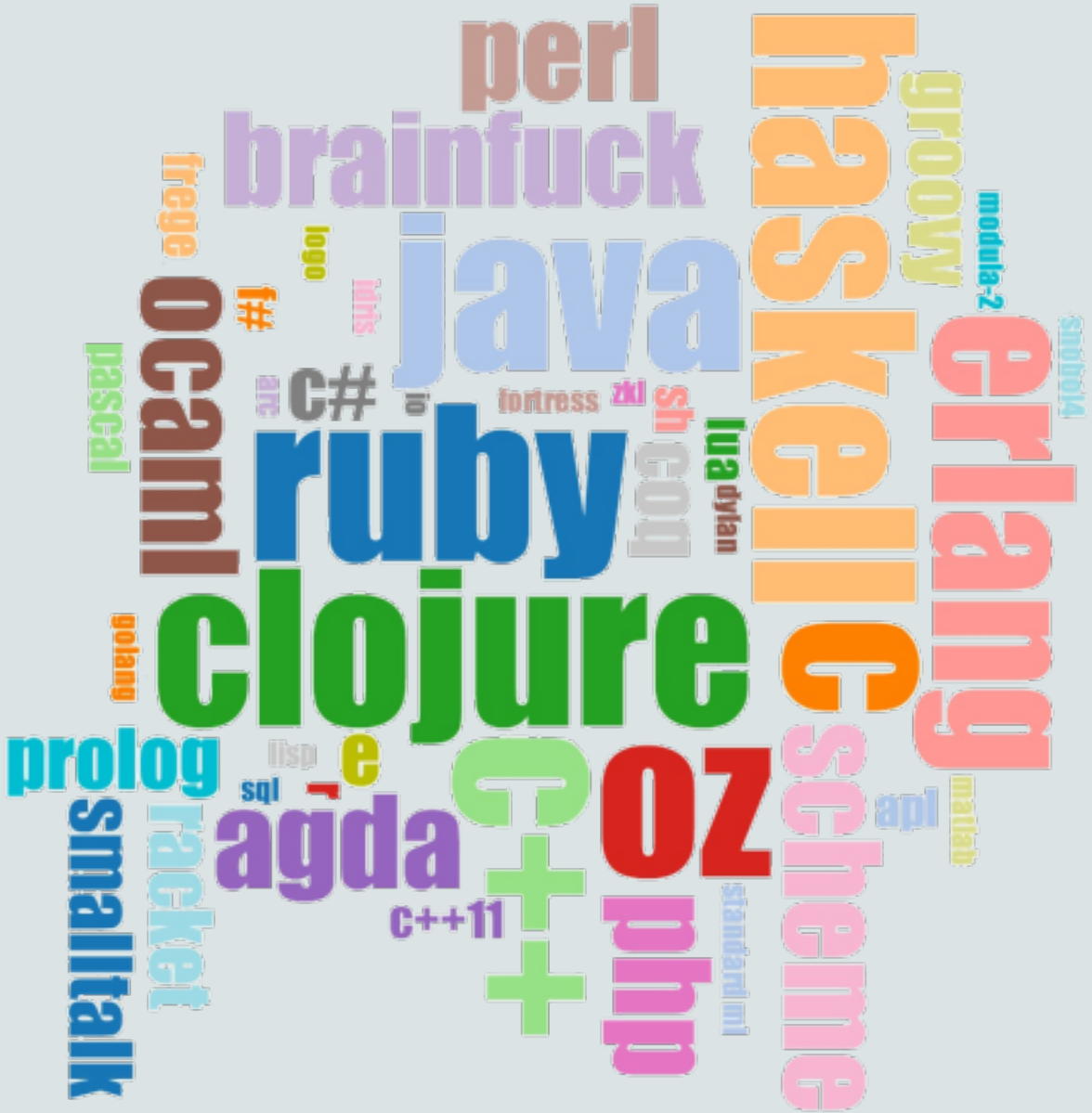
Dynamic Compilation with Truffle

Thomas Wuerthinger
@thomaswue
Oracle Labs

Christian Humer
@grashalm_
Oracle Labs

DSLDI Summer School
2015



One Language to Rule Them All?

Let's ask a search engine…

JavaScript: **One language to rule them all** | VentureBeat

venturebeat.com/2011/.../javascript-**one-language-to-rule-them**-...

by Peter Yared - in 23 Google+ circles

Jul 29, 2011 - Why code in two different scripting languages, one on the client and one on the server? It's time for **one language to rule them all**. Peter Yared ...

[PDF] Python: **One** Script (**Language**) **to rule them all** - Ian Darwin

www.darwinsys.com/python/python4unix.pdf

Another **Language**? ▸ Python was invented in 1991 by Guido van. Rossum. ▫ Named after the comedy troupe, not the snake. ▸ Simple. ▫ They **all** say that!

Q & Stuff: **One Language to Rule Them All - Java**

qstuff.**blogspot**.com/2005/10/**one-language-to-rule-them-all-java**.html

Oct 10, 2005 - **One Language to Rule Them All - Java**. For a long time I'd been hoping to add a scripting language to LibQ, to use in any of my (or other ...

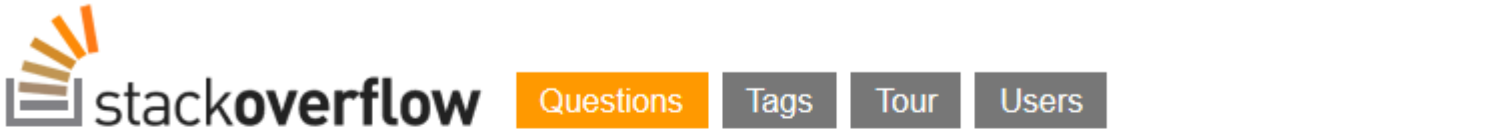
Dart : **one language to rule them all** - MixIT 2013 - Slideshare

fr.slideshare.net/sdeleuze/dart-mixit2013en

DartSébastien Deleuze - @sdeleuzeMix-IT 2013One **language to rule them all** ...

One Language to Rule Them All?

