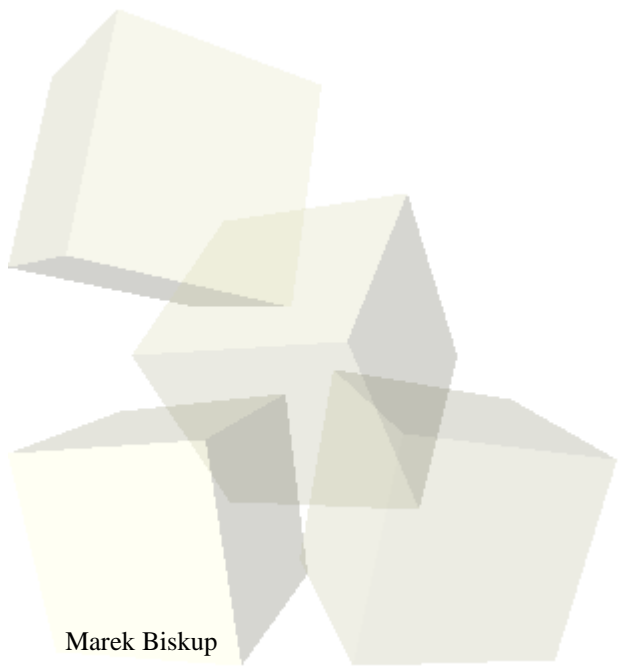




# PROOF GUI and API/Selectors

Marek Biskup, CERN



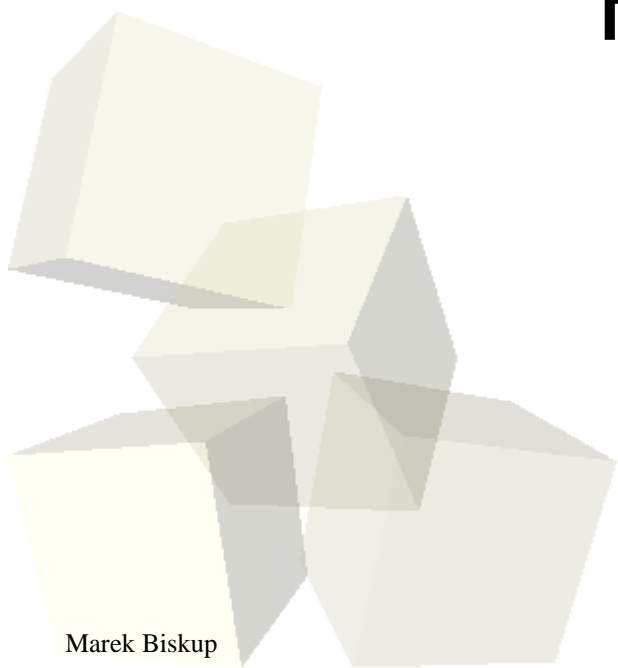


# PROOF GUI and API/Selectors

or rather:

## How to use PROOF

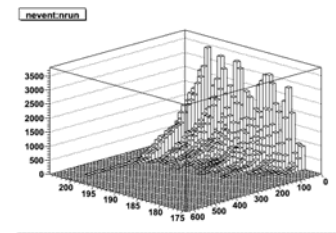
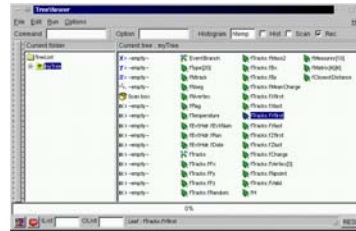
Marek Biskup, CERN





