Cint version 6

2 May 2005 @ CERN

Masaharu Goto

Agenda

- Version 5 Issues
- Simplification of operation
- Version 5 / 6 schematics
- Execution flow
- Status

Version 5 issues

- Scope problem
  - Block scope variables behave differently
- Loop bugs
  - Due to complicated loop compilation mechanism
- Bytecode limitation
  - Eventually, macro runs much much slower
- Maintenance
  - Badly organized source code. Hard to fix bugs

Simplification of operation

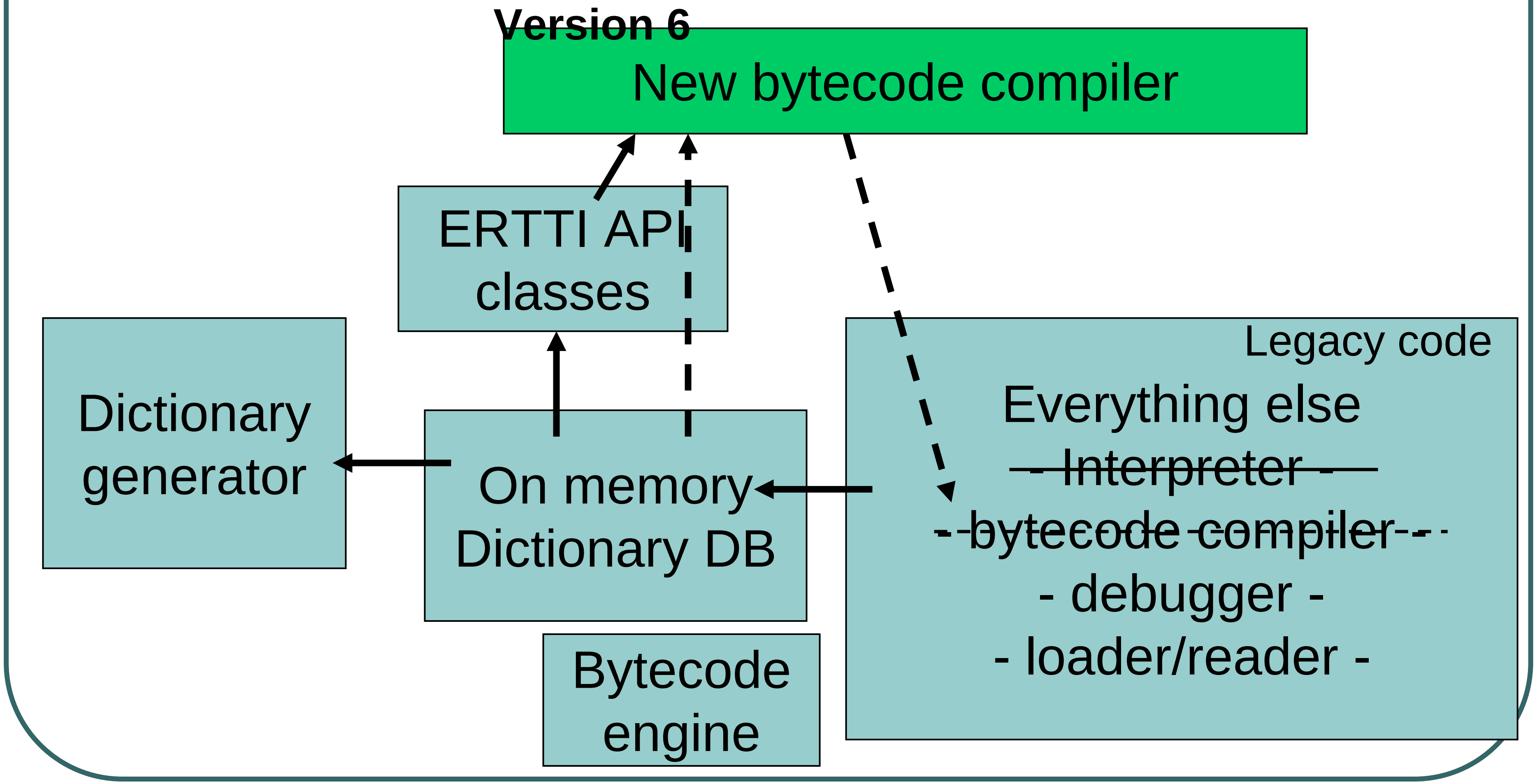
- Version 5
  - On the fly interpretation
  - Loop compilation
  - Function compilation
  - Native execution

