ROOT and PROOF Tutorial

Arsen Hayrapetyan

Yerevan Physics Institute, Yerevan, Armenia;
European Organization for Nuclear Research (CERN)

Arsen.Hayrapetyan@cern.ch

Martin Vala

Institute of Experimental Physics,
Slovak Academy of Sciences;
European Organization for Nuclear Research
(CERN)

Martin.Vala@cern.ch

Outline

- ➢ Introduction to ROOT
- ✓ ROOT hands-on exercises
- ➢ Introduction to PROOF
- ✓ PROOF hands-on exercises

What is ROOT?

- Object-oriented data handling and analysis framework
  - Framework: ROOT provides building blocks (root classes) to use in your program.
  - Data handling: ROOT has classes designed specifically for storing large amount of data (GB, TB, PB) to enable effective data analysis.
  - Analysis: ROOT has complete collection of statistical, graphical, networking and other classes that user can use in their analysis.
  - Object-oriented: ROOT is based on OO programming paradigm and is written in C++.

Who is developing ROOT?

- ROOT is an open source project started in 1995 by René Brun and Fons Rademakers.
- The project is developed as a collaboration between:
  - Full time developers:
    - 7 developers at CERN (PH/SFT)
    - 2 developers at Fermilab (US)
  - Large number of part-time contributors (160 in CREDITS file included in ROOT software package)
  - A vast army of users giving feedback, comments, bug fixes and many small contributions
    - ~5,500 users registered to RootTalk forum
    - ~10,000 posts per year

Who is using ROOT?

- All High Energy Physics experiments in the world
- Astronomy: AstroROOT (http://www.isdc.unige.ch/astroroot/index)
- Biology: xps package for Bioconductor project

  (http://prs.ism.ac.jp/bioc/2.7/bioc/html/xps.html)
- Telecom: Regional Internet Registry for Europe, RIPE (Réseaux IP Européens) NCC Network Coordination Centre

  (http://www.ripe.net/data-tools/stats/ttm/current-hosts/analyzing-test-box-data)
- Medical fraud detection, Finance, Insurance, etc.

ROOT is used in a many scientific fields as well as in industry.

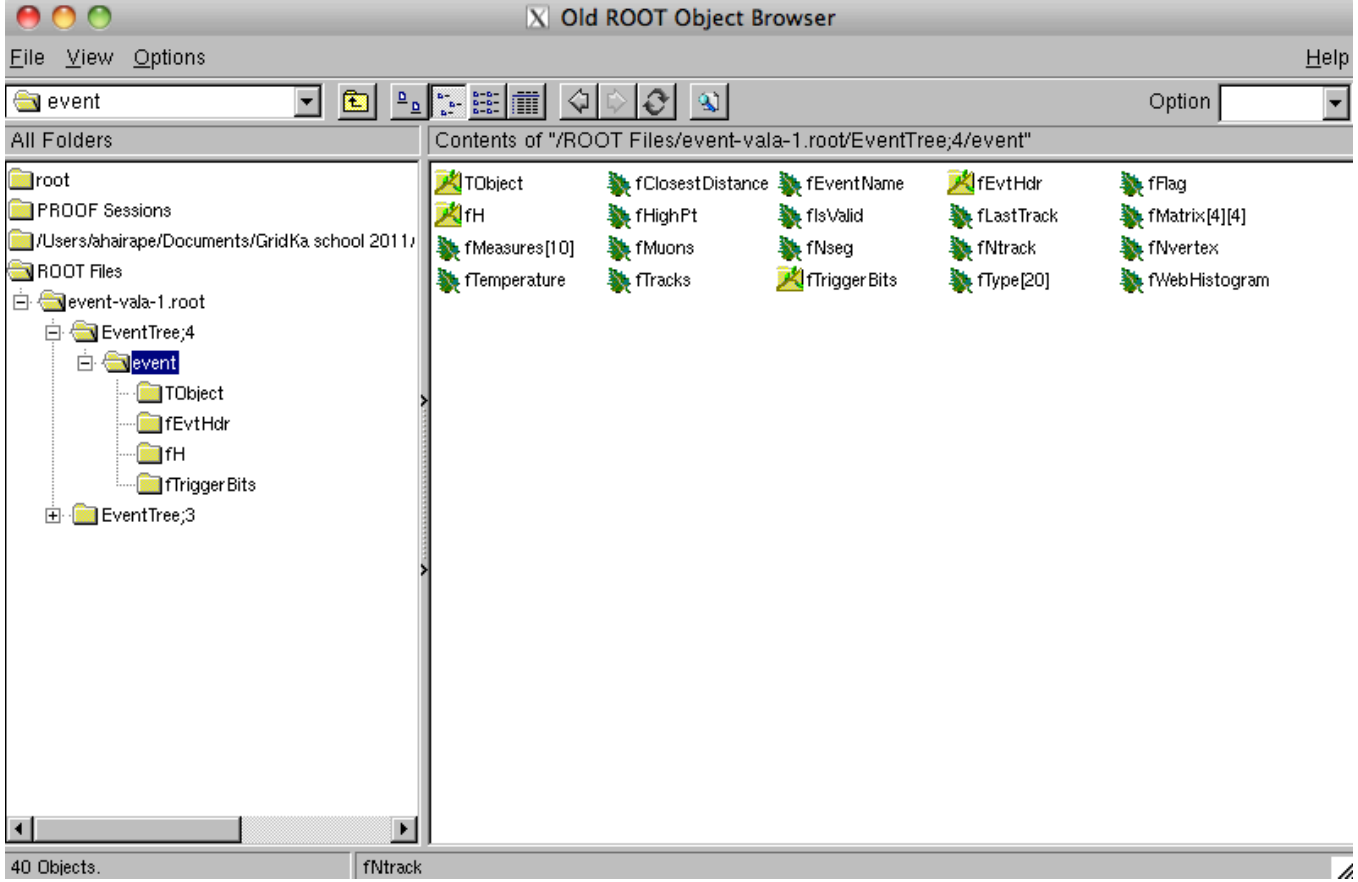
What can I do with ROOT?

You can:

- ✓ Store large amount of data (GB, TB, PB) in ROOT-provided containers: files, trees, tuples.
- ✓ Visualize the data in one of numerous ways provided by ROOT: histograms (1, 2 and 3-dimensional), graphs, plots, etc.
- ✓ Use physics analysis tools: physics vectors, fitting, etc.
- ✓ Write your own C++ code to process the data stored in ROOT containers.

ROOT features: Data containers

- ROOT provides different types of data containers:
  - Files, folders
  - Trees, Chains, etc.



