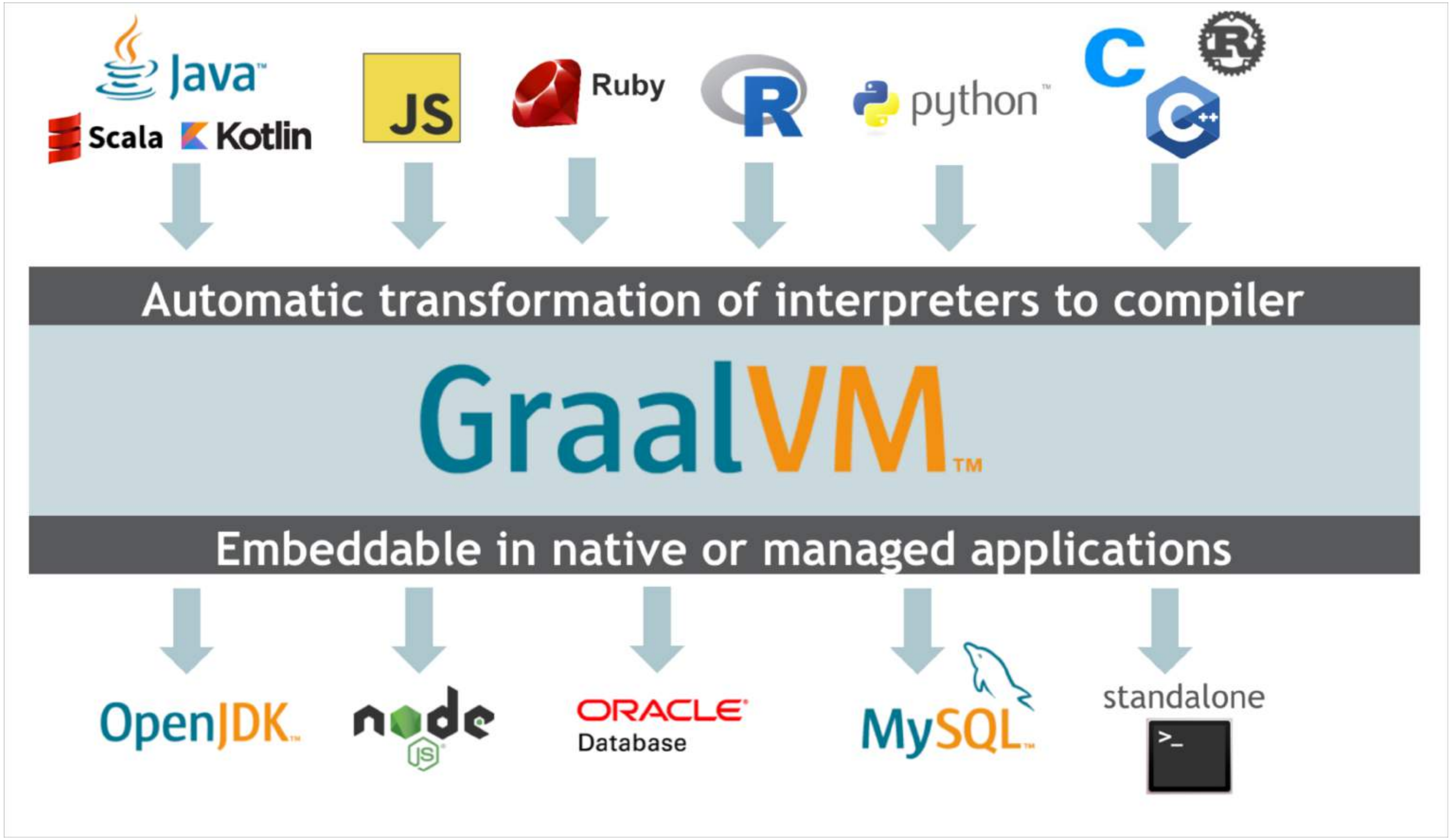
Graal: where it’s come from and where it’s going

Chris Seaton
Research Manager
Oracle Labs
February 2019

ORACLE

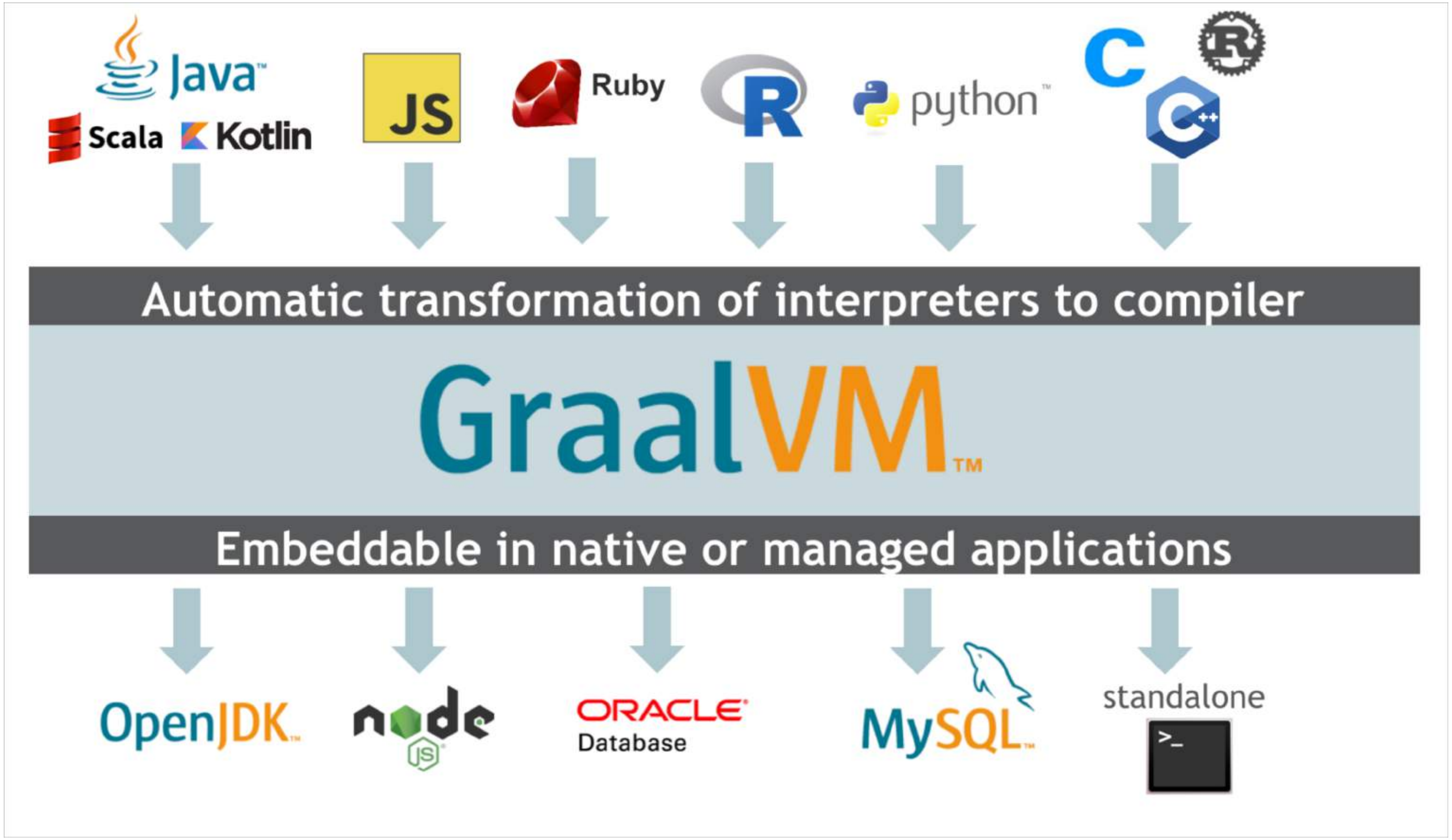
Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

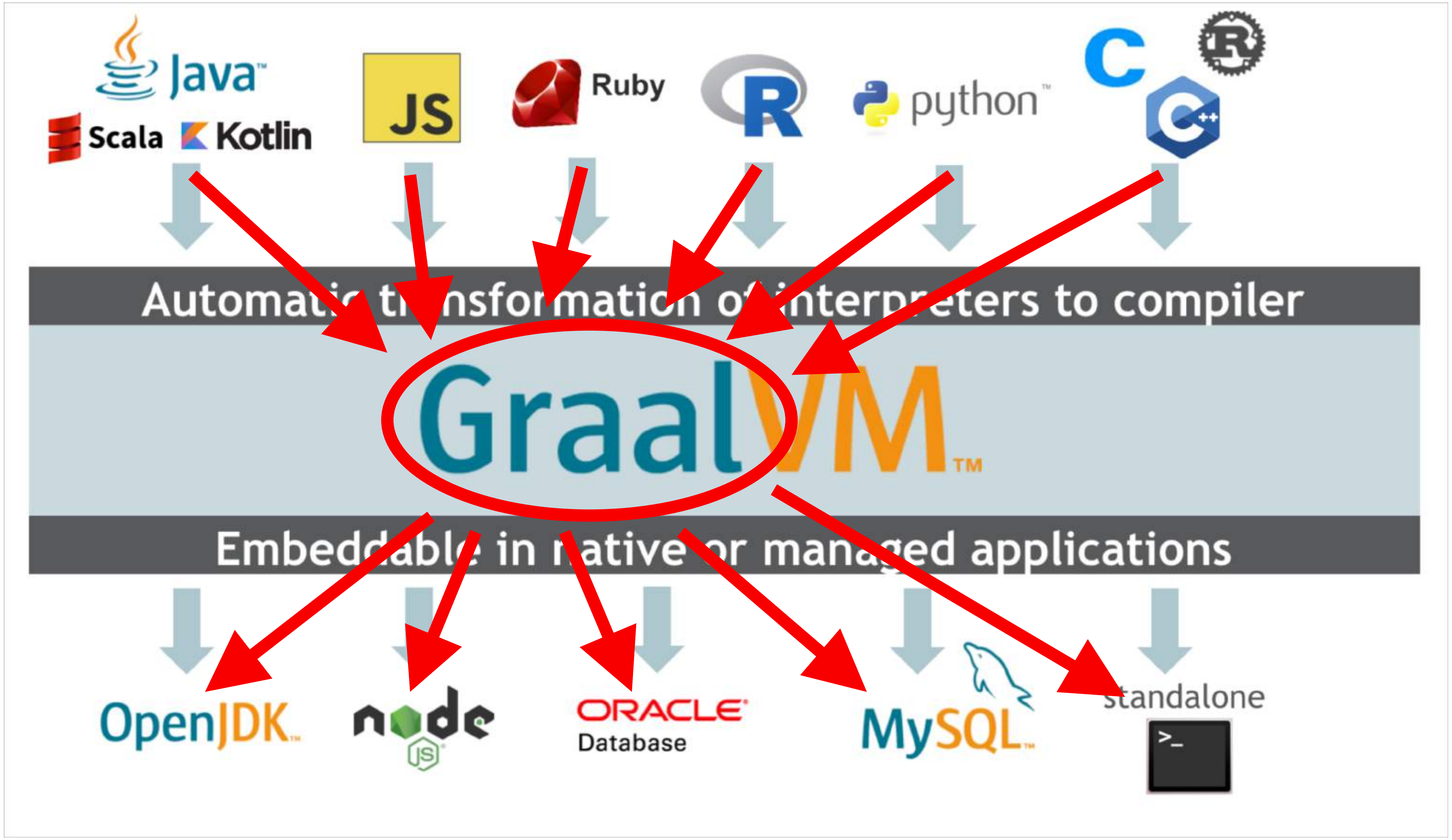

Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_

ORACLE

Why is this one group working on all these diverse things?



Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node
ORACLE Database
MySQL
standalone



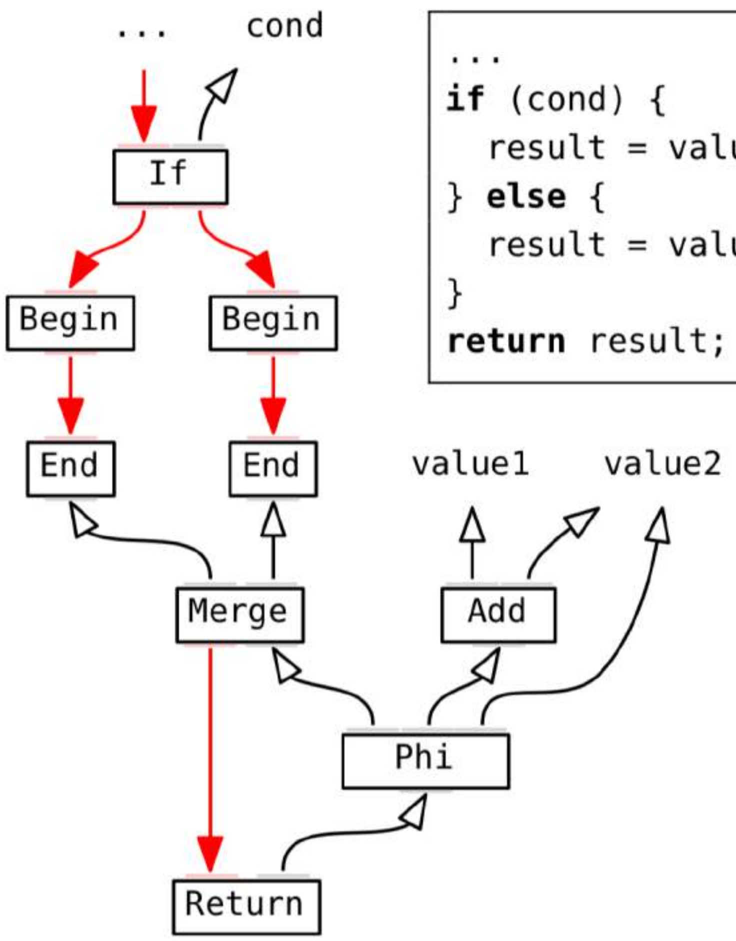
Java
Scala
Kotlin
JS
Ruby
R
python
C
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_

What *is* Graal?

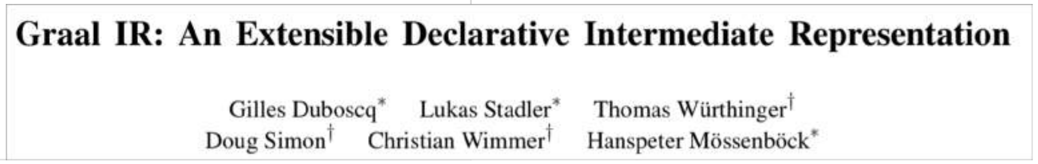
What *is* Graal?

- A compiler from Java intermediate representations to native machine code*
  - Intermediate representation could be JVM bytecode or Graal's own IR
  - 'Java' means any JVM language in general
  - Not a compiler from Java source code to Java bytecode
- It's a compiler library rather than being an executable, like GCC
- Some things I didn't tie Graal down to there:
  - I didn't say it was specifically a just-in-time compiler
  - I didn't say the bytecode or IR had to come directly from a program as written

Graal is *graphical*



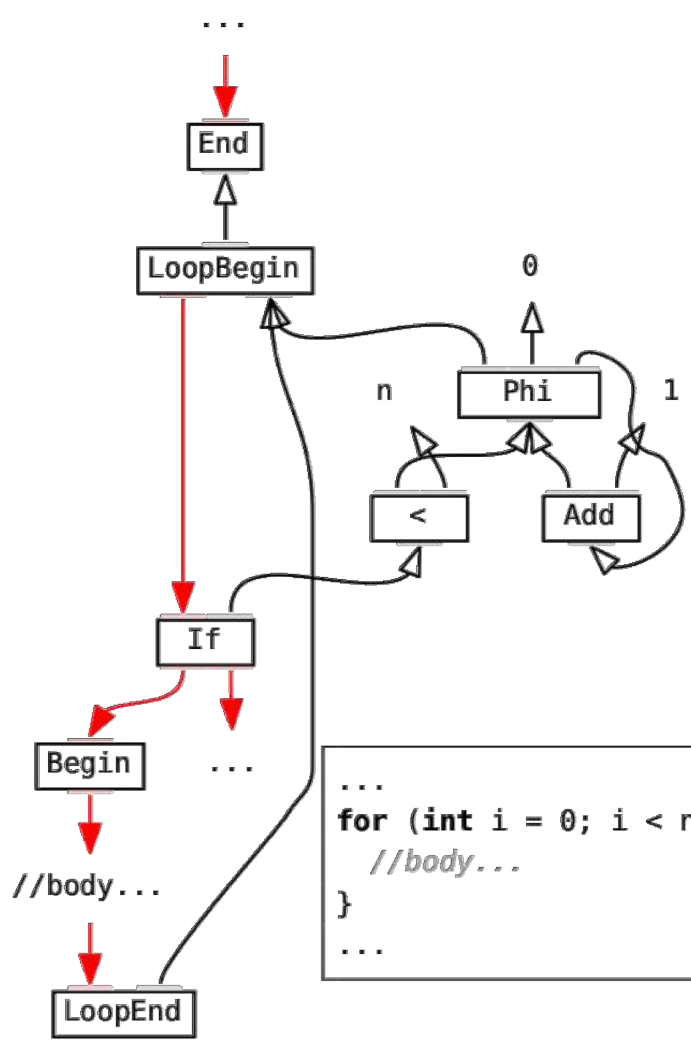
```
...
if (cond) {
  result = value1 + value2;
} else {
  result = value2;
}
return result;
```



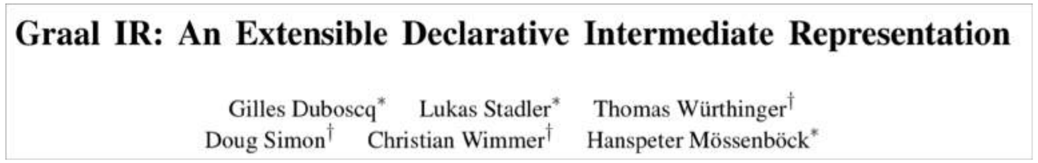
Graal IR: An Extensible Declarative Intermediate Representation

Gilles Duboscq* Lukas Stadler* Thomas Würthinger†
Doug Simon† Christian Wimmer† Hanspeter Mössenböck*

Graal is *graphical*



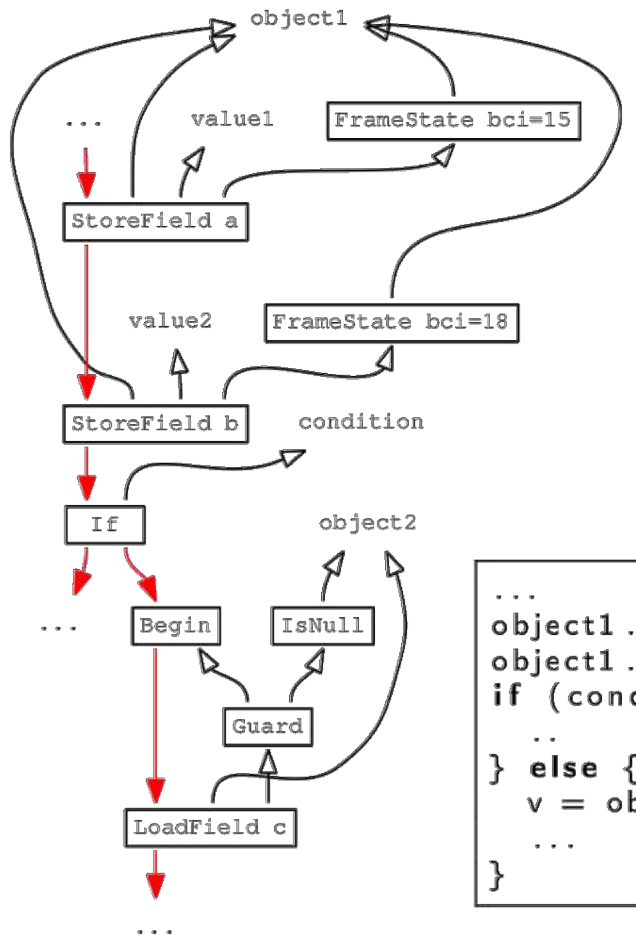
```
...
for (int i = 0; i < n; i++) {
  //body...
}
...
```

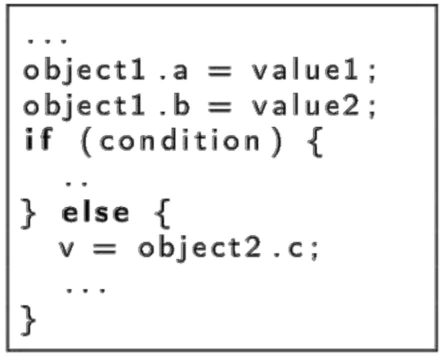


Graal IR: An Extensible Declarative Intermediate Representation

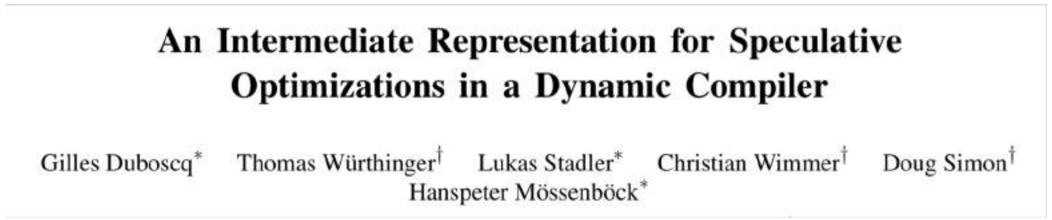
Gilles Duboscq[*] Lukas Stadler[*] Thomas Würthinger[†]
Doug Simon[†] Christian Wimmer[†] Hanspeter Mössenböck[*]

Graal is *optimizing* and *speculative**





```
...
object1.a = value1;
object1.b = value2;
if (condition) {
  ..
} else {
  v = object2.c;
  ...
}
```



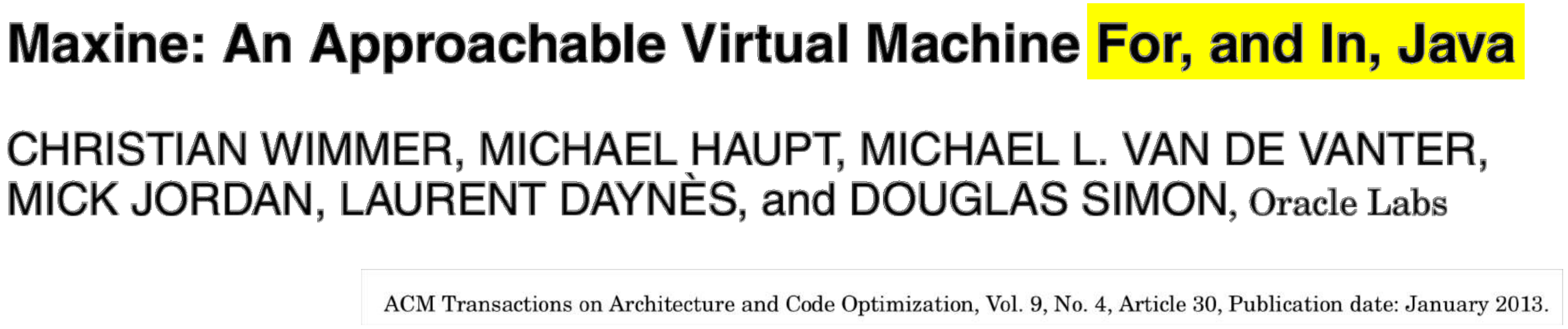
An Intermediate Representation for Speculative Optimizations in a Dynamic Compiler

Gilles Duboscq* Thomas Würthinger† Lukas Stadler* Christian Wimmer† Doug Simon† Hanspeter Mössenböck*

Graal is a *Java library*

```java
public byte[] compile(byte[] bytecode) {
  ...
}
```

Where did Graal come from?



Maxine: An Approachable Virtual Machine For, and In, Java

CHRISTIAN WIMMER, MICHAEL HAUPT, MICHAEL L. VAN DE VANTER, MICK JORDAN, LAURENT DAYNÈS, and DOUGLAS SIMON, Oracle Labs

ACM Transactions on Architecture and Code Optimization, Vol. 9, No. 4, Article 30, Publication date: January 2013.

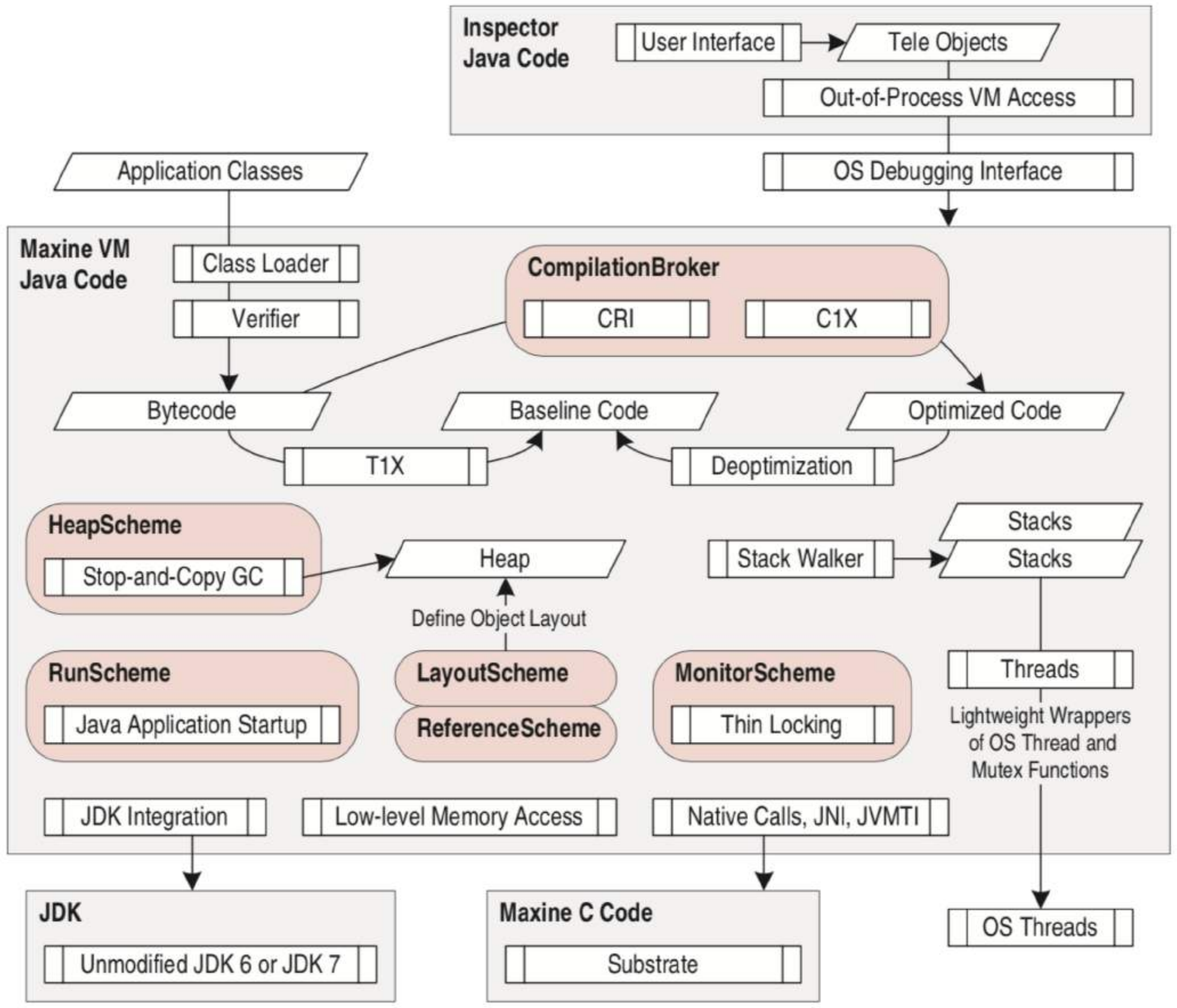


Fig. 1. Structure of the Maxine VM.