## ■ Data analysis with PROOF

- TreeViewer
- Chain.Draw()
- Selectors



```

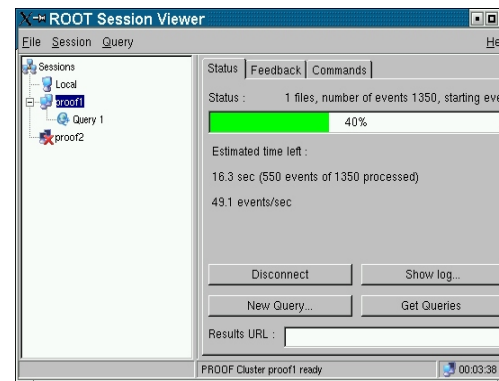
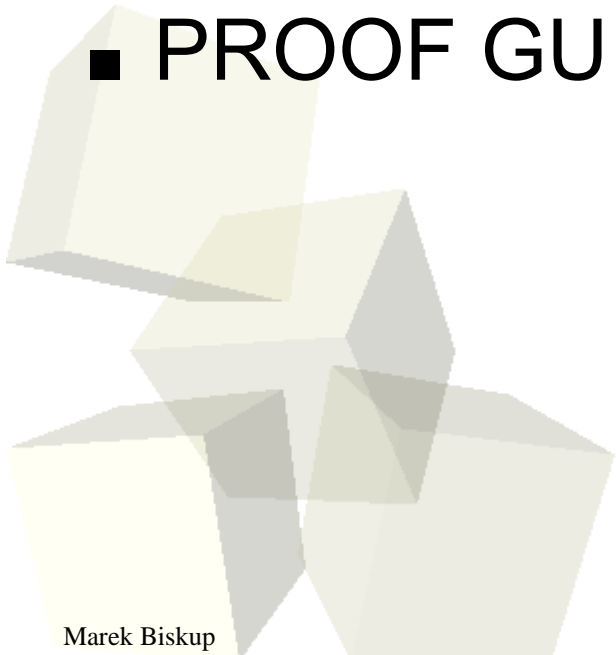
Terminal
void MySelector::Begin(TTree *tree)
{
  // function called before starting the event loop
  fPtBranch = tree->GetBranch("Pt");
  fPtBranch->SetAddress(&fPt);
  fMyHist = new TH1("Pt", "Pt");
}

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();
}

```

## ■ PROOF GUI: the SessionViewer





# Tree Viewer

TreeViewer

File Edit Run Options Help

Command Option Histogram htemp Hist Scan Rec

Current folder

- TreeList
- myTree

Current tree : myTree

- X: -empty-
- Y: -empty-
- Z: -empty-
- empty-
- Scan box
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -empty-
- E<> -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

