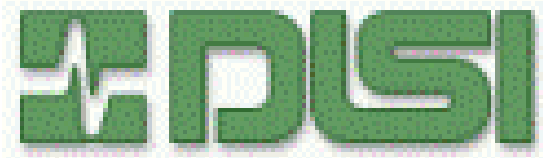
PyPy 1.1 - Present and Future

Antonio Cuni
DISI - Università di Genova
Merlinux GmbH

PyCon Tre 2009 - Firenze

May 9 2009



What this talk is about

- Why we work on PyPy?
- Details about recent 1.1 release
- What you can run on top of PyPy
- How fast is PyPy?
- Sandboxing
- Questions and Answers

PyPy - user motivation

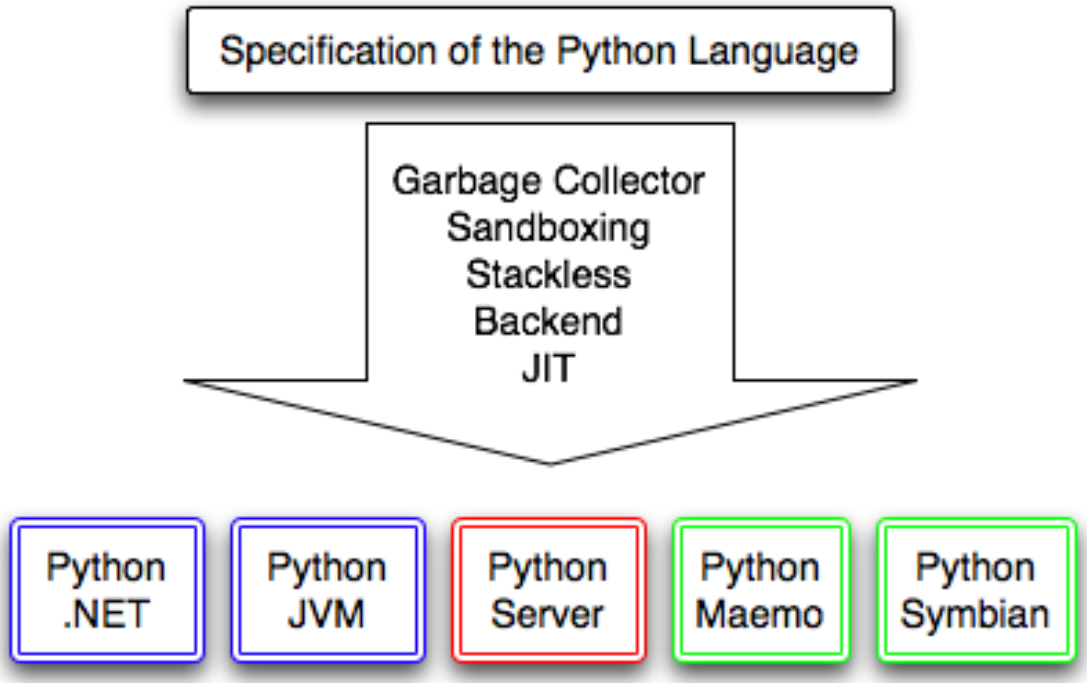
- use Python rather than C for performance
- have a more speedy, resource efficient interpreter
- support more programming paradigms

PyPy - motivation

- CPython is nice, but not flexible enough
- IronPython, Jython - bound to the specific VM
- Psyco and Stackless Python hard to maintain
- PyPy: flexible and easy to experiment with

PyPy: generating Python Interpreter

- **high level Python specification**!
- layer GCs, JIT, Stackless atop the spec
- **generate interpreters** for targets



Brief history of PyPy

- first sprint 2003, about 30 more by now
- CPython/Psyco/Jython/Stackless developers participating
- MIT-License, more sprints
- EU Research project 2004-2007
- 2007-now - open source project
- some Google sponsoring (thanks Guido :-))

1.1 release

- more than two years of work
- compatible to Python 2.5.2
- well tested on win/linux 32 bit
- speed improvements over 1.0
- running major packages unmodified
- easy_install/distutils working
- help e.g. by writing ctypes modules
- sandboxing
- support for Maemo devices

Getting Production ready

- we worked a lot on running existing applications on top of PyPy
- sometimes requiring to change applications slightly
- especially refcounting details tend to be a problem

```
open('xxx', 'w').write('stuff')
```

CTypes

- official way to have bindings to external (C) libraries for PyPy
- can handle i.e. pysqlite-ctypes, pyglet, pymunk or Sole Scion, almost whatever....
- contribution to original ctypes (better errno handling, bugfixes, tests...)
- part of Google sponsoring
- note: a bit slow

Sqlite

- part of cpython stdlib since 2.5
- we use Gerhard Haering's CTypes version
- works reasonably well after some fixes

Django

- we run unmodified Django 1.0
- only sqlite DB backend for now

http://www.djangoproject.com
http://code.djangoproject.com/wiki/DjangoAndPyPy

Pylons

- worked almost out of the box once eggs were working (1 day)
- no SQLAlchemy yet, obscure problems ahead
- unmodified passes all tests
- http://pylonshq.com/