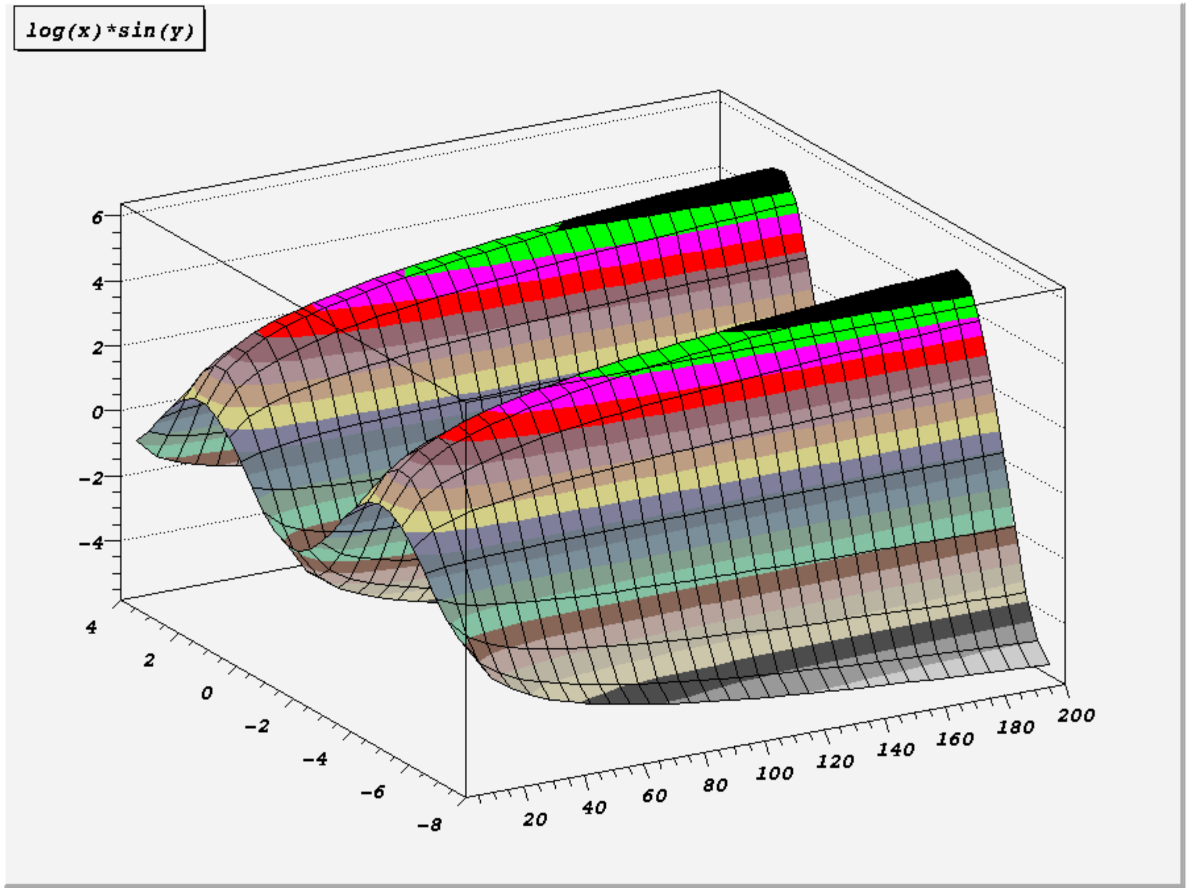
Contents of "/ROOT Files/event-vala-1.root/EventTree;4/event"

40 Objects.

fNtrack

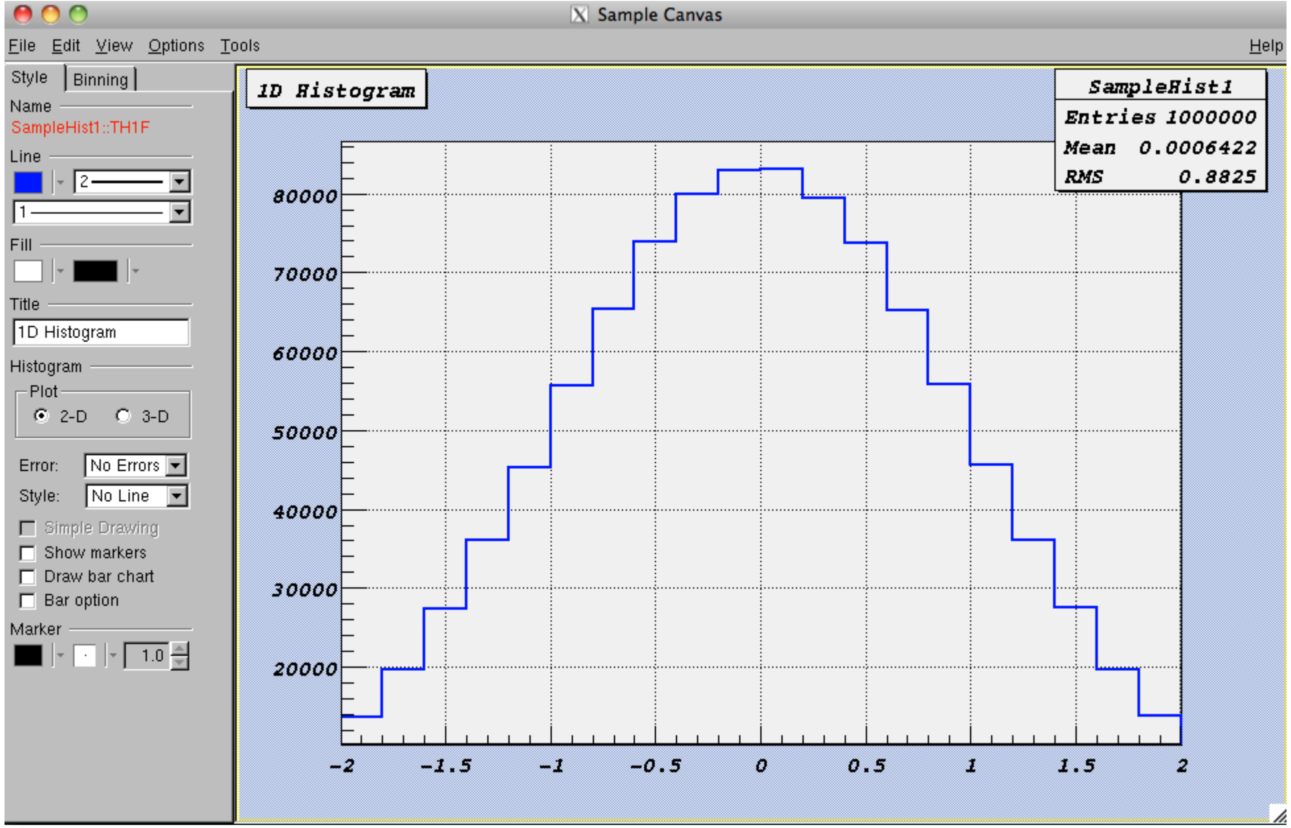
ROOT features: Data visualization

- ROOT provides a range of data visualization methods: histograms (one- and multi-dimensional), graphs, plots (scatter, surface, lego, …)



ROOT features: GUI

The Graphical User Interface (CLI) allows you to manipulate graphical objects (histograms, canvases, graphs, axes, plots, …) clicking on buttons and typing values in text boxes.



