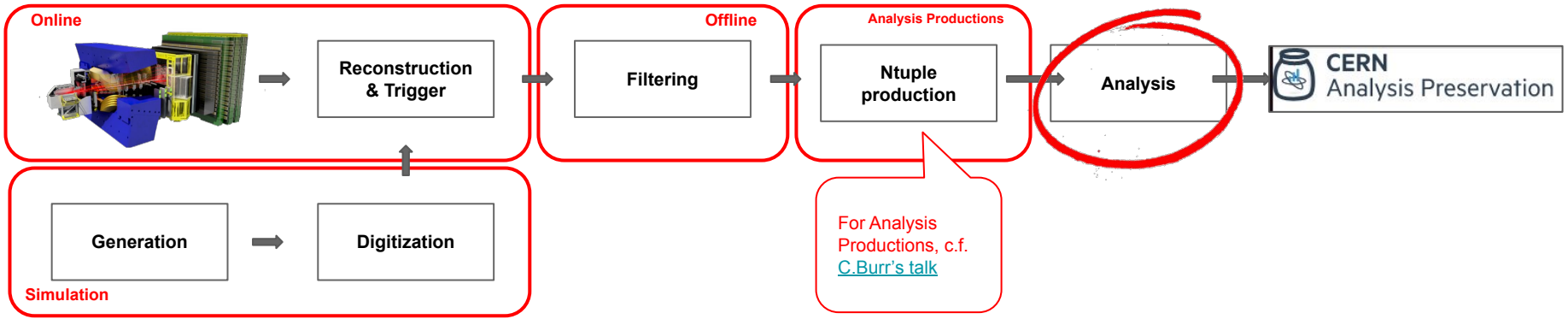
Facilitating the preservation of LHCb analyses

C. Burr, B. Couturier, R. O'Neil
on behalf of the LHCb collaboration

CERN
Università degli Studi di Ferrara
THE UNIVERSITY of EDINBURGH

Facilitating Analysis Preservation



from https://analysispreservation.cern.ch/docs/general/what.html

"The analysis preservation effort pursues the goal of describing and structuring the knowledge behind a physics analysis so that it is understandable and reusable in the future."

⇒ Crucial to track the provenance of the results, preserve the tools and associated software environments

All production steps from trigger/simulation to filtering are carefully tracked, their artefacts referenced in the LHCb bookkeeping.
Analysis productions track the production of ntuples, their artefacts also in the bookkeeping

Analyses of course cannot be standardized, but how can we make it easier to track and preserve the workflows for preservation?

Analysis environments

We would like to:

- **Track the provenance** of all artefacts used in papers
- **Preserve the environments** used to derive those artefacts

To prepare and preserve the environments:

- Publish the LHCb specific tools to conda-forge
- **Prepare environments with the tools requested by LHCb analysts** (ROOT, Scikit-HEP packages, ML etc…)
- **Deploy conda environments to CVMFS** (a.k.a “lb-conda”)
  - Can of course also be deployed locally
  - System of deployments “on-demand” via GitLab MRs and CI

See from CHEP 2019 https://cds.cern.ch/record/2699537?ln=en

Analysis provenance

How can we help with provenance tracking?

- Encourage the use of **workflow tools from the start of the Analysis**
- Make it easy by **integrating the workflow with the LHCb bookkeeping**

Solving this problem is the idea behind the **apd** (Analysis Productions Data):

- A python package that interfaces with the bookkeeping database to find file locations
- Uses metadata (tag name/value pairs) provided by users to select the data

It is integrated with **Snakemake,** already popular within LHCb Analysts (and taught during the LHCb Starterkit)

There are other advantages as well:

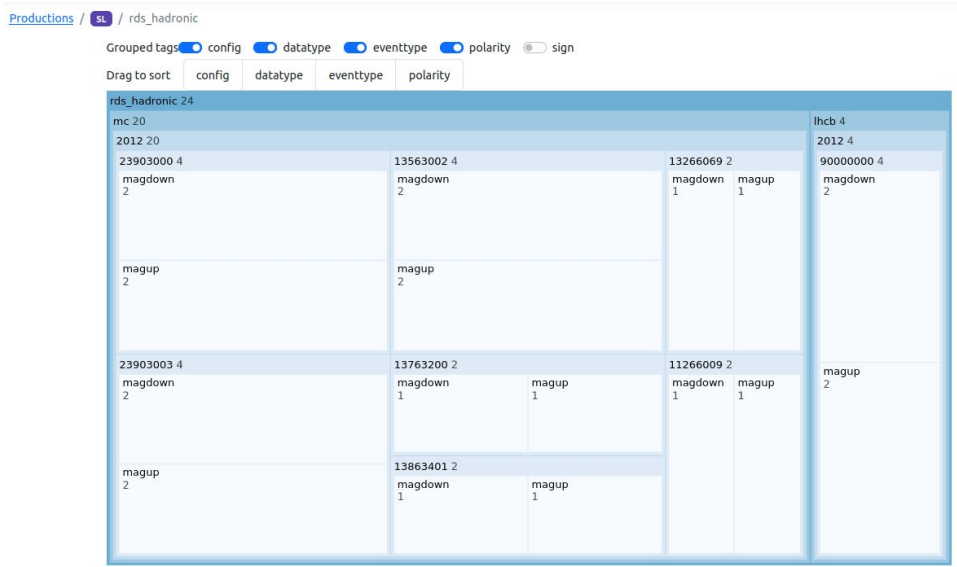
- Avoid hardcoding paths, as that is error-prone and inflexible
- Resource utilization: avoid duplication in each analysts EOS area

Enriching the metadata

Tags are simple meaningful key →value mappings

- Some tags are **automatically generated** e.g.
  - **Config** (mc/lhcb)
  - **Datatype** (e.g. data taking year)
  - **Event type** (identifier for decay chain)
  - **Magnet polarity**

- **Additional tags can/should be manually added**
  - E.g. decay=d2kpi/sign=ws/leptons=mumu



Treeview of the number of samples available, for a specific analysis, organized by configuration/year/event type/magnet polarity

Datasets can therefore be identified by WG/Analysis/tags, helping for communication and provenance tracking

Physical file locations

The dataset object can be called with a number of tags,
and returns the list of associated physical file names:

```
>>> from apd import get_analysis_data
>>> dataset = get_analysis_data("sl", "rds_hadronic")
>>> urls = dataset(config="lhcb", datatype="2012", polarity="magdown", eventtype="90000000", sign="rs")
>>> urls[0]
'root://eoslhcb.cern.ch//eos/lhcb/grid/prod/lhcb/LHCb/Collision12/DATA_BS.ROOT/00173027/0000/00173027_00000012_1.data_bs.root'
>>> len(urls)
195
```

An error is returned if the tags map to more than one sample:

```
>>> dataset(config="lhcb", datatype="2012", polarity="magdown", eventtype="90000000")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/cvmfs/lhcbdev.cern.ch/conda/envs/default/2023-03-05_13-05/linux-64/lib/python3.10/site-packages/apd/analysis_data.py", line 355, in __call__
    raise ValueError("Error loading data: " + error_txt)
ValueError: Error loading data: 1 problem(s) found
{'config': 'lhcb', 'datatype': '2012', 'polarity': 'magdown', 'eventtype': '90000000'}: 2 samples for the same configuration found, this is ambiguous:
     {'config': 'lhcb', 'polarity': 'magdown', 'eventtype': '90000000', 'datatype': '2012', 'sign': 'rs', 'version': 'v0r0p4882558', 'name': '2012_magdown_data_bsdstaunu', 'state': 'ready'}
     {'config': 'lhcb', 'polarity': 'magdown', 'eventtype': '90000000', 'datatype': '2012', 'sign': 'ws', 'version': 'v0r0p4882558', 'name': '2012_magdown_data_bsdstaunu_ws', 'state': 'ready'}
>>>
```

