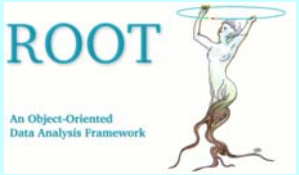
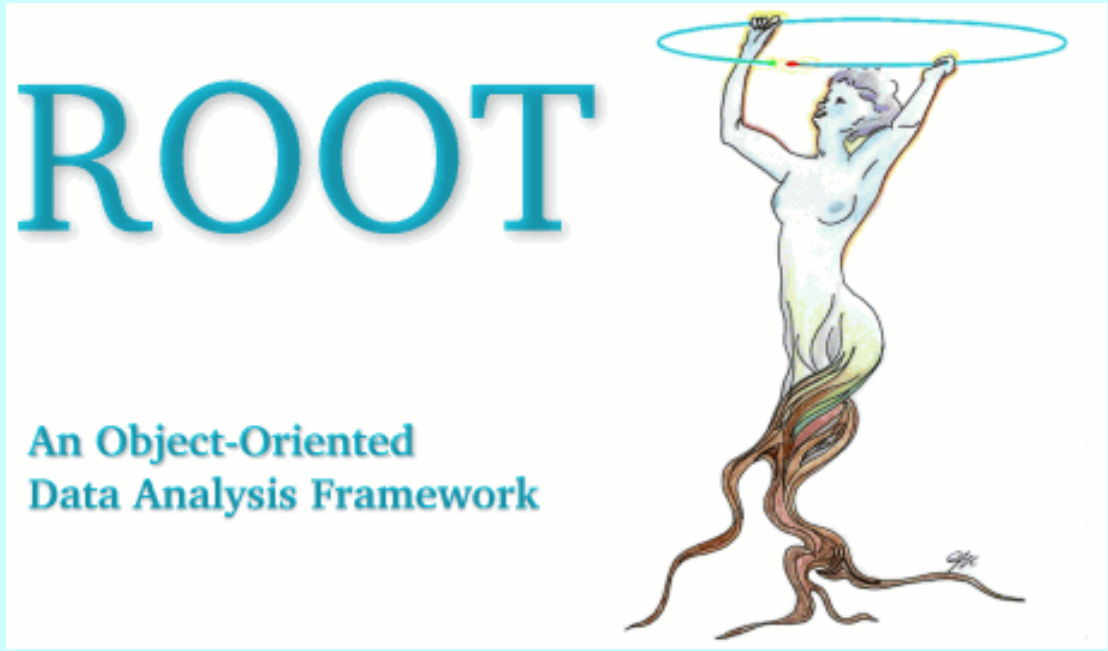
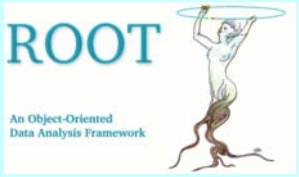




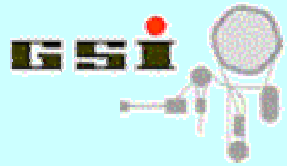
The ROOT framework 2

J. Adamczewski

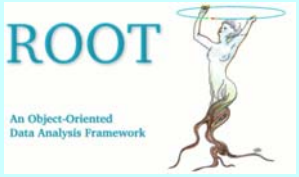
C++ course 2005



Schedule of part 2



- ROOT framework as class hierarchy
- Collection classes
- TFile and TDirectory
- TTree class (set up; read and analyse data)
- (Adding own classes to ROOT)



The ROOT system

C++ based framework for (high energy-) physics

(developed at CERN by R.Brun et al. since 1997)

- **C++ as script language with interpreter CINT**
- **GUI for interactive visualization (TCanvas, TBrowser,...)**
- **I/O and analysis of large amount of data (TFile, TTree,...)**
- **Histograming, plotting, fits (TH1x, TGraph, TF1,...)**
- **Physics and mathematics (TMatrix, TLorentzVector, TMath,..)**
- **Object organisation (TCollection, TDirectory, TFolder,...)**
- **Parallel analysis via network (TProof)**
- **...**
- **see http://root.cern.ch for further info!**

TObject: ROOT top base class

- defines interface of fundamental virtual methods:

```
Draw(), Print(), Dump(), GetName(), Clear(),
Compare(), Streamer(), Clone(), Write(),...
```

- base type for root collections

```
TObject* ob= new TH1F("hpx","title",2048,0,2047);
TList* list=new TList;
list->Add(ob);
TObject* ob2=list->FindObject("hpx");
```

