
ORACLE®

Using LLVM and Sulong for Language C Extensions

Chris Seaton
Research Manager
VM Research Group
Oracle Labs

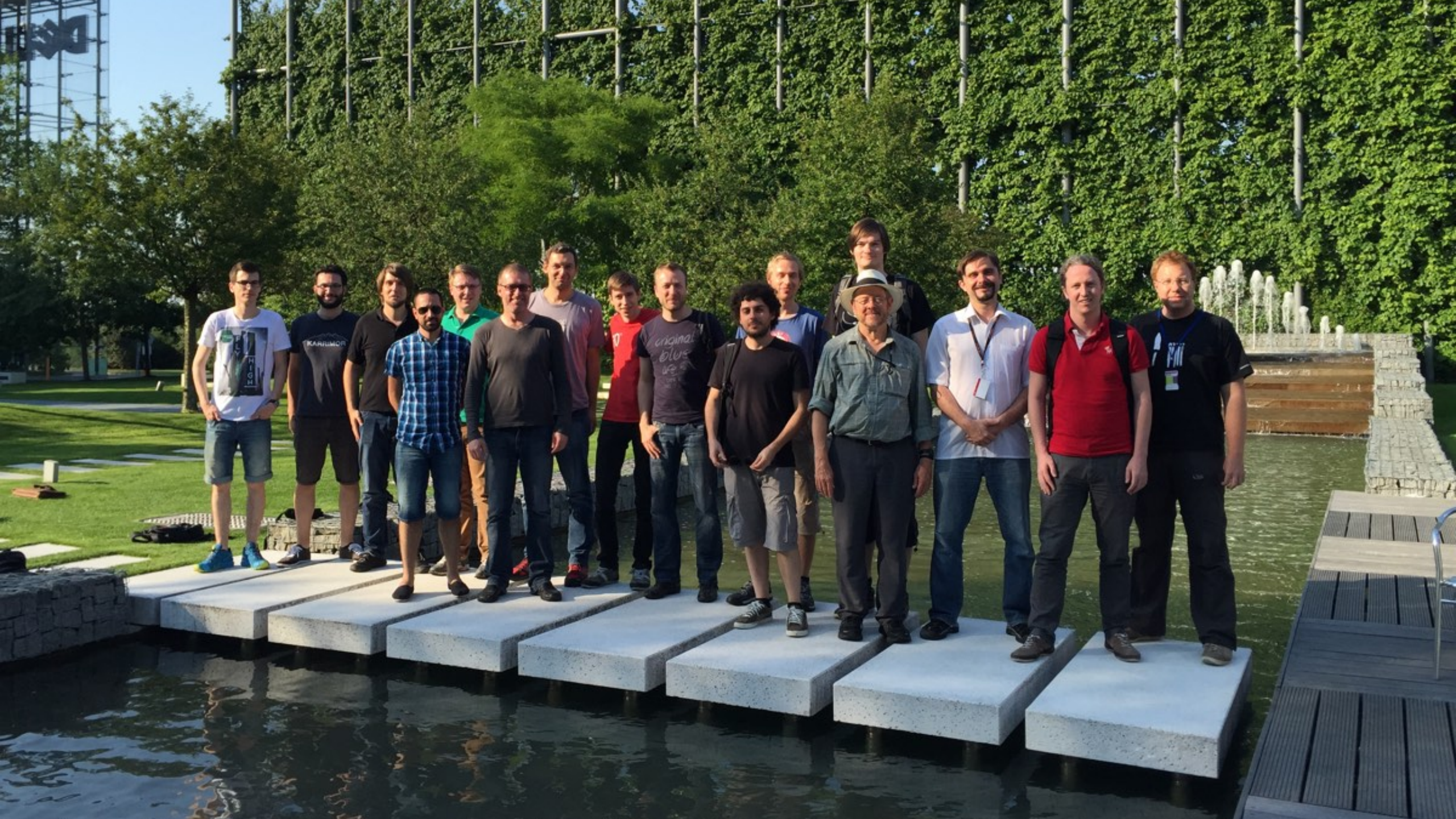
Manuel Rigger, Matthias Grimmer
JKU Linz

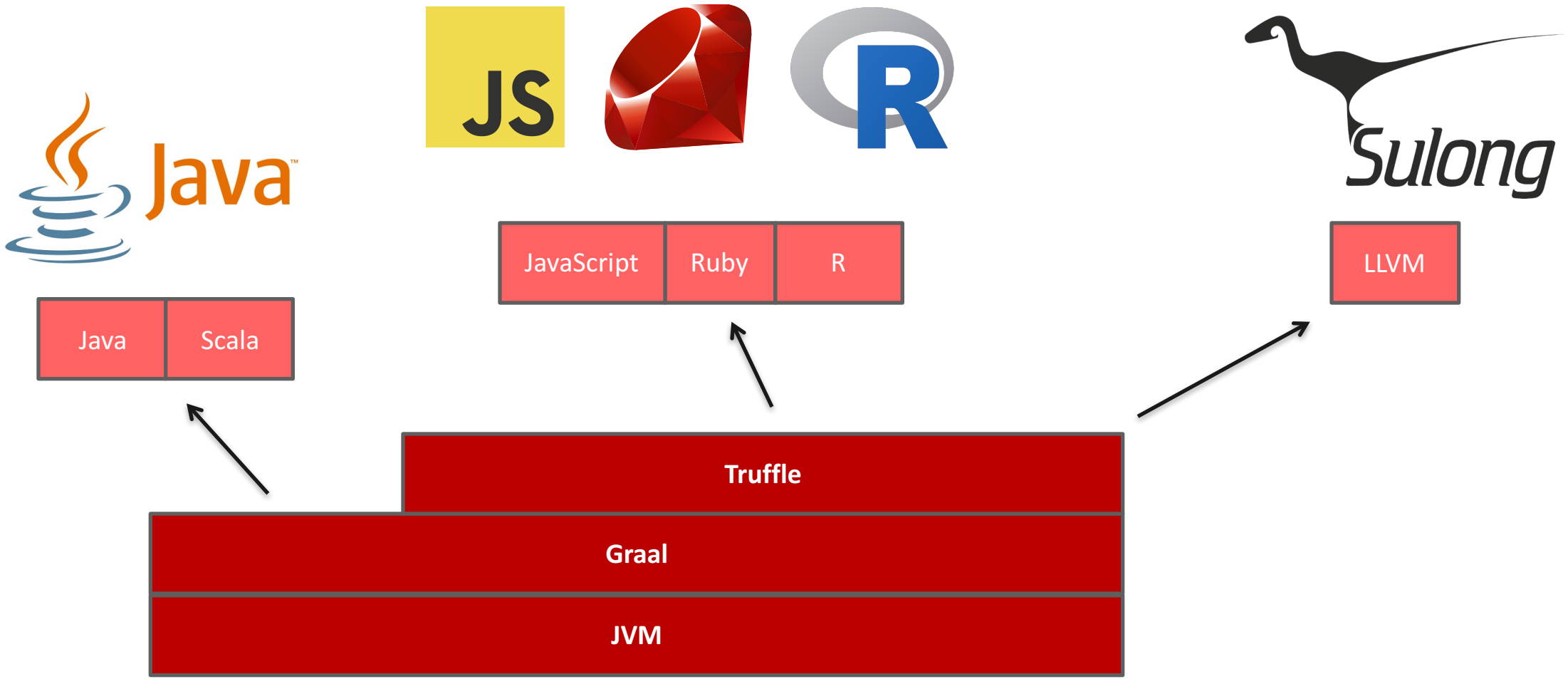
ORACLE

Safe Harbor Statement

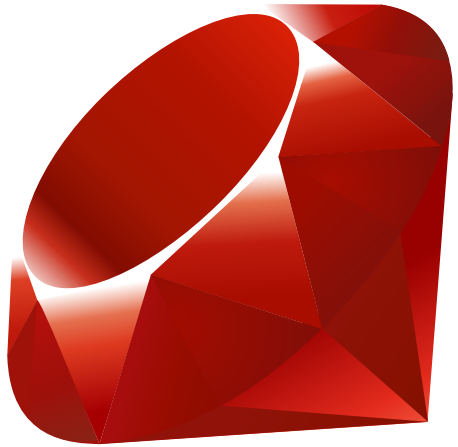
The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing 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.

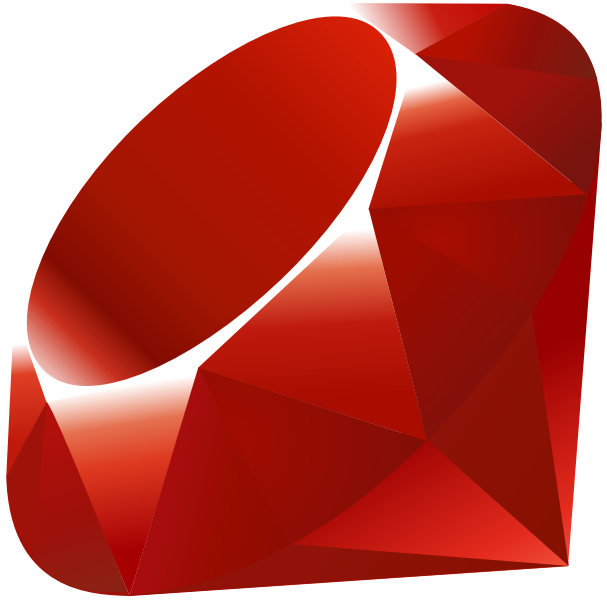
Who we are and what we're doing





Language C extensions



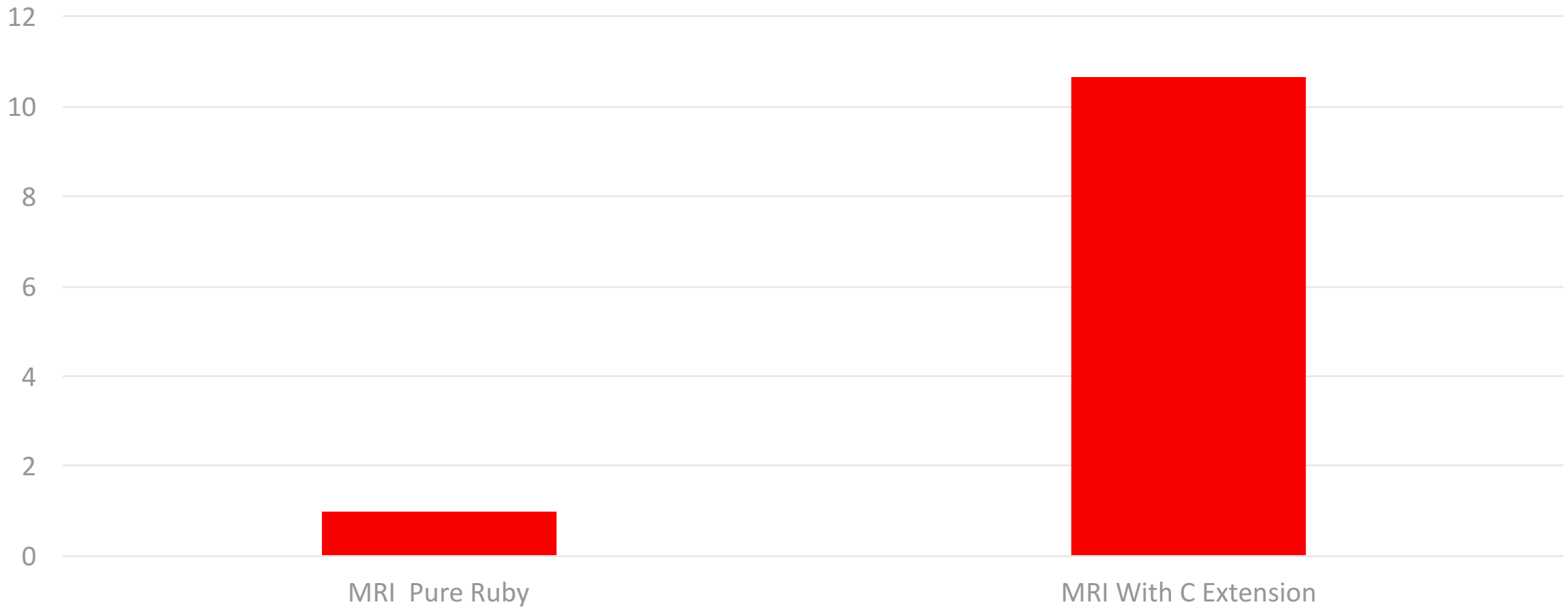


```ruby
def clamp(num, min, max)
  [min, num, max].sort[1]
end
```

```
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);
}
```

Performance on Ruby C Extensions Oily PNG and PSD Native



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.

