
ORACLE®

Ruby's C Extension Problem and How We're Solving It

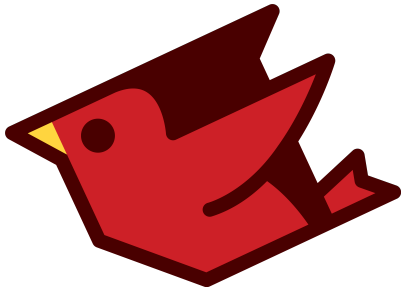
RubyConf 2016

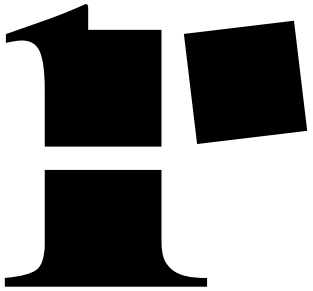
Chris Seaton
Research Manager
Oracle Labs
November 2016

Safe Harbor Statement

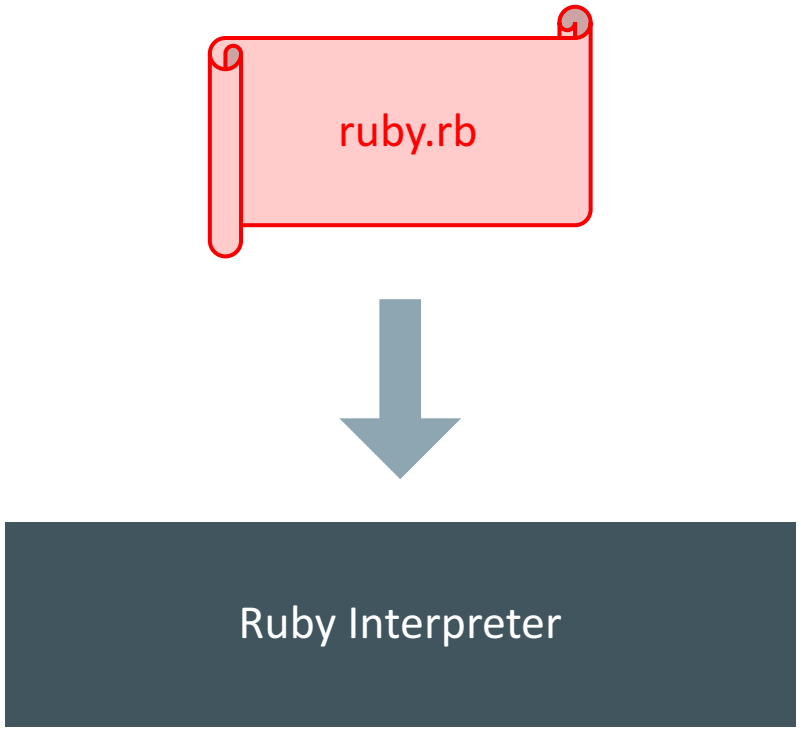
The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing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

Lots of people want to make Ruby faster

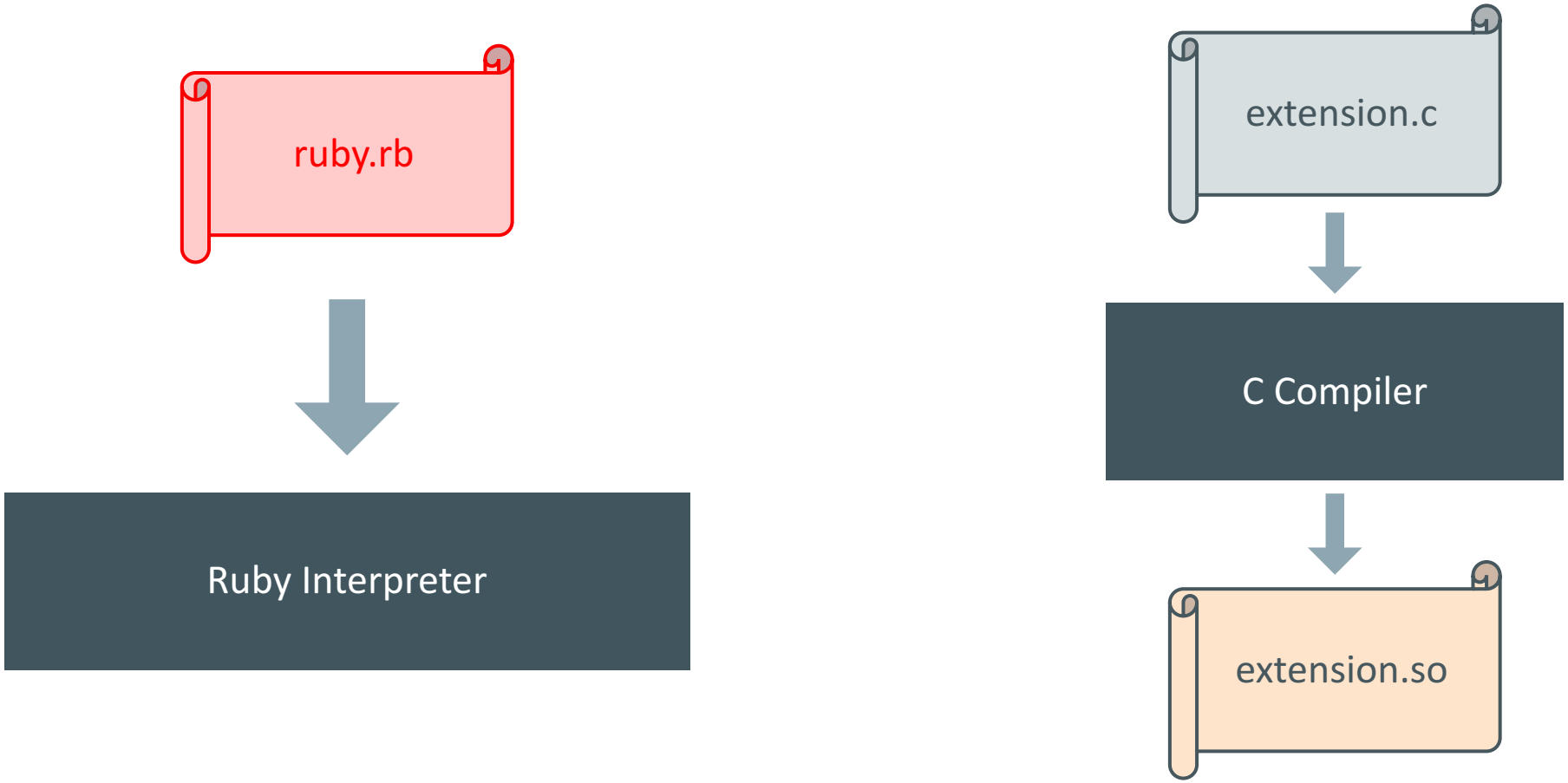




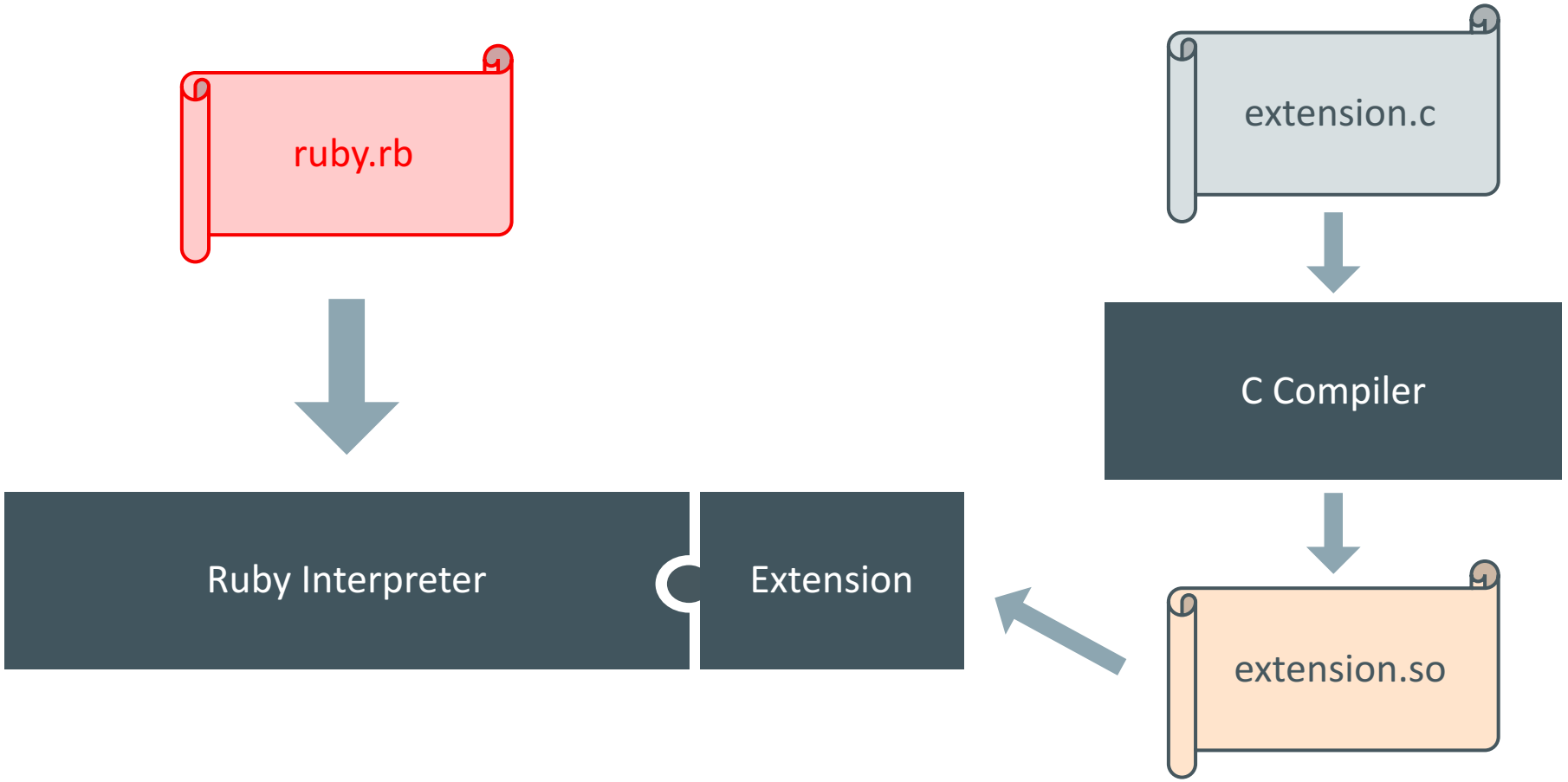
C extensions – the original solution for performance



C extensions – the original solution for performance



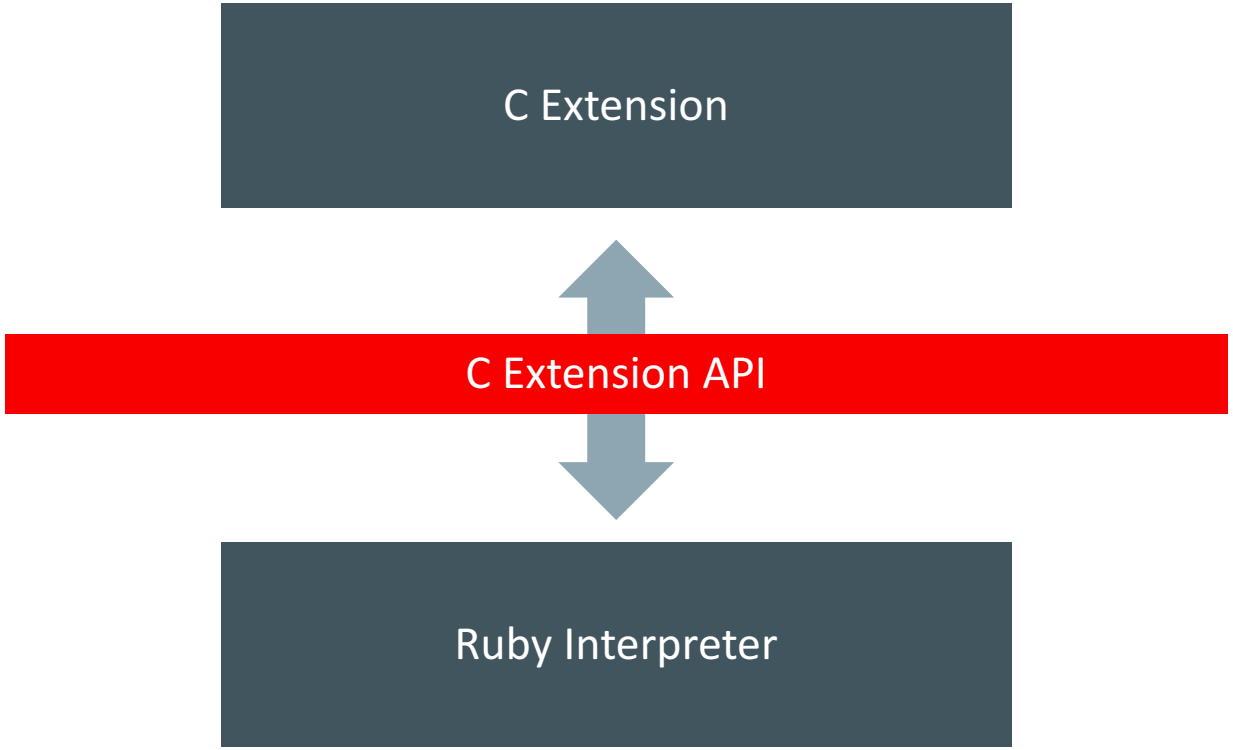
C extensions – the original solution for performance

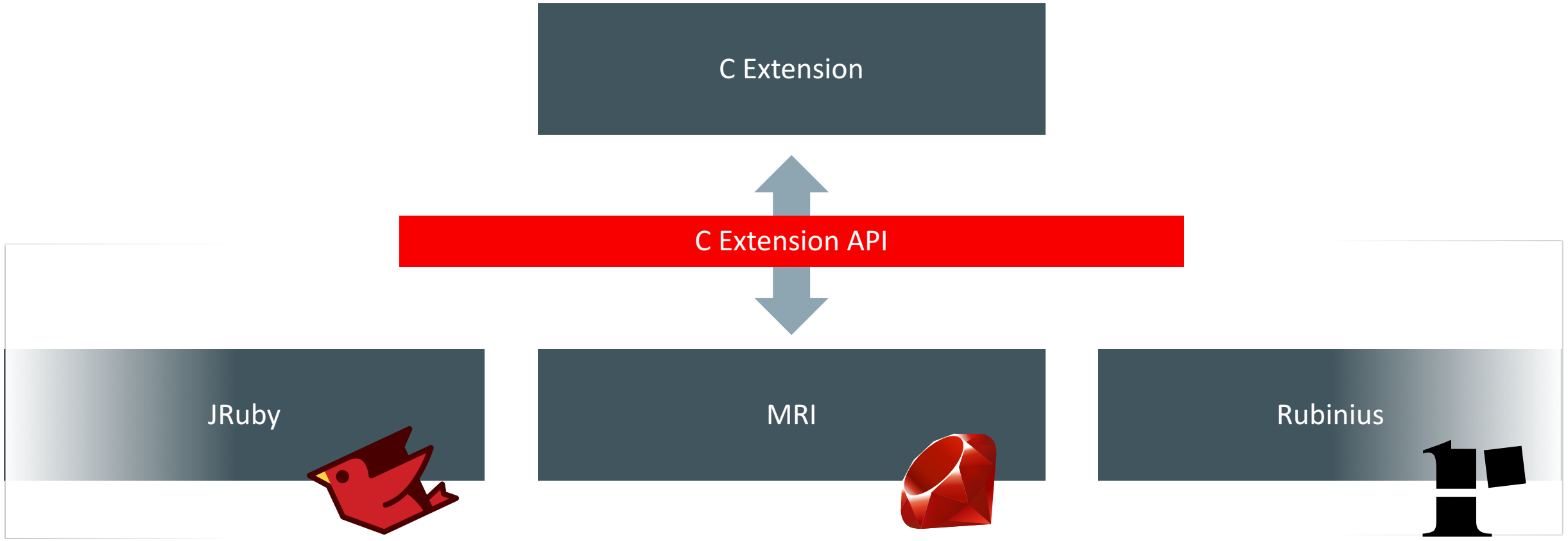


```ruby
def clamp(num, min, max)
  [min, num, max].sort[1]
end
```

