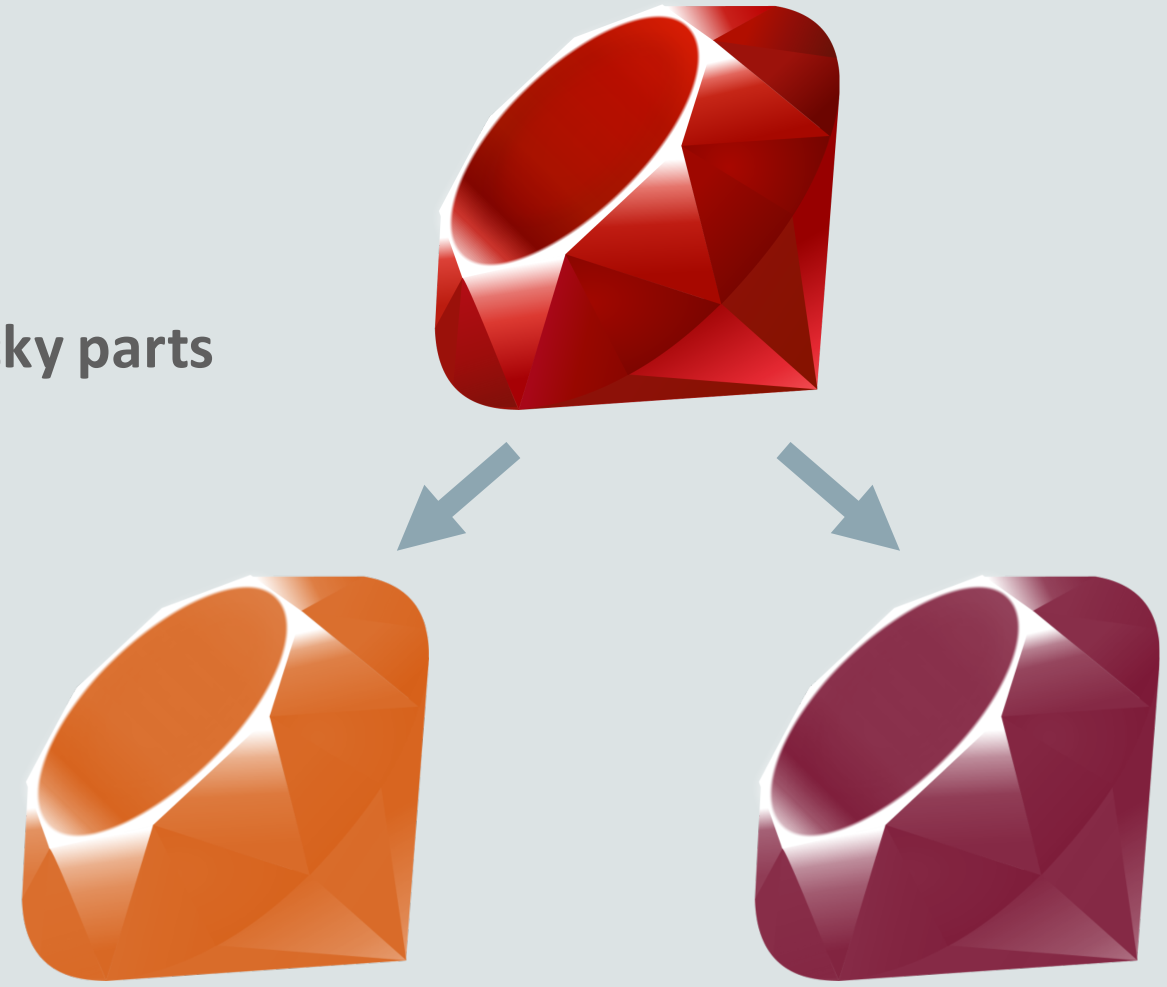
JRuby+Truffle

Why it's important to optimise the tricky parts



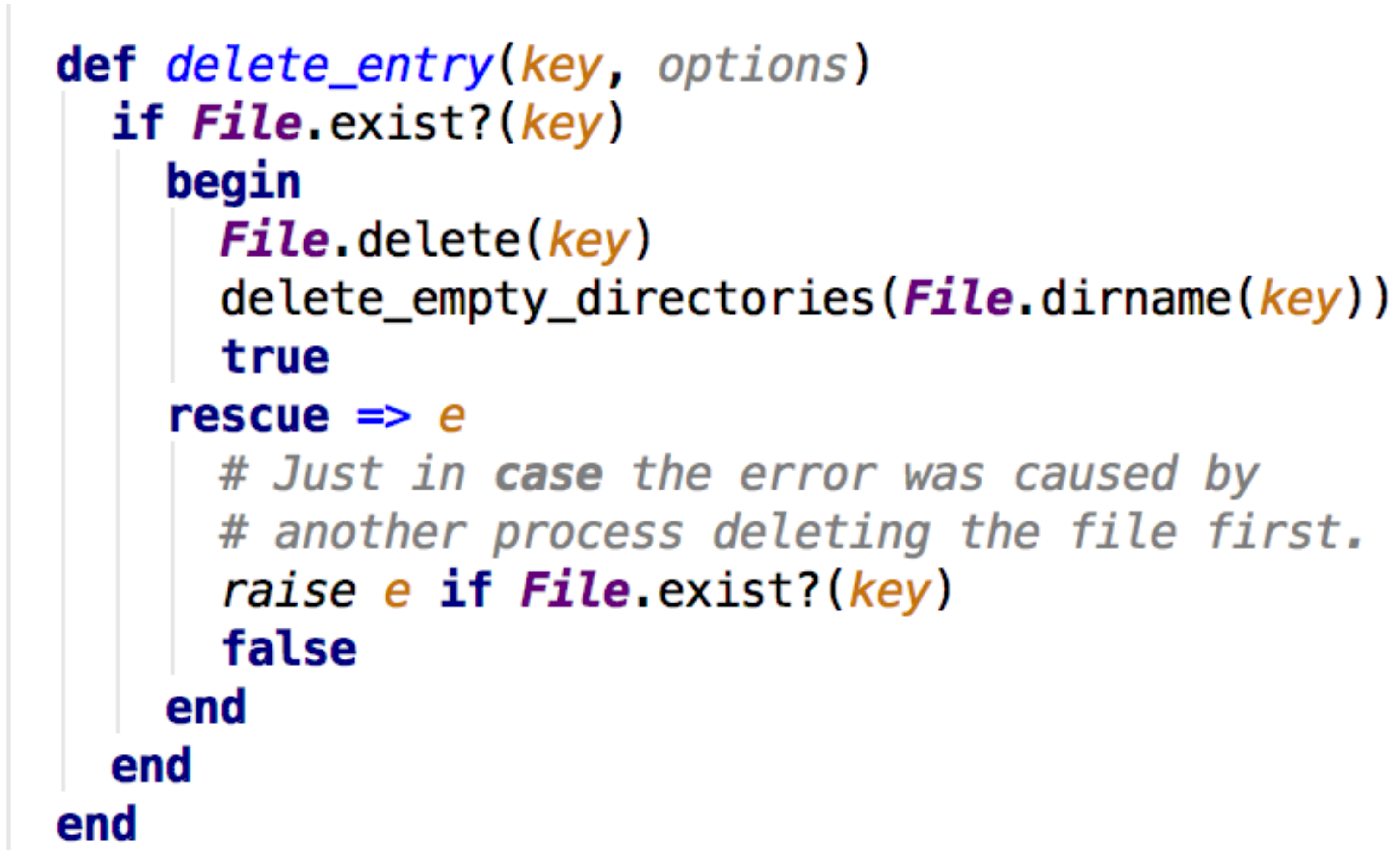
Chris Seaton
Research Manager
Oracle Labs
2 June 2016

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.

Ruby

Imperative
‘Scripting’ (Perl)
Object-oriented (Smalltalk)
Batteries included



```ruby
def delete_entry(key, options)
  if File.exist?(key)
    begin
      File.delete(key)
      delete_empty_directories(File.dirname(key))
      true
    rescue => e
      # Just in case the error was caused by
      # another process deleting the file first.
      raise e if File.exist?(key)
      false
    end
  end
end
```

MRI

Simple bytecode interpreter
Implemented in C
Core library implemented in C

ORACLE

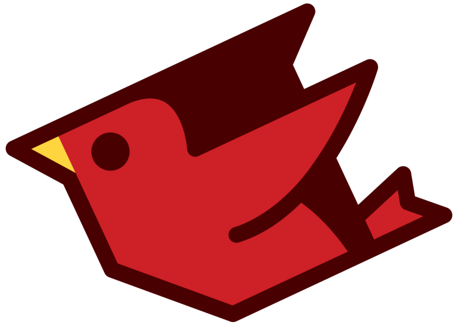
JRuby

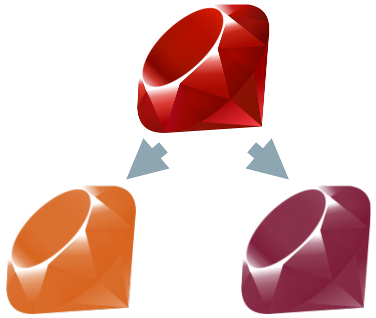
JITs by emitting JVM bytecode
VM in Java
Core library mostly in Java

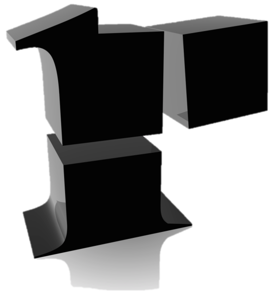
Rubinius

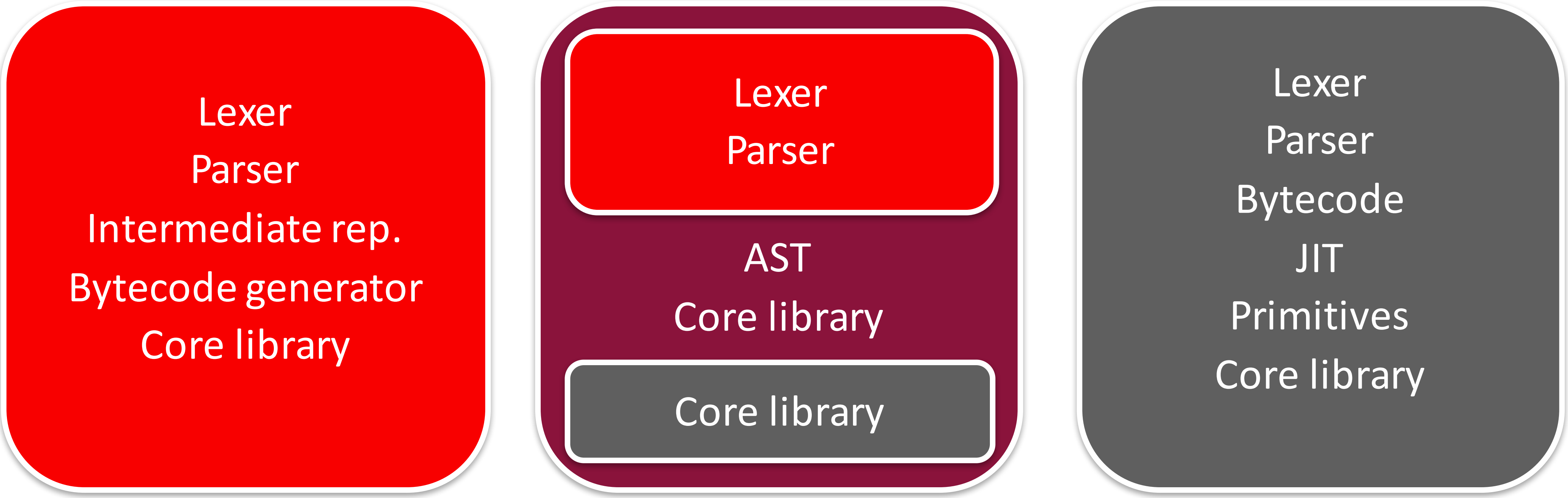
JITs by emitting LLVM IR
VM in C++
Core library mostly in Ruby

+ Truffle and Graal









Lexer
Parser
Intermediate rep.
Bytecode generator
Core library

Lexer
Parser
AST
Core library
Core library

Lexer
Parser
Bytecode
JIT
Primitives
Core library

ORACLE

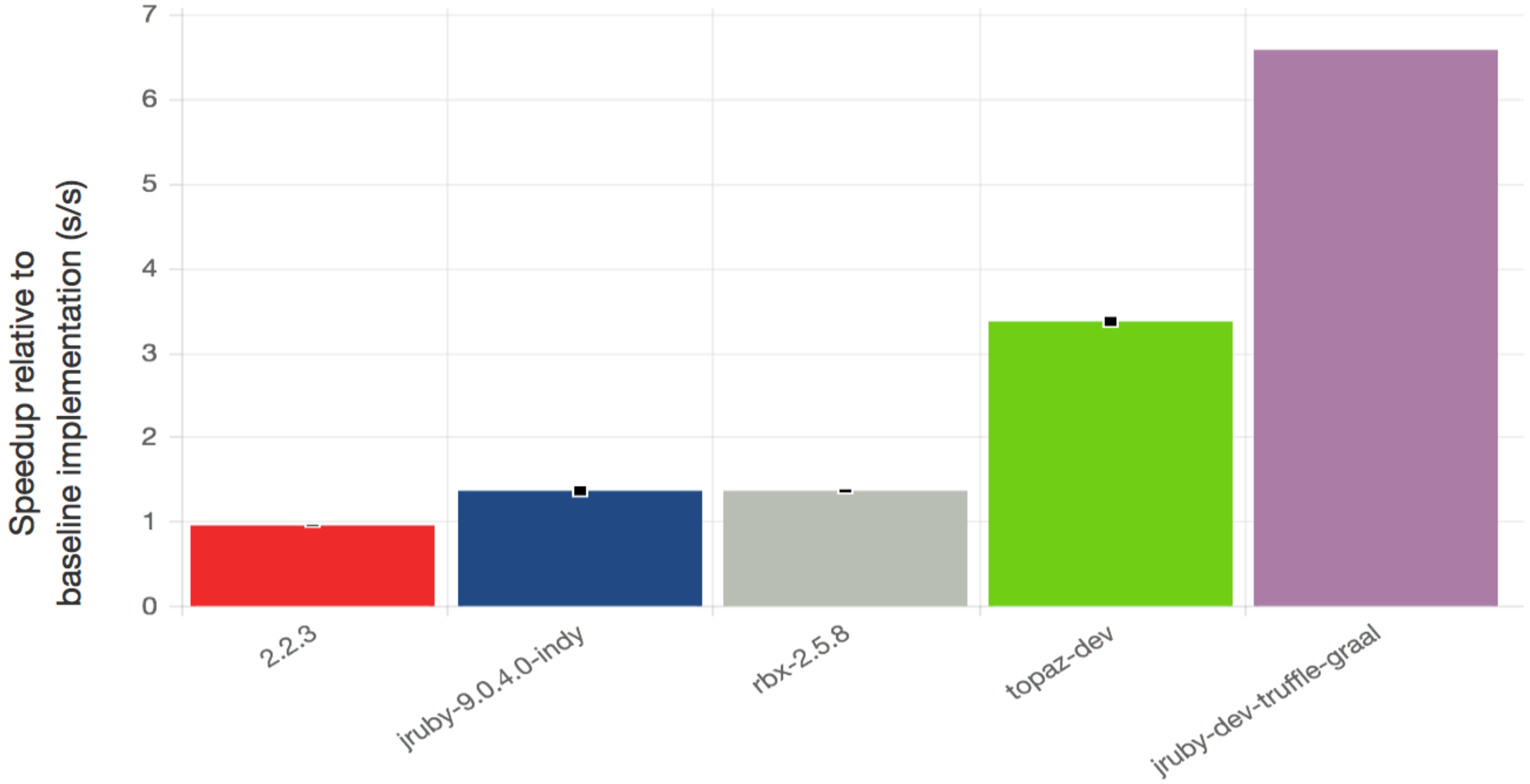
100%

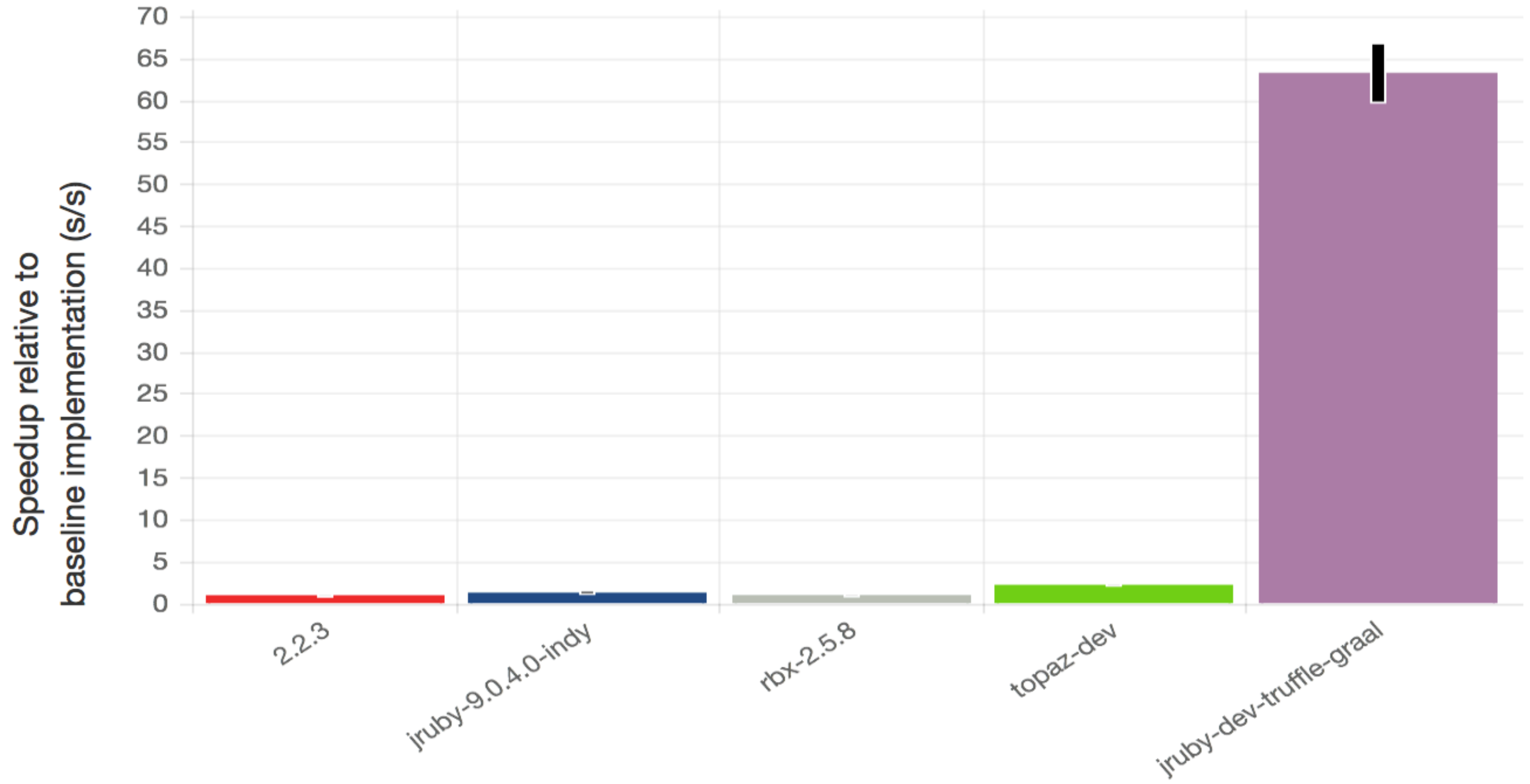
Compatibility with the language (spec/ruby)

ORACLE

90%

Compatibility with the core library (spec/ruby)


Speedup relative to baseline implementation (s/s)
0
1
2
3
4
5
6
7
2.2.3
jruby-9.0.4.0-indy
rbx-2.5.8
topaz-dev
jruby-dev-truffle-graal


Speedup relative to baseline implementation (s/s)
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
2.2.3
jruby-9.0.4.0-indy
rbx-2.5.8
topaz-dev
jruby-dev-truffle-graal

Why aren’t you using more of JRuby? Such as the existing Java implementation of the core library?

What makes Ruby difficult to optimise?

How do people want to write Ruby?

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

```ruby
def include?(value)
  if value.is_a?(::Range)
    # 1...10 includes 1..9 but it does not include 1..10.
    operator = exclude_end? && !value.exclude_end? ? :< : :<=
    super(value.first) && value.last.send(operator, last)
  else
    super
  end
end
```

```
class Object
  # An object is blank if it’s false, empty, or a whitespace string.
  # For example, ’’, ’   ’, +nil+, [], and {} are all blank.
  def blank?
    respond_to?(:empty?) ? !!empty? : !self
  end
end
```

```
def hard_mix(fg, bg, opts={})
  return apply_opacity(fg, opts)
    if fully_transparent?(bg)

  return bg if fully_transparent?(fg)

  mix_alpha, dst_alpha = calculate_alphas(
    fg, bg, DEFAULT_OPTS.merge(opts))

  new_r = blend_channel(r(bg), (r(bg)
    + r(fg) <= 255) ? 0 : 255, mix_alpha)
  new_g = blend_channel(g(bg), (g(bg)
    + g(fg) <= 255) ? 0 : 255, mix_alpha)
  new_b = blend_channel(b(bg), (b(bg)
    + b(fg) <= 255) ? 0 : 255, mix_alpha)

  rgba(new_r, new_g, new_b, dst_alpha)
end

def method_missing(method, *args, &block)
  return ChunkyPNG::Color.send(method, *args)
    if ChunkyPNG::Color.respond_to?(method)
  normal(*args)
end
```

```ruby
class Duration
  attr_accessor :value

  def initialize(value)
    @value = value
  end

  def as_json
    ...
  end

  def inspect
    ...
  end

  def method_missing(method, *args, &block)
    value.send(method, *args, &block)
  end
end
```

```
def grayscale_entry(bit_depth)
  value = ChunkyPNG::Canvas.send(
    :"decode_png_resample_#{bit_depth}bit_value",
    content.unpack('n')[0])
  ChunkyPNG::Color.grayscale(value)
end
```

```
def delegate(method)
  method_def = (
    "def #{method}(*args, &block)\n" +
    "  delegated.#{method}(*args, &block)\n" +
    "end"
  )
  module_eval(method_def, file, line)
end
```

```ruby
#
# Executes the generated ERB code to produce a completed template, returning
# the results of that code.  (See ERB::new for details on how this process
# can be affected by _safe_level_.)
#
# _b_ accepts a Binding object which is used to set the context of
# code evaluation.
#
def result(b=new_toplevel)
  if @safe_level
    proc {
      $SAFE = @safe_level
      eval(@src, b, (@filename || '(erb)'), @lineno)
    }.call
  else
    eval(@src, b, (@filename || '(erb)'), @lineno)
  end
end
```

Why can’t a conventional VM optimise this?

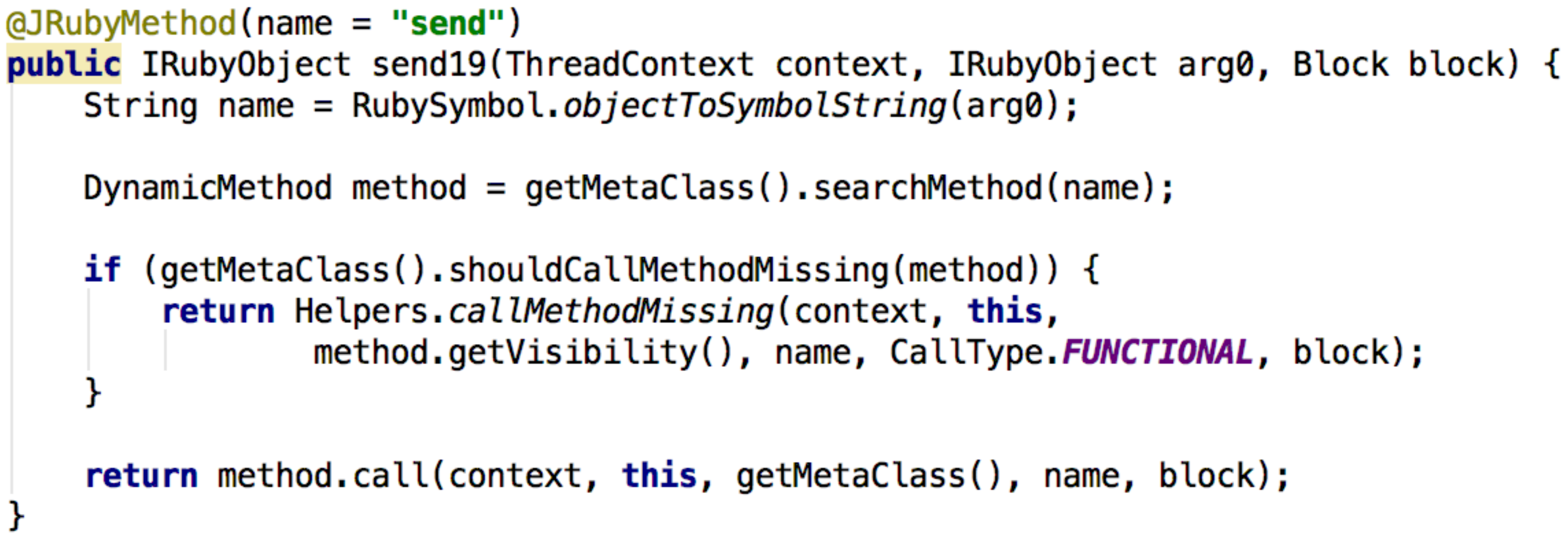
Why can’t JRuby make this as fast as we want?

