XRootD

Presentation at NSDF

February 7, 2022

Andrew Hanushevsky, SLAC
http://xrootd.org

CERN SLAC UCSD

Brief history of the last ~20 years

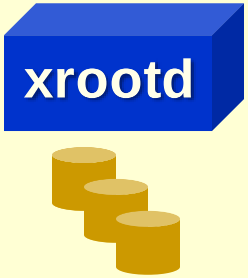
- 2001 – BaBar decides to use root framework vs Objectivity
- 2002 Collaboration with INFN, Padova & SLAC created
  - Design & develop a network-based HP data access system
    - In the days of limited network b/w and high expense
- 2003 – First deployment of **XRootD** system at SLAC
- 2013 – Wide deployment across most of HEP
  - Protocol also re-implemented (Java) in dCache
- 2022 – **XRootD** is now a popular internal framework
  - Supports http, https, and xroots as well as xroot protocol
  - Third party software projects use it; leading to the moniker
    - "**XRootD** Inside!"

Todays's XRootD Project

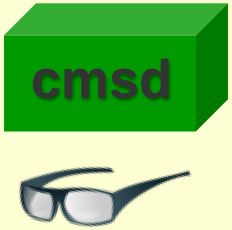
- A structured Open Source community supported project to provide a framework for clustering distributed storage services available via github, EPEL, & OSG
  - The project also supplies the fundamentals
    - A packaged storage service that meets many needs
      - But one that is also highly customizable

What Is XRootD?

- A system for scalable cluster data access



Data Access

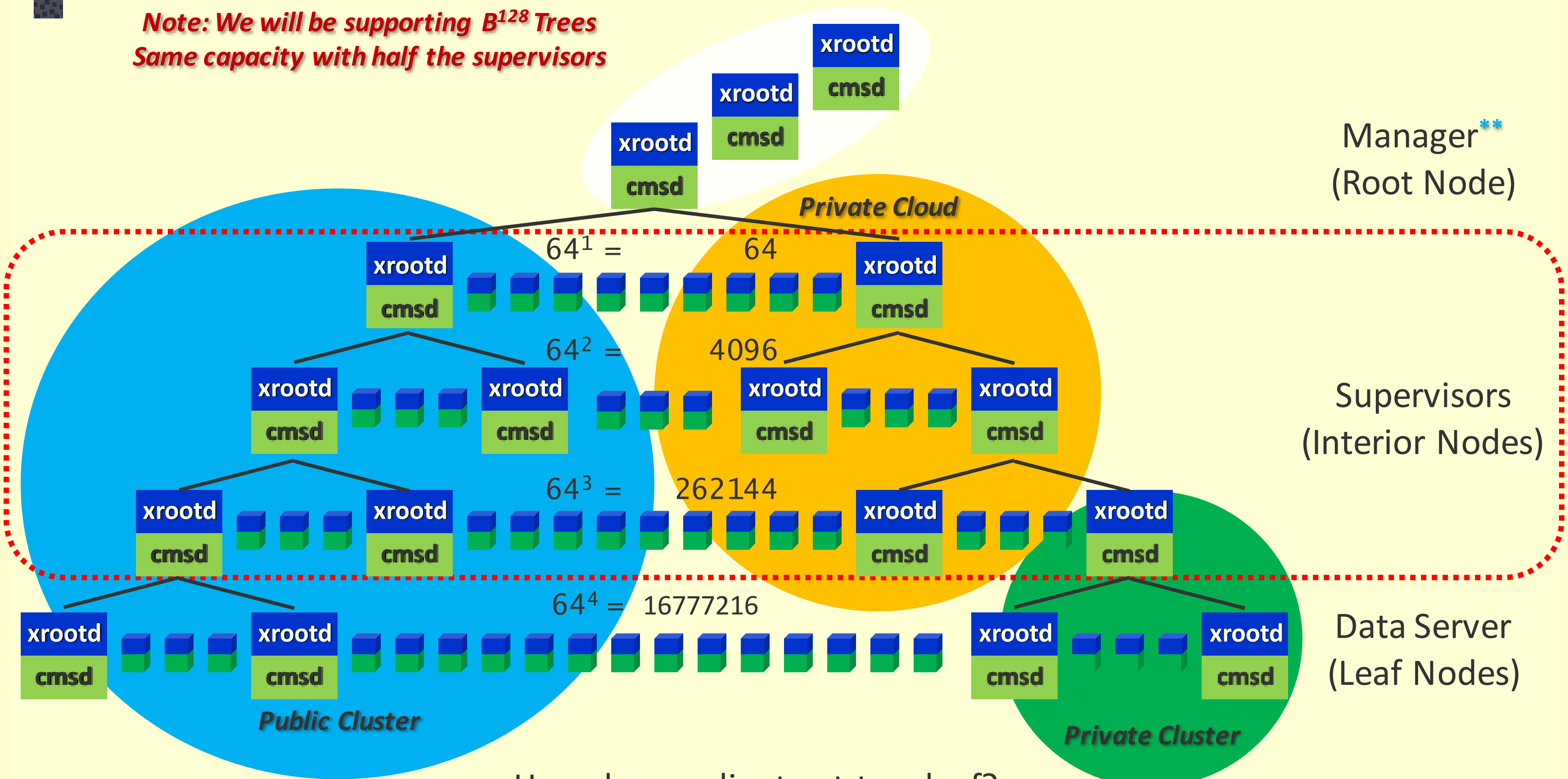


Data Clustering

- Not a file system & not *just* for file systems
- If you can write a plug-in you can cluster it
  - E.G. Used by LSST Qserv for clustered mySQL
- Hang tight for the next 62 slides!

