Truffle: your favorite language on JVM

without compromises

Štěpán Šindelář

geecon

Kraków, 17-19 May 2017

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.

Program Agenda

1. Motivation
2. How Truffle Works
3. Graal Compiler
4. Demos

Implementing another (dynamic) language

Prototype a new language

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

Write a “real” VM

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

People start using it

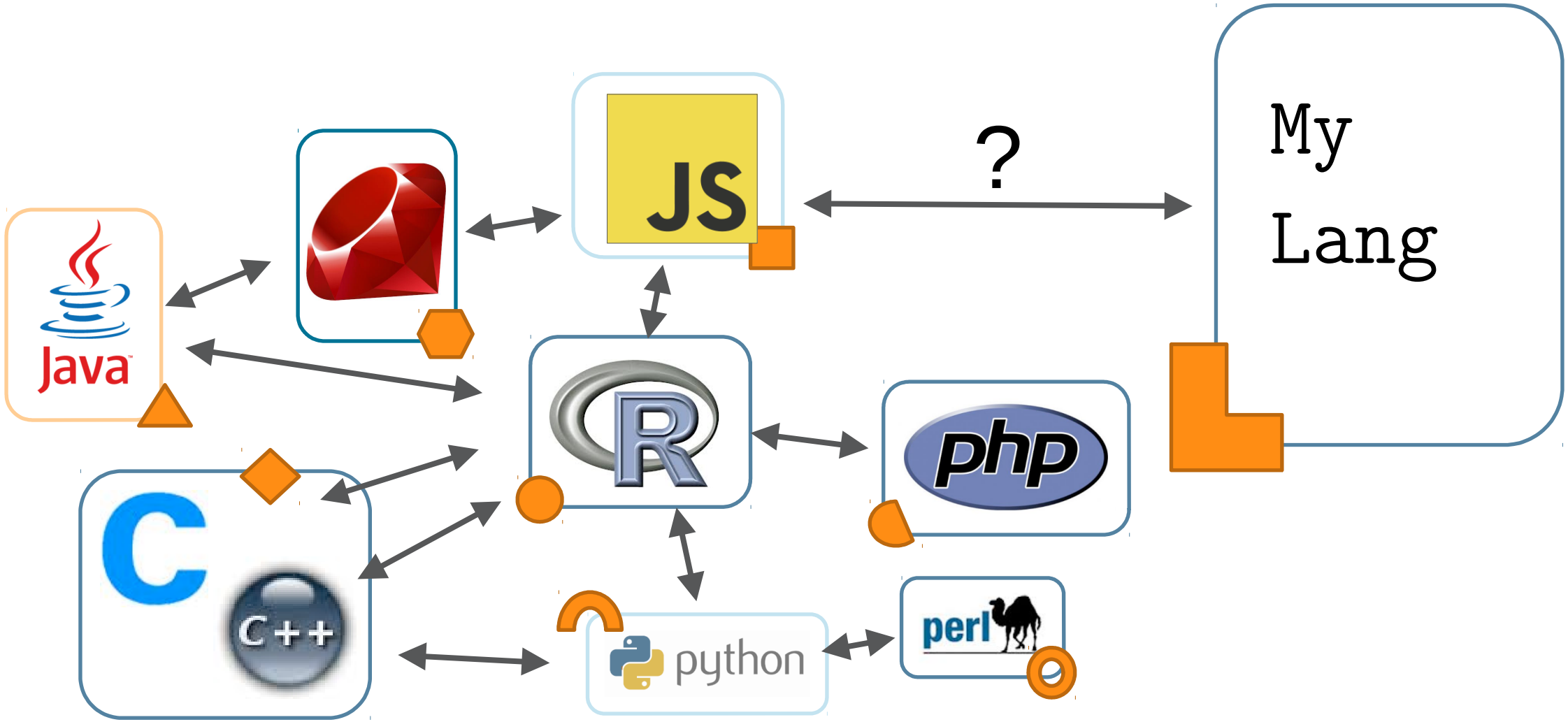
People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector
Write an optimizing JIT compiler

Systems come with Various Interfaces



You can execute any language on the JVM / CLR

You can execute any language on the JVM / CLR

- as long as it looks like Java / C#.

Truffle: write your own language with ease!

