

Holy Grail of compilers

Cursul nr. 2
Mihai Zaharia

Ce înseamnă de fapt programarea?

Written by:

Application
Developer

Guest Language Application

Written in:

Guest Language

Language
Developer

Guest Language Implementation

Managed Host Language

VM Expert

Host Services

Managed Host Language
or Unmanaged Language

OS Expert

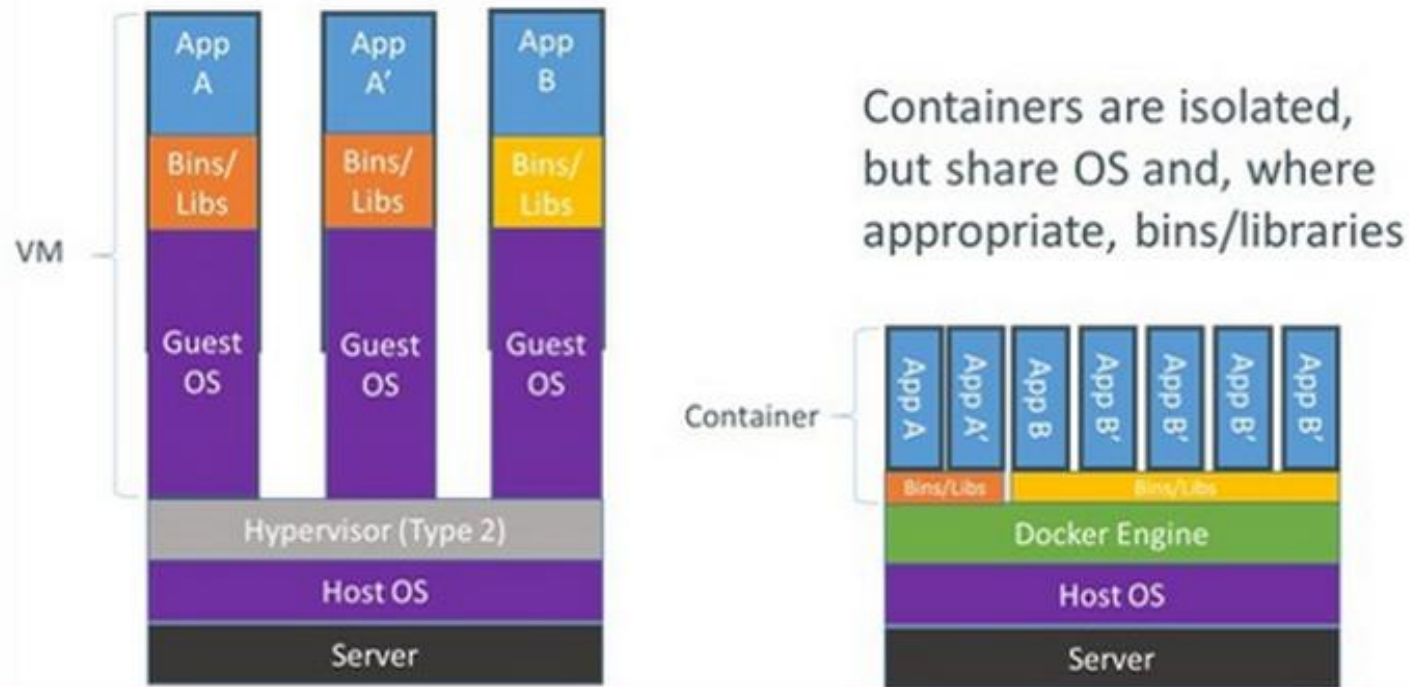
OS

Unmanaged Language
(typically C or C++)

Java mai este la modă?

