Urmărirea mașinii virtuale pentru o compilare

- cu -XX:+PrintCompilation sau -Dgraal.PrintCompilation=true pentru notificări și informații sumare -XX:+TraceDeoptimization poate fi utilizată pentru a vedea dacă durează prea mult compilarea
- De exemplu inspectarea compilării pentru o metodă dacă vreau să văd structurile de date utilizate când se compilează Node.updateUsages, se folosește următoarea comandă
- > mx vm -XX:+UseJVMCICompiler -XX:+BootstrapJVMCI -XX:-TieredCompilation -Dgraal.Dump= -Dgraal.MethodFilter=Node.updateUsages -version
- Bootstrapping JVMCI....Dumping debug output in /home/bugs/graal/graal/compiler/dumps/1497910458736
- ..... in 38177 ms (compiled 5206 methods)
- java version "1.8.0_121"
- Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
- Java HotSpot(TM) 64-Bit Server VM (build 25.71-b01-internal-jvmci-0.26, mixed mode)
- > find dumps/1497910458736 -type f
- dumps/1497910458736/HotSpotCompilation-539[org.graalvm.compiler.graph.Node.updateUsages(Node, Node)].bgv
- dumps/1497910458736/HotSpotCompilation-539[org.graalvm.compiler.graph.Node.updateUsages(Node, Node)].cfg

Dumping

- **HIR graphs** (de ex instante ale Graph) către Ideal Graph Visualizer (IGV), și
- **LIR register allocation si generated code** către C1Visualizer
- optiunea -Dgraal.Dump

Exemplu metrici (fișier ieșire) -Dgraal.MetricsFile

- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseNodes_PhaseSuite;1
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseNodes_GraphBuilderPhase;1
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseCount_GraphBuilderPhase;1
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseMemUse_GraphBuilderPhase_Accm;896536
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseMemUse_GraphBuilderPhase_Flat;896536
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseTime_GraphBuilderPhase_Accm;31095000
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseTime_GraphBuilderPhase_Flat;27724000

- *coloanele din fișierul ieșire CSV al -Dgraal.AggregatedMetricsFile sunt:*
  - metric_name:
  - metric_value:
  - metric_unit:

metrici

- **compilable:**
- **compilable_identity:**
- **compilation_nr:**
- **compilation_id:**
- **metric_name:**
- **metric_value:**
- **metric_unit:**

metrici

ca și TimerKey, o valoare acumulată și flat este raportată de tracker-ul pentru memorie

CompilationMemory_Accm=100000 bytes

CompilationMemory_Flat=1000 bytes

Pentru ambele opțiuni -Dgraal.AggregatedMetricsFile și -Dgraal.MetricsFile ieșirea este determinată de sufixul numelui

.csv produce format cu separare punct și virgulă

Coloanele rezultate din -Dgraal.MetricsFile CSV sunt:

metrici

CompilationTime_Accm=100.0 ms
CompilationTime_Flat=10.0 ms

O cheie MemUseTrackerKey urmărește rezervarea de memorie pentru un scop specific de ex

- // declaratie
- private static final MemUseTrackerKey CompilationMemory = DebugContext.memUseTracker("CompilationMemory");
- // utilizare
- DebgContext debug = ...;
- try (DebugCloseable a = CompilationMemory.start(debug)) {
- ...
- }
- // octeții folosiți aici vor fi sumați la valoarea debug `CompilationMemory`

metrici

CounterKey (cantitativ)

```
// declaratiie
private static final CounterKey ByteCodesCompiled =
DebugContext.counter("ByteCodesCompiled");
// utilizare
DebugContext debug = ...;
long compiled = ... ;
ByteCodesCompiled.add(debug, compiled);
```

TimerKey (timp scurs)

```
// declaratie
private static final TimerKey CompilationTime = DebugContext.timer("CompilationTime");
// usage
DebgContext debug = ...;
try (DebugCloseable scope = CompilationTime.start(debug)) {
   ...
}
// timp scurs pentru respectivul scop se va adăuga la `CompilationTime` din `debug`
```

Metrici

- Fiecare metrică are un nume unic.
- Metricile sunt colectate pentru fiecare compilare
- Pentru a le lista -Dgraal.MetricsFile option.

Logging

Ca o facilitate suplimentar[

```
DebugContext debug = ...;
InstalledCode code = null;
try (Scope s = debug.scope("CodeInstall", method))
{
   code = ...
   debug.log("installed code for %s", method);
} catch (Throwable e) {
   throw debug.handle(e);
}
```

exemplu Sulong native pipe

```
#include <jni.h>
#include "com_oracle_truffle_llvm_pipe_CaptureNativeOutput.h"

#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <errno.h>

static bool check_error(JNIEnv *env, int ret) {
  if (ret < 0) {
    char *message = strerror(errno);
    jclass ioex = env->FindClass("java/io/IOException");
    env->ThrowNew(ioex, message);
    return true;}
else {return false;}
}

JNIEXPORT jint JNICALL
Java_com_oracle_truffle_llvm_pipe_CaptureNativeOutput_startCapturing(JNIEn
v *env, jclass self, jint stdFd, jstring filename) {
  const char *path = env->GetStringUTFChars(filename, NULL);
  int fd = open(path, O_WRONLY);
  bool error = check_error(env, fd);
  env->ReleaseStringUTFChars(filename, path);
  if (error) {  return -1; }

  int oldFd = dup(stdFd);
  if (check_error(env, oldFd)) {
    close(fd);
    return -1;  }
  int result = dup2(fd, stdFd);
  if (check_error(env, result)) {
    close(fd);
    close(oldFd);
    return -1;  }
  close(fd);
  return oldFd;
}

JNIEXPORT void JNICALL
Java_com_oracle_truffle_llvm_pipe_CaptureNativeOutput_stopCapturing(JN
IEnv *env, jclass self, jint oldStdOut, jint oldStdErr) {
  if (check_error(env, fflush(stdout))) {    return;  }
  if (check_error(env, fflush(stderr))) {    return;  }
  if (check_error(env, dup2(oldStdOut,
com_oracle_truffle_llvm_pipe_CaptureNativeOutput_STDOUT)))
{    return;  }
  if (check_error(env, dup2(oldStdErr,
com_oracle_truffle_llvm_pipe_CaptureNativeOutput_STDERR))) {    return;  }
  if (check_error(env, close(oldStdOut))) {
    return;
  }
  check_error(env, close(oldStdErr));
}
```

detecția unei durate prea mare de execuție

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import org.graalvm.polyglot.Context;
import org.graalvm.polyglot.PolyglotException;
import org.graalvm.polyglot.Value;


public class CancelExecution {

   public static void main(String[] args) throws
InterruptedException {
      ExecutorService
serviceExecutors.newFixedThreadPool(1);
      try {
         Context context = Context.create("js");
  // se trimit pt testare scipt-uri cu bucle infinite
Future<Value> future = service.submit(() -> context.eval("js",
"while(true);"));
          // se asteapta un timp pentru a putea porni executia
         Thread.sleep(1000);

         // se opreste executarea si se inchide contextul
// se poate face in orice thread in paralel
// se mai poate utiliza context.close(true) pentru a inchide toate
contextele aflata in executie
         context.close(true);
         try {  future.get();
         } catch (ExecutionException e)
            {
            PolyglotException polyglotException =
(PolyglotException) e.getCause();
            polyglotException.printStackTrace();
            // dupa oprirea executiei thread va fi generata o exceptie
             PolyglotException cu flagu-ul cancelled activat.
            assert polyglotException.isCancelled();
         }
      } finally {
         service.shutdown();
      }
   }

