Introduction to Software Development – CS 6010
Lecture 17 – Templates

Master of Software Development (MSD) Program

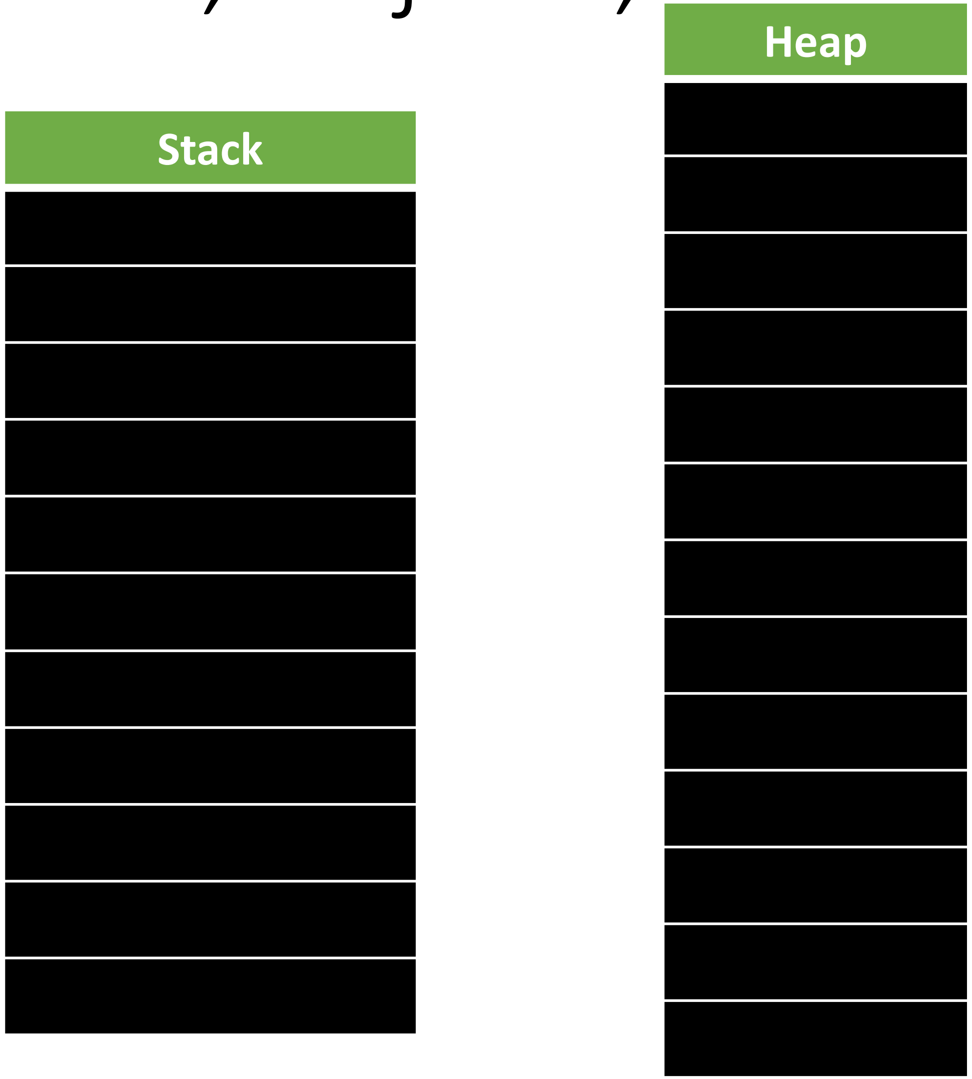
Varun Shankar

Fall 2023

Miscellaneous

- Methods, Objects, and This
- operator=
- function plus, method plus, operator plus… and const
- Rule of 3 –
  - MyVector Class?

Methods, Objects, and This



Stack

Heap

Point.h

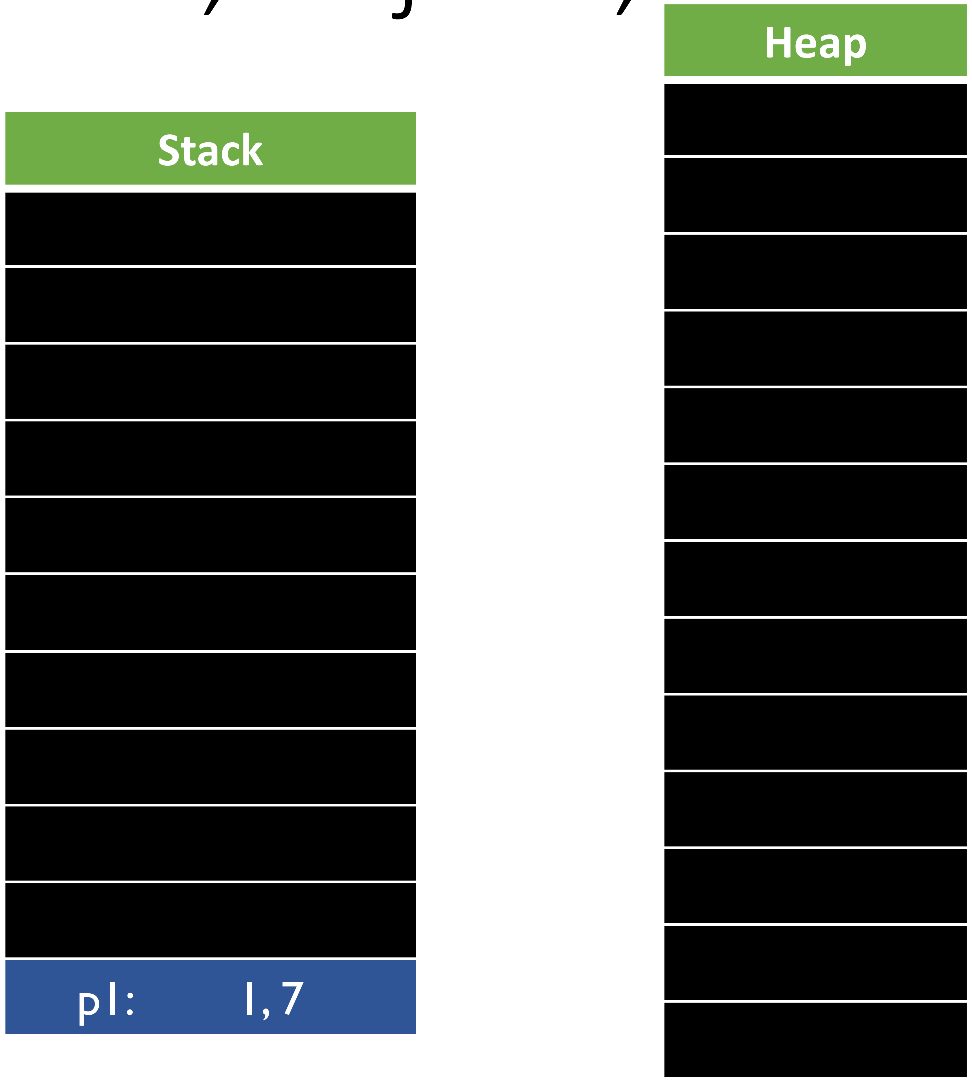
```
class Point {
        float x;
        float y;
}
```

