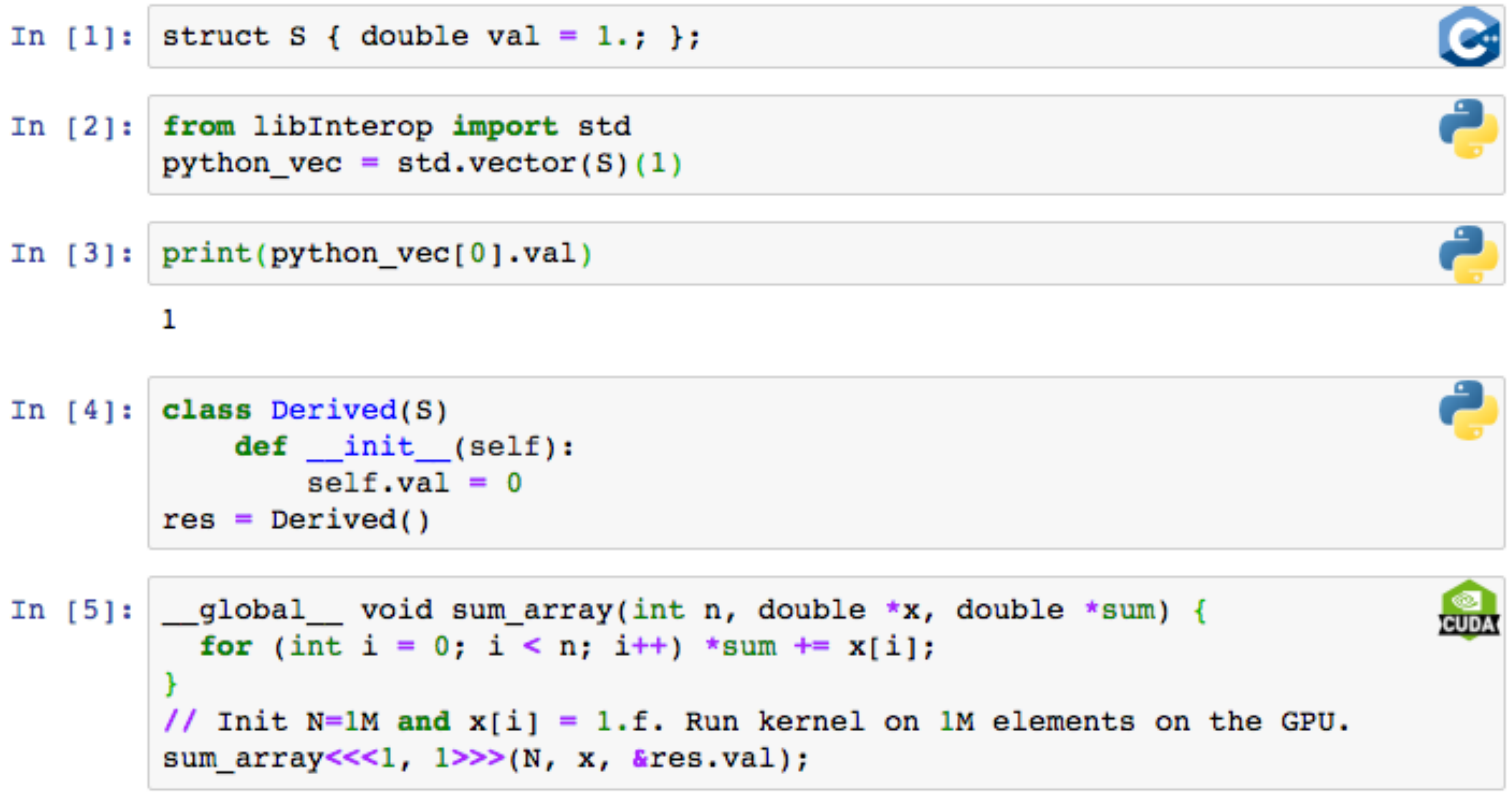
The CaaS Project. Plans Q1, Q2

Vassil Vassilev

Project Goals

- Support for incremental compilation (clang::libInterpreter, Clang-Repl)
- Language interoperability layer (cppyy, libInterOp)
- Heterogeneous hardware support (offload execution, clad demonstrator)
- Use case development & community outreach (tutorial development, demonstrators)

