Networking, Remote Access, SQL, PROOF and GRID

Fons Rademakers

René Brun

Networking

Networking Classes

- Provide a simple but complete set of networking classes
  - TSocket, TServerSocket, TMonitor, TInetAddress, TMessage
- Optimize for large messages over WAN
  - TPSocket, TPServerSocket
- Operating system independent
  - Via abstract OS interface TSystem

Setting Up a Connection

Server side

```
// Open server socket waiting for connections on specified port
TServerSocket *ss = new TServerSocket(9090, kTRUE);

// Accept a connection and return a full-duplex communication socket
TSocket *sock = ss->Accept();

// Close the server socket (unless we will use it later to wait for
// another connection).
ss->Close();
```

Client side

```
// Open connection to server
TSocket *sock = new TSocket("pcrdm.cern.ch", 9090);
```

Sending Objects

Client Side

```
TH1 *hpx = new TH1F("hpx","This is the px distribution",100,-4,4);
TMessage mess(kMESS_OBJECT);
mess.WriteObject(hpx);
sock->Send(mess);
```

Server Side

```
TMessage *mess;
while (sock->Recv(mess)) {
   if (mess->What() == kMESS_OBJECT) {
      if (mess->GetClass()->InheritsFrom("TH1")) {
         TH1 *h = (TH1 *)mess->ReadObject();
         . . .
      }
   } else if (mess->What() == kMESS_STRING)
      . . .
   delete mess;
}
```

Long Fat Pipes

- Long fat pipes are WAN links with a large bandwidth*delay product
- For optimal performance keep pipe full
- By default this is not the case
  - maximum TCP buffer size is 64KB
  - for a pipe with a 192KB bandwidth*delay product the pipe is empty 60% of the time

TCP Window Scaling (RFC 1323)

- A solution is to use a TCP buffer size equal to the bandwidth*delay product
- This support for large TCP buffers (window scaling) is described in RFC 1323

- Problem: system administrators are needed to change maximum TCP buffer sizes on source and destination machines, e.g. for Linux:
  - echo 200000 > /proc/sys/net/core/rmem_max

Parallel Sockets

- Buffer is striped over multiple sockets in equal parts
- Ideal number of parallel sockets depends on bandwidth*delay product (assuming default 64KB TCP buffer size). No system manager needed to tune network

- Same performance as with large buffers

Setting Up a Parallel Socket Connection

Server side

```
// Open server socket waiting for connections on specified port
TServerSocket *ss = new TPServerSocket(9090, kTRUE);

// Accept a connection and return a full-duplex communication socket
TSocket *sock = ss->Accept();

// Close the server socket (unless we will use it later to wait for
// another connection).
ss->Close();
```

Client side

```
// Open connection to server
TSocket *sock = new TPSocket("pcrdm.cern.ch", 9090);
```

Main Networking Features

- Objects can be passed by value over a network connection
- Easy multiplexing on multiple sockets via TMonitor or via the main event loop
- Support for non-blocking sockets
- Support for long fat pipes (Grid environment)

Remote Access

Remote Access

- Several classes deriving from TFile provide remote file access:
  - TNetFile performs remote access via the special rootd daemon
  - TWebFile performs remote read-only access via an Apache httpd daemon
  - TRFIOFile performs remote access via the rfiod daemon (RFIO is part of the CERN SHIFT software). RFIO is the interface to several mass storage systems: Castor, HPSS

Access Transparency

```
// Local file, uses TFile
TFile *f1 = TFile::Open("local.root")

// Remote file, uses TNetFile and rootd
TFile *f2 = TFile::Open("root://cdfsga.fnal.gov/bigfile.root")

// Remote file, uses TRFIOFile and rfiod
TFile *f3 = TFile::Open("rfio:/alice/run678.root")

// Remote file, uses TWebFile and httpd
TFile *f4 = TFile::Open("http://root.cern.ch/geom/atlas.root")
```

The rootd Daemon

- The rootd is a “simple” byte server
- Listens on port 1094 (allocated by IANA)
- Each connection has its own instance
- Link stays open for duration of connection
- Secure authentication (via SRP protocol)
- Supports parallel sockets
- Also supports basic FTP command set
- Performance better than NFS or RFIO

Parallel FTP

- Parallel FTP implemented via the TFTP class and the rootd daemon
- Uses the TPSocket class
- Supports all standard ftp commands
- Anonymous ftp
- Performance, CERN - GSI:
  - wu-ftp: 1.4 MB/s
  - TFTP: 2.8 MB/s

