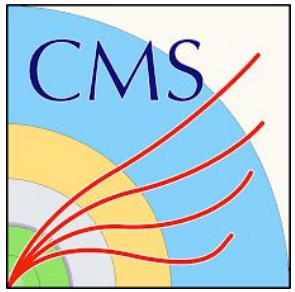
RICE

