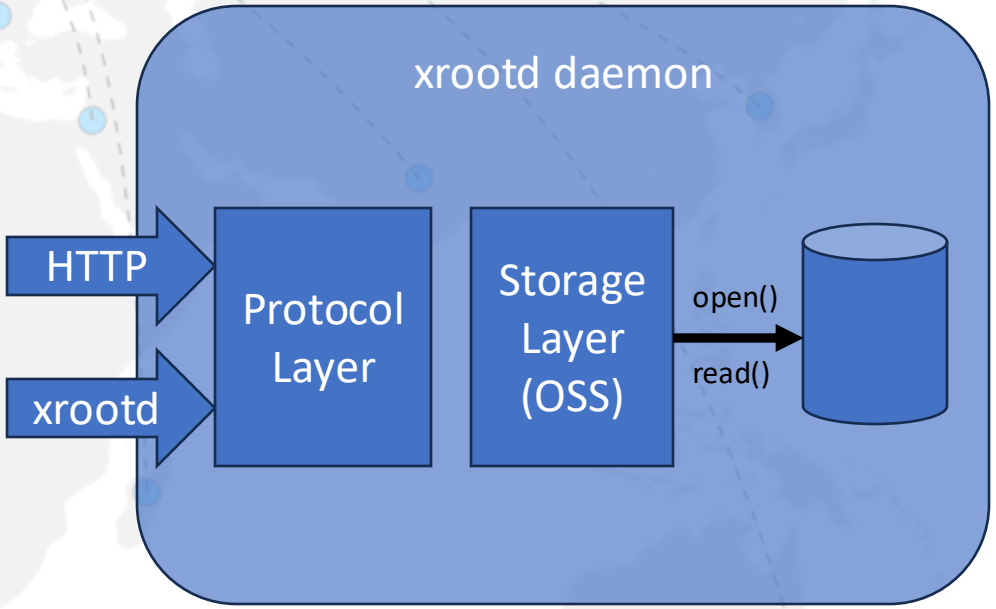
Pelican: Backends to Globus, HTTP, and S3

Pelican and Storage Backends

OSG has a long, storied history in using POSIX backends:

- Built-in POSIX backend translates XRootD storage API to POSIX open/read/write/close/etc calls as a fixed user ("xrootd").
- The "multiuser" backend additionally changes the per-thread filesystem UID/GID. Requires you tell XRootD what Unix user to use (story for a different day).

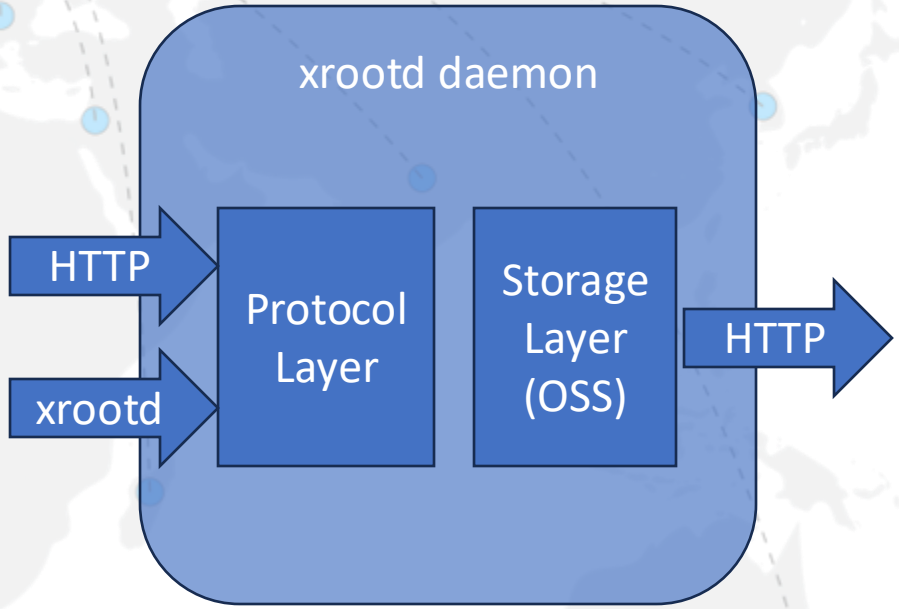


Going beyond POSIX

- However, part of Pelican's success will be based on how many scientific data repositories we can integrate into OSDF.
- *Rarely* do we have direct POSIX access to remote dataset repositories.
- More commonly, a HTTP-based protocol!
- We have begun developing new plugins for HTTP-like things:
  - https://github.com/PelicanPlatform/xrootd-s3-http

