PyROOT

Reference Talk
presented at:

ROOT 2004 Users Workshop, SLAC

LBNL, February 2004

Wim Lavrijsen, LBNL

Outline

- Motivation, history, and context
- Goal
- Overview
  - User-level and examples
  - Design and implementation
- Status, future development
- Resources

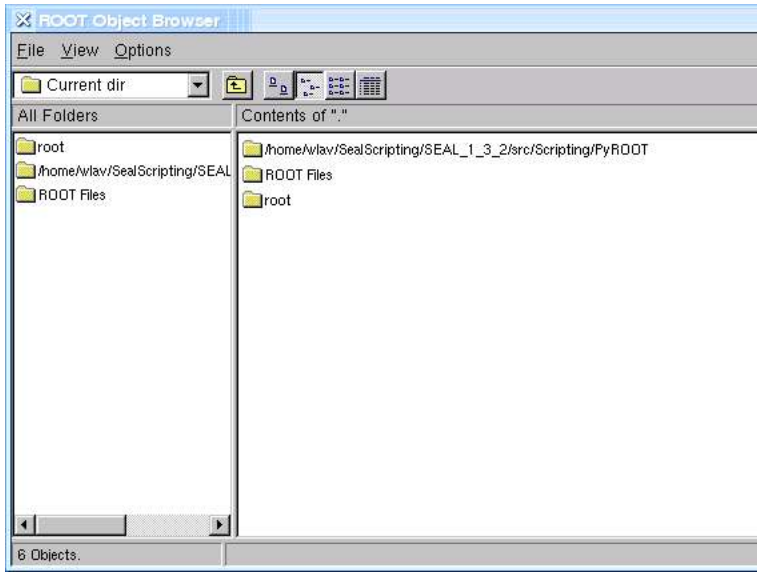
Motivation

- The case for Python:
  - Used for scientific programming (www.scipy.org)
    - Including high performance and distributed parallel code
  - 2nd most popular scripting language
  - Wide-spread use as a "glue language"
- The case for PyROOT:
  - Access ROOT from non-ROOT applications
  - Utilize pre-existing Python bindings from ROOT

Example: pyQt

```
import sys, ROOT
from qt import *
theApp = QApplication( sys.argv )
box = QVBox()
box.resize( QSize(40,10).expandedTo(box.minimumSizeHint()) )
class myButton ( QPushButton ):
   def __init__( self, label, master ):
      QPushButton.__init__( self, label, master )
      self.setFont( QFont( "Times", 18, QFont.Bold ) )
   def browse( self ):
      self.b = ROOT.TBrowser()
bb = myButton( "browser", box )
QObject.connect( bb, SIGNAL( "clicked()" ), bb.browse );
theApp.setMainWidget( box );
box.show();
theApp.exec_loop()
```



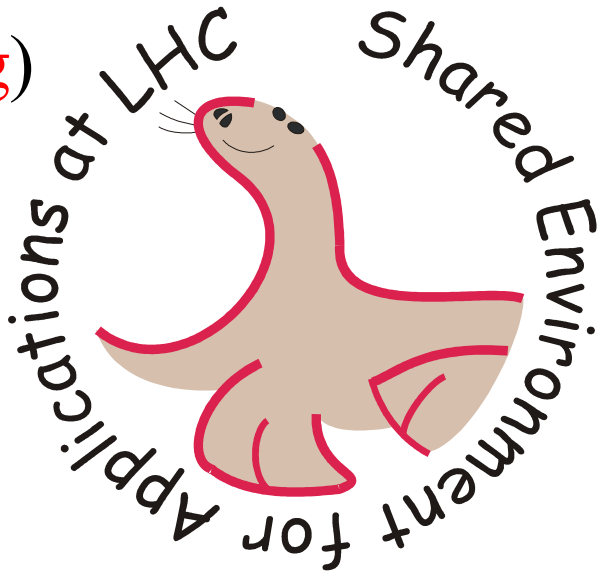


History

- PyROOT saw light as “RootPython”
  - Developed by Pere Mato
  - Resides in the Gaudi (the LHCb framework) repository
- PyROOT inherited ideas, but all code rewritten:
  - Use Python C-API only
  - Closer in style to Python, more generic towards ROOT
  - Large improvement in performance
  - Part of SEAL scripting