First problem: JRuby’s core library is **megamorphic**

```
@JRubyMethod(name = "+")
public IRubyObject op_plus(ThreadContext context, IRubyObject other) {
    if (other instanceof RubyFixnum) {
        return addFixnum(context, (RubyFixnum) other);
    }
    if (other instanceof RubyBignum) {
        return ((RubyBignum) other).op_plus(context, this);
    }
    if (other instanceof RubyFloat) {
        return context.runtime.newFloat(
                (double) value + ((RubyFloat) other).getDoubleValue());
    }
    return coerceBin(context, "+", other);
}
```

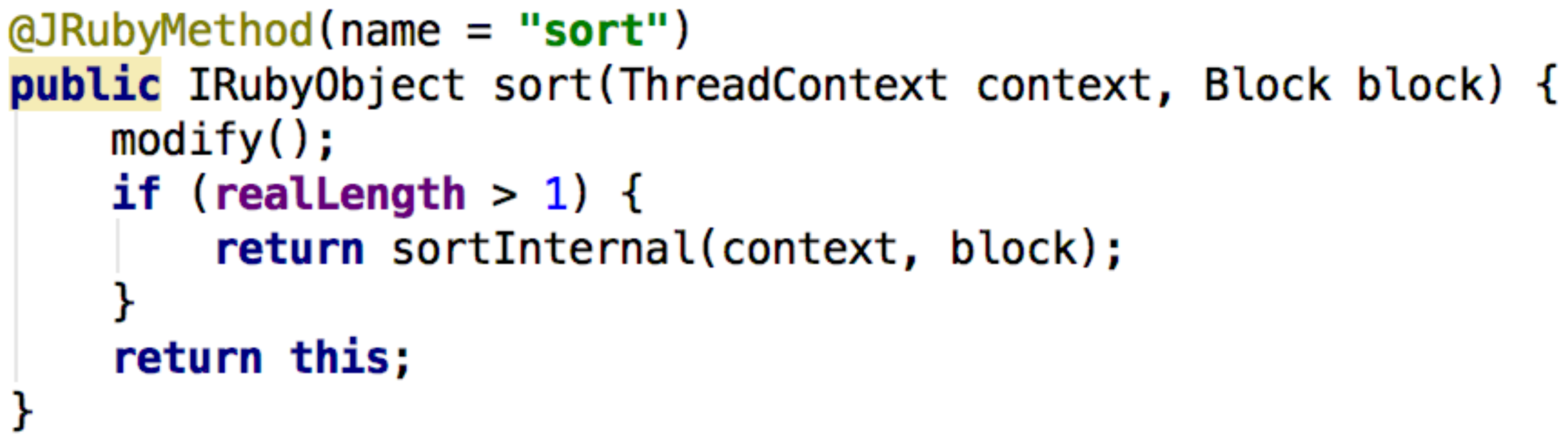
ORACLE

Second problem: JRuby’s core library is **stateless**

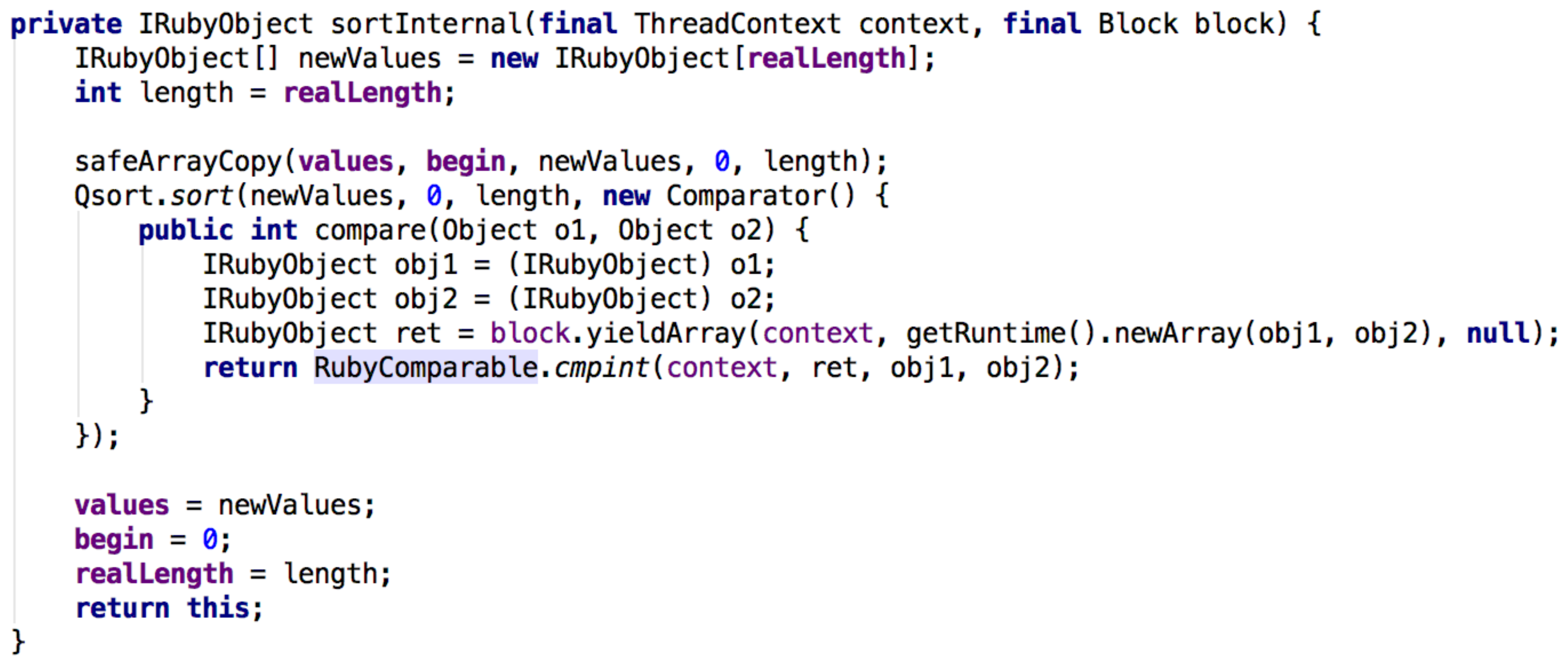


```java
@JRubyMethod(name = "send")
public IRubyObject send19(ThreadContext context, IRubyObject arg0, Block block) {
    String name = RubySymbol.objectToSymbolString(arg0);

    DynamicMethod method = getMetaClass().searchMethod(name);

    if (getMetaClass().shouldCallMethodMissing(method)) {
        return Helpers.callMethodMissing(context, this,
                method.getVisibility(), name, CallType.FUNCTIONAL, block);
    }

    return method.call(context, this, getMetaClass(), name, block);
}
```

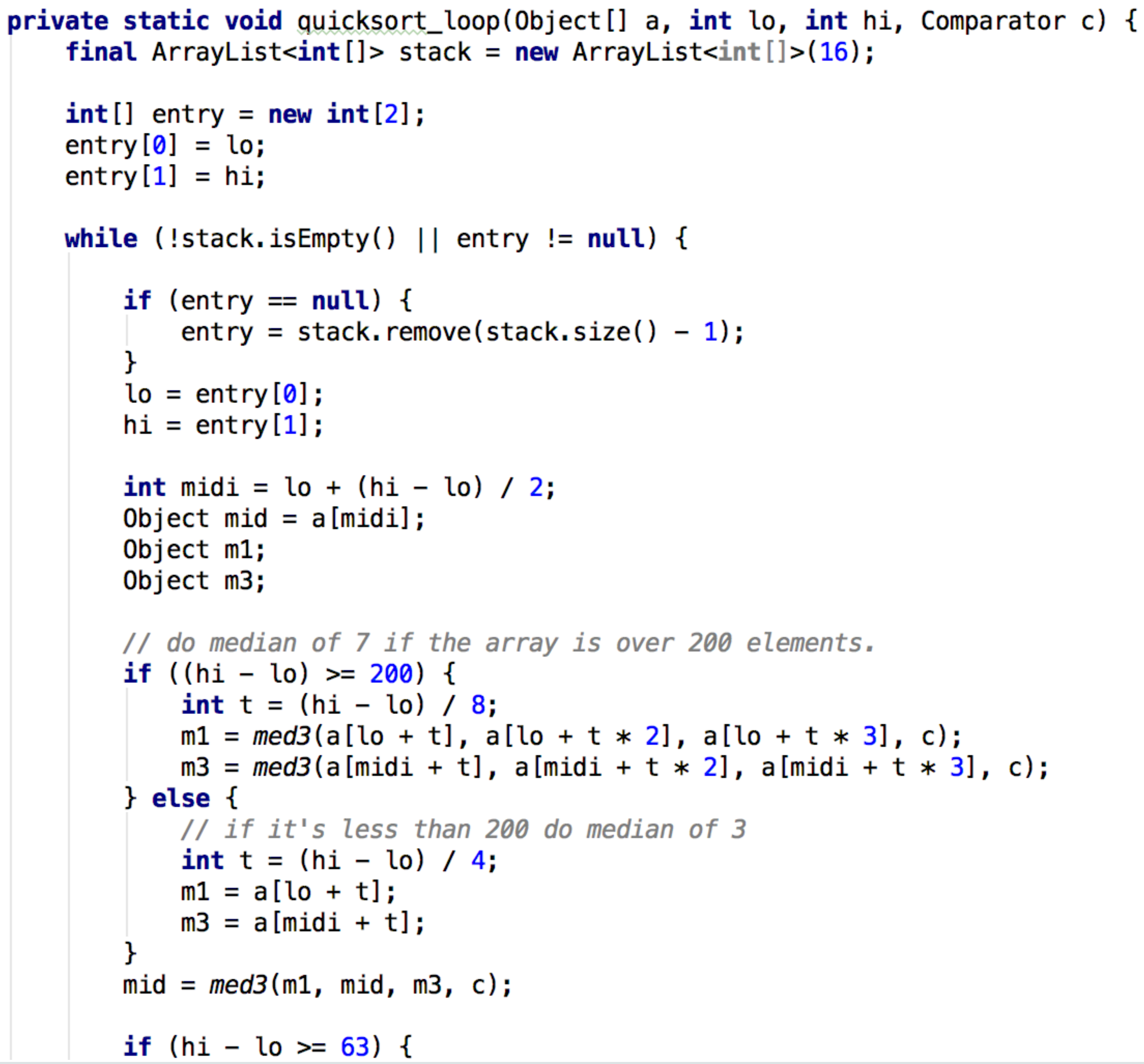
Third problem: JRuby’s core library is **very deep**



```java
@JRubyMethod(name = "sort")
public IRubyObject sort(ThreadContext context, Block block) {
    modify();
    if (realLength > 1) {
        return sortInternal(context, block);
    }
    return this;
}
```

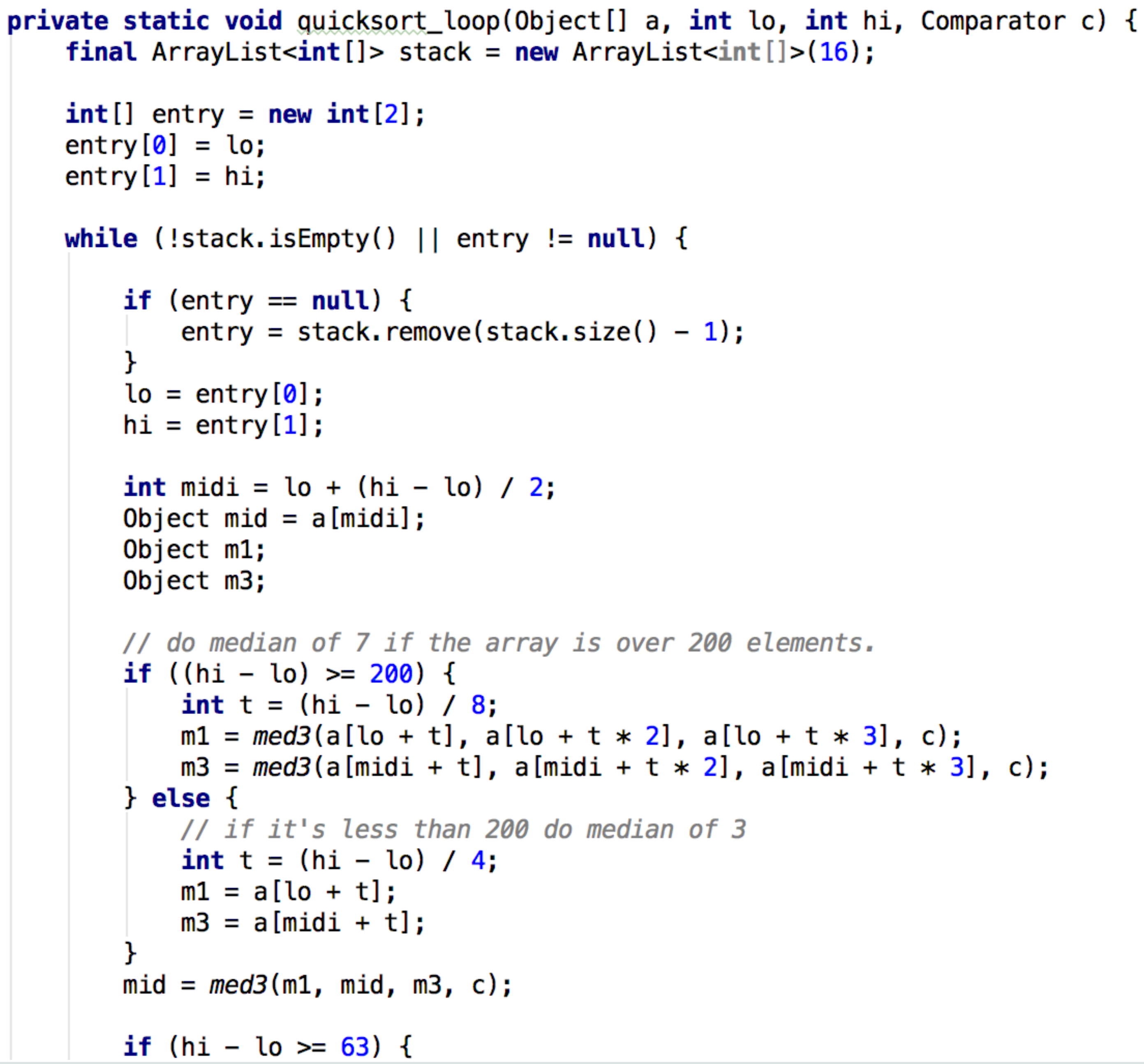


```java
private IRubyObject sortInternal(final ThreadContext context, final Block block) {
    IRubyObject[] newValues = new IRubyObject[realLength];
    int length = realLength;

    safeArrayCopy(values, begin, newValues, 0, length);
    Qsort.sort(newValues, 0, length, new Comparator() {
        public int compare(Object o1, Object o2) {
            IRubyObject obj1 = (IRubyObject) o1;
            IRubyObject obj2 = (IRubyObject) o2;
            IRubyObject ret = block.yieldArray(context, getRuntime().newArray(obj1, obj2), null);
            return RubyComparable.cmpint(context, ret, obj1, obj2);
        }
    });

    values = newValues;
    begin = 0;
    realLength = length;
    return this;
}
```



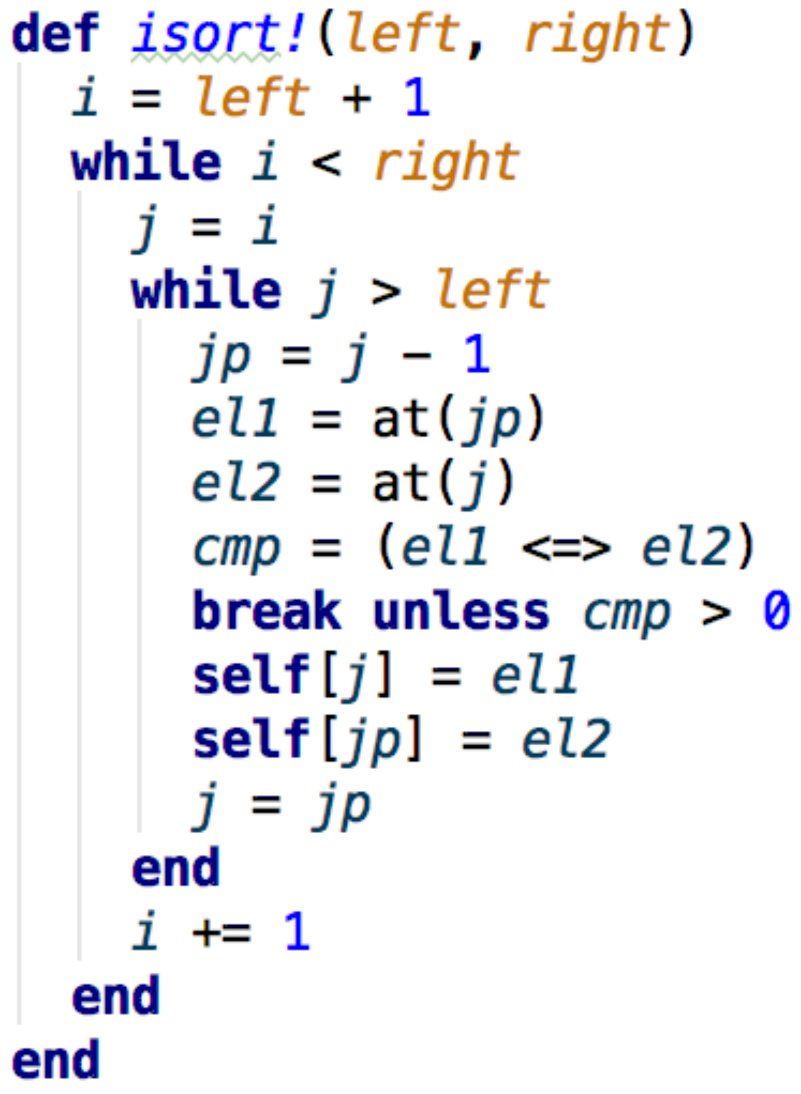
```java
private static void quicksort_loop(Object[] a, int lo, int hi, Comparator c) {
    final ArrayList<int[]> stack = new ArrayList<int[]>(16);

    int[] entry = new int[2];
    entry[0] = lo;
    entry[1] = hi;

    while (!stack.isEmpty() || entry != null) {

        if (entry == null) {
            entry = stack.remove(stack.size() - 1);
        }
        lo = entry[0];
        hi = entry[1];

        int midi = lo + (hi - lo) / 2;
        Object mid = a[midi];
        Object m1;
        Object m3;

        // do median of 7 if the array is over 200 elements.
        if ((hi - lo) >= 200) {
            int t = (hi - lo) / 8;
            m1 = med3(a[lo + t], a[lo + t * 2], a[lo + t * 3], c);
            m3 = med3(a[midi + t], a[midi + t * 2], a[midi + t * 3], c);
        } else {
            // if it's less than 200 do median of 3
            int t = (hi - lo) / 4;
            m1 = a[lo + t];
            m3 = a[midi + t];
        }
        mid = med3(m1, mid, m3, c);

        if (hi - lo >= 63) {
```

Fourth problem: JRuby’s core library **isn’t amenable to optimisations**

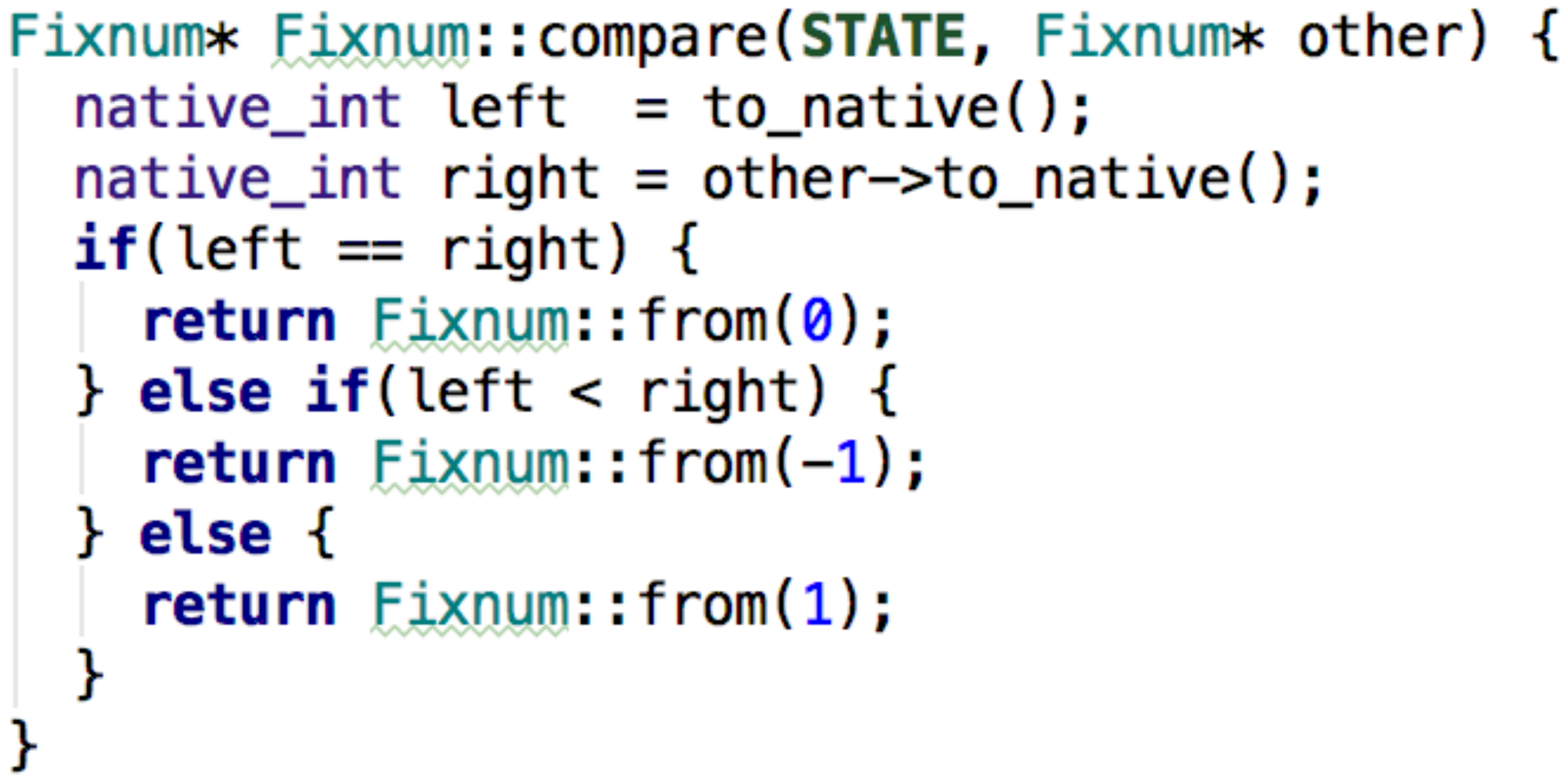


```java
private static void quicksort_loop(Object[] a, int lo, int hi, Comparator c) {
    final ArrayList<int[]> stack = new ArrayList<int[]>(16);

    int[] entry = new int[2];
    entry[0] = lo;
    entry[1] = hi;

    while (!stack.isEmpty() || entry != null) {

        if (entry == null) {
            entry = stack.remove(stack.size() - 1);
        }
        lo = entry[0];
        hi = entry[1];

        int midi = lo + (hi - lo) / 2;
        Object mid = a[midi];
        Object m1;
        Object m3;

        // do median of 7 if the array is over 200 elements.
        if ((hi - lo) >= 200) {
            int t = (hi - lo) / 8;
            m1 = med3(a[lo + t], a[lo + t * 2], a[lo + t * 3], c);
            m3 = med3(a[midi + t], a[midi + t * 2], a[midi + t * 3], c);
        } else {
            // if it's less than 200 do median of 3
            int t = (hi - lo) / 4;
            m1 = a[lo + t];
            m3 = a[midi + t];
        }
        mid = med3(m1, mid, m3, c);

        if (hi - lo >= 63) {
```

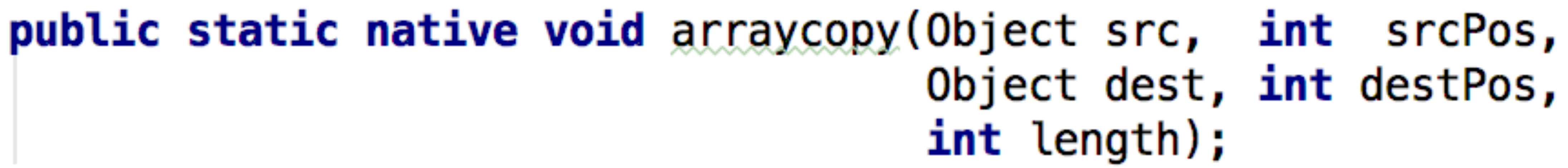
The same problems apply to Rubinius, even though the core library is mostly written in Ruby



```ruby
def isort!(left, right)
  i = left + 1
  while i < right
    j = i
    while j > left
      jp = j - 1
      el1 = at(jp)
      el2 = at(j)
      cmp = (el1 <=> el2)
      break unless cmp > 0
      self[j] = el1
      self[jp] = el2
      j = jp
    end
    i += 1
  end
end
```



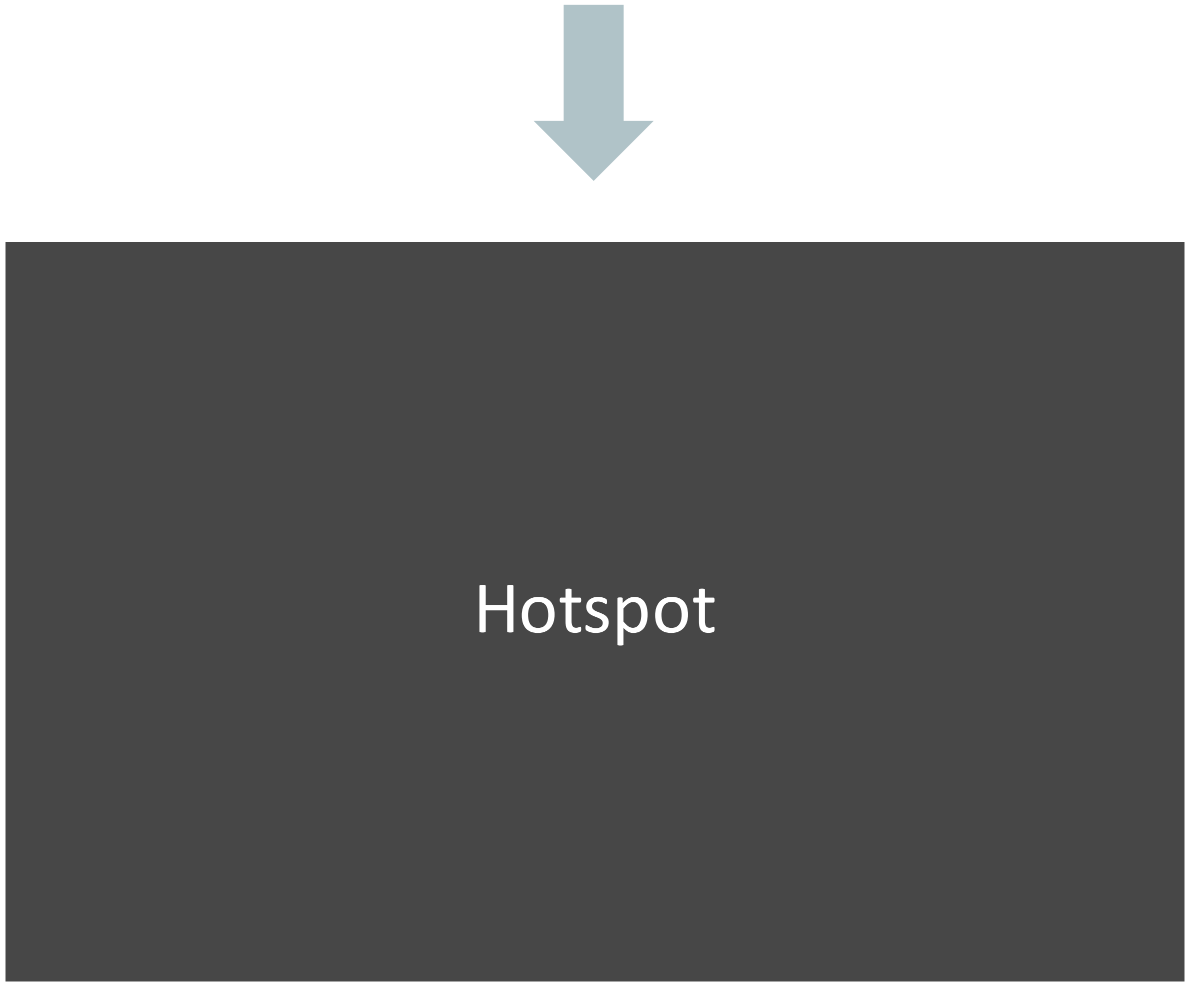
```
Fixnum* Fixnum::compare(STATE, Fixnum* other) {
  native_int left  = to_native();
  native_int right = other->to_native();
  if(left == right) {
    return Fixnum::from(0);
  } else if(left < right) {
    return Fixnum::from(-1);
  } else {
    return Fixnum::from(1);
  }
}
```