Basic Concept - HTTP

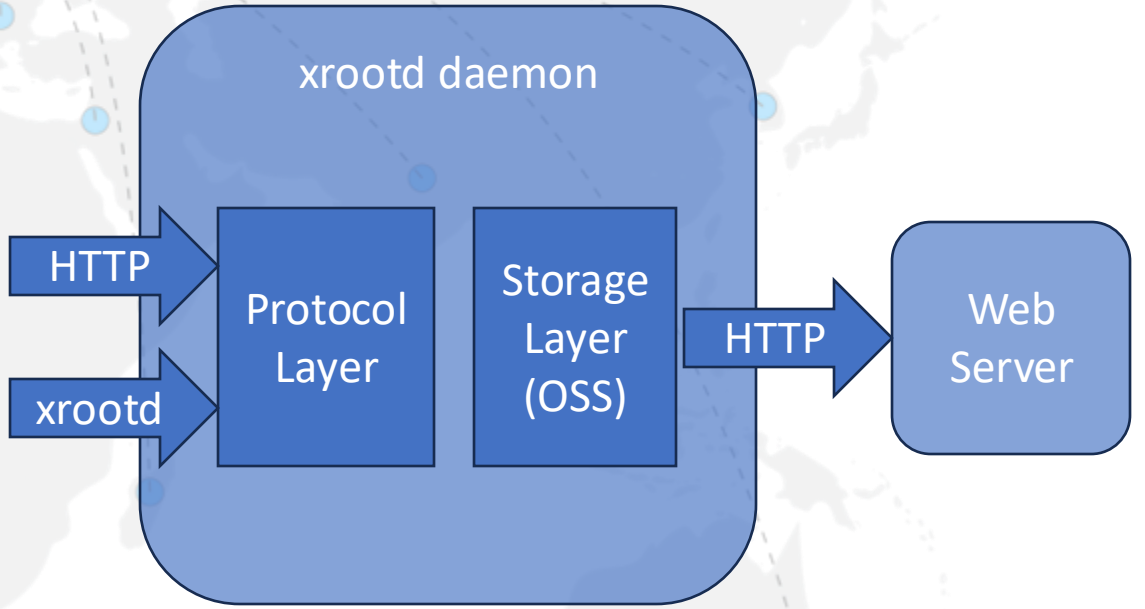
- The repository builds two libraries, libXrdS3 and libXrdHttpServer.
- libXrdHttpServer translates the OSS API to corresponding libcurl requests.
  - Requests are performed in an inline, blocking manner.
  - Open, Stat => HEAD
  - Read => GET
  - Write => PUT



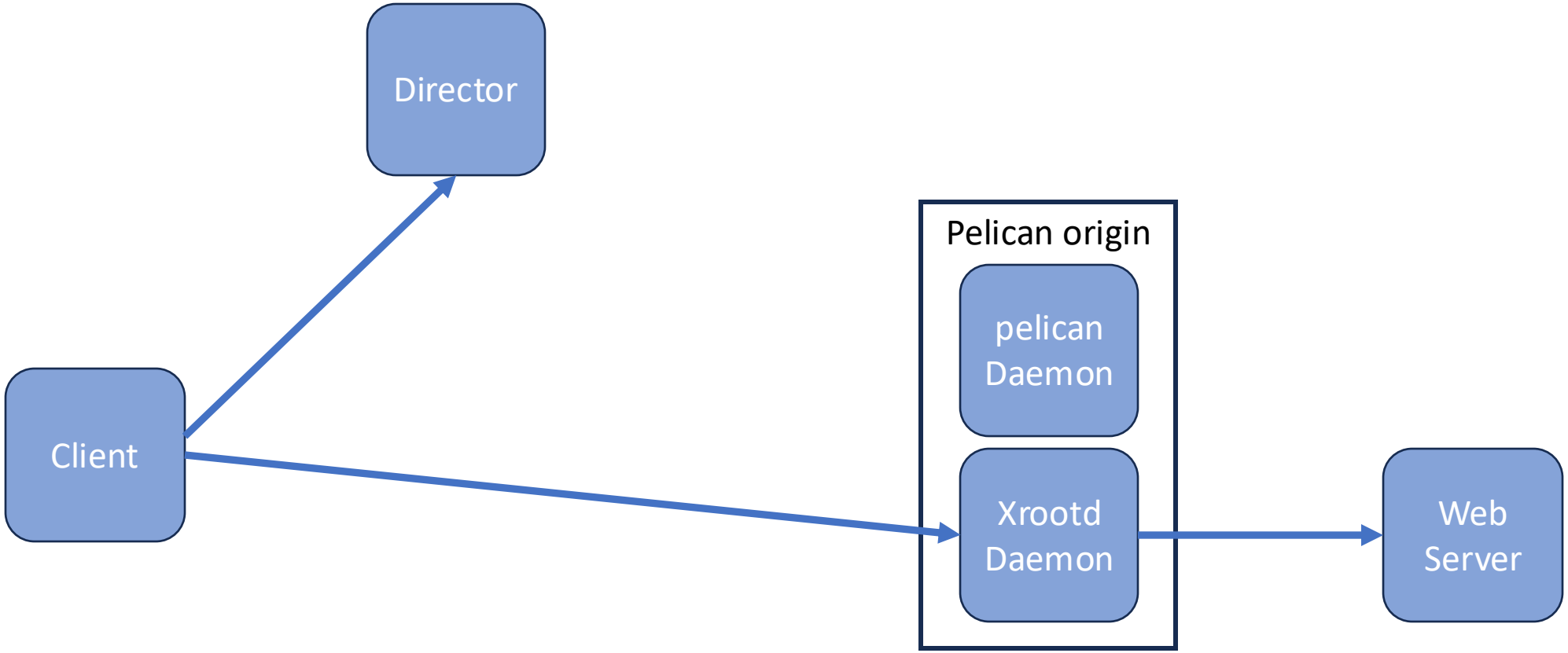
Basic Concept - HTTP

- For HTTP, the setup is designed to export a simple, nearby web server.
  - E.g., it's not meant to proxy an entire data federation; does not do anything clever with redirects.
- Sample simple configuration:

```
ofs.osslib libXrdHTTPServer.so
httpserver.url_base https://example.com/foo
```



