
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

#1 **ENG. AND CONSTRUCTION**
9 out of 10 top companies

#1 **FINANCIAL SERVICES**
20 out of 20 top banks

**HIGH TECHNOLOGY**
20 out of 20 top companies

#1 **INSURANCE**
20 out of 20 top insurers

**MANUFACTURING**
20 out of 20 top companies

#1 **MEDICAL DEVICES**
20 out of 20 top companies

**OIL AND GAS**
20 out of 20 top companies

#1 **PHARMACEUTICALS**
20 out of 20 top companies

#1 **PUBLIC SECTOR**
20 out of 20 top governments

#1 **RETAIL**
20 out of 20 top companies

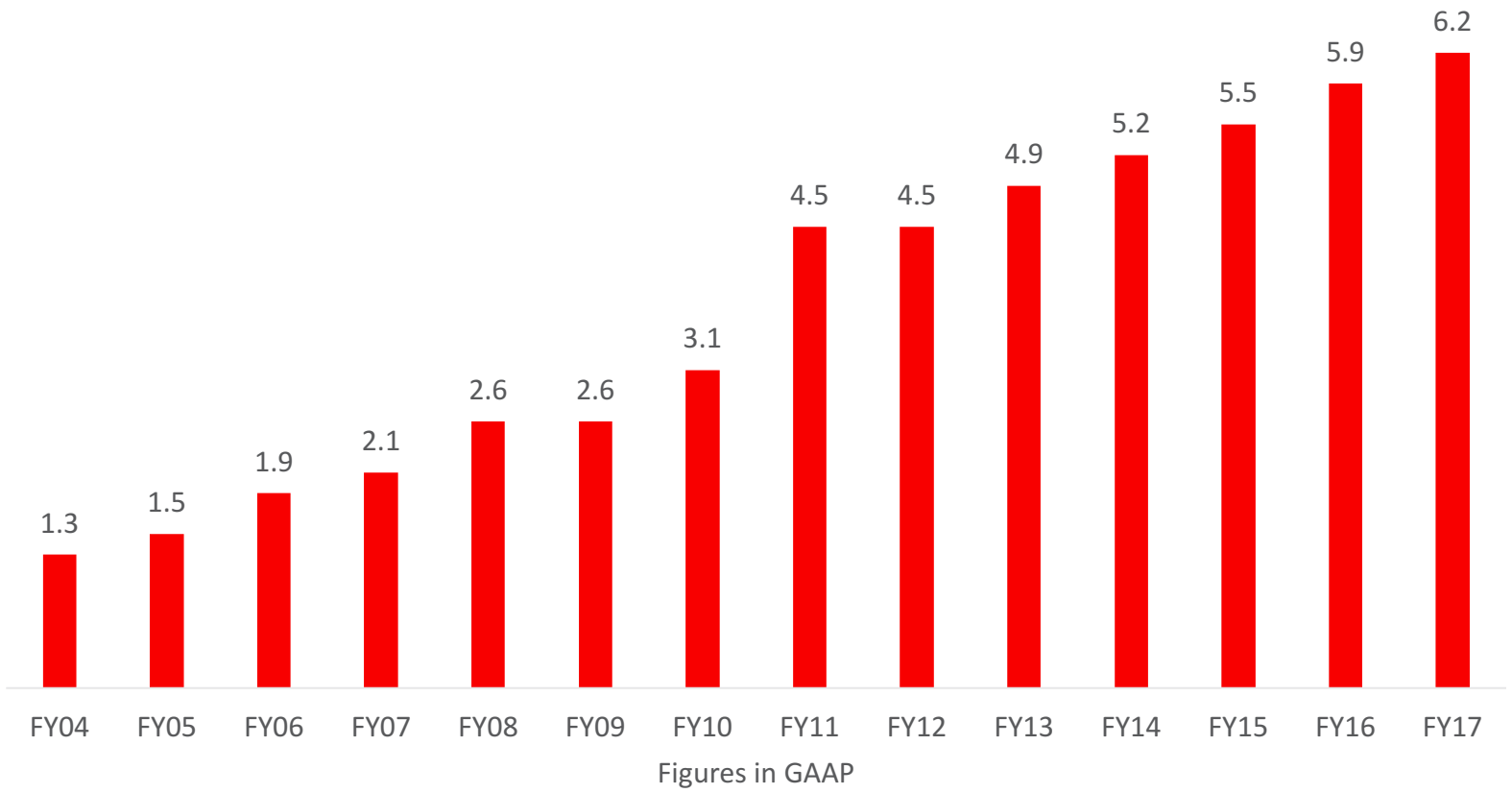
#1 **SUPPLY CHAINS**
20 out of 20 top companies

#1 **TELECOMMUNICATION**
20 out of 20 top companies

**TRAVEL AND TRANSPORT**
20 out of 20 top airlines

#1 **UTILITIES**
10 out of 10 top companies

# Investment in Research and Development




**MORE THAN $51 BILLIONS SINCE 2004**

ORACLE LABS

"The Mission of Oracle Labs is straightforward:

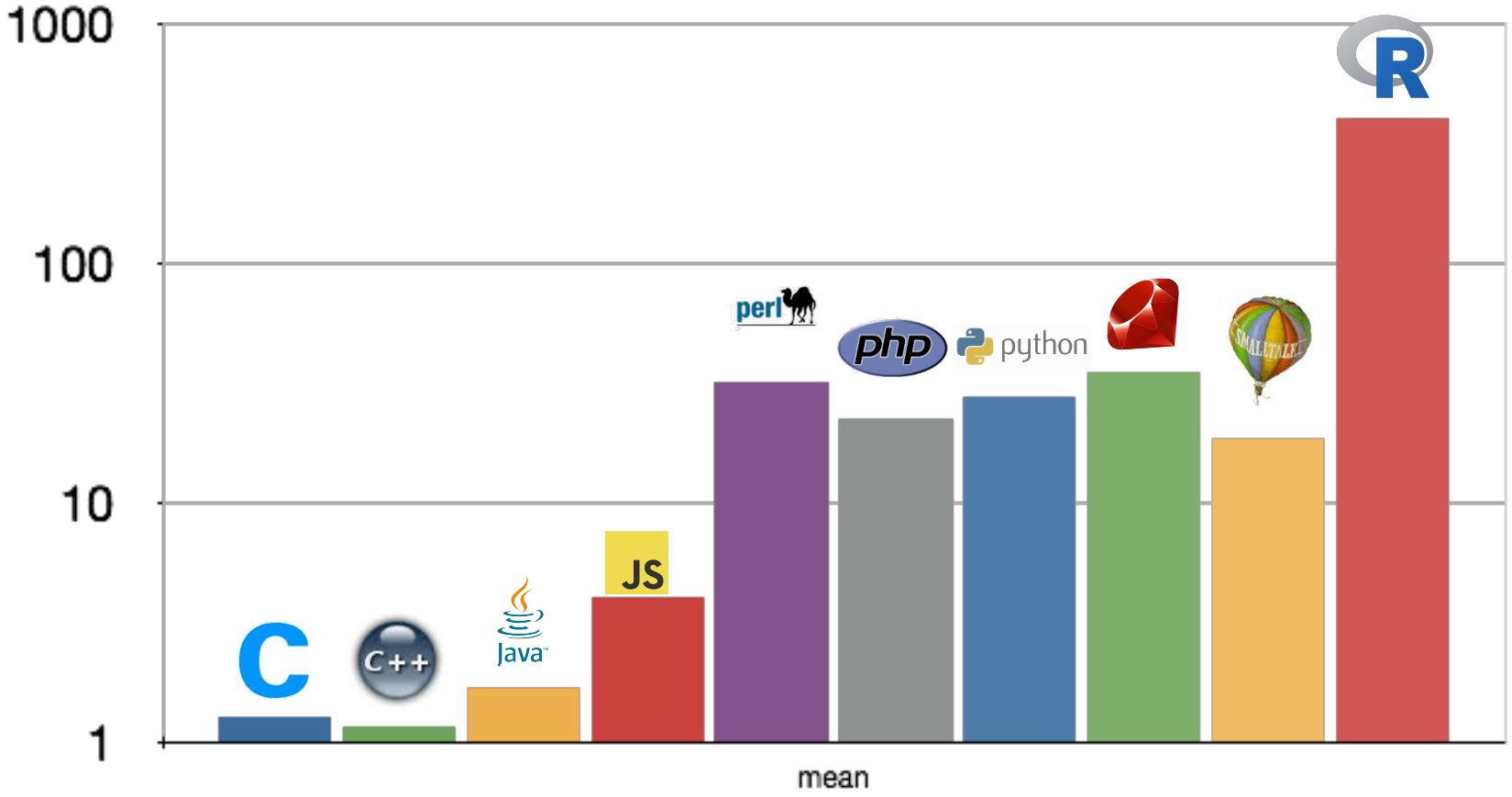
**Identify**, **explore**, and **transfer** *new technologies that have the potential to substantially improve Oracle's business*."

– 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

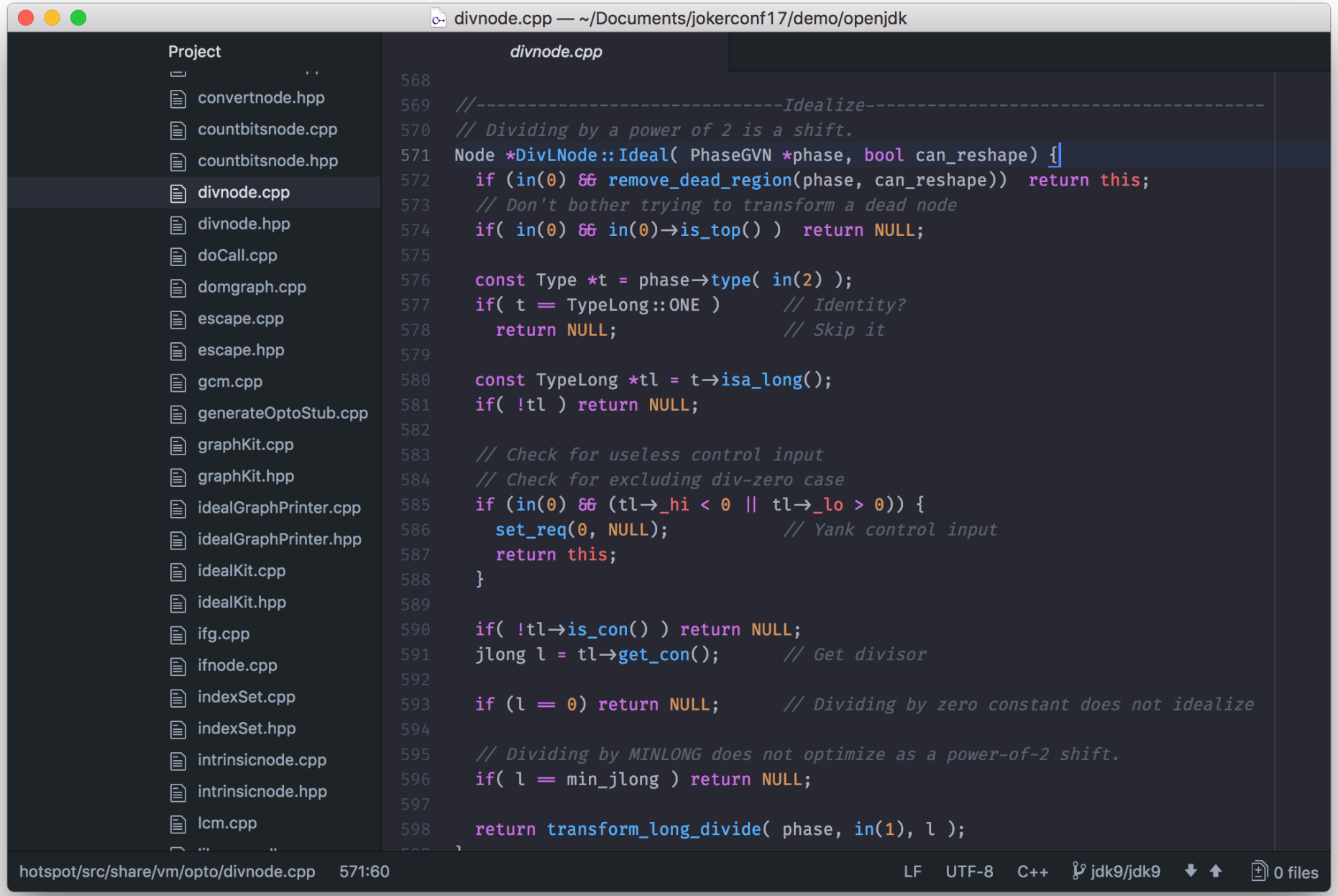
## a Java JIT compiler written in Java

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:

