PROOF tutorial
PROOF basics

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

Terminology (1)

- Client
  - Machine running a ROOT session opening the connection to the PROOF master
- Master
  - PROOF machine running a ROOT application coordinating the work between workers and merging the results
- Worker (or Slave)
  - PROOF machine running a ROOT application doing the actual work
- PROOF session
  - A set {client, master, workers} started by TProof::Open
- Query
  - Process request submitted by the client to the Master; consists of a selector and possibly a chain

Terminology (2)

- Package / PAR file
  - Additional code needed by the selector, not available on the PROOF cluster, loaded as a separate library
  - Gzipped tarball containing all what needed to enable the package
- Selector
  - A class deriving from TSelector providing the code to be processed
- Dataset
  - Set of files containing the TTree to be processed
  - Can be a TChain, TFileCollection, TDSet
  - Can be the name of a TFileCollection stored on the master

Open a PROOF-Lite session

- Once the daemon is running we can open a PROOF session

```
$ root -l
root [0] TProof *proof = TProof::Open("")
 +++ Starting PROOF-Lite with 2 workers +++
Opening connections to workers: OK (2 workers)
Setting up worker servers: OK (2 workers)
PROOF set to parallel mode (2 workers)
(class TProof*)0x8330140
root [1]
```

- Now we are ready to go, but ... what's TProof ?

TProof: the PROOF shell

- TProof is the interface class to interact with the PROOF session
- Everything you will do on the session will be through the TProof class methods, e.g.
  - Print() gives information about the session
  - Exec("*CINT command*") allows to execute simple commands on the workers
  - AddInput() make objects available to the selector
  - Process(...) allows to run a selector
  - GetOutputList() returns the list of output objects
  - DrawSelect() allows to draw distributions
  - ...
- We will see others as they come in the game

TProof: gProof

- You can create as many PROOF sessions as you want
  - Each session is controlled by its TProof object
- Running TProof::Open again on the same master does not hurt, just returns the pointer to the existing open session
- The global gProof points to the latest TProof created or attached
  - TProof::cd() allows to change the session pointed to by gProof

TProof::Print (1)

- Gives information about the session

```
$ root [1] p->Print()
*** PROOF-Lite cluster (parallel mode, 2 workers):
Host name:                      macphsft12.local
User:                           ganis
ROOT version|rev|tag:           5.32/02|r43514
Architecture-Compiler:          macosx64-gcc421
Protocol version:               33
Working directory:              /Users/ganis/local/root/opt/root
Communication path:             /var/folders/uC/uC0RGjQUFlmzR689bg+JJU+
+0gQ/-Tmp-/plite-38583
Log level:                      0
Number of workers:              2
Number of active workers:   2
Number of unique workers:   1
Number of inactive workers: 0
Number of bad workers:          0
Total MB's processed:           0.00
Total real time used (s):   0.000
Total CPU time used (s):    0.000
```

TProof::Print (2)

- TProof::Print("a") gives full details about the workers

```
List of workers:
*** Worker 0.0  (valid)
    Worker session tag:      0.0-macphsft12.local-1334609972-38703
    ROOT version|rev|tag:    5.32/02|r43514|5.32/02
    Architecture-Compiler:   macosx64-gcc421
    Working directory:       /Users/ganis/.proof/local-root-opt-
root/session-macphsft12.local-1334609971-38698/worker-0.0
    MB's processed:          0.00
    MB's sent:               0.00
    MB's received:           0.00
    Real time used (s):      0.000
    CPU time used (s):       0.000
*** Worker 0.1  (valid)
    Worker session tag:      0.1-macphsft12.local-1334609972-38705
    ROOT version|rev|tag:    5.32/02|r43514|5.32/02
    Architecture-Compiler:   macosx64-gcc421
    Working directory:       /Users/ganis/.proof/local-root-opt-
root/session-macphsft12.local-1334609971-38698/worker-0.1
    MB's processed:          0.00
    MB's sent:               0.00
    MB's received:           0.00
    Real time used (s):      0.000
    CPU time used (s):       0.000
```

A few remarks

- Workers are uniquely identified by the ordinal number 0.n
  - Master has always ordinal 0

- For PROOF-Lite, the working directories are under

  $HOME/.proof/path-where-we-started/...

