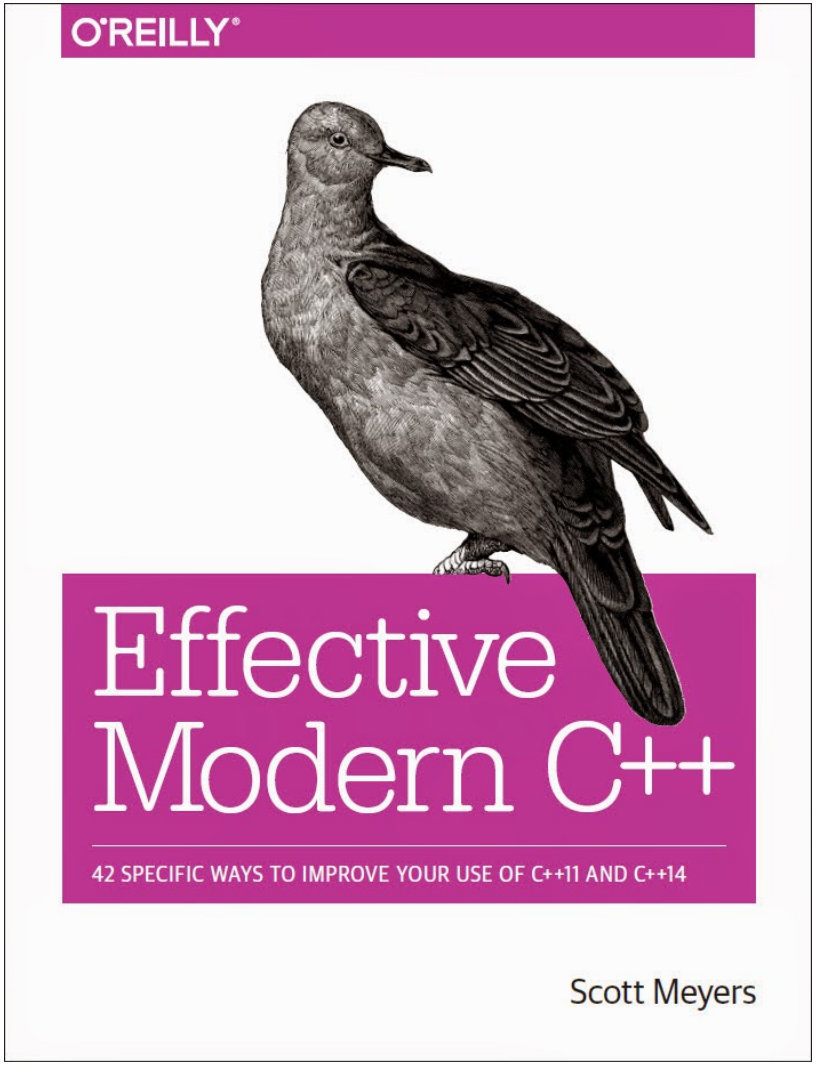
SOME THINGS TO CONSIDER WHEN USING MOVE SEMANTICS

Mantid Developer Meeting 2016

Anton Piccardo-Selg

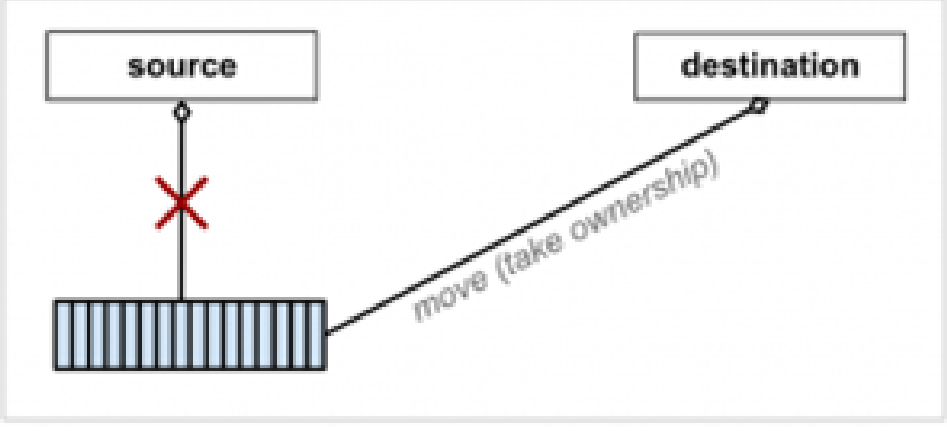
MOVE SEMANTICS 101



1. DEAD VARIABLES

std::move

```
std::unique_ptr<Foo> source(new Foo);
...
std::unique_ptr<Foo> destination = std::move(source);
```




```
...
source->doSomething(); // unspecified
```

2. SPECIAL MEMBER FUNCTION GENERATION

- move-constructor + move-assignment operator
- rule of five
- rules have become considerably more complex
- ...

Example:

```
Foo createFoo () {
  return Foo();
}
```

```
class Foo {
public:
  void doSomething() {...};
};

...
auto foo = createFoo(); // <- Move assignment
```

```
class Foo {
public:
  void doSomething() {...};
  ~Foo() {...}; // Want to do some logging
};
...
auto foo = createFoo(); // <- Copy assignment !!!
```

3. RVALUE OVERLOADS

Method overloading via rvalue references:

```
class Foo {
public:
        void doSomething(Bar& input);
        void doSomething(Bar&& input);
}
```

```
class Bar {
public:
        void doSomething()&;
        void doSomething()&&;
}
```

Example:

```
class MANTID_API_DLL MatrixWorkspace : public IMDWorkspace{
public:
        MatrixWorkspace_uptr clone() const {...}
...
};
...
MatrixWorkspace_uptr ws1 = ws->clone(); //Ok
...
```

```
MatrixWorkspace_sptr ws2 = ws1; // Error
```

```
MatrixWorkspace_sptr ws3 = ws1->clone(); // Ok
```

Overloaded of move-assignment operator

```
template<class Y, class D>
shared_ptr& operator=(std::unique_ptr<Y, D>&& r)
```