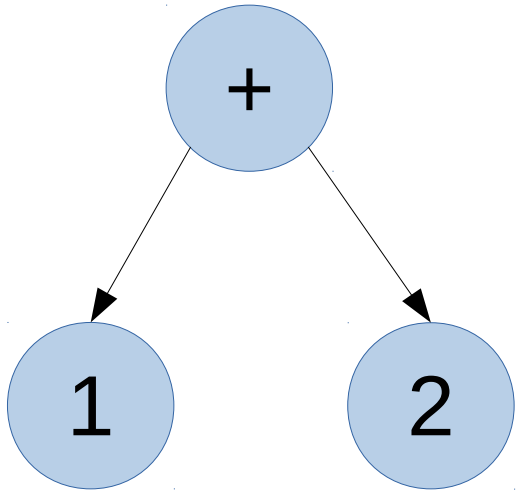
Current situation	How it should be
Prototype a new language	Prototype a new language in Java
Parser and language work to build syntax tree (AST), AST Interpreter	Parser and language work to build syntax tree (AST) Execute using AST interpreter
Write a “real” VM	Integrate with VM-building framework
In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …	Integrate with Modular VM Add small language-specific parts
People start using it	People start using it
People complain about performance	And it is already fast
Define a bytecode format and write bytecode interpreter	
Performance is still bad	
Write a JIT compiler Improve the garbage collector	

Program Agenda

1. Motivation
2. How Truffle Works
3. Graal Compiler
4. Demos

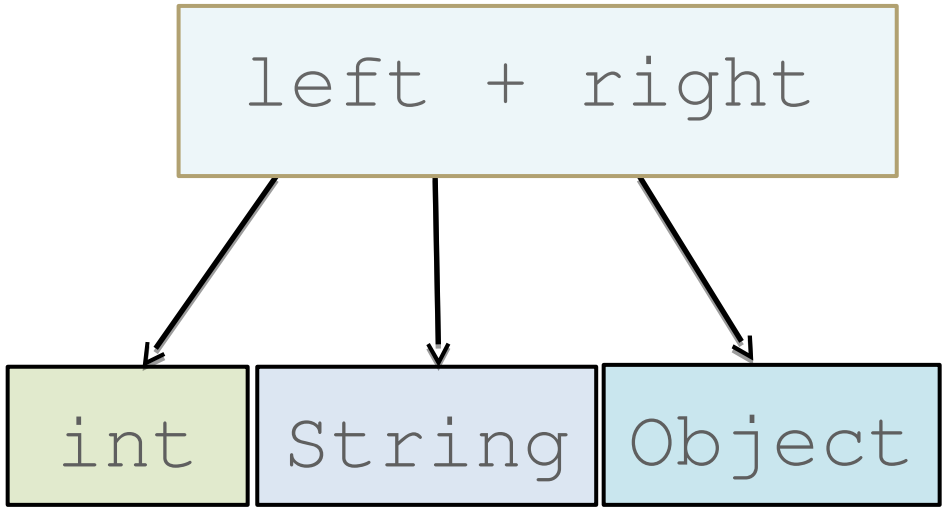
AST Interpreter

```
interface Node { Object execute(); }

class Add implements Node {
  public Node left;
  public Node right;
  @Override
  public Object execute() {
    return left.execute() + right.execute();
  }
}

class Constant implements Node {
  public int value;
  @Override
  public Object execute() { return value; }
}
```



