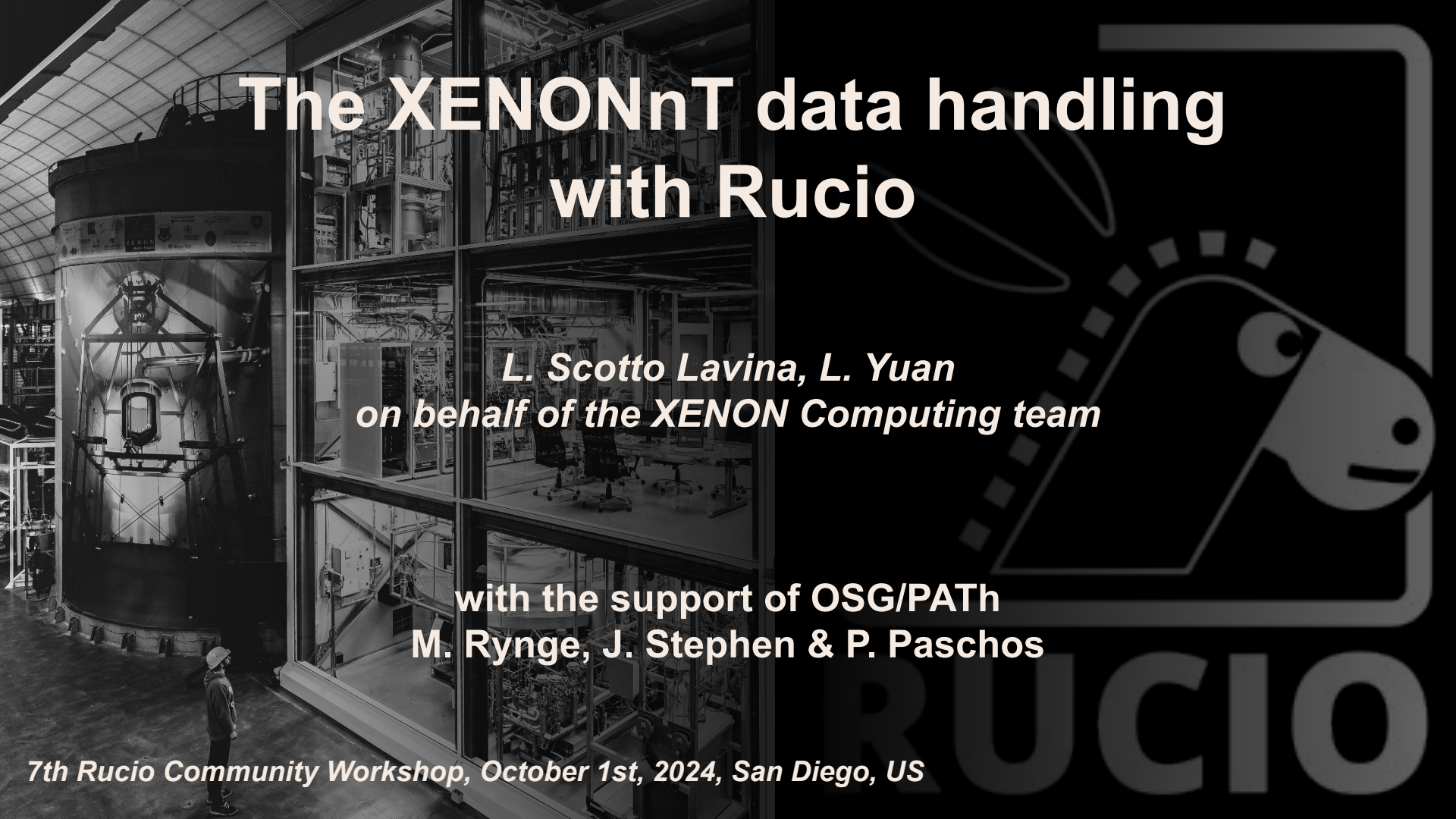


The XENONnT data handling with Rucio

L. Scotto Lavina, L. Yuan
on behalf of the XENON Computing team

with the support of OSG/PATh
M. Rynge, J. Stephen & P. Paschos

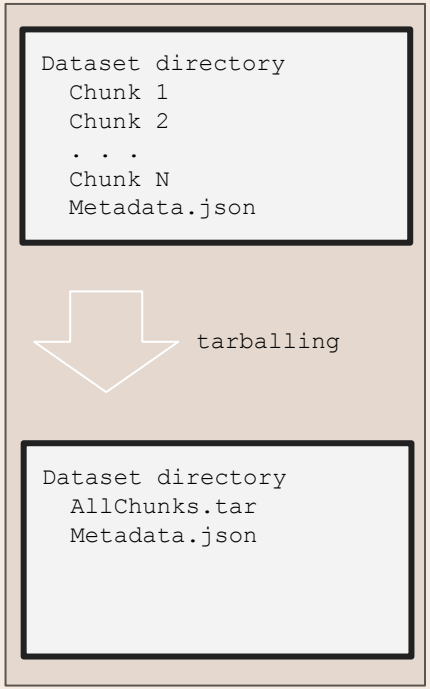
7th Rucio Community Workshop, October 1st, 2024, San Diego, US

Table of contents

- Overview of XENONnT on:
  - data format
  - cyber infrastructure
  - data pipelines
  - data processing software
- List of criticalities and space for new features in Rucio

Raw -data format

- A **run** is a ~1-hour long data acquisition
- **Raw data** of a single run are made by a directory containing several file chunks, plus a very light (ascii) json file to map the chunks
- To improve the performances of data processing, the **chunk size** must be smaller than the RAM associated to a core
- Typical Science Search Run = 56 GB → O(10) chunks
- Typical Calibration Run = 524 GB → O(100) chunks
- Drawback: **large number of small files**
- Mitigation for tape-based storage: **tarballing** the chunks of an entire run



Rucio Data IDentifier (DID) → scope:name

