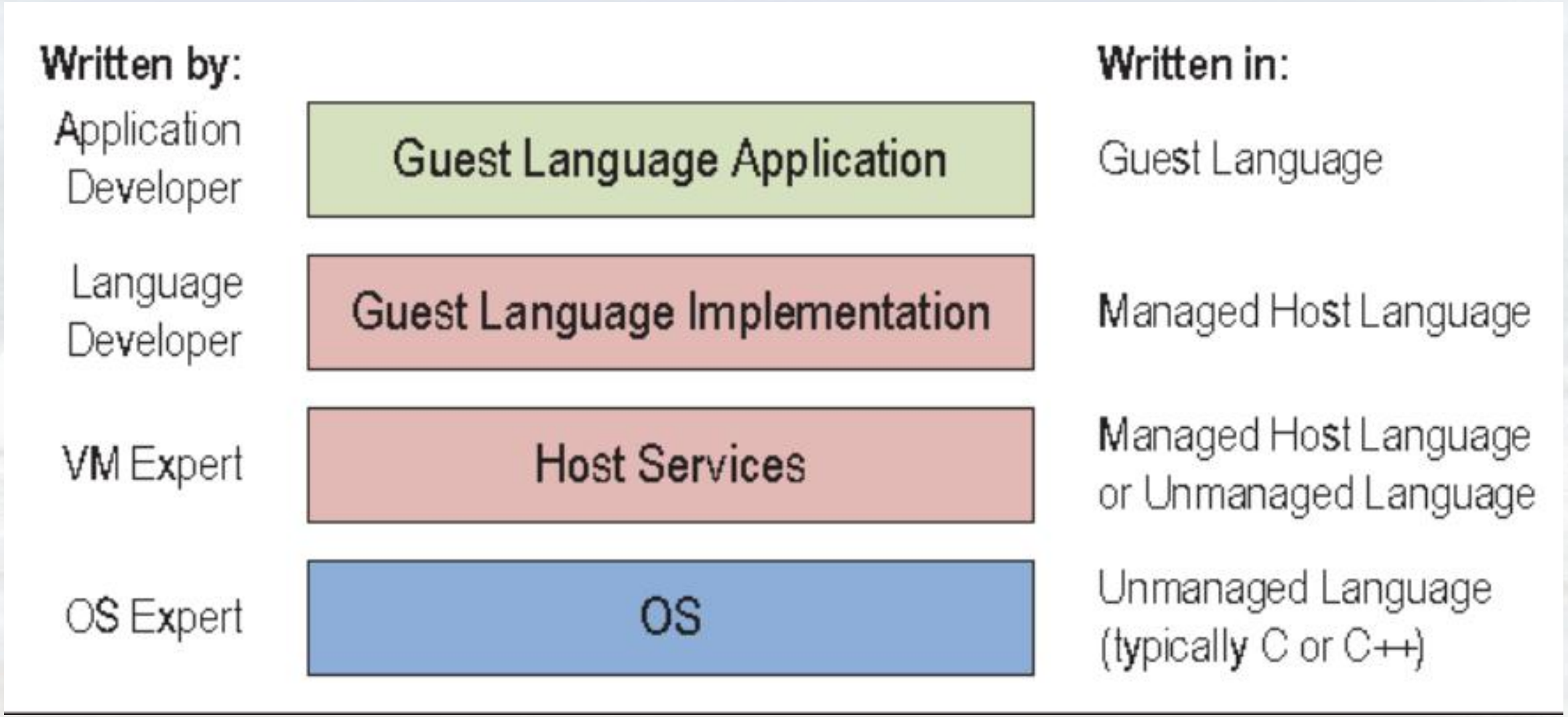
Holy Grail of compilers

Cursul nr. 2

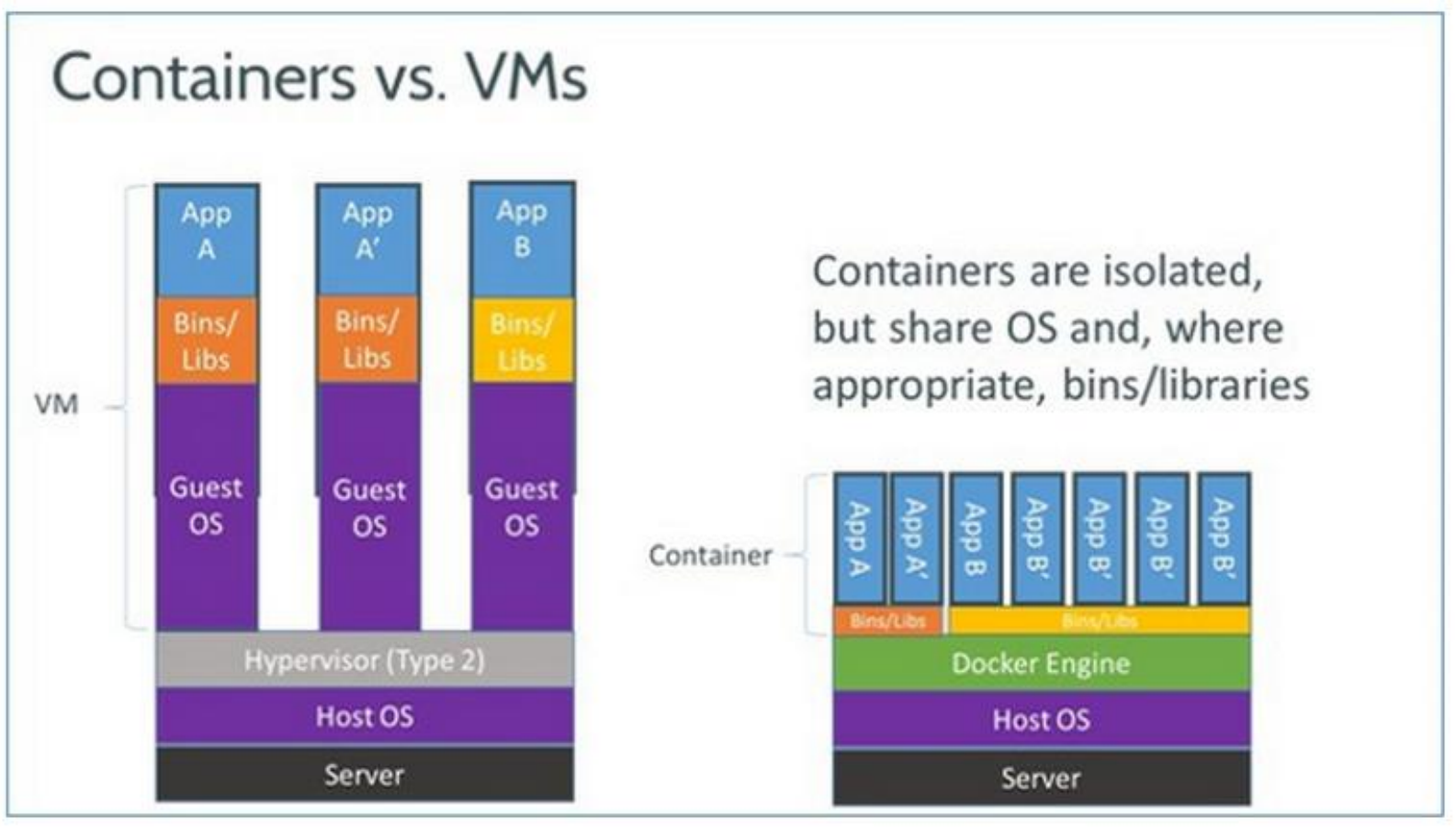
Mihai Zaharia

Ce înseamnă de fapt programarea?



