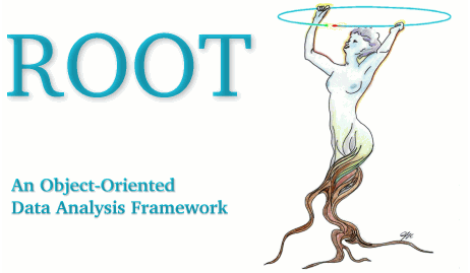
Reflex
&
new rootcint

Stefan Roiser

CERN / PH / SFT



Content

- Reflection in general
- Reflex introduction
  - Design
  - API
  - Code Examples
- Reflex Integration with ROOT/CINT
- Generation of Dictionaries
- Status
- Outlook

Definitions

- Reflection is the ability of a language to introspect it's own structure at runtime and interact with it in a generic way
- A dictionary provides reflection information about types of a certain language to the user

What is Reflection?

Without prior knowledge about objects

Dictionary

```
ClassBuilder("Particle").
 AddDataMember("m_mass").
 AddFunctionMember("mass");
```

```
class Particle {
 public:  double mass();
 private: double m_mass; };
```

```
Type t =
 Type::ByName("Particle");

Object o = t.Construct();

std::cout <<
  object_cast<double>
  (o.Invoke("mass"));

o.Destruct();
```

Reflection and C++

- Very limited support by the language (RTTI)
  - Other languages do (e.g. Java, Python,...)
- CINT
  - Supports reflection of C++ constructs
- Several other approaches exist, but ...
  - Tied to a system or
  - Instrumentation of code needed
- Stroustrup proposed XTI (eXtended Type Information)
  - Dormant project

Goals

- Enhance C++ with reflection capabilities
  - non intrusive, automated
- Close to the C++ ISO 14882 standard
  - Extra information stored in “properties”
- Light and standalone system
- Small memory footprint
- Multi platform
  - Linux, Windows, Mac OSX, ...

3 Levels of Reflection

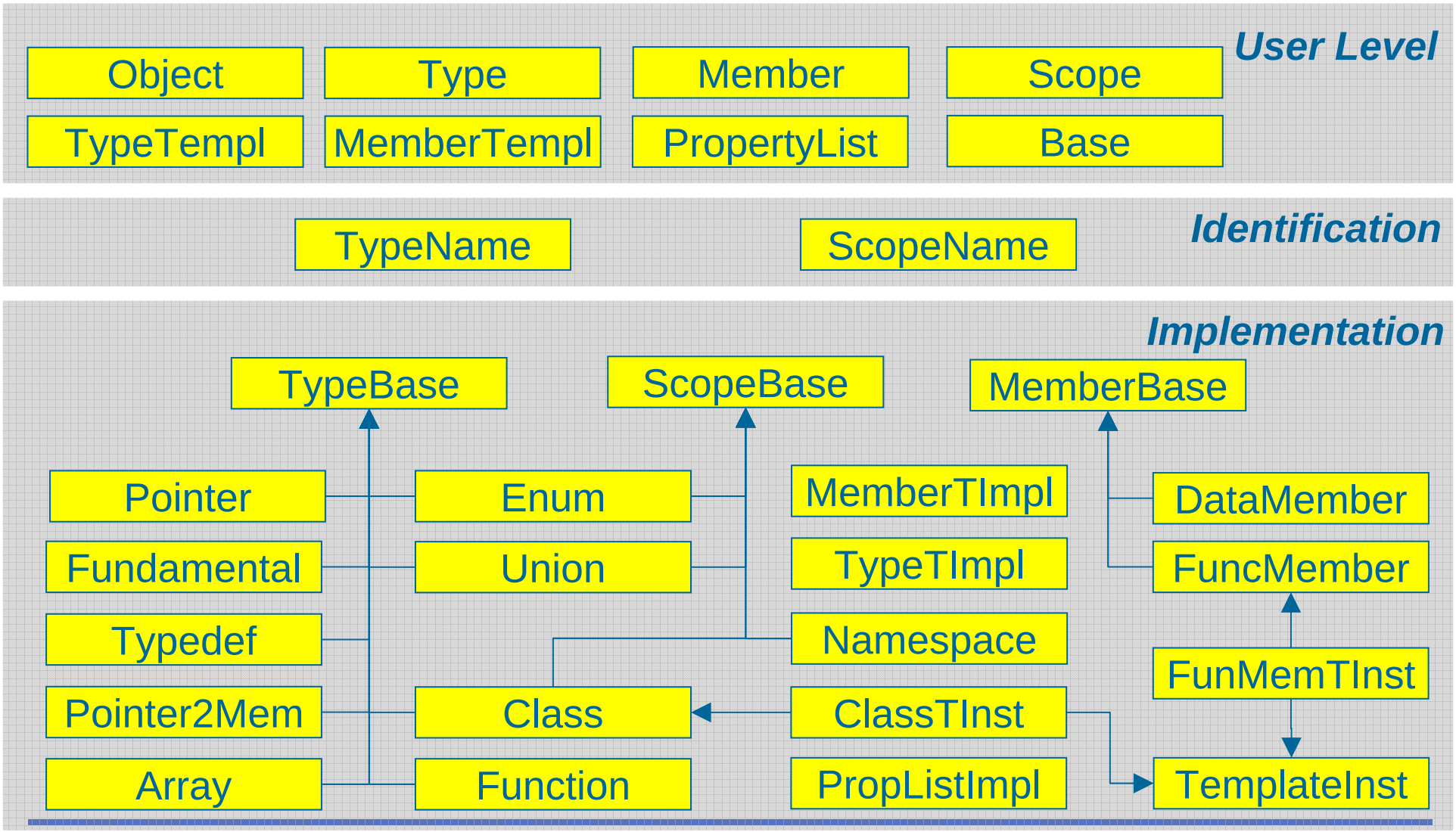
Reflex supports:

- 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

- API
  - Handful (i.e. 8 ☺) classes the user has to deal with
  - Full functionality, mainly through forward functions
- Identification
  - A unique immutable id of each type/scope (no delete)
- Implementation
  - Provides the content of the reflection information

Reflection Model



User Classes (general)

- Small memory allocation
  - ~ sizeof (Pointer) + sizeof(int)
  - By value semantics
- operator bool()
  - Will return true if the Type, Scope, etc. is declared and implemented
- Function “std::string Name(unsigned)”
  - Returns the “SCOPED | FINAL | QUALIFIED” name
- Many “Is*” functions to ask questions
  - e.g. IsVolatile, IsClass, etc.

User Classes

- Type
  - Pointer, Pointer to Member, Typedef, Array, Function, Union, Enum, Fundamental, Class, Instantiated Template Class
  - Contains CV qualification (plus Reference)
  - Function return/parameter types, array length, typedef types
- Scope
  - Namespace, Class, Union, Enum
  - Walk up/down hierarchy, access type/member information
- Member
  - Function Member, Data Member
  - Invoke function members, get/set data members

User Classes (ctd.)

- Object
  - Represents a C++ class instance (address + type)
  - Invoke functions, get/set members, dynamic type
- PropertyList
  - Attach collection of key/value pairs to Types, Scopes, Members
  - Key is string, Value is “Any” object
- Base
  - Base type, access, offset
- TypeTemplate / MemberTemplate
  - Template instances, template parameters

