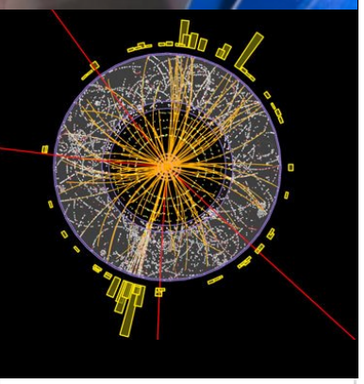
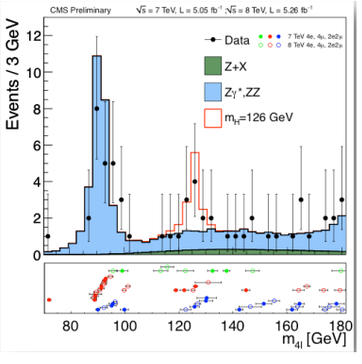
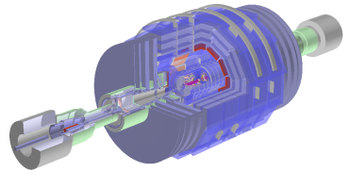






ROOT

Data Analysis Framework

6

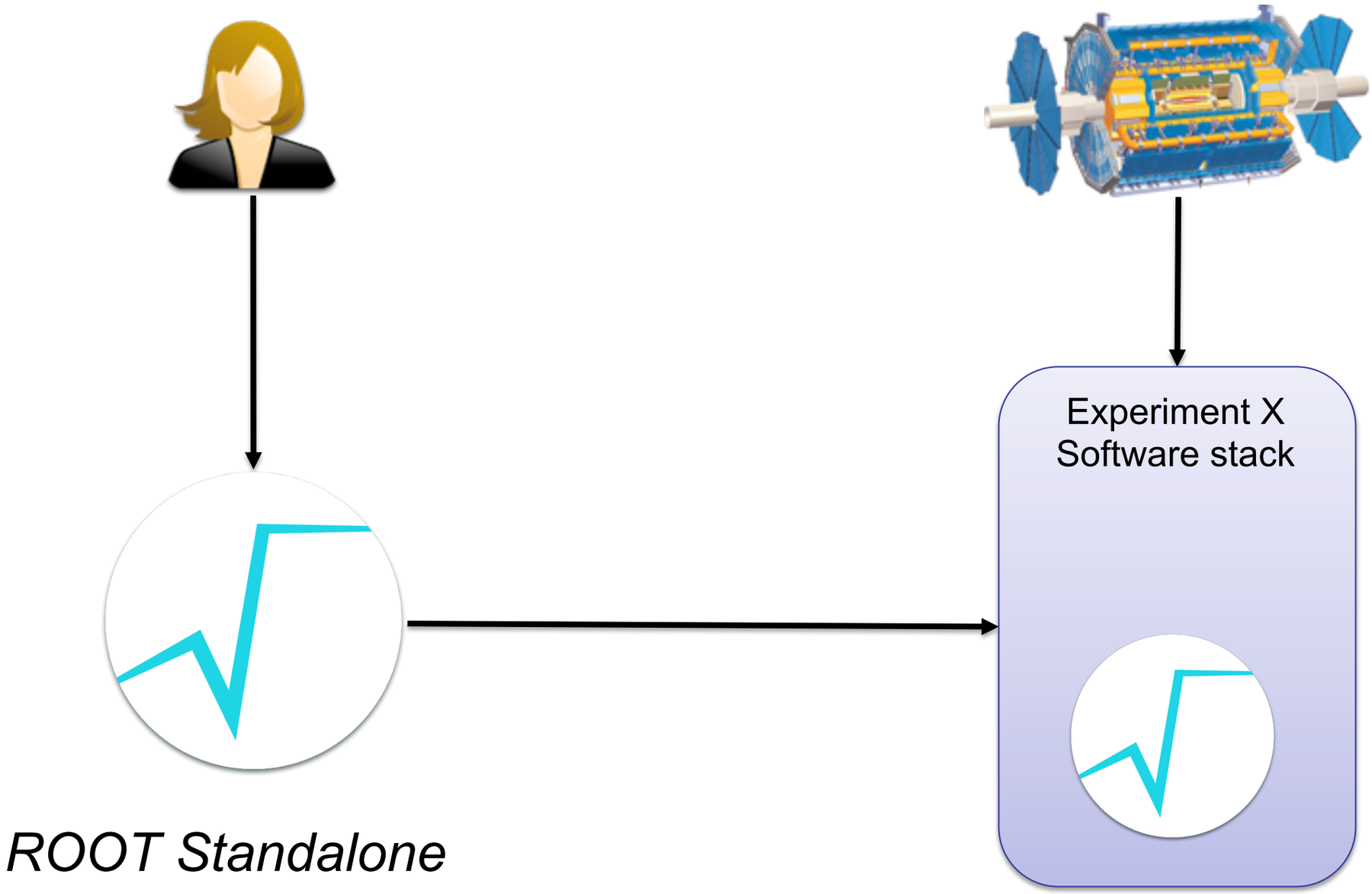
ROOT 6

Vassil Vassilev

CERN, PH-SFT

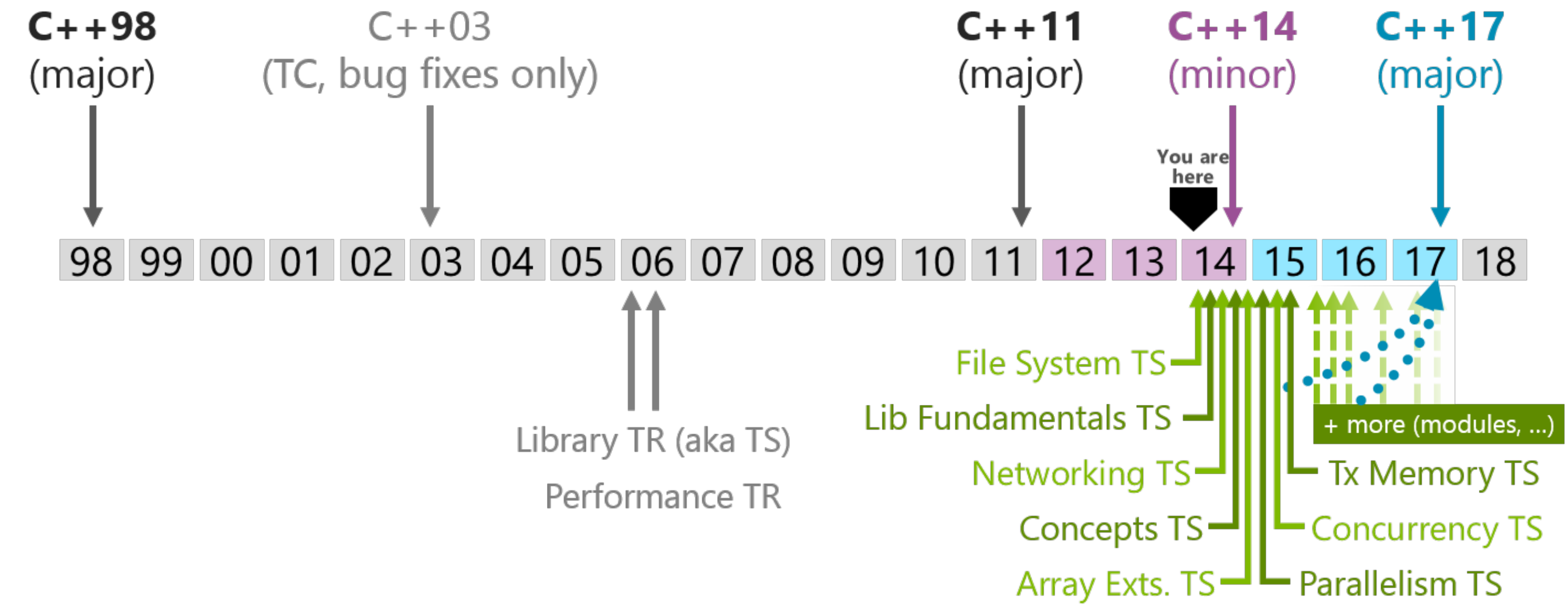
Firstname Lastname
CERN PH-SFT
CH-1211 Geneva 23
root.cern.ch

ROOT Usage Scenarios



ROOT Standalone

C++ & ROOT



TS: Technical Specification
TR: Technical Report

ROOT had to follow, offering these features to the HEP community

Why Do You Want Better C++

```
std::vector<MyClass*>::const_iterator I = vec.begin();
```

```
auto I = vec.begin();
```

```
for(auto &E : vec) { // do things }
```

```
for(std::vector<MyClass*>::const_iterator I = vec.begin(); I != E; ++I) {
  // do things
}
```

The compiler will issue error that ‘fRenamed’ doesn’t override anything

```