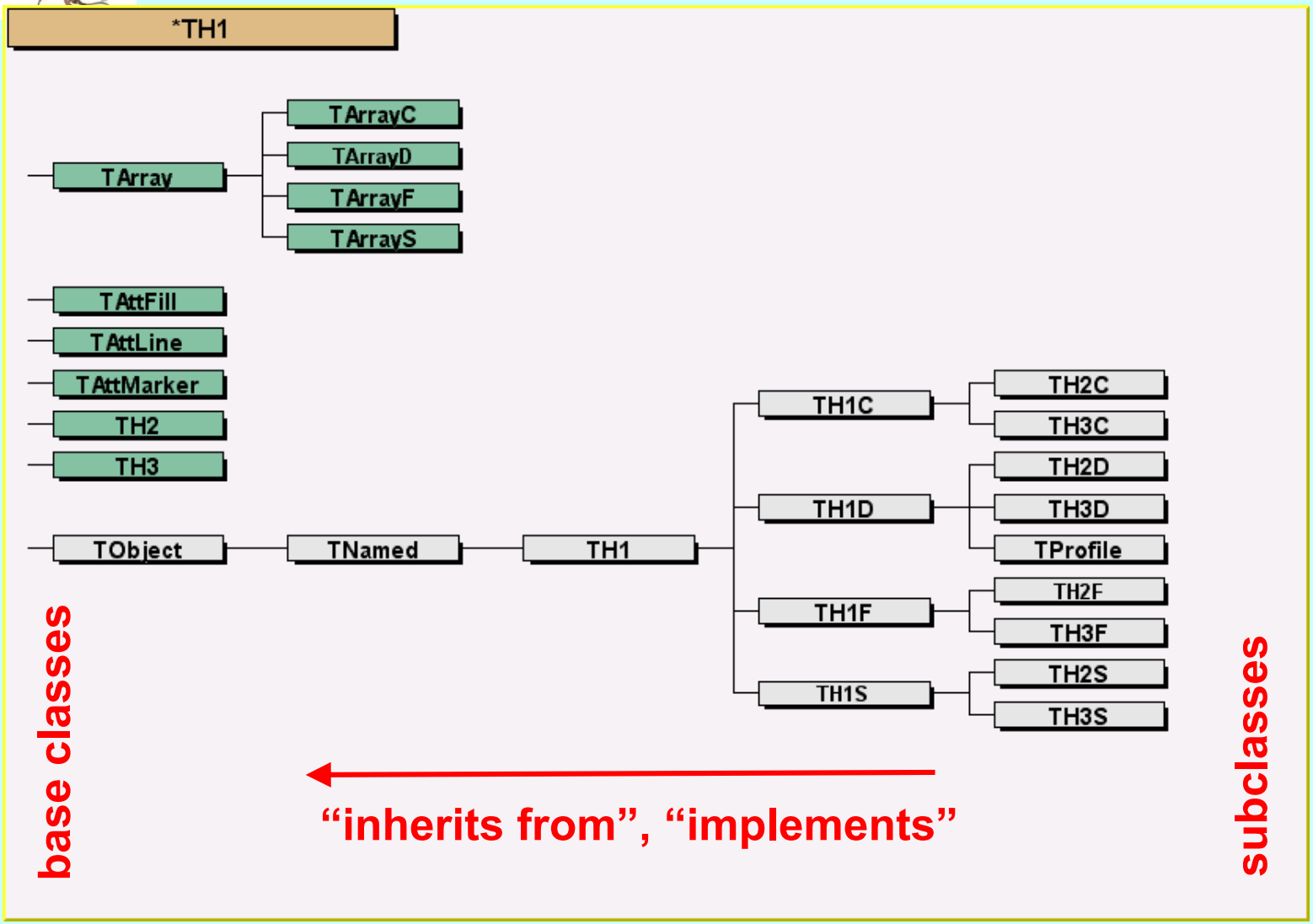
- IO (via TObject::Streamer()):

```
ob->Write(); ob->Clone();
```

- runtime class introspection:

```
TClass* cl=ob->Class(); // full info on methods
                        and datamembers in memory
if(ob->InheritsFrom("TH1"))... // check type
```

