SLAC NATIONAL ACCELERATOR LABORATORY

Xrootd and Xcache

Andy Hanushevsky
Wei Yang

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC

- Xrootd
  - Plugin architecture
  - Scaling up
  - Monitoring data streams
- Xcache
  - Features of Xcache
  - Expanding functionalities of Xcache, two examples
  - Thoughts on security and general purpose, shared cache

Xrootd - open, plugin architecture

SLAC

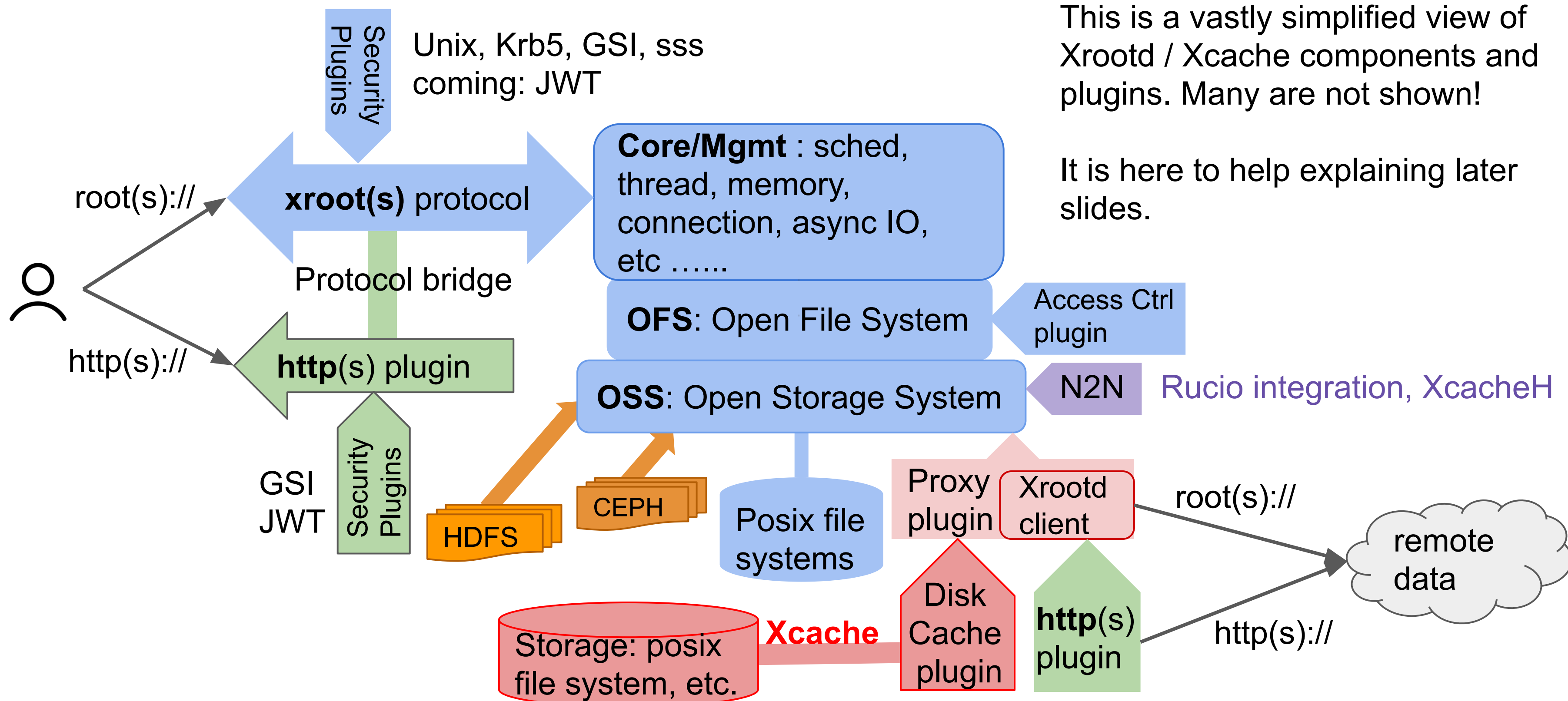
Xrootd was originally an open source storage system