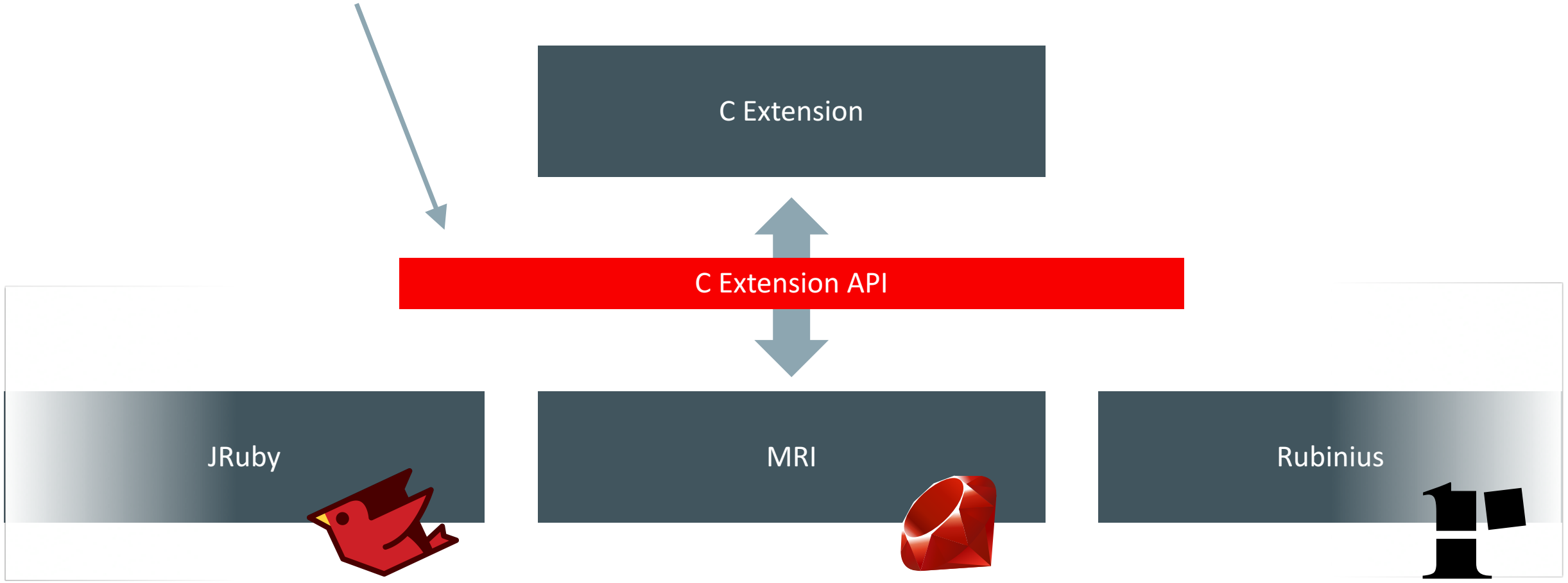
```c
VALUE psd_native_util_clamp(VALUE self,
    VALUE r_num, VALUE r_min, VALUE r_max) {
  int num = FIX2INT(r_num);
  int min = FIX2INT(r_min);
  int max = FIX2INT(r_max);

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

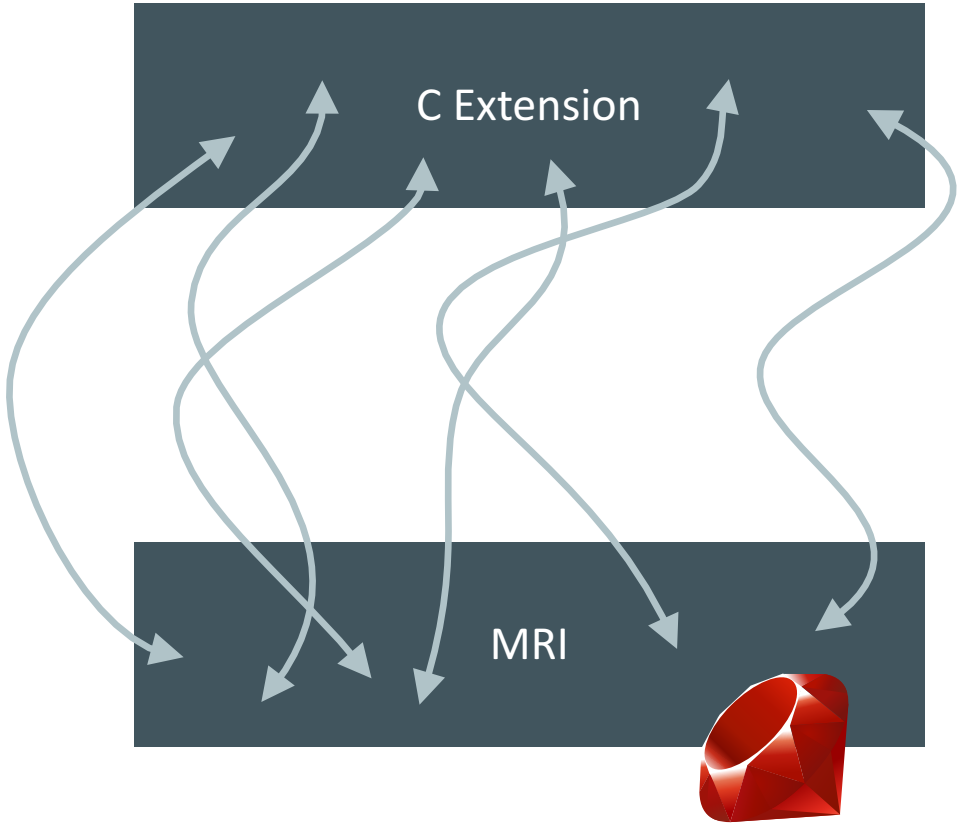
Why C extensions hold us back

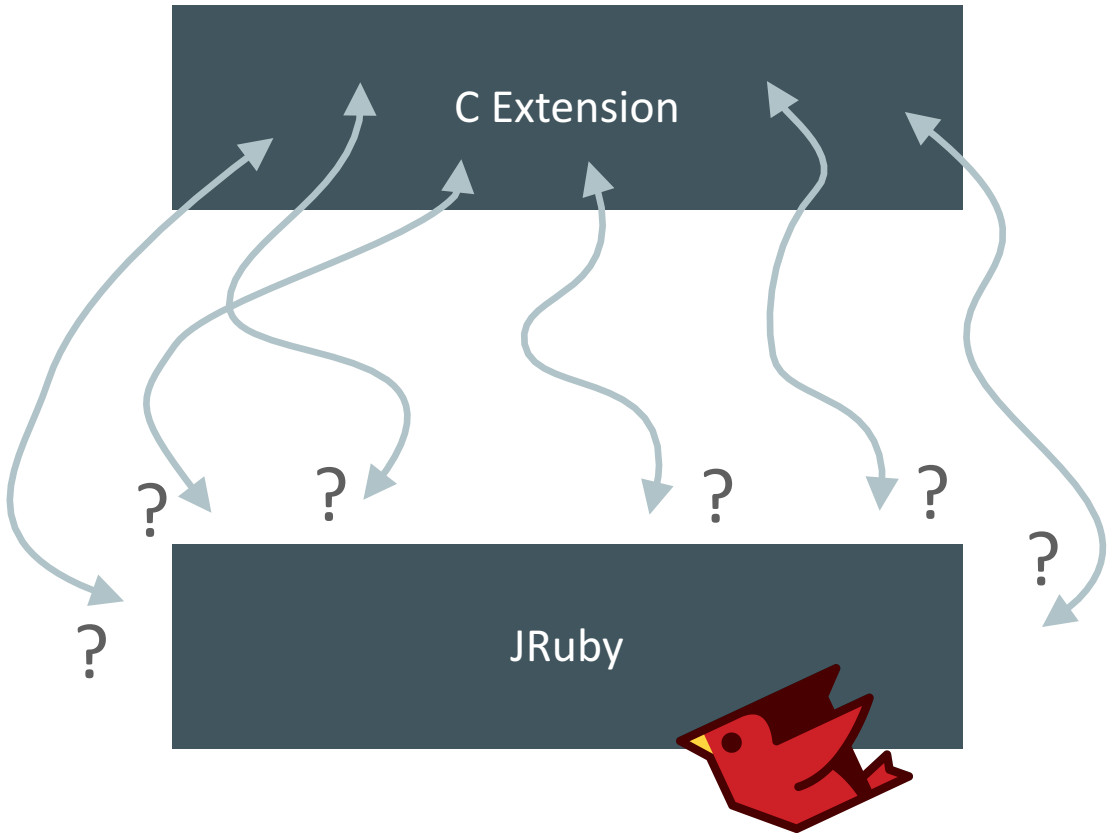


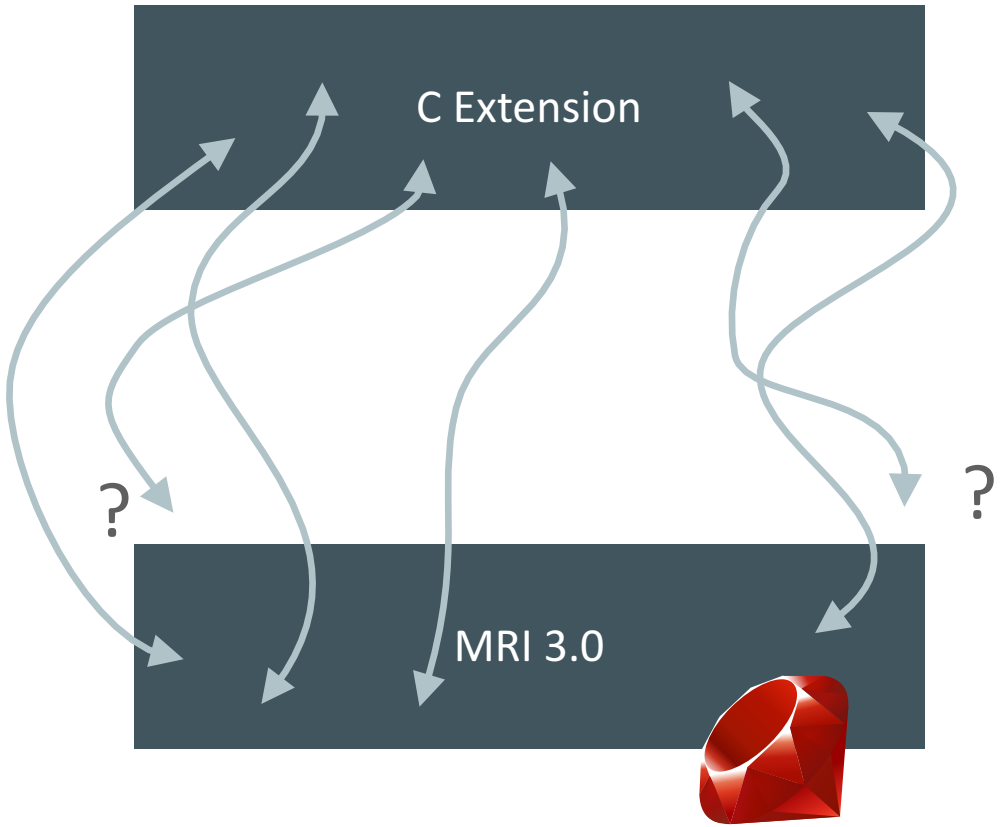


Bad news – this isn't really a thing in practice









String pointers

```
char *RSTRING_PTR(VALUE string);

static VALUE
ossl_dsa_export(int argc, VALUE *argv, VALUE self)
{
    char *passwd;
    ...
    passwd = RSTRING_PTR(pass);
    ...
    PEM_write_bio_DSAPrivateKey(out, pkey->pkey.dsa, ciph,
            NULL, 0, ossl_pem_passwd_cb, passwd)
    ...
}
```

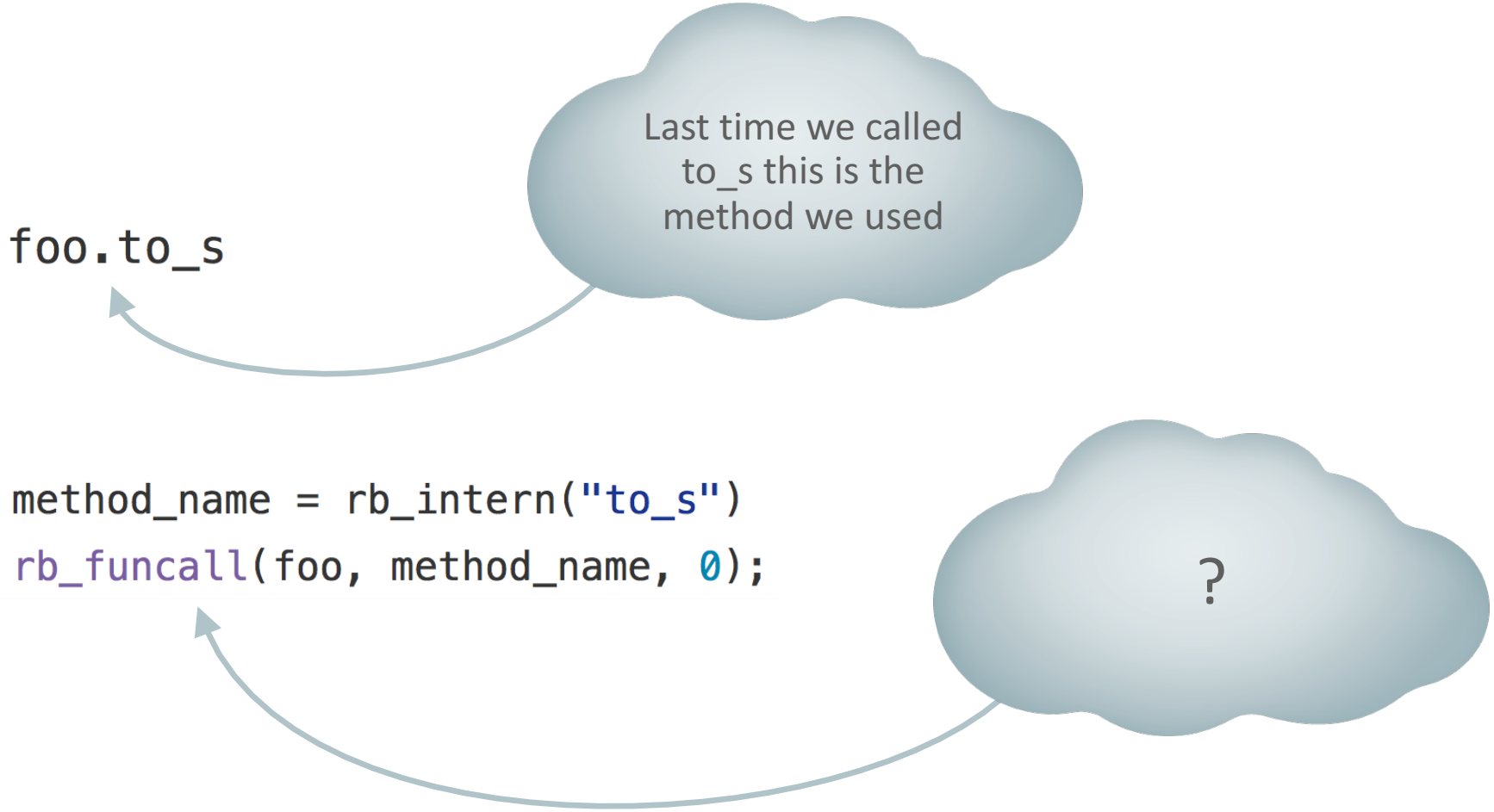
Array pointers

