Experimental PyROOT - Heading for 6.22

Enric Tejedor, Stefan Wunsch, Massimiliano Galli

ROOT
Data Analysis Framework
https://root.cern

Outline

- PyROOT - a Quick Overview
- Experimental PyROOT
  - PyROOT for 6.22
  - New Structure
  - Current Status
- Building and Installing PyROOT - MultiPython
  - MultiPython Building
  - Pythonic Installation
- Interoperability with NumPy and Pandas
  - RDataFrame and NumPy
  - From RDataFrame to Pandas
  - From NumPy to RVec

Outline

- More New Features
  - Cpp Callables
  - (Py)ROOT Installation with Conda
- Future Plans
  - Python2 & Python3
  - User Pythonizations
- Summary

PyROOT - a Quick Overview

- Python bindings offered by ROOT
- Access all the ROOT C++ functionality from Python
  - Python usability, 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

Experimental PyROOT

ROOT
Data Analysis Framework
https://root.cern

PyROOT for 6.22

- Most of the current development effort is going to experimental PyROOT
  - Already available in ROOT master (link)
  - -Dpyroot_experimental=ON
  - Goal: switch the default PyROOT to the experimental one for 6.22
- Limited effort put on current PyROOT
  - Fix critical bugs, blockers

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, …)

Current Status

- Interactive graphics ✓
- Pass ROOT tests and tutorials ✓
- Updated to latest Cppyy version ✓
- Multi-version builds ✓ COMING SOON
- Installation in Python directories ✓ COMING SOON
- Interoperability with main Python scientific libraries ✓
- Pass tests from the experiments (e.g. LHCb, PyCool)
- Support user pythonisations
- Documentation

Building and Installing PyROOT - MultiPython

ROOT

Data Analysis Framework

https://root.cern

MultiPython Build

Problem:

- Experiments (and users) have analysis scripts written both in Python 2 and 3
- Two builds (and even more) are time-, CPU- and disk- consuming, as well as useless (most of the libraries are the same)

MultiPython Build

Problem:

- Experiments (and users) have analysis scripts written both in Python 2 and 3
- Two builds (and even more) are time-, CPU- and disk- consuming, as well as useless (most of the libraries are the same)

Solution:

- MultiPython build: rebuild only the libraries which change with the Python version

MultiPython Build

- Easy to build…

```
$ cmake -DPYTHON_EXECUTABLE=/usr/bin/python3.6 ../root
```

… and rebuild:

```
$ cmake -DPYTHON_EXECUTABLE=/usr/bin/python2.7 ../root
```

MultiPython Build

- Easy to source:
  - Pick the version: `$ PYTHON_VERSION=3.6 source bin/thisroot.sh`
  - By default the last one you built is picked:

```
$ source bin/thisroot.sh
```

  - Notifies available versions if a non-existing one is chosen:

```
$ PYTHON_VERSION=3.7 source bin/thisroot.sh
ERROR: build with Python version 3.7 not found.
Available versions:
2.7
3.6
```

MultiPython Build

```
build_dir
└── lib
    ├── python2.7
    └── python3.7
        ├── cppyy
        ├── cppyy_backend
        ├── JsMVA
        ├── JupyROOT
        ├── ROOT
        ├── libcppyy_backend.so
        ├── libcppyy.so
        ├── libJupyROOT.so
        ├── libROOTPython.so
```

- More organized and pythonic structure

Pure Python modules (cppyy, cppyy_backend, JsMVA, JupyROOT, ROOT)

C++ Extension Modules (libcppyy_backend.so, libcppyy.so, libJupyROOT.so, libROOTPython.so)

Pythonic Installation

No need to update PYTHONPATH

- Default Python packages installation scheme:

```
/usr/local/lib/pythonX.Y/site-packages
```

- Custom installation still available:

```
-DCMAKE_INSTALL_PREFIX=/path/to/installation/directory
```