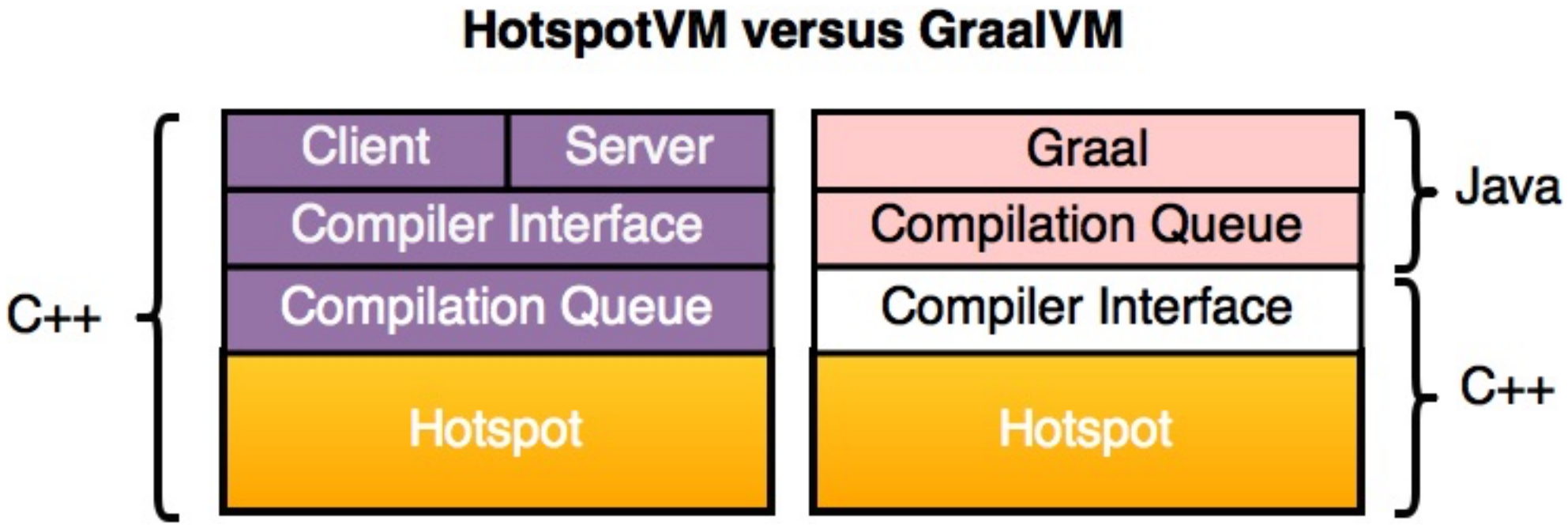
– http://chrisseaton.com/rubytruffle/jokerconf17/

```
class Demo {
  public static void main(String[] args) {
    while (true) {
      workload(14, 2);
    }
  }

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

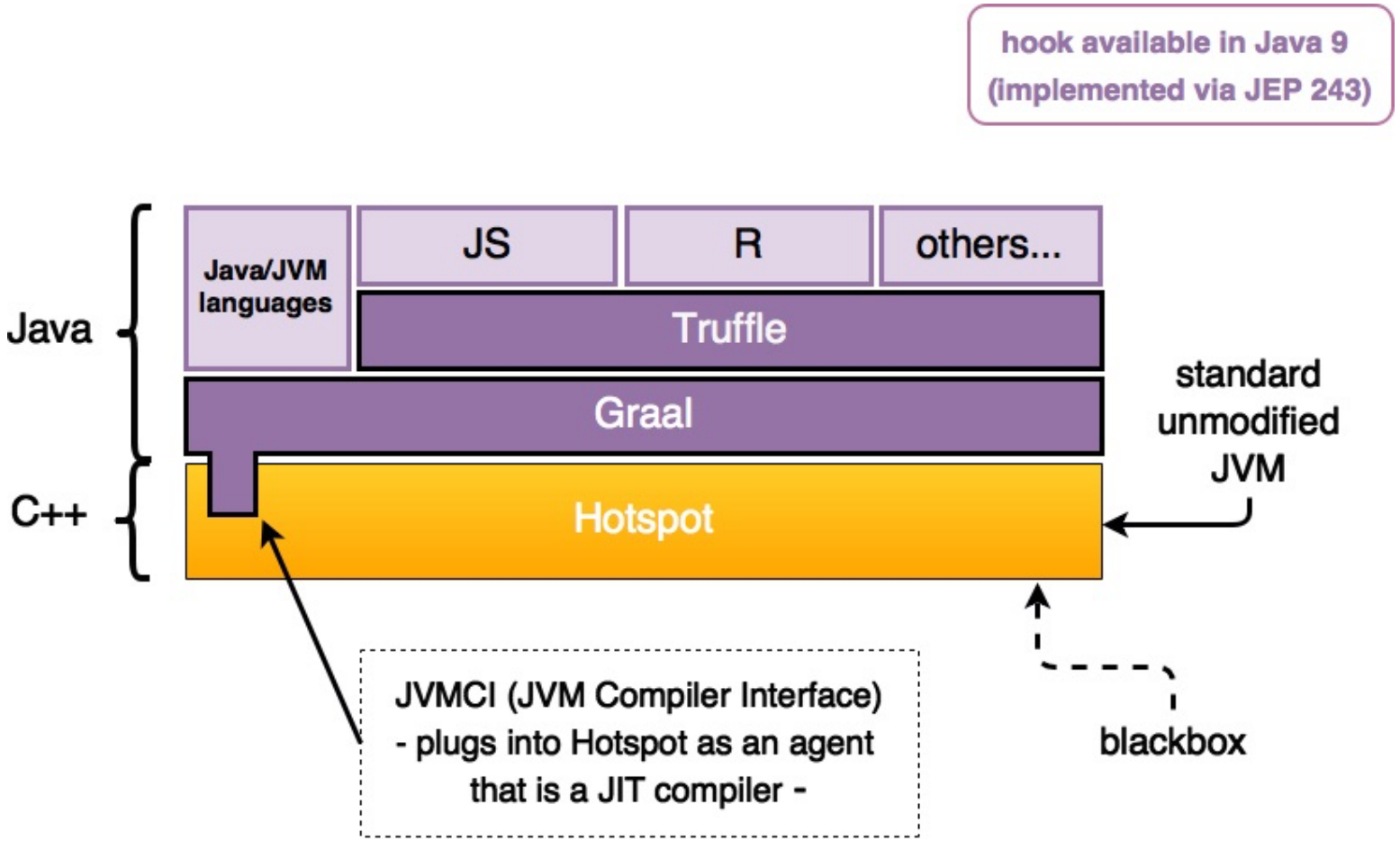
```
$ javac Demo.java
$ java \
  -XX:+PrintCompilation \
  -XX:CompileOnly=Demo::workload \
  Demo
...
    113    1       3       Demo::workload (4 bytes)
...
```



# Graal VM



# Graal Compiler

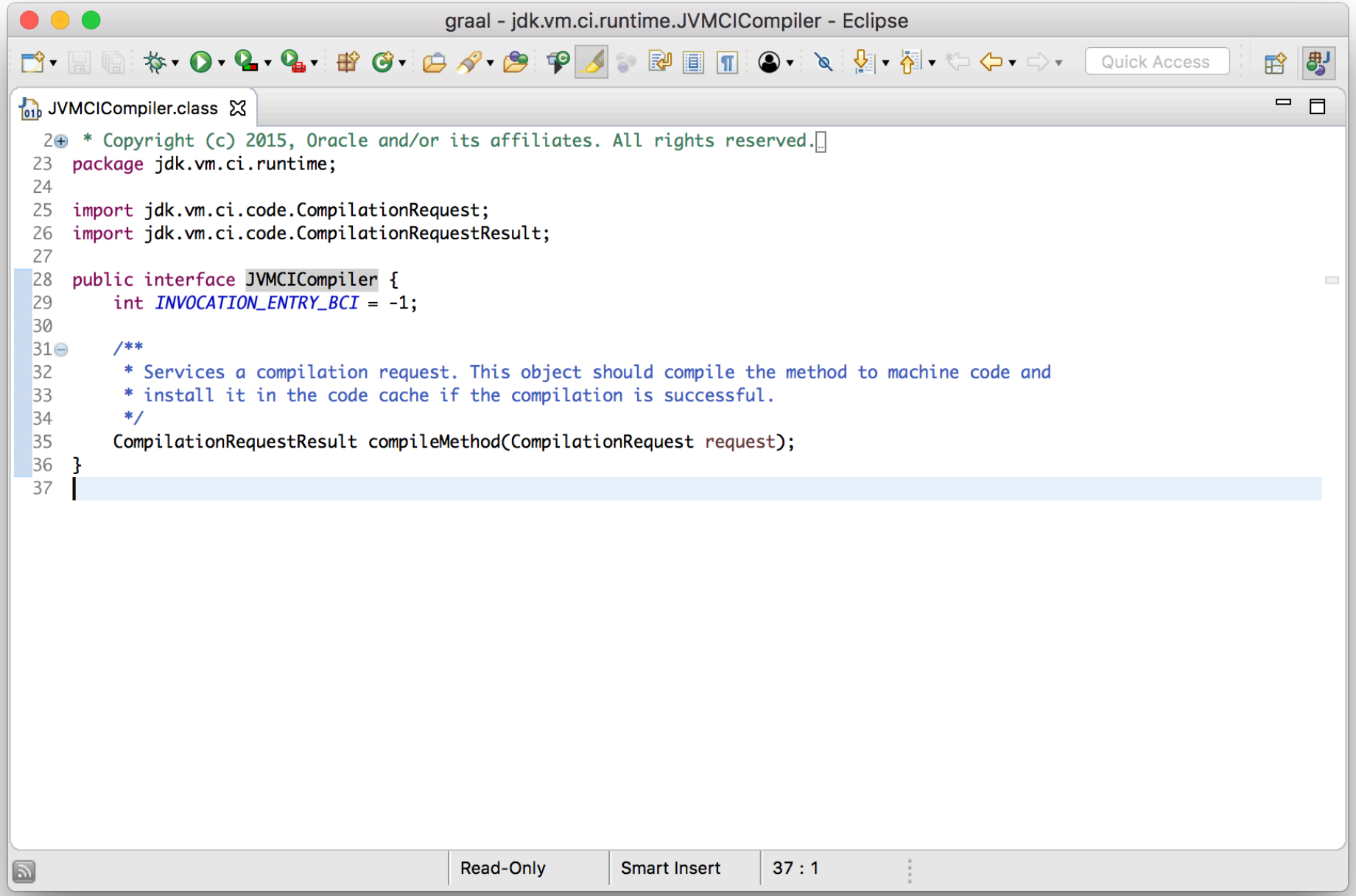


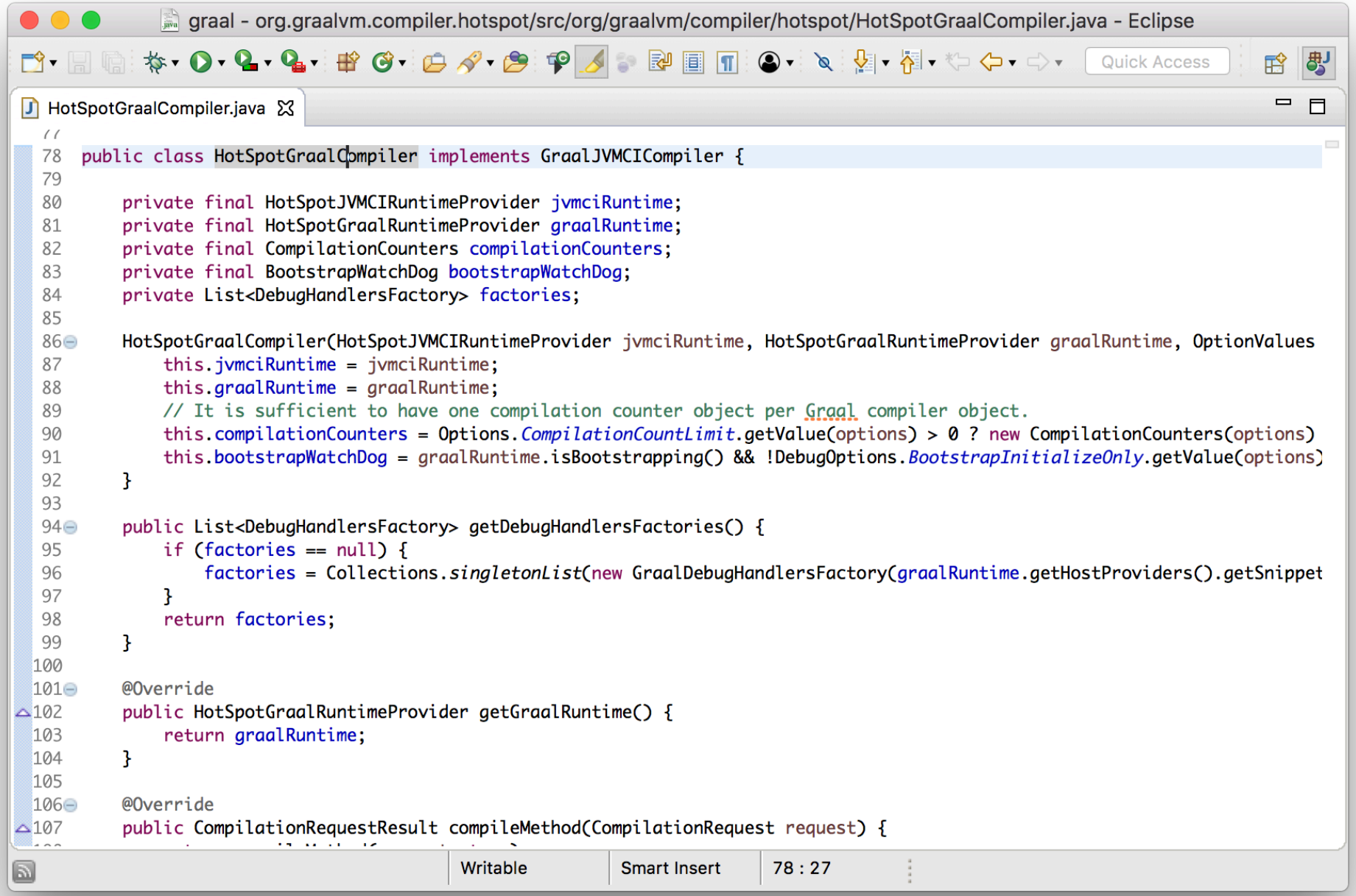
```
$ java \
  --module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/..... \
  --upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler \
  -XX:-TieredCompilation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

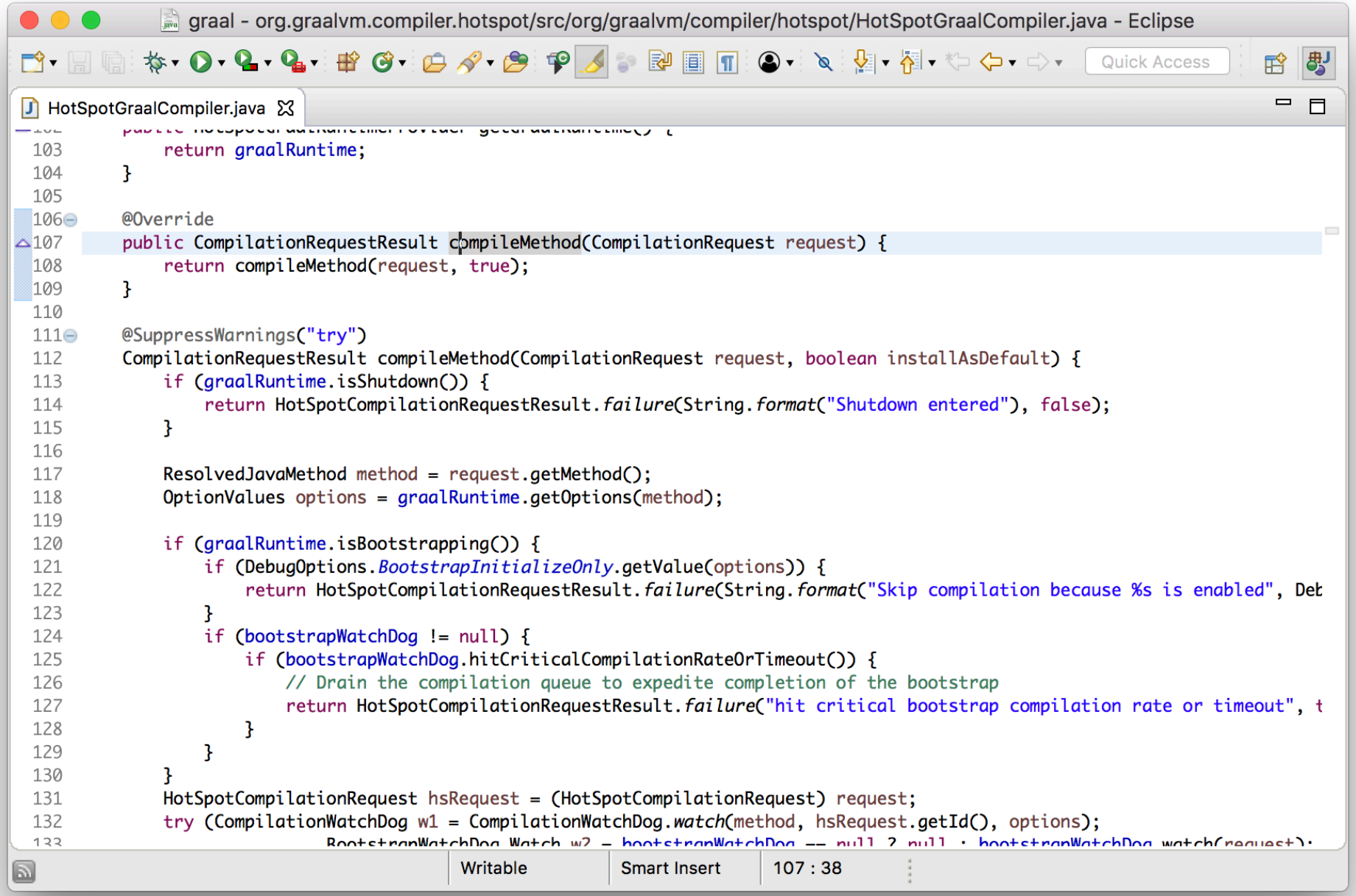
```
interface JVMCICompiler {
    byte[] compileMethod(byte[] bytecode);
}
```

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

interface CompilationRequest {
    JavaMethod getMethod();
}

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

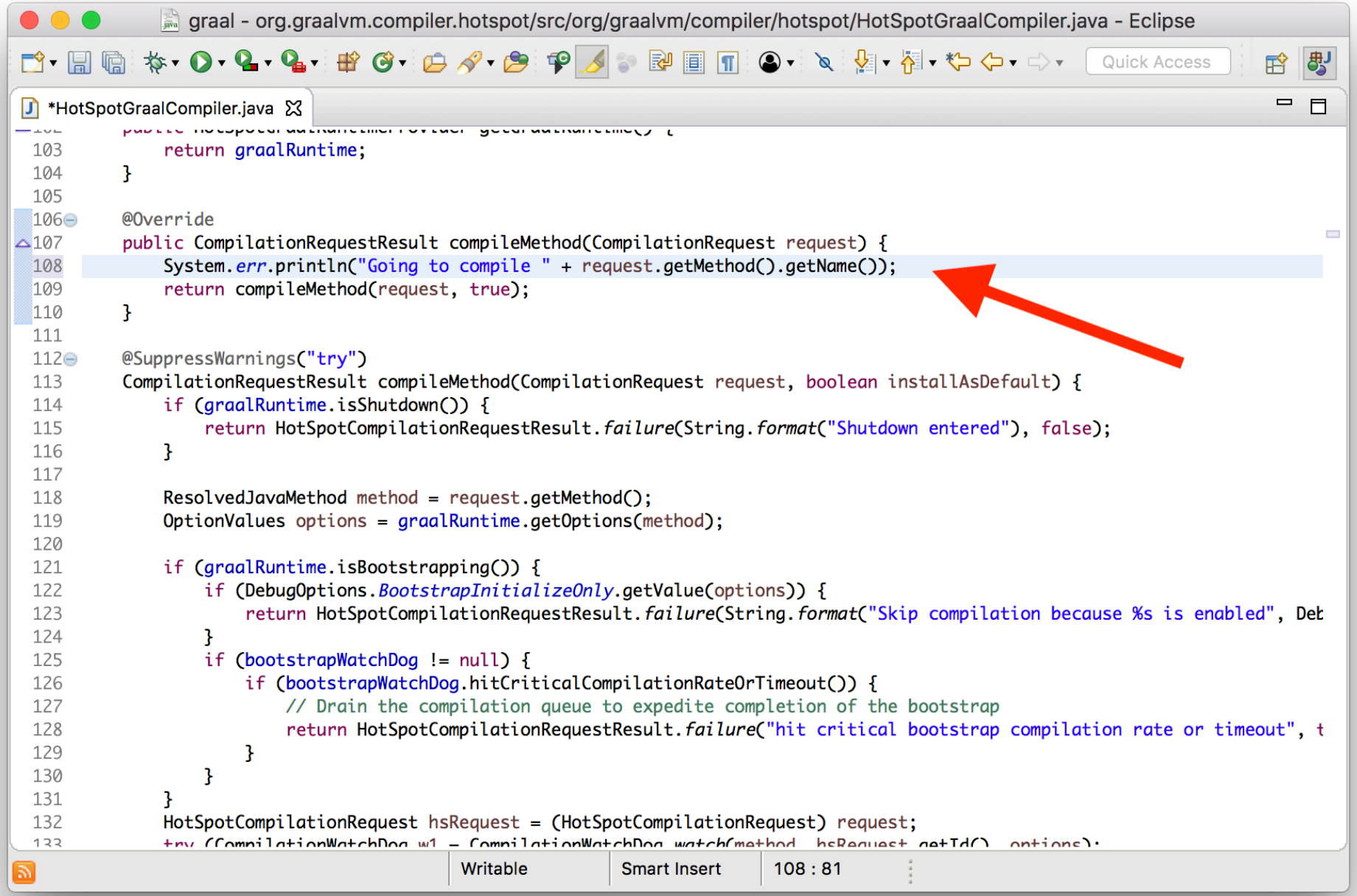
```
HotSpot.installCode(targetCode);
```







