PROOF tutorial
Trees Basics

Gerardo Ganis, CERN, PH-SFT
gerardo.ganis@cern.ch

Why trees?

- Typical HEP data analysis
  - Large number of independent events (>10^9 @ LHC)
  - Write Once Ready Many times (WORM case)
  - Analysis use only subsets of the stored info
- Tree-like structures allows fast direct and random access to any (part of) the entry
  - Sequential access remains the fastest
- Provides handle to optimize network access
  - Selective read-ahead
- In ROOT trees buffered to disk (via TFile)
  - I/O integral part of the tree concept

The tree structure in ROOT

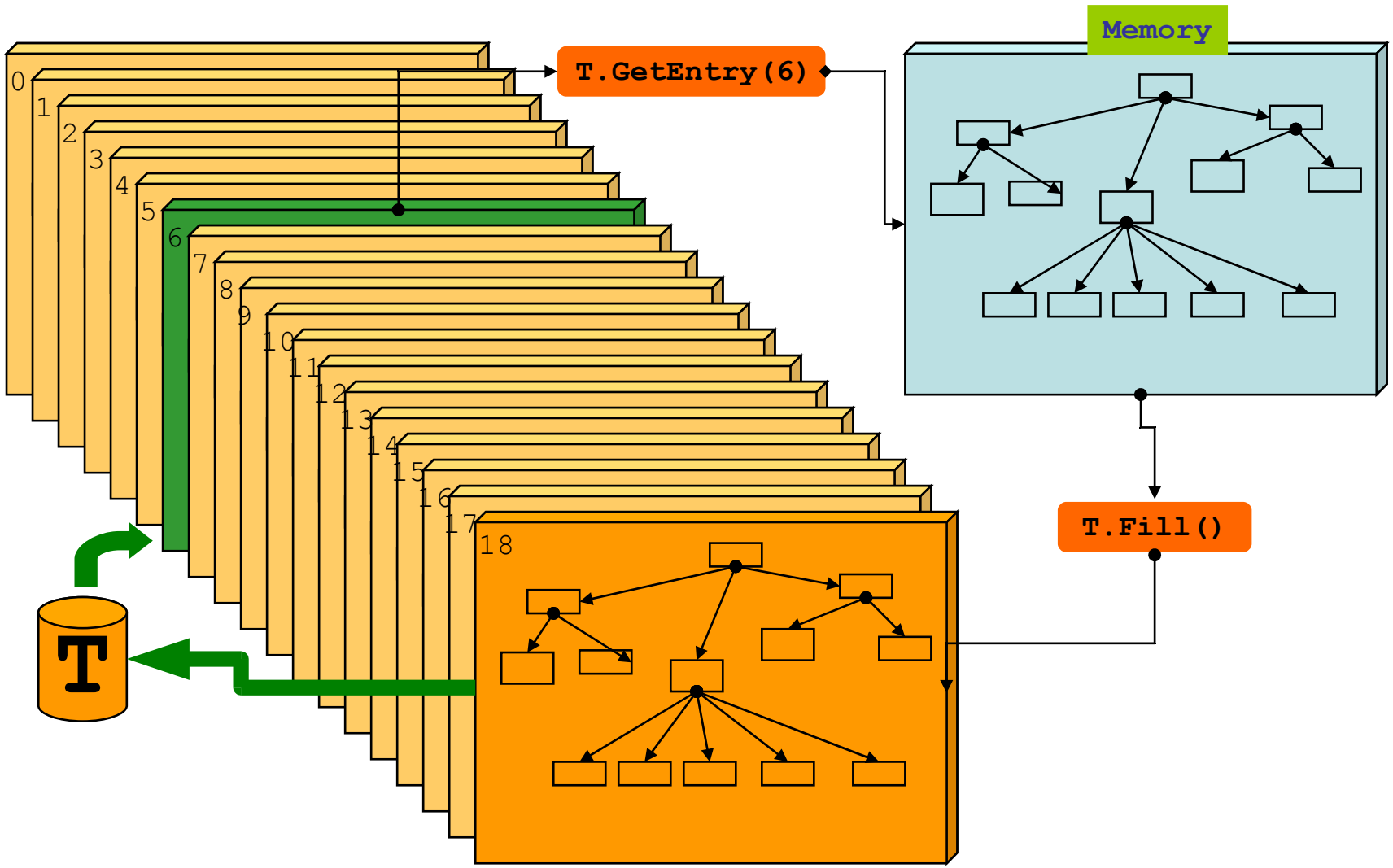
- TTree, the managing class
- TBranch, branch description
  - Directories
- TLeaf, the end leaf description
  - Data types
- Reading a subsets of branches / leaves speeds-up data analysis
  - Layout should be optimized for a specific analysis
- Can even store branches to separate files to further increase reading perfomance