The C extension problem

```
struct RString {
  struct RBasic basic;
  union {
    struct {
      long len;
      char *ptr;
      union {
        long capa;
        VALUE shared;
      } aux;
    } heap;
    char ary[RSTRING_EMBED_LEN_MAX + 1];
  } as;
};
```

Structs declared in the public API

Implementation details like embedded strings and sharing exposed

```c
static VALUE
ossl_rand_bytes(VALUE self, VALUE len)
{
    VALUE str;
    int n = NUM2INT(len);
    int ret;

    str = rb_str_new(0, n);
    ret = RAND_bytes((unsigned char *)RSTRING_PTR(str), n);
    if (ret == 0) {
       ossl_raise(eRandomError, "RAND_bytes");
    } else if (ret == -1) {
       ossl_raise(eRandomError, "RAND_bytes is not supported");
    }

    return str;
}
```

Ruby String object

Managed Ruby value on the C stack

Exposes inner char*

Call to arbitrary C code

String object with char* already exposed now returned to Ruby

```ruby
(0...@num_pixels).step(pixel_step) do |i|
  .....
  rgb = PSD::Color.cmyk_to_rgb(255 - c, 255 - m, 255 - y, 255 - k)
    .....
end
```

```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
```

Call from Ruby to native is extremely hot

Values need to be converted as they go from Ruby to native

```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);
}
```

Array implementation pointer taken and stored for later

```
VALUE* bg_pixels = RARRAY_PTR(rb_funcall(self, rb_intern("pixels"), 0));
VALUE* fg_pixels = RARRAY_PTR(rb_funcall(other, rb_intern("pixels"), 0));

long x = 0;
long y = 0;
for( y = 0; y < other_height; y++ ){
  for( x = 0; x < other_width; x++ ){
    bg_index = ( x + offset_x ) + ( y + offset_y ) * self_width;
    bg_pixels[bg_index] = UINT2NUM(
      oily_png_compose_color(
        NUM2UINT( fg_pixels[x+ y * other_width] ),
        NUM2UINT( bg_pixels[bg_index] ) ) );
  }
}
```

When they're used there's no indication someone else is managing them

Previous solutions

```
bool
RubyString::jsync(JNIEnv* env)
{
    if (rwdata.readonly && rwdata.rstring != NULL) {
        // Don't sync anything, just clear the cached data
        rwdata.rstring = NULL;
        rwdata.readonly = false;

        return false;
    }

    if (rwdata.rstring != NULL && rwdata.rstring->ptr != NULL) {
        jobject byteList = env->GetObjectField(obj, RubyString_value_field);
        jobject bytes = env->GetObjectField(byteList, ByteList_bytes_field);
        jint begin = env->GetIntField(byteList, ByteList_begin_field);
        checkExceptions(env);

        env->DeleteLocalRef(byteList);

        RString* rstring = rwdata.rstring;
        env->SetByteArrayRegion((jbyteArray) bytes, begin, rstring->len,
                (jbyte *) rstring->ptr);
        checkExceptions(env);
        env->SetIntField(byteList, ByteList_length_field, rstring->len);

        env->DeleteLocalRef(bytes);
    }

    return true;
}
```

Copy

```
bool
RubyString::nsync(JNIEnv* env)
{
    jobject byteList = env->GetObjectField(obj, RubyString_value_field);
    checkExceptions(env);
    jobject bytes = env->GetObjectField(byteList, ByteList_bytes_field);
    checkExceptions(env);
    jint begin = env->GetIntField(byteList, ByteList_begin_field);
    checkExceptions(env);
    long length = env->GetIntField(byteList, ByteList_length_field);
    checkExceptions(env);
    jint capacity = env->GetArrayLength((jarray) bytes) - begin;
    checkExceptions(env);
    env->DeleteLocalRef(byteList);

    RString* rstring = rwdata.rstring;

    if ((capacity > rstring->capa) || (rstring->capa == 0)) {
        rstring->capa = capacity;
        rstring->ptr = (char *) realloc(rstring->ptr, rstring->capa + 1);
    }

    env->GetByteArrayRegion((jbyteArray) bytes, begin, length,
            (jbyte *) rstring->ptr);
    checkExceptions(env);
    env->DeleteLocalRef(bytes);

    rstring->ptr[rstring->len = length] = 0;

    return true;
}
```

Copy

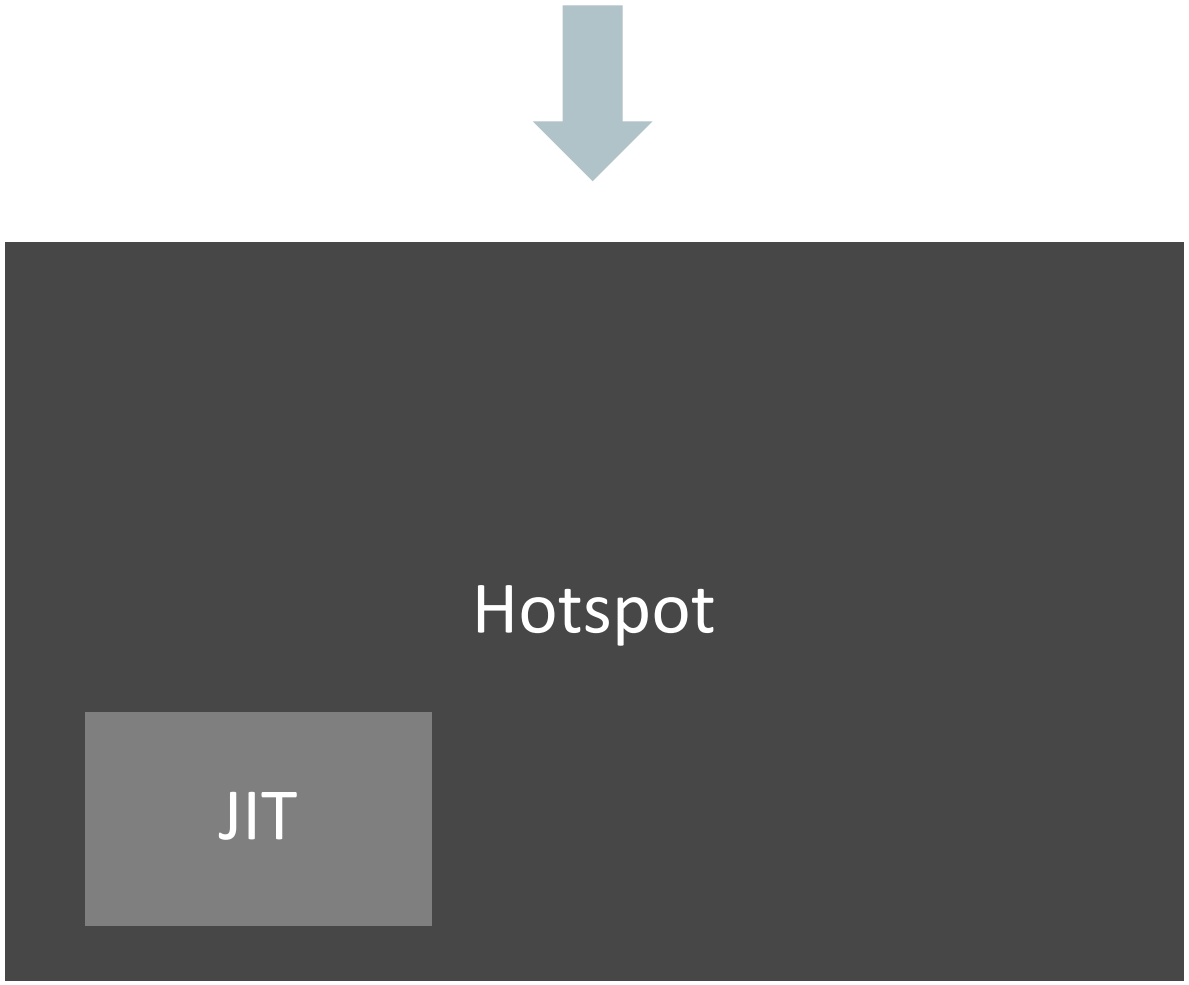
Our new solution

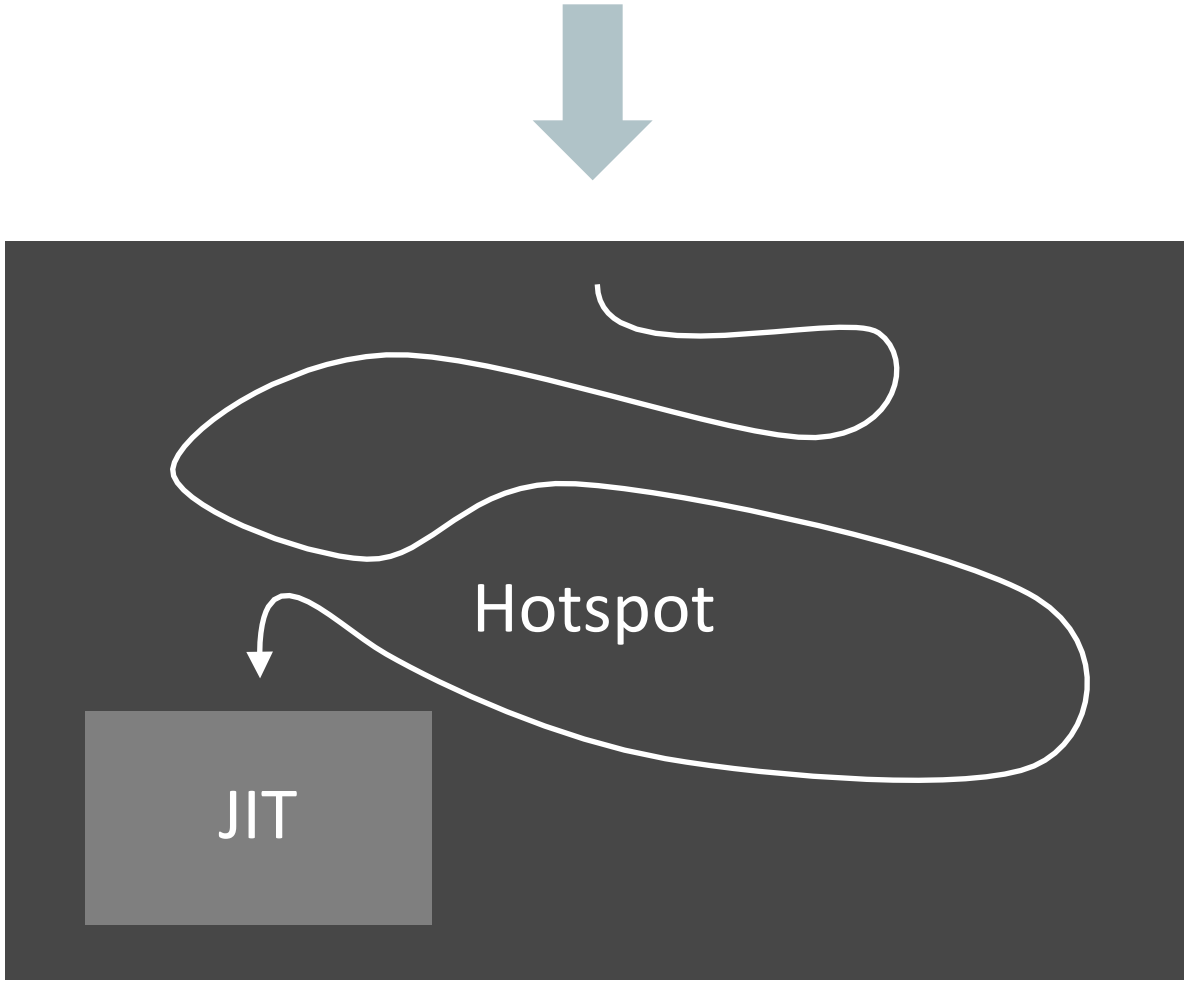
- Interpret both the Ruby and the C
- Actually, interpret the LLVM IR of the C to simplify
- JIT compile the Ruby and the C
- Use a single high and low level IR for both
- Forget which language the IR came from and optimise them together
- Give virtualised pointers to the C program

