JRuby and the JVM

Christopher Mark Gore

cgore.com

Monday, May 8, AD 2017

Ruby is my ~~most~~ ~~second~~ third favorite programming language of all time.

1. My own super-awesome programming language, Teepee
   (but it's not that awesome just yet)
2. Common Lisp
3. Ruby
4. C
5. Clojure

   …

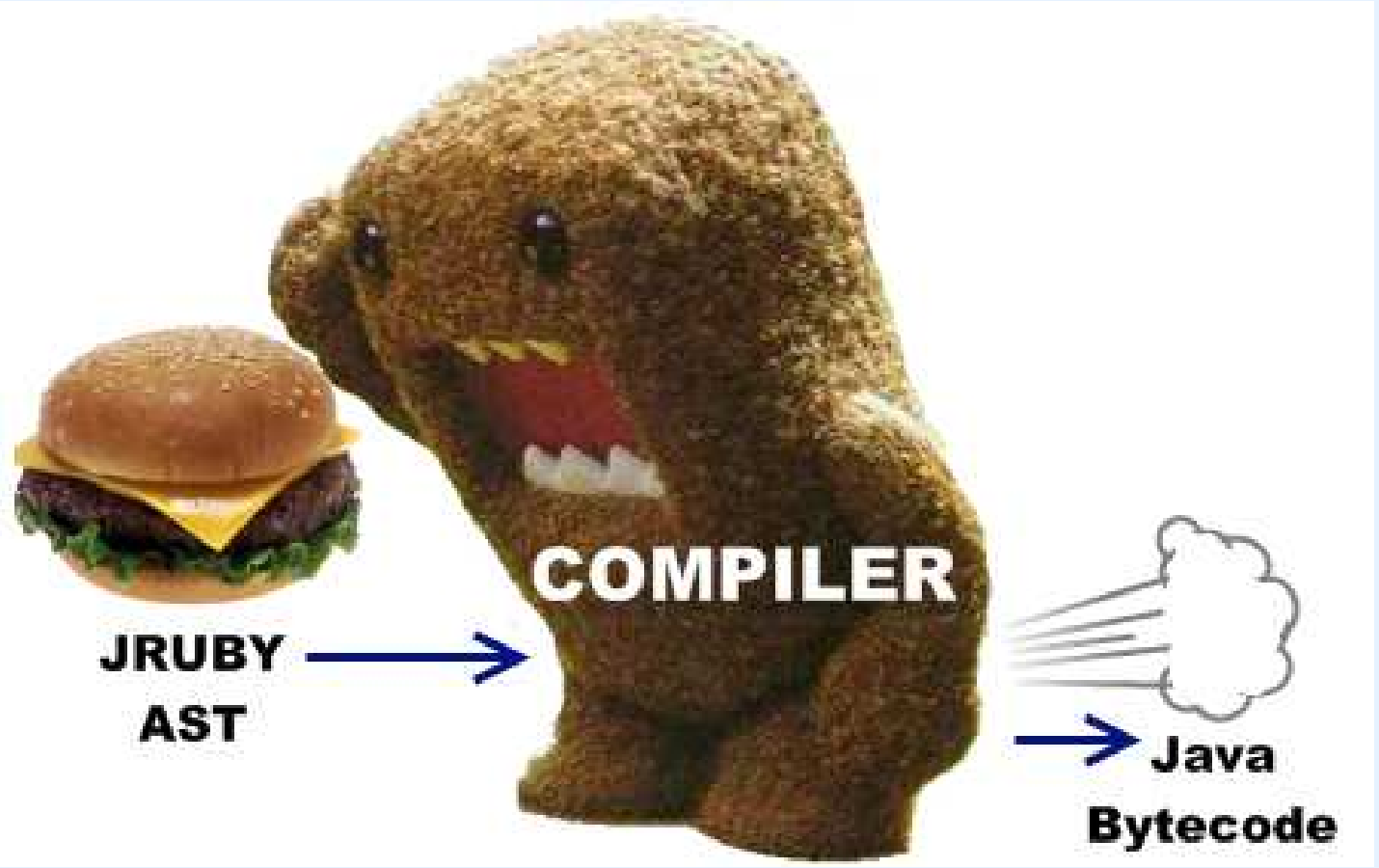
999. Java

JRuby is Ruby, on the JVM.

