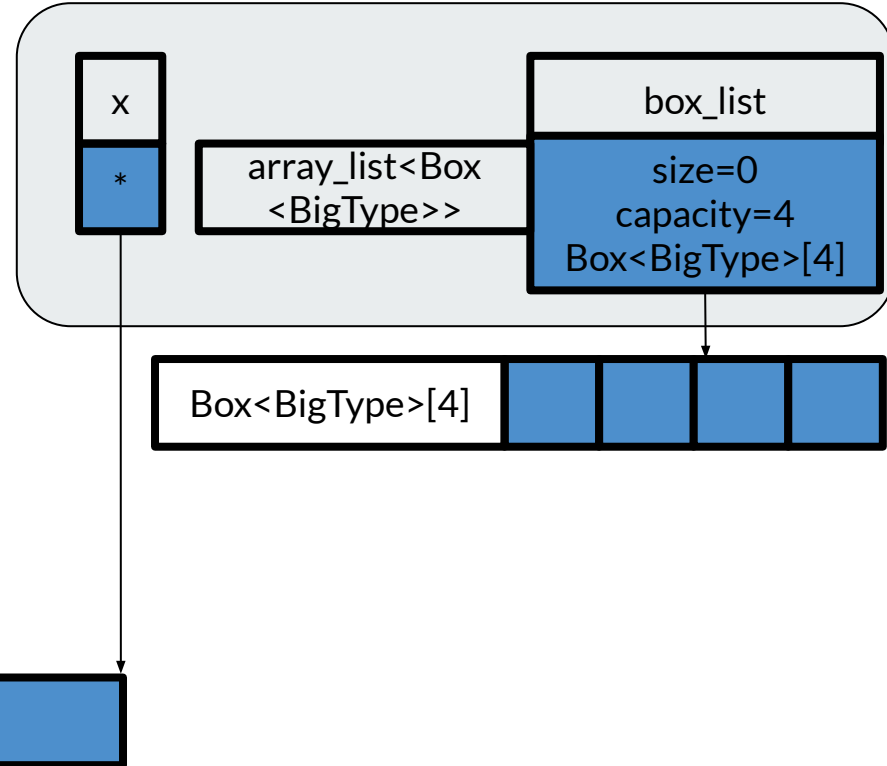
```
template <class E> class Box {
private:
    E *_e;
public:
    Box() {
        _e = new E();
    }
    Box(E& e) {
        _e = new E(e);
    }
    Box(Box& o) {
        _i = new E(o.get());
    }
    E& get() {
        return *_e;
    }
    ~Box() {
        if (_e != null) delete _e;
    }
    Box(E&& e) {
        _e = new E(move(e));
    }
    Box(Box&& o) {
        _e = o._e; o._e = null;
    }
}
```





“move” in practice

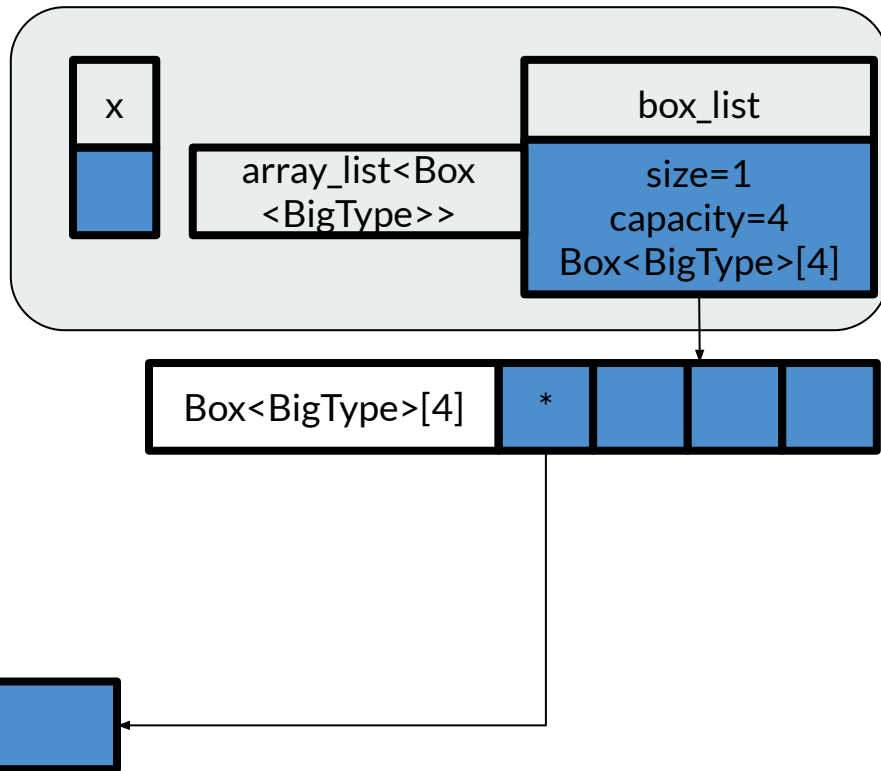
```
Box<BigType> x = Box<BigType>(args);  
array_list<Box<BigType>> box_list;
```





