Automatic Interoperability Between C++ and Python

Baidyanath Kundu, Vassil Vassilev, Wim Lavrijsen

Former RSE @ CERN & Princeton | *CERN & Princeton* | *LBL*

[Compiler Research](#)

Cppyy

Cppyy is an automatic C++ - Python runtime bindings generator which supports a wide range of C++ features.

C++ code (MyClass.h)

```
struct MyClass {
  MyClass(int i) : fData(i) {}
  virtual ~MyClass() {}
  virtual int add(int i) {
    return fData + i;
  }
  int fData;
};
```

Python Interpreter

```
>>> import cppyy
>>> import cppyy.gbl as Cpp
>>> cppyy.include("MyClass.h")
>>> class PyMyClass(Cpp.MyClass):
...  def add(self, i):
...    return self.fData + 2*i
...
>>> m = Cpp.MyClass(1)
>>> m.add(2)
3
>>> m = PyMyClass(1)
>>> m.add(2)
5
```

Cling/Clang-REPL

Cling is an interactive C++ interpreter, built on the top of LLVM and Clang libraries.

Clang-REPL can be thought of as a generalization of Cling in LLVM.

Numba

Numba is a JIT compiler for a subset of Python code. It works best with NumPy arrays and loops.

```python
from numba import jit
import numpy as np

x = np.arange(100).reshape(10, 10)

@jit(nopython=True)
def go_fast(a):
    trace = 0.0

    for i in range(a.shape[0]):
        trace += np.tanh(a[i, i])

    return a + trace

print(go_fast(x))
```

Motivation

Can we make cppyy faster and lighter?

Original design

Disadvantages of using ROOT/meta in Cppyy:

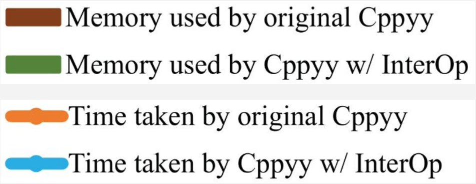
- Performance penalty from its abstraction
- Difficult to extend
- Hard to evolve reflection interfaces

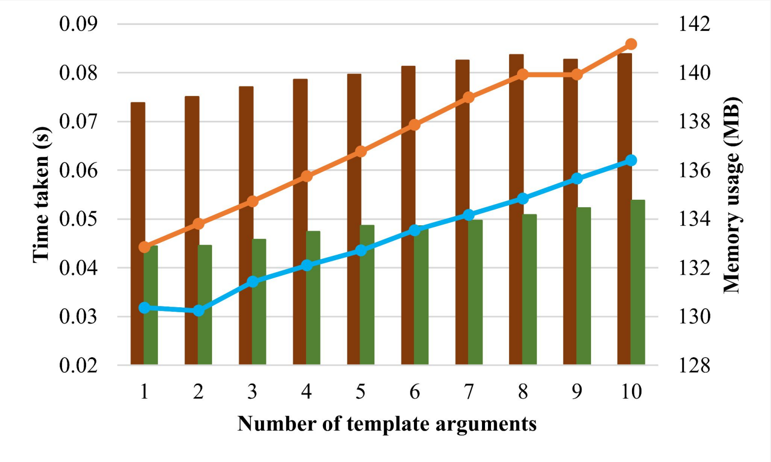
Goal

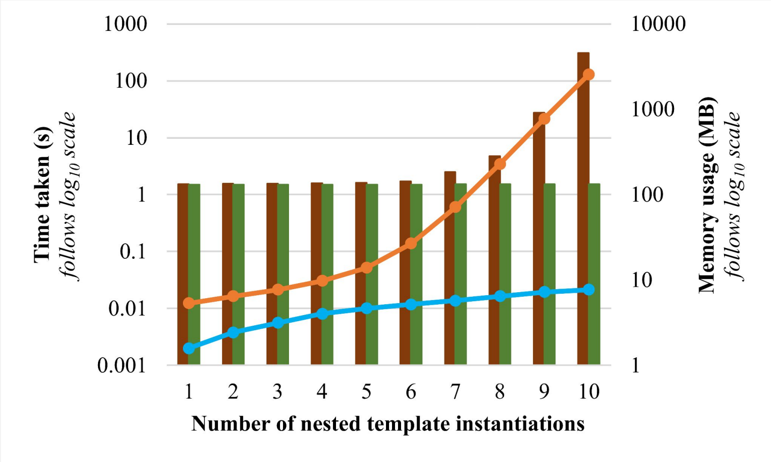
Our Design

Our goal was to rebase Cppyy on top of pure LLVM to address the disadvantages. Thus we created a thin layer on top of the interpreter, called **CppInterOp**, to provide easy to use interfaces for reflection information. This will eventually be a part of upstream LLVM.