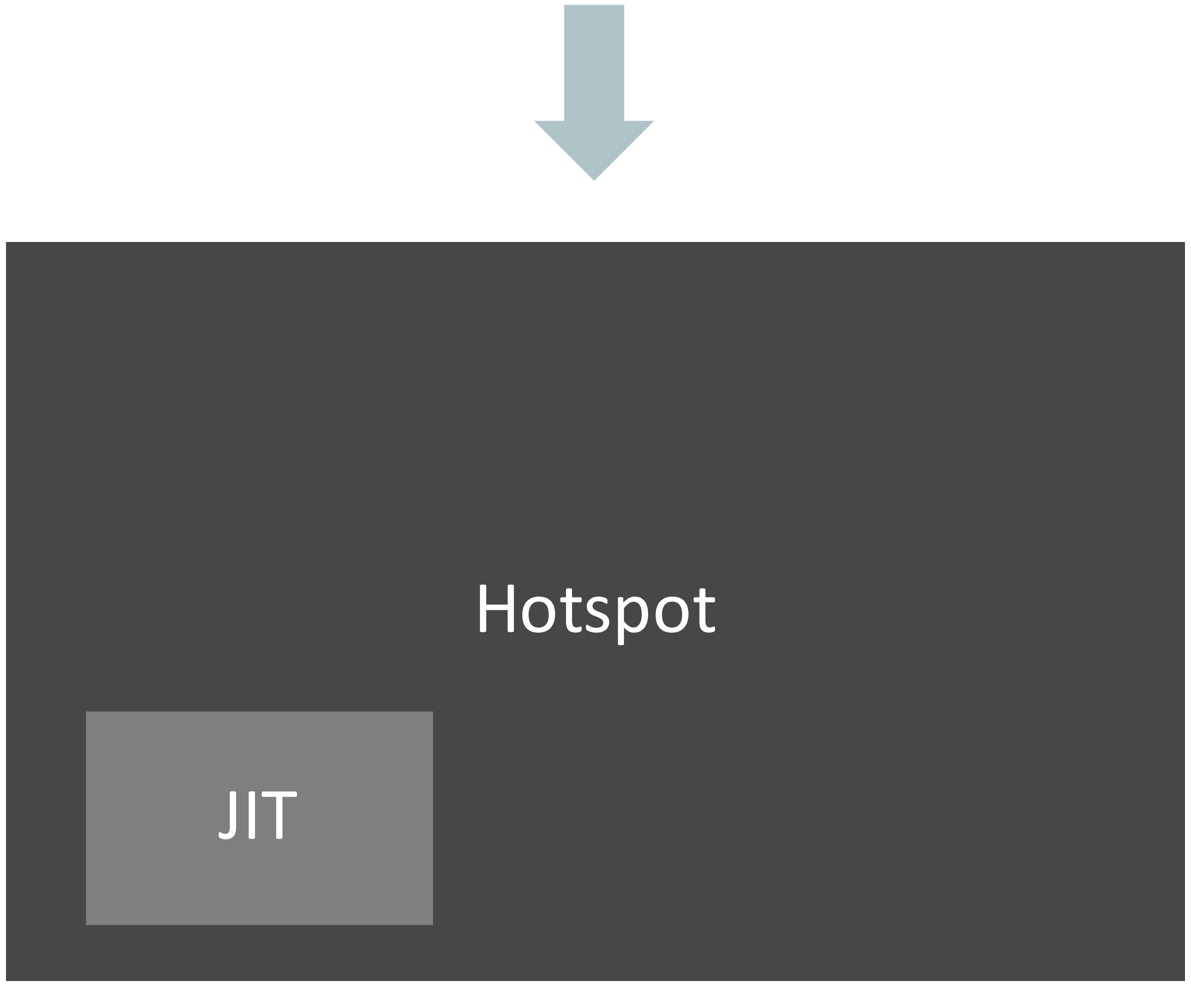


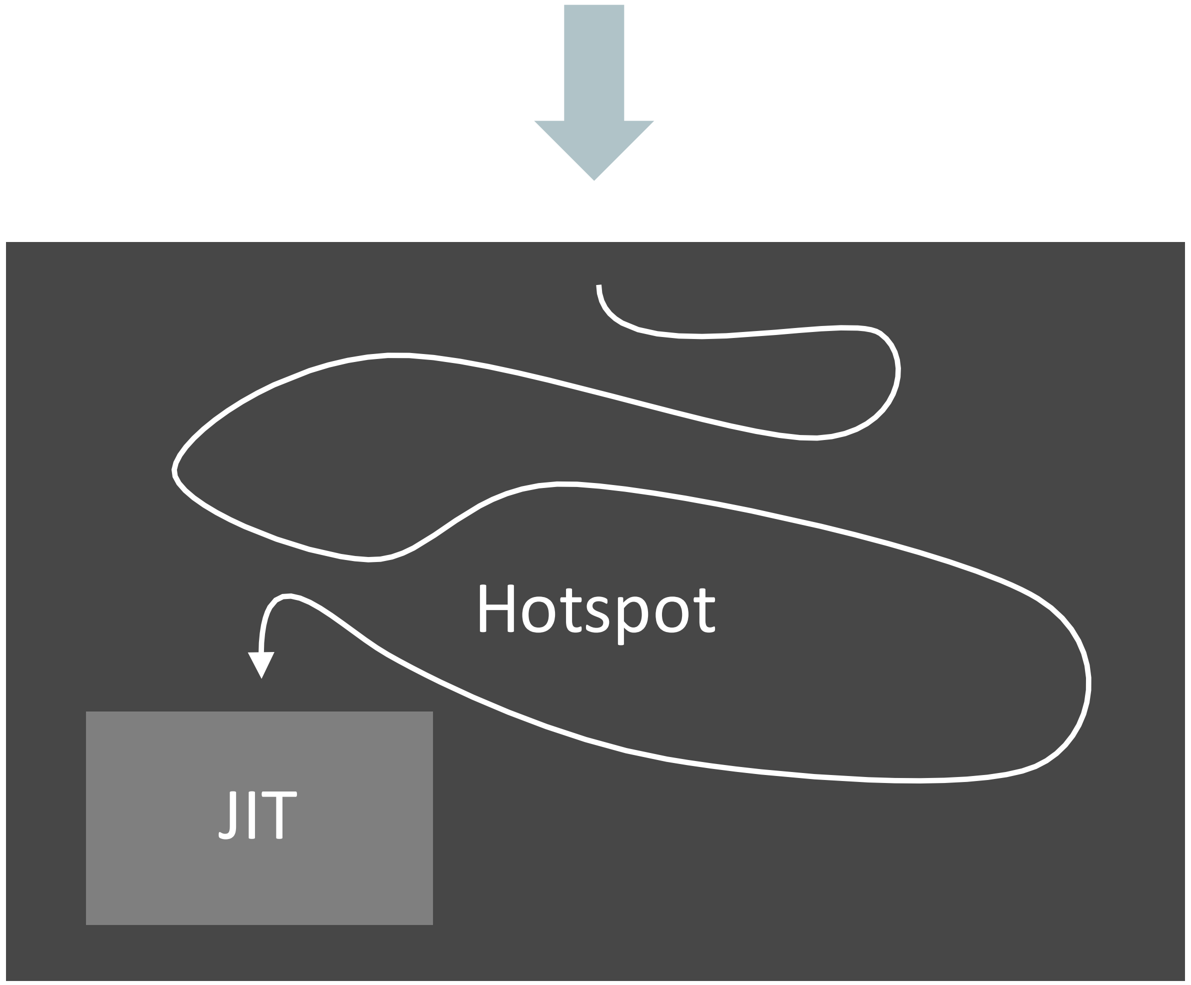
```java
public static native void arraycopy(Object src,  int  srcPos,
                                    Object dest, int destPos,
                                    int length);
```

Interlude: Truffle and Graal

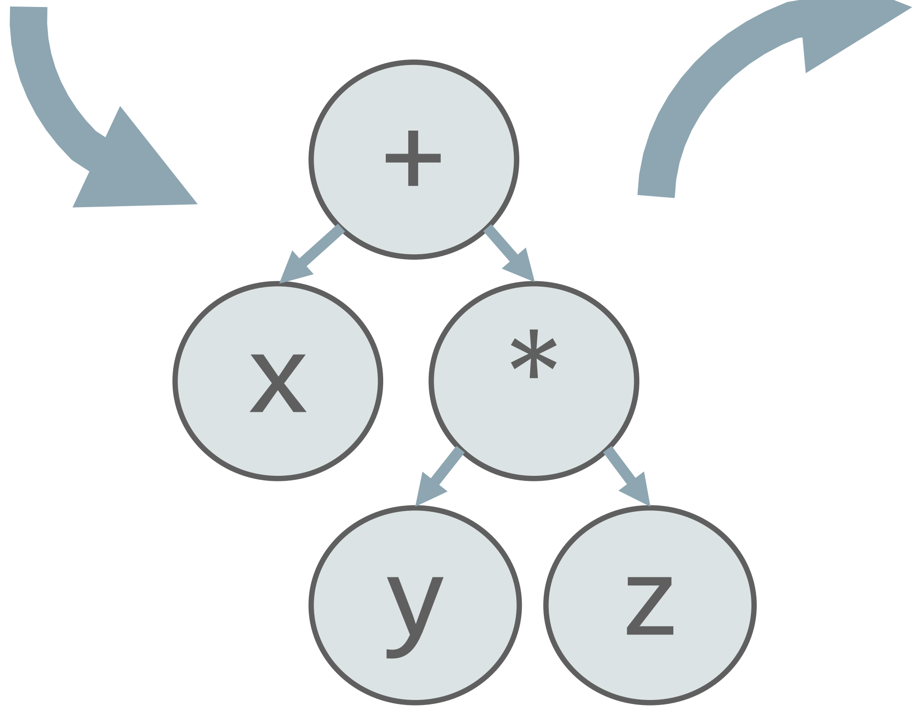
Hotspot


Hotspot


Hotspot
JIT


Hotspot
JIT

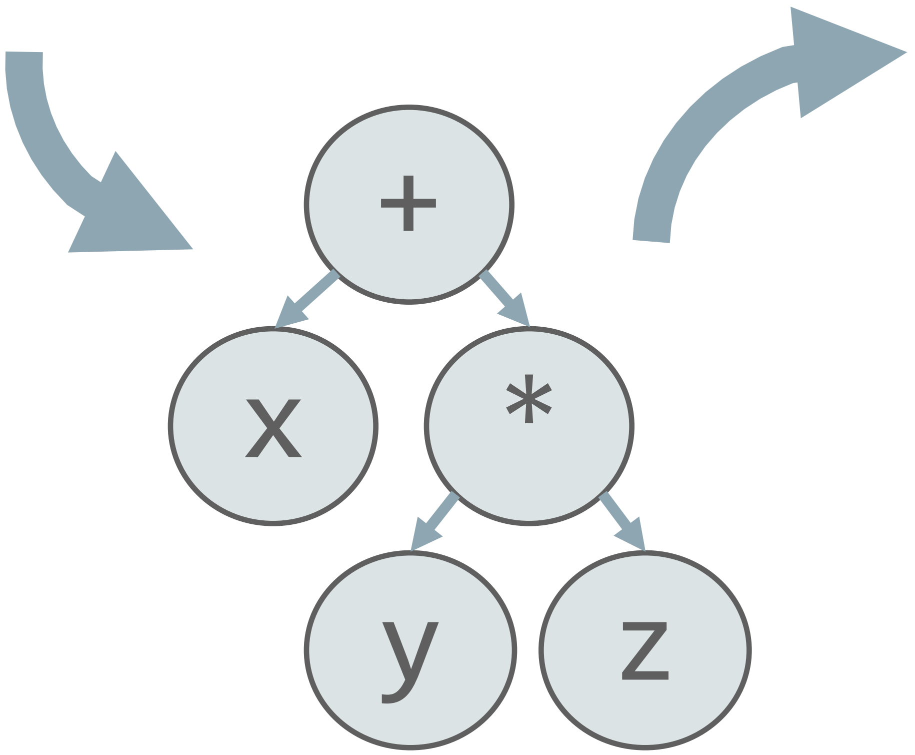
```
x + y * z
```



```
load_local x
load_local y
load_local z
call :*
call :+
```

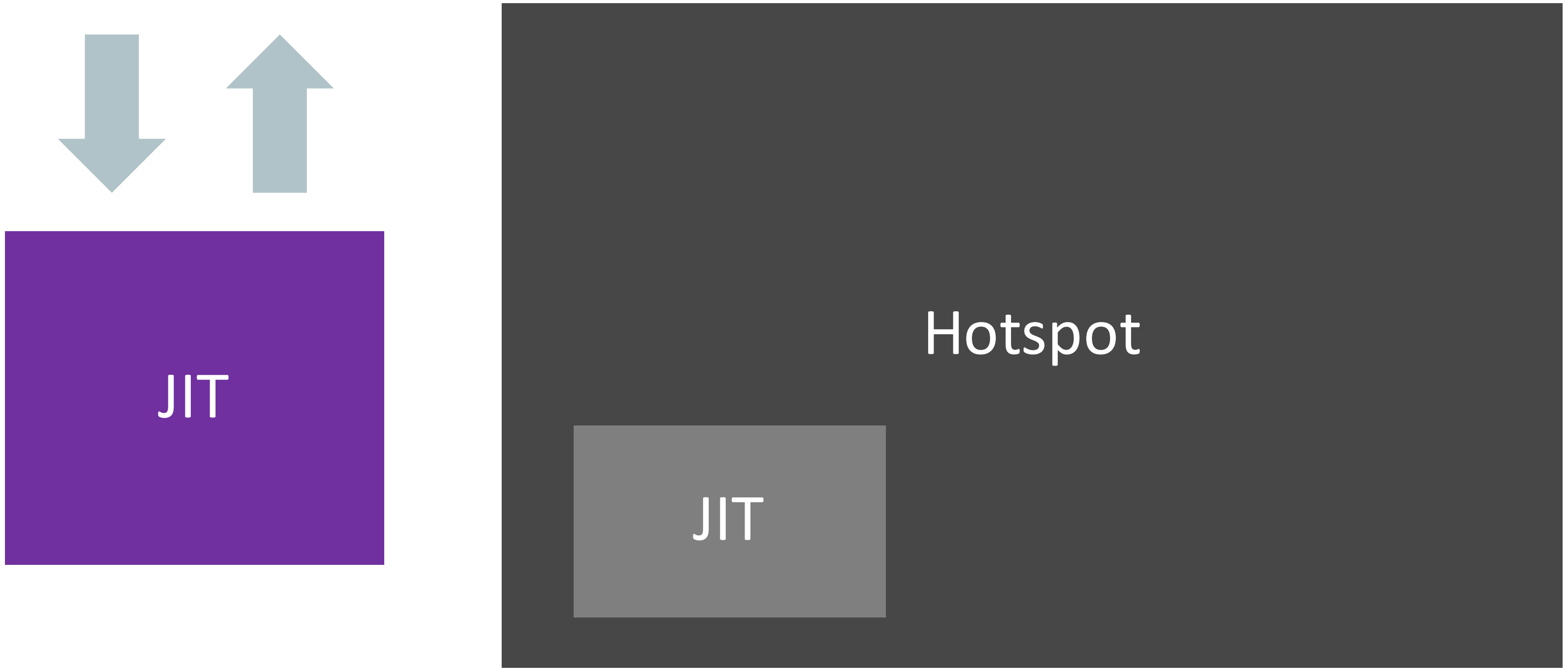
```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rdx, -24(%rbp)
movq  -16(%rbp), %rax
movl  %eax, %edx
movq  -24(%rbp), %rax
imull %edx, %eax
movq  -8(%rbp), %rdx
addl  %edx, %eax
popq  %rbp
ret
```

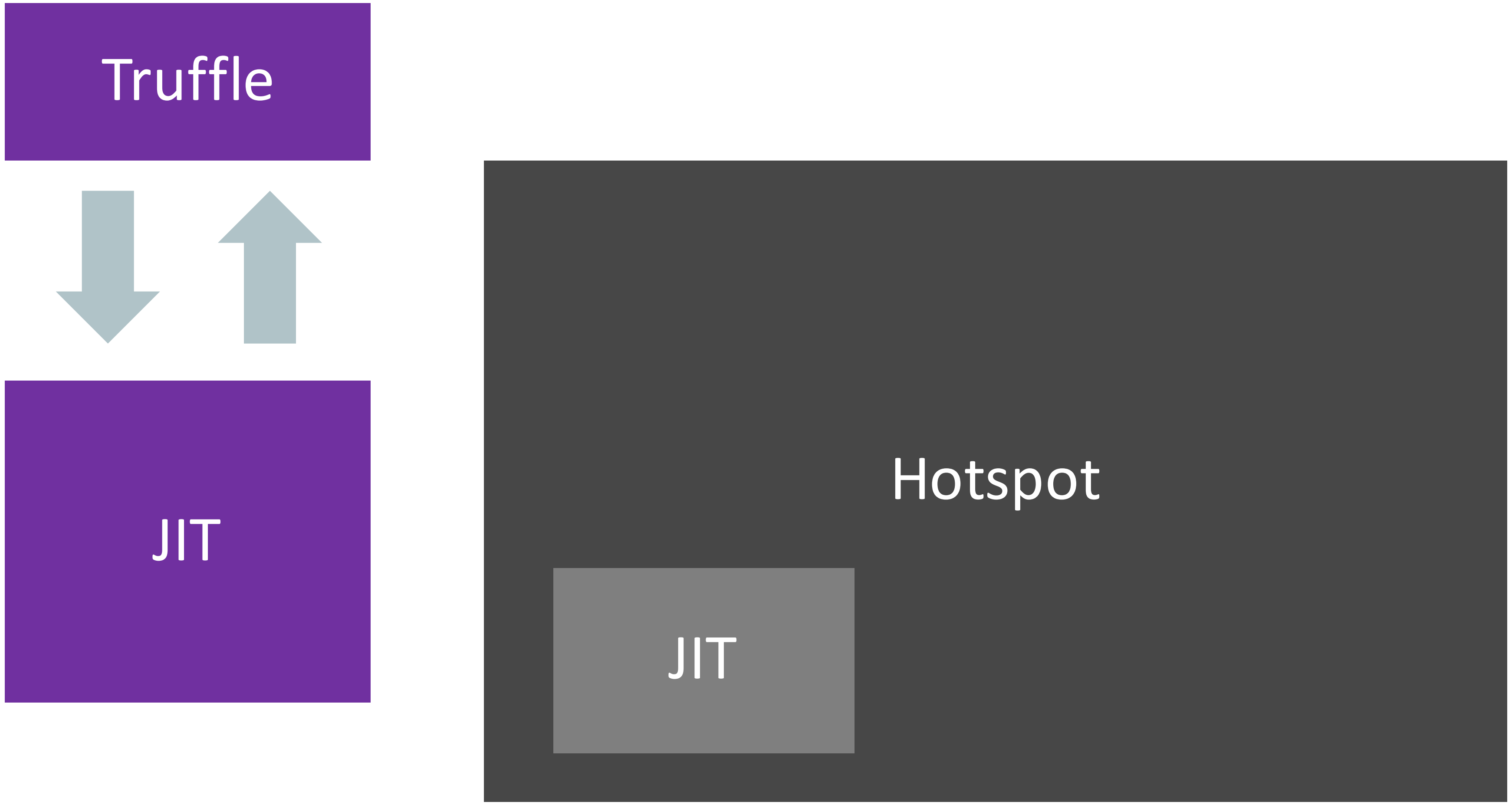
```
x + y * z
```

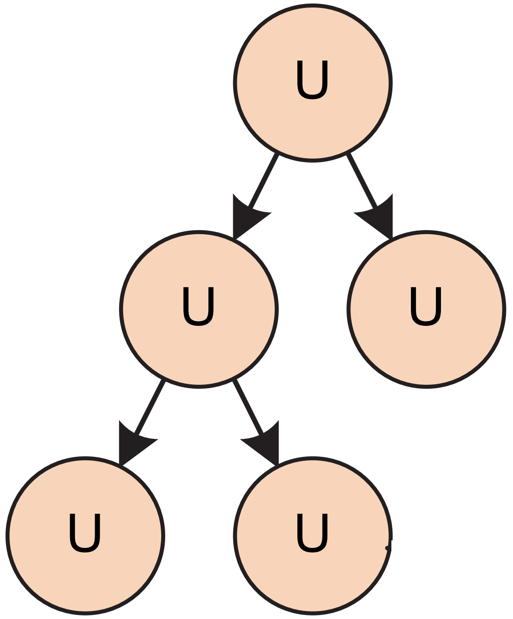


```
load_local x
load_local y
load_local z
call :*
call :+
```

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rdx, -24(%rbp)
movq  -16(%rbp), %rax
movl  %eax, %edx
movq  -24(%rbp), %rax
imull %edx, %eax
movq  -8(%rbp), %rdx
addl  %edx, %eax
popq  %rbp
ret
```


JIT
Hotspot
JIT

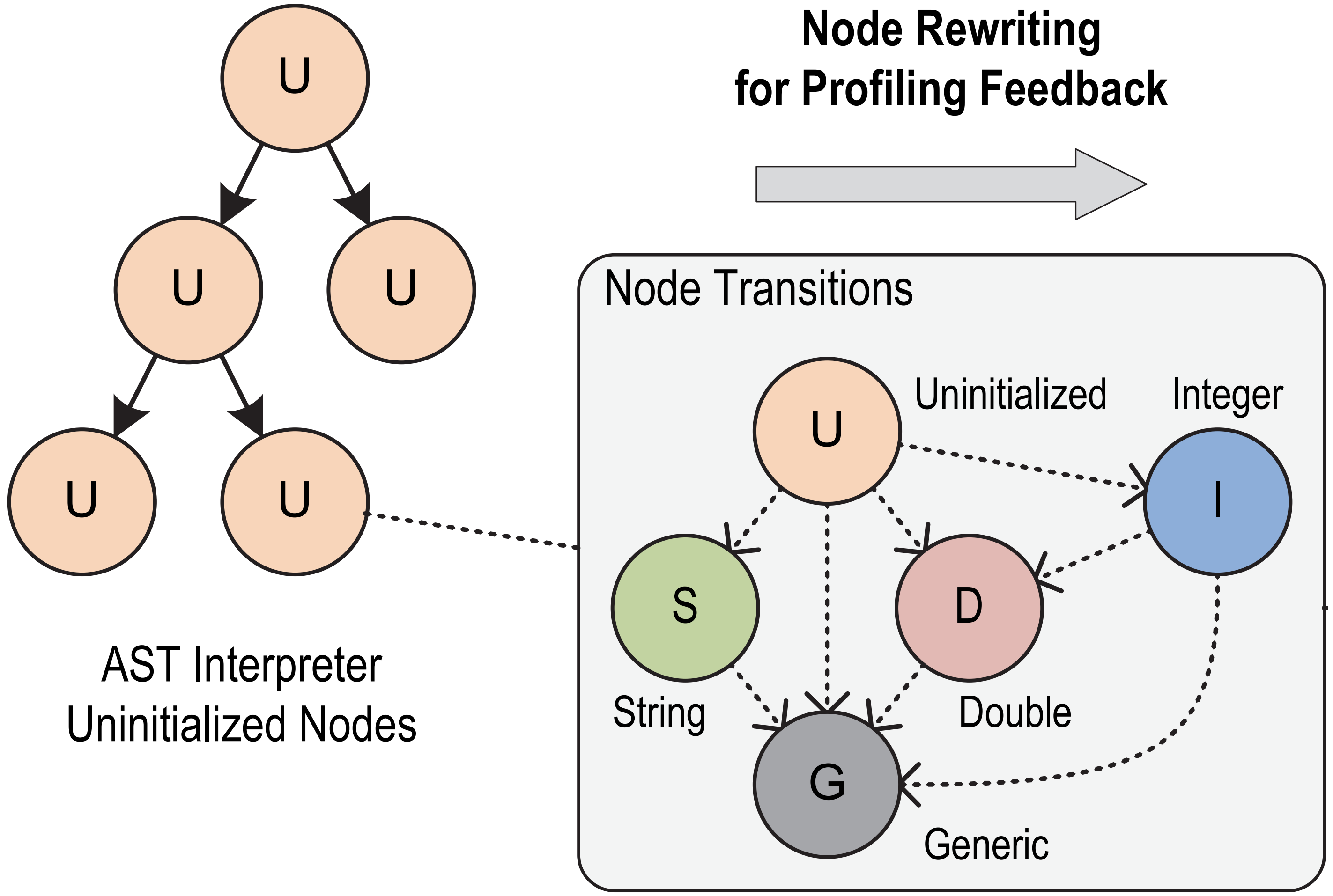

Truffle
JIT
Hotspot
JIT



AST Interpreter
Uninitialized Nodes

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.

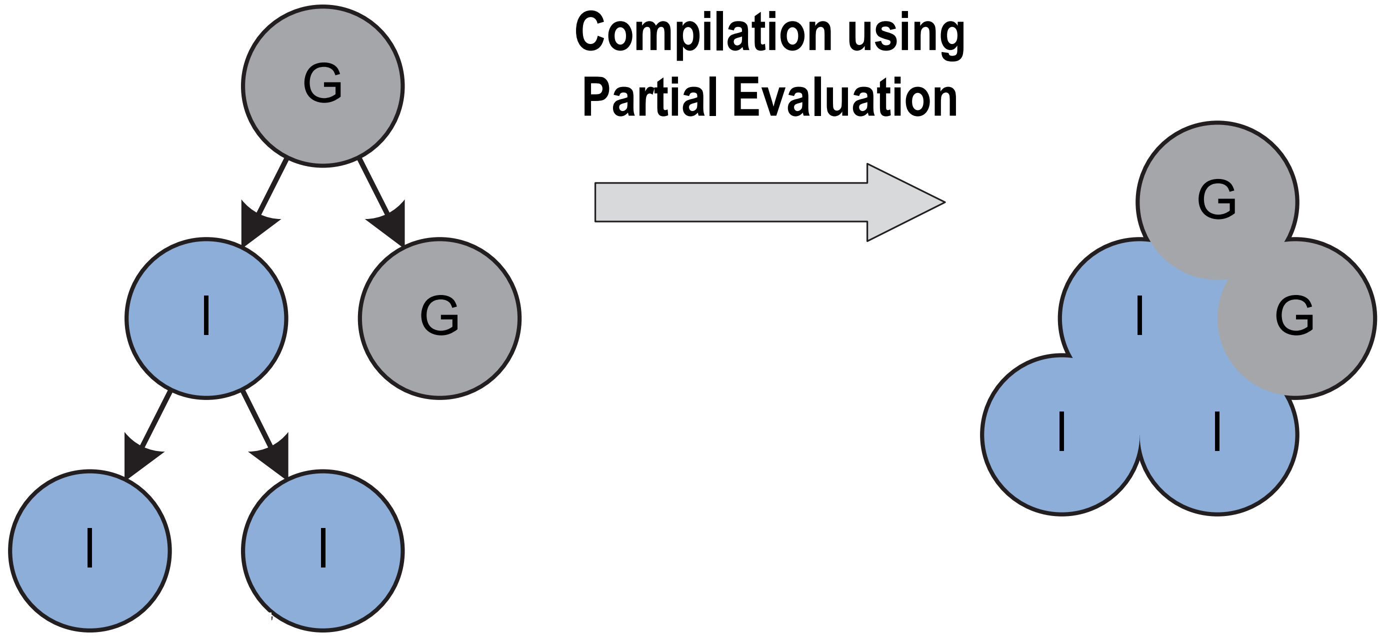
Node Rewriting for Profiling Feedback



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.

