ORACLE®

Truffle

A language implementation framework

Boris Spasojević
Senior Researcher

VM Research Group, Oracle Labs

Slides based on previous talks given by Christian Wimmer, Christian Humer and Matthias Grimmer.

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. 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

ORACLE

One Language to Rule Them All?

Let's ask Stack Overflow…

stackoverflow Questions Tags Tour Users

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Why can't there be an "ultimate" programming language?

closed as not constructive by Tim, Bo Persson, Devon_C_Miller, Mark, Graviton Jan 17 at 5:58

ORACLE

“Write Your Own Language”

Current situation

- 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

How it should be

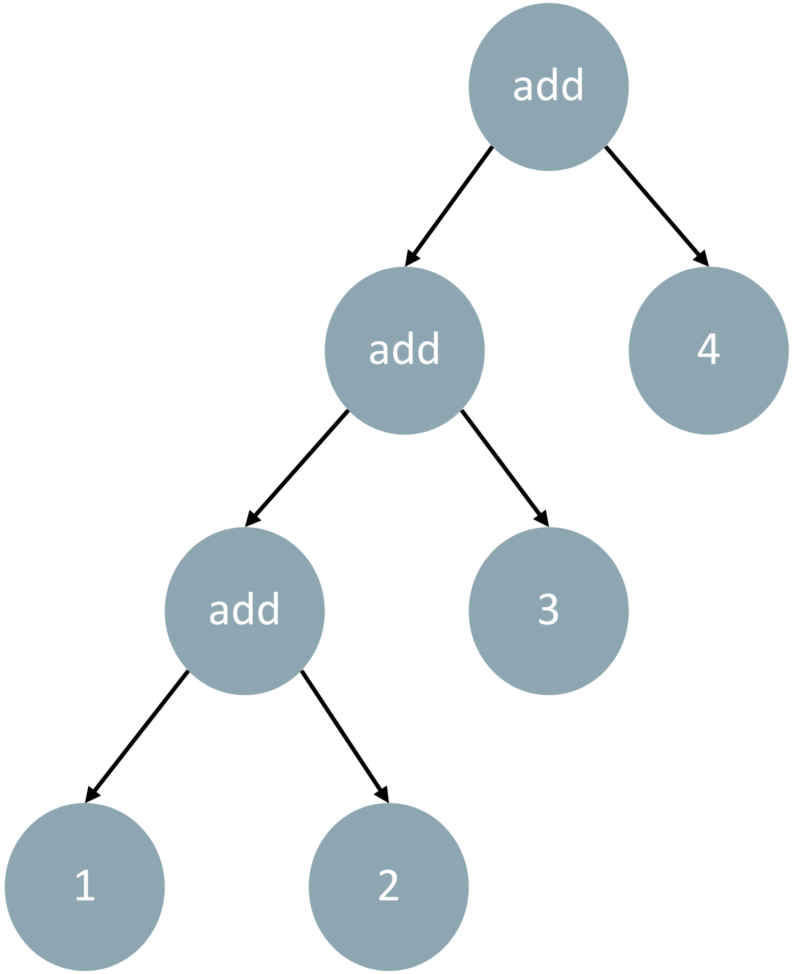
- Prototype a new language in Java
  - Parser and language work to build syntax tree (AST)
  - Execute using AST interpreter
- People start using it
  - **And it is already fast**
  - **And it integrates with other languages**
  - **And it has tool support, e.g., a debugger**

ORACLE

Background: AST Interpreter

1. Source code -> Abstract Syntax Tree
eg:

```
1+2+3+4
```

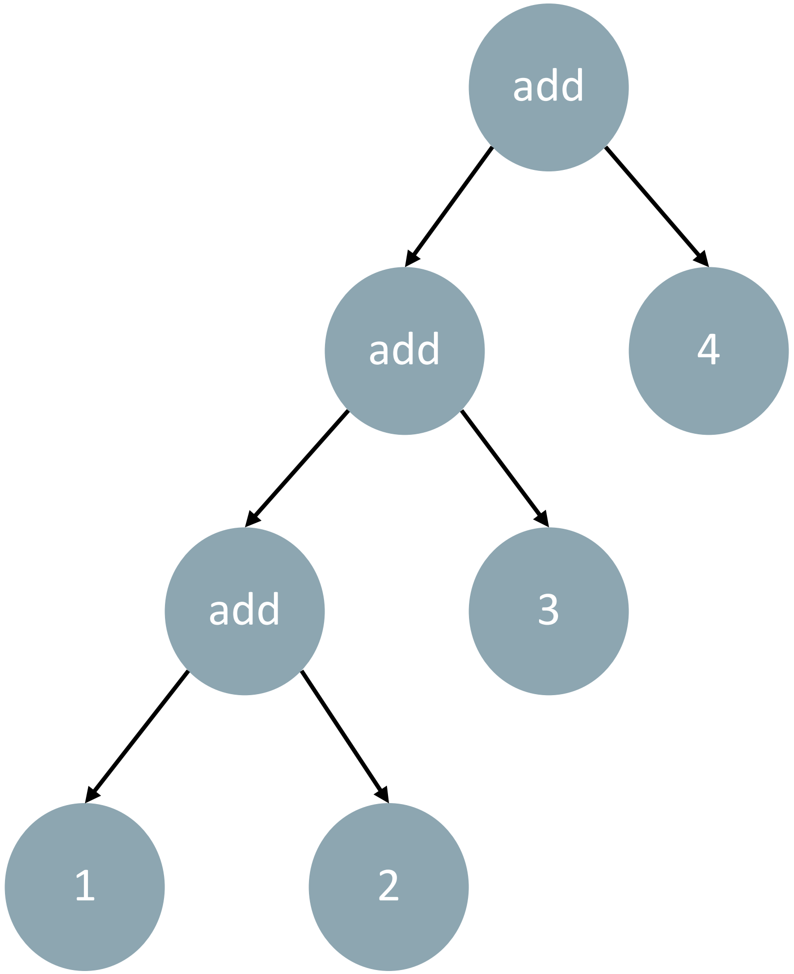


Background: AST Interpreter

2. Implement “execute” for each node

eg:

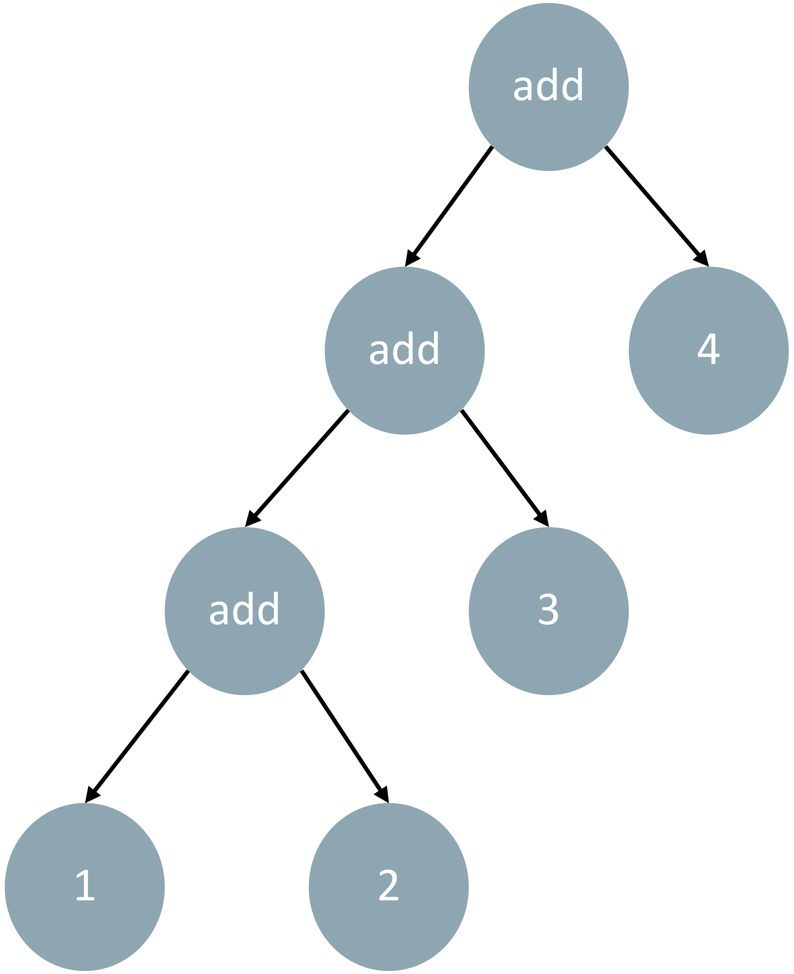
```
class AddNode extends Node {
  Node left;
  Node right;
  int execute {
    return left.execute()
        + right.execute();
  }
}
```



