Converting Python functions to dynamically compiled C

Ilan Schnell
Enthought, Inc.

Motivation

- Many tools for creating and wrapping C
- Function needs to written in C, pyrex, ...
- Significant overhead:
  - Learning the tool
  - Additional files
  - Extra build step

PyPy

- CPython implemented in C
- PyPy was first implemented in Python
- For speed, PyPy in now written in RPython
- RPython is a restricted subset of Python
- Rpython can be translated to C, LLVM, ...
- PyPy project includes this translator
- Translator can compile entire interpreter!

Compile decorator

Uses pypy to convert RPython to C, and wrapping C code into extension module.

```
# compdec.py
from pypy.translator.interactive import Translation

class compdec:
    def __init__(self, func):
        self.func = func
        self.argtypes = None

    def __call__(self, *args):
        argtypes = tuple(type(arg) for arg in args)
        if argtypes != self.argtypes:
            self.argtypes = argtypes
            t = Translation(self.func)
            t.annotate(argtypes)
            self.cfunc = t.compile_c()

        return self.cfunc(*args)
```

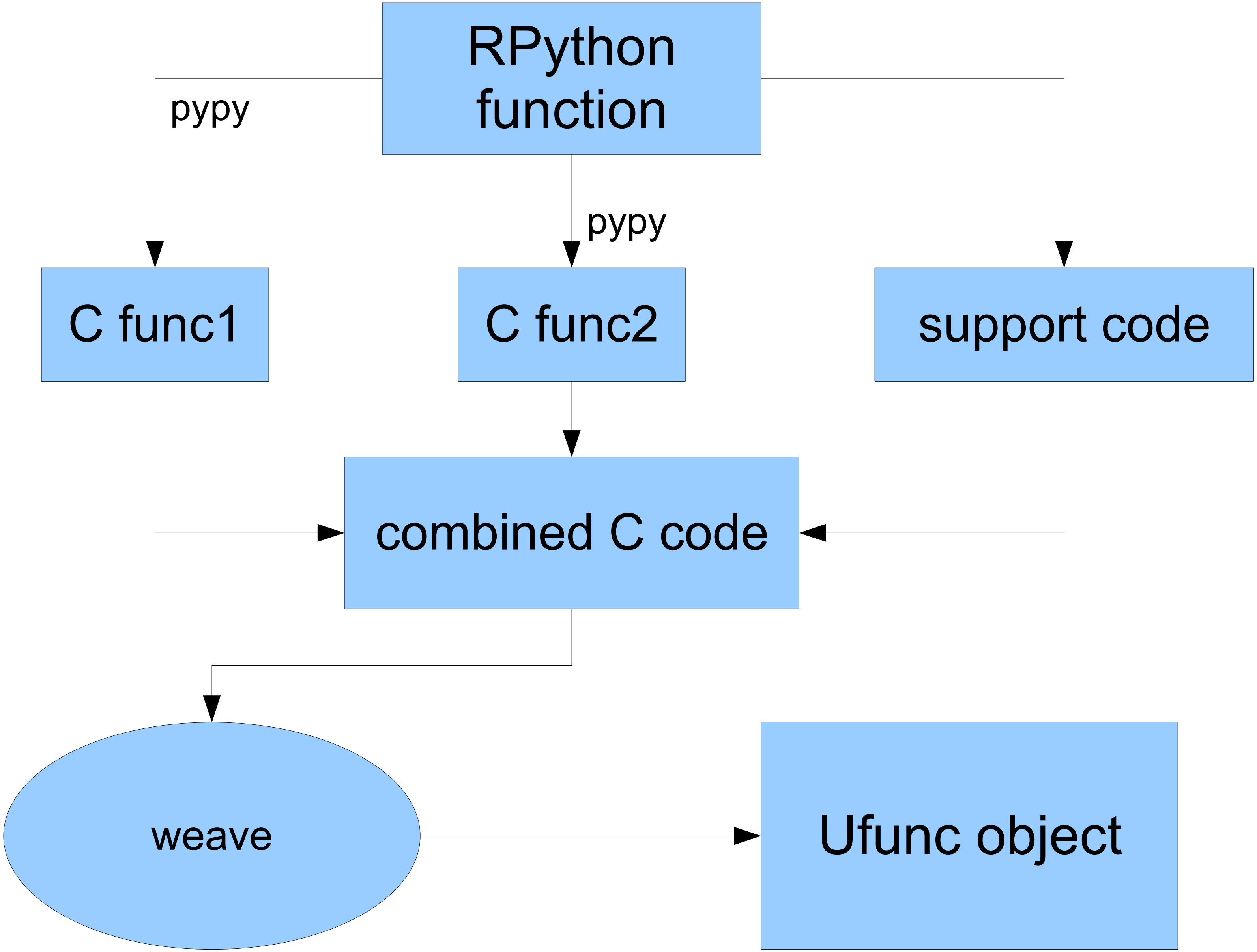
Using the compile decorator

```
from compdec import compdec

@compdec
def is_prime(n):
    if n < 2:
        return False
    for i in xrange(2, n):
        if n%i == 0:
            return False
    return True

print sum(is_prime(n) for n in xrange(100000))
```

easy to use!!!

But no notion of numpy :-(

fast_vectorize



Example

```
>>> from numpy import arange
>>> from ??? import fast_vectorize
>>> @fast_vectorize
... def foo(x):
...     return 4.2 * x*x - x + 6.3
...
>>> a = arange(5)
>>> a
array([0, 1, 2, 3, 4])
>>> foo(a)
array([  6.3,   9.5,  21.1,  41.1,  69.5])
```

Benchmark, example but with an array of size 10Million (2.4Ghz Linux)

Method	Runtime (sec)	Speed vs.
numpy.vectorize	8.674	69.9
x as numpy.array	0.467	3.8
fast_vectorize	0.124	1.0
inline (weave C)		

Using fast_vectorize

```
@fast_vectorize              # will assume float
def foo(x):
    return 3.4 * x + math.sin(x)

@fast_vectorize(int)
def bar(n):
    return n % 7 + 1

@fast_vectorize([(float, float, int)])
def mandel(cr, ci):
    d = 1; zr = cr; zi = ci
    for d in xrange(1, 1000):
        zr2 = zr * zr
        zi2 = zi * zi
        if zr2 + zi2 > 16: return d
        zi = 2.0 * zr * zi + ci
        zr = zr2 - zi2 + cr
    else:
        return -1
```