Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Federal Ministry of Education and Research

Hands-on Analysis Exercise

$H \to ZZ \to 4l$

with

Authors

Matteo Bonanomi, Philip Keicher

Daniel Savoiu, Ana Andrade

June 2024

This exercise was originally created for the Higgs PAG exercise at the CMS Physics Objects & Data Analysis School held in Hamburg in October 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction to ColumnFlow

ColumnFlow is intended as a back-end for analyses in order to facilitate processing large amounts of data. It is purely python-based and employs multiple packages that are well-received and -maintained in the HEP community. At the time of writing these instructions, the team of developer's purely consists of data analysts at the CMS experiment. Therefore, this exercise is structured accordingly. Please note that ColumnFlow is in principle designed in an experiment-agnostic way, such that it can also be extended to other use cases.

Additionally, please note that this hands-on exercise is not meant to fully document all available functionalities. The purpose of this exercise is to give an overview of the most fundamental aspects and concepts that are available at the time of writing. For a more comprehensive overview, please visit the official documentation [1]. In case of any questions are comments, feel free to contact the maintainers for example via the git repository [1].

1.1 General Structure

The guiding principle of ColumnFlow is that all analyses share basic work packages that need to be done when processing data. Examples for such packages could be the calibration of relevant objects, applying selections to define a fiducial phase space for the analysis or the calculation of some sensitive observables, which are discussed in more detail in later chapters of this document. ColumnFlow defines these work packages as law tasks, which can define dependencies amongst each other and will only run necessary tasks to obtain the requested output.

Figure 1.1 depicts an overview of the available tasks and their dependencies. The highlighted regions indicate use cases that are discussed in chapter 2. This chain of jobs starts with obtaining the list of logical file names (LFNs) that contain the events to be analysed in a flat tuple format (e.g. the nanoAOD format within CMS). The first block is dedicated to prepare these events for further analysis. Such a preparation can entail different things, such as a calibration of the relevant objects in an analysis or the application of selection criteria to define a relevant work