Clustering Using B^{64} Trees

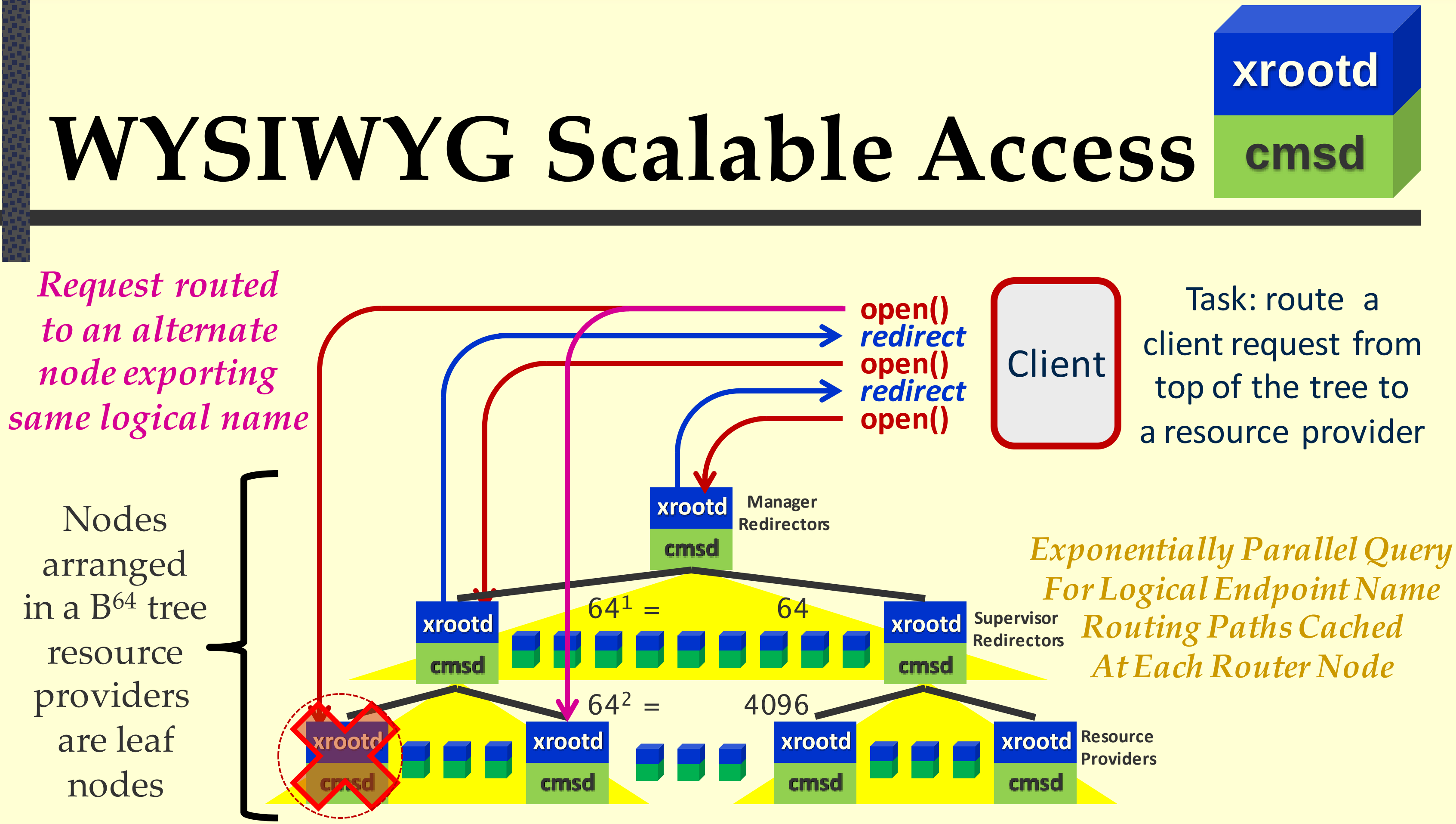
Note: We will be supporting B^{128} Trees
Same capacity with half the supervisors



How does a client get to a leaf?

** *Managers are also called redirectors but in practice any node is able to redirect*

WYSIWYG Scalable Access



Request routing is very different from traditional data management models
This implements a structured network of request routers (i.e. redirectors)
Capable of automatically recovering from adverse conditions
Much like internet routing it essentially implements an NDN

Applied Clustering

- XRootD clustering has many uses
  - Creating a uniform name space
    - Even though the name space is distributed
  - Load balancing & scaling
    - In situations where all servers are the "same"
      - Serving data from distributed file systems (e.g. Lustre)
      - Proxy servers (inherently identical)
      - Caching servers (inherently fungible, e.g.Xcache)
  - Reliability & recoverability
    - When mirror copies exist across sites

Deploying Clusters

- Things to keep in mind
  - Every **cmsd** has a companion **xrootd**
    - Both should be on the same h/w box
  - 64 (soon 128) servers per **cmsd**
    - If more than 64 servers use supervisor nodes
      - #Sup = upper(log64(#servers + upper(log64(#servers)))
      - Add one or two extras for enhanced reliability
  - Manager & Supervisor nodes on separate h/w
    - Using same node reduces reliability

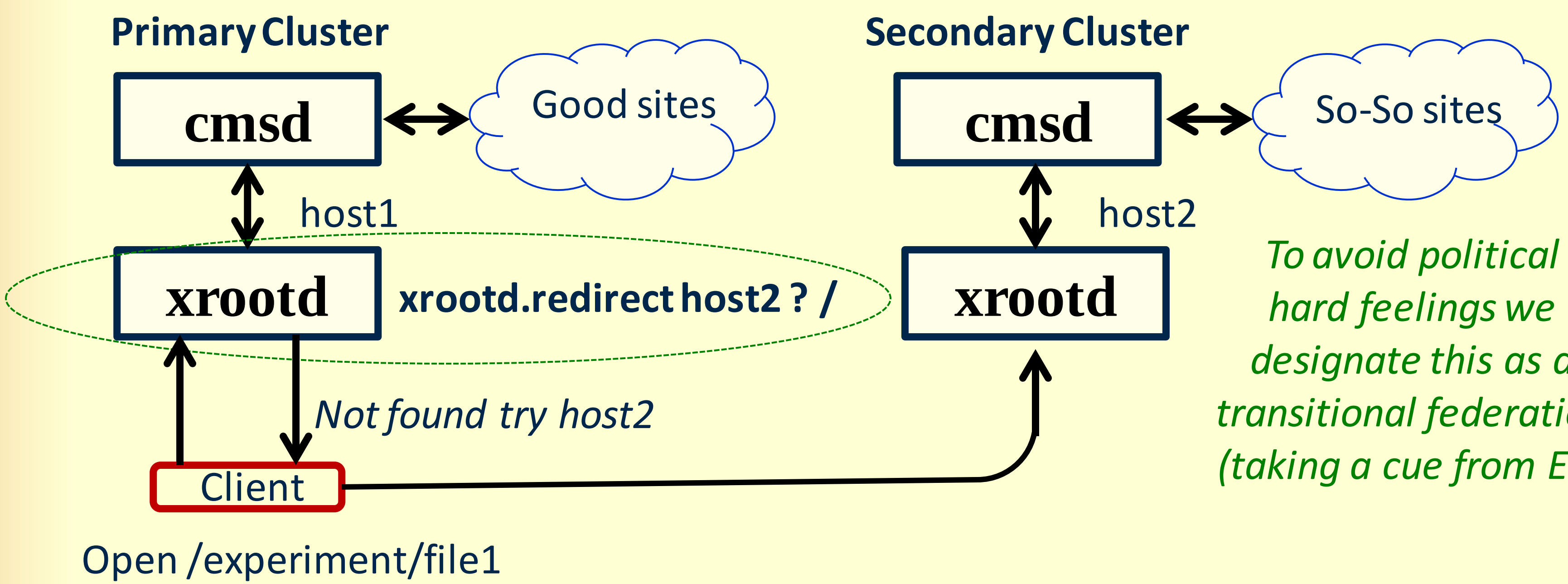
SLAC NATIONAL ACCELERATOR LABORATORY

LAN vs WAN Clusters

- LAN based clusters are reliable
  - You should not have any problems
- WAN based clusters are problematic
  - You may have little control over remote sites
  - What we have learned
    - Only accept reliable and well connected sites
      - Relegate problematic sites to secondary selection
        - Only used if you can't find a primary resource
    - Otherwise, you will be definitely disappointed