- Developed during Babar era as a static scientific data storage (HEP data)
  - Lightweight and reliable, hardened by the Babar experiment.
- The current Xrootd software stack allows plugin to almost everywhere.
- This greatly expands Xrootd’s functionalities
- Attracted many contributions from people outside of the core Xrootd team
- Supported by: dCache, EOS, DPM, RAL-ECHO/CEPH, Posix file systems

Such an architecture also bring challenges:

- Keeping track and keeping peace of those contributions
- Complex configuration and long list of functions validation
  - Every plugin has something to config
- Documentation

Open, plugin architecture



Scaling up

A cmsd daemon pairing with an xrootd daemon, to form a cluster of nodes

No data striping across data servers **& No** locking

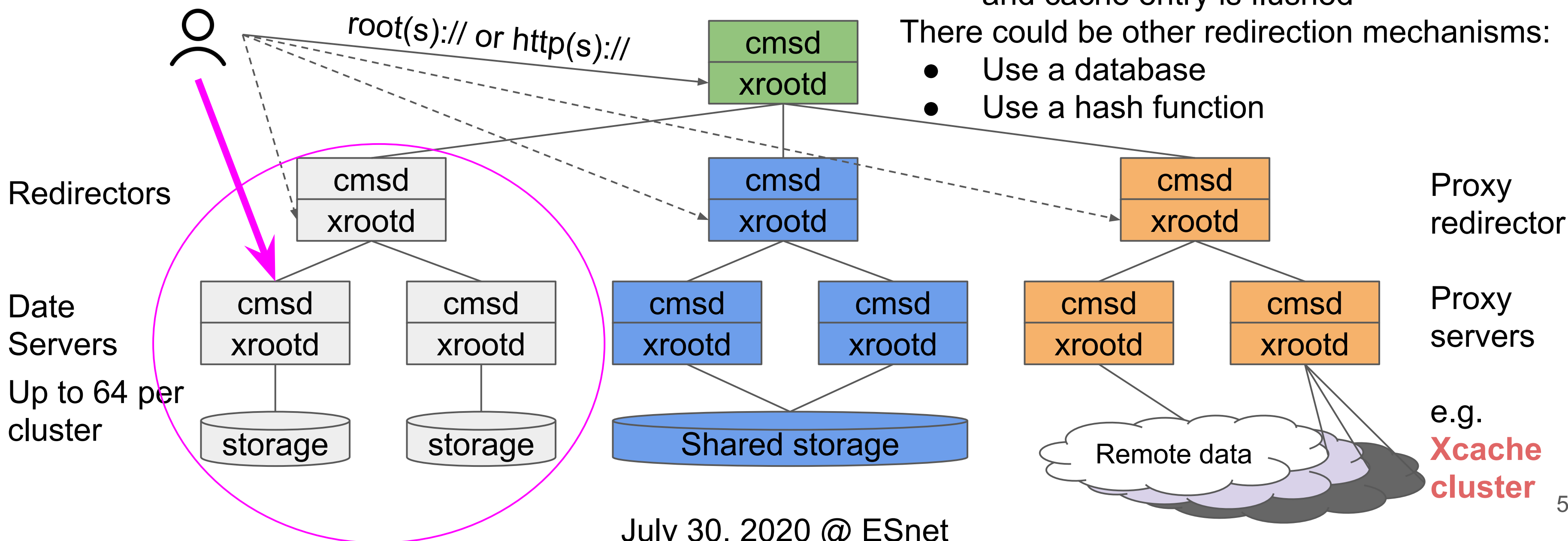
- **Redirect** client to where data reside **!**
- Dramatically simplify metadata operation
- Good for analyzing static science data

Redirection is based on real time query

- "**Who has this file**" ?
- Info is cached with an expiration time
- If cached info is wrong, client complains and cache entry is flushed

There could be other redirection mechanisms:

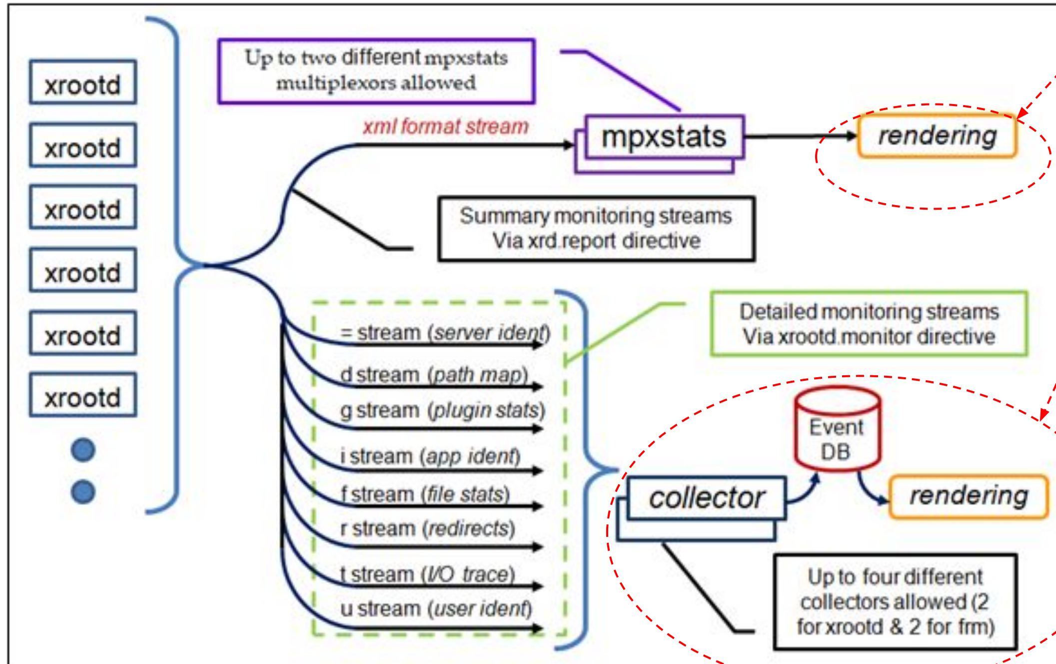
- Use a database
- Use a hash function



Monitoring

Not provided by Xrootd

- Xrootd doesn't provide monitoring dashboards and analytics
  - There are plenty of industry tools to do that.
- Xrootd sends out summary/performance and event data - in **UDP packets**



(periodical) **Summary** data examples:

- Bytes into the cache
- Bytes out of the cache by requests
- Number of times cache hit
- Number of times cache missed
- Number of bytes read but not cached.

(real time) **Detail Events** data, examples:

- "u" stream: client login and identity info
- "d" stream: who opened that file
- "t" stream: IO patterns

Other interesting things about Xrootd

SLAC

1. **XrootdFS**: mount an Xrootd cluster as a posix filesystem on desktop
2. Third Party Copy (**TPC**): a replacement of GridFTP by the WLCG
3. GridFTP plugin from Xrootd storage systems:
    - It is a GridFTP Data System Interface (DSI) written in pure Posix I/O functions
    - Working with Xrootd **posix preload library** -- There is a posix I/O layer upon xrootd protocol
4. Scalable Service Interface (**SSI**): client asks servers to execute arbitrary requests, server response with results.
5. File Residence Manager (**FRM**)
    - Originated from tape stage-in: it runs a custom script
        - The script can: cp, xrdcp, ftp, curl, globus-url-copy, checksum, cook lunch...
    - **It also functions as a (whole file) cache**:
        - Put client on hold until data is staged in, or
        - Client can ask for pre-staging.
6. Server-less Cache: an cache on your desktop without a running daemon

Xcache

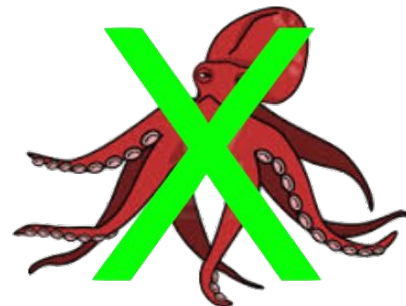
SLAC

For most people, Xcache is the whole software stack that does caching on disk

- Internally, Xcache refers to a plugin developed by UCSD, assisted by SLAC

It is a Squid like cache: we **learned a lot from the Squid** (very old "Squid FAQ")