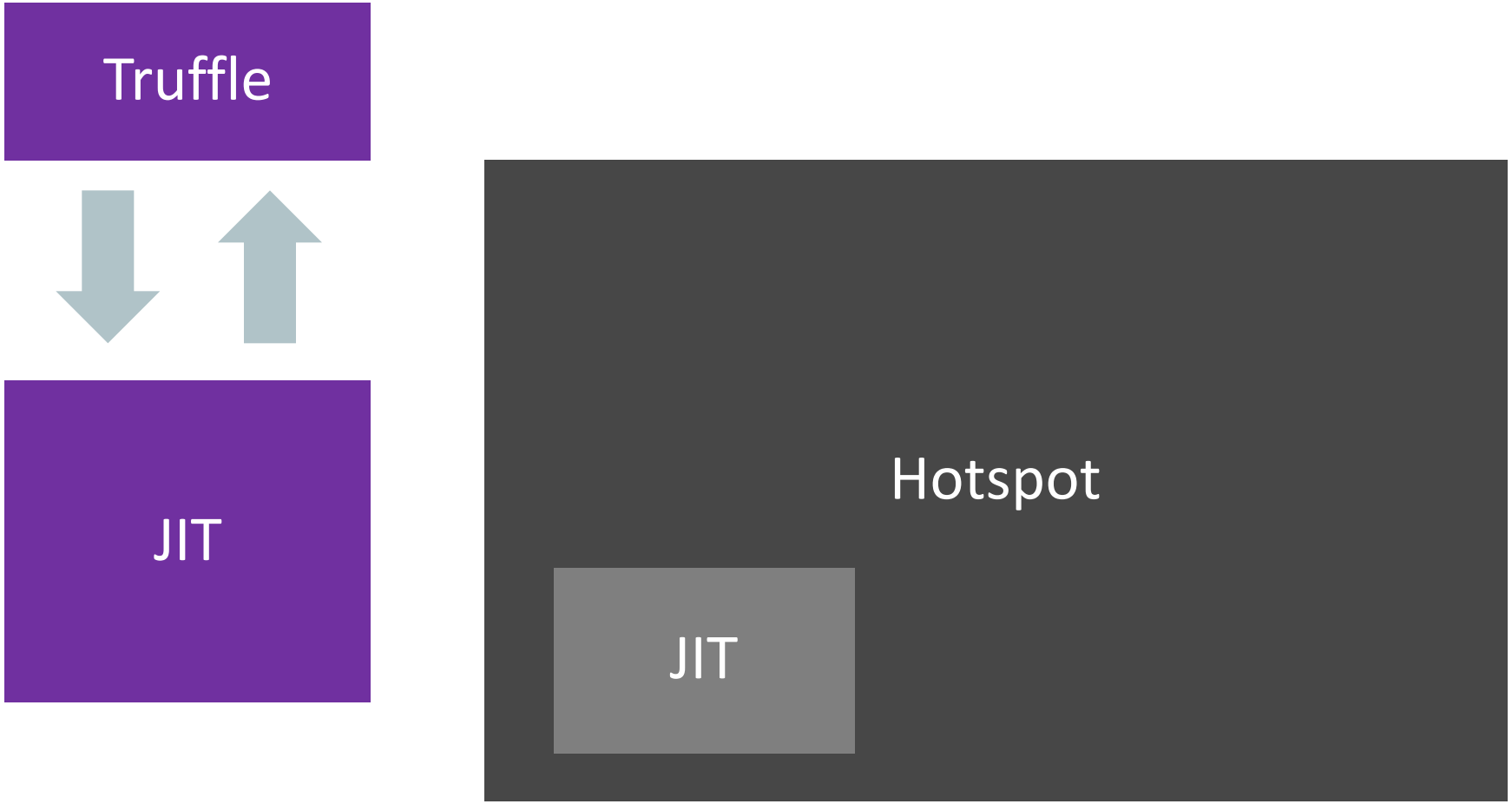
How Sulong and JRuby+Truffle work

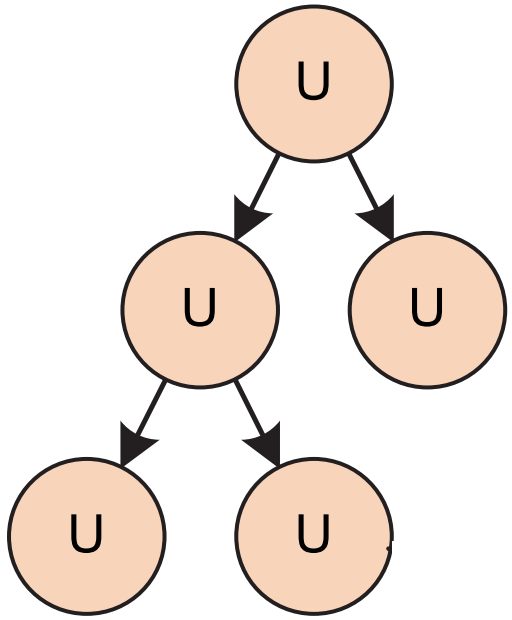
Hotspot

Hotspot



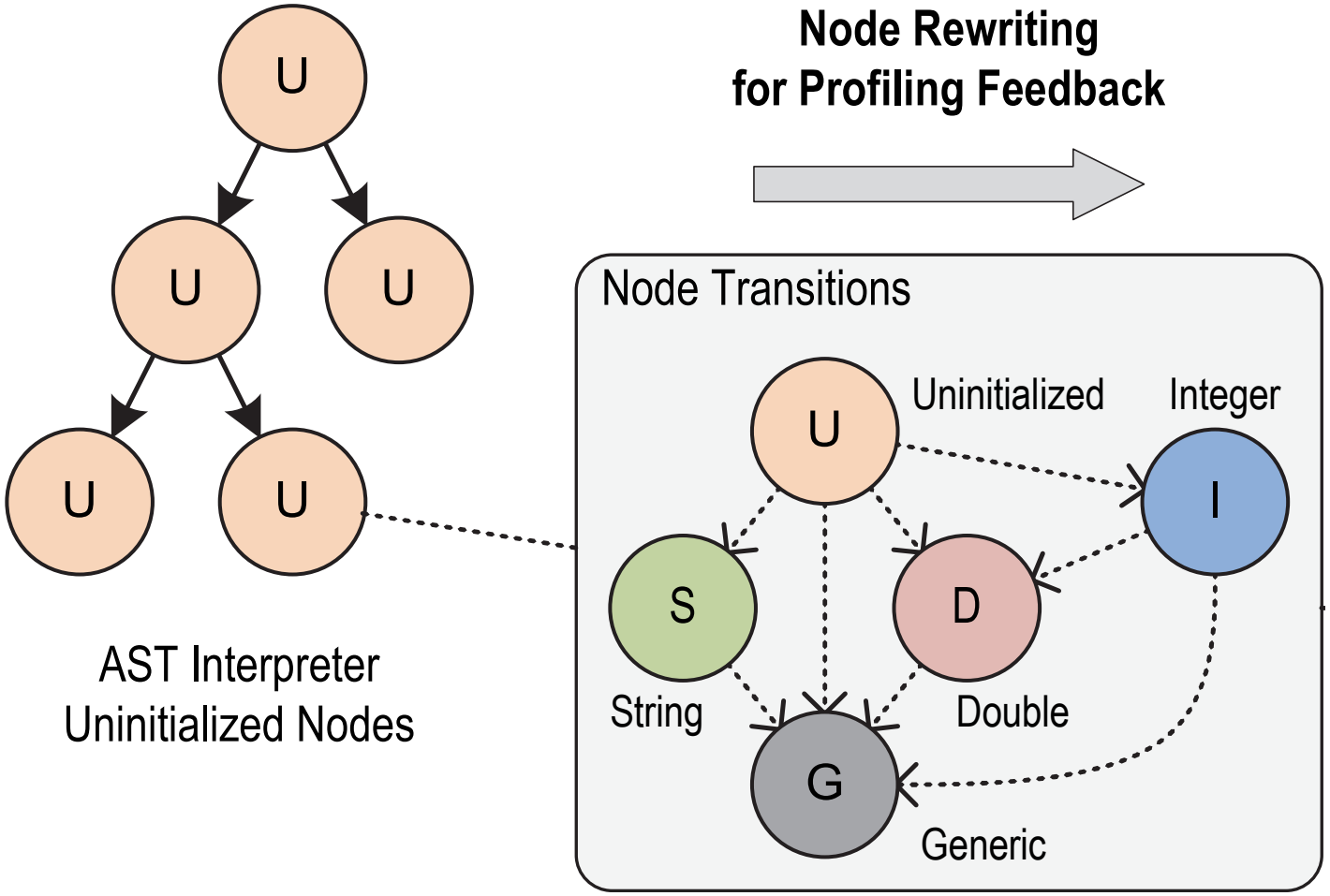




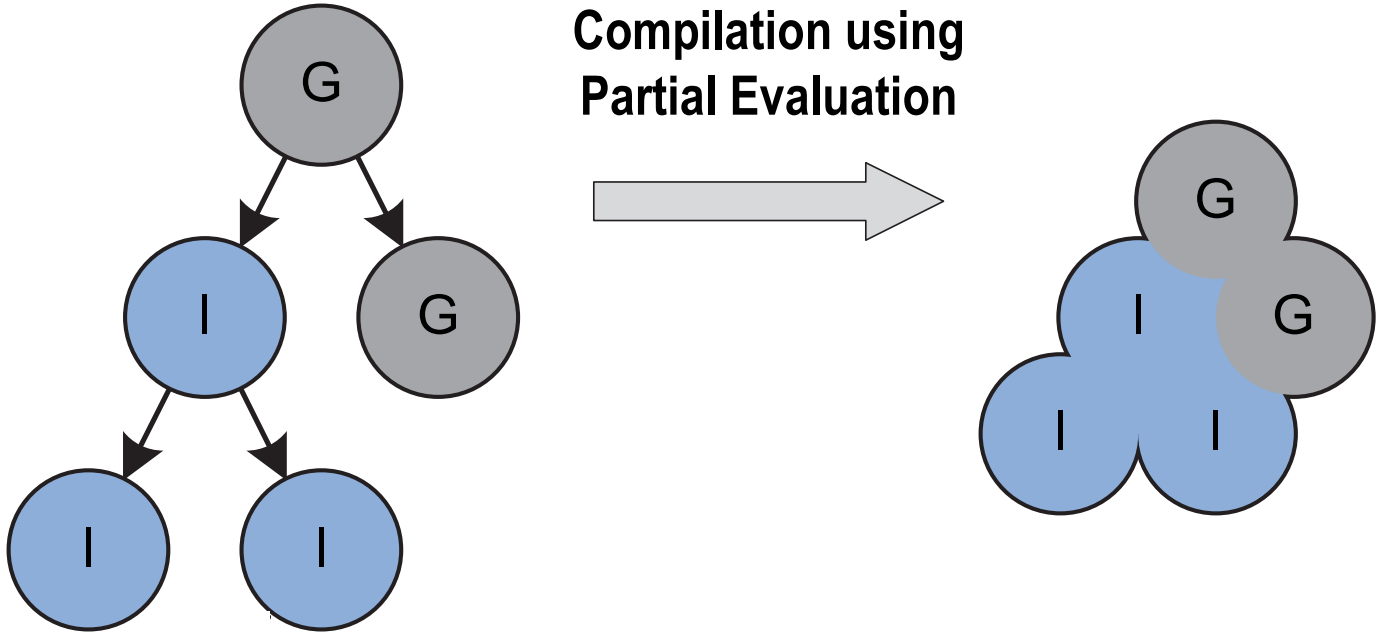


AST Interpreter
Uninitialized Nodes

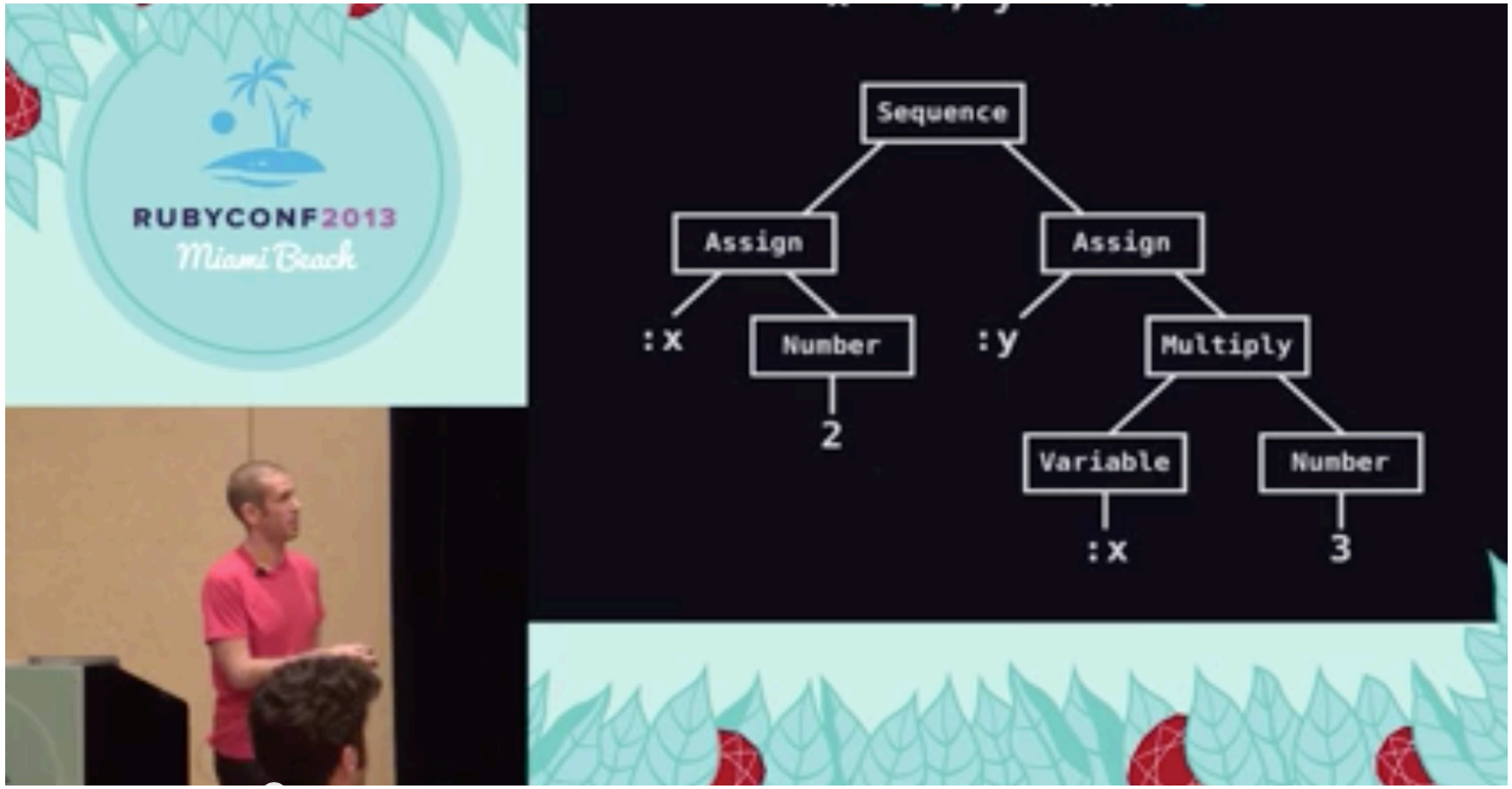
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



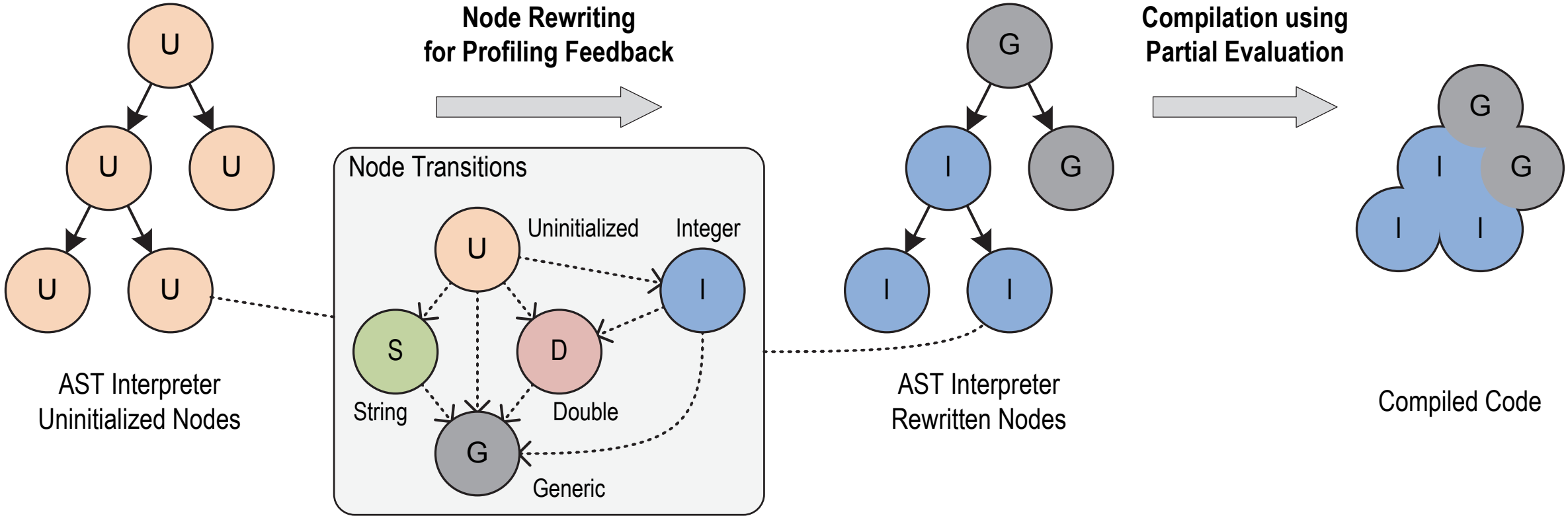
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