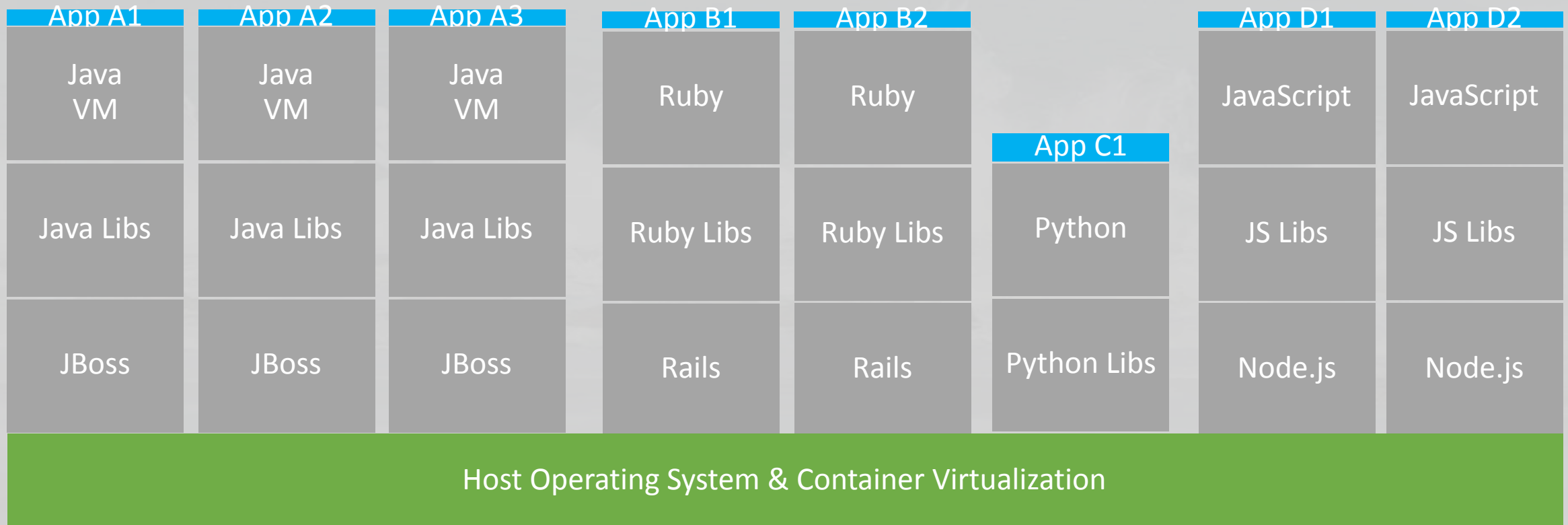
- Încă.... nu se știe cât

Containers vs. VMs

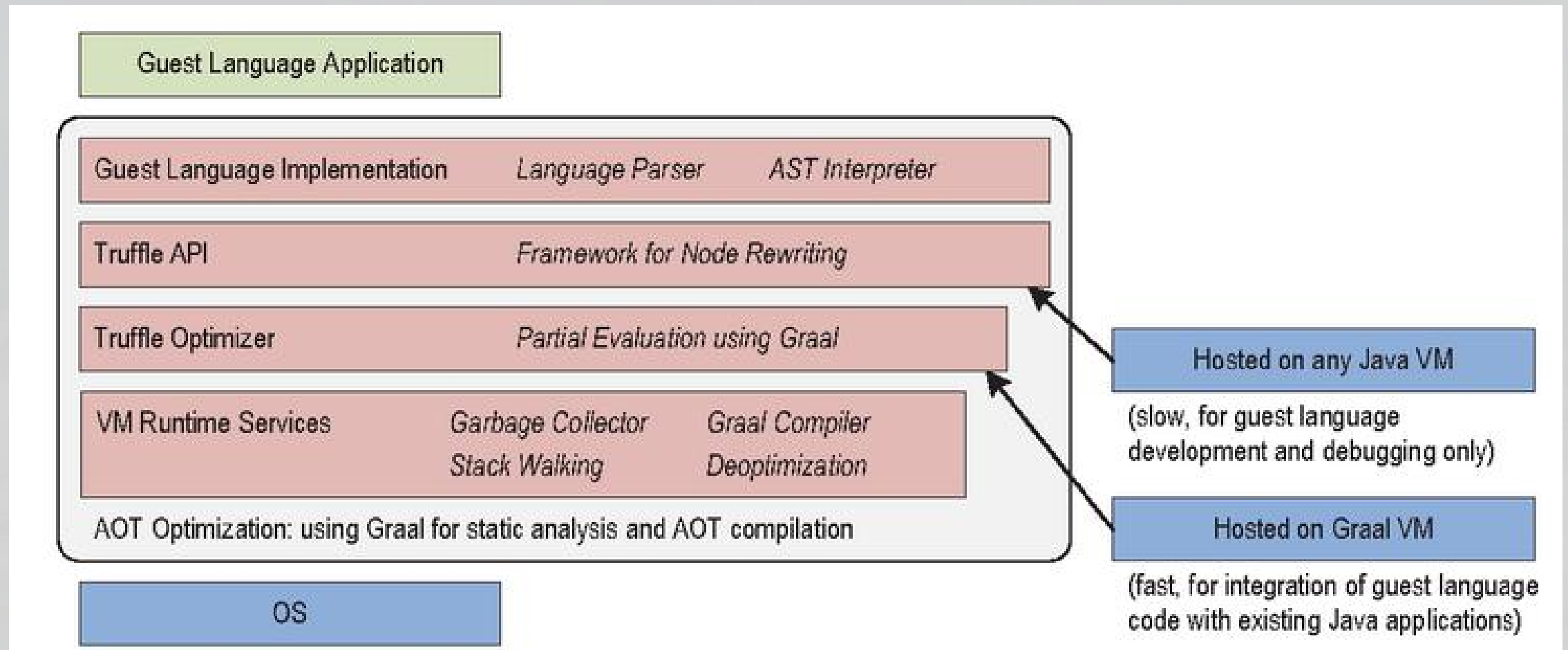


Aplicatiile din containere sunt și ele mari

