From Data Federations to Data Lakes

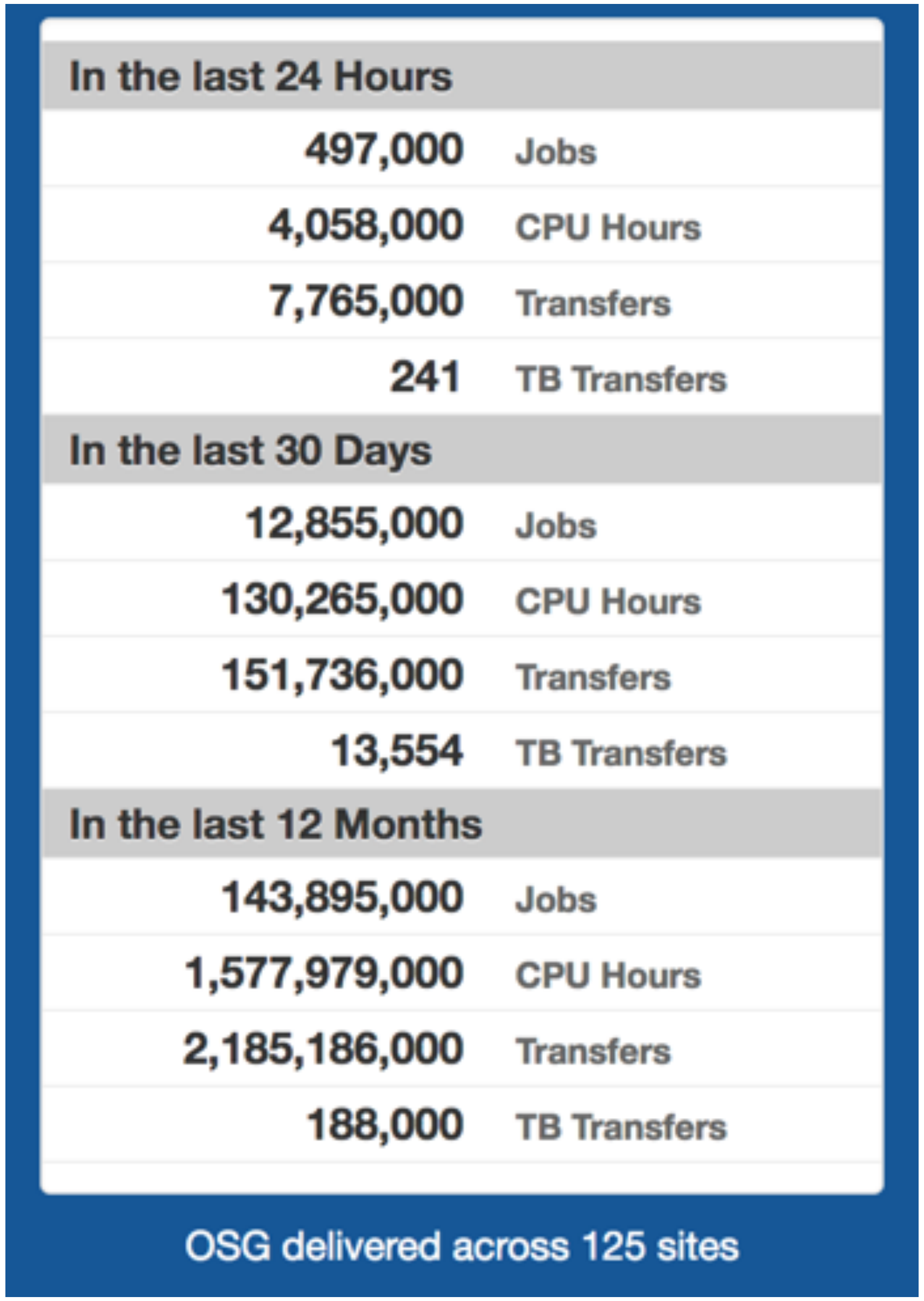
Evolving Data Access for HEP

Data Access in HEP

- HEP has historically pushed the boundaries of data volumes - starting from the days when “gigabytes were Big Data.”
    - Now looking down the barrel of exabytes.
- Fundamentally, HEP experiments are made up of large collaborations. LHC experiments are made up of universities and labs across the continents.
- (Distributed Collaborations) + (Challenging Storage Needs) = **Drive for Distributed Computing**

Distributed Computing in HEP

- The “atomic” unit of processing in HEP is the event.
- Multiple events can be processed independently, leading to a pleasantly parallel system.
    - E.g., given 1B events to simulate and 10 sites, one could request 100M events from each site.
- Easy, right?
    - Output data from simulations or the detector must be broken into manageable datasets.
    - Datasets must be made available for analysis and reprocessing.
    - Across the entire system, one must manage and balance resources (networking, storage, CPU, IOPS).
- Today’s fundamental entities of the computing system: jobs and files.
    - Tomorrow’s: event processors and object stores?
- Today’s fundamental tools: batch systems and **storage elements**.
    - Tomorrow’s: ??? Data Lakes?



