Mixing Python and C

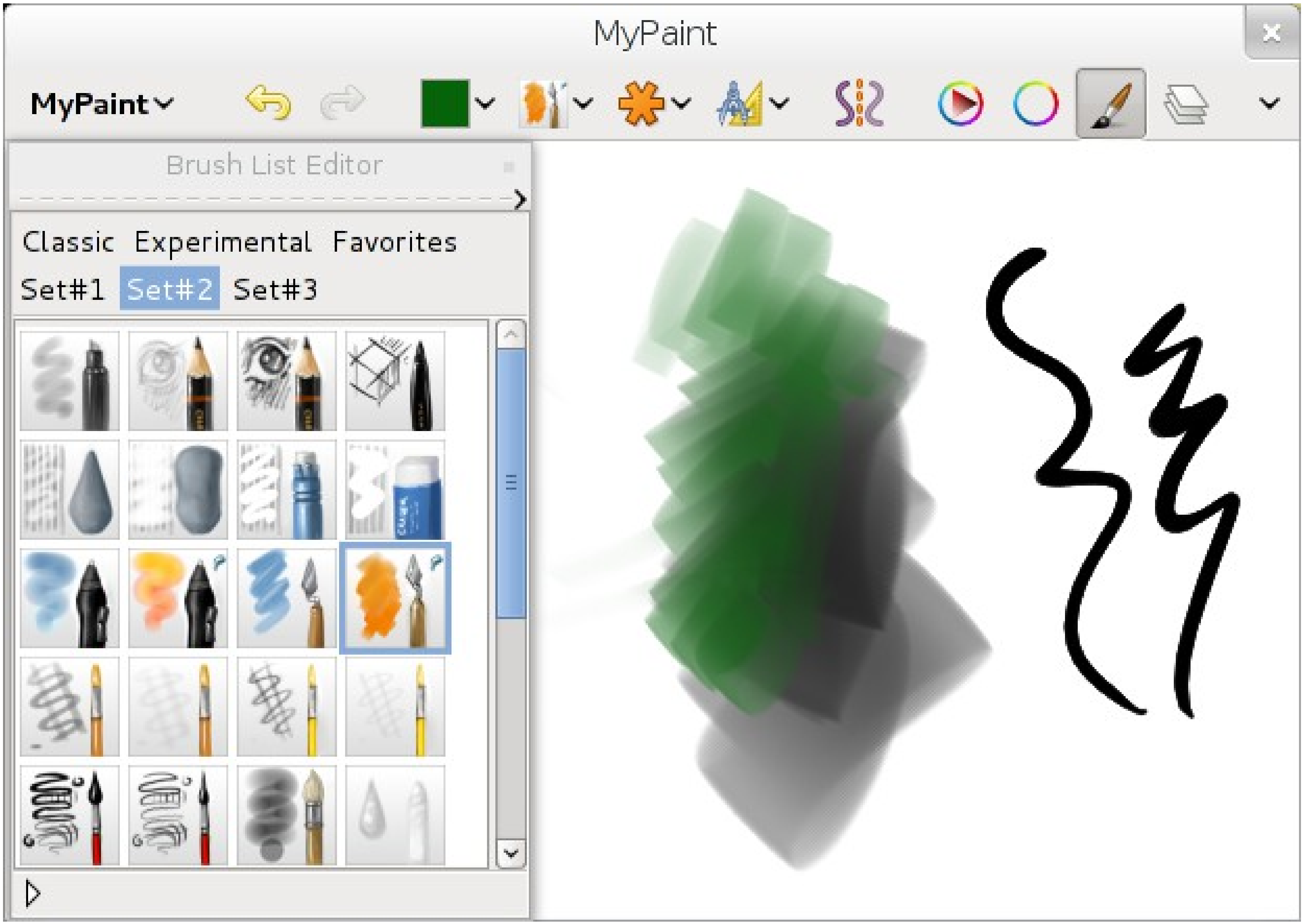
2012-07-10

Martin Renold

Content

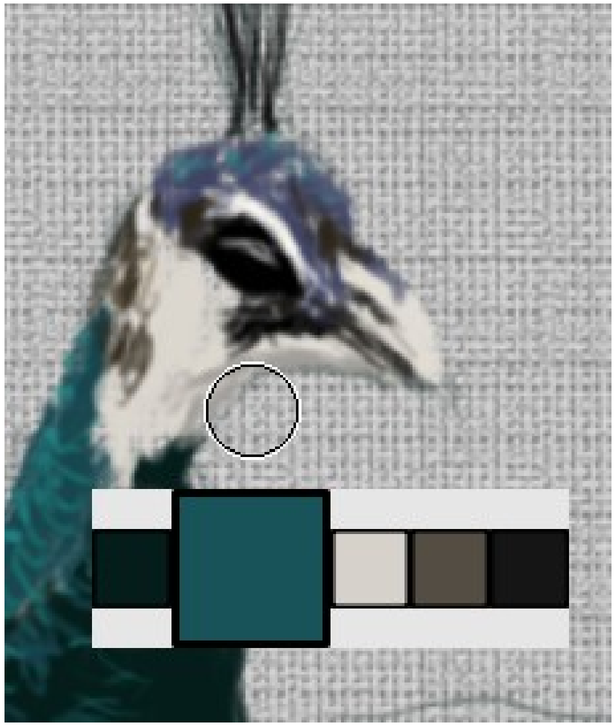
- MyPaint, Python and C
- Profiling (demo)

- Tools to speed up a Python app
- SWIG for minimalists


MyPaint
MyPaint
Brush List Editor
Classic
Experimental
Favorites
Set#1
Set#2
Set#3

MyPaint

- Painting / Sketching
- Easy to use
- Graphic tablets
  - Stylus pressure
  - Subpixel motion



- Related Projects
  - Krita (full digital workflow, more complex)
  - GIMP (main focus is manipulation)

MyPaint

