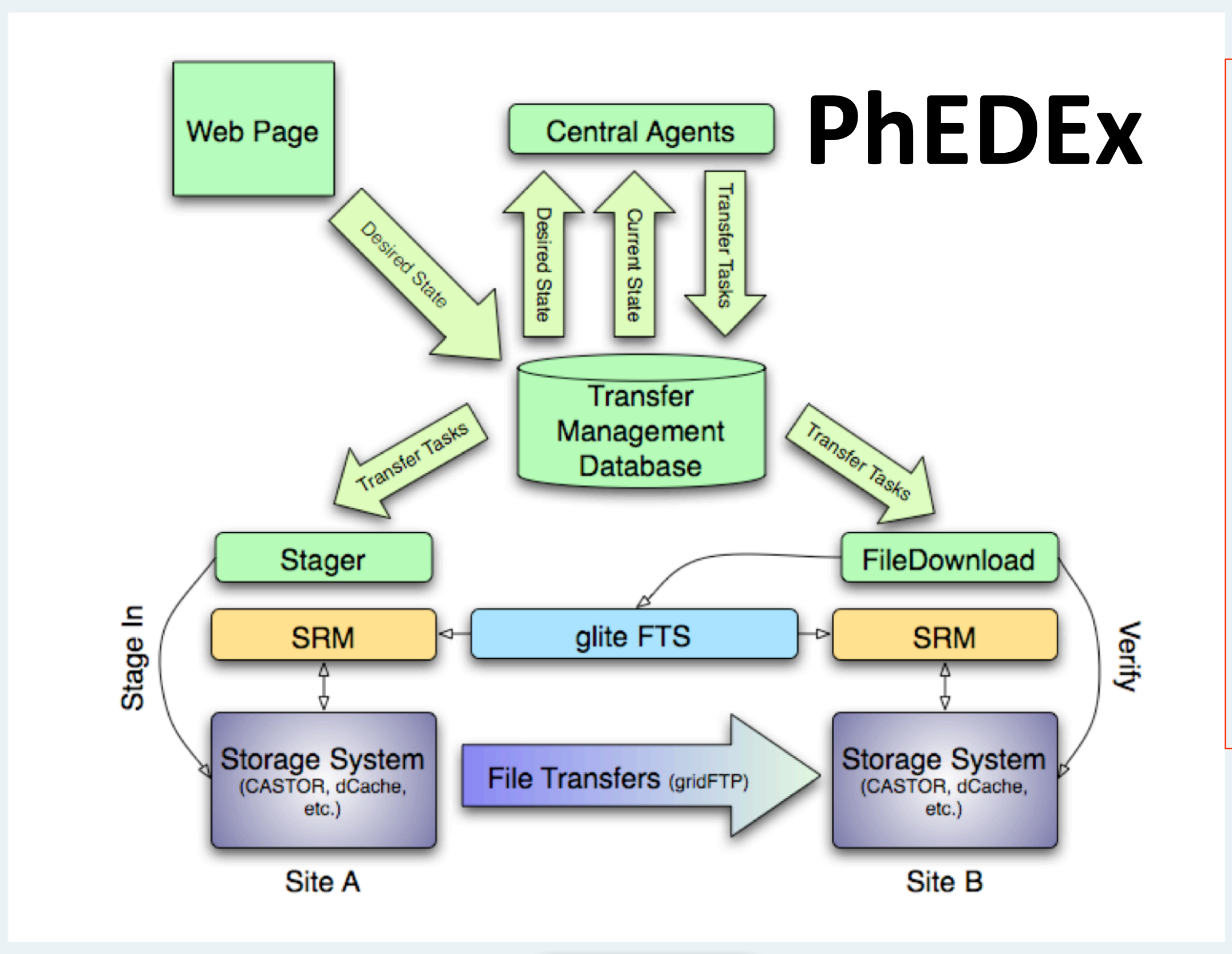
Dimensions of Data Management: a taxonomy of data-transfer solutions in ATLAS & CMS

Tony Wildish [1], Campana Simone [2], Garonne Vincent [2], Lassnig Mario [2], Di Girolamo Alessandro [2], Serfon Cedric [2]
[1] Princeton University, USA [2] CERN, Switzerland