0%

IList OList Leaf : fTracks.fYfirst RESET

Drag and drop variables to create expressions

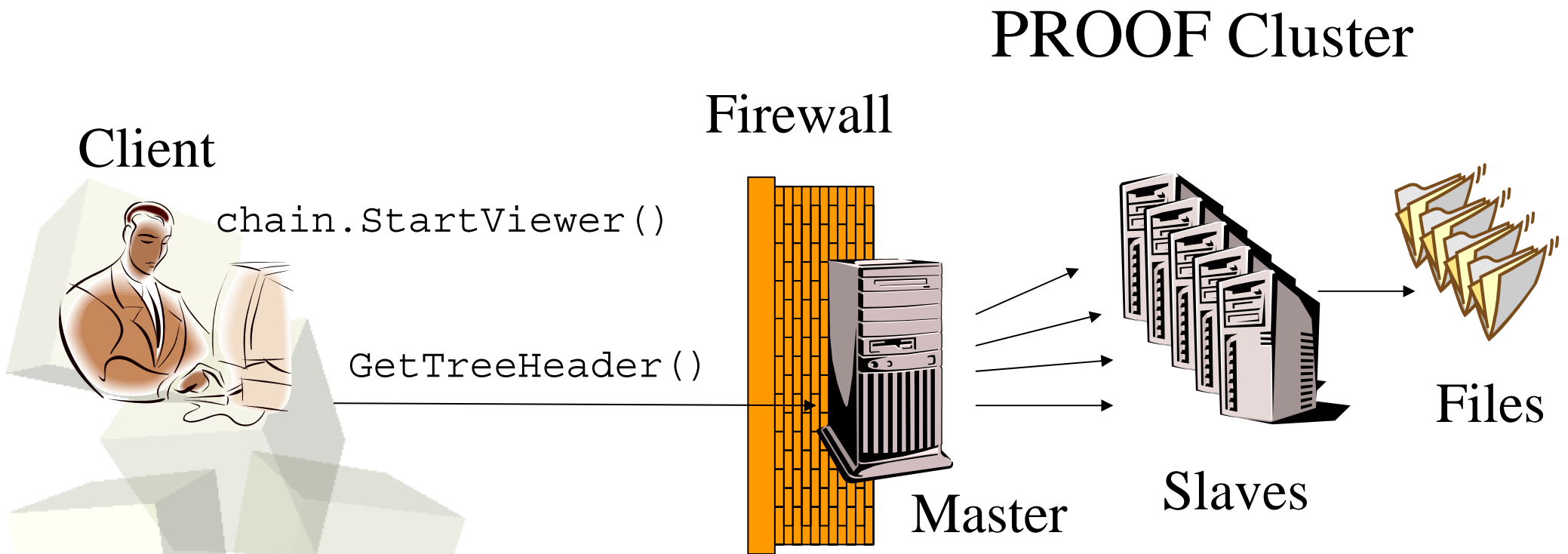
And click the Draw button



# TreeViewer with PROOF

The tree header is fetched from the PROOF master

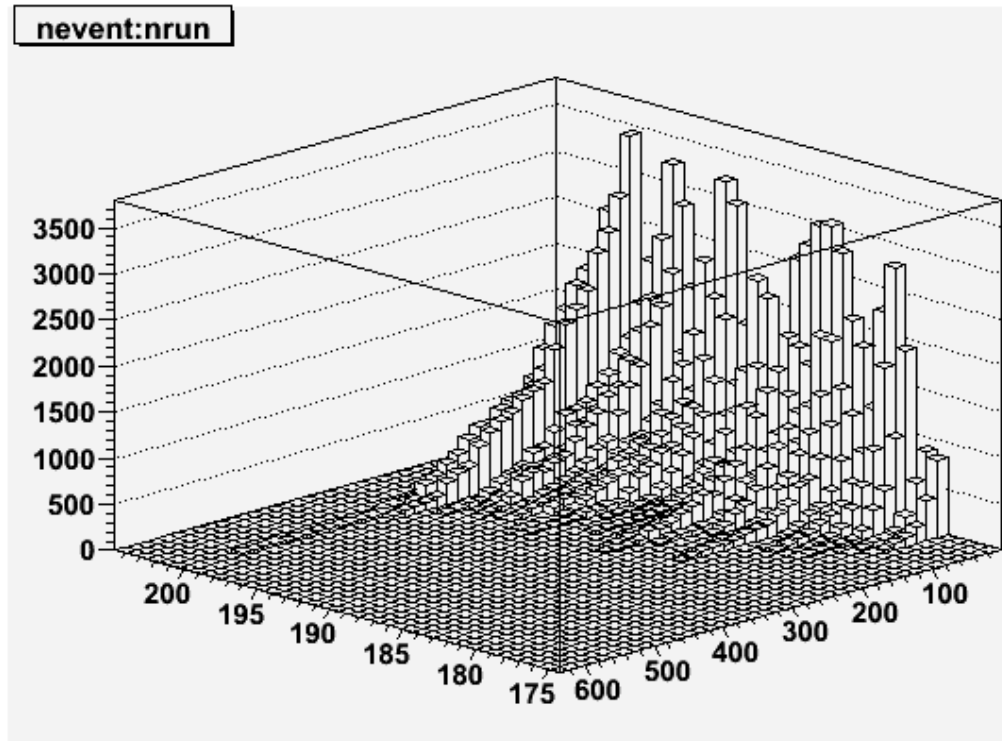
Works even if files are inaccessible directly