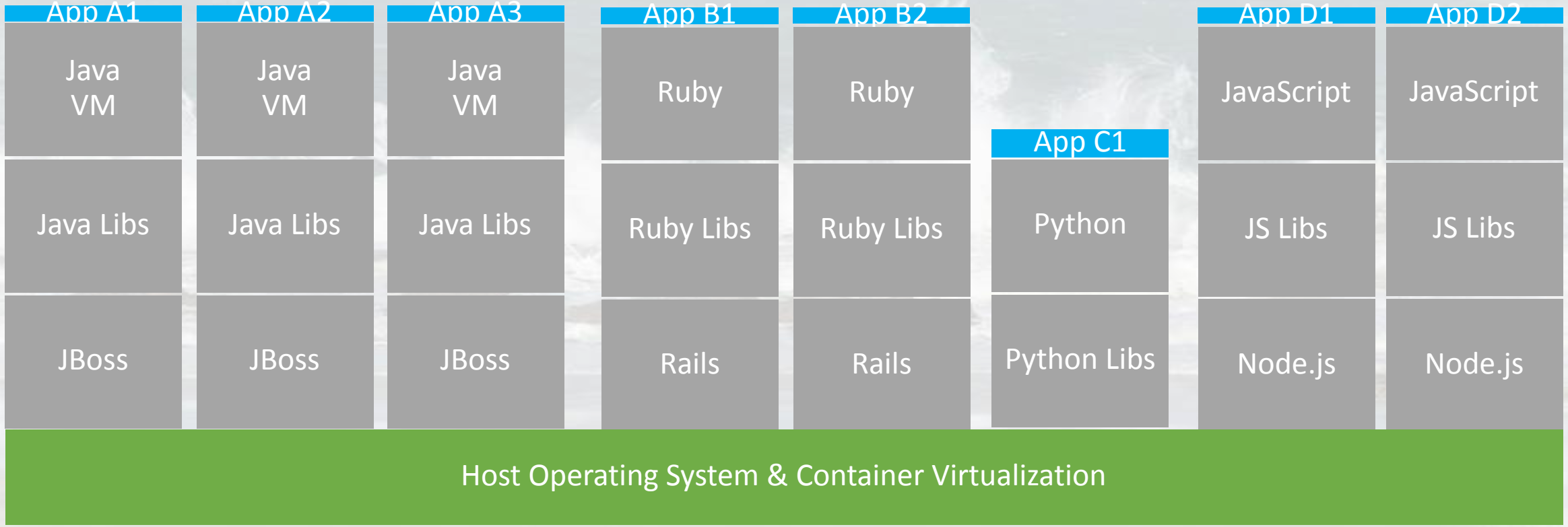
Java mai este la modă?

- Încă.... nu se știe cât



Aplicatiile din containere sunt și ele mari

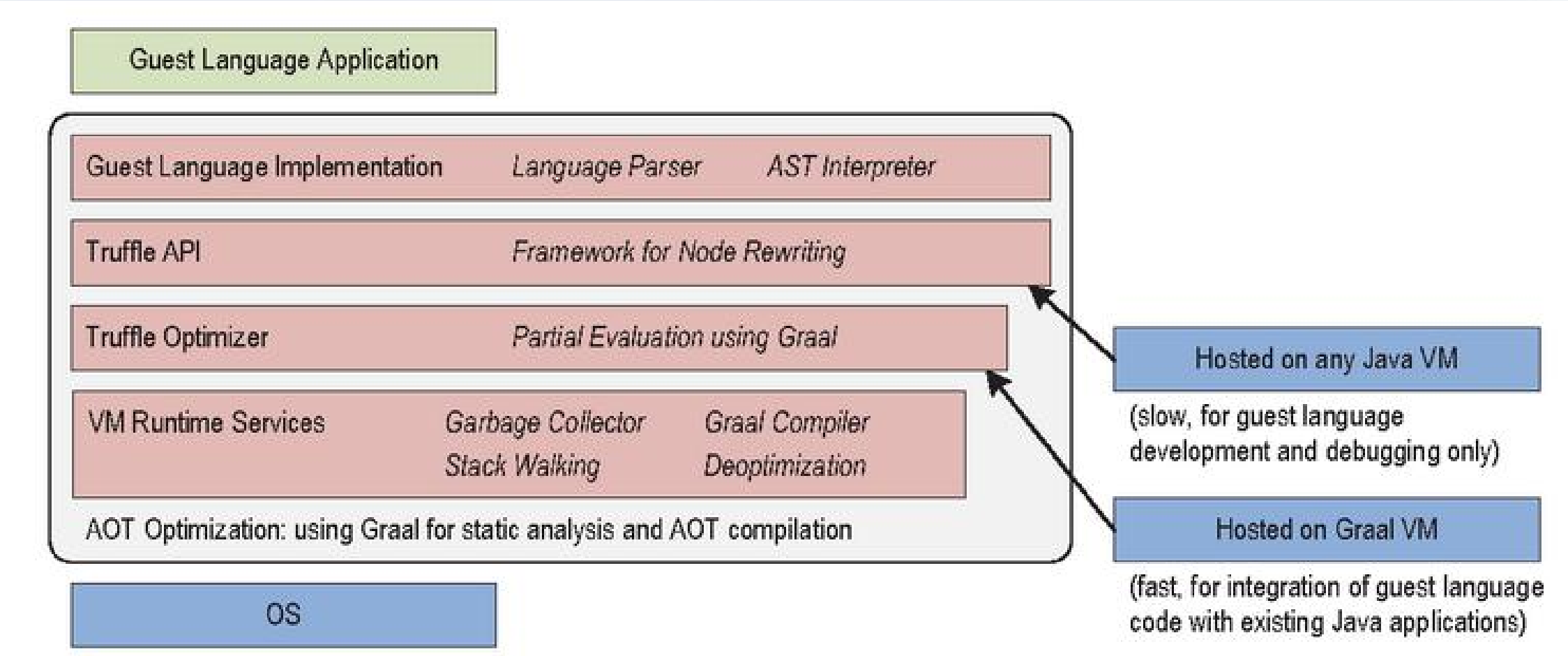
- “Hello World” în Java: 24 MB
- “Hello World” în JavaScript (V8): 18 MB
- “Hello World” în Ruby/Rails: 8 MB