Benefits (Measured)





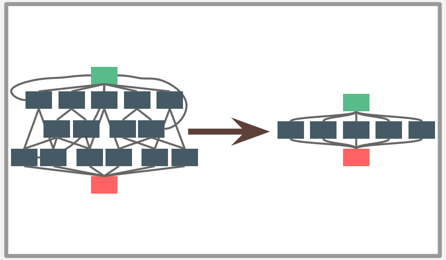


Time taken and memory used during class template instantiation

Cppyy with CppInterOp is about twice as fast in instantiating templates and this holds true when we increase the number of template arguments as well

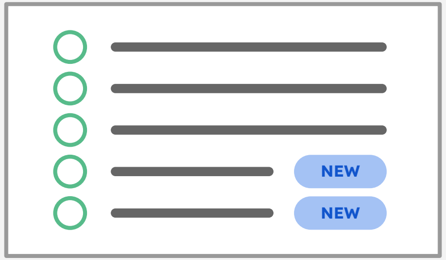
Cppyy with CppInterOp scales better for nested template instantiations when compared to Cppyy with ROOT/meta

Benefits (Unmeasurable)

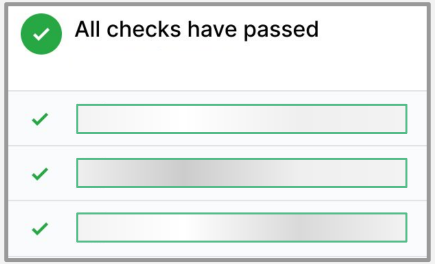


Simpler codebase

LLVM umbrella

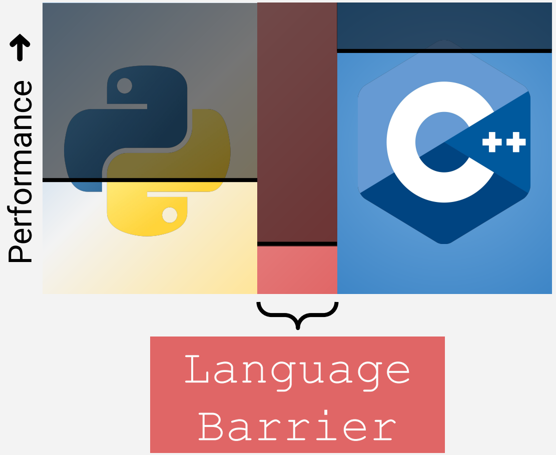


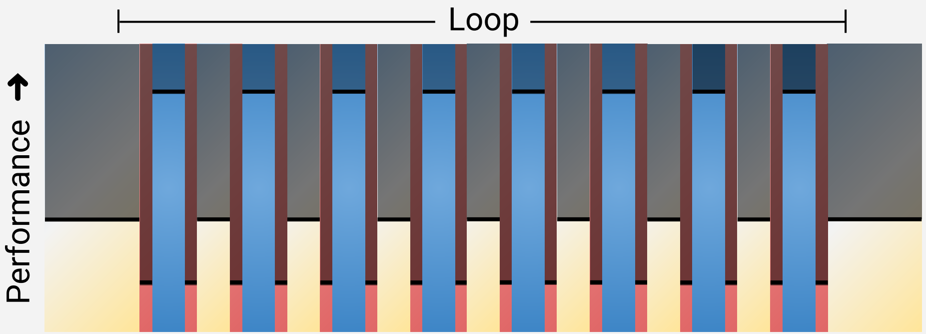
Better C++ feature set support



Well tested interoperability layer

What else is there to be optimize?