- Code
    - 80% Python, 20% C/C++
    - 25K lines of code
    - Using GTK

- Project
    - Started in 2004
    - Quite popular today

Why Python?

Python

```
for i in items:
    do_something(i)
```

C++

```
for(std::vector<std::string>::const_iterator
    i = items.begin(); i != items.end(); ++i)
{
    do_something(*i);
```

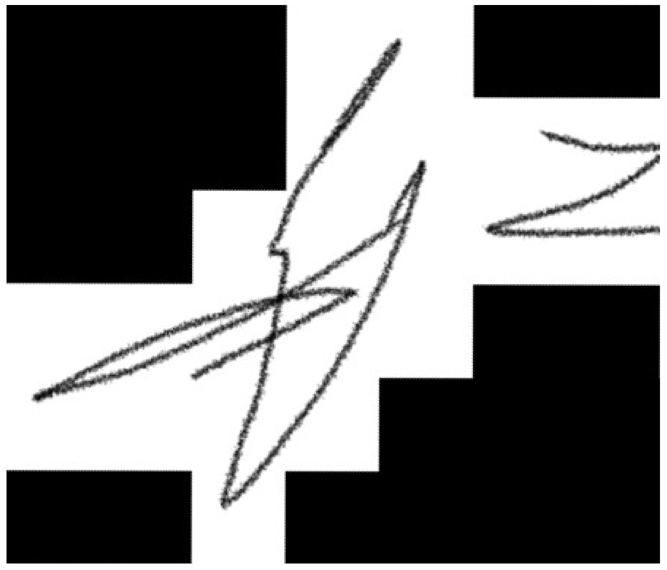
But... Python is slow!

- Press a key, wait
    - 0.0001 seconds, instead of 0.00001 seconds

- 90% of the code is fast enough in any language.
- Now about the 10%...

Fast Enough?

