New XRootD client plug-ins

Łukasz Janyst

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

XRootD

Federated Storage Workshop
Menlo Park, 11.04.2014

CERN

Outline

- The new client - XRootD 4.0.0
- XrdCl and its API stack

- The plug-in mechanism:
  - Where?
  - What?
  - How?
  - Why?/What for?
  - When?

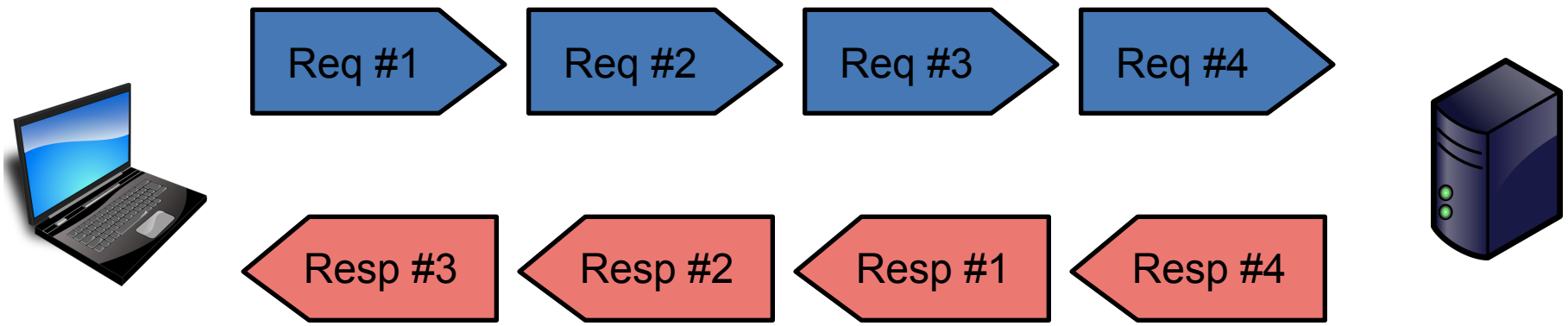
The new XRootD client

- First released with **XRootD 3.3.0**

- New client library: **libXrdCl.so**
- New command line utilities:
  - **xrdcopy** - replacement for xrdcp (the same interface)
  - **xrdfs** - replacement for xrd (new interface)

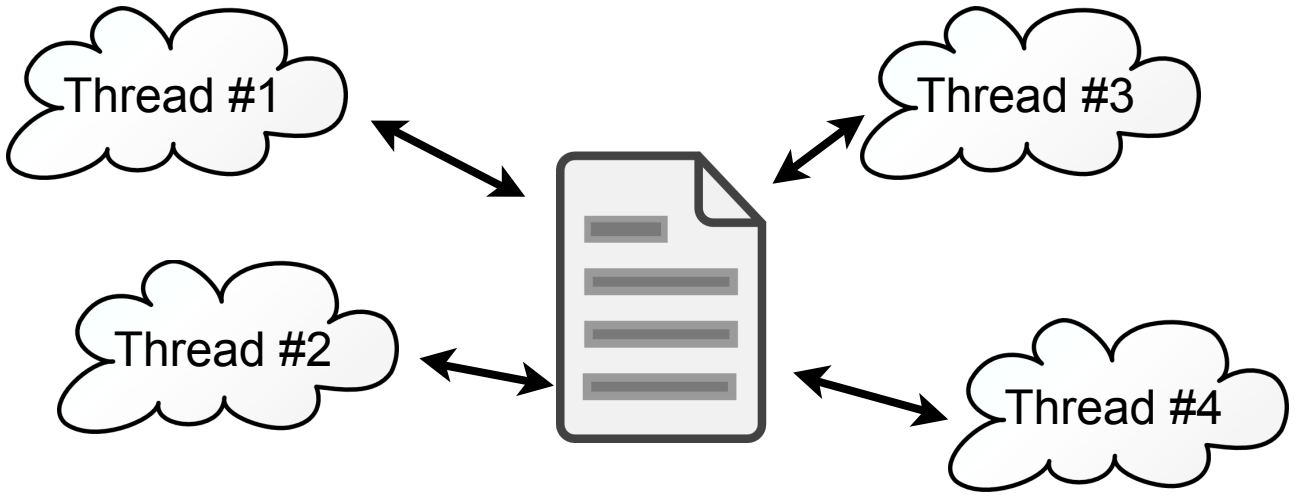
- Default in **XRootD 4.0.0**:
  - Old client (XrdClient) officially deprecated
  - xrdcp becomes a symlink to xrdcopy

Async & stream multiplexing



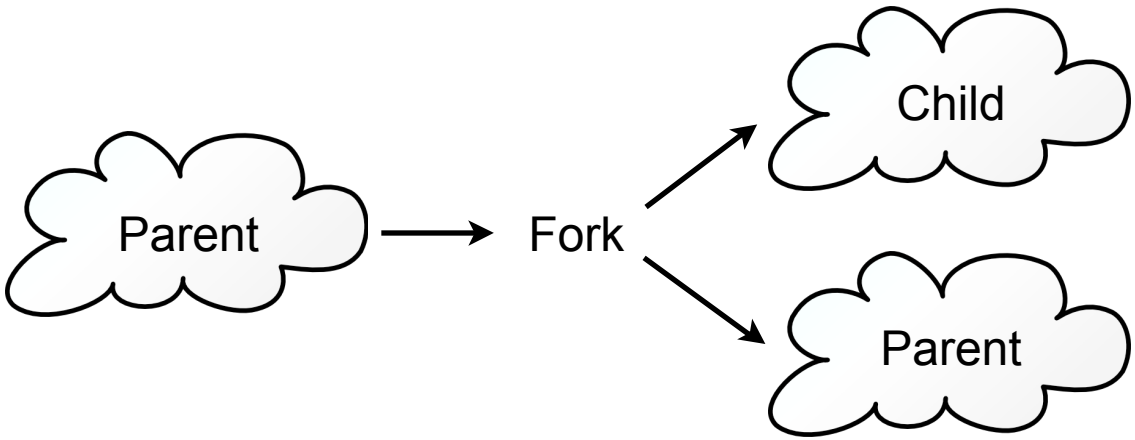
- The XRootD protocol supports virtual streams
- There may be many requests outstanding and the server may respond in the order it chooses
- The new client handles responses as soon as they come calling the user call-back function, the order is unimportant