Project Goals



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

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

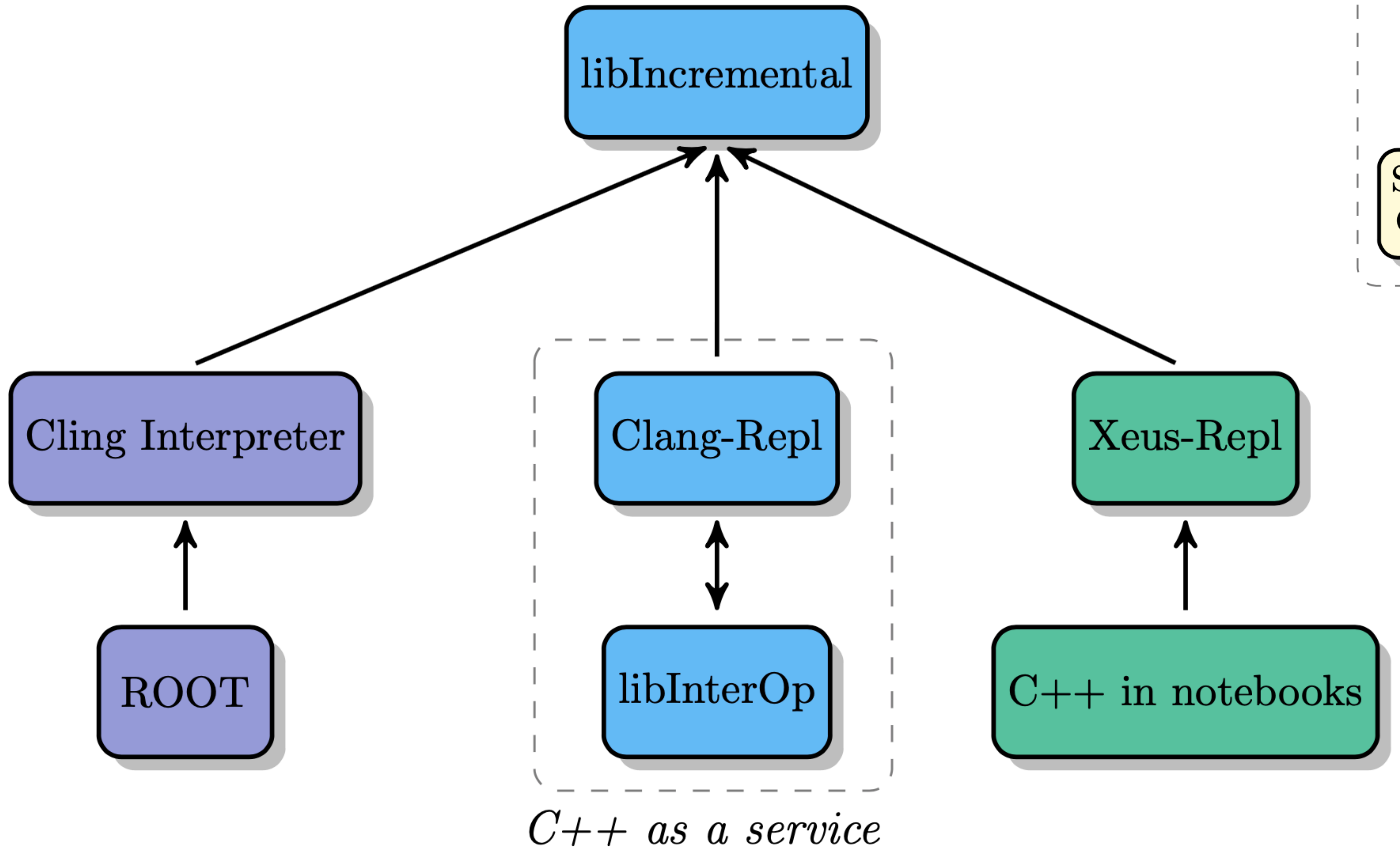
```
In [3]: print(python_vec[0].val)

        1
```

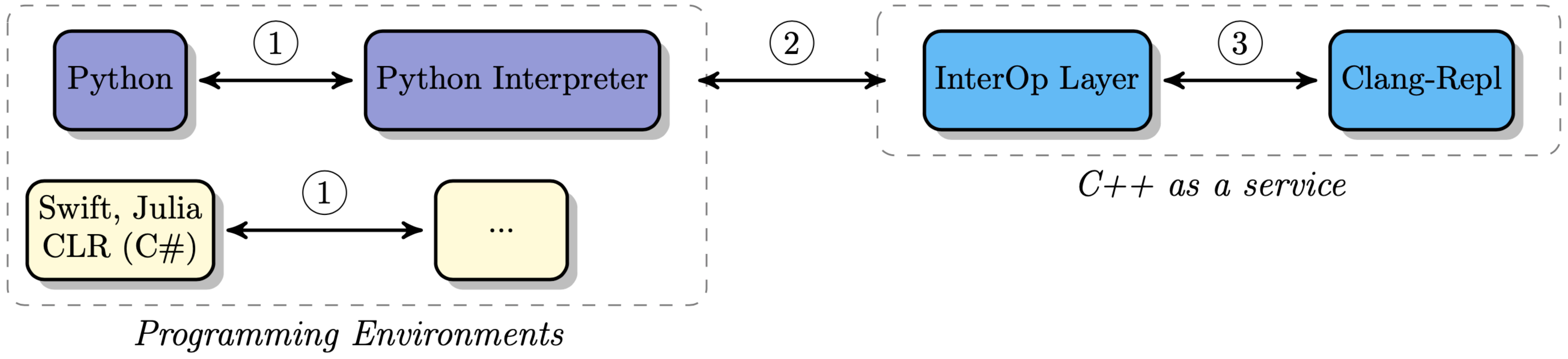
```
In [4]: class Derived(S)
            def __init__(self):
                self.val = 0
        res = Derived()
```

```
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);
```

