Engine Yard™

Brian Ford

Rubinius Developer

Who wants a faster Ruby?

Yugo

AMK SKOPJE
AMCM
52
FUCHS

The experiment

```
i = 0
while i < 10_000
  j = 0
  while j < 10_000
    j += 1
    m = i + 1
  end
  i += 1
end
```

45.149 s

```c
int main(int argc, char *argv[]) {
  int i, j, m;

  for (i = 0; i < 10000; i++) {
    for (j = 0; j < 10000; j++) {
      m = i + 1;
    }
  }
}
```

0.368 s

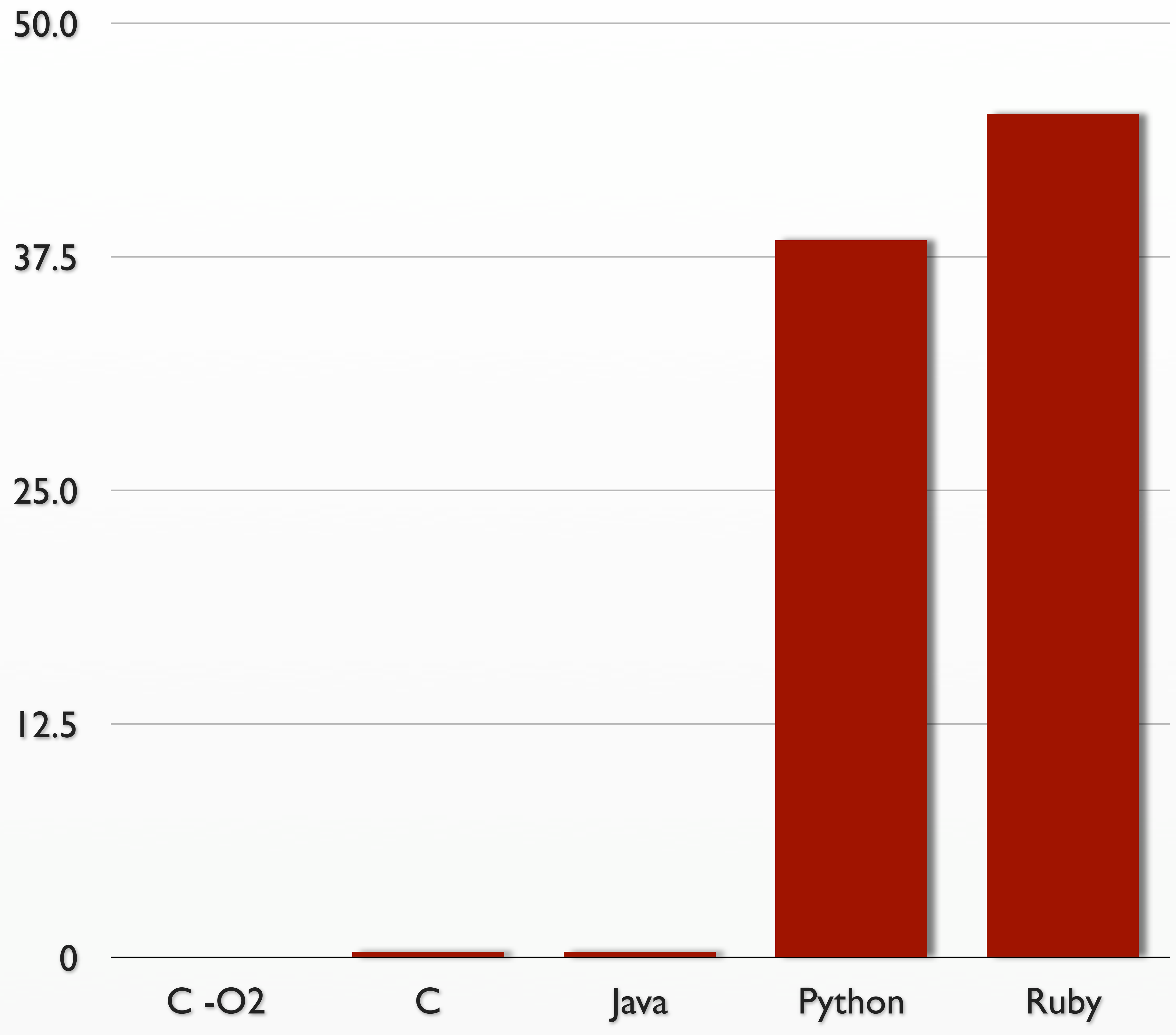
0.056 s

```java
class Bench {
  public static void bench() {
    int i, j, m = 0;

    for (i = 0; i < 10000; i++) {
      for (j = 0; j < 10000; j++) {
        m = i + 1;
      }
    }
  }

  public static void main(String[] args) {
    new Bench().bench();
  }
}
```