Thread safe/aware



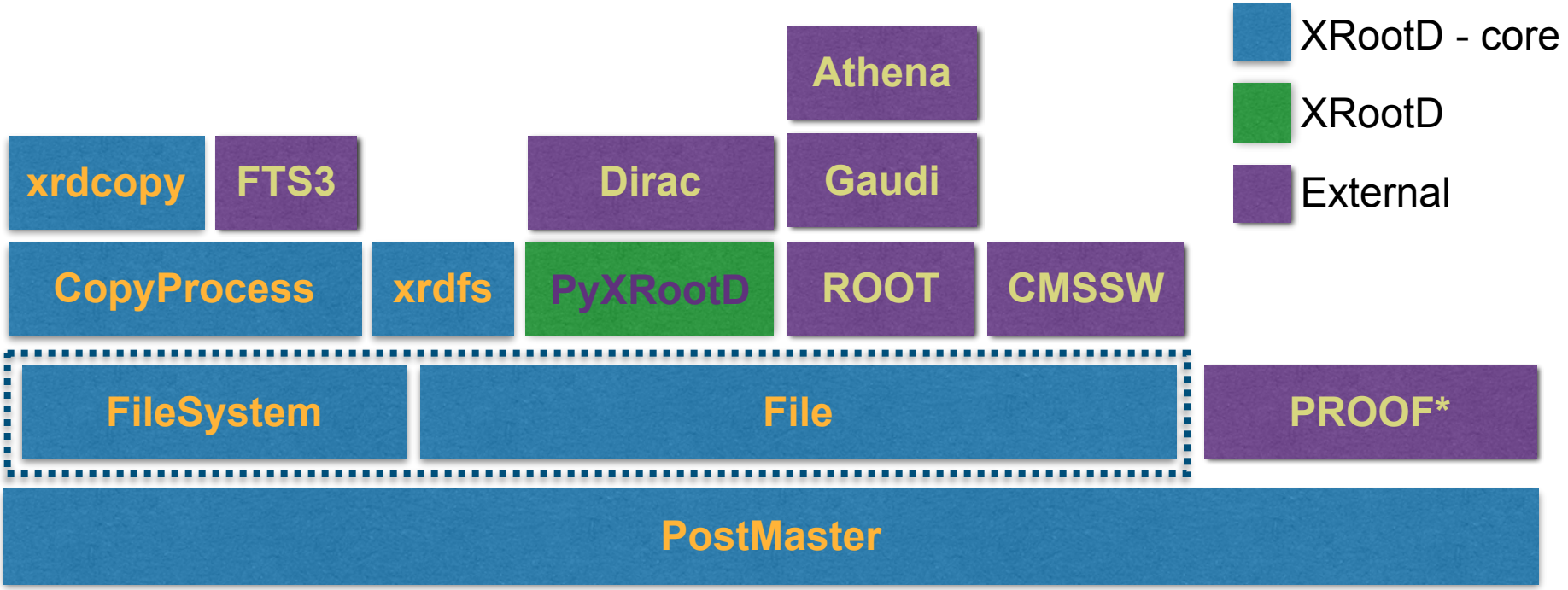
- File and FileSystem objects can be safely accessed from multiple execution threads
- Internally uses a worker thread pool to handle call-backs

Fork safe

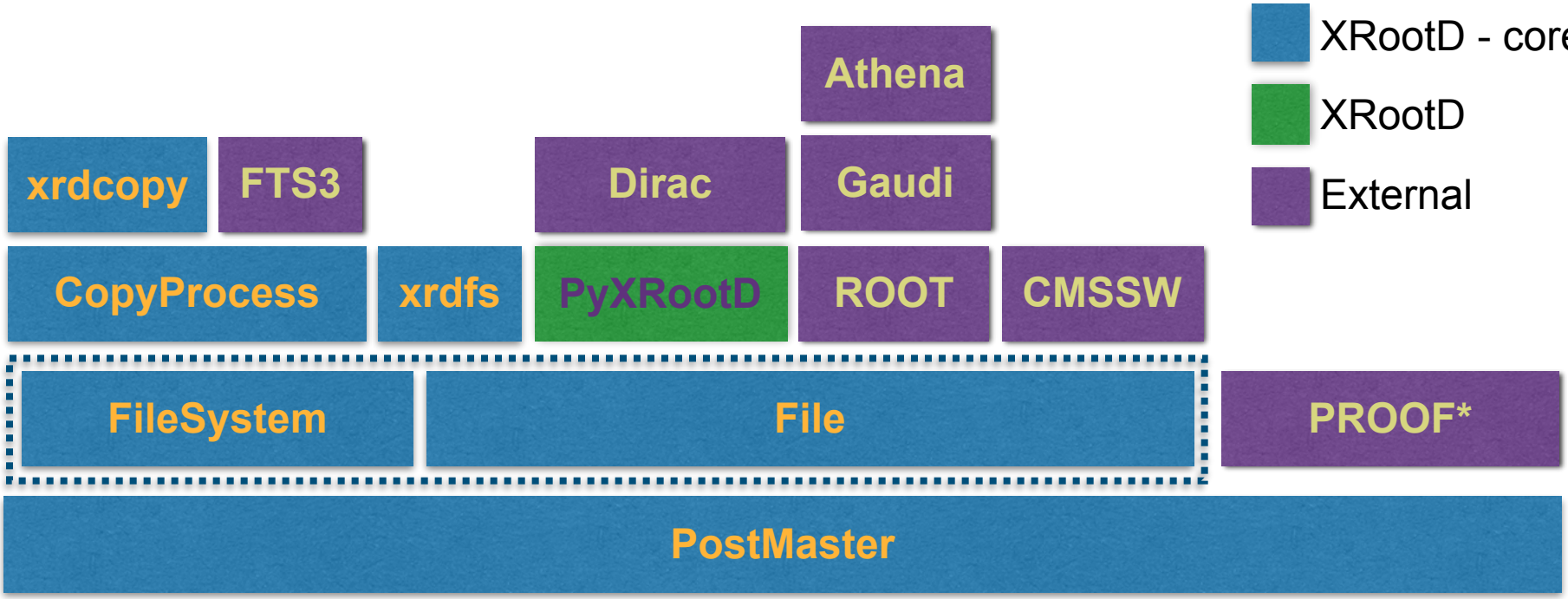


- Can handle forking even when the IO operations are in progress
- File and FileSystem objects remain valid in both parent and child
- The operations in the parent continue after the fork
- The objects in the child will run recovery procedure (like in the case of a broken connection)

