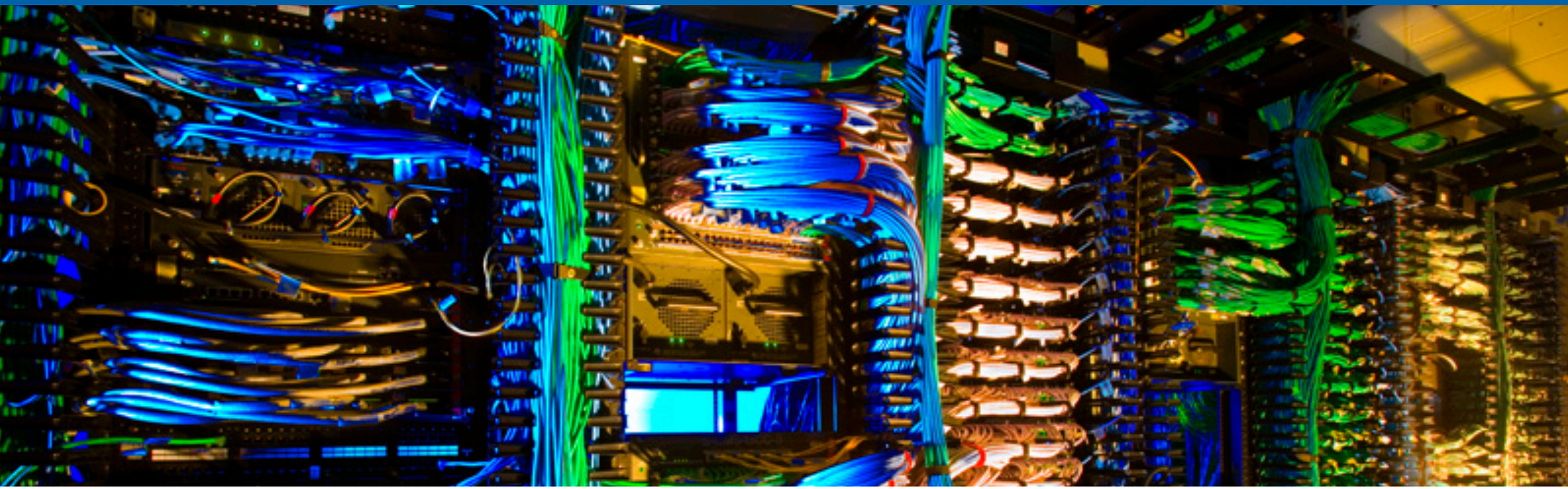


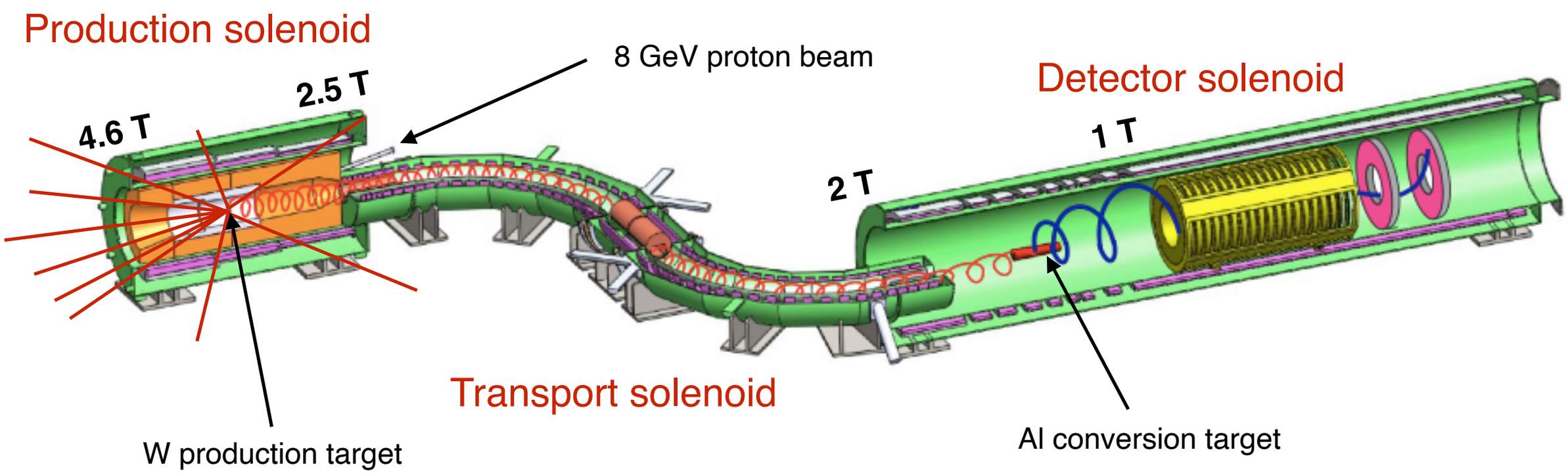
Mu2e and its Framework Usage

Stefano Roberto Soleti

HSF Frameworks WG Meeting

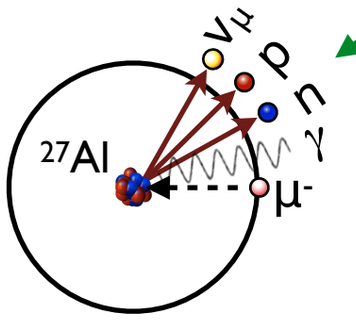
4 May 2022

Mu2e in one slide



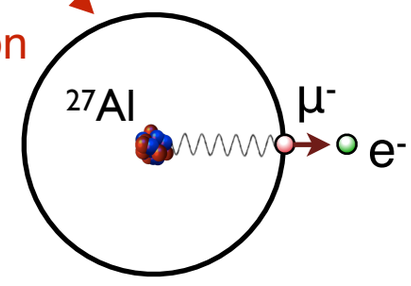
The Mu2e experiment is looking for the conversion of a muon into an electron in the field of a nucleus

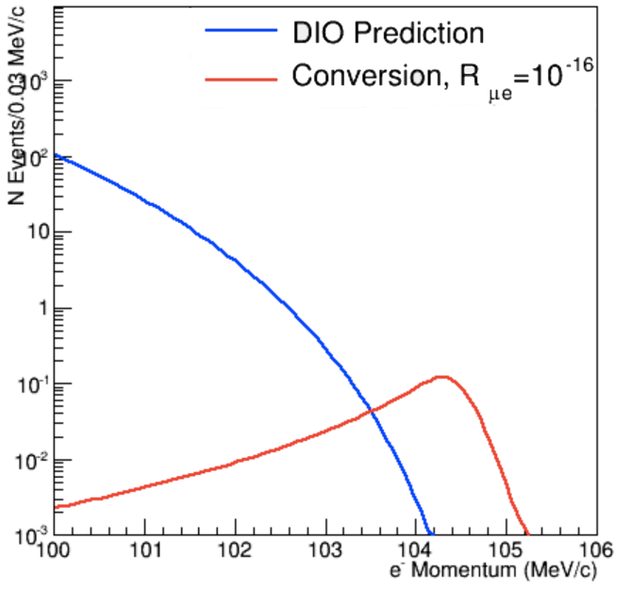
$$R_{\mu e} = \frac{\Gamma(\mu^- + N(A,Z) \rightarrow e^- + N(A,Z))}{\Gamma(\mu^- + N(A,Z) \rightarrow \nu_\mu + N(A,Z-1))} < 8 \times 10^{-17} \ @ \ 90\,\%\,\mathrm{C.L.}$$



