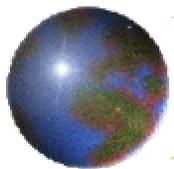


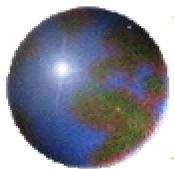
*PROOF*

As of June 2004



## *Parallel ROOT Facility*

- ❖ The PROOF system allows:
  - ❑ Parallel analysis of trees in a set of files
  - ❑ Parallel analysis of objects in a set of files
  - ❑ Parallel execution of scriptson clusters of heterogeneous machines
- ❖ Its design goals are:
  - ❑ Transparency, scalability, adaptability



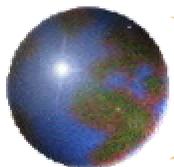
## *File Specification*

### ✦ Via TDataSet

- ✦ Similar to TChain
- ✦ No restricted to TTree (as the main objects)
- ✦ No ordering implied

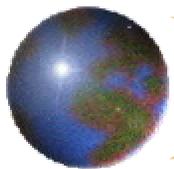
### ✦ Design for

- ✦ (quasi) interactive work
- ✦ Repetitive runs
  - Requires availability of all files at query startup time



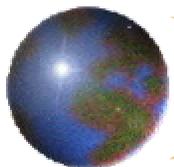
## *File Access*

- ⊕ Direct Access
- ⊕ Rootd
- ⊕ Xrootd
  - ⊞ Highly reliable and configurable file server (peer to peer type, can continue even after some servers become unavailable).
- ⊕ Dcache, Rfio, etc.
  - ⊞ No 'local file optimization' in this case



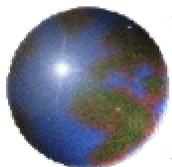
## *Deployment Issues*

- ⊕ Needs ROOT installed
  - ⊠ Version might matter
    - When using pre-compiled user code
- ⊕ Needs prooftd configured/started by
  - ⊠ user
  - ⊠ Xinetd
  - ⊠ Condor
  - ⊠ Grid
  - ⊠ Etc.



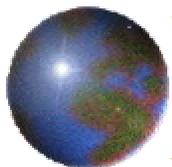
## *What Proof does*

- ⊕ Given a set of 'slaves'
  - ⊞ distribute the load among slaves
  - ⊞ Try to give preference to slave having direct access to file
- ⊕ What Proof does not do
  - ⊞ Select slaves to be used
  - ⊞ This needs to be configured
    - Configuration Files
    - PEAC
    - Grid



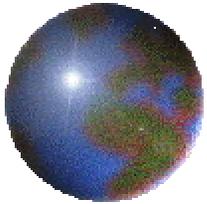
## *PEAC/CDF*

- ✚ Prototype of PEAC at SC2003
  - ☒ Connection between Condor and Proof
  - ☒ Connection with File Storage
    - User select "file set"
    - PEAC copies files out of File Storage
    - PEAC assigns files and slaves 'appropriately'
  - ☒ Working prototype
  - ☒ More work to be done to add the checks and balances needs to run on a "shared" production farm.
  - ☒ New prototype for SC2004
    - Grid Middleware for authentication (instead of kerberos)
- ✚ STNtuple ported to PROOF
  - ☒ Christoph Paus



## *What's in the Pipeline*

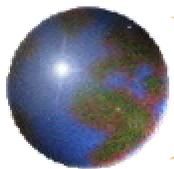
- ✦ Packetizer improvements (found in efficiencies during the test on the large FNAL cluster. Thanks to CDF and Frank Wuerthwein)
- ✦ Full support TDataSet::Draw() command (currently only 1D histo)
- ✦ Monitoring framework (mostly there) allows measuring of latencies, packet sizes, packet processing times, etc, etc.
- ✦ Support for tree friends and eventlists
- ✦ File catalog access via TGrid (currently AliEn/EGEE grid, US grid support requires to create a TGrid class for that grid)
- ✦ Slave startup on grid via interactive grid jobs
- ✦ proofd/rootd proxy servers needed for the workers to talk to the master



# *PROOF with AliEn and GLite*

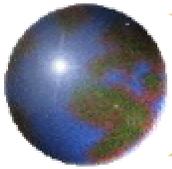
Fons Rademakers

Bring the KB to the PB not the PB to the KB



## *Parallel ROOT Facility*

- ❖ The PROOF system allows:
  - ❑ Parallel analysis of trees in a set of files
  - ❑ Parallel analysis of objects in a set of files
  - ❑ Parallel execution of scriptson clusters of heterogeneous machines
- ❖ Its design goals are:
  - ❑ Transparency, scalability, adaptability



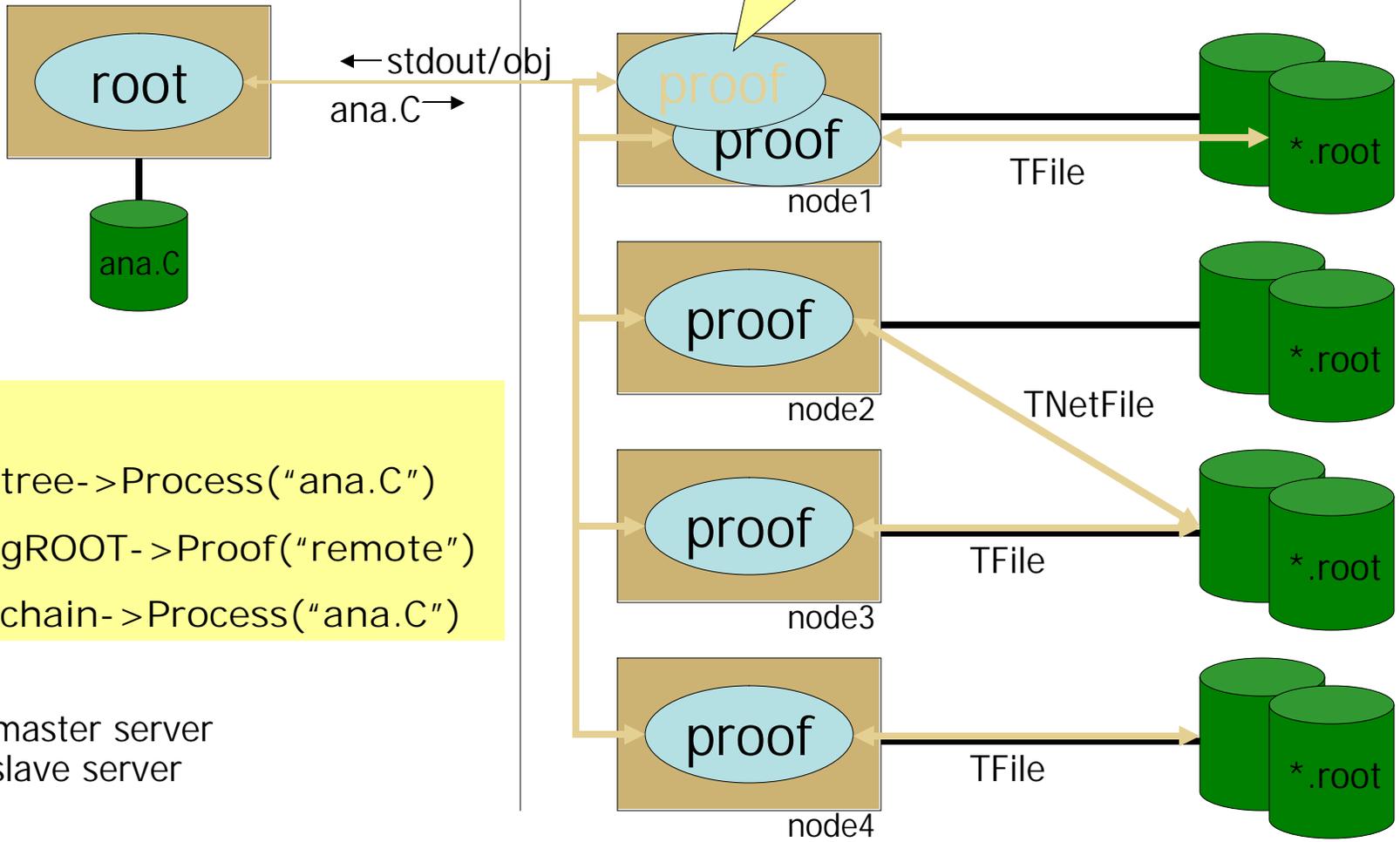
# Parallel Script Execution

```
#proof.conf
slave node1
slave node2
slave node3
slave node4
```