struct B { virtual void f(short) {} };
struct D : public B { virtual void fRenamed(int) override {} };
```

The compiler will issue error that ‘f’ cannot be overridden.

```
struct B { virtual void f(int) {} };
struct D : public B { virtual void f(int) override final {} };
struct F : public D { virtual void f(int) override {} };
```

Why Do You Want Better C++

- Smart pointers

  Helps with defining ownerships. Improves readability/understandability of interfaces.

- Move semantics

  Reduces copying objects. Improves performance.

- Concurrency

  Threads. Mutual exclusions. Condition variables. Futures.

- Many, many, many more.

ROOT's Plan

ROOT needed to catch up and offer the new C++ features keeping:

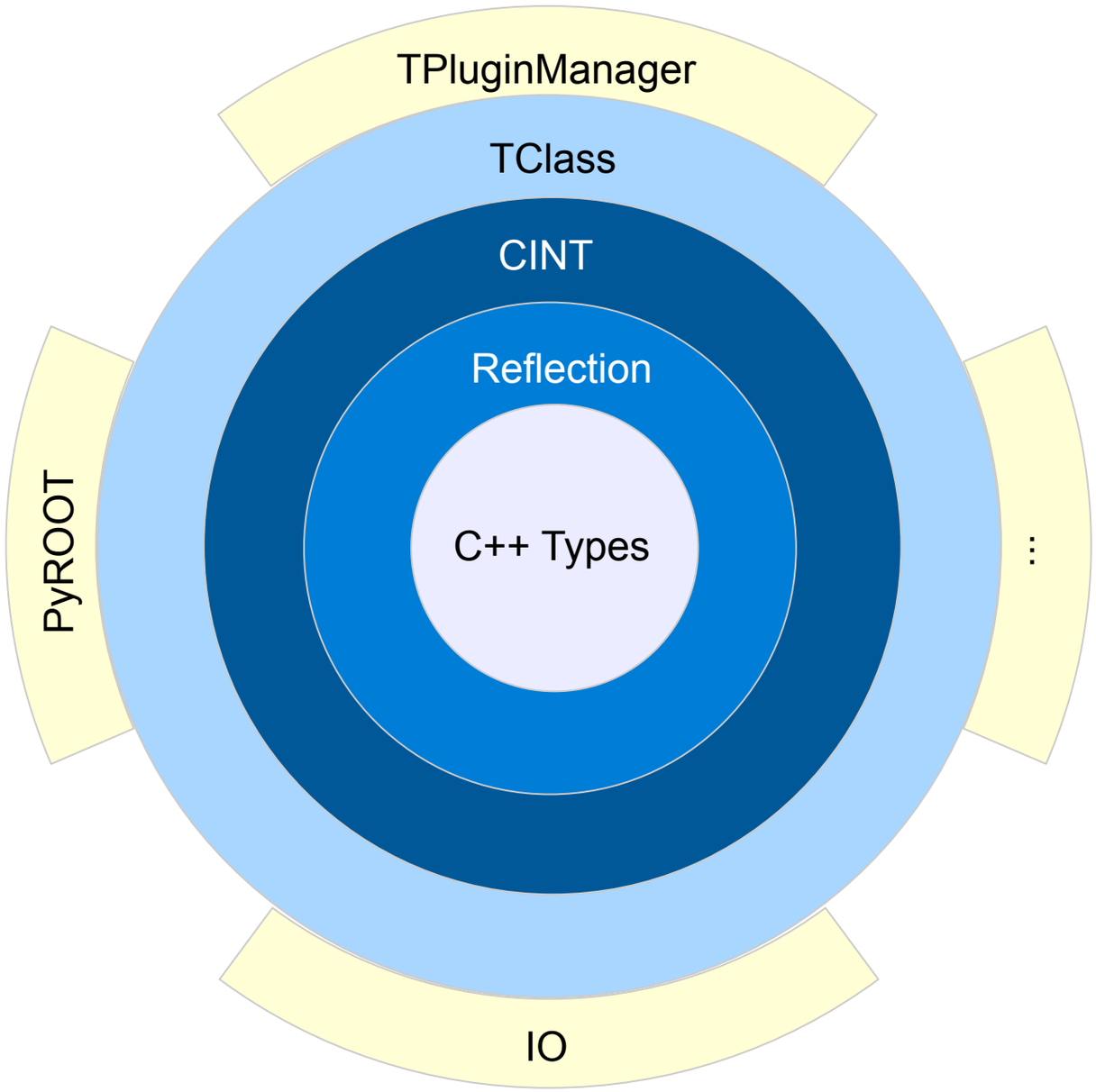
- Backward and forward compatibility
- Performance 'compatibility'
- Designing the new system having 'concurrency' in mind