The Challenge

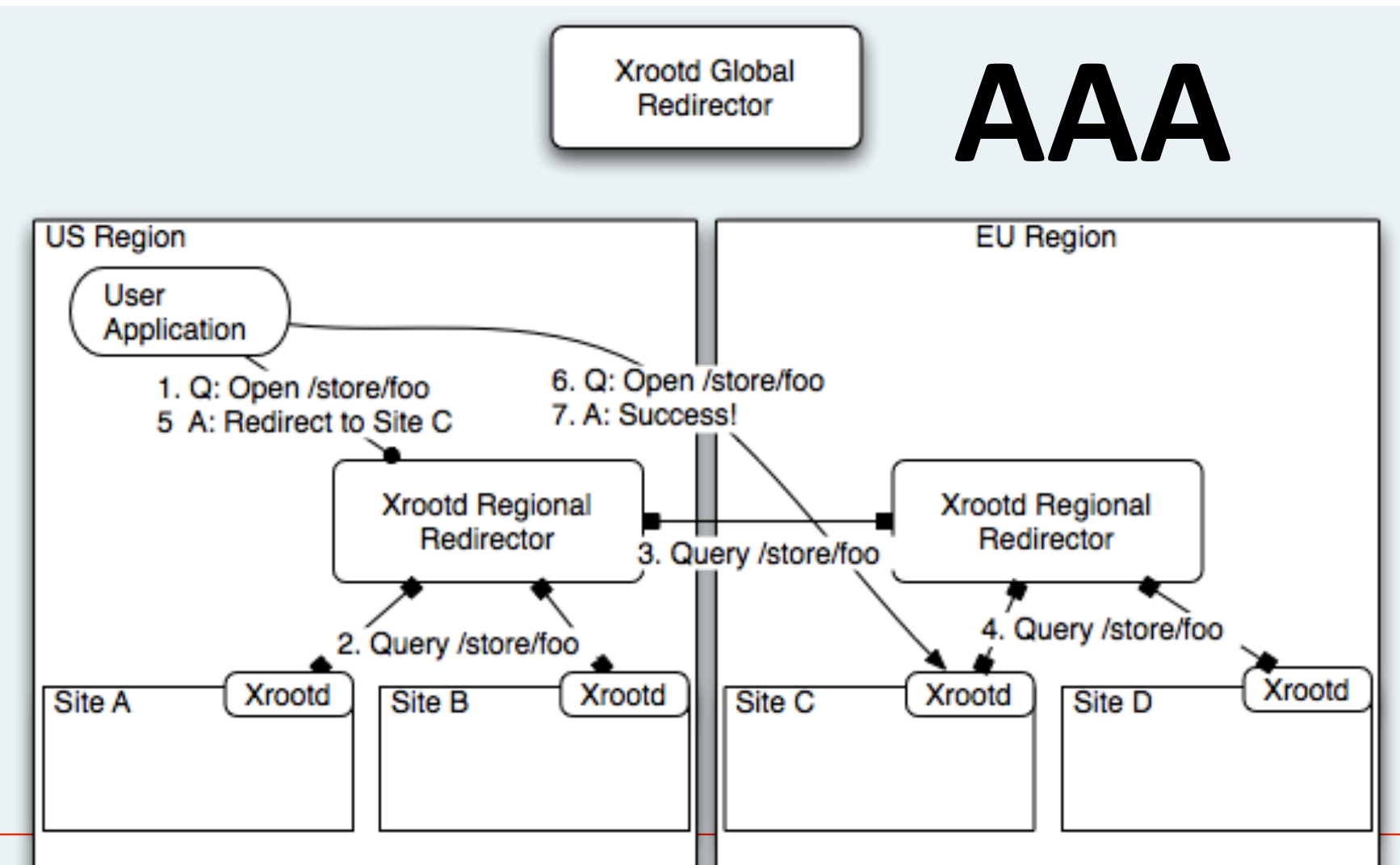
HEP software is traditionally designed to satisfy certain use-cases or requirements, either based on a-priori estimates of the experiments' needs or on the shortcomings of an existing solution which no longer performs satisfactorily. However, designing against a requirements-document is only useful if the requirements are complete and stable.
Even if we could write down a complete set of requirements for data management systems for CMS and ATLAS, we cannot pretend that they will be stable over the course of several years.
Experiments' data-models change, types and distributions of resources change, and the patterns of data-use also change. New technologies can disrupt the way experiments use data, such as the move to multi-core CPUs or cloud-based infrastructure. We should not be surprised if we discover that data management use-cases arise during Run-2 that we are not currently aware of, and that our systems may struggle to cope with.
We propose a framework for designing data management systems that takes a step back from the normal requirements-driven process and looks instead at data and the ways it can be manipulated.
By identifying a set of 'dimensions of data management' we provide a way of breaking down the structure of data management systems into orthogonal components, whatever the use-cases they are built to serve. This makes the final product more flexible, more adaptable. New requirements will, we expect, map to only one or two of our dimensions, which means that the changes needed to support them will be well-contained, not spread throughout the code. New technologies, too, will be limited in their impact, because the technologies they replace are well isolated in the software.
This should lead to software designs which require less maintenance, do not become brittle over time, and which therefore last longer. Although it may appear to impose an overhead on the initial design of a new system, it should enable easier unit-testing of components and re-use of either code or at least the design of those components. It also provides a natural factorization that can allow multiple developers to work on the same system coherently.