Background: AST Interpreter

3. Execute the AST root node
eg:

```
Int result = root.execute();
assert(result == 10);
```



Background: JIT Compiler

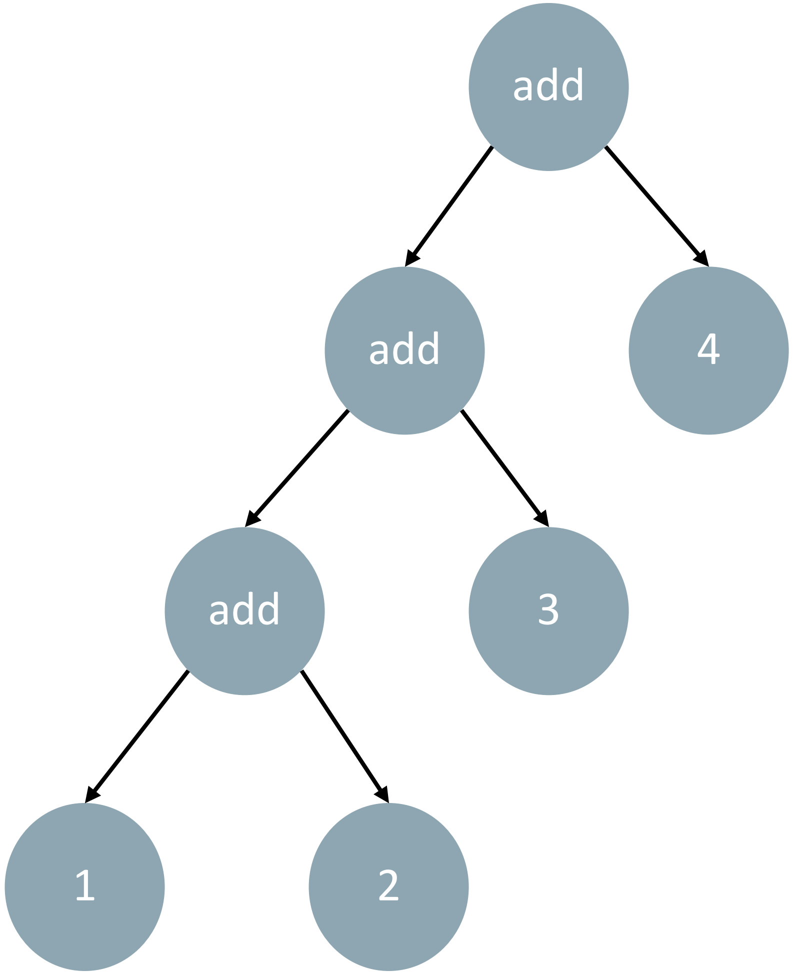
Compile once hot and profiled

eg:

```
while(root.execute() == 10);
```

eventually :

```
mov eax, 10
```



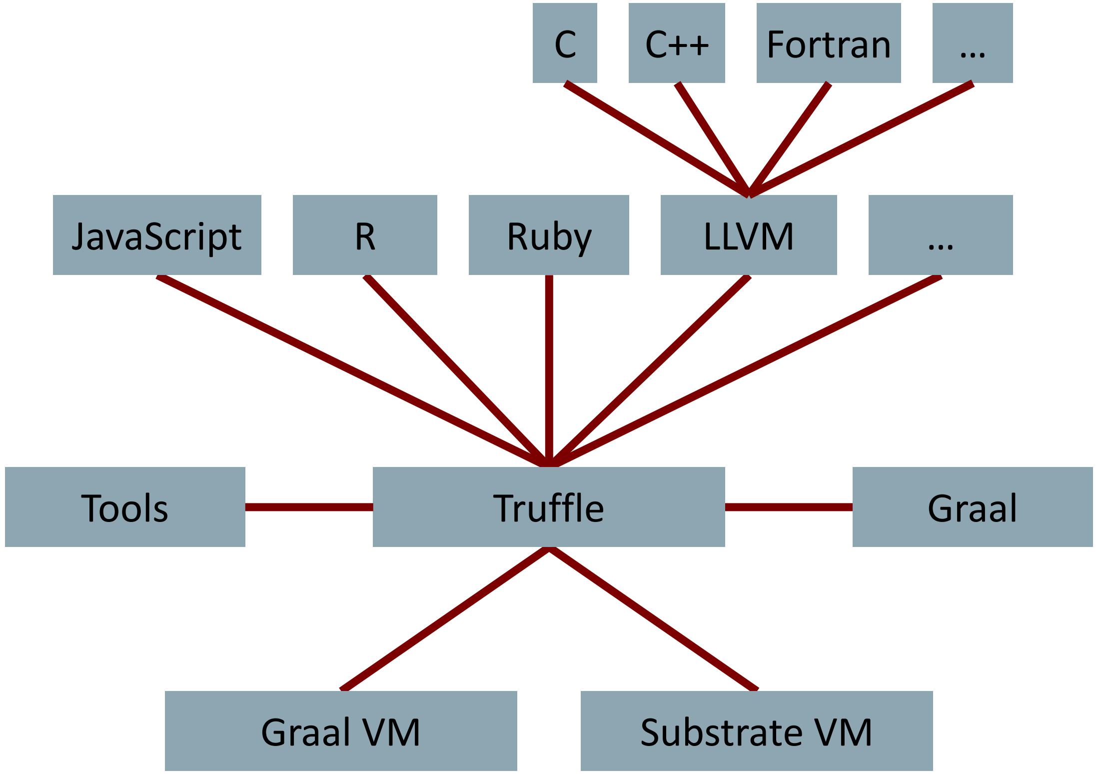
Lets talk about JavaScript...

```
function negate(a) {
   return -a
}
```

```
> negate(42)
-42

> negate(-"42")
"-42"

> negate({})
NaN

> negate([])
0
```

ORACLE

Overall System Structure



Interpreter for every language

Common API separates language implementation, optimization system, and tools (debugger)