How? Simple Things Didn't Work…

```
int f(int& arg[]) { return arg[1]; }
void CINTfail() {
   int var = 13;
   {
      int var  = 14;
   }
   printf("var should be 13 but is %d\n", var);
   int arr[3] = { 12, 13, 14 };
   printf("f(arr) should be an error (expected 13) but is %d\n", f(arr));
}
root [2] .x CINTfail.cxx
var should be 13 but is 14
f(arr) should be an error (expected 13) but is 0
```

ROOT ‘Architecture’

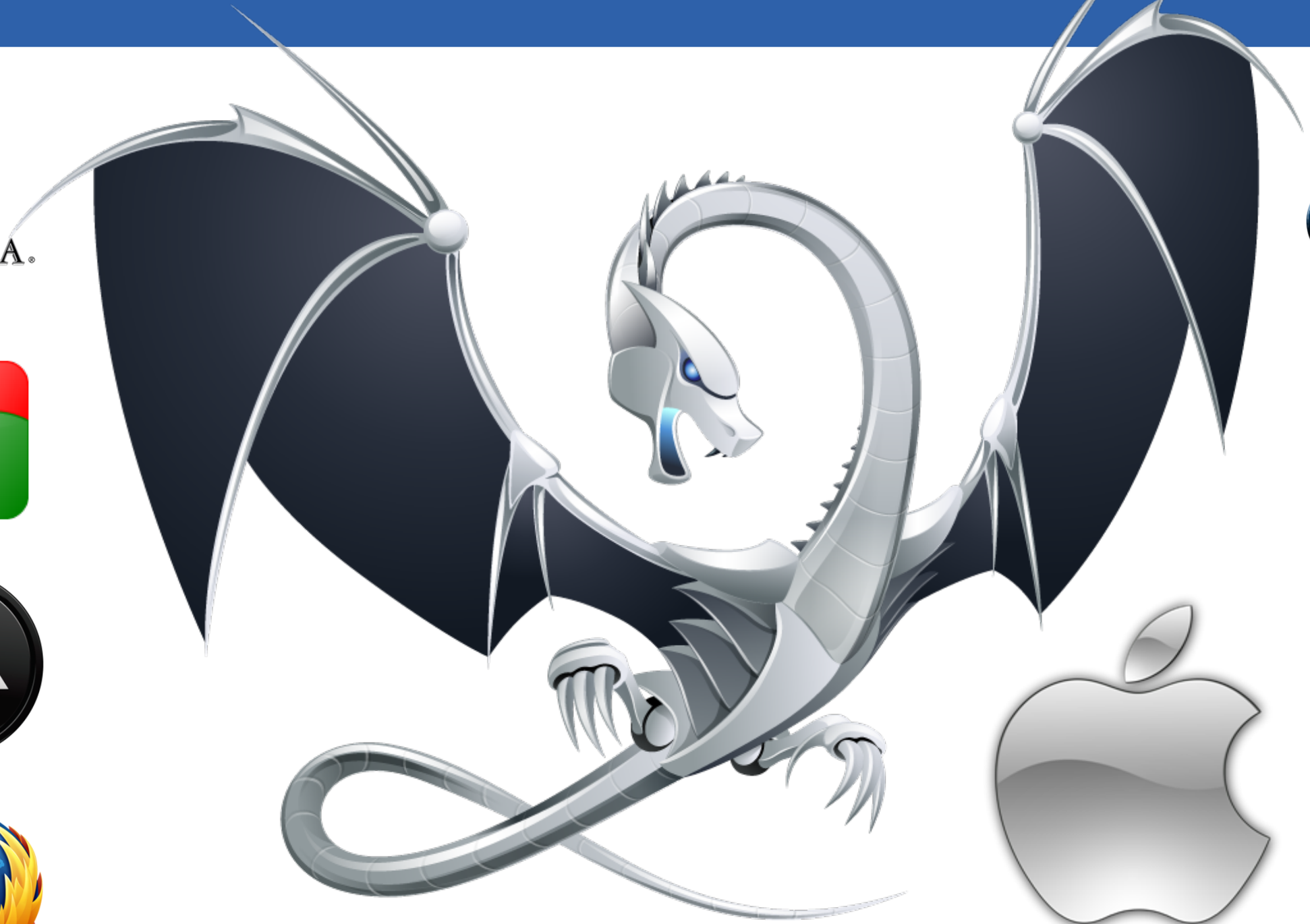
- Load/Store C++ objects
- Runtime Dynamism
  - `TFile::Open(“http://...”)`
  - `gDirectory->Get(“hist”)`
  - `python runReco.py`
- Fast Prototyping



Limited C++ language support for reflection and type introspection.