Equivalent to JVMCI

Became Graal

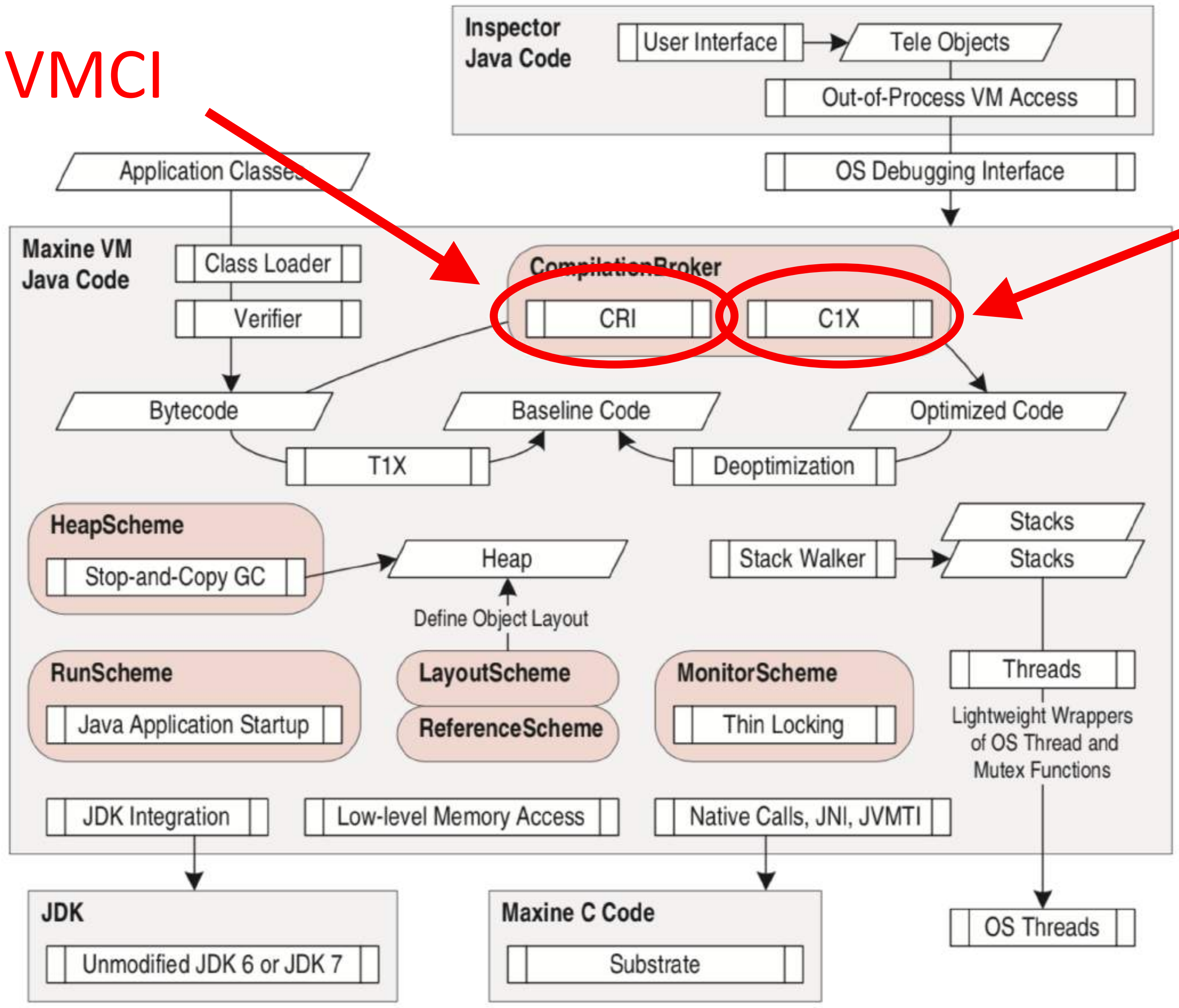
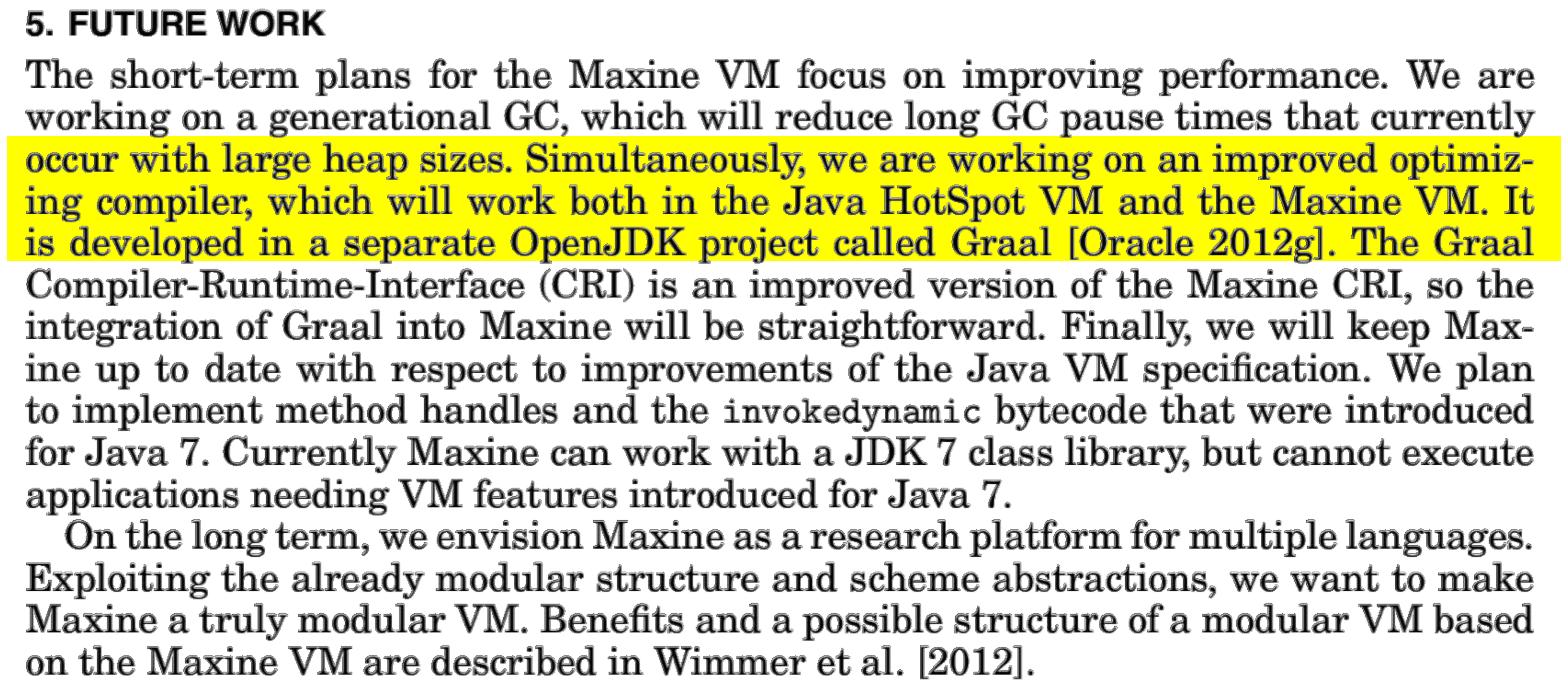


Fig. 1. Structure of the Maxine VM.

(This is the ‘future’ as at 2013.)



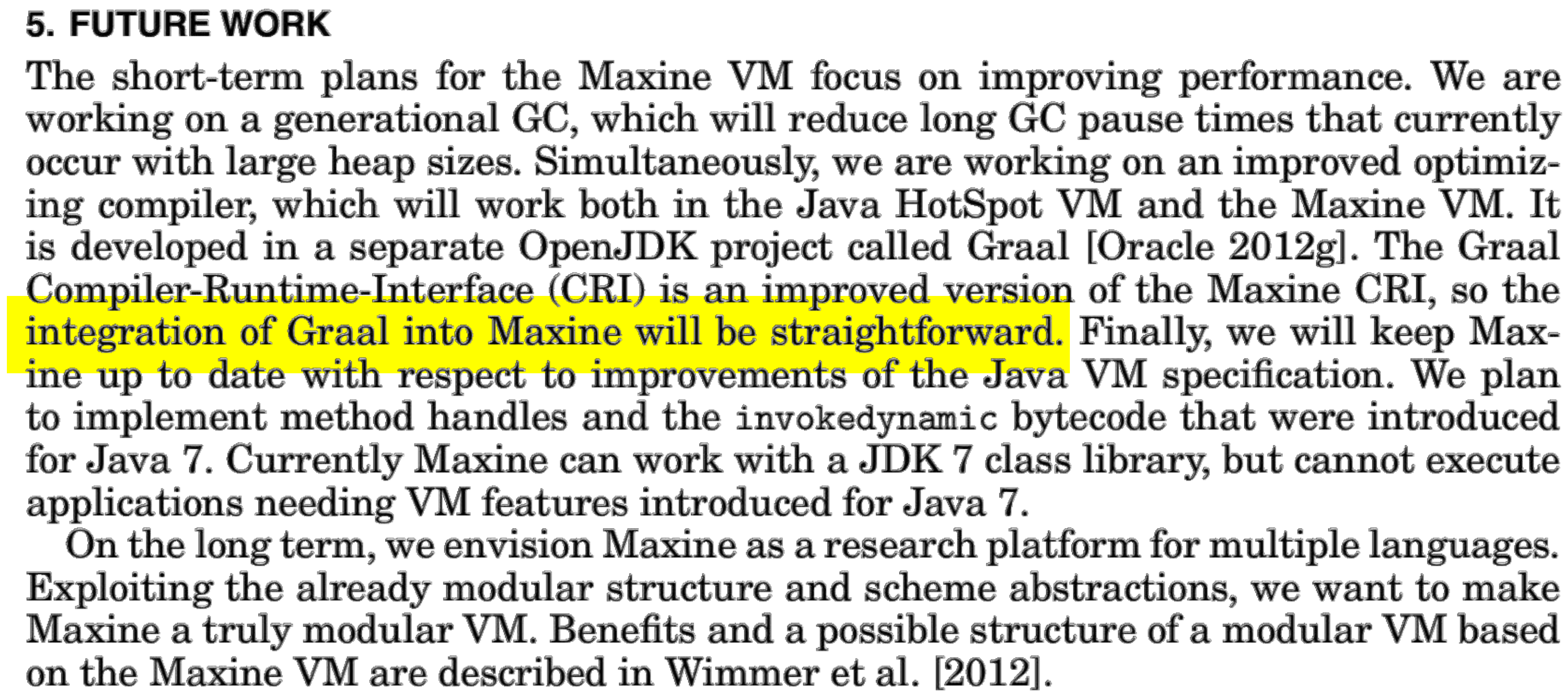
5. FUTURE WORK

The short-term plans for the Maxine VM focus on improving performance. We are working on a generational GC, which will reduce long GC pause times that currently occur with large heap sizes. Simultaneously, we are working on an improved optimizing compiler, which will work both in the Java HotSpot VM and the Maxine VM. It is developed in a separate OpenJDK project called Graal [Oracle 2012g]. The Graal Compiler-Runtime-Interface (CRI) is an improved version of the Maxine CRI, so the integration of Graal into Maxine will be straightforward. Finally, we will keep Maxine up to date with respect to improvements of the Java VM specification. We plan to implement method handles and the `invokedynamic` bytecode that were introduced for Java 7. Currently Maxine can work with a JDK 7 class library, but cannot execute applications needing VM features introduced for Java 7.

On the long term, we envision Maxine as a research platform for multiple languages. Exploiting the already modular structure and scheme abstractions, we want to make Maxine a truly modular VM. Benefits and a possible structure of a modular VM based on the Maxine VM are described in Wimmer et al. [2012].

C1 to Graal

- C1 (the 'client' compiler)
  - I believe it's actually newer then C2 (the 'server', or 'optimizing' or 'opto' compiler)
  - Designed to produce reasonably good code reasonably quickly
  - More than a template compiler
- Became C1X when rewritten in Java for Maxine
  - "more or less literal Java port of the C++ code of C1"
- Became Graal when made a component usable outside of Maxine
  - Not really sure this is quite an accurate thing to say
  - Graal used the same LIR as C1X, but the IR in Graal is sea-of-nodes, while C1 is CFG
  - The LIR has evolved a lot since then as well

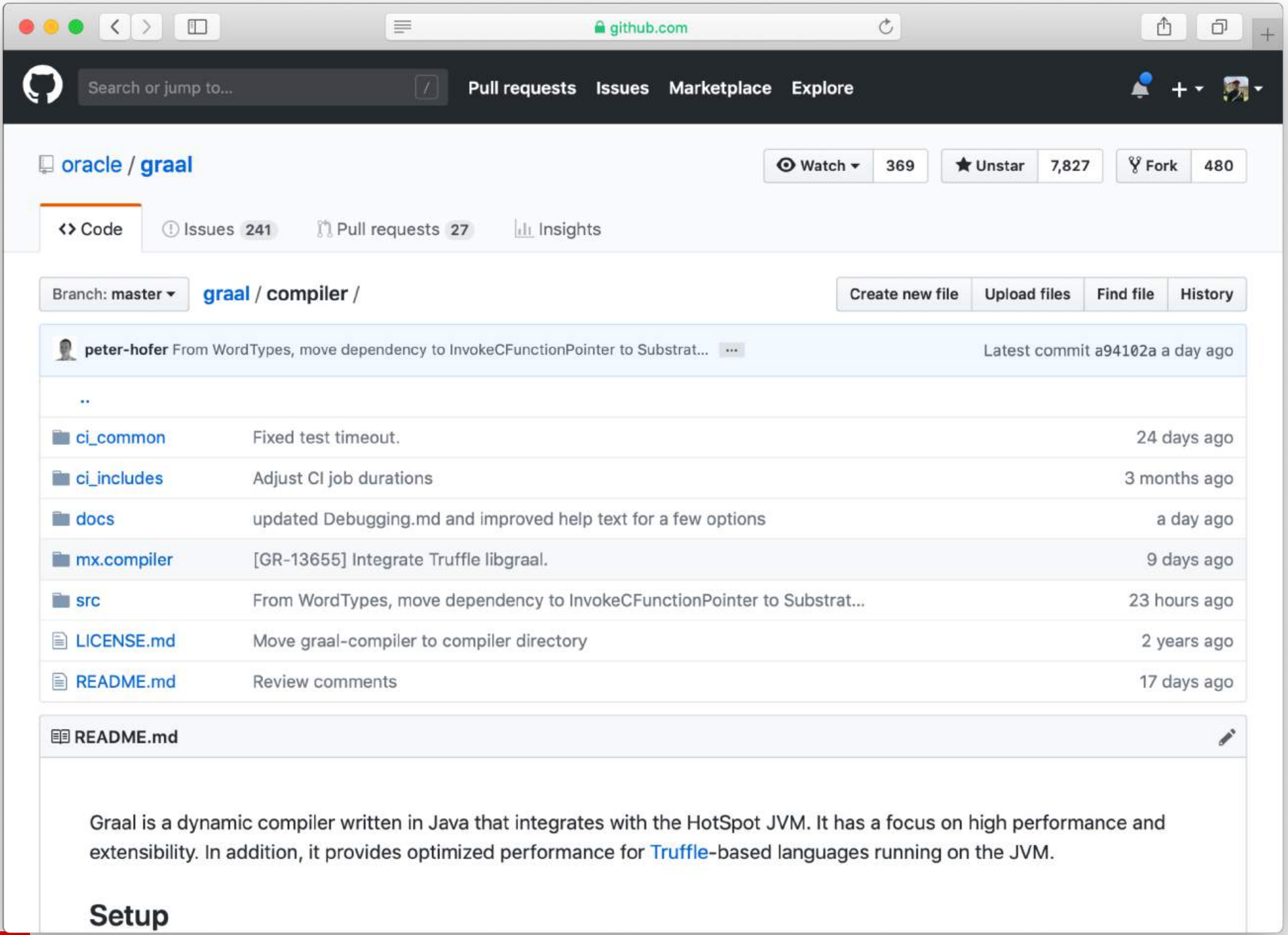


5. FUTURE WORK

The short-term plans for the Maxine VM focus on improving performance. We are working on a generational GC, which will reduce long GC pause times that currently occur with large heap sizes. Simultaneously, we are working on an improved optimizing compiler, which will work both in the Java HotSpot VM and the Maxine VM. It is developed in a separate OpenJDK project called Graal [Oracle 2012g]. The Graal Compiler-Runtime-Interface (CRI) is an improved version of the Maxine CRI, so the integration of Graal into Maxine will be straightforward. Finally, we will keep Maxine up to date with respect to improvements of the Java VM specification. We plan to implement method handles and the `invokedynamic` bytecode that were introduced for Java 7. Currently Maxine can work with a JDK 7 class library, but cannot execute applications needing VM features introduced for Java 7.

On the long term, we envision Maxine as a research platform for multiple languages. Exploiting the already modular structure and scheme abstractions, we want to make Maxine a truly modular VM. Benefits and a possible structure of a modular VM based on the Maxine VM are described in Wimmer et al. [2012].

What does Graal look like today?



oracle / graal

graal / compiler /

peter-hofer From WordTypes, move dependency to InvokeCFunctionPointer to Substrat... Latest commit a94102a a day ago

..

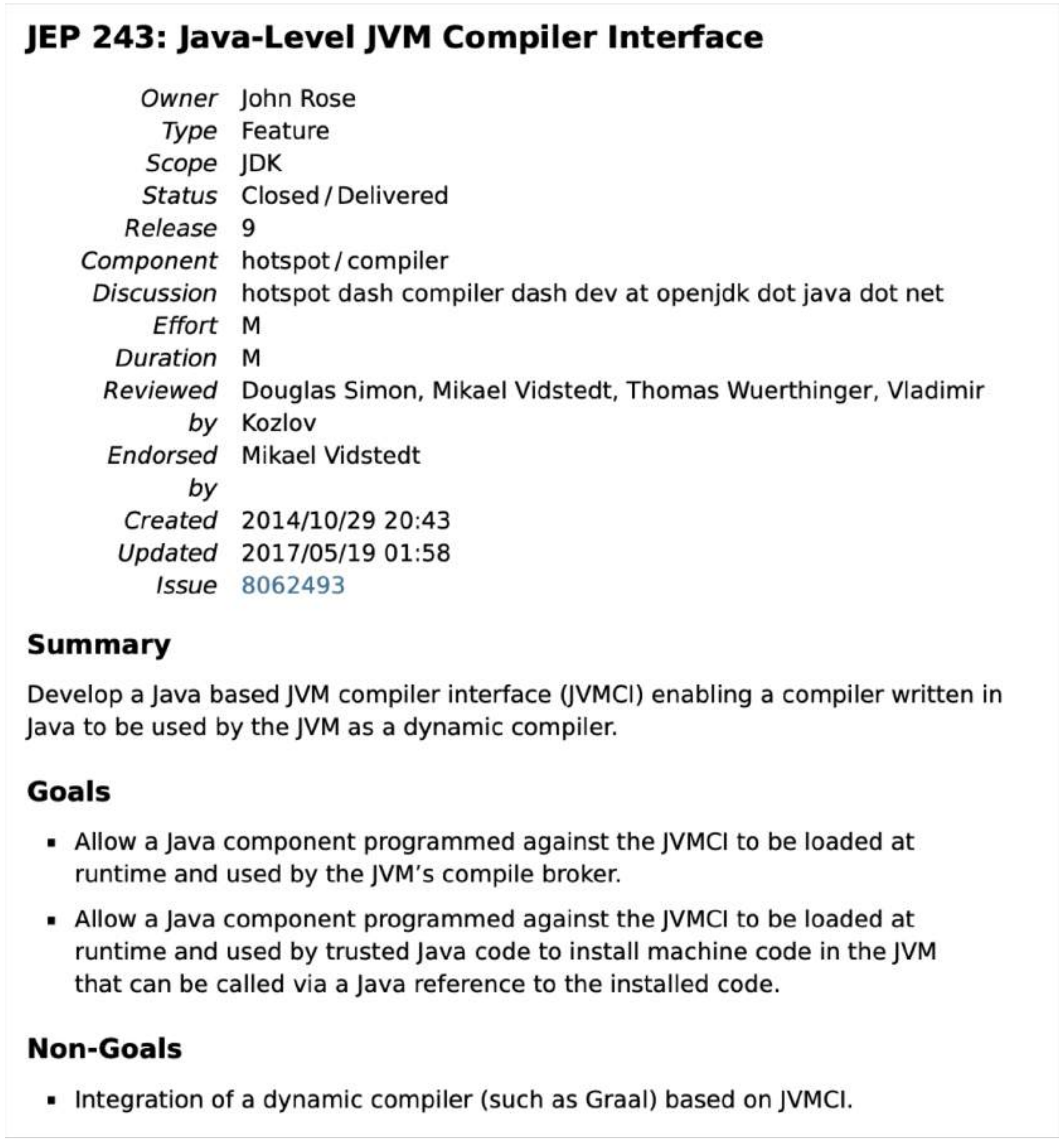
ci_common	Fixed test timeout.	24 days ago
ci_includes	Adjust CI job durations	3 months ago
docs	updated Debugging.md and improved help text for a few options	a day ago
mx.compiler	[GR-13655] Integrate Truffle libgraal.	9 days ago
src	From WordTypes, move dependency to InvokeCFunctionPointer to Substrat...	23 hours ago
LICENSE.md	Move graal-compiler to compiler directory	2 years ago
README.md	Review comments	17 days ago

README.md

Graal is a dynamic compiler written in Java that integrates with the HotSpot JVM. It has a focus on high performance and extensibility. In addition, it provides optimized performance for Truffle-based languages running on the JVM.

Setup

ORACLE



JEP 243: Java-Level JVM Compiler Interface

Owner	John Rose
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	9
Component	hotspot / compiler
Discussion	hotspot dash compiler dash dev at openjdk dot java dot net
Effort	M
Duration	M
Reviewed by	Douglas Simon, Mikael Vidstedt, Thomas Wuerthinger, Vladimir Kozlov
Endorsed by	Mikael Vidstedt
Created	2014/10/29 20:43
Updated	2017/05/19 01:58
Issue	8062493

Summary

Develop a Java based JVM compiler interface (JVMCI) enabling a compiler written in Java to be used by the JVM as a dynamic compiler.

Goals

- Allow a Java component programmed against the JVMCI to be loaded at runtime and used by the JVM's compile broker.
- Allow a Java component programmed against the JVMCI to be loaded at runtime and used by trusted Java code to install machine code in the JVM that can be called via a Java reference to the installed code.

Non-Goals

- Integration of a dynamic compiler (such as Graal) based on JVMCI.

```java
public byte[] compile(byte[] bytecode) {
  ...
}
```

```java
public byte[] compile(byte[] bytecode,
                                MetaData profiledData) {
  ...
}
```

```java
public void compile(byte[] bytecode,
                           MetaData profiledData,
                           CodeCache codeCache) {
  codeCache.install(...);
}
```

```
public void compile(Method method,
                    MetaData profiledData,
                    CodeCache codeCache) {
  codeCache.install(... method.getBytecode() ...);
}
```

```
public interface JVMCICompiler {
    CompilationRequestResult compileMethod(CompilationRequest request);
}
```

JVMCI

- An interface from a JVM to a compiler implemented in Java
- A bit like JVM agents
- Equivalent to CRI in Maxine
- Graal implements JVMCI
- Other compilers could as well
- And other JVM's could implement JVMCI
- Not really quite as simple as the interface suggests…
  - The compiler needs to know about the VM's object layout, deoptimization, safepoint mechanism, GC barriers, and more

Run on OpenJDK 11 with Graal enabled

```
$ java -XX:+UnlockExperimentalVMOptions \
       -XX:+EnableJVMCI \
       -XX:+UseJVMCICompiler \
       …
```

This is using Java 11, not GraalVM!

Run on OpenJDK 11 with a custom Graal build

```
$ java -XX:+UnlockExperimentalVMOptions \
       -XX:+EnableJVMCI \
       -XX:+UseJVMCICompiler \
       -Djvmci.class.path.append=graal.jar \
       …
```

This is using Java 11, and using a custom Graal build

Is this metacircularity for the sake of it?

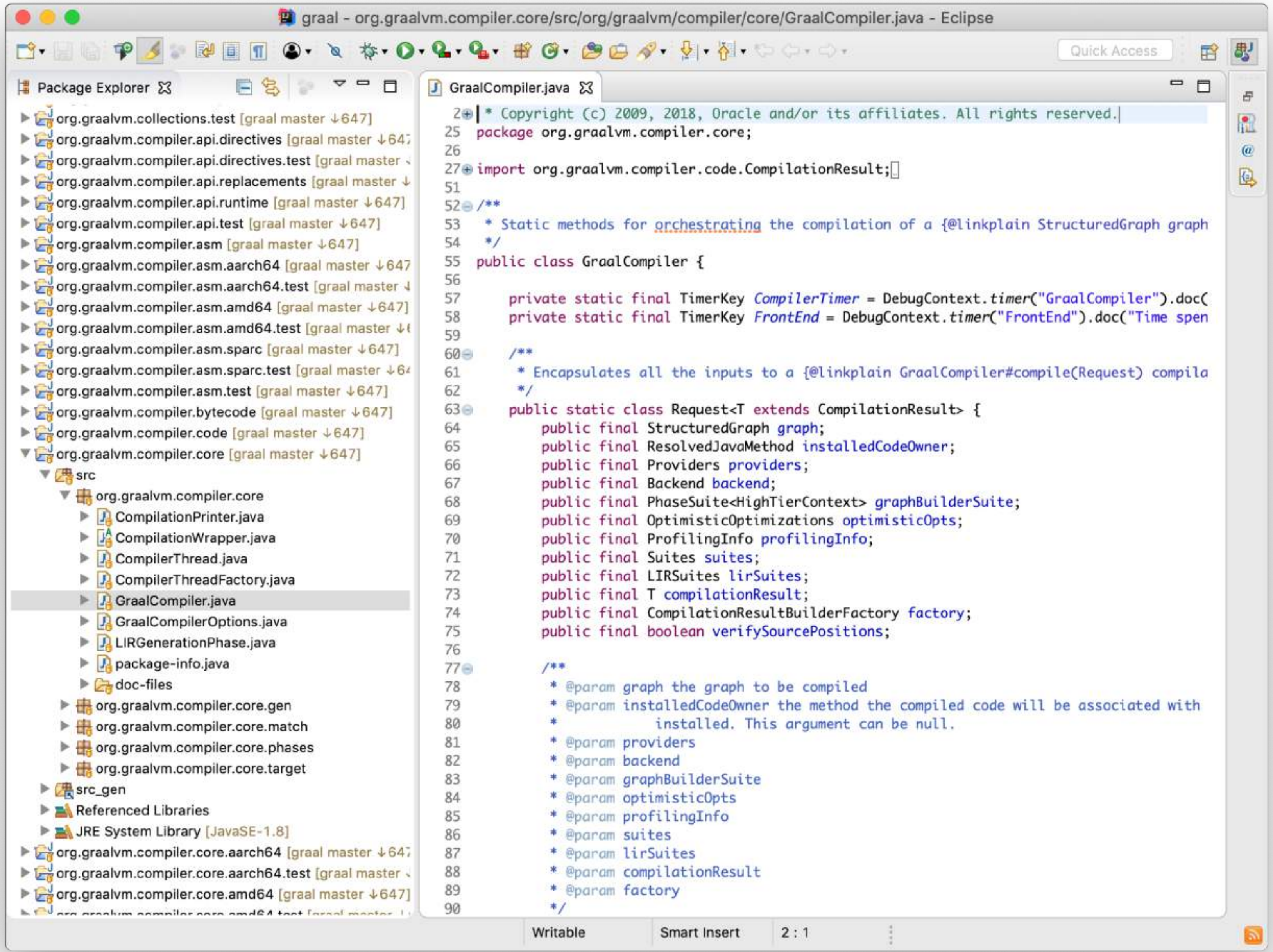
Maxine: An Approachable Virtual Machine For, and In, Java

CHRISTIAN WIMMER, MICHAEL HAUPT, MICHAEL L. VAN DE VANTER, MICK JORDAN, LAURENT DAYNÈS, and DOUGLAS SIMON, Oracle Labs

ACM Transactions on Architecture and Code Optimization, Vol. 9, No. 4, Article 30, Publication date: January 2013.

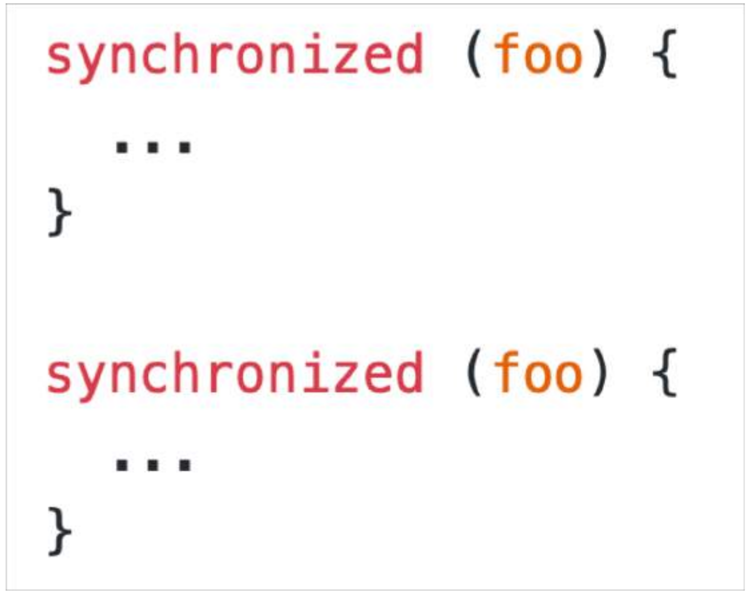
A highly productive platform accelerates the production of research results. The design of a Virtual Machine (VM) written in the Java^{TM} programming language can be simplified through exploitation of interfaces, type and memory safety, automated memory management (garbage collection), exception handling, and reflection. Moreover, modern Java IDEs offer time-saving features such as refactoring, auto-completion, and code navigation. Finally, Java annotations enable compiler extensions for low-level “systems programming” while retaining IDE compatibility. These techniques collectively make complex system software more “approachable” than has been typical in the past.

The Maxine VM, a metacircular Java VM implementation, has aggressively used these features since its inception. A co-designed companion tool, the Maxine Inspector, offers integrated debugging and visualization of all aspects of the VM’s runtime state. The Inspector’s implementation exploits advanced Java language features, embodies intimate knowledge of the VM’s design, and even reuses a significant amount of VM code directly. These characteristics make Maxine a highly approachable VM research platform and a productive basis for research and teaching.

