

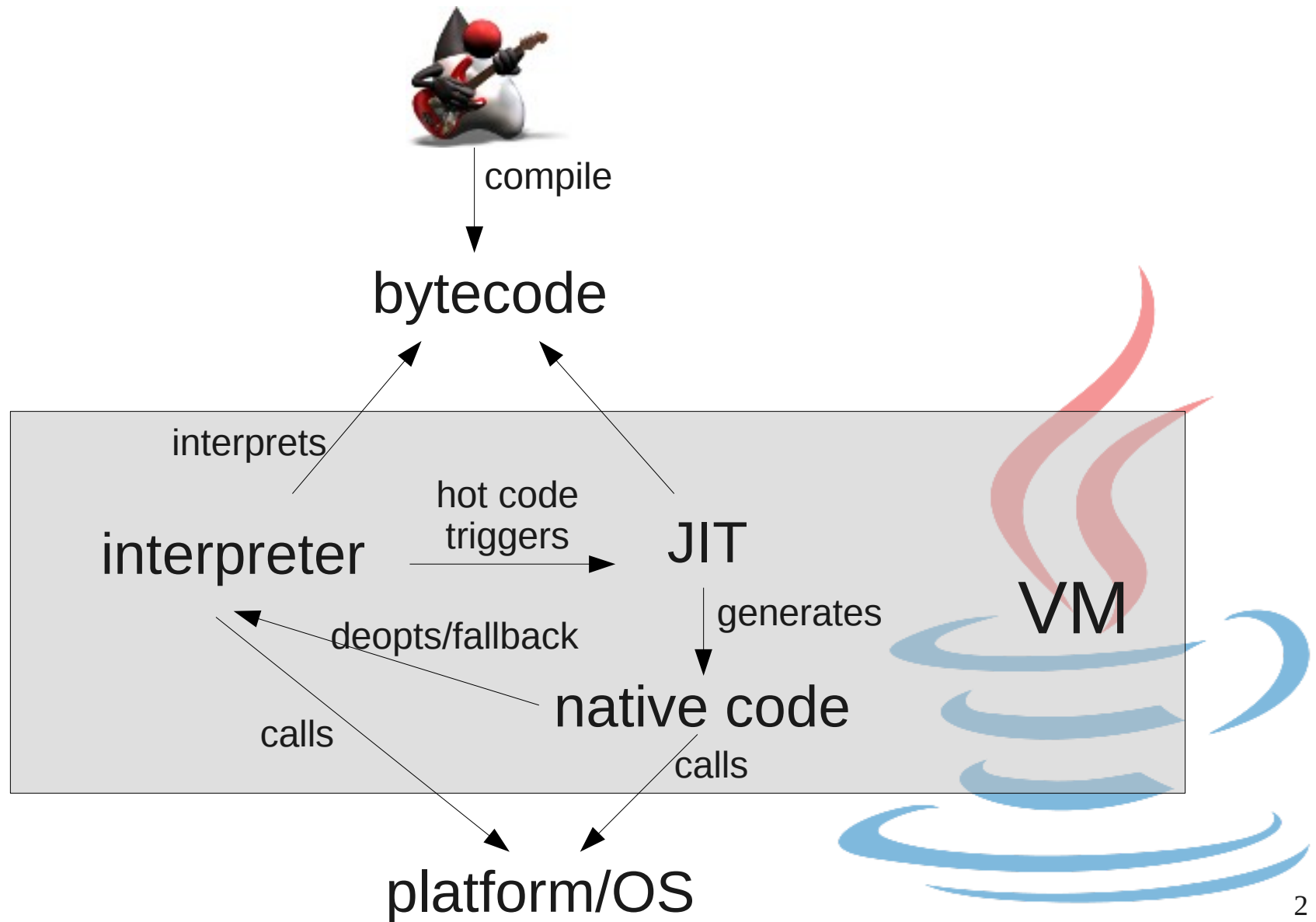
# JSR 292 / PHP.reboot

## Rémi Forax

