ORACLE

Graal and Truffle: Modularity and Separation of Concerns as Cornerstones for Building a Multipurpose Runtime

Thomas Wuerthinger
Oracle Labs
@thomasswue

24-April-2014,
Keynote at MODULARITY in Lugano

Disclaimer

Agenda

- **Graal**
- Truffle
- Community
- Q&A

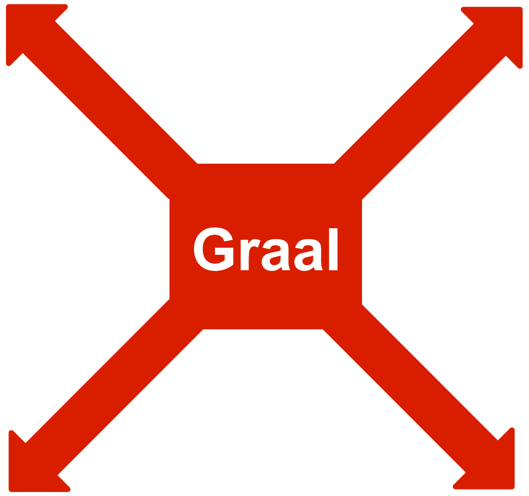
Dimensions of Extensibility

X86, SPARC, HSAIL, PTX, ...

Architectures

HotSpotVM, SubstrateVM, ...

Host Runtimes



Compilation Policy

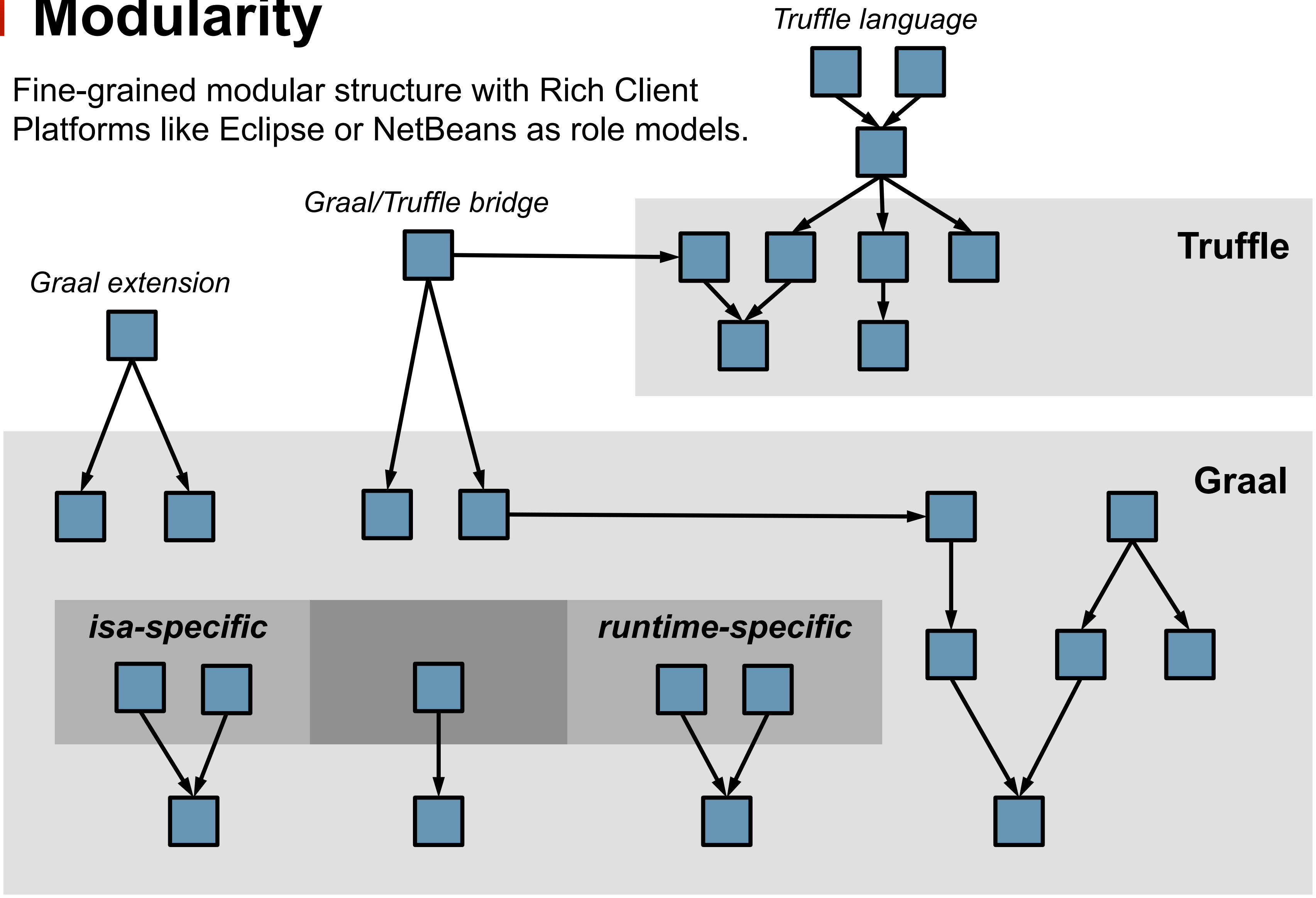
Baseline, Mid-tier, Optimizing, Vectorizing, ...

Programming Languages

Java, JavaScript, Python, R, Ruby, C, Smalltalk, ...

