ROOT Tutorials – Session 8

Dictionary Generation, **`rootcint`**,
Simple I/O, Hands-on

Fons Rademakers

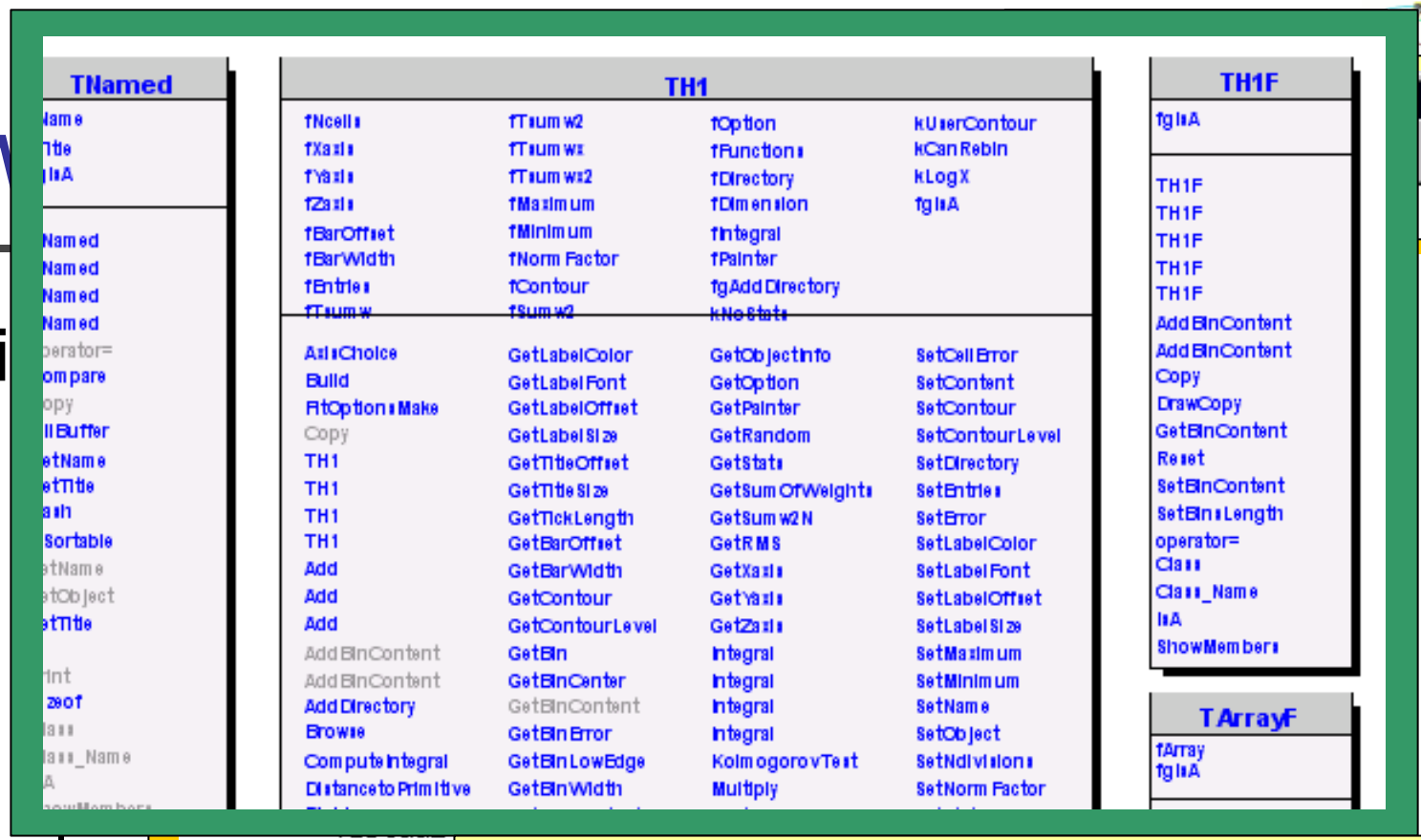
Introduction

- The following items will be covered:
  - Writing a small MyClass
  - Generating a dictionary for MyClass
  - Execute MyClass methods in interpreter
  - Save and restore MyClass objects

