# ROOT Part 2

# Recap

- ROOT is a commonly used set of libraries for HEP
- Extensive documentation available online
- ROOT useful as a calculator, with many TMath function available
- Huge number of ROOT classes available
- TF1 class used for 1D functions
- TH1* classes used for 1D histograms
- Information drawn to TCanvas objects to create plots

# ROOT Files

- ROOT files (*.root) can hold any type of ROOT objects
- Most commonly used to hold histograms and trees
  - .root files holding only a tree are often referred to as ntuples
- Open a .root file as a TFile using: root myfile.root
- From within ROOT, open with:

```
TFile *f = new TFile("myfile.root","<mode>")
```

  - <mode> can be NEW or CREATE, RECREATE, UPDATE, READ (default is READ)
- Check if file opened correctly using IsZombie()
- List contents of open file with: .ls
- Close a TFile using Close()

# Trees and branches

- A tree (TTree) is a list of independent columns (called branches) of data
  - Branches are represented by the TBranch class
- Trees are access by entries (rows of data)
  - Often representing each event (LHC collision), but can be divided in other ways
- Each branch holds information for every entry
  - Default branch values are often used in case data is missing for an entry
- Branches can hold primitive types, vectors, strings, or more complex types
- Buffers for reading/writing branch information are used behind the scenes
  - High performance work can involve optimizing buffers

# TTree Print

- Explore the branches in a TTree using Print()

```
root [2] analysis->Print()
******************************************************************************
*Tree    :analysis  : My analysis ntuple                                     *
*Entries :    20000 : Total =        12213898 bytes  File  Size =    4232156 *
*        :          : Tree compression factor =   2.88                       *
******************************************************************************
*Br    0 :RunNumber : RunNumber/i                                            *
*Entries :    20000 : Total  Size=      80741 bytes  File Size  =        687 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression= 116.80     *
*............................................................................*
*Br    1 :EventNumber : EventNumber/l                                        *
*Entries :    20000 : Total  Size=     161009 bytes  File Size  =       1375 *
*Baskets :        6 : Basket Size=      32000 bytes  Compression= 116.72     *
*............................................................................*
*Br    2 :nX        : nX/I                                                   *
*Entries :    20000 : Total  Size=      80692 bytes  File Size  =       6458 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=  12.42     *
*............................................................................*
*Br    3 :X_pt      : vector<float>                                          *
*Entries :    20000 : Total  Size=     350383 bytes  File Size  =     136726 *
*Baskets :       14 : Basket Size=      32000 bytes  Compression=   2.56     *
*............................................................................*
```

- Use string argument with wildcards (*) to see subset

# TTree Scan

- Print example values from TTree using Scan()
- Useful when trying to quickly understand what a TTree holds
  - Does not print out all branches by default
  - Use "var1:var2:var3" string as an argument to print var1, var2 and var3 branches

```
root [6] analysis->Scan()
************************************************************************************************************
*    Row   * Instance * RunNumber * EventNumb *     nX.nX *      X_pt *     X_eta *     X_phi *       X_m *       X_e *
************************************************************************************************************
*        0 *        0 *         0 *         0 *         1 * 110869.48 * -1.700609 * 1.4408147 * 1500007.6 * 1532469.7 *
*        1 *        0 *         0 *         0 *         1 * 341185.78 * 1.7406495 * 2.5384585 * 1499996.3 *   1804150 *
*        2 *        0 *         0 *         0 *         0 *           *           *           *           *           *
*        3 *        0 *         0 *         0 *         1 * 191936.14 * -2.939772 * -2.727361 * 1500005.1 * 2358461.2 *
*        4 *        0 *         0 *         0 *         1 * 94515.007 * -0.856753 * -0.491988 * 1499994.1 * 1505736.5 *
*        5 *        0 *         0 *         0 *         1 * 55717.816 * 1.6148117 * 1.6601346 * 1500000.1 * 1507048.7 *
*        6 *        0 *         0 *         0 *         1 * 731311.56 * 0.8595652 * -2.041293 * 1499955.3 *   1813081 *
*        7 *        0 *         0 *         0 *         1 * 43621.820 * -3.739025 * 0.0899386 * 1500004.5 * 1758523.8 *
*        8 *        0 *         0 *         0 *         1 * 71207.757 * 3.8558113 * -2.675648 * 1500004.6 * 2254918.7 *
*        9 *        0 *         0 *         0 *         1 * 34468.082 * -0.841401 * -1.371616 * 1499988.2 * 1500737.1 *
*       10 *        0 *         0 *         0 *         1 * 141170.39 * 1.0811071 * -1.115002 * 1499999.2 * 1517838.3 *
```

# TTree Draw

- Quickly draw the contents of a branch using Draw()

```
tree->Draw("branch")
```