Pull Request

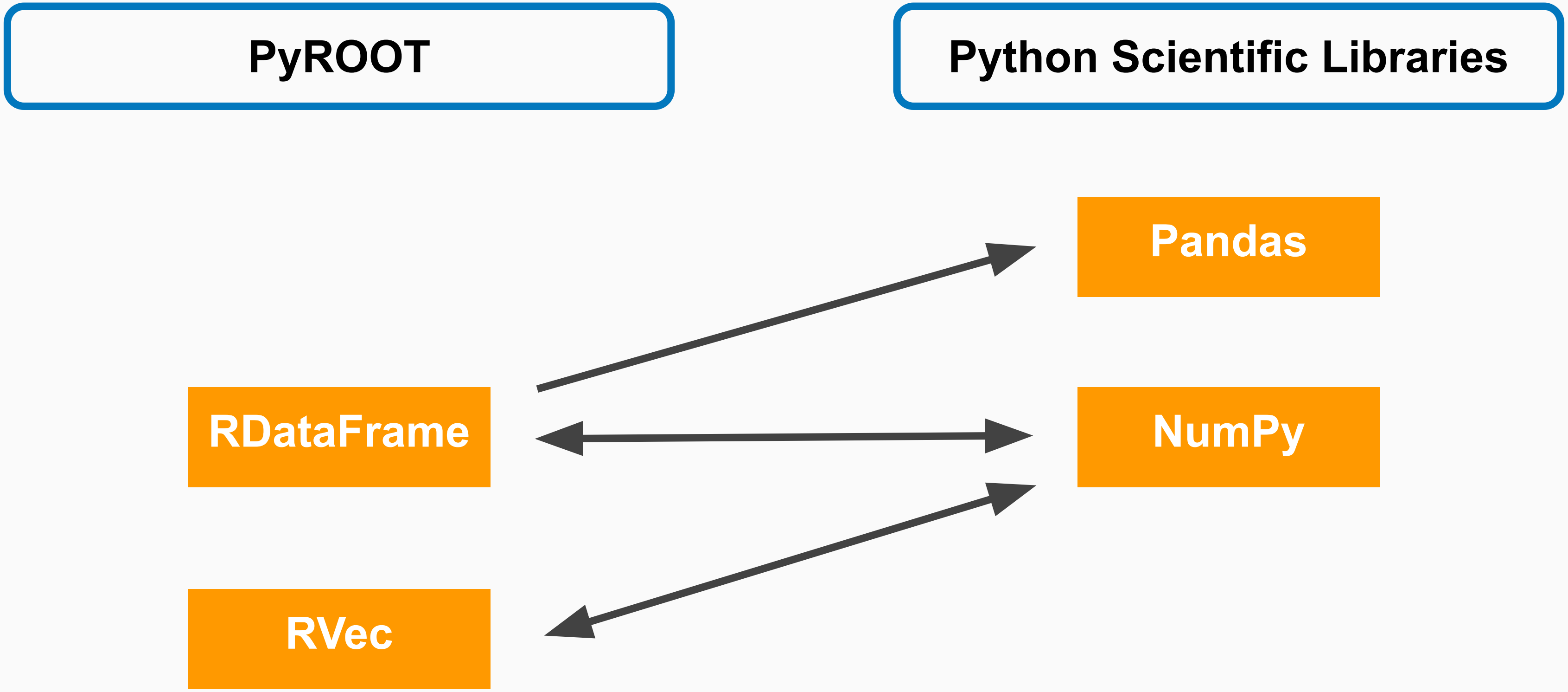
Interoperability with NumPy and Pandas

ROOT

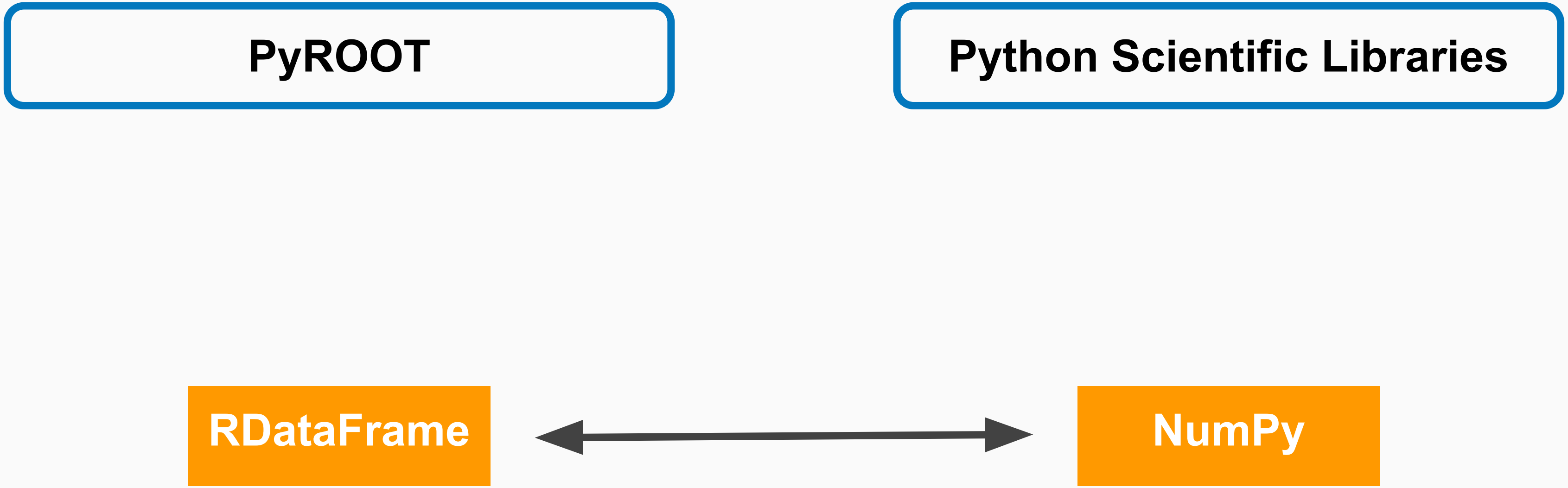
Data Analysis Framework

https://root.cern

Interoperability with NumPy and Pandas



RDataFrame and NumPy



RDataFrame and NumPy

PyROOT