WAN Secondary Selection

Manager nodes (a.k.a redirector)



To avoid political hard feelings we designate this as a transitional federation (taking a cue from EU)

Deploying Manager Nodes I

- Many sites use at least two
  - Can be load balanced or simply a backup
  - Load balanced managers now preferred
    - Allows for much larger name spaces
    - **all.manager all** ….
      - https://xrootd.slac.stanford.edu/doc/dev54/cms_config.htm#_Toc53611061
    - Also read all vs. any options (default is any)
      - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611062

Deploying Manager Nodes II

- Don’t bother with DNS load balancing
  - It really doesn’t work all that well
  - Plus the **XRootD** client ignores it so it’s useless
  - Using HA devices adds far more complexity
    - Not worth the effort as **cmsd** does s/w HA anyway

Default Load Balancing Servers

- By default manager selects servers
  - Uses a augmented round robin algorithm
    - Within the set of servers that have the file o/w
    - Within set of servers that have enough space
      - Tuning knobs: **cms.space** and **cms.sched linger**
        - Defines what enough means
          - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611078
          - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611076
- This works reasonably well
  - For non-stressed systems

Load based Balancing Servers

- Can enable load-based selection
  - Must supply a load reporter (script or plug-in)
    - See **cms.perf** directive
      - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611073
      - We already have two basic scripts in utils directory
        - Bash: cms_monPerf and Perl: XrdOlbMonPerf
  - Load is computed using a config formula
    - Percentage of each of cpu, io, memory, paging, runq
    - That yields a value 0 to 100.
    - See **cms.sched** directive
      - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611076

Load based server selection I

- Manager selects least loaded server
  - Within set of servers that have the file
    - Definition of "least" controlled by fuzzing
      - See cms.sched fuzz
  - Within set of servers that have enough space
    - Tuning knob: **cms.space**
      - Defines what enough means
      - **cms.sched linger** is not applied

Load based server selection II

- This works well in all situations
  - Load periodically reported
    - Default is every 10 minutes
    - Configurable via **cms.ping** directive
      - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611094
  - Load is also asynchronously reported
    - If load delta of previous > **cms.sched fuzz**
      - The default fuzz is 20%
    - Requires script/plug-in supply data more often
      - I.e. more often than periodic reporting interval

DFS Clusters

- These are clusters of
  - Servers who all export the same DFS
    - Distributes File System
  - Proxy servers
  - Proxy servers all with a cache
    - Xcache
- Tuning knob is cms.dfs directive
  - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611070

Subordinate Clusters

- These are local cluster of servers
  - Need to be part of another local cluster
- Subordinate resources are independent
  - This allows mixing cluster types
    - E.G. A DFS cluster can be a member of a non-DFS cluster (but not the other way around)
- Defined by the **cms.subcluster** directive
  - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611099

Federated Clusters

- Cluster of administratively independent clusters anywhere in the world
  - Headed by a Meta-Manager
    - Managers of each site cluster subscribe to the Meta-Manager (the federation head node)
  - Examples:
    - CMS AAA
    - OSG Xcache CDN
      - https://display.opensciencegrid.org/

SLAC NATIONAL ACCELERATOR LABORATORY

Cluster deployment practices

- How you deploy depends on what it is
  - Local vs. regional vs. US vs. world cluster
  - Data servers vs. Proxies vs. Caching proxies
  - Native vs. containers
    - If containers the management scheme (e.g. k8s)
- Considerations discussed in references
  - Under each type of server
- OSG can be of immense help here

SLAC NATIONAL ACCELERATOR LABORATORY

What about data server nodes?

- The easiest of all to deploy
  - Fairly straightforward like an NFS box
  - Using real HD's (JBOD or otherwise)?
    - Want QOS or grow and shrink the space?
      - See the **oss.space** directive
        - https://xrootd.slac.stanford.edu/doc/dev54/ofs_config.htm#_Toc89982406
    - Using tape?
      - Want automatic staging & migration?
        - See File Residency Manager Reference
          - https://xrootd.slac.stanford.edu/doc/dev50/frm_config.htm

Networking Considerations

- IPv6 and IPv4 fully supported
- However, there still is your topology
  - Firewalls
    - You may need to deploy proxy servers
  - Private vs. public networks
    - You may need to specify the relationship mix
      - Usually due to non-standard deployments
      - See **xrd.network** directive
        - https://xrootd.slac.stanford.edu/doc/dev53/xrd_config.htm#_network

Security Considerations

- This is the hardest part, as always
  - Decide on authentication
    - X509 and Kerberos are most popular today
      - Can have more than one available or none at all
        - If using JWT's (e.g. SciTokens)
  - Decide on authorization
    - Built-in identity based authorization popular
    - JWT's are fast moving up the list
      - SciTokens fully supported for xroots and https
        - But it's still a moving target
  - https://xrootd.slac.stanford.edu/doc/dev54/sec_config.htm

SLAC NATIONAL ACCELERATOR LABORATORY

Operational Considerations I

- Monitoring is your friend
  - **XRootD** has robust full-featured monitoring
    - However, you must supply collector & visualizer
      - See OSG for collector and recommended visualizer
  - A number of directives apply
    - https://xrootd.slac.stanford.edu/doc/dev54/xrd_config.htm#_Toc88513955
    - https://xrootd.slac.stanford.edu/doc/dev54/xrd_config.htm#_Toc88513988
  - What's missing?
    - Alerts, we never could get agreement on it
      - Many sites drive it via monitoring aberrations

SLAC NATIONAL ACCELERATOR LABORATORY

Operational Considerations II

- One config file rules the world!
  - Try very hard to have a single config file
    - One file for all types of nodes in a site helps!
      - Eliminates divergence promotes consistency
      - The config file has if/else/fi features to make it possible
        - https://xrootd.slac.stanford.edu/doc/dev49/Syntax_config.htm
    - The **cconfig** command is your helper
      - Displays actual config file in server's context
        - Host, instance, and whether cmsd or xrootd
        - Can be run from anywhere

