

**ORACLE®**

# Compilation at Runtime

**IN4303 Compiler Construction**

Guido Wachsmuth

Principal Member of Technical Staff  
Oracle Labs Zurich

December 12, 2017

This talk includes slides from talks by Thomas Wuerthinger and Chris Seaton.

# Agenda

Oracle Labs

Graal Compiler

- dynamic compilation
- intermediate representation
- static compilation

Truffle Framework

- partial evaluation
- self-Optimizing AST interpreters
- polyglot AST interpreters

Software Language Engineering Projects

# Safe Harbor Statement

The following 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.

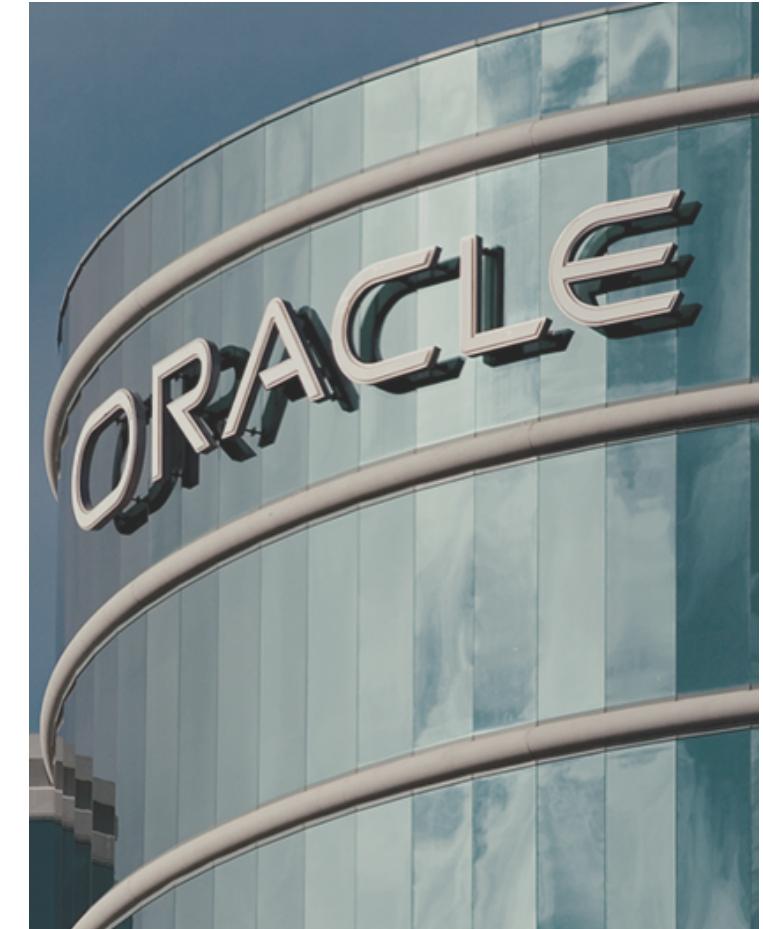
# Oracle Corporation

## Scale

- US\$37.7B total GAAP revenue in FY 2017
- 430K customers in 175 countries
- 25K partners
- More than 138K employees
- 16K support and services specialists
- 19K implementation consultants

## Innovation

- More than US\$45B in R&D over the last 10 years
- 40K developers and engineers
- 3.5M students supported annually in 120 countries
- More than 17K patents
- 5M registered members of the Oracle Developer Community
- 469 independent user communities in 97 countries



# Oracle Leadership

#1

## Technologies

- Application Server
- Database
- Database on Linux and Unix
- Data warehouse
- Deployment-centric applications platform
- Embedded database
- Engineered systems
- Middleware

## Applications

- Business analytics
- Database management
- Enterprise performance management
- Lead management
- Marketing automation
- Structural data management
- Supply chain execution
- Talent management

# Oracle Customers



AEROSPACE AND DEFENSE  
10 out of 10 top companies



AUTOMOTIVE  
20 out of 20 top companies



CLOUD  
10 out of 10 top SaaS providers



CONSUMER GOODS  
9 out of 10 top companies



EDUCATION AND RESEARCH  
20 out of 20 top universities



ENG. AND CONSTRUCTION  
9 out of 10 top companies



FINANCIAL SERVICES  
20 out of 20 top banks



HIGH TECHNOLOGY  
20 out of 20 top companies



INSURANCE  
20 out of 20 top insurers



MANUFACTURING  
20 out of 20 top companies



MEDICAL DEVICES  
20 out of 20 top companies



OIL AND GAS  
20 out of 20 top companies



PHARMACEUTICALS  
20 out of 20 top companies



PUBLIC SECTOR  
20 out of 20 top governments



RETAIL  
20 out of 20 top companies



SUPPLY CHAINS  
20 out of 20 top companies



TELECOMMUNICATION  
20 out of 20 top companies

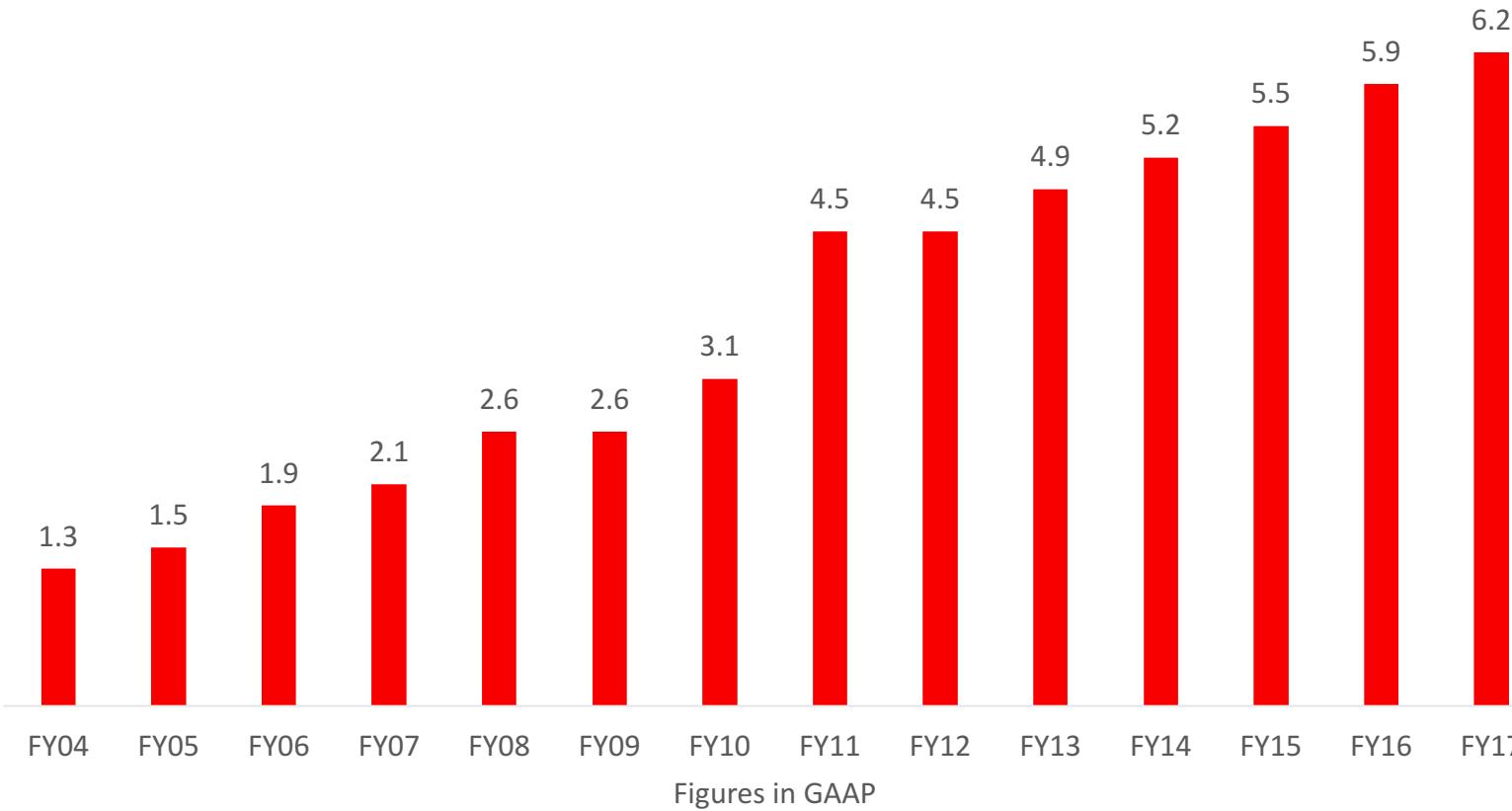


TRAVEL AND TRANSPORT  
20 out of 20 top airlines



UTILITIES  
10 out of 10 top companies

# Investment in Research and Development



**MORE THAN  
\$51 BILLIONS  
SINCE 2004**



“The Mission of Oracle Labs is straightforward:

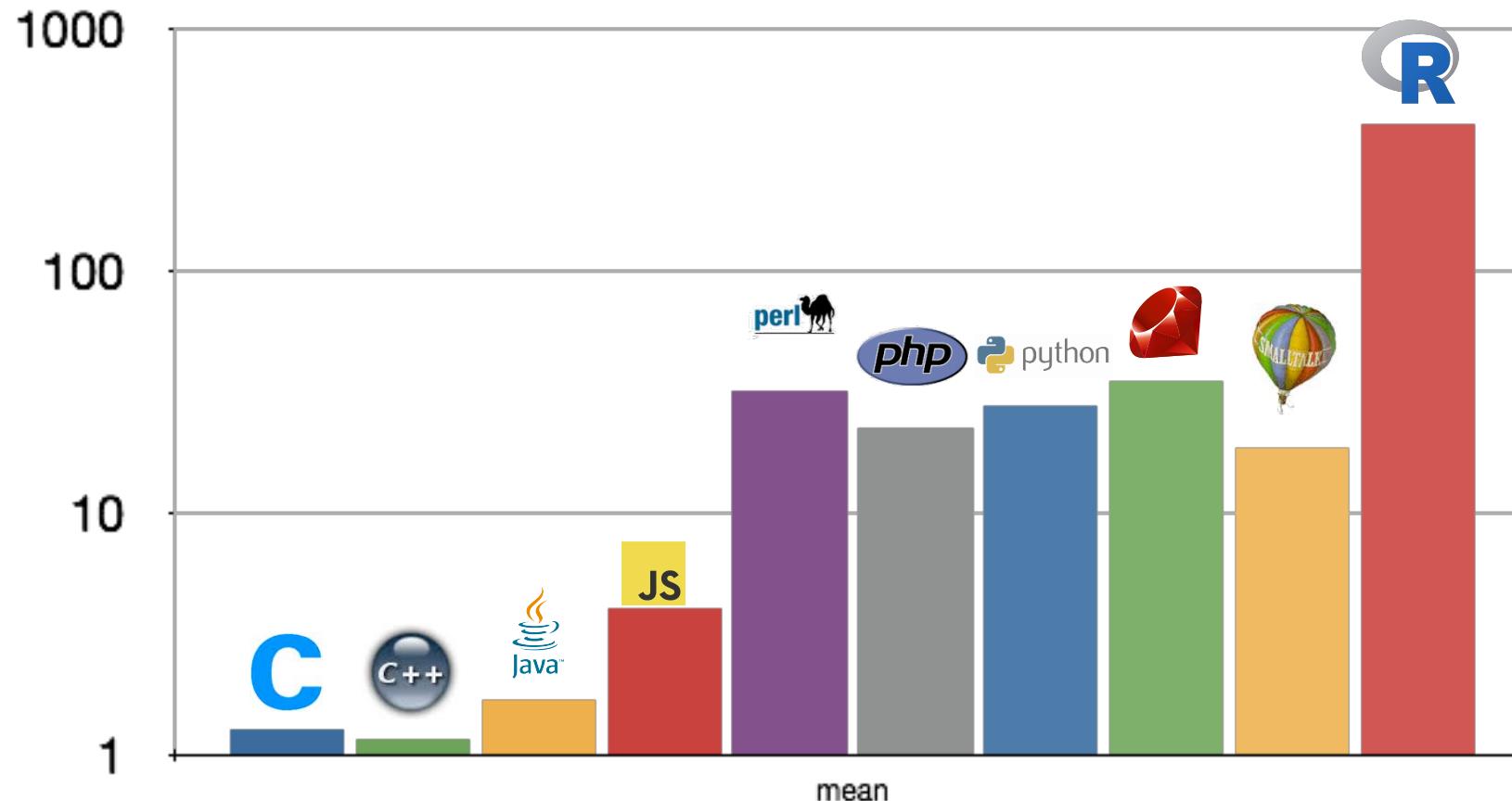
**Identify, explore, and transfer new technologies  
that have the potential to substantially improve  
Oracle's business.”**

- Oracle Labs mission statement

# Dynamic Compilation

compiling interpreted code to machine code

# The Computer Language Benchmarks Game



# Static versus Dynamic Compilation

## Static Compilation (ahead-of-time)

- happens **before** program is running
- can include **profiling feedback** from sample application runs

## Dynamic Compilation (just-in-time)

- happens **while** program is running
- base line execution gathers **profiling feedback**
- **speculation:** optimization, de-optimization, re-optimization cycles
- **switch** between tiers (execution modes)

# Static versus Dynamic Compilation

## Static Compilation (ahead-of-time)

- **fast start-up:** compilation and profiling are not part of application execution time
- **predictable performance:** only source program affects generated machine code

## Dynamic Compilation (just-in-time)

- can exploit exact target platform properties
- profiling feedback captures part of the application behavior
- can use assumptions about the current state of the system (e.g., loaded classes)

# Graal Compiler

a Java JIT compiler written in Java

The following slides are heavily inspired by a talk on Graal  
by Chris Seaton. You can watch his talk or read his  
excellent write-up on his blog:

- <http://chrisseaton.com/rubytruffle/jokerconf17/>

```
class Demo {  
    public static void main(String[] args) {  
        while (true) {  
            workload(14, 2);  
        }  
    }  
  
    private static int workload(int a, int b) {  
        return a + b;  
    }  
}
```

```
$ javac Demo.java
$ java \
  -XX:+PrintCompilation \
  -XX:CompileOnly=Demo::workload \
  Demo
...
      113      1          3           Demo::workload (4 bytes)
...
```

The screenshot shows a terminal window with a dark theme. The title bar reads "divnode.cpp — ~/Documents/jokerconf17/demo/openjdk". The left pane is labeled "Project" and lists several source files: convertnode.hpp, countbitsnode.hpp, countbitsnode.hpp, divnode.cpp, divnode.hpp, doCall.cpp, domgraph.cpp, escape.cpp, escape.hpp, gcm.cpp, generateOptoStub.cpp, graphKit.cpp, graphKit.hpp, idealGraphPrinter.cpp, idealGraphPrinter.hpp, idealKit.cpp, idealKit.hpp, ifg.cpp, ifnode.cpp, indexSet.cpp, indexSet.hpp, intrinsicnode.hpp, intrinsicnode.hpp, lcm.cpp, and ... . The file "divnode.cpp" is selected and highlighted in blue. The main pane displays the content of "divnode.cpp", specifically the "Idealize" method of the "DivLNode" class. The code is annotated with comments explaining its behavior, such as handling division by zero and power-of-2 shifts.

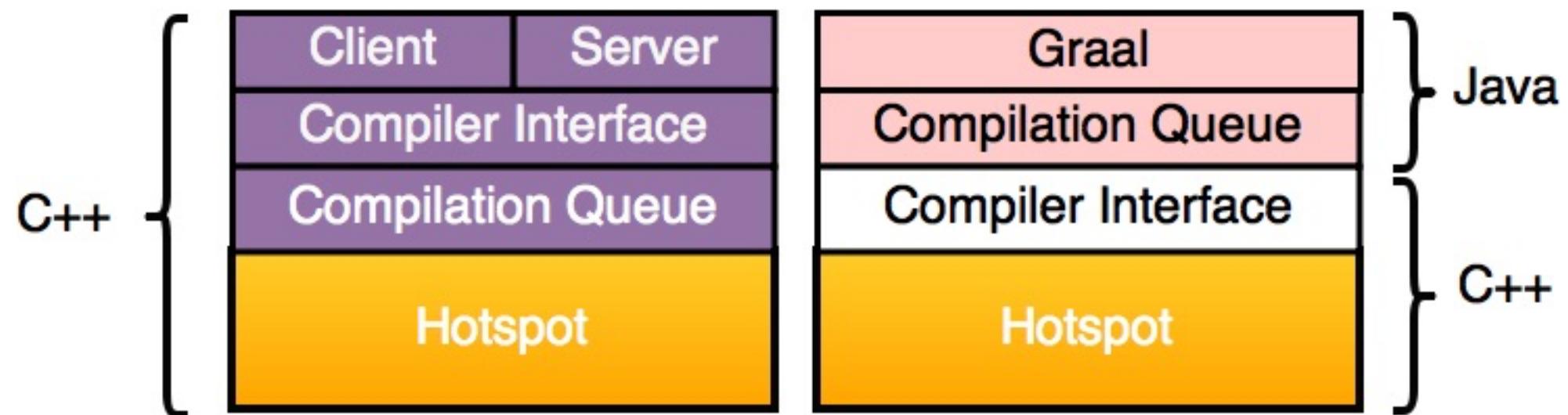
```
568
569 //-----Idealize-----
570 // Dividing by a power of 2 is a shift.
571 Node *DivLNode::Ideal( PhaseGVN *phase, bool can_reshape ) {
572     if (in(0) && remove_dead_region(phase, can_reshape)) return this;
573     // Don't bother trying to transform a dead node
574     if( in(0) && in(0)->is_top() ) return NULL;
575
576     const Type *t = phase->type( in(2) );
577     if( t == TypeLong::ONE )           // Identity?
578         return NULL;                  // Skip it
579
580     const TypeLong *tl = t->isa_long();
581     if( !tl ) return NULL;
582
583     // Check for useless control input
584     // Check for excluding div-zero case
585     if (in(0) && (tl->hi < 0 || tl->lo > 0)) {
586         set_req(0, NULL);           // Yank control input
587         return this;
588     }
589
590     if( !tl->is_con() ) return NULL;
591     jlong l = tl->get_con();      // Get divisor
592
593     if (l == 0) return NULL;        // Dividing by zero constant does not idealize
594
595     // Dividing by MINLONG does not optimize as a power-of-2 shift.
596     if( l == min_jlong ) return NULL;
597
598     return transform_long_divide( phase, in(1), l );
599 }
```