Let's ask Stack Overflow…

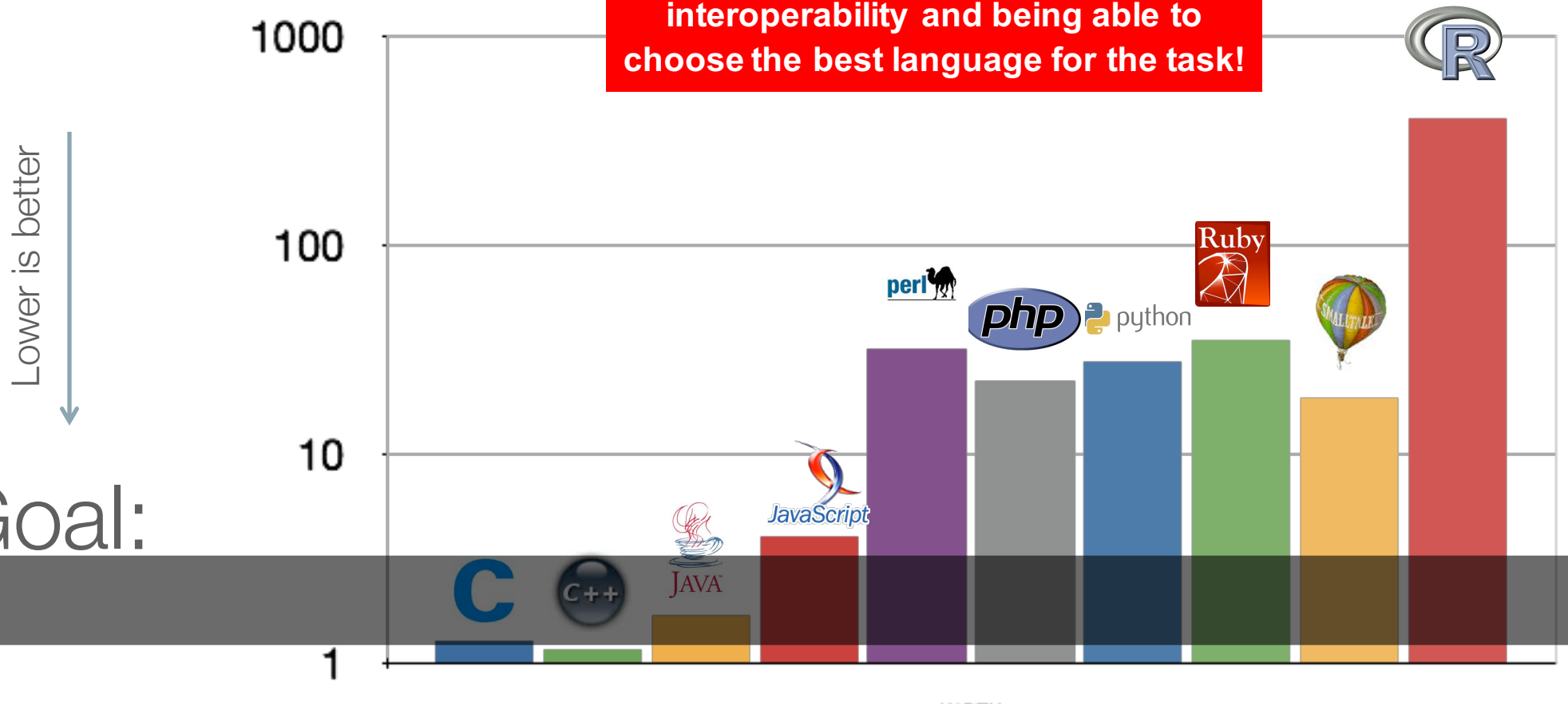


Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

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

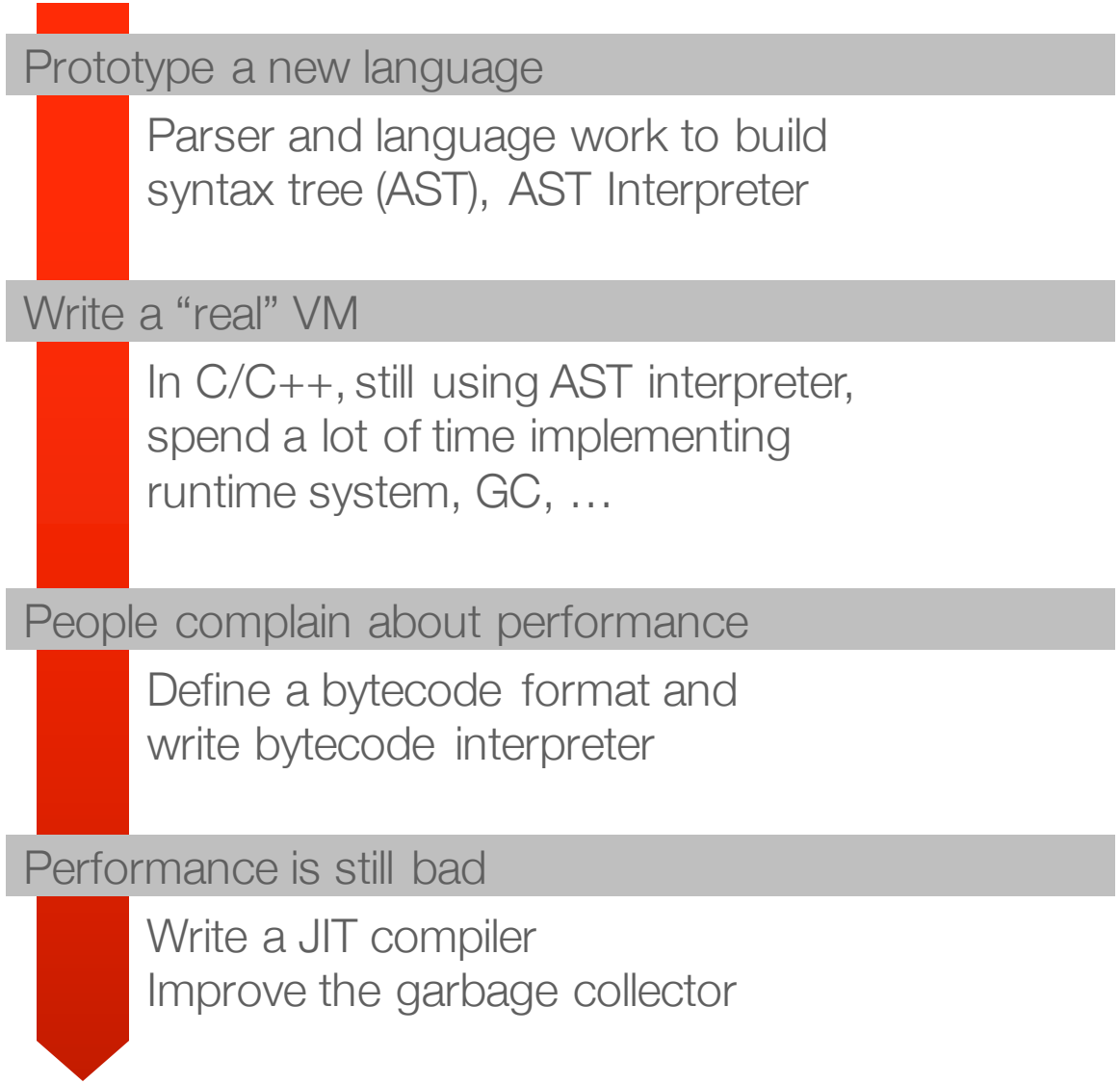
One VM for all languages means interoperability and being able to choose the best language for the task!



Goal:

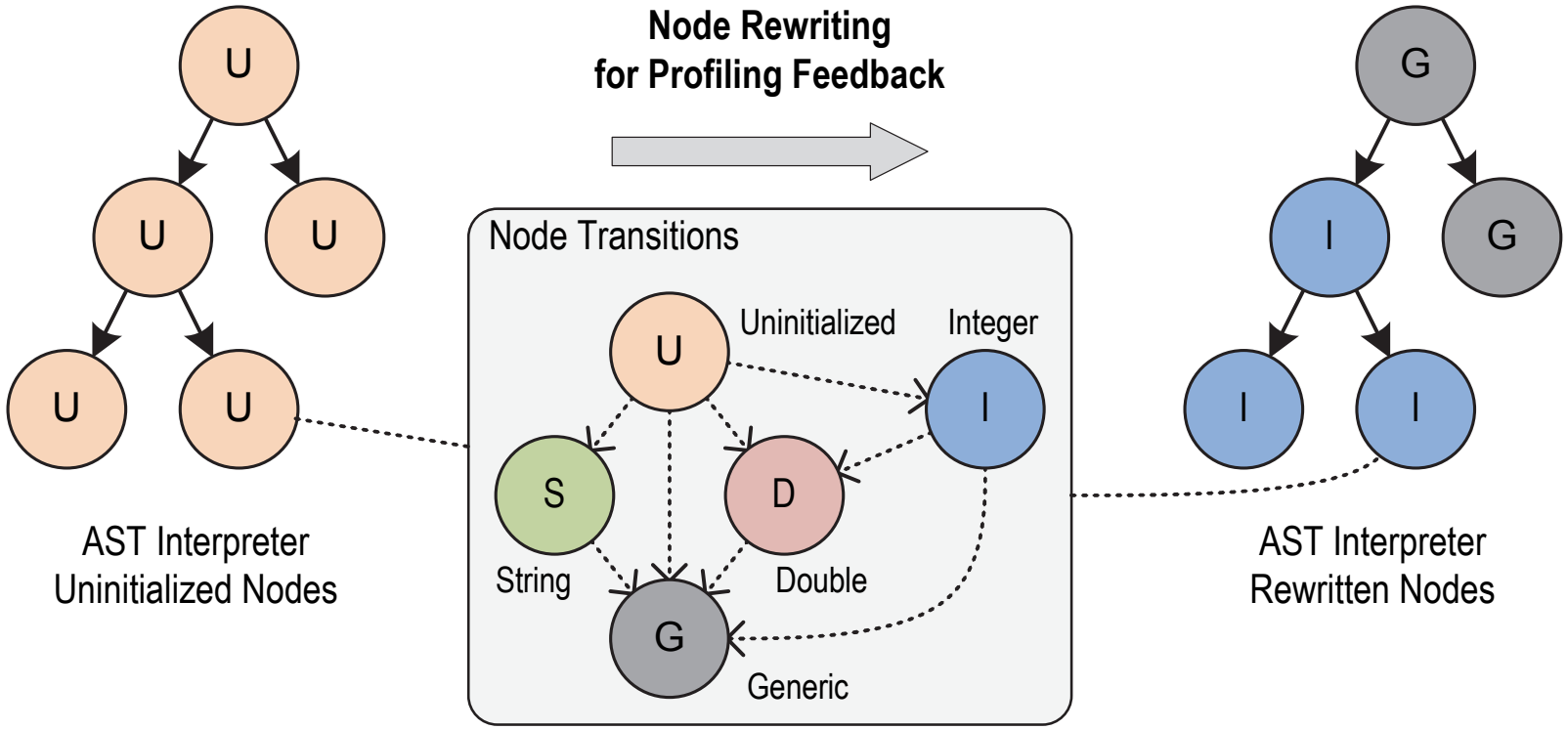
You can execute any language on the JVM / CLR

