Award #: OAC-1931408

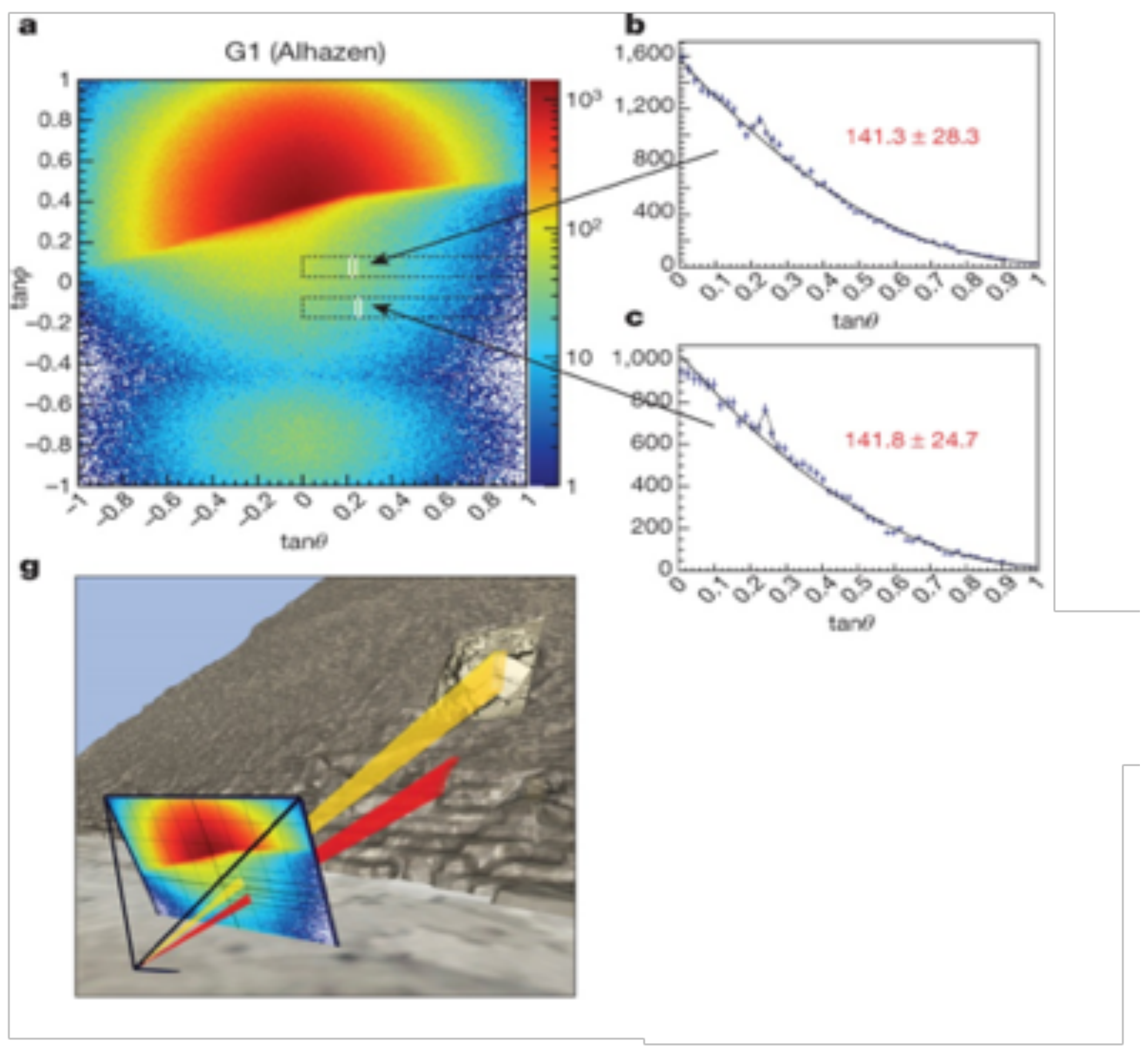
# CSSI Element: C++ as a service - rapid software development and dynamic interoperability with Python and beyond

