Xrootd explained

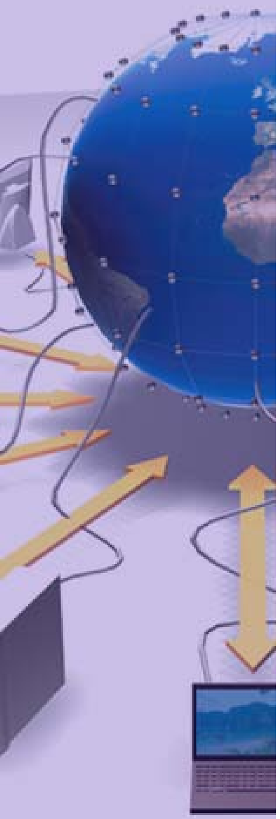
A general introduction

Fabrizio Furano

CERN IT/GS

Sept 2008

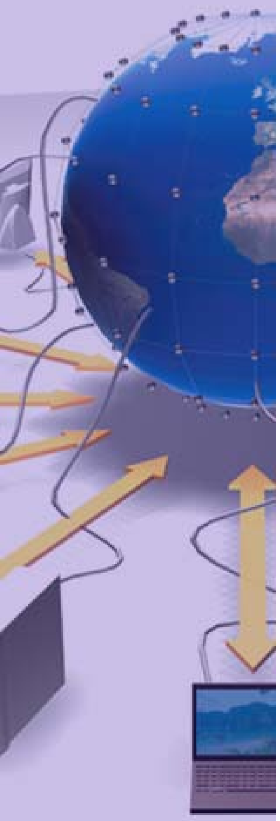
CERN Fellow seminars



http://savannah.cern.ch/projects/xrootd

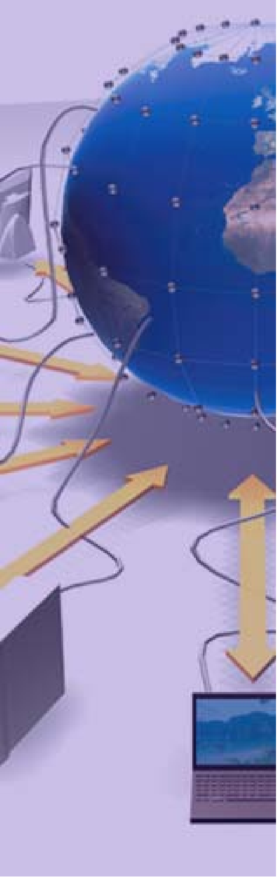
http://xrootd.slac.stanford.edu

The historical Problem: data access



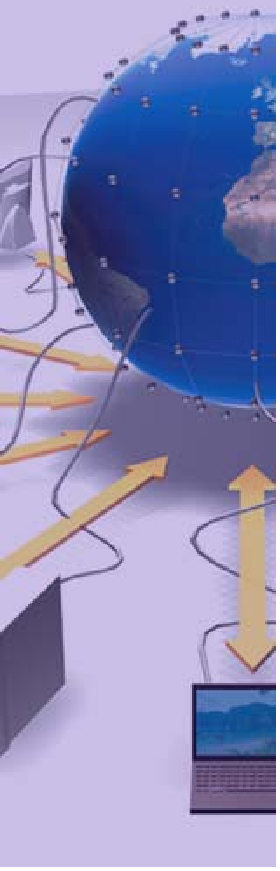
- Physics experiments rely on rare events and statistics
  - Huge amount of data to get a significant number of events
    - The typical data store can reach 5-10 PB… now
    - Millions of files, thousands of concurrent clients
      - Each one opening many files (about 100-150 in Alice, up to 1000 in GLAST)
      - Each one keeping many open files
  - The transaction rate is very high
    - Not uncommon O(10^3) file opens/sec per cluster
      - Average, not peak
      - Traffic sources: local GRID site, local batch system, WAN
- Need scalable high performance data access
  - No imposed limits on performance and size, connectivity
    - Do we like clusters made of 5000 servers? We MUST be able to do it.
  - Need a way to avoid WN under-utilization

What is Scalla/XRootd?



- The evolution of the BaBar-initiated xrootd project (SLAC+INFN-PD)
- Data access with HEP requirements in mind
  - – But a completely generic platform, however
- Structured Cluster Architecture for Low Latency Access
  - – Low Latency Access to data via xrootd servers
    - POSIX-style byte-level random access
      - – By default, arbitrary data organized as files
      - – Hierarchical directory-like name space
    - Protocol includes high performance features
    - Exponentially scalable and self organizing
  - – Tools and methods to cluster, harmonize, connect, …

The XRootd “protocol”