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

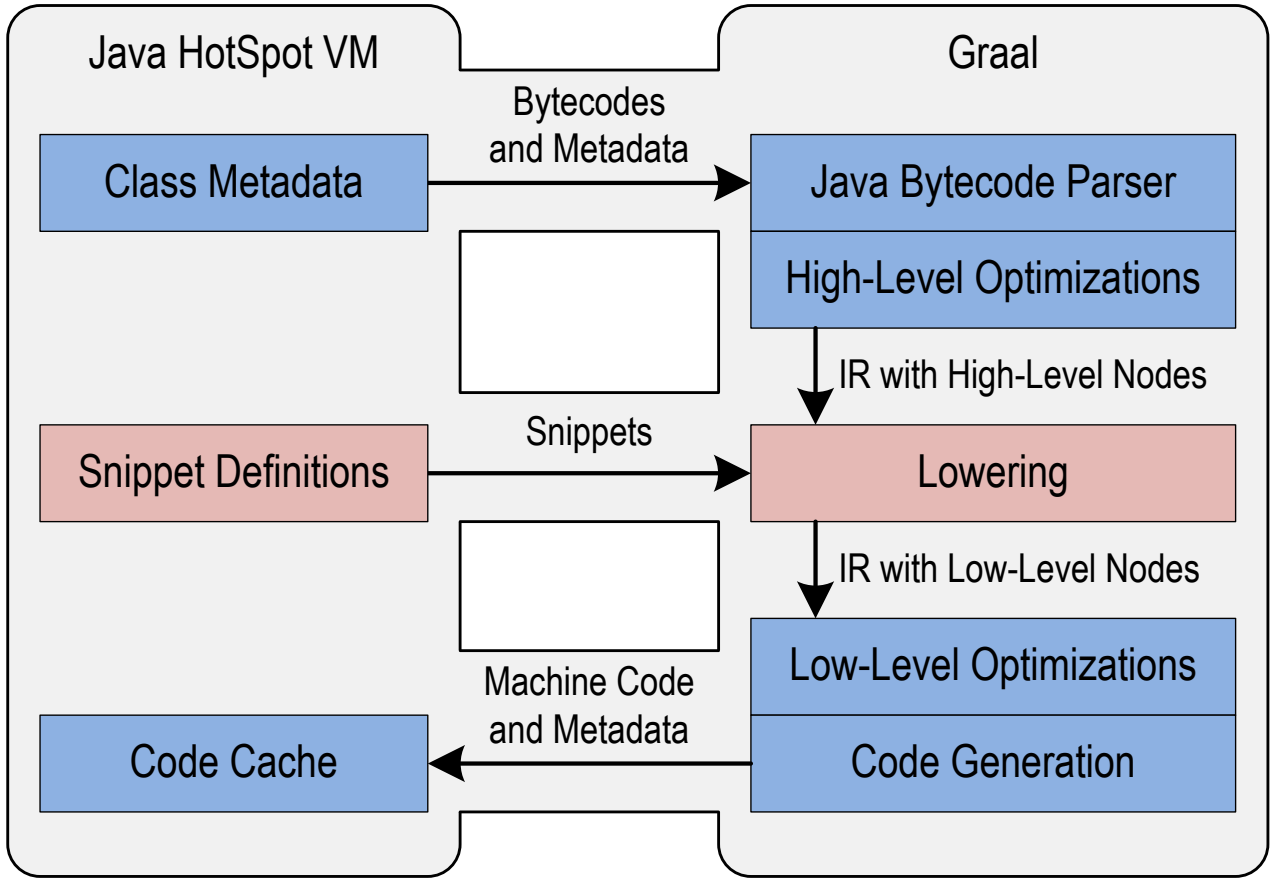


```
$ java \
  --module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/modules/..... \
  --upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler \
  -XX:-TieredCompilation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

# Intermediate Representation

## data flow and control flow graphs

# Graal Compiler

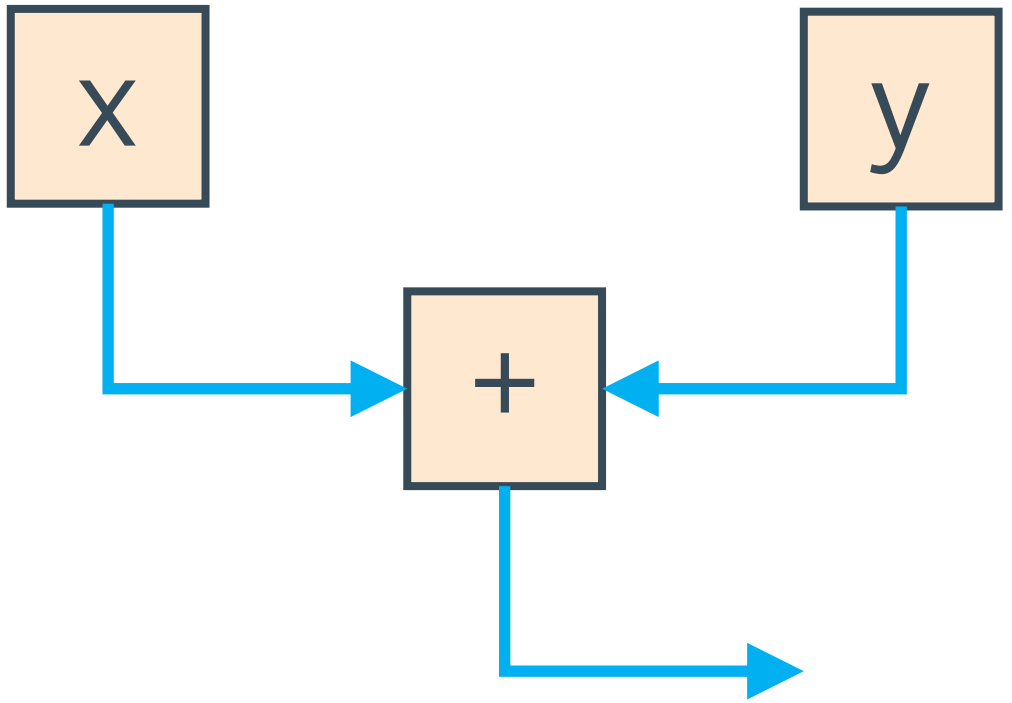


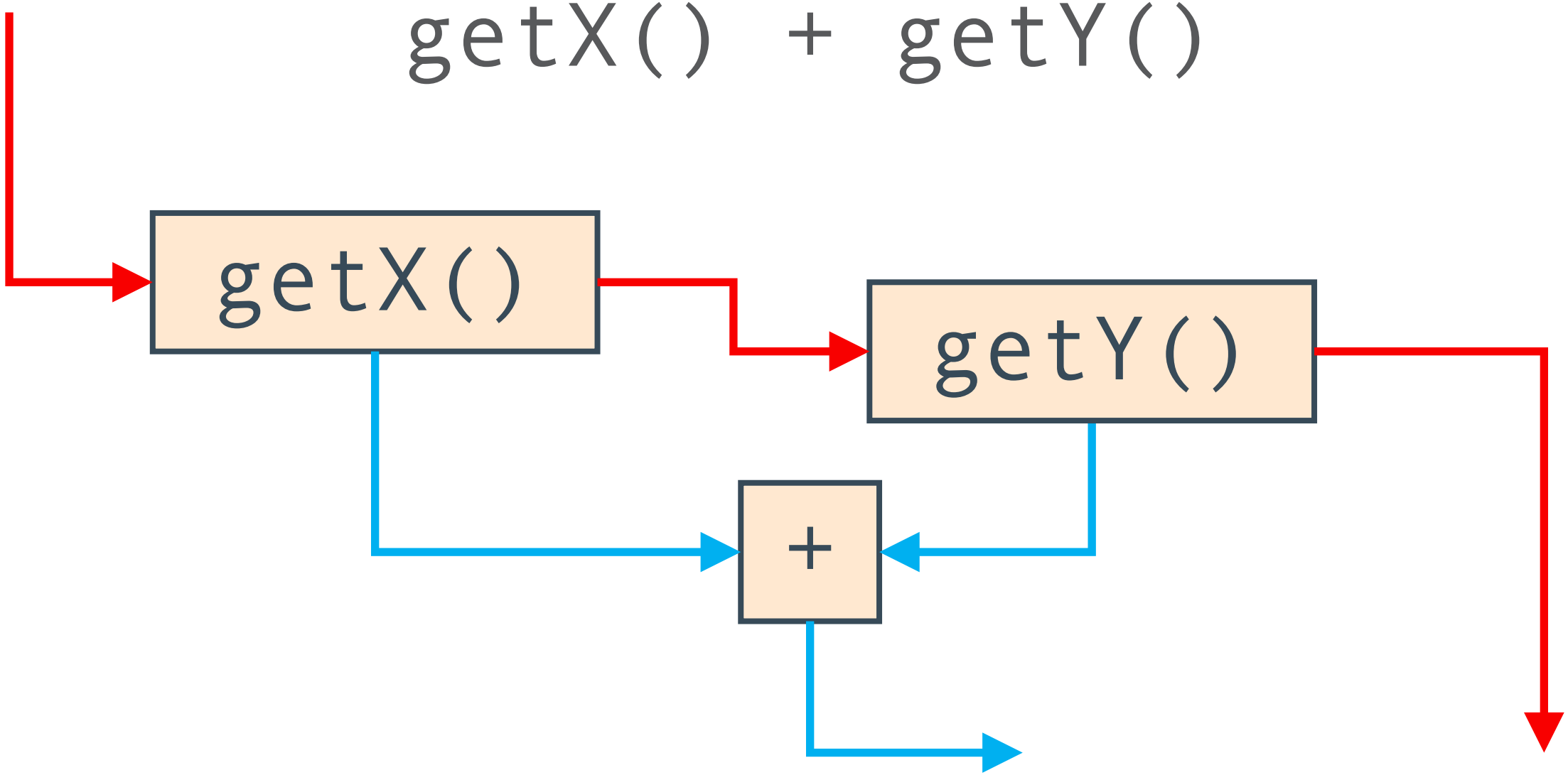
# Static Single Assignment (SSA) Form

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

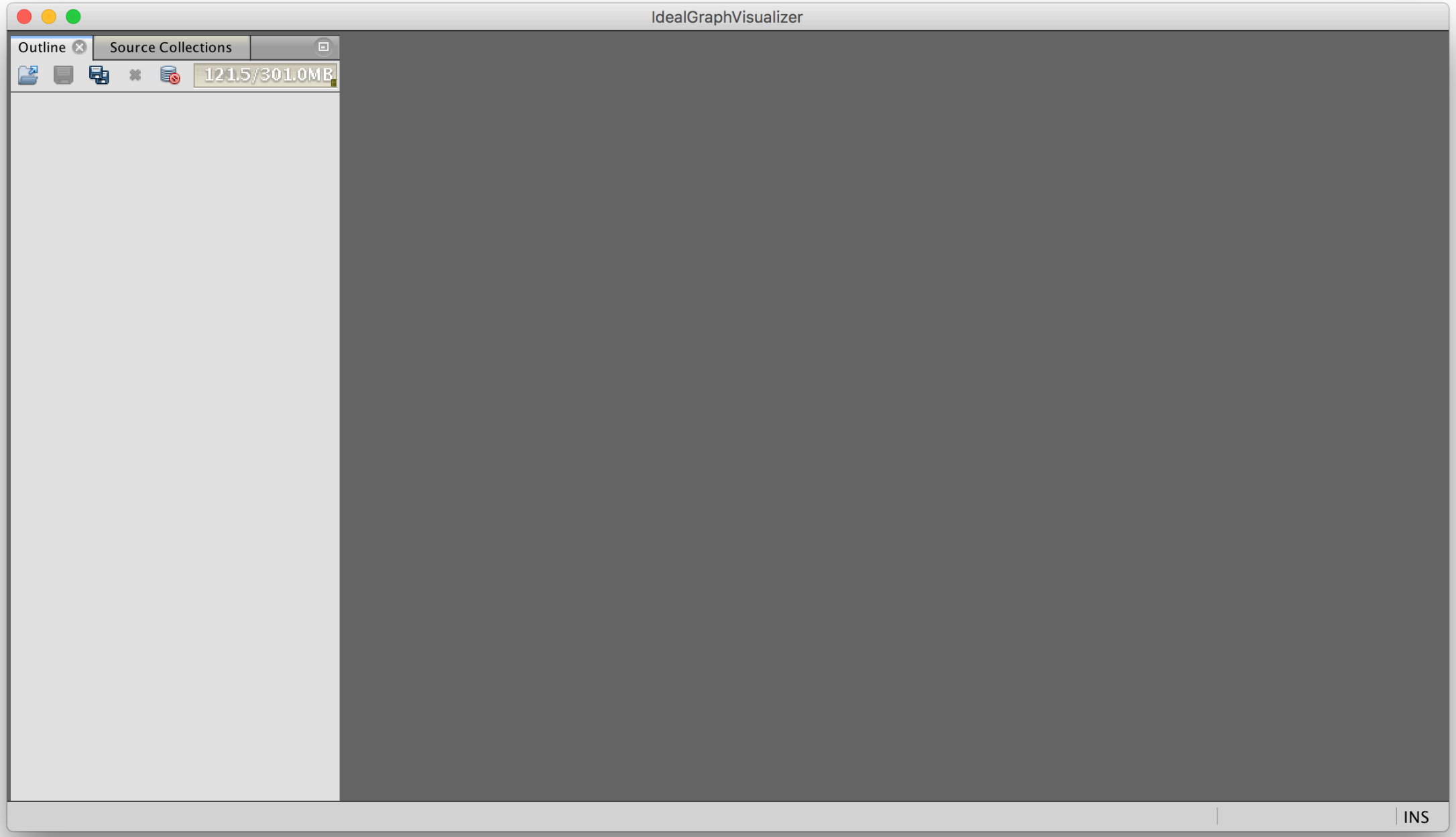
```
if (condition) {
   x1 = value1 + value2;
} else {
   x2 = value2;
}
x3 = phi(x1, x2);
return x3;
```

x + y



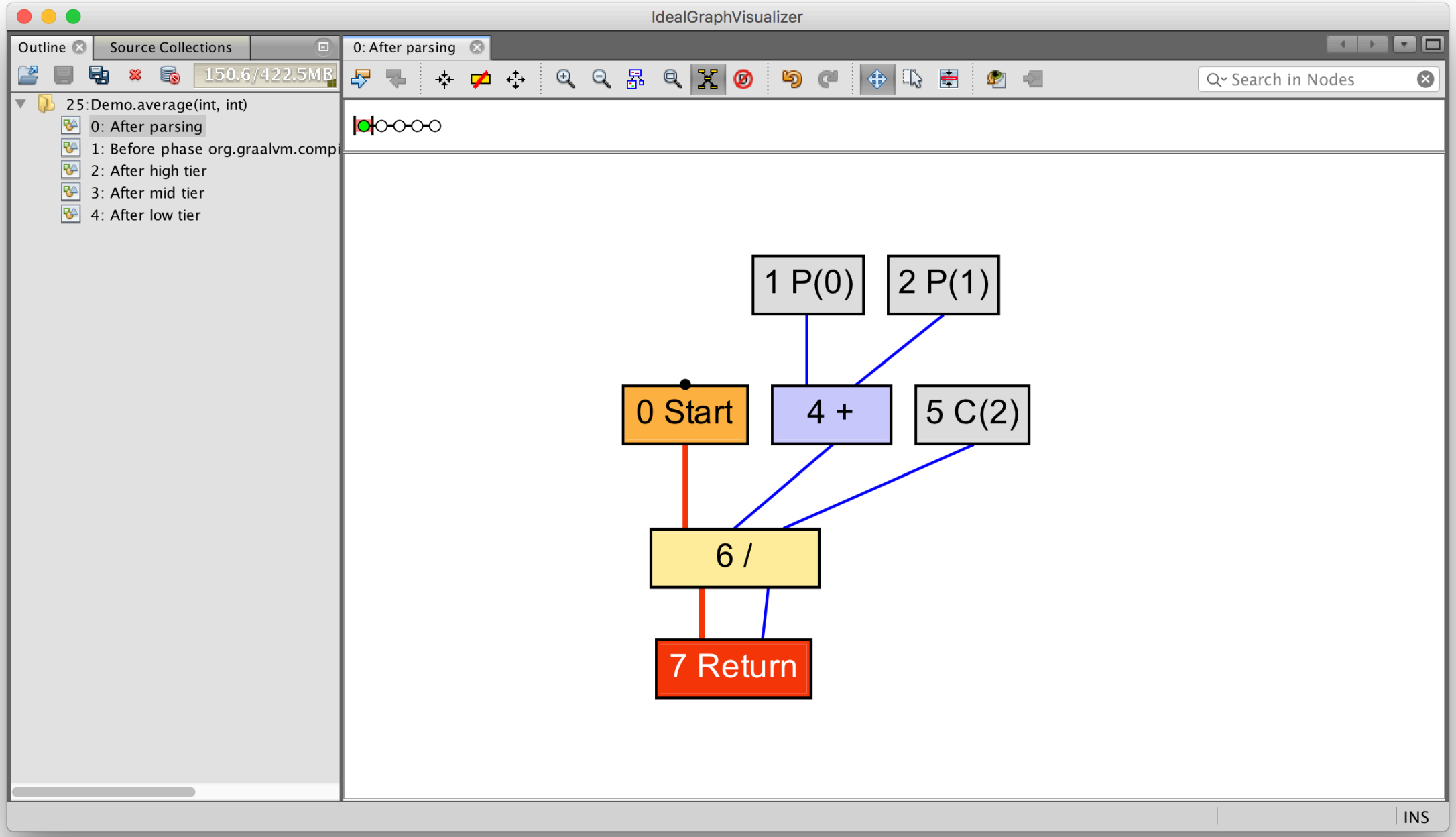


```
mx igv
```

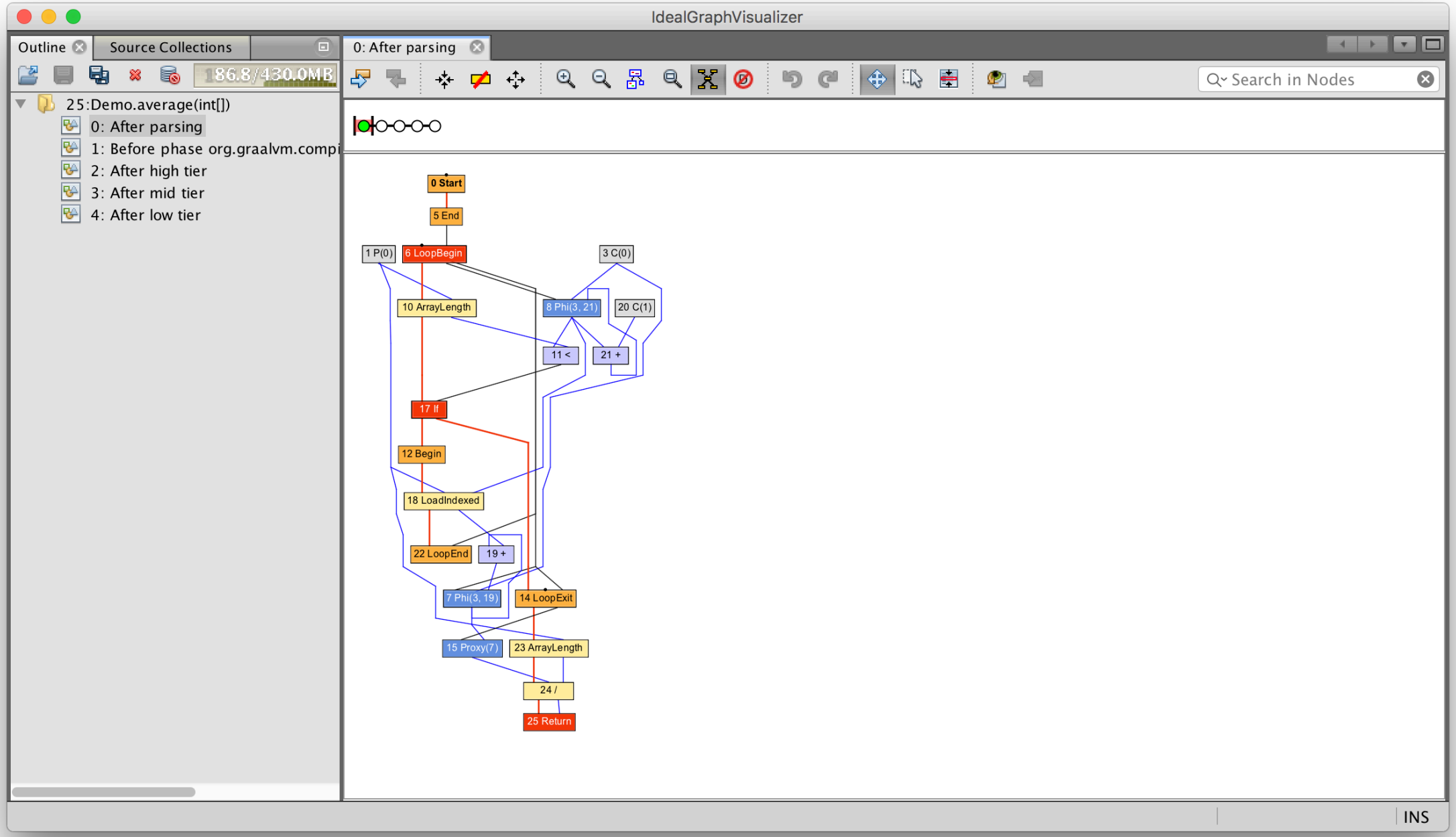


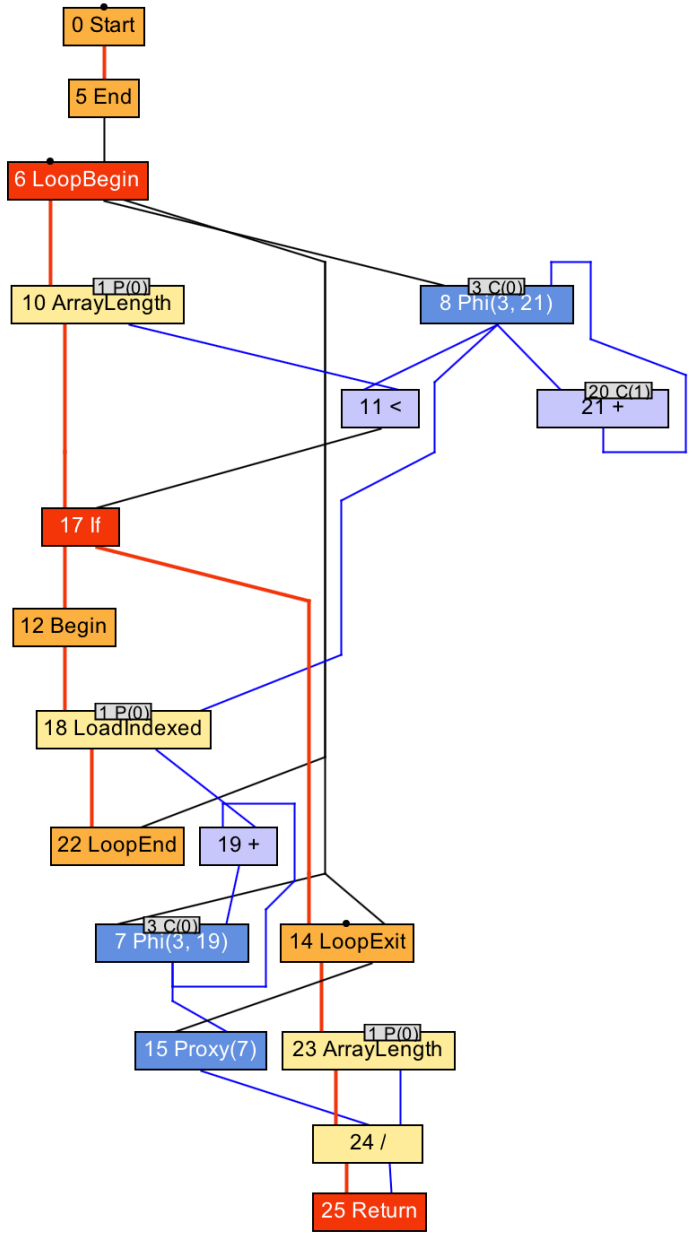
```
-Dgraal.Dump
```