And into the data federation



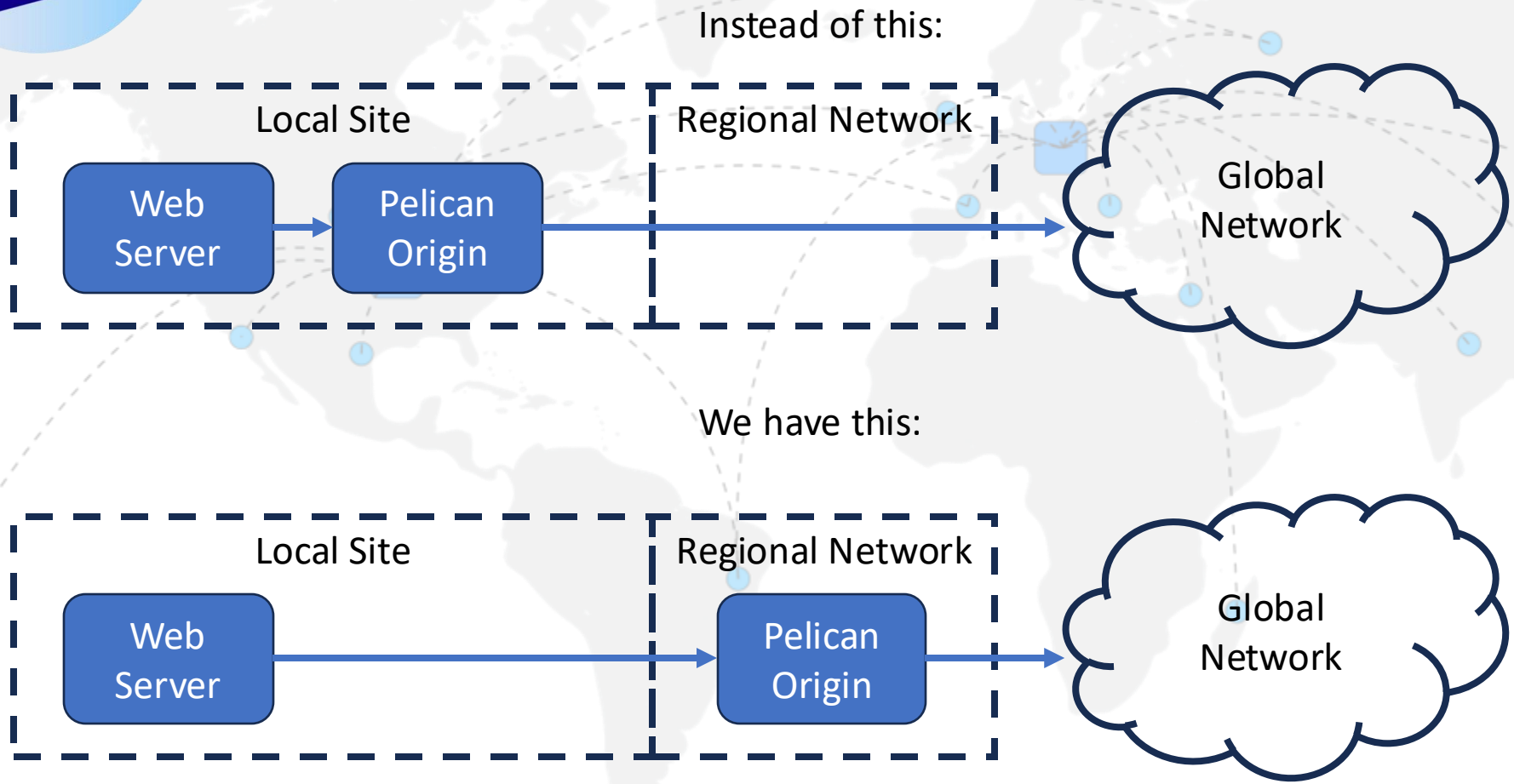
HTTP/S3 and the OSDF

- Note we have no locality requirements – it matters little if the Pelican origin runs in the same physical rack as the backend webserver or simply “nearby”.
- In the US, we utilize the **National Research Platform** (NRP) to place the origin at the service at the nearest network point of presence we have available.

HTTP/S3 and the OSDF

- The NRP provides a single, “stretched” Kubernetes cluster.
- Can use the “typical” Kubernetes control plane methods – it just happens the cluster is spread across the US.
  - We use flux V2 / GitOps for the production deploys.

HTTP and the OSDF



Configuration - HTTP

- The configuration allows “mounting” URLs at any storage prefix.
- Example: suppose you want the contents of https://example.com/foo to be exported as prefix /bar from XRootD:

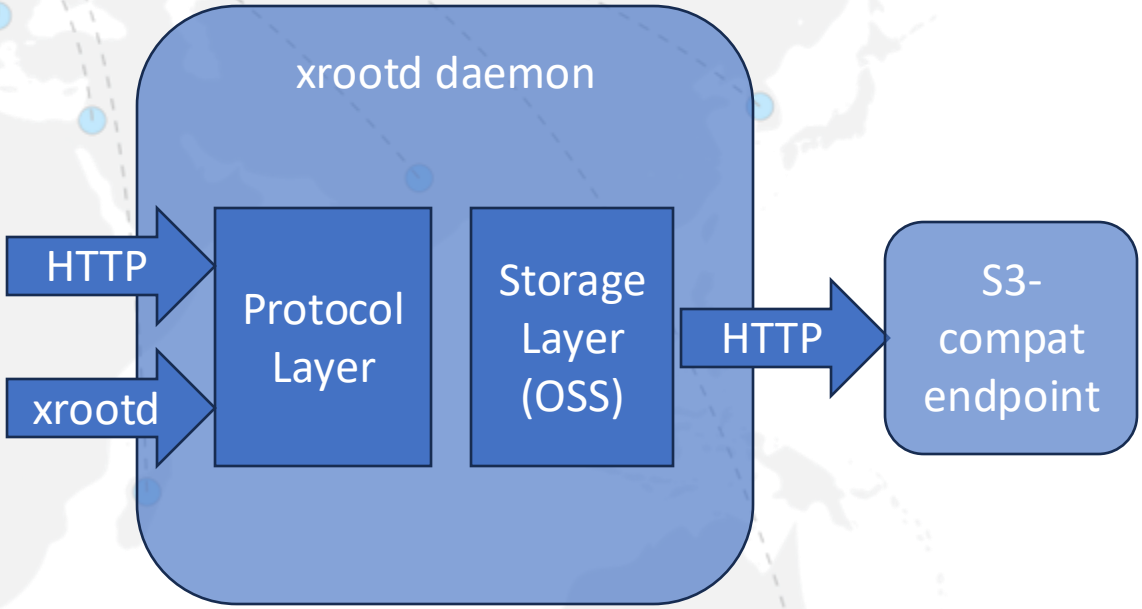
```
ofs.osslib libXrdHTTPServer.so
httpserver.url_base https://example.com/foo
httpserver.storage_prefix /bar
```

HTTP backend – functionality and ‘gotchas’

- The following works:
  - Opening & Reading files, including vector reads & page reads.
  - ‘Stat’-type functionality
  - Single-operation writes
- The following doesn’t (but are in the plans):
  - Multipart writes
  - Checksums
  - Directory listings

S3 Plugin

- Sharing the common libcurl-based infrastructure, we have the S3 OSS plugin.
- Similar concept: translate the XRootD OSS API into a sequence of S3 commands.
  - Includes support for “directory” listing!
- Tested with AWS S3, Ceph’s RGW, and MinIO.

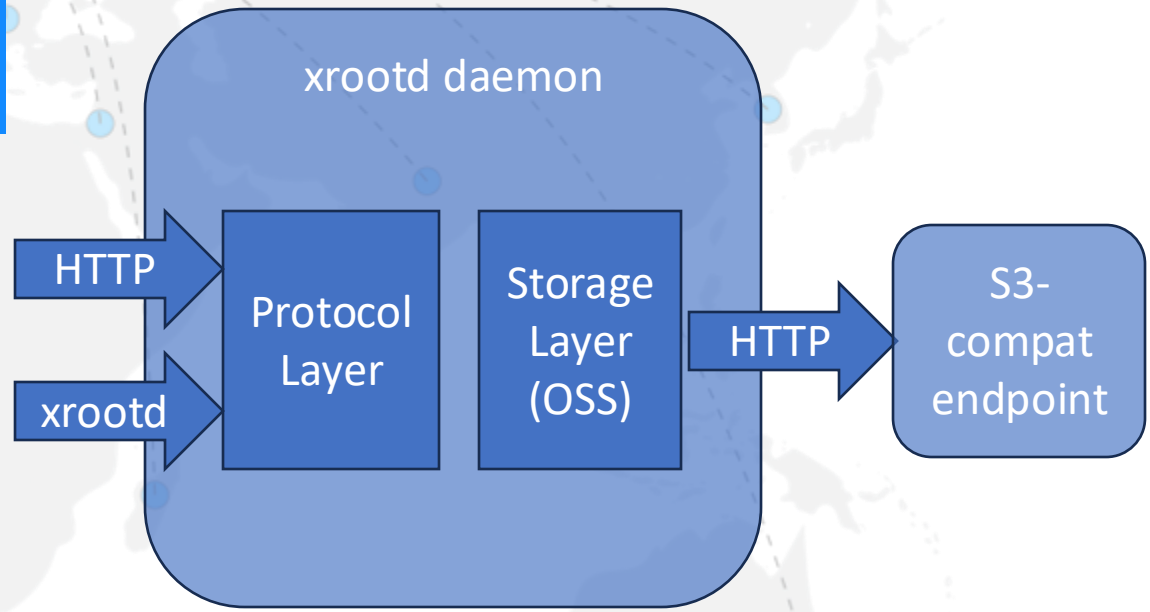


S3 Plugin – Export all Buckets

The plugin can export all buckets at the S3 endpoint.

https://origin.example.com/prefix/foo/bar/baz

- https://origin.example.com: origin URL (could be 'root://origin.example.com' as well).
- /prefix: Admin-specified export prefix.
- foo: name of bucket.
- bar/baz: name of object.



