LHC GRID COMPUTING SCHOOL
SANG UN AHN
SAHN@KISTI.RE.KR

PROOF & POD

TUTORIAL

TUTORIAL OUTLINE

- Part I
  - Introduction to PROOF
- Part II
  - PROOF Basics (based on PoD @ KIAF)

AIM OF THE TUTORIAL

- The aim is twofold
  - Introduce the concept and basic of PROOF to you
  - Teach you where to find when you need something
- Documentation always helps
- Personal support is available at

sahn@kisti.re.kr

(or sang.un.ahn@cern.ch)

(or realapyo@gmail.com)

PART I

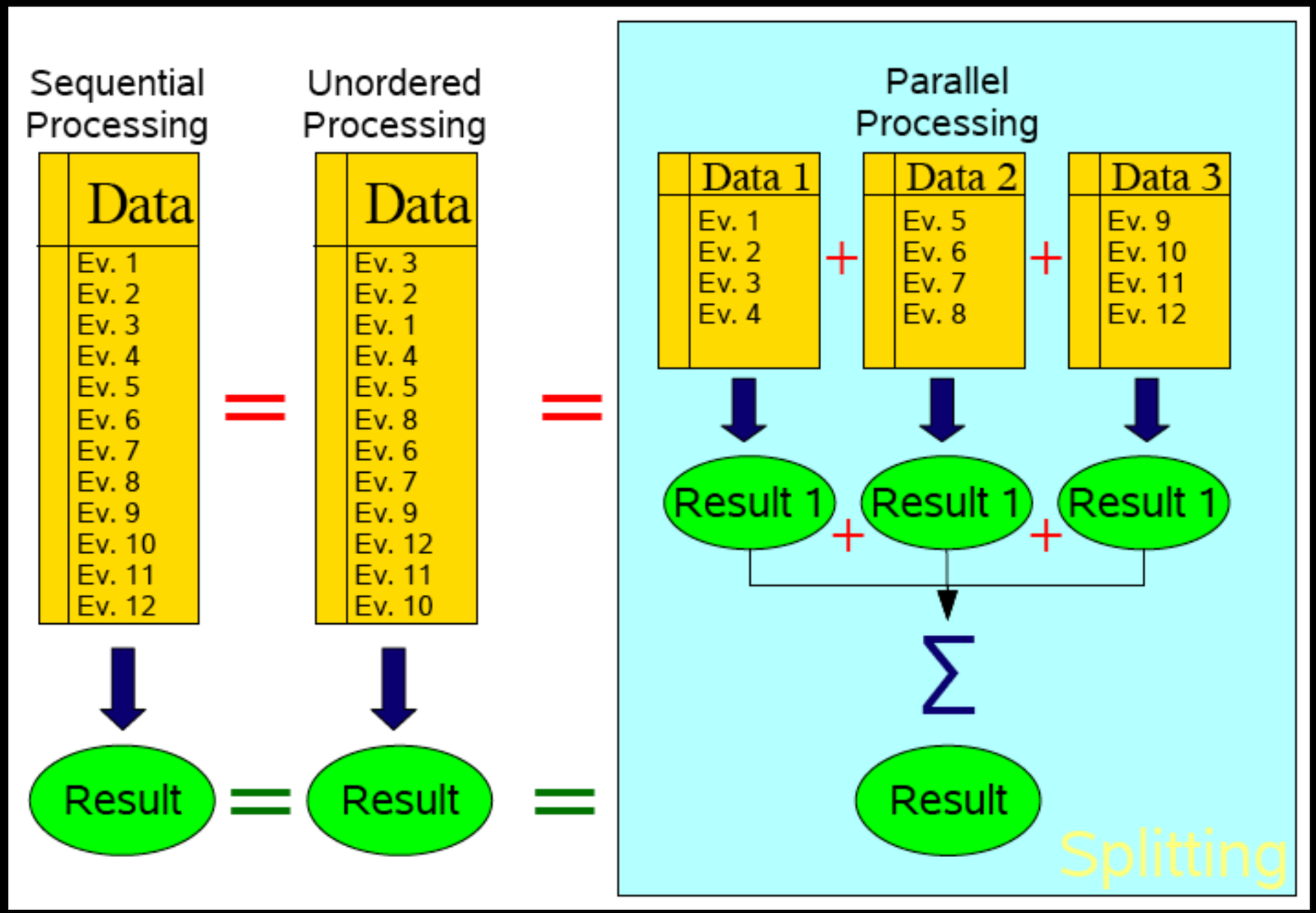
PROOF INTRODUCTION

PROOF

- **P**arallel **ROO**T **F**acility
  - Parallel coordination of distributed ROOT sessions
  - A software that enables parallel processing of data analysis task either on desktop/laptop or on a localhost or a cluster
- ROOT?
  - Popular and basic tool coded in C/C++ for data analysis in science
  - For more information, visit http://root.cern.ch
- Parallel?

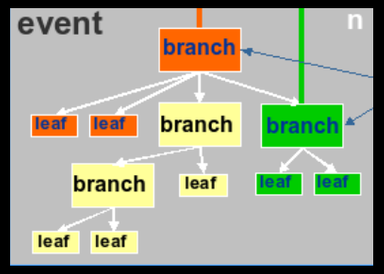
PARALLELISM-IDEAL

Typically **embarrassingly parallel tasks**: just split to get ideal parallel speedup



EVENT-LEVEL PARALLELISM:

TSelector framework



Events

1

2

n

last

Begin()

- Create histos, …
- Define output list

Process()

preselection

OK

analysis

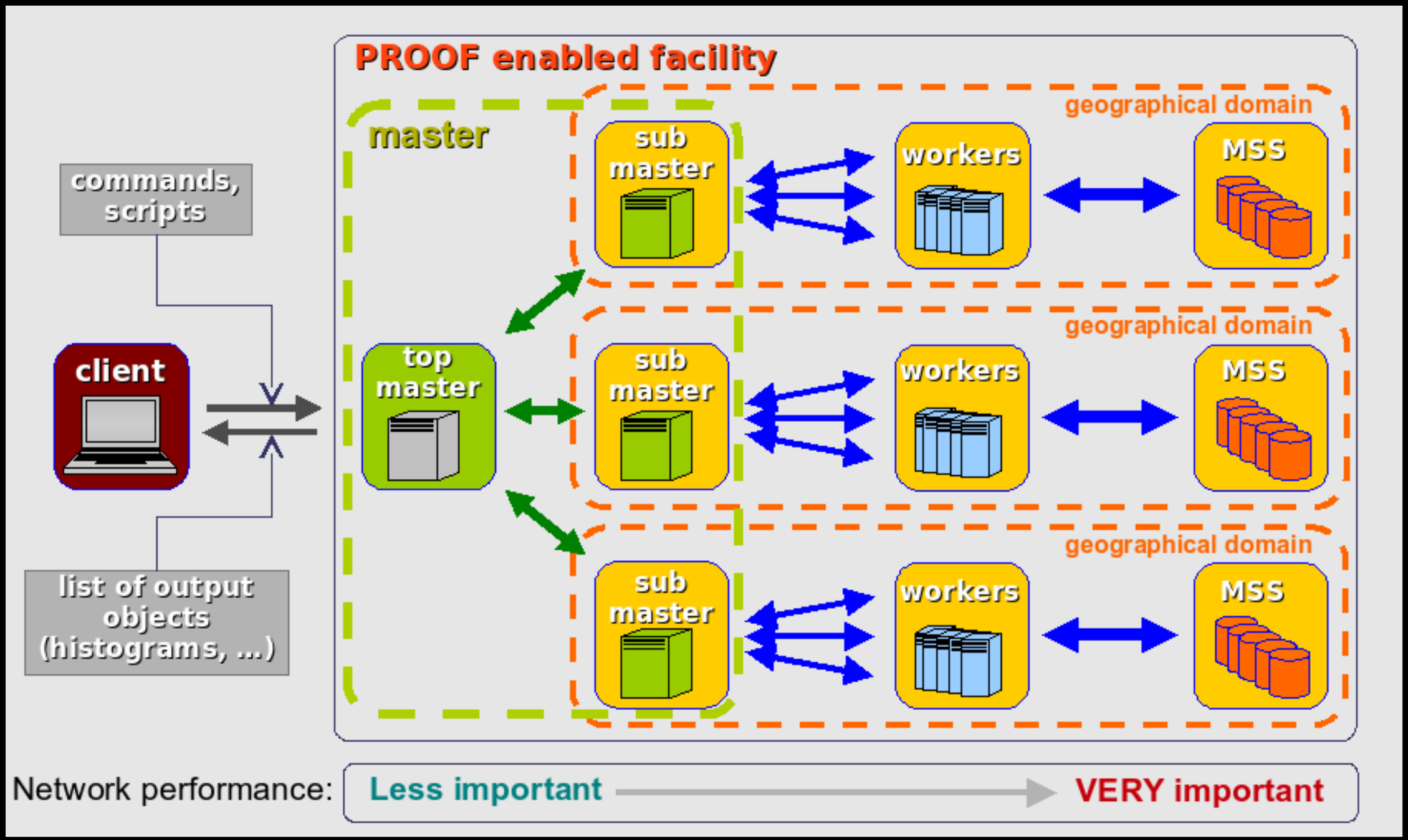
Output List

PARALLELIZABLE EVENT LOOP

Terminate()