DID
(dataset)

```
rucio list-rules xnt_050297:raw_records-rfzvpzj4mf
```

Scope
(detector and run number)

Name
(data type and hash)

Raw data in Rucio

- A **run** is stored as a **dataset**
- The **Chunks** and the **metadata file** are uploaded as **files** attached to that dataset
- DIDs are:

xnt_<*runnumber*>:raw_records-<*hash*>

- xnt_<*runnumber*>:raw_records-<*hash*>-*000000*
- xnt_<*runnumber*>:raw_records-<*hash*>-*000001*
- ...
- xnt_<*runnumber*>:raw_records-<*hash*>-<*nchunks-1*>
- xnt_<*runnumber*>:raw_records-<*hash*>-*metadata.json*

The XENONnT data processing chain

The reconstruction code, **Straxen**, first executed on raw data, produces at different stages of the computation different kinds of processed data, with decreasing size

highest level

lowest level

Data Kind	Description	Typical Science Search 1-hour Run	Intense Calibration 1-hour Run
events-level	time-clustered peaks	<0.1 GB	<0.1 GB
peaks-level	time-clustered PMT waveforms	2.4 GB	46 GB
peaklets	preliminary time-clustered PMT waveforms	8 GB	90 GB
raw_records	recorded PMT waveforms in each channel	56 GB	524 GB