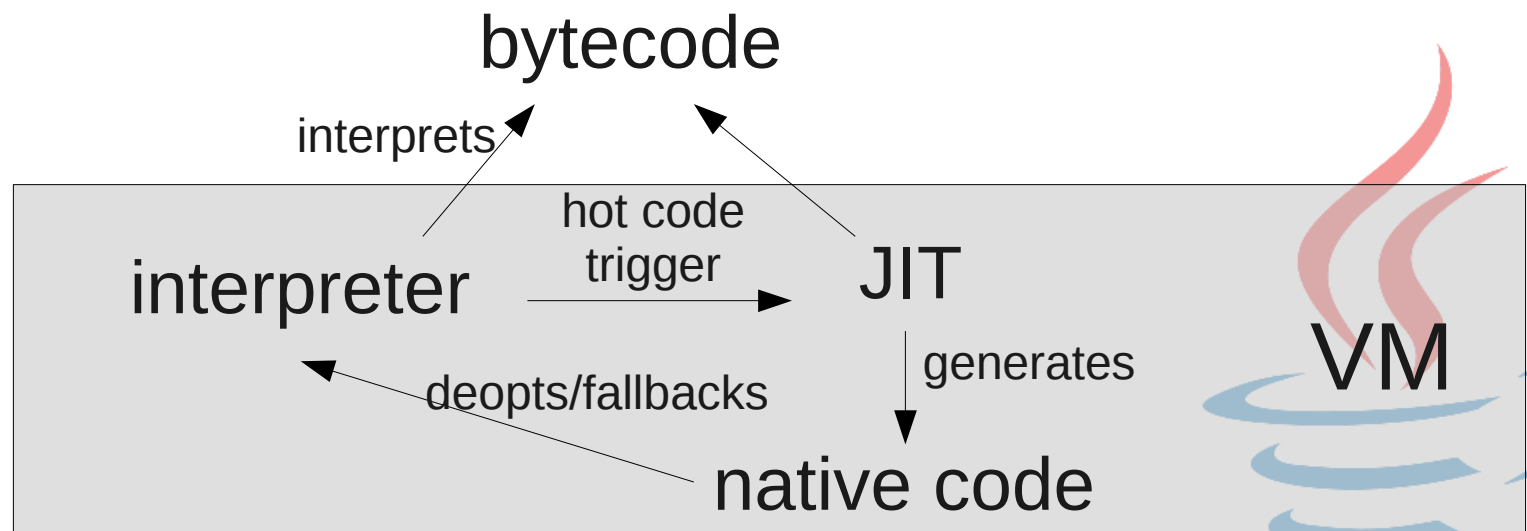
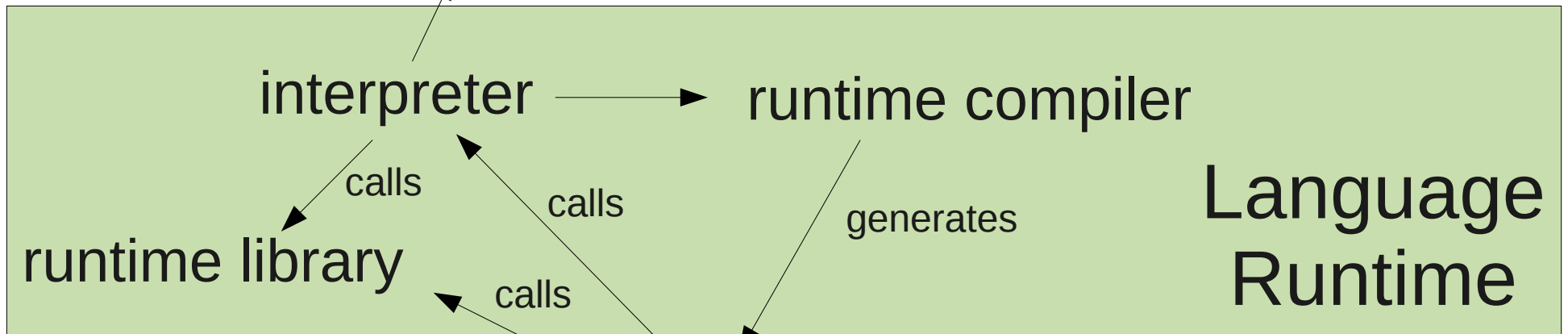


