Deoptimizing Ruby

JRuby+Truffle and the antidote to JITs

Chris Seaton
@ChrisGSeaton

Oracle Labs

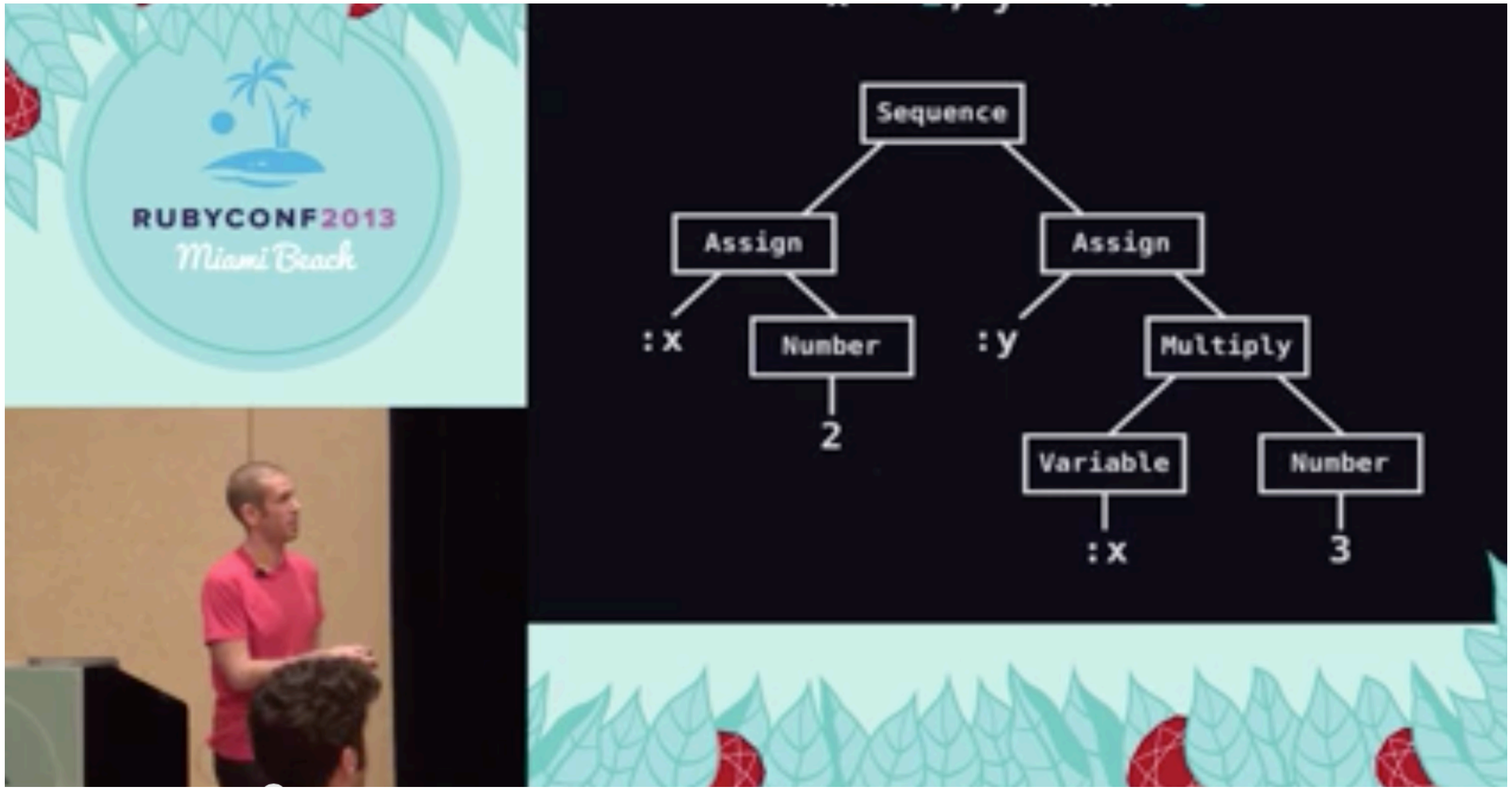
Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

chrisseaton.com/rubytruffle/deoptimizing

JRuby+Truffle

A new open source implementation of Ruby by **Oracle Labs** with a **JIT** using **next-gen JVM** technology and **partial evaluation**, now part of **JRuby**



codon.com/compilers-for-free

Why is Ruby hard to optimize?