CMS Data Management



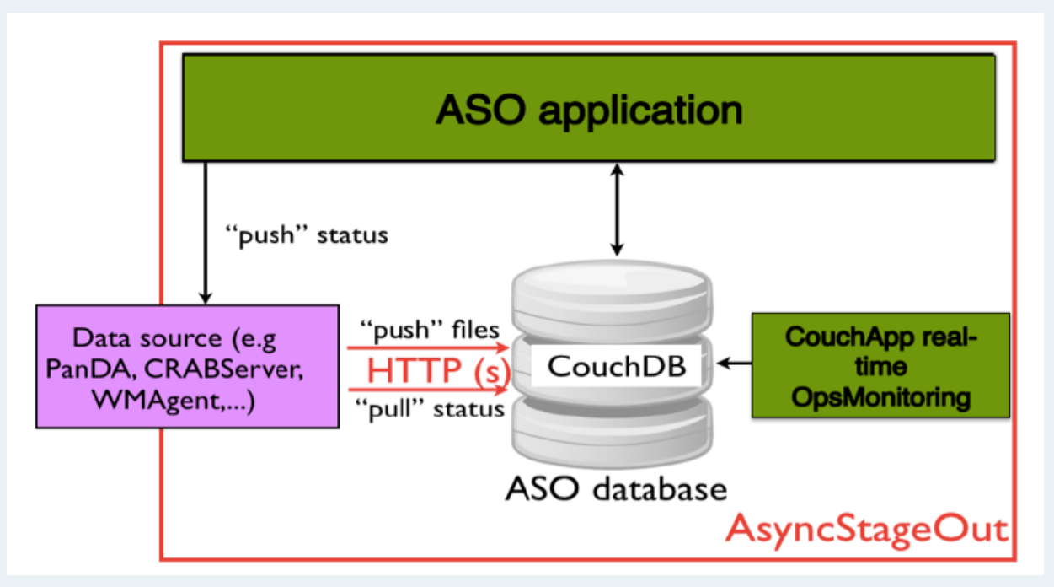
PhEDEx

PhEDEx is the data-placement management system for CMS. It handles all our scheduled data-movement, moving raw and reconstructed experiment data to archival storage, analysis data to T2's, and monte-carlo data from production sites to places where it is stored and analysed.
PhEDEx transfers in units of 'blocks' or datasets of files, where a dataset is composed of several blocks. A block is (ideally) ~1 TB, so several hundred files.



AAA

AAA stands for 'Anydata, Anytime, Anywhere'. It's the CMS implementation of an xrootd federation.
AAA does unscheduled transfers, in that there is no central planning that decides when they take place(*). Instead, the main use-case is for allowing jobs to run where CPU is available even if the data is not.