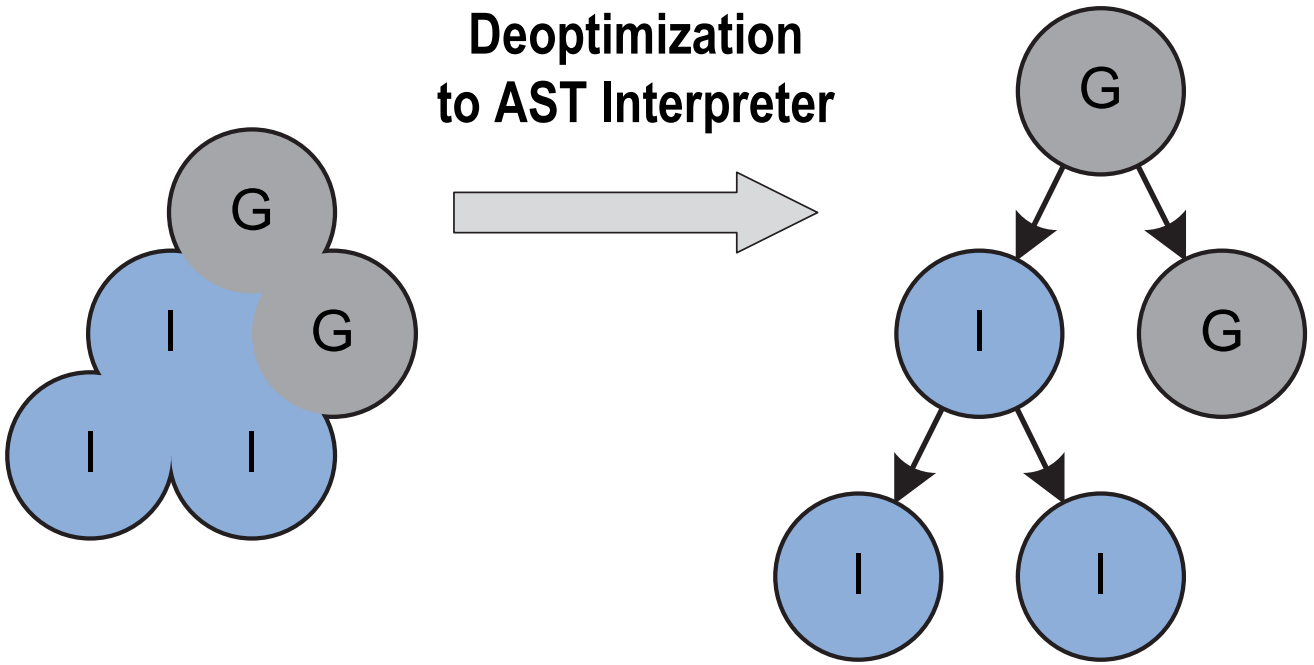
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



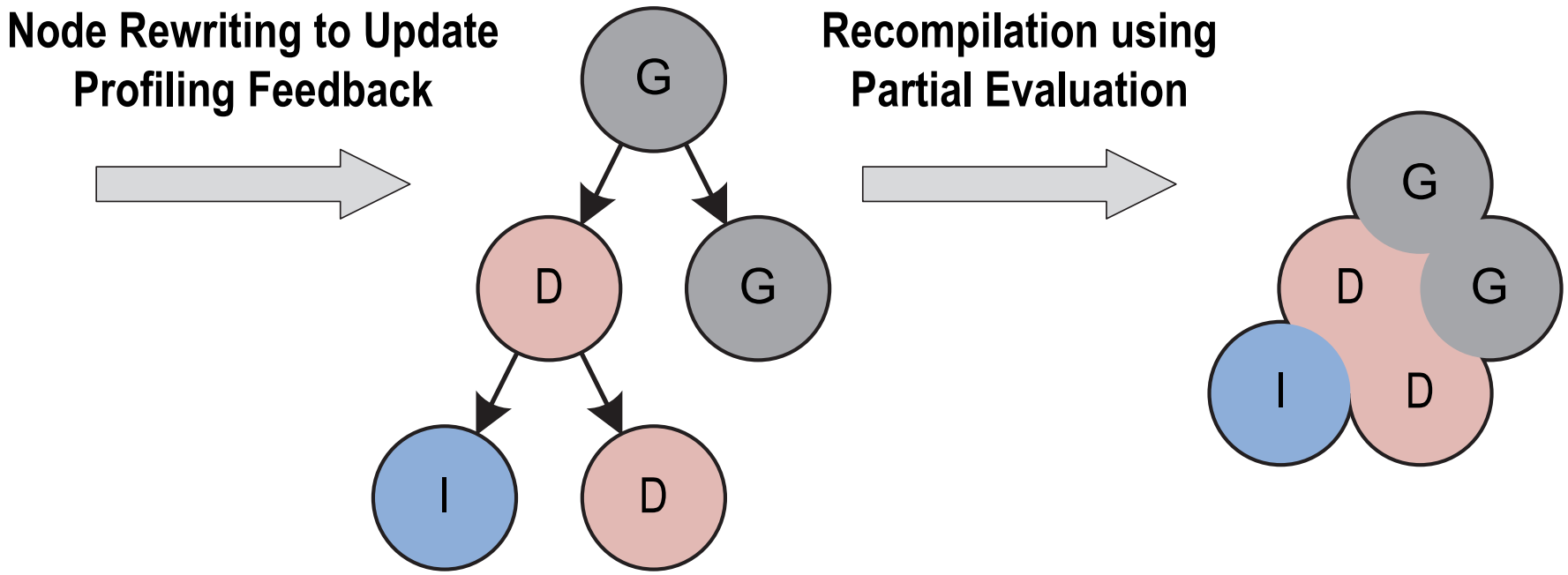
codon.com/compilers-for-free



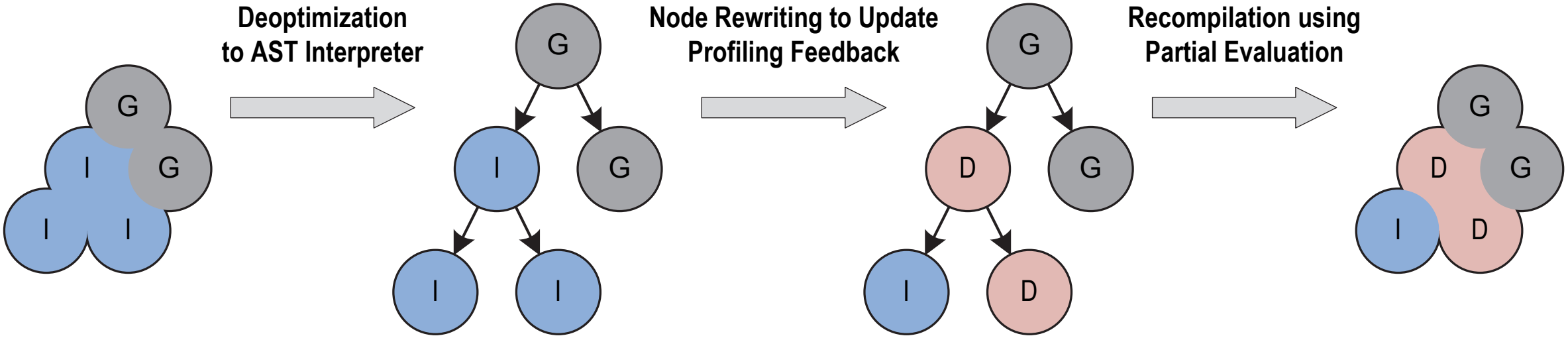
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



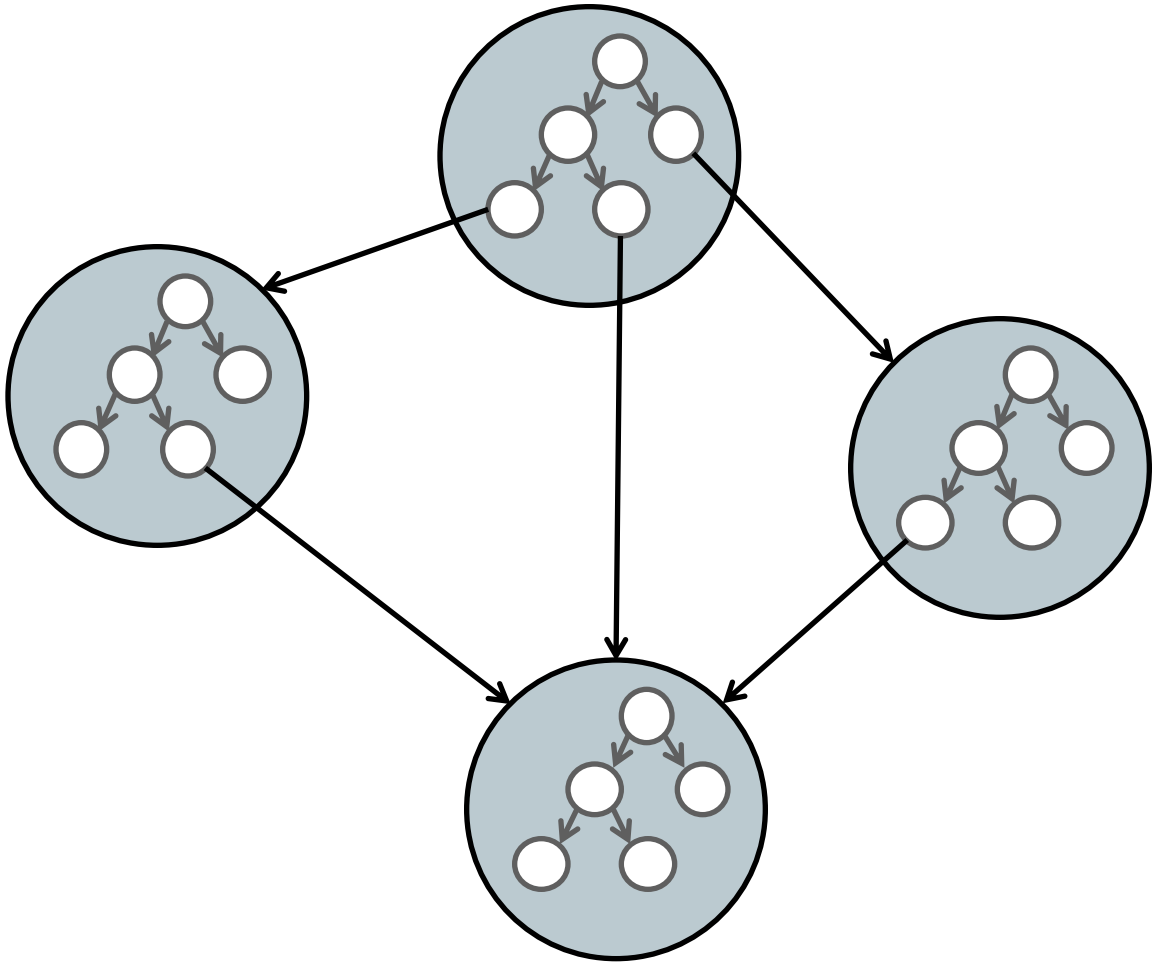
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

