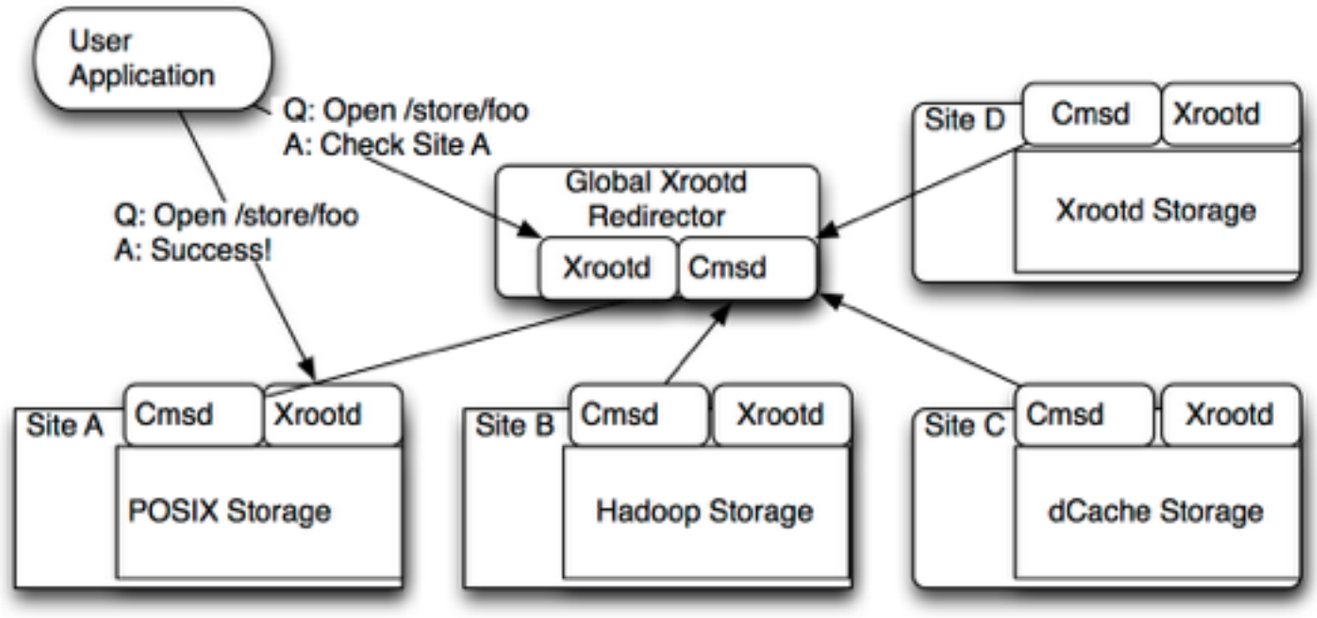
CVMFS and Data Federations

The Big Picture

The Big Picture - Data Federations

- WLCG has been working with Data Federations for many years now.
  - Typically, clients connect to a centralized - or lightly distributed - “redirector” and ask for a file.
- The redirector determines the location of the file - often by broadcasting a location query to many sites.
- Clients are told to access & stream file from some other site.

CMS’s AAA federation



Note:

- Data goes from site to client.
- Federation *overlays* existing storage system.

Data Federations - Why?

- Data federations offer very simple access semantics to clients.
- Allows for identical access methods for local and non-local data.
- Scales well in practice.
- Work well with large datasets - and cache unfriendly workflows.

Data Federations - Why Not?

- Trade simplicity of access for greater dependence on the WAN.
    - Particularly, keeping data locality can be difficult.
- **METADATA**: Most of the data federations have relatively weak metadata guarantees: VOs need to have their own cataloging system “on top”.

CVMFS

- CVMFS provides an extraordinarily scalable distributed metadata interface.
    - Updates are released as transactions; namespace can be aggressively cached via a network of HTTP proxies.
- POSIX! Never underestimate how much users love POSIX.
- However, the existing squid networks are not designed for large (multi-TB) working set sizes.

The Big Picture:

Can we combine the data scalability of data federations with the metadata scalability of CVMFS?

