



Graal Extensibility

JVM Language Summit 2011 Workshop



Introduction

- Lukas Stadler
 - Working at Johannes Kepler University Linz, Austria
 - Working with Thomas and the Maxine team



Basic Graal Structure

- Based on the Maxine CRI
- Modifications to native HotSpot code (CRI implementation)
- Graal and Maxine classes added to boot classpath
- Bootstrapping phase



Simple Example

- Command line: `-graal`
- Simple IR
- Supports `IdealGraphVisualizer`



Example: Inlining Guide

- Registers as a service in META-INF
- Influence inlining decisions: NEVER, LESS, MORE, ALWAYS
- Context information: current inlining depth, caller, callee, ...



Example: Intrinsicification

- Registers as a service in META-INF
- Can replace invocation with arbitrary compiler graph
- Context information: caller, callee, parameters, ...
- In this example:
 - New node type: SafeAddNode (addition with overflow checking)
 - Deoptimization on overflow



Example: Deoptimize

- Registers as a service in META-INF
- Intrinsicification example still needs to handle all cases in bytecode
- Solution: deoptimize to interpreter loop (written in Java)
- Uses escape analysis code to box current state (vars, stack, ...)



Example: Optimizations

- Registers as a service in META-INF
- Can do arbitrary changes to the IR graph
- In this example:
 - Look at all SafeAddNodes
 - Replace with normal AddNode if they:
 - Are a loop counter and
 - Cannot overflow



Questions & Discussion ...

- Does this level of extensibility make sense?
- What are the pain points language implementors currently face?
- How low-level should you get?
- ... ?