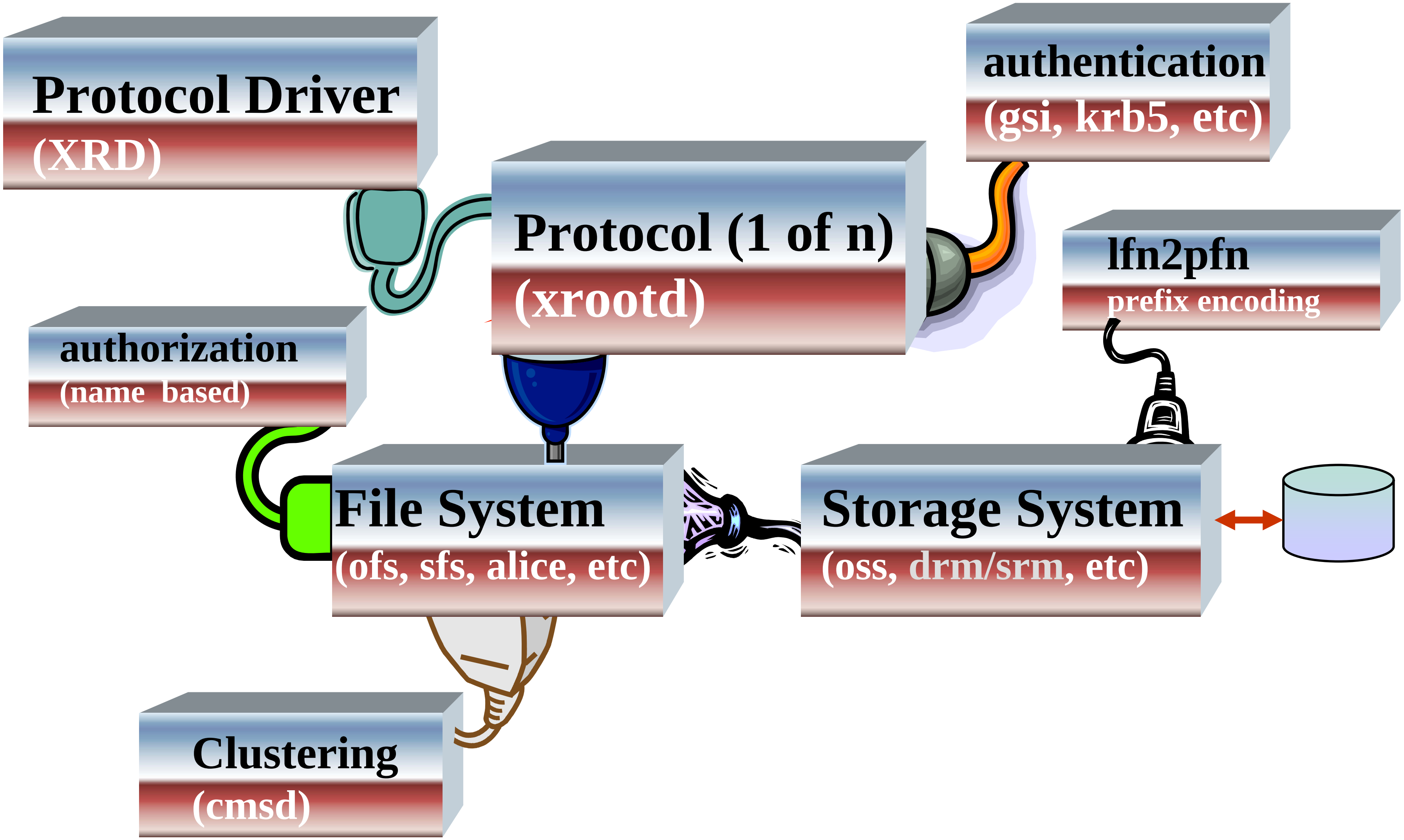


- More or less, with Scalla/Xrootd come:
  - An architectural approach to data access
    - Which you can use to design your scalable systems
  - A data access protocol (the xrootd protocol)
  - A clustering protocol
  - A set of sw pieces: daemons, plugins, interfaces, etc.
  - When someone tests Xrootd, he is:
    - Designing a system (even small)
    - Running the tools/daemons from this distribution
    - The xrootd protocol is just one (quite immaterial) piece
- There are no restrictions at all on the served file types
  - ROOT, MPEG, TXT, MP3, AVI, …..

The Xrootd “protocol”

- The XRootD protocol is a good one
  - But doesn’t do any magic
    - It does not multiply your resources
    - It does not overcome bottlenecks
  - One of the aims of the project still is sw quality
    - In the carefully crafted pieces of sw which come with the distribution
  - What makes the difference with Scalla/XRootD is:
    - Scalla/XRootD Implementation details (performance + robustness)
      - And bad performance can hurt robustness (and vice-versa)
    - Scalla SW architecture (scalability + performance + robustness)
    - You need a clean design where to insert it
    - Sane deployments (The simpler, the better – á la Google!)

xrootd Plugin Architecture



Protocol Driver
(XRD)

authentication
(gsi, krb5, etc)

Protocol (1 of n)
(xrootd)

lfn2pfn
prefix encoding

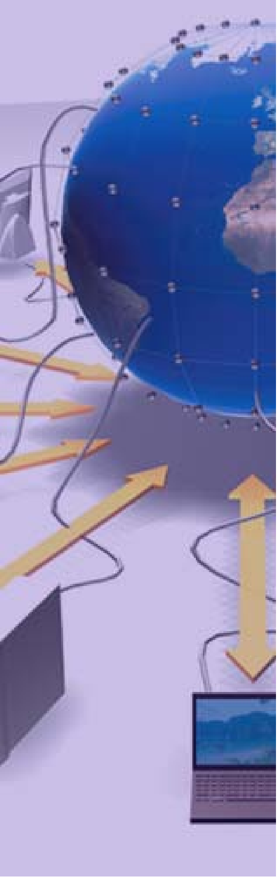
authorization
(name based)

File System
(ofs, sfs, alice, etc)

Storage System
(oss, drm/srm, etc)

Clustering
(cmsd)

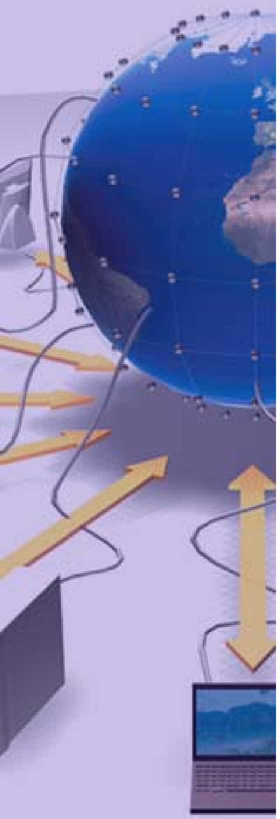
Different usages



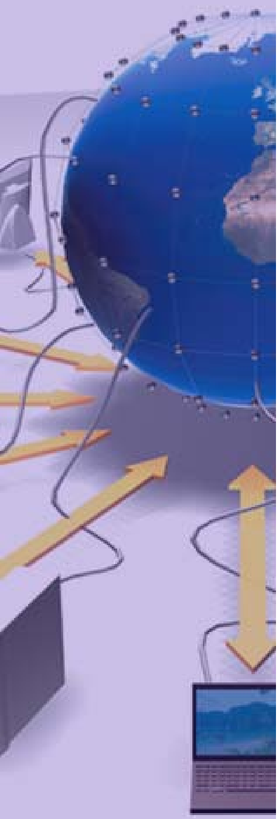
- Default set of plugins :
  - Scalable file server functionality
    - Its main historical function
  - To be used in common data mngmt schemes
- Example:The ROOT framework bundles it as it is
  - And provides one more plugin: XrdProofdProtocol
  - Plus several other ROOT-side classes
  - The heart of PROOF: the Parallel ROOT Facility
    - A completely different task by loading a different plugin
      - With a different configuration of course…
    - Massive low latency parallel computing of independent items (‘events’ in physics…)
    - Using the core characteristics of the xrootd framework

File serving

- Very open platform for file serving
  - Can be used in many many ways, even crazy
    - An example? Xrootd-over-NFS-over-XFS data serving
      - BTW Do your best to avoid this kind of things!
      - In general, the really best option is 'local disks' and redundant cheap servers (if you want) + some form of data redundancy/MSS
      - Additional complexity can impact performance and robustness
- Scalla/Xrootd is only one, always up-to-date!
    - http://savannah.cern.ch/projects/xrootd and http://xrootd.slac.stanford.edu
  - Many sites historically set up everything manually from a CVS snapshot
    - Or wrote new plugins to accommodate their reqs
      - Careful manual config (e.g. BNL-STAR)
  - Many others rely on a standardized setup (e.g. several Alice sites, including the CERN Castor2 cluster up to now)
      - More about this later
  - Others take it from the ROOT bundle (e.g. for PROOF)
      - … which comes from a CVS snapshot
      - Again, careful and sometimes very delicate manual config