Changes Needed for CVMFS

What's needed?

- To access a data federation like AAA:
  - Data federation files cannot be changed (compressed or change filename).
    - May be infeasible to download them all to the repository.
    - Cannot store files on the Stratum-1 network. Cannot pull them through the existing HTTP proxy network.
- So, we set out on 4 major modifications…

(1) “External Data”

- CVMFS hashes both metadata and data; files are downloaded from the Stratum-1 (via a proxy), by combining a prefix (http://cvmfs.example.com/cms.cern.ch) and the hash (d0/55d84d5d0c0af343b43a8aeb5e9ca2a8d3a351).
- Added ability for files to be marked as “external”.
    - When such a file is encountered, CVMFS will access an alternate prefix (http://federation.example.com/cms.cern.ch) and **filename** (/store/mc/foo/bar).

(2) File Grafting

- CVMFS needs various file metadata to add it to its catalog.
    - Usually this is generated when the file is placed on the repository's disk, ten compressed and checksummed during the publish operation.
        - However, we cannot necessarily copy all files to repository disk.
- Hence, we added *grafting*:
    - One CLI tool will produce the checksum information in a small file.
    - Simply add the checksum file to the repository and publish; CVMFS will treat this as if the file was actually present in the repo.
        - Mostly useful when combined with (1) external data.

(3) Uncompressed Files

- For some repositories, it is prohibitively expensive to keep both uncompressed and compressed copies of data. For other use cases, it may be ineffective to compress files in CVMFS.

- We can now mark each file's compression algorithm scheme in the catalog - currently zlib and no-compression are allowed.

- Putting together (1), (2), and (3), we can effectively publish files from a data federation and allow clients to access them in place.

(4) Security

- Unlike software, many large datasets are *much* sensitive - it's detector output, unpublished work, etc.

- We extended CVMFS so a repository can be marked as secure and requiring a credential to access.

    - When a process accesses the mount, the cvmfs2 process will inspect the user's credentials and compare it to a whitelist of allowed DNs or VOMS FQANs*.

    - If the credential is authorized - and the file is not in cache - then CVMFS will use it to secure a HTTPS connection to the remote host.

