ORACLE

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

Thomas Wuerthinger
Oracle Labs
@thomaswue

24-April-2014,
Keynote at MODULARITY in Lugano

Disclaimer

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.

Agenda

- Graal
- Truffle
- Community
- Q&A

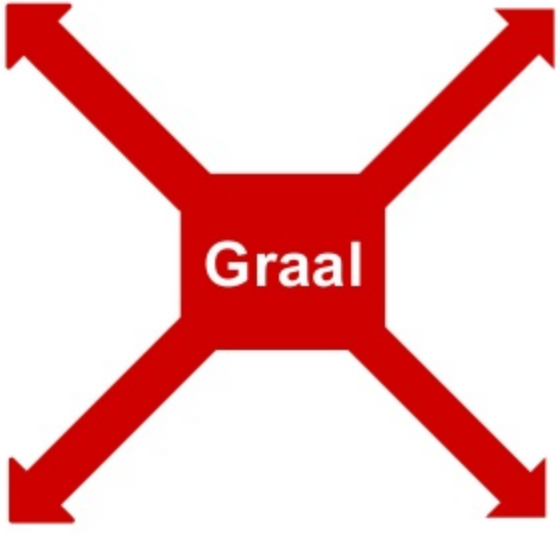
Dimensions of Extensibility

X86, SPARC, HSAIL, PTX, ...

Architectures

HotSpotVM, SubstrateVM, ...

Host Runtimes



Compilation Policy

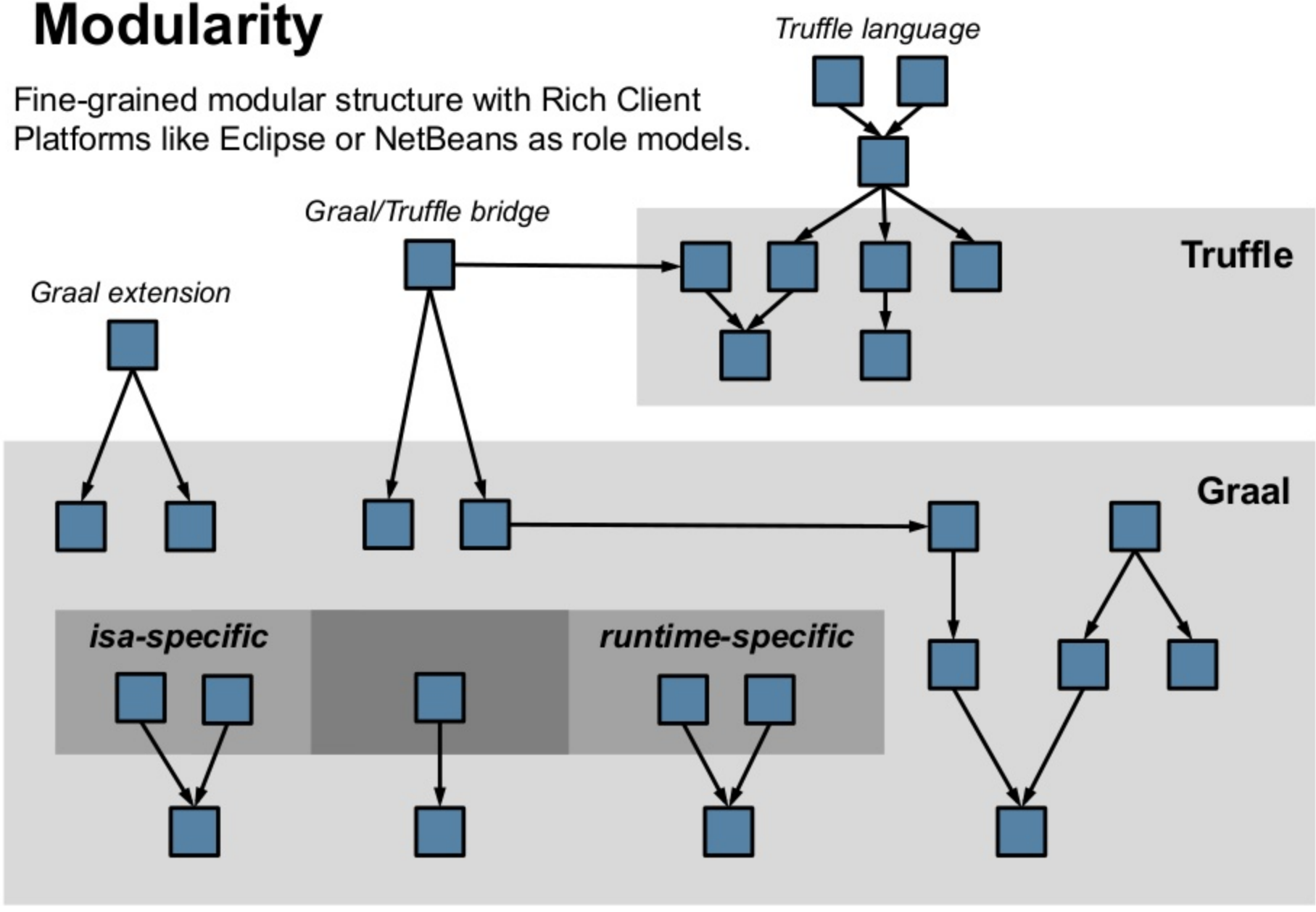
Baseline, Mid-tier, Optimizing, Vectorizing, ...