SQL Interface

SQL Interface

- RDBMS access via a set of abstract base classes
  - TSQLServer, TSQLResult and TSQLRow
- Concrete implementations exist for:
  - MySQL, Oracle, PostgreSQL, SapDB and ODBC
- DB specific modules (plug-ins) are dynamically loaded only when needed

SQL Interface Usage

```
{
   TSQLServer *db = TSQLServer::Connect("mysql://localhost/test",
                                        "nobody", "");
   TSQLResult *res = db->Query("select count(*) from runcatalog "
                               "where tag&(1<<2)");

   int nrows   = res->GetRowCount();
   int nfields = res->GetFieldCount();
   for (int i = 0; i < nrows; i++) {
      TSQLRow *row = res->Next();
      for (int j = 0; j < nfields; j++) {
         printf("%s\n", row->GetField(j));
      }
      delete row;
   }

   delete res;
   delete db;
}
```

Performance Comparison

SQL

```
CREATE TABLE runcatalog (
   dataset       VARCHAR(32) NOT NULL,
   run           INT NOT NULL,
   firstevent    INT,
   events        INT,
   tag           INT,
   energy        FLOAT,
   runtype       ENUM('physics'
                 'cosmics','test'),
   target        VARCHAR(10),
   timef         TIMESTAMP NOT NULL,
   timel         TIMESTAMP NOT NULL,
   rawfilepath   VARCHAR(128),
   comments      VARCHAR(80)
)
```

C++/ROOT

```
class RunCatalog : public TObject {
public:
   enum ERunType { kPhysics, kCosmics,
                   kTest };

   char        fDataSet[32];
   Int_t       fRun;
   Int_t       fFirstEvent;
   Int_t       fEvents;
   Int_t       fTag;
   Float_t     fEnergy;
   ERunType    fRunType;
   char        fTarget[10];
   UInt_t      fTimeFirst;
   UInt_t      fTimeLast;
   char        fRawFilePath[128];
   char        fComments[80];
};
```

Performance Comparison Filling 500000 Entries

MySQL

- 177 s real-time
- 0.3 MB/s
- 54.6 MB DB file

TTree

- 42 s real-time (43 via rootd)
- 3.4 MB/s
- 11.5 MB DB file (compression level 1)

All results on PII 366, 256 MB RAM, RH 6.1

Performance Comparison Select of 2 Columns

MySQL

- 4.5 s real-time
- 12.1 MB/s

TTree

- 2.8 s real-time (2.9 via rootd)
- 4.1 MB/s (19.5 MB/s)

```
SELECT dataset,rawfilepath FROM runcatalog
WHERE tag&7 AND (run=490001 OR run=300122)
```

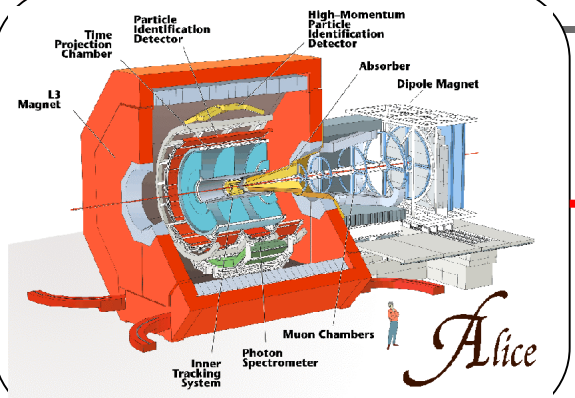
Performance Comparison

- ROOT TTree's are in this case a factor 4 smaller
- Filling time of TTree's is 4.2 times faster
- Query time of TTree's is 2 times faster
- However, MySQL and especially Oracle have the typical advantages of: locking, transactions, roll-back, SQL, etc.