Dictionary Generation

W

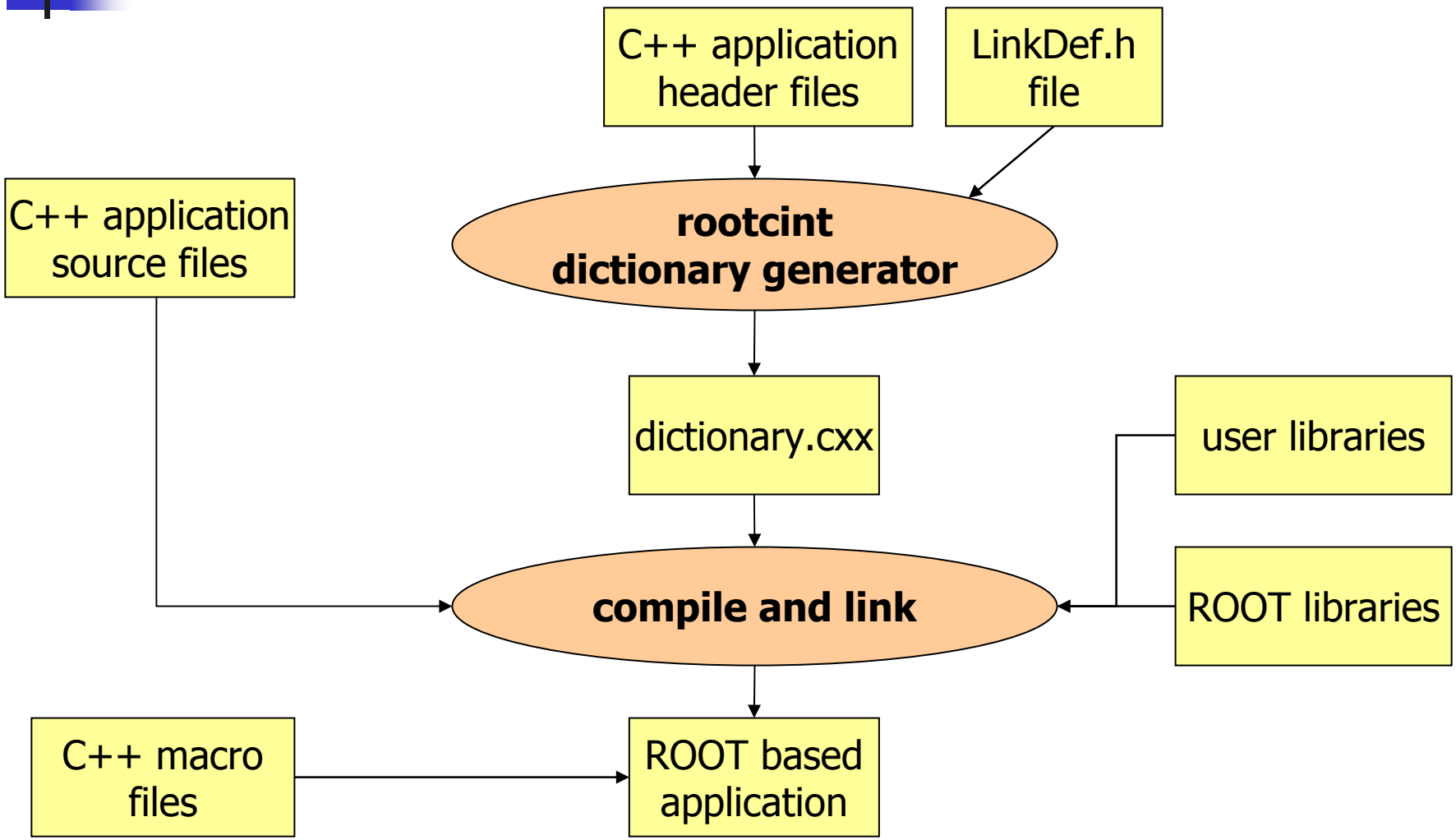
- Di
  - class
  - Gene
  - Prov




```
   virtual
   virtual
 static TCl
   virtual
   virtual
 virtual St
virtual TCl
   virtual void Reset(Option_t* option)
   virtual void SetBinContent(Int_t bin, Stat_t content)
   virtual void SetBinsLength(Int_t nx)
   virtual void ShowMembers(TMemberInspector& insp, char* parent)
   virtual void Streamer(TBuffer& b)
```