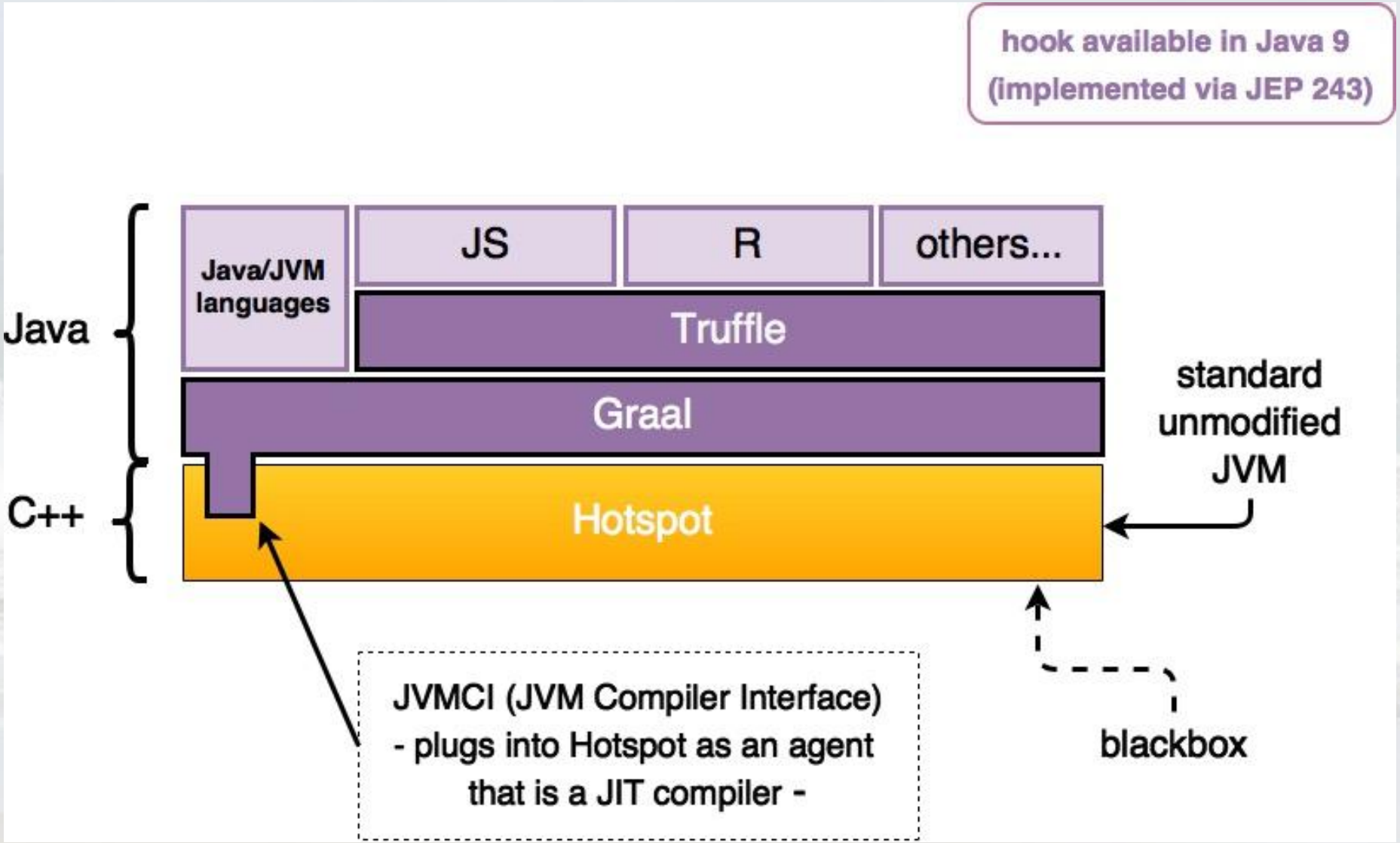
Cum s-a ajuns aici?