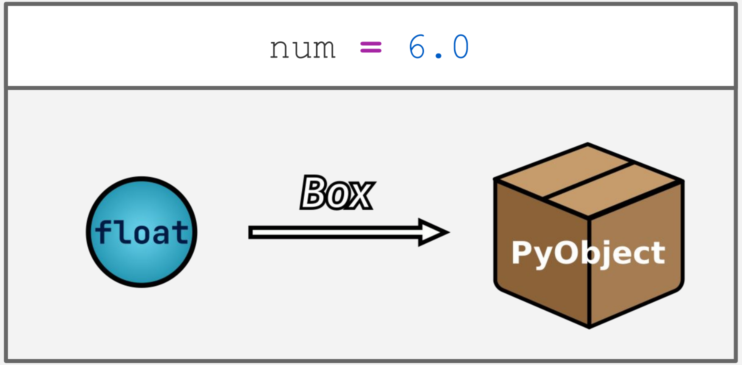


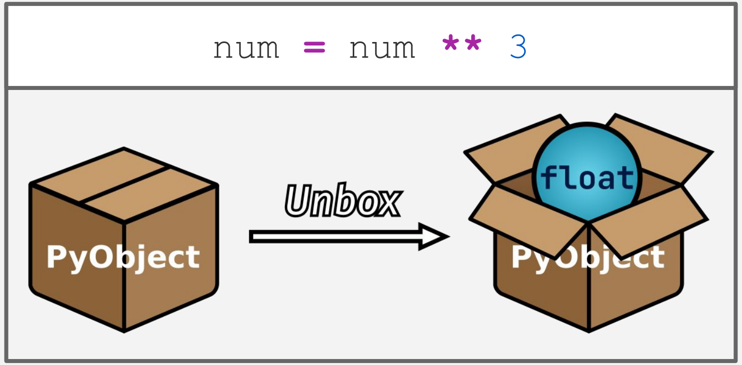


Usual usage with Cppyy

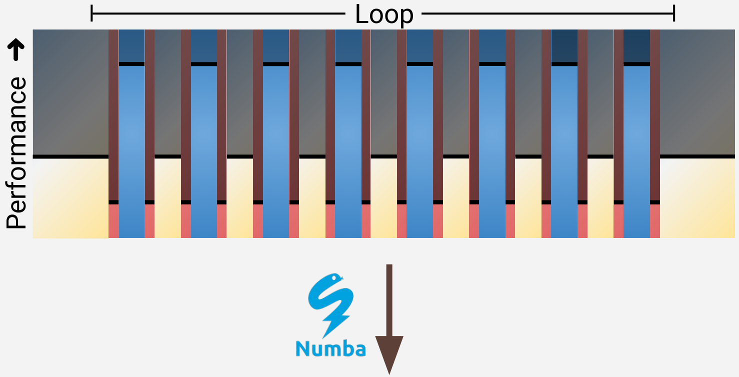
Cause of Language Barrier in Python

Python Duck Typing

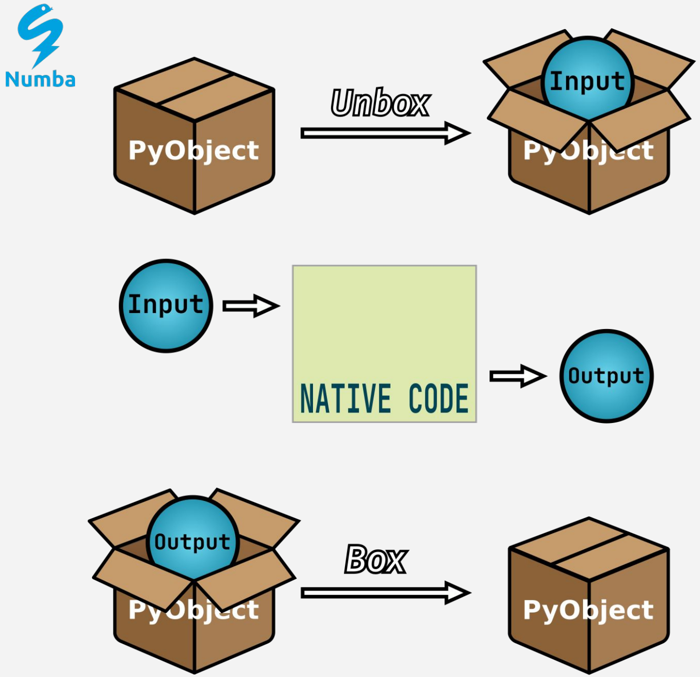




Removing Barriers Inside the Loop



Numba removes the language barriers in the loop



Numba - PyROOT Example

```
import numba
import math
import ROOT
import ROOT.NumbaExt
# □ Import the Numba extension
myfile=ROOT.TTree("vec_lv.root")
vector_of_lv=myfile.Get("vec_lv")
# □ Vector of TLorentzVector

# □ Pure Python function
def calc_pt(lv):
    return math.sqrt(lv.Px() ** 2 + lv.Py() ** 2)

def calc_pt_vec(vec_lv):
    pt = []
    for i in range(vec_lv.size()):
        pt.append((calc_pt(vec_lv[i]),
                   vec_lv[i].Pt()))
    return pt
```

```
@numba.njit # □ Numba decorator
def numba_calc_pt(lv):
    return math.sqrt(lv.Px()**2 +lv.Py()**2)

def numba_calc_pt_vec(vec_lv):
    pts = []
    for i in range(vec_lv.size()):

pts.append((numba_calc_pt(vec_lv[i]),
                     vec_lv[i].Pt()))
    return pts

Pts = calc_pt_vec(vector_of_lv)
Pts = numba_calc_pt_vec(vector_of_lv)
```

When the traditional **PyROOT pipeline** is compared against the **Numba pipeline** in the above example we get a **17x** speedup. link
Available in ROOT master so you can try it out.

Ongoing Work

1. Maximize the C++ feature set supported in Numba.
2. Upstream libInterOp into LLVM master
3. Leverage Python-C++ interop in Jupyter using cppyy

```
In [1]: class CppClass {
          int m_Value = 42;
        public:
          int getValue() {return m_Value; }
        };

In [2]: %%python

        cpp = cppyy.gbl.CppClass()
        i = cpp.getValue();
        print("`i` comes from C++ land: ", i)

        `i` comes from C++ land:  42

In [ ]:
```

Ioana Ifrim & Alexander Penev

Personal Goals of this Workshop

- How do our tools (cppyy, Jupyter with C++, etc.) fit into the future of HEP analysis?
- How does HEP community want analysis to look like?
- Packaging of tools (how big is too big?)
- Discussions about open source development.

Thank you