Python Scientific Libraries

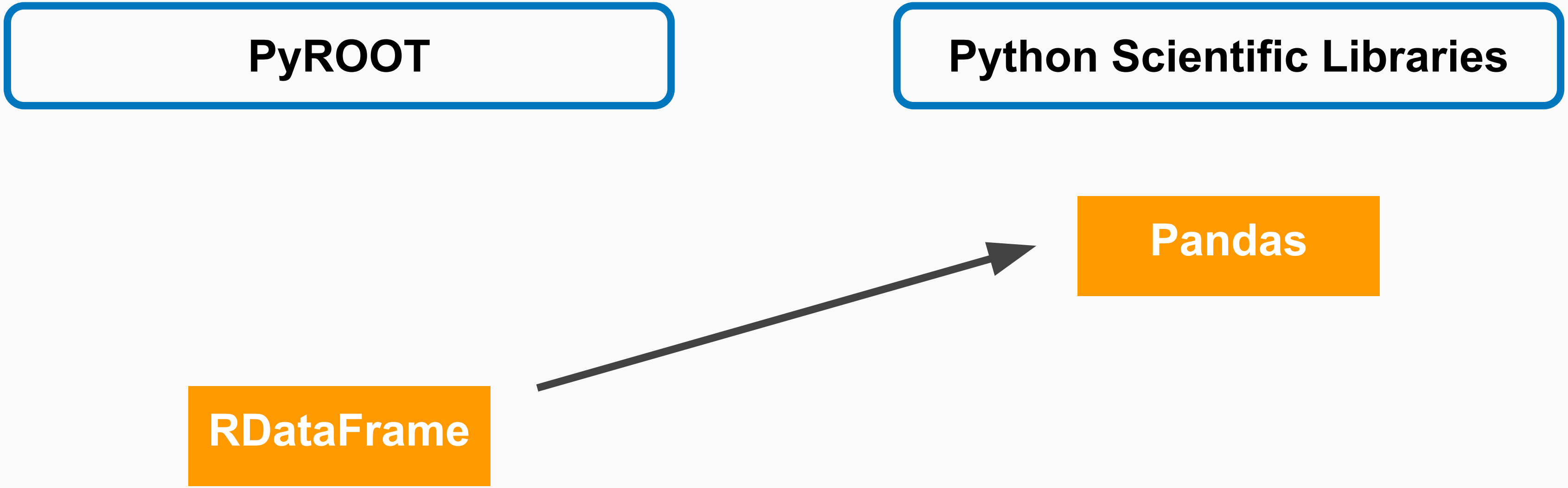
pt_x	pt_y	pt_z	theta
1	0	0	30
0	1	0	60
0	0	1	90

Dictionary of NumPy arrays

```
{'pt_x': ndarray([1.,0.,0.]),
'pt_y': ndarray([0.,1.,0.]),
'pt_z': ndarray([0.,0.,1.]),
'theta': ndarray([30.,60.,90.])}
```