- The location of the working directory can be changed with

  ProofLite.Sandbox /tmp

The sandbox

- Each user get a working space on the cluster (sandbox)
  - Default location $HOME/.proof
- The *sandbox* has several sub-directories
  - cache
    - Cache package tarballs, selector code and binaries
  - packages
    - Area where packages are actually build / installed
  - session-*sessionUniqueID*
    - Working area for session "*sessionUniqueID*"
  - queries (on master only)
    - Where the results of processing are stored
  - datasets (on master only)
    - Information about datasets
- In PROOF-Lite, queries and session-sessionUniqueID under $HOME/.proof/*path-from-where-we-started*

Session Unique ID

- Each PROOF session has a unique ID in the form

  hostname-creationtime-processID

```
root [] p->GetSessionTag()
(const char* 0x1016c9d50)"macphsft12.local-1334609971-38698"
```

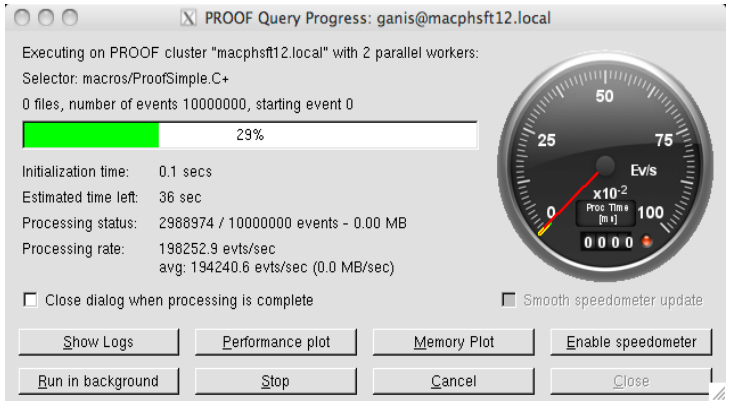
  referring to the master (or the client session in PROOF-Lite)
- The Session Unique ID is used to create the session working area in the sandbox

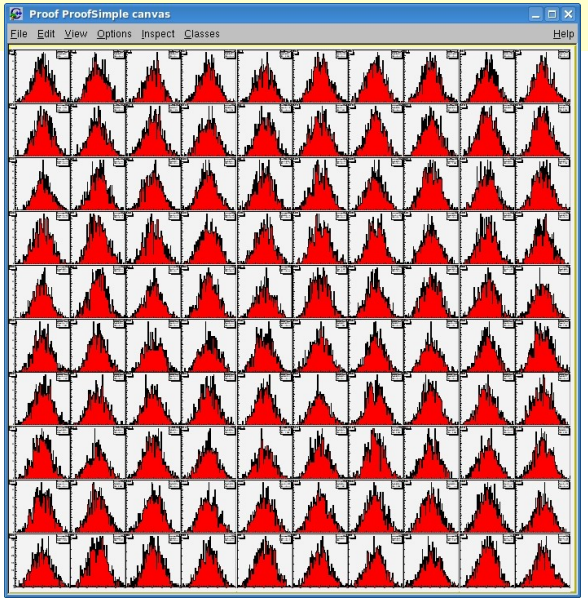
  sandbox/session-*sessionUniqueID*
- The working area contains for each process
  - The actual working subdirectory master-0-*processUniqueID* or worker-0.m-*processUniqueID*
  - Three files: log, environment settings and ROOT environment settings

First processing (1)

- We are ready to run a first query
- macros/ProofSimple.C,.h defines a TSelector which fills 100 histograms with gaussian random numbers
- Just do

```
root [] p->Process("macros/ProofSimple.C+",10000)
Mst-0: grand total: sent 101 objects, size: 94354 bytes
(Long64_t)0
root []
```

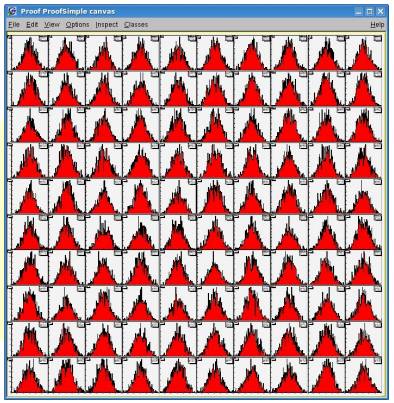




First processing (2)