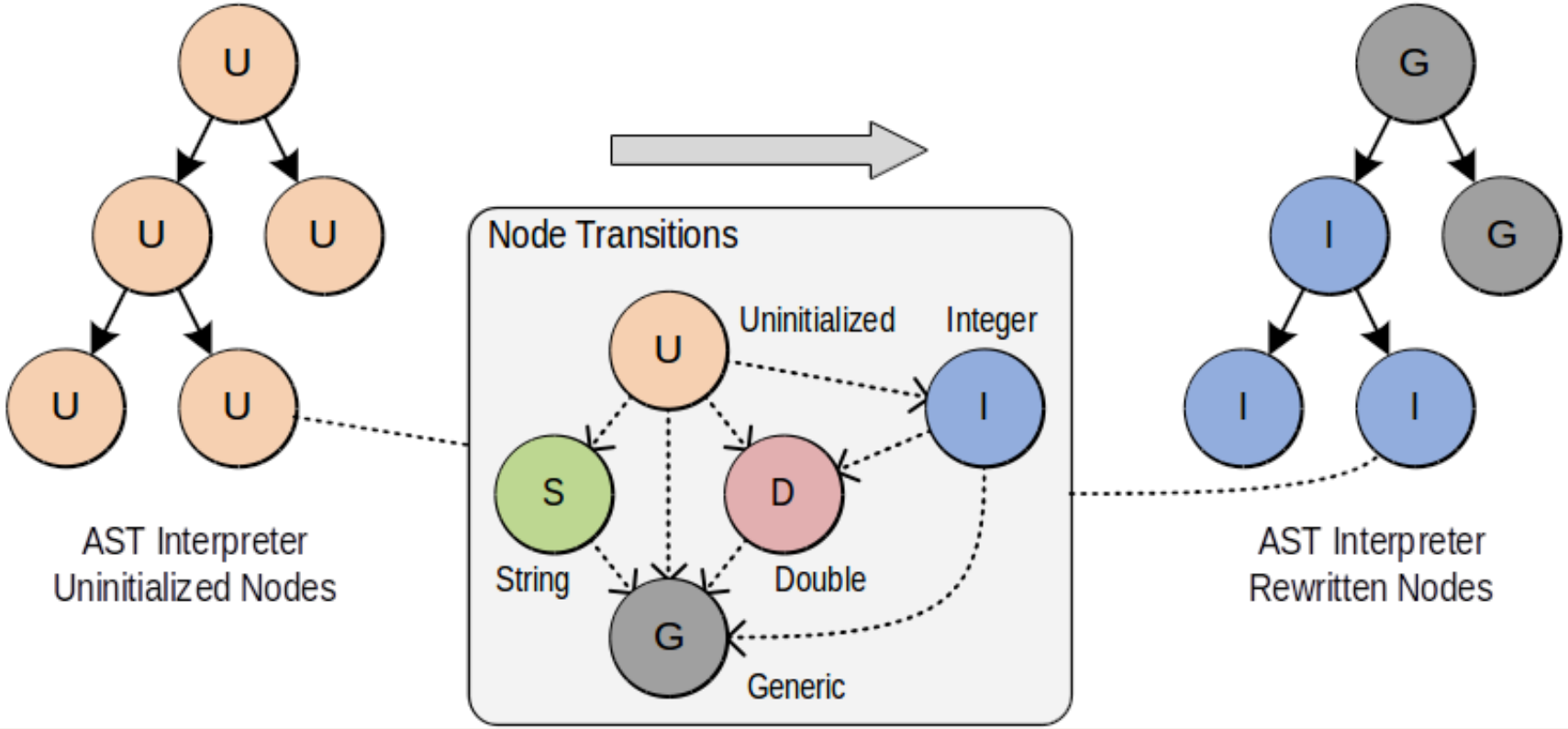
The Biggest Problem of Dynamic Languages

```
interface Node { Object execute(); }

class Add implements Node {
  public Node left;
  public Node right;
  @Override
  public Object execute() {
    return left.execute() + right.execute();
  }
}

class Constant implements Node {
  public int value;
  @Override
  public Object execute() { return value; }
}
```



