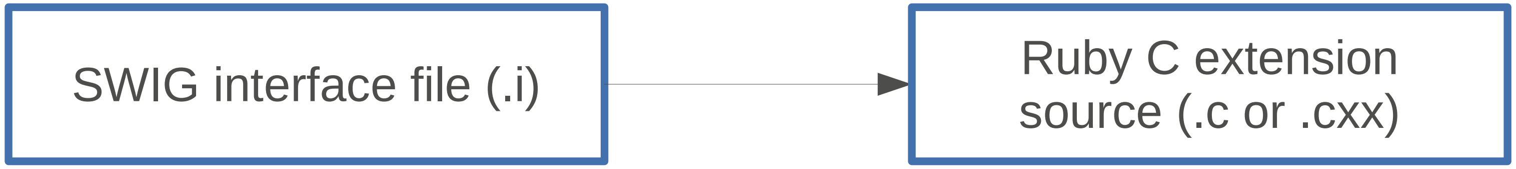
SWIG and Ruby

David Grayson
Las Vegas Ruby Meetup
2014-09-10

SWIG

- SWIG stands for:
  Simplified Wrapper and Interface Generator
- SWIG helps you access C or C++ code from 22 different languages, including Ruby

SWIG inputs and outputs



Simple C++ example

libdavid.h:

```cpp
#include <stdio.h>

class David
{
public:
    David(int x)
    {
        this->x = x;
    }

    void announce()
    {
        printf("David %d\n", x);
    }

    int x;
};
```

libdavid.i

```
%module "david"
%{
#include <libdavid.h>
%}

class David
{
public:
    David(int x);
    void announce();
    int x;
};
```

Compiling Simple C++ example

extconf.rb

```
require 'mkmf'
system('swig -c++ -ruby libdavid.i') or abort
create_makefile('david')
```

Commands to run:

```
$ ruby extconf.rb   # create libdavid_wrap.cxx and Makefile
$ make              # compile david.so
$ irb -r./david     # try it out

irb(main):001:0> d = David::David.new(4)
=> #<David::David:0x007f40090a5280 @__swigtype__="_p_David">
irb(main):002:0> d.announce
David 4
=> nil
```

(This example worked for me with SWIG 3.0.2 and Ruby 2.1.2.)

That example was pretty simple

- All code was in a .h file
- No external libraries
- Simple data types
- No consideration of deployment

...but SWIG has tons of features

C:

- *All* ISO C datatypes
- Global functions
- Global variables, constants
- Structures and unions
- Pointers
- (Multidimensional) arrays
- Pointers to functions
- Variable length arguments
- Typedefs
- Enums

C++:

- All C++ datatypes
- References
- Pointers to members
- Classes
- (Multiple) inheritance
- Overloaded functions
- Overloaded methods
- Overloaded operators
- Static members
- Namespaces
- Templates
- Nested classes

  ...

SWIG Typemaps

- Define custom ways to map between scripting-language types and C++ types.
- Can be used to add and remove parameters from of exposed functions.
  - http://stackoverflow.com/a/14427814/28128

SWIG supports 22 languages:

- Allegro CL
- C#
- CFFI
- CLISP
- Chicken
- D
- Go
- Guile
- Java
- Javascript
- Lua
- Modula-3
- Mzscheme
- OCAML
- Octave
- Perl
- PHP
- Python
- R
- Ruby
- Tcl
- UFFI

SWIG History

- Originally developed in 1995 by scientists in the Theoretical Physics Division at Los Alamos
- Actively developed today
  - https://github.com/swig/swig
  - 5 releases in the last 12 months, including 3.0.0

SWIG and freedom

- SWIG philosophy: programmers are smart and that tools should just stay out of their way.
- My question: which programmers?

