ORACLE®

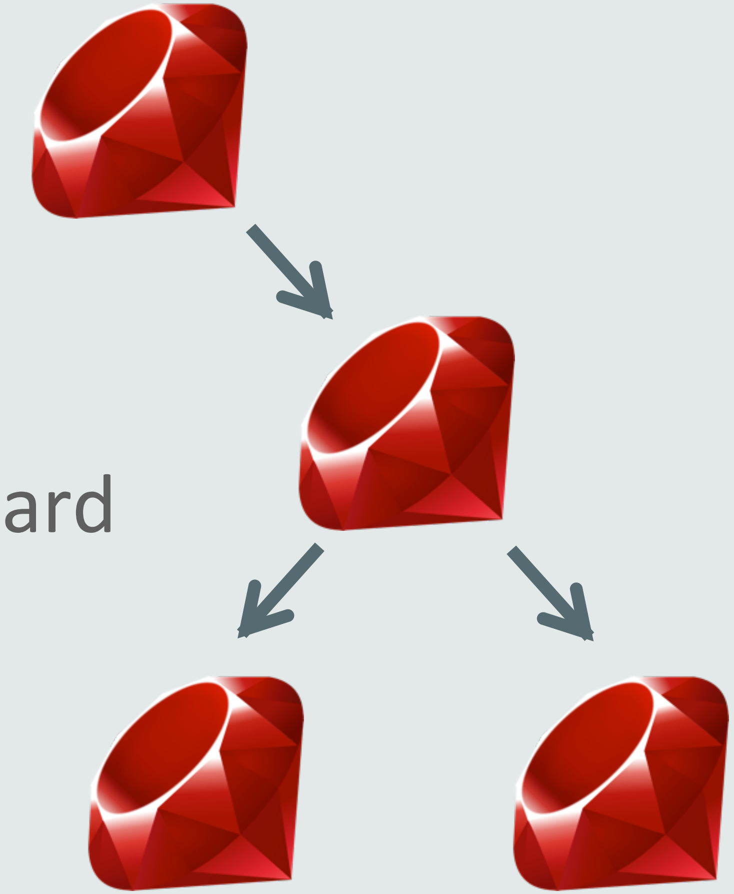
JRuby+Truffle

A tour through a new Ruby implementation

Chris Seaton
@ChrisGSeaton
Oracle Labs

Benoit Daloze
@eregontp
JKU Linz

Kevin Menard
@nirvdrum
Oracle Labs

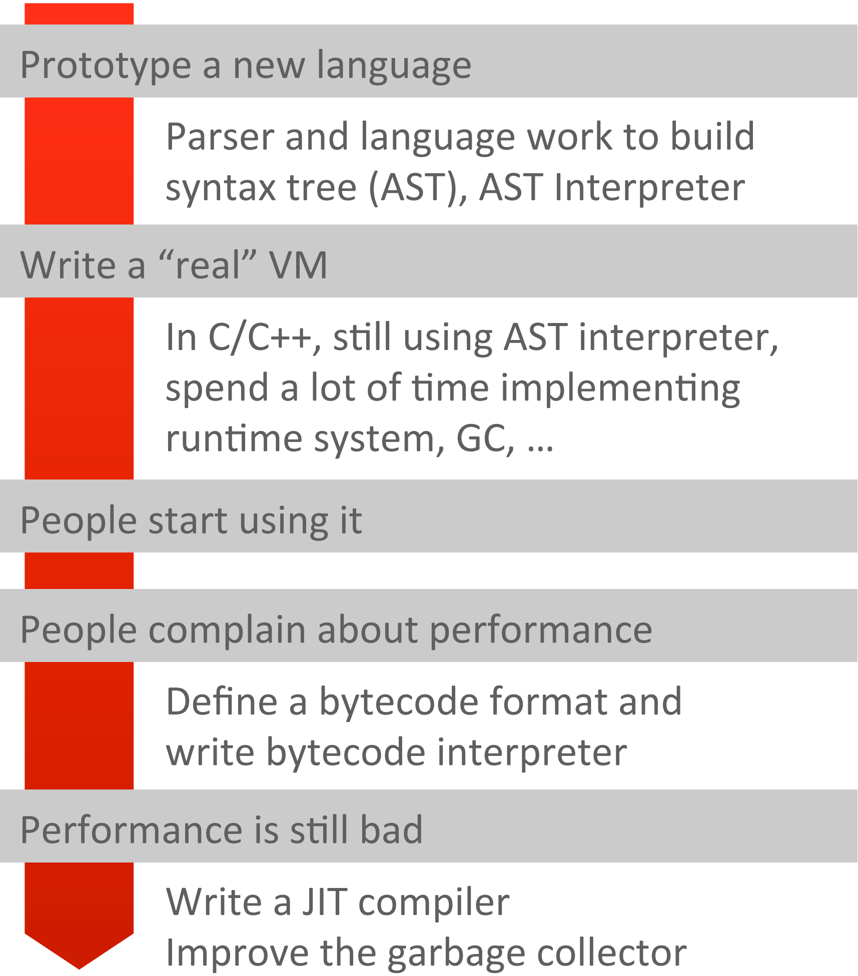


Safe Harbor Statement

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.

What is the big idea?

Current situation



Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …

People start using it

People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

Current situation

Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …

People start using it

People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

How it should be

Prototype a new language in Java

Parser and language work to build syntax tree (AST)
Execute using AST interpreter

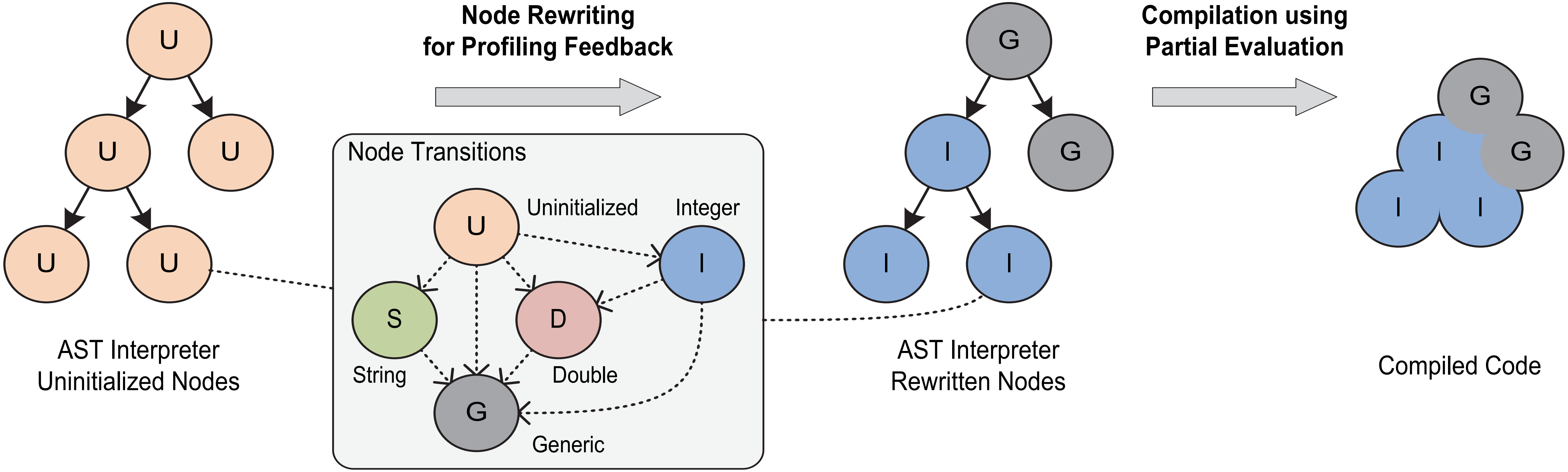
People start using it

And it is already fast

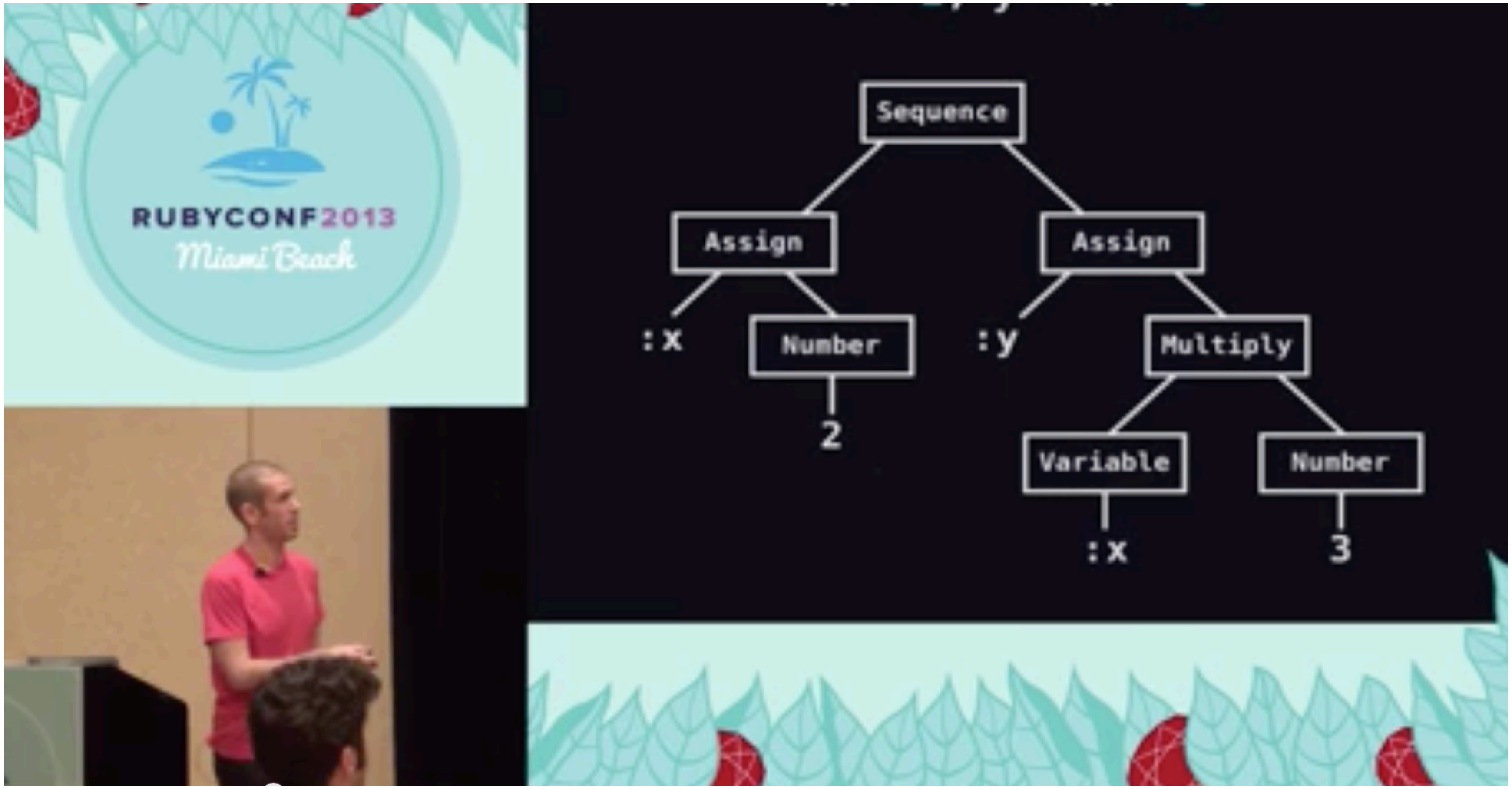
What are Truffle and Graal?