}
```

Script-uri în paralel

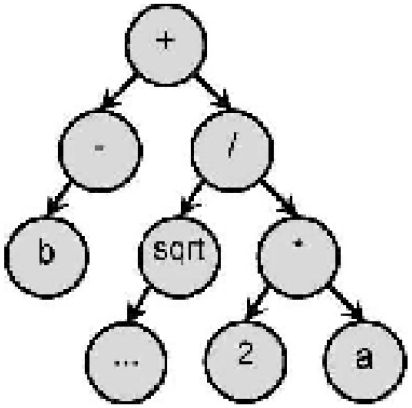
```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import org.graalvm.polyglot.Context;
public class ParallelEval
{
  public static void main(String[] args) throws InterruptedException, ExecutionException
 {
       ExecutorService service = Executors.newFixedThreadPool(1);
       Context context = Context.create("js");
       Future<Integer> future = service.submit(() -> context.eval("js", "21 + 21").asInt());
       // executa altceva in timp ce scriptul Java se executa
       int result = future.get();
       assert 42 == result;
       context.close();
    }
}
```

Programare în PolyGlot

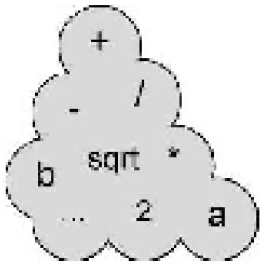
```
import org.graalvm.polyglot.*;