Modularity

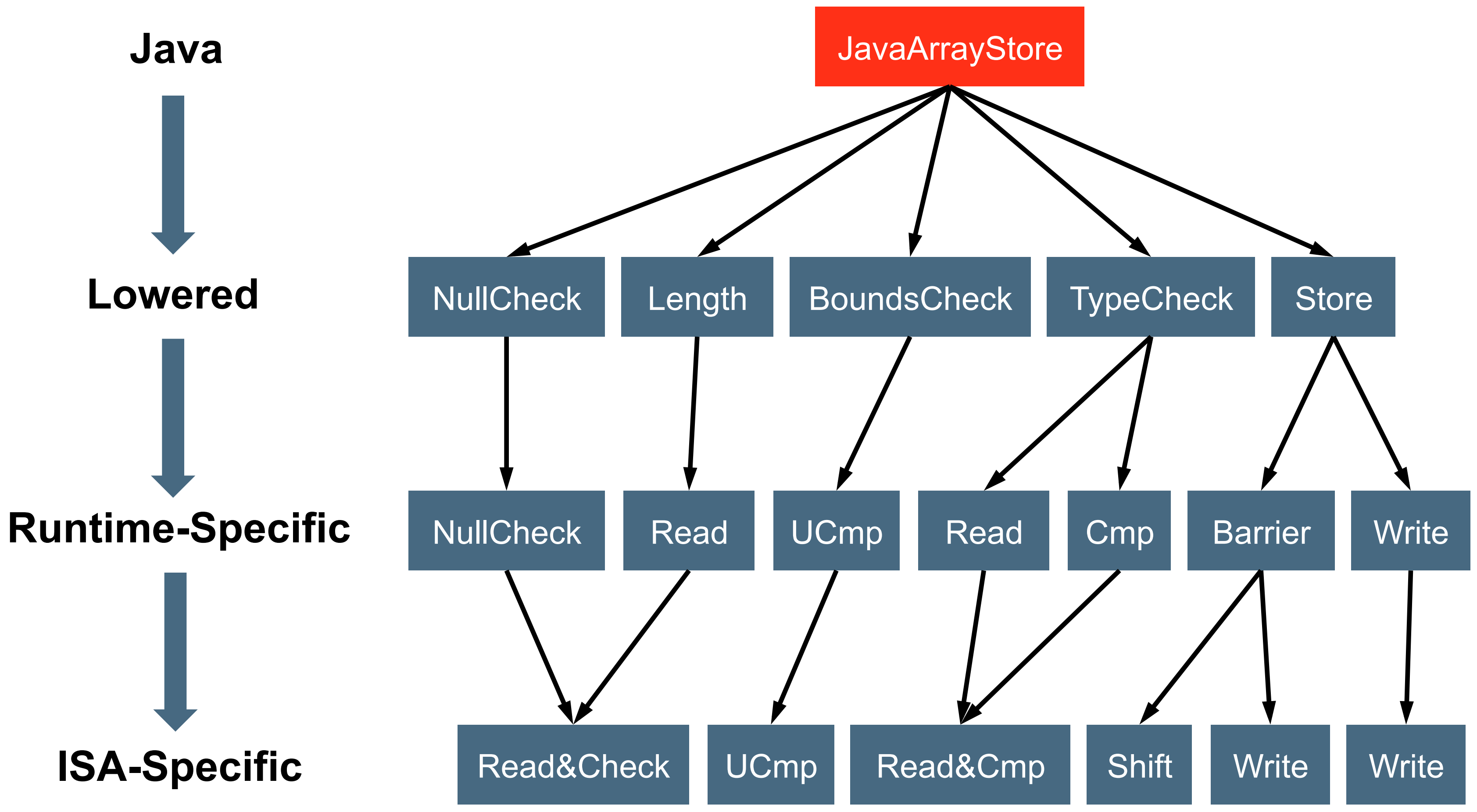
Fine-grained modular structure with Rich Client Platforms like Eclipse or NetBeans as role models.



Specific to Host Runtime

- Field/Array Access
  - object/array layout, read/write barriers, …
- Allocation
  - garbage collector, thread-local buffer, …
- Type Checks
  - class hierarchy organization, …
- Locking
  - monitor system, monitor enter/exit, …
- JDK intrinsifications
  - hashCode, clone, reflection, …
- Invocations
- Safepoints

Levels of Lowering



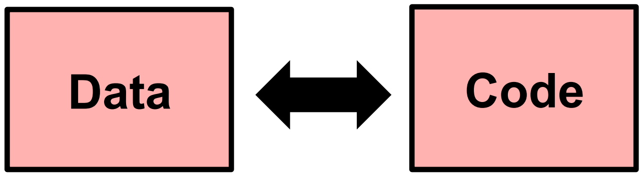
Snippets for Graph Construction

Manual construction:

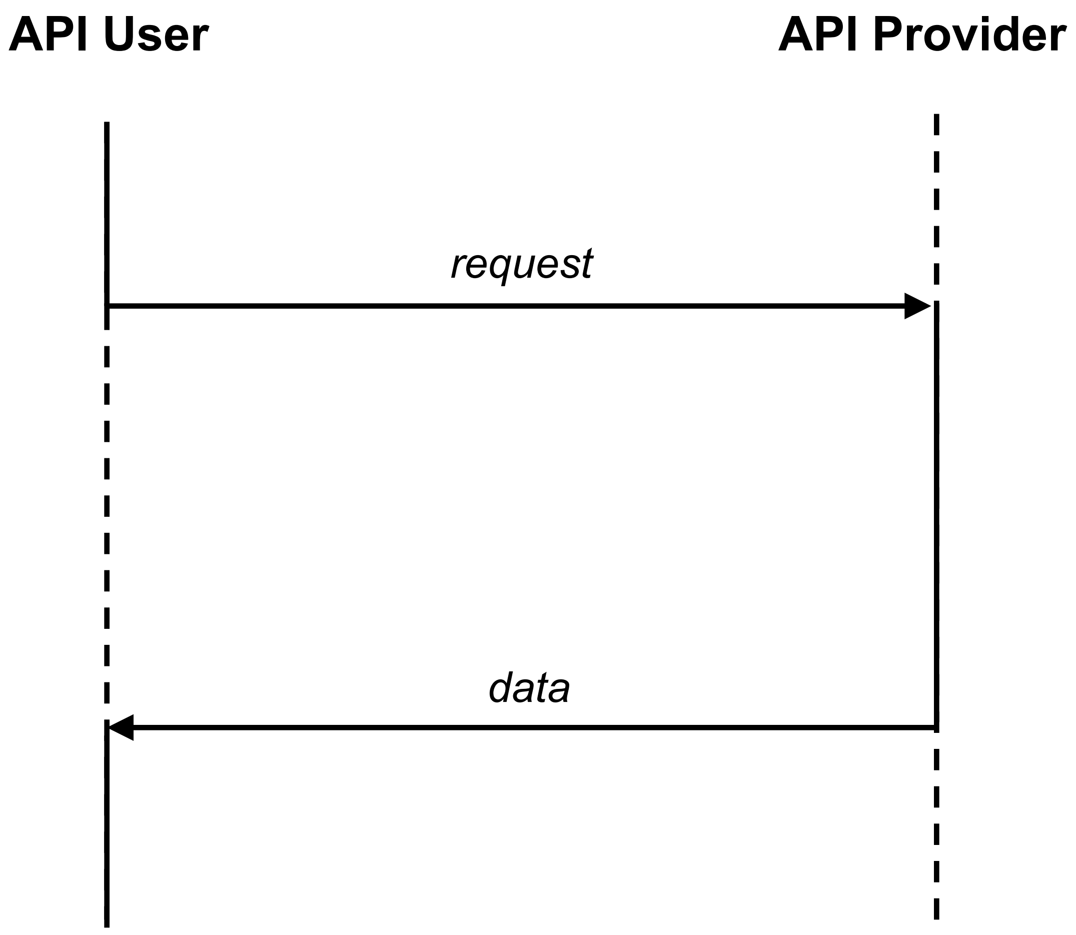
```
Node max(ValueNode a, ValueNode b) {
    IfNode ifNode = new IfNode(new IntegerLessThanNode(a, b));
    ifNode.trueSuccessor().setNext(new ReturnNode(a));
    ifNode.falseSuccessor().setNext(new ReturnNode(b));
    return ifNode;
}
```

Expression as snippet:

```
int max(int a, int b) {
    if (a > b) return a;
    else return b;
}
```