hotspot/src/share/vm/opto/divnode.cpp 571:60

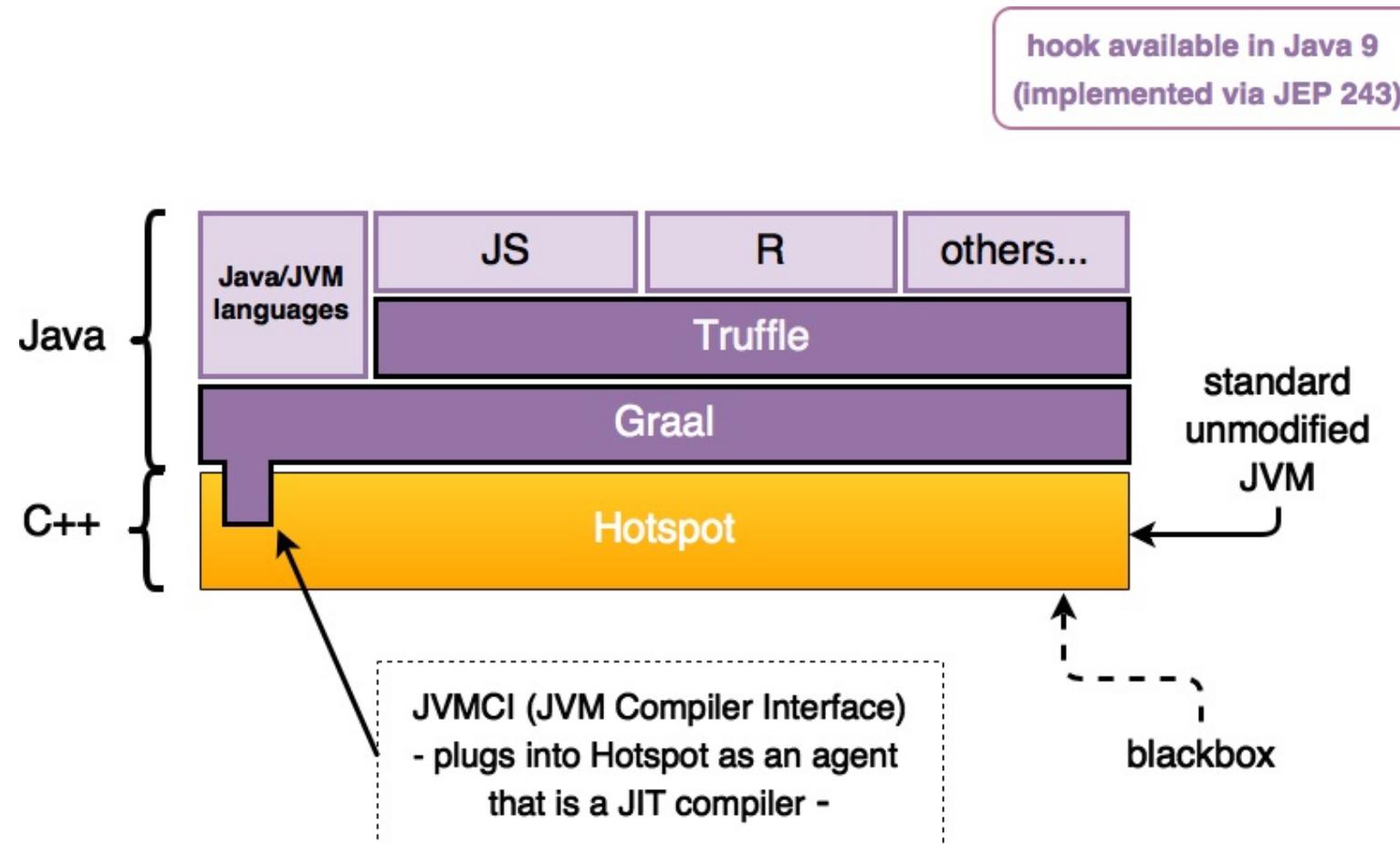
LF UTF-8 C++ jdk9/jdk9 ↓ ↑ 0 files

# Graal VM

## HotspotVM versus GraalVM



# Graal Compiler



```
$ java \
--module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/..... \
--upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
-XX:+UnlockExperimentalVMOptions \
-XX:+EnableJVMCI \
-XX:+UseJVMCICompiler \
-XX:-TieredCompilation \
-XX:+PrintCompilation \
-XX:CompileOnly=Demo::workload \
Demo
...
      583   25          Demo::workload (4 bytes)
...
```

```
interface JVMMCCompiler {  
    byte[] compileMethod(byte[] bytecode);  
}
```

```
interface JVMMCCompiler {
    void compileMethod(CompilationRequest request);
}

interface CompilationRequest {
    JavaMethod getMethod();
}

interface JavaMethod {
    byte[] getCode();
    int getMaxLocals();
    int getMaxStackSize();
    ProfilingInfo getProfilingInfo();
    ...
}
```

```
HotSpot.installCode(targetCode);
```

graal - jdk.vm.ci.runtime.JVMCICompiler - Eclipse

The screenshot shows the Eclipse IDE interface with the title bar "graal - jdk.vm.ci.runtime.JVMCICompiler - Eclipse". The toolbar contains various icons for file operations, code navigation, and project management. The main editor window displays the Java code for the "JVMCICompiler" interface. The code is as follows:

```
2+ * Copyright (c) 2015, Oracle and/or its affiliates. All rights reserved.|
23 package jdk.vm.ci.runtime;
24
25 import jdk.vm.ci.code.CompilationRequest;
26 import jdk.vm.ci.code.CompilationRequestResult;
27
28 public interface JVMCICompiler {
29     int INVOCATION_ENTRY_BCI = -1;
30
31     /**
32      * Services a compilation request. This object should compile the method to machine code and
33      * install it in the code cache if the compilation is successful.
34     */
35     CompilationRequestResult compileMethod(CompilationRequest request);
36 }
37 |
```

The code is color-coded, with keywords in purple and comments in blue. Line numbers are visible on the left. The status bar at the bottom shows "Read-Only", "Smart Insert", and "37 : 1".

The screenshot shows the Eclipse IDE interface with the title bar "graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse". The toolbar contains various icons for file operations, search, and navigation. The code editor window displays the Java file "HotSpotGraalCompiler.java" with line numbers 78 to 107. The code implements the GraalJVMCICompiler interface, managing JvmciRuntime, GraalRuntime, CompilationCounters, and a BootstrapWatchDog. It also provides methods for getting DebugHandlersFactories and GraalRuntime, and for compiling a method.

```
78 public class HotSpotGraalCompiler implements GraalJVMCICompiler {  
79  
80     private final HotSpotJVMCIRuntimeProvider jvmciRuntime;  
81     private final HotSpotGraalRuntimeProvider graalRuntime;  
82     private final CompilationCounters compilationCounters;  
83     private final BootstrapWatchDog bootstrapWatchDog;  
84     private List<DebugHandlersFactory> factories;  
85  
86     HotSpotGraalCompiler(HotSpotJVMCIRuntimeProvider jvmciRuntime, HotSpotGraalRuntimeProvider graalRuntime, OptionValues  
87         this.jvmciRuntime = jvmciRuntime;  
88         this.graalRuntime = graalRuntime;  
89         // It is sufficient to have one compilation counter object per Graal compiler object.  
90         this.compilationCounters = Options.CompilationCountLimit.getValue(options) > 0 ? new CompilationCounters(options)  
91         this.bootstrapWatchDog = graalRuntime.isBootstrapping() && !DebugOptions.BootstrapInitializeOnly.getValue(options)  
92     }  
93  
94     public List<DebugHandlersFactory> getDebugHandlersFactories() {  
95         if (factories == null) {  
96             factories = Collections.singletonList(new GraalDebugHandlersFactory(graalRuntime.getHostProviders().getSnippet  
97         }  
98         return factories;  
99     }  
100  
101    @Override  
102    public HotSpotGraalRuntimeProvider getGraalRuntime() {  
103        return graalRuntime;  
104    }  
105  
106    @Override  
107    public CompilationRequestResult compileMethod(CompilationRequest request) {  
...  
}
```

The screenshot shows the Eclipse IDE interface with the title bar "graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse". The toolbar includes standard icons for file operations, search, and navigation. The code editor displays the `HotSpotGraalCompiler.java` file, which contains Java code for a GraalVM compiler. The code handles compilation requests, checks for shutdown, and manages compilation options and watchdogs. The code is annotated with line numbers and various Java annotations like `@Override`, `@SuppressWarnings`, and `try`.

```
103     return graalRuntime;
104 }
105
106 @Override
107 public CompilationRequestResult compileMethod(CompilationRequest request) {
108     return compileMethod(request, true);
109 }
110
111 @SuppressWarnings("try")
112 CompilationRequestResult compileMethod(CompilationRequest request, boolean installAsDefault) {
113     if (graalRuntime.isShutdown()) {
114         return HotSpotCompilationRequestResult.failure(String.format("Shutdown entered"), false);
115     }
116
117     ResolvedJavaMethod method = request.getMethod();
118     OptionValues options = graalRuntime.getOptions(method);
119
120     if (graalRuntime.isBootstrapping()) {
121         if (DebugOptions.BootstrapInitializeOnly.getValue(options)) {
122             return HotSpotCompilationRequestResult.failure(String.format("Skip compilation because %s is enabled", DebugOptions.BootstrapInitializeOnly.name));
123         }
124         if (bootstrapWatchDog != null) {
125             if (bootstrapWatchDog.hitCriticalCompilationRateOrTimeout()) {
126                 // Drain the compilation queue to expedite completion of the bootstrap
127                 return HotSpotCompilationRequestResult.failure("hit critical bootstrap compilation rate or timeout", true);
128             }
129         }
130     }
131     HotSpotCompilationRequest hsRequest = (HotSpotCompilationRequest) request;
132     try (CompilationWatchDog w1 = CompilationWatchDog.watch(method, hsRequest.getId(), options);
133          BootstrapWatchDog Watch w2 = bootstrapWatchDog == null ? null : bootstrapWatchDog.watch(request);
```

```
class HotSpotGraalCompiler implements JVMMCCompiler {  
    CompilationRequestResult compileMethod(CompilationRequest request) {  
        System.err.println("Going to compile " + request.getMethod().getName());  
        ...  
    }  
}
```



```
graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse
*HotSpotGraalCompiler.java X
103     return graalRuntime;
104 }
105
106 @Override
107 public CompilationRequestResult compileMethod(CompilationRequest request) {
108     System.err.println("Going to compile " + request.getMethod().getName());
109     return compileMethod(request, true);
110 }
111
112 @SuppressWarnings("try")
113 CompilationRequestResult compileMethod(CompilationRequest request, boolean installAsDefault) {
114     if (graalRuntime.isShutdown()) {
115         return HotSpotCompilationRequestResult.failure(String.format("Shutdown entered"), false);
116     }
117
118     ResolvedJavaMethod method = request.getMethod();
119     OptionValues options = graalRuntime.getOptions(method);
120
121     if (graalRuntime.isBootstrapping()) {
122         if (DebugOptions.BootstrapInitializeOnly.getValue(options)) {
123             return HotSpotCompilationRequestResult.failure(String.format("Skip compilation because %s is enabled", DebugOptions.BootstrapInitializeOnly.name()), false);
124         }
125         if (bootstrapWatchDog != null) {
126             if (bootstrapWatchDog.hitCriticalCompilationRateOrTimeout()) {
127                 // Drain the compilation queue to expedite completion of the bootstrap
128                 return HotSpotCompilationRequestResult.failure("hit critical bootstrap compilation rate or timeout", true);
129             }
130         }
131     }
132     HotSpotCompilationRequest hsRequest = (HotSpotCompilationRequest) request;
133     +new CompilationWatchDog(w1 - CompilationWatchDog.watch(method, hsRequest.getTd(), options));
}

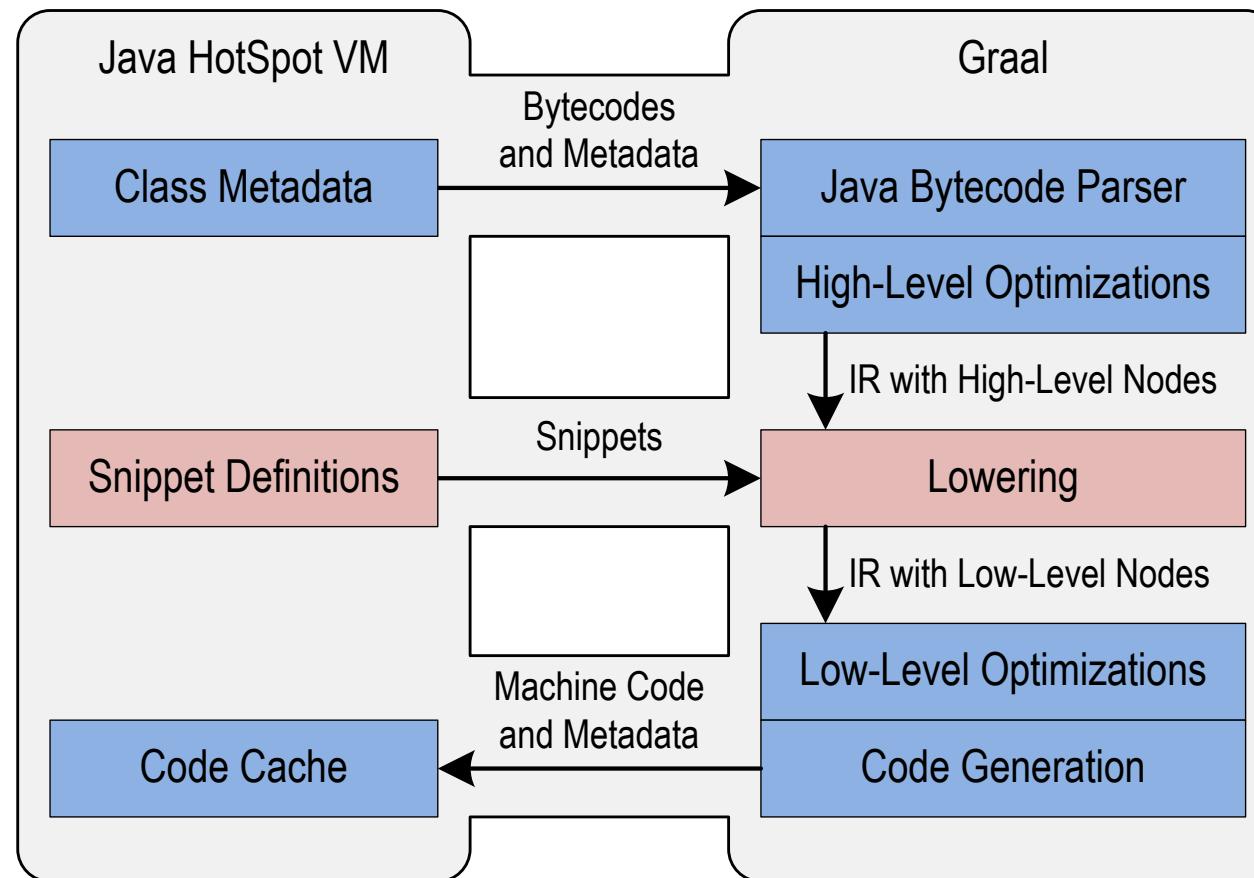
```

```
$ java \
--module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/modules/..... \
--upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
-XX:+UnlockExperimentalVMOptions \
-XX:+EnableJVMCI \
-XX:+UseJVMCICompiler \
-XX:-TieredCompilation \
-XX:CompileOnly=Demo::workload \
Demo
Going to compile workload
```