Language agnostic dynamic compiler

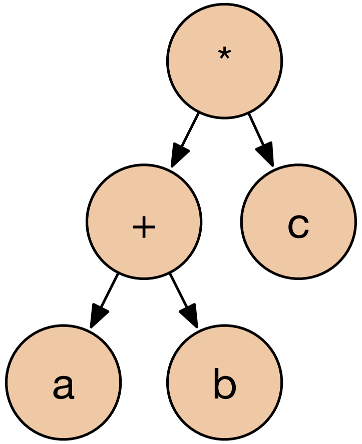
Integrate with Java applications

Low-footprint VM, also suitable for embedding

ORACLE

Truffle

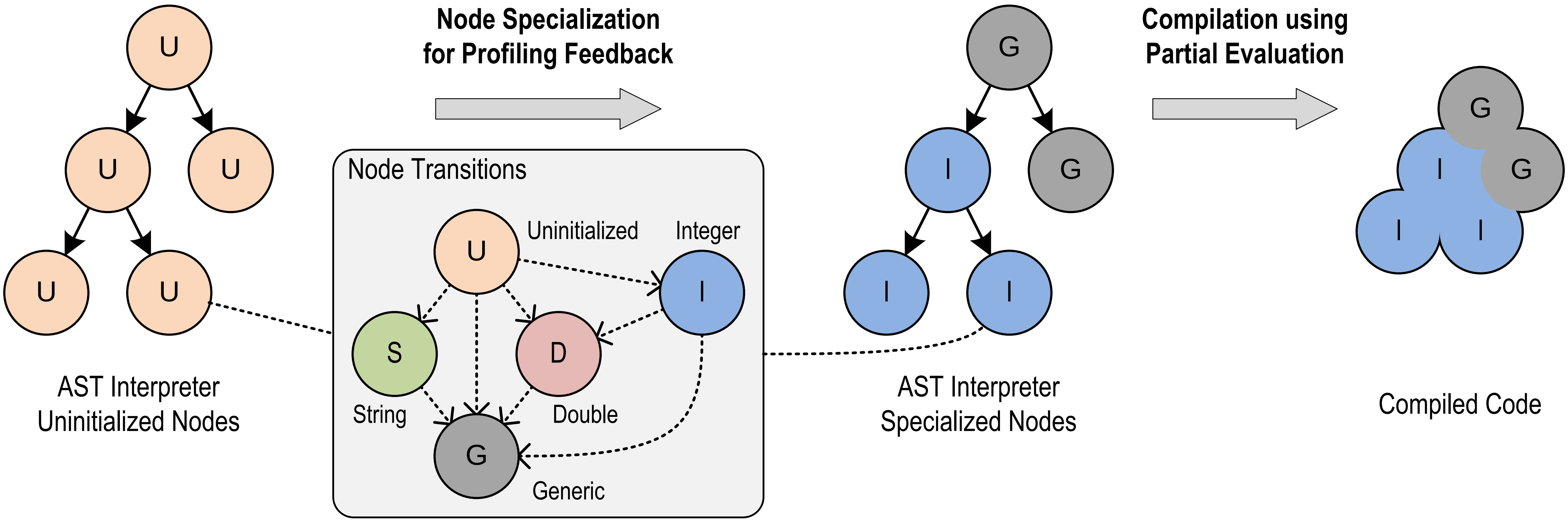
```
(a + b) * c
```



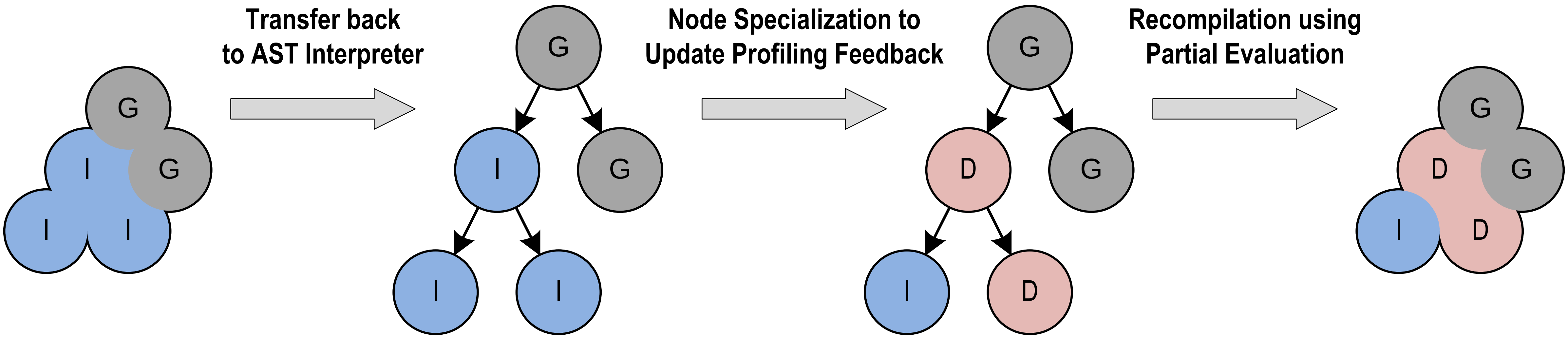

```
Object execute(VirtualFrame frame) {
   Object a = left.execute(frame);
   Object b = right.execute(frame);
   return add(a, b);
}
Object add(Object a, Object b) {
   if(a instanceof Integer && b instanceof Integer) {
      return (int)a + (int)b;
   } else if (a instanceof String && b instanceof String) {
      return (String)a + (String)b;
   } else {
      return genericAdd(a, b);
   }
}
```

- Abstract syntax tree IS the interpreter
- Every node has an execute method
- Running Java program that interprets JavaScript (or any other language)

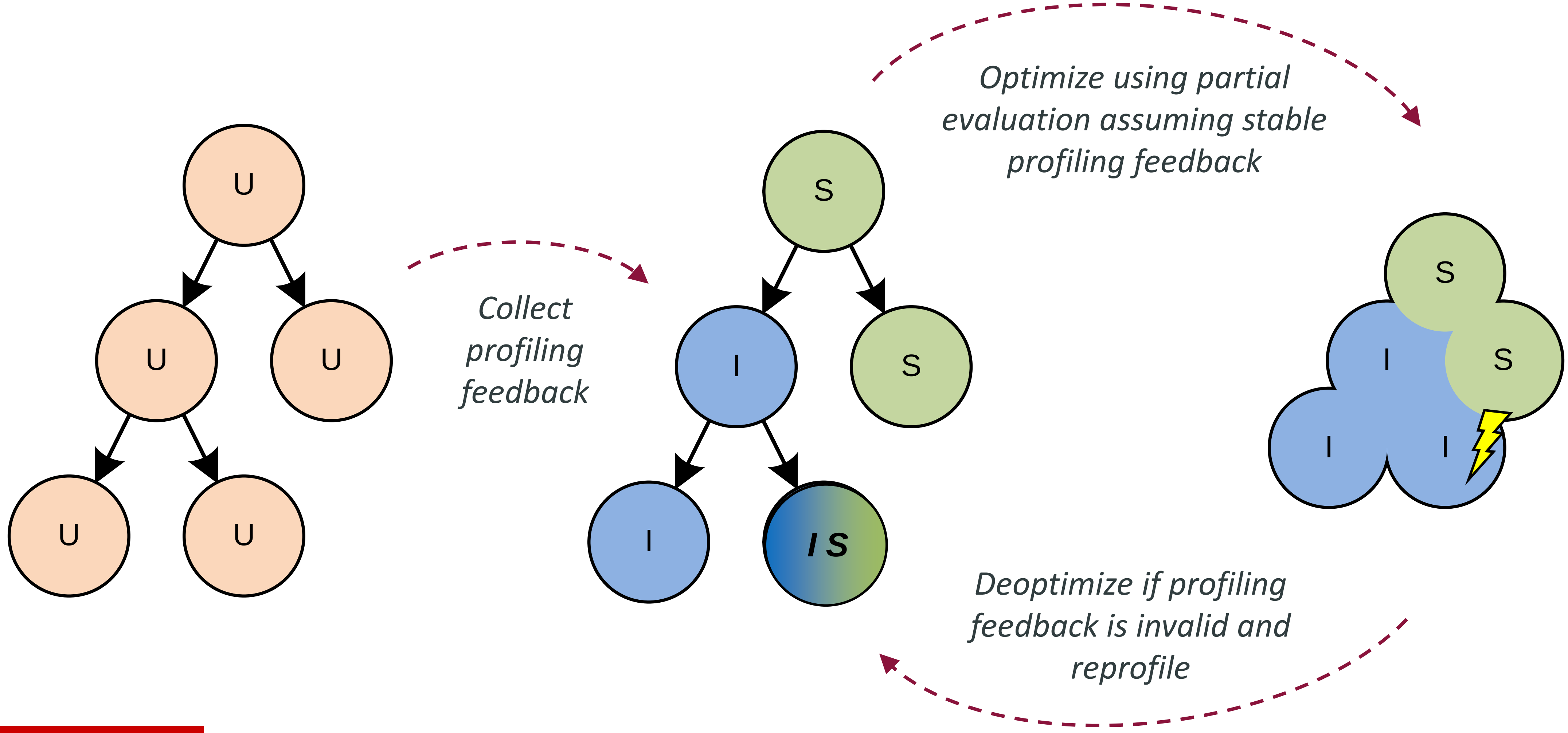
Speculate and Optimize …



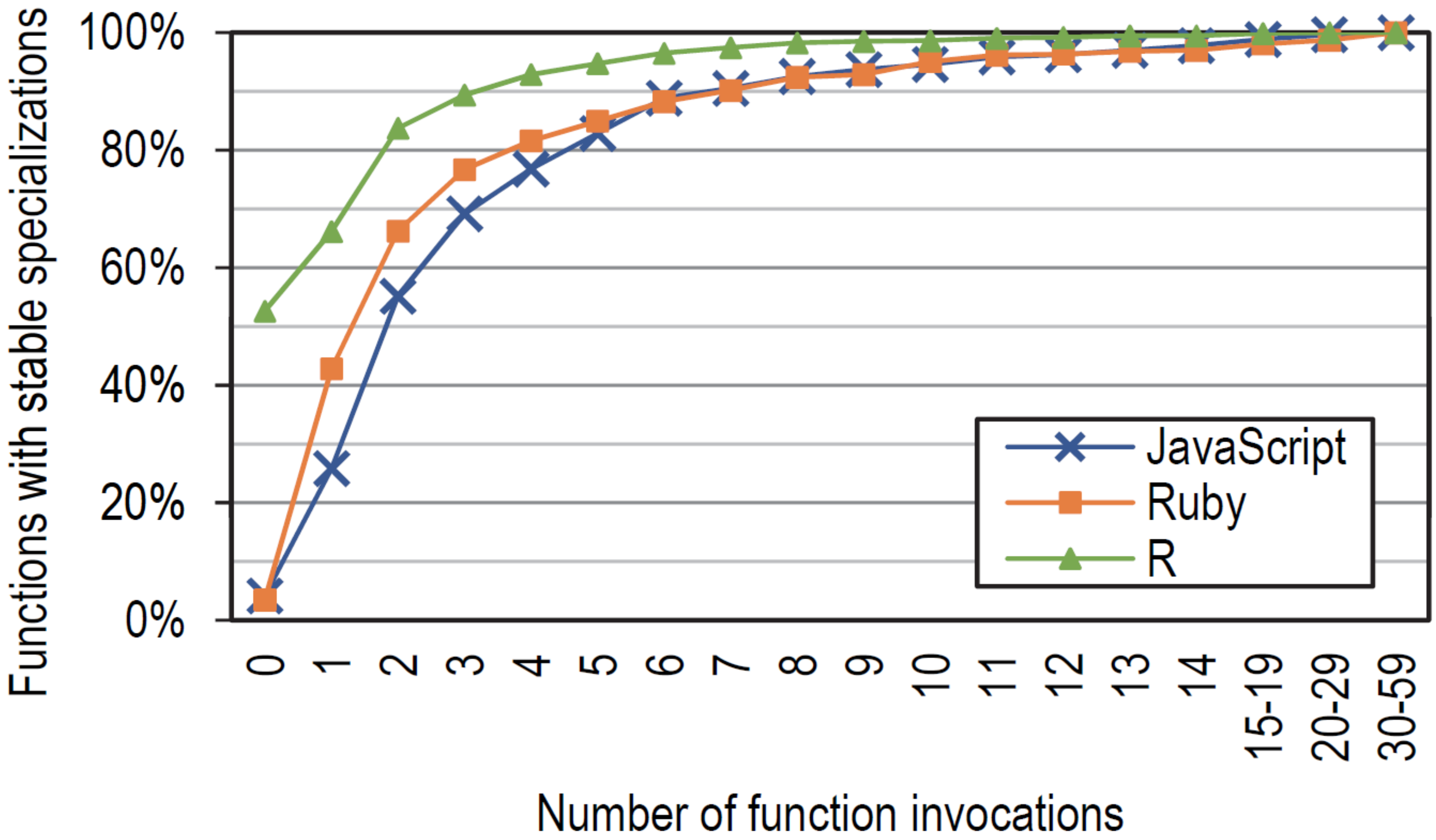
… and Transfer to Interpreter and Reoptimize!