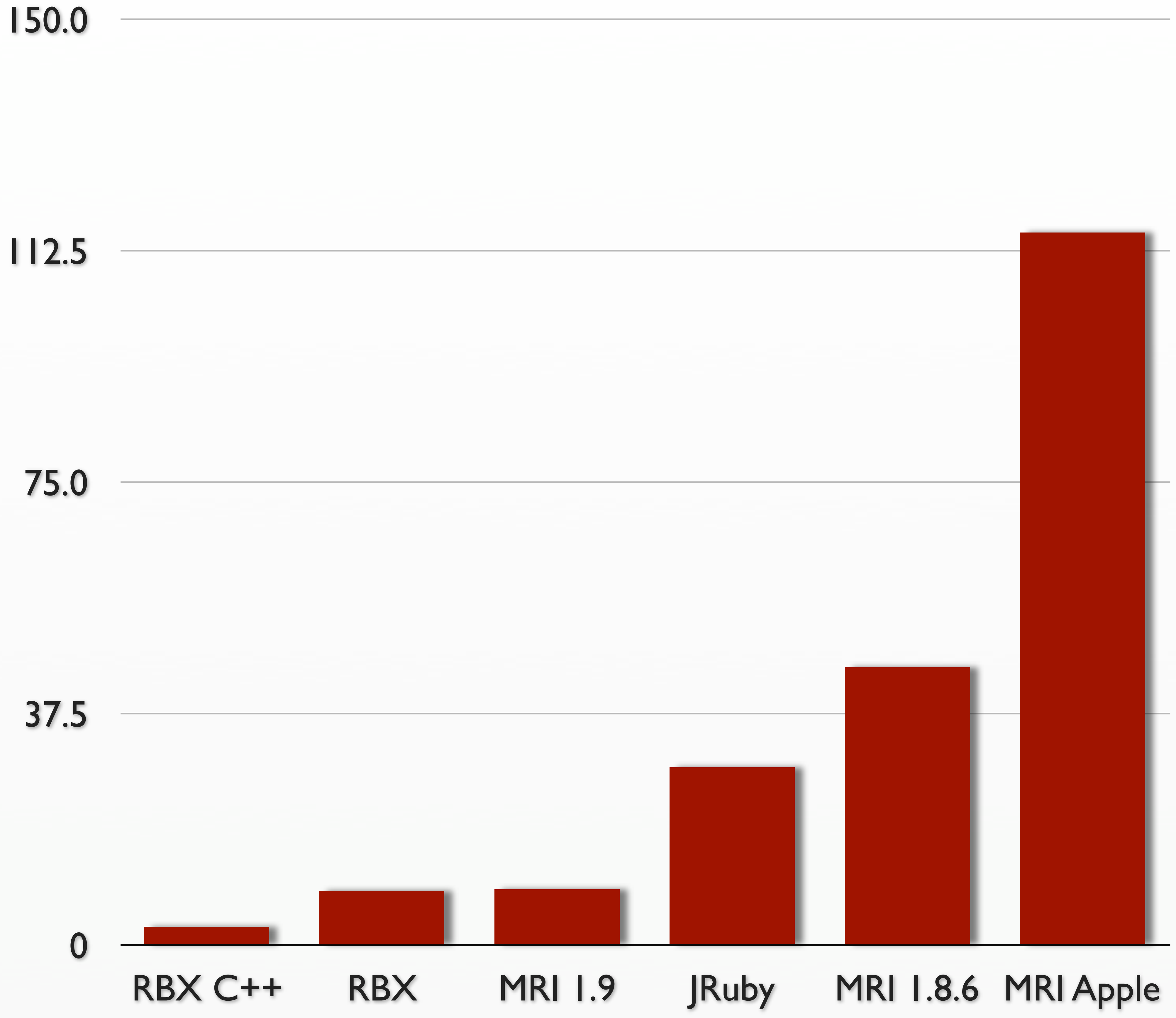
0.343 s

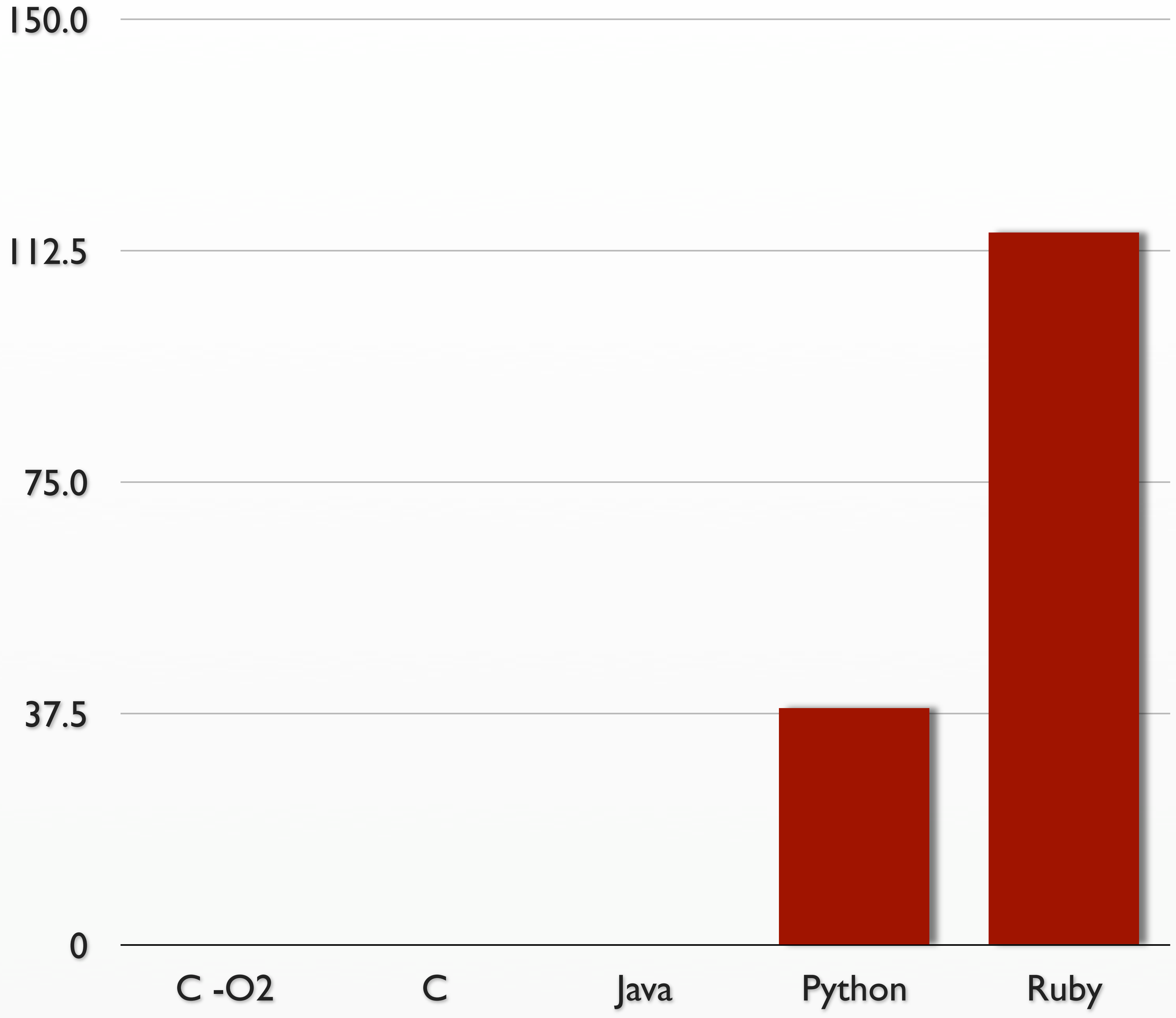
```
i = 0
while i < 10000:
  j = 0
  while j < 10000:
    j += 1
    m = i + 1
  i += 1
```