```
VALUE *RARRAY_PTR(VALUE array);

VALUE psd_native_blender_compose_bang(VALUE self) {
  ...
  VALUE bg_pixels = rb_funcall(bg_canvas, rb_intern("pixels"), 0);
  VALUE *bg_pixels_ptr = RARRAY_PTR(bg_pixels);
  ...
  for (i = 0, len = RARRAY_LEN(bg_pixels); i < len; i++) {
    ... bg_pixels_ptr[i] ...
  }
  ...
}
```

Data fields

```
struct RData {
  struct RBasic basic;
  void (*dmark)(void *data);
  void (*dfree)(void *data);
  void *data;
};

#define RDATA(value)    ((struct RData *)value)

#define DATA_PTR(value) (RDATA(value)->data)

static VALUE
ossl_x509req_copy(VALUE self, VALUE other)
{
    ...
    DATA_PTR(self) = X509_REQ_dup(b);
    ...
}
```

Lack of caching when you are in C



The black box

```ruby
def add(a, b)
  a + b
end

add(14, 2)
```

```c
VALUE add(VALUE self, VALUE a, VALUE b) {
  return INT2FIX(FIX2INT(a) + FIX2INT(b));
}

add(14, 2)
```

The black box

```
def add(a, b)
  a + b
end

add(14, 2)
```

= 16

```
VALUE add(VALUE self, VALUE a, VALUE b) {
  return INT2FIX(FIX2INT(a) + FIX2INT(b));
}

add(14, 2)
```

The black box

```
def add(a, b)
  a + b
end

add(14, 2)
```

= ?

```
VALUE add(VALUE self, VALUE a, VALUE b) {
  return INT2FIX(FIX2INT(a) + FIX2INT(b));
}

add(14, 2)
```

Previous solutions to the C extension problem

Denial

- Everyone should use the FFI or Fiddle
  - FFI and Fiddle are two ways to call C functions directly from Ruby
  - 2.1 billion lines of code in RubyGems, 0.5 billion of it is C extension code
  - It might be nice if people used FFI instead of C extensions… but they don't… so little point in continuing to argue about it

