
ORACLE®

# What Language Would You Like to Run?

Michael Haupt
Oracle Labs
March 2015

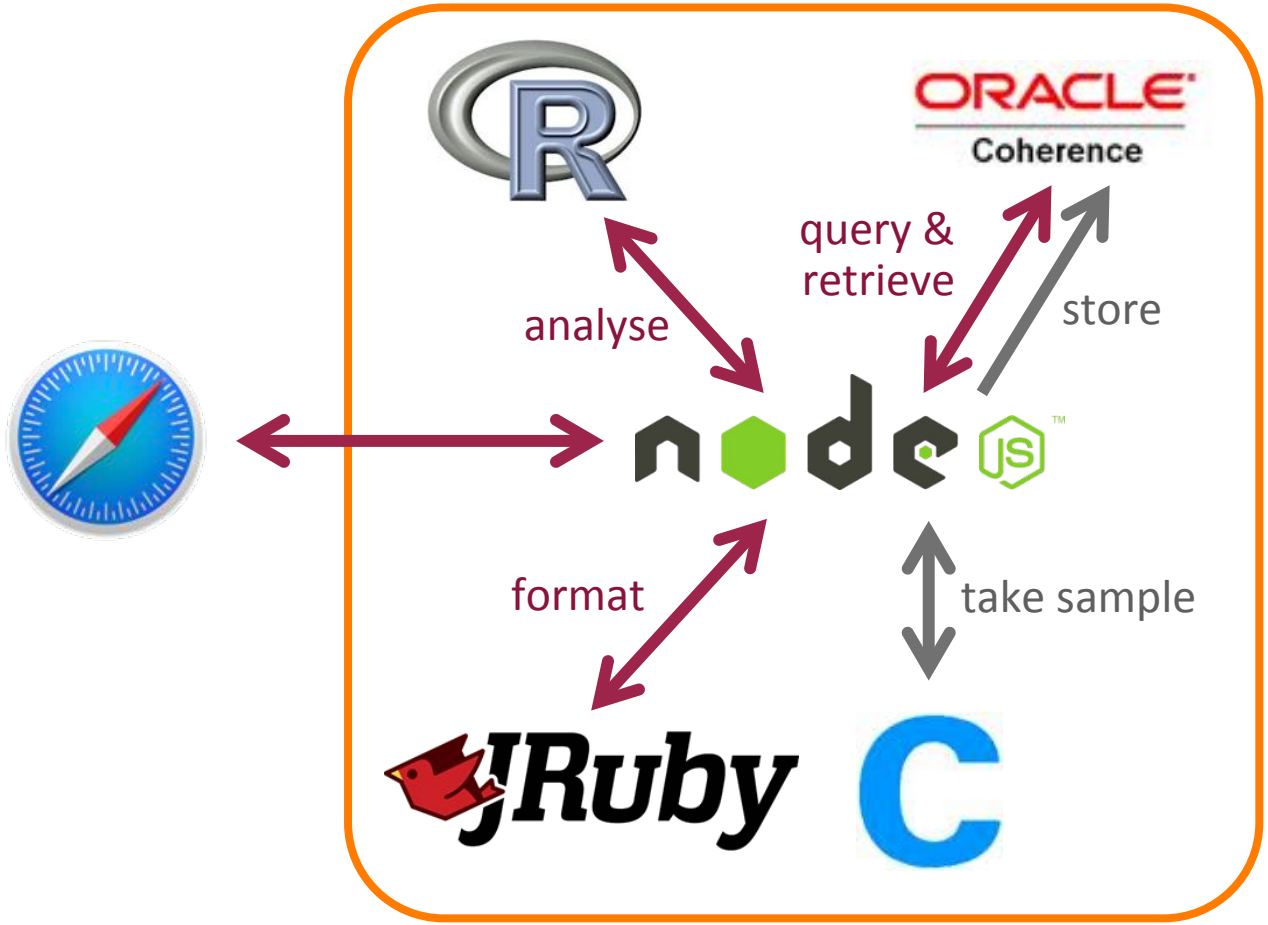
ORACLE

# 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.

# Demo!

# What We Have Seen ...




**... all in one runtime engine.**

# Spot the Glue

**Lines of code (not counting comments): JavaScript: ~100, C: ~40, R: ~10, Ruby: ~30**

```
// set up Coherence
var CoherenceCacheFactory = Java.type('com.tangosol.net.CacheFactory')
CoherenceCacheFactory.ensureCluster()

var coherenceCache = CoherenceCacheFactory.getCache('javaland-demo'),
    Q = Java.type('com.tangosol.util.QueryHelper')

// query
var query = Q.createFilter("..."),
    result = coherenceCache.entrySet(query)
```

