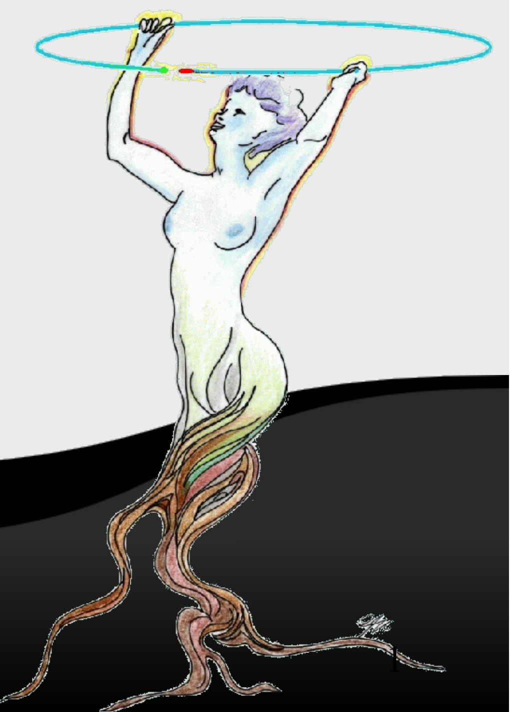
Interactive, Introspected C++ at CERN

Vassil Vassilev



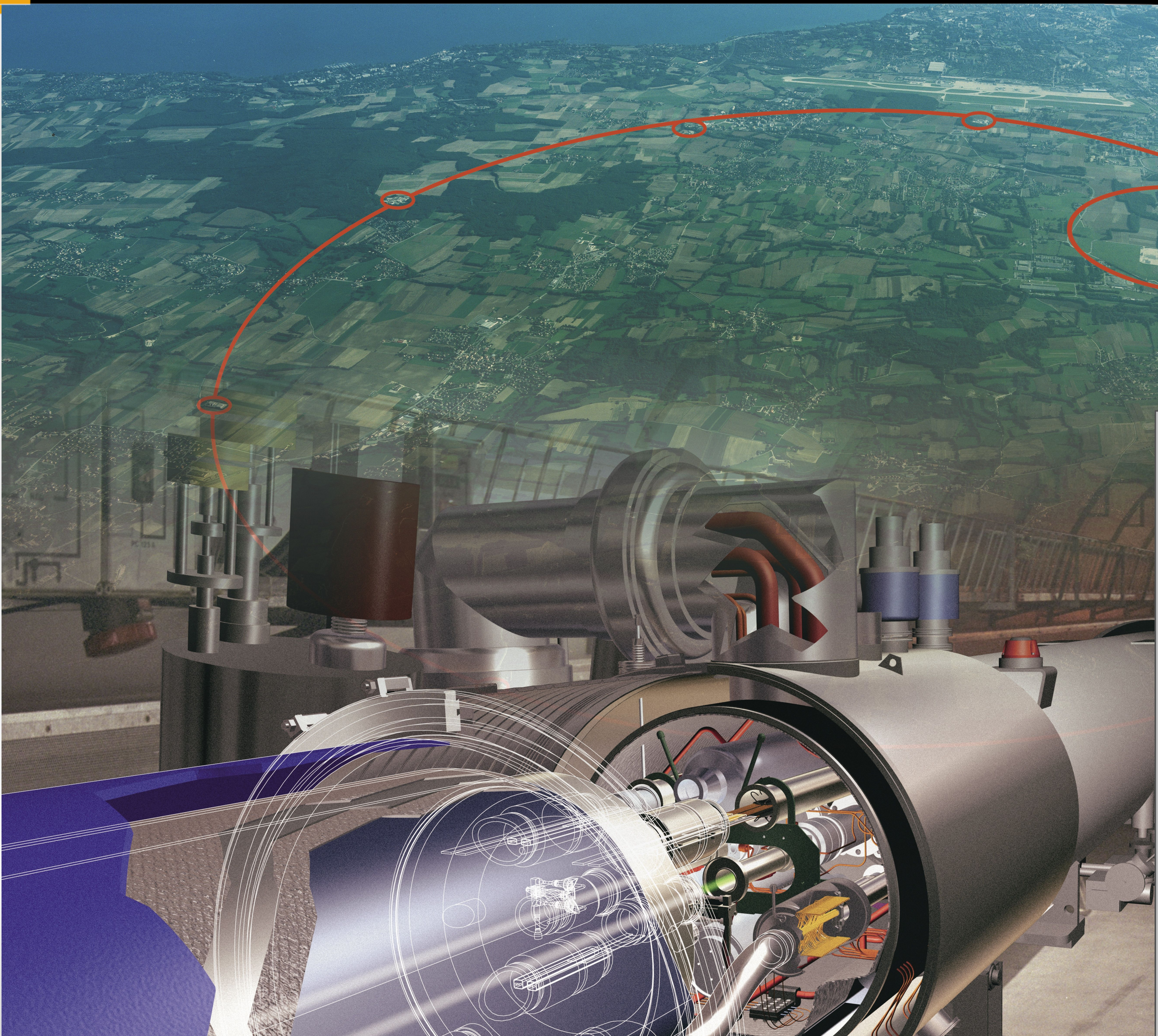
Vassil Vassilev
vvasilev@cern.ch

CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

CERN, PH-SFT


ANGELS&D
MAY 200
© 2009 Sony Pictures Digital Inc. All Ri

CERN Overview



Situated in Geneva, Switzerland

~ 3500 Members of personnel

~ 10500 Users

World’s biggest high-energy physics lab

20 Member States

Experiments at CERN

More than 20 different experiments:

