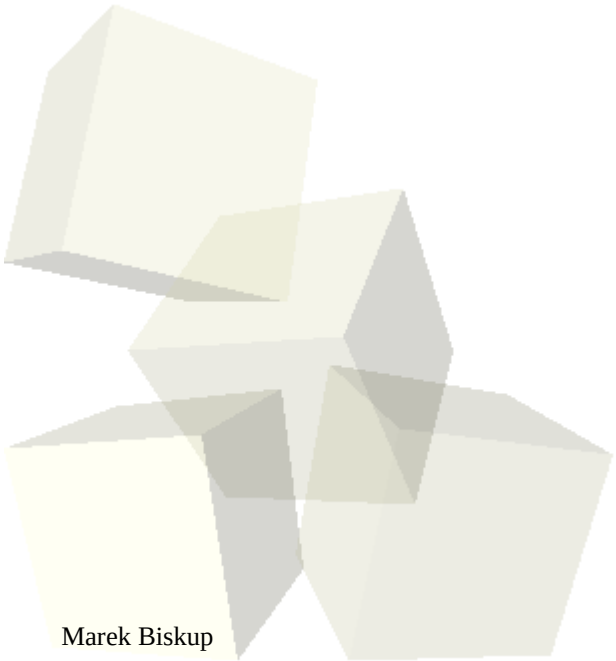
PROOF GUI and API/Selectors

Marek Biskup, CERN



PROOF GUI and API/Selectors

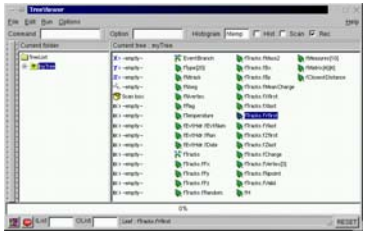
or rather:

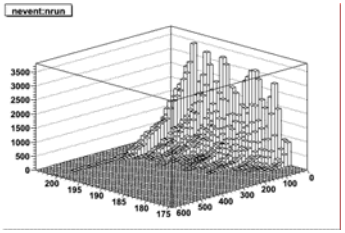
How to use PROOF

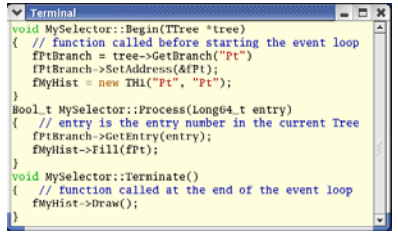
Marek Biskup, CERN

Outline

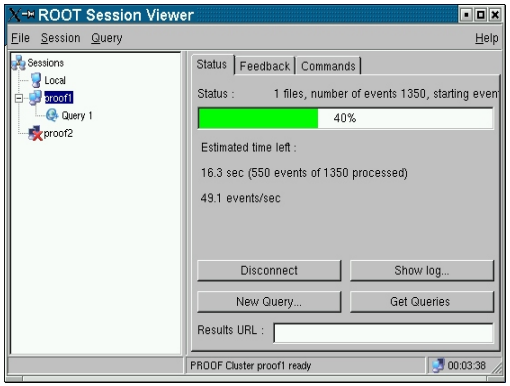
- Data analysis with PROOF
  - TreeViewer
  - Chain.Draw()
  - Selectors