- Support root(s) protocol and http(s) protocol
- Multi-thread
- Async data fetching (with root(s))
- Caching either file blocks, or whole files
- Designed for both large and small static data files
  - Mostly science data
- Clusterable for scaling up (avoiding sibling query via ICP)
- Customizable cache behavior
  - Mainly through the N2N plugin (slide "Expanding functionalities of Xcache")



A few details about Xcache

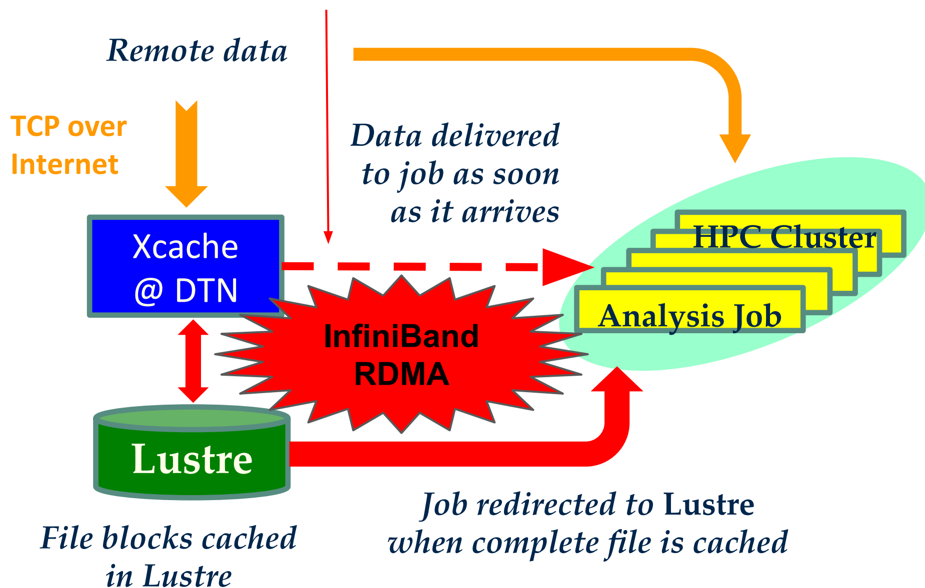
SLAC

- Keep on mind: **Xcache is both a server, and a client**
- A state information file is maintained in parallel for each cached file:
  - Info: original file’s size, blocks committed to storage, # of open/read/bytes read, etc.
- Adjustable RAM buffer to cache data (before they are committed to storage)
- Tunable write-queues to optimize write performance on storage
- Configurable policies to manage cache storage
  - low/high watermark, LRU, unconditional purging of cold file, etc.
- A plugin to decide whether a file should be cached or not
- Handle overload by sending client to somewhere else
  - The CMS redirection (in previous slides - “Scaling up”) is probably a better option

- We have thought of whether we should make Xcache writable
  - So far most people are only interested in a read-only Xcache

Optimize Xcache on HPC

SLAC



Optimization driven by a LDRD @ LBNL

- Run on NERSC DTNs
- Uses main shared filesystem as cache storage
- InfiniBand-like network for communication and data delivery

What we want to achieve:

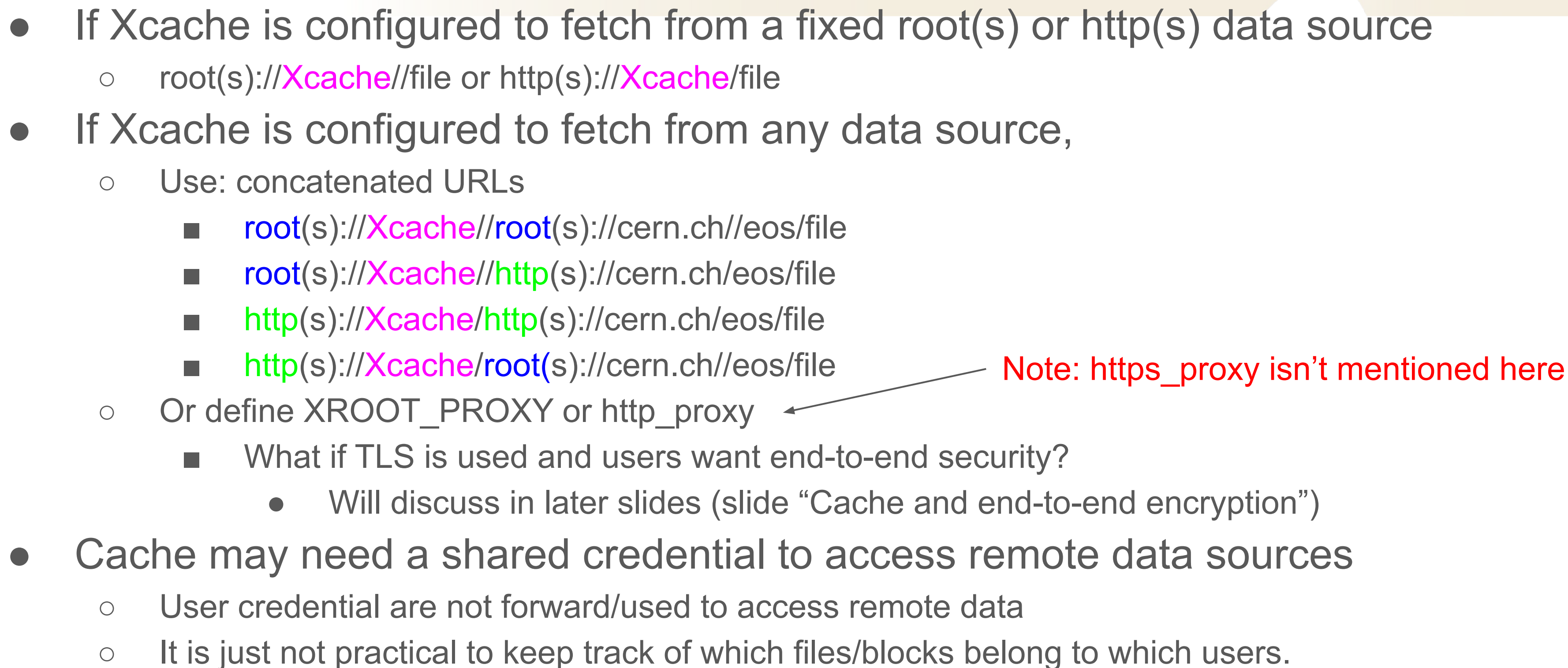
- Cluster ✓
- Deliver fully cached files via the shared file system ✓
- Deliver partially cached files via Xroot protocol over RDMA -- does not exist yet !

Protocols

SLAC

- Support xroot protocol and HTTP protocol, plus their TLS siblings
  - TLS is based on the messy OpenSSL libraries
  - Xroot(s) is the *de facto* standard in HEP
    - A generic remote data access protocol, support posix semantics and preload library
    - Stateful, binary protocol
    - Support async network I/O
  - HTTP(s) is an industry standard
    - Stateless, text header very flexible, at the expense of overhead
    - HTTPS credential caching: turns HTTPS into a semi-stateful protocol
    - HTTP pipelining to achieve some async I/O
- Xrootd server: natively support xroot(s) protocol
  - A protocol bridge to map other protocol to the Xroot protocol + the HTTP(s) plugin (XrdHTTP)
- Xrootd client library (XrdCl) also has a plugin architecture
  - Load plugins based on protocol, default is xroot(s) protocol
  - HTTP plugin to XrdCl (XrdCl-HTTP) is based on Davix library (developed by CERN)

Access Xcache



Expanding functionalities of Xcache

SLAC

- Several plugins exists to expand functionalities of a plain Xcache
  - All of them explore Name2Name translation (N2N is a C++ class in Xrootd)
- **N2N** has 2 key functions that are called for every cache request:
  - lfn2pfn(): convert an incoming URL to an outgoing URL
  - pfn2lfn(): given an outgoing URL, determine storage path for the corresponding cache entry
  - One can program those functions to do many other things
- I am aware of three such plugins:
  - Caching S3-type objects
    - Handle object doesn't start with a slash "/" (absolute path)
  - RucioN2N plugin: An example to show what we can do when a central DM system exists
    - Utilized a central Data Management DB to choose best data source & provide failover
  - XcacheH: An example to show the limitation of caching when end-to-end encryption is required
    - Mainly for HTTP(s) protocol
    - Detect updates at data sources
    - Use Cache Context Manager (CCM) to flush cache entry if cache origin is updated