- Python:
  - GUI
  - „for each tile“
  - „for each motion event“



- C/C++:
  - „for each pixel“
  - low-level algorithms (eg. interpolation)

Profiling

- Classical mistake:

  1. Guess what is slow
  2. Optimize the wrong code

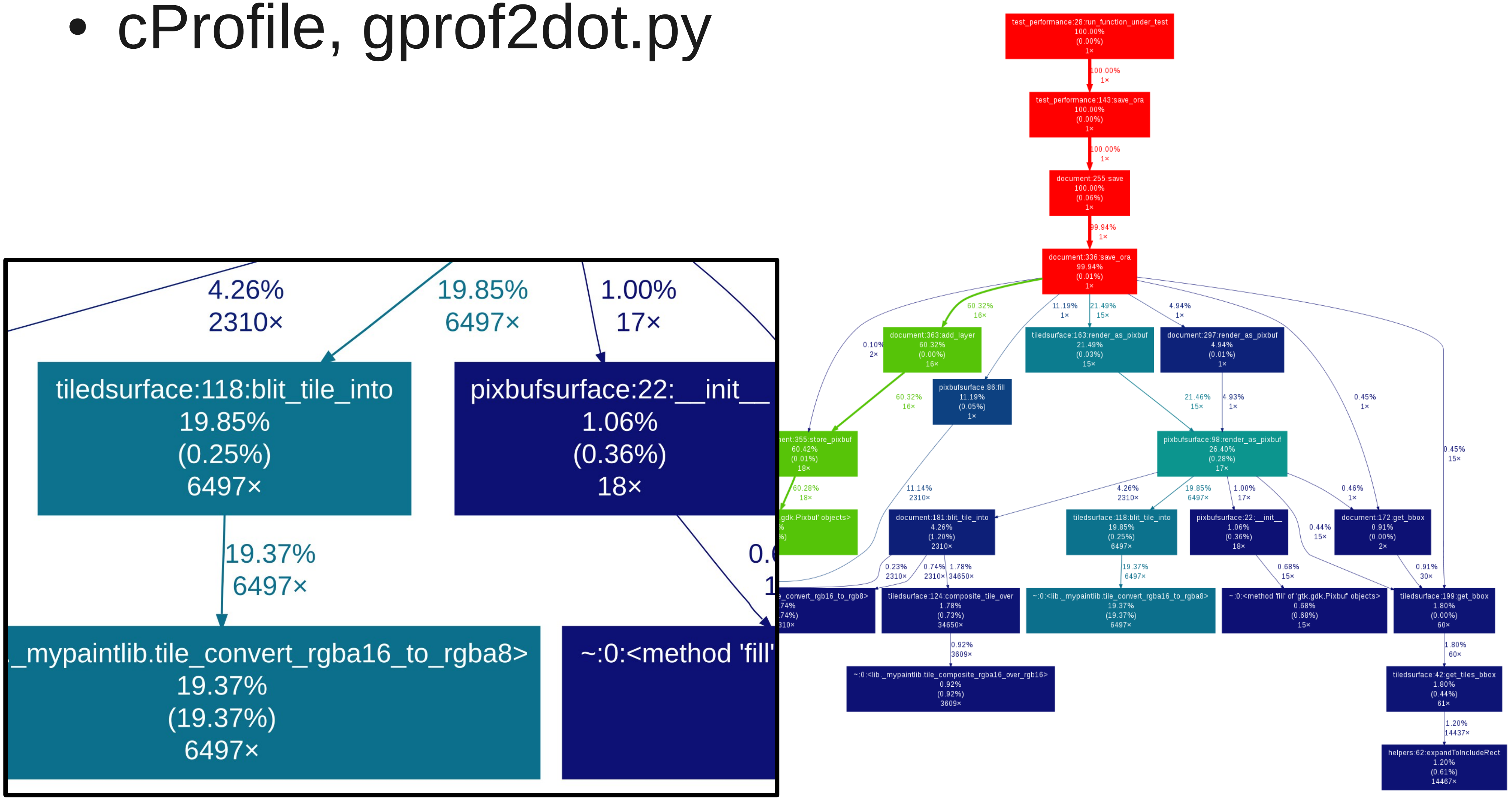
- Measure it!