```ruby
module MyLib
  extend FFI::Library
  ffi_lib 'c'
  attach_function :sqrt, [ :double ], :double
end
```

Bargaining

- Attempt to implement the C extension API as best as possible, alongside optimisations
- Generally involves a lot of copying
- JRuby used this approach in the past, Rubinius still uses it
  - JRuby only ran 60% of C extensions I tried
  - Rubinius ran 90%
  - Worse: when they didn't work they just ground to a halt, no clear failure point

Bargaining

- Try to improve the C extension API over time
  - The JavaScript (V8) and Java C extension APIs don't have these problems because they have better designed APIs that don't expose internals
  - Steady progress in this direction, has helped
  - But even OpenSSL doesn't use these new methods!

> **Don't touch pointers directly**
>
> In MRI (include/ruby/ruby.h), some macros to acquire pointers to the internal data structures are supported such as RARRAY_PTR(), RSTRUCT_PTR() and so on.
>
> DO NOT USE THESE MACROS and instead use the corresponding C-APIs such as rb_ary_aref(), rb_ary_store() and so on.

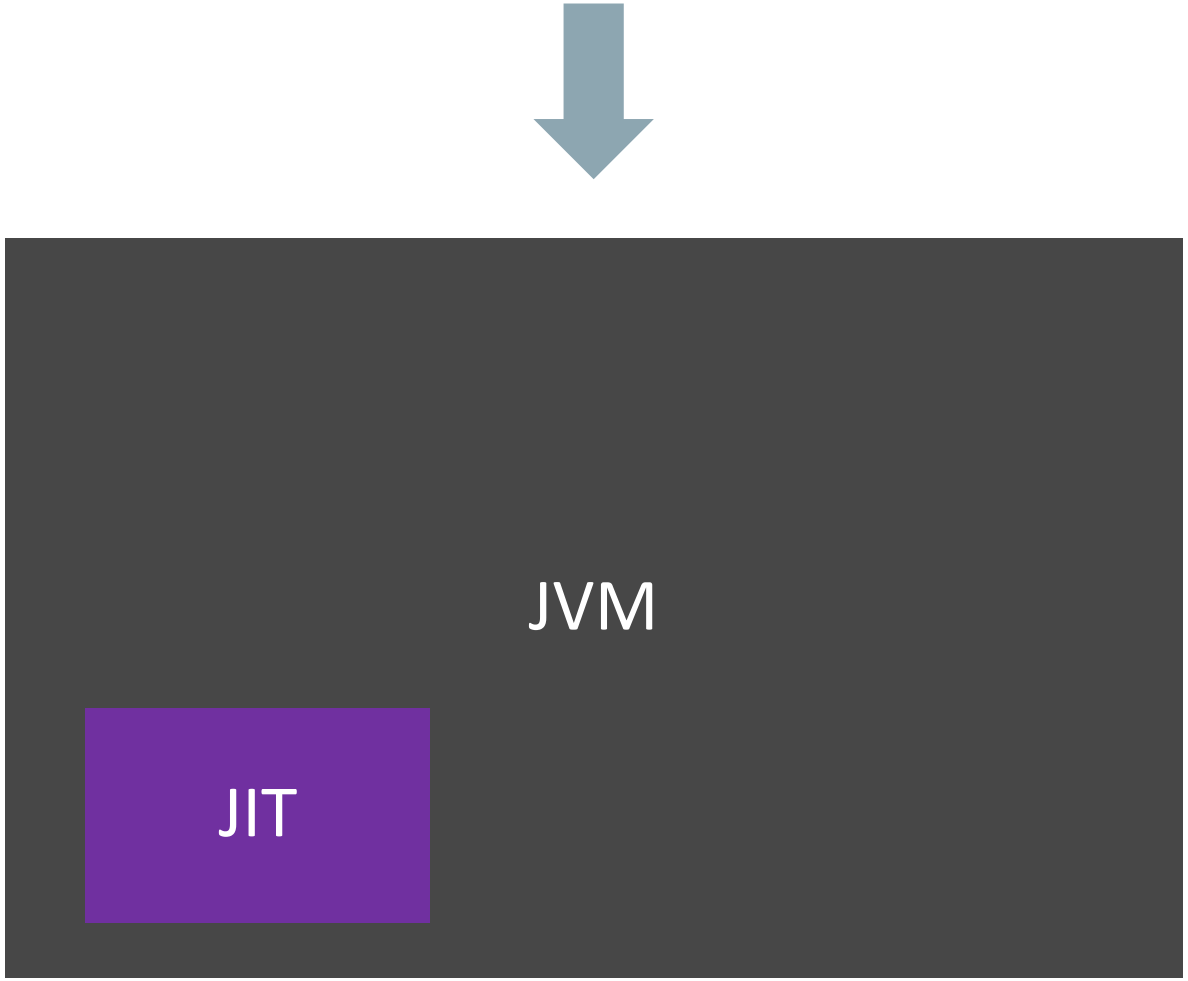
Depression

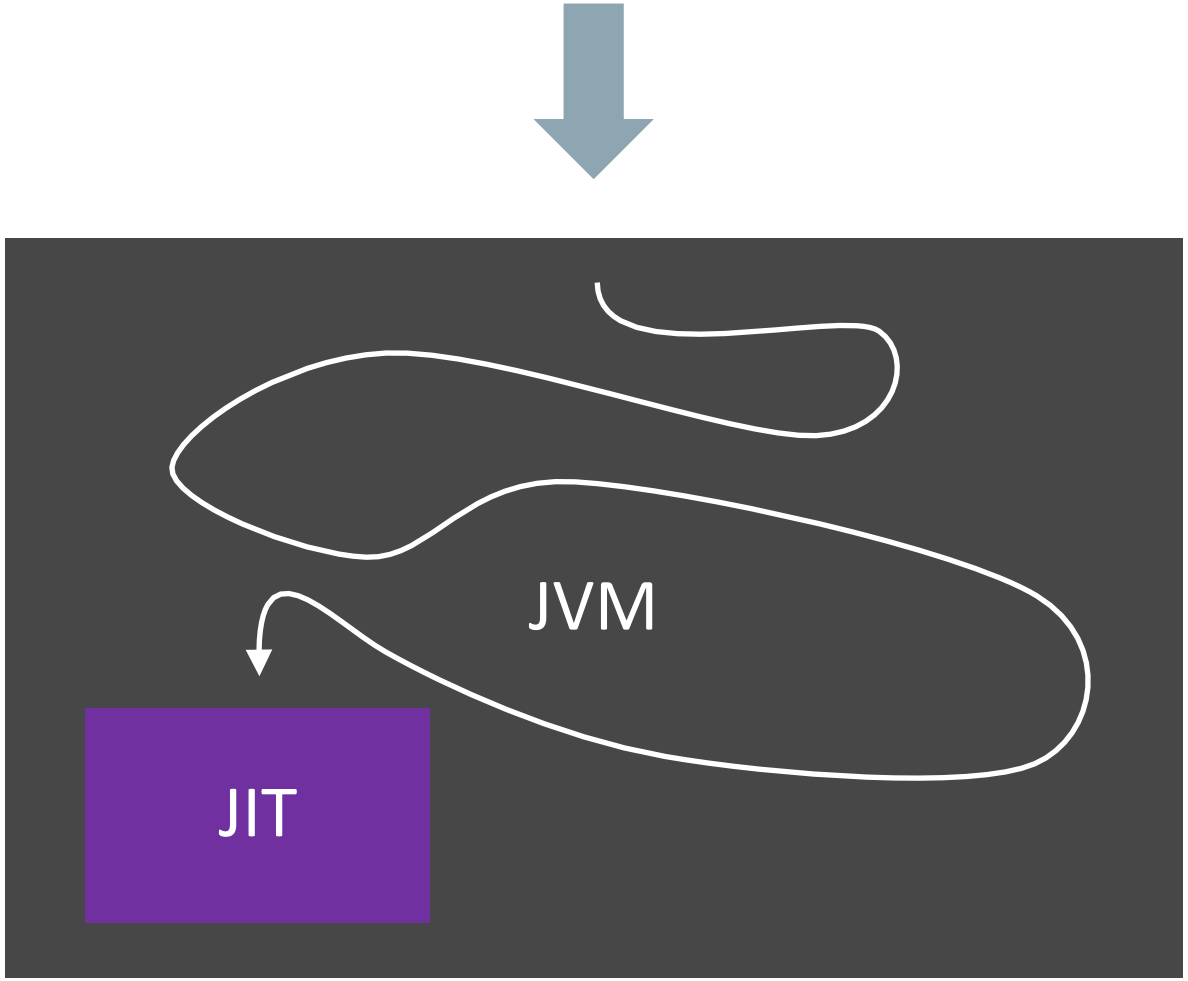
- JRuby unfortunately had to give up on their C extension work
  - They didn't have the resources to maintain it after the original developer moved on
  - Limited compatibility and limited performance
  - In the end, in was removed entirely
  - Maybe it'll return in the future (they could use the same approach as us)

Acceptance

- JRuby encourage Java extensions instead of C extensions
- Try to optimise Ruby while keeping most of the internals the same
  - IBM's OMR adds a new GC and JIT to Ruby while keeping support for C extensions
  - The techniques they can use are therefore limited
  - And so performance increases expected from OMR are more modest

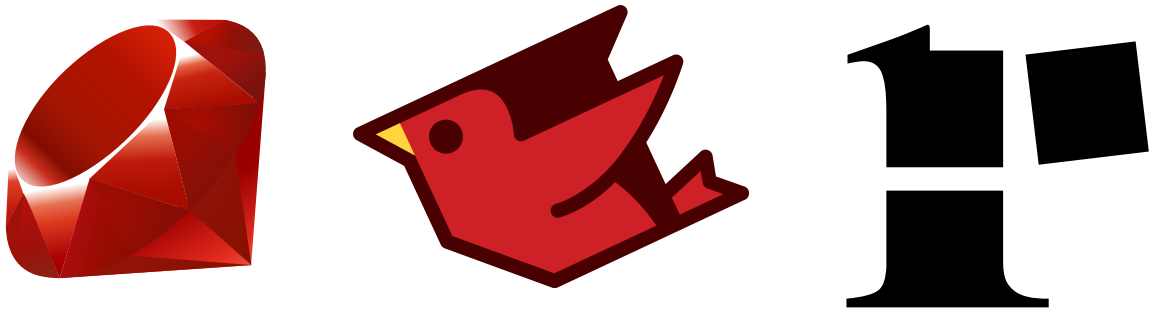
Interlude: JRuby+Truffle





Graal

JVM

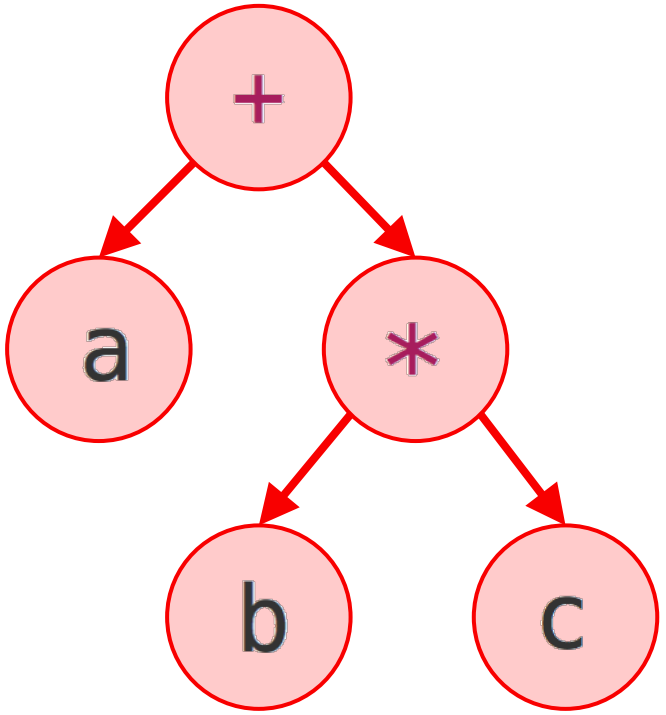


Truffle

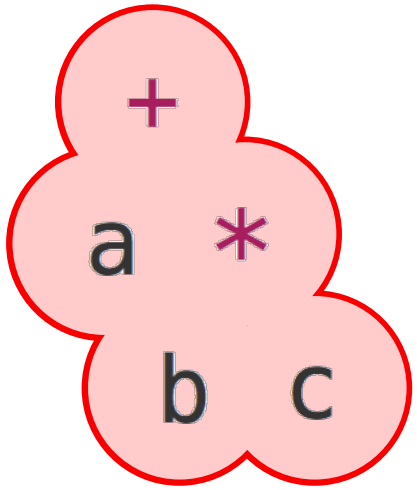
Graal VM

JVM

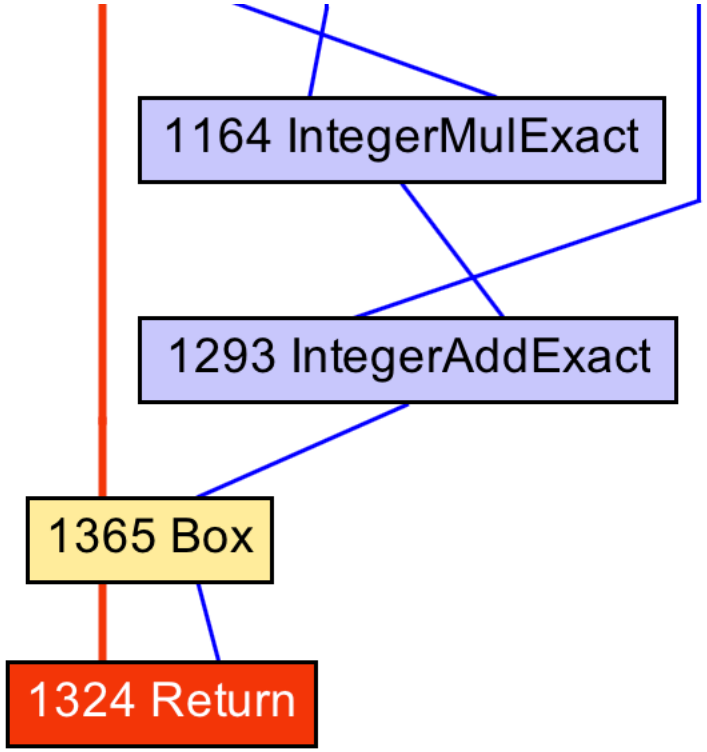
a + b * c



a + b * c



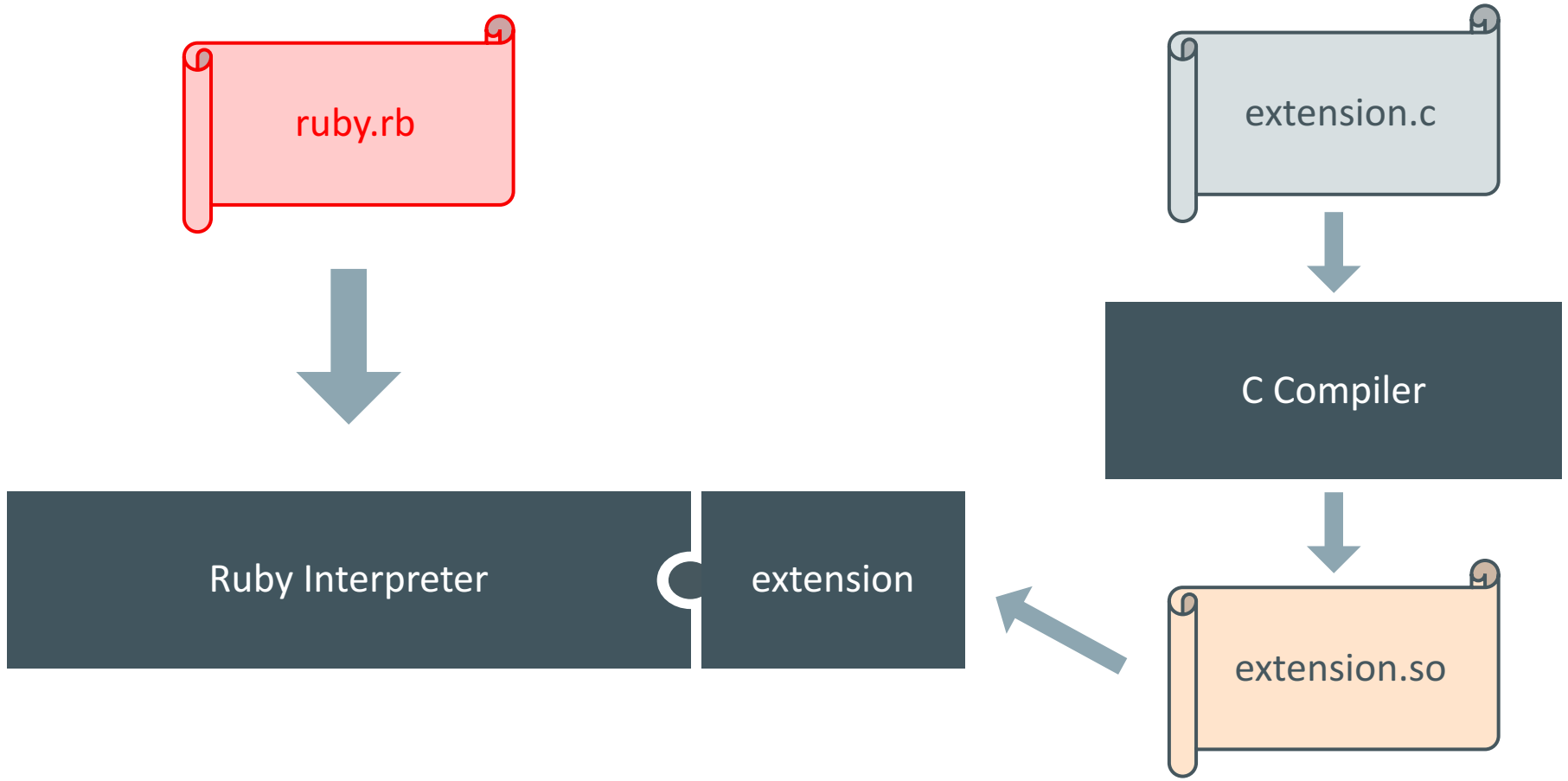
a + b * c

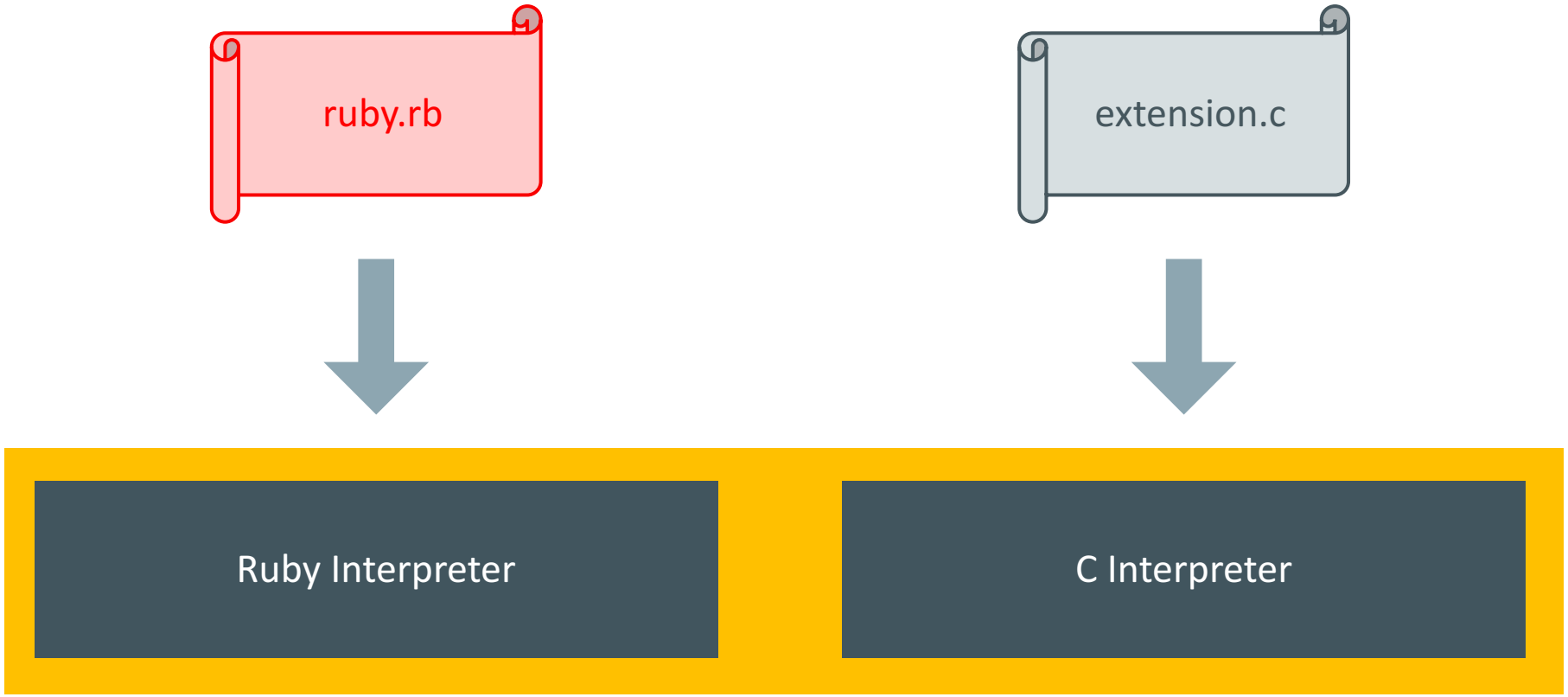


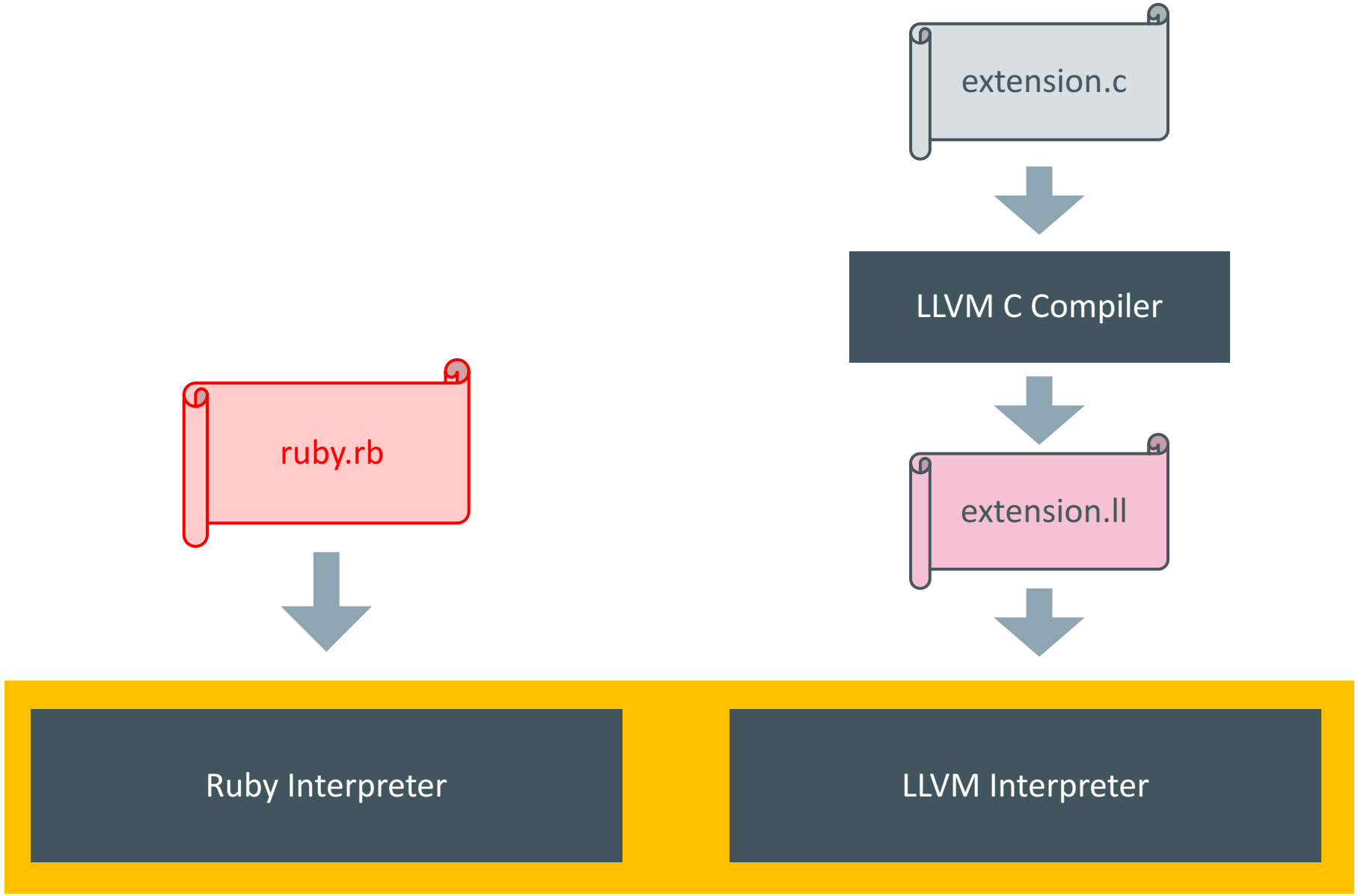
a + b * c

```
imul    %rsi,%rdx
jo      0x00000001171a3fa5
add     %rdi,%rdx
jo      0x00000001171a3fc7
```

Our radical new solution for C extensions…