  - If branch is a vector, all entries in the vector will be drawn by default

- Add selection criteria with a second string argument

```
tree->Draw("branch","branch > 3 && branch < 10")
```

- Simple branch manipulations can be done

```
tree->Draw("branch1+branch2")
```

- Output can be piped into a histogram for later use

```
tree->Draw("branch>>h1")
```

- More advanced logic is possible in draw commands

# TBrowser

- TBrowser GUI makes exploring ROOT files easier

  new TBrowser     or     TBrowser t

  - Tab autocomplete is useful for being lazy
- New version of ROOT have a faster web-based browser
  - Use old version by opening root with --web=off
- TBrowser interface can be used to modify what is drawn and save to image
- Often painfully slow when working on a remote machine
- Not designed to get reproducible canvases

# ROOT Macros

- C++ macros (*.C files) can be used to call available ROOT functions
- Main function needs to have the same name as macro
- Header files for used classes do need to be included if using CLING

mymacro.C:

```
void myMacro() {
    TH1F *h1 = new TH1F("h1","h1",20,0,10);
    h1->Fill(6.7);
    std::cout << h1->Integral() << std::endl;
    return;
}
```

# Running ROOT Macros

- Macros can be called through the CLING interpreter or compiled
  - CLING interprets C++ similar to the way python is interpreted
- Within root, execute a macro using:

```
.x mymacro.C
```

- Or call using (for CLING interpreter):

```
root mymacro.C
```

- Or with (to compile the code):

```
root mymacro.C+
```

# Opening TFile and getting objects

```
TFile *f = new TFile("file.root","READ");
```

- Retrieve objects saved in TFile using Get()
  - Get() returns a TObject, so it needs to be explicitly cast into the correct class

```
TTree *tree = (TTree*)f->Get("atree")
```

- Object is linked to original TFile, so do not close TFile while using object
- Make a clone of an object using Clone() (be sure to cast)

```
TTree *mytree = (TTree*)tree->Clone("mytree")
```

# Access TTree

- Local variables need to be declared and linked to branches in TTree

```
int m_var1;
std::vector<float> *m_var2 = nullptr;
tree->SetBranchAddress("var1",&m_var1);
tree->SetBranchAddress("var2",&m_var2);
```

- Useful, but not necessary to use the same name for variable and branch
- When each entry is retrieved, the local variables store the branch data

# Loop over TTree

- Usually you will want to define procedure for each TTree entry
- Get the number of entries in the TTree:

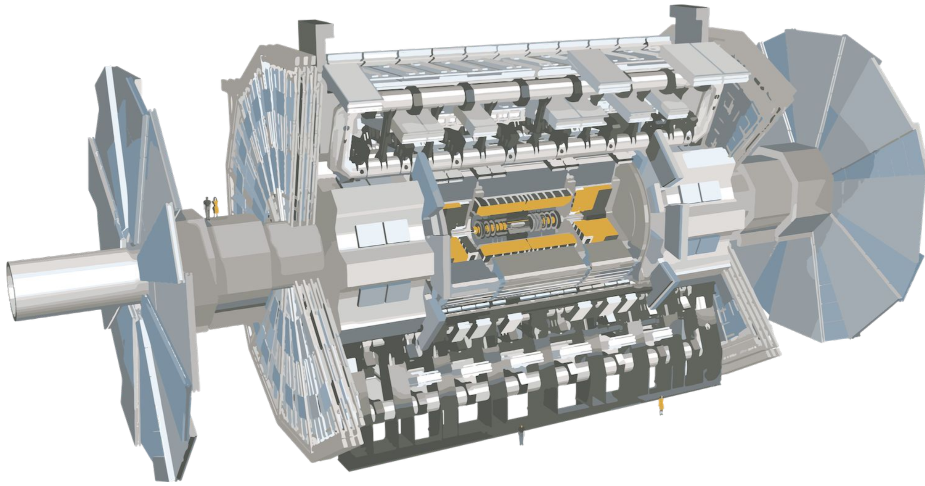
```
Long64_t nEntries = tree->GetEntries();
```

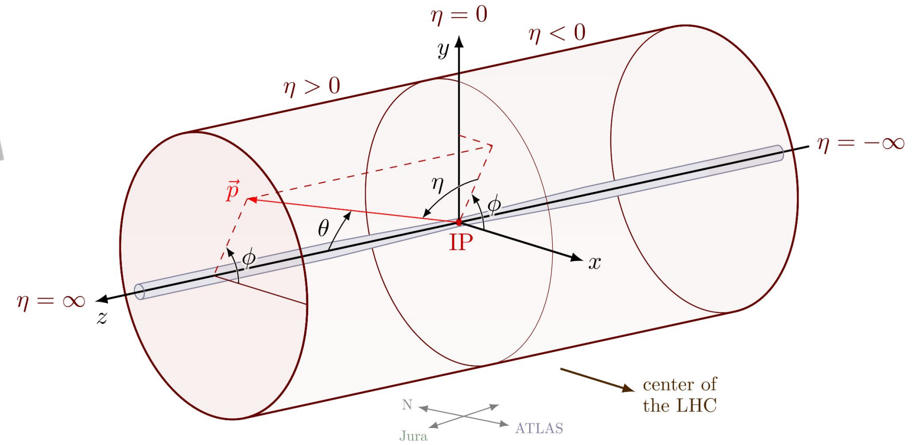
- Iterate over entries with a for loop and use GetEntry() to access each entry:

```
for (Long64_t i = 0; i < nEntries; i++) {
    tree->GetEntry(i);
    // put other per-entry code here
}
```