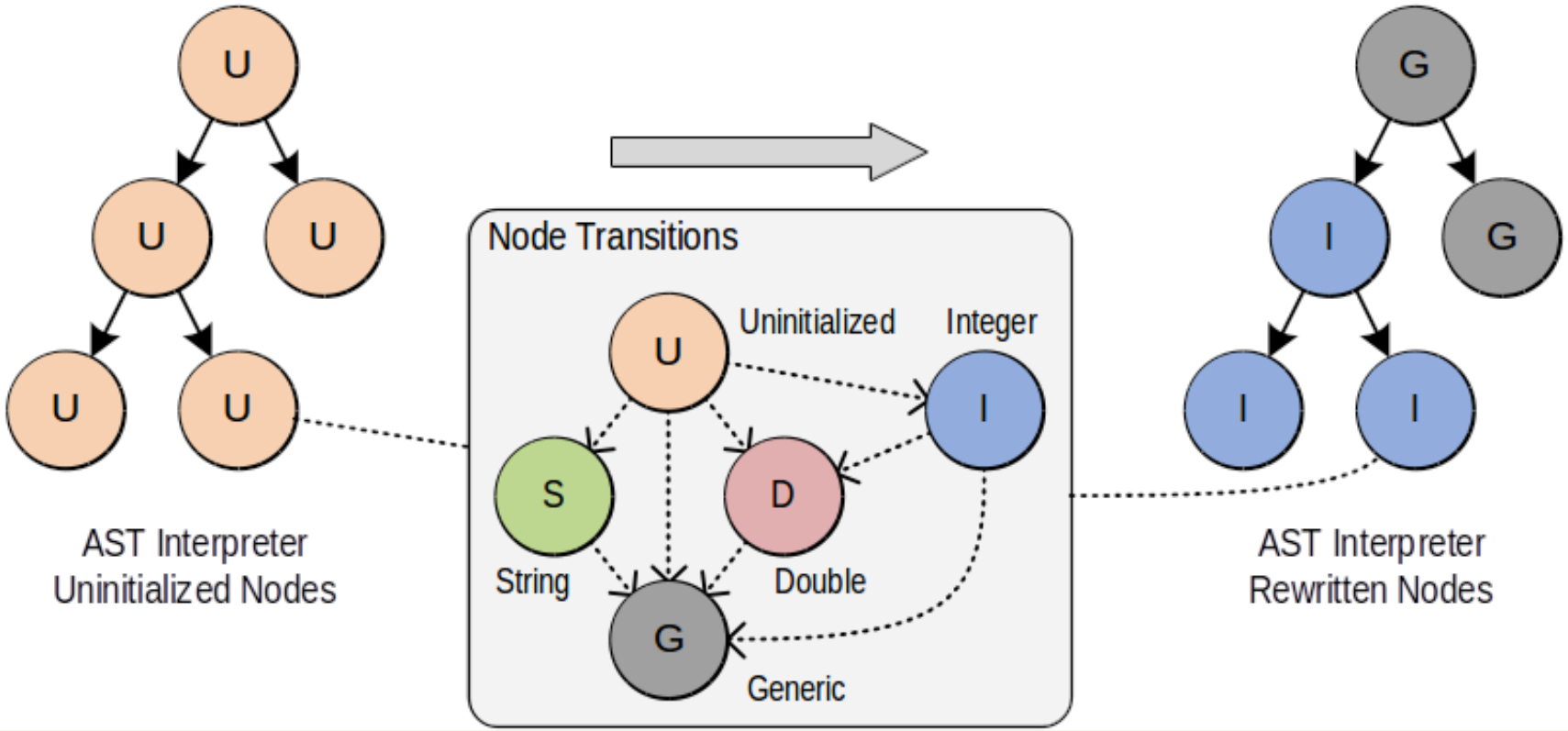
Self Optimizing AST Interpreter

- Unitinitialized node
  - are left and right operands integers => rewrite to integer addition
  - are left and right operands strings => rewrite to string concatenation
  - ...



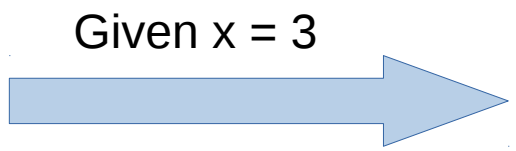
Self Optimizing AST Interpreter

- Integer addition node
  - operands are not of integer types => rewrite the node
  - integer addition



Intermezzo: Partial Evaluation

```
int f(int x, int y){
   return x*42 + y;
}
```




```
int f(int y) {
   return 126 + y;
}
```

Intermezzo: Partial Evaluation

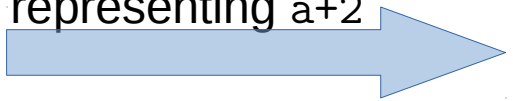
```
int f(int x, int y){
   return x*42 + y;
}
```

Given x = 3

```
int f(int y) {
   return 126 + y;
}
```

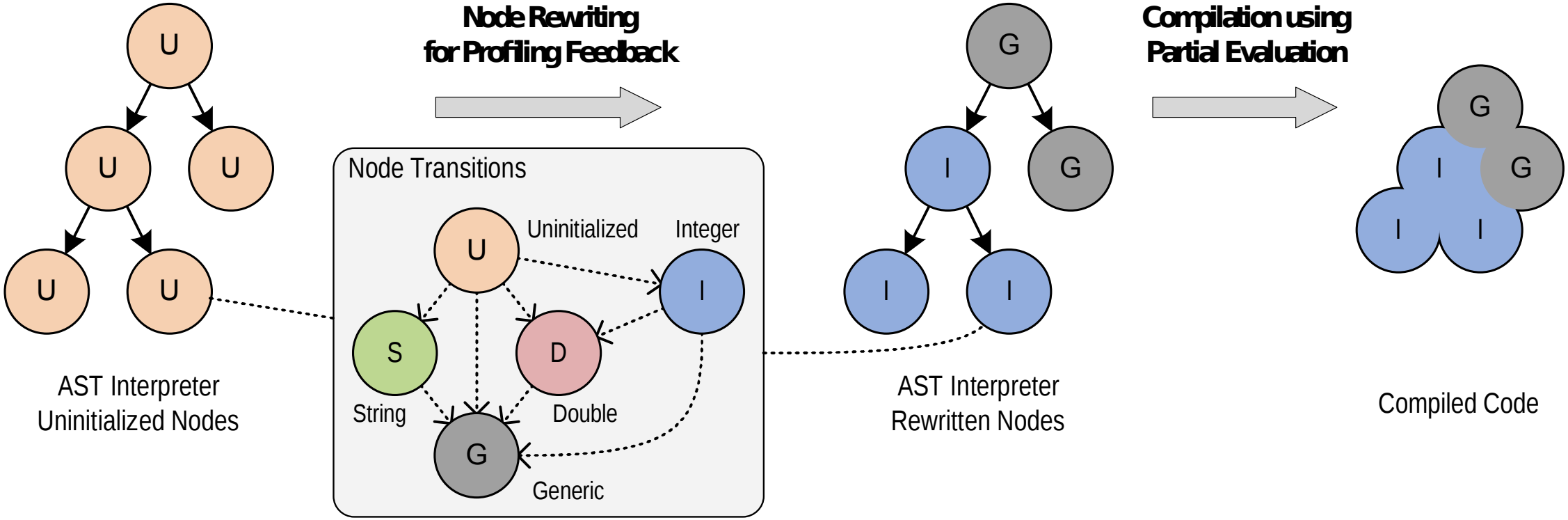
```
run(node, vars){
   node.execute(vars);
}
```

