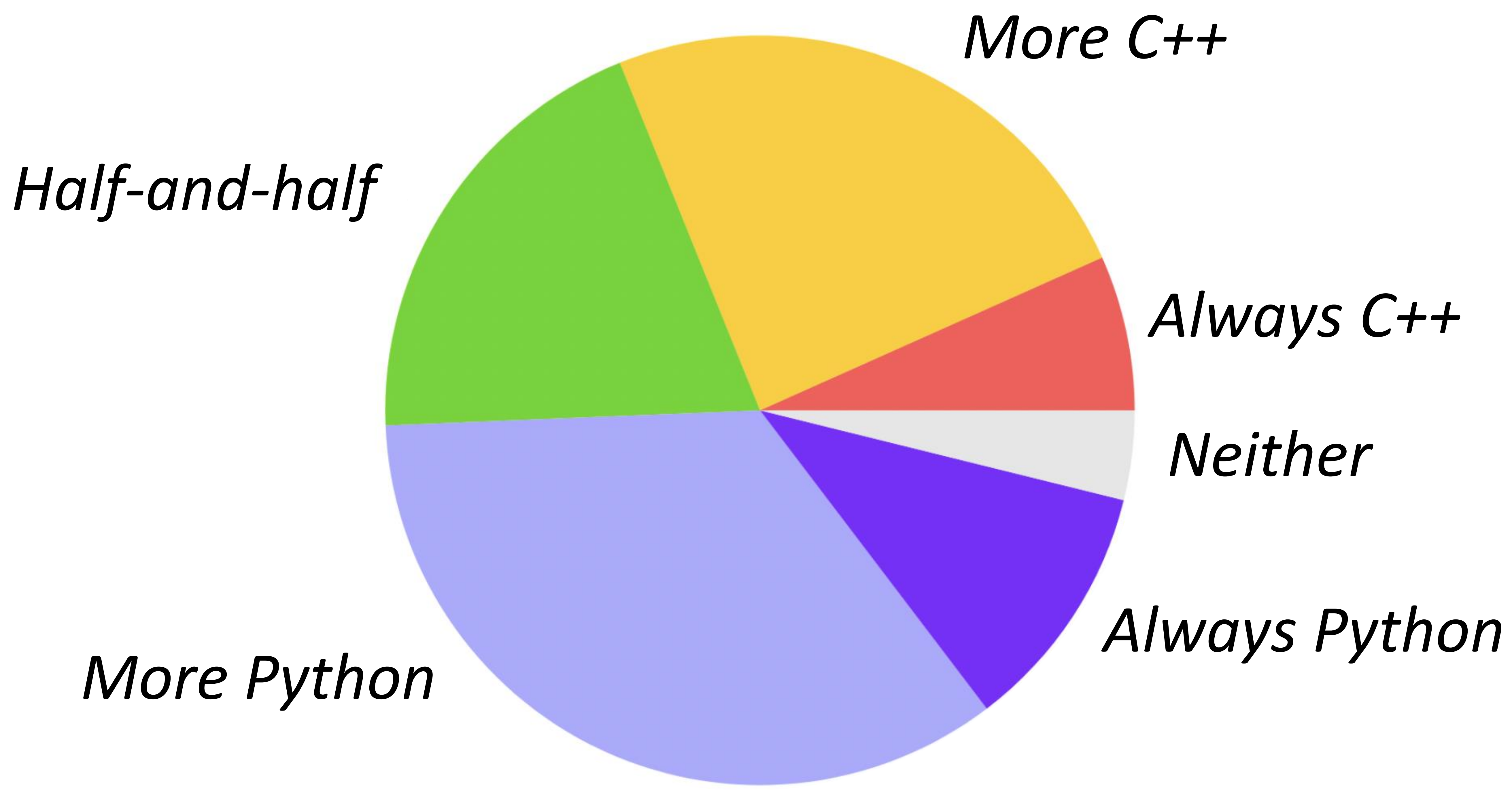
Adapting C++ for Data Science

Vassil Vassilev, Princeton University
compiler-research.org

The current work is partially supported by National Science Foundation under Grant OAC-1931408. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

How often do you use Python relative to C/C++?



[PyHEP 2020, J. Pivarski]

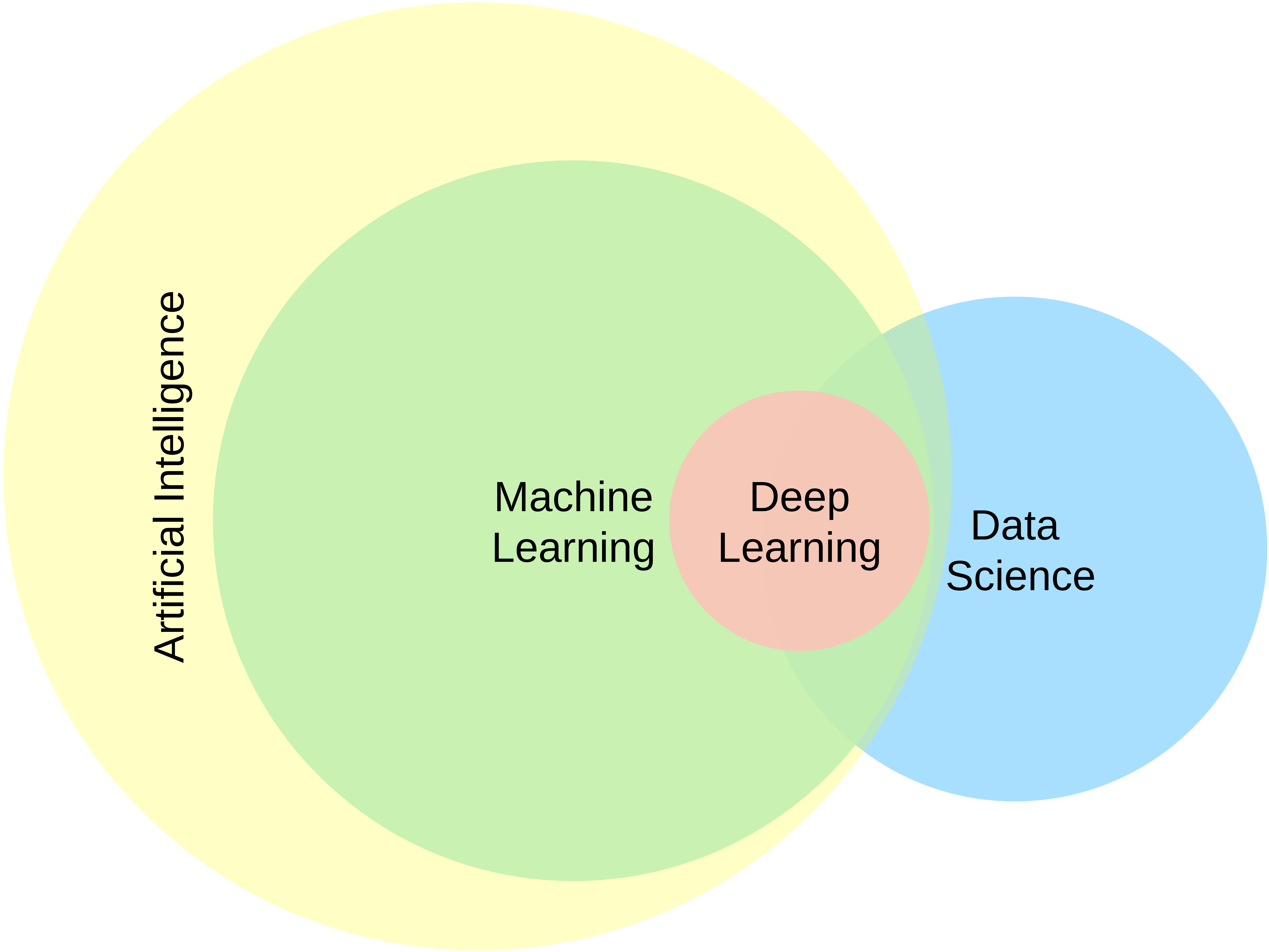
HEP has ~O(20M) LoC written in C++

What do we do with existing code written in C++?

rm --rf ‘em all?

Keep expanding them randomly?

Scope


Artificial Intelligence
Machine Learning
Deep Learning
Data Science

Tools

“Human brain has not evolved structurally a lot since 17th century however the development of mankind has, because we learned how to build better tools.”

