Introducing cling

Axel Naumann, CERN

@ Google Zurich, 2012-03-15

cling?

C++

cling?

LHC++

cling?

cling$ LHC++

cling?

cling$ LHC++
(int const) 42

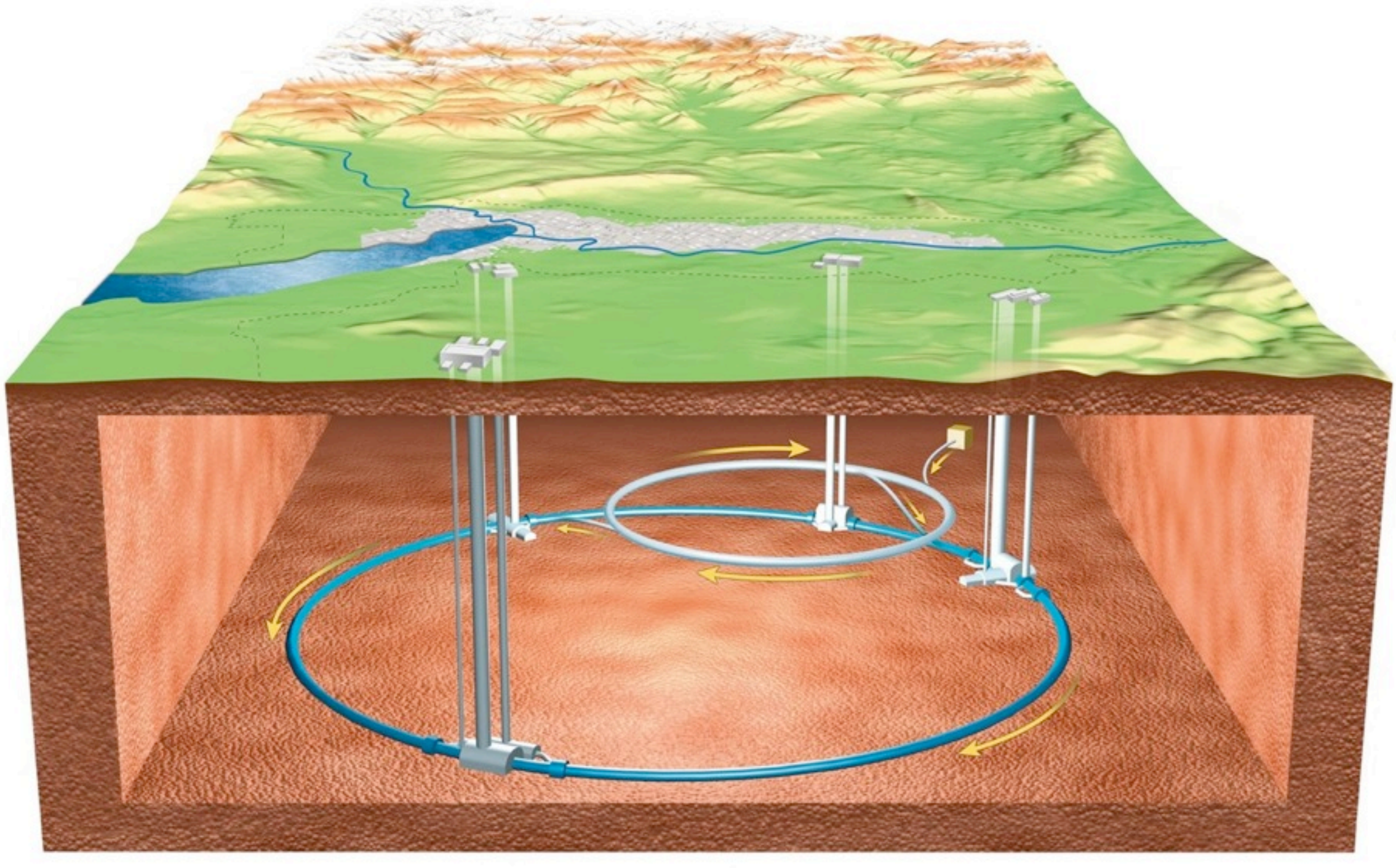
CERN, LHC

- fundamental research: high energy physics
- international organization in Geneva, CH
- main tool: Large Hadron Collider
  - proton smasher's measurements: 8TeV, 2K, 27km
  - several experiments, 10'000 users