ACE

AEGIS

ALICE

ALPHA

AMS

ASACUSA

ATLAS

ATRAP

CAST

CLOUD

CMS

COMPASS

DIRAC

ISOLDE

LHCb

LHCf

MOEDAL

NA61/SHINE

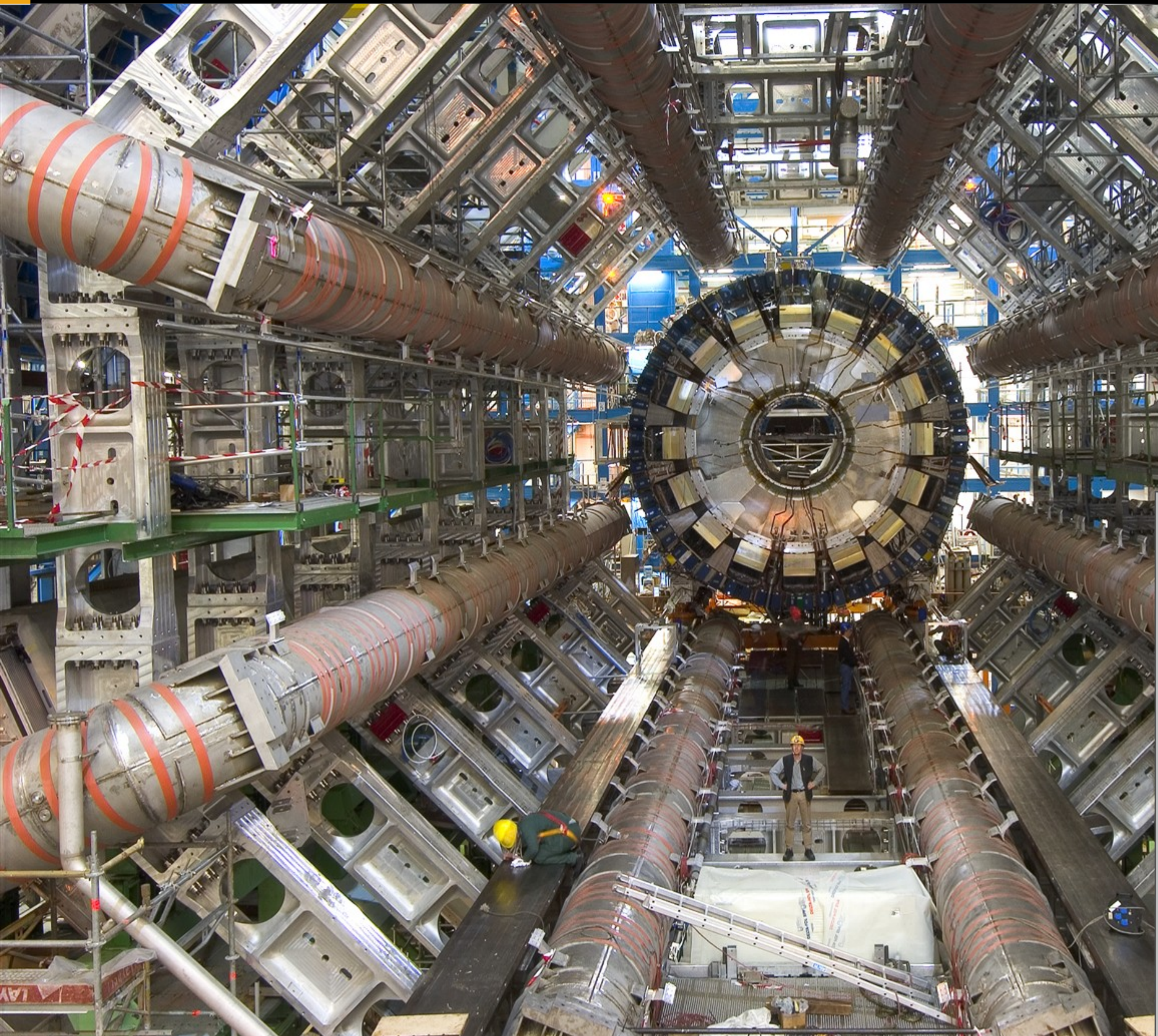
NA62

nTOF

OSQAR

TOTEM

ATLAS Experiment



People: > 3000

Offline software:
5M LOC C++

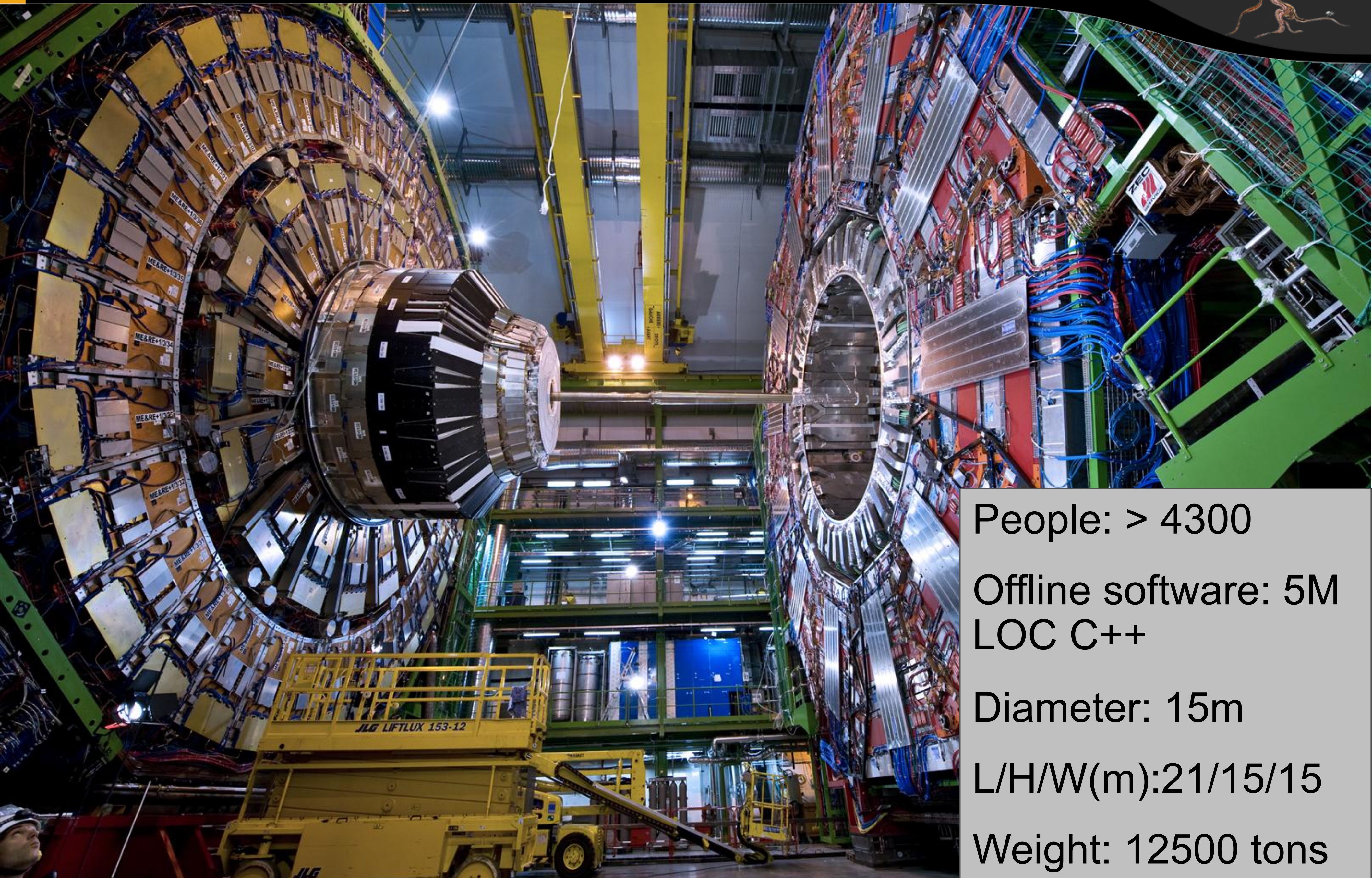
Diameter: 25m

L/H/W(m): 46/25/25

Weight: 7000 tons

3000km cables

CMS Experiment



People: > 4300

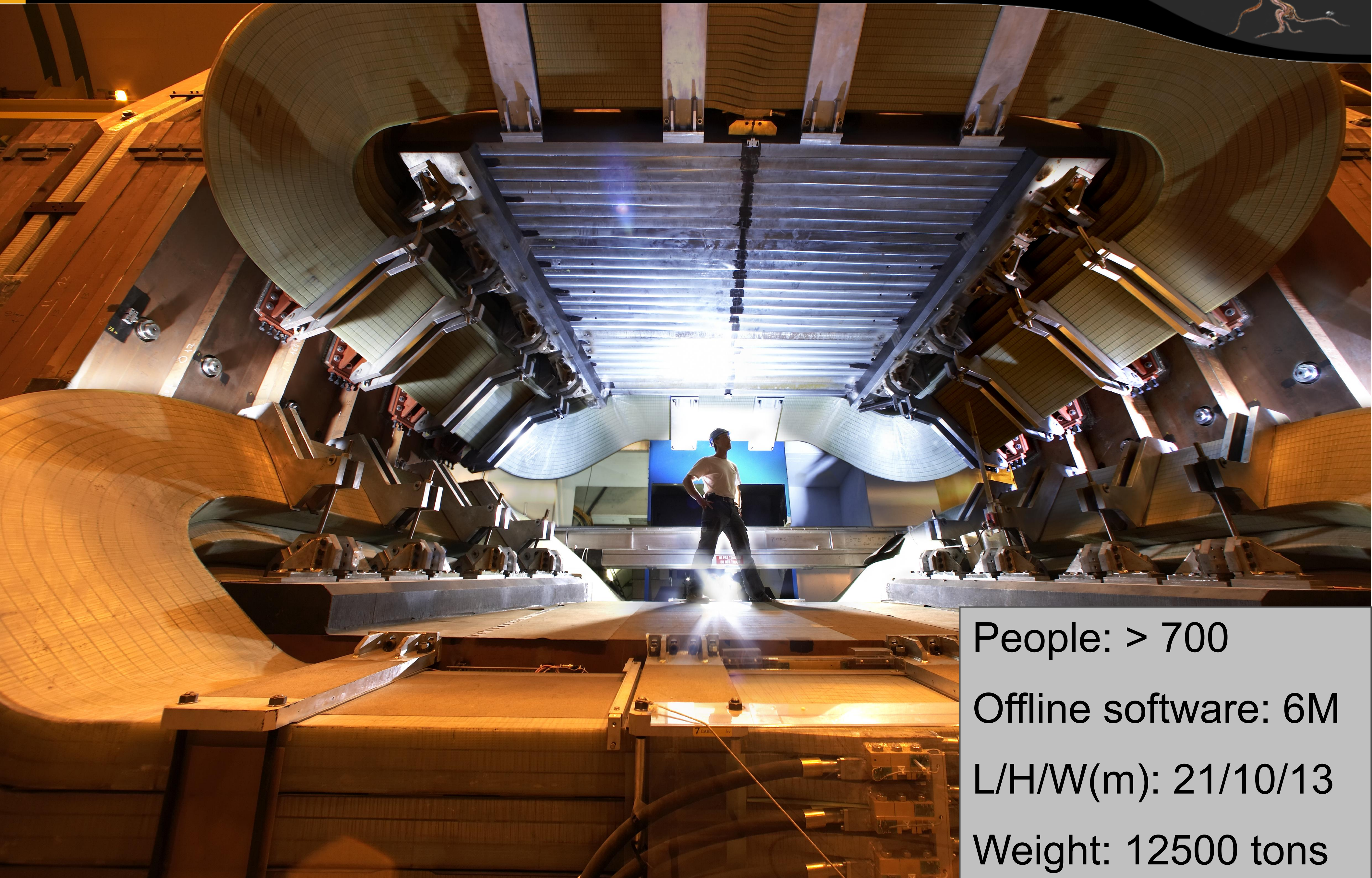
Offline software: 5M LOC C++

Diameter: 15m

L/H/W(m):21/15/15