TH1: example of class hierarchy



Class hierarchy: some facts

- **Subclass objects are of parent class type:**

  a TH1D histogram „is a“ TObject

- **Subclasses have all members /methods of parent classes**

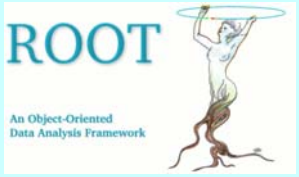
```
TH1D* his=new TH1D("hpx","example",100,0,10);
cout <<"histogram name:" << his->GetName()<<endl;
```

  TH1D uses name property of TNamed

- **Subclasses may redefine virtual methods:**

  TObject::Print() overridden by TH1::Print()

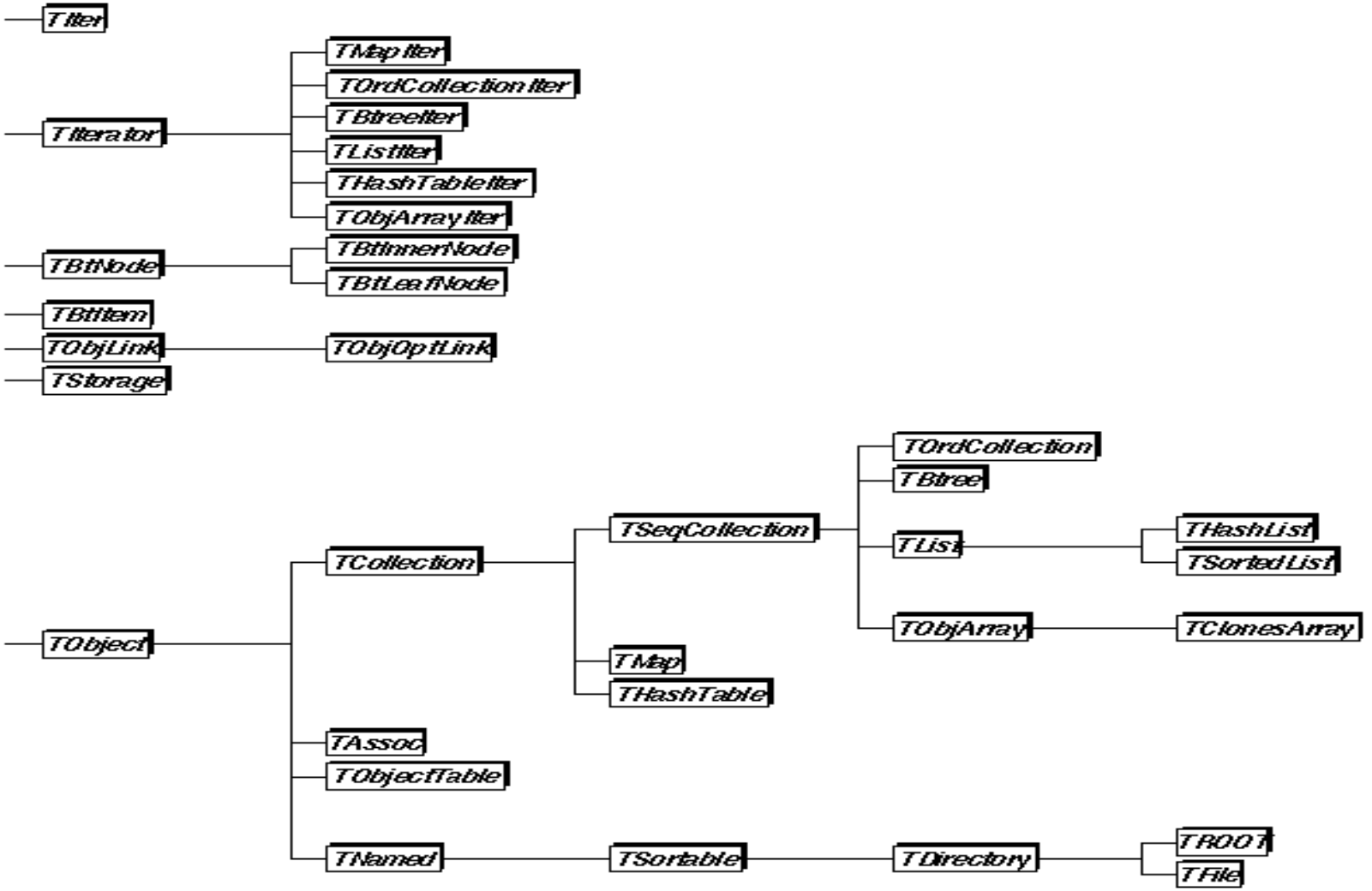
```
TObject* ob=new TH2I("map", "example", 50, 0, 1000, 50, 0, 1000);
ob->Print(); // C++ automatically calls TH2I::Print();
```