→

- Version 6
  - Function compilation
  - Native execution

Reduced complexity of execution system

Version 5 / 6 Schematics



Version 6

New bytecode compiler

ERTTI API classes

Dictionary generator

On memory Dictionary DB

Bytecode engine

Legacy code

Everything else

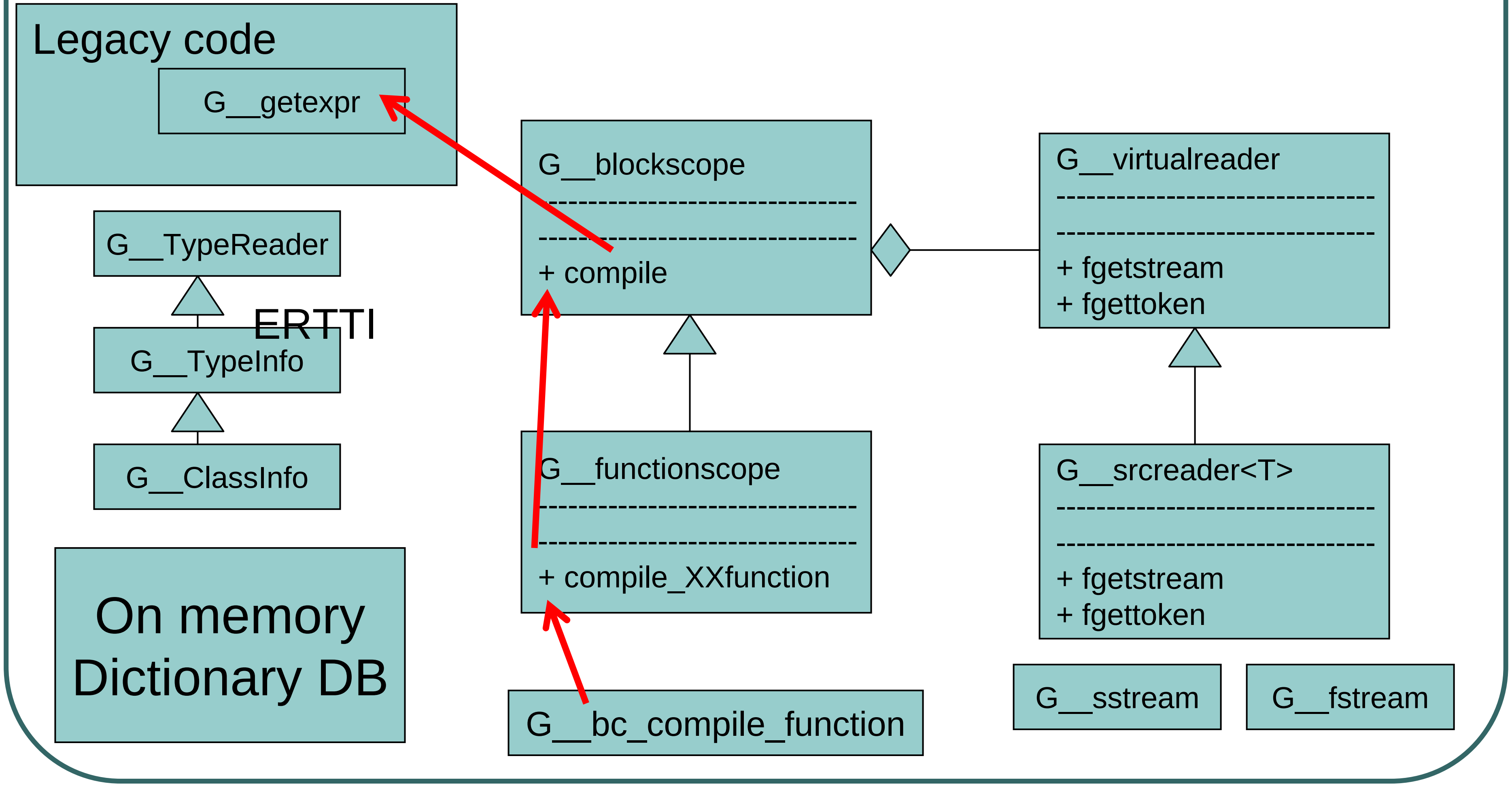
~~- Interpreter -~~

~~- - bytecode compiler - -~~

- debugger -

- loader/reader -

Simplified version 6 class diagram



Version 6 execution flow

- Pre-run
  - Read source file
  - generate on memory dictionary
  - generate virtual table
- At execution, for every function call
  - bytecode compilation G__bc_compile_function()
  - execution G__exec_bytecode()

Simple example

```
#include <cstdio>
using namespace std;

void f(int a) {
   printf(“a=%d\n”,a);
}

int main() {
   printf(“start\n”);
   f(1234);
   return(0);
}
```

1: Compile “main”

 f() is resolved but not compiled yet