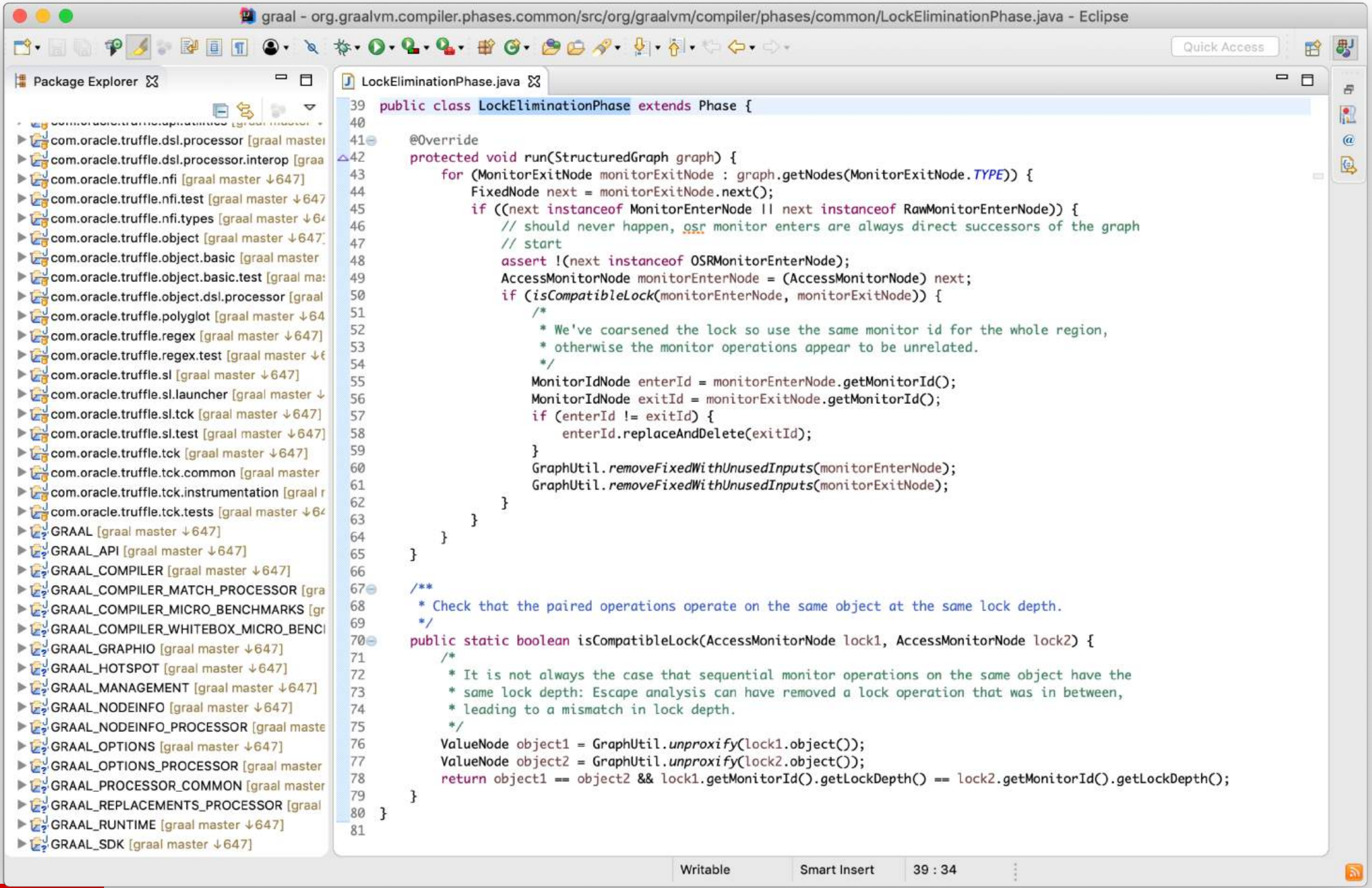

graal - org.graalvm.compiler.core/src/org/graalvm/compiler/core/GraalCompiler.java - Eclipse
Quick Access
Package Explorer
GraalCompiler.java
Writable
Smart Insert
2 : 1

```
org.graalvm.collections.test [graal master ↓647]
org.graalvm.compiler.api.directives [graal master ↓647
org.graalvm.compiler.api.directives.test [graal master
org.graalvm.compiler.api.replacements [graal master ↓
org.graalvm.compiler.api.runtime [graal master ↓647]
org.graalvm.compiler.api.test [graal master ↓647]
org.graalvm.compiler.asm [graal master ↓647]
org.graalvm.compiler.asm.aarch64 [graal master ↓647
org.graalvm.compiler.asm.aarch64.test [graal master ↓
org.graalvm.compiler.asm.amd64 [graal master ↓647]
org.graalvm.compiler.asm.amd64.test [graal master ↓
org.graalvm.compiler.asm.sparc [graal master ↓647]
org.graalvm.compiler.asm.sparc.test [graal master ↓64
org.graalvm.compiler.asm.test [graal master ↓647]
org.graalvm.compiler.bytecode [graal master ↓647]
org.graalvm.compiler.code [graal master ↓647]
org.graalvm.compiler.core [graal master ↓647]
  src
    org.graalvm.compiler.core
      CompilationPrinter.java
      CompilationWrapper.java
      CompilerThread.java
      CompilerThreadFactory.java
      GraalCompiler.java
      GraalCompilerOptions.java
      LIRGenerationPhase.java
      package-info.java
      doc-files
    org.graalvm.compiler.core.gen
    org.graalvm.compiler.core.match
    org.graalvm.compiler.core.phases
    org.graalvm.compiler.core.target
  src_gen
  Referenced Libraries
  JRE System Library [JavaSE-1.8]
org.graalvm.compiler.core.aarch64 [graal master ↓647
org.graalvm.compiler.core.aarch64.test [graal master
org.graalvm.compiler.core.amd64 [graal master ↓647]
```

```java
 * Copyright (c) 2009, 2018, Oracle and/or its affiliates. All rights reserved.
package org.graalvm.compiler.core;

import org.graalvm.compiler.code.CompilationResult;
/**
 * Static methods for orchestrating the compilation of a {@linkplain StructuredGraph graph
 */
public class GraalCompiler {

    private static final TimerKey CompilerTimer = DebugContext.timer("GraalCompiler").doc(
    private static final TimerKey FrontEnd = DebugContext.timer("FrontEnd").doc("Time spen

    /**
     * Encapsulates all the inputs to a {@linkplain GraalCompiler#compile(Request) compila
     */
    public static class Request<T extends CompilationResult> {
        public final StructuredGraph graph;
        public final ResolvedJavaMethod installedCodeOwner;
        public final Providers providers;
        public final Backend backend;
        public final PhaseSuite<HighTierContext> graphBuilderSuite;
        public final OptimisticOptimizations optimisticOpts;
        public final ProfilingInfo profilingInfo;
        public final Suites suites;
        public final LIRSuites lirSuites;
        public final T compilationResult;
        public final CompilationResultBuilderFactory factory;
        public final boolean verifySourcePositions;

        /**
         * @param graph the graph to be compiled
         * @param installedCodeOwner the method the compiled code will be associated with
         *            installed. This argument can be null.
         * @param providers
         * @param backend
         * @param graphBuilderSuite
         * @param optimisticOpts
         * @param profilingInfo
         * @param suites
         * @param lirSuites
         * @param compilationResult
         * @param factory
         */
```



```
synchronized (foo) {
  ...
}

synchronized (foo) {
  ...
}
```



graal - org.graalvm.compiler.phases.common/src/org/graalvm/compiler/phases/common/LockEliminationPhase.java - Eclipse

Package Explorer

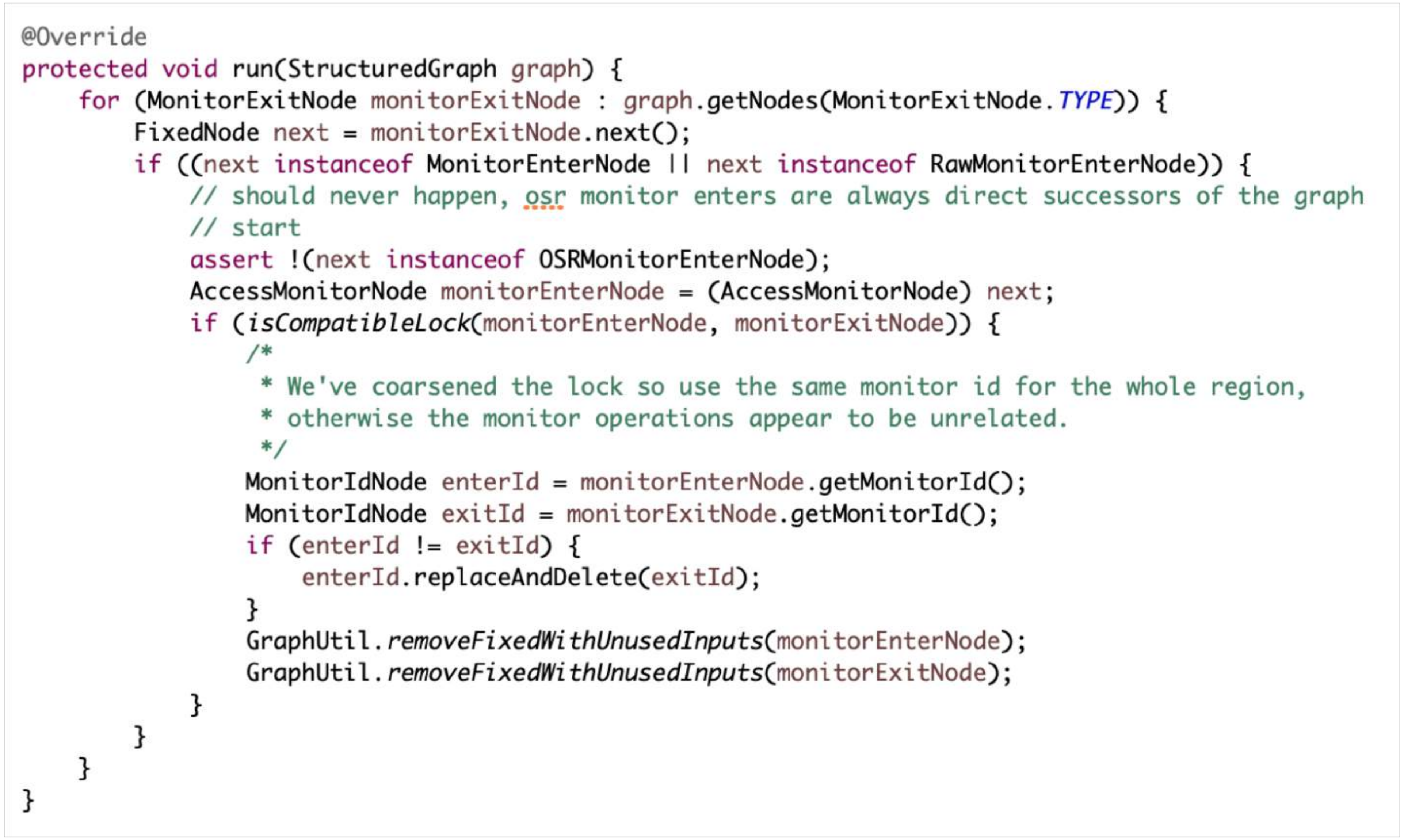
com.oracle.truffle.dsl.processor [graal master
com.oracle.truffle.dsl.processor.interop [graa
com.oracle.truffle.nfi [graal master ↓647]
com.oracle.truffle.nfi.test [graal master ↓647
com.oracle.truffle.nfi.types [graal master ↓64
com.oracle.truffle.object [graal master ↓647]
com.oracle.truffle.object.basic [graal master
com.oracle.truffle.object.basic.test [graal ma
com.oracle.truffle.object.dsl.processor [graal
com.oracle.truffle.polyglot [graal master ↓64
com.oracle.truffle.regex [graal master ↓647]
com.oracle.truffle.regex.test [graal master ↓6
com.oracle.truffle.sl [graal master ↓647]
com.oracle.truffle.sl.launcher [graal master ↓
com.oracle.truffle.sl.tck [graal master ↓647]
com.oracle.truffle.sl.test [graal master ↓647]
com.oracle.truffle.tck [graal master ↓647]
com.oracle.truffle.tck.common [graal master
com.oracle.truffle.tck.instrumentation [graal r
com.oracle.truffle.tck.tests [graal master ↓64
GRAAL [graal master ↓647]
GRAAL_API [graal master ↓647]
GRAAL_COMPILER [graal master ↓647]
GRAAL_COMPILER_MATCH_PROCESSOR [gra
GRAAL_COMPILER_MICRO_BENCHMARKS [gr
GRAAL_COMPILER_WHITEBOX_MICRO_BENC
GRAAL_GRAPHIO [graal master ↓647]
GRAAL_HOTSPOT [graal master ↓647]
GRAAL_MANAGEMENT [graal master ↓647]
GRAAL_NODEINFO [graal master ↓647]
GRAAL_NODEINFO_PROCESSOR [graal maste
GRAAL_OPTIONS [graal master ↓647]
GRAAL_OPTIONS_PROCESSOR [graal master
GRAAL_PROCESSOR_COMMON [graal master
GRAAL_REPLACEMENTS_PROCESSOR [graal
GRAAL_RUNTIME [graal master ↓647]
GRAAL_SDK [graal master ↓647]

LockEliminationPhase.java

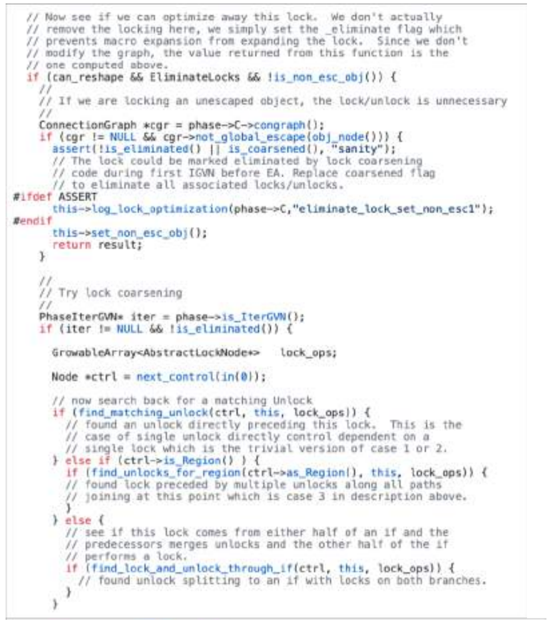
```java
public class LockEliminationPhase extends Phase {

    @Override
    protected void run(StructuredGraph graph) {
        for (MonitorExitNode monitorExitNode : graph.getNodes(MonitorExitNode.TYPE)) {
            FixedNode next = monitorExitNode.next();
            if ((next instanceof MonitorEnterNode || next instanceof RawMonitorEnterNode)) {
                // should never happen, osr monitor enters are always direct successors of the graph
                // start
                assert !(next instanceof OSRMonitorEnterNode);
                AccessMonitorNode monitorEnterNode = (AccessMonitorNode) next;
                if (isCompatibleLock(monitorEnterNode, monitorExitNode)) {
                    /*
                     * We've coarsened the lock so use the same monitor id for the whole region,
                     * otherwise the monitor operations appear to be unrelated.
                     */
                    MonitorIdNode enterId = monitorEnterNode.getMonitorId();
                    MonitorIdNode exitId = monitorExitNode.getMonitorId();
                    if (enterId != exitId) {
                        enterId.replaceAndDelete(exitId);
                    }
                    GraphUtil.removeFixedWithUnusedInputs(monitorEnterNode);
                    GraphUtil.removeFixedWithUnusedInputs(monitorExitNode);
                }
            }
        }
    }

    /**
     * Check that the paired operations operate on the same object at the same lock depth.
     */
    public static boolean isCompatibleLock(AccessMonitorNode lock1, AccessMonitorNode lock2) {
        /*
         * It is not always the case that sequential monitor operations on the same object have the
         * same lock depth: Escape analysis can have removed a lock operation that was in between,
         * leading to a mismatch in lock depth.
         */
        ValueNode object1 = GraphUtil.unproxify(lock1.object());
        ValueNode object2 = GraphUtil.unproxify(lock2.object());
        return object1 == object2 && lock1.getMonitorId().getLockDepth() == lock2.getMonitorId().getLockDepth();
    }
}
```

Writable | Smart Insert | 39 : 34

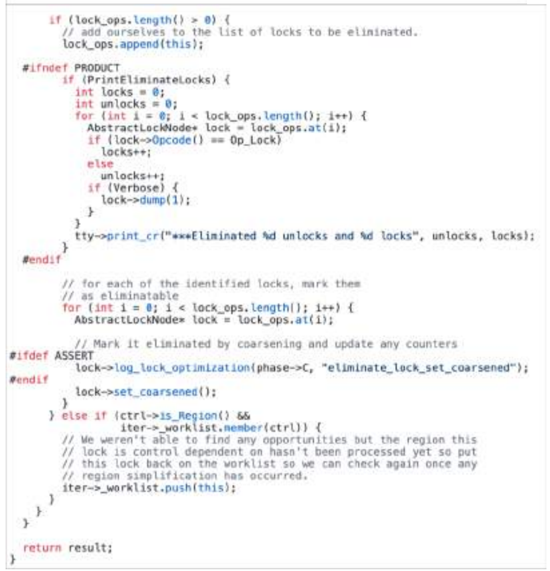
ORACLE®



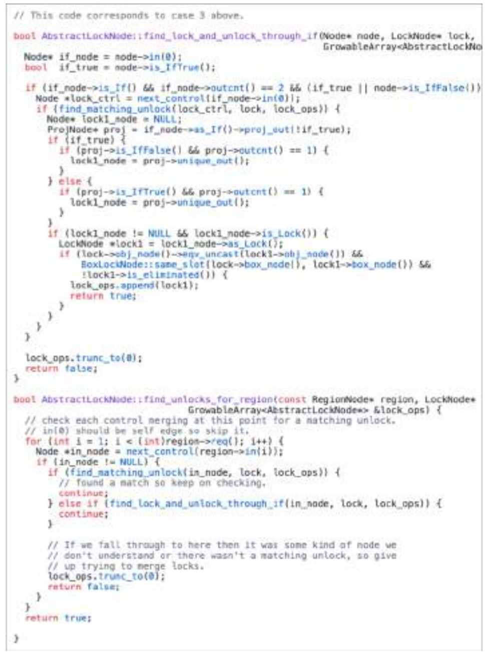
```
@Override
protected void run(StructuredGraph graph) {
    for (MonitorExitNode monitorExitNode : graph.getNodes(MonitorExitNode.TYPE)) {
        FixedNode next = monitorExitNode.next();
        if ((next instanceof MonitorEnterNode || next instanceof RawMonitorEnterNode)) {
            // should never happen, osr monitor enters are always direct successors of the graph
            // start
            assert !(next instanceof OSRMonitorEnterNode);
            AccessMonitorNode monitorEnterNode = (AccessMonitorNode) next;
            if (isCompatibleLock(monitorEnterNode, monitorExitNode)) {
                /*
                 * We've coarsened the lock so use the same monitor id for the whole region,
                 * otherwise the monitor operations appear to be unrelated.
                 */
                MonitorIdNode enterId = monitorEnterNode.getMonitorId();
                MonitorIdNode exitId = monitorExitNode.getMonitorId();
                if (enterId != exitId) {
                    enterId.replaceAndDelete(exitId);
                }
                GraphUtil.removeFixedWithUnusedInputs(monitorEnterNode);
                GraphUtil.removeFixedWithUnusedInputs(monitorExitNode);
            }
        }
    }
}
```



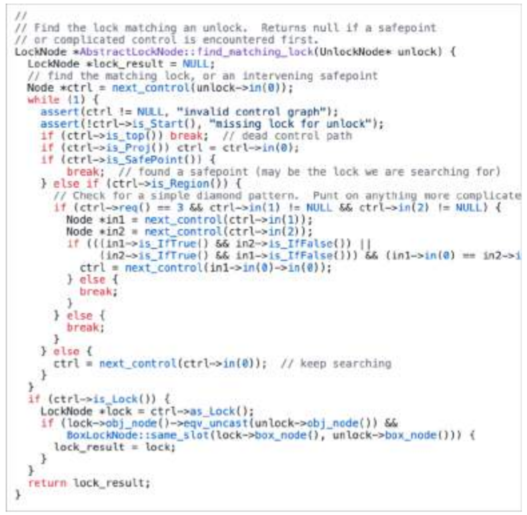
```
  // Now see if we can optimize away this lock.  We don't actually
  // remove the locking here, we simply set the _eliminate flag which
  // prevents macro expansion from expanding the lock.  Since we don't
  // modify the graph, the value returned from this function is the
  // one computed above.
  if (can_reshape && EliminateLocks && !is_non_esc_obj()) {
    //
    // If we are locking an unescaped object, the lock/unlock is unnecessary
    //
    ConnectionGraph *cgr = phase->C->congraph();
    if (cgr != NULL && cgr->not_global_escape(obj_node())) {
      assert(!is_eliminated() || is_coarsened(), "sanity");
      // The lock could be marked eliminated by lock coarsening
      // code during first IGVN before EA. Replace coarsened flag
      // to eliminate all associated locks/unlocks.
#ifdef ASSERT
      this->log_lock_optimization(phase->C,"eliminate_lock_set_non_esc1");
#endif
      this->set_non_esc_obj();
      return result;
    }

    //
    // Try lock coarsening
    //
    PhaseIterGVN* iter = phase->is_IterGVN();
    if (iter != NULL && !is_eliminated()) {

      GrowableArray<AbstractLockNode*>   lock_ops;

      Node *ctrl = next_control(in(0));

      // now search back for a matching Unlock
      if (find_matching_unlock(ctrl, this, lock_ops)) {
        // found an unlock directly preceding this lock.  This is the
        // case of single unlock directly control dependent on a
        // single lock which is the trivial version of case 1 or 2.
      } else if (ctrl->is_Region() ) {
        if (find_unlocks_for_region(ctrl->as_Region(), this, lock_ops)) {
        // found lock preceded by multiple unlocks along all paths
        // joining at this point which is case 3 in description above.
        }
      } else {
        // see if this lock comes from either half of an if and the
        // predecessors merges unlocks and the other half of the if
        // performs a lock.
        if (find_lock_and_unlock_through_if(ctrl, this, lock_ops)) {
          // found unlock splitting to an if with locks on both branches.
        }
      }
```



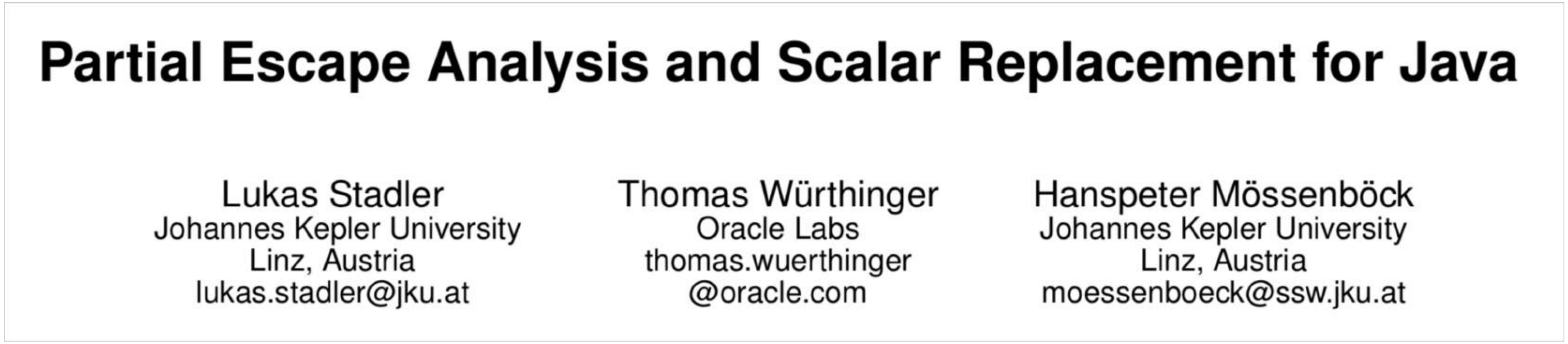
```
      if (lock_ops.length() > 0) {
        // add ourselves to the list of locks to be eliminated.
        lock_ops.append(this);

  #ifndef PRODUCT
        if (PrintEliminateLocks) {
          int locks = 0;
          int unlocks = 0;
          for (int i = 0; i < lock_ops.length(); i++) {
            AbstractLockNode* lock = lock_ops.at(i);
            if (lock->Opcode() == Op_Lock)
              locks++;
            else
              unlocks++;
            if (Verbose) {
              lock->dump(1);
            }
          }
          tty->print_cr("***Eliminated %d unlocks and %d locks", unlocks, locks);
        }
  #endif

        // for each of the identified locks, mark them
        // as eliminatable
        for (int i = 0; i < lock_ops.length(); i++) {
          AbstractLockNode* lock = lock_ops.at(i);

          // Mark it eliminated by coarsening and update any counters
#ifdef ASSERT
          lock->log_lock_optimization(phase->C, "eliminate_lock_set_coarsened");
#endif
          lock->set_coarsened();
        }
      } else if (ctrl->is_Region() &&
                 iter->_worklist.member(ctrl)) {
        // We weren't able to find any opportunities but the region this
        // lock is control dependent on hasn't been processed yet so put
        // this lock back on the worklist so we can check again once any
        // region simplification has occurred.
        iter->_worklist.push(this);
      }
    }
  }

  return result;
}
```



```
// This code corresponds to case 3 above.

bool AbstractLockNode::find_lock_and_unlock_through_if(Node* node, LockNode* lock,
                                                        GrowableArray<AbstractLockNo
  Node* if_node = node->in(0);
  bool  if_true = node->is_IfTrue();

  if (if_node->is_If() && if_node->outcnt() == 2 && (if_true || node->is_IfFalse())
    Node *lock_ctrl = next_control(if_node->in(0));
    if (find_matching_unlock(lock_ctrl, lock, lock_ops)) {
      Node* lock1_node = NULL;
      ProjNode* proj = if_node->as_If()->proj_out(!if_true);
      if (if_true) {
        if (proj->is_IfFalse() && proj->outcnt() == 1) {
          lock1_node = proj->unique_out();
        }
      } else {
        if (proj->is_IfTrue() && proj->outcnt() == 1) {
          lock1_node = proj->unique_out();
        }
      }
      if (lock1_node != NULL && lock1_node->is_Lock()) {
        LockNode *lock1 = lock1_node->as_Lock();
        if (lock->obj_node()->eqv_uncast(lock1->obj_node()) &&
            BoxLockNode::same_slot(lock->box_node(), lock1->box_node()) &&
            !lock1->is_eliminated()) {
          lock_ops.append(lock1);
          return true;
        }
      }
    }
  }

  lock_ops.trunc_to(0);
  return false;
}

bool AbstractLockNode::find_unlocks_for_region(const RegionNode* region, LockNode*
                                               GrowableArray<AbstractLockNode*> &lock_ops) {
  // check each control merging at this point for a matching unlock.
  // in(0) should be self edge so skip it.
  for (int i = 1; i < (int)region->req(); i++) {
    Node *in_node = next_control(region->in(i));
    if (in_node != NULL) {
      if (find_matching_unlock(in_node, lock, lock_ops)) {
        // found a match so keep on checking.
        continue;
      } else if (find_lock_and_unlock_through_if(in_node, lock, lock_ops)) {
        continue;
      }

      // If we fall through to here then it was some kind of node we
      // don't understand or there wasn't a matching unlock, so give
      // up trying to merge locks.
      lock_ops.trunc_to(0);
      return false;
    }
  }
  return true;

}
```



```
//
// Find the lock matching an unlock.  Returns null if a safepoint
// or complicated control is encountered first.
LockNode *AbstractLockNode::find_matching_lock(UnlockNode* unlock) {
  LockNode *lock_result = NULL;
  // find the matching lock, or an intervening safepoint
  Node *ctrl = next_control(unlock->in(0));
  while (1) {
    assert(ctrl != NULL, "invalid control graph");
    assert(!ctrl->is_Start(), "missing lock for unlock");
    if (ctrl->is_top()) break;  // dead control path
    if (ctrl->is_Proj()) ctrl = ctrl->in(0);
    if (ctrl->is_SafePoint()) {
        break;  // found a safepoint (may be the lock we are searching for)
    } else if (ctrl->is_Region()) {
      // Check for a simple diamond pattern.  Punt on anything more complicate
      if (ctrl->req() == 3 && ctrl->in(1) != NULL && ctrl->in(2) != NULL) {
        Node *in1 = next_control(ctrl->in(1));
        Node *in2 = next_control(ctrl->in(2));
        if (((in1->is_IfTrue() && in2->is_IfFalse()) ||
             (in2->is_IfTrue() && in1->is_IfFalse())) && (in1->in(0) == in2->i
          ctrl = next_control(in1->in(0)->in(0));
        } else {
          break;
        }
      } else {
        break;
      }
    } else {
      ctrl = next_control(ctrl->in(0));  // keep searching
    }
  }
  if (ctrl->is_Lock()) {
    LockNode *lock = ctrl->as_Lock();
    if (lock->obj_node()->eqv_uncast(unlock->obj_node()) &&
        BoxLockNode::same_slot(lock->box_node(), unlock->box_node())) {
      lock_result = lock;
    }
  }
  return lock_result;
}
```



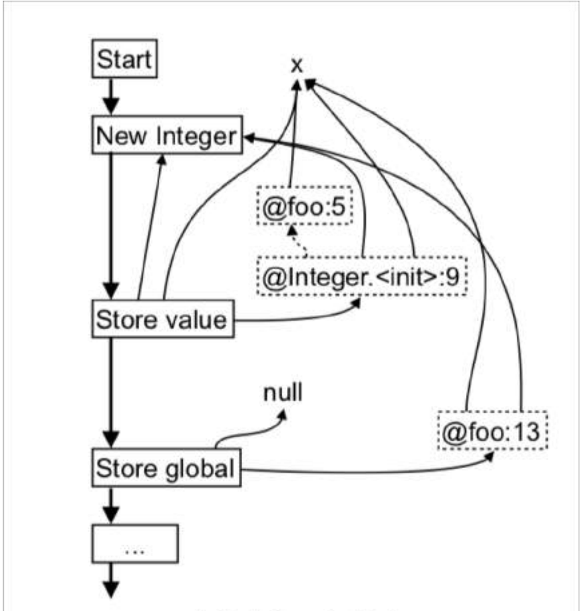
Partial Escape Analysis and Scalar Replacement for Java

Lukas Stadler
Johannes Kepler University
Linz, Austria
lukas.stadler@jku.at

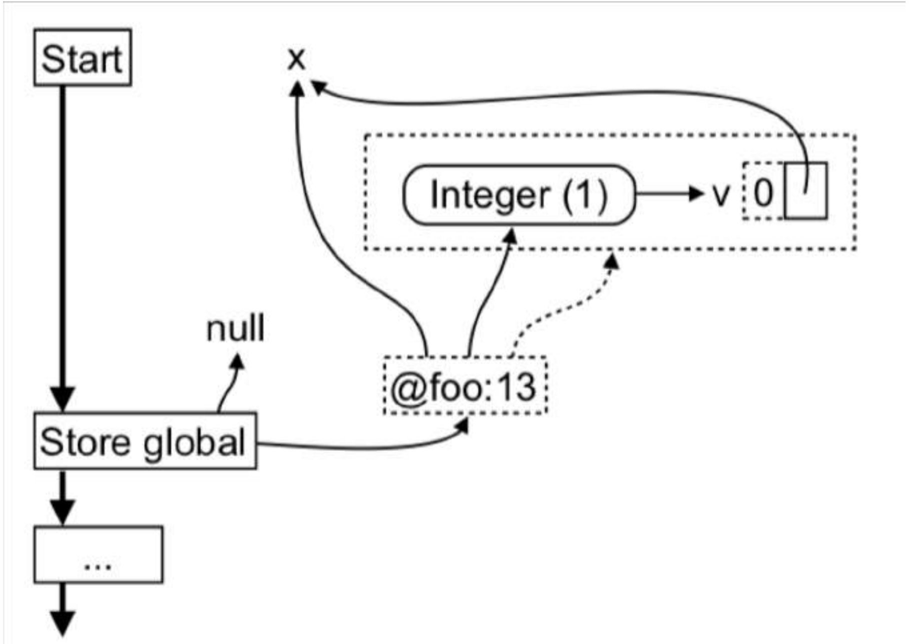
Thomas Würthinger
Oracle Labs
thomas.wuerthinger
@oracle.com

Hanspeter Mössenböck
Johannes Kepler University
Linz, Austria
moessenboeck@ssw.jku.at

```
static Object global;
void foo(int x) {
  Integer i = new Integer(x);
  global = null;
  ...
}
```



(a) After inlining.



(b) After Partial Escape Analysis.