Local PC

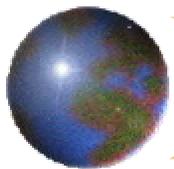
Remote

DF Cluster



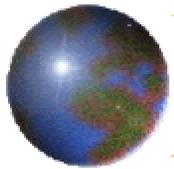
```
$ root
root [0] tree->Process("ana.C")
root [1] gROOT->Proof("remote")
root [2] chain->Process("ana.C")
```

proof = master server  
proof = slave server



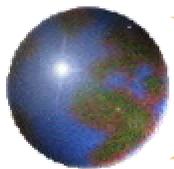
## *Data Access Strategies*

- ✦ Each slave get assigned, as much as possible, packets representing data in local files
- ✦ If no (more) local data, get remote data via rootd and rfio (needs good LAN, like GB eth)
- ✦ In case of SAN/NAS just use round robin strategy



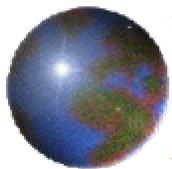
## *PROOF Transparency*

- ✦ Make working on PROOF as similar as working on your local machine
- ✦ 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



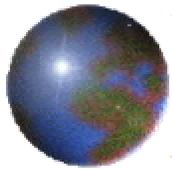
## *PROOF Scalability*

- ✦ 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

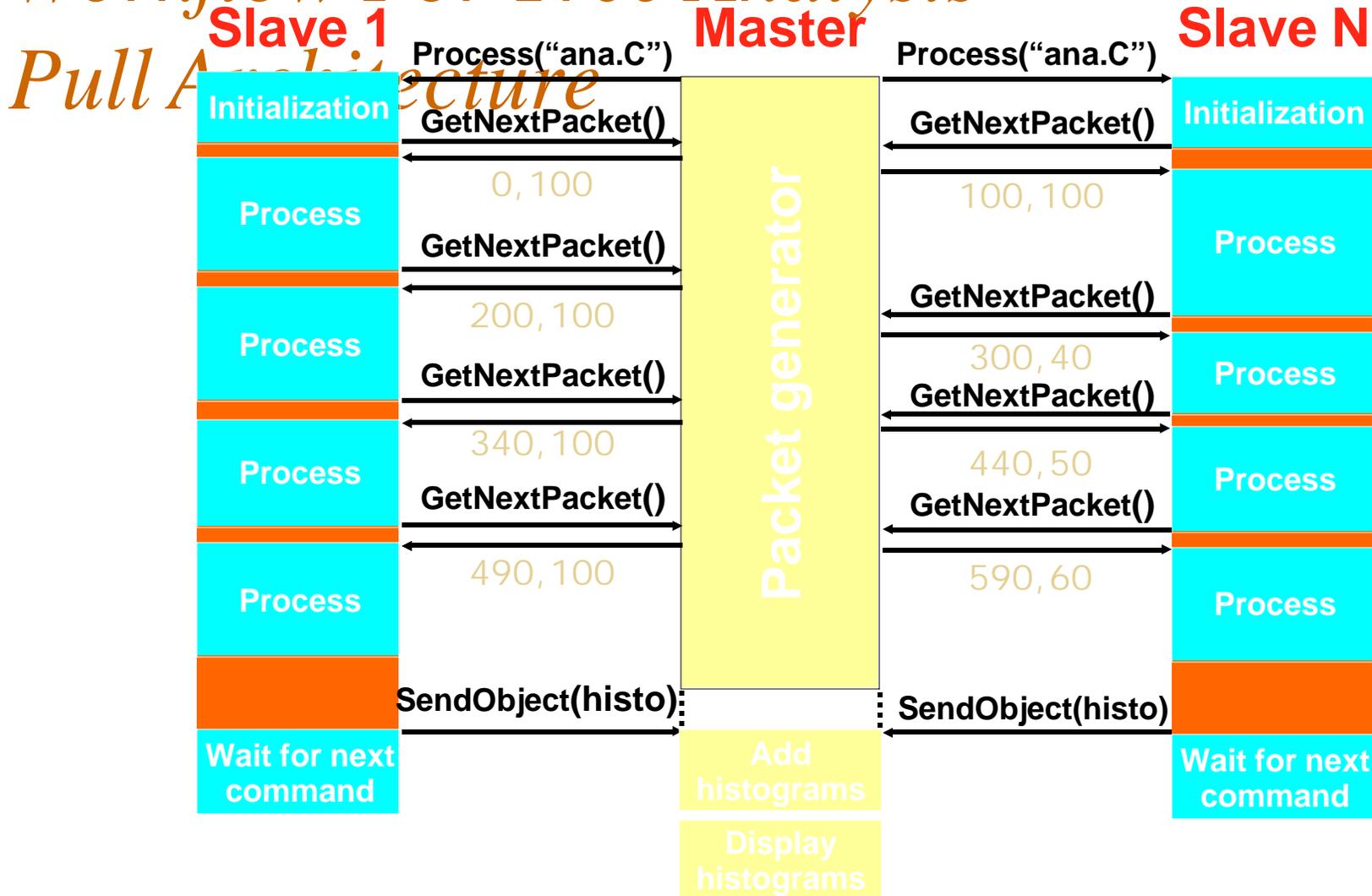


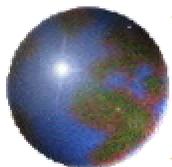
## *PROOF Adaptability*

- ✦ 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



# Workflow For Tree Analysis





# *PROOF Error Handling*

## ⊕ Handling death of PROOF servers

### ⊞ Death of master

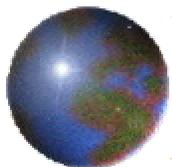
- Fatal, need to reconnect

### ⊞ Death of slave

- Master can resubmit packets of death slave to other slaves

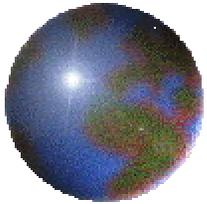
## ⊕ Handling of ctrl-c

- ⊞ OOB message is send to master, and forwarded to slaves, causing soft/hard interrupt

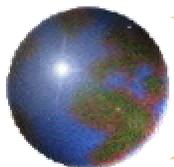


## *PROOF Authentication*

- ✦ PROOF supports secure and un-secure authentication mechanisms
- ✦ Same as for rootd
  - ✦ UsrPwd
  - ✦ SRP
  - ✦ Kerberos
  - ✦ Globus
  - ✦ SSH
  - ✦ UidGid

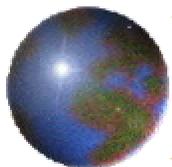


# *Architecture and Implementation*



# *TSelector – The Algorithms*

```
Basic DDD TSelector
// Abbreviated version
class TSelector : public TObject {
Protected:
    TList *fInput;
    TList *fOutput;
public
    void Init(TTree*);
    void Begin(TTree*);
    void SlaveBegin(TTree *);
    Bool_t Process(int entry);
    void SlaveTerminate();
    void Terminate();
};
```



## *TDSet – The Data*

- Specify a collection of TTrees or files with objects

```
root[0] TDSet *d = new TDSet("TTree", "tracks", "/");
```

OR

```
root[0] TDSet *d = new TDSet("TEvent", "", "/objs");
```

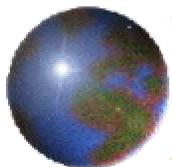
```
root[1] d->Add("root://rcrs4001/a.root");
```

...

```
root[10] d->Print("a");
```

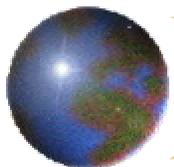
```
root[11] d->Process("mySelector.C", nentries, first);
```

- Returned by DB or File Catalog query etc.
- Use logical filenames ("lfn:...")