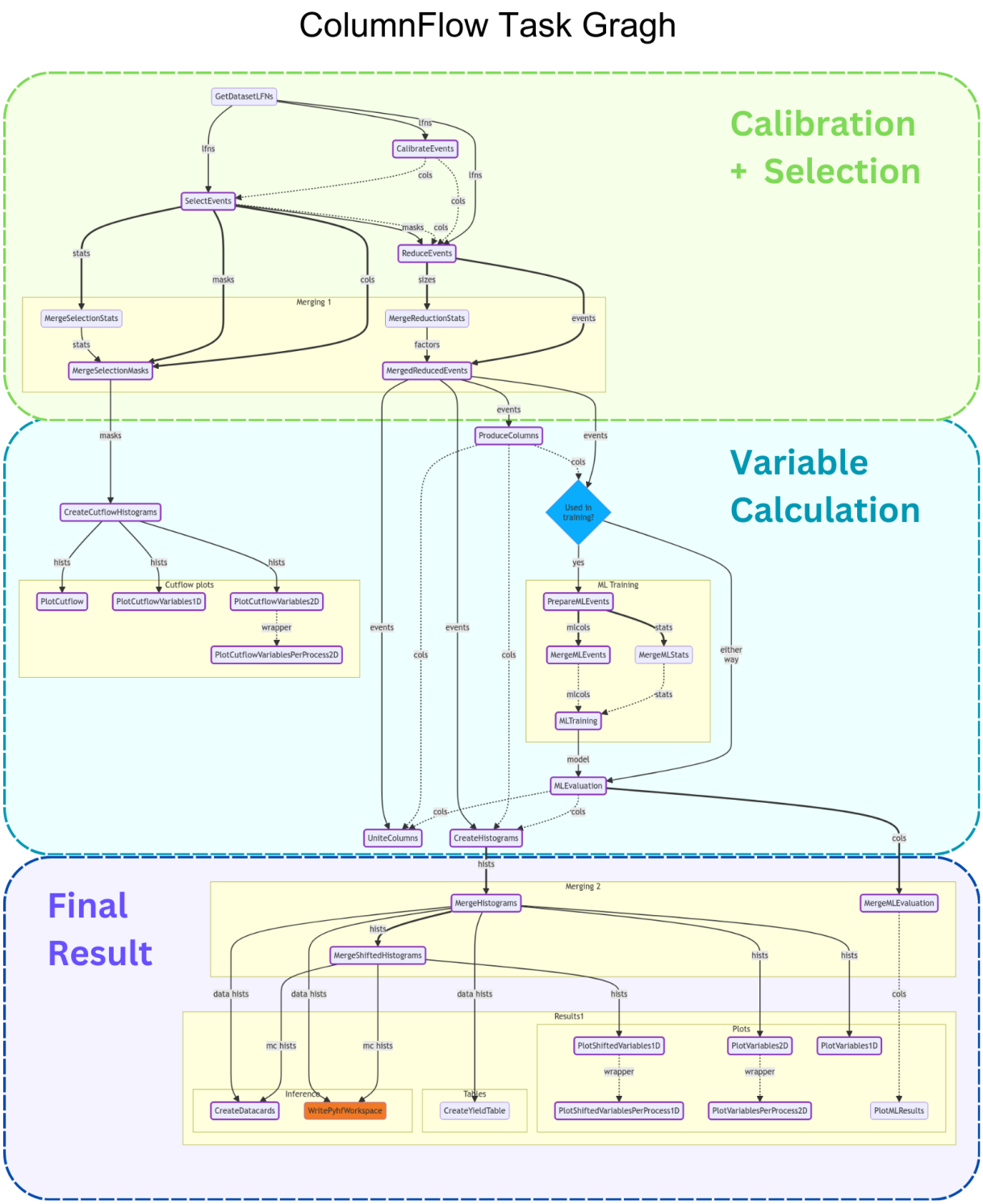


Figure 1.1: **ColumnFlow task graph hierarchy**. The tasks are arranged in three sections that correspond to general work packages when analysing data. The line strengths and styles indicate the behaviour when propagating information between tasks. For more information, please consider ref. [1].

space. In order to facilitate a more efficient calculation in later parts of this workflow, the amount of data is reduced as a last step of the first plot.

The second block illustrated in fig. 1.1 is dedicated to the calculation of different observables and metrics. At the time of writing these instructions, this blocks offers metrics such as a summary of efficiencies for different stages of the selection(s) and their effect on observables, the calculation of completely new variables and also more complex calculations based on machine learning. Moreover, it offers the functionality to collect all information of the workflow and save it as a flat tuple in the e.g. ROOT or parquet format. The modular structure of the individual tasks allows for an easy extensions to calculate a variety of observables.

Finally, the last block is dedicated to the final quantities that are needed for the analysis. Most of these endpoints of the workflow aim to facilitate a data analysis in a binned format, though this is not a hard criterion. This includes producing figures illustrating one- or two-dimensional distributions of multiple physics processes under consideration of a wide variety of systematic uncertainties, as well as the input needed for a statistical inference based on the data (e.g. datacards for the Combine tool within CMS).

This structure allows for a full end-to-end analysis. The explicit definition of dependencies in the code and the implicit check for existing outputs provided by luigi and law result in a sustainable and reproducible workflow that is easily triggered with a single command. In the following, these capabilities are illustrated using an example that is based on the $H \rightarrow 4l$ analysis, for which we will build a selection of the aforementioned modules. Please note that this example is by no means as complex and sophisticated as the real CMS analysis, and should therefore not be expected to yield the same results.