XrdCl API stack & neighbours

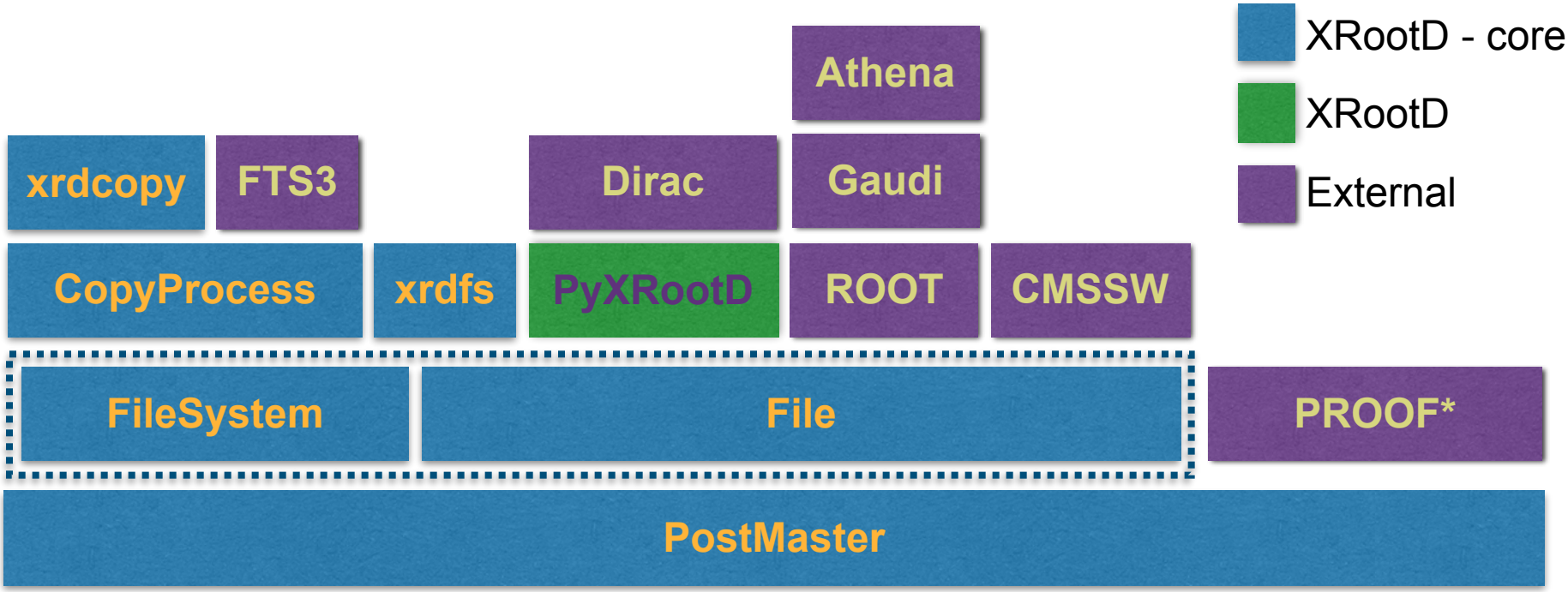


XrdCl API stack & neighbours



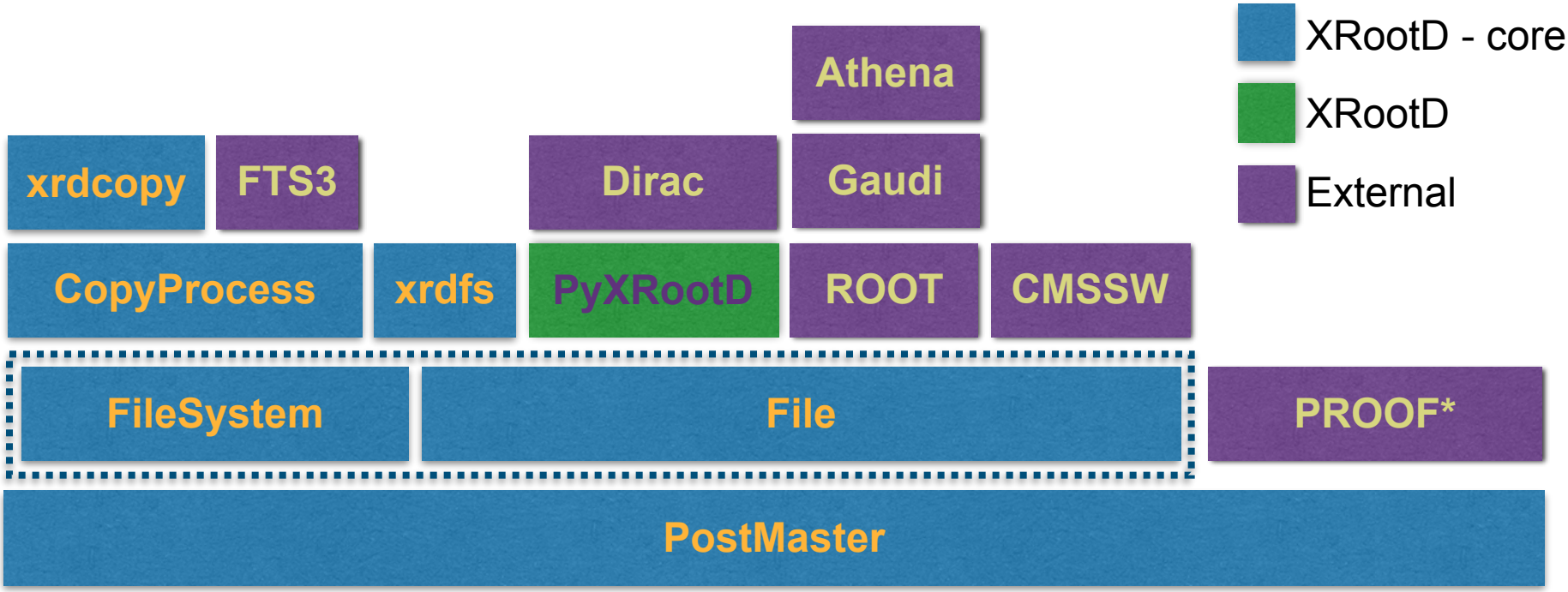
- **PostMaster** - low-level message handling API
  - sends messages
  - asynchronous - notifies message handlers about sent/ incoming messages
  - notifies about stream status changes (disconnections)