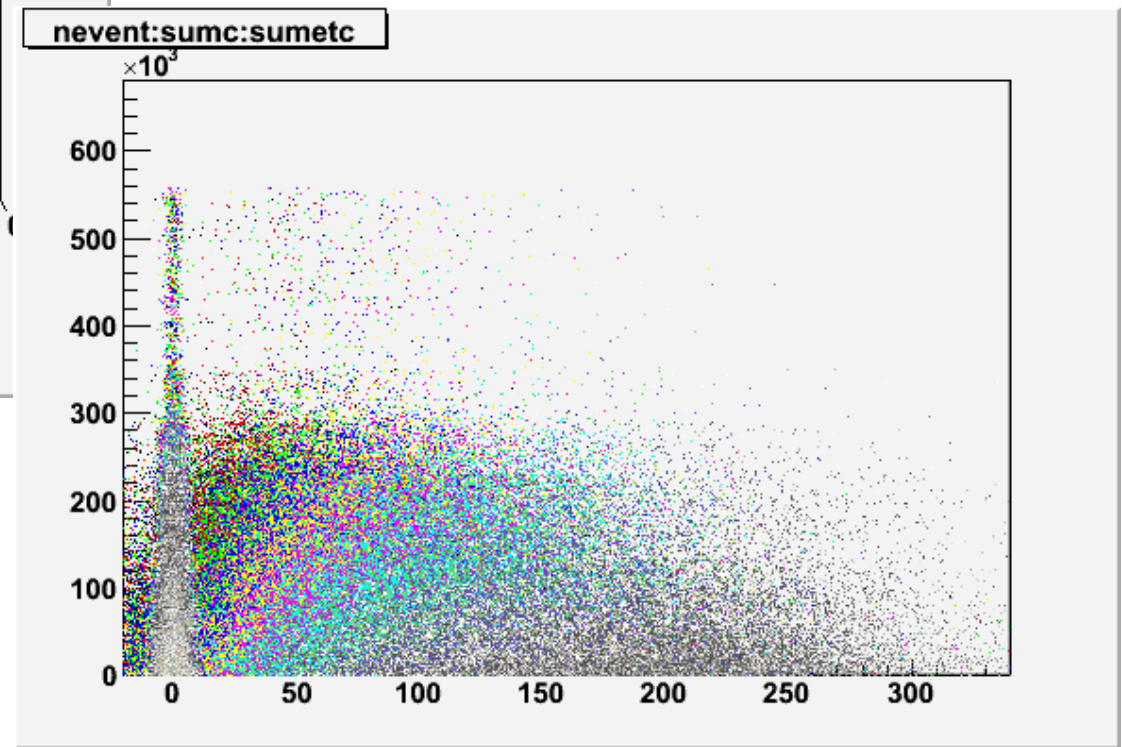


# Chain.Draw()

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



```
chain.Draw( "nevent:nrun", "", "lego");
```



```
chain.Draw( "sumetc:nevent:nrun", "", "col");
```

Processing done by PROOF if there is an active proof session

Selectors contain functions important for processing

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

Entries are processed in an arbitrary order

```
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();
}
```