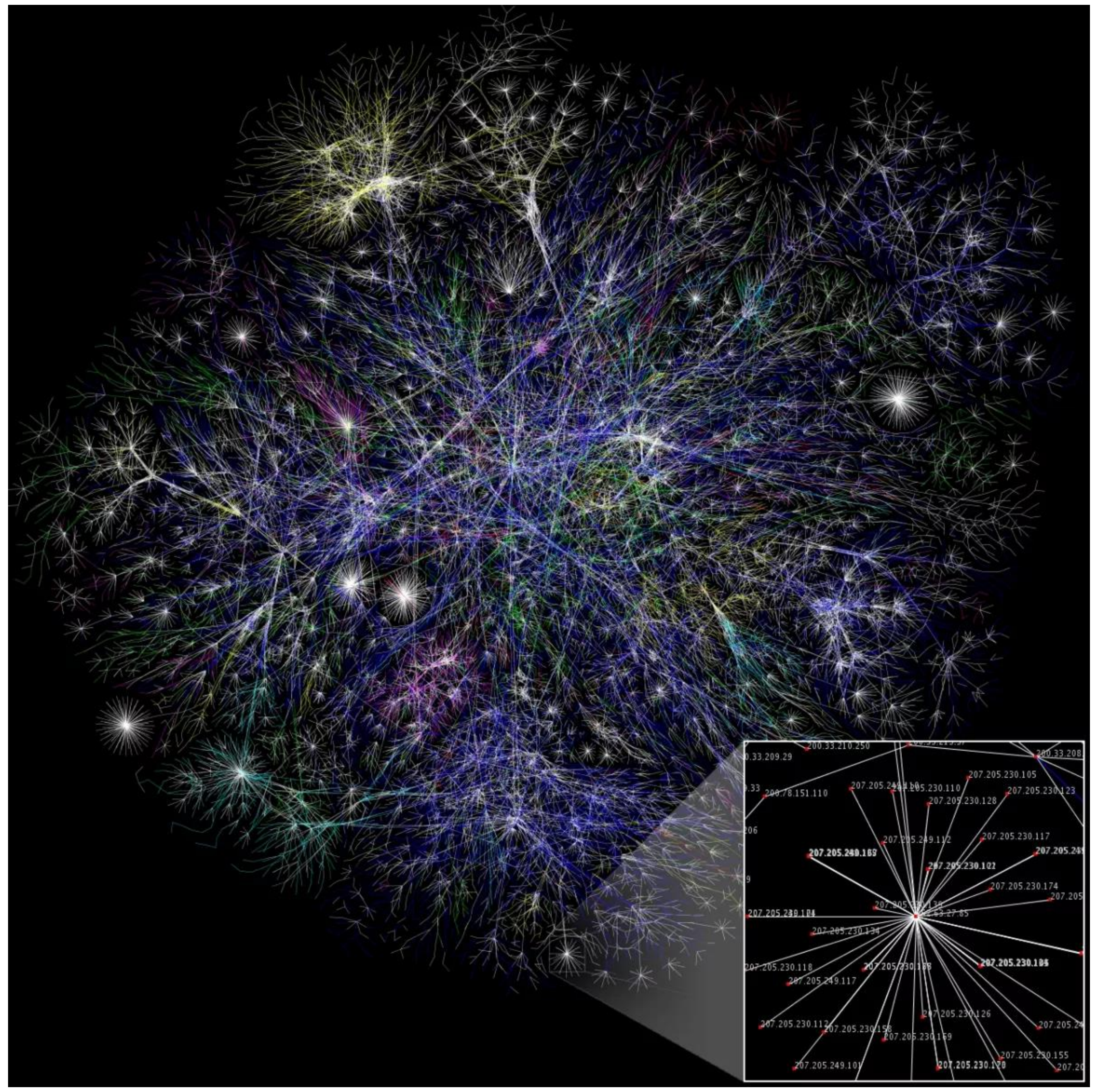
The Printing Press

(Image credit: Britannica Fine Art Images/Heritage Image/age fotostock)

The Lightbulb

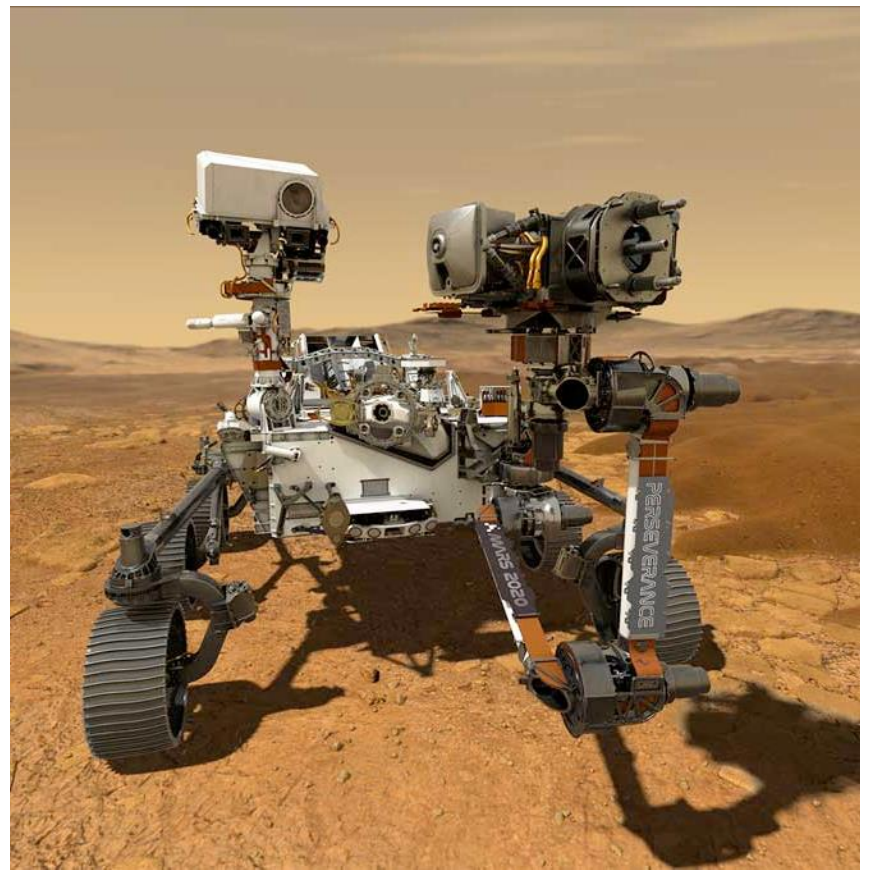
(Image credit: Terren | Creative Commons)

The Internet



(Image credit: Creative Commons | The Opte Project)

Language Design Principles



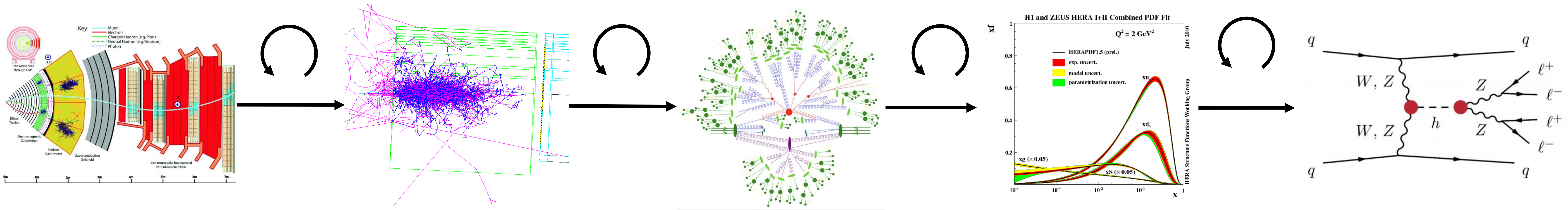
C++
- Efficiency
- Stability
- Backward compatibility

"Prioritizes Performance over Surprise which is sometimes surprising" T. Winters [Link]

Python
- Readability
- Simplicity
- Flexibility

"Special cases aren't special enough to break the rules" Zen of Python [Link]

Talking to a Dataset



Understanding the Language of a Dataset – a multistep, iterative, interactive, exploratory process:

- Interactivity = [human] productivity + *just enough* performance

"Interactive Supercomputing for Data Science" W. Reus [Link]()

Just Enough Performance

```
def f(N = 100, M = 1000, L = 10000):
    for i in range(N):
        for j in range(M):
            for k in range(L):
                g(i, j, k)
```

Three desirata:

1. A language people already know
2. Covers the whole language, not a subset
3. Delivers bare-metal speed, not just a factor-of-several above X

Approaches:

- JIT compile using Numba – (1) & (3)
- Compile with Pypy – (1) & (2)
- Use a language such as Julia – (2) & (3)
- This talk offers a way to cover (1), (2) and (3)

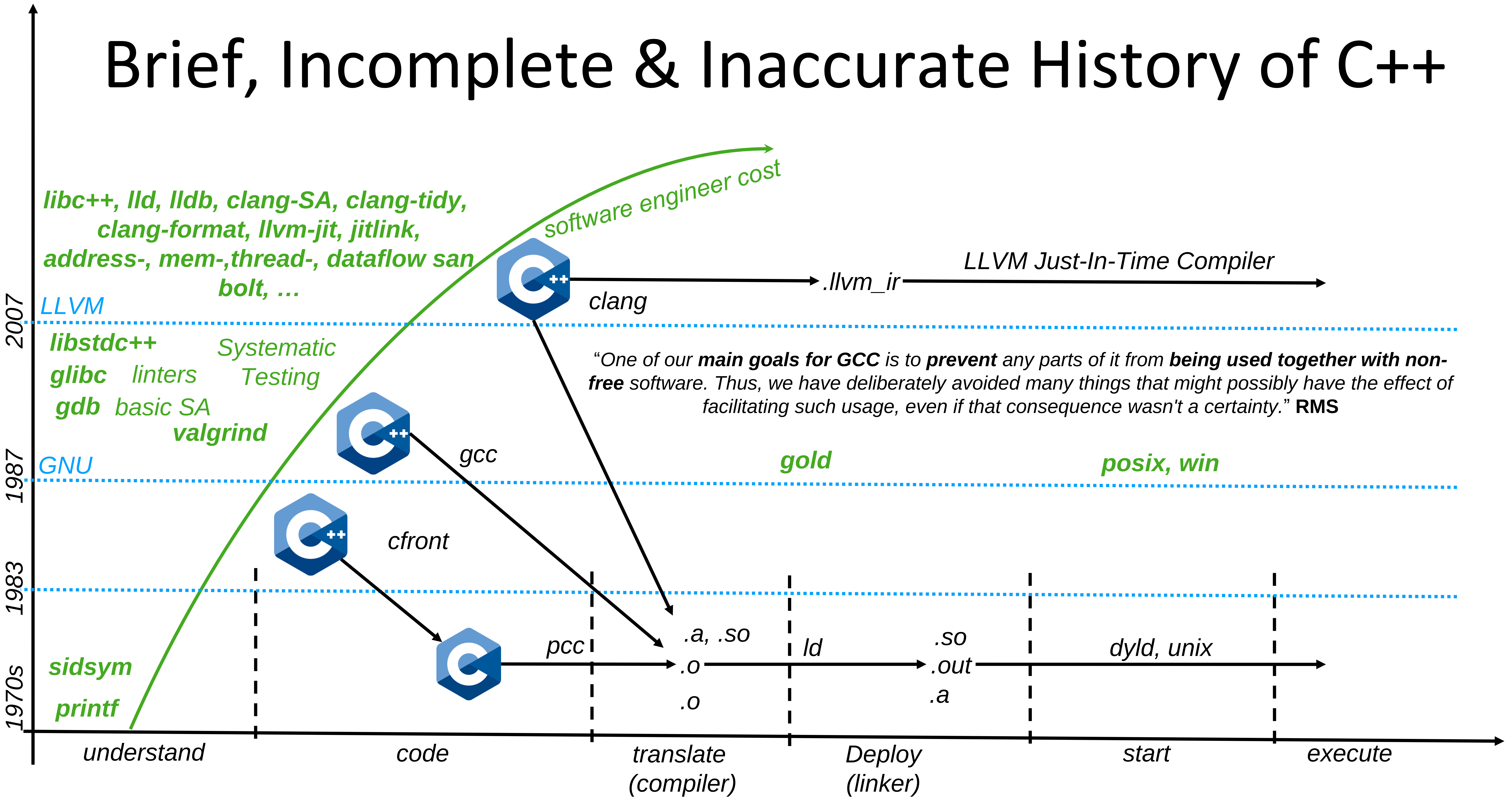
"The inner loop principle" J. Pivarski, private exchanges

What Is Python?