Weight: 12500 tons

LHCb Experiment



People: > 700

Offline software: 6M

L/H/W(m): 21/10/13

Weight: 12500 tons

ALICE Experiment

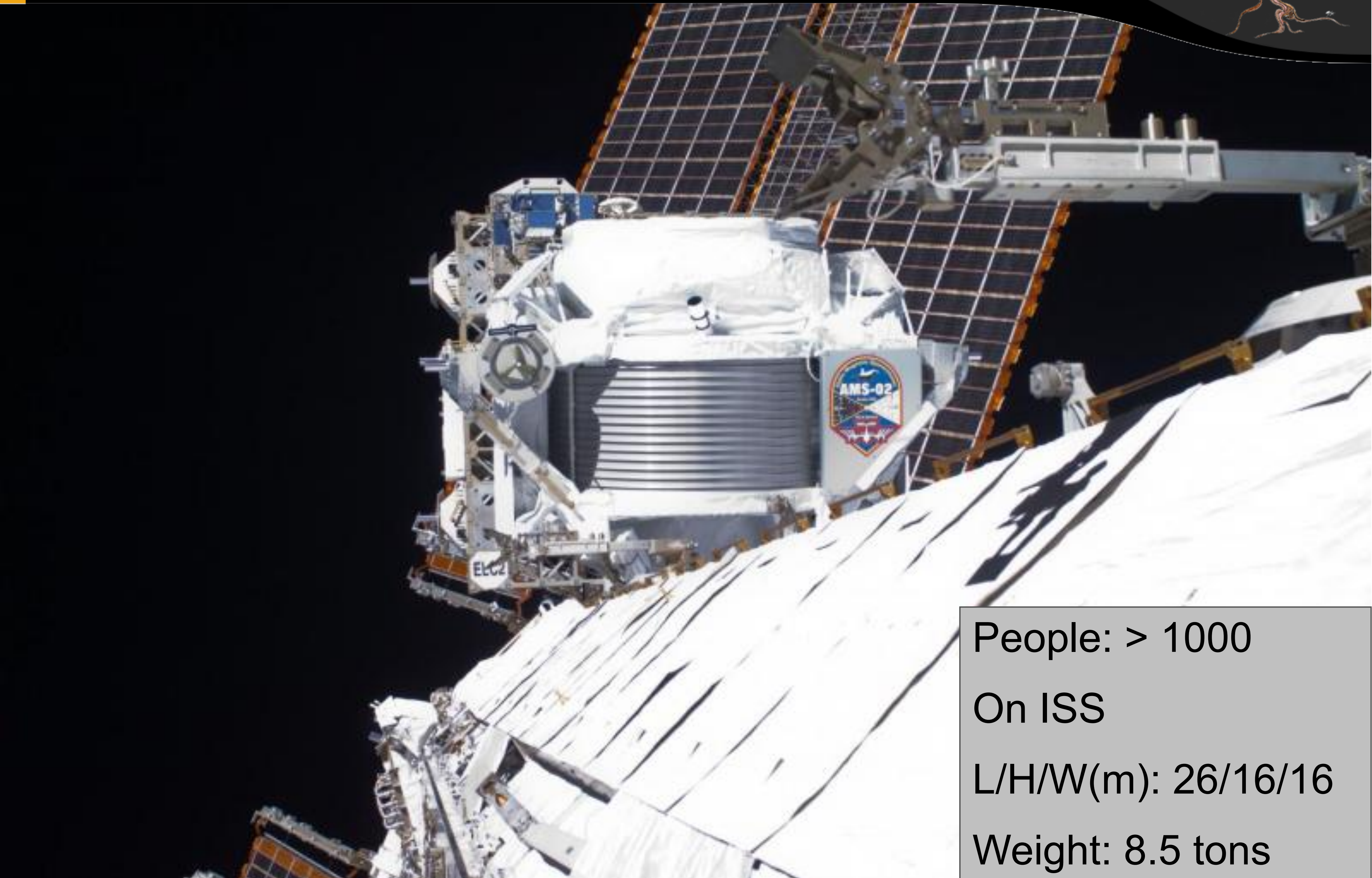
People: > 1000

Offline software: 1M

L/H/W(m): 26/16/16

Weight: 10000 tons

AMS Experiment



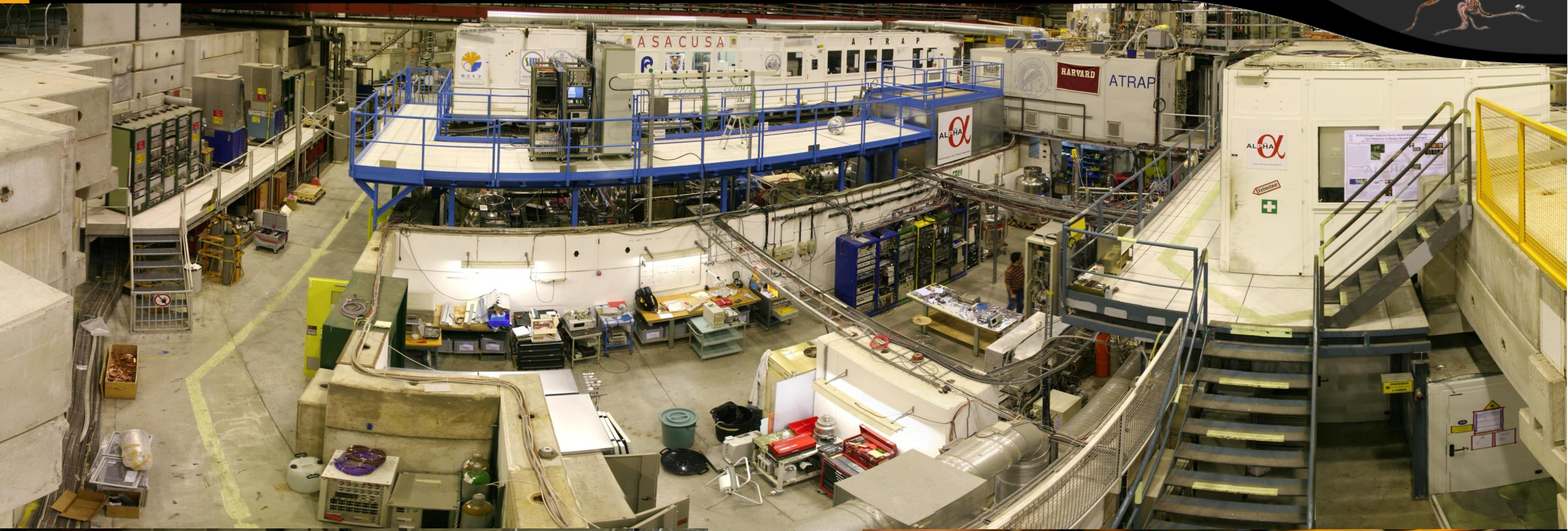
People: > 1000

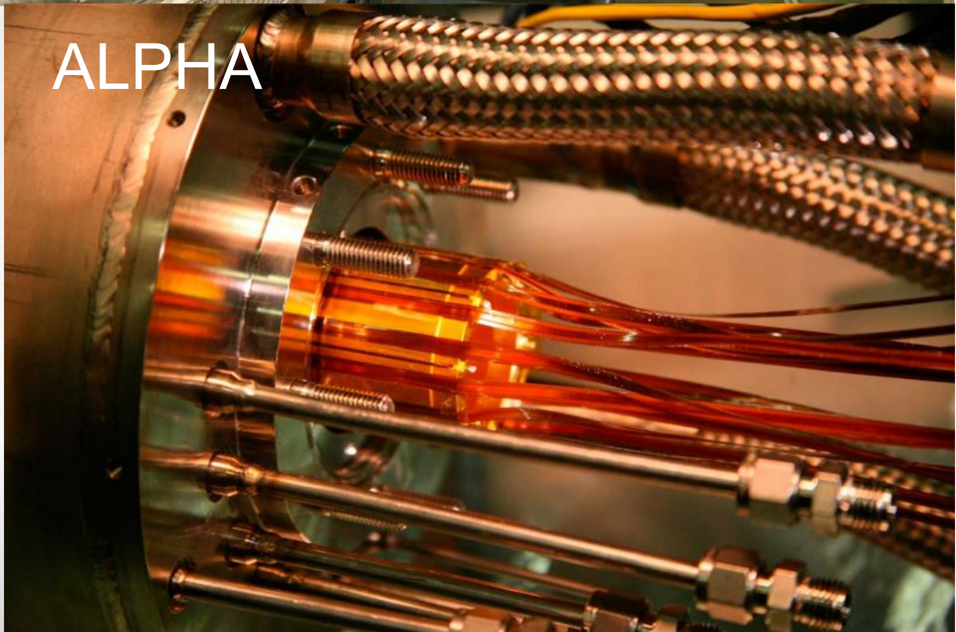
On ISS

L/H/W(m): 26/16/16

Weight: 8.5 tons

Antiproton Decelerator



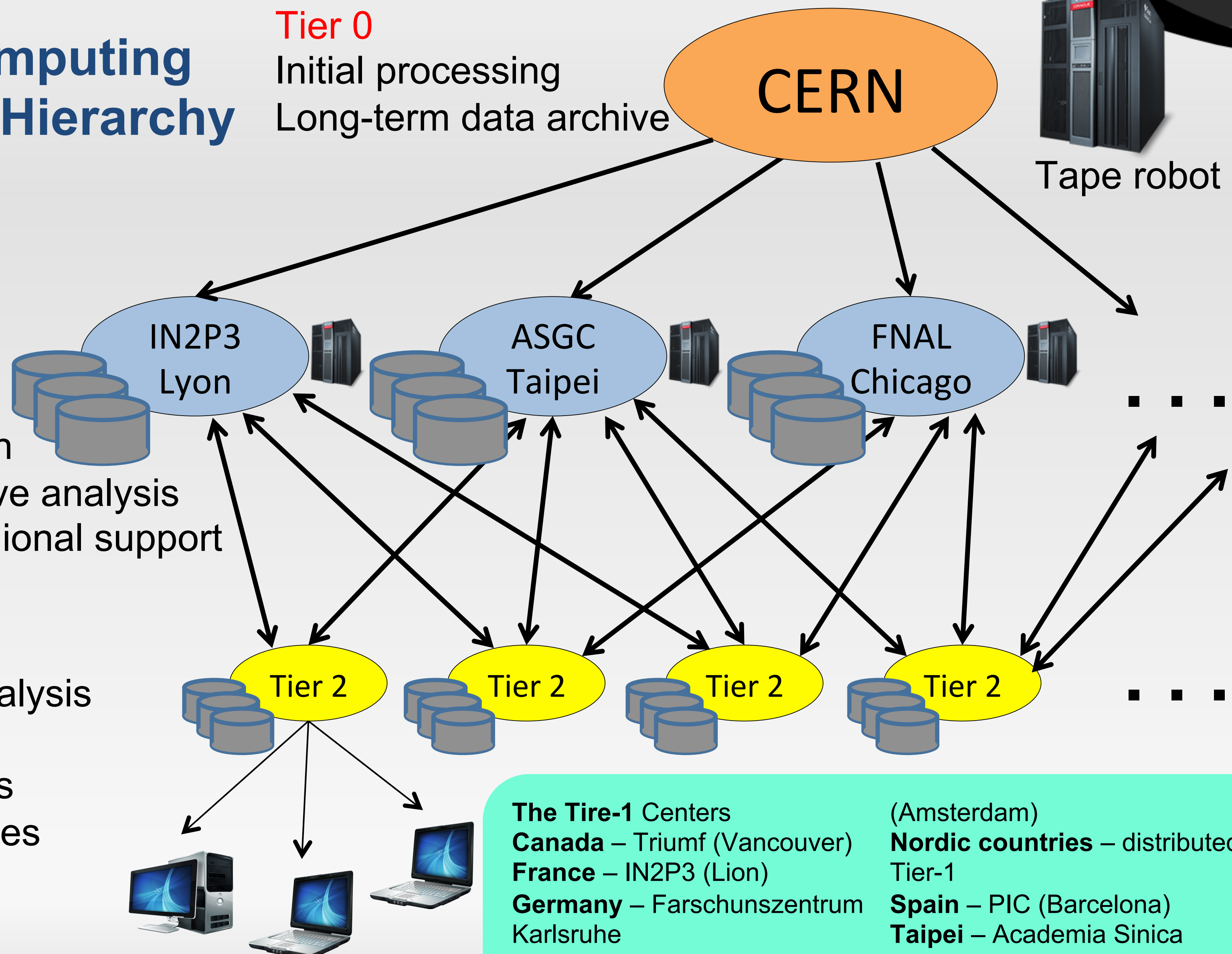


CERN Data Flow (video)