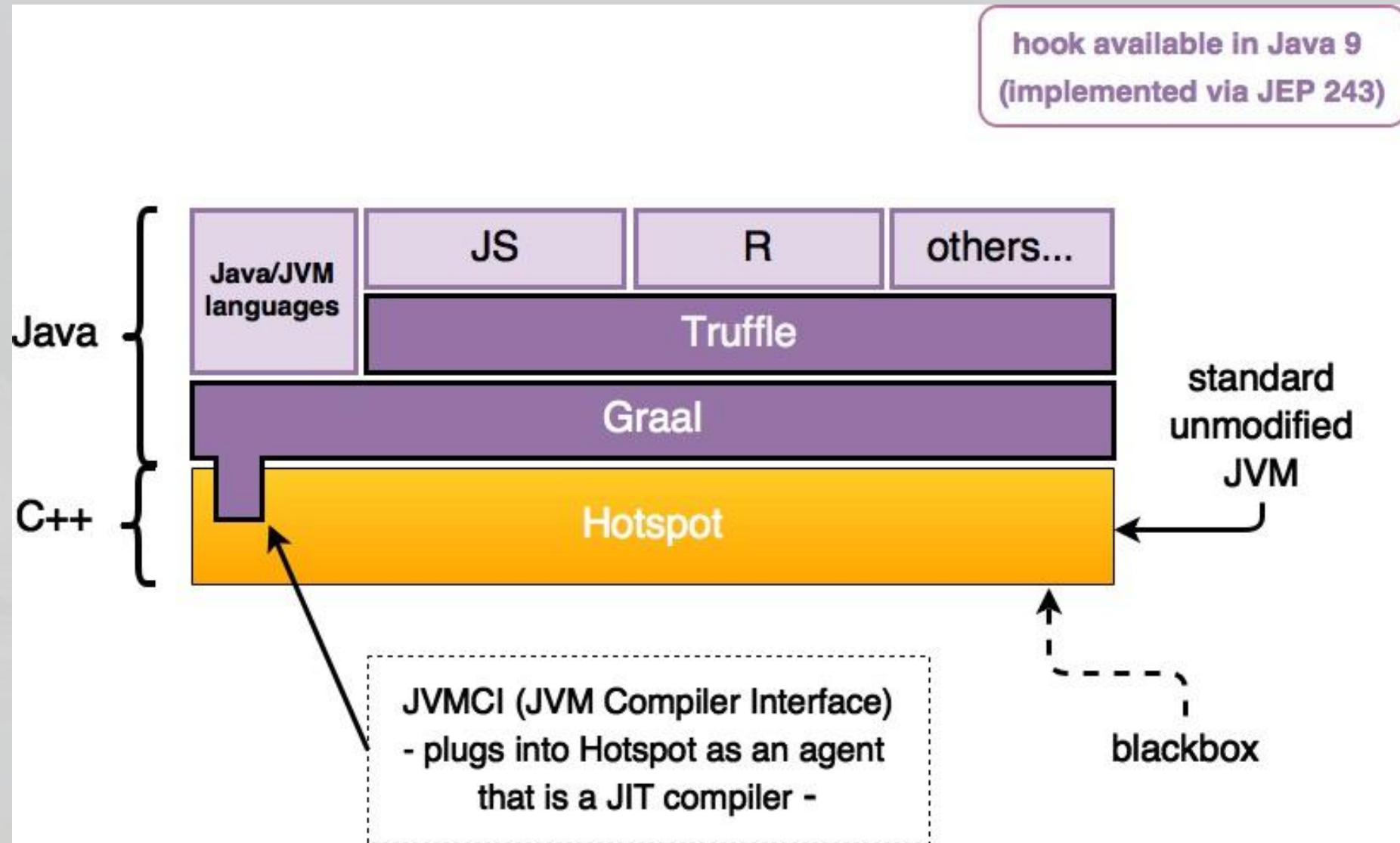
- “Hello World” în Java: 24 MB
- “Hello World” în JavaScript (V8): 18 MB
- “Hello World” în Ruby/Rails: 8 MB