```
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;
}
```





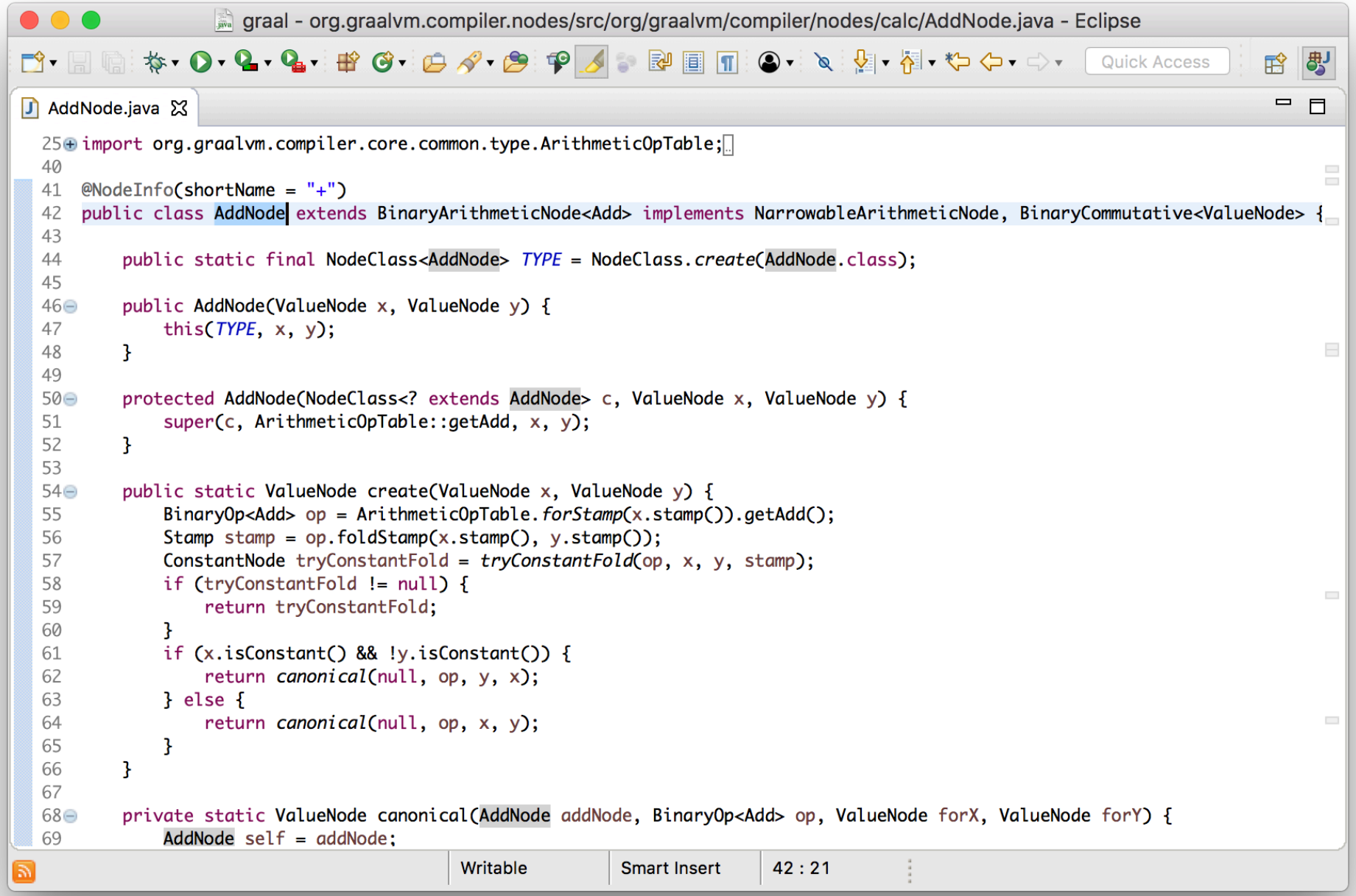
# *Bytecode in*

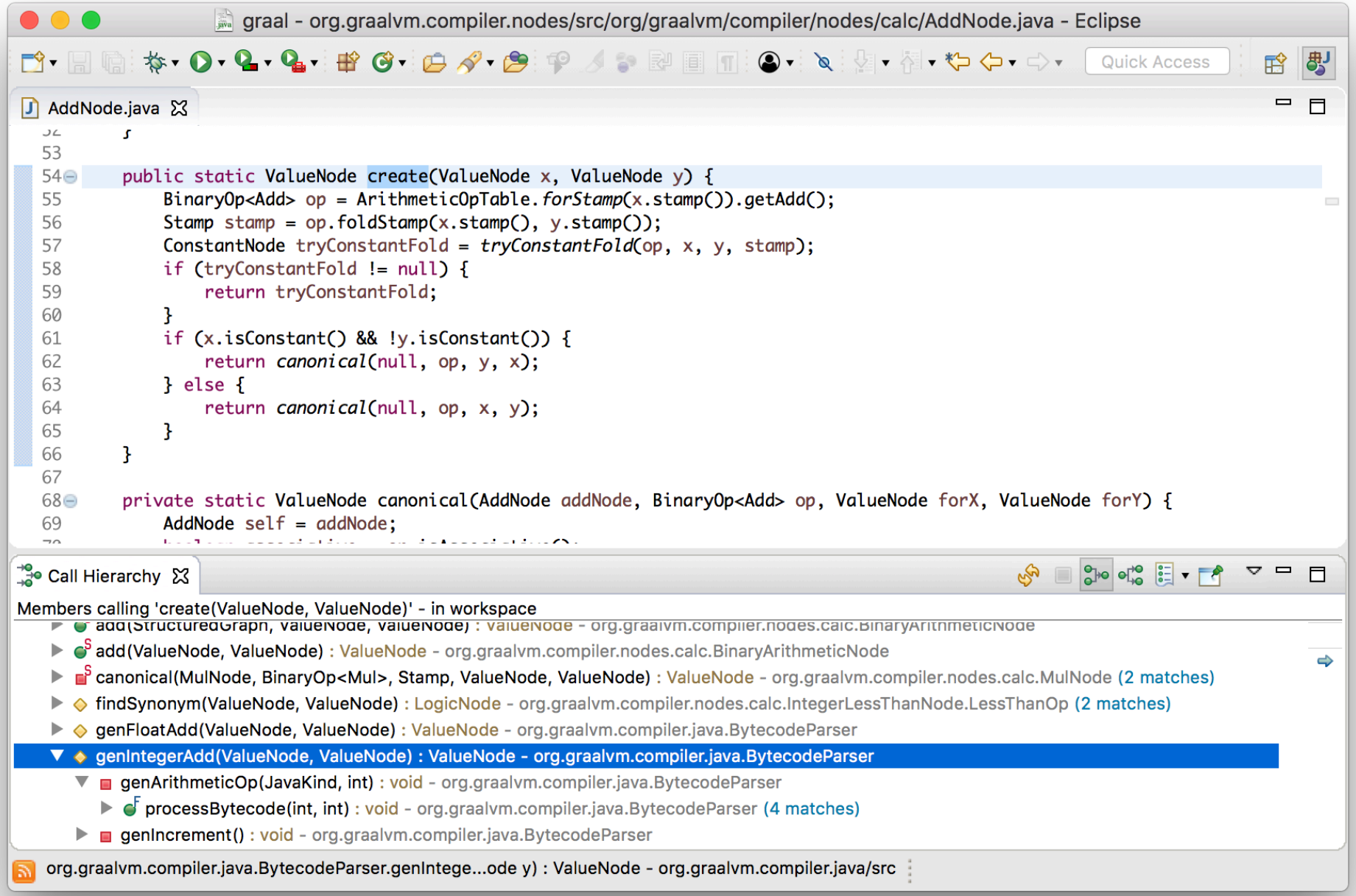
```
int workload(int a, int b) {
   return a + b;
}
```

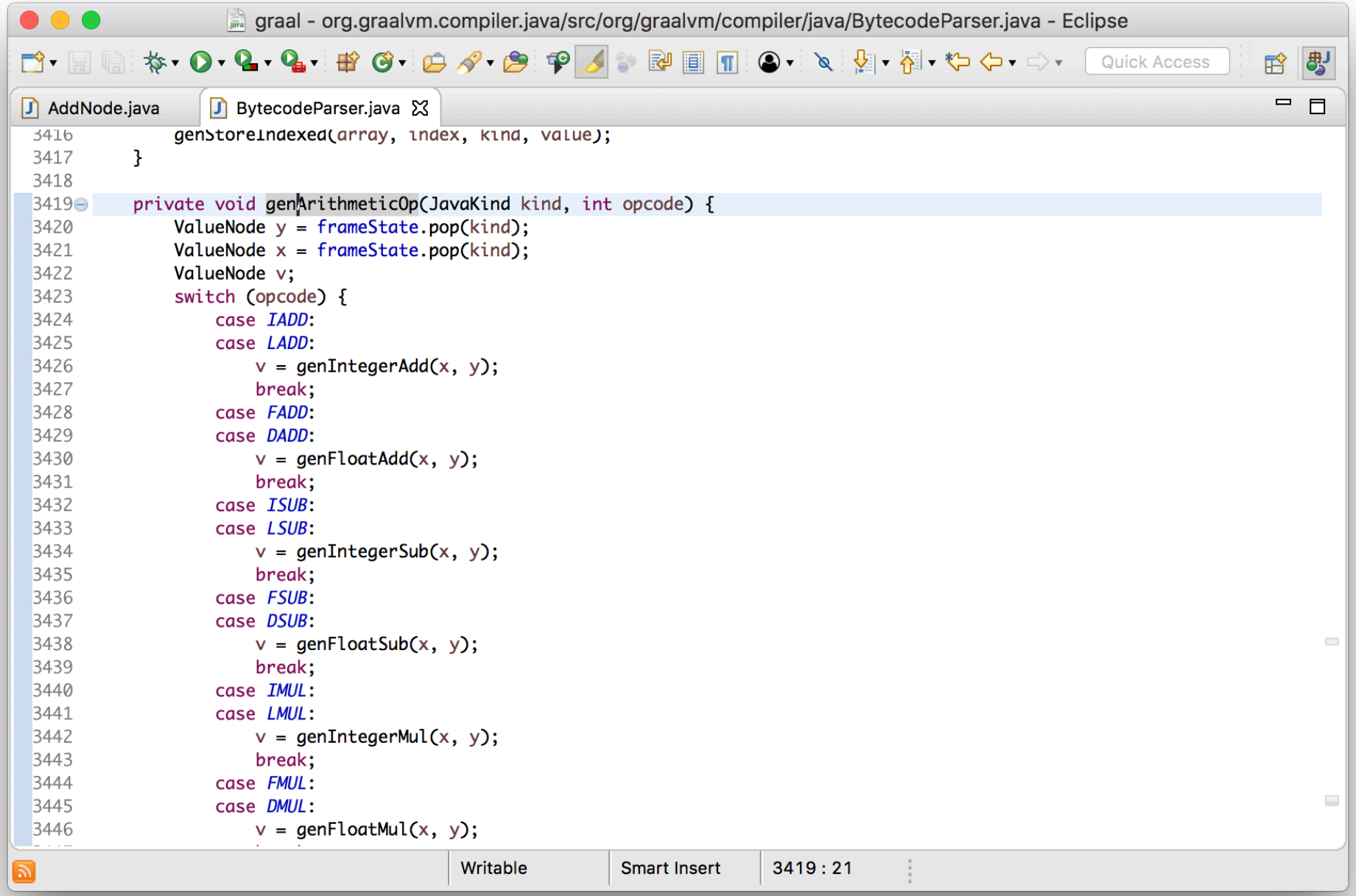
```
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationRequestResult compileMethod(CompilationRequest request) {
    System.err.println(request.getMethod().getName() + " bytecode: "
      + Arrays.toString(request.getMethod().getCode()));
    ...
  }
}
```

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

# *The bytecode parser*







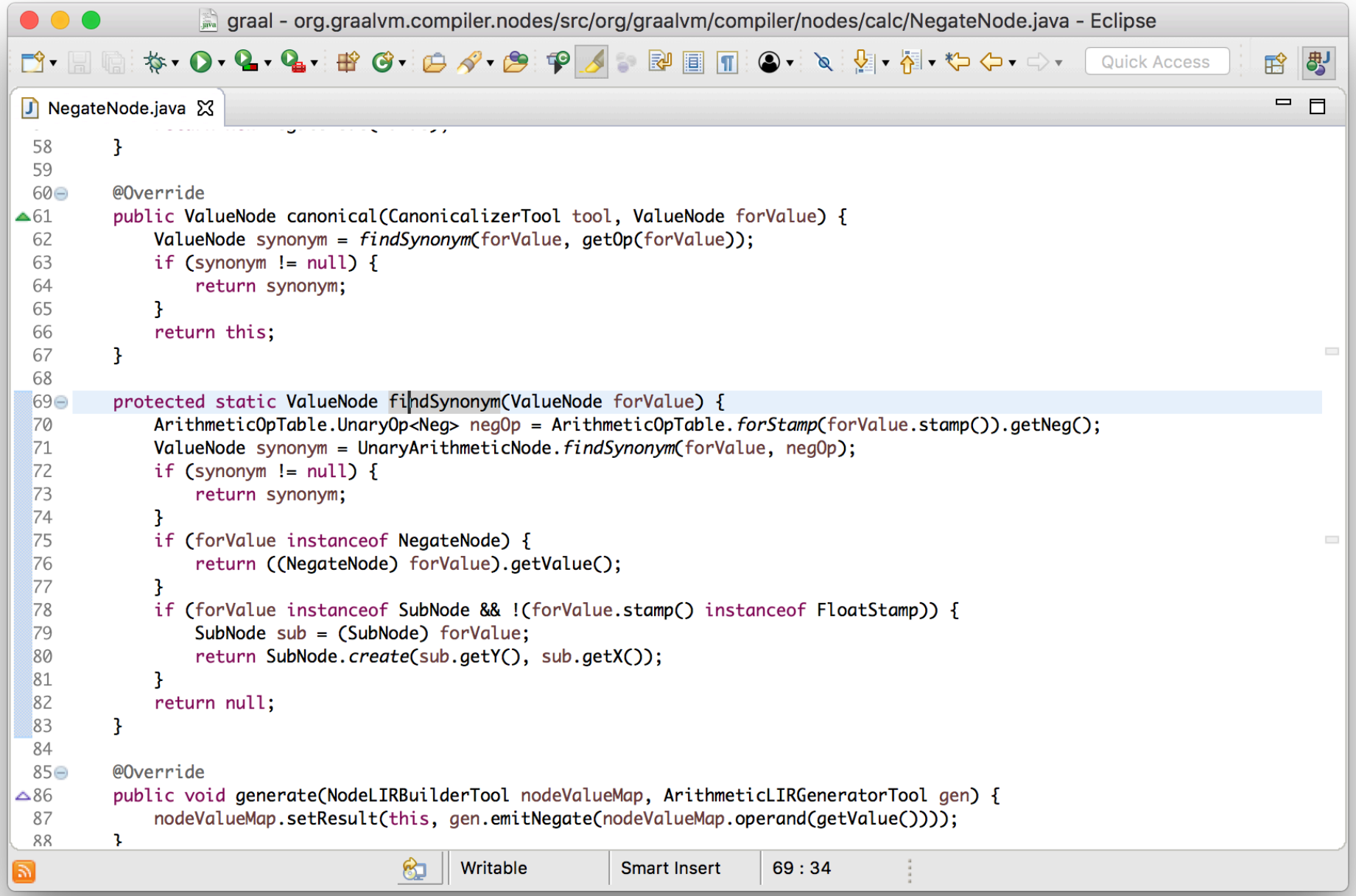
```
private void genArithmeticOp(JavaKind kind, int opcode) {
    ValueNode y = frameState.pop(kind);
    ValueNode x = frameState.pop(kind);
    ValueNode v;
    switch (opcode) {
        ...
        case LADD:
            v = genIntegerAdd(x, y);
            break;
        ...
    }
    frameState.push(kind,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

# *Canonicalisation*

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

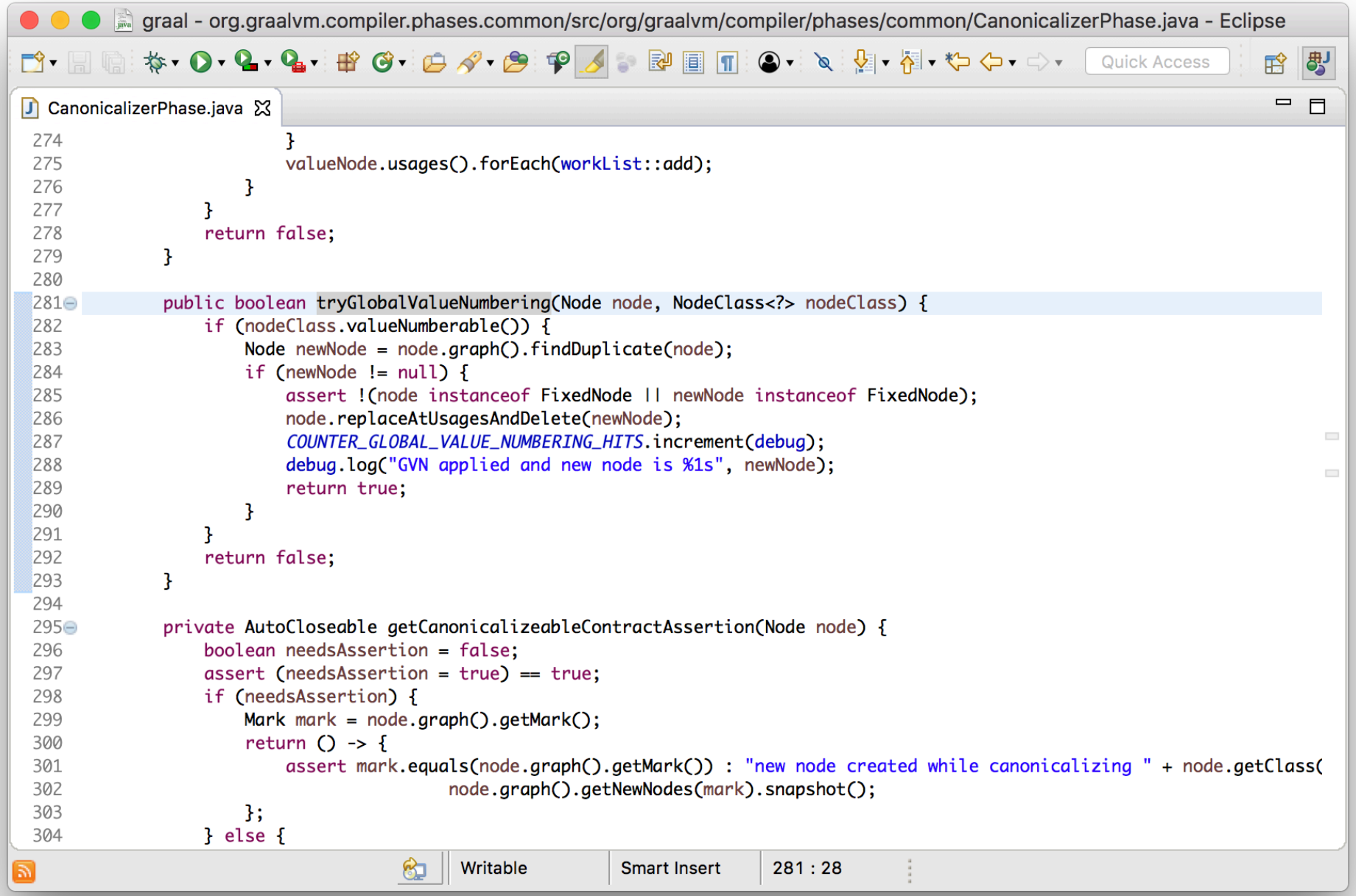
```
interface Node {
  Node canonical();
}
```