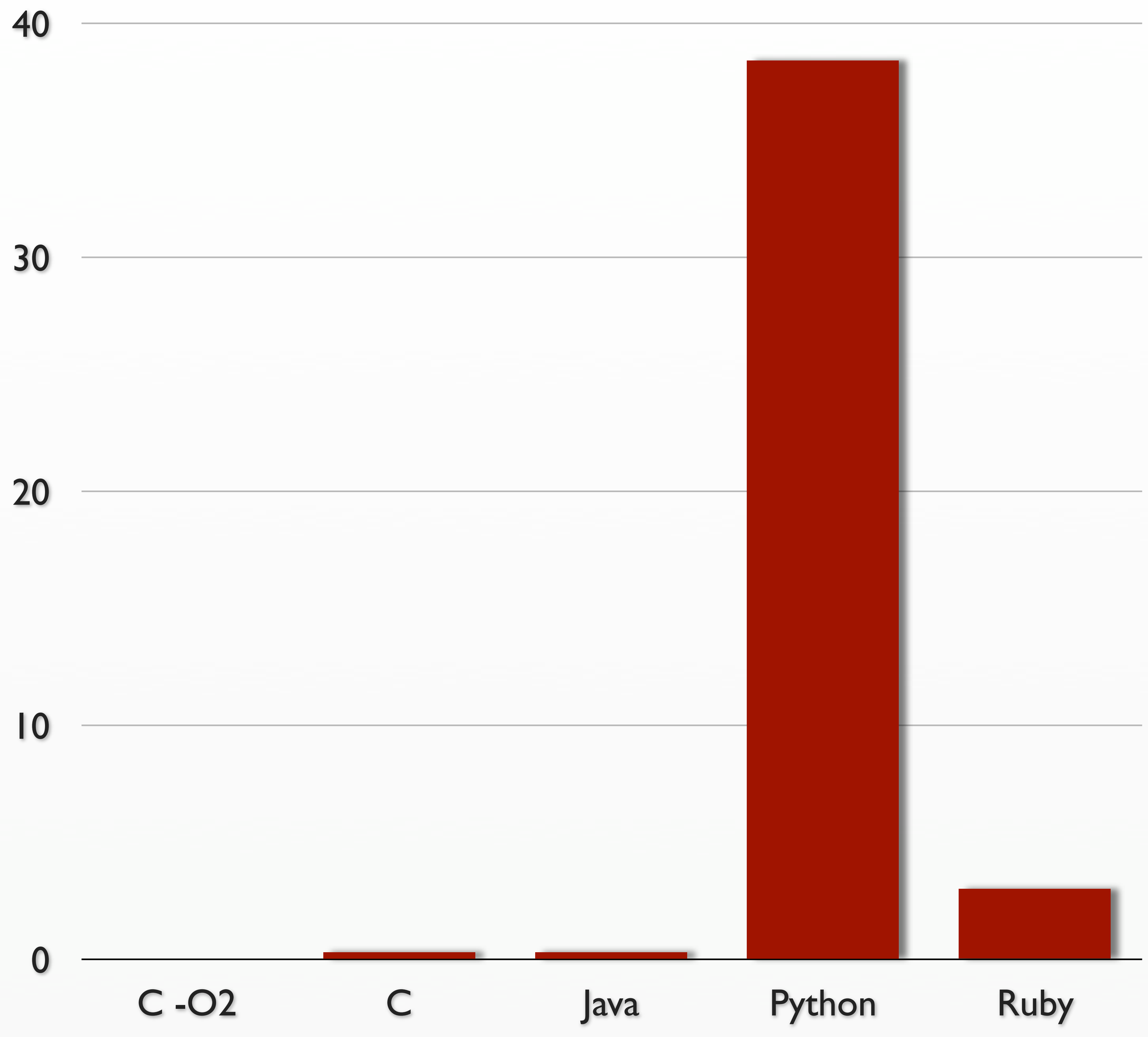
38.437 s


50.0
37.5
25.0
12.5
0
C -O2
C
Java
Python
Ruby

The plot thickens


150.0
112.5
75.0
37.5
0
RBX C++
RBX
MRI 1.9
JRuby
MRI 1.8.6
MRI Apple


150.0
112.5
75.0
37.5
0
C -O2
C
Java
Python
Ruby



40
30
20
10
0
C -O2
C
Java
Python
Ruby

That’s a d*mn lie

There are three kinds of lies:
lies, damn lies, and statistics

The secret to Ruby performance

The secret to Ruby performance

Do Less

Profile to see what’s happening

MRI

```
$ ruby benchmark/borasky/bm_MatrixBenchmark.rb 32
Hilbert matrix of dimension 32 times its inverse =
identity? true
  3.930000   0.030000   3.960000 (  4.026112)
```