Arhitectura originală/inițială

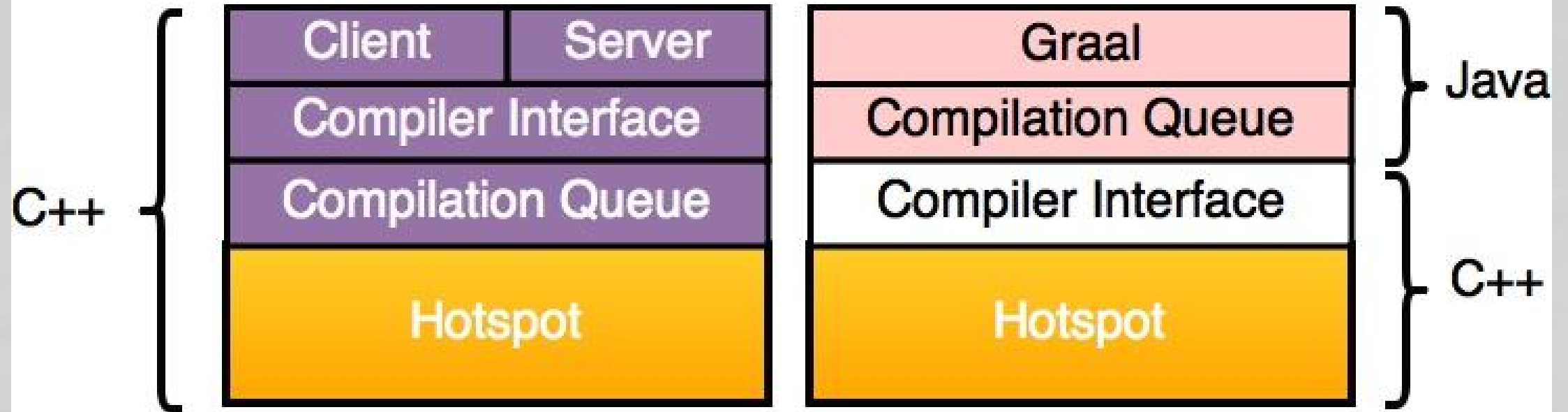


Structura internă

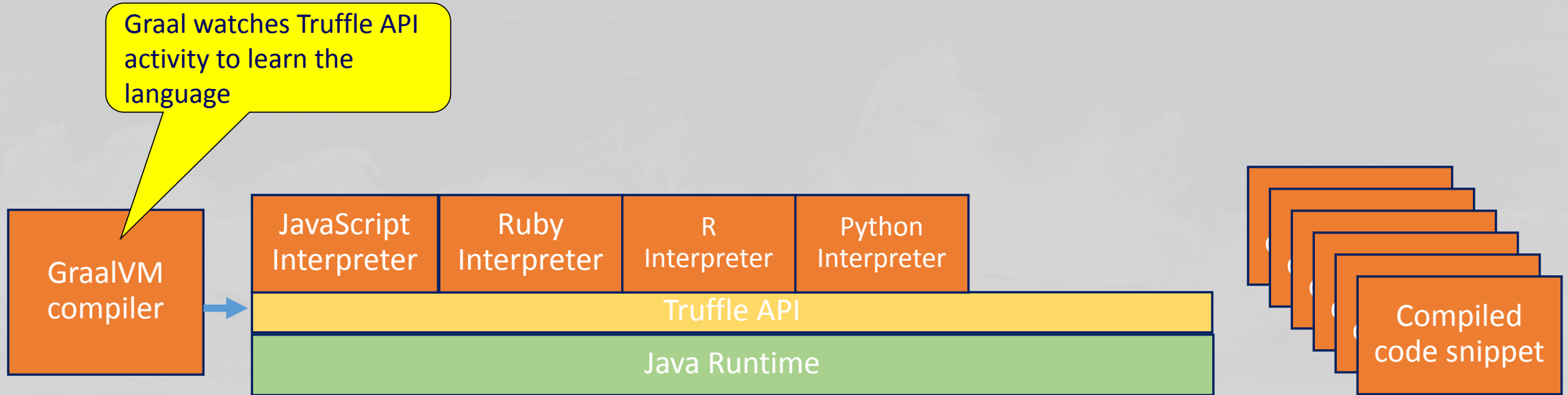


HotSpot

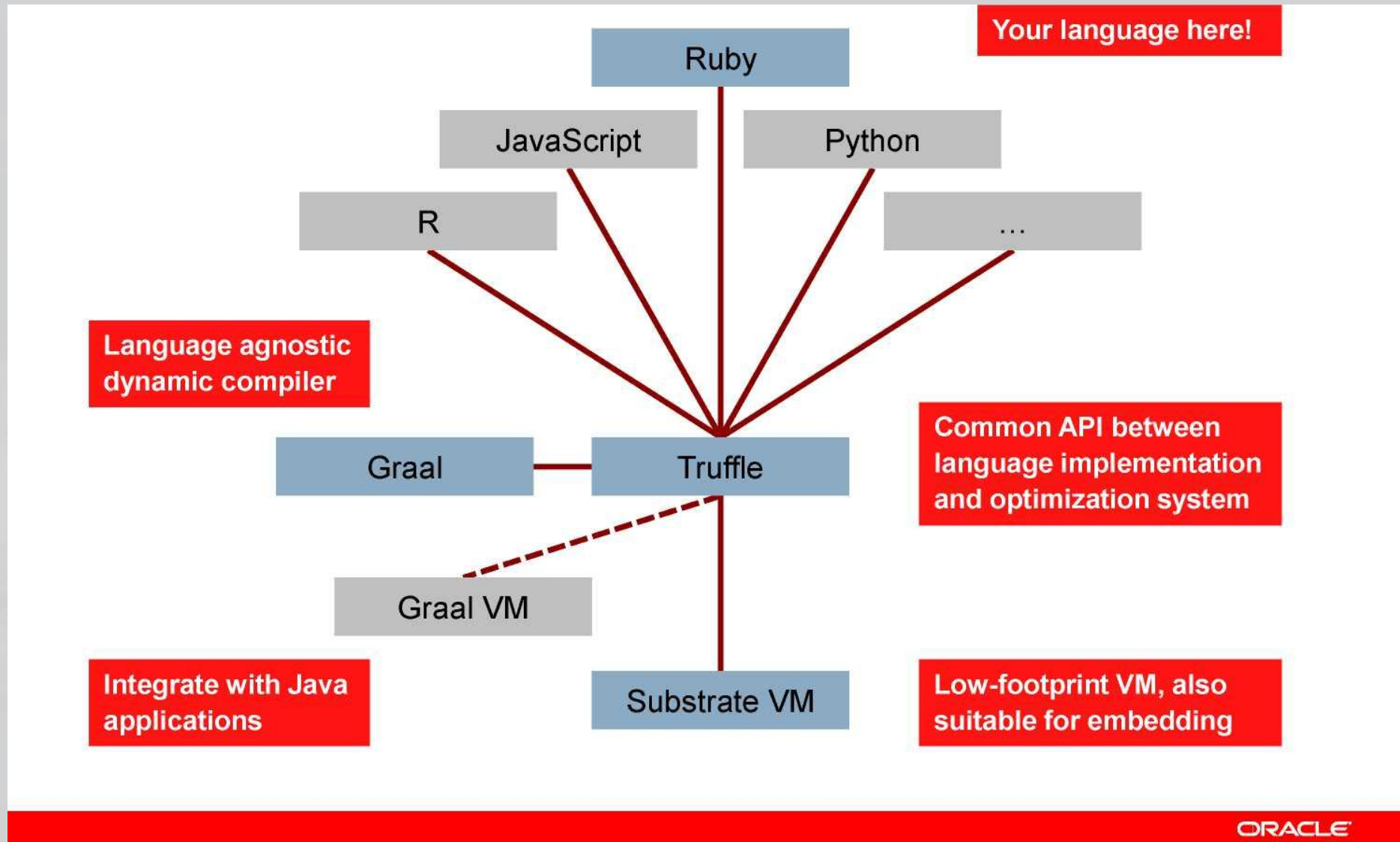
HotspotVM versus GraalVM



GraalVM este o virtualizare a Polyglot (Multilingual)