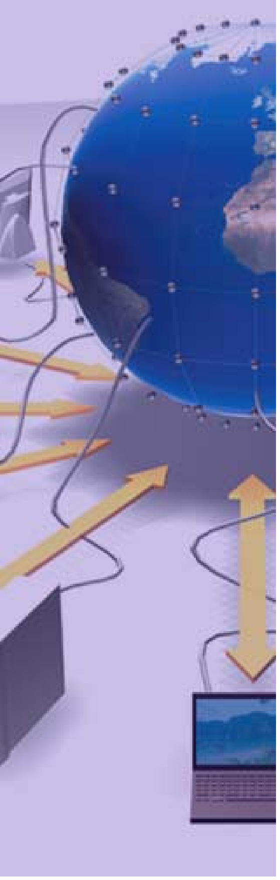


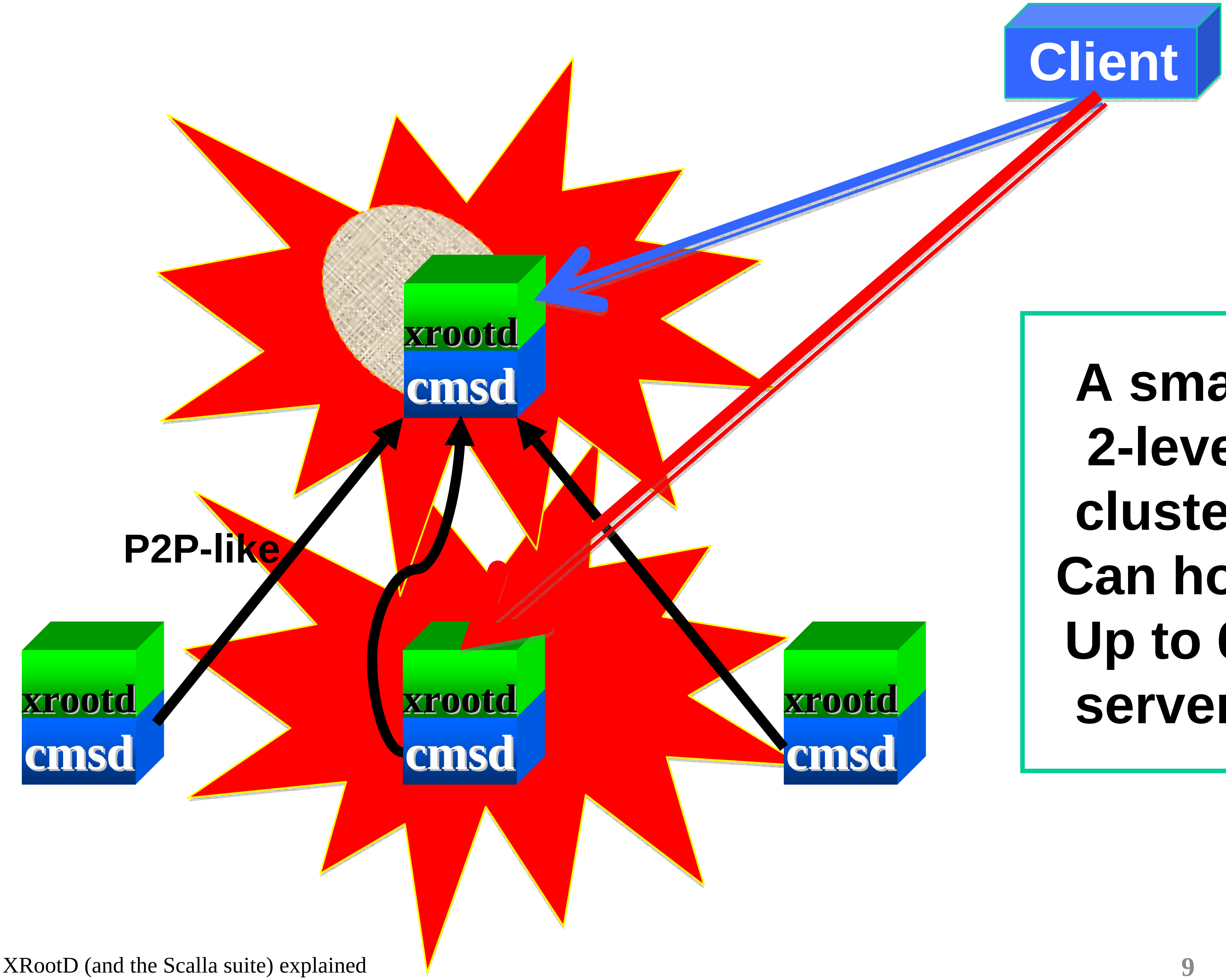
Most famous basic features



- No weird configuration requirements
  - Scale setup complexity with the requirements' complexity. No strange SW dependencies.
- Highly customizable
- Fault tolerance
- High, scalable transaction rate
  - Open many files per second. Double the system and double the rate.
  - NO DBs for filesystem-like funcs! Would you put one in front of your laptop's file system? How long would the boot take?
  - No known limitations in size and total global throughput for the repo
- Very low CPU usage on servers
- Happy with many clients per server
  - Thousands. But check their bw consumption vs the disk/net performance!
- WAN friendly (client+protocol+server)
  - Enable efficient remote POSIX-like direct data access through WAN
- WAN friendly (server clusters)
  - Can set up WAN-wide huge repositories by aggregating remote clusters
  - Or making them cooperate