Partial Escape Analysis and Scalar Replacement for Java

Lukas Stadler
Johannes Kepler University
Linz, Austria
lukas.stadler@jku.at

Thomas Würthinger
Oracle Labs
thomas.wuerthinger
@oracle.com

Hanspeter Mössenböck
Johannes Kepler University
Linz, Austria
moessenboeck@ssw.jku.at

Trace-based Register Allocation in a JIT Compiler

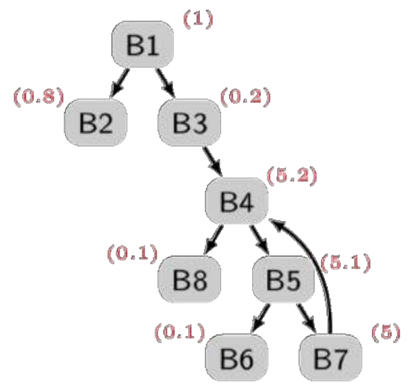
Josef Eisl[†]
Institute for System Software
Johannes Kepler University
Linz, Austria

Matthias Grimmer[†]
Institute for System Software
Johannes Kepler University
Linz, Austria

Doug Simon[‡]
Oracle Labs
Switzerland

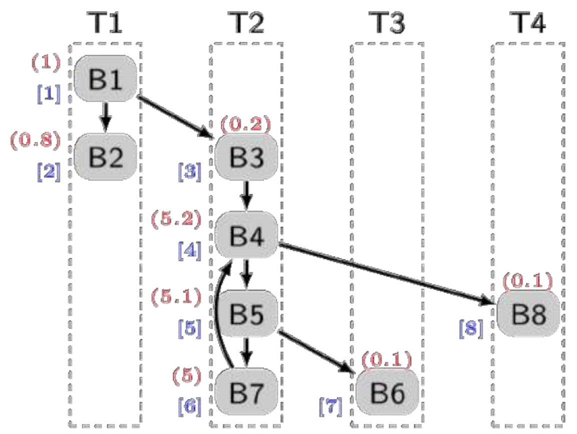
Thomas Würthinger[‡]
Oracle Labs
Switzerland

Hanspeter Mössenböck[†]
Institute for System Software
Johannes Kepler University
Linz, Austria

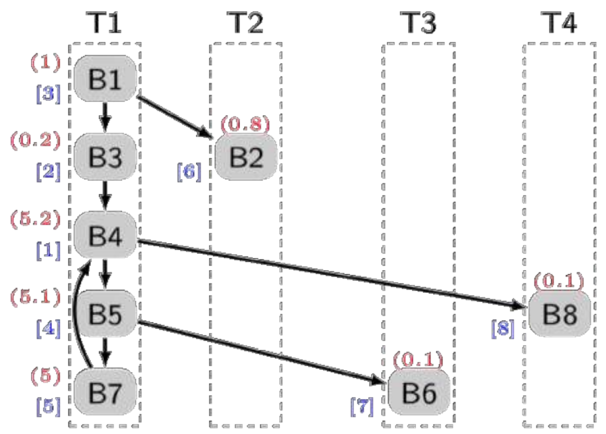


(a) Control-flow Graph

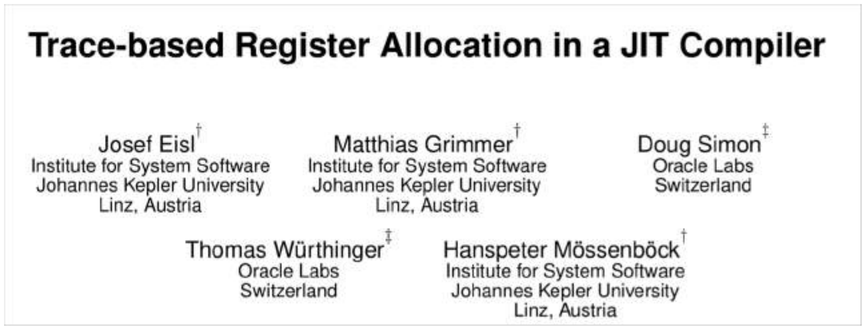
```
boolean equals(int[] a, int[] b) {
  /*B1*/  if (b.length != a.length)
  /*B2*/      return false;
  /*B3*/  int i = 0;
  /*B4*/  while (i < a.length) {
  /*B5*/      if (a[i] != b[i])
  /*B6*/          return false;
  /*B7*/      i++;
          }
  /*B8*/  return true;
}
```
(b) Java Source



(c) Unidirectional Trace Builder



(d) Bidirectional Trace Builder


Trace-based Register Allocation in a JIT Compiler

Josef Eisl[†]
Institute for System Software
Johannes Kepler University
Linz, Austria

Matthias Grimmer[†]
Institute for System Software
Johannes Kepler University
Linz, Austria

Doug Simon[‡]
Oracle Labs
Switzerland

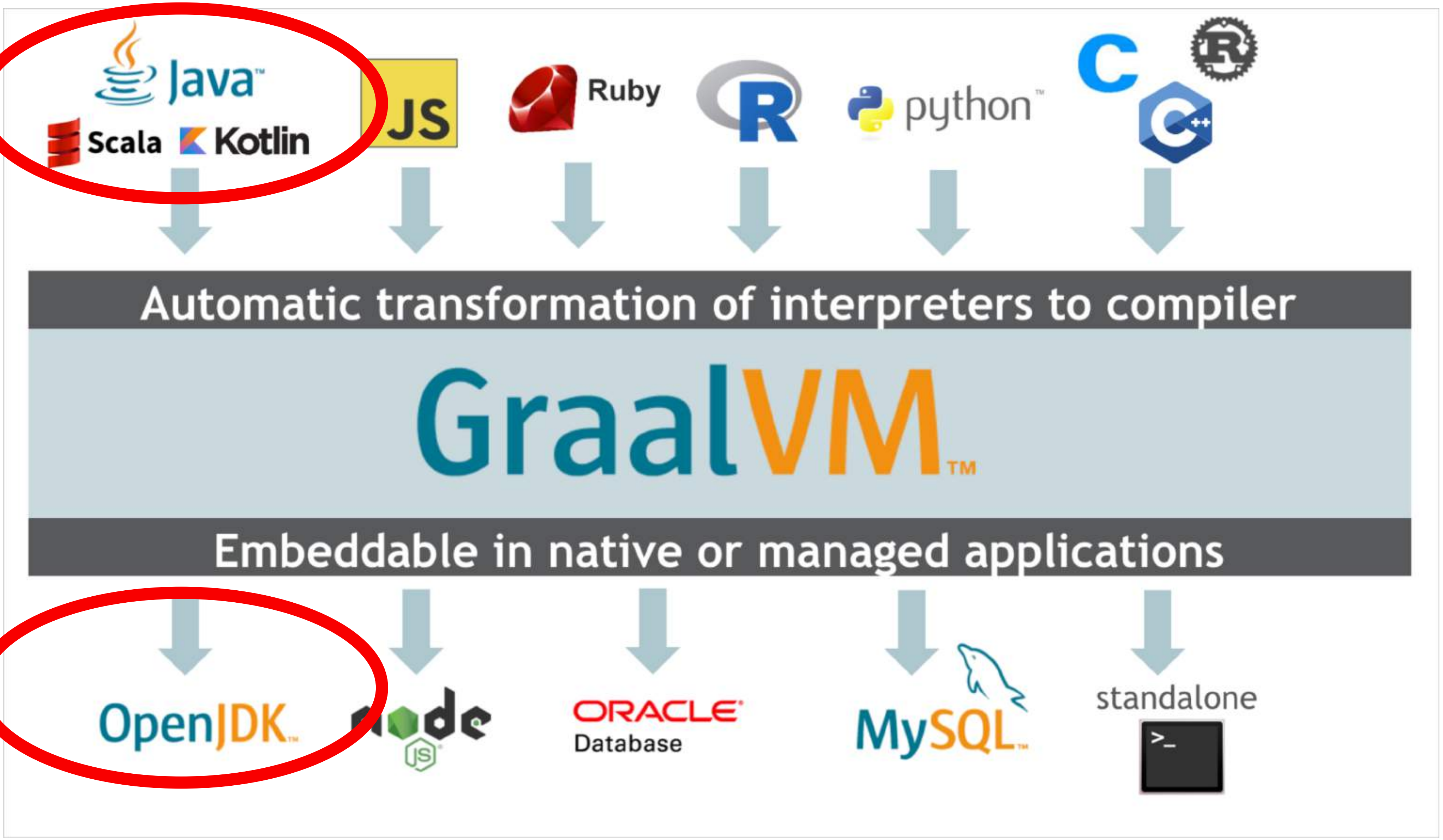
Thomas Würthinger[‡]
Oracle Labs
Switzerland

Hanspeter Mössenböck[†]
Institute for System Software
Johannes Kepler University
Linz, Austria

What are the applications of Graal?

What are the applications of Graal?

Graal as a JIT compiler for JVM languages


Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_

```java
public static void main(String[] args) {
    Arrays.stream(args)
            .flatMap(TopTen::fileLines)
            .flatMap(line -> Arrays.stream(line.split("\\b")))
            .map(word -> word.replaceAll("[^a-zA-Z]", ""))
            .filter(word -> word.length() > 0)
            .map(word -> word.toLowerCase())
            .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()))
            .entrySet().stream()
            .sorted((a, b) -> -a.getValue().compareTo(b.getValue()))
            .limit(10)
            .forEach(e -> System.out.format("%s = %d%n", e.getKey(), e.getValue()));
}
```

Running using the GraalVM EE distribution

```
$ javac TopTen.java
$ time java TopTen large.txt
…
real 0m18.905s
```

Compare to standard OpenJDK

```
$ time java -XX:-UseJVMCICompiler TopTen large.txt
…
real 0m23.102s
```

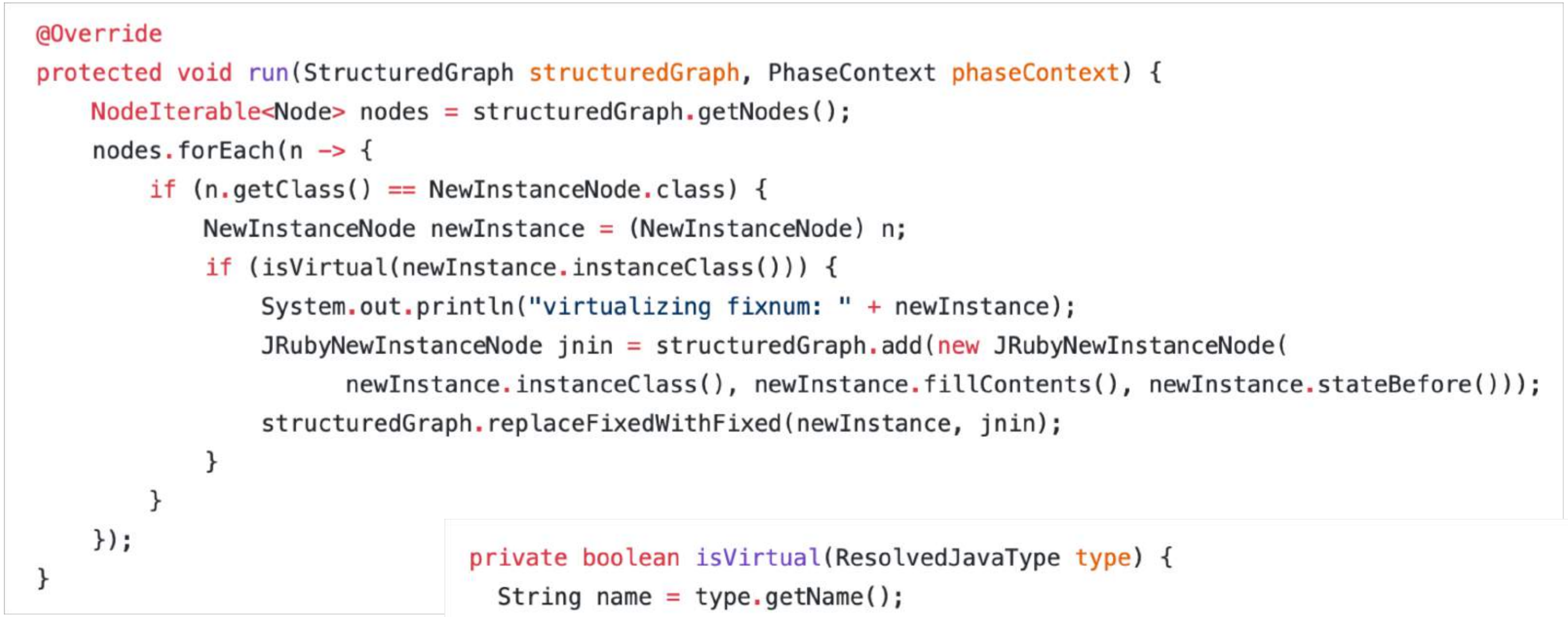
What are the applications of Graal?

Graal as a specialized compiler for JVM applications

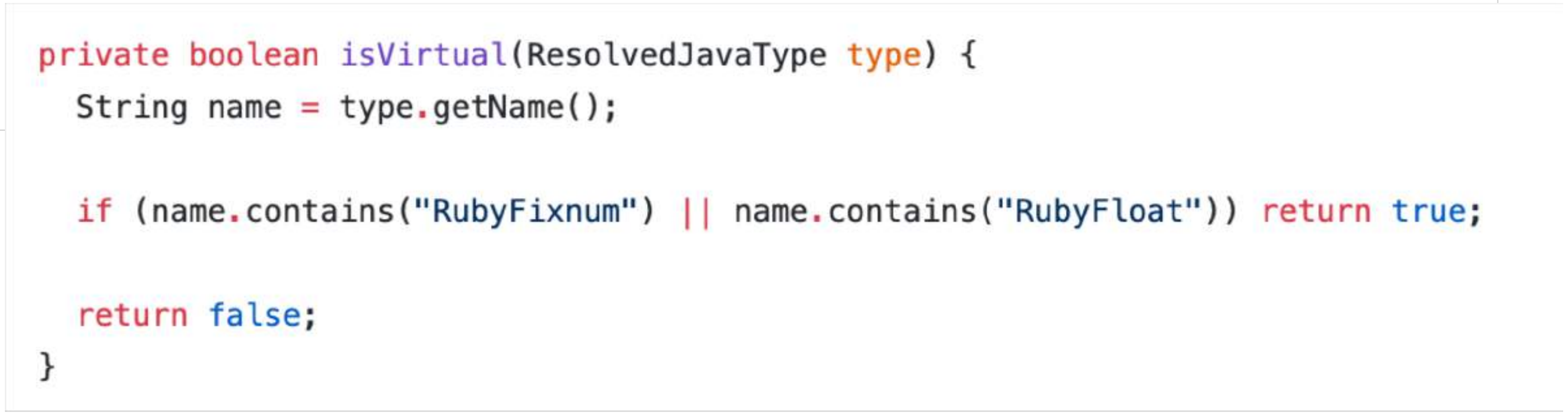
Graal as a specialized compiler for JVM applications

- Graal is modular, and the JVMCI has a plug-in architecture
- Can we write custom optimization passes for specific applications and plug them into Graal?
  - Optimizations that only make sense for your application?
  - Optimizations that break the normal rules of the JVM Spec but you're happy they're safe for your application?
- github.com/jruby/jruby-graal

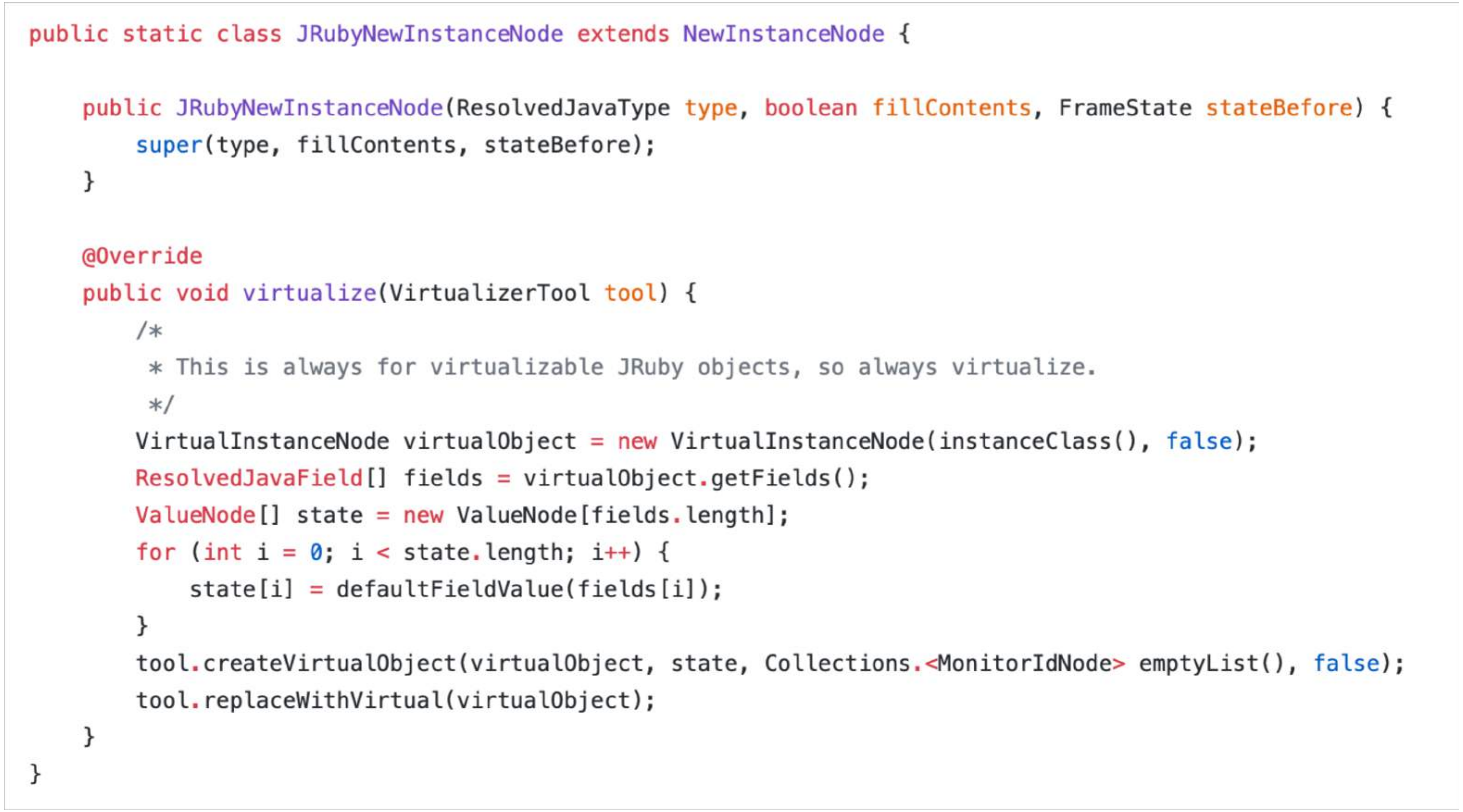
```java
public static class JRubyGraalCompilerConfiguration extends CoreCompilerConfiguration {
    @Override
    public PhaseSuite<HighTierContext> createHighTier(OptionValues options) {
        HighTier highTier = new HighTier(options);
        ListIterator iter = highTier.findPhase(PartialEscapePhase.class);
        iter.previous();
        iter.add(new JRubyVirtualizationPhase());
        return highTier;
    }
}
```



```java
@Override
protected void run(StructuredGraph structuredGraph, PhaseContext phaseContext) {
    NodeIterable<Node> nodes = structuredGraph.getNodes();
    nodes.forEach(n -> {
        if (n.getClass() == NewInstanceNode.class) {
            NewInstanceNode newInstance = (NewInstanceNode) n;
            if (isVirtual(newInstance.instanceClass())) {
                System.out.println("virtualizing fixnum: " + newInstance);
                JRubyNewInstanceNode jnin = structuredGraph.add(new JRubyNewInstanceNode(
                      newInstance.instanceClass(), newInstance.fillContents(), newInstance.stateBefore()));
                structuredGraph.replaceFixedWithFixed(newInstance, jnin);
            }
        }
    });
}
```



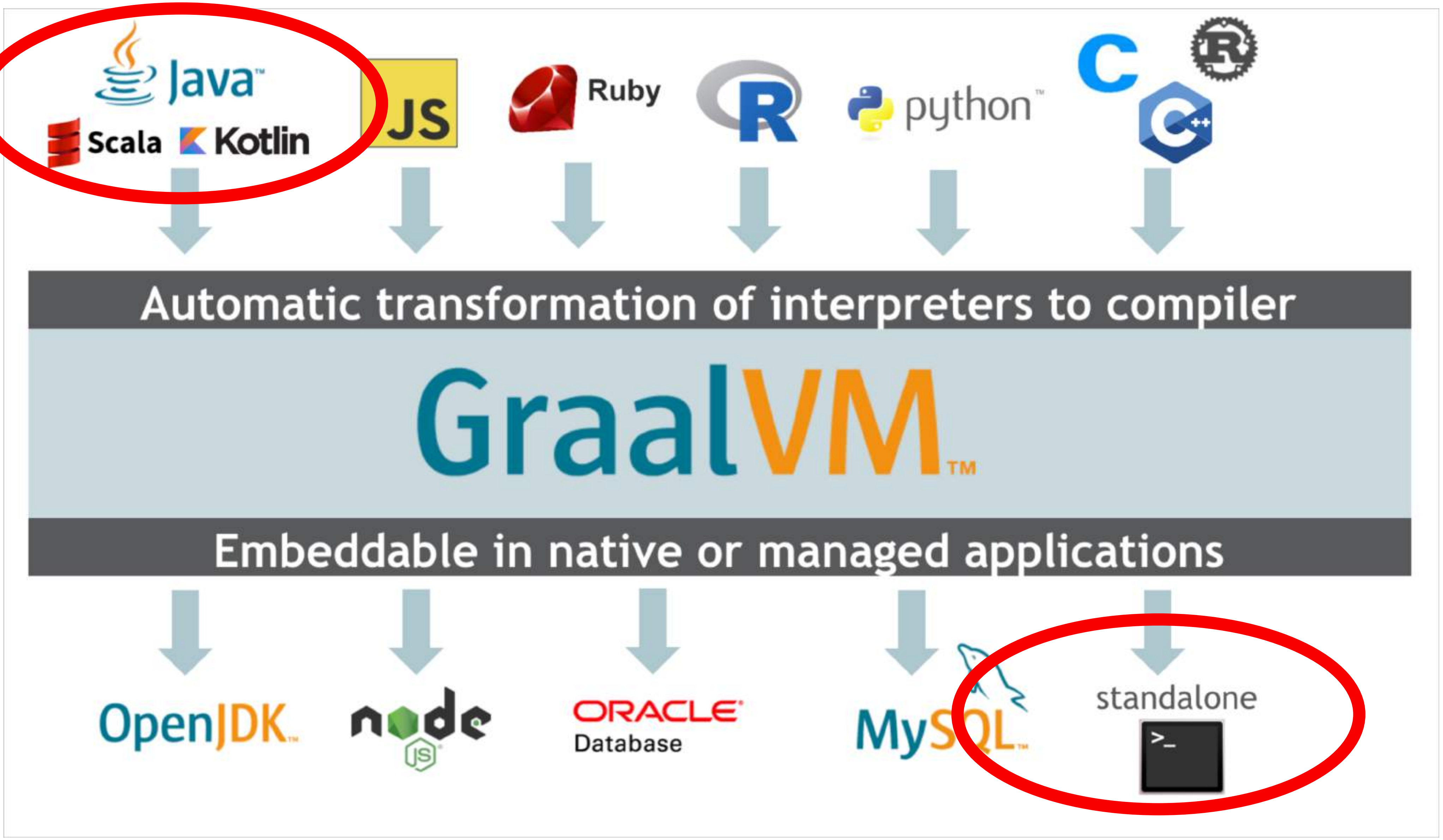
```java
private boolean isVirtual(ResolvedJavaType type) {
  String name = type.getName();

  if (name.contains("RubyFixnum") || name.contains("RubyFloat")) return true;

  return false;
}
```



```
public static class JRubyNewInstanceNode extends NewInstanceNode {

    public JRubyNewInstanceNode(ResolvedJavaType type, boolean fillContents, FrameState stateBefore) {
        super(type, fillContents, stateBefore);
    }

    @Override
    public void virtualize(VirtualizerTool tool) {
        /*
         * This is always for virtualizable JRuby objects, so always virtualize.
         */
        VirtualInstanceNode virtualObject = new VirtualInstanceNode(instanceClass(), false);
        ResolvedJavaField[] fields = virtualObject.getFields();
        ValueNode[] state = new ValueNode[fields.length];
        for (int i = 0; i < state.length; i++) {
            state[i] = defaultFieldValue(fields[i]);
        }
        tool.createVirtualObject(virtualObject, state, Collections.<MonitorIdNode> emptyList(), false);
        tool.replaceWithVirtual(virtualObject);
    }
}
```

What are the applications of Graal?

Graal as an AOT compiler for JVM languages

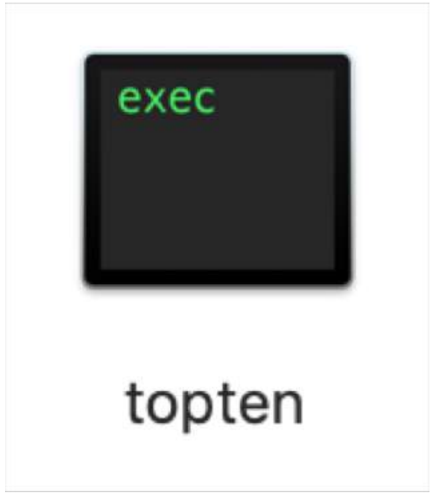

Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_

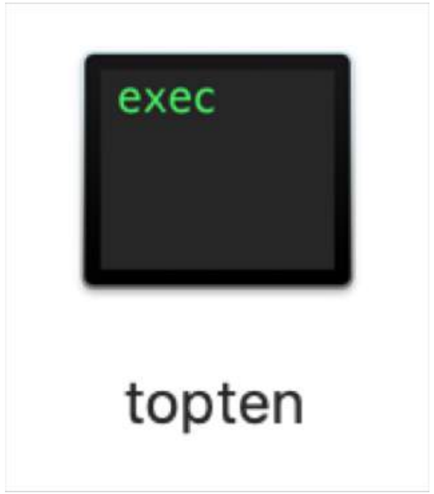
Run as normal

```
$ time java TopTen small.txt
…
real 0m0.408s
```

Compile to native using native-image

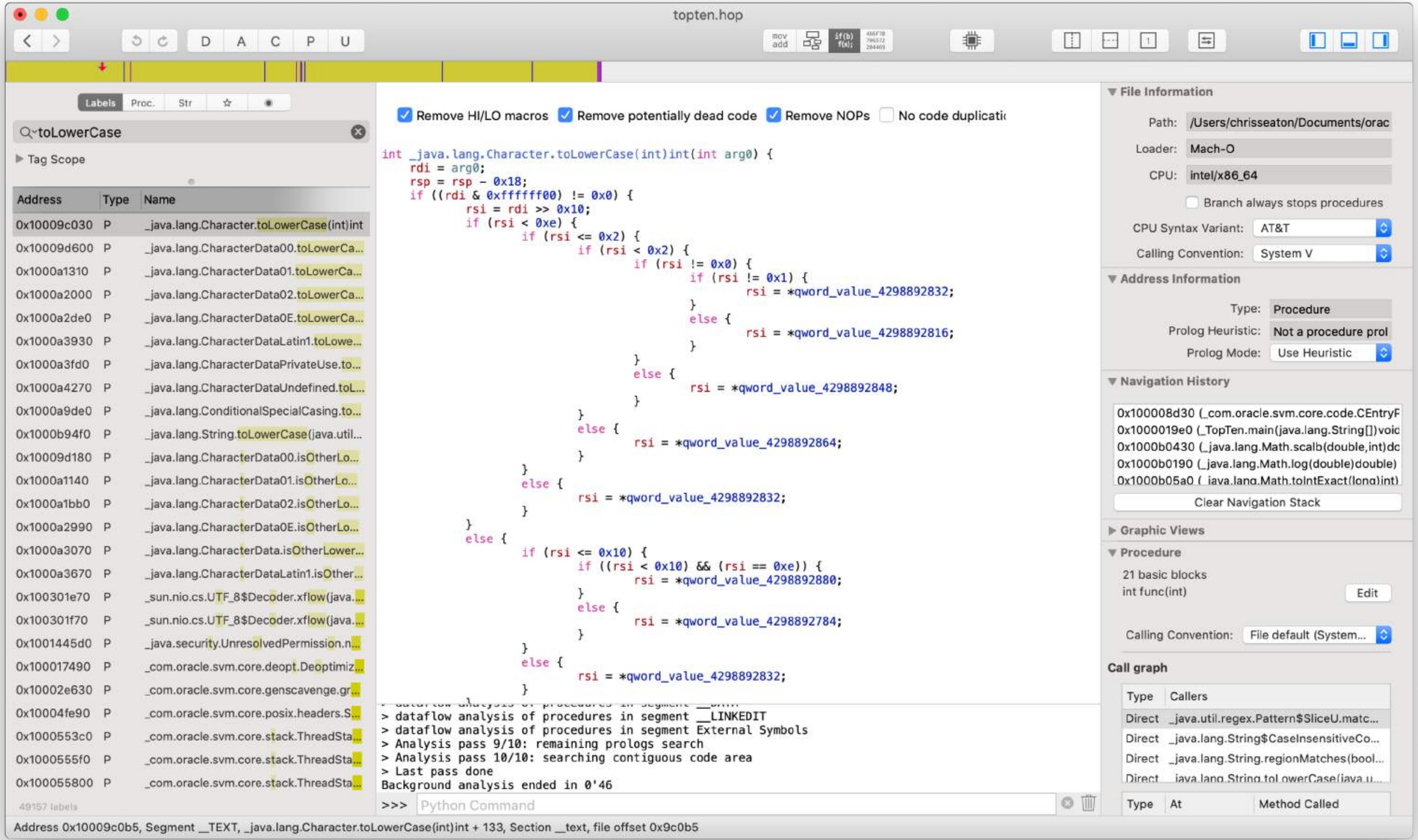
```
$ native-image TopTen
…
$ time ./topten small.txt
…
real 0m0.112s
```


exec
topten



```
$ du -h topten
8.8M topten
```

```
$ otool -L topten
topten:
/usr/lib/libSystem.B.dylib
/usr/lib/libz.1.dylib
/System/Library/CoreFoundation
```



Address	Type	Name
0x10009c030	P	_java.lang.Character.toLowerCase(int)int
0x10009d600	P	_java.lang.CharacterData00.toLowerCa...
0x1000a1310	P	_java.lang.CharacterData01.toLowerCa...
0x1000a2000	P	_java.lang.CharacterData02.toLowerCa...
0x1000a2de0	P	_java.lang.CharacterData0E.toLowerCa...
0x1000a3930	P	_java.lang.CharacterDataLatin1.toLowe...
0x1000a3fd0	P	_java.lang.CharacterDataPrivateUse.to...
0x1000a4270	P	_java.lang.CharacterDataUndefined.toL...
0x1000a9de0	P	_java.lang.ConditionalSpecialCasing.to...
0x1000b94f0	P	_java.lang.String.toLowerCase(java.util...
0x10009d180	P	_java.lang.CharacterData00.isOtherLo...
0x1000a1140	P	_java.lang.CharacterData01.isOtherLo...
0x1000a1bb0	P	_java.lang.CharacterData02.isOtherLo...
0x1000a2990	P	_java.lang.CharacterData0E.isOtherLo...
0x1000a3070	P	_java.lang.CharacterData.isOtherLower...
0x1000a3670	P	_java.lang.CharacterDataLatin1.isOther...
0x100301e70	P	_sun.nio.cs.UTF_8$Decoder.xflow(java....
0x100301f70	P	_sun.nio.cs.UTF_8$Decoder.xflow(java....
0x1001445d0	P	_java.security.UnresolvedPermission.n...
0x100017490	P	_com.oracle.svm.core.deopt.Deoptimiz...
0x10002e630	P	_com.oracle.svm.core.genscavenge.gr...
0x10004fe90	P	_com.oracle.svm.core.posix.headers.S...
0x1000553c0	P	_com.oracle.svm.core.stack.ThreadSta...
0x1000555f0	P	_com.oracle.svm.core.stack.ThreadSta...
0x100055800	P	_com.oracle.svm.core.stack.ThreadSta...