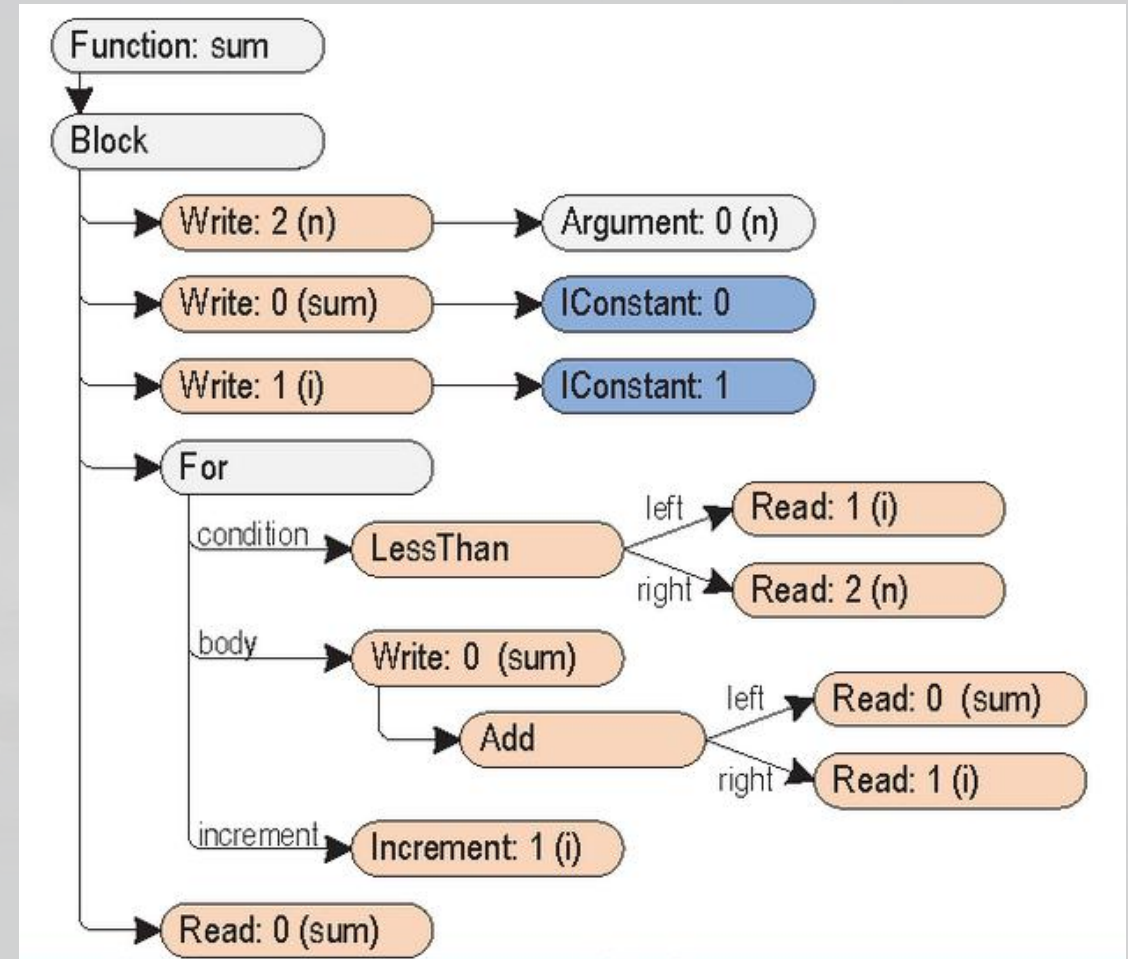


Limbaje suportate

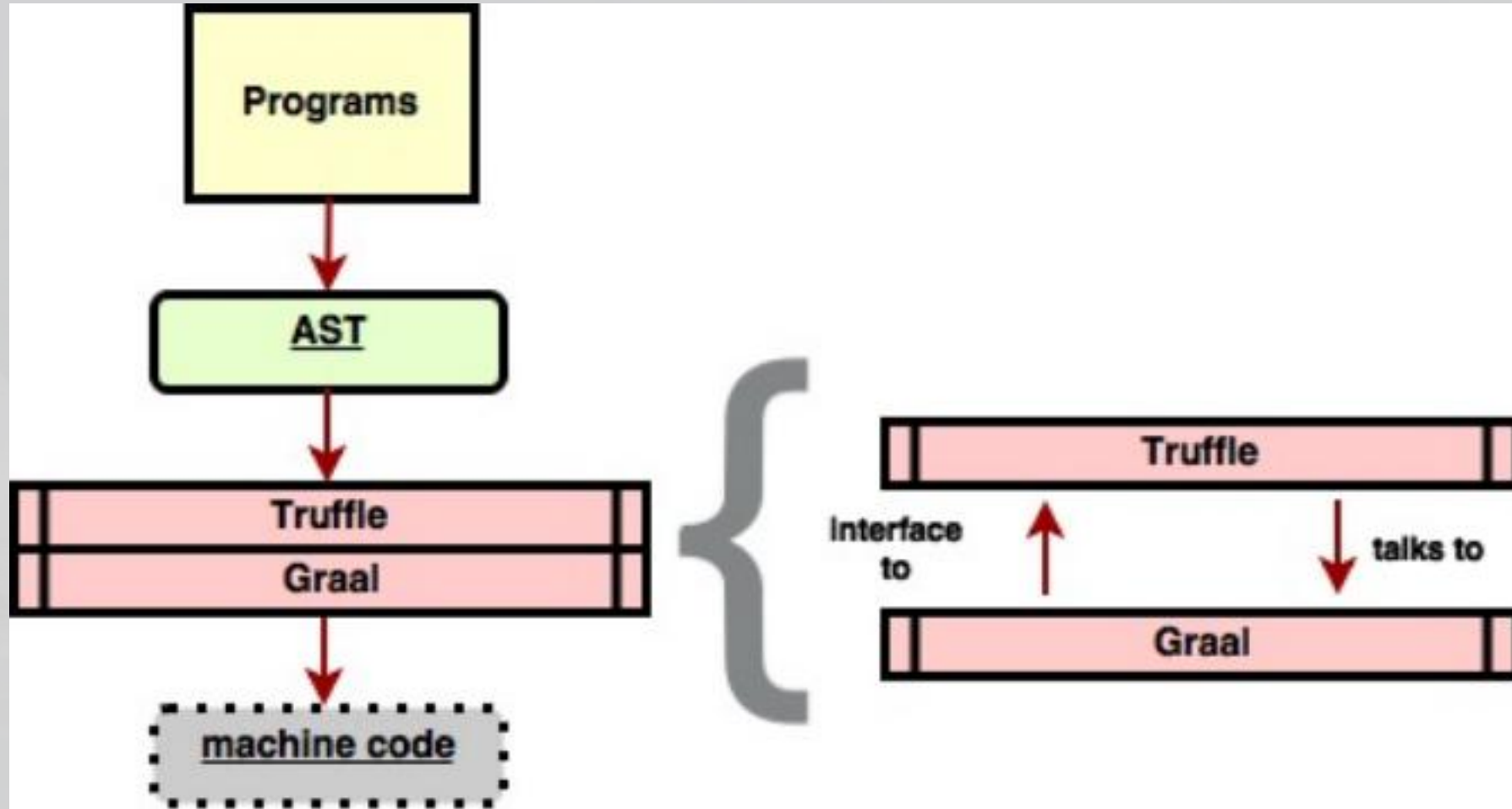


Ce este un AST-Abstract Syntax Tree

```
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 performanță 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:

$\lambda x \rightarrow x+x$

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 înțeles formal funcțiilor Curry.

De exemplu:

$\text{add } x \ y = x+y$

Înseamnă

$\text{add} = \lambda x \rightarrow (\lambda y \rightarrow x+y)$

