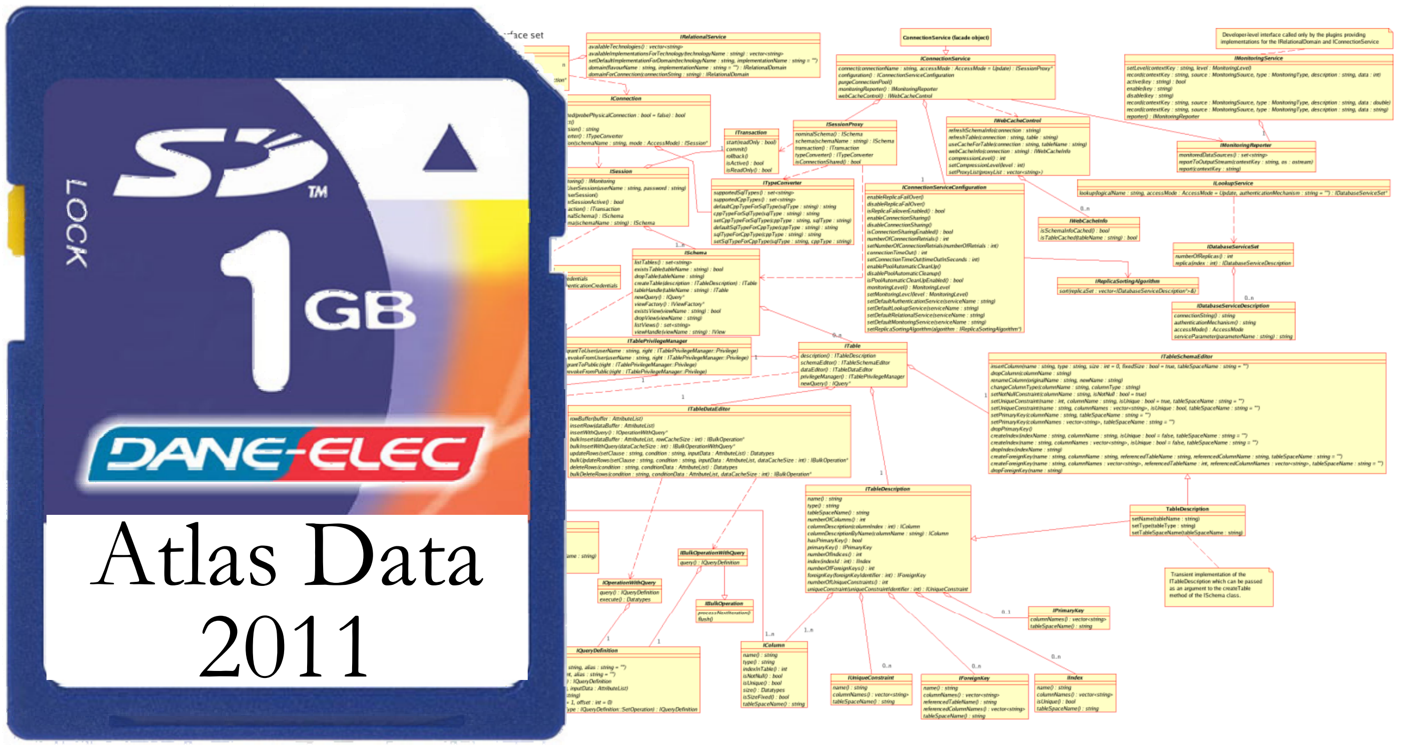
C++ and Data

An overview of serialization in C++



Axel Naumann (CERN), Philippe Canal (Fermilab)

What Data? Why C++?

- Experiments' frameworks: C++
- Physicists' analyses: C++
- High performance, collaborative development,…
- Experiments' data: C++ objects on tape

→ Serialization with C++!

Ingredients

Storing data means:

- Reflection: types? members?
- Introspection: what is its type?
- Object instantiation from type / destruction
- I/O: memory to disk and back (endianness)

- Pointer / References ([un]swizzling)
- Schema Evolution: enabling changes

Ingredients

» Language support requested:

- Reflection: types? members?
- Introspection: what is its type?
- Object instantiation from type / destruction
- I/O: memory to disk and back (endianness)

» I/O framework's job (e.g. ROOT):

- Pointer / References ([un]swizzling)
- Schema Evolution: enabling changes

Ingredients

» Language support requested:

- Reflection: types? members?
- Introspection: what is its type?
- Object instantiation from type / destruction
- I/O: memory to disk and back (endianness)

» I/O framework's job (e.g. ROOT):

- Pointer
- Schema Evolution: enabling changes

not covered here!