Université Paris-Est Marne-la-Vallée

# Classic JVM



# Dynamic Language Runtime



# Pain features of dynamic languages



no declared type

operators are overloaded

overflow

add/remove fields at runtime

add/remove/replace methods

*monkey patching*



Image © Thomas Hellberg - flickr

real dynamic code is uncommon

>> type inference & typechecking

monkey patching & overflow are rare

>> bailout guard + deoptimization

operator overload & duck typing

>> inlining cache/tree

# Hotspot JVM >> Multi-Language VM

OpenJDK - Da Vinci Project

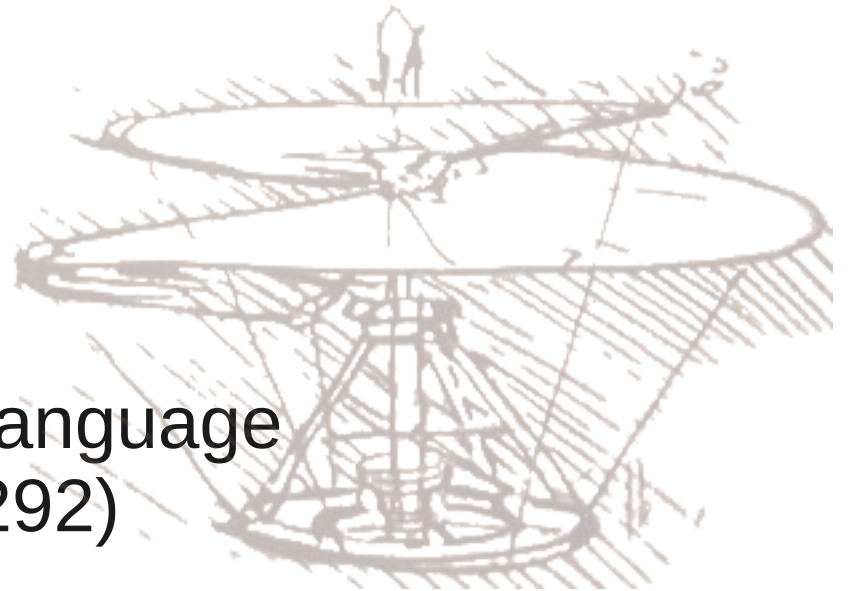
setup in 2007

First step

Support dynamically typed language  
on the Java Platform (JSR 292)

Next steps

Tailcalls, coroutine, continuation, stack inspection,  
interface extensions, immediate wrapper type, etc.





# PHP.reboot

# In few words ...

Secure!

No eval, no magic quote

DSLs for XML, JSON, SQL, Xpath, Xquery, etc.