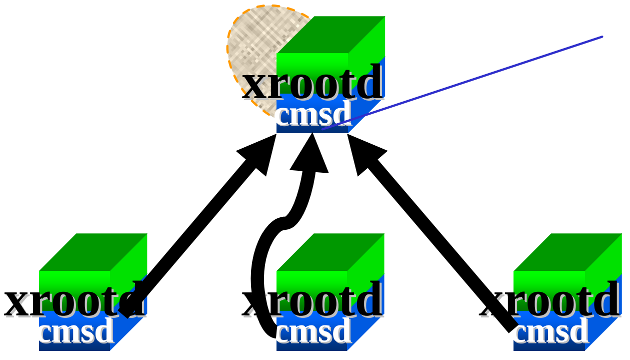
Basic working principle



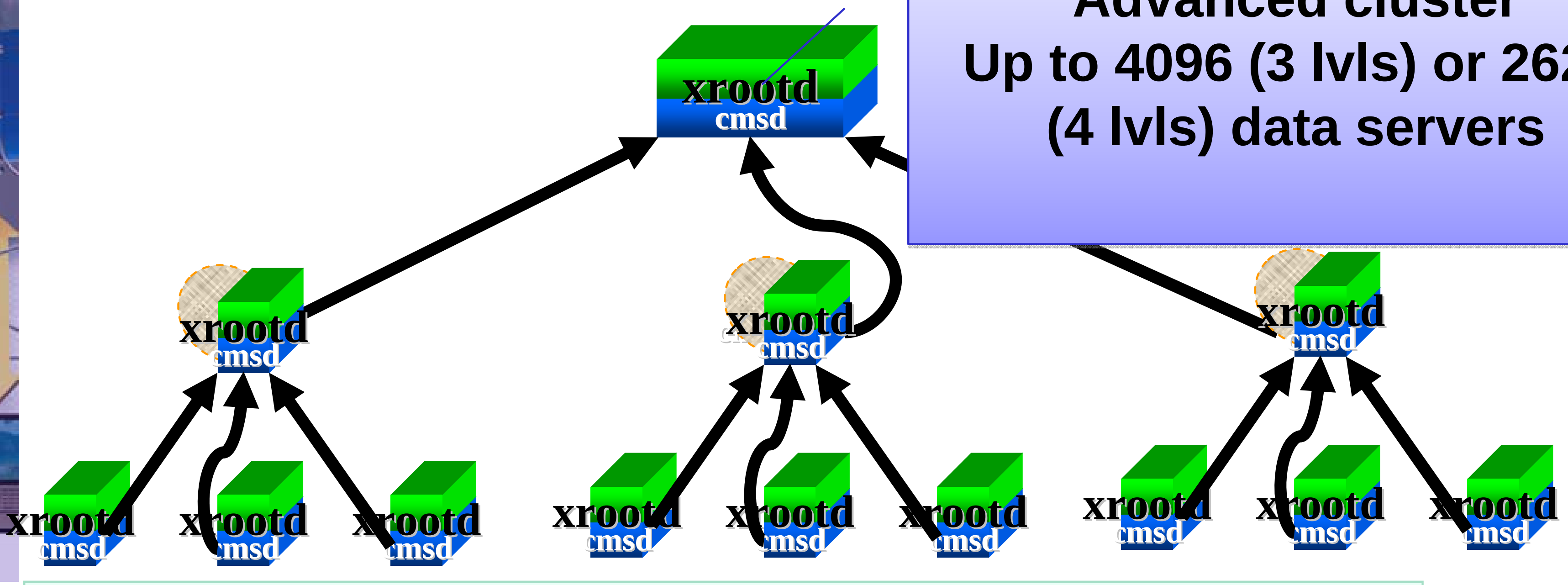


A small 2-level cluster. Can hold Up to 64 servers

Simple LAN clusters



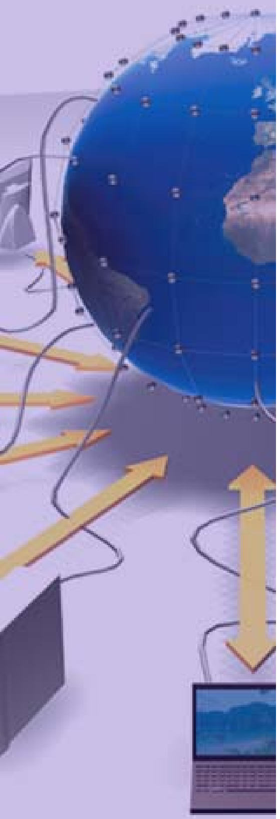
Simple cluster
Up to 64 data servers
1-2 mgr redirectors



Advanced cluster
Up to 4096 (3 lvls) or 262K
(4 lvls) data servers

Everything can have hot spares

Client and server



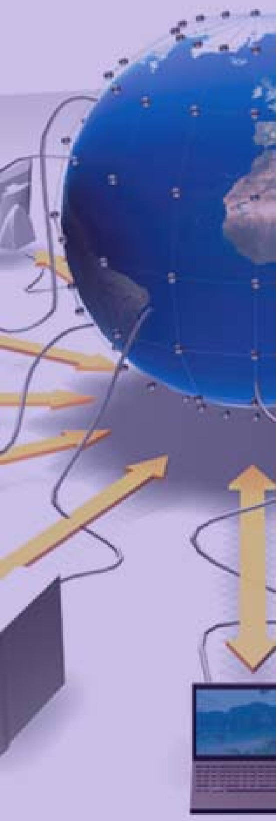
- Very carefully crafted, heavily multithreaded
  - Server side: lightweight, promote speed and scalability
    - High level of internal parallelism + stateless
    - Exploits OS features (e.g. async i/o, polling, selecting, sendfile)
    - Many many speed+scalability oriented features
    - Supports thousands of client connections per server
    - No interactions with complicated things to do simple tasks
    - Can easily be connected to MSS devices
  - Client: Handles the state of the communication
    - Reconstructs everything to present it as a simple interface
    - Fast data path
    - Network pipeline coordination + r/w latency hiding
    - Connection multiplexing (just 1 connection per couple process-server, no matter the number of file opens)
    - Intelligent server cluster crawling
- Server and client exploit multi core CPUs natively

Fault tolerance – what do we mean?

- Server side
  - If servers go down, the overall functionality can be fully preserved
    - Redundancy, MSS staging of replicas, …
    - “Can” means that weird deployments can give it up
      - E.g. storing in a DB the physical endpoint server address for each file. (saves ~1ms per file but gives up the fault tolerance)
- Client side (+protocol)
  - The client crawls the server metacluster looking for data
  - The application never notices errors
    - Totally transparent, until they become fatal
      - i.e. when it becomes really impossible to get to a working endpoint to resume the activity
  - Typical tests (try it!)
    - Disconnect/reconnect network cables while processing
    - Kill/restart any server

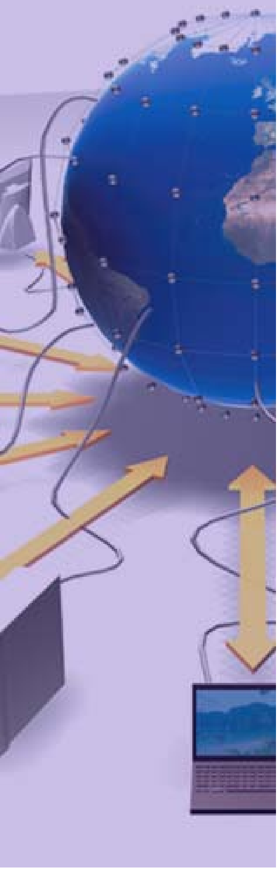
Available auth protocols

Courtesy of Gerardo Ganis (CERN PH-SFT)