Example: Introspection

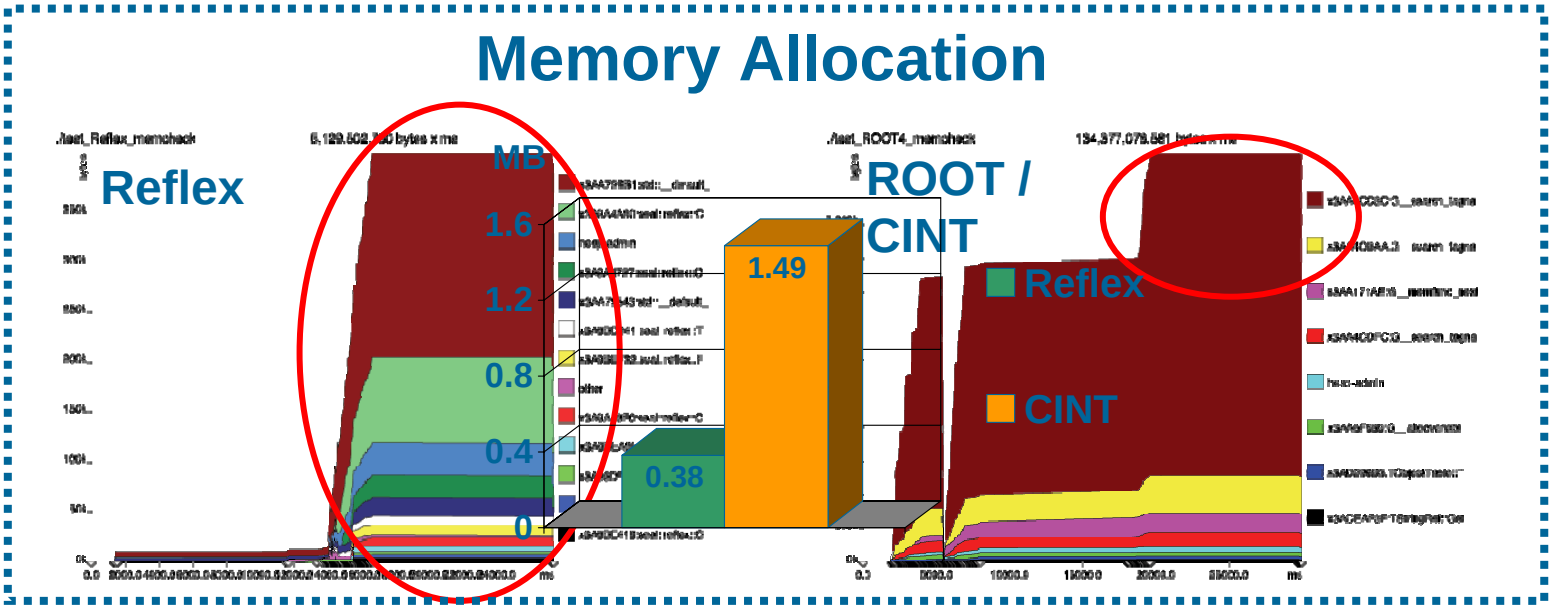

```
// The Reflex namespace inside Root
using namespace ROOT::Reflex;

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

// If class print all data members
if ( cl.IsClass() ) {
  for ( size_t d = 0; d < cl.DataMemberCount(); d++ ) {
    Member dm = cl.DataMemberNth(d);
    cout << dm.Type().Name(SCOPED) << " " << dm.Name() <<";";
    // output comment line if exists
