# PyPy
## So what is this, exactly?

Atul Varma
The Chicago Python Users Group
April 12, 2007

# PyPy is ...

*a common translation and support framework for producing implementations of dynamic languages, emphasising a clean separation between language specification and implementation aspects.*

http://codespeak.net/pypy/dist/pypy/doc/architecture.html#mission-statement

http://www.flickr.com/photos/kimhartig/127166291/

*... implementations of dynamic languages, ...*

Think Python
Javascript
Ruby
Prolog
Lisp
Lua
HyperTalk

# ... *implementations* *of dynamic languages, ...*

Think CPython
Jython
IronPython
JRuby
Cardinal

*... between language specification and ...*

Think Python 1.0
Python 2.5
Python 3000

*... between language specification and ...*

# Syntax
(structural rules)

# Semantics
(what statements and expressions in the language actually *mean*)

*... specification and implementation aspects.*

Think Target platform/language
(x86, .NET, Parrot, LLVM, Javascript)

Just-in-time compilation

*... specification and implementation aspects.*

Think Concurrency model
(GIL-based, non-GIL-based)

Memory management
(mark-and-sweep GC, reference counting GC)

# ... specification and *implementation aspects.*

In CPython and other hand-written interpreters, these are things that are tightly interwoven into the code.

Light blue highlight indicates reference-counting code.

Light green highlight indicates GIL manipulation code.

```
if (PyUnicode_Check(po1) || PyUnicode_Check(po2)) {
    PyObject *wpath1;
    PyObject *wpath2;
    wpath1 = _PyUnicode_FromFileSystemEncodedObject(po1);
    wpath2 = _PyUnicode_FromFileSystemEncodedObject(po2);
    if (!wpath1 || !wpath2) {
        Py_XDECREF(wpath1);
        Py_XDECREF(wpath2);
        return NULL;
    }
    Py_BEGIN_ALLOW_THREADS
    /* PyUnicode_AS_UNICODE OK without thread
       lock as it is a simple dereference.  */
    res = (*wfunc)(PyUnicode_AS_UNICODE(wpath1),
                   PyUnicode_AS_UNICODE(wpath2));
    Py_END_ALLOW_THREADS
    Py_XDECREF(wpath1);
    Py_XDECREF(wpath2);
    if (res != 0)
        return posix_error();
    Py_INCREF(Py_None);
    return Py_None;
}
```

*from posixmodule.c*

# Think Stacklessness

It took Christian Tismer about 6 months of work to create Stackless Python as a series of CPython patches.

It took a couple of days and about 300 lines of code to implement the same thing as a "localized translation aspect" in PyPy.

so, what is
*“a common translation
and support framework?”*

Basically, it's something that allows you to do things like:

Take any code written in a *restricted* subset of Python and translate it to any language or platform.

(e.g., Javascript, C, .NET, JVM, LLVM.)

Create a custom dynamic language interpreter implemented in any language or platform, with your choice of language and implementation features.

(e.g., a Python interpreter implemented in C that supports JIT compilation and Stacklessness.)

Easily add new language or implementation features to an existing dynamic language interpreter.

(e.g., create a Python interpreter without a GIL, or implement object tainting.)

So how does it work?