49157 labels

[✓] Remove HI/LO macros [✓] Remove potentially dead code [✓] Remove NOPs [ ] No code duplicati

```
int _java.lang.Character.toLowerCase(int)int(int arg0) {
    rdi = arg0;
    rsp = rsp - 0x18;
    if ((rdi & 0xffffff00) != 0x0) {
            rsi = rdi >> 0x10;
            if (rsi < 0xe) {
                    if (rsi <= 0x2) {
                            if (rsi < 0x2) {
                                    if (rsi != 0x0) {
                                            if (rsi != 0x1) {
                                                    rsi = *qword_value_4298892832;
                                            }
                                            else {
                                                    rsi = *qword_value_4298892816;
                                            }
                                    }
                                    else {
                                            rsi = *qword_value_4298892848;
                                    }
                            }
                            else {
                                    rsi = *qword_value_4298892864;
                            }
                    }
                    else {
                            rsi = *qword_value_4298892832;
                    }
            }
            else {
                    if (rsi <= 0x10) {
                            if ((rsi < 0x10) && (rsi == 0xe)) {
                                    rsi = *qword_value_4298892880;
                            }
                            else {
                                    rsi = *qword_value_4298892784;
                            }
                    }
                    else {
                            rsi = *qword_value_4298892832;
                    }
```

```
> dataflow analysis of procedures in segment __LINKEDIT
> dataflow analysis of procedures in segment External Symbols
> Analysis pass 9/10: remaining prologs search
> Analysis pass 10/10: searching contiguous code area
> Last pass done
Background analysis ended in 0'46
>>>
```

File Information

Path: /Users/chrisseaton/Documents/orac
Loader: Mach-O
CPU: intel/x86_64
[ ] Branch always stops procedures
CPU Syntax Variant: AT&T
Calling Convention: System V

Address Information

Type: Procedure
Prolog Heuristic: Not a procedure prol
Prolog Mode: Use Heuristic

Navigation History

0x100008d30 (_com.oracle.svm.core.code.CEntryF
0x1000019e0 (_TopTen.main(java.lang.String[])voic
0x1000b0430 (_java.lang.Math.scalb(double,int)dc
0x1000b0190 (_java.lang.Math.log(double)double)
0x1000b05a0 (_java.lang.Math.toIntExact(long)int)

Graphic Views

Procedure

21 basic blocks
int func(int)
Calling Convention: File default (System...

Call graph

Type	Callers
Direct	_java.util.regex.Pattern$SliceU.matc...
Direct	_java.lang.String$CaseInsensitiveCo...
Direct	_java.lang.String.regionMatches(bool...
Direct	_java.lang.String.toLowerCase(java.u...

Type	At	Method Called

Address 0x10009c0b5, Segment __TEXT, _java.lang.Character.toLowerCase(int)int + 133, Section __text, file offset 0x9c0b5

SubstrateVM using Graal

- The native-image tool is using SubstrateVM and Graal
- SubstrateVM does closed-world analysis on a Java application
- Produces a set of methods to compile
- Runs Graal to compile them
  - (literally calls a method called `GraalCompiler.compile`)
  - Replaces the in-memory code-cache with an on-disk code cache
- Details around configuring for static code, but the big idea translates pretty literally to the code

Maxine: An Approachable Virtual Machine For, and In, Java

CHRISTIAN WIMMER, MICHAEL HAUPT, MICHAEL L. VAN DE VANTER,
MICK JORDAN, LAURENT DAYNÈS, and DOUGLAS SIMON, Oracle Labs

ACM Transactions on Architecture and Code Optimization, Vol. 9, No. 4, Article 30, Publication date: January 2013.

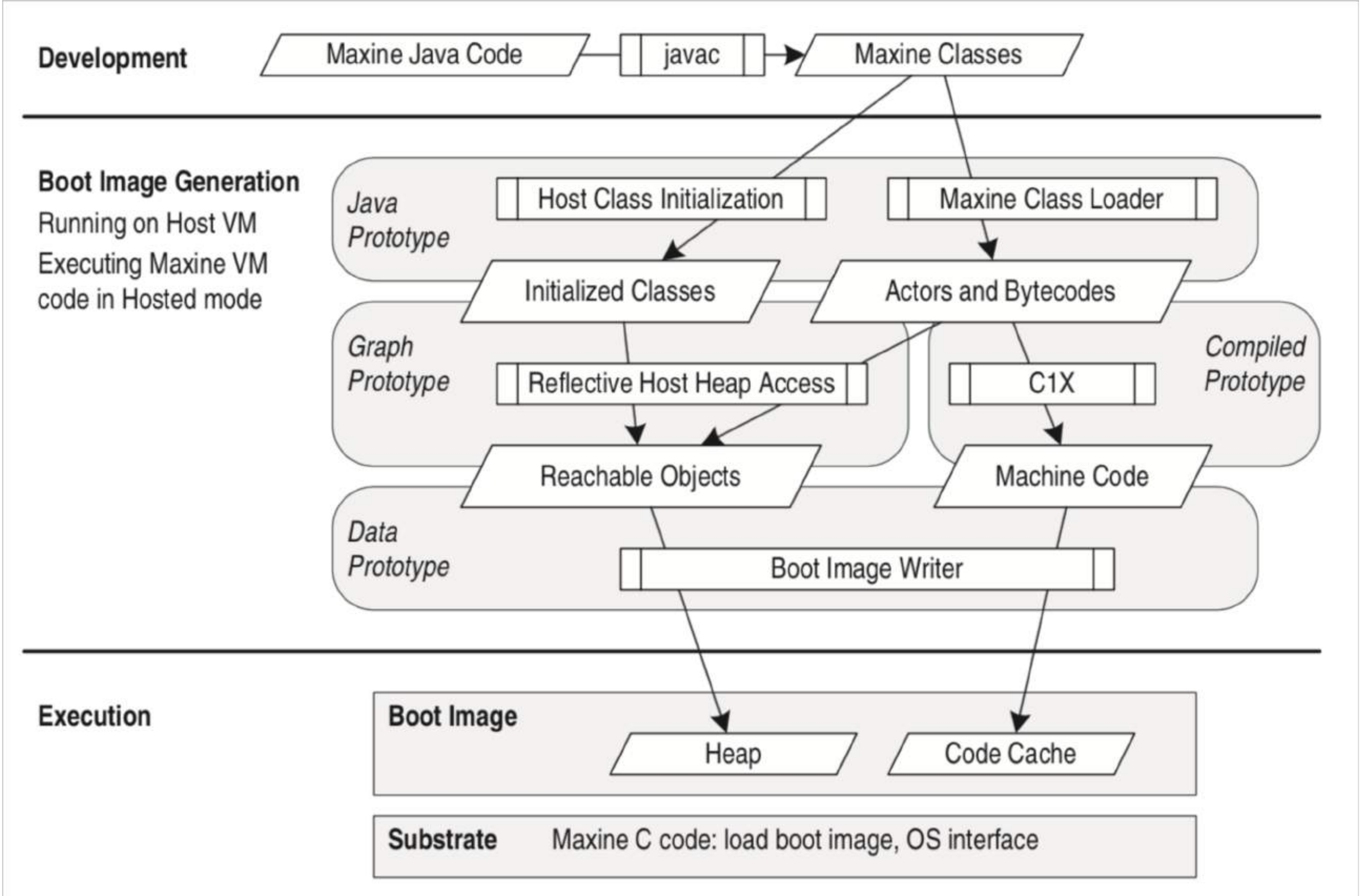


Fig. 4. Maxine VM boot image generation process.

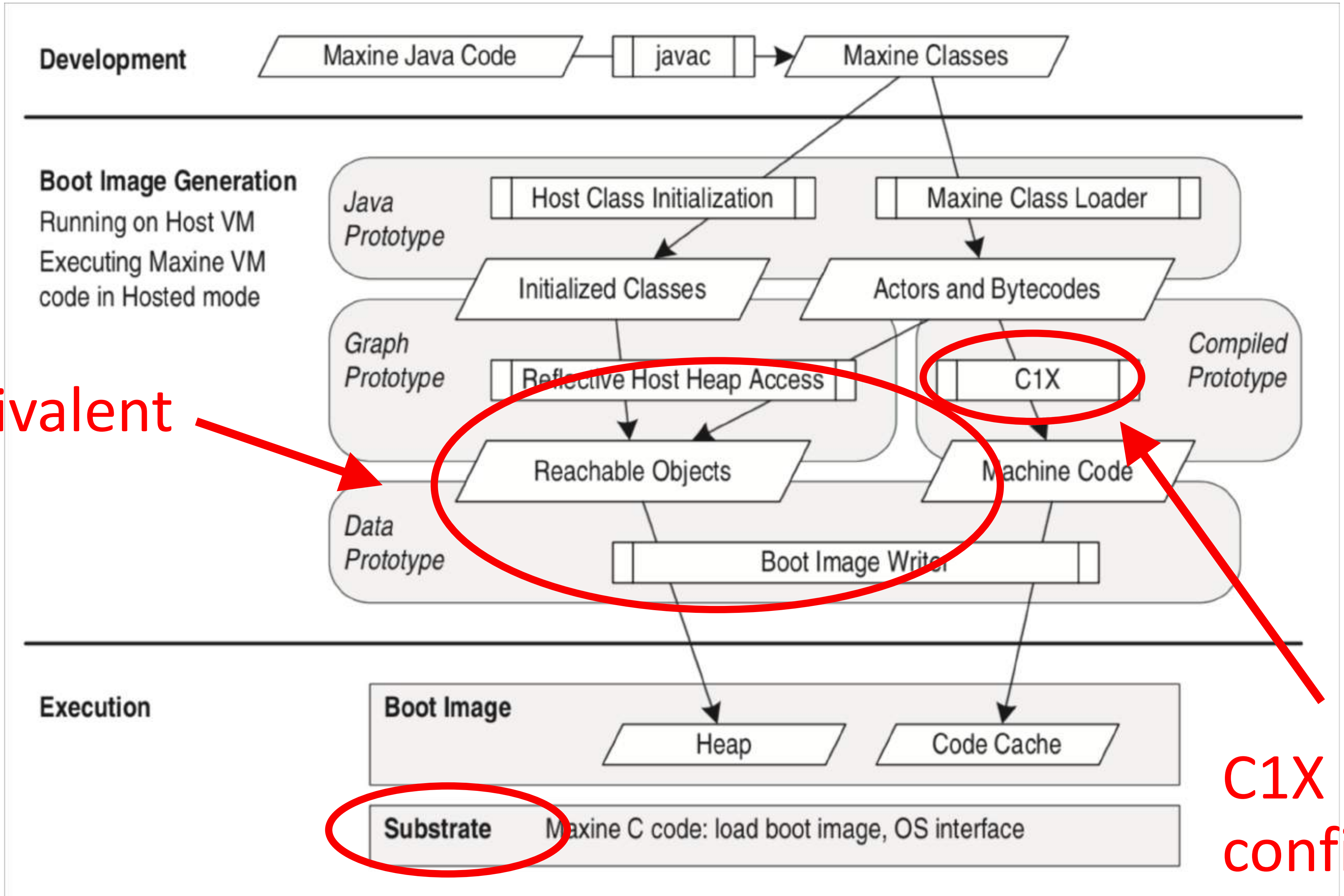


Fig. 4. Maxine VM boot image generation process.

SVM equivalent

C1X in an AOT configuration

SVM provides much of this, much like Maxine

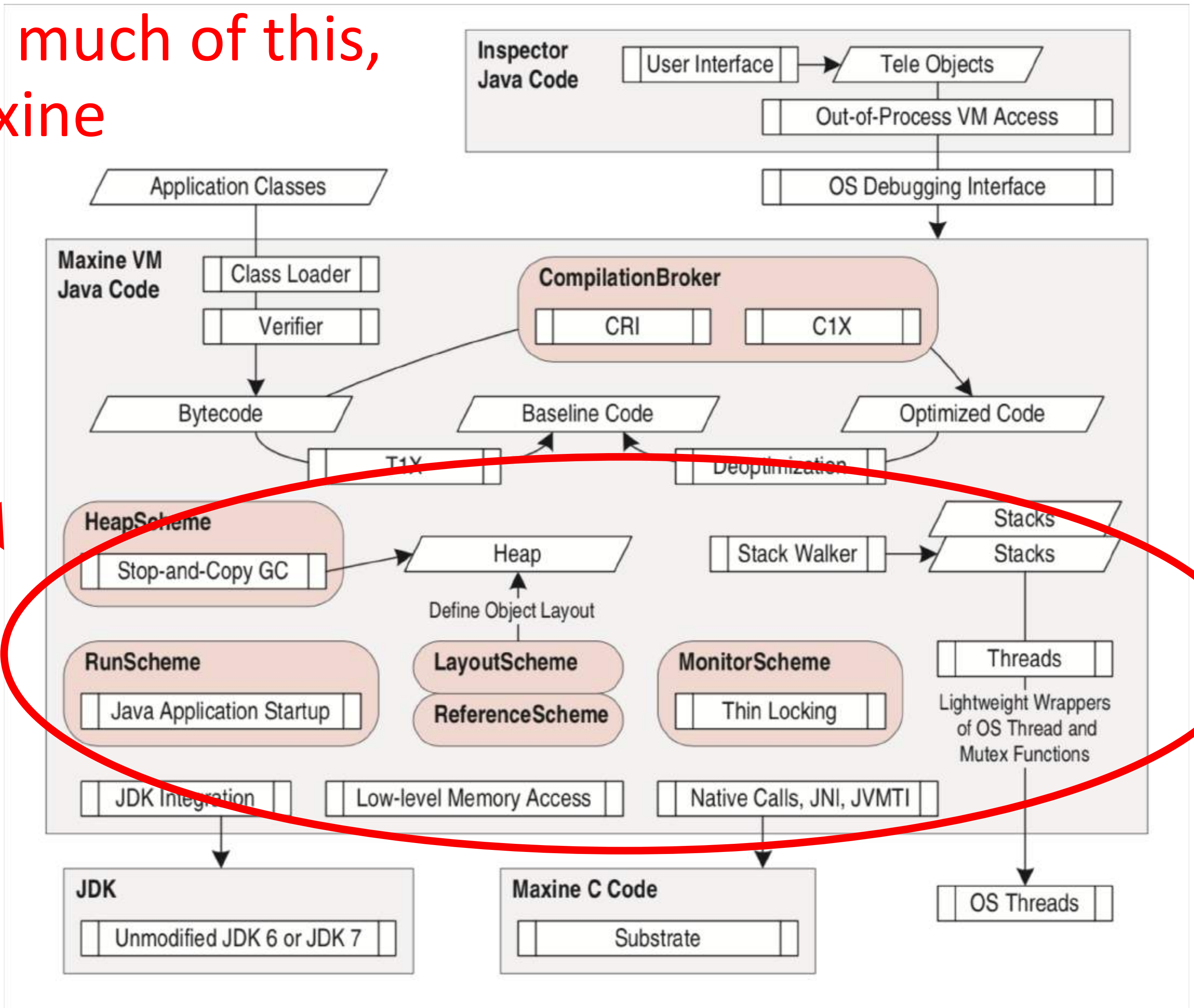


Fig. 1. Structure of the Maxine VM.

Doesn't provide this

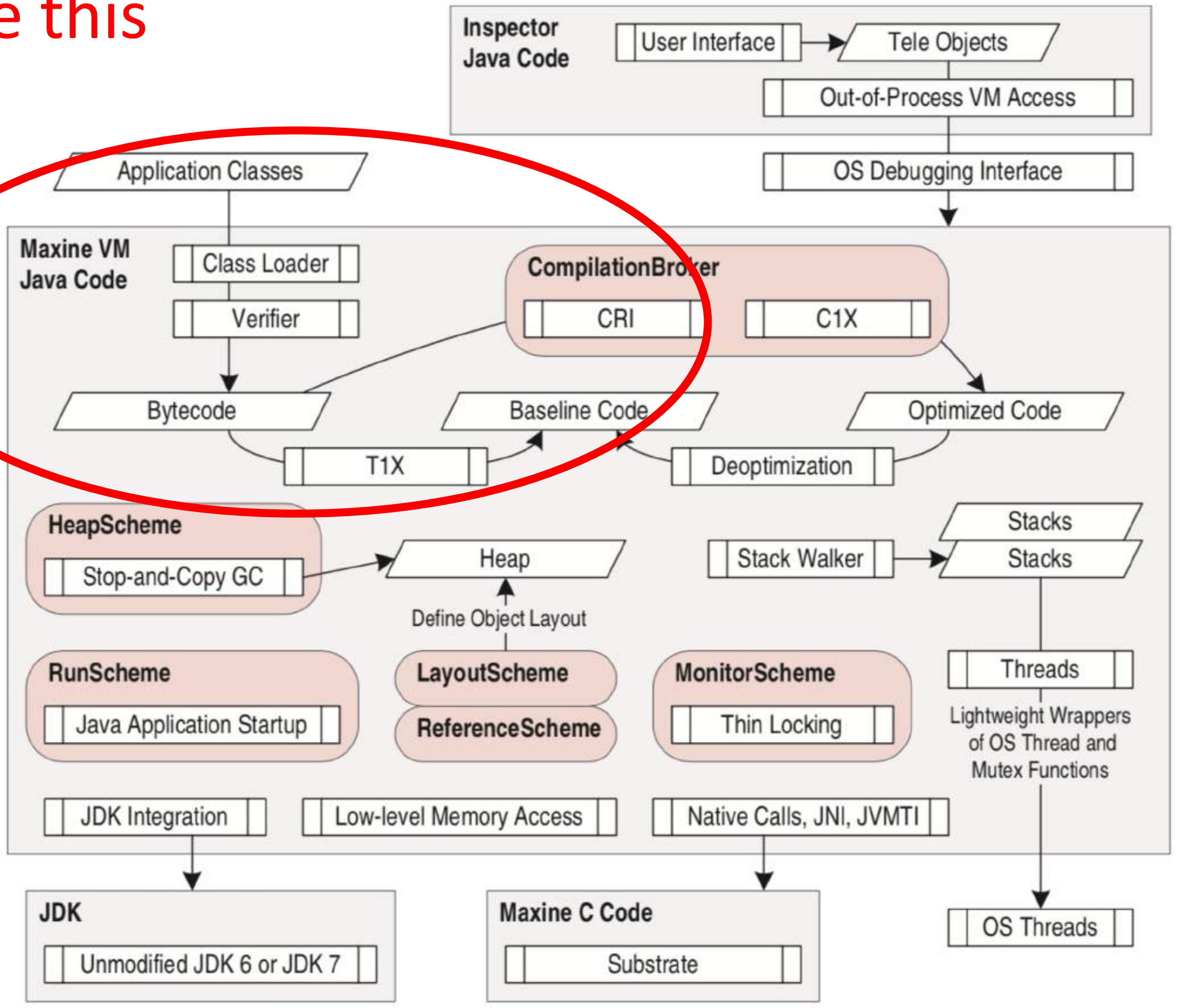
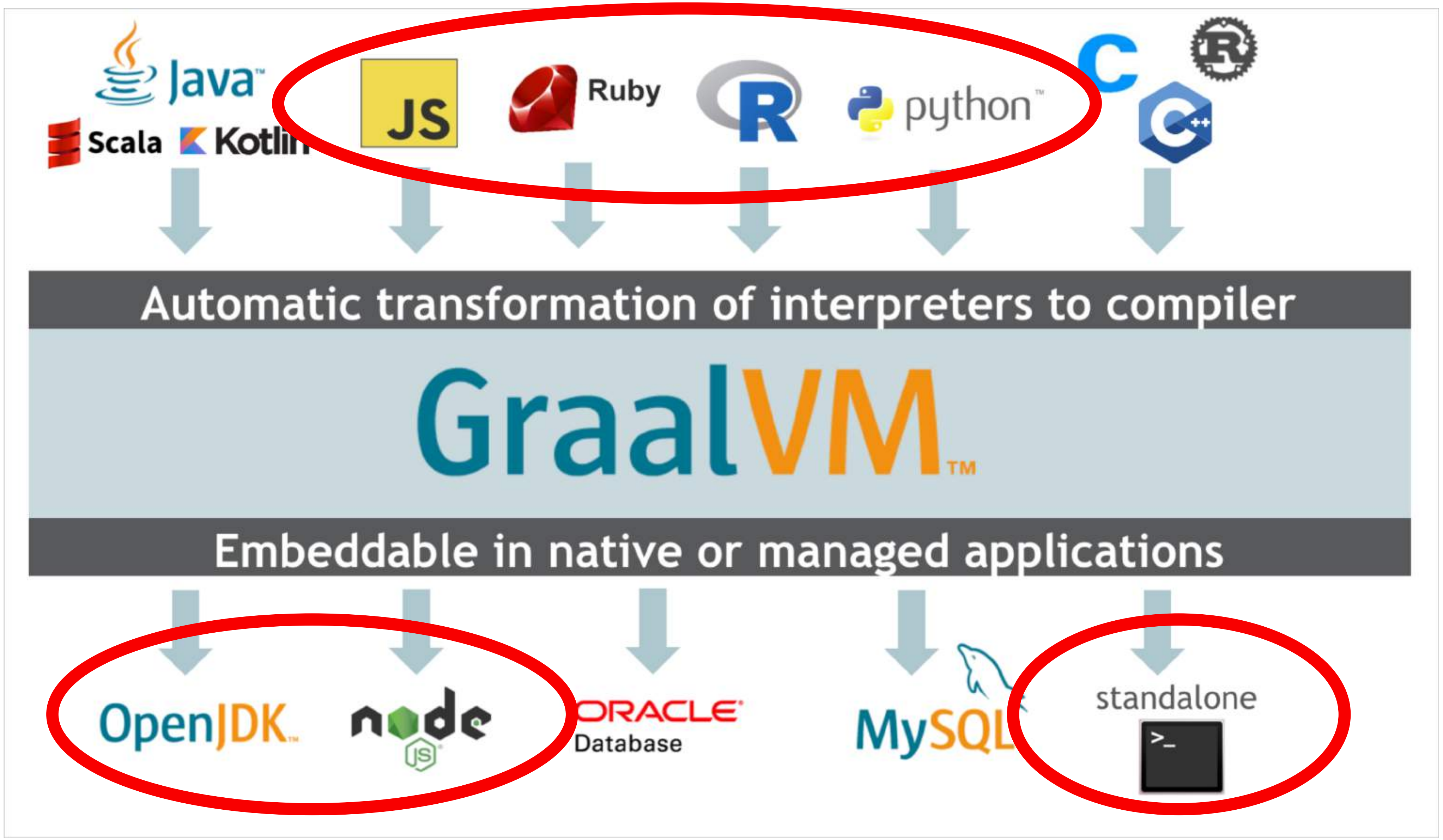
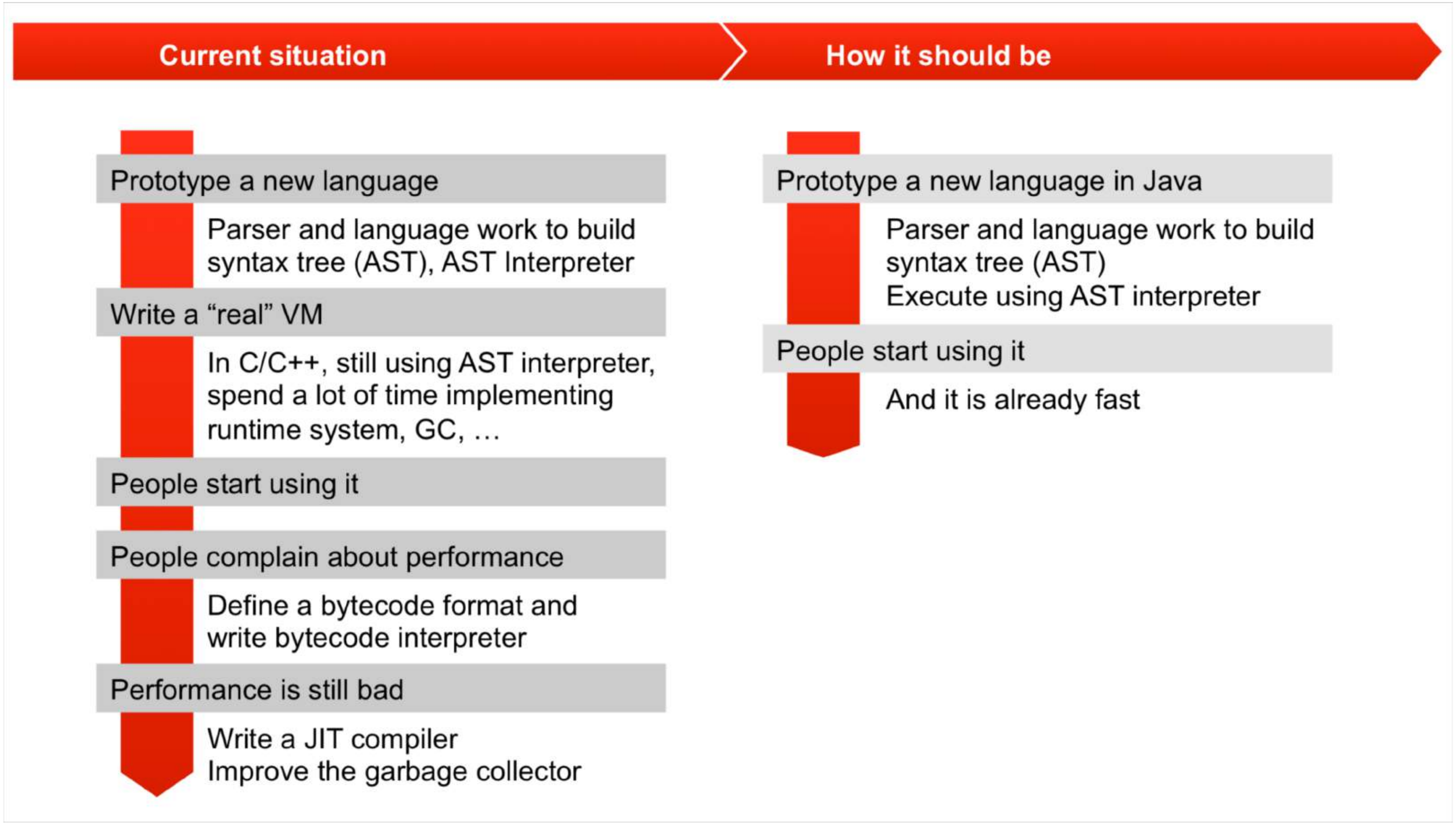


Fig. 1. Structure of the Maxine VM.

What are the applications of Graal?

Graal as a JIT compiler for dynamic languages


Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_



Current situation

Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a "real" VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …

People start using it

People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

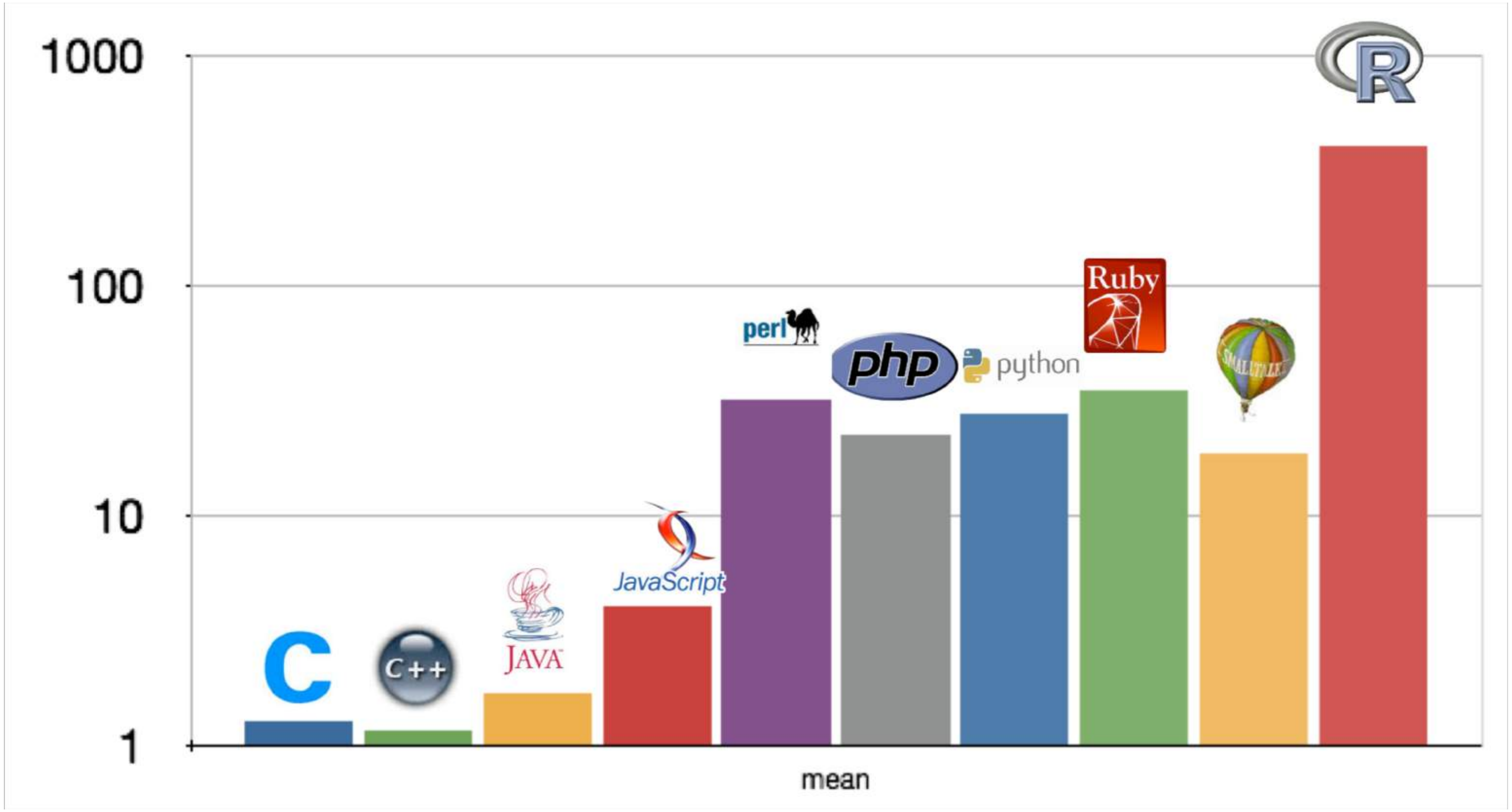
How it should be

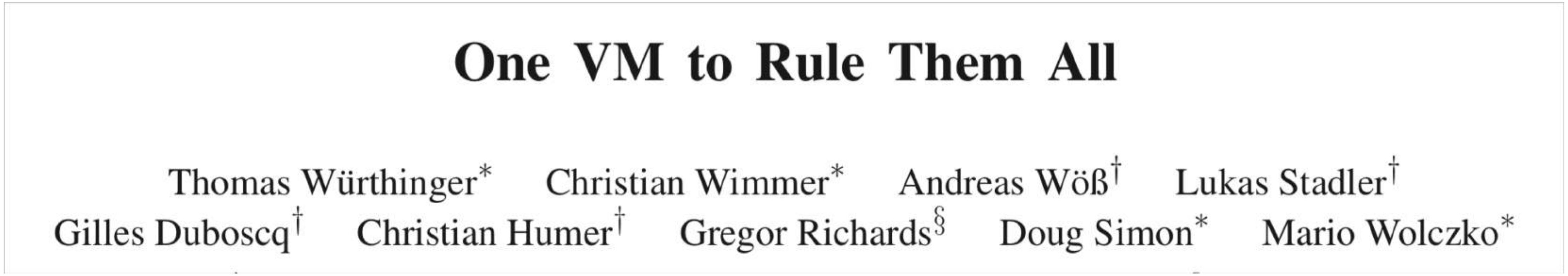
Prototype a new language in Java

Parser and language work to build syntax tree (AST)
Execute using AST interpreter