Very Large Databases

- We propose to use a combination of the ROOT object store and an RDBMS to create VLDBs
- The RDBMS is typically used as catalog to keep track of the many ROOT object stores
- Used in the ALICE Data Challenges and by all experiments using ROOT I/O

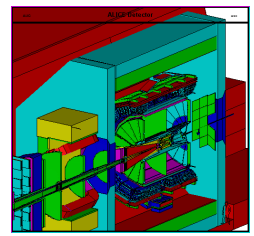
ALICE Data challenges



Raw Data

Simulated Data

AliRoot



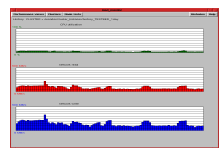
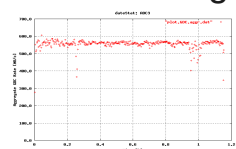
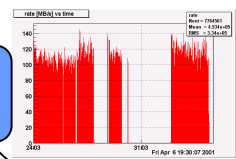
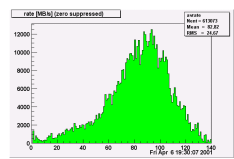
GEANT3
GEANT4
FLUKA

DAQ

File Catalogue

ROOT I/O

Performance Monitoring

CERN TIER 0 TIER 1

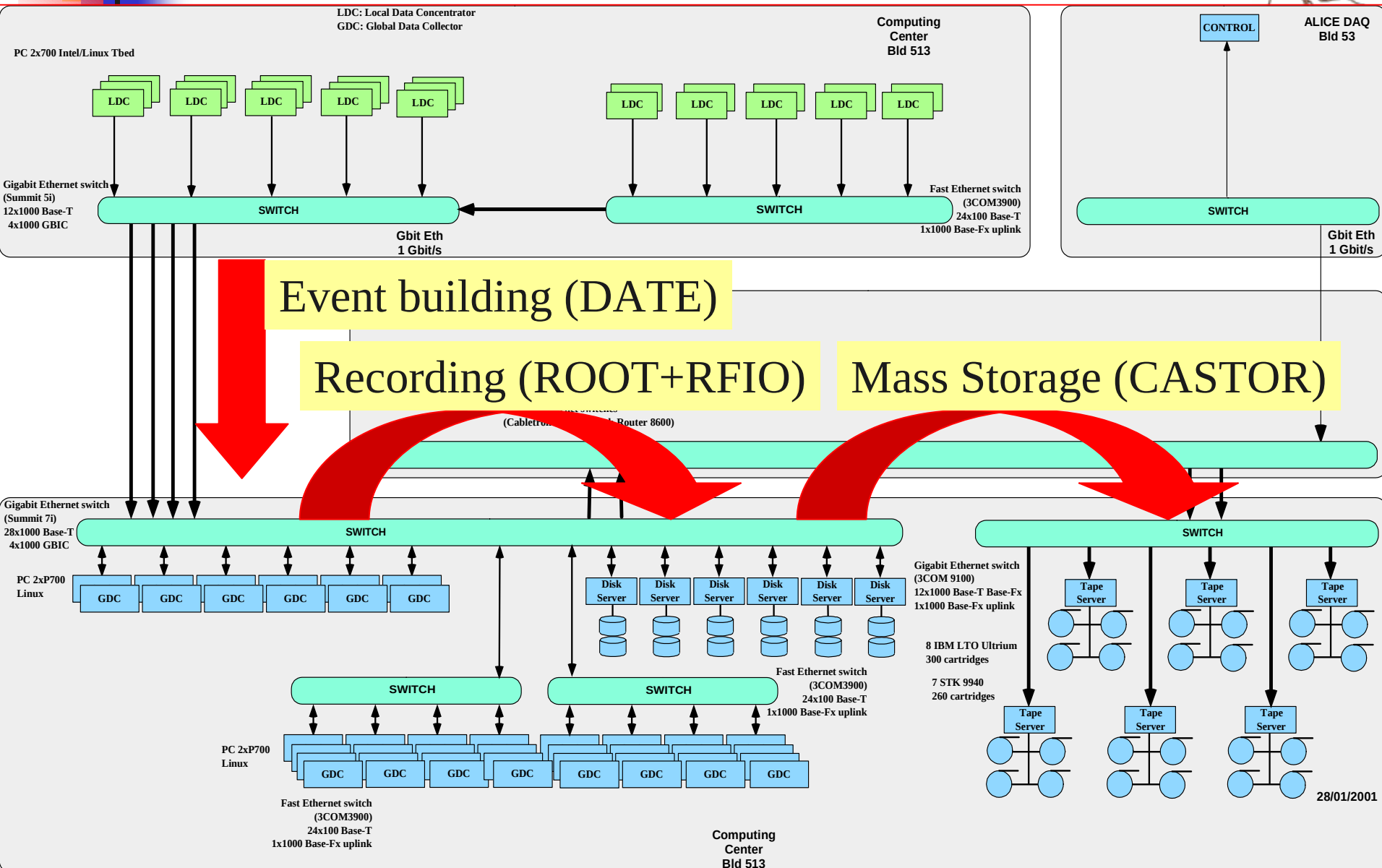
ROOT

CASTOR

GRID

Regional
TIER 1 TIER 2

ALICE Data Challenge III



ADC III Highlights

- Stable and high performances:
  - Max throughput in DATE+ROOT: 240 MB/s
  - Max throughput in DATE+ROOT+CASTOR: 120 MB/s
  - Average during several days: 85 MB/s (>50 TB/week)
- Total amount of data over the complete chain (up to tape):
  - CASTOR file system: >100.000 files of 1 GB (110 TB)
  - File catalogue with 10^5 entries

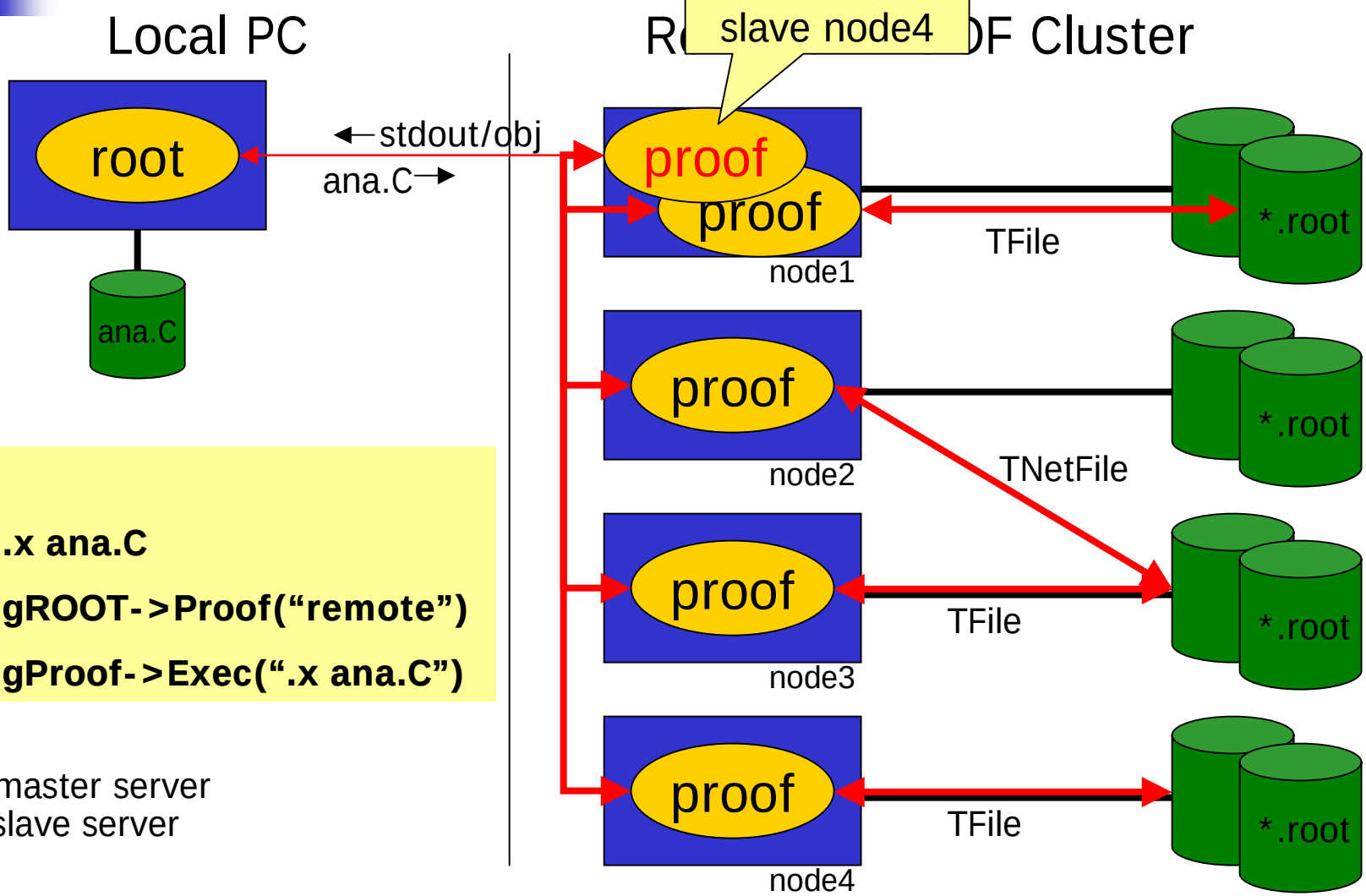
PROOF and GRID

Parallel ROOT Facility

- The PROOF system allows:
  - parallel execution of scripts
  - parallel analysis of chains of trees