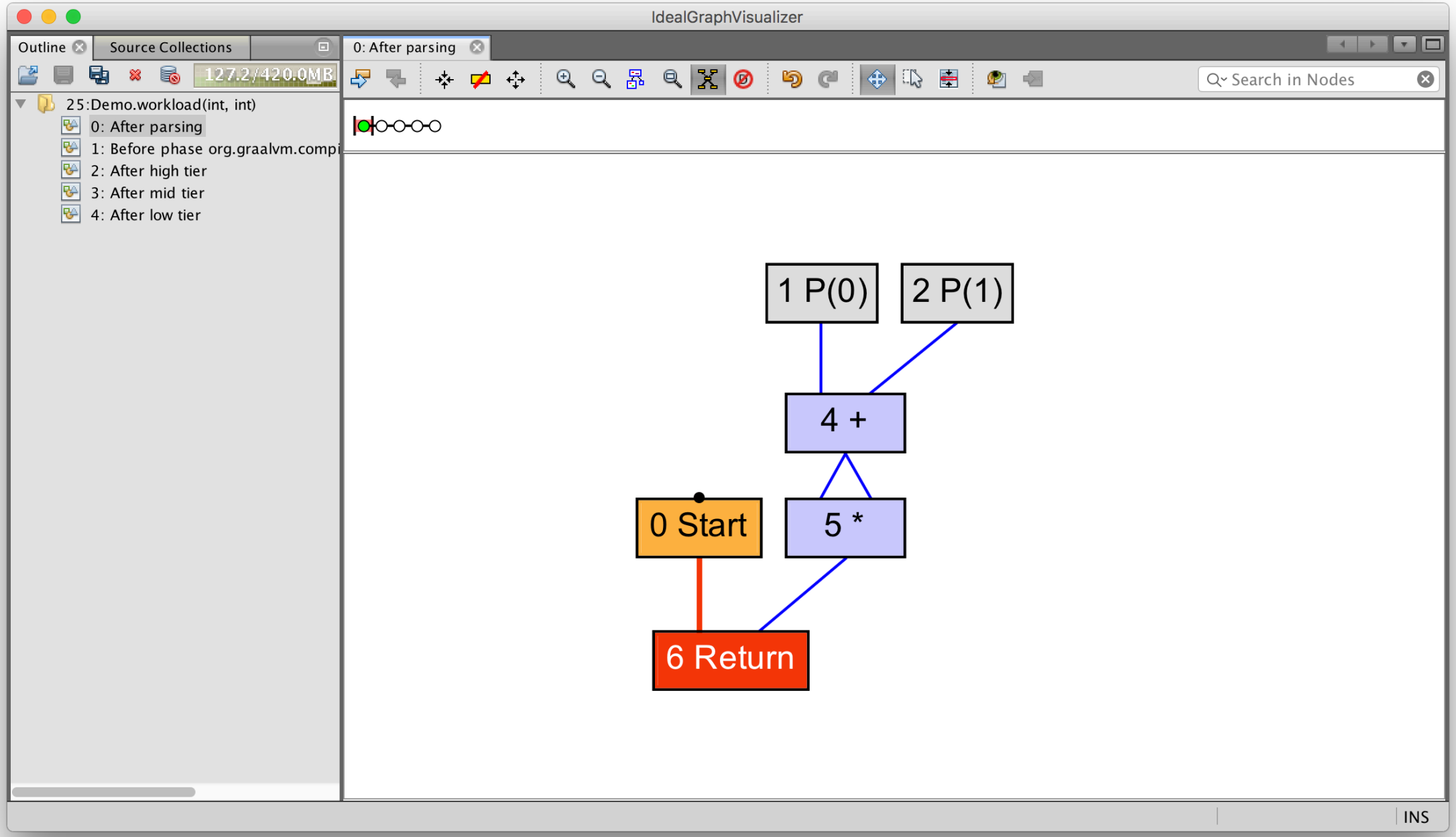
```
class NegateNode implements Node {
  Node canonical() {
    if (value instanceof NegateNode) {
      return ((NegateNode) value).getValue();
    } else {
      return this;
    }
  }
}
```



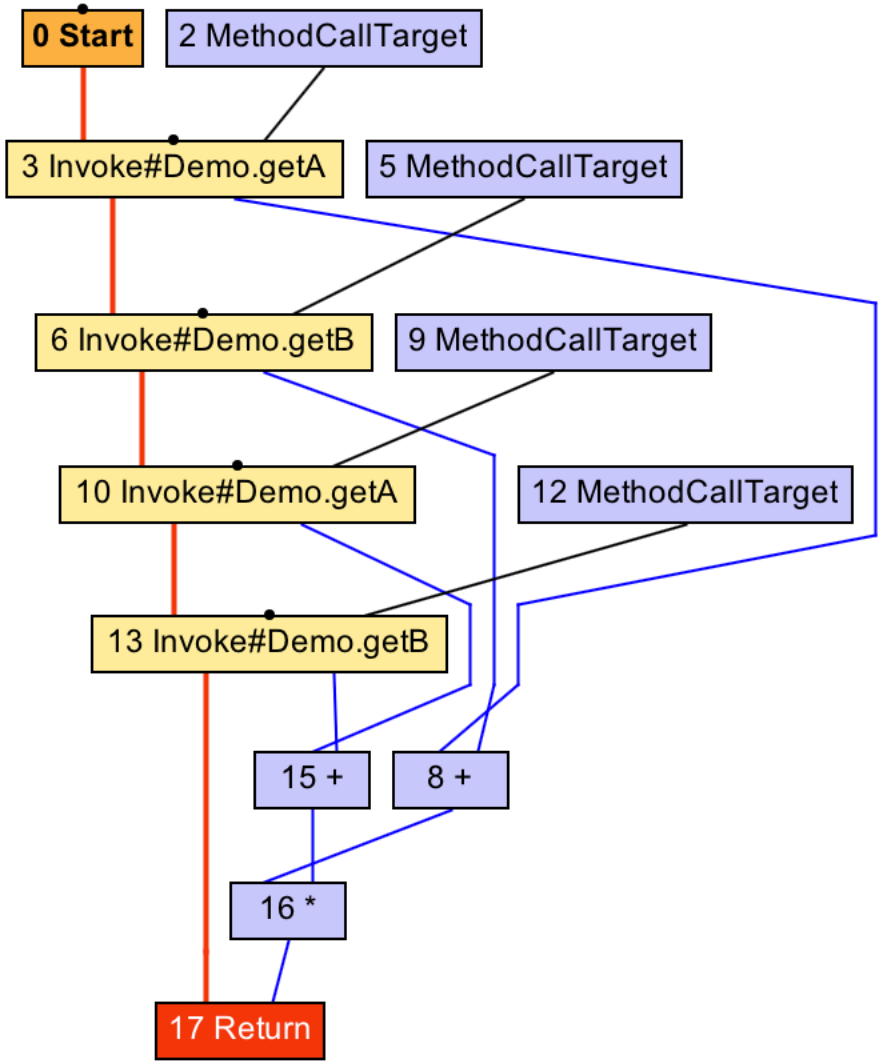
# *Global value numbering*

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





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



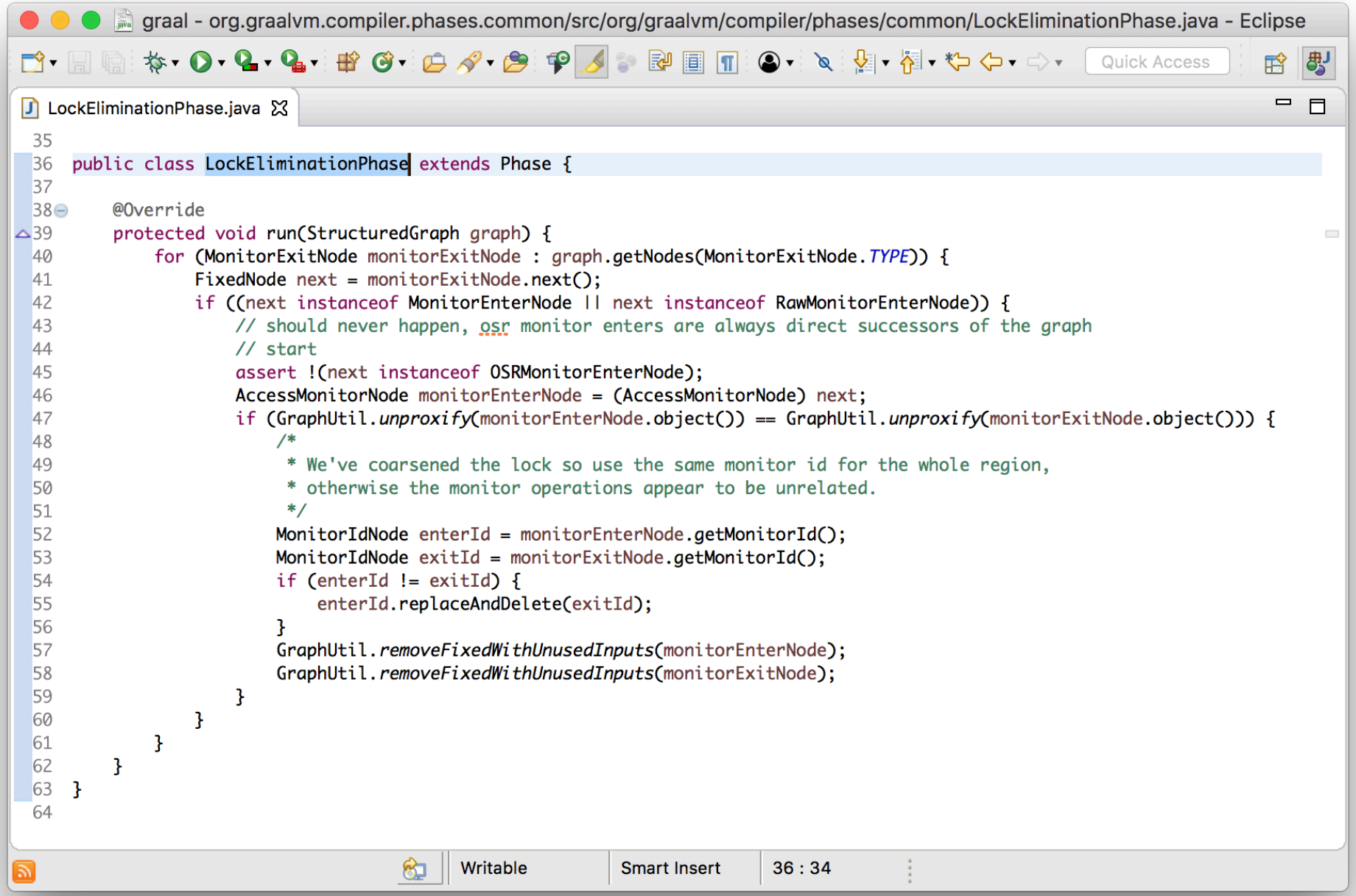
# *Lock coarsening*

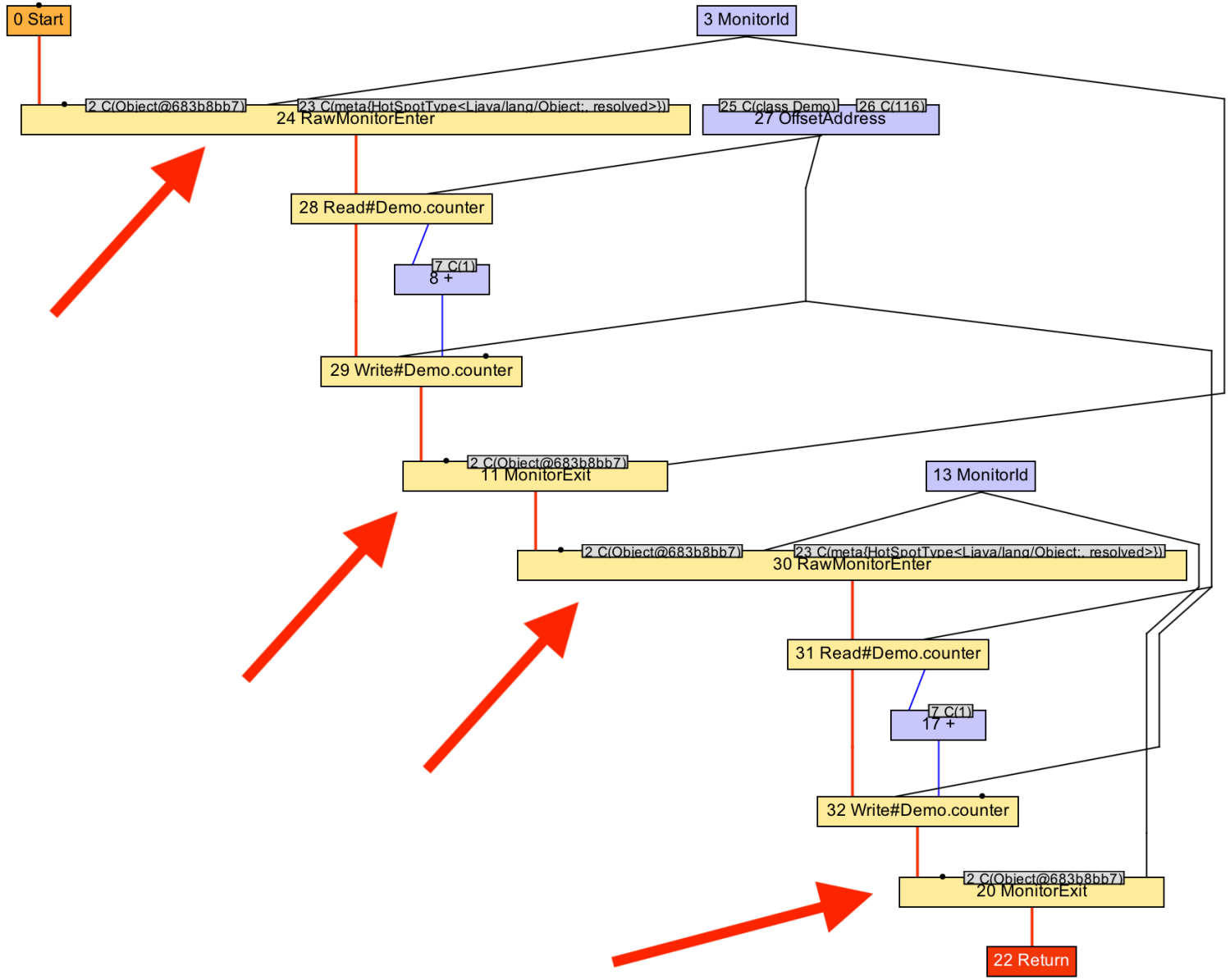
```
void workload() {
  synchronized (monitor) {
    counter++;
  }
  synchronized (monitor) {
    counter++;
  }
}
```

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

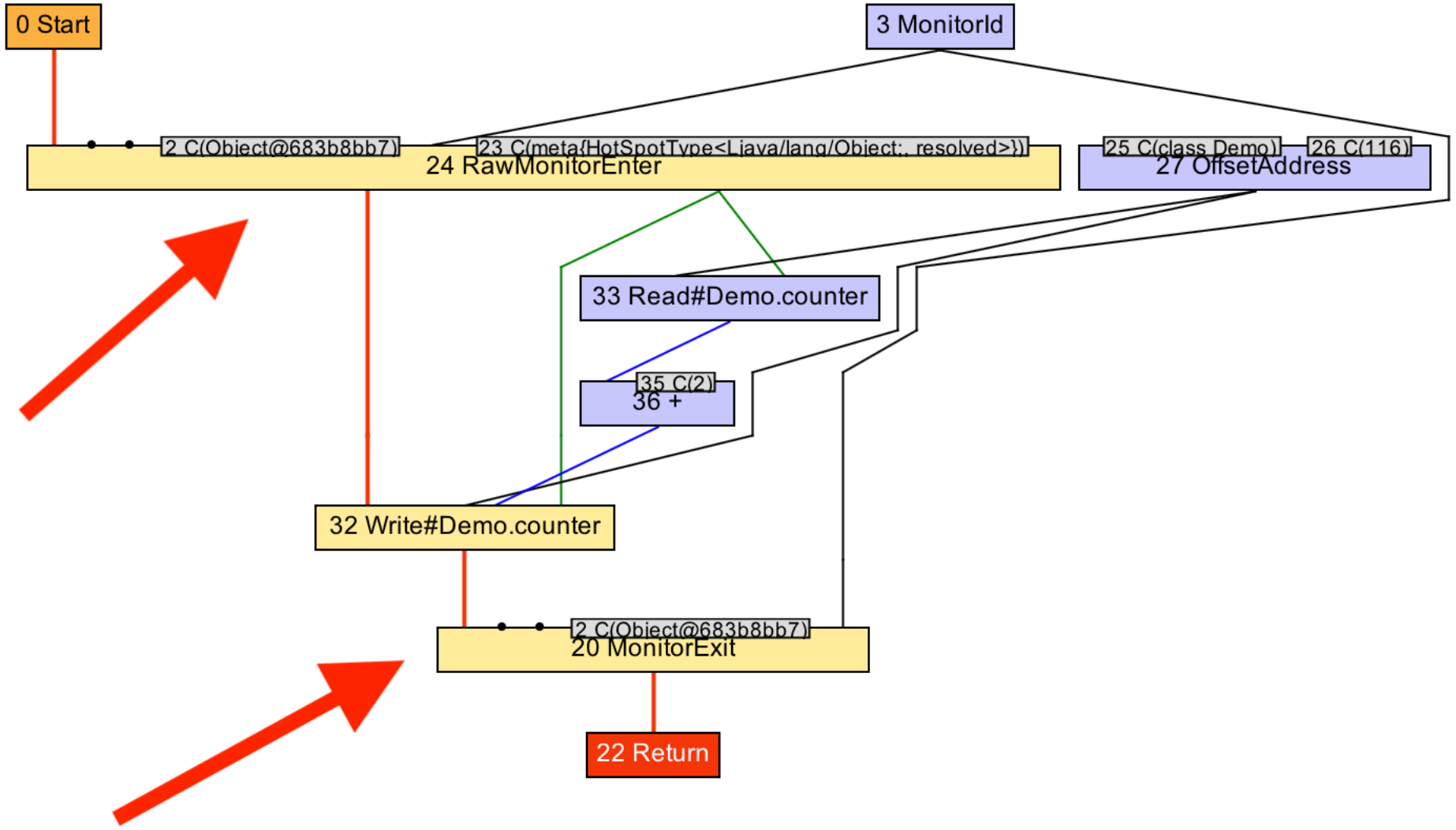
```
void workload() {
  monitor.enter();
  counter++;
  counter++;
  monitor.exit();
}
```

```
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();
      }
    }
  }
}
```