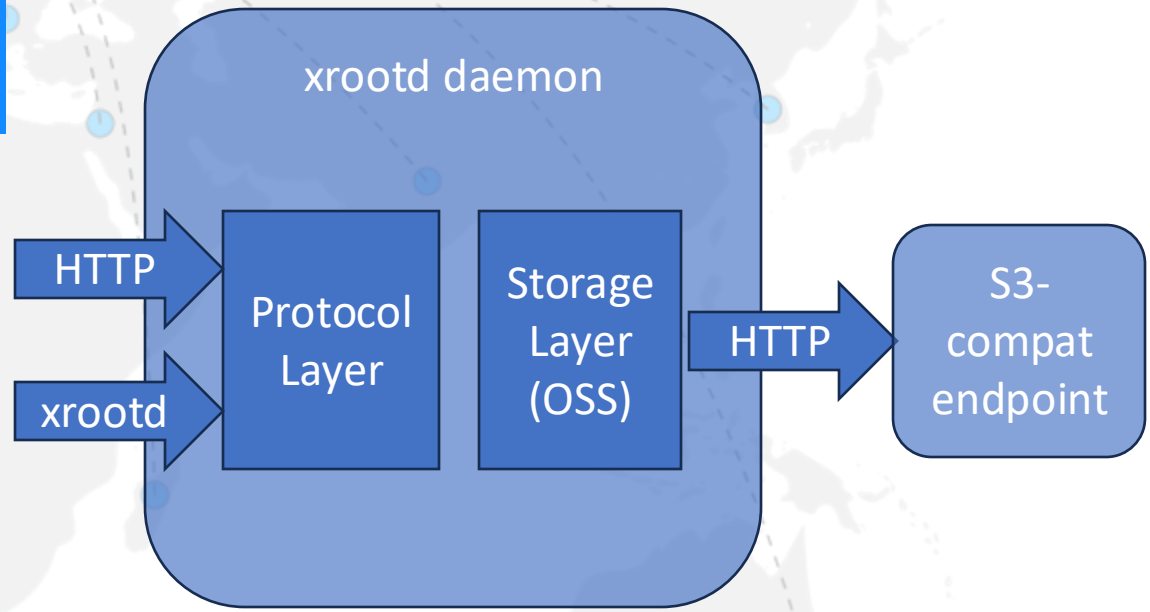
S3 Plugin – Export specific buckets

The plugin can also export just a specified bucket.

In either mode, you can specify the credentials to use with S3

https://origin.example.com/prefix/foo/bar/baz

- https://origin.example.com: origin URL (could be 'root://origin.example.com' as well)
- /prefix: Admin-specified export prefix.
- foo/bar/baz: name of object
- Bucket name to use is specified in configuration and not from the URL.



S3 Configuration

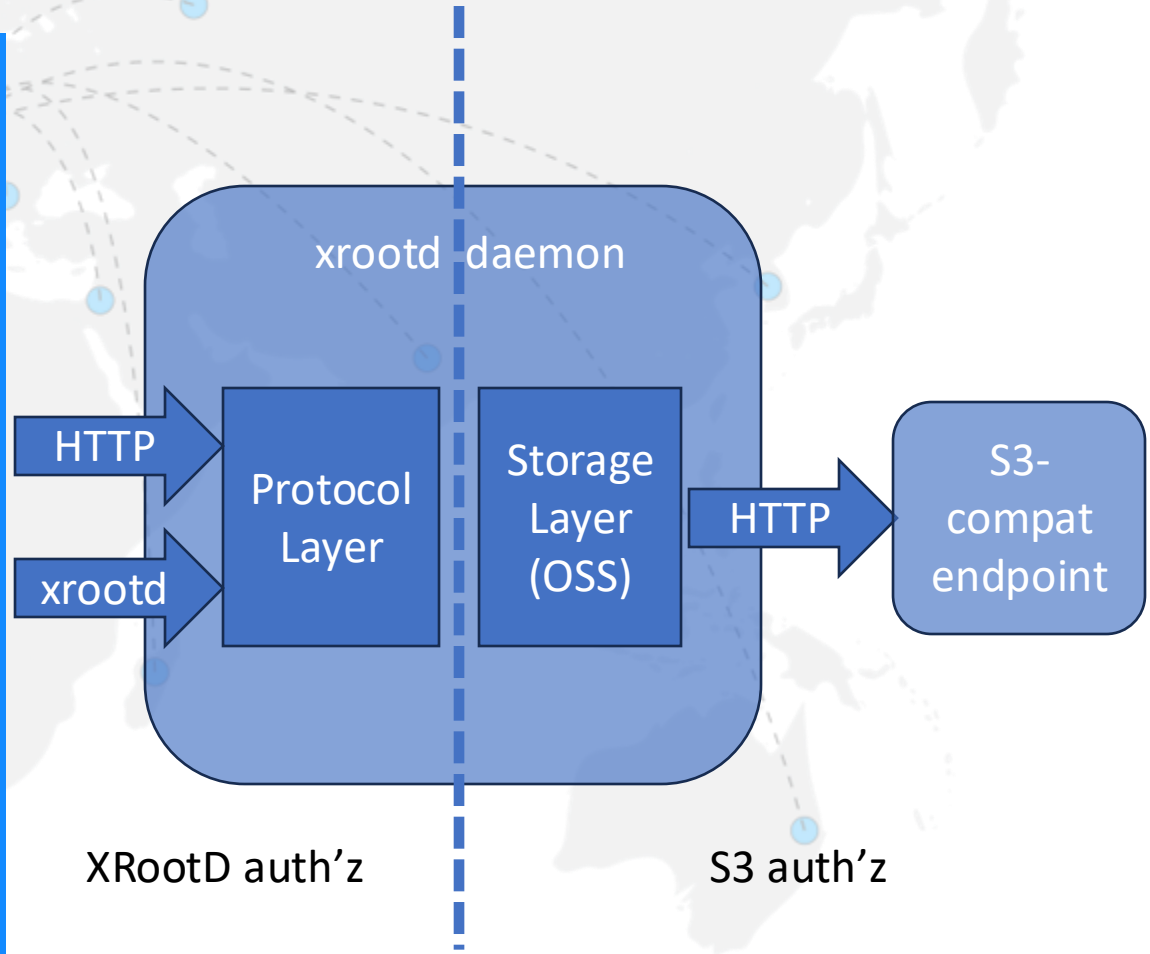
- Each configured S3 backend goes between an “s3.begin” and “s3.end” directive.
  - Specify one per bucket; if no bucket is given, it’ll switch to “export all buckets” mode.
- s3.url_style can be “virtual” or “path”, depending on your S3 service’s configuration.
- s3.region is optional for some implementations (Ceph RGW).

```
s3.begin
s3.path_name      /prefix
s3.bucket_name    my-great-bucket
s3.service_name   s3.amazonaws.com
s3.region         us-east-1
s3.service_url    https://s3.amazonaws.com
s3.url_style      virtual
s3.access_key_file /home/b/access-key
s3.secret_key_file /home/b/secret-key
s3.end
```