XrdCl API stack & neighbours



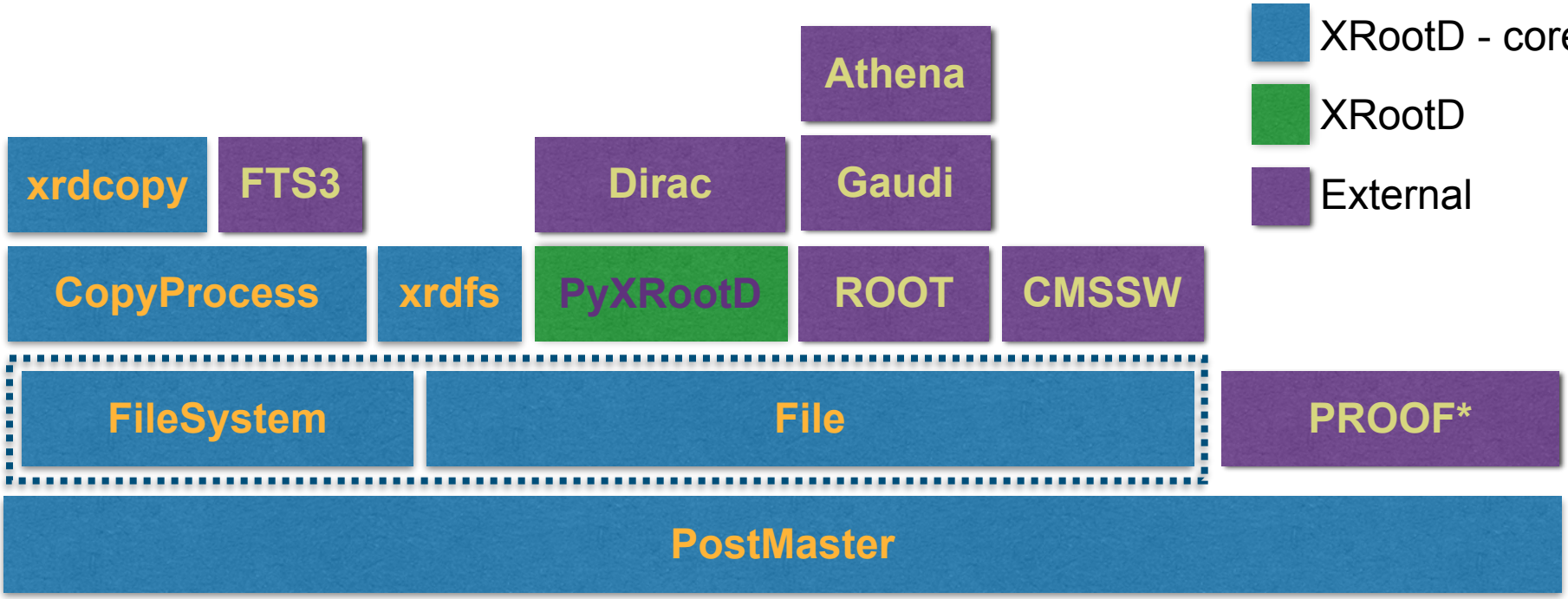
- **File/FileSystem** - implement XRootD operations
  - user-facing C++ API
  - does reads, writes, mkdirs, listings, staging and the like
  - asynchronous - call back when response is read