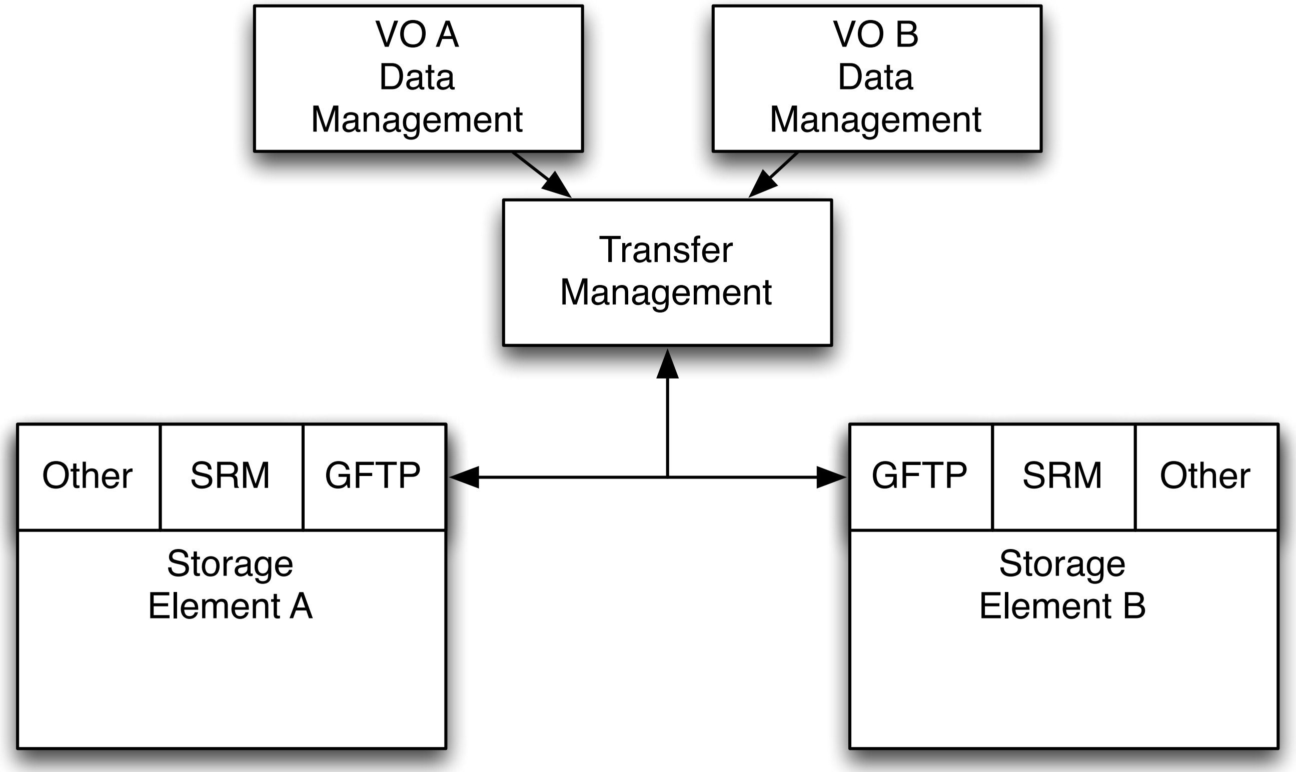
In the last 24 Hours	
497,000	Jobs
4,058,000	CPU Hours
7,765,000	Transfers
241	TB Transfers
In the last 30 Days	
12,855,000	Jobs
130,265,000	CPU Hours
151,736,000	Transfers
13,554	TB Transfers
In the last 12 Months	
143,895,000	Jobs
1,577,979,000	CPU Hours
2,185,186,000	Transfers
188,000	TB Transfers

OSG delivered across 125 sites

Review:
Storage Elements

Storage Element

- In this model, we have multiple services exposing a POSIX-like filesystem.
    - Each storage element acts independently.
- A higher-level transfer management layer moves files between SEs.
- VOs develop their own data management layer on top of that. Not quite so simple…
- Driving model for 15+ years!



(… plus 50 more sites!)