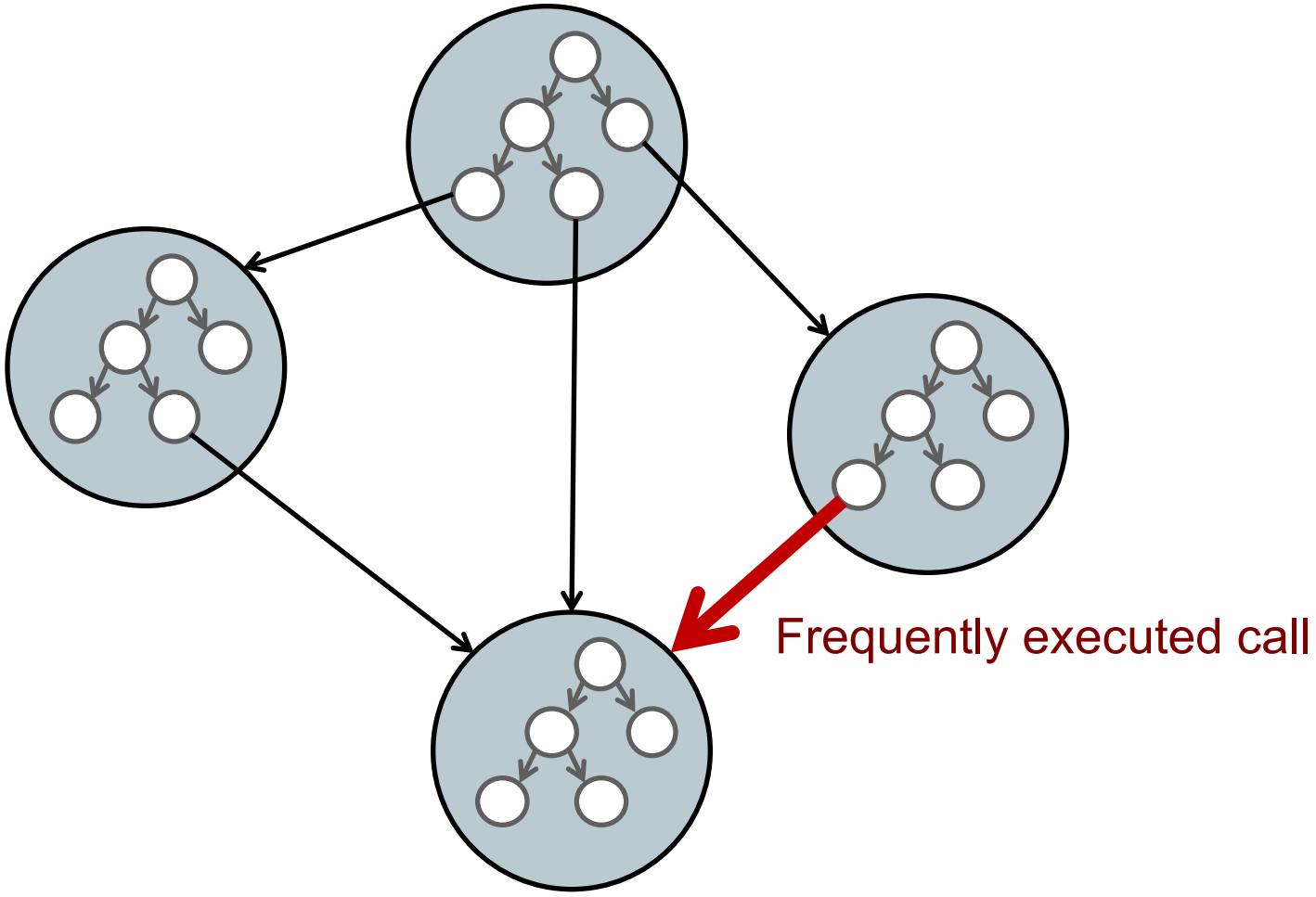


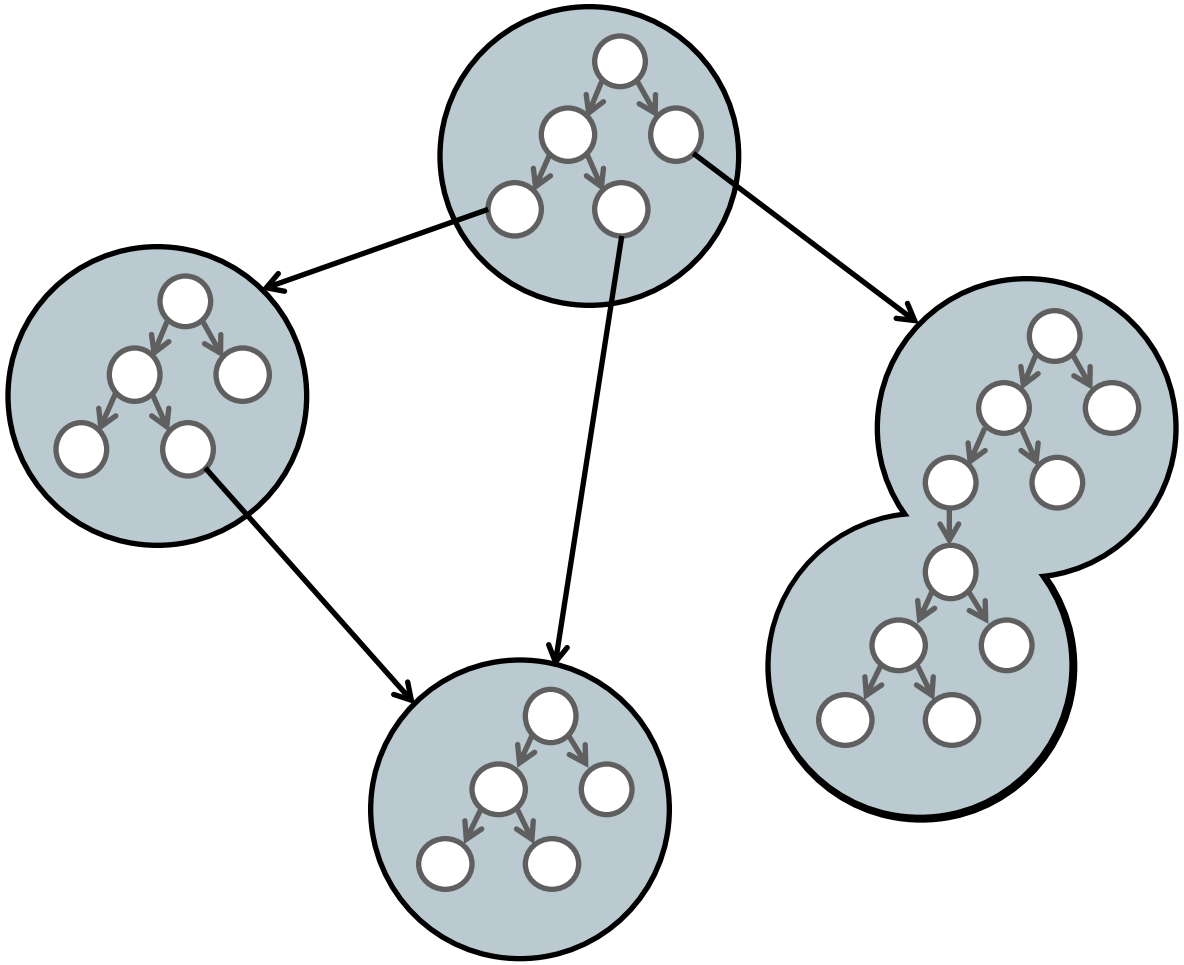
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

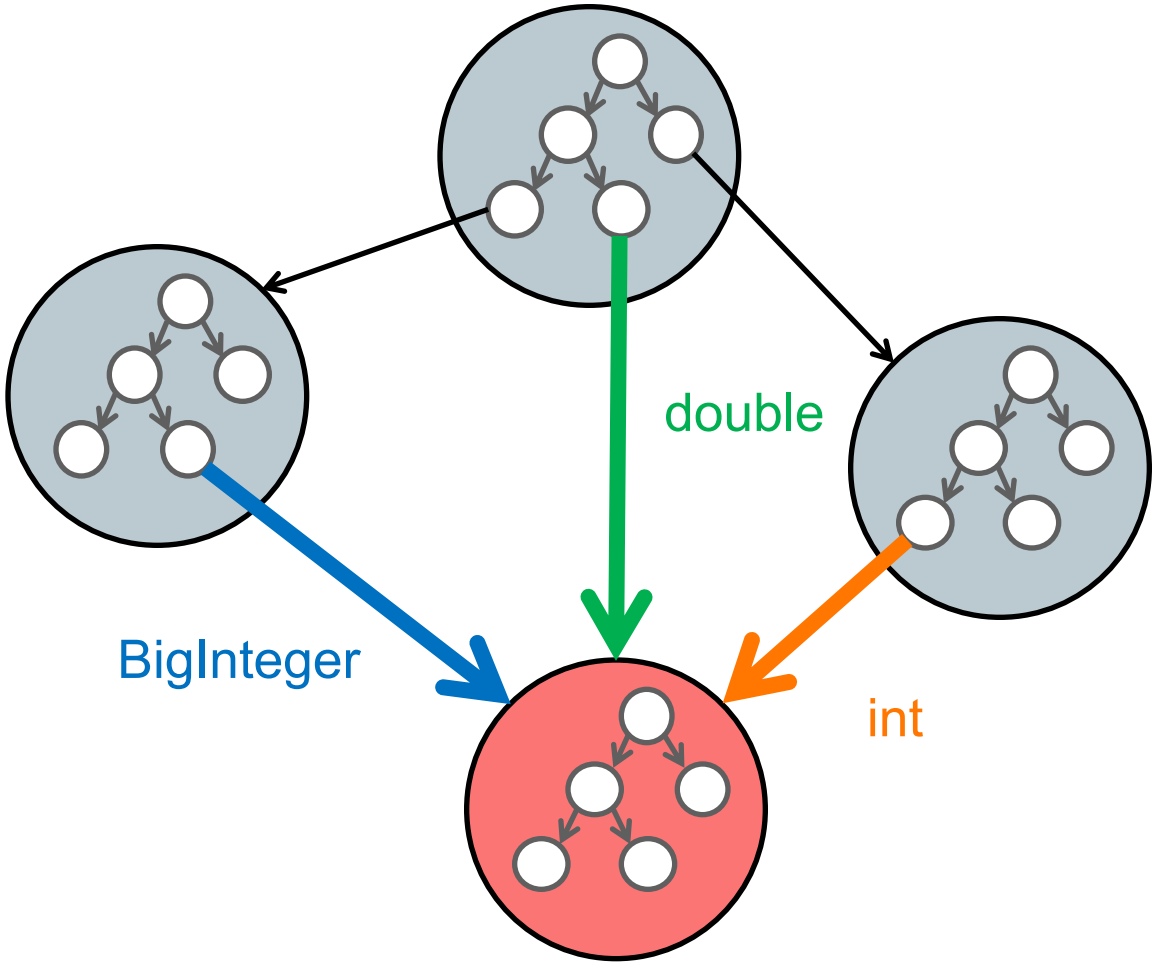


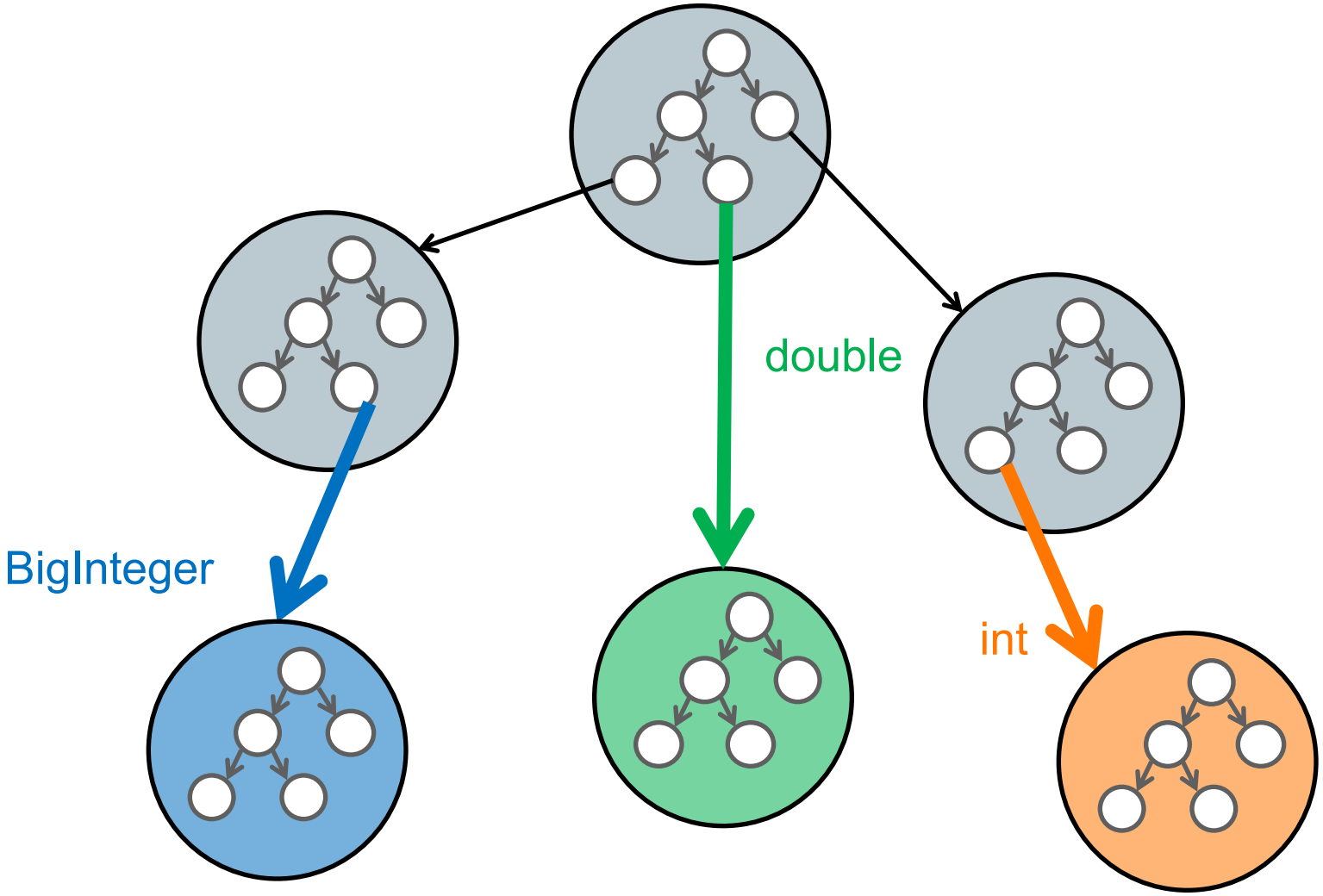
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