SEAL

- Shared Environment for Applications at LHC
- Provide LHC common core and services libraries
  - Foundation class libraries (system, utility, etc. libs)
  - Framework services (includes scripting)
- Improve coherency of LCG apps
- Strategy of pluggable components
  - One such component is **PyROOT**



Goal

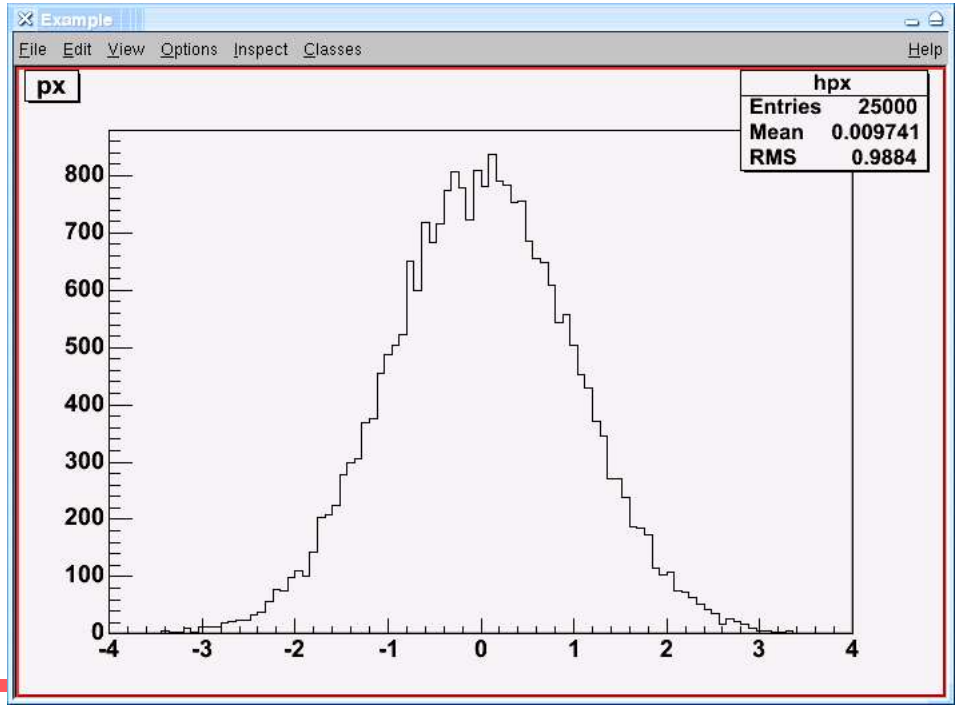
- A “bridge” between ROOT and Python
  - Main library loadable in both CINT and Python
  - Objects can cross interpreters' boundary
    - Note: work in progress ... see later in this presentation
  - Allow Python callbacks from CINT and vv.
- Completely reflection-based
  - No code generation necessary
  - Run-time binding generated as needed

ROOT Features

- ROOT basically uses only ROOT classes
  - Almost all derive from TObject and have a TClass
  - Practically every class name starts with 'T'
  - Simple C++ (no cov. return, templates, fct hiding, etc.)
  - *None of the above is needed, but makes life real easy!*
- Extensive reflection information
  - CINT dictionary
  - Includes signatures, scopes, default arguments, etc.