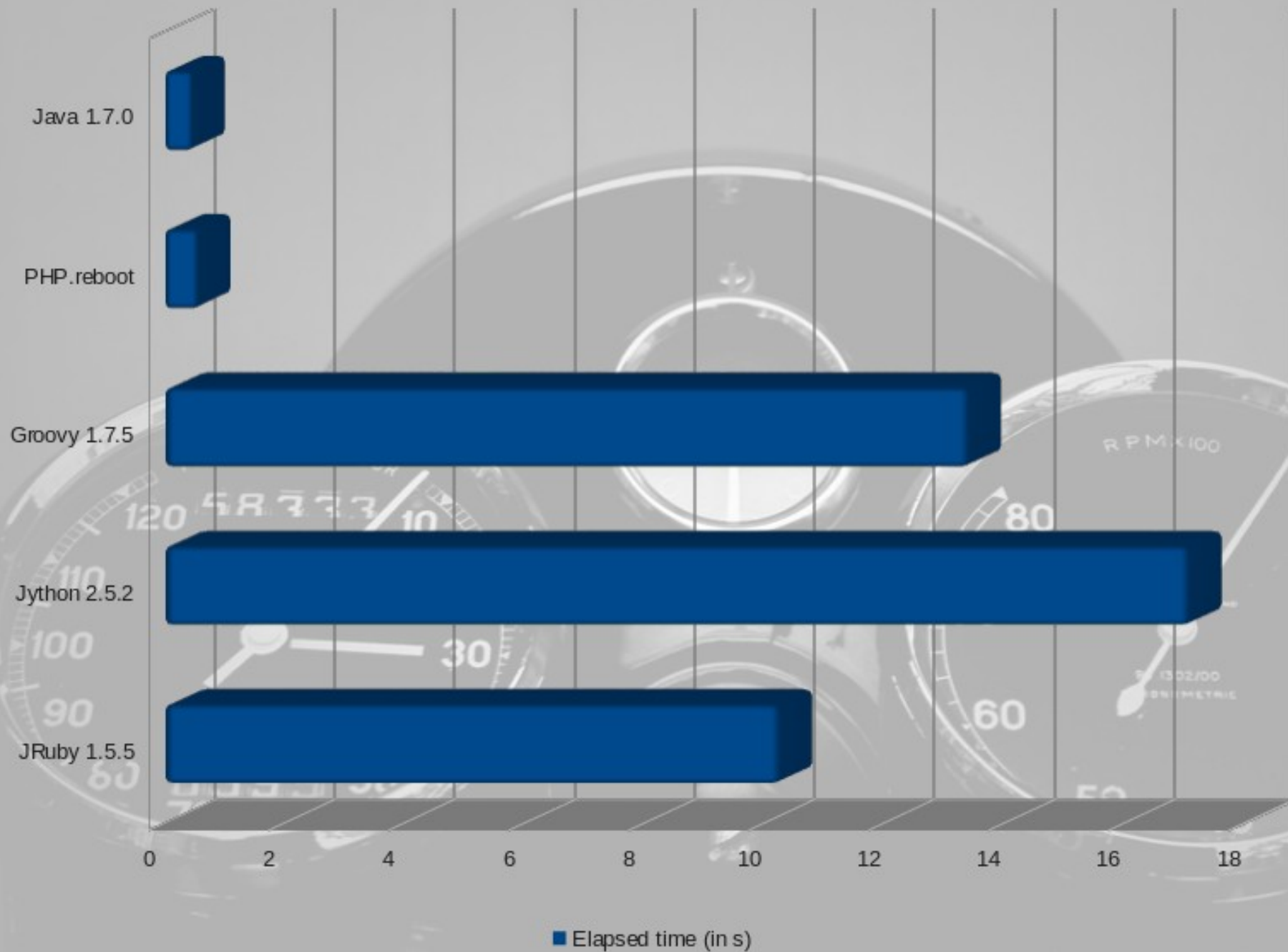
Sanitize depending on the context

Dynamically typed with gradual type system

Interpreter, ahead of time & runtime compilers

Compatible with 1.6/1.7 VMs





Render Mandelbrot, 1024x1024 tile, 50 iterations

# Toolchain

## **Tatoo** – parser generator (LALR)

Generate ASTs from grammars, change at runtime

Keyword are *local* by default

Evaluate AST asap (fast reduce)

## **JSR 292** + lightweight loading

Share runtime logic between interpreter and bytecode

Lambda/closure for free

## **ASM** – fast bytecode generation

Full supports of Java 7 classfile format



# PHP.reboot runtime

First interpret then compile

- update already existing call sites

Compile hot method and loops body

- interpreter profiles interpreted code

In loops:

- speculatively use profiled runtime types

  - prove with a simple forward typechecker

- specialize functions

- generate same code as javac

  - Revert and use invokedynamic if variable is really dynamic

- record untaken paths to generate escape exit

- profile field use/alloc-site to generate adhoc object



Image © Joe Penniston - flickr

# A simple dynamic example

```
for(i = 0; i < 350; i = i + 1) {  
    if (i == 100) {  
        i = 102.0  
    }  
}
```