ONLINE DEMO
CLICK HERE

From RDataFrame to Pandas



From RDataFrame to Pandas

PyROOT

Python Scientific Libraries

pt_x	pt_y	pt_z	theta
1	0	0	30
0	1	0	60
0	0	1	90

→

	pt_x	pt_y	pt_z	theta
0	1.0	0.0	0.0	30.0
1	0.0	1.0	0.0	60.0
2	0.0	0.0	1.0	90.0

ONLINE DEMO
CLICK HERE

RVec and NumPy

PyROOT

Python Scientific Libraries

NumPy

RVec

More New Features

ROOT
Data Analysis Framework
https://root.cern

Cpp Callables

```
@ROOT.DeclareCppCallable(["float"], "float")
def f(x):
    return 2.0 * x

df = ROOT.RDataFrame(4).Define("x",
"CppCallable::f(rdfentry_)")
df.AsNumpy()
# Returns {'x': numpy.array([0., 2., 4., 6.],
dtype=float32)}
```

(Py)ROOT Installation with Conda

- New and easy way to install PyROOT and its dependencies
- Currently available on Linux, Mac support underway
- Brief set of instructions:
  - Installing

```
conda create --name myenv --channel conda-forge python=3 root
```

  - Activating the environment

```
conda activate myenv
```

  - Deactivating the environment

```
conda deactivate
```

C. Burr,
E. Guiraud

Future Plans

ROOT

Data Analysis Framework

https://root.cern

Python2 & Python3

- PyROOT supports both versions
  - Also the new PyROOT
- Not in our plans to discontinue support for Python2
  - We are aware of the Py2 end of life
  - But we will go at the pace of the experiments

User Pythonizations

- User Pythonizations: allow ROOT users to define pythonizations for their own classes
    - Lazily executed

Python proxy of the class

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

Summary

- PyROOT 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
- Interoperability with NumPy and Pandas is fundamental

Examples from the demos (references)

From RDataFrame to NumPy

- Even more powerful way to read TTrees into NumPy
  - All `RDataFrame` operations available
  - Optional parallelism

```
>>> from ROOT import RDataFrame
>>> df = RDataFrame('myTree', 'file.root')
>>> # Column dictionary, each column is a NumPy array
>>> df.AsNumpy()
{'pt_x': ndarray([1., 0., 0.]), 'pt_y': ndarray([0., 1.,
0.]), 'pt_z': ndarray([0., 0., 1.]), 'theta':
ndarray([30., 60., 90.])}
```

From RDataFrame to NumPy

- Even more powerful way to read TTrees into NumPy
  - All `RDataFrame` operations available
  - Optional parallelism

```
>>> # Apply cuts, define new columns
>>> df = df.Filter('pt_x > 0').Define('a', 'pt_x*pt_y’)
>>> df.AsNumpy()
{'a': ndarray([0.]), 'pt_x': ndarray([1.]),
'pt_z': ndarray([0.]), 'theta': ndarray([30.])
```

From NumPy to RDataFrame

```python
import ROOT
import numpy

data = {
    "x": numpy.array([1, 2, 3]),
    "y": numpy.array([4, 5, 6])
}

df = ROOT.RDF.MakeNumpyDataFrame(data)
df = df.Define("z", "x + y")

print(df.Mean("z").GetValue()) # Returns 7.0
```

From RDataFrame to Pandas

```
# Run input pipeline with C++ performance that can process TBs of data, reads from remote, ...
df = RDataFrame('tree', 'file.root')
     .Filter('pT_j0>30', 'Trigger requirement')
     .Filter('n_jet >= 2', 'Jet multiplicity cut')
     .Define('r_j0', 'sqrt(eta_j0*eta_j0 + phi_j0*phi_j0)')

# Read out final selection with defined variables as NumPy arrays
col_dict = df.AsNumpy(['r_j0', 'eta_j0', 'phi_j0'])

# Wrap data with pandas
import pandas
p = pandas.DataFrame(col_dict)
print(p)

   r_j0  eta_j0  phi_j0
0  0.26  0.1    -0.5
1  1.0  -1.0     0.0
2  4.45  2.1     0.2
```

From NumPy to RVec

- Convert NumPy arrays to RVecs
  - Pass them into C++ functions
  - Conversion could be done implicitly in the future

```
arr = np.array([1,2,3])
vec = ROOT.AsRVec(arr) # zero-copy operation
my_cpp_fun(vec)
```