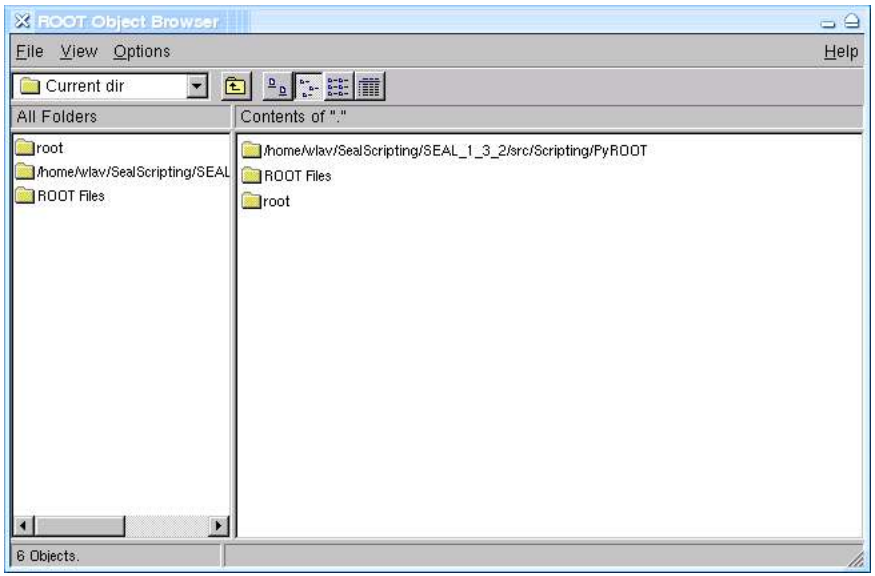
Ex.: ROOT from Python

```
>>> from ROOT import gRandom, TCanvas, TH1F
>>> c1 = TCanvas('c1','Example',200,10,700,500)
>>> hpx = TH1F('hpx','px',100,-4,4)
>>> for i in xrange(25000):
...     px = gRandom.Gaus()
...     hpx.Fill(px)
...
>>> hpx.Draw()
>>> c1.Update()
```