public class HelloPolyglotWorld {

public static void main(String[] args) throws Exception {
 System.out.println("Hello polyglot world Java!");
 Context context = Context.create();
 context.eval("js", "print('Hello polyglot world JavaScript!');");
 context.eval("ruby", "puts 'Hello polyglot world Ruby!'");
 context.eval("R", "print('Hello polyglot world R!');");
 context.eval("python", "print('Hello polyglot world Python!');");
 }
}
```

Optimizare performanțe cu Graal
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

•execute b
•check that the negate method in Float has not changed
•calculate negation execute b
•check that the power method in Float has not changed
•calculate power execute a
•check that the multiply method in Float has not changed
•calculate multiplication execute c
•check that the multiply method in Float has not changed
•calculate multiplication
•check that Math has not changed
•check that the sqrt method in Math has not changed
•calculate sqrt execute a
•check that the multiply method in Float has not changed
•calculate multiplication
•check that the division method in Float has not changed
•calculate division

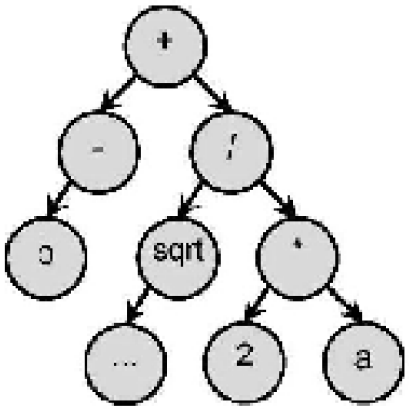


execute b calculate negation execute b calculate power execute a
calculate multiplication execute c
calculate multiplication calculate sqrt execute a
calculate multiplication calculate divis on

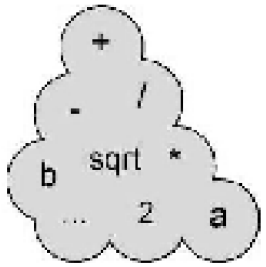


Optimizare performanțe cu Truffle
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

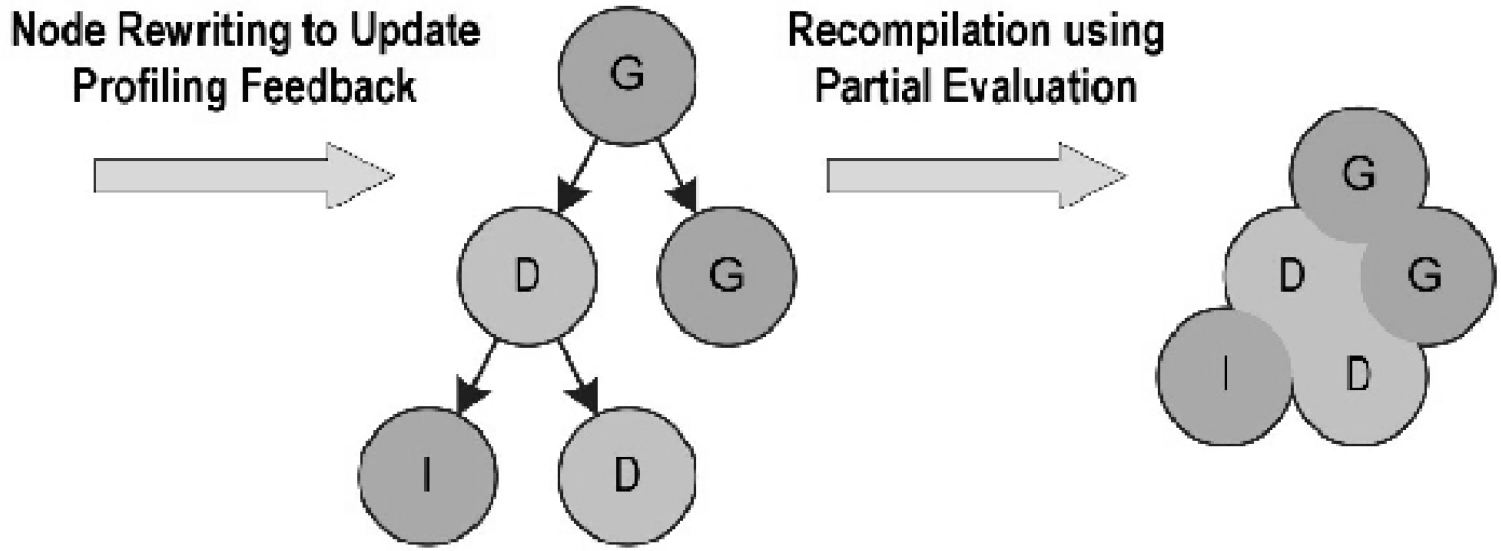
•execute b
•check that b is a Float
•check that the negate method in Float has not changed
•calculate negation
•check the result of that is a Float
•execute 2
•check that b is a Float
•check that the power method in Float has not changed
•calculate power
•check the result of that is a Float
•execute a
•check that a is a Float
•check that the multiply method in Float has not changed