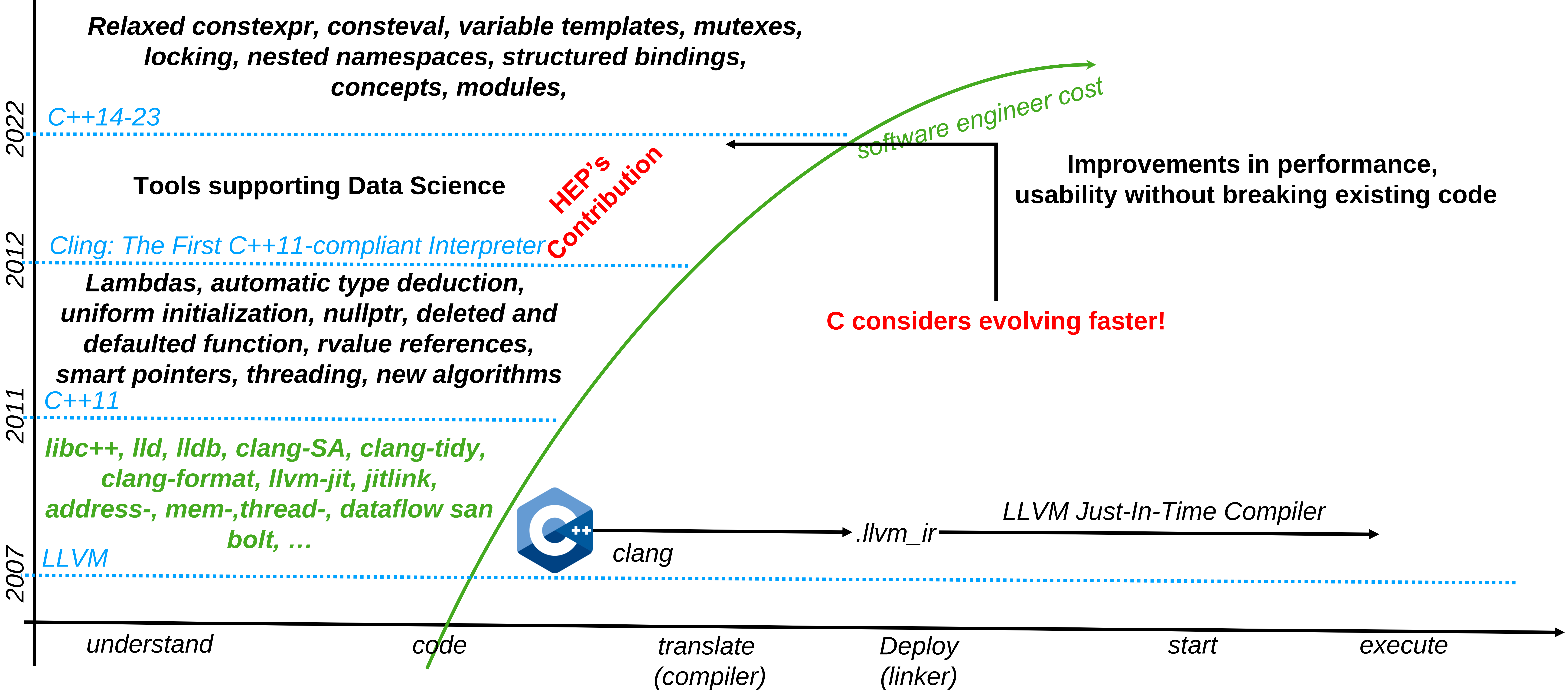
- Just enough performance when relying on bare-metal technologies
- NumPy is an enabler for an entire data science ecosystem
- NumPy is very good but sometimes far from bare metal, accelerators and across nodes (means to address the problem such as CuPy or Dask).

"This is why I love C++ and use Python for most of the work I do...", a happy user on the internet

Brief, Incomplete & Inaccurate History of C++



Brief, Incomplete & Inaccurate History of C++



My Pillars of Data Exploration

Recent C++ tool advancement is an enabling factor for:

- Interactive C/C++
- Automatic Language Interoperability
- Advanced bare-metal toolbox

Exploratory Programming With Interactive C++

Interactive C++. Key Insights

- Incremental Compilation
- Handling errors
  - Syntactic
  - Semantic
- Execution of statements
- Displaying execution results
- Entity redefinition

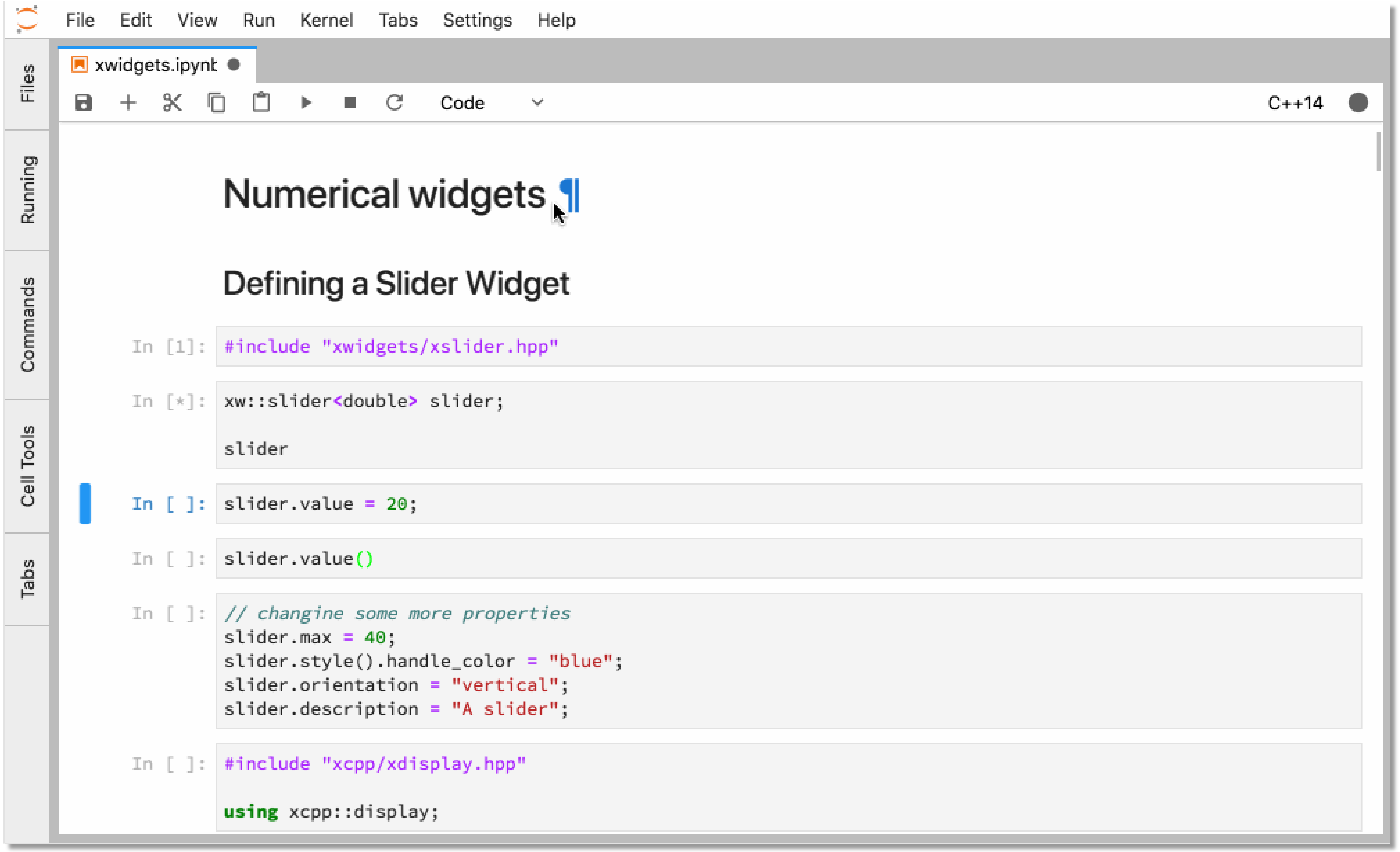
```
[cling] #include <vector>
[cling] std::vector<int> v = {1,2,3,4,5};

[cling] std.sort(v.begin(), v.end());
input_line_1:1:1: error: unexpected namespace
name 'std': expected expression
std.sort(v.begin(), v.end());
^

[cling] std::sort(v.begin(), v.end());
[cling] v // No semicolon
(std::vector<int> &) { 1, 2, 3, 4, 5 }

[cling] std::string v = "Hello World"
(std::string &) "Hello World"
```