Given a fixed node representing a+2



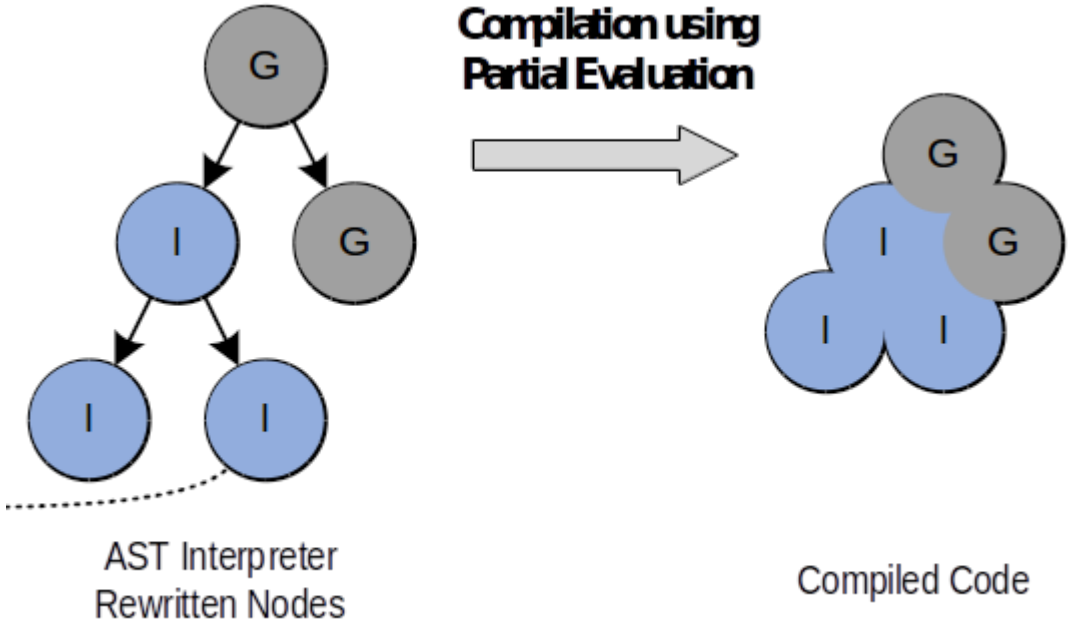
```
run(vars) {
   vars.a + 2;
}
```

Partially Evaluated Self Optimizing AST



Partially Evaluated Self Optimizing AST

- Integer addition node
  - operands are not of integer type => rewrite the node
  - integer addition