Operational Considerations III

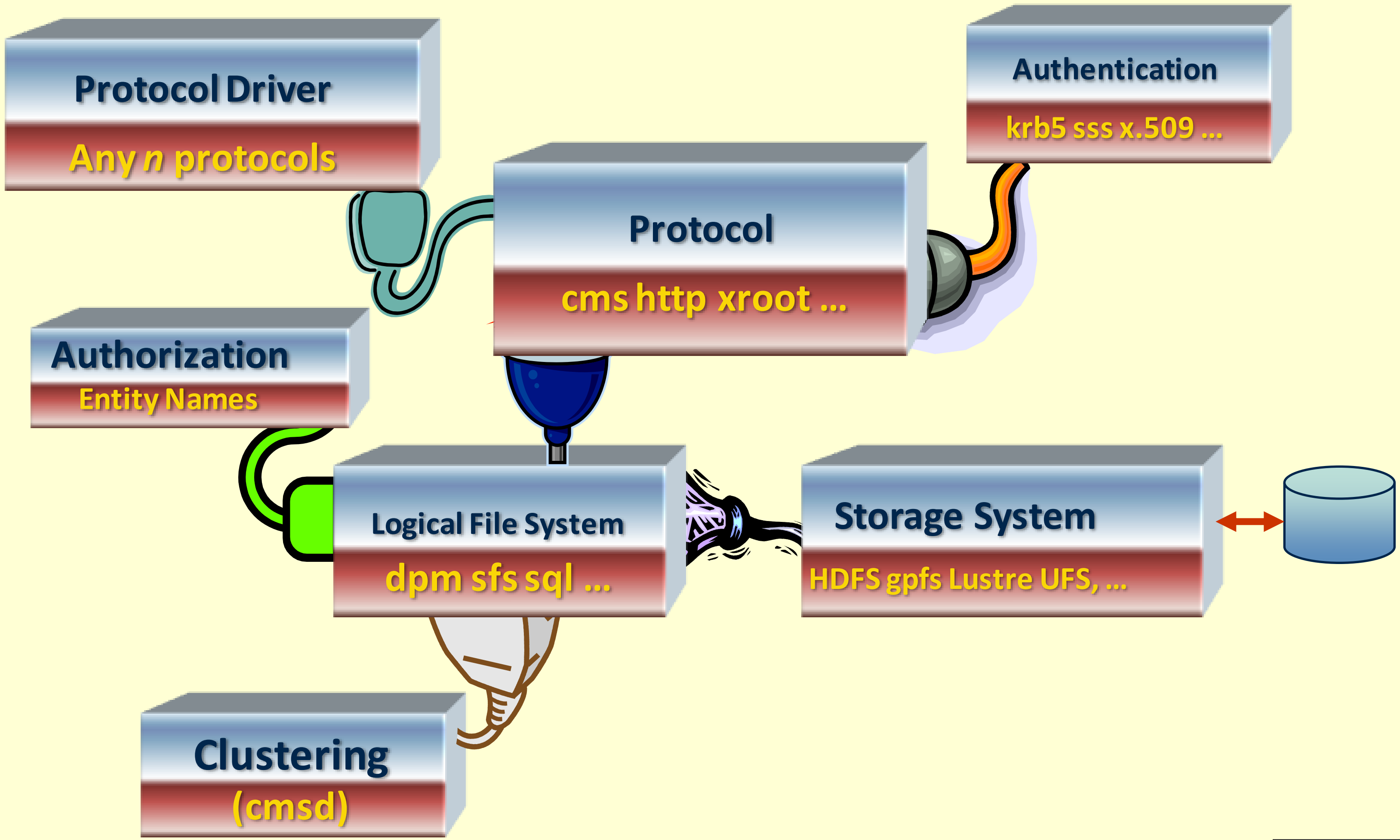
- Consider enabling remote debugging
  - Very useful for large deployments
  - Provides *standardized* view of server internals
    - Config file, core files, log files, process info
      - Regardless of server layout you always get same view
    - Can add additional views or restrict native views
  - Allowed for authenticated authorized users
  - Can only be used against a running server
  - https://xrootd.slac.stanford.edu/doc/dev54/xrd_config.htm#_diglib

SLAC NATIONAL ACCELERATOR LABORATORY

Transition to developers

- Next set of slides is a deep dive
  - Architecture
  - Request/response flow
  - What to be careful about
- Your chance to ditch
  - If you don't want an internal deep dive

XRootD Plug-in Architecture



Why Plug-ins?

- Makes it much easier to
  - Adapt, customize, add new features
- Any cons?
  - Need to know available plug-in points
    - These are documented but not in one spot
      - Described under the relevant directive
        - Usually xxx**lib** (e.g. xrootd.fslib)
      - However, we did make it a bit easier….

The plug-in points

- A lot and more plug-ins than points!
- Get a list using **xrdpinls** command

```
>xrdpinls
Required >=  5.0  @logging
Optional >=  5.0  bwm.policy
Required >=  5.0  cms.perf
Required >=  5.0  cms.vnid
Optional >=  5.0  gsi-authzfun
Optional >=  5.0  gsi-gmapfun
Optional >=  5.0  gsi-vomsfun
Required >=  4.8  http.exthandler
Required >=  4.0  http.secxtractor
Required >=  5.0  ofs.authlib
Required >=  5.0  ofs.ckslib
Required >=  5.0  ofs.cmslib
Required >=  5.0  ofs.ctllib
Required >=  5.0  ofs.osslib
Required >=  5.0  ofs.preplib
Required >=  5.0  ofs.xattrlib
```

```
Optional >=  5.0  oss.namelib
Required >=  5.0  oss.statlib
Optional >=  5.0  pfc.decisionlib
Required >=  5.0  pss.cachelib
Required >=  5.0  pss.ccmlib
Required >=  5.0  sec.protocol
Required >=  5.0  sec.protocol-gsi
Required >=  5.0  sec.protocol-krb5
Required >=  5.0  sec.protocol-pwd
Required >=  5.0  sec.protocol-sss
Required >=  5.0  sec.protocol-unix
Untested >=  5.0  xrd.protocol
Required >=  5.0  xrdcl.monitor
Required >=  5.0  xrdcl.plugin
Required >=  5.0  xrootd.fslib
Required >=  5.0  xrootd.seclib
```

32 but actual 27

BTW are missing a few due to forgetfulness. Will be corrected!