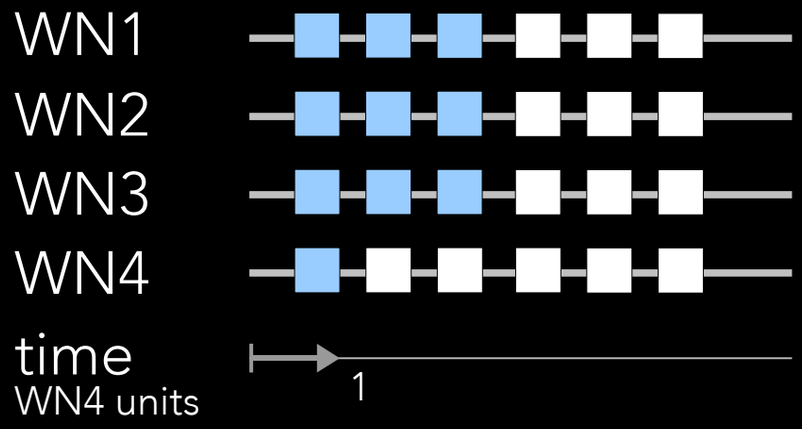
- Final analysis, fitting, …

PROOF ARCHITECTURE



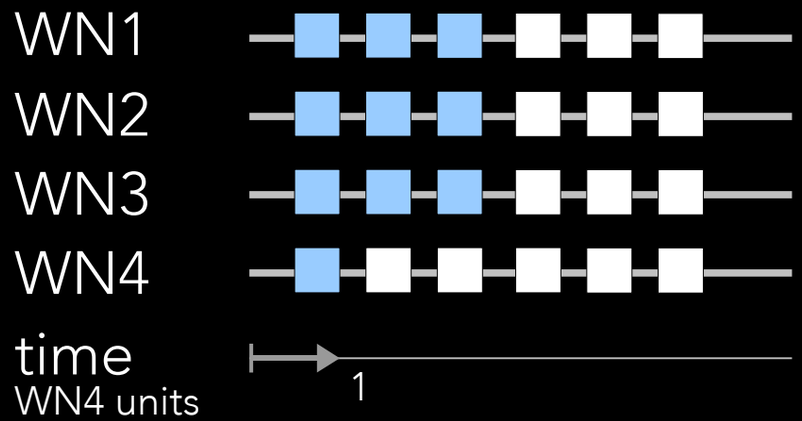
LOAD BALANCING: STATIC

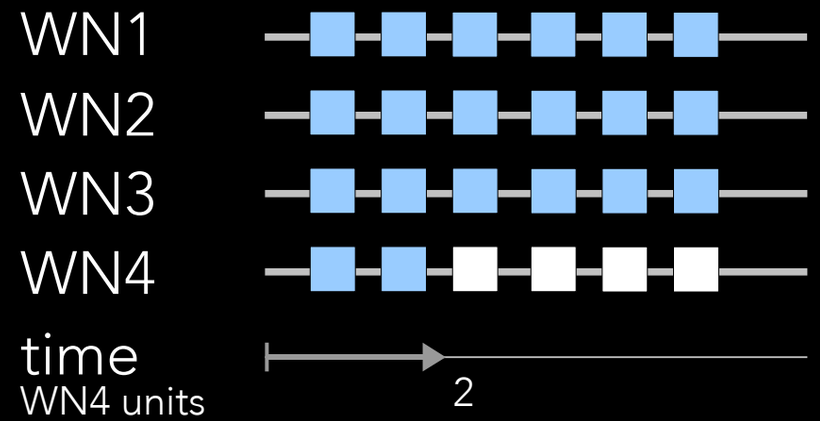
Example: 24 files on 4 worker nodes, one under-forming



LOAD BALANCING: STATIC

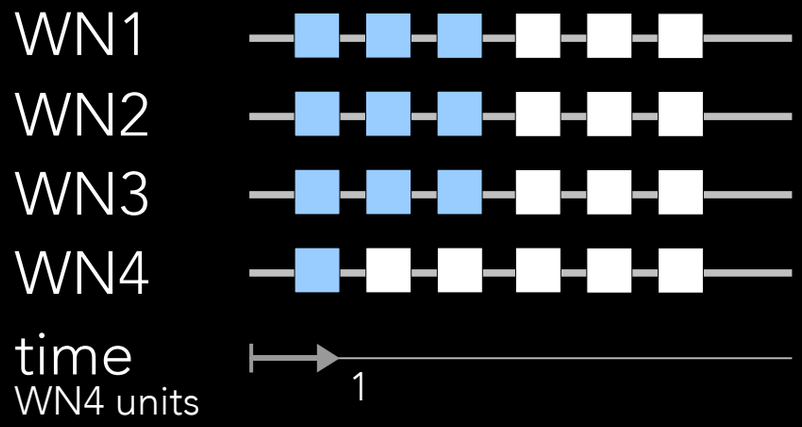
Example: 24 files on 4 worker nodes, one under-forming