Funcții Curry

Funcțiile cu argumente multiple sunt permise prin întoarcerea funcțiilor ca rezultat

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

add și add' produc același 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 funcții 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:

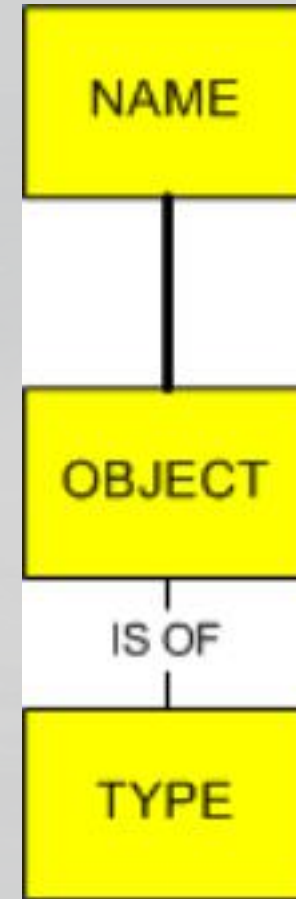
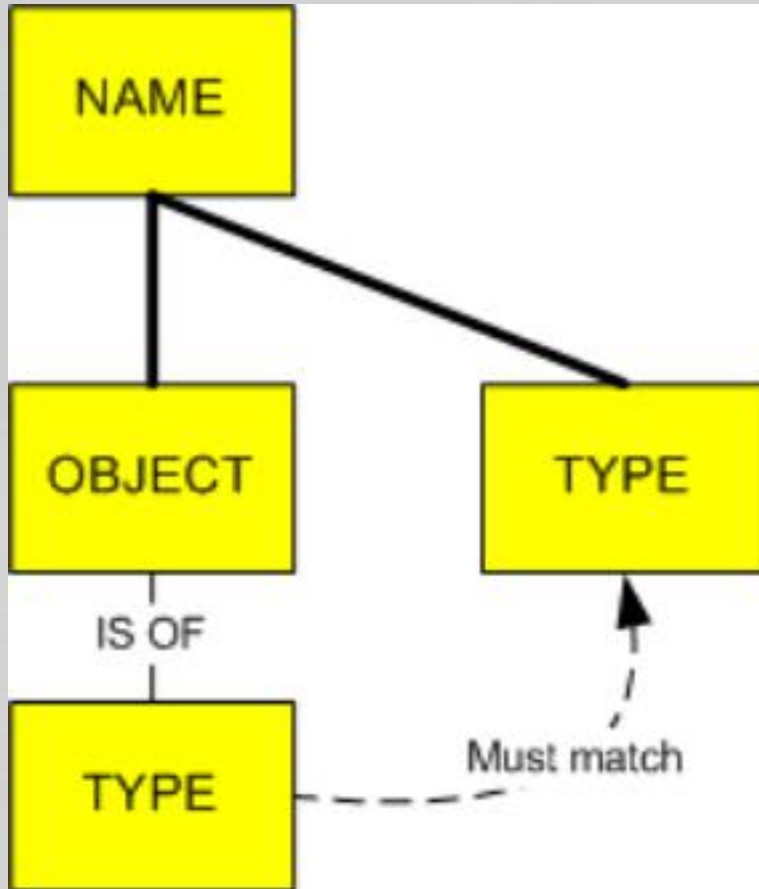
- Utilizarea săgeții → care face asociere la dreapta