The Truffle Idea



Stability



Functions with stable specializations
100%
80%
60%
40%
20%
0%
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15-19
20-29
30-59
Number of function invocations
JavaScript
Ruby
R

More Details on Truffle Approach

https://wiki.openjdk.java.net/display/Graal/Publications+and+Presentations
https://github.com/graalvm/simplelanguage
Oracle Labs VM Research YouTube channel

One VM to Rule Them All

Thomas Würthinger[*] Christian Wimmer[*] Andreas Wöß[†] Lukas Stadler[†]
Gilles Duboscq[†] Christian Humer[†] Gregor Richards[§] Doug Simon[*] Mario Wolczko[*]

[*]Oracle Labs [†]Institute for System Software, Johannes Kepler University Linz, Austria [§]S³ Lab, Purdue University
{thomas.wuerthinger, christian.wimmer, doug.simon, mario.wolczko}@oracle.com
{woess, stadler, duboscq, christian.humer}@ssw.jku.at gr@purdue.edu

Abstract

Building high-performance virtual machines is a complex and expensive undertaking; many popular languages still have low-performance implementations. We describe a new approach to virtual machine (VM) construction that amortizes much of the effort in initial construction by allowing new languages to be implemented with modest additional effort. The approach relies on abstract syntax tree (AST) interpretation where a node can rewrite itself to a more specialized or more general node, together with an optimizing com-

as Microsoft's Common Language Runtime, the VM of the .NET framework [43]. These implementations can be characterized in the following way:

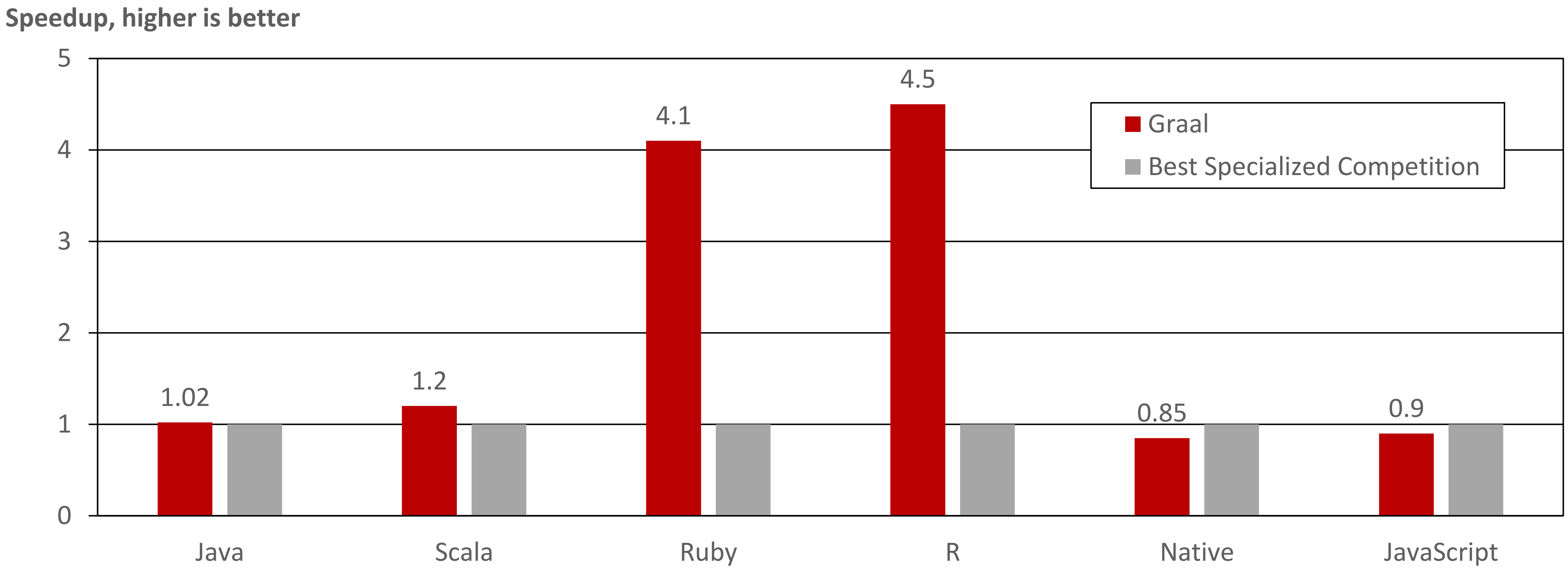
- Their performance on typical applications is within a small integer multiple (1-3x) of the best statically compiled code for most equivalent programs written in an unsafe language such as C.
- They are usually written in an unsafe, systems programming language (C or C++).

Performance Disclaimers

- All Truffle numbers reflect a development snapshot
  - Subject to change at any time (hopefully improve)
  - You have to know a benchmark to understand why it is slow or fast
- We are not claiming to have complete language implementations
  - JavaScript: passes 100% of ECMAscript standard tests
    - Working on full compatibility with V8 for Node.JS
  - Ruby: passing 100% of RubySpec language tests
    - Passing around 90% of the core library tests
  - R: prototype, but already complete enough and fast for a few selected workloads
    - Benchmarks that are not shown – may not run at all, or – may not run fast

ORACLE

Performance: GraalVM Summary

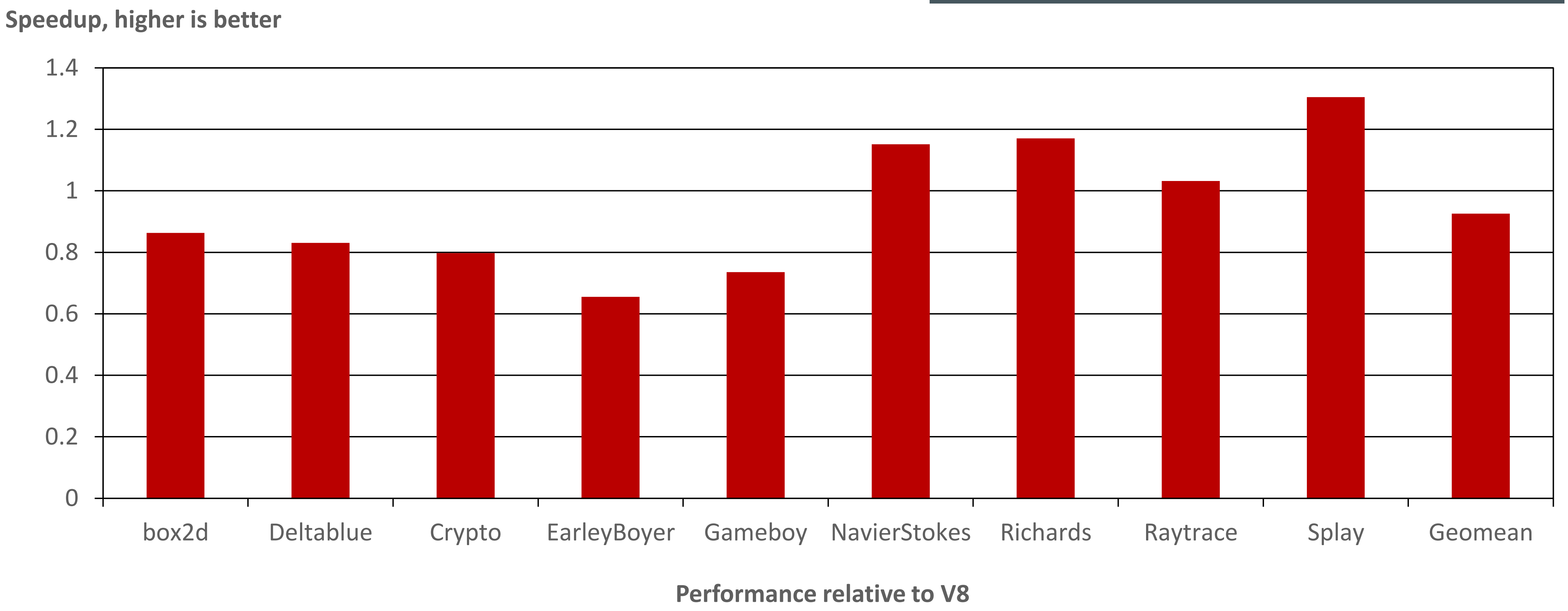


Performance relative to:
HotSpot/Server, HotSpot/Server running JRuby, GNU R, LLVM AOT compiled, V8