”Optimism is the essential ingredient for innovation. How else can the individual welcome change over security, adventure over staying in safe places” *R. Noyce, Intel Cofounder*

In The Meanwhile: LLVM



LLVM and Clang

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

ROOT's New CINT = Cling

- A new development at CERN
  Implemented to fit ROOT's specific needs.
- 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 ROOT with Clang or having clang installed on the OS

Full C++ Support

- Templates and STL are not an issue

```
 ------------------------------------------------------------------
| Welcome to ROOT 6.03/01                        http://root.cern.ch |
|                                  (c) 1995-2014, The ROOT Team     |
| Built for macosx64                                                |
| From heads/master@v6-00-01-1807-g5744632, Nov 17 2014, 12:08:46   |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'        |
 ------------------------------------------------------------------

root [0] #include <vector>
root [1] #include <map>
root [2] #include <string>
root [3] #include <set>
root [4] using namespace std;
root [5] vector<map<string, set<int>>> a
(vector<map<string, set<int> > > &) @0x11a995058
root [6]
```

Full C++ Support

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

```
 ------------------------------------------------------------------
| Welcome to ROOT 6.03/01                        http://root.cern.ch |
|                                  (c) 1995-2014, The ROOT Team     |
| Built for macosx64                                               |
| From heads/master@v6-00-01-1807-g5744632, Nov 17 2014, 12:08:46  |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
 ------------------------------------------------------------------

root [0] #include <vector>
root [1] std::vector<int> v = {1,2,3}
(std::vector<int> &) @0x119f7f058
root [2] for(auto el : v) {
root (cont'ed, cancel with .@) [3]printf("%d\n", el);
root (cont'ed, cancel with .@) [4]}
1
2
3
root [5]
```

Full C++ Support

- Natural path to the new standards C++11/C++*, just works.

```
------------------------------------------------------------------
| Welcome to ROOT 6.03/01                        http://root.cern.ch |
|                                  (c) 1995-2014, The ROOT Team |
| Built for macosx64                                            |
| From heads/master@v6-00-01-1807-g5744632, Nov 17 2014, 12:08:46 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'    |
------------------------------------------------------------------

root [0]
root [1] template<typename... T> int f(T... args) {
root (cont'ed, cancel with .@) [2]printf("Argument size = %lu\n", sizeof...(T));
root (cont'ed, cancel with .@) [3]auto lambda = [&]() mutable ->int { printf("I am a lambda.\n"); return 0;};
root (cont'ed, cancel with .@) [4]return lambda();
root (cont'ed, cancel with .@) [5]}
root [6] int result = f(1,1,2,3,5,8,13,21);
Argument size = 8
I am a lambda.
root [7]
```

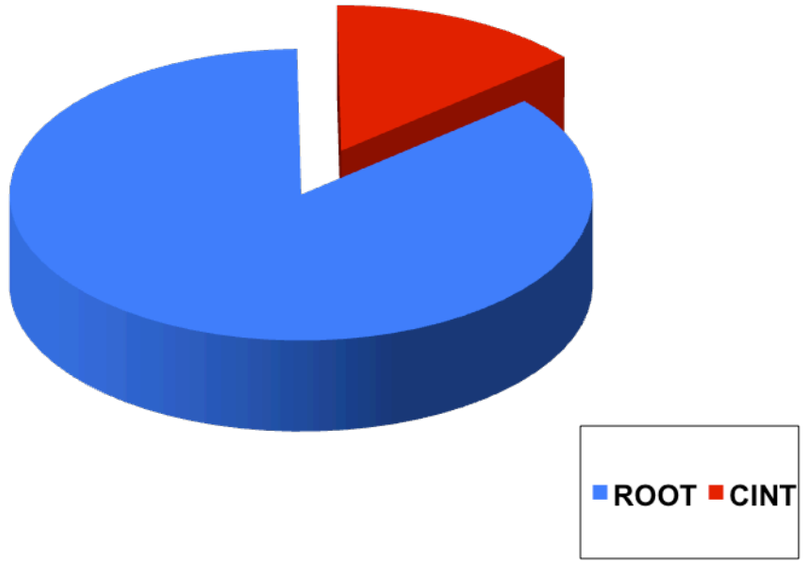
ROOT6 Is Better Than ROOT5

- Backward- and forward-compatible
- Full C++ support incl. C++11, soon C++14.
- Core is being developed/improved with the vision to be used in multithreaded environment

ROOT6 Is More Maintainable

Other ROOT – 1400K SLOC*

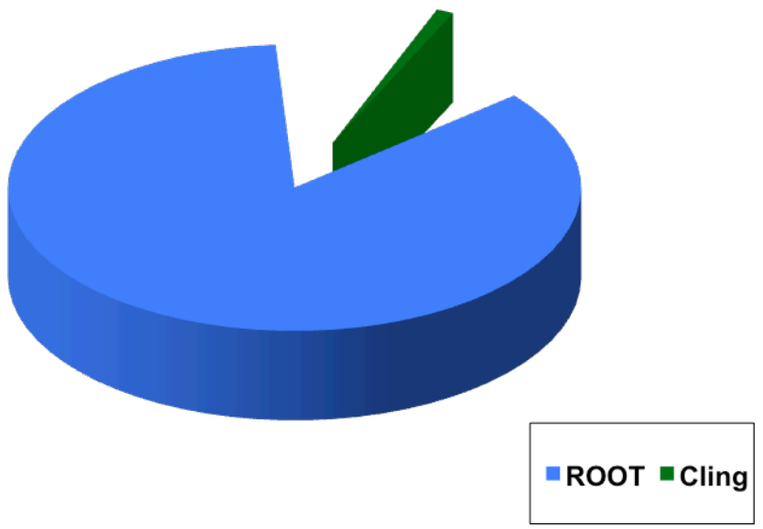
CINT+Reflex – 230K SLOC*



Cling – 16K SLOC*

externals:

LLVM + Clang – 800K SLOC*



** No testsuites included.*
Credits: generated using David A. Wheeler's 'SLOCCount'


EVERYBODY
LIES

Diagnostics Matter!

```
int f(int& arg[]) { return arg[1]; }
int CINTfail() {
  int arr[3] = { 12, 13, 14 };
  return f(arr);
} // CINT Returns 0 instead of error.
```