TClass *TObject::IsA()
Bool_t TObject::InheritsFrom()
const char *TObject::ClassName()

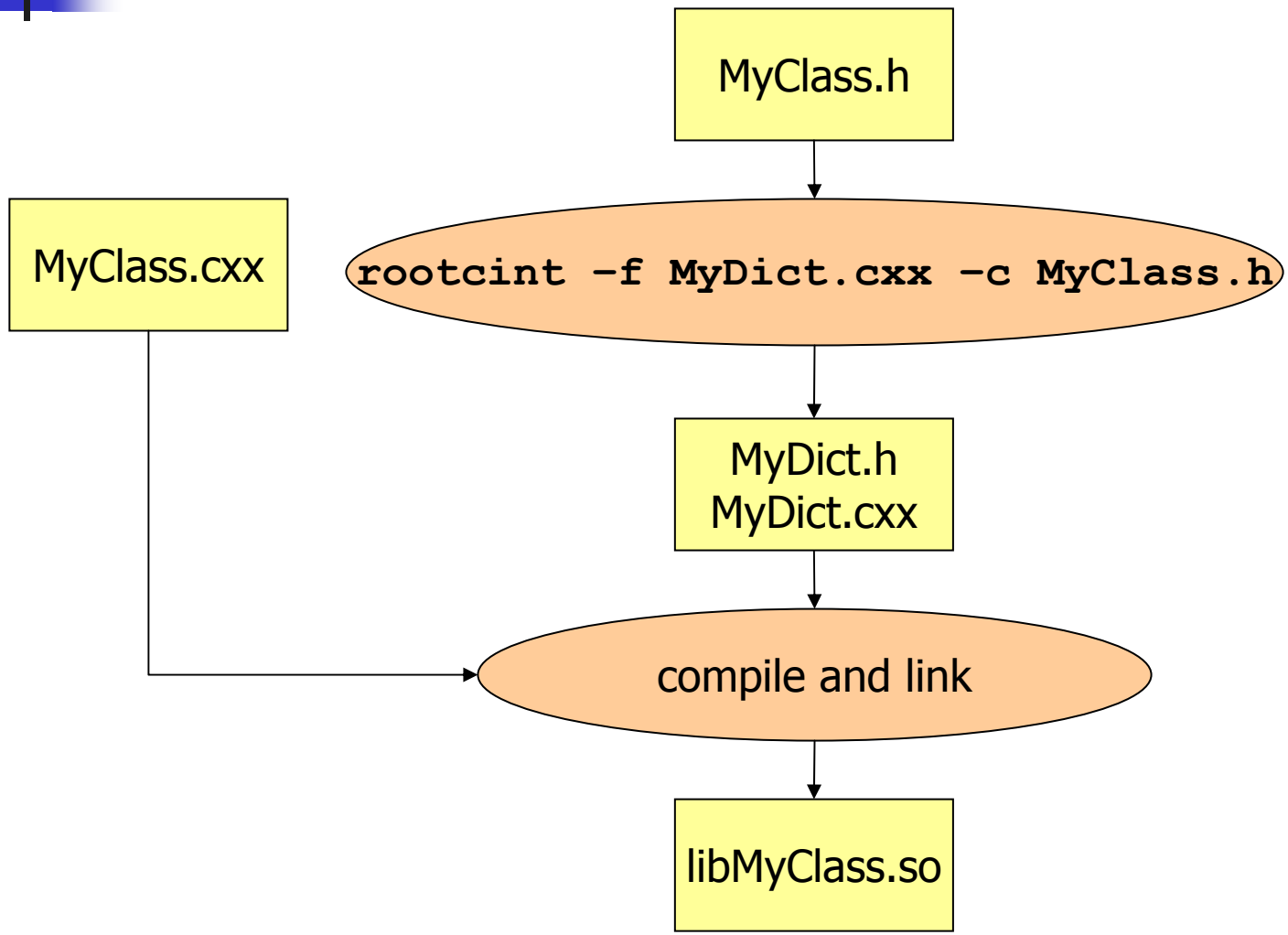
ROOT Environment and Tools



Providing Interactive Access

- To provide interactive access to your classes from CINT you only need to generate a dictionary for your classes
  - No other instrumentation is needed

Generating a Dictionary



Exercise 1

- Write simple class, call it MyClass and store it in MyClass.h and MyClass.cxx
- Generate dictionary:

```
rootcint -f MyDict.cxx -c MyClass.h
```

- Compile MyClass.cxx and mydict.cxx into a shared library:

```
g++ -shared -fPIC `root-config --cflags` -o
libMyClass.so MyClass.cxx MyDict.cxx
```

Exercise 1 (cont.)

- Start ROOT, load libMyClass.so:

```
$ root
root [0] gSystem->Load("libMyClass")
```

- Create and play with MyClass object:

```
root [1] MyClass a
root [2] a.Print()
root [3] a.P<TAB>
```

Linking Class to ROOT RTTI

- To link a class to the ROOT RTTI system two macros need to be added to MyClass:
  - ClassDef(*class name*, *version id*)

```
// MyClass.h
class MyClass {
public:
   . . .
   ClassDef(MyClass,1)  // analyse my data
};
```

  - ClassImp(*class name*)

```
// MyClass.cxx
#include "MyClass.h"
ClassImp(MyClass)
```

ClassDef and ClassImp

- These macros provide:
  - Several static methods to obtain reflection/meta class information

```
static TClass *Class();
static const char *Class_Name();
virtual TClass *IsA() const;
```

  - Inspection and Streamer (I/O) methods

```
virtual void ShowMembers(TMemberInspector &insp, char *parent);