Truffle: a framework for writing AST interpreters for languages in Java

Graal: a dynamic compiler (JIT) for Java, written in Java, as a Java library

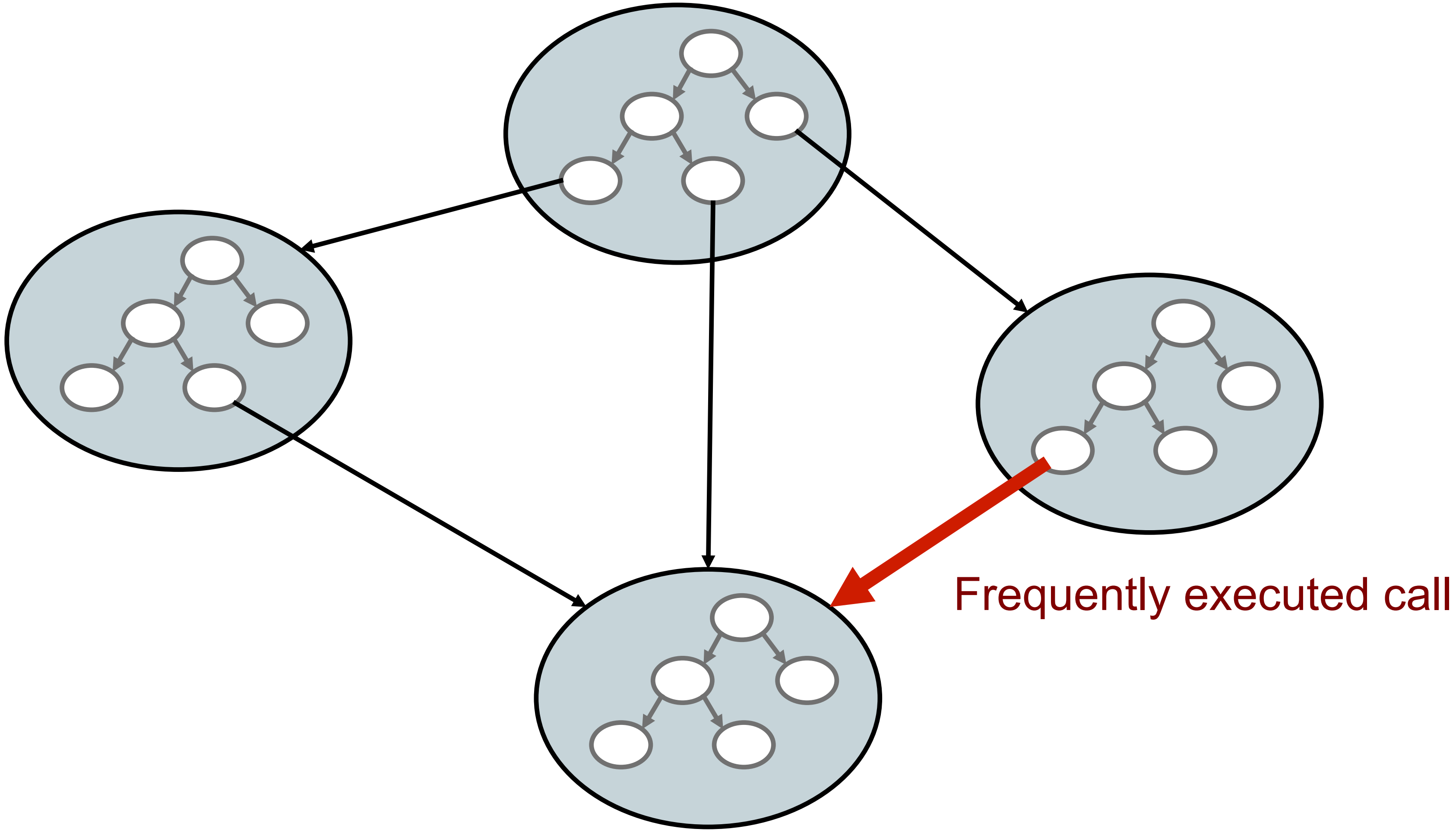


T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

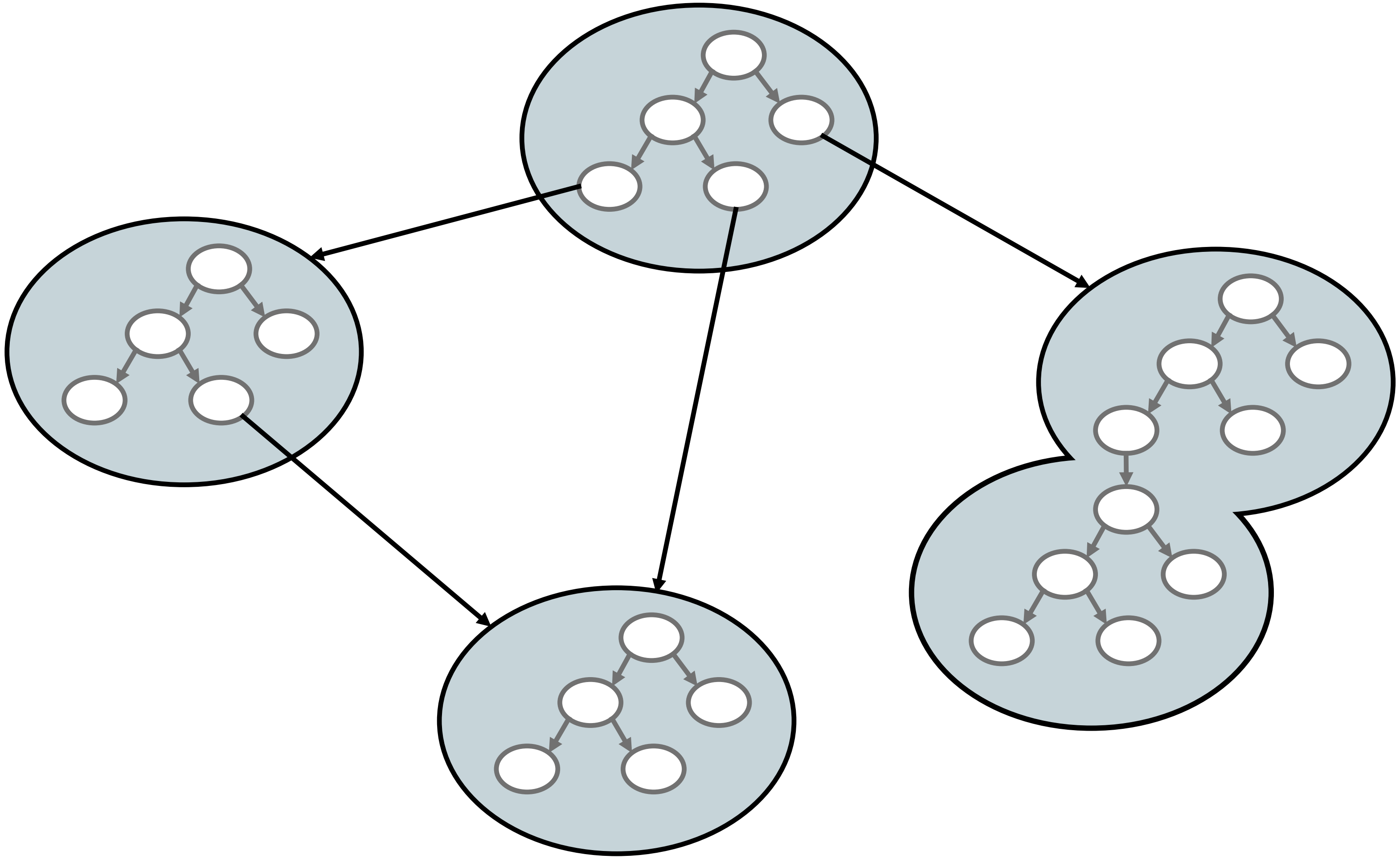


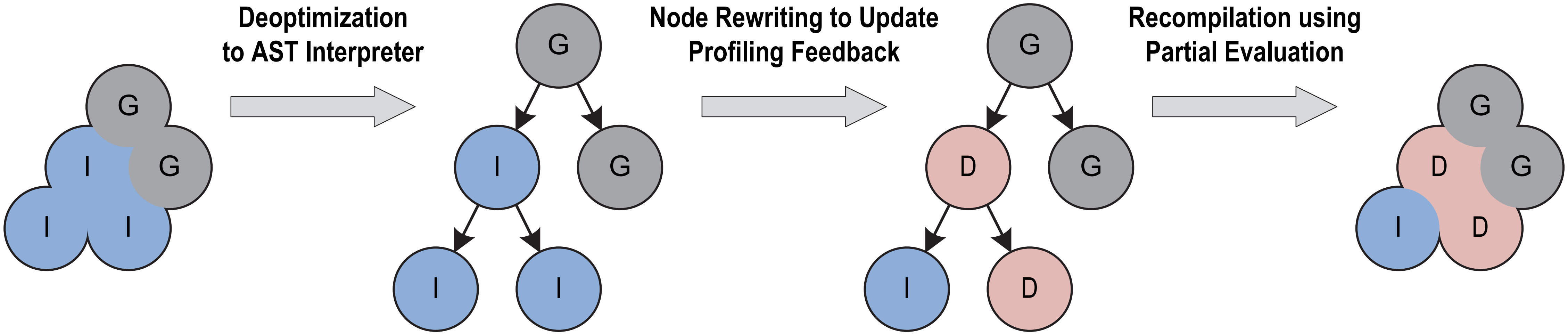
codon.com/compilers-for-free

Presentation, by Tom Stuart , licensed under a Creative Commons Attribution ShareAlike 3.0