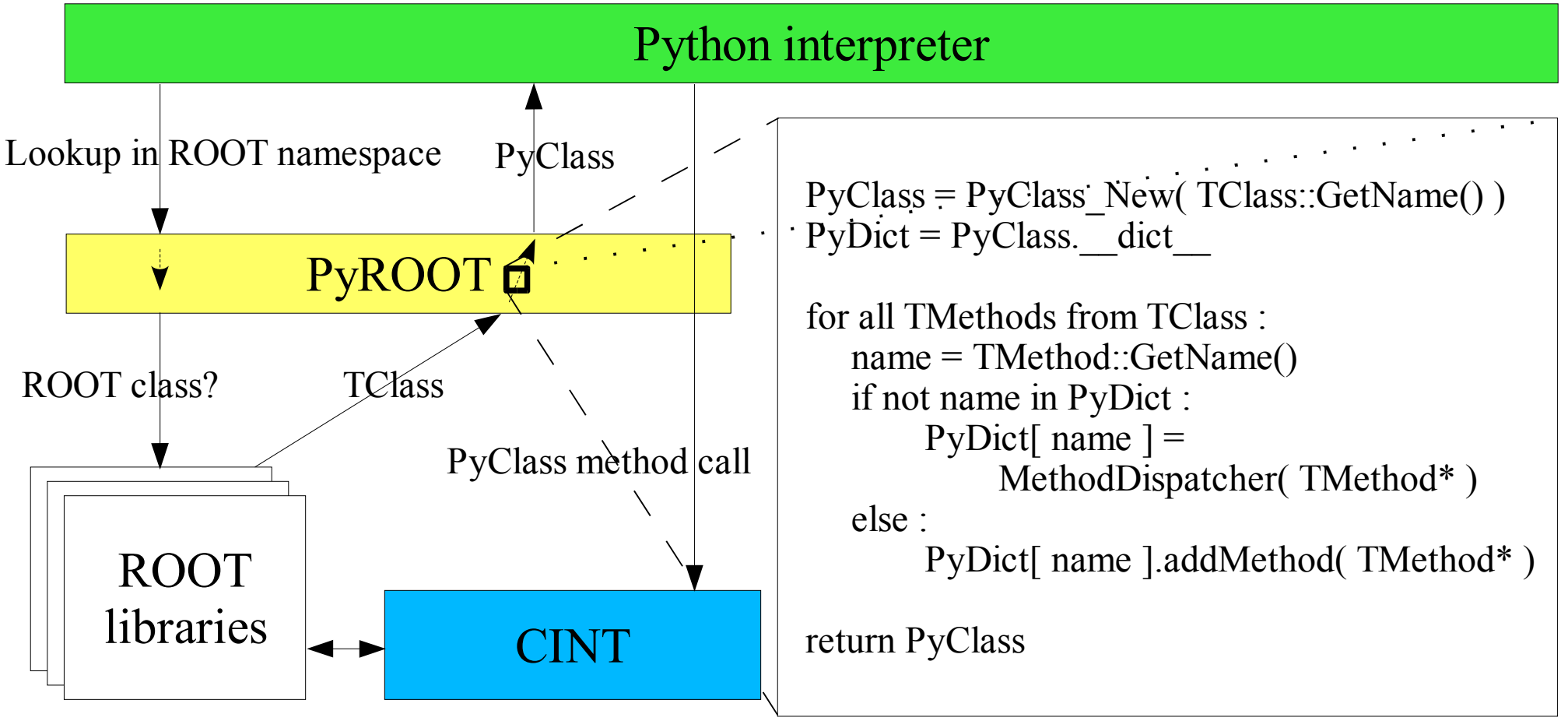


Ex.: Python from ROOT

```
root [0] .L PyROOT.so
root [1] Python::exec( "print 1 + 1" )
2
root [2] b = Python::eval( "TBrowser()" )
(class TObject*)0x8d1daa0
root [3] Python::prompt()
>>> i = 2; ^D
root [4] Python::prompt()
>>> print i
2
```



Conceptual Design - 1



Example in Detail ...

```
>>> from ROOT import TCanvas
   [ import ROOT.py, attempts to retrieve TCanvas ]

>>> c1 = TCanvas('c1','Example',200,10,700,500)
   [ object construction in both Python and ROOT ]

>>> c1.Update()
   [ member call into ROOT from Python ]
```

ROOT.py