David Lange (PI) and Vassil Vassilev
Princeton University

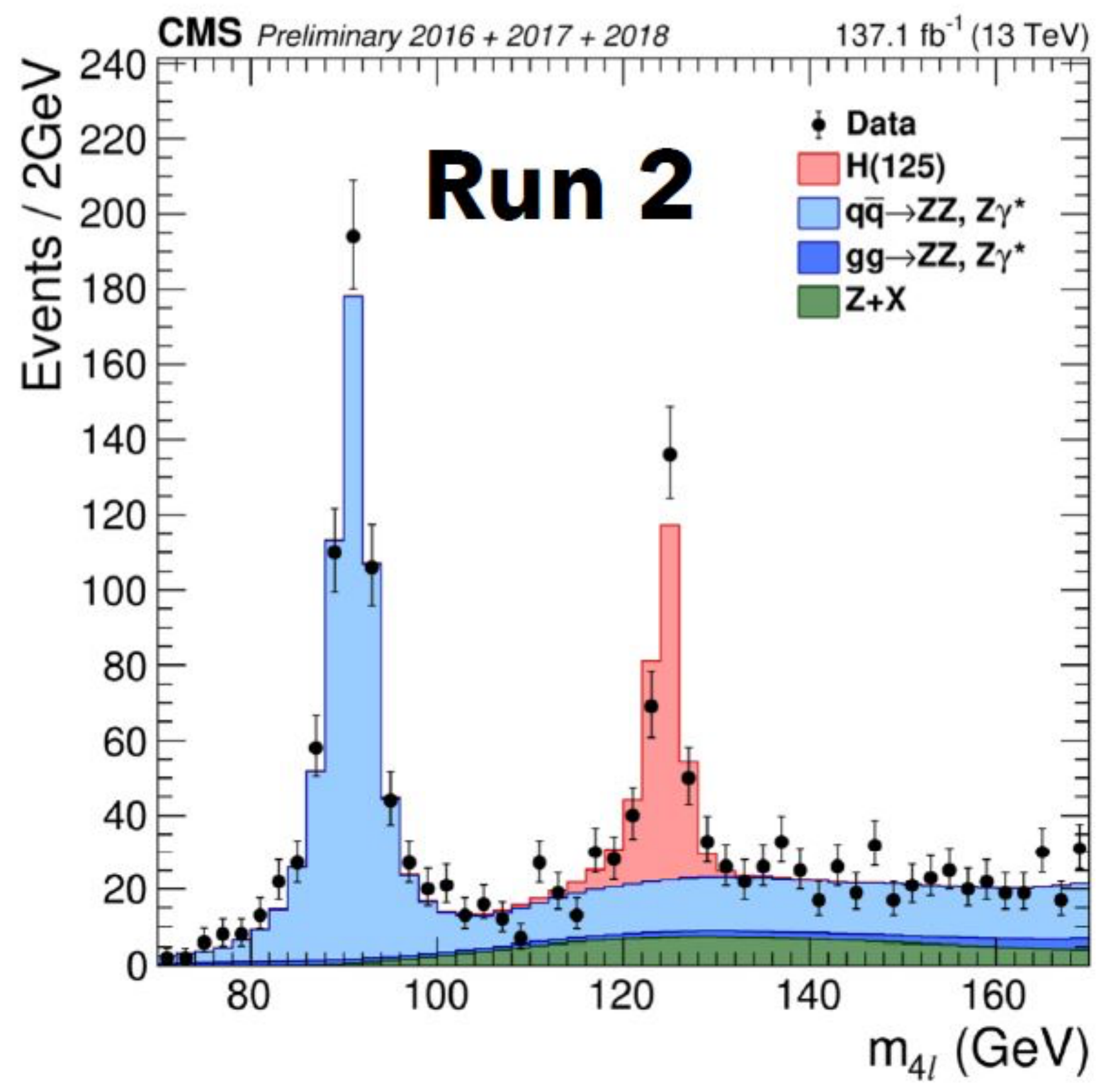
CaaS aims to provide programmers and data scientists a simple and general solution to language interoperability:

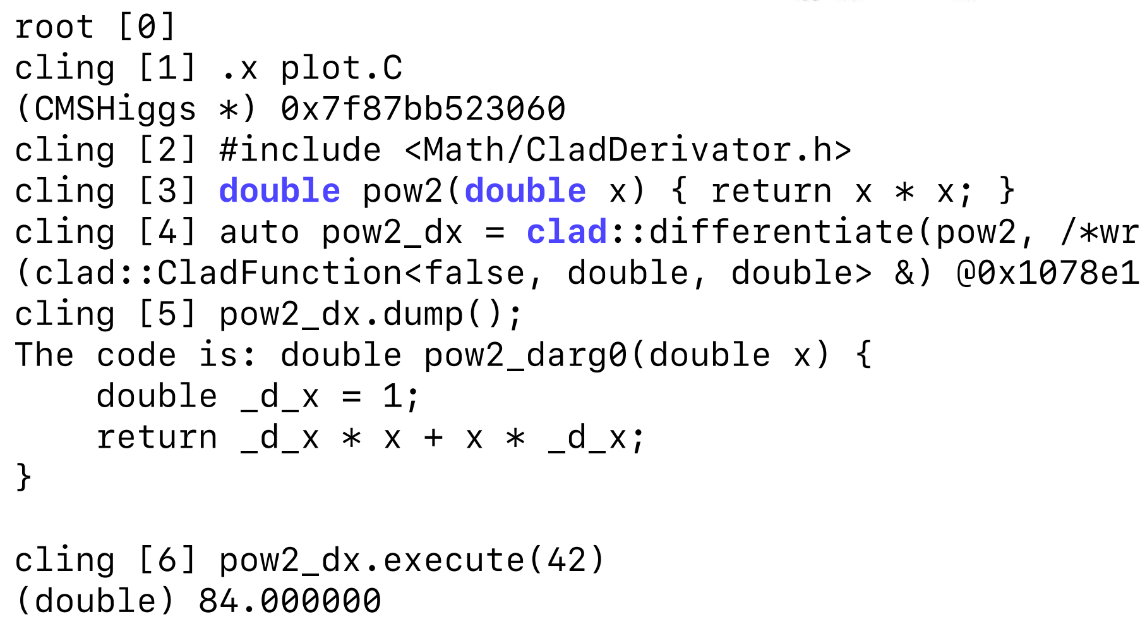
- Advance the interpretative technology to provide scientists a state-of-the-art C++ execution environment
- Enable functionality which can provide dynamic, native-like, runtime interoperability between C++ and Python
- Allow seamless utilization of heterogeneous hardware (e.g., hardware accelerators)
- To enable rapid application development even for with a complex codebase