Muon capture

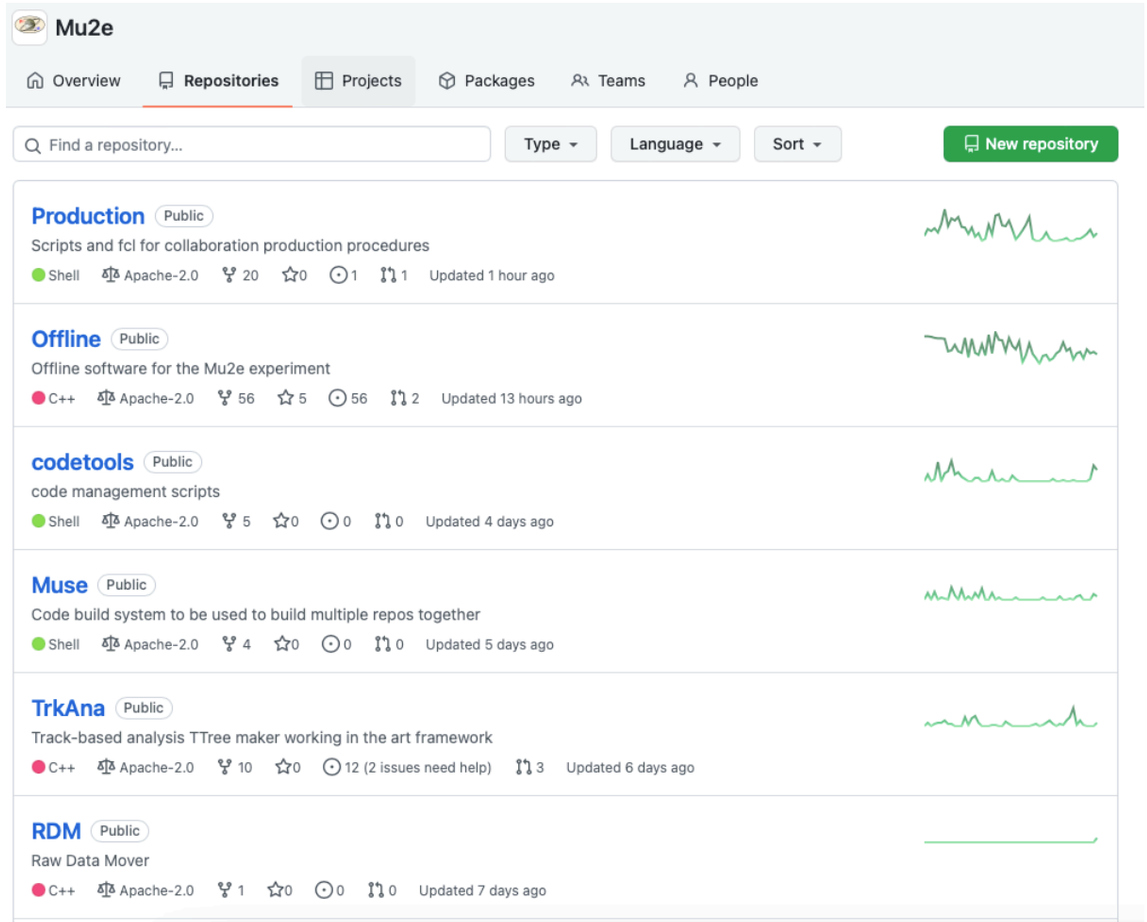
Muon conversion





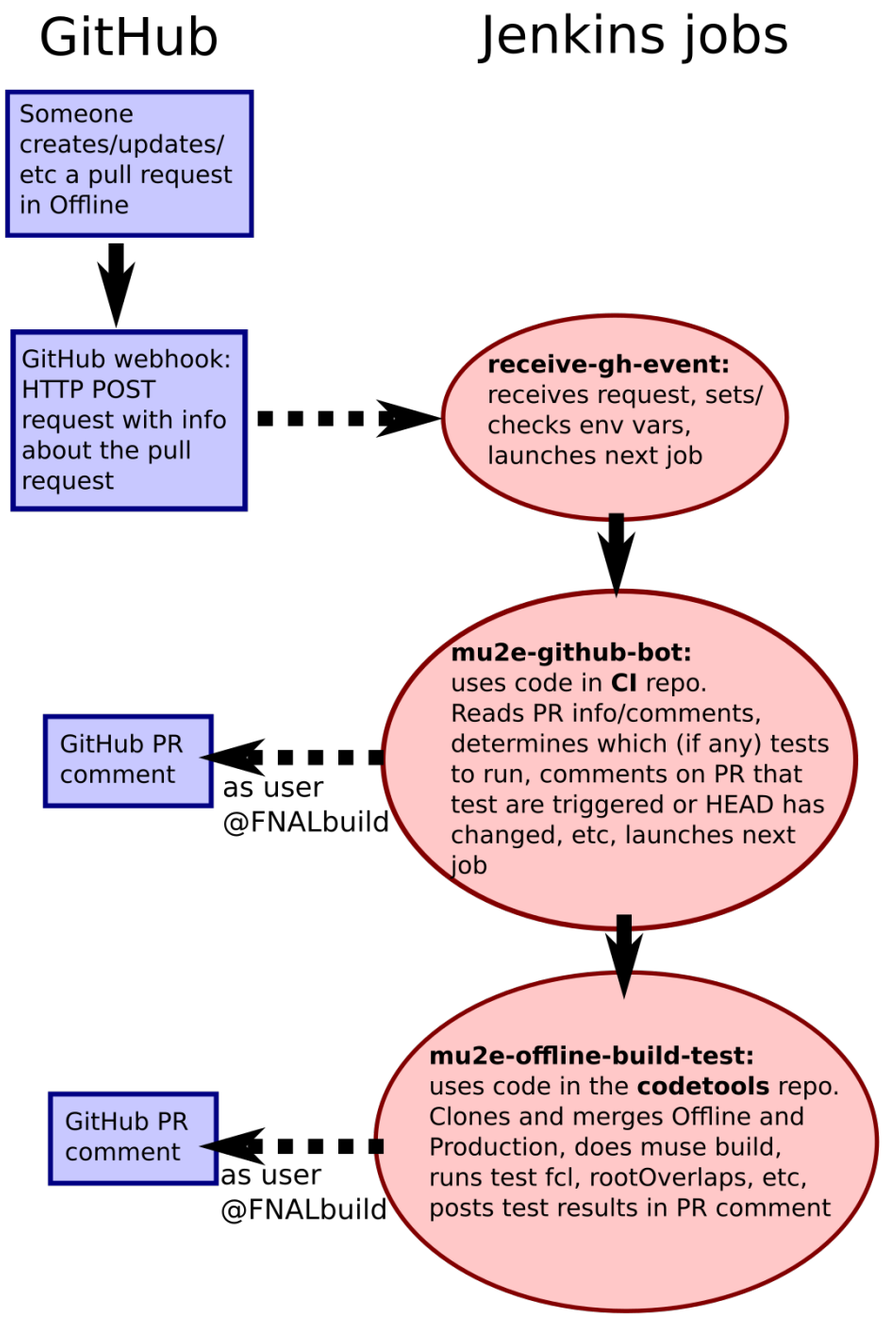
The Mu2e framework

- **Mu2e** is an experiment at the intensity frontier: we are looking for an extremely rare experimental signature with an unprecedented sensitivity. **An accurate simulation is of paramount importance**.
- Our codebase is split into several parts, which can be developed independently, e.g.:
  - **Offline**: art instance used for simulation, triggering, calibration and reconstruction
  - **Production**: set of FCL files used for production campaigns
  - **TrkAna**: general-purpose art analyzer which produces TTrees useful for analyses
  - **REve**: web-based event display built on Eve-7



- Each part is hosted on an **independent GitHub repository**.
- **Fork-and-pull collaborative development model** adopted with success: each developer can make a personal fork of the repository and then open a pull request with the changes.