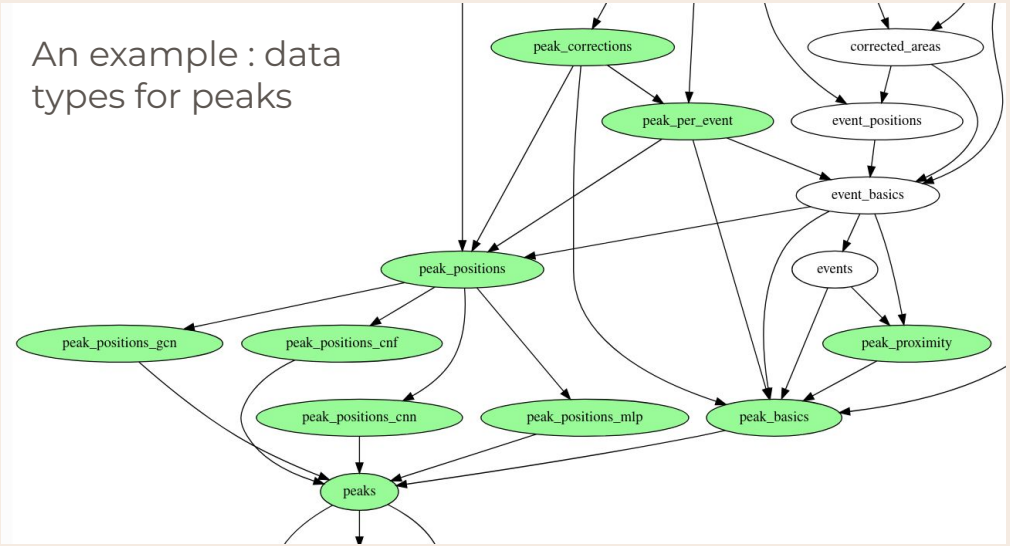
A complex data structure

Each level of a given **data kind** is composed by many **data types** sharing the same indexing system

Advantages:

- If a reconstruction algorithm changes, no need to reconstruct from scratch
- Users don't need to download the entire structure to perform a specific study

An example : data types for peaks



Drawbacks:

- Increased complexity (failures recovery more hard to follow and debug)
- High number of small files