Twisted & Nevow

- twisted works (60/4500 tests failing)
- nevow works
- we don't support PyCrypto nor PyOpenSSL and we won't anytime soon (if nobody contributes CTypes or rpython versions)
- http://twistedmatrix.com/

Other software

- pure python should just work
- BitTorrent
- PyPy translation toolchain
- py lib
- sympy
- various smaller things, templating engines

Obscure details that people rely on

- non-string keys in __dict__ of types
- exact naming of a list comprehension variable
- relying on untested and undocumented private stuff
- exact message matching in exception catching code
- refcounting details

Conclusion on Compatibility

- lessons learned: *there is no feature obscure enough for people not to rely on it.*
- pypy-c interpreter probably the most compatible to CPython 2.5
- main blocker for running apps will be missing external modules
  - greatest way to enter PyPy :-)

Speed - comparison with CPython

- we're something between 0.8-4x slower than CPython on various benchmarks without JIT
- our JIT will be super-fast (hopefully :-))
- pypy-c has fastest Interpreter startup

Speed - JIT generator (1)

- not included in 1.1
- big refactoring in-progress
- 5th generation ("... and easy to experiment with")
- x86 and CLI/.NET backends
- very easy to port to x86-64 (contributions welcome!)

Speed - JIT generator (2)

- 20-30x faster on small examples
- nice proof of concept
- a bit of time needed to speed up large python programs
- completely separated from the interpreter
- current plan: make it correct, make it fast

Speed - JIT for CLI/.NET

- originally written for the 2nd JIT generation
- can compile small dynamic languages - not full PyPy yet
- same speed as C# in numeric benchmarks
- up to 40% **faster** than C# for some OO benchmarks
- porting to 5th generation in-progress

Sandboxing - Definition

- ability to run untrusted Python code
- no way to:
  - interfere with the calling application
  - access system resources (files, etc.)
  - make a DoS by memory exhaustion
  - make a DoS by using 100% CPU

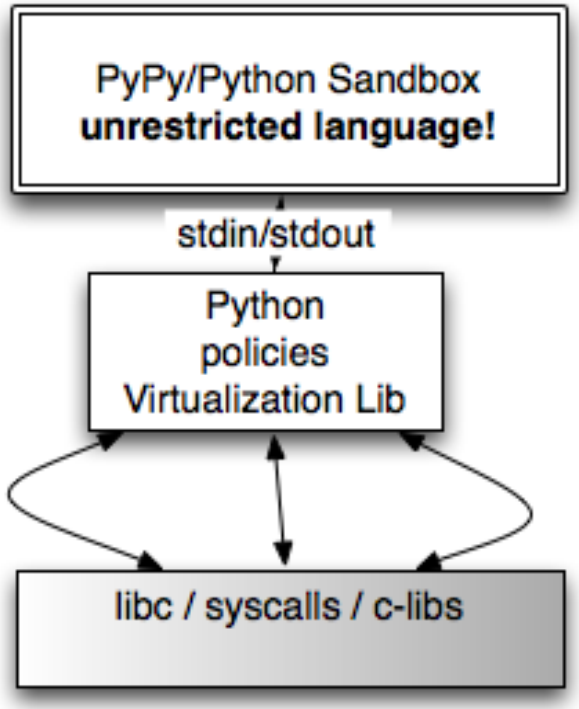
Sandboxing - Approaches

- bytecode/source verification
- CPython source modification/monkey patching
- platform-level security (GAE)
- restrict python language to something harmless (zope's restricted python)

Sandboxing - Problems

- Patchy approach - "*we fix all places that might be potentially dangerous*"
- Tradeoffs - either usability suffer or security is hard to control
- "Nobody cracked it so far" approach is not "security by design"

Sandboxing - PyPy approach



- C API implemented by parent process
  - virtual file system
- Memory limit
- CPU time limit

How to use it today?

- translate pypy with --sandbox (takes a while)
- run using pypy_interact.py
- demo
- you win a beer if you break it :-)

`http://codespeak.net/pypy/dist/pypy/doc/sandbox.html`

`http://codespeak.net/svn/user/getxsick/django-sandbox/`

Memory - comparison with CPython

- PyPy has smaller Python objects
- user class instances often 50% of CPython size!
- PyPy has pluggable Garbage Collection

Threading / Stackless

- currently using GIL
- free threading? “it’s work”
- pypy-c has software threading / stackless
  - tasklets, frame pickling, greenlets
  - fully cross-platform
- no modifications to interpreter involved

Other backends

- pypy-cli, pypy-jvm
- general speed improvements
- both backends are progressing - very slowly though
- contributors wanted!

pypy-c on small devices

- cross-compilation
- startup time
- security
- RAM usage
- share interpreter state across processes
- pypy approach a very good fit!

Contact / Q&A

Antonio Cuni at http://merlinux.eu
PyPy: http://codespeak.net/pypy
Blog: http://morepypy.blogspot.com