## Everything You Ever Wanted To Know About Move Semantics

### (and then some)

Howard Hinnant Senior Software Engineer Ripple Labs

April 12, 2014

## Outline

- The genesis of move semantics
- Special member functions
- Introduction to the special move members

### How Did Move Semantics Get Started?

- It was all about optimizing std::vector<T>.
- And everything else just rode along on its coattails.

## What is std::vector?

• Anatomy of a vector (simplified)



## How does a vector copy?



## How does a vector move?



### How Did Move Semantics Get Started?

- Remember these fundamentals about move semantics and vector, and you will have a basic understanding of all of move semantics.
- The rest is just details...

## Outline

- The genesis of move semantics
- Special member functions
- Introduction to the special move members

• What are they?

 Special members are those member functions that the compiler can be asked to automatically generate code for.

• How many special members are there?



- They are:
  - default constructor
  - destructor
  - copy constructor
  - copy assignment
  - move constructor
  - move assignment

X(); ~X(); X(X const&); X& operator=(X const&); X(X&&); X& operator=(X&&);

• The special members can be: \_ deleted

- not declared
- implicitly declared or defaulted

or

• user declared

user-defined

• What counts as user-declared?

```
struct X
{
    X() {} // user-declared
    X(); // user-declared
    X() = default; // user-declared
    X() = delete; // user-declared
};
```

• What is the difference between not-declared and deleted?

```
Consider:
       struct X
       \left\{ \right.
            template <class ... Args>
                 X(Args&& ...args);
            // The default constructor
            // is not declared
       };
```

### Special Members struct X template <class ...Args> X(Args&& ...args); // The default destructor // is not declared };

• X can be default constructed by using the variadic constructor.

```
Special Members
struct X
    template <class ... Args>
       X(Args&& ...args);
    X() = delete;
};
```

- Now X() binds to the deleted default constructor instead of the variadic constructor.
- X is no longer default constructible.

```
Special Members
struct X
    template <class ... Args>
       X(Args&& ...args);
    X() = delete;
};
```

- Deleted members participate in overload resolution.
- Members not-declared do not participate in overload resolution.

 Under what circumstances are special members implicitly provided?

#### compiler implicitly declares

|         | default<br>constructor | destructor | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|---------|------------------------|------------|---------------------|--------------------|---------------------|--------------------|
| Nothing | defaulted              | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |

- If the user declares no special members or constructors, all 6 special members will be defaulted.
- This part is no different from C++98/03

#### compiler implicitly declares

|         | default<br>constructor | destructor | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|---------|------------------------|------------|---------------------|--------------------|---------------------|--------------------|
| Nothing | defaulted              | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |

- "defaulted" can mean "deleted" if the defaulted special member would have to do something illegal, such as call another deleted function.
- Defaulted move members defined as deleted, actually behave as not declared.
  - No, I'm not kidding!

#### compiler implicitly declares

|                    | default<br>constructor | destructor | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|--------------------|------------------------|------------|---------------------|--------------------|---------------------|--------------------|
| Nothing            | defaulted              | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |
| Any<br>constructor | not<br>declared        | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |

• If the user declares any non-special constructor, this will inhibit the implicit declaration of the default constructor.

#### compiler implicitly declares

|                        | default<br>constructor | destructor | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|------------------------|------------------------|------------|---------------------|--------------------|---------------------|--------------------|
| Nothing                | defaulted              | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |
| Any<br>constructor     | not<br>declared        | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |
| default<br>constructor | user<br>declared       | defaulted  | defaulted           | defaulted          | defaulted           | defaulted          |

• A user-declared default constructor will not inhibit any other special member.

#### compiler implicitly declares

|                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|------------------------|------------------------|------------------|---------------------|--------------------|---------------------|--------------------|
| Nothing                | defaulted              | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| destructor             | defaulted              | user<br>declared | defaulted           | defaulted          | not<br>declared     | not<br>declared    |

- A user-declared destructor will inhibit the implicit declaration of the move members.
- The implicitly defaulted copy members are deprecated.
  - If you declare a destructor, declare your copy members too, even though not necessary.