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



# Static Compilation

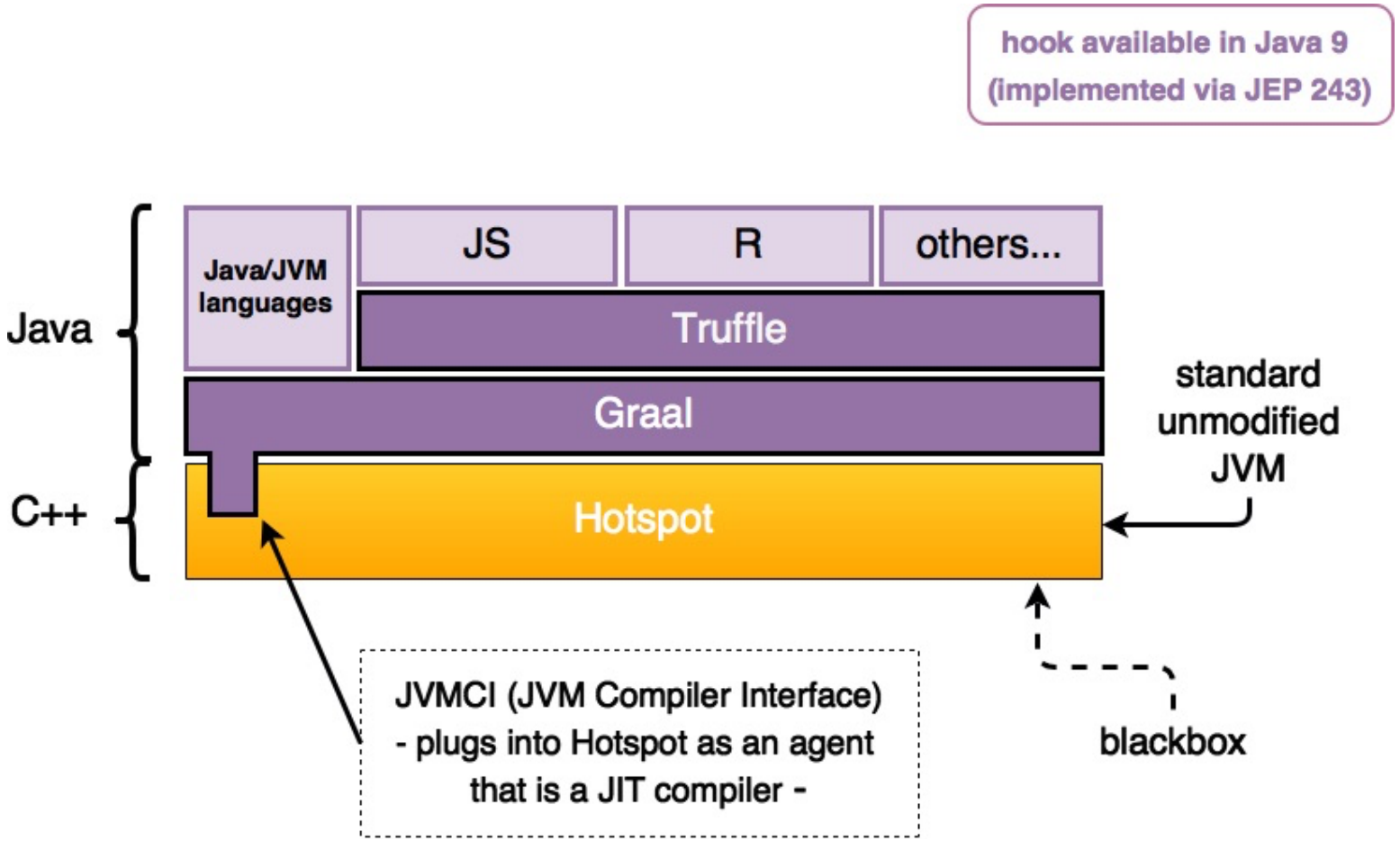
```
$ javac Hello.java

$ 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();
}
```