```
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
```

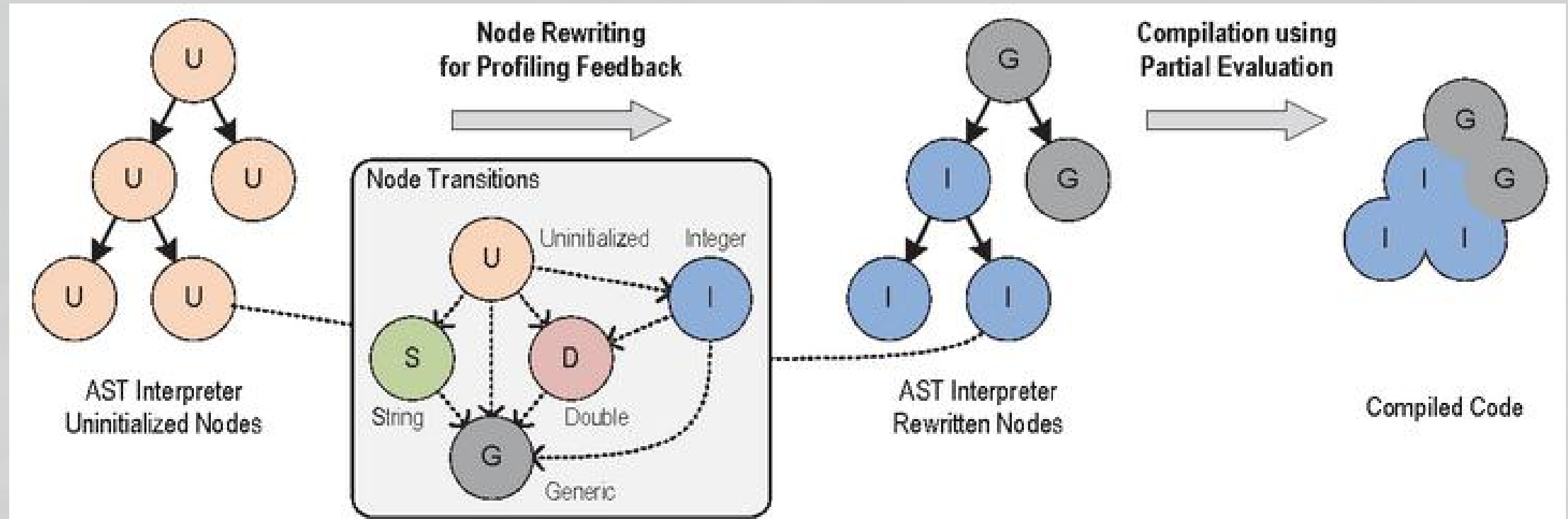
```
a = 9
```

```
b = "9"
```

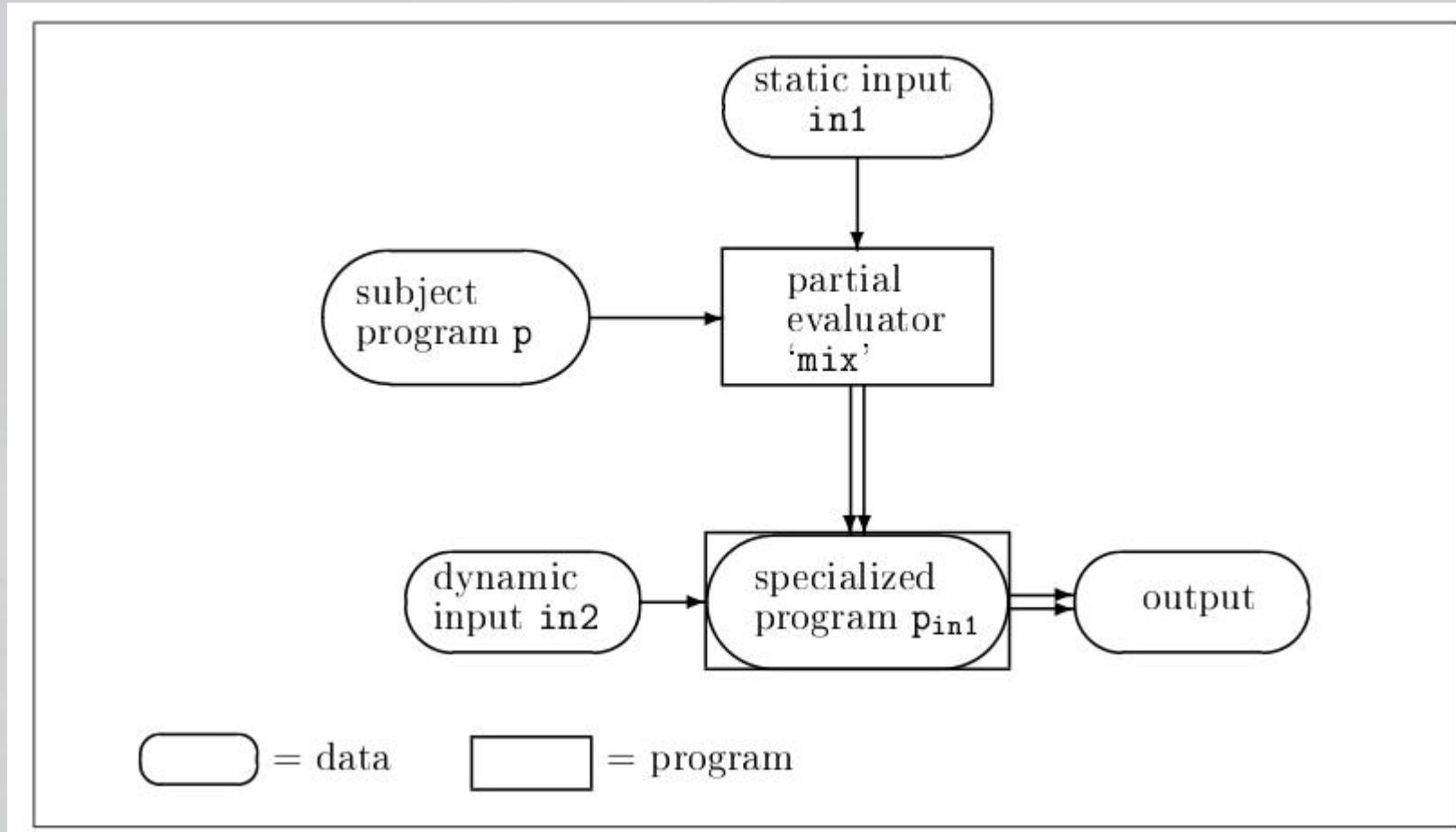
```
c = concatenate( str(a), b)//Grrr.....
```