The Storage Element

- How is data managed in the SE paradigm?
    - *Access*: Each SE has its own twist on data access protocols. **Not always POSIX**!
        - Users access the data by finding the right endpoint and filename. **Think: pre-Google Internet**.
    - *Movement*: A service is given a set of files from endpoint A to endpoint B. The files are usable once files are at endpoint B.
    - *Catalogue*: Some central service tracks the location of each file.
        - Catalogs must be kept in sync for this to work!

See Rucio Talk!

-
    - *Data management*: Rules engine verifies that all files are in the “correct” location according to some set of rules. If not, make new copies with the movement service.
- Data lost? Site initiates a recovery procedure. In CMS, the site admin opens a ticket.
    - It is assumed this is an *exceptional event* which does not happen frequently.
    - If a file is not in the correct location, it can be considered an error.

Plan B?

- Motivation for change: **current model has high cost**.
  - The storage element model is a complex, manpower-intensive way to manage data. Requires significant sysadmin attention and central ops teams.
  - Current HEP implementations of this model have little overlap between experiments.
  - Hence, total cost to the field is bounded by
    $(BIG_NUMBER) x $(# of SITES) x $(# of experiments)

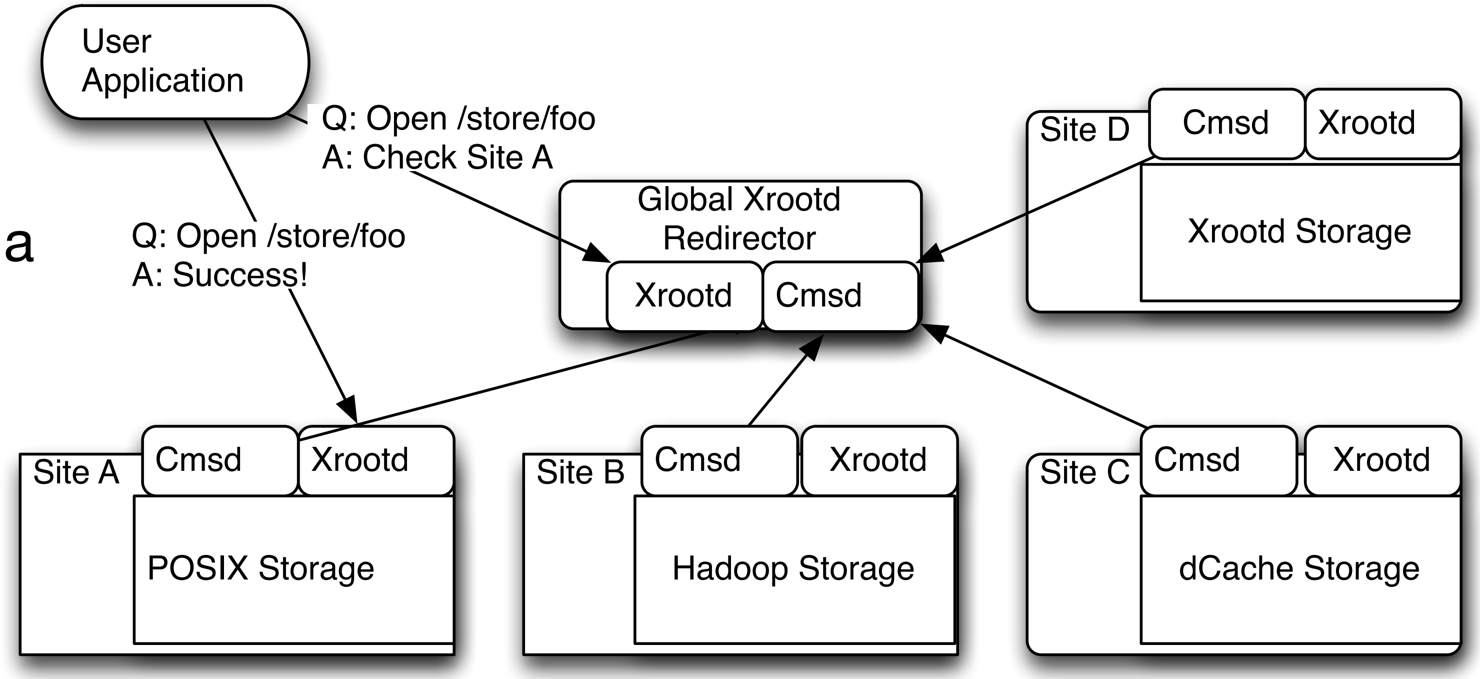
Insert Data Lakes Here

- How to reduce this bound?
  - Reduce number of sites that have their own storage element.
  - Reduce number of (unique) VO data management systems. **Sadly, we have a poor track record here!**
  - Alternate data management models.