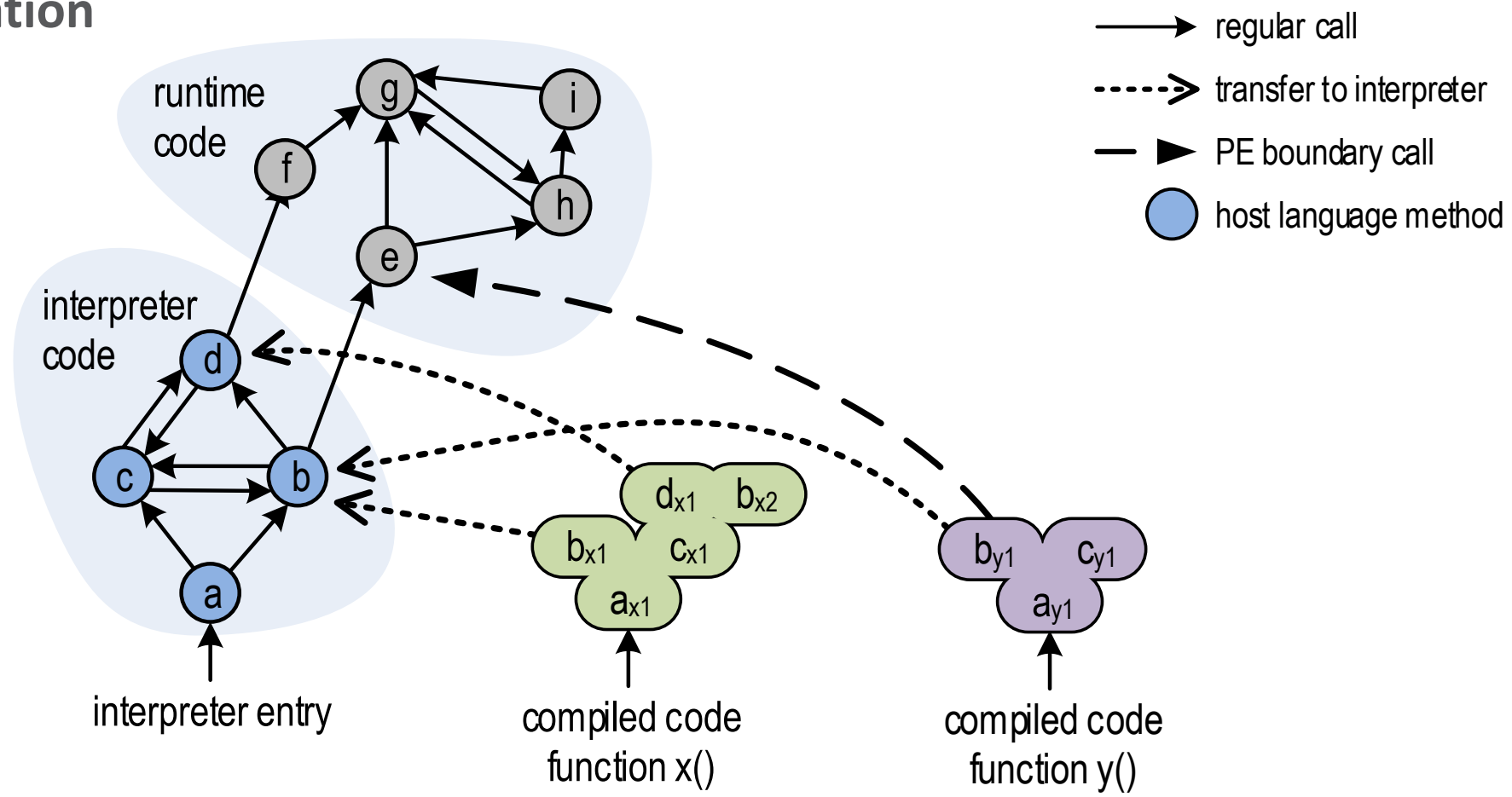
# Truffle

## 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

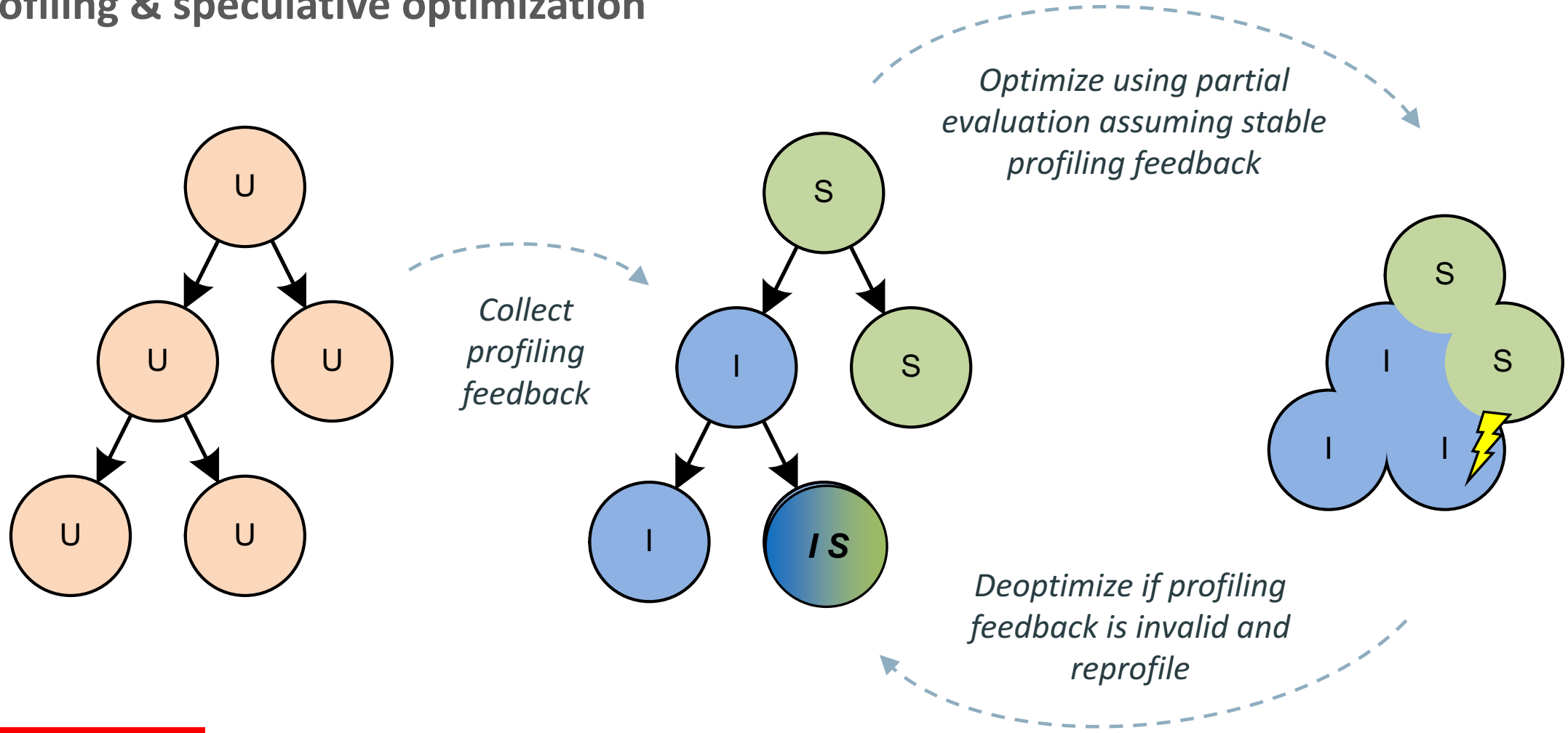
## partial evaluation



# Self-optimizing AST interpreters

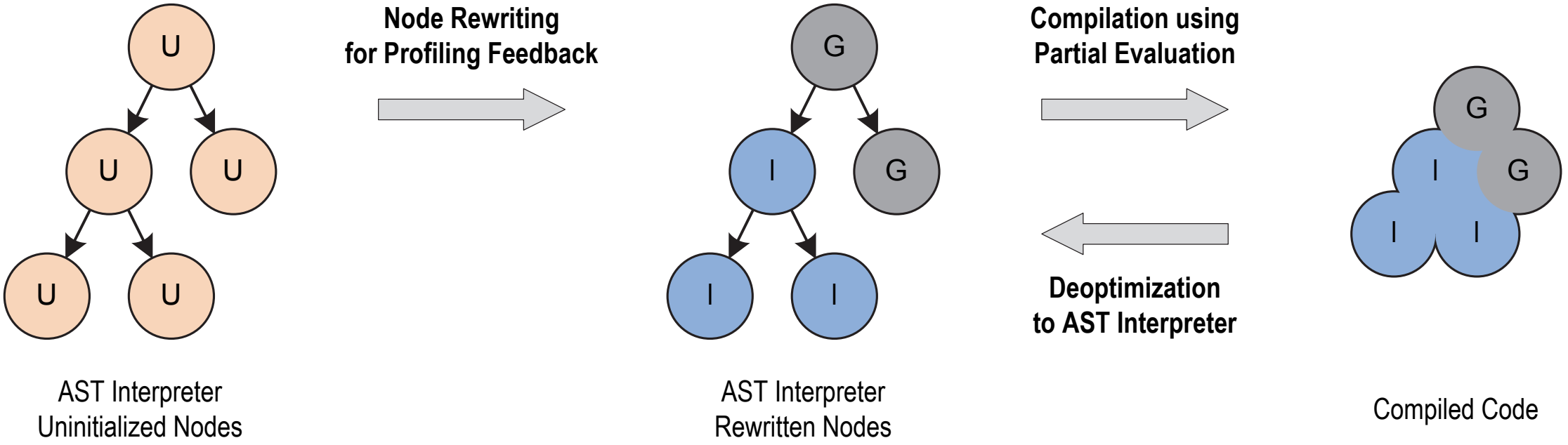
# Truffle

## profiling & speculative optimization



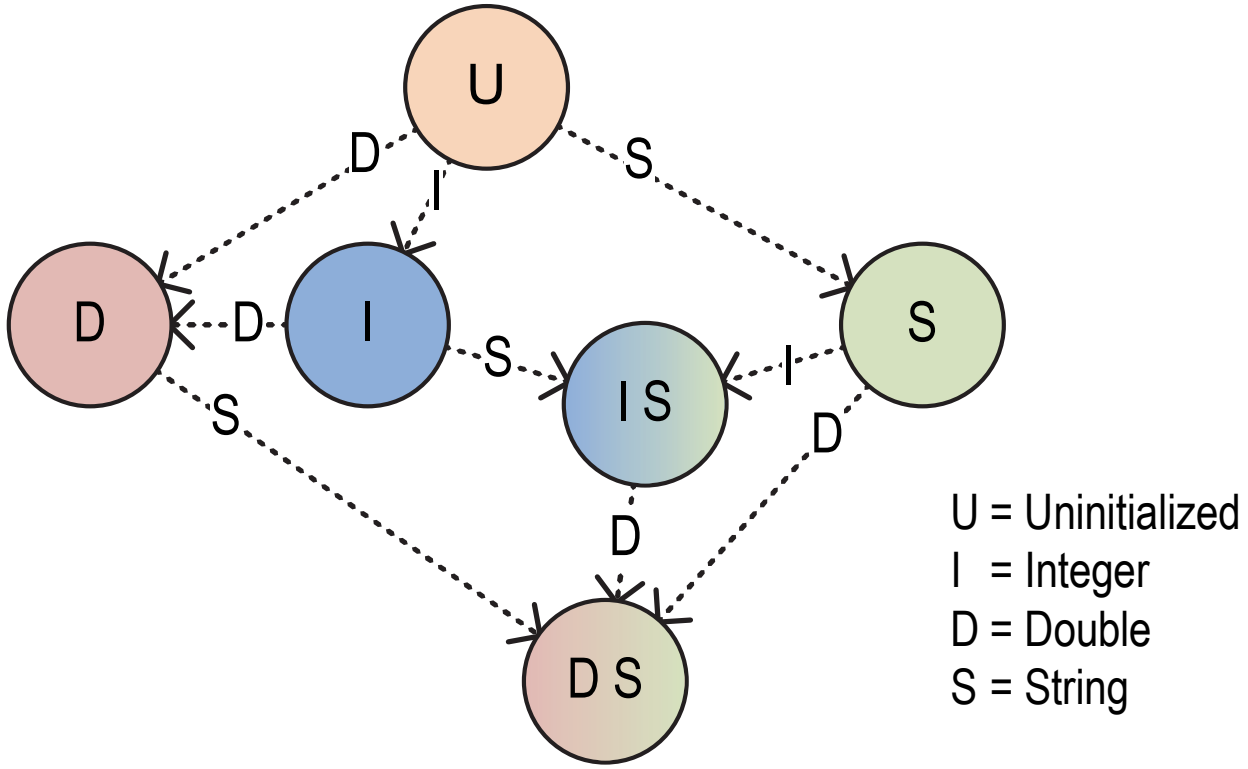
# Rewriting AST Nodes

## profiling & speculative optimization



# Rewriting AST Nodes

## transition graph

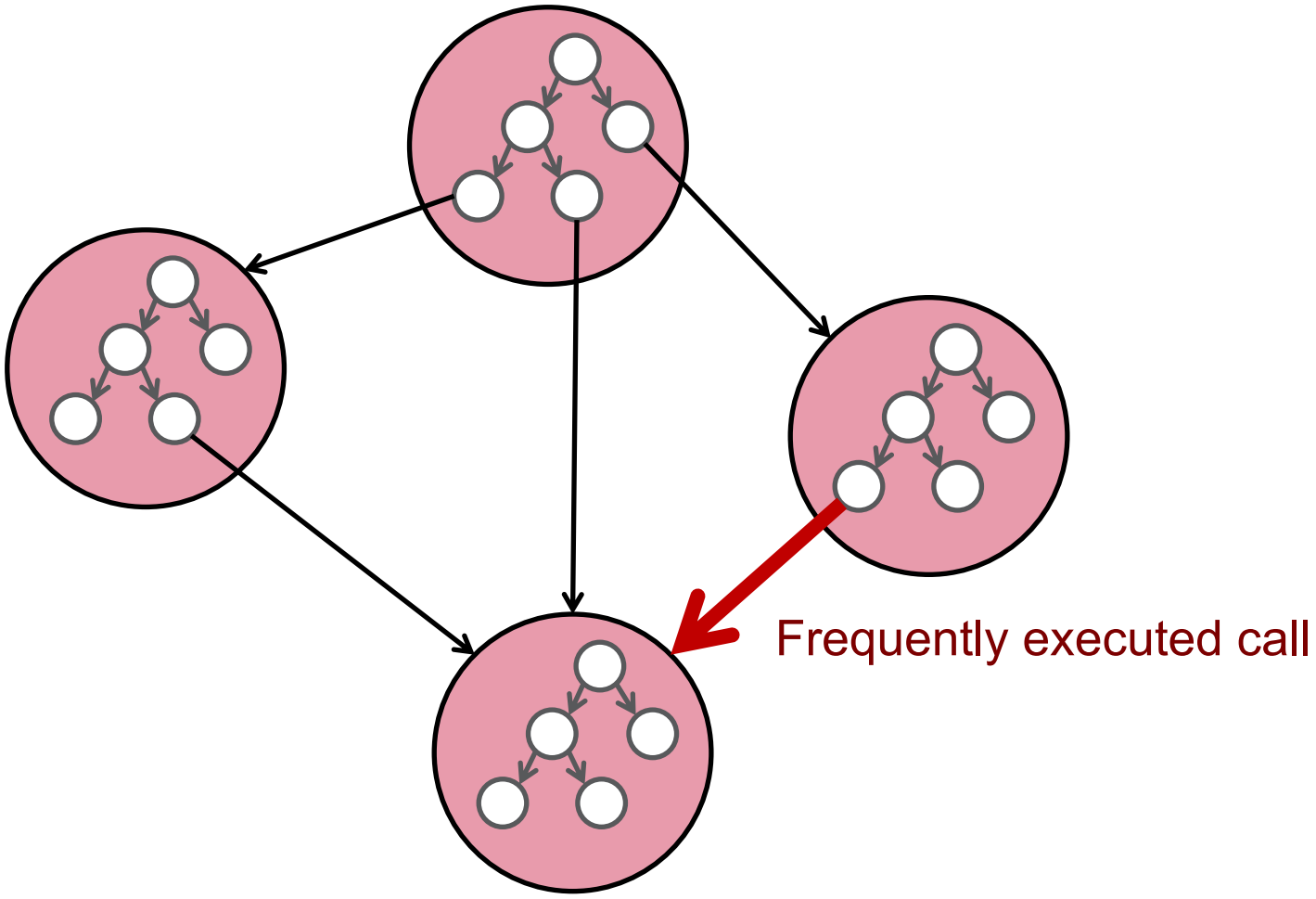


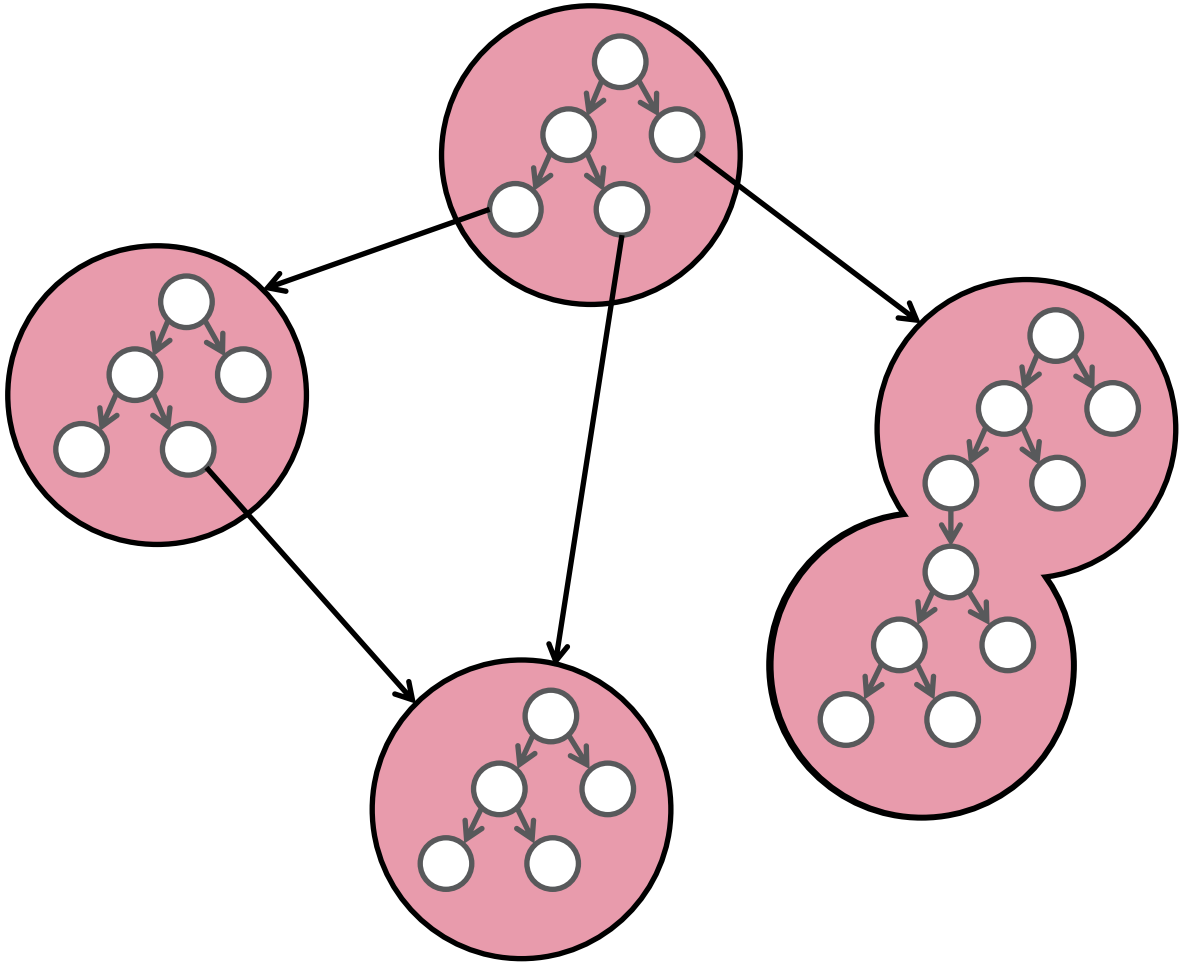
# Rewriting AST Nodes

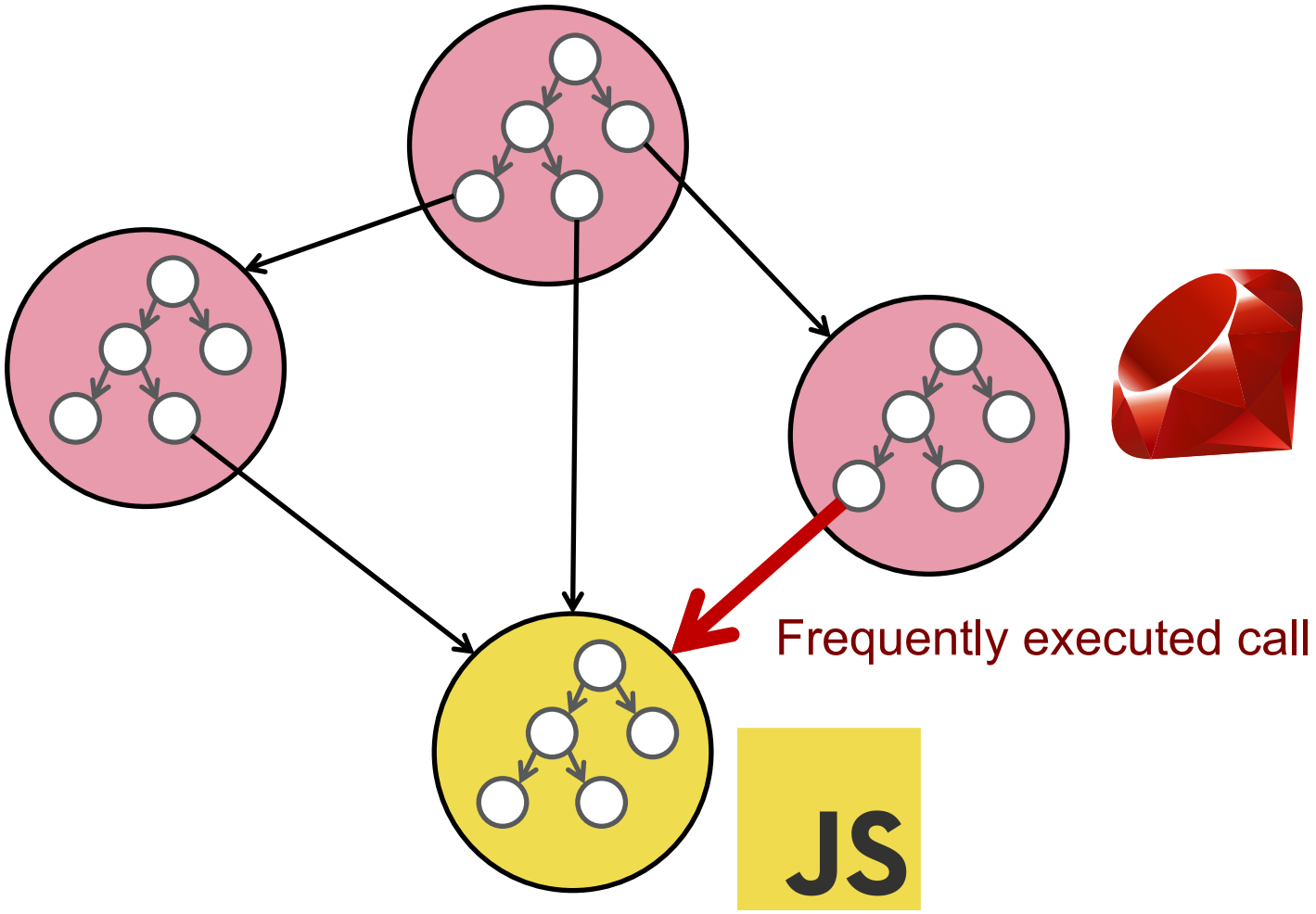
## specialization

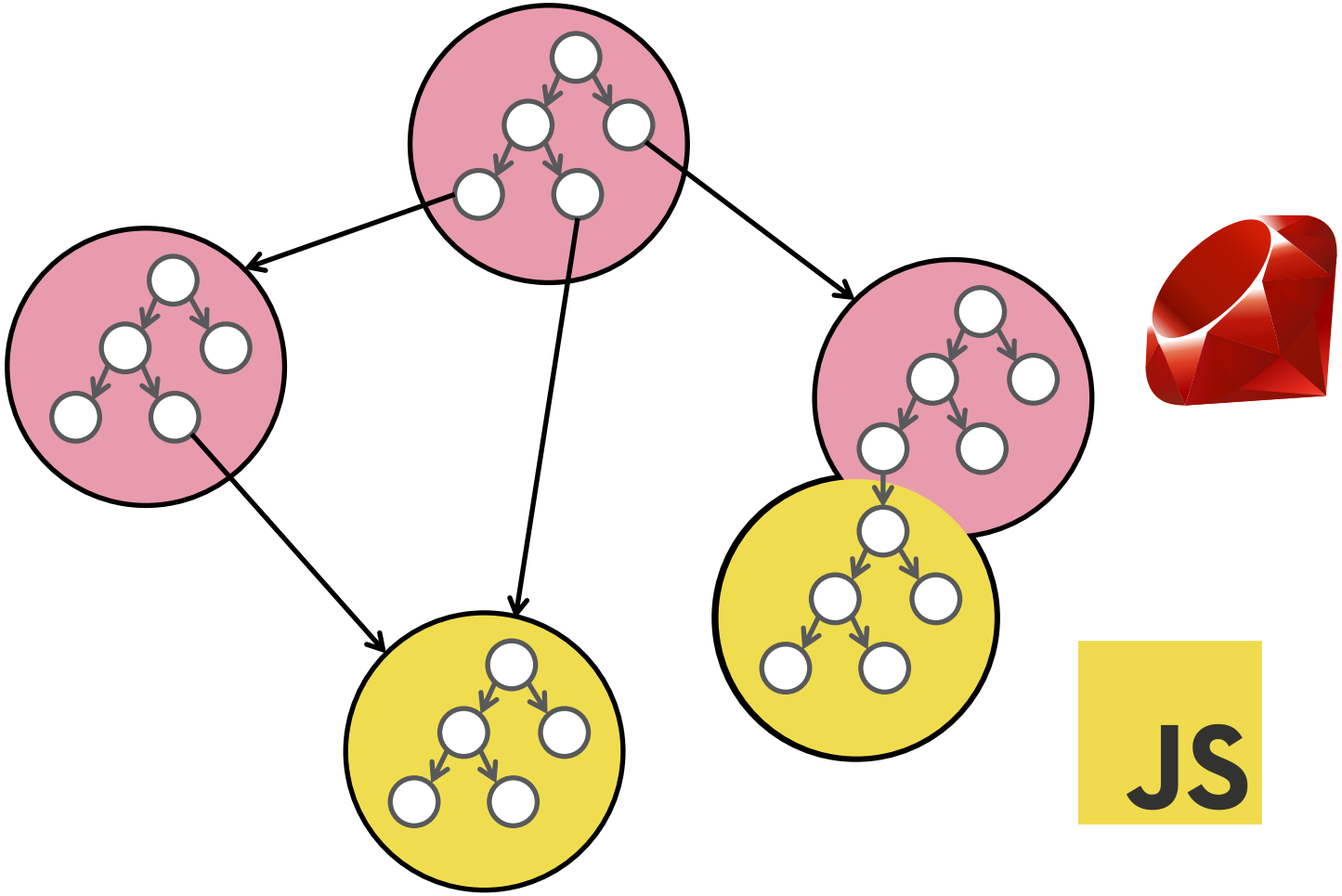
```
@NodeChild("left") @NodeChild("right")
abstract class AddNode extends BaseNode {

  @Specialization(rewriteOn=ArithmeticException.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;
  }
}
```

