JRuby 9000

Optimizing Above the JVM

Me

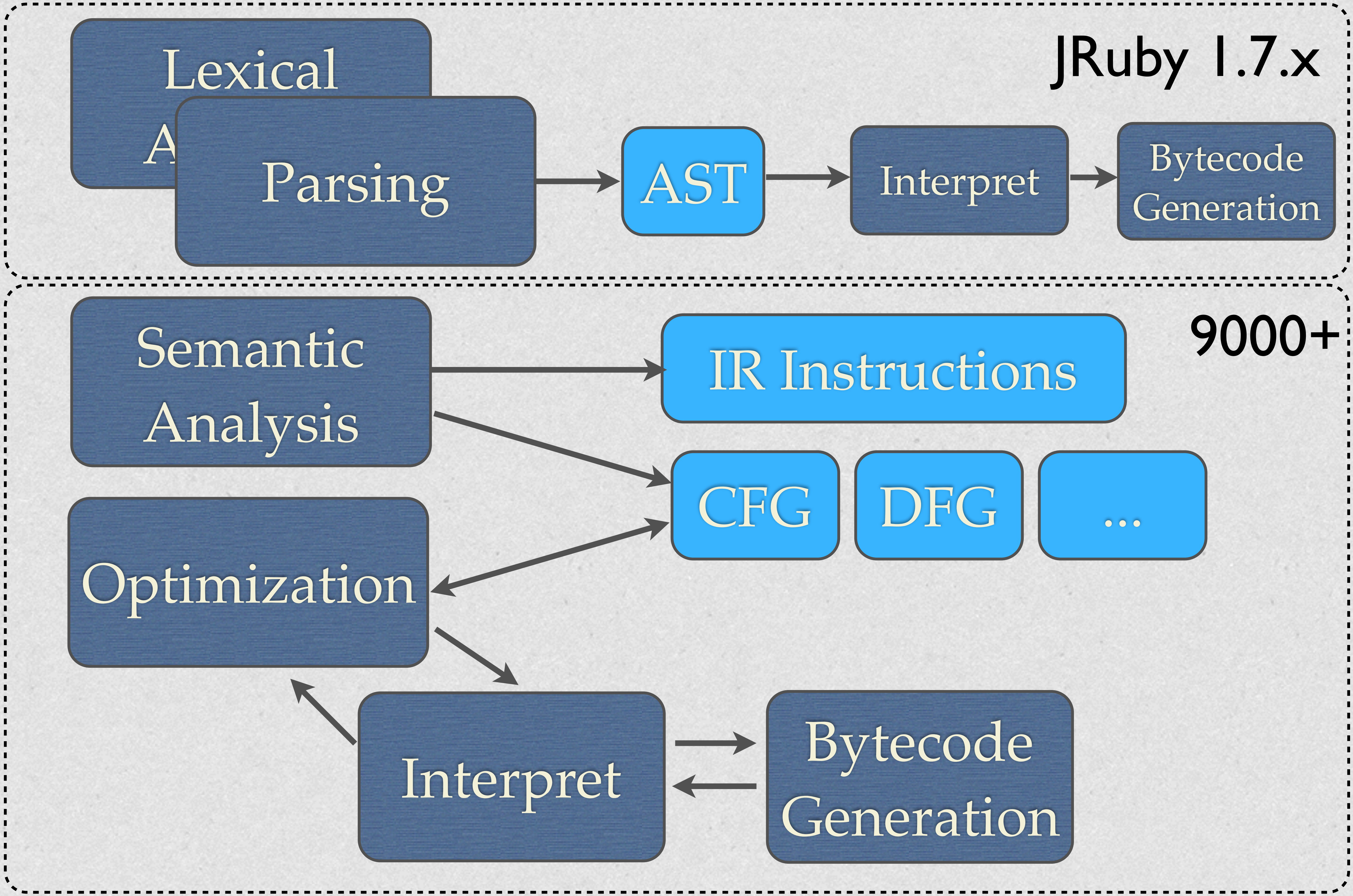
- Charles Oliver Nutter
- @headius
- Red Hat
- Based in Minneapolis, Minnesota
- Ten years working on JRuby (uff da!)

JRuby 9000

- Optimizable intermediate representation
- Mixed mode runtime (now with tiers!)
- Lazy JIT to JVM bytecode
- byte[] strings and regular expressions

Intermediate Representation

- AST to semantic representation
- Traditional compiler design
- Register machine



JRuby 1.7.x

Lexical A

Parsing

AST

Interpret

Bytecode Generation

9000+

Semantic Analysis

IR Instructions

CFG

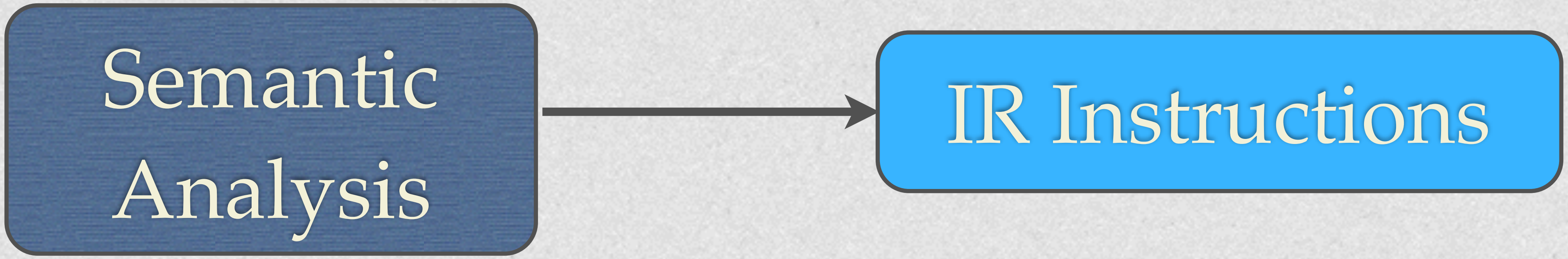
DFG

...

Optimization

Interpret

Bytecode Generation



Semantic Analysis

IR Instructions

Register-based

```
def foo(a, b)
 c = 1
 d = a + c
end
```

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
2  b = recv_pre_reqd_arg(1)
3  %block = recv_closure
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

3 address format

Optimization

```
-Xir.passes=LocalOptimizationPass,
            DeadCodeElimination
```

```ruby
def foo(a, b)
 c = 1
 d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
2  b = recv_pre_reqd_arg(1)
3  %block = recv_closure
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

-Xir.passes=LocalOptimizationPass,
DeadCodeElimination

```ruby
def foo(a, b)
  c = 1
  d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
2  b = recv_pre_reqd_arg(1)
3  %block = recv_closure
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

-Xir.passes=LocalOptimizationPass,
DeadCodeElimination

```ruby
def foo(a, b)
  c = 1
  d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
2  b = recv_pre_reqd_arg(1)
3  %block = recv_closure
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

-Xir.passes=LocalOptimizationPass,
DeadCodeElimination

```ruby
def foo(a, b)
  c = 1
  d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

```
-Xir.passes=LocalOptimizationPass,
                DeadCodeElimination
```

```ruby
def foo(a, b)
 c = 1
 d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

```
-Xir.passes=LocalOptimizationPass,
            DeadCodeElimination
```

```ruby
def foo(a, b)
  c = 1
  d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
5  line_num(1)
6  c = 1
7  line_num(2)
8  %v_0 = call(:+, a, [c])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

```
-Xir.passes=LocalOptimizationPass,
            DeadCodeElimination
```

```ruby
def foo(a, b)
  c = 1
  d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
5  line_num(1)
6  c =
7  line_num(2)
8  %v_0 = call(:+, a, [1])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

-Xir.passes=LocalOptimizationPass,
DeadCodeElimination

```
def foo(a, b)
 c = 1
 d = a + c
end
```

→

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
5  line_num(1)
7  line_num(2)
8  %v_0 = call(:+, a, [1])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

-Xir.passes=LocalOptimizationPass,
DeadCodeElimination

```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
5  line_num(1)
7  line_num(2)
8  %v_0 = call(:+, a, [1])
9  d = copy(%v_0)
10 return(%v_0)
```

Optimization