Serialization Everywhere

Other languages offer some of these ingredients, usually excluding pointer / reference swizzling, schema evolution:

- Java
```
class C implements Serializable
```
- Python
```
cPickle.dump(myObj, file, -1)
```
- .NET
```
[Serializable] class C
```

Serialization In C++?

C++ supports the following ingredients:

- Reflection: **missing**
- Introspection: basic, fragile (`typeid`)
- Type to instantiation: **missing**
- Raw I/O: yes, endianness: **missing**
- Pointer Swizzling: **missing**
- Schema Evolution: **missing**

Serialization Not In C++

None of the relevant ingredients supported

Must rely on external packages, using e.g.

- templates (type description level)
- `typeid` (introspection)
- CPP macros

Look at consequence of matching external packages to custom code

Intrusiveness

Types modified to add serialization support

Often base, friend, etc.

» Do I need to change the header?

Example:

Microsoft's MFC: inheritance from CObject;

```
class C: public CObject
```

Reflex: no requirements

Dictionary: What's In A Type

Explicit enumeration of members / bases?

Example: boost::serialization (paraphrased)

```
class C {
  void serialize(Archive& ar) {
    ar & m;
  }
  std::string m;
};
```

Reflex: dictionaries from headers; part of build

Object Construction

Serialization:

1. create object in memory given its type name,
2. fill object with stored data

Object construction needs access to constructor

Why not add access to *all* public functions?

Interpreter Access

Reflection allows interactive use of classes

```
gROOT->ProcessLine("A::f(1)")
```

Interpreter knows type A, function f(), e.g.:

```
ClassBuilder("A").
AddFunction("f", Type("int"))
```

And how to pass arguments – using **stubs**:

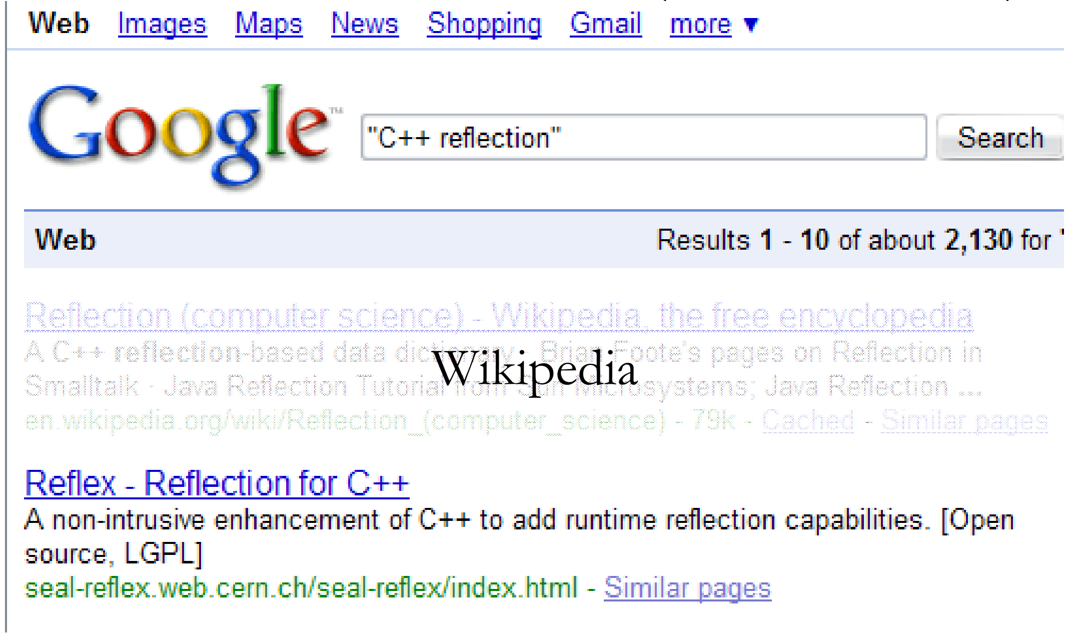
```
void stub_A_f(void* args[]){
  A::f((int)args[0]); }
```