People start using it

And it is already fast

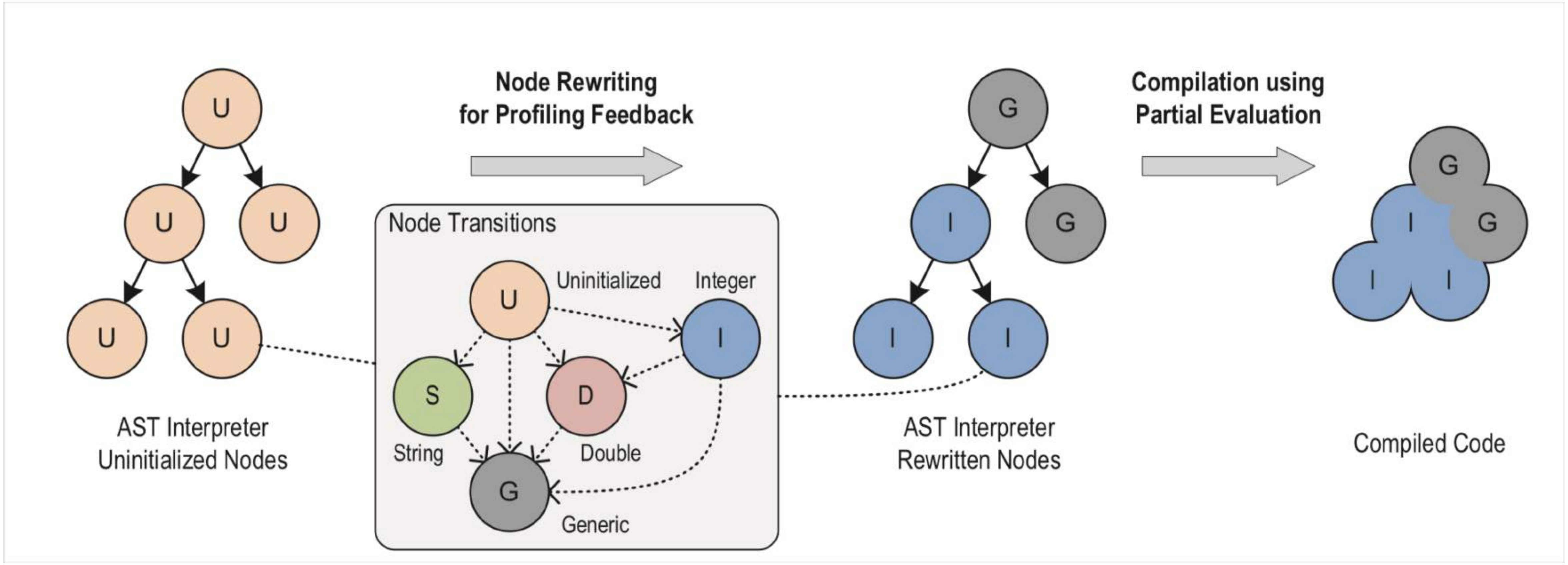

1000
100
10
1
mean
C
C++
JAVA
JavaScript
perl
php
python
Ruby
SMALLTALK
R

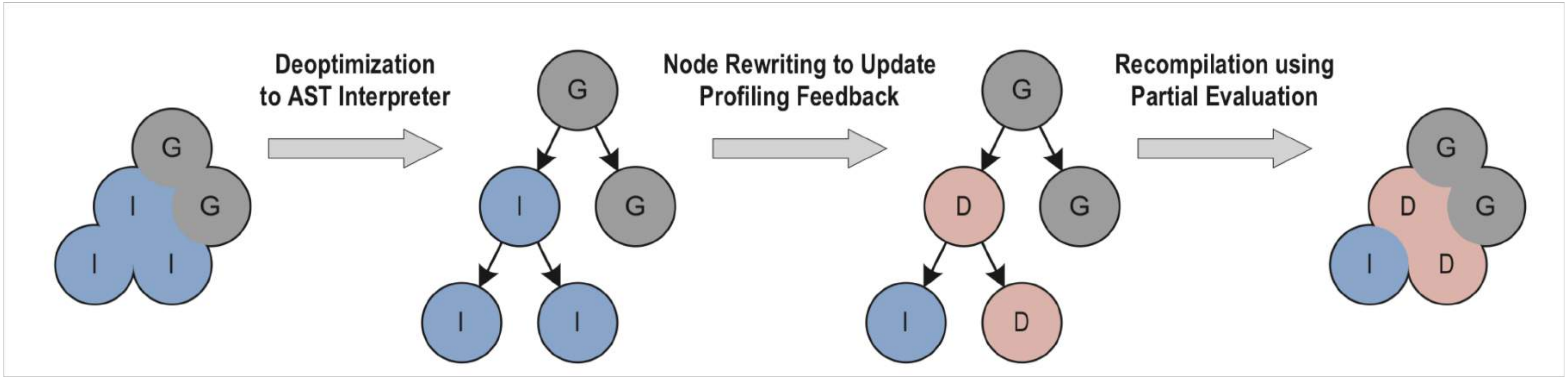


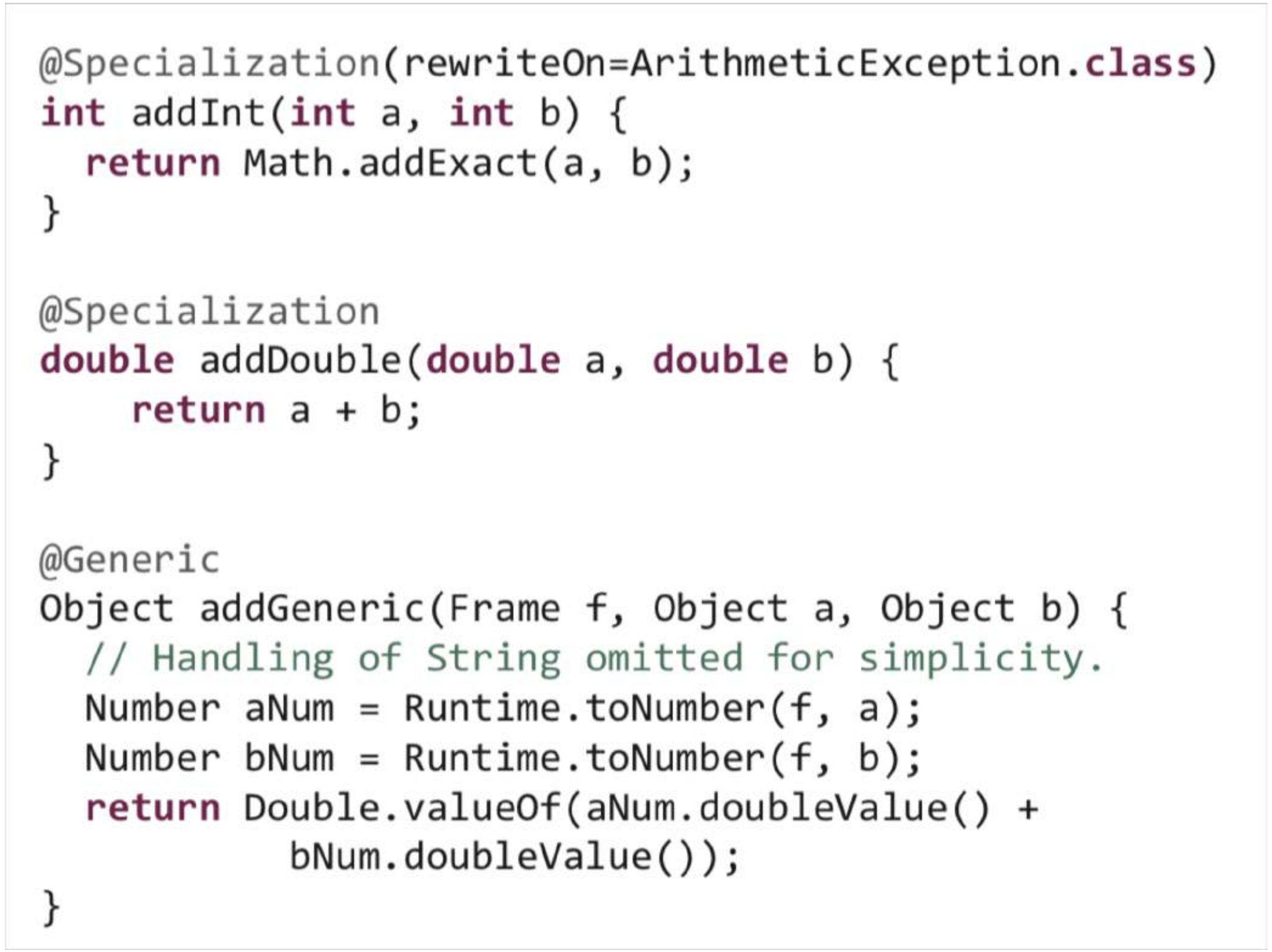
One VM to Rule Them All

Thomas Würthinger[*] Christian Wimmer[*] Andreas Wöß[†] Lukas Stadler[†]
Gilles Duboscq[†] Christian Humer[†] Gregor Richards[§] Doug Simon[*] Mario Wolczko[*]

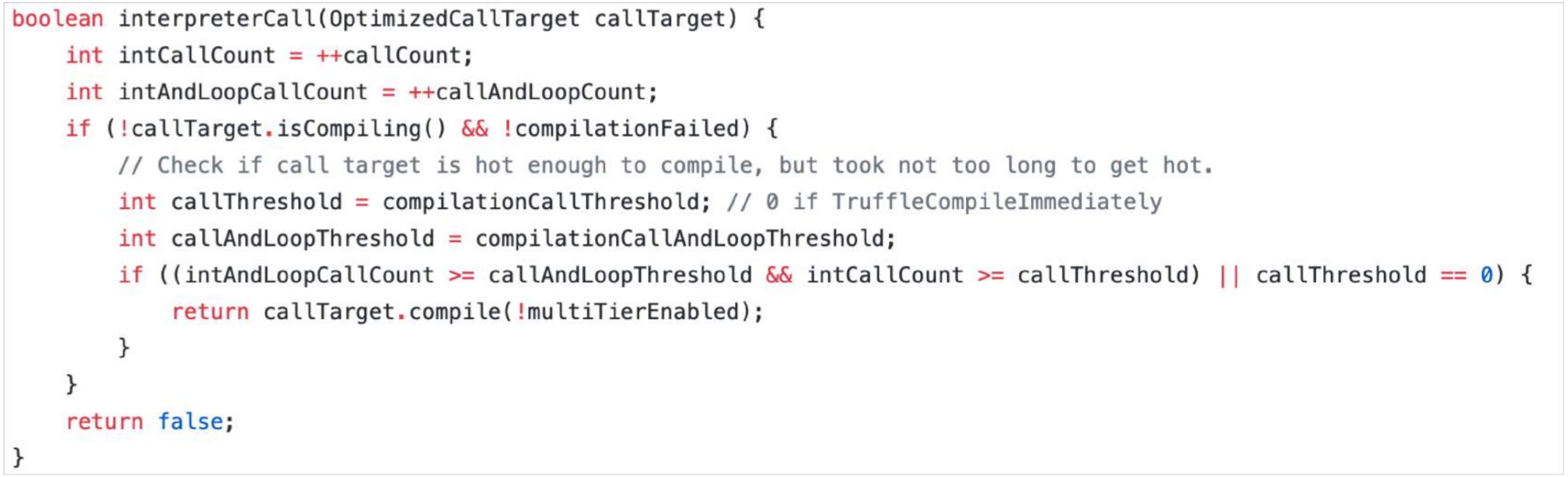
Onward! 2013, October 29–31, 2013, Indianapolis, Indiana, USA.


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


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



```java
@Specialization(rewriteOn=ArithmeticException.class)
int addInt(int a, int b) {
  return Math.addExact(a, b);
}

@Specialization
double addDouble(double a, double b) {
    return a + b;
}

@Generic
Object addGeneric(Frame f, Object a, Object b) {
  // Handling of String omitted for simplicity.
  Number aNum = Runtime.toNumber(f, a);
  Number bNum = Runtime.toNumber(f, b);
  return Double.valueOf(aNum.doubleValue() +
           bNum.doubleValue());
}
```



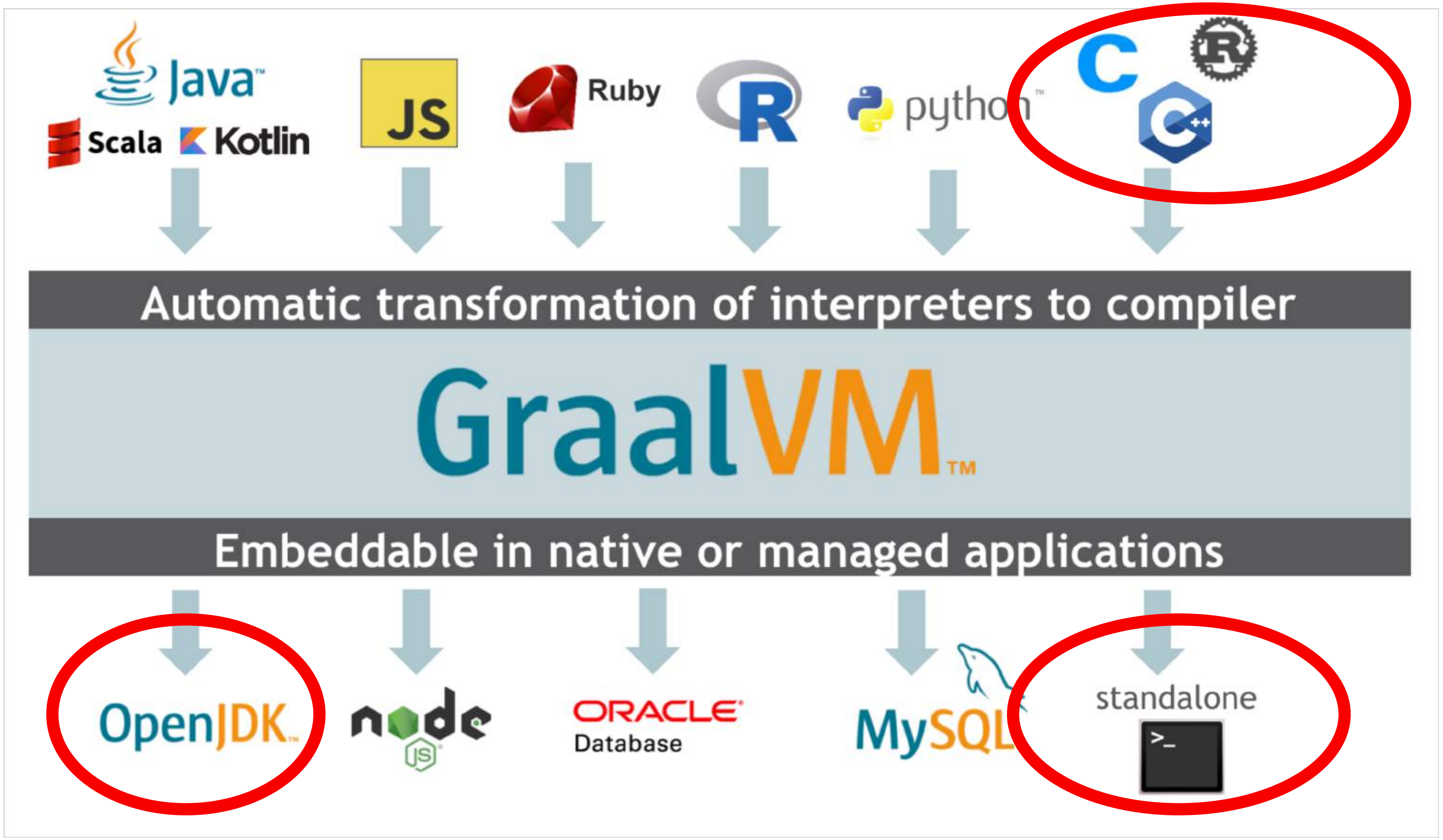
```java
boolean interpreterCall(OptimizedCallTarget callTarget) {
    int intCallCount = ++callCount;
    int intAndLoopCallCount = ++callAndLoopCount;
    if (!callTarget.isCompiling() && !compilationFailed) {
        // Check if call target is hot enough to compile, but took not too long to get hot.
        int callThreshold = compilationCallThreshold; // 0 if TruffleCompileImmediately
        int callAndLoopThreshold = compilationCallAndLoopThreshold;
        if ((intAndLoopCallCount >= callAndLoopThreshold && intCallCount >= callThreshold) || callThreshold == 0) {
            return callTarget.compile(!multiTierEnabled);
        }
    }
    return false;
}
```

```java
protected PEGraphDecoder createGraphDecoder(...) {
    final GraphBuilderConfiguration newConfig = configForParsing.copy();
    ...
    Plugins plugins = newConfig.getPlugins();
    ...
    plugins.appendInlineInvokePlugin(new ParsingInlineInvokePlugin(replacements, parsingInvocationPlugins, loopExplosionPlugin));
    ...
    return new CachingPEGraphDecoder(..., newConfig, ...);
}
```

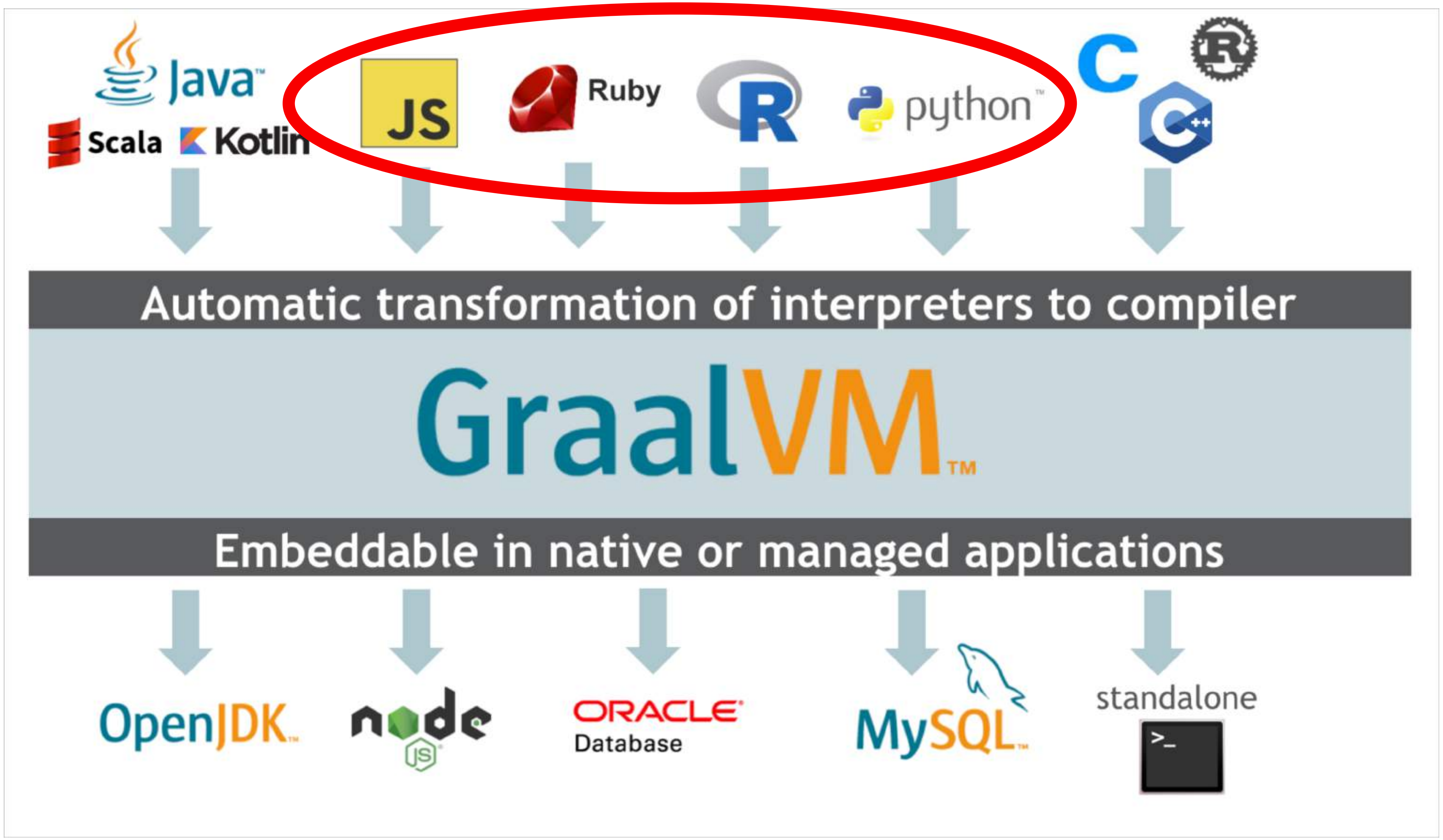
```java
public InlineKind getInlineKind(ResolvedJavaMethod original, boolean duringPartialEvaluation) {
    TruffleBoundary truffleBoundary = getAnnotation(TruffleBoundary.class, original);
    if (truffleBoundary != null) {
        if (duringPartialEvaluation || !truffleBoundary.allowInlining()) {
            // Since this method is invoked by the bytecode parser plugins, which can be invoked
            // by the partial evaluator, we want to prevent inlining across the boundary during
            // partial evaluation,
            // even if the TruffleBoundary allows inlining after partial evaluation.
            if (!truffleBoundary.throwsControlFlowException() && truffleBoundary.transferToInterpreterOnException()) {
                return InlineKind.DO_NOT_INLINE_DEOPTIMIZE_ON_EXCEPTION;
            } else {
                return InlineKind.DO_NOT_INLINE_WITH_EXCEPTION;
            }
        }
    } else if (getAnnotation(TruffleCallBoundary.class, original) != null) {
        return InlineKind.DO_NOT_INLINE_WITH_EXCEPTION;
    }
    return InlineKind.INLINE;
}
```

What are the applications of Graal?

Graal as a JIT compiler for native languages


Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_

ORACLE


Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone

```
VALUE psd_native_util_clamp(VALUE self,
    VALUE r_num, VALUE r_min, VALUE r_max) {
  int num = FIX2INT(r_num);
  int min = FIX2INT(r_min);
  int max = FIX2INT(r_max);

  return num > max ? r_max : (num < min ? r_min : r_num);
}
```

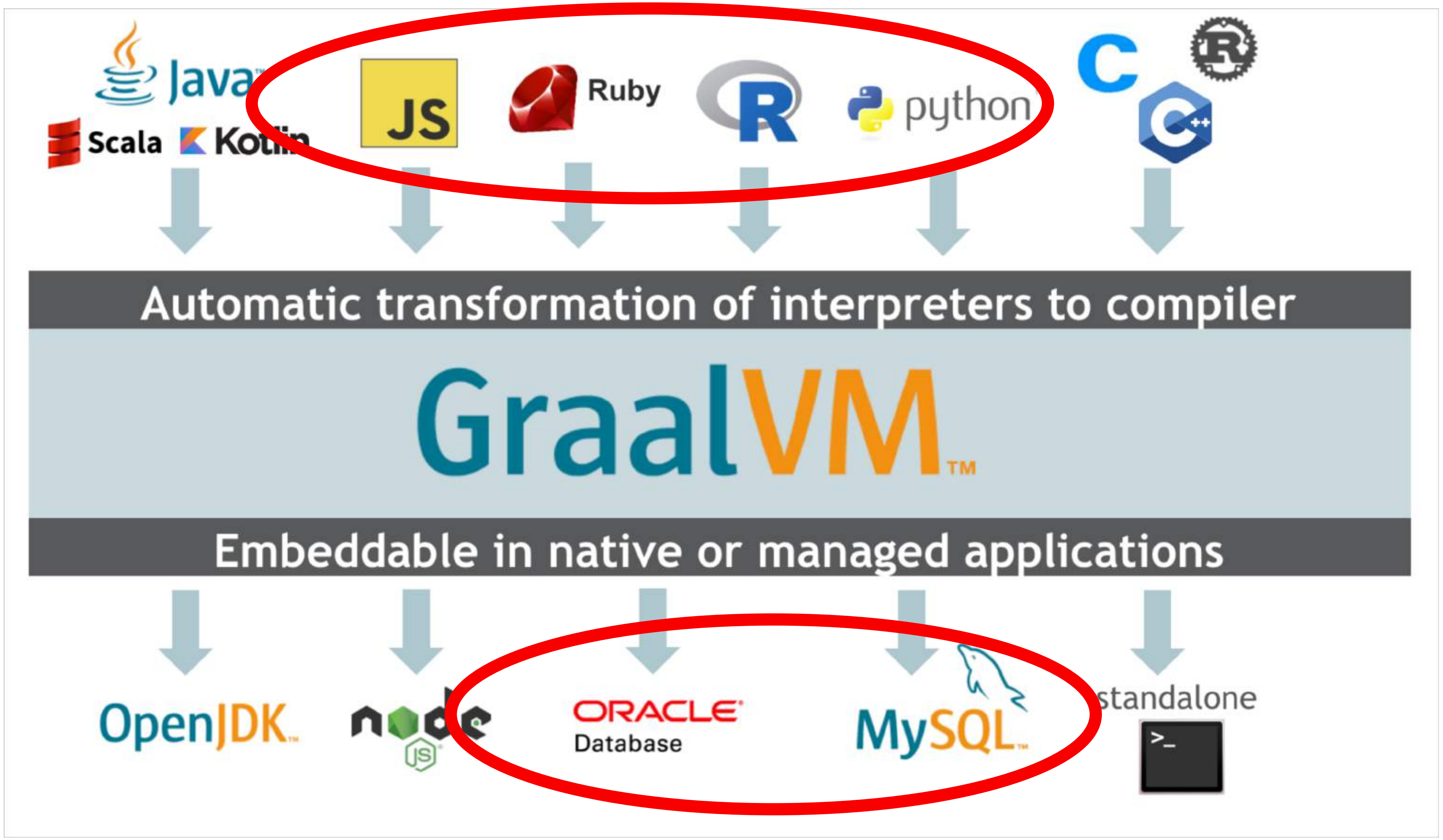
```
define i8* @psd_native_util_clamp(i8* %self,
    i8* %r_num, i8* %r_min, i8* %r_max) nounwind uwtable ssp {
  %1 = call i32 @FIX2INT(i8* %r_num)
  %2 = call i32 @FIX2INT(i8* %r_min)
  %3 = call i32 @FIX2INT(i8* %r_max)
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5                                       ; preds = %0
  br label %12
; <label>:6                                       ; preds = %0
  %7 = icmp slt i32 %1, %2
  br i1 %7, label %8, label %9
; <label>:8                                       ; preds = %6
  br label %10
; <label>:9                                       ; preds = %6
  br label %10
; <label>:10                                      ; preds = %9, %8
  %11 = phi i8* [ %r_min, %8 ], [ %r_num, %9 ]
  br label %12
; <label>:12                                      ; preds = %10, %5
  %13 = phi i8* [ %r_max, %5 ], [ %11, %10 ]
  ret i8* %13
}
```

```
  %4 = icmp sgt i32 %1, %3
  br i1 %4, label %5, label %6
; <label>:5
  br label %12
; <label>:6
  %7 = icmp slt i32 %1, %2
  br i1 %7, label %8, label %9
```

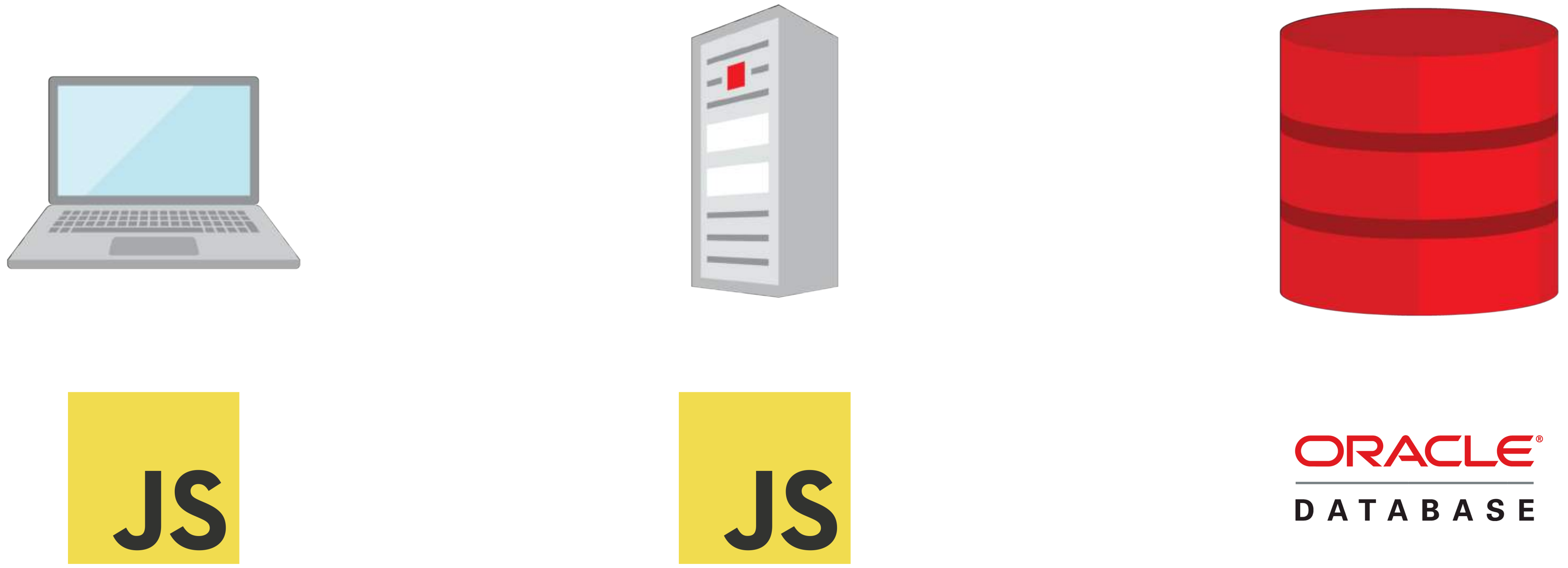
```
    t4 = t1 > t3
    if t4
      goto l5
    else
      goto l6
    end
l5: goto l12
l6: t7 = t1 < t2
    if t7
      goto l8
    else
      goto l9
    end
```

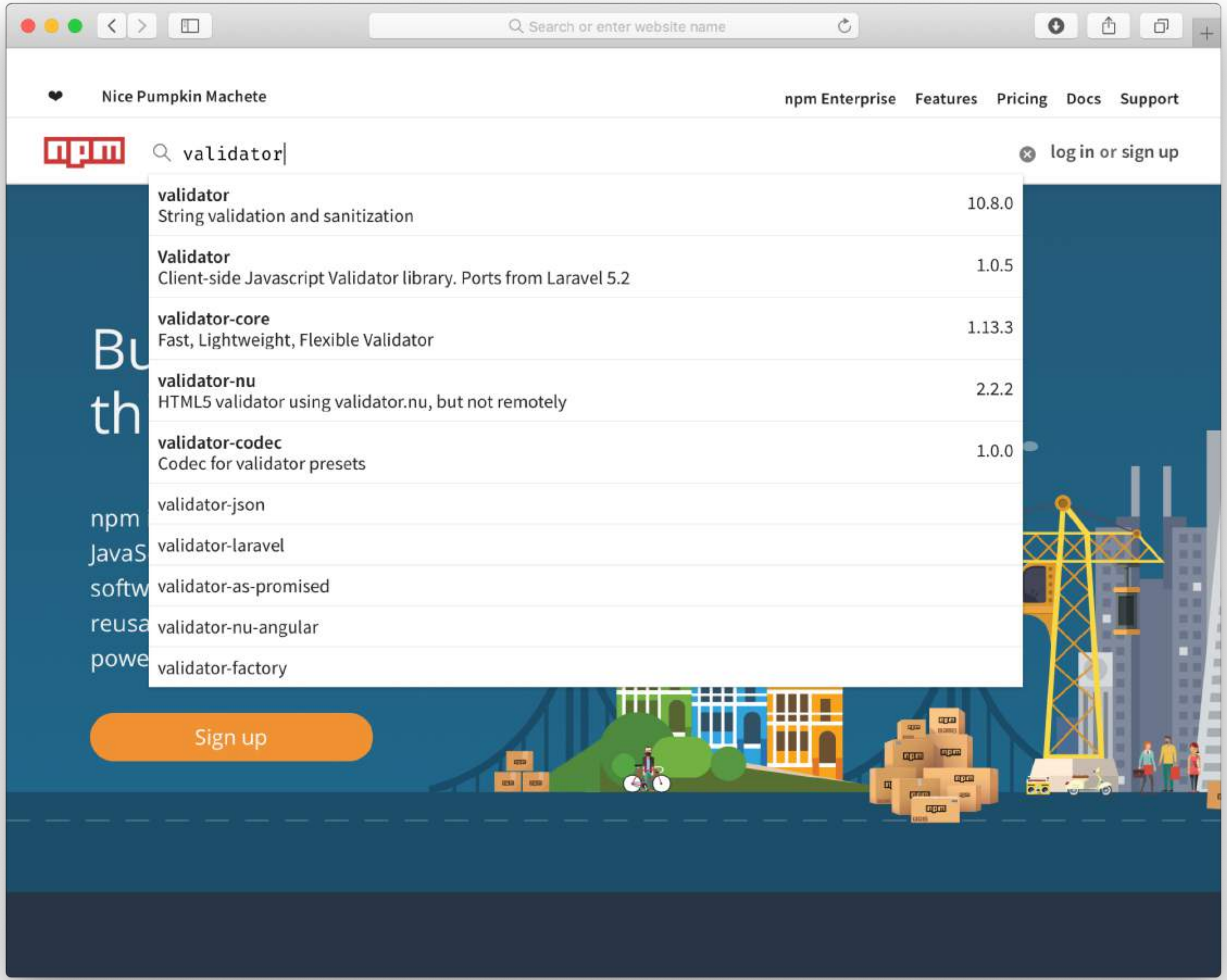
What are the applications of Graal?