main.cpp

```
int main() {
        Point p1( 1, 7 );


}
```

Methods, Objects, and This



Point.h

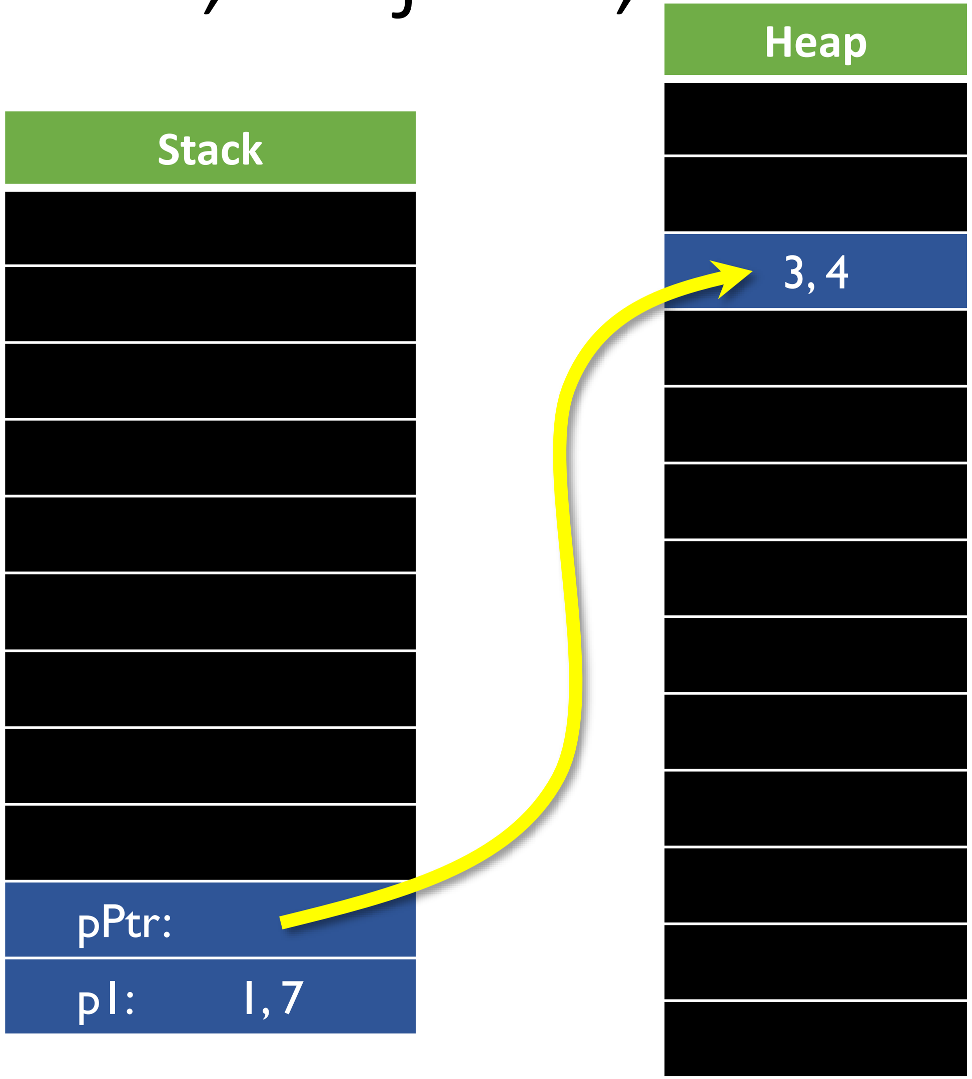
```
class Point {
        float x;
        float y;
}
```

main.cpp

```
int main() {
        Point p1( 1, 7 );
        Point * pPtr = new Point( 3, 4);


}
```

Methods, Objects, and This



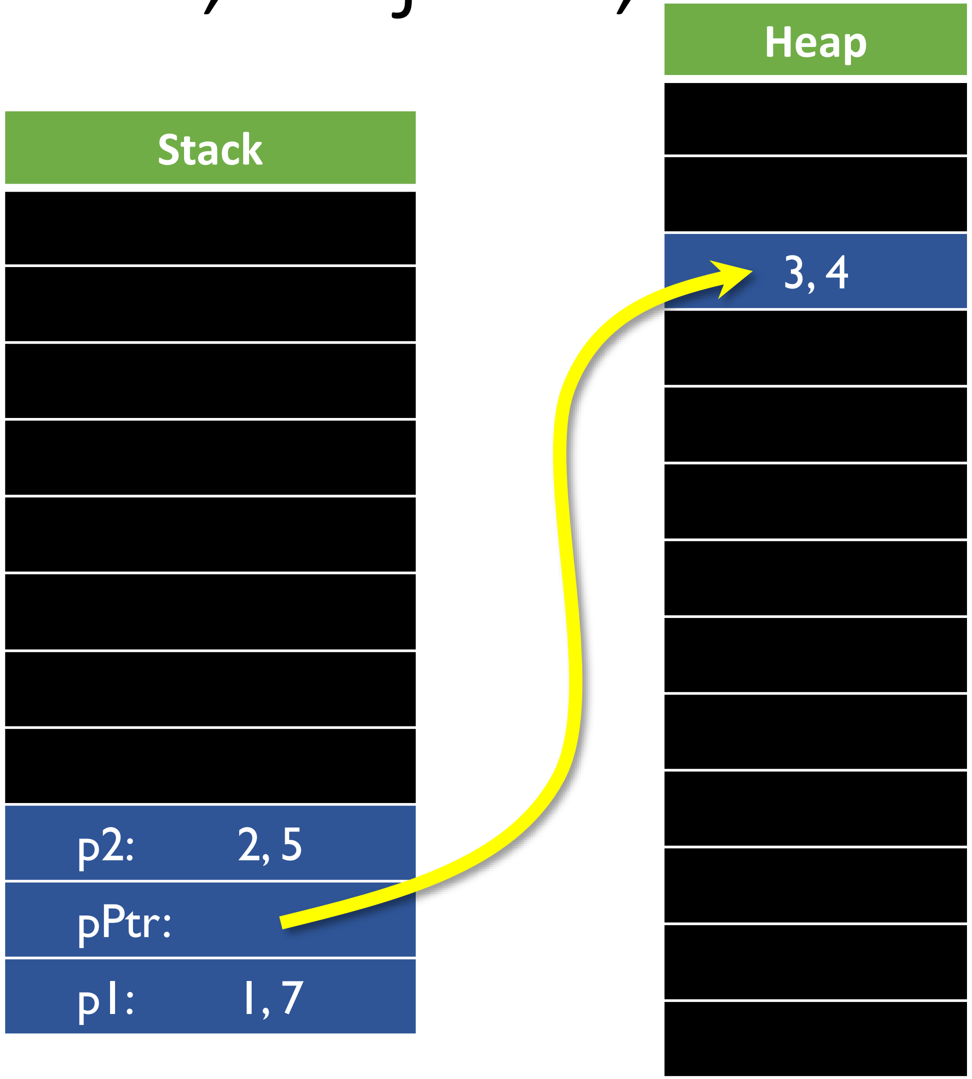
Point.h

```
class Point {
        float x;
        float y;
}
```

main.cpp

```
int main() {
        Point p1( 1, 7 );
        Point * pPtr = new
Point( 3, 4);
        Point p2( 2, 5 );
}
```