Skeleton can be generated from a Tree

Only the needed data is read



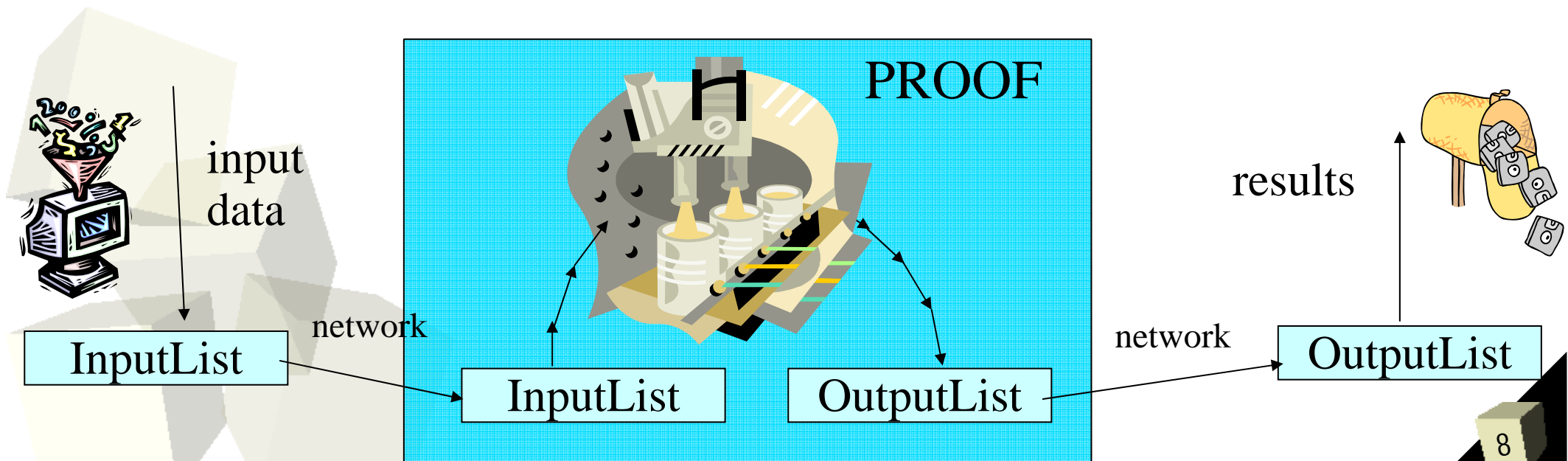
But life is more complicated:

- Many computers to initialize and clean-up
- Many trees in a chain
- Input and output results should be transparently sent over network

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

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

```
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

Initialize each slave

Slaves

Many Trees are being processed

Slaves

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

Slaves

No user's control on the order

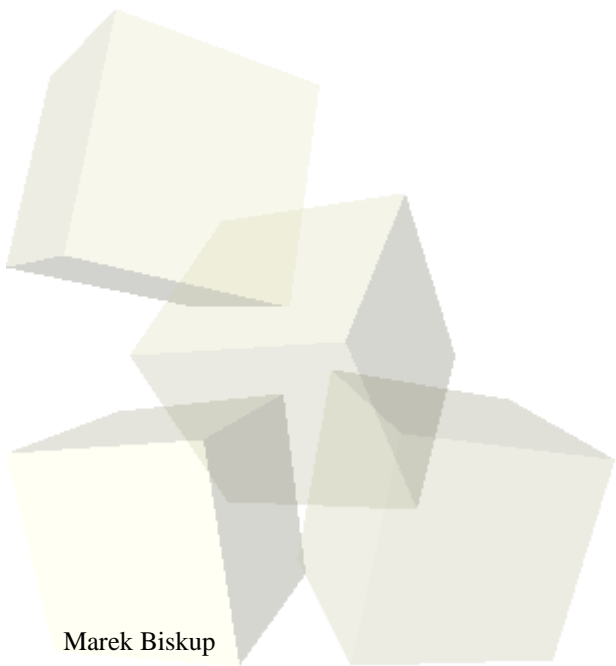
Slaves

Client



# 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();
}
```

- **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

The image displays the ROOT Session Viewer interface, which is used for managing database sessions. It features a tree view on the left, a configuration panel on the right, and a progress bar at the bottom.

- Predefined sessions:** A callout box points to the 'proof1' and 'proof2' entries in the 'Sessions' tree view.
- Define a new session:** A callout box points to the 'Connection Name' field in the 'New Server' configuration panel, which contains the text 'proof2'.
- Session startup progress bar:** A callout box points to a green progress bar at the bottom of the window, which is labeled '16%' and 'Setting up worker servers'.
- Session startup status:** A callout box points to the 'Connect' button in the 'New Server' configuration panel, which is highlighted with a red border.



# Query definition

ROOT Session Viewer

File Session Query Help

Sessions

- Local
- proof1
- Query 1
- proof2

Status Feedback Commands

Status : 1 files, number of events 1350, starting event

40%

Estimated time left :  
16.3 sec (550 events of 1350 processed)  
49.1 events/sec

Disconnect Show log...  
New Query... Get Queries

Results URL :

Query Dialog

New Query

Query Name : Query 1

TChain : bg\_filtered Browse...

