What is ROOT?

Tool and a Framework for OO Data Analysis

- As a Tool
  - Enter commands to display and manipulate data
  - Commands are C++
    - (covers ~85% of full language – including simple templates)
  - Can use to prototype new code
- As a Framework
  - Use ROOT as a library and link in user C++ code
  - Can use result as a tool
- Supports User Classes
  - User can define new classes
  - Can do this either as Tool or Framework
  - These can inherit from ROOT classes

Caution

- Rapidly Developing
  - Has a few rough edges!
  - But support team respond rapidly to bug reports
- Now CERN Supported
  - CERN's official replacement is LHC++
    - However CERN now also support ROOT, particularly for analysis
  - ROOT development lead by Rene Brun
    - One of founders of PAW
    - Principle: Use free packages and support wide range of platforms
      - (including Linux and gcc)
  - Does have a wide user base
    - Both within HEP:-
      - NA49, CMS, PHOBOS, E907, ATLAS, BABAR, STAR, ALICE, CDF, D0, MINOS, BLAST ...
    - and Beyond:-
      - Financial sector, health screening, network performance, satellite monitoring,...

ROOT Components

- Base Classes
  - Objects
  - Files and Directories
  - I/O
  - System interface
  - Basics maths
- Containers
  - Collections
  - Lists
  - Arrays
  - Maps (hashs)
- Histogram and Minimisation
  - Up to 3D
  - Profile
  - Minuit (general fitting package)
- Trees and N-tuples
  - Generalise n-tuple concept to a binary tree of objects
- Matrices

Components (cont).

- 2D Graphics
  - lines, text, shapes etc.
- 3D Graphics and Geometry
  - 3D shapes e.g. cone, helix
  - Geometry description
    - Tree of nodes
    - Nodes properties: shape, material, rotation
- GUI
  - Toolkit to build GUI
- Meta Data
  - For describing classes
- Network
  - Access to network, including HTTP
- Documentation
  - Build HTML directly from source code
- Interactive Interface
  - For user application to act as a tool
- Parallel Process Support

C++ Syntax Primer

- Just enough to understand example

- Creating an Object
  - E.g ROOT class TH1F (1D histogram with one float/chan):-
    - `TH1F my_hist;`
  - Creating an object causes its *constructor* function to be called
  - Often constructor passed args e.g.:-
    - `TH1F my_hist("hfix", "hfix title",nbins,xlow,xup);`

- Interacting with an Object
  - By calling its member functions e.g.:-
    - `my_hist.Fill(x);`
      `my_hist.Draw(”option”);`

    to fill and draw the histogram.

- Above Syntax is for Auto Objects
  - Created on program stack
  - 🕓 FORTRAN local variables
  - O.K. for short lived data

C++ Syntax Primer (cont)

- Snags with Auto Objects
  - Local to a single function
    - Disappear when function exits
    - But objects should be independent
  - Must decide how many at compile time
    - May want to decide at execution time
- Solution: Heap Variables
  - Created in pool of free memory
  - Space reserved using 'new' function
  - Exist as long as required
  - Snag:
    - Compiler does not know objects address
    - Have to use a pointer
    - Notation:

```
TH1F *my_hist
```

is a pointer to object of type TH1F

C++ Syntax Primer (cont)