# Intermediate Representation

## data flow and control flow graphs

# Graal Compiler

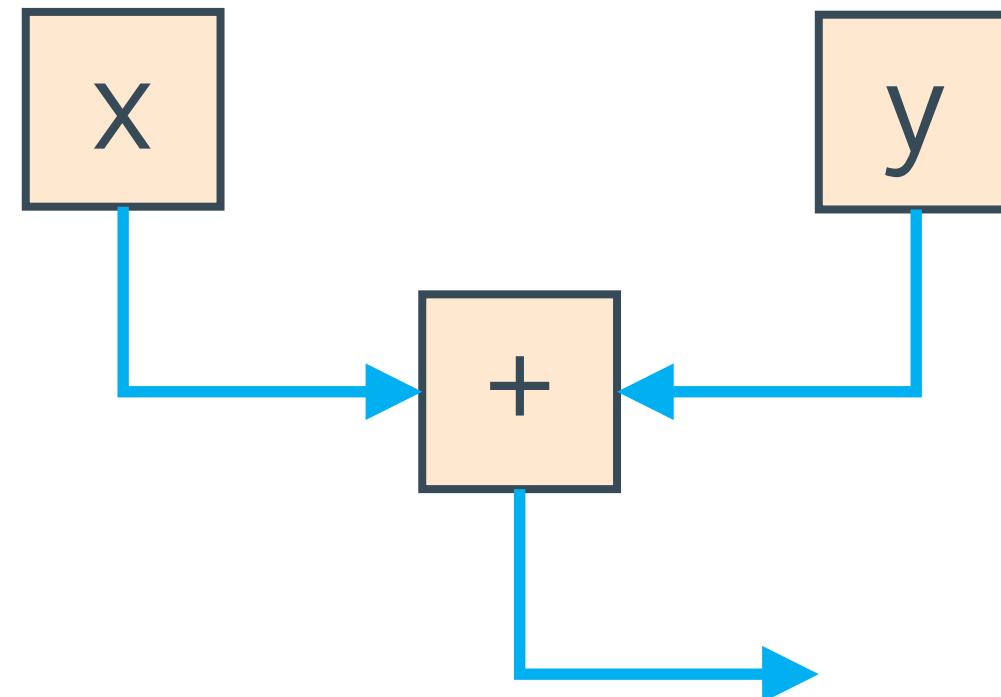


# Static Single Assignment (SSA) Form

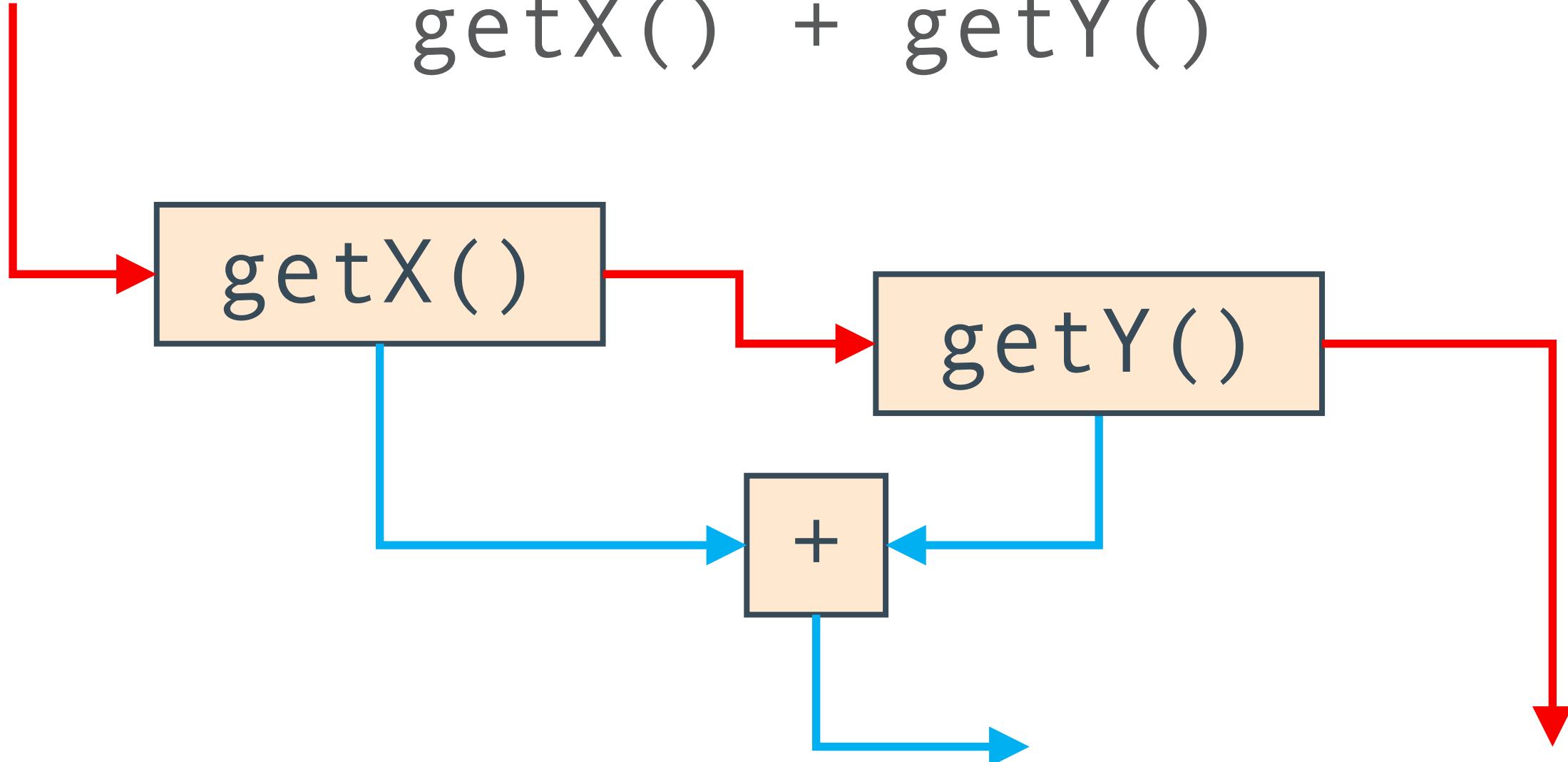
```
if (condition) {  
    x = value1 + value2;  
} else {  
    x = value2;  
}  
return x;
```

```
if (condition) {  
    x1 = value1 + value2;  
} else {  
    x2 = value2;  
}  
x3 = phi(x1, x2);  
return x3;
```

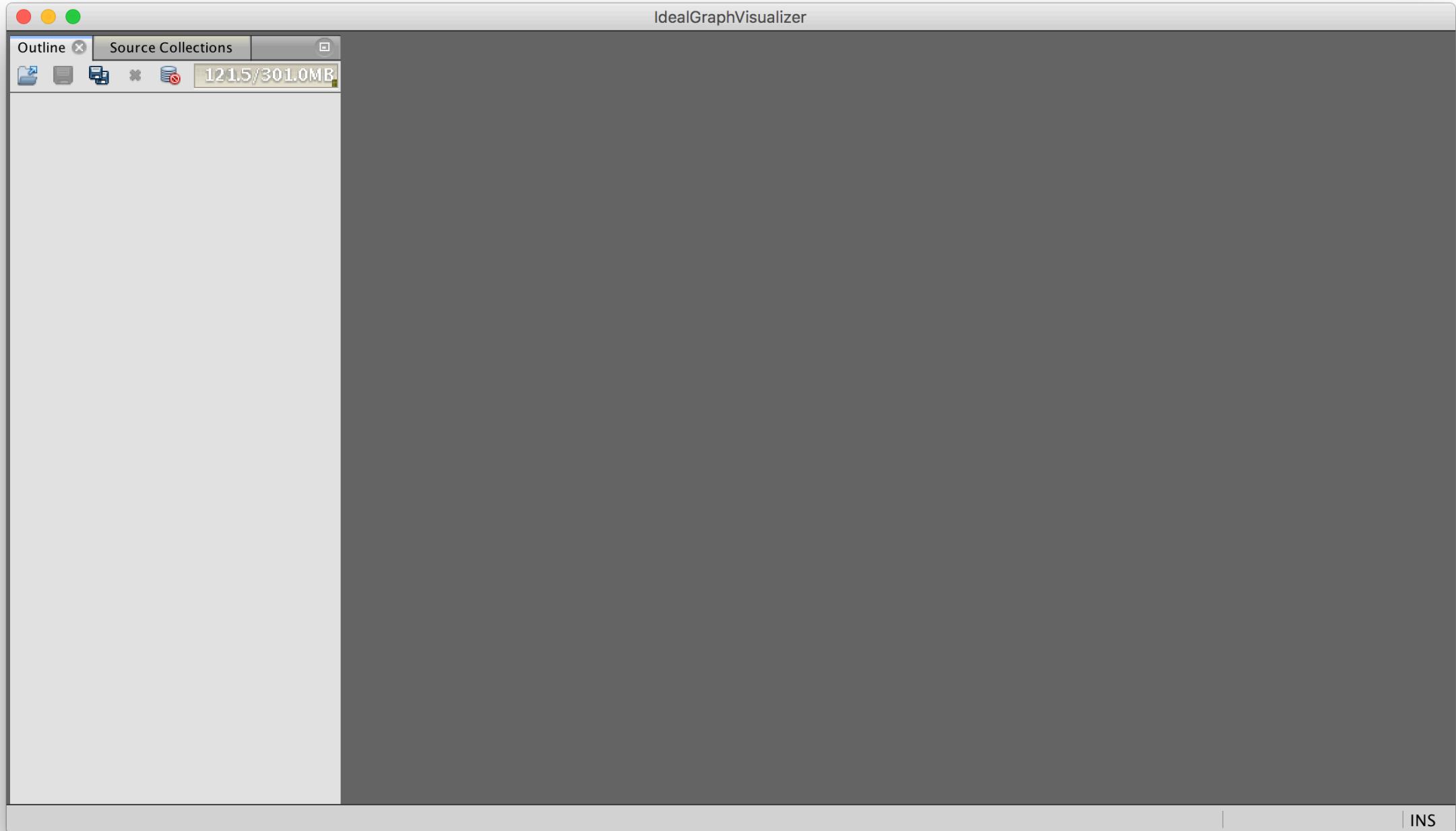
$$x + y$$



# `getX() + getY()`

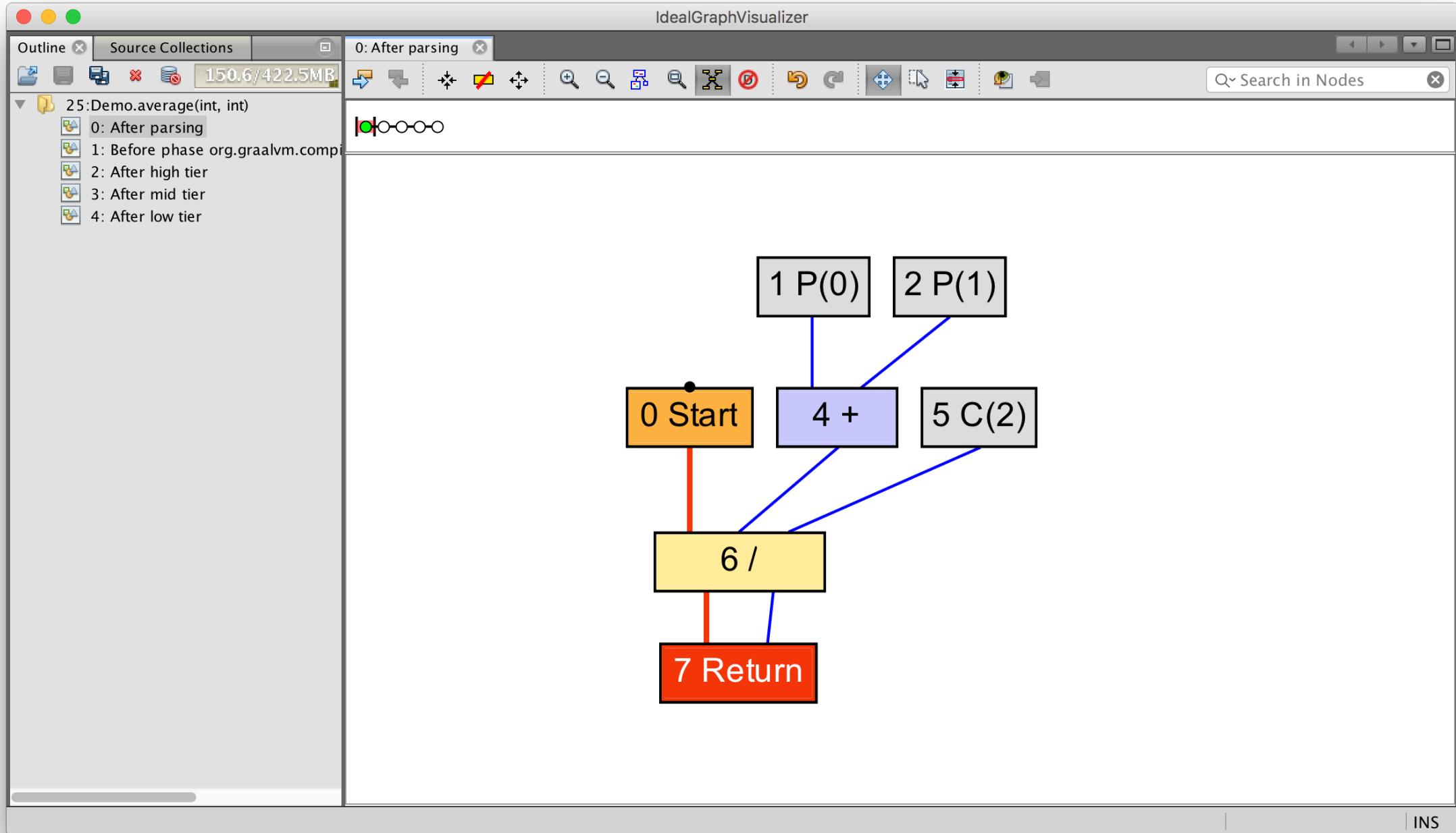


mx igv

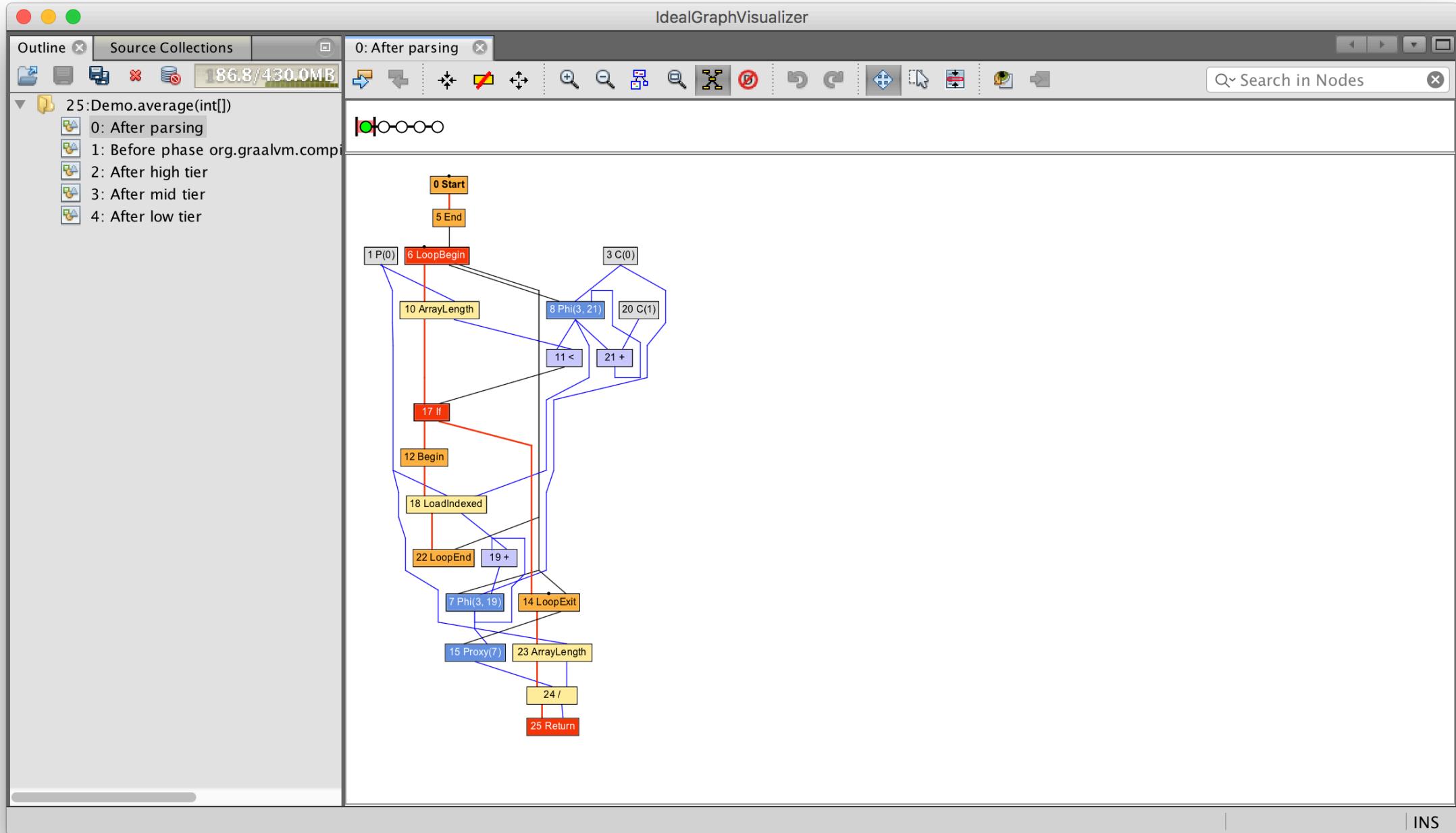


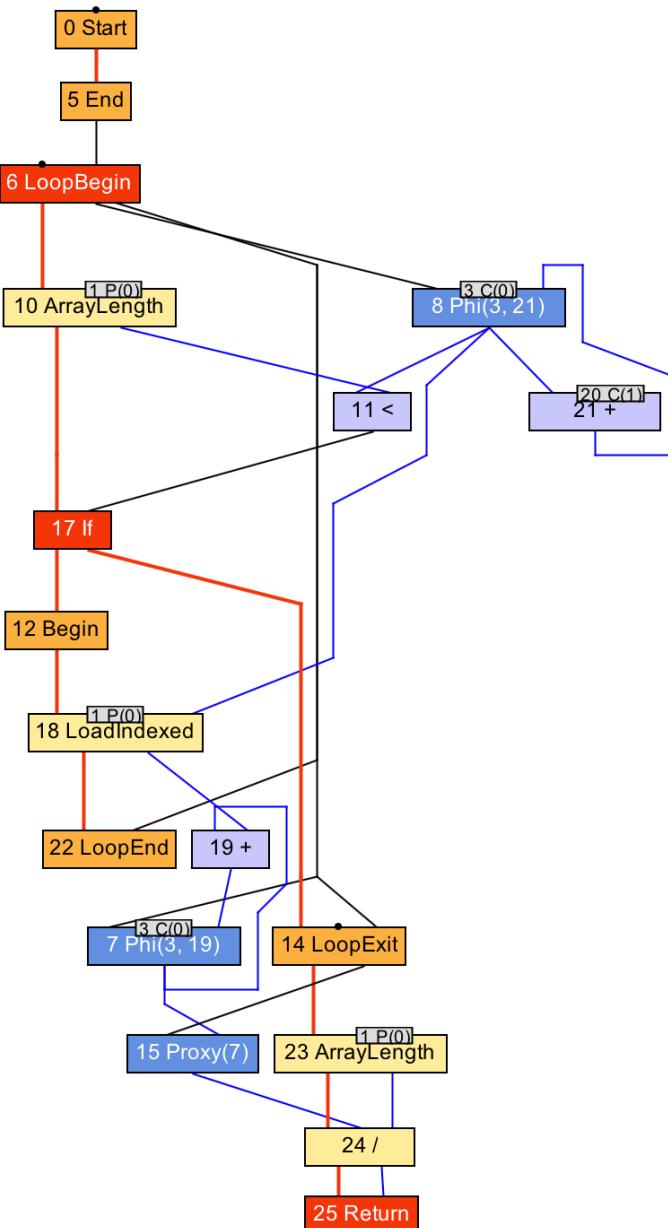
**-Dgraal.Dump**

```
int average(int a, int b) {  
    return (a + b) / 2;  
}
```



```
int average(int[] values) {  
    int sum = 0;  
    for (int n = 0; n < values.length; n++) {  
        sum += values[n];  
    }  
    return sum / values.length;  
}
```





# *Bytecode in*

```
int workload(int a, int b) {  
    return a + b;  
}
```

```
class HotSpotGraalCompiler implements JVMCICompiler {  
    CompilationRequestResult compileMethod(CompilationRequest request) {  
        System.err.println(request.getMethod().getName() + " bytecode: "  
            + Arrays.toString(request.getMethod().getCode()));  
        ...  
    }  
}
```

workload bytecode: [26, 27, 96, -84]