Reflection – Market Overview

Database of available types and their structure

Main available C++ reflection libraries (unordered) :

- XCppRefl
- CppReflection



Web Images Maps News Shopping Gmail more ▾

"C++ reflection" Search

Web Results 1 - 10 of about 2,130 for '

Reflection (computer science) - Wikipedia, the free encyclopedia
A C++ reflection-based data dictionary; Brian Foote's pages on Reflection in Smalltalk - Java Reflection Tutorial from Sun Microsystems; Java Reflection ...
en.wikipedia.org/wiki/Reflection_(computer_science) - 79k - Cached - Similar pages

Wikipedia

Reflex - Reflection for C++
A non-intrusive enhancement of C++ to add runtime reflection capabilities. [Open source, LGPL]
seal-reflex.web.cern.ch/seal-reflex/index.html - Similar pages

- ROOT's Reflex:

  Google's #1 product for "C++ reflection" – no wonder industry cares about it…

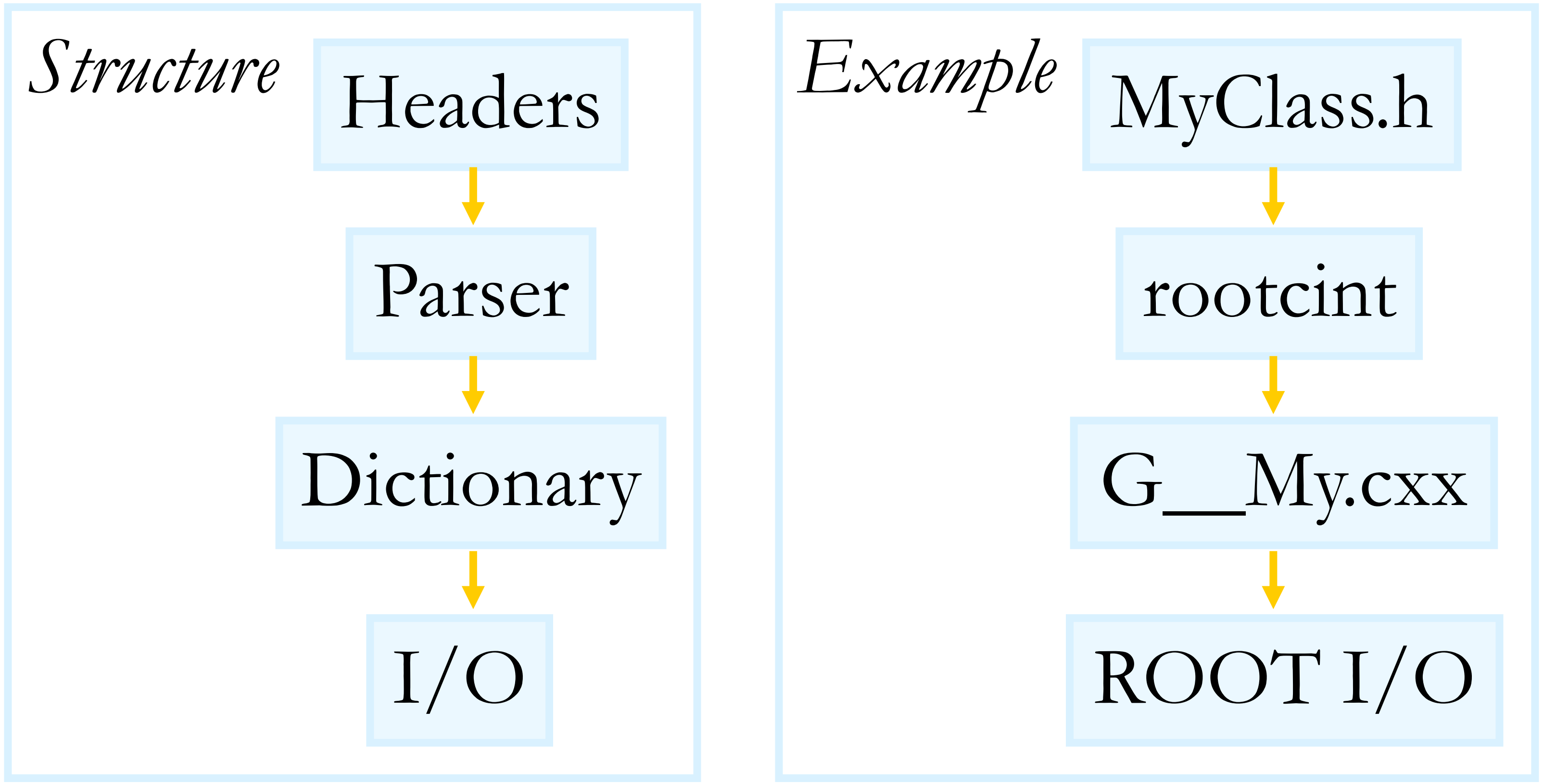
Reflection and Dictionaries

Reflection is database, dictionary is data.