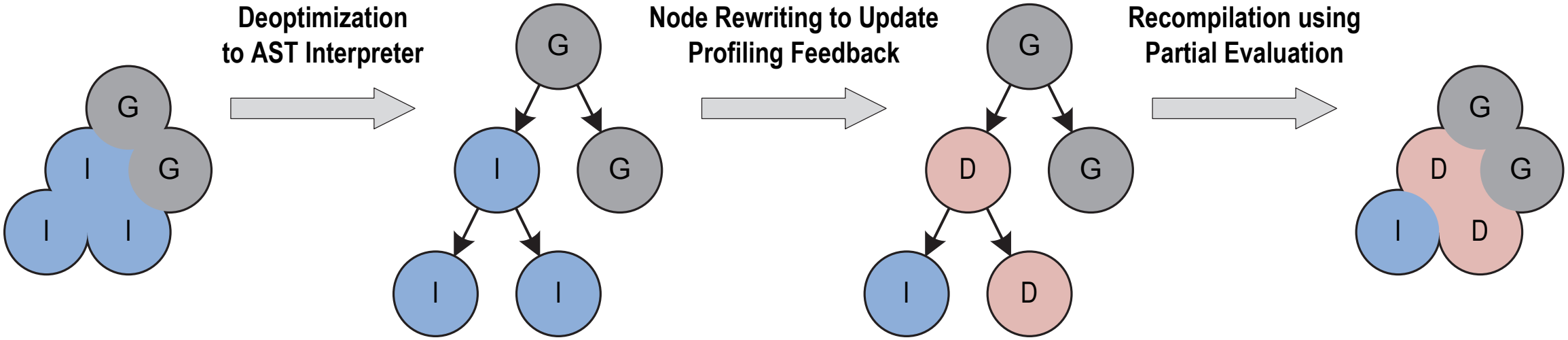
- as long as it looks like Java / C#.



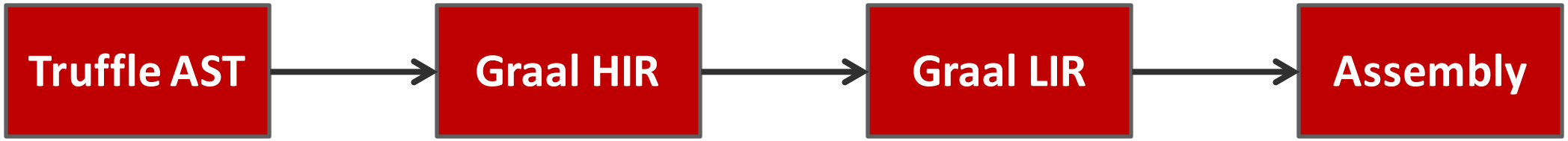
Prototype a new language

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





Compilation Pipeline



Next up: Demos – Part I

- Partial evaluation and compilation
  - Compiler graphs and assembly
- How I stopped worrying and learned to love abstractions!
- The profiling, compilation, deoptimization, reprofiling, recompilation cycle.

Truffle API

```
class ANode extends Node {

    public int execute() {
        return 21 + 21;
    }

}
```

```
class ARootNode extends RootNode {
    @Child ANode childNode = new ANode();
    @Override
    public Object execute(VirtualFrame arg0) {
        return childNode.execute();
    }
}
```

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

Truffle API

```
public interface TruffleRuntime {

    CallTarget createCallTarget(RootNode rootNode);

    DirectCallNode createDirectCallNode(CallTarget target);

    IndirectCallNode createIndirectCallNode();

    Assumption createAssumption();

    <T> T iterateFrames(FrameInstanceVisitor<T> visitor);
    ...
}
```

Truffle API

```
public class CompilerDirectives {

    public static void transferToInterpreter() {...}

    public static void transferToInterpreterAndInvalidate() {...}

    public @interface CompilationFinal {}

    public @interface ValueType {}

    public @interface TruffleBoundary {}

    ...
}
```

Truffle API

Used in the next examples

```
public abstract class Node {
    ...
}
```

```
public final class CompilerDirectives {

    public static void transferToInterpreterAndInvalidate() {...}

    public @interface CompilationFinal {}
    ...

}
```

Truffle API Example

```
class NegateNode extends Node {

    @CompilationFinal boolean minValueVisited;

    public int execute(int operand) {
        if (operand == Integer.MIN_VALUE) {
            if (!minValueVisited) {
                transferToInterpreterAndInvalidate();
                minValueVisited = true;
            }
            return Integer.MAX_VALUE;
        }
        return -operand;
    }
}
```

minValueVisited = true

```
if (operand == Integer.MIN_VALUE) {
    return Integer.MAX_VALUE;
}
return -operand;
```

minValueVisited = false

```
if (operand == Integer.MIN_VALUE) {
    transferToInterpreterAndInvalidate();
}
return -operand;
```

Branch Profiles

```
class NegateNode extends Node {

    final BranchProfile minValueProfile = BranchProfile.create();

    public int execute(int operand) {
        if (operand == Integer.MIN_VALUE) {
            minValueProfile.enter();
            return Integer.MAX_VALUE;
        }
        return -operand;
    }
}
```

Condition Profiling

```
class AbsNode extends Node {

    final ConditionProfile smallerZero = ConditionProfile.createBinaryProfile();

    public int execute(int operand) {
        if (smallerZero.profile(operand < 0)) {
            return -operand;
        } else {
            return operand;
        }
    }
}
```

Identity Profiling

```
public class IdentityValueProfile extends ValueProfile {
    private static final Object UNINITIALIZED = new Object();
    private static final Object GENERIC = new Object();

    @CompilationFinal private Object cachedValue = UNINITIALIZED;

    public <T> T profile(T value) {
        if (cachedValue != GENERIC) {
            if (cachedValue == value) {
                return (T) cachedValue;
            } else {
                transferToInterpreterAndInvalidate();
                if (cachedValue == UNINITIALIZED) {
                    cachedValue = value;
                } else {
                    cachedValue = GENERIC;
                }
            }
        }
        return value;
    }
```

Type Profiling

```
public class ExactClassValueProfile extends ValueProfile {

    @CompilationFinal protected Class<?> cachedClass;
    @Override
    public <T> T profile(T value) {
        if (cachedClass != Object.class) {
            if (cachedClass != null && cachedClass.isInstance(value)) {
                return (T) cachedClass.cast(value);
            } else {
                CompilerDirectives.transferToInterpreterAndInvalidate();
                if (cachedClass == null) {
                    cachedClass = value.getClass();
                } else {
                    cachedClass = Object.class;
                }
            }
        }
        return value;
    }
}
```

Profiles: Summary

- BranchProfiles to speculate on unlikely branches
- ConditionProfile to speculate on binary conditions
- Identity Profiles to speculate on constant values
- Type Profiles to speculate on constant type
- …

Profiles: Limitations

- Polymorphism:
  - profiles only work with monomorphic situations
  - requires the use of inline caches

- For local speculation only:
  - *`transferToInterpreterAndInvalidate()`* just invalidates the current compilation unit.
  - requires the use of non-local assumptions

Non-local assumptions

```
public interface Assumption {

    boolean isValid();

    void invalidate();
}

Assumption a = Truffle.getRuntime().createAssumption();
```