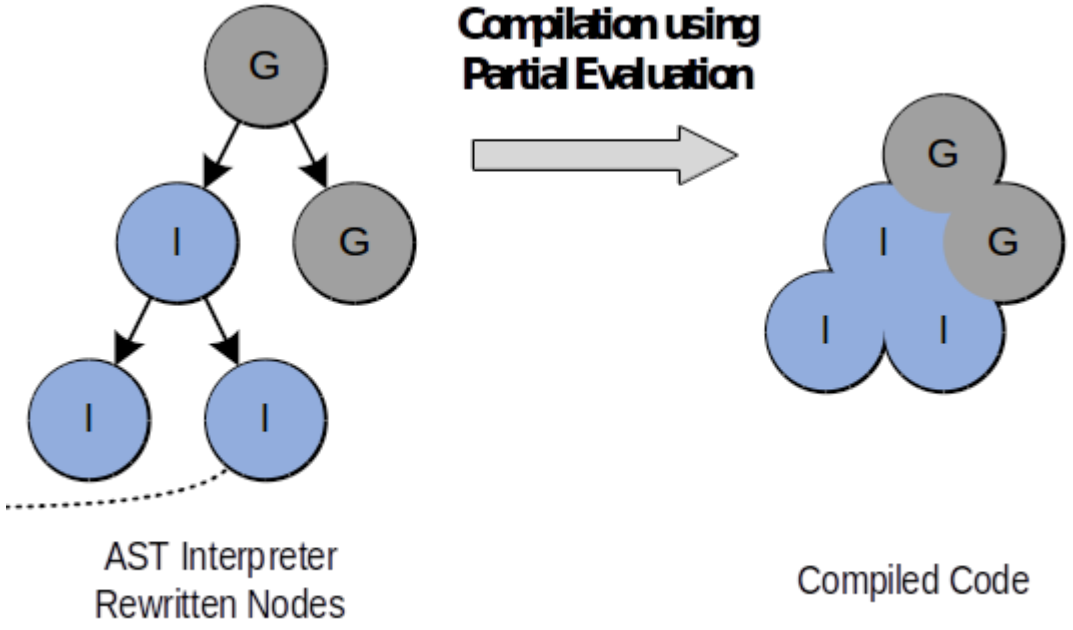


Partially Evaluated Self Optimizing AST

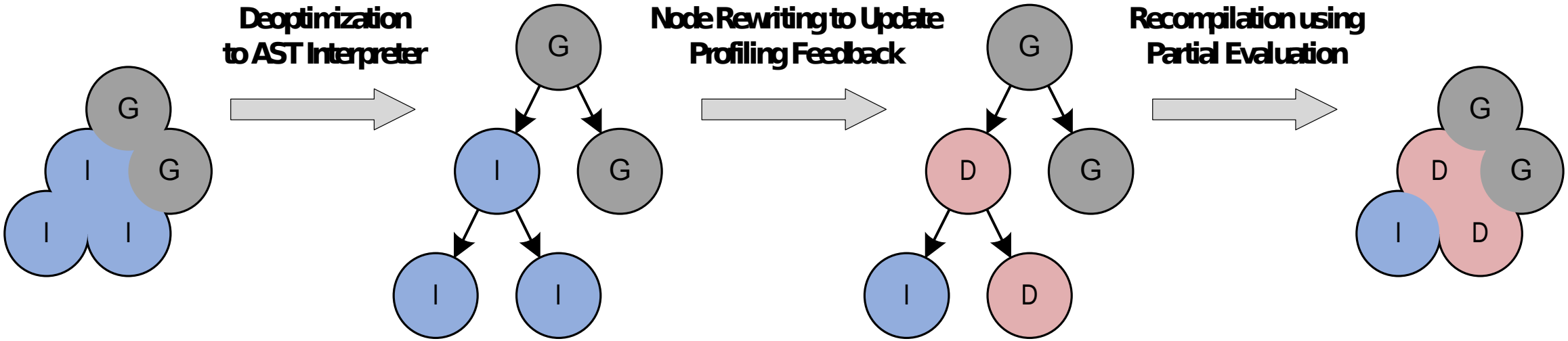
- Integer addition node
  - operands are not of integer type => rewrite the node
  - integer addition