Insert Data Federations Here

Data Federations

- Alternate **data access model**.
- **Idea**: Provide transparent access to a uniform namespace across multiple, independently administered storage elements.
    - User file requests are sent to a **redirector**, which broadcasts a location query across participating sites.
    - User is redirected to the best available replica. **Ideally, bytes start to stream in seconds**.
- As with the storage element model, sites make their own decisions about



Federations provide data access.

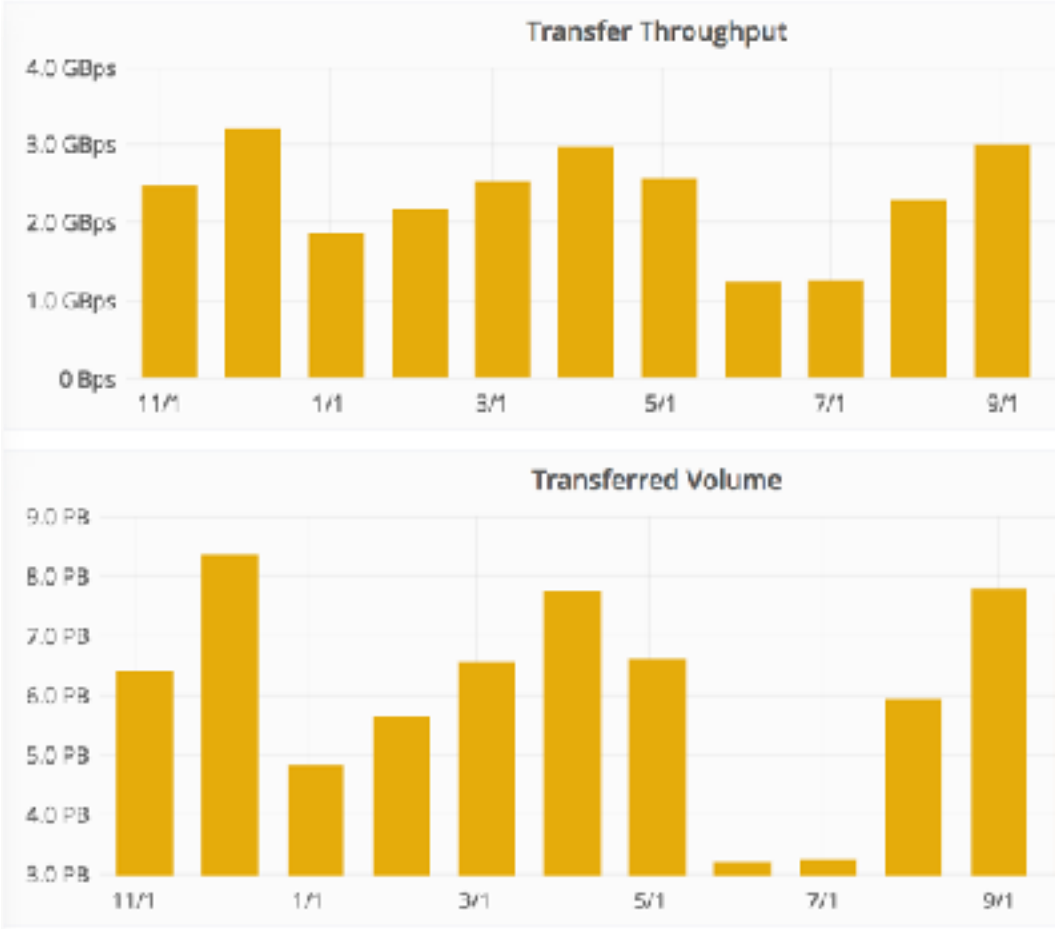
However, there are several things that they don't provide:

- **Namespaces**. No source of authority on what should be in the federation or its contents.
- **Data movement or replication**.

These are often added by combining the federation with other technologies for data management.

Federations Example: AAA

- The first federation I was involved with was the “Any Data, Anytime, Anywhere” (AAA) project in the CMS experiment.
- Uses the XRootD software suite to federate about 50 sites containing 65PB of data.
- Allows CMS users to access data immediately; previously, needed to submit a batch job to read out even 1 byte.
- With AAA, CMS can process petascale datasets across multiple sites via streaming.