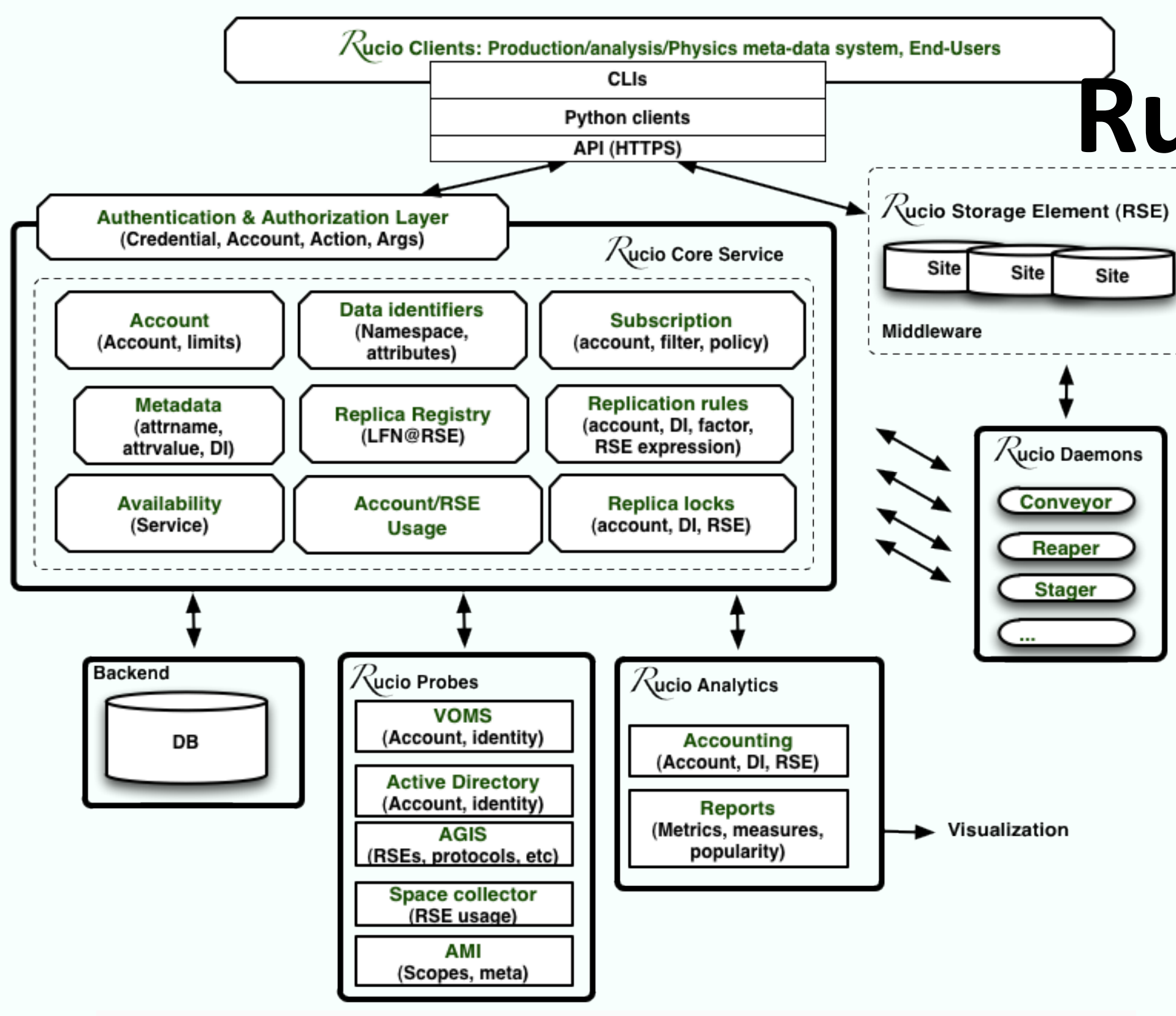


ASO

ASO is the Asynchronous StageOut component of CRAB. It's primary purpose is to transfer user-files from the site where they are created by a batch job to the final destination specified by the user.

ASO is designed to deal with up to 200K files per day, where files are expected to be ~1 GB or less. In fact, files are often _much_ less than 1 GB, so we can estimate about 20-50 TB per day on average, or about the same level as AAA (c.f. ~290 TB/day for PhEDEx transfers).

ATLAS Data Management


Rucio Clients: Production/analysis/Physics meta-data system, End-Users
CLIs
Python clients
API (HTTPS)
Authentication & Authorization Layer (Credential, Account, Action, Args)
Rucio Core Service
Account (Account, limits)
Data identifiers (Namespace, attributes)
Subscription (account, filter, policy)
Metadata (attrname, attrvalue, DI)
Replica Registry (LFN@RSE)
Replication rules (account, DI, factor, RSE expression)
Availability (Service)
Account/RSE Usage
Replica locks (account, DI, RSE)
Rucio Storage Element (RSE)
Networking
Site
Site
Site
File Transfer Service (FTS)
Middleware
Rucio Daemons
Conveyor
Reaper
Stager
...
Backend
DB
Rucio Probes
VOMS (Account, identity)
Active Directory (Account, identity)
AGIS (RSEs, protocols, etc)
Space collector (RSE usage)
AMI (Scopes, meta)
Rucio Analytics
Accounting (Account, DI, RSE)
Reports (Metrics, measures, popularity)
Visualization

