Booster rockets for Birds: New Backends for the Pelican Origin

Justin Hiemstra

CHTC Research Software Engineer

What is Pelican Here to Accomplish?

Pelican's goals are to:

- Enable access to data wherever it is needed, regardless of where it comes from — without having to learn multiple backend technologies. This access could take place in a Jupyter notebook, a campus cluster, or from national-scale computing infrastructure like the OSPool ↗.
- Support Open Science initiatives by supporting inter-disciplinary data sharing
- Encourage and support FAIR ↗ data practices
- Allow computing providers to stage data on-site as it's needed

Pelican enables researchers to:

- Integrate their data with national-scale computing infrastructure, with a focus on easy setup, distributed data caching and object delivery efficiency
- Make their data accessible to a broad range of users while maintaining control over how their data is accessed and by whom
- Coalesce disparate data repositories like S3, Globus and Posix under a common namespace

What is Pelican Here to Accomplish?

Pelican's goals are to:

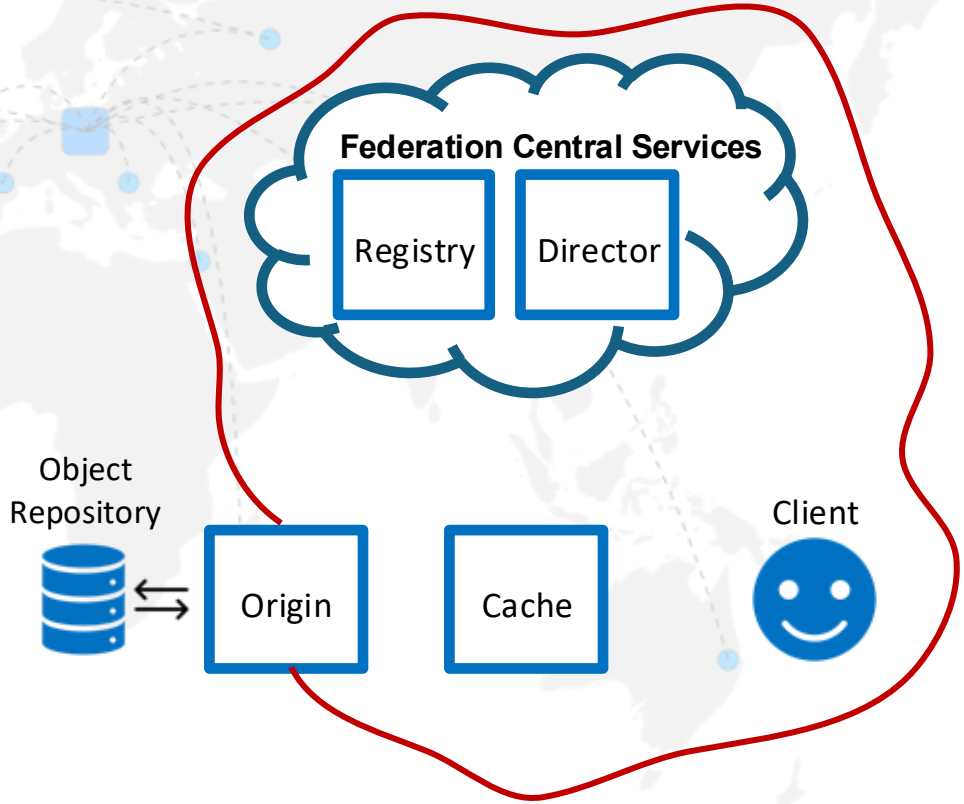
- Enable access to data wherever it is needed, regardless of where it comes from — without having to learn multiple backend technologies. This access could take place in a Jupyter notebook, a campus cluster, or from national-scale computing infrastructure like the OSPool ↗.

Pelican enables researchers to:

- Coalesce disparate data repositories like S3, Globus and Posix under a common namespace

Origins

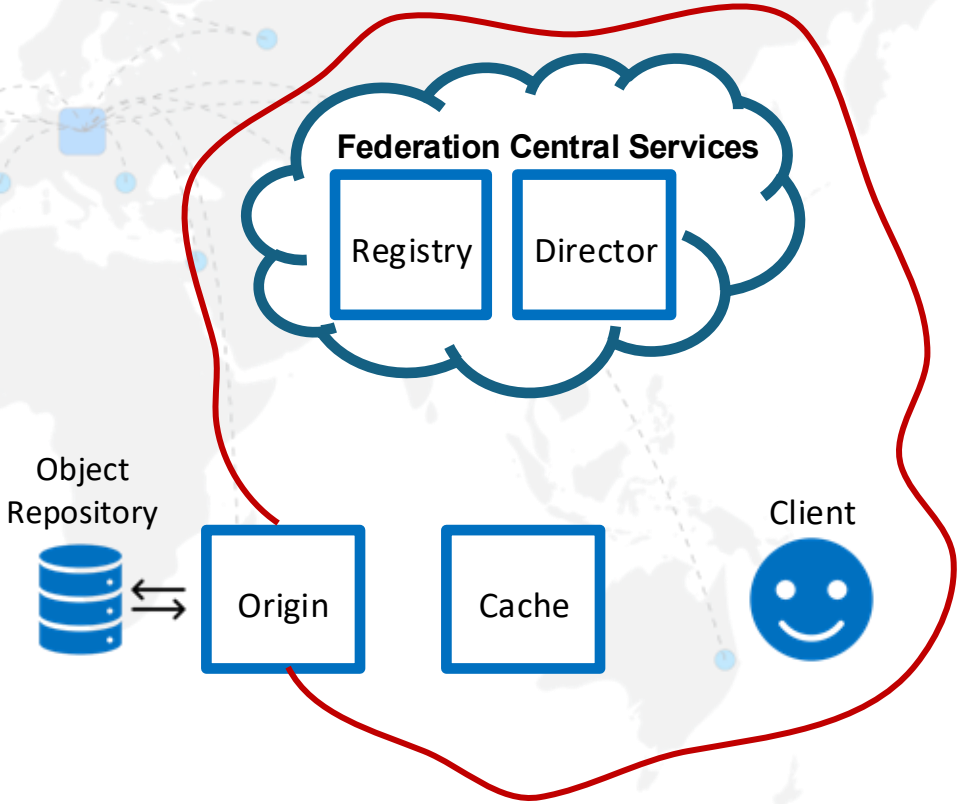
Origins play a loaded role:



Origins

Origins play a loaded role:

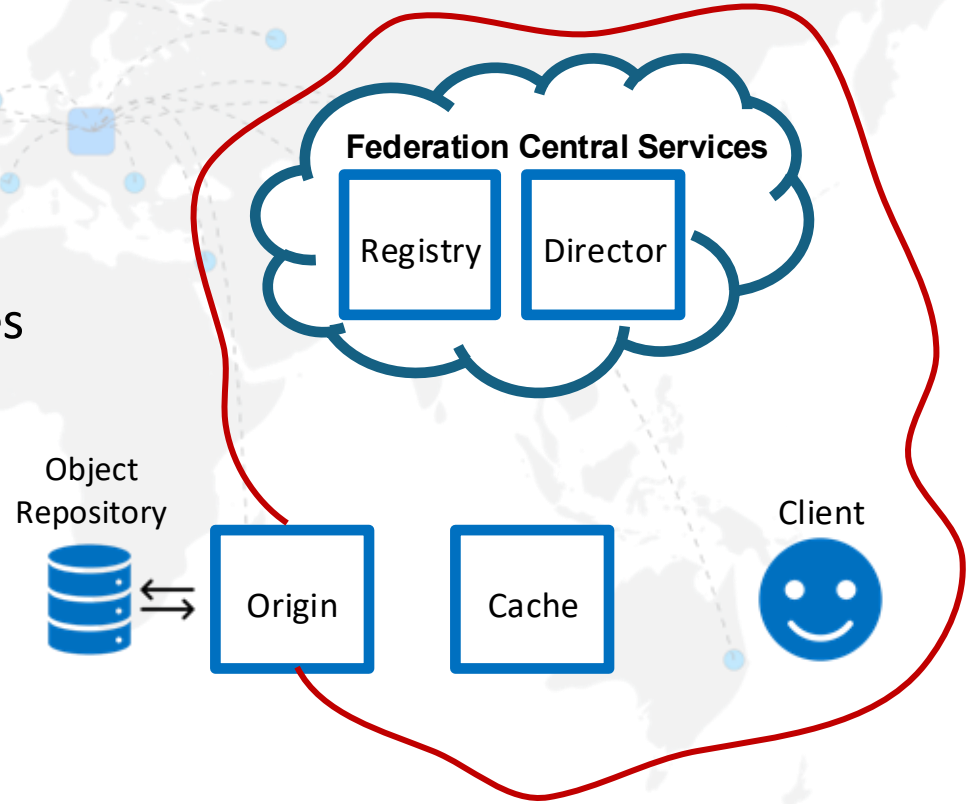
- "Translation" between object repository and federation



Origins

Origins play a loaded role:

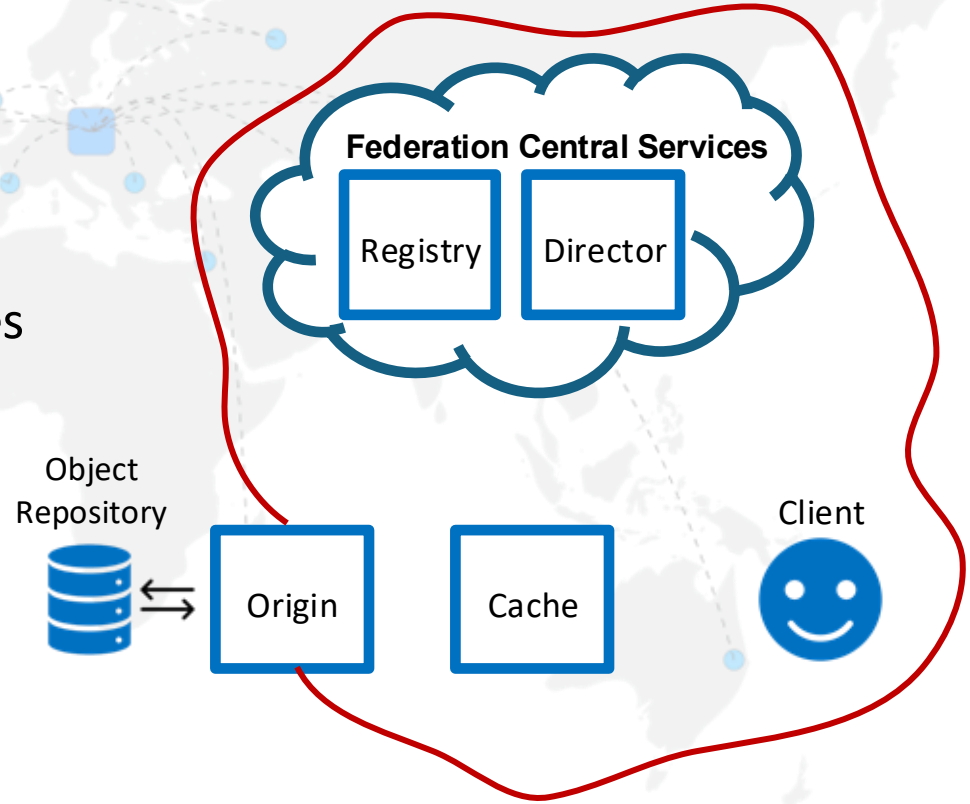
- "Translation" between object repository and federation
- Implementation of access rules (authz, throttling, etc)



Origins

Origins play a loaded role:

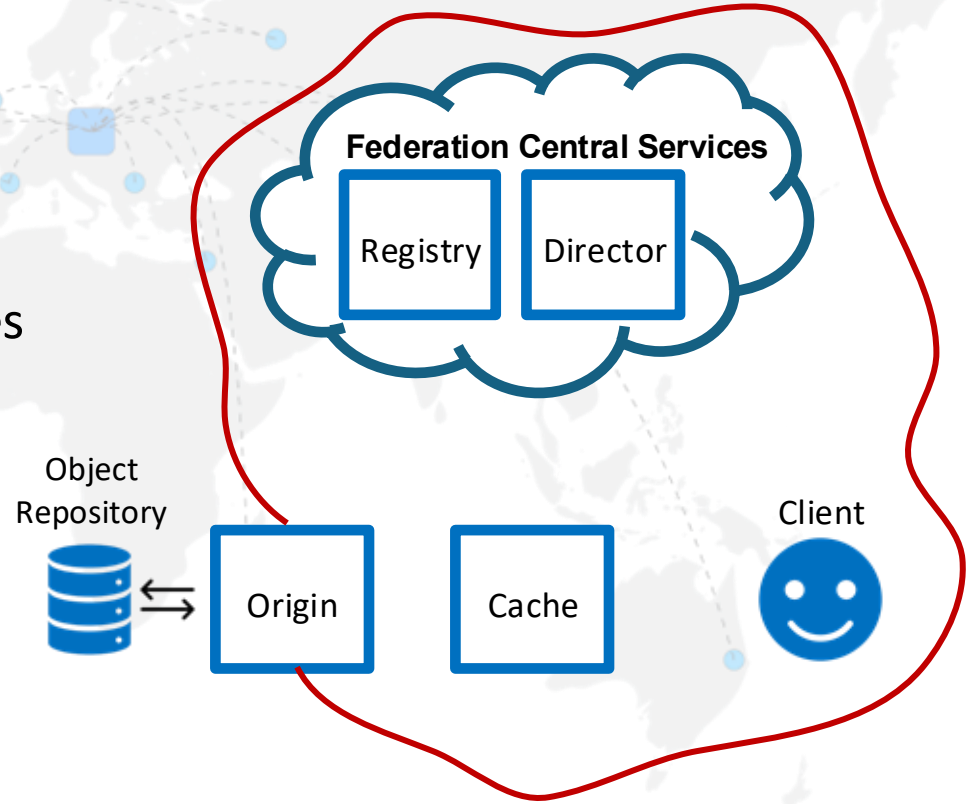
- "Translation" between object repository and federation
- Implementation of access rules (authz, throttling, etc)
- Circuit breaker duties



Origins

Origins play a loaded role:

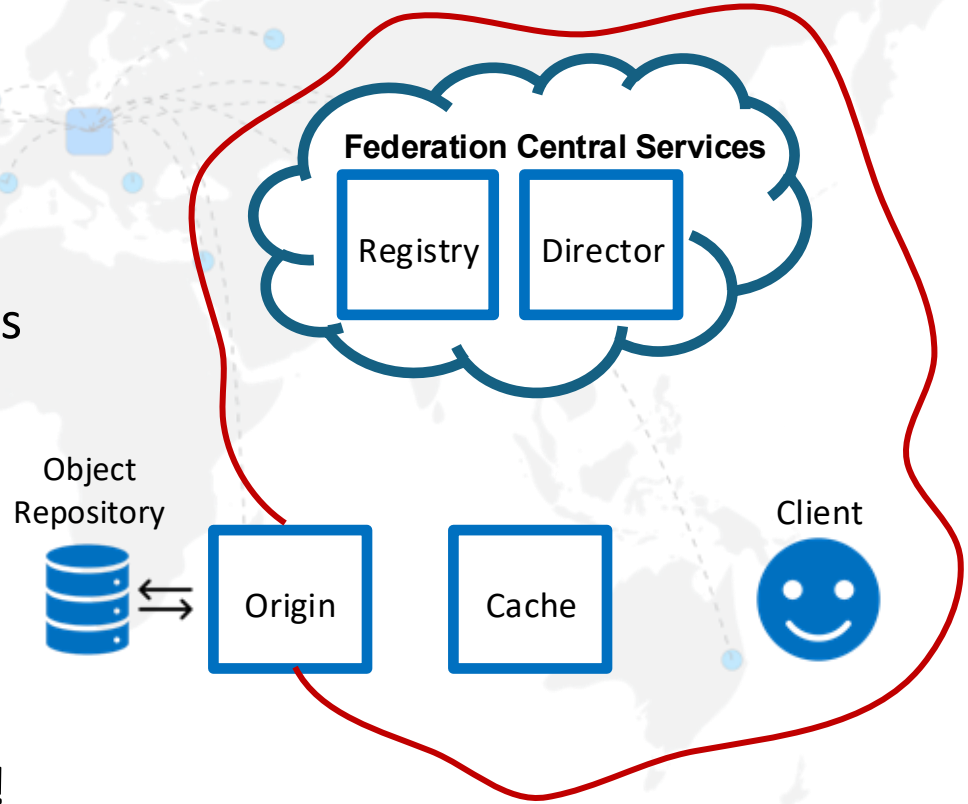
- "Translation" between object repository and federation
- Implementation of access rules (authz, throttling, etc)
- Circuit breaker duties
- Origin ↔ object repository "health" monitoring



Origins

Origins play a loaded role:

- "Translation" between object repository and federation
- Implementation of access rules (authz, throttling, etc)
- Circuit breaker duties
- Origin ↔ object repository "health" monitoring