- Creating a Heap Object
  - Example:-

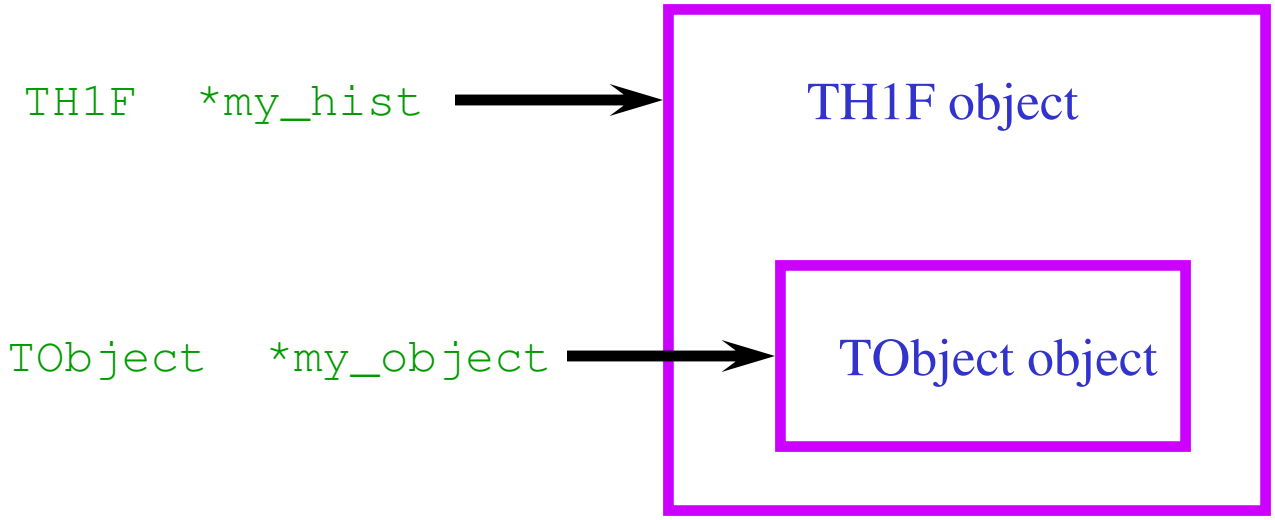
```
TH1F *my_hist =
  new TH1F ("hfix",
       "hfix title",nbins,xlow,xup);
```

    - new:-
      - creates a TH1F object on heap
      - calls its constructor function
      - returns a pointer
- Interacting with Heap Objects
  - Follow pointer and then call member
  - This is done with -> operator e.g.:-

```
my_hist->Fill(x);
my_hist->Draw(”option”);
```

C++ Syntax Primer (cont)

- Pointers and Inheritance
  - ROOT Class TH1F inherits from class TObject
  - So a TH1F object contains a TObject object.
  - Pointers can point to either:-



- Pointers can be converted
  - Moving down inheritance tree:-

    ```
    my_object = my_hist;
    ```
  - Moving up inheritance tree:-

    ```
    my_hist = (TH1F*) my_object;
    ```

    Can be dangerous: What if TObject wasn't in a TH1F?

ROOT Primer

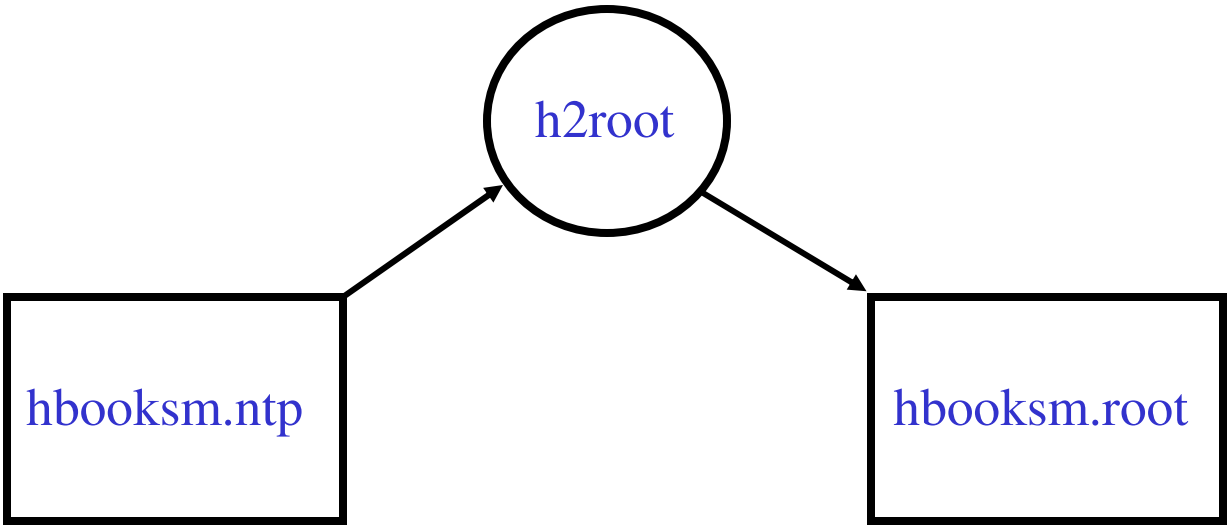
- Naming Convention
  - TName e.g. TList
- TObject
  - Is the primordial object
  - Most other classes inherit from it
  - Provides base for generic operations such as:-
    - I/O, Graphics, Containerisation
- Graphics: TCanvas, TPad
  - TCanvas is rectangular window holding TPads.
  - TPad maps to a rectangular area on a TCanvas
  - TPad holds a list of objects (including TPads) to be displayed