```java
@NodeInfo(shortName = "+")
public abstract class SLAddNode extends SLBinaryNode {

    public SLAddNode(SourceSection src) {
        super(src);
    }

    @Specialization(rewriteOn = ArithmeticException.class)
    protected long add(long left, long right) {
        return ExactMath.addExact(left, right);
    }

    @Specialization
    @TruffleBoundary
    protected BigInteger add(BigInteger left, BigInteger right) {
        return left.add(right);
    }

    @Specialization(guards = "isString(left, right)")
    @TruffleBoundary
    protected String add(Object left, Object right) {
        return left.toString() + right.toString();
    }

    protected boolean isString(Object a, Object b) {
        return a instanceof String || b instanceof String;
    }
}
```

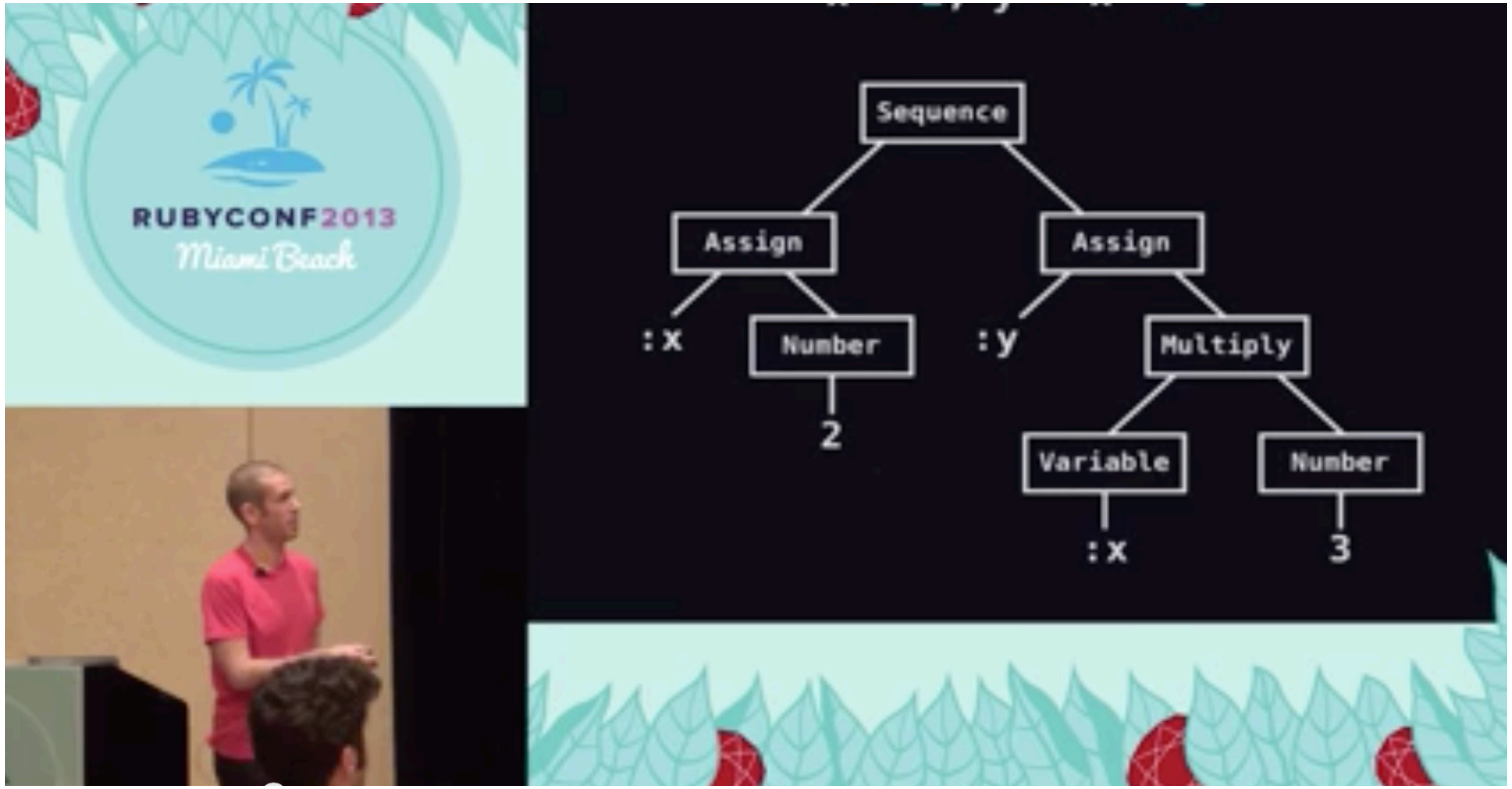
```java
@NodeInfo(shortName = "eval")
public abstract class SLEvalBuiltin extends SLBuiltinNode {

    @SuppressWarnings("unused")
    @Specialization(guards = {
                    "stringsEqual(mimeType, cachedMimeType)",
                    "stringsEqual(code, cachedCode)"
    })
    public Object evalCached(VirtualFrame frame,
                    String mimeType, String code,
                    @Cached("mimeType") String cachedMimeType,
                    @Cached("code") String cachedCode,
                    @Cached("create(parse(mimeType, code))") DirectCallNode callNode) {
        return callNode.call(frame, new Object[]{});
    }

    @TruffleBoundary
    @Specialization(contains = "evalCached")
    public Object evalUncached(String mimeType, String code) {
        return parse(mimeType, code).call();
    }

}
```



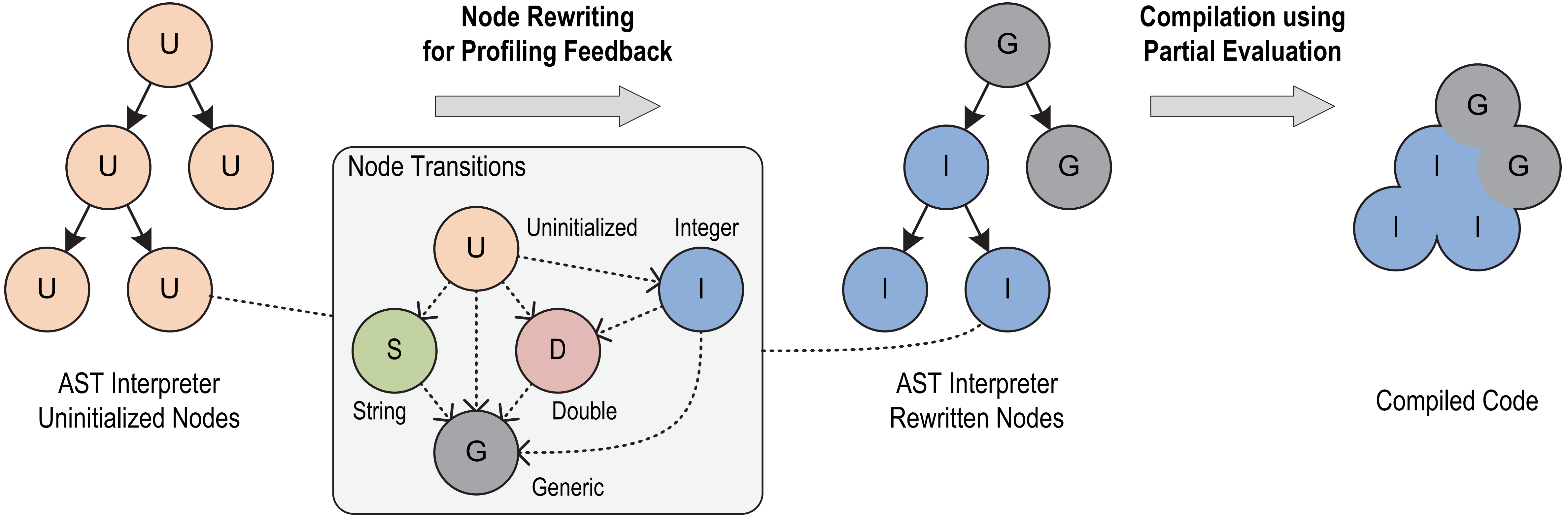
AST Interpreter
Rewritten Nodes

Compiled Code

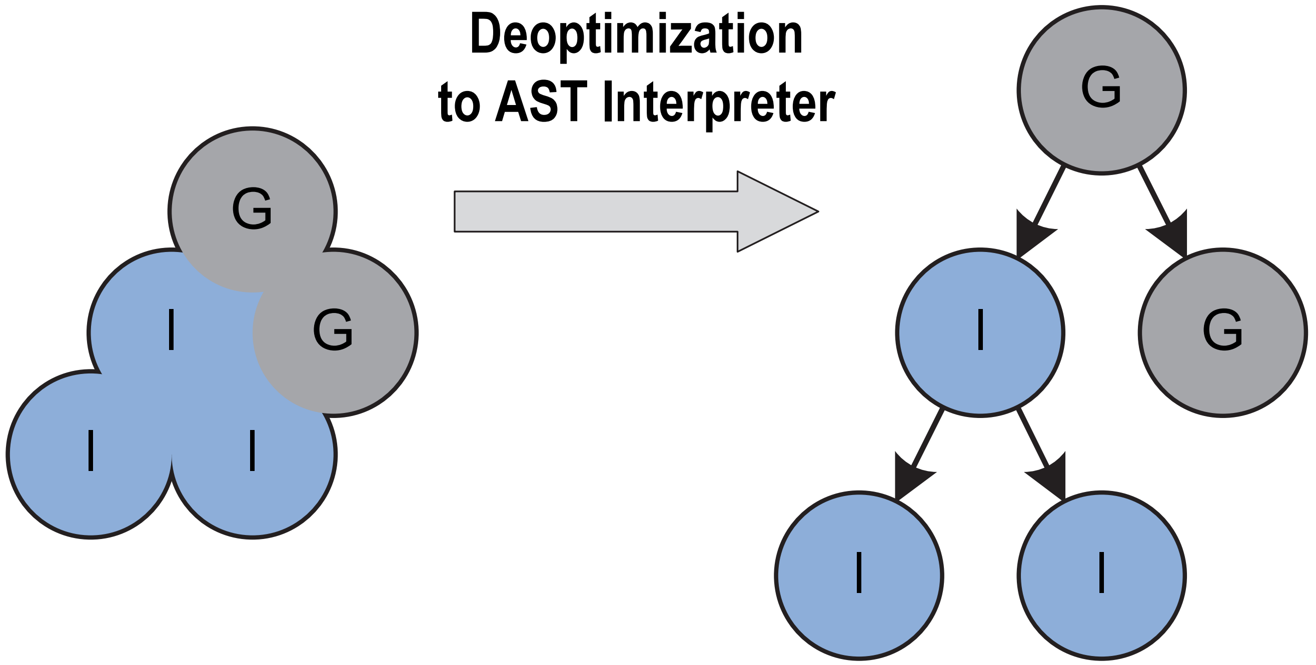
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



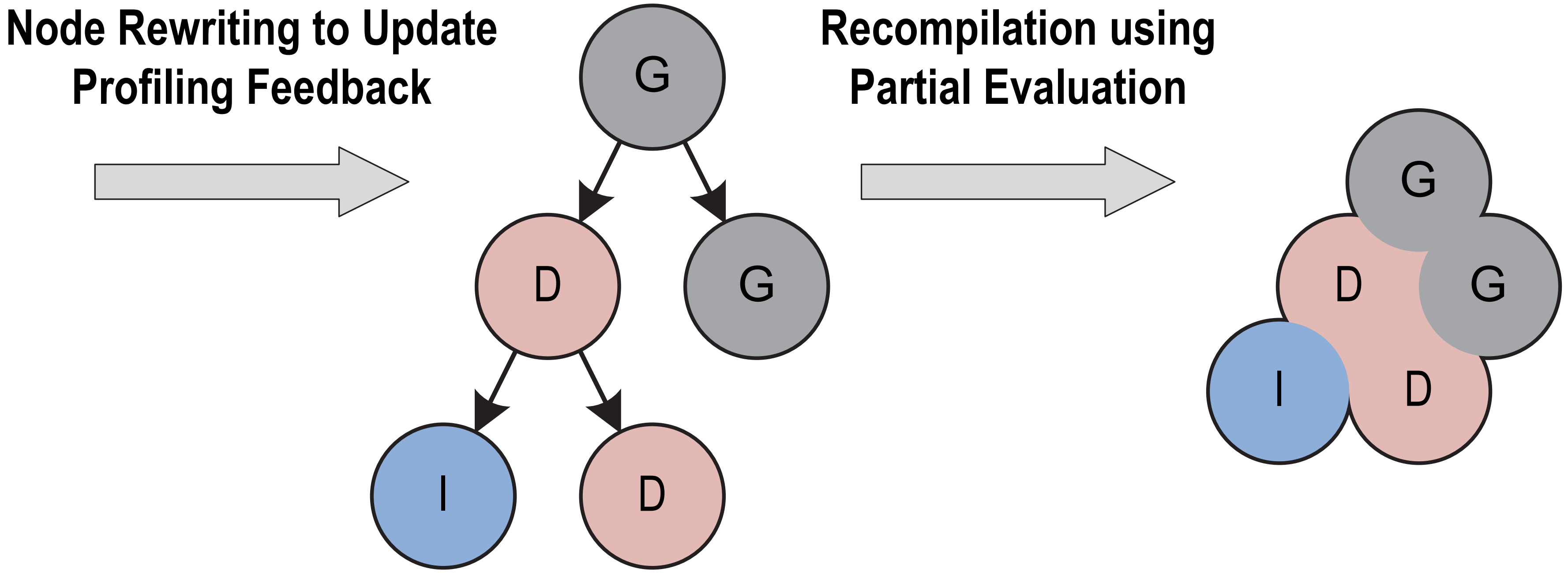
codon.com/compilers-for-free



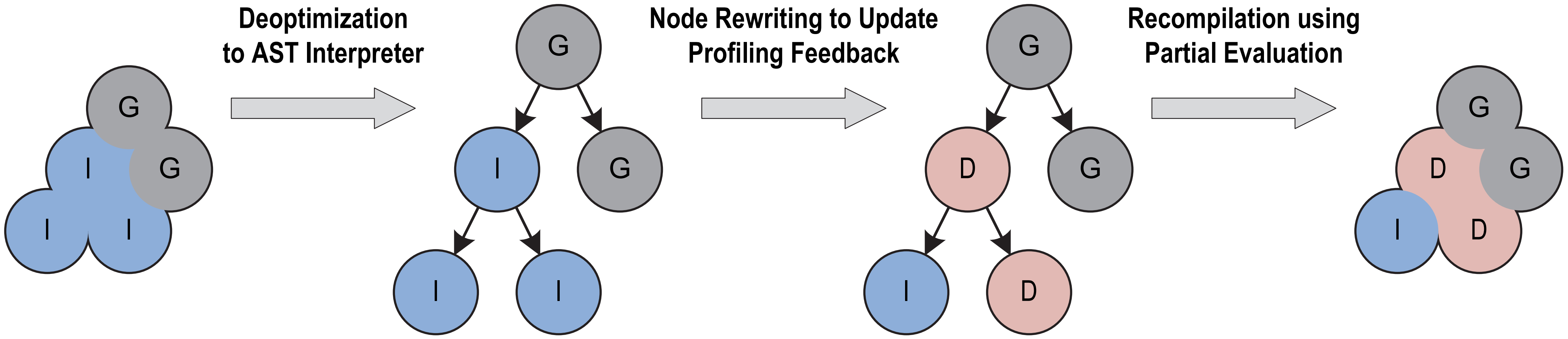
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



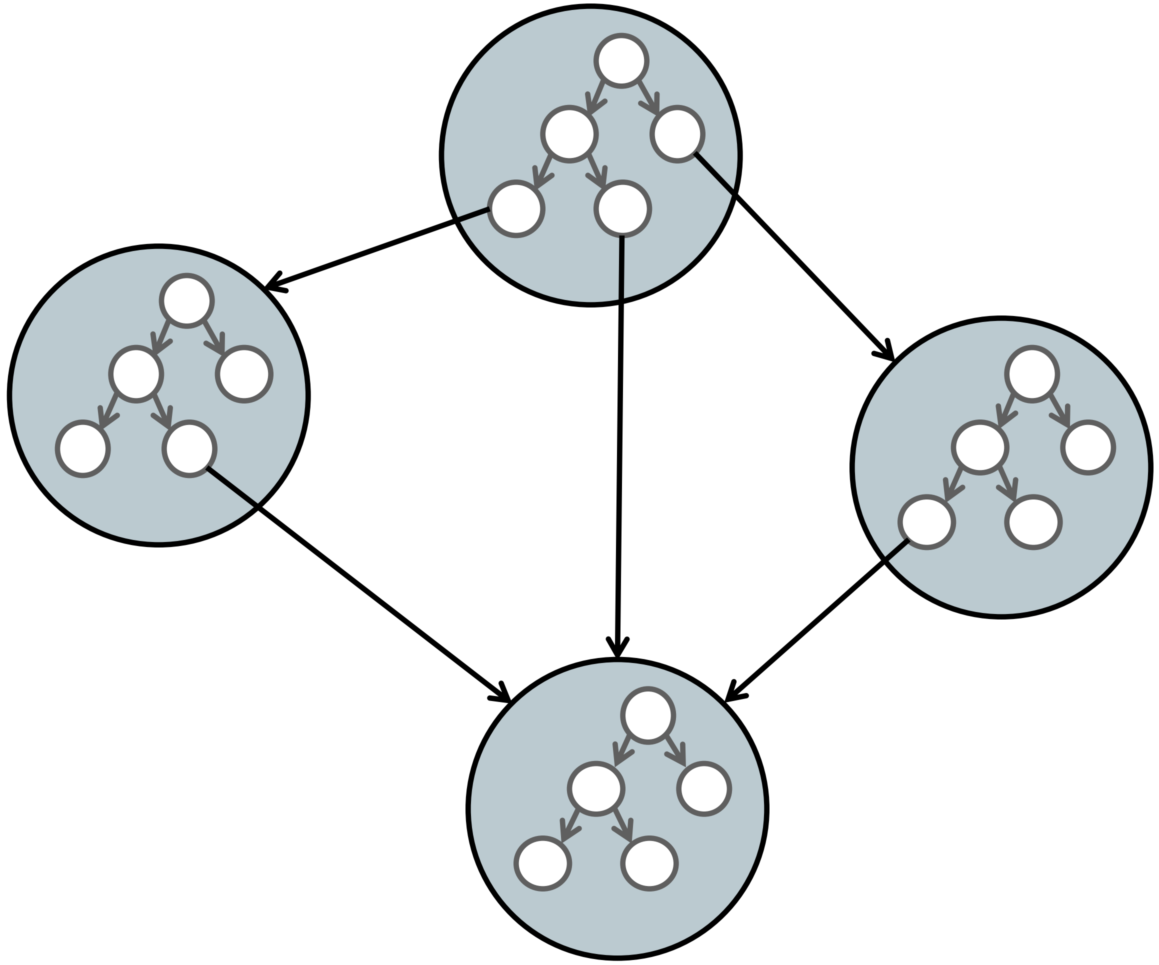
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

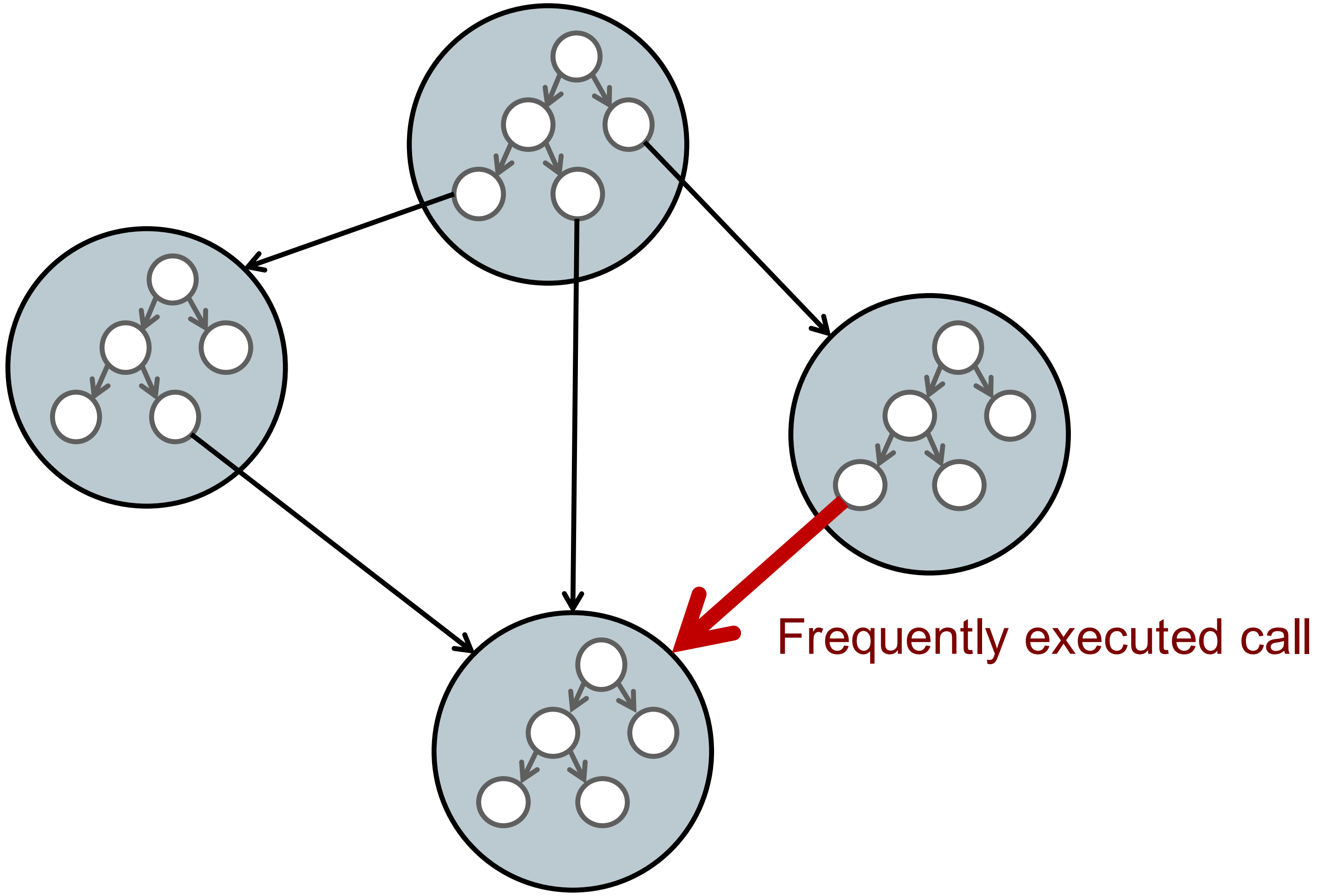

Node Rewriting to Update Profiling Feedback
G
D
G
I
D
Recompilation using Partial Evaluation
G
D
G
I
D

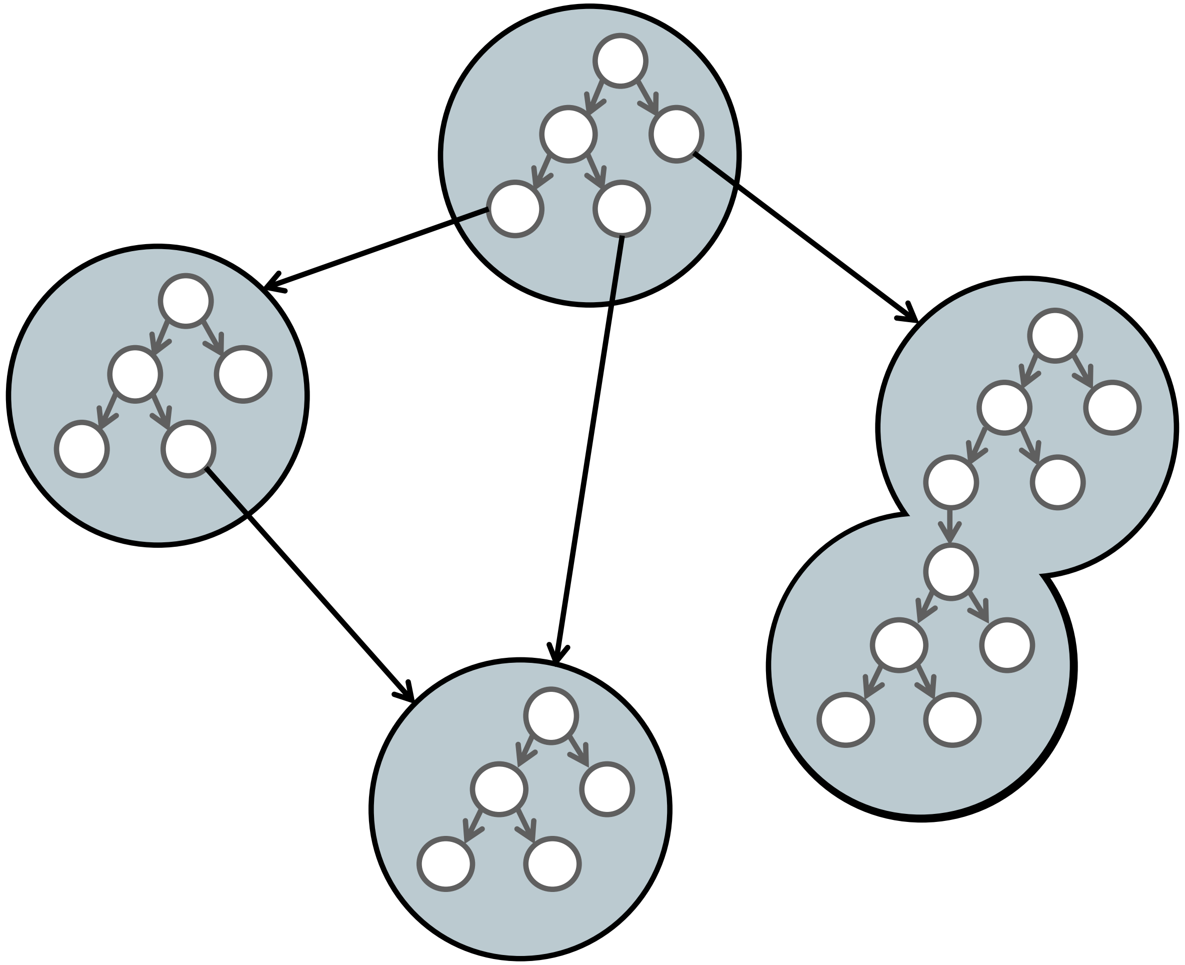
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

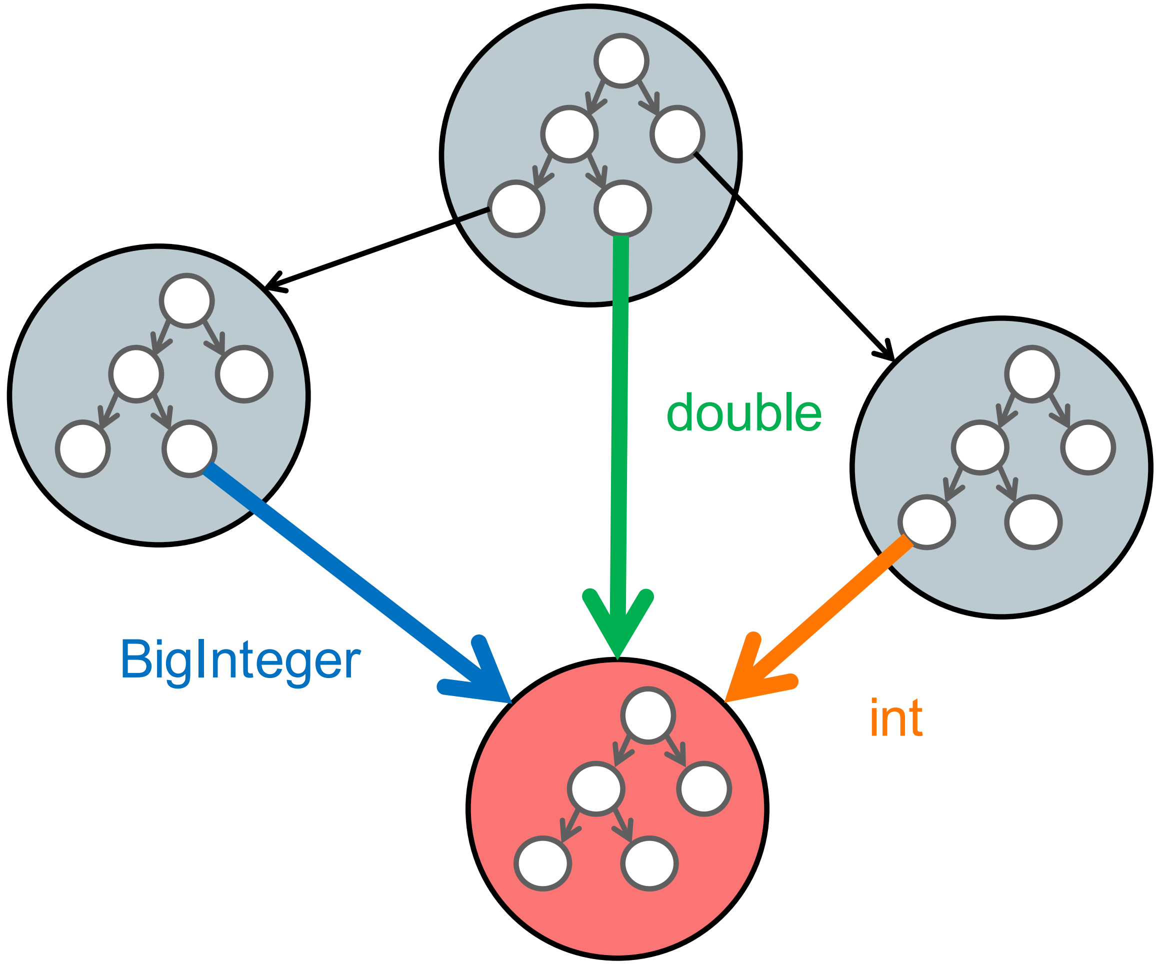

Deoptimization to AST Interpreter
Node Rewriting to Update Profiling Feedback
Recompilation using Partial Evaluation
G
I
D

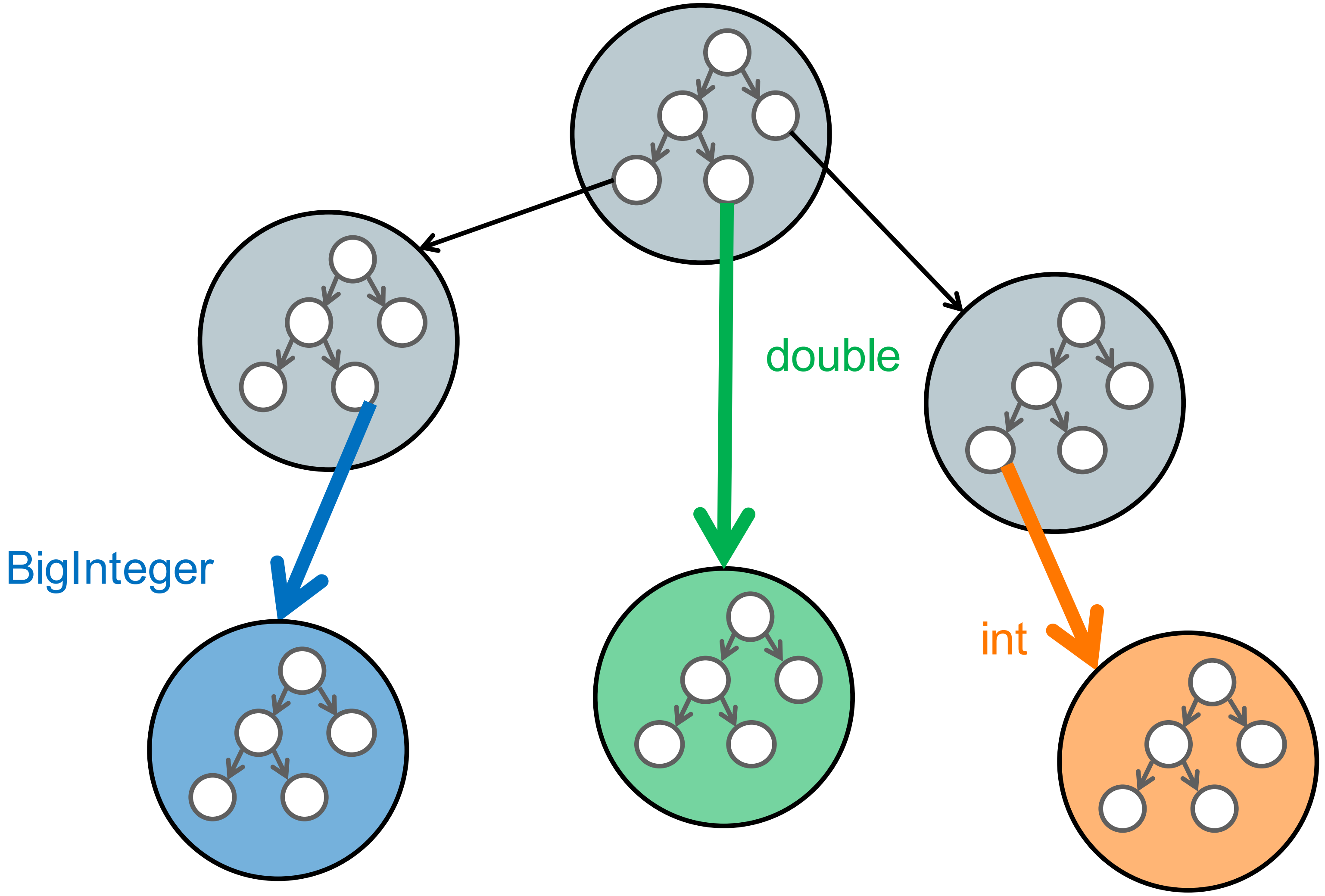
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

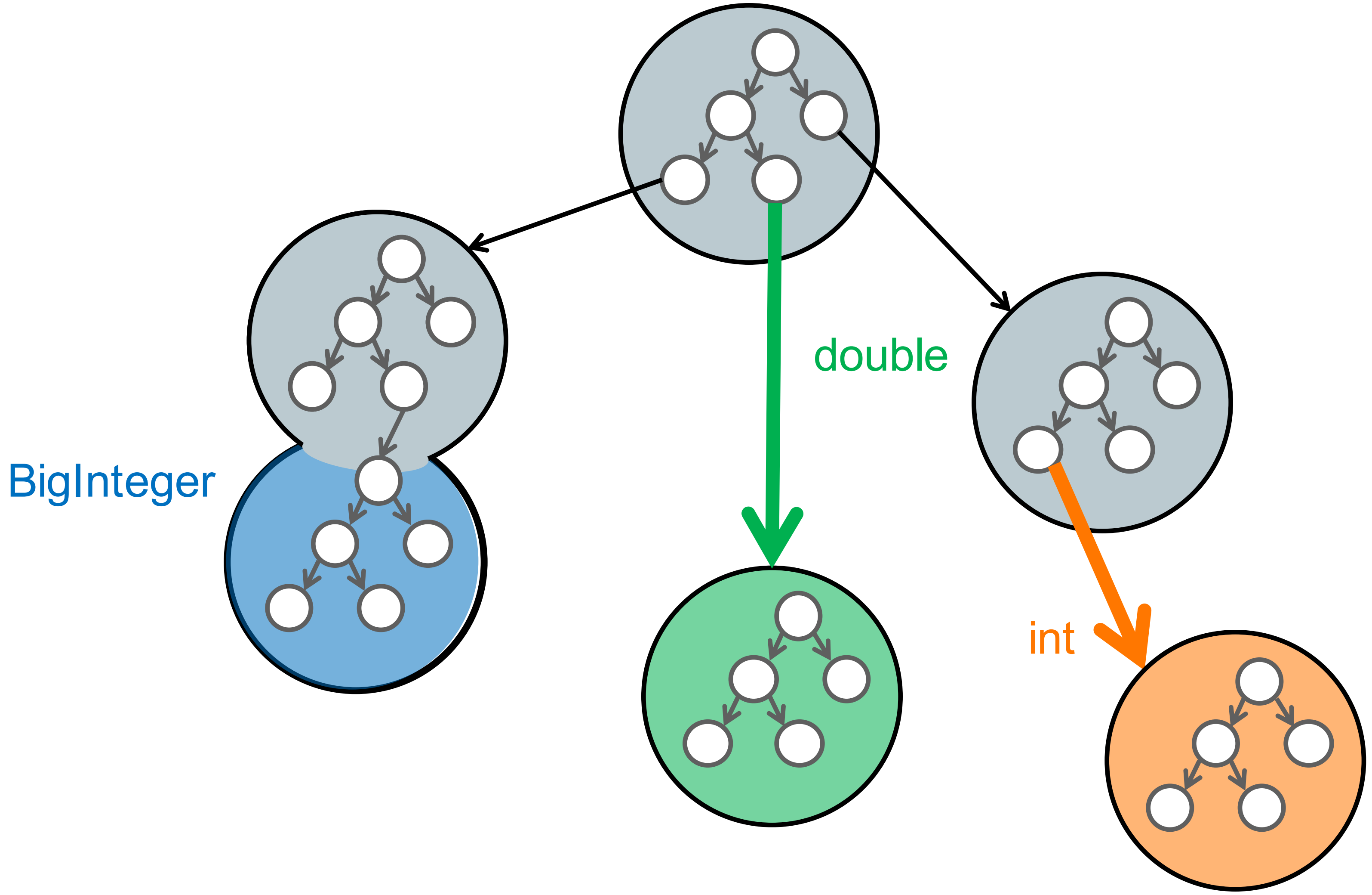



Frequently executed call

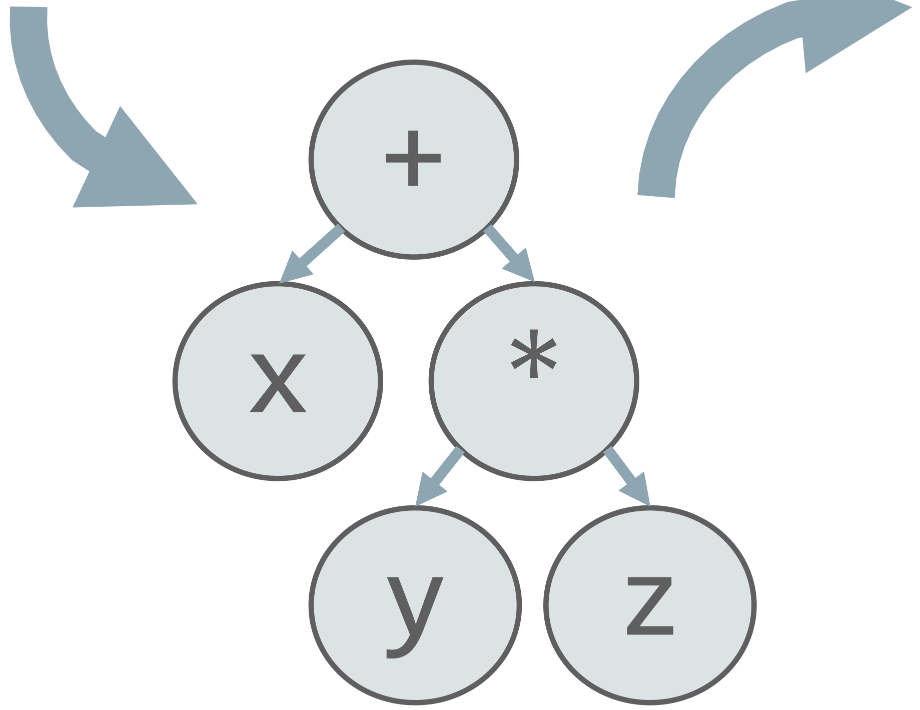



double
BigInteger
int


double
BigInteger
int


double
BigInteger
int

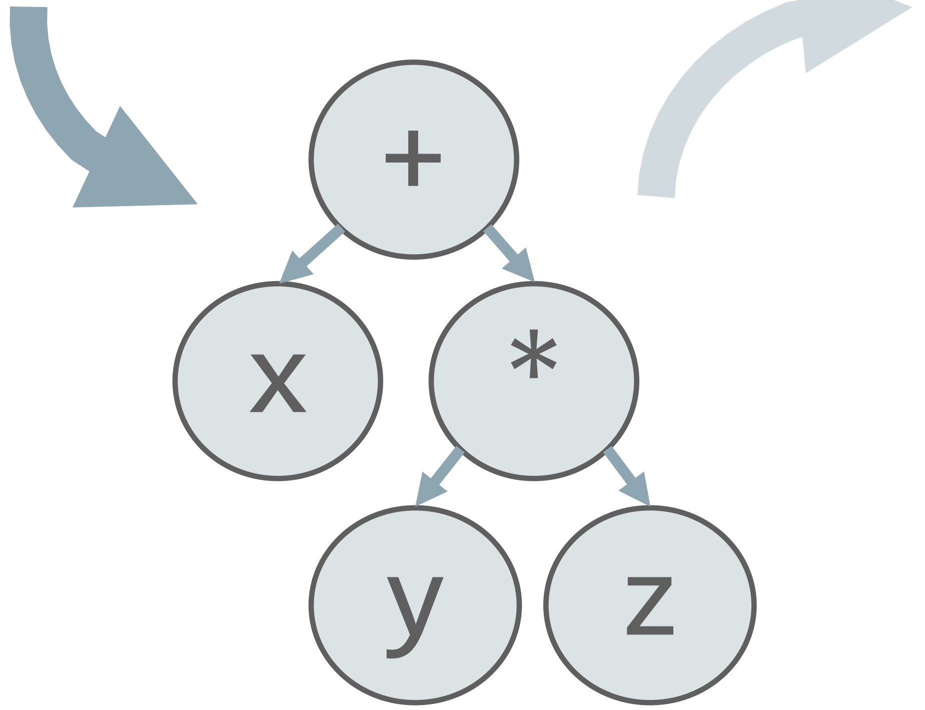
```
x + y * z
```



