Reflex
reflection for C++

Stefan Roiser, Pere Mato
CERN / PH / SFT
Philippe Canal
FNAL

Content

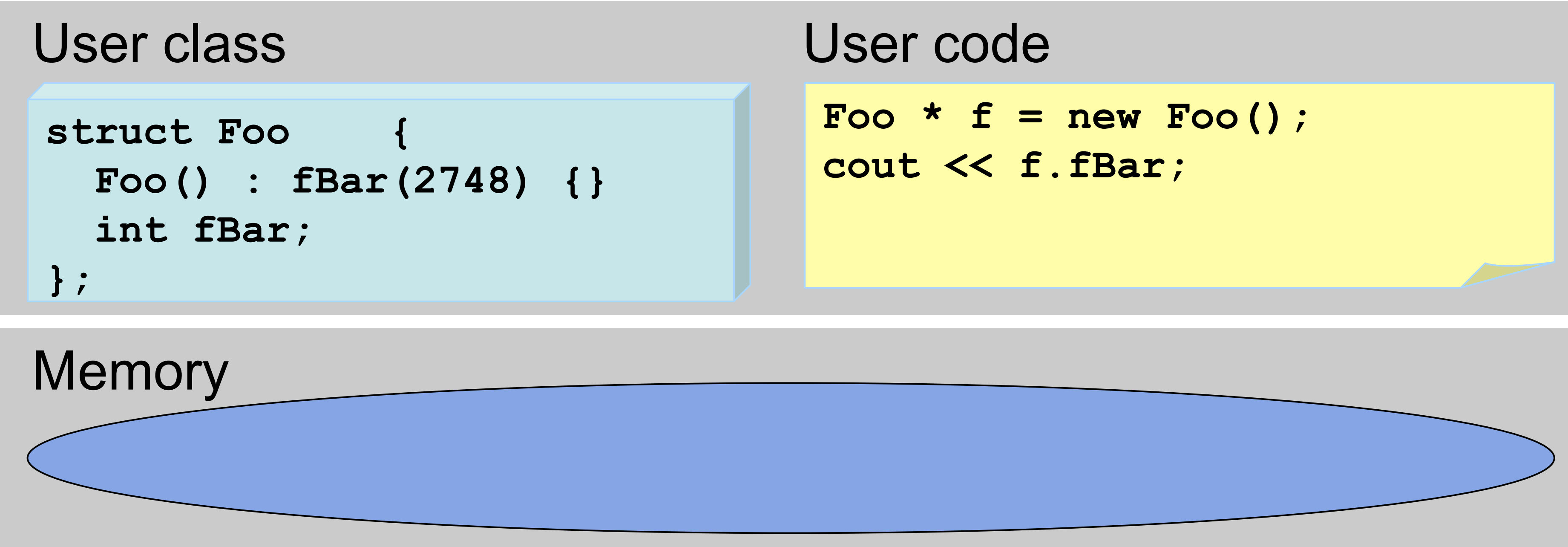
- Software runtime reflection
- Reflex
  - Design
  - API
  - Code Examples
  - Generating dictionaries
- Reflex in the context of ROOT
- Status / Summary

Definitions

- Reflection is the ability of a language to introspect it’s own structures at runtime and interact with them in a generic way

- A dictionary provides reflection information about types of a certain language to the user

What is Reflection?



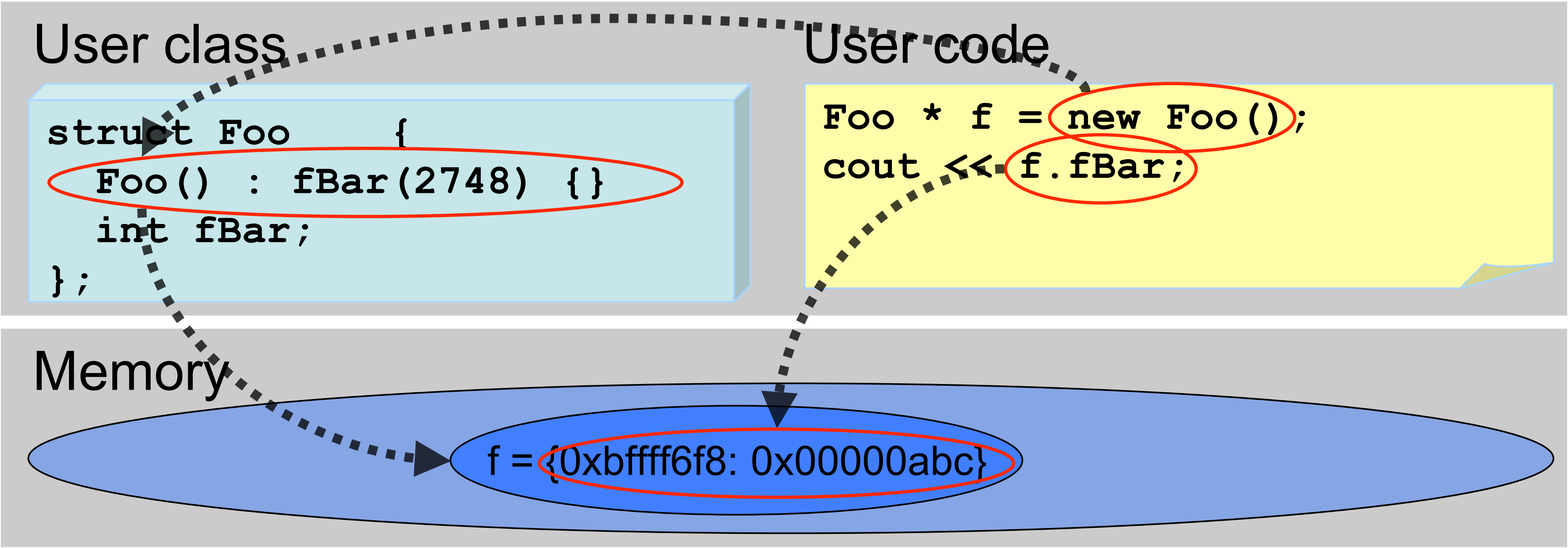
User class

```
struct Foo    {
  Foo() : fBar(2748) {}
  int fBar;
};
```

User code

```
Foo * f = new Foo();
cout << f.fBar;
```

What is Reflection?