```
-Xir.passes=LocalOptimizationPass,
                    DeadCodeElimination
```

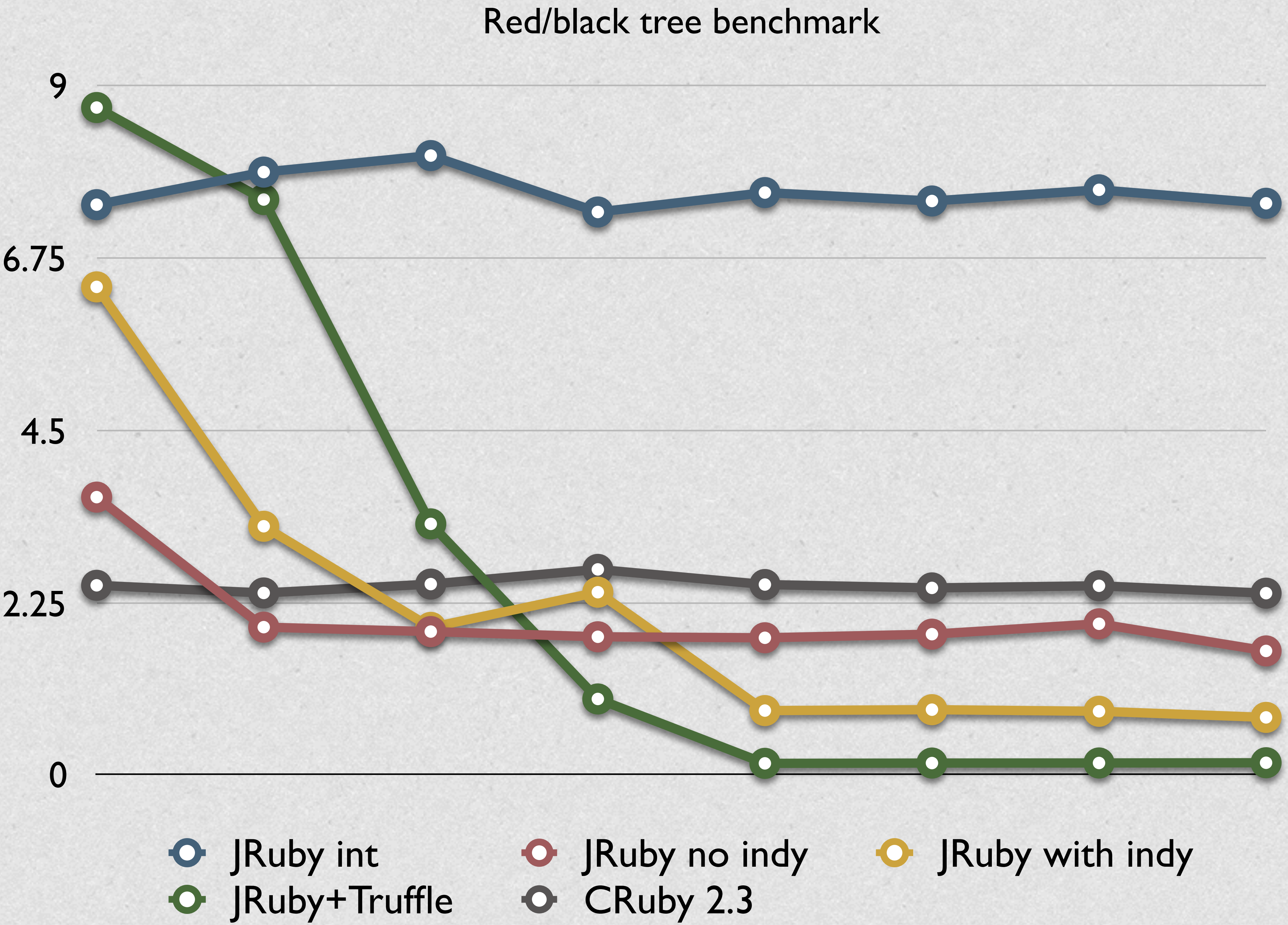
```
0  check_arity(2, 0, -1)
1  a = recv_pre_reqd_arg(0)
4  thread_poll
7  line_num(2)
8  %v_0 = call(:+, a, [1])
9  d = copy(%v_0)
10 return(%v_0)
```

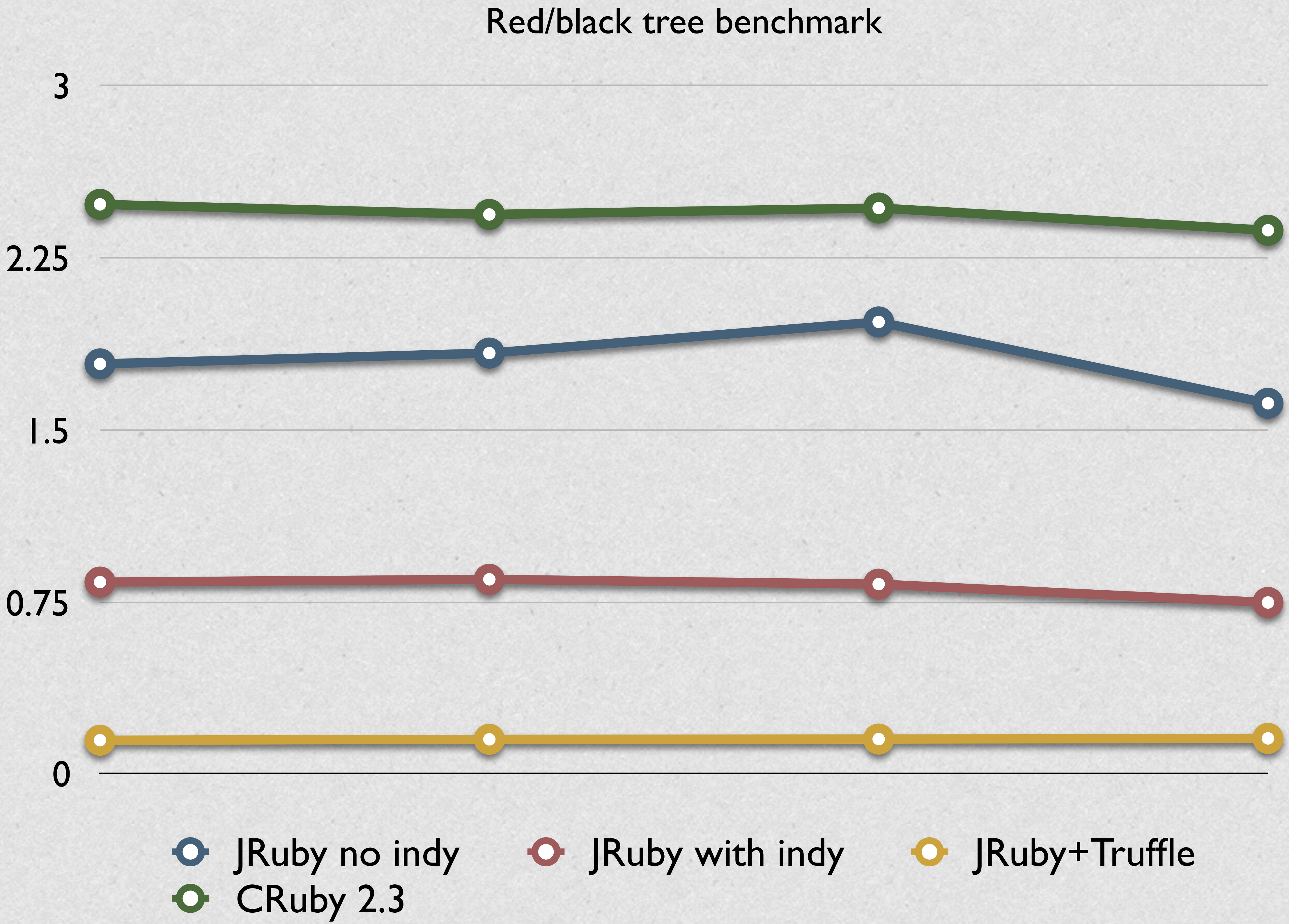
Tiers!

- Tier 1: Simple interpreter (no passes run)
- Tier 2: Full interpreter (static optimization)
- Tier 3: Full interpreter (profiled optz)
- Tier 4: JVM bytecode
- Tiers 5+: Whatever JVM does from there

Why Not Truffle?

- Startup and memory use are worse
- No integration with other JVM langs yet
- We still want to target JVM
- It's not ready yet!


Red/black tree benchmark
9
6.75
4.5
2.25
0
JRuby int
JRuby no indy
JRuby with indy
JRuby+Truffle
CRuby 2.3


Red/black tree benchmark
3
2.25
1.5
0.75
0
JRuby no indy
JRuby with indy
JRuby+Truffle
CRuby 2.3

Recent Wins

- JITable blocks
- define_method performance
- Reduced-cost transient exceptions

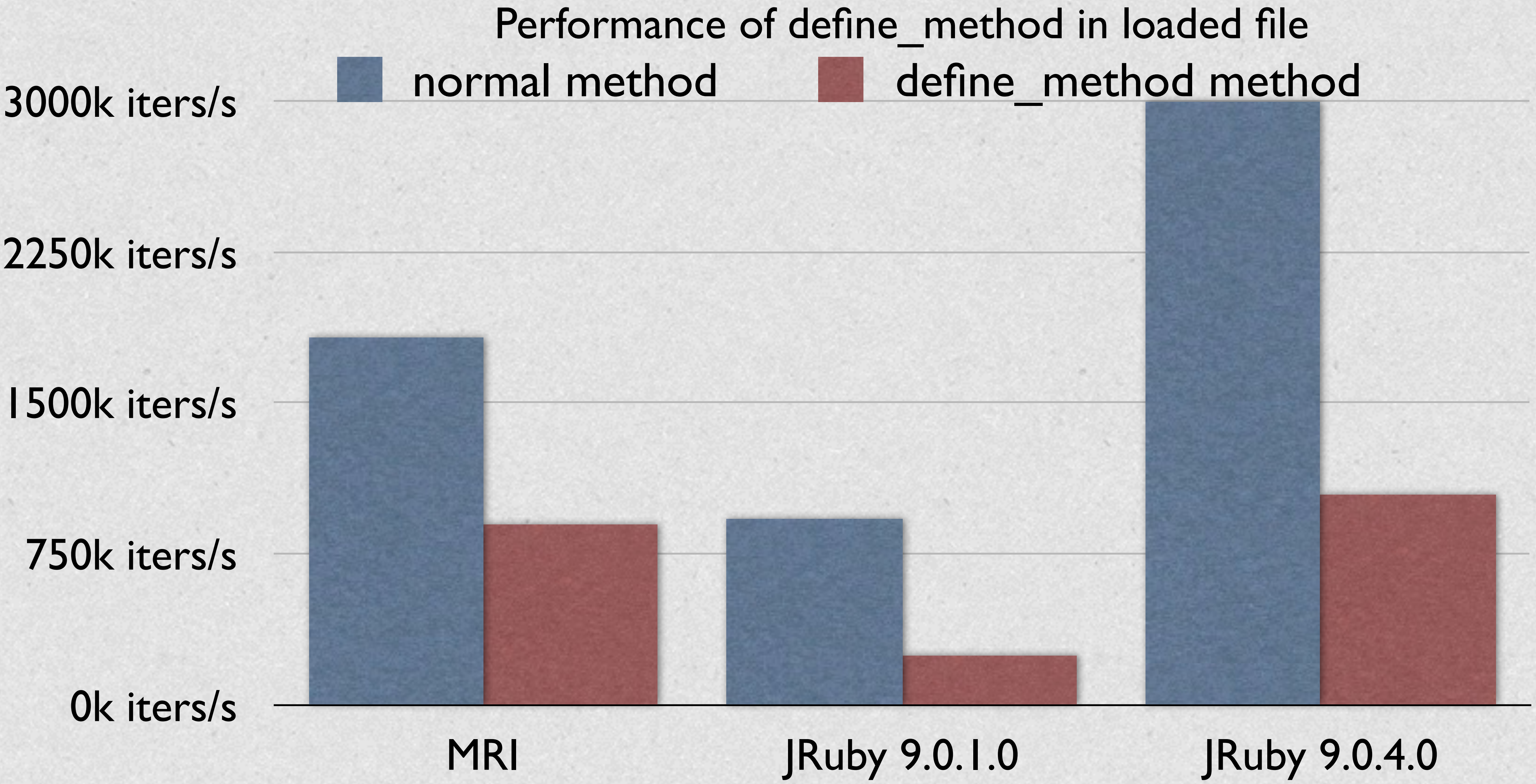
Block Jitting

- JRuby 1.7 only jitted methods
  - Not free-standing procs/lambdas
  - Not define_method blocks
- Easier to do now with 9000's IR
- Blocks JIT as of 9.0.4.0