In the last 6 months…

143PB

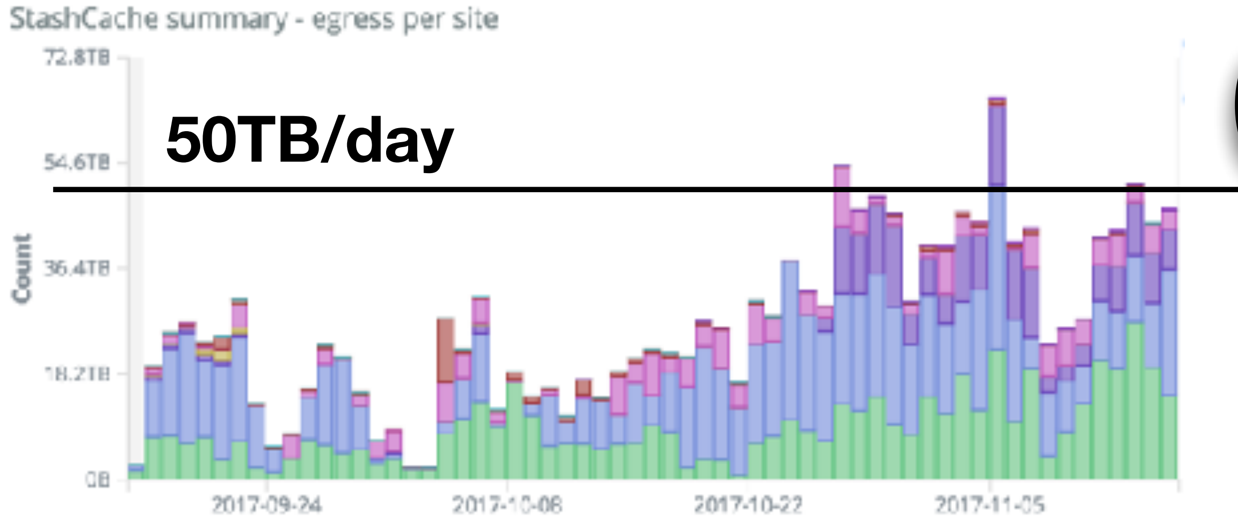
Read local to sites

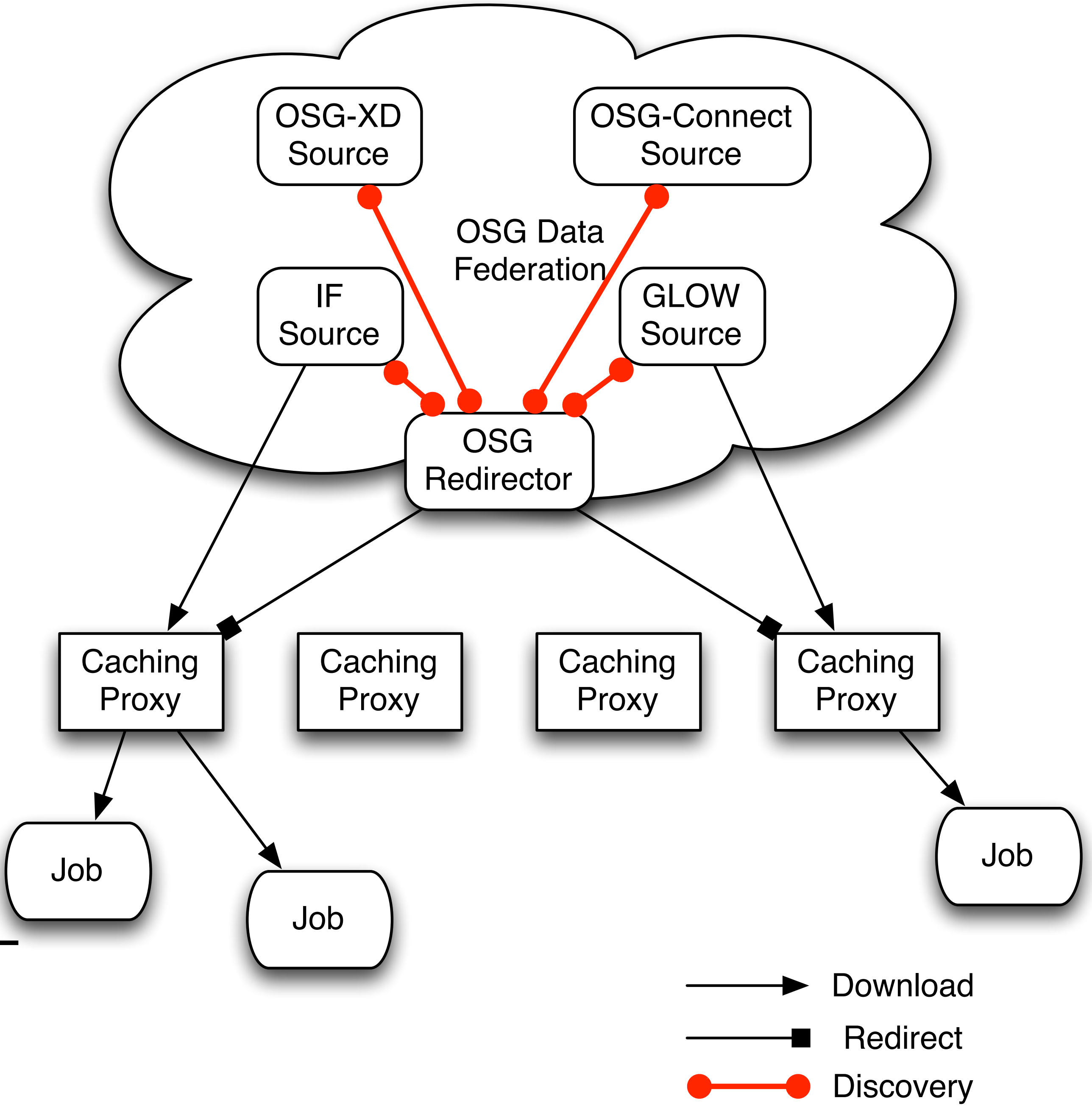
39PB

Read via data federation

Federation Example: StashCache

- Data federations do not need to be unique to one experiment.
- OSG's data federation combines several small sources and places them behind a layer of caches.
- Caching layer reduces load on the data federation.
- OSG users can access this via HTTPS directly or via a POSIX read-only interface.