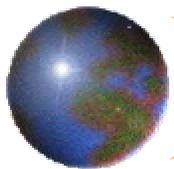
## *Sandbox – The Environment*

- ⊕ Each slave runs in its own sandbox
  - ▣ Identical, but independent
- ⊕ Multiple file spaces in a PROOF setup
  - ▣ Shared via NFS, AFS, shared nothing
- ⊕ File transfers are minimized
  - ▣ Cache
  - ▣ Packages



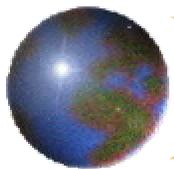
## *Sandbox – The Cache*

- ⊕ Minimize the number of file transfers
  - ▣ One cache per file space
- ⊕ Locking to guarantee consistency
- ⊕ File identity and integrity ensured using
  - ▣ MD5 digest
  - ▣ Time stamps
- ⊕ Transparent via `TProof::Sendfile()`



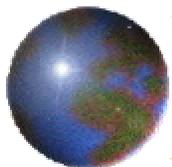
## *Sandbox – Package Manager*

- ✦ Provide a collection of files in the sandbox
- ✦ Binary or source packages
- ✦ PAR files: **P**ROOF **A**Rchive. Like Java jar
  - ▣ Tar file, ROOT-INF directory
  - ▣ BUILD.sh
  - ▣ SETUP.C, per slave setting
- ✦ API to manage and activate packages

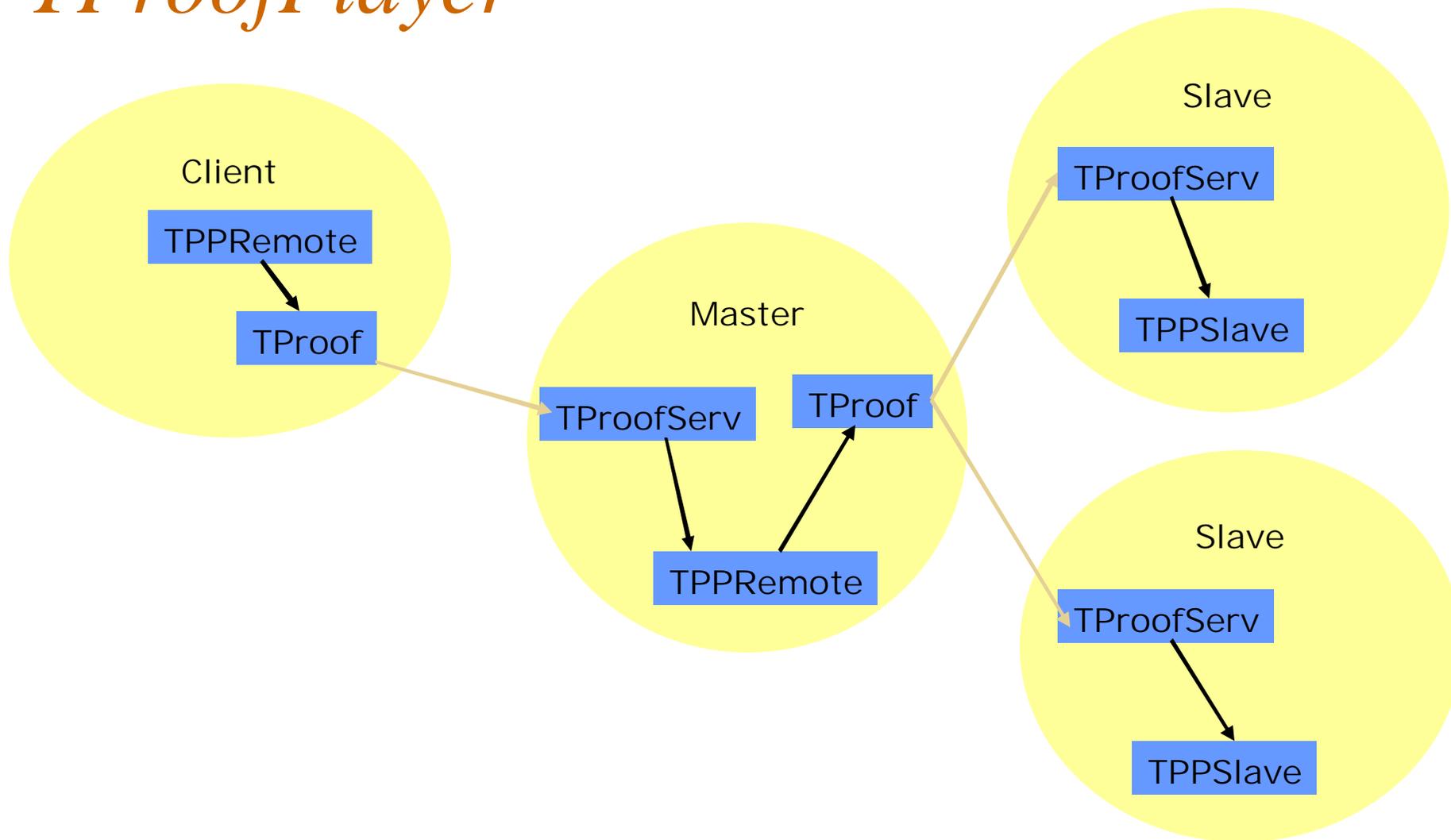


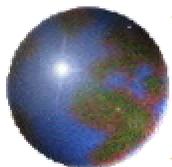
## *Implementation Highlights*

- ✦ TProofPlayer class hierarchy
  - ▣ Basic API to process events in PROOF
  - ▣ Implement event loop
  - ▣ Implement proxy for remote execution
- ✦ TEventIter
  - ▣ Access to TTree or TObject derived collection
  - ▣ Cache file, directory, tree