S3 Plugin – A note on authorization

Note we have decoupled the XRootD authorization from the S3 authorization!

- The incoming request or session is authenticated/authorized based on the XRootD authorization system (e.g., X.509, VOMS, tokens).
- Once authorized, the generated request to S3 is based on the admin-provided S3 access / secret key.
  - The credential is not passed through from the incoming request.
  - Onus is on admin to ensure the auth'z aligns between the two layers.



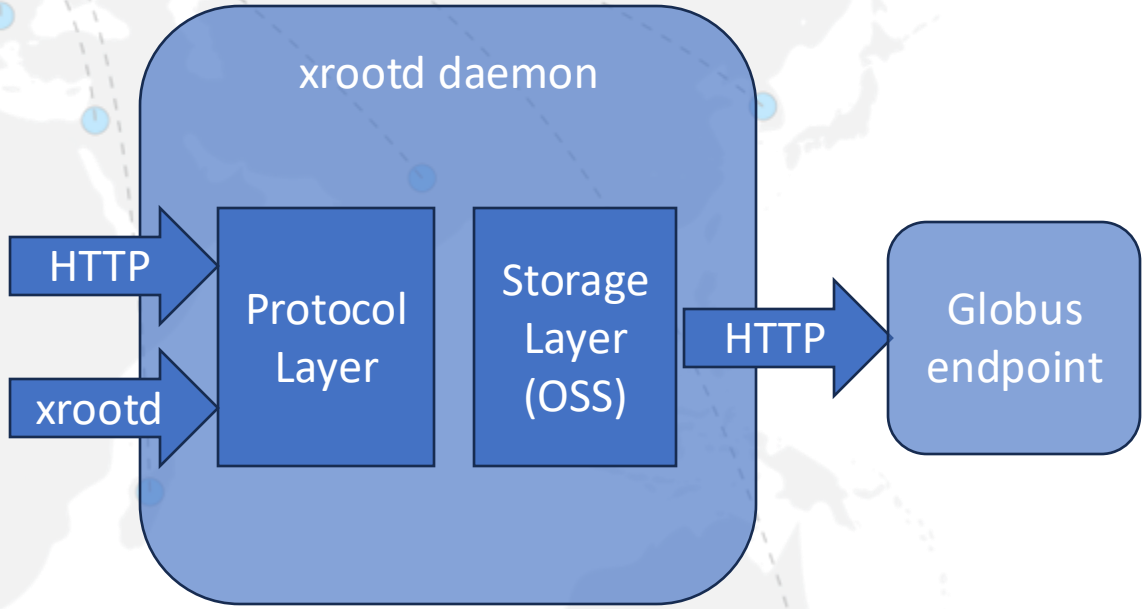
S3 and the OSDF

In the OSDF, S3 is used for:

- Exporting AWS OpenData
  - Amazon has a program where large, popular scientific datasets are hosted for free (no egress costs!).
  - One S3-backed origin for each relevant AWS region.
- (In-progress) Exporting NOAA datasets in AWS:
  - NOAA (National Oceanic and Atmospheric Administration; a US government agency) has negotiated an egress-free contract with Amazon.
- Exporting a Ceph RGW instance at UW-Madison:
  - Used for small-to-medium-scale datasets for teams that have nowhere else to go.
  - ~400TB of capacity.