Fixnum to Bignum promotion

Monkey patching methods

#binding

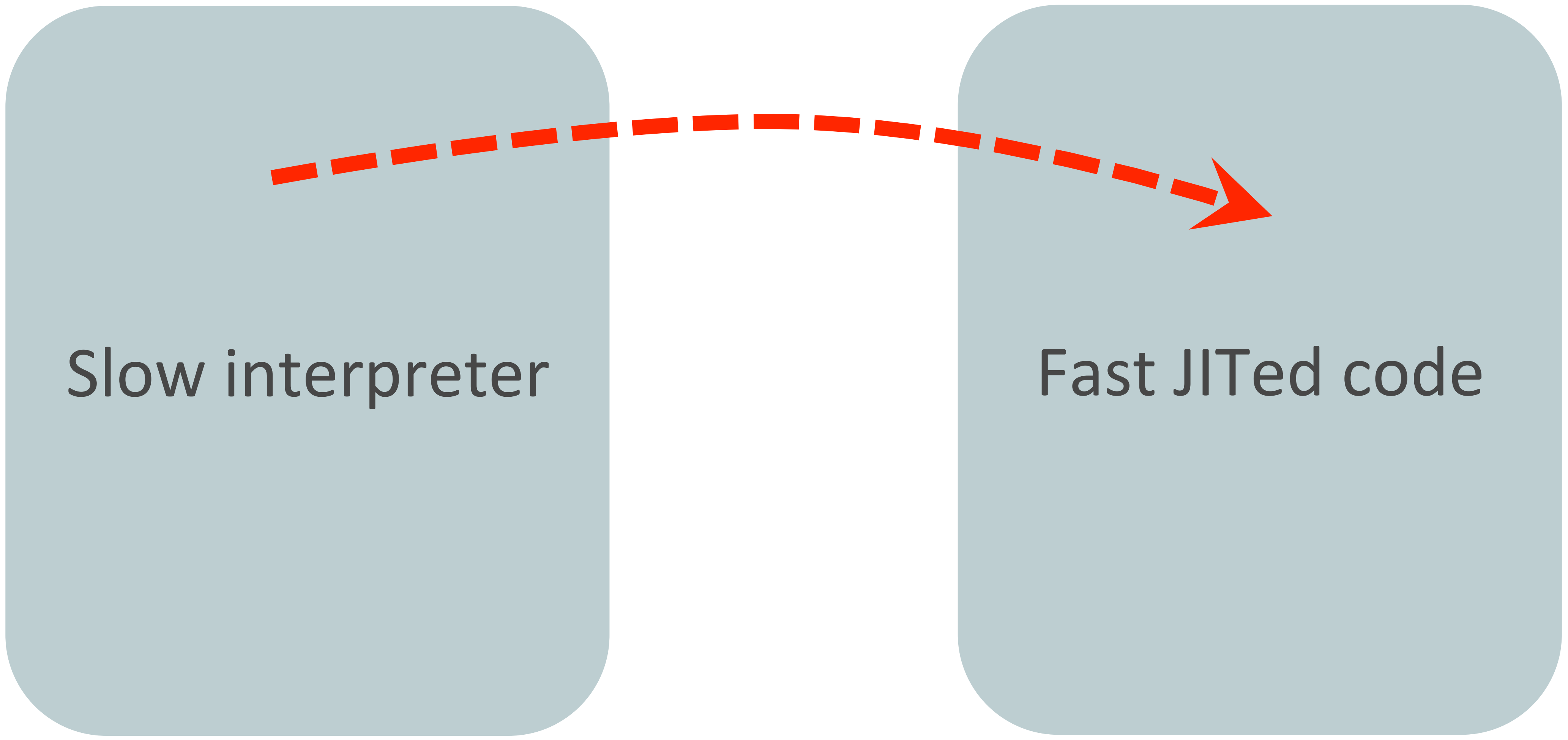
ObjectSpace

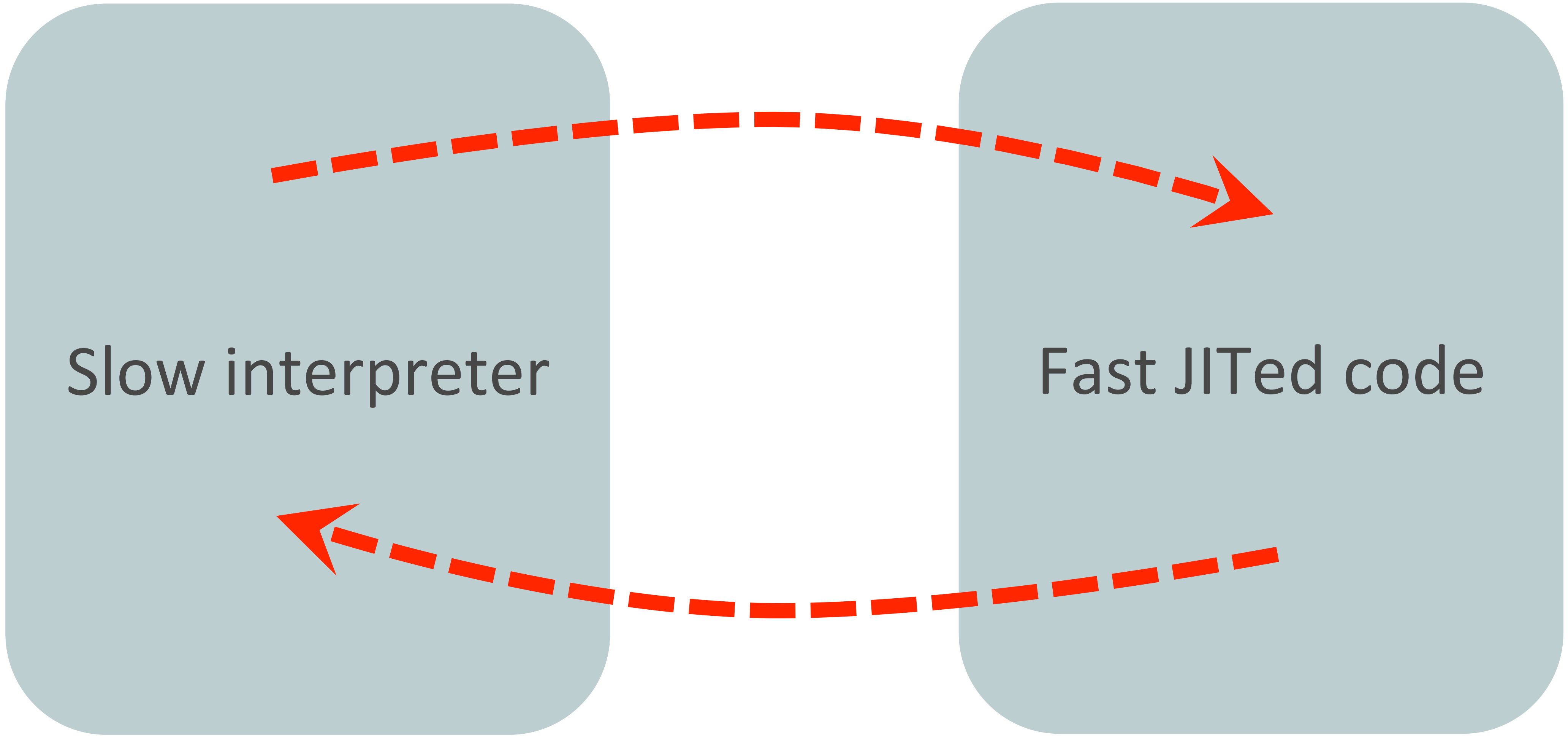
set_trace_func

Thread#raise

Deoptimization

elegantly solves all these problems


Slow interpreter
Fast JITed code


Slow interpreter
Fast JITed code

Illustrating Deoptimization

John Tenniel illustrations public domain in the UK and US

John Tenniel illustrations public domain in the UK and US

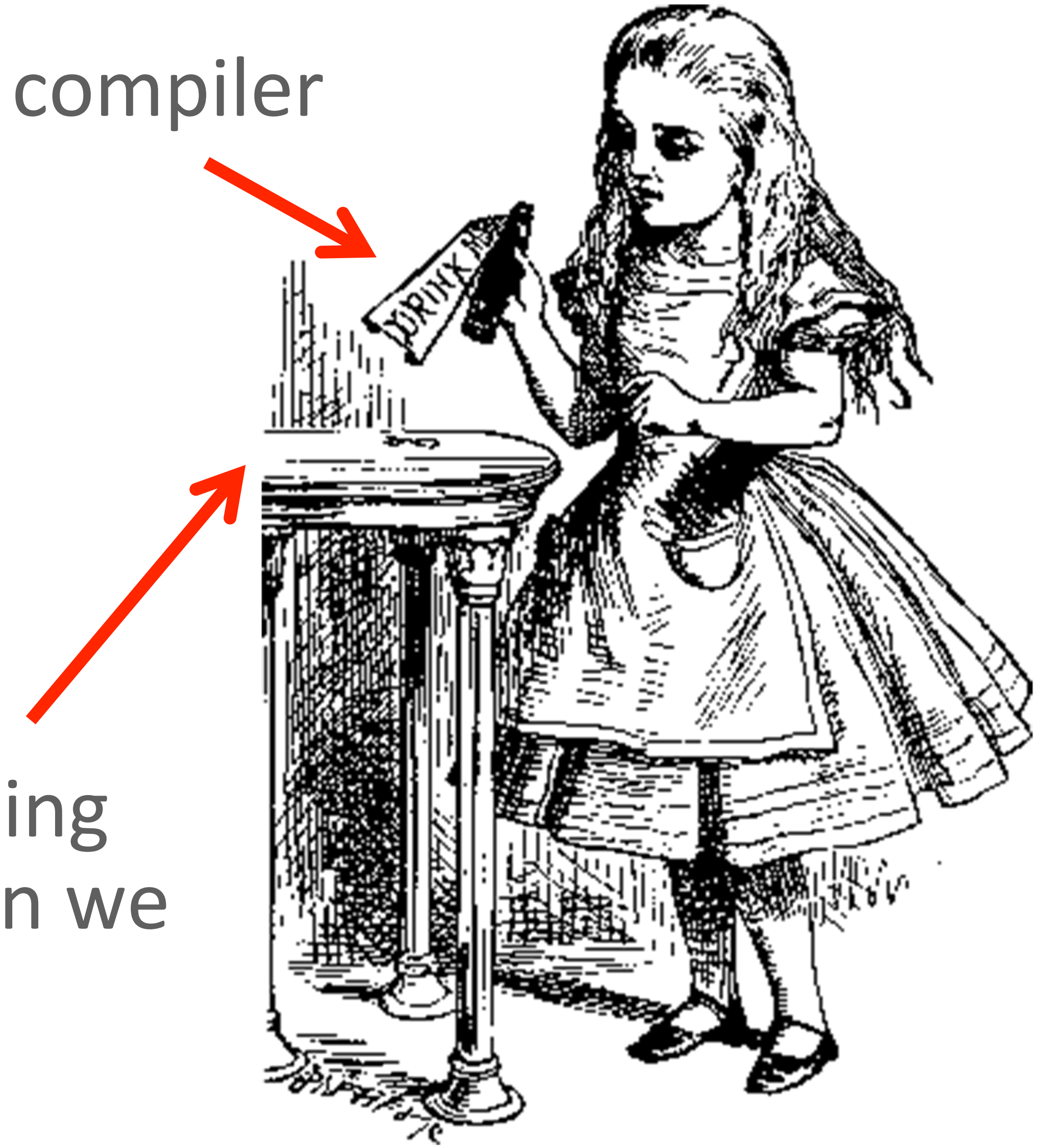
John Tenniel illustrations public domain in the UK and US

Ruby

Door to utopia of high performance

John Tenniel illustrations public domain in the UK and US

Just-in-time compiler



Left something behind when we compiled

John Tenniel illustrations public domain in the UK and US

Deoptimization reverses the effects of the JIT

John Tenniel illustrations public domain in the UK and US

What does deoptimization do for Ruby?

Fixnum to Bignum promotion

```
a + b + c
```

We’ll assume we already know these are Fixnums

```
t1 = Fixnum(a) + Fixnum(b)
if t1.overflowed?
  t1 = Bignum(a) + Bignum(b)
  t2 = Bignum(t1) + Bignum(c)
else
  t2 = Fixnum(t1) + Fixnum(c)
  if t2.overflowed?
    t2 = Bignum(t1) + Bignum(c)
  end
end
```

```
t1 = Fixnum(a) + Fixnum(b)
deoptimize! if t1.overflowed?
t2 = Fixnum(t1) + Fixnum(c)
deoptimize! if t2.overflowed?
```

```
t1 = Fixnum(a) + Fixnum(b)
if t1.overflowed?
  t1 = Bignum(a) + Bignum(b)
  t2 = Bignum(t1) + Bignum(c)
else
  t2 = Fixnum(t1) + Fixnum(c)
  deoptimize! if t2.overflowed?
end
```

Monkey patching methods

```
my_object.my_method(x, y)
```

```
lookup my_method in my_object
call it with (x, y)
```