# Type inference without escape

```
trace(LEvalEnv;Ljava/lang/Object;LVar;)Z
```

```
l0: aload 1
```

```
    sipush 350
```

```
    invokedynamic [#op] > (LObject;I)Z
```

```
    ifne l1
```

```
    aload 1
```

```
    bipush 100
```

```
    invokedynamic [#op] == (LObject;I)Z
```

```
    ifne l2
```

```
    ldc 102.0
```

```
    invokestatic Double.valueOf(D)LDouble; // boxing
```

```
    astore 1
```

```
l2: aload 1
```

```
    iconst_1
```

```
    invokedynamic [#op] + (LObject;I)LObject;
```

```
    astore 1
```

```
    goto l0
```

```
l1: aload 2    // post instruction, update interpreter variables
```

```
    aload 1
```

```
    invokevirtual Var.setValue (LObject;)V
```

```
    ...
```

# Optimization with escape trace

```
trace(LEvalEnv;LMethodHandle;LVar;)Z
```

```
l0: iload 1
    sipush 350
    if_icmpgt l1          // no loop peeling !
    iload 1
    sipush 100
    if_icmpne l2
    aload 2
    aload 0
    iload 1
    invokevirtual invoke(LEvalEnv;I)V
    iconst_0
    ireturn              // return false, trace escape !
```

```
l2: iload 1
    iconst_1
    iadd
    istore 1
    goto l0
```

```
l1: ...                // post instruction, update interpreter variables
```



# Bailout to interpreter

Need to restore:

- Interpreter's stack of values

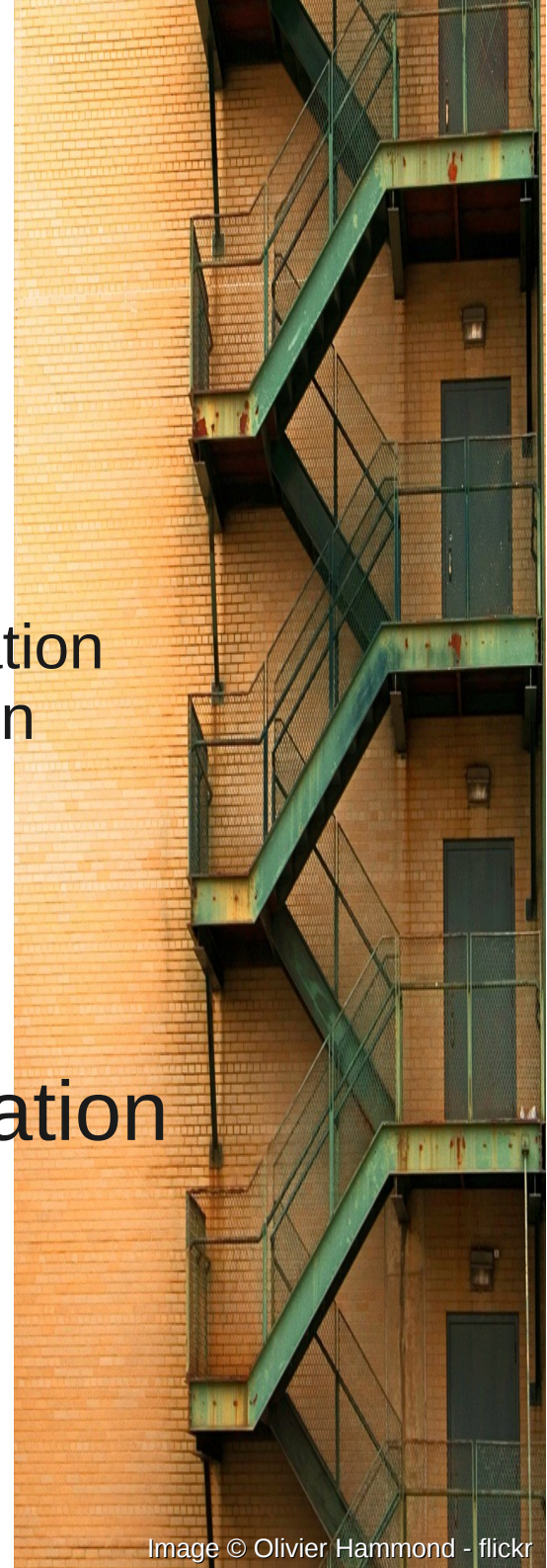
  - Typechecker prepares a blank stack + association between parameters position and stack position