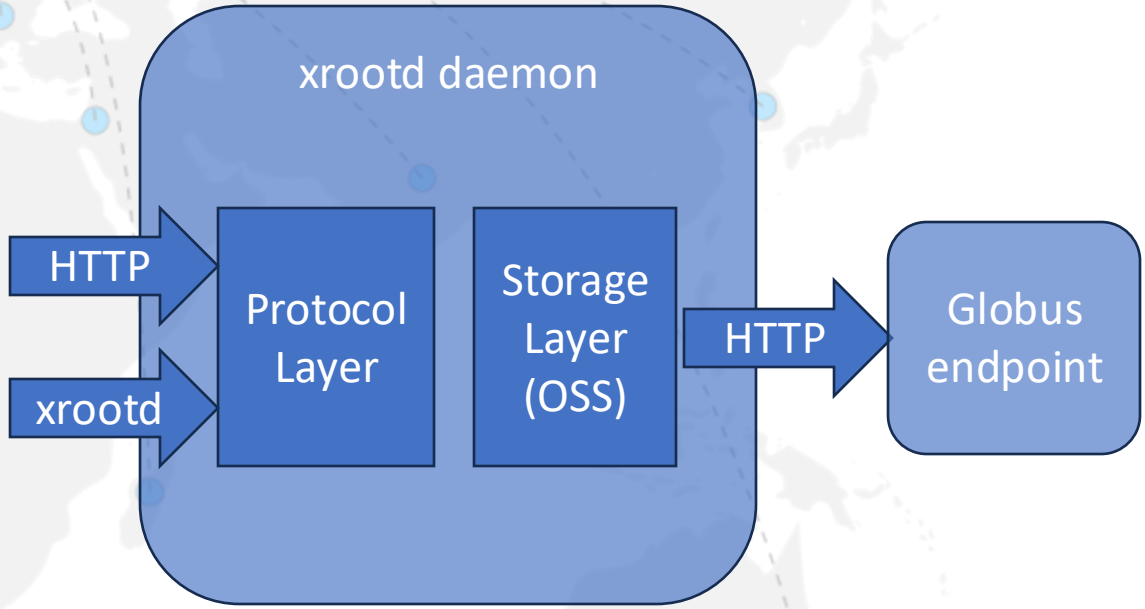
One more topic - Globus

- One common question we get from US universities: why should we use Pelican when we already have Globus?
  - A bit of a misnomer: Pelican & Globus do different, but complimentary things.
  - OTOH, we have lots of empathy for small institutions: it's a big "ask" to learn or run anything new!
- **Idea**: if the site has a Globus DTN, can we use Globus's new HTTP-based API as a backend?



One more topic - Globus

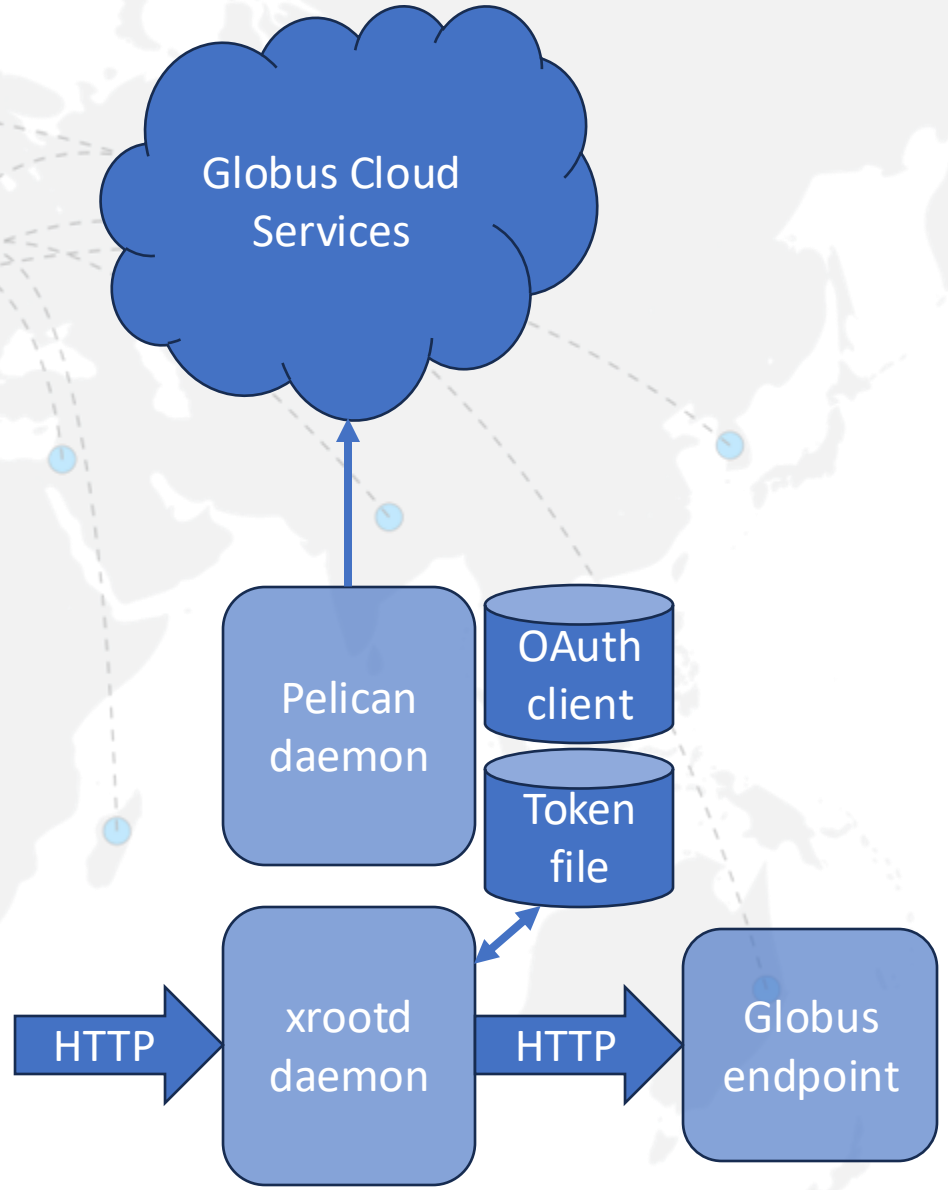
- Globus's "bread and butter" is transferring files between two Globus endpoints.
  - Proprietary protocol (GridFTP-ish), no guarantee of version stability.
  - Historically, no such thing as "downloading" from a Globus endpoint – closed system.
- Recently, Globus added HTTP functionality and a corresponding API.
  - Can even do "curl" if you'd like!