```
load_local x
load_local y
load_local z
call :*
call :+
```

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rdx, -24(%rbp)
movq  -16(%rbp), %rax
movl  %eax, %edx
movq  -24(%rbp), %rax
imull %edx, %eax
movq  -8(%rbp), %rdx
addl  %edx, %eax
popq  %rbp
ret
```

```
x + y * z
```

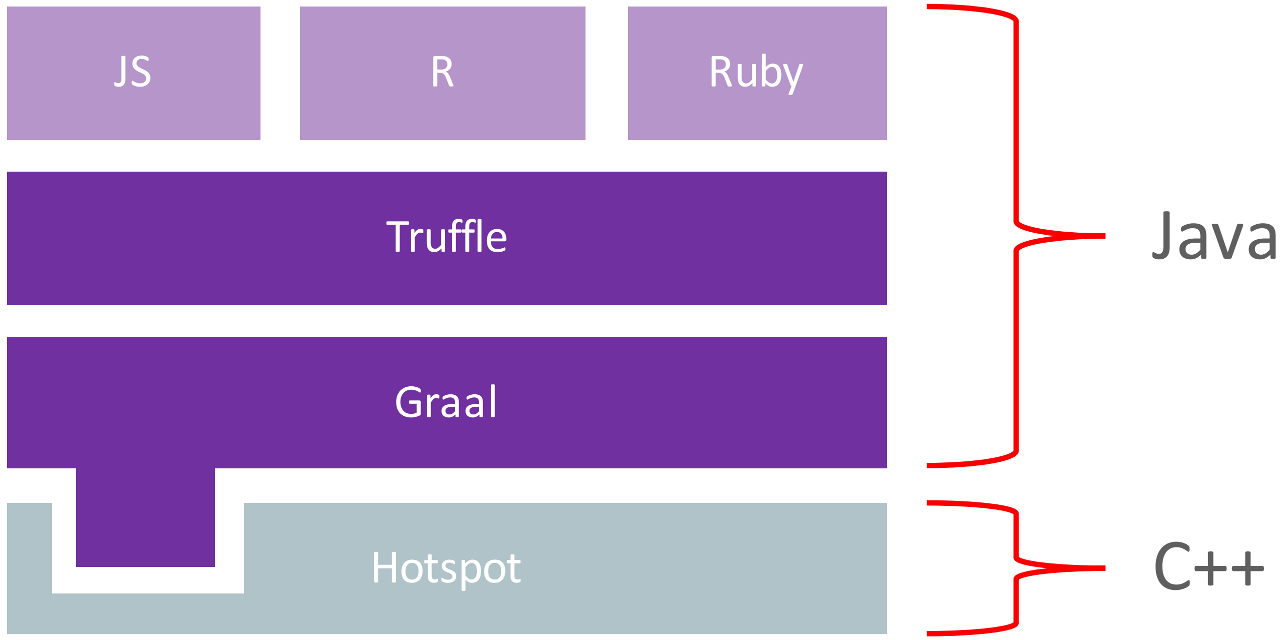


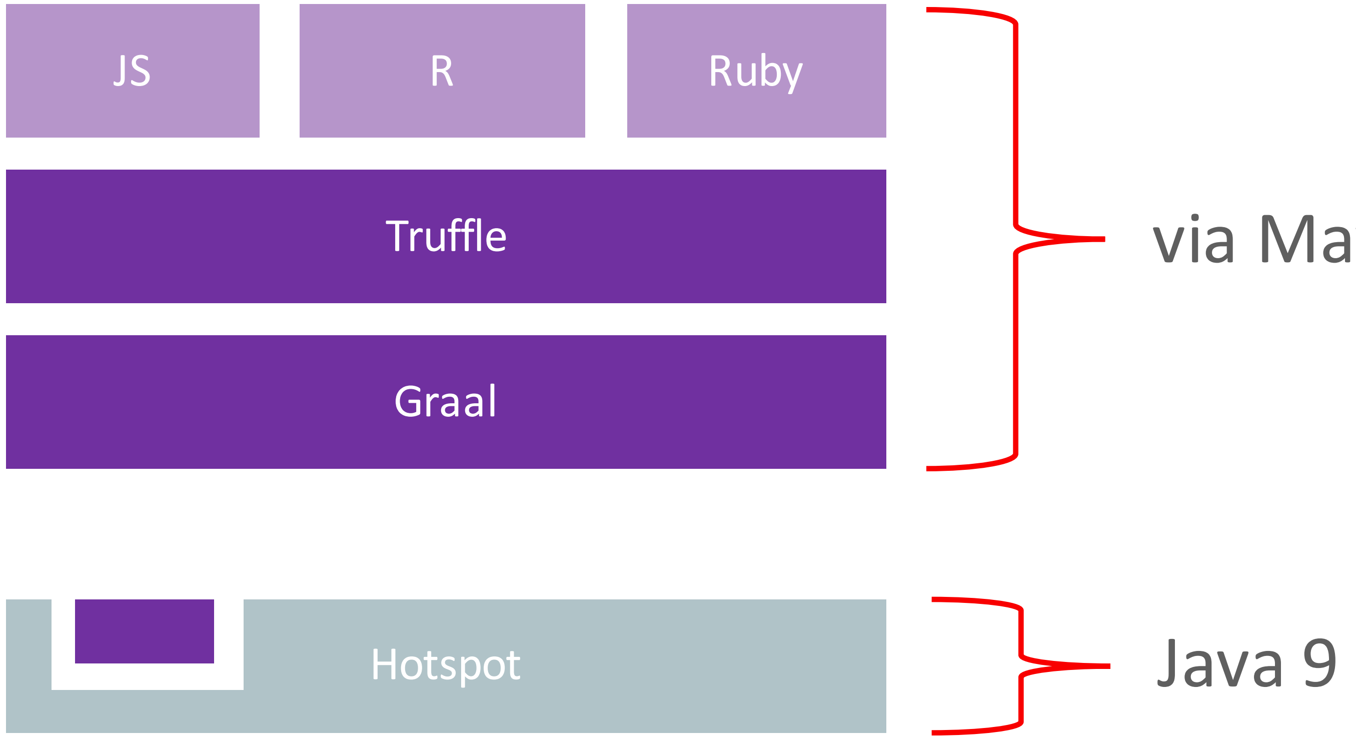
```
load_local x
load_local y
load_local z
call :*
call :+
```

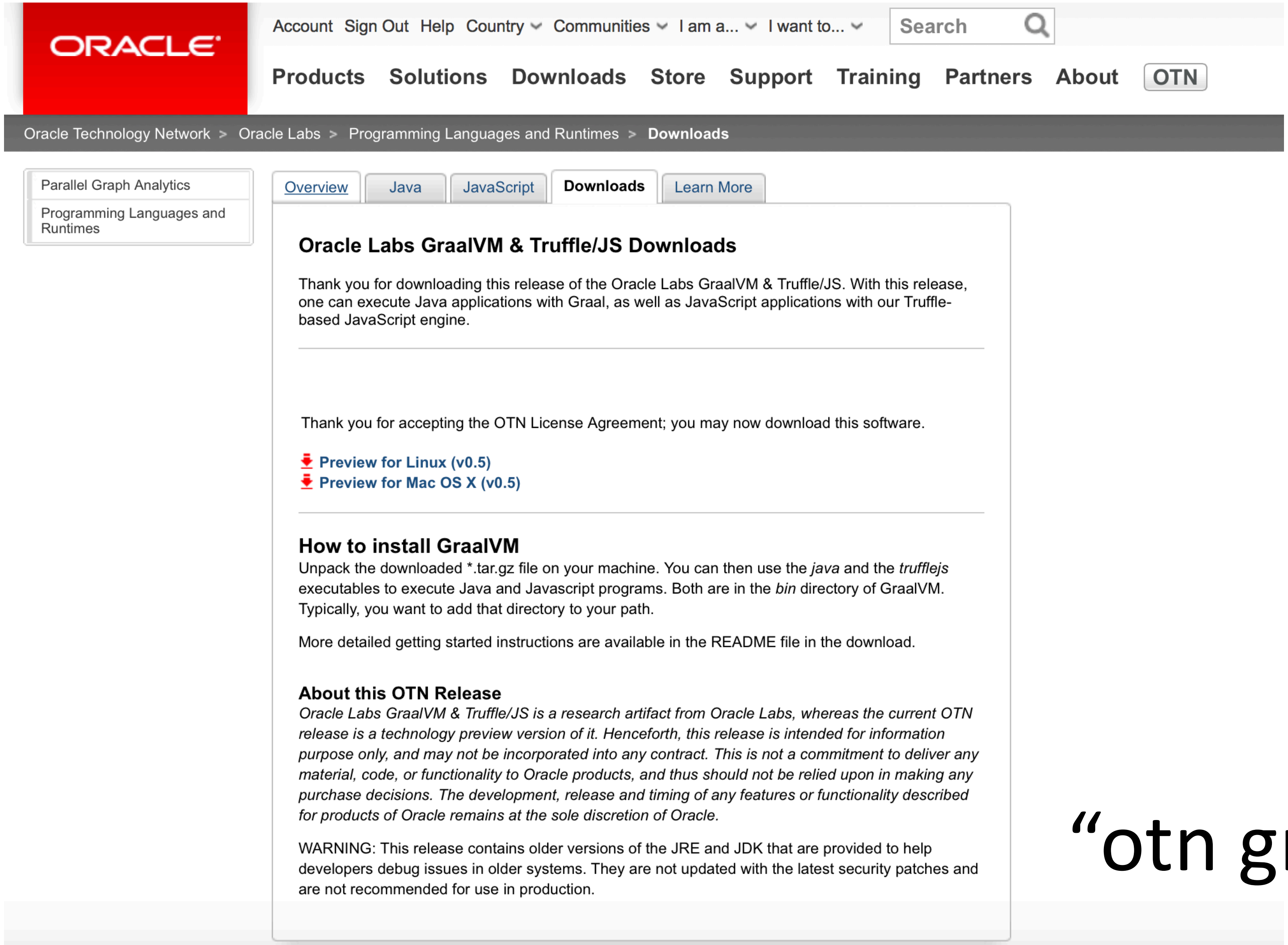
```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rdx, -24(%rbp)
movq  -16(%rbp), %rax
movl  %eax, %edx
movq  -24(%rbp), %rax
imull %edx, %eax
movq  -8(%rbp), %rdx
addl  %edx, %eax
popq  %rbp
ret
```

Will I be able to use Truffle and Graal for real?

JVMCI
(JVM Compiler Interface)




JS
R
Ruby
Truffle
Graal
Hotspot
via Maven etc
Java 9


Account Sign Out Help Country Communities I am a... I want to... Search
Products Solutions Downloads Store Support Training Partners About OTN
Oracle Technology Network > Oracle Labs > Programming Languages and Runtimes > Downloads
Parallel Graph Analytics
Programming Languages and Runtimes
Overview Java JavaScript Downloads Learn More

Oracle Labs GraalVM & Truffle/JS Downloads

Thank you for downloading this release of the Oracle Labs GraalVM & Truffle/JS. With this release, one can execute Java applications with Graal, as well as JavaScript applications with our Truffle-based JavaScript engine.

Thank you for accepting the OTN License Agreement; you may now download this software.

- **Preview for Linux (v0.5)**
- **Preview for Mac OS X (v0.5)**

How to install GraalVM

Unpack the downloaded *.tar.gz file on your machine. You can then use the *java* and the *trufflejs* executables to execute Java and Javascript programs. Both are in the *bin* directory of GraalVM. Typically, you want to add that directory to your path.

More detailed getting started instructions are available in the README file in the download.