Name SampleHist1::TH1F

Line 2 1

Title 1D Histogram

Plot ● 2-D ○ 3-D

Error: No Errors

Style: No Line

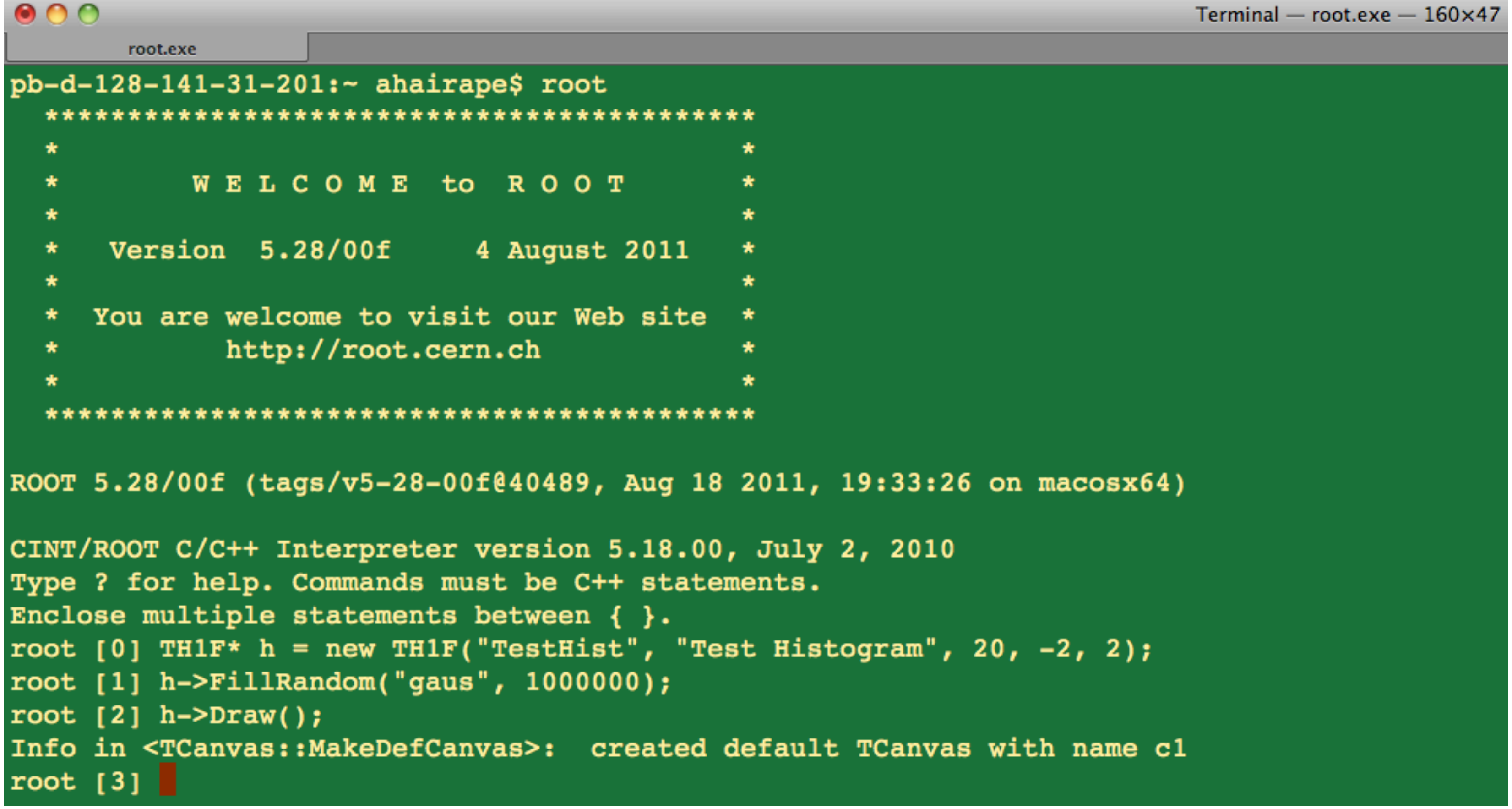
□ Simple Drawing
□ Show markers
□ Draw bar chart
□ Bar option

Marker 1.0

SampleHist1	
Entries	1000000
Mean	0.0006422
RMS	0.8825

ROOT features: CLI

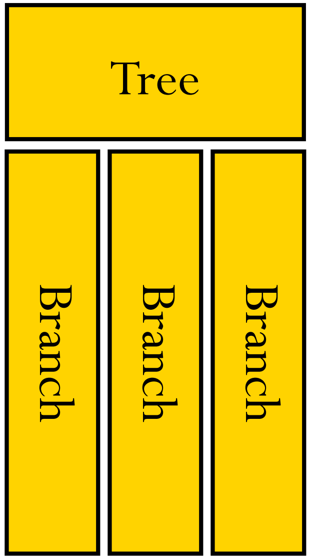
The Command Line Interface (CLI) allows you to type in the commands (C++, root-specific, OS shell) and processes them interactively via CINT – C++ interpreter.

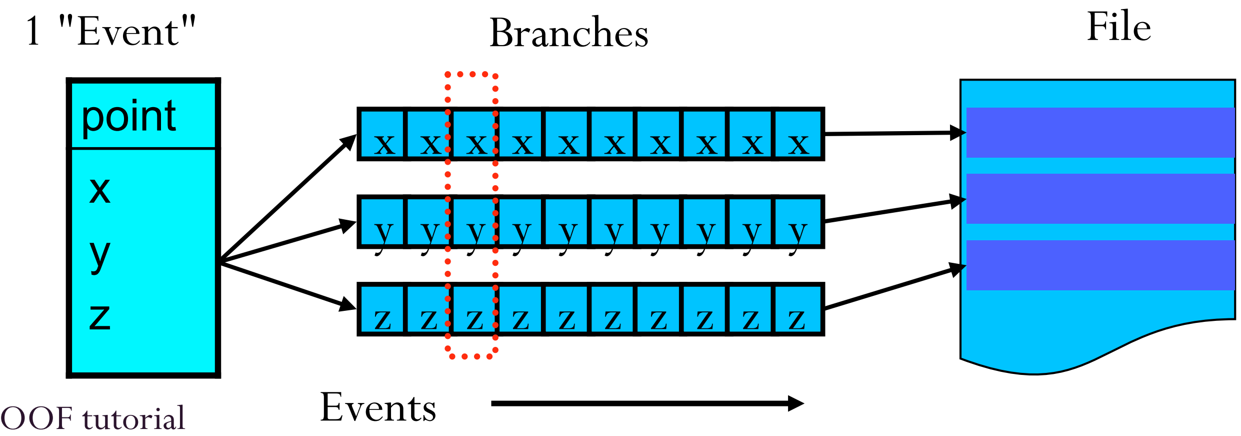


```
pb-d-128-141-31-201:~ ahairape$ root
  *******************************************
  *                                         *
  *        W E L C O M E  to  R O O T       *
  *                                         *
  *   Version   5.28/00f      4 August 2011   *
  *                                         *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch            *
  *                                         *
  *******************************************

ROOT 5.28/00f (tags/v5-28-00f@40489, Aug 18 2011, 19:33:26 on macosx64)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] TH1F* h = new TH1F("TestHist", "Test Histogram", 20, -2, 2);
root [1] h->FillRandom("gaus", 1000000);
root [2] h->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3] 
```

Trees (class TTree)

- A tree is a container for data storage
- It consists of several *branches*
  - These can be in one or several files
  - Branches are stored contiguously (split mode)
- Set of helper functions to visualize the content (e.g. Draw, Scan)
- Compressed





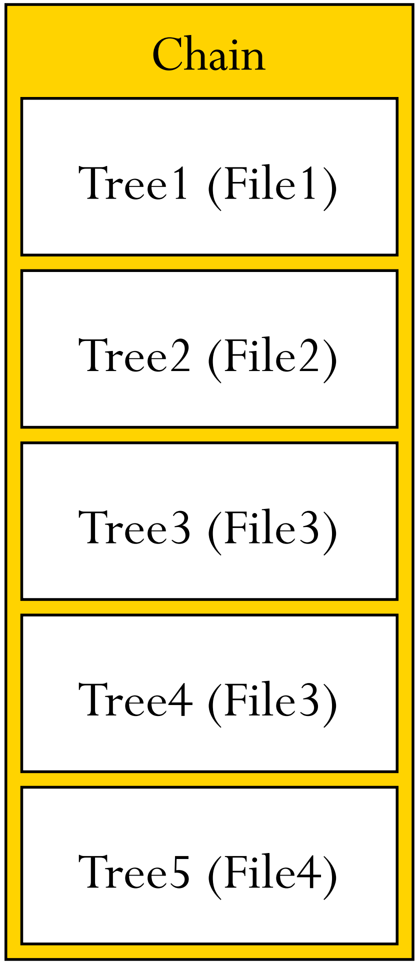
Events

Events are units of data which are stored in trees and can be processed independently from each other (PROOF’s event level parallelism is based on these properties).