- GetEntry() assigns current branch value to each linked variable

# Detector Coordinates






https://atlas.cern/Discover/Detector

https://tikz.net/axis3d_cms/

# TLorentzVector

- Relativistic calculations are central to ROOT functionality
- 4-vectors are defined with 4 components:
  - $p_x$, $p_y$, $p_z$, E
  - $p_x$, $p_y$, $p_z$, m
  - $p_T$, eta, phi, E
  - $p_T$, eta, phi, m
- TLorentzVector class is used for 4-vector manipulation
  - Define 4-vectors
  - Add, subtract, transform
  - Retrieve 4-vector components

# TLorentzVector II

- Define a TLorentzVector using:

```
TLorentzVector myTLV;
myTLV.SetPtEtaPhiM(pt,eta,phi,m);
myTLV.SetPtEtaPhiE(pt,eta,phi,E); // alternative
```

- Set individual components using SetE(), SetM(), SetEta(), etc.
- Access individual components using:
  - Pt() or Perp() to get transverse momentum
  - M() to get mass
  - Phi() to get azimuthal angle
  - Eta() to get the pseudorapidity

# TLorentzVector III

- Two or more TLorentzVectors can be added together to create a new TLV

```
TLorentzVector sumTLV = TLV1 + TLV2;
TLorentzVector diffTLV = TLV1 - TLV2;
```

- Note that individual components do not sum together directly

```
TLV1.M() + TLV2.M() // sum of two masses
```

is **not** equal to

```
(TLV1 + TLV2).M() // invariant mass
```

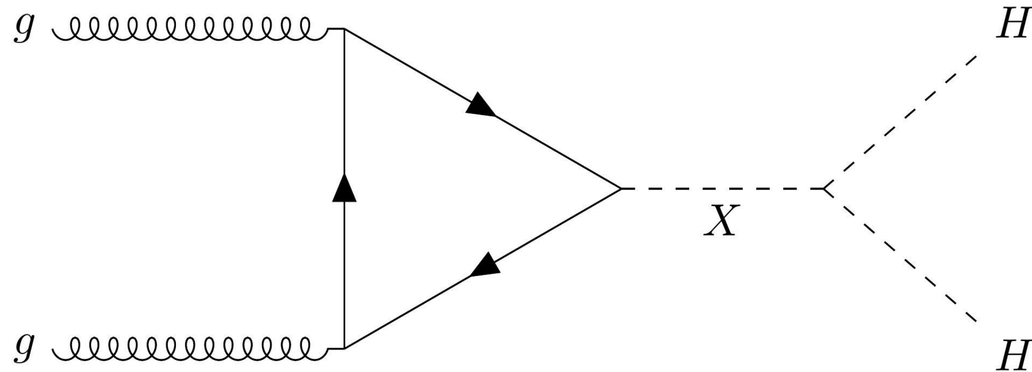
- Use Boost(), Rotate(), and Transform() to modify TLV in well-defined ways
    - Beyond the scope of this class

# Save objects to output ROOT file

- ROOT objects need to be added to files explicitly
- The TObject Write() function saves object to current directory
- Latest directory (or file) to be used is the current directory
- Best practice is to call file->cd() before calling Write()
- If writing same object to file multiple times, multiple snapshots are saved

# DiHiggs signal

- Many Beyond the Standard Model (BSM) theories predict heavy particles
- One possibility is a new scalar (X) that decays into two Higgs bosons (H)
- Look at events where one H decays to bb and the other decays to ττ



# ROOT Documentation

- Extensive documentation available on ROOT website
  - https://root.cern/manual/basics/ - good starting point
  - https://root.cern/doc/master/ - provides all class definitions
  - https://root.cern/doc/master/group__Tutorials.html - good tutorials
  - https://root-forum.cern.ch/ - ask questions to experts (or find existing questions)
- ROOT naming conventions:
  - Class/namespace and member functions are in UpperCamelCase (a.k.a. PascalCase)
  - Most classes/namespaces begin with T
  - Non-class types end in _t
- When using Google, begin search with “CERN ROOT”
  - ROOT refers to the top level directory in a file system or the name of an admin account