Methods, Objects, and This



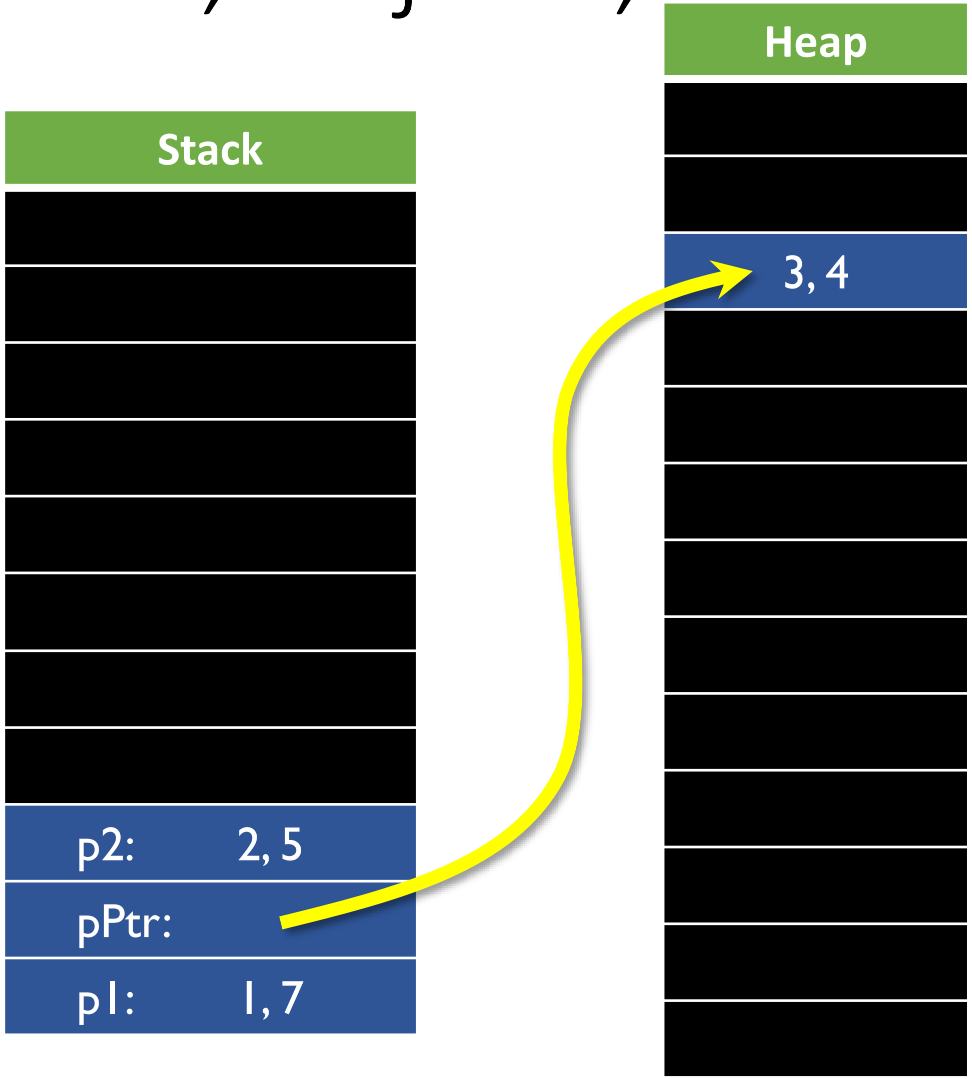
Point.h

```
class Point {
        float x;
        float y;
}
```

main.cpp

```
int main() {
        Point p1( 1, 7 );
        Point * pPtr = new
Point( 3, 4);
        Point p2( 2, 5 );
}
```

Methods, Objects, and This



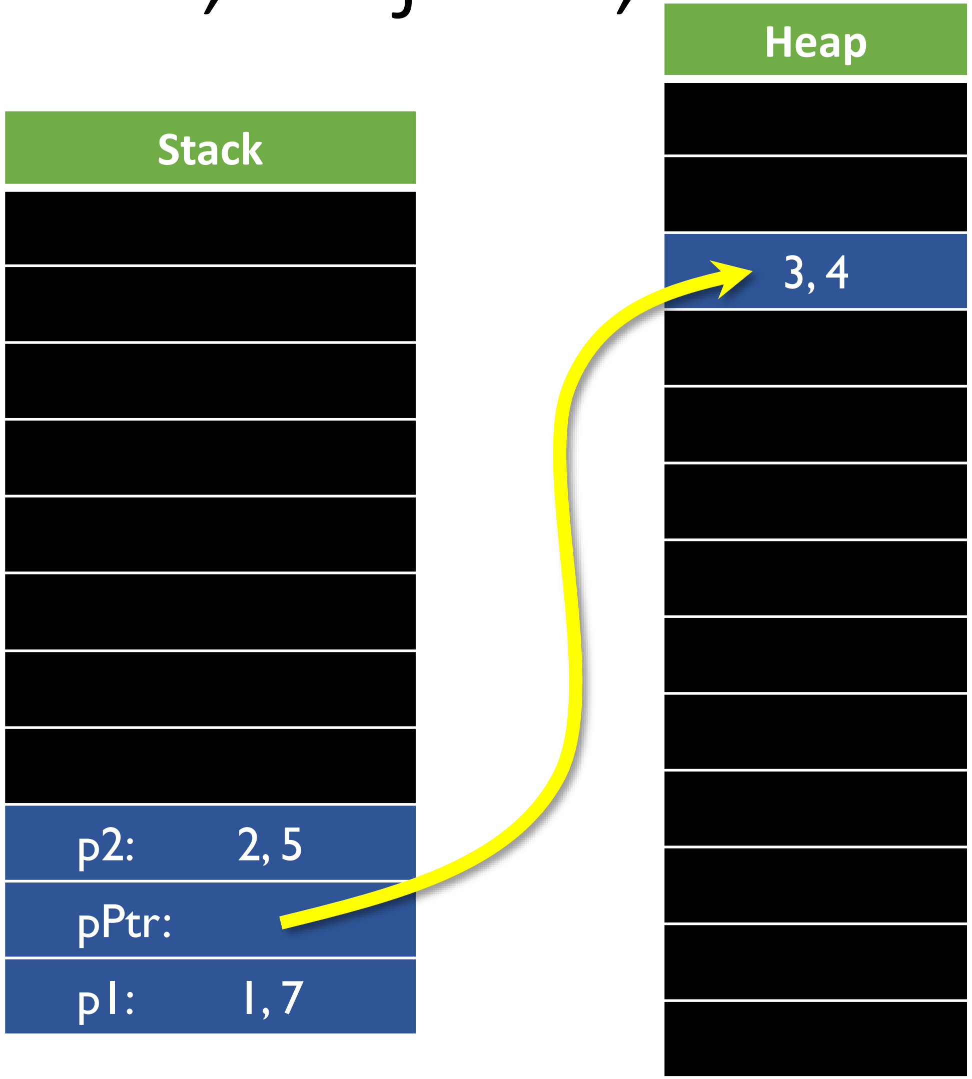
main()

```
p2 += p1; // What function gets called?
```