    if ( dm.PropertyList().HasKey("comment") ) {
      cout << m.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("function");

// Alternatively
for ( size_t f = 0; f < cl.FunctionMemberCount(); f++ ) {
  if (cl.FunctionMemberNth(f).Name() == "function") {
    ret = cl.FunctionMemberNth(f).Invoke(obj);
  }
}

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

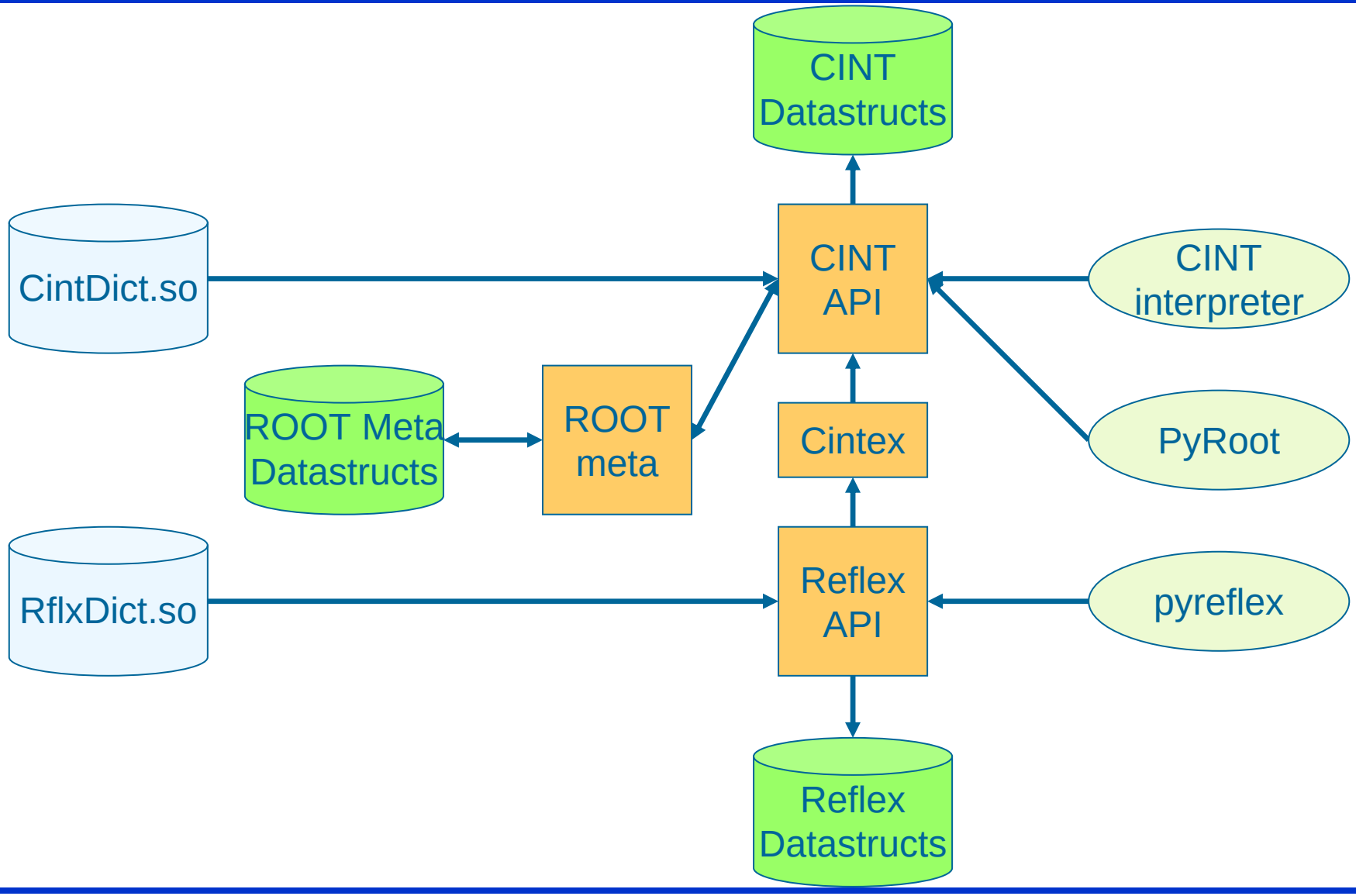