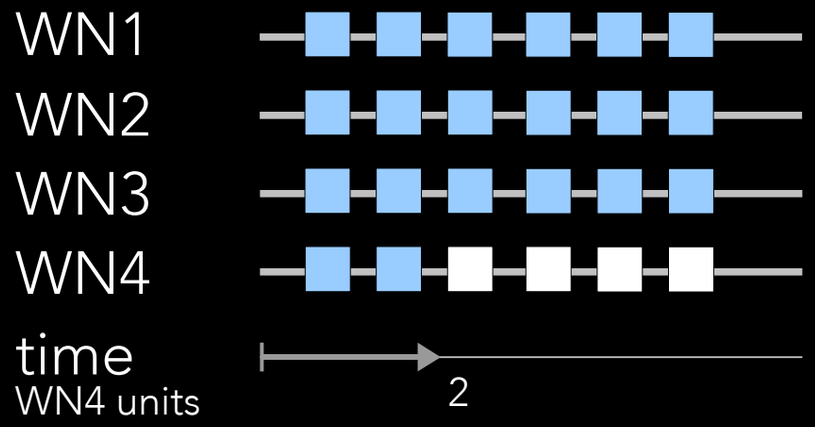


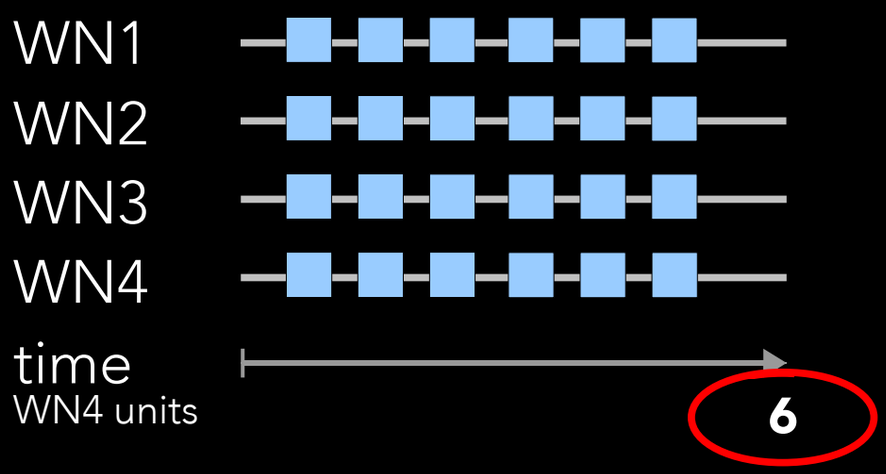


LOAD BALANCING: STATIC

Example: 24 files on 4 worker nodes, one under-forming

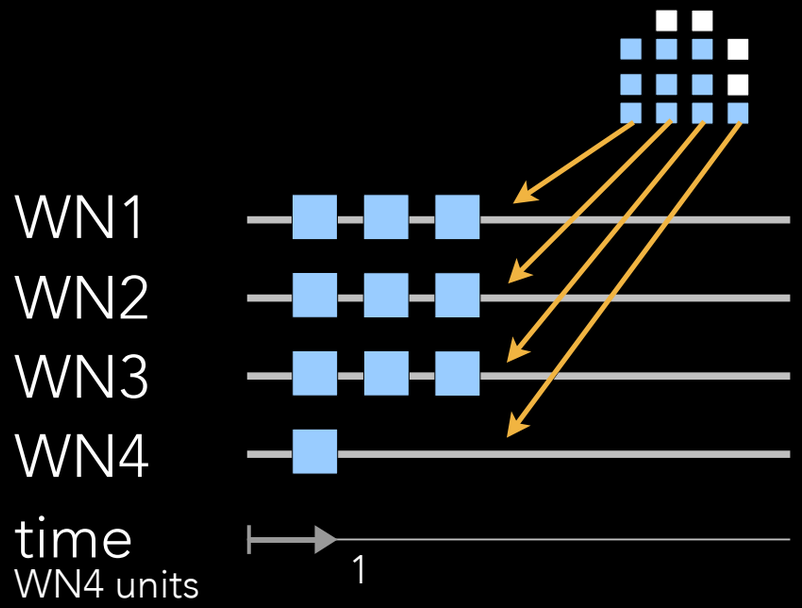




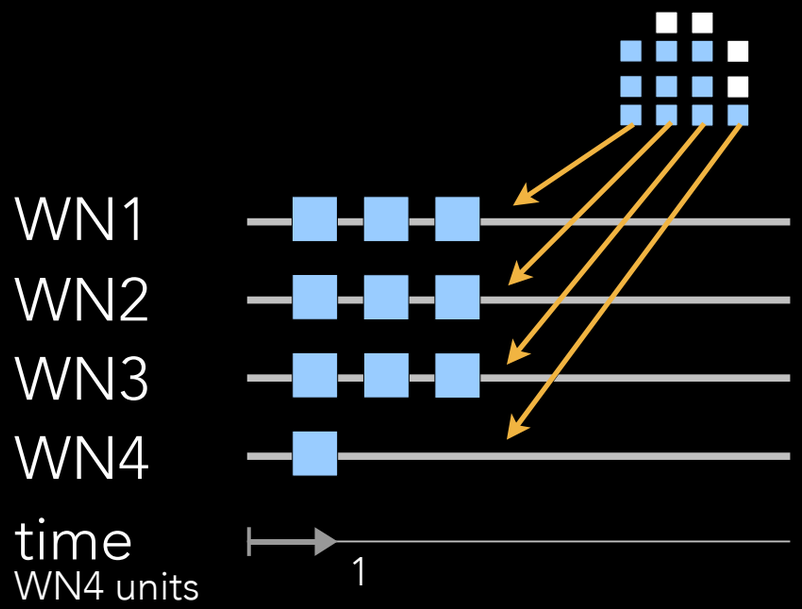


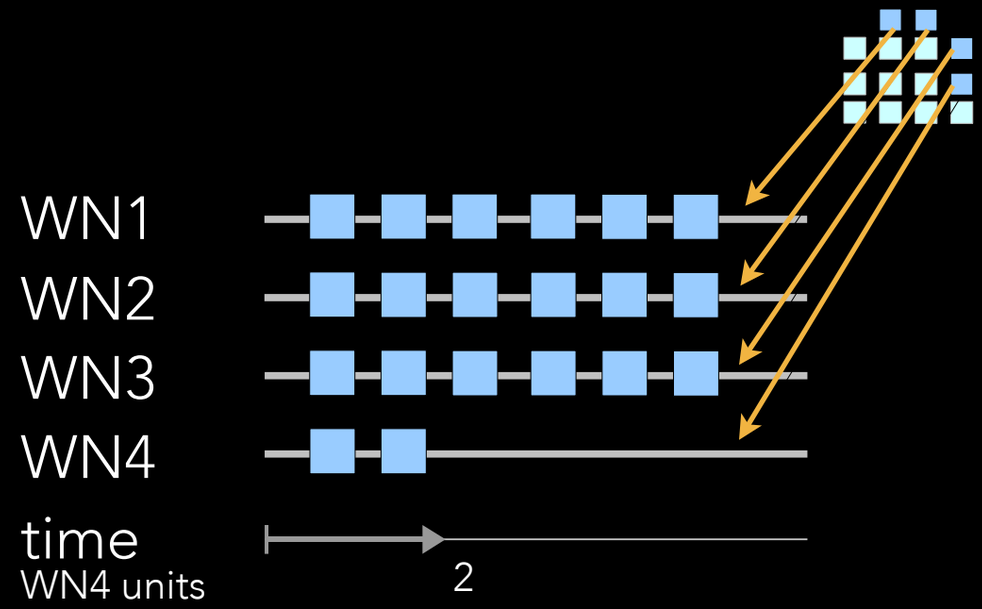
The slowest worker node sets the processing time