User class

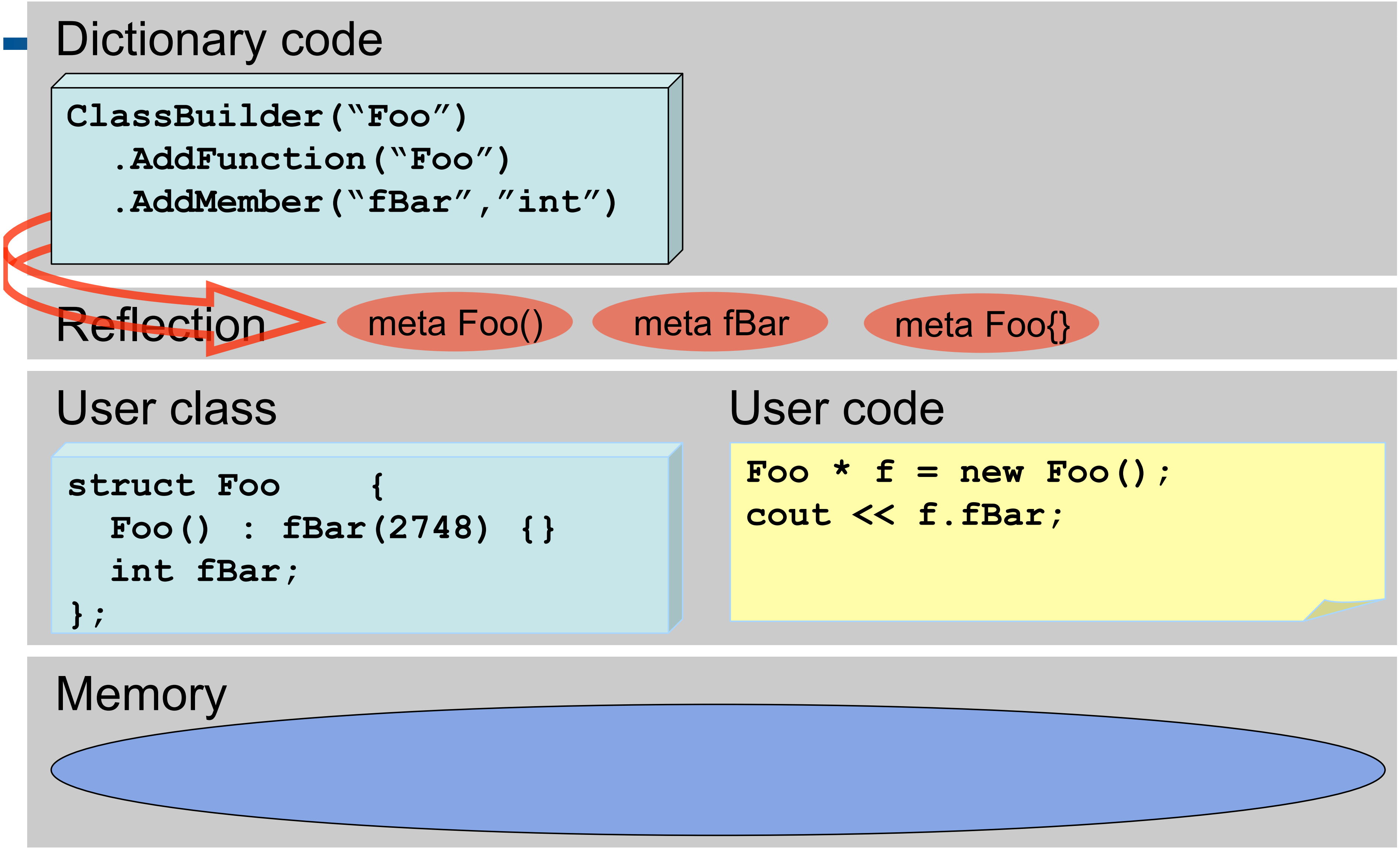
```
struct Foo    {
  Foo() : fBar(2748) {}
  int fBar;
};
```

User code

```
Foo * f = new Foo();
cout << f.fBar;
```

Memory

What is Reflection?



Dictionary code

```
ClassBuilder(“Foo”)
   .AddFunction(“Foo”)
   .AddMember(“fBar”,”int”)
```

Reflection

meta Foo()

meta fBar

meta Foo{}

User class

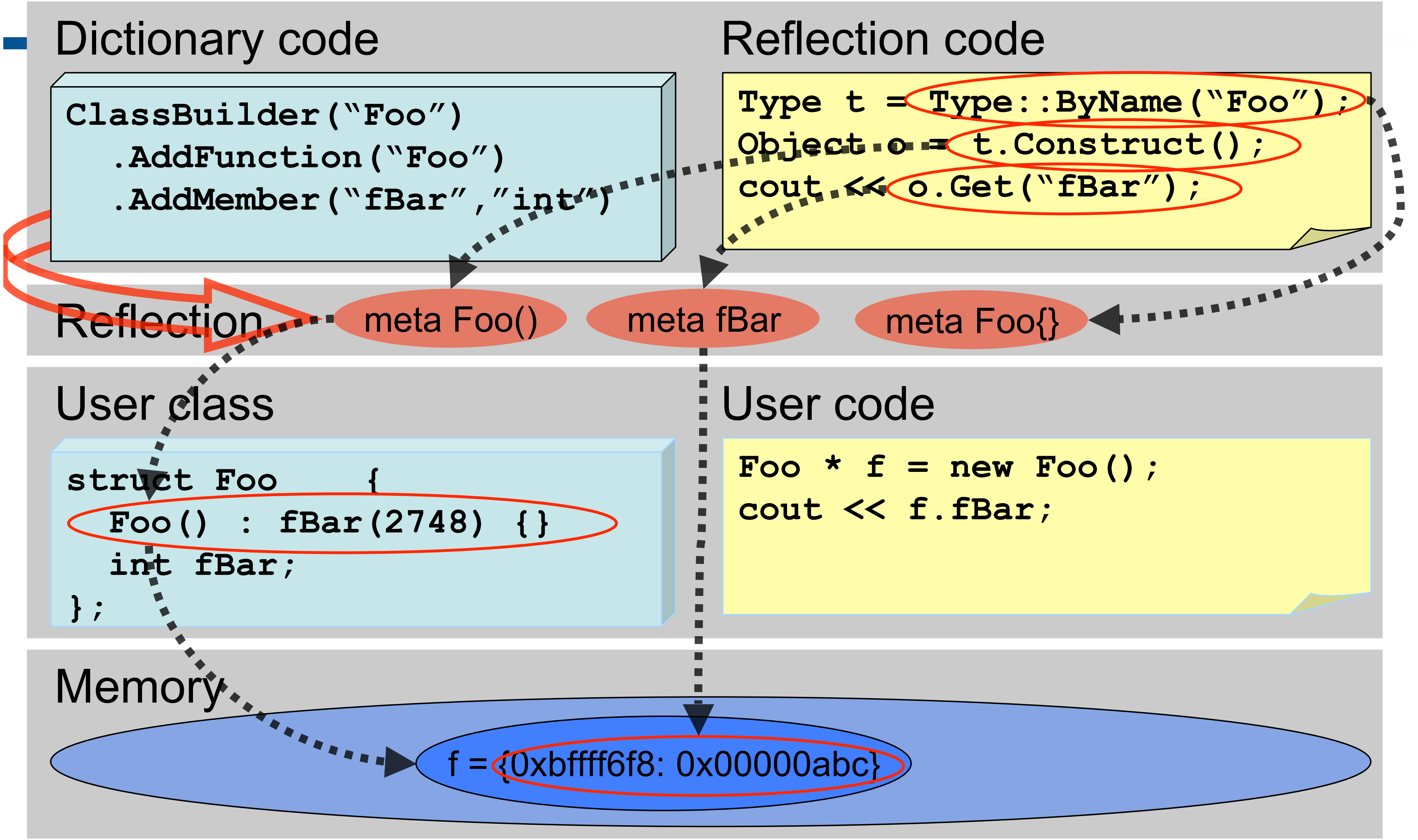
```
struct Foo     {
   Foo() : fBar(2748) {}
   int fBar;
};
```

User code

```
Foo * f = new Foo();
cout << f.fBar;
```

Memory

What is Reflection?