Worldwide LHC Computing Grid

LHC Computing Service Hierarchy

Tier 0
Initial processing
Long-term data archive

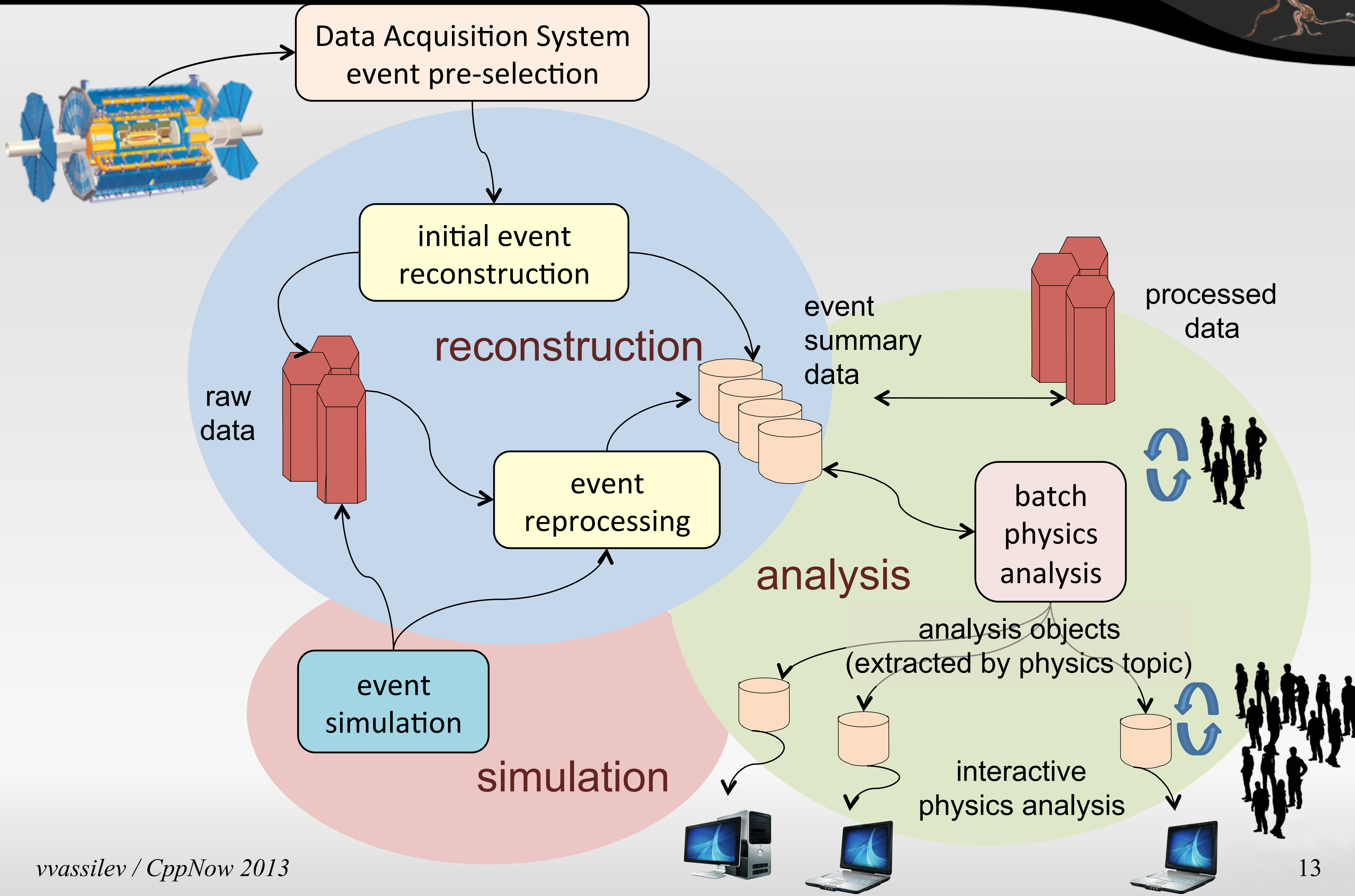


Tier 1s
data curation
data-intensive analysis
national, regional support

Tier 2s
end-user analysis
Simulation
~130 centers
in 33 countries

The Tire-1 Centers
Canada – Triumf (Vancouver)
France – IN2P3 (Lion)
Germany – Farschunszentrum Karlsruhe
Italy – CNAF (Bologna)
Netherlands – NIKHEF/SARA (Amsterdam)
Nordic countries – distributed Tier-1
Spain – PIC (Barcelona)
Taipei – Academia Sinica
UK – Rutherford Lab (Oxford)
US – FermiLab (Illinois)
US – Brookhaven (NY)

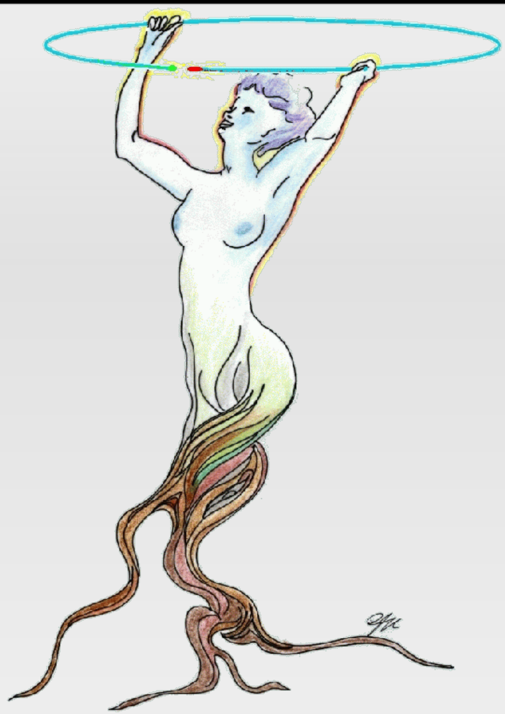
Data Workflow



Physicists and C++

- Physicists' analysis is C++ program written by physicists
- Interfaces with many of libraries:
  - Experiment's
  - CERN-provided
  - External
- Many novices in computing

The ROOT Framework



"ROOT is a framework for data processing, born at CERN, at the heart of the research on high-energy physics. Every day, thousands of physicists use ROOT applications to analyze their data or to perform simulations."

The ROOT Framework Overview

- **Started in 1995**

  By Rene Brun and Fons Rademakers. ROOT has continued to develop and evolve for almost two decades.

- **Single Language Concept**

  Use C++ as a common denominator.

- **Applications outside HEP**

  ROOT is being used in domains different from High Energy Physics (HEP), such as Finance and Astronomy.

- **Open source**

The ROOT Framework

At the boundary between physics and computing:

- **Save data**
  Serialization/Deserialization of data represented as C++ objects in very efficient data structure optimized for fast readout.
- **Access data**
  Self-describing data structures (ROOT Files), being able to be chained.
- **Process data**
  Powerful tools for analysis, parallel processing and simulation.
- **Show results**
  Sophisticated graphics subsystem.
- **Interactively build apps**
  Exploratory programming, REPL concept, Quick and easy prototyping, Interactive shell-like prompt

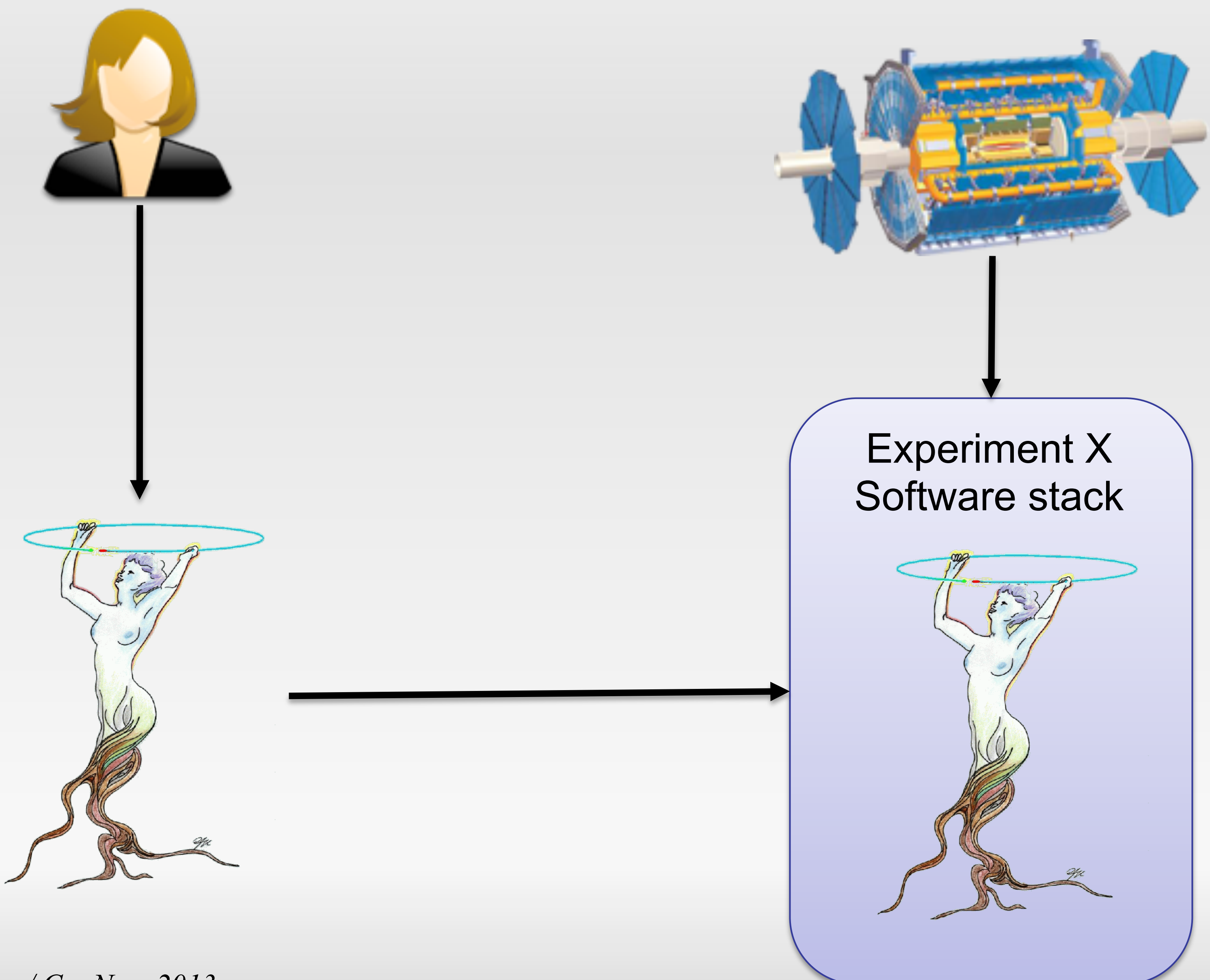
The ROOT Framework Overview

More than 1200 classes, grouped in around 60 libraries in 19 main categories:

- Containers
- Physics
- Matrices
- Histograms
- Minimization
- Tree and n-tuples
- 2D graphics
- 3D graphics
- Image processing
- Detector geometries
- C++ support (interpreter)
- Networking
- SQL
- …