Globus Integration

- To contact a Globus endpoint, you need a valid Globus token.
  - Globus uses traditional OAuth2 flows to hand tokens to web applications.
  - **Idea**: The Pelican daemon exports a web interface – use that as the OAuth2 client!
- We then use libXrdHttpServer.so to communicate with Globus.
  - No Globus-specific code!

httpserver.token_file /tmp/foo



Globus Collections and Authentication

Pelican will:

- (One-time) Request user to perform an OAuth2 flow with Globus, approving the origin's access to the configured collection.
  - Pelican receives refresh and access token, writes it to disk.
- (Periodically) Pelican runs refresh flow to get a new access token, writes it to disk.
- (Per-request) libXrdHttpServer loads token from disk, adds it to the Authorization header of the HTTP request.

Like S3, the Globus auth'z and XRootD auth'z is decoupled.

Pelican configuration YAML snippet:

```
Origin:
  - GlobusCollectionID: ffc70472-e145-49e6-a41a-f7c695d31a0b
  - GlobusCollectionName: Human-Friendly Name
  - GlobusClientIDFile: /etc/pelican/globus-client
  - GlobusClientSecretFile: /etc/pelican/globus-secret
```

Globus – What Works, What Doesn’t

Works:

- Read-only file operations.
- ‘Stat’ files

Doesn’t:

- Writes (easy to fix, shared with HTTP).
- Directory listing (will need to spawn Globus-specific code!).

Vision: Re-exporting Globus into the WLCG ecosystem

WLCG has historically struggled with HPC sites whose only data movement option is Globus.

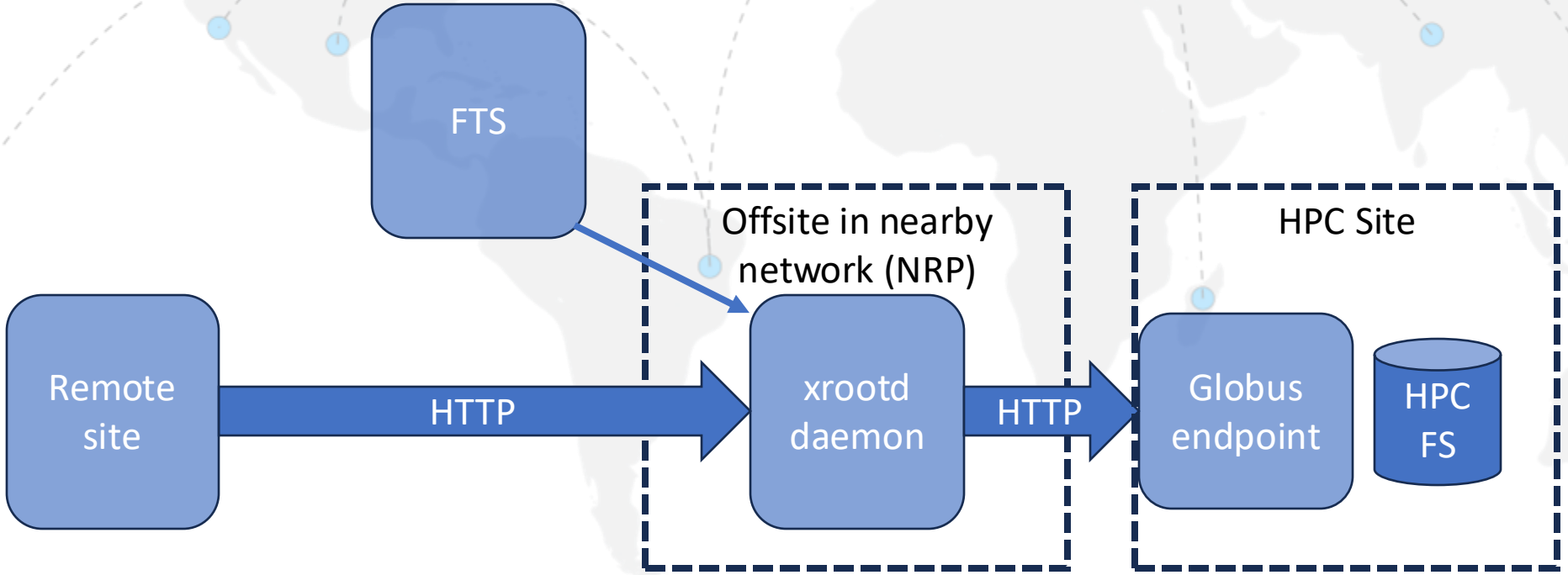
- One approach is to add Globus support to the WLCG data management stack.
  - Has been hard to maintain – need to plug in to several software stacks.
- Another approach is to ask the HPC site to run XRootD.
  - Also hard to maintain: HPC site sees this as alien technology, hard to motivate them to keep it alive.

Vision: Re-exporting Globus into the WLCG ecosystem

Idea: Run XRootD an proxy “nearby” the site.

No presence needed at the HPC site, they (theoretically) don’t need to be aware that XRootD exists.

No changes to WLCG DM tooling, no changes to the HPC site.



Think of what exists as a “tech preview” – will need help finishing it off!

Conclusions

- For Pelican, it's important to integrate as many data repositories as possible.
  - This has led us into developing a series of HTTP-esque OSS plugins.
  - Often, we run the origin 'nearby' on behalf of the site by using the NRP's Kubernetes cluster.
- The S3 backend is seeing production usage.
- The Globus backend is more "tech preview" but potentially has large impact in extending Pelican's reach.
- Next up? More specialized data repositories and their APIs, such as the **DataVerse software**.

Questions?

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2331480. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.