# *The bytecode parser*

The screenshot shows the Eclipse IDE interface with the title bar "graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/AddNode.java - Eclipse". The toolbar contains various icons for file operations, code navigation, and search. The left margin has line numbers from 25 to 69. The code implements the AddNode class, which extends BinaryArithmeticNode<Add> and implements NarrowableArithmeticNode and BinaryCommutative<ValueNode>. It includes methods for creating nodes, performing constant folding, and canonicalization.

```
25+ import org.graalvm.compiler.core.common.type.ArithmeticOpTable;...
40
41 @NodeInfo(shortName = "+")
42 public class AddNode| extends BinaryArithmeticNode<Add> implements NarrowableArithmeticNode, BinaryCommutative<ValueNode> {
43
44     public static final NodeClass<AddNode> TYPE = NodeClass.create(AddNode.class);
45
46     public AddNode(ValueNode x, ValueNode y) {
47         this(TYPE, x, y);
48     }
49
50     protected AddNode(NodeClass<? extends AddNode> c, ValueNode x, ValueNode y) {
51         super(c, ArithmeticOpTable::getAdd, x, y);
52     }
53
54     public static ValueNode create(ValueNode x, ValueNode y) {
55         BinaryOp<Add> op = ArithmeticOpTable.forStamp(x.stamp()).getAdd();
56         Stamp stamp = op.foldStamp(x.stamp(), y.stamp());
57         ConstantNode tryConstantFold = tryConstantFold(op, x, y, stamp);
58         if (tryConstantFold != null) {
59             return tryConstantFold;
60         }
61         if (x.isConstant() && !y.isConstant()) {
62             return canonical(null, op, y, x);
63         } else {
64             return canonical(null, op, x, y);
65         }
66     }
67
68     private static ValueNode canonical(AddNode addNode, BinaryOp<Add> op, ValueNode forX, ValueNode forY) {
69         AddNode self = addNode;
```

graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/AddNode.java - Eclipse

AddNode.java

```
53
54  public static ValueNode create(ValueNode x, ValueNode y) {
55      BinaryOp<Add> op = ArithmeticOpTable.forStamp(x.stamp()).getAdd();
56      Stamp stamp = op.foldStamp(x.stamp(), y.stamp());
57      ConstantNode tryConstantFold = tryConstantFold(op, x, y, stamp);
58      if (tryConstantFold != null) {
59          return tryConstantFold;
60      }
61      if (x.isConstant() && !y.isConstant()) {
62          return canonical(null, op, y, x);
63      } else {
64          return canonical(null, op, x, y);
65      }
66  }
67
68  private static ValueNode canonical(AddNode addNode, BinaryOp<Add> op, ValueNode forX, ValueNode forY) {
69      AddNode self = addNode;
70      ...
```

Call Hierarchy

Members calling 'create(ValueNode, ValueNode)' - in workspace

- ▶ add(StructureGraph, ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.nodes.calc.BinaryArithmeticNode
- ▶ add(ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.nodes.calc.BinaryArithmeticNode
- ▶ canonical(MulNode, BinaryOp<Mul>, Stamp, ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.nodes.calc.MulNode (2 matches)
- ▶ findSynonym(ValueNode, ValueNode) : LogicNode - org.graalvm.compiler.nodes.calc.IntegerLessThanNode.LessThanOp (2 matches)
- ▶ genFloatAdd(ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.java.BytecodeParser
- ▼ **genIntegerAdd(ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.java.BytecodeParser**
- ▼ genArithmeticOp(JavaKind, int) : void - org.graalvm.compiler.java.BytecodeParser
- ▶ processBytecode(int, int) : void - org.graalvm.compiler.java.BytecodeParser (4 matches)
- ▶ genIncrement() : void - org.graalvm.compiler.java.BytecodeParser

org.graalvm.compiler.java.BytecodeParser.genIntege...ode y) : ValueNode - org.graalvm.compiler.java/src

The screenshot shows the Eclipse IDE interface with the title bar "graal - org.graalvm.compiler.java/src/org/graalvm/compiler/java/BytecodeParser.java - Eclipse". The toolbar contains various icons for file operations, search, and navigation. Below the toolbar, there are two tabs: "AddNode.java" and "BytecodeParser.java", with "BytecodeParser.java" currently selected. The main editor area displays Java code for generating arithmetic operations. The code uses a switch statement to handle different opcodes (IADD, LADD, FADD, DADD, ISUB, LSUB, FSUB, DSUB, IMUL, LMUL, FMUL, DMUL) by calling corresponding helper methods like genIntegerAdd, genFloatAdd, etc. The code is annotated with line numbers from 3416 to 3446.

```
3416     genStoreIndexed(array, index, kind, value);
3417 }
3418
3419 private void genArithmeticOp(JavaKind kind, int opcode) {
3420     ValueNode y = frameState.pop(kind);
3421     ValueNode x = frameState.pop(kind);
3422     ValueNode v;
3423     switch (opcode) {
3424         case IADD:
3425         case LADD:
3426             v = genIntegerAdd(x, y);
3427             break;
3428         case FADD:
3429         case DADD:
3430             v = genFloatAdd(x, y);
3431             break;
3432         case ISUB:
3433         case LSUB:
3434             v = genIntegerSub(x, y);
3435             break;
3436         case FSUB:
3437         case DSUB:
3438             v = genFloatSub(x, y);
3439             break;
3440         case IMUL:
3441         case LMUL:
3442             v = genIntegerMul(x, y);
3443             break;
3444         case FMUL:
3445         case DMUL:
3446             v = genFloatMul(x, y);
```

```
private void genArithmeticOp(JavaKind kind, int opcode) {
    ValueNode y = frameState.pop(kind);
    ValueNode x = frameState.pop(kind);
    ValueNode v;
    switch (opcode) {
        ...
        case LADD:
            v = genIntegerAdd(x, y);
            break;
        ...
    }
    frameState.push(kind, append(v));
}
```

# *Canonicalisation*

```
interface Phase {  
    void run(Graph graph);  
}
```

```
interface Node {  
    Node canonical();  
}
```

```
class NegateNode implements Node {  
    Node canonical() {  
        if (value instanceof NegateNode) {  
            return ((NegateNode) value).getValue();  
        } else {  
            return this;  
        }  
    }  
}
```

The screenshot shows the Eclipse IDE interface with the title bar "graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/NegateNode.java - Eclipse". The toolbar contains various icons for file operations, code navigation, and toolbars. The main editor window displays the Java code for NegateNode.java. The code implements the ValueNode interface and provides canonicalization and generation logic for arithmetic operations. The method `findSynonym` is highlighted with a blue selection bar.

```
58     }
59
60     @Override
61     public ValueNode canonical(CanonicalizerTool tool, ValueNode forValue) {
62         ValueNode synonym = findSynonym(forValue, getOp(forValue));
63         if (synonym != null) {
64             return synonym;
65         }
66         return this;
67     }
68
69     protected static ValueNode findSynonym(ValueNode forValue) {
70         ArithmeticOpTable.UnaryOp<Neg> negOp = ArithmeticOpTable.forStamp(forValue.stamp()).getNeg();
71         ValueNode synonym = UnaryArithmeticNode.findSynonym(forValue, negOp);
72         if (synonym != null) {
73             return synonym;
74         }
75         if (forValue instanceof NegateNode) {
76             return ((NegateNode) forValue).getValue();
77         }
78         if (forValue instanceof SubNode && !(forValue.stamp() instanceof FloatStamp)) {
79             SubNode sub = (SubNode) forValue;
80             return SubNode.create(sub.getY(), sub.getX());
81         }
82         return null;
83     }
84
85     @Override
86     public void generate(NodeLIRBuilderTool nodeValueMap, ArithmeticLIRGeneratorTool gen) {
87         nodeValueMap.setResult(this, gen.emitNegate(nodeValueMap.operand(getValue())));
88     }

```

# *Global value numbering*

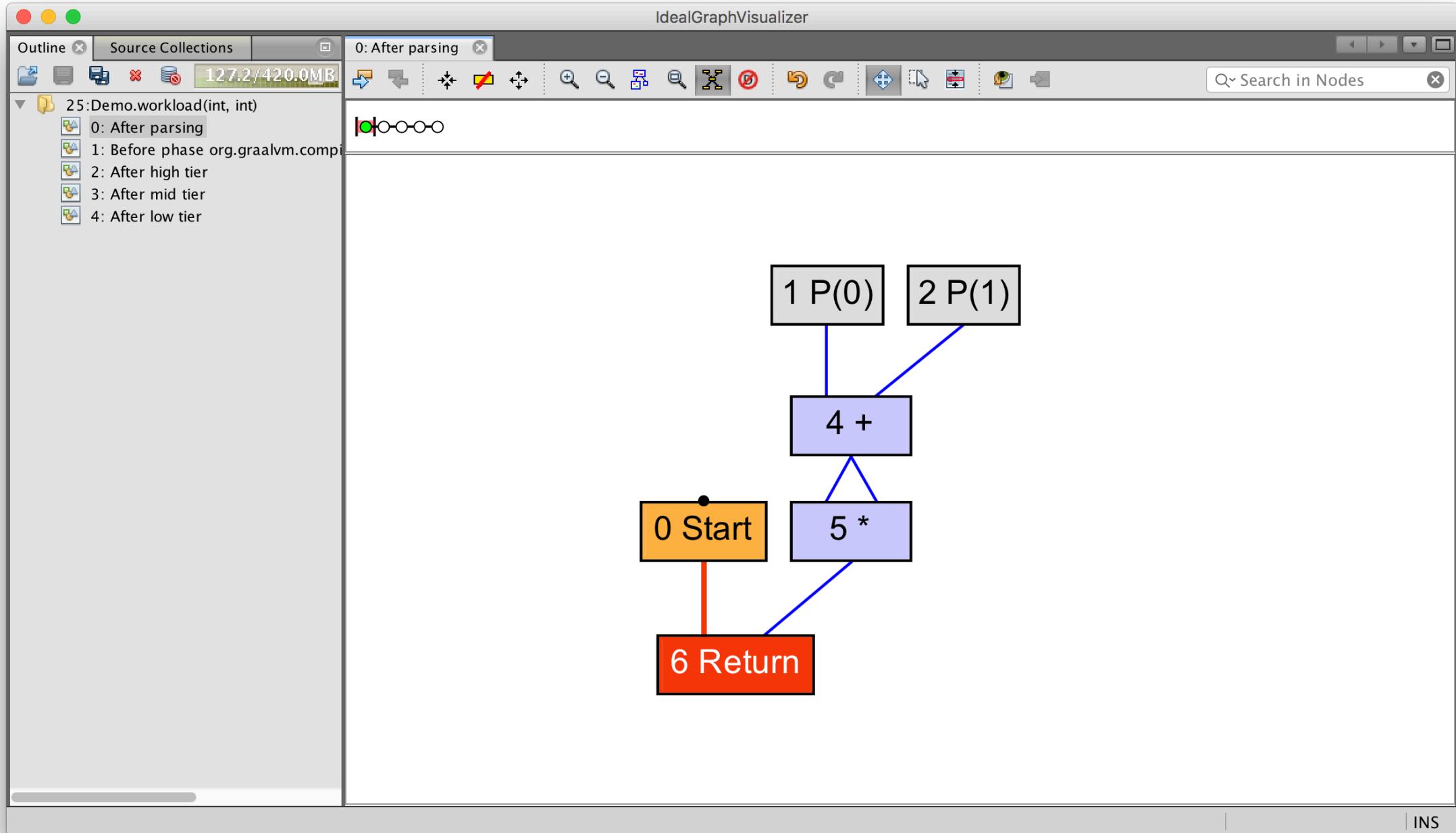
```
int workload(int a, int b) {  
    return (a + b) * (a + b);  
}
```

The screenshot shows the Eclipse IDE interface with the title bar "graal - org.graalvm.compiler.phases.common/src/org/graalvm/compiler/phases/common/CanonicalizerPhase.java - Eclipse". The toolbar contains various icons for file operations like save, cut, copy, paste, and search. The main editor window displays the code for CanonicalizerPhase.java. The code implements a phase for Global Value Numbering (GVN). It includes methods for trying GVN on a node, getting canonicalizable contract assertions, and asserting node equality. The code uses Java 8 features like `Optional` and `Stream`.

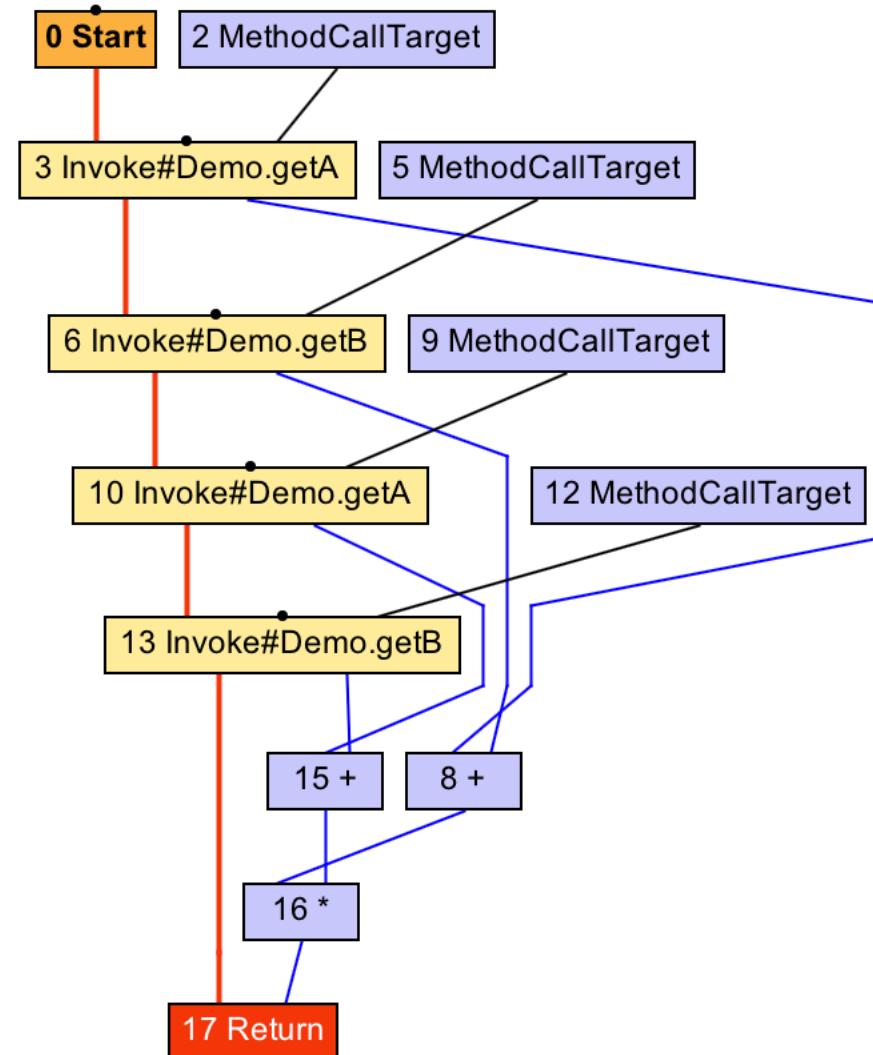
```
    }
    valueNode.usages().forEach(workList::add);
}
return false;
}

281 public boolean tryGlobalValueNumbering(Node node, NodeClass<?> nodeClass) {
282     if (nodeClass.valueNumberable()) {
283         Node newNode = node.graph().findDuplicate(node);
284         if (newNode != null) {
285             assert !(node instanceof FixedNode || newNode instanceof FixedNode);
286             node.replaceAtUsagesAndDelete(newNode);
287             COUNTER_GLOBAL_VALUE_NUMBERING_HITS.increment(debug);
288             debug.log("GVN applied and new node is %1s", newNode);
289             return true;
290         }
291     }
292     return false;
293 }

294 private AutoCloseable getCanonicalizeableContractAssertion(Node node) {
295     boolean needsAssertion = false;
296     assert (needsAssertion = true) == true;
297     if (needsAssertion) {
298         Mark mark = node.graph().getMark();
299         return () -> {
300             assert mark.equals(node.graph().getMark()) : "new node created while canonicalizing " + node.getClass();
301                         .node.graph().getNewNodes(mark).snapshot();
302         };
303     } else {
304 }
```



```
int workload() {  
    return (getA() + getB()) * (getA() + getB());  
}
```



# *Lock coarsening*

```
void workload() {  
    synchronized (monitor) {  
        counter++;  
    }  
    synchronized (monitor) {  
        counter++;  
    }  
}
```

```
void workload() {  
    monitor.enter();  
    counter++;  
    monitor.exit();  
    monitor.enter();  
    counter++;  
    monitor.exit();  
}
```