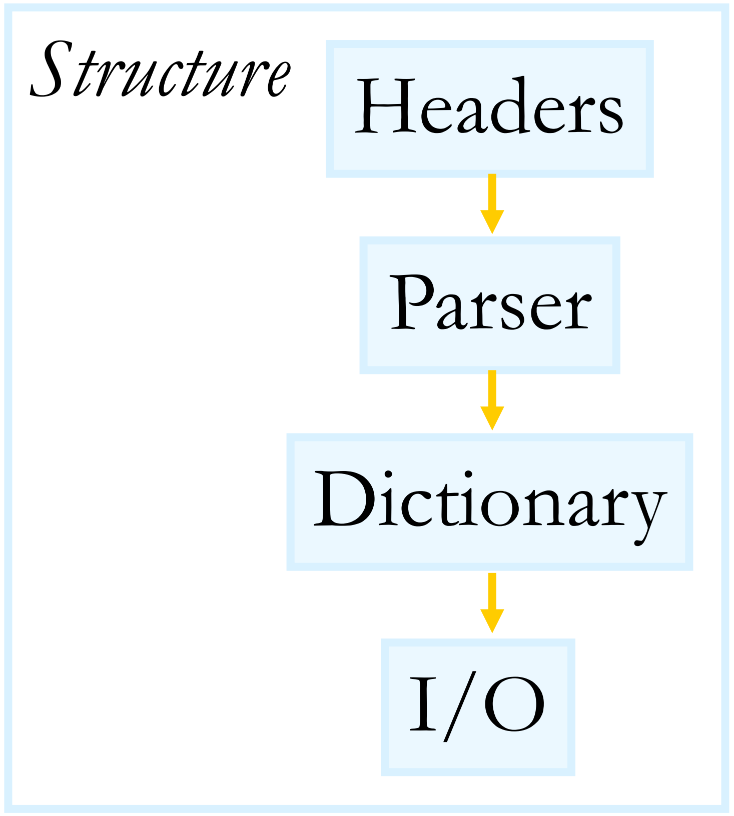
Refection data generated from

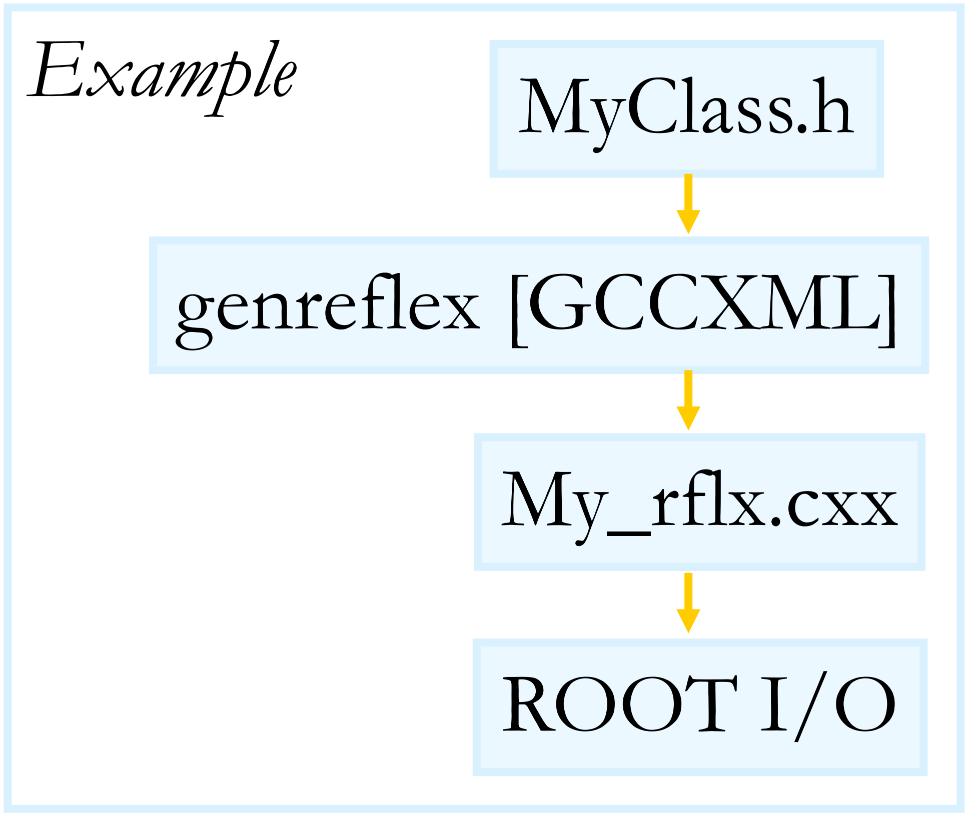
- user: `Reflect::AddClass("MyClass")`
- headers using modified compiler: GCCXML
- headers using custom parser: CINT
- debug information

Overview Of Reflection Data



Overview Of Reflection Data





Reflection Data

Dictionary creation: time consuming

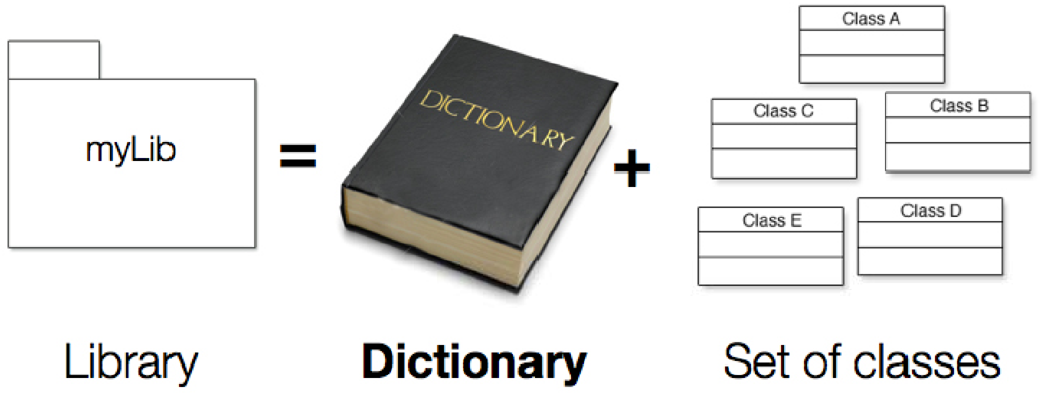
Currently persistent as generated C++, excerpt:

```
static void G__setup_memfuncTObjArray(void) {
G__tag_memfunc_setup(G__get_linked_tagnum(&G__G__ContLN_TObjArray));
G__memfunc_setup("BoundsOk",805,(G__InterfaceMethod) NULL, 103,
  -1, G__defined_typename("Bool_t"), 0, 2, 1, 2, 8,
  "C - - 10 - where i - 'Int_t' 0 - at", (char*)NULL, (void*) NULL, 0);
G__memfunc_setup("Init",404,(G__InterfaceMethod) NULL, 121, -1, -1, 0, 2, 1, 2, 0,