# JavaScript / C Interoperability

```
// C code
struct _sample {
    int vm_size;
    int resident_set_size;
    ...
};

struct _sample* take_sample(...) {
    struct _sample *sample = (struct _sample*) malloc(sizeof(struct _sample));
    ...
    return sample;
}
```

```
// take sample and store in cluster
var sample = take_sample()
coherenceCache.put(..., sample.vm_size)
```

# JavaScript / R / Ruby Interoperability

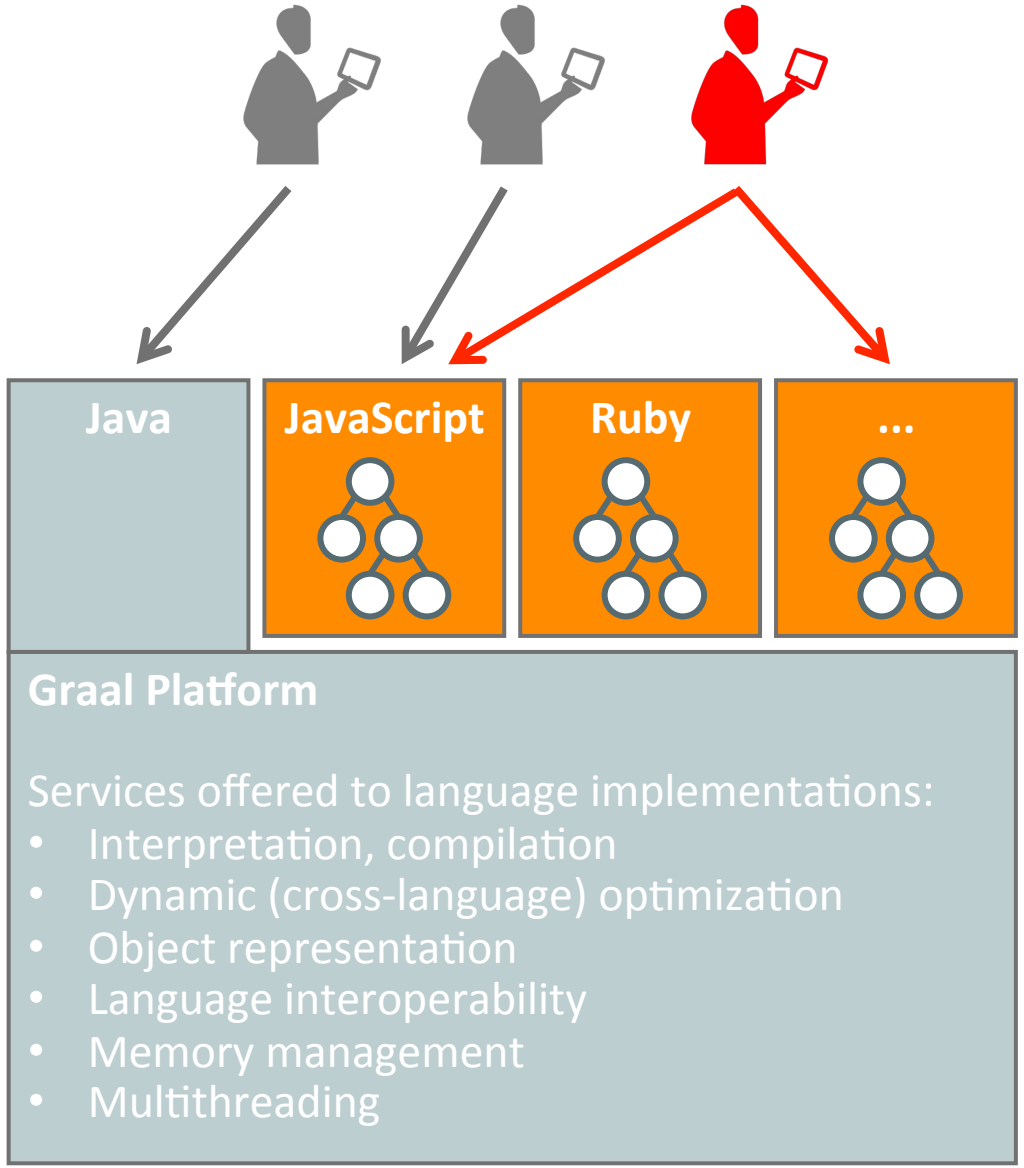
```
// Ruby code
DATA_TEMPLATE = ERB.new %{
  Summary for <%= data.n %> samples:
  <ul>
    <li>Range [<%= data.min %>..<%= data.max %>]</li>
    ...
  </ul>
}

def formatData(data)
    DATA_TEMPLATE.result(binding)
end
```

```
// R code
sample_stats <- function(receiver, data) {
    dn <- length(data)
    dmin <- min(data)
    dmax <- max(data)
    dmean <- mean(data)
    dsd <- sd(data)
    return(c(n=dn, min=dmin, max=dmax,
        mean=dmean, sd=dsd))
}
```

```
// JavaScript code
var vms = sample_stats(query("..."))
return formatData(vms)
```