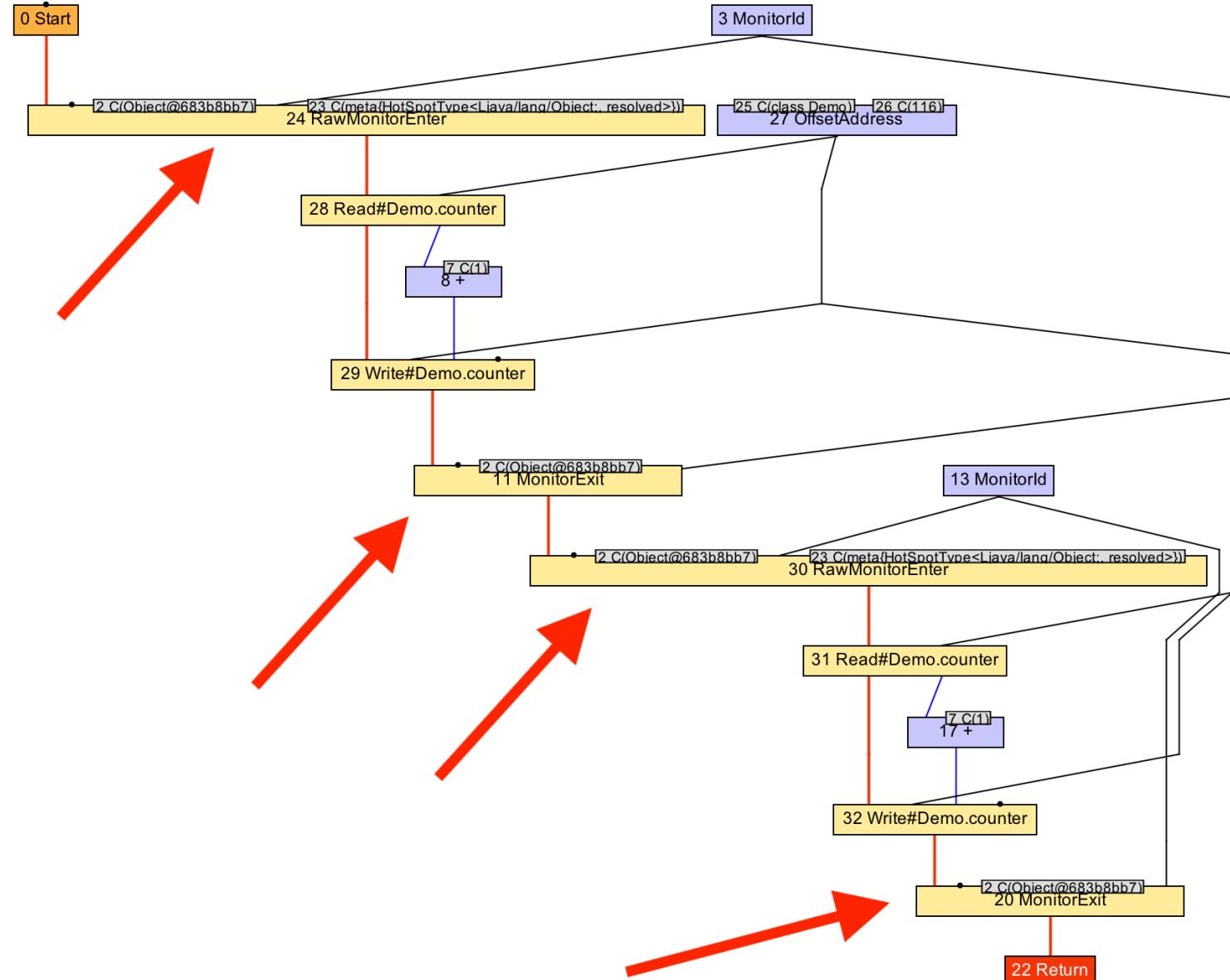
```
void workload() {  
    monitor.enter();  
    counter++;  
    counter++;  
    monitor.exit();  
}
```

```
void run(StructuredGraph graph) {
    for (monitorExitNode monitorExitNode : graph.getNodes(monitorExitNode.class)) {
        FixedNode next = monitorExitNode.next();
        if (next instanceof monitorEnterNode) {
            AccessmonitorNode monitorEnterNode = (AccessmonitorNode) next;
            if (monitorEnterNode.object() == monitorExitNode.object()) {
                monitorExitNode.remove();
                monitorEnterNode.remove();
            }
        }
    }
}
```

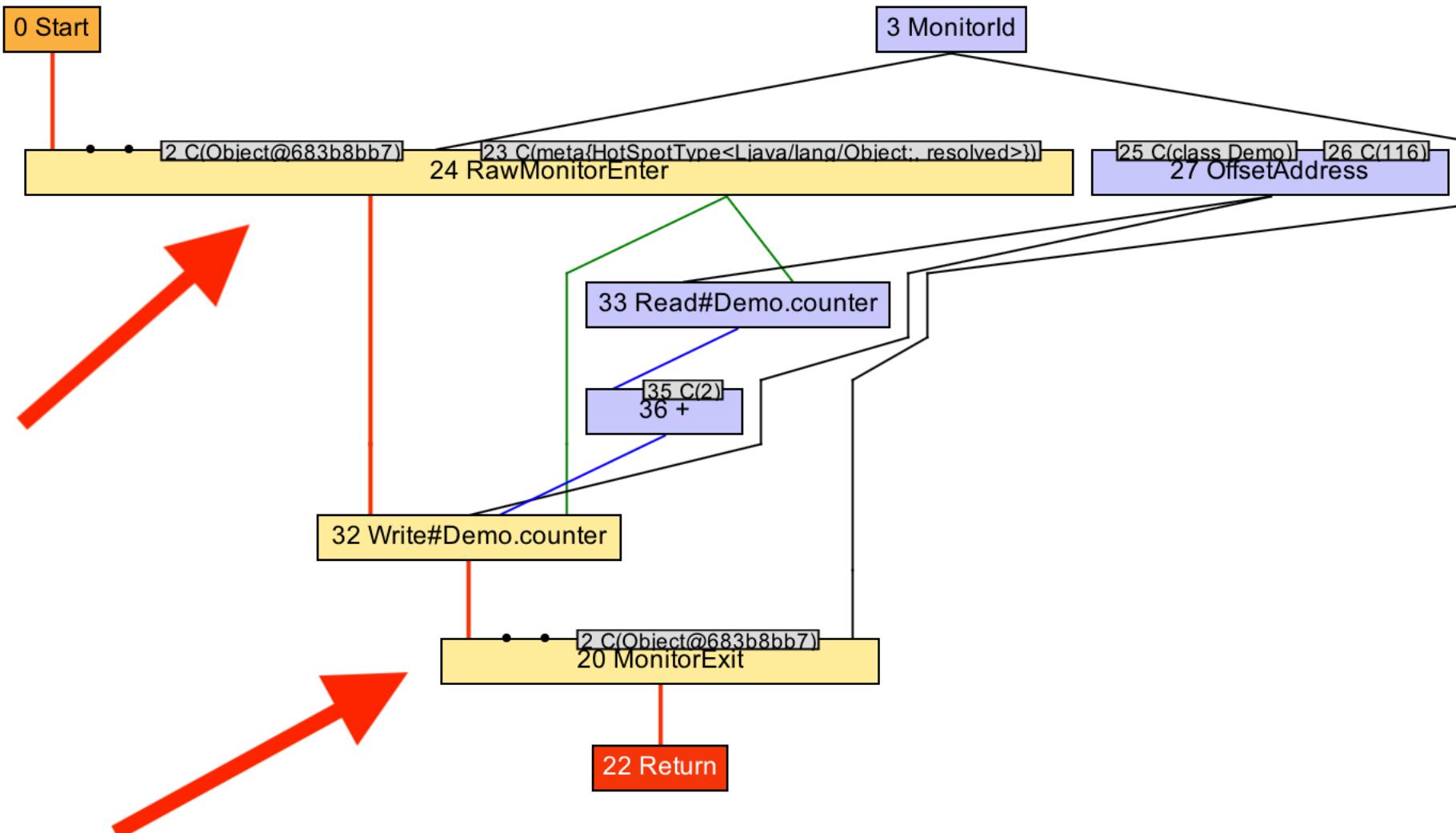
The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** graal - org.graalvm.compiler.phases.common/src/org/graalvm/compiler/phases/common/LockEliminationPhase.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and navigation.
- Quick Access:** A dropdown menu labeled "Quick Access" is visible on the right side of the toolbar.
- Code Editor:** The main window displays the Java code for `LockEliminationPhase.java`. The code implements the `Phase` interface and overrides the `run` method to process `MonitorExitNode` and `MonitorEnterNode` nodes in a `StructuredGraph`.
- Code Content:**

```
35
36 public class LockEliminationPhase extends Phase {
37
38     @Override
39     protected void run(StructuredGraph graph) {
40         for (MonitorExitNode monitorExitNode : graph.getNodes(MonitorExitNode.TYPE)) {
41             FixedNode next = monitorExitNode.next();
42             if (!(next instanceof MonitorEnterNode || next instanceof RawMonitorEnterNode)) {
43                 // should never happen, osr monitor enters are always direct successors of the graph
44                 // start
45                 assert !(next instanceof OSRMonitorEnterNode);
46                 AccessMonitorNode monitorEnterNode = (AccessMonitorNode) next;
47                 if (GraphUtil.unproxy(monitorEnterNode.object()) == GraphUtil.unproxy(monitorExitNode.object())) {
48                     /*
49                     * We've coarsened the lock so use the same monitor id for the whole region,
50                     * otherwise the monitor operations appear to be unrelated.
51                     */
52                     MonitorIdNode enterId = monitorEnterNode.getMonitorId();
53                     MonitorIdNode exitId = monitorExitNode.getMonitorId();
54                     if (enterId != exitId) {
55                         enterId.replaceAndDelete(exitId);
56                     }
57                     GraphUtil.removeFixedWithUnusedInputs(monitorEnterNode);
58                     GraphUtil.removeFixedWithUnusedInputs(monitorExitNode);
59                 }
60             }
61         }
62     }
63 }
64 }
```
- Bottom Status Bar:** Icons for RSS feed, Writable status, Smart Insert, and line numbers (36 : 34).



```
void workload() {  
    monitor.enter();  
    counter += 2;  
    monitor.exit();  
}
```



# Static Compilation

```
$ javac Hello.java

$ graalvm-0.28.2/bin/native-image Hello
  classlist:      966.44 ms
    (cap):      804.46 ms
    setup:     1,514.31 ms
  (typeflow):   2,580.70 ms
    (objects):   719.04 ms
  (features):    16.27 ms
    analysis:   3,422.58 ms
    universe:   262.09 ms
    (parse):    528.44 ms
    (inline):   1,259.94 ms
  (compile):   6,716.20 ms
    compile:   8,817.97 ms
    image:     1,070.29 ms
  debuginfo:    672.64 ms
    write:     1,797.45 ms
 [total]:   17,907.56 ms
```

```
$ ls -lh hello
-rwxr-xr-x  1 chrisseaton  staff   6.6M  4 Oct 18:35 hello

$ file ./hello
./hellojava: Mach-O 64-bit executable x86_64

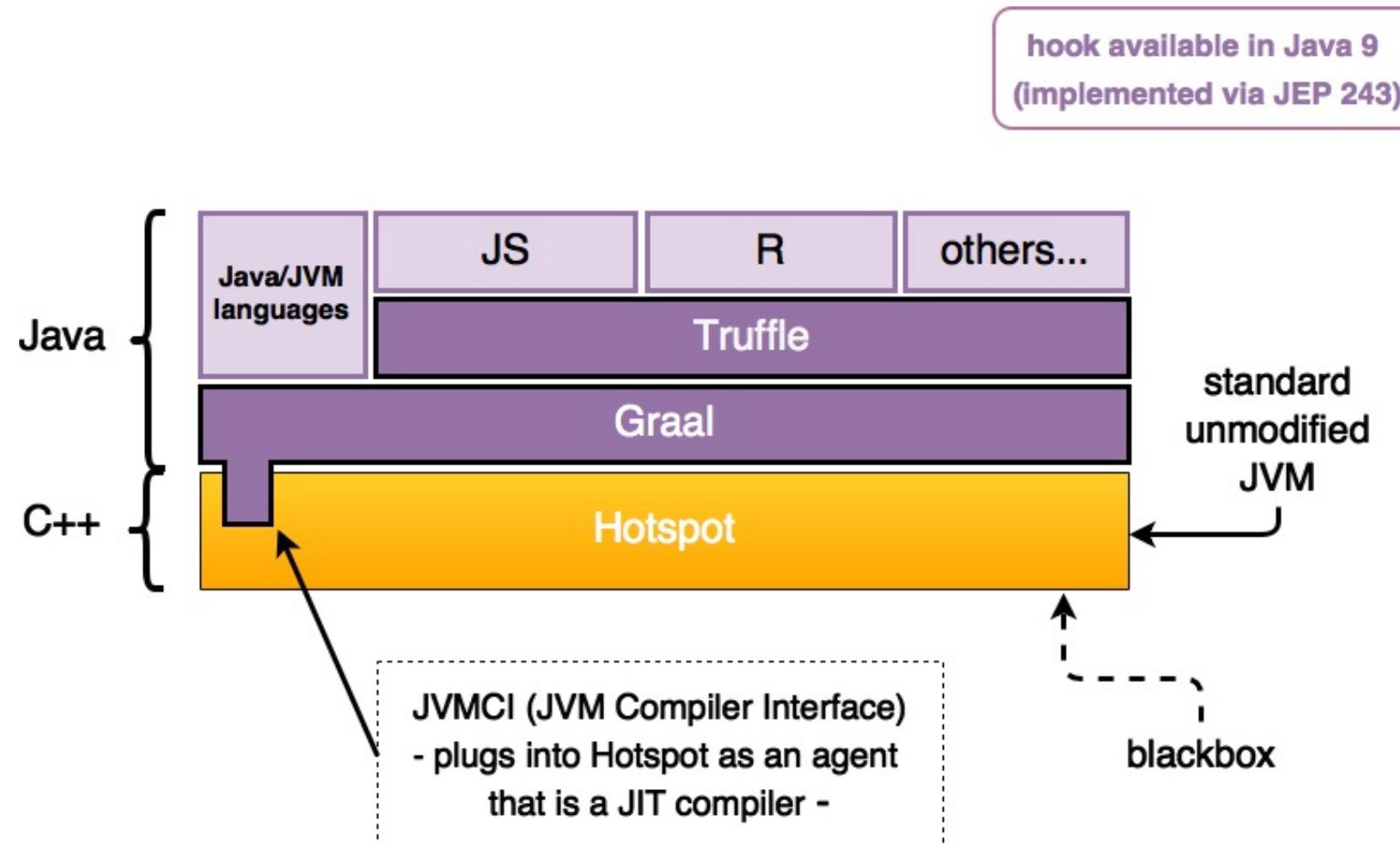
$ time ./hello
Hello!

real    0m0.010s
user    0m0.003s
sys  0m0.003s
```

# Truffle

a Java framework for implementing languages as simple interpreters

# Graal Compiler



# Partial Evaluation

# Program Specialization

- program optimization technique
- input: program + some of its input data (**static input data**)
- output: specialized program (to be run on **dynamic input data**)
- yields same result as original program on all input data
- applied to interpreters
- input: interpreter + program to interpreter
- output: specialized interpreter
- dynamic input data: program input
- can significantly reduce or completely remove **interpretive overhead**

# Simple Program

```
int p(int n, int x) {  
    if (n == 0) {  
        return 1;  
    } else if (even(n)) {  
        return p(n/2, x) * p(n/2, x);  
    } else {  
        return x * p(n-1, x);  
    }  
}
```

```
int p3(int x) {  
    if (3 == 0) {  
        return 1;  
    } else if (even(3)) {  
        return p(3/2, x) * p(3/2, x);  
    } else {  
        return x * p(3-1, x);  
    }  
}
```

# Simple Program

```
int p(int n, int x) {  
    if (n == 0) {  
        return 1;  
    } else if (even(n)) {  
        return p(n/2, x) * p(n/2, x);  
    } else {  
        return x * p(n-1, x);  
    }  
}
```

```
int p3(int x) {  
    if (false) {  
        return 1;  
    } else if (false) {  
        return p(3/2, x) * p(3/2, x);  
    } else {  
        return x * p(2, x);  
    }  
}
```

# Simple Program

```
int p(int n, int x) {  
    if (n == 0) {  
        return 1;  
    } else if (even(n)) {  
        return p(n/2, x) * p(n/2, x);  
    } else {  
        return x * p(n-1, x);  
    }  
}
```

```
int p3(int x) {  
    return x * p(2, x);  
}
```

# Simple Program

```
int p(int n, int x) {  
    if (n == 0) {  
        return 1;  
    } else if (even(n)) {  
        return p(n/2, x) * p(n/2, x);  
    } else {  
        return x * p(n-1, x);  
    }  
}
```

```
int p3(int x) {  
    return x * p(2, x);  
}  
  
int p3(int x) {  
    return x * p(1, x) * p(1, x);  
}  
  
int p3(int x) {  
    return x * x * p(0, x) * x * p(0, x);  
}
```

# Simple Program

```
int p(int n, int x) {  
    if (n == 0) {  
        return 1;  
    } else if (even(n)) {  
        return p(n/2, x) * p(n/2, x);  
    } else {  
        return x * p(n-1, x);  
    }  
}
```

```
int p3(int x) {  
    return x * x * p(0, x) * x * p(0, x);  
}  
  
int p3(int x) {  
    return x * x * 1 * x * 1;  
}  
  
int p3(int x) {  
    return x * x * x;  
}
```

# Interpreter

```
int interpret(Program p, int[] args) {  
    return p.execute(args);  
}
```

```
sample = new Program(new Add(new Add(new Arg(0), new Arg(1)), new Arg(2)));
```

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

# Interpreter

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

```
class Program {  
    final Exp exp;  
    public Program(Exp exp) {  
        this.exp = exp;  
    }  
    public Object execute(Object[] args) {  
        return exp.execute(args);  
    }  
}
```

# Interpreter

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

```
class Add extends Exp {  
    final Exp left, right;  
    public Add(Exp left, Exp right) {  
        this.left = left;  
        this.right = right;  
    }  
    @Override int execute(int[] args) {  
        return left.execute(args) +  
            right.execute(args);  
    }  
}
```

# Interpreter

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

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

```
class Add extends Exp {  
    final Exp left, right;  
    public Add(Exp left, Exp right) {  
        this.left = left;  
        this.right = right;  
    }  
    @Override int execute(int[] args) {  
        return left.execute(args) +  
               right.execute(args);  
    }  
}
```

# Interpreter

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

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

```
class Arg extends Exp {  
    final int index;  
    public Arg(int index) {  
        this.index = index;  
    }  
    @Override int execute(int[] args) {  
        return args[index];  
    }  
}
```

# Interpreter

```
int interpret(Program p, int[] args) {  
    return p.execute(args);  
}
```

```
sample = new Program(new Add(new Add(new Arg(0), new Arg(1)), new Arg(2)));
```

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

# Truffle

## AST interpreters

```
class Program extends RootNode {  
    @Child Expression exp;  
    public Program(Expression exp) {  
        super(null);  
        this.exp = exp;  
    }  
    @Override  
    public Object execute(VirtualFrame frame) {  
        return exp.execute(frame getArguments());  
    }  
}
```

# Truffle

## AST interpreters

```
abstract class Expression extends Node {  
    abstract Object execute(VirtualFrame frame);  
}
```

# Truffle

## AST interpreters

```
class Arg extends Expression {  
    final int index;  
    public Arg(int index) {  
        this.index = index;  
    }  
    @Override  
    Object execute(VirtualFrame frame) {  
        return frame.getArguments()[index];  
    }  
}
```

# Truffle

## partial evaluation

```
Object interpret(Program p, Object[] args) {  
    CallTarget target = Truffle.getRuntime().createCallTarget(p);  
    return target.call(args);  
}
```