Frequently executed call





Deoptimization to AST Interpreter

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation

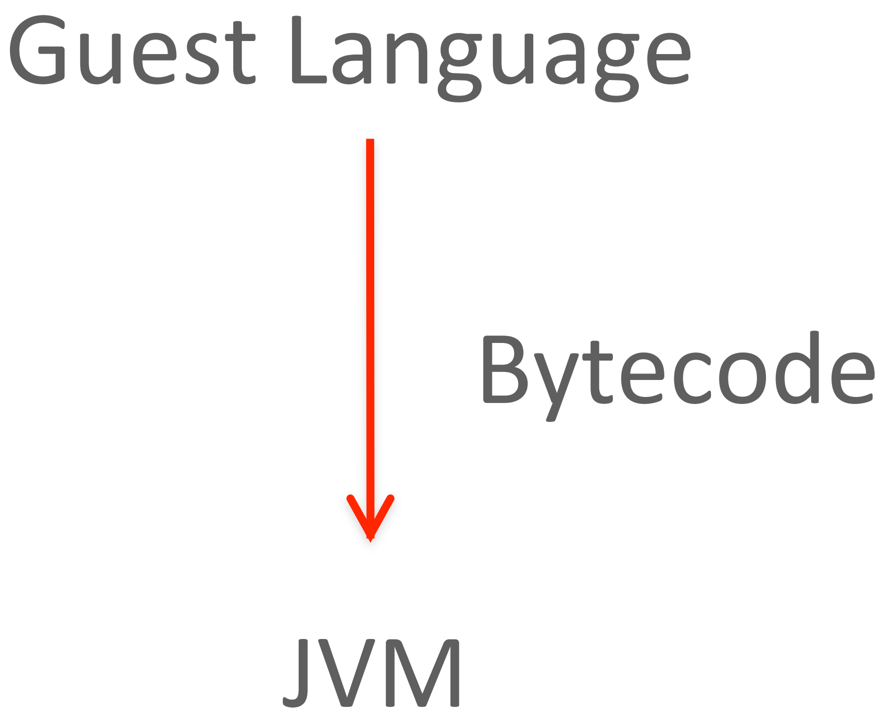
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

John Tenniel illustrations public domain in the UK and US

John Tenniel illustrations public domain in the UK and US

```
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
```

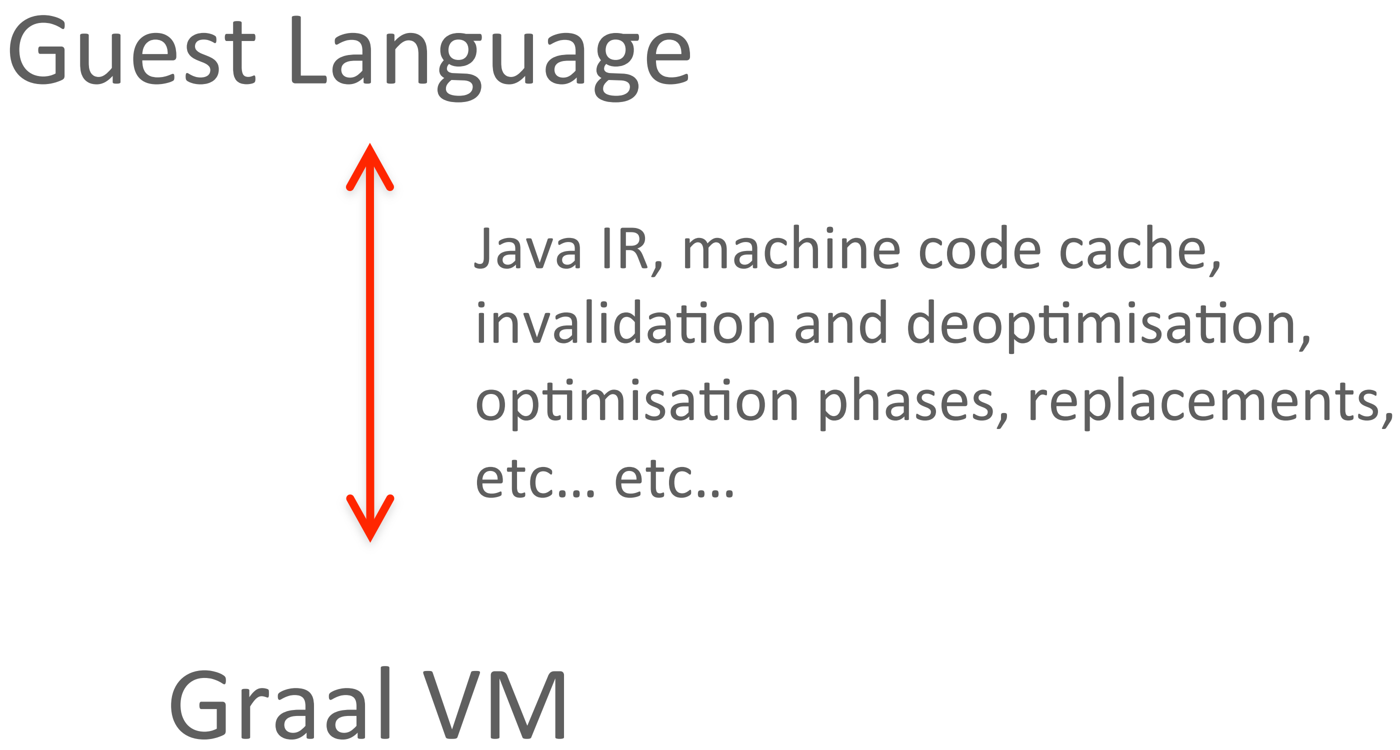
```
t1 = Fixnum(a) + Fixnum(b)
deoptimize! if t1.overflowed?
t2 = Fixnum(t1) + Fixnum(c)
deoptimize! if t2.overflowed?
```



Guest Language

Bytecode

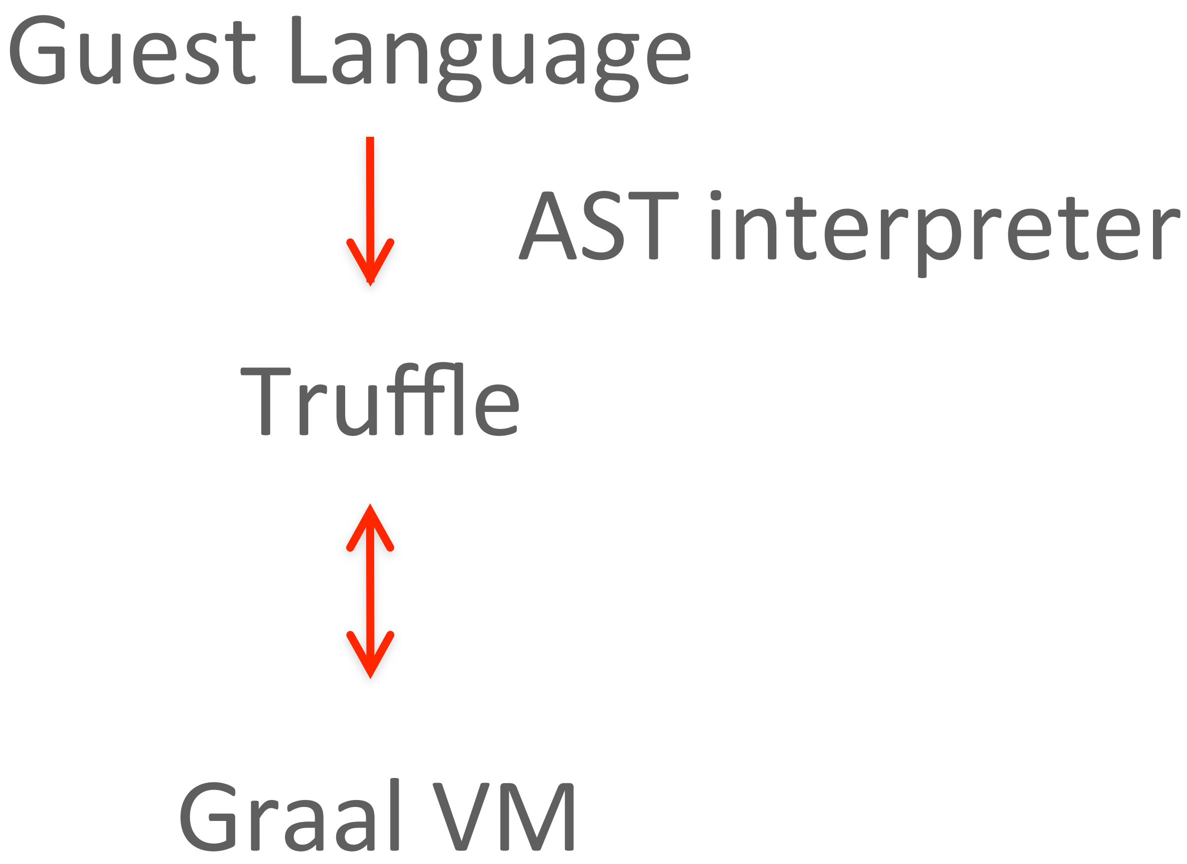
JVM



Guest Language

Java IR, machine code cache, invalidation and deoptimisation, optimisation phases, replacements, etc… etc…

Graal VM



Guest Language

AST interpreter

Truffle

Graal VM

A tour through Ruby, Truffle and Graal

Specializations

```
class Array

  def [](index=Fixnum())
    # return element at index
  end

  def [](index=Fixnum(), num=Fixnum())
    # return num elements starting at index
  end

  def [](range=Range())
    # return elements from range.start to range.end
  end

  def [](index)
    # coerce index and dispatch
  end
```