Rubinius

```
$ bin/rbx benchmark/borasky/bm_MatrixBenchmark.rb 32
Hilbert matrix of dimension 32 times its inverse =
identity? true
  4.870182   0.000000   4.870182 (  4.870170)
```

MRI

```
$ ruby -rprofile benchmark/borasky/bm_MatrixBenchmark.rb 32
Hilbert matrix of dimension 32 times its inverse = identity? true
266.280000  20.810000 287.090000 (292.106996)
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 21.19    56.43     56.43   112909     0.50     0.84  Integer#gcd
 11.52    87.10     30.67   112909     0.27     1.38  Rational#reduce
  7.77   107.79     20.69   135352     0.15     0.21  Rational#initialize
  7.61   128.05     20.26   122242     0.17     1.70  Rational#*
  5.10   141.64     13.59     3073     4.42    85.18  Integer#upto
  4.23   152.91     11.27   643741     0.02     0.02  Kernel.kind_of?
  3.91   163.33     10.42   112909     0.09     1.49  Object#Rational
  3.50   172.66      9.33   211699     0.04     0.06  Comparable.>
  3.32   181.50      8.84   332763     0.03     0.03  Fixnum#%
  3.02   189.54      8.04   225061     0.04     0.11  Fixnum#*
  2.75   196.86      7.32   464221     0.02     0.02  Fixnum#>

...
```

Rubinius

```
$ bin/rbx -p benchmark/borasky/bm_MatrixBenchmark.rb 32
Hilbert matrix of dimension 32 times its inverse = identity? true
   5.316938   0.000000   5.316938 (  5.316942)
Total slices: 1263, 5294473 clocks

=== FLAT PROFILE ===

 % time   slices   name
  24.47      309    VM.primitive => bignum_mod
  14.25      180    VM.primitive => bignum_div
  13.14      166    VM.garbage_collection
   6.18       78    Bignum#%
   3.80       48    VM.primitive => regexp_search_region
   3.72       47    Integer#gcd
   3.64       46    Rational#initialize
   3.64       46    Rational#*
   3.40       43    Numeric#math_coerce
   2.30       29    VM.primitive => opt_kind_of
   2.22       28    Numeric#div

...
```

What’s going on?

```
 % time   slices   name
  24.06      306    VM.primitive => bignum_mod
  14.62      186    VM.primitive => bignum_div
  12.58      160    VM.garbage_collection
   6.84       87    Bignum#%
   4.25       54    Integer#gcd
   4.09       52    Numeric#math_coerce
   3.85       49    VM.primitive => regexp_search_region
   3.38       43    Rational#initialize
   2.28       29    VM.primitive => opt_kind_of
   2.12       27    Rational.reduce
   1.97       25    Rational#*
   1.73       22    Numeric#%
   1.73       22    VM.primitive => fixnum_mul
   1.73       22    VM.primitive => numeric_coerce
   1.65       21    Object#Rational
   1.57       20    Class#new
   1.49       19    Numeric#div
   1.42       18    Numeric#abs
   1.26       16    VM.primitive => bignum_equal
   1.18       15    Matrix#inverse_from
   1.18       15    VM.primitive => set_ivar
   1.02       13    String#=~
```

VM

```
% time   slices   name
 24.06      306    VM.primitive => bignum_mod
 14.62      186    VM.primitive => bignum_div
 12.58      160    VM.garbage_collection
  6.84       87    Bignum#%
  4.25       54    Integer#gcd
  4.09       52    Numeric#math_coerce
  3.85       49    VM.primitive => regexp_search_region
  3.38       43    Rational#initialize
  2.28       29    VM.primitive => opt_kind_of
  2.12       27    Rational.reduce
  1.97       25    Rational#*
  1.73       22    Numeric#%
  1.73       22    VM.primitive => fixnum_mul
  1.73       22    VM.primitive => numeric_coerce
  1.65       21    Object#Rational
  1.57       20    Class#new
  1.49       19    Numeric#div
  1.42       18    Numeric#abs
  1.26       16    VM.primitive => bignum_equal
  1.18       15    Matrix#inverse_from
  1.18       15    VM.primitive => set_ivar
  1.02       13    String#=~
```

GC

```
0.94         12    Regexp.last_match=
0.86         11    VM.primitive => bignum_lt
0.79         10    MethodContext#const_defined?
0.71          9    Rational#-
0.63          8    Rational.new!
0.63          8    Rational#+
0.55          7    VM.primitive => bignum_add
0.55          7    Module#normalize_const_name
0.55          7    Numeric#floor
0.55          7    Matrix#*
0.47          6    Kernel#respond_to?
0.47          6    Module#valid_const_name?
0.47          6    Integer#upto
0.47          6    Module#normalize_name
0.39          5    MethodContext.current
0.39          5    VM.primitive => bignum_sub
0.39          5    Matrix#[]
0.39          5    VM.primitive => fixnum_modulo
0.31          4    VM.primitive => array_aref
0.31          4    Regexp#match_from
0.31          4    VM.primitive => bignum_mul
0.31          4    MethodContext#sender
0.31          4    VM.primitive => load_file
```

standard lib

```
0.94        12    Regexp.last_match=
0.86        11    VM.primitive => bignum_lt
0.79        10    MethodContext#const_defined?
0.71         9    Rational#-
0.63         8    Rational.new!
0.63         8    Rational#+
0.55         7    VM.primitive => bignum_add
0.55         7    Module#normalize_const_name
0.55         7    Numeric#floor
0.55         7    Matrix#*
0.47         6    Kernel#respond_to?
0.47         6    Module#valid_const_name?
0.47         6    Integer#upto
0.47         6    Module#normalize_name
0.39         5    MethodContext.current
0.39         5    VM.primitive => bignum_sub
0.39         5    Matrix#[]
0.39         5    VM.primitive => fixnum_modulo
0.31         4    VM.primitive => array_aref
0.31         4    Regexp#match_from
0.31         4    VM.primitive => bignum_mul
0.31         4    MethodContext#sender
0.31         4    VM.primitive => load_file
```