- ~~Interpreter's call stack (walk on the AST)~~

Solution >>

Adhoc evaluator that completes the iteration

Loop may be re-optimized later



# Optimization after escape trace

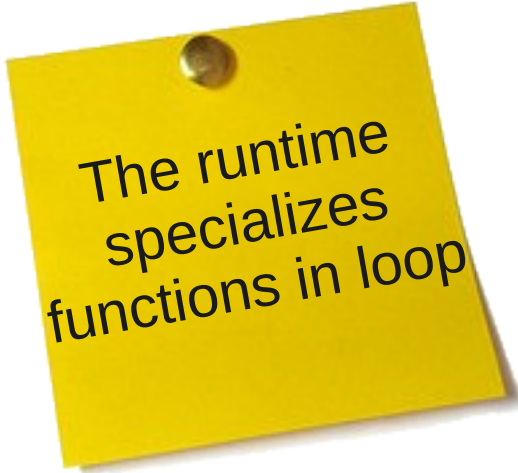
**trace(LEvalEnv;DLVar;)V**

```
l0: dload 1
    sipush 350
    i2d          // no constant propagation !
    dcmpg
    ifge l1
    dload 1
    bipush 100
    i2d
    dcmpg
    ifne l2
    ldc 102.0
    dstore 1
l2: dload 1
    iconst_1
    i2d
    dadd
    dstore 1
    goto l0
l1: ...          // post instruction, update interpreter variables
```



# And with function calls ?

```
function fibo(a) {  
  if (a < 2)  
    return 1  
  return fibo(a - 1) + fibo(a - 2)  
}  
  
i = 0  
while (i < 200) {  
  fibo(i % 7)  
  i = i + 1  
}
```



The runtime  
specializes  
functions in loop

# Fibo is called often

```
fibonacci(LObject;LObject;)LObject;
  aload 1
  iconst_2
  invokedynamic [#op] > (LObject;I)Z
  ifne l0
  iconst_1
  invokestatic Integer.valueOf(I)LInteger;    // boxing
  areturn
l0: aload 0
  aload 1
  iconst_1
  invokedynamic [#op] - (LObject;I)LObject;
  invokedynamic [#call] fibonacci(LObject;LObject;)LObject;
  aload 0
  aload 1
  iconst_2
  invokedynamic [#op] - (LObject;I)LObject;
  invokedynamic [#call] fibonacci(LObject;LObject;)LObject;
  invokedynamic [#op] + (LObject;LObject;)LObject;
  areturn
```

# 'while' is traced/compiled

**trace(LEvalEnv;LVar;)Z**

```
l0: iload 1
    sipush 200
    if_icmpge l1
    aload 0
    aload 1
    bipush 7
    irem
    invokedynamic [#call] fibo(LEvalEnv;l)LObject; // fibo is specialized !
    pop
    iload 1
    iconst_1
    iadd
    istore 1
    goto l0
l1: aload 2 // post instruction, update interpreter variables
    aload 1
    invokestatic Integer.valueOf(I)LInteger; // boxing
    invokevirtual Var.setValue(LObject;)V
    iconst_1
    ireturn
```

# Specialized Fibo

```
fibonacci(LObject;I)LObject;
  iload 1
  iconst_2
  if_icmpge l0
  iconst_1
  invokestatic Integer.valueOf(I)LInteger;    // boxing
  areturn
l0: aload 0
  iload 1
  iconst_1
  isub
  invokedynamic [#call] fibonacci(LObject;I)LObject;
  aload 0
  iload 1
  iconst_2
  isub
  invokedynamic [#call] fibonacci(LObject;I)LObject;
  invokedynamic [#op] + (LObject;LObject;)LObject;
  areturn
```

# Perspectives

Background compilation

Meta Protocol (Groovy 2)

# More ?

Da Vinci VM

<http://openjdk.java.net/projects/mlvm/>

JSR 292

<http://jcp.org/en/jsr/detail?id=292>

JSR 292 backport

<http://code.google.com/p/jvm-language-runtime/>

ASM

<http://asm.ow2.org/>

Tatoo

<http://gforgeigm.univ-mlv.fr/projects/tatoo/>

PHP.reboot

<http://code.google.com/p/phpreboot/>