About this OTN Release

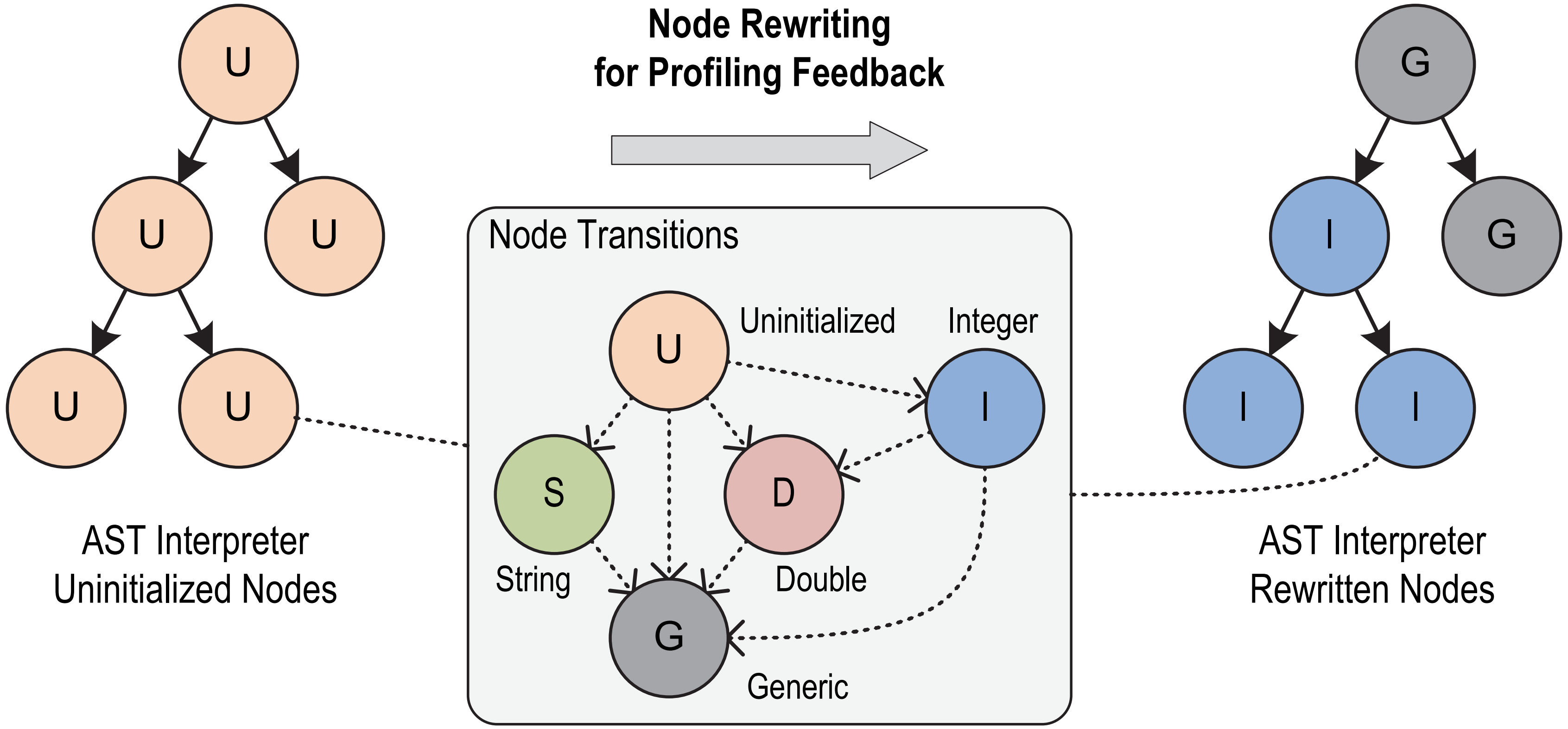
Oracle Labs GraalVM & Truffle/JS is a research artifact from Oracle Labs, whereas the current OTN release is a technology preview version of it. Henceforth, this release is intended for information purpose only, and may not be incorporated into any contract. This is not a commitment to deliver any material, code, or functionality to Oracle products, and thus should not be relied upon in making any purchase decisions. The development, release and timing of any features or functionality described for products of Oracle remains at the sole discretion of Oracle.

WARNING: This release contains older versions of the JRE and JDK that are provided to help developers debug issues in older systems. They are not updated with the latest security patches and are not recommended for use in production.

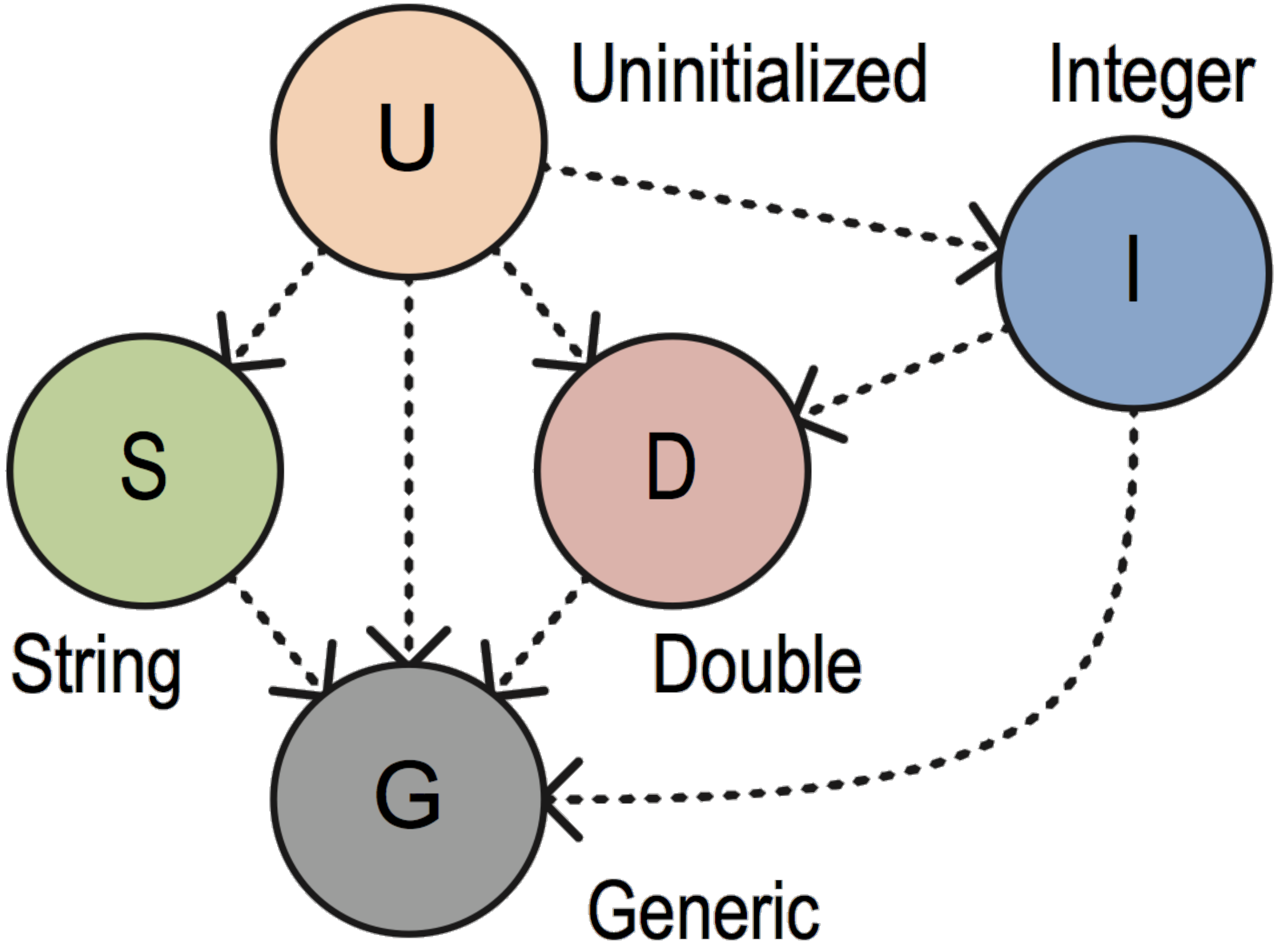
"otn graal"

How Truffle solves the problem of optimising Ruby

First problem: JRuby’s core library is **megamorphic**



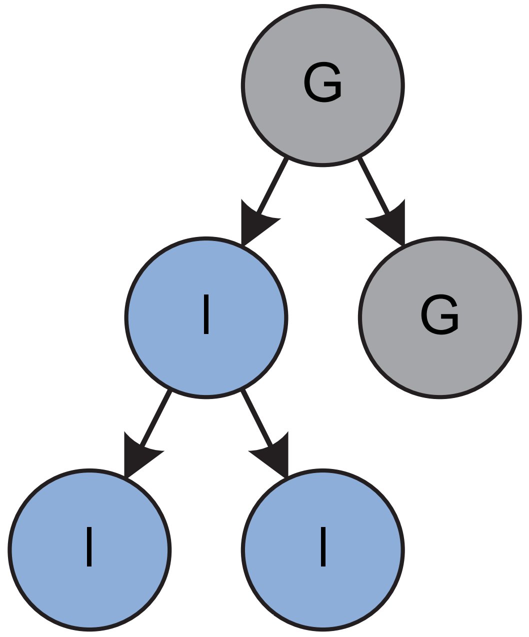
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



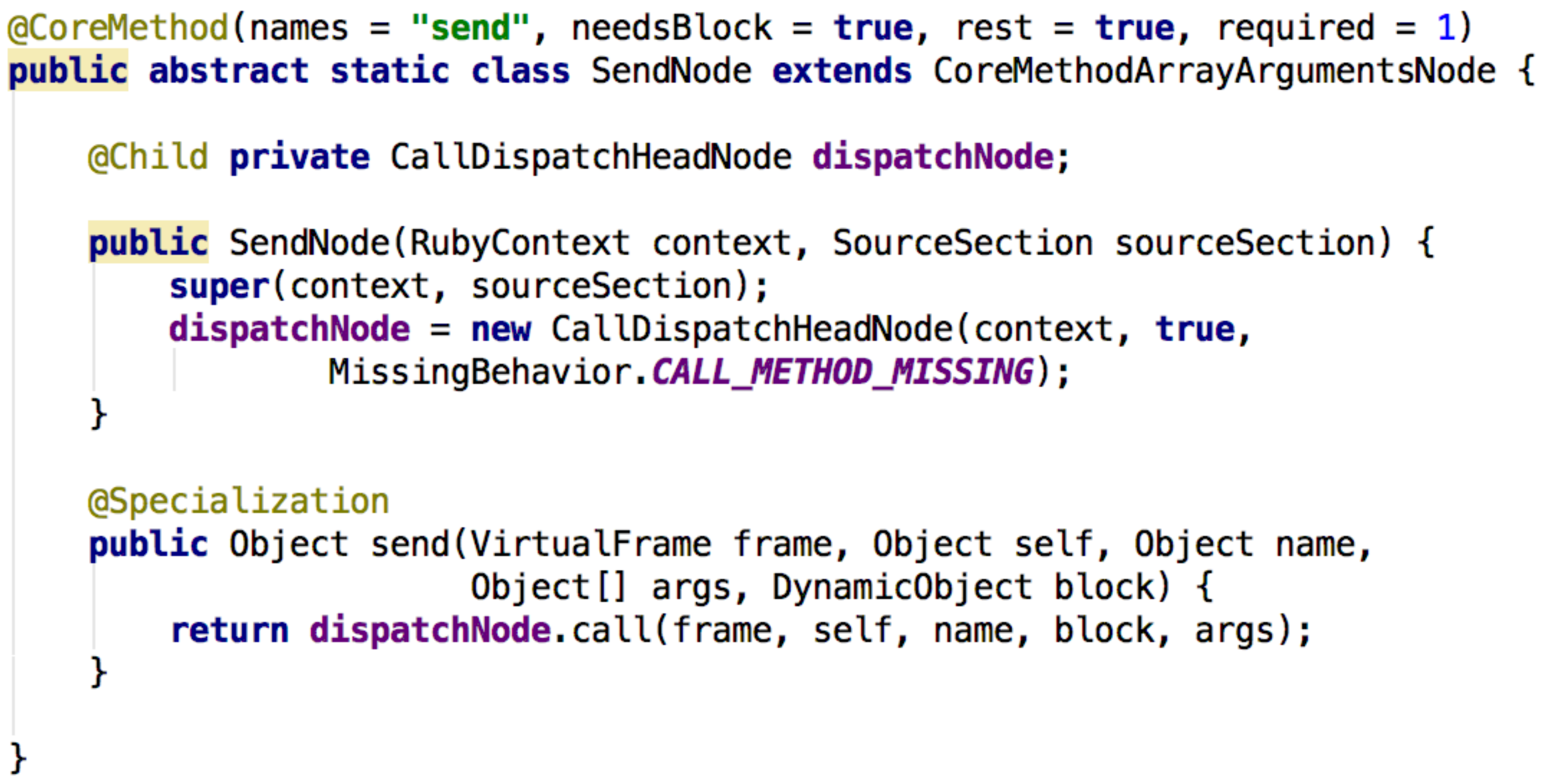
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

```java
@Specialization(rewriteOn = ArithmeticException.class)
public int add(int a, int b) {
    return ExactMath.addExact(a, b);
}

@Specialization(rewriteOn = ArithmeticException.class)
public long add(long a, long b) {
    return ExactMath.addExact(a, b);
}

@Specialization
public Object addWithOverflow(long a, long b) {
    return fixnumOrBignum(BigInteger.valueOf(a).add(BigInteger.valueOf(b)));
}

@Specialization
public double add(long a, double b) {
    return a + b;
}
```

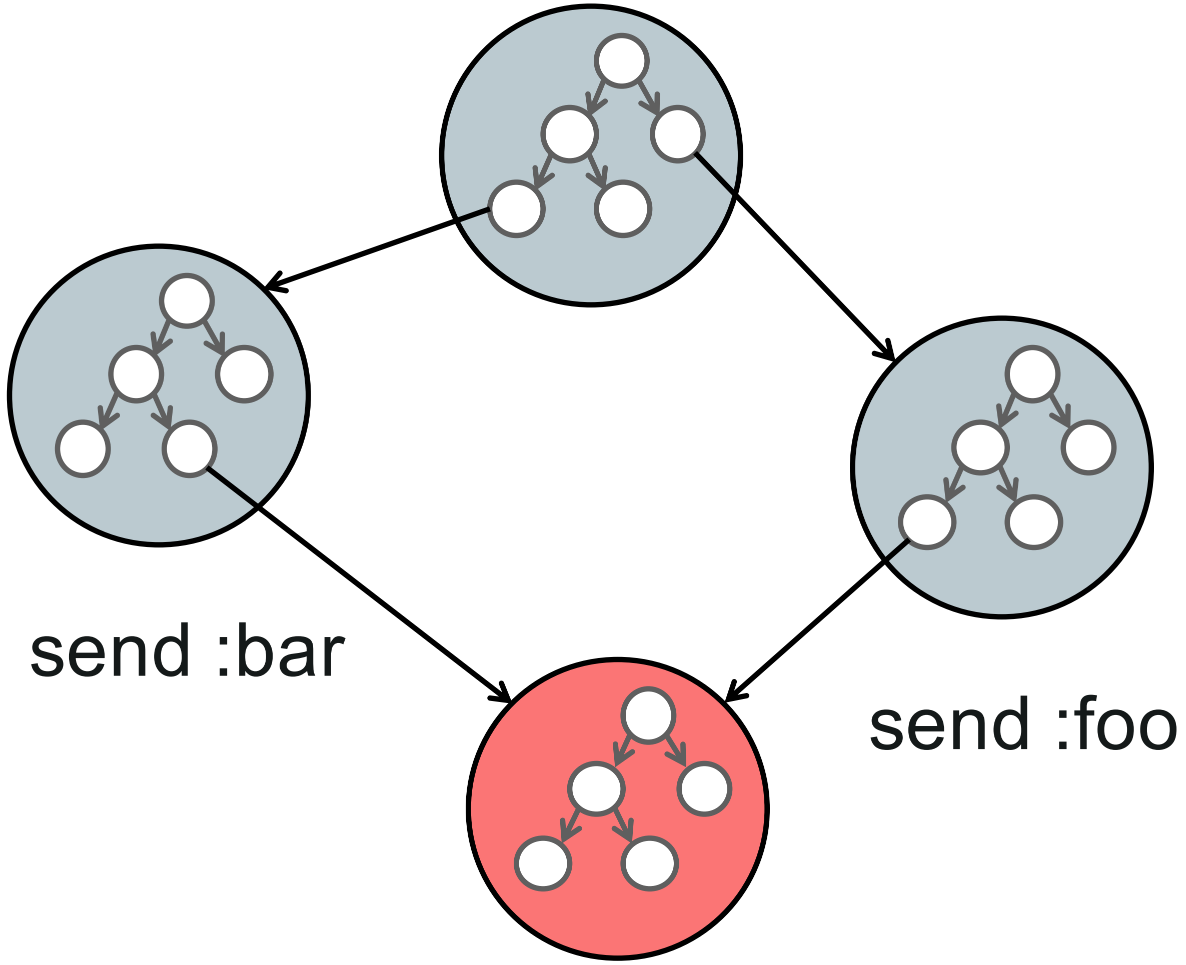
Second problem: JRuby’s core library is **stateless**

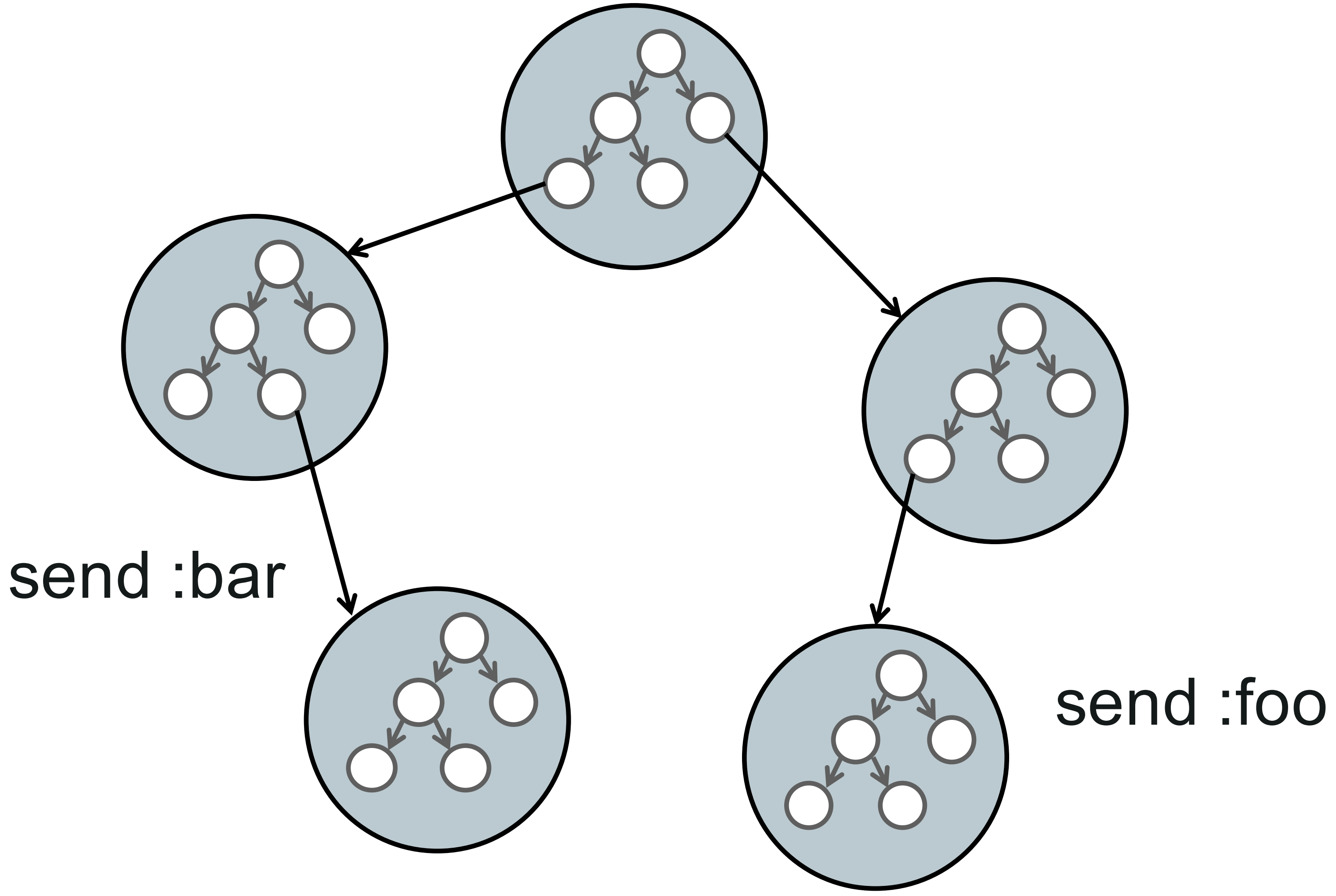


T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



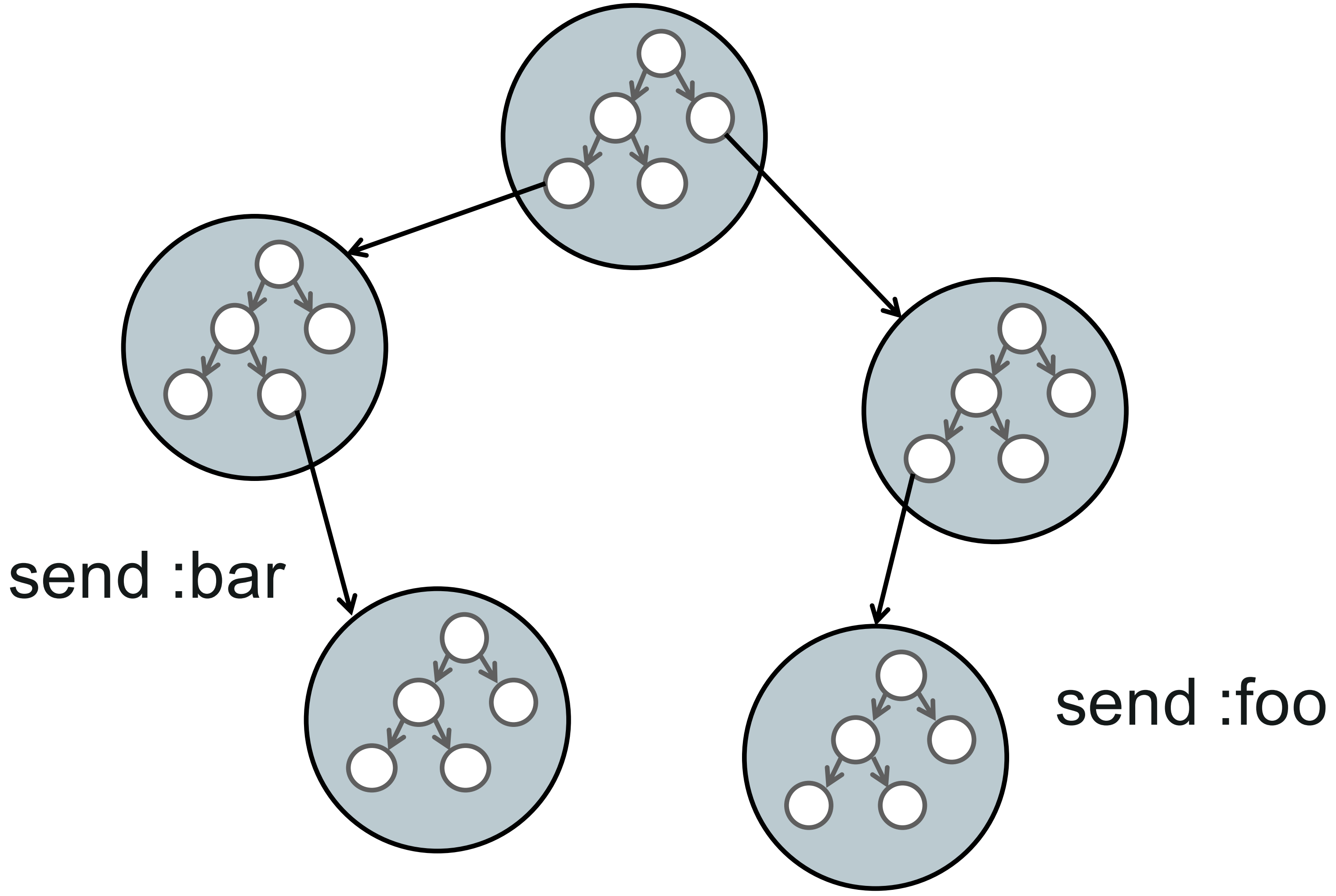
```java
@CoreMethod(names = "send", needsBlock = true, rest = true, required = 1)
public abstract static class SendNode extends CoreMethodArrayArgumentsNode {

    @Child private CallDispatchHeadNode dispatchNode;

    public SendNode(RubyContext context, SourceSection sourceSection) {
        super(context, sourceSection);
        dispatchNode = new CallDispatchHeadNode(context, true,
                MissingBehavior.CALL_METHOD_MISSING);
    }

    @Specialization
    public Object send(VirtualFrame frame, Object self, Object name,
                       Object[] args, DynamicObject block) {
        return dispatchNode.call(frame, self, name, block, args);
    }

}
```


send :bar
send :foo


send :bar
send :foo

```java
public static class IntegerArrayBuilderNode extends ArrayBuilderNode {

    private final int expectedLength;

    public IntegerArrayBuilderNode(RubyContext context, int expectedLength) {
        super(context);
        this.expectedLength = expectedLength;
    }

    @Override
    public Object start() {
        return new int[expectedLength];
    }
```

Third problem: JRuby’s core library is **very deep**


send :bar
send :foo

Fourth problem: JRuby’s core library isn’t **amenable to optimisations**

```java
@CoreMethod(names = "sort", needsBlock = true)
public abstract class SortNode extends ArrayCoreMethodNode {

    @Child private CallDispatchHeadNode compareDispatchNode;

    @ExplodeLoop
    @Specialization
    public DynamicObject sortVeryShort(VirtualFrame frame, DynamicObject array) {
        final int size = getSize(array);

        // Copy with a exploded loop for PE

        for (int i = 0; i < getContext().getOptions().ARRAY_SMALL; i++) {
            if (i < size) {
                store.set(i, originalStore.get(i));
            }
        }

        // Selection sort - written very carefully to allow PE

        for (int i = 0; i < getContext().getOptions().ARRAY_SMALL; i++) {
            if (i < size) {
                for (int j = i + 1; j < getContext().getOptions().ARRAY_SMALL; j++) {
                    if (j < size) {
                        final Object a = store.get(i);
                        final Object b = store.get(j);
                        if (((int) compareDispatchNode.call(frame, b, "<=>", null, a)) < 0) {
                            store.set(j, a);
                            store.set(i, b);
                        }
                    }
                }
            }
        }

        return createArray(getContext(), store, size);
    }

}
```

ORACLE

@ExplodeLoop

```
// Selection sort - written very carefully to allow PE

for (int i = 0; i < getContext().getOptions().ARRAY_SMALL; i++) {
    if (i < size) {
        for (int j = i + 1; j < getContext().getOptions().ARRAY_SMALL; j++) {
            if (j < size) {
                final Object a = store.get(i);
                final Object b = store.get(j);
                if (((int) compareDispatchNode.call(frame, b, "<=>", null, a)) < 0) {
                    store.set(j, a);
                    store.set(i, b);
                }
            }
        }
    }
}
```