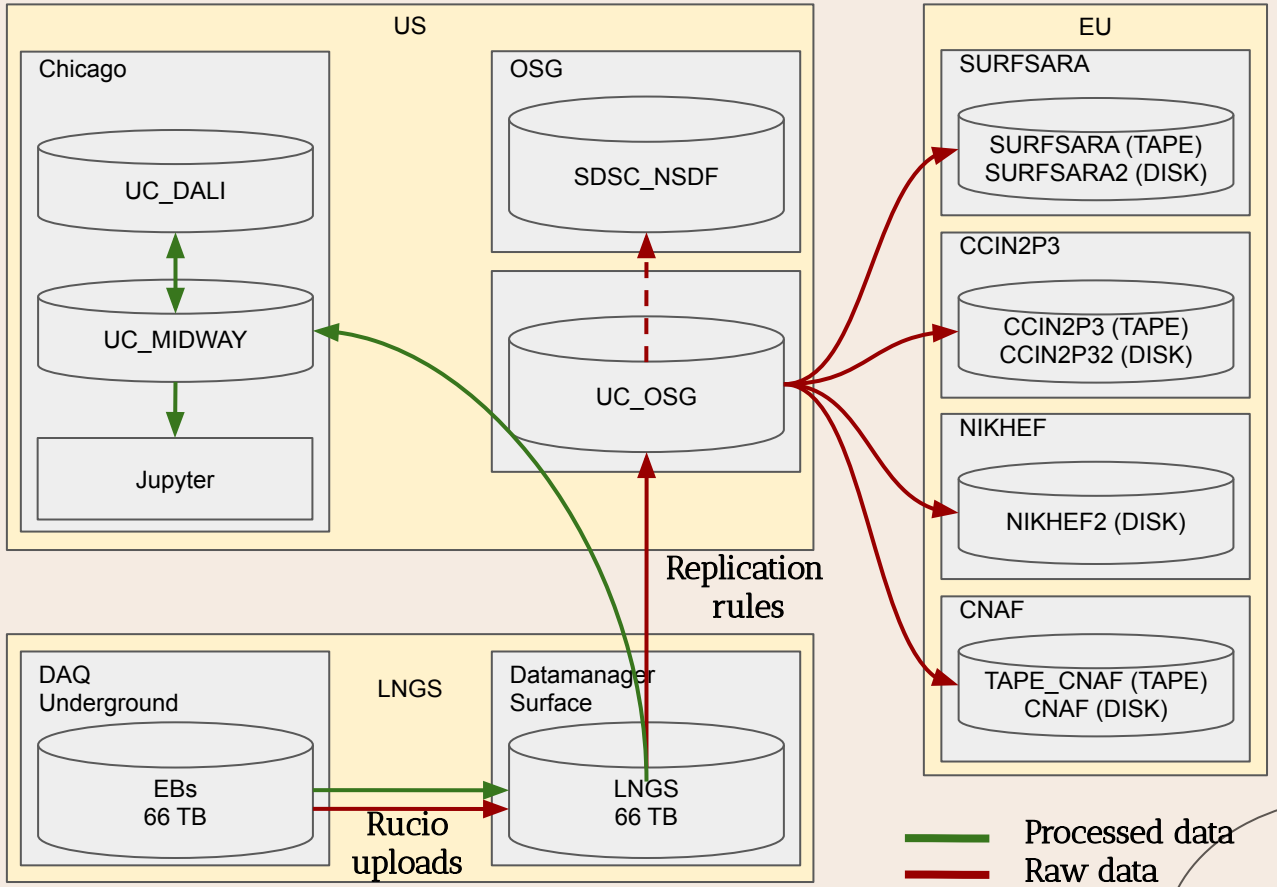
Daily online data pipeline

Raw data are **processed online** by DAQ machines and buffered (66 TB capacity)

Both raw data and processed data are **uploaded on-site** (dedicated local RSE 66TB capacity)

They are then dispatched in different RSEs (**replication rules**) depending on the level (processed data at Chicago)

With a second **conditional rule**, we instruct Rucio to perform a second copy of raw data in another RSE (usually one copy in US, one in EU)



