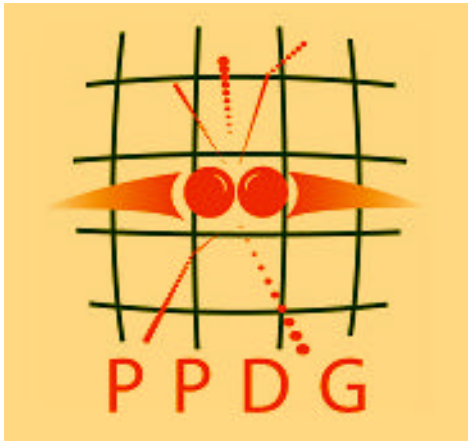
Clarens remote analysis enabling environment

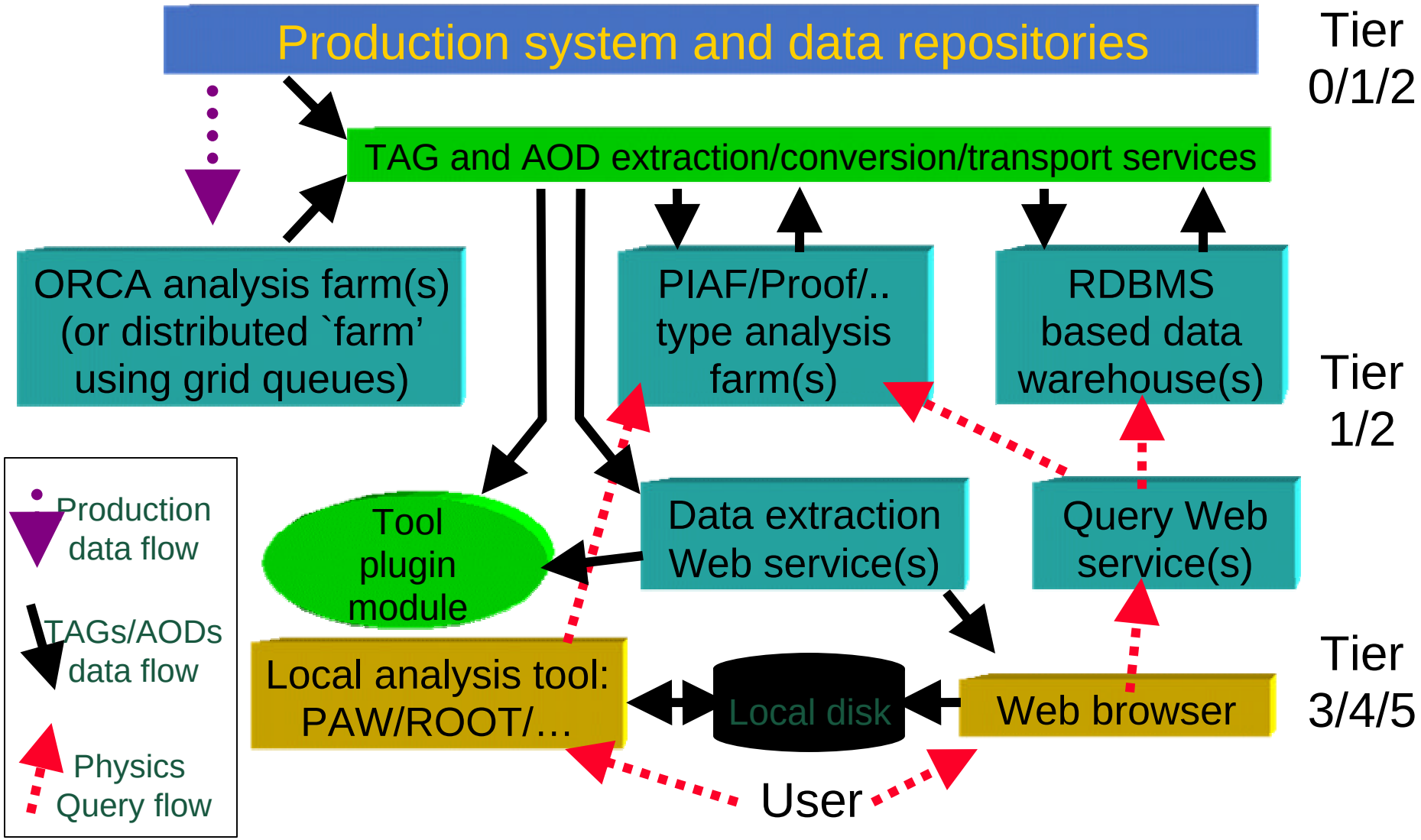
May 22, 2002



E. Aslakson, J. Bunn, K. Holtman, S. Iqbal, I. Legrand, V. Litvin, H. Newman, E. Soedarmadji, S. Singh, C. D. Steenberg, R. Wilkinson