LOAD BALANCING: DYNAMIC

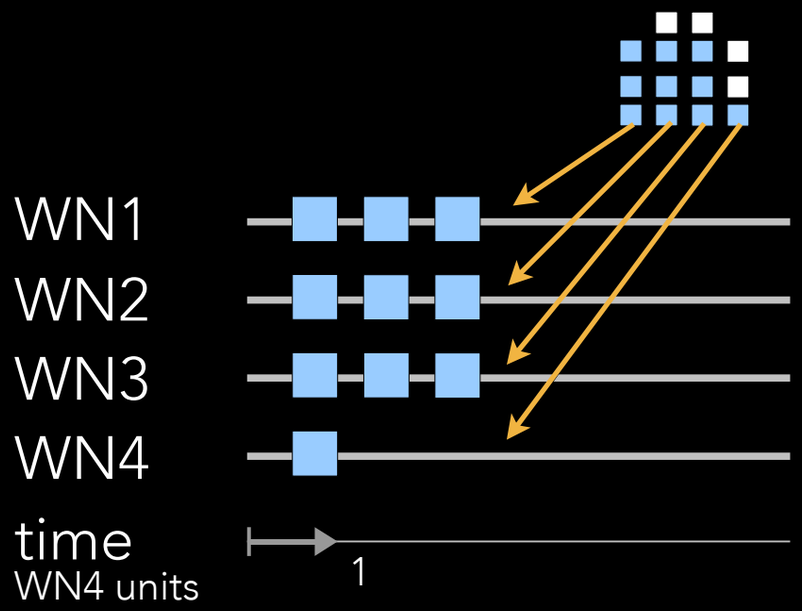


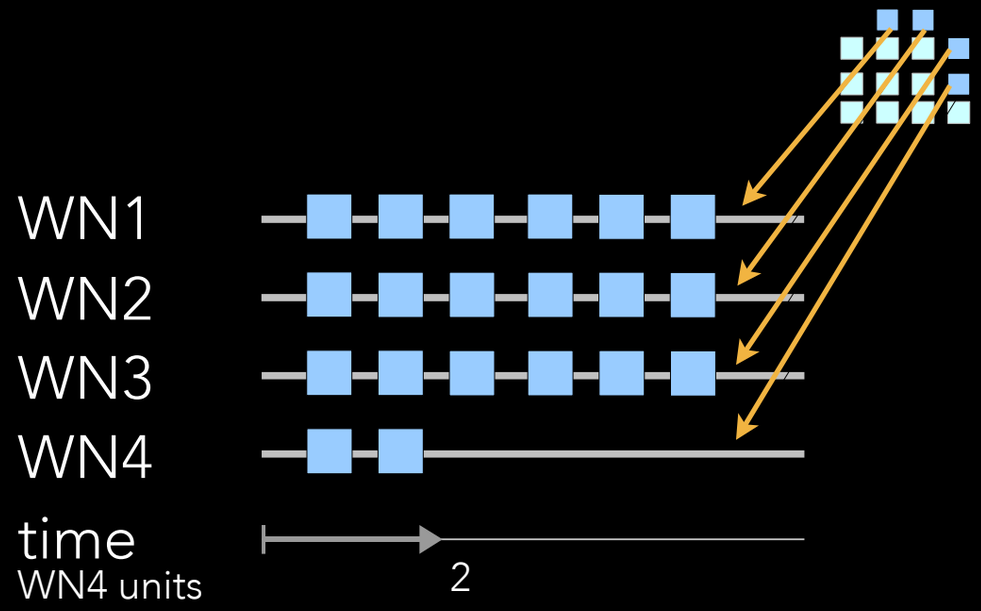
LOAD BALANCING: DYNAMIC

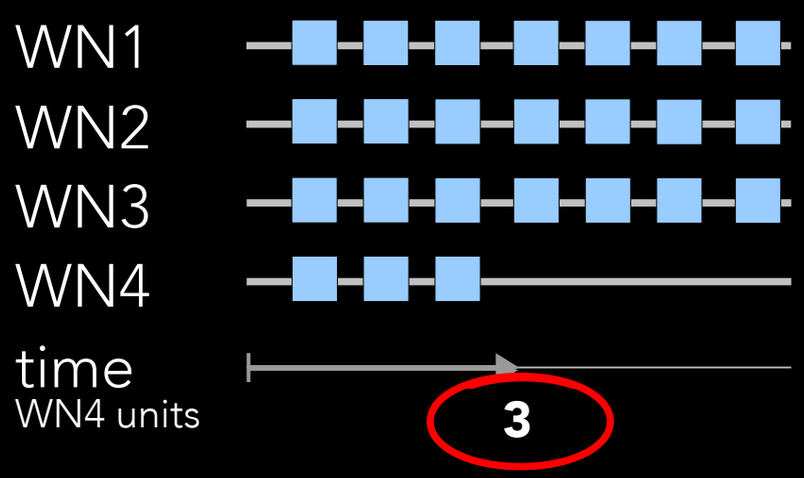




LOAD BALANCING: DYNAMIC





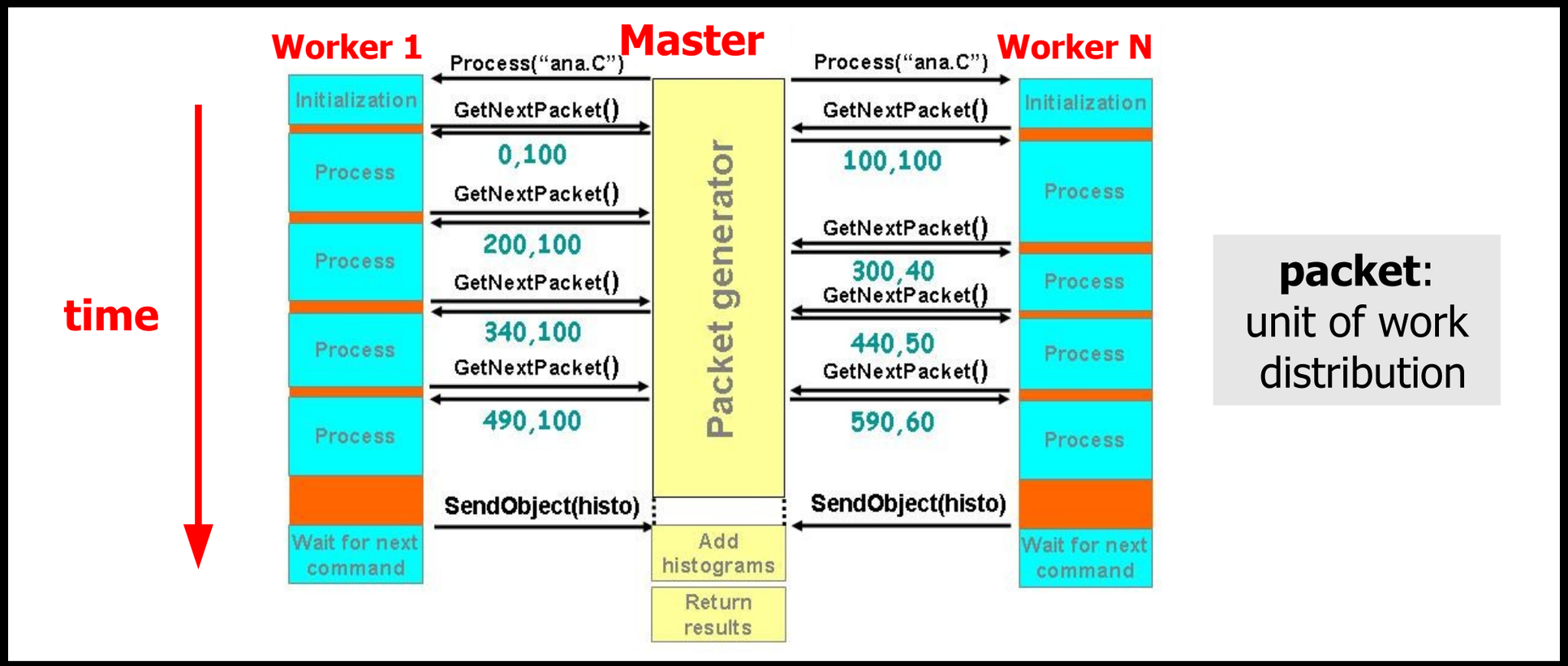


The slowest worker node gets less work to do: the processing time is less affected by its under performance

PULL ARCHITECTURE

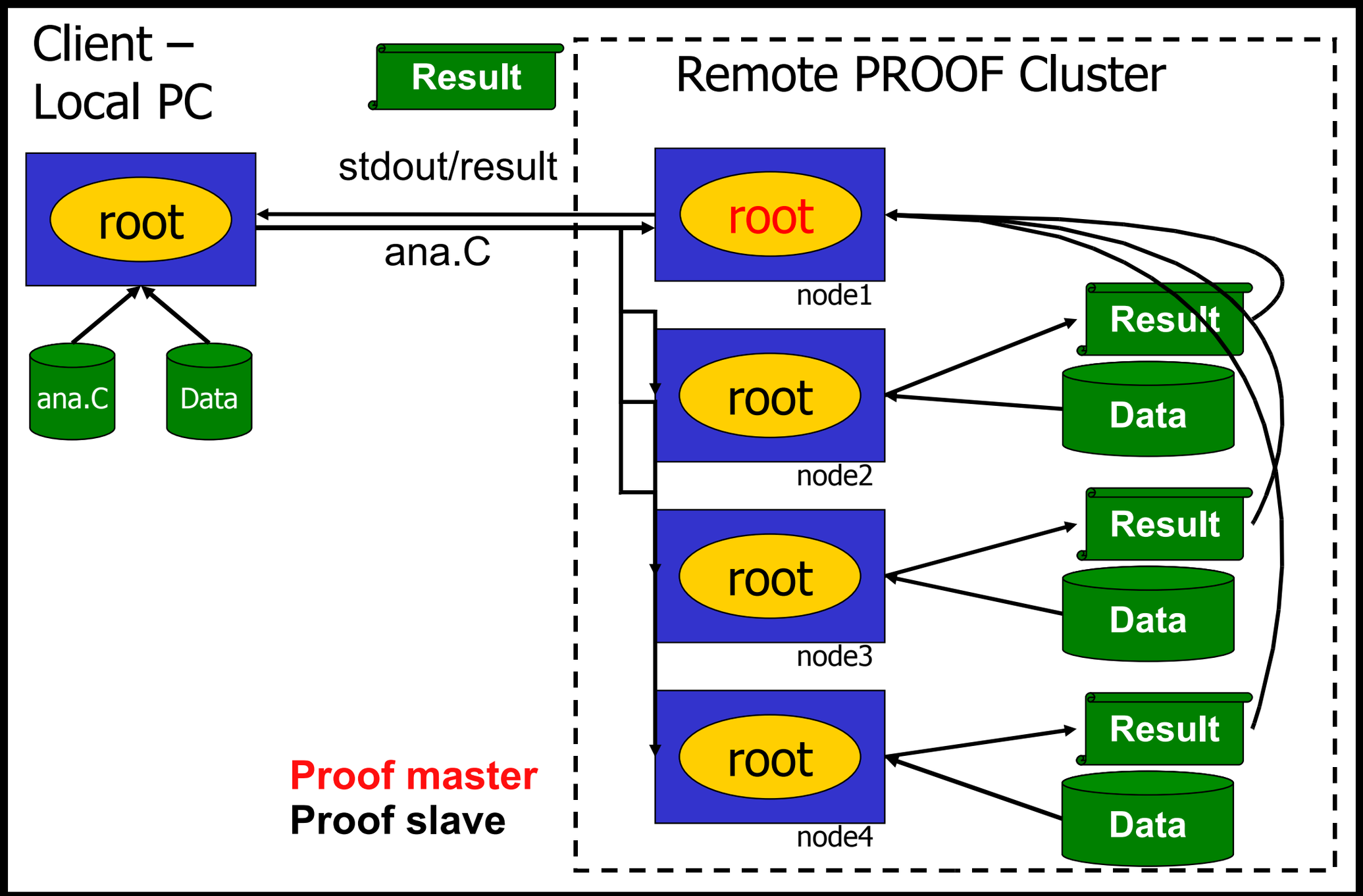
- Let the workers ask for more work when idle
  - Adapt dynamically the work load to their speed
- The unit of work is called **packet**: a range of events
- **Packetizer** is the heart of the system
  - Controls packet generation and distribution
    - Measures real-time worker performance to use it in the determination of the next packet size