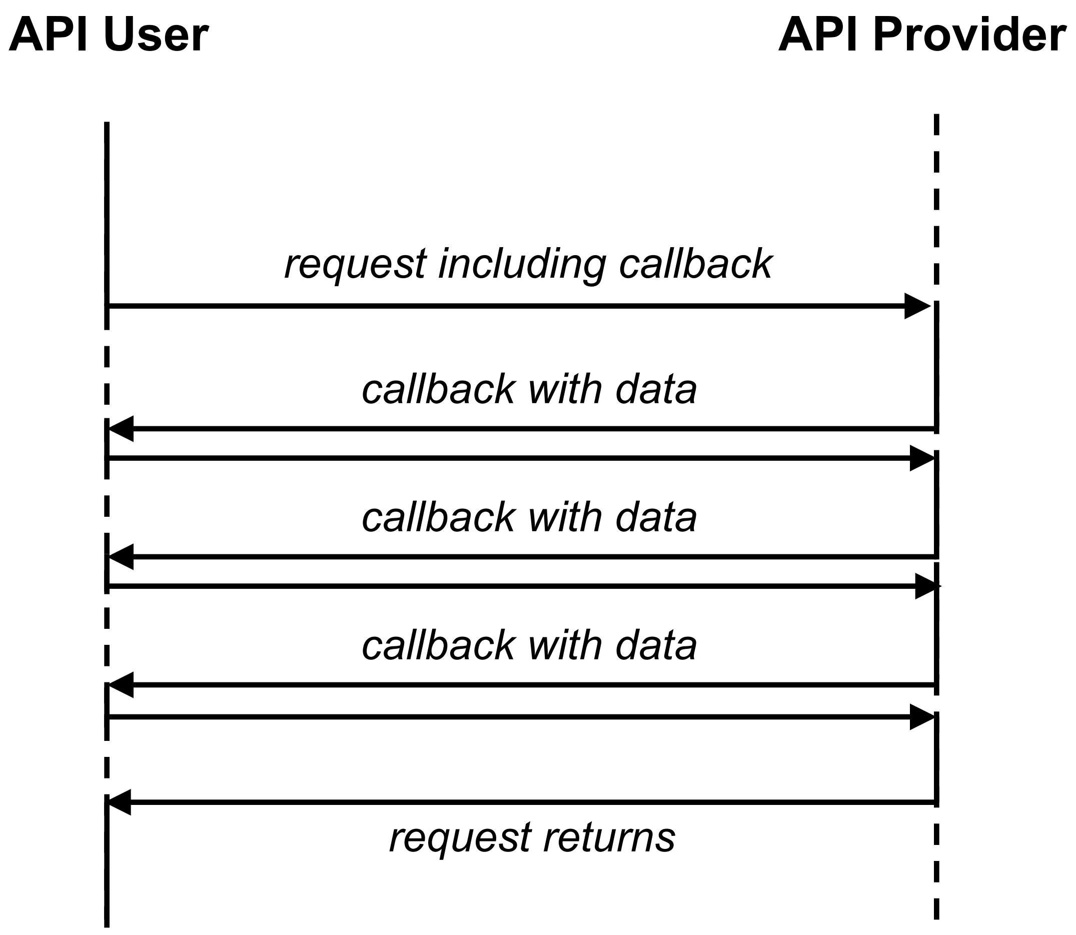
Simple API



✓ Can capture statically

x Limited flexibility

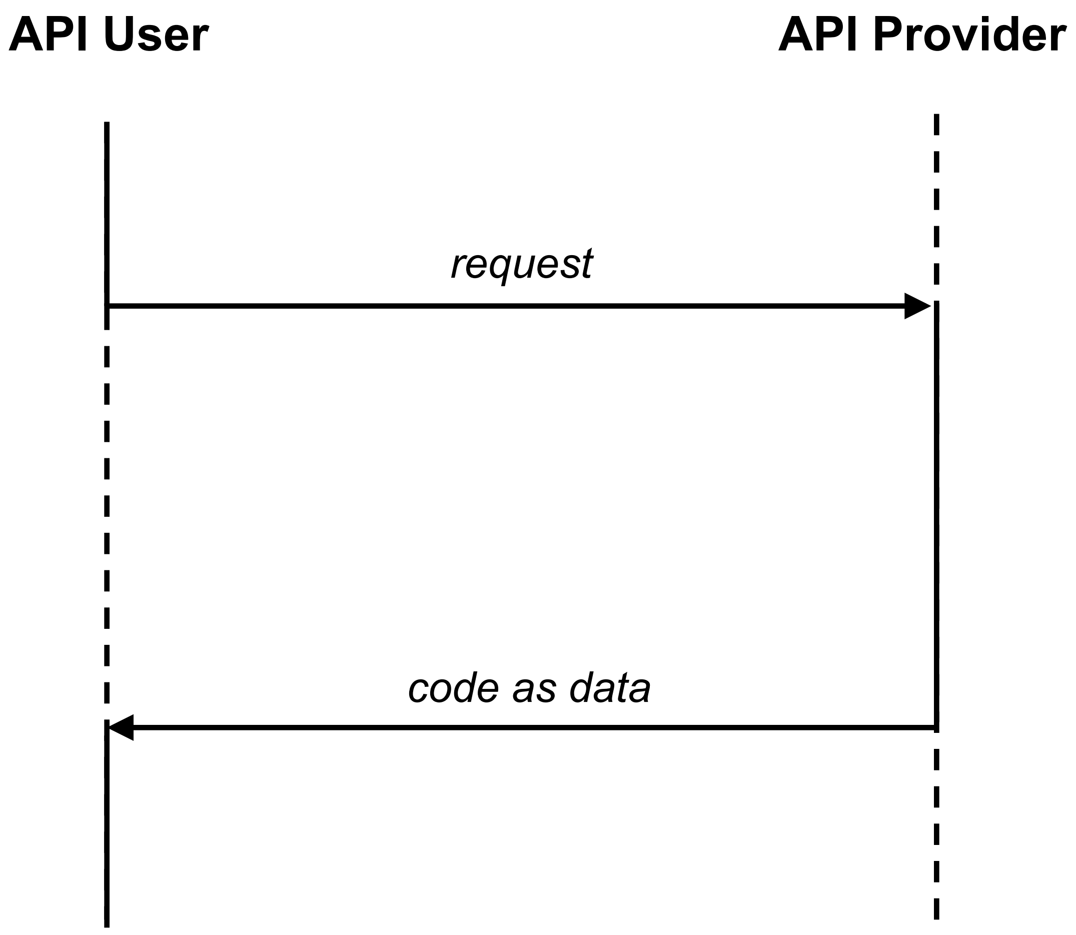
Callback API



✓ High flexibility

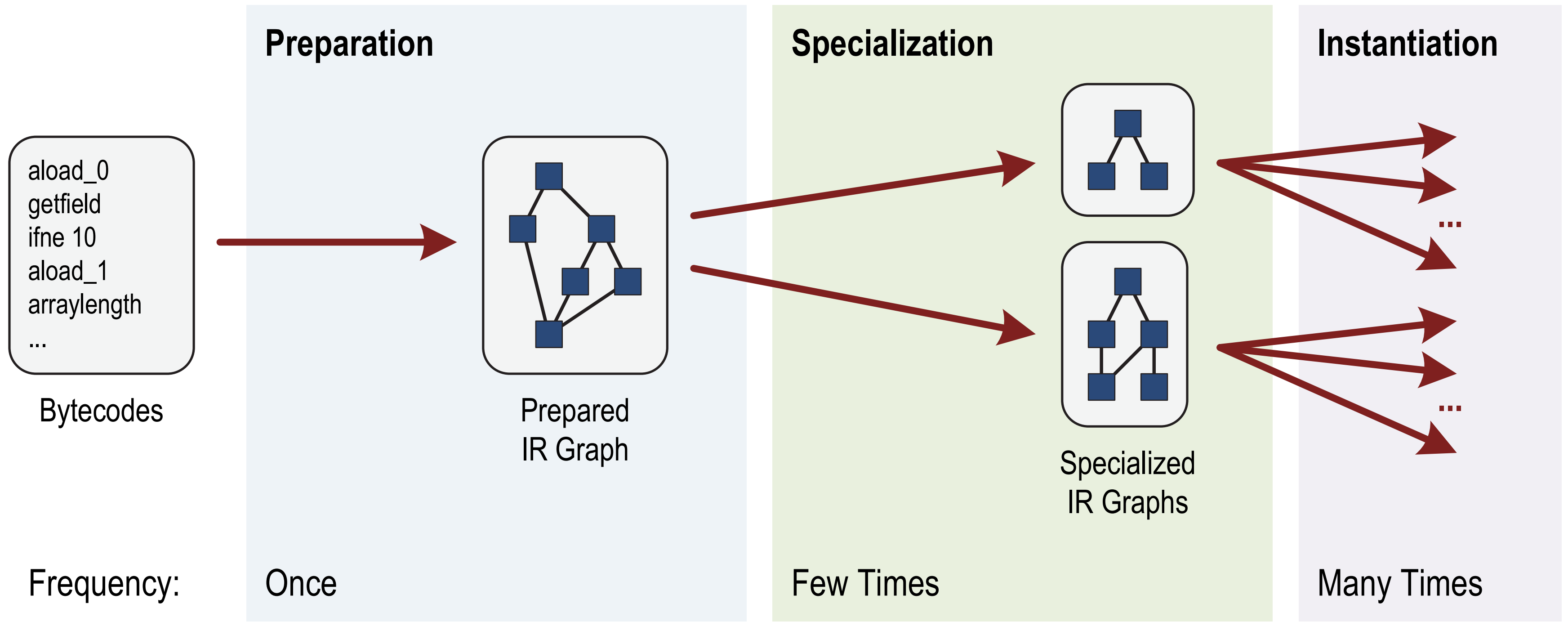
x Cannot capture statically

Snippet API



✓ High flexibility ✓ Can capture statically

Snippet Lifecycle