no reallocation of BigType

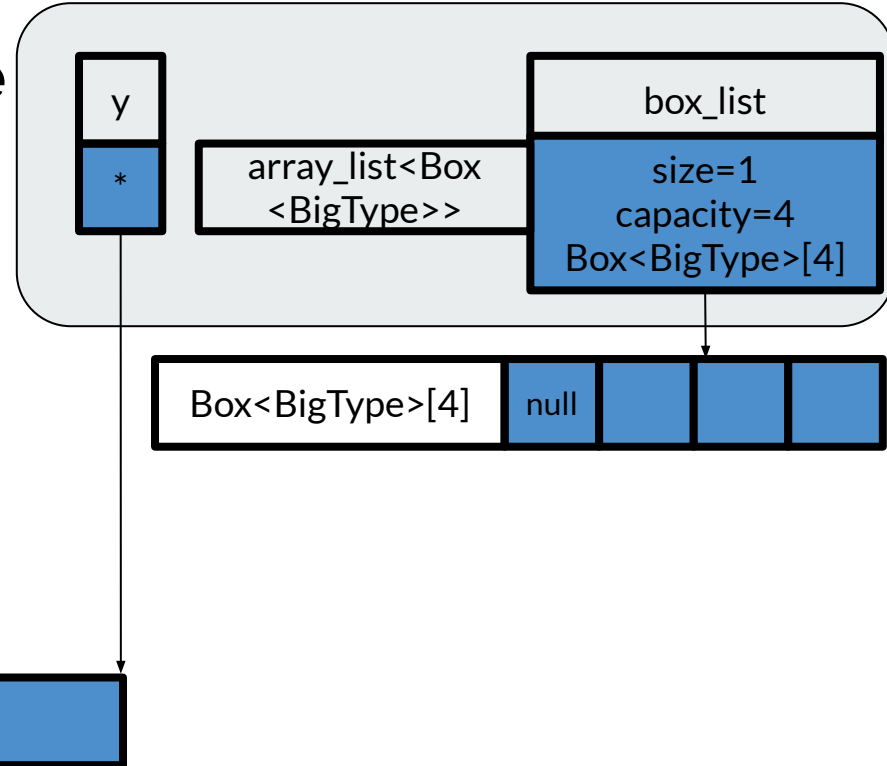
```
Box<BigType> x = Box<BigType>(args);  
array_list<Box<BigType>> box_list;  
box_list.add(move(x));
```