```
class Foo
  define_method :test do
    self
  end
end
```

```
loop do
  puts Benchmark.measure { 1_000_000.times { call some method }}
end
```

Block Jitting



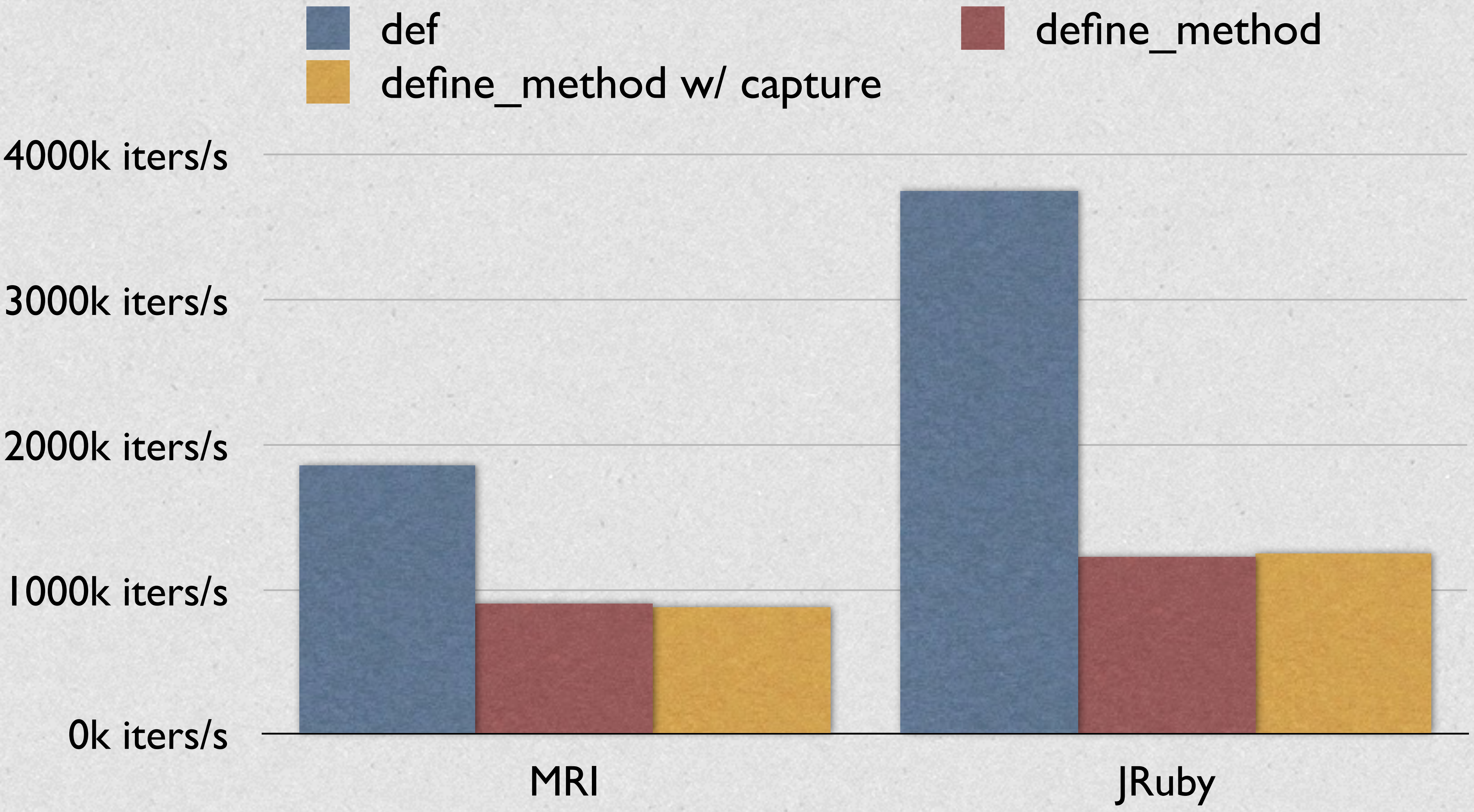
```
ruby -e 'load "bench_define_method.rb"'
```

define_method