Diagnostics in ROOT6

- 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 in ROOT6

- 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 in ROOT6

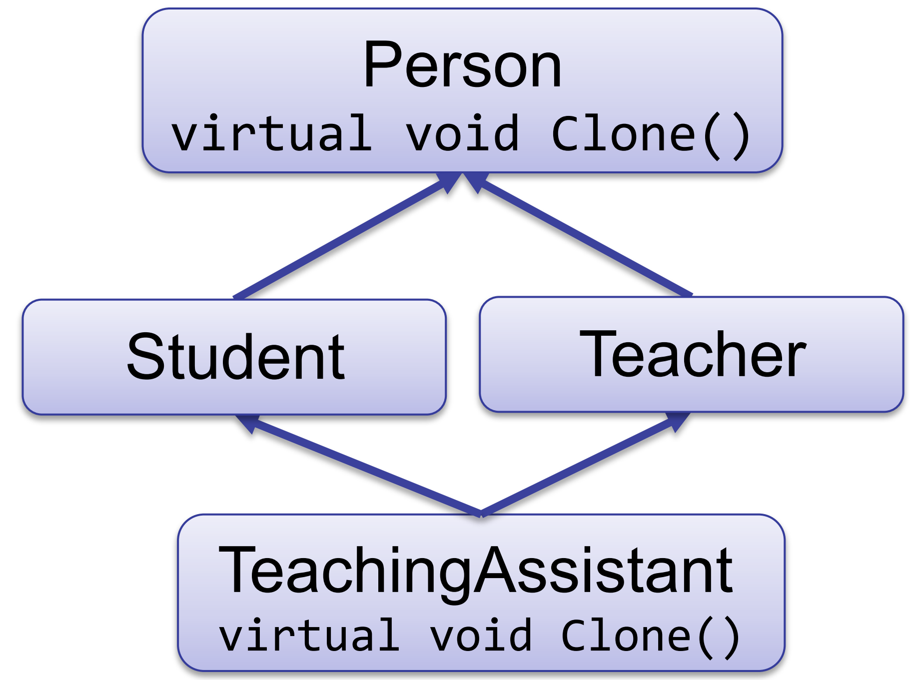
- Ambiguities

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 in ROOT6

- 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() );
                  ^     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