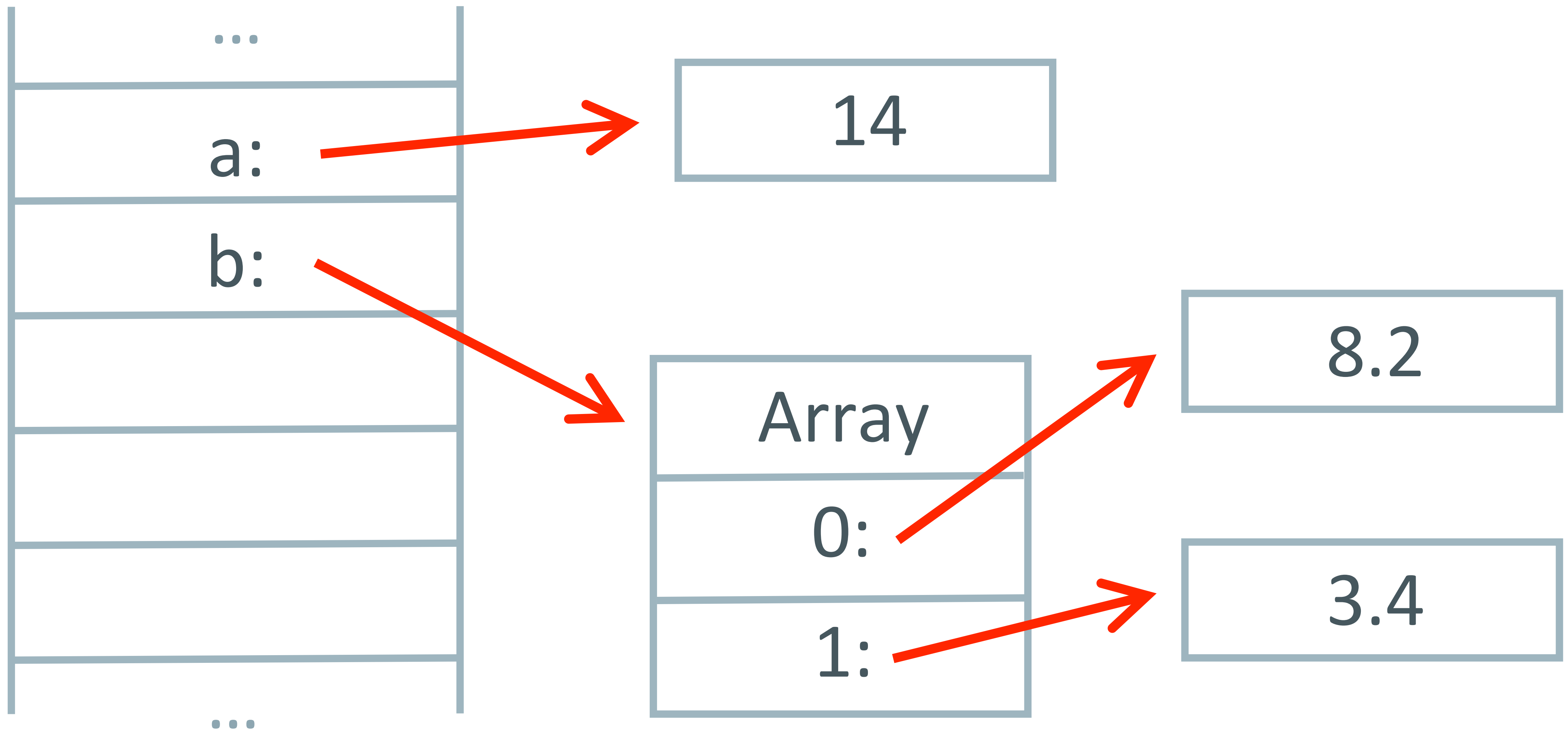
```
if my_object.changed?
  lookup my_method in my_object
  call it with (x, y)
else
  use cached my_method
  call it with (x, y)
end
```

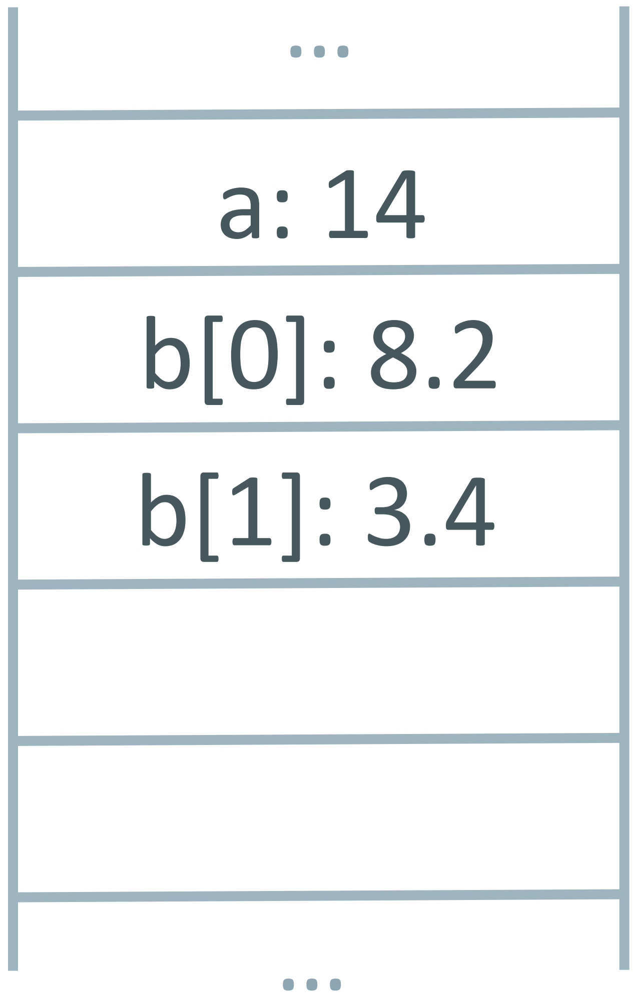
```
if my_object.changed?
  deoptimize!
else
  use cached my_method
  call it with (x, y)
end
```