VM/core lib

```
0.24          3    VM.primitive => allocate
0.24          3    VM.primitive => bignum_gt
0.24          3    Numeric#*
0.24          3    VM.primitive => bignum_neg
0.24          3    SymbolTable#symbol_to_string
0.24          3    VM.primitive => opt_push_self
0.24          3    VM.primitive => fixnum_div
0.24          3    Fixnum#%
0.24          3    VM.primitive => opt_push_my_field
0.24          3    VM.primitive => get_ivar
0.16          2    Symbol#to_s
0.16          2    VM.primitive => string_dup
0.16          2    Module#===
0.16          2    VM.primitive => at
0.16          2    VM.primitive => array_aset
0.16          2    Module#const_defined?
0.16          2    MatchData#begin
0.16          2    Rational#coerce
0.16          2    VM.primitive => fastctx_get_field
0.08          1    Object#hilbert
0.08          1    Rational#/
0.08          1    VM.primitive => sampler_stop
0.08          1    Proc#call
```

core lib

core lib

```
0.24          3     VM.primitive => allocate
0.24          3     VM.primitive => bignum_gt
0.24          3     Numeric#*
0.24          3     VM.primitive => bignum_neg
0.24          3     SymbolTable#symbol_to_string
0.24          3     VM.primitive => opt_push_self
0.24          3     VM.primitive => fixnum_div
0.24          3     Fixnum#%
0.24          3     VM.primitive => opt_push_my_field
0.24          3     VM.primitive => get_ivar
0.16          2     Symbol#to_s
0.16          2     VM.primitive => string_dup
0.16          2     Module#===
0.16          2     VM.primitive => at
0.16          2     VM.primitive => array_aset
0.16          2     Module#const_defined?
0.16          2     MatchData#begin
0.16          2     Rational#coerce
0.16          2     VM.primitive => fastctx_get_field
0.08          1     Object#hilbert
0.08          1     Rational#/
0.08          1     VM.primitive => sampler_stop
0.08          1     Proc#call
```

user code

```
0.00        0    Compiler::MethodDescription#to_cmethod
0.00        0    Enumerable#collect
0.00        0    Array#reverse_each
0.00        0    Range#each
0.00        0    Process.times
0.00        0    Array#collect
0.00        0    Matrix::Scalar.__class_init__
0.00        0    Benchmark::Tms#to_s
0.00        0    IO#puts
0.00        0    Exception2MessageMapper#bind
0.00        0    Compiler.compile_string
0.00        0    CompiledMethod#as_script
0.00        0    Benchmark::Tms#format
0.00        0    String#gsub
0.00        0    Matrix#inv
0.00        0    Compile.compile_string
0.00        0    Object#do_benchmark
0.00        0    Kernel#extend
0.00        0    Array#map!
0.00        0    main.__script__
0.00        0    Bignum#+
0.00        0    Compiler::Generator#encode_literals
0.00        0    Struct::Tms.new
0.00        0    Kernel#require
```

compiler

compiler

Components

- virtual machine
- garbage collector
- compiler
- core library
- standard library
- user code

Challenges to making Ruby fast

- inefficient algorithms
- method dispatch
- VM operation
- garbage collection

Case-study: class String

- a loop instead of a block
- `str[0..n]` vs `str[0,n+1]`
- `str[a] = char` vs `a + char + b`
- composable primitives: `“ “ * 50`

Moral of the story:

it’s easy to make Ruby 1000x slower than it needs to be

Method dispatch

```ruby
class F
  def bell
    puts 'zzring'
  end
end
```

```ruby
module N
  def bell
    puts 'ding-dong'
  end
end

class A < F
  include N
end
```

```ruby
class C
  def method_missing(sym, *args)
    puts 'ding' if sym == :bell
  end
end
```

```ruby
class B < F
  def bell
    puts 'zwink'
  end
end
```

```ruby
d = Object.new
def d.bell
  puts 'doink'
end
```

```ruby
a = A.new
b = B.new
c = C.new
e = F.new

a.bell  # => ding-dong
b.bell  # => zwink
c.bell  # => ding
d.bell  # => doink
e.bell  # => zzring
```

```
$ bin/rbx describe dispatch.rb
...

0118:  push_local                         :a
0120:  send_method                        #<SendSite:0x849
name=bell hits=0 misses=0>
0122:  pop
0123:  push_local                         :b
0125:  send_method                        #<SendSite:0x869
name=bell hits=0 misses=0>
0127:  pop
0128:  push_local                         :c
0130:  send_method                        #<SendSite:0x889
name=bell hits=0 misses=0>
0132:  pop
0133:  push_local                         :d
0135:  send_method                        #<SendSite:0x8a9
name=bell hits=0 misses=0>
0137:  pop
0138:  push_local                         :e
0140:  send_method                        #<SendSite:0x8c9
name=bell hits=0 misses=0>
0142:  pop
```

Inside the VM

SendSite

- monomorphic inline cache
- FFI methods
- Primitives

Bytecode interpreter

A bytecode interpreter is (sort of) a formalism for converting stack manipulations into register allocations

Simple Instruction Set

```
set:   reg[op1] = op2
add:   reg[op1] = reg[op1] + op2
ret:   printf(“=> %d\n”, reg[op1]
```

Interpreter dispatch loop

```
fetch instruction
jump code[instruction]
<set code>
jump dispatch
fetch instruction
jump code[instruction]
<add code>
jump dispatch
fetch instruction
jump code[instruction]
<ret code>
```

Direct threaded

```
fetch instruction
jump instruction
<set code>
fetch instruction
jump instruction
<add code>
fetch instruction
jump instruction
<ret code>
```

Super-instructions

```
<set code>
<add code>
<ret code>
```

Super-instructions

Fuses the opcodes, essentially *eliminating* dispatch overhead

What is LLVM?