- Loads PyROOT.so module
- Exc. hook and module facade to capture names
  - Lookup names by cache or by ROOT ('T', 'g')
  - Instantiated/bound if correctable
  - Re-execute code in the proper stack frame
  - *Transparent to the end-user*
- Fires up a thread to feed system events to ROOT

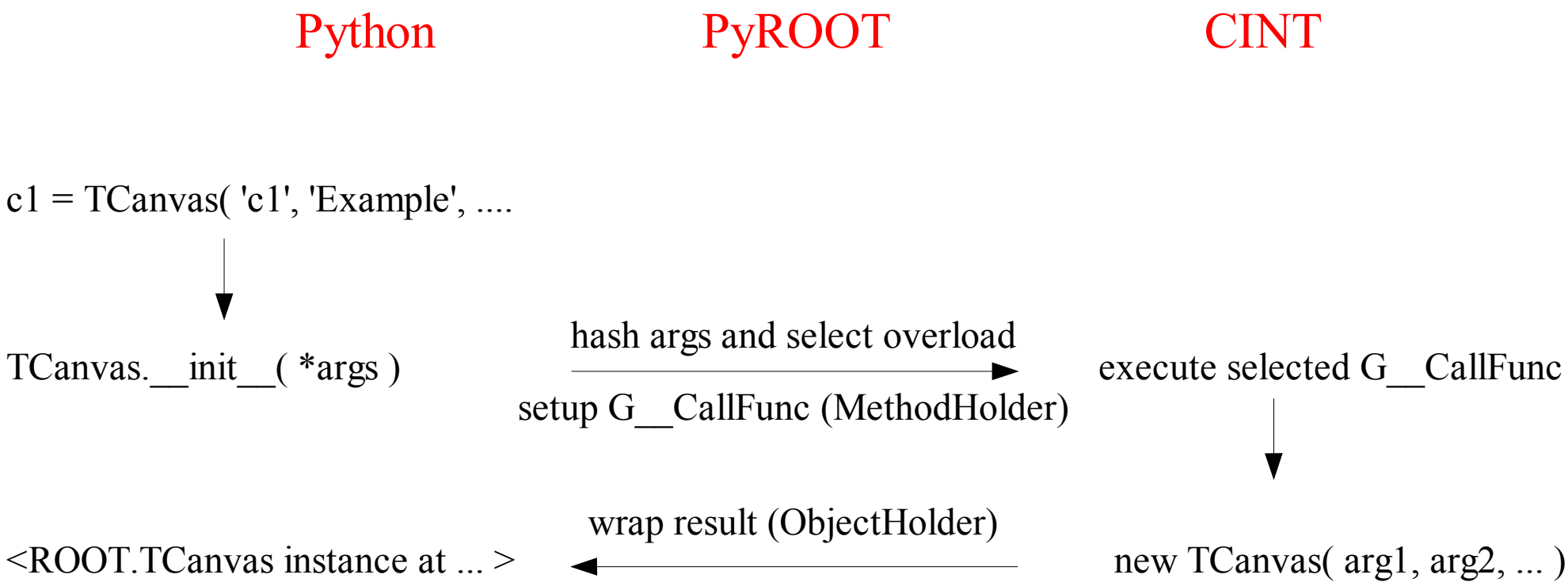
Example in Detail ...

```
>>> from ROOT import TCanvas
   [ import ROOT.py, attempts to retrieve TCanvas ]

>>> c1 = TCanvas('c1','Example',200,10,700,500)
   [ object construction in both Python and ROOT ]

>>> c1.Update()
   [ member call into ROOT from Python ]
```