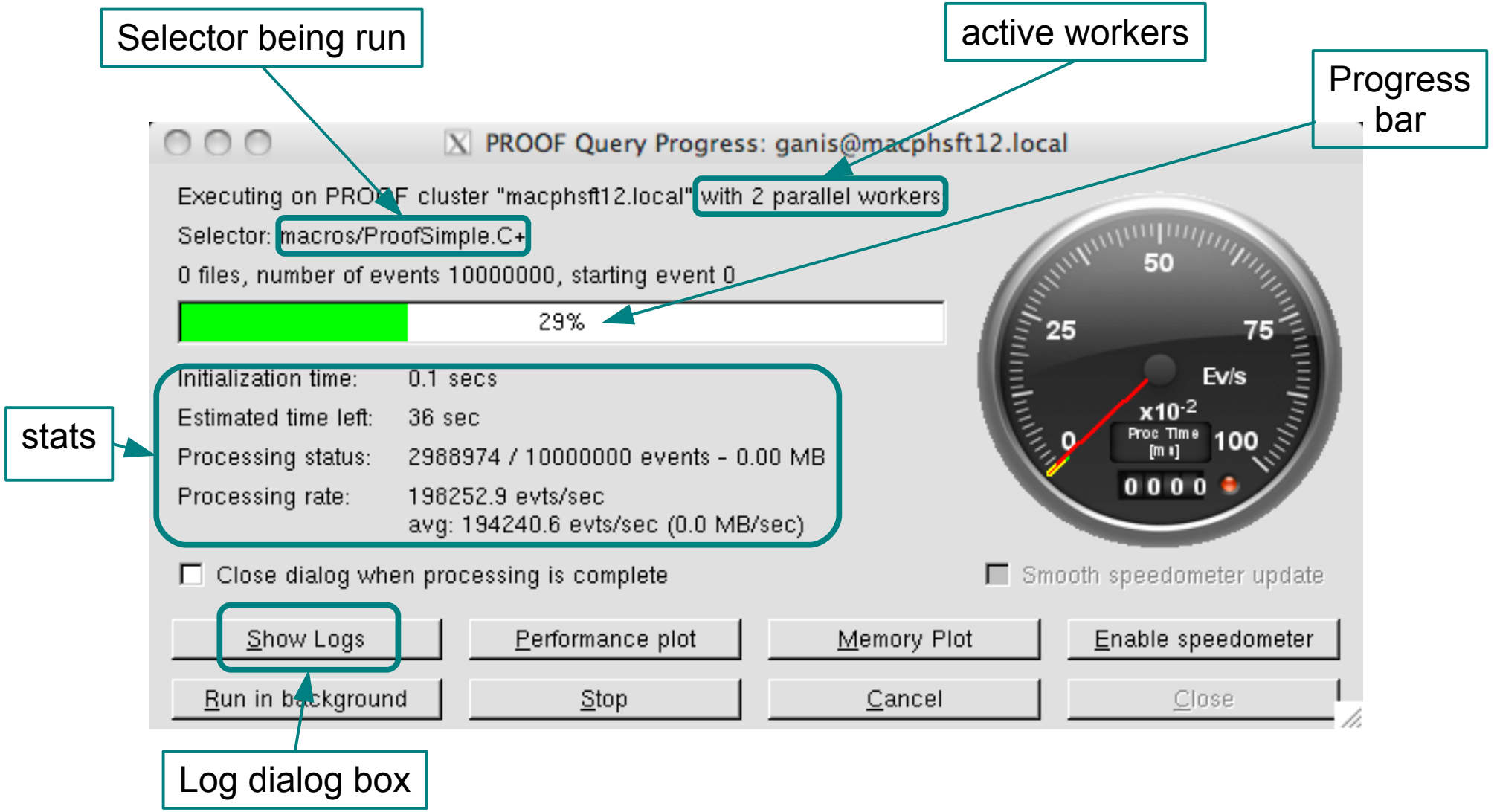
- We can repeat on the local session: the macro macros/processLocal.C allows you to do that
- Just do

```
root [] .L macros/processLocal.C
root [] processLocal("macros/ProofSimple.C+",10000)
root []
```