Results of the analysis of the gas detectors



K Morishima *et al.* *Nature* **552,** 386–390 (2017) doi:10.1038/nature24647





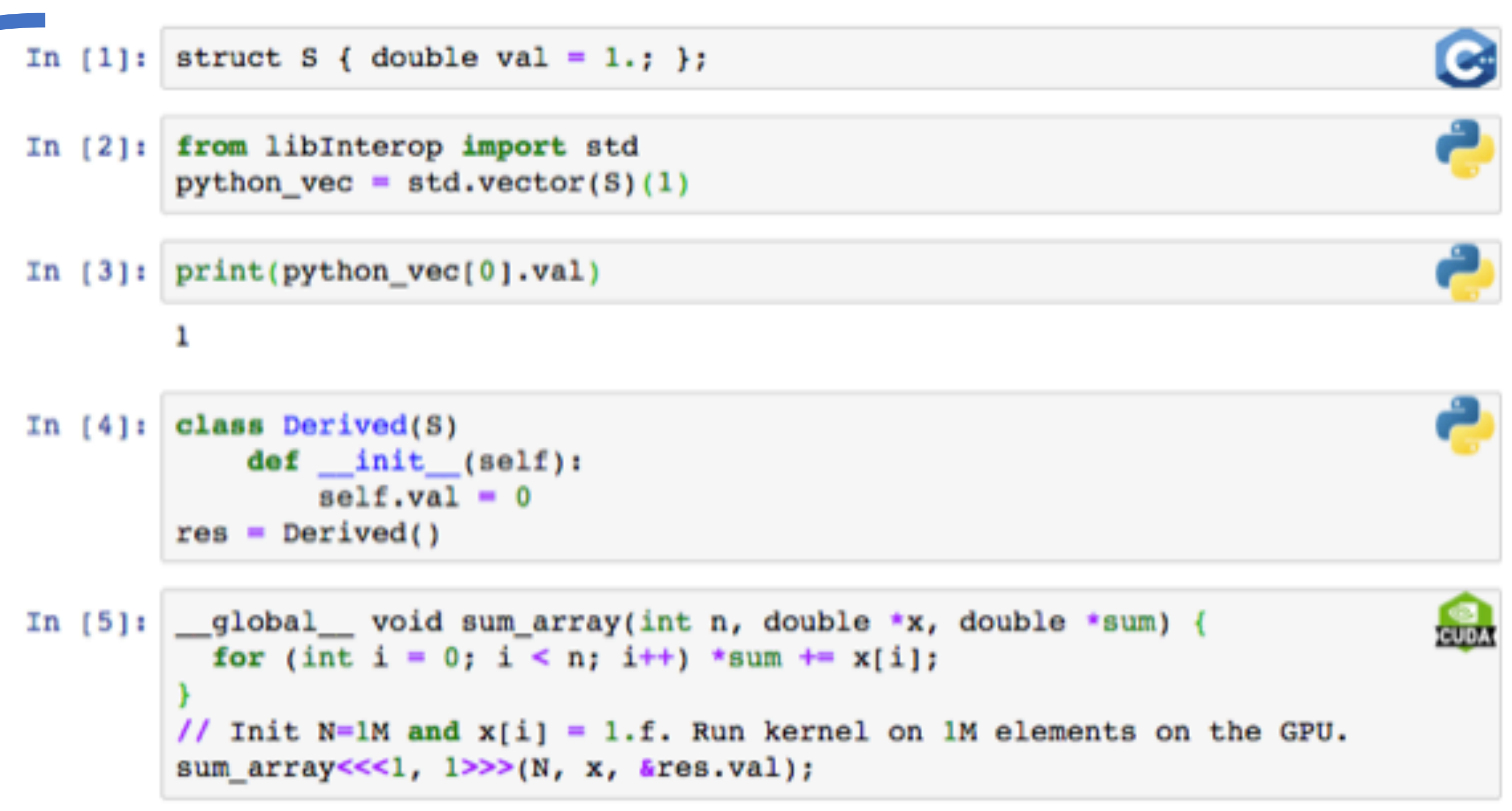
```
root [0]
cling [1] .x plot.C
(CMSHiggs *) 0x7f87bb523060
cling [2] #include <Math/CladDerivator.h>
cling [3] double pow2(double x) { return x * x; }
cling [4] auto pow2_dx = clad::differentiate(pow2, /*wr
(clad::CladFunction<false, double, double> &) @0x1078e1
cling [5] pow2_dx.dump();
The code is: double pow2_darg0(double x) {
    double _d_x = 1;
    return _d_x * x + x * _d_x;
}

cling [6] pow2_dx.execute(42)
(double) 84.000000
```

Our approach is to generalize a high-energy physics analysis tool ("Cling") to a generally accessible and fully functional tool that is part of LLVM/Clang.