# Truffle

## partial evaluation

```
@Override
Object execute(VirtualFrame frame) {
    Object leftResult = left.execute(frame);
    Object rightResult = right.execute(frame);
    if (leftResult instanceof Integer && rightResult instanceof Integer) {
        return (int) leftResult + (int) rightResult;
    }
    return concatObjects(leftResult, rightResult);
}
@TruffleBoundary
private String concatObjects(Object leftResult, Object rightResult) {
    return leftResult.toString() + rightResult.toString();
}
```

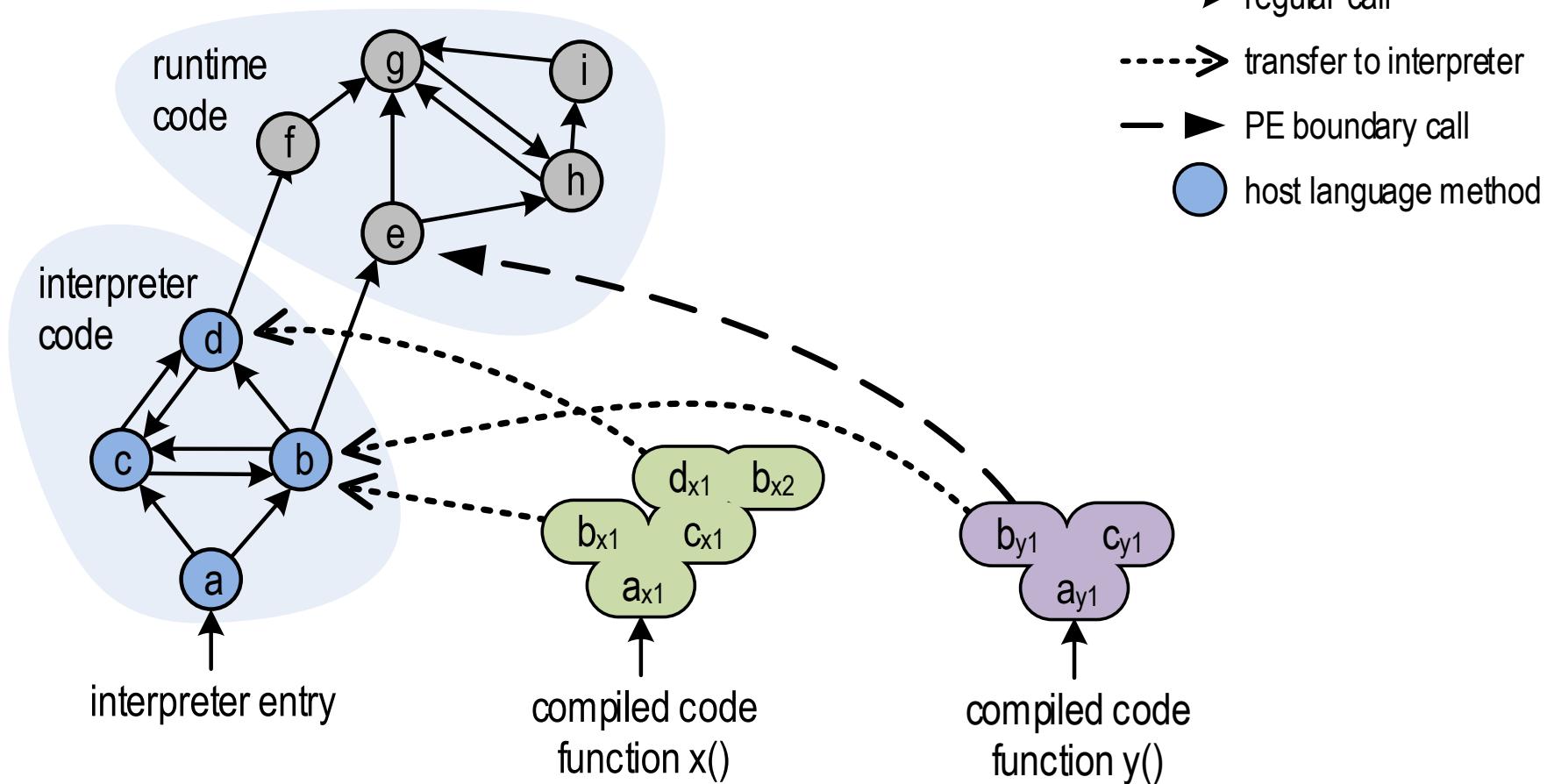
# Truffle

## partial evaluation

```
@Override Object execute(VirtualFrame frame) {  
    Object leftResult = left.execute(frame);  
    Object rightResult = right.execute(frame);  
    if (leftResult instanceof Integer && rightResult instanceof Integer) {  
        return (int) leftResult + (int) rightResult;  
    }  
    if (leftResult instanceof String || rightResult instanceof String) {  
        return concatStrings(leftResult, rightResult);  
    } else {  
        transferToInterpreter();  
        throw new IllegalArgumentException();  
    }  
}
```