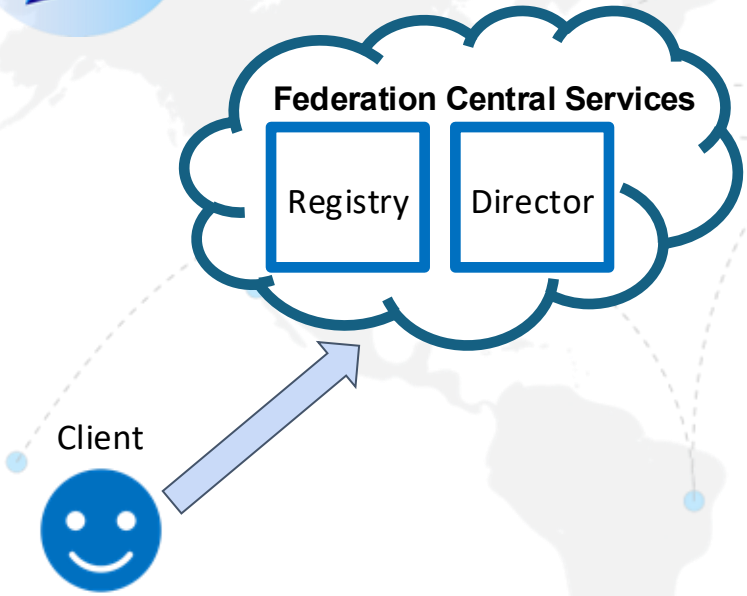


Every one of these may be unique to the particular object repository!

Origins as Translators


```
$ pelican object get \
    pelican://osg-htc.org/some/namespace/dog-photo.jpg \
    dog-photo.jpg
```

Origins as Translators




```
$ pelican object get \
    pelican://osg-htc.org/some/namespace/dog-photo.jpg \
    dog-photo.jpg
```

Origins as Translators

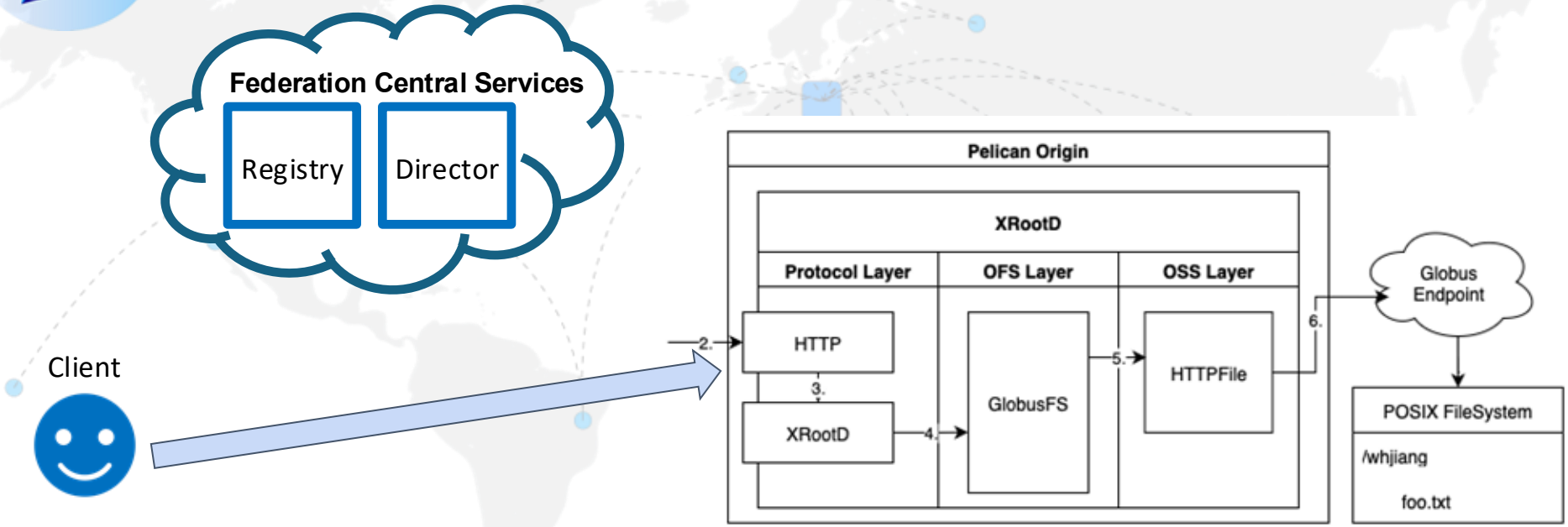


Diagram courtesy of CHTC Fellow alumnus William Jiang

```
$ pelican object get \
    pelican://osg-htc.org/some/namespace/dog-photo.jpg \
    dog-photo.jpg
```

Protocol Translations

Pelican verb	HTTP (client→Origin)	XRootD Internal	posix backend	https backend	s3 backend	globus backend	xroot backend
get	HTTP GET (ranged)	open + read x N + close	open + pread	HEAD + GET (ranged)	HeadObject + GetObject (ranged)	HEAD + GET (ranged)	kXR_open + kXR_read
put	HTTP PUT	open + write x N + close	open + pwrite	PUT (chunked)	CreateMPU + UploadPart x N + CompleteMPU	PUT (chunked)	kXR_open + kXR_write
stat	WebDAV PROPFIND depth=0	stat	stat(2)	HEAD	HeadObject (+ListObjectsV2 for dirs)	Transfer API stat	kXR_stat
ls	WebDAV PROPFIND depth=1	opendir + readdir x N	opendir + readdir	GET	ListObjectsV2 (paginated)	Transfer API ls	kXR_dirlist

Abstraction solves all problems… right?

Abstractions are a *vital* tool in software engineering. So what's the problem?

Abstraction solves all problems… right?

Abstractions are a *vital* tool in software engineering. So what's the problem?

> "*There are more things in heaven and earth, [Justin], Than are dreamt of in your philosophy.*" – Hamlet

Abstraction solves all problems… right?

Abstractions are a *vital* tool in software engineering. So what's the problem?