•calculate multiplication
•check the result of that is a Float
•execute c
•check that c is a Float
•check that the multiply method in Float has not changed
•calculate multiplication
•check the result of that is a Float
•check that Math has not changed
•check that the sqrt method in Math has not changed
•calculate sqrt
•check the result of that is a Float
•execute a
•check that a is a Float
•check that the multiply method in Float has not changed
•calculate multiplication
•check the result of that is a Float
•check that the division method in Float has not changed
•calculate division



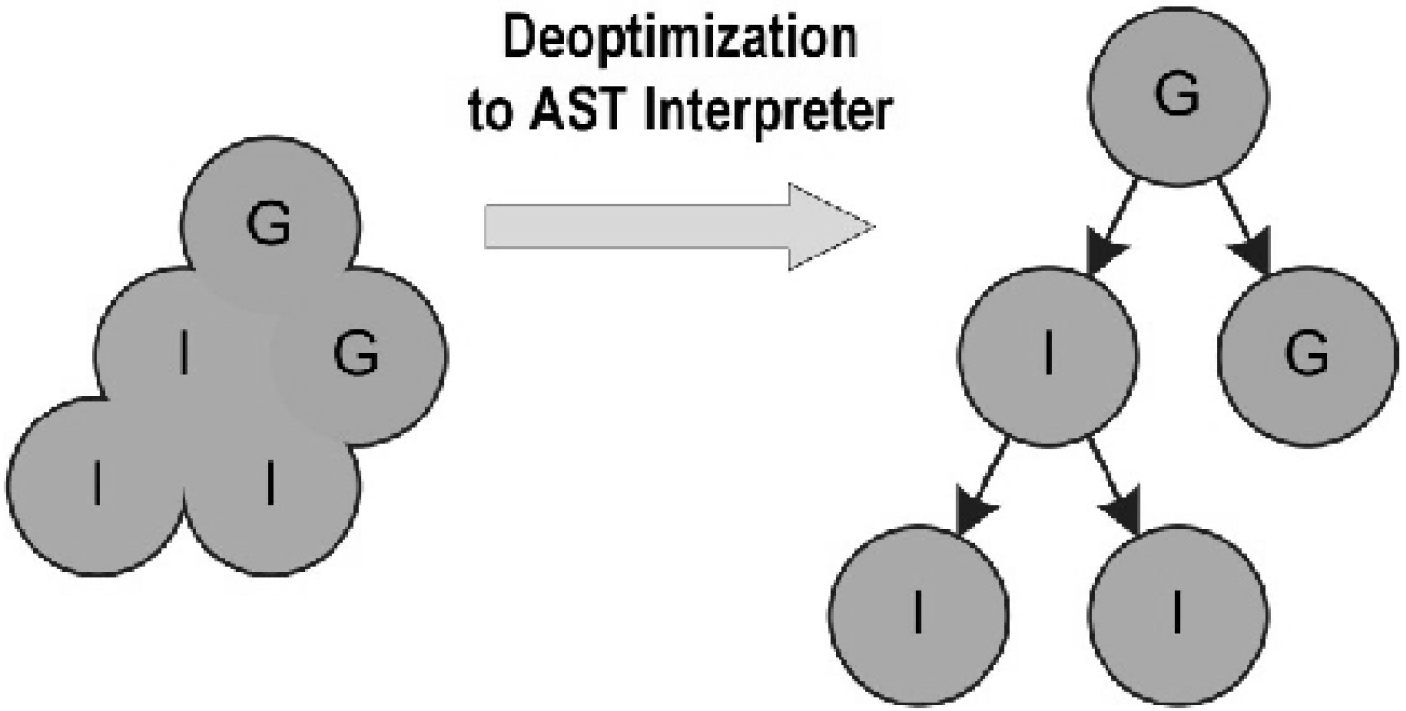
execute b
check that the negate method in Float has not changed
calculate negation execute b
check that the power method in Float has not changed
calculate power execute a
check that the multiply method in Float has not changed
calculate multiplication execute c
check that the multiply method in Float has not changed
calculate multiplication
check that Math has not changed
check that the sqrt method in Math has not changed
calculate sqrt execute a
check that the multiply method in Float has not changed
calculate multiplication
check that the division method in Float has not changed
calculate division



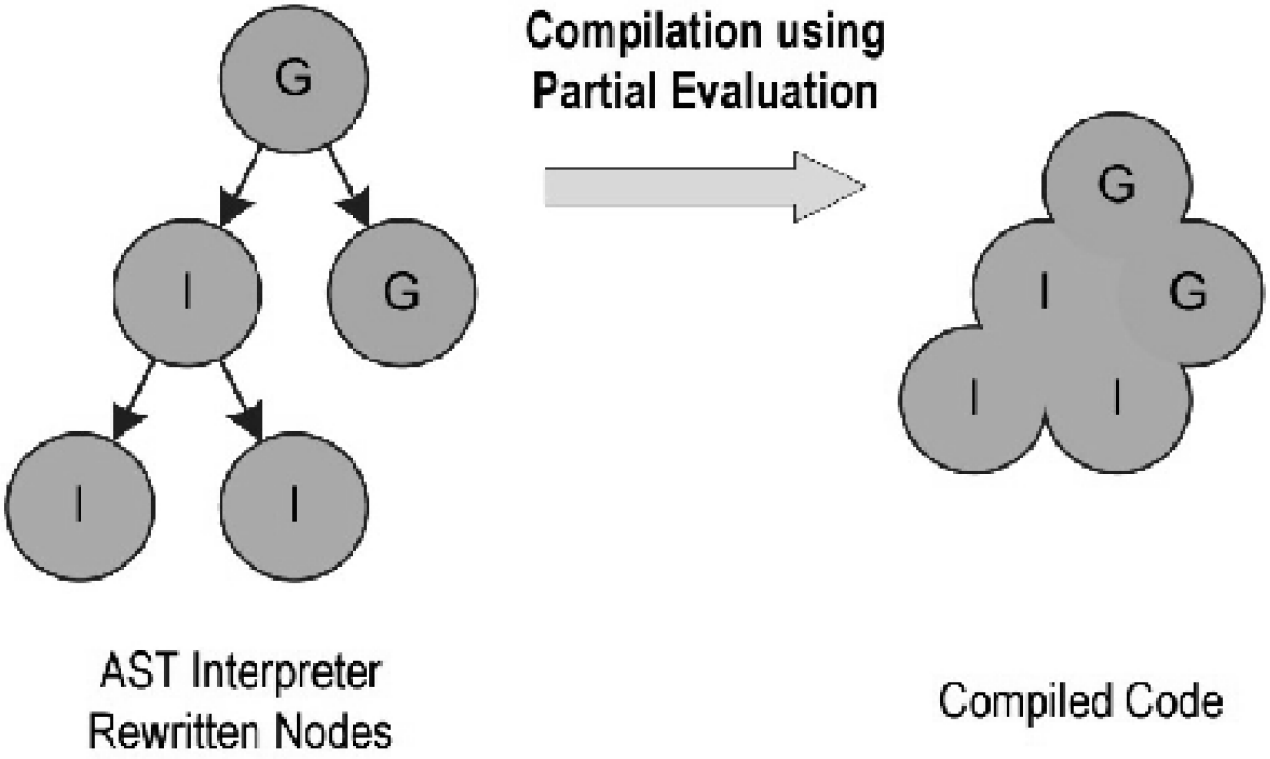
deoptimizarea



De-optimizarea (De-optimisation)



still..... folding



Ce este constant folding?

Teorema s-m-n a lui Kleene

Specializarea unui program de calcul x^n

A two-input program $p =$

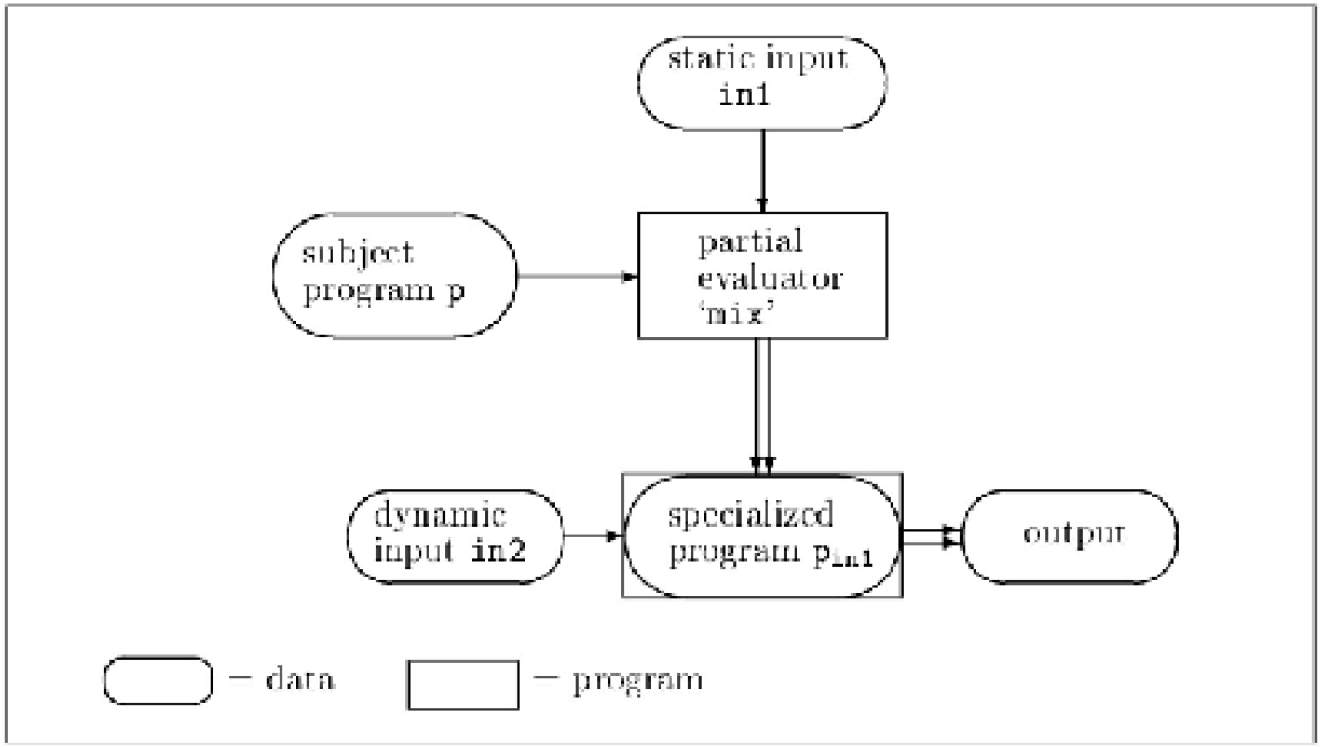
```
f(n,x) =if n = 0 then 1
        else if even(n) then f(n/2,x)↑2
        else x * f(n-1,x)
```

Program **p**, specialized to static input **n** = 5:

$p_5 =$

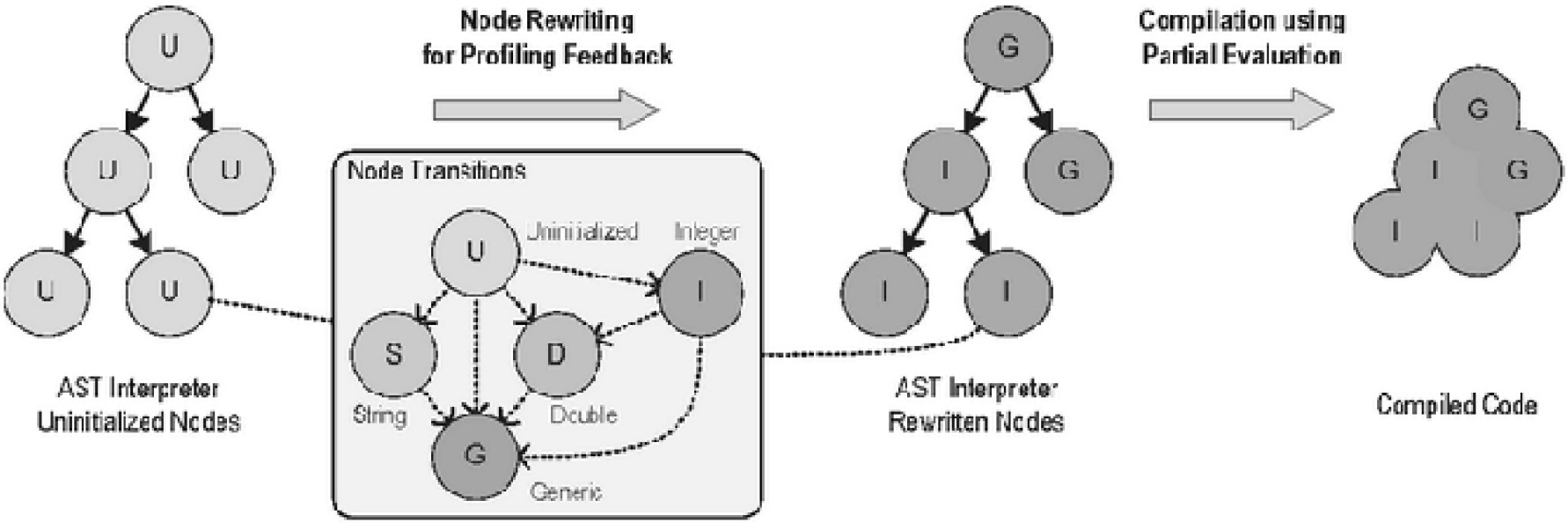
```
f5(x) = x * ((x↑2)↑2)
```

evaluator partial



Evaluare parțială

Reorganizarea nodurilor în arborele de evaluare



Cum se obțin astfel de performanțe?

exemplu