- Compilation strategy
- Virtual instruction set
- Compiler infrastructure

Native code generation

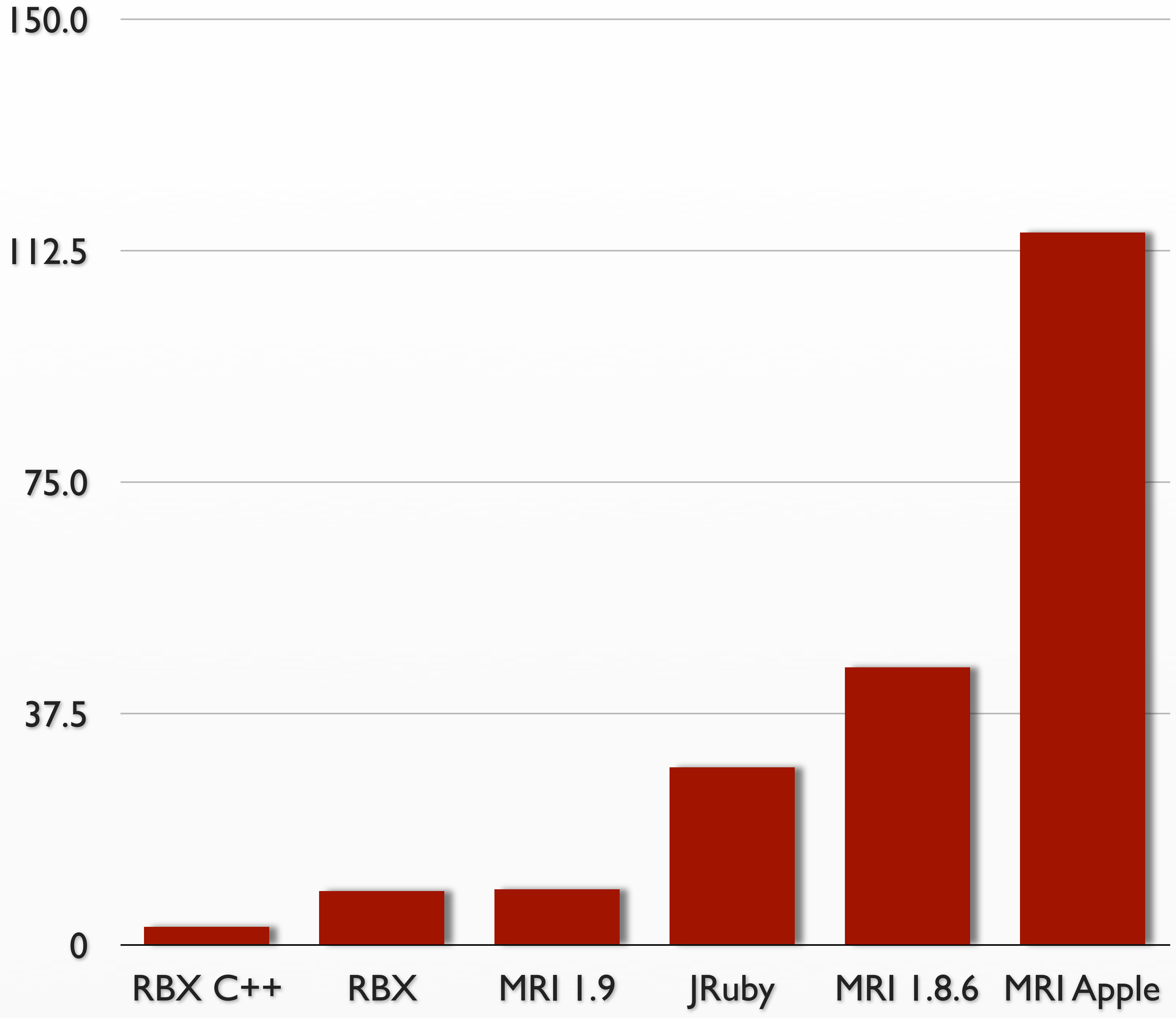
Native code generation

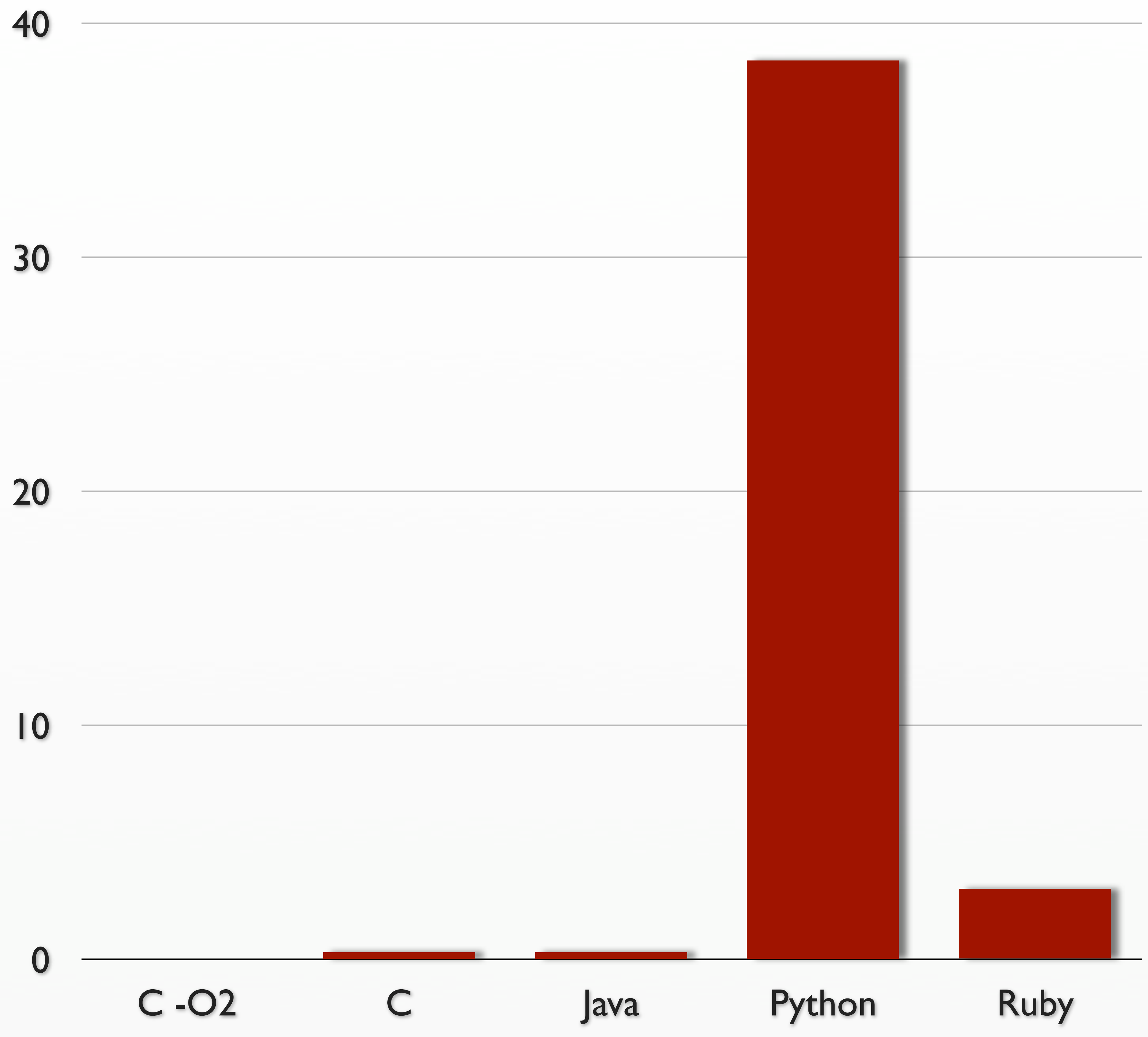
Flavors: JIT, AOT, hot spot