Selector : SimpleSel.C Browse...

Options : "ASYN" More >>

Add Query Save & Submit Close

Open

Look in: demo

- SimpleSel.C
- h1analysis.C
- prepSimple.C
- preph1.C

File name: SimpleSel.C Open Cancel

Files of type: C files (\*.C)

**Choose TSelector**

Chains Selection Dialog

List of Chains in Memory :

- bg\_filtered

Selected chain : TTree : bg\_filtered

- SimpleSel.C
- h1analysis.C
- prepSimple.C
- preph1.C
- ..

Apply & Return Cancel

**Select a chain**

**Execute to create chain**



# Query processing

ROOT Session Viewer

File Session Query Help

Sessions

- Local
  - proof1
  - Query 1
  - proof2

Status Feedback Commands

Status : 1 files, number of events 1350, starting event

40%

Estimated time left :  
16.3 sec (550 events of 1350 processed)  
49.1 events/sec

Disconnect Show log...

New Query...

Results URL :

PROOF Cluster proof1 ready

Processing information

ROOT Sess

File Session Query

Sessions

- Local
  - proof1
  - Query 1
  - proof2

Submit

Stop

Show Log

Results URL :

PROOF Cluster proof1 ready

ROOT Session Viewer

File Session Query Help

Sessions

- Local
  - proof1
  - Query 1
  - Query 2
  - proof2

Status Feedback Commands

Events processed per Slave

Slave	Events processed
S1	25000
S2	27000
S3	26000
S4	21000
S5	22000
S6	22000
S7	21000
S8	22000
S9	20000
S10	21000
S11	23000