Programming Languages

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

Modularity

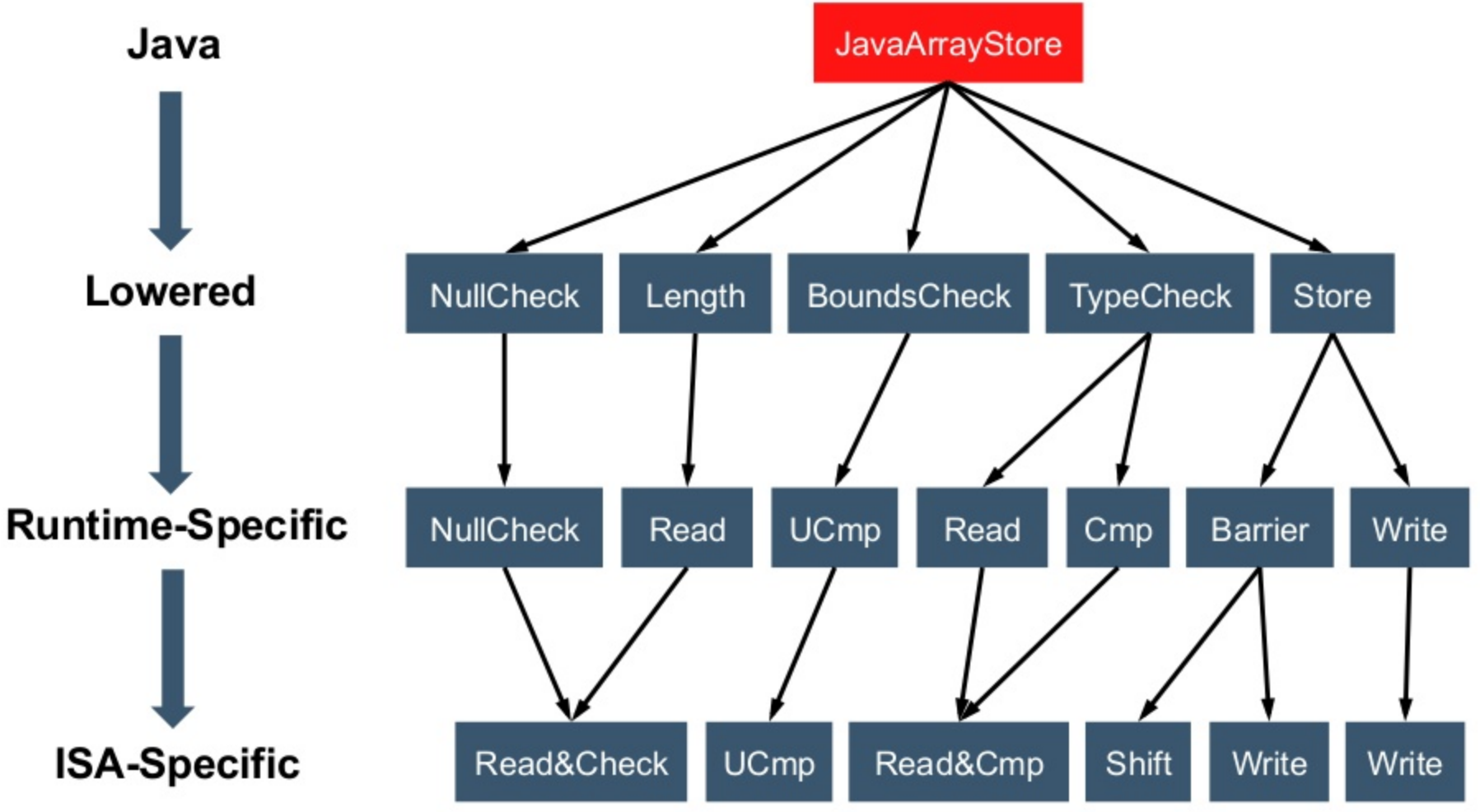
Fine-grained modular structure with Rich Client Platforms like Eclipse or NetBeans as role models.