- Password-based (pwd)
  - Either system or dedicated password file
    - User account not needed
- GSI (gsi)
  - Handle GSI proxy certificates
  - VOMS support should be inserted (Andreas, Gerri)
  - No need of Globus libraries (and super-fast!)
- Kerberos IV, V (krb4, krb5)
  - Ticket forwarding supported for krb5
- Security tokens (used by ALICE)
  - Emphasis on ease of setup and performance

A nice example: The “many” paradigm

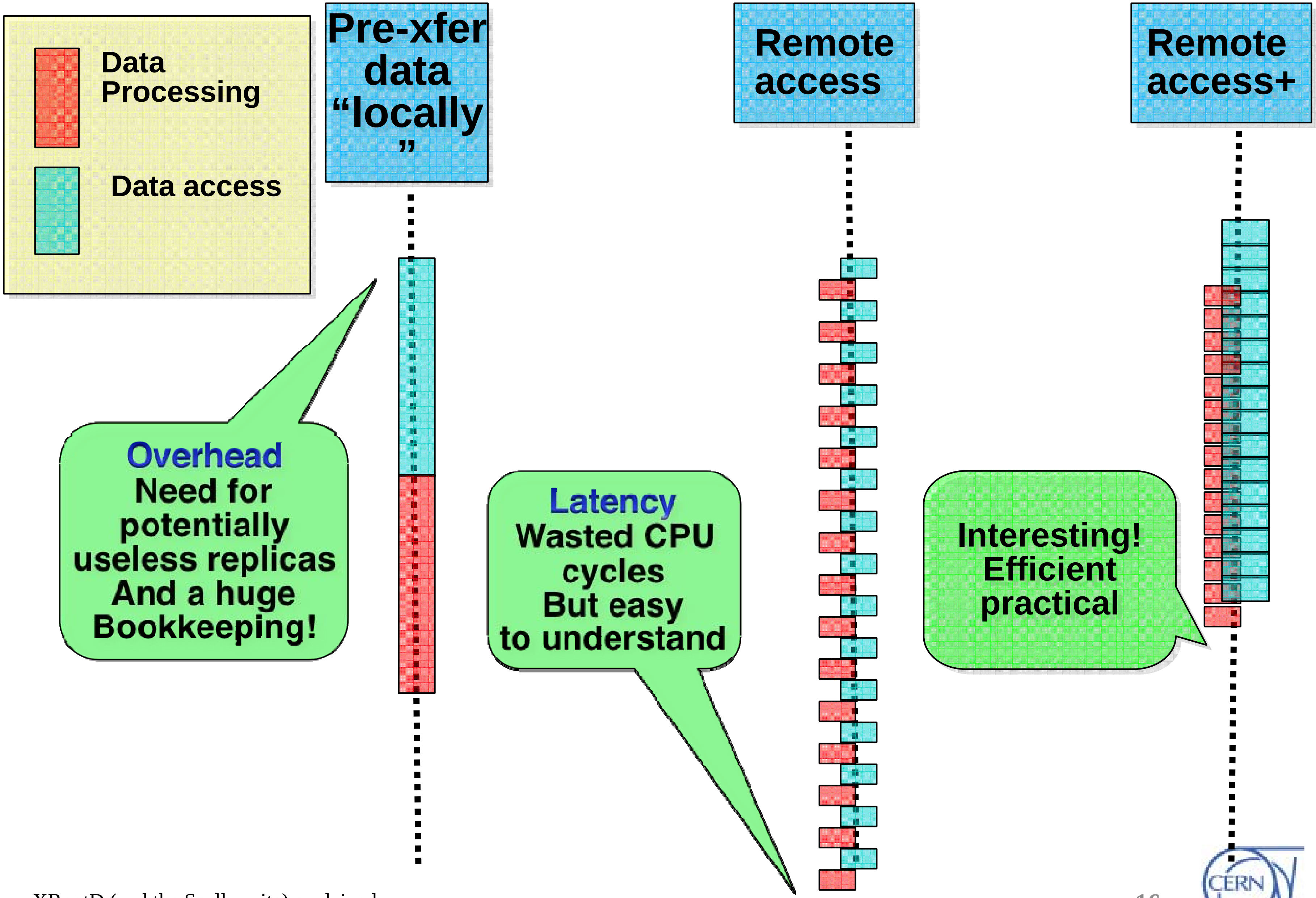


- Creating big clusters scales linearly
  - The throughput and the size, keeping latency very low
- We like the idea of disk-based cache
  - The bigger (and faster), the better
- So, why not to use the disk of every WN ?
  - In a dedicated farm
  - 500GB * 1000WN → 500TB
  - The additional cpu usage is anyway quite low
    - And no SAN-like bottlenecks, just pure internal bus + disk speed
- Can be used to set up a huge cache in front of a MSS
    - No need to buy a faster MSS, just lower the miss rate and higher the cache performance !
- Adopted at BNL for STAR (up to 6-7PB online)
    - See Pavel Jakl’s (excellent) thesis work
    - They also optimize MSS access to nearly double the staging performance
  - Quite similar to the PROOF approach to storage
    - Only storage. PROOF is very different for the computing part.

WAN direct access – Motivation

- WAN bw is growing with time
  - But r/w analysis apps typically do not exploit it, they just copy files around, useful and not
  - USING the WAN bw is just a little more technologically difficult
    - The technology can be encapsulated, and used transparently
  - The very high level of performance and robustness of direct data access cannot even be compared with the ones got from managing a universe of replicas
    - Example: Some parts of the ALICE computing model rely on direct WAN access. Not one glitch in production, since one year. Just power cuts.
- With XrdClient+XRootD
  - Efficient WAN read access needs to know in advance the chunks to read
    - Statistic-based (e.g. the embedded sequential read ahead)
    - Exact prefetching (The ROOT framework does it for ROOT files)
  - Efficient WAN write access… just works
    - VERY new feature, in beta testing phase as we speak

WAN direct access – hiding latency



🟥 Data Processing

🟦 Data access

Pre-xfer data “locally”

Remote access

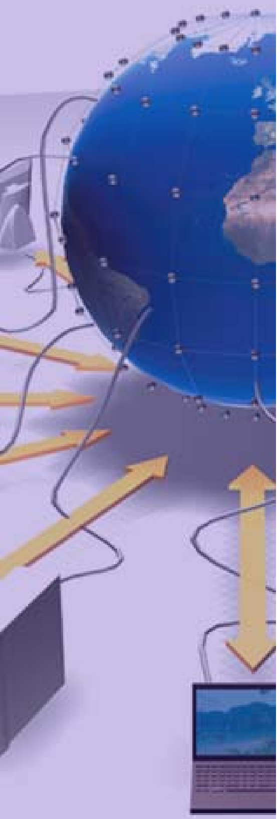
Remote access+

Overhead
Need for potentially useless replicas
And a huge Bookkeeping!