#### compiler implicitly declares

|                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|------------------------|------------------------|------------------|---------------------|--------------------|---------------------|--------------------|
| Nothing                | defaulted              | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| destructor             | defaulted              | user<br>declared | defaulted           | defaulted          | not<br>declared     | not<br>declared    |
| copy<br>constructor    | not<br>declared        | defaulted        | user<br>declared    | defaulted          | not<br>declared     | not<br>declared    |

• A user-declared copy constructor will inhibit the default constructor and move members.

#### compiler implicitly declares

|                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|------------------------|------------------------|------------------|---------------------|--------------------|---------------------|--------------------|
| Nothing                | defaulted              | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| destructor             | defaulted              | user<br>declared | defaulted           | defaulted          | not<br>declared     | not<br>declared    |
| copy<br>constructor    | not<br>declared        | defaulted        | user<br>declared    | defaulted          | not<br>declared     | not<br>declared    |
| copy<br>assignment     | defaulted              | defaulted        | defaulted           | user<br>declared   | not<br>declared     | not<br>declared    |

user declares

• A user-declared copy assignment will inhibit the move members.

#### compiler implicitly declares

|                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|------------------------|------------------------|------------------|---------------------|--------------------|---------------------|--------------------|
| Nothing                | defaulted              | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| destructor             | defaulted              | user<br>declared | defaulted           | defaulted          | not<br>declared     | not<br>declared    |
| copy<br>constructor    | not<br>declared        | defaulted        | user<br>declared    | defaulted          | not<br>declared     | not<br>declared    |
| copy<br>assignment     | defaulted              | defaulted        | defaulted           | user<br>declared   | not<br>declared     | not<br>declared    |
| move<br>constructor    | not<br>declared        | defaulted        | deleted             | deleted            | user<br>declared    | not<br>declared    |

• A user-declared move member will implicitly delete the copy members.

#### compiler implicitly declares

|       |                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|-------|------------------------|------------------------|------------------|---------------------|--------------------|---------------------|--------------------|
|       | Nothing                | defaulted              | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
|       | Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| lares | default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| decl  | destructor             | defaulted              | user<br>declared | defaulted           | defaulted          | not<br>declared     | not<br>declared    |
| JSer  | copy<br>constructor    | not<br>declared        | defaulted        | user<br>declared    | defaulted          | not<br>declared     | not<br>declared    |
|       | copy<br>assignment     | defaulted              | defaulted        | defaulted           | user<br>declared   | not<br>declared     | not<br>declared    |
|       | move<br>constructor    | not<br>declared        | defaulted        | deleted             | deleted            | user<br>declared    | not<br>declared    |
|       | move<br>assignment     | defaulted              | defaulted        | deleted             | deleted            | not<br>declared     | user<br>declared   |

#### compiler implicitly declares

|                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment |
|------------------------|------------------------|------------------|---------------------|--------------------|
| Nothing                | defaulted              | defaulted        | defaulted           | defaulted          |
| Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          |
| default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          |
| destructor             | defaulted              | user<br>declared | defaulted           | defaulted          |
| copy<br>constructor    | not<br>declared        | defaulted        | user<br>declared    | defaulted          |
| copy<br>assignment     | defaulted              | defaulted        | defaulted           | user<br>declared   |

This is C++98/03

#### compiler implicitly declares

|       |                        | default<br>constructor | destructor       | copy<br>constructor | copy<br>assignment | move<br>constructor | move<br>assignment |
|-------|------------------------|------------------------|------------------|---------------------|--------------------|---------------------|--------------------|
|       | Nothing                | defaulted              | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
|       | Any<br>constructor     | not<br>declared        | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| lares | default<br>constructor | user<br>declared       | defaulted        | defaulted           | defaulted          | defaulted           | defaulted          |
| decl  | destructor             | defaulted              | user<br>declared | defaulted           | defaulted          | not<br>declared     | not<br>declared    |
| JSer  | copy<br>constructor    | not<br>declared        | defaulted        | user<br>declared    | defaulted          | not<br>declared     | not<br>declared    |
|       | copy<br>assignment     | defaulted              | defaulted        | defaulted           | user<br>declared   | not<br>declared     | not<br>declared    |
|       | move<br>constructor    | not<br>declared        | defaulted        | deleted             | deleted            | user<br>declared    | not<br>declared    |
|       | move<br>assignment     | defaulted              | defaulted        | deleted             | deleted            | not<br>declared     | user<br>declared   |