Feedback Histos

- PacketsHist
- EventsHist
- NodeHist
- LatencyHist

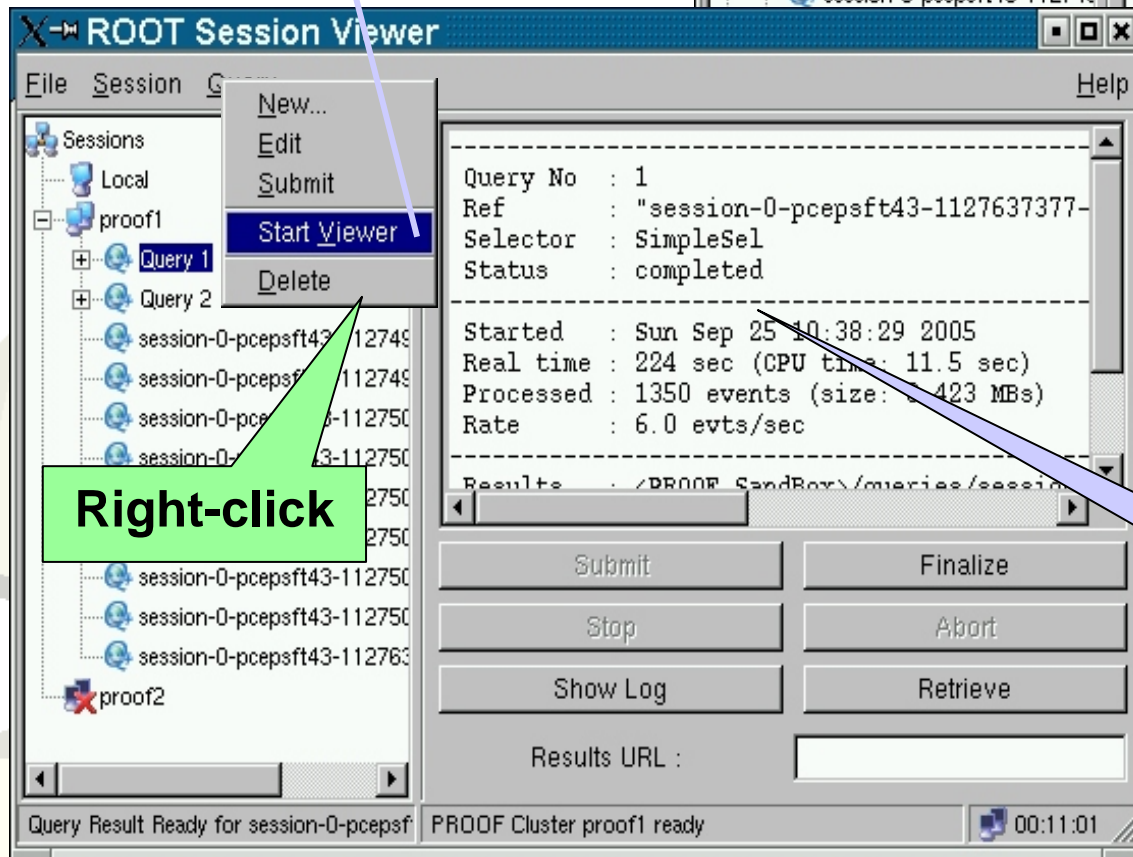
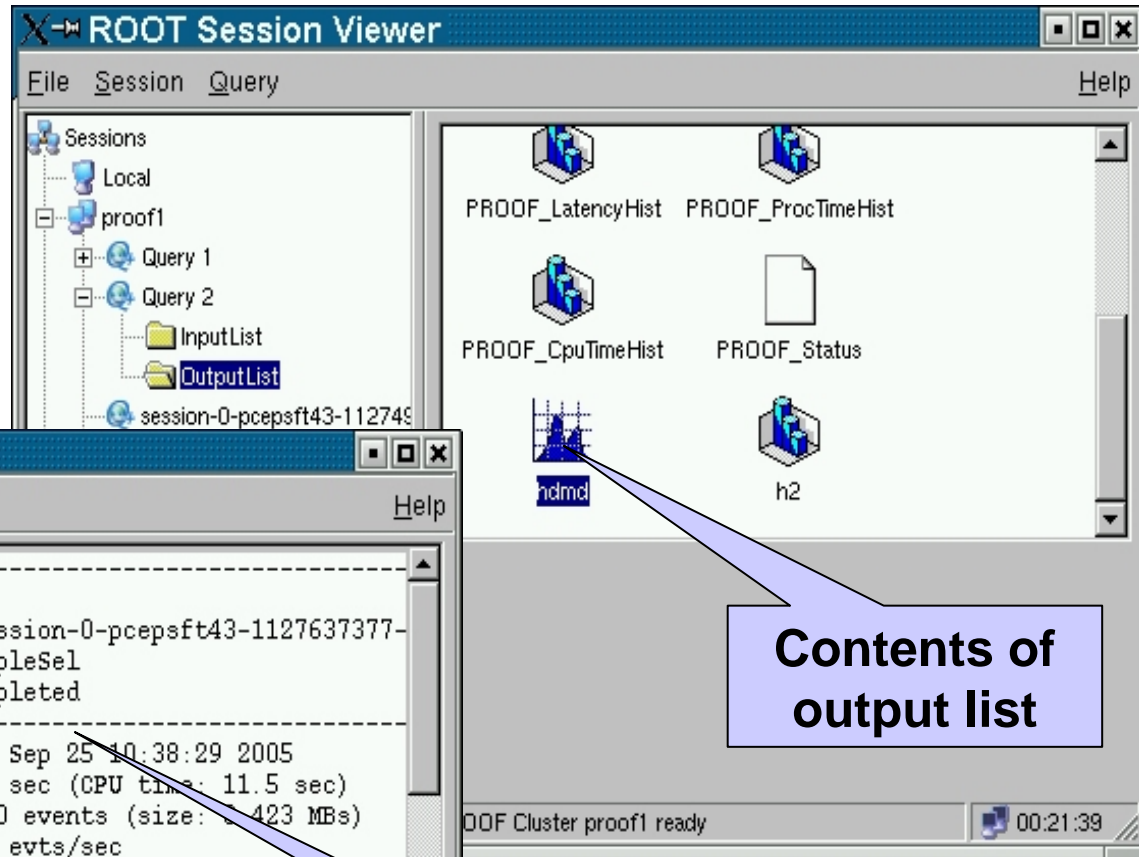
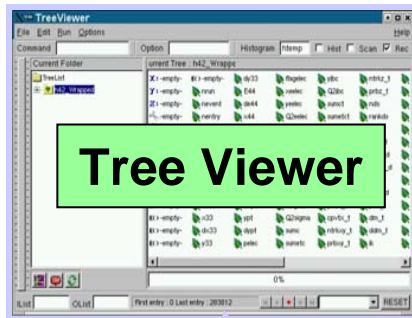
Feedback