Rucio

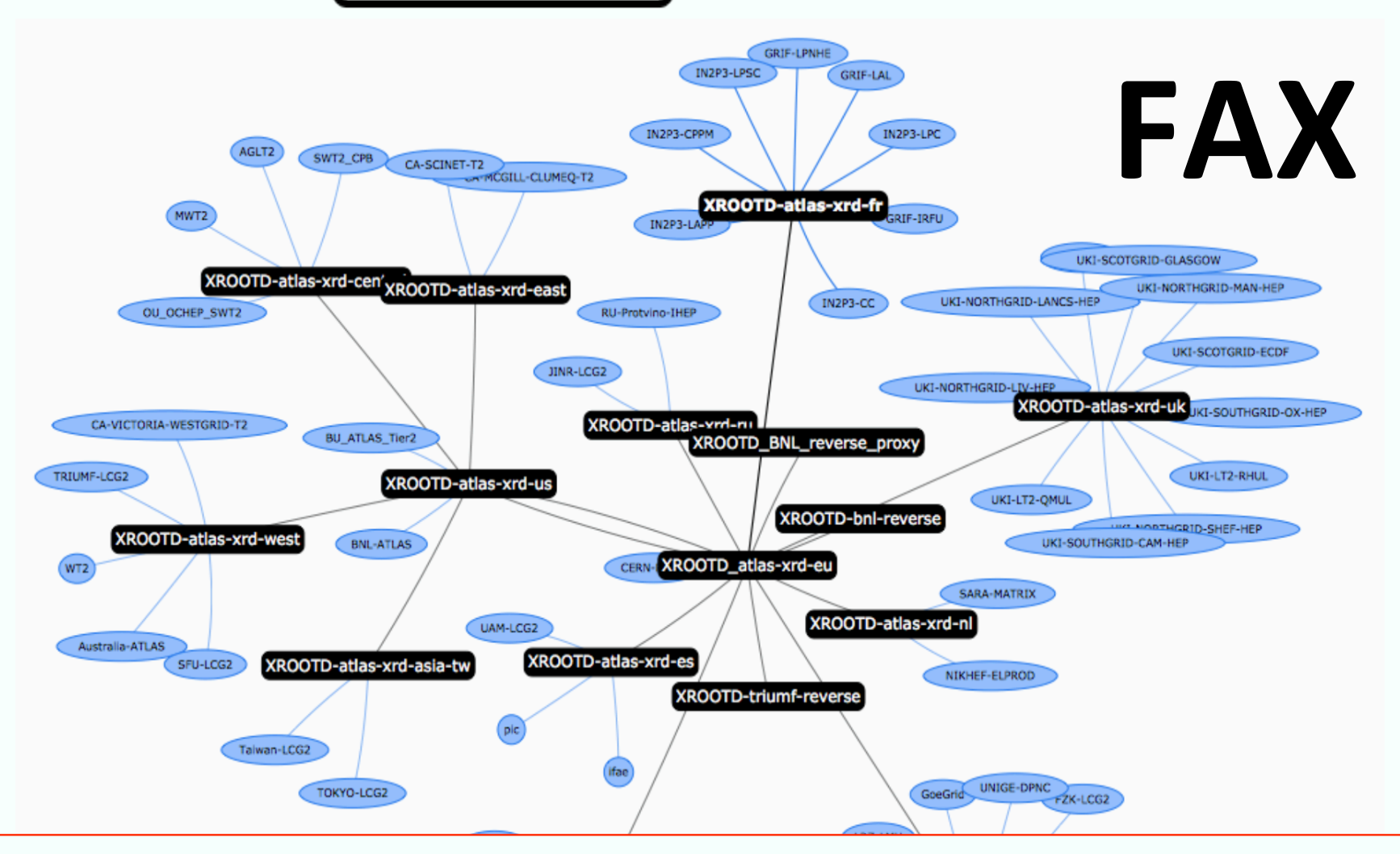
Rucio is the new distributed data management system for ATLAS. It is charged with managing all ATLAS data like detector, monte-carlo and end-user data on the grid.
All for the purpose of helping the collaboration store, manage and process LHC data in a heterogeneous distributed environment. The requirements are:

- Discover data
- Transfer data to/from sites
- Delete data from sites
- Ensure data consistency at sites
- Enforce ATLAS computing model

Files, Datasets & Containers Concepts
Files are grouped into datasets. Datasets/Containers are grouped into containers.