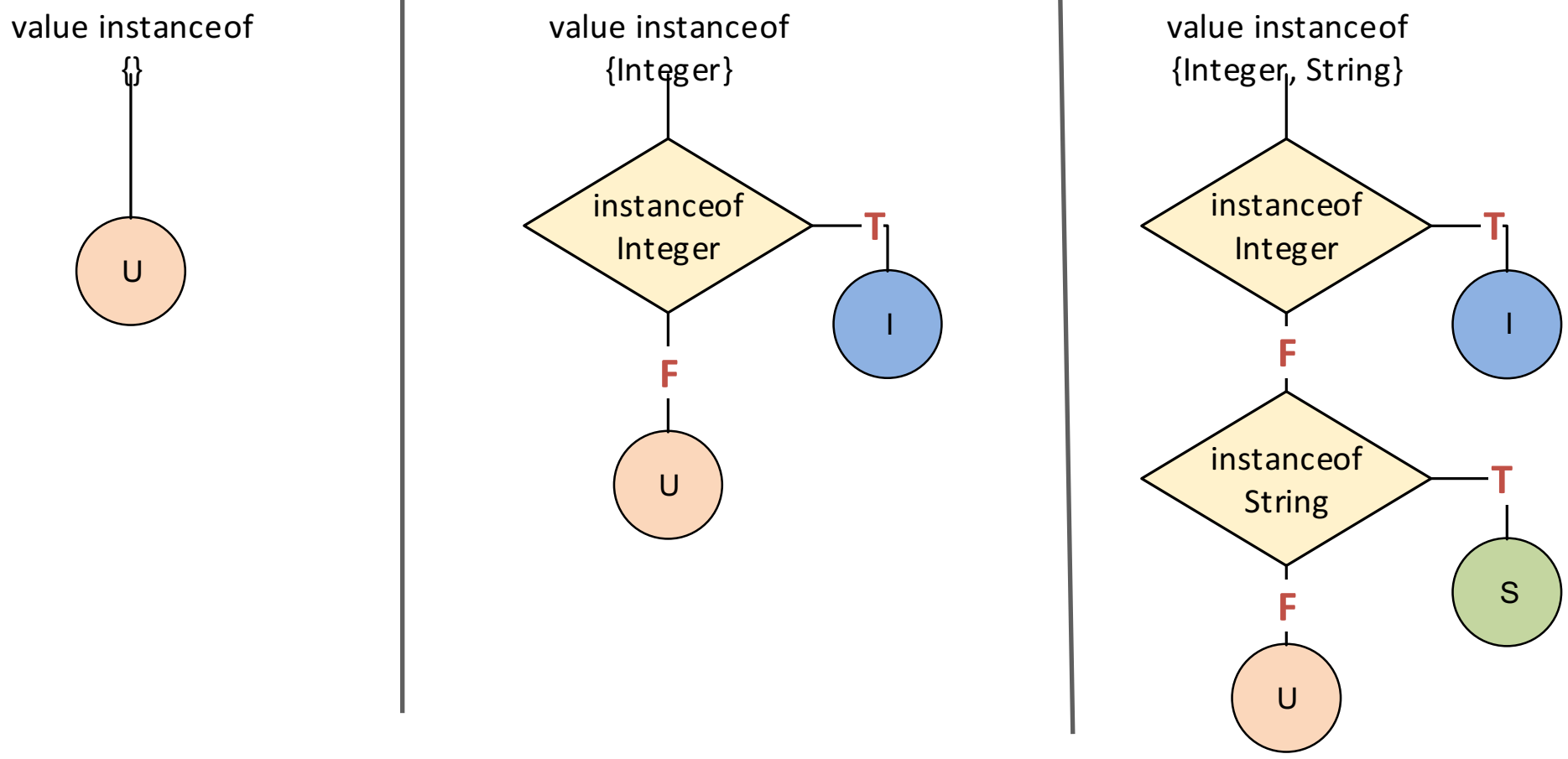
Non-local assumptions

```
public class ANode extends Node {

    private final Assumption assumption = getInstrumentationDisabled();

    public void execute() {
        if (assumption.isValid()) {
            // do nothing
        } else {
            // do instrument
        }
    }
}
```

Use-cases for non-local assumptions

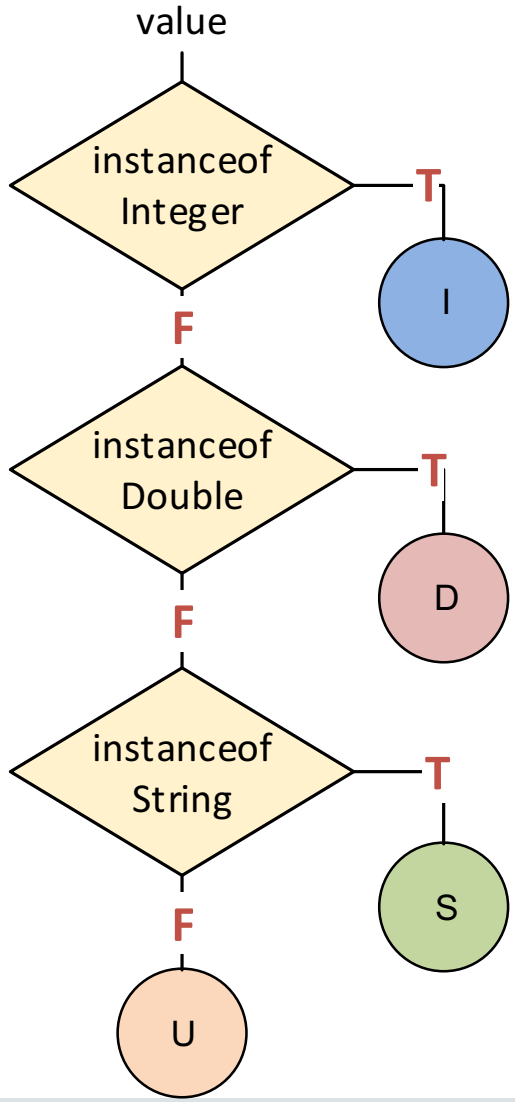
- Function redefinition
- Assumed global values
- Tracing / Debugging / Instrumentation
- ...

Inline Caching



Inline Caching using Truffle DSL

```
class OperationNode extends Node {

    @Specialization
    int doInt(int value) {
        // int implementation
    }

    @Specialization
    double doDouble(double value) {
        // double implementation
    }

    @Specialization
    String doString(String value) {
        // String implementation
    }
}
```



Identity Inline Caching

```
public abstract class ANode extends Node {

    public abstract Object execute(Object operand);

    @Specialization(guards = "operand == cachedOperand", limit = "3")
    protected Object doCached(AType operand,
                    @Cached("operand") AType cachedOperand) {
        // implementation
        return cachedOperand;
    }

    @Specialization(contains = "doCached")
    protected Object doGeneric(AType operand) {
        // implementation
        return operand;
    }
}
```

Type Inline Caching

```
public abstract class ANode extends Node {

    public abstract Object execute(Object operand);

    @Specialization(guards = "operand.getClass() == cachedClass", limit = "3")
    protected Object doCached(AType operand,
                    @Cached("operand.getClass()") Class<? extends AType> cachedClass) {
        AType operand = cachedClass.cast(operand);
        // implementation
        return operand2;
    }

    @Specialization(contains = "doCached")
    protected Object doGeneric(AType operand) {
        // implementation
        return operand;
    }
}
```

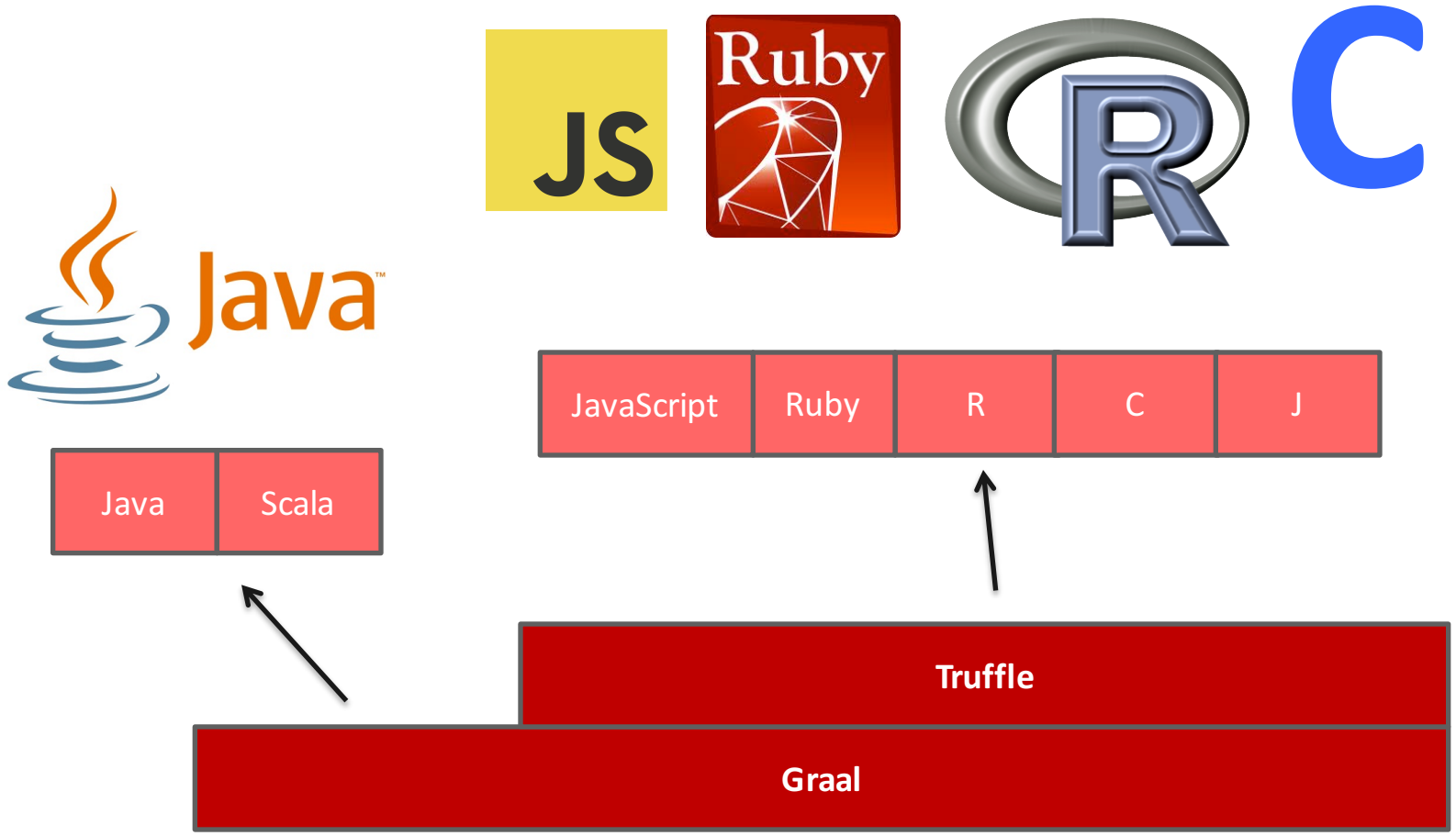
Truffle Speculations

Profile, Inline Cache or Assumption?