ORACLE

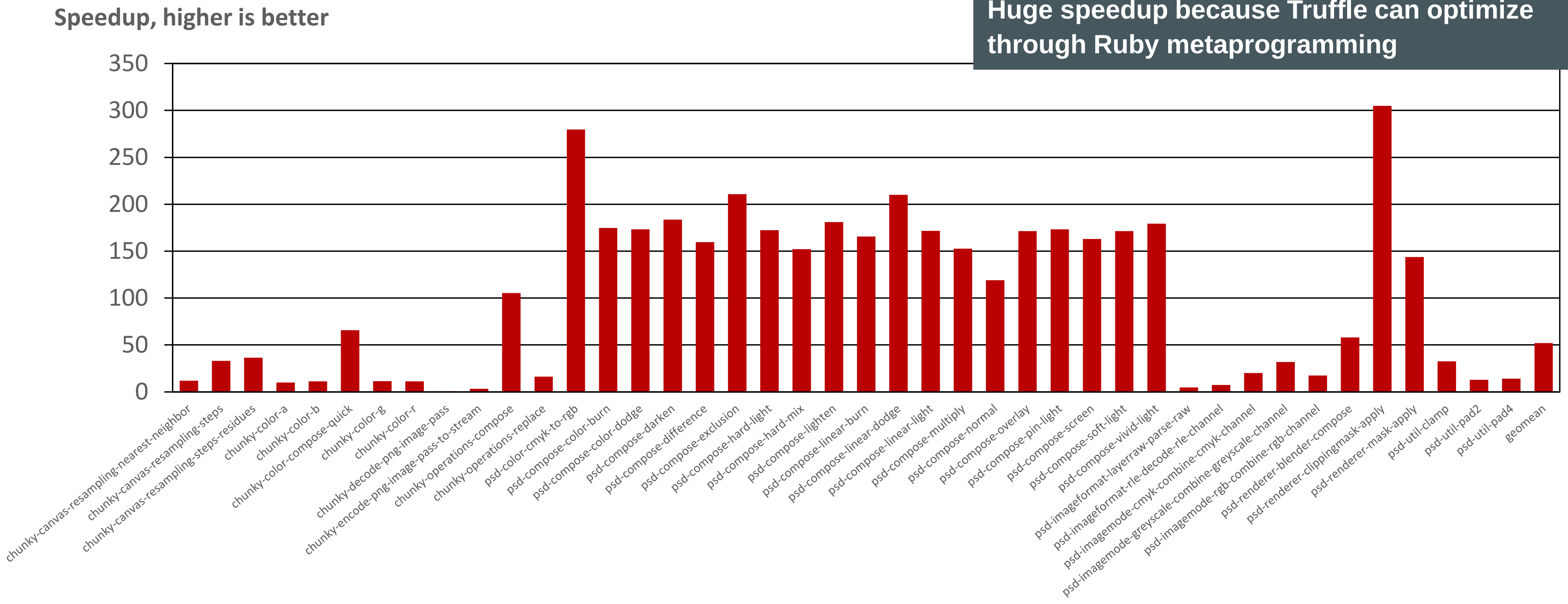
Performance: JavaScript

JavaScript performance: similar to V8



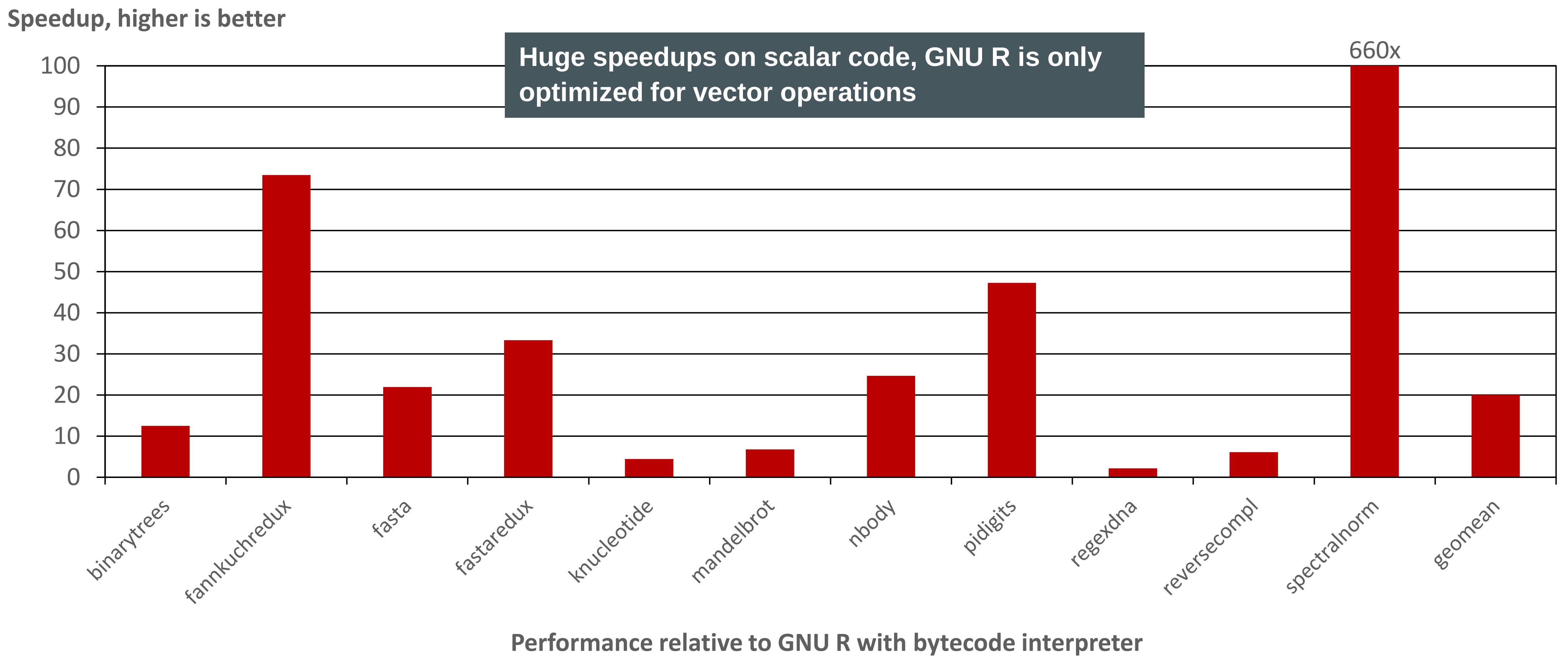
Performance: Ruby Compute-Intensive Kernels

Huge speedup because Truffle can optimize through Ruby metaprogramming



Performance relative to JRuby running with Java HotSpot server compiler

Performance: R with Scalar Code



Truffle Core Features

- Partial Evaluation (with Explicit Boundaries)
- Speculation with Internal Invalidation (guards)
- Speculation with External Invalidation (assumptions)

ORACLE

Introduction to Partial Evaluation

```
abstract class Node {
    abstract int execute(int[] args);
}

class AddNode extends Node {
    final Node left, right;

    AddNode(Node left, Node right) {
        this.left = right; this.right = right;
    }

    int execute(int args[]) {
        return left.execute(args) + right.execute(args);
    }
}
```

```
class Arg extends Node {
    final int index;
    Arg(int i) {this.index = i;}

    int execute(int[] args) {
        return args[index];
    }
}
```

```
int interpret(Node node, int[] args) {
    return node.execute(args);
}
```

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```

Introduction to Partial Evaluation

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```

```
int interpret(Node node, int[] args) {
    return node.execute(args);
}
```

```
partiallyEvaluate(interpret, sample)
```

```
int interpretSample(int[] args) {
    return sample.execute(args);
}
```

Introduction to Partial Evaluation

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```

```
int interpretSample(int[] args) {
    return sample.execute(args);
}
```

```
int interpretSample(int[] args) {
    return sample.left.execute(args)
         + sample.right.execute(args);
}
```

```
int interpretSample(int[] args) {
    return sample.left.left.execute(args)
         + sample.left.right.execute(args)
         + args[sample.right.index];
}
```

```
int interpretSample(int[] args) {
    return args[sample.left.left.index]
         + args[sample.left.right.index]
         + args[sample.right.index];
}
```