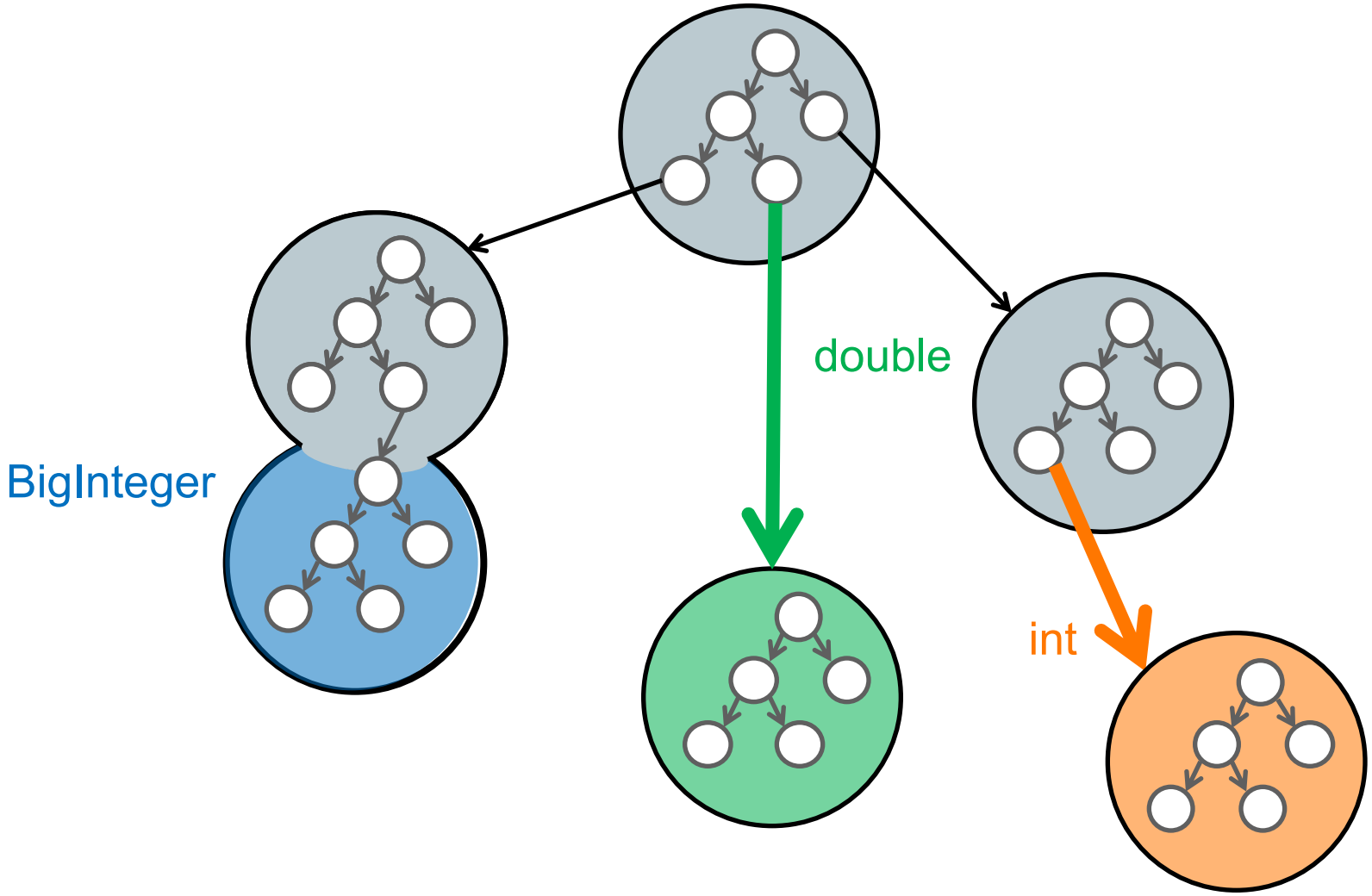




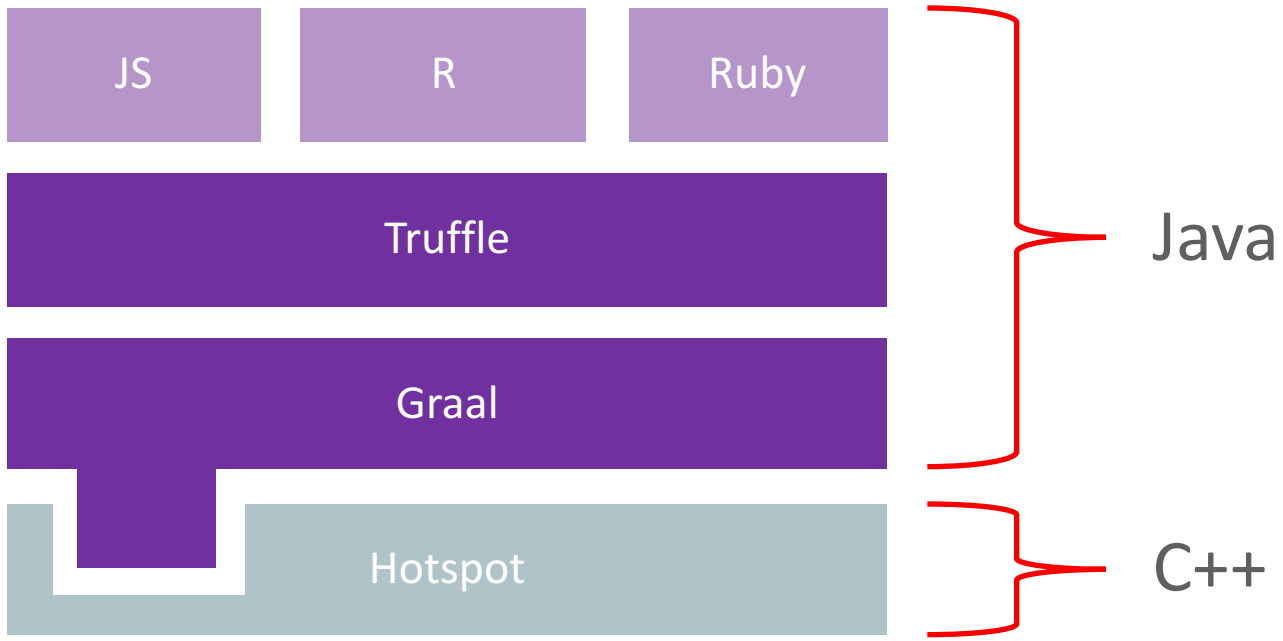


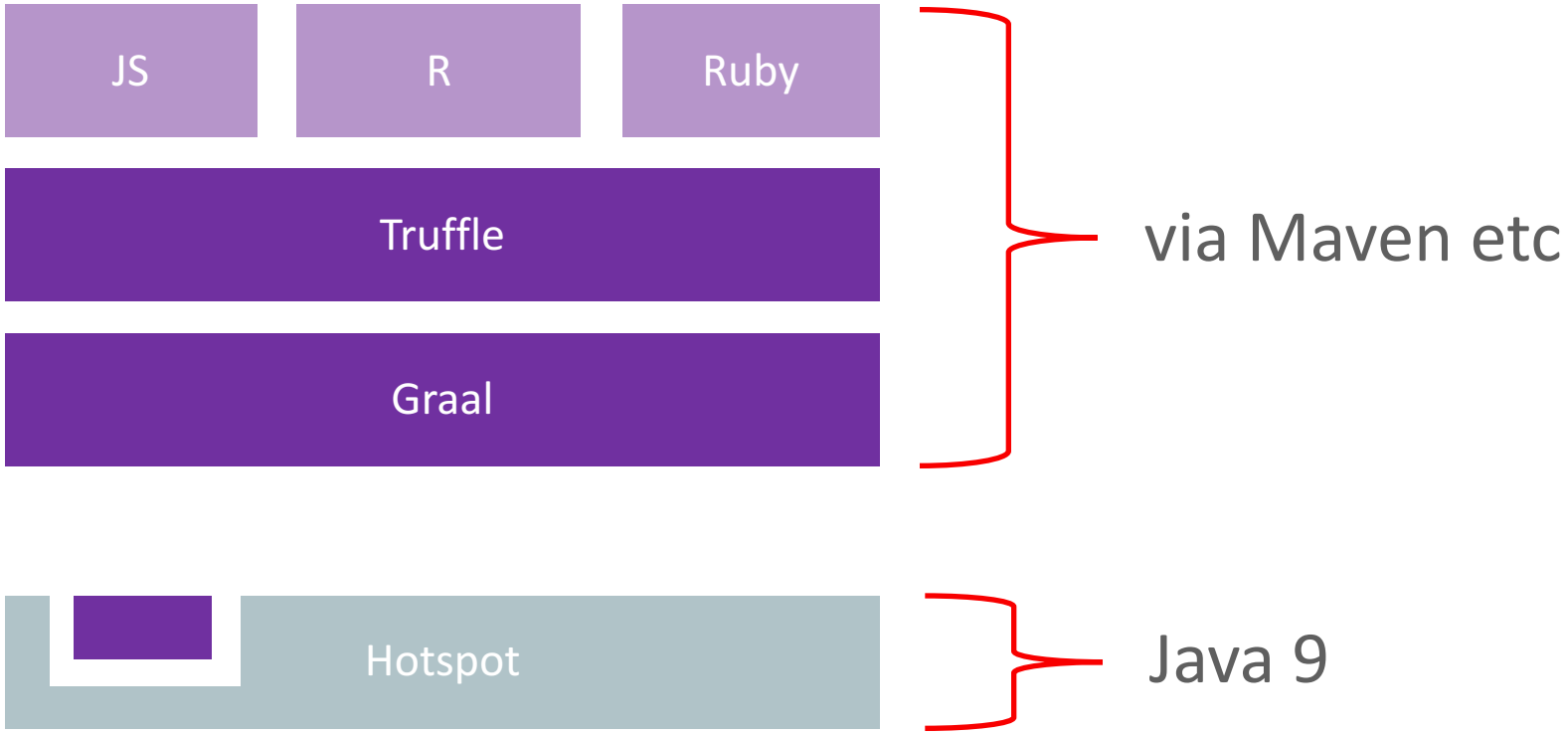


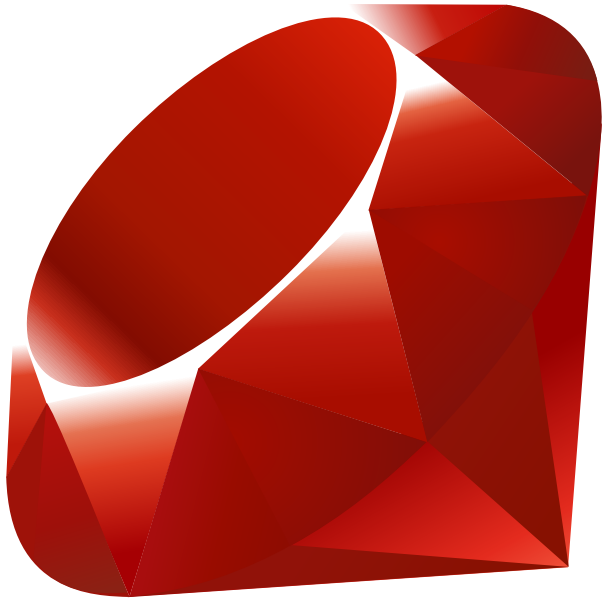




JVMCI
(JVM Compiler Interface)