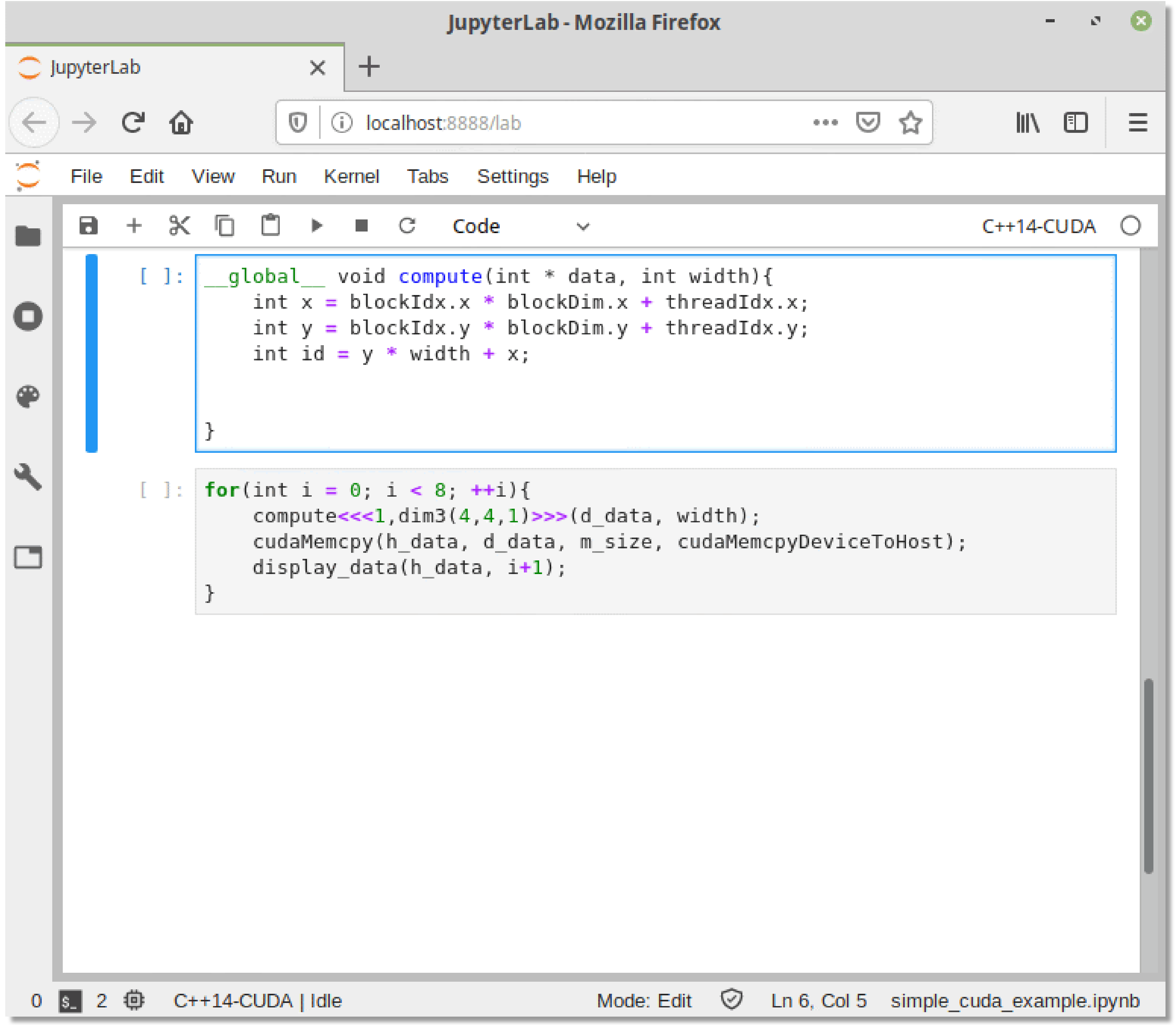
C++ in Notebooks



Xwidgets – User-defined controls

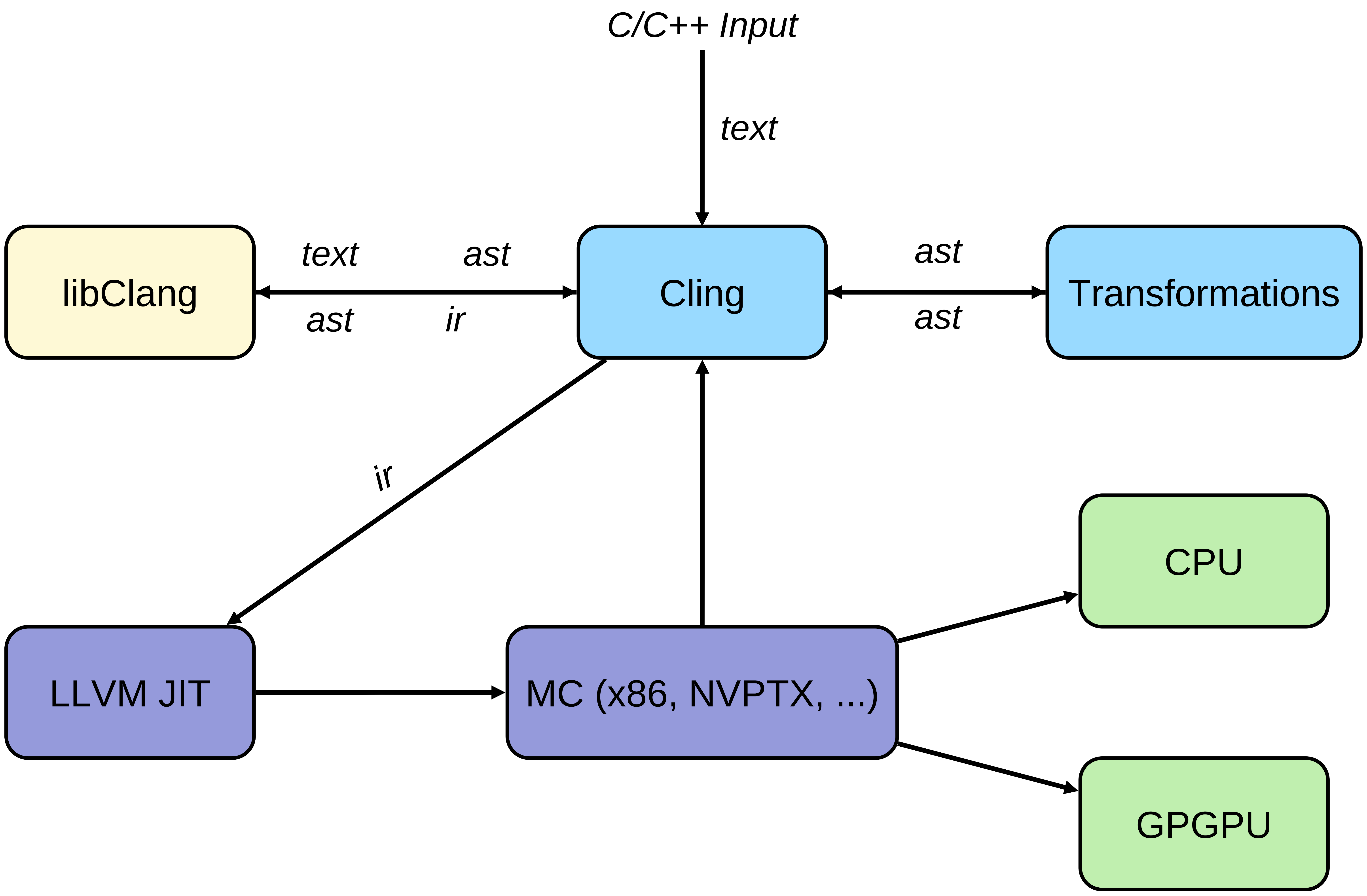
S. Corlay, Quantstack, *Deep dive into the Xeus-based Cling kernel for Jupyter*, May 2021, compiler-research.org