  --> Tool Demo

  - gprof2dot.py (Python)
  - perf (C, Linux)

Profiling Python

- cProfile, gprof2dot.py



test_performance:28:run_function_under_test 100.00% (0.00%) 1×
test_performance:143:save_ora 100.00% (0.00%) 1×
document:255:save 100.00% (0.06%) 1×
document:336:save_ora 99.94% (0.01%) 1×
document:363:add_layer 60.32% (0.00%) 16×
tiledsurface:163:render_as_pixbuf 21.49% (0.03%) 15×
document:297:render_as_pixbuf 4.94% (0.01%) 1×
pixbufsurface:86:fill 11.19% (0.05%) 1×
pixbufsurface:98:render_as_pixbuf 26.40% (0.28%) 17×
document:181:blit_tile_into 4.26% (1.20%) 2310×
tiledsurface:118:blit_tile_into 19.85% (0.25%) 6497×
pixbufsurface:22:__init__ 1.06% (0.36%) 18×
document:172:get_bbox 0.91% (0.00%) 2×
tiledsurface:124:composite_tile_over 1.78% (0.73%) 34650×
~:0:<lib._mypaintlib.tile_convert_rgba16_to_rgba8> 19.37% (19.37%) 6497×
~:0:<method 'fill' of 'gtk.gdk.Pixbuf' objects> 0.68% (0.68%) 15×
tiledsurface:199:get_bbox 1.80% (0.00%) 60×
~:0:<lib._mypaintlib.tile_composite_rgba16_over_rgb16> 0.92% (0.92%) 3609×
tiledsurface:42:get_tiles_bbox 1.80% (0.44%) 61×
helpers:62:expandToIncludeRect 1.20% (0.61%) 14467×
4.26% 2310×
19.85% 6497×
1.00% 17×
19.37% 6497×

Optimize Saving

- Use libpng directly

- Saving PNG (libpng) too slow?
  - Decrease compression rate
  - Tell libpng not to try all possible filters!

Speeding up Python

Fast code is...	**Tool**
Pure Python	PyPy
Python superset	Cython
Pure C	SWIG | CPython API
C++	SWIG | Boost.Python | SIP
C with GObject	GObject Introspection

SWIG: Code

hello.hpp

```
int answer() {
    return 42;
}
```

hello.i

```
%module hello
%{
#include "hello.hpp"
%}
%include "hello.hpp"
```

SWIG: Compiling

setup.py

```
from distutils.core import setup, Extension

setup(ext_modules=[
  Extension("_hello", ["hello.i"])
])
```

```
$ python setup.py build_ext -i
$ python
>> import hello
>> hello.answer()
42
```

SWIG: The End.

- Do not learn more SWIG!
  - People have died while trying to figure out SWIG Typemaps
- Use the Python/C API
  - SWIG supports this

Python/C API

- Reference Counting
  - Py_DECREF, Py_INCREF macros

```
PyObject * func(PyObject * arg);
```