Point.cpp

```
Point& operator+=( const Point & rhs ) {
  // Which variable does rhs refer to?
  // What about the left hand side?
  // But there is no lhs variable…
  // Inside this function, what is the
  // lhs?
  // this object inside (p2 outside)
method
  // p2.operator+=( p1 ); // could call like this
  // This method is part of / belongs to p2
}
```

Methods, Objects, and This



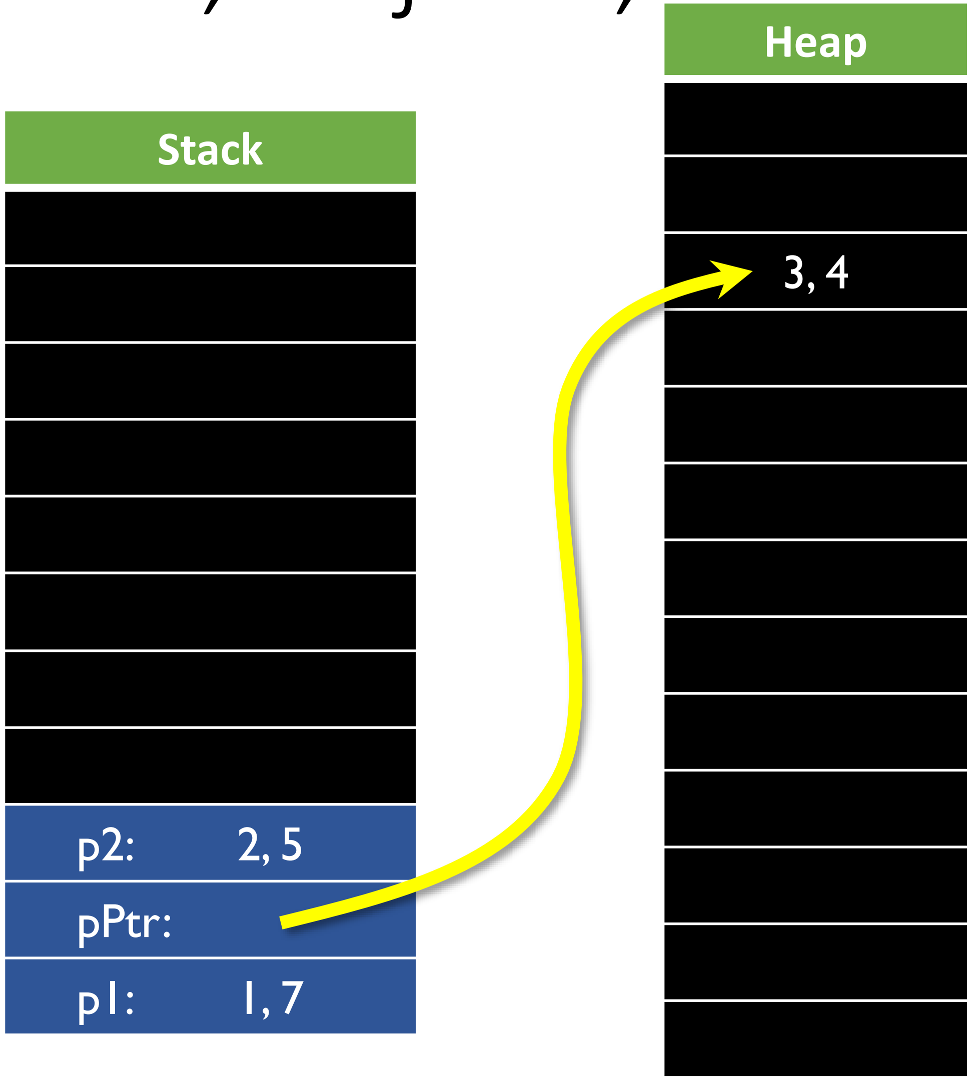
main()

// How to get rid of pPtr? What does

// it mean to "get rid of" it?

delete pPtr;

Methods, Objects, and This



main()

// How to get rid of pPtr? What does

// it mean to "get rid of" it?

delete pPtr;