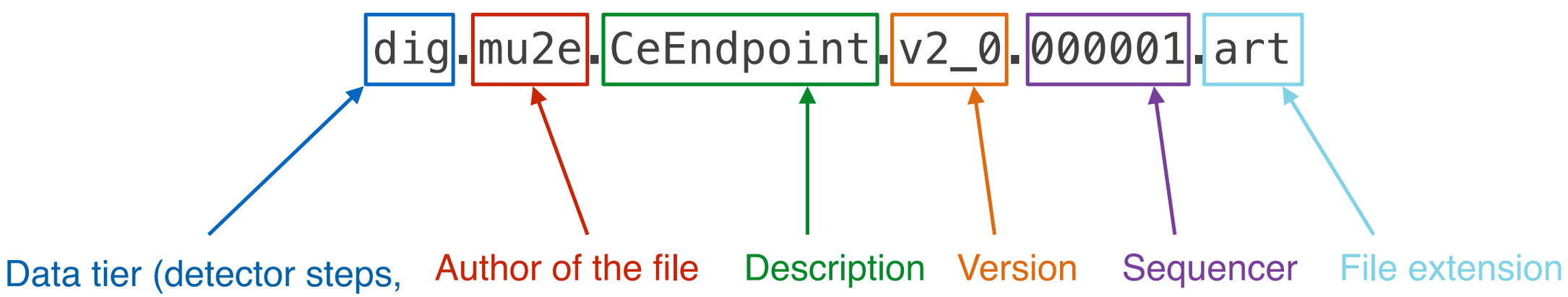
Continuous integration with GitHub



- GitHub offers sophisticated hooks for the **implementation of continuous integration** workflows.
- The goal of the continuous integration is to **test that our codes and scripts** function without formal errors and to catch build and/or execution level problems.
- When there's a new Pull Request (PR), a PR is updated, or new comments are made, GitHub reaches out to Jenkins via a web-hook.
- Jenkins then requests the PR information from GitHub and decides if it should run the tests
- **Runs tests when there is a new PR**, or the tests are deliberately requested and are not already in progress.
- Extremely flexible system: **different PR combinations** from separate repositories can be tested.

Dataset production

- In Mu2e we have a strictly enforced file naming and dataset naming convention which allows us to determine the location of a file without in principle querying SAM:



- This file can be saved to scratch, persistent or tape, and in each case it will have a deterministic file location, e.g.

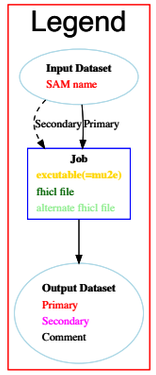
```
/pnfs/mu2e/tape/phy-sim/dig/mu2e/CeEndpoint/v2_0/art/28/0f
```

“stretcher” directories based on the SHA256 hash of the filename

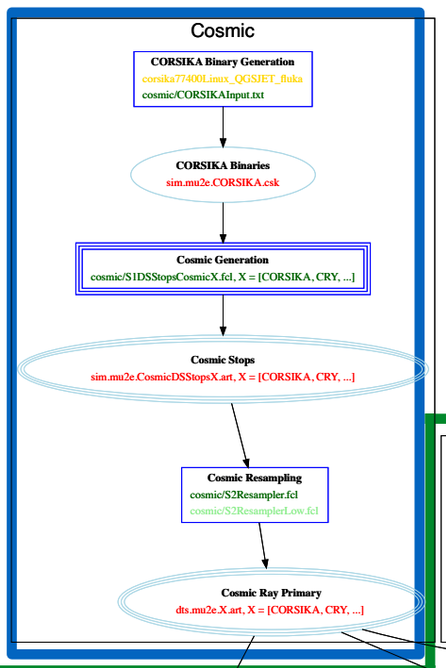
Production workflow

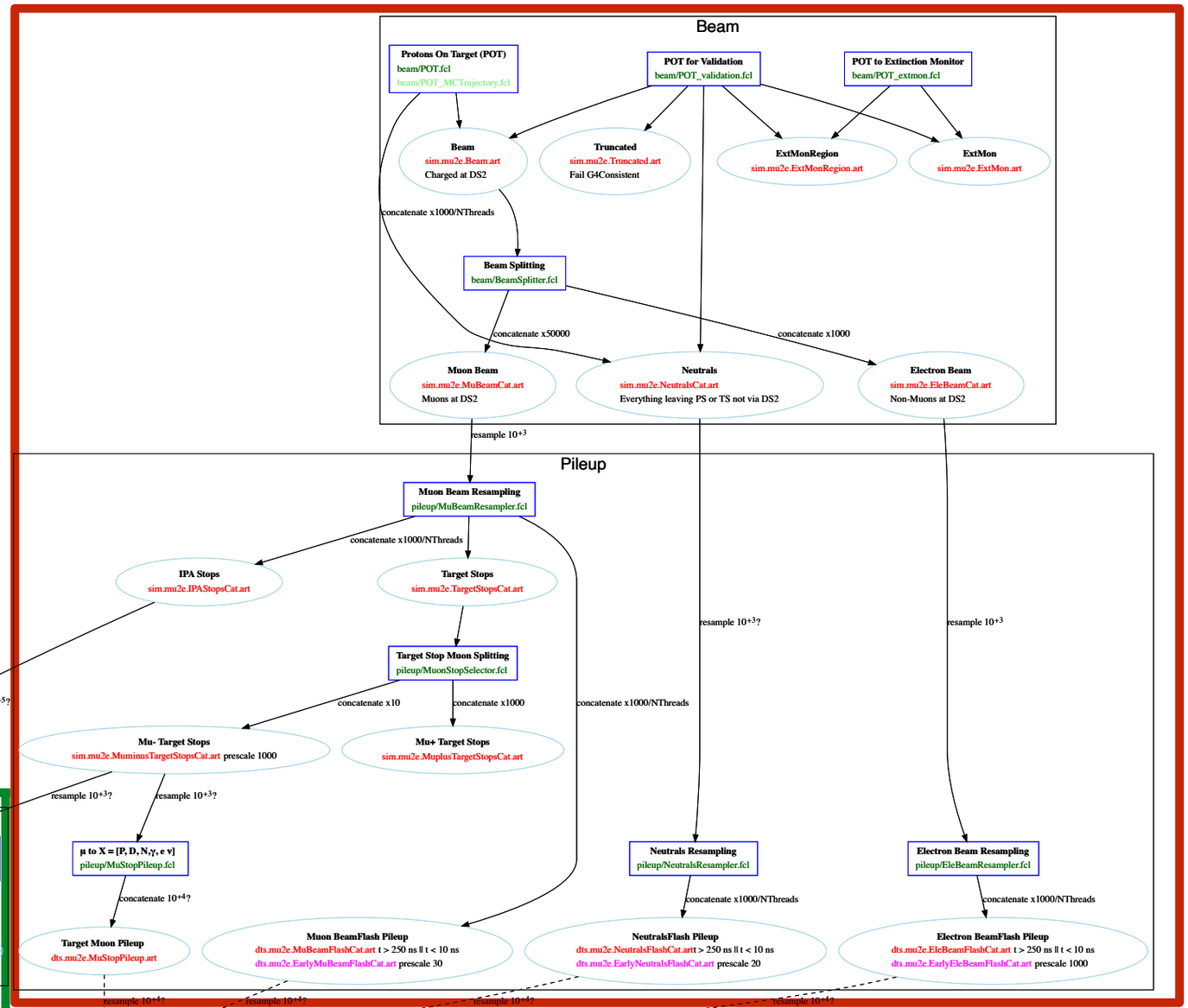
- The MDC2020 production **aims to simulate events in our detector starting from the protons-on-target**. However, it's computationally unfeasible to simulate all the ~10^{20} POTs we expect to collect.
- A workaround is represented by the ***resampling* technique**: we re-use the output of some simulation stages several times. The stochastic nature of the simulation ensures (to a certain extent, that we need to quantify) that we are not simulating the same event several times. We also need to make sure we are covering enough phase space.
- Our most recent production is organized into **three conceptual parts**: the cosmic-ray simulation, the beam pile-up simulation, and the primary simulation.

