The PyPy Project and You

PyCon UK, Birmingham

Michael Hudson micahel@gmail.com
Canonical Ltd.

What you're in for in the next hour

- Quick intro and motivation
- Quick overview of architecture and current status
- Introduction to features unique to PyPy, including the JIT, with the odd demo
- A little talk about what the future holds

What is PyPy?

- PyPy is:
  - An implementation of Python in Python
  - A very flexible compiler framework (with some features that are especially useful for implementing interpreters)
  - An open source project (MIT license)
  - A lot of fun!

What was PyPy?

- PyPy *was* a Structured Targeted REsearch Proposal (STREP), part funded by the EU
- The funding period ended at the end of March 2007
- In May we had our final technical review, and

  "[PyPy] fully achieved its objectives and tech goals and has even exceeded expectations"

What we've got

- We can produce a binary that looks very much like CPython to the user
- Some, but not all, extension modules supported – socket, mmap, termios, …
- Can produce binary for CLR/.NET (watch out IronPython! :-)
- Can also produce binaries with more features (stackless, thunk, JIT, …)

Motivation

- PyPy grew out of a desire to modify/extend the *implementation* of Python, for example to:
    - Increase performance (psyco-style JIT compilation, better garbage collectors)
    - Add expressiveness (stackless-style coroutines, logic programming)
    - Ease porting (to new platforms like the JVM or CLI or to low memory situations)

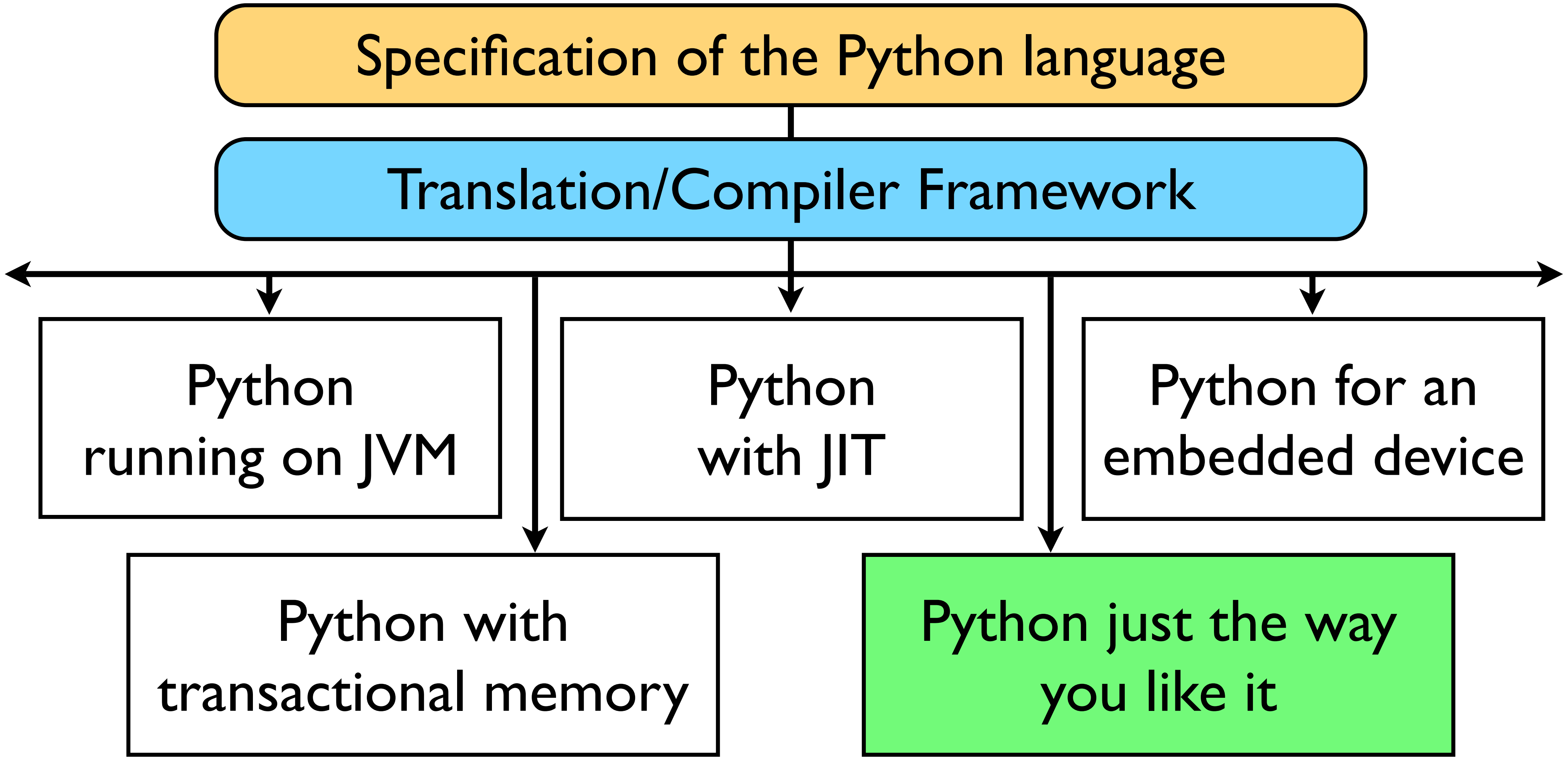
Problems with CPython

- CPython is a fine implementation of Python but:
  - It's written in C, which makes porting to, for example, the CLI hard
  - While psyco and stackless exist, they are very hard to maintain as Python evolves
  - Some implementation decisions are very hard to change (e.g. refcounting)