Postcard From CERN

Postcard From CERN

Underground Science



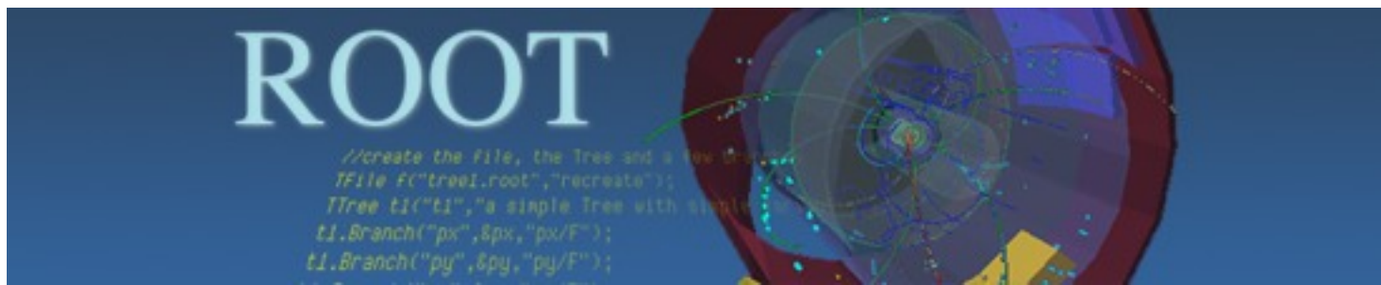
Big Questions, Big Tools

The C++ in CERN

- data analysis = interfacing with experiments' code
- several GB of libraries, hundreds of thousands of types / templates, 50MLOC C++ code
- physics is the goal, computing the tool

http:// ROOT .cern.ch

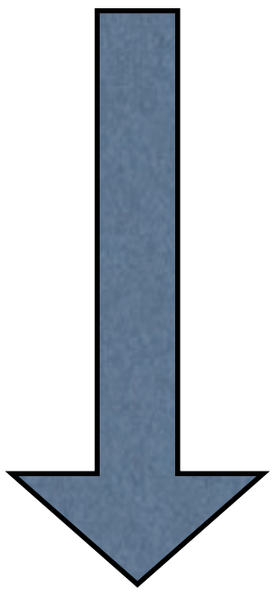
- data analysis (math), persistency (I/O), visualization (graphics),...
- about 20’000 users, also outside science
- core software element for experiments
- interface point for experiment’s code
- C++ interpreter CINT almost 20 years old



Use of Interpreters

1970

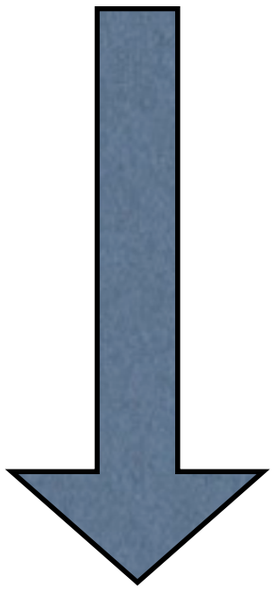
access data



today

Use of Interpreters

1970



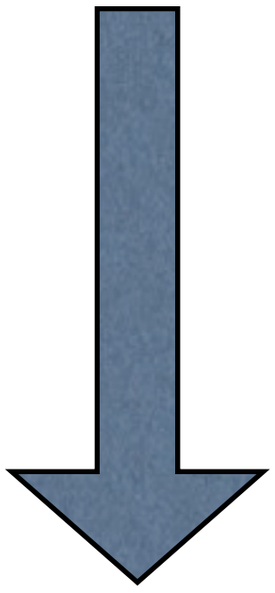
today

access data

simple expressions

Use of Interpreters

1970



today

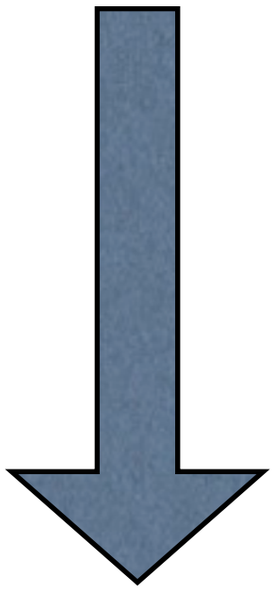
access data

simple expressions

simple algorithms

Use of Interpreters

1970



today

access data

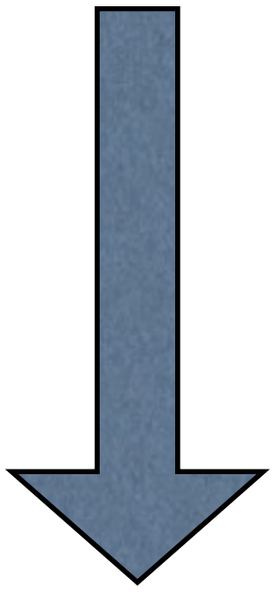
simple expressions

simple algorithms

prototype algorithm

Use of Interpreters

1970



today

access data

simple expressions

simple algorithms

prototype algorithm

develop code

Setup

Experiment's code

Third-party libraries

Persistent data

User code

Interpreter

Current Use

- IDE-on-the-prompt
- interpreter = not linking + state
- rapid edit / “compile” / run cycles
- exploration-driven development
- matches physics analysis approach with its gradual optimizations

