PyPy 1.0 and beyond

PyCon 07, Addison TX

Michael Hudson mwh@python.net
Heinrich-Heine-Univeristät Düsseldorf

What you're in for in the next 45 mins

- Quick intro and motivation
- Quick overview of architecture
- Some demos of features unique to PyPy, including the JIT
- A little talk about what the future holds

What is PyPy?

- PyPy is:
  - An implementation of Python, and a very flexible compiler framework (with some features that are especially useful for implementing interpreters)
  - An open source project (MIT license)
  - A STREP ("Specific Targeted REsearch Project"), partially funded by the EU
  - A lot of fun!

Status

- We can (still :-) produce a binary that looks very much like CPython to the user
- It's still a bit slower, depending on options and what you're doing
- Can also produce binaries that are more capable than CPython – with stackless-style coroutines, with logic variables, ...
- Can also produce binary for CLR (i.e. .NET)

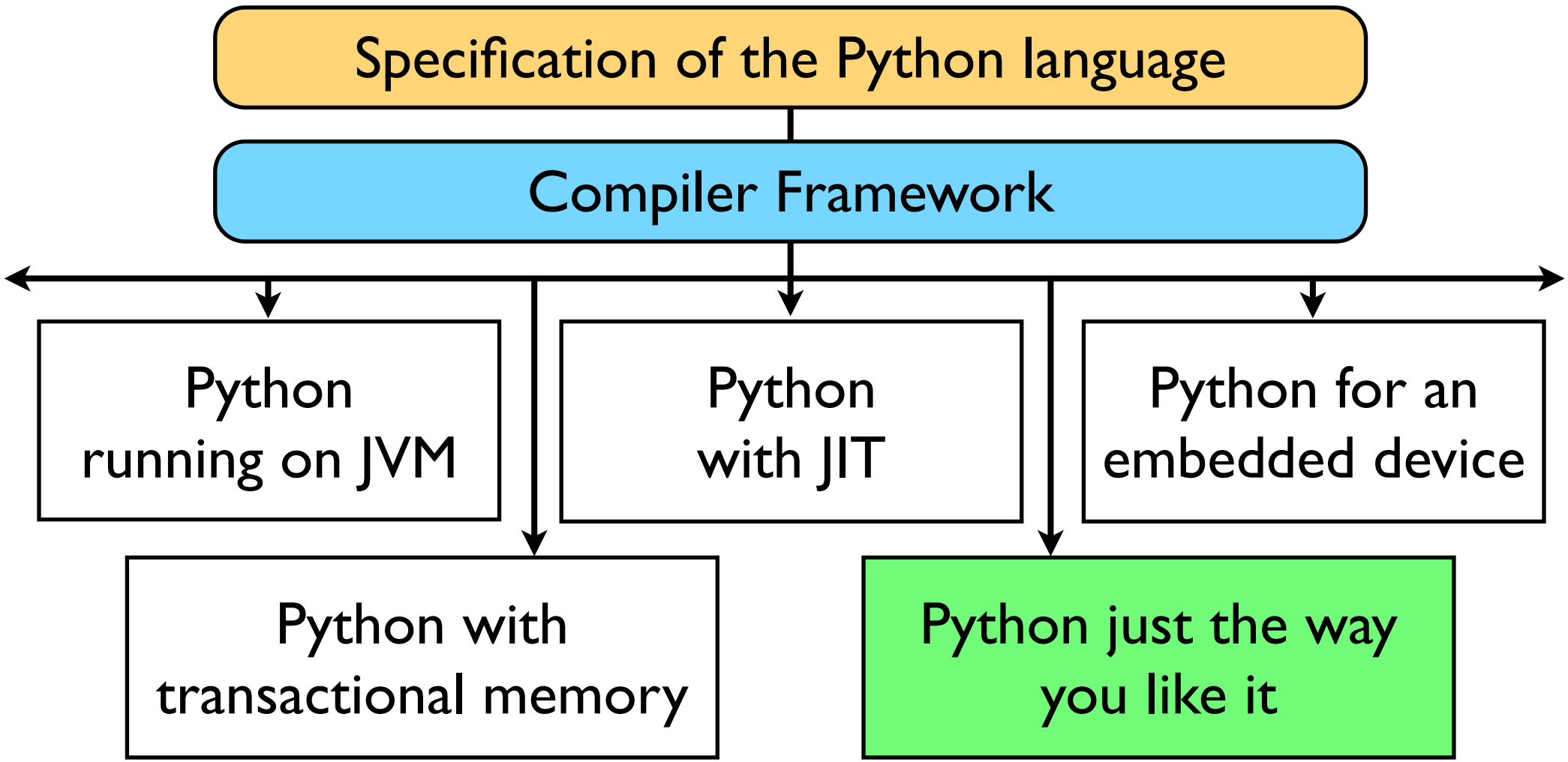
Motivation

- 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)

Lofty goals, but first...

- 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)

Enter 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 lets us write unit tests for our specification/implementation that run on top of CPython
- Can also test entire specification/ implementation in same way

The "What is RPython?" question

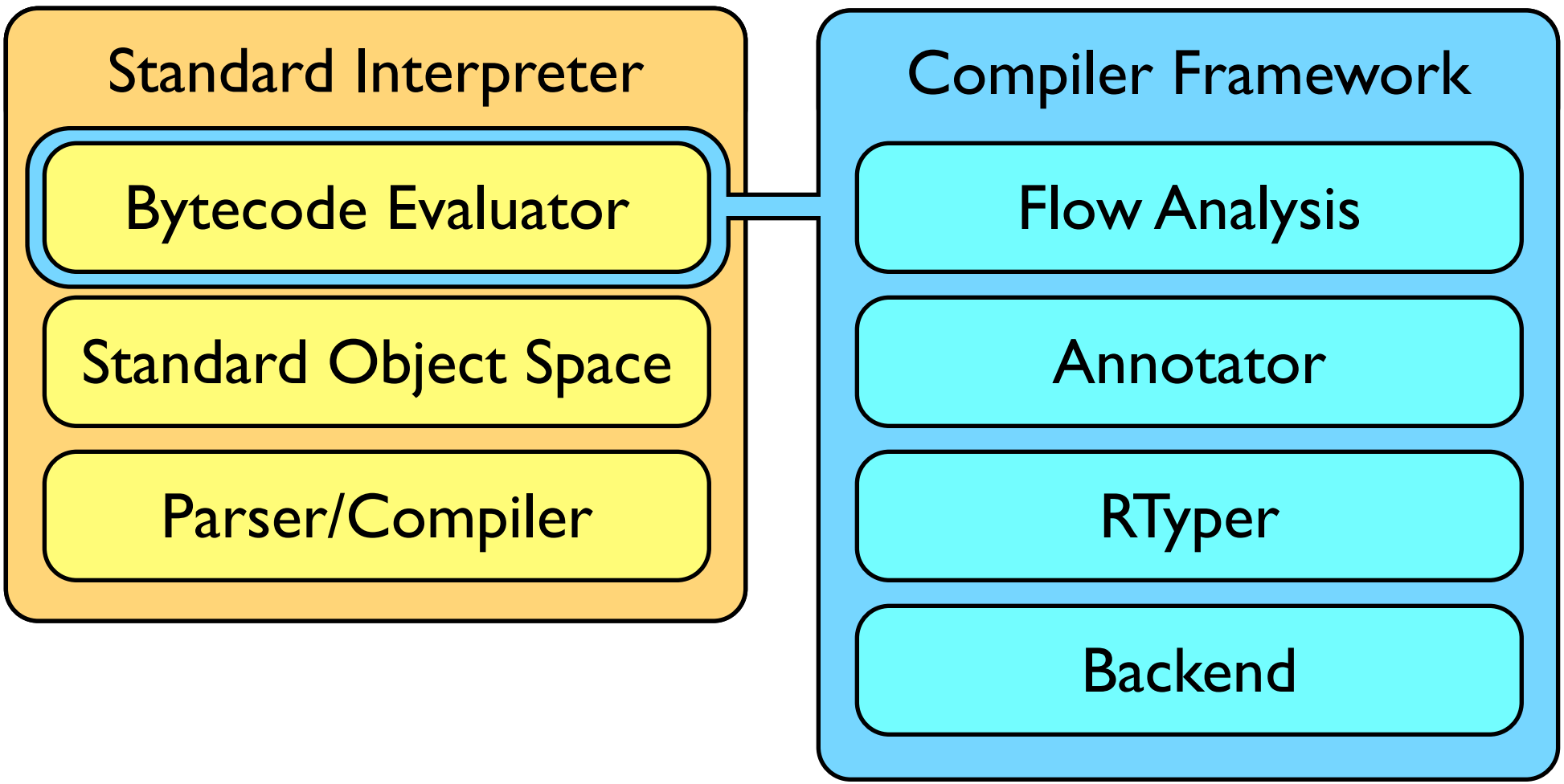
- Restricted Python, or RPython, first and foremost *is* Python
- It is a subset of Python that is static enough – *after initialization code has run* – for our analysis tools to cope with
- Somewhat Java-like – classes, methods, no pointers, no operator overloading – but at least has function pointers