  on clusters of heterogeneous machines
- Its design goals are:
  - transparency, scalability, adaptability
- Prototype developed in 1997 as proof of concept (only for simple queries resulting in 1D histograms)

Parallel Script Ex n



$ root

root [0] .x ana.C

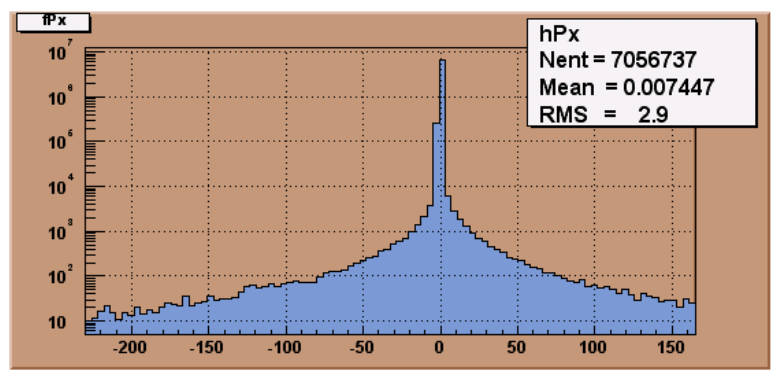
root [1] gROOT->Proof("remote")

root [2] gProof->Exec(".x ana.C")

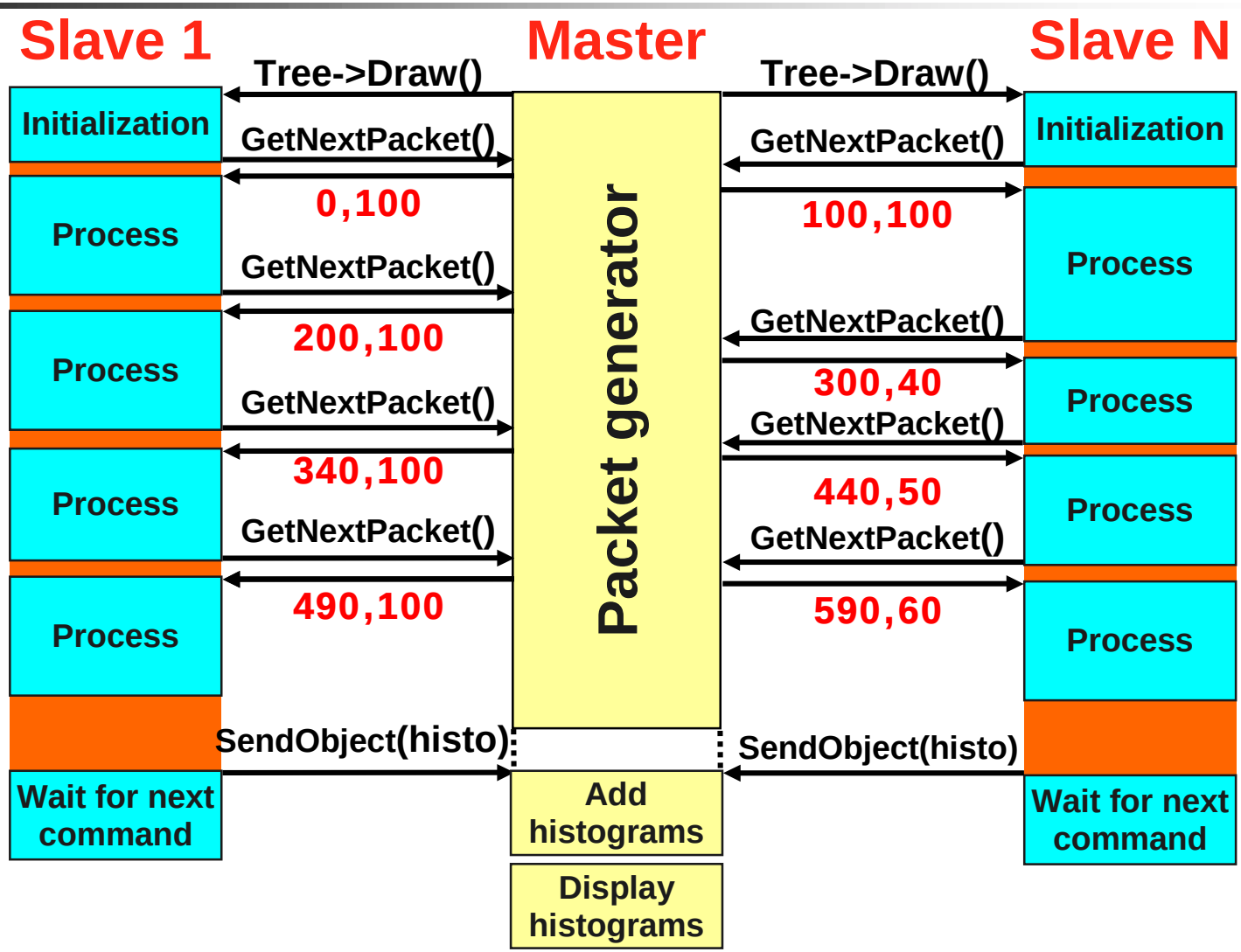
proof = master server
proof = slave server