Varied components and data flows

Many promising alternative service providers



Demonstration for ROOT

- Examples of distributed analysis services
  - PIAF/PROOF
  - RDBMS-based data warehouse (mysql, pgsql done)
  - Remotely stored ROOT files (rootd, webfile)
  - Remote TAGs/AODs
  - Grid services (files movement, job submission)
- Need at least protocols for:
  - RDBMS access (ODBC/JDBC/Native DB connect)
  - Remote file access (maybe through Globus)
  - Some connection to access remote TAGS
- Now repeat the process for a web front-end, Lizard, JAS, C++ code etc.

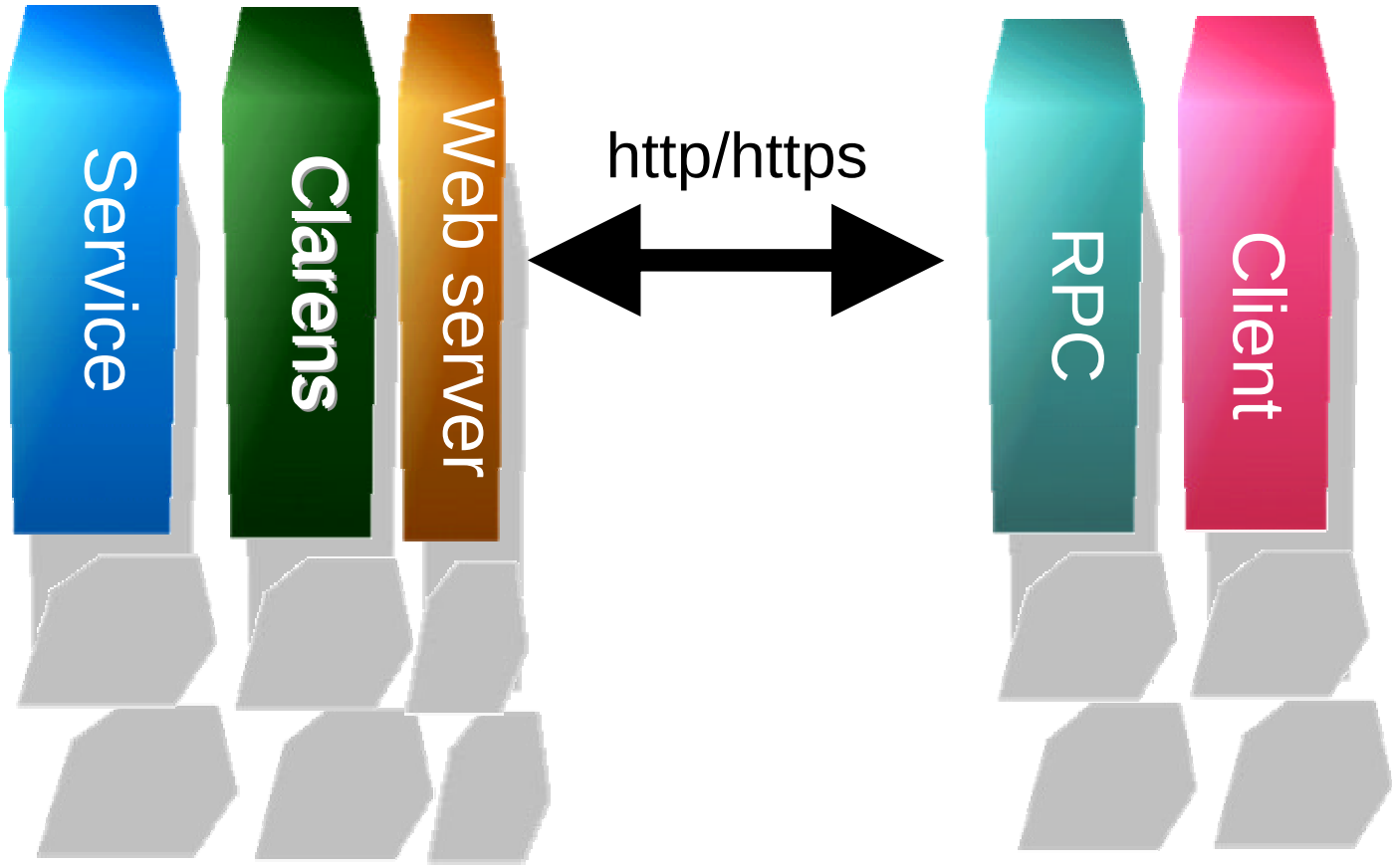
  Remember that a web front-end is just another client of the service.