A simple example

```
def min(a, b)
  [a, b].sort[0]
end


puts min(2, 8)
```

```
~~def min(a, b)~~
~~    [a, b].sort[0]~~
~~end~~


puts [2, 8].sort[0]
```

```
t0 = 2 <=> 8
t1 = t0 < 0 ? 2 : 8
t2 = t0 > 0 ? 8 : 2
t3 = [t1, t2]

puts t3[0]
```

```
t0 = 2 <=> 8
t1 = t0 < 0 ? 2 : 8
~~t2 = t0 > 0 ? 8 : 2~~
~~t3 = [t1, t2]~~

puts t1
```

```
t0 = -1
t1 = t0 < 0 ? 2 : 8


puts t1
```

```
~~t0 = -1~~
t1 = -1 < 0 ? 2 : 8


puts t1
```

```
t1 = true ? 2 : 8

puts t1
```

```
t1 = 2

puts t1
```

```
~~t1 = 2~~

puts 2
puts 2
```

```
t0 = a <=> b
t1 = t0 < 0 ? a : b


puts t1
```

```
~~t0 = a <=> b~~
t1 = (a <=> b) < 0 ? a : b


puts t1
```

~~`t1 = (a <=> b) < 0 ? a : b`~~

```
puts (a <=> b) < 0 ? a : b
puts (a <=> b) < 0 ? a : b
```

A deliberately extreme example

```
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

```
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

```ruby
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end
```

```ruby
class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

ORACLE®

```
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

```
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

```
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

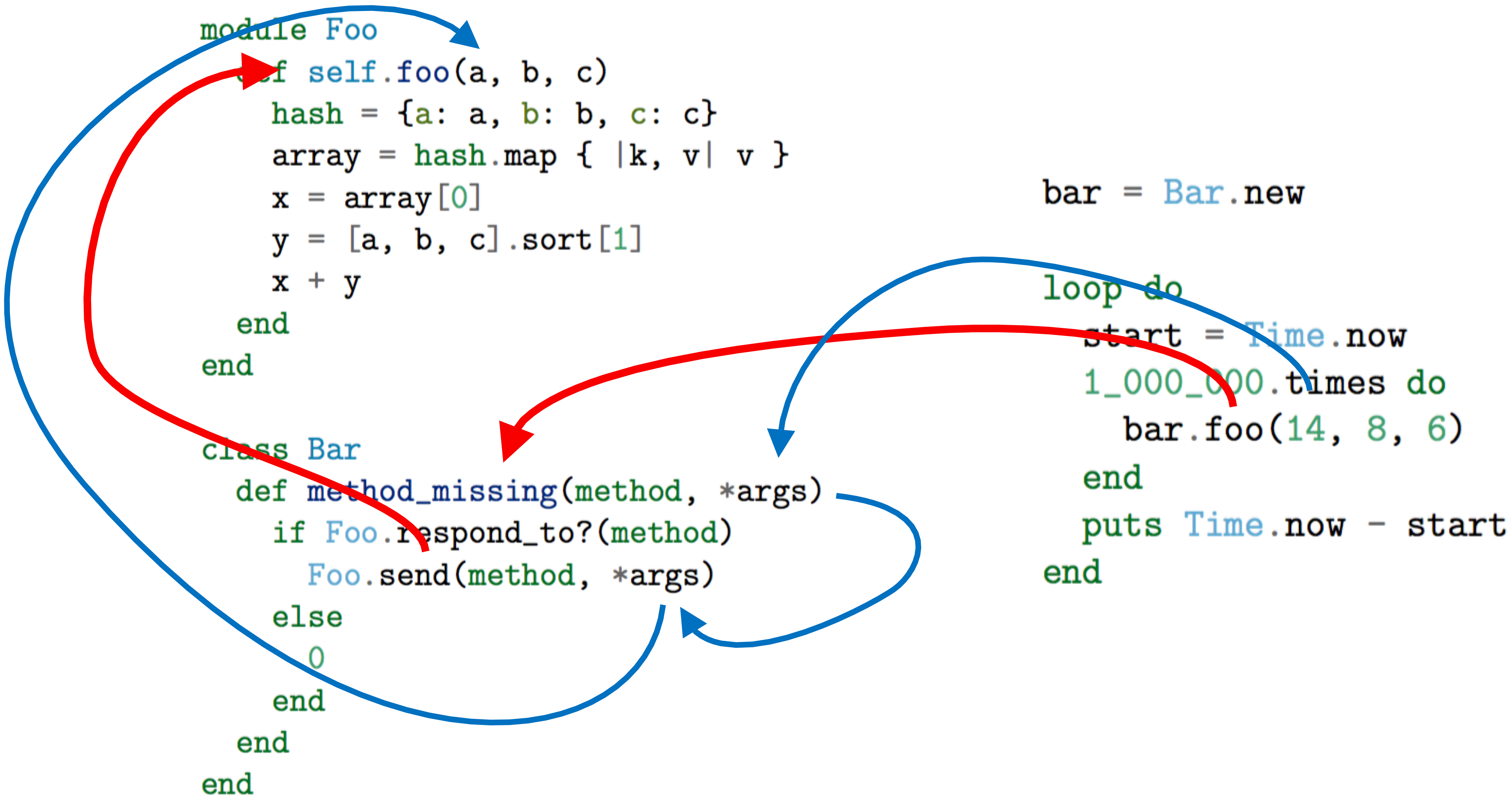
```ruby
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

```ruby
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

```ruby
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

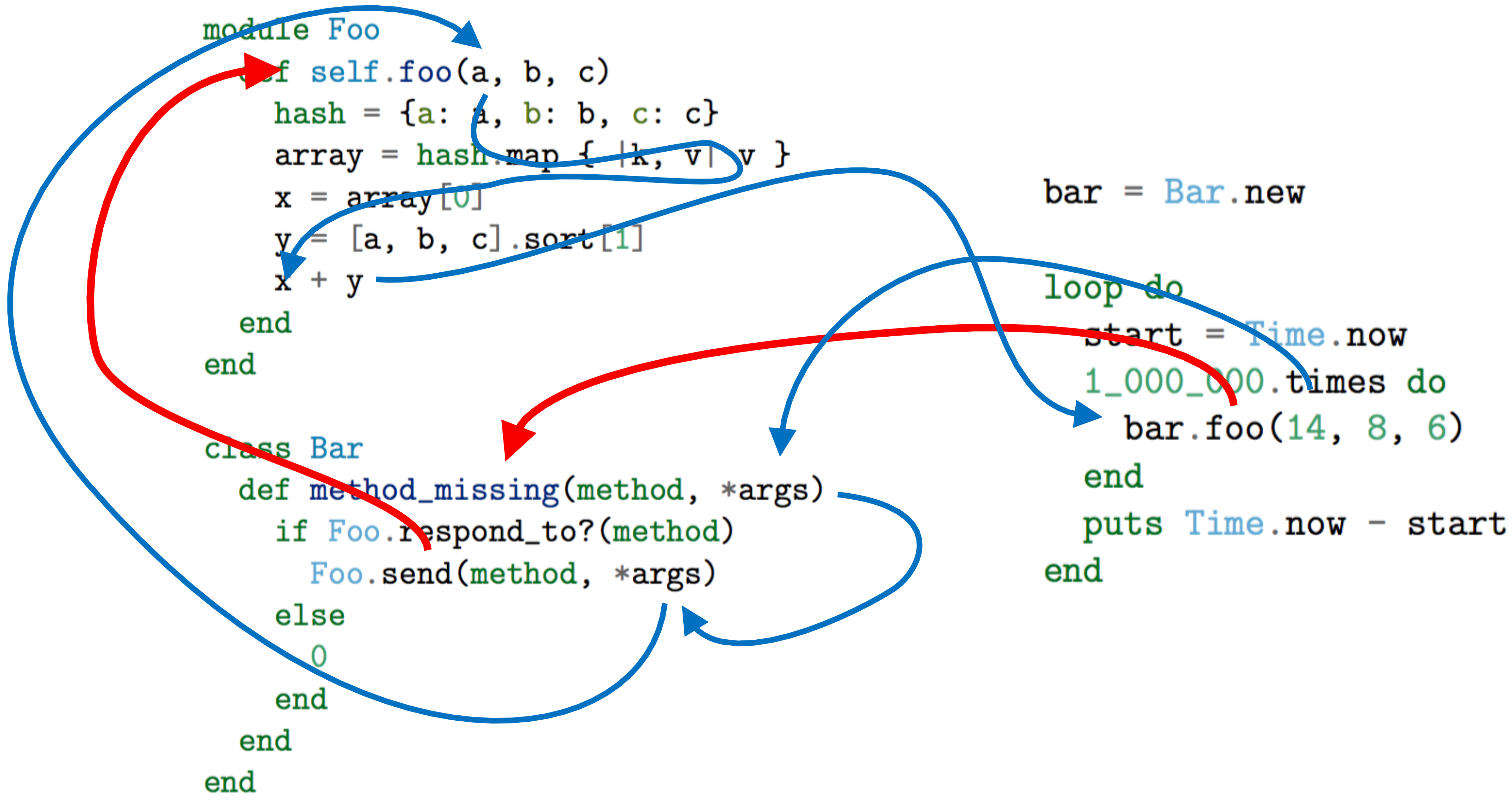
```ruby
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

ORACLE



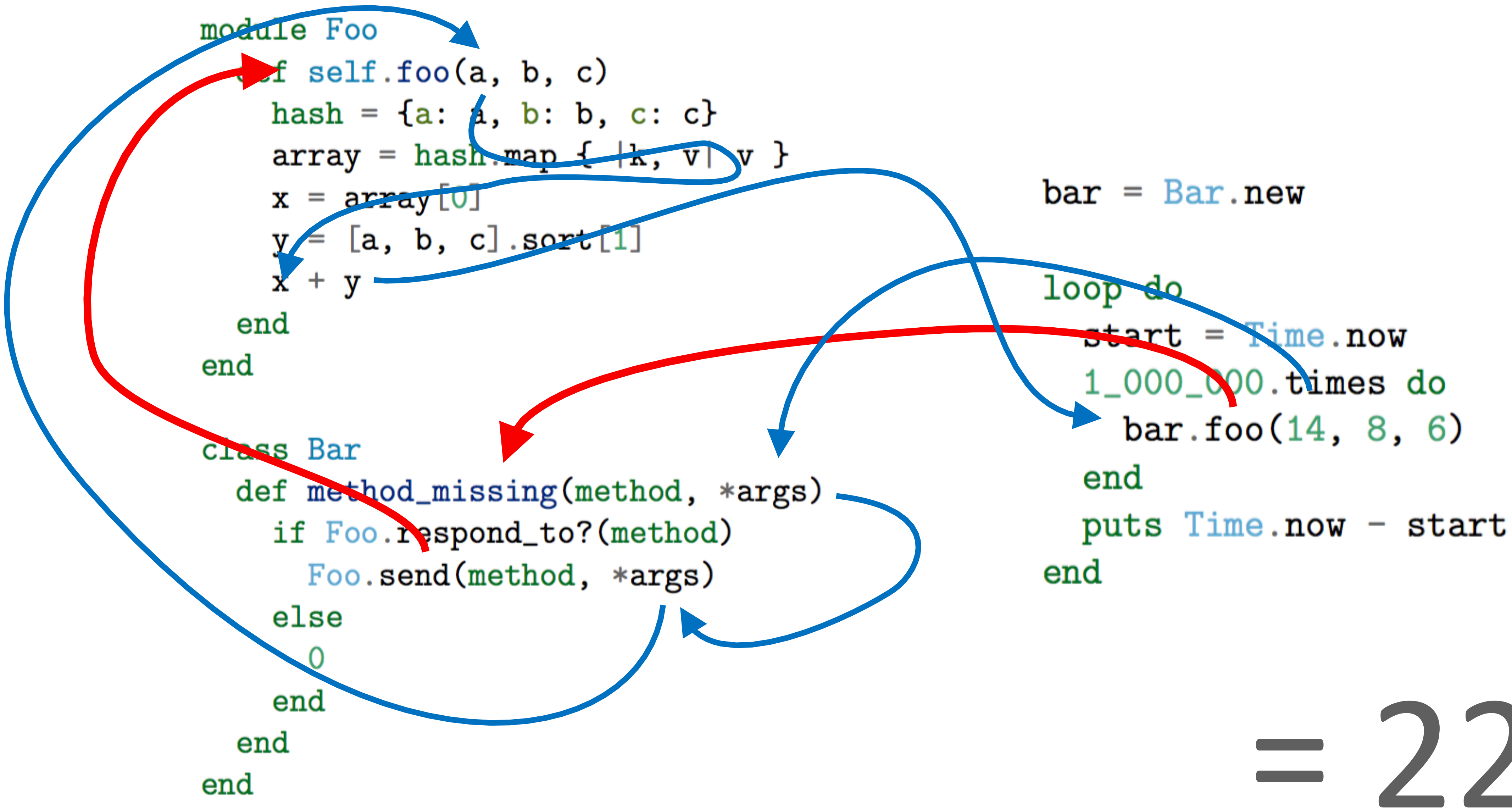
```ruby
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

```ruby
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```



```
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

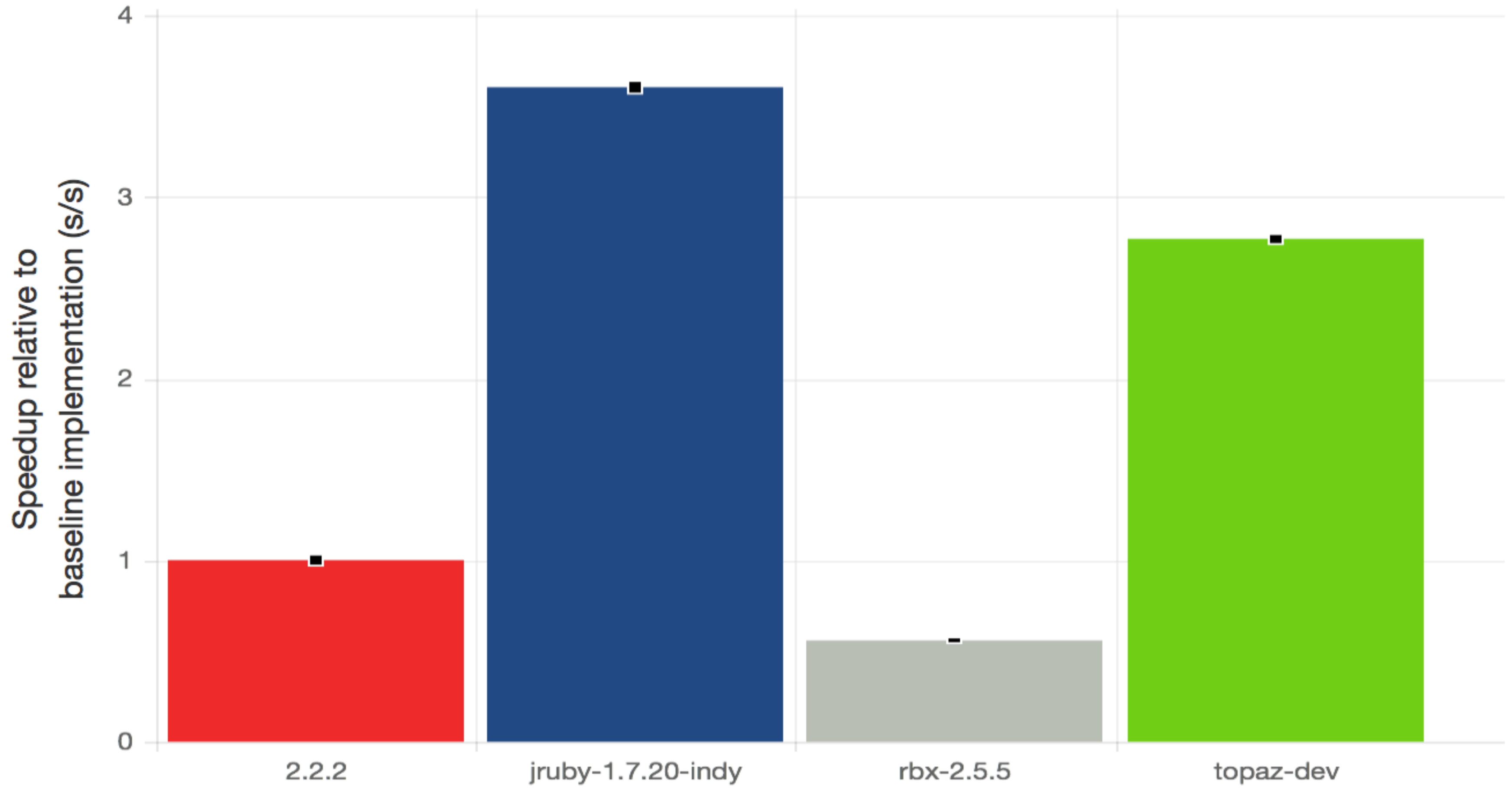
```
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

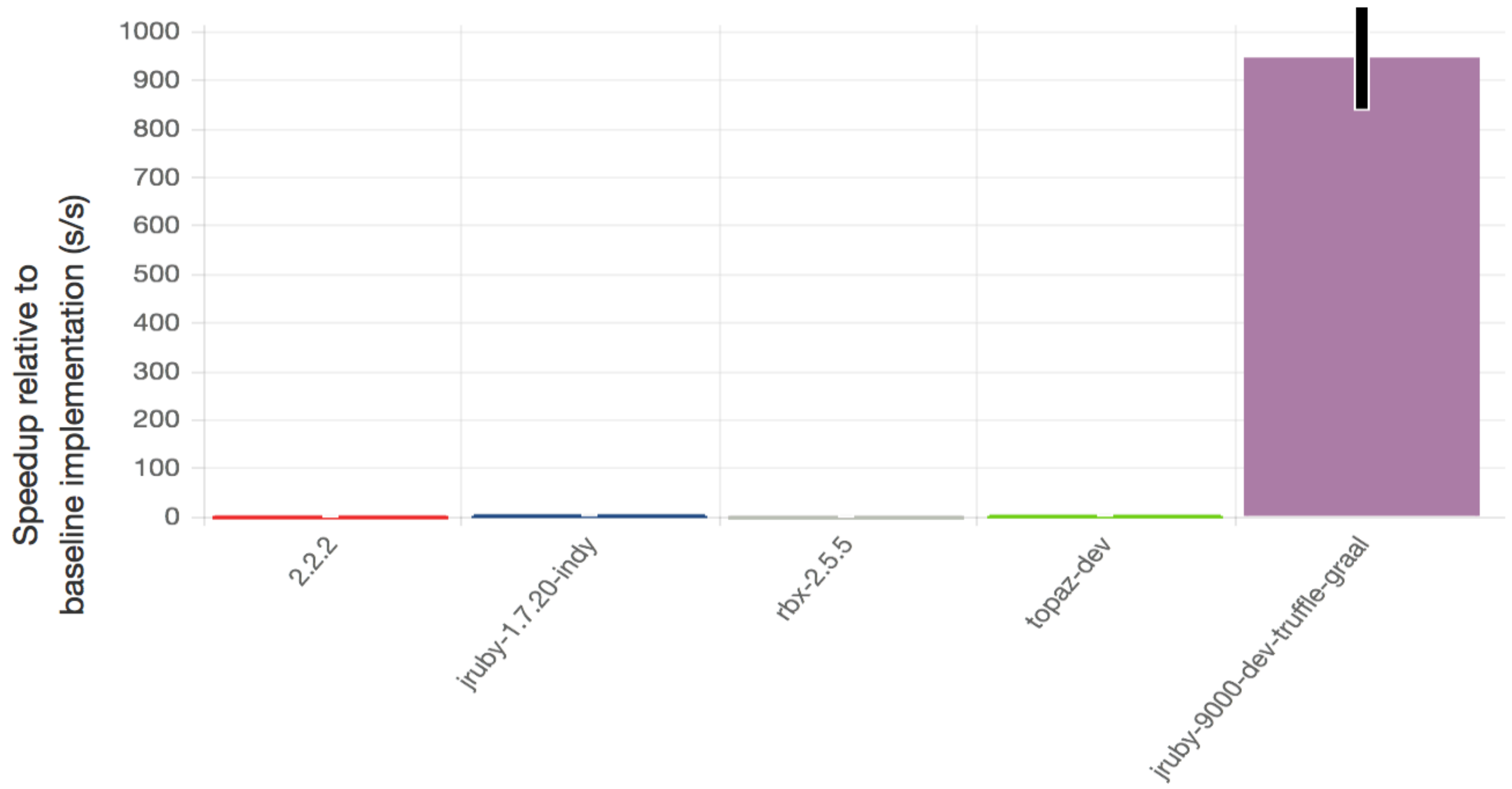


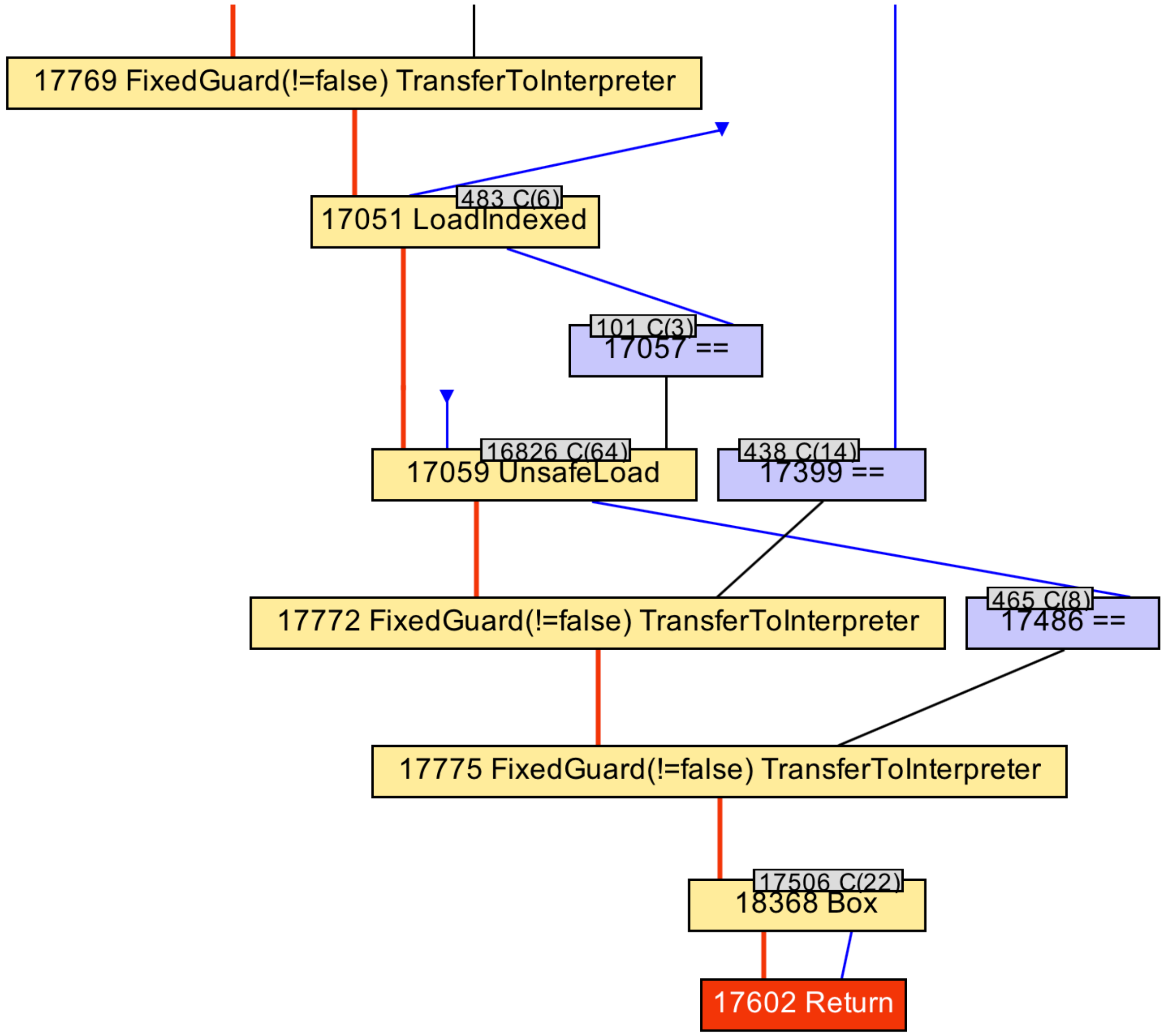
```ruby
module Foo
  def self.foo(a, b, c)
    hash = {a: a, b: b, c: c}
    array = hash.map { |k, v| v }
    x = array[0]
    y = [a, b, c].sort[1]
    x + y
  end
end

class Bar
  def method_missing(method, *args)
    if Foo.respond_to?(method)
      Foo.send(method, *args)
    else
      0
    end
  end
end
```

```ruby
bar = Bar.new

loop do
  start = Time.now
  1_000_000.times do
    bar.foo(14, 8, 6)
  end
  puts Time.now - start
end
```

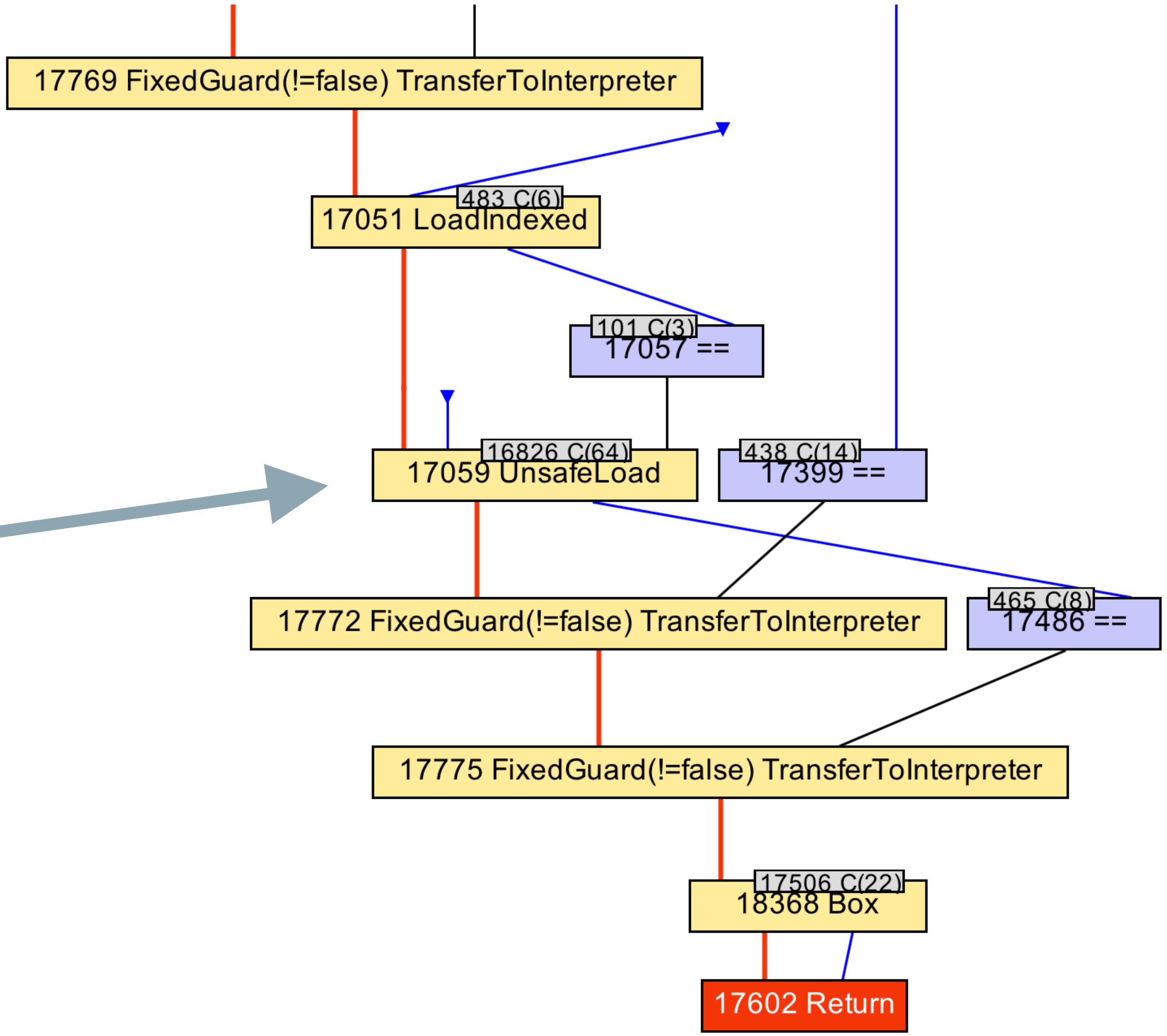
= 22 !