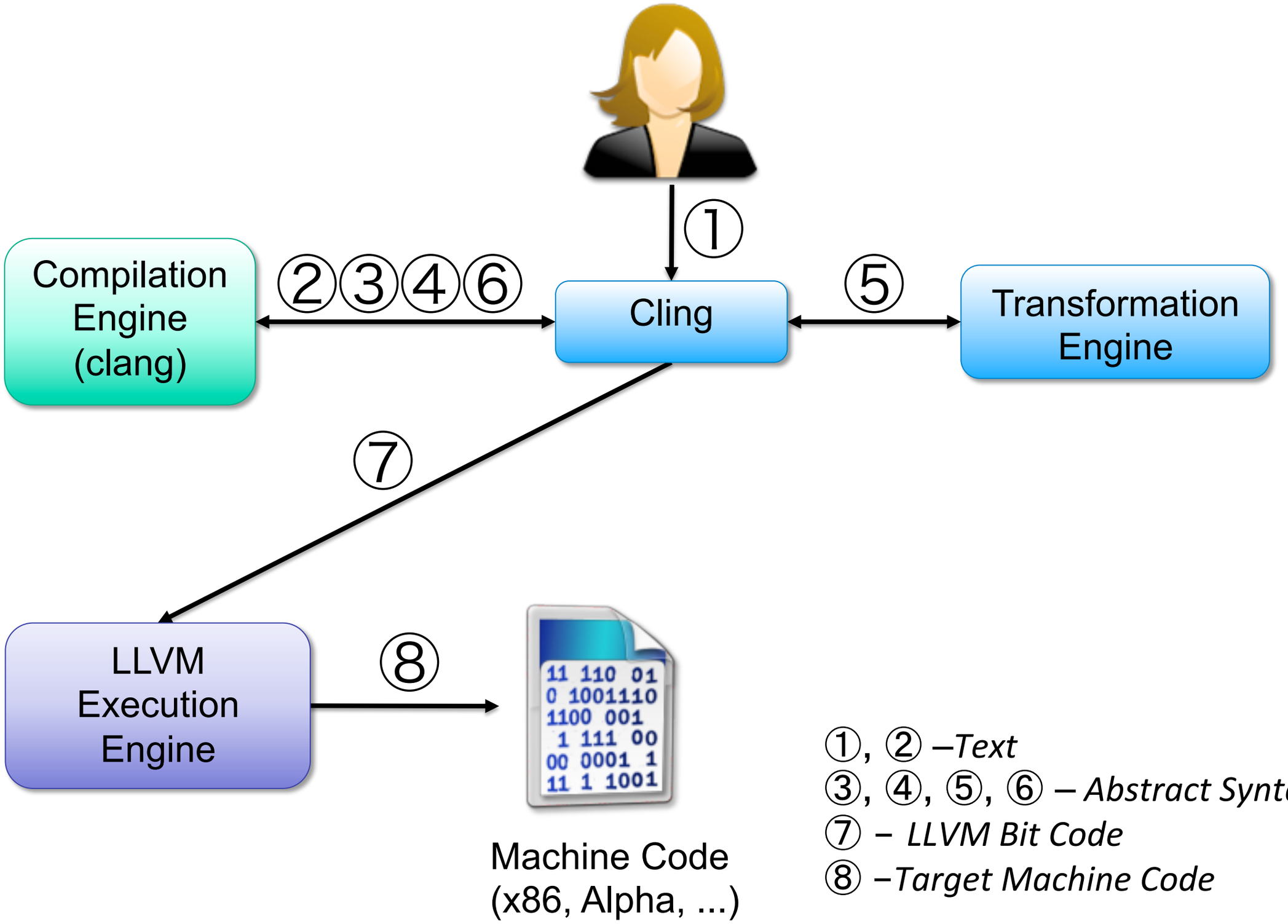
Math/GenVector/LorentzVector.h:77:8: note: ***candidate constructor not viable: no known conversion from 'const char [9]' to 'const Scalar' (aka 'const double') for 1st argument***

```
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***

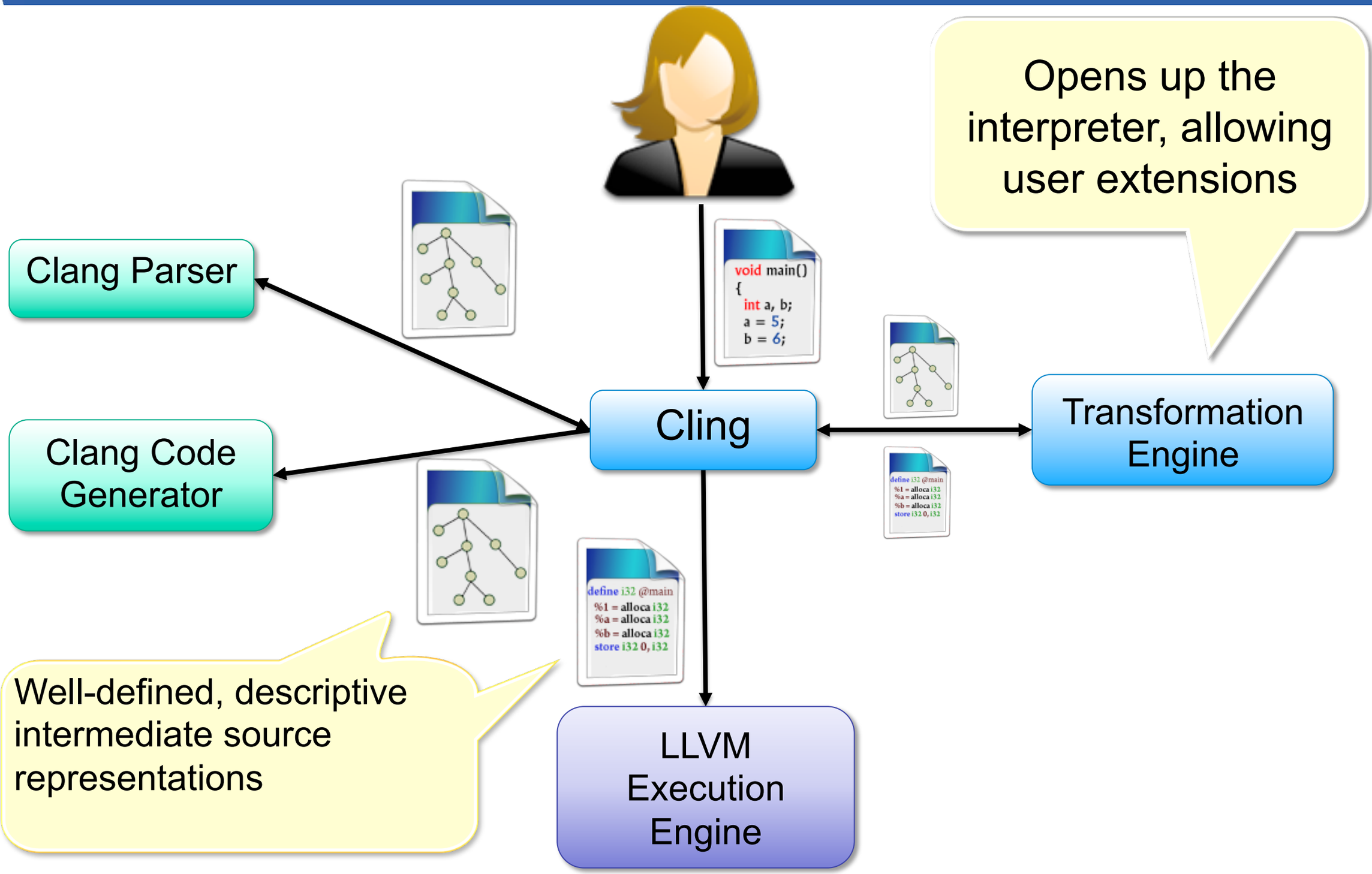
```
 explicit LorentzVector(const LorentzVector<Coords> & v ) :
                 ^
```

‘Data Flow’ in ROOT6

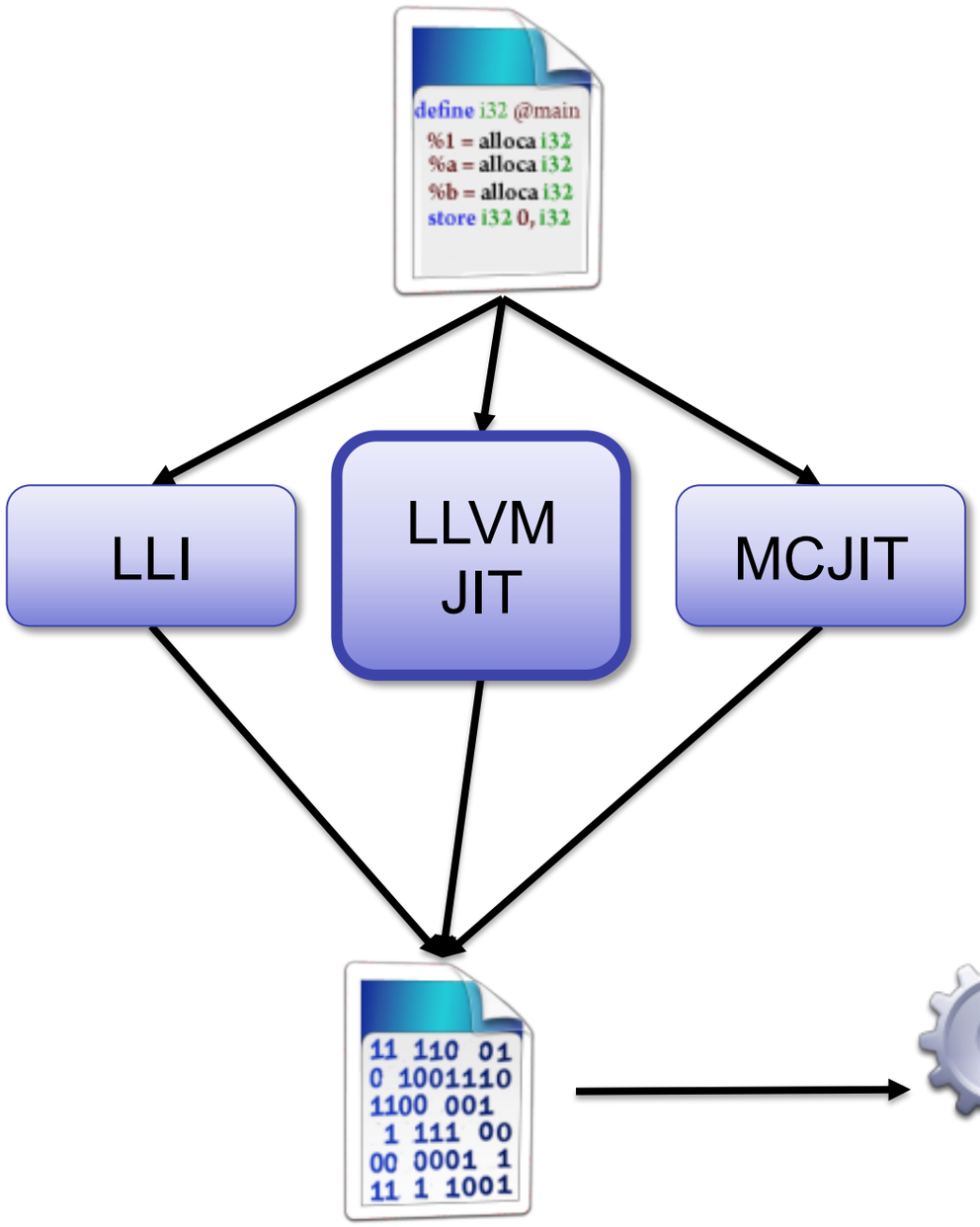


①, ② –*Text*
③, ④, ⑤, ⑥ – *Abstract Syntax Tree*
⑦ – *LLVM Bit Code*
⑧ –*Target Machine Code*

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.