   "i - 'Int_t' 0 - s i - 'Int_t' 0 - lowerBound", (char*)NULL, (void*) NULL, 0);
G__memfunc_setup("TObjArray",878,G__G__Cont_81_0_5, 105,
   G__get_linked_tagnum(&G__G__ContLN_TObjArray), -1, 0, 2, 1, 1, 0,
   "i - 'Int_t' 0 'TCollection::kInitCapacity' s i - 'Int_t' 0 '0' lowerBound",
   (char*)NULL, (void*) NULL, 0);
G__memfunc_setup("TObjArray",878,G__G__Cont_81_0_6, 105,
   G__get_linked_tagnum(&G__G__ContLN_TObjArray), -1, 0, 1, 1, 1, 0,
   "u 'TObjArray' - 11 - a", (char*)NULL, (void*) NULL, 0);
G__memfunc_setup("operator=",937,G__G__Cont_81_0_7, 117,
   G__get_linked_tagnum(&G__G__ContLN_TObjArray), -1, 1, 1, 1, 1, 0,
   "u 'TObjArray' - 11 - - ", (char*)NULL, (void*) NULL, 0);
G__memfunc_setup("Clear",487,(G__InterfaceMethod) NULL,121, -1,-1, 0, 1, 1, 1,
   0, "C - 'Option_t' 10 '\"\"' option", (char*)NULL, (void*) NULL, 1);
```