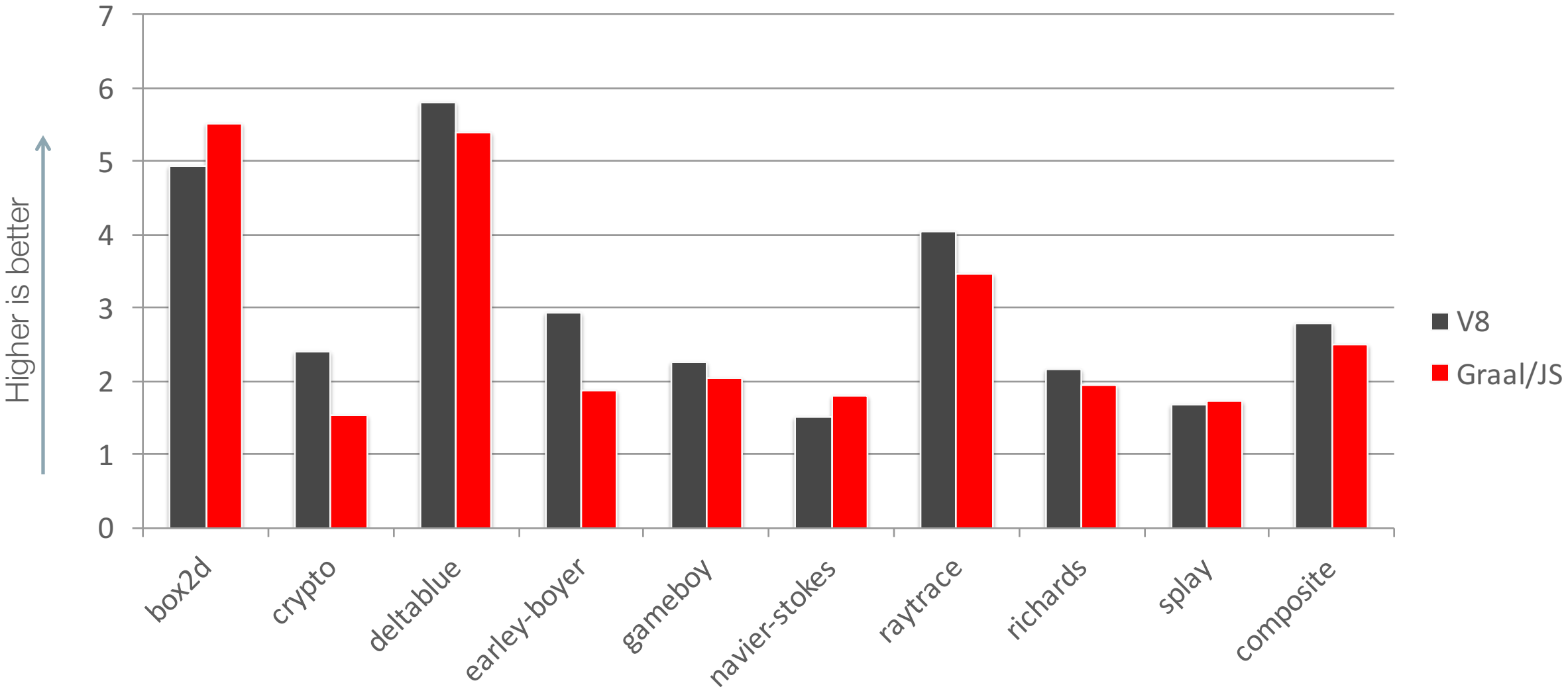
- Use Profiles where monomorphic speculation is sufficient
- Use Inline Caches for speculations where polymorphism is required
- Use Assumptions for non-local, global speculation

Next up: Demos – Part II

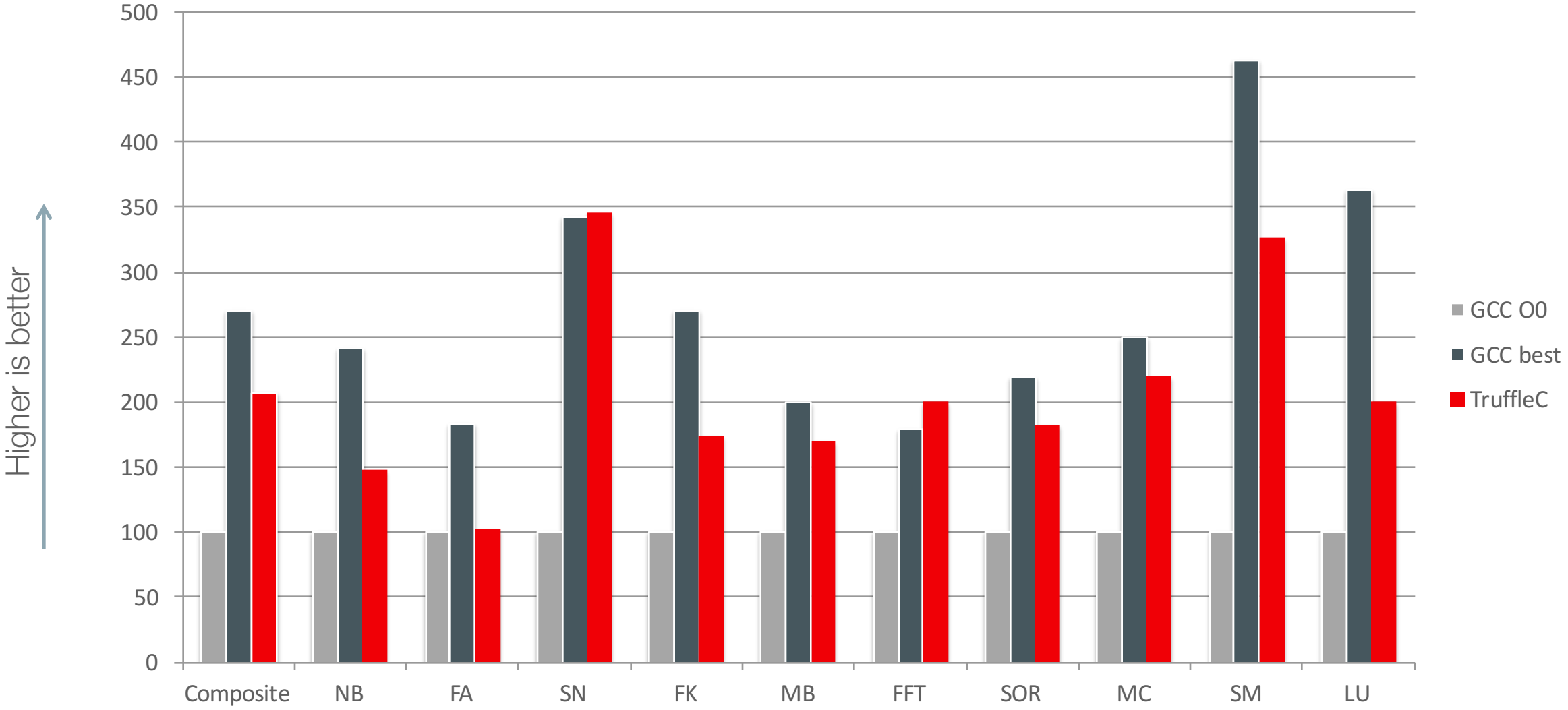
- SimpleLanguage:
  - Demonstration language for Truffle features (well documented)
- Division speculation
- Zero-overhead tracing