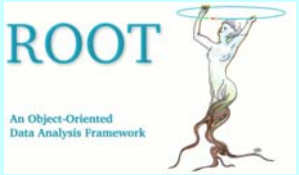


Root collections

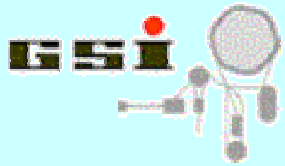
- Example of polymorphic containers
- A ROOT collection keeps any TObject objects
  (different classes may be in same collection!)
- Different types of collections as hierarchy,
  base classes: TCollection, TIterator
  - Ordered: TList, TObjArray, THashList,...
  - Sorted: TBtree, TSortedList
  - Unordered: TMap, THashTable
- Methods to search contents by name; iterate, sort,...

Root collections



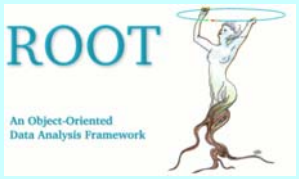


Example: filling a collection




```
TCollection* mylist=new TList;
TObject* h1= new TH1F("hpx","title",2048,0,2047);
mylist->Add(h1);
TObject* g1= new TGraph(50,&a,&b);
mylist->Add(g1);
TObject* n1= new TNamed("Remark1","a short description");
mylist->Add(n1);
....
```

Other TCollection methods:
Remove(), Clear(), Delete(), Draw(), Print(), Write()...



Getting objects from collection

Find by name:

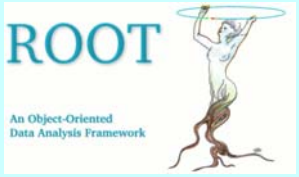
```
TObject* ob=mylist->FindObject("hpx");
if(ob->InheritsFrom(„TH1")){
        TH1* histo=dynamic_cast<TH1*>(ob);
        if(histo) histo->Fill(5); // works only for TH1
}
```

Scan all with Iterator:

```
TIter liter(mylist); // TIter wraps correct TIterator
TObject* out=0;
while((out=liter.Next()) !=0) {
        out->Print(); // implemented for each TObject
        TH1* histo=dynamic_cast<TH1*>(out);
        if(histo) histo->Fill(5); // only for TH1
}
```

TFile and TDirectory

- TDirectory: logical organisation of TObjects (ownership, subdirectory structure,...)
- TFile is a TDirectory related to a storage medium (disk, tape, remote server,...)
- Last created TFile is current file:
  - TH1 and TTrees created afterwards are owned by file and deleted on file close! (default, can be changed by user)
  - TObject::Write() saves to current file
- A Root object belongs to one TDirectory only, but may be in several collections and TFolder



TTree features

- Designed to store large number of events
  (buffering, compression, IO with TFile)
- Data organized hierarchically into “branches“
- Branches may be read back partially (performance!)
- TTree::Draw(“...“) –
  implicit analysis loop by string expression
- TTree::MakeClass() –
  automatic code generation for explicit analysis loop
- TChain : public TTree–
  process sequentially trees of same structure in several files
- TTree::AddFriend –
  access parallel events of a friend tree with different branch structure (for TTree::Draw() expression)

Creating a TTree

```
TFile* hfile = new TFile("AFile.root","RECREATE","Example");
TTree* mytree = new TTree(”Mytree”,”cppworkshop”);
TXXXEvent *event = new  TXXXEvent(); // structure to save
myTree->Branch("EventBranch",“TXXXEvent",&event, 32000,99);
```