Chains (class TChain)

- A chain is a list of trees (in several files)
- TTree methods can be used
  - **Draw(), Scan(), etc.**

    → these iterate over all elements of the chain
- Selectors can be used with chains
  - **Process(const char* selectorFileName)**



Chain

Tree1 (File1)

Tree2 (File2)

Tree3 (File3)

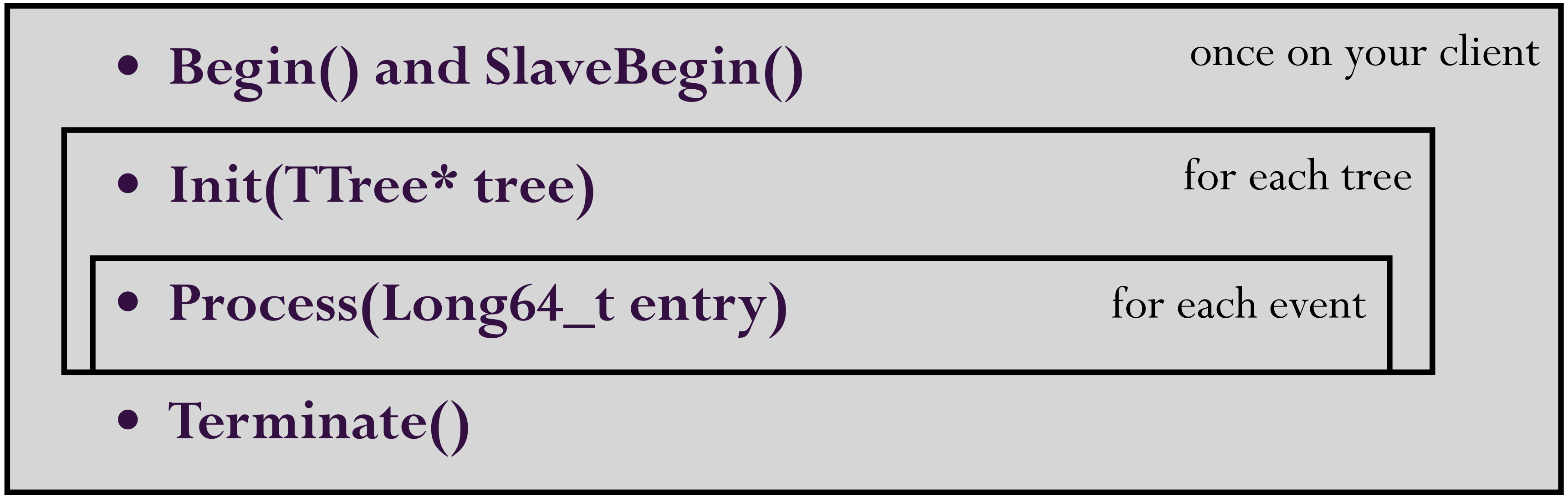
Tree4 (File3)

Tree5 (File4)

Selectors (class TSelector)

Local analysis case

- Classes derived from TSelector can run locally



- **Begin() and SlaveBegin()**

once on your client

- **Init(TTree* tree)**

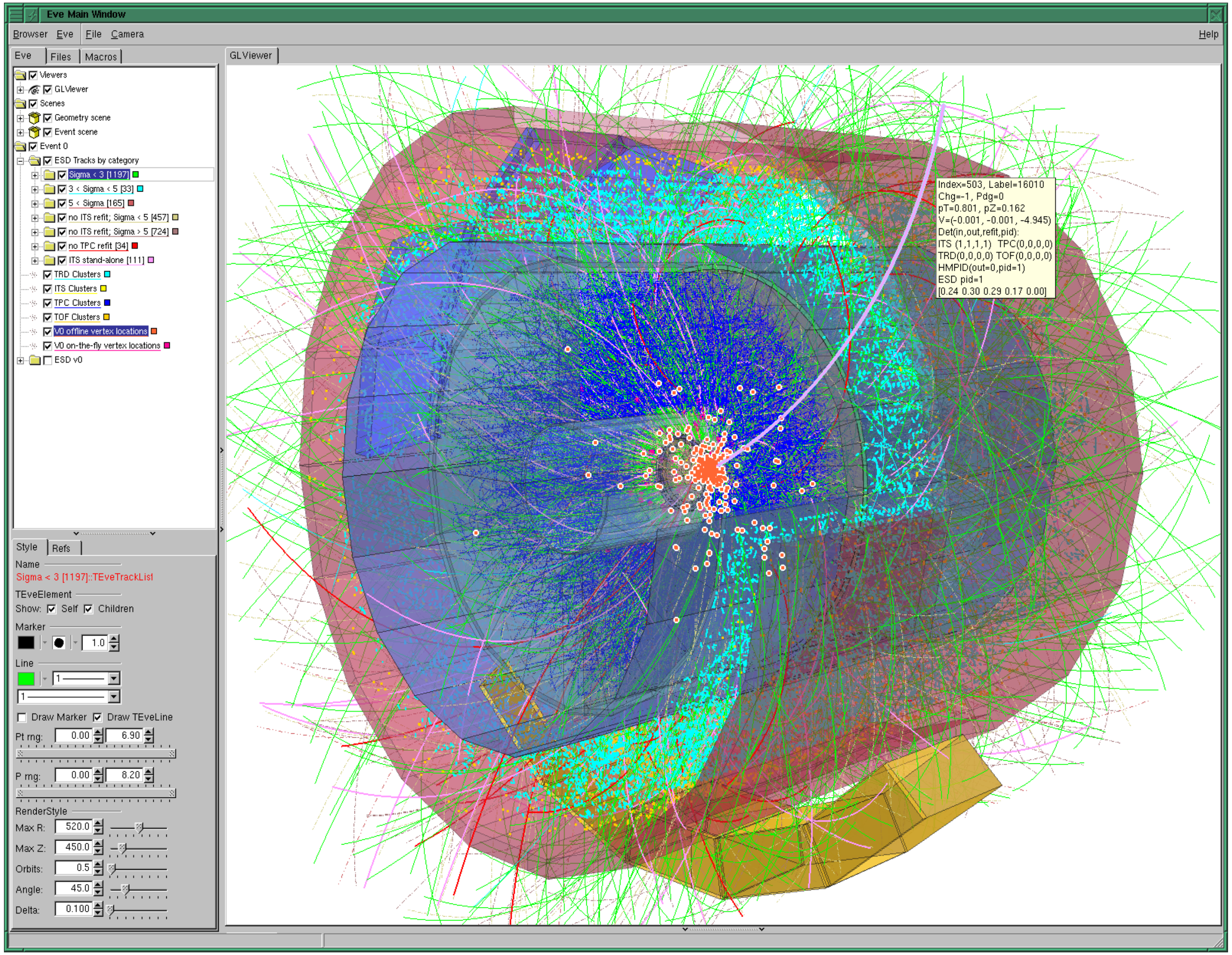
for each tree

- **Process(Long64_t entry)**

for each event

- **Terminate()**

ROOT Features: Data Analysis



- ☑ Viewers
  - ☑ GLViewer
- ☑ Scenes
  - ☑ Geometry scene
  - ☑ Event scene
- ☑ Event 0
  - ☑ ESD Tracks by category
    - ☑ Sigma < 3 [1197] 🟩
    - ☑ 3 < Sigma < 5 [33]
    - ☑ 5 < Sigma [165]
    - ☑ no ITS refit; Sigma < 5 [457]
    - ☑ no ITS refit; Sigma > 5 [724]
    - ☑ no TPC refit [34] 🟥
    - ☑ ITS stand-alone [111]
  - ☑ TRD Clusters
  - ☑ ITS Clusters
  - ☑ TPC Clusters
  - ☑ TOF Clusters
  - ☑ V0 offline vertex locations
  - ☑ V0 on-the-fly vertex locations
- ☐ ESD v0