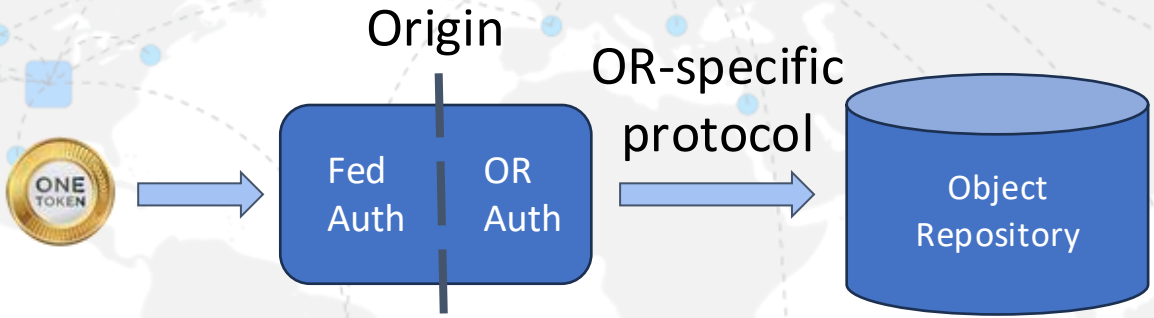
> "*There are more things in heaven and earth, [Justin], Than are dreamt of in your philosophy.*" – Hamlet

Differing object store and federation semantics lead to messy situations:

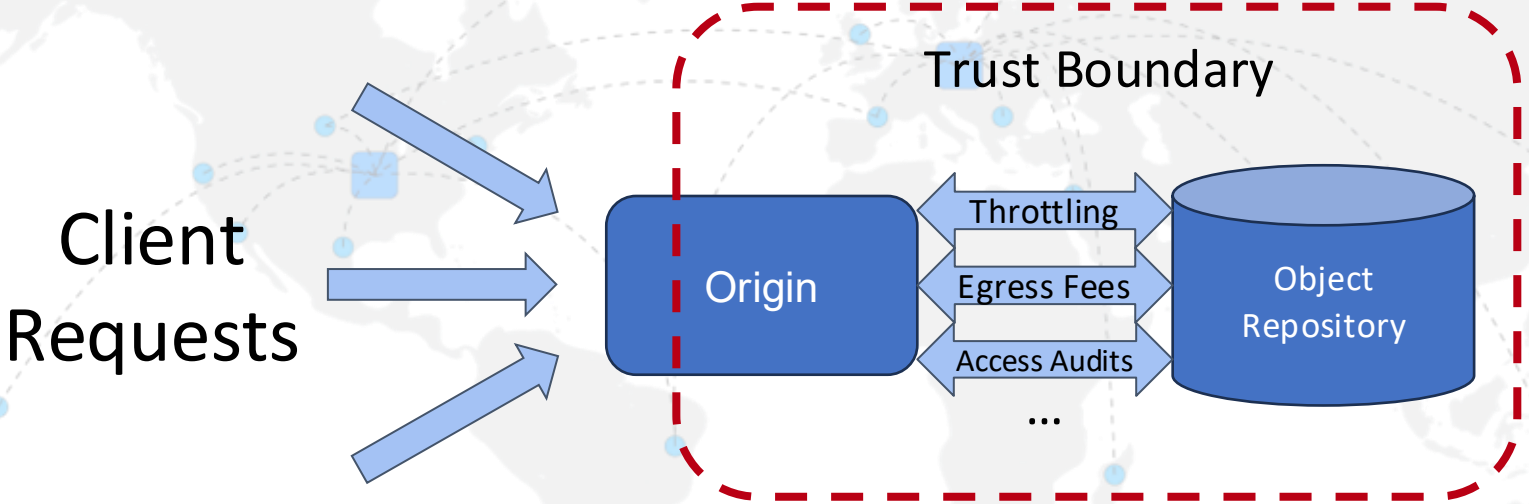
- Pelican's hierarchical namespaces vs S3's flat buckets
- POSIX partial writes vs Pelican's whole-object PUT model

Not everything can be mapped from one "philosophy" into another.