no reallocation of BigType

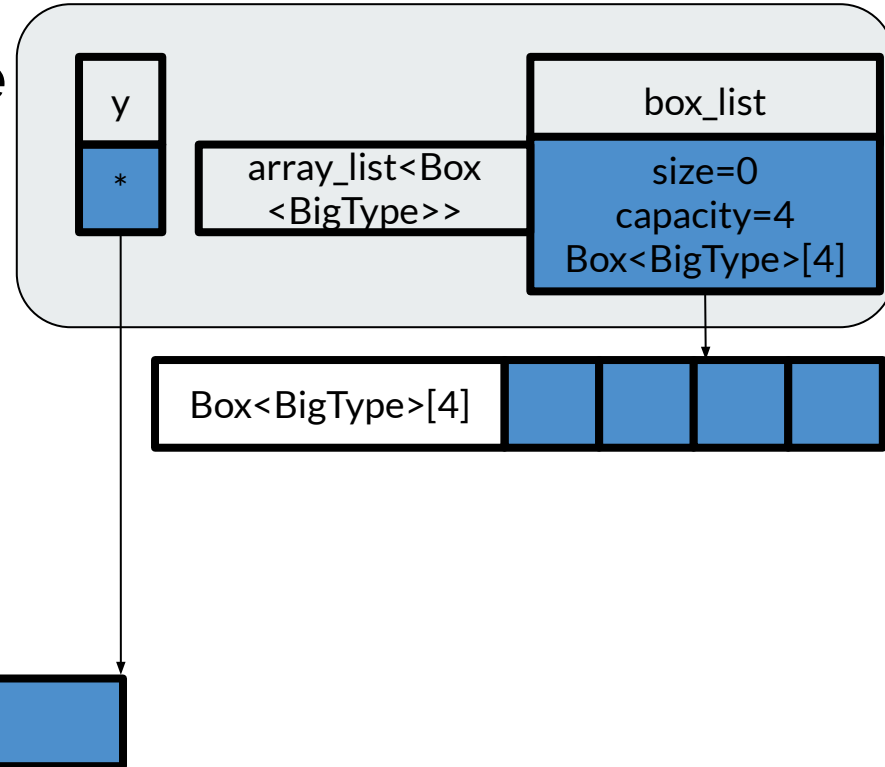
```
array_list<Box<BigType>> box_list;  
  
// ... much later ...  
  
Box<BigType> y = move(box_list[0]);
```





no reallocation of BigType

```
array_list<Box<BigType>> box_list;  
  
// ... much later ...  
  
Box<BigType> y = move(box_list[0]);  
  
box_list.pop_back();
```





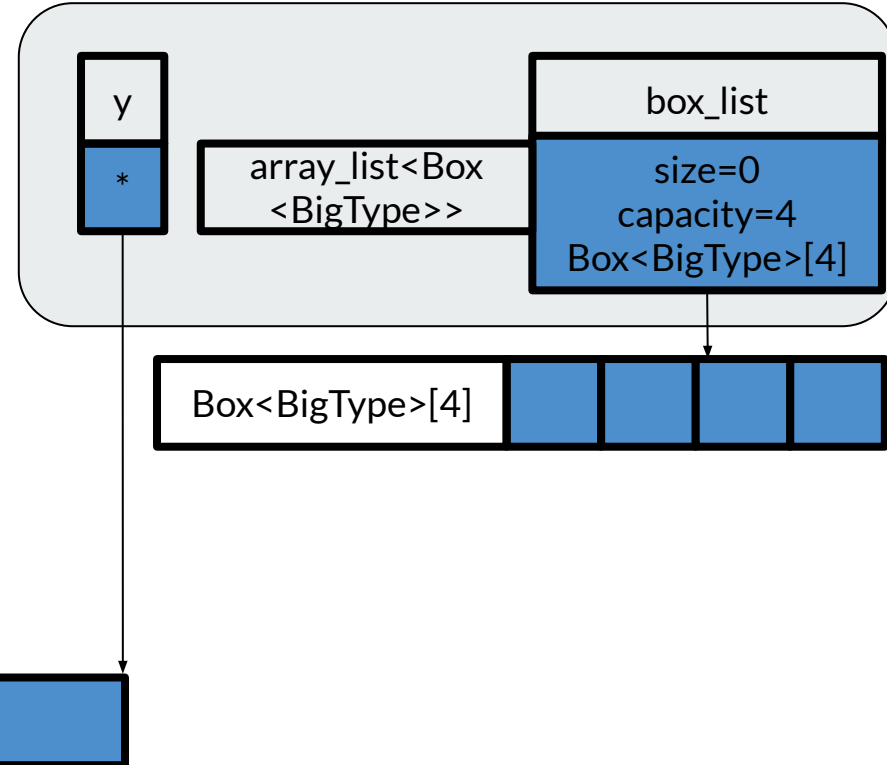
no memory management in client code

```
array_list<Box<BigType>> box_list;
```

```
// ... much later ...
```

```
Box<BigType> y = move(box_list[0]);
```

```
box_list.pop_back();
```





Questions?

- Email Andrew — andrew.wonnacott@oldmissioncapital.com
technology questions
- Email Janine — janine.manolakos@oldmissioncapital.com
trading questions
- Email Kyle — kyle.gubbins@oldmissioncapital.com
recruiting questions