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

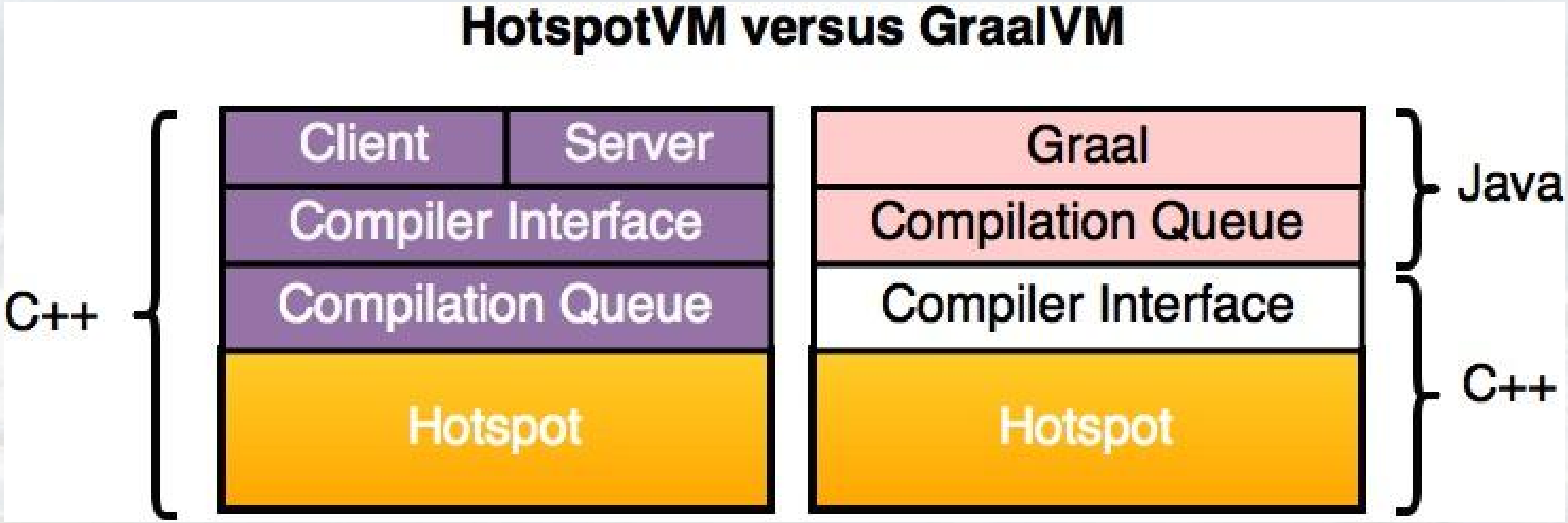
Arhitectura originală/inițială



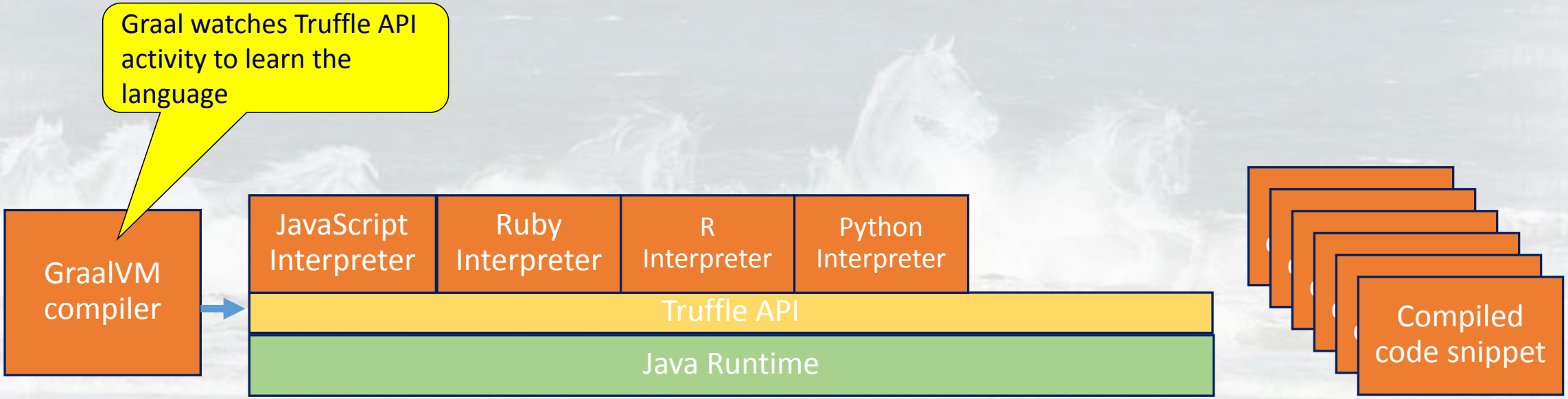
Structura internă



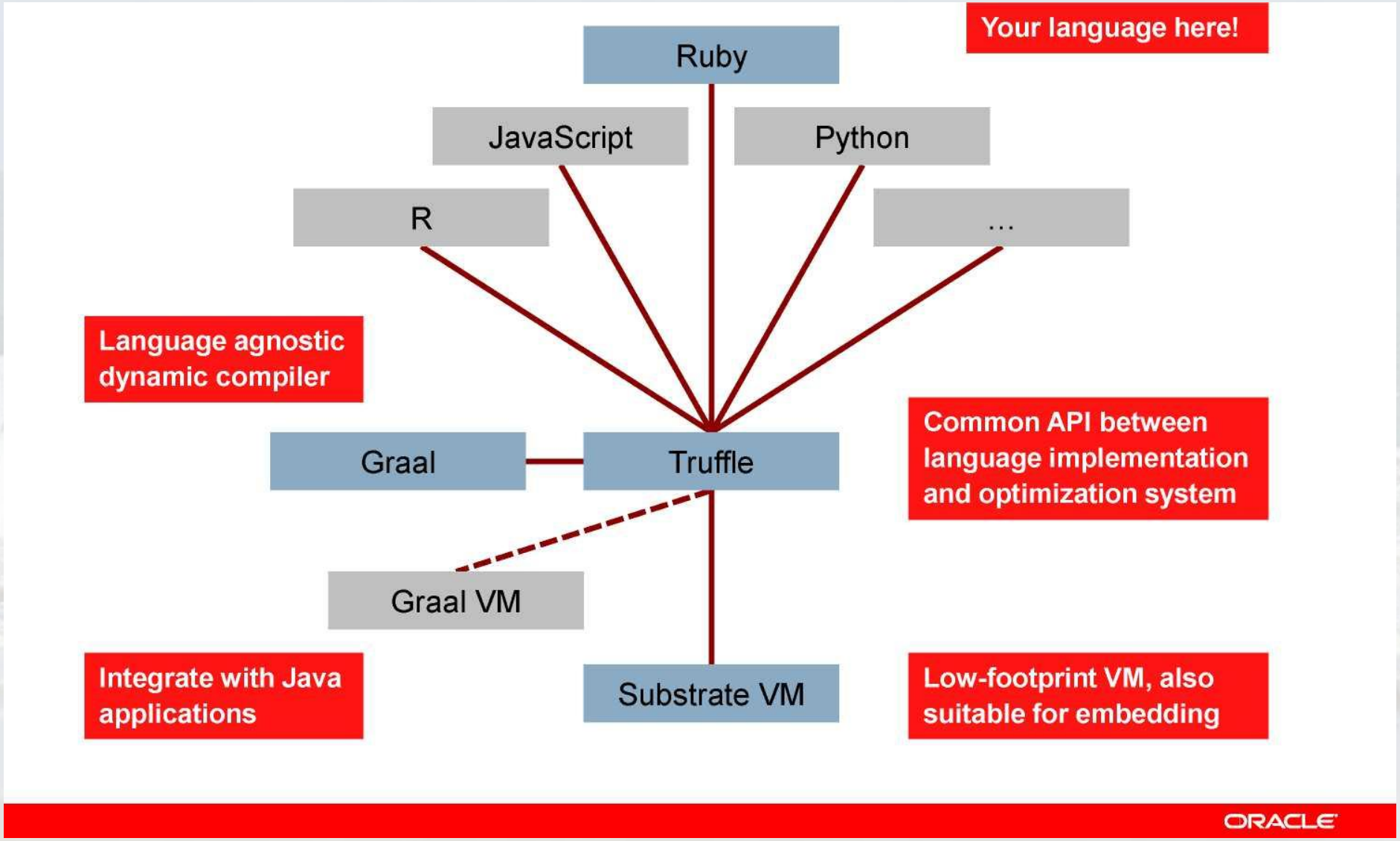
HotSpot


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

GraalVM este o virtualizare a Polyglot (Multilingual)