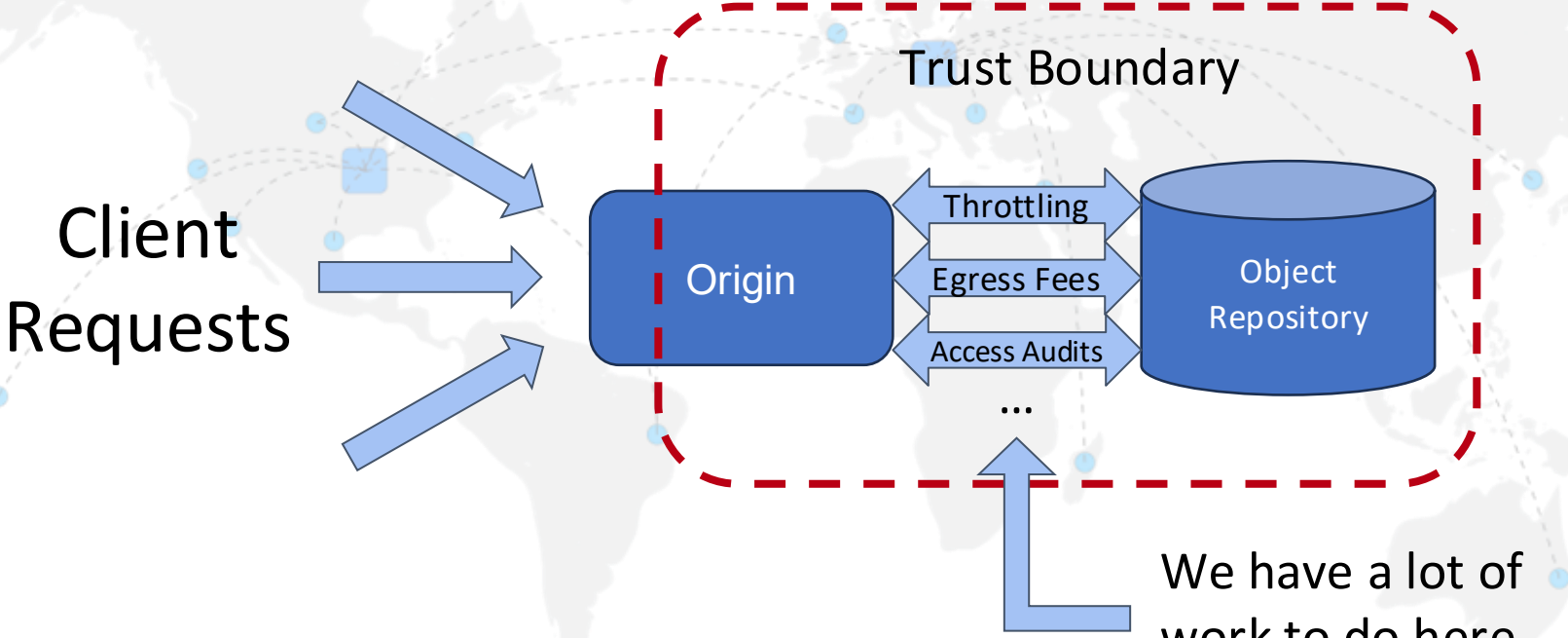
Origins must also translate authz



… and other Object Repository policies



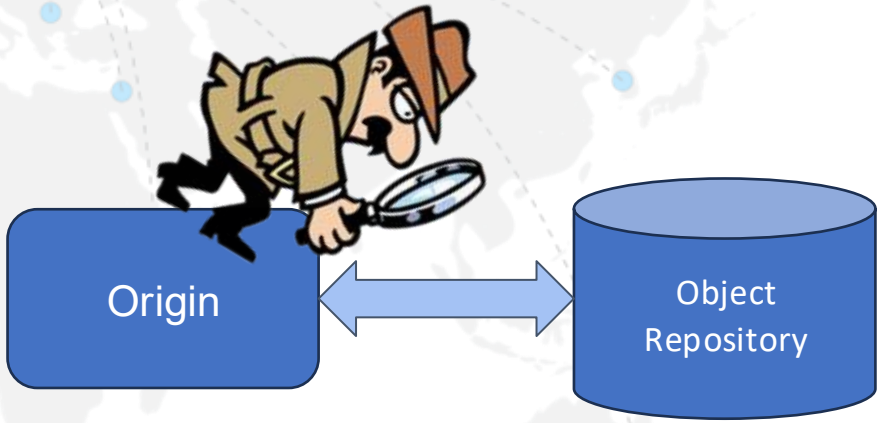
… and other Object Repository policies



Does it even work?

Origins provide a service to clients on behalf of data providers.

How can we tell if that service is functioning as "expected"?



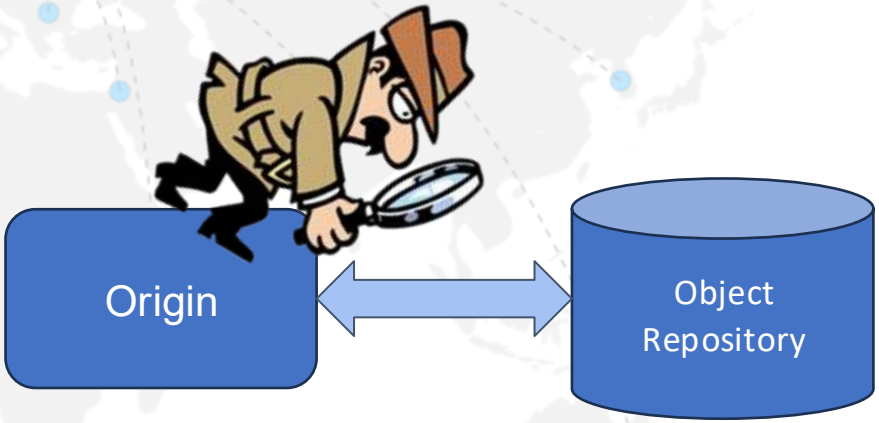
Does it even work?

Origins provide a service to clients on behalf of data providers.

How can we tell if that service is functioning as "expected"?

We use various assessments of "health" to probe this:

- Director ↔ Origin transfer tests
- Origin self tests
- I/O monitoring



Does it even work?

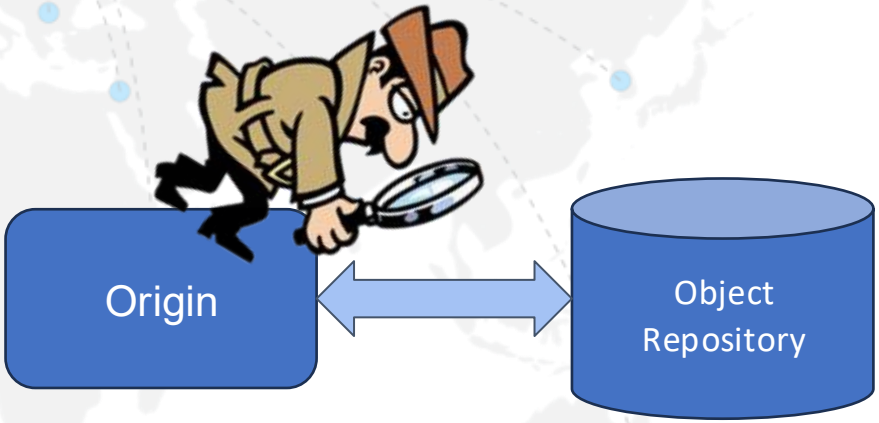
Origins provide a service to clients on behalf of data providers.

How can we tell if that service is functioning as "expected"?

We use various assessments of "health" to probe this:

- Director ↔ Origin transfer tests
- Origin self tests
- I/O monitoring

Testing/determining the "health" of this connection is backend dependent.