Snippet Example: Convert

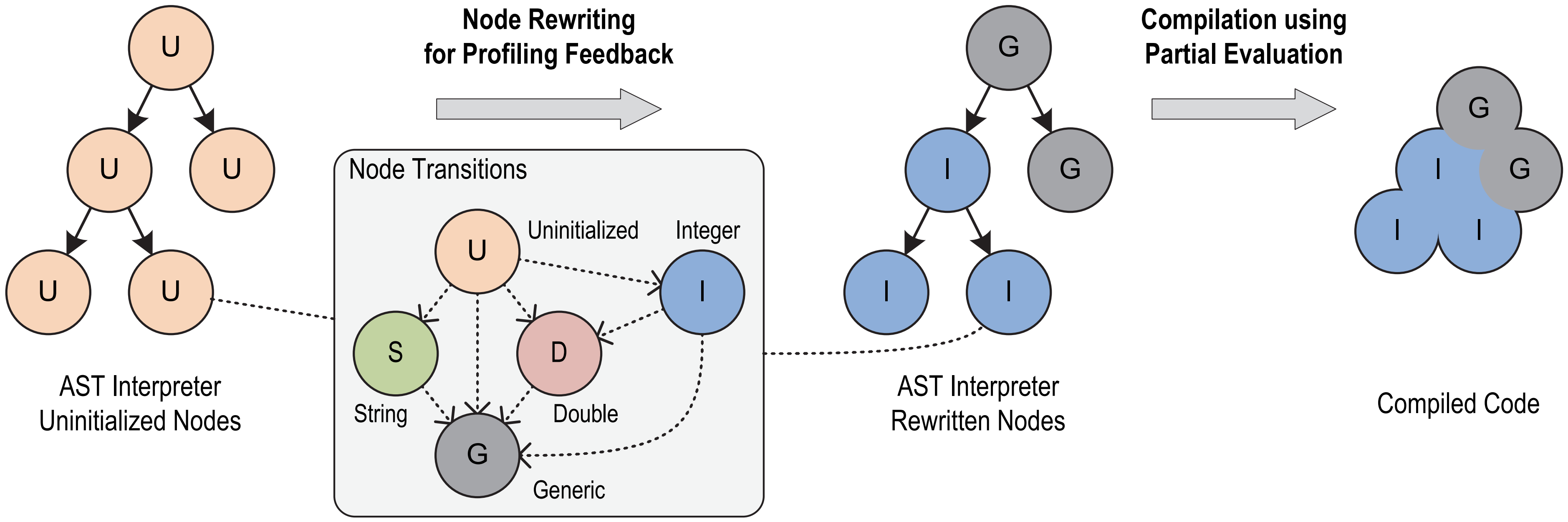
```
@Snippet
static int f2i(float input, int result) {
  if (probability(SLOW_PATH,
                  result == Integer.MIN_VALUE)) {

    if (Float.isNaN(input)) {
      return 0;
    } else if (input > 0.0f) {
      return Integer.MAX_VALUE;
    }
  }
  return result;
}
```