```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);
}
```

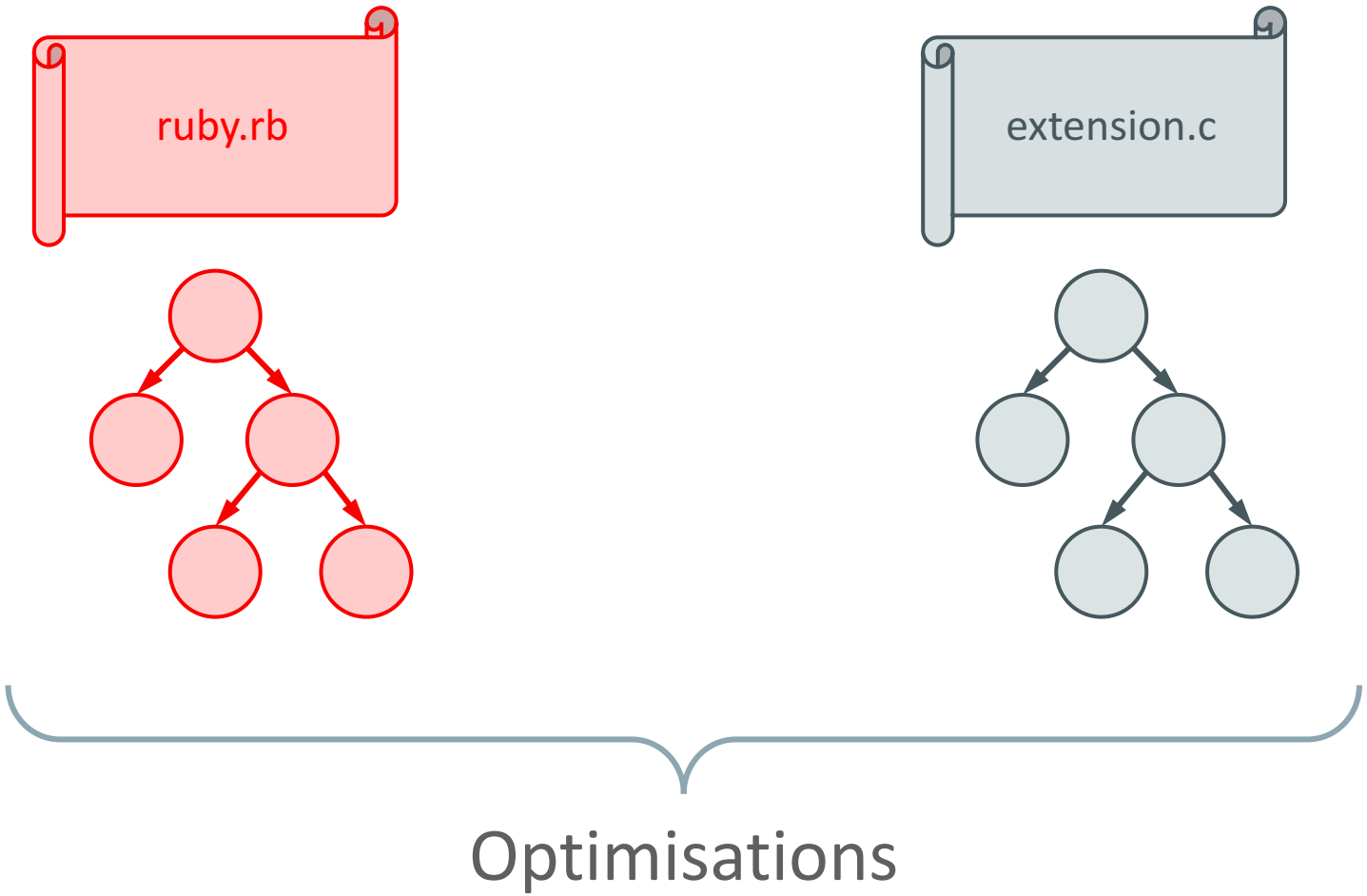
```
define i8* @psd_native_util_clamp(i8* %self,
    i8* %r_num, i8* %r_min, i8* %r_max) nounwind uwtable ssp {
  %1 = call i32 @FIX2INT(i8* %r_num)
  %2 = call i32 @FIX2INT(i8* %r_min)
  %3 = call i32 @FIX2INT(i8* %r_max)
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5                                       ; 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
}
```

```
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5
  br label %12
; <label>:6
  %7 = icmp slt i32 %1, %2
  br i1 %7, label %8, label %9
```

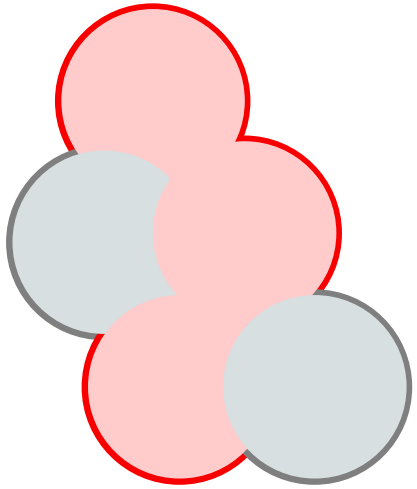
```
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5
  br label %12
; <label>:6
  %7 = icmp slt i32 %1, %2
  br i1 %7, label %8, label %9
```

```
    t4 = t1 > t3
    if t4
      goto l5
    else
      goto l6
    end
l5: goto l12
l6: t7 = t1 < t2
    if t7
      goto l8
    else
      goto l9
    end
```

Optimise Ruby and C together



Optimise Ruby and C together



Optimisations

Interesting problems and their solutions

Defining the C extension API in Ruby

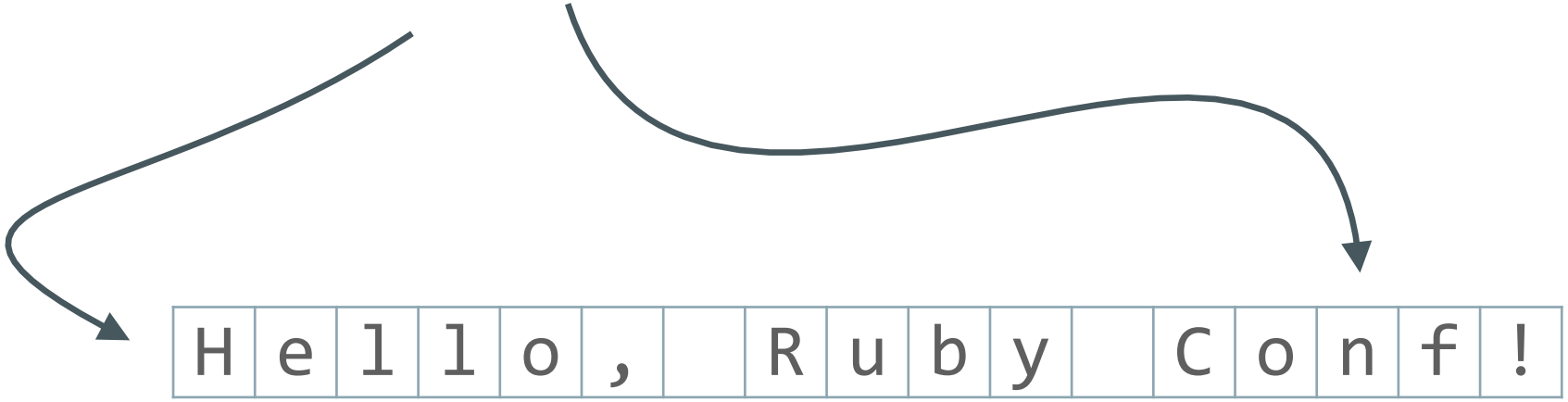
```
int FIX2INT(VALUE value);
```

```
int FIX2INT(VALUE value) {
  return truffle_invoke_i(RUBY_CEXT, "FIX2INT", value);
}
```

```
module Truffle::CExt

  def rb_fix2int(value)
    if value.nil?
      raise TypeError
    else
      int = value.to_int
      raise RangeError if int >= 2**32
      int
    end
  end

end
```

Imaginary strings

```
char *chars = RSTRING_PTR(my_string);
chars[14]
```



Imaginary strings

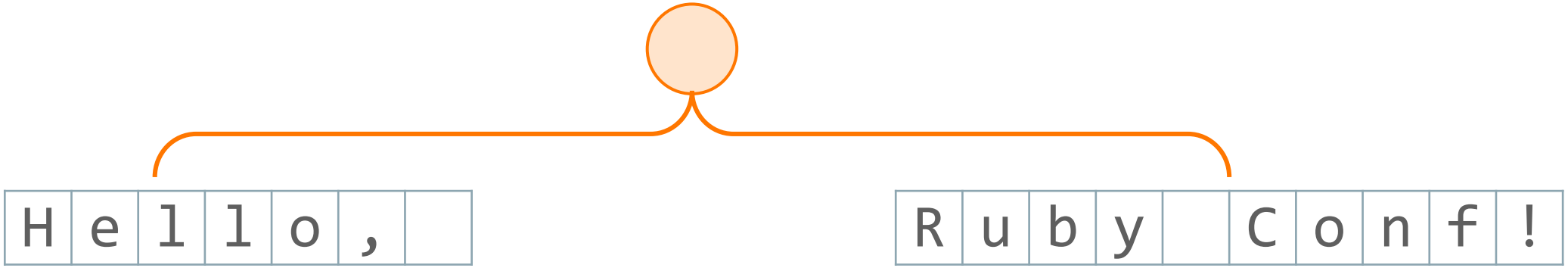
A Tale of Two String Representations

Kevin Menard - RubyKaigi 2016

Imaginary strings

```
%1 = call @RSTRING_PTR(%my_string)
%2 = getelementptr %14, 14
```

```
char *chars = RSTRING_PTR(my_string);
chars[14]
```

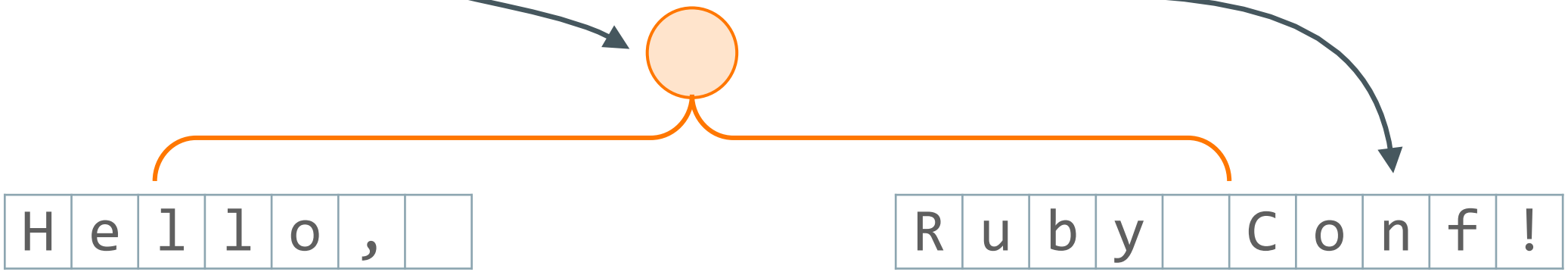


Imaginary strings

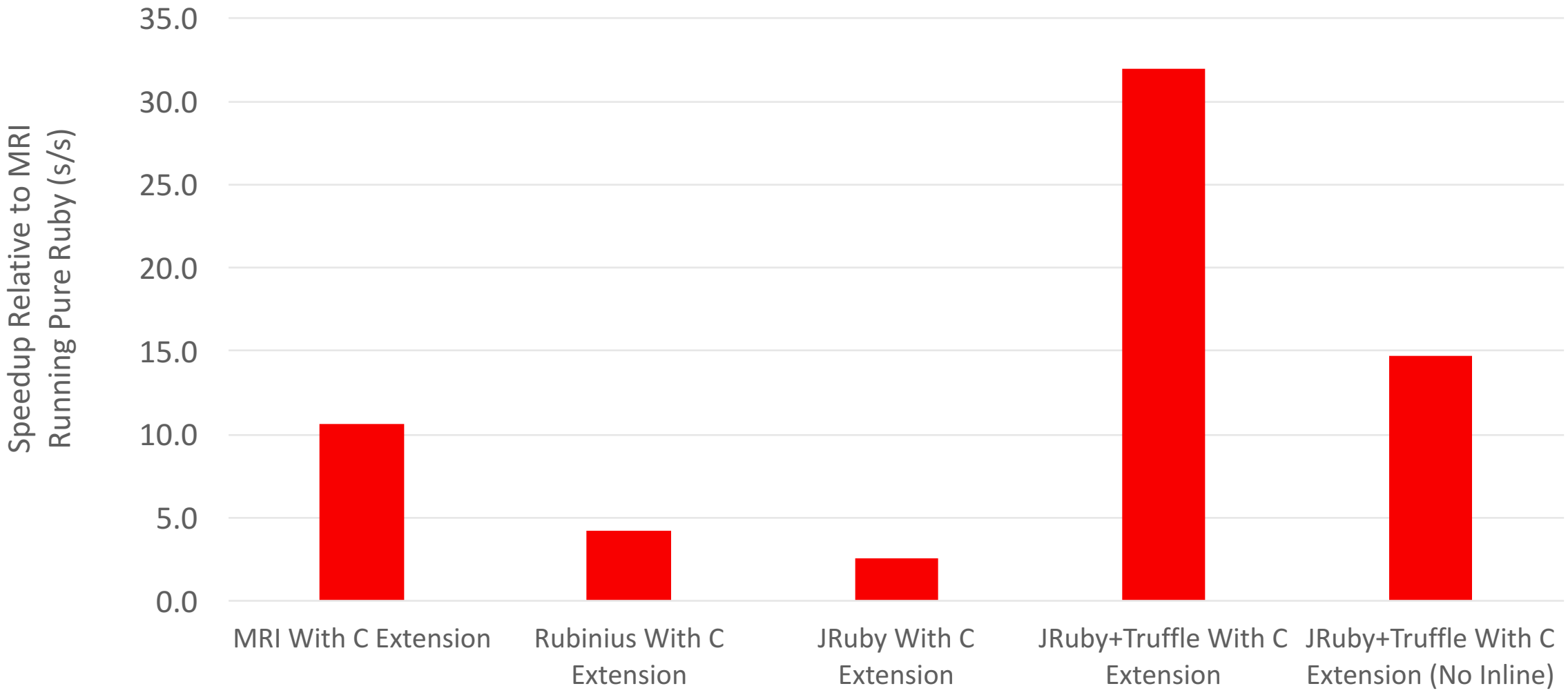
```
%1 = call @RSTRING_PTR(%my_string)
%2 = getelementptr %14, 14
```

String#[]

