How our trading platform got 40 times faster by switching to RPython

Simon Burton

Richard Emslie

EWT LLC.

Richard Emslie

EWT

EWT

- small company

EWT

- small company
- traders (football players)

EWT

- small company
- traders (football players) 
- nerds 

EWT

- small company
- traders (football players) 
- nerds 
- trading: stocks, bonds, futures, currency, energy, ... ?

Working at EWT

Working at EWT

- Los Angeles (palm trees and movie making)

Working at EWT

- Los Angeles (palm trees and movie making)
- 6 am phone calls

Working at EWT

- Los Angeles (palm trees and movie making)
- 6 am phone calls
- big monitors

Working at EWT

- Los Angeles (palm trees and movie making)
- 6 am phone calls
- big monitors
- aggressive deployment

Working at EWT

- Los Angeles (palm trees and movie making)
- 6 am phone calls
- big monitors
- aggressive deployment
- bugs cost money (but sometimes they make money)

EWT trading platform

- co-located with the electronic exchange
- receive some 10^5 market updates / second
- can place up to 2500 orders / second

EWT trading platform

The platform...

- big python daemon with many twisted services
- c modules (v. scary)

How to get rapid prototyping *and* fast code ?

EWT trading platform

The platform...

- big python daemon with many twisted services
- c modules (v. scary)

How to get rapid prototyping *and* fast code ?

This year we re-wrote it in RPython.

RPython: example

Binary tree + iterator

RPython

What is it ?
(It’s the 10% of Python that is easy to perform type inference on.)

RPython is GOOD

- Compiler for python (makes fast code)

RPython is GOOD

- Compiler for python (makes fast code)
- Interpreter for C (with python syntax)

RPython is GOOD

- Compiler for python (makes fast code)
- Interpreter for C (with python syntax)
- rctypes interface with external C libraries

RPython is GOOD

- Compiler for python (makes fast code)
- Interpreter for C (with python syntax)
- rctypes interface with external C libraries
  - libc
  - SDL
  - cairo

RPython is GOOD: rctypes

(Easily?) interface to C libraries

- ctypes code generator works well
- build a wrapper "dynamically" before compilation

example....

RPython is BAD

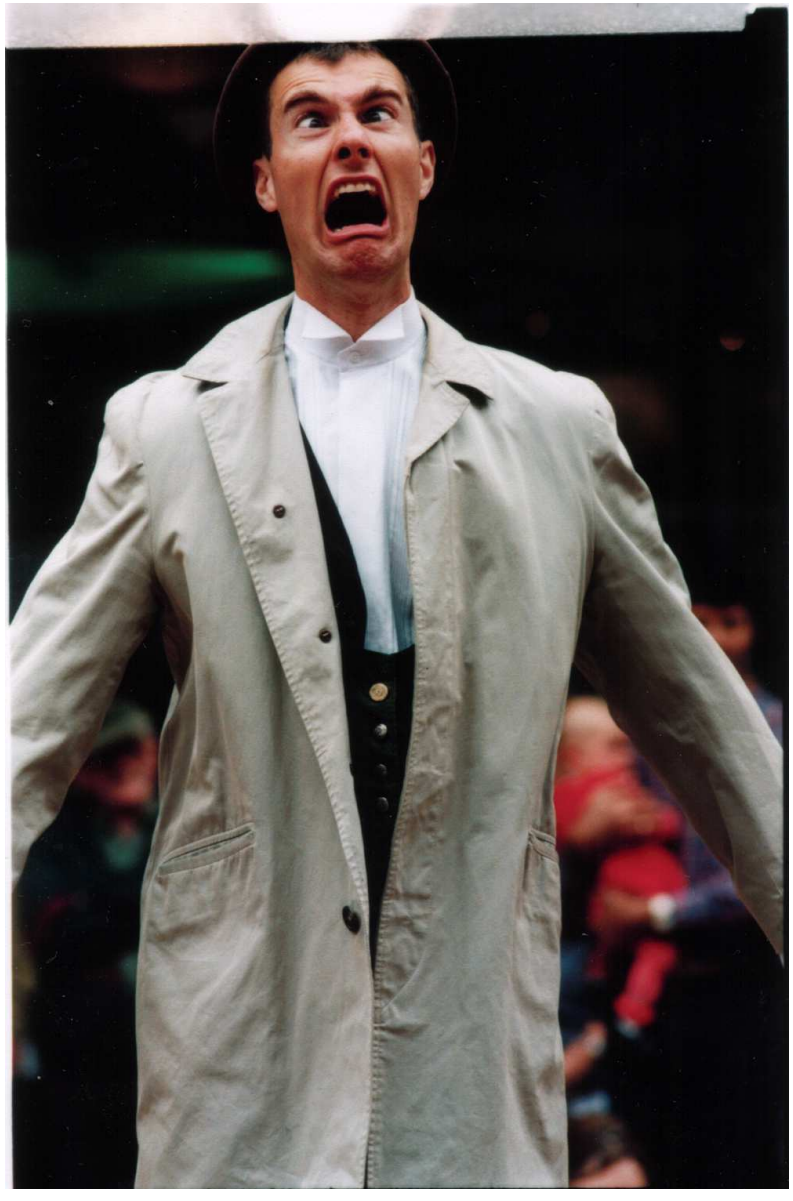
- Confusing, useless, compiler error messages
  - bisection debugging method
- Runtime segfaults (what else did you expect?)