Xcache with RucioN2N

SLAC

- Rucio is a central data management system developed by ATLAS
    - Data grouped as datasets (sets of data files)
    - Each data file has a logical file name (LFN), along with file size, adler32, expiration, etc.
    - Records replica locations around the world. These locations can change over time
    - Once created, a data file never change (static). New version has a new file name, is a new file
- Users use LFN to access ATLAS data file via an Xcache with this plugin
    - lfn2pfn() asks Rucio for a list of data sources, in form of a Metalink (sorted by GEOIP)
        - If the first data source fails, try the second data source
        - XrdCl handles of Rucio metalink, and complex site failure scenarios
        - Metalink is cached in memory for 1 day
    - pfn2lfn() will decide the cache entry location based on LFN, regardless of data source used.
    - Benefits to users: they don't have to keep track of the location of data replicas
- For completeness, it can still function as a plain Xcache

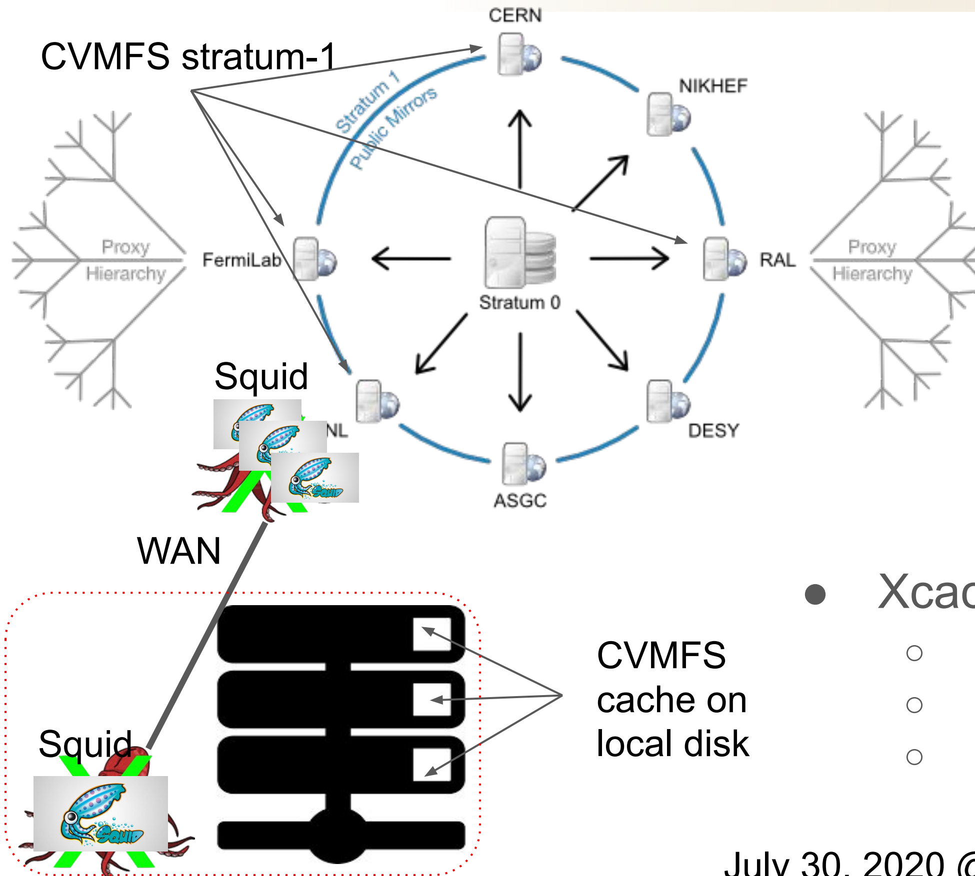
A challenge: Can Xcaches siblings discover each other's contents?