```ruby
define_method(:add) do |a, b|
  a + b
end

names.each do |name|
  define_method(name) { send :"do_#{name}" }
end
```

Convenient for metaprogramming,
but blocks have more overhead than methods.

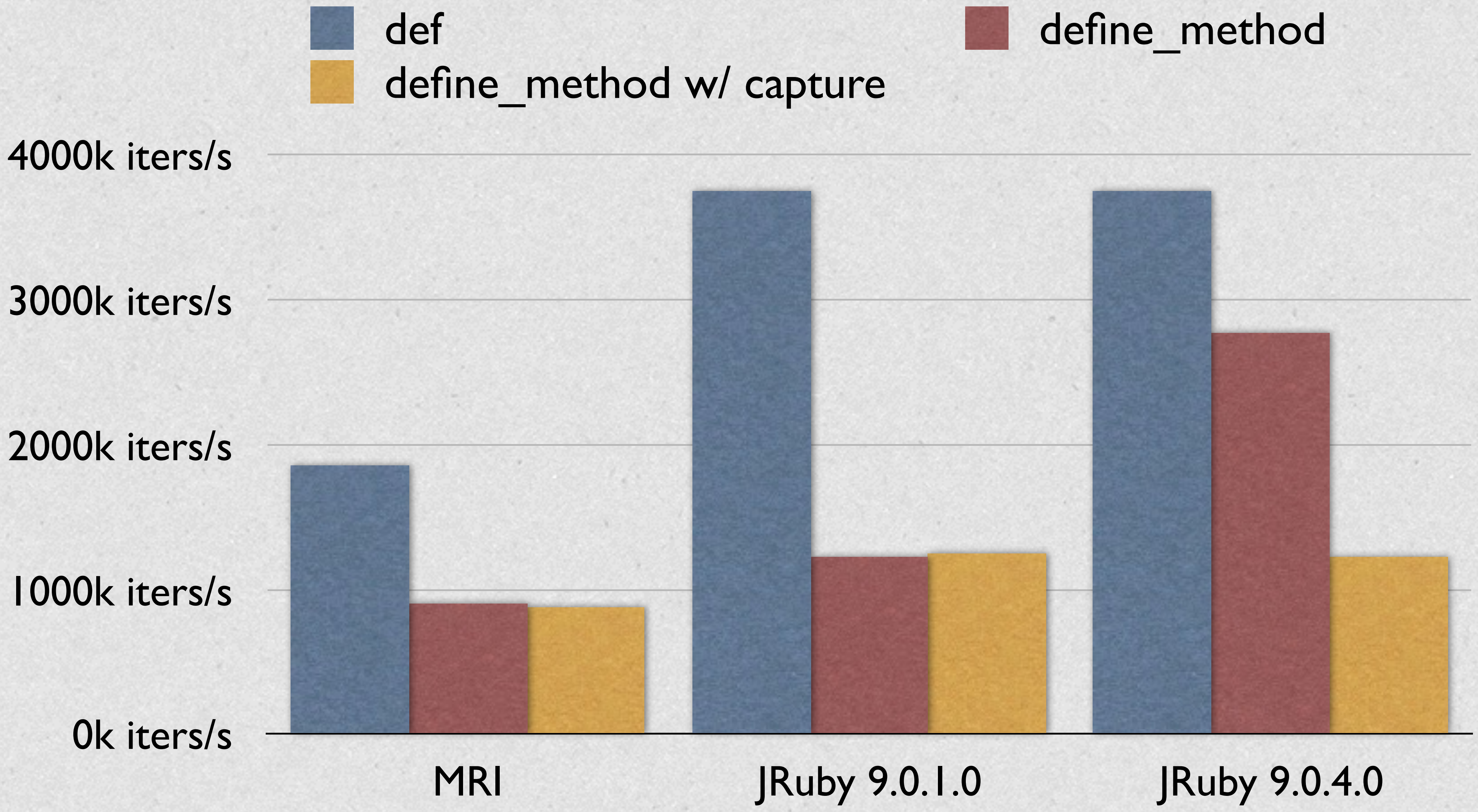
:-(


def
define_method
define_method w/ capture
4000k iters/s
3000k iters/s
2000k iters/s
1000k iters/s
0k iters/s
MRI
JRuby

Optimizing define_method

- Noncapturing
  - Treat as method in compiler
  - Ignore surrounding scope
- Capturing (future work)
  - Lift read-only variables as constant

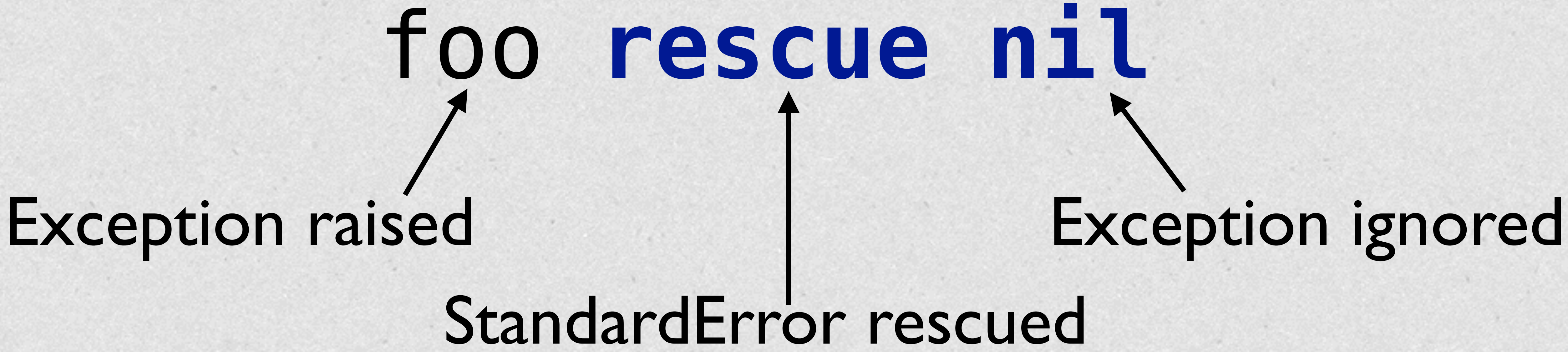
Getting Better!



Low-cost Exceptions

- Backtrace cost is VERY high on JVM
  - Lots of work to construct
- Exceptions frequently ignored
  - ...or used as flow control (shame!)
- If ignored, backtrace is not needed!

Postfix Antipattern