XrdCl API stack & neighbours



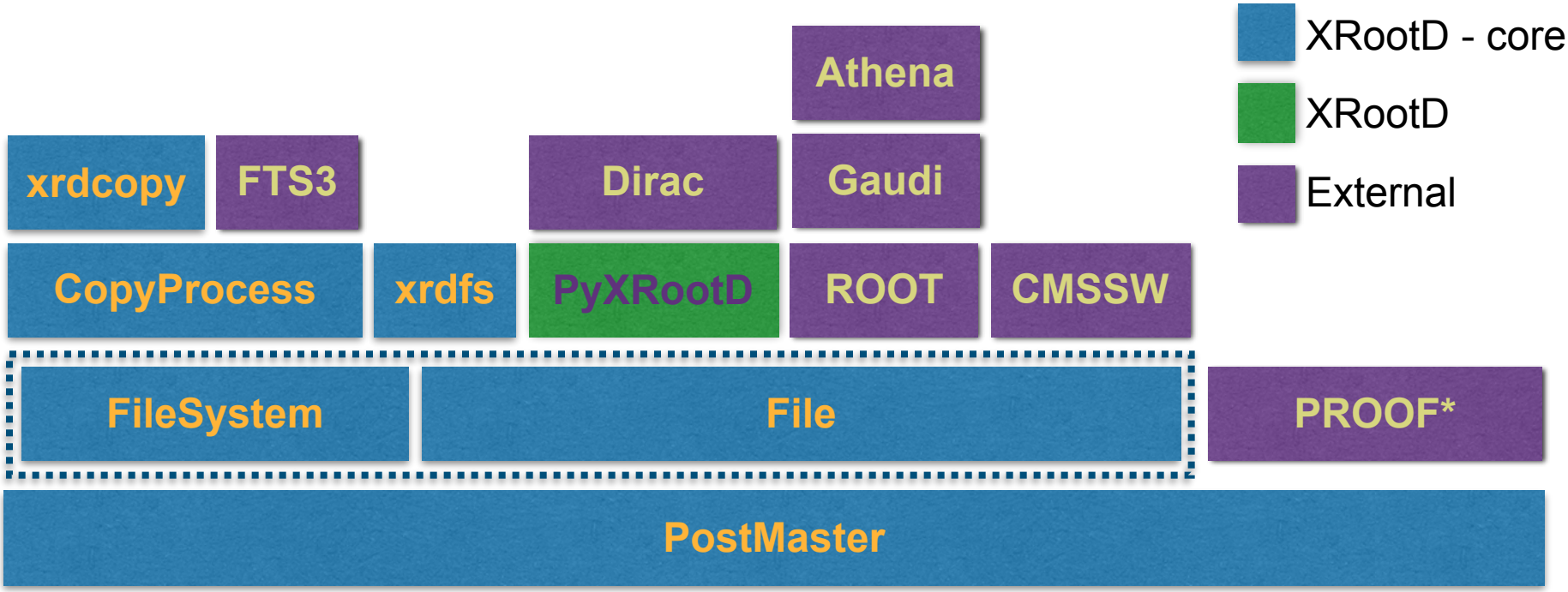
- **CopyProcess** - implement copy operations
  - user-facing C++ API, a library call
  - take source/target URLs, couple more parameters and do the magic
  - notify periodically about progress using a call-back

XrdCl API stack & neighbours



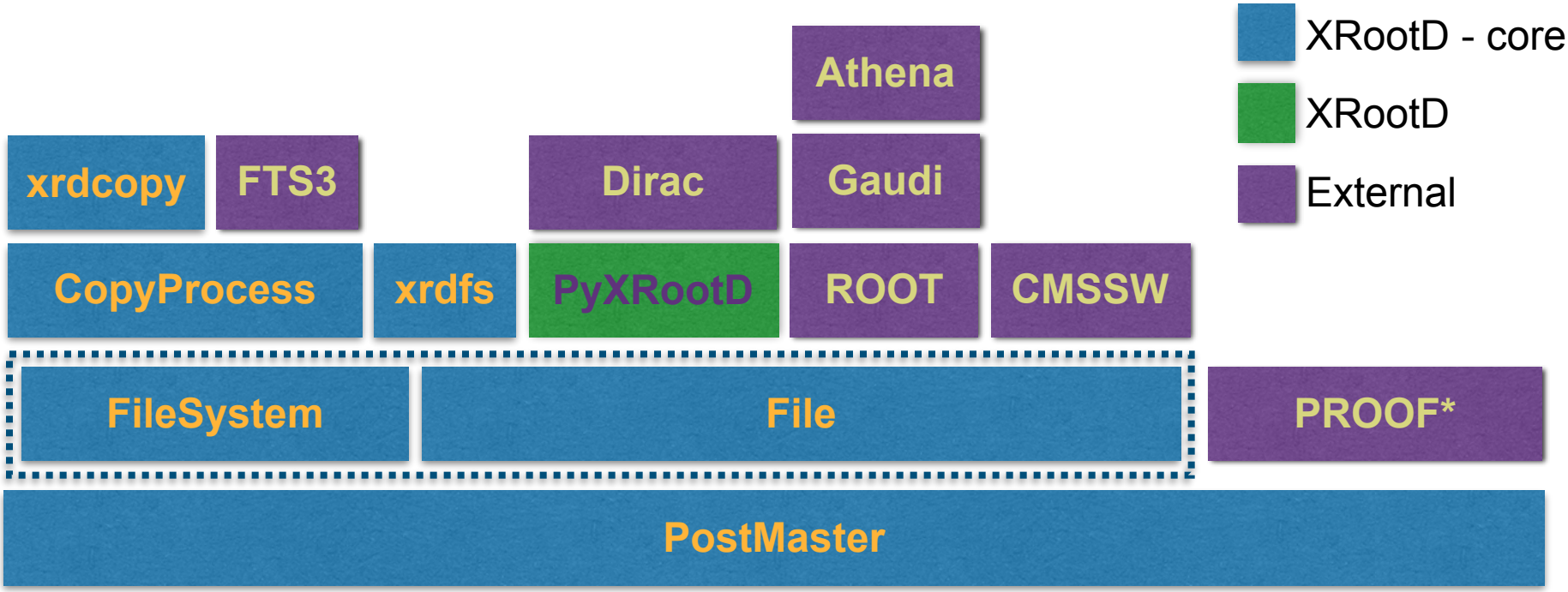
- **xrdcopy** - the copy command
  - translate command line parameters to CopyProcess calls
- **xrdfs** - the meta-data command
  - translate command line parameters to FileSystem calls