DYNAMIC LOAD BALANCING



- Pull architecture guarantees scalability
- Adapts to performance variations

WORKFLOW


Client –
Local PC
Result
Remote PROOF Cluster
stdout/result
ana.C
root
root
node1
root
node2
root
node3
root
node4
Result
Data
Result
Data
Result
Data
ana.C
Data
Proof master
Proof slave

MERGING OUTPUT OBJECTS

- Standard ROOT objects provide **Merge** method

```
Long64_t Merge(TCollection *objectsToBeMerged)
```

- PROOF automatically merges standard ROOT objects
  - `TH1, TH2, TH3, TTree, …`
- If **Merge** method is not found, N objects received by the workers are just sent back to the client as they are

FEATURES

- Transparent to ROOT
  - Basically PROOF can run the ROOT code that can be run on localhost
- Interactive parallel execution of independent tasks
- Interactive-Batch
  - Can leave a processing session in the background, disconnect and reconnect later on to check the result
- Real-time Feedback

WHERE TO USE PROOF

- CAF, e.g. KIAF
- Workgroup facilities
- Multi-core desktops/laptops

RECAP

- PROOF features dynamic load balancing
  - Pull architecture
  - Event-level parallelism
    - Automatic splitting / merging
- PROOF is transparent to ROOT
  - The same code can be run locally and in PROOF

TERMINOLOGY

- Client
  - Machine running a ROOT session opening the connection to the PROOF master
- Master
  - PROOF machine running a ROOT session coordinating the work between workers and merging the results
- Worker (or Slave)
  - PROOF machine running a ROOT application doing the actual work
- Query
  - Process request submitted by the client to the Master
- Package / PAR file
  - Additional code needed by the selector but 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
- Chain
  - Instance of `TChain` with the list of files containing the `TTree` to be processed
- Dataset, TFileCollection
  - list of files containing the `TTree` to be processed

REFERENCES

- TWiki pages
  - http://root.cern.ch/drupal/content/proof
- ROOT site
  - http://root.cern.ch/root/html/ClassIndex.html
- ROOT forum
  - http://root.cern.ch/phpBB2/index.php
- This tutorial
  - https://root.cern.ch/drupal/content/proof-hands-tutorial
  - And Gerardo Ganis (CERN)’s ROOT tutorial

PART II

PROOF BASICS

PREREQUISITE

- Xming Setup for X11 forwarding
- X11 forwarding enabling at Putty

STARTING PROOF@KIAF

- Connect UI
- SSH to `kiaf.sdfarm.kr`
- Use your credentials (ID/PW)
- Setup for ROOT environment
  - Just run

```
$ source runFirst.sh
```

KIAF

- KISTI Analysis Facility
  - Full functionality is available for ALICE user only
    - Requires grid authentication
    - E.g. dataset registration
- Running PROOF code is still allowed w/o authentication

KIAF SPECIFICATION

- Structure
  - 1 Master
  - 8 Slaves
    - 24 workers per slave provides 192 workers in total
- Resource Management System
  - HTCondor
    - Common HTCondor tool is available

```
$ condor_status
$ condor_q
```

- **PROOF on Demand** (setup of PoD is out of scope from this tutorial)
- Xrootd enabled for dataset management (out of scope)

PoD

- PROOF on Demand
  - Tool-set to setup PROOF on any **R**esource **M**anagement **S**ystem
  - Developed at GSI, Darmstadt, Germany
    - Author: Anar Manafov (A.Manafov@gsi.de)
    - Documented at http://pod.gsi.de
    - Plug-in based interaction with RMS
      - gLite, LSF, PBS, OGE, Condor, LoadLeveler
      - SSH plug-in for PW-less SSH connected clusters
- We will use PoD in this tutorial

POD BASIC

- PoD server is managed by pod-server command
  - pod-server -h
  - pod-server start | stop | restart
- Workers are claimed by pod-submit command
  - pod-submit -h
  - pod-submit -r condor -n 10
- Information is retrieved by pod-info command, e.g. number of workers
  - pod-info -n

START PROOF ON DEMAND

```
$ source runFirst.sh
$ pod-server start
…
$ pod-submit -r condor -n 10
…
$ pod-info -n
10
```

OPEN A PROOF SESSION

```
$ root -l
root [0] TProof *p = TProof::Open(“pod://")
Starting master: opening connection ...
Starting master: OK
Opening connections to workers: OK (10 workers)
Setting up worker servers: OK (10 workers)
PROOF set to parallel mode (10 workers)
(class TProof*)0x1e925b00
root [1]
```

OR

```
$ root -l
root [0] TProof *p = TProof::Open(“")
 +++ Starting PROOF-Lite with 24 workers +++
Opening connections to workers: OK (24 workers)
Setting up worker servers: OK (24 workers)
PROOF set to parallel mode (24 workers)
(class TProof*)0x15c9e2a0
root [1]
```

OPEN A PROOF SESSION

```
$ root -l
root [0] TProof *p = TProof::Open(“pod://")
Starting master: opening connection ...
Starting master: OK
Opening connections to workers: OK (10 workers)
Setting up worker servers: OK (10 workers)
PROOF set to parallel mode (10 workers)
(class TProof*)0x1e925b00
root [1]
```

OR

WHAT IS DIFFERENCE?

```
$ root -l
root [0] TProof *p = TProof::Open(“")
 +++ Starting PROOF-Lite with 24 workers +++
Opening connections to workers: OK (24 workers)
Setting up worker servers: OK (24 workers)
PROOF set to parallel mode (24 workers)
(class TProof*)0x15c9e2a0
root [1]
```

OPEN A PROOF SESSION

```
$ root -l
root [0] TProof *p = TProof::Open(“pod://")
Starting master: opening connection ...
Starting master: OK
Opening connections to workers: OK (10 workers)
Setting up worker servers: OK (10 workers)
PROOF set to parallel mode (10 workers)
(class TProof*)0x1e925b00
root [1]
```

PoD Farm URL

OR

```
$ root -l
root [0] TProof *p = TProof::Open(“")
 +++ Starting PROOF-Lite with 24 workers +++
Opening connections to workers: OK (24 workers)
Setting up worker servers: OK (24 workers)
PROOF set to parallel mode (24 workers)
(class TProof*)0x15c9e2a0
root [1]
```

PROOF-Lite

TPROOF: THE PROOF SHELL

```
$ root -l
root [0] TProof *p = TProof::Open(“pod://")
Starting master: opening connection ...
Starting master: OK
Opening connections to workers: OK (10 workers)
Setting up worker servers: OK (10 workers)
PROOF set to parallel mode (10 workers)
(class TProof*)0x1e925b00
root [1]
```