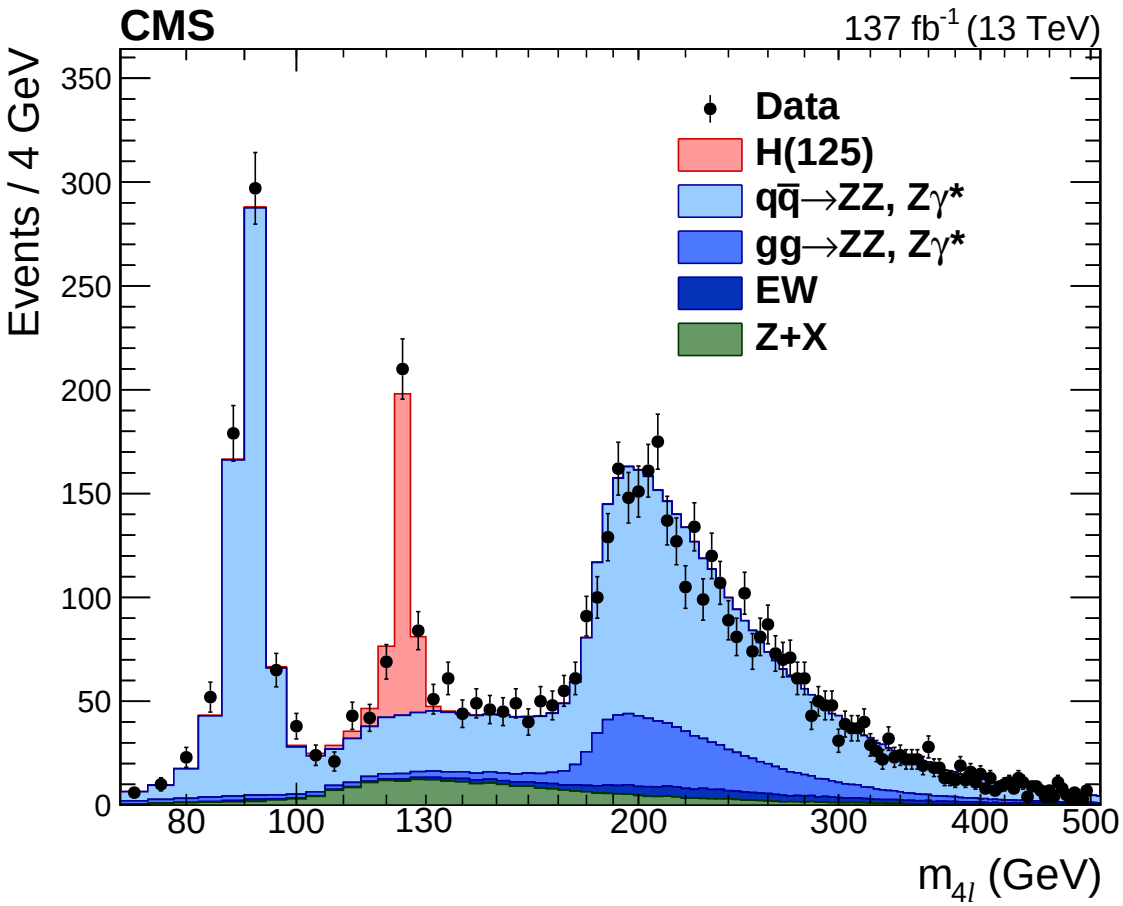


Figure 1.2: **Reconstructed four-lepton invariant mass m_{4l} with full Run2 data**. The SM Higgs boson signal with $m_H = 125\,\text{GeV}$, denoted as $H(125)$, and the ZZ backgrounds are normalized to the SM expectation. The $Z + X$ background is normalized to the estimation from data. Figure taken from ref. [2].

1.2 Physics example: $H \to ZZ \to 4l$

The goal of this exercise is to reconstruct the Standard Model (SM) Higgs boson mass, using a selection targeting the four-lepton final state. This is considered a *golden* channel to rediscovered the Higgs because:

- there is a **large signal to background ratio** – it is easy to discriminate between the peak of the reconstructed four-lepton mass (m_{4l}) and the overall flat background shape;
- we have excellent **mass resolution** – thanks to the great resolution power of CMS, we have optimal shape reconstruction of m_{4l};
- it is a **resolved final state** – detection of the four leptons in the final state ensures good discrimination of signal and background.