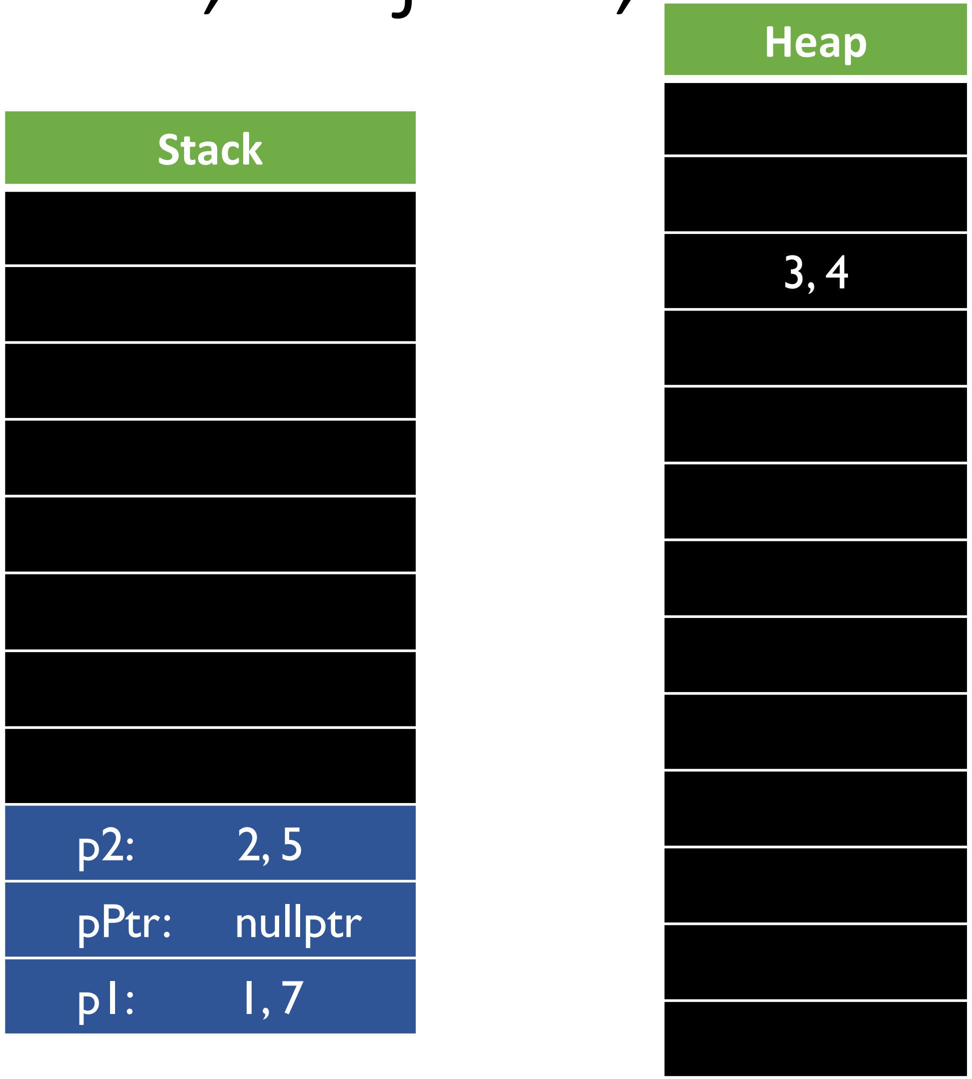
// Memory has been returned to the

// system. But nothing else has changed.

// What can we do to cleanup more?

pPtr = nullptr;

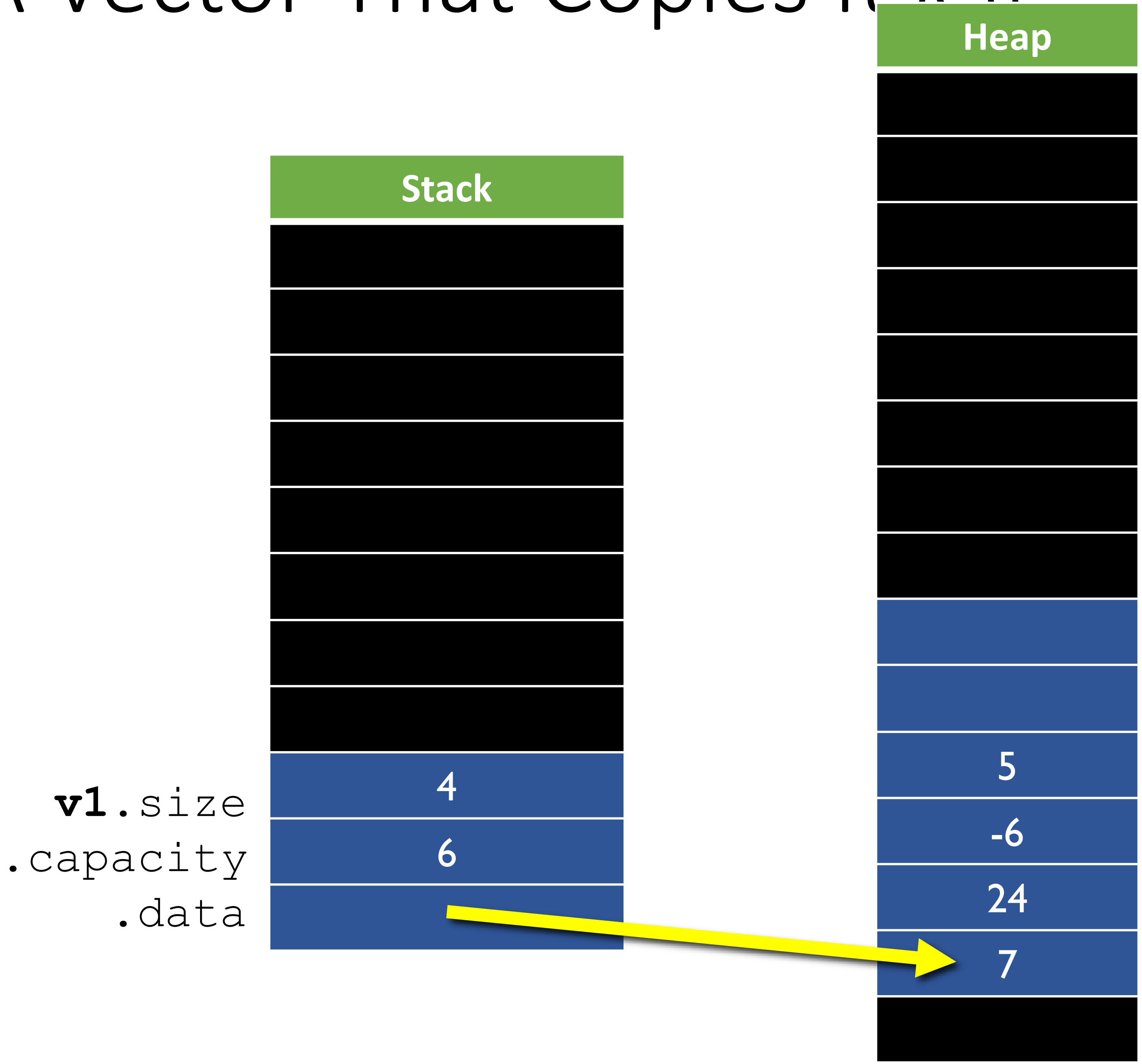
Methods, Objects, and This



main()