2: Run “main”

 When it comes to run f()

  1: Compile “f”

  2: Run “f”

How to use version 6

- When you compile Cint
  - Define G__CINT_VER6 in SYSMACRO
  - Add CINT_V6 source files bc_XXX.o

- When you run Cint
  - Use -@ command line option

    (without -@, Cint behaves as version 5)

Status : Sept 2005

- Re-engineering started in Apr 2004
- Simple script begin to run in Aug 2004
  - Scope issue is cleared
- Gone through most of the cint/test test-suite
- Work to be done
  - Implement missing features
  - Go through test-suite
- Challenges
  - Virtual base class and other complicated C++ features

Cint & Reflex

Issues and Plans

What we want to do

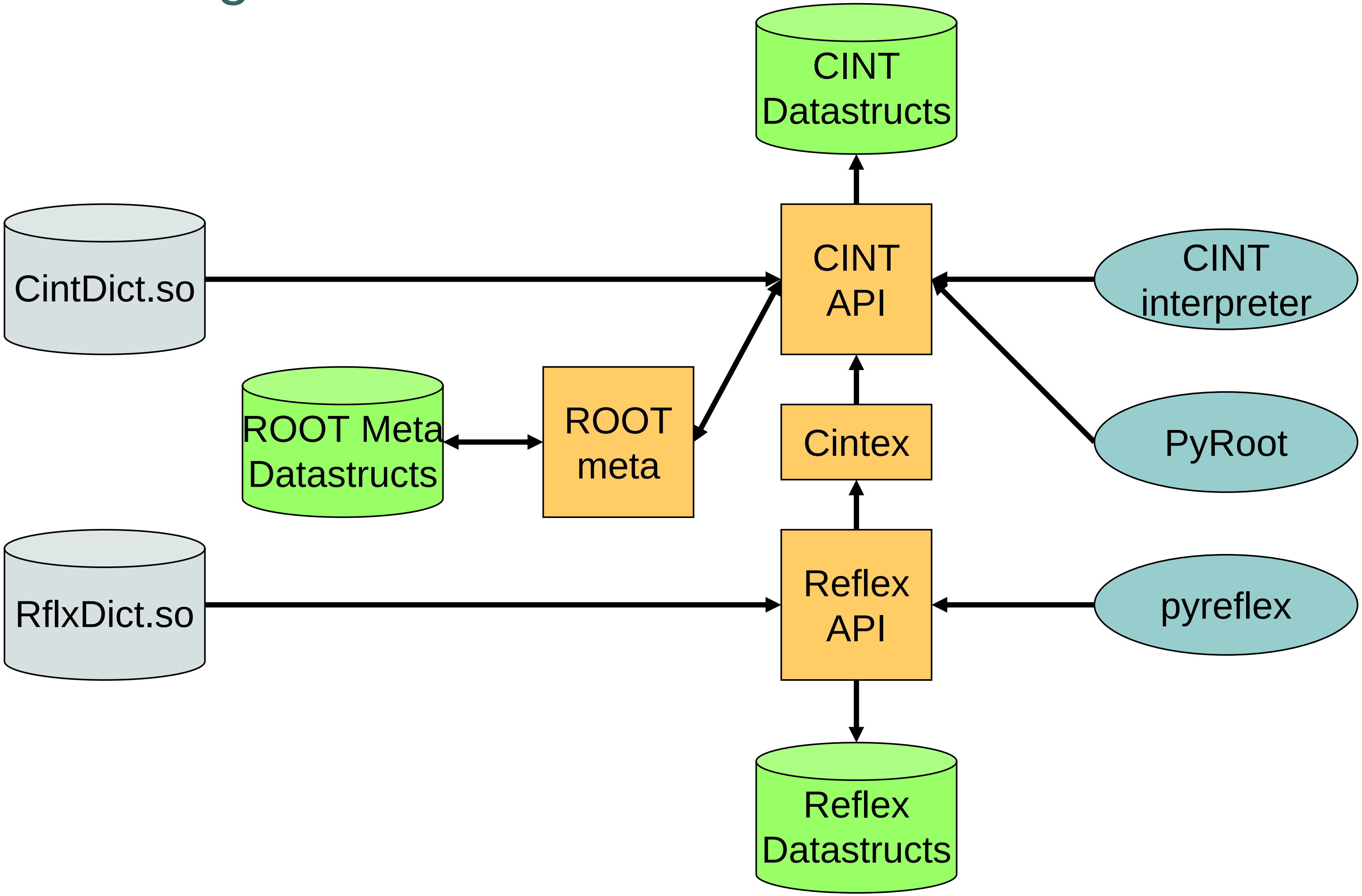
- Modernize the data structures (G__struct replacement)
- Offer the optional ability to use gcc_xml for parsing the header files
- Unify C++ dictionaries for LHC experiments