Graal as a tool for embedding languages


Java
Scala
Kotlin
JS
Ruby
R
python
C
C++
Automatic transformation of interpreters to compiler
GraalVM™
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE Database
MySQL
standalone
>_

ORACLE


JS
JS
ORACLE
DATABASE



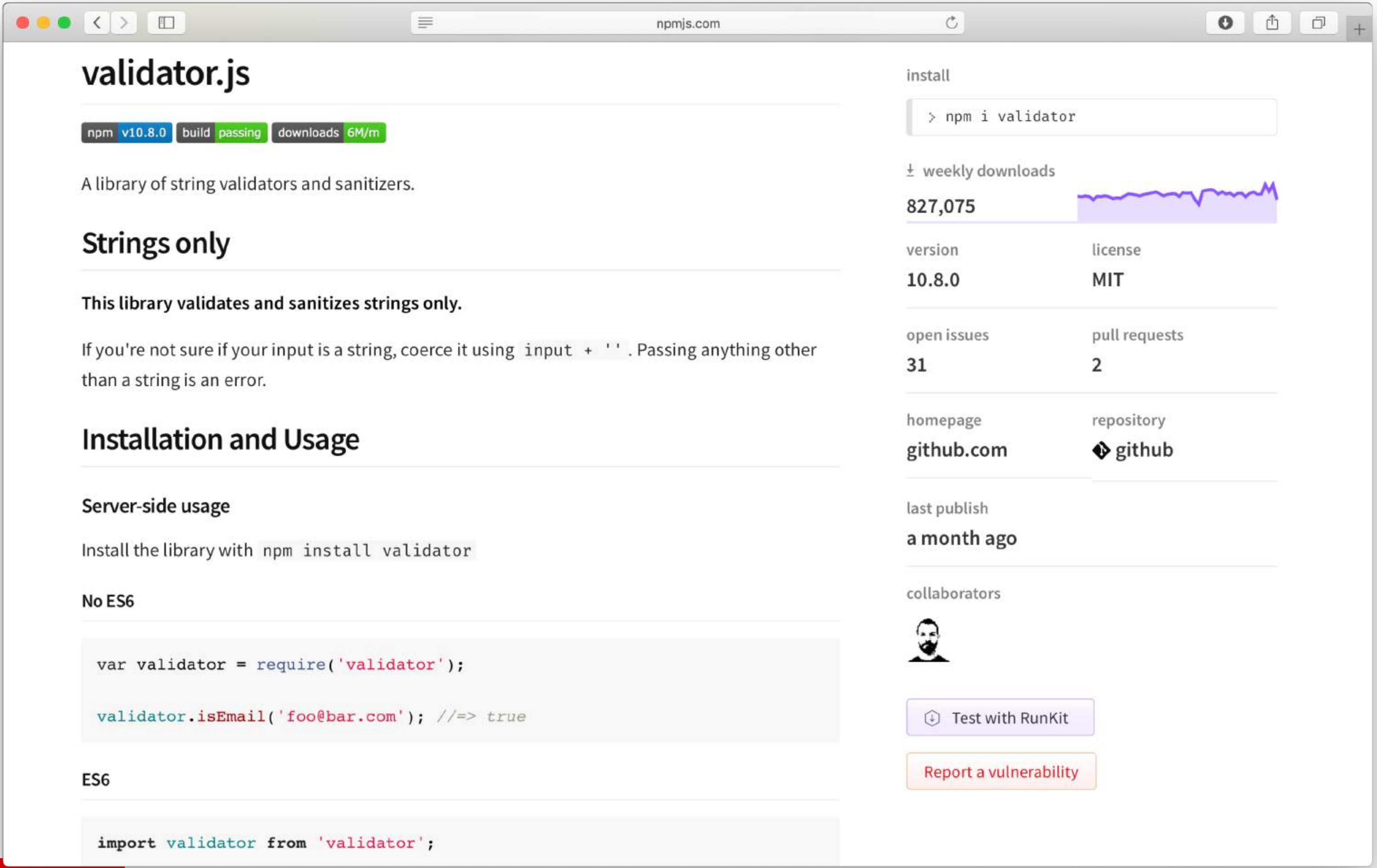
Nice Pumpkin Machete

npm Enterprise Features Pricing Docs Support

validator

log in or sign up

validator String validation and sanitization	10.8.0
Validator Client-side Javascript Validator library. Ports from Laravel 5.2	1.0.5
validator-core Fast, Lightweight, Flexible Validator	1.13.3
validator-nu HTML5 validator using validator.nu, but not remotely	2.2.2
validator-codec Codec for validator presets	1.0.0
validator-json	
validator-laravel	
validator-as-promised	
validator-nu-angular	
validator-factory	



npmjs.com

validator.js

npm v10.8.0 build passing downloads 6M/m

A library of string validators and sanitizers.

Strings only

This library validates and sanitizes strings only.

If you're not sure if your input is a string, coerce it using `input + ''`. Passing anything other than a string is an error.

Installation and Usage

Server-side usage

Install the library with `npm install validator`

No ES6

```
var validator = require('validator');

validator.isEmail('foo@bar.com'); //=> true
```

ES6

```
import validator from 'validator';
```

install

```
> npm i validator
```

weekly downloads
827,075

version	license
10.8.0	MIT
open issues	pull requests
31	2
homepage	repository
github.com	github

last publish
a month ago

collaborators

Demo using the Oracle Database MLE

- Multi-lingual (polyglot) edition
- Available as a Docker image
- Subject to the Oracle Technology Network license agreement, so you need to accept that and download it yourself

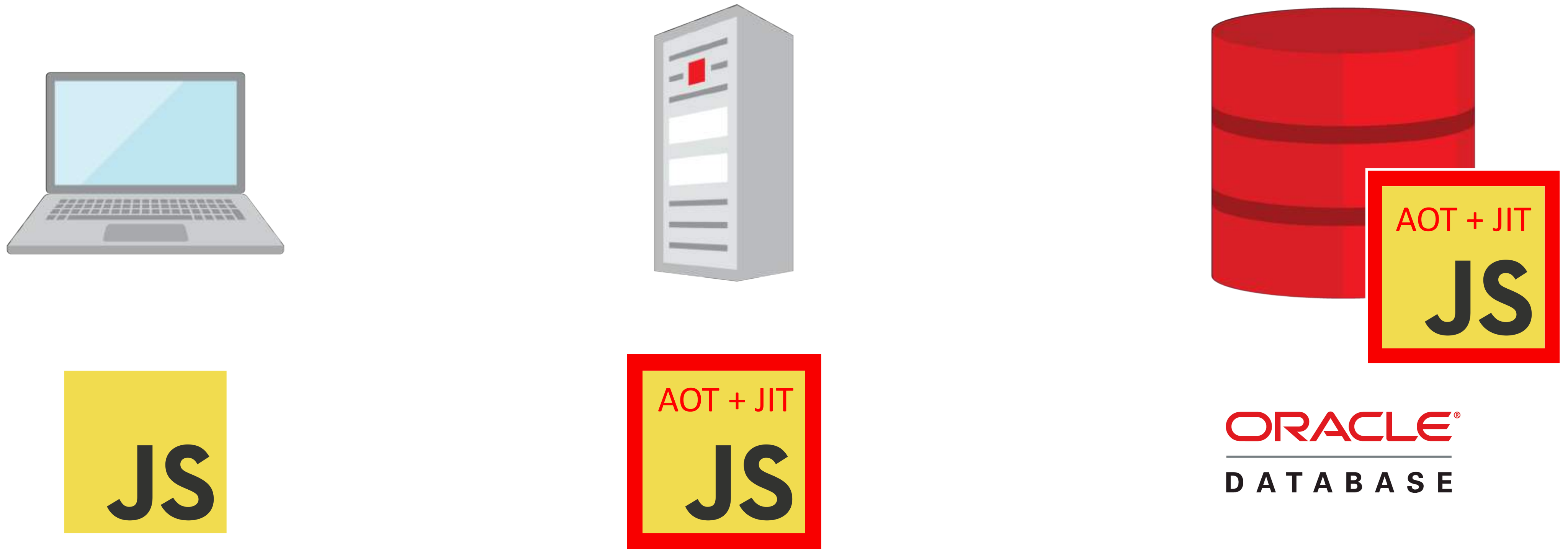
https://oracle.github.io/oracle-db-mle/releases/0.2.7/docker/

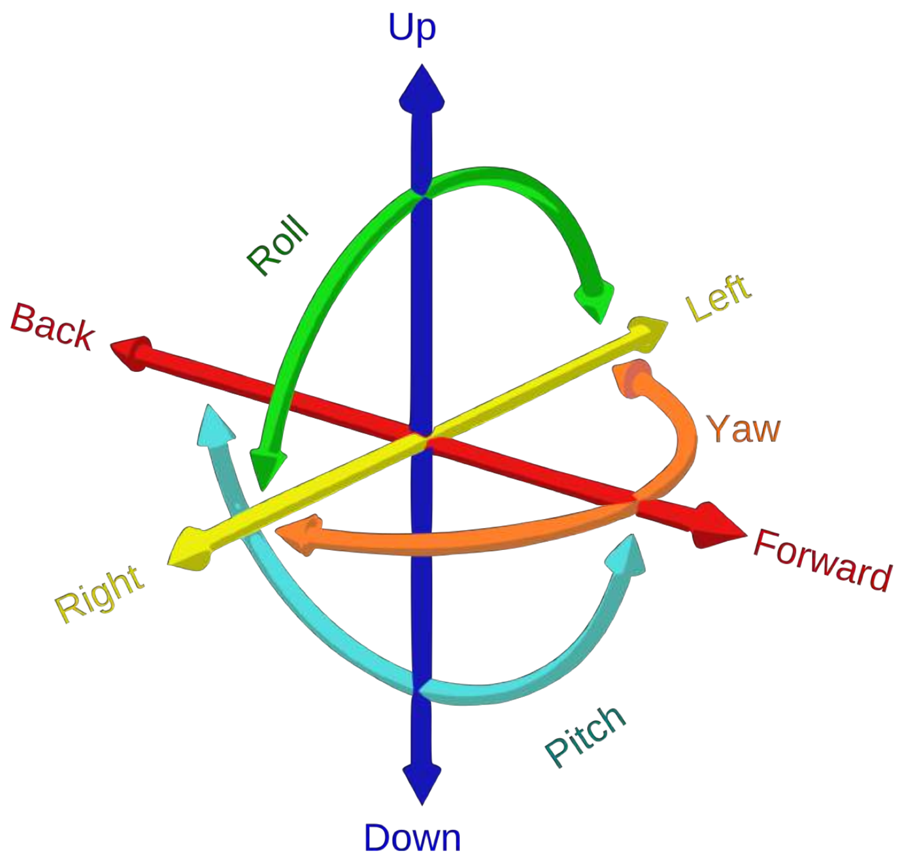
```
$ npm install validator
$ npm install @types/validator

$ dbjs deploy -u … -p … -c localhost:1521/ORCLCDB validator

$ sqlplus …/…@localhost:1521/ORCLCDB
```

(Demo simplified - see the website to get specifics)

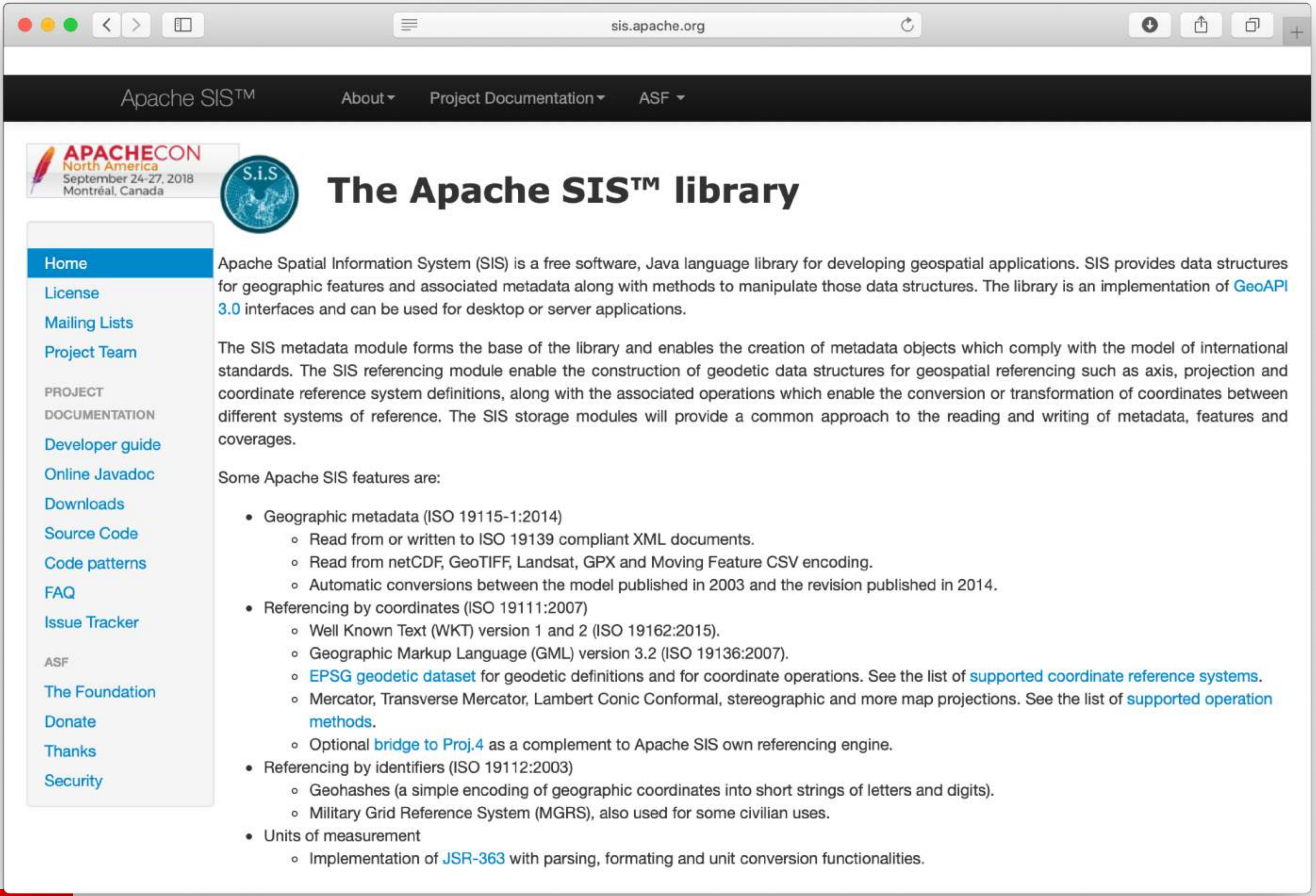
```
SQL> select validator.isEmail('chris.seaton@oracle.com');
0

SQL> select validator.isEmail('chris.seaton');
1
```

(Demo simplified - see the website to get specifics)


JS
AOT + JIT
JS
AOT + JIT
JS
ORACLE
DATABASE



https://commons.wikimedia.org/wiki/File:6DOF.svg
Creative Commons Attribution-Share Alike 4.0 International


sis.apache.org
Apache SIS™
About
Project Documentation
ASF
APACHECON North America September 24-27, 2018 Montréal, Canada
S.i.S

The Apache SIS™ library

Apache Spatial Information System (SIS) is a free software, Java language library for developing geospatial applications. SIS provides data structures for geographic features and associated metadata along with methods to manipulate those data structures. The library is an implementation of GeoAPI 3.0 interfaces and can be used for desktop or server applications.

The SIS metadata module forms the base of the library and enables the creation of metadata objects which comply with the model of international standards. The SIS referencing module enable the construction of geodetic data structures for geospatial referencing such as axis, projection and coordinate reference system definitions, along with the associated operations which enable the conversion or transformation of coordinates between different systems of reference. The SIS storage modules will provide a common approach to the reading and writing of metadata, features and coverages.

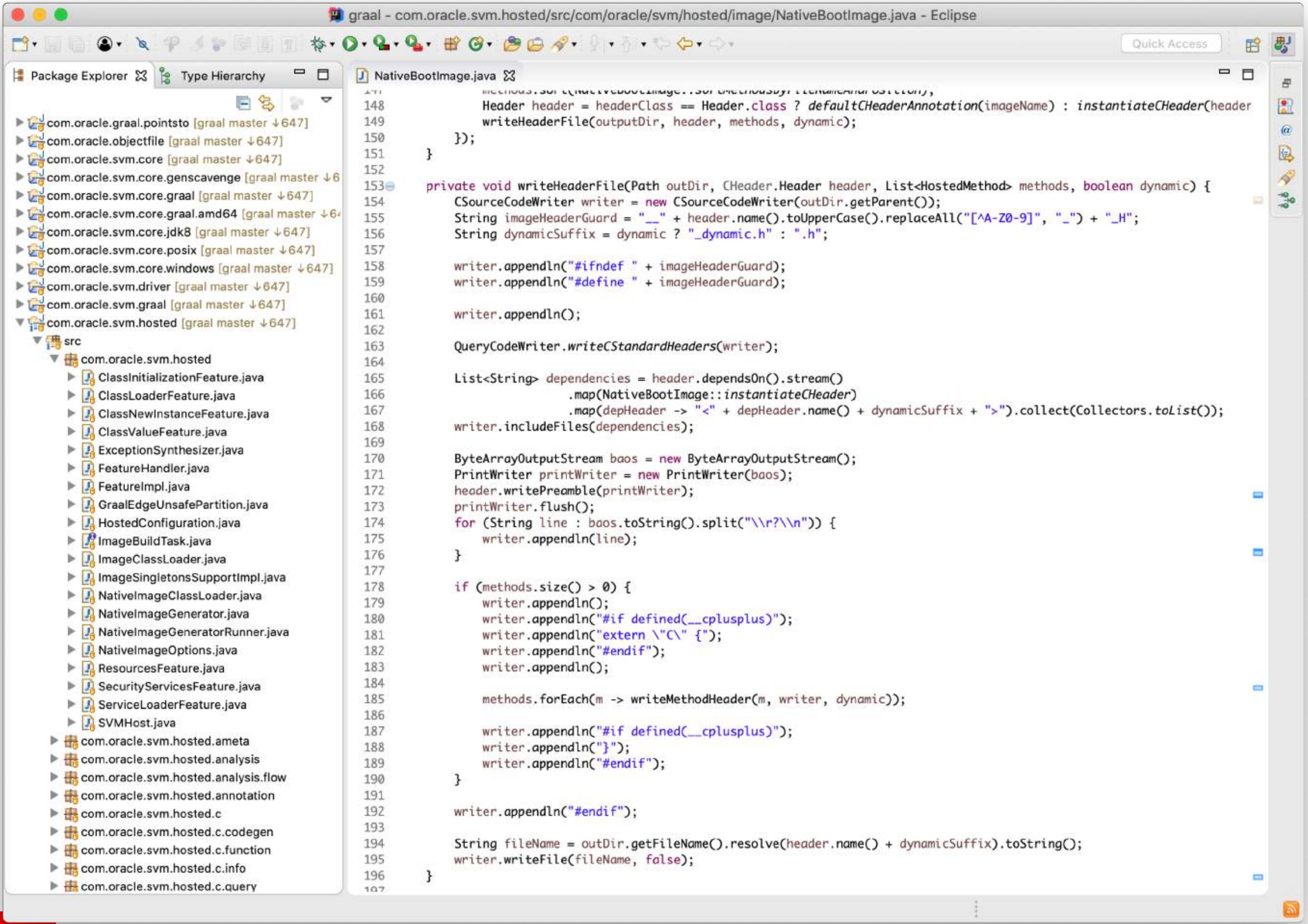
Some Apache SIS features are:

- Geographic metadata (ISO 19115-1:2014)
  - Read from or written to ISO 19139 compliant XML documents.
  - Read from netCDF, GeoTIFF, Landsat, GPX and Moving Feature CSV encoding.
  - Automatic conversions between the model published in 2003 and the revision published in 2014.
- Referencing by coordinates (ISO 19111:2007)
  - Well Known Text (WKT) version 1 and 2 (ISO 19162:2015).
  - Geographic Markup Language (GML) version 3.2 (ISO 19136:2007).
  - EPSG geodetic dataset for geodetic definitions and for coordinate operations. See the list of supported coordinate reference systems.
  - Mercator, Transverse Mercator, Lambert Conic Conformal, stereographic and more map projections. See the list of supported operation methods.
  - Optional bridge to Proj.4 as a complement to Apache SIS own referencing engine.
- Referencing by identifiers (ISO 19112:2003)
  - Geohashes (a simple encoding of geographic coordinates into short strings of letters and digits).
  - Military Grid Reference System (MGRS), also used for some civilian uses.
- Units of measurement
  - Implementation of JSR-363 with parsing, formating and unit conversion functionalities.

```java
...
import org.graalvm.nativeimage.IsolateThread;
import org.graalvm.nativeimage.c.function.CEntryPoint;

public class Distance {

    ...

    @CEntryPoint(name = "distance")
    public static double distance(IsolateThread thread,
          double a_lat, double a_long,
          double b_lat, double b_long) {
        return DistanceUtils.getHaversineDistance(a_lat, a_long, b_lat, b_long);
    }

    ...

}
```

```
$ native-image -cp sis.jar:. -H:Kind=SHARED_LIBRARY \
       -H:Name=libdistance
```

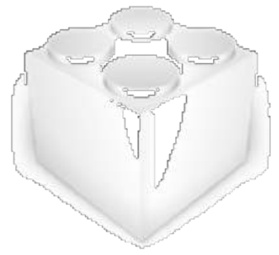
```c
#include <stdlib.h>
#include <stdio.h>

#include <libdistance.h>

int main(int argc, char **argv) {
  graal_isolate_t *isolate = NULL;
  graal_isolatethread_t *thread = NULL;

  if (graal_create_isolate(NULL, &isolate) != 0 || (thread = graal_current_thread(isolate)) == NULL) {
    fprintf(stderr, "initialization error\n");
    return 1;
  }

  double a_lat   = strtod(argv[1], NULL);
  double a_long  = strtod(argv[2], NULL);
  double b_lat   = strtod(argv[3], NULL);
  double b_long  = strtod(argv[4], NULL);

  printf("%f km\n", distance(thread, a_lat, a_long, b_lat, b_long));

  return 0;
}
```


graal - com.oracle.svm.hosted/src/com/oracle/svm/hosted/image/NativeBootImage.java - Eclipse
Quick Access
Package Explorer
Type Hierarchy
NativeBootImage.java

```
com.oracle.graal.pointsto [graal master ↓647]
com.oracle.objectfile [graal master ↓647]
com.oracle.svm.core [graal master ↓647]
com.oracle.svm.core.genscavenge [graal master ↓6
com.oracle.svm.core.graal [graal master ↓647]
com.oracle.svm.core.graal.amd64 [graal master ↓6
com.oracle.svm.core.jdk8 [graal master ↓647]
com.oracle.svm.core.posix [graal master ↓647]
com.oracle.svm.core.windows [graal master ↓647]
com.oracle.svm.driver [graal master ↓647]
com.oracle.svm.graal [graal master ↓647]
com.oracle.svm.hosted [graal master ↓647]
  src
    com.oracle.svm.hosted
      ClassInitializationFeature.java
      ClassLoaderFeature.java
      ClassNewInstanceFeature.java
      ClassValueFeature.java
      ExceptionSynthesizer.java
      FeatureHandler.java
      FeatureImpl.java
      GraalEdgeUnsafePartition.java
      HostedConfiguration.java
      ImageBuildTask.java
      ImageClassLoader.java
      ImageSingletonsSupportImpl.java
      NativeImageClassLoader.java
      NativeImageGenerator.java
      NativeImageGeneratorRunner.java
      NativeImageOptions.java
      ResourcesFeature.java
      SecurityServicesFeature.java
      ServiceLoaderFeature.java
      SVMHost.java
    com.oracle.svm.hosted.ameta
    com.oracle.svm.hosted.analysis
    com.oracle.svm.hosted.analysis.flow
    com.oracle.svm.hosted.annotation
    com.oracle.svm.hosted.c
    com.oracle.svm.hosted.c.codegen
    com.oracle.svm.hosted.c.function
    com.oracle.svm.hosted.c.info
    com.oracle.svm.hosted.c.query
```

```java
            Header header = headerClass == Header.class ? defaultCHeaderAnnotation(imageName) : instantiateCHeader(header
            writeHeaderFile(outputDir, header, methods, dynamic);
        });
    }

    private void writeHeaderFile(Path outDir, CHeader.Header header, List<HostedMethod> methods, boolean dynamic) {
        CSourceCodeWriter writer = new CSourceCodeWriter(outDir.getParent());
        String imageHeaderGuard = "__" + header.name().toUpperCase().replaceAll("[^A-Z0-9]", "_") + "_H";
        String dynamicSuffix = dynamic ? "_dynamic.h" : ".h";

        writer.appendln("#ifndef " + imageHeaderGuard);
        writer.appendln("#define " + imageHeaderGuard);

        writer.appendln();

        QueryCodeWriter.writeCStandardHeaders(writer);

        List<String> dependencies = header.dependsOn().stream()
                        .map(NativeBootImage::instantiateCHeader)
                        .map(depHeader -> "<" + depHeader.name() + dynamicSuffix + ">").collect(Collectors.toList());
        writer.includeFiles(dependencies);

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        PrintWriter printWriter = new PrintWriter(baos);
        header.writePreamble(printWriter);
        printWriter.flush();
        for (String line : baos.toString().split("\\r?\\n")) {
            writer.appendln(line);
        }

        if (methods.size() > 0) {
            writer.appendln();
            writer.appendln("#if defined(__cplusplus)");
            writer.appendln("extern \"C\" {");
            writer.appendln("#endif");
            writer.appendln();

            methods.forEach(m -> writeMethodHeader(m, writer, dynamic));

            writer.appendln("#if defined(__cplusplus)");
            writer.appendln("}");
            writer.appendln("#endif");
        }

        writer.appendln("#endif");

        String fileName = outDir.getFileName().resolve(header.name() + dynamicSuffix).toString();
        writer.writeFile(fileName, false);
    }
```

```
$ clang -I. -L. -ldistance distance.c -o distance
$ otool -L distance
distance:
     libdistance.dylib
     /usr/lib/libSystem.B.dylib
$ ./distance 51.507222 -0.1275 40.7127 -74.0059
5570.25 km
```



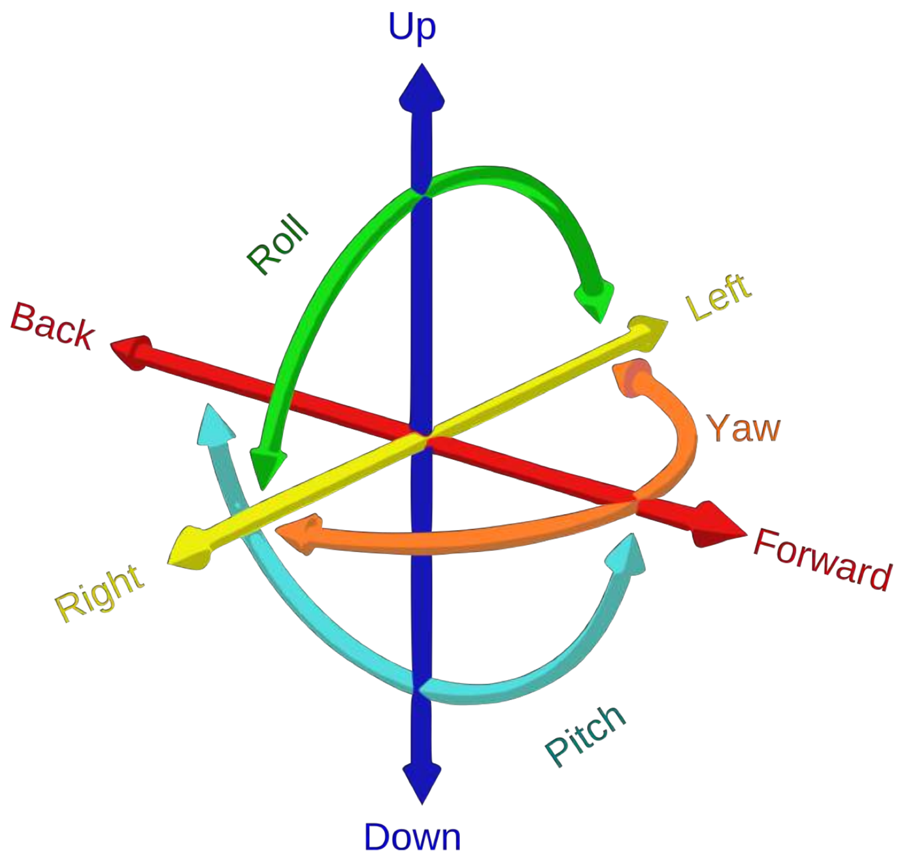
How is Graal being used today?