C/C++ memory issues

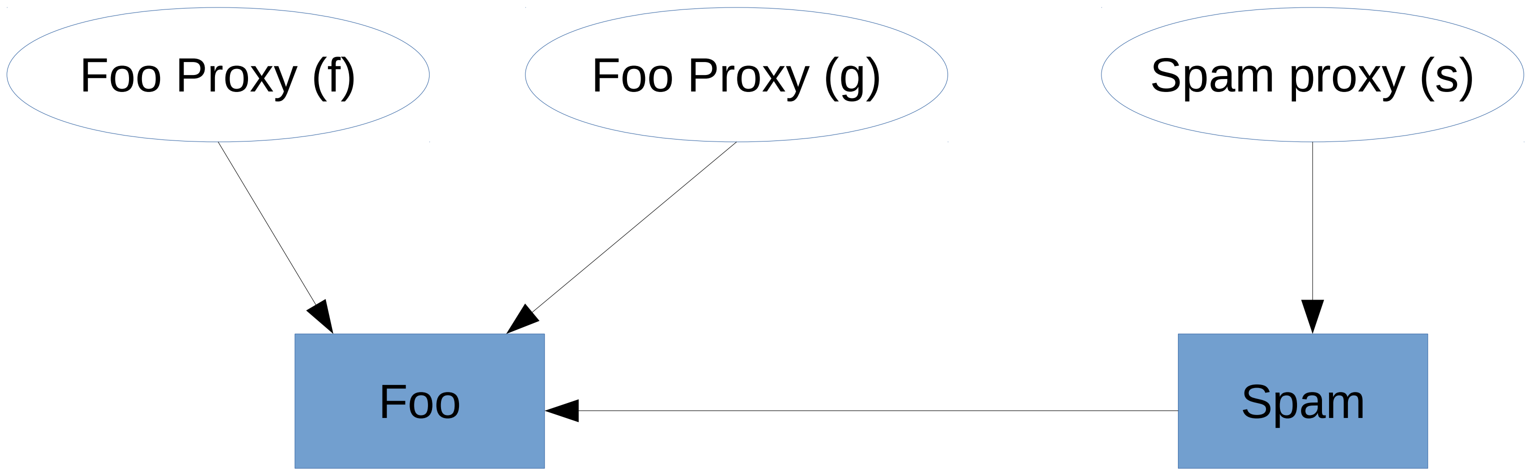
- Segmentation faults
- Memory leaks
- Freeing objects that might still be used
- Improper sharing of memory

Proxy Classes

- SWIG generates one proxy Ruby class for each wrapped C++ class
- Proxy instances know whether they *own* the underlying class.
- Ownership can change, sometimes automatically
- Not perfect

Proxy Classes (cont.)