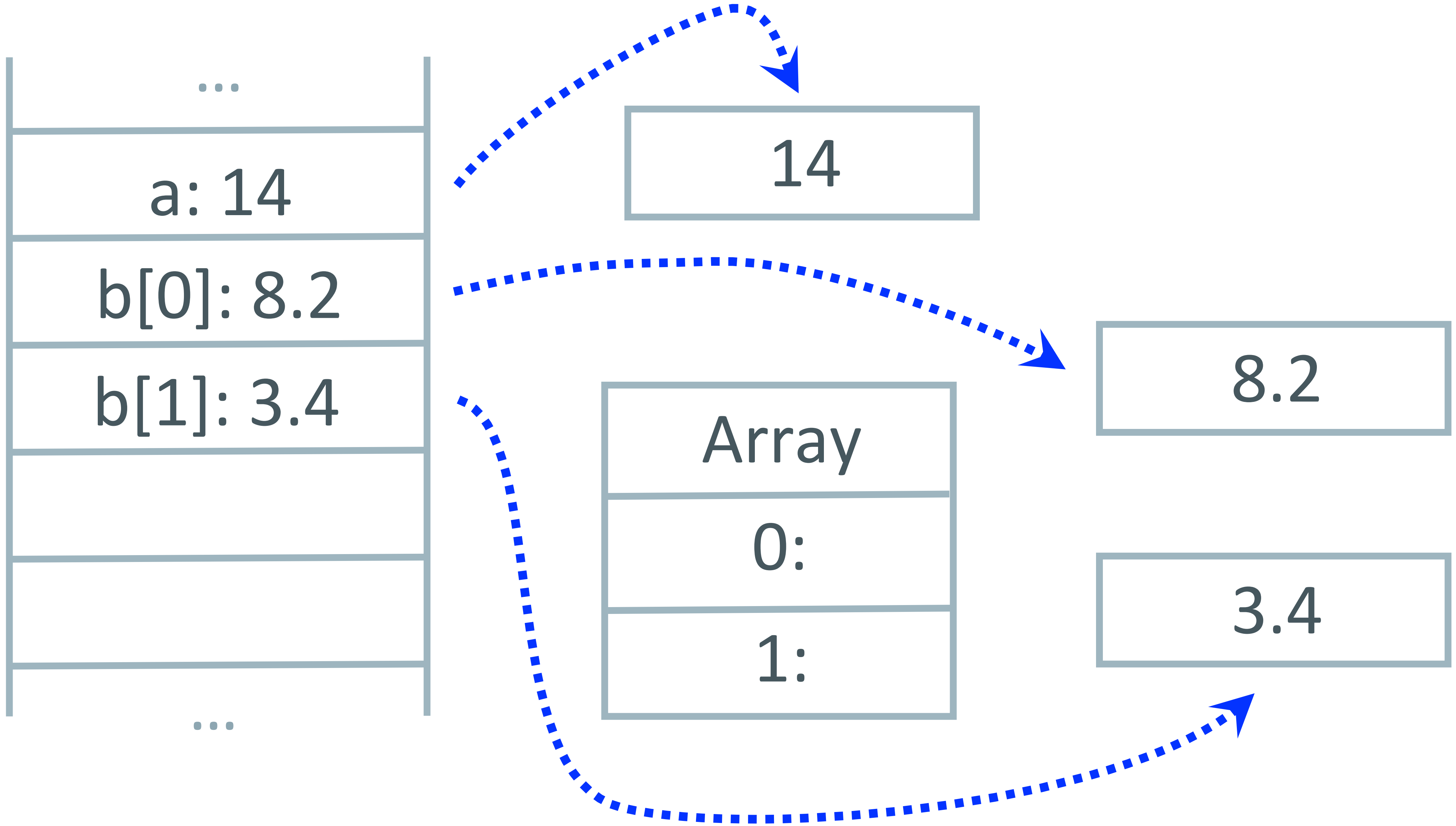
```
use cached my_method
call it with (x, y)
```

#binding

```
a = 14
b = [8.2, 3.4]
```


...
a:
b:
...
14
Array
0:
1:
8.2
3.4


...
a: 14
b[0]: 8.2
b[1]: 3.4
...


...
a: 14
b[0]: 8.2
b[1]: 3.4
...
14
Array
0:
1:
8.2
3.4

ObjectSpace

set_trace_func

How JRuby+Truffle Deoptimizes

1. Recreate the interpreter stack frame
2. Jump from the JITed code into the interpreter
3. Allow us to force threads to do this

```
loop do
  a = 14
  b = 2
  a + b
end
```

```
loop do
  a = 14
  b = 2
  a + b
  deoptimize! if should_deoptimize?
end
```

```
loop do
  a = 14
  b = 2
  a + b
  read the safepoint page
end
```