Dictionary code

```
ClassBuilder(“Foo”)
   .AddFunction(“Foo”)
   .AddMember(“fBar”,”int”)
```

Reflection code

```
Type t = Type::ByName(“Foo”);
Object o = t.Construct();
cout << o.Get(“fBar”);
```

Reflection

User class

```
struct Foo     {
   Foo() : fBar(2748) {}
   int fBar;
};
```

User code

```
Foo * f = new Foo();
cout << f.fBar;
```

Memory

Reflection and C++

- C++ inherently provides Runtime Type Information (RTTI)
  - RTTI gives you a (mangled) name
  - plus a unique address of a type
- We want to provide full C++ reflection
  - Useful for
    - Persistence of objects
    - Interactive usage of objects

Reflex

- Was already presented at CHEP’04 as design
- In Dec. ‘05 Reflex moved from SEAL to ROOT
- Goals
  - Enhance C++ with runtime reflection capabilities
  - Non intrusive towards user code
  - Automated dictionary code generation
  - Close to the C++ ISO/IEC 14882 standard
  - Light and standalone system
  - Small memory footprint
  - Multi platform (linux, win32, mac os, …)
  - Supports introspection, interaction and modification

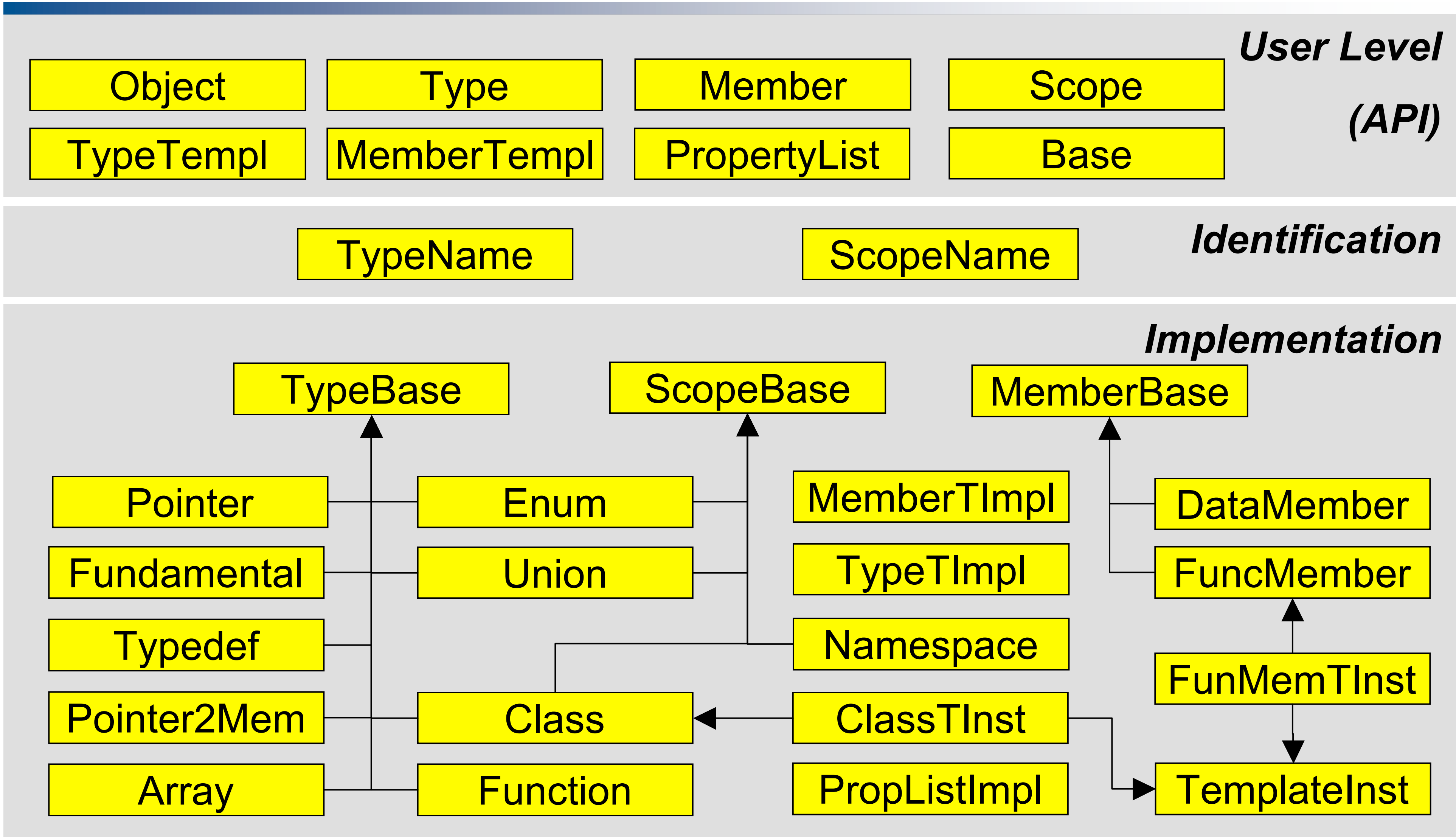
3 Levels of Reflection

- Introspection
  - – Retrieve information (e.g. class name, return type)
- Interaction
  - – Handle objects (e.g. create instance, call function, get/set data member)
- Modification
  - – Change information (e.g. add function member, add properties, add class template instance)

Design ideas

- User classes (API)
  - 8 lightweight classes provide full C++ reflection
  - Functionality, mainly through forwarding functions
  - Small memory allocation for user classes
    - ~ (sizeof (Pointer) + sizeof(int))
    - By value semantics
- Implementation of state pattern
  - Done via an “Identification” layer (~ meta RTTI)
  - Seamless loading / unloading of dictionary information

Reflection Model



User Classes

With examples for introspection and interaction

class Type sizeof, declaring scope, array length, construct, destruct	**class Scope** declaring scope, sub scopes, sub types, members
class Member type, declaring scope, get/set data member, call function	**class Object** type, address, destruct, get/set data member, call function
class Base base type, offset, modifiers	**class PropertyList** get/set key(string)/value(Any) pairs to Types/Scopes/Members
class MemberTemplate template parameters, default parameters, template instances	**class TypeTemplate** template parameters, default parameters, template instances

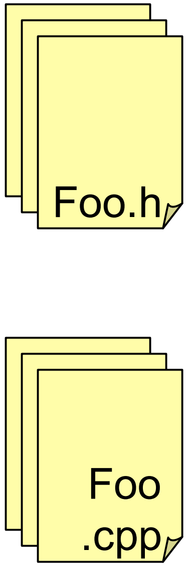
Example: Introspection

```
// The Reflex namespace inside Root
using namespace ROOT::Reflex;

// Get type by its name
Type cl = Type::ByName(“Particle”);

// If class print all data members
if ( cl.IsClass() ) {
  for ( Member_Iterator mi = cl.DataMember_Begin();
        mi != cl. DataMember_End(); ++mi ) {
    cout << mi->Type().Name(SCOPED) << " " << mi->Name() <<";";
    // output comment line if exists
    if ( mi->PropertyList().HasKey("comment") ) {
      cout << mi->PropertyListGet().PropertyAsString("comment");
    }
    cout << endl;
  }
}
```

Example: Interaction

```
// Get a type by its name
Type cl = Type::ByName(“Particle”);

// Instantiate an instance
Object obj = cl.Construct();

// Call a method
Object ret = obj.Invoke(“myFunction”);

// Alternatively
for ( Member_Iterator mi = cl.FunctionMember_Begin();
      mi != cl.FunctionMember_End(); ++mi ) {
  if (mi->Name() == “myFunction”) {
    ret = mi->Invoke(obj);
  }
}

// Delete the instance
obj.Destruct();
```

