A different approach: Just-in-time compilation

Marcel Schneider
2018-06-05

What we discussed so far

- C++ and Python
- Compiler runs once at Compile Time
- Output is executed instruction by instruction

- But: Compiler is limited to statically available information

A different approach

- Let’s run the compiler concurrently with the program
- Optimize the running code
- Use run-time information for optimization

- A powerful idea: Tracing Just-In-Time compilation

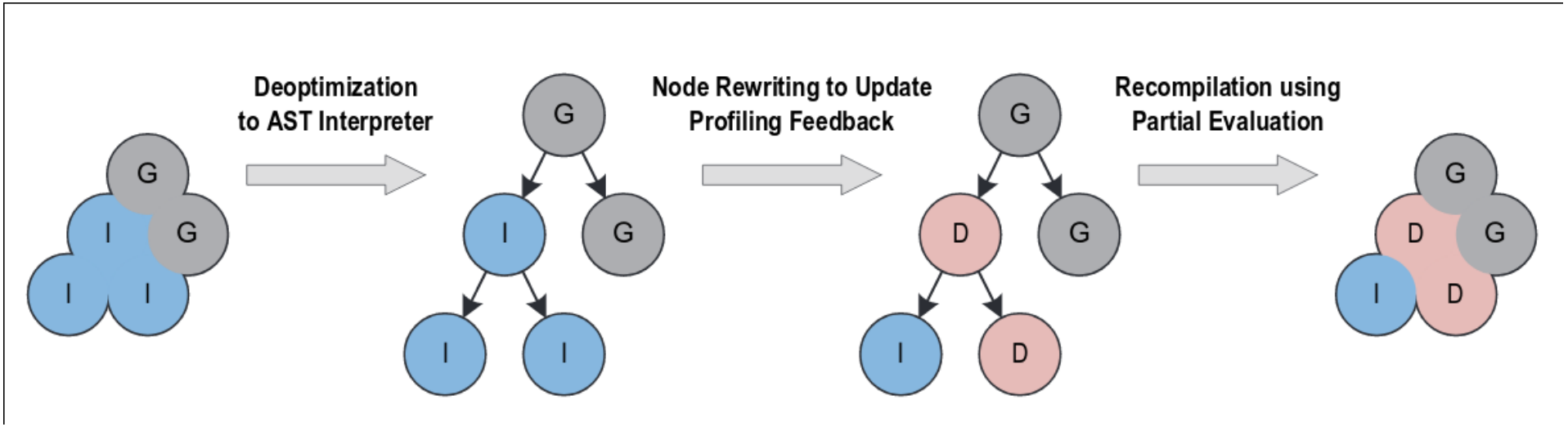
Tracing JIT

```
MOV_A_R     0   # i = a
MOV_A_R     1   # copy of 'a'

# 4:
MOV_R_A     0   # i--
DECR_A
MOV_A_R     0

MOV_R_A     2   # res += a
ADD_R_TO_A  1
MOV_A_R     2

MOV_R_A     0   # if i!=0: goto 4
JUMP_IF_A   4

MOV_R_A     2   # return res
RETURN_A
```

```
loop_start(a0, regs0, bytecode0, pc0)
# MOV_R_A 0
opcode0 = strgetitem(bytecode0, pc0)
pc1 = int_add(pc0, Const(1))
guard_value(opcode0, Const(2))
n1 = strgetitem(bytecode0, pc1)
pc2 = int_add(pc1, Const(1))
a1 = call(Const(<* fn list_getitem>), regs0, n1)
# DECR_A
opcode1 = strgetitem(bytecode0, pc2)
pc3 = int_add(pc2, Const(1))
guard_value(opcode1, Const(7))
a2 = int_sub(a1, Const(1))
# MOV_A_R 0
opcode1 = strgetitem(bytecode0, pc3)
pc4 = int_add(pc3, Const(1))
guard_value(opcode1, Const(1))
n2 = strgetitem(bytecode0, pc4)
pc5 = int_add(pc4, Const(1))
call(Const(<* fn list_setitem>), regs0, n2, a2)
# MOV_R_A 2
opcode2 = strgetitem(bytecode0, pc5)
pc6 = int_add(pc5, Const(1))
guard_value(opcode2, Const(2))
n3 = strgetitem(bytecode0, pc6)
pc7 = int_add(pc6, Const(1))
a3 = call(Const(<* fn list_getitem>), regs0, n3)
# ADD_R_TO_A 1
opcode3 = strgetitem(bytecode0, pc7)
pc8 = int_add(pc7, Const(1))
guard_value(opcode3, Const(5))
n4 = strgetitem(bytecode0, pc8)
pc9 = int_add(pc8, Const(1))
i0 = call(Const(<* fn list_getitem>), regs0, n4)
a4 = int_add(a3, i0)
# MOV_A_R 2
opcode4 = strgetitem(bytecode0, pc9)
pc10 = int_add(pc9, Const(1))
guard_value(opcode4, Const(1))
n5 = strgetitem(bytecode0, pc10)
pc11 = int_add(pc10, Const(1))
call(Const(<* fn list_setitem>), regs0, n5, a4)
# MOV_R_A 0
opcode5 = strgetitem(bytecode0, pc11)
```