Why using Reflex?

- Advantages
  - 100 % C++ code
  - Better C++ coverage
  - Smaller memory footprint

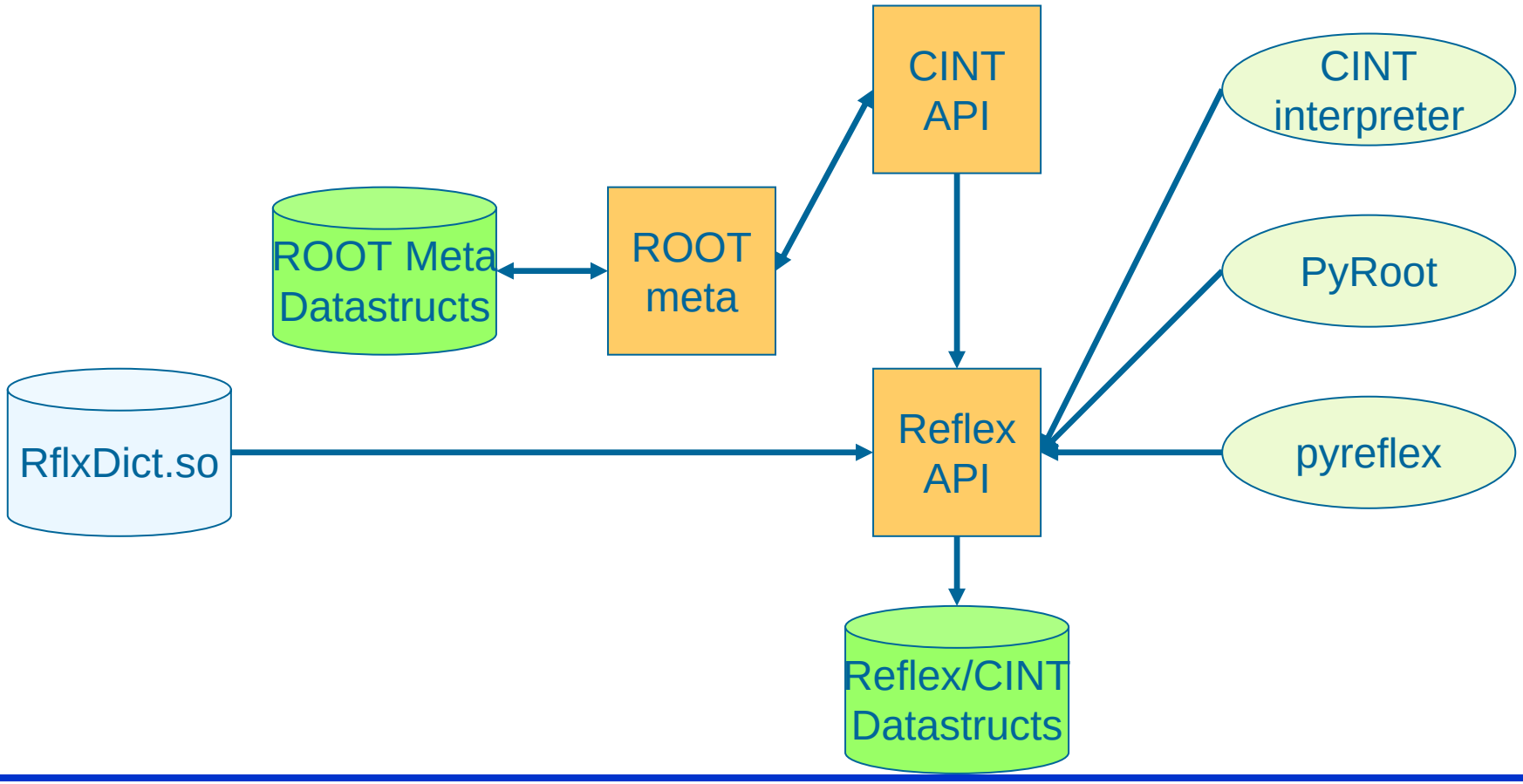


-> Mini workshop 2005 @ CERN - Decision to use Reflex

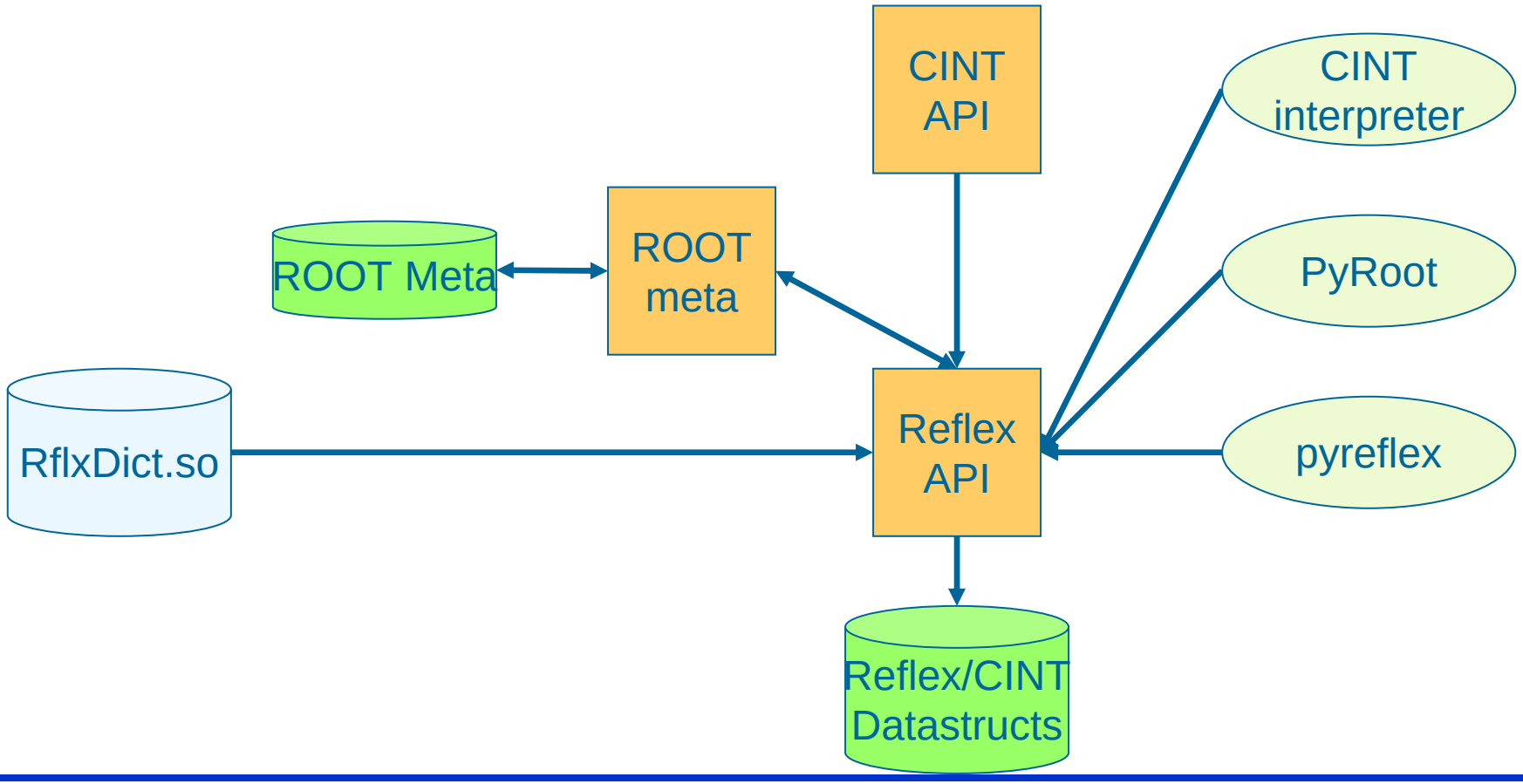
Integration with ROOT



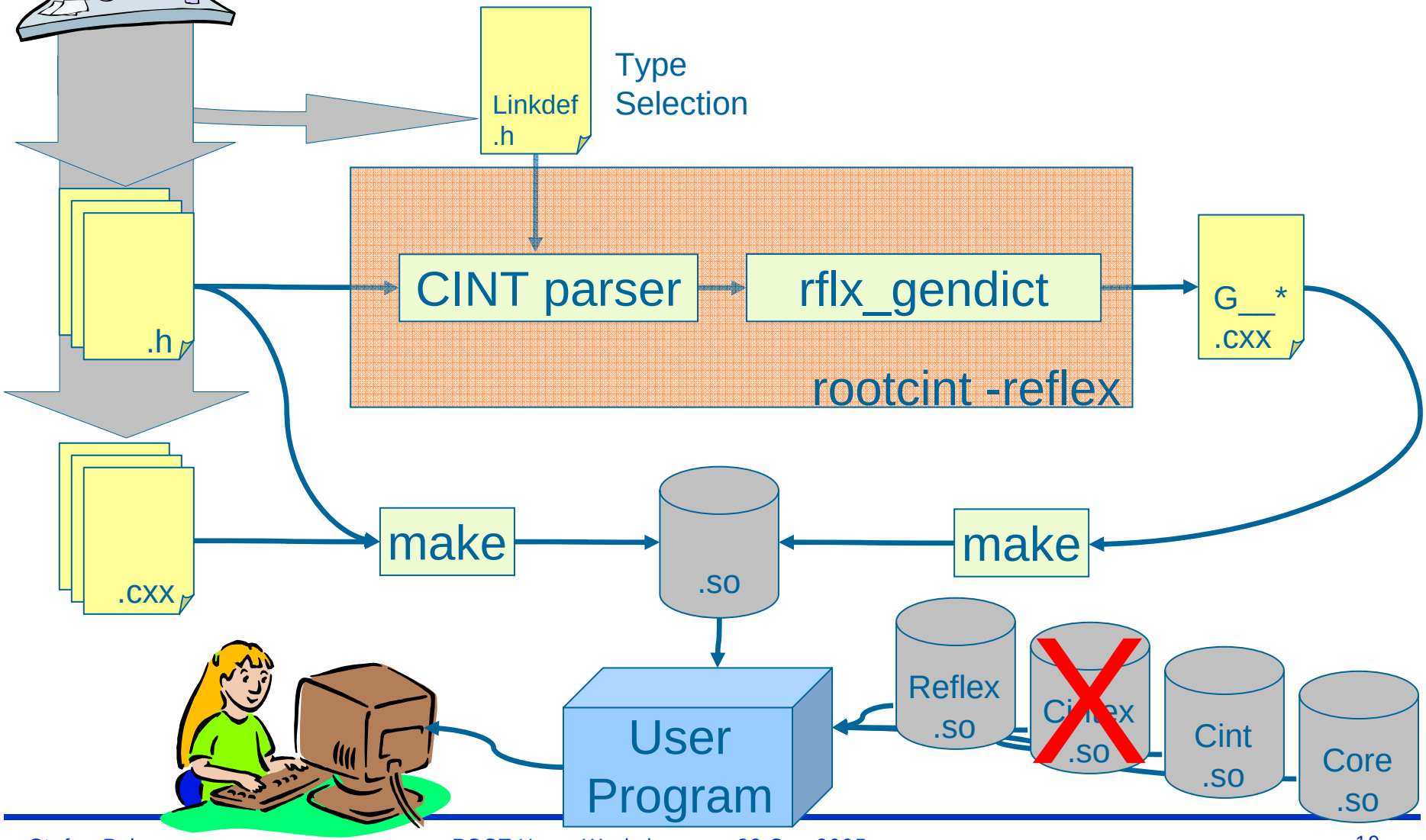
Integration with ROOT



Integration with ROOT



Dictionary Generation with CINT



Dictionary Generation with gccxml

Linkdef .h

selection .xml

Type Selection

.h

gcc_xml

(temp) .xml

gendict.py

rootcint -gccxml

G__* .cxx

.cxx

make

.so

make

User Program

Reflex .so

Cintex .so

Cint .so

Core .so

selection.xml file

- Functionality
  - Allows inclusion/exclusion of types
  - Usage of patterns
  - Apply special information (transient, class ID)
  - Non-intrusive way of attaching information
- Simple adaptation to user requests