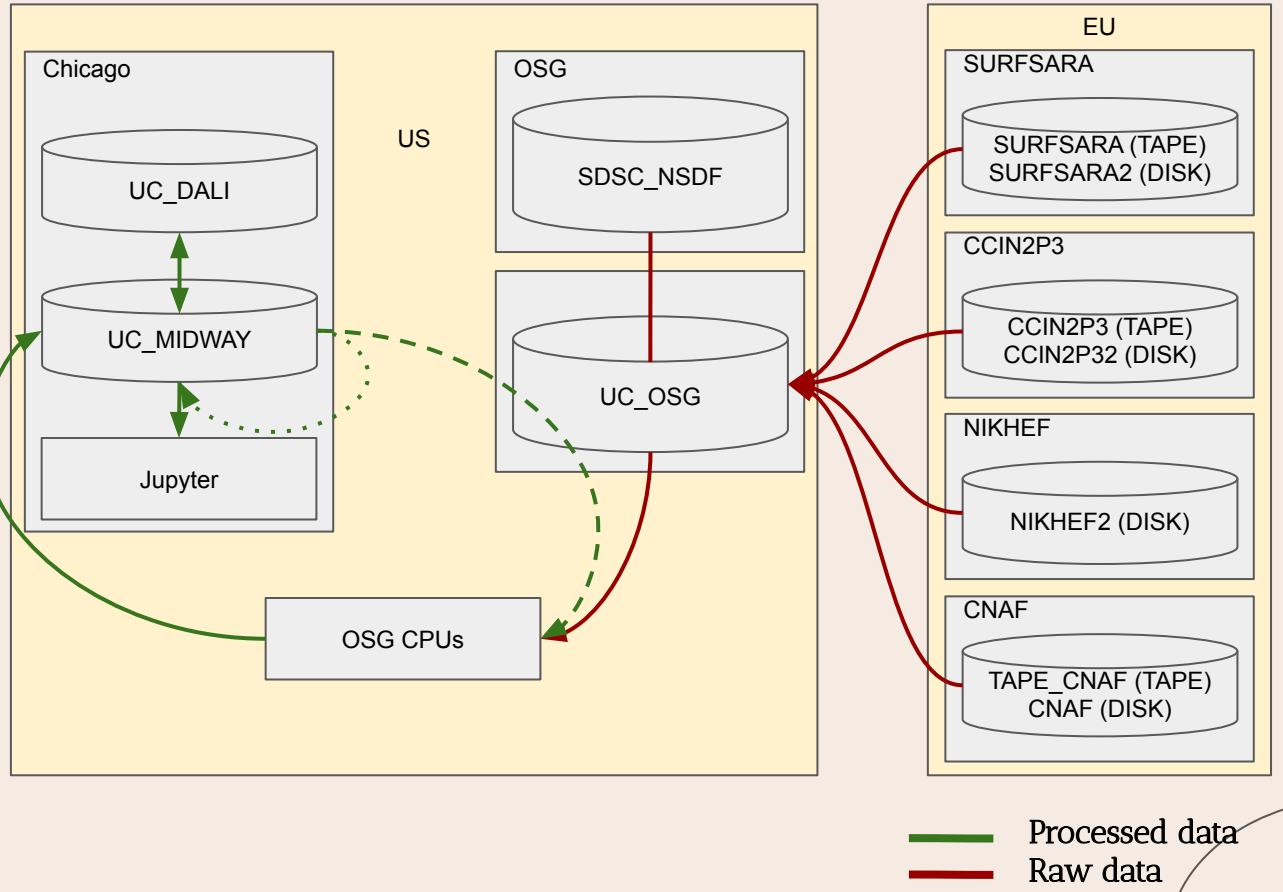
Two data-processing pipelines

Massive `raw_records` → `event` processing (~once per year)

`raw_records` from specific Science Campaigns are reprocessed using **OSG and EGI resources**. All data types, up to `events-level`, are generated.

Massive `peaklets (or peak-level)` → `event` processing (more frequent)

`peaklets` or `peaks-level` files from specific Science Campaigns are reprocessed using **OSG** and/or **UChicago resources.** All data types "above" the starting ones are regenerated.



Data management software

To manage XENONnT data, we developed a package called **aDMIX** (Advanced Data Management in XENONnT):

https://github.com/XENONnT/admix

- it is a sort of wrapper of Rucio
- it takes care of data uploading, data moving, data cleaning, data tarballing
- it keeps Rucio catalogue synchronised with our run Database (using MongoDB)

Data reprocessing software

To process the data, we use a XENONnT package called **outsource**:

https://github.com/XENONnT/outsource/

- It takes care of submitting jobs to the GRID
- These jobs will be sent to one of the slots (WMS GlideIN slots)
- Input is queried with Rucio and downloaded to the local storage of the computing node
- Once processed, data are uploaded to Rucio and sent to the desired RSE

The software/packages that are involved are:

- **CI Connect** is used for the authentication
- **Pegasus** for workflow management
- **GlideinWMS** for resource provisioning
- **HTCondor** for job scheduling

Critical points where Rucio could help

We identified few points for which we believe there is space for new features in Rucio
Most of them are meant to improve the handling of possible low performances of RSE sites:

- Copy **timeouts** during data download (from GRID to local disk)
- recovery of **failed uploads** (e.g. short network issues)
- **pre-staging** of files for tape-based RSEs

Timeouts during data download

When downloading data from a **busy disk-based RSE**, we notice that downloading using **CLI** is more robust than calling similar instructions with the **python client**

CLI command :

```
rucio download xnt_050663:raw_records-rfzvpzj4mf --rse UC_OSG_USERDISK
```

Python client :

```
rucio.client.downloadclient.download_dids(did_list, num_threads=num_threads)
```

We played with the download_dids parameters (**transfer_timeout**, **transfer_speed_timeout**) unsuccessfully

Downloading with python client, we have a higher rate of timeouts with respect to CLI

We know that timeouts depend on which backend is doing the actual copy (gfal, etc...) **but we are wondering if download_dids could have implemented an automatic retry system**

Recovery of failed uploads

If a **network issue happens during an upload**, a dataset remains **partially uploaded**. On the same dataset, we can observe (even at the same time) the following cases:

1. A file copied in the RSE, but not appearing in the catalogue
2. A file appearing in the catalogue (hence, a DID associated to it), but the file is not copied in the RSE

We would like to know what is the best way to :

- either **resume the upload**, but this would require to:
  - 1) properly add in the catalogue what has been already copied (attach file)
  - 2) physically copy a file whose DID has been already included in the catalogue
- or properly **delete** the whole dataset with all attached files and try the **upload again**

In both cases, we found issues with DID claiming to exist already even if files are not present in the RSE. Removing the DID does not help, like if the DID is still somewhere in the Rucio database.