Reflection Data

Dictionary sources compiled, linked into library

Become part of *enhanced* library:



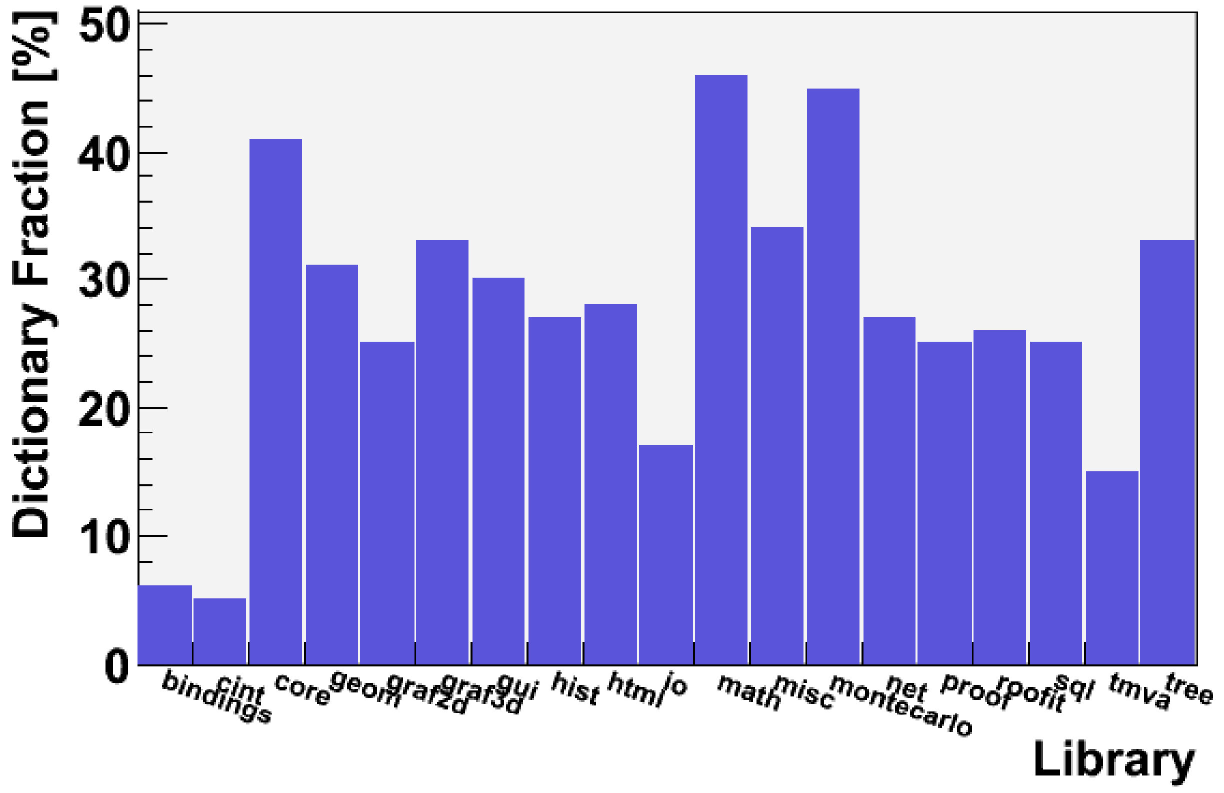
Alternative: keep separate dictionary library

