PyROOT Automatic Python bindings for ROOT

Enric Tejedor, Stefan Wunsch, Guilherme Amadio
for the ROOT team

PyHEP 2018
Sofia, Bulgaria

ROOT
Data Analysis Framework

https://root.cern

Outline

- Introduction: What is PyROOT?
- New Features
  - In 6.14: Interoperability with Numpy
  - Coming soon: PyRDataFrame
- Experimental PyROOT
- Future Plans
  - Python 2 & Python 3
  - User Pythonizations
  - Cppyy on Cling

Introduction

ROOT

Data Analysis Framework

https://root.cern

PyROOT

- Python bindings offered by ROOT
- Access all the ROOT C++ functionality from Python
  - Python façade, C++ performance
- Automatic, dynamic
  - No static wrapper generation
  - Dynamic python proxies for C++ entities
  - Lazy class/variable lookup
- Powered by the ROOT type system and Cling
  - Reflection information, JIT C++ compilation, execution
- Pythonizations
  - Make it simpler, more pythonic

A Concrete Example

- Automatic bindings + Pythonizations

```python
import ROOT

f = ROOT.TFile('myfile.root')

t = f.mytree

for event in t:
   ...
```

f is a (dynamic) Python proxy of a C++ object

Pythonization: access tree as an attribute

Pythonization: iterate over tree events in a Pythonic way

Current Status

- The ROOT team has increased the effort in PyROOT
  - We are aware of the importance of Python for HEP!
- Main objective is to improve PyROOT in two ways:
  1. Modernize PyROOT with a new implementation on top of Cppyy
  2. In parallel: consolidate current PyROOT: add new features, fix issues

New Features in 6.14

ROOT

Data Analysis Framework

https://root.cern

New Features

New in 6.14

- Zero-copy C++ to Numpy array conversion
  - Objects with contiguous data (`std::vector`, RVec)
  - Pythonization: tell Numpy about data and shape

```python
import ROOT
import numpy as np

vec = ROOT.std.vector('int')(2)
arr = np.asarray(vec) # zero-copy operation
vec[0], vec[1] = 1, 2

assert arr[0] == 1 and arr[1] == 2
```

Memory adopted!

New Features (II)

New in 6.14

- Read a TTree into a Numpy array
  - Branches of arithmetic types

```
myTree # Contains branches x and y of type float

# Convert to numpy array and apply numpy methods
myArray = myTree.AsMatrix()
m = np.mean(myArray, axis = 0)

# Read only specific branches, specify output type
xAsInts = myTree.AsMatrix(columns = ['x'], dtype = 'int')
```

Forthcoming Features

ROOT
Data Analysis Framework
https://root.cern

Forthcoming Features

- RDataFrame to Numpy
  - All RDataFrame operations available
  - Implicit parallelism

```
from ROOT.ROOT import RDataFrame

df = RDataFrame('myTree', 'file.root')

# Apply cuts, define new columns
df = df.Filter('x > 0').Define('z', 'x*y')

np_arr = df.AsMatrix()
```

JITted C++ expression

Forthcoming Features (II)

- Use Python callables in `RDataFrame`
  - For `Filter` and `Define` operations
  - Implementation with Numba?

```
df = RDataFrame('myTree', 'file.root')

df.Filter('x > 0') # Already possible, jitted C++ expression

def my_cut(x):
 return x > 0

df.Filter(my_cut, ['x']) # Uses Python callable
```