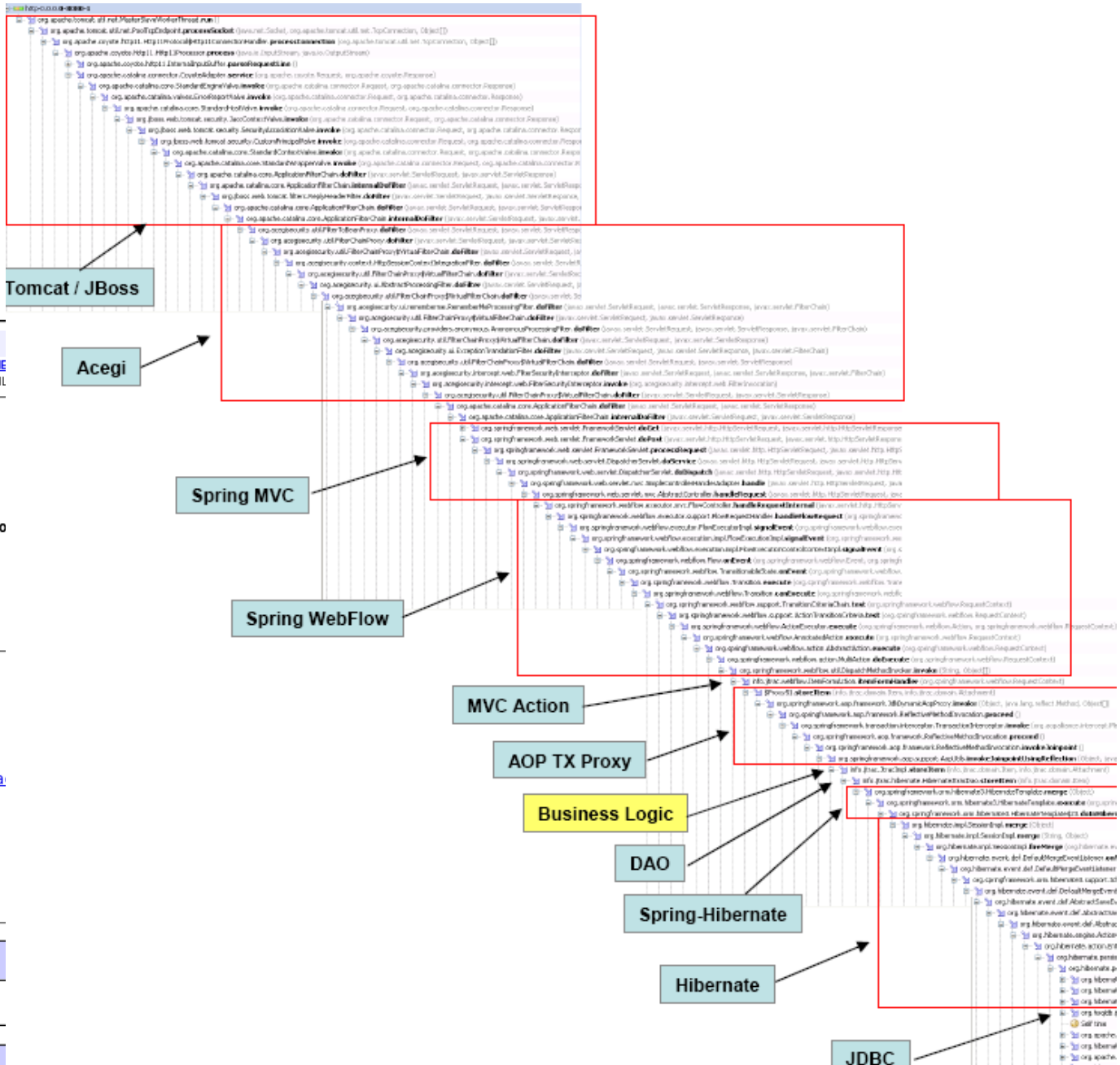
Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, Armin Rigo:
Tracing the meta-level: PyPy's tracing JIT compiler. ICOOOLPS@ECOOP 2009: 18-25