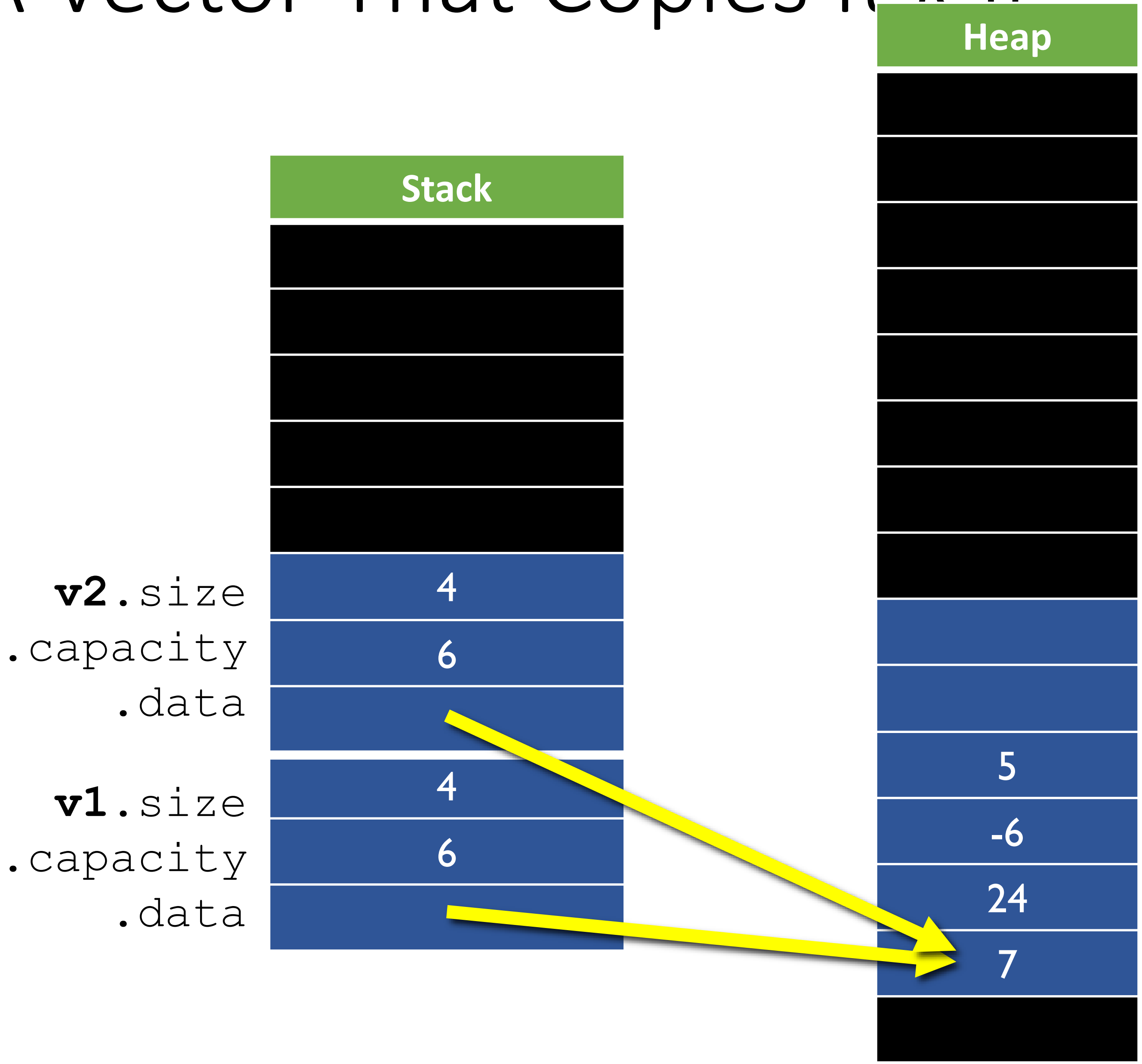
```
// How to get rid of pPtr? What does
// it mean to "get rid of" it?
delete pPtr;
// Memory has been returned to the
// system. But nothing else has changed.
// What can we do to cleanup more?
pPtr = nullptr;
```

A Vector That Copies Itself



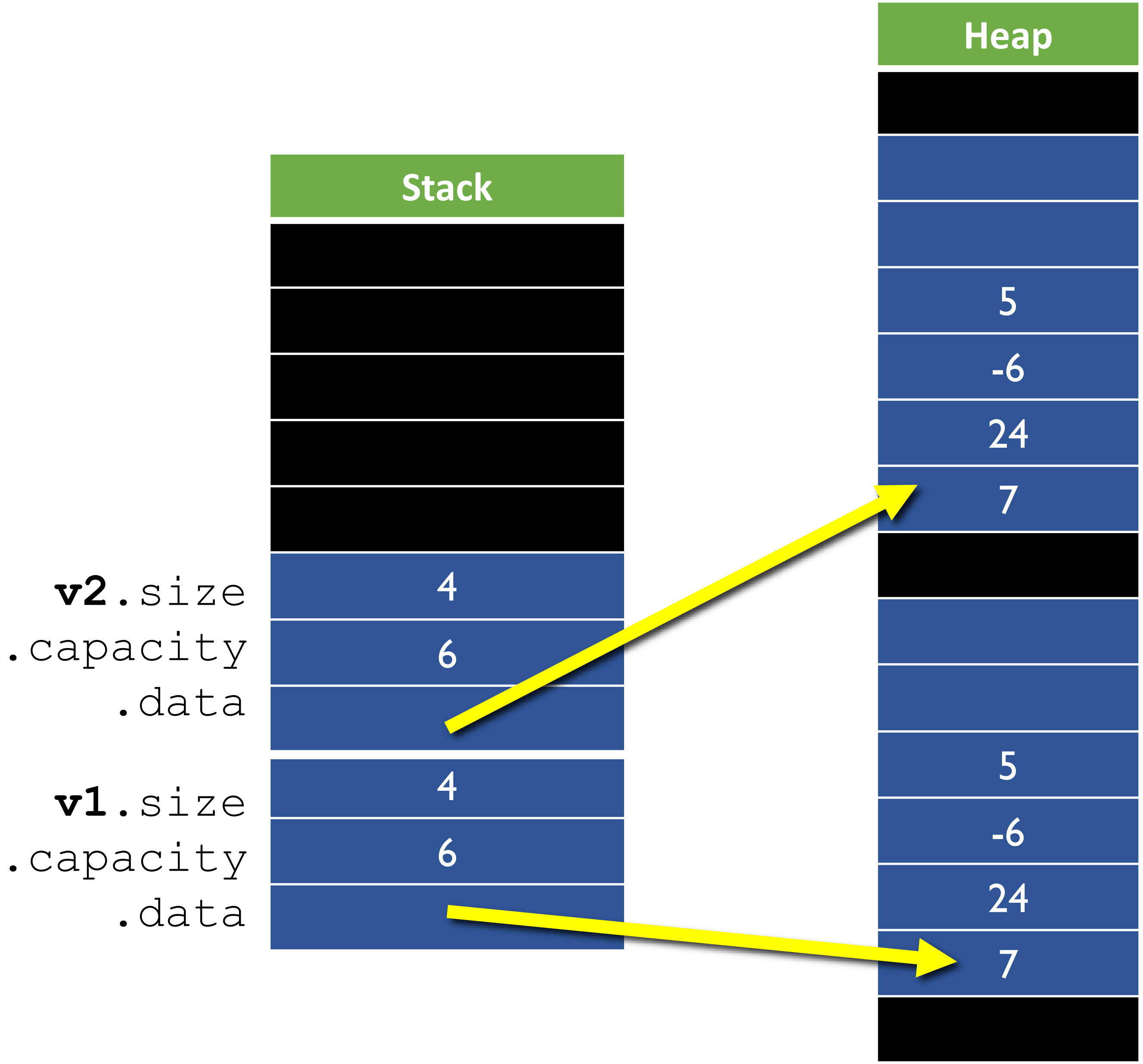
- Given a MyVector v1 as seen here… what happens when we do this:
- `MyVector v2 = v1;`
  - If we are not smart about the copy constructor (or operator=), we get…