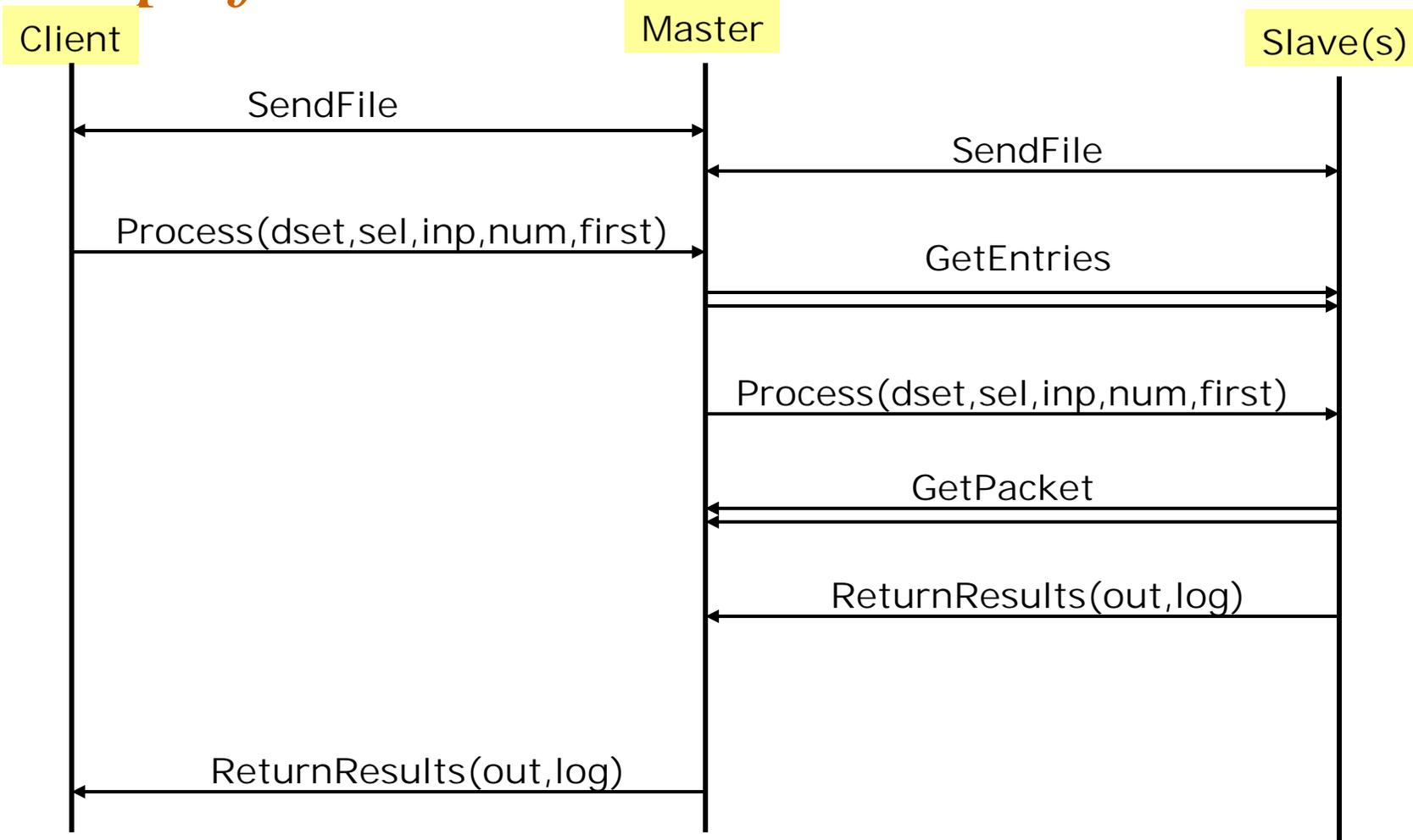


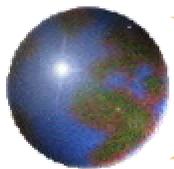
# *TProofPlayer*





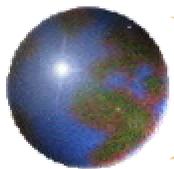
# *Simplified Message Flow*





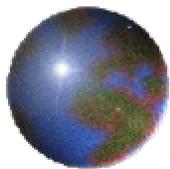
## *Dynamic Histogram Binning*

- ⊕ Implemented using THLimitsFinder class
- ⊕ Avoid synchronization between slaves
- ⊕ Keep score-board in master
  - ⊠ Use histogram name as key
  - ⊠ First slave posts limits
  - ⊠ Master determines best bin size
  - ⊠ Others use these values



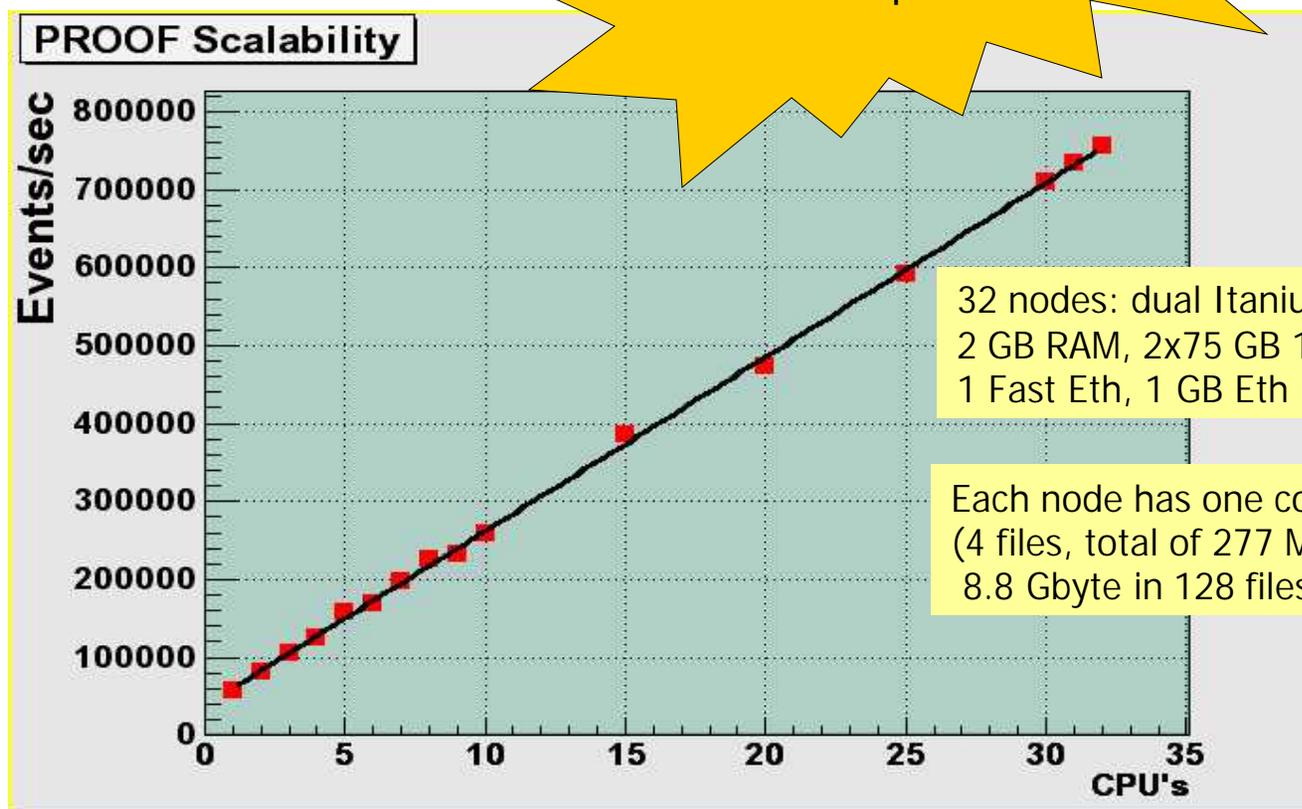
## *Merge API*

- ✦ Collect output lists in master server
- ✦ Objects are identified by name
- ✦ Combine partial results
- ✦ Member function: Merge(TCollection \*)
  - ▣ Executed via CINT, no inheritance required
- ✦ Standard implementation for histograms and (in memory) trees
- ✦ Otherwise return the individual objects



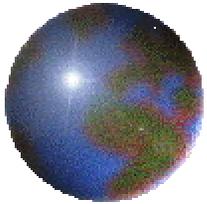
# *PROOF Scalability*

8.8GB, 128 files  
1 node: 325 s  
32 nodes in parallel: 12 s

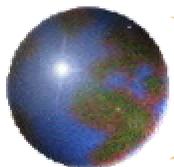


32 nodes: dual Itanium II 1 GHz CPU's,  
2 GB RAM, 2x75 GB 15K SCSI disk,  
1 Fast Eth, 1 GB Eth nic (not used)

Each node has one copy of the data set  
(4 files, total of 277 MB), 32 nodes:  
8.8 Gbyte in 128 files, 9 million events

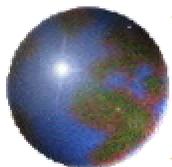


# *Setting Up PROOF*



## *Setting Up PROOF*

- ✦ Install ROOT system
- ✦ For automatic execution of daemons add proofd and rootd to /etc/inetd.conf (or in /etc/xinetd.d) and /etc/services (not mandatory, servers can be started by users)
  - ✦ The rootd (1094) and proofd (1093) port numbers have been officially assigned by IANA
- ✦ Setup proof.conf file describing cluster
- ✦ Setup authentication files (globally, users can override)

