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 statues
- 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 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 we've got

- We can produce a binary that looks very much like CPython to the user (it's 2-4x slower, depending on details)
- 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

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)

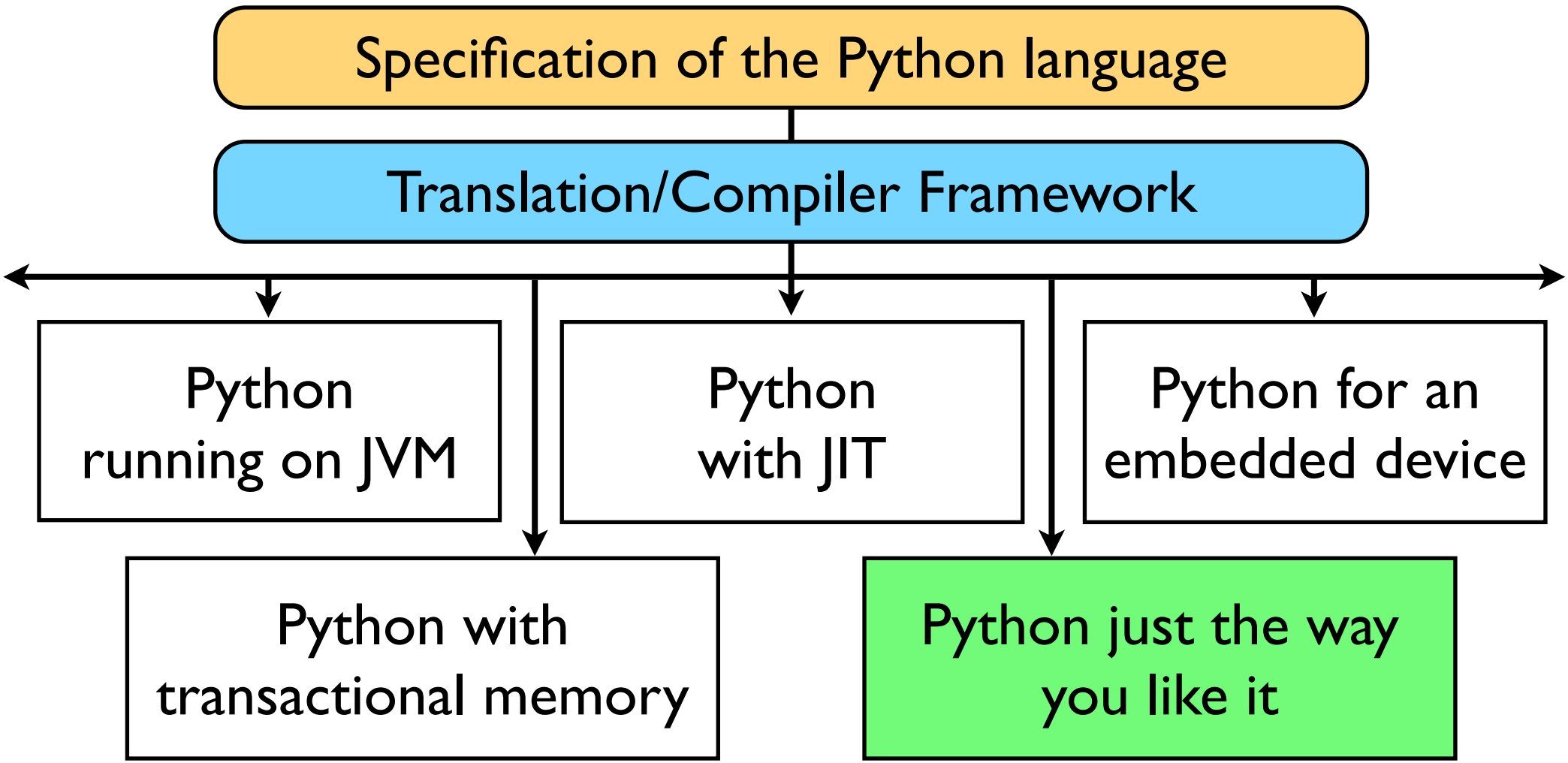
Problems with CPython

- 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



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

If you have a hammer…

- We'd written this compiler framework, with only one expected 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 some extent)

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.

Translation Aspects

- Our Python implementation/specification is very high level
- One of our Big Goals is to produce our customized Python implementations without compromising on this point
- We do this by weaving in so-called ‘translation aspects’ during the compilation process

Status

- The Standard Interpreter almost completely compatible with Python 2.4.4
- The compiler framework:
  - Produces standalone binaries
  - C, LLVM and CLI backends well supported, JVM very nearly complete
  - JavaScript backend works, but not for all of PyPy (not really intended to, either)

Status

- The C backend support “stackless” features – coroutines, tasklets, recursion only limited by RAM
- Can use OS threads with a simple “GIL-thread” model
- Our Python specification/implementation has remained free of all these implementation decisions!

What we're working on now

- The JIT
  - i386, PowerPC and LLVM backends
- Object optimizations
  - Dict variants, method caching, …
- Integration with .NET
- Security and distribution prototypes
  - Not trying to revive rexec for now though…

Things that make PyPy unique

- The Just-In-Time compiler (and the way it has been made)
- Transparent Proxies
- Runtime modifiable Grammar
- Thunk object space
- JavaScript (demos: b-n-b and rxconsole)
- Logic programming

About the project

- Open source, of course (MIT license)
- Distributed – the 12 paid developers live in 6 countries, contributers from many more
- Sprint driven development – focussed week long coding sessions, every ~6 weeks during funding period
- 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?