beam pile-up



cosmic





primary

Using POMS for job submission

Production
Operations
Management
System

The **P**roduction **O**perations **M**anagement **S**ystem is a system that enables automated jobs submission on distributed resources (the “grid”).

POMS provides a coherent way to fully exploit these resources and it is interfaced to three main systems:

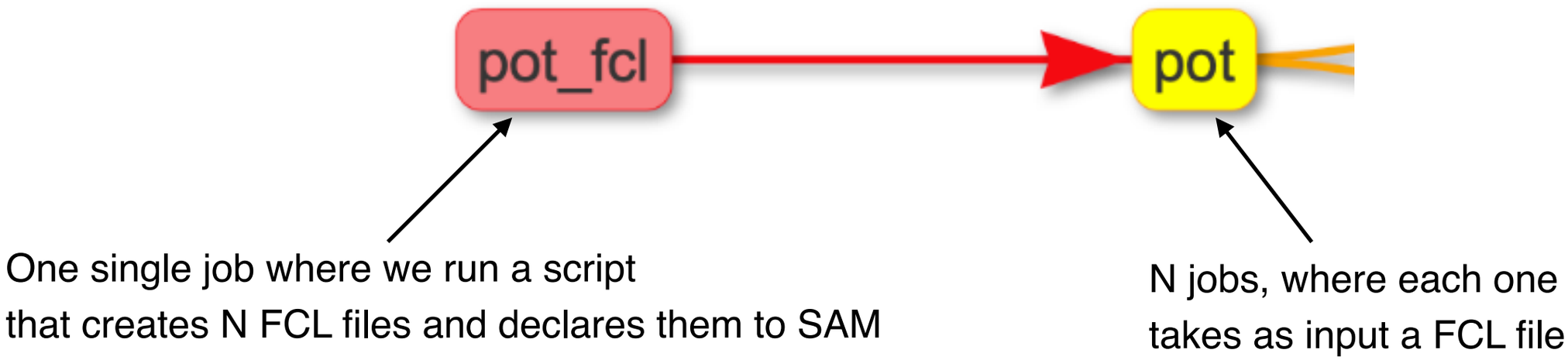
- Jobsub, for jobs management
- SAM, for data handling
- FIFEmon for monitoring

The system provides high flexibility in the job submission:

- It is built around the concept of “campaign”, a combination of job stages that can run in sequence or in parallel.
- Each stage can run a different executable and any number of bash commands before and after the main executable.
- It is possible to specify the dependencies of each stage: the submission of the stage jobs will start automatically once the dependencies are satisfied (e.g. a previous stage completed successfully.)

From production workflow to POMS

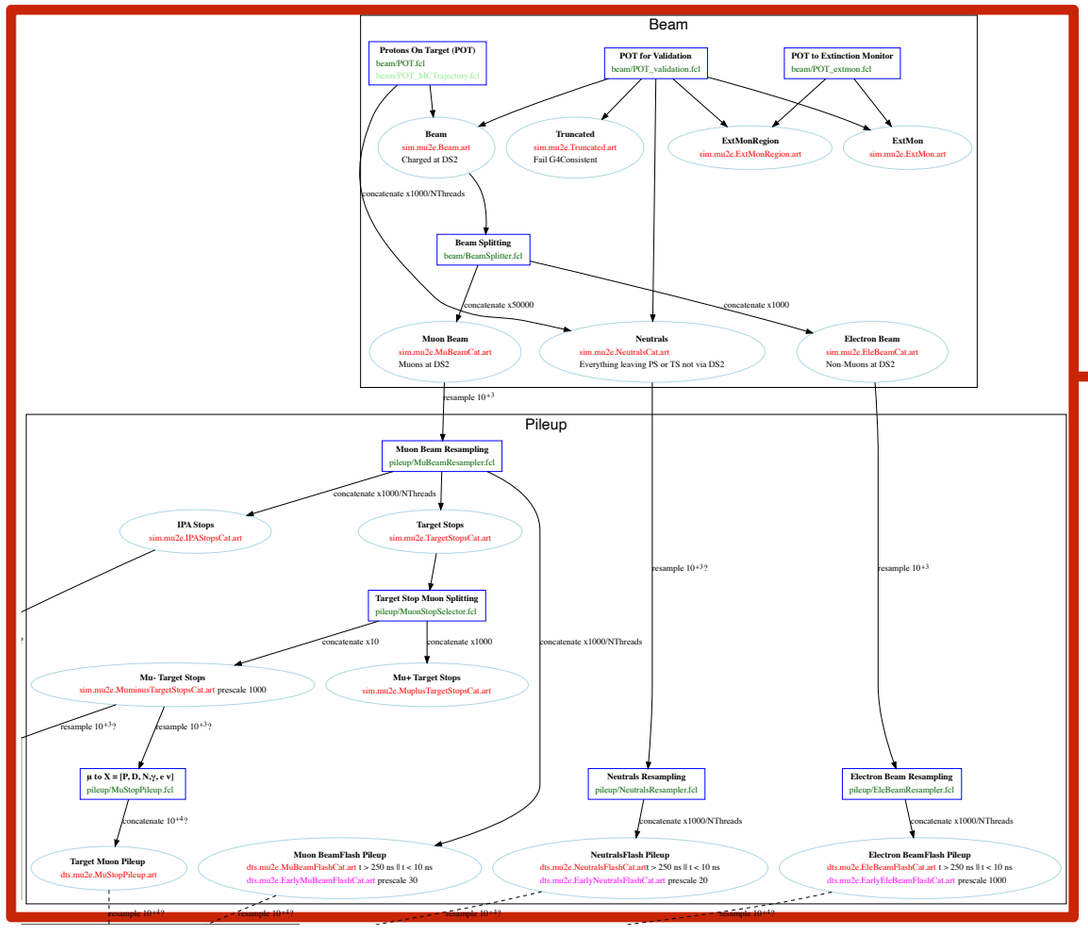
- Our workflow is translated into a **complex chain of POMS stages**.
- We require each job to be exactly reproducible, with the exact **same random seed and same input files**. A way to achieve this is to store a FCL file for each job we submit, which allow us to easily re-run a single job, both on the grid and interactively.
- However, this becomes onerous for large campaigns, since it means storing hundreds of thousands of small text files and keep track of which stage corresponds to which FCL files.
- POMS allows to automatically submit a sequence of stages: in our workflow, before running N jobs, we **run a single job which produces the N fcl files** that are used as input. Storing them to dCache is too slow for large campaigns: we store them in a dedicated database and then declare them to SAM.