class X An alternate presentation of the same information.  ${$ public: X() = default; $\sim X() = default;$ X(X const&) = default;X& operator=(X const&) = default; X(X&&) = default;X& operator=(X&&) = default; };

class X An alternate presentation of the same information.  ${$ public: X() = default; $\sim X() = default;$ X(X const&) = default;X& operator=(X const&) = default; X(X&&) = default;X& operator=(X&&) = default; };

class X
{
public:

An alternate presentation of the same information.

~X() = default; X(X const&) = default; X& operator=(X const&) = default;



class X
{
public:

An alternate presentation of the same information.

~X() = default; X(X const&) = delete; X& operator=(X const&) = delete; X(X&&) = default;

};

class X An alternate presentation of the same information.  ${\mathbf{f}}$ public: X() = default; $\sim X() = default;$ X(X const&) = delete;X& operator=(X const&) = delete;

X& operator=(X&&) = default;
};

## Outline

- The genesis of move semantics
- Special member functions
- Introduction to the special move members

```
What does a defaulted
  move constructor do?
class X
 : public Base
{
 Member m_;
 X(X&& x)
   : Base(static_cast<Base&&>(x))
     m_(static_cast<Member&&>(x.m_))
 {}
```

```
What does a typical user-
defined move constructor do?
class X
  : public Base
{
  Member m_;
  X(X&& x)
    : Base(std::move(x))
      m_(std::move(x.m_))
  {
    x.set_to_resourceless_state();
```

```
What does a defaulted
  move assignment do?
class X
 : public Base
{
 Member m_;
 X& operator=(X&& x) {
   Base::operator=
           (static_cast<Base&&>(x));
   m_ = static_cast<Member&&>(x.m_);
   return *this;
```

```
What does a typical user-
defined move assignment do?
class X
  : public Base
{
  Member m_;
  X& operator=(X&& x) {
    Base::operator=(std::move(x));
    m_{-} = std::move(x.m_);
    x.set_to_resourceless_state();
    return *this;
```

Yes.

# 



- Give each special member the tender loving care it deserves.
- The entire point of move semantics is to boost performance.

```
Should I define one special
 member in terms of another?
 Case study: the copy/swap idiom
class X
{
 std::vector<int> v_;
public:
 X& operator=(X x) { // Implements
   v_.swap(x.v_); // both copy and
   return *this; // move assignment
 }
           What's not to love?
```

Case study: the copy/swap idiom

class X |'
{
 std::vector<int> v\_;
 public:
 X& operator=(X const& x);
 X& operator=(X&& x);
};

I've written highly optimized versions of the copy and move assignment operators.

### Case study: the copy/swap idiom



Case study: the copy/swap idiom

How hard is it to make separate optimized copy and move assignment operators for this case?

Should I define one special member in terms of another? Case study: the copy/swap idiom class X { std::vector<int> v\_; public: // Just keep your grubby fingers // off of the keyboard. // The defaults are optimal!

What's not to love?

Case study: the copy/swap idiom

But the copy/swap idiom gives me strong exception safety!

Good point. Are all of your clients willing to pay a giant performance penalty for strong exception safety on assignment?

Case study: the copy/swap idiom

Perhaps you could interest the portion of your clients that do need strong exception safety in this generic function:

template <class C>

C& strong\_assign(C& dest, C src) {
 using std::swap;
 swap(dest, src);
 return dest;

Case study: the copy/swap idiom

Now clients who need speed can:

$$x = y;$$

And clients who need strong exception safety can:

strong\_assign(x, y);

## In A Hurry?

 If you don't have time to carefully consider all 6 special members, then just delete the copy members:

```
class X
{
  public:
    X(X const&) = delete;
    X& operator=(X const&) = delete;
};
```

## Summary

- Know when the compiler is defaulting or deleting special members for you, and what defaulted members will do.
- Always define or delete a special member when the compiler's implicit action is not correct.
- Give tender loving care to each of the 6 special members, even if the result is to let the compiler handle it.