In total 3.5M SLOC C++

ROOT Usage in HEP

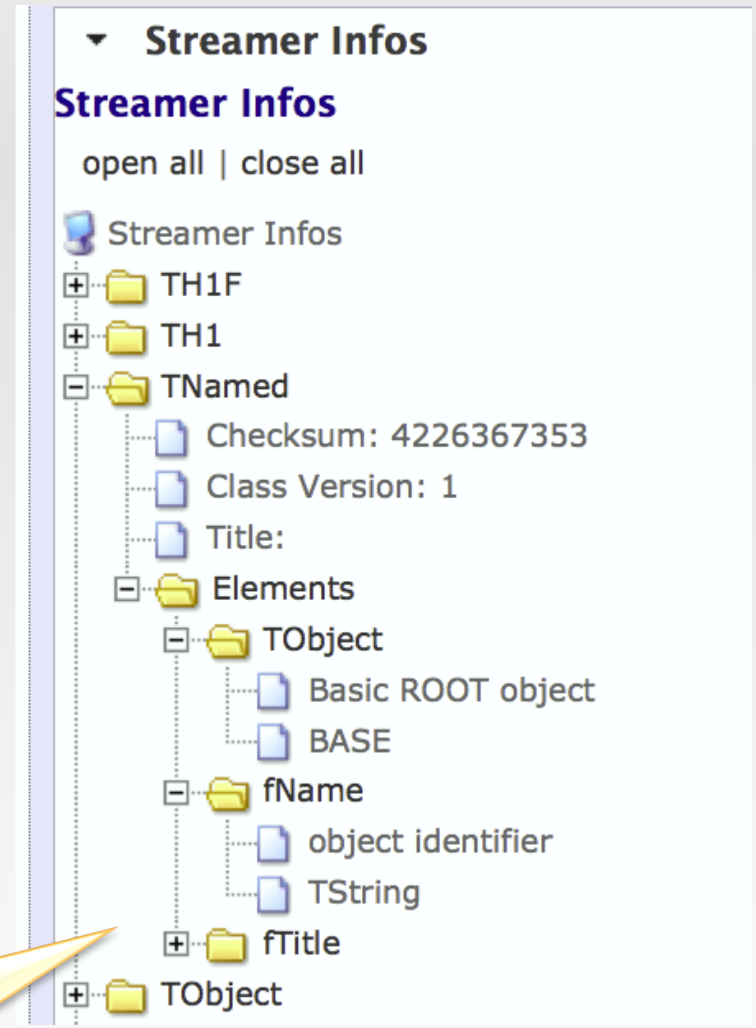


ROOT Files

- Write-Once-Read-Many
- Store about 180 PBs
  Almost all experiments' physics data
- Optimized for sequential, aggregated reading
  Resulting in "disk-friendliness" and low latencies due to less requests/ round trips.
- Machine/Language/Architecture Independent
  There is ROOT file reader implemented in JavaScript
- Self-describing
  ROOT files describe their content and the way it should be read

ROOT Trees in Files

- Vertical data storage, recursively
- High performance reading
- Better compression



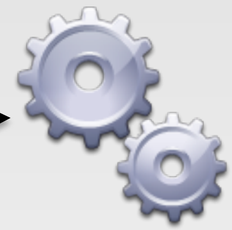
JS ROOT file reader
(at http://root.cern.ch/js/)

ROOT Trees in Files

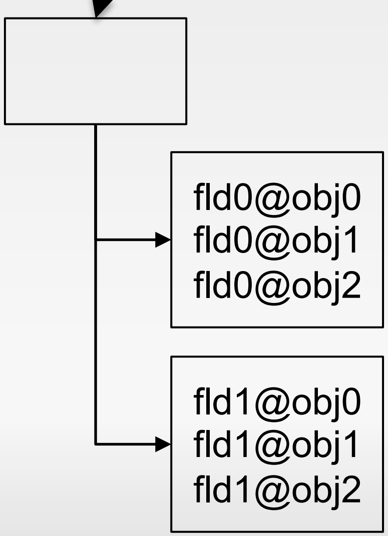
```
class MyClass {
private:
int fld0; // IO annotations
double fld1; // IO annotations

public:
int getFld0() {return fld0;}
void setFld0(int v) {fld0 = v;}
double getFld1() {return fld1;}
void SetFld1(double v) {fld1 = v;}
} obj[3];
```

MyClass.root



Compiled libs(.so)



Objects' data transformed into tree:

- Allows rebuilding the class layout without library
- Allows selective reading
- Allows code changes (schema evolution)

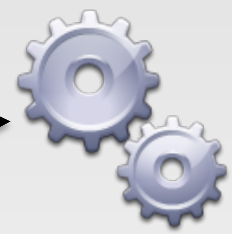
Reflection Layer in ROOT

```
class MyClass {
private:
int fld0; // IO annotations
double fld1; // IO annotations

public:
int getFld0() {return fld0;}
void setFld0(int v) {fld0 = v;}
double getFld1() {return fld1;}
void SetFld1(double v) {fld1 = v;}
} obj[3];
```

MyClass.root

TClass
`IsA()`
`GetListOfBases()`
`GetListOfMethods()`
`...`



Compiled libs(.so)

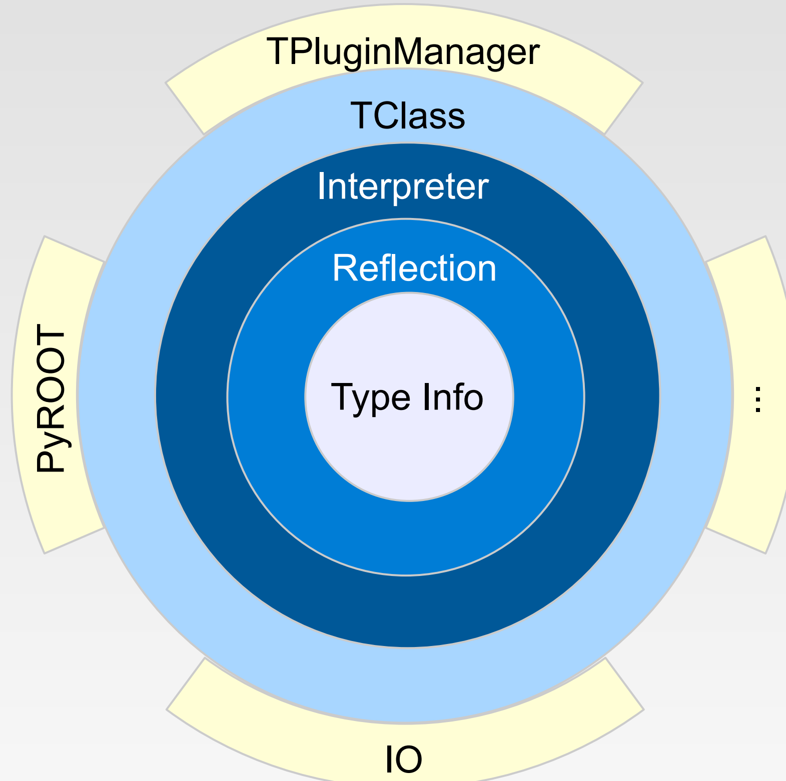
TClass is ROOT's entry point for the reflection world:

- Allows selective rebuilding of the objects

Reflection Information Flow

Due to the limited C++ language support for reflection and type introspection (both compile-time and runtime):

- Opening up interpreter's internals
- String-based lookups (eg. findType("std::vector<int>"))



Role of C++ Interpreter

- **Translate-time type information**
  Opens up the internals and yields access to type information, memory layout, etc. as a source of introspection data
- **Reflection**
  Interface to interpreter's internals.
- **Dynamism**
  Enhances C++ runtime, increasing its dynamic power. For example: dynamic scoping or dynamic languages binding (python)
- **Interactivity**
  Proven to be very helpful in learning both C++ and the framework.

Language Bindings in ROOT

- Descriptive reflection layer

  Dynamic binding through introspection, not predefined (externally described) like SWIG

- Allows implementing bindings for more dynamic languages such as Python and Ruby

  Interface to interpreter's internals.

```
// Example: accessing the Python interpreter from ROOT
// either load PyROOT explicitly or rely on auto-loading
root[] gSystem->Load( "libPyROOT" );
root[] TPython::Exec("print1+1");
2
```

```
root[] TRuby::Exec("require '/usr/local/lib/root/libRuby'");
root[] TRuby::Exec("c1 = TBrowser.new");
root[] TRuby::Eval("c1.GetName");
```

PyROOT

```
root [0] gSystem->Load("libPyROOT");
root [1] TPython::Exec( "print 1 + 1" );
2
root [2] TPython::Prompt();
>>> i = 12;
>>> ^D
root [3] TPython::Prompt();
>>> print i
12
>>> ^D
root [4]
```

```
root [2] gSystem->Load("libPyROOT");
root [3] std::vector<int> v; v.push_back(22);
root [4] TPython::Prompt();
>>> from ROOT import *
>>> for i in v: print i
...
22
>>> ^D
root [5]
```

ROOT's reflection layer could be used to bridge dynamically both worlds!

CINT

Current ROOT's production interpreter for over 17 years:

- Written by Masaharu Goto in 1991
- Laid the foundation of our understanding of dynamic C++
- ~400K SLOC
- Not fully C++ compliant
- Very hard to implement new C++* features
- Not very good diagnostics
- Boundary between compiled and interpreted code
- Hard to make it thread safe

Motivation For Cling

- Inherited from clang full C++ support
  - STL + templates
  - Path to C++11
- Correctness
- Better type information and representations
- Always compile in memory
- Much less code to maintain (15K SLOC)

Cling Uses Clang & LLVM

LLVM and Clang

“The LLVM Project is a collection of modular and reusable compiler and toolchain technologies...”

Cling's Dual Personality

- An interpreter – looks like an interpreter and behaves like an interpreter
  Cling follows the read-evaluate-print-loop (repl) concept.
- More than interpreter – built on top of compiler libraries (Clang and LLVM)
  Contains interpreter parts and compiler parts. More of an interactive compiler or an interactive compiler interface for clang.

No need to compile Cling/ROOT with Clang/LLVM or having clang installed on the OS

Cling's Key Strengths

- Full C++ support incl. C++11
- Stable and informative intermediate representations of the source
- Being developed with the vision to be used in multithreaded environment (an interpreter object per thread)

Full C++ Support

- Templates and STL are not an issue

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ #include <vector>
[cling]$ #include <map>
[cling]$ #include <string>
[cling]$ #include <set>
[cling]$ using namespace std;
[cling]$ vector<map<string,set<int> > > a
(vector<map<string, set<int> > >) @0x10b190020
[cling]$ 
```

Full C++ Support

- Natural path to the new standards C++11/C++*

$ cling –std=c++11

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ #include <vector>
[cling]$ #include <cstdio>
[cling]$ std::vector<double> a{1., 2., 3., 4.}
(std::vector<double>) @0x7f5f59b19018
[cling]$ for (auto i:a) {
[cling]$ ?   printf("%g\n", i);
[cling]$ ?   }
1
2
3
4
[cling]$
```

Full C++ Support

- Natural path to the new standards C++11/C++*

$ cling –std=c++11

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ .rawInput
Using raw input
[cling]! extern "C" int printf(const char* fmt,...);
[cling]! template <typename T> void F(T arg) {
[cling]! ?   auto func = [arg]() mutable -> T { printf("I am a lambda!\n"); return arg + T();};
[cling]! ?   func();
[cling]! ?   }
[cling]! .rawInput
Not using raw input
[cling]$ F(11.)
I am a lambda!
[cling]$
```

Full C++ Support

- Boost is not a dream