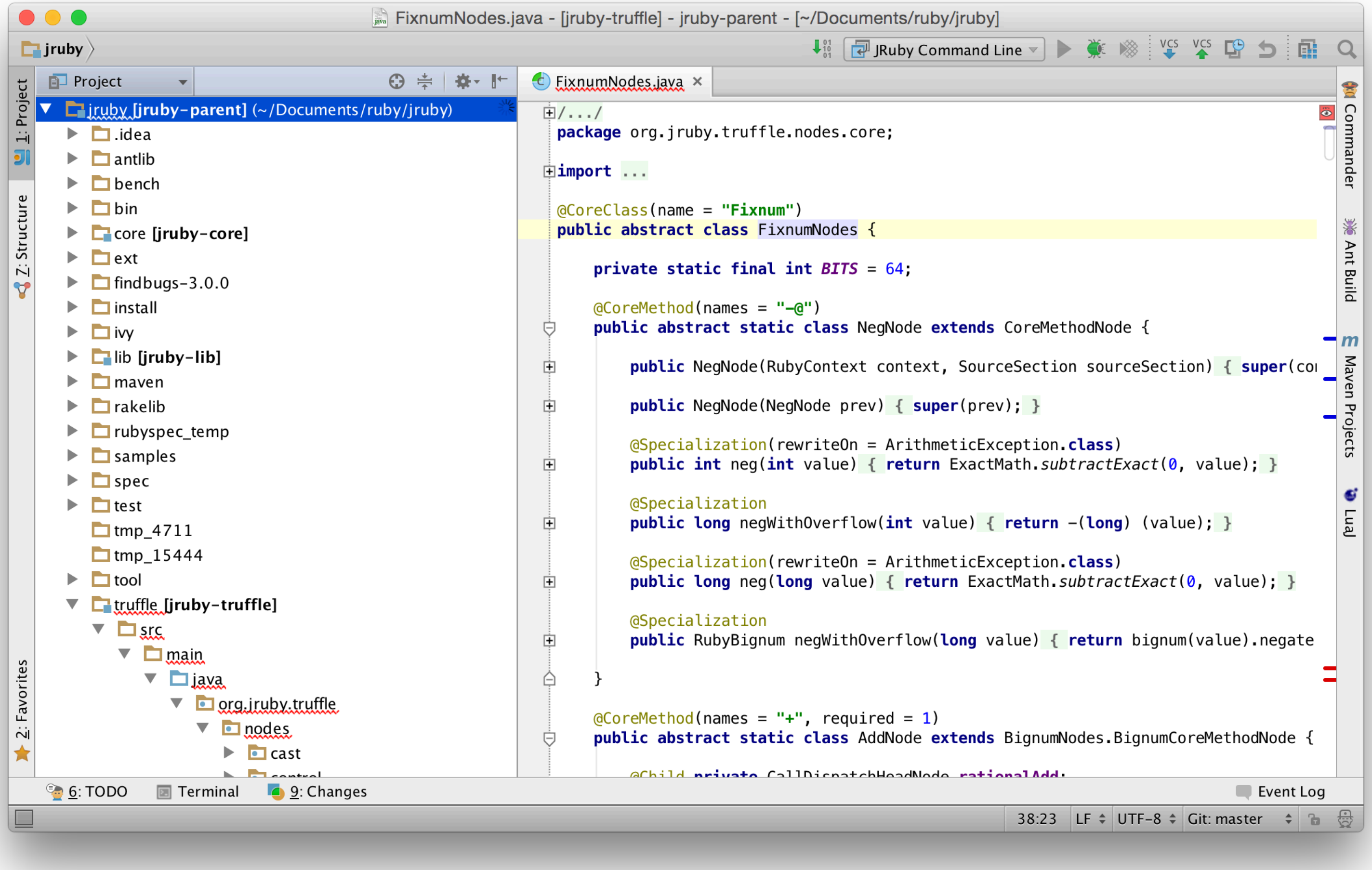
B. Shirai, Rubinius 3.0, Part 5, The Language, http://rubini.us/2014/11/14/rubinius-3-0-part-5-the-language/

Specializations

```
def clamp(num, min, max)
  [min, num, max].sort[1]
end
```

chunky_png and psd.rb, Willem van Bergen, Ryan LeFevre, Kelly Sutton, Layer Vault, Floorplanner et al

From Fixnum#+ to 0x03 0x70



```
jruby [jruby-parent] (~/Documents/ruby/jruby)
  .idea
  antlib
  bench
  bin
  core [jruby-core]
  ext
  findbugs-3.0.0
  install
  ivy
  lib [jruby-lib]
  maven
  rakelib
  rubyspec_temp
  samples
  spec
  test
  tmp_4711
  tmp_15444
  tool
  truffle [jruby-truffle]
    src
      main
        java
          org.jruby.truffle
            nodes
              cast
```

```java
/.../
package org.jruby.truffle.nodes.core;

import ...

@CoreClass(name = "Fixnum")
public abstract class FixnumNodes {

    private static final int BITS = 64;

    @CoreMethod(names = "-@")
    public abstract static class NegNode extends CoreMethodNode {

        public NegNode(RubyContext context, SourceSection sourceSection) { super(co

        public NegNode(NegNode prev) { super(prev); }

        @Specialization(rewriteOn = ArithmeticException.class)
        public int neg(int value) { return ExactMath.subtractExact(0, value); }

        @Specialization
        public long negWithOverflow(int value) { return -(long) (value); }

        @Specialization(rewriteOn = ArithmeticException.class)
        public long neg(long value) { return ExactMath.subtractExact(0, value); }

        @Specialization
        public RubyBignum negWithOverflow(long value) { return bignum(value).negate

    }

    @CoreMethod(names = "+", required = 1)
    public abstract static class AddNode extends BignumNodes.BignumCoreMethodNode {

        @Child private CallDispatchHeadNode rationalAdd;
```

Digging through ObjectSpace and deoptimization

- Deoptimize
- Get a consistent view of memory: safepoints
- Find all reachable objects
- Iterate through them

Digging through ObjectSpace and deoptimization

```
public Map<Long, RubyBasicObject> collectLiveObjects() {
    liveObjects = new HashMap<>();

    visitor = new ObjectGraphVisitor() {
        @Override
        public boolean visit(RubyBasicObject object) {
            return liveObjects.put(object.getObjectID(), object) == null;
        }
    };

    context.getSafepointManager().pauseAllThreadsAndExecute(new Consumer<RubyThread>() {

        @Override
        public void accept(RubyThread currentThread) {
            synchronized (liveObjects) {
                visitor.visit(currentThread);
                context.getCoreLibrary().getGlobalVariablesObject().visitObjectGraph(visitor);
                context.getCoreLibrary().getMainObject().visitObjectGraph(visitor);
                context.getCoreLibrary().getObjectClass().visitObjectGraph(visitor);
                visitCallStack(visitor);
            }
        }

    });

    return Collections.unmodifiableMap(liveObjects);
}
```

Does it really implement Ruby?