```
char *chars = RSTRING_PTR(my_string);
chars[14]
```



Results

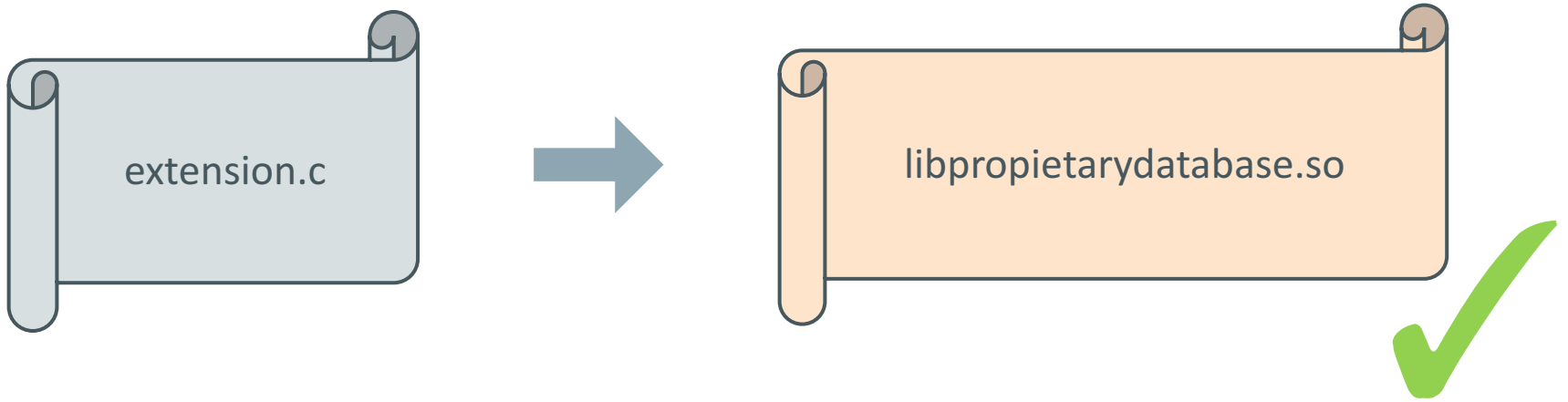


Matthias Grimmer, Chris Seaton, Thomas Wuerthinger, Hanspeter Moessenboeck:
Dynamically Composing Languages in a Modular Way: Supporting C Extensions for Dynamic Languages
Modularity '14 Proceedings of the 14th International Conference on Modularity

Some limitations

You do need the source code of the C extension

- Means no closed source C extensions
  - Is this a problem in reality for anyone?
  - I'm not aware of any closed source C extensions
  - C extensions in turn using closed source libraries like database drivers is fine



You can’t store pointers to Ruby objects in native code

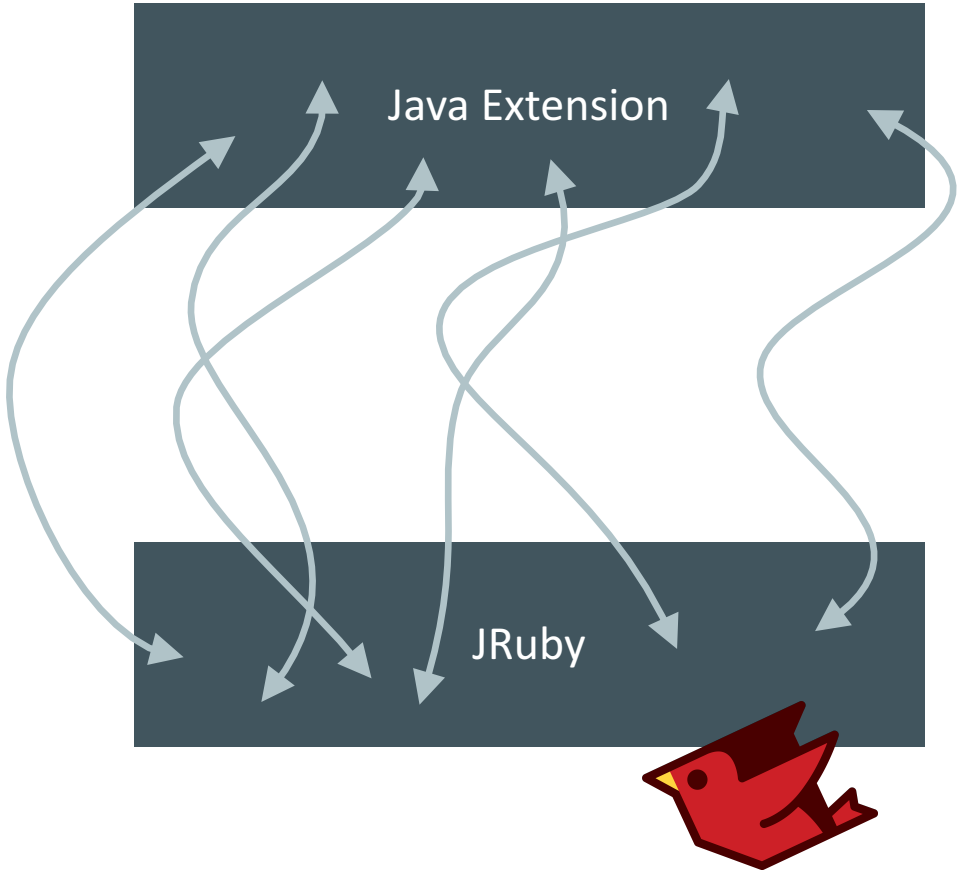
- If your C extension uses a compiled library, such as libssl.so
  - You can’t give that compiled library a reference to a Ruby object
  - The Ruby object may not really exist
  - The GC may want to move the object

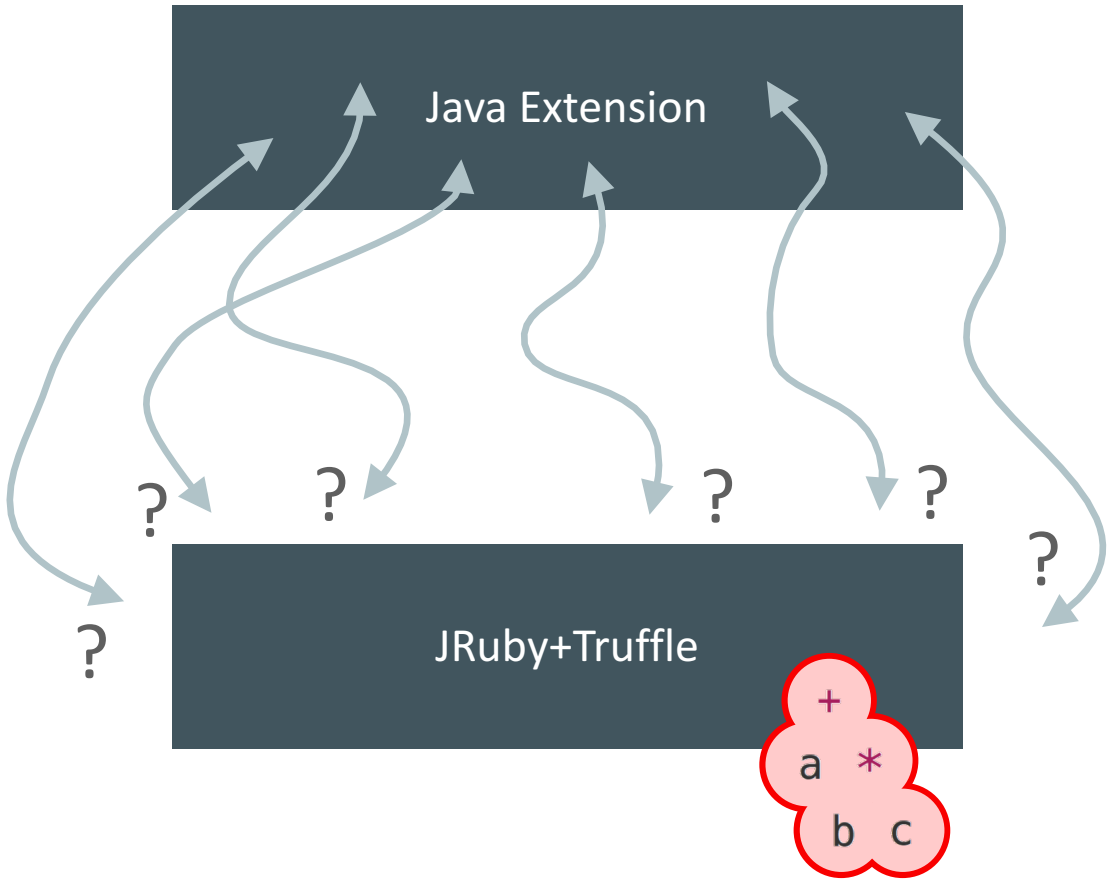
```
void *rb_jt_to_native_handle(VALUE managed);
VALUE rb_jt_from_native_handle(void *native);

SSL_CTX_set_ex_data(ctx, ossl_ssl_ex_ptr_idx, obj);