```
[cling]$ #include <iostream>
[cling]$ #include "boost/random.hpp"
[cling]$ #include "boost/generator_iterator.hpp"
[cling]$ using namespace std;
[cling]$ .rawInput
Using raw input
[cling]! void f() {
[cling]! ?   typedef boost::mt19937 RNGType;
[cling]! ?         RNGType rng;
[cling]! ?         boost::uniform_int<> one_to_six( 1, 6 );
[cling]! ?         boost::variate_generator< RNGType, boost::uniform_int<> >
[cling]! ?                     dice(rng, one_to_six);
[cling]! ?         for ( int i = 0; i < 6; i++ ) {
[cling]! ?               cout << dice() << ((i != 5) ? "," : "\n");
[cling]! ?           }
[cling]! ?   }
[cling]! .rawInput
Not using raw input
[cling]$ f()
5,1,6,6,1,6
[cling]$
```

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ #include <boost/thread.hpp>
LLVM ERROR: JIT does not support inline asm!
```

Inline ASM an issue with the current JIT


EVERYBODY
LIES

Diagnostics

- Column numbers and caret diagnostics

  CaretDiagnostics.C:4:13: warning: ***'.*' specified field precision is missing a matching 'int' argument***

  ```
  printf("%.*d");
            ~~^~
  ```

- Range highlighting

  RangeHighlight.C:14:39: error: *invalid operands to binary expression ('int' and 'A')*

  ```
  return y + func(y ? ((SomeA.X + 40) + SomeA) / 42 + SomeA.X : SomeA.X);
                       ~~~~~~~~~~~~~~ ^ ~~~~~~~
  ```

Diagnostics

- Pointer vs References

```
input_line_410:2:6: error: member reference type 'TNamed' is not a pointer
nRef->GetName();
~~~~^

input_line_413:2:7: error: member reference type 'TNamed *' is a pointer; maybe you meant to
use '->'?
nPtr1.GetName();
~~~~^
    ->
```

- Fix-it hints

```
FixItHints.C:7:27: warning: use of GNU old-style field designator extension
struct point origin = { x: 0.0, y: 0.0 };
                        ^~
                         .x =
FixItHints.C:12:3: error: use of undeclared identifier 'floot'; did you mean 'float'?
 floot p;
 ^~~~
 float
```

Diagnostics

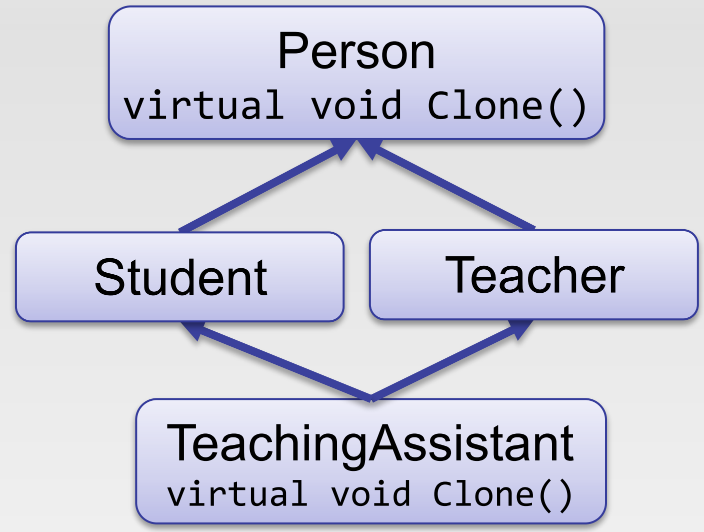
- Ambiguities

Ambiguities.C:20:30: error: *return type of virtual function 'Clone' is not covariant with the return type of the function it overrides (ambiguous conversion from derived class 'TeachingAssistant' to base class 'Person':*
class TeachingAssistant -> class Student -> class Person
class TeachingAssistant -> class Teacher -> class Person)

```
virtual TeachingAssistant* Clone() const;
                           ^
```

Ambiguities.C:7:19: note: *overridden virtual function is here*

```
virtual Person* Clone() const;
                ^
```



- Templates

input_line_401:2:2:error: *use of class template LorentzVector requires template arguments*

```
 LorentzVector v;
 ^
```

Math/GenVector/LorentzVectorfwd.h:28:39: note: *template is declared here*

```
 template<class CoordSystem> class LorentzVector;
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~         ^
```

Diagnostics

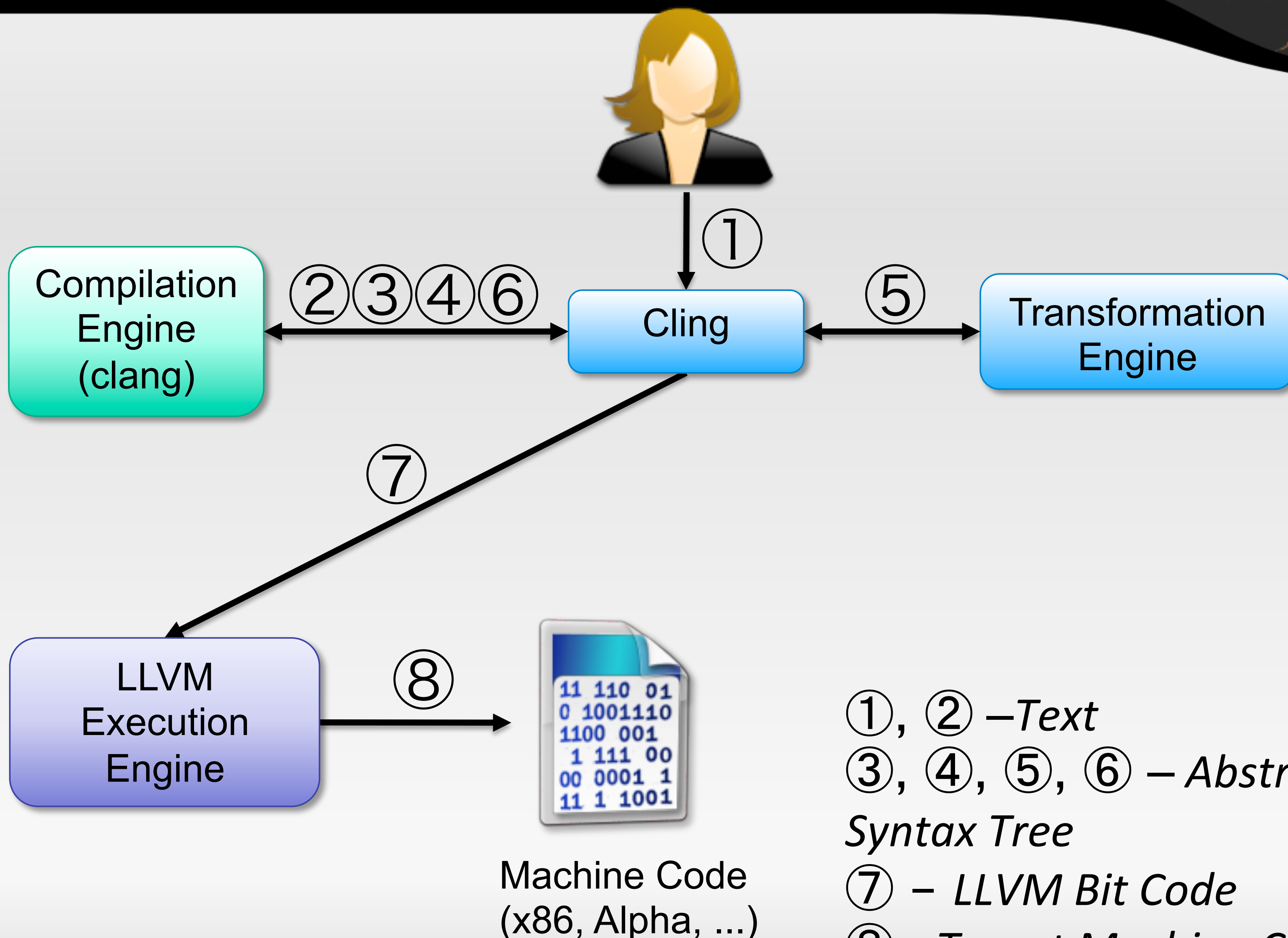
- Macro expansions

```
MacroExpansionInformation.C:14:7: error: invalid operands to binary expression ('int' and 'A')
 X = MAX(X, *SomeA);
     ^~~~~~~~~~~~~~
MacroExpansionInformation.C:5:24: note: expanded from macro 'MAX'
#define MAX(A, B) ((A) > (B) ? (A) : (B))
                   ~~~ ^ ~~~
```

- Template instantiations

```
input_line_395:2:18: error: no matching constructor for initialization of 'PtEtaPhiEVector' (aka
'LorentzVector<PtEtaPhiE4D<double> >')
PtEtaPhiEVector v2( "v1.Rho()", v1.Eta(), v1.Phi(), v1.E() );
                ^   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Math/GenVector/LorentzVector.h:77:8: note: candidate constructor not viable: no known
conversion from 'const char [9]' to 'const Scalar' (aka 'const double') for 1st argument
LorentzVector(const Scalar & a,
^
Math/GenVector/LorentzVector.h:88:17: note: candidate constructor template not viable:
requires single argument 'v', but 4 arguments were provided
 explicit LorentzVector(const LorentzVector<Coords> & v ) :
          ^
```