Name

Sigma < 3 [1197]::TEveTrackList

TEveElement

Show: ☑ Self ☑ Children

Marker

1.0

Line

1

1

☐ Draw Marker ☑ Draw TEveLine

Pt rng: 0.00 6.90

P rng: 0.00 8.20

RenderStyle

Max R: 520.0

Max Z: 450.0

Orbits: 0.5

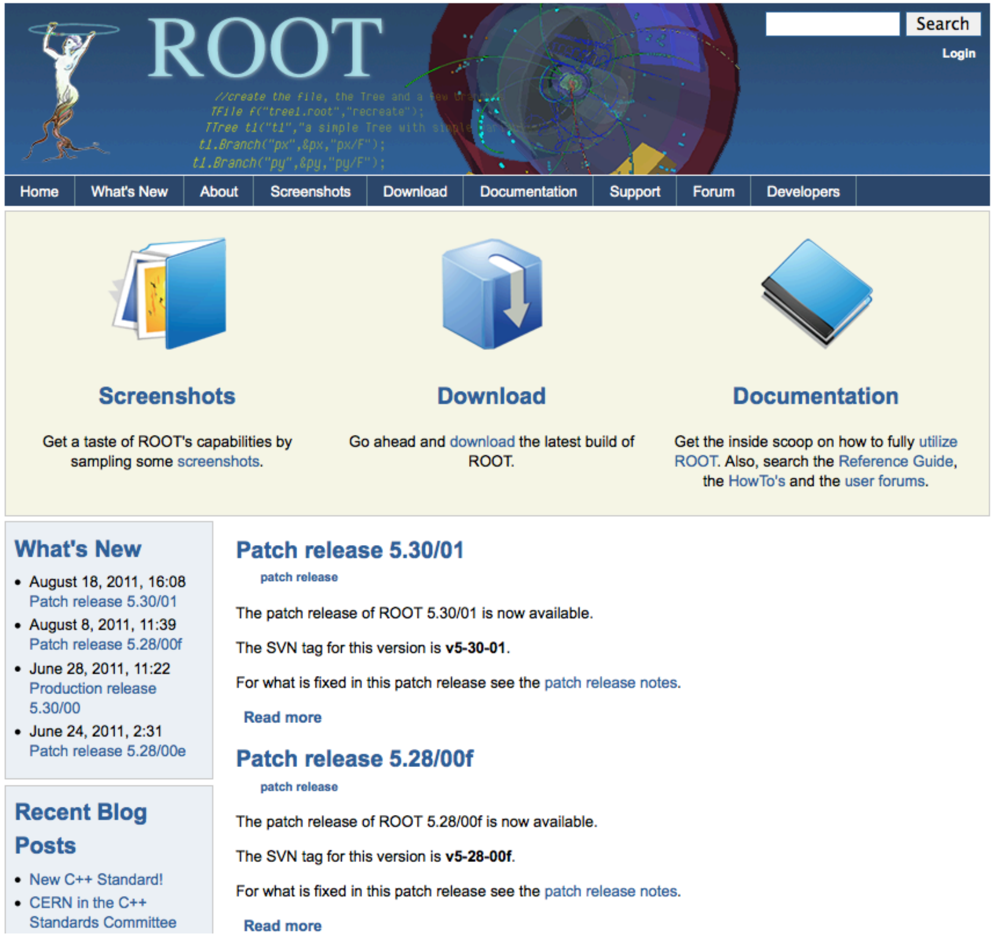
Angle: 45.0

Delta: 0.100

```
Index=503, Label=16010
Chg=-1, Pdg=0
pT=0.801, pZ=0.162
V=(-0.001, -0.001, -4.945)
Det(in,out,refit,pid):
ITS (1,1,1,1)  TPC(0,0,0,0)
TRD(0,0,0,0) TOF(0,0,0,0)
HMPID(out=0,pid=1)
ESD pid=1
[0.24 0.30 0.29 0.17 0.00]
```

More information on ROOT

- http://root.cern.ch
  - Download
    - binaries, source
  - Documentation
    - User's guide
    - Tutorials
    - FAQ
  - Mailing list
  - Forum



```
//create the file, the Tree and a few branches
TFile f("tree1.root","recreate");
TTree t1("t1","a simple Tree with simple
t1.Branch("px",&px,"px/F");
t1.Branch("py",&py,"py/F");
```

Screenshots

Get a taste of ROOT's capabilities by sampling some screenshots.

Download

Go ahead and download the latest build of ROOT.

Documentation

Get the inside scoop on how to fully utilize ROOT. Also, search the Reference Guide, the HowTo's and the user forums.

What's New

- August 18, 2011, 16:08
  Patch release 5.30/01
- August 8, 2011, 11:39
  Patch release 5.28/00f
- June 28, 2011, 11:22
  Production release 5.30/00
- June 24, 2011, 2:31
  Patch release 5.28/00e

Recent Blog Posts

- New C++ Standard!
- CERN in the C++ Standards Committee

Patch release 5.30/01

patch release

The patch release of ROOT 5.30/01 is now available.

The SVN tag for this version is **v5-30-01**.

For what is fixed in this patch release see the patch release notes.

Read more

Patch release 5.28/00f

patch release

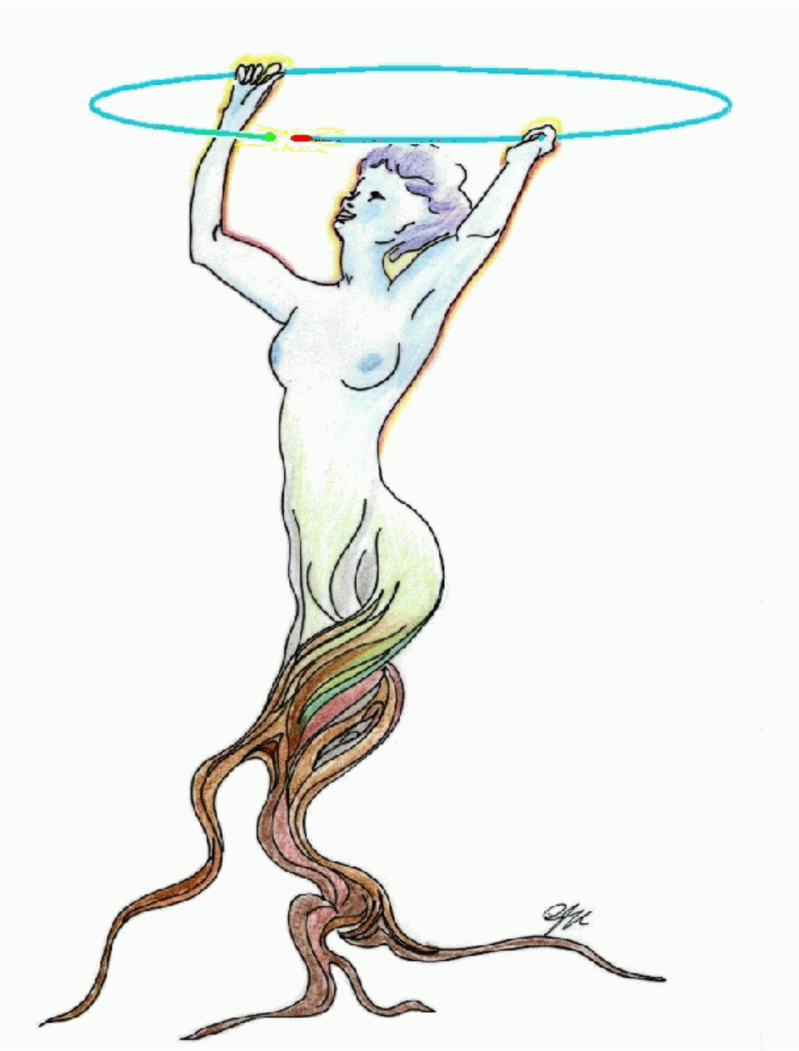
The patch release of ROOT 5.28/00f is now available.