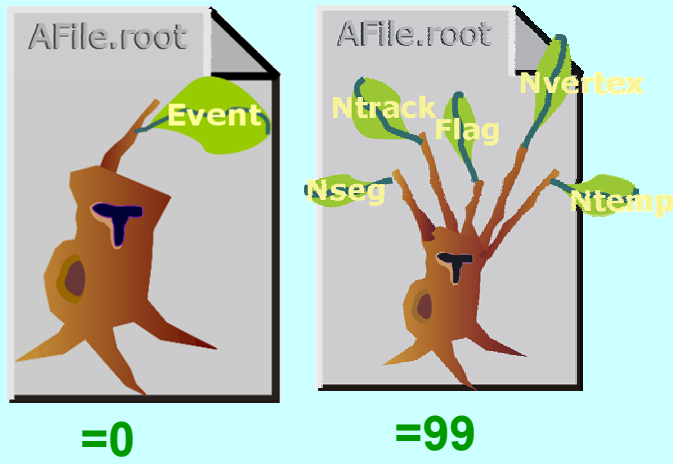
branch name

eventclass name

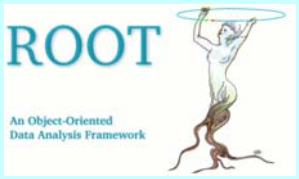
address of pointer
to eventobject

buffersize

splitlevel



=0 =99



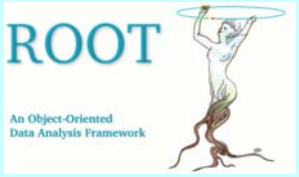
Filling the TTree

```
for(Int_t i=0; i<500000; ++i){
Float_t random = gRandom->::Rndm(1);
event->fValue=random*3; // put values into event structure
event->fSum+=random;
...
mytree->Fill(); // will write event into tree basket buffer
}
mytree->Write(); // will write tree buffers and header to file
delete myfile; // destructor of TFile will close it
```

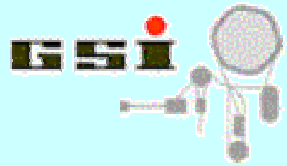
- TTree::Fill will write data from all active branches
  (deactivate branch with TTree::SetBranchStatus)
- Different TTree::Branch() methods for data from simple variables, collections, folders, class objects (see ROOT doc)

Reading a TTree explicitely

```
TFile hfile("AFile.root");
TTree* tr= dynamic_cast<TTree*>(hfile.Get("Mytree"));
if(tr==0){
            cerr << "error: did not find tree!";
            return 1; // or may throw exception here...
      }
TObject* h1= new TH1F("hpx","title",2048,0,2047);
TXXXEvent* eve= new TXXXEvent;
tr->SetBranchAddress("EventBranch",&eve); // by branchname!
Int_t all=tr->GetEntries(); // number of events
for(Int_t i=0; i<all; ++i){
      tr->GetEntry(i); // read event #i into memory
      h1->Fill(eve->fValue);
      // do analysis on members of event class here!
      //...
}
```



Reading Tree explicitely (cont.)



- Event object at SetBranchAddress must match the structure used on writing the tree
- TTree::GetEntry will read data from active branches only (deactivate branch with SetBranchStatus("branchname",0))
- Use TChain instead of TTree to sum trees of same structure in different files (see ROOT doc).
- TTree::MakeClass() generates sourcecode for event reading from given TTree (eventclass need not be known!) See example below...
- Explicit reading of events is not necessary for simple analysis, use TTree::Draw() feature (GUI: treeviewer)!

TTree::Draw() examples

```
TTree* tr= .. // got from file
tr->Draw("fValue","fValue>100 && fValue<500");
       // fill default histogram htemp with fValue if
       // condition is true; draw htemp
tr->Draw("fX:fY >> hpxpy","","lego");
       // fill existing 2d histogram of name "hpxpy"
       // and display as "lego" plot
tr->Draw("fMatrix[][]/fValue >>+hmatrix","");
       // continue filling histogram hmatrix
       // with sum of all elements of matrix by fValue
tr->Draw(">>myeventlist","sqrt(fValue)>fMatrix[0][2]");
       // mark all events in tree that fulfill
       // the condition into TEventList "myeventlist"
```

