# Back to the Future in one Week — Implementing a Smalltalk VM in PyPy

Carl Friedrich Bolz[1] Adrian Kuhn[2] Adrian Lienhard[2]
Nicholas D. Matsakis[3] Oscar Nierstrasz[2]
Lukas Renggli[2] Armin Rigo Toon Verwaest[2]

[2]Software Composition Group
University of Bern, Switzerland

[1]Softwaretechnik und Programmiersprachen
Heinrich-Heine-Universität Düsseldorf

[3]ETH Zürich, Switzerland

Squeak e.V. Meeting, May 17 2008
(presented at S3)

# Scope

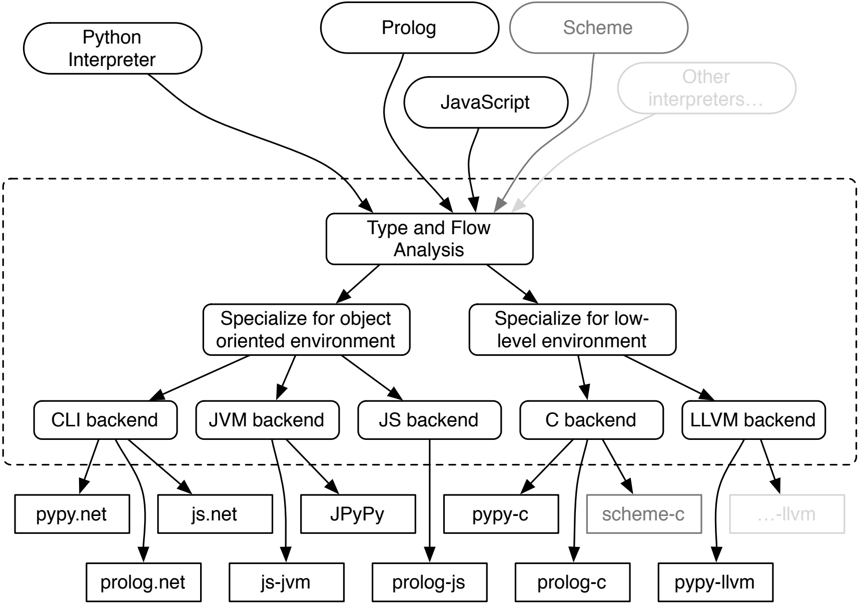
This demo is about:

- writing a Squeak implementation (called "SPy") in Python
- with eight people
- in five days
- using PyPy

# What is PyPy?

- started as a Python implementation in Python
- Open Source project, MIT license
- developed into a general environment for implementing dynamic languages
- supports the language developer with a lot of infrastructure
- most important goal: abstracting over low-level details
- don’t fix decisions about low-level details early

# What is PyPy?



Python Interpreter
Prolog
Scheme
JavaScript
Other interpreters…
Type and Flow Analysis
Specialize for object oriented environment
Specialize for low-level environment
CLI backend
JVM backend
JS backend
C backend
LLVM backend
pypy.net
js.net
JPyPy
pypy-c
scheme-c
…-llvm
prolog.net
js-jvm
prolog-js
prolog-c
pypy-llvm

# The SPy VM

- complete but simple, straight-forward Squeak interpreter in RPython
- goal is to fully support loading and running Squeak images
- source code essentially free of low-level details, no GC
- written during a five-day sprint in October in Bern

# Current Status

- bytecodes fully implemented
- many primitives implemented: arithmetic primitives, object primitives
- can load Squeak images (both 2.0 and 3.9)
- runs tiny benchmarks
- runs arbitrary code
- resume image

# Current Status

- bytecodes fully implemented
- many primitives implemented: arithmetic primitives, object primitives
- can load Squeak images (both 2.0 and 3.9)
- runs tiny benchmarks
- runs arbitrary code
- resume image

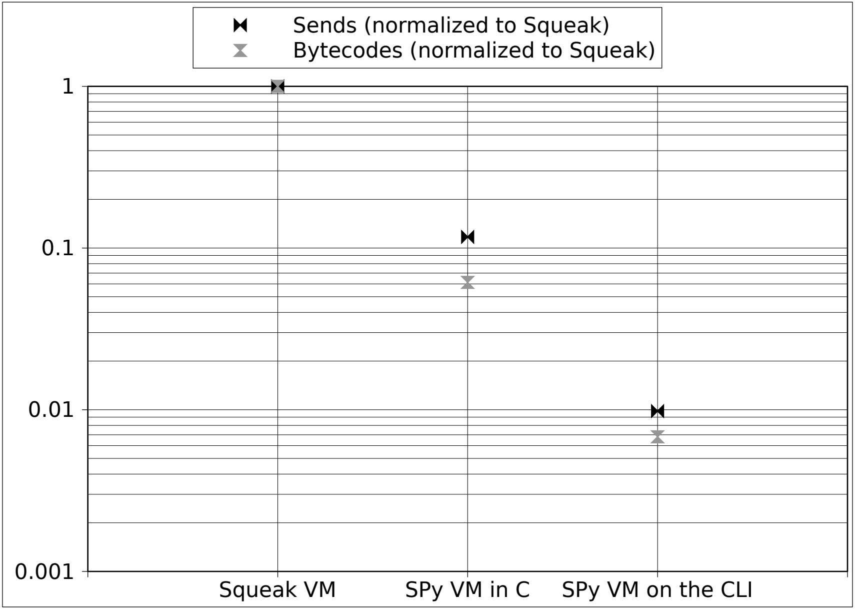
Open Issues:

- any I/O
- graphics
- performance
- #someInstance, #nextInstance
- #become
- tests for threading and exceptions

```
primitiveSquareRoot
  | rcvr |
  self var: #rcvr type: ’double ’.
  rcvr := self popFloat.
  self success: rcvr >= 0.0.
  successFlag
      ifTrue: [self pushFloat:
          (self cCode: ’sqrt(rcvr)’
                inSmalltalk: [rcvr sqrt])]
      ifFalse: [self unPop: 1]
primitiveSquareRoot
  | rcvr |
  self var: #rcvr type: ’double ’.
  rcvr := self popFloat.
  self success: rcvr >= 0.0.
  successFlag
      ifTrue: [self pushFloat:
          (self cCode: ’sqrt(rcvr)’
                inSmalltalk: [rcvr sqrt])]
      ifFalse: [self unPop: 1]
```

```
  @expose_primitive(FLOAT_SQUARE_ROOT,
                    unwrap_spec=[float])
def func(interp, f):
    if f < 0.0:
        raise PrimitiveFailedError
    w_res = utility.wrap_float(math.sqrt(f))
    return w_res
```

# Performance (tiny Benchmark)



# Results

**Good Points of the Approach:**

- simple, understandable, high-level Squeak implementation
- mostly free of low-level details: no GC, no manual pointer tagging
- written in a short amount of time
- runs on various platforms
- extensible tool chain

# Results

**Good Points of the Approach:**

- simple, understandable, high-level Squeak implementation
- mostly free of low-level details: no GC, no manual pointer tagging
- written in a short amount of time
- runs on various platforms
- extensible tool chain

**Bad Points of the Approach:**

- not really fast (yet)
- RPython isn't Python

# Outlook

- do graphical builtins, to actually start the full environment
- Squeak-specific optimizations:
- method-cache (should be easy with shadows)
- JIT
- lessons learned for a "SqueaSquea"?

## Join the Sprint!

Saturday - Thursday, C-Base Berlin