```
//Limbaj cu tipuri slabe
a  = 9
b = "9"
c = concatenate(a, b)  // rezultat "99"
d = add(a, b)          // rezultat 18
```

```
//limbaj cu tipuri tari
a  = 9
b = "9"
c = concatenate(  str(a),  b)//Grrr.....
d = add(a,  int(b)  ) //Mrrr.....
```

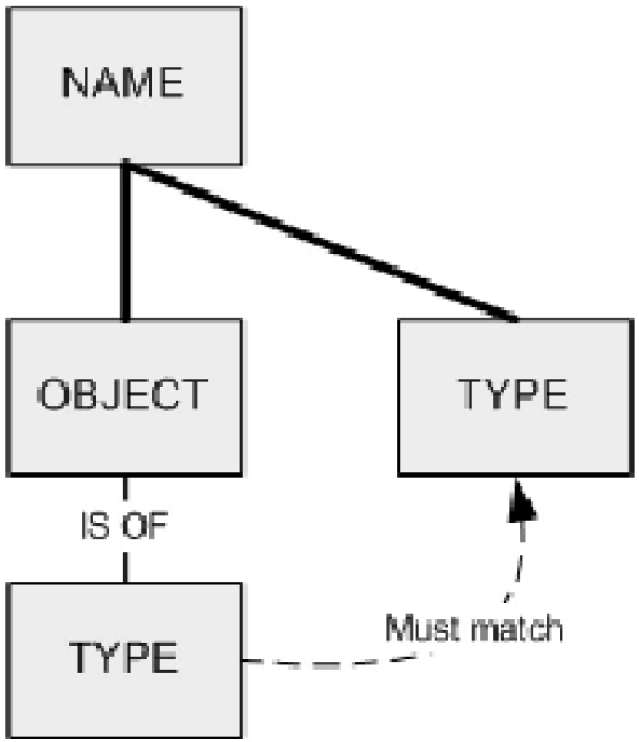
mere vs pere

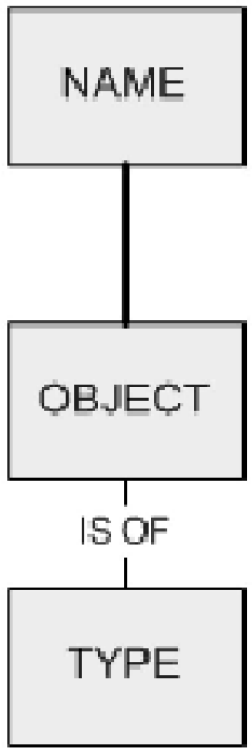
De exemplu într-un limbaj cu tipuri statice următoarea secvență de cod este ilegală

employeeName = 9 (s-a făcut asociere de tip int și gata!)

employeeName = "Steve Ferg"

Stabilirea dinamică vs statică a tipurilor într-un limbaj de programare





Stabilirea dinamică vs statică a tipurilor într-un limbaj de programare

Tipuri

Care este avantajul funcțiilor Curry?

Sunt mult mai flexibile decât funcțiile bazate pe tuple în special datorită faptului că pot fi aplicate lor parțial:

```
add' 1 :: Int → Int
take 5 :: [Int] → [Int]
drop 5 :: [Int] → [Int]
```

Convenții specifice funcțiilor Curry

Pentru a evita folosirea în exces a parantezelor atunci când se folosesc funcții Curry s-au adoptat două convenții simple:

- Utilizarea săgeții → care face asociere la dreapta

```
Int → Int → Int →
Int
```

- Ca o consecință apare o a doua convenție: funcțiile vor folosi asocierea la stânga

```
mult x y z
```

Funcții Curry

Funcțile cu argumente multiple sunt permise prin întoarcerea funcțiilor ca rezultat

```
add'     :: Int → (Int → Int)
add' x y = x+y
```

add și add' produc acelasi rezultat final dar add primește două argumente simultan în timp ce add' ia câte unul odată

```
add  :: (Int,Int) → Int
add' :: Int → (Int → Int)
```

Funcțiile cu mai mult de două argumente pot fi transformate în functii Curry prin apel recursiv de funcții la parametri

```
mult       :: Int → (Int → (Int → Int))
mult x y z = x*y*z
```

Expresii Lambda

Prin utilizarea lor funcțiile pot fi create fără a le asigna un nume explicit:

```
λ x → x+x
```

În matematică este folosit simbolul $\mapsto$, deci $x \mapsto x+x$.

La ce e bun calculul Lambda ?

Expresiile Lambda pot fi folosite pentru a da un înteles formal functiilor Curry.

De exemplu:

```
add x y = x+y
```

înseamna

```
add = λ x → (λ y → x+y)
```

compilatoare și translatoare

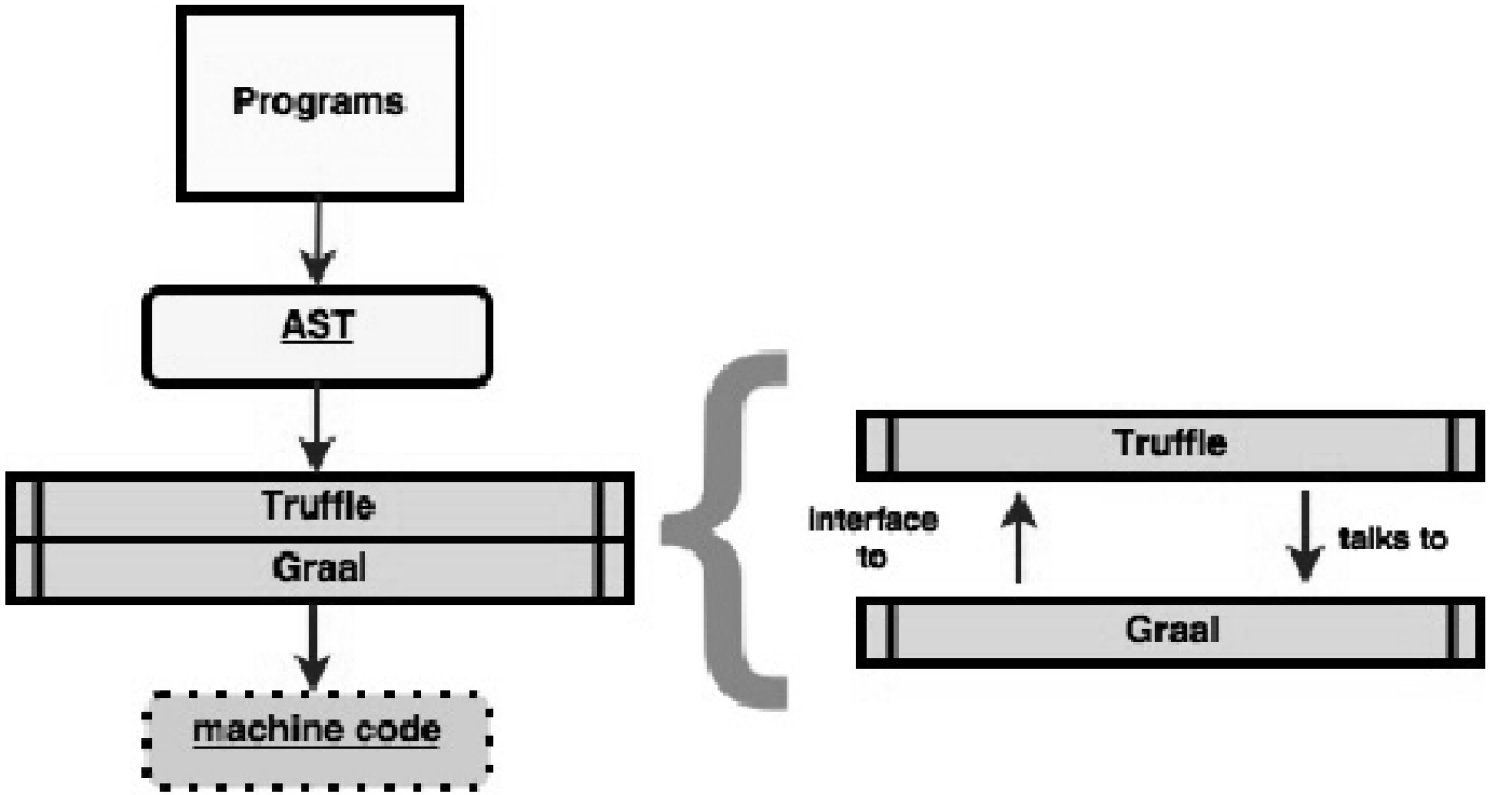
- calcul funcțional
- church &turing
- Untyped λ-calculus

Creșteri de performață obținute prin utilizare Graal

Datele de mai jos sunt date de ORACLE pe baza unor analize complexe care au implicat utilizarea unui numar mare de benchmark-uri (programe test standard)

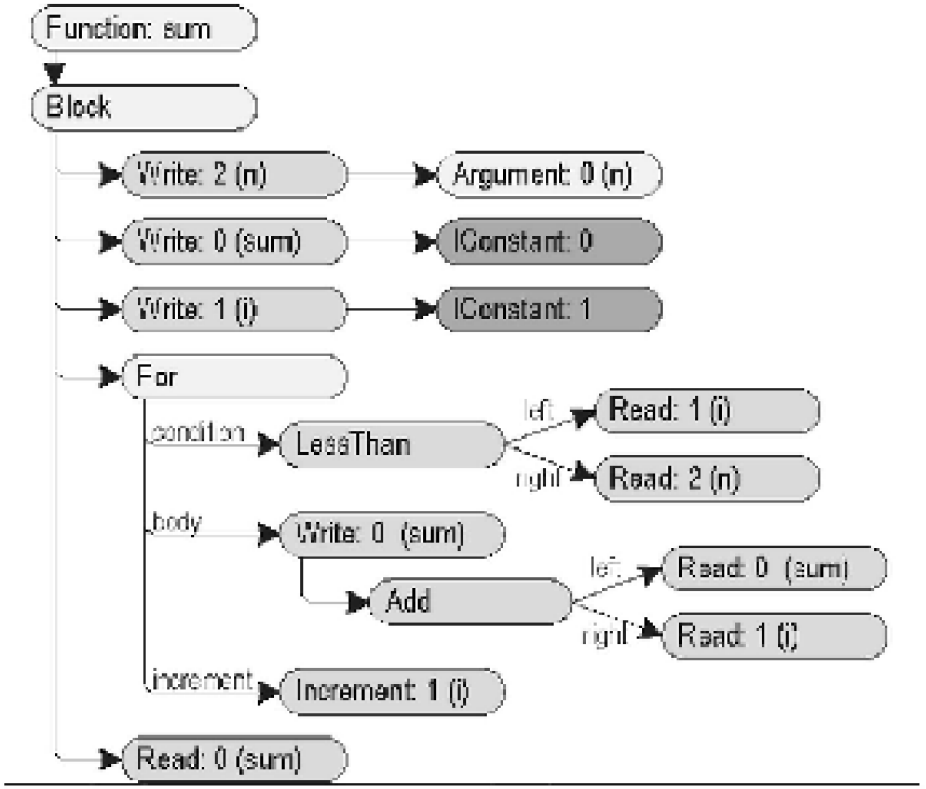
	Range			Speedup (Geomean)	Comparison Versus
Java	0.8x	-	2x	1.1x	JDK8
Scala	0.8x	-	2x	1.3x	JDK8
JavaScript	0.5x	-	1.5x	1.05x	Google V8
Ruby	1x	-	100x	5x	JRuby
R	1x	-	100x	5x	GNU R