SSL_CTX_set_ex_data(ctx, ossl_ssl_ex_ptr_idx, rb_jt_to_native_handle(obj));
```

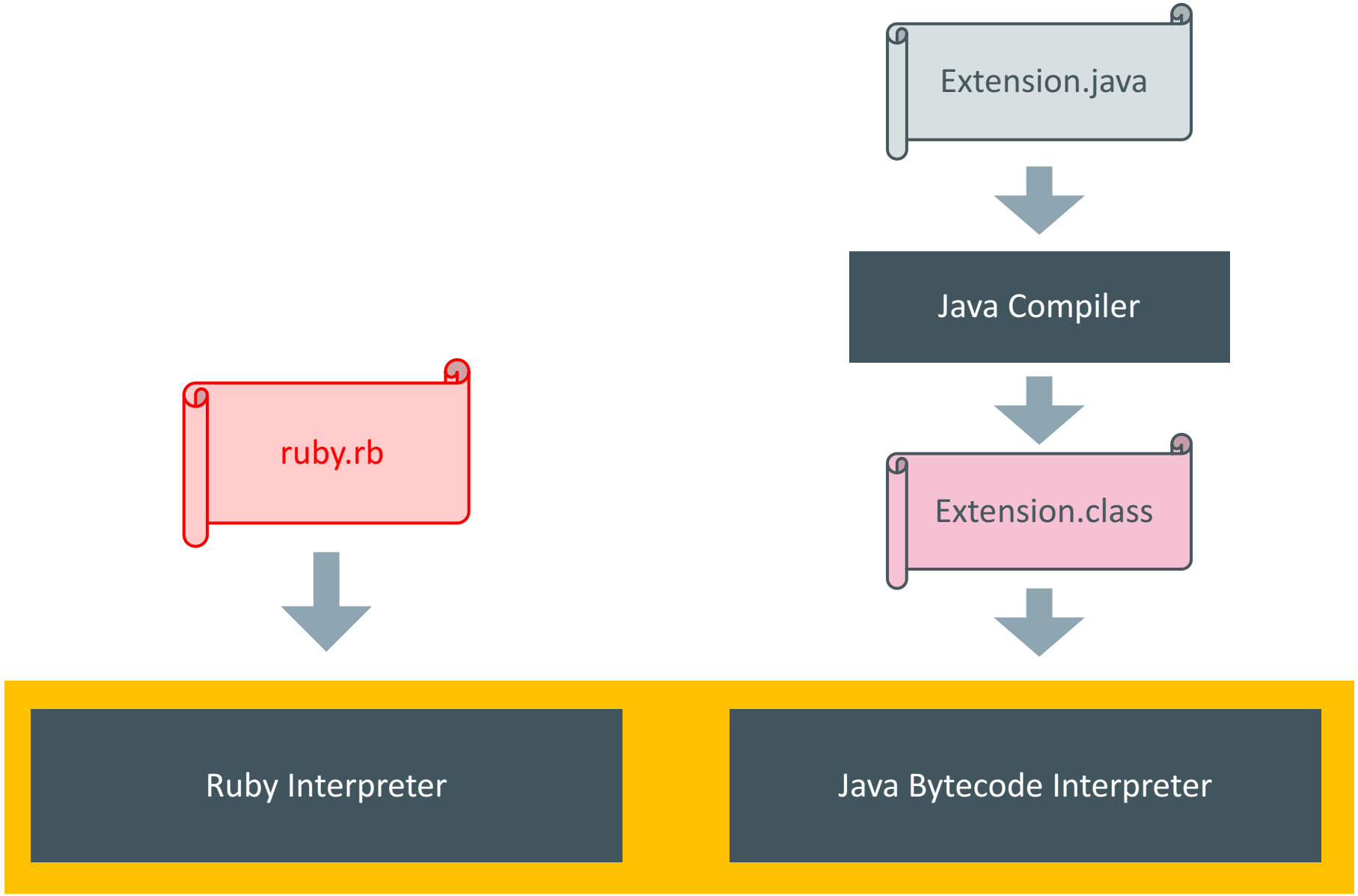
By the way…

- It is probably still best to use the FFI if you are writing new extensions
  - Wide support across Ruby implementations
  - Although we don't actually implement the FFI in JRuby+Truffle yet
  - Implementing the FFI in JRuby+Truffle would be a great internship project!
- If you do write a C extension for performance
  - Write a pure Ruby baseline version as well
- Or if you just needed better performance:
  - Write pure Ruby code
  - Run with JRuby+Truffle

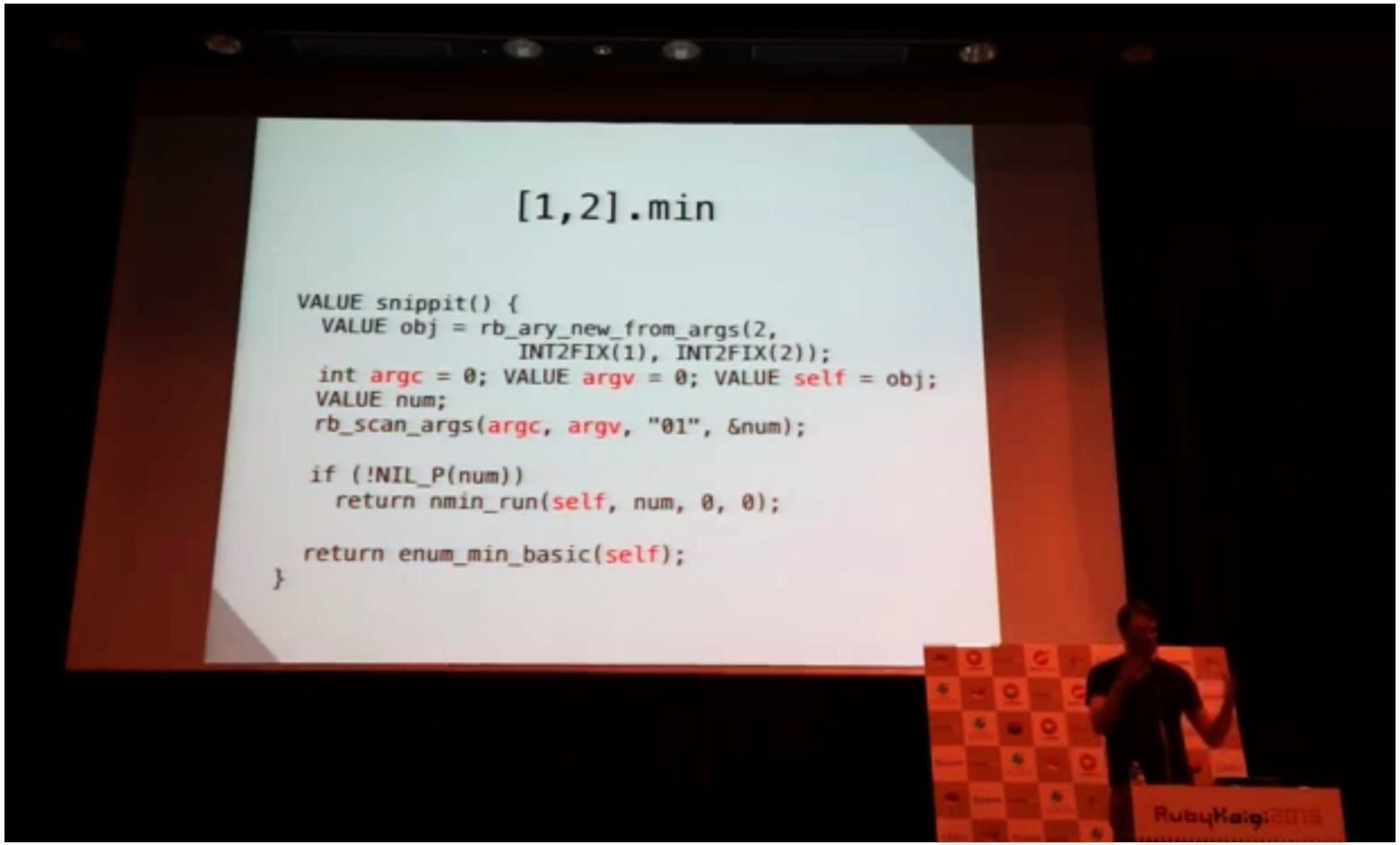
Java extensions







This could be a direction for MRI as well

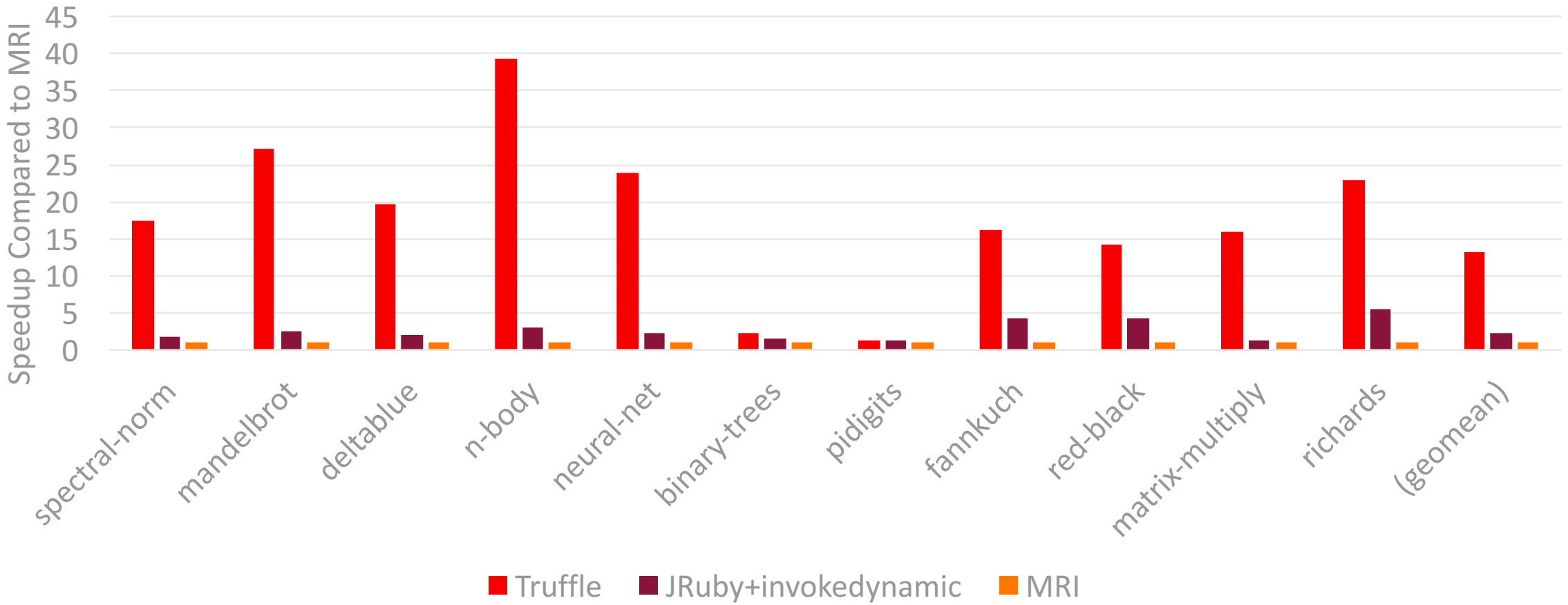


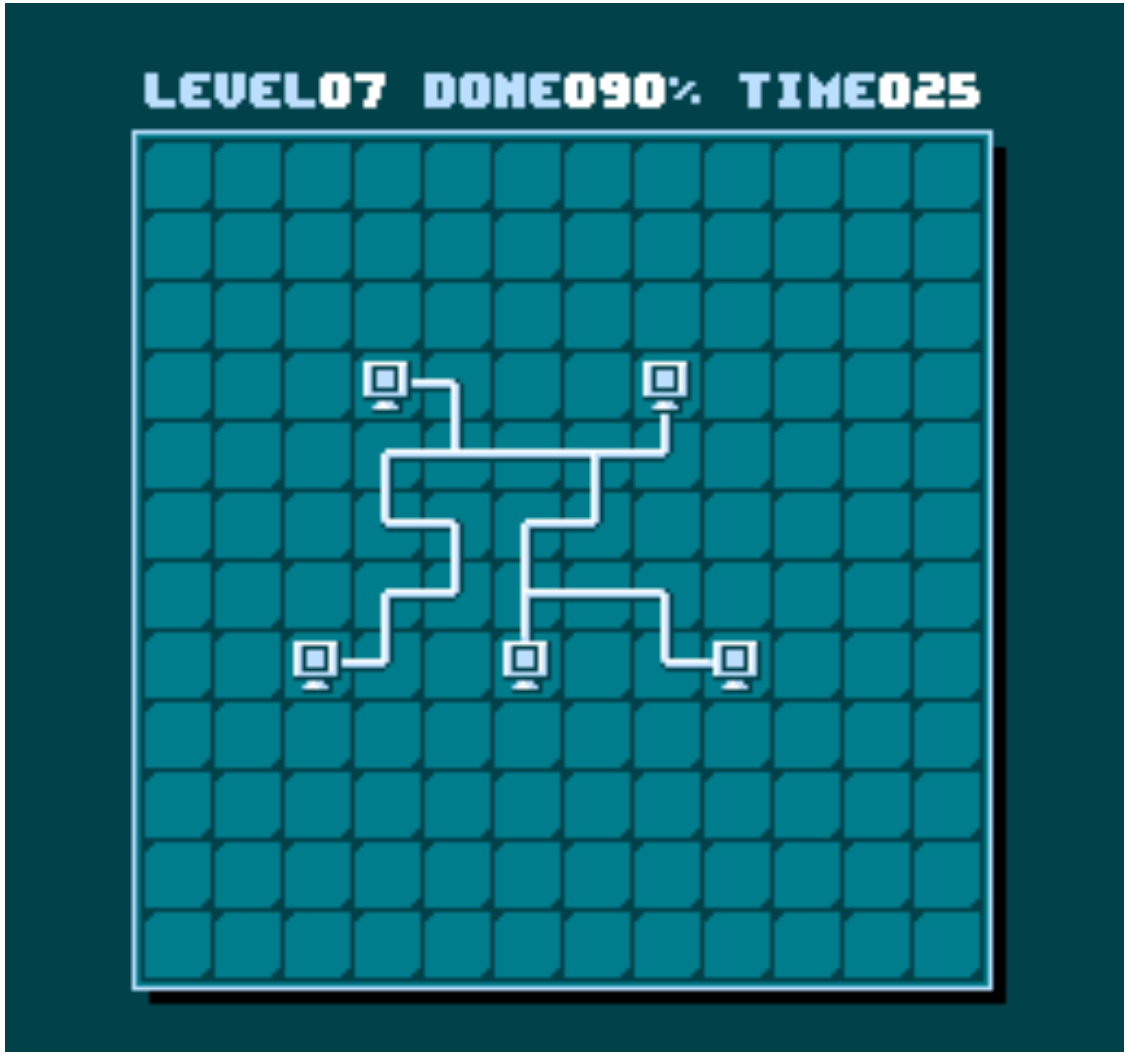
Evan Phoenix: store the LLVM IR of the MRI implementation code and JIT it

Ruby: 2020 - RubyKaigi 2015 Keynote

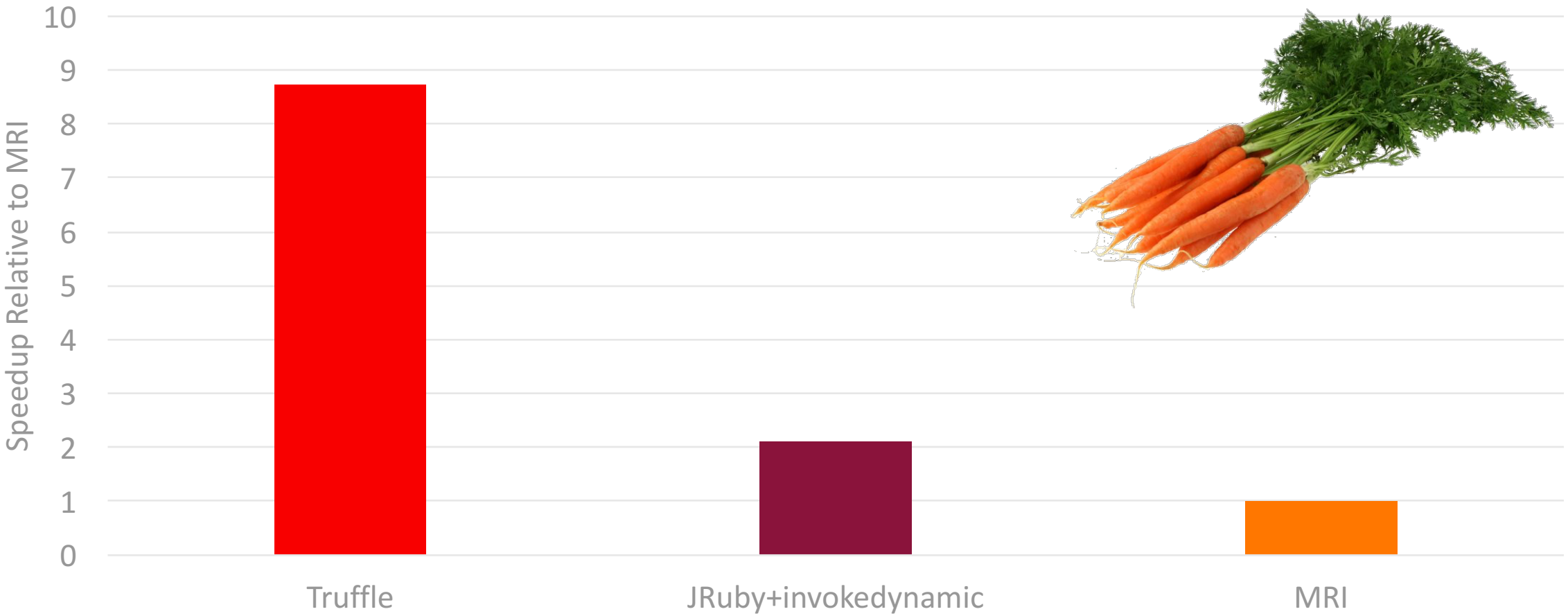
A quick status update on JRuby+Truffle

Classic research benchmarks – 10-20x faster than MRI