Agenda

- Graal
- **Truffle**
- Community
- Q&A

Technical Approach

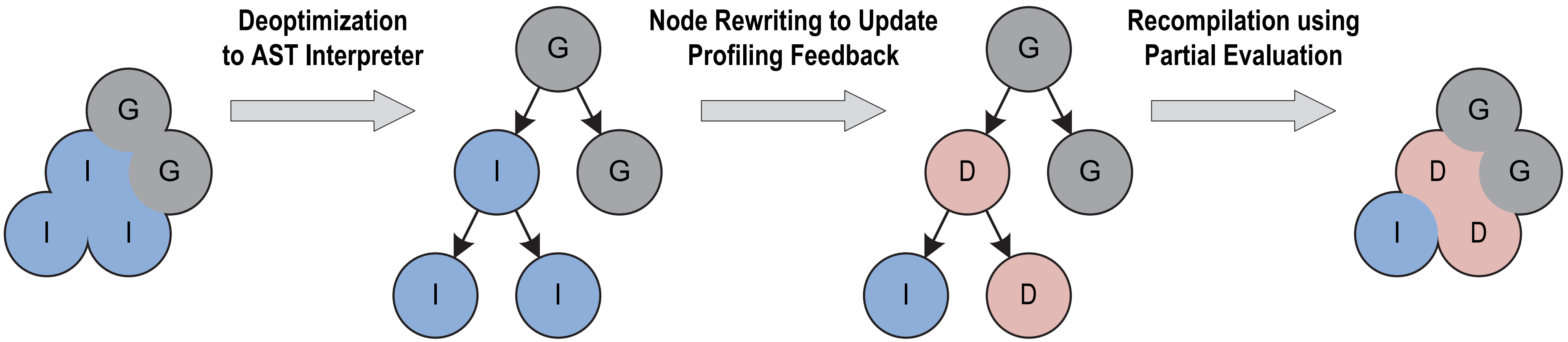
Speculate and Optimize…



U
U
U
U
U
AST Interpreter Uninitialized Nodes
Node Rewriting for Profiling Feedback
Node Transitions
U
Uninitialized
Integer
I
S
D
String
Double
G
Generic
G
I
G
I
I
AST Interpreter Rewritten Nodes
Compilation using Partial Evaluation
G
I
G
I
I
Compiled Code

Technical Approach

… and Deoptimize and Reoptimize!


G
I
G
I
I
Deoptimization to AST Interpreter
G
I
G
I
I
Node Rewriting to Update Profiling Feedback
G
D
G
I
D
Recompilation using Partial Evaluation
G
D
G
I
D