JRuby+Truffle Performance

86%

RubySpec language specs

rubyspec.org, Brian Shirai et al

#send

#binding

Method invalidation

Float

Threads

Frame-local variables

C extensions

Encodings

ObjectSpace

Regexp

Thread#raise

#eval

Fixnum to Bignum promotion

set_trace_func

Proc#binding

Closures

Concurrency

Constant invalidation

Debugging

Method invalidation

#send

#binding

Float

Threads

Frame-local variables

C extensions

Encodings

ObjectSpace

Regexp

Thread#raise

#eval

Fixnum to Bignum promotion

set_trace_func

Proc#binding

Closures

Concurrency

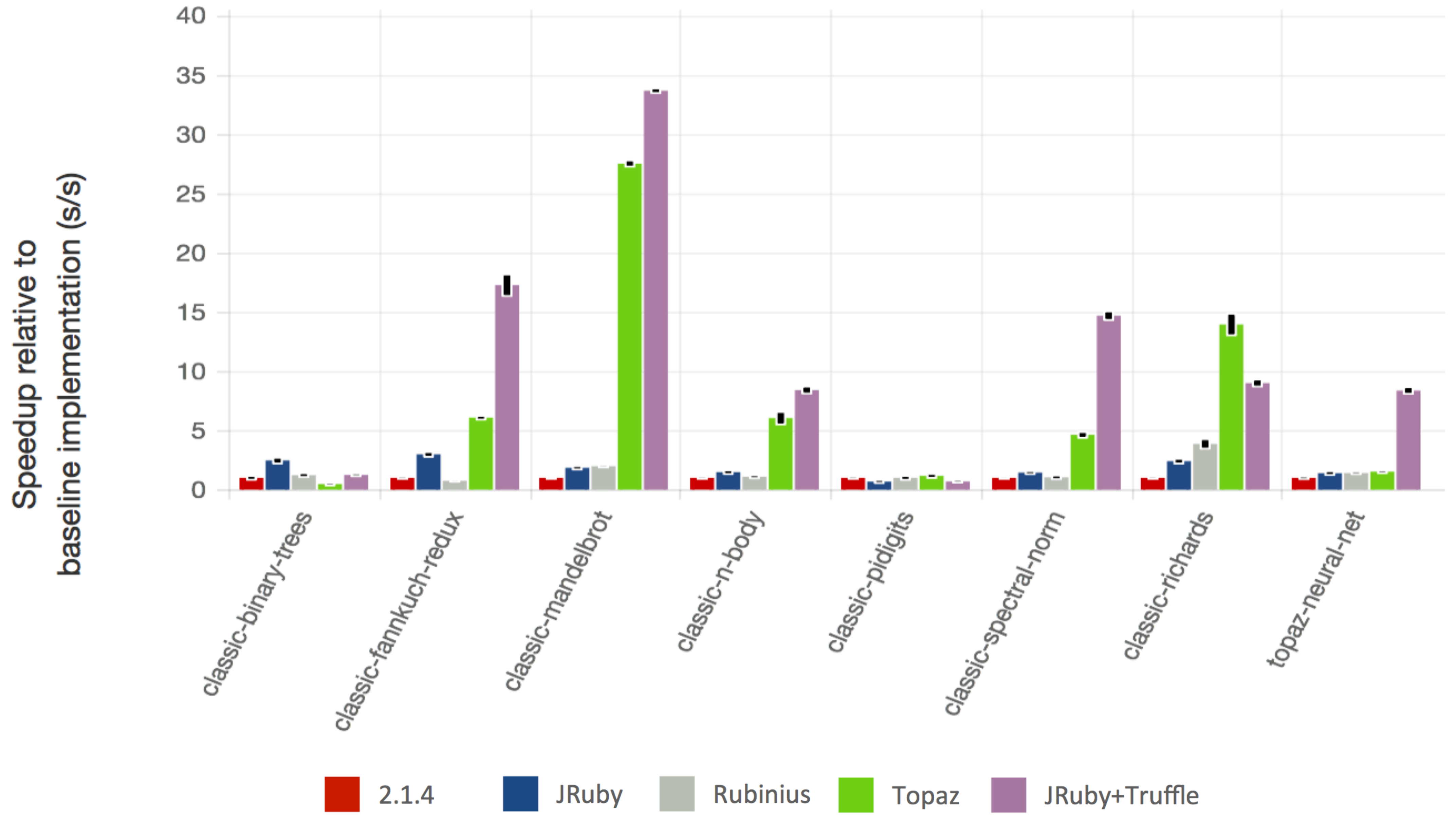
Constant invalidation