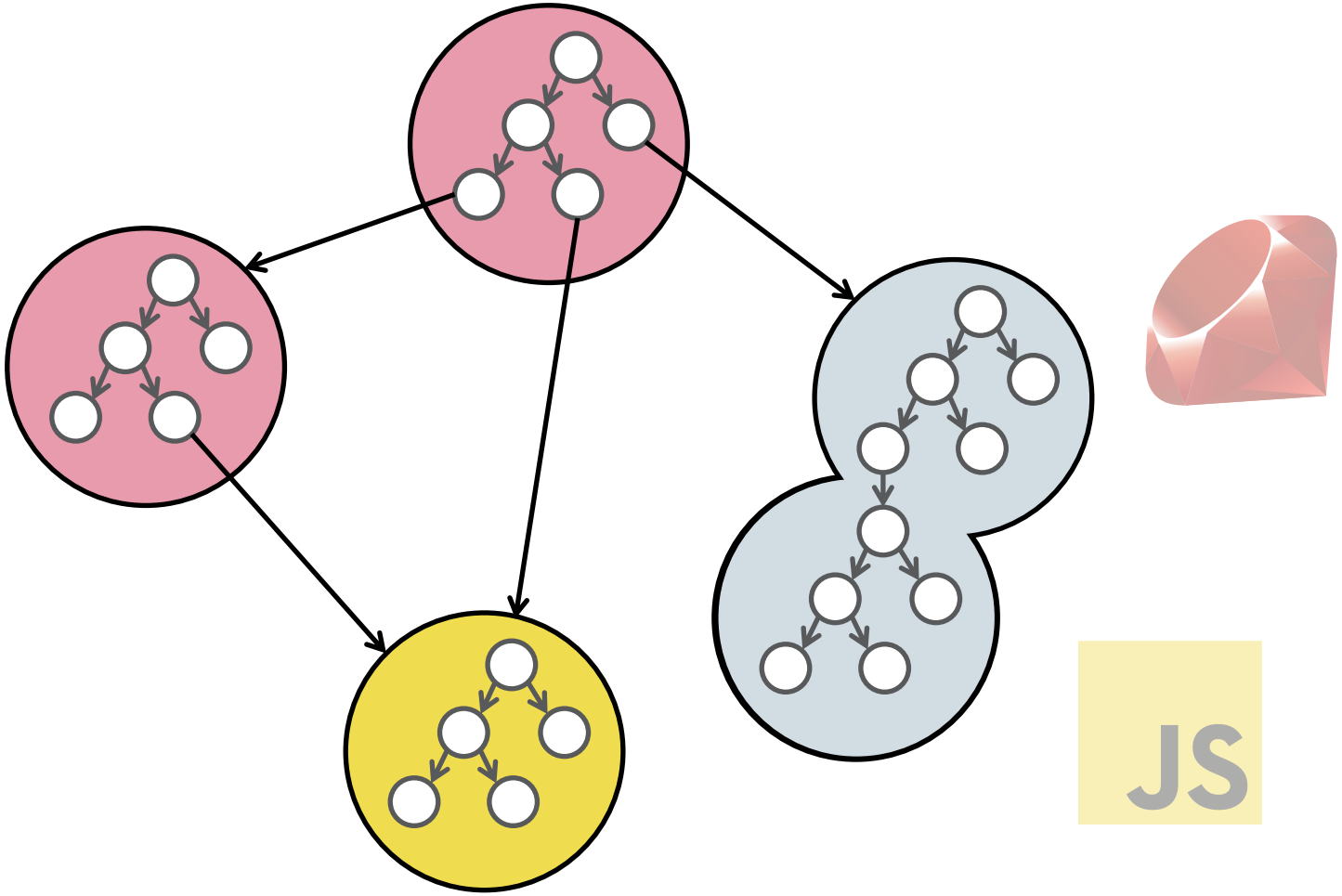
# Polyglot AST interpreters











- 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

## Inter-operability

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

## Dynamic compilation

- Apply modern runtime compilation technologies to data processing engine

## 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

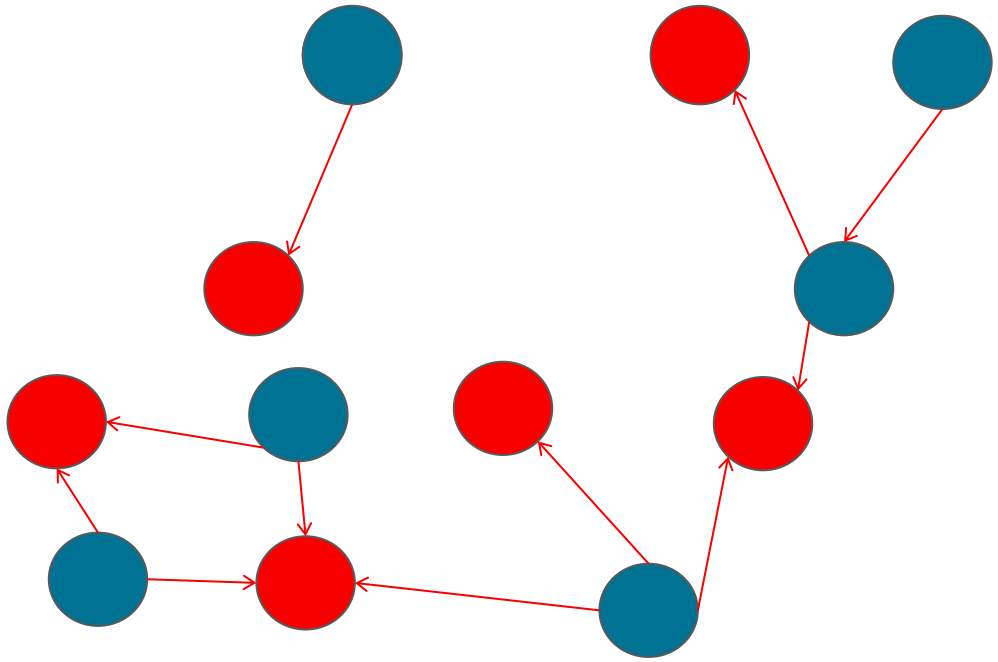
## Truffle framework

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

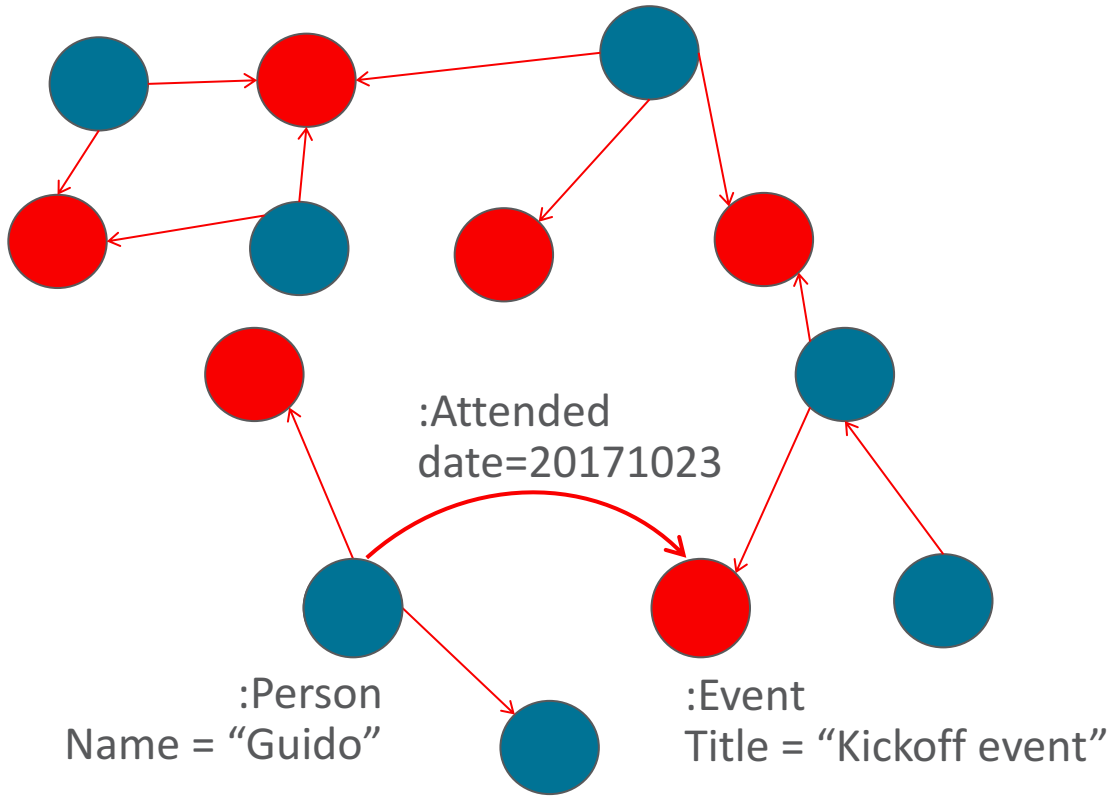
## Substrate VM

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

# Parallel Graph AnalytiX



# Data as a Graph



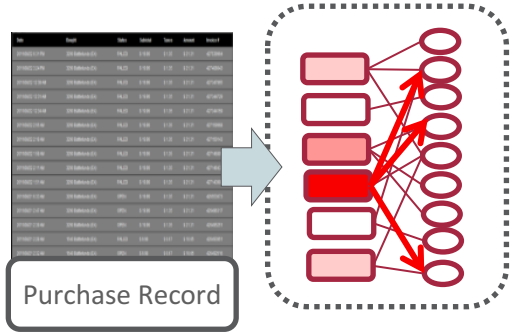

- Vertex: entity
- Edge: relationship

- Labels: identify vertices and edges
- Properties: describe vertices and edges

# Example Applications

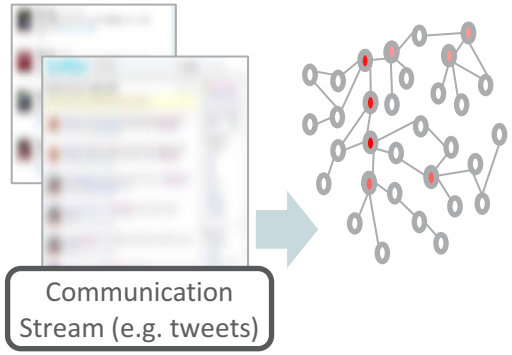
## Product recommendation

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



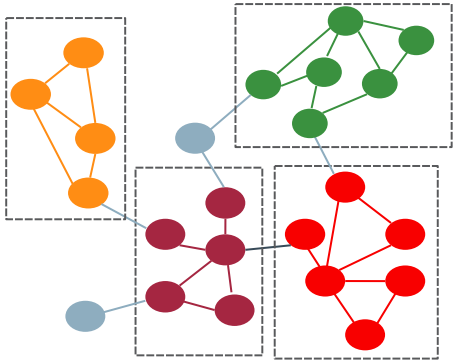

## Influencer identification

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



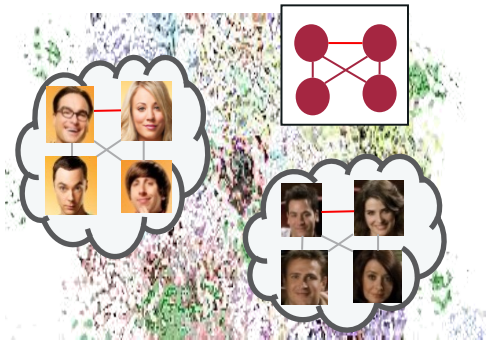

## Community detection

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



## Graph pattern matching

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



# Graph Workloads

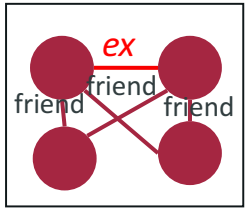
## Computational graph analytics

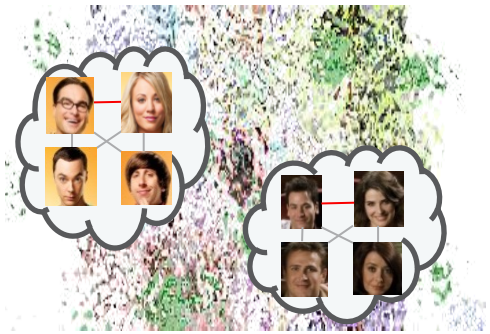
- **Compute** values on vertices and edges
- While **traversing** or **iterating** on the graph
- In **procedural** ways
- Connected Components, Shortest Path, Spanning Tree, Pagerank, Centrality

**Domain-specific language: Green-Marl**

## Graph pattern matching

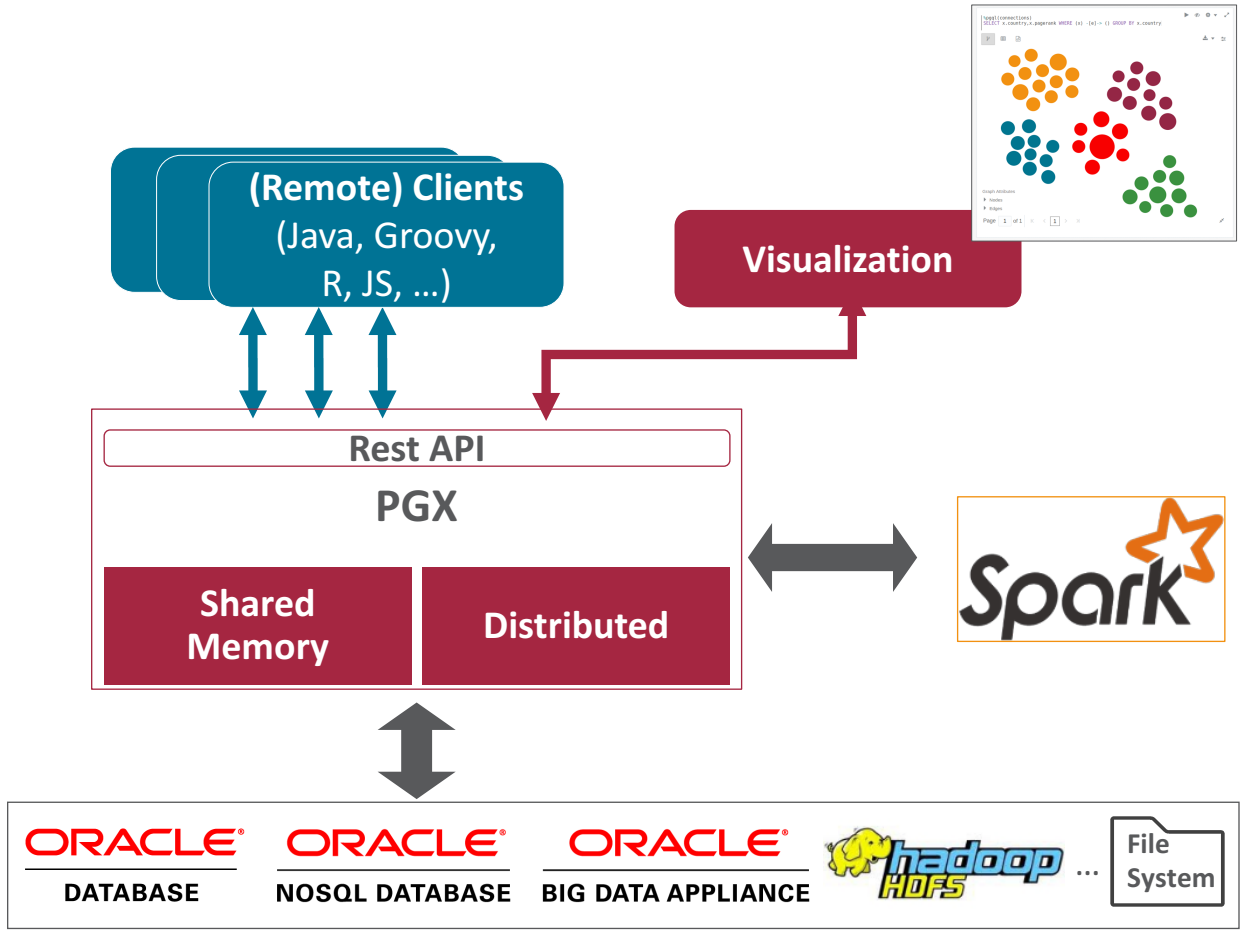
- Given a **description** of a pattern
- Find every subgraph which **matches** the pattern



**Domain-specific language: PGQL**

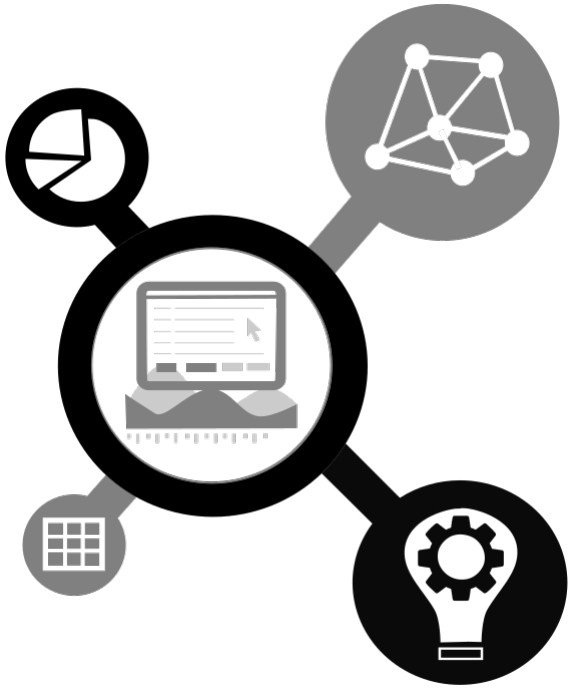
# Architecture Overview




- 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

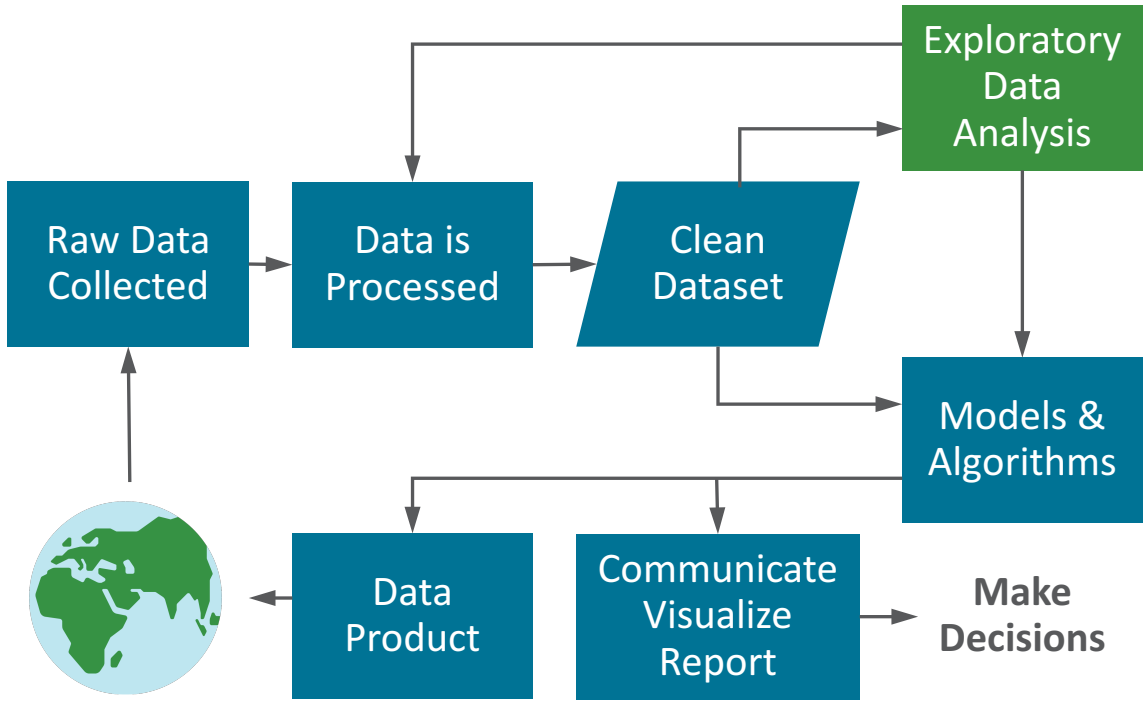
# Oracle Labs Data Studio

## Data Science with Interactive and Collaborative Notebooks



# Data Science: What tools are typically required?

## Data Science process




"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

**Notebook**

**IDE for Data Science**

## Backend tools

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

ORACLE DATABASE

Spark

# Rise of Notebook Technology

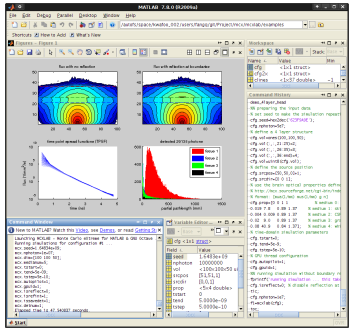
## Early tools

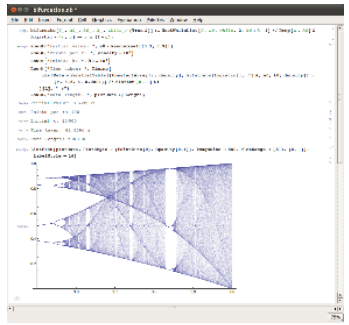
MATLAB

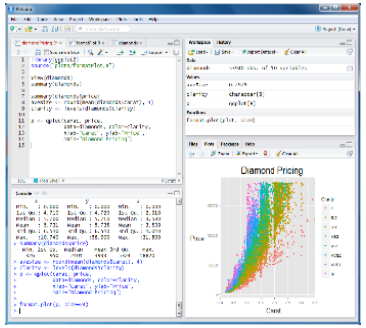
Wolfram Mathematica

RStudio

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







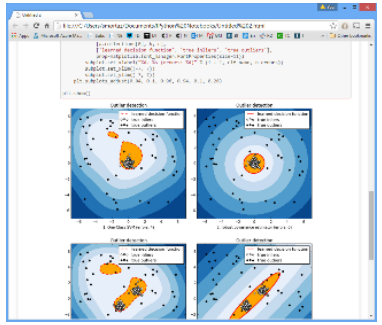

## Recent trends

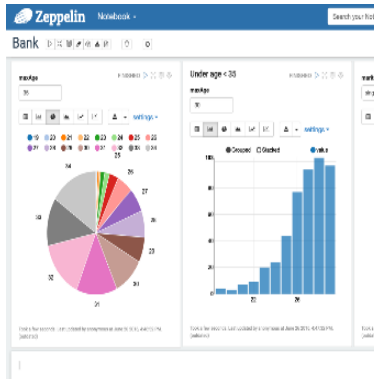
jupyter

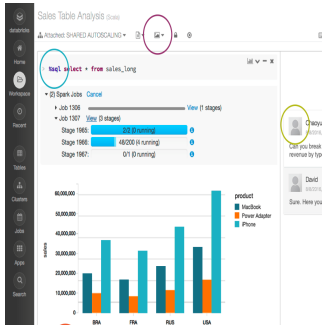
Apache Zeppelin

databricks

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



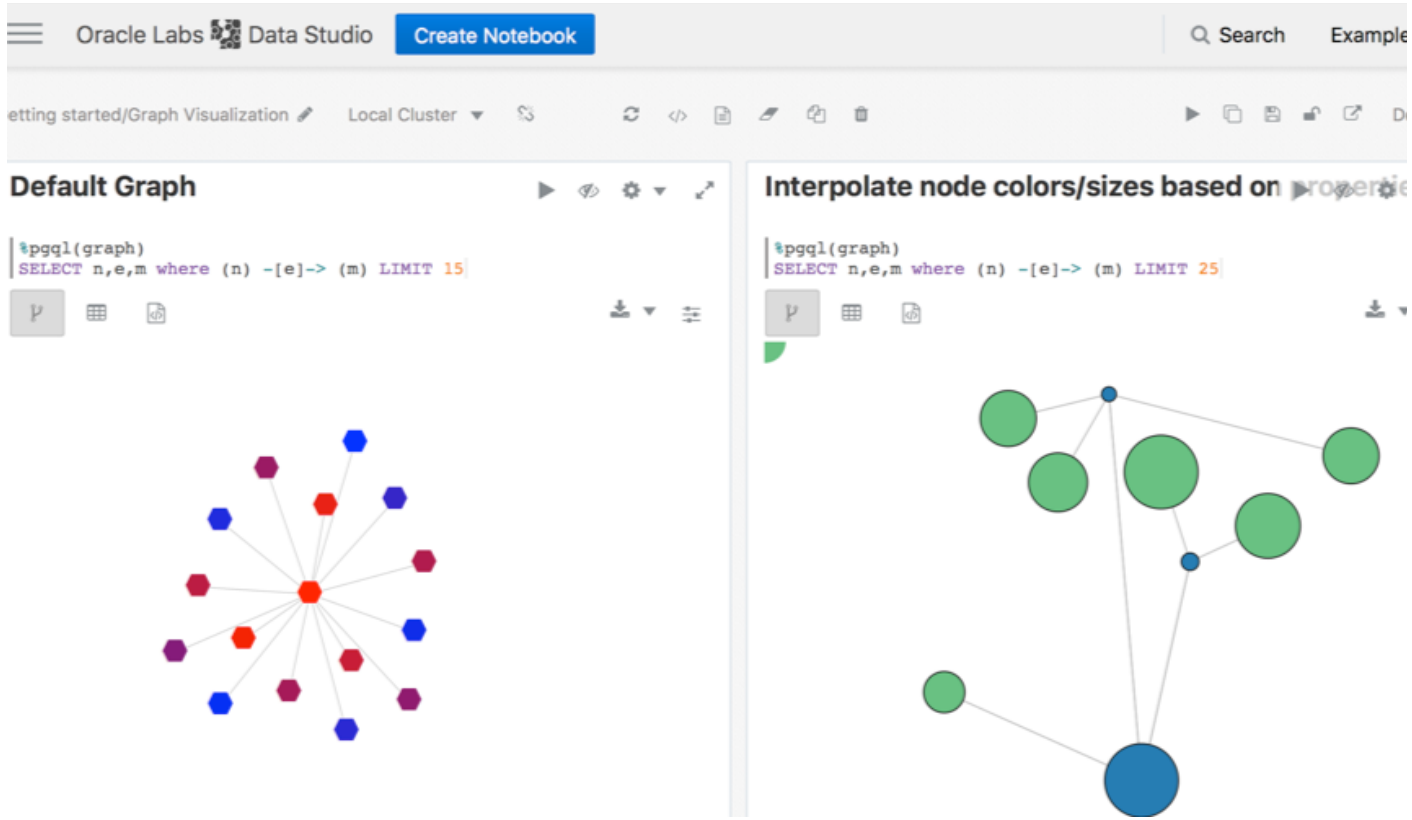


# Oracle Labs Data Studio

## 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)



# Integrated Cloud

## Applications & Platform Services


ORACLE®