- About 1.5PB in the last 2 months.





Data Lake Concept

- Instead of one-SE-per-site, have a single logical SE that encompasses a significant amount of high-performance storage.

- Sites outside a data lake have no persistent experiment storage.

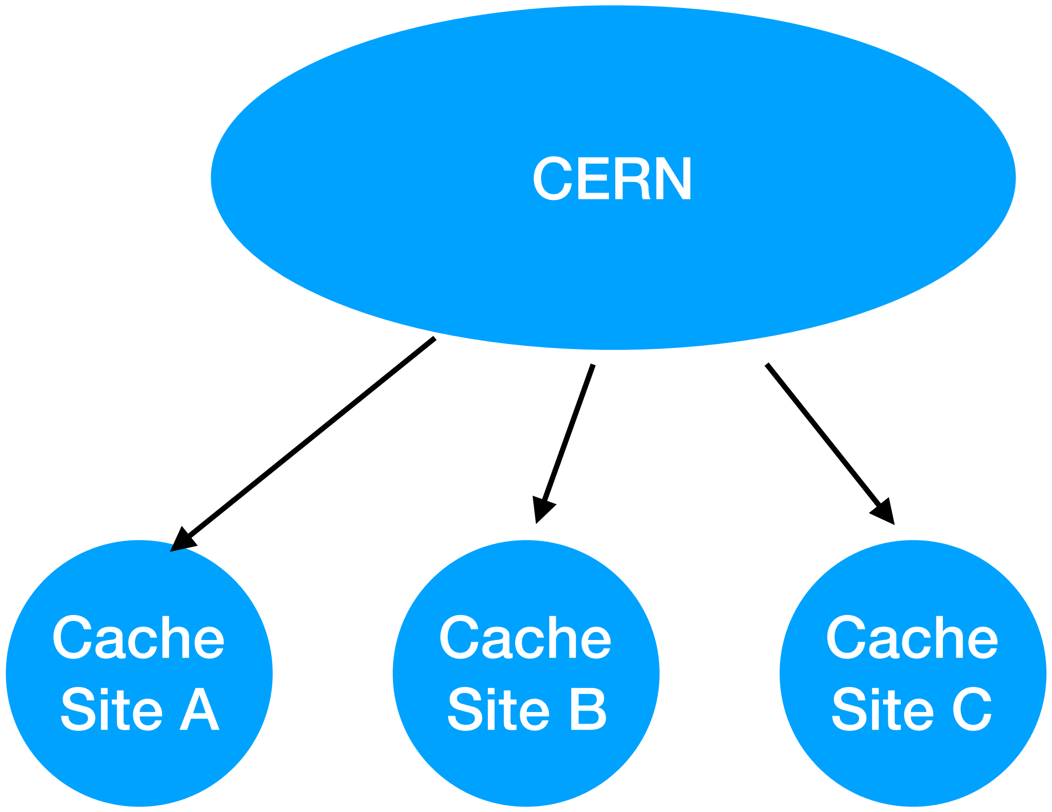
  - E.g., all is cached or streamed.

- Non-experiment private data (think: **user outputs**) goes directly to destination site user is associated with.

  - Regardless of whether the site is in-or-out of the lake.