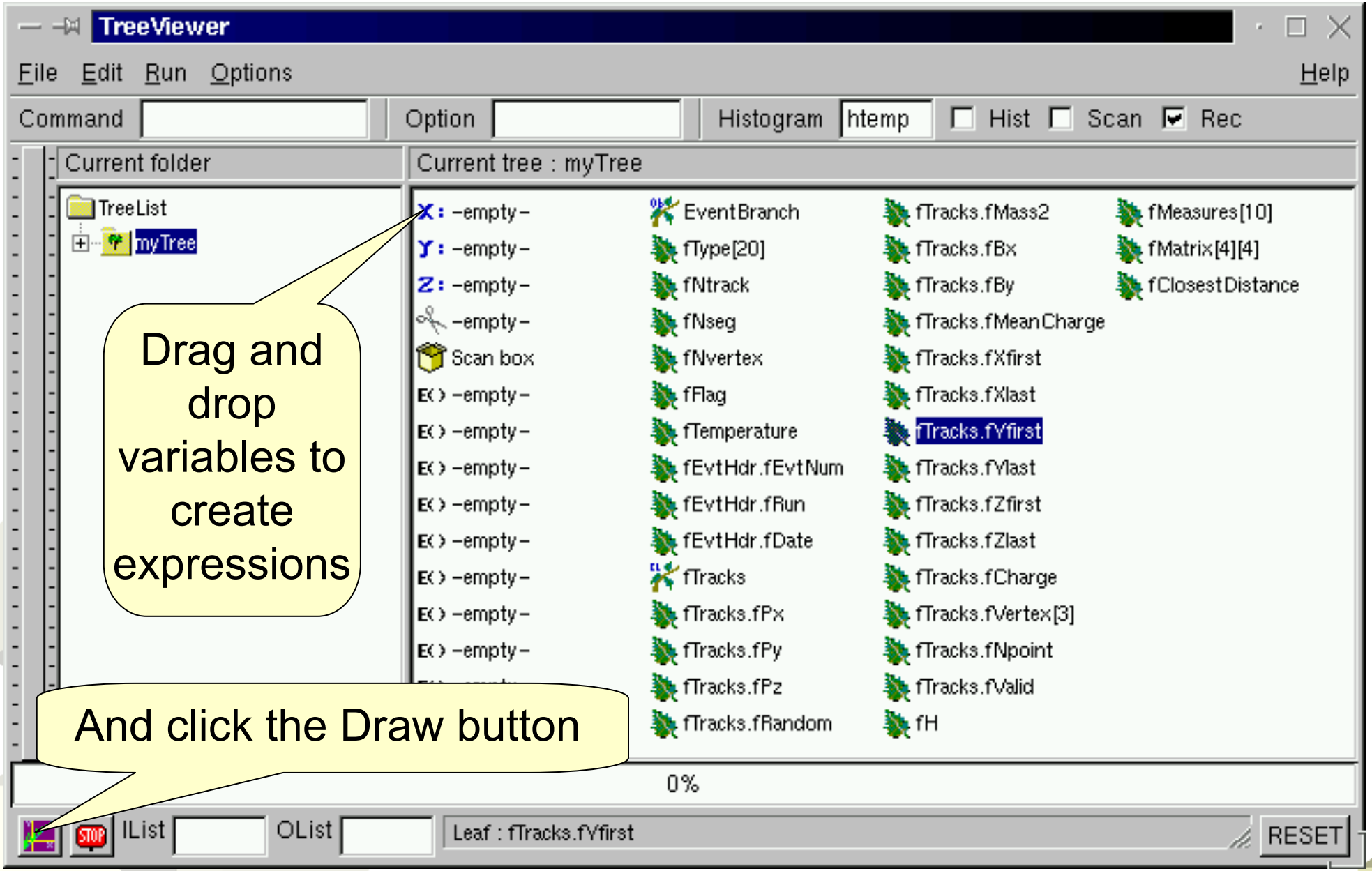




- PROOF GUI: the SessionViewer

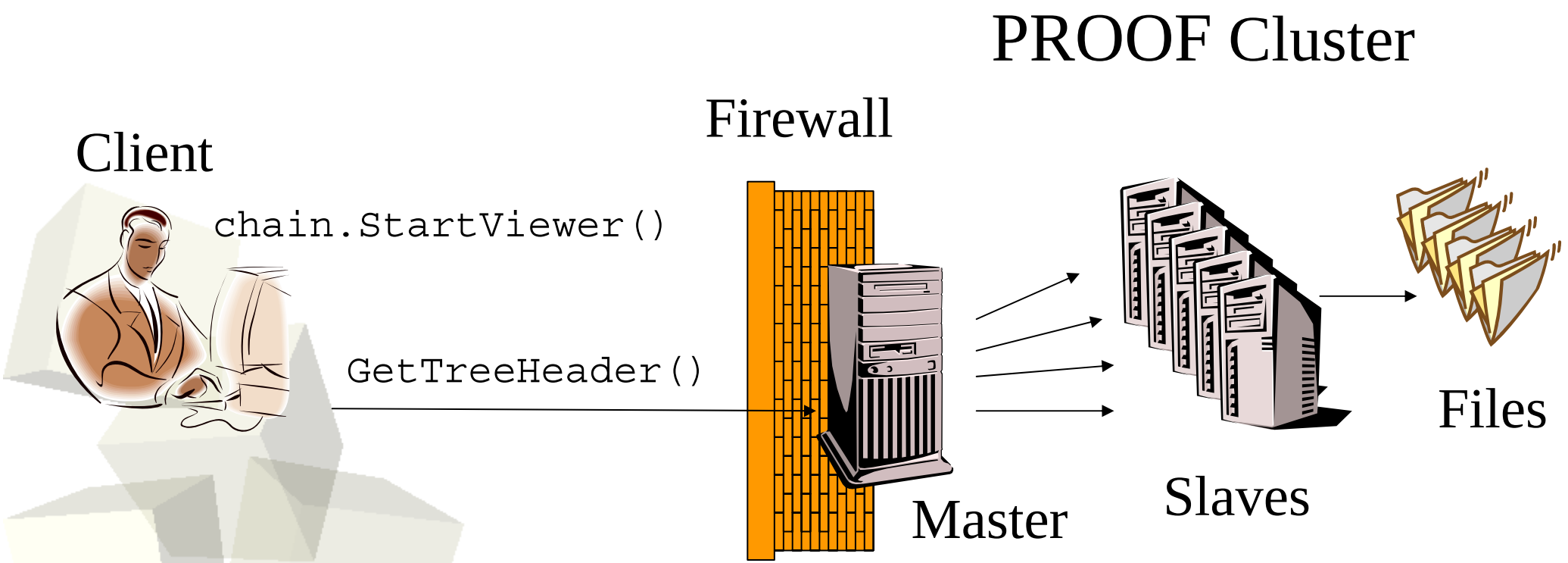


Tree Viewer


TreeViewer
File Edit Run Options Help
Command Option Histogram htemp Hist Scan Rec
Current folder
Current tree : myTree
TreeList
myTree
X: -empty-
Y: -empty-
Z: -empty-
-empty-
Scan box
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
EventBranch
fType[20]
fNtrack
fNseg
fNvertex
fFlag
fTemperature
fEvtHdr.fEvtNum
fEvtHdr.fRun
fEvtHdr.fDate
fTracks
fTracks.fPx
fTracks.fPy
fTracks.fPz
fTracks.fRandom
fTracks.fMass2
fTracks.fBx
fTracks.fBy
fTracks.fMeanCharge
fTracks.fXfirst
fTracks.fXlast
fTracks.fYfirst
fTracks.fYlast
fTracks.fZfirst
fTracks.fZlast
fTracks.fCharge
fTracks.fVertex[3]
fTracks.fNpoint
fTracks.fValid
fH
fMeasures[10]
fMatrix[4][4]
fClosestDistance
Drag and drop variables to create expressions
And click the Draw button
0%
STOP
IList
OList
Leaf : fTracks.fYfirst
RESET