Completeness – language and core library

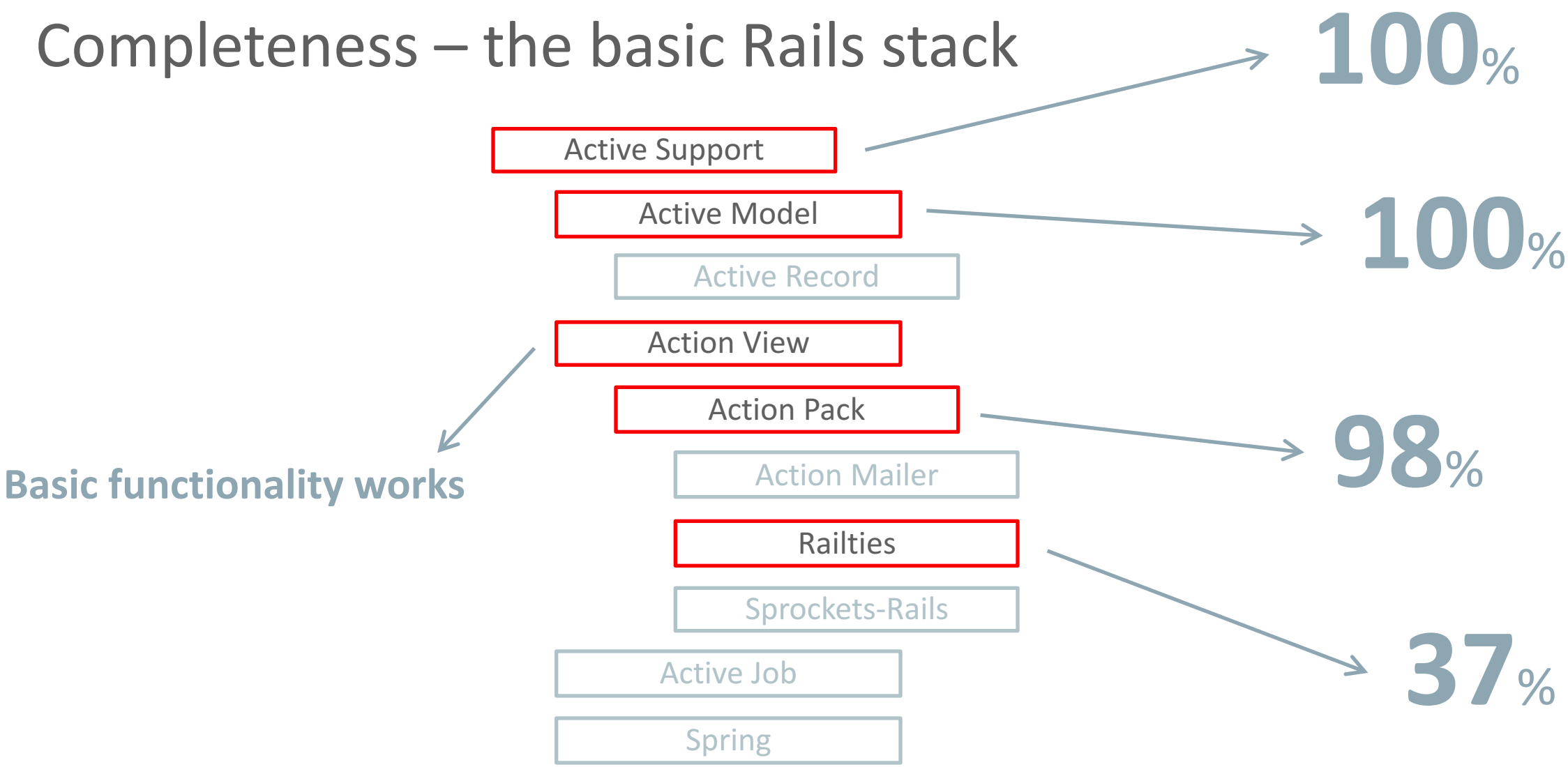
99% Ruby language

JRuby passes 94%

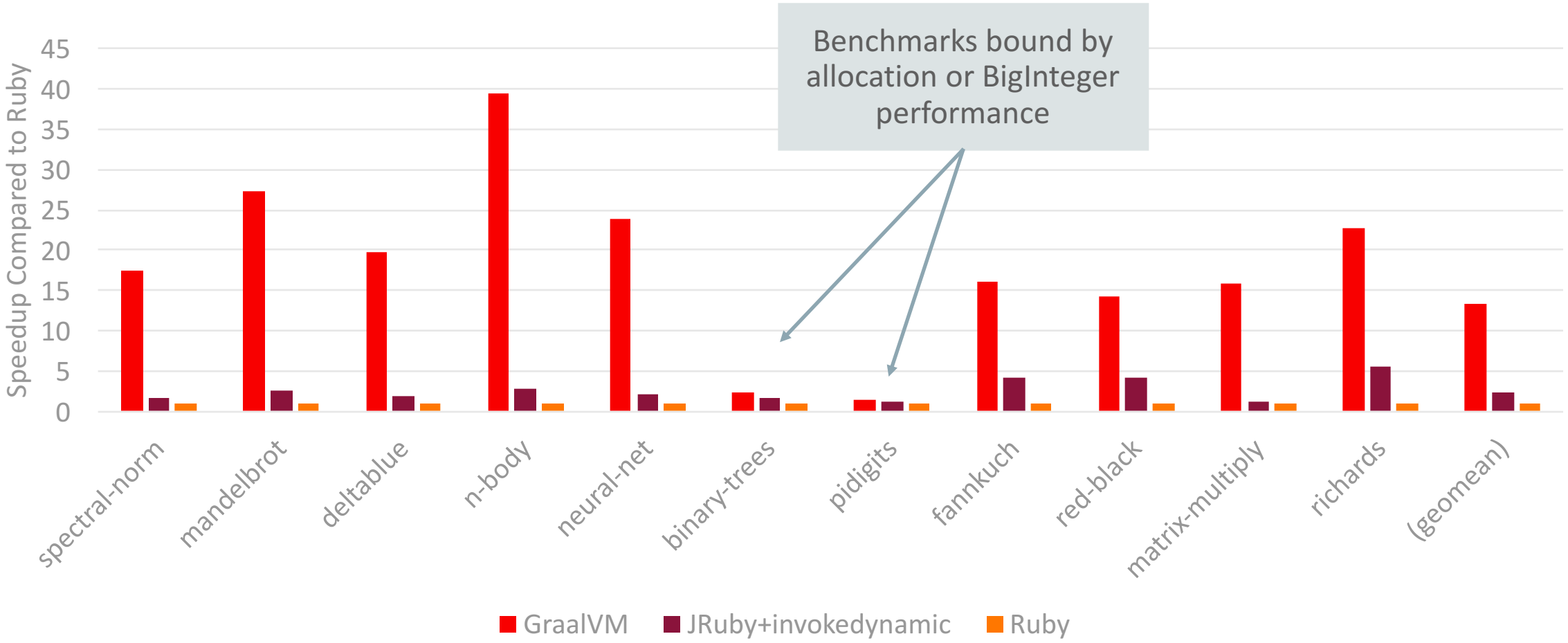
95% Ruby core library

JRuby passes 95%

Completeness – the basic Rails stack



Classic research benchmarks – 10-20x faster

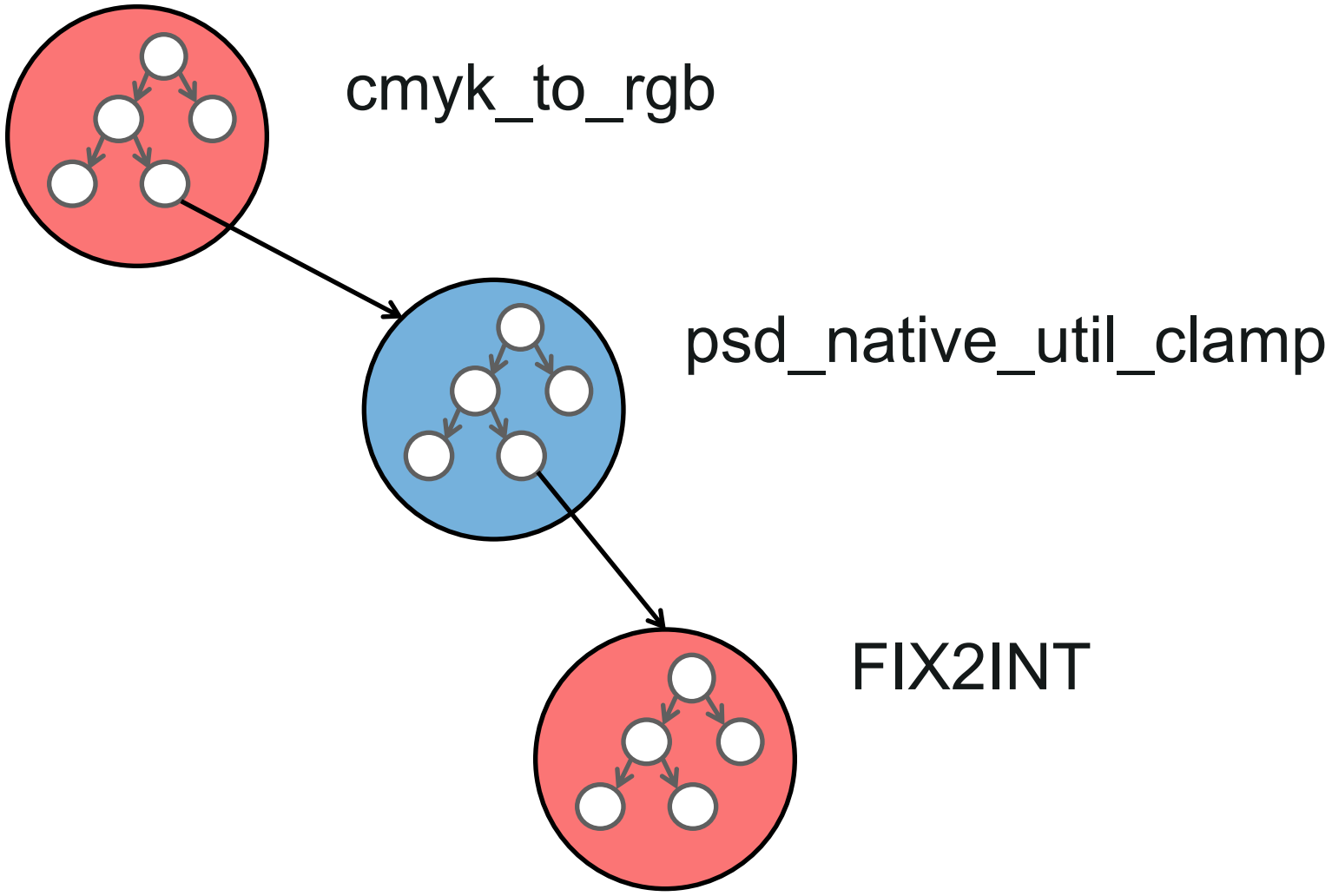


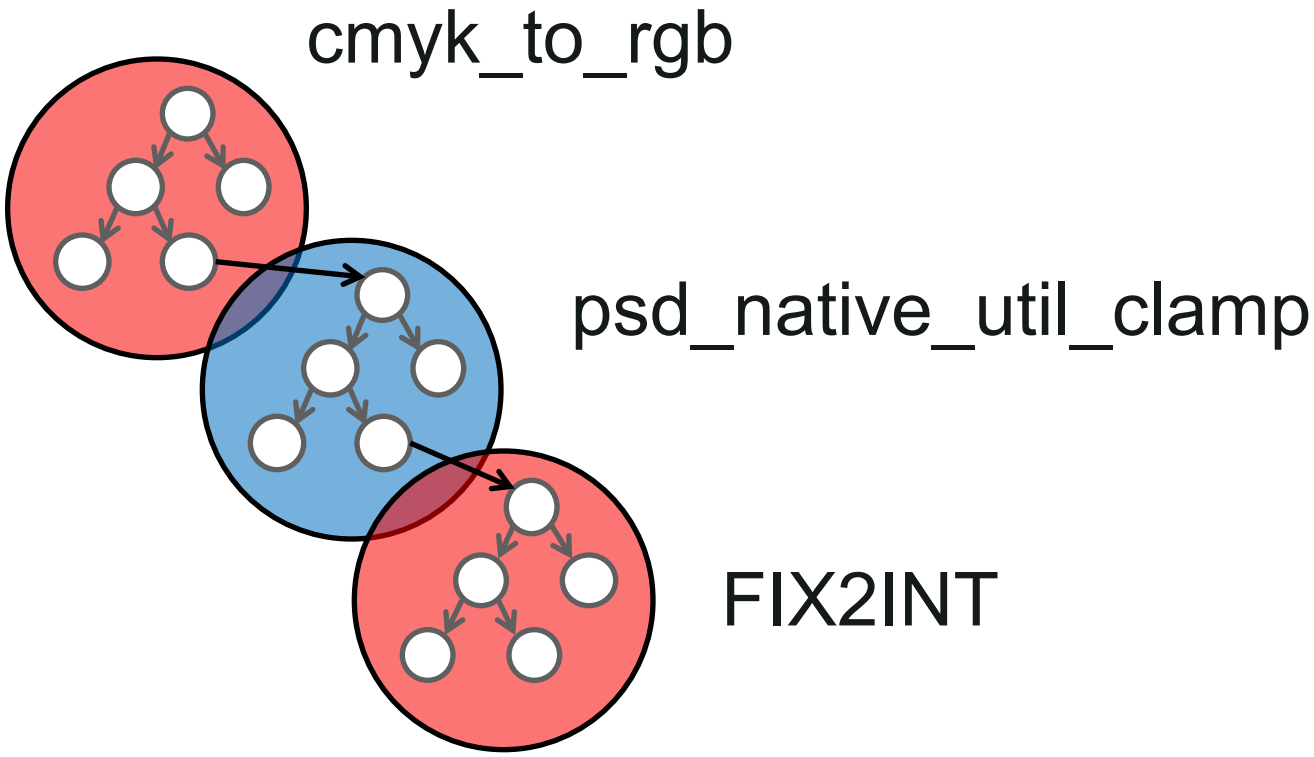
```
VALUE psd_native_util_clamp(VALUE self,
      VALUE r_num, VALUE r_min, VALUE r_max) {
  int num = FIX2INT(r_num);
  int min = FIX2INT(r_min);
  int max = FIX2INT(r_max);

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

```
define i8* @psd_native_util_clamp(i8* %self, i8* %r_num, i8* %r_min, i8* %r_max)
  %1 = call i32 @FIX2INT(i8* %r_num)
  %2 = call i32 @FIX2INT(i8* %r_min)
  %3 = call i32 @FIX2INT(i8* %r_max)
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5                                       ; preds = %0
  br label %12
; <label>:6                                       ; preds = %0
  %7 = icmp slt i32 %1, %2
  br i1 %7, label %8, label %9
; <label>:8                                       ; preds = %6
  br label %10
; <label>:9                                       ; preds = %6
  br label %10
; <label>:10                                      ; preds = %9, %8
  %11 = phi i8* [ %r_min, %8 ], [ %r_num, %9 ]
  br label %12
; <label>:12                                      ; preds = %10, %5
  %13 = phi i8* [ %r_max, %5 ], [ %11, %10 ]
  ret i8* %13
}
```

How we implement C extensions





```
VALUE* bg_pixels = RARRAY_PTR(rb_funcall(self, rb_intern("pixels"), 0));
VALUE* fg_pixels = RARRAY_PTR(rb_funcall(other, rb_intern("pixels"), 0));

long x = 0;
long y = 0;
for( y = 0; y < other_height; y++ ){
  for( x = 0; x < other_width; x++ ){
    bg_index = ( x + offset_x ) + ( y + offset_y ) * self_width;
    bg_pixels[bg_index] = UINT2NUM(
      oily_png_compose_color(
        NUM2UINT( fg_pixels[x+ y * other_width] ),
        NUM2UINT( bg_pixels[bg_index] ) ) );
  }
}
```

Instead of RARRAY_PTR returning a pointer (a number), return a proper Java object

```
.....
%42 = call i8** @RARRAY_PTR(i8* %41)
.....
%55 = getelementptr inbounds i8** %42, i64 %54
%56 = load i8** %55, align 8
.....
```

Operations like getelementptr can return a new Java object that remembers the original object, and what offset to use

Let SSA names store Java objects as well as numbers

The load can then use whatever logic we want to actually read a value from the Java object – reuse normal Ruby array logic

```
void*
```

```
public final class LLVMTruffleObject {
    private final TruffleObject object;
    private final long offset;
}
```

The C program has no way of knowing it's not a real char *

Virtualised pointer – really points to a Java object which represents our Ruby string