The SVN tag for this version is **v5-28-00f**.

For what is fixed in this patch release see the patch release notes.

Read more

ROOT Tutorial



http://mon1.saske.sk/peac/doc/peac-tut/PEACTutorial_PROOFtutorial.html

http://root.cern.ch/drupal/content/peac

In this tutorial you will learn how to…

- Use CLI and GUI
- Create functions and histograms
  - Visualize (draw) them
- Create and explore files
- Create and explore trees
- Create chains
- Write a selector class
- Analyze data contained in trees and chains on your machine

Preparations for the tutorial

- Connect to your UI login server
  - Attention! Use –Y option for SSH:
    - e.g. *ssh –Y –p 24 gks098@gks-211.scc.kit.edu*
- Connect to machines gks-NNN.scc.kit.edu
    - e.g. *ssh –Y gs023@gks-032.scc.kit.edu*
    - *We will tell you the number of machine you should connect to*
    - *Verify that you have connected to proper machine running "hostname –f"*
- Run the following command:
    - source /opt/PEAC/sw/current/VO_PEAC/ROOT/v5-34-01/peac-env.sh

    It will set system paths to include ROOT binary and the libraries
- Start root:
    - *root*
  - You should see ROOT start screen with logo and the ROOT version: 5-34-01

Macros for tutorial

- Go to the page
  http://mon1.saske.sk/peac/doc/peac-tut/PEACTutorial.html
- Download the archive by the link specified in section 1.1, "Tutorials"
- Unpack the archive:

  $> tar -zxvf GridKa2012.tar.gz

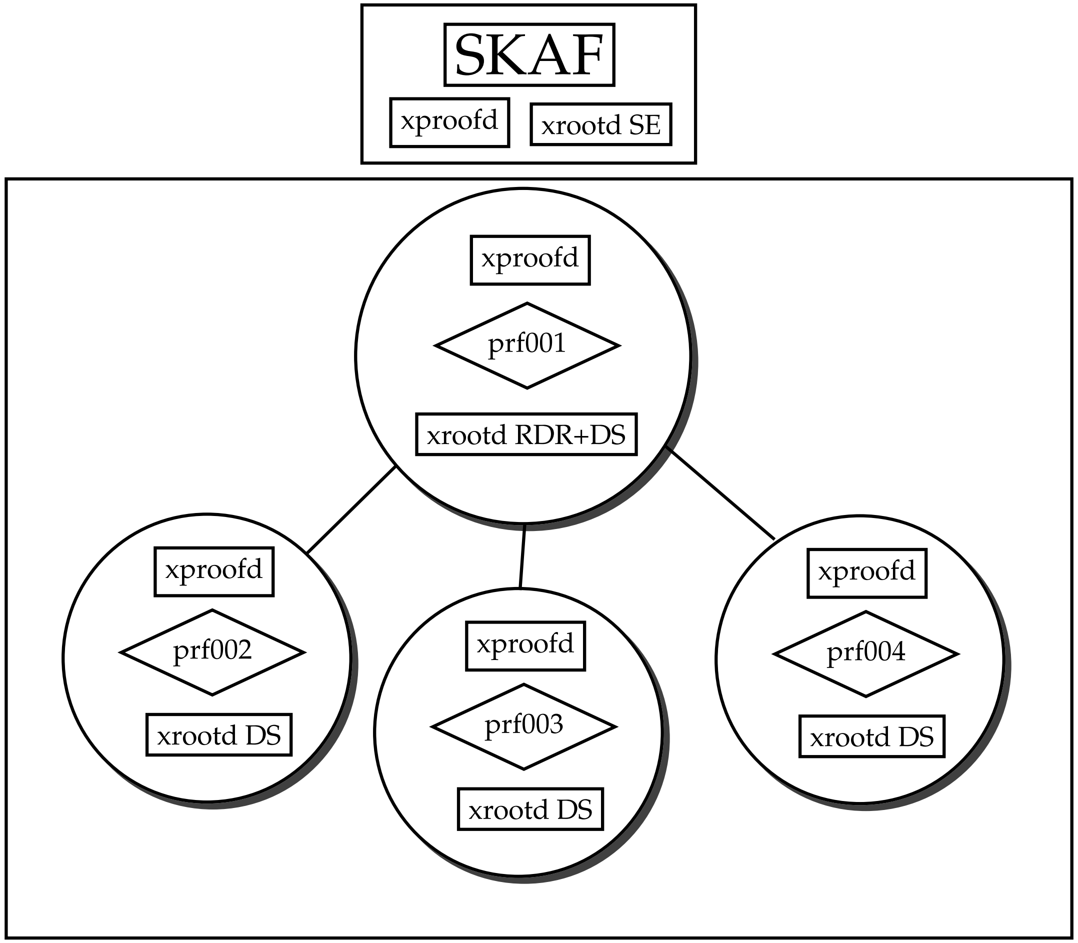
  Directory *GridKa2012* will be created containing tutorial macros.

We strongly recommend you to type the code you find at tutorial documentation page!

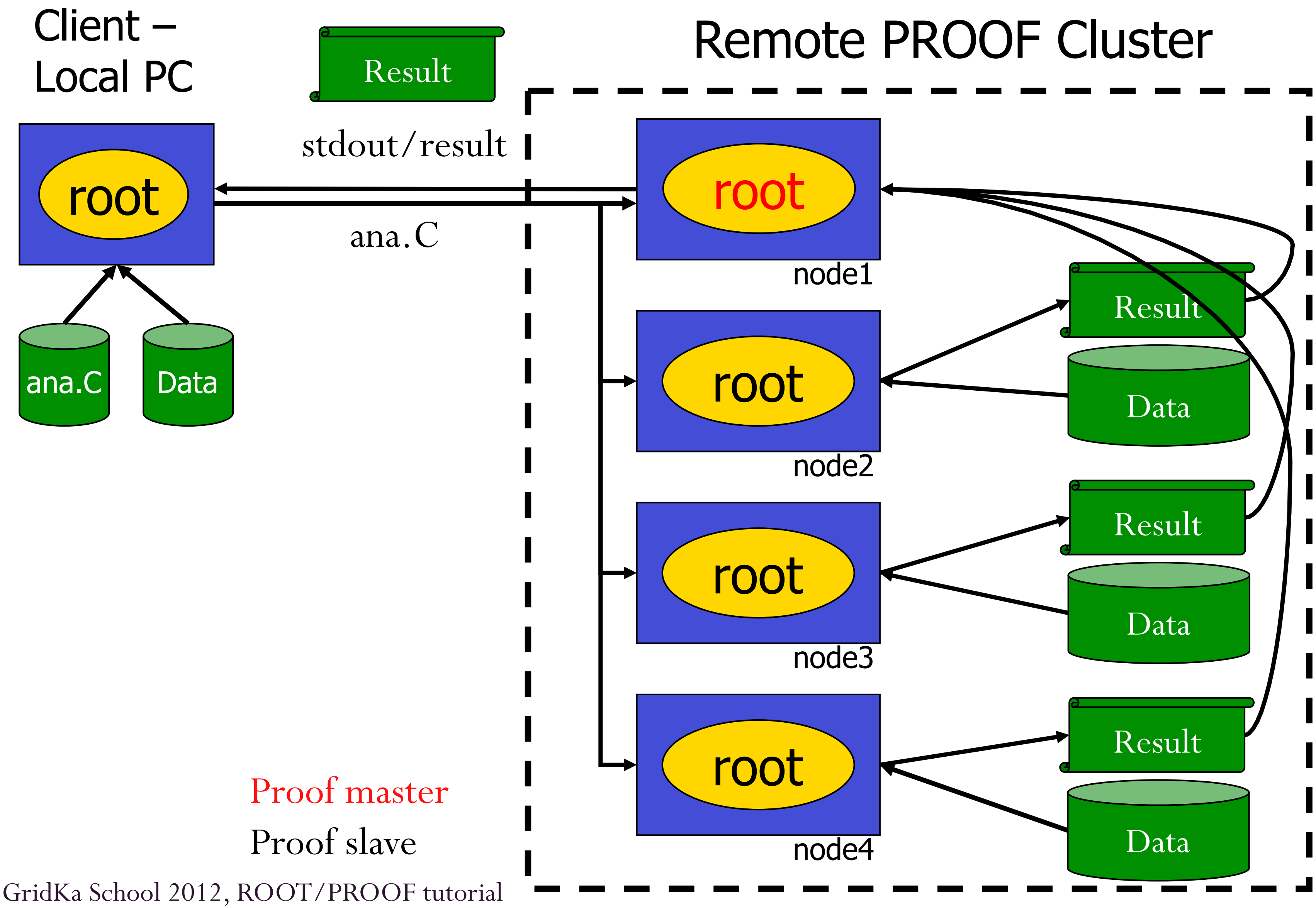
What is PROOF? Why PROOF?