```ruby
foo rescue nil
```

Exception raised

StandardError rescued

Exception ignored

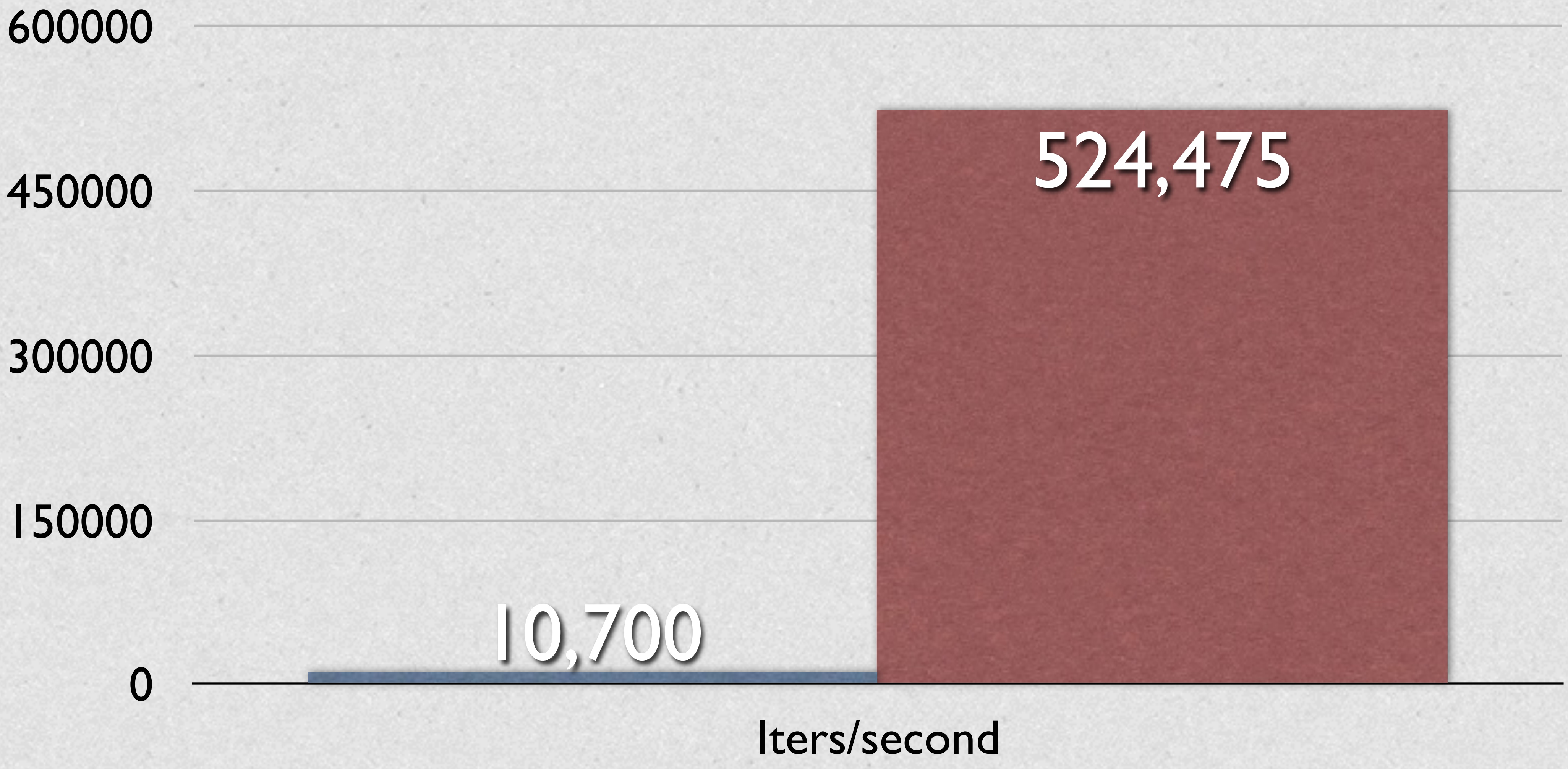
Result is simple expression, so exception is never visible.

csv.rb Converters

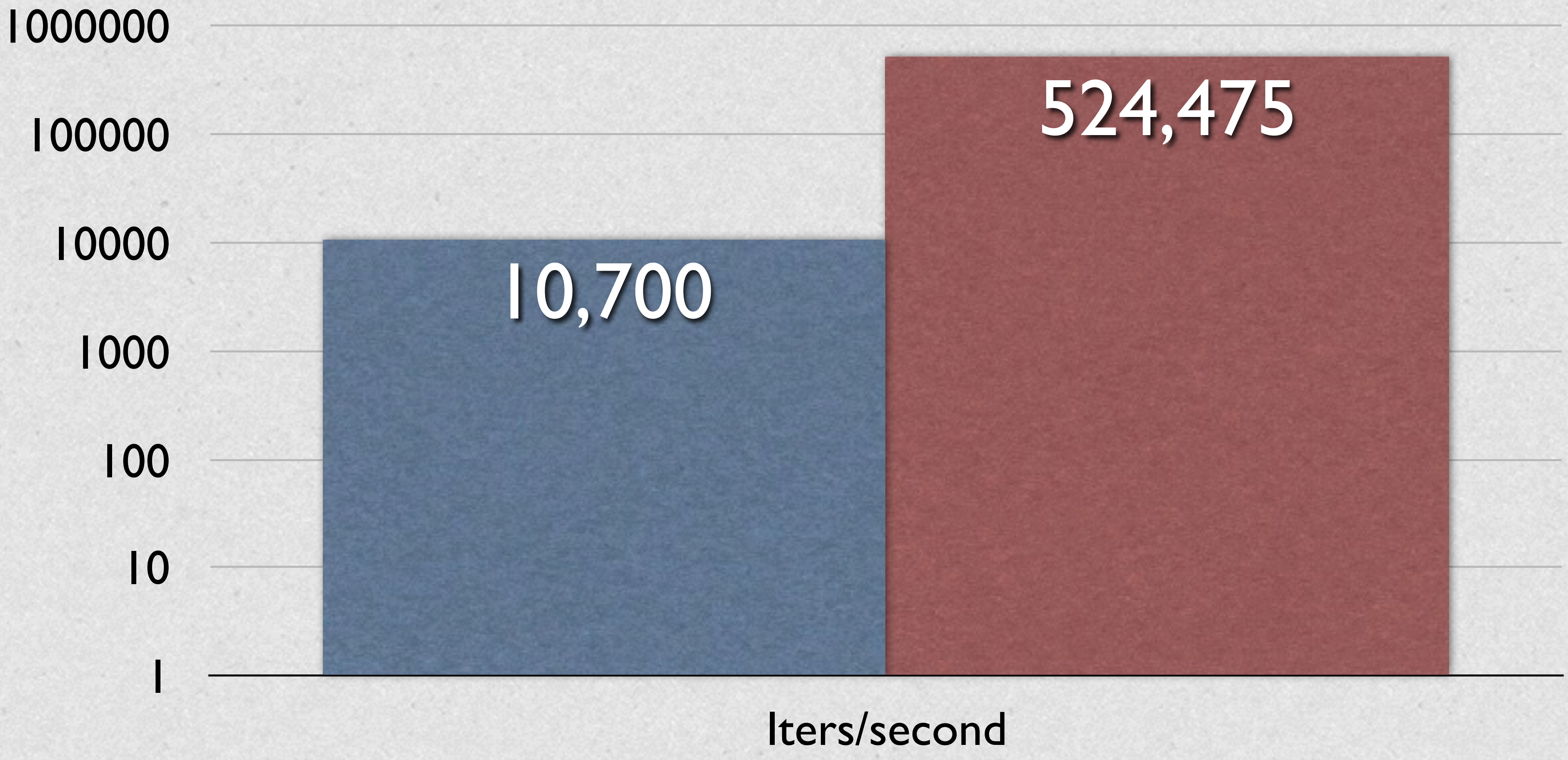
```
Converters  = { integer:    lambda { |f|
                    Integer(f) rescue f
                },
                float:      lambda { |f|
                   Float(f) rescue f
                },
                ...
```

All trivial rescues, no traces needed.

Simple rescue Improvement



Much Better!


1000000
100000
10000
1000
100
10
1
10,700
524,475
Iters/second

Current Work

Inlining

- 900 pound gorilla of optimization
  - shove method/closure back to callsite
  - eliminate call frames
  - eliminate parameter passing/return

But... JVM?

- JVM will inline for us, but...
  - only if we use invokedynamic
  - and the code isn't too big
  - and there's no polymorphic code
  - and it feels like it today

Today’s Inliner

```
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i > 0
  i = decrement_one(i)
end
```

→

```
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i < 0
  if guard_same? self
    i = i - 1
  else
    i = decrement_one(i)
  end
end
```

Today’s Inliner

```
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i > 0
  i = decrement_one(i)
end
```

→

```
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i < 0
  if guard_same? self
    i = i - 1
  else
    i = decrement_one(i)
  end
end
```

Today’s Inliner

```
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i > 0
  i = decrement_one(i)
end
```

```
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i < 0
  if guard same? self
    i = i - 1
  else
    i = decrement_one(i)
  end
end
```

Today’s Inliner

```ruby
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i > 0
  i = decrement_one(i)
end
```

→

```ruby
def decrement_one(i)
  i - 1
end

i = 1_000_000
while i < 0
  if guard_same? self
    i = i - 1
  else
    i = decrement_one(i)
  end
end
```

Inlining Today

- Conceptually simple
- Does not deoptimize
- Works in interpreter and JIT
- Still experimental
- Stack trace reconstitution TBD

Profiling

- You can't inline if you can't profile!
- For each call site record call info
  - Which method(s) called
  - How frequently
- Inline most frequently-called method

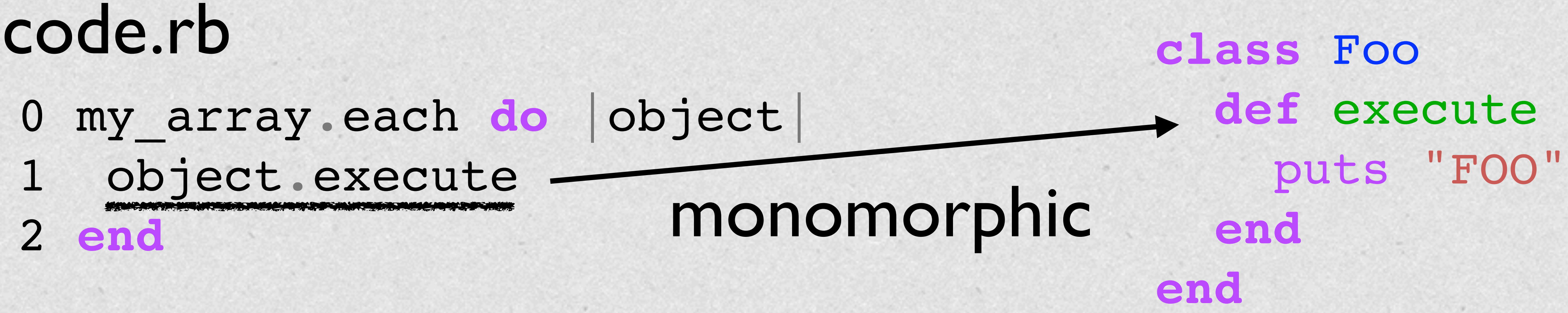
Profiling

code.rb

callsite →