```
<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 compilable program can be used as input
- “rootcint -gccxml” uses the temporary XML file to produce dictionary C++ source files

Parser commands

Current Status

```
[] ~ > rootcint -f dict.cxx -c …              (Cint dictionaries)
```

Transition phase

```
[] ~ > rootcint -f dict.cxx -c …              (Cint dictionaries)
[] ~ > rootcint -f dict.cxx -c -reflex …      (Reflex dict wth Cint)
[] ~ > rootcint -f dict.cxx -c -gccxml …      (Reflex dict wth gccxml)
```

Final status

```
[] ~ > rootcint -f dict.cxx -c …              (Reflex dict wth Cint)
[] ~ > rootcint -f dict.cxx -c -gccxml …      (Reflex dict wth gccxml)
```

Status

- Reflex
  - Ready to use (available through ROOT cvs)
  - For the time being optional in ROOT (--enable-reflex)
- Cintex
  - Ready to use (available through ROOT cvs)
  - For the time being optional in ROOT (--enable-cintex)
- rootcint -reflex
  - rootcint integration missing (trivial)
  - makecint produces reflex dictionaries (~ 80 % of C++ standard)
- rootcint -reflex -gccxml
  - rootcint integration missing (trivial)
  - Works with selection.xml file
  - Understand Linkdef.h syntax

Outlook

- Reflex/Cintex
  - New versions will be imported in ROOT cvs
  - Selection classes for Reflex (allows type selection from within C++)
- rootcint -reflex
  - October release: most functionality will be available
  - December release: ready
- rootcint -reflex -gccxml
  - October release: gendict.py understanding Linkdef.h syntax
  - December release: ready
- Reflex/Cint datastructures merge
  - See Philippe's talk

Pointers

- Seal Dictionary page
  - http://seal.web.cern.ch/seal/snapshot/work-packages/dictionary/index.html
- Reflex page (work in progress ...)
  - http://cern.ch/seal-reflex
- GCC_XML
  - http://www.gccxml.org