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



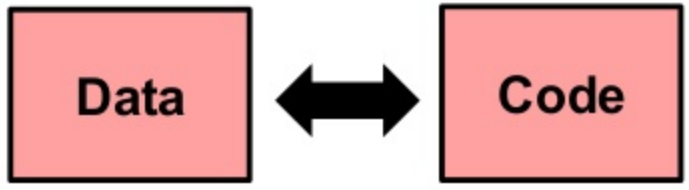
Snippets for Graph Construction

Manual construction:

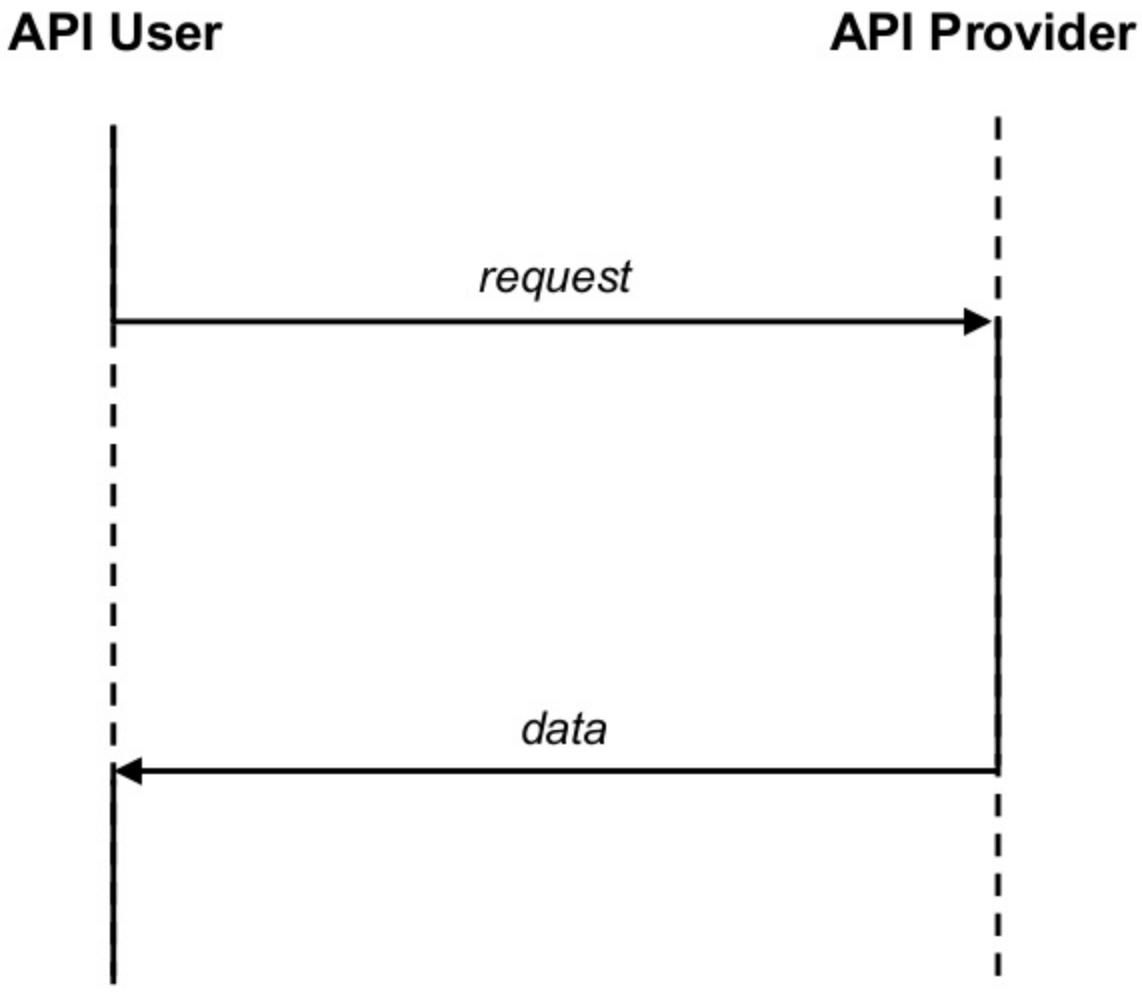
```
Node max(ValueNode a, ValueNode b) {
    IfNode ifNode = new IfNode(new IntegerLessThanNode(a, b));
    ifNode.trueSuccessor().setNext(new ReturnNode(a));
    ifNode.falseSuccessor().setNext(new ReturnNode(b));
    return ifNode;
}
```