From production workflow to POMS

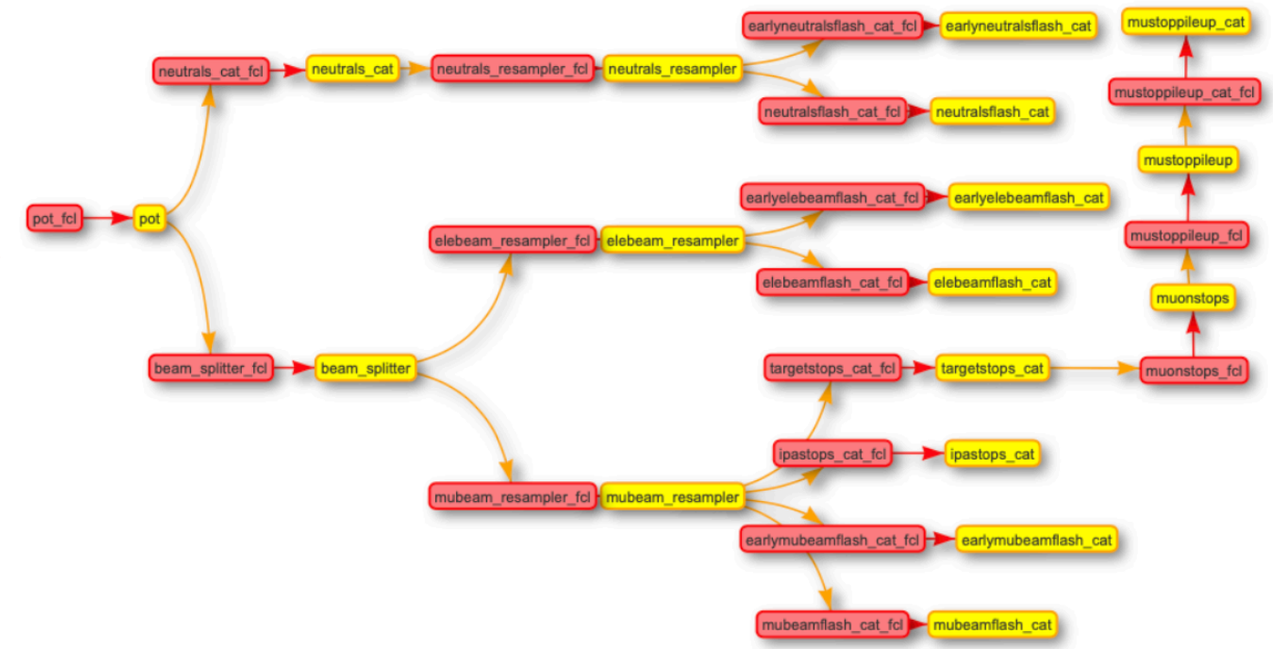
- So, in our POMS campaign we have **generate_fcl** stages, where we generate the FCL files we need (one per job), and **mu2e** stages, where we run the `mu2e` process (our instance of *art*).