Interactive CUDA C++

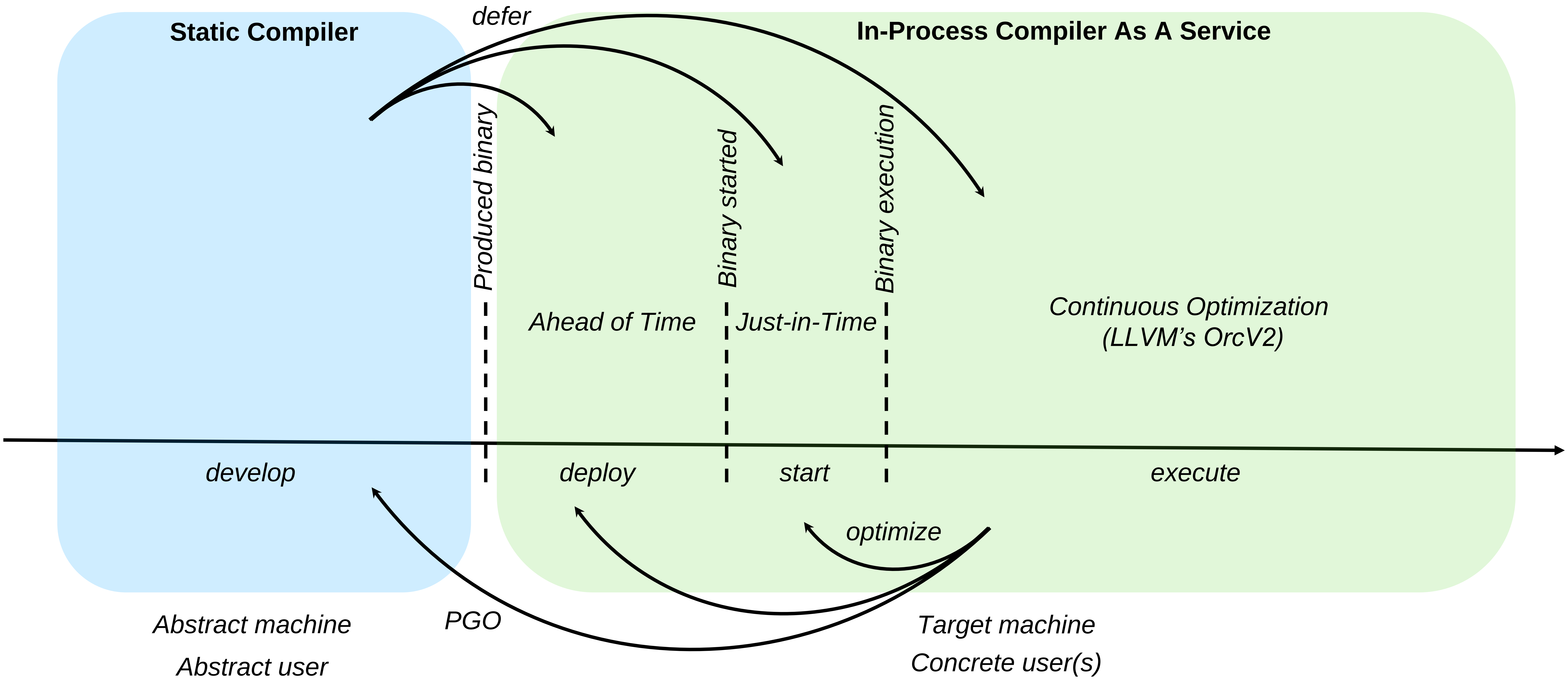


S. Ehrig, HZDR, *Cling's CUDA Backend: Interactive GPU development with CUDA C++*, Mar 2021, compiler-research.org

Interpreting C++. Cling



Compiler (C++) As A Service



CaaS. Programming Model

```
/// Call an interpreted function using its symbol address.
void callInterpretedFn(cling::Interpreter& interp) {
  // Declare a function to the interpreter. Make it extern "C"
  // to remove mangling from the game.
  interp.declare("#pragma cling optimize(1)"
      extern \"C\" int cube(int x) { return x * x * x; }");
  void* addr = interp.getAddressOfGlobal("cube");
  using func_t = int(int);
  func_t* pFunc = cling::utils::VoidToFunctionPtr<func_t*>(addr);
  std::cout << "7 * 7 * 7 = " << pFunc(7) << '\n';
}
```

```
// caas-demo.cpp
// g++ ... caas-demo.cpp; ./caas-demo
int main(int argc, const char* const* argv) {
  cling::Interpreter interp(argc, argv, LLVMDIR);

  callInterpretedFn(interp);
  return 0;
}
```

```
vvassilev@vv-nuc ~/.../builddir $ ./caas-demo
7 * 7 * 7 = 343
vvassilev@vv-nuc ~/.../builddir $
```

Automatic Language Interoperability

Automatic Language InterOp. Python

Performance Compared to Static Approaches

- No fundamental CPU performance difference

Note carefully that *everything* in Python is runtime: compile-time just means that the bindings *recipe* is compiled, not the actual bindings themselves!

- But heavy Cling/LLVM dependency:
  - ~25MB download cost; ~100MB memory overhead
  - Complex installation (and worse build)

BERKELEY LAB - 24 - U.S. DEPARTMENT OF ENERGY

Basic Performance Test: overload

Tool	Execution time (ms/call)*
C++ (Cling w/ -O2; out-of-line)	1.8E-6
cppyy / pypy-c	0.50
cppyy / CPython	1.25
swig (builtin)	1.29
swig (default)	4.23
pybind11	6.97

⇒ C++ overload is resolved at compile time, not based on dynamic type
⇒ Largest overhead: Python instance type checking (avoidable, but clumsy)
⇒ There is no obvious benefit to "static" over runtime bindings

(*) lower is better

BERKELEY LAB - 27 - U.S. DEPARTMENT OF ENERGY

W. Lavrijsen, LBL, cppyy, Sep 2021, compiler-research.org

The approach does not require the project maintainer to bother providing static bindings

Extending Data Scientist's Toolbox

Crossing the language barrier is expensive

Our Compiler-As-A-Service Approach solves that

```
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);
```

compiler-research.org's Compiler-As-A-Service Project Final Goal

Extending Data Scientist’s Toolbox. Results

Numba - PyROOT Example

```python
import numba
import math
import ROOT
import cppyy.numba_ext
#  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
```

```python
@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 **Pure Python pipeline** is compared against the **Numba pipeline** in the above example we get a **17x** speedup. link

100x should be within reach

15

B. Kundu, Princeton, *Efficient and Accurate Automatic Python Bindings with Cppyy & Cling*, Tue, ACAT22

Advanced Bare-Metal Toolbox For Data Science

Domain-Specific Tools For Data Science

Opening up the toolchain allows us to build domain-specific extensions better adapted for our field. We also can extract dataset-specific knowledge:

- Reasoning about algorithm precision and numerical stability
- Providing exact and fast gradients using automatic differentiation techniques
- Enabling sensitivity analysis across HEP components using differentiable pipelines

CaaS. Domain-Specific Data Science Tools

```cpp
int main(int argc, const char* const* argv) {
  std::vector<const char*> argvExt(argv, argv+argc);
  argvExt.push_back("-fplugin=etc/cling/plugins/lib/clad.so");
  cling::Interpreter interp(argvExt.size(), &argvExt[0], LLVMDIR);
  gimme_pow2dx(interp);
  return 0;
}
```

```cpp
#include <...>
// Derivatives as a service.
void gimme_pow2dx(cling::Interpreter &interp) {
  // Definitions of declarations injected also into cling.
  interp.declare("double pow2(double x) { return x*x; }");
  interp.declare("#include <clad/Differentiator/Differentiator.h>");
  interp.declare("#pragma cling optimize(2)");
  interp.declare("auto dfdx = clad::differentiate(pow2, 0);

  cling::Value res; // Will hold the evaluation result.
  interp.process("dfdx.getFunctionPtr();", &res);
  ...
}
```

Can generate computationally efficient gradients for codes with differentiable properties

```
vvassilev@vv-nuc ~/.../builddir $ ./caas-demo
dfdx at 1 = 2.000000
dfdx code: double pow2_darg0(double x) {
    double _d_x = 1;
    return _d_x * x + x * _d_x;
}