Expression as snippet:

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



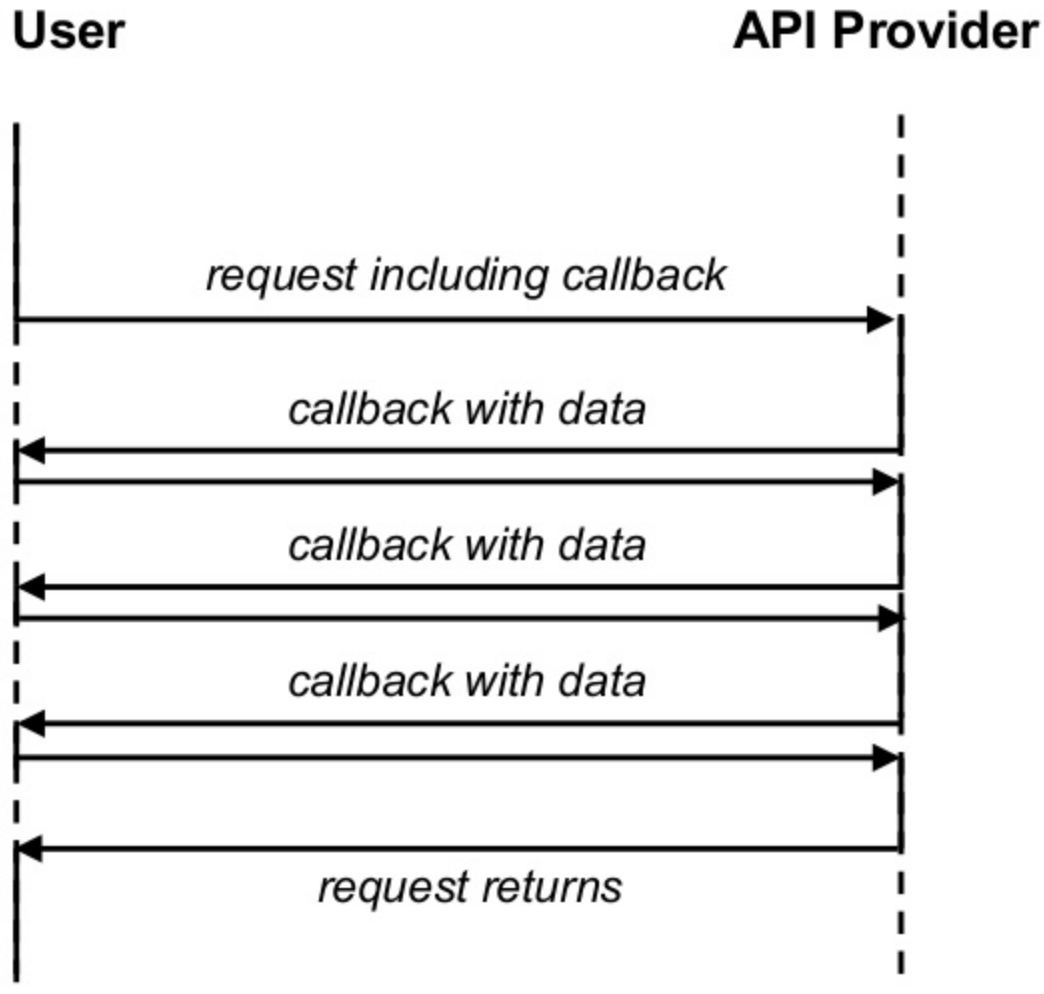
Simple API



✓ Can capture statically

x Limited flexibility