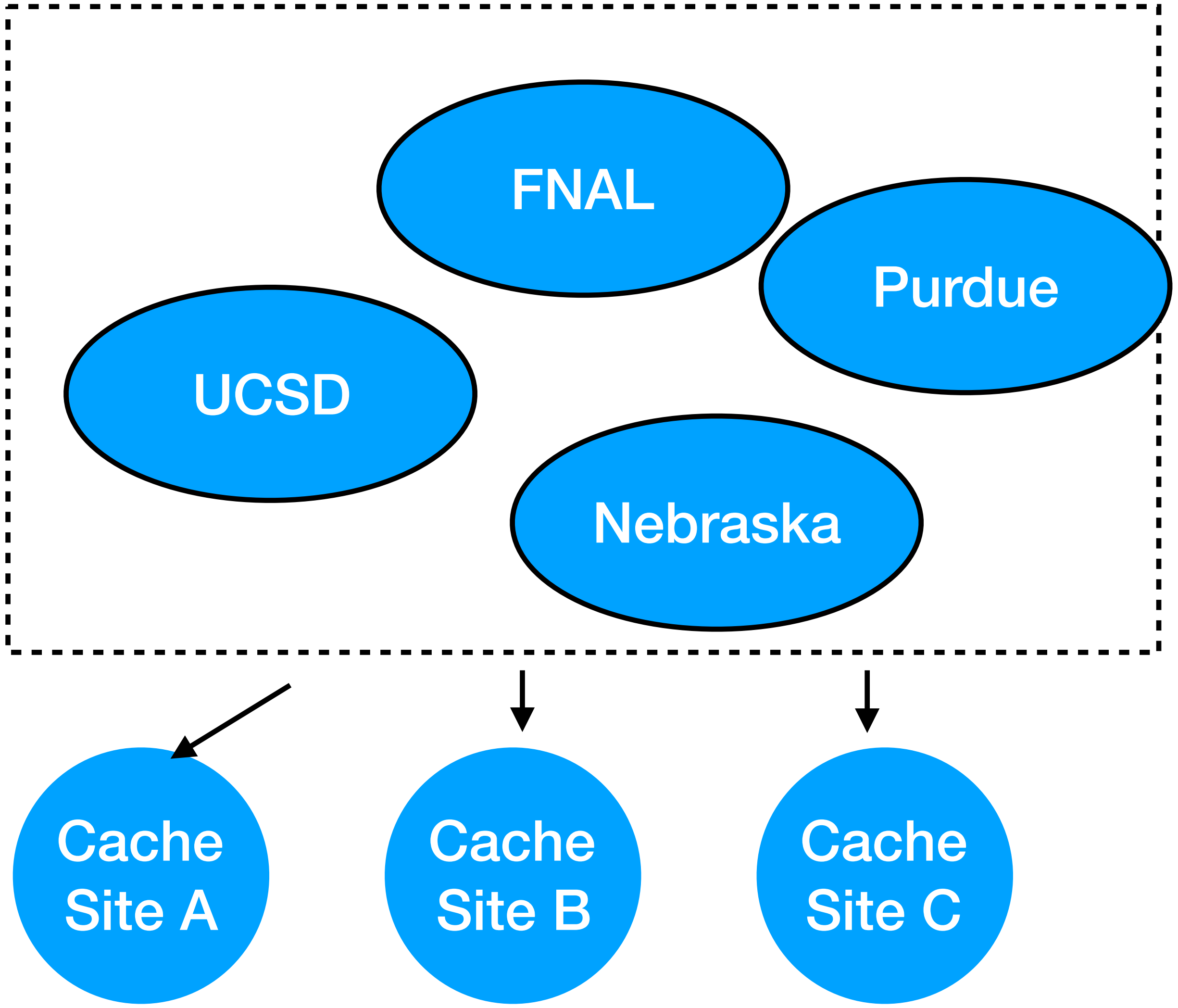
Data Lake Example 1

- All data lives at a single, large site (e.g., host lab).
- Data is streamed on-demand to all sites that support the experiment.
- All user and official data goes back to host lab.