A Vector That Copies Itself



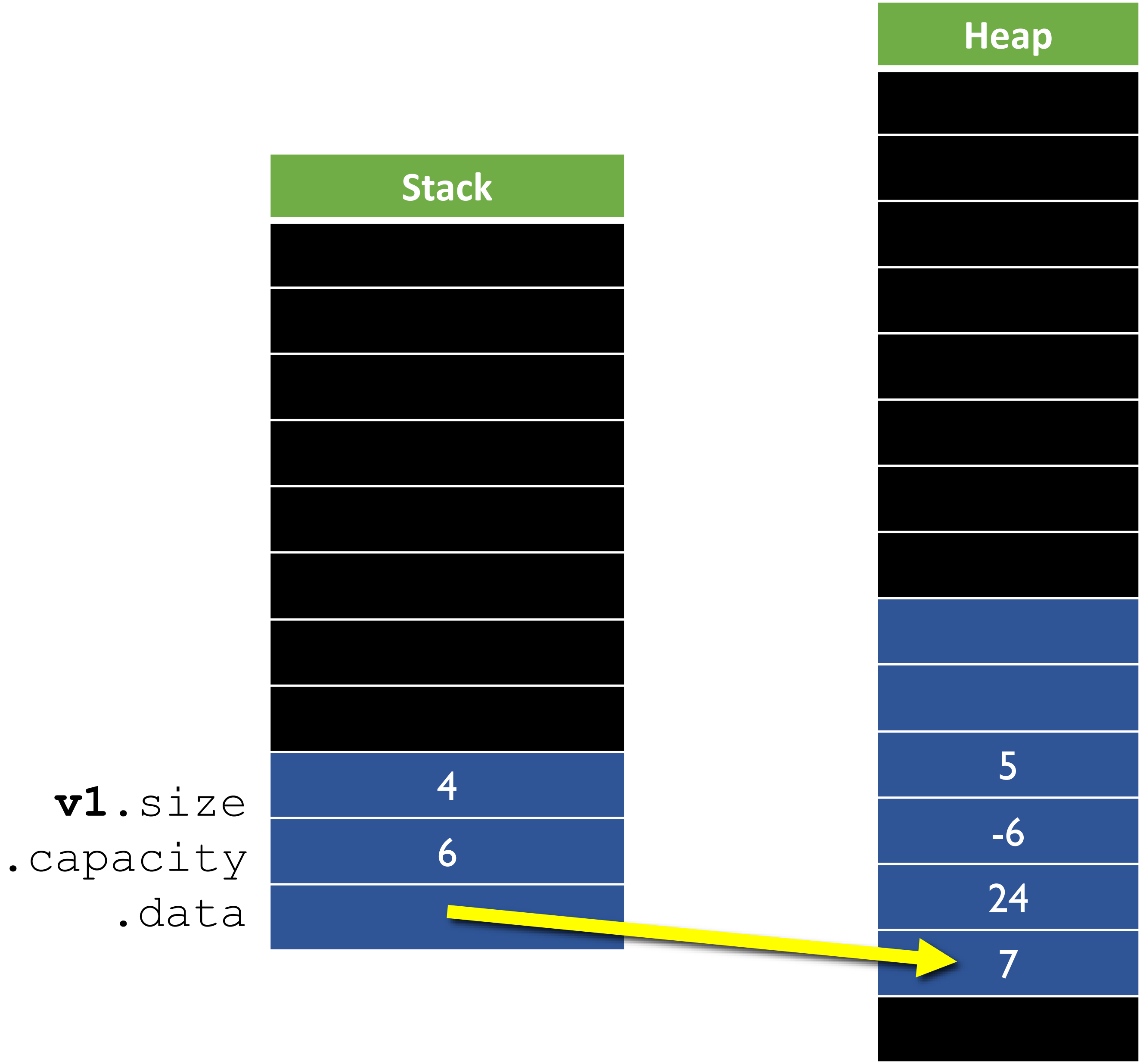
- Is this what we want? What happens when v2 is destructed?
    - No, v2 would corrupt v1's data.
- So what we really want is this…

A Vector That Copies Itself…



- v2 should allocate its own memory and copy the values into it.
- This is what memory should look like if v1 and v2 are working properly.
- Note, once v1 is copied into v2, they are separate variables and any change to either one will not (and should not) have an effect on the other.