- PROOF stands for **P**arallel **ROO**t **F**acility
- It allows parallel processing of large amount of data. The output results can be directly visualized (e.g. the output histogram can be drawn at the end of the proof session).
- PROOF is NOT a batch system.
- The data which you process with PROOF can reside on your computer, PROOF cluster disks or grid.
- The usage of PROOF is transparent: you should not rewrite your code you are running locally on your computer.
- No special installation of PROOF software is necessary to execute your code: PROOF is included in ROOT distribution.

How PROOF cluster works



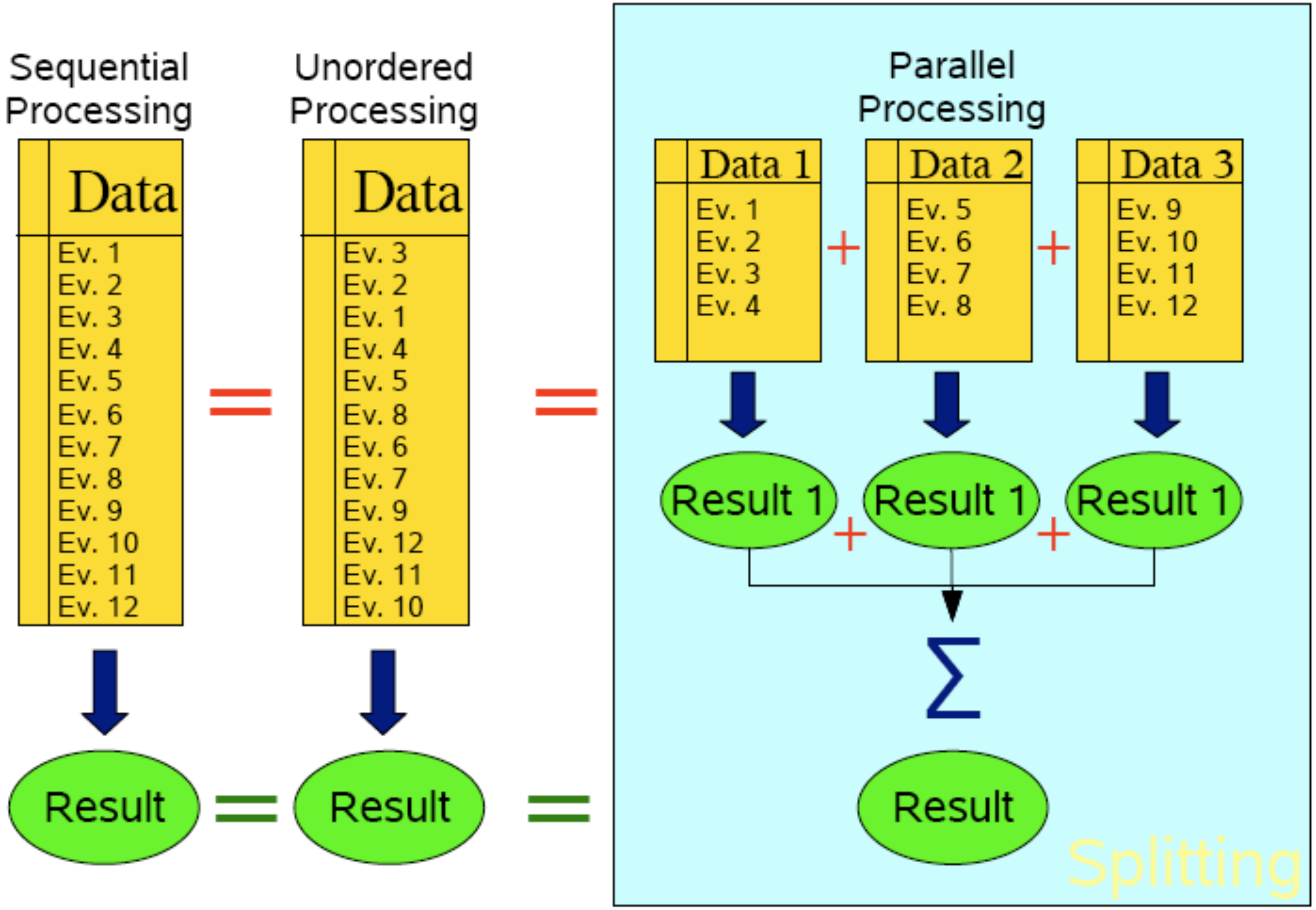
How does PROOF analysis work?



Proof master

Proof slave

Trivial parallelism



PROOF terminology

The following terms are used in PROOF:

- PROOF cluster
  - Set of machines communicating with PROOF protocol. One of those machines is normally designated as Master (multi-Master setup is possible as well). The rest of machines are Workers.
- Client
  - Your machine running a ROOT session that is connected to a PROOF master.
- Master
  - Dedicated node in PROOF cluster that is in charge of assigning workers the chunks of data to be processed, collecting and merging the output and sending it to the Client.
- Slave/Worker
  - Entity which processes portion of overall data split in packets. Every worker has its own root session controlled by proofserv.exe process.
- Query
  - A job submitted from the Client to the PROOF cluster.

    A query consists of a selector and a chain.
- Selector
  - A class containing the analysis code (more details later)
- Chain
  - A list of files (trees) to process (more details later)
- PROOF Archive (PAR) file
  - Archive file containing files for building and setting up a package on the PROOF cluster. Normally is used to supply extra packages used by user job.