Native code generation

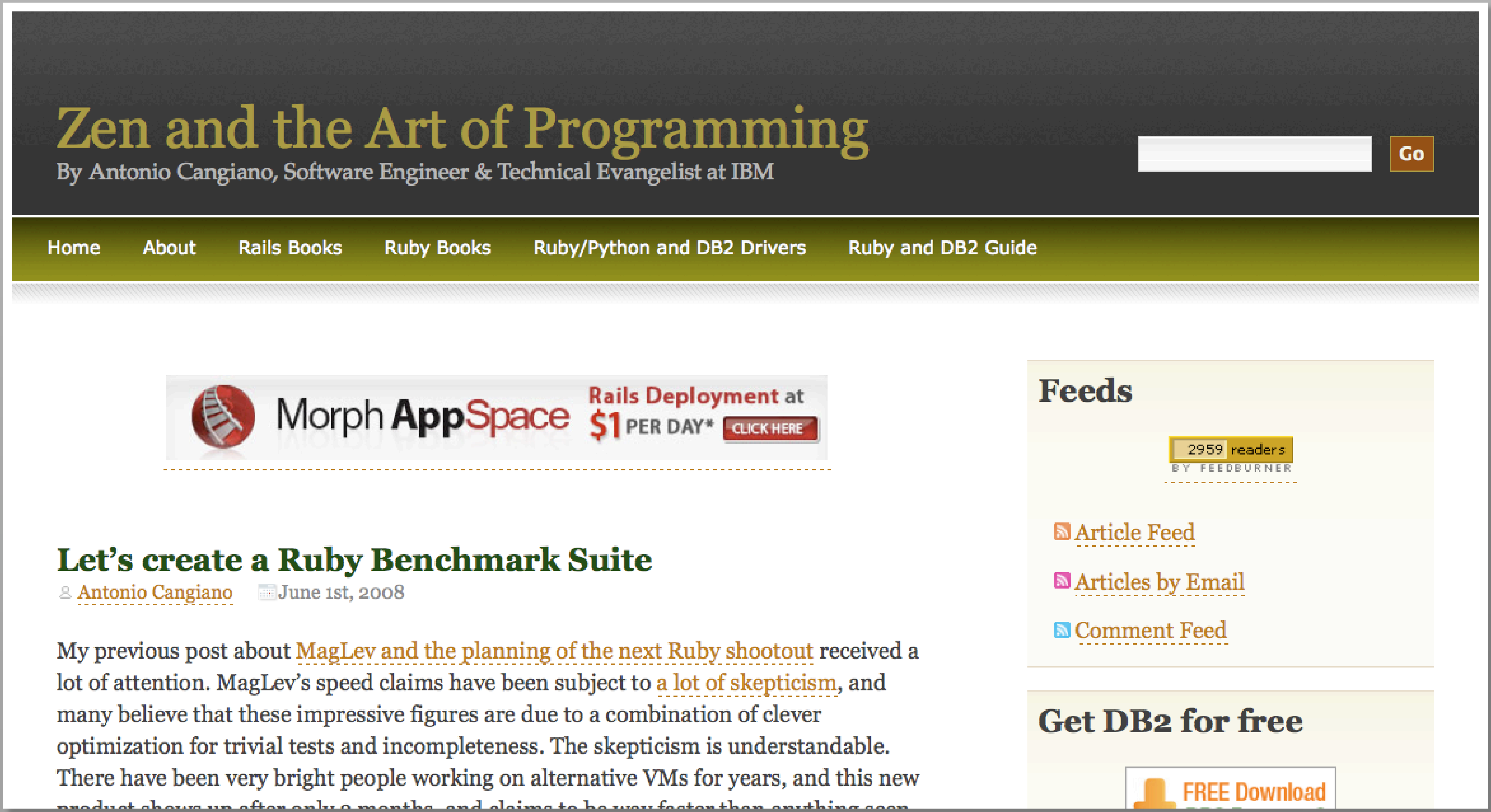
1. Convert all instructions to LLVM functions
2. Convert a CompiledMethod into a call sequence
3. Send through LLVM optimizer
4. Call the native code function

Where to now?