Initial Science (and beyond) use cases

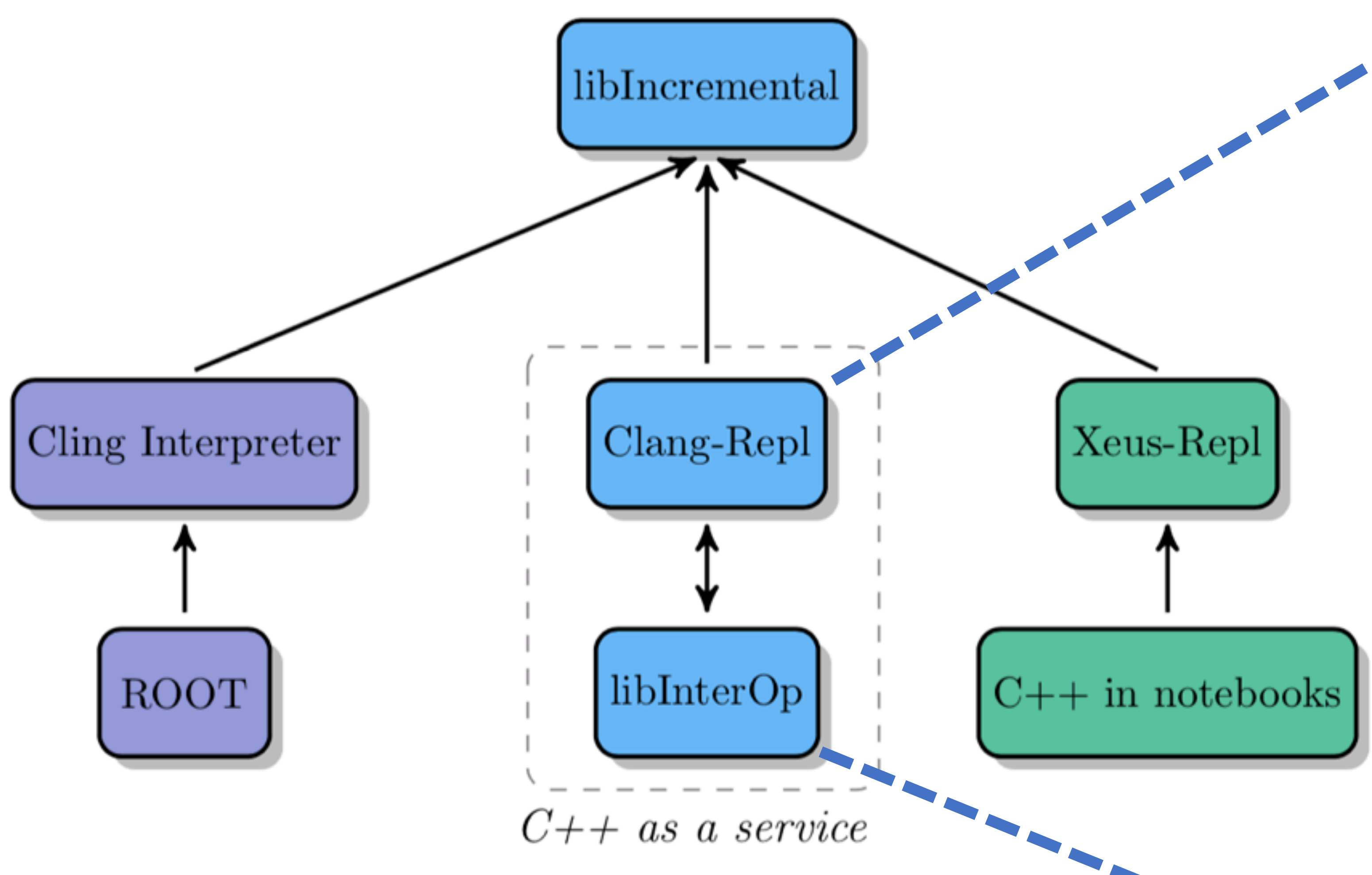
- Molecular science
- Quantum simulations
- High-energy physics
- Laser particle acceleration
- Training / Education
- Data science applications