ROOT Primer (cont)

- File access: TDir and TFile
  - A TDir is a directory.
    - It holds a list of named objects (can include other TDir)
  - A TFile is a file.
    - It consists of a series of TDir objects.
  - Reading from a file
    - Involves passing TFile the name of the object to be retrieved. It returns pointer.
- Event I/O: TTree and TBranch
  - General concept of an event
    - A heterogeneous collection of objects.
  - All have to be output together
  - A TBranch holds a collection of objects
    - It can include TBranch. It has its own buffer.
  - A TTree is a collection of TBranch.
    - It synchronises I/O.
  - But, can just input partial event
    - Select TBranch
    - Input rest of Ttree conditionally

ROOT Demo

- Working with an HBOOK N-tuple
  - Converted using h2root:-



- hbooksm.ntp
  - n-tuple 800
  - has variables:-
    - Vx_x
    - Vx_y
    - Vx_z

- hbooksm.root
  - TTree h800
  - TBranch:-
    - Vx_x
    - Vx_y
    - Vx_z

ROOT Demo Code

```
{
// Clear out any object created by user in the current session by
// sending Reset message to the master object gROOT.

  gROOT->Reset();

// Create a canvas (window) and within it define 3 pads (sub-windows
// holding graphical material).

// Create canvas giving name, title,  top left corner, width and height
// in pixels.

  c1 = new TCanvas("c1","ROOT Demo",200,10,700,500);

// Create pads giving name, title, limits (as fraction of canvas) and
// background colour (8 = white)

  pad1 = new TPad("pad1","Pad1: (Top half)",    0.02,0.52,0.98,0.98,8);
  pad2 = new TPad("pad2","Pad2: (Bottom left)", 0.02,0.02,0.48,0.48,8);
  pad3 = new TPad("pad3","Pad3: (Bottom right)",0.52,0.02,0.98,0.48,8);

// Tell the pads to draw themselves.  This actually adds them to the list
// of objects that the canvas holds.  Later, when the canvas is sent the
// Update message, it will send an Update message to all its pads.

  pad1->Draw();
  pad2->Draw();
  pad3->Draw();

// Create a File object as an input from the file hbooksm.root.

  TFile *hfile = new TFile("hbooksm.root", "READ");

// Set up a Tree object pointer by asking hfile to find the object whose
// name is h800 (the name created by h2root for n-tuple 800).  The Get
// message returns a pointer to Object so have to be cast up to a Tree.

  TTree *my_ntuple = (TTree *) hfile->Get("h800");
```

ROOT Demo Code (cont)

```
// Make pad1 the current working graphics directory by sending it cd
// (cf. Unix).  From now on, any Draw message will draw in this pad.

   pad1->cd();

// Send the n-tuple a Draw message, supplying the expression to be drawn.
// This automatically creates and fills a histogram object (like PAW).
// pad1 will contain this histogram.

   my_ntuple->Draw("Vx_z");

// In a similar way, plot a 2d histogram in pad2.

   pad2->cd();

// This time we tell the n-tuple to change the defaults to be used when
// creating the 2d histogram.

   my_ntuple->SetFillColor(5);
   my_ntuple->SetMarkerStyle(3);

// Note the syntax is different to PAW (Vx_z%Vx_x).

   my_ntuple->Draw("Vx_z:Vx_x");

// Finally plot a 3d plot in pad3.  This time, we also place a cut on the
// data to be plotted.

   pad3->cd();
   my_ntuple->SetMarkerStyle(7);
   my_ntuple->Draw("Vx_z:Vx_y:Vx_x",
                   "sqrt(Vx_x**2+Vx_y**2+Vx_z**2)<5000.");

// Now tell the canvas to update itself, causing all its pads to tell all
// the objects they contain to paint themselves.

   c1->Update();

}
```

ROOT Demo Code (cont)
Result:-