93%

RubySpec language specs

Brian Shirai et al

24%

RubySpec core library specs

Brian Shirai et al

Method invalidation

#send

#binding

Float

Threads

Frame-local variables

C extensions

Encodings

ObjectSpace

Regexp

Thread#raise

Fixnum to Bignum promotion

#eval

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

Fixnum to Bignum promotion

#eval

set_trace_func

Proc#binding

Closures

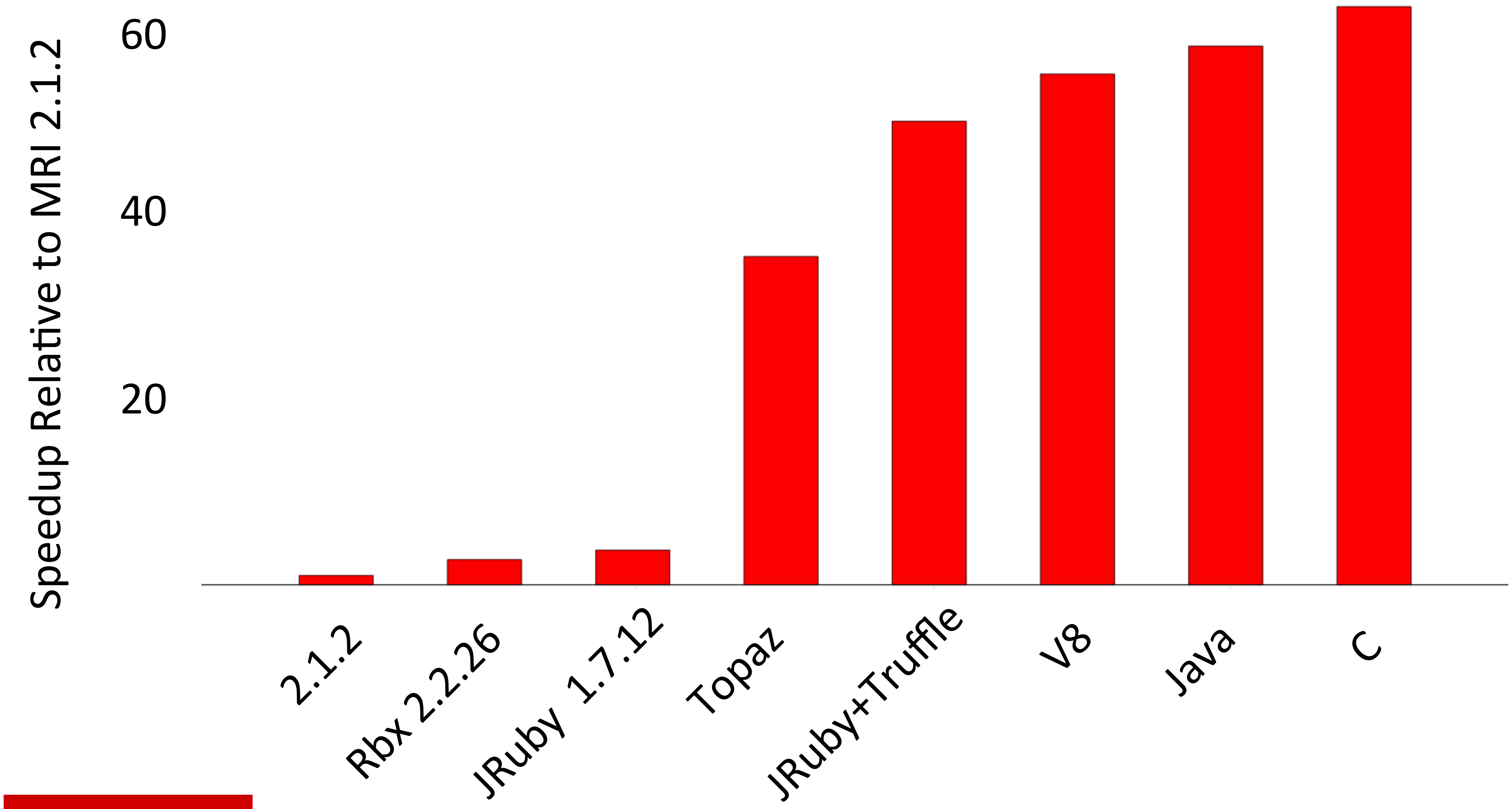
Concurrency

Constant invalidation

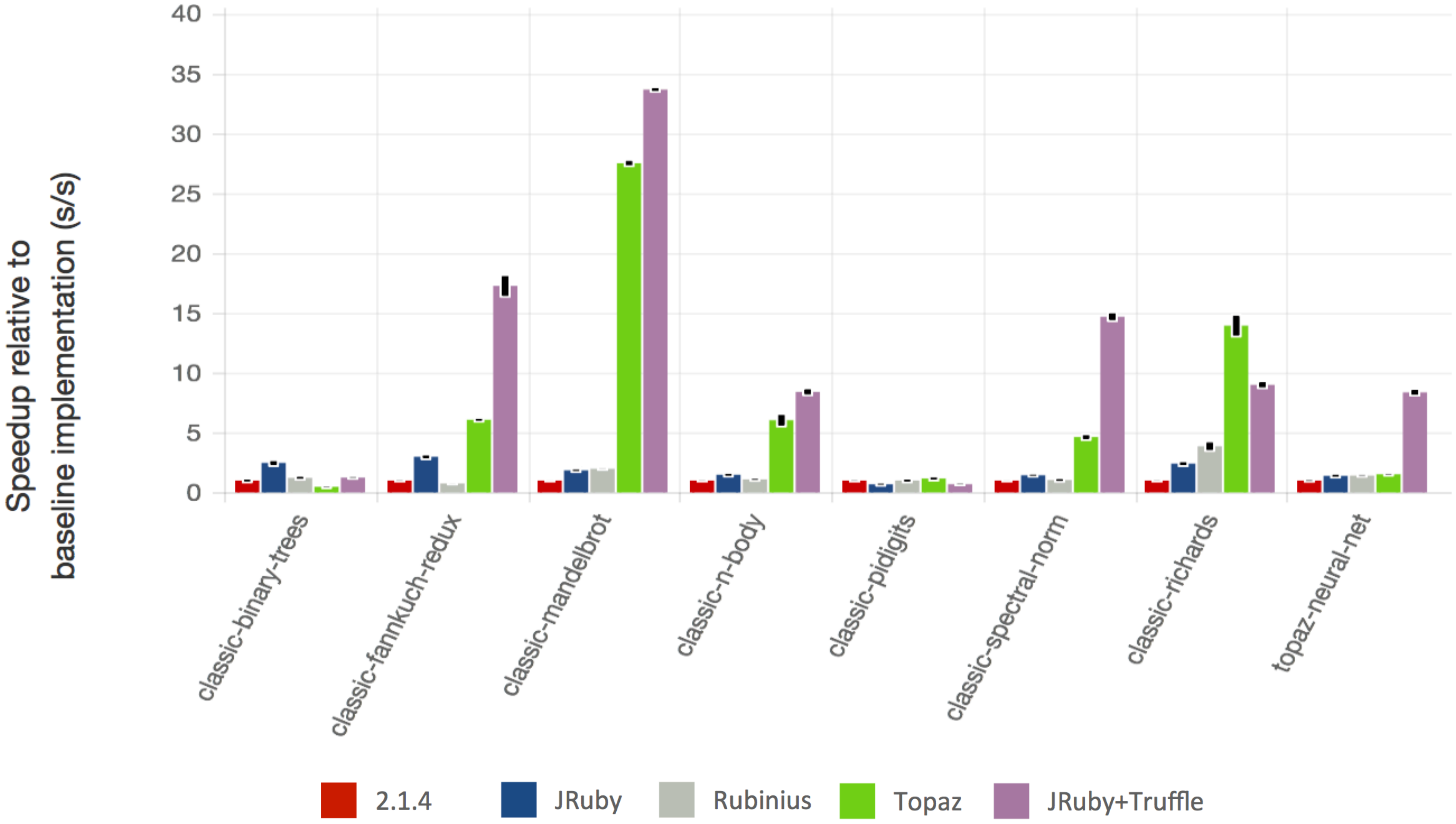
Debugging

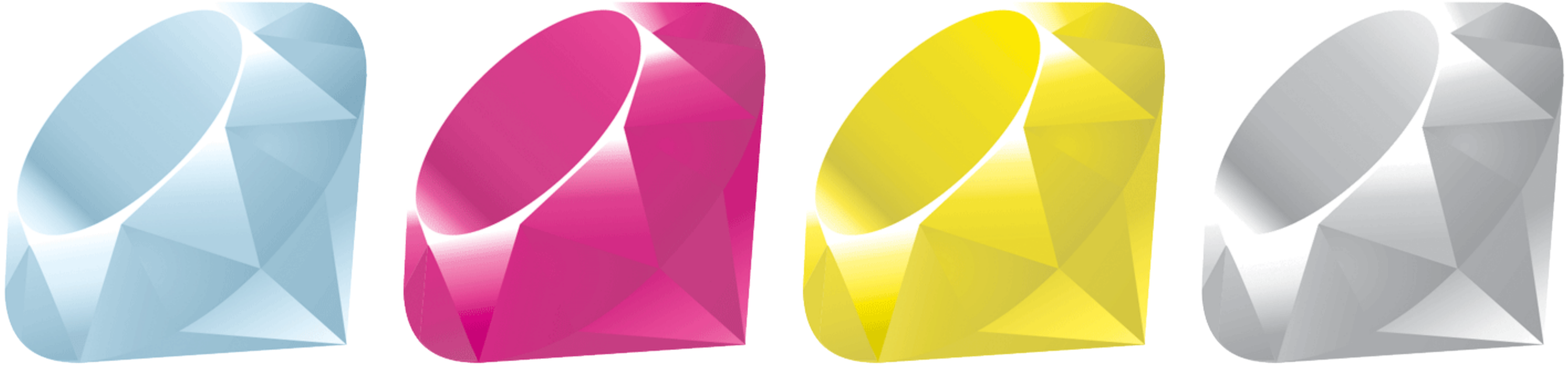
How fast is it?

Mandelbrot



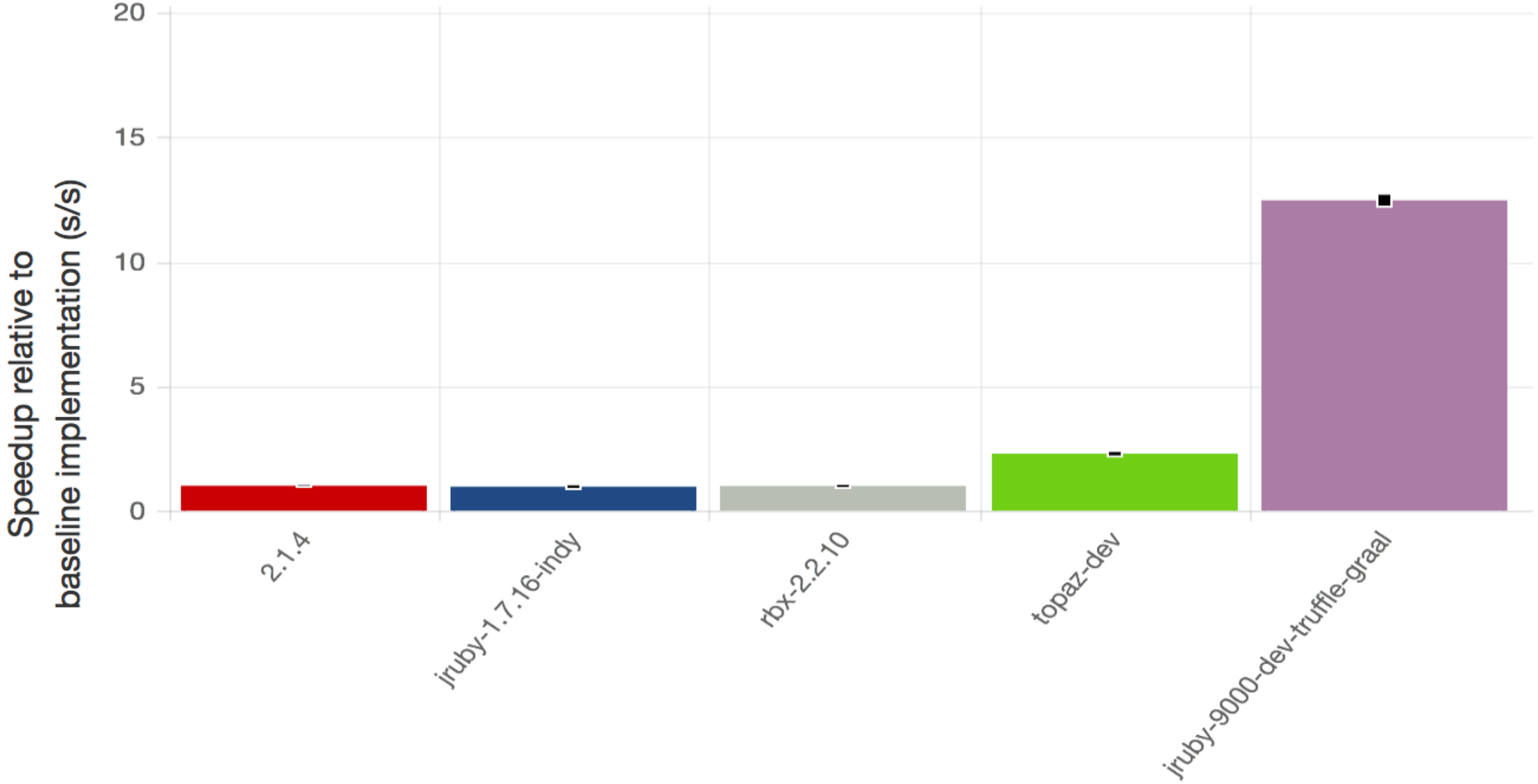
Speedup Relative to MRI 2.1.2
60
40
20
2.1.2
Rbx 2.2.26
JRuby 1.7.12
Topaz
JRuby+Truffle
V8
Java
C