CaaS programming model

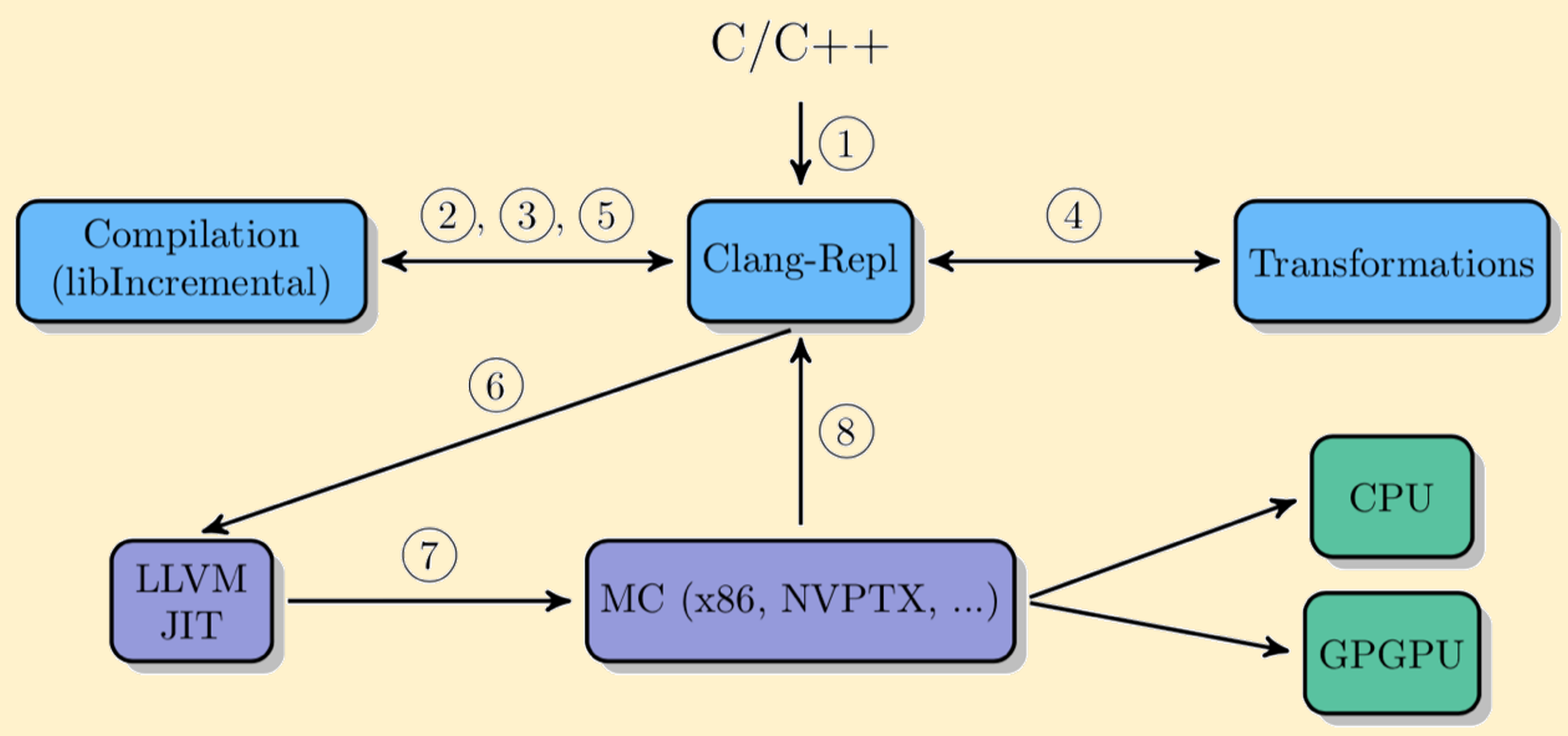


```
In [1]: struct S { double val = 1.; };

In [2]: from libInterop import std
        python_vec = std.vector(S)(1)

In [3]: print(python_vec[0].val)

        1

In [4]: class Derived(S)
            def __init__(self):
                self.val = 0
        res = Derived()

In [5]: __global__ void sum_array(int n, double *x, double *sum) {
          for (int i = 0; i < n; i++) *sum += x[i];
        }
        // Init N=1M and x[i] = 1.f. Run kernel on 1M elements on the GPU.
        sum_array<<<1, 1>>>(N, x, &res.val);
```

