Redesign of DeclareCppCallable

How to integrate numba as a first-class citizen

Enrico Guiraud and Stefan Wunsch

ROOT
Data Analysis Framework
https://root.cern

Current status

- We allow to decorate Python callables with `ROOT.DeclareCppCallable`, which jits a wrapper function that can be called from C++
- Main use-case is Python based analysis with `RDataFrame`
- We support a generic implementation or jitting with numba (see next slide)

```python
@ROOT.DeclareCppCallable(["float", "int"], "float")
def pypow(x, y):
    return x**y

ROOT.gInterpreter.ProcessLine('cout << "2^3 = " << CppCallable::pypow(2, 3) << endl;')

data = ROOT.RDataFrame(4).Define("x", "(float)rdfentry_")\
                         .Define("xpow2", "CppCallable::pypow(x, 2)")\
                         .AsNumpy()
```

Numba and generic

- If nothing is specified, e.g., the `numba_only` flag,
  - we try to jit a standalone function with numba (fast and free of locks)
  - otherwise warn the user and fall back to a generic wrapper code that calls directly into the Python interpreter (slow and protected by locks / GIL)

```python
# This is jitted with numba, enforced by the flag (no fallback to the generic wrapper)
@ROOT.DeclareCppCallable(["float", "int"], "float", numba_only=True)
def pypow(x, y):
    return x**y

# This falls back to the generic wrapper calling into the Python interpreter
@ROOT.DeclareCppCallable(["vector<float>", "int"], "float")
def pypowsize(vec, y):
    return vec.size()**y
```

Where we would like to improve

- **Allow to treat RVecs as numpy arrays**
  - Proof of concept made by Enrico
  - Allows to jit the Python callable ...
  - ... thanks to some numba magic
- **Make numba a first-class citizen**
  - The usage of numba and the fact that the code is very efficient is not visible
  - Currently the usage of numba is hidden behind an invisible logic
- **Protect users from using inefficient code generated by the generic wrapper**
  - The feature will be misused!
  - Do we really want to allow this?
- **Proposal**
  - Enhance the Numba approach to RVecs
  - Covers most use-cases in Python based analysis (NanoAOD, analysis ntuples, ...)
  - **Clean, simple, efficient**

```python
# Decorator only using numba
# - Allows to use fundamental types and RVecs thereof
# - No fallback to any generic and inefficient implementation
# - Add the feature in the Numba namespace of the ROOT module
# - The types of the arguments in the function are now
#   Python/Numpy arrays or fundamental types

@ROOT.Numba.DeclareCppCallable(["RVec<float>", "int"], "float")
def pysumpow(x: numpy.ndarray, y: int):
    return numpy.sum(x)**y
```