Latency
Wasted CPU cycles
But easy to understand

Interesting! Efficient practical

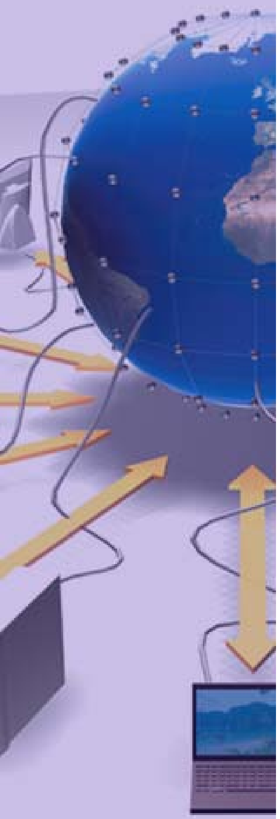
Cluster globalization



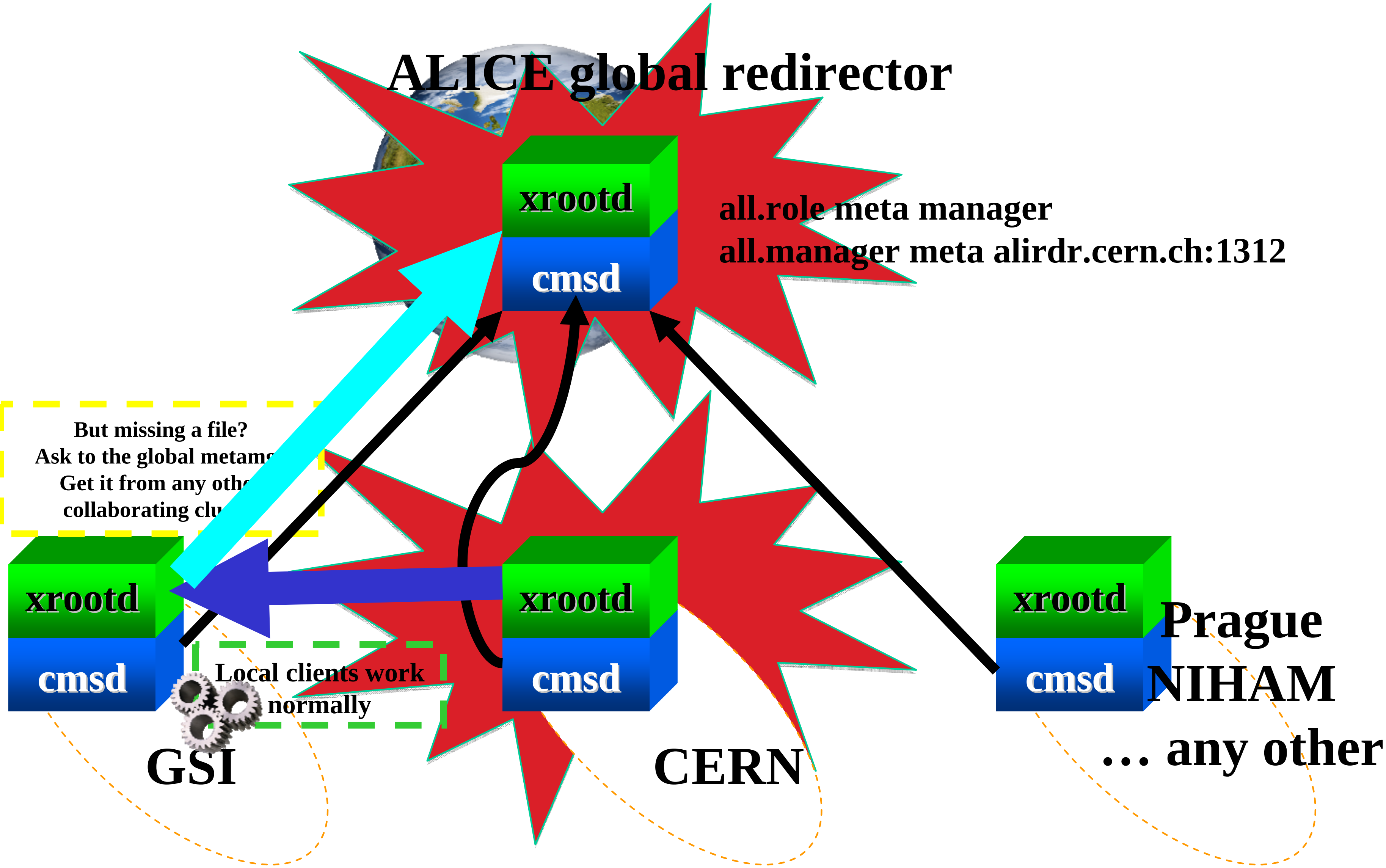
- Up to now, xrootd clusters could be populated
  - With xrdcp from an external machine
  - Writing to the backend store (e.g. CASTOR/DPM/HPSS etc.)
- E.g. FTD in ALICE now uses the first. It “works”…
    - Load and resources problems
    - All the external traffic of the site goes through one machine
      - Close to the dest cluster
- If a file is missing or lost
  - For disk and/or catalog screwup
  - Job failure
    - ... manual intervention needed
    - With 10^7 online files finding the source of a trouble can be VERY tricky

Virtual Mass Storage System – What’s that?

- Basic idea:
  - A request for a missing file comes at cluster X,
  - X assumes that the file ought to be there
    - And tries to get it from the collaborating clusters, from the fastest one
    - The MPS (MSS intf) layer can do that in a very robust way
- Note that X itself is part of the game
  - And it’s composed by many servers
- In practice
  - Each cluster considers the set of ALL the others like a very big online MSS
  - This is much easier than what it seems
    - Slowly Into production for ALICE in tier-2s
- NOTE: it is up to the computing model to decide where to send jobs (which read files)



The Virtual MSS in ALICE



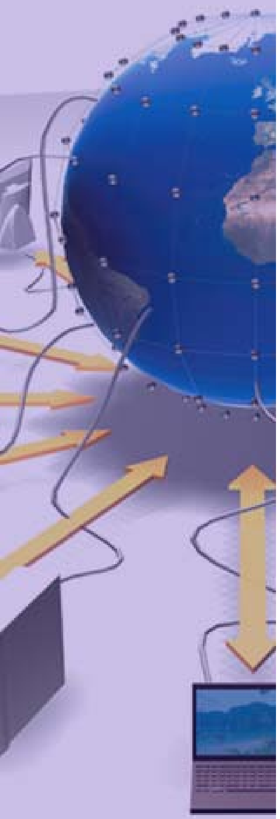
ALICE global redirector

```
all.role meta manager
all.manager meta alirdr.cern.ch:1312
```