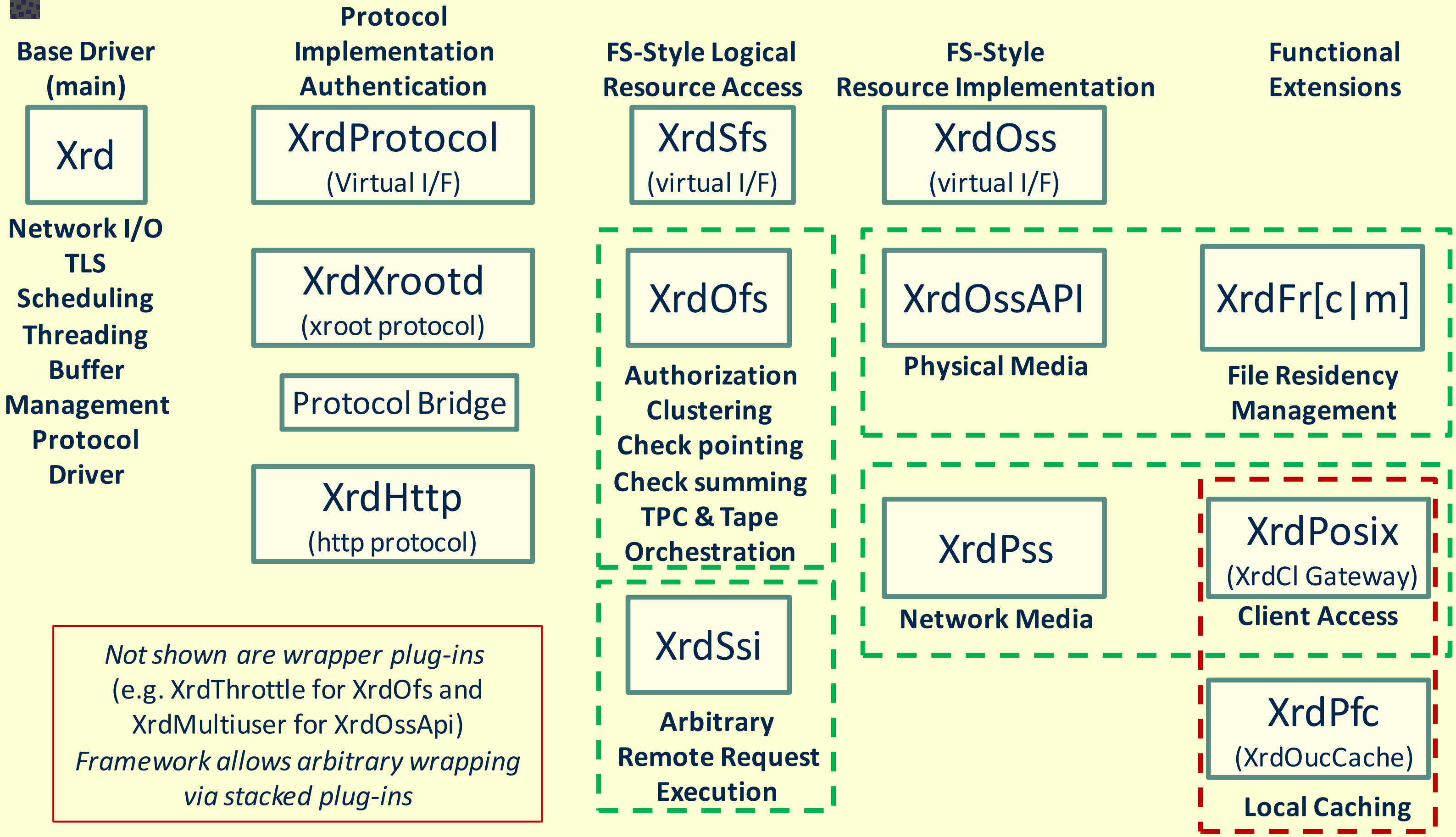
Plug-in points explained I

@logging	Log message handler (server – cli option)
bwm.policy	Network bandwidth management policy
cms.perf	Performance monitor for cmsd (not script based)
cms.vnid	Virtual network identifier generator for cms
gsi-authzfun	Specialized gsi authz function
gsi-gmapfun	Specialized gsi gridmap function
gsi-vomsfun	Specialized gsi VOMS function
http.exthandler	HTTP post processing handler
http.secxtractor	HTTPS security information extraction
ofs.authlib	Authorization plug-in
ofs.ckslib	Checksum plug-in (individual and manager)
ofs.cmslib	Cluster management service client plug-in
ofs.ctllib	Specialized file system control plug-in
ofs.osslib	Storage system plug-in
ofs.preplib	Prepare request plug-in

Plug-ins points explained II

ofs.xattrlib	Extended attribute handler plug-in
oss.namelib	Name mapping plug-in
oss.statlib	Functional stat() plug-in
pfc.decisionlib	Cache purging decision plug-in
pss.cachelib	Cache implementation plug-in
pss.ccmlib	Cache context management plug-in
sec.protocol	Authentication protocol plug-in (overloaded)
xrd.protocol	Communications protocol plug-in (overloaded)
xrdcl.monitor	Client-side action monitor plug-in
xrdcl.plugin	Client-side API implementation plug-in
xrootd.fslib	File system plug-in
xrootd.seclib	Security manager plug-in

Architectural Plug-in Interplay



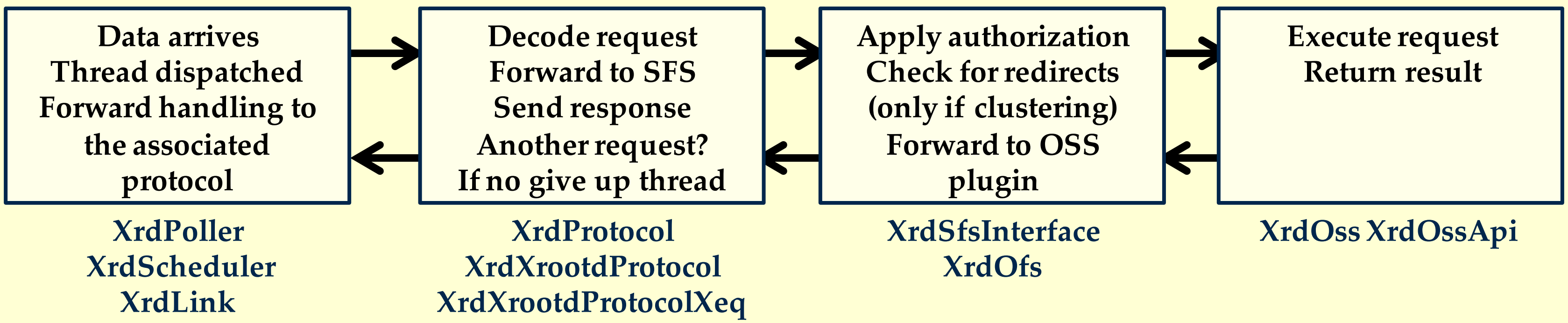
Handshake used to determine protocol to be used

It starts with a client handshake

- Upon success client sends info request
  - Server returns capabilities and security reqs
    - Client configures connection for server capabilities
      - This is when TLS & request signing are established
        - The connection *may* convert to using TLS here
  - Client issues login request
    - The server may then ask for authentication
      - This is a negotiable process
        - Server supplies list of supported protocols
        - Client needs to eventually pick one that works
  - Upon success client can start issuing requests

SLAC NATIONAL ACCELERATOR LABORATORY

Typical request/response flow



This is run to completion semantics and is the most cost-effective way of handling large numbers of clients; though it is thread intensive.