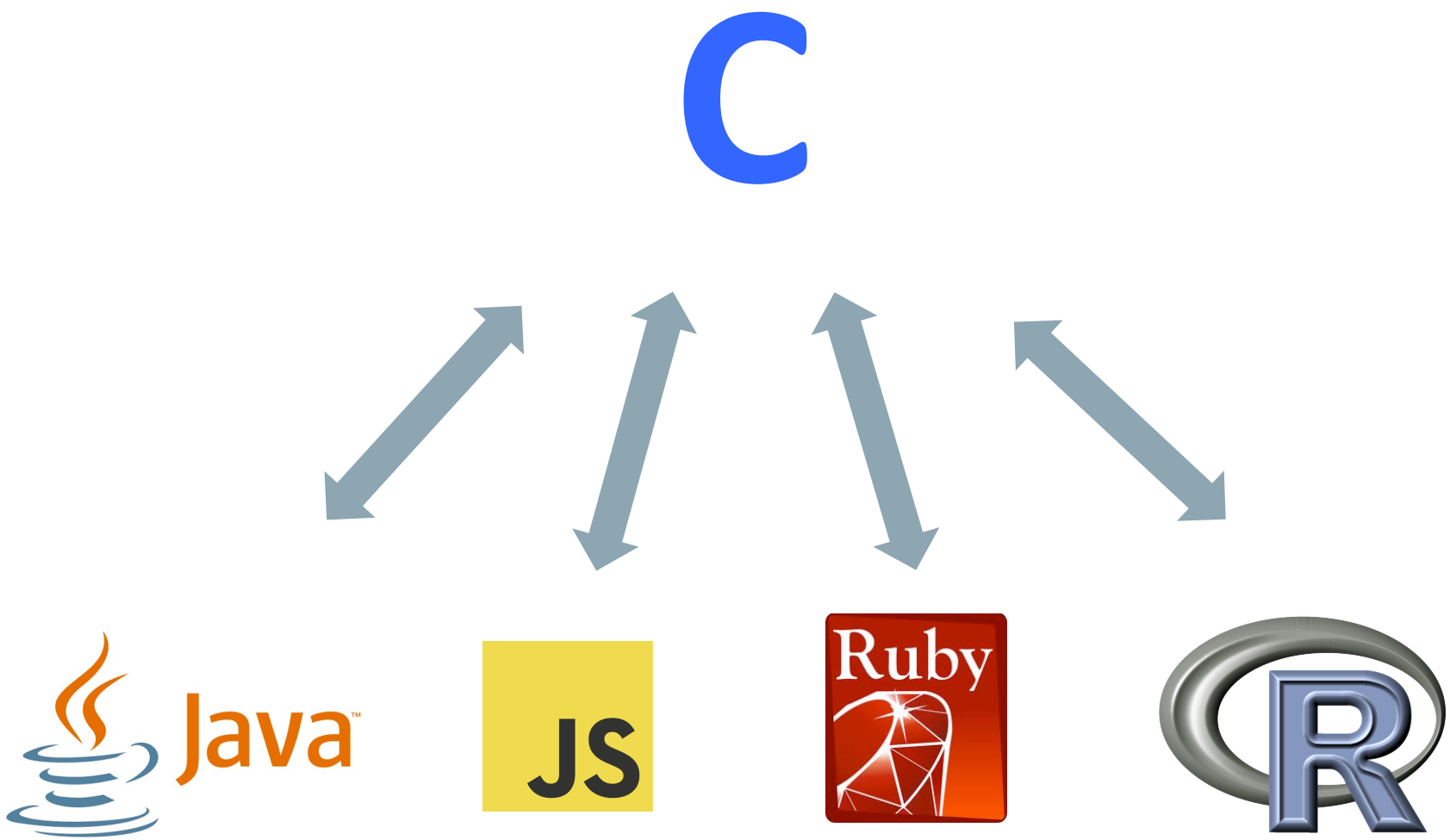


Performance – JavaScript



Performance – C





stackoverflow

Questions | Tags | Users | Badges

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

Why shouldn't I use PyPy over CPython if PyPy is 6.3 times faster?

345

87

I've been hearing a lot about the PyPy project. They claim it is 6.3 times faster than the CPython interpreter on their site.

Whenever we talk about dynamic languages like Python, speed is one of the top issues. To solve this, they say PyPy is 6.3 times faster.

The second issue is parallelism, the infamous Global Interpreter Lock (GIL). For this, PyPy says it can give GIL-less Python.

If PyPy can solve these great challenges, what are its weaknesses that are preventing wider adoption? That is to say, what's preventing someone like me, a typical Python developer, from switching to PyPy *right now*?

python | performance | jit | pypy | cpython

ORACLE

Questions Tags Users Badges

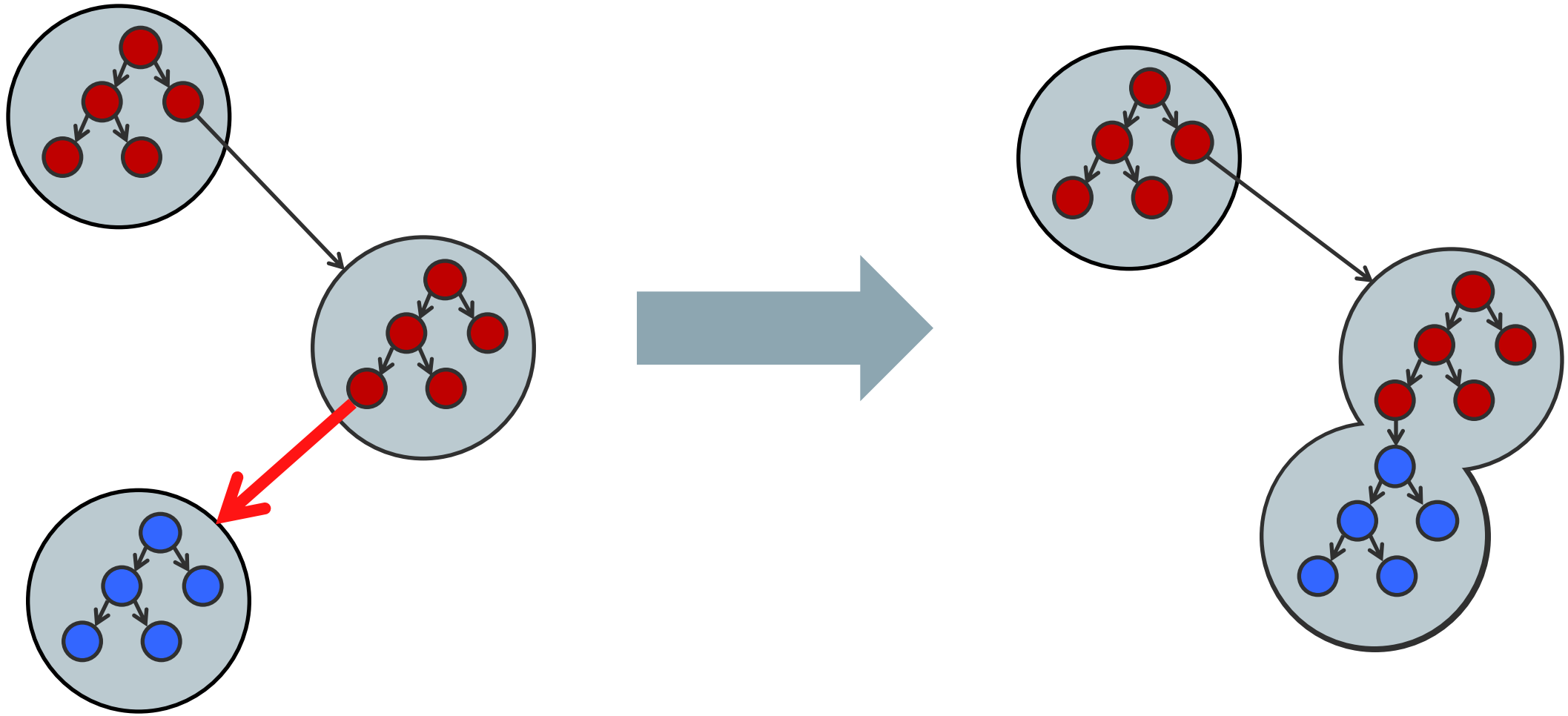
Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

422

PyPy, as others have been quick to mention, has tenuous support for C extensions. It *has* support, but typically at slower-than-Python speeds and it's iffy at best. Hence a lot of modules simply *require* CPython. Cython and Numpy are *awesome* for numerics, and most people who actually need speed in Python are using those (+ Pandas, SciPy, etc.) heavily. Since they're either non-existent or tenuously supported and slow **the people who need a fast Python often are better off with CPython both for speed and ease-of-use**.