Polished & Upcoming Origin backends

Several backends are maturing from "reads work" to production-ready:

- **Globus, HTTPS**: writes, stats, listings were the hard part – now complete
- **xroot**: name remapping lets the federation namespace differ from the upstream XRootD layout

XRootD

Polished & Upcoming Origin backends

Several backends are maturing from "reads work" to production-ready:

- **Globus, HTTPS**: writes, stats, listings were the hard part – now complete
- **xroot**: name remapping lets the federation namespace differ from the upstream XRootD layout

We're also expanding reach to data that has no API at all:

- **ssh backend (WIP)**: most HPC data lives behind SSH and nothing else

XRootD

Polished & Upcoming Origin backends

Several backends are maturing from "reads work" to production-ready:

- **Globus, HTTPS**: writes, stats, listings were the hard part – now complete
- **xroot**: name remapping lets the federation namespace differ from the upstream XRootD layout

We're also expanding reach to data that has no API at all:

- **ssh backend (WIP)**: most HPC data lives behind SSH and nothing else

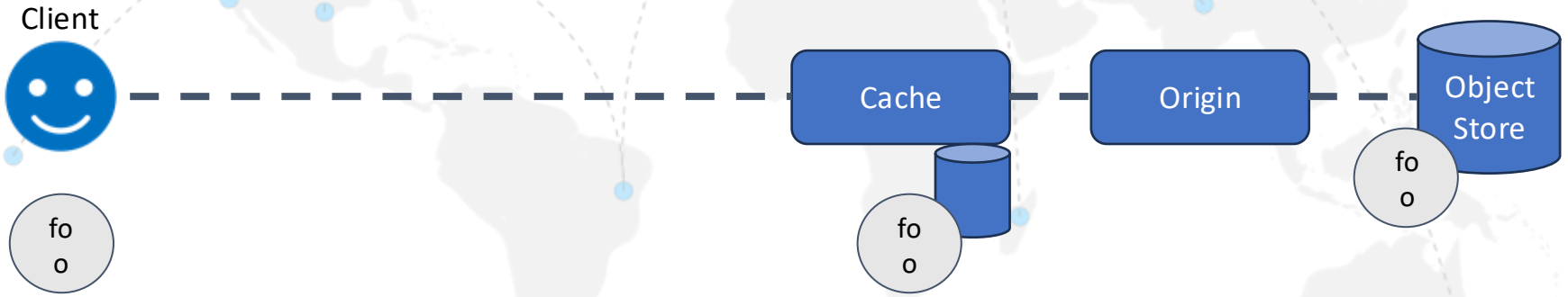
Rethinking the architecture with XRootD-less backends:

- **posixv2** is already running in production; lighter, simpler to deploy
- **globusv2**, **httpsv2** and **s3v2** coming soon!

XRootD

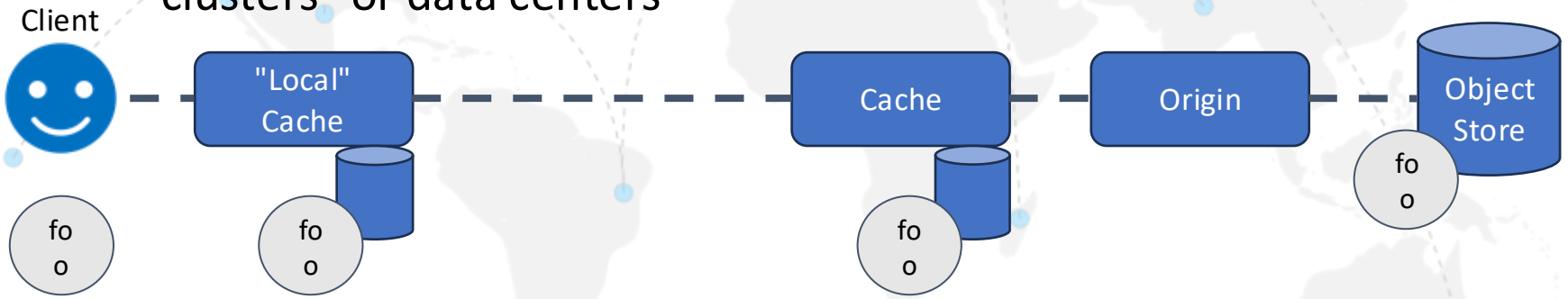
Caching everywhere!

The typical setup relies on federation-wide caches/staging devices



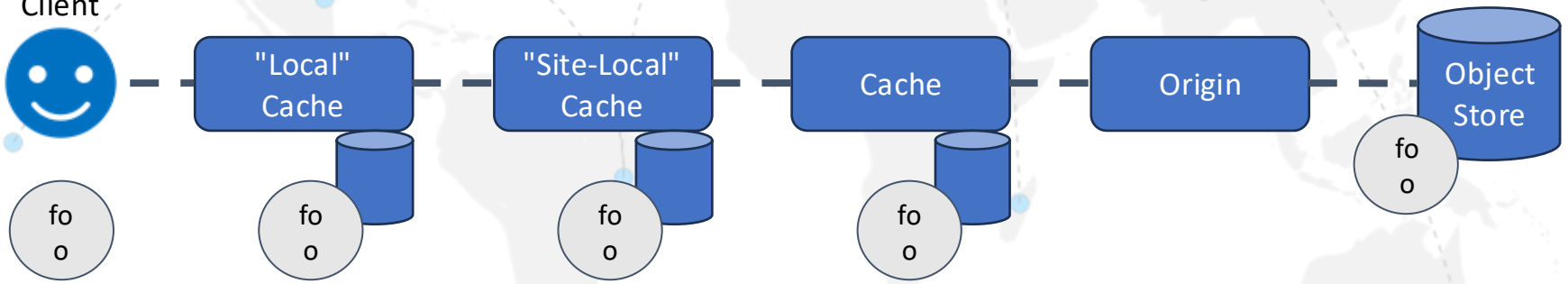
Caching everywhere!