```
[cling]$ .L libz.so
[cling]$ #include "zlib.h"
[cling]$ zlibVersion()
(const char * const) "1.2.3.3"
[cling]$
```


ECO
12:49
x10 r/min
km/h
20℃
8km
0.0
TRIP

ROOT6 Performance

”Premature optimization is the root of all evil” D. Knuth
BUT

- ALICE were first to try it. Thanks for the work and feedback to Peter Hristov
- Experiments started adopting ROOT6 and reported +1GB more memory usage in comparison to ROOT5
- Execution speed slightly in favor of ROOT6

ROOT6 Performance Footprint

- Experiments started adopting ROOT6 and reported +1GB more memory usage. We knew why.
  We were seeing much more reflection information, than was necessary in most cases. Also, we found out bugs and issues in the implementation in ROOT.
- For ~3months we reduced the overhead by -900MB to -1010GB (i.e. 10MB less than ROOT 5) depending on the workflows.
  We have a plan how to reduce it even further, C++ Modules.

ROOT6-Specific Developments

- New TTree reader
- New TFormula
- IO of interpreted classes
- More to come

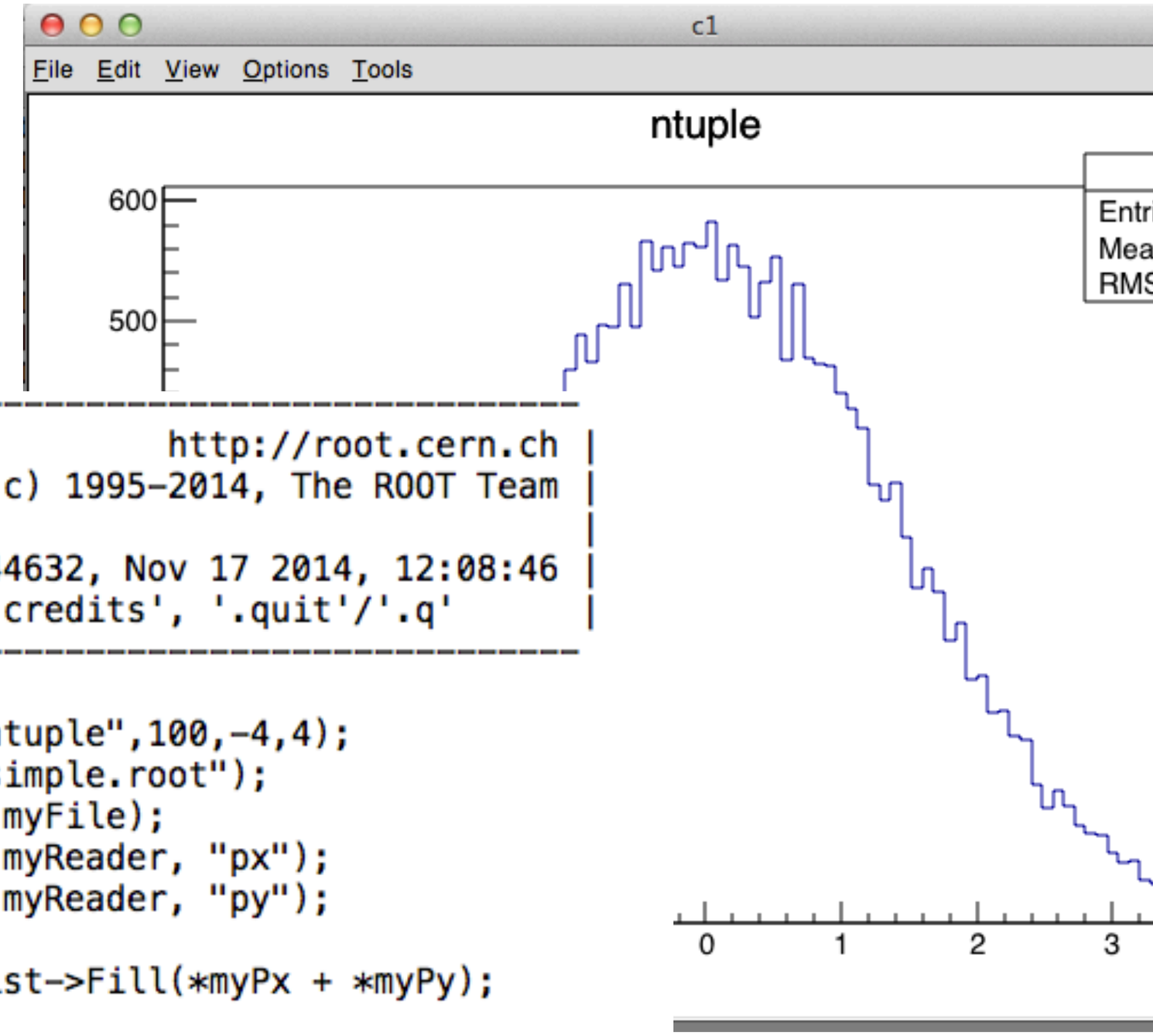
  Automatic differentiation. Runtime dictionary creation.

IO of interpreted classes