However, exceptions are allowed for certain long running requests.

That looks simple enough!

- Be careful, many requests are not simple
  - Verify request signature if signing enabled
  - Does request perform I/O (explicit or implicit)?
    - Eligible for asynchronous execution?
      - Segment request and run segments in parallel
    - Does request require data checksums?
      - Generate or verify checksums on the fly
    - Should file be check pointed prior to modification?
      - If so, rollback changes upon failure
- All of these are run-time actions

Can even be complicated in SFS

- Certain requests are "call back" eligible
  - The logical fs uses for long running tasks
    - E.G. checksums
  - Typical SFS plug-in scenario
    - Start operation on new thread
    - Return result as "operation started"
    - Protocol tells client to wait for a resp call back
    - When operation completes SFS issues an async call back to the protocol with the result
    - Result is then sent to the client in a call back

Client can issue additional requests while waiting for a request callback!

SLAC NATIONAL ACCELERATOR LABORATORY

More on callbacks

- Eligible requests
  - close, locate, open, prepare, stat, statx, truncate
  - Query for Qopaquf, Qopaqug, Qvisa, Qxattr
  - Pointer to callback object passed via error obj
    - Callback performs all synchronization
      - Avoids sending result before callback response sent
  - Typically used to accommodate tape systems
    - The oss plug-in can ask for an async callback too
      - Done by returning -EINPROGRESS on file open
        - Done for file staging from tape

The I/O architecture I

- Three types of read requests
  - read (async or sync)
    - This the one most used
  - readv (only sync)
    - Used to aggregate many small reads
      - Root file applications use this most often
  - pgread (async or sync)
    - Provides data checksums for transport integrity
      - Used by Xcache and xrdcp

The I/O architecture II

- Three types of write requests
  - write (async or sync)
    - This the one most used
  - writev (only sync)
    - Used to aggregate many small writes
      - Practically no one uses this so far
  - pgwrite (async or sync)
    - Provides data checksums for transport integrity
      - Used by xrdcp

SLAC NATIONAL ACCELERATOR LABORATORY

Standard read & write (sync)

- Reads and writes of data from/to socket
  - By default uses up to a 2 MB buffer
    - That means data is segmented in 2MB units
  - Can use secret option for any size you want
    - Secret because sites would misuse this option
    - Practical reasons for 2MB default limit
  - Buffer allocated using serpentine algorithm
    - Minimizes reallocations
    - NUMA friendly

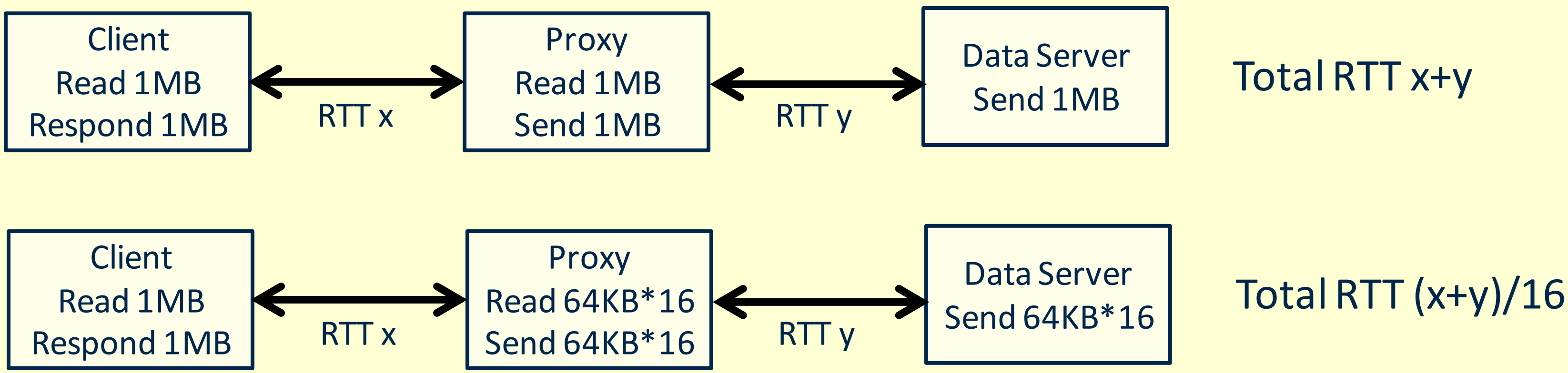
Standard read & write (async)

- Reads and writes of data from/to socket
  - By default uses 64KB buffers
    - That means data is segmented in 64KB units
  - Can set segment size to arbitrary length
    - 64KB used is to avoid store/forward latency
  - Train algorithm used to schedule buffers
    - Default is 8 cars but can configure it
      - See xrootd.async directive
        - https://xrootd.slac.stanford.edu/doc/dev53/xrd_config.htm#_Toc60181783

SLAC NATIONAL ACCELERATOR LABORATORY

Why 64K async read size

- Store/Forward effect in proxy servers
  - This also includes Xcache



- Chunking a read keeps the pipe full
  - Almost streaming but at a lower CPU cost
    - Aggregate performance can be achieved

Why a default of 8 buffers

- The train of 8 based US consideration
  - Minimize latency between east & west coasts
    - Works for the US
    - Not ideal for international links
      - Likely 2x increase in parallel buffer usage
        - But we have not got any complaints
- Async I/O only used for network devices

Standard Read optimization

- A read can also supply a pre-read list
  - Vector of (file_handle, length, offset)
    - Data to make ready for a subsequent read
      - I.E. data will be in memory for the next read
    - Note that data can come from multiple files
    - Vector is limited to 1024 items
  - No one uses this so far
    - Which is good because it has issues
      - Historical artifacts that should be corrected
        - Then we can add it to xrdcp

Vector reads and writes (sync)