```
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

$p =$

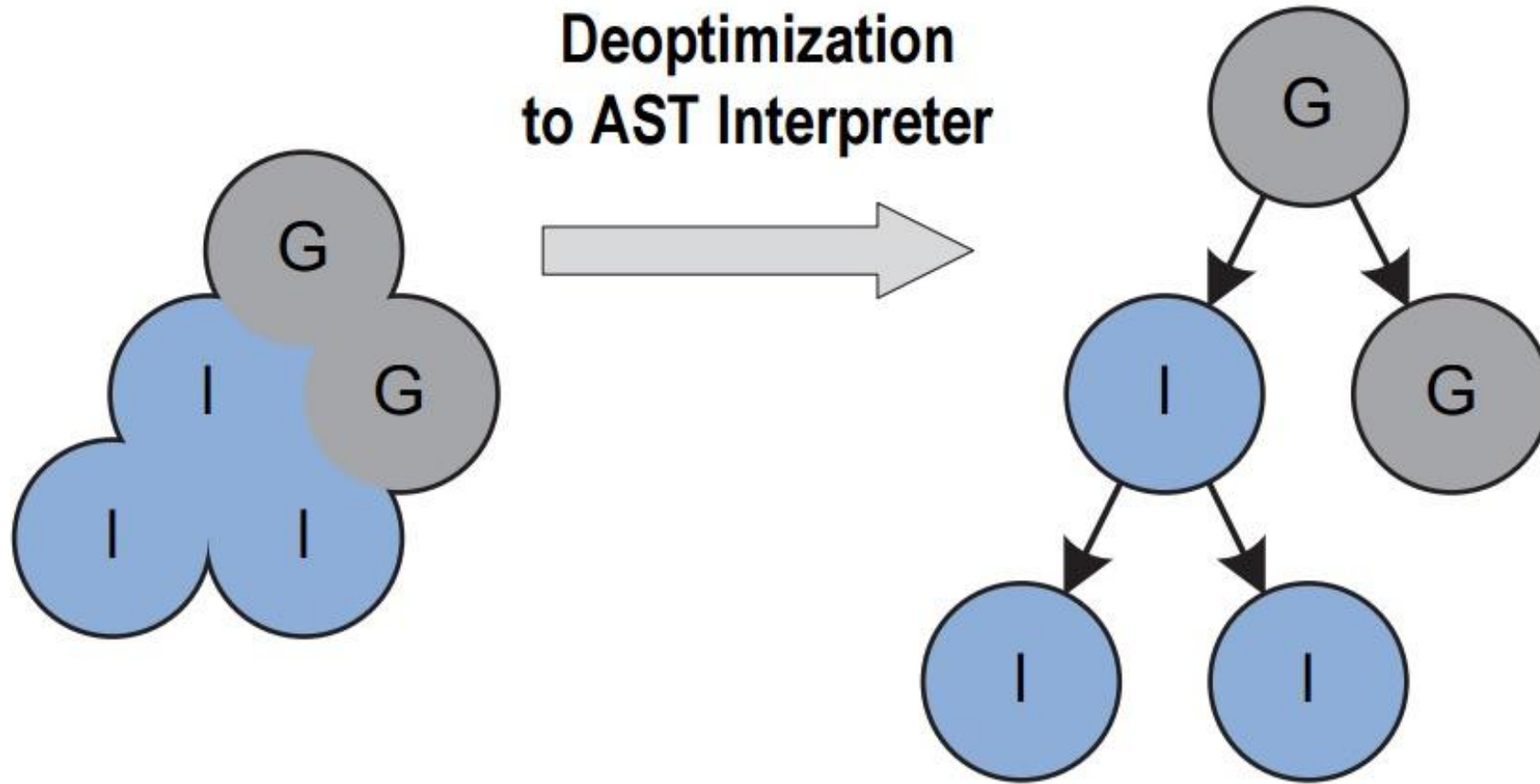
```
f(n,x) =if n = 0 then 1  
        else if even(n) then f(n/2,x)2  
        else x * f(n-1,x)
```

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

$p_5 =$

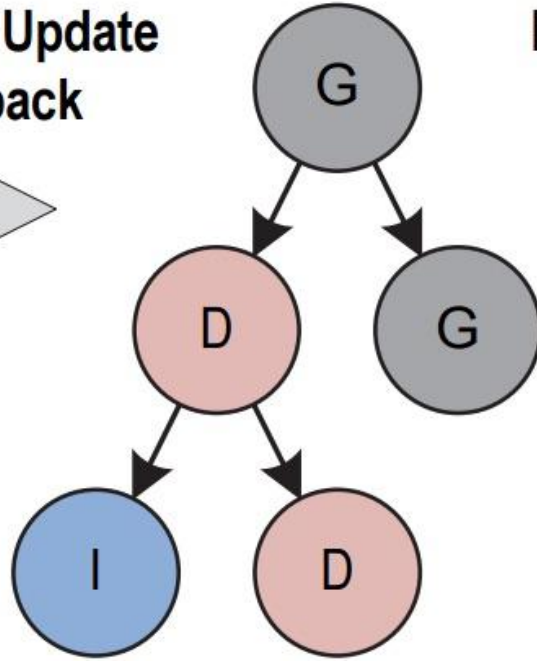
```
f5(x) = x * ((x2)2)
```


eliminarea optimizarilor - de-optimizarea (De-optimisation)



deoptimizarea

**Node Rewriting to Update
Profiling Feedback**



**Recompilation using
Partial Evaluation**