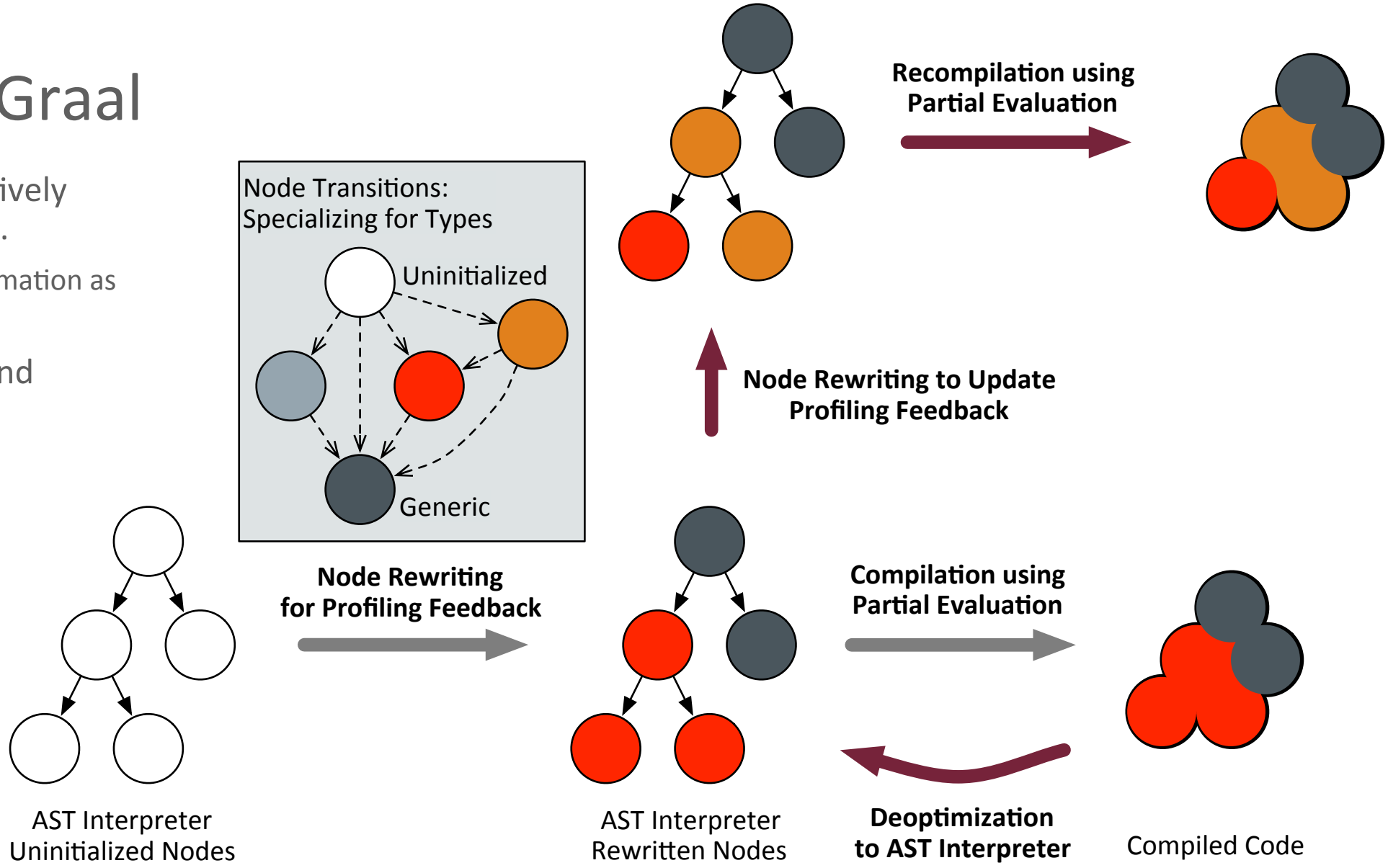
# The Graal Platform

- All languages are treated as equal
  - The bird's eye view is as on a runtime environment for the language(s) at hand
  - Java is implicitly part of the platform, but invisible unless required
- Language interoperability
  - Java interoperability comes for free
  - All optimization works transparently across language boundaries