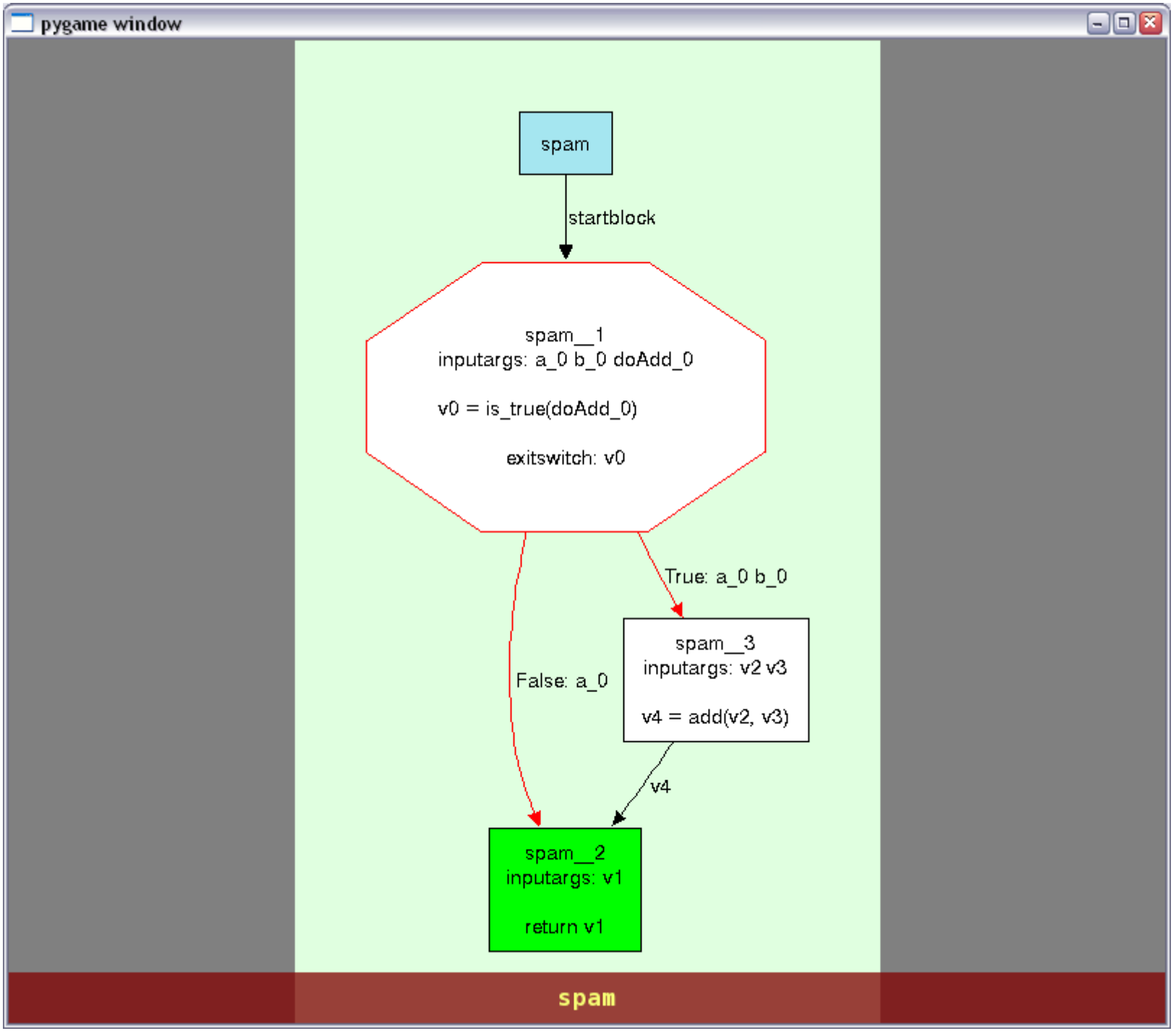
# Let's disassemble some Python code.

```
>>> def spam(a, b, doAdd):
...  if doAdd:
...    return a + b
...  else:
...    return a
>>> spam.func_code.co_varnames
('a', 'b', 'doAdd')
>>> import dis    # This is a standard Python library module.
>>> dis.dis(spam)
         2           0 LOAD_FAST                2 (doAdd)
                     3 JUMP_IF_FALSE           12 (to 18)
                     6 POP_TOP

         3           7 LOAD_FAST                0 (a)
                    10 LOAD_FAST                1 (b)
                    13 BINARY_ADD
                    14 RETURN_VALUE
                    15 JUMP_FORWARD             5 (to 23)
               >>   18 POP_TOP

         5          19 LOAD_FAST                0 (a)
                    22 RETURN_VALUE
               >>   23 LOAD_CONST               0 (None)
                    26 RETURN_VALUE
```

```
t = Translation(spam); t.view()
```



```
t = Translation(spam); t.annotate([int,int,bool]); t.view()
```