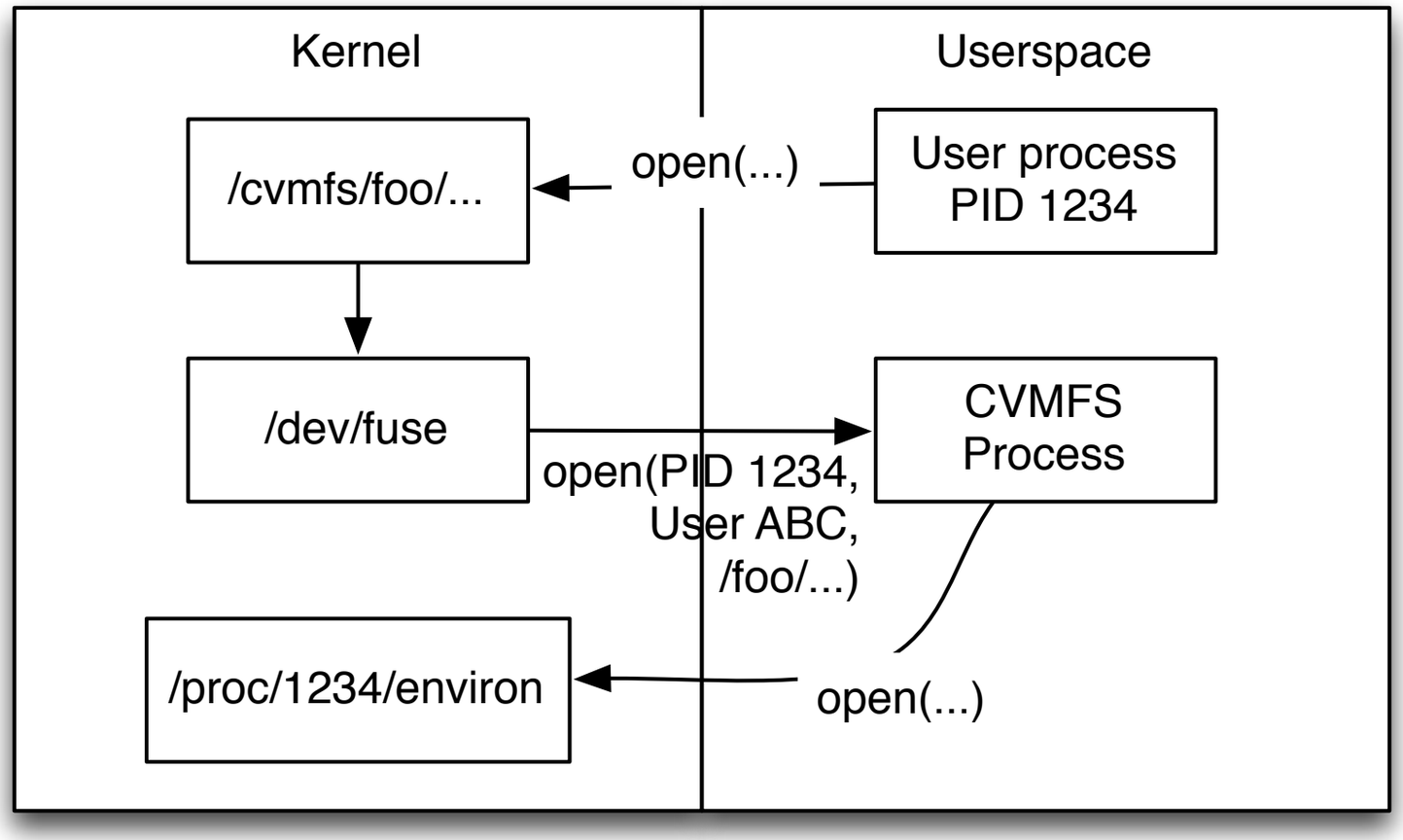
    - HTTPS can be used for both metadata *and* data: only the root metadata catalog *must* be available over HTTP (this way root directory can be mounted on boot).

- Note we authorize twice - once on the worker node, and again on the server.