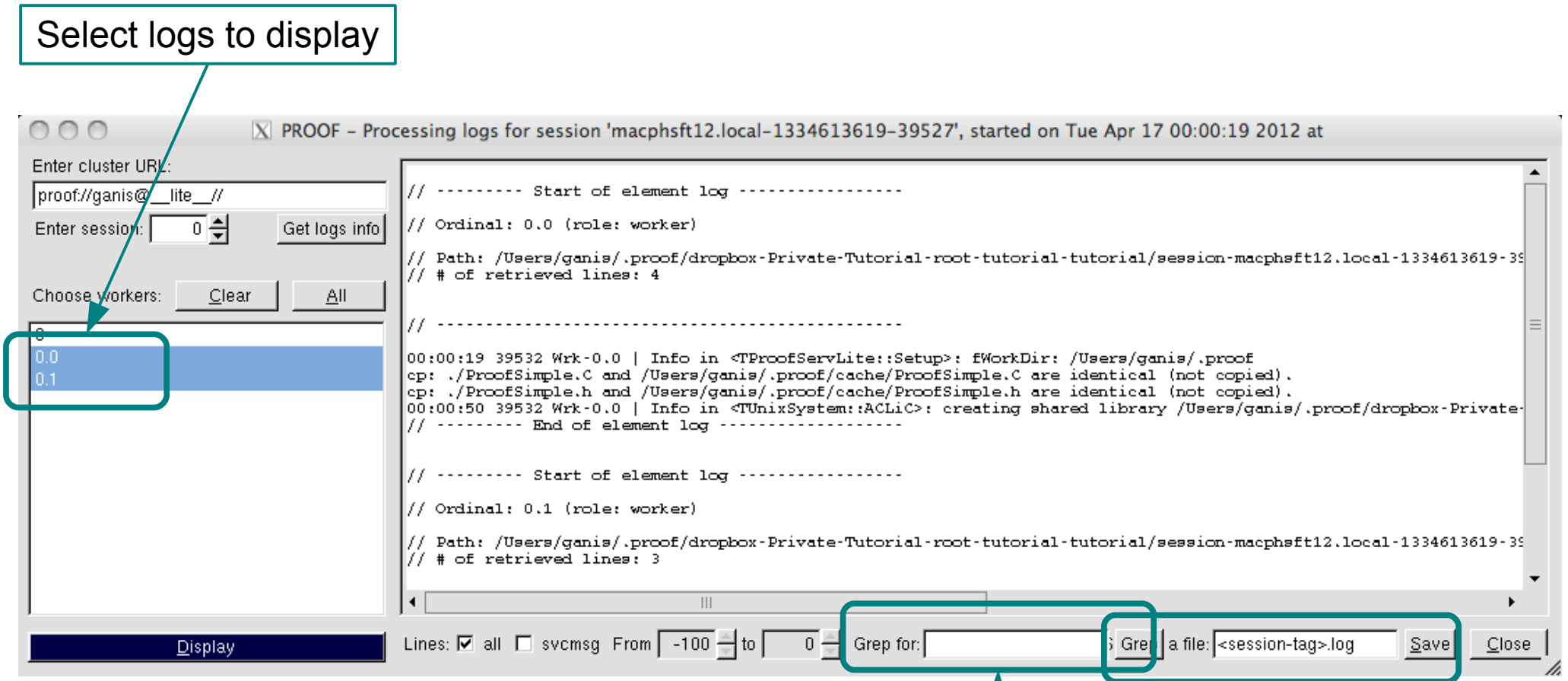


- Try measure the used time with gROOT->Time() in the two cases (PROOF and Local)
  - What do you find?

The dialog box



The log dialog box



Can be started also with TProof::fLogViewer

Where are the results?

- In the output list ...

```
root [] gProof->GetOutputList()
(class Tlist*)0x89eae58
root [] gProof->GetOutputList()->ls()
OBJ: TStatus    PROOF_Status      : 0 at: 0x8a264a8
OBJ: TH1F       h0       h0 : 0 at: 0x89d5b48
OBJ: TH1F       h1       h1 : 0 at: 0x8a22de0
OBJ: TH1F       h2       h2 : 0 at: 0x8a21f88
OBJ: TH1F       h3       h3 : 0 at: 0x8a215f8
OBJ: TH1F       h4       h4 : 0 at: 0x8a24100
OBJ: TH1F       h5       h5 : 0 at: 0x8a288b8
OBJ: TH1F       h6       h6 : 0 at: 0x8a31c20
...
OBJ: TH1F       h96      h96 : 0 at: 0x89e9b38
OBJ: TH1F       h97      h97 : 0 at: 0x89ea000
OBJ: TH1F       h98      h98 : 0 at: 0x89ea4c8
OBJ: TH1F       h99      h99 : 0 at: 0x89ea990
root []
```

How did they get there?