PyPy's Big Idea

- Take a *description* of the Python programming language
- Analyze this description:
  - Decide whether to include stackless- or psyco-like features
  - Decide which GC to use
  - Decide the target platform
- Translate to a lower-level, efficient form

The PyPy platform



Specification of the Python language
Translation/Compiler Framework
Python running on JVM
Python with JIT
Python for an embedded device
Python with transactional memory
Python just the way you like it

How do you specify the Python language?

- The way we did it was to write an interpreter for Python in *RPython* – a subset of Python that is amenable to analysis
- This allowed us to write unit tests for our specification/implementation that run on top of CPython
- Can also test entire specification/implementation in same way

The Translation/ Compiler Framework

- The compiler framework takes as input live Python objects (as opposed to source code)
- It *abstractly interprets* the bytecode of functions to produce flow graphs
- Further layers of abstract interpretation perform more analysis and gradually reduce the level of abstraction
- Finally C or other source code is generated

If you have a hammer…

- We'd written this compiler framework, with only one expected non-trivial input (our Python interpreter)
- We realized that it would be suitable for implementations of other dynamically-typed programming languages
- Now have implementations of Prolog, JavaScript and Scheme (to varying extents)

The *L×O×P* problem

This leads to one of PyPy's meta-goals, ameliorating the so-called *L×O×P* problem: given

- *L* dynamic languages
- *O* target platforms
- *P* implementation decisions

we don't want to have to write *L×O×P* different interpreters by hand.

The *L×O×P* problem

- PyPy aims to reduce this to an *L+O+P* problem:
  - Implement *L* language front-ends
  - Write backends for *O* platforms
  - Take *P* implementation decisions
- Then let the *magic of PyPy*™ tie it all together :-)

Status – Interpreter

- PyPy's Python interpreter supports 2.4.4 by default
- Can activate 2.5 with an option, 2.6 should be easy enough
- No Py3K yet :-) (shouldn't be too hard)
- The "__pypy__" module includes a variety of mysterious and exciting things, depending on options supplied

Status – compiled interpreter

- When compiled to (standalone) C with all the optimizations turned on, our interpreter is ~1.5-4 times slower than CPython
- The still-experimental Just-In-Time compiler runs some programs 60 times faster than CPython!
- Can also build interpreters with threading and with stackless features

Status – backends

- We currently have two complete backends:
  - C/POSIX (like CPython)
  - CLI (like IronPython)
- Also have LLVM (low-level virtual machine) backend, which is currently unmaintained
- Incomplete backends include JVM (nearly there!), Common Lisp, Squeak, …

Status – backends

- Also have a special-case backend: the JavaScript backend
  - Can't translate all of PyPy, but not really expected to
  - Can be used to write some of the client side parts of your web-app in [R]Python

(Demo)

Things that make PyPy unique

- The Just-In-Time compiler (and the way it has been made)
- Sandboxed Executable
- Transparent Proxies
- Runtime modifiable Grammar
- Thunk object space
- JavaScript backend
- Logic programming

About the project

- Open source, of course (MIT license)
- Entering a transitional time
  - After the funding period ended, everyone slept for a couple of months :-)
  - Starting to wake up again now

About the project

- Sprint driven development – focused week long coding sessions
  - Every ~6 weeks during funding period
  - Every ??? now – anyone want to host a sprint? (John tried to have one here, timing didn't work out)
- Extreme Programming practices: pair programming, test-driven development

Future Facts

- Funding period ends March 31st
  - Some funding related admin remains – reports, reviews
- So PyPy development will end? Of course not!
  - PyPy was a hobbyist open source project before funding, will return to that state
  - … for a while, at least

Future Hopes

- At least in my opinion, the work so far on PyPy has mostly been preparatory – the real fun is yet to come.
- Likely future work includes:
  - More work on the JIT
  - Reducing code duplication
  - Improved C gluing, better GIL handling

Future Dreams

- High performance compacting, generational, etc GC (steal ideas from Jikes?)
- Implementations of other dynamic languages such as JavaScript, Prolog (already started), Ruby (?), Perl (??) (which will get a JIT essentially for free)
- The ability to have dynamically loaded extension modules

Join the fun!

- Project relies more than ever on getting the community involved
- Read documentation:

  http://codespeak.net/pypy/
- Come hang out in `#pypy` on freenode, post to pypy-dev
- Probably will be easier to keep up now…

Thanks for listening!

Any Questions?