Requirements

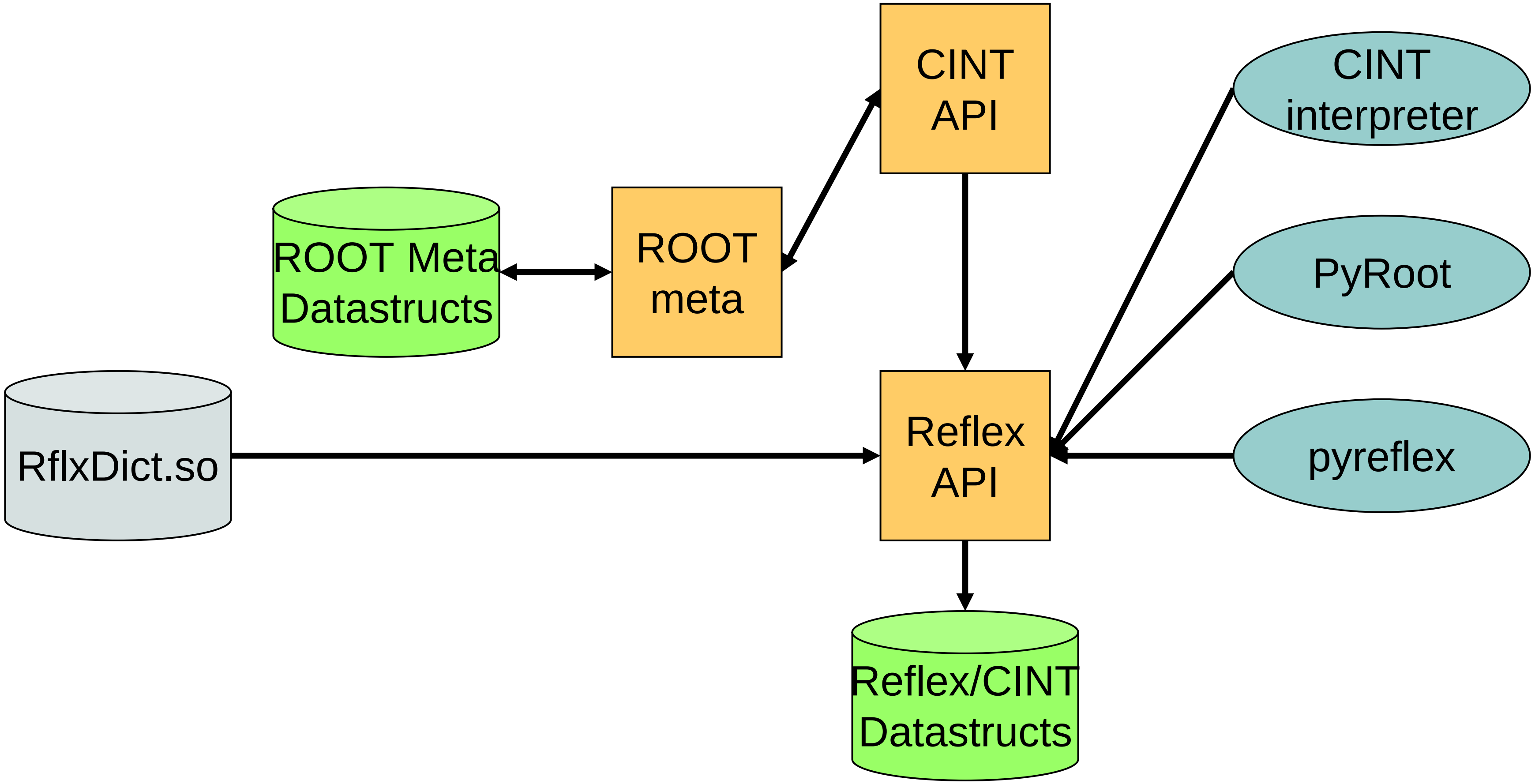
- No loss in functionality
- User level backward compatibility
  - including dictionary generation steps
- Support for the current platforms