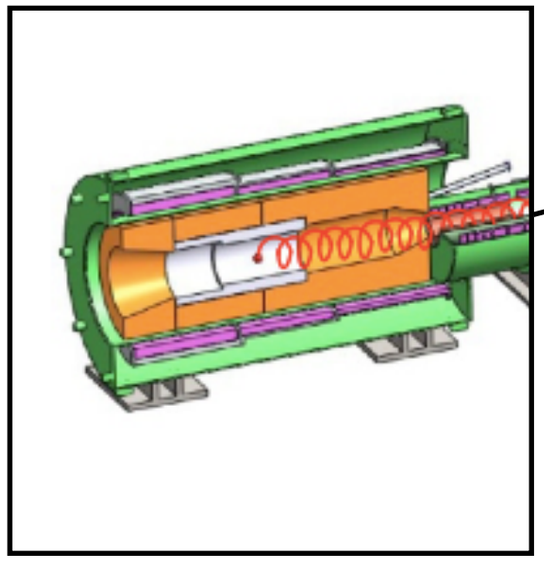
Workflow



beam+pileup

POMS campaign

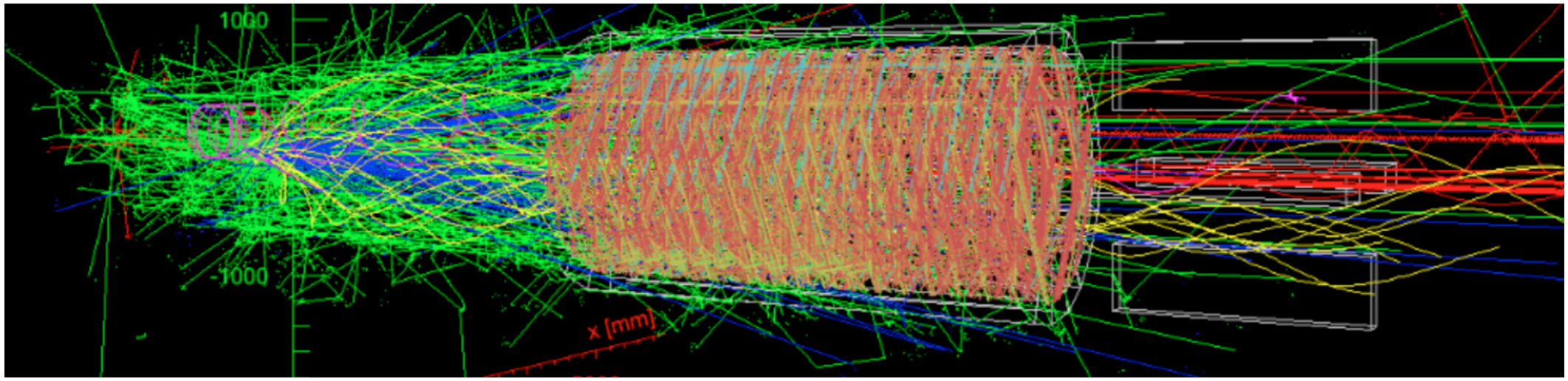




Beam
Protons On Target (POT)
beam/POT.fcl
beam/POT_MCTrajectory.fcl
POT for Validation
beam/POT_validation.fcl
POT to Extinction Monitor
beam/POT_extmon.fcl
Beam
sim.mu2e.Beam.art
Charged at DS2
Truncated
sim.mu2e.Truncated.art
Fail G4Consistent
ExtMonRegion
sim.mu2e.ExtMonRegion.art
ExtMon
sim.mu2e.ExtMon.art
concatenate x1000/NThreads
Beam Splitting
beam/BeamSplitter.fcl
concatenate x50000
concatenate x1000
Muon Beam
sim.mu2e.MuBeamCat.art
Muons at DS2
Neutrals
sim.mu2e.NeutralsCat.art
Everything leaving PS or TS not via DS2
Electron Beam
sim.mu2e.EleBeamCat.art
Non-Muons at DS2
resample 10+3
Pileup
Muon Beam Resampling
pileup/MuBeamResampler.fcl
concatenate x1000/NThreads
IPA Stops
sim.mu2e.IPAStopsCat.art
Target Stops
sim.mu2e.TargetStopsCat.art
resample 10+3?
resample 10+3
Target Stop Muon Splitting
pileup/MuonStopSelector.fcl
concatenate x10
concatenate x1000
concatenate x1000/NThreads
Mu- Target Stops
sim.mu2e.MuminusTargetStopsCat.art prescale 1000
Mu+ Target Stops
sim.mu2e.MuplusTargetStopsCat.art
resample 10+3?
resample 10+3?
μ to X = [P, D, N,γ, e ν]
pileup/MuStopPileup.fcl
concatenate 10+4?
Target Muon Pileup
dts.mu2e.MuStopPileup.art
Muon BeamFlash Pileup
dts.mu2e.MuBeamFlashCat.art t > 250 ns || t < 10 ns
dts.mu2e.EarlyMuBeamFlashCat.art prescale 30
resample 10+4?
resample 10+4?
resample 10+4?
am Resampling
amResampler.fcl
concatenate x1000/NThreads
n BeamFlash Pileup
FlashCat.art t > 250 ns || t < 10 ns
BeamFlashCat.art prescale 1000

Primary production

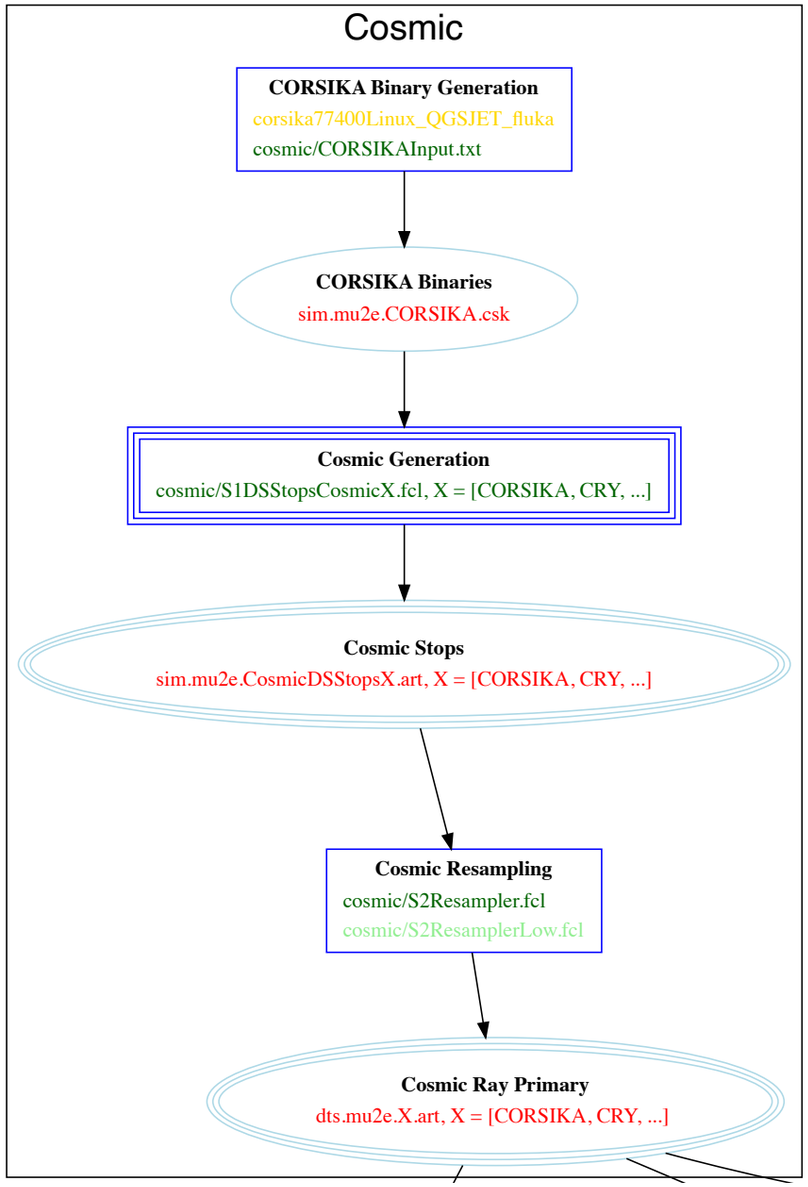
- The beam pileup production simulates the particles created by the **beam spill**, divided into three main streams: muons, neutrons, electrons.
- A Mu2e event is very busy: the muon beam impinging on the conversion target produces prompt backgrounds that need to be overlaid with the primary particle we want to simulate (e.g. a conversion electron or decay-in-orbit electron).
- This is done in the “primary” part of the campaign, which also performs detector simulation (“digitization”) and reconstruction.



Signal electron + background hits

Cosmic production

- Being Mu2e placed just below surface, the detector is constantly hit by cosmic rays, which represent the **largest experimental background**.
- In order to suppress this background, the solenoids are surrounded by a **Cosmic Ray Veto** (CRV).
- Simulating this background is extremely important to be able to estimate the sensitivity of the experiment and develop **calibration and alignment algorithms**.
- We've been using CRY in the past and we're now moving to **CORSIKA**, which provides a more accurate simulation.
- In order to produce the sky-time needed, we first submit a grid stage where we run the CORSIKA executable and perform the Geant4 simulation until the detector solenoid (DS)
- The events are then re-used N times as input for the second step of the simulation (from the DS border onwards).