XcacheH

- Focus on supporting HTTP(s) protocol
  - Cache entry are mapped to storage using full URL from client, including CGI
  - After an initial period, XcacheH will check “mtime” at the origin to detect changes
    - Then decide whether a cache purge / refetch is needed.
    - It is possible that source data changed during the initial period
      - This risk always exists.
      - Working with static science data avoids this issue

- Still rooms to improve (XcacheH is a very recent development)
  - Response from web servers differ by server type, site configure, or even individual files
    - Always ask: How does Squid handle this issue? Should XcacheH do the same thing ?
  - Better to have place to save info related to remote file metadata
    - For example, can XrdPosixCache interface allow XcacheH to write to cache file’s xattr ?
  - Code optimization
    - XrdHTTP: extra stat(); XrdCl-HTTP/David: open()/read()/close() cycle.
  - Davix: metalink function doesn’t work - note: curl is talking about dropping metalink support

XcacheH and CVMFS



- CVMFS is a read-only global FS
  - Data “published” at stratum-0
  - Replicated to stratum-1
    - spread load; shorten latency
  - Usually have multiple layers of cache
    - Some are Squid clusters
- Squid doesn’t prefer large files
  - CVMFS chop a large file to small pieces
  - CVMFS avoids distributing large file
    - Mostly distribute software
- XcacheH can replace Squid (tested at SLAC)
  - CVMFS will be free from the constraint by Squid
  - Can efficient distribute large data files
  - Much large cache cluster on fewer HW

XcacheH: working with curl and wget

SLAC

- There are several ways for curl and wget to use XcacheH
  - Concatenated URL: http(s)://XcacheH:port/http(s)://cern.ch/index.html, or
  - http_proxy=http://XcacheH:port curl http://cern.ch/index.html
  - http_proxy=https://XcacheH:port curl http://cern.ch/index.html
  - https_proxy=http://XcacheH:port curl https://cern.ch/index.html
- The above all works
  - As long as one of the following not HTTPS
    - Xcache URL
    - Destination URL
    - https_proxy
- One combination left behind:
  - https_proxy=https://XcacheH:port curl https://cern.ch/index.html
  - It doesn't work with XcahceH, but work with Squid

Cache and end-to-end encryption

What if everything is https ?

SLAC

Compare two different ways of using an Xcache (https://osggridftp01.slac.stanford.edu:8443)

```
$ curl -v https://osggridftp01.slac.stanford.edu:8443/https://wt2.slac.stanford.edu/images/junk1
...
> GET /https://wt2.slac.stanford.edu/images/junk1 HTTP/1.1
...
< HTTP/1.1 200 OK
```

Traffic went through successfully

- XcacheH encrypts traffic with both ends. XcacheH can see the data
- This is NOT an end-to-end encryption. XcacheH is a Man-in-the-middle

```
$ https_proxy=https://osggridftp01.slac.stanford.edu:8443 curl -v https://wt2.slac.stanford.edu/images/junk1
...
> CONNECT wt2.slac.stanford.edu:443 HTTP/1.1
...
< HTTP/1.1 400 Unknown
```

Traffic could not go through.

- HTTP CONNECT is meant to create an end-to-end encrypted tunnel
- Such a tunnel will bypass the cache
- Squid will honor such a request and **route traffic**, but will cache nothing
- XcacheH will refuse the HTTP CONNECT request

Thoughts: it is about "Trust"

SLAC

- Curl and wget are just two applications
  - They choose to send GET or CONNECT under those scenarios
- Other applications may behave differently
- This bring out a number of issues with general purpose, shared cache
  - End-to-end encryption excludes the idea of such a cache
  - User authentication/authorization also exclude a shared cache
    - Because keeping track of who owns which file/block in a cache is not practical
- But dedicated user private cache is still possible
  - Users can tell their own applications to trust his/her own cache sitting in between two ends of a supposedly TLS connection
  - User can supply a credential to his/her own cache to authenticate with the remote data source.
- No such problem when TLS and authentication is not used.

Summary

SLAC

- With an open, plugin architecture, Xrootd expands from a storage system to other type of services
- Xcache is one of those. It generated lots of interests
- Xcache's functions are expandable too by plugins. We gave
  - An example of integrating Xcache with a central data management system
  - An example of emulating Squid, but with Xcache's innate high performance characteristics.
- The development of XcacheH forced us to think of the desire and relation of
  - a general purpose, shared cache
  - end-to-end encryption
  - access control
  - Something to think of when we design applications that utilize caches.