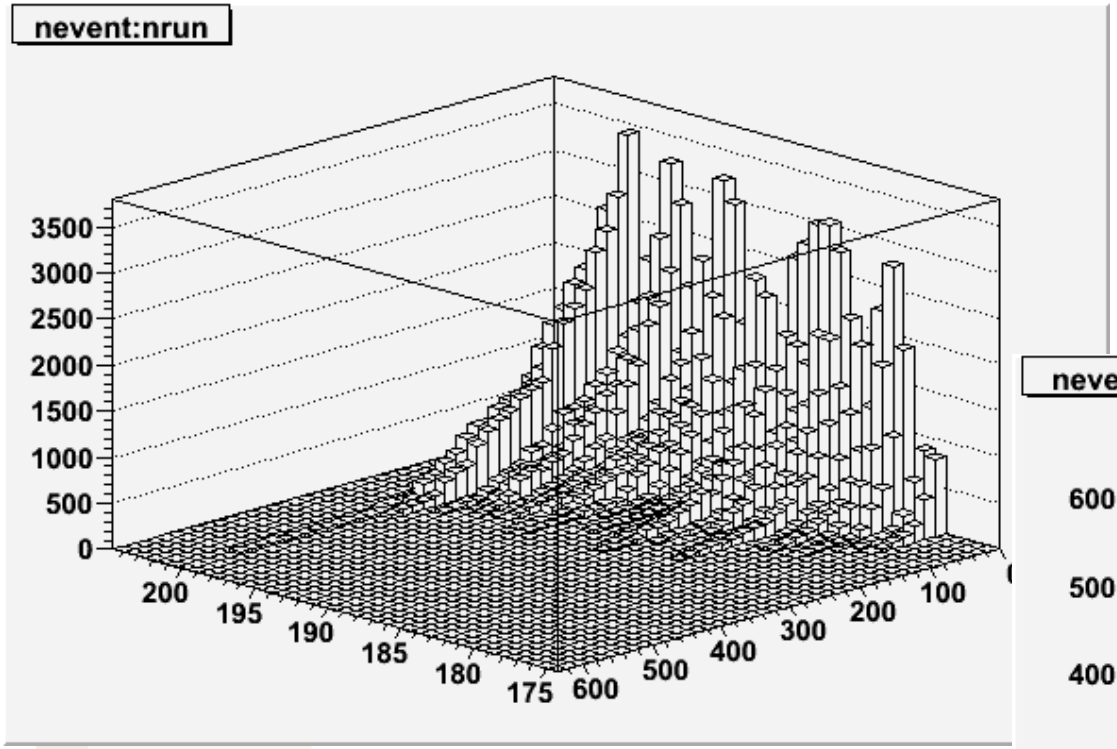
The tree header is fetched from the PROOF master

Works even if files are inaccessible directly

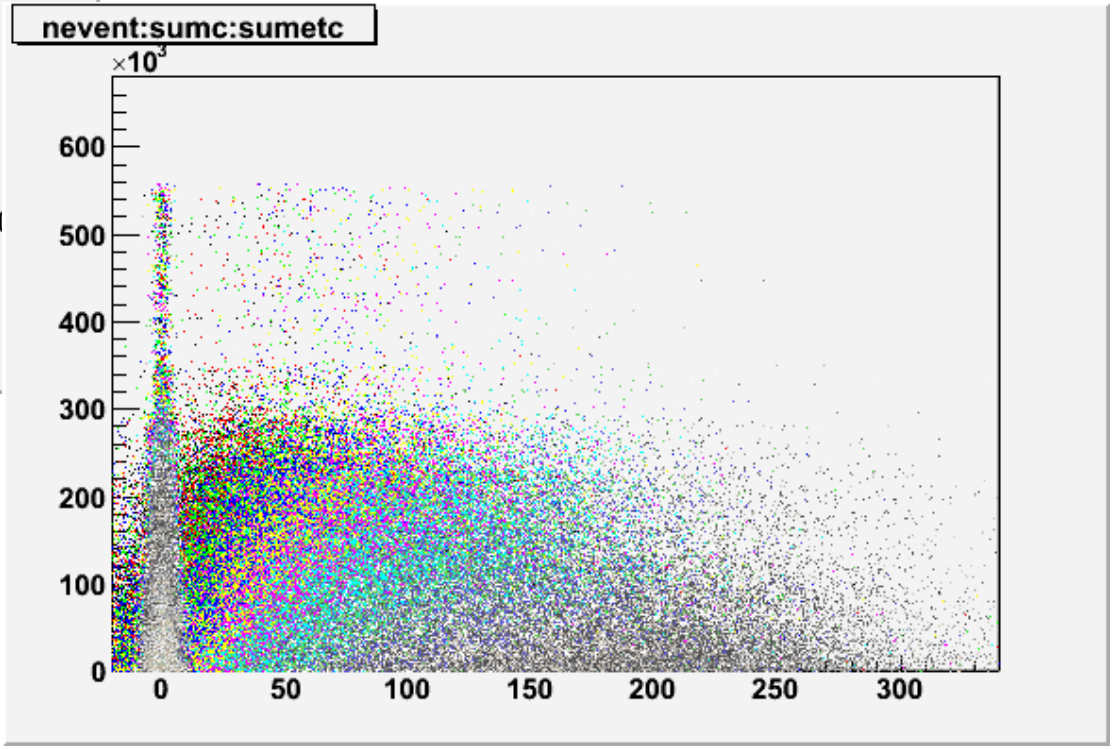


Chain.Draw()

chain.Draw() is a function called by the GUI for drawing



chain.Draw(“nevent:nrun”, “”, “lego”);



Processing done by PROOF if there is an active proof session

chain.Draw(“sumetc:nevent:nrun”, “”, “col”);