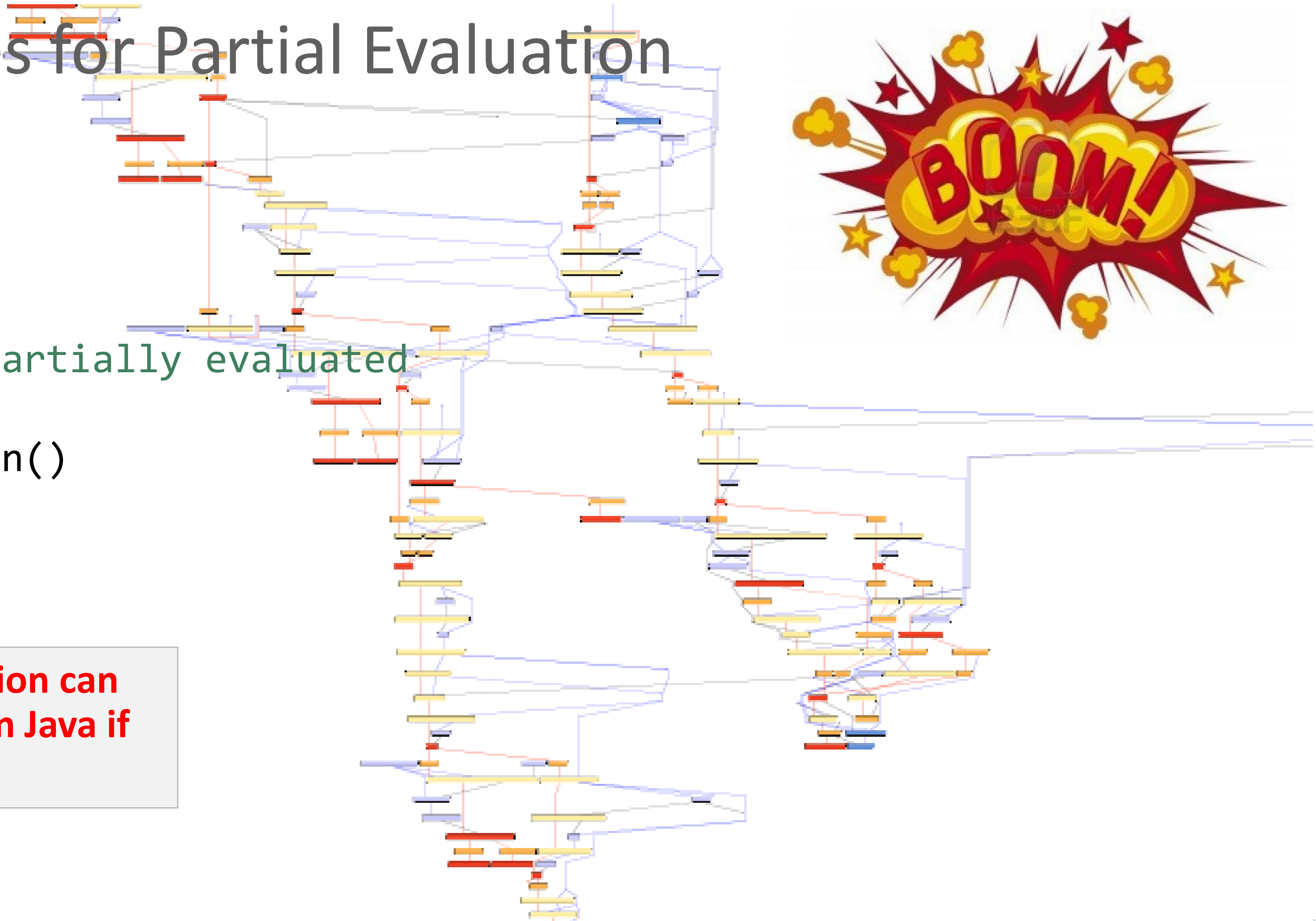
```
int interpretSample(int[] args) {
    return args[0]
         + args[1]
         + args[2];
}
```

Explicit Boundaries for Partial Evaluation

```
Object parseJSON(Object value) {
    String s = objectToString(value);
    return parseJSONString(s);
}

@TruffleBoundary
Object parseJSONString(String value) {
    // complex JSON parsing code
}
```

ORACLE

Explicit Boundaries for Partial Evaluation




```
// no boundary, but partially evaluated
void printLn() {
    System.out.println()
}
```

=> Partially evaluated version can be significantly slower than Java if not handled with care!

ORACLE

Initiate Partial Evaluation

```
class Function extends RootNode {
    @Child Node child;

    Object execute(VirtualFrame frame) {
        return child.execute(frame)
    }
}

public static void main(String[] args) {
    CallTarget target = Truffle.getRuntime().createCallTarget(new Function());

    for (int i = 0; i < 10000; i++) {
        // after a few calls partially evaluates on a background thread
        // installs partially evaluated code when ready
        target.call();
    }
}
```

ORACLE

Speculation with Internal Invalidation

```java
class NegateNode extends Node {

    @CompilationFinal boolean objectSeen = false;

    Object execute(Object v) {
        if (v instanceof Double) {
            return -((double) v);
        } else {
            if (!objectSeen) {
                transferToInterpreter();
                objectSeen = true;
            }
            // slow-case handling of all
            // other types
            return objectNegate(v);
        }
    }
}
```

Compiler sees: objectSeen = false

```java
if (v instanceof Double) {
    return -((double) v);
} else {
    deoptimize;
}
```

Compiler sees: objectSeen = true

```java
if (v instanceof Double) {
    return -((double) v);
} else {
    return objectNegate(v);
}
```

ORACLE

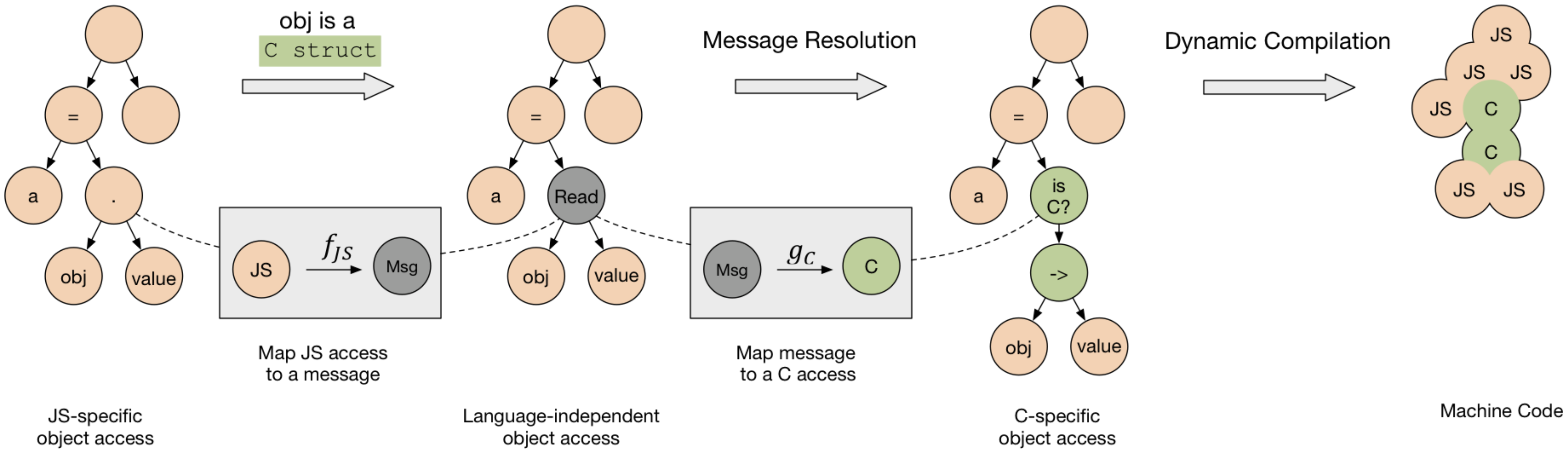
Speculation with External Invalidation

```
@CompilationFinal static Assumption addNotDefined = new Assumption();

class AddNode extends Node {

    int execute(int left, int right) {
        if (addNotDefined.isValid()) {
            return left + right;
        }
        ... // complicated code to call user-defined add
    }
}

static void defineFunction(String name, Function f) {
    if (name.equals("+")) {
        addNotDefined.invalidate();
        ... // register user-defined add
    }
}
```

Polyglot Demo.