static Version_t Class_Version() { return id; }
virtual void Streamer(TBuffer &b);
friend TBuffer &operator>>(TBuffer &buf, name *&obj);
```

  - Methods returning header and source file name
  - Small class and static object to register class to ROOT when library is loaded

Exercise 2

- Add ClassDef() and ClassImp() macros to MyClass
  - And also add: **`#include <Rtypes.h>`** to MyClass.h
- Repeat same steps as in exercise 1:

```
rootcint -f MyDict.cxx -c MyClass.h
```

```
g++ -shared -fPIC `root-config --cflags` -o
libMyClass.so MyClass.cxx MyDict.cxx
```

```
$ root
root [0] gSystem->Load("libMyClass")
```

```
root [1] MyClass a
root [2] a.Print()
root [3] TClass *c = a.IsA()
```

Full Integration into ROOT

- To make your class a fully ROOT supported class, with full I/O capabilities, just add derivation from TObject to MyClass

```
// MyClass.h
class MyClass : public TObject {
public:
   . . .
   ClassDef(MyClass,1)  // analyse my data
};
```

Exercise 3

- Add public derivation from TObject to your class
  - Replace the Rtypes.h include by TObject.h
- Repeat same two build steps as in the previous exercises
- Create object of MyClass and Dump() its run time contents:

```
root [1] MyClass a
root [2] a.Dump()
Root [3] …
```

Exercise 3 (cont.)

- Open a file and write the object:

```
root [1] MyClass a
root [2] TFile f("test.root", "recreate")
root [3] a.Write("a1")
root [4] f.Close()
```

- Open a file and read the object:

```
root [1] TFile f("test.root")
root [2] MyClass *a = (MyClass*) f.Get("a1")
root [3] f.Close()
root [4] a->Dump()
```