Basics of Selectors

Selectors contain functions important for processing

- Preprocessing and initialization
- Processing each event
- Post processing and clean-up

Entries are processed in an arbitrary order

Skeleton can be generated from a Tree

Only the needed data is read

Terminal

```
void MySelector::Begin(TTree *tree)
{  // function called before starting the event loop
   fPtBranch = tree->GetBranch("Pt");
   fPtBranch->SetAddress(&fPt);
   fMyHist = new TH1("Pt", "Pt");
}
Bool_t MySelector::Process(Long64_t entry)
{   // entry is the entry number in the current Tree
   fPtBranch->GetEntry(entry);
   fMyHist->Fill(fPt);
}
void MySelector::Terminate()
{   // function called at the end of the event loop
   fMyHist->Draw();
}
```

But life is more complicated:

- Many computers to initialize and clean-up

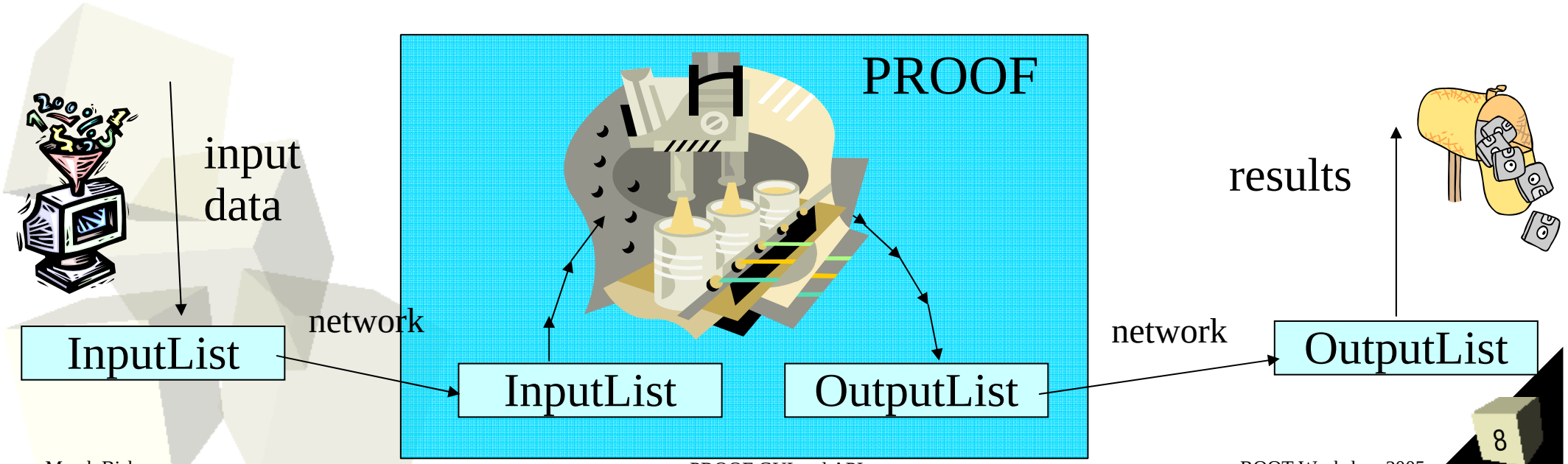
```
TSelector::SlaveBegin();
TSelector::SlaveTerminate();
```

- Many trees in a chain

```
TSelector::Init(TTree*)
```

- Input and output results should be transparently sent over network

```
TList* fInput, fOutput;
```



PROOF and Selectors

Terminal

```
void MySelector::Begin(TTree *tree)
{// called on the client before processing
}
void MySelector::SlaveBegin(TTree *tree)
{// called on each slave before processing
   fMyHist = new TH1F("Pt", "Pt");
   fOutput->Add(fMyHist);
}
void MySelector::Init(TTree* tree)
{// called  each time a tree is changed
   fPtBranch = tree->GetBranch("Pt")
   fPtBranch->SetAddress(&fPt);
}
Bool_t MySelector::Process(Long64_t entry)
{// called on each slave for their entries
   fPtBranch->GetEntry(entry);
   fMyHist->Fill(fPt);
}
void MySelector::SlaveTerminate()
{// called on each slave after processing
}
void MySelector::Terminate()
{// called on the client after processing
   fMyHist = (TH1F*)fOutput->FindObject("Pt");
   fMyHist->Draw();
}
```

Client

Slaves

Slaves

Slaves

Client

Initialize each slave

Many Trees are being processed

The same code works also without PROOF (of course!)

No user's control on the order

Selectors - summary

- Skeletons generated from a tree
- Only methods need to be filled
- Simplify programs' structure
- Can be used for parallel processing as well as for local analysis