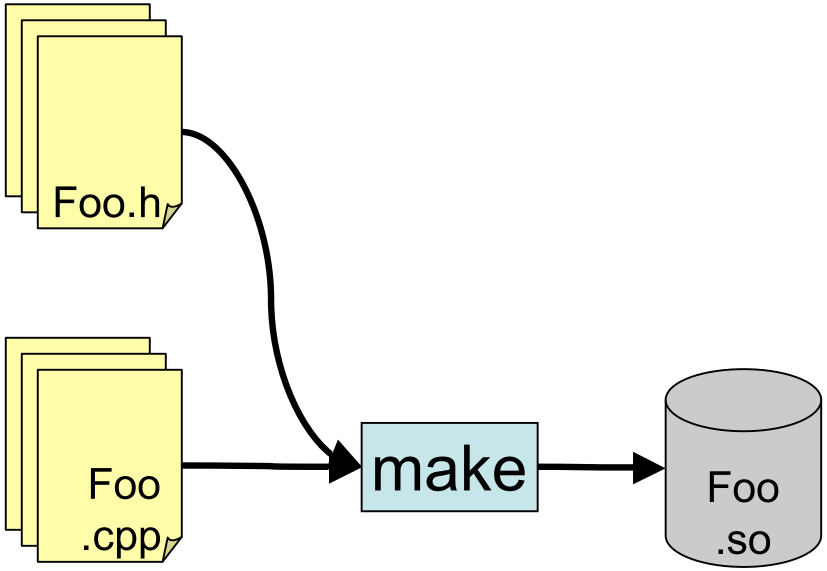
Dictionary Generation



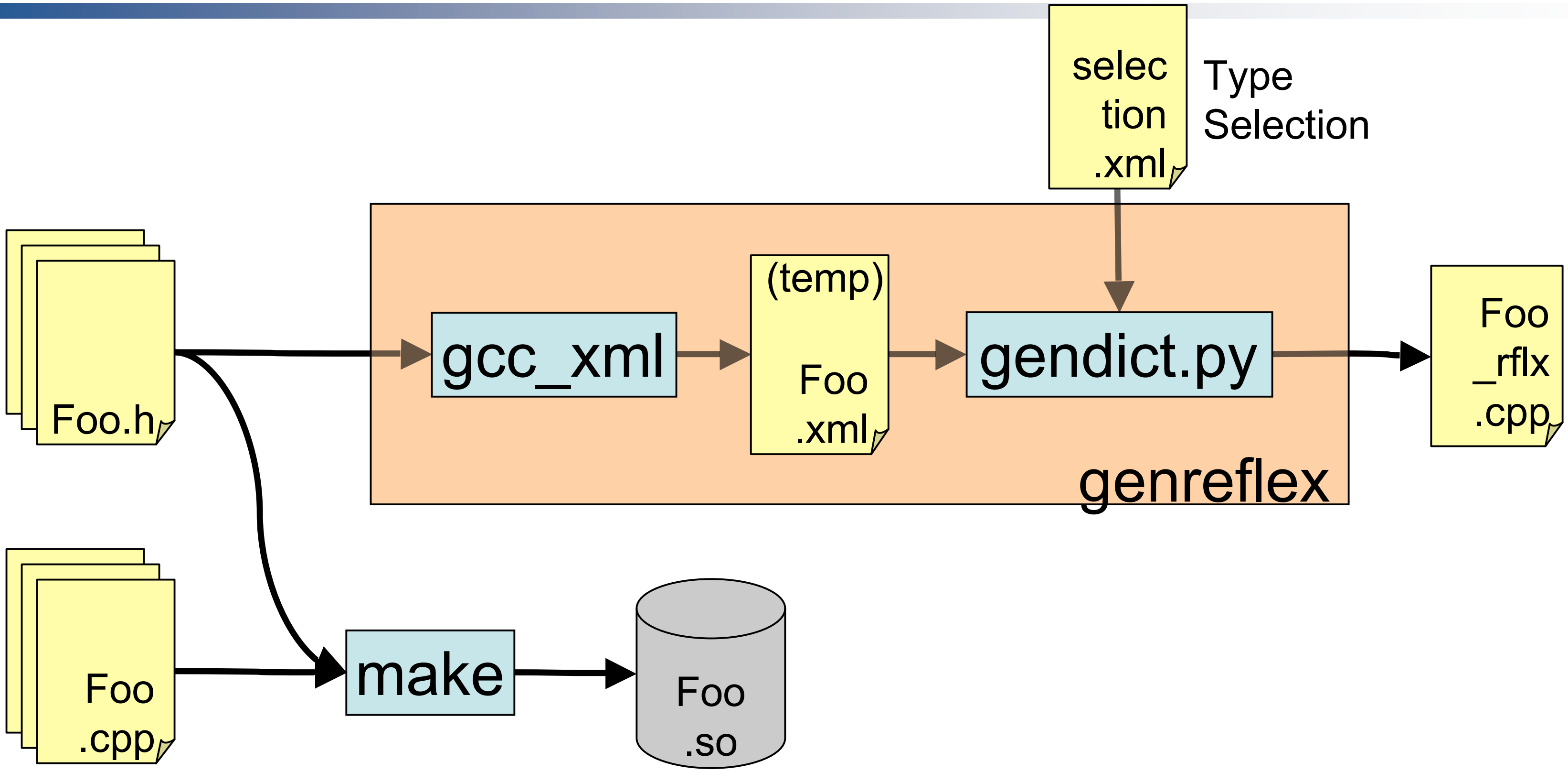
Foo.h

Foo
.cpp

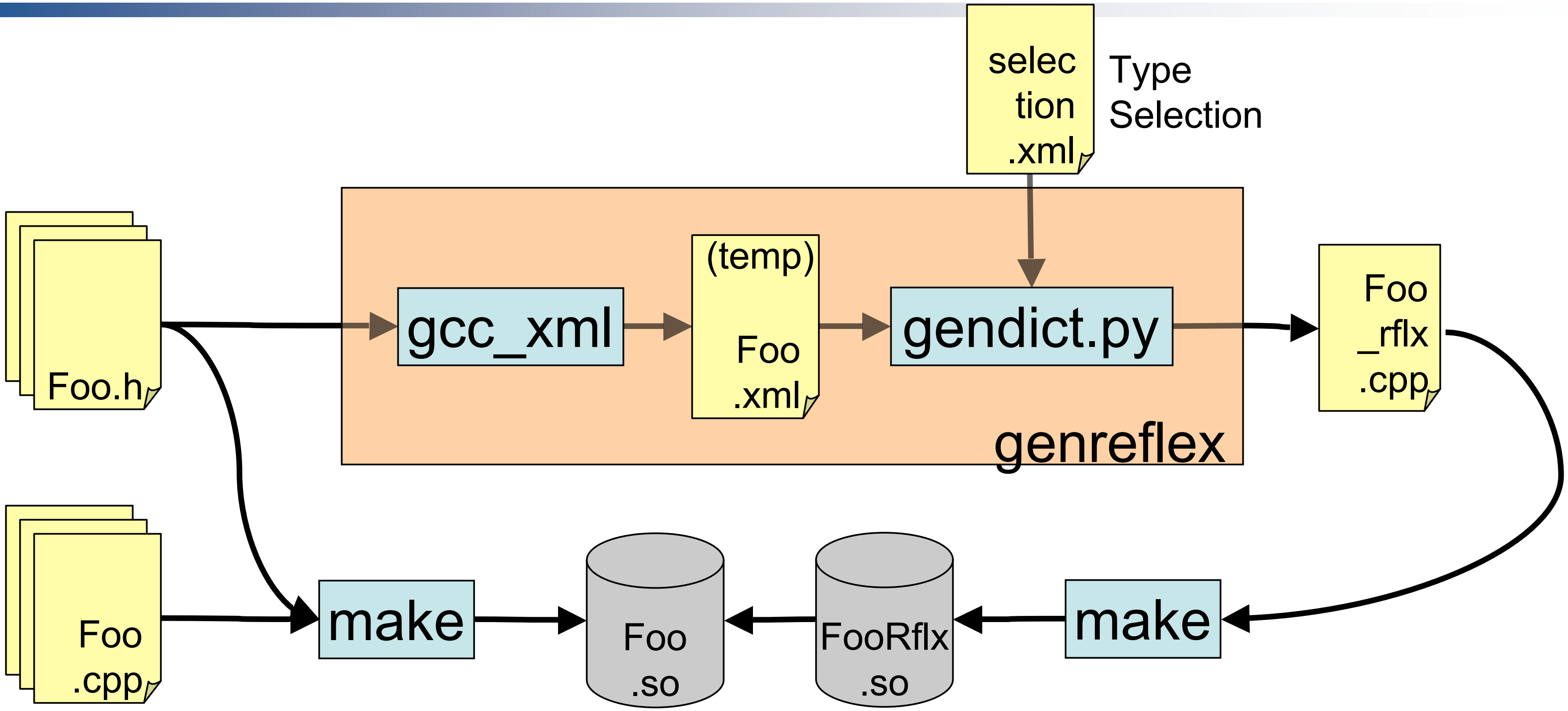
Dictionary Generation



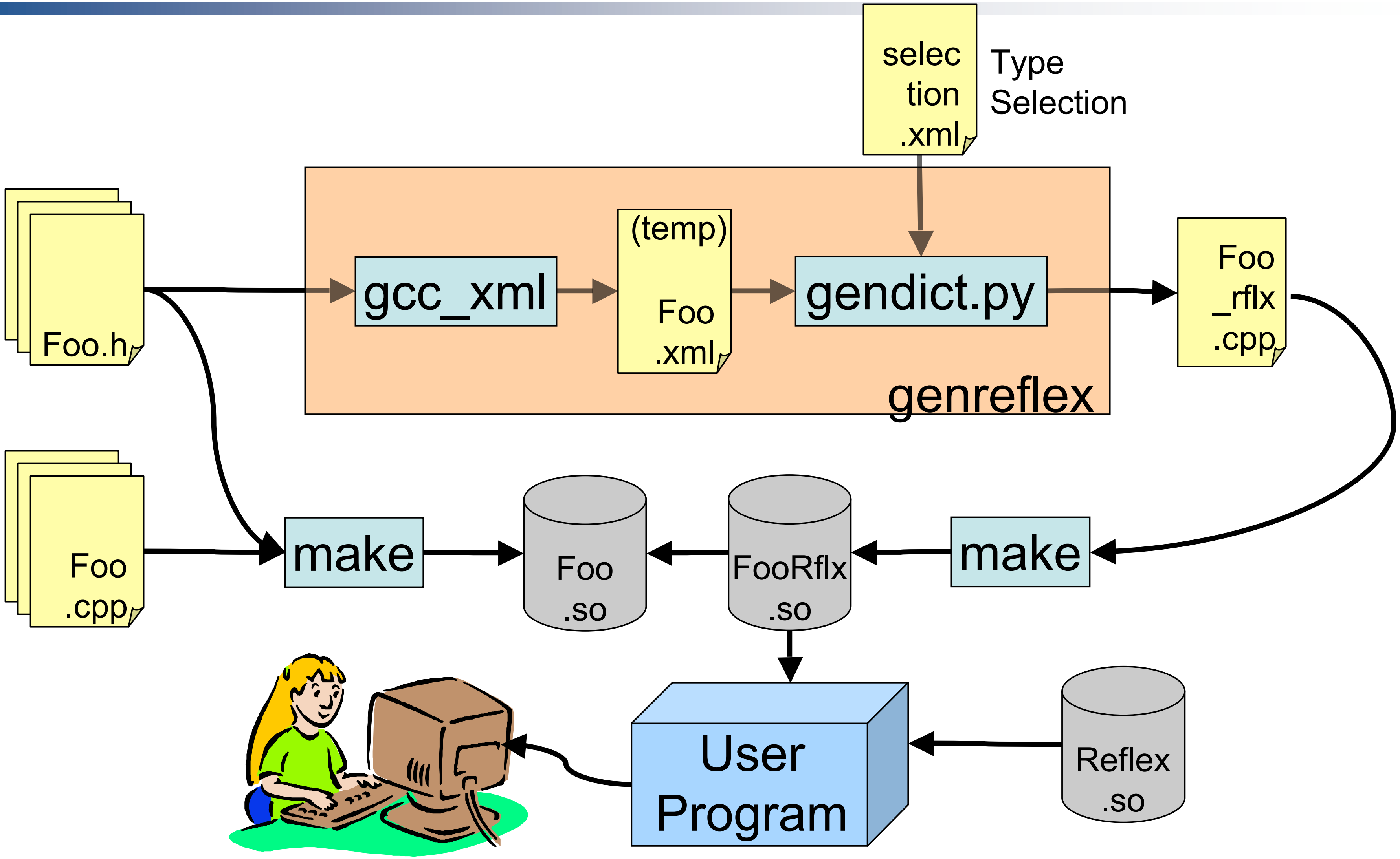
Dictionary Generation



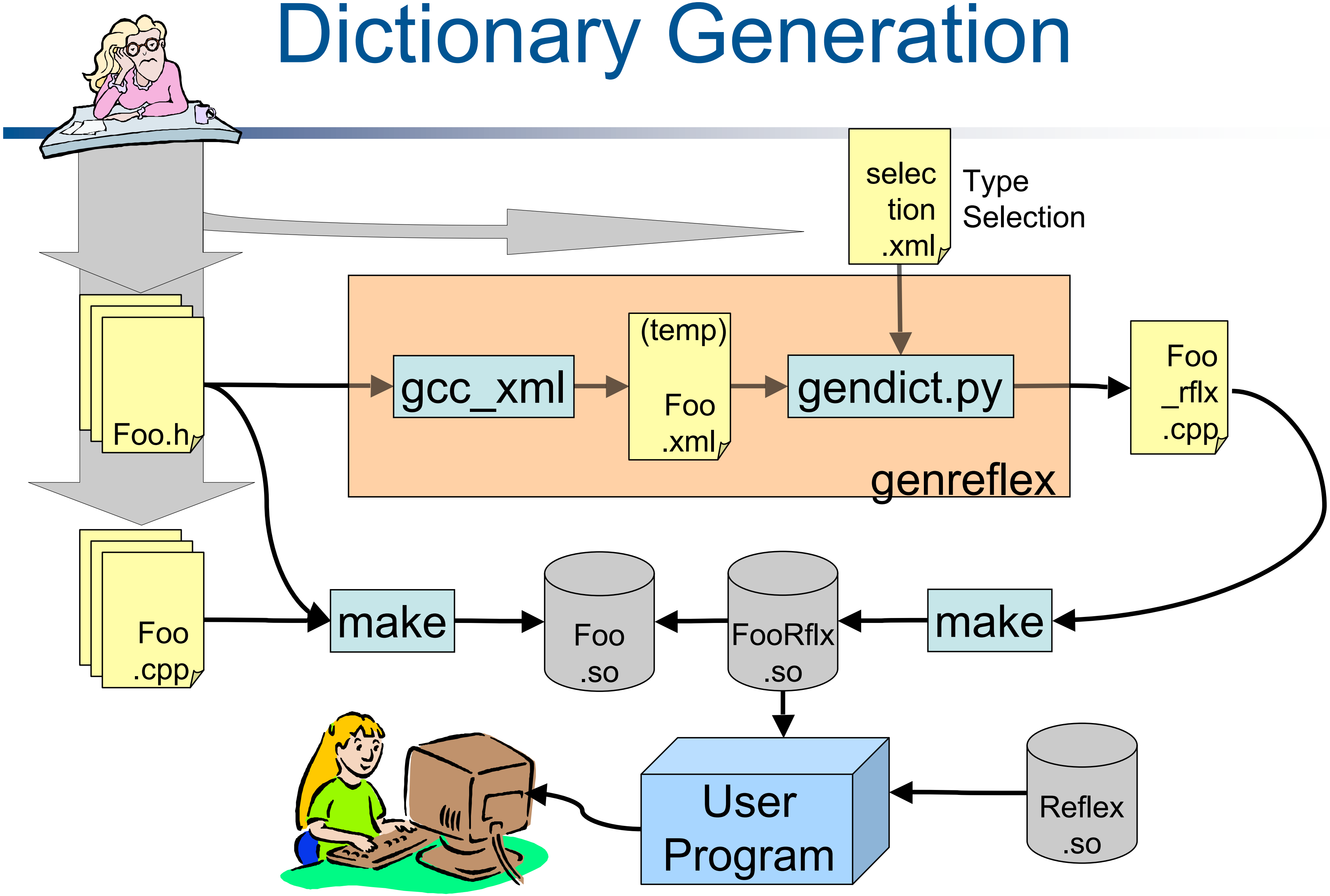
Dictionary Generation



Dictionary Generation



Dictionary Generation



Selecting types

- Written in XML
- Functionality
  - Allows inclusion/exclusion of types
  - Usage of patterns
  - Apply special information (transient, class ID)
  - Non-intrusive way of attaching information

```
<lcgdict>
  <class pattern=“T*”/>
  <class name=“Particle”>
    <field name=“fOnlyInMemory” transient=“true”/>
  </class>
  <exclusion>
    <class name=“Vertex”/>
  </exclusion>
</lcgdict>
```