New Reference

Borrowed Reference

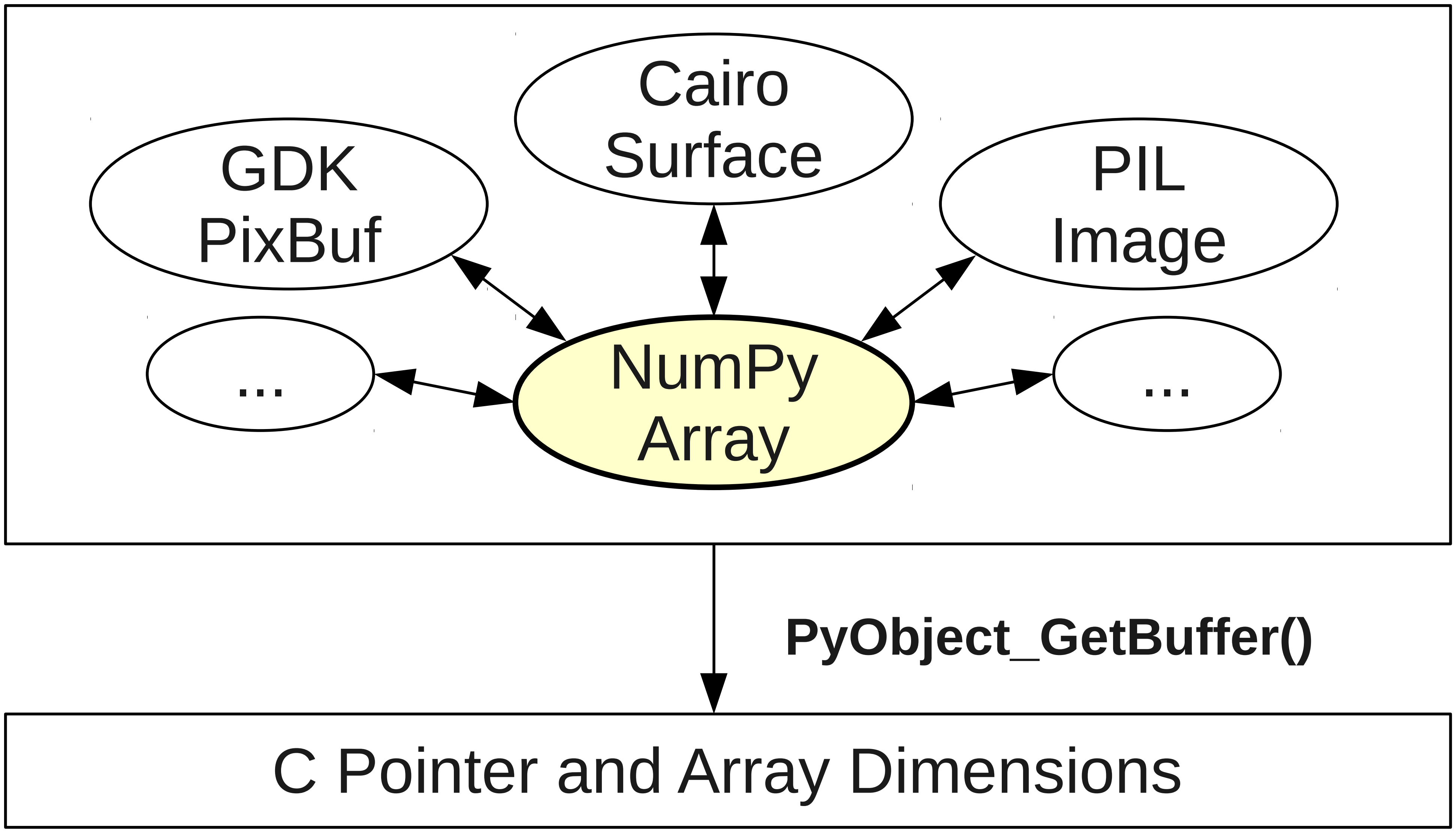
Example

```
class Gradient {
  public:
  float parm1;

  PyObject * get_color(float x, float y) {
    int r, g, b;
    // ...
    return Py_BuildValue("ddd", r, g, b);
  }
};
```

```
>> g = hello.Gradient()
>> g.parm1 = 2.8
>> r, g, b = g.get_color_at(0, 0)
```

Memory Access („Buffer Protocol“)



Debug and Profile

- Like a C/C++ library

```
$ gdb /usr/bin/python
(gdb) run program.py
```

Memory Leaks

- Unused References (common)
  - Hard to find, no tools (?)
- Reference Cycles with __del__
  - check gc.garbage
  - SWIG generates empty __del__ (disable it)
- Missing Py_DECREF (rare)

Thanks

- Code Samples:

`http://github.com/martinxyz/python`

BACKUP

NumPy (and SciPy)

```
from pylab import *

pix = zeros((64, 8, 3), 'uint8')
pix[:,:,0] = 255
pix[:,:,1] = 128 + 60 * randn(64,8)
pix[:,:,2] = 0

imshow(pix)
```