C/C++	0.4x	-	1.2x	0.9x	LLVM native

Maniera de tratare a uni program

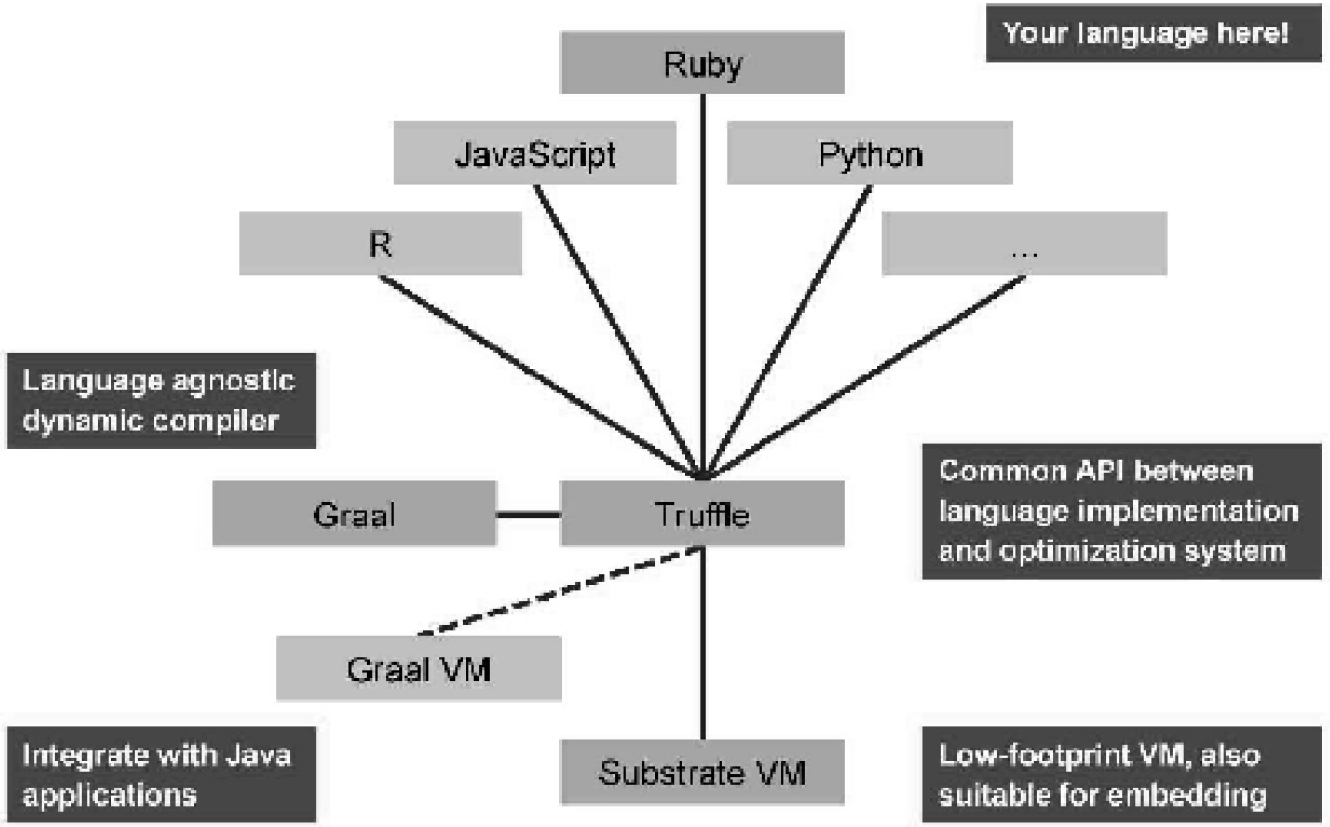


Ce este un AST-Abstract Syntax Tree

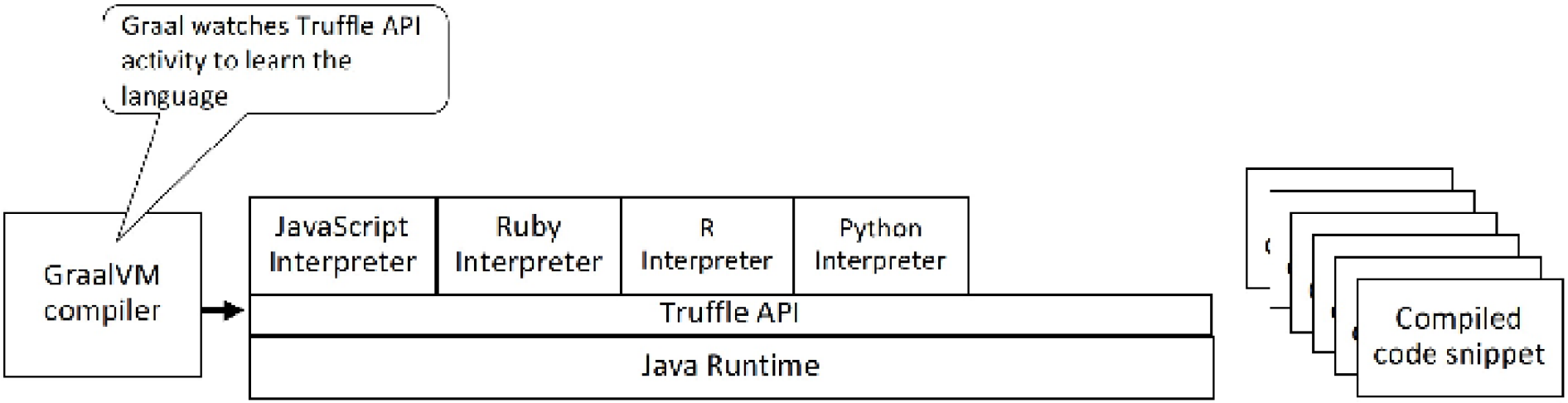
```
function sum(n) {
  var sum = 0;
  for (var i = 1; i < n; i++) {
    sum += i;
  }
  return sum;
}
```



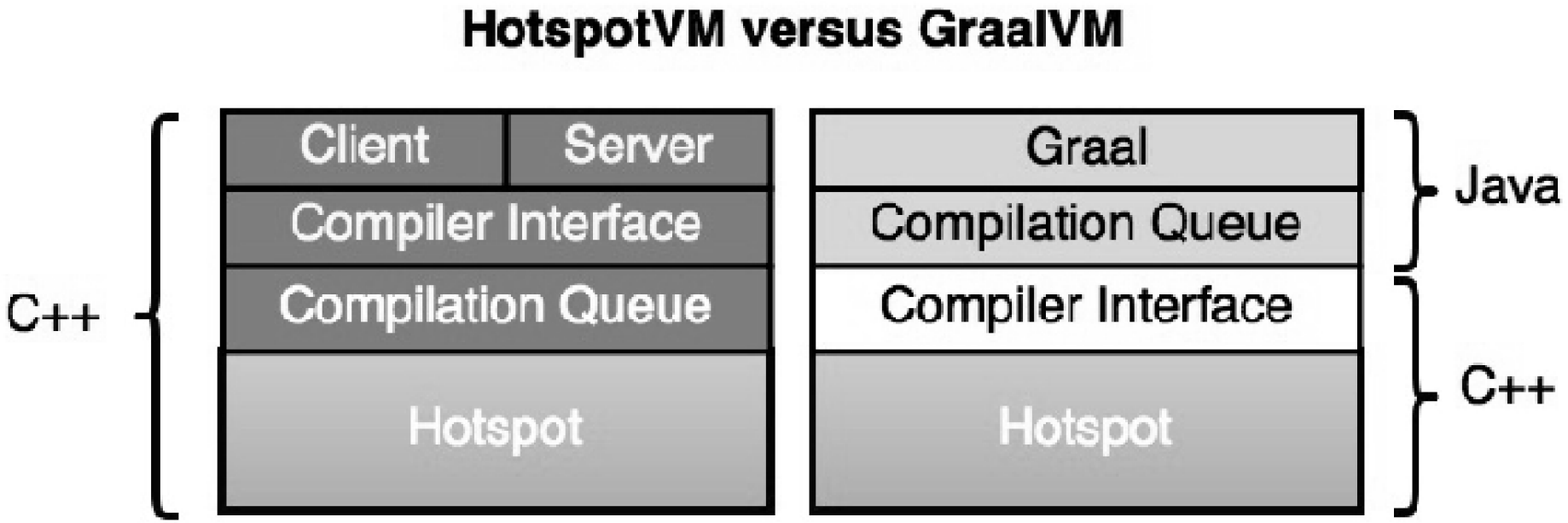
Limbaje suportate



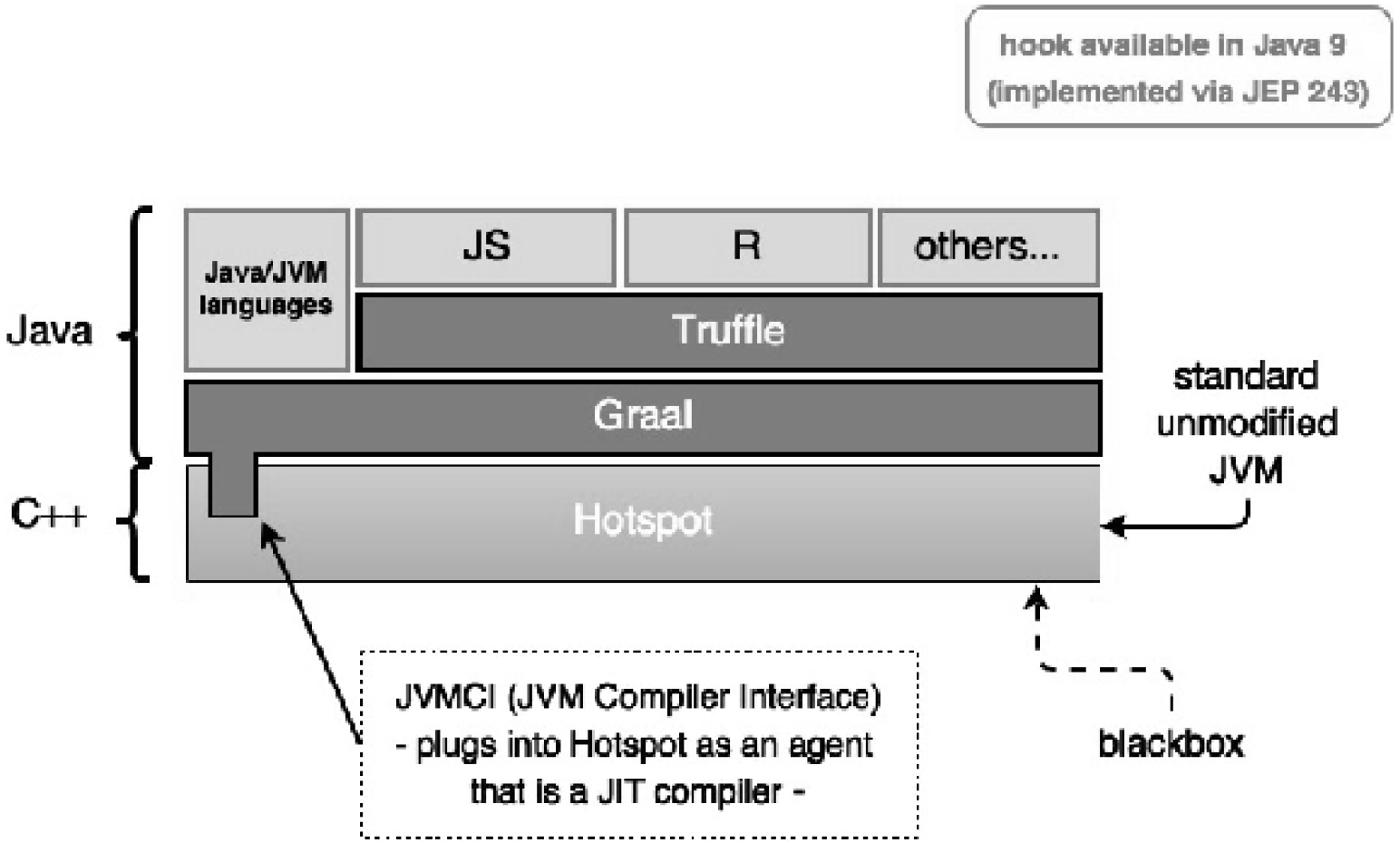
GraalVM este o virtualizare a Polyglot (Multilingual)



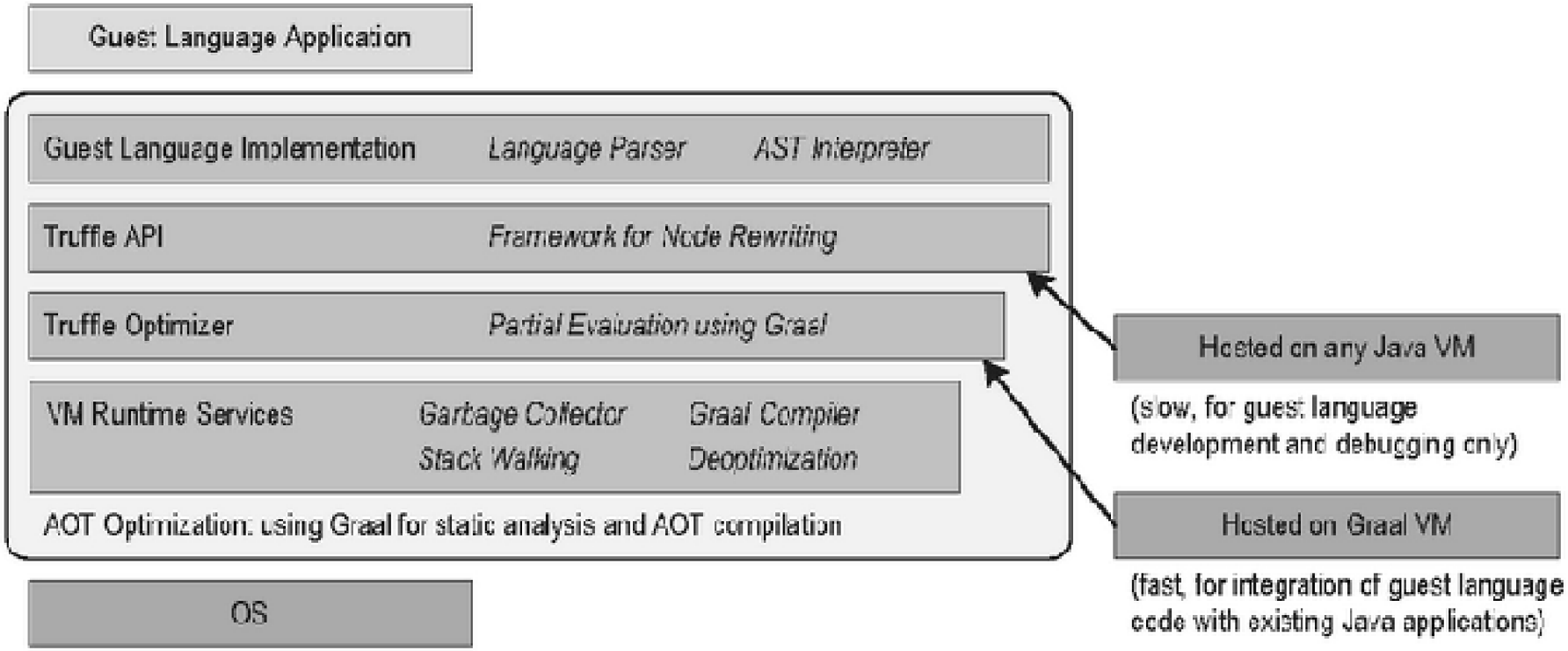
HotSpot


HotspotVM versus GraalVM
C++
Client
Server
Compiler Interface
Compilation Queue
Hotspot
Graal
Compilation Queue
Compiler Interface
Hotspot
Java