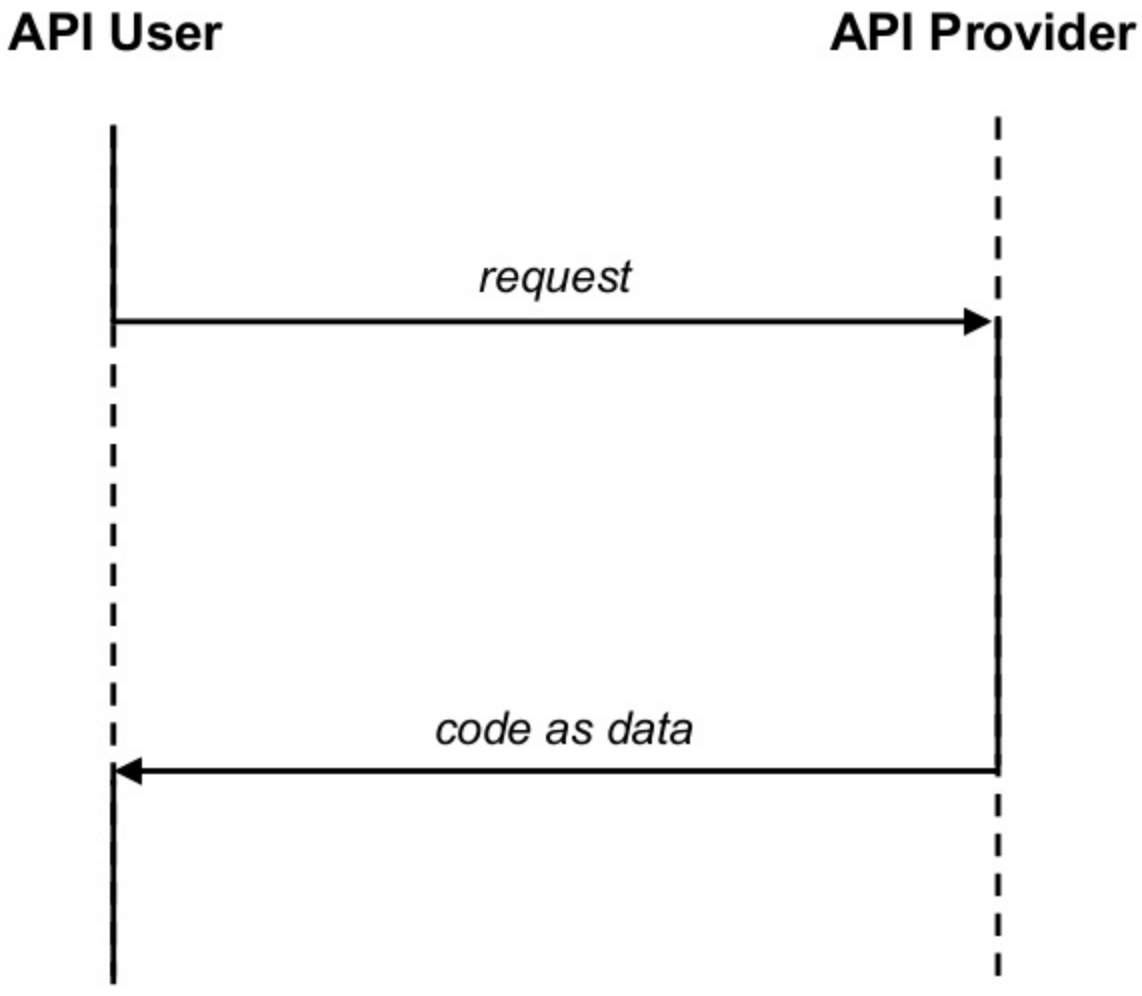
Callback API



✓ High flexibility

x Cannot capture statically

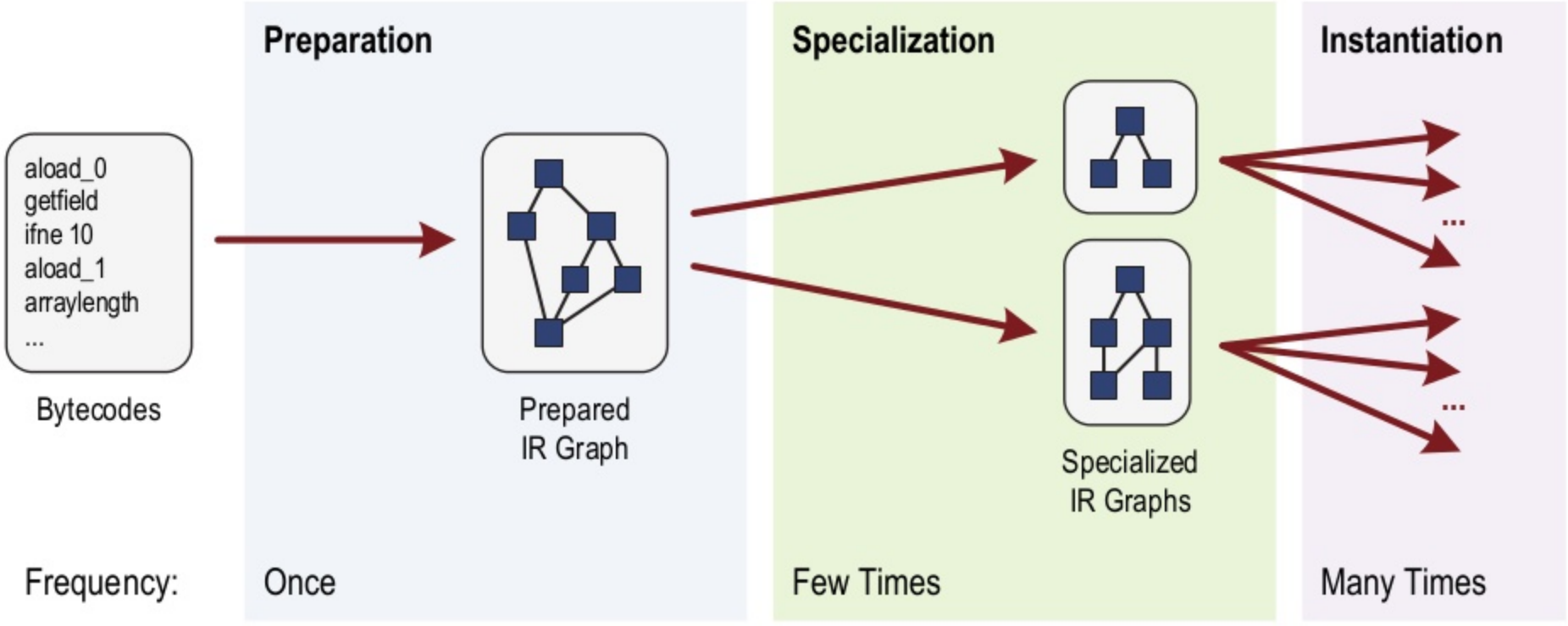
Snippet API