# Truffle

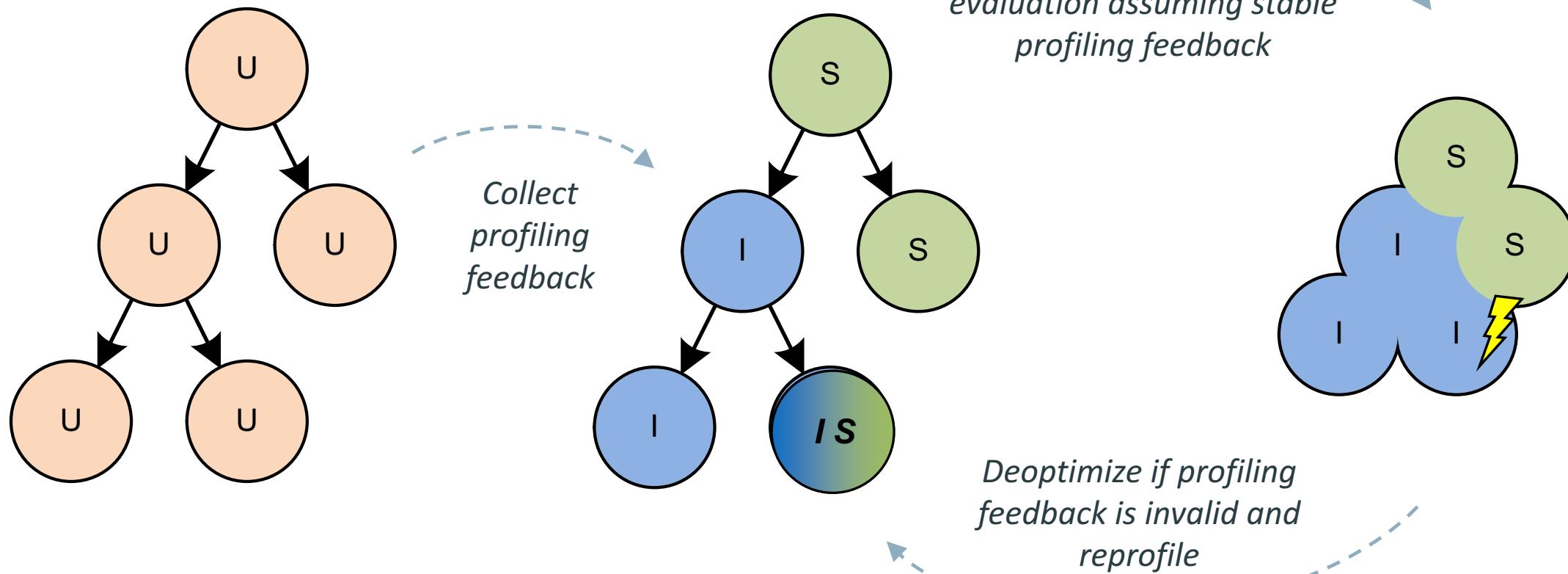
## partial evaluation



# Self-optimizing AST interpreters

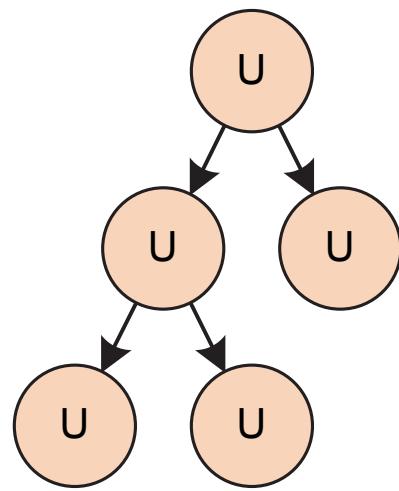
# Truffle

## profiling & speculative optimization



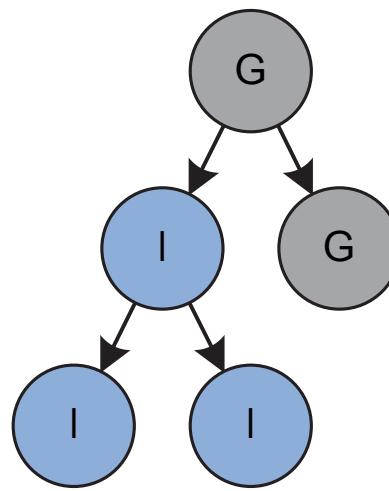
# Rewriting AST Nodes

## profiling & speculative optimization



AST Interpreter  
Uninitialized Nodes

Node Rewriting  
for Profiling Feedback

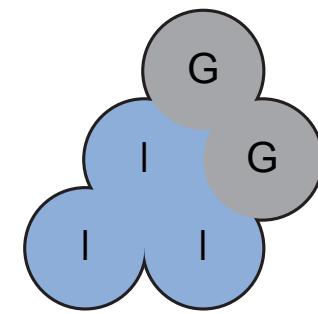


AST Interpreter  
Rewritten Nodes

Compilation using  
Partial Evaluation



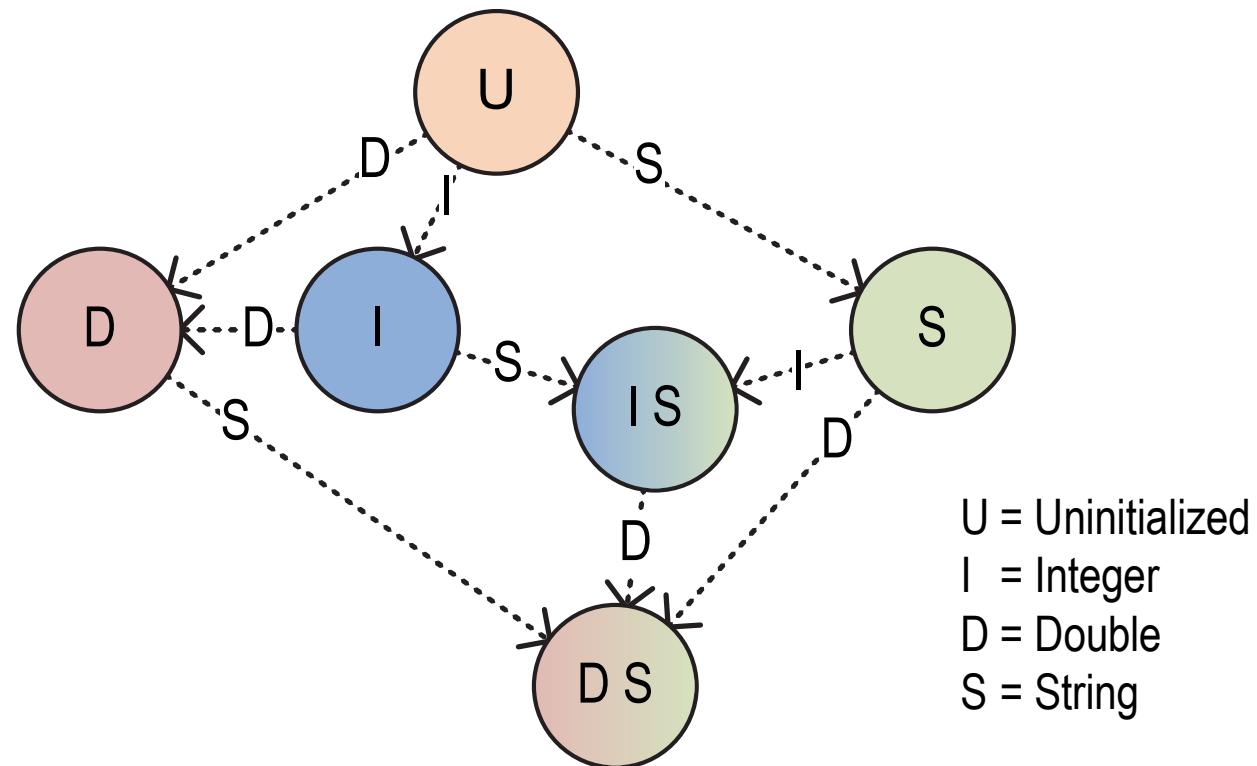
Deoptimization  
to AST Interpreter



Compiled Code

# Rewriting AST Nodes

## transition graph



# Rewriting AST Nodes specialization

```
@NodeChild("left") @NodeChild("right")
abstract class AddNode extends BaseNode {

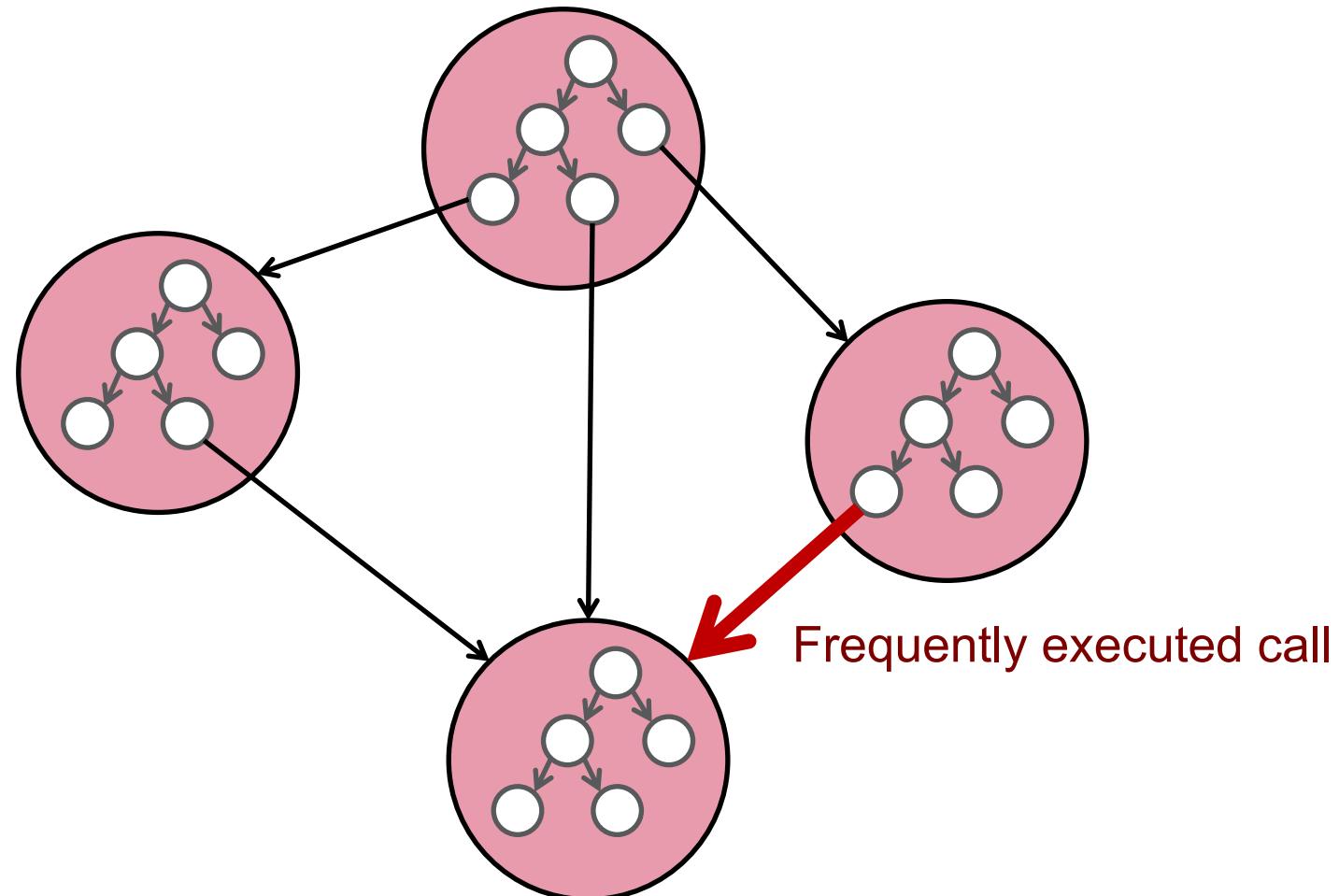
    @Specialization(rewriteOn=ArithException.class)
    int doInt(int leftValue, int rightValue) {
        return Math.addExact(leftValue, rightValue);
    }

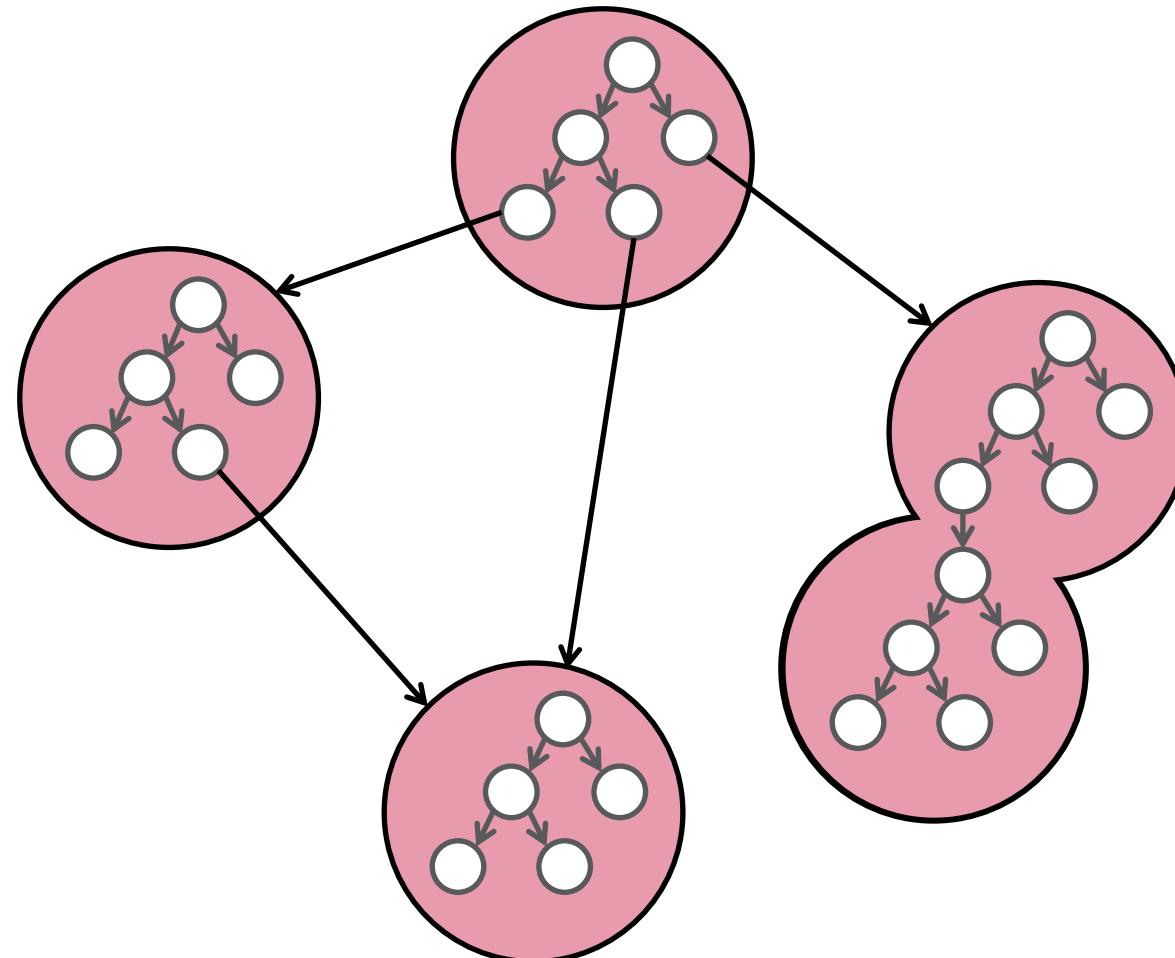
    @Specialization
    @Contains("doInt")
    double doDouble(double leftValue, double rightValue) {
        return leftValue + rightValue;
    }

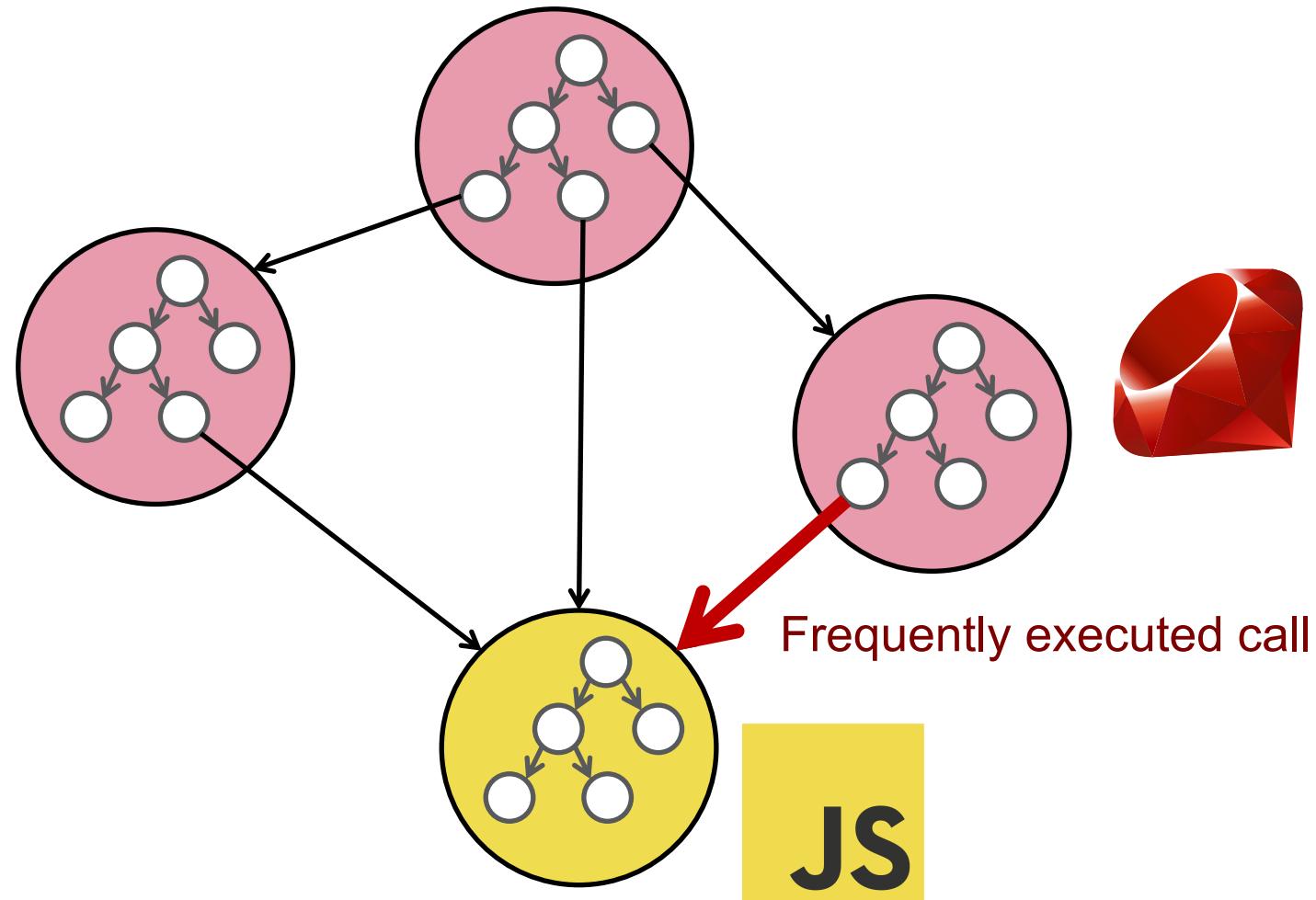
    @Specialization(guards = "isString")
    String doString(Object leftValue, Object rightValue) {
        return leftValue.toString() + rightValue.toString();
    }

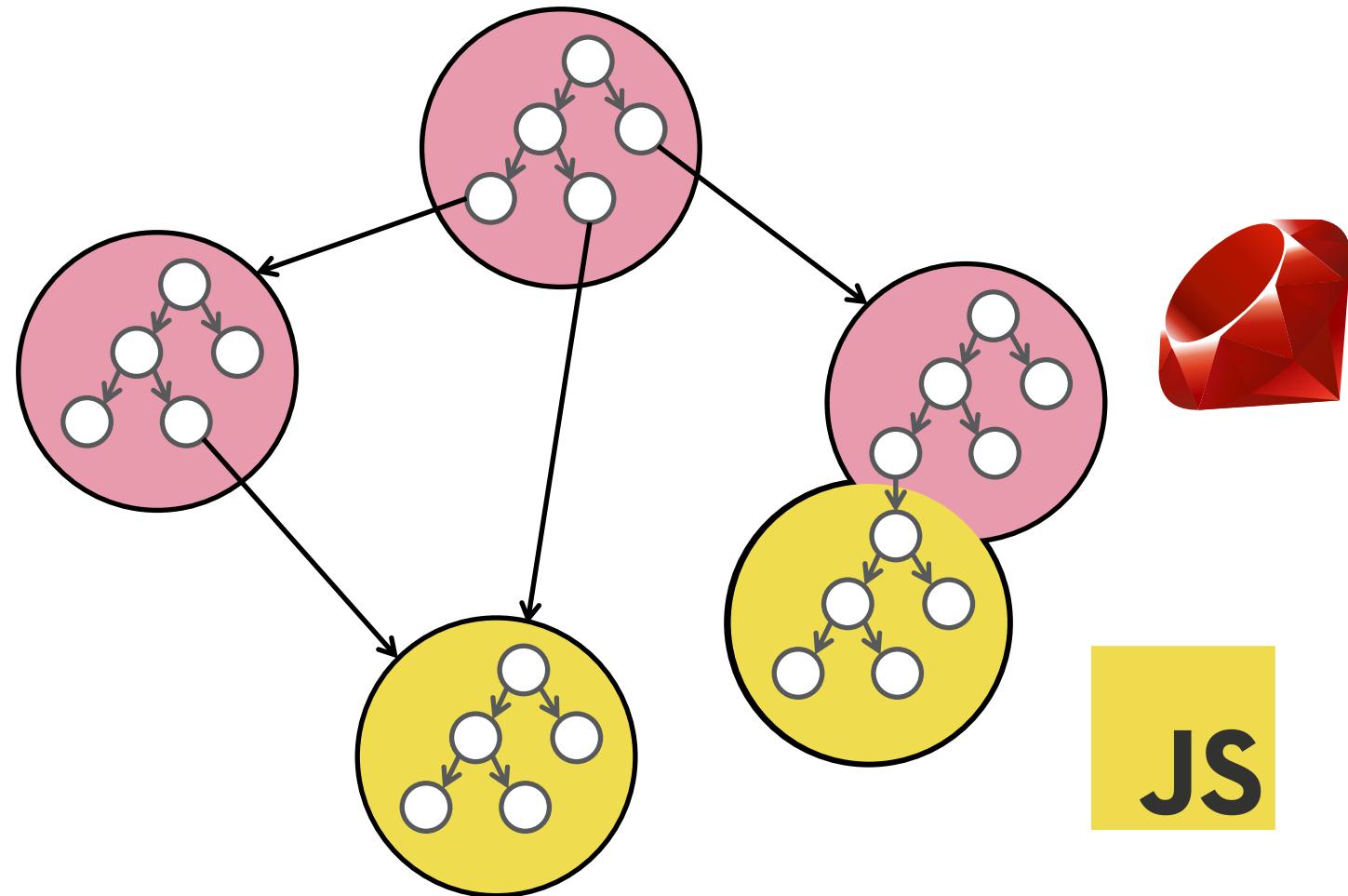
    boolean isString(Object leftValue, Object rightValue) {
        return leftValue instanceof String
            || rightValue instanceof String;
    }
}
```

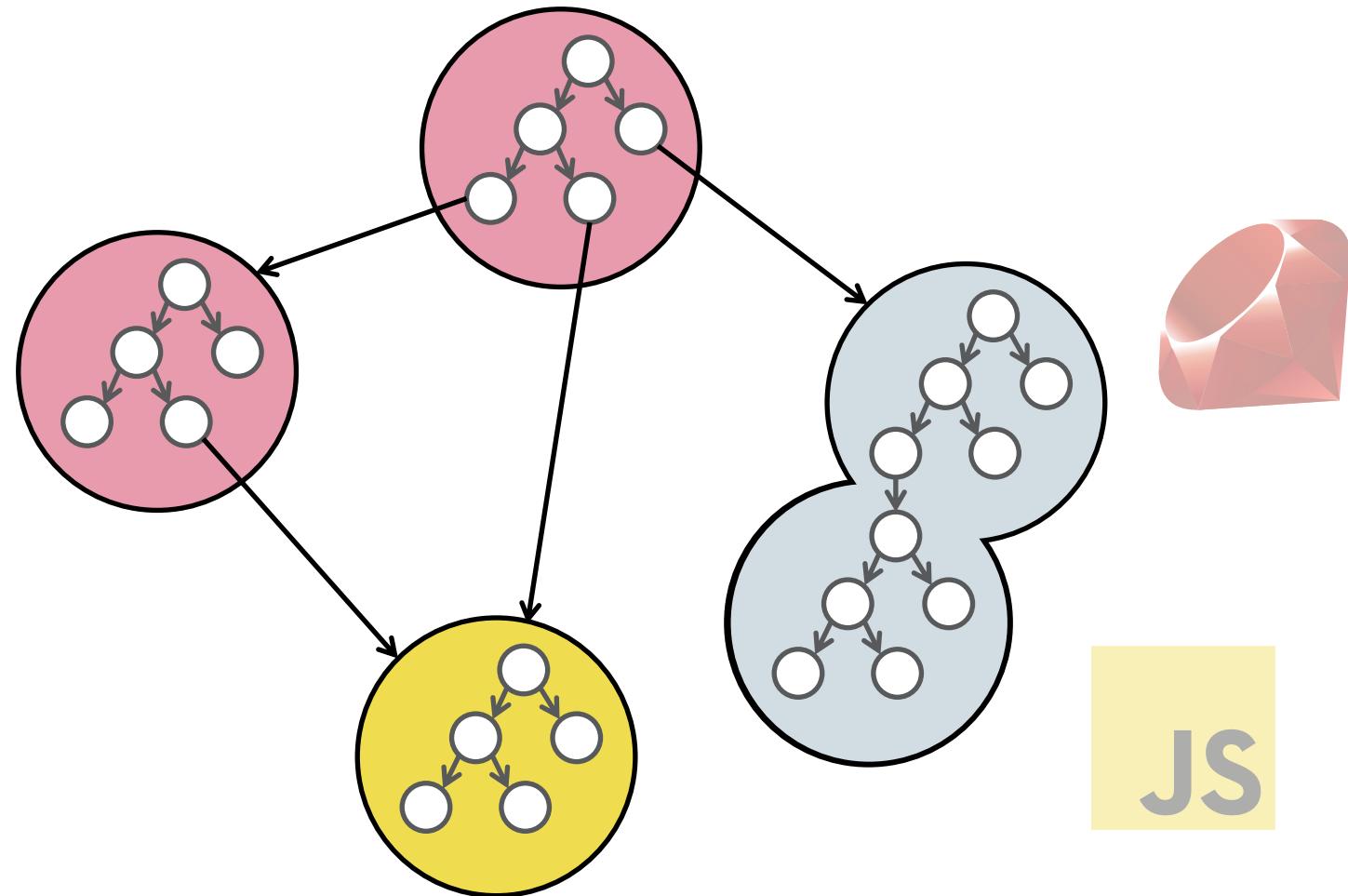
# Polyglot AST interpreters











- We're not really suggesting that people routinely write alternate methods in different languages
- More about removing the consideration of performance from the decision if you do want to combine languages

- Could make all library ecosystems available to all applications
- May be useful for unifying a front-end and back-end
- May be useful in handling legacy applications and incremental changes in implementation language

# Walnut

One Data Processing Platform to Rule Them All



# Goals

## Multi-lingual data processing

- Integrate new programming languages into RDBMS
- Rapid, efficient, eco-system friendly

## Dynamic compilation

- Apply modern runtime compilation technologies to data processing engine

## Inter-operability

- Data models: table, vector/matrix, graph
- Computational models: relational algebra, linear algebra, ...
- Efficient interoperability between engines

## Language integration

- Idioms for query, analytics, Machine Learning

# In-Database JavaScript Programming

## JavaScript

```
namespace myFunctions {  
    export function hello(name : string) : string {  
        return "Hello " + name;  
    }  
}  
export = myFunctions;
```

- JavaScript & TypeScript
- Tools from JavaScript ecosystem
- Build user-defined extension packages using packages from [npm](#)

## Oracle Database MLE

```
$> db.js -c $DBCONN -u $USER -p $PASSWORD  
deploy hello.ts
```

```
CREATE FUNCTION HELLO(p0 IN VARCHAR2)  
RETURN VARCHAR2 AS LANGUAGE JS  
LIBRARY "hello.js" NAME "hello"  
PARAMETERS(p0 STRING)
```

```
SELECT hello(ENAME) FROM EMP
```

- Stored procedures, user-defined aggregations, table functions

# Powered by Graal

## Graal compiler

- Dynamic compiler
- Aggressive speculative optimization
- Feedback-driven
- De-optimization into interpreted code

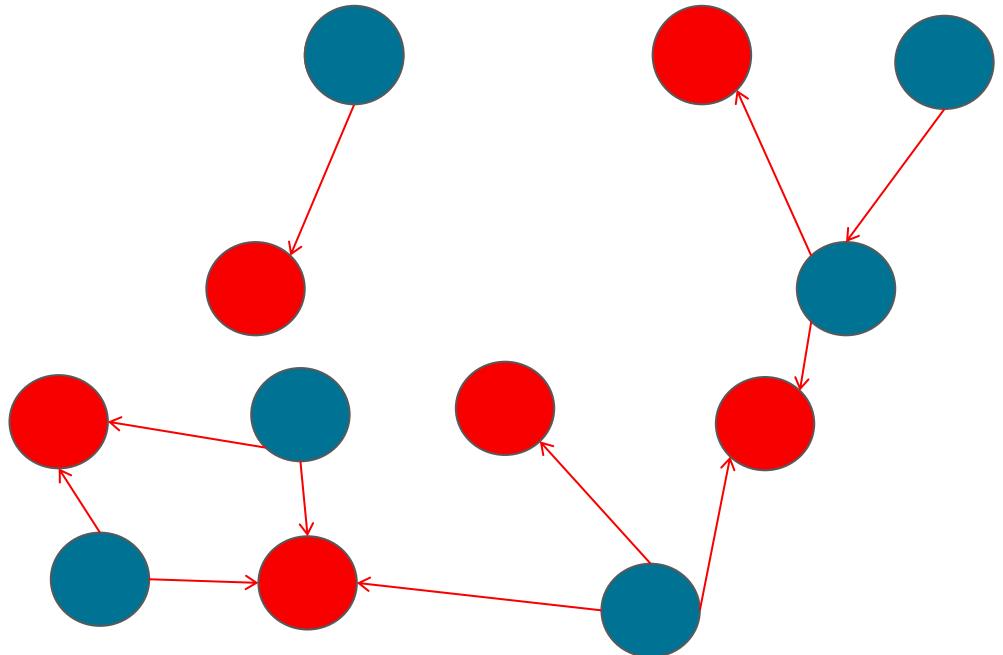
## Truffle framework

- Framework to implement programming languages
- High performance
- Self-optimizing AST interpreters
- Front-end to Graal compiler
- JavaScript, Ruby, R, Python implementations