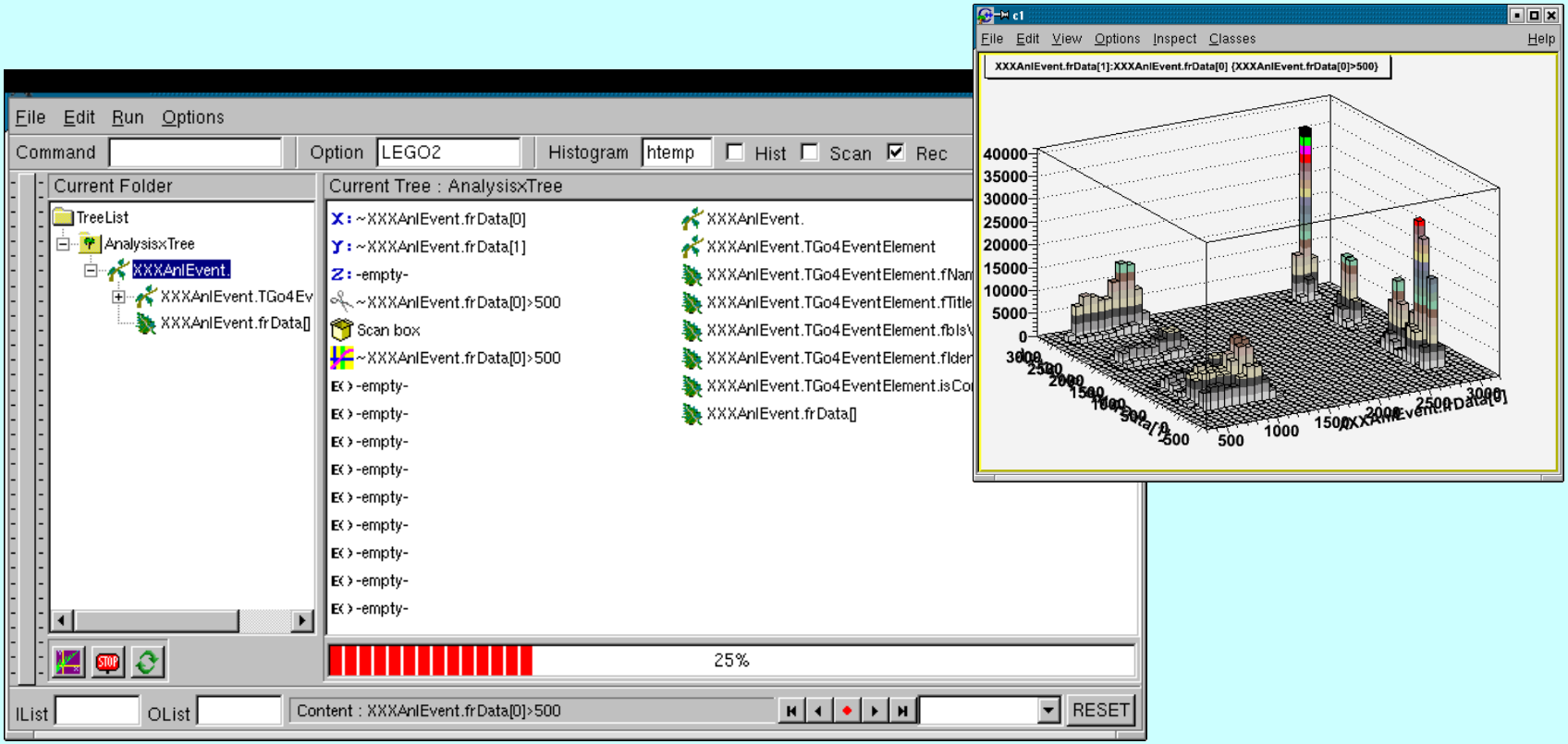
TTree::Draw() (cont.)

`TTree::Draw(expression,selection,option)`

- May fill histogram/graph from expression, or will mark matching events in a TEventList
- Expression may contain any combination of known branch names
- Expression may specify output histogram name and dimensions, or output eventlist
- Selection gives condition between branch values of one event; this must be true to execute expression
- Option may contain draw option for result histogram
- SEE ROOT DOC for complete list of features!

The treeviewer

TTree::Draw by click / drag and drop of tree leaves

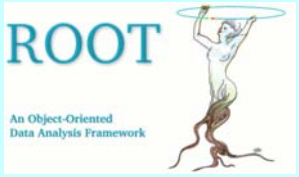


From TBrowser: rmb menu on tree icon in file -> „StartViewer“

TTree::MakeClass()

```
TFile hfile("AFile.root");
TTree* tr= dynamic_cast<TTree*>(hfile.Get("Mytree"));
if(tr!=0) tr->MakeClass("MyAnalysis");
```

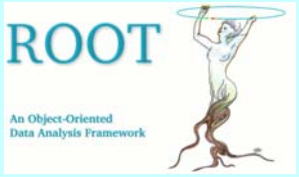
- Generates code sceleton for analysis of any TTree
  (files MyAnalysis.h, MyAnalysis.C)
- Tree is analyzed by generated class MyAnalysis:
  - members contain each branch/leaf found in tree
  - constructor initializes tree/chain from file(s)
  - Init(TTree*) sets branch addresses to members
  - Show(int num) dumps entry #num
  - Loop() – here user can put own analysis code