✓ 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;
}
```

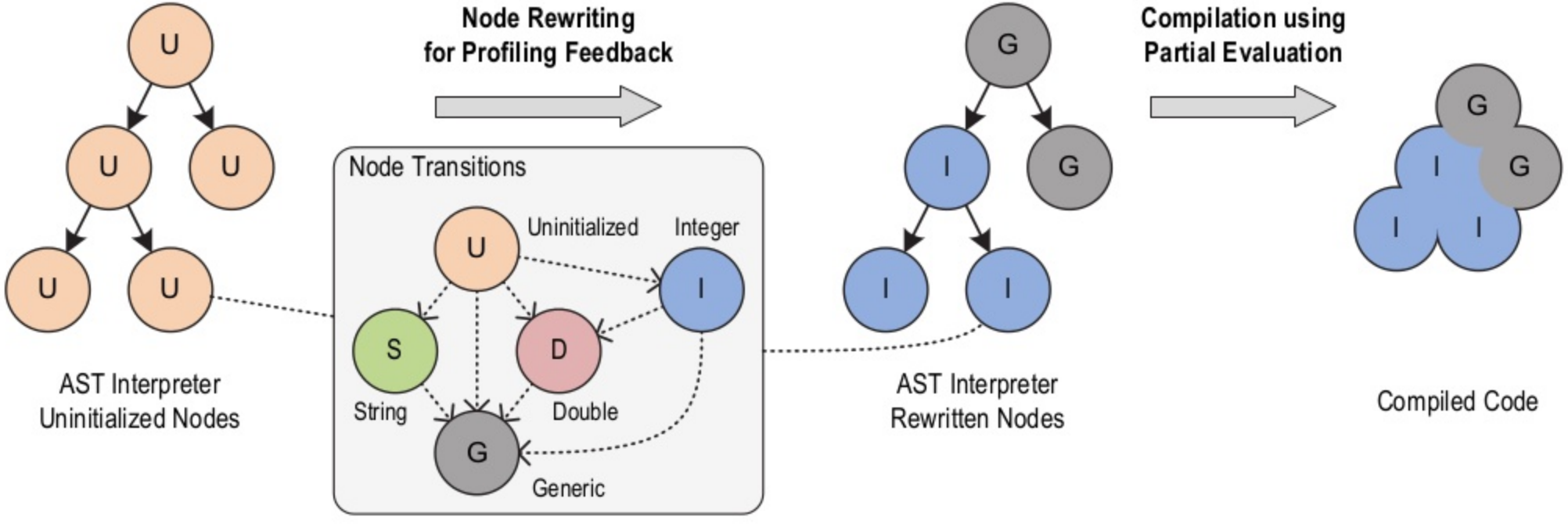
Agenda

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

ORACLE