→

- Do not PE
- Insert DEOPT

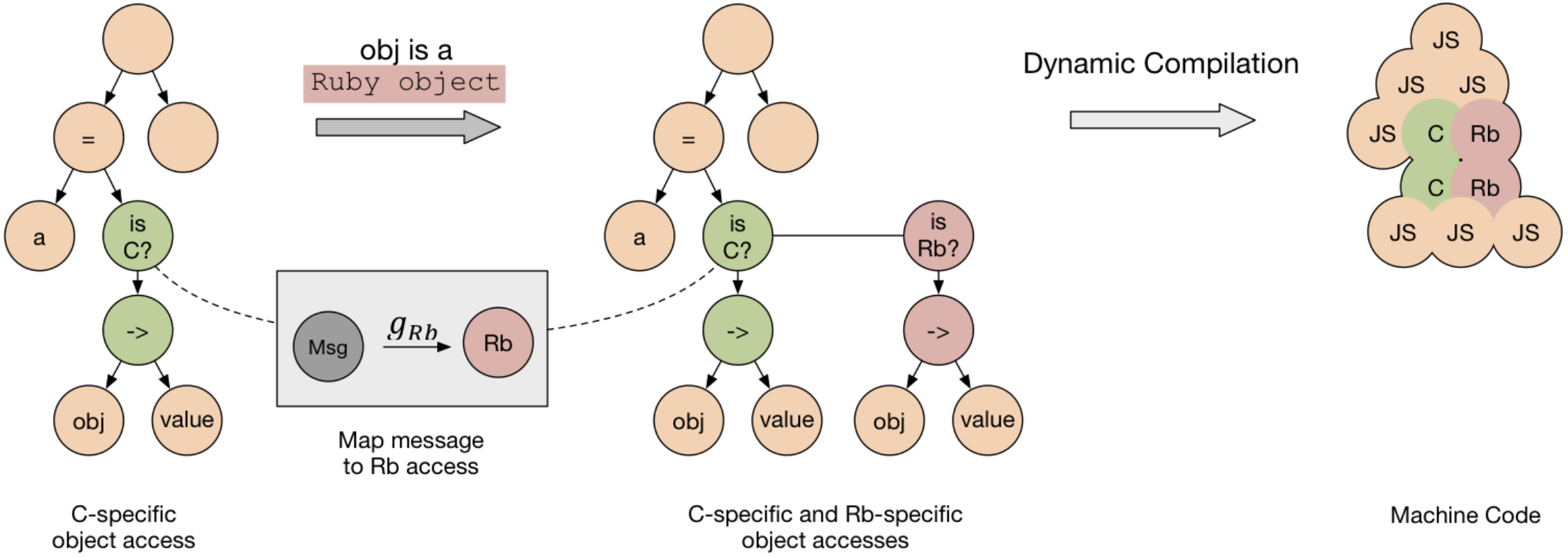


The argument is suddenly double...