- Objects wanted back must be added to the list fOutput of the selector in the method SlaveBegin
  - They can also be added in SlaveTerminate, but then they will not be able for feedback

```
void ProofSimple::SlaveBegin(TTree * /*tree*/)
{
   ...
   fNhist = 100; // number of histos
   ...
   fHist = new TH1F*[fNhist];

   // Create the histogram
   for (Int_t i=0; i < fNhist; i++) {
      fHist[i] = new TH1F(Form("h%d",i), Form("h%d",i), 100, -3., 3.);
      fHist[i]->SetFillColor(kRed);
      fOutput->Add(fHist[i]);
   }
   ...
}
```

Create a macro to run PROOF

```
// macro to steer PROOF running

void runProof(const char *option = "simple",
              const char *master = "localhost")
{
   // Get the option into a Tstring for easier manipulation
   TString opt(option);

   // Start or attach to PROOF
   TProof *proof = TProof::Open(master);
   if (!proof) {
      Printf("runProof: could not get PROOF at '%s'", master);
      return;
   }

   // Run according to option
   if (opt.Contains("simple")) {
      proof->Process("macros/ProofSimple.C+",10000);
   }
}
```

macros/runProof.C

Adding feedback

- PROOF allows to get back the realtime status of the output objects for feedback on the ongoing analysis
- The object must be registered in the output list
- The setup is quite simple

```
// Add the name of the object(s) wanted to the feedback list
proof->AddFeedback("h10");

// Setup something that can display the temporary result
// TDrawFeedback creates on TCanvas per object
TDrawFeedback fb(proof);
```

Feedback on evts per worker

```
// Create histogram with events per worker
gEnv->SetValue("Proof.StatsHist",1);

// Add the name of the object(s) wanted to the feedback list
proof->AddFeedback("PROOF_EventsHist");

// Setup something that can display the temporary result
// TDrawFeedback creates on TCanvas per object
TDrawFeedback fb(proof);
```

Processing data: H1 example

- Using the chain

```
root [] .L macros/createH1Chain.C
root [] TChain *chain = createH1Chain()
root [] chain->Process("macros/h1analysis.C+")
```

- To Process the chain under PROOF

```
root [] TProof *p = TProof::Open("seoul..@localhost")
root [] chain->SetProof()
root [] chain->Process("macros/h1analysis.C+")
```

- Could you add a case for this in macros/runProof.C

Processing data: H1 example

- Using the TFileCollection

```
root [] TProof *p = TProof::Open("localhost")
root [] .L CreateFileCollection.C
root [] TFileCollection *h1fc =
                    CreateFileCollection("files/h1-http.txt")
```

- Process the TFileCollection

```
root [] p->Process(h1fc, "macros/h1analysis.C+")
```

- Why the complication?

TFileCollection: the dataset

- The concept of dataset is very useful in HEP: it refers to a set of files containing homogeneous data
  - e.g. all the data taken during Summer 2009 under uniform detector conditions.
- All useful is to refer to a dataset by name
- TFileCollection: named list of TFileInfo
- TFileInfo: most generic way of describing a file
  - Multiple URLs, meta-information
- A TFileCollection is the typically the result of a query to a catalog

Datasets in PROOF

- TProof has a set of methods to perform basic operations on datasets
  - RegisterDataSet(const char *name, TFileCollection *)
  - VerifyDataSet(const char *name)
  - ShowDataSets(const char *uri)
  - TFileCollection *GetDataSet(const char *name)
  - ...
- The name is in the form /group/user/datasetname
  - 'group' is an advanced PROOF concept: by default anybody is in group 'default'

Register / Verify the H1 dataset

- Register the file collection with name that is desired

```
root [] TProof *p = TProof::Open("localhost")
root [] .L CreateFileCollection.C
root [] TFileCollection *h1fc =
                    CreateFileCollection("files/h1-http.txt")
root [] p->RegisterDataSet("h1-http", fc);
```