More things we can do



http://vjovanov.github.io/dsldi-summer-school/materials/friday/TruffleSummerSchool2015.pdf

Why?



org.springframework.aop.framework

Class AbstractSingletonProxyFactoryBean

java.lang.Object
  └ org.springframework.aop.framework.ProxyConfig
    └ **org.springframework.aop.framework.AbstractSingletonProxyFacto**

All Implemented Interfaces:
Serializable, BeanClassLoaderAware, FactoryBean, InitializingBean

Direct Known Subclasses:
TransactionProxyFactoryBean

---

public abstract class **AbstractSingletonProxyFactoryBean**
extends ProxyConfig
implements FactoryBean, BeanClassLoaderAware, InitializingBean

Convenient proxy factory bean superclass for proxy factory beans that create only singletons.

Manages pre- and post-interceptors (references, rather than interceptor names, as in ProxyFa

Since:
2.0
Author:
Juergen Hoeller
See Also:
Serialized Form

Constructor Summary

AbstractSingletonProxyFactoryBean()

Method Summary

Why?

- Extremely effective for huge codebases
- All the work done by the compiler/interpreter
- (Actually, the line between compilers and interpreters becomes blurry)

Why not?

- Extremely unpredictable performance
- Behaviour accurately described as “magic”
- Performance changes over time
- Small changes can have large consequences

- And: Extremely complicated

How to try?

- Java + HotSpot JVM 

- Truffle/Graal (JVM written in Java) 

- PyPy (Python interpreter written in Python) 

- LuaJIT 

- Formerly:
- TraceMonkey (old Javascript engine of Firefox)
- SPUR (Tracing JIT for .net CIL)

Finally...

- We see unpredictable, changing behaviour all the time!

Modern CPUs are JIT compilers for machine code.

Pointers

- PyPy:
  - https://pypy.org/
  - Paper: https://www3.hhu.de/stups/downloads/pdf/BoCuFiRi09_246.pdf
- Truffle/Graal:
  - https://www.graalvm.org/
  - Talks: https://www.youtube.com/playlist?list=PLaR98oFkCJzvqCgldUIzORVvejboipS3B
- LuaJIT: http://luajit.org/
- TraceMonkey:
  - Blogpost on why it failed: https://hacks.mozilla.org/2010/03/improving-javascript-performance-with-jagermonkey/