gcc_xml

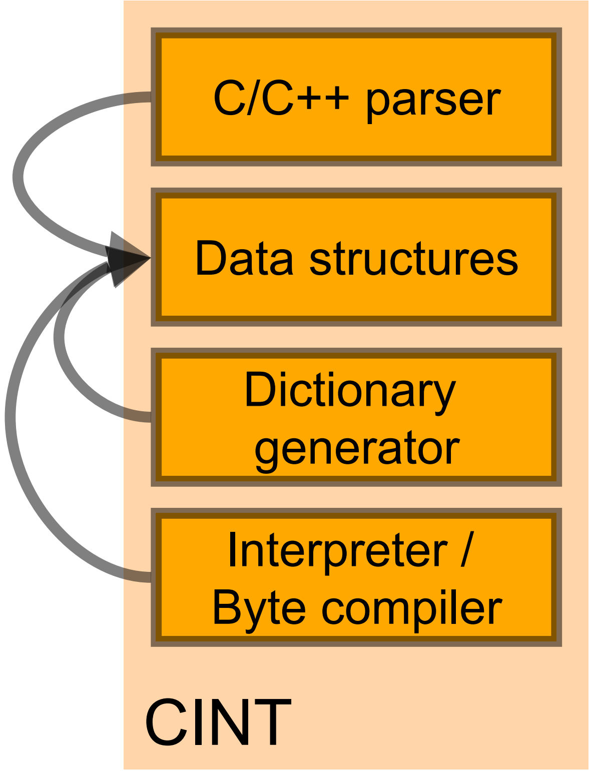
- “[…] generate an XML description of a C++ program from GCC's internal representation.”
- Any gcc compatible program can be used as input
- Multi platform (linux, win32, macos, …)

Reflex in ROOT

- CINT is the interpreter/dictionary system in ROOT
  - We change CINT to use Reflex
    - Agreed in CINT/Reflex workshop May ‘05 at CERN
    - Advantages
      - Less memory consumption
      - Better C++ compliance
      - LCG/POOL users only need to load 1 dictionary

Reflex in ROOT

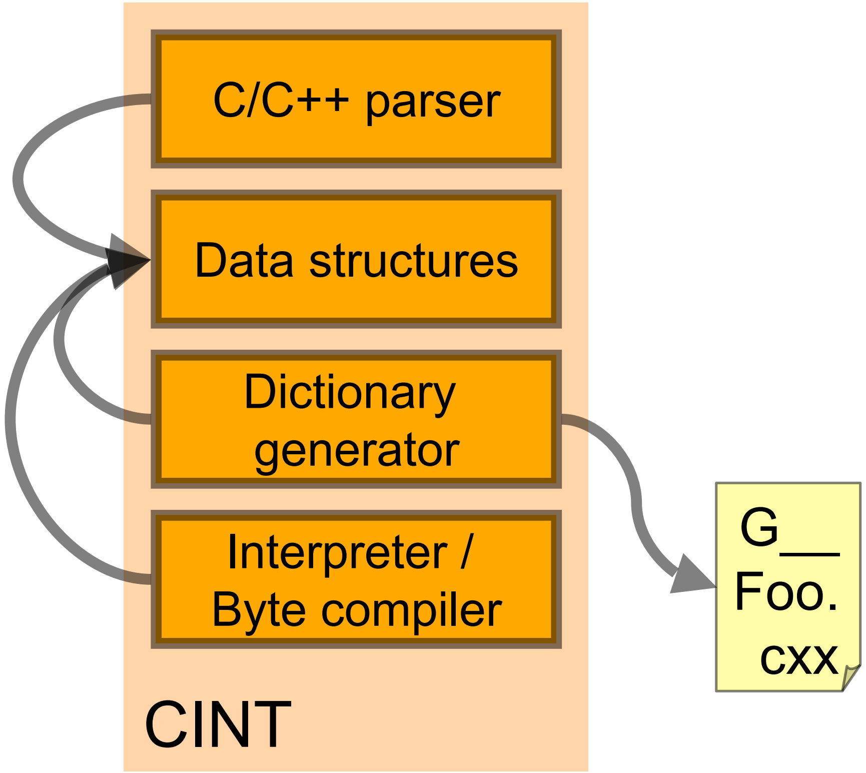
- CINT is the interpreter/dictionary system in ROOT
  - We change CINT to use Reflex



The work is done in 3 steps:

Reflex in ROOT

- CINT is the interpreter/dictionary system in ROOT
  - We change CINT to use Reflex

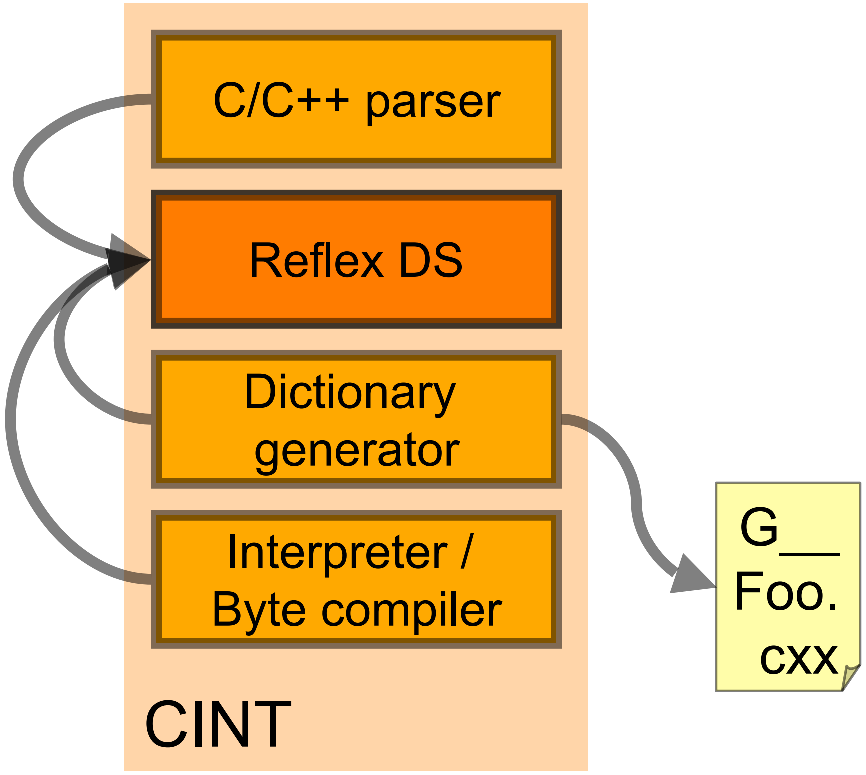


The work is done in 3 steps:

1. Generate Reflex dictionary source code via rootcint

Reflex in ROOT

- CINT is the interpreter/dictionary system in ROOT
  - We change CINT to use Reflex