Tree access

- Databases access data row-wise
  - Can only access the full record, i.e. the full event
- ROOT trees access data column-wise
  - Direct access to any branch, any leaf
  - Direct access to subsets of object attributes
    - E.g. the particle energy
  - High compression efficiency
    - Members of the same type accessed consecutively, e.g. object {X,Y,Z} and E, first all X, then all Y, then all Z and finally all E

Memory ↔ Tree

Each Node is a branch in the Tree



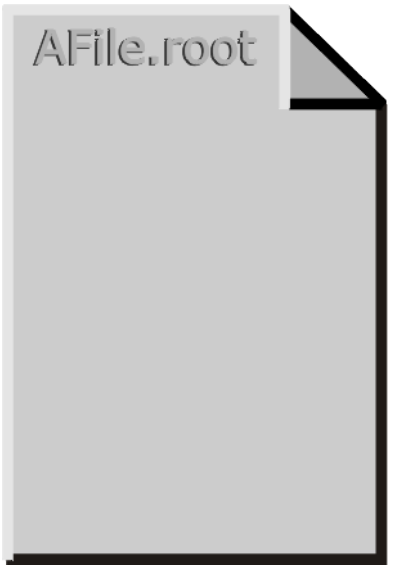
Five Steps to Build a Tree

Steps:

1. Create a TFile
2. Create a TTree
3. Add TBranch to the TTree
4. Fill the tree
5. Write the file

Step 1: Create a TFile Object

Trees can be huge → need file for swapping filled entries

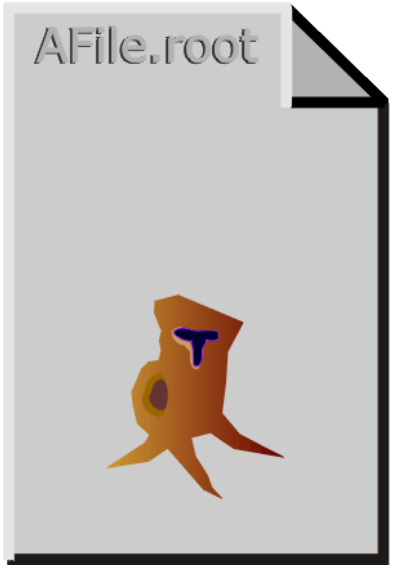



```
TFile *hfile = new TFile("AFile.root");
```

Step 2: Create a TTree Object

The TTree constructor:

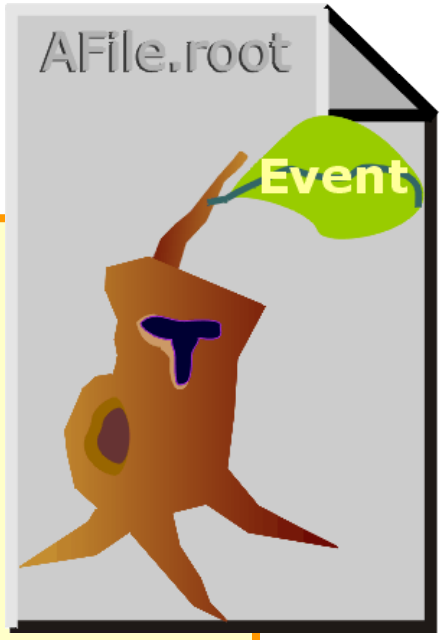
- Tree name (e.g. "myTree")
- Tree title




```
TTree *tree = new TTree("myTree","A Tree");
```