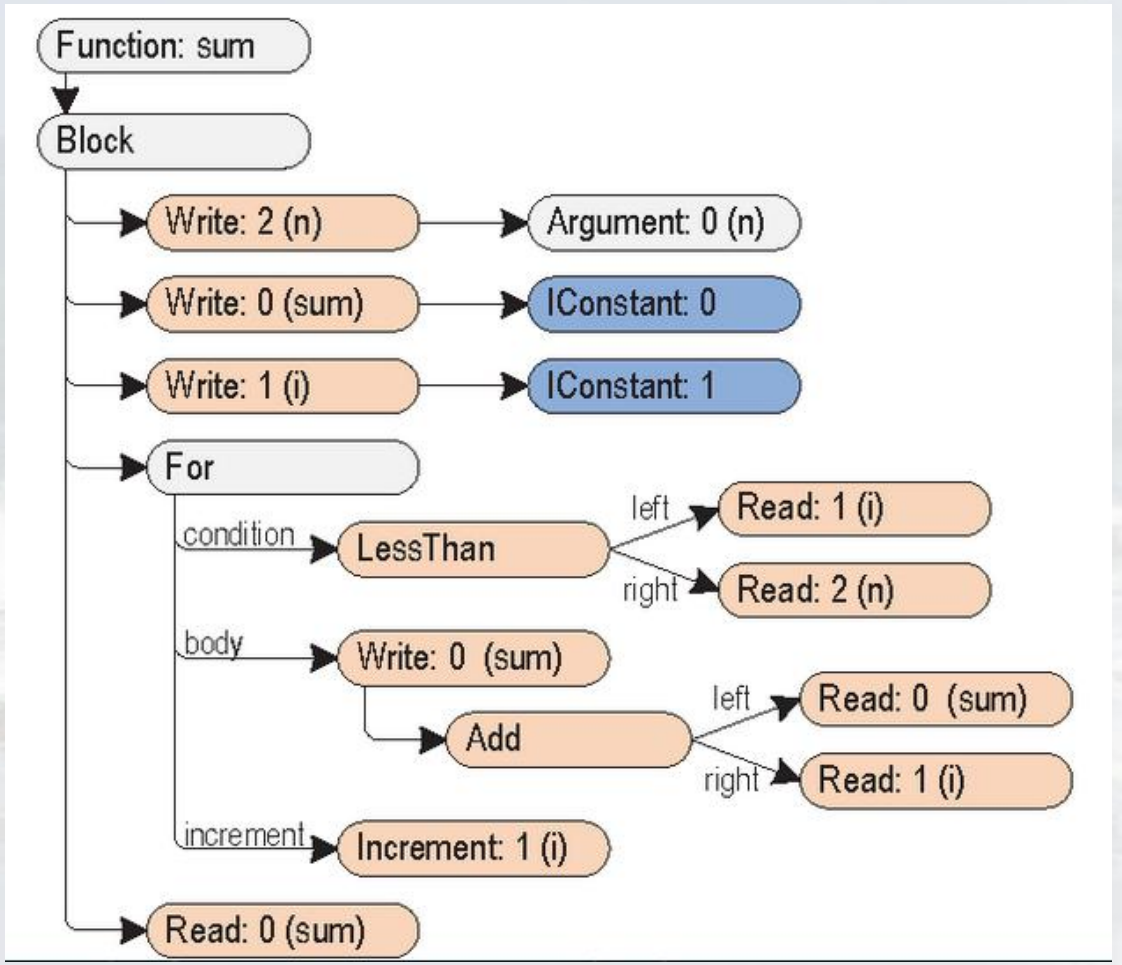


Limbaje suportate

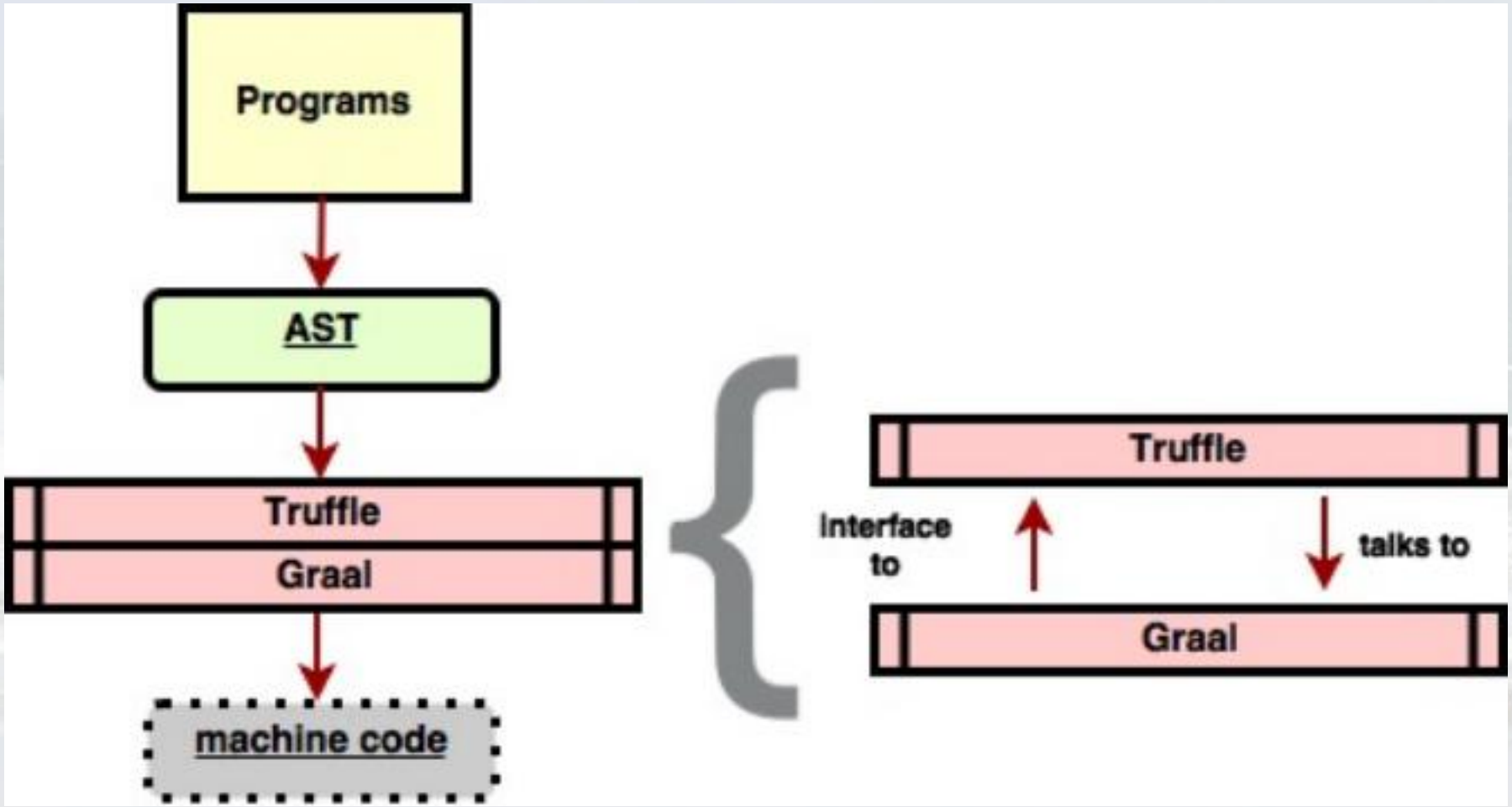


Ce este un AST-Abstract Syntax Tree

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



Maniera de tratare a unui program



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

compilatoare și translatoare

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

Expresii Lambda

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

```
λx → 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 = λx → (λy → x+y)
```

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
```

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:

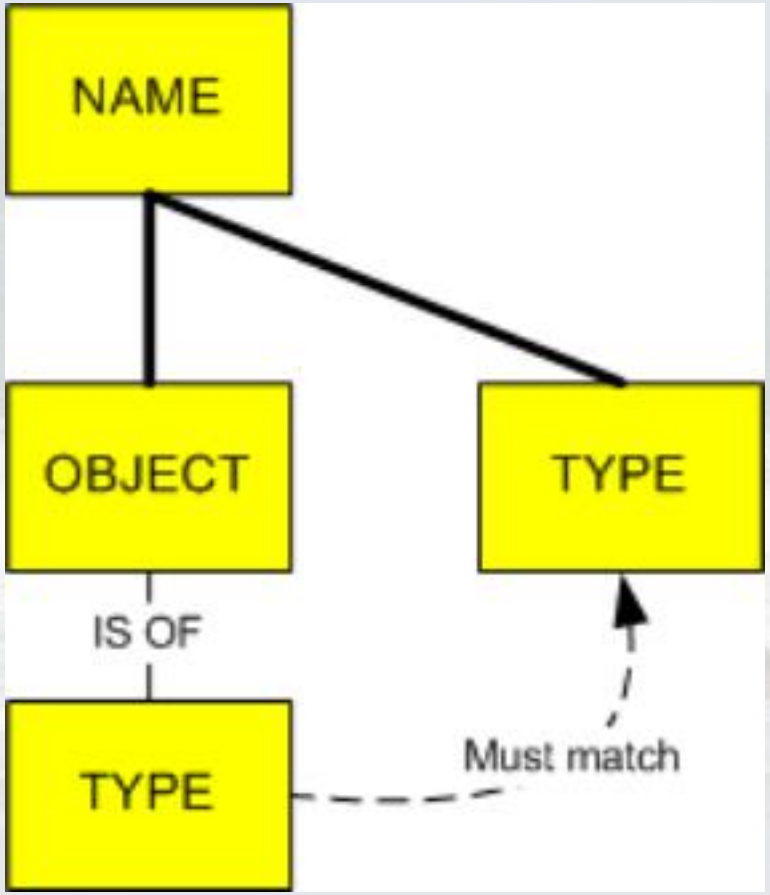
- Utilizarea săgeții → care face asociere la dreapta

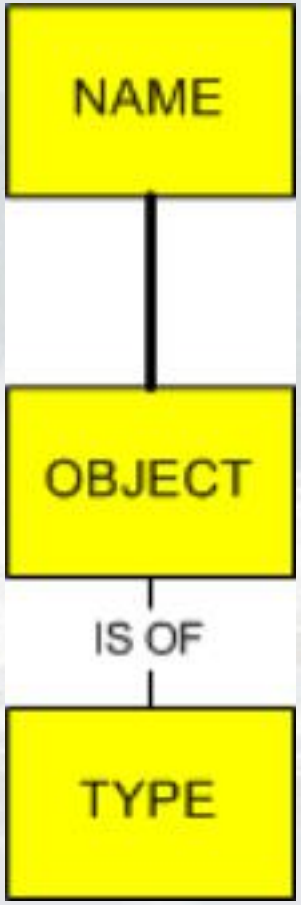
```
Int → Int → Int →
Int
```

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

