National Energy Research Scientific Computing Center (NERSC)

PyROOT: A CINT — Python Bridge

Wim T.L.P. Lavrijsen
NERSC HENPC, LBNL

ROOT 2005 Users Workshop, CERN - 09/28/05

Outline

- Overview and Timeline
- Design / Implementation Overview
- Ongoing Developments:
  – ROOT, STL “pythonization”
  – Interpreter-side ease-of-use features
  – Low-level C++ access
- Future Plans, Reflex
- Resources

Overview

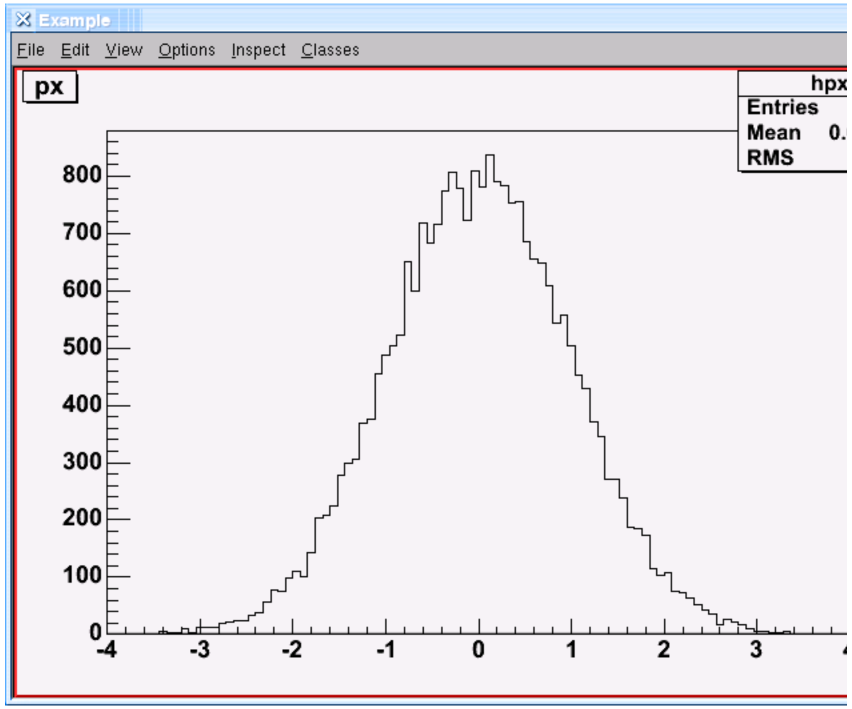
- **Python bindings to ROOT**
  - Python is a very popular dynamic language
  - Every app/lib out there has pybindings
    - Including LHCb/Atlas framework: Gaudi/Athena
- **Two-way: access to Python from RINT**
  - Allow physicists full choice w/o loosing the ability to use eachother's codes
- **Interactive mixing of interpreters**
  - Start Python from CINT and v.v.