If PyPy can solve these great challenges, what are its weaknesses that are preventing wider adoption? That is to say, what's preventing someone like me, a typical Python developer, from switching to PyPy *right now*?

python performance jit pypy cpython



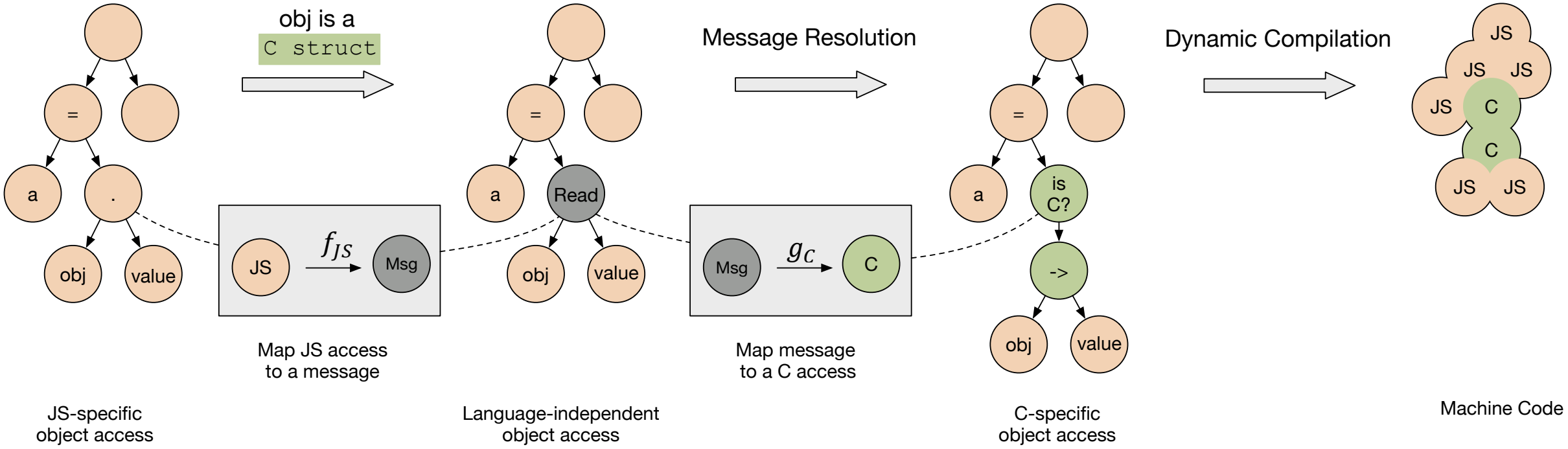
main.c

```
#include<stdio.h>

struct complex {
   double r;
   double i;
}


int main() {
        struct complex *a = …;
        struct complex *b = …;

        add(a, b)
}
```

complex.js

```
function add(a, b) {
   var result = {r:0, i:0};

   result.r = a->r + b->r

   result.i = a->i + b->i

   return result;
}
```

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



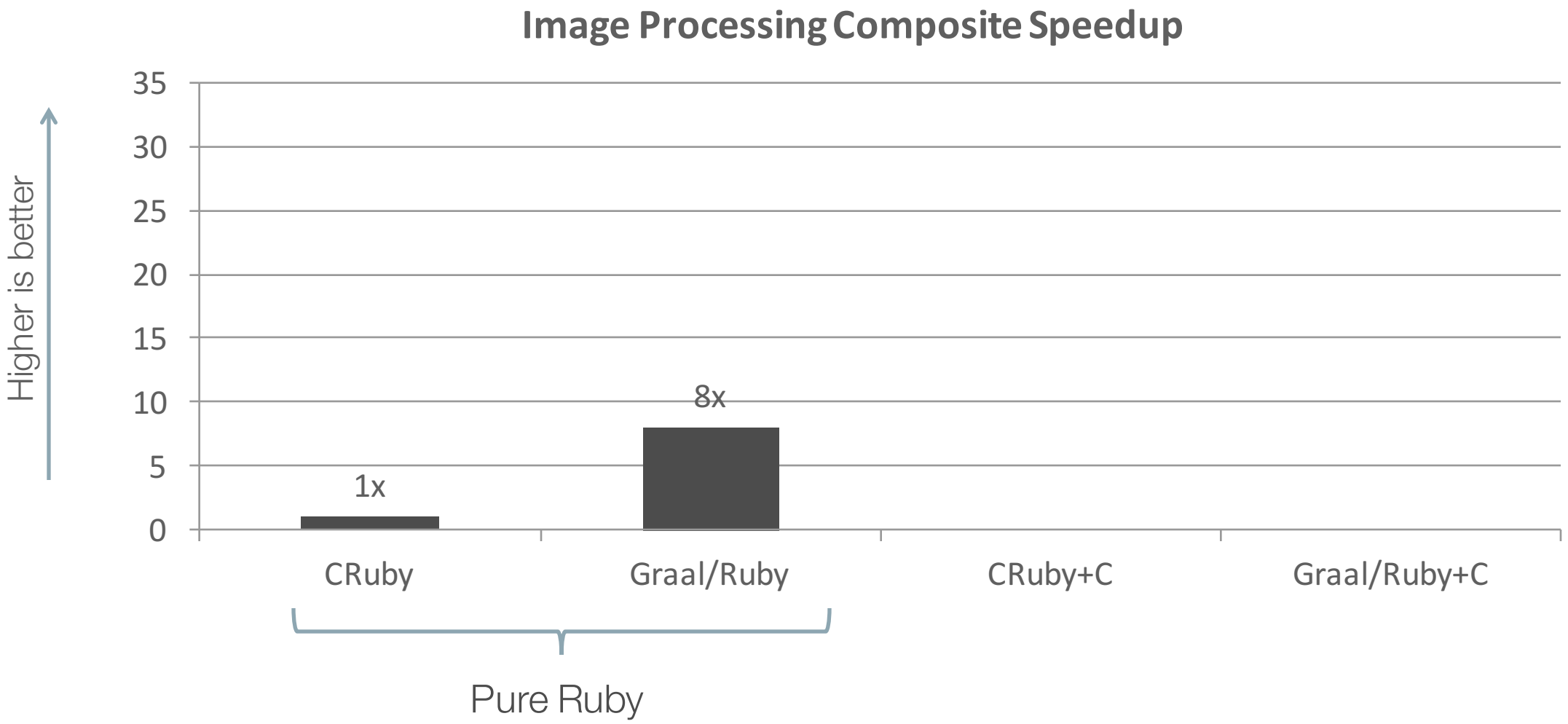
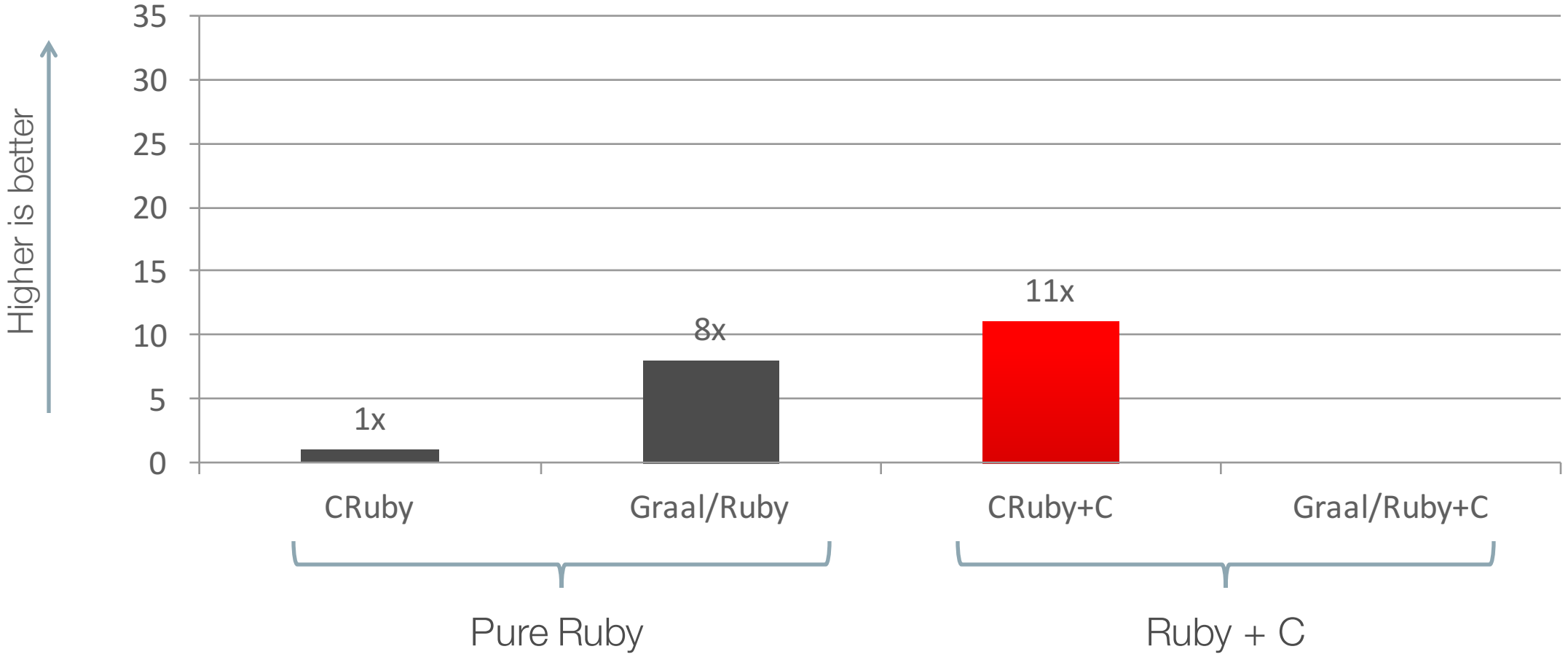
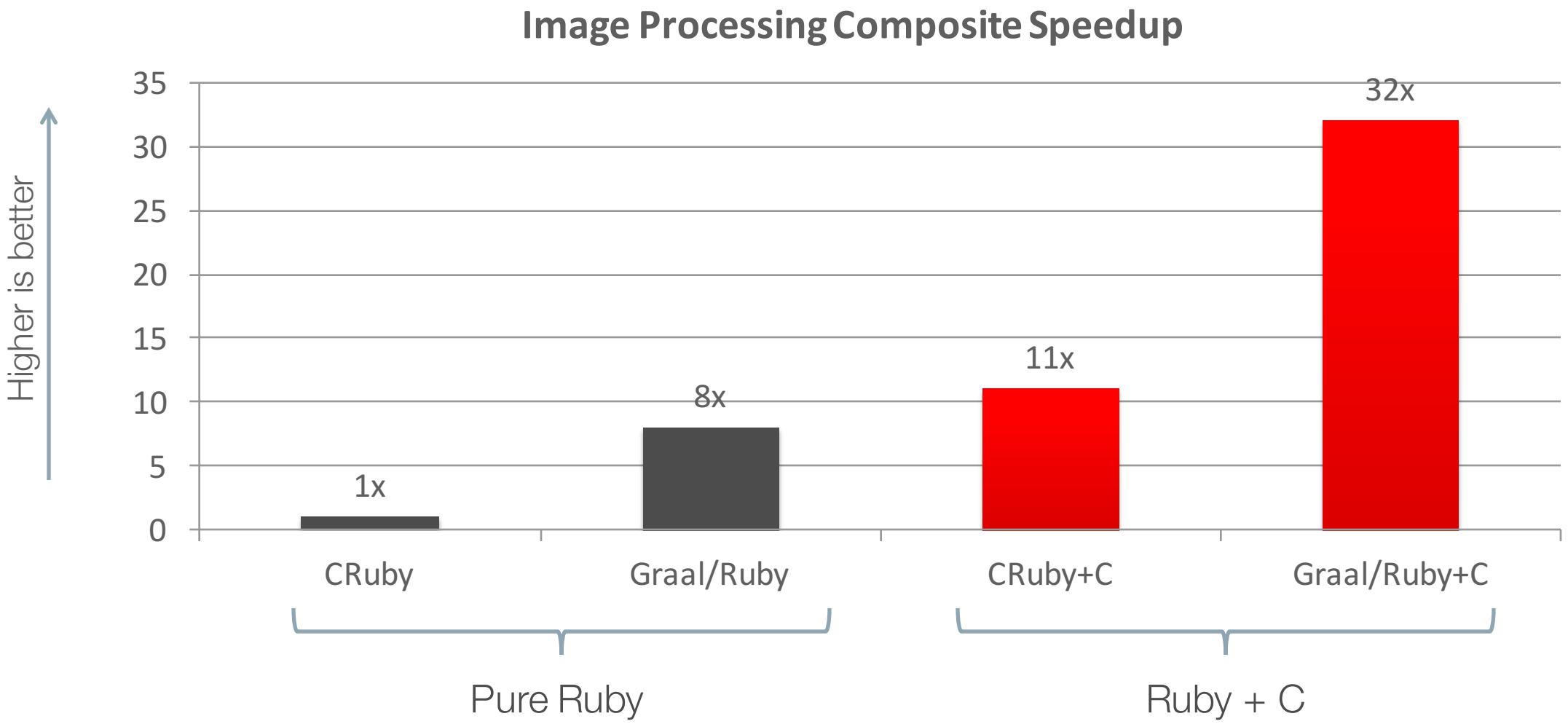