```
mult x y z
```

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





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!)

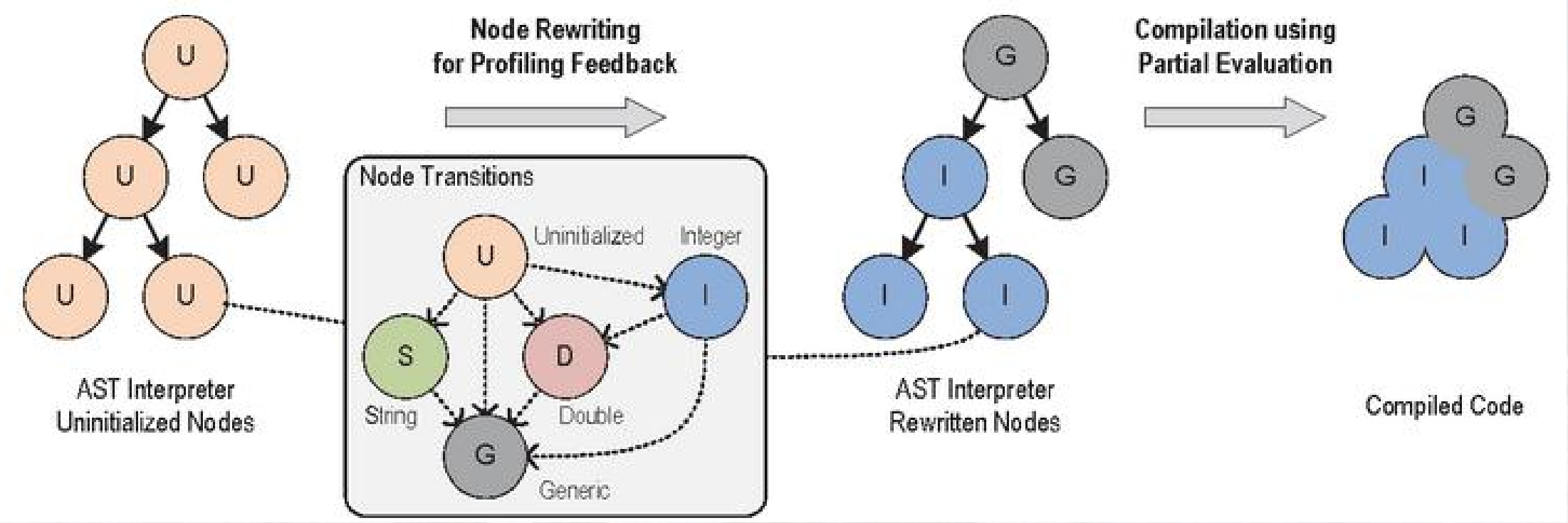
employeeName = "Steve Ferg"

exemplu

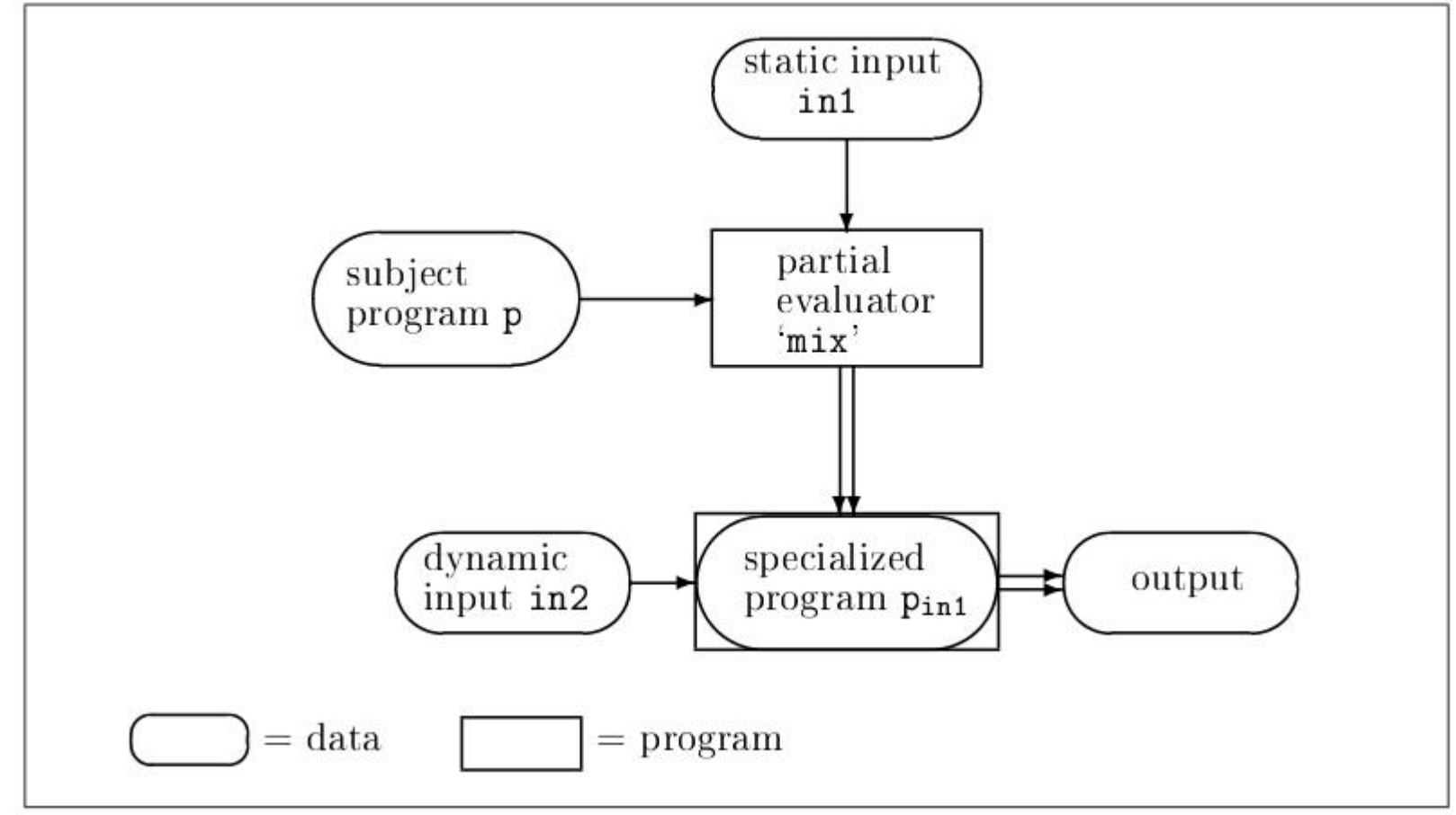
```
//Limbaj cu tipuri slabe
a  = 9
b = "9"
c = concatenate(a, b)  // rezultat "99"
d = add(a, b)          // rezultat 18
```