Step 3: Adding a Branch

- Branch name
- Address of the pointer to the object

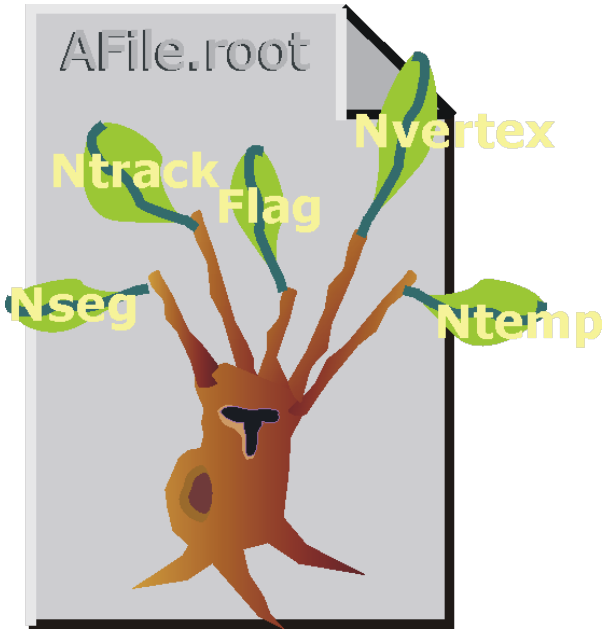



```
// Basic type
Double_t pt;
myTree->Branch("pt", &pt, "pt/D");
// Array
Float_t f[10];
myTree->Branch("f", f, "f[10]/F");
// Variable size array
Int_t nPhot;
Float_t E[500];
myTree->Branch("nPhot", &nPhot, "nPhot/I");
myTree->Branch("E", E, "E[nPhot]/F");
// New Type
Event *event = new Event();
myTree->Branch("EventBranch", &event);
```

Splitting a Branch

Setting the split level (default = 99)

Split level = 0



Split level = 99

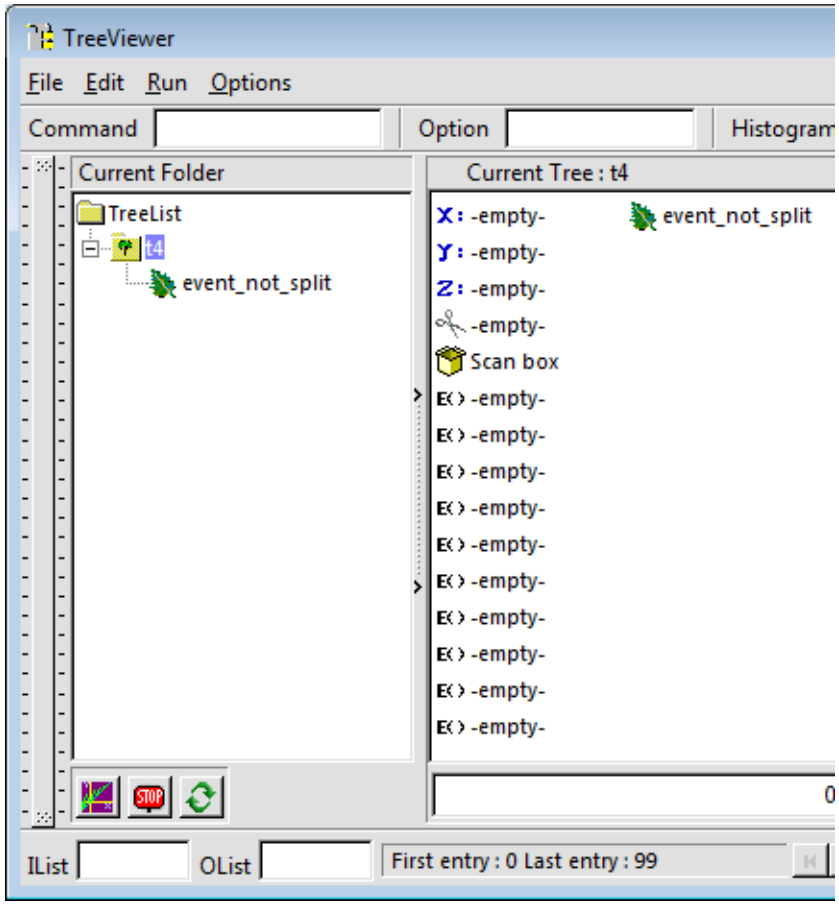
```
tree->Branch("EvBr", &event, 64000, 0 );
  tree->Branch("EvBr", &event, 64000, 99 );
```