Translation Aspects

- Our Python implementation 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

1,000 ft view


PY.


Standard Interpreter
Bytecode Evaluator
Standard Object Space
Parser/Compiler
Compiler Framework
Flow Analysis
Annotator
RTyper
Backend

The Annotator

- Type annotation associates variables with information about which values they can take at run time
- An unusual feature of PyPy's approach is that the annotator works on live objects which means it never sees initialization code, so that can use `exec` and other dynamic tricks

The Annotator

- Annotation starts at a given entry point and discovers as it proceeds which functions may be called by the input program
- Does not modify the graphs; end result is essentially a big dictionary
- Read "Compiling dynamic language implementations" on the web site for more than is on these slides

The RTyper

- The RTyper takes as input an annotated RPython program (e.g. our Python implementation)
- It reduces the abstraction level of the graphs towards that of the target platform
- This is where the magic of PyPy really starts to get going :-)

The RTyper

- Can target a C-ish, pointer-using language or an OO language like Java or Smalltalk with classes and instances
- Resulting graphs are not completely low-level: still assume automatic memory management for example

Reducing Abstraction

- Many high level operations apply to different types – the most extreme example probably being calling an object
- For example, calling a function is RTyped to a `direct_call` but calling a class is RTyped to a sequence of operations including allocating the memory for the instance and calling any `__init__` function

Further Transforms

- RTyping is followed by a sequence of further transforms, depending on target platform and options supplied:
  - GC transformer – inserts explicit memory management operations
  - Stackless transform – inserts bookkeeping and extra operations to allow use of coroutines, tasklets etc
  - Exception transform – makes exception handling explicit

The Backend(s)

- Maintained backends: C, LLVM, CLI/.NET, and JavaScript
- All proceed in two phases:
  - Traverse the forest of rtyped graphs, computing names for everything
  - Spit out the code

Status – what works

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

Status – what works

- 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 Just-In-Time compiler – early stages, works for a very simple language
- More home-grown GCs (e.g. a semispace copying collector) and more GCs for LLVM
- Logic programming – some working code, interface and integration in progress

What we're working on now

- "rctypes", a uniform way of calling external functions based on the now-standard "ctypes" module for CPython
- CLI (.NET) and Smalltalk backends
- supporting stackless features in other backends

About the project

- Open source, of course (MIT license)
- Distributed – the 12 paid developers live in 6 countries, contributers from more
- Sprint driven development – focussed week long coding sessions every ~6 weeks
- Extreme Programming practices: pair programming, test-driven development

“We’re Hiring!”

- In the open source sense:
  - Read documentation:

    http://codespeak.net/pypy/

  - Come hang out in `#pypy` on freenode, post to pypy-dev
- Some opportunities for paid work too.