XrdCl API stack & neighbours



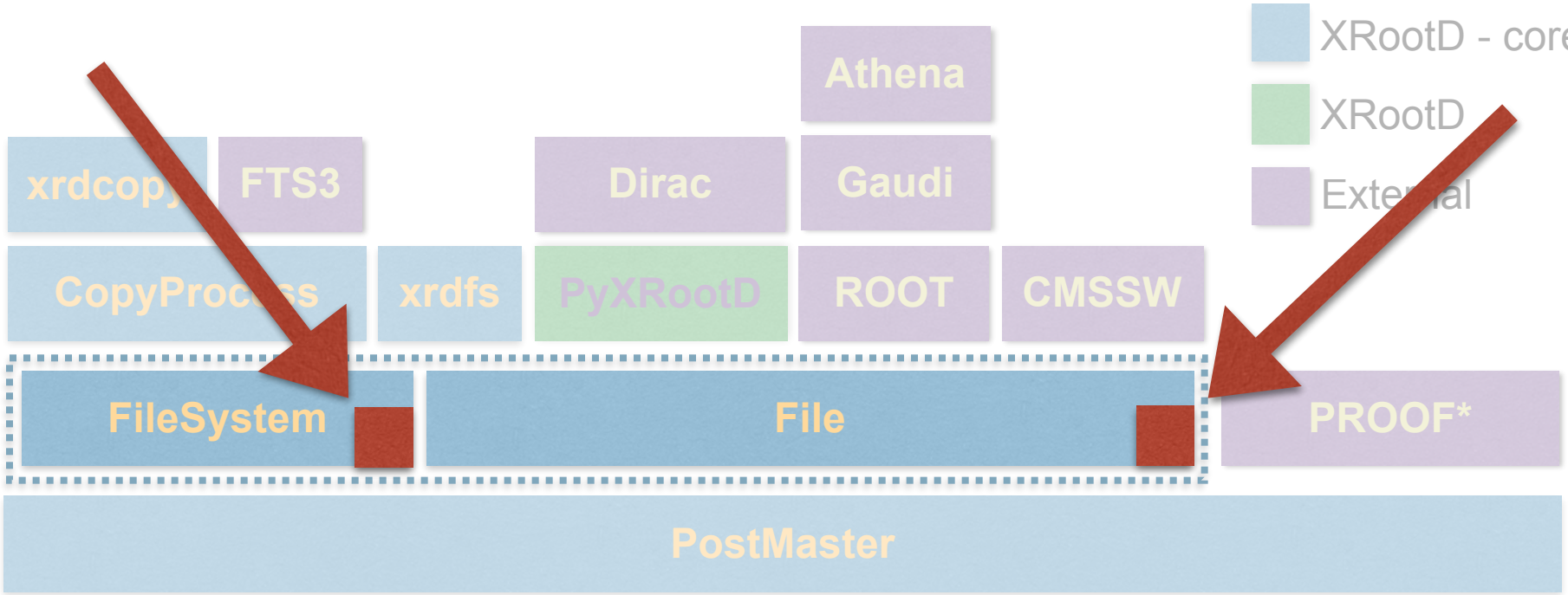
- **PyXRootD** - make the user APIs available in Python
  - all APIs user APIs available: File, FileSystem and CopyProcess
  - uses Python callables to handle call-backs

XrdCl API stack & neighbours



- **Neighbours** - all the purple boxes above and quite a bit more that did not fit the diagram
  - **EOS** uses a bit of everything
  - significant number of packages use **XrdPosix** interface

XrdCl plug-ins



- **Plug-ins** - replace the original implementation
  - both for File and FileSystem objects
  - all calls can be replaced
  - all the layers above can benefit without changing a single line of code

XrdCl user API

- All xroot protocol requests are implemented as asynchronous methods
- The calls queue the request and return, **never block**

```
XRootDStatus File::Open( const std::string &url,
                         OpenFlags::Flags   flags,
                         Access::Mode       mode,
                         ResponseHandler   *handler,
                         uint16_t           timeout )
```

- The response handler is called when the response is ready
- Synchronous versions implemented in terms of asynchronous ones, with a semaphore

XrdCl plug-in API

- The plug-in API is exactly the same - except for the **virtual** keyword
- Only asynchronous calls may be overloaded

```
virtual
XRootDStatus File::Open( const std::string &url,
                         OpenFlags::Flags   flags,
                         Access::Mode       mode,
                         ResponseHandler   *handler,
                         uint16_t           timeout )
```

Plug-in Manager

- Process plug-in **environment** configuration
  - covered later in the presentation
- Manage a map between URLs and plug-in factories:
  - ie. `root://eosatlas.cern.ch:1094` ▷ `XrdEosFactory`
  - factories are objects that instantiate plug-ins (think of: `new XrdEosFile`) given URLs

Flow

- **File object creation** (constructor)
  - ask the plug-in manager whether a plug-in for a given URL is known
  - if so, install the plug-in
- **File object usage** (method calls)
  - call the plug-in if it is installed
  - call the normal XRootD code if there is no plug-in present

Deployment - config files


```
]==> cat eos.conf
# example configuration

url = eosatlas.cern.ch;eoscms.cern.ch
lib = /usr/lib64/libXrdEosClient.so
enable = true
customarg1=customvalue2
customarg2=customcalue2
```

- The plug-ins are discovered and configured by scanning configuration files
- There is one config file per plug-in
- It's a set of key value pairs

Deployment - search paths

- The plug-in manager will search for global configuration files in:

```
/etc/xrootd/client.plugins.d/
```

Deployment - search paths

- The plug-in manager will search for global configuration files in:

```
/etc/xrootd/client.plugins.d/
```

- The global settings may be overridden by configuration files found in:

```
~/.xrootd/client.plugins.d/
```

Deployment - search paths

- The plug-in manager will search for global configuration files in:

```
/etc/xrootd/client.plugins.d/
```

- The global settings may be overridden by configuration files found in:

```
~/.xrootd/client.plugins.d/
```

- Any of the previous settings may be overridden by configuration files found in a directory pointed to by:

```
XRD_PLUGINCONFDIR
```

Plug-in packaging

- Plug-ins may be developed and distributed **independently** of the XRootD code
- The plug-in manager performs strict interface **version checking**
  - will refuse to load ABI incompatible plug-ins
- Plug-in package (RPM) needs to contain:
  - the plug-in shared library
  - a config file in /etc/xrootd/client.plugins.d/