vvassilev@vv-nuc ~/.../builddir $
```

CaaS. Precision Tuning With Clad

Taylor-based Estimation Floating Point Errors for a dataset using AD:

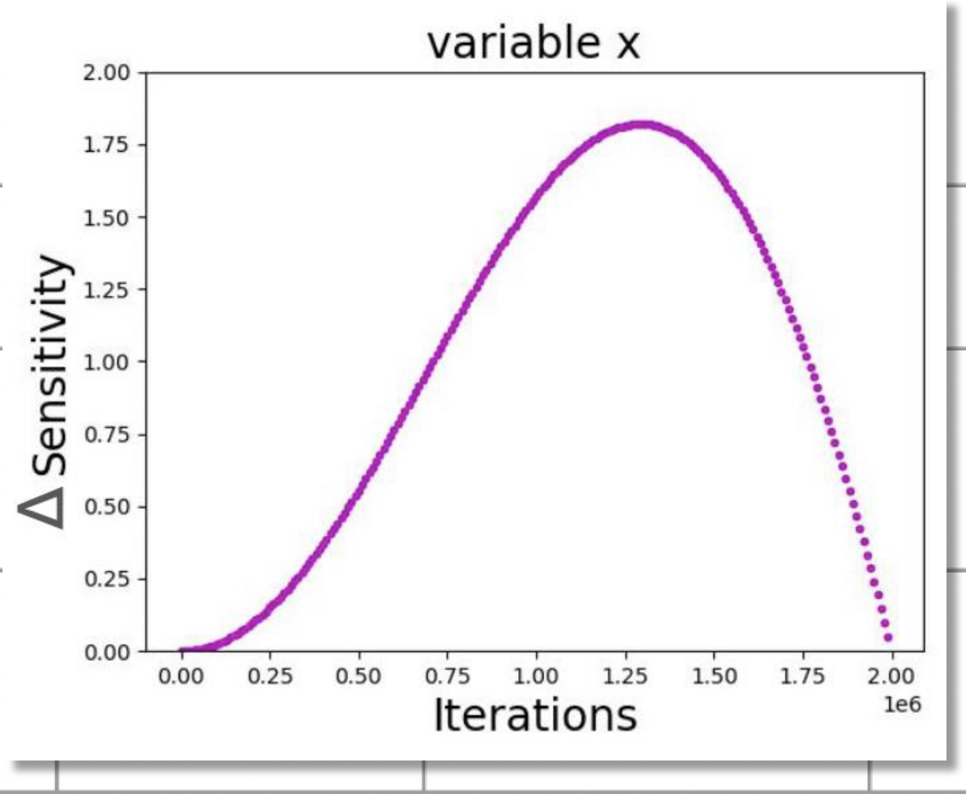
$$S_{x_i} \equiv \left|\frac{\partial f}{\partial x_i} \cdot x_i\right|$$

AD enables sensitivity analyses we could not do before.

Case Study: Simpson's Rule

Results

Precision configurations	Absolute Error	Clad's Estimated Upperbound	Variables in lower precision (out of 11)
10-byte extended precision (*long double*)			0
Clad's mixed precision			6
IEEE double-precision (*double*)			-
IEEE single-precision (*float*)			-

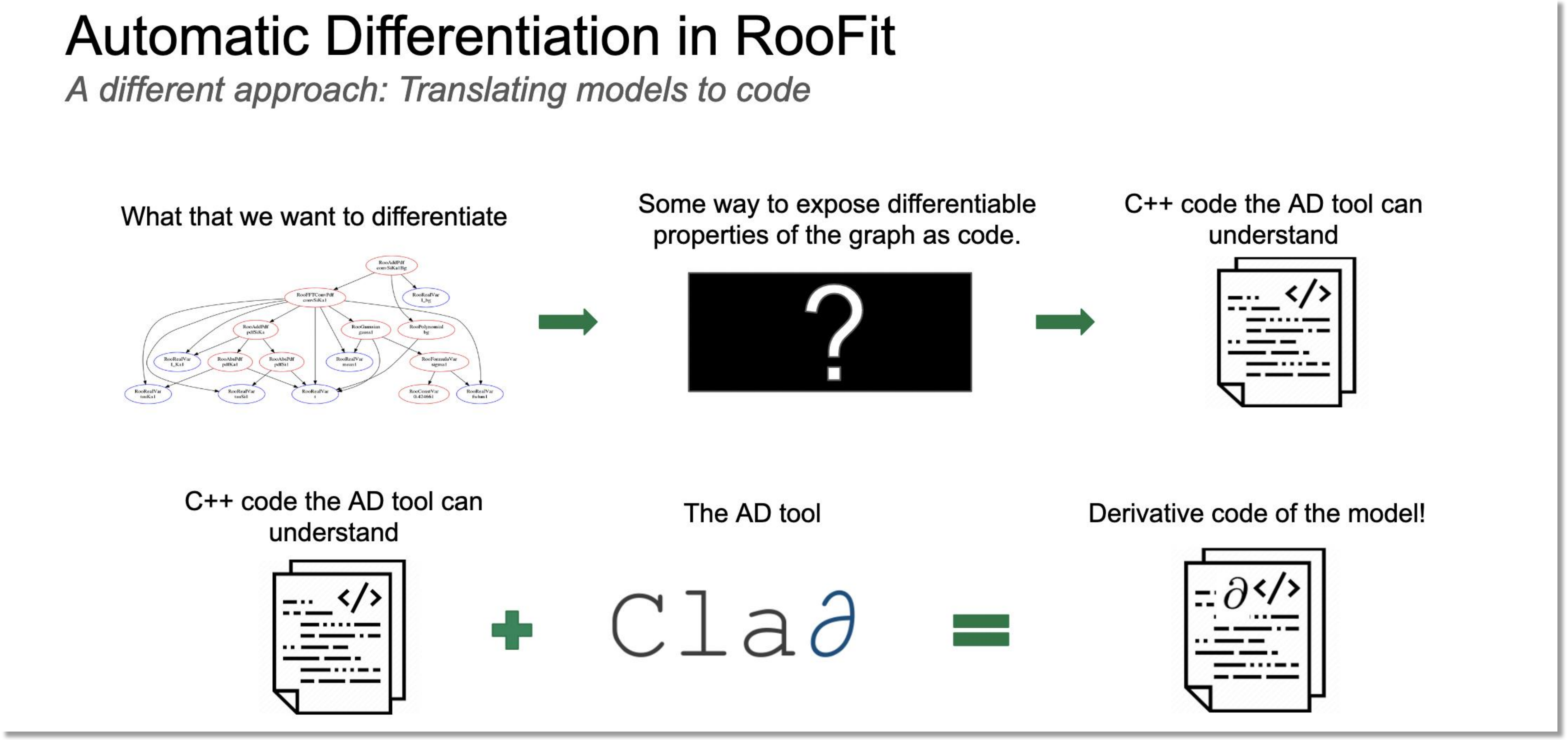


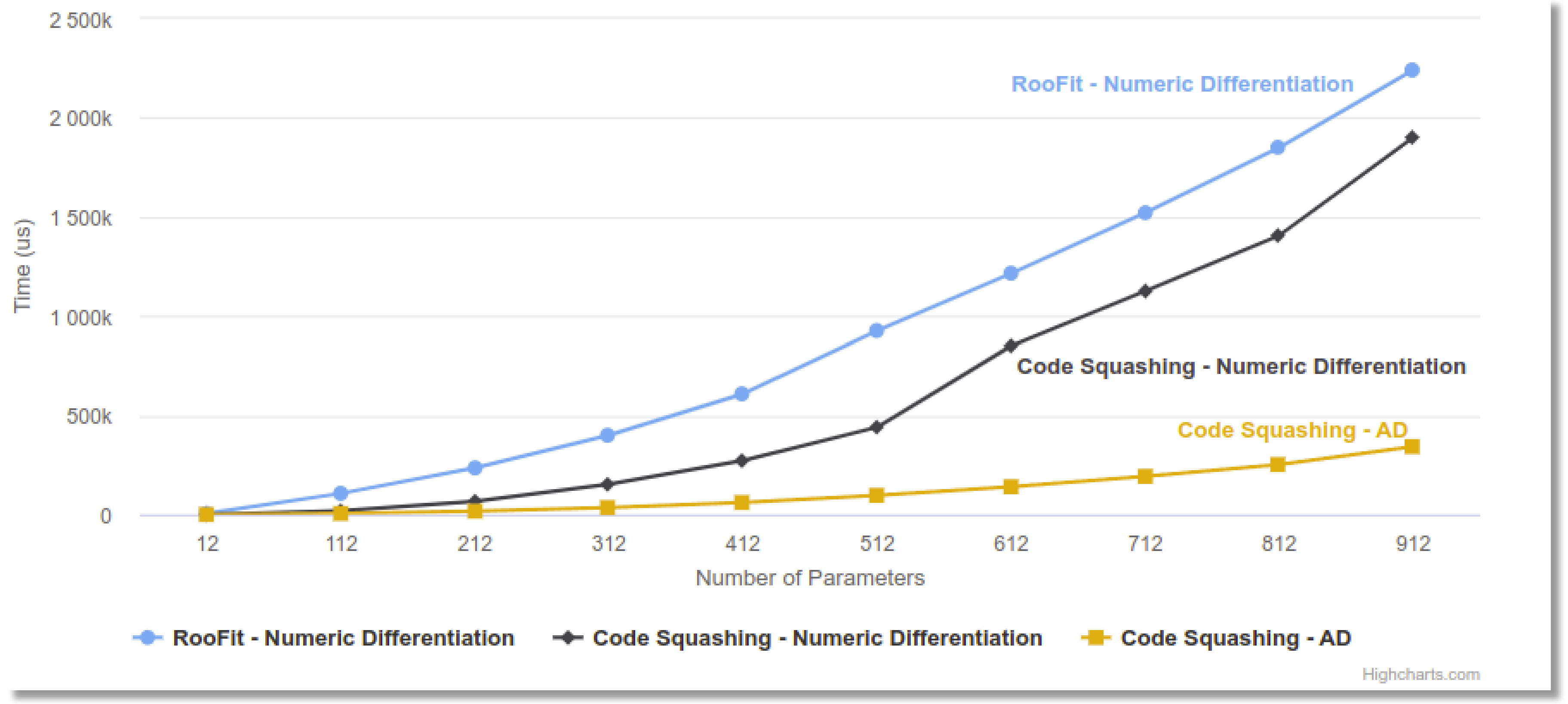
"Demoting" low-sensitivity variables to lower precision improves performance by ~10% in this example.

Clad's estimate also agrees that there is no significant change in the final error. This can be useful in the cases where an accurate ground-truth comparison is not available.

G. Singh, Princeton, *Floating Point Error Estimation Using AD*, SIAM UQ22

CaaS. Exact and Fast Gradients With Clad





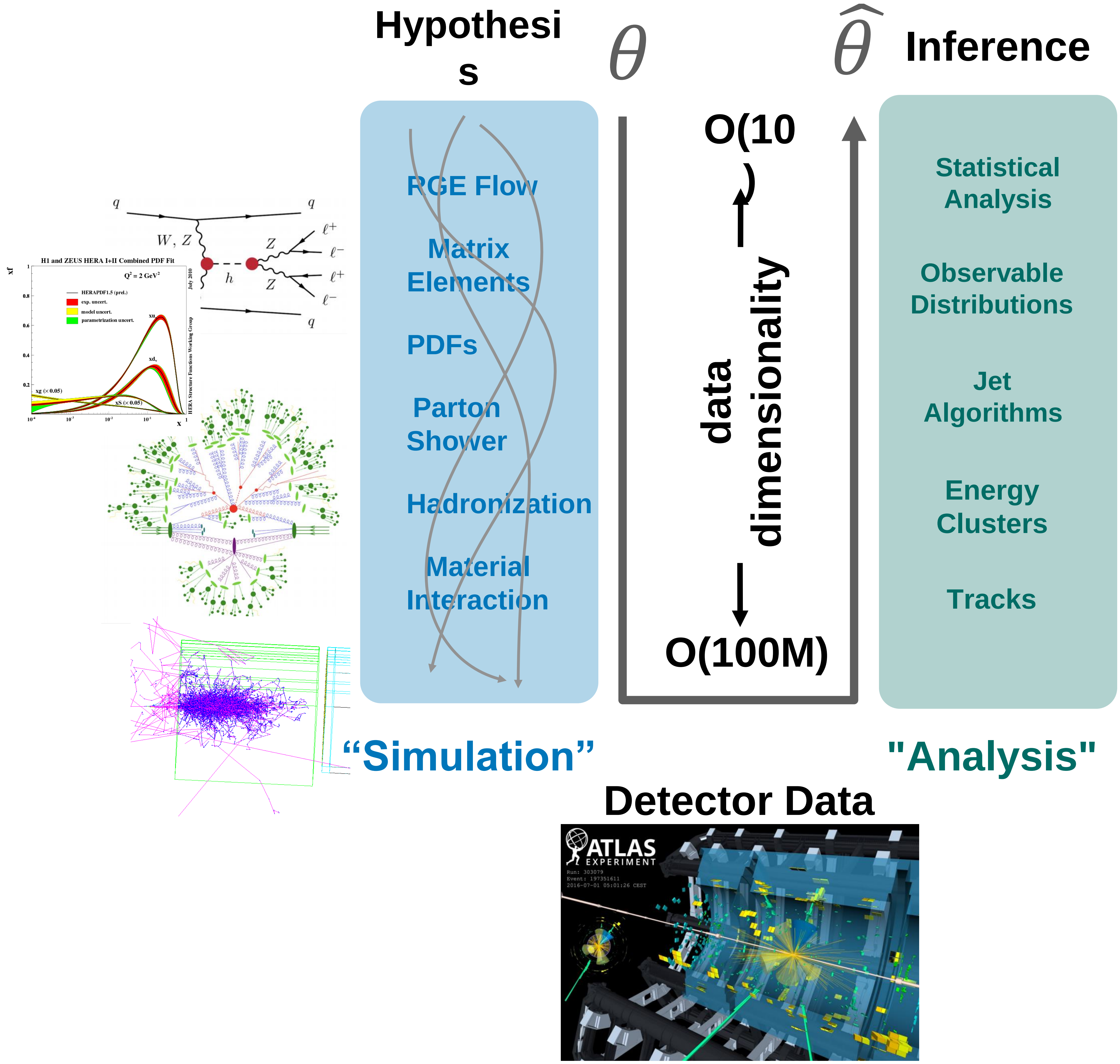
G. Singh, Princeton, [Automatic Differentiation of Binned Likelihoods With Roofit and Clad](), *Wed,* ACAT 22