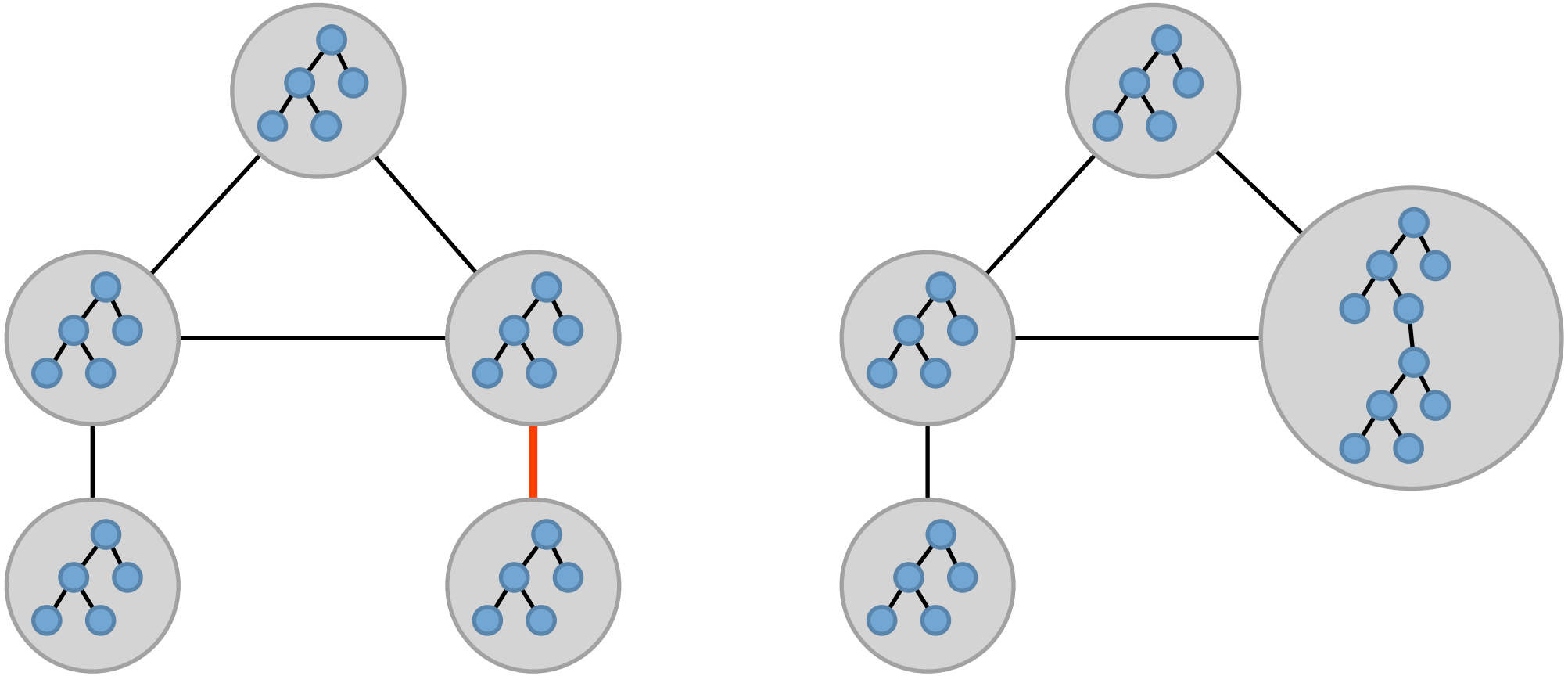
# Truffle and Graal

- Profile and speculatively optimize the code ...
  - Learn the type information as the program runs
- ... and deoptimize and reoptimize!
- Compiler has no dependencies on language type system



# Cross-Language Optimisation at the AST Level

## Inlining Across Language Boundaries

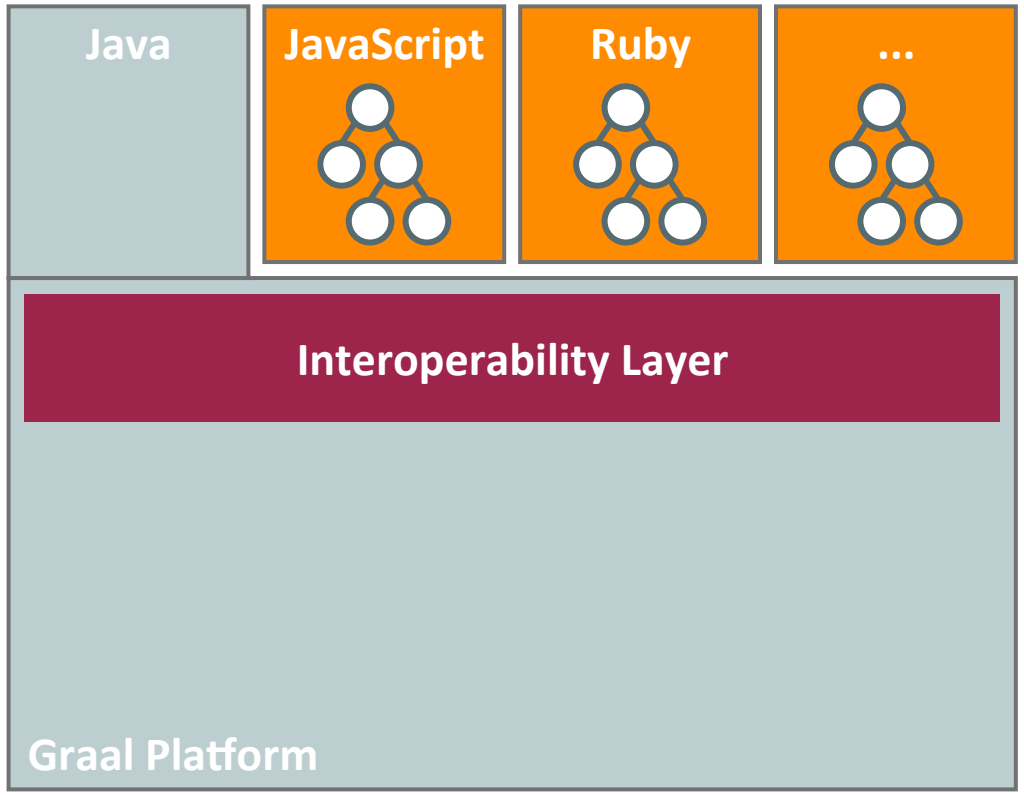


# Graal Compiler and Language Creation APIs

- Graal is a dynamic compiler written in Java that compiles Java
- Graal APIs allow you to write an interpreter for your language, and Graal will specialize it into a compiler for you
  - Learns types an application uses by executing the interpreter
  - Normally compilers are 100's of man-years to achieve quality
- Graal can compile itself
  - This makes it easy to add efficient optimizations at the simpler interpreter level
  - Supports classes of specializations or "fast paths" through code
- Speculates on data structures and specialized code paths
  - On exceptions, will deoptimize to interpreter and reoptimize