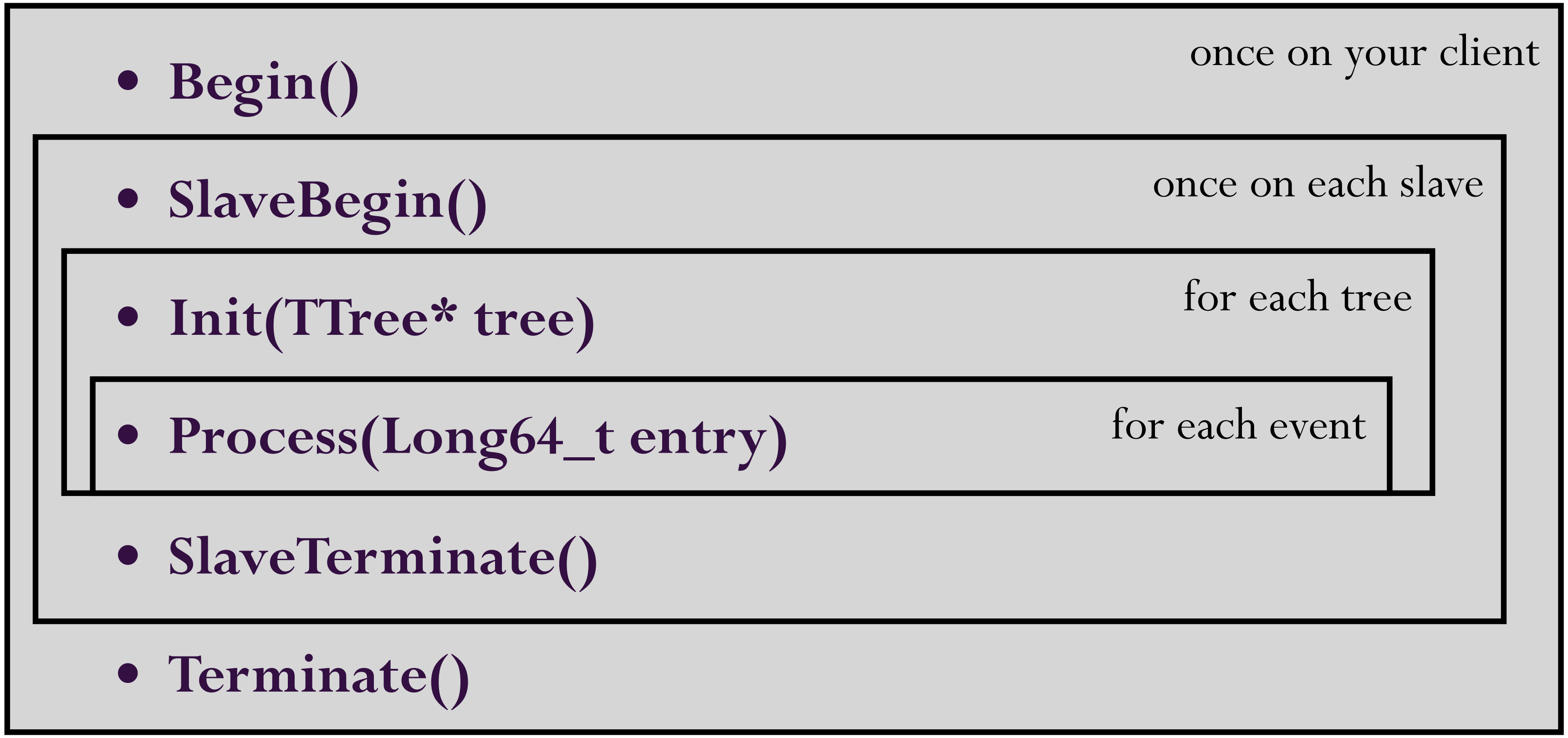
What should I do to run a job on PROOF cluster?

- Create a chain (dataset) containing the files you want to analyze.
- Write your job code and put it in the selector (class deriving from TSelector).
- Define inputs and outputs via predefined (by class TSelector) lists (TList objects) *fInput* and *fOutput*.
- Create extra packages (if any) which you need and put them in PAR file to be deployed on the PROOF cluster.

Selectors (Class TSelector)

PROOF analysys case

- Classes derived from TSelector can run in PROOF



once on your client

- **Begin()**

once on each slave

- **SlaveBegin()**

for each tree

- **Init(TTree* tree)**

for each event

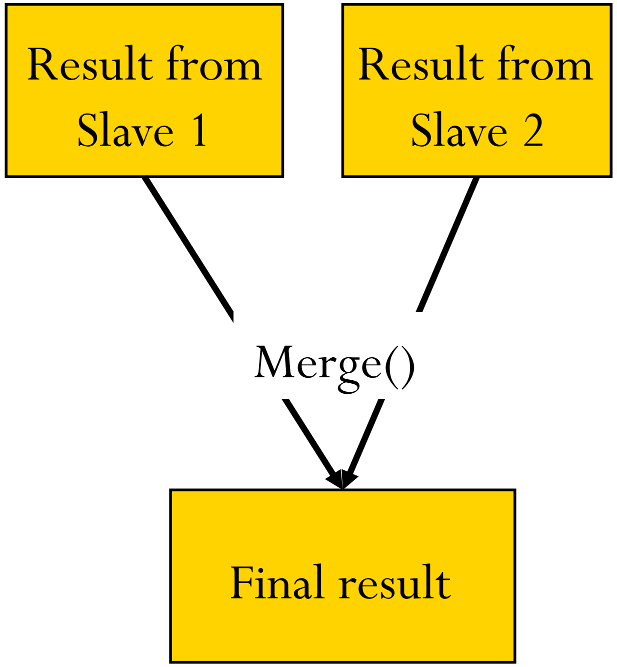
- **Process(Long64_t entry)**
- **SlaveTerminate()**
- **Terminate()**

Input / Output (1)

- Output list
  - The output has to be added to the output list on each slave (in **SlaveBegin/SlaveTerminate**) **fOutput->Add(fResult)**
  - PROOF merges the results from each slave automatically (see next slide)
  - On the client (in **Terminate**) you retrieve the object and save it, display it, or do any other operation on it: **fOutput->FindObject("myResult")**

Input / Output (2)

- Merging
  - Objects are identified by name
  - Standard merging implementation for histograms, trees, n-tuples available
  - Other classes need to implement **Merge(TCollection*)**
  - When no merging function is available all the individual objects are returned



The structure of the PAR files

- PAR files: **P**ROOF **AR**chive
  - Gzipped tar file
  - PROOF-INF directory
    - BUILD.sh, building the package, executed per Worker
    - SETUP.C, set environment, load libraries, executed per Worker
- API to manage and activate packages

```
gProof->UploadPackage("package.par")
gProof->EnablePackage("package")
```

Datasets

- A dataset represents a list of files
- Users register datasets
  - The files contained in a dataset are automatically copied from external storage (e.g. grid)
  - Datasets are used for processing with PROOF
    - Contain all relevant information to start processing (location of files, abstract description of content of files)
- Datasets are public for reading
- Dataset is a ***TFileCollection*** object

Running locally vs. PROOF Lite vs. PROOF

```
TProof::Open("gks-016.scc.kit.edu");
lite://");
TChain* ch = new TChain(<tree name>, <chain title>);
ch->AddFile("<file1.root>");
ch->AddFile("<file2.root>");
ch->AddFile("<file3.root>");
ch->SetProof();
ch->Process("MySelector.cxx+");
```

PROOF Tutorial

http://mon1.saske.sk/peac/doc/peac-tut/PEACTutorial_PROOFtutorial.html

http://root.cern.ch/drupal/content/peac

In this tutorial you will learn how to…

- Analyze on PROOF Lite
- Create PAR files
- Process data stored in dataset
- Generate data for analysis
- Analyze with PROOF

Installation of PROOF cluster

- Install root on all workers
- Start xproofd daemon
  - By hand
  - Using PoD
    - http://pod.gsi.de
  - Using PEAC (using SSH plugin from PoD)
- Start xrootd and cmsd daemons
  - Using PEAC data management setup (available soon)