```
char *virtualised_string;
virtualised_string[n] = 'x';
```

```
virtualised_string.read_at_offset(n)
```

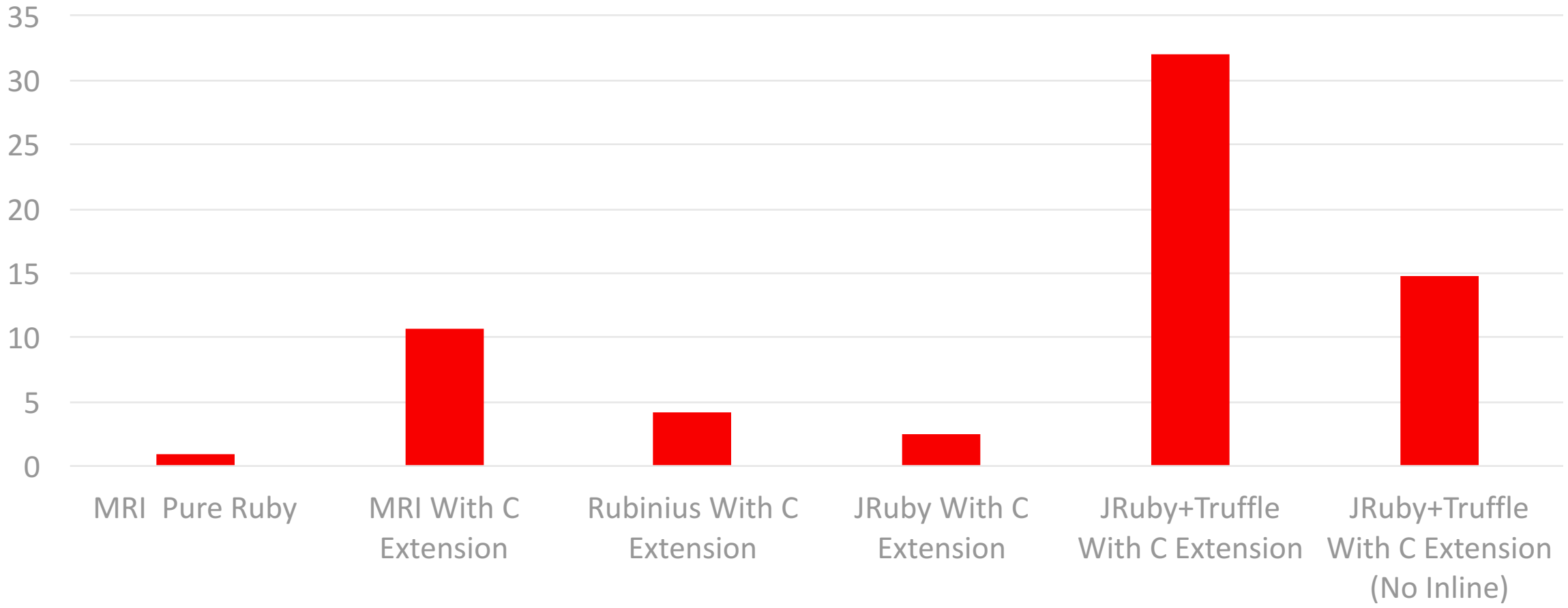
Intrinsic operations on the pointer are redirected to be method calls

Evaluation

Evaluation is based on earlier work

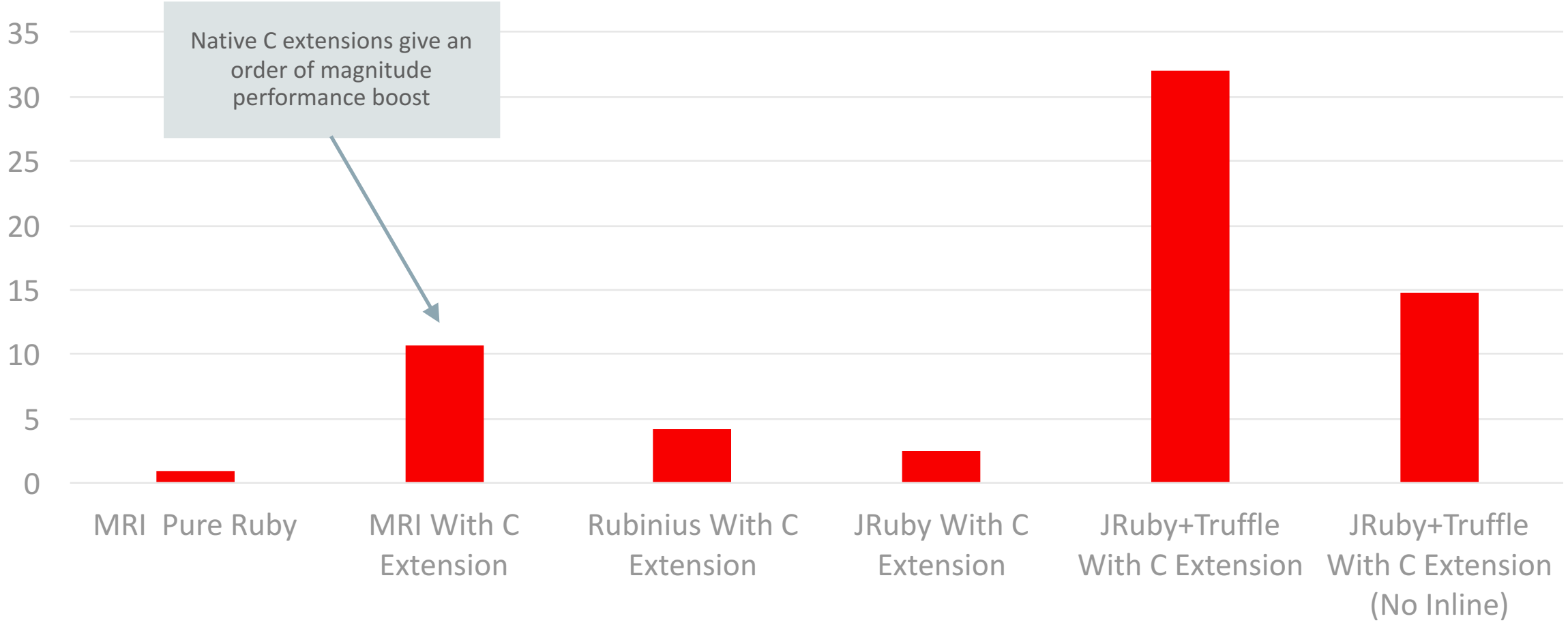
- We used to have a C interpreter – TruffleC
- We've moved on from this, because we want to support more languages
- But we aren't able to run all the same benchmarks yet
- So we've showing results from our old implementation in the mean time
- We're pretty sure results will be similar, as the compiled code is similar

Performance on Ruby C Extensions Oily PNG and PSD Native



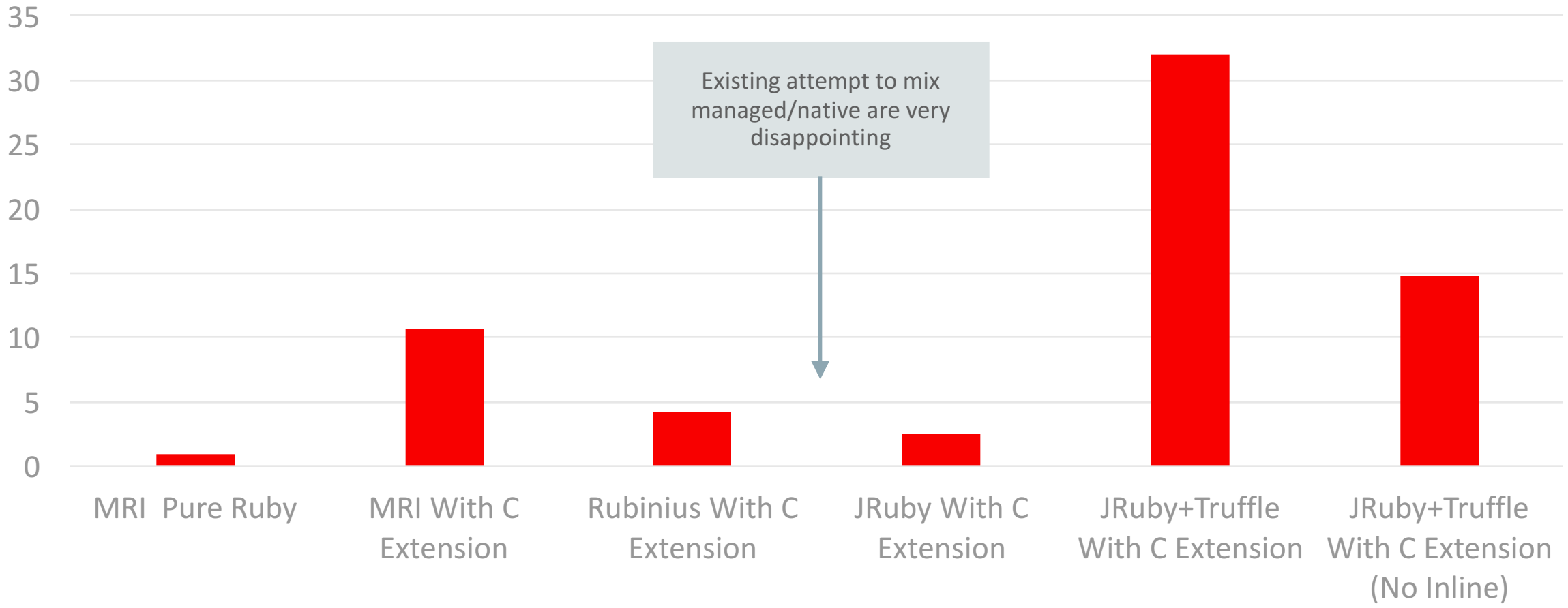
M. Grimmer, C. Seaton, T. Würthinger, H. Mössenböck. Dynamically Composing Languages in a Modular Way: Supporting C Extensions for Dynamic Languages. In Proceedings of the 14th International Conference on Modularity, 2015.

Performance on Ruby C Extensions Oily PNG and PSD Native



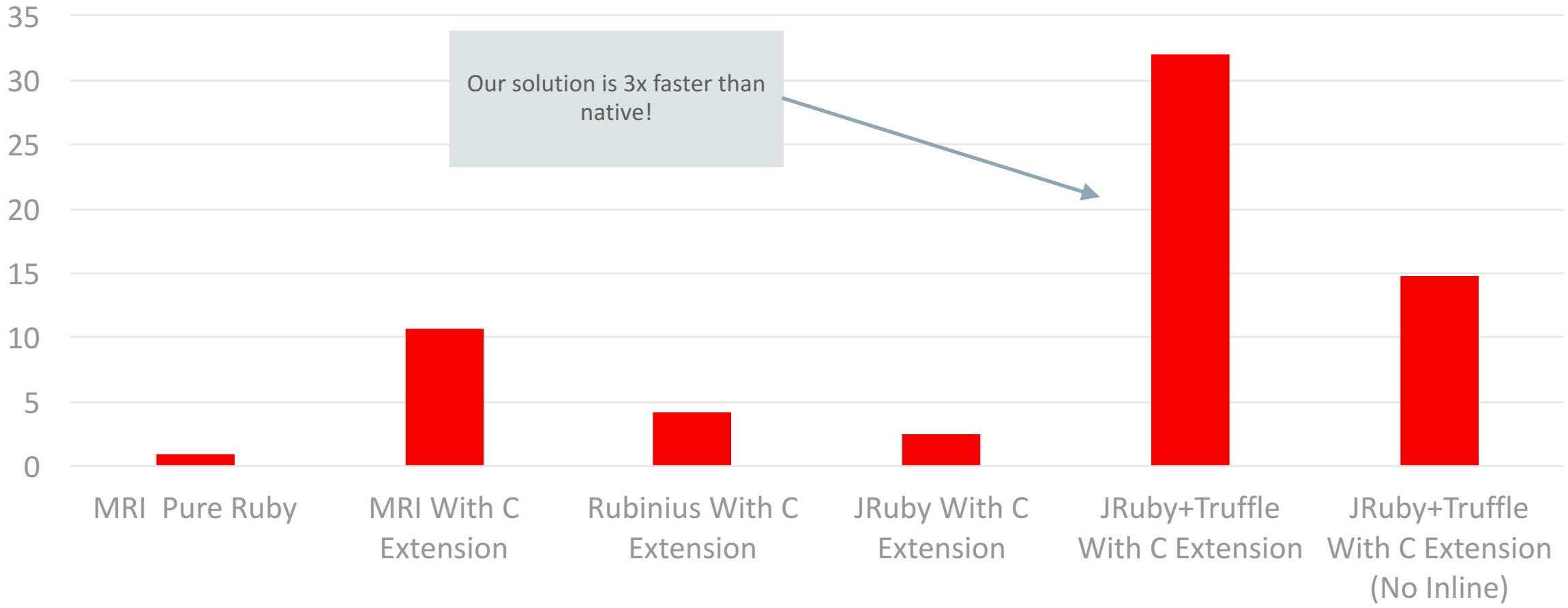
M. Grimmer, C. Seaton, T. Würthinger, H. Mössenböck. Dynamically Composing Languages in a Modular Way: Supporting C Extensions for Dynamic Languages. In Proceedings of the 14th International Conference on Modularity, 2015.

Performance on Ruby C Extensions Oily PNG and PSD Native



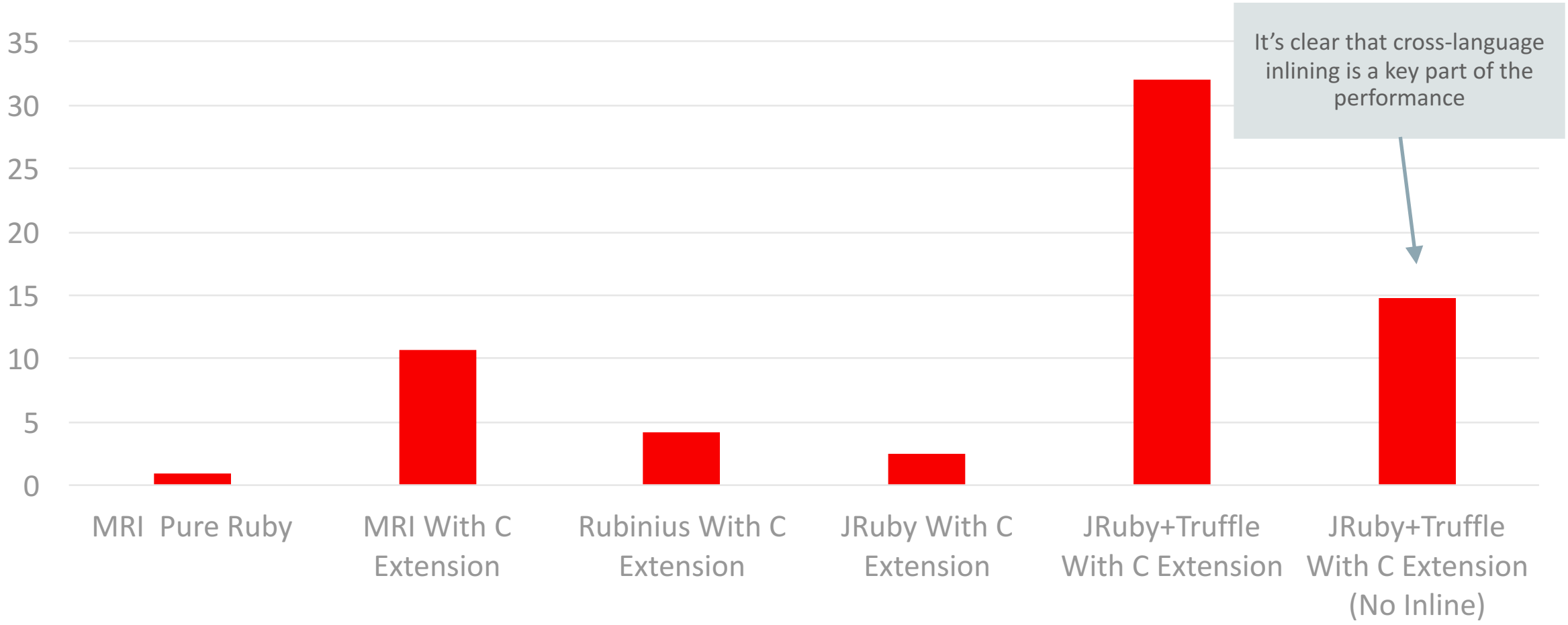
M. Grimmer, C. Seaton, T. Würthinger, H. Mössenböck. Dynamically Composing Languages in a Modular Way: Supporting C Extensions for Dynamic Languages. In Proceedings of the 14th International Conference on Modularity, 2015.

Performance on Ruby C Extensions Oily PNG and PSD Native



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.

Performance on Ruby C Extensions Oily PNG and PSD Native



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.

Conclusions

Parallel Graph Analytics
Programming Languages and Runtimes
Souffle
Datasets

Overview Java Polyglot Downloads Learn More

Oracle Labs GraalVM and JVMCI JDK Downloads

Thank you for downloading this release of the Oracle Labs GraalVM. With this release, one can execute Java applications with Graal, as well as applications written in JavaScript, Ruby, and R, with our Polyglot language engines.

You must accept the OTN License Agreement to download this software.
Accept License Agreement | Decline License Agreement

GraalVM preview for Linux (v0.15), Development Kit
GraalVM preview for Linux (v0.15), Runtime Environment
GraalVM preview for Mac OS X (v0.15), Development Kit
GraalVM preview for Mac OS X (v0.15), Runtime Environment

labsjdk-8u92-jvmci-0.20-darwin-amd64.tar.gz
labsjdk-8u92-jvmci-0.20-linux-amd64.tar.gz
labsjdk-8u92-jvmci-0.20-solaris-sparcv9.tar.gz

How to install GraalVM

Unpack the downloaded *.tar.gz file on your machine. You can then use the java executable to

Open Source

- https://github.com/graalvm/graal-core
  - Graal compiler
- https://github.com/graalvm/truffle
  - Truffle language implementation framework
- https://github.com/graalvm/fastr
  - Fast R runtime
- https://github.com/graalvm/sulong
  - Dynamic runtime for LLVM bitcode
- https://github.com/jruby/jruby/wiki/Truffle
  - Fast Ruby runtime

Acknowledgements

Oracle
Danilo Ansaloni
Stefan Anzinger
Cosmin Basca
Daniele Bonetta
Matthias Brantner
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Martin Entlicher
Brandon Fish
Bastian Hossbach
Christian Humer
Mick Jordan
Vojin Jovanovic
Peter Kessler
David Leopoldseder
Kevin Menard
Jakub Podlešák
Aleksandar Prokopec
Tom Rodriguez

Oracle (continued)
Roland Schatz
Chris Seaton
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Lukas Stadler
Codrut Stancu
Jan Štola
Jaroslav Tulach
Michael Van De Vanter
Adam Welc
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger

Oracle Interns
Brian Belleville
Miguel Garcia
Shams Imam
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
David Piorkowski
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Alumni
Erik Eckstein
Michael Haupt
Christos Kotselidis
Hyunjin Lee
David Leibs
Chris Thalinger
Till Westmann

JKU Linz
Prof. Hanspeter Mössenböck
Benoit Daloze
Josef Eisl
Thomas Feichtinger
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Huber
Stefan Marr
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
Gulfem Savrun Yeniceri
Wei Zhang

Purdue University
Prof. Jan Vitek
Tomas Kalibera
Petr Maj
Lei Zhao

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

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

University of Lugano, Switzerland
Prof. Walter Binder
Sun Haiyang
Yudi Zheng

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.

Integrated Cloud

Applications & Platform Services


ORACLE®