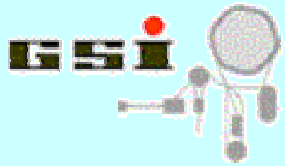
TTree::MakeClass()

```
root> .L MyAnalysis.C
root> MyAnalysis an; // should initialize tree
root> an.Loop();     // run implicit analysis loop
```

- Class MyAnalysis may be used from CINT or can be compiled
- Quick code generation even for unknown data structures!
- Substructure of original event class may be lost, all tree branches are mapped flat to MyAnalysis data members
- See Root users guide for further description...



Adding own classes to ROOT



Motivation: User subclasses of TObject may benefit from ROOT IO, collections, runtime introspection,...

(event structure, analysis parameters,...)

- **Interpreter: just load class definition (see MakeClass example), but no IO for new class possible!**
- **Compiled into user library:**
  - **Add ClassImp / ClassDef statements in class sources**
  - **prepare LinkDef.h file**
  - **provide dictionary generation in Makefile**

Adding own classes to ROOT

MyEvent.h

```
class MyEvent : public TObject{

public:
        MyEvent();
        fValue;
        fMatrix[100][100];
        // lots of data members here...

ClassDef(MyEvent,1)
};
```

Macros, create code for Streamer, type information, etc.

MyEvent.cxx

```
ClassImp(MyEvent)

MyEvent::MyEvent(){}
// may contain othermethod definitions...
```

Adding own classes to ROOT

LinkDef.h

```
#ifdef __CINT__
#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;
#pragma link C++ class MyEvent;
#endif
```

Class name may have Options:

`#pragma link C++ class MyEvent-;`

do not generate automatic streamer (for objects with customized streamers)

`#pragma link C++ class MyEvent!;`

do not generate the operator >> (for classes not inheriting from TObject)

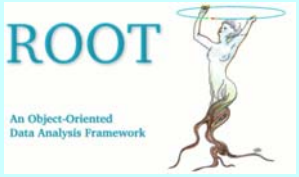
`#pragma link C++ class MyEvent+;`

enable new ROOT IO (from Root >v.3)

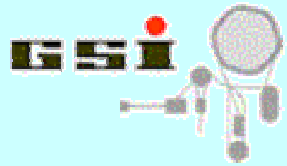
Adding own classes to ROOT

in Makefile (for generation and linking of ROOT dictionary):

```
libMyEvent.so: MyEvent.o MyEventDict.o
   g++ -shared -Wl -soname libMyEvent.so -O MyEvent.o
   MyEventDict.o -o libMyEvent.so
#  ....
#  ....
MyEventDict.cxx MyEvent.h LinkDef.h
    $(ROOTSYS)/bin/rootcint -f MyEventDict.cxx -c
      MyEvent.h LinkDef.h
```

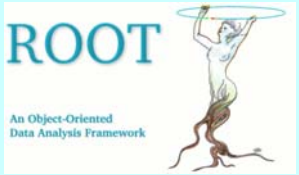


Summary and outlook

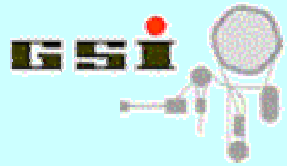


- **ROOT offers collections, folders, directories and files for organization of TObjects**
- **TTree is powerful class for keeping and analyzing eventdata**

  (Draw(), MakeClass(), MakeSelector(),..)
- **User can „ROOTify“ own classes with a simple recipe (ClassImp, ClassDef, LinkDef.h, Makefile)**



Exercises



1. Create different TH1 and put them into a TObjArray. Use TIterator to fill/modify all of them, or do something under certain conditions. Try collection methods: FindObject(), Draw(), Print(), Write(), Clear(), Delete()...
2. Open existing TTree from file in CINT. Play with treeviewer. Write macro that uses TTree::Draw to fill histogram with defined binning
3. Open TTree from file in CINT and use MakeClass() to produce analysis skeleton. Edit the Loop() method to do something with leaves and run this in CINT/ACliC. Try to compile this to library.
4. Define an own event structure class (TObject subclass) and compile it into library. Use this class to fill a TTree with random numbers (or real data) and save it to TFile. Get back events from tree with explicit eventloop. Enjoy!