TProof: the PROOF shell

- Interface class
- Everything on the session will be through the TProof class methos
  - Print(), Exec(), AddInput(), Process(), GetOutputList(), DrawSelect(), …

gProof: global pointer to the latest TProof intance

TRY IT OUT:PRINT()

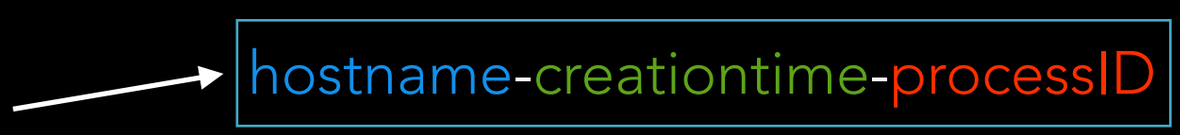
```
root [0] p->Print()
*** PROOF-Lite cluster (parallel mode, 10 workers):
Host name:                     afmaster01.sdfarm.kr
User:                          gcs-test
ROOT version|rev|tag:          5.34/08|r49361
Architecture-Compiler:         linuxx8664gcc-gcc412
Protocol version:              35
Working directory:             /home/gcs-test/.proof/analysis
Communication path:            /tmp/plite-28443
Log level:                     0
Number of workers:             10
Number of active workers:      10
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
root [1]
```

SANDBOX

- Working space
    - `$HOME/.proof` (default location)
    - PROOF-Lite: `$HOME/.proof/path-from-where-we-started`
- 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

SANDBOX

- Working space
  - `$HOME/.proof` (default location)
  - PROOF-Lite: `$HOME/.proof/path-from-where-we-started`
- 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



TRY IT OUT:EXEC()

```
root [0] p->Exec(“.!ls",kTRUE)
analysis  data    last-lite-session    packages  session-
afmaster01-1422340675-25648  session-
afmaster01.sdfarm.kr-1422340416-24431
cache      datasets  last-master-session  queries   session-
afmaster01-1422340998-26320  session-
afmaster01.sdfarm.kr-1422340884-25980
(Int_t)0
root [1]
```

SWITCHING MODE:
PARALLEL ⟷ SEQUENTIAL

```
root [9] p->SetParallel(0)
PROOF set to sequential mode
PROOF set to sequential mode
(Int_t)0
root [10] gProof->SetParallel(99999)
PROOF set to parallel mode (24 workers)
PROOF set to parallel mode (24 workers)
(Int_t)24
root [11]
```

SIMPLE TASK PROCESSING

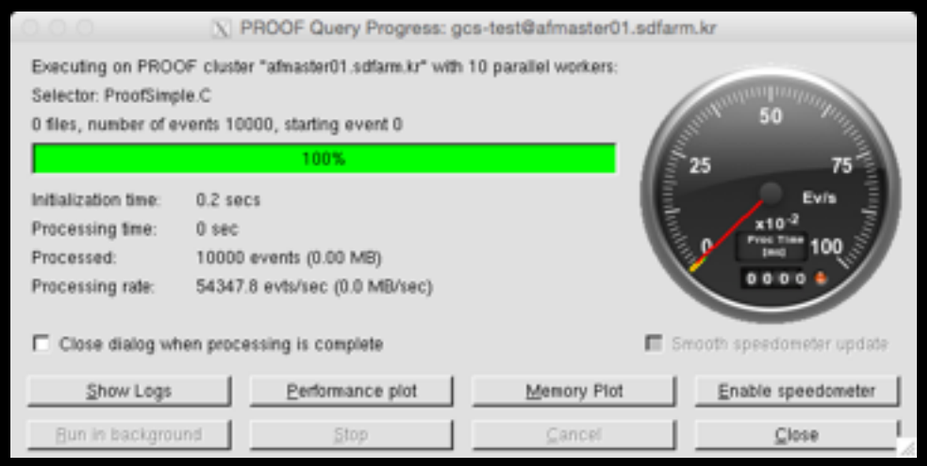
- We are ready to run a first query
- `tutorial/ProofSimple.C`
  - Defines a `TSelector` which fills 100 histograms with gaussian random numbers
- Just do

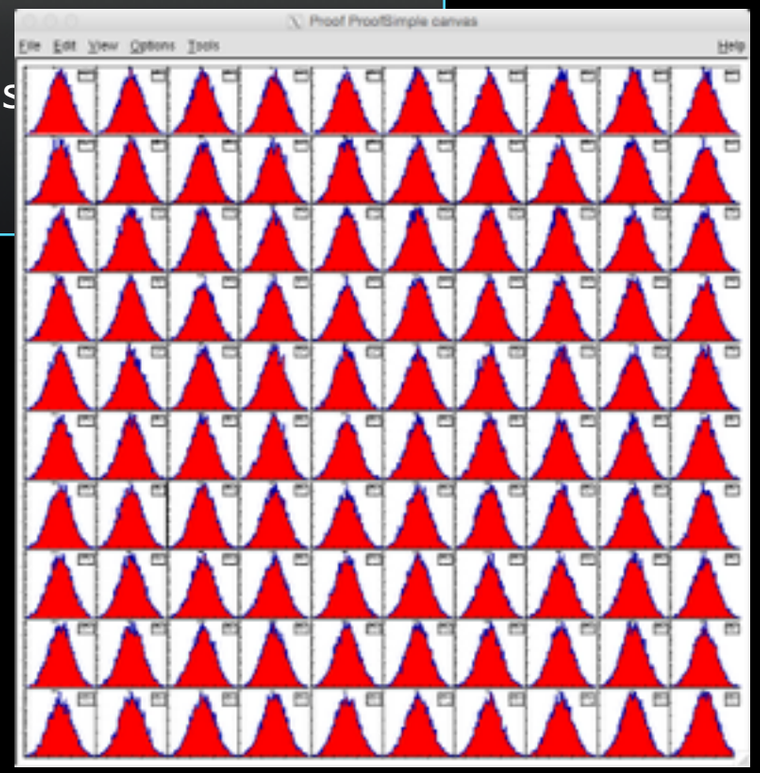
```
root [1] p->Process(“tutorial/ProofSimple.C+",10000)
Mst-0: merging output objects ... done
Mst-0: grand total: sent 107 objects, size: 93962 bytes
(Long64_t)0
root [2]
```