High performance interoperability

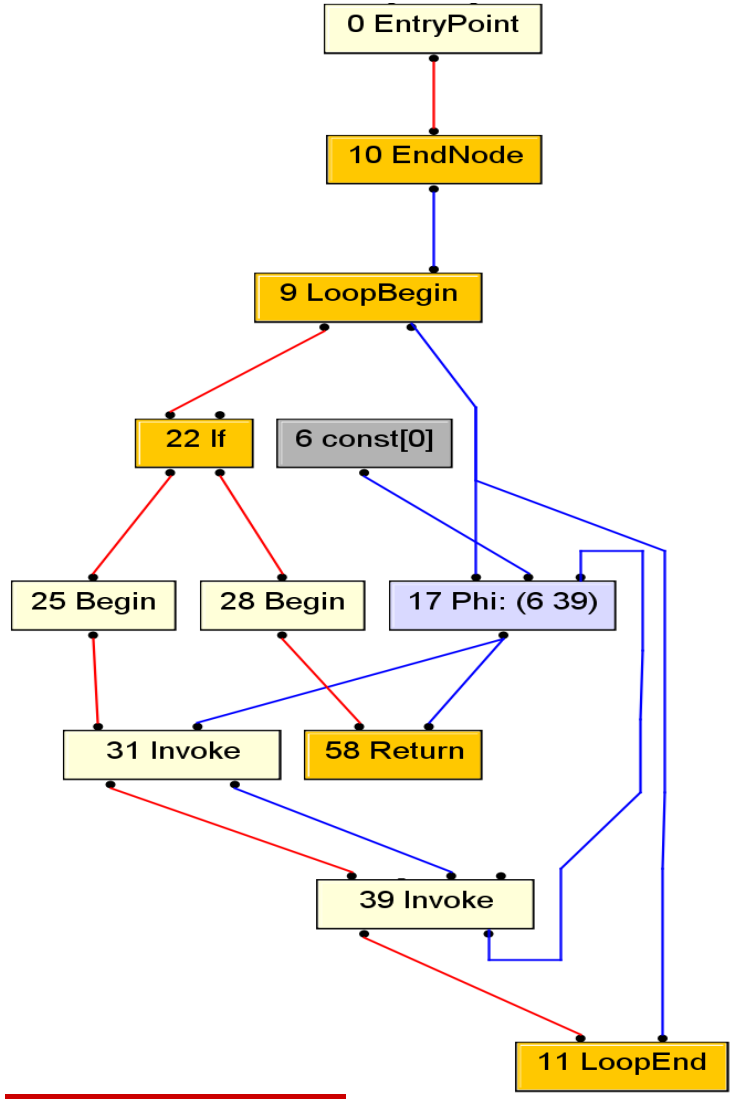
```
var a = obj.value;
```



Program Agenda

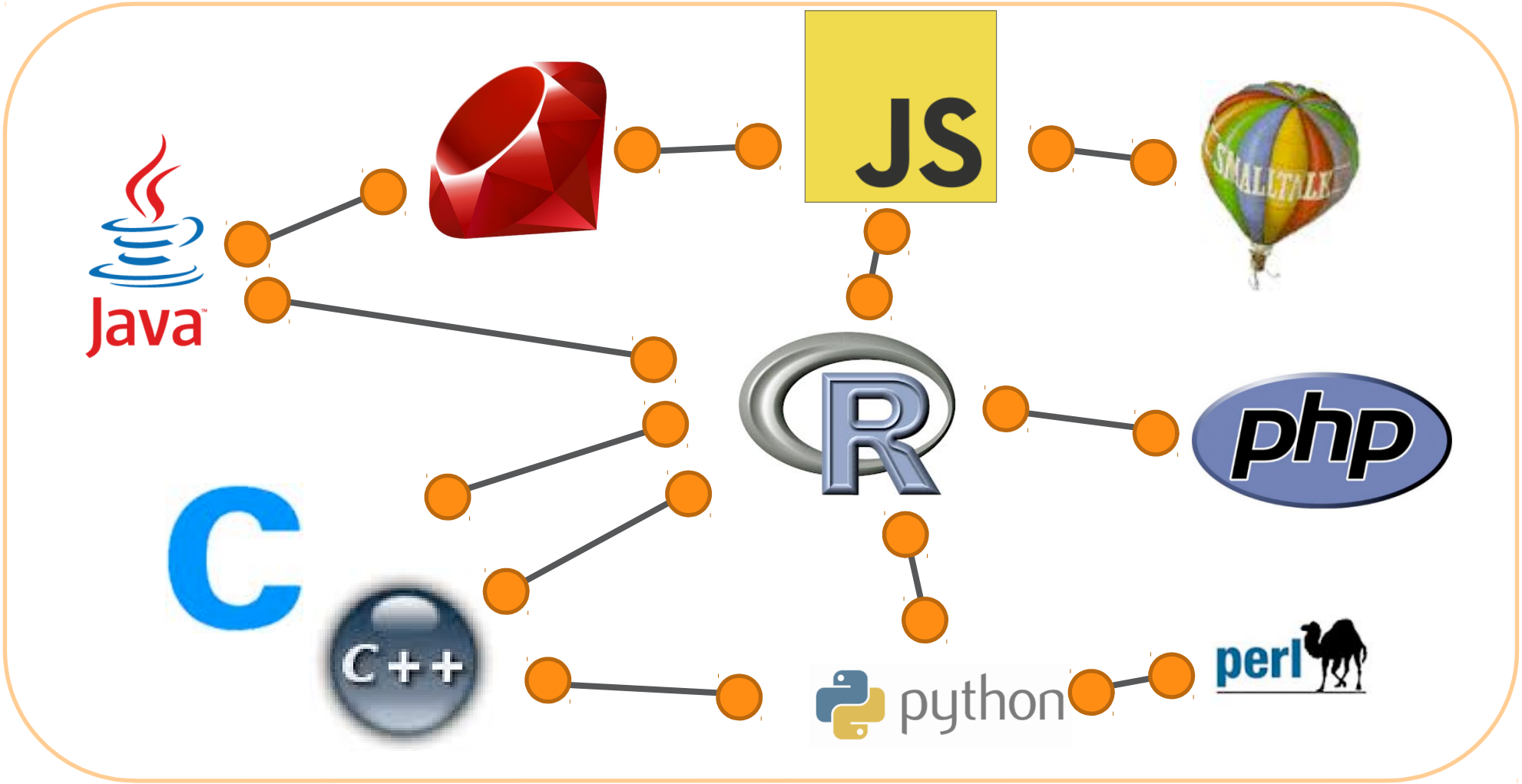
1. Motivation
2. How Truffle Works
3. Graal Compiler
4. Demos

Graal Compiler



- Modern alternative to HotSpot C2
  - Maintainable code base
  - Toolable, approachable
  - Ready for today's code
  - JEP 243: Java Compiler Interface (JVMCI)
- Partial evaluation
- Aggressive speculations
- Smooth de-optimizations

GraalVM: One VM to Rule them all!



Program Agenda

1. Motivation
2. How Truffle Works
3. Graal Compiler
4. Demos

Acknowledgements

Oracle
Danilo Ansaloni
Stefan Anzinger
Cosmin Basca
Daniele Bonetta
Matthias Brantner
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Martin Entlicher
Bastian Hossbach
Christian Humer
Mick Jordan
Vojin Jovanovic
Benoit Daloze
Brandon Fish
Petr Chalupa
Duncan MacGregor
Kevin Menard
Chris Seaton
Peter Kessler
David Leopoldseder
Kevin Menard
Jakub Podlešák

Oracle (continued)
Aleksandar Prokopec
Tom Rodriguez
Roland Schatz
Chris Seaton
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Lukas Stadler
Codrut Stancu
Jan Štola
Jaroslav Tulach
Michael Van De Vanter
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger
Miloslav Metelka
Tomáš Stupka
Florian Angerer
Tomáš Myšík
Petr Pišl
Svatopluk Dědic
Martin Entlicher

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
Adam Welc

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

Q/A

oracle.com/technetwork/oracle-labs/program-languages
github.com/graalvm

Integrated Cloud

Applications & Platform Services