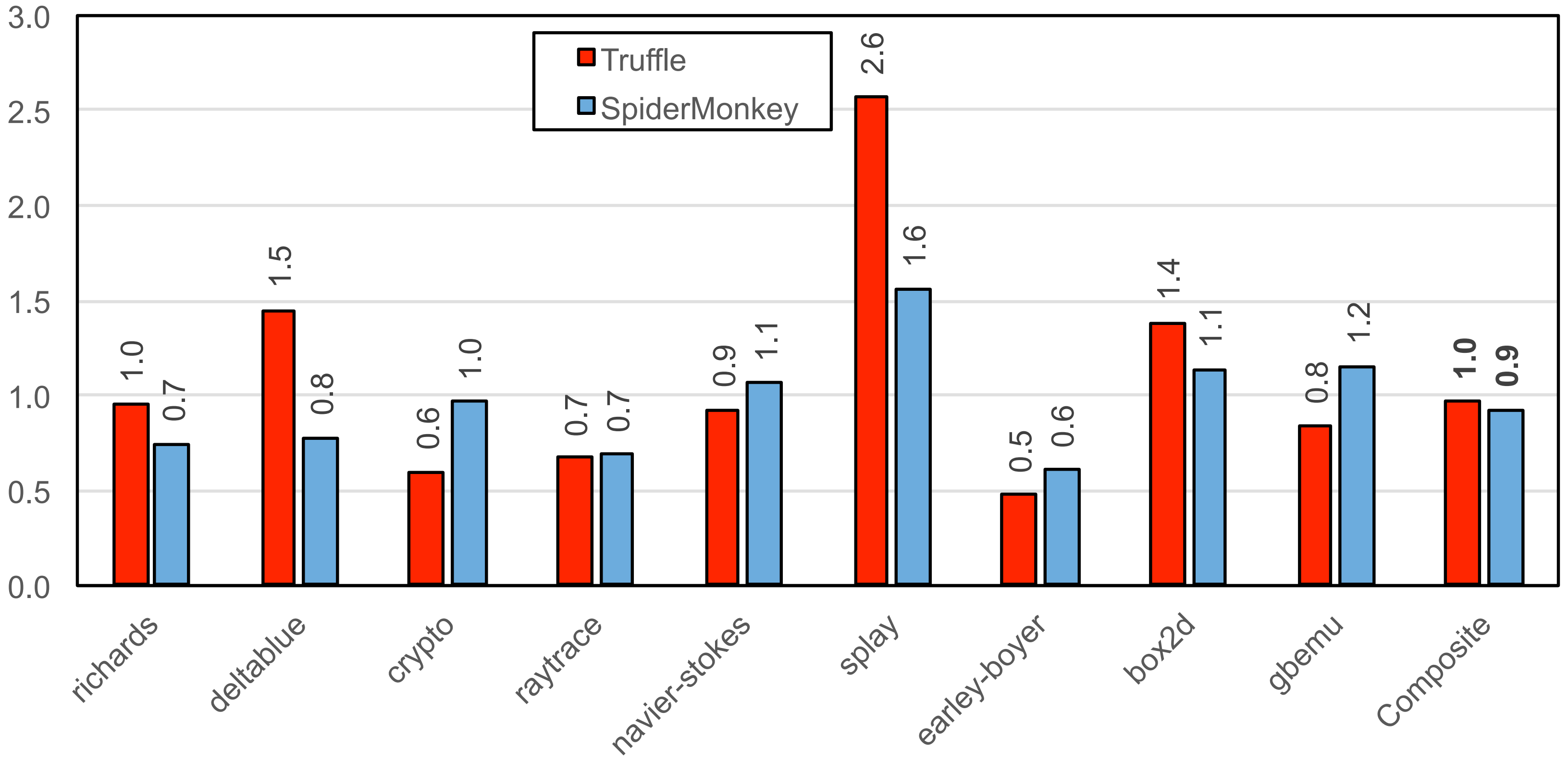
Technical Approach

Three main parts for driving partial evaluation

- Limit partial evaluation expansion
  - Annotation `@SlowPath` on a method stops the inclusion of a method in the expansion.
- Dynamic speculation
  - Call to `CompilerDirectives.transferToInterpreter()` advises the partial evaluator to stop and place a deoptimization exit.
- Global speculation
  - Assumption objects can be used for global speculations about the system state. Checking the assumption in compiled code poses no runtime overhead.

Peak Performance: JavaScript

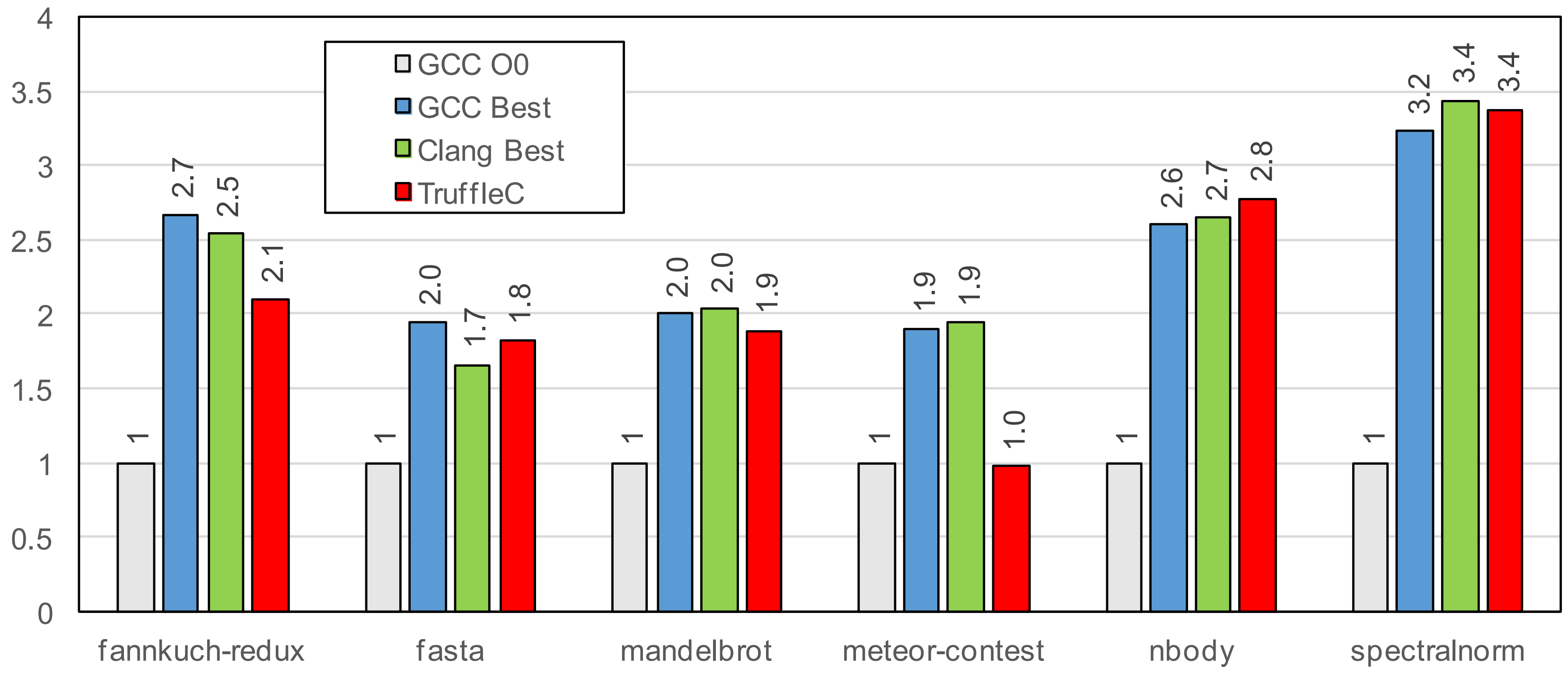
Speedup relative to V8



Selection of benchmarks from Google's Octane benchmark suite v1.0
latest versions of V8, Truffle, and SpiderMonkey as of December 2013

Peak Performance: C

Speedup relative to GCC O0



Grimmer, Rigger, Schatz, Stadler, Mössenböck: *TruffleC: Dynamic Execution of C on the Java Virtual Machine*; to be submitted