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

Reorganizarea nodurilor în arborele de evaluare



evaluator partial



Specializarea unui program de calcul x^n

A two-input program $p =$

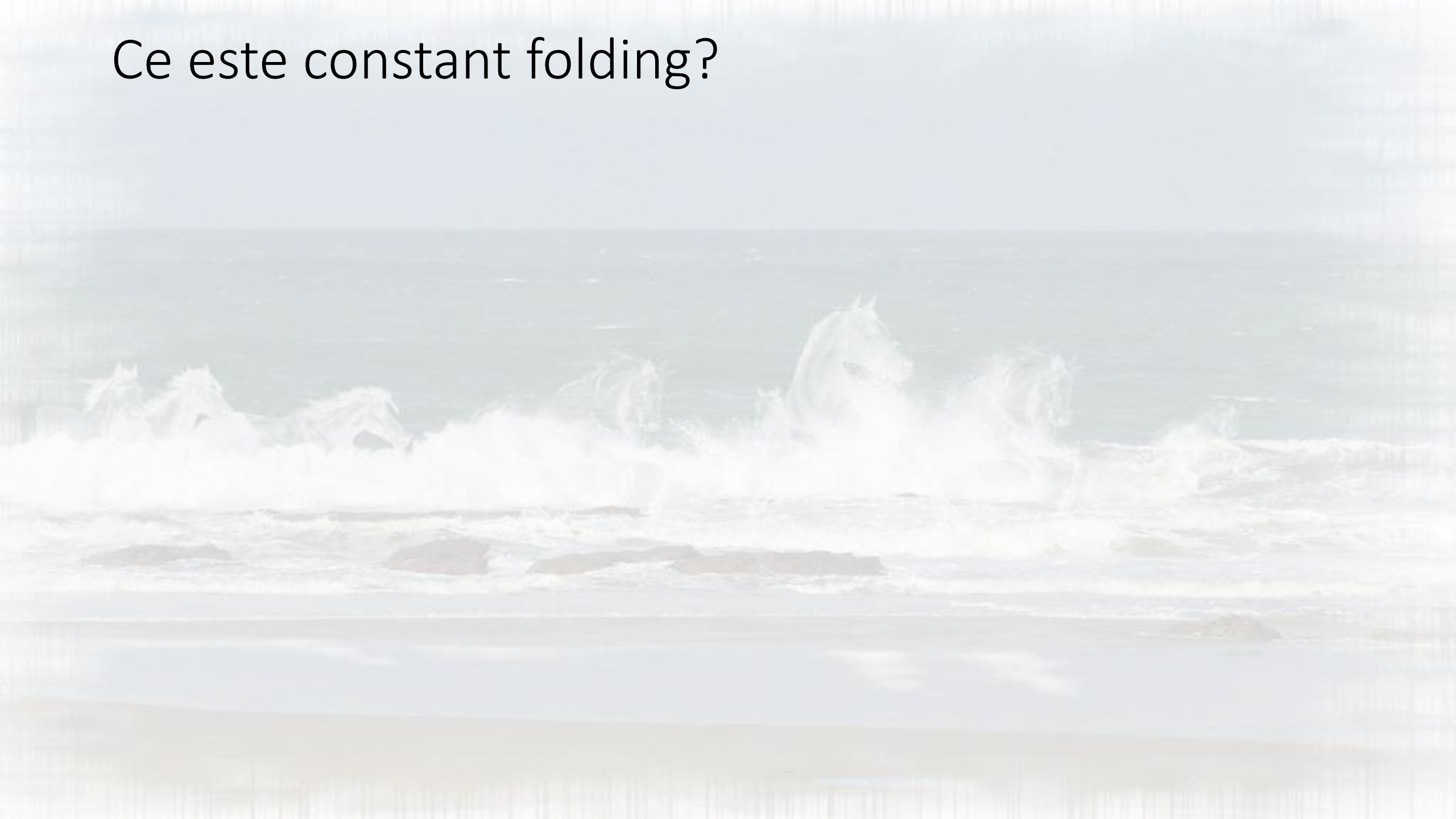
```
f(n,x) =if n = 0 then 1
        else if even(n) then f(n/2,x)↑2
        else x * f(n-1,x)
```

Program p, specialized to static input n = 5:

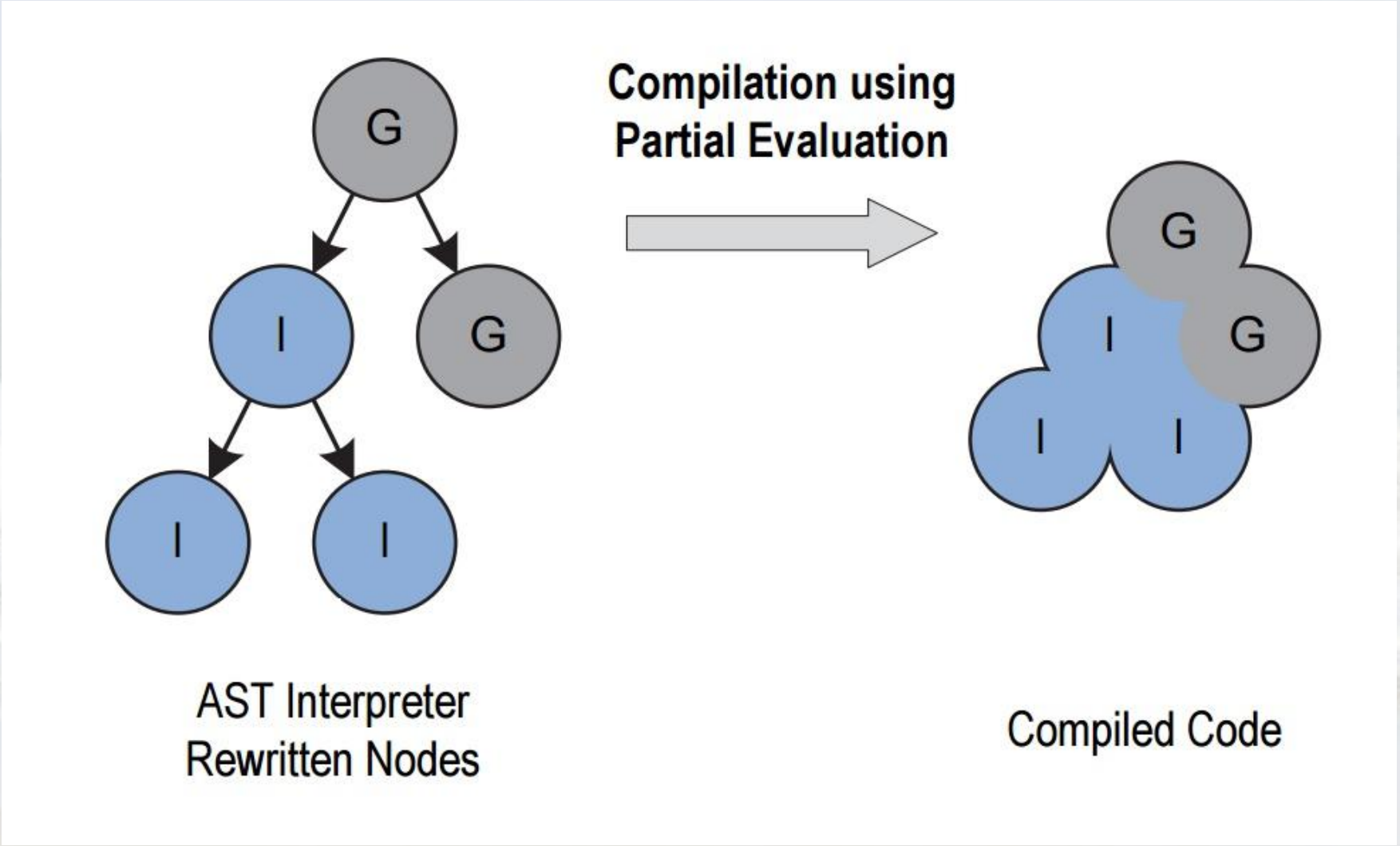
$p_5 =$