- Avoid (as much as practical) code duplication in particular we would like to avoid having the whole CINT code having to support 2 dictionaries

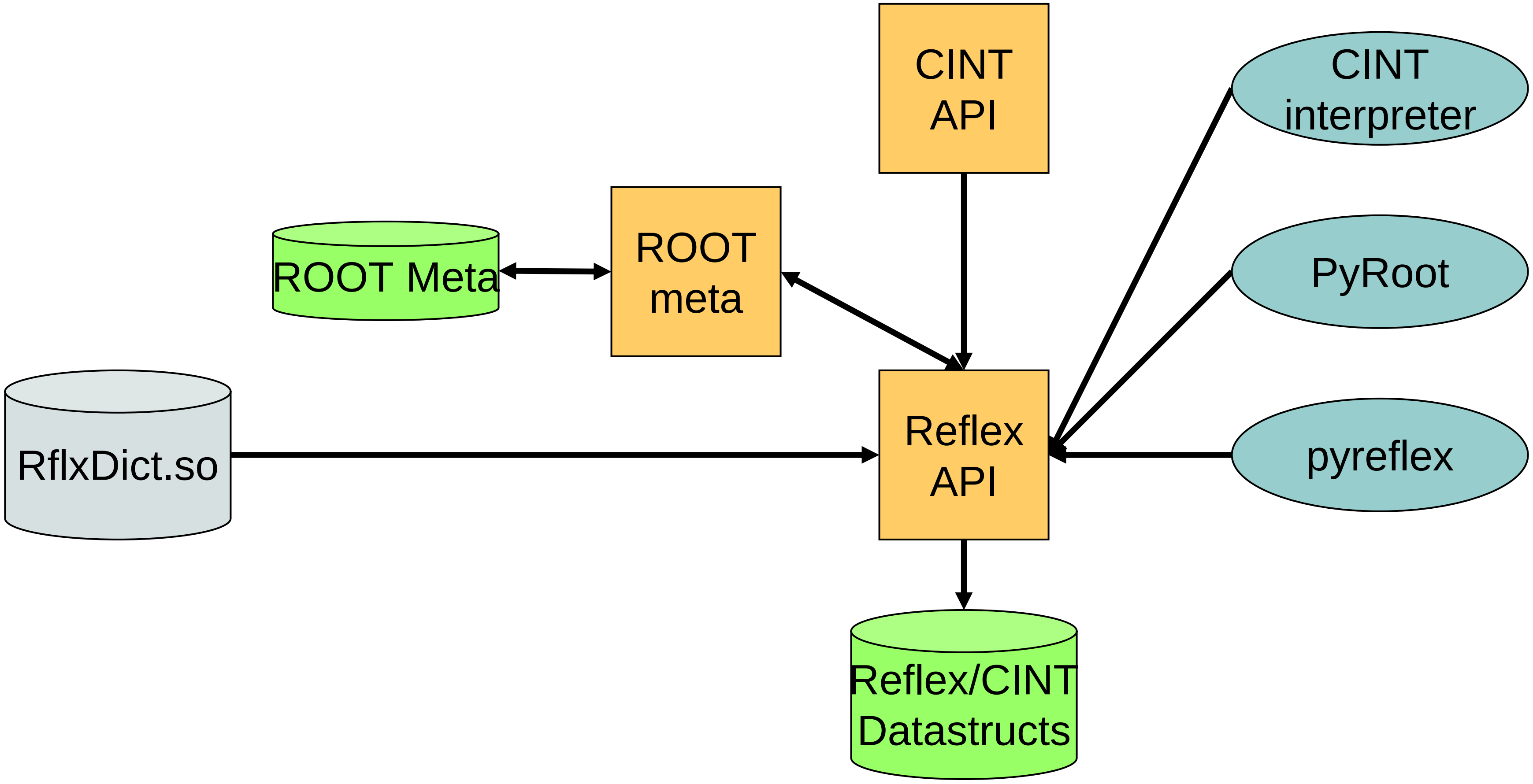
Integration with ROOT



Integration with ROOT



Integration with ROOT



Cint/Reflex Workshop

- Backward compatibility
  - Of course, but how much?
- Distribution/Coding Issues
  - New code is in C++ but existing CINT code was in C
  - Python dependency (not fundamental)
  - Optional gcc xml dependency
    - portability and ease of build
    - Coordination with non gcc compilers
  - Distribution
    - how does Masa access/use the Reflex code
  - CINT Code development
    - To CVS or not to CVS?

Cint/Reflex Workshop in May

- Input for the Dictionary Generation
  - LCGdict uses an XML files as input
  - makecint/rootcint uses #pragma as input
  - For backward compatibility we should support both.

CINT/Reflex Structures

G__tagtable	*Reflex::Type*
G__var_array	*vector<Reflex::Member>*
G__ifunc_table	*vector<Reflex::Member>*
G__inheritance	vector<Reflex::Base>
G__typedef	Reflex::Type
G__...template…	Reflex::….Template…

Transition Path

- Both CINT and Reflex can refer to any class using an '*int*'. A translation table can be kept to be able to switch back and forth between the two.
- Reflex is able to fill (***most***) of the CINT in-memory structure using Cintex
- From then on, we need to

a) insure that Reflex is complete by migrating rootcint/makecint and doing extensive test

b) from then on we would know that the Reflex data and the CINT data are exact duplicate