The New PyROOT

ROOT

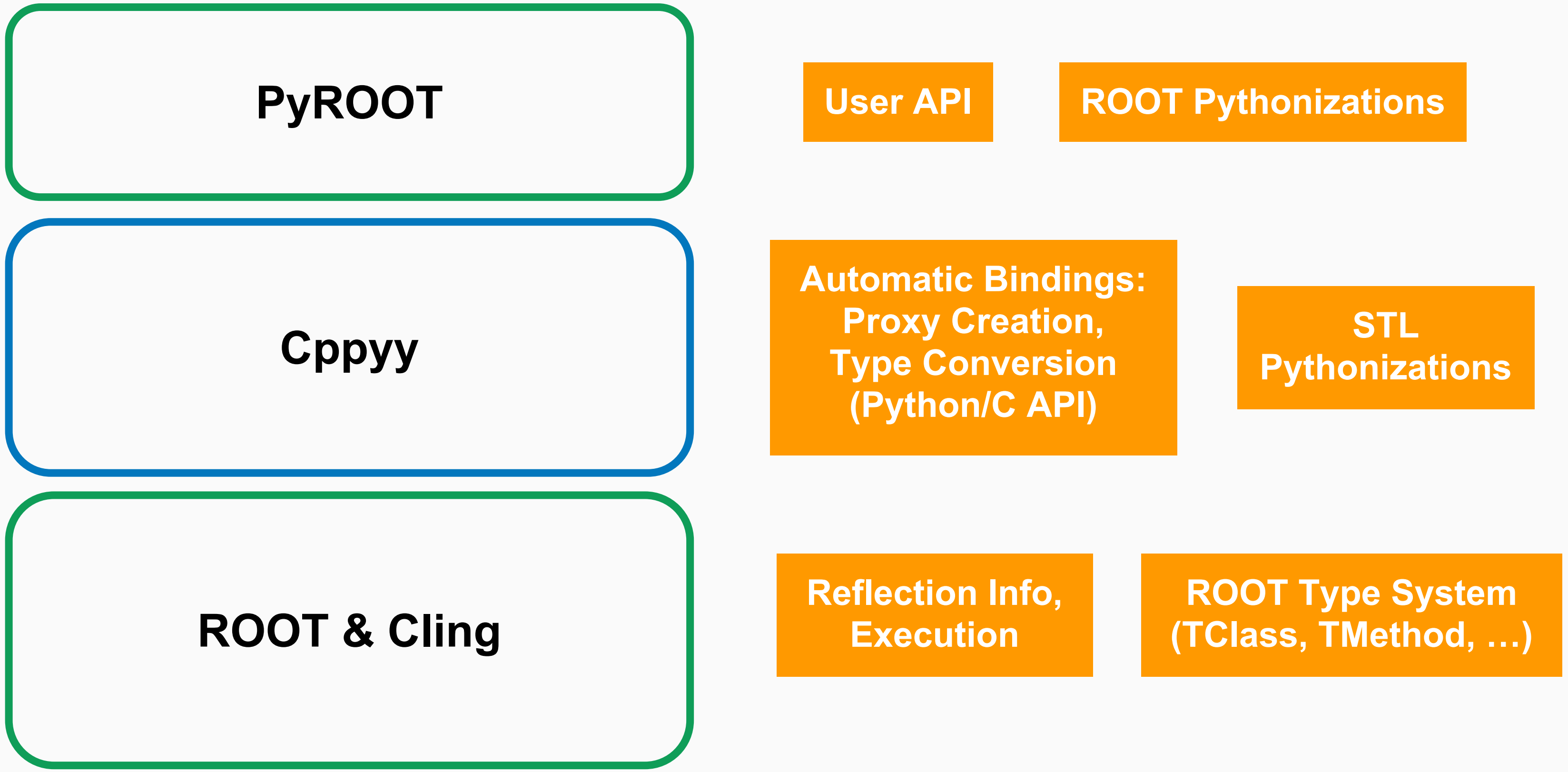
Data Analysis Framework

https://root.cern

The New PyROOT

- A new (experimental) PyROOT implementation is in the making
  - Already available in ROOT master (link)
  - -Dpyroot_experimental=ON
- Based on current Cppyy
  - Set of packages for automatic Python-C++ binding generation
  - Written by Wim Lavrijsen, former PyROOT developer
- Goal: benefit from all the new features of Cppyy
- ROOT-specific Pythonizations added on top
  - A few available at the moment, more will come

The New Structure


PyROOT
User API
ROOT Pythonizations
Cppyy
Automatic Bindings: Proxy Creation, Type Conversion (Python/C API)
STL Pythonizations
ROOT & Cling
Reflection Info, Execution
ROOT Type System (TClass, TMethod, …)

New PyROOT: Lambdas

- Possible to use C++ lambdas from Python

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine(
"auto mylambda = [](int i) { std::cout << i << std::endl; };")
140518947094560L
>>> ROOT.mylambda
<cppyy.gbl.function<void(int)>* object at 0x35f9570>
>>> ROOT.mylambda(2)
2
```

New PyROOT: Variadic Templates

- Support for variadic template arguments of functions

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine("""
template<typename... myTypes>
int f() { return sizeof...(myTypes); }
""")
0L
>>> ROOT.f['int', 'double', 'void*']()
3
```

Future Plans

ROOT
Data Analysis Framework
https://root.cern

Python2 & Python3

- PyROOT supports both versions
  - Also in the new PyROOT
- Not in our plans to discontinue support for Python2
  - At least in the next few years
  - However, end of life for Py2 is very close (2020)
- Building ROOT: we will remove the limitation of one Python version per build
  - If requested, PyROOT libraries will be generated for both Py2 and Py3

User Pythonizations

- Allow users to define pythonizations for their own classes
  - Lazily executed

```
@pythonization('MyCppClass')
def my_pythonizor_function(klass):
    # Inject new behaviour in the class
    klass.some_attr = ...
```

Python proxy of the class

Medium Term: Cppyy on Cling

- Both current PyROOT and Cppyy rely on ROOT meta classes (TClass, TMethod, ...)
  - I.e. reflection data from ROOT
- Not needed: Cppyy could be rebased on top of Cling
  - Use cling and its clang binding directly
  - Access a more powerful API

Summary

ROOT
Data Analysis Framework
https://root.cern

Summary

- PyROOT's automatic Python bindings: unique!
- The ROOT team is aware of the growing importance of Python in HEP
  - Dedicating more effort to PyROOT
- Our goal is to modernize PyROOT
  - Modern C++ with Cppyy, new features
- Pythonizations are key for usability
  - Being tracked here for PyROOT experimental

Backup Slides

ROOT

Data Analysis Framework

https://root.cern

New PyROOT: Move Semantics

- Support for rvalue reference parameters

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine(
'void myfunction(std::vector<int>&& v) {
   for (auto i : v) std::cout << i << " ";
 }')
0L
>>> v = ROOT.std.vector['int'](range(10))
>>> ROOT.myfunction(ROOT.std.move(v))
0 1 2 3 4 5 6 7 8 9
>>> ROOT.myfunction(ROOT.std.vector['int'](range(10)))
0 1 2 3 4 5 6 7 8 9
```