SIMPLE TASK PROCESSING

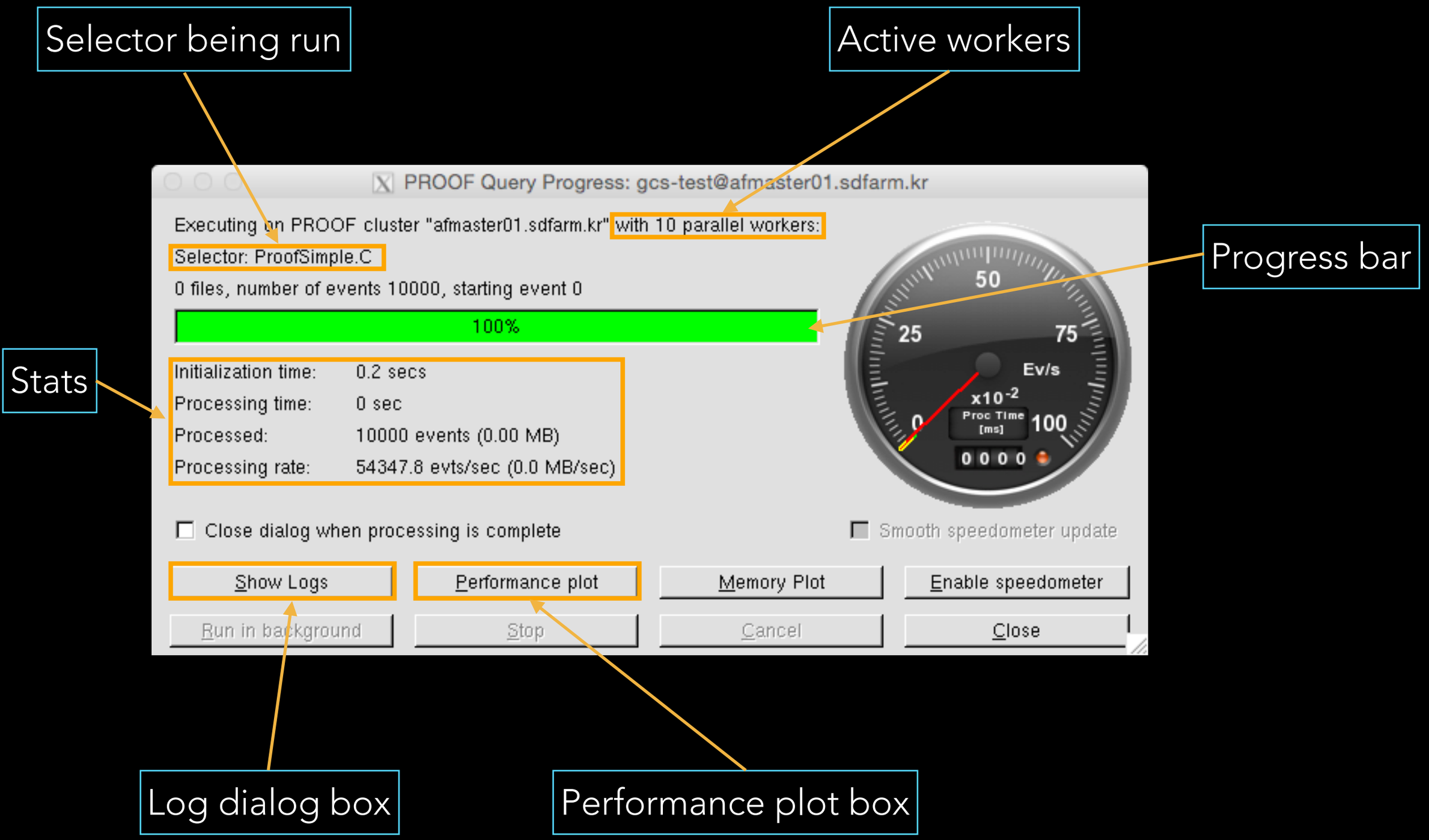
- We are ready to run a first query
- `tutorial/ProofSimple.C`
  - Defines a `TSelector` which fills 100 histograms with gaussian random numbers
- Just do

```
root [1] p->Process("tutorial/ProofSimple.C+",10000)
Mst-0: merging output objects ... done
Mst-0: grand total: sent 107 objects, size: 93962 bytes
(Long64_t)0
root [2]
```



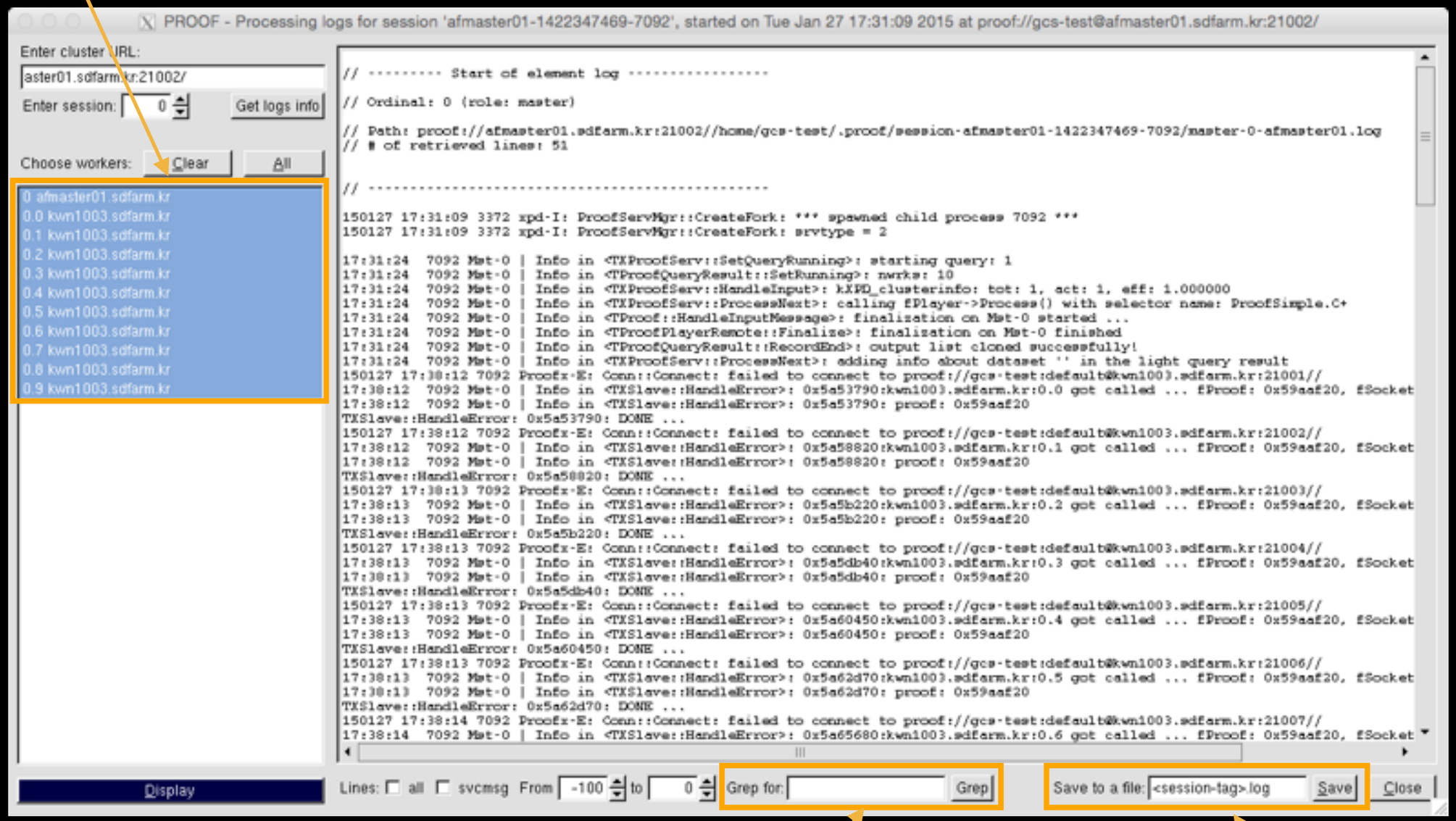


DIALOG BOX


Selector being run
Active workers
PROOF Query Progress: gcs-test@afmaster01.sdfarm.kr
Executing on PROOF cluster "afmaster01.sdfarm.kr" with 10 parallel workers:
Selector: ProofSimple.C
0 files, number of events 10000, starting event 0
100%
Progress bar
Stats
Initialization time: 0.2 secs
Processing time: 0 sec
Processed: 10000 events (0.00 MB)
Processing rate: 54347.8 evts/sec (0.0 MB/sec)
50
25
75
Ev/s
x10-2
0
Proc Time [ms]
100
0 0 0 0
Close dialog when processing is complete
Smooth speedometer update
Show Logs
Performance plot
Memory Plot
Enable speedometer
Run in background
Stop
Cancel
Close
Log dialog box
Performance plot box