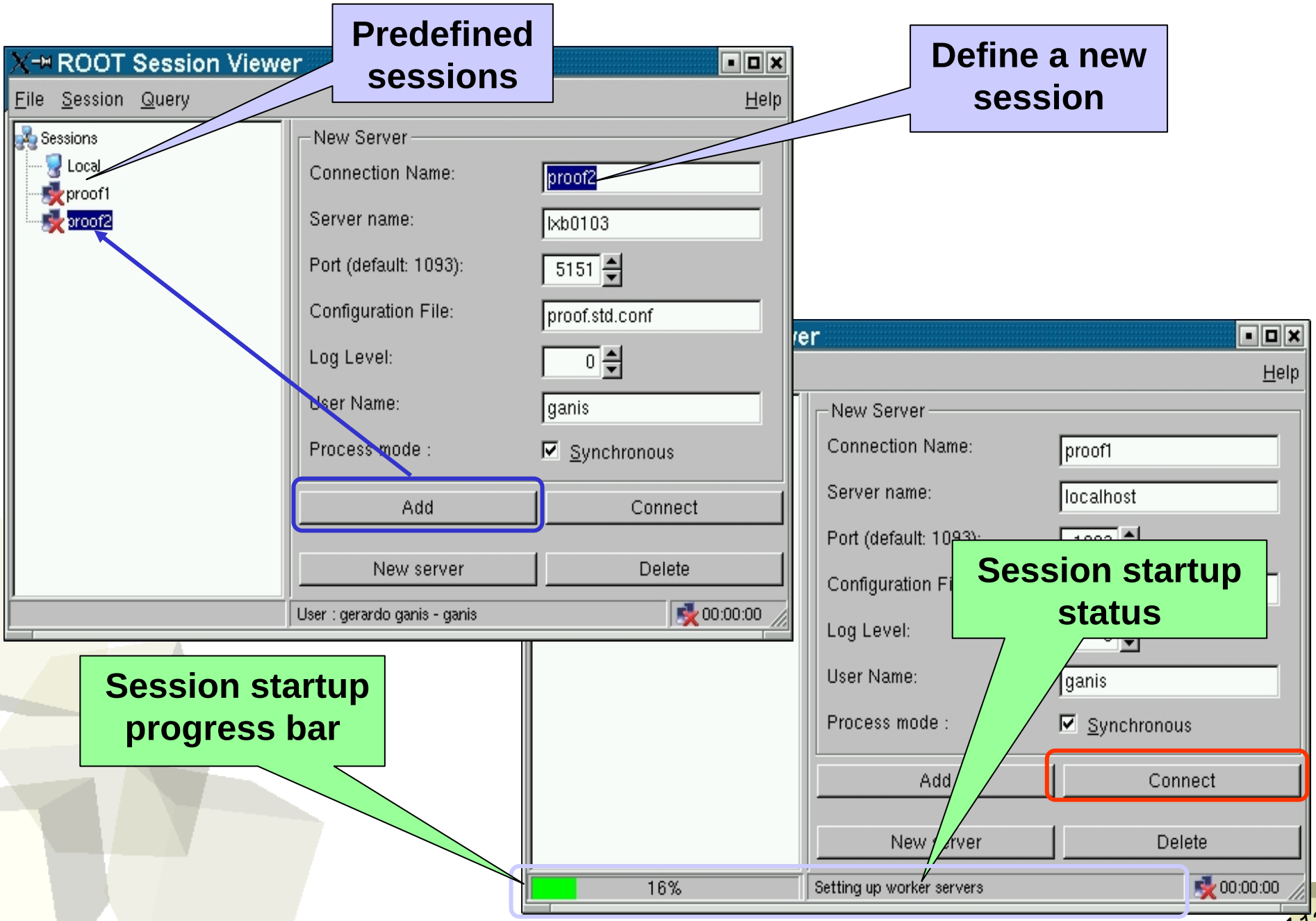
Terminal

```
void MySelector::Begin(TTree *tree)
{// called on the client before processing
}
void MySelector::SlaveBegin(TTree *tree)
{// called on each slave before processing
   fMyHist = new TH1F("Pt", "Pt");
   fOutput->Add(fMyHist);
}
void MySelector::Init(TTree* tree)
{// called  each time a tree is changed
   fPtBranch = tree->GetBranch("Pt")
   fPtBranch->SetAddress(&fPt);
}
Bool_t MySelector::Process(Long64_t entry)
{// called on each slave for their entries
   fPtBranch->GetEntry(entry);
   fMyHist->Fill(fPt);
}
void MySelector::SlaveTerminate()
{// called on each slave after processing
}
void MySelector::Terminate()
{// called on the client after processing
   fMyHist = (TH1F*)fOutput->FindObject("Pt");
   fMyHist->Draw();
}
```