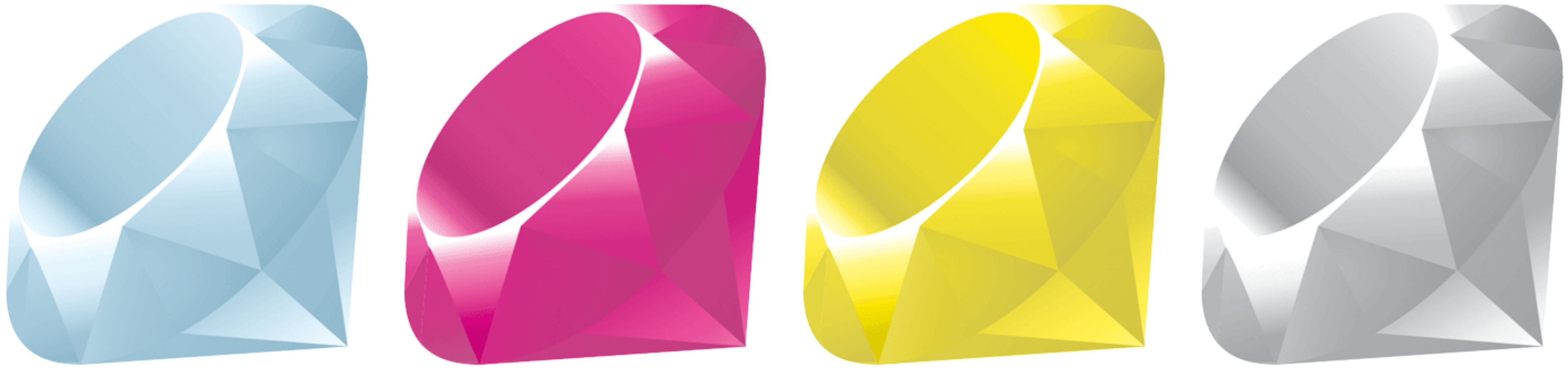
Debugging

ORACLE

No, we can’t run Rails yet

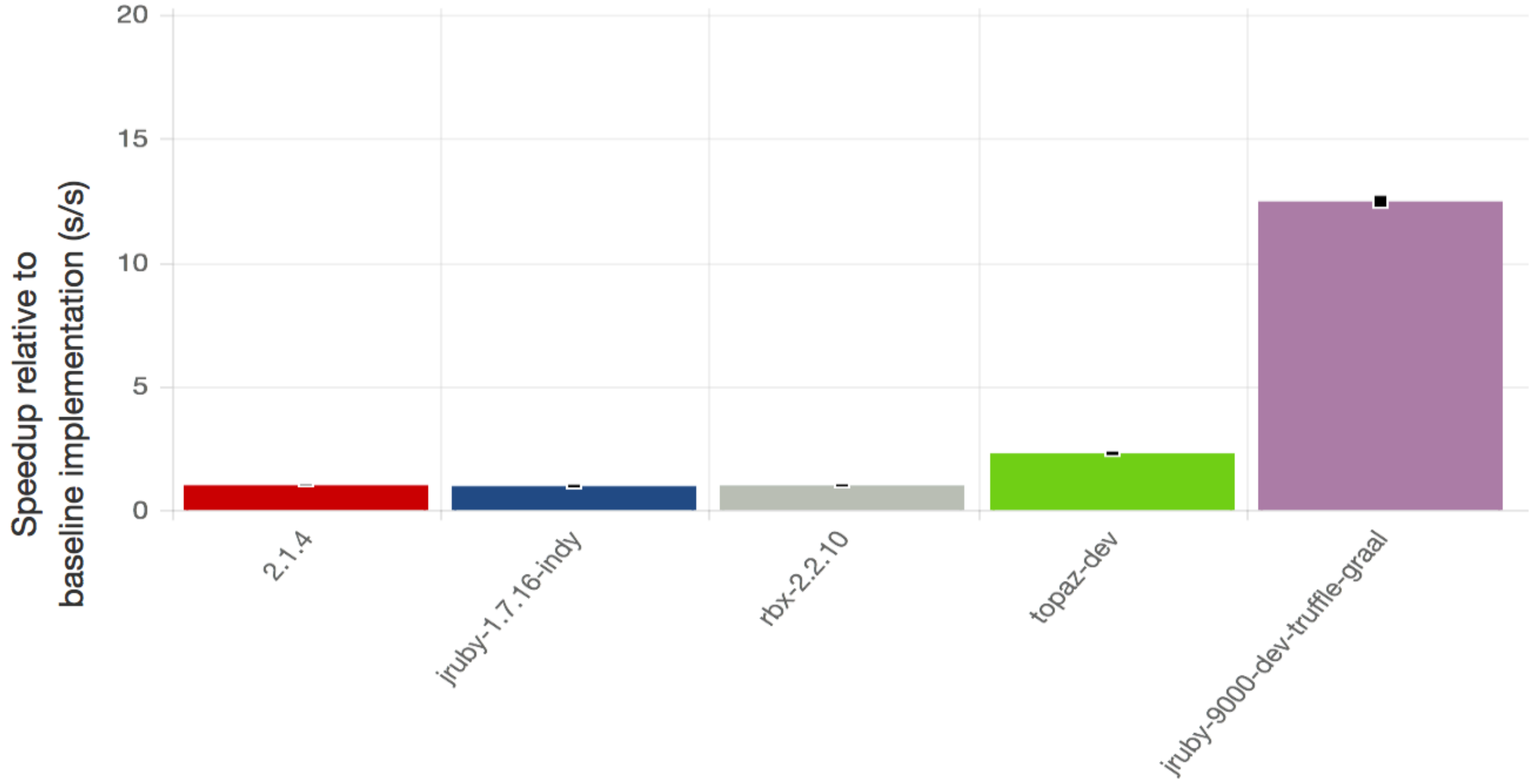
but we’re working towards it

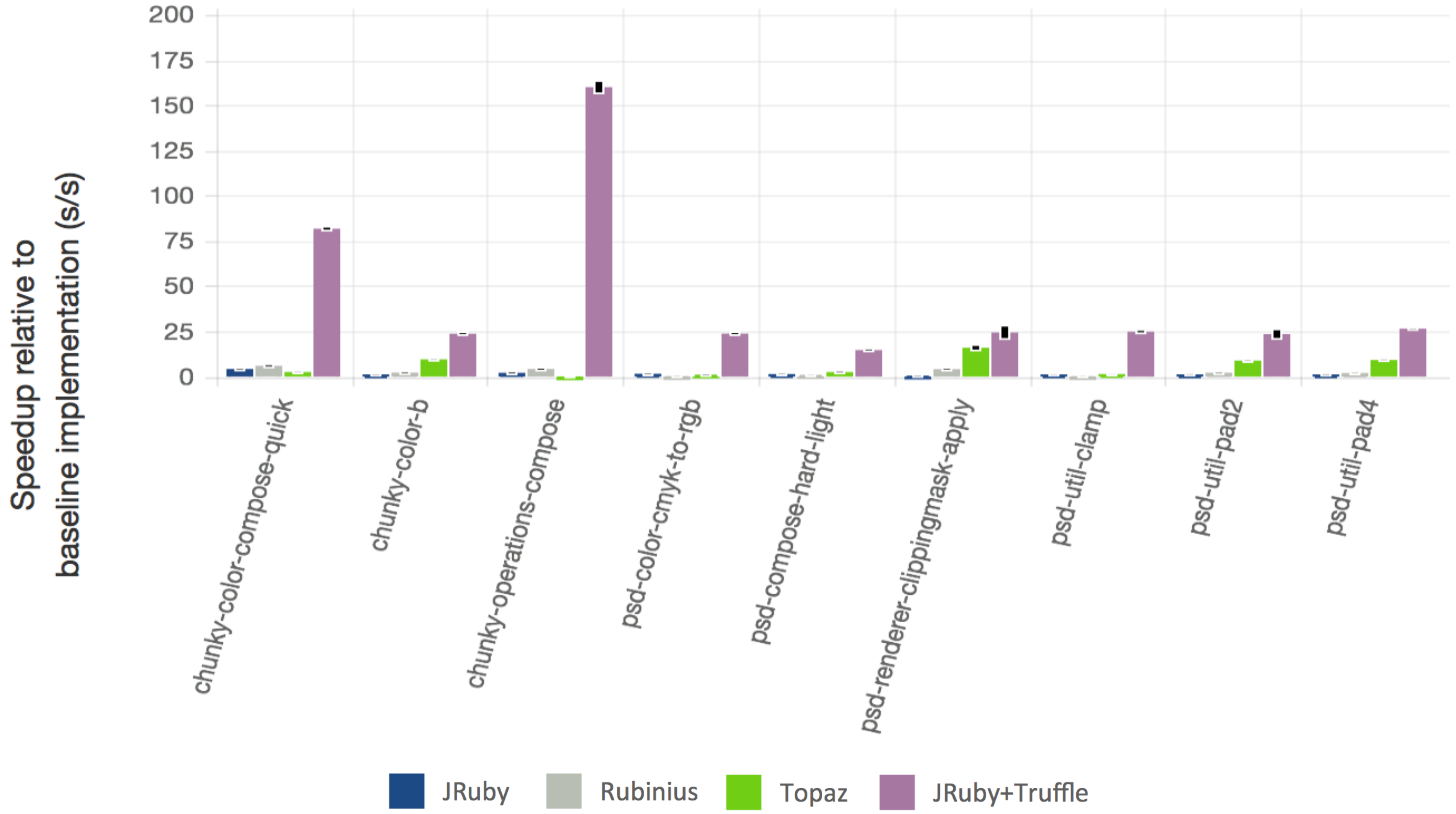

Speedup relative to baseline implementation (s/s)
40
35
30
25
20
15
10
5
0
classic-binary-trees
classic-fannkuch-redux
classic-mandelbrot
classic-n-body
classic-pidigits
classic-spectral-norm
classic-richards
topaz-neural-net
2.1.4
JRuby
Rubinius
Topaz
JRuby+Truffle



chunky_png and psd.rb

Willem van Bergen, Ryan LeFevre, Kelly Sutton, Layer Vault, Floorplanner et al

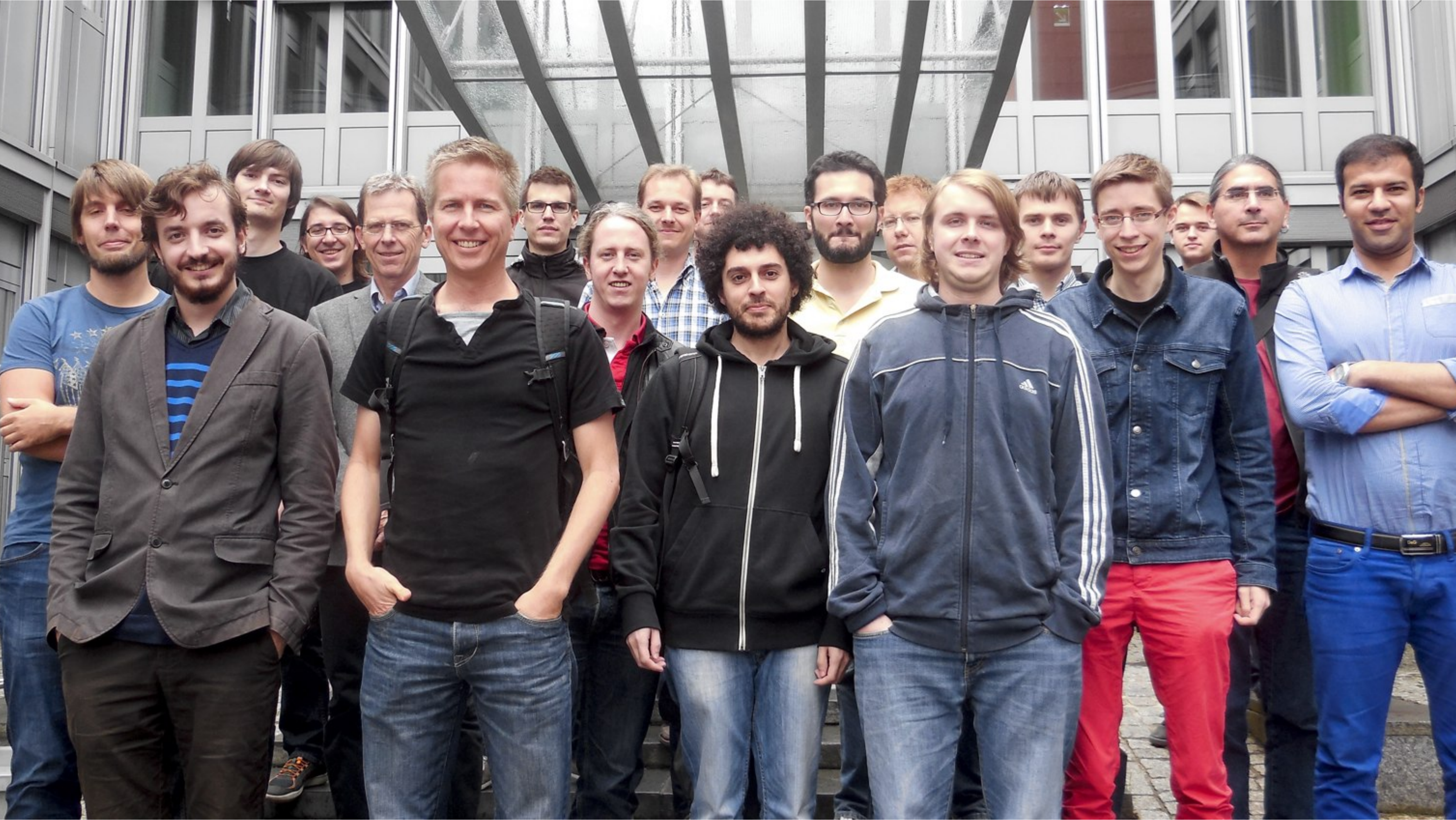

Speedup relative to baseline implementation (s/s)
20
15
10
5
0
2.1.4
jruby-1.7.16-indy
rbx-2.2.10
topaz-dev
jruby-9000-dev-truffle-graal


Speedup relative to baseline implementation (s/s)
200
175
150
125
100
75
50
25
0
chunky-color-compose-quick
chunky-color-b
chunky-operations-compose
psd-color-cmyk-to-rgb
psd-compose-hard-light
psd-renderer-clippingmask-apply
psd-util-clamp
psd-util-pad2
psd-util-pad4
JRuby
Rubinius
Topaz
JRuby+Truffle

ORACLE

Conclusions

Building on other projects



@ChrisGSeaton

chrisseaton.com/rubytruffle/deoptimizing

Safe Harbor Statement

The preceding is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

ORACLE

Hardware and Software

Engineered to Work Together