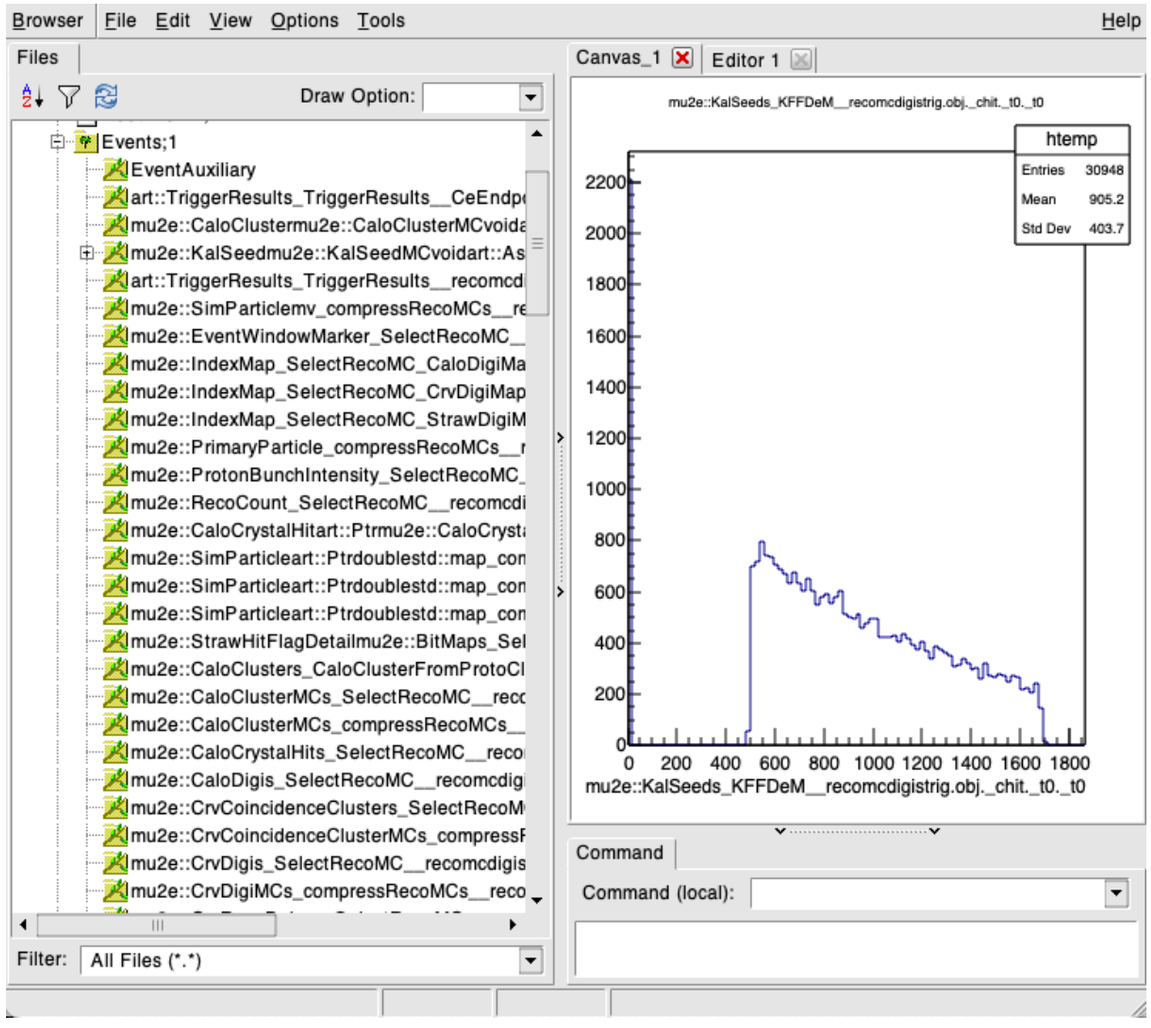
Multi-threading

- We use multi-threading for simulation stages where we run Geant4.
- Our Geant4 code (“Mu2eG4”) has a large static memory footprint: (1950 + N * 65) MiB where N is the number of threads.
- Using N > 1 allows to free memory for other jobs: memory mitigation.
- However, If you request a grid job with > 1 CPU then it may wait longer to match to a slot on the grid. We are being conservative, using only 2 threads per job.

# Threads	Avg Time (hh:mm:ss)			Max Time (hh:mm:ss)			Wall
	Wall	CPU	Efficiency	Wall	CPU	Efficiency	/Sequential
Sequential	1:41:36	1:37:47	96.2%	2:49:51	2:47:37	98.7%	100.0%
2	1:45:03	3:20:43	95.5%	3:21:59	6:15:01	92.8%	103.4%
4	1:48:19	6:49:04	94.4%	3:26:22	12:51:19	93.4%	106.6%
6	2:11:42	12:32:32	95.2%	3:35:52	20:32:10	95.1%	125.4%
8	2:04:46	15:38:18	94.0%	4:00:22	29:49:35	93.1%	115.2%

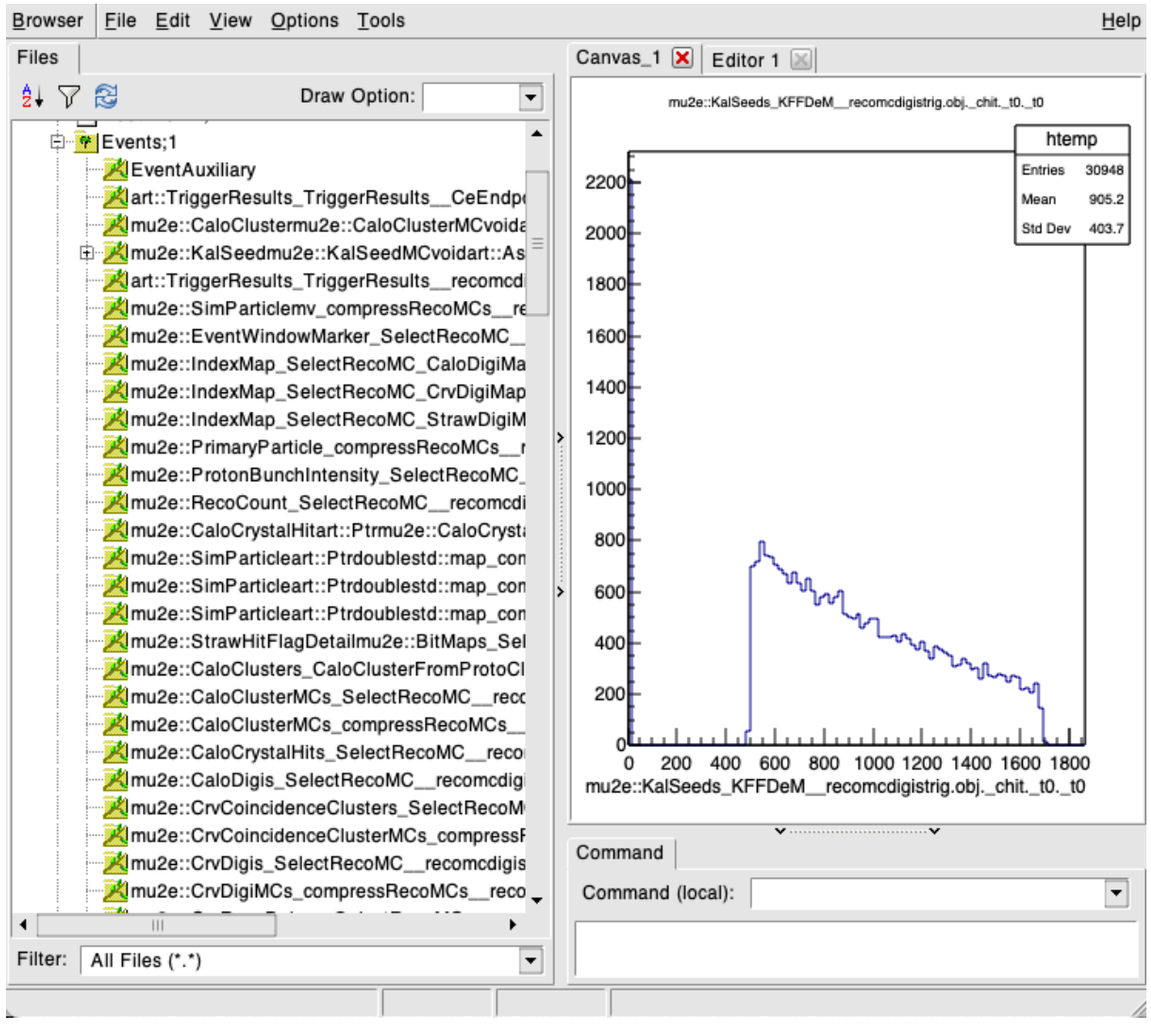
ROOT usage

- The outputs of each stage are stored in the **art-ROOT format**. The *art* objects we store in the output file can be very complex: treating them as normal TTrees to try to analyze the data quickly becomes unfeasible.
- The user has the option of writing their own *art* analyzer and save the information they need in simpler TTrees or directly into histograms.
- We also provide a “standard” analyzer called **TrkAna** which creates a TTree with several key information (reconstructed momentum, t0, track hits, etc.) and can be used to make quick comparisons or last-mile analyses.