Timeline

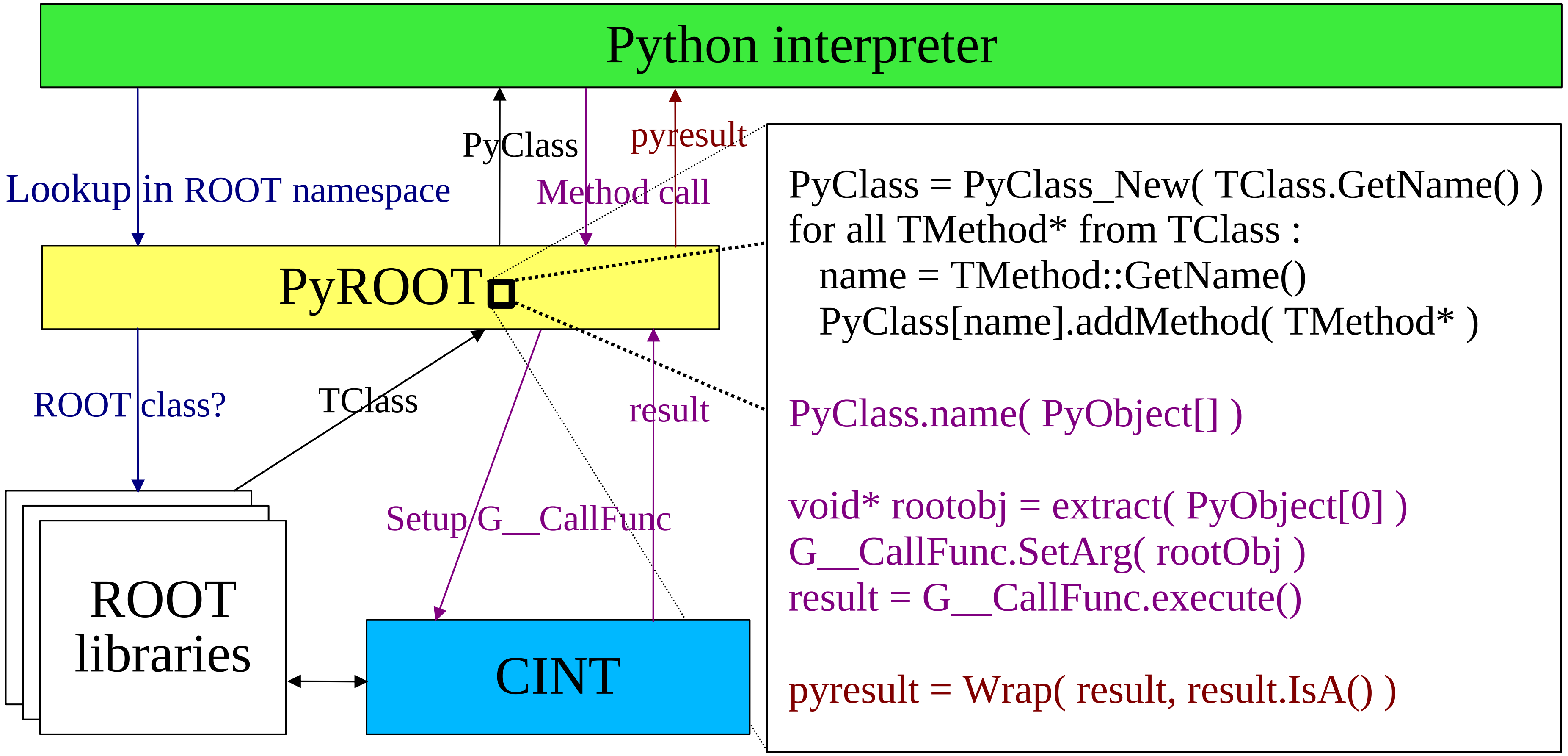
- Pere Mato's RootPython (07/02)
  - Based on boost.python v1, in Gaudi CVS
- Rewrite: PyROOT in SEAL (03/03)
  - Based on boost.python v2
- Python C-API based PyROOT (02/04)
  - Released with ROOT 4.00/04 and later
- Optimized PyROOT (02/05)
  - Current, stable core

Example: 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()
```



Conceptual Design - 1



```
PyClass = PyClass_New( TClass.GetName() )
for all TMethod* from TClass :
   name = TMethod::GetName()
   PyClass[name].addMethod( TMethod* )

PyClass.name( PyObject[] )

void* rootobj = extract( PyObject[0] )
G__CallFunc.SetArg( rootObj )
result = G__CallFunc.execute()

pyresult = Wrap( result, result.IsA() )
```

Example: Python from CINT

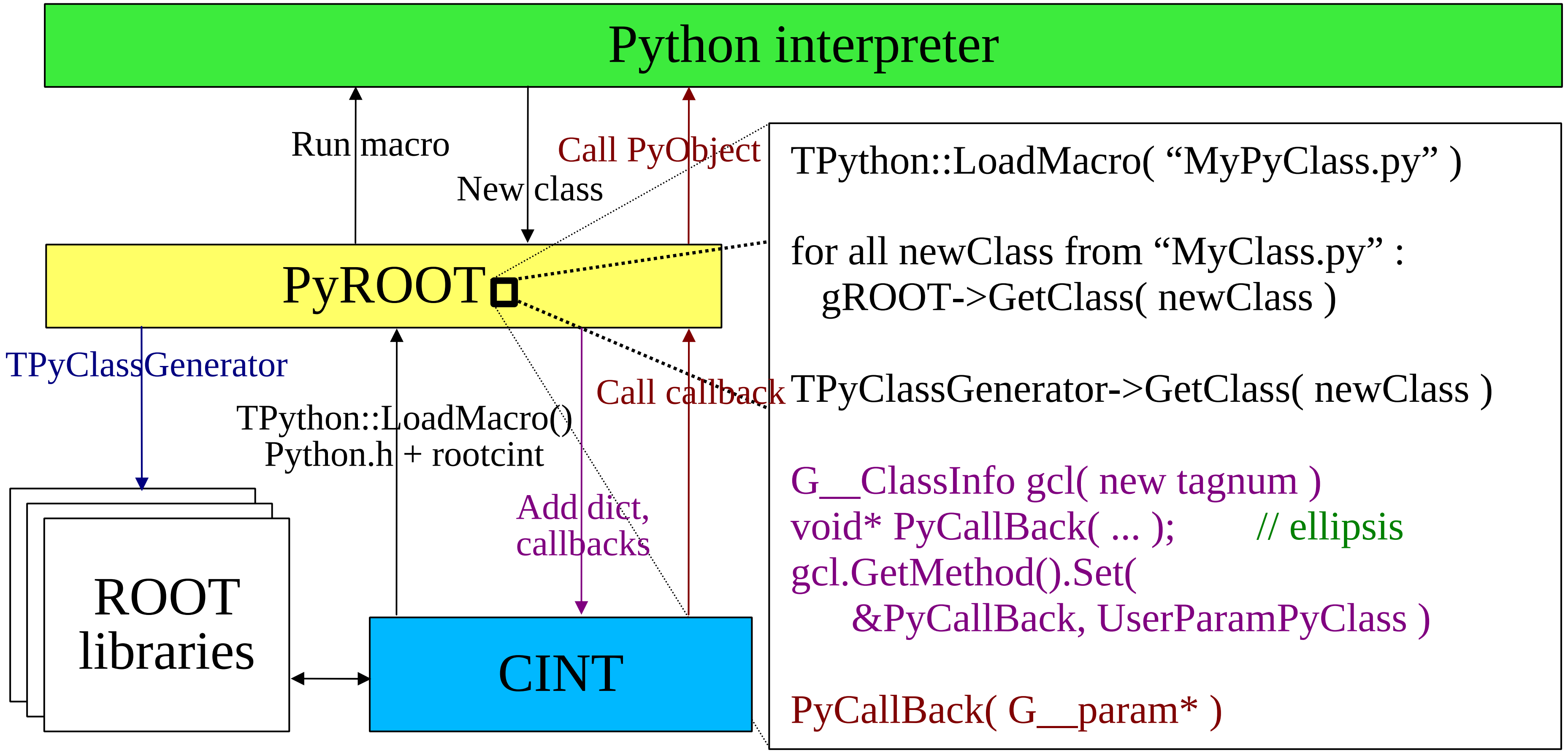
```
class MyPyClass:
   def __init__( self ):
      print 'in MyPyClass.__init__'
   def gime( self, what ):
      return what