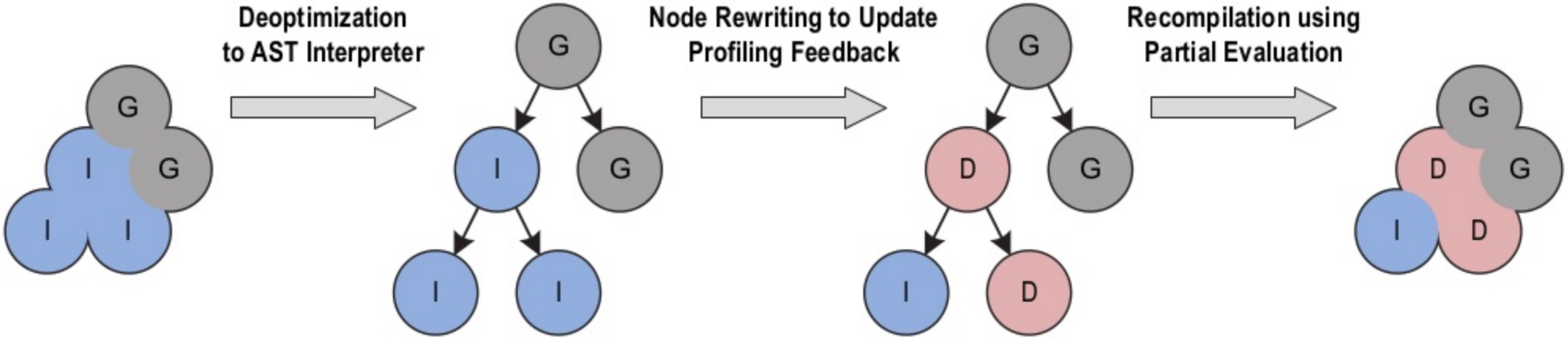
Technical Approach

Speculate and Optimize…



Technical Approach

… and Deoptimize and Reoptimize!



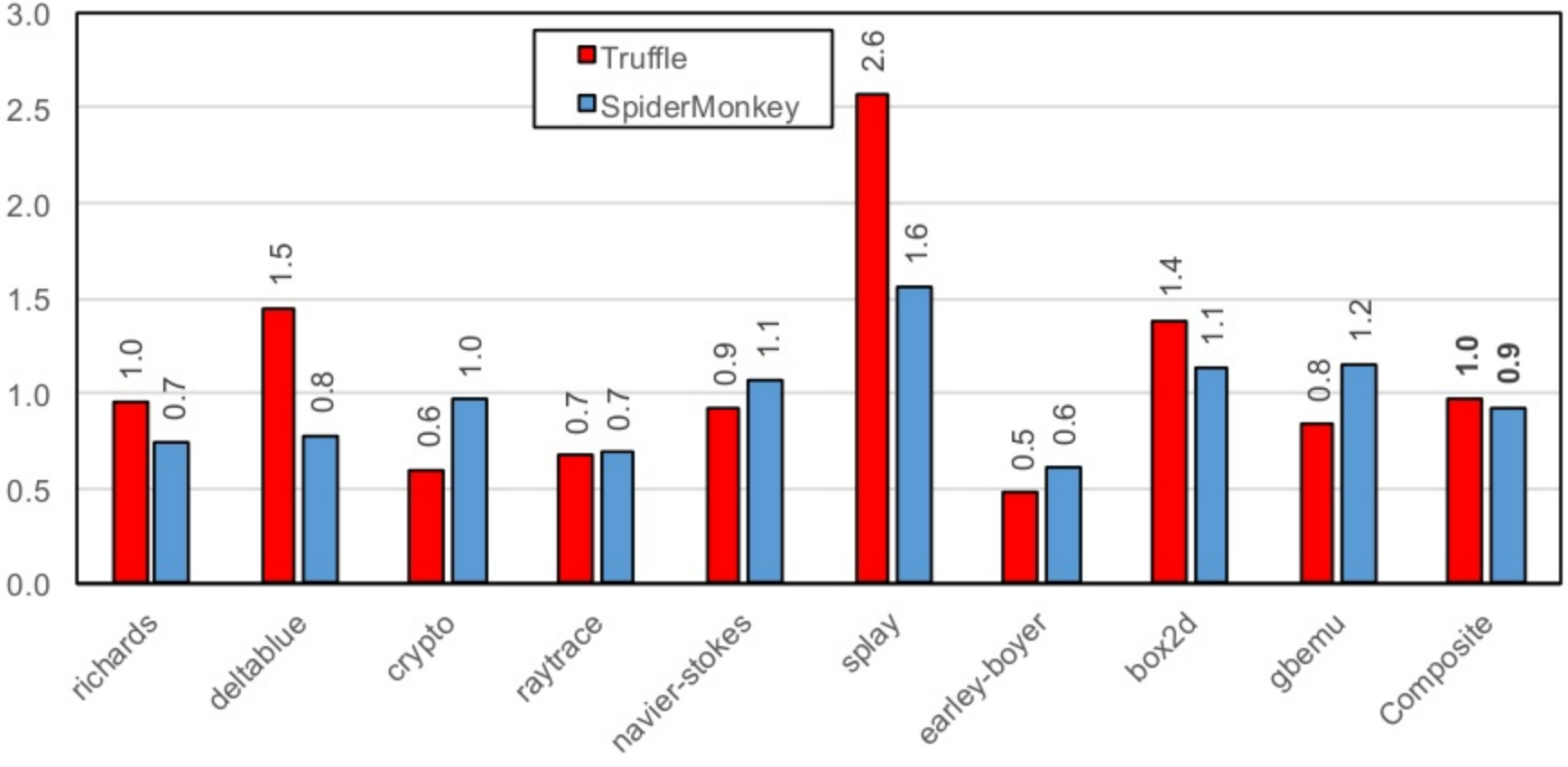
Technical Approach