# Language Interoperability Architecture

- Graal platform provisions
  - Common abstractions
  - ASTs: abstraction for execution logic
- Shared interface for interoperability
  - Execution (calls)
  - Access (objects, fields, variables)
- Guest language implementations implement the interop interface



# Supported Languages and Performance Characteristics

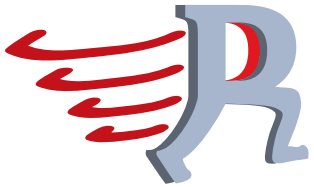
- Java
  - About as fast as HotSpot™ server compiler (SPECjvm2008, SPECjbb2013)
- JavaScript (ECMAScript 5.1)
  - About as fast as V8 (Octane, Node.js)
- Ruby
  - 13x MRI (synthetic and image processing benchmarks)
- R
  - 10x GNU R (shootout)
- C
  - Ca. 2x gcc -O0, within 10 % of best gcc/ clang performance (shootout)
  - Ruby+C: 3x MRI+C, 32x MRI
- Not maintained by Oracle
  - Python (ZipPy, UCI)
  - Smalltalk (SOM, Stefan Marr)

# A Word About Debugging

- Usual approach: debugging as an afterthought
- Graal platform has language-agnostic debugging capabilities


```
(JavaScript) break ../eg/demo.js:6
==> breakpoint set at demo.js:6
(JavaScript) loadr
    1  function demo(x, n) {
    2      result = x;
    3      for(var i = 0; i < n; ++i) {
    4              result = result + i;
    5      }
-->  6      return result;
    7  }
    8
    9  b = demo(2, 4);
   10  print("result=" + b);
(<1> demo.js:6) eval result
==> 8
(<1> demo.rb:6)
...
```


```
(Ruby 2.1.0) break ../eg/demo.rb:6
==> breakpoint set at demo.rb:6
(Ruby 2.1.0) loadr
    1  def demo(x, n)
    2    result = x
    3    n.times do |i|
    4      result = result + i
    5    end
-->  6    result
    7  end
    8
    9  b = demo(2, 4)
   10  puts "result=#{b}"
(<1> demo.rb:6) eval result
==> 8
(<1> demo.rb:6)
...
```



```
(Fastr) break ../eg/demo.r:6
==> breakpoint set at demo.r:6
(Fastr) loadr
    1  demo <- function(x, n) {
    2    result <- x
    3    for (i in 1:n-1) {
    4      result <- result + i
    5    }
-->    6   result
    7  }
    8
    9  b <- demo(2, 4)
   10  cat("result=", b, "\n")
(<1> demo.r:6) eval result
==> 8
(<1> demo.r:6)
...
```

# Available Language Implementations

- Java and JavaScript
  http://www.oracle.com/technetwork/oracle-labs/program-languages/overview/index.html
- JRuby+Truffle
  https://github.com/jruby/jruby/wiki/Truffle
- FastR
  https://bitbucket.org/allr/fastr
- Not maintained by Oracle:
  - ZipPy (Python)
    https://bitbucket.org/ssllab/zippy
  - TruffleSOM (Smalltalk)
    https://github.com/smarr/TruffleSOM

# Acknowledgements


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

**Oracle Labs (continued)**
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 Kotselidis

**JKU Linz**
Prof. Hanspeter Mössenböck
Benoit Daloze
Josef Eisl
Thomas Feichtinger
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Huber
David Leopoldseder
Manuel Rigger
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
Petr Maj
Lei Zhao

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

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

# Hardware and Software
## Engineered to Work Together


ORACLE®