- Verify the dataset (opens the file, takes a while)

```
root [] p->VerifyDataSet("h1-http");
...
root [] p->ShowDataSets()
Dataset URI                                | # Files | ...
/default/ganis/h1-http                     |       4 | ...
```

Verify caches the information: the validation step is much faster for verified datasets

Process the H1 dataset

- Process the dataset by name

```
root [] p->Process("h1-http", "macros/h1analysis.C+")
```

- Or even do some drawing

```
root [] p->DrawSelect("h1-http", "dm_d")
```

Loading additional code

- When the selector requires additional code, e.g. A new class MyClass, PROOF provides two ways to make it available
  - TProof::Load("MyClass.C")
    - Equivalent of .L on the ROOT shell
    - Convenient for simple things
  - Package ARchives (PAR)
    - Structured archives with build and setup facilities
    - Convenient for more complex and stable things, e.g. the experiment analysis suite

PARs

- Zipped tarballs identified by a name and the .par extension, e.g. *pack*.par
- The tarball contains a structure like this

  ./pack

  ./pack/PROOF-INF

  ./pack/PROOF-INF/BUILD.sh

  ./pack/PROOF-INF/SETUP.C
- The code (.C, .h, makefiles, ...) should be put in the top level directory
- BUILD.sh is the script to build the package, e.g. runs 'make'
- SETUP.C is a macro running the final setup

PARs

- $ cp $ROOTSYS/tutorials/proof/event.par .
- root [] TProof *p = TProof::Open(“localhost”)
- root [] p->ClearPackages()
- root [] p->UploadPackage(“event.par”)
- root [] p->EnablePackage(“event”)
- root [] p->Exec(“Event *ev = 0”)

PAR example: event.par

- Have a look at par/event.par

```
$ tar tzvf par/event.par
drwxr-xr-x ganis/sf          0 2008-07-21 15:17 event/
-rw-r--r-- ganis/sf      13885 2008-07-21 15:17 event/Makefile.arch
-rw-r--r-- ganis/sf       2282 2008-07-21 15:17 event/Makefile
-rw-r--r-- ganis/sf       7902 2008-07-21 15:17 event/Event.h
-rw-r--r-- ganis/sf        259 2008-07-21 15:17 event/EventLinkDef.h
-rw-r--r-- ganis/sf      14695 2008-07-21 15:17 event/Event.cxx
drwxr-xr-x ganis/sf          0 2008-07-21 15:17 event/PROOF-INF/
-rwxr-xr-x ganis/sf        101 2008-07-21 15:17 event/PROOF-INF/BUILD.sh
-rw-r--r-- ganis/sf         86 2008-07-21 15:17 event/PROOF-INF/SETUP.C
```

Handling PARs in PROOF

- TProof provides the following methods to work w/ PARs
  - UploadPackage(const char *name)
  - EnablePackage(const char *name)
  - ClearPackage(const char *name)
  - ClearPackages()
  - ShowPackages()
  - ShowEnabledPackages()
- Try to upload and enable par/event

```
root [] TProof *p = TProof::Open("localhost")
root [] p->UploadPackage("par/event");
root [] p->ShowPackages();
...
root [] p->EnablePackage("event");
```

Handling PARs locally

- PAR concept very useful, but currently available only via PROOF
- In the local session (i.e. w/o starting PROOF) there are not (yet) tools to handle PAR files
  - TPackageManager under preparation
- The macro macros/loadPackage.C allows you to load a package locally to run comparisons between PROOF and local sessions

```
root [] .L macros/loadPackage.C
root [] loadPackage("par/event")
```

Event example

- How does it work for real processing?
- EventTree_Proc.C and EventTree_ProcOpt.C are selectors for 'event' reading different fractions of data
- We need also some files: 10 files, ~750 MB
  - files/event-http.txt
    - From http://root.cern.ch/data
  - files/event-nfs.txt
    - From the NFS server /repository/...
- Use createFileCollection to create a TFileCollection
  - Define a dataset
  - Process it by name

Event example: details

```
root [] TProof *p = TProof::Open("proof..@localhost:3000")
root [] .L macros/createFileCollection.C
root [] TFileCollection *evtfc =
            createFileCollection("files/event-http.txt")
root [] p->UploadPackage("par/event")
root [] p->EnablePackage("event")
root [] p->Process(evtfc,"macros/EventTree_ProcOpt.C+")
```