A few years ago, we started working on "local" cache Pelican component that's meant to serve individual "clusters" or data centers



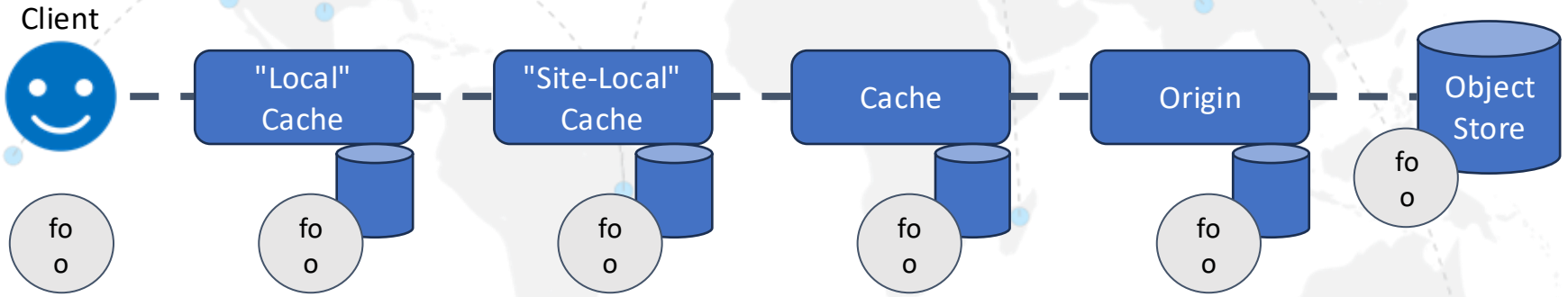
Caching everywhere!

In the last year, we introduced first-class "institutional" or "site-local" caches that aren't open to all federation users.



Caching everywhere!

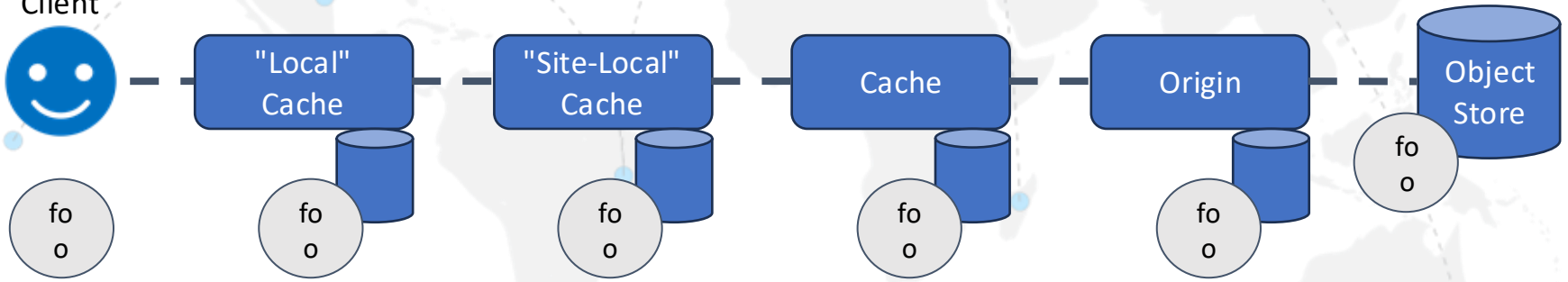
We're also now exploring Origins with their own caching capabilities.



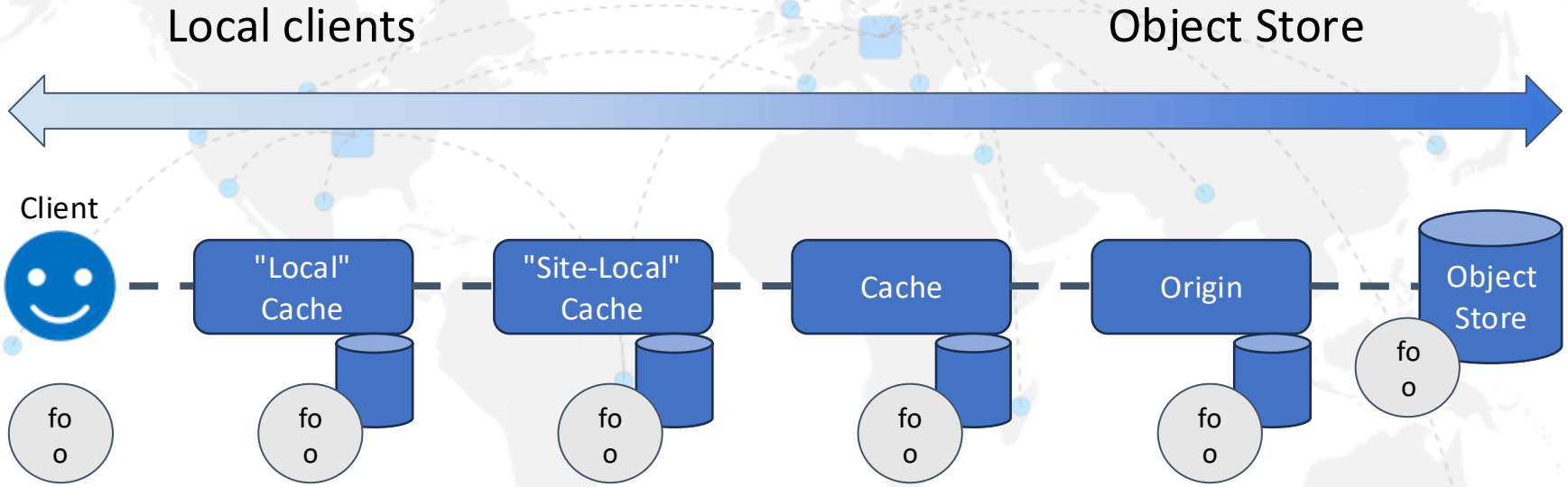
Caching everywhere!

Why do this?

Whom are these caches meant to serve?



Caching everywhere!




Questions?

Acknowledgements

This material is based upon work supported by the National Science Foundation under Cooperative Agreements OAC-2209645. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.