cling: interpreting C++

- http://cern.ch/cling
- based on LLVM http://llvm.org
  + clang http://clang.llvm.org
- developed by team @ CERN and Fermilab
- maps interpreter to compiler concept
- not an interpreter: JIT!

cling's Translation Unit

- AST keeps growing as input comes
- incremental parsing, codegen, evaluation
  - action at end of translation unit now at end of input transaction
    - pending template instantiation
    - codegen

Expr vs Decl

```
cling$ int i = 1; sin(i++)
```

- decls must stay visible across input lines
- expressions must be evaluated
- ...and is it decl or expr or both?

Expr vs Decl (2)

```
cling$ int i = 1; sin(i++)
```

1. determine whether decl or expr:
   manipulation of input string

```
void wrap_0() {
  int i = 1; sin(i++); }
```

Expr vs Decl (3)

```
cling$ int i = 1; sin(i++)
```

```
void wrap_0() {
  int i = 1; sin(i++); }
```

2. extend decl lifetime: move onto global scope (AST editing)

```
int i = 1;
void wrap_0() { sin(i++); }
```

Expressions

```
int i = 1;
void wrap_0() { sin(i++); }
```

3. call wrap_0() to evaluate expressions
4. print value if no trailing ';'
   uses template magic on AST level:

```
cling$ int i = 1; sin(i++)
(double const) 8.414710e-01
```

Details, details

- global initialization after each input - but not re-initialization
- collect global destructors to run at ~cling()
- error recovery reverting whole input transaction and its AST nodes

Dynamic Scopes

```
if (date % 2) {
  TFile f(getFilename());
  objInFile->Draw();
}
```

- inject serialized C++ objects into scope
- delay expression evaluation until runtime
- compiler as a service - sort of like DLR

JIT

- optimized code
- ABI-compatible:
  in-memory layout vs serialization
- calls into native libraries

```
$ echo 'const char* zlibVersion();
zlibVersion()' | cling -x c -lz
(const char * const) "1.2.3.4"
```

Reflection

- weak point in C++
- can tap clang's AST!
  + target info!
  + ABI!
- dynamic, two ways: query + edit AST

Growing cling

- clang vs. Windows C++ ABI!
- clang as front-end, thus C++11
- ObjC[++] can be extended (not by CERN)
- OpenCL could be done
- or new front-ends? LINQ, anyone?

Where?

- available in subversion:
  http://root.cern.ch/svn/root/trunk/cint/cling
- stable: few interface changes, only 10k LOC
- works on anything with clang + LLVM-JIT
  (thus not native Windows ABI)
- stand-alone binary plus modular C++
  libraries like LLVM + clang

Where?

- available in subversion: http://root.cern.ch/svn/root/trunk/cint/cling
- stable: few interface changes, only 10k LOC
- works on anything with clang + LLVM-JIT (thus not native Windows ABI)
- stand-alone binary plus modular C++ libraries like LLVM + clang

Interpreter Language

- C++: complex syntax, verbose, precise; improvements from C++11
- Current #1 alternative python: simple syntax, casual code

PyROOT

- bi-directional

```
TPython::Exec("print \'PyROOT!\'");
```

```
from ROOT import TLorentzVector
b = TLorentzVector()
b.SetX(1.0)
```

- map concepts (iterators, dictionary)

PyROOT Internals

- reflection-, not stub-based: highly dynamic
- injects C++ methods into python
- objects traverse language boundary
- performant: caches, annotates python objects with PyROOT metadata

iC++ vs iPython

- migrating python code to C++: difficult
- interfacing C++ though python: difficult
- writing 1 MLOC python: easy
- python is slow ("thanks to python, we are not I/O bound anymore!")