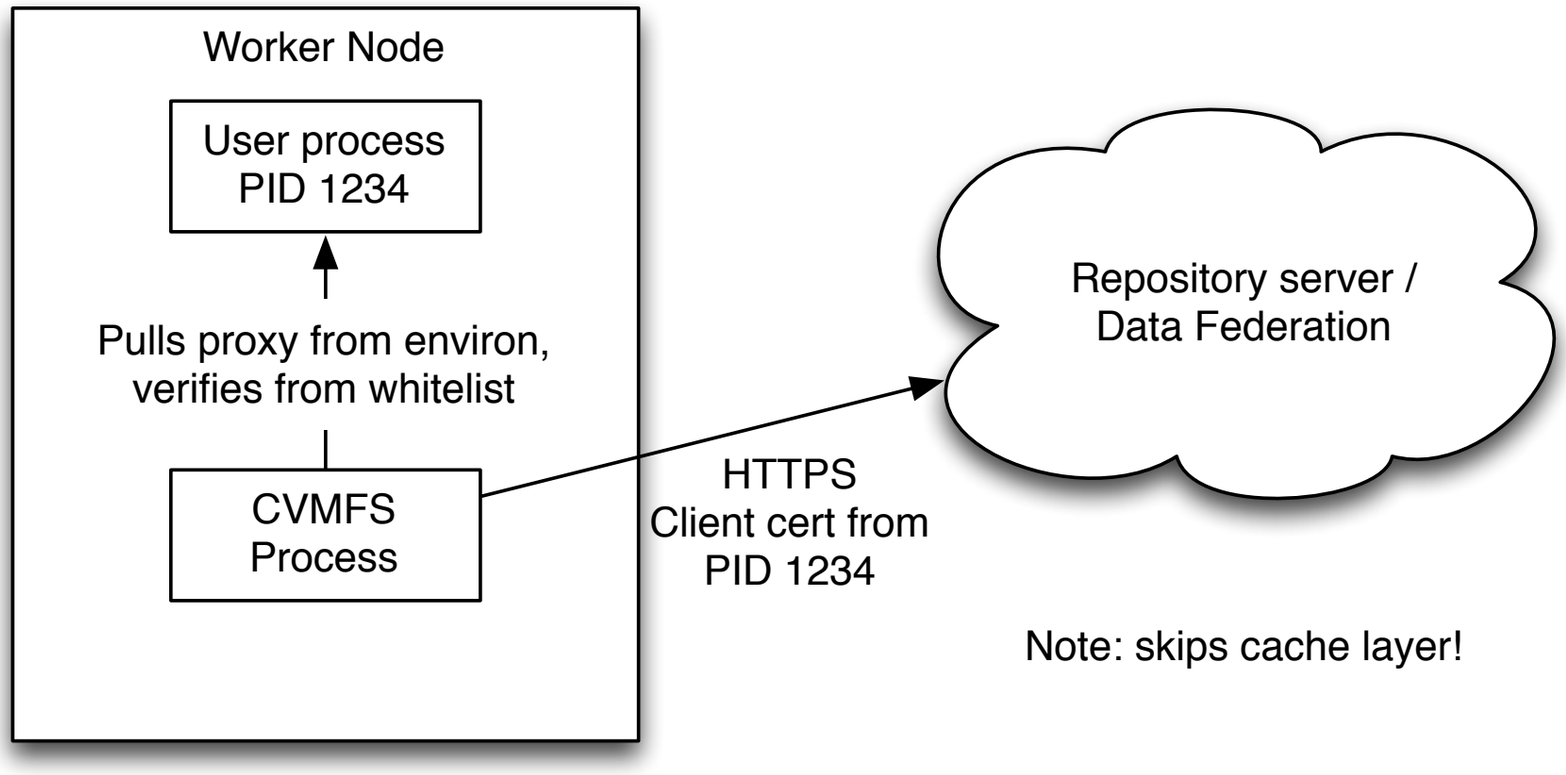
- This setup - especially interaction with Globus/VOMS - turned out to be … well, tricky. Didn't make it in time for 2.2.x; currently in 2.3.0.

