

Ruby C Extensions

Fort Worth Ruby Brigade
Tuesday, February 26, 2013

Dan Swain @dantswain <http://dantswain.com>
dan.t.swain@gmail.com
Lead Software Engineer
Simpli.fi

Who am I and who cares what I have to say?

- Simpli.fi is a real-time bidding (RTB) platform
 - ~300,000 queries per second
 - < 50 msec latency
 - Built on Ruby!
- We use a lot of C extensions at Simpli.fi
 - Build highly specialized and performant components in C/C++
 - Integrate with our Ruby code base using extensions
 - String parsing, large data structure handling

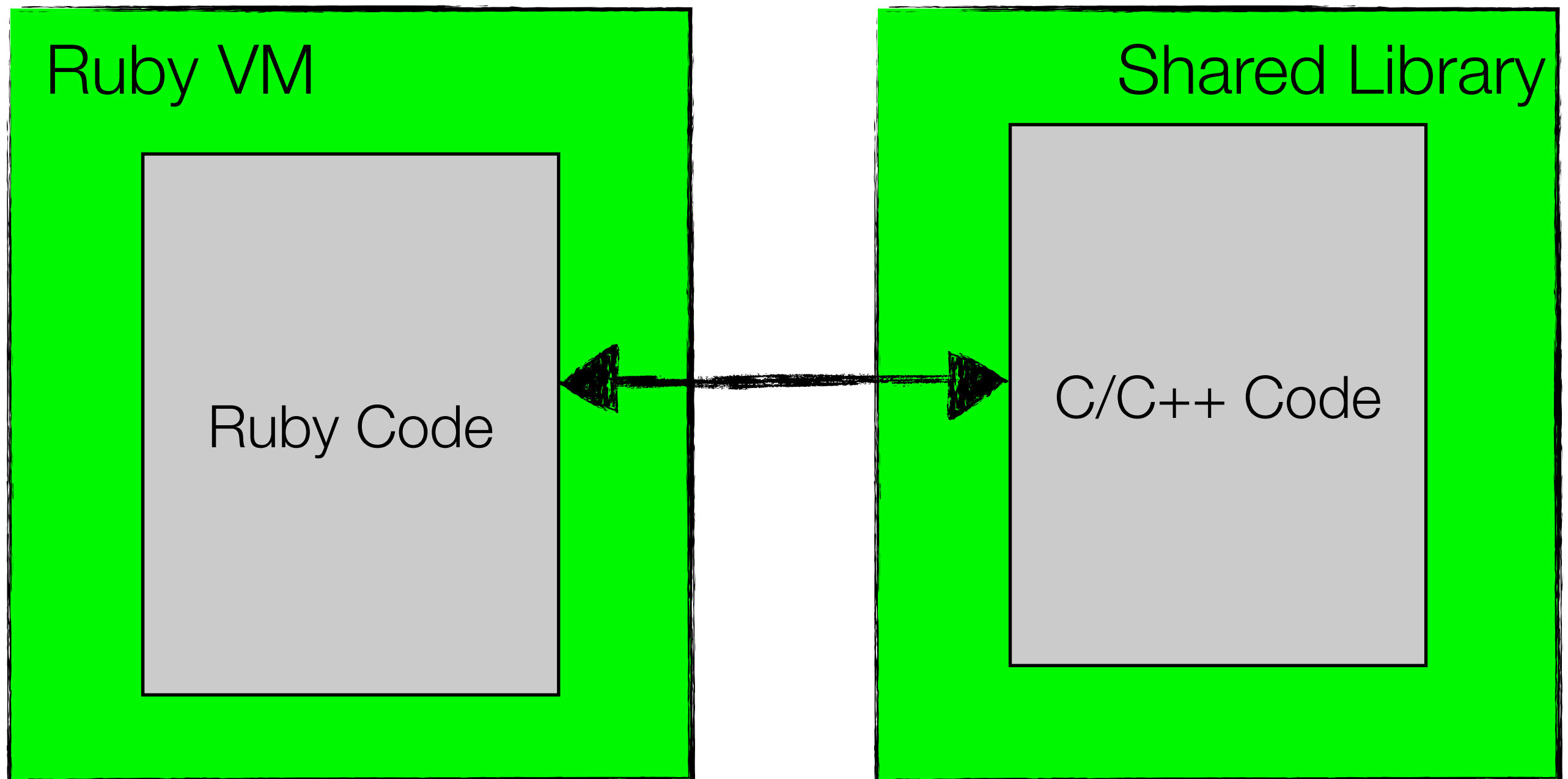


Outline

- What is a C extension?
- When (not) to use C extensions
- Basic procedure for creating a C extension
- The Ruby/C API by way of an example
- Creating a C extension gem (if time)

What is a C extension?

Basic Idea: Use the Ruby/C API to get Ruby to talk to C and vice versa



When to use C extensions

- Use C for raw performance
 - Repeated operations that can be highly-optimized (e.g., string parsing)
 - Optimally handling very large data structures
 - Fine-grained control over memory and garbage collection
 - Raw access to binary data
 - Numerical processing
- Use the Ruby/C API to interface with existing C/C++ libraries
 - Example: hiredis

C EXTENSION



When *not* to use C extensions

Don't try to rewrite MRI/YARV (it's already in C)

Don't try to write around bad architecture choices



Compatibility / Gotchas

- The API was developed for MRI (“normal” Ruby) (now works with YARV)
- Limited compatibility with other C/C++-based VMs (Rubinius)
- Notably does NOT work with JRuby (Java-based)
- The API is C, but can be interfaced with C++ code
- Lose some speed at the interface layer

Basic Procedure

1. Create your C source code - 'my-extension.c'

- Your algorithm
- Interface / API code

2. Create an 'extconf.rb' file

3. `ruby extconf.rb` (generates Makefile)

4. `make` (generates my-extension.bundle)

5. In Ruby: `require 'my-extension'`

6. Optional: Create wrapper code in Ruby

OS X

```
> RbConfig::CONFIG['DLEXT']  
=> "bundle"
```

Linux

```
> RbConfig::CONFIG['DLEXT']  
=> "so"
```

C/Ruby API

- `#include <ruby.h>`
- `VALUE` - Ruby object (actually a pointer)
- `ID` - Symbols (use `SYM2ID(VALUE id)` or `rb_inter(const char* symbol_name)`)
- Functions: Typically `rb_<function>`
- Conversions: `FIX2INT/INT2FIX`, `NUM2DBL/DBL2NUM`, etc...
- Strings: `char* StringValueCStr(VALUE)`, `VALUE rb_str_new2(const char*)`
- It's possible to wrap structs as Ruby objects

C/Ruby API

```
#include <ruby.h>

static VALUE rb_cFoo;

static VALUE dostuff(VALUE self, VALUE arg);

void Init_foo(void)
{
    rb_cFoo = rb_define_class("Foo", rb_cObject);
    rb_define_method(rb_cFoo, "dostuff", dostuff, 1);
}

static VALUE dostuff(VALUE self, VALUE arg)
{
    return arg;
}
```

```
class Foo
  def dostuff arg
    arg
  end
end
```

Resources

- Pick-axe chapter: http://media.pragprog.com/titles/ruby3/ext_ruby.pdf
- <https://github.com/ruby/ruby/blob/trunk/README.EXT>
- Ruby source code cross-reference: <http://rxr.whitequark.org/mri/source>