We need help to implement a reliable clean up procedure to retry an upload

Recovery of failed uploads (our solution)

So far we fixed with a function, called preupload, that does the following:

```
def preupload(path, rse, did):
        """
        A function supposed to be run before upload to avoid orphan files failing the upload.
        It does the following
        - It adds the dataset associated to the did we wanted to upload
        - It loops over all local files to be uploaded, so to know their number and their names.
        For each file, it searches in Rucio catalogue if such a filename is already present.
        If so, it attaches it to the dataset
        - Finally, it creates a replication rule on the RSE (the RSE is an input parameter of the
        preupload function, however, it's important that the RSE must be the same chosen by the
        previous upload attempt). After this latest operation, the did will show up in Rucio.
        """
```

```
if not os.path.isdir(path):
   return

local_files = os.listdir(path)
nfiles = len(local_files)
scope, name = did.split(':')
try:
   clients.did_client.add_dataset(scope,name)
except:
   print("DID {0} already exists".format(did))
for local_file in local_files:
   try:
      clients.did_client.attach_dids(scope,name,[{'scope':scope,'name':local_file}])
   except:
      print("File {0} could not be attached".format(local_file))
try:
   clients.rule_client.add_replication_rule([{'scope':scope,'name':name}],1,rse)
except:
   print("The rule for DID {0} already exists".format(did))
```

This solution does not work for the totality of use cases

Pre-staging of files for tape-based RSEs

In case we want to get some data stored in tapes (to download it, to create a new replication rule, etc.), the operation takes a lot of time because data need first to be staged by the tape-based RSE.

We developed in our aDMIX software a tool that move first data on the RSE disk buffer:

```
import gfal2
def bring_online(self,did,rse):
        print("Bringing online {0} from {1}".format(did,rse))
        scope = did.split(':')[0]
        dataset = did.split(':')[1]

        file_replicas = Client().list_replicas([{'scope':scope,'name': dataset}],rse_expression=rse)
        files = [list(replica['pfns'].keys())[0] for replica in file_replicas]

        print("Bringing online {0} files".format(len(files)))

        ctx = gfal2.creat_context()
        try:
                pintime = 3600*48
                timeout = 3600
                (status, token) = ctx.bring_online(files, pintime, timeout, True)
                if token:
                print(("Got token %s" % token))
                else:
                print("No token was returned. Are all files online?")
        except gfal2.GError as e:
                print("Could not bring the files online:")
                print(("\t", e.message))
                print(("\t Code", e.code))
```

This feature uses the bring_online function of the gfal2 library

Wondering if something similar could be included in the Rucio tools:

```
rucio –prestage-did <DID> <RSE>
```

Dynamic handling of RSEs hostname:port

By time in time, some GRID sites hosting our RSEs update their parameters, namely :

- the host name
- the port
- the protocol (migration to webdav, for instance)

In the bring_online function implemented in aDMIX,, for certain RSEs we have been obliged to replace the **pfns** to switch from gsiftp protocol to srm (and the **port** number), otherwise the gfal2 bring_online function would be unusable

```
files = [list(replica['pfns'].keys())[0] for replica in file_replicas]

      if rse=="SURFSARA_USERDISK":
            for i, file in enumerate(files):
            files[i] = files[i].replace("gsiftp","srm")
            files[i] = files[i].replace("gridftp","srm")
            files[i] = files[i].replace("2811","8443")
```

If a "`rucio -prestage-did`" is implemented in Rucio, we don't have to do it anymore, since we could benefit from the Rucio feature of handling multiple protocol://hostname:port for each RSE

Summary and outlook

The Rucio experience in XENONnT is great

Several requirements of the XENONnT pipeline brought us to few critical points for which Rucio could help :

- **Wondering if rucio.client.downloadclient.download_dids() could have implemented an automatic retry system**
- **We need help to implement a reliable clean up procedure to retry a failed upload (due to network issue, for instance)**
- **Proposing a new Rucio command (rucio prestage <DID>) allowing to bring online data (from the tape to the buffer system associated). If implemented in Rucio, we could profit of his capability to handle multiple protocol://hostname:port for each RSE**