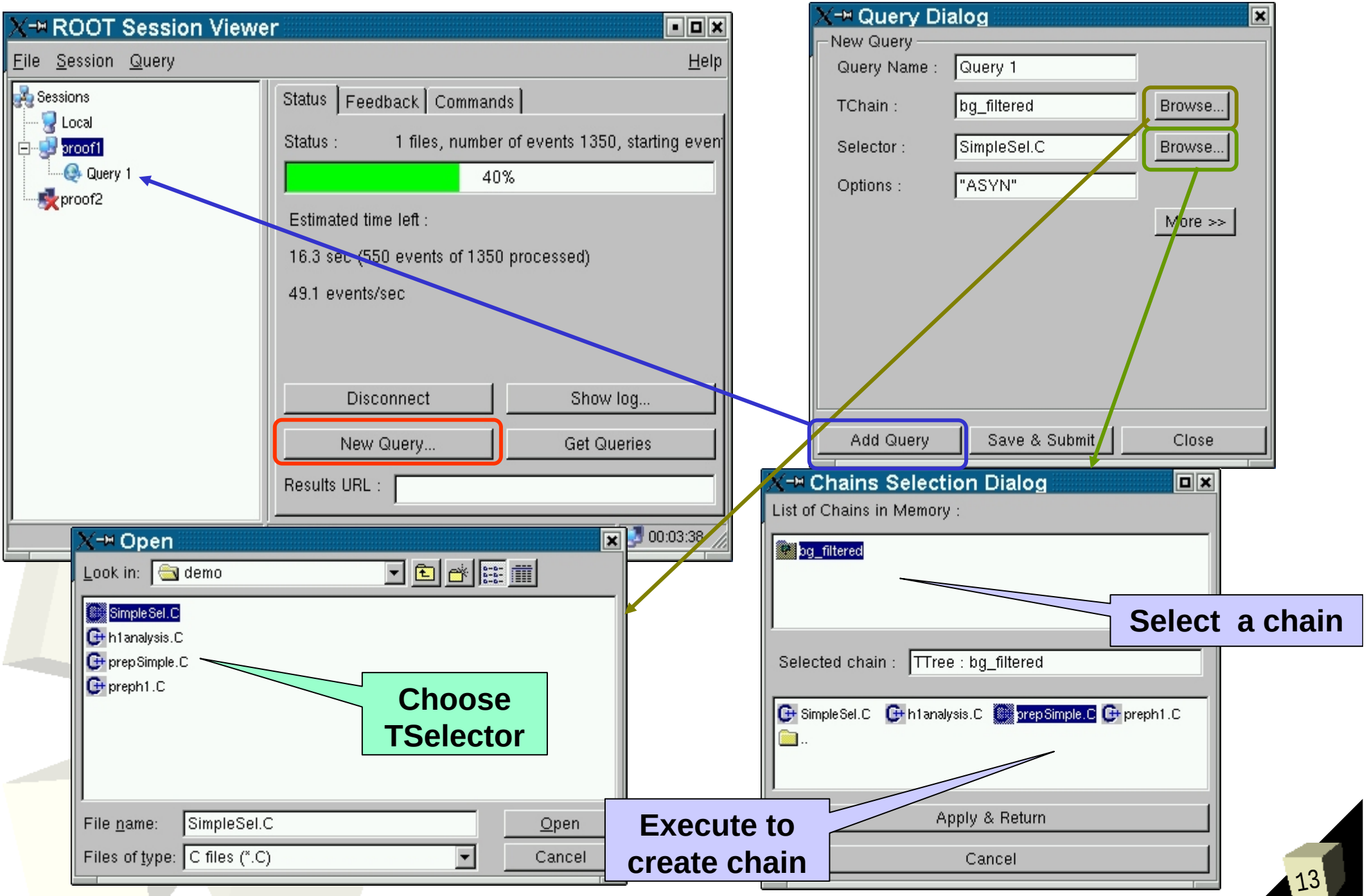
The Session Viewer

- Allows full *on-click* control on everything
  - define a new session, choose a predefined one
  - submit a query, execute a command
  - query editor
    - execute macro to define a **TChain** or pick one up from the existing ones
    - browse directories with selectors
  - online monitoring of feedback histograms
  - browse folders with results of query
  - retrieve, delete, archive functionality
  - start viewer for fast **TChain** browsing