```
root [0] struct MyStruct { int Member; } var;
root [1] var.Member = 12;
root [2] TFile* file = TFile::Open("test.root", "RECREATE");
root [3] file->WriteObjectAny(&var, "MyStruct", "var");
root [4] .ls
TFile**         test.root
 TFile*         test.root
  KEY: MyStruct var;1   object title
root [5]
```

ROOT6-specific developments

ROOT's new TTree reader




```
 ------------------------------------------------------------------
| Welcome to ROOT 6.03/01                        http://root.cern.ch |
|                                      (c) 1995-2014, The ROOT Team |
| Built for macosx64                                                |
| From heads/master@v6-00-01-1807-g5744632, Nov 17 2014, 12:08:46   |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'        |
 ------------------------------------------------------------------

root [0] TH1F *myHist = new TH1F("h1","ntuple",100,-4,4);
root [1] TFile *myFile = TFile::Open("hsimple.root");
root [2] TTreeReader myReader("ntuple", myFile);
root [3] TTreeReaderValue<Float_t> myPx(myReader, "px");
root [4] TTreeReaderValue<Float_t> myPy(myReader, "py");
root [5]  while (myReader.Next()) {
root (cont'ed, cancel with .@) [6]  myHist->Fill(*myPx + *myPy);
root (cont'ed, cancel with .@) [7]}
root [8] myHist->Draw();
```

Why using ROOT6

- Get latest and greatest C++ language and library features easily
- Improve correctness
- No difference between compiled/interpreted worlds
- Backward and forward compatible
- Open up to new developments coming through ROOT
- Ensure your framework's sustainability

Future Release Plans

- v6.03/02 (end of January)
- v6.03/04 (end of March)
- v6.04/00 (end of May)

Backporting new features to ROOT5 not be feasible.

Thank you!

Backup slides

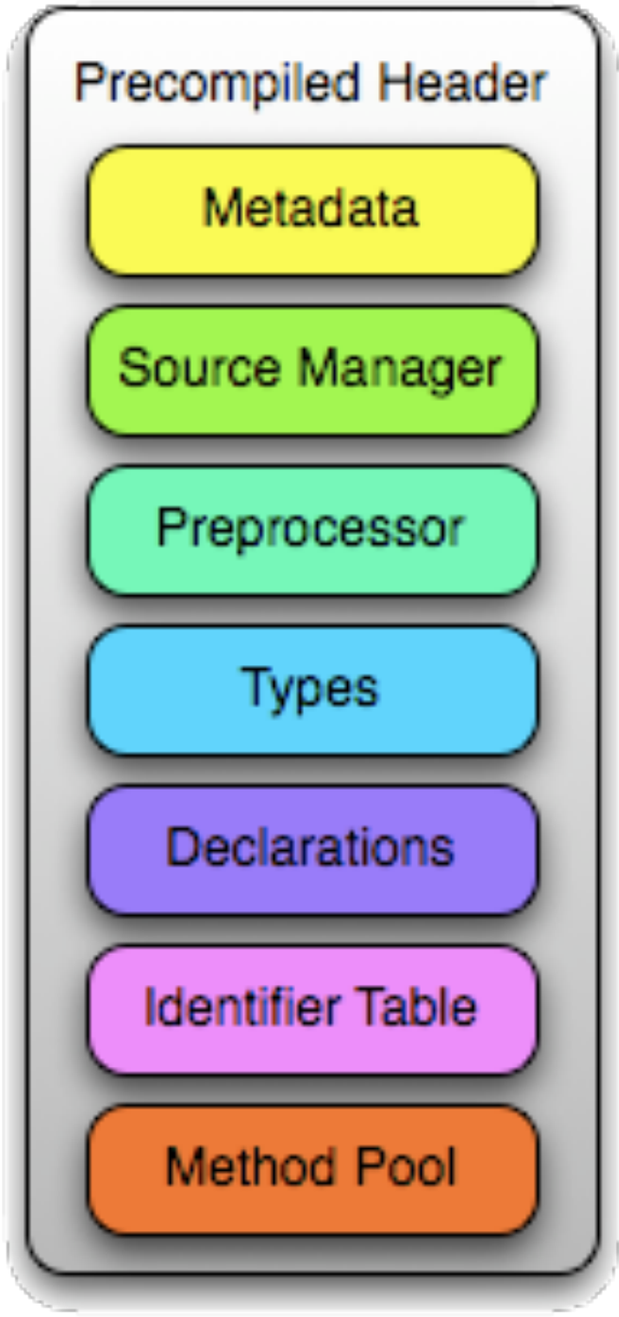
Future Plans

- Migrate to MCJIT
  - Object file emitted to memory
  - Runtime dynamic linker
- Windows 64 Support
- Tools
  - Automatic Differentiation

Pre-Compiled Headers/Modules

Carefully crafted data structures designed to improve translator's performance:

- Reduce lexical, syntax and semantic analysis
- Loaded “lazily” on demand



Pre-Compiled Headers/Modules

Design advantages:

- Loading is significantly faster than re-parsing
- Minimize the cost of reading
  - Read times don't depend on size
  - Cost of generating isn't large

PCH

PCM

And thus PCM much more flexible