ConstructorDispatcher



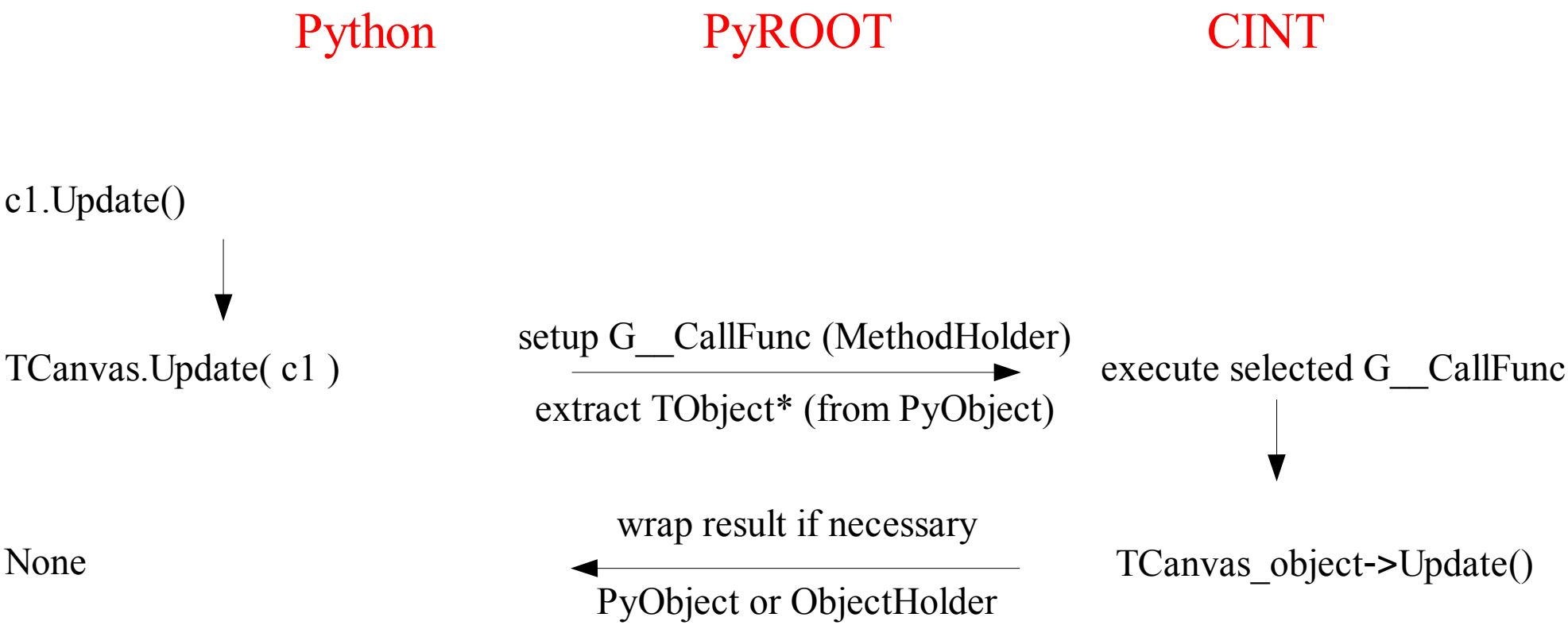
Example in Detail ...

```
>>> from ROOT import TCanvas
   [ import ROOT.py, attempts to retrieve TCanvas ]

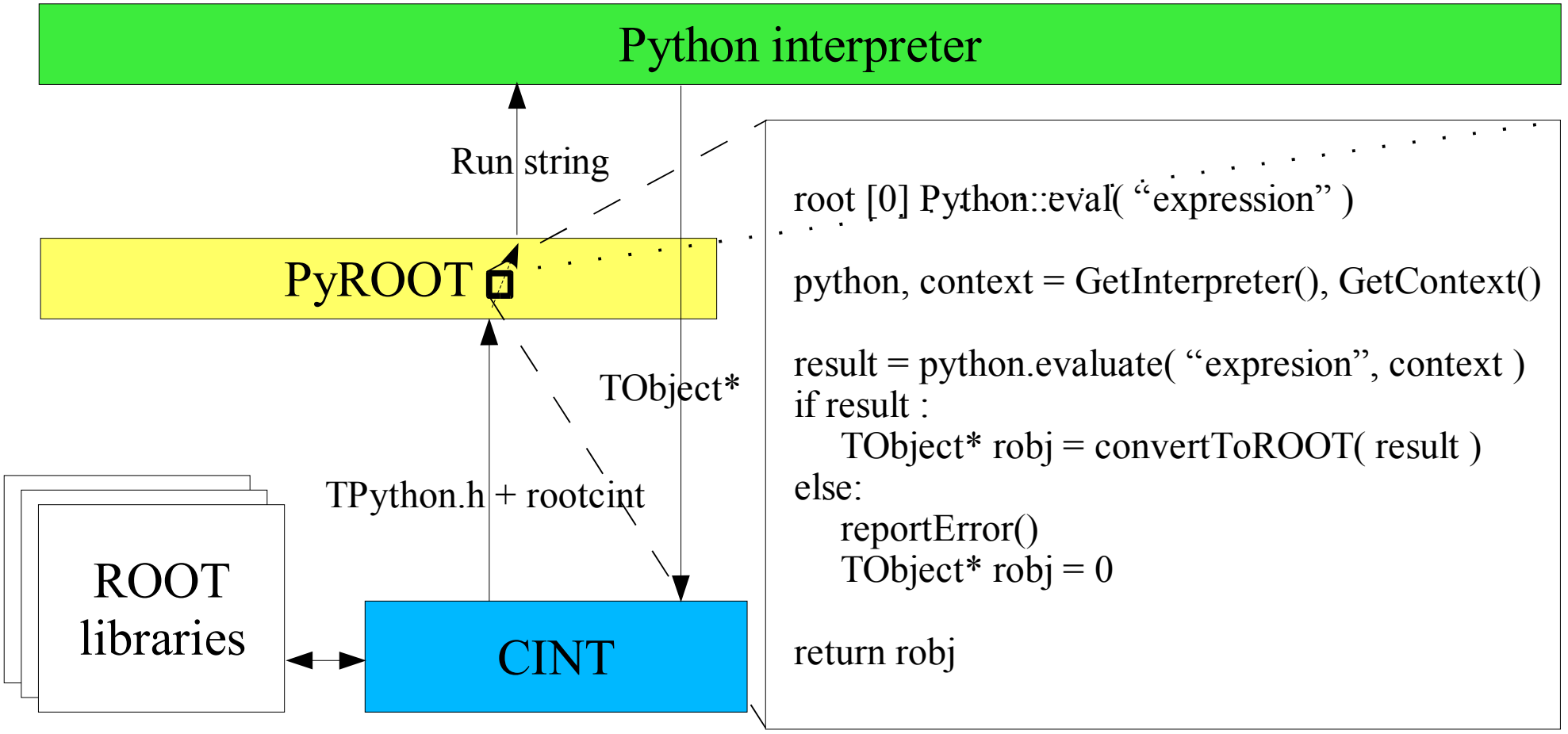
>>> c1 = TCanvas('c1','Example',200,10,700,500)
   [ object construction in both Python and ROOT ]

>>> c1.Update()
   [ member call into ROOT from Python ]
```