A Vector That Copies Itself…

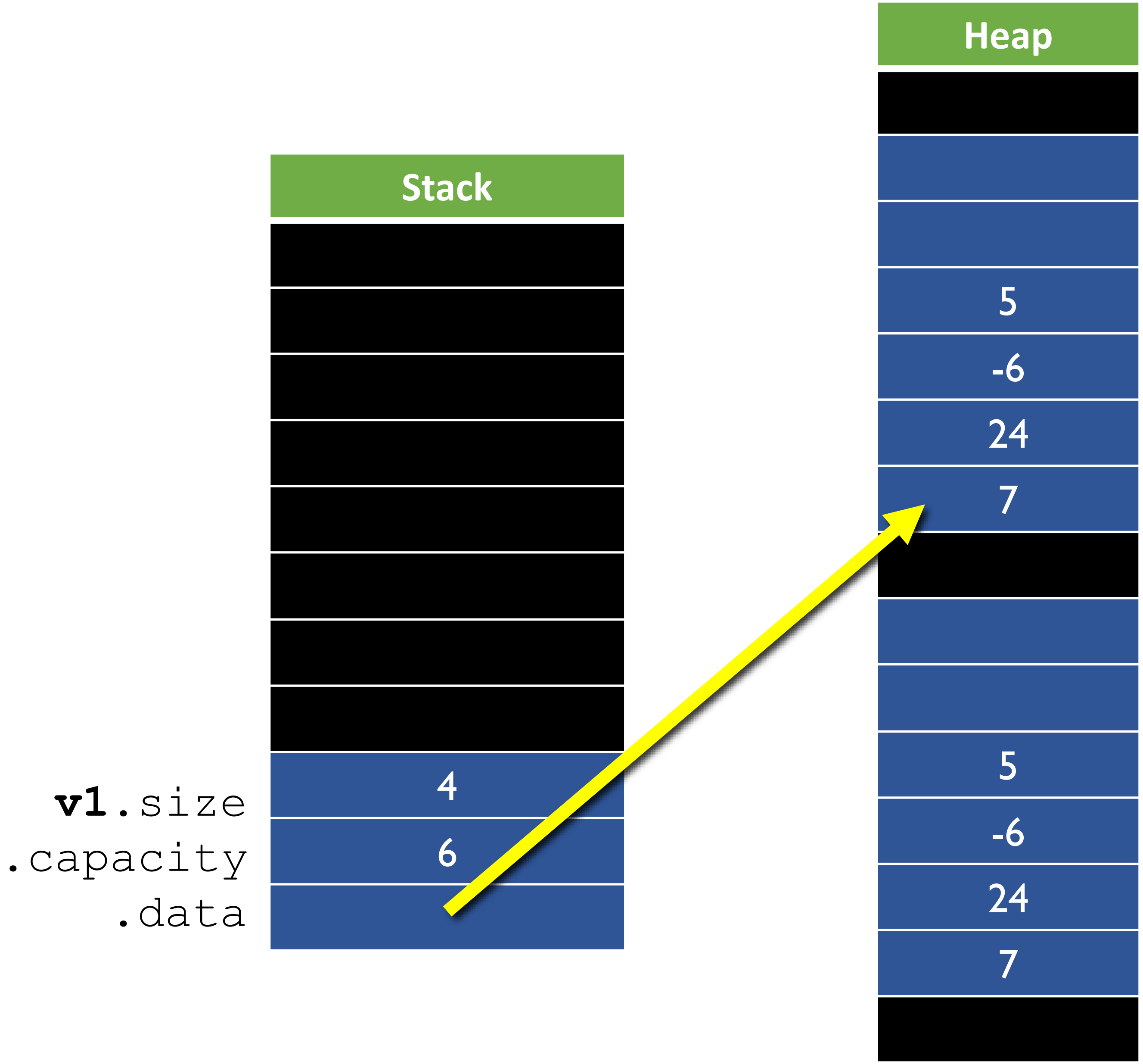


- So what happens if we write:

v1 = v1;

- Well if we are not careful, we get this:

A Vector That Copies Itself…



- So what happens if we write:

v1 = v1;

- Well if we are not careful, we get this:

- We asked for new memory (just like when we copied v2) and then copied the values over…
- What has happened?
  - We've lost the pointer to the original data array – we have *leaked* memory.

A Vector That Copies Itself… Fixed

- How do we fix this problem?

```
MyVector & operator=( const MyVector & rhs )
{
        if( this == &rhs ) { // Guard against self assignment!
                return *this;
        }
        // Otherwise do the copy…
}
```

Operator +, Method plus(), function plus()

- `Point p1, p2, p3; // With some initial values.`
- When we write: `p1 = p2 + p3;  // Do p2 or p3 change?`
  - No!
  - How can we get the compiler to enforce this for us?
    - Use *const*
- `Point plus( const & Point p1, const & Point p2 );     // function version`
  - `p1 = plus( p2, p3 );  // Usage`
- `Point plus( const & Point rhs ) const;               // method version`
  - `p1 = p2.plus( p3 );  // Usage`
- `Point operator+( const & Point rhs ) const;          // operator (method) version`
  - `p1 = p2 + p3;        // Usage`
- This applies to all of these type of functions. You should go back to your previous assignments and make sure you are using const. If not, add it – if your code fails to compile – even if it previously “passed all the tests” – you are doing something incorrect.
- Also notice that all of these functions return a Point – NOT a reference to a point. This is because addition creates something new (a new answer) – so we are not referring to (referencing) something that already exists.

Rule of 3 – When to Use

- MyVector Class
  - Do we need to implement the copy constructor, destructor, and operator =?
    - Yes – we are managing memory
    - In general, if you have pointers in your class (to memory you are actively managing), then you most likely need to implement the Rule of 3.
- When not to implement rule of 3?
  - If all the data for the class is on the stack, and we're not managing any memory ourselves.

Lecture 17 – Templates

- Topics
  - C++ Templates

A final update to our MyVector

- What is the difference between std::vector and MyVector?
  - MyVector only supports ints.
- How does std::vector support other types?
  - vector<string> words;
  - vector<Fraction> fractions;
- What is the “<type>”?
  - The template specification.
  - This allows us to create a class or function that does not explicitly specify (at least some of) the type of data it will work on.
  - When the programmer supplies a specific type to the templated class, the compiler will use our template to create a specific version of the class.
  - So this is a template for a class or a function!

Template Syntax (In Header File)

```
template<typename T>  // “T” can be any name we choose,
class MyClass {      // but “T” is a common choice.
      // Inside the class declaration, “T” is a place holder that
      // can be used anywhere a type would be specified.  The
      // actual type will be inserted by the compiler when the user
      // of this class chooses a specific type.
public:
      void someMethod( T data ); // data is of type T
      T getData();  // getData() returns data of type T
private:
      T myMemberVar_; // myMemberVar is of type T
      std::vector<T> myDataItems_; // List of items, all of type T
}
```

Template Syntax (In Header File)

- When defining (implementing) your templated methods, you must do so in the header (.h) file.
- The syntax is a bit ugly, but you eventually get used to it.

```
template<typename T>  // <- Keyword/syntax necessary before function
MyClass<T>::MyClass() { // Constructor }

template<typename T>
T MyClass<T>::getData() { ... }

template<typename T>
void MyClass<T>::someMethod( T data ) { ... }
```

- Basically, the name of the class becomes “MyClass<T>” when prefixing functions / constructors with the name of the class.

Using a Class Template

- What does it mean when you are asked to use something?
  - In terms of classes/data, it means to:
    - Create an object (aka a variable)!
  - In terms of functions/methods:
    - Call that function/method.
  - We need to understand the difference between declaring, defining, and using.
- We’ve already seen this with:
  - vector<string> words;
- We do the same with our new class:
  - MyClass<int> myVariable;
    - When the compiler sees myVariable and that it is of type MyClass<int>, it will replace the “T” everywhere in the class definition with “int”
  - MyClass<string> anotherVariable;
    - “T” is replaced everywhere in the class with “string”.

Compilation Errors

- The compiler does not know what T is until you create a variable of type class<T> and give it a specific type.
- It therefore does very little error checking until you fill in the type (by making a variable).
- Auto-completion in XCode gets much worse when working with a templated class.
- Also, remember that all the code you are used to putting in the .cpp file, must go in the .h file when creating a templated class.

Function Templates

- So far we have been talking about creating a new class that is templated.
- You can also create standalone functions that are templated:

```
template<typename T>
void print( const T & item ) {
        cout << item << “\n”;
}
```

- These must also go in the .h file.
- Most of the time the compiler figures out the type for you. For example:
  - print(10) // compiler knows 10 is an integer so creates a “print(const int & item)” function.
- However, you can force the compiler to use a specific type (though this is rarely needed):
  - print<double>(15); // make the compiler treat 15 as a double.

Function Template

- Generic template function definition

```
template<typename T>
T addOne( T x ) {
        return x + 1;
}
int y = addOne( 7 );
MyVector f1( 1, 2 );
MyVector f2 = addOne( f1 );  // ERROR (at this point).
```

- Compiler does not know how to do: Fraction + 1.
- Specific template function instantiation (definition):

```
template<>
MyVector addOne(MyVector x ) {
                return x + MyVector( 1, 1 ); // Add one in terms of fractions.
}
```

Assignment

- Code Review catch-up
- Homework – Templatize Your Vector