```ruby
my_array.each do |object|
  object.execute
end
```

Profiling



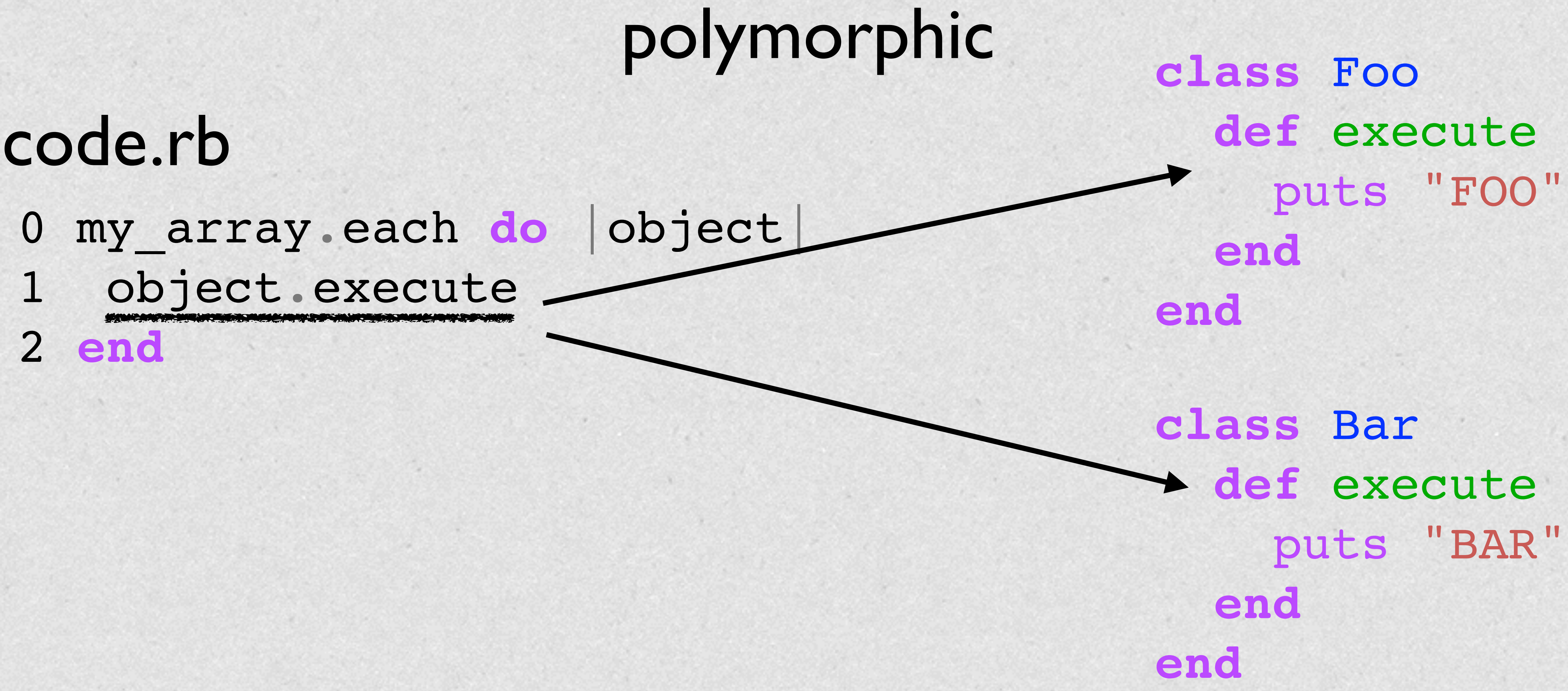
code.rb

```ruby
my_array.each do |object|
  object.execute
end
```

monomorphic

```ruby
class Foo
  def execute
    puts "FOO"
  end
end
```

Profiling



polymorphic

code.rb

```ruby
my_array.each do |object|
  object.execute
end
```

```ruby
class Foo
  def execute
    puts "FOO"
  end
end

class Bar
  def execute
    puts "BAR"
  end
end
```

Profiling

- <2% overhead (to be reduced more)
- Working* (interpreter AND JIT)
- Feeds directly into inlining

* Fragile and buggy!

Profiling

```
def small_loop(i)
  k = 10
  while k > 0
    k = yield(k)
  end
  i - 1
end

def big_loop(i)
  i = 100_000
  while true
    i = small_loop(i) { |j| j - 1 }
    return 0 if i < 0
  end
end

900.times { |i| big_loop i }
```

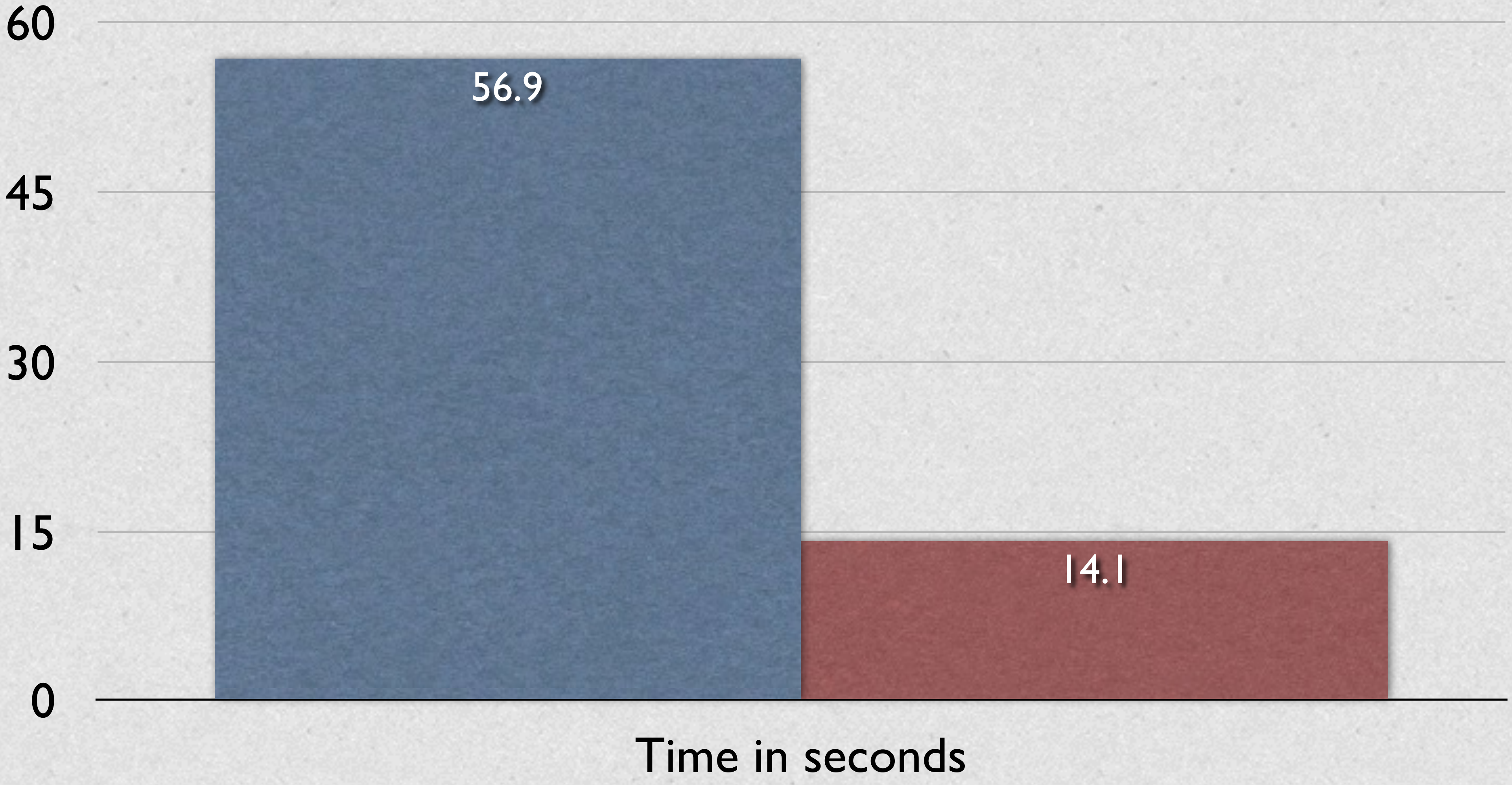
Like an Array#each

May see many blocks

JVM will not inline this

hot & monomorphic

Inlining FTW!



Numeric Specialization

- Everything's an object
- JVM has only references and primitives
    - Not compatible in bytecode
- Need to optimize numerics as primitive

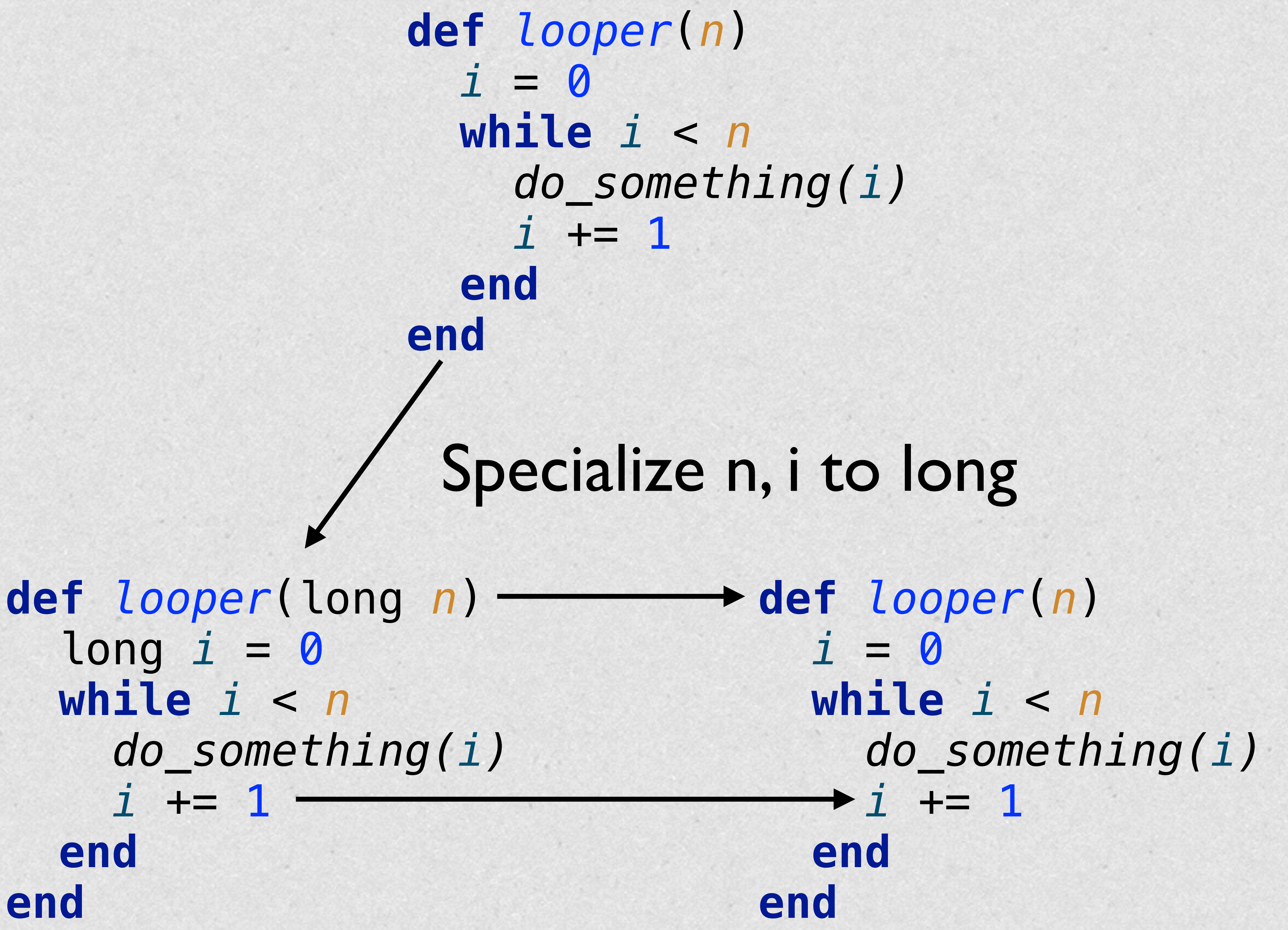
```
def looper(n)
  i = 0
  while i < n
    do_something(i)
    i += 1
  end
end
```

Probably a Fixnum?

Cached object

Call with i

New Fixnum i + 1