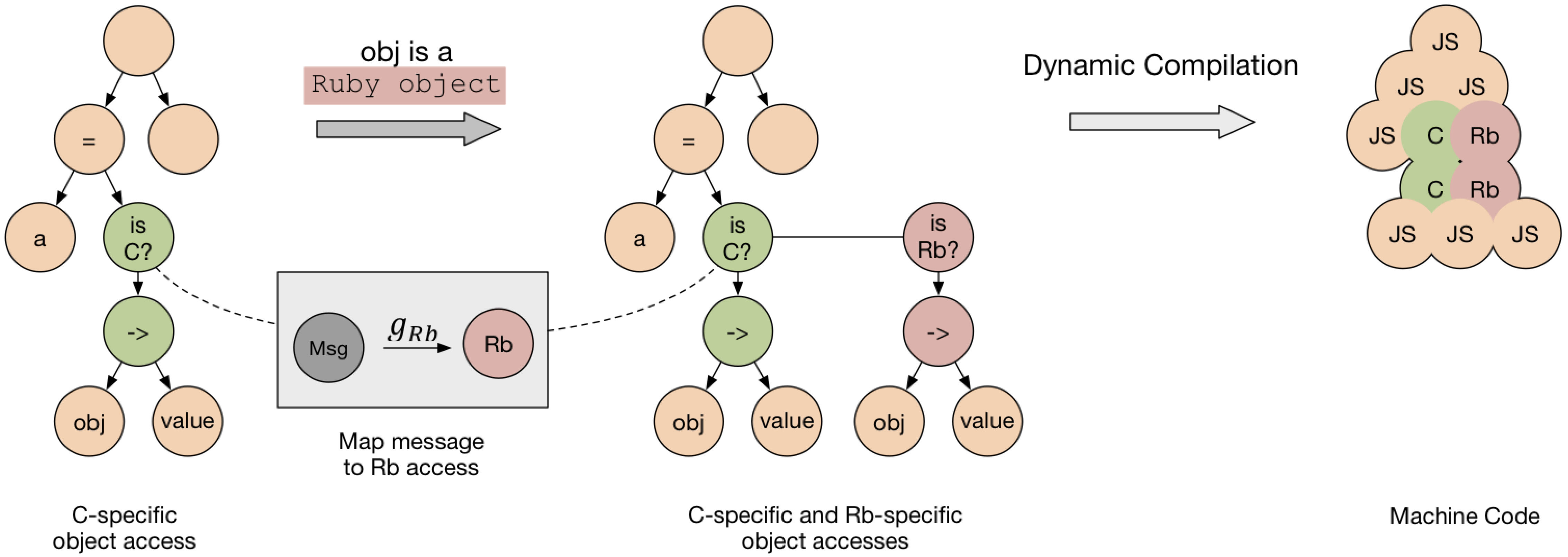
High-Performance Language Interoperability (1)

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



High-Performance Language Interoperability (2)

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



More Details on Language Integration

http://dx.doi.org/10.1145/2816707.2816714

High-Performance Cross-Language Interoperability in a Multi-language Runtime

Matthias Grimmer
Johannes Kepler University Linz, Austria
matthias.grimmer@jku.at

Chris Seaton
Oracle Labs, United Kingdom
chris.seaton@oracle.com

Roland Schatz
Oracle Labs, Austria
roland.schatz@oracle.com

Thomas Würthinger
Oracle Labs, Switzerland
thomas.wuerthinger@oracle.com

Hanspeter Mössenböck
Johannes Kepler University Linz, Austria
hanspeter.moessenboeck@jku.at

Abstract

Programmers combine different programming languages because it allows them to use the most suitable language for a given problem, to gradually migrate existing projects from one language to another, or to reuse existing source code.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Run-time environments, Code generation, Interpreters, Compilers, Optimization

Keywords cross-language; language interoperability; virtual machine; optimization; language implementation

Tools

Tools: We Don’t Have It All

(Especially for Debuggers)

- Difficult to build
  - Platform specific
  - Violate system abstractions
  - Limited access to execution state

- Productivity tradeoffs for programmers
  - Performance – disabled optimizations
  - Functionality – inhibited language features
  - Complexity – language implementation requirements
  - Inconvenience – nonstandard context (debug flags)

ORACLE

Tools: We Can Have It All

- Build tool support into the Truffle API
  - High-performance implementation
  - Many languages: any Truffle language can be tool-ready with minimal effort
  - Reduced implementation effort

- Generalized *instrumentation* support
  1. Access to execution state & events
  2. Minimal runtime overhead
  3. Reduced implementation effort (for languages *and* tools)

ORACLE

Implementation Effort: Language Implementors

- Treat AST syntax nodes specially
  - Precise source attribution
  - Enable probing
  - Ensure stability

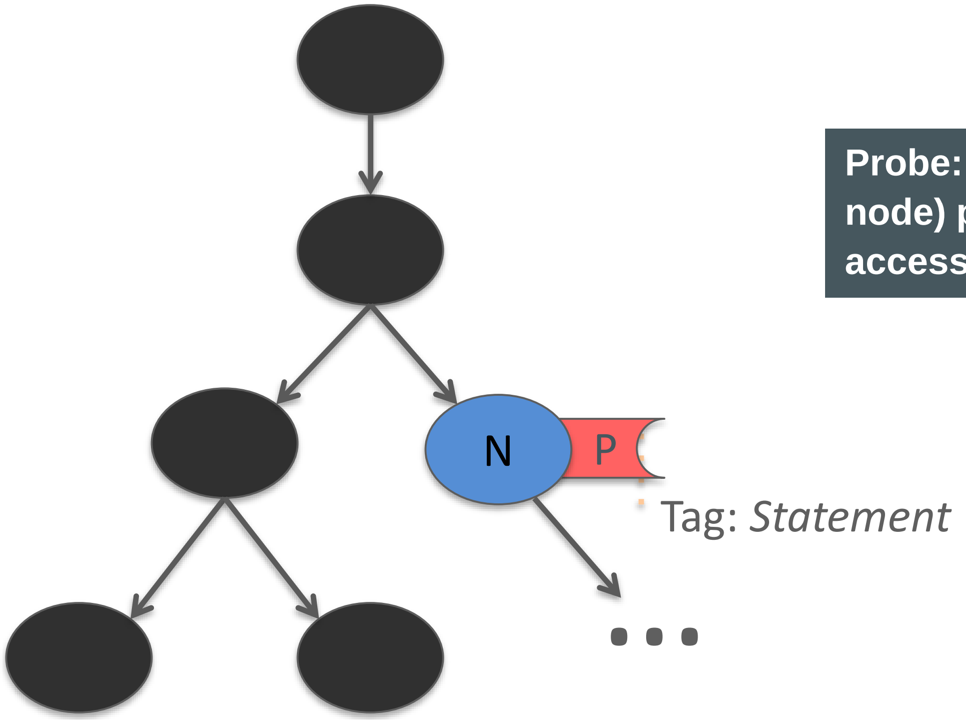
- Add default tags, e.g., Statement, Call, ...
  - Sufficient for many tools
  - Can be extended, adjusted, or replaced dynamically by other tools

- Implement debugging support methods, e.g.
  - Eval a string in context of any stack frame
  - Display language-specific values, method names, …