C++

Structura internă



Arhitectura originală/inițială



Cum s-a ajuns aici?

Wiirthinger et al 2013
Oracle Labs - Institute for System Software.
Johannes Kepler University Linz

Evoluție?

Java

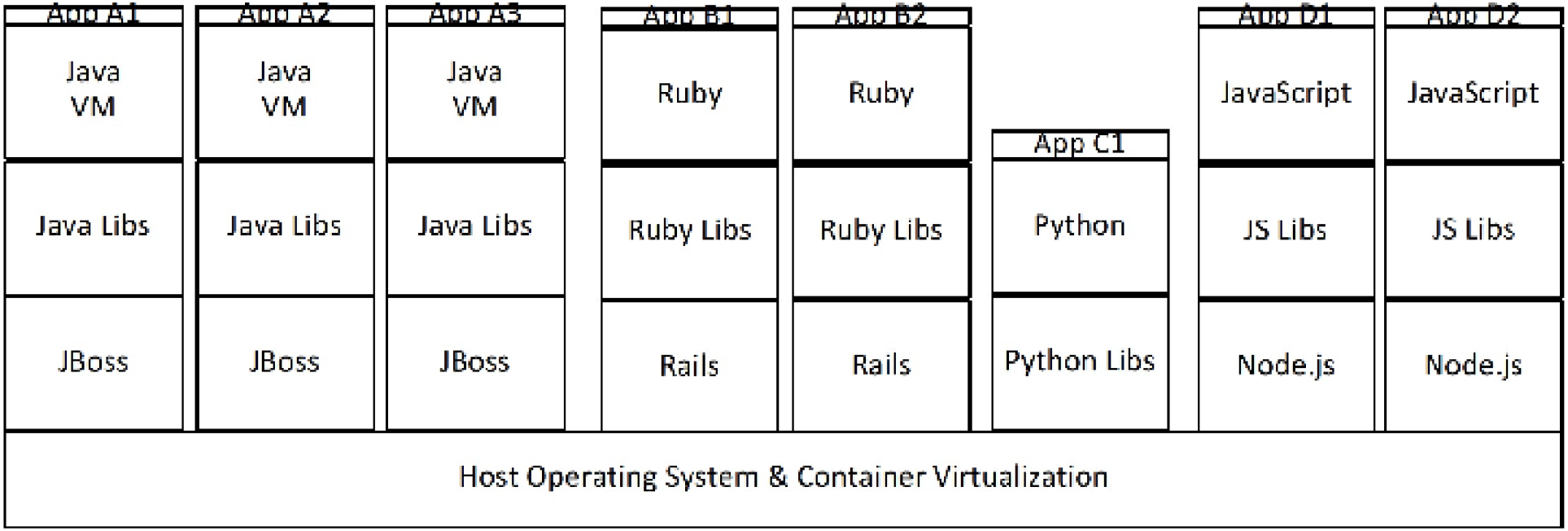
JVM

Poliglot

Graal

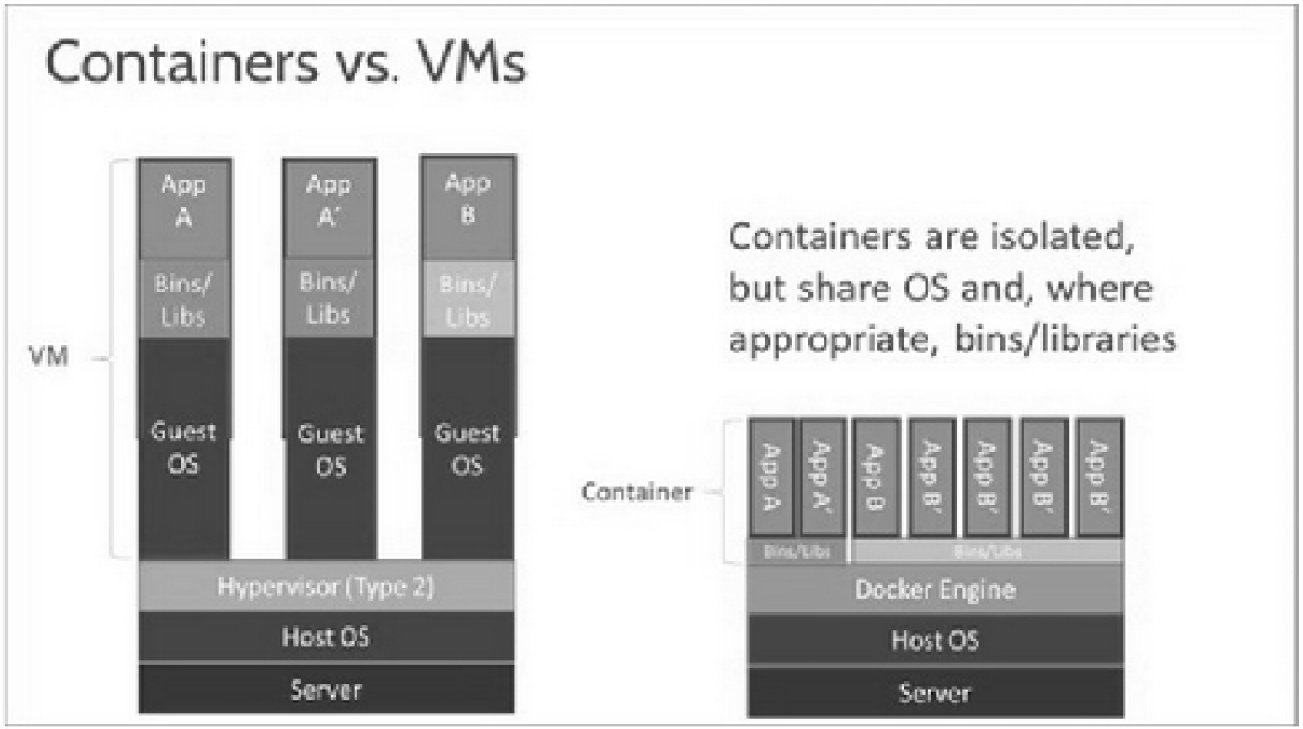
Aplicatiile din containere sunt și ele mari

- “Hello World” în Java: 24 MB
- “Hello World” în JavaScript (V8): 18 MB
- “Hello World” în Ruby/Rails: 8 MB

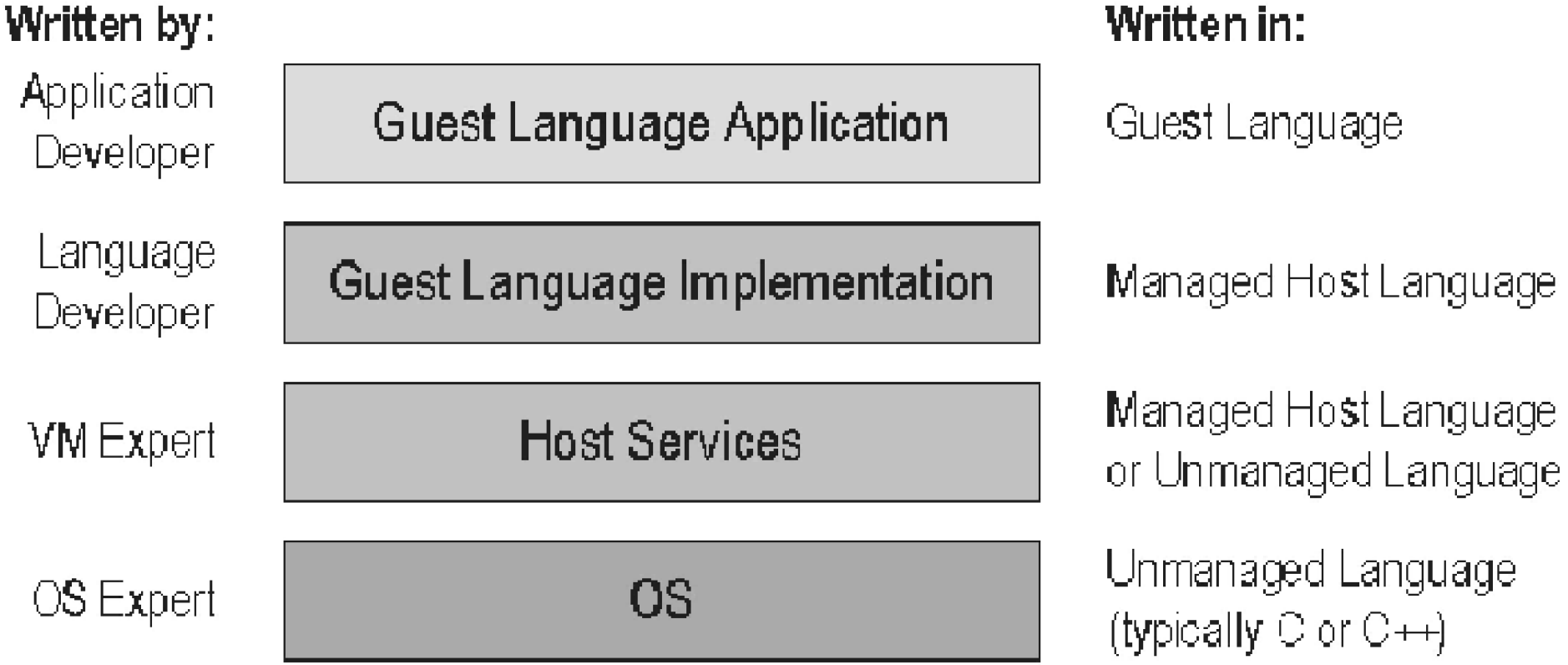


Java mai este la modă?

- Încă.... nu se știe cât 



Ce înseamnă de fapt programarea?



Holy Grail of compilers

Cursul nr. 2

Mihai Zaharia