MethodDispatcher



Conceptual Design - 2



TPython.h

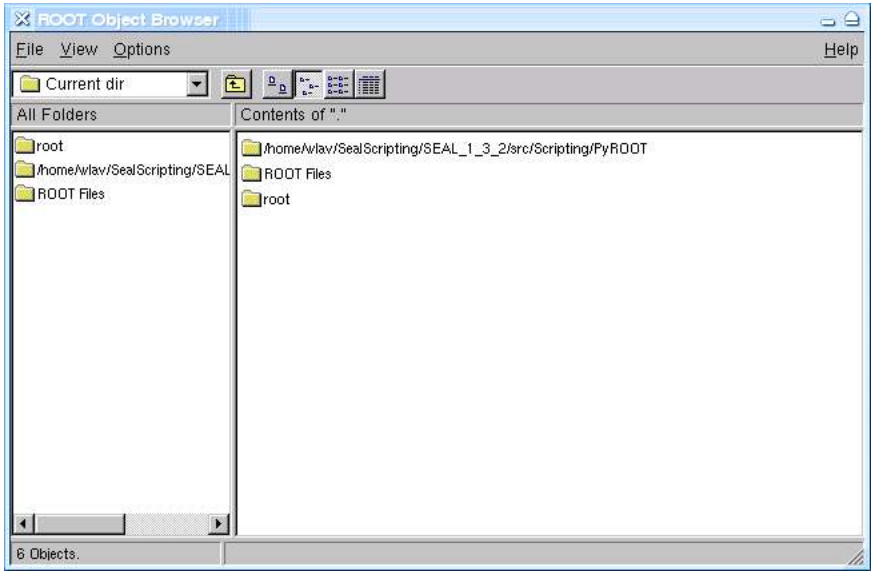
Use rootcint and appropriate LinkDef.h:

```
class Python {
public:
   void exec( char* statement );
   TObject* eval( char* expression );
   bool bind( TObject* obj, char* label );
   void prompt();
// ...
}
```

Callbacks

```
>>> from ROOT import TControlBar, TBrowser
>>> bar = TControlBar('vertical','Demo')
>>> bar.AddButton('browser',
...     r'Python::exec("b = TBrowser()”);',
...     'The ROOT browser')
...
>>> bar.Show()
```





Interpreter Crossing

For now, only for objects w/ CINT dictionary:

```
root [1] TBrowser* b = new TBrowser()
root [2] Python::bind( b, “b” )
(bool)1
root [3] b1 = Python::eval( “b” )
(class TObject*)0x894d3d8
root [4] b == b1
(int)1
```

Status

- ROOT tutorial examples all work
- Performance is good enough

- Python from ROOT is minimal so far
- Little (natural) support for basic ROOT classes
  - TString, TArrayF, etc.
- Some functionality got lost in the move to C-API

PyROOT Resources

- Released as part of SEAL
  - /afs/cern.ch/sw/lcg/app/releases/SEAL
  - SEAL repository: Scripting/PyROOT
- Documentation:
  - http://cern.ch/seal/snapshot/devguide/PyROOT-howto.html
  - http://cern.ch/wlav/scripting/scripting.html#PyROOT
- Examples provided with PyROOT installation

Other Resources

- Python, boost.python bindings
  - http://www.python.org
  - http://www.boost.org/libs/python/doc
- ROOT
  - http://root.cern.ch
- SEAL project portal
  - http://cern.ch/seal