Reflection Data

Dictionaries contain large amount of data

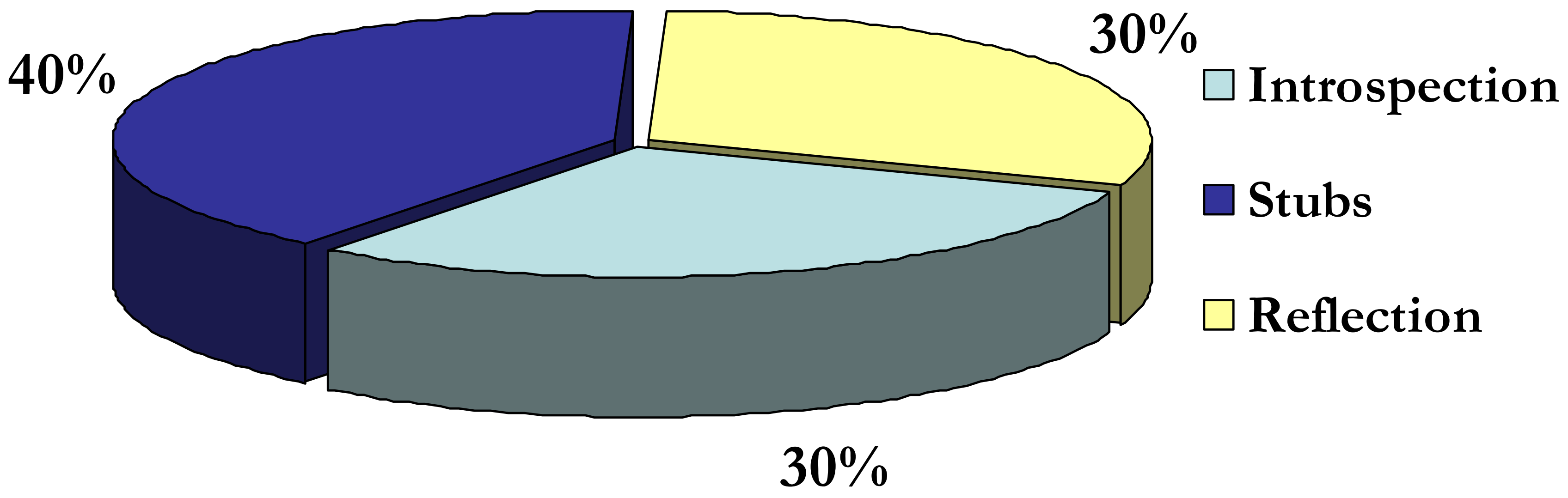
About 1/3 of library size: depends on amount of

- templates,
- functions,…



Dictionary Size

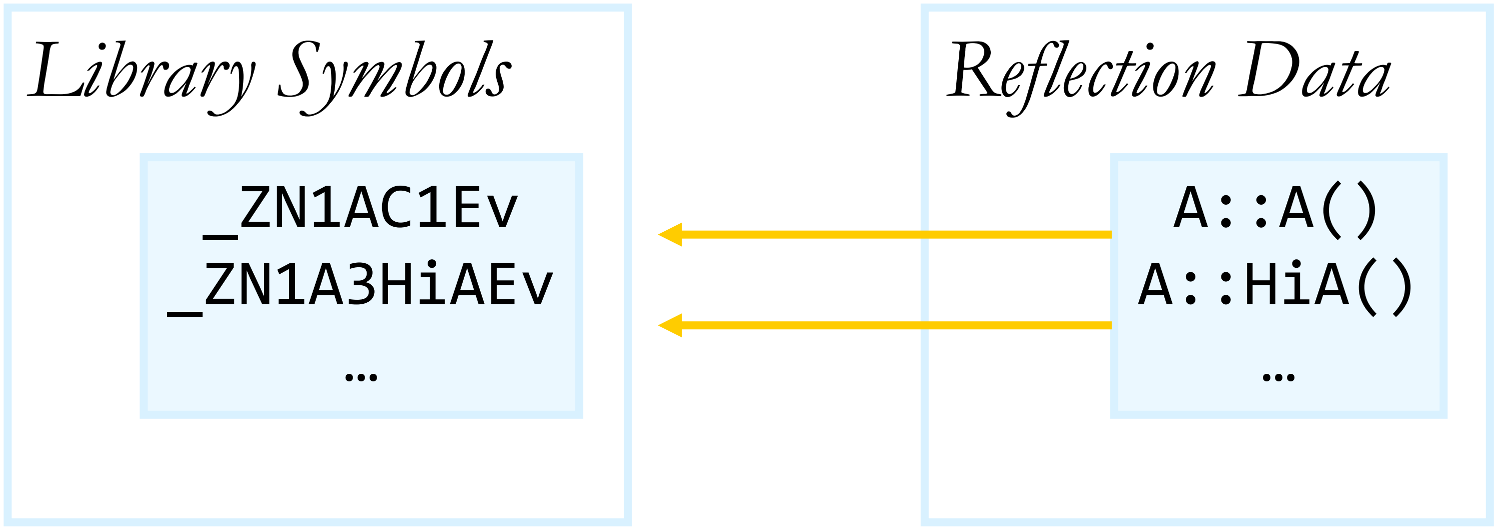
- Dictionary stub for each function
- Entry for each type
- Entry for each member (data, function)
- Names, types, parameters…