But missing a file?
Ask to the global metam
Get it from any othe
collaborating clu

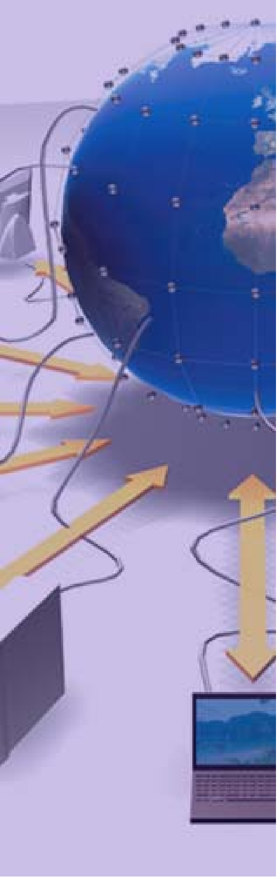
Local clients work normally

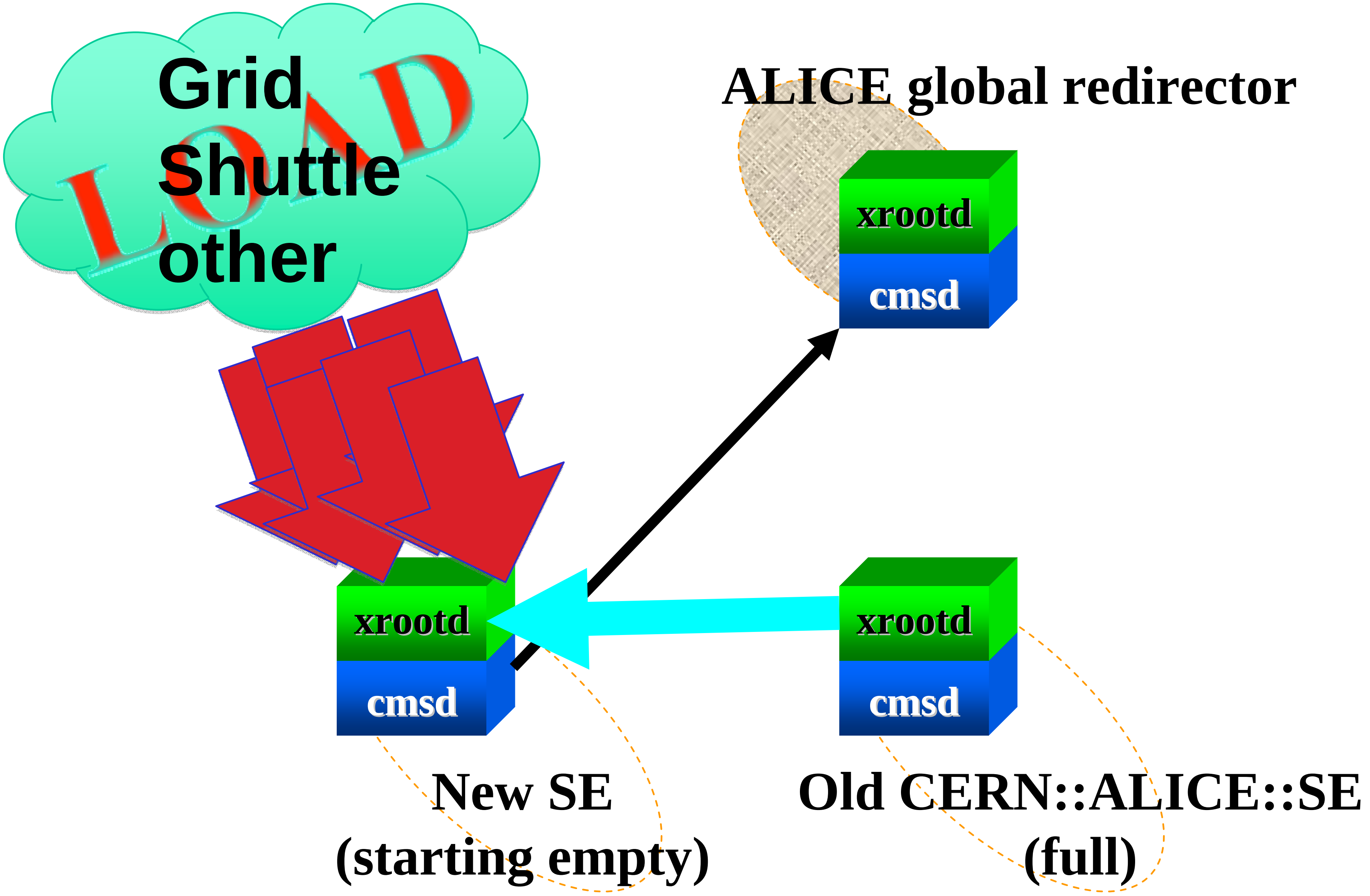
The ALICE::CERN::SE July trick



- All of the ALICE cond data is in ALICE:CERN::SE
  - 5 machines pure xrootd, and all the jobs access it, from everywhere
  - There was the need to refurbish that old cluster (2 yrs old) with completely new hw
  - VERY critical and VERY stable service. Stop it and every job stops.
- A particular way to use the same pieces of the vMSS
- In order to phase out an old SE
  - Moving its content to the new one
    - Can be many TBs
    - `rsync` cannot sync 3 servers into 5 or fix the organization of the files
- Advantages
  - Files are spread evenly → load balancing is effective
  - More used files are fetched typically first
  - No service downtime, the jobs did not stop
    - Server downtime of 10-20min
    - The client side fault tolerance made the jobs retry with no troubles

The ALICE::CERN::SE July trick





Grid
Shuttle
other

LOAD

ALICE global redirector

xrootd
cmsd

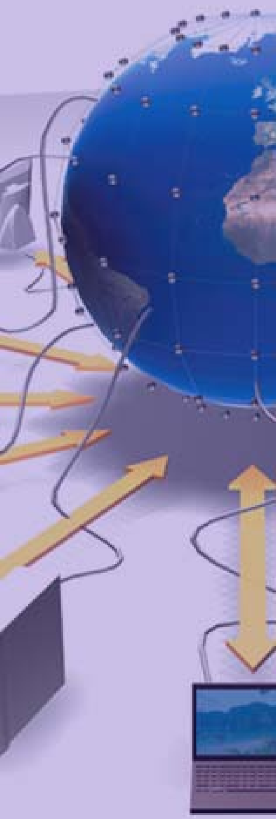
xrootd
cmsd

xrootd
cmsd

New SE
(starting empty)