ROOT usage

- The outputs of each stage are stored in the **art-ROOT format**. The *art* objects we store in the output file can be very complex: treating them as normal TTrees to try to analyze the data quickly becomes unfeasible.
- The user has the option of writing their own *art* analyzer and save the information they need in simpler TTrees or directly into histograms.
- We also provide a “standard” analyzer called **TrkAna** which creates a TTree with several key information (reconstructed momentum, t0, track hits, etc.) and can be used to make quick comparisons or last-mile analyses.



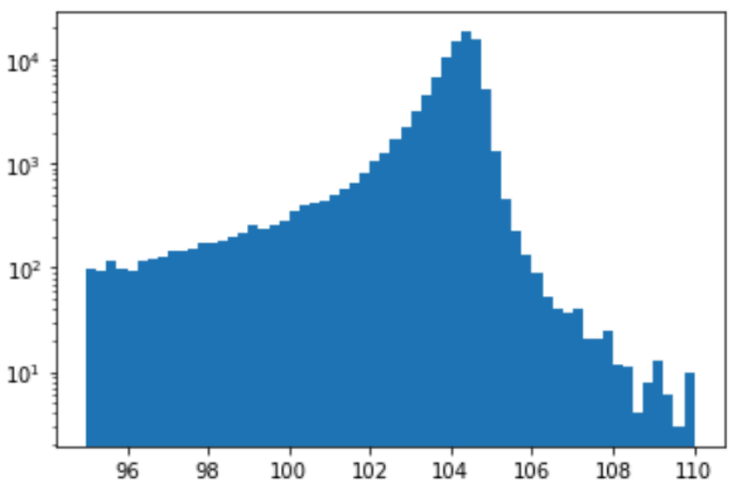
TrkAna

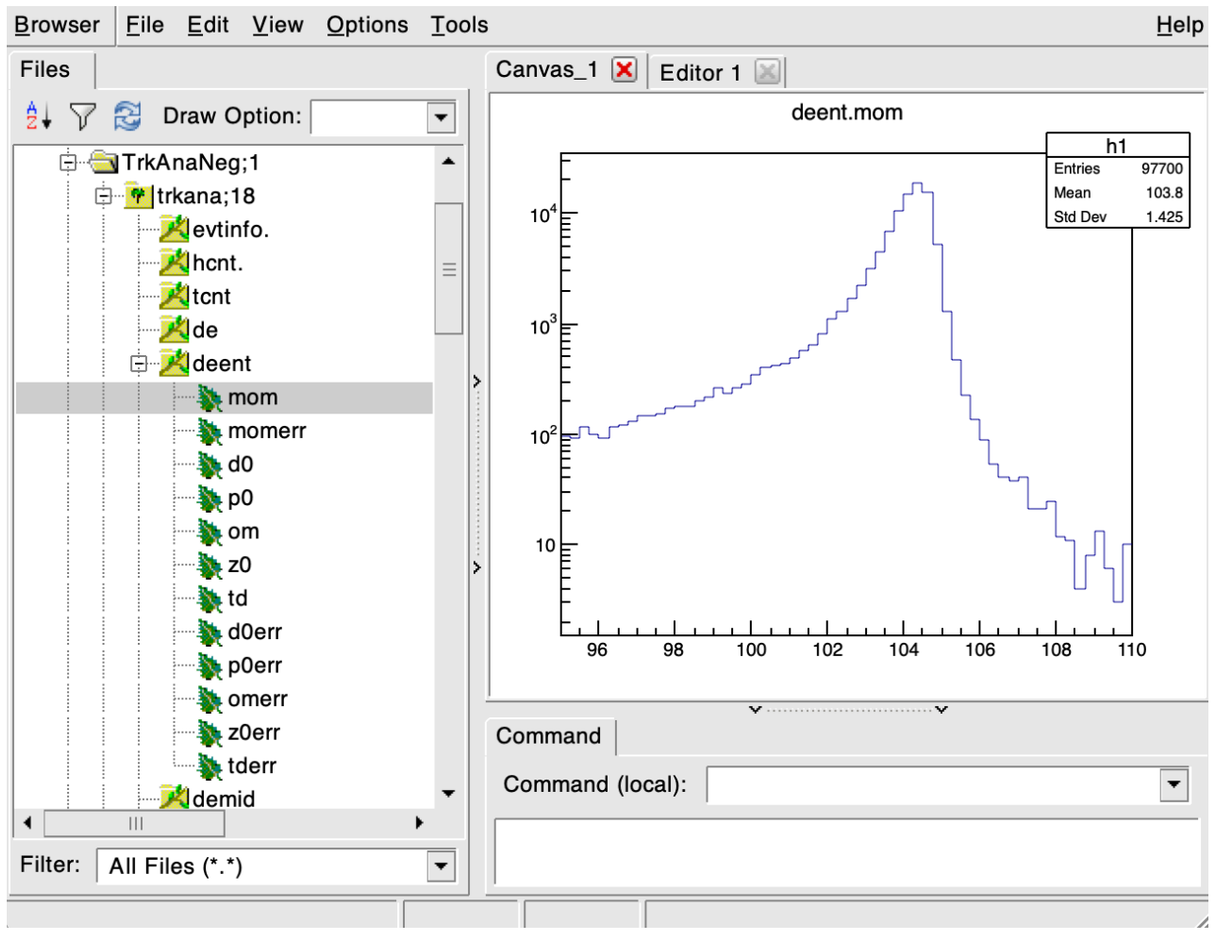
- The TrkAna output consists of a TTree that can be easily read both through ROOT macro and dedicated Python libraries (e.g. uproot).
- This makes easier to prototype and develop ML algorithms with **popular Python libraries** (TensorFlow, scikit-learn).

```
[1]: import uproot
     import matplotlib.pyplot as plt

[2]: file_ce = uproot.open("trkana.root")["TrkAnaNeg"]["trkana"]
     ce = file_ce.arrays(filter_name="deent")

[3]: fig, ax = plt.subplots(1,1)
     _=ax.hist(ce["deent"]["mom"],bins=60,range=(95,110))
     ax.set_yscale("log")
```





PyWrap

- In some cases, it would be helpful to **use a piece of C++ Offline code in a Python script**. A good example is importing the enums from Offline into a Python analysis routine, so numbers recorded in an ntuple could be interpreted.
- The wrappers are implemented using `swig`. This open-source package can interpret a C++ class header file and produce a piece of C++ code with hooks into the class, and piece of Python code which can use the hooks to present the C++ class to the user as a Python class.
- To create a wrapper, there must be a file, `pywrap.i` in the `src` directory where the source class is created. The content of this file is a bit abstruse, and in order to try to reproduce the features of C++ in Python, it can get complicated.

```
import MCDataProducts
gid = MCDataProducts.GenId()
gid.name(1)
```

```
particleGun
```