Splitting

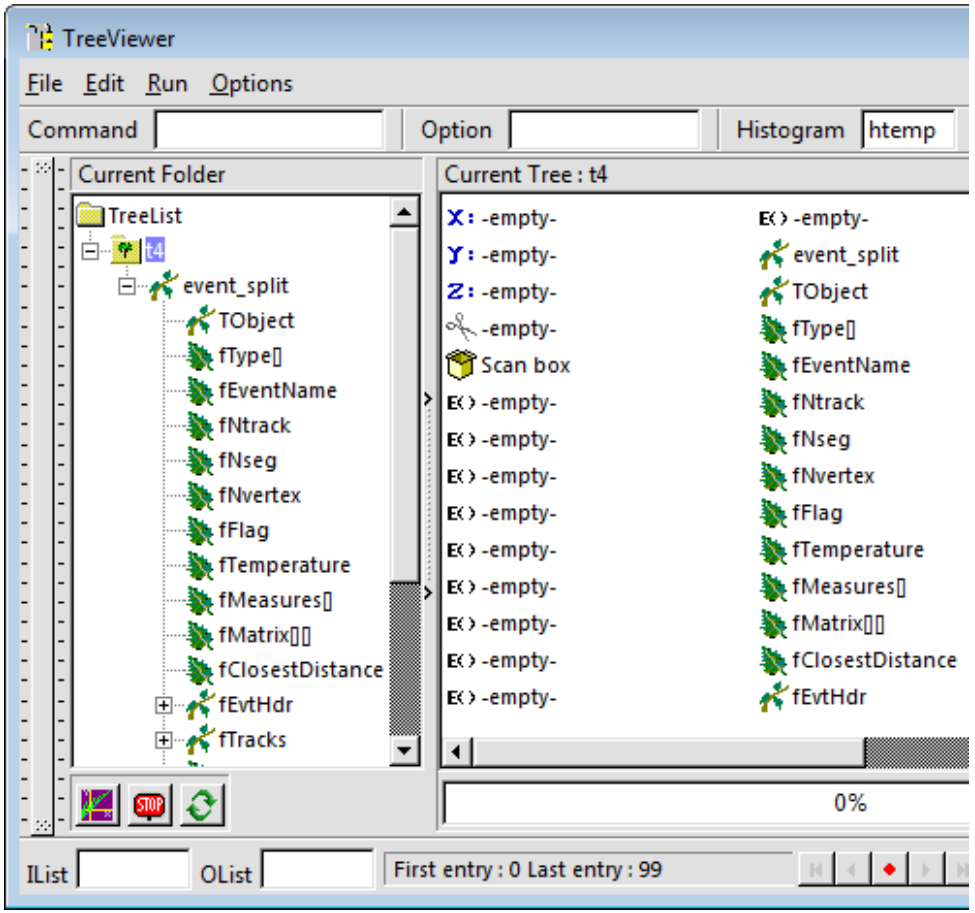
- Creates one branch per member – recursively
- Makes same members consecutive
  - E.g. for object with position in X, Y, Z, and energy E, all X are consecutive, then come Y, then Z, then E

    → Higher zip efficiency!
- Fine grained branches allow fain-grained I/O - read only members that are needed, instead of full object
- Supports STL containers, too!

Splitting (real example)



Split level = 0



Split level = 99

- In the TTreeViewer, tree->StartViewer(), the unsplit class is not breaken in pieces

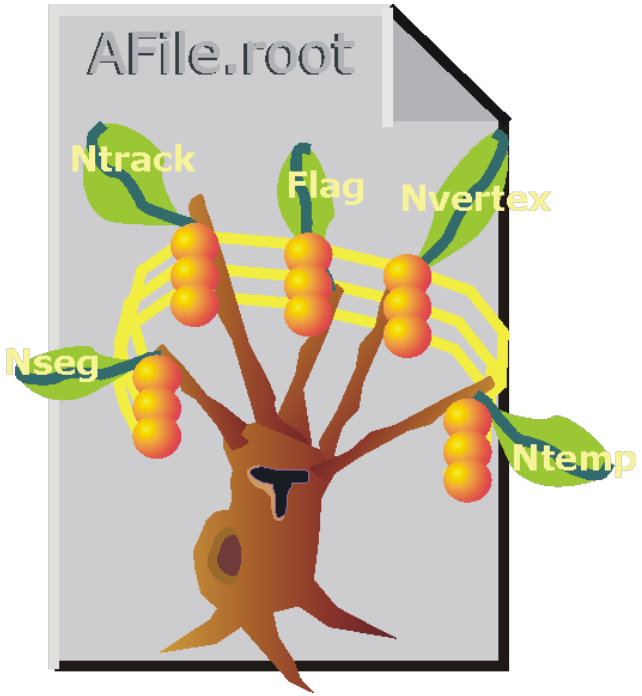
Splitting: Performance Considerations

A split branch is:

- Faster to read
  - The full entry does not have to be read each time
- Slower to write due to the large number of buffers