c) Starting moving code little by little from using the current CINT structure to access the Reflex structure (since we know the data to be the same this should only be a coding issue).

[This includes both reading and writing into the dictionary]

Proposed plan

1. *Move to CVS code dvpt environment*
2. *Incorporate* gcc_xml, Reflex and Cintex in CINT and ROOT build system

3. Start compiling the existing CINT code in C++, declaring the existing C public API as ‘extern C’ (for full backward compatibility)
4. a) Provide an equivalent to makecint generating reflect dictionary
5. b) Provide an equivalent to makecint using lcg_dict

[At this point we can check that Reflex cover all the CINT data structure]

6. Replace access to data members of G__struct to calls to the Reflex equivalent
7. Repeat 6 for all data members
8. Remove G__struct
9. Repeat 6/8 with the various CINT C structures.

Main advantages of this plan is that after each step we always have a fully functioning CINT. Albeit slower and bigger until we remove all duplications

10. Integrate Into ROOT

Done so far

- Migrated CINT source code to CVS
- Migrated CINT source code to C++
- Kept the extern C interface
  - i.e. the CINT library is binary backward-compatible
- Wrote a version of rootcint issuing Reflex dictionary

Next steps

- For the October release
  - Releasing the rootcint option –reflex
- For the December release
  - Add option to rootcint to use gcc_xml as the parser (when available)
- Adapting the CINT source code to access the Reflex in-memory database
  - Expected completion by the end of April 2006