```
def looper(n)
  i = 0
  while i < n
    do_something(i)
    i += 1
  end
end
```

Specialize n, i to long

```
def looper(long n)
  long i = 0
  while i < n
    do_something(i)
    i += 1
  end
end
```

```
def looper(n)
  i = 0
  while i < n
    do_something(i)
    i += 1
  end
end
```

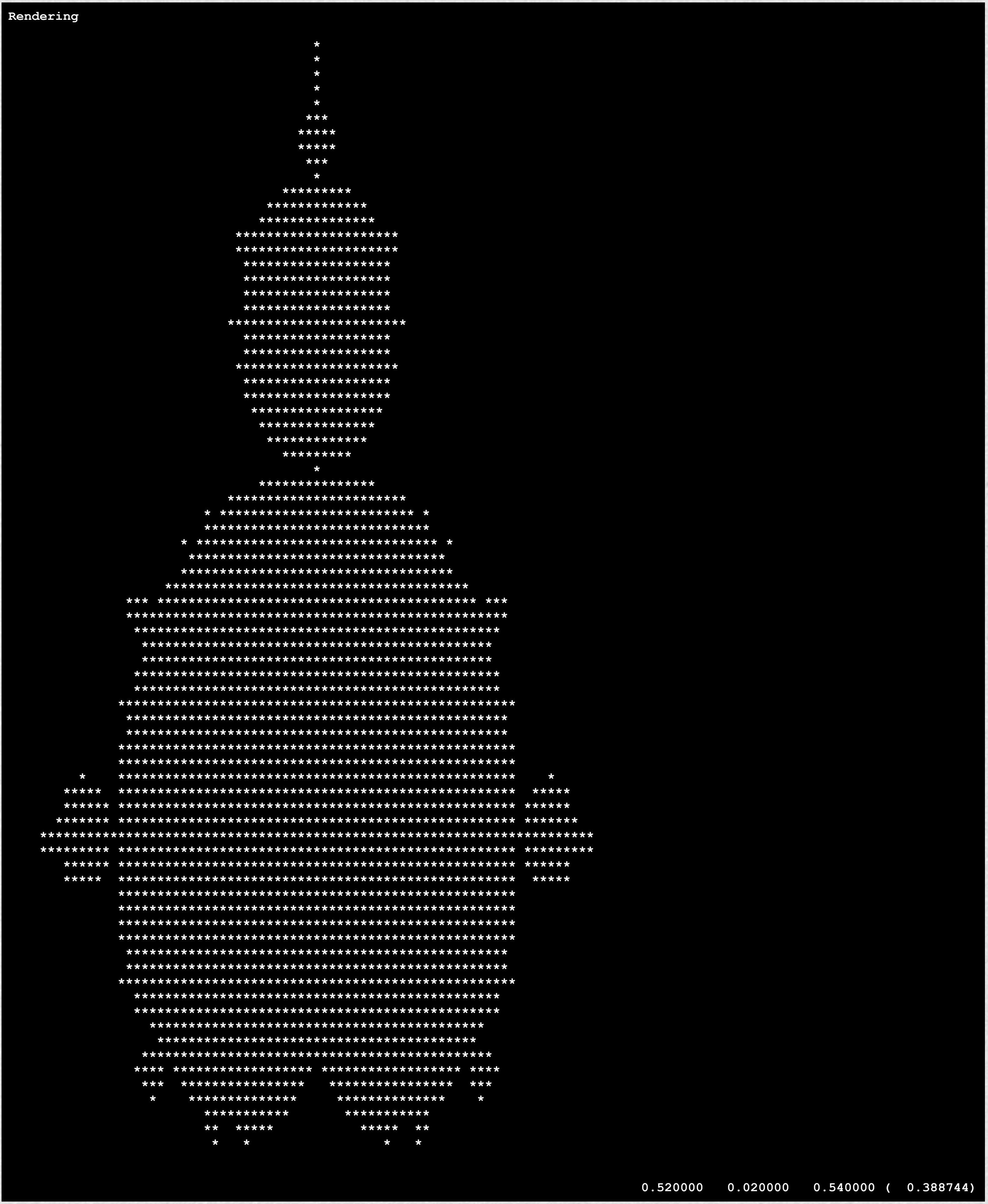
Deopt to object version if n or i + 1 is not Fixnum

Interpreter FTW!

- Deopt is much simpler with interpreter
  - Collect local vars, instruction index
  - Raise exception to bail out
  - Fire up interpreter, keep going
- Much cheaper than resuming bytecode

Current Status

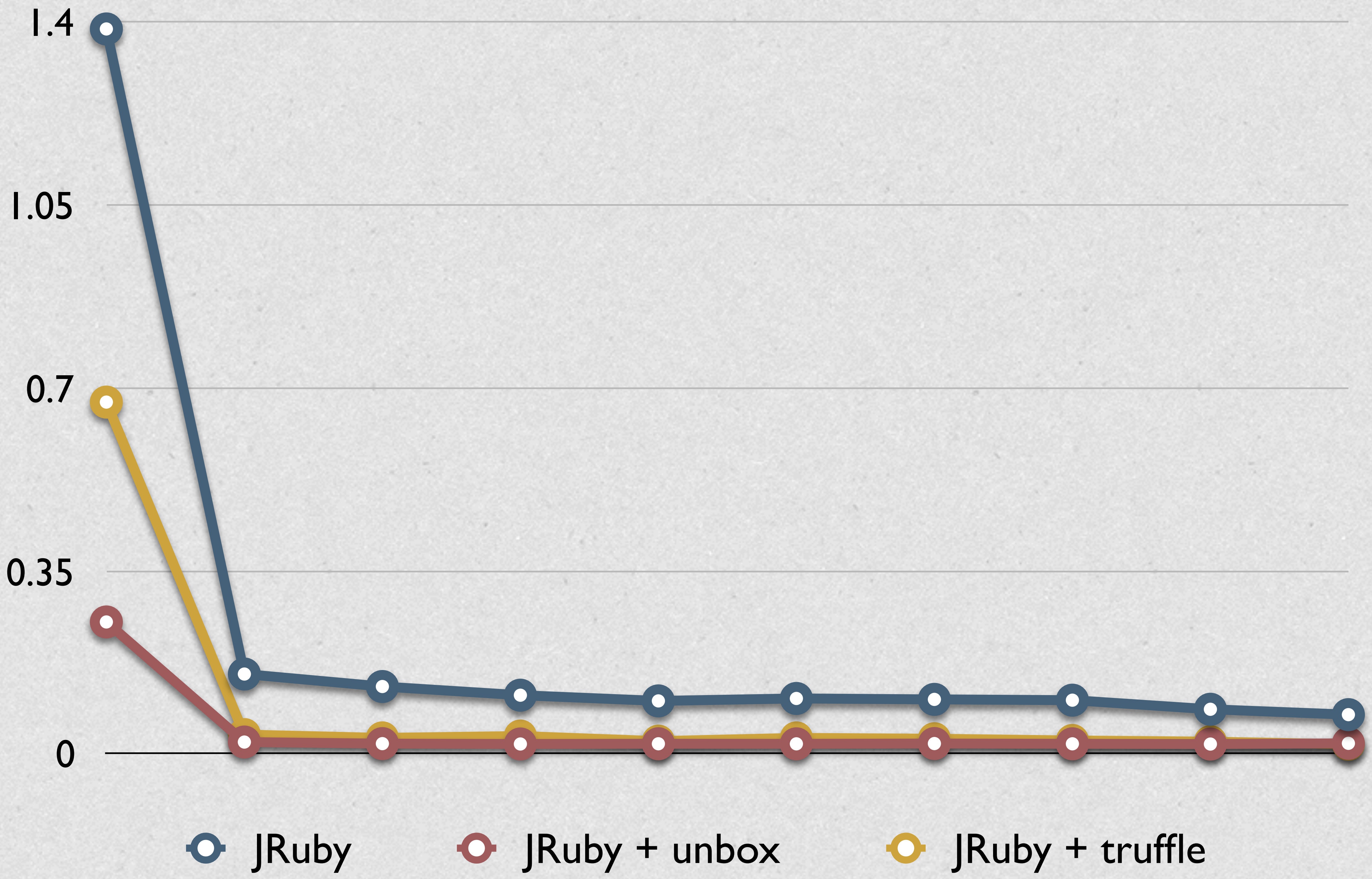
- Working prototype
- No deopt
- No type guards
- No overflow check for Fixnum/Bignum

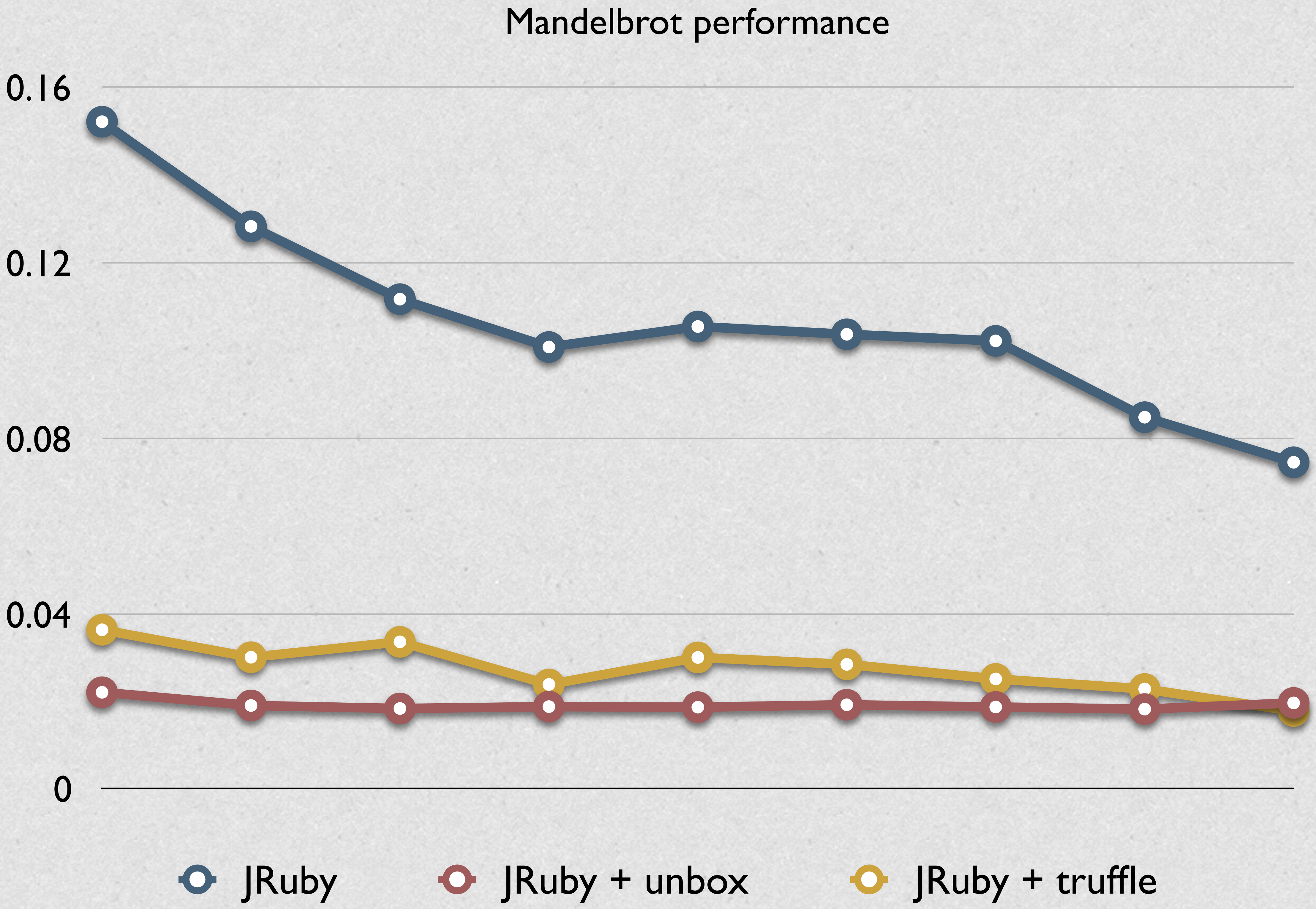