Step 4: Fill the Tree

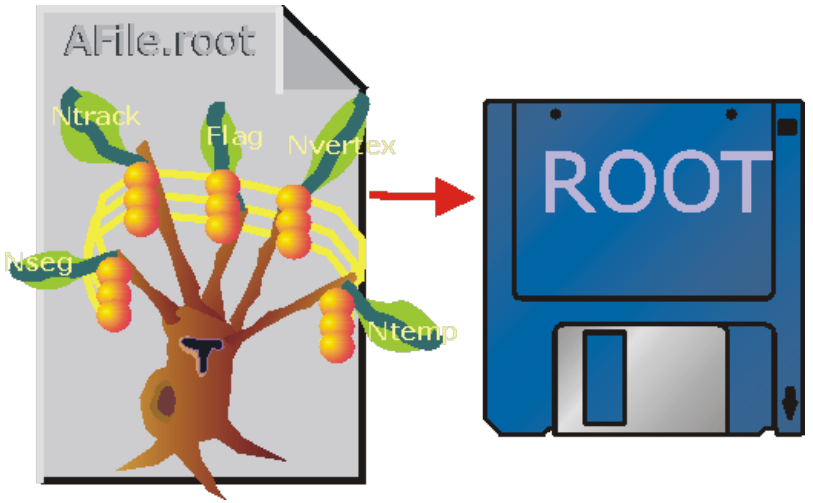
- Create a for loop
- Assign values to the object contained in each branch
- TTree::Fill() creates a new entry in the tree: snapshot of values of branches' objects




```
for (int e=0;e<100000;++e) {
   event->Build(e, 50); // fill event
   myTree->Fill();      // fill the tree
}
```

Step 5: Write Tree To File

```
myTree->Write();
```



Getting Event and instances

- Get eventclass/Event.h, .cxx

```
root[] .L eventclass/Event.cxx+
```

- Have a look at macros/CreateEventTree.C
  - The most relevant part shown in the next slide
- Going into details in the next module ...

Example macro

```
{
   Event *evt = new Event();
   TFile f("mytree.root");
   TTree *t = new TTree("myTree","A Tree");
   t->Branch("SplitEvent", &evt);
   // t->Branch("UnsplitEvent", &evt, 32000, 0);
   for (int e = 0; e < 1000; ++e) {
      evt->Build(e);
      t->Fill();
   }
   t->Write();
}
```

Reading a TTree

- Looking at a tree
- How to read a tree
- Trees, friends and chains

Looking at the Tree

TTree::Print() shows the data layout

```
root [] TFile f("AFile.root")
root [] myTree->Print();
******************************************************************************
*Tree    :myTree    : A ROOT tree                                              *
*Entries :       10 : Total =           867935 bytes  File  Size =     390138 *
*        :          : Tree compression factor =   2.72                       *
******************************************************************************
*Branch  :EventBranch                                                         *
*Entries :       10 : BranchElement (see below)                               *
*.............................................................................*
*Br    0 :fUniqueID :                                                         *
*Entries :       10 : Total  Size=        698 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      64000 bytes  Compression=   1.00     *
*.............................................................................*
…
…
```

TTree Selection Syntax

```
MyTree->Scan();
```

Prints the first 8 variables of the tree.

```
MyTree->Scan("*");
```

Prints all the variables of the tree.

Select specific variables:

```
MyTree->Scan("var1:var2:var3");
```

Prints the values of var1, var2 and var3.

A selection can be applied in the second argument:

```
MyTree->Scan("var1:var2:var3", "var1==0");
```

Prints the values of var1, var2 and var3 for the entries where var1 is exactly 0.

Looking at the Tree

TTree::Scan("leaf:leaf:….") shows the values

```
root [] myTree->Scan("fNseg:fNtrack"); > scan.txt

root [] myTree->Scan("fEvtHdr.fDate:fNtrack:fPx:fPy","",
                     "colsize=13 precision=3 col=13:7::15.10");
```