Conclusion: Interpreter

- interpreters enable dynamic access to huge binary worlds
- different approach to programming
- reflection is the key: interpreter binding, serialization, dynamic behavior

Conclusion: Language

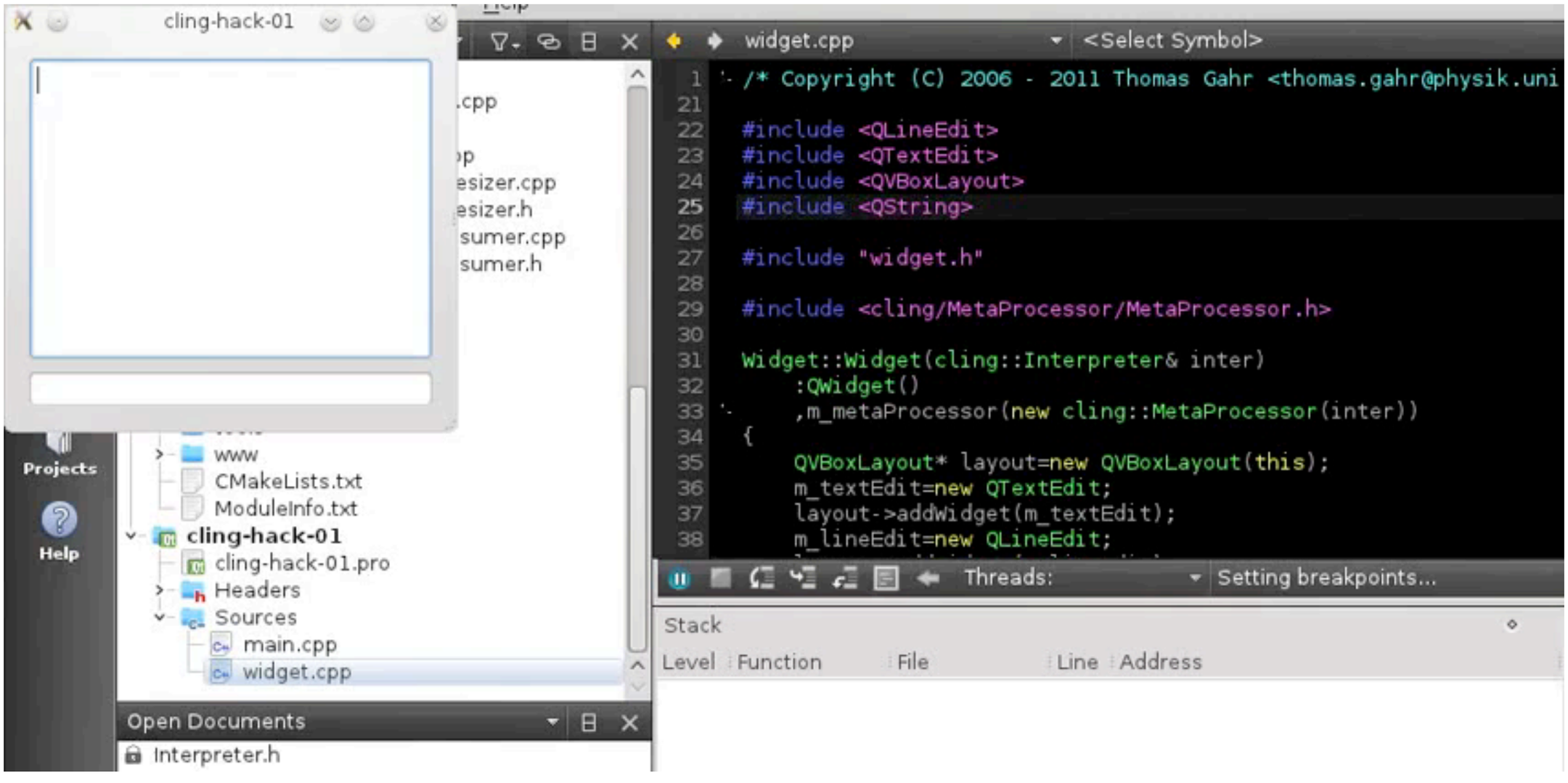
- choice of languages still limits us
- python is simple but slow; interface with C++ difficult but possible
- C++ easily too complex for novices
- either language dead-end: no interfaces from other languages (extern “C++”)

Conclusion: cling

- based on decades of experience with novices, code development, large libraries - and CINT
- clang + llvm make miracles happen
- it's stable and fun: enjoy it!

Offline Demo

- demo by “arbitrary (smart) user” Thomas Gahr showing powers of Qt + cling
- http://youtu.be/BrjV1ZgYbbA
- recursive youtube!



http://youtu.be/BrjV1ZgYbbA