Data Flow



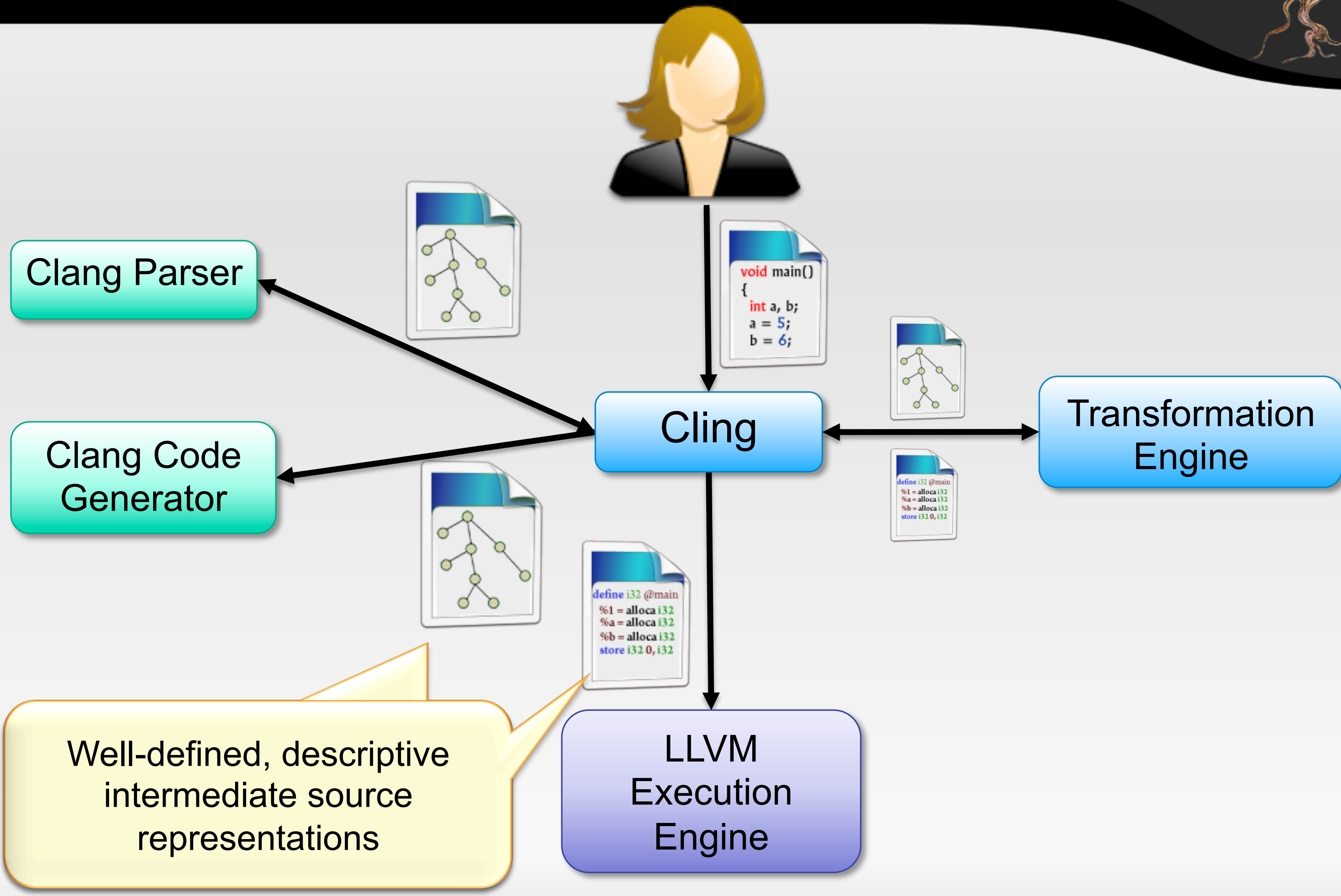
①, ② –*Text*

③, ④, ⑤, ⑥ – *Abstract Syntax Tree*

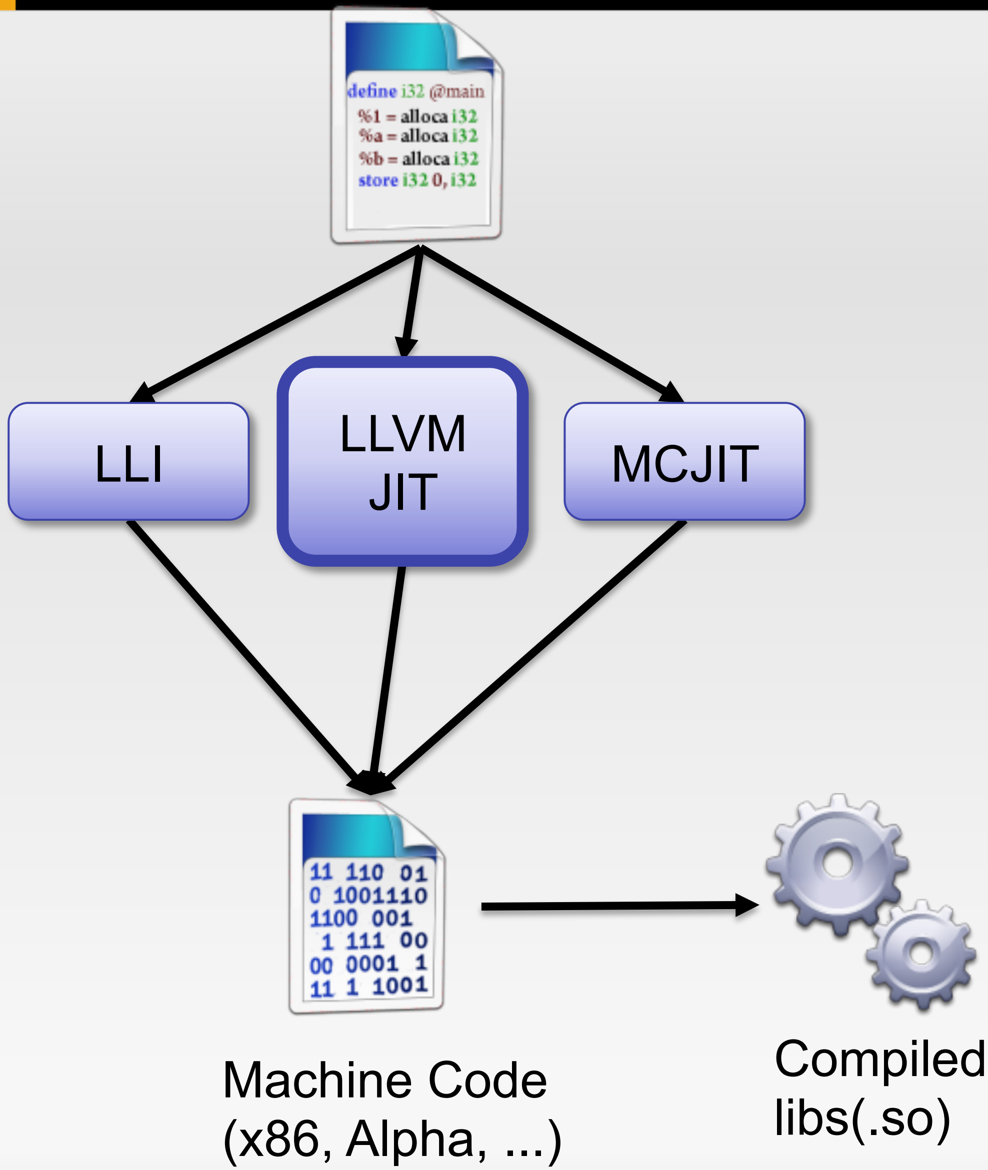
⑦ – *LLVM Bit Code*

⑧ –*Target Machine Code*

Compilation Engine



Execution Engine



- *LLVM EE-s have complete target info*

  Thus calling into compiled libraries is not an issue.

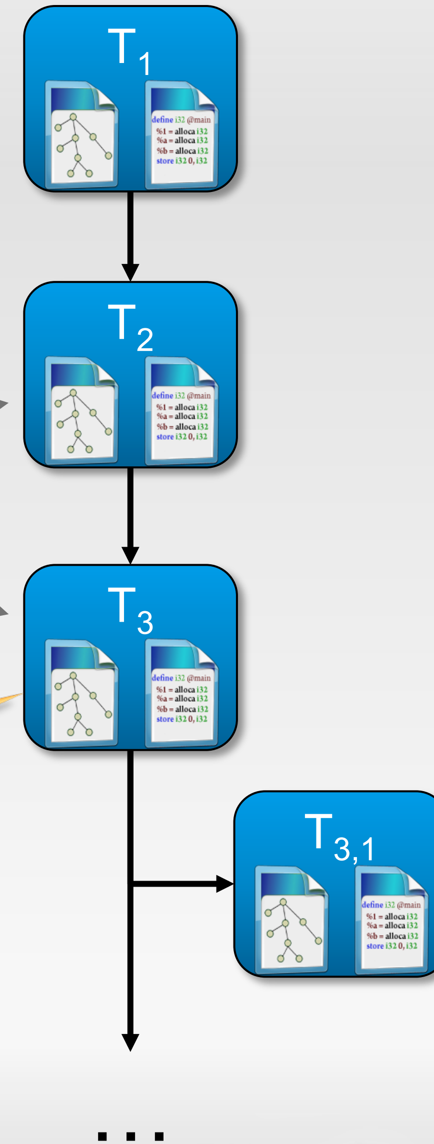
- *No boundary interpreted/ compiled world*

  Possible to derive from compiled classes, proper calculation of offsets and so on.

```
vvassilev@vvBook:~$ cling --nologo -lz
[cling]$ #include "zlib.h"
[cling]$ zlibVersion()
(const char *) "1.2.5"
[cling]$
```

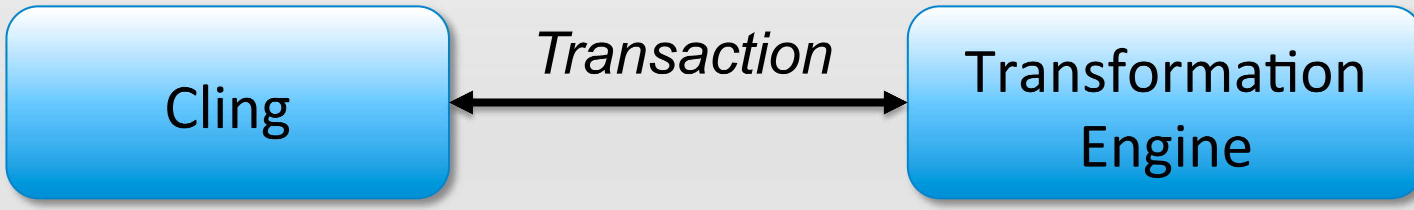
Incremental Input In Transactions

```
[cling]$ #include <myheader.h>
[cling]$ int i = 12; printf("%d\n",i);
[cling]$ lookup.findScope("std::vector<int>")
         (const clang::Decl *) 0x5d60260
```

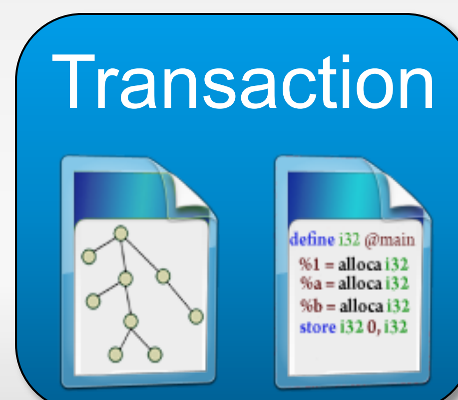


Available through InterpreterCallbacks.
(Thus ROOT implements its reflection client)

Transformation Engine



- **Transaction**
  Cling represents the incremental input as a set of AST node.
- **Transaction Transformers**
  Cling enables each transaction to be further customized by other clients by implementing a transaction transformer.
- **Interpreter Callbacks**
  Implements callbacks for the "interesting" events.



Challenges

- Incompatible concepts like compilation and interpretation
  Many tasks that are trivial for an interpreter become a nightmare for a compiler.
- Make C++ usable at the prompt
  Incorporate the experience we have with CINT. First step: adopt the successful usability extensions from CINT.