Image Processing Composite Speedup





Acknowledgements

Oracle Labs
Danilo Ansaloni
Stefan Anzinger
Daniele Bonetta
Matthias Brantner
Petr Chalupa
Laurent Daynès
Gilles Duboscq
Michael Haupt
Christian Humer
Mick Jordan
Peter Kessler
Hyunjin Lee
David Leibs
Kevin Menard
Tom Rodriguez
Roland Schatz
Chris Seaton
Doug Simon
Lukas Stadler
Jaroslav Tulach

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 Leopoltseder
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

University of Edinburgh
Christophe Dubach
Juan José Fumero Alfonso
Ranjeet Singh
Toomas Remmelg

LaBRI

Floréal Morandat

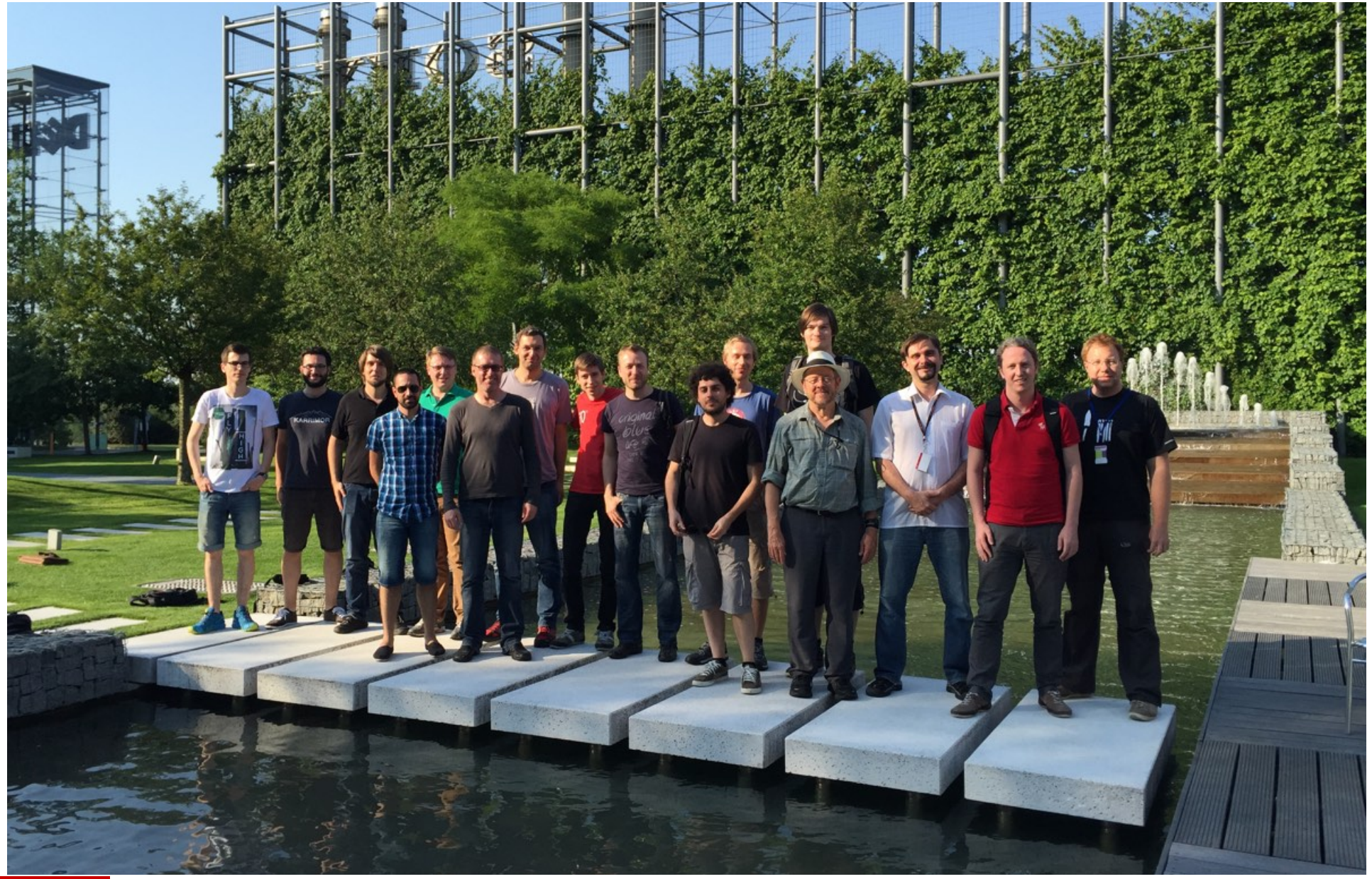
University of California, Irvine
Prof. Michael Franz
Codrut Stancu
Gulfem Savrun Yeniceri
Wei Zhang

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

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



QA

@thomaswue

@grashalm_

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. 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.