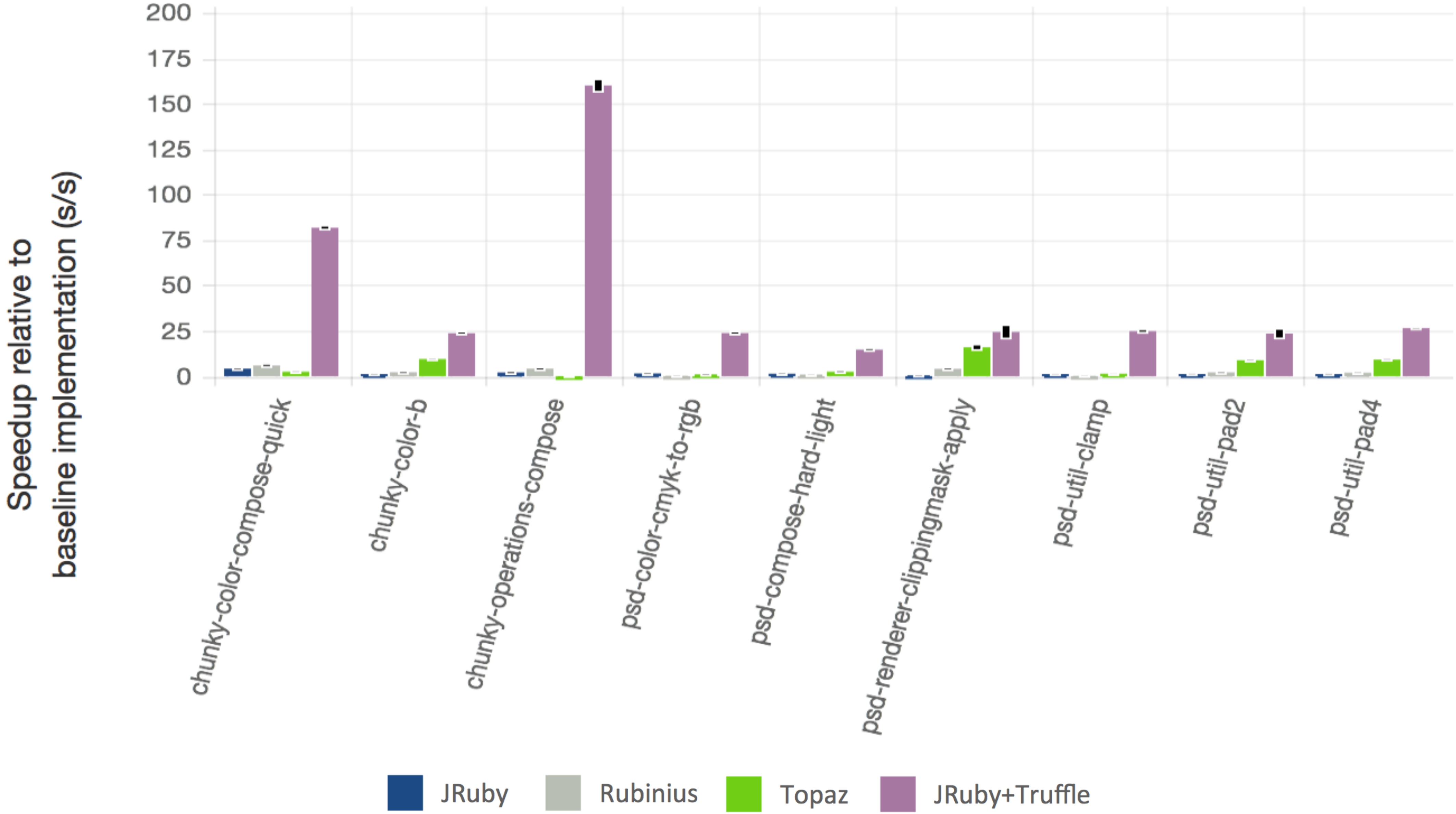

Speedup relative to baseline implementation (s/s)
40
35
30
25
20
15
10
5
0
classic-binary-trees
classic-fannkuch-redux
classic-mandelbrot
classic-n-body
classic-pidigits
classic-spectral-norm
classic-richards
topaz-neural-net
2.1.4
JRuby
Rubinius
Topaz
JRuby+Truffle



chunky_png and psd.rb

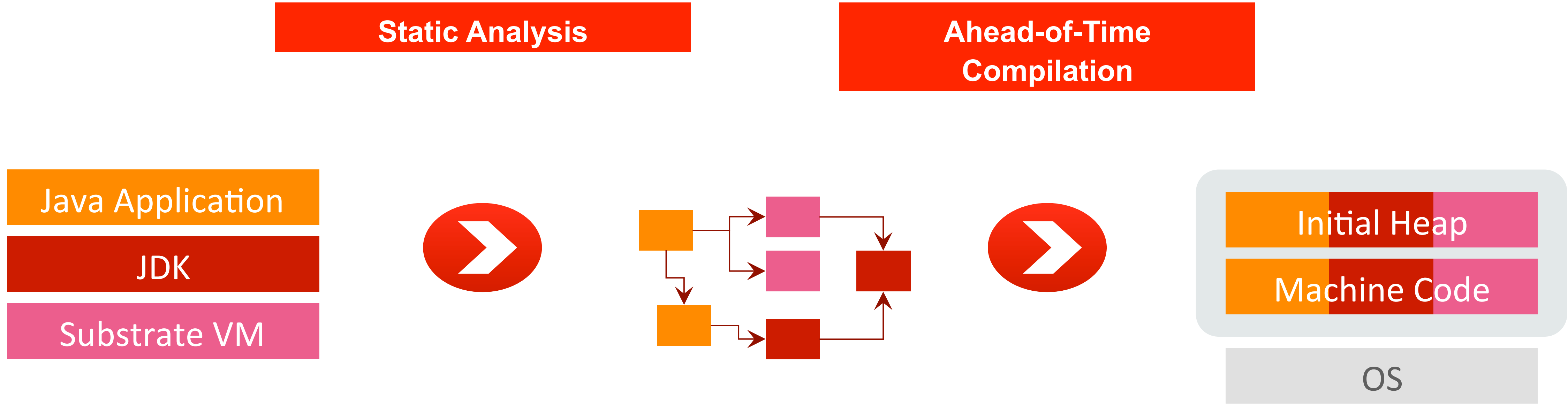
Willem van Bergen, Ryan LeFevre, Kelly Sutton, Layer Vault, Floorplanner et al


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

github.com/jruby/bench9000

How will we solve startup time, memory footprint and the JVM dependency?