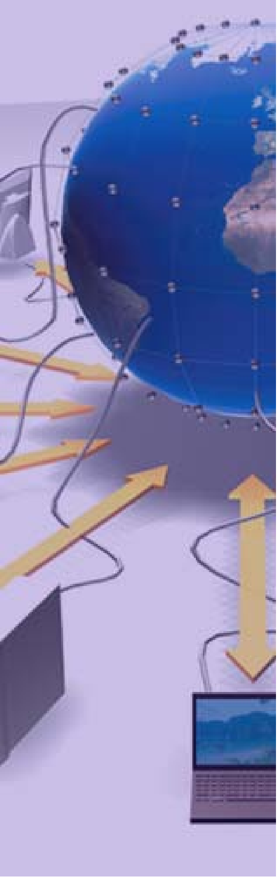
Old CERN::ALICE::SE
(full)

Virtual MSS



- The mechanism is there, fully “boxed”
  - The new setup does everything it’s needed
- A side effect:
  - Pointing an app to the “area” global redirector gives complete, load-balanced, low latency view of all the repository
  - An app using the “smart” WAN mode can just run
    - Probably now a full scale production/analysis won’t
      - But what about a small debug analysis on a laptop?
      - After all, HEP sometimes just copies everything, useful and not
    - I cannot say that in some years we will not have a more powerful WAN infrastructure
      - And using it to copy more useless data looks just ugly
      - If a web browser can do it, why not a HEP app? Looks just a little more difficult.
- Better if used with a clear design in mind

Scalla/Xrootd simple setup



- The xrd-installer setup has been refurbished to include all the discussed items
    - Originally developed by A.Peters and D.Feichtinger
- Usage: same of before
    - Download the installer script
    - `wget http://project-arda-dev.web.cern.ch/project-arda-dev/xrootd/tarballs/installbox/xrd-installer`
    - Run it
        - `xrd-installer –install –prefix <inst_pfx>`
- Then, there are 2 small files containing the parameters
    - For the token authorization library (if used)
    - For the rest (about 10 parameters)
        - Geeks can also use the internal xrootd config file template
            - » And have access to really everything
            - » Needed only for very particular things

Setup important parameters

VMSS_SOURCE: where this cluster tries to fetch files from, in the case they are absent.

LOCALPATHPFX: the prefix of the namespace which has to be made "facultative” (not needed by everybody)

LOCALROOT is the local (relative to the mounted disks) place where all the data is put/kept by the xrootd server.

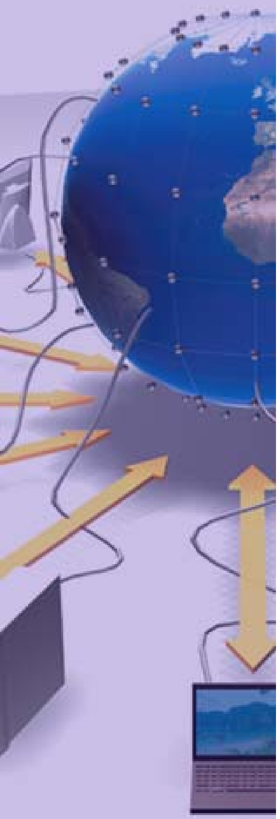
OSSCACHE: probably your server has more than one disk to use to store data. Here you list the mountpoints.

MANAGERHOST: the redirector’s name

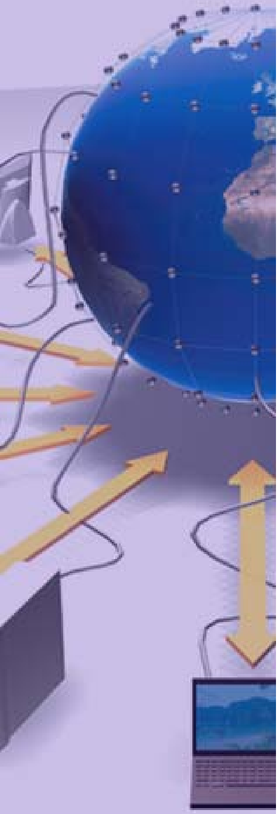
SERVERONREDIRECTOR: is this machine both a server and redirector?

OFSLIB: the authorization plugin library to be used in xrootd.

METAMGRHOST, METAMGRPORT: host and port number of the meta-manager (global redirector)



Setup: start and stop



- Check the status of the daemons

```
xrd.sh
```

- Check the status of the daemons and start the ones which are currently not running (and make sure they are checked every 5 mins). Also do the log rotation.

```
    xrd.sh -c
```

- Force a restart of all daemons

```
xrd.sh -f
```

- Stop all daemons

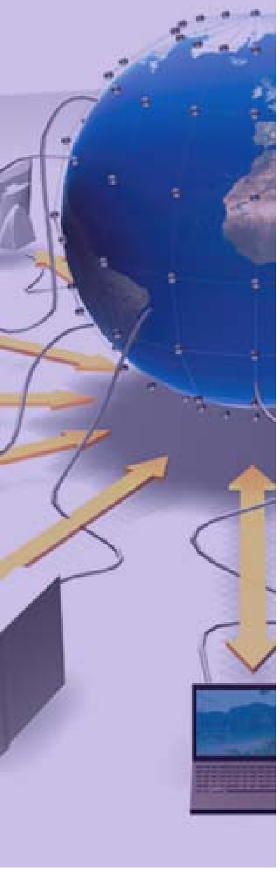
```
xrd.sh -k
```

Conclusion

CERN IT Department

- We spoke about
  - Base features and philosophy
    - Maximum performance and robustness with the least cost
  - Basic working principle
  - WAN access and WAN-wide clusters
    - But the WAN could be a generic ADSL (which has quite a high latency)
    - We like the idea of being able to access data efficiently and without pain, no matter from where
    - And this is quite difficult to achieve with typical distributed FSs
  - WAN-wide clusters and inter-cluster cooperation
  - Setup
- A lot of work going on
  - Assistance, integration, debug, deployments
  - New fixes/features
  - Documentation to update on the website (!)

Acknowledgements



- Old and new software Collaborators
  - Andy Hanushevsky, Fabrizio Furano (client-side), Alvise Dorigo
  - Root: Fons Rademakers, Gerri Ganis (security), Bertrand Bellenot (MS Windows porting)
  - Derek Feichtinger, Andreas Peters, Guenter Kickinger
  - STAR/BNL: Pavel Jakl, Jerome Lauret
  - Cornell: Gregory Sharp
  - SLAC: Jacek Becla, Tofigh Azemoon, Wilko Kroeger, Bill Weeks
  - Peter Elmer
- Operational collaborators
  - BNL, CERN, CNAF, FZK, INFN, IN2P3, RAL, SLAC