root [0] TPython::LoadMacro( "MyPyClass.py" );
root [1] MyPyClass m;
in MyPyClass.__init__
root [2] char* s = m.gime( "aap" );
root [3] s
(char* 0x41ee7754)"aap"
```

Mapping Dynamic onto Static

- Python is dynamic, C++ is static
  - C++: full class spec available before use
  - Python: specs created as-needed
- C++ from Python: flexible and easy
  - Get spec when you want as you want
- Python from C++: may restricted
  - Has to get spec early, but may not exist
  - Really only works for stable modules

Conceptual Design - 2



```
TPython::LoadMacro( “MyPyClass.py” )

for all newClass from “MyClass.py” :
   gROOT->GetClass( newClass )

TPyClassGenerator->GetClass( newClass )

G__ClassInfo gcl( new tagnum )
void* PyCallBack( ... );         // ellipsis
gcl.GetMethod().Set(
      &PyCallBack, UserParamPyClass )

PyCallBack( G__param* )
```

Pythonization

- More than just wrapping, e.g.:
  - Use of TString where PyString expected
  - Use an std::vector in a Python loop
  - Use a Python function with TMinuit
- Often inherited (e.g. TCollection)
  - User classes are also pythonized
- Lots of work still left to do here
  - Users expect it, as it is “natural”
  - Often non-trivial to automate

Example: Pythonization

```
>>> from ROOT import *
>>> f = Tfile( 'staff.root' )
>>> T.GetEvent(0)          # automatically available
45                      #  just like RINT
>>> T.Age                  # 'Age' is a leaf of T ...
58.0
>>> T.Grade                # ... so is 'Grade'
10.0
>>> T.GetEvent(1)
45
>>> T.Age                  # properly updated (erased)
63                      #  for new event
```

Interpreter-side Ease-of-Use

- Python help() shows C++ signatures
  - To be extended (e.g. through THTML)
- Implements RINT-like shortcuts
  - E.g. direct access to any object in a file
  - Commands such as .q, .!, .x, etc.
- Script-safe “from ROOT import *”
  - But comes with a performance penalty that scales linearly with script complexity

Low-level C++ access

- Some C++ has no python equivalent
  - Pointer arithmetic, memory mgmt, etc.
- Preferably, avoid it in the first place
  - If you need it all the time: write C++
- Some new implementations:
  - Dedicated ROOT.NULL object (!= None)
  - MakeNullPointer() for typed NULL
  - Pythonization of TTree::Branch()
  - Settable object ownership

Future Plans, Reflex

- Depends on CINT/Reflex integration
  - Solid integration means little work needed
  - For now, detour based on Cintex
    - Already led to several PyROOT improvements
  - Current design allows multiple execution engines, but not setup for two inputs
- Evaluate importance of separate tool
  - So far, maybe two request (PyG4, PyCool)
  - No Atlas requirement

Resources

- Documentation
  - Chapter 18 of the ROOT User's Guide
  - root.cern.ch/root/HowtoPyROOT.html
  - cern.ch/wlav/pyroot
- Examples
  - $ROOTSYS/tutorials/*.py
- Code Repository
  - root.cern.ch/viewcvs/pyroot