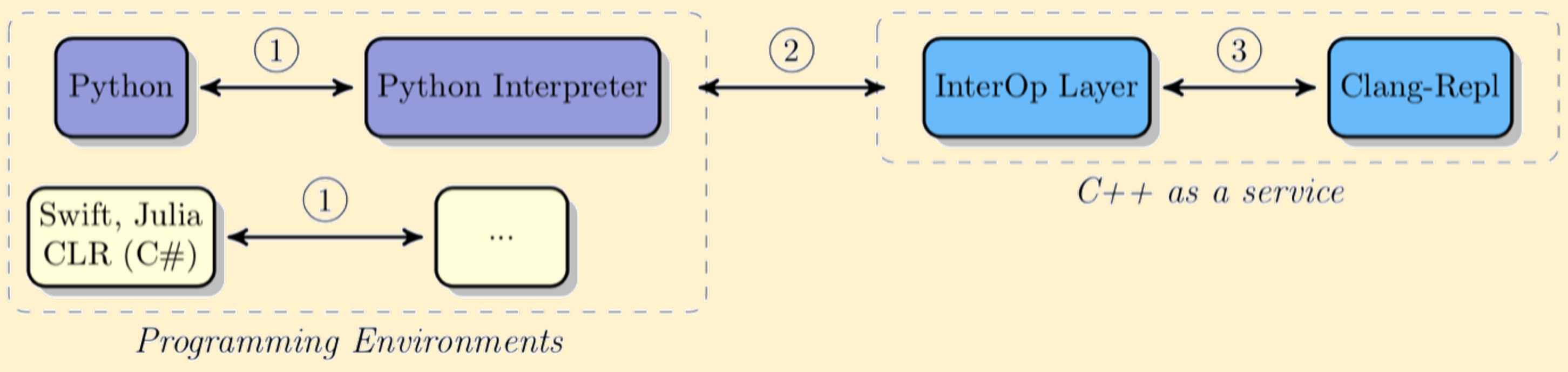
## libIncremental Design



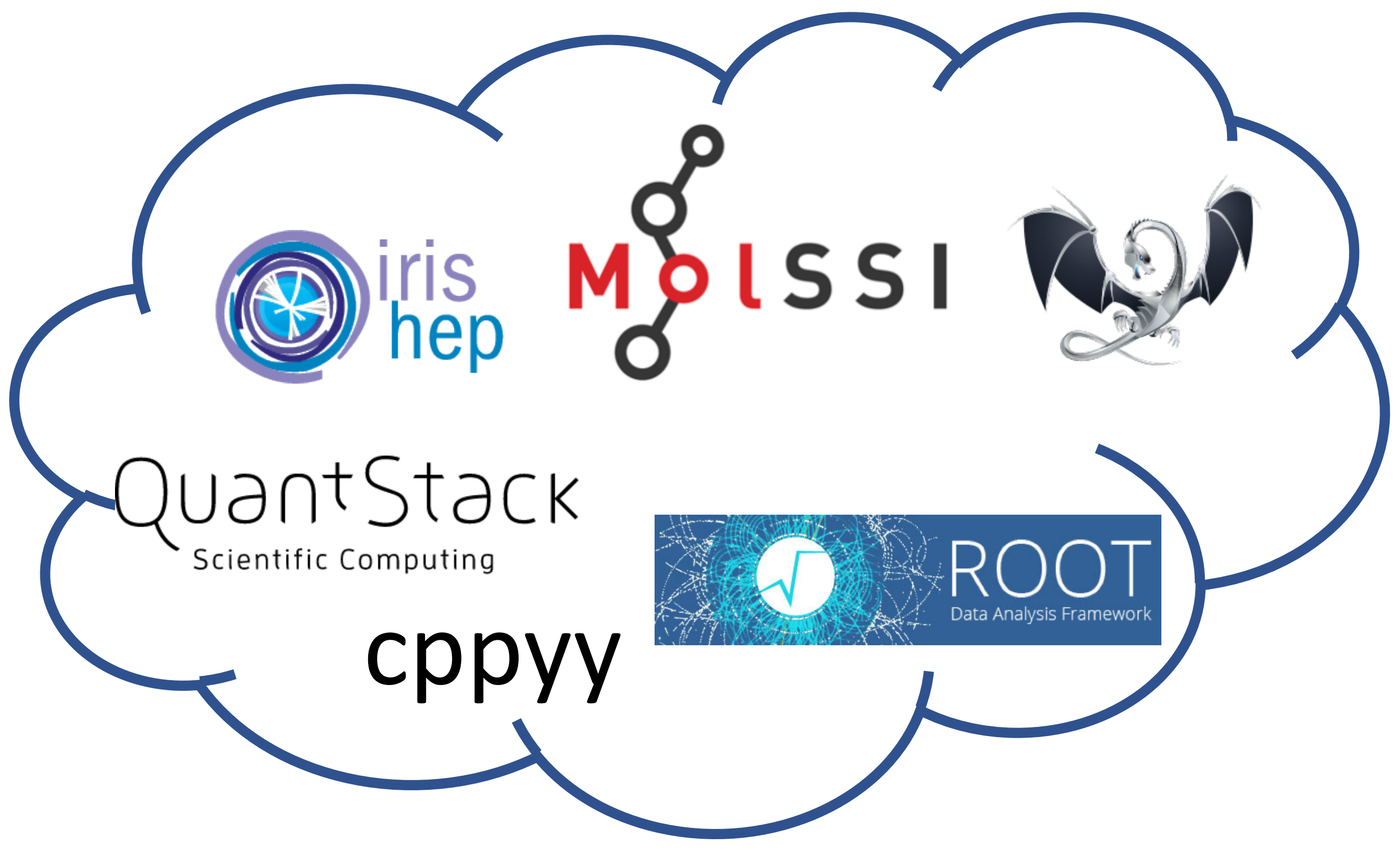
## Clang-Repl Design



## libInterOp Design



## Initial Collaborators



PRINCETON UNIVERSITY