150.0
112.5
75.0
37.5
0
RBX C++
RBX
MRI 1.9
JRuby
MRI 1.8.6
MRI Apple


40
30
20
10
0
C -O2
C
Java
Python
Ruby

RubySpec



Zen and the Art of Programming

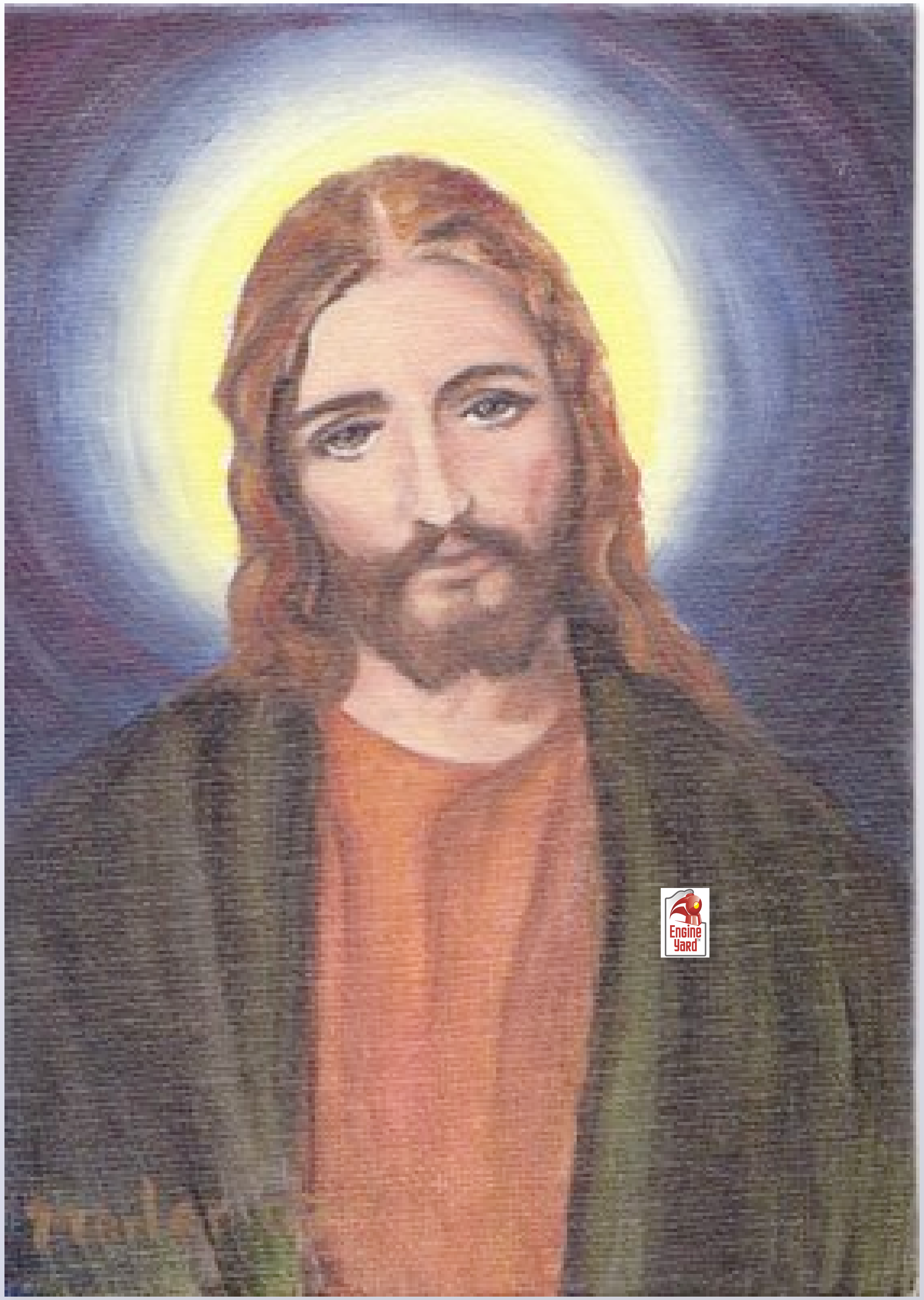
By Antonio Cangiano, Software Engineer & Technical Evangelist at IBM

Let's create a Ruby Benchmark Suite

Antonio Cangiano June 1st, 2008

My previous post about MagLev and the planning of the next Ruby shootout received a lot of attention. MagLev's speed claims have been subject to a lot of skepticism, and many believe that these impressive figures are due to a combination of clever optimization for trivial tests and incompleteness. The skepticism is understandable. There have been very bright people working on alternative VMs for years, and this new

http://github.com/acangiano/ruby-benchmark-suite/tree/master



Uncle Evan Wants You

to do one thing for Ruby today

http://rubini.us

http://git.rubini.us

Getting Rubinius

```
$ git clone git://git.rubini.us/code rubinius
$ cd rubinius
$ rake
```

Rubinius C++ VM

```
$ git checkout -b cpp origin/cpp
$ cd vm
$ rake
$ rake vm
```

No contribution is too small

thank you
merci
gracias
grazie
danke
ありがとう