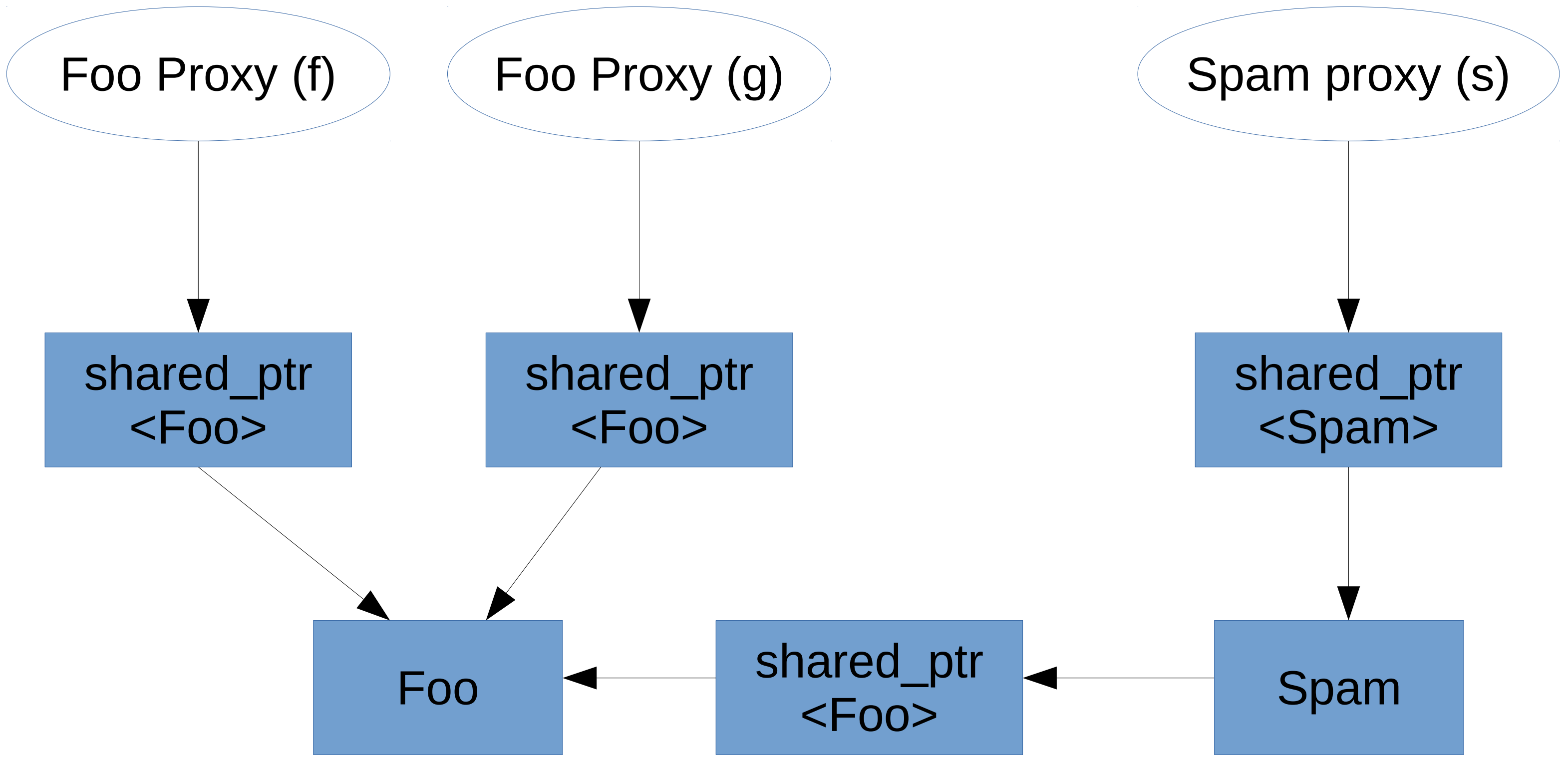
```
f = Foo.new     # Creates a new Foo
s = Spam.new    # Creates a new Spam
s.foo = f       # Stores a pointer to f inside s
g = s.foo       # Returns stored reference
```



http://www.swig.org/Doc3.0/SWIGPlus.html#SWIGPlus_nn40

Smart Pointers

- Uses reference counting
- C++11 standard: std::shared_ptr class



* Should actually be drawn as a cluster of 3 shared_ptr objects

Smart Pointers in SWIG

- Smart pointers available for some languages

```
$ find /usr/share/swig -name boost_shared_ptr.i
/usr/share/swig/3.0.2/csharp/boost_shared_ptr.i
/usr/share/swig/3.0.2/octave/boost_shared_ptr.i
/usr/share/swig/3.0.2/r/boost_shared_ptr.i
/usr/share/swig/3.0.2/java/boost_shared_ptr.i
/usr/share/swig/3.0.2/d/boost_shared_ptr.i
/usr/share/swig/3.0.2/python/boost_shared_ptr.i
```

- Alternative is the “ref” and “unref” features
  - I could not get it to work

http://www.swig.org/Doc3.0/SWIGPlus.html#SWIGPlus_ref_unref

Conclusion

- SWIG is really powerful, but it is too easy to create code with memory issues.
- I would like a recipe of simple rules to follow to avoid all memory issues.

Resources

- http://swig.org/
- http://www.davidegrayson.com/presentations/20131023-ruby-c/