How to deal with this ?

- Debugging with `gdb`

example...

RPython is UGLY



RPython is UGLY

- no special methods (except for `__init__`)
- lack of builtins: enumerate, zip, ...
- lack of modules
- no `long` type
- no list sort
- lots of other stuff you don’t miss until it’s gone

UGLY (?)

```
def bubble(items, lt):
    # NB. We can only use this function with one kind of list
    swapped = True
    while swapped:
        swapped = False
        idx = 0
        while idx + 1 < len(items):
            if lt(items[idx+1], items[idx]):
                items[idx+1], items[idx] = items[idx], items[idx+1]
                swapped = True
            idx += 1
```

UGLY (?)

```
class Compare(object):
    def lt(self, a, b):
        return a < b

class Bubble(object):
    _annspecialcase_ = "specialize:ctr_location"
    def __init__(self, comparator):
        self.comparator = comparator
    def sort(self, items):
        swapped = True
        while swapped:
            swapped = False
            idx = 0
            while idx + 1 < len(items):
                if self.comparator.lt(items[idx+1], items[idx]):
                    items[idx+1], items[idx] = items[idx], items[idx+1]
                    swapped = True
                idx += 1
```

UGLY (?)

```
class Thing(object):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return "Thing(%s)"%self.value

class CompareThing(Compare):
    def lt(self, a, b):
        return a.value < b.value

def main(argv):
    a = [ 1, 9, 0, -33, 22 ]
    aa = [Thing(i) for i in a]
    bubble = Bubble(CompareThing())
    bubble.sort(aa)
    print [x.__str__() for x in aa]

    return 0
```

Annotation Dump

```
__main__.Bubble
comparator : SomeInstance(can_be_None=False, classdef=__main__.CompareTha

__main__.Compare

__main__.CompareThing
lt(
  self : SomeInstance(can_be_None=False, classdef=__main__.CompareThing),
  a : SomeInstance(can_be_None=False, classdef=__main__.Thing),
  b : SomeInstance(can_be_None=False, classdef=__main__.Thing),
): SomeBool()

__main__.Thing
value : SomeInteger(knowntype=int, nonneg=False, unsigned=False)
__str__(
  self : SomeInstance(can_be_None=False, classdef=__main__.Thing),
): SomeString(can_be_None=False)
```

Tricks

- Embedding Python
- Code generation

Tricks: embedding Python

```
def py_mktime(hour, minute, second):
    "NOT_RPYTHON"
    from datetime import datetime
    from time import mktime, time
    now = time()
    tt = datetime.fromtimestamp(now).timetuple()
    tt = tt[:3] + (hour, minute, second) + (0, 0, -1)
    t = mktime(tt)+get_pst_offset()
    return t
```

Tricks: embedding Python

```
def mktime(hour, minute, second):
    from mtt.rlib.cpython import capi as py
    _module = py.PyImport_ImportModule("modulename") # new ref
    ns = py.PyModule_GetDict(_module) # borrowed ref
    _py_mktime = py.PyDict_GetItemString(ns, "py_mktime") # borrowed ref
    _hour = py.PyInt_FromLong(hour) # new ref
    _minute = py.PyInt_FromLong(minute) # new ref
    _second = py.PyInt_FromLong(second) # new ref
    _args = py.PyTuple_New(3)
    py.PyTuple_SetItem(_args, 0, _hour) # steals ref
    py.PyTuple_SetItem(_args, 1, _minute) # steals ref
    py.PyTuple_SetItem(_args, 2, _second) # steals ref
    _t = py.PyObject_CallObject(_py_mktime, _args)
    py.Py_DecRef(_args)
    py.Py_DecRef(_module)
    t = py.PyFloat_AsDouble(_t)
    return t
```

Tricks: code generation

```
src = Source()
block = src.define('foo', 'i')
block.j = (block.i == 42)
block.do_print('foo: i is', block.i)
block.do_print('foo: j is', block.j)
block.do_return(block.j)
src.do_exec(globals())
```

$\longrightarrow$

```
def foo(i):
    j = i == 42
    print 'foo: i is', i
    print 'foo: j is', j
    return j
```

Human Element

The RPython Wall

- porting python code Vs. starting from scratch
- can actually often write nicer code (don’t nead to inline for speed)

Questions?