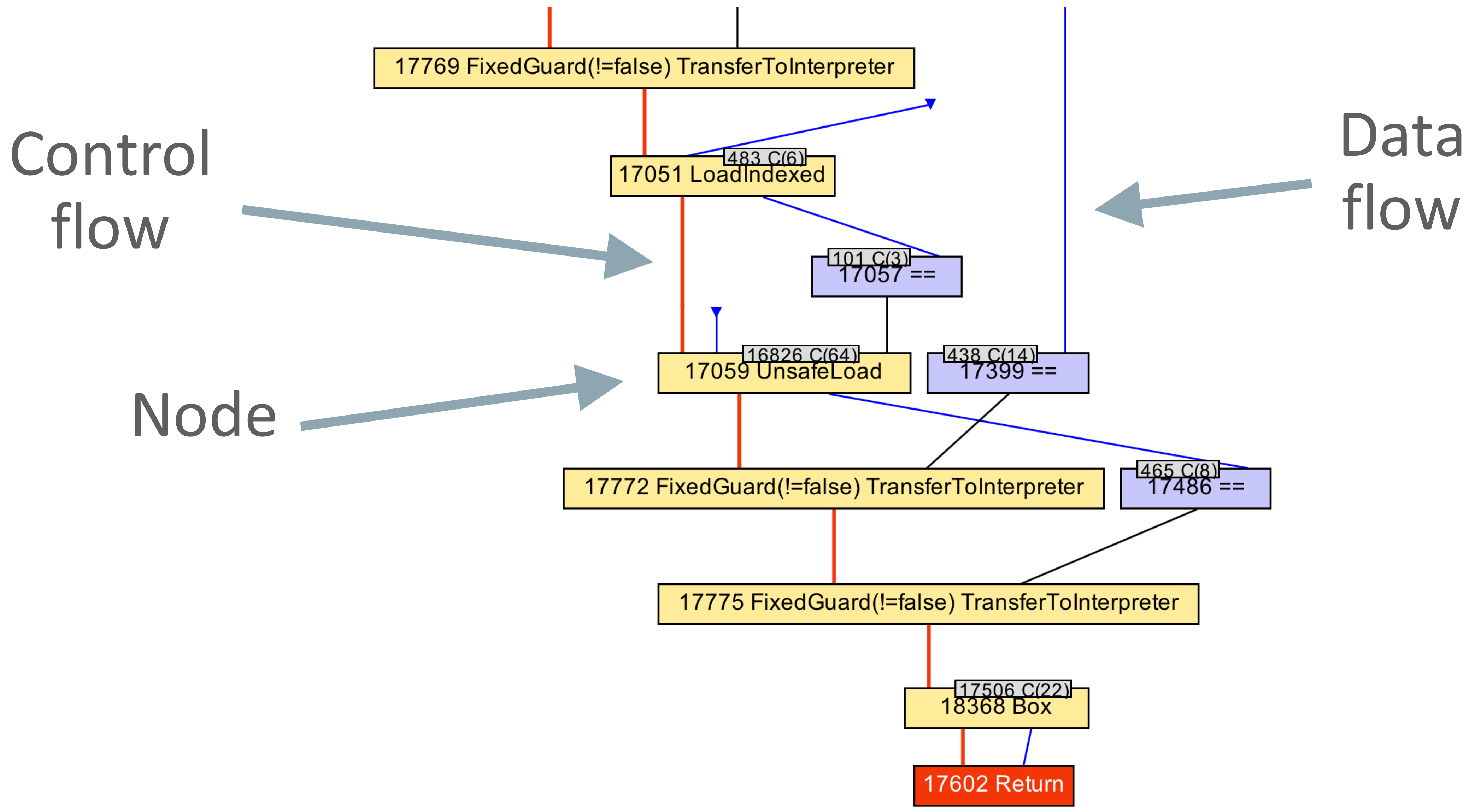

Speedup relative to baseline implementation (s/s)
4
3
2
1
0
2.2.2
jruby-1.7.20-indy
rbx-2.5.5
topaz-dev

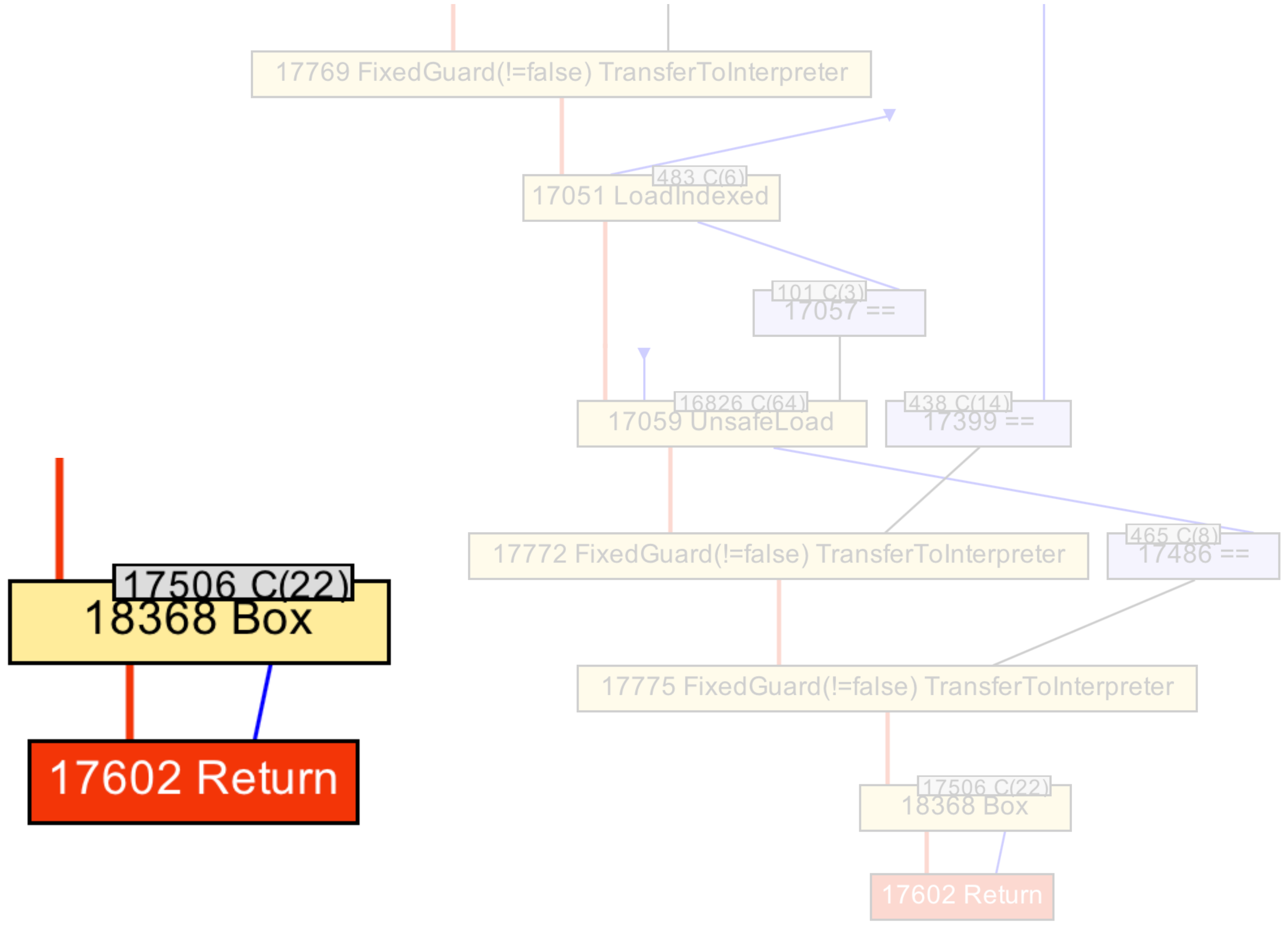

Speedup relative to baseline implementation (s/s)
1000
900
800
700
600
500
400
300
200
100
0
2.2.2
jruby-1.7.20-indy
rbx-2.5.5
topaz-dev
jruby-9000-dev-truffle-graal


17769 FixedGuard(!=false) TransferToInterpreter
483 C(6)
17051 LoadIndexed
101 C(3)
17057 ==
16826 C(64)
17059 UnsafeLoad
438 C(14)
17399 ==
17772 FixedGuard(!=false) TransferToInterpreter
465 C(8)
17486 ==
17775 FixedGuard(!=false) TransferToInterpreter
17506 C(22)
18368 Box
17602 Return

Node


17769 FixedGuard(!=false) TransferToInterpreter
483 C(6)
17051 LoadIndexed
101 C(3)
17057 ==
16826 C(64)
17059 UnsafeLoad
438 C(14)
17399 ==
17772 FixedGuard(!=false) TransferToInterpreter
465 C(8)
17486 ==
17775 FixedGuard(!=false) TransferToInterpreter
17506 C(22)
18368 Box
17602 Return


Control flow
Data flow
Node
17769 FixedGuard(!=false) TransferToInterpreter
483 C(6)
17051 LoadIndexed
101 C(3)
17057 ==
16826 C(64)
17059 UnsafeLoad
438 C(14)
17399 ==
17772 FixedGuard(!=false) TransferToInterpreter
465 C(8)
17486 ==
17775 FixedGuard(!=false) TransferToInterpreter
17506 C(22)
18368 Box
17602 Return


17769 FixedGuard(!=false) TransferToInterpreter
483 C(6)
17051 LoadIndexed
101 C(3)
17057 ==
16826 C(64)
17059 UnsafeLoad
438 C(14)
17399 ==
17772 FixedGuard(!=false) TransferToInterpreter
465 C(8)
17486 ==
17775 FixedGuard(!=false) TransferToInterpreter
17506 C(22)
18368 Box
17602 Return
17506 C(22)
18368 Box
17602 Return

```
...
movabs 0x11e2037a8, %rax ; {oop(a 'java/lang/Integer' = 22)}
...
retq
```

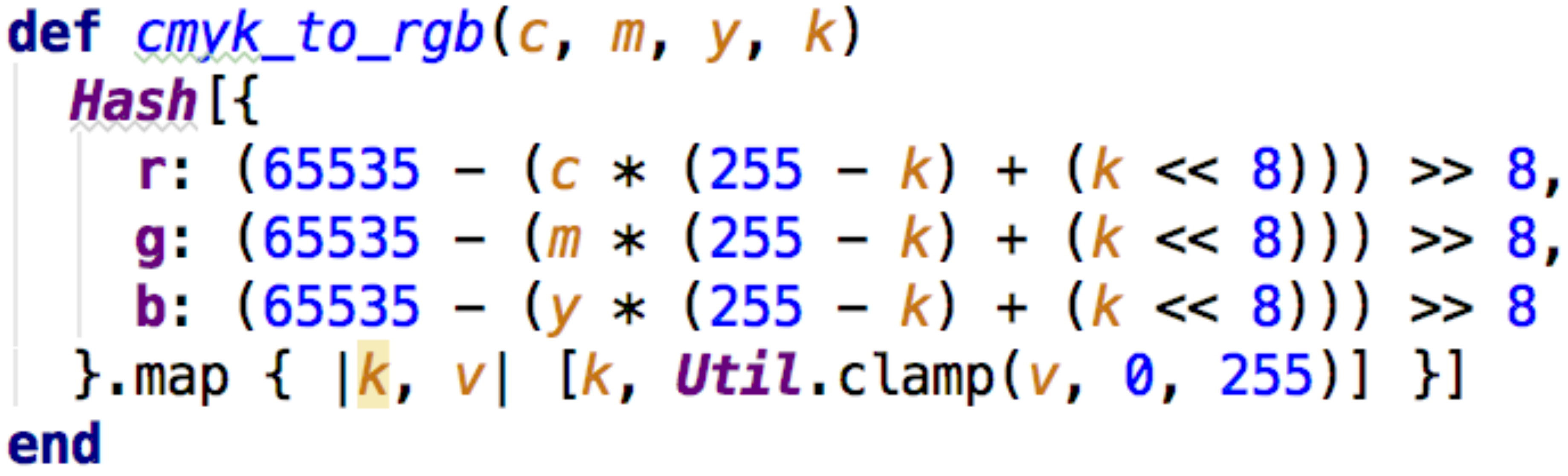
C extensions

C extensions are a hack to workaround performance, but now they stop us really fixing performance

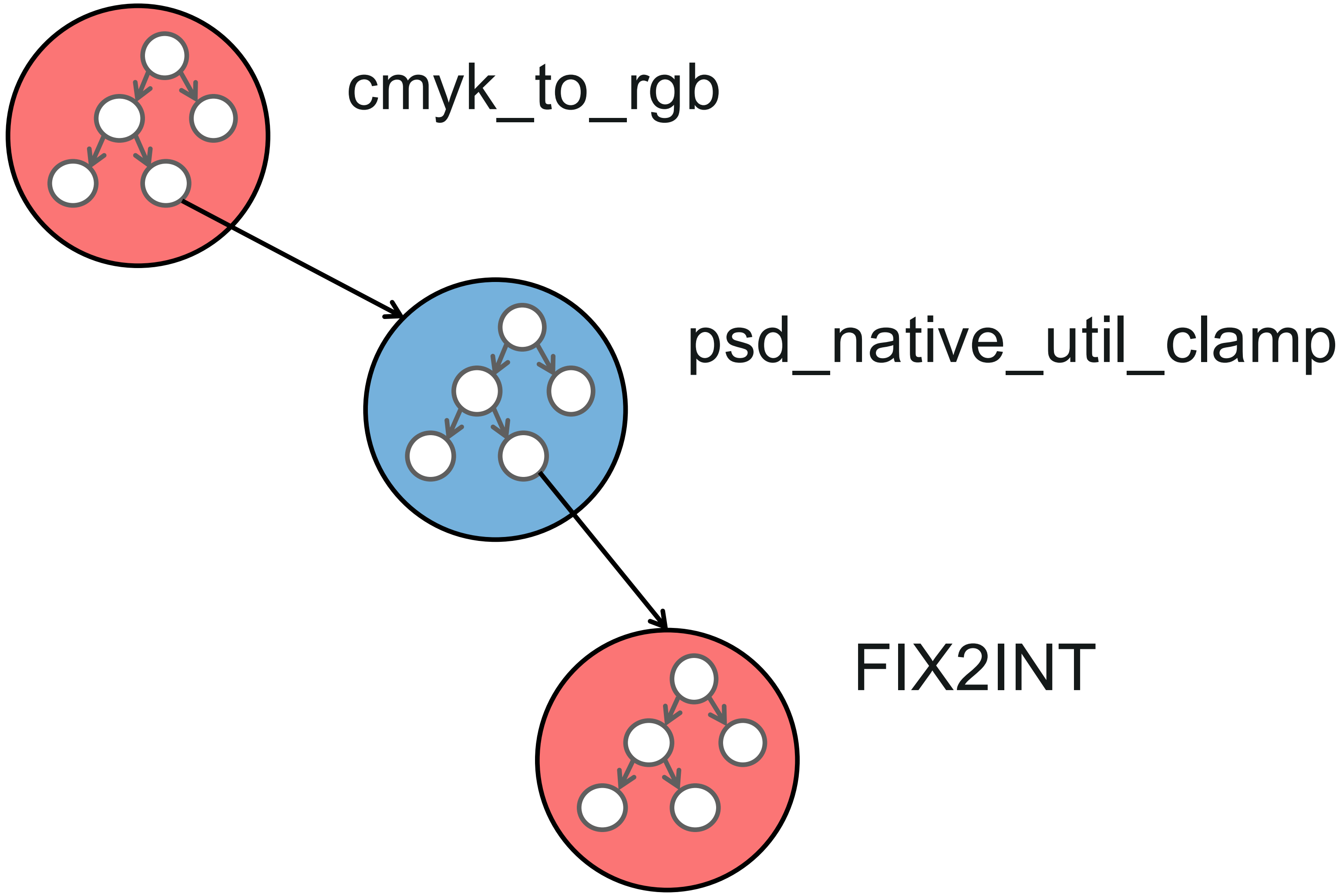
A lot of this has been about removing barriers to the excellent optimisations we already have

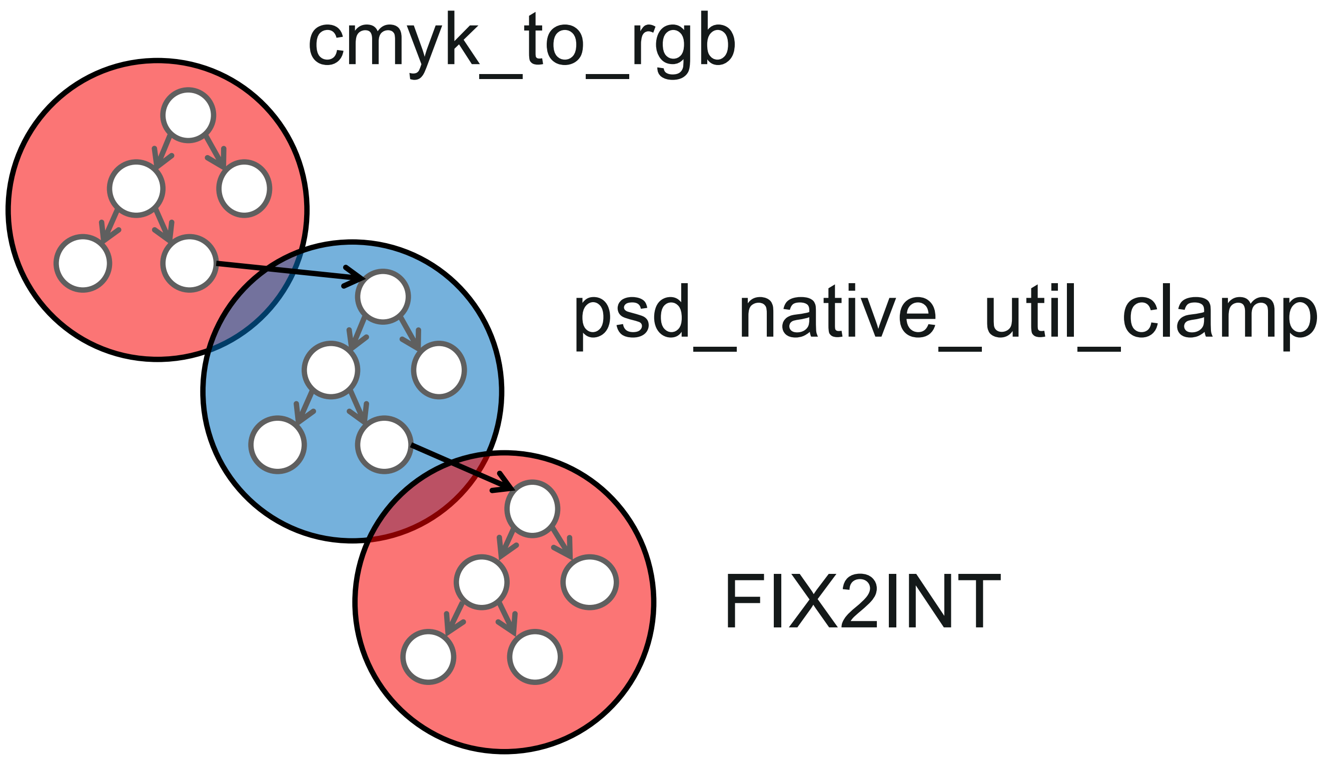
```
def clamp(num, min, max)
  [min, num, max].sort[1]
end
```

```c
VALUE psd_native_util_clamp(VALUE self, VALUE r_num, VALUE r_min, VALUE r_max) {
  int num = FIX2INT(r_num);
  int min = FIX2INT(r_min);
  int max = FIX2INT(r_max);

  return num > max ? r_max : (num < min ? r_min : r_num);
}
```

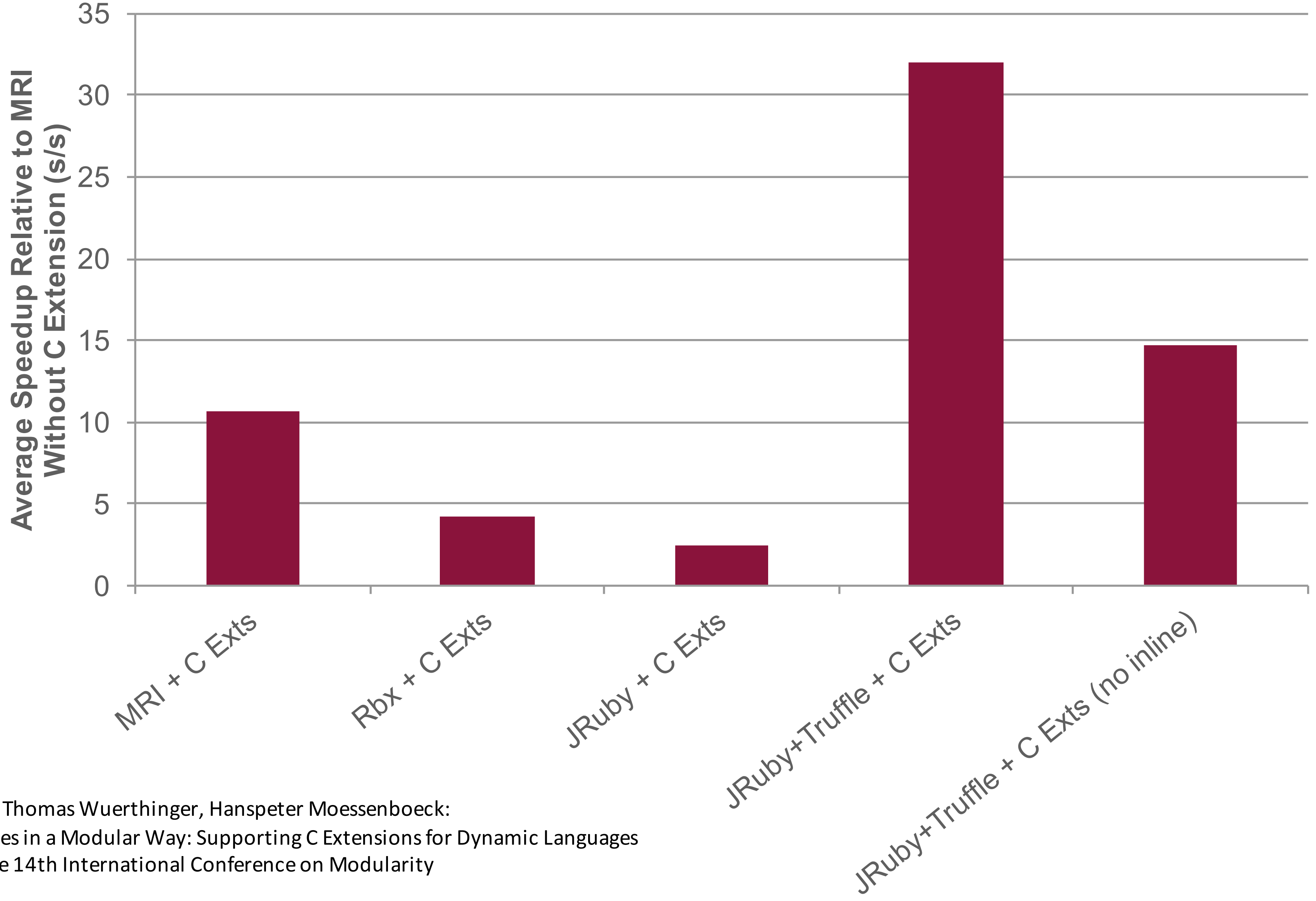


```ruby
def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  }.map { |k, v| [k, Util.clamp(v, 0, 255)] }]
end
```


cmyk_to_rgb
psd_native_util_clamp
FIX2INT


cmyk_to_rgb
psd_native_util_clamp
FIX2INT

C Extension Performance for psd_native and oily_png



Matthias Grimmer, Chris Seaton, Thomas Wuerthinger, Hanspeter Moessenboeck:
Dynamically Composing Languages in a Modular Way: Supporting C Extensions for Dynamic Languages
Modularity '14 Proceedings of the 14th International Conference on Modularity

Conclusions

The blocker for performance of idiomatic Ruby code is the core library, not basic language features

This extends to everything that forms a barrier – including C extensions

Specialisation, splitting, inlining, partial evaluation, inline caching are all solutions to this problem

Truffle makes it easy to add these to a language implementation

Can result in an order of magnitude performance increase with reasonable effort

ORACLE

Acknowledgements

Benoit Daloze
Kevin Menard
Petr Chalupa

Oracle Labs
Danilo Ansaloni
Stefan Anzinger
Daniele Bonetta
Matthias Brantner
Laurent Daynès
Gilles Duboscq
Michael Haupt
Christian Humer
Mick Jordan
Peter Kessler
Hyunjin Lee
David Leibs
Tom Rodriguez
Roland Schatz
Chris Seaton
Doug Simon
Lukas Stadler

Oracle Labs (continued)
Michael Van de Vanter
Adam Welc
Till Westmann
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger

Oracle Labs Interns
Shams Imam
Stephen Kell
Gero Leinemann
Julian Lettner
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Oracle Labs Alumni
Erik Eckstein
Christos Kotselidi

JKU Linz
Prof. Hanspeter Mössenböck
Josef Eisl
Thomas Feichtinger
Matthias Grimmer
Christian Häub
Josef Haider
Christian Hube
David Leopoltsederr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

University of Edinburgh
Christophe Dubach
Juan José Fumero Alfonso Ranjeet Singh
Toomas Remmelg

LaBRI
Floréal Morandat

University of California, Irvine
Prof. Michael Franz
Codrut Stancu
Gulfem Savrun Yeniceri
Wei Zhang

Purdue University
Prof. Jan Vitek
Tomas Kalibera
Romand Tsegelskyi
Prahlad Joshi
Petr Maj Lei Zhao

T. U. Dortmund
Prof. Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis
Prof. Duncan Temple Lang
Nicholas Ulle

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®

Integrated Cloud

Applications & Platform Services