- More to be added to support new tools & services

ORACLE

“Mark Up” Important AST Nodes for Instrumentation



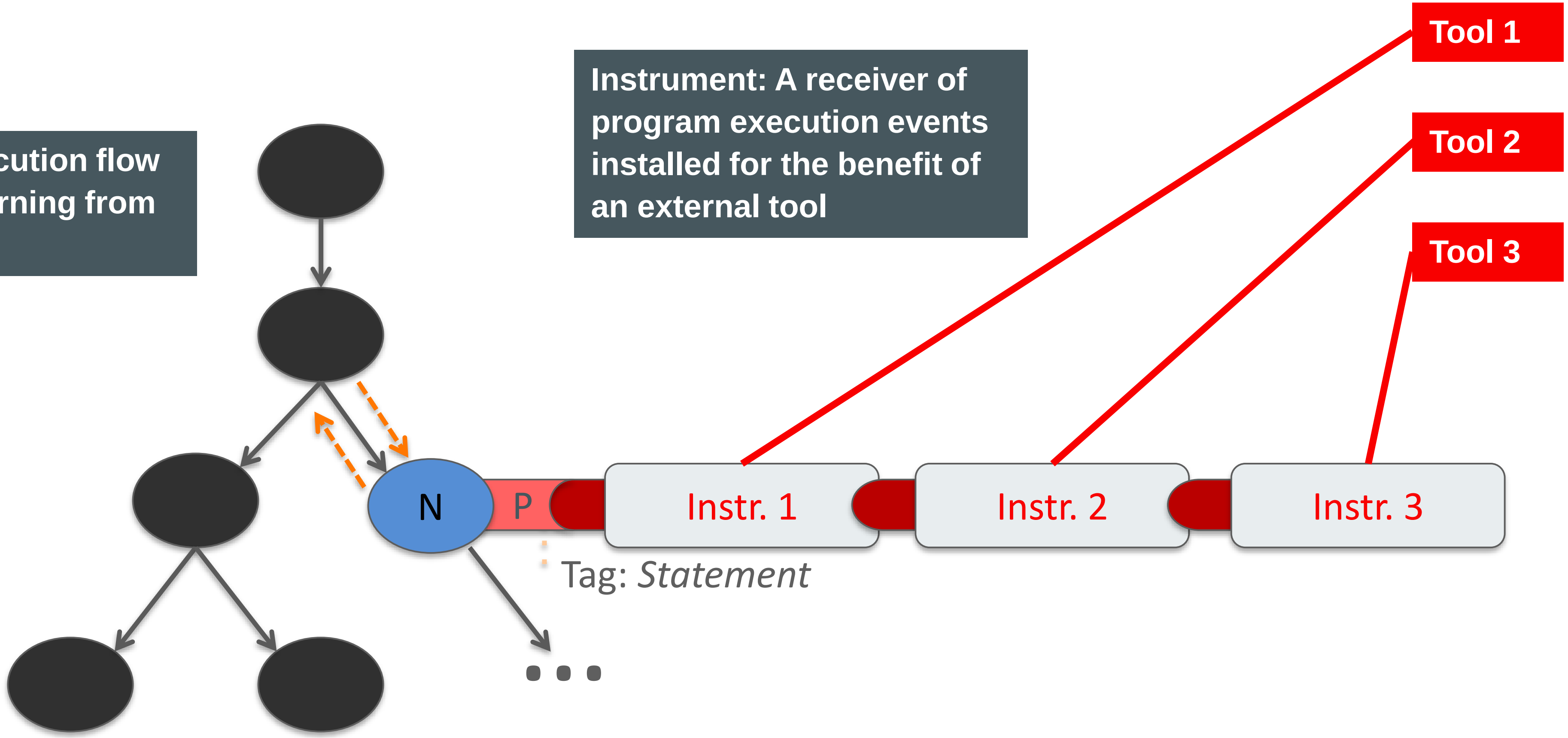
Probe: A program location (AST node) prepared to give tools access to execution state.

Tag: An annotation for configuring tool behavior at a Probe. Multiple tags, possibly tool-specific, are allowed.

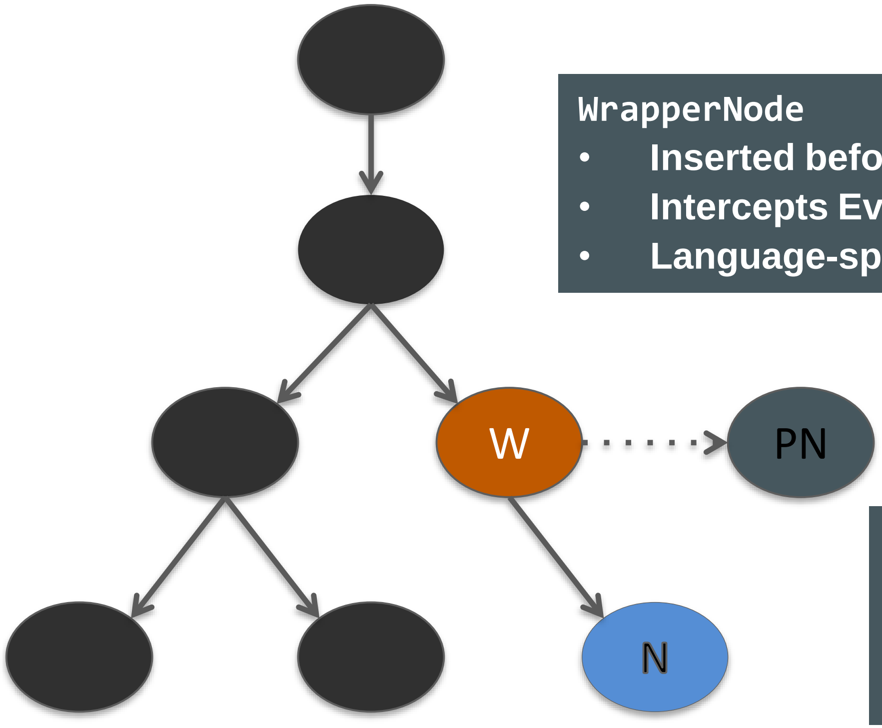
Access to Execution Events

Event: AST execution flow entering or returning from a node.

Instrument: A receiver of program execution events installed for the benefit of an external tool



Implementation: Nodes



`WrapperNode`
- Inserted before any execution
- Intercepts Events
- Language-specific Type

`ProbeNode`
- Manages “instrument chain” dynamically
- Propagates Events
- Instrumentation Type

ORACLE

43

More Details on Instrumentation and Debugging

http://dx.doi.org/10.1145/2843915.2843917

Building Debuggers and Other Tools: We Can "Have it All"

Position Paper ICOOOLPS '15

Michael L. Van De Vanter

Oracle Labs

michael.van.de.vanter@oracle.com

Abstract

Software development tools that "instrument" running programs, notably debuggers, are presumed to demand difficult tradeoffs among *performance*, *functionality*, *implementation complexity*, and *user convenience*. A fundamental change in our thinking about such tools makes that presumption obsolete.

By building instrumentation directly into the core of a high-performance language implementation framework, tool-support can be *always on*, with confidence that optimization will apply uniformly to instrumentation and result in near zero overhead. Tools can be always available (and fast), not only for end user programmers, but also for language implementors throughout development.

2. Roadblocks

Why is it so difficult to have tools that are as good and timely as our programming languages? Why can't we "have it all"?

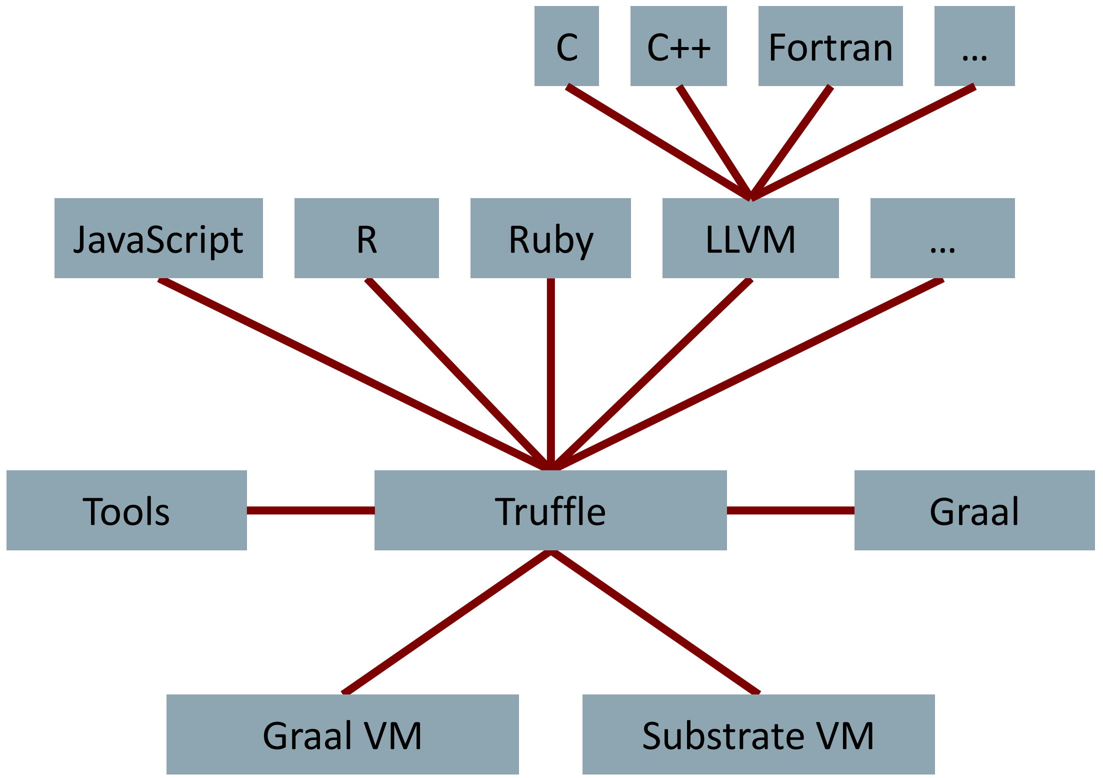
2.1 Tribes

One perspective is historical and cultural. Concerns about program execution speed (utilization of *expensive machines*) came long before concerns about software development rate and correctness (utilization of *expensive people*).

Our legacy is that people who write compilers and people who build developer tools essentially belong to different *tribes*, each with its own technologies and priorities[1]. More significantly, each

ORACLE®

Overall System Structure



Interpreter for every language

Common API separates language implementation, optimization system, and tools (debugger)

Language agnostic dynamic compiler

Integrate with Java applications

Low-footprint VM, also suitable for embedding

ORACLE

Internships at Oracle

Safe Harbor Statement

The preceding is intended to outline our general product direction. 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. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Integrated Cloud

Applications & Platform Services

ORACLE

ORACLE®