- Application supplies a vector
  - (file_handle, length, offset)
    - Allows read/writes from/to multiple files
      - No one uses this feature as far as we know
    - Maximum item length is 2MB-16
      - Why -16? Results are framed as they can be unordered
    - Maximum vector length is 1024
- Only useful for certain applications
  - Xcache never uses it because all reads are big
    - It unrolls vector reads to page size units

SLAC NATIONAL ACCELERATOR LABORATORY

Why no async for vector reads

- Trade off between read size & latency
  - Typically we need at least 64KB of data
    - Less and overhead may swamp latency
- Implementation simplicity
  - Async I/O in a multi-file request is hard
    - Given that most reads are small we ditched it

Page read/write

- These are page aligned reads/writes
  - 4K pages on 4K boundaries
    - Does allow misalignment for 1st page
  - Each page is check summed using crc32c
    - crc32c is hardware assisted and really fast
  - Client/server perform on-the-fly correction
    - Reads: client rereads pages in error
    - Writes: server supplies pages in error to rewrite

Why page read/write

- Transmission errors do occur
  - Some not caught by the TCP 16 bit checksum
    - Reports of errors on international links
      - Typically during high usage periods
  - Avoids retransmission of large files (> 10GB)
    - When only a few bits are corrupted
  - Avoids having sticky errors in Xcache
    - A serious concern in a long-lived page cache
- Page read/write correct data in 4K units
  - Good size for crc32c

Page read/write sync vs. async

- Checksum processing restricts I/O size
  - Sync: 2,093,056 max bytes per I/O seg
    - Accounts for checksum overhead
      - Data + checksums ~= 2 MB (max default buffer size)
        - 2093056/4096 = 511
        - 511*4+2093056 = 2095100
          - 52 bytes shy of 2MB
  - Async: 64K per I/O segment
    - Sweet spot to minimize latency
  - Values cannot be adjusted

Final Notes on Async I/O

- Async only enabled for networked devices
  - Linux async I/O useless for locally attached disk
    - Implemented at user level via threads
- May change with new io_uring interface
  - Available since Linux Kernel version 5.1
    - Unfortunately, Red Hat has yet to get to that version
      - RH 8.5 (the latest release) uses 4.18
- But seems to be a very long way off

The network oriented features

- XRootD was developed for networks
  - The design goals were
    - Minimize bandwidth usage
      - Don't send unnecessary data
    - Maximize bandwidth utilization
      - Optimally use what you have to the fullest extent
    - Work around network & server failures
      - Automatic recovery whenever possible (usually can)
    - Be flexible
      - Adapt to the ever changing network configurations
  - Let's see what we did

Network bandwidth usage I

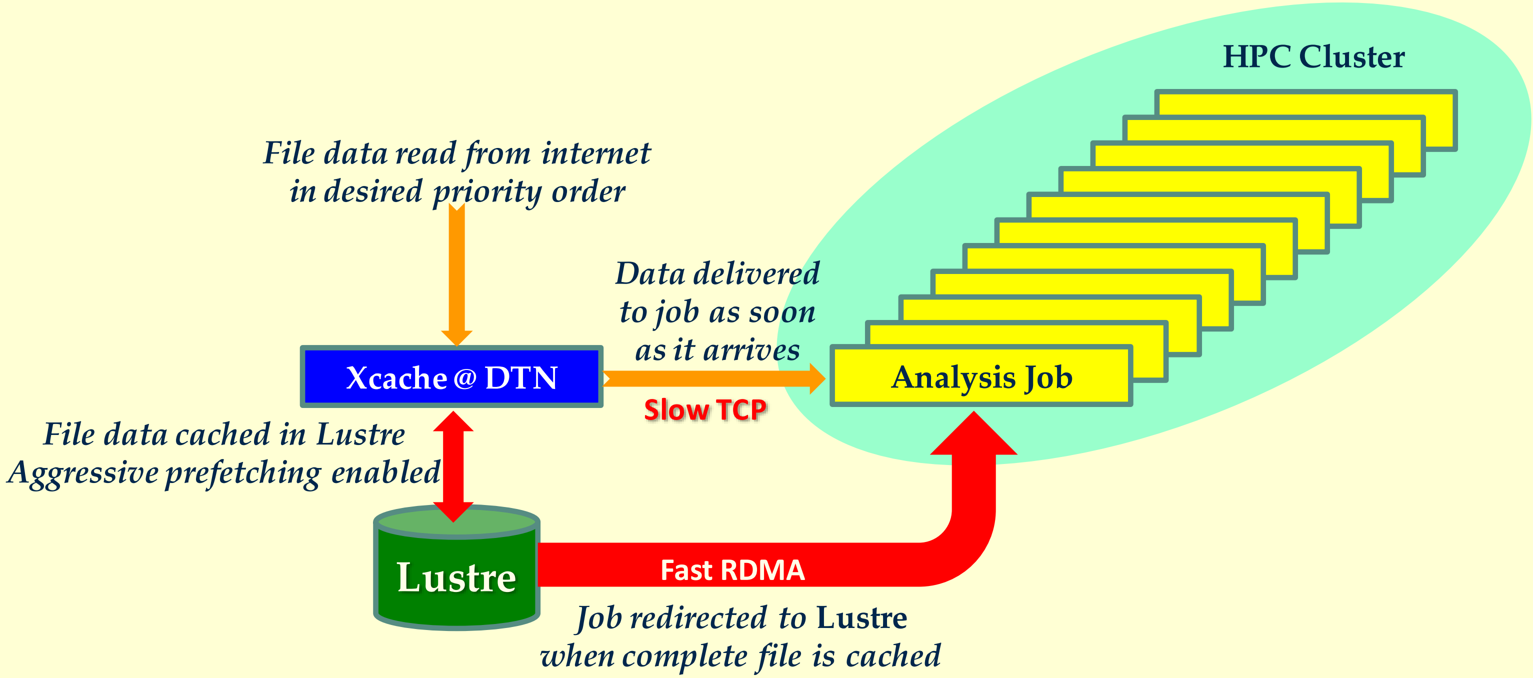
- Protocol has exceedingly low framing overhead
  - 24 bytes for a request and 8 bytes for a response
    - Application data is typically 99.99% of the packet
- Does it really matter?
  - Depends on who you are and what you are doing
    - If you sell bandwidth it's a lousy protocol
      - **XRootD** tries to minimize bandwidth waste
    - If you buy bandwidth it definitely may matter
      - When doing random small sized reads it likely matters
        - This is typical for many HEP/Astro analysis jobs
      - But when transferring multi-gigabyte files, not really
- Protocol can easily fill a 100Gb pipe in aggregate
  - xrootd server architecture favors aggregate performance