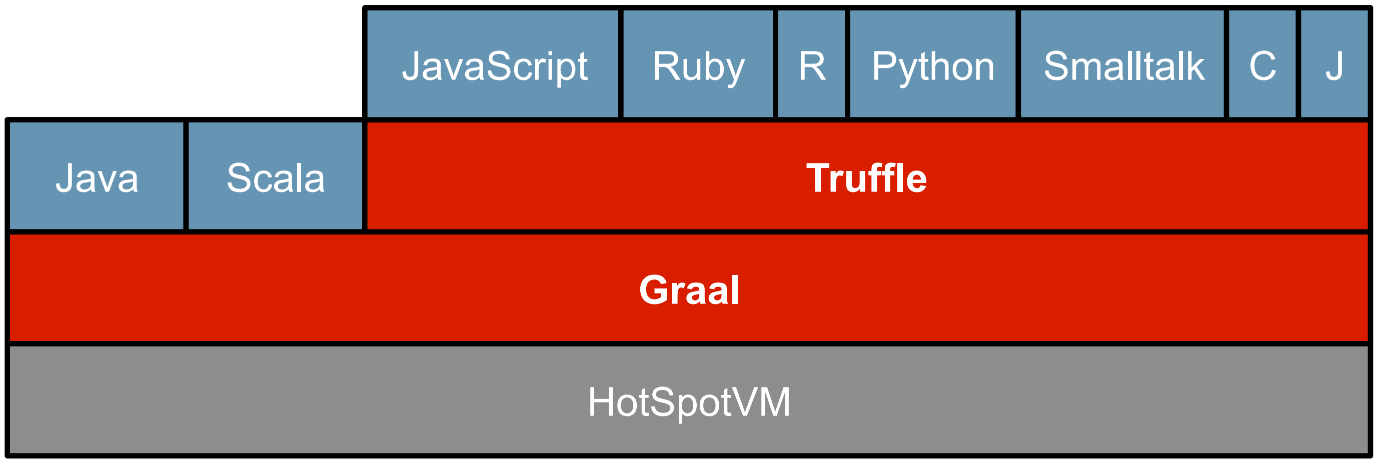
Agenda

- Graal
- Truffle
- **Community**
- Q&A

Graal OpenJDK Project

http://openjdk.java.net/projects/graal/

- Development of Graal/Truffle core artifacts and APIs
- Highly active: 30+ contributors over last 12 months
- Highly modular: 80+ individual modules



Research Areas

Language Implementation	Experimentation with new language features, new languages, new execution models	**Truffle Interpreters**
General Language Research	Language-independent instrumentation, cross-language research, automatic partial evaluation experiments	**Truffle**
Compiler Construction	Core compiler construction research, heterogenuous computing, advanced architectures and backends	**Graal**

Graal/Truffle Related Research Projects (1)

- TruffleRuby
  - Development in the JRuby repository (lead Chris Seaton).
  - https://github.com/jruby/jruby
  - http://blog.jruby.org/2014/01/truffle_graal_high_performance_backend/
- FastR
  - Joint effort of a group from Purdue University (Prof. Jan Vitek) and a team at Oracle Labs (lead Michael Haupt).
  - https://bitbucket.org/allr/fastr
- ZipPy
  - Development by a group from University of California, Irvine (Prof. Michael Franz).
  - https://bitbucket.org/ssllab/zippy
- TruffleSOM
  - Development by Stefan Marr at: https://github.com/smarr/

Graal/Truffle Related Research Projects (2)

- C and Language Interoperability
  - Experiment by students at JKU Linz (Matthias Grimmer and Manuel Rigger).
- JavaScript
  - Effort done by the core Graal/Truffle team.
- Debugging
  - Effort by Micheal van de Vanter from Oracle Labs.
- SubstrateVM
  - Team at Oracle Labs led by Christian Wimmer is developing an alternative host runtime.
- Graal IR Instrumentation
  - Research by Yudi Zheng (USI Lugano) on instrumenting Graal IR.
- GPU Offload
  - Research by Christopher Dubach et al. from the University of Edinburgh.
  - Graal is the compiler of choice for Project Sumatra (HSAIL/PTX offload).

ORACLE

Your Language or Compiler Extension?

http://openjdk.java.net/projects/graal/

graal-dev@openjdk.java.net

```
$ hg clone http://hg.openjdk.java.net/graal/graal
$ cd graal
$ ./mx --vm server build
$ ./mx ideinit
$ ./mx --vm server unittest SumTest
```

- Graal Resources

  https://wiki.openjdk.java.net/display/Graal/Main

- Truffle API License: GPLv2 with Classpath Exception
- Graal License: GPLv2

Acknowledgements

Oracle Labs
Danilo Ansaloni
Daniele Bonetta
Laurent Daynès
Erik Eckstein
Michael Haupt
Peter Kessler
David Leibs
Mick Jordan
Tom Rodriguez
Roland Schatz
Chris Seaton
Doug Simon
Lukas Stadler
Michael Van De Vanter
Adam Welc
Christian Wimmer
Christian Wirth
Mario Wolczko
Thomas Würthinger
Laura Hill

Interns
Miguel Garcia Gutierrez
Shams Imam
Stephen Kell
Gregor Richards
Rifat Shariyar

JKU Linz
Prof. Hanspeter Mössenböck
Stefan Anzinger
Gilles Duboscq
Josef Eisl
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Humer
Christian Huber
David Leopoldseder
Manuel Rigger
Georg Schmid
Bernhard Urban
Andreas Wöß

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

And many more…

http://openjdk.java.net/projects/graal/

graal-dev@openjdk.java.net

@thomaswue

Q/A

Hardware and Software

ORACLE®

Engineered to Work Together

ORACLE®