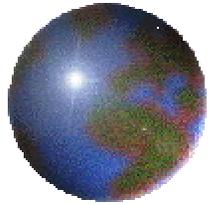


# *PROOF Configuration File*

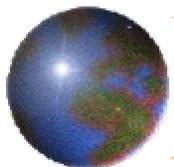
```
# PROOF config file. It has a very simple format:
#
# node <hostname> [image=<imagename>]
# slave <hostname> [perf=<perfindex>]
#           [image=<imagename>] [port=<portnumber>]
#           [srp | krb5]
# user <username> on <hostname>

node csc02    image=nfs

slave csc03   image=nfs
slave csc04   image=nfs
slave csc05   image=nfs
slave csc06   image=nfs
slave csc07   image=nfs
slave csc08   image=nfs
slave csc09   image=nfs
slave csc10   image=nfs
```



# *The AliEn GRID*

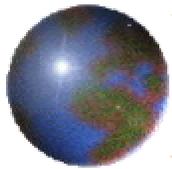


## *AliEn - A Lightweight GRID*

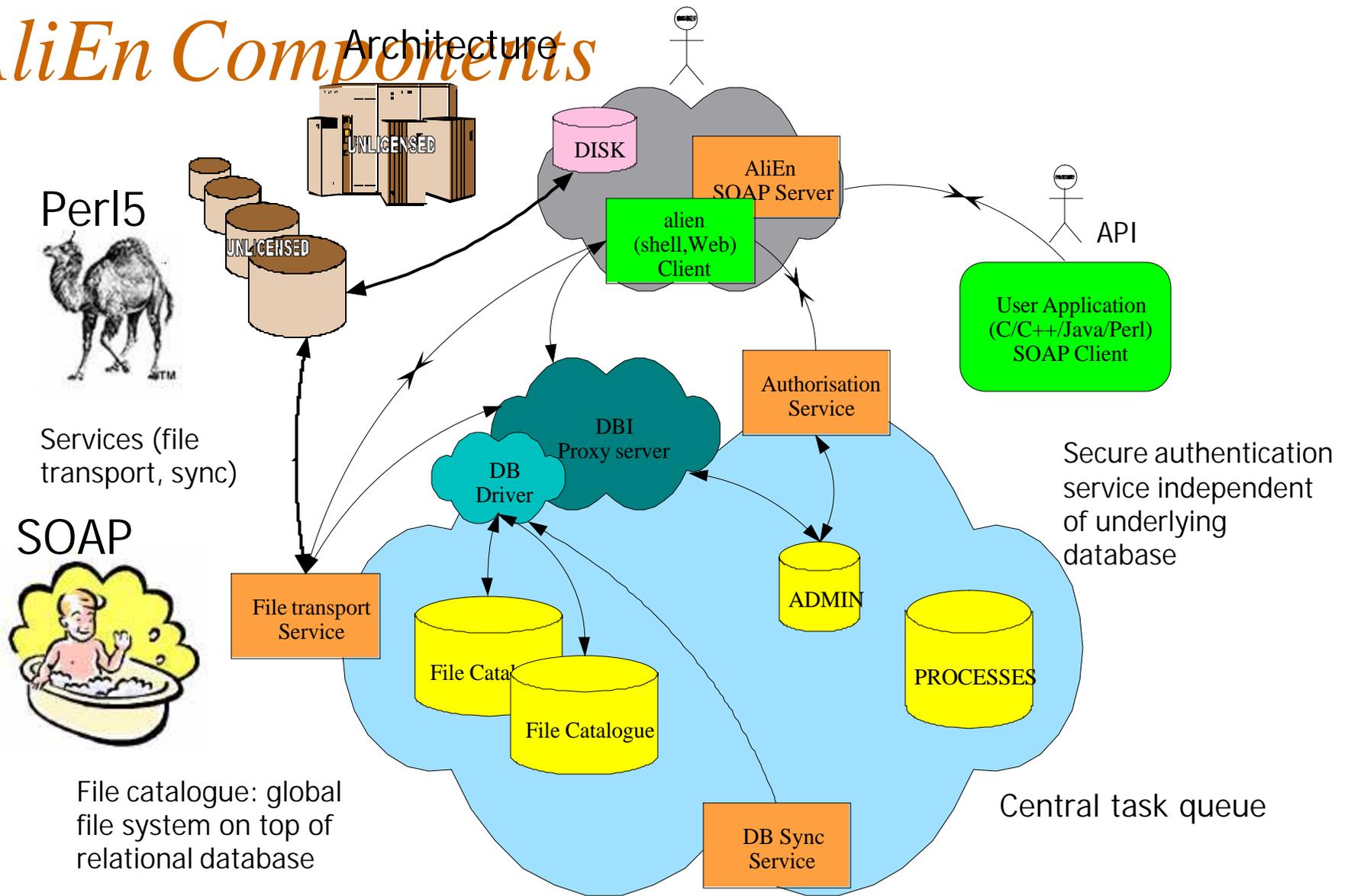
- ✦ AliEn (<http://alien.cern.ch>) is a lightweight alternative to full blown GRID based on standard components (SOAP, Web services)
  - ✦ Distributed file catalogue as a global file system on a RDBMS
  - ✦ TAG catalogue, as extension
  - ✦ Secure authentication
  - ✦ Central queue manager ("pull" vs "push" model)
  - ✦ Monitoring infrastructure
  - ✦ C/C++/perl API
  - ✦ Automatic software installation with AliKit

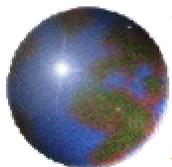
## The Core GRID Functionality !!

- ✦ AliEn is routinely used in the different ALICE data challenges
- ✦ AliEn has been released as the EGEE GLite prototype



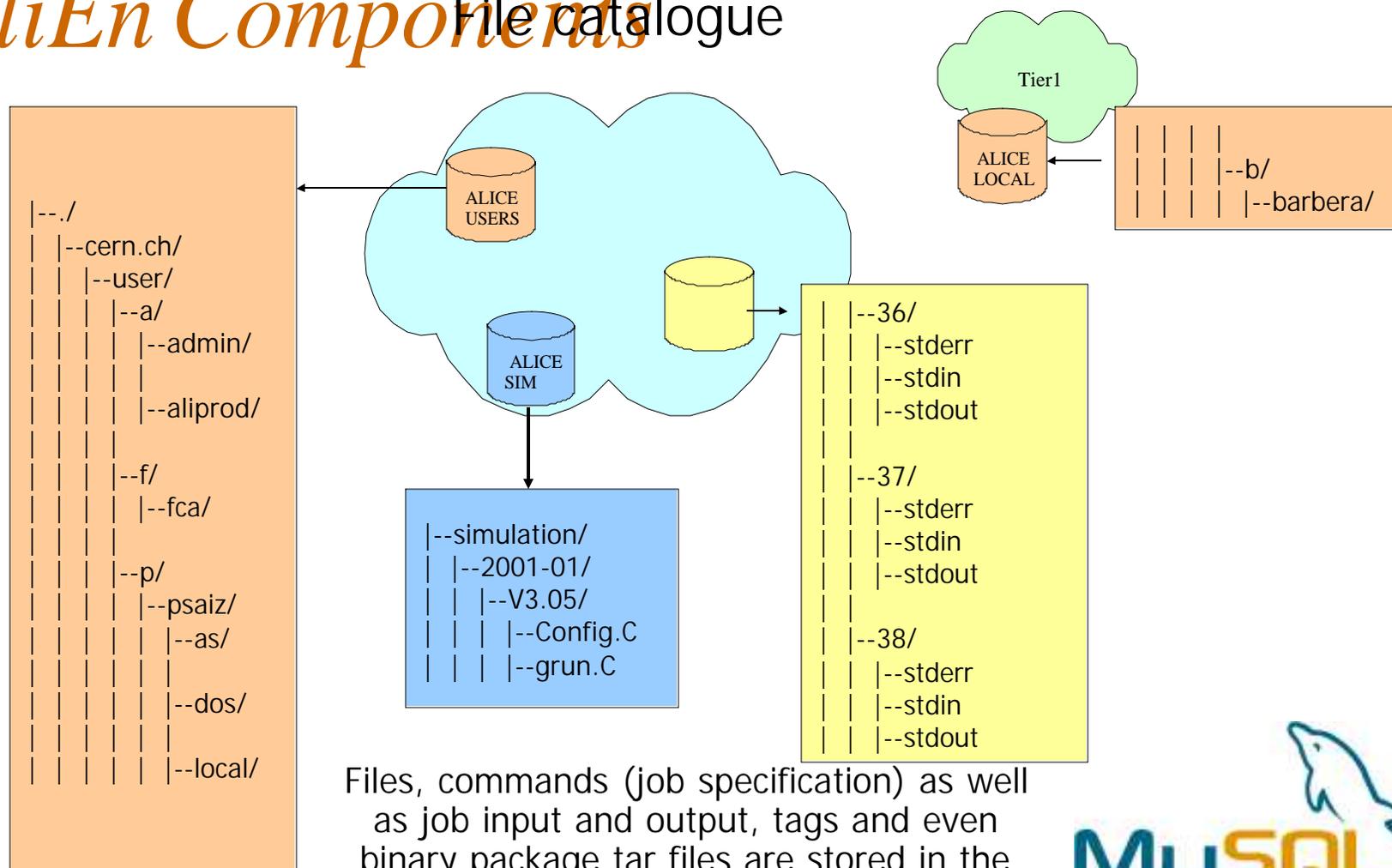
# AliEn Components Architecture





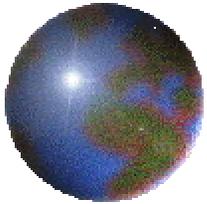
# AliEn Components

## File catalogue

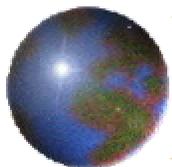


Files, commands (job specification) as well as job input and output, tags and even binary package tar files are stored in the catalogue



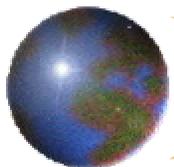


# *PROOF and the GRID*



## *PROOF Grid Interface*

- ✦ PROOF can use a 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 PFN's
- ✦ PROOF daemons can be started by Grid job scheduler
- ✦ PROOF can use Grid Monitoring Services
- ✦ Access via abstract Grid interface



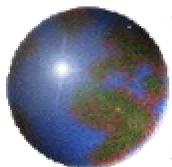
## *TGrid Class* —

```
class TGrid : public TObject {
public:
    virtual Int_t      AddFile(const char *lfn, const char *pfn) = 0;
    virtual Int_t      DeleteFile(const char *lfn) = 0;
    virtual TGridResult *GetPhysicalFileNames(const char *lfn) = 0;
    virtual Int_t      AddAttribute(const char *lfn,
                                   const char *attrname,
                                   const char *attrval) = 0;
    virtual Int_t      DeleteAttribute(const char *lfn,
                                       const char *attrname) = 0;
    virtual TGridResult *GetAttributes(const char *lfn) = 0;
    virtual void        Close(Option_t *option="") = 0;

    virtual TGridResult *Query(const char *query) = 0;

    static TGrid *Connect(const char *grid, const char *uid = 0,
                          const char *pw = 0);

    ClassDef(TGrid,0) // ABC defining interface to GRID services
};
```



## *Running PROOF Using AliEn*

```
TGrid *alien = TGrid::Connect("alien");

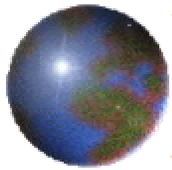
TGridResult *res;
res = alien->Query("lfn:///alice/simulation/2001-04/V0.6*.root");

TDSet *treeset = new TDSet("TTree", "AOD");
treeset->Add(res);

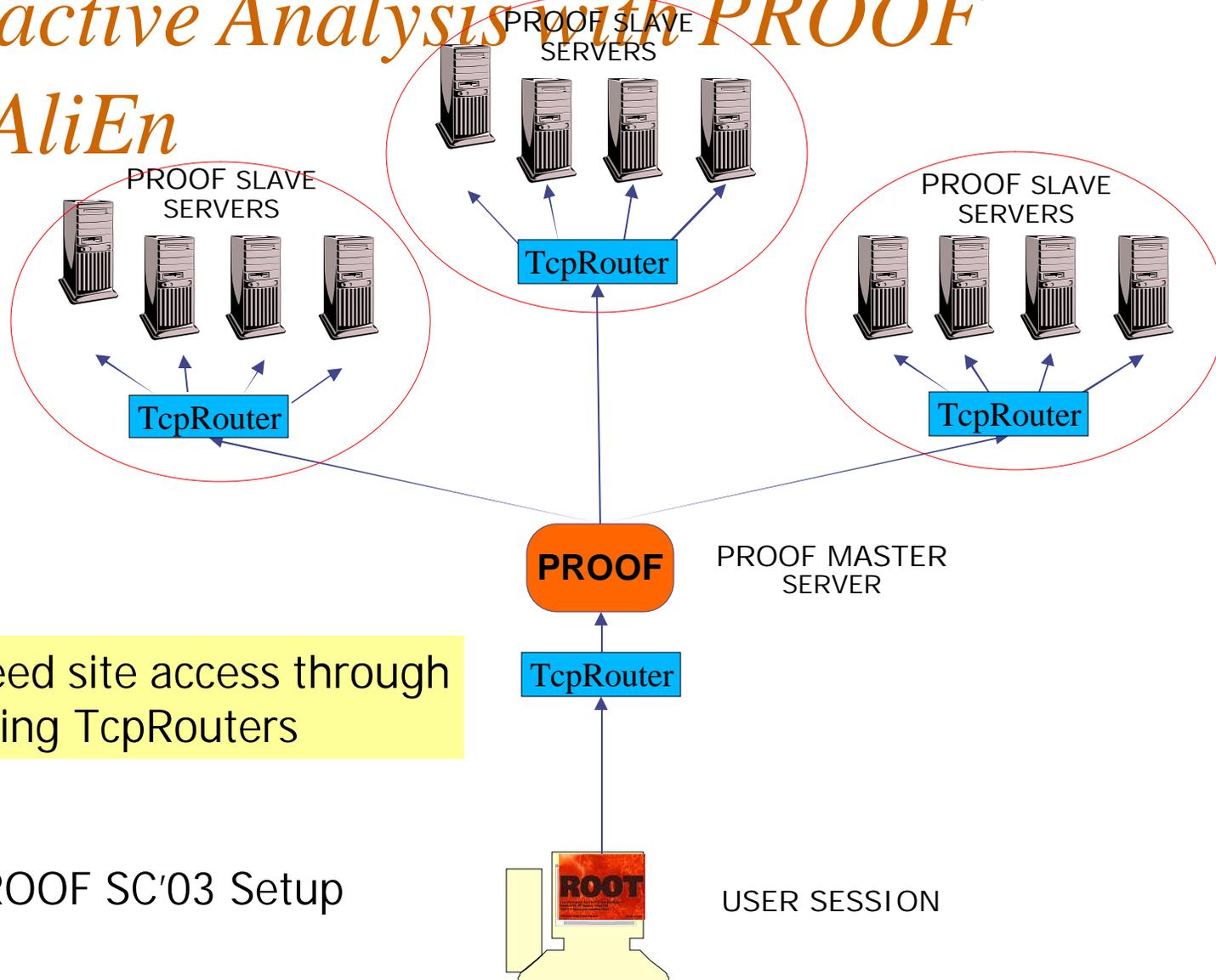
gROOT->Proof(res); // use files in result set to find remote nodes
treeset->Process("myselector.C");

// plot/save objects produced in myselector.C
. . .
```

This scenario was demonstrated by ALICE at SC'03 in Phoenix

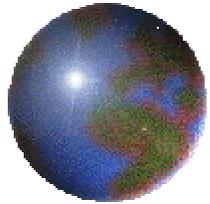


# Interactive Analysis with PROOF and AliEn

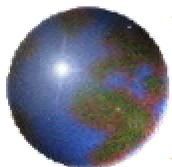


Guaranteed site access through Multiplexing TcpRouters

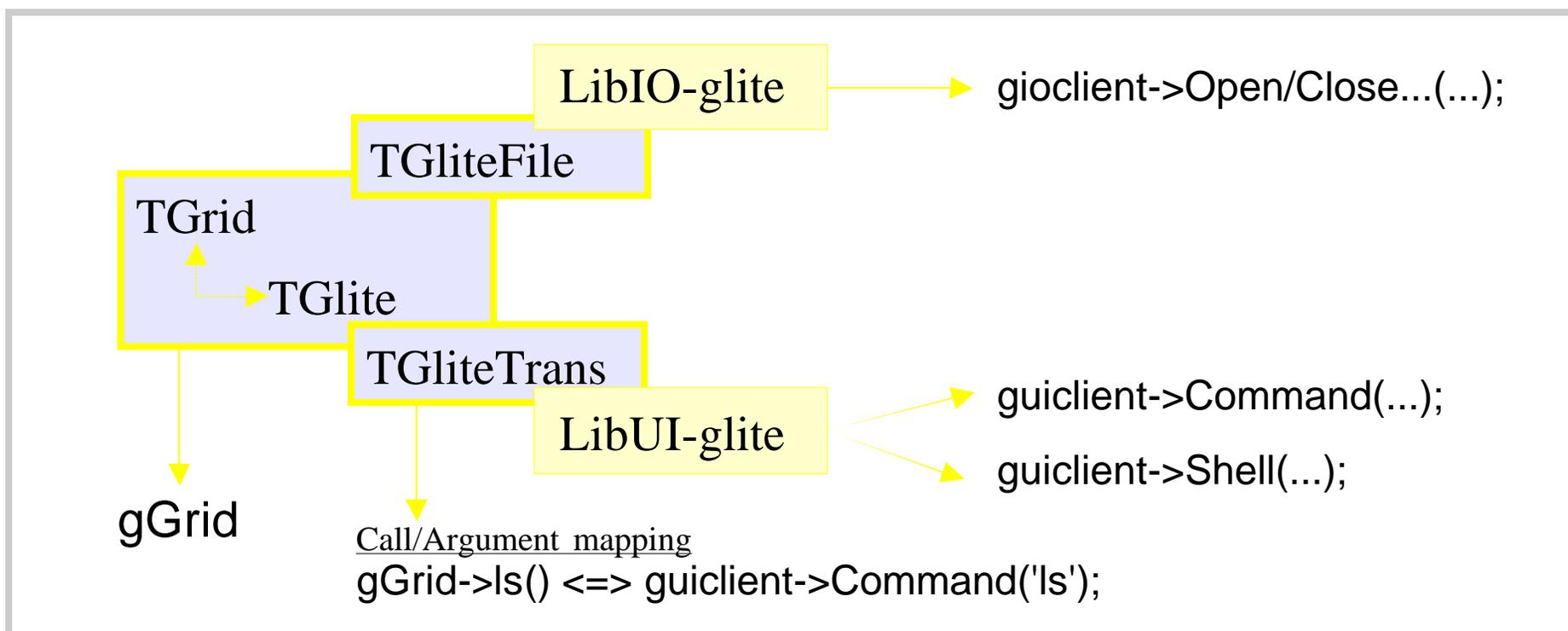
AliEn/PROOF SC'03 Setup

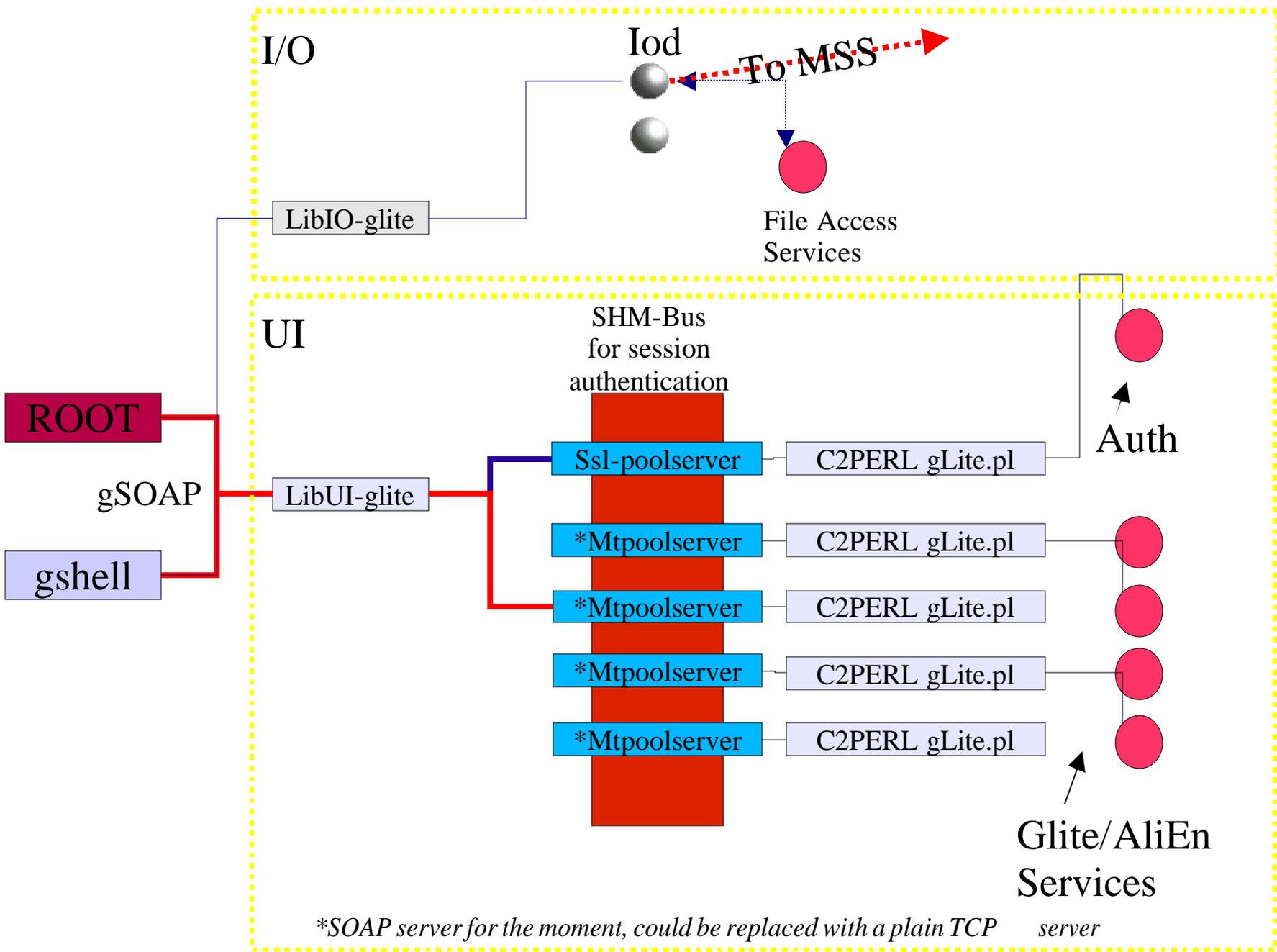


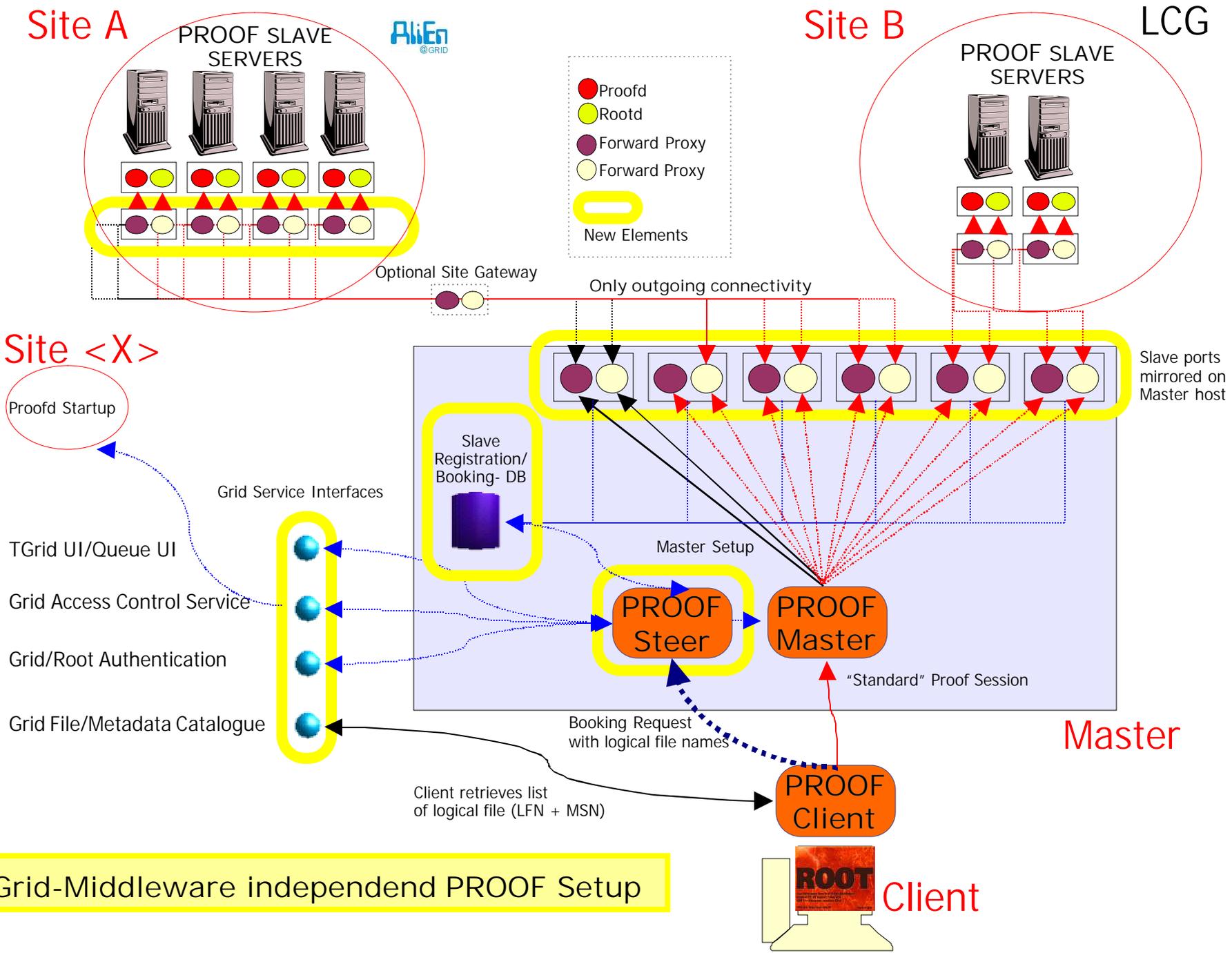
# *PROOF and GLite*

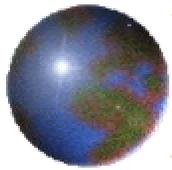


# *TGlite Interface*

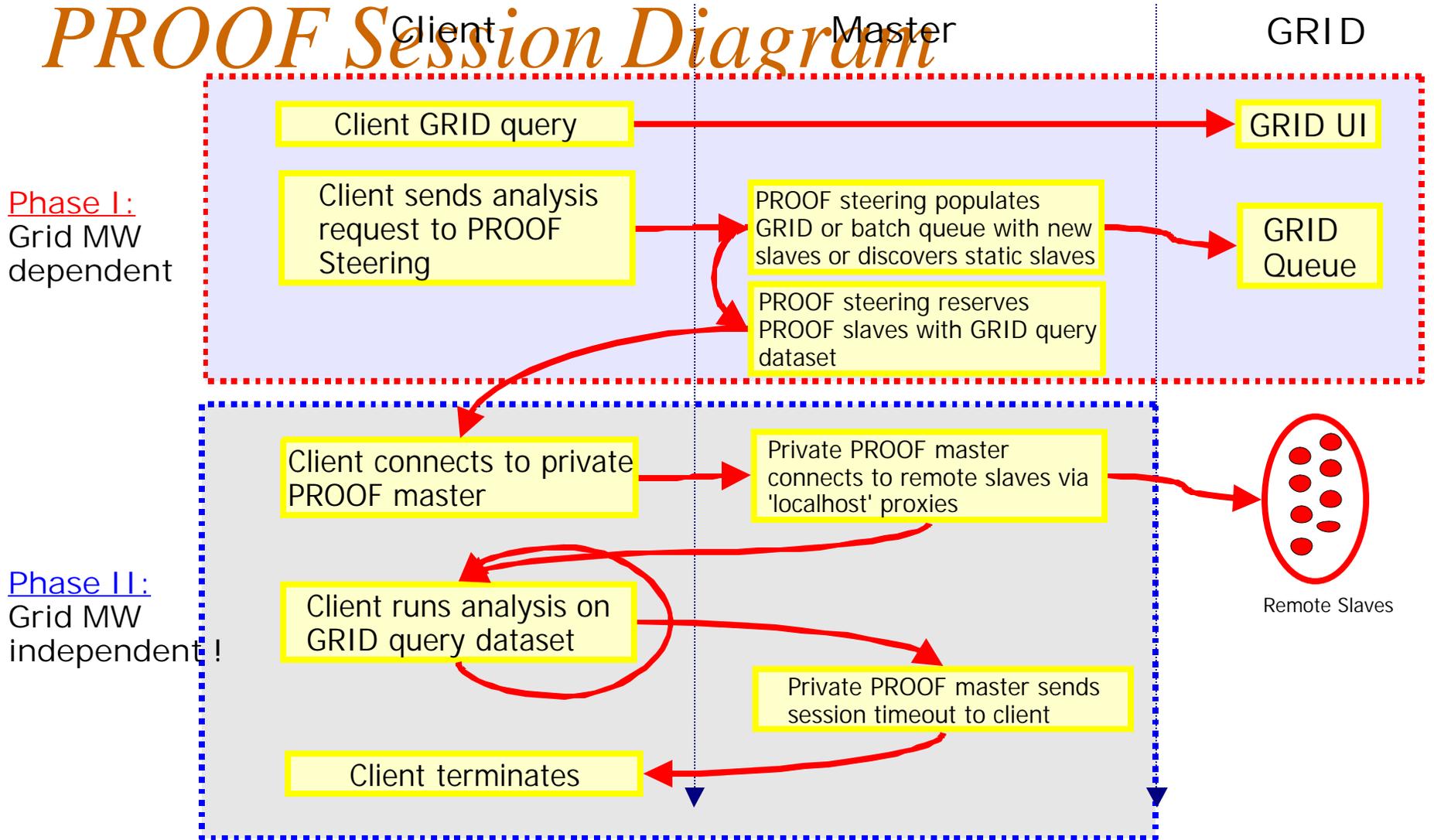


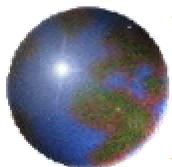






# PROOF Session Diagram





## *Conclusions*

- ✦ The PROOF system on local clusters provides efficient parallel performance on up to  $O(100)$  nodes
- ✦ Combined with Grid middleware it becomes a powerful environment for “interactive” parallel analysis of globally distributed data
- ✦ ALICE plans to use the PROOF/GLite combo for phase 3 of its PDC'04