```
f5(x) = x * ((x↑2)↑2)
```

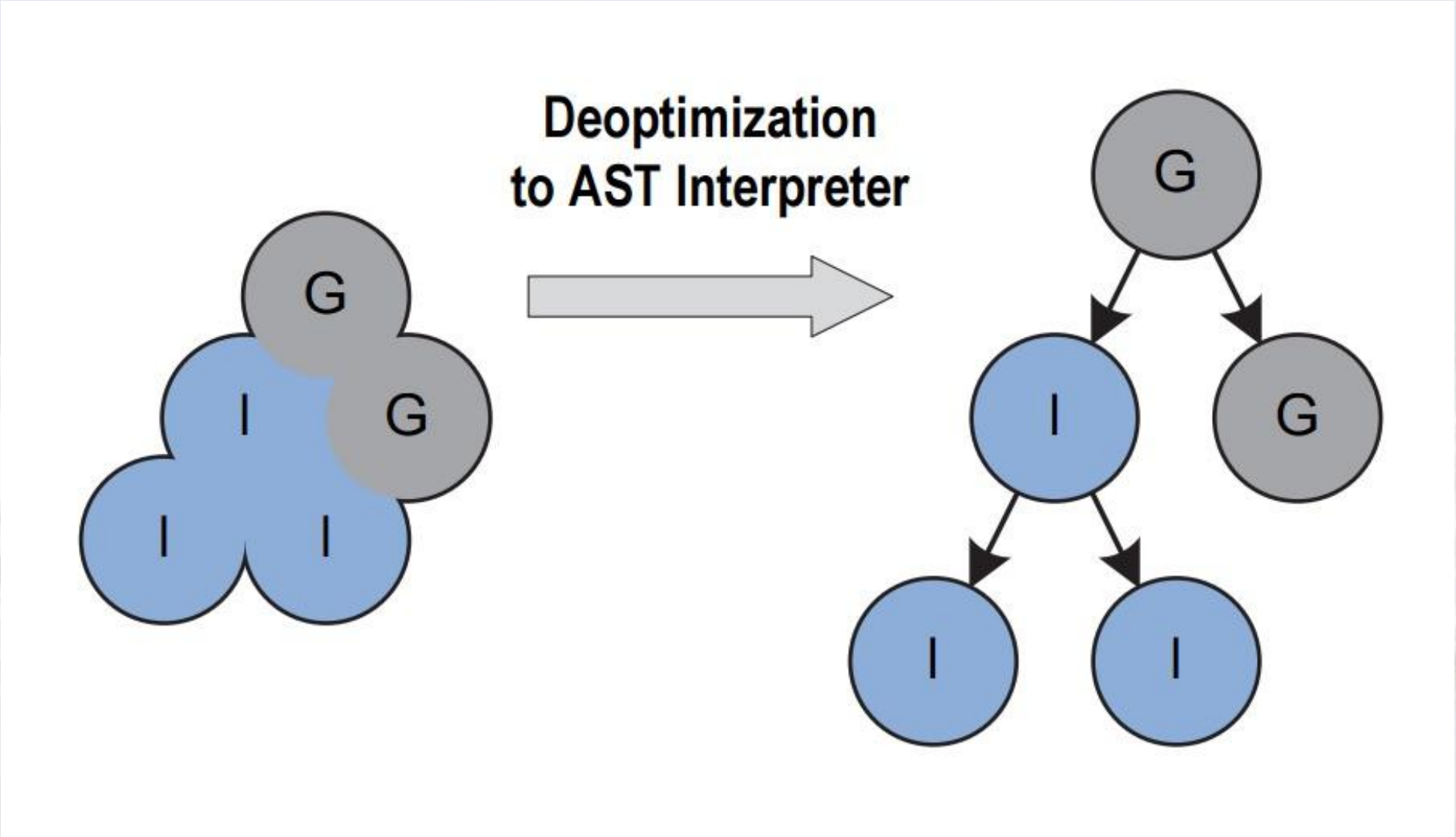
Ce este constant folding?



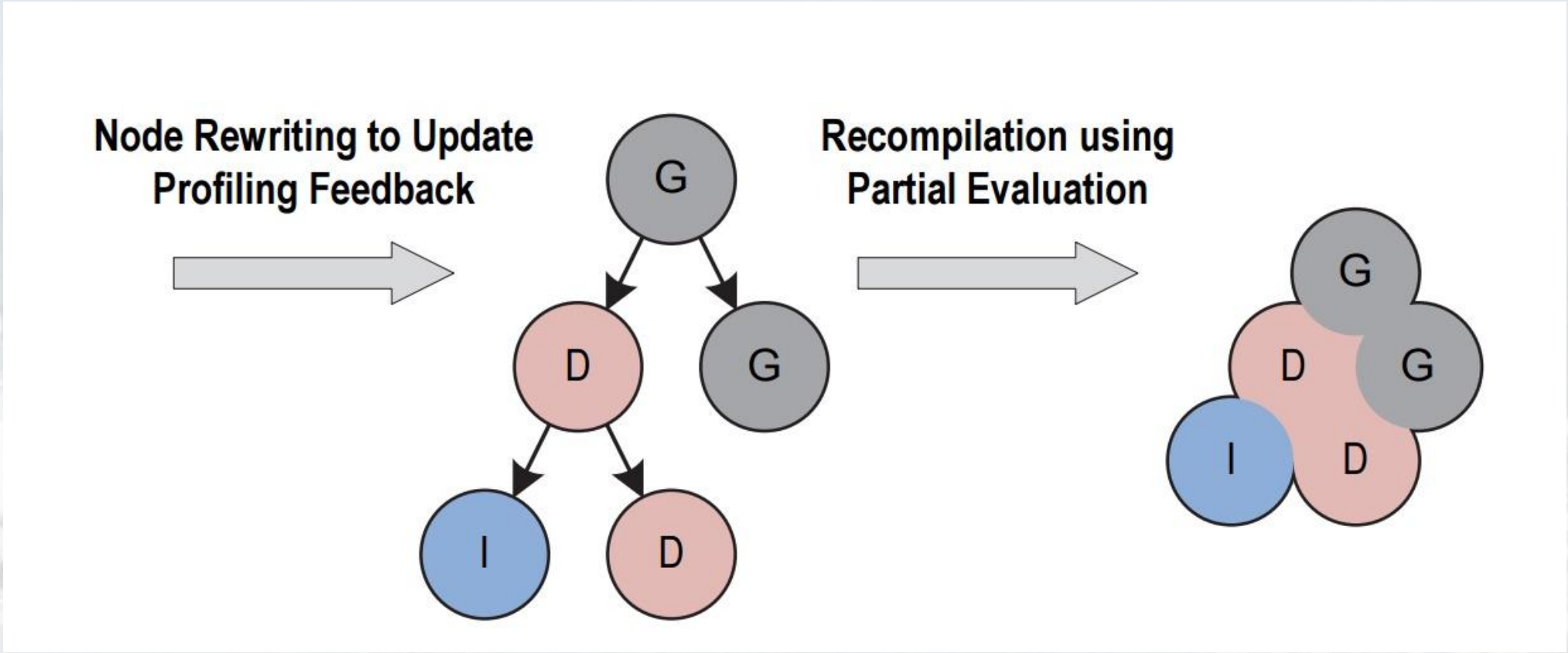
still..... folding



eliminarea optimizarilor - de-optimizarea (De-optimisation)



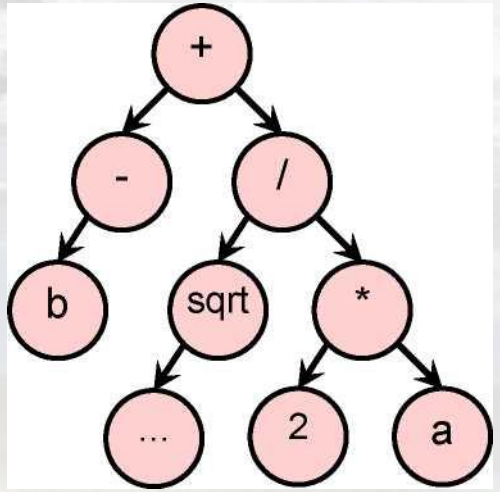
deoptimizarea

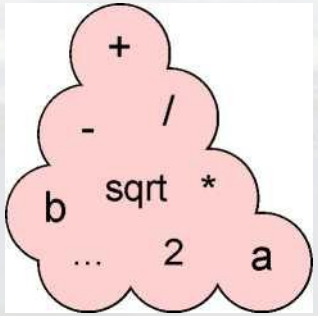


Optimizare performanțe cu Truffle
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

- 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 b
- 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

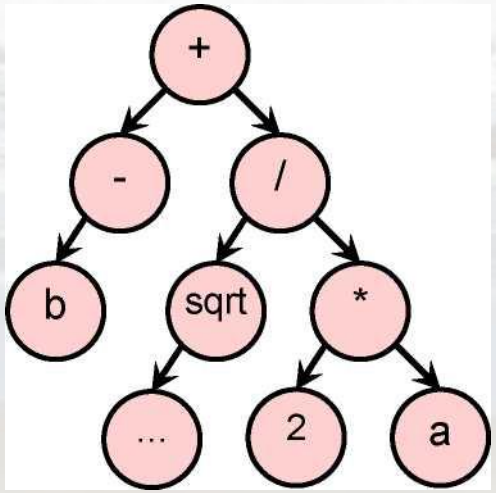
execute b
check that the negate method in Float has not changed
calculate negation execute b
check that the power method in Float has not changed
calculate power execute a
check that the multiply method in Float has not changed
calculate multiplication execute c
check that the multiply method in Float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



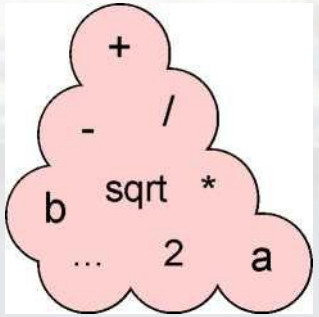


Optimizare performanțe cu Graal
-b + (Math.sqrt(b**2 - 4*a*c)) / 2*a

- 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 division



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!');");
 }
}
```