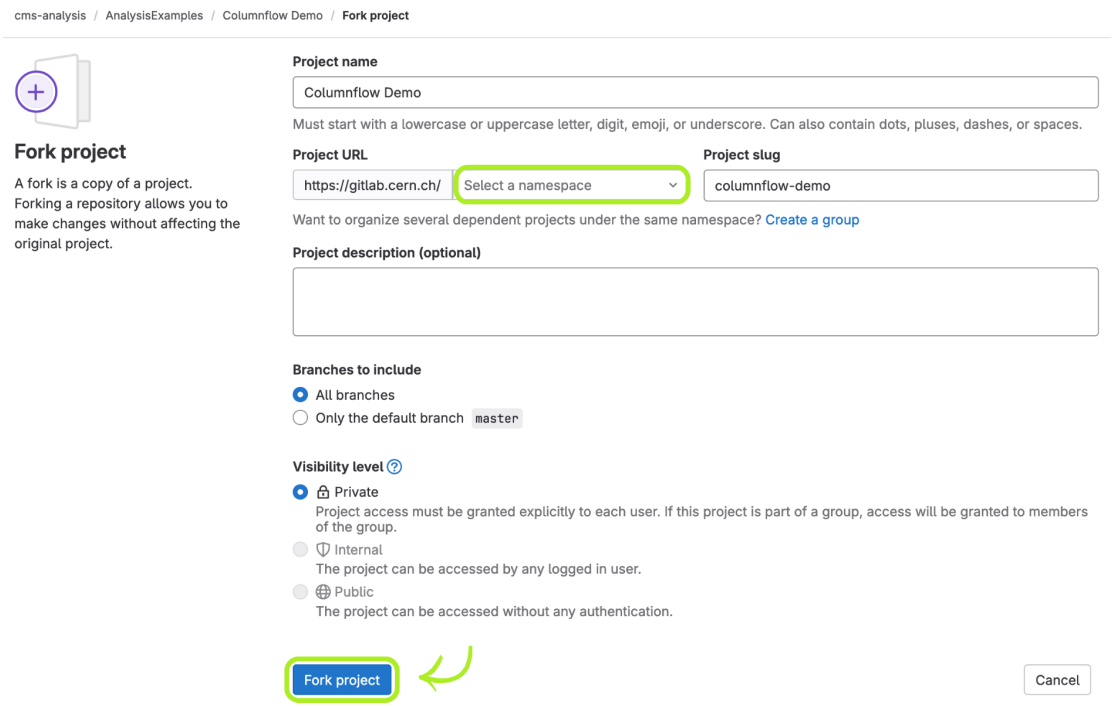
1.3 Installation & Setup

> Note: ColumnFlown only runs on Linux and may require up to 4 GB of disc space.
>
> Also, the machine where you run this exercise must be mounted with CERN AFS.

Start by going to the GitLab repository of this exercise:

https://gitlab.cern.ch/cms-analysis/analysisexamples/columnflow-demo

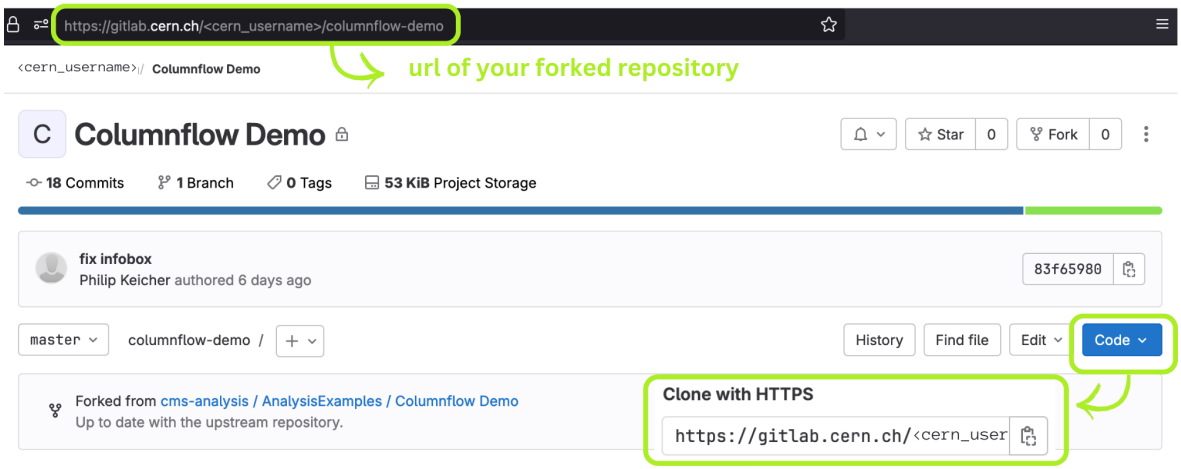
To have your own copy of the code, fork the repository into your personal area. You can do this by clicking the `Fork` button on the upper right corner of the page. To set your Project URL please type your CERN username in the `Select a namespace` option.



