An Introduction to PyPy

Michael Hudson  
Canonical  
michael.hudson@canonical.com

Abstract
An overview of the ideas behind PyPy[1], its current status, future plans and why you should care about it.

The core idea of PyPy is to produce a flexible and fast implementation of the Python programming language. The talk will cover the interpreter, translator and jit parts of the code and their relationships and the fundamental ways in which PyPy differs from other virtual machine implementations.

1. Introduction
PyPy started out as an open source project to experiment with the implementation of the programming language Python, with the goal being to improve flexibility, performance, expressiveness and portability.

This lead to “PyPy's Big Idea”: to convert a high level description of Python into a low-level, efficient form. The way we specified Python was to implement an interpreter for it in a restricted subset of Python (called “RPython”) that is amenable to static analysis.

The translator that takes this description and converts it to a low level form proceeds in several steps: first type annotation, then successive reduction of the level of abstraction (deciding how the objects are laid out in memory and how to implement the garbage collector, for two examples).

Recent work as concentrated on automatically generating a tracing Just-In-Time compiler from the RPython interpreter.

What resulted from this effort is a translator framework that's particularly suited to implementations of dynamic languages, so an offshoot has been implementations of JavaScript, Prolog and Scheme using this framework.

The status of the Python implementation is that, when compiled to C without the JJIT, it is generally roughly the same speed as CPython, sometimes faster, more often slower. With the JIT enabled, some programs are several times faster than CPython[2], with this being true of more programs every week as work continues.

2. References
http://codespeak.net/pypy/dist/pypy/doc/  [3 November 2009]