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

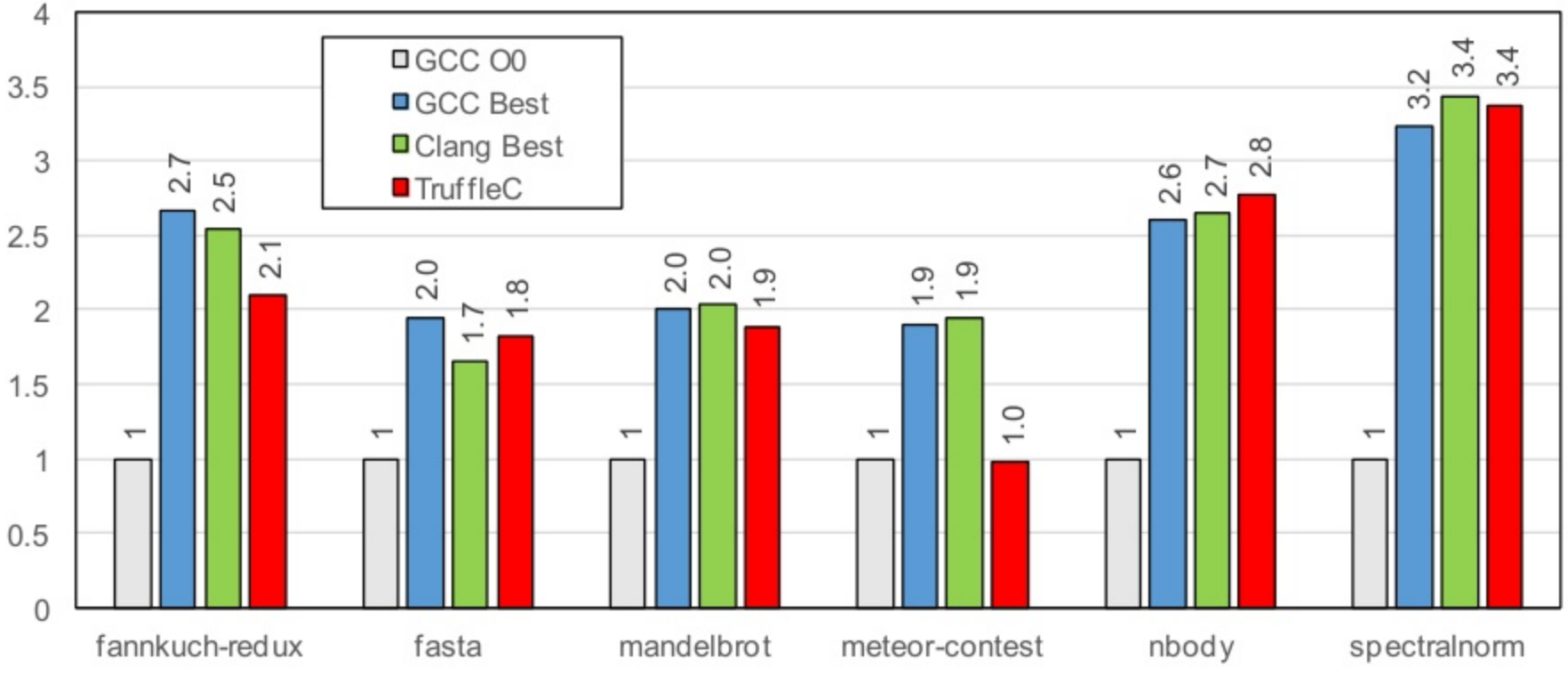
Speedup relative to V8



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