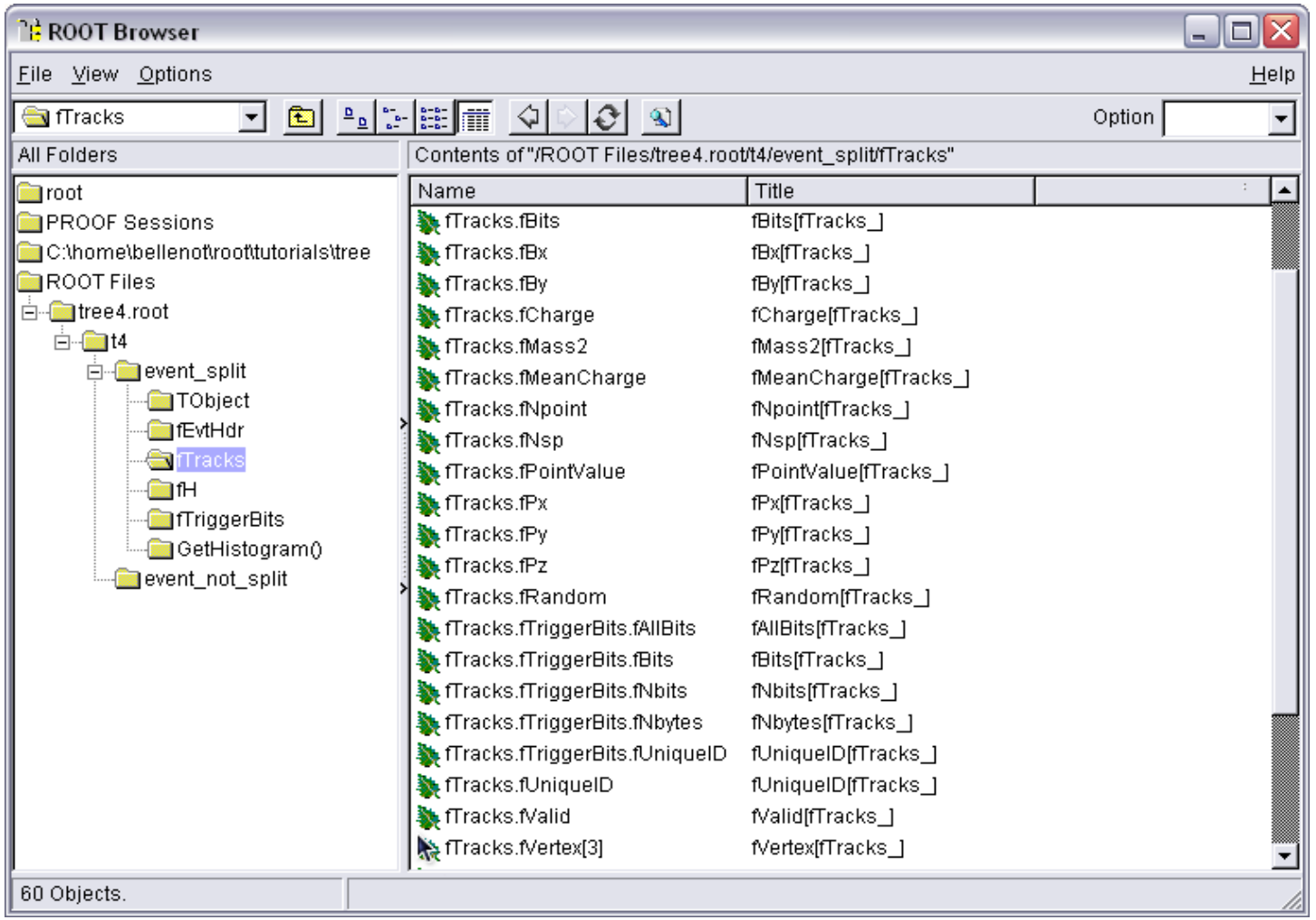
```
************************************************************************
* Row * Instance * fEvtHdr.fDate * fNtrack *           fPx *            fPy *
************************************************************************
*   0 *        0 *        960312 *     594 *          2.07 *    1.459911346 *
*   0 *        1 *        960312 *     594 *         0.903 *   -0.4093382061 *
*   0 *        2 *        960312 *     594 *         0.696 *    0.3913401663 *
*   0 *        3 *        960312 *     594 *        -0.638 *    1.244356871 *
*   0 *        4 *        960312 *     594 *        -0.556 *   -0.7361358404 *
*   0 *        5 *        960312 *     594 *         -1.57 *   -0.3049036264 *
*   0 *        6 *        960312 *     594 *        0.0425 *    -1.006743073 *
*   0 *        7 *        960312 *     594 *          -0.6 *    -1.895804524 *
```

Looking at the Tree

TTree::Show(entry_number) shows the values for one entry

```
root [] myTree->Show(0);
======> EVENT:0
 EventBranch     = NULL
 fUniqueID       = 0
 fBits           = 50331648
 [...]
 fNtrack         = 594
 fNseg           = 5964
 [...]
 fEvtHdr.fRun    = 200
 [...]
 fTracks.fPx     = 2.066806, 0.903484, 0.695610, -0.637773,...
 fTracks.fPy     = 1.459911, -0.409338, 0.391340, 1.244357,...
```

Browsing the tree structure



Unsplits on the file

How To Read a TTree

Example:

1. Open the TFile

```
TFile *f = TFile::Open("tree4.root")
```

1. Get the TTree

```
TTree *t4 = (TTree *) f->GetObject("t4")
```

How to Read a TTree

3. Create a variable pointing to the data

```
root [] Event *event = 0;
```

4. Associate a branch with the variable:

```
root [] t4->SetBranchAddress("event_split", &event);
```

5. Read one entry in the TTree

```
root [] t4->GetEntry(0)
root [] event->GetTracks()->First()->Dump()
==> Dumping object at: 0x0763aad0, name=Track,
   class=Track
fPx                0.651241     X component of the momentum
fPy                1.02466      Y component of the momentum
fPz                1.2141       Z component of the momentum
[...]
```

Example macro

```
{
   Event *evt = 0;
   TFile f("mytree.root");
   TTree *myTree = (TTree*)f->Get("myTree");
   myTree->SetBranchAddress("Event", &evt);
   for (int e=0; e<1000; ++e) {
      myTree->GetEntry(e);
      evt->Analyse();
   }
}
```

TChain

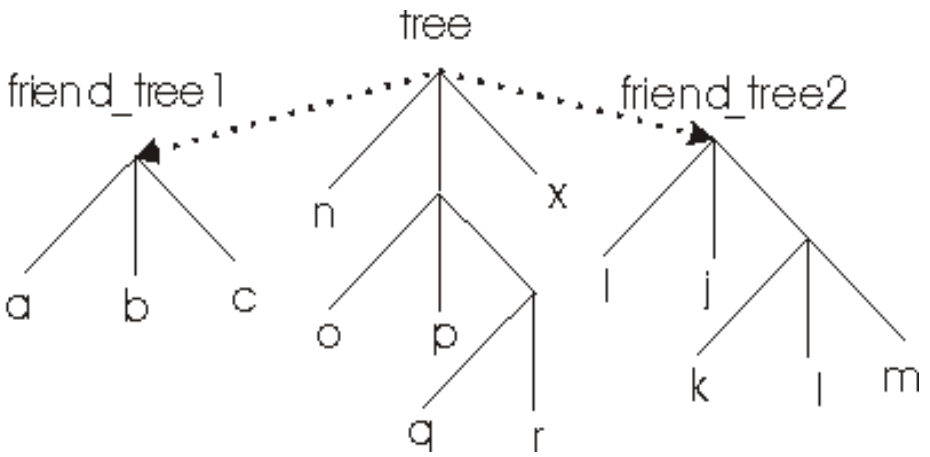
- Collection of ROOT files containing the same tree
- Same semantics as TTree

As an example, assume we have three files called file1.root, file2.root, file3.root. Each contains tree called "T". Create a chain:

```
   TChain chain("T"); // argument: tree
name
   chain.Add("file1.root");
   chain.Add("file2.root");
   chain.Add("file3.root");
```

Now we can use 'chain' like a TTree!