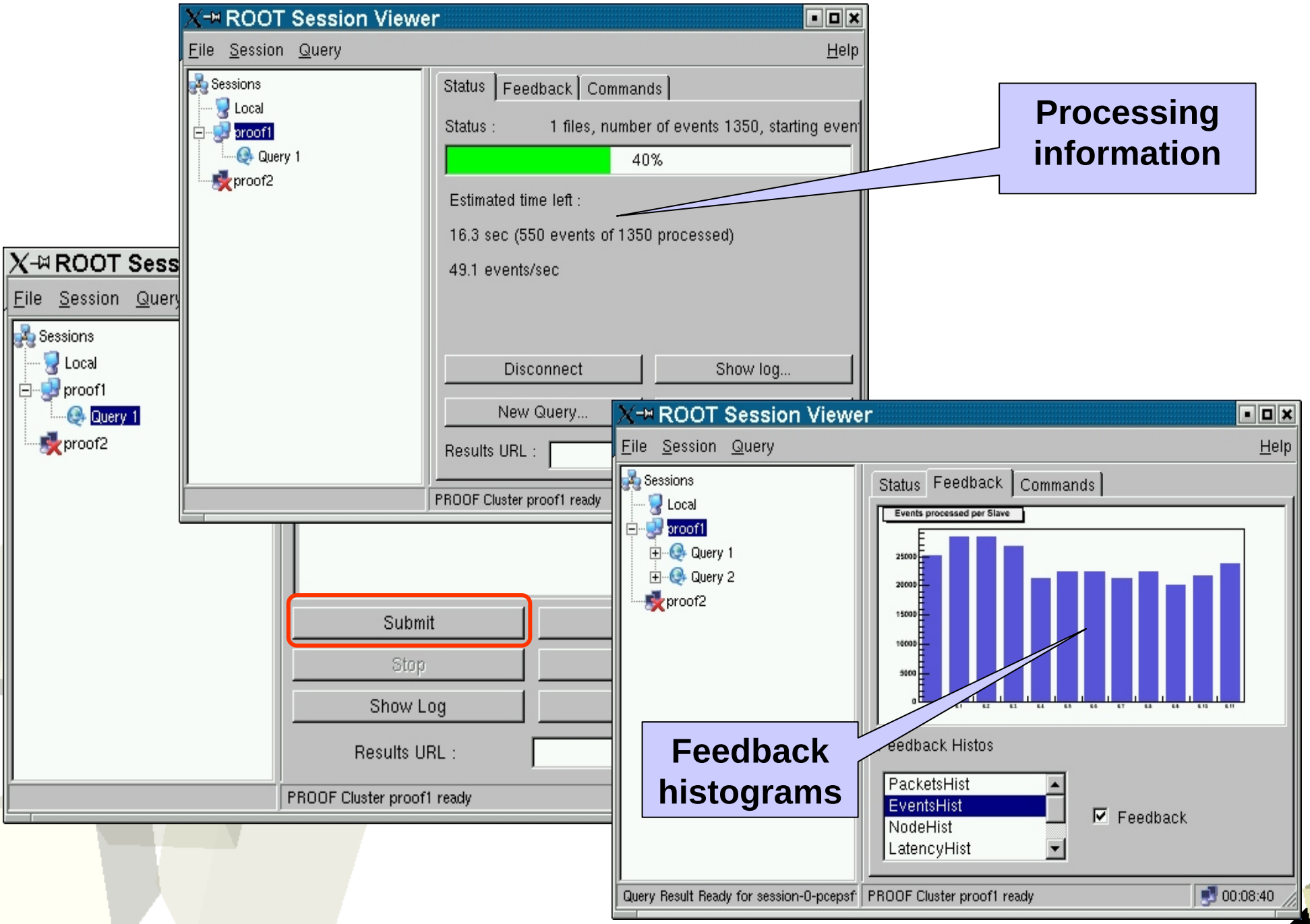
Session definition and connection



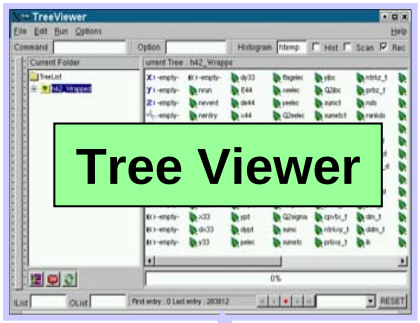
Query definition

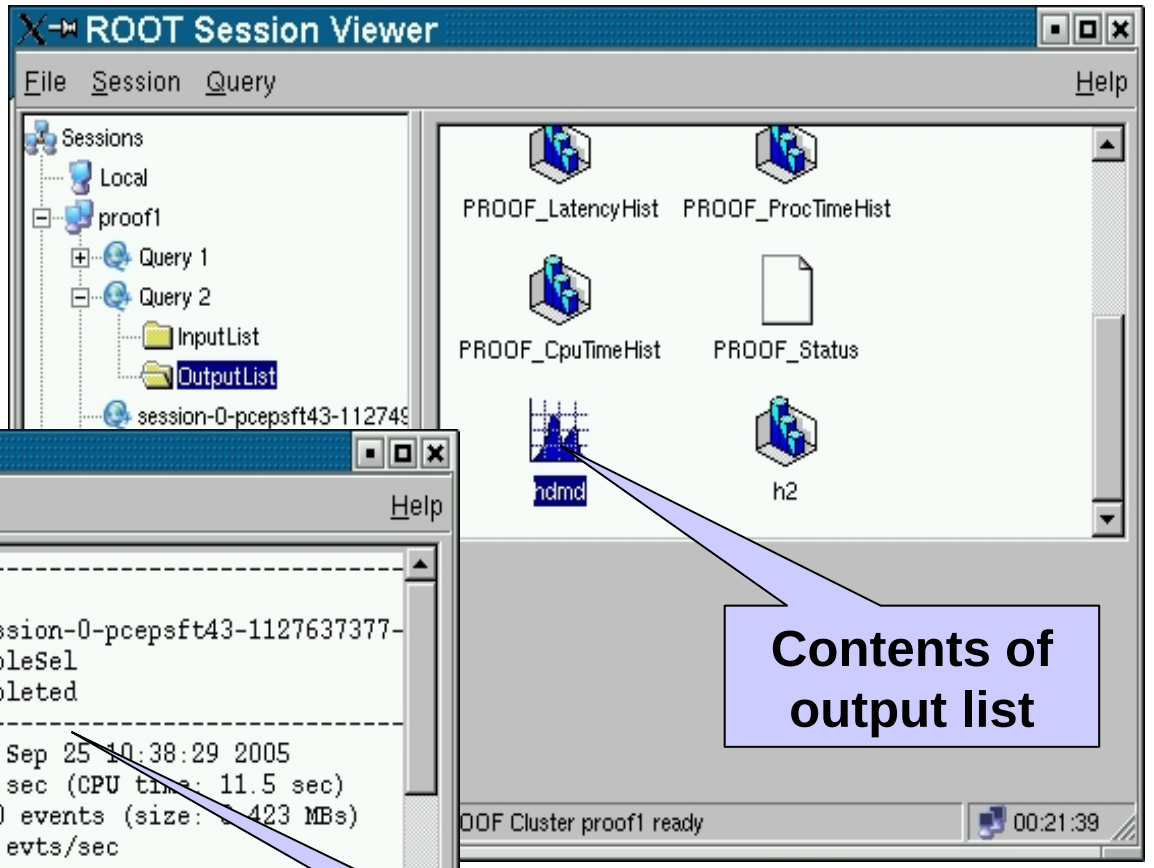


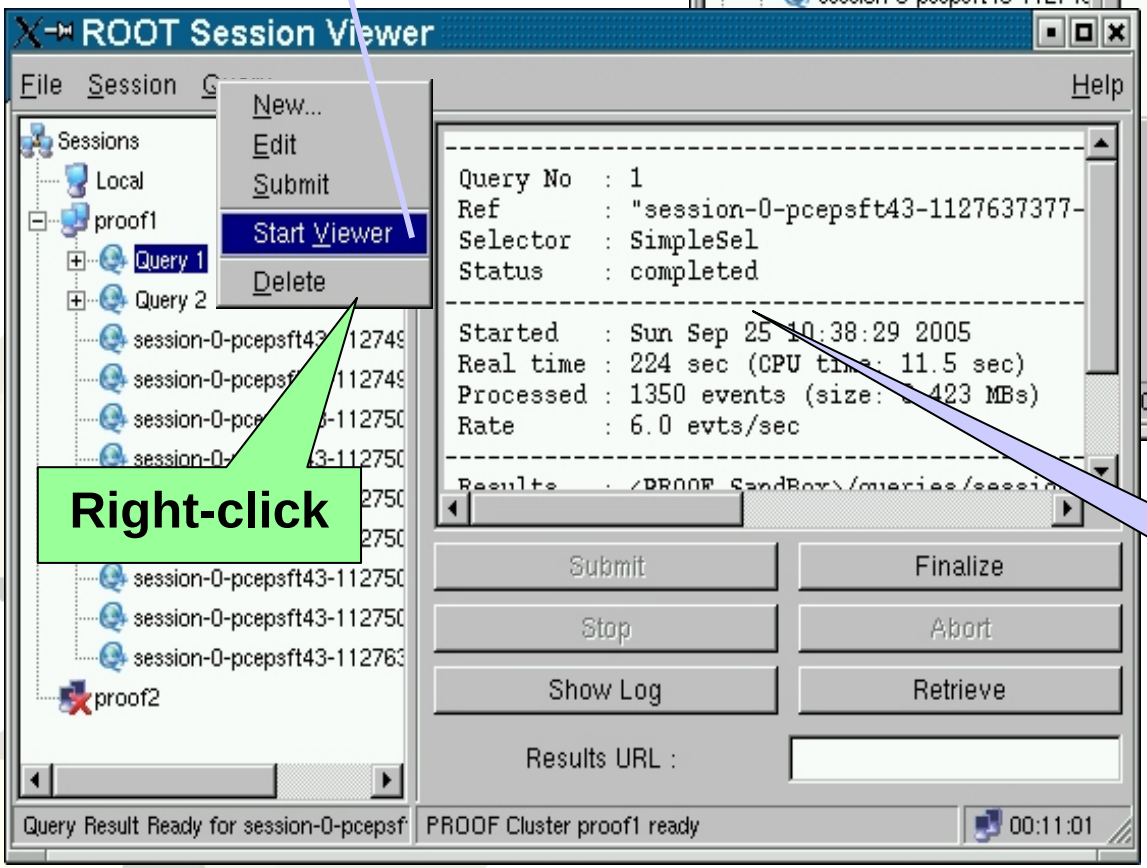
Query processing



Query browsing

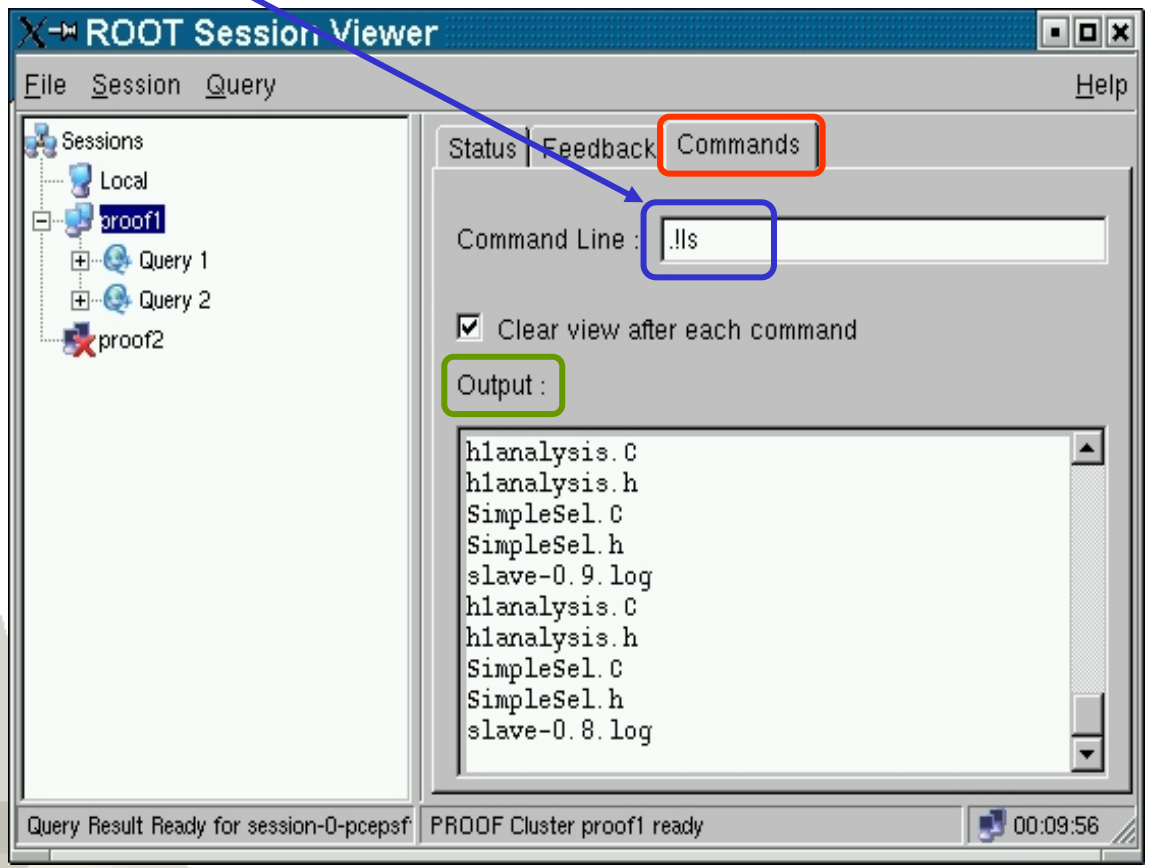






Command execution

- **`ProcessLine`** functionality
  - interface to **`gProof->Exec("<ROOT directive>")`**
  - **e.g.,** **`ls`**



Summary

- PROOF uses the same analysis methods:
  - TreeViewer
  - Chain.Draw()
  - Selectors
- But selectors have to be written carefully with distributed processing kept in mind
  - SlaveBegin() and SlaveTerminate()
  - Input List, Output List
- Session Viewer helps to organize
  - Sessions (local and PROOF)
  - Queries
  - Results
- And monitor queries being processed