Static Analysis

Ahead-of-Time Compilation

Java Application

JDK

Substrate VM

Initial Heap

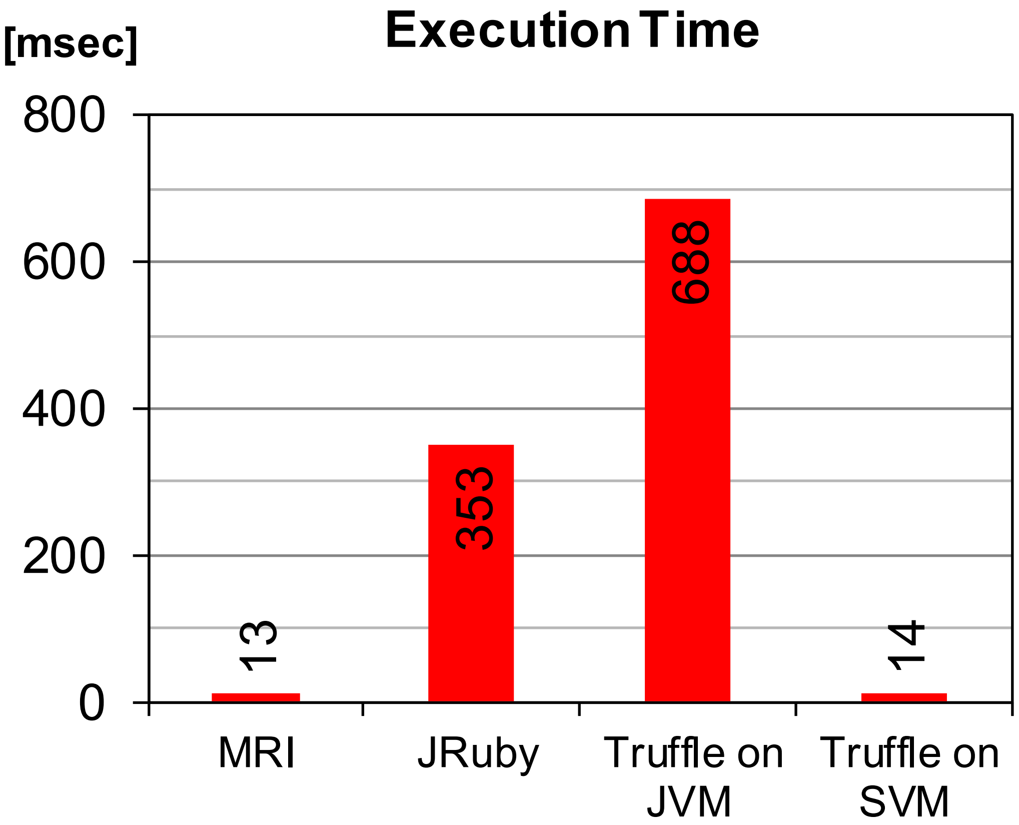
Machine Code

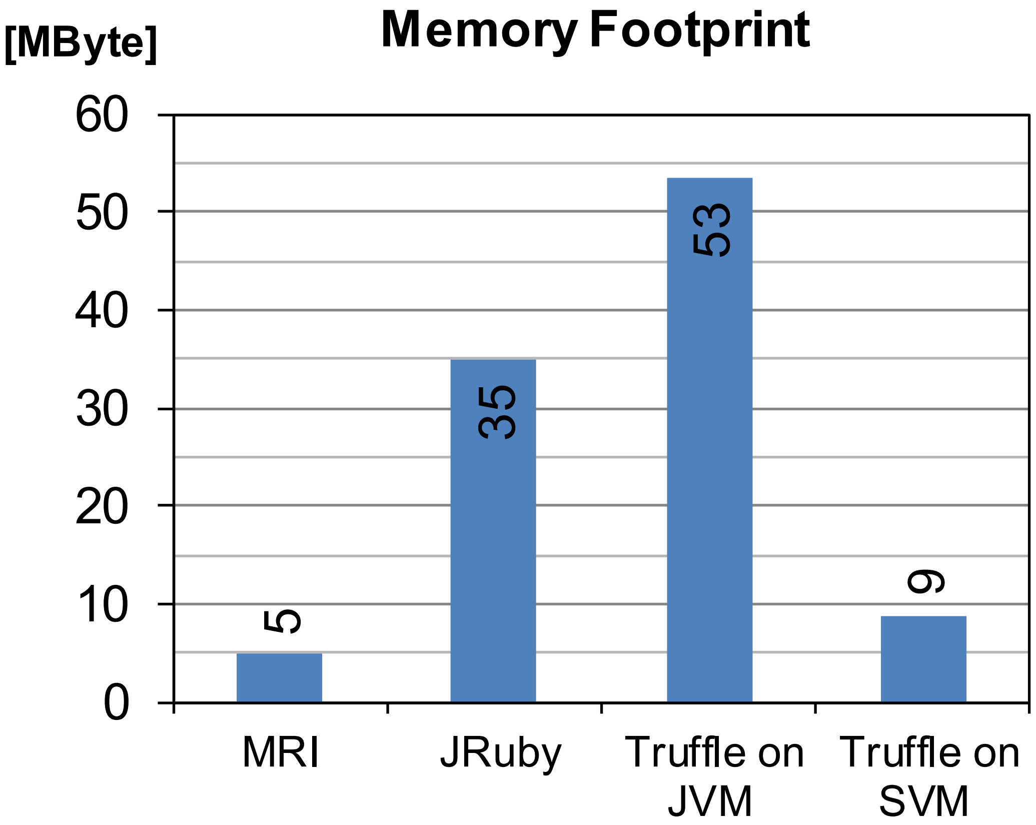
OS

All Java classes from application, JDK, and Substrate VM

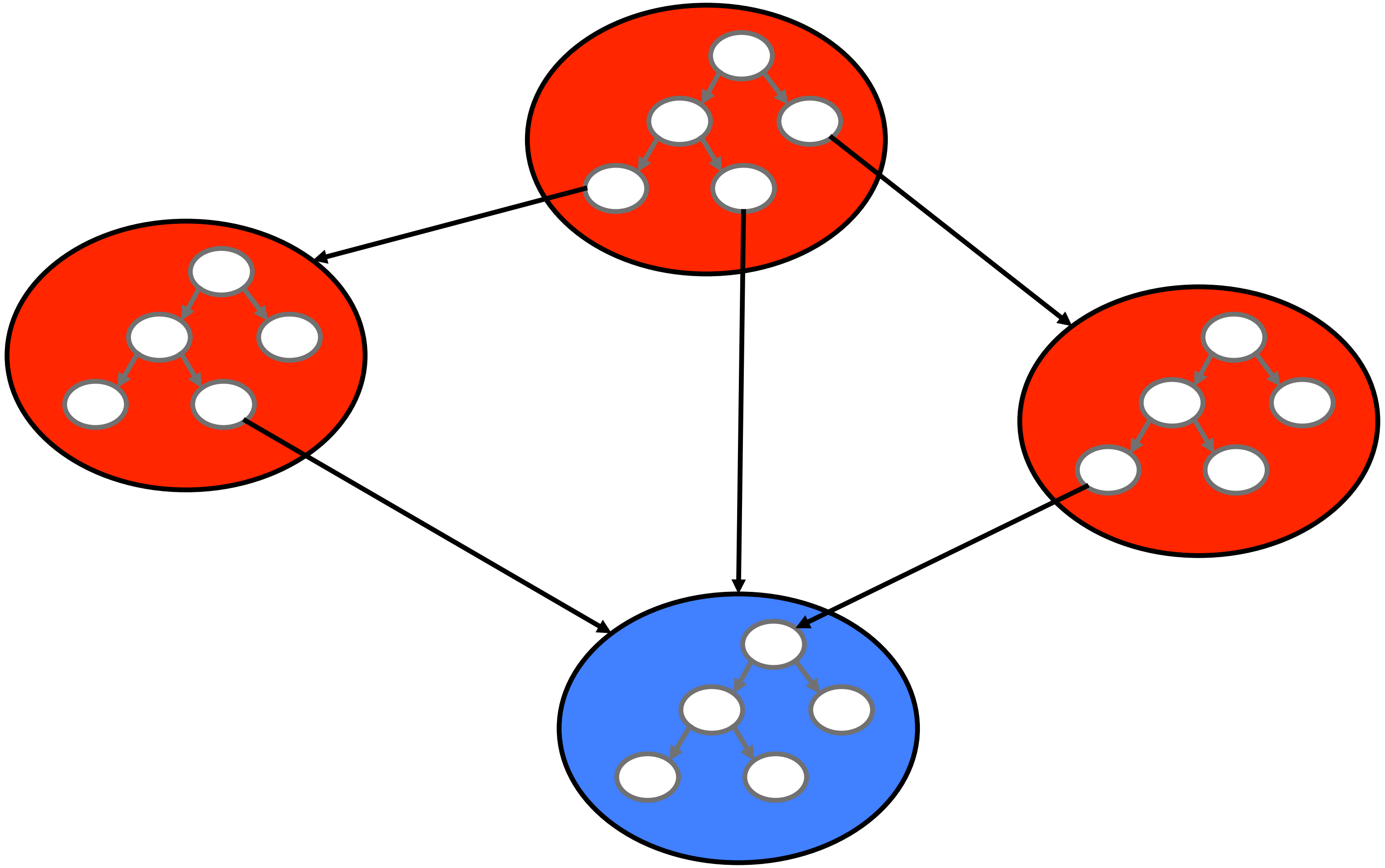
Reachable methods, fields, and classes

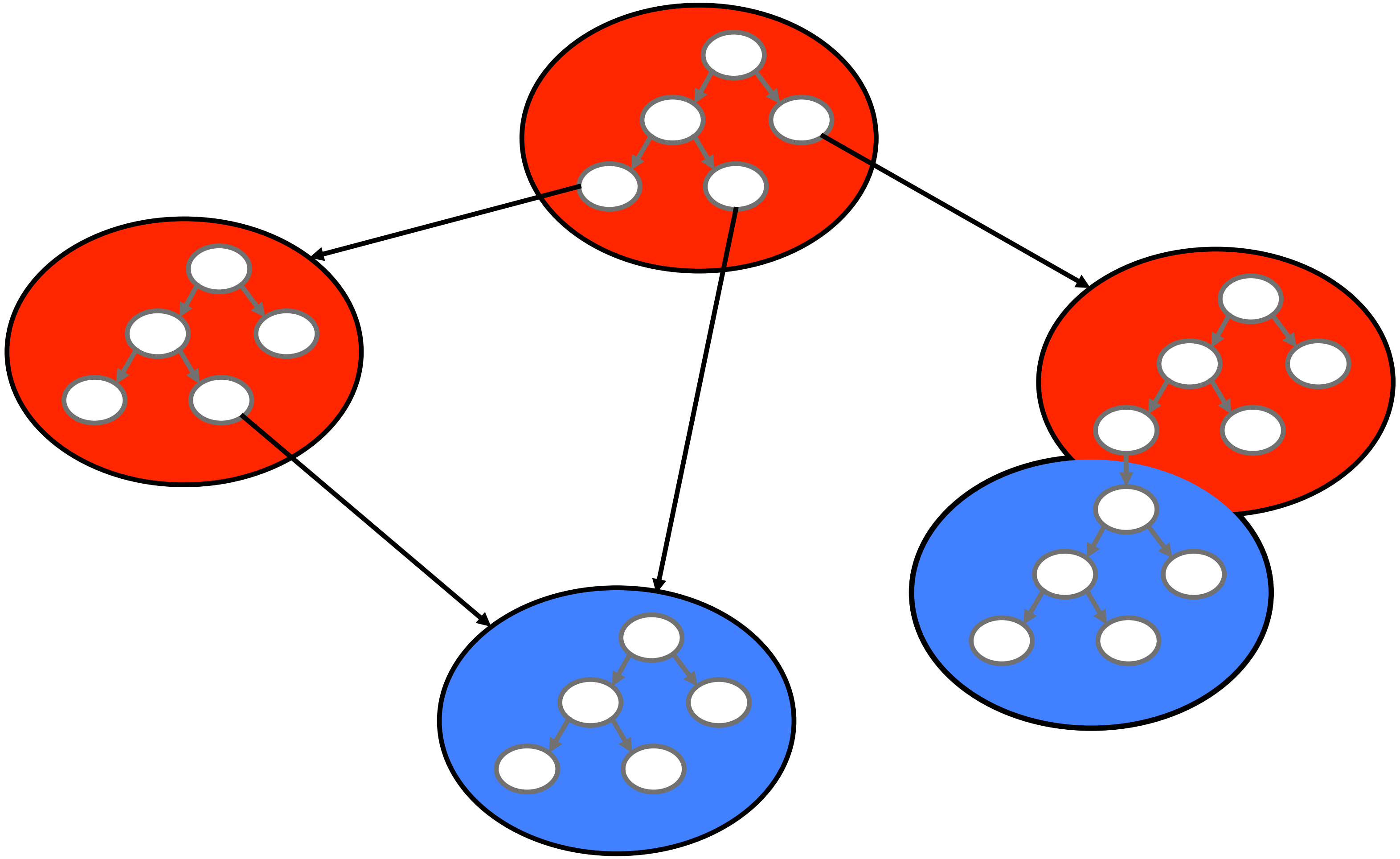
Application running without compilation or class loading