Others concepts: Account, meta-data, replica management & policies, accounting and quota.

Architecture
Rucio has client, server and daemons architecture with a strong focus on using open and standard technologies like WSGI, RESTful APIs and token-based authentication.
The daemons are active components that are orchestrating the collaborative work of all the system like file transfers and data deletion. The daemons are lightweight, thread-safe and scale horizontally.



FAX

FAX stands for 'Federated ATLAS XRootD'. It's the ATLAS implementation of an xrootd federation.
It spans 67 largest sites and is integrated in the ATLAS workload management system(PanDA). It is capable of remotely delivering data to all of the ATLAS analysis job where CPU is available.

Dimensions of Data Management

Data-volume

the amount of data being handled by a system is a fundamental property. An experiments' file-catalogue, for example, knows about all the files the experiment has produced. A data-movement system, on the other hand, may know only about the files which are in transit, or which are queued for transfer, at any particular point in time.

Metadata management

No system is complete without some sort of bookkeeping. How much metadata, where it comes from, how it is stored, are all important considerations. This also covers interaction with the system by users or external components. How does a request enter the system, how is it monitored, what sort of interaction is possible (e.g. suspending or resuming transfers).

Reliability

What does reliability mean for a given use-case? Clearly for custodial raw data the reliability should be close to 100%, all the data should be delivered if it is not lost beforehand. For data from opportunistic resources, volunteer computing or from monte-carlo production, a certain amount of loss may be acceptable.

Latency

minimize latency! Different things in different situations. For archiving custodial raw data, a latency of a few hours is acceptable. For a batch-job waiting to read a file, latency of the order of a few seconds is more important. In situations where latency demands cannot be met (e.g. a server holding the data crashes and must be rebooted) the response of the system can vary. Does it simply give up and report an error, or does it continue to try until it gets the data eventually?

Network structure

This has several components, such as:

- Number of nodes: today T0/T1/T2/T3 sites, tomorrow could be every personal desktop/laptop in the collaboration, every worker-node that produces or accesses data.
- Node topology: fixed or dynamic. The set of T0/T1/T2 sites is essentially a static topology. Opportunistic resources, volunteer computing, or personal laptops show far more dynamic behavior, they may join and leave the network topology with little notice. In the case of laptops they can also leave and re-join from different locations as they move from place to place. Not only the number of nodes can be dynamic, their underlying connectivity may change too.
- Types of data-flows: The type, or characteristics, of the data in a network may vary. The traffic between any two nodes may be bursty (e.g. raw data from the detector, following the accelerator-cycle) or more continuous (e.g. data from monte carlo production). The data-flows themselves may be statically defined (e.g. raw data from the detector to the T0), or the set of data-flows may change continuously (e.g. a T2 that downloads analysis data from multiple sources).

Users and security

This covers a few related items. How many users does the system have to handle? What kind of authentication, authorization and accounting are needed? Are all users considered equal or is there a hierarchy, with some users more important than others?

Others?

What else??

Throughput

The throughput characteristics may vary considerably. For most data, 'as fast as possible' is the requirement. There may be specific deadlines involved, e.g. in evacuating data from an opportunistic or shared resource which is scheduled to be returned to another user. There may also be benefits from managing the traffic, for example in scheduling traffic from thousands of worker nodes in order not to overwhelm the destination.

Conclusion

We have described the architecture of several data management systems used by the ATLAS and CMS experiments, their primary use-cases and the considerations behind their design. Even with this plethora of systems, it is not clear that all the future use-cases of the experiments will be satisfied automatically.
We propose a set of principles, 'dimensions of data management', which we believe are a more fundamental way of looking at these systems than use-cases and requirements documents. We suggest that the future evolution of these systems, and the design of any new systems, should follow these principles as a key factor in defining their architecture.
We believe this will lead to more maintainable, more flexible systems, with components and code that can be re-used in ways that are not anticipated when they are created. Given the long lifetime of LHC, we think this will be an important contribution to future productivity, with shorter development times and less overhead for maintenance and operations.