Clarens Architecture I

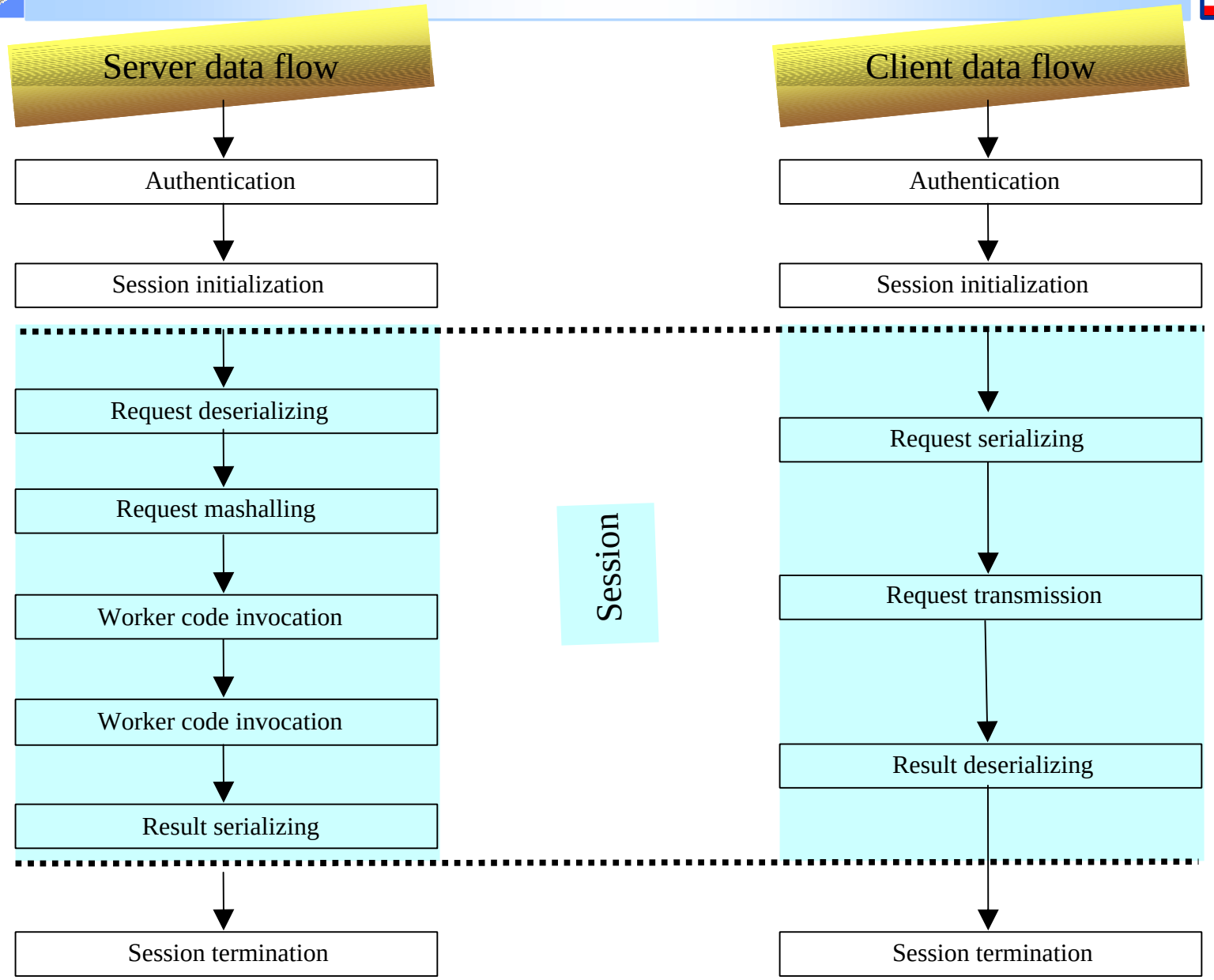
- **Common protocol** spoken by all types of clients to all types of services
- Implement service **once** for **all clients**
- Implement client access to service once for each client type using ***common protocol*** *already implemented for "all" languages (C++, Java, Fortran, etc. :-)*
- *Common protocol is XML-RPC with SOAP close to working, CORBA doable, but would require different server above Clarens (uses IIOP, not HTTP)*
- *Handles **authentication** using Grid certificates, **connection** management, data **serialization**, optionally **encryption***
- *Immplementation uses **stable**, well-known server infrastructure (**Apache**) that is debugged/audited over a long period by many*
- *Clarens layer itself implemented in Python, but can be reimplemented in C++ should performance be inadequate*