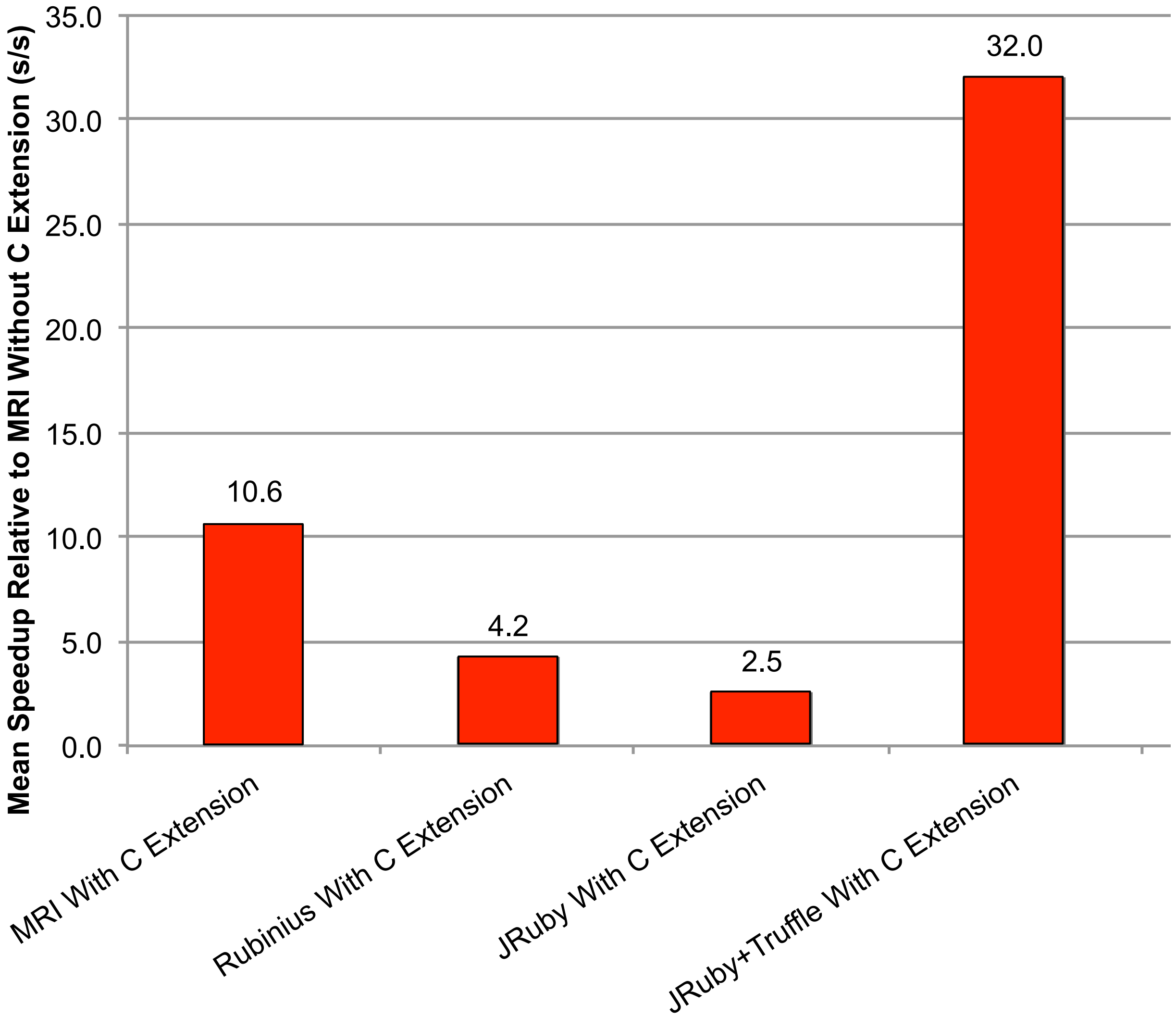

[msec]
Execution Time
800
600
400
200
0
13
353
688
14
MRI
JRuby
Truffle on JVM
Truffle on SVM


[MByte]
Memory Footprint
60
50
40
30
20
10
0
5
35
53
9
MRI
JRuby
Truffle on JVM
Truffle on SVM

How do we implement C extensions?





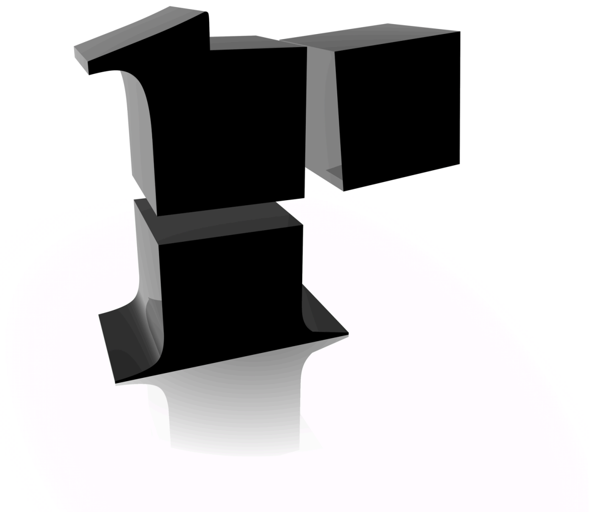


Mean Speedup Relative to MRI Without C Extension (s/s)
35.0
30.0
25.0
20.0
15.0
10.0
5.0
0.0
10.6
4.2
2.5
32.0
MRI With C Extension
Rubinius With C Extension
JRuby With C Extension
JRuby+Truffle With C Extension

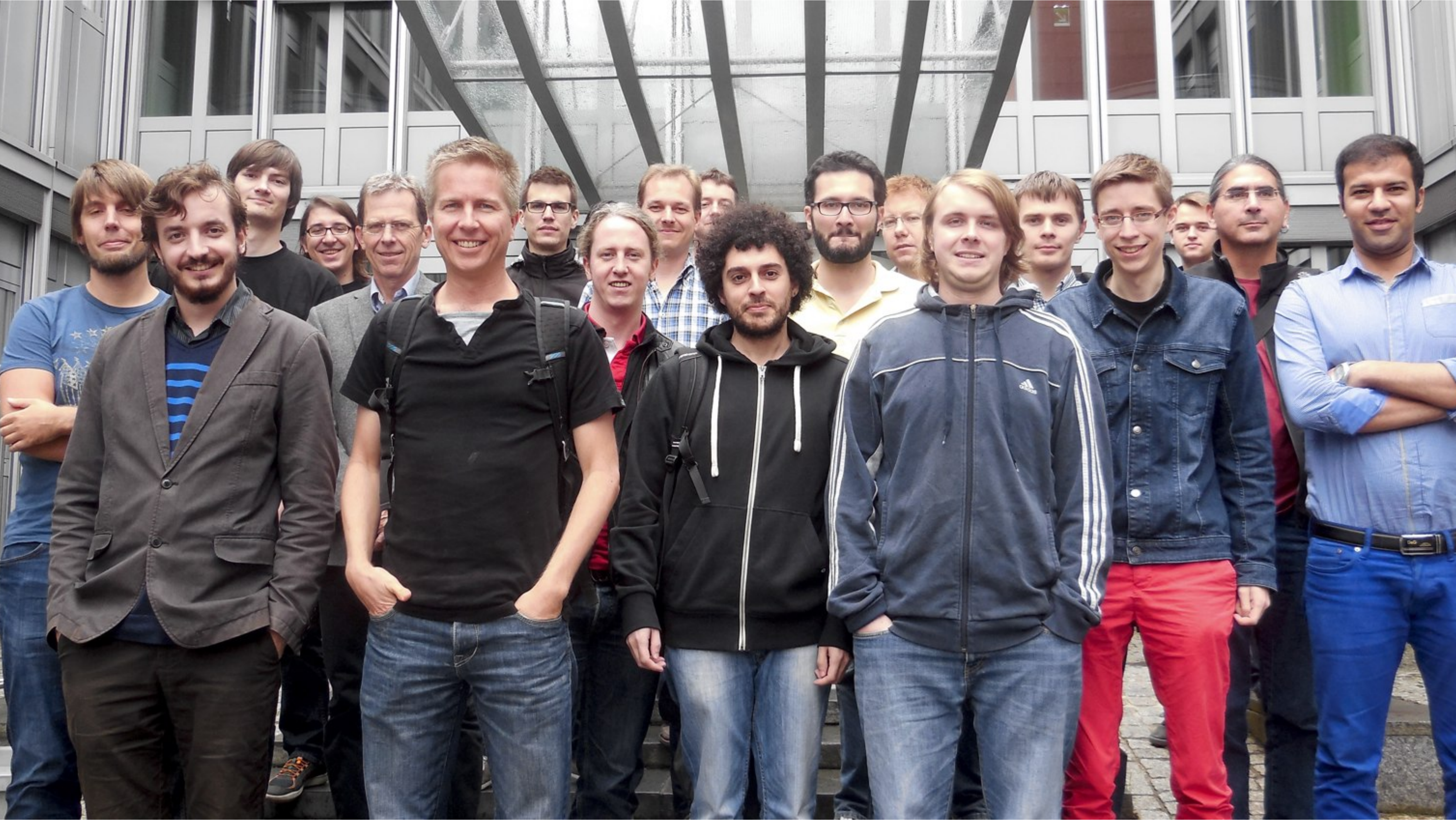
How does it build on other projects?

- Lexer, parser
- Strings, regexps, IO
- Command line
- Build and distribution infrastructure
- Cannot re-use more of the core library due to very different approaches

- The language design
- The parts of the standard library written in Ruby
- Considering trying to use some of the C code



- Parts of the core library
- Parts of the standard library
- RubySpec
- We have our own implementations of the Rubinius primitives



What other big ideas do we have?

Wrapping up

M. Grimmer, C. Seaton, T. Würthinger, H. Mössenböck. **Dynamically Composing Languages in a Modular Way: Supporting C Extensions for Dynamic Languages**. In Proceedings of the 14th International Conference on Modularity, 2015 (to appear)

A. Wöß, C. Wirth, D. Bonetta, C. Seaton, C. Humer, and H. Mössenböck. **An object storage model for the Truffle language implementation framework**. In Proceedings of the International Conference on Principles and Practices of Programming on the Java Platform (PPPJ), 2014.

C. Seaton, M. L. Van De Vanter, and M. Haupt. **Debugging at full speed**. In Proceedings of the 8th Workshop on Dynamic Languages and Applications (DYLA), 2014

T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, M. Wolczko. **One VM to Rule Them All**. In Proceedings of Onward!. 2013.

T. Würthinger, A. Wöß, L. Stadler, G. Duboscq, D. Simon, C. Wimmer. **Self-Optimizing AST Interpreters**. In Proceedings of the Dynamic Languages Symposium (DLS), 2012

#jruby

github.com/jruby/jruby

chrisseaton.com/rubytruffle

@chrisgseaton @eregontp @nirvdrum

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.

Hardware and Software

Engineered to Work Together

ORACLE®