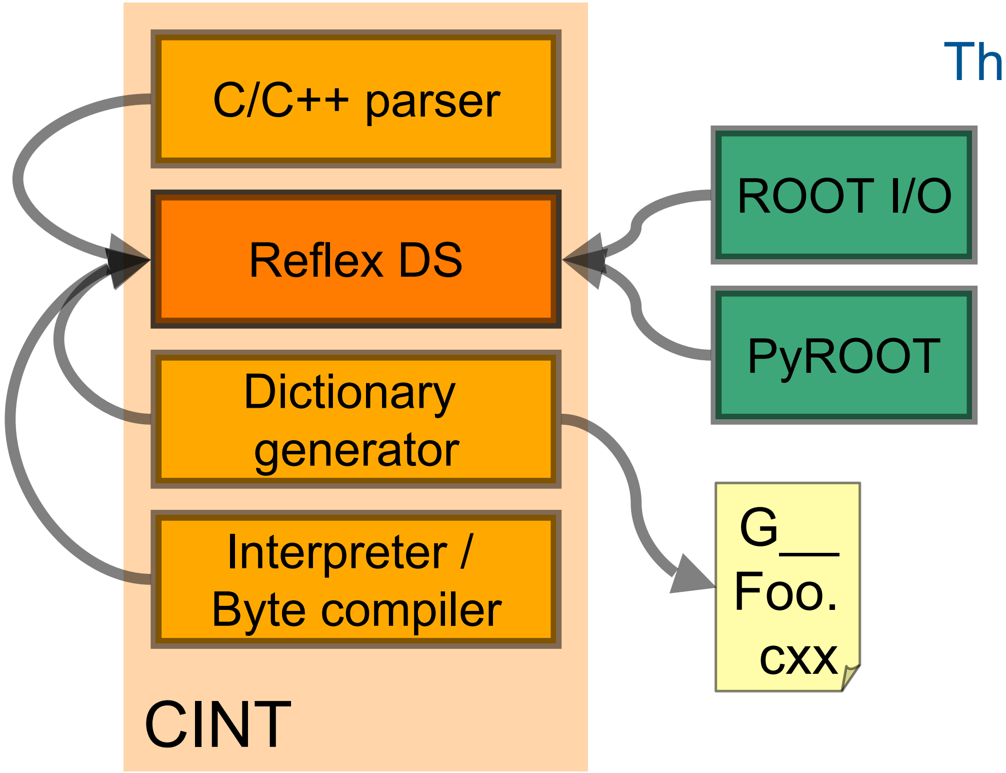


The work is done in 3 steps:

1. Generate Reflex dictionary source code via rootcint
2. Integrate Reflex data structures into CINT

Reflex in ROOT

- CINT is the interpreter/dictionary system in ROOT
  - We change CINT to use Reflex

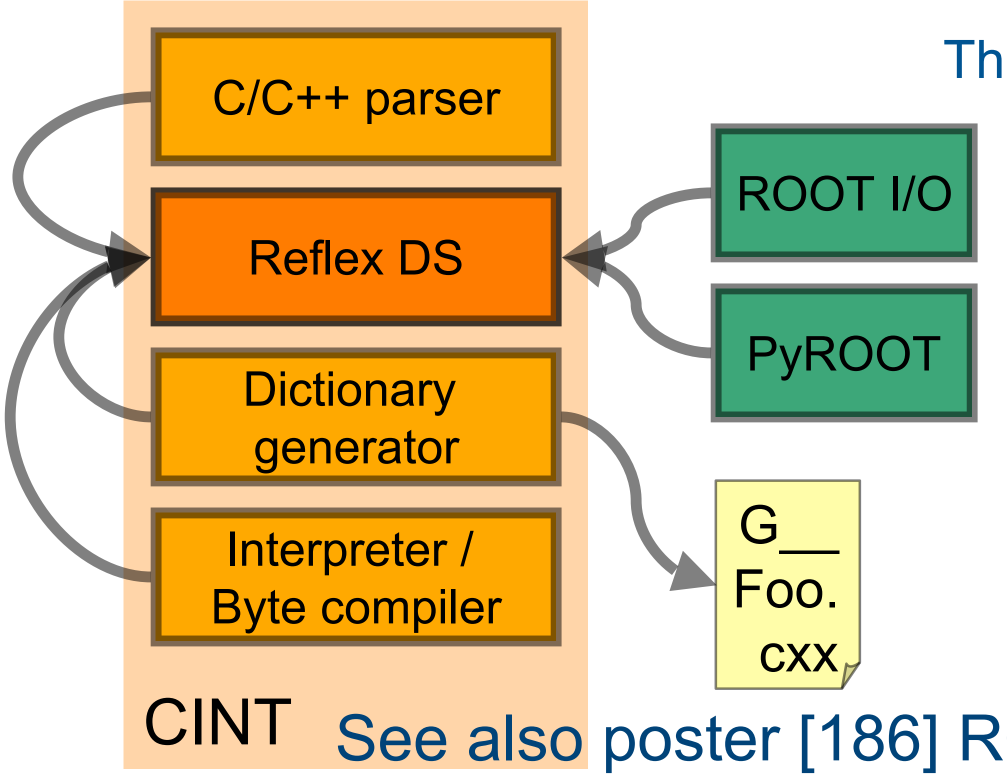


The work is done in 3 steps:

1. Generate Reflex dictionary source code via rootcint
2. Integrate Reflex data structures into CINT
3. Adapt other ROOT dictionary users (e.g. ROOT I/O) to Reflex

Reflex in ROOT

- CINT is the interpreter/dictionary system in ROOT
  - We change CINT to use Reflex



The work is done in 3 steps:

1. Generate Reflex dictionary source code via rootcint
2. Integrate Reflex data structures into CINT
3. Adapt other ROOT dictionary users (e.g. ROOT I/O) to Reflex