Parallel Tree Analysis

```
root [0] .! ls -l run846_tree.root
-rw-r-r--  1  rdm    cr   598223259  Feb 1  16:20  run846_tree.root

root [1] TFile f("run846_tree.root")

root [2] gROOT->Time()

root [3] T49->Draw("fPx")
Real time 0:0:11, CP time 10.860

root [4] gROOT->Proof()
*** Proof slave server : pcna49a.cern.ch started ***
*** Proof slave server : pcna49b.cern.c
*** Proof slave server : pcna49c.cern.c
*** Proof slave server : pcna49d.cern.c
*** Proof slave server : pcna49e.cern.c
Real time 0:0:4, CP time 0.140

root [5] T49->Draw("fPx")
Real time 0:0:3, CP time 0.240
```



Workflow For Tree Analysis



PROOF Session Statistics

```
root [6] T49->Print("p")
Total events processed:                      10585
Total number of packets:                     147
Default packet size:                         100
Smallest packet size:                        20
Average packet size:                         72.01
Total time (s):                              2.78
Average time between packets (ms):           10.93
Shortest time for packet (ms):               99
Number of active slaves:                     5
   Number of events processed by slave 0: 1890
   Number of events processed by slave 1: 2168
   Number of events processed by slave 2: 2184
   Number of events processed by slave 3: 2667
   Number of events processed by slave 4: 1676
```

PROOF Transparency

- On demand, make available to the PROOF servers any objects created in the client
- Return to the client all objects created on the PROOF slaves
  - the master server will try to add “partial” objects coming from the different slaves before sending them to the client

PROOF Scalability

- Scalability in parallel systems is determined by the amount of communication overhead (Amdahl's law)
- Varying the packet size allows one to tune the system. The larger the packets the less communications is needed, the better the scalability
  - Disadvantage: less adaptive to varying conditions on slaves

PROOF Adaptability

- Adaptability means to be able to adapt to varying conditions (load, disk activity) on slaves
- By using a “pull” architecture the slaves determine their own processing rate and allows the master to control the amount of work to hand out
  - disadvantage: too fine grain packet size tuning hurts scalability

PROOF Error Handling

- Handling death of PROOF servers
  - death of master
    - fatal, need to reconnect
  - death of slave
    - master will resubmit packets of death slave to other slaves
- Handling of ctrl-c
  - OOB message is send to master, and forwarded to slaves, causing soft/hard interrupt

PROOF Authentication

- PROOF supports secure and un-secure authentication mechanisms
  - Un-secure
    - mangled password send over network
  - Secure
    - SRP, Secure Remote Password protocol (Stanford Univ.), public key technology
    - Soon: Globus authentication

PROOF Grid Interface

- PROOF can use Grid Resource Broker to detect which nodes in a cluster can be used in the parallel session
- PROOF can use Grid File Catalogue and Replication Manager to map LFN's to chain of PFN's
- PROOF can use Grid Monitoring Services
- Access will be via abstract GRID interface