Reflection Data Optimization

Reflection data, function access optimization

- Load on demand
- Less / no string copies
- No stubs: use library symbols, prepare call stack

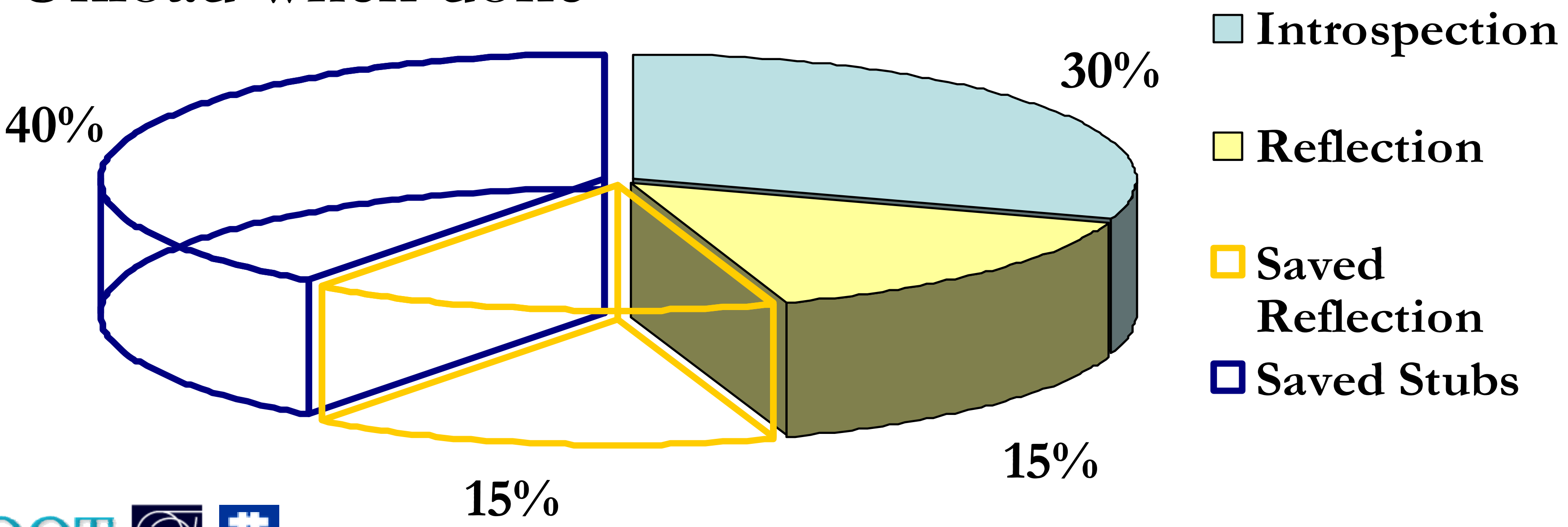


Reflection Data Optimization

ROOT will soon serialize reflection objects

Proof of concept already implemented

- Reduce disk space
- Improve build (no libraries)
- Unload when done



Summary: C++ And Data

An incredibly complex relationship

Understood, mastered, optimized in HEP

Visible outside HEP, sought-after by industry

And we did not even talk about I/O…