* This is a generic mechanism; could be extended to oauth2

Secure CVMFS



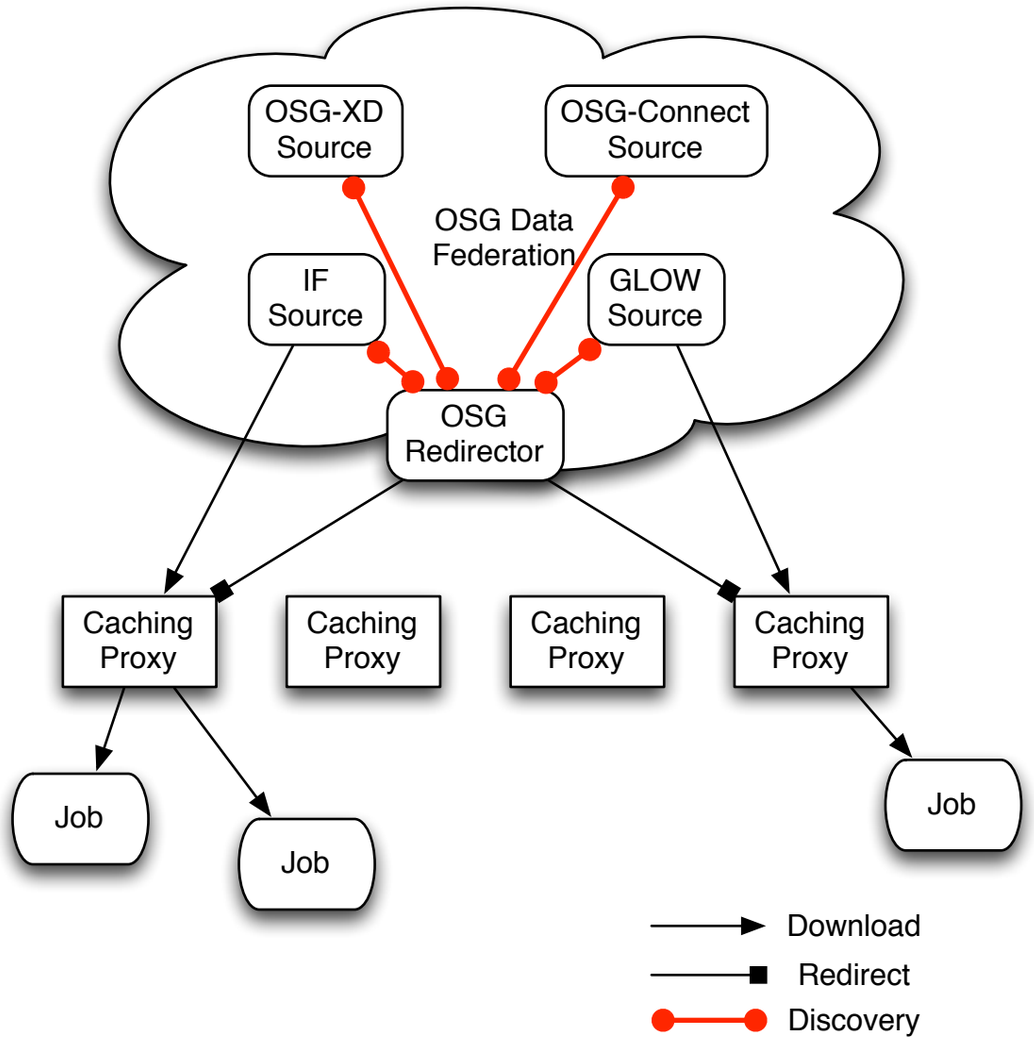
Secure CVMFS



Putting it together

StashCache

- OSG runs a data federation (similar to AAA) that VOs can join.
  - Currently limited to world-writable data.
- We then front the federation with a series of caches. This way, OSG-run caches do the brunt of data serving.
  - Targeting workflows with working set sizes of 1GB-10TB.



stash.osgstorage.org

- A separate repository host, walks the directory tree of the OSG VO's origin server.
  - Synchronizes and publishes about once an hour. Stratum-1 delays mean propagation time is ~1.5 hrs.
- From the login host, users write into their ~/public/ directory; waits until the files show up in /cvmfs, then submit corresponding jobs.
  - Hope is users can treat this as a global read-only filesystem.
- We have a similar repo for the nova experiment - hosts files for a workflow that needs about ~500MB / job.

ligo.osgstorage.org

- Nebraska hosts the last few runs of LIGO data - about 5-10TB.
    - As data comes from the LIGO detector, we publish new files within ~5min.
- Nebraska runs a bank of 10 HTTPS hosts that allow only LIGO.
- Authorization in the repository itself:

```
$ attr -qg authz /cvmfs/ligo.osgstorage.org/ | grep -v '^/'
x509%
osg:/osg/ligo
LIGO:/ligo
```

Looking Forward

- I believe we've made a breakthrough in providing a scalable POSIX interface for data federations.
    - We are partnering with a few VOs to roll this out in practice.
- A next project is cms.osgstorage.org; providing a CVMFS-based front to the entirety of the CMS dataset.
    - Prototype already exists - greater than 1PB published. Publishes sites that export HTTPS with AAA.

Open Projects

- **Low-latency publication**: Users have to wait 1-2 hours for data to show up in stash.osgstorage.org; I think it's realistic to decrease this to 5-10 minutes.
    - See Jose's talk about Stratum-1 improvements.
    - Would like users to be able to force an update.
- **Performance**: Probably can achieve a potential 10-20% performance boost.
- **Multi-user repos**: Each repo is owned by a single Unix user. Repos are somewhat difficult to operate: to give the 50-100 users their own repo would be prohibitively difficult.
    - I want to have the existing POSIX-based write API available for a multi-user repo like stash.osgstorage.org.
- **Large-scale caching**: For PB-scale repos, I would like to see multi-tiered cache based on Ceph (or similar object store technology).
- **Runtime Data Verification**: As we move an increasing amount of data through the disk, I worry about disk errors that occur *after* initial download. I want each byte delivered to the user from the cache to be checksummed.

Questions?