Pythia8 example

- This is an example of running a real MonteCarlo simulation on PROOF
  - PYTHIA8 is the first usable C++ version of a famous HEP generator
- It needs a PAR file to setup the environment and and environment setting for the sessions (new concept)
- To run the example, we need to link to pythia8 also locally
  - I have installed it under /repository/root/pythia8/pro

pythia8.par

```
$ tar tzvf par/pythia8.par
drwxr-xr-x mslawins/sf       0 2008-07-03 19:45 pythia8/
-rw-r--r-- mslawins/sf    1521 2008-07-03 19:45 pythia8/main03.cmnd
drwxr-xr-x mslawins/sf       0 2008-07-03 19:46 pythia8/PROOF-INF/
-rw-r--r-- mslawins/sf     233 2008-07-03 19:46 pythia8/PROOF-INF/SETUP.C
```

```
void SETUP()
{
   // Load the libraries

   gSystem->Load("$PYTHIA8/lib/libpythia8.so");
   gSystem->Load("libEG");
   gSystem->Load("libEGPythia8");

   // Set the include paths
   gROOT->ProcessLine(".include $PYTHIA8/include");
}
```

Pythia8 example (2)

- Pythia8 needs the environment variables PYTHIA8 and PYTHIA8DATA set correctly, both locally

```
$ export PYTHIA8=/repository/root/pythia8/pro
$ export PYTHIA8DATA=/repository/root/pythia8/pro/xmldoc
```

and on the PROOF sessions: how can we set the latter?

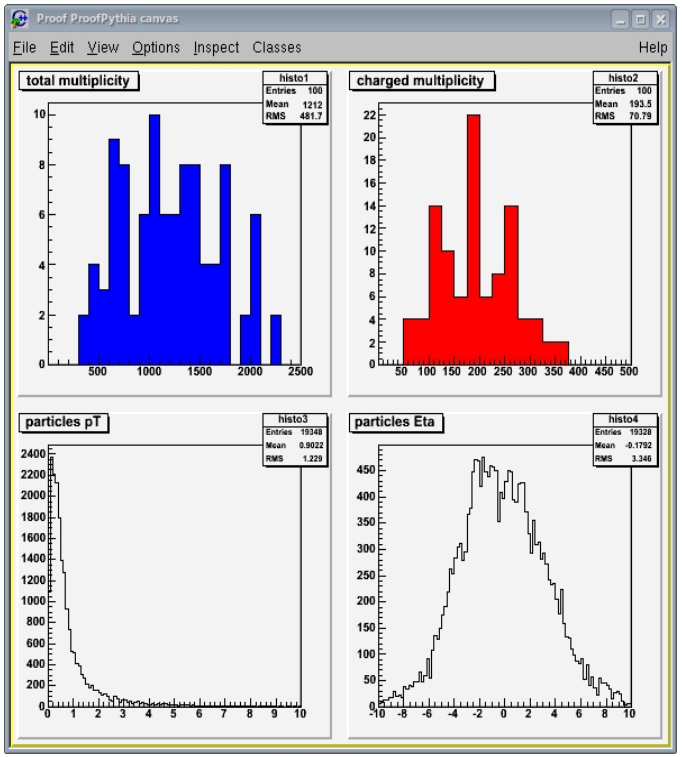
- We can set an environment before starting the servers like this:

```
root [] TString env("echo export PYTHIA8=/opt/pythia8/pro;");
root [] env += " export PYTHIA8DATA=/opt/pythia8/pro/xmldoc";
root [] TProof::AddEnvVar("PROOF_INITCMD", env.Data())
root [] p = TProof::Open("proof..@localhost:3000")
root [] p->Exec("gSystem->Getenv(\"PYTHIA8\")")
(const char* 0xbf8fd94a)"/opt/pythia8/pro"
(const char* 0xbfdd294a)"/opt/pythia8/pro"
(Int_t)(0)
```

Pythia8 example (3)

- We can then load the PAR file and run the selector

```
root [] p->UploadPackage("par/pythia8");
root [] p->EnablePackage("pythia8");
root [] p->Process("macros/ProofPythia.C+",100);
```



Summary

We have learned

- How to start PROOF on the local session
- How to run simple generic queries
- How to process a small dataset
- How to draw a variable from a dataset
- How to set the environment
- How to PAR files

Next we can try to run a query on a real cluster at CERN ...