‘optcarrot’ NES emulator benchmark – 9x faster than MRI



Ruby specs

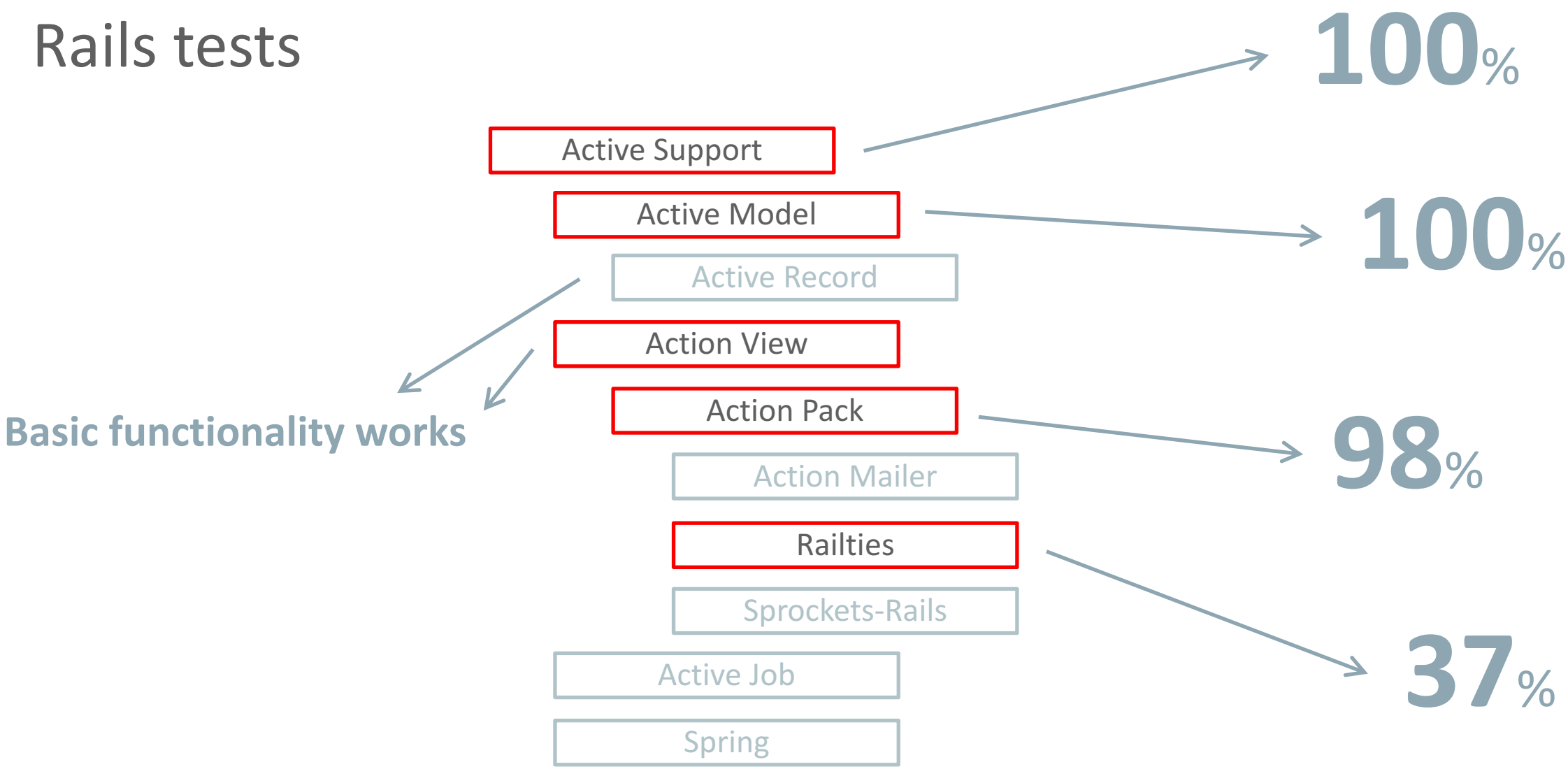
99% Language specs

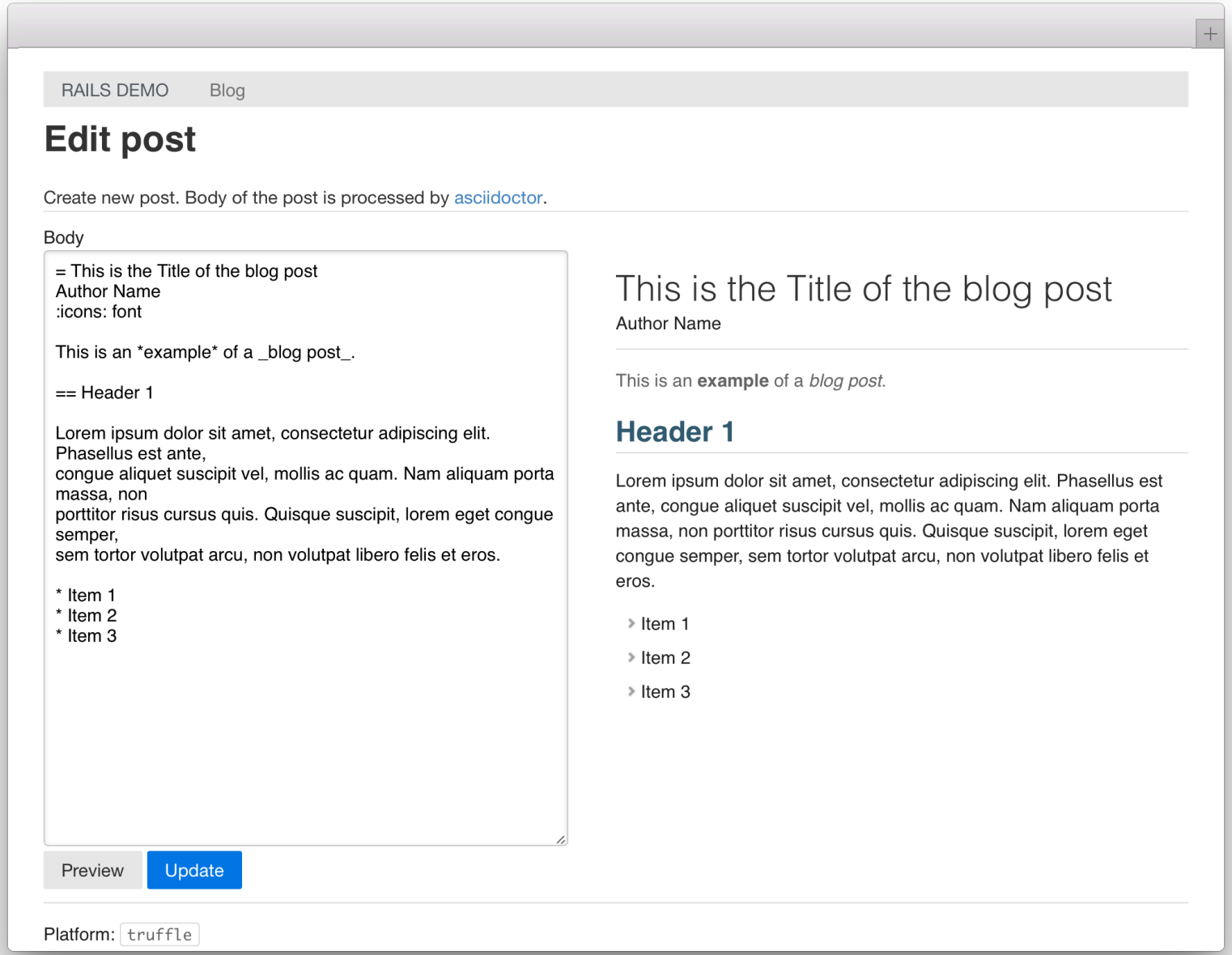
96% Core library specs

78% Standard library specs*

coverage is very limited here; probably a bit misleading

Rails tests

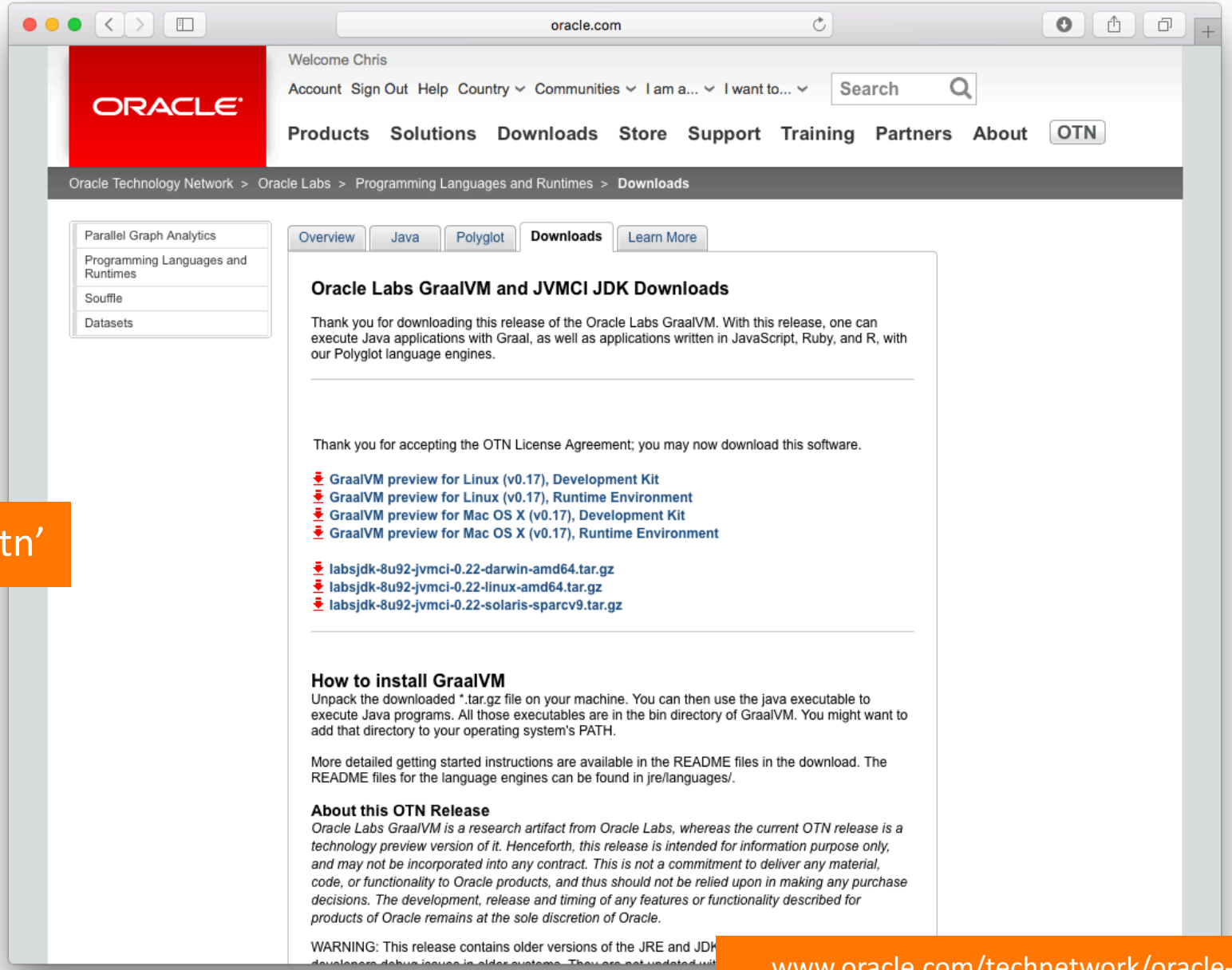




So then why can’t we run real applications yet?

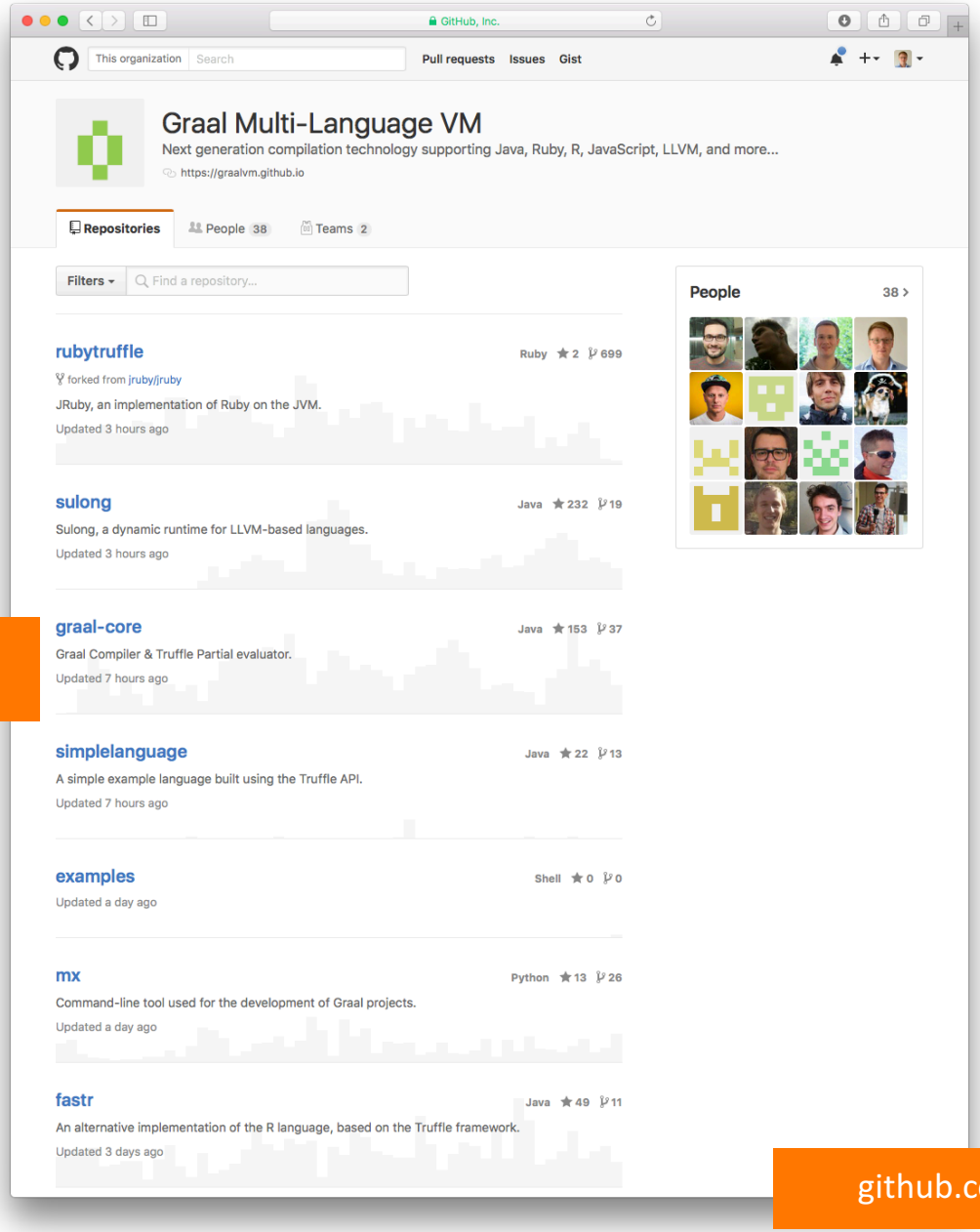
- C extensions are still a work in progress
  - Almost no database drivers
  - No openssl
  - No nokogiri
  - Prevents us running almost everything unfortunately
- The specs don’t have perfect coverage
- Our sophisticated optimisations mean the program state space is huge
  - Lots more to test
  - Lots more to tune for performance

Search for ’graal otn’



www.oracle.com/technetwork/oracle-labs/program-languages

Search for ‘github graalvm’



github.com/graalvm

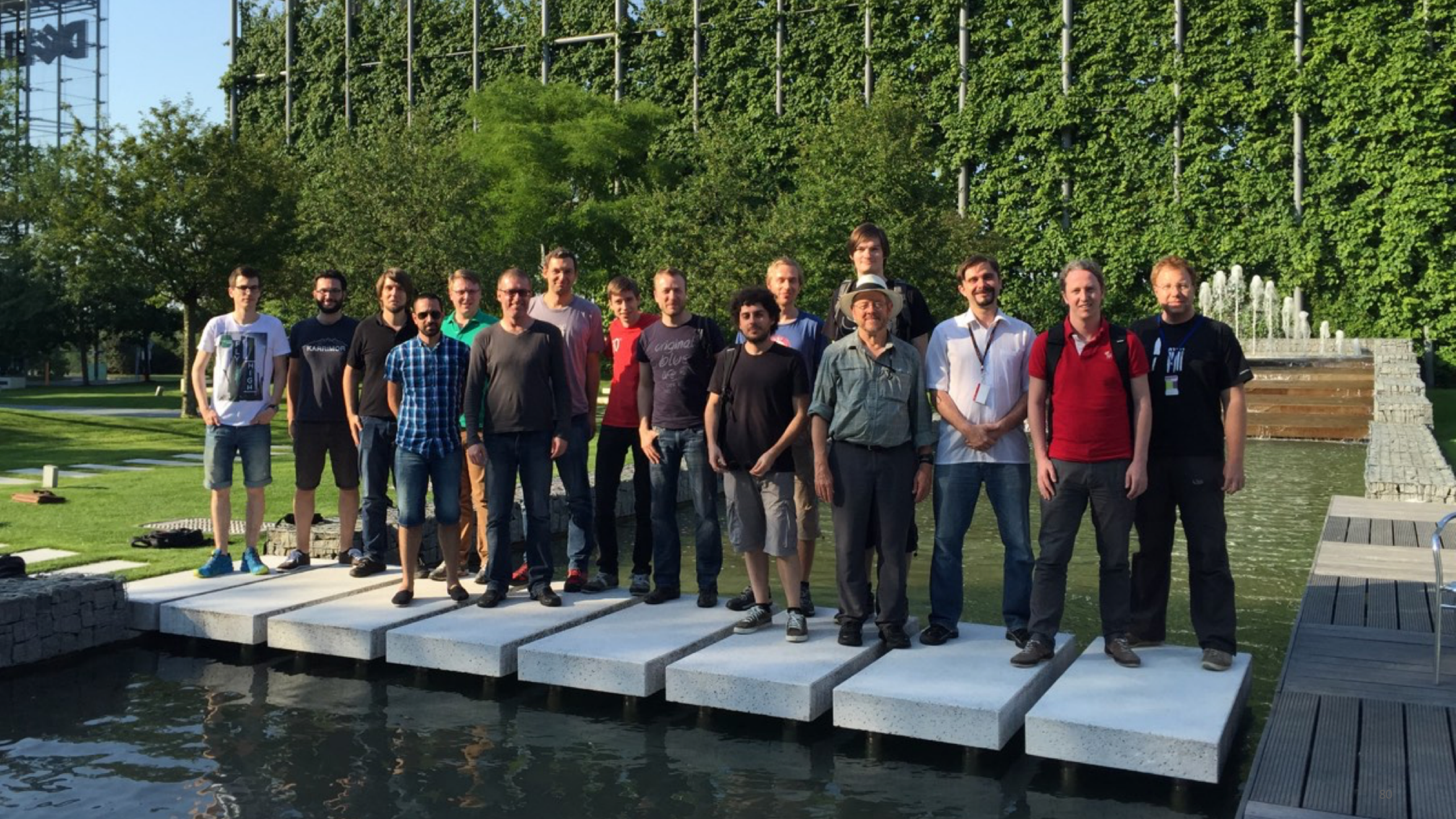
chrisseaton.com/rubytruffle

Freenode #jruby

gitter.im/jruby/jruby

@ChrisGSeaton

‘jruby truffle’



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®