Friends of Trees

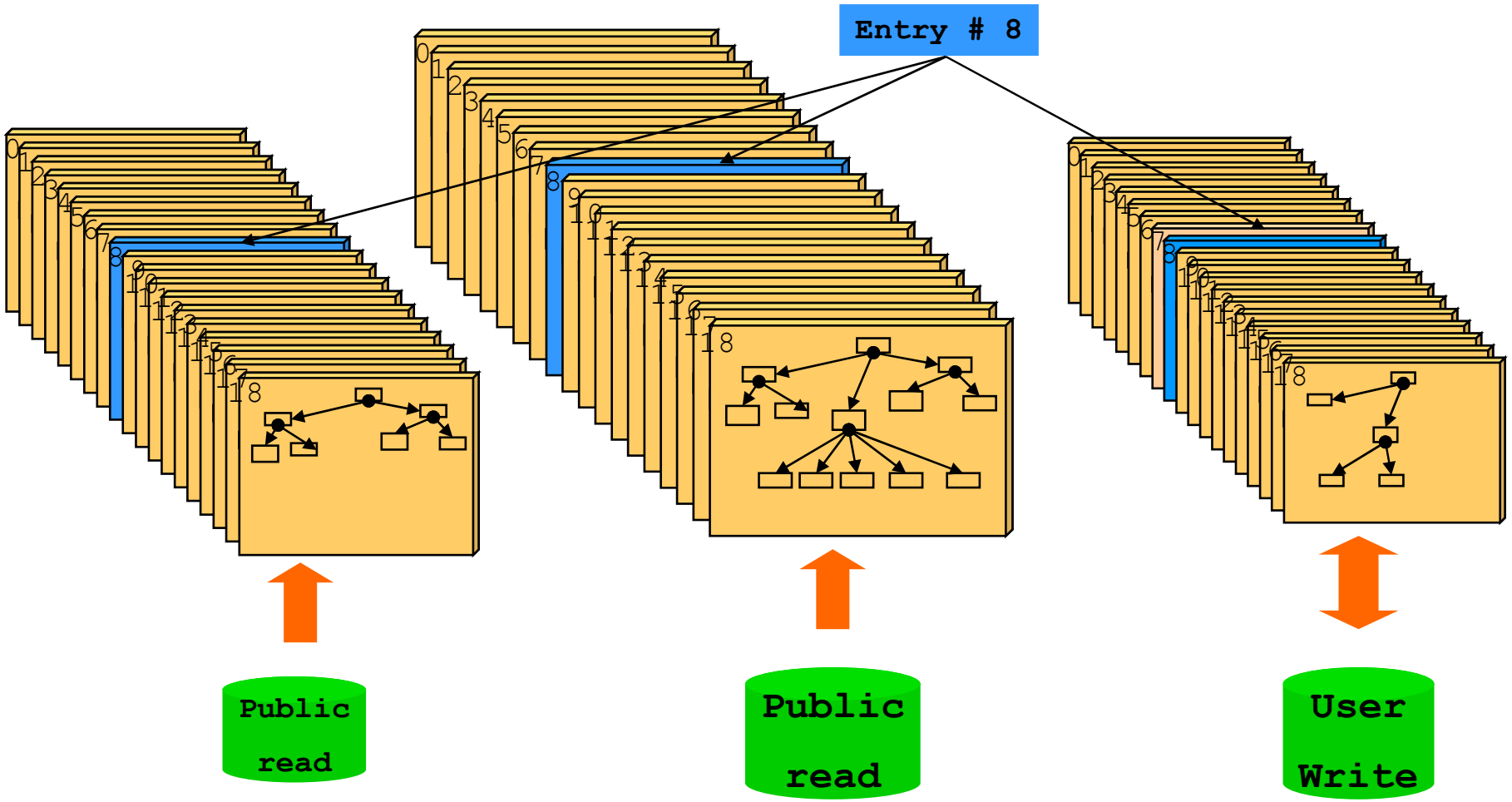


- Adding new branches to existing tree without touching it, i.e.:

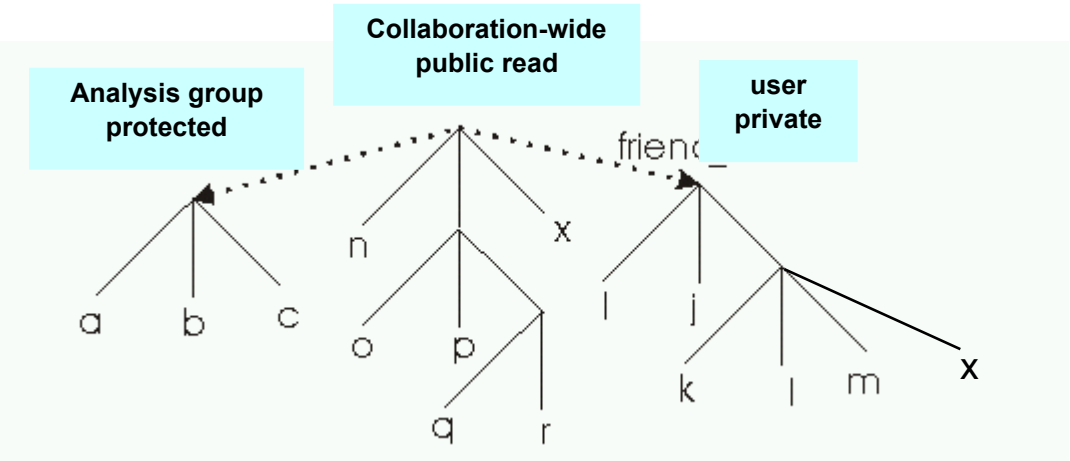
```
myTree->AddFriend("ft1", "friend.root")
```

- Unrestricted access to the friend's data via virtual branch of original tree

Tree Friends



Tree Friends



Processing time independent of the number of friends unlike table joins in RDBMS

```
TFile f1("tree1.root");
tree.AddFriend("tree2", "tree2.root")
tree.AddFriend("tree3", "tree3.root");
tree.Draw("x:a", "k<c");
tree.Draw("x:tree2.x", "sqrt(p)<b");
```

To know more

User's Guide, Tutorials, HowTo's:

http://root.cern.ch

Reference Guide (full class documentation):

http://root.cern.ch/root/html