Extending C++ Language

```
[root]$ sin(12);
```

```
void wrapper() {
  sin(12);
}
```

We want to be able to run statements

```
[root]$ int i = 12;
[root]$ sin(i);
```

```
void wrapper1() {
  int i = 12;
}
```

```
void wrapper2() {
  sin(i);
}
```

?

Extending C++ Language

Wrap the input

Look for declarations

Extract the declarations one level up, as global declarations

```
int i = 12;
```

```
void wrapper1() {
  int i = 12;
  printf("%d\n",i);
}
```

```
[cling]$ int i = 12; printf("%d\n",i);
[cling]$ printf("%f\n",sin(i));
```

```
void wrapper2() {
  printf("%f\n", sin(i));
}
```

Streaming Execution Results

```
[cling]$    int i = 12
            (int) 12
[cling]$    sin(i)
            (double) -5.365729e-01
```

No semicolon (;)

```
[cling]$    std::string s = “Hello”
            (std::string) @0x7fff65ae783c
            c_str: “Hello”
[cling]$    enum e { e1 = 12, e2 = 13, e3 = 13}; e1
            (enum e) (e::e1) : (int) 12
```

Precise type information

```
[cling]$    HelloWorld
            (void (void)) Function @0x10888 030
              at /tmp/HelloWorld.h:2:
            void HelloWorld() {
              printf("HelloWorld!\n");
            }
```

Precise location information

Error Recovery

Filled input-by-input (Transaction-by-Transaction)

Incorrect inputs must be discarded as a whole

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ int i; ERROR_HERE; int j;
input_line_4:2:9: error: use of undeclared identifier 'ERROR_HERE'
 int i; ERROR_HERE; int j;
        ^
[cling]$ i
input_line_5:2:2: error: use of undeclared identifier 'i'
 i
 ^
[cling]$ 
```

Implicit auto keyword

We meant int i = 5;
or in C++11
auto i = 5;

TNamed * f = …
or in C++11
auto f = …

```
i = 5; f = new TNamed(“a”, ”b”)
```

Cling will mark the AST Node as an implicit auto candidate and later on a custom AST pass will do the work.

Late Binding

✓ Defined in the root file

```
if (cond) {
  TFile F;
  if (is_day_of_month_even())
    F = TFile::Open("even.root");
  else
    F = TFile::Open("odd.root");
  hist->Draw();
}
hist->Draw();
```

✗ The root file is gone. Issue an error.

✚ Opens a dynamic scope. It tells the compiler that cling will take over the resolution of possible unknown identifiers

Late Binding

Automatically transformed into valid C++ code on AST level

```
if (cond) {
  TFile* F = 0;
  if (is_day_of_month_even())
    F = TFile::Open("even.root");
  else
    F = TFile::Open("odd.root");
  gCling->EvaluateT<void>("hist->Draw()", ...);
}
hist->Draw();
```

- Tell the compiler the identifier will be resolved at runtime
- Wrap it into valid C++ code
- Partially recompile at runtime

Late Binding

```
if (cond) {
  int x = 1; double y = 2.;
  TFile* F = 0;
  if (is_day_of_month_even())
    F = TFile::Open("even.root");
  else
    F = TFile::Open("odd.root");
  if (hist->CanDraw(x, y))
    hist->Draw();

}
hist->Draw();
```

Type information (cast back mask)

Placeholders, which are replaced with the real addresses at runtime

```
if (gCling->EvaluateT<bool>("hist->CanDraw(*(int*)@, *(double*)@",
                                  (void*[2]){&x, &y}))...
```

Instantiated with the expected return type

Relevant context stored as array of void* addresses

Code Unloading

```
[cling]$ .L Calculator.h
[cling]$ Calculator calc;
[cling]$ calc.Add(3, 1)
         (int) 2  //WTF!?*
[cling]$ .L Calculator.h
[cling]$ Calculator calc;
[cling]$ calc.Add(3, 1)
         (int) 4  //☺
[cling]$
```

```
// Calculator.h
struct Calculator {
  int Add(int a, int b) {
    return a - b;
  }
...
};
```

```
// Calculator.h
struct Calculator {
  int Add(int a, int b) {
    return a + b;
  }
...
};
```

* What's That Function

More than C++

$ cling –x objective-c

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ .L /System/Library/Frameworks/Foundation.framework/Foundation
[cling]$ .rawInput
Using raw input
[cling]! #import <Foundation/Foundation.h>
[cling]! void f() {
[cling]! ?   NSLog (@"Hello, World!");
[cling]! ?   }
[cling]! .L /System/Library/Frameworks/Foundation.framework/Foundation
[cling]! .rawInput
Not using raw input
[cling]$ f();
2013-05-05 12:35:22.929 cling[10174:707] Hello, World!
[cling]$
```

No “real” linker

Optimizations

- Less parsing
  Use optimization structures such as PCMs.
- Less JIT-ting
  No trampolines for function argument set up.
- Smart optimizations of user code (eg. Devirtualization)
- Tracing JIT?

Future Plans

- Migrate to MCJIT
  - Object file emitted to memory
  - Runtime dynamic linker
  - Less trampolines
- Windows 64 Support
- Null pointer derefs
- Tools
  - Automatic Differentiation

References

http://cern.ch/cling

http://www.youtube.com/watch?v=eoIuqLNvzFs (Cling Interactive OpenGL Demo)

http://www.youtube.com/watch?v=wZZdDhf2wDw (Qling/cling: recursive C++ interpreting)

https://www.youtube.com/watch?v=BrjV1ZgYbbA (Qt + Cling, the LLVM based C++ interpreter)

References

http://cern.ch/cling

[0] http://home.web.cern.ch/about/

[1] “Software release build process and components in ATLAS offline” http://cds.cern.ch/record/1321876/files/ATL-SOFT-PROC-2011-012.pdf

[2] “The ROOT Framework” http://root.cern.ch/drupal

[3] Rolf Landua - http://indico.cern.ch/getFile.py/access?resId=2&materialId=slides&confId=134761

[4] http://root.cern.ch/root/html/TTree.html

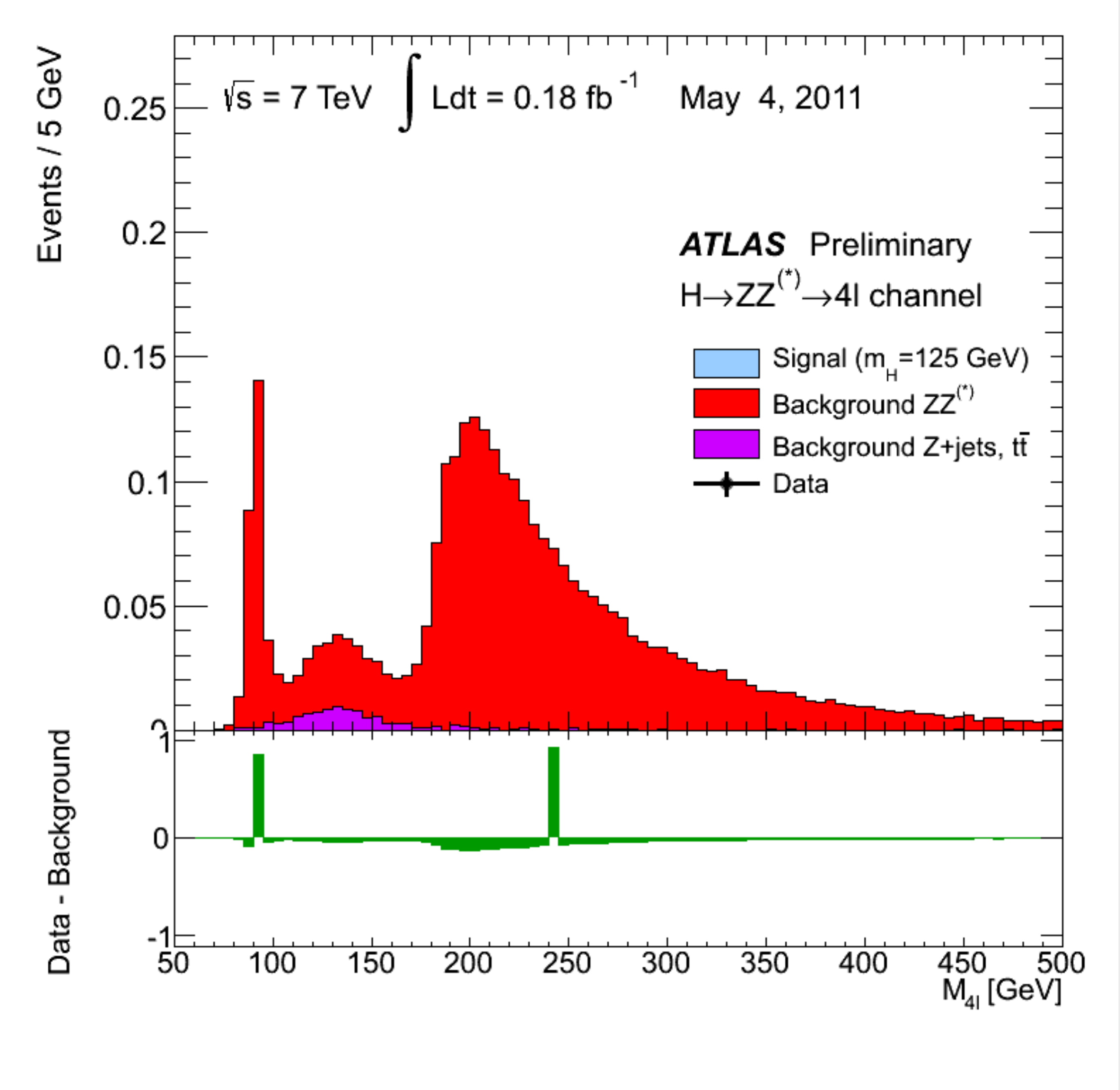
[5] http://physicsworld.com/cws/article/news/2013/apr/30/alpha-weighs-in-on-antimatter

[6] “LHC – Marvel of Technology”, EPFL Press

[7] “Implementing Dynamic Scopes In Cling” – Euro LLVM Dev, http://www.llvm.org/devmtg/2011-09-16/

[8] http://www.atlas.ch/photos/plots.html

[9] http://root.cern.ch/download/doc/19PythonRuby.pdf



$\sqrt{s}$ = 7 TeV $\int$ Ldt = 0.18 fb^{-1} May 4, 2011
ATLAS Preliminary
H→ZZ$^{(*)}$→4l channel
Signal (m_H=125 GeV)
Background ZZ$^{(*)}$
Background Z+jets, $t\bar{t}$
Data
Events / 5 GeV
Data - Background
M_{4l} [GeV]

Thank you!