After clicking the `Fork project` button, your fork url should be:

`https://gitlab.cern.ch/<cern_username>/columnflow-demo`

In your forked project, go to the `Code` button on the right hand side of the page and copy the address under the `Clone with HTTPS` option. If you have an SSH key registered on GitLab prior to this exercise, you can also use the `Clone with SSH` option.



Next, open a new terminal window and clone your code to your machine by running one of the following commands (depending on which cloning method you chose):

```
git clone --recursive https://gitlab.cern.ch/<cern_username>/columnflow-
    demo.git
```

```
git clone --recursive ssh://git@gitlab.cern.ch:7999/<cern_username>/
    columnflow-demo.git
```

The directory you have thus created will be referred to as `basedir`. You can now go inside your local repository and install ColumnFlow. The `setup.sh` bash script will initialize the software environment with `micromamba`. Here, we define `dev` as the setup name, but you are free to name it as you wish.

```
cd columnflow-demo
source setup.sh dev
```

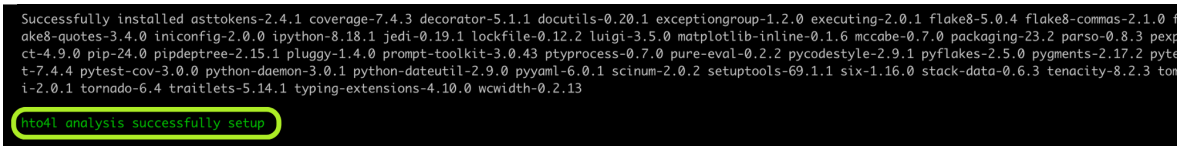
You will be asked to define a series of variables, the first of which is your CERN username. For all other variables you can keep the default name by just pressing `Enter`. Variables specific to this exercise will start with `H4L_`, while ColumnFlow specific variables start with `CF_`. You can find all variables in the `.setups/dev.sh` bash file. We invite you to check out this file and familiarize yourself with these variables.

```
CERN username (CF_CERN_USER, default         ): <cern_username>
Local data directory (CF_DATA, default ./data): <Enter>
Relative path used in store paths (see next queries) (CF_STORE_NAME, default h4l_store): <Enter>
Default local output store (CF_STORE_LOCAL, default $CF_DATA/$CF_STORE_NAME): <Enter>
Local directory for caching remote files (CF_WLCG_CACHE_ROOT, default $CF_DATA/h4l_cache): <Enter>
Local directory for installing software (CF_SOFTWARE_BASE, default $CF_DATA/software): <Enter>
Local directory for storing job files (CF_JOB_BASE, default $CF_DATA/jobs): <Enter>
Use a local scheduler for law tasks (CF_LOCAL_SCHEDULER, default True): <Enter>
Install and bundle CMSSW sandboxes for job submission? (H4L_BUNDLE_CMSSW, default True): <Enter>

variables written to /afs/desy.de/user/a/alvesand/dust/columnflow-demo/.setups/dev.sh

installing conda with micromamba interface at /afs/desy.de/user/a/alvesand/dust/columnflow-demo/data/software/conda
initialized conda with micromamba interface and python 3.9

setting up conda / micromamba environment
```

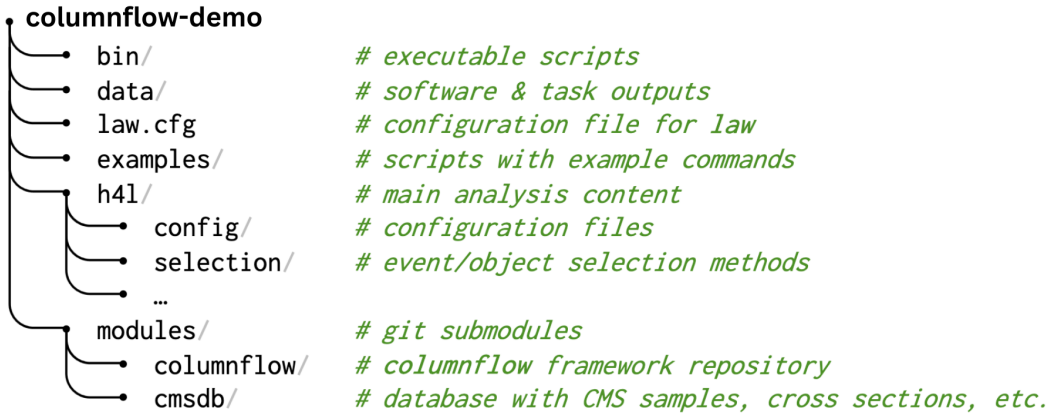
Note that the first installation of the software can take up to several minutes.