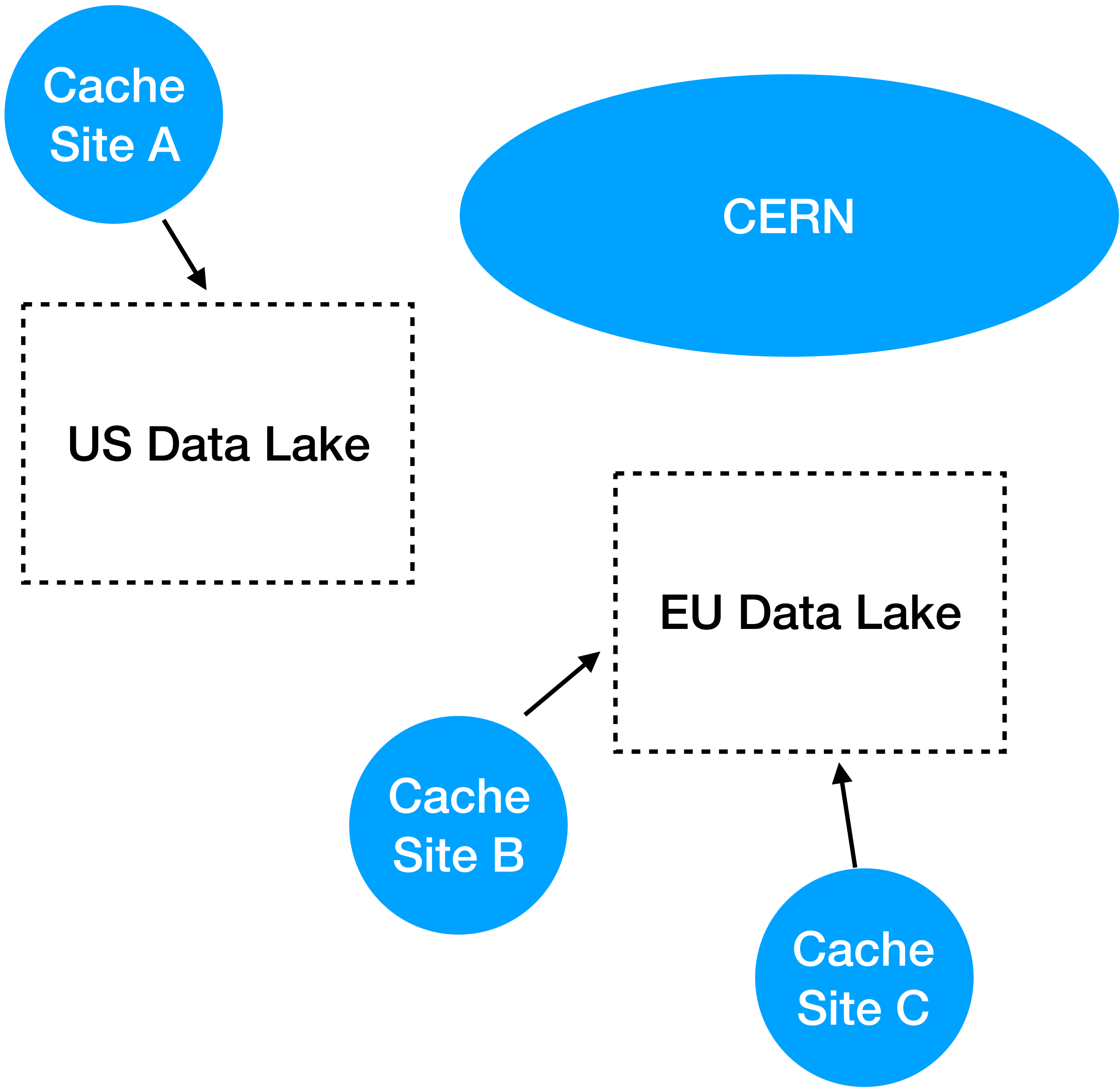
Data Lake Example 2

- Several large sites with significant intra-connectivity form a single logical storage element from **heterogeneous storage systems**.
- Experiment places data into the lake, but has no/little control over where that data is physically stored.
  - Official data *only* persists in the lake.
  - User data can live at any individual site.
- As in “data lake example 1”, experiment’s data management system only has to deal with a single SE.
- This **looks a lot like a data federation**! Data access is very similar, but I think a data lake critically needs a concept of a namespace and file integrity.



Data Lake Example 3

- Large experiment (HL-LHC) has too much data for a single lake; a small number of these exist.
- Experiment data system only deals with a small number of well-run endpoints.
- Caching / processing sites attach to a single lake.



Perceived Benefits

- Small-to-medium sites need no storage element at all.
  - Simply a cache and network connectivity!
  - **Note**: In most discussions, “small-to-medium” sites seem to be those <1PB, sometimes <200TB.
- Data management systems become simplified if they only need to manage 3-5 sites — even at “HL-LHC scale”!
  - Experiments are exposed to less risk in this area!

Federations and Lakes: Outlooks and the Path Forward

- Data federations are an important data management paradigm that emphasizes *transparent access* and *simplicity*.
    - Democratizes the data access: both user laptops and large-scale workflows can access it equally well.
    - For a large experiment like CMS, they complement the existing data management systems and allow new use cases.
- Data lakes are a new concept, aiming to reduce the cost of running distinct storage elements.
    - At the simplest, a data lake could be a single site exporting to a series of caches.
    - Activity in this space is just starting! Keep an eye out…
- Both aim to reduce the cost of distributing data across the distributed scientific computing infrastructures common to HEP.