DONE

See also poster [186] ROOT/CINT/Reflex integration

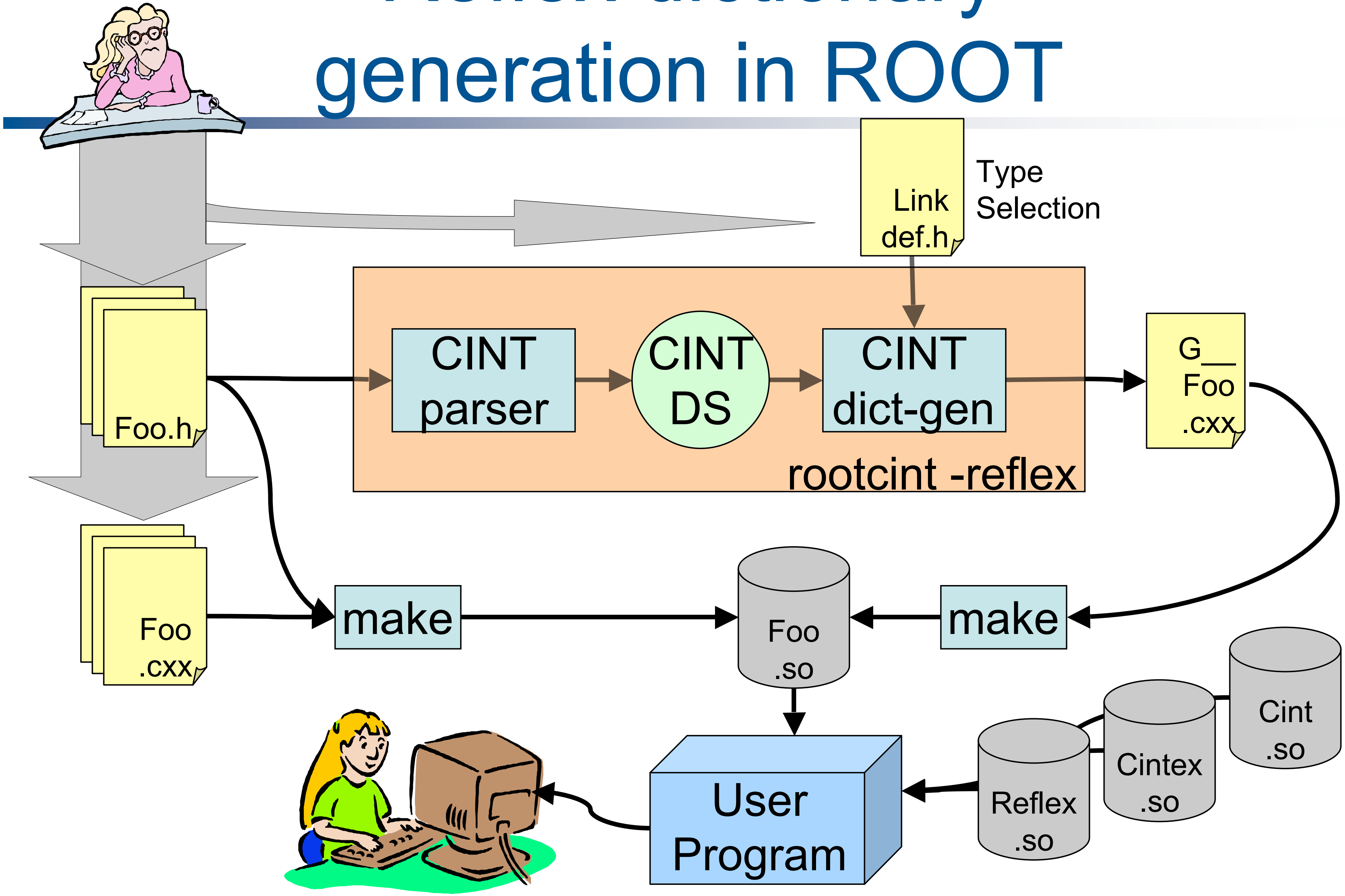
Cintex

- Is a ROOT module
- Provides a gateway from Reflex to CINT
  - Loads Reflex dictionary information into CINT
- Used for
  - Persistence of objects (eg. POOL)
  - Interactive usage (eg. CINT, PyROOT)

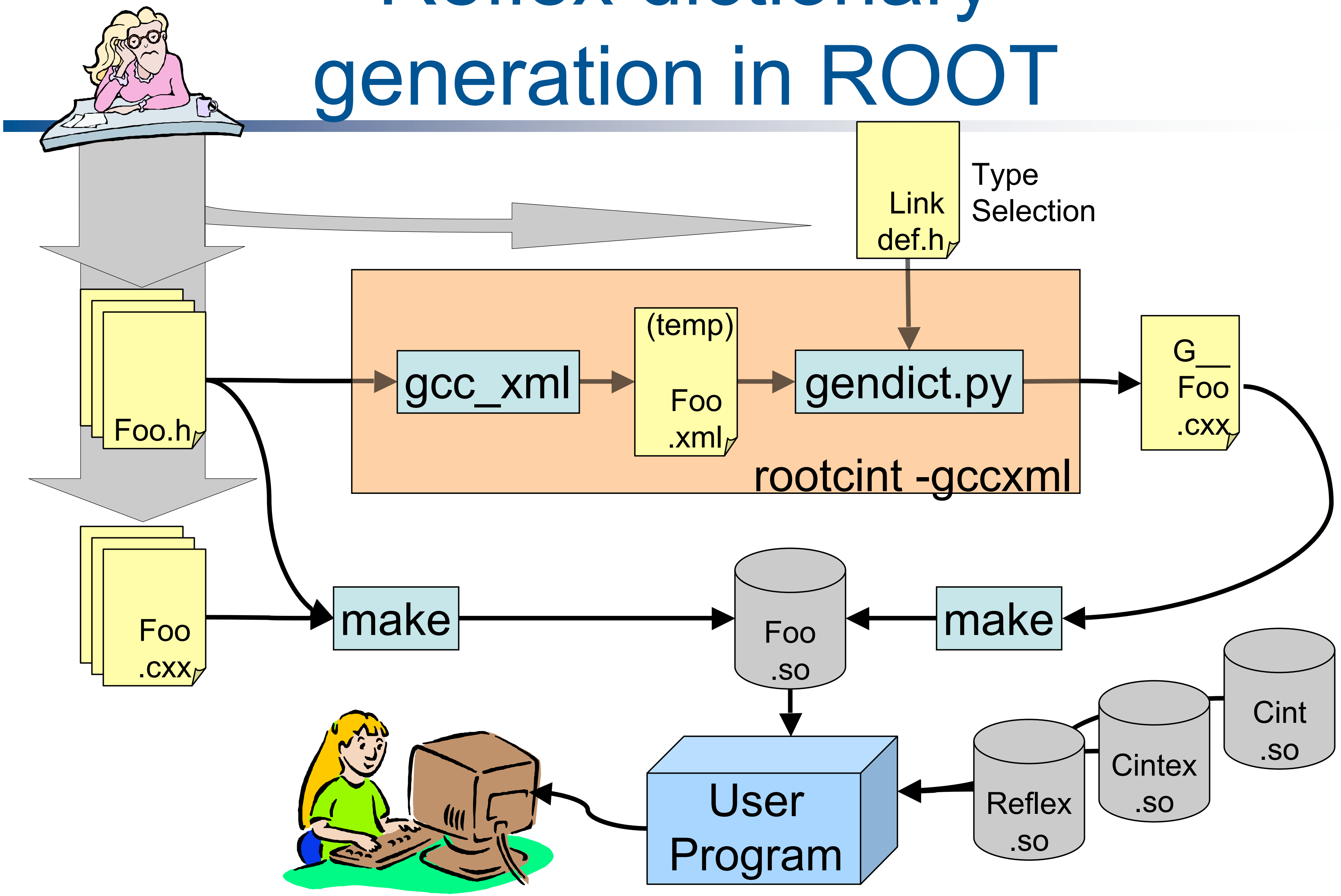
```
root [0] gSystem->Load("libCintex.so")
root [1] Cintex::Enable()
root [2] gSystem->Load(“libCLHEPRflx.so”)
root [3] CLHEP::HepLorentzVector v(1,2,3)
root [4] v.x()
(double)1.000000000000000000e+00
```

```
>>> from PyCintex import *
>>> loadDictionary(‘libCLHEPRflx.so’)
>>> v=ROOT.CLHEP.HepLorentzVector(1,2,3)
>>> v.x()
1.0
```

Reflex dictionary generation in ROOT



Reflex dictionary generation in ROOT



How to generate a Reflex dictionary

- Selection of types via Linkef.h
  - Using CINT as parser

```
rootcint -reflex TFoo.h Linkdef.h
```

  - Using gcc (gcc_xml) as parser

```
rootcint -gccxml TFoo.h Linkdef.h
```

- Selection of types via selection.xml
  - Using gcc (gcc_xml) as parser

```
genreflex Foo.h -s selection.xml
```

Summary

- Reflex is ready to use
  - as standalone package
  - or as part of a ROOT release
- Reflex in the context of ROOT
  - Usage of Reflex dictionaries works through Cintex
  - Step 1: Dictionary generation finished
  - Step 2 and 3:
    - Integration of Reflex data structures into CINT
    - Adaptation of other clients (eg. ROOT I/O)

    are ahead of us

Pointers

- Reflex
  - http://cern.ch/reflex
  - (Reflex as standalone package)
- ROOT
  - http://root.cern.ch
  - (Reflex as ROOT module)
- GCC_XML
  - http://www.gccxml.org
  - http://cern.ch/service-spi/external/distribution