How is Graal being used today?

- Graal as a Java JIT
  - JVM bytecode → Graal → machine code
- Graal as a custom Java JIT
  - JVM bytecode → Graal → custom phases → Graal → machine code
- Graal as a Java AOT
  - JVM classes → Graal → executable or shared library
- Graal as a dynamic language JIT
  - source → interpreter → Truffle PE → Graal → machine code
- Graal as a native language JIT
  - LLVM bitcode → Sulong → Truffle PE → Graal → machine code
- Graal as a language embedded
  - interpreter → Graal → executable or shared library

Why is there both Graal and GraalVM?

“We’re running on Graal / GraalVM”

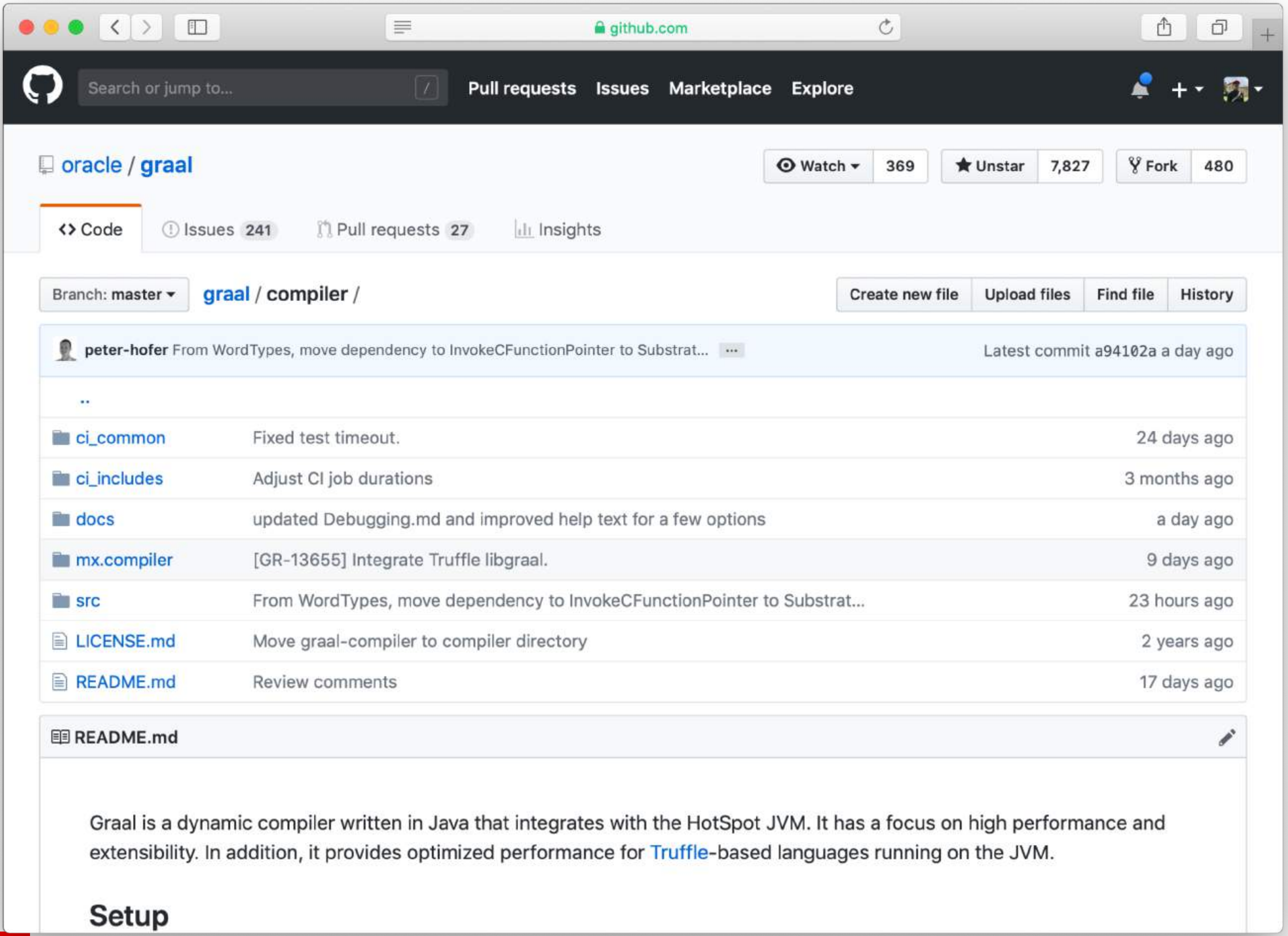


https://commons.wikimedia.org/wiki/File:6DOF.svg
Creative Commons Attribution-Share Alike 4.0 International

Graal and GraalVM?

- Graal is a compiler
  - Runs in different configurations
    - Ahead-of-time
    - Just-in-time
  - Used in different applications
    - Java compiler
    - Dynamic language compiler
    - Native language compiler
  - Distributed in different ways
    - Inside OpenJDK
    - In GraalVM

- GraalVM is a distribution of the wider Graal ecosystem
  - Compilers, tools, language implementations
  - Includes Graal configured as
    - A JVM language JIT compiler
    - A JVM language AOT compiler
    - A JIT compiler for dynamic languages
    - A JIT compiler for native languages
  - Realizes the potential of Graal
  - Graal is the enabler of all this



oracle / graal

graal / compiler /

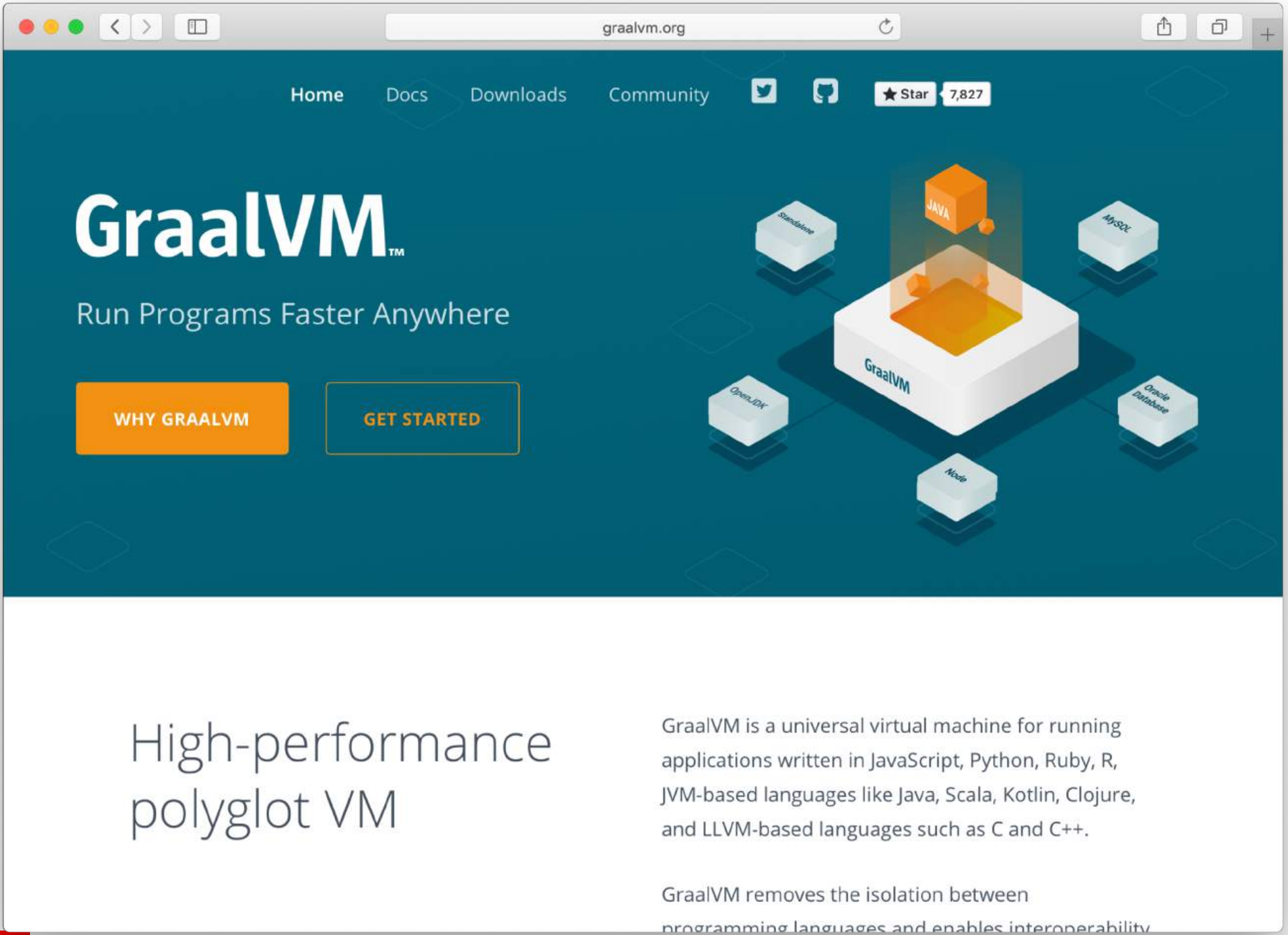
peter-hofer From WordTypes, move dependency to InvokeCFunctionPointer to Substrat... ··· Latest commit a94102a a day ago

..		
ci_common	Fixed test timeout.	24 days ago
ci_includes	Adjust CI job durations	3 months ago
docs	updated Debugging.md and improved help text for a few options	a day ago
mx.compiler	[GR-13655] Integrate Truffle libgraal.	9 days ago
src	From WordTypes, move dependency to InvokeCFunctionPointer to Substrat...	23 hours ago
LICENSE.md	Move graal-compiler to compiler directory	2 years ago
README.md	Review comments	17 days ago

README.md

Graal is a dynamic compiler written in Java that integrates with the HotSpot JVM. It has a focus on high performance and extensibility. In addition, it provides optimized performance for Truffle-based languages running on the JVM.

Setup

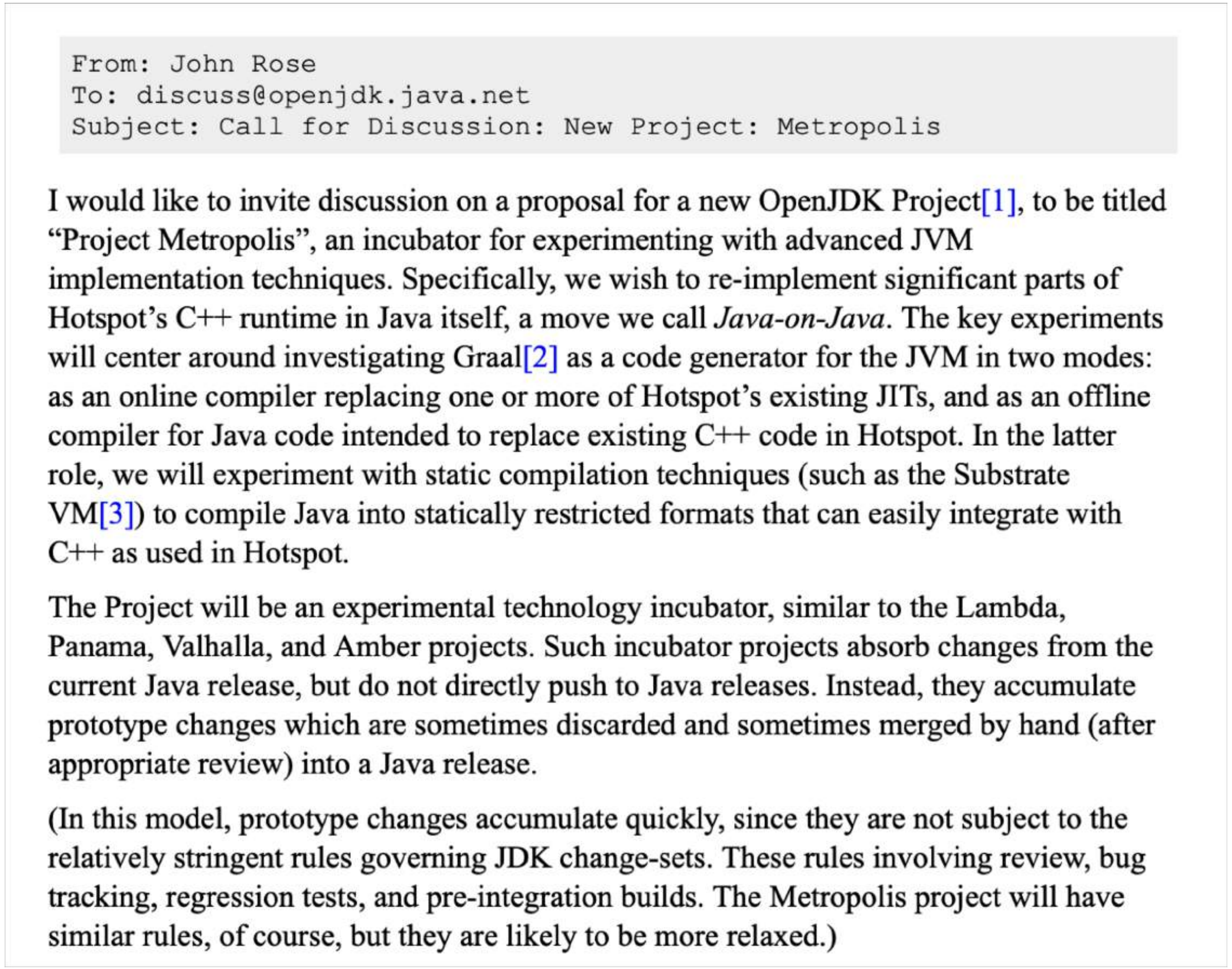


High-performance polyglot VM

GraalVM is a universal virtual machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Kotlin, Clojure, and LLVM-based languages such as C and C++.

GraalVM removes the isolation between

Where is Graal going in the future?

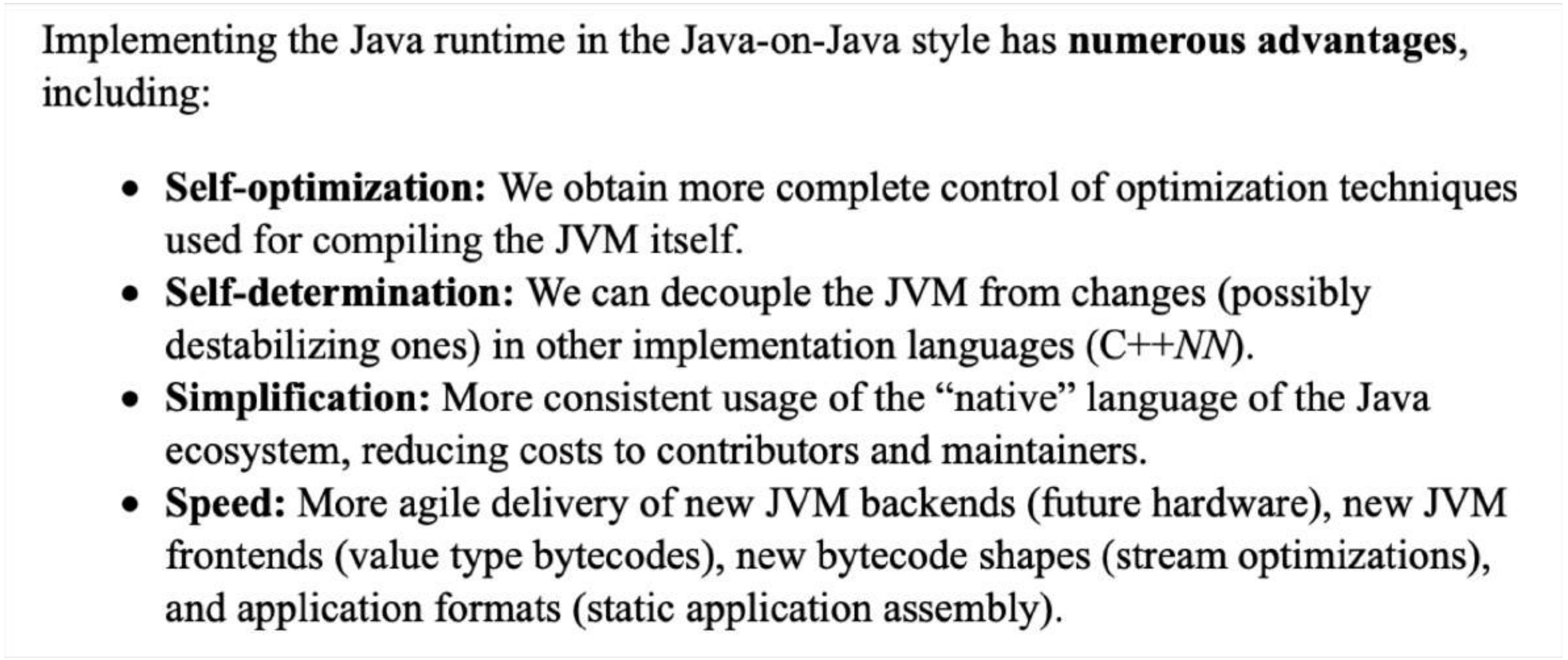


```
From: John Rose
To: discuss@openjdk.java.net
Subject: Call for Discussion: New Project: Metropolis
```

I would like to invite discussion on a proposal for a new OpenJDK Project[1], to be titled "Project Metropolis", an incubator for experimenting with advanced JVM implementation techniques. Specifically, we wish to re-implement significant parts of Hotspot's C++ runtime in Java itself, a move we call *Java-on-Java*. The key experiments will center around investigating Graal[2] as a code generator for the JVM in two modes: as an online compiler replacing one or more of Hotspot's existing JITs, and as an offline compiler for Java code intended to replace existing C++ code in Hotspot. In the latter role, we will experiment with static compilation techniques (such as the Substrate VM[3]) to compile Java into statically restricted formats that can easily integrate with C++ as used in Hotspot.

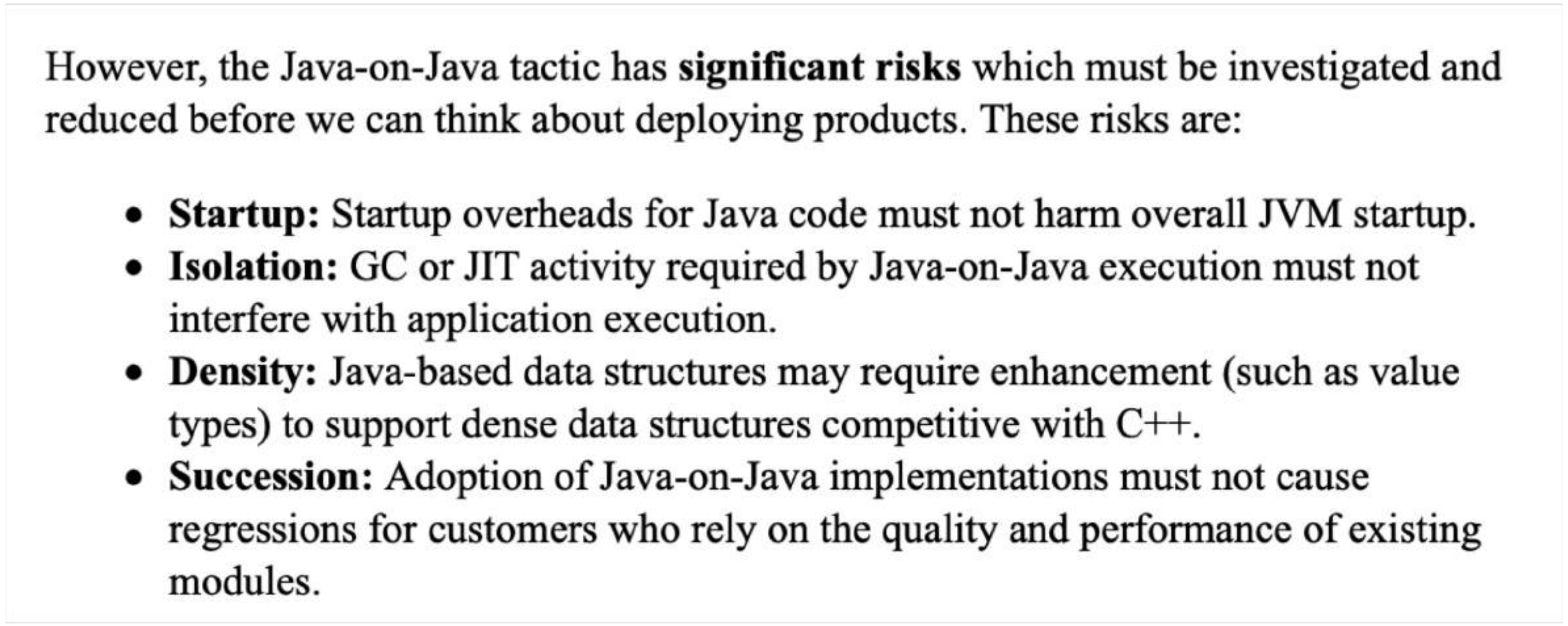
The Project will be an experimental technology incubator, similar to the Lambda, Panama, Valhalla, and Amber projects. Such incubator projects absorb changes from the current Java release, but do not directly push to Java releases. Instead, they accumulate prototype changes which are sometimes discarded and sometimes merged by hand (after appropriate review) into a Java release.

(In this model, prototype changes accumulate quickly, since they are not subject to the relatively stringent rules governing JDK change-sets. These rules involving review, bug tracking, regression tests, and pre-integration builds. The Metropolis project will have similar rules, of course, but they are likely to be more relaxed.)



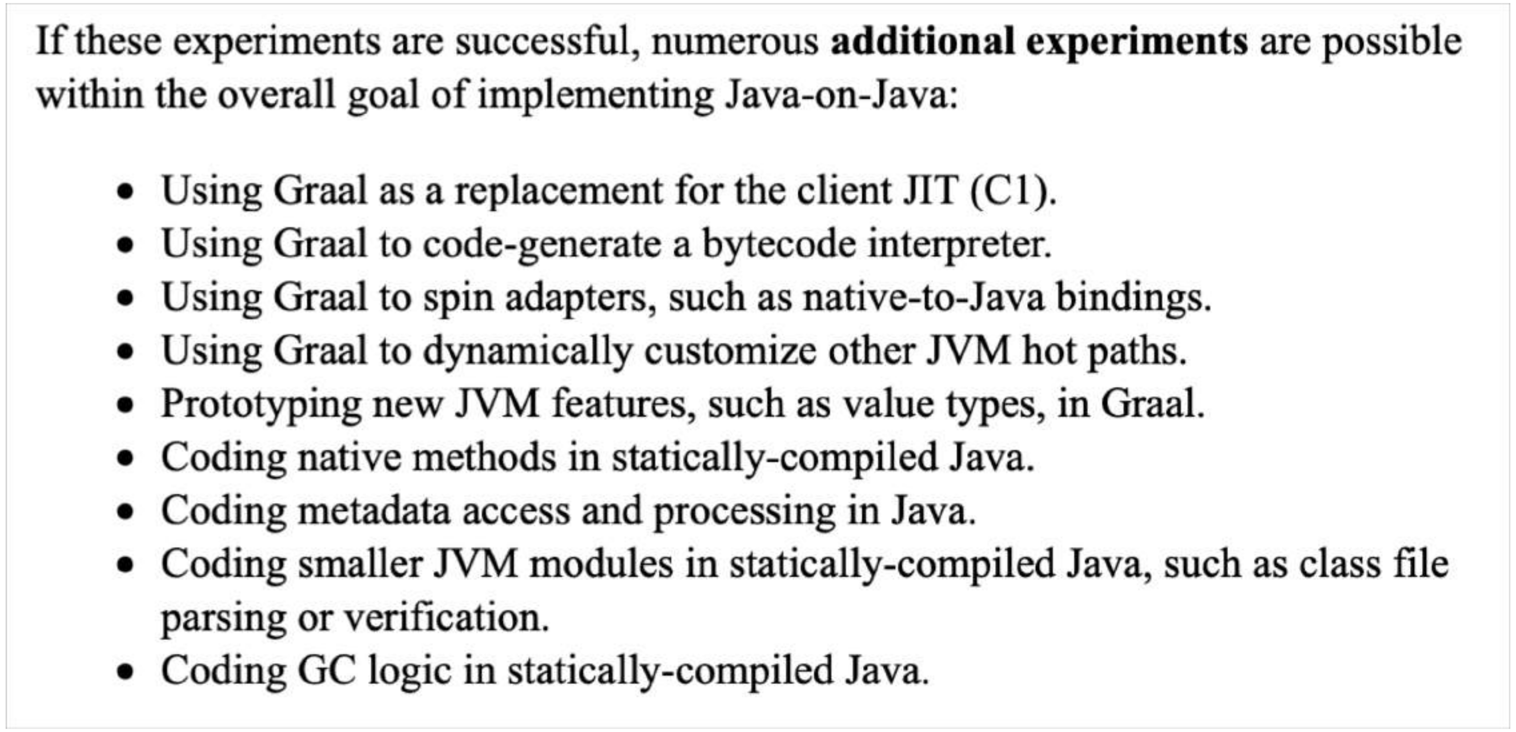
Implementing the Java runtime in the Java-on-Java style has **numerous advantages**, including:

- **Self-optimization:** We obtain more complete control of optimization techniques used for compiling the JVM itself.
- **Self-determination:** We can decouple the JVM from changes (possibly destabilizing ones) in other implementation languages (C++*NN*).
- **Simplification:** More consistent usage of the "native" language of the Java ecosystem, reducing costs to contributors and maintainers.
- **Speed:** More agile delivery of new JVM backends (future hardware), new JVM frontends (value type bytecodes), new bytecode shapes (stream optimizations), and application formats (static application assembly).



However, the Java-on-Java tactic has **significant risks** which must be investigated and reduced before we can think about deploying products. These risks are:

- **Startup:** Startup overheads for Java code must not harm overall JVM startup.
- **Isolation:** GC or JIT activity required by Java-on-Java execution must not interfere with application execution.
- **Density:** Java-based data structures may require enhancement (such as value types) to support dense data structures competitive with C++.
- **Succession:** Adoption of Java-on-Java implementations must not cause regressions for customers who rely on the quality and performance of existing modules.



If these experiments are successful, numerous **additional experiments** are possible within the overall goal of implementing Java-on-Java:

- Using Graal as a replacement for the client JIT (C1).
- Using Graal to code-generate a bytecode interpreter.
- Using Graal to spin adapters, such as native-to-Java bindings.
- Using Graal to dynamically customize other JVM hot paths.
- Prototyping new JVM features, such as value types, in Graal.
- Coding native methods in statically-compiled Java.
- Coding metadata access and processing in Java.
- Coding smaller JVM modules in statically-compiled Java, such as class file parsing or verification.
- Coding GC logic in statically-compiled Java.

Learning more and getting in touch

Learning more and getting in touch

- Website – graalvm.org
- Papers – graalvm.org/community/publications
- GitHub – github.com/oracle/graal
- Gitter – gitter.im/graalvm/graal-core
- Twitter – @GraalVM
- Mailing lists – graal-dev@openjdk.java.net, graalvm-users@oss.oracle.com
- Email – chris.seaton@oracle.com

Team

Oracle
Florian Angerer
Danilo Ansaloni
Stefan Anzinger
Martin Balin
Cosmin Basca
Daniele Bonetta
Dušan Bálek
Matthias Brantner
Lucas Braun
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Svatopluk Dědic
Martin Entlicher
Pit Fender
Francois Farquet
Brandon Fish
Matthias Grimmer
Christian Häubl
Peter Hofer
Bastian Hossbach
Christian Humer
Tomáš Hůrka
Mick Jordan

Oracle (continued)
Vojin Jovanovic
Anantha Kandukuri
Harshad Kasture
Cansu Kaynak
Peter Kessler
Duncan MacGregor
Jiří Maršík
Kevin Menard
Miloslav Metelka
Tomáš Myšík
Petr Pišl
Oleg Pliss
Jakub Podlešák
Aleksandar Prokopec
Tom Rodriguez
Roland Schatz
Benjamin Schlegel
Chris Seaton
Jiří Sedláček
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Boris Spasojevic
Lukas Stadler
Codrut Stancu

Oracle (continued)
Jan Štola
Tomáš Stupka
Farhan Tauheed
Jaroslav Tulach
Alexander Ulrich
Michael Van De Vanter
Aleksandar Vitorovic
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger
Tomáš Zezula
Yudi Zheng

Red Hat
Andrew Dinn
Andrew Haley

Intel
Michael Berg

Twitter
Chris Thalinger

Oracle Interns
Brian Belleville
Ondrej Douda
Juan Fumero
Miguel Garcia
Hugo Guiroux
Shams Imam
Berkin Ilbeyi
Hugo Kapp
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
Tristan Overney
Aleksandar Pejovic
David Piorkowski
Philipp Riedmann
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Oracle Alumni
Erik Eckstein
Michael Haupt
Christos Kotselidis
David Leibs
Adam Welc
Till Westmann

JKU Linz
Hanspeter Mössenböck
Benoit Daloze
Josef Eisl
Thomas Feichtinger
Josef Haider
Christian Huber
David Leopoldseder
Stefan Marr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

TU Berlin:
Volker Markl
Andreas Kunft
Jens Meiners
Tilmann Rabl

University of Edinburgh
Christophe Dubach
Juan José Fumero Alfonso
Ranjeet Singh
Toomas Remmelg

LaBRI
Floréal Morandat

University of California, Irvine
Michael Franz
Yeoul Na
Mohaned Qunaibit
Gulfem Savrun Yeniceri
Wei Zhang

Purdue University
Jan Vitek
Tomas Kalibera
Petr Maj
Lei Zhao

T. U. Dortmund
Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis
Duncan Temple Lang
Nicholas Ulle

University of Lugano, Switzerland
Walter Binder
Sun Haiyang

Safe Harbor Statement

The preceding is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Integrated Cloud

Applications & Platform Services