```
Rendering
                                       *
                                       *
                                       *
                                       *
                                       *
                                      ***
                                     *****
                                     *****
                                      ***
                                       *
                                   *********
                                 *************
                                ***************
                             *********************
                             *********************
                              *******************
                              *******************
                              *******************
                              *******************
                            ***********************
                              *******************
                              *******************
                             *********************
                              *******************
                              *******************
                               *****************
                                ***************
                                 *************
                                   *********
                                       *
                                ***************
                            ***********************
                         * ************************* *
                         *****************************
                      * ******************************* *
                       *********************************
                      ***********************************
                    ***************************************
               *** ***************************************** ***
               *************************************************
                ***********************************************
                 *********************************************
                 *********************************************
                ***********************************************
                ***********************************************
              ***************************************************
               *************************************************
               *************************************************
              ***************************************************
              ***************************************************
         *    ***************************************************    *
       *****  ***************************************************  *****
       ****** *************************************************** ******
      ******* *************************************************** *******
    ***********************************************************************
    ********* *************************************************** *********
       ****** *************************************************** ******
       *****  ***************************************************  *****
              ***************************************************
              ***************************************************
              ***************************************************
              ***************************************************
               *************************************************
               *************************************************
              ***************************************************
                ***********************************************
                ***********************************************
                  *******************************************
                   *****************************************
                 *********************************************
                **** ****************** ****************** ****
                 ***  ****************   ****************  ***
                  *    **************     **************    *
                         ***********       ***********
                         **  *****           *****  **
                          *   *                 *   *

                                                                                  0.520000   0.020000   0.540000 (  0.388744)
```

```
def iterate(x,y)
  cr = y-0.5
  ci = x
  zi = 0.0
  zr = 0.0
  i = 0
  bailout = 16.0
  max_iterations = 1000

  while true
    i += 1
    temp = zr * zi
    zr2 = zr * zr
    zi2 = zi * zi
    zr = zr2 - zi2 + cr
    zi = temp + temp + ci
    return i if (zi2 + zr2 > bailout)
    return 0 if (i > max_iterations)
  end
end
```

Mandelbrot performance




Mandelbrot performance
0.16
0.12
0.08
0.04
0
JRuby
JRuby + unbox
JRuby + truffle

When?

- Profiling, inlining mostly need testing
- Specialization needs guards, deopt
- Probably landing in next couple months

JRuby 9.1.0.0

- Coming soon (end of Feb?)
- Ruby 2.3 support
- Some of these optimizations
- More attention to performance overall

medstro
MX
eazyBI
NASA
IBM
loggly
OBJECTFAB
simple is beautiful.
gettyimages®
FAVEOD
intuitive software engineering
HoodQ
innoQ
asd
twitpic
KINETIC DATA
diverza.
Square
swiftype
SOUNDCLOUD
comcast
puppet labs®
Gemfury
ThoughtWorks®
Rabobank
On-Site.com
Disney
Social Games
VISA
BBC NEWS
SIMPLE
DIUS
EVRONE
inovex
S·D SPORTSDATA
DRIVING INNOVATION IN SPORTS
ATOMIC OBJECT
Mediapinе
mingle
Travis CI
ORACLE
Lookout
elasticsearch.
Telmate
Mobile System 7
UNITED SIGNALS
Certified Investment
HomeAway
FastPencil
livingsocial
Burt.
GROUPON
xnlogic
Constant Contact®
UNIVERSITY OF FERRARA
- EX LABORE FRUCTUS -
maestrano

Thank You

- @headius
- @tom_enebo
- http://jruby.org