Network bandwidth usage II

- **Xcache** may be used to further lower B/W usage
  - **XRootD** software component similar to Squid
    - Provides high performance multi-threaded disk file block caching
      - Something that Squid was not designed to do
  - Suitable for locales where data is reused
    - Typically analysis farms that fetch data over the WAN
  - Some sites have reported a 40% reduction of WAN usage
    - On average there is a 20% reduction in typical HEP use cases
  - Two factors in HEP make **Xcache** useful
    - Many applications only use 30-50% of a file
      - **Xcache** only transfers the part of the file that an application actually needs
    - Analysis jobs are rerun several times with different parameters
      - Much of the same data is needed in a subsequent run

Network bandwidth usage III

- **Xcache** can be configured to better use LAN resources
  - This is specific to HPC's but the usual setup is as follows



Network bandwidth usage IV

- In **XRootD** 5.x provides data-in-motion integrity
  - Driven by **Xcache** requirement to avoid caching dirty data
    - Implemented via pgread requests when not using TLS
      - When TLS is being used falls back to standard read and local checksums
    - Each 4K block is protected by a H/W assisted CRC32C checksum
- Checksum errors are corrected on-the-fly
  - When reading the client requests retransmission
  - When writing the server requests retransmission
- Data-at-rest integrity (in future release)
  - Can configure **XRootD** to save network checksums
    - Data can be checked upon reading (**Xcache**) from disk
    - Network checksum can be reused for transfers

Network bandwidth utilization

- **XRootD** supports multiple data streams
  - An application may get up to 15 additional data streams
    - Useful for improving the speed of WAN file transfers
      - This has been well documented and is a way to mitigate TCP recovery of dropped packets
  - Multiple data streams are also used to mitigate TLS performance
    - The protocol naturally splits into control and data streams
      - Control stream is encrypted
      - Data stream is not encrypted unless required by the site to be so
    - This is automatically handled for the application
      - Site requirements may force all data to be encrypted
        - This is negotiated between the client and server

Network tuning

- See the **xrd.network** directive
  - Rich set of tuning options
    - https://xrootd.slac.stanford.edu/doc/dev53/xrd_config.htm#_network
  - Defaults, though usually work quite well
    - May need adjustment in certain environments
      - For example, k8s or VM's

Container orchestration support

- **XRootD** supports container orchestration
  - Typical ones are Kubernetes (k8s) or Swarm
  - Both introduce issues for network clustered services
    - Virtual networking
      - IP address is arbitrary and can unpredictably change
    - Dynamic DNS
      - IP addresses are dynamically added and removed
      - Registration is essentially ephemeral
  - Supporting orchestration requires some rethinking
    - **XRootD** provides configurable options to address these issues
      - Essentially, the IP address is no longer a useful management tool

SLAC NATIONAL ACCELERATOR LABORATORY

Virtual networking support

- Virtual networks need virtual namespaces
  - XRootD implements such a namespace
  - Site assigns accessible resources relative unique names
    - Normally we think of a resource as a server but it's no longer relevant
    - For file system based services it's actually the file system
      - Any server can export any file system via orchestration
      - For non data services (e.g. via SSI) it's usually the server
    - See the **cms.vnid** directive
      - https://xrootd.slac.stanford.edu/doc/dev53/cms_config.htm#_Toc53611101
  - This name is called a vnid

Virtual Networking ID (vnid)

- The Virtual Network ID (vnid)
  - Clustering component tracks resources by vnid not IP address
    - It also makes sure that the xrootd - cmsd pair is consistent
      - That they are looking at the same file system which might not be the case anymore
  - We do not recommend virtual networking due to overhead
    - Commercial cloud providers have substantially reduced the overhead
    - Open software solutions have not

Dynamic DNS support

- DNS entries are now a spur of the moment thing
  - Orchestration frameworks register IP address whenever
    - Registration can occur in any order irrespective of any other server
  - If you tell xrootd's and cmsd's that DNS is dynamic
    - Mitigation is enabled for delayed registration
      - This prevents failures that would normally be expected to occur in a real network
        - For instance, a non-registered service is configured
    - See **xrd.network dyndns**
    - https://xrootd.slac.stanford.edu/doc/dev53/xrd_config.htm#_network
- **XRootD** is very comfortable with the cloud
  - With containerization features sites have deployed cloud clusters

SLAC NATIONAL ACCELERATOR LABORATORY

Other net oriented features

- Full-fledged clustered proxy server support
  - Scalable load-sensitive mechanism to deal with firewalls
- Configurable TCP keep alive support
  - Additionally, idle socket timeout with forced close
    - Addresses typical “close_wait” issues with certain VM clients
- Full support for public/private 4/6 IP networks
  - Site can optionally describe its IP address rules
    - Used by the clustering component to route requests
      - Automatic matching of compatible addresses for routing
      - Can be used to minimize internal network hops
      - Allows use of a preferred interface when possible
    - Largely to accommodate HPC centers with unique networks
      - Currently used at GSI, Darmstadt

Enhanced Write Support (backend)

- Distributed write recovery
  - For systems that support it (e.g. EOS)
    - Eliminates full file retransmission upon error
      - Writes can proceed using another data server
        - Normally writes are tied to the server of 1st write
- Part of **XRootD** file copy framework
  - Automatically extends to gfal and xrdcp

SLAC NATIONAL ACCELERATOR LABORATORY

XcacheH plug-in (coming soon)

- Accessing **Xcache** origins using **http**[**s**]
  - Broadens data access reach
    - Oriented toward multi-discipline sites
  - Can be used as a Squid replacement
    - Better performance and scalability
  - Based on the plug-in by Radu Popescu
    - Formerly at CERN now at Proton Tech AG
      - Further developed by Wei Yang - SLAC
  - Prototype being tested by ESNET & ESCAPE

Erasure coding client plug-in

- Client side plug-in to support EC writes
  - Based on Intel ISAL
    - Hardware accelerated encoding
  - Leverages **XRootD** pgWrite capability
    - Data in motion integrity with recoverability
- Driven by ALICE requirements
  - Direct writes from the DAQ system to file store
- Developed by Michal Simon (CERN IT-ST-PDS)

Conclusion

- **XRootD** is facile, flexible, and sound
  - Applicable to a wide variety of problems
    - Current release is 5.4.0 (wait until 5.4.1)
      - Next release 5.5.0 at the end of April
- Our core partners
  - CERN SLAC UCSD
- Community & funding partners (*not a complete list*)
  - OSG WLCG ATLAS EXPERIMENT ALICE CMS LSST dCache GSI

Funding from US Department of Energy contract DE-AC02-76SF00515 with Stanford University