Query Result Ready for session-0-pcepsf PROOF Cluster proof1 ready 00:08:40

Feedback histograms



# Query browsing

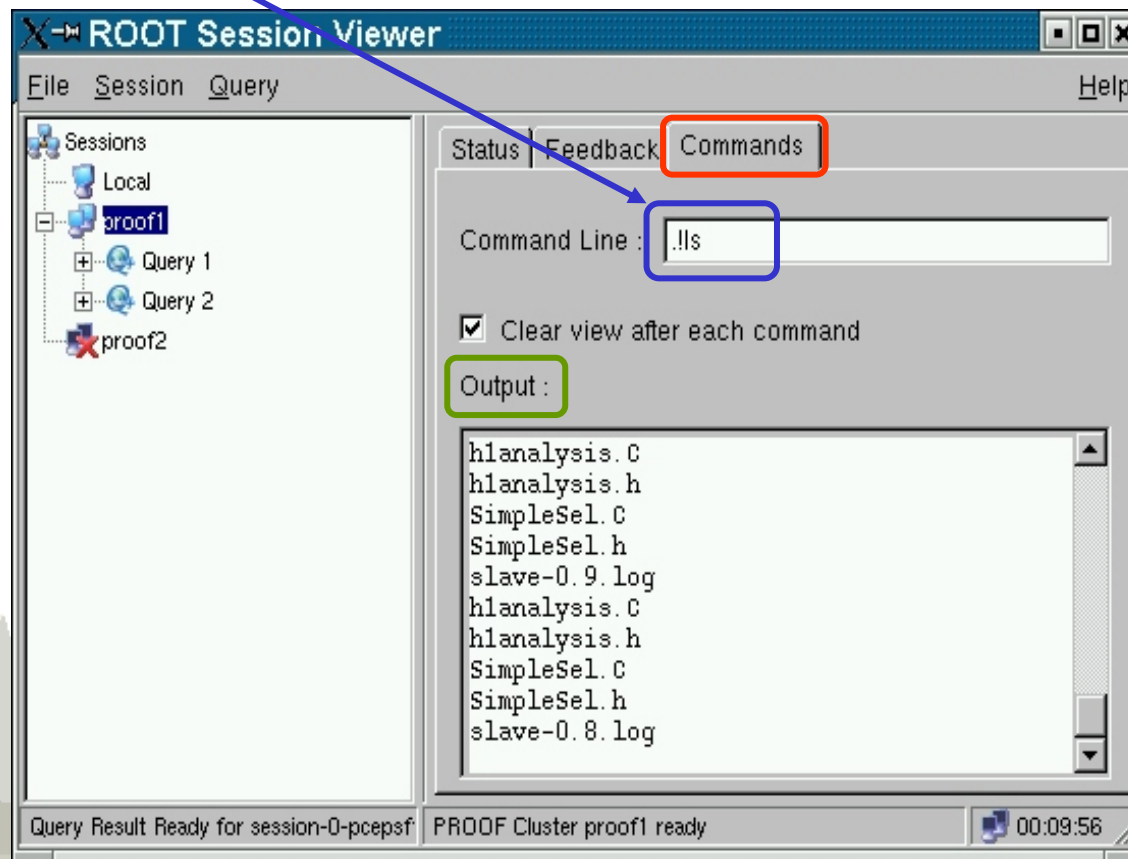


**Contents of output list**

**Details about the query**

# Command execution

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







- 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