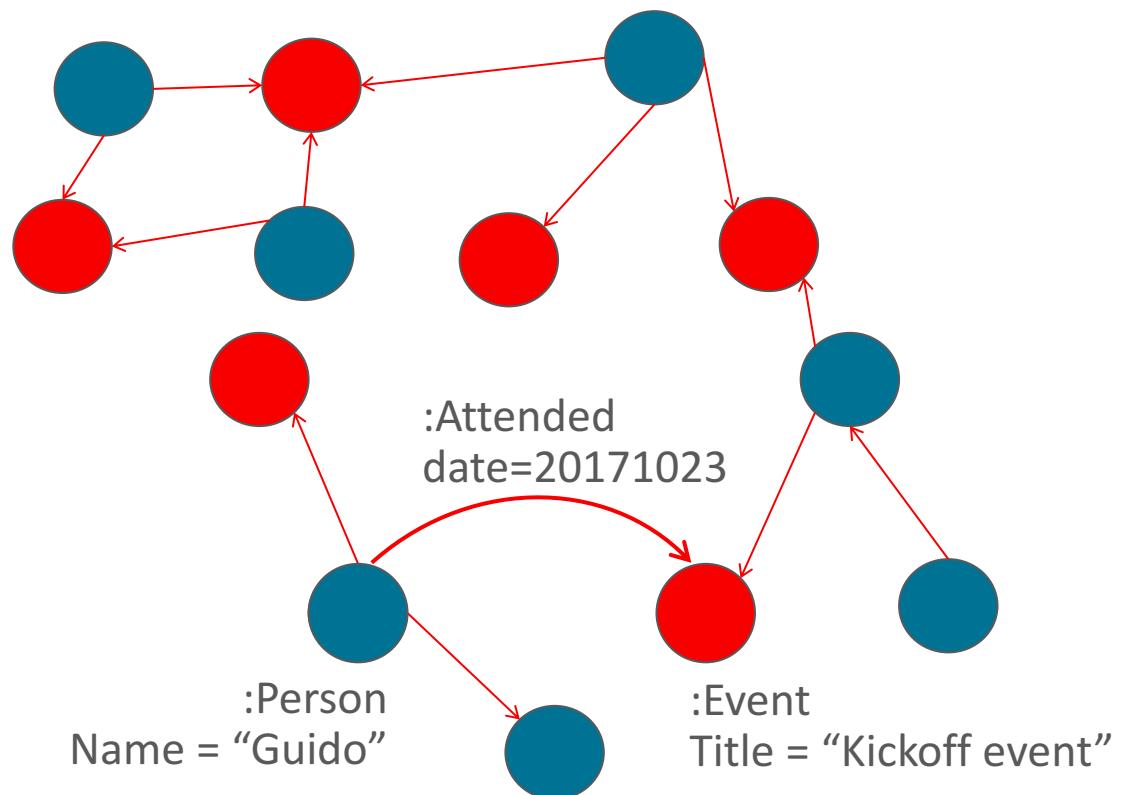
## Substrate VM

- SDK to build lightweight embeddable VMs
- Memory management
- Stack walking
- De-optimization
- Multi-threading
- Lightweight isolation

# Parallel Graph AnalytiX



# Data as a Graph

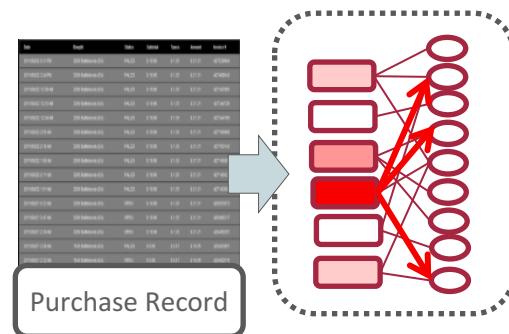


- Vertex: entity
- Edge: relationship
- Labels: identify vertices and edges
- Properties: describe vertices and edges

# Example Applications

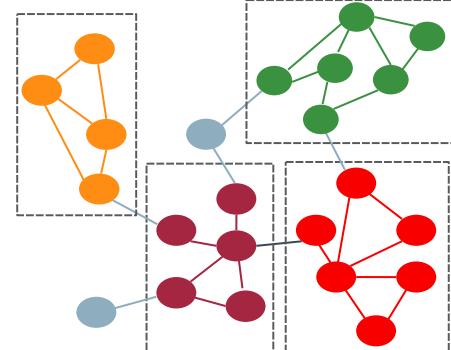
## Product recommendation

Recommend the most **similar** item purchased by **similar** people



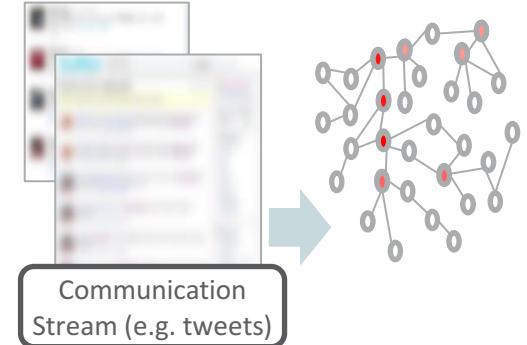
## Community detection

Identify group of people that are **close to each other** – e.g. target group marketing



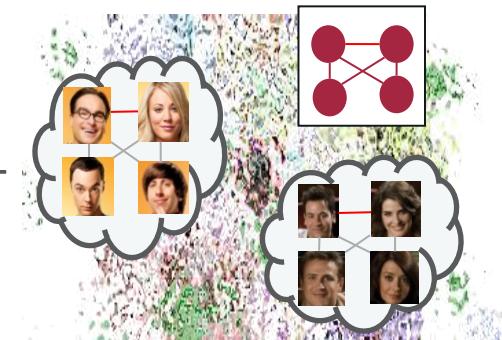
## Influencer identification

Find out people that are **central** in the given network – e.g. influencer marketing



## Graph pattern matching

Find out all sets of entities that **match a given pattern** – e.g. fraud detection



# Graph Workloads

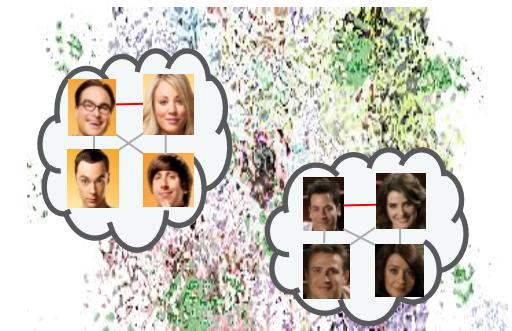
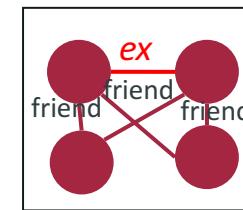
## Computational graph analytics

- Compute values on vertices and edges
- While traversing or iterating on the graph
- In procedural ways
- Connected Components, Shortest Path, Spanning Tree, Pagerank, Centrality

Domain-specific language: Green-Marl

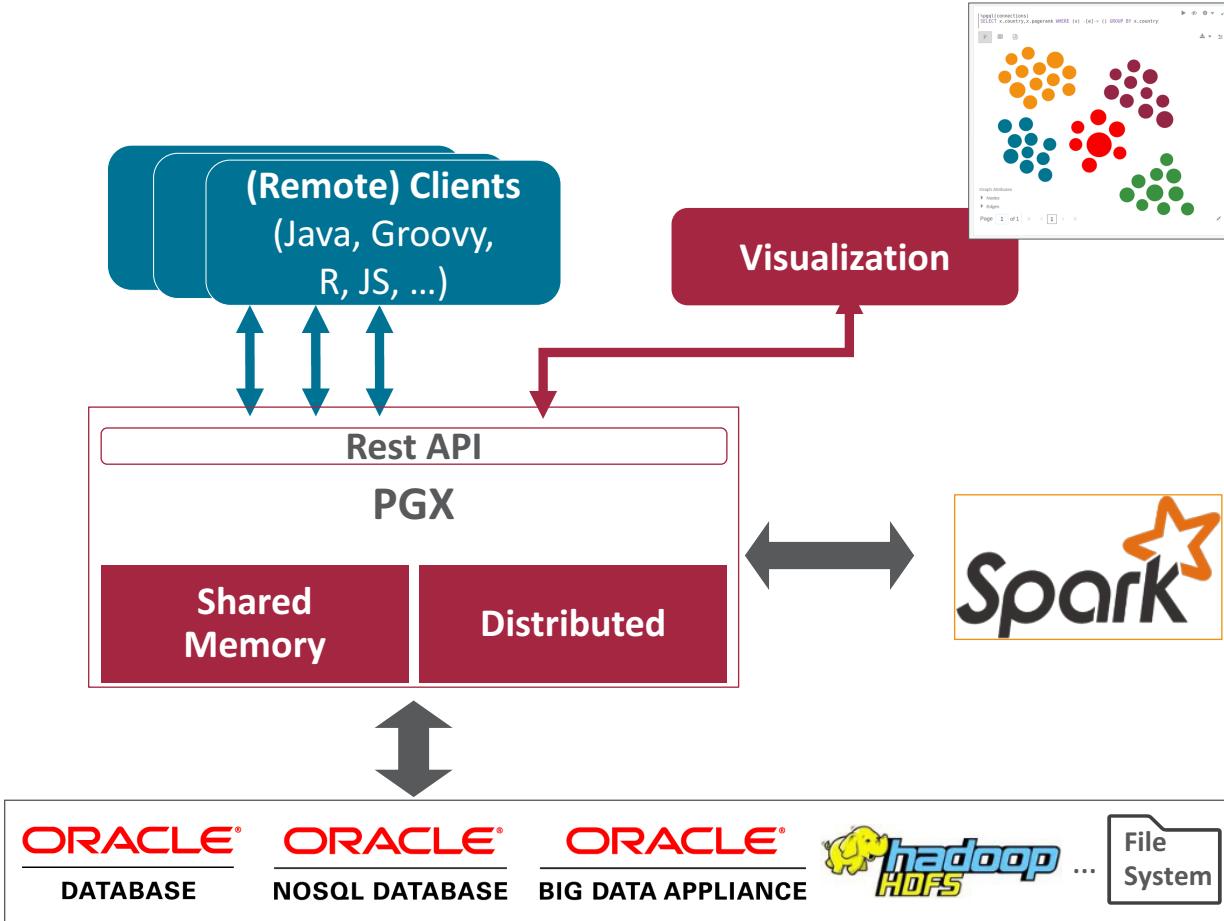
## Graph pattern matching

- Given a **description** of a pattern
- Find every subgraph which **matches** the pattern



Domain-specific language: PGQL

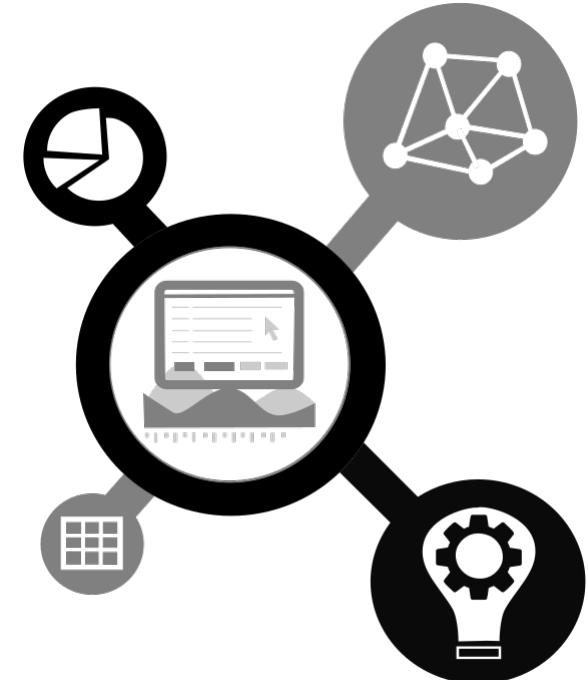
# Architecture Overview



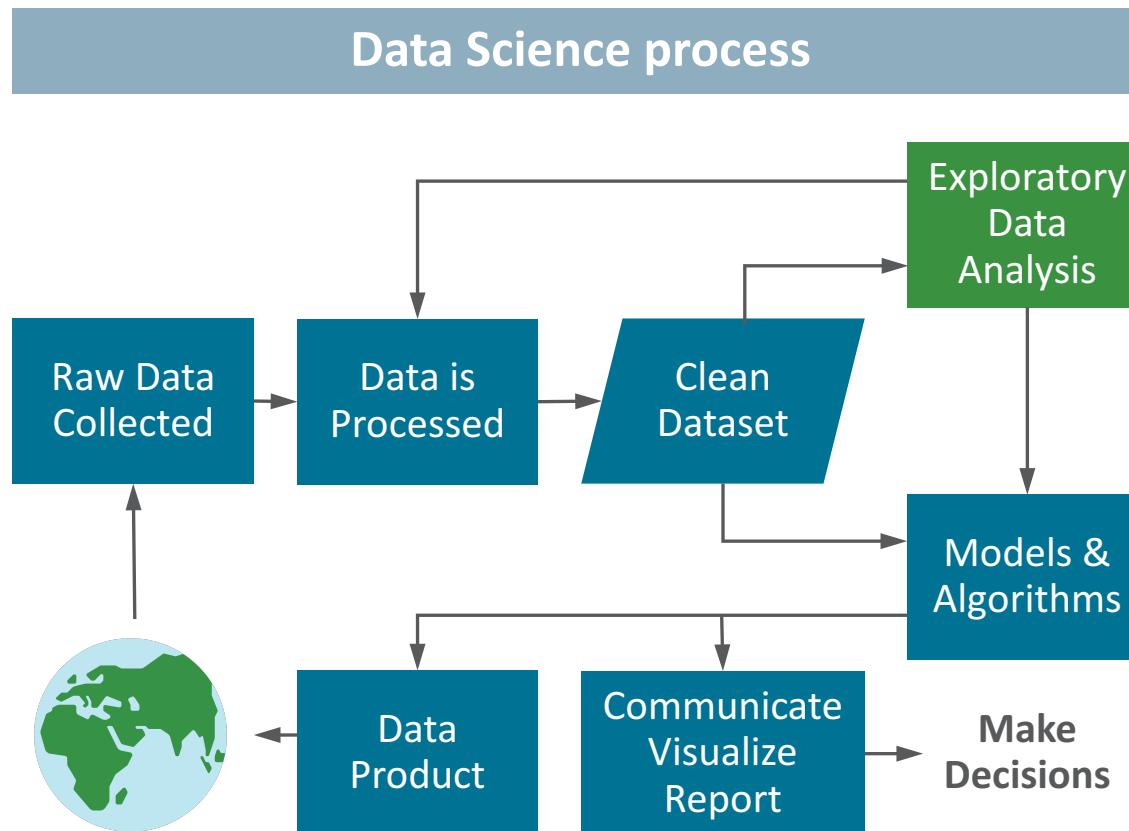
- Integrated with various Oracle and open source data storage technologies
- Interoperates with open-source frameworks, e.g. Apache Spark
- In-memory engines for single-machine or distributed execution
- Multiple language bindings for remote execution via REST
- Notebooks and visualization

# Oracle Labs Data Studio

Data Science with Interactive and Collaborative Notebooks



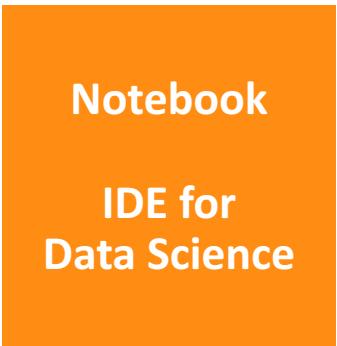
# Data Science: What tools are typically required?



"Doing Data Science", Cathy O'Neil and Rachel Schutt, 2013

## Frontend tools

- Quick, interactive execution
- Visualization of results
- Support for modern programming languages
- Remote execution to Cloud



## Backend tools

- Scalable data manipulation
- Efficient computational engines
- Built-in algorithms and methodologies
- Customization via end-user software engineering

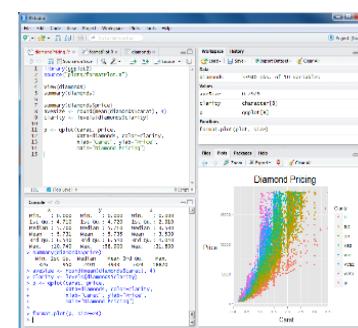
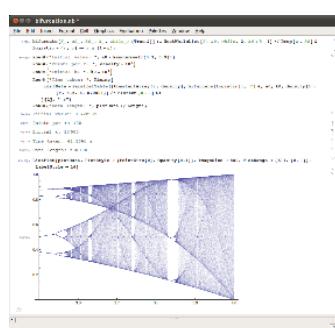
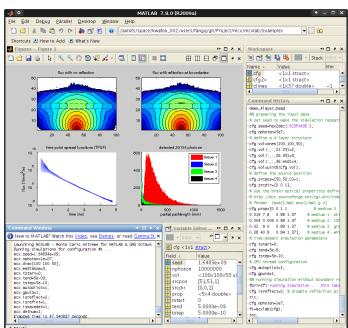


# Rise of Notebook Technology

## Early tools



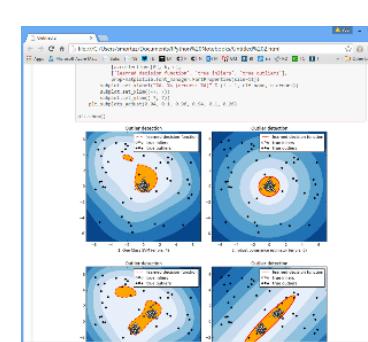
- Single, dedicated programming language
- Stand-alone desktop applications
- Local execution
- REPL-style execution, embedded visualization



## Recent trends



- Multiple languages, general-purpose languages
- Browser-based
- Execution in Cloud, enables collaborative editing



# Oracle Labs Data Studio

## Notebook technology from Oracle Labs

- Built using Oracle JET
- Extensible
- Focus on enterprise
- Integration with Oracle technologies:
  - Graph analysis
  - Graph visualization
  - Advanced polyglot support (Graal)
  - In-database execution (Walnut)

The screenshot shows the Oracle Labs Data Studio interface. At the top, there's a navigation bar with 'Oracle Labs Data Studio' and a 'Create Notebook' button. Below the bar, there are two main sections:

- Default Graph:** This section contains a code editor with the following query:

```
#pgql(graph)
SELECT n,e,m WHERE (n) -[e]-> (m) LIMIT 15
```

A graph visualization below the code shows a cluster of nodes connected by edges. Nodes are colored in shades of red, blue, and purple, likely representing different degrees or properties.
- Interpolate node colors/sizes based on properties:** This section also contains a code editor with a similar query:

```
#pgql(graph)
SELECT n,e,m WHERE (n) -[e]-> (m) LIMIT 25
```

A graph visualization here shows a more complex network. Nodes are large green circles, and the size of each node appears to be proportional to its properties, with a central large blue node connected to several smaller green nodes.

# Integrated Cloud Applications & Platform Services

**ORACLE®**