While speeding up RooFit, after completion of the project we will be able to ask:
How sensitive is an output with respect to a given input parameter?

Sensitivity Analysis At Scale

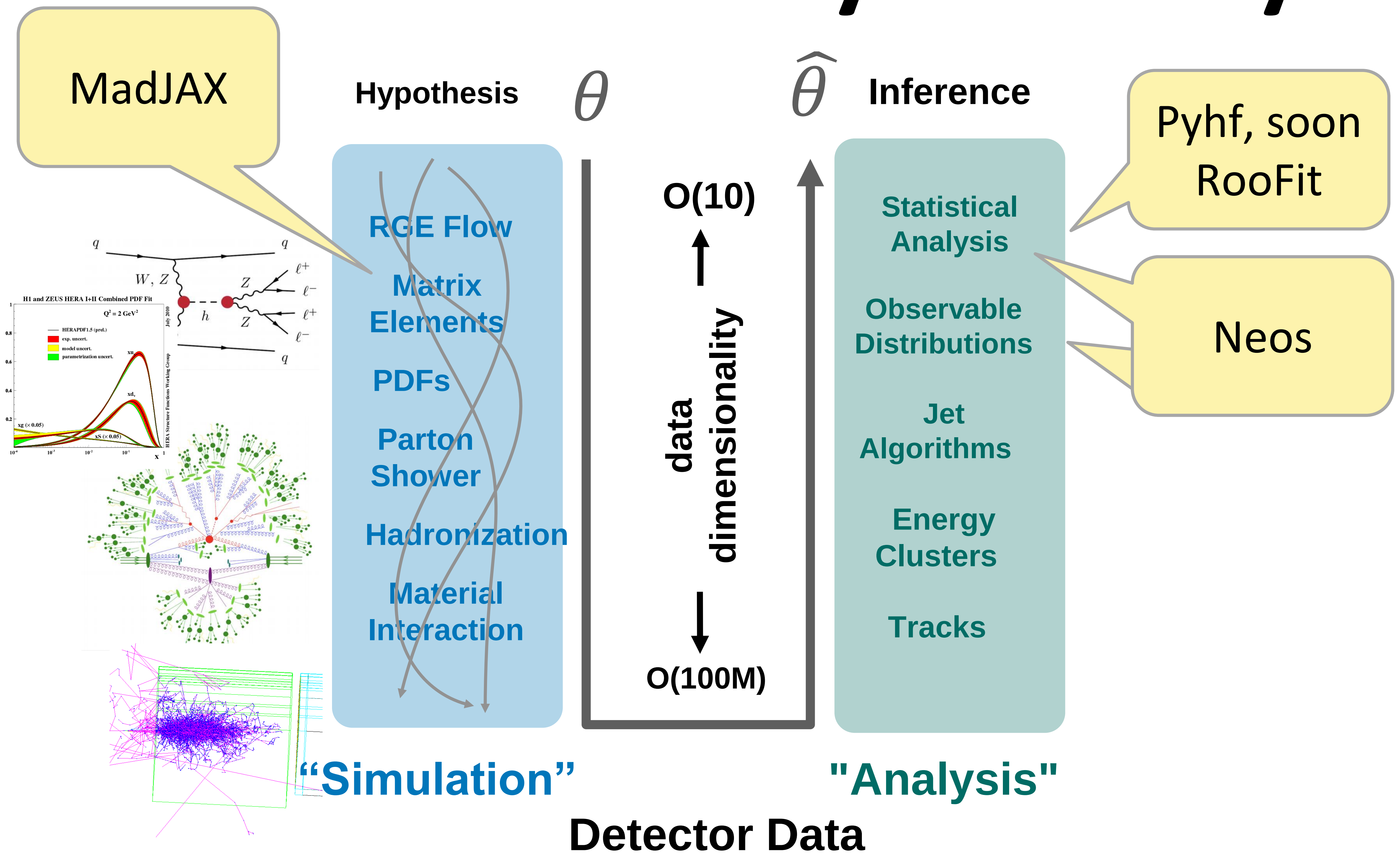
Adapting our hypothesis to the data is an optimization problem

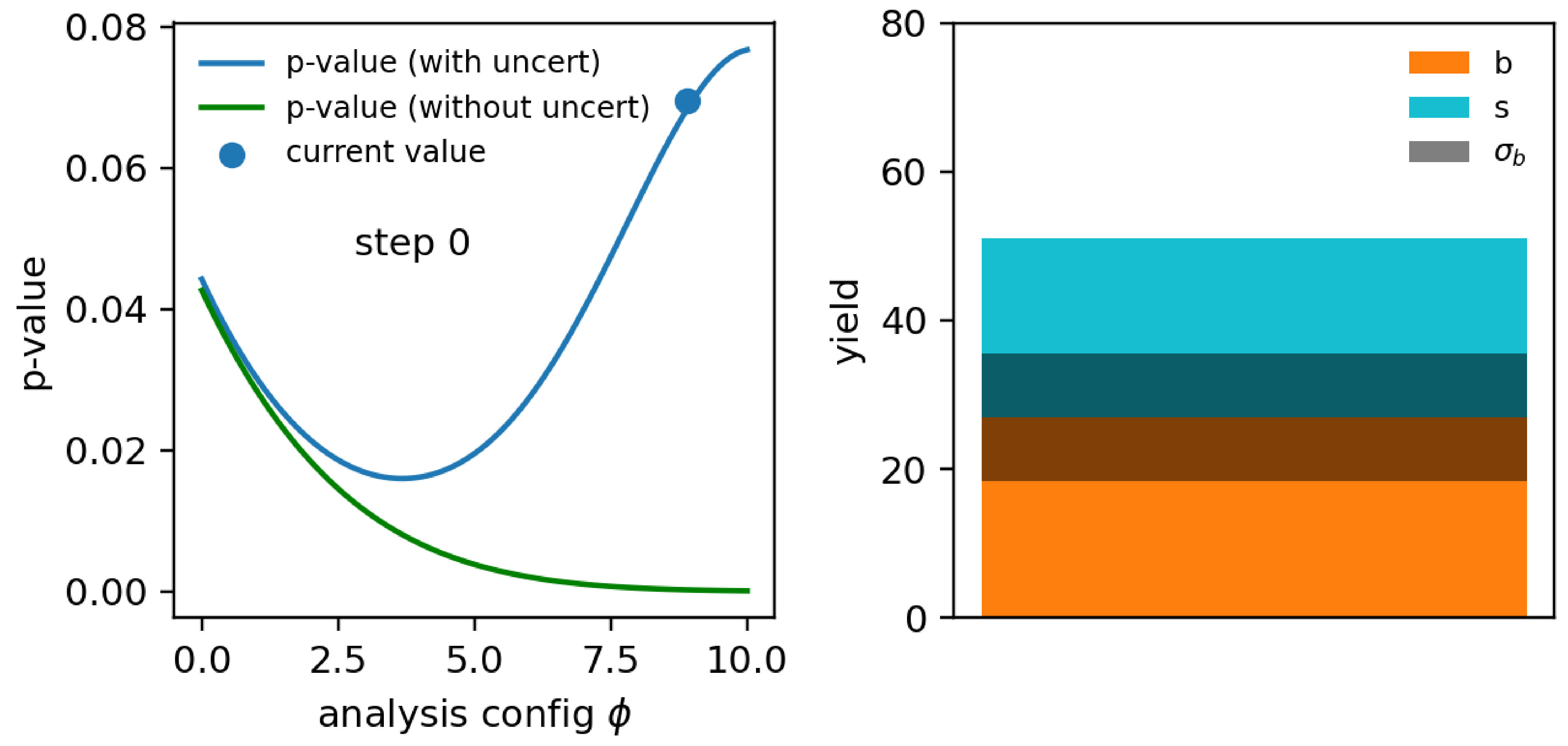
Differential programming is a programming paradigm in which software is susceptible to automatic differentiation.



L. Heinrich, TUM, *Differentiable Programming for High Energy Physics*, *2022*, Future Trends in Nuclear Physics Computing

Sensitivity Analysis At Scale



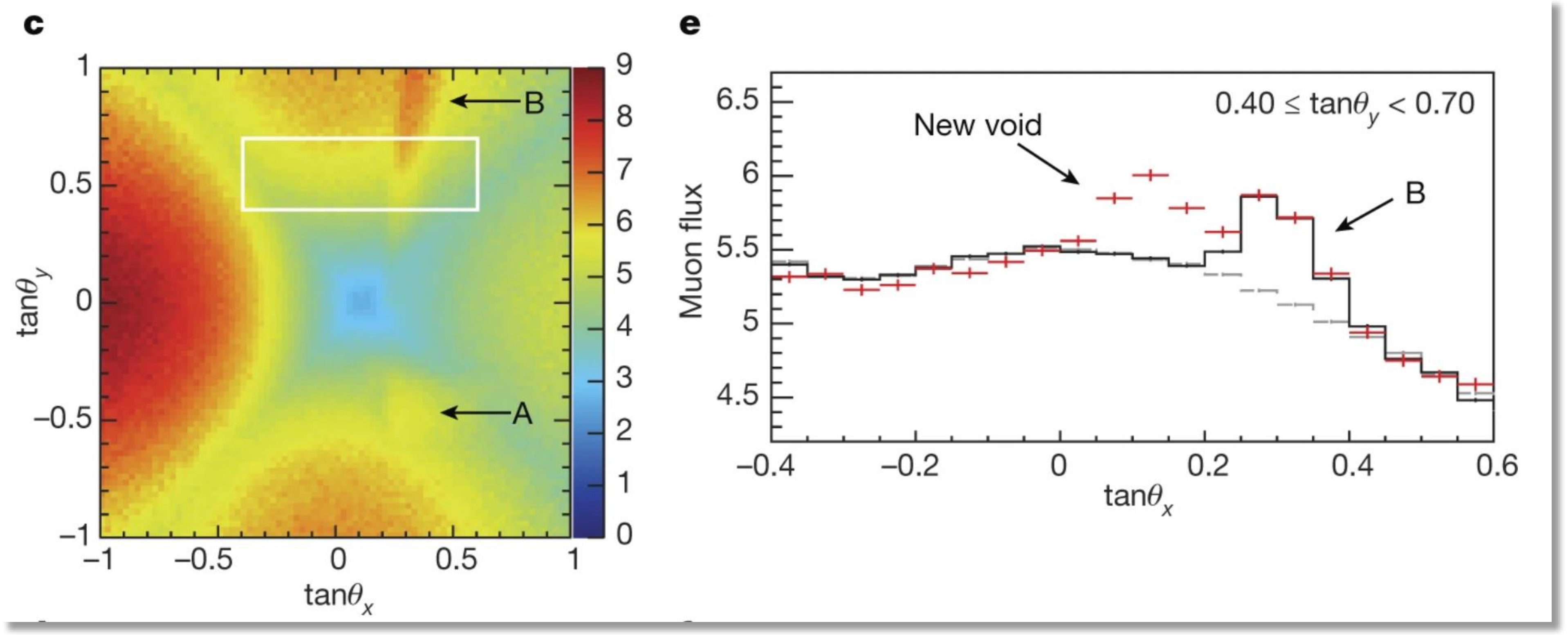


N. Simpson, *differentiable-analysis-examples*,*Single-bin toy analysis optimized to balance uncertainty*

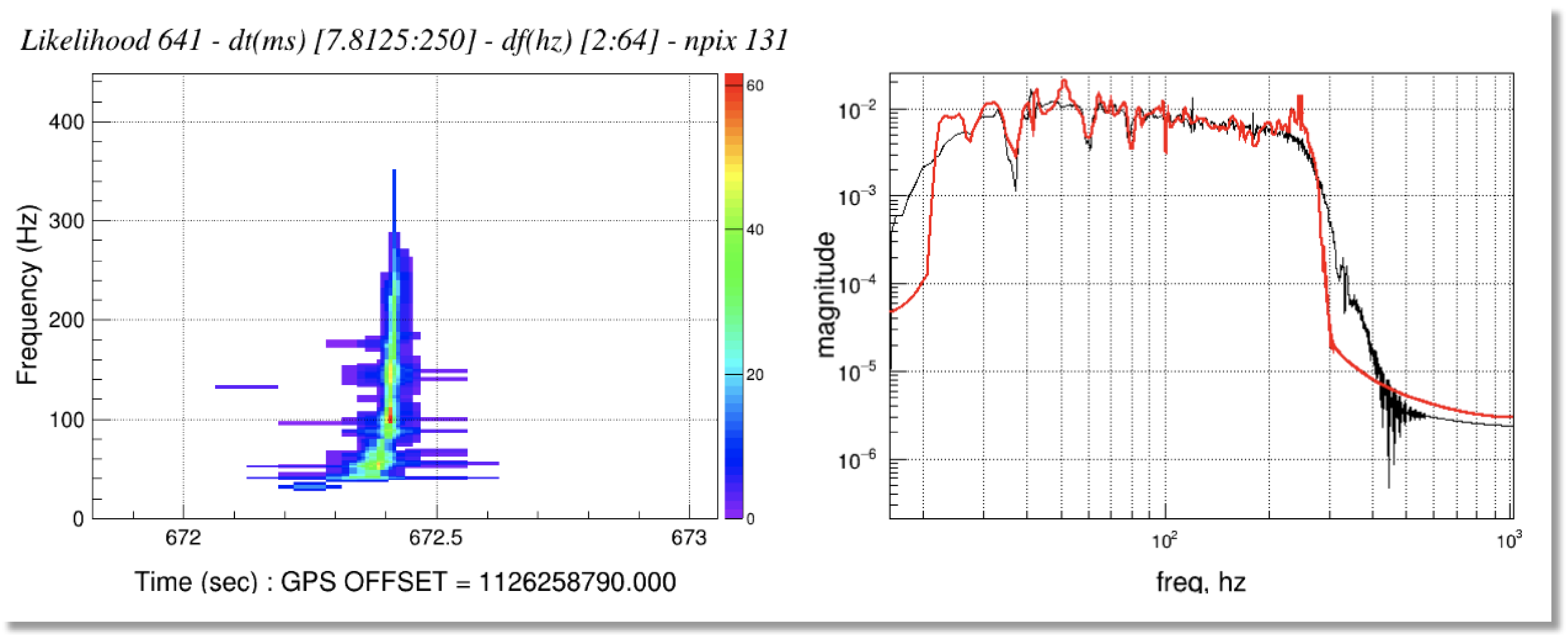
The "Analysis" steps have started moving forward including ROOT.
The "Simulation" steps follow. G4 is the biggest challenge.

Progress in the area will be discussed at **Differentiable and Probabilistic Programming for Fundamental Physics**, in June 2023 in TUM.

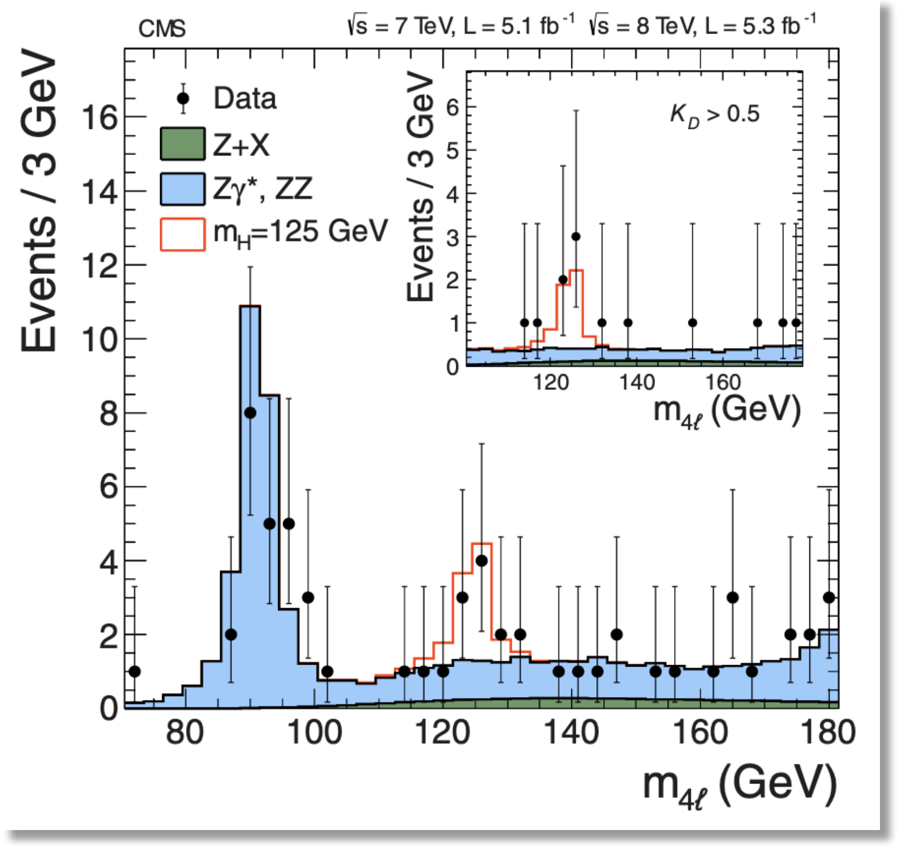
Impact of Interactive C++ in Physics



[1]



[2]



[3]

Scientific breakthroughs such as the discovery of the big void in the Khufu's Pyramid, the gravitational waves and the Higgs boson heavily rely on the ROOT software package which uses interactive C++ and Cling.

[1] K. Morishima et al, **Discovery of a big void in Khufu's Pyramid by observation of cosmic-ray muons**, *Nature*, *2017*
[2] Abbott et al, **Observation of gravitational waves from a binary black hole merger**. *Physical review letters, 2016*
[3] CMS Collab, **Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC**. *Physics Letters B, 2012*

Conclusion

- C++ tools can bring us bare metal performance
- Existing tools can be reorganized and/or generalized with minimal efforts to enable new opportunities
- We should maintain them and grow them focusing on what they are good for
- Our community has unique multi-language expertise that can allow us doing more science with the same budget

Thank You!

Selected References

- https://blog.llvm.org/posts/2020-11-30-interactive-cpp-with-cling/
- https://blog.llvm.org/posts/2020-12-21-interactive-cpp-for-data-science/
- https://blog.llvm.org/posts/2021-03-25-cling-beyond-just-interpreting-cpp/
- https://Compiler-Research.org
- https://root.cern
- Interactive C++ for Data science, CppCon21
- Differentiable programming in C++, CppCon21

https://github.com/vgvassilev/

https://www.linkedin.com/in/vgvassilev/