Snakemake integration

```
# Import the apd tools, versions customized for Snakemake
from apd.snakemake  import get_analysis_data, remote

# Get the APD dataset for my analysis
dataset = get_analysis_data( "s1", "rds_hadronic" )

# templated rule to produce a ROOT file with the histogram for B_M in a
# specific sample, notice that:
#  - the input uses the dataset object and specifies the wildcards to use
#  - the output is local in this case, we could temp() if we want them cleared
#     after completion of the workflow
#
rule create_histo:
   input:
       data= lambda  w: dataset(config=w.config,  datatype=w.datatype,  eventtype=w.eventtype,  polarity=w.polarity)
   output: "bmass_{config}_{datatype}_{eventtype}_{polarity}.root"
   run:
       import ROOT
       rdf = ROOT.RDataFrame( "SignalTuple/DecayTree" , [ f for f in input ])
       h = rdf.Histo1D((hname, hname,  200, 0., 25e3), "B_M")
       [...]
```

Looking up the Analysis production information

We pass the wildcards corresponding to the tags to the dataset to get a list of files matching
(in fact Snakemake XRootD remotes)

This rule can be used to produce files with names matching this form. Each name within {} is a “wildcard”

Making life simpler for the Analysts

With **apd**, it is very easy to:

- **Locate ntuples** for various analyses
- **Share ntuple lists** between Analysts and Analyses (i.e. a tuple of tag names instead of list of bookkeeping paths)
- **Process using Snakemake** workflows

*It is also **possible to cache (part of) the data locally** and use the local copy transparently for analysis*
(e.g. for analysis on own laptop)

***Local installations are easy** to setup in a conda environment with analysis tools*

To make sure that the workflows are correct, we can use GitLab/GitLab Ci to rerun the code

Short(ish) analysis steps can be run in GitLab CI runners, but longer ones could be sent to the grid:
(c.f. **A gateway between GitLab CI and DIRAC**, CHEP 2019)

Authentication and authorisation in GitLab CI

Snakemake workflows can be run within a GitLab CI … BUT

- **apd requires credentials** to access the information
- Data access implies **GitLab CI having a proxy or kerberos credentials**

The **apd-login** command has been integrated with GitLab CI:

- **Job gets a token for access to apd data when provided with a valid JSON Web Token** from GitLab (c.f. var CI_JOB_JWT_V2)

  If the GitLab project has been registered in the Analysis Productions application settings page

- It is also possible to **specify EOS directories to which the CI will have access**,

N.B. Tokens have a duration corresponding to the max lifetime of the GitLab CI job

Using EOS tokens

EOS tokens have to be appended to the URL of the ROOT file as parameters, e.g. a file URL becomes:

```
root://eoslhcb.cern.ch//eos/lhcb/grid/prod//MYFILE.root?xrd.wantprot=unix&authz=<eos_token>
```

For this reason apd provides method to add the correct token to the file URL:

- **apd.auth**: adds a read-only token
- **apd.authw**: adds a read-write token

The **Snakemake interface automatically wraps the input and output files with tokens**
(rule output should ask for a rw one)

This is independent from the Analysis Productions themselves and could be used in a more generic way…

Derived Analysis Productions

An analysis is composed of many workflows… **How can we compose the results of many steps** ?

- naming convention in EOS?
- register derived data in the bookkeeping ?

Plans to allow for “Derived Analysis Productions”, e.g. ntuple enriched with BDT information

- For computationally expensive steps early in the analysis chain.
- Not required to be embarrassingly parallel (as is the case for analysis productions)
- Preferably with a notable reduction in data volume

Design:

- Git repo where the CI produces artefacts copied to EOS
- With a Snakemake workflow that controls the production of the artefacts
- And registers them to the LHCb bookkeeping

Conclusion

apd helps LHCb Analysts exploit the ntuples from by Analysis Productions

- A lot of interest, several early measurements are already using it

C.f. C.Burr's talk in track 6 concerning Analysis Production

Integration of Snakemake workflows and of GitLab CI helps tracking the provenance of the results

- *GitLab CI not appropriate for all steps of course*
  e.g. complicated fits requiring loads of resources or special hardware do not fit the model

Interesting start but need to see this works in practice