Every time you want to work with ColumnFlow (e.g. if you open a new terminal window), you will need to source the `setup.sh` script again.

Once the installation is complete you should see a line of green text stating that the analysis has been successfully set up. You are now ready to start working with ColumnFlow!



Inside of your newly created `columnflow-demo` directory, you will find the following project structure:



This exercise is organized in the form of `law` tasks, where different tasks create some form of output. By default, these tasks will save their output on a remote file system (e.g. `WLGC`), for which you will require a `voms-proxy`. If you would like to save certain/all outputs locally, we recommend to create a directory on a system with a larger amount of disk space (e.g. `EOS`). For such cases, you will need to update the `law.cfg` file accordingly. You can view the available tasks by running:

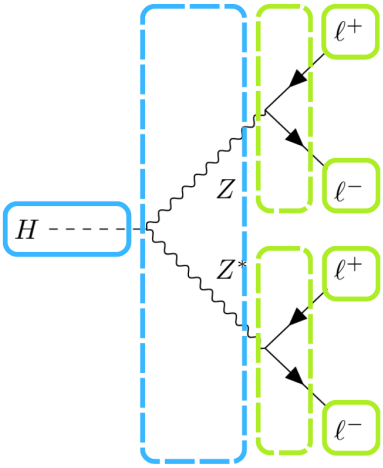
```
law index --verbose
```

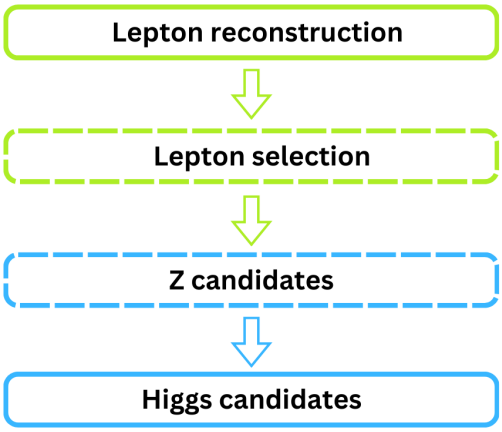
This exercise will focus on the following tasks:

- `cf.CalibrateEvents` / `cf.SelectEvents`
- `cf.ProduceColumns`
- `cf.PlotCutflow`
- `cf.PlotVariables1D` / `cf.PlotVariables2D`
- `cf.CreateDatacards`

1.4 Analysis strategy

In order to find Higgs boson candidates we will first need to reconstruct the four leptons in the final state. We will do this by writing a `Selector` (section 2.3).





Chapter 2

Basic Functionalities

2.1 The Mother of all: TaskArrayFunctions

2.2 Writing a Calibrator

2.3 Writing a Selector

The lepton selection we want to implement in this exercise is the following:

Loose Electrons

-

2.4 Writing a producer

Chapter 3

Advanced Topics

3.1 Defining Categories

3.2 Writing datacards

Bibliography

[1] The ColumnFlow Team. *The ColumnFlow project*. Version 3b8e0f3. Documentation available at `https://columnflow.readthedocs.io/en/latest/`. 2024. URL: `https://github.com/columnflow/columnflow`.

[2] The CMS Collaboration. *Measurements of production cross sections of the Higgs boson in the four-lepton final state in proton–proton collisions at* $\sqrt{s} = 13\,TeV$. June 2021. DOI: `10.1140/epjc/s10052-021-09200-x`. URL: `http://dx.doi.org/10.1140/epjc/s10052-021-09200-x`.