Enable bi-directional language communication capable of controlling accelerator hardware

Project Goals





Reroute the cling-based ecosystem more to llvm upstream

How to get there. Q1

1. Upgrade to LLVM 13 — Q1/VV
2. Update Cling to use more of LLVM13 — Q1/VV?
3. Construct simple patches to upstream dashboard to track — Q1/?
4. Upstream Cling-specific patches — Q1-Q4/VV, GS, BK?
5. Keep track of Cling SLoC — Q1-Q4/DL

How to get there. Q1

6. Connect Clang-Repl to the Python Interpreter — Q1/BK
   The python interpreter provides C API which allows to expose itself and switch to writing python code on the prompt. In ROOT this happens via TPython::Prompt and we want the modern version of this for clang-repl.

7. Rebase cppyy to use cling-only interfaces (making cppyy ROOT-independent) — Q1/BK
   The task is about transforming the various ROOT Meta layer calls to their underlying clang/cling analogs

8. Define a set of new classes which handle what's needed (eg TClingCallFunc, etc) — Q1/BK
   The task is about extracting the common cases where we need a lot of boilerplate code and provide abstractions for it. For example, the mechanism to call functions in a uniform way (currently done with TClingCallFunc) needs to modernized into its own ROOT-independent entity in libInterOp

How to get there. Q1

9. Connect libInterOp with clang-repl (see 6)— Q1/BK
   The python interpreter provides C API which allows to expose itself and switch to writing python code on the prompt. In ROOT this happens via TPython::Prompt and we want the modern version of this for clang-repl.

10. Improve test cases and demonstrators — Q1/II
    The task is about updating the existing demonstrators and developing new ones given the advances in Clad.

11. Differentiate CUDA kernels — Q1/II

12. ACAT proceedings — Q1/II

13. Support Tensors and showcase differentiation of Eigen entities — Q1/PA

14. Deliver error estimation talk at SIAM incl the req. development— Q1/GS

How to get there. Q2

15. Implement in clang an extension to allow statements on the global scope — Q2/VV

16. Add extensible value printing facility — Q2/VV

17. Advance error recovery and code unloading — Q2/PC
    The task is to make clang-repl more robust when it comes to surviving from errors.

18. Design and Develop a CUDA engine working along with C++ mode —Q2/II,SSP
    The task is to improve and generalize the implementation of the PTX support in cling and demonstrate it in clang-repl.

19. Rebase cppyy to use clang-repl/libInterpreter interfaces — Q2/BK

20. Develop demonstrators (eg the one from the Jupiter mockup) — Q2/BK

21. Design and implement a backend capable of offloading computations to a GPGPU. Assess technical performance of gradient produced by Clad on GPGPU — Q2/II,VV

How to get there. Q2

22. Add more clad benchmarks — Q2/DL

23. Add extensible value printing facility — Q2/VV

24. Write a paper on AD for the aggregate types — Q2/PA

25. Write an Error Estimation paper — Q2/GS

Extra items for Summer Students

- Upstream Type sugaring patches
- Implement autocompletion in clang-repl
- Develop documentation, examples and tutorials (in llvm documentation as well)
- Initiate tutorial development within the Clang-Repl community and integrate Clang- Repl into Xeus. Blog post on working notebook demonstrating tutorial
- Implement the LLVM extension of binding C++ memory management model more accurately and implement prototype using cppyy. [if yet feasible]

Extra items for Summer Students

- [ROOT] Improve robustness of dictionary to module lookups.
  The few run time failures in the modules integration builds are due to dictionaries that can not be found in the modules system. These dictionaries are present as the mainstream system is able to find them using a broader search. The modules setup in ROOT needs to be extended to include a dictionary extension to track dictionary<->module mappings for C++ entities that introduce synonyms rather than declarations (`using std::vector<A<B>> = MyVector` where the dictionaries of A, B are elsewhere). This is estimated to be 1.5 weeks of full time work.

- [ROOT] Optimize ROOT use of modules for large codebases (eg, CMSSW)
  One source of performance loss is the need for symbol lookups across the very large set of CMSSW modules. ROOT needs to be improved to optimize this lookup so that it does not pull all modules defining namespace `edm` on `edm::X` lookups. This is estimated to be 3 weeks of full time work.

- Link to the CaaS GSoC proposal doc.