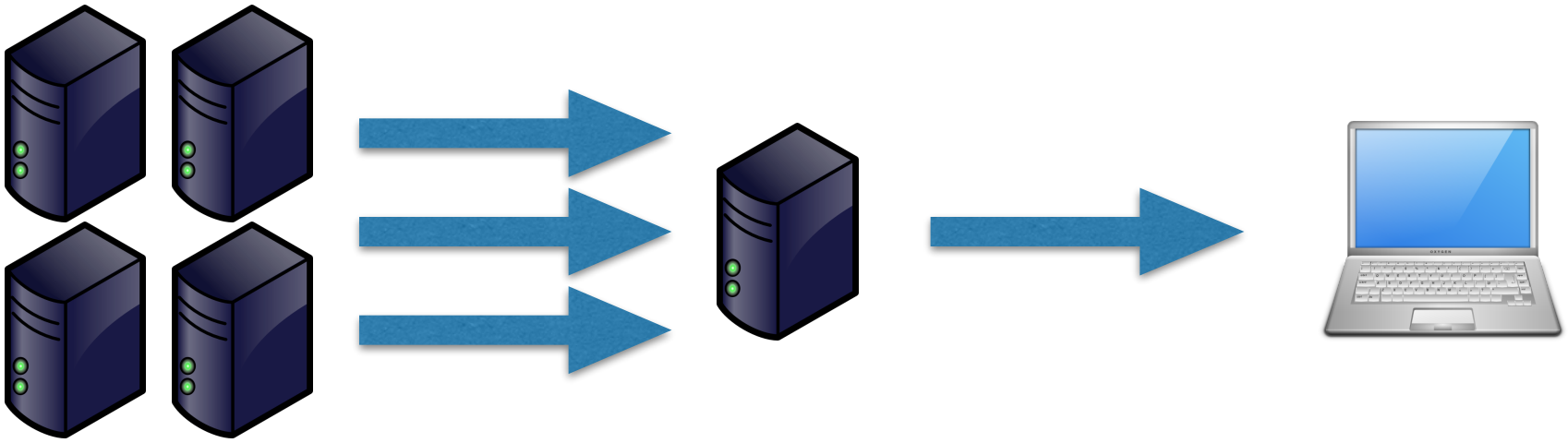
Primary motivation

RAIN

Redundant **A**rray of **I**ndependent **N**odes

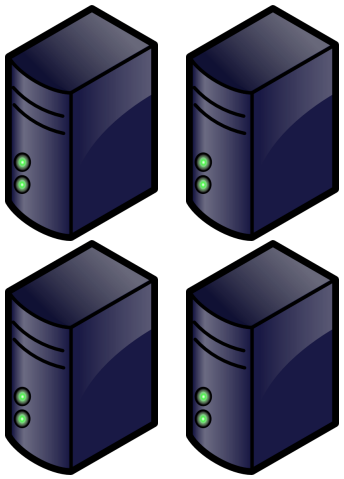
- Stripe files and use erasure coding to increase fault tolerance
- Primarily for archiving and similar use-cases
- Multiple techniques:
  - Hamming parity
  - Reed-Solomon error correction
  - Low-density parity-check

EOS Erasure Coding - now

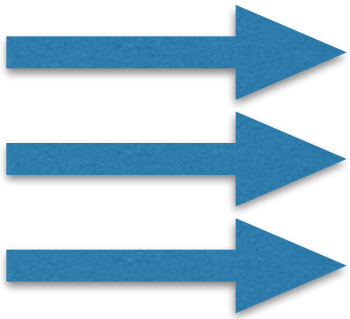


- The client needs to see the file as a whole
- File reconstruction needs to be done at a gateway
- CPU and bandwidth scalability issues

EOS Erasure Coding - 4.0.0



Stripe servers



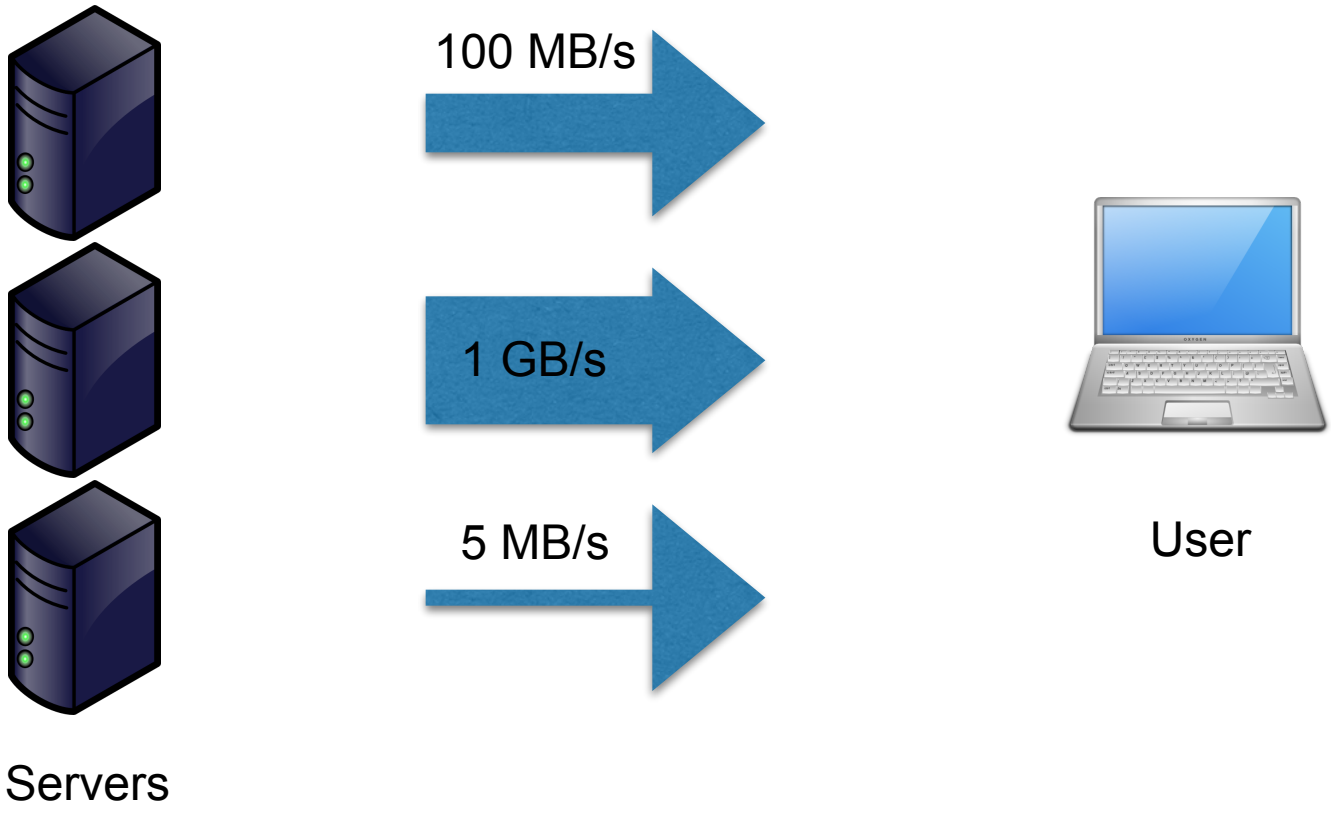
User

- When contacting EOS the client is able to execute specialised code
- Can contact the stripe servers directly
- Can reconstruct the data at the client machine
- **Transparently to the users - whatever they are!**