LOG DIALOG BOX

Select logs to display

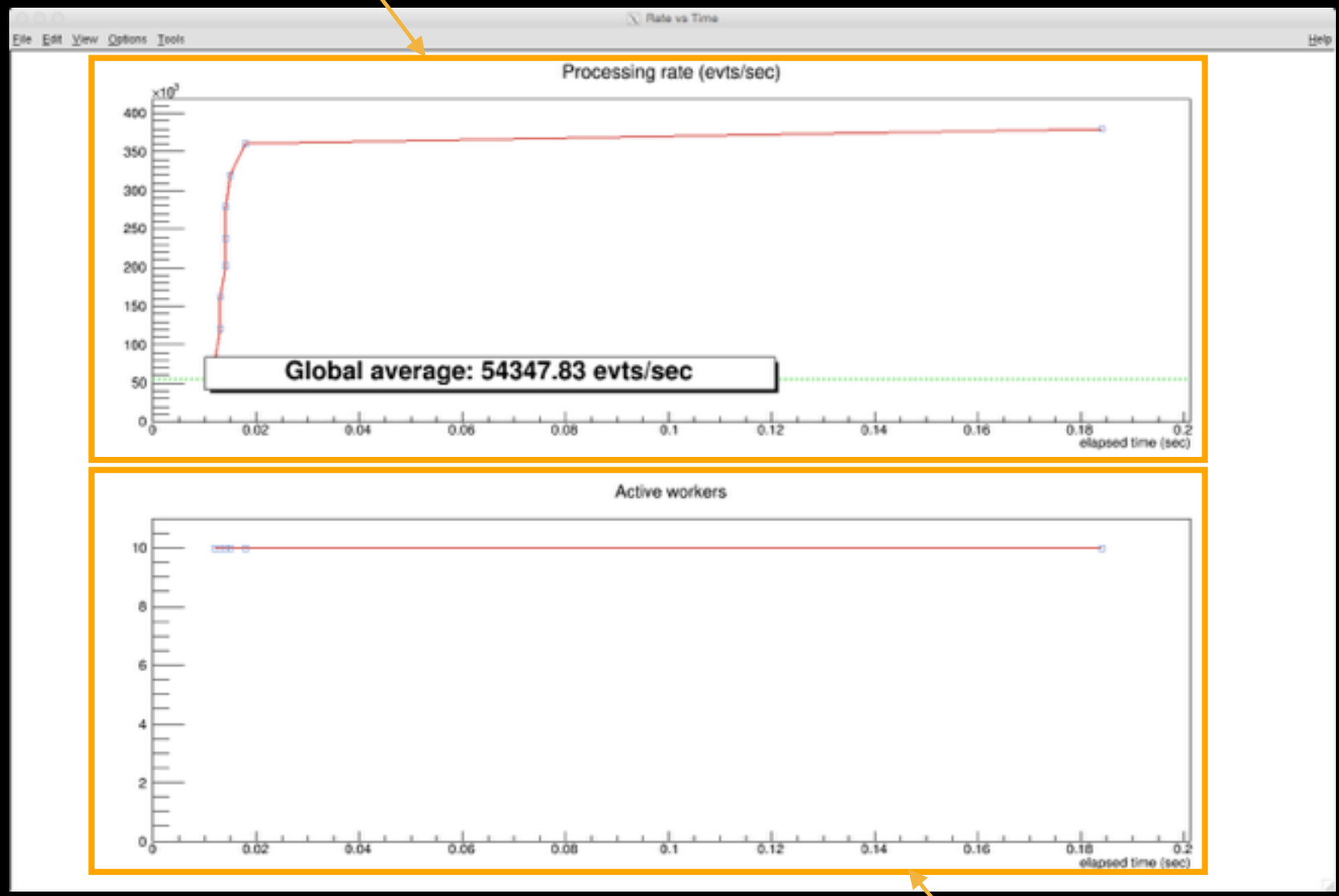


Grep functionality

Save to file

PERFORMANCE PLOT

Processing performance



Number of active workers during the process

OUTPUT LIST

```
root [2] p->GetOutputList()->ls()
OBJ: TList    TList     Doubly linked list : 0
 OBJ: TH1F    h0   h0 : 0 at: 0x2afed00b92a0
 OBJ: TH1F    h1   h1 : 0 at: 0x2afed015b160
 OBJ: TH1F    h2   h2 : 0 at: 0x2afed015b770
 OBJ: TH1F    h3   h3 : 0 at: 0x2afed015bd80
 OBJ: TH1F    h4   h4 : 0 at: 0x2afed015c390
 OBJ: TH1F    h5   h5 : 0 at: 0x2afed015c9a0
 OBJ: TH1F    h6   h6 : 0 at: 0x2afed015cfb0

 OBJ: TH1F    h93  h93 : 0 at: 0x2afed017df20
 OBJ: TH1F    h94  h94 : 0 at: 0x2afed017e530
 OBJ: TH1F    h95  h95 : 0 at: 0x2afed017eb40
 OBJ: TH1F    h96  h96 : 0 at: 0x2afed017f150
 OBJ: TH1F    h97  h97 : 0 at: 0x2afed017f760
 OBJ: TH1F    h98  h98 : 0 at: 0x2afed017fd70
 OBJ: TH1F    h99  h99 : 0 at: 0x2afed0180380
root [3]
```

WHERE OUTPUT IS DEFINED?

tutotial/ProofSimple.C

```
ProofSimple::ProofSimple()
{
   // Constructor
   fNhist = 100;
   …
}
void ProofSimple::SlaveBegin(TTree * /*tree*/)
{
   …
   fHist = new TH1F*[fNhist];
   …
   TString hn;
   // Create the histogram
   for (Int_t i=0; i < fNhist; i++) {
      hn.Form("h%d",i);
      fHist[i] = new TH1F(hn.Data(), hn.Data(), 100, -3., 3.);
      fHist[i]->SetFillColor(kRed);
      fOutput->Add(fHist[i]);
   }
```

CREATE A MACRO TO RUN PROOF

- Avoid repeating all commands each time to start PROOF session
- Here is an example:

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

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

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

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
  - **P**ackage **AR**chives (PAR)
    - Structured archives with build and setup facilities
    - Convenient for more complex and stable things, e.g. the experiment analysis suite

PAR

- 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

PAR EXAMPLE

tutotial/event.par

```
$ tar tzvf tutorial/event.par
drwxr-xr-x ganis/sf          0 2010-05-25 19:06:34 event/
drwxr-xr-x ganis/sf          0 2011-02-18 21:04:44 event/PROOF-INF/
-rw-r--r-- ganis/sf        433 2011-02-18 20:54:10 event/PROOF-INF/SETUP.C
-rwxr-xr-x ganis/sf        414 2011-02-18 21:04:35 event/PROOF-INF/BUILD.sh
-rw-r--r-- ganis/sf      14695 2010-05-25 19:06:34 event/Event.cxx
-rw-r--r-- ganis/sf       2345 2010-05-25 19:06:34 event/Makefile
-rw-r--r-- ganis/sf       7901 2010-05-25 19:06:34 event/Event.h
-rw-r--r-- ganis/sf      13220 2010-05-25 19:06:34 event/Makefile.arch
-rw-r--r-- ganis/sf        259 2010-05-25 19:06:34 event/EventLinkDef.h
```

HANDLING PAR IN PROOF

- TProof provides the following methods to work with PAR

```
UploadPackage(const char *name)
EnalbePackage(const char *name)
ClearPackage(const char *name)
ClearPackages()
ShowPackages()
ShowEnabledPackages()
```

- Try to upload and enable tutorial/event.par

```
root [0] TProof *p = TProof::Open("pod://")
root [1] p->ClearPackages()
root [2] p->UploadPackage(“tutorial/event.par")
root [3] p->EnablePackage("event")
root [4] p->Exec("Event *ev=0")
```

TRY IT OUT

- tutorial/runProof.C
  - Provides useful examples to try out
  - How to run:

```
$ root -l
root [0] .L tutorial/runProof.C+
root [1] runProof("<action>","<url>")
```

- Simple example (the same as we already did)

```
root [2] runProof("simple","pod://")
```

- h1 analysis example (featuring "feedback")

```
root [3] runProof("h1","pod://")
```

- Event(PAR) example

```
root [4] runProof("event","pod://")
root [5] runProof("eventproc","pod://")
```

- Other actions described in runProof.C

TRY IT OUT

- tutorial/runProof.C
  - Provides useful examples to try out
  - How to run:

```
$ root -l
root [0] .L tutorial/runProof.C+
root [1] runProof("<action>","<url>")
```

What could be possible?

  - Simple example (the same as we already did)

```
root [2] runProof("simple","pod://")
```

  - h1 analysis example (featuring "feedback")

```
root [3] runProof("h1","pod://")
```

  - Event(PAR) example

```
root [4] runProof("event","pod://")
root [5] runProof("eventproc","pod://")
```

  - Other actions described in runProof.C

SUMMARY

- In this tutorial, we have learned
  - How to start PoD server and claim workers
  - How to start PROOF on local session
  - How to run simple queries
  - How to manage PAR

REMEMBER

- Documentation about ROOT & PROOF:

http://root.cern.ch

- PoD User's Guide:

http://pod.gsi.de

HOPE YOU GET SOME IDEA WHERE TO BEGIN A LONG JOURNEY TO ANALYSIS WORLD

THANK YOU