Clarens Architecture II

Diagram:



Clarens Architecture III



Server notes

- **Server consists of a pool of processes**
- **Any process can respond to a request**
- **Every RPC call is a request**
- **Server processes cannot make the main server crash - bad compiled modules only affect that service**

- **Session data stored in an embedded Berkeley DB**
- **Other objects may be re-used by subsequent connections to the same process - e.g. Connections to a RDBMS**
- **Compiled modules linked against shared libraries need to have library search path set in the .so file (RPATH for elf files)**
- **Compiled modules using shared libraries that apache or python was linked against, must use the same version - e.g. can't use Berkeley DB4 if apache/python was linked against DB3**

Server installation

Directory structure:

<top>/
- clarens_server.py
- /echo
  - __init__.py
  - method echo
- /objy
  - __init__.py
  - method objy
  - clarens_int.so (compiled module - ObjyDB)
- /file
  - __init__.py
  - method file
- ~clarens/
  - myservice/
    - __init__.py
    - method mymethod

Dependencies:

apache, python, py-xmlrpc, m2crypto, Berkeley-DB3/4, py-bsddb3

Server example

Access OBJY Tags/Histograms

```
method_list={"getDBs":              getDBs,
             "getHistos":           getHistos,
             .......
             "getTagColumnInfo":getTagColumnInfo
            }

def getDBs(req,method_name,args): # Handler method
    try:
        dblist <Get list of databases>
    except:
      return apache.HTTP_ERROR
    req.write(encode(dblist))
    return apache.OK
```

Server example II

File access:

```
def read(req,name,offset,len):
<write len bytes from file name from offset>
return apache.OK

method_list={"read":   read}
```

Client example

Access OBJY Tags/Histograms

Python:

```
dbsvr=Clarens.clarens_client('http://host/xmlrpc/clarens_server.py')
dbsvr.objy.getDBs('/home/mydir/myfederation.boot')
```

Root:

```
dbsvr=Clarens('http://host/xmlrpc/clarens_server.py')
dbsvr.call('objy.getDBs','/home/mydir/myfederation.boot')
```

Gives a list of databases in 'myfederation'

Call hierarchy (objy.GetDBs):

- objy=dir to place file in
- GetDBs=method name

To implement new service:

- Create the above
- Create methods in 'method list'

Client example II

File access

Python:

```
dbsvr=Clarens.clarens_client('http://host/xmlrpc/clarens_server.py')
dbsvr.file.read('/home/myfile.root')
dbsvr.logout()
```

Root:

```
dbsvr=Clarens('http://host/xmlrpc/clarens_server.py')
TCWebfile F(dbsvr,'/home/myfile.root')
TCWebfile G(dbsvr,'~user/myfile.root')
TBrowser T;
dbsvr.logout()
```

- Browse file with root tree browser
- Read file like any other
- Seek method is implemented on client side
- File offset is persistent data stored by the client

Conclusions

- Clarens is a simple way to implement web services on the server
- Provides some basic connectivity functionality common to all services
- Uses commodity protocols
- No Globus needed on client side, only certificate
- Simple to implement clients in scripts and compiled code
- Needed: more services, E.g.:
  - data warehouse service (Saima)
  - NTUPLE streaming from reconstructed data (Edwin, Julian)
  - Reconstruction on demand (?)
  - Job submission
- Also TODO: implement discovery mechanisms