Consequences

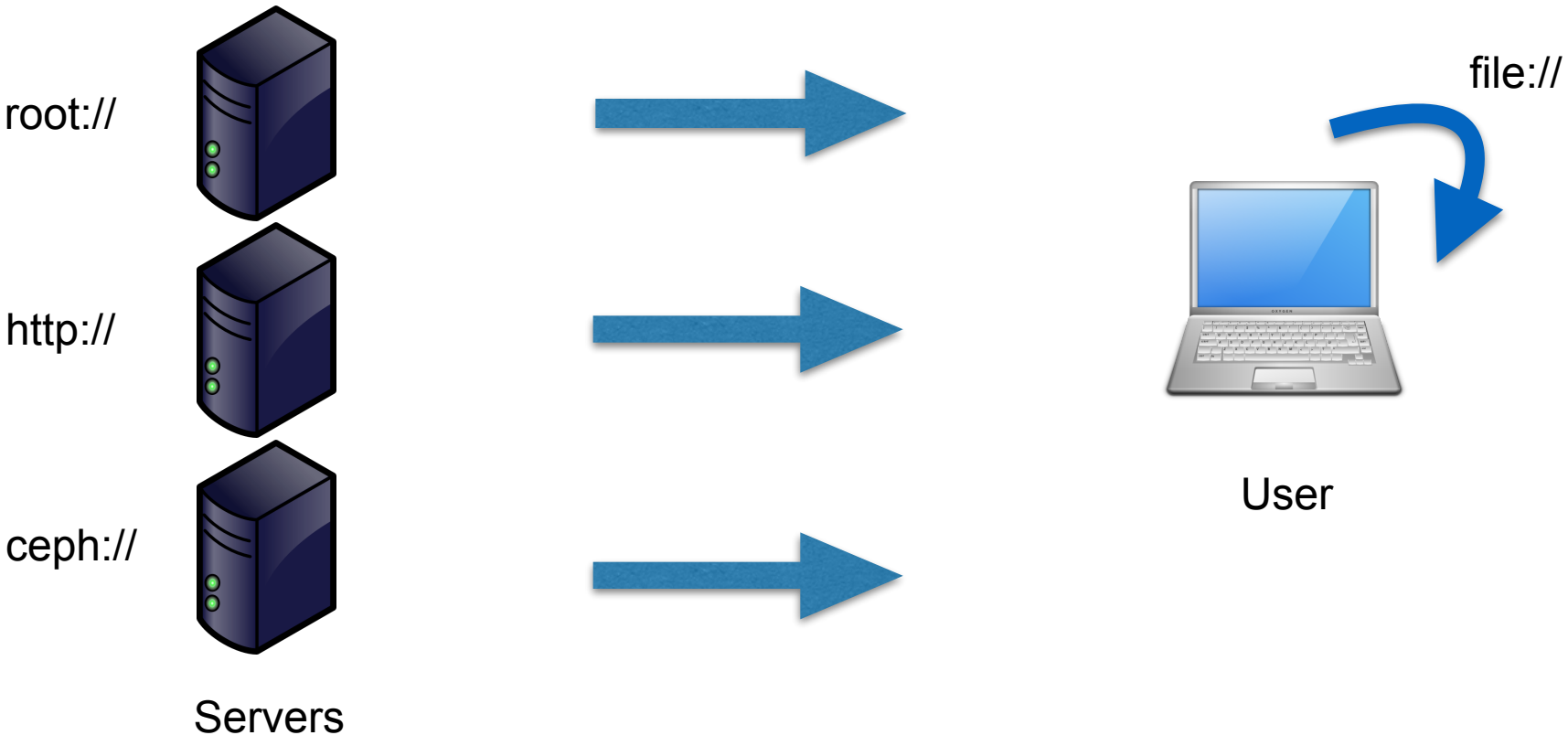
- Client plug-ins provide a way for the XRootD community to **play, tinker and hack** the client
  - exactly what made the XRootD server so successful!
- All possible calls may be overridden
- Everything is **transparent** to the layers above!

Client-side load balancing



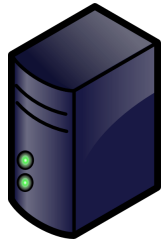
Open multiple files at the same time and fetch data from the fastest server - CMS-style as presented by Brian Bockelman at CHEP 2013.

Redirect to other protocols



Redirect and transparently handle other protocols if needed.

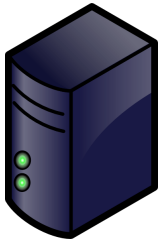
Caching and monitoring

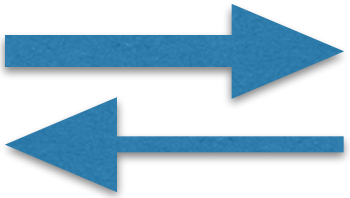


Server

User

Cache the data locally at the user box and re-use when needed.





Server

User

Gather and send back custom monitoring information to the server.

Conclusion

- The plug-ins are:
  - **flexible** - override all possible calls, do whatever you want
  - **independent** - development and deployment may be completely detached from XRootD core
  - **ready to play with** in XRootD 4
  - give a possibility to freely **experiment** and then incorporate new things into the core package

Thanks!

Thanks for your attention!

Questions? Comments?

Notes

- Most of the artwork in this presentation comes from: Open Icon Library