Ruby on the JVM?
Why would we want Ruby on the JVM?

Because there's tons of great libraries for everything.

I found this image on the internet that describes very accurately how JRuby works.



Getting Started

1. Download it from `http://jruby.org/download`
   ...or `brew install jruby` on OS X
   ...or `rvm install jruby` if you use RVM
2. Run `jruby` from your shell
3. Make code!

CON: The JVM takes forever to start up

```
$ time jruby -e "puts 'hi'"
hi

real 0m1.761s
user 0m4.800s
sys 0m0.235s
$ time ruby -e "puts 'hi'"
hi

real 0m0.595s
user 0m0.054s
sys 0m0.050s
```

CON: Until the JIT kicks in it's actually considerably slower than MRI.

CON: Even after the JIT kicks in, it's not really that much faster.

CON: THE JVM WANTS ALL OF YOUR RAM AND IT WANTS IT NOW.

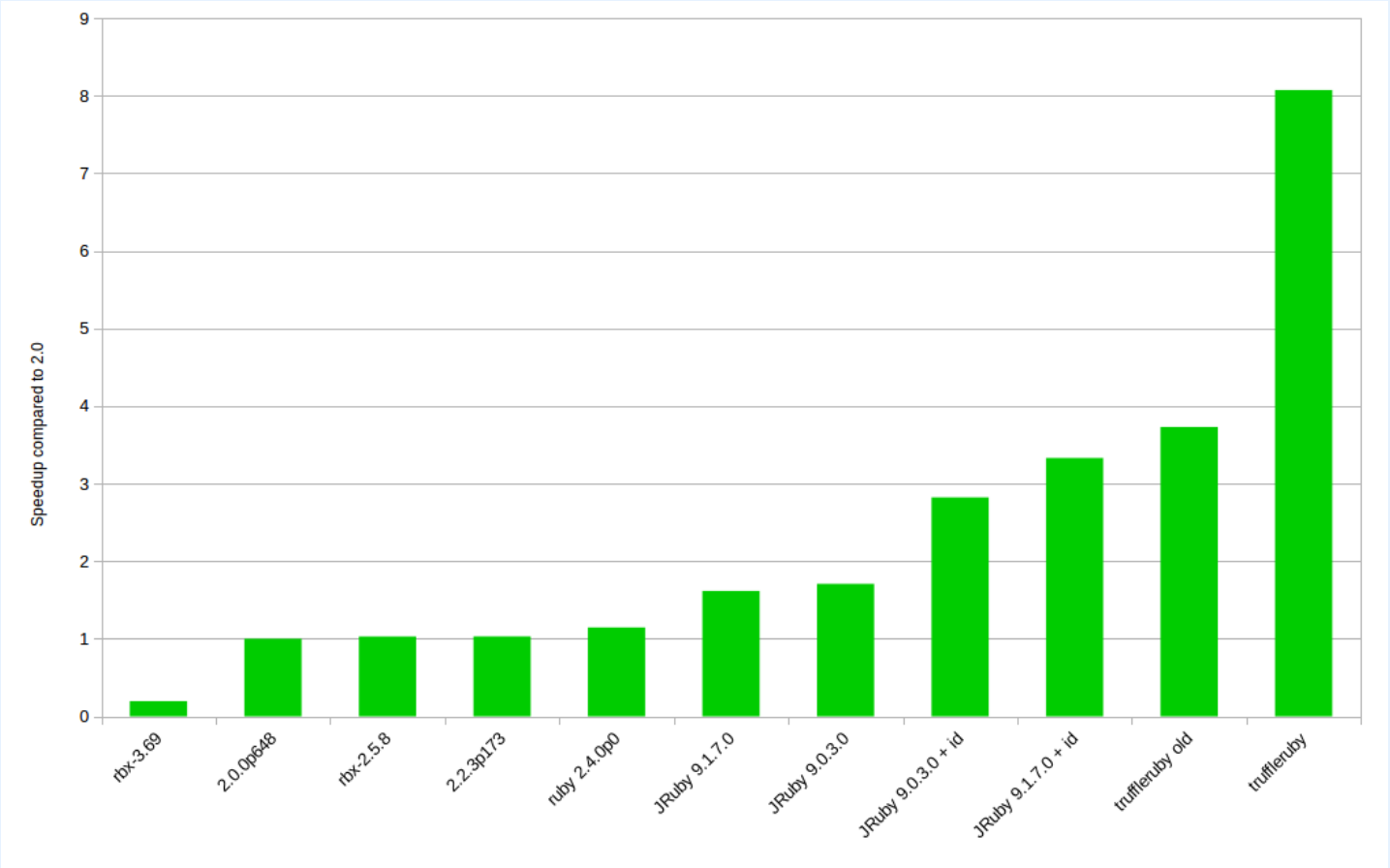
```
%CPU %MEM      VSZ    RSS  COMMAND
 0.0  0.1  2475044   9300  ruby -e sleep 60
%CPU %MEM      VSZ    RSS  COMMAND
 0.0  1.0  8325372 164520  ... org.jruby.Main -e sleep 60
```

VSZ: virtual memory size, all memory that the process can access, including memory that is swapped out and memory that is from shared libraries.

RSS: resident set size, how much memory is allocated to that process and is in RAM.

No (practical) C extension support, just FFI and Java stuff.

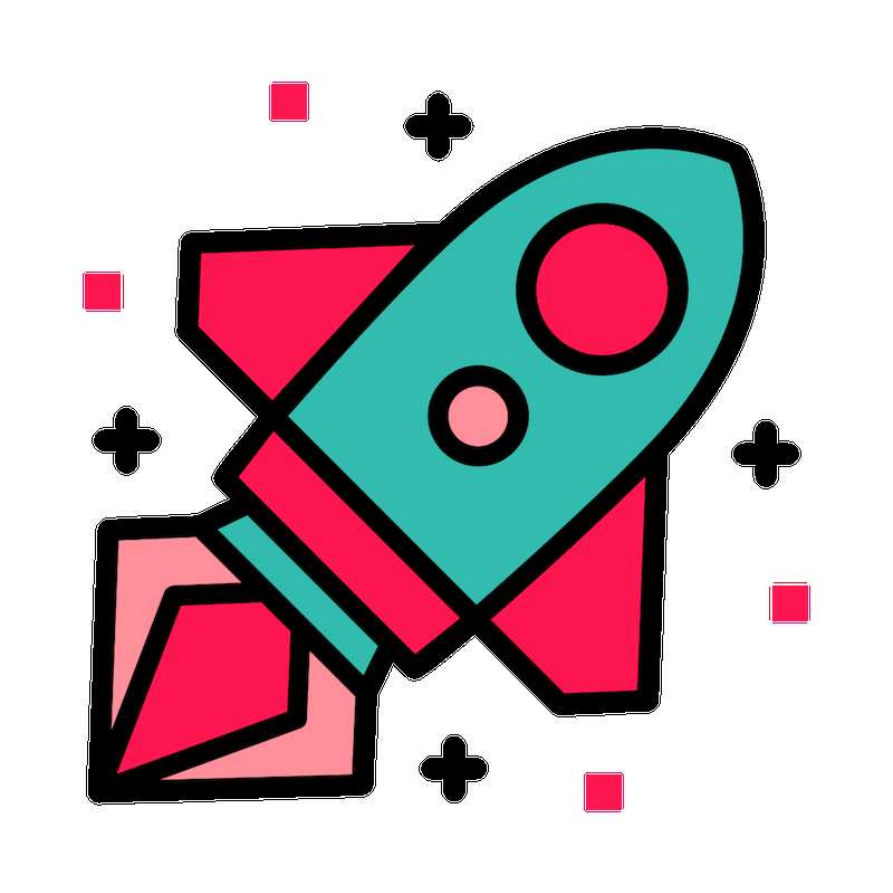
PRO: It's often faster than MRI.



https://pragtob.wordpress.com

/2017/01/24/benchmarking-a-go-ai-in-ruby-cruby-vs-rubinius-vs-jruby-vs-truffle-a-year-later/

. . . Although, nowhere near as fast as TruffleRuby apparently, also on the JVM.



http://chrisseaton.com/rubytruffle/

So let's just use TruffleRuby, right?

- No OpenSSL support
- No Nokogiri
- No ActiveRecord device drivers
- Only some of the Rails test suite passes

Maybe in a few years?

So let's stay with JRuby for now.
You can run (nearly) any normal Ruby code.

```
$ jruby -e '5.times {|i| puts "hi #{i}"}'
hi 0
hi 1
hi 2
hi 3
hi 4
```

It's easy to use use in scripts.

```
1 #! /usr/bin/env jruby
2 # -*- mode: ruby -*-
3
4 puts "Hello, JVM!"
```

Most Ruby gems are available and work.

```
$ jgem install nokogiri
```

```
#! /usr/bin/env jruby
# -*- mode: ruby -*-
require 'nokogiri'
doc  = Nokogiri::XML \
          "<root>
␣␣␣␣␣␣␣␣␣␣␣␣<aliens>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<alien>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<name>Alf</name>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</alien>
␣␣␣␣␣␣␣␣␣␣␣␣</aliens>
␣␣␣␣␣␣␣␣␣␣</root>"
puts doc.xpath("//name").first.content # Alf
```

Let's play with Java.

```
1  #! /usr/bin/env jruby
2  # -*- mode: ruby -*-
3  require 'java' # you want Java
4  # Java classes
5  frame = javax.swing.JFrame.new "Window"
6  label = javax.swing.JLabel.new "Hello"
7  # Java methods
8  frame.add label
9  frame.setDefaultCloseOperation \
10         javax.swing.JFrame::EXIT_ON_CLOSE
11 frame.pack
12 frame.setVisible true
```

But I've got my own really awesome Java code that there's no way I'd ever be able to reimplement in Ruby, it's just too awesome.

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World");
4     }
5 }
```

```
$ javac HelloWorld.java
$ jar cvfe HelloWorld.jar HelloWorld HelloWorld.class
```

Don't worry, we can get to it.

```
1 #! /usr/bin/env jruby
2 # -*- mode: ruby -*-
3 require 'java'
4 require './HelloWorld.jar'
5 Java::HelloWorld.main [""]
```

But.Java.Classes.Are.Namespaced.Forever.Deep

```
1  #! /usr/bin/env jruby
2  # -*- mode: ruby -*-
3  require 'java'
4  java_import javax.swing.JFrame
5  java_import javax.swing.JLabel
6  frame = JFrame.new "Window"
7  label = JLabel.new "Hello"
8  frame.add label
9  frame.setDefaultCloseOperation \
10         JFrame::EXIT_ON_CLOSE
11 frame.pack
12 frame.setVisible true
```

You don't need sillyCamelCaseNonsense or getThatThing or setThatThing.

```
x.getSomething                          x.something
x.setSomething(newValue)   x.something = new_value
x.isSomething                           x.something?
```

So our code looks a lot more reasonable now.

```
1  #! /usr/bin/env jruby
2  # -*- mode: ruby -*-
3  require 'java'
4  java_import javax.swing.JFrame
5  java_import javax.swing.JLabel
6  frame = JFrame.new "Window"
7  label = JLabel.new "Hello"
8  frame.add label
9  frame.default_close_operation =
10   JFrame::EXIT_ON_CLOSE
11 frame.pack
12 frame.visible = true
```

You can implement Java interfaces with Ruby classes.

```
1 class SomeJRuby
2   include java.lang.Runnable
3   include java.lang.Comparable
4
5   # ... do stuff ...
6 end
```

`jrubyc`, for when you want to get to your JRuby from Java (or Clojure, or Scala, or ...)

```ruby
class Foo
  def bar(a, b)
    puts a + b
  end
end
```

```
$ jrubyc --javac ruby_foo.rb
```

Now there will be `Foo.java` and `Foo.class` files for you to use in your other JVM stuff.

What does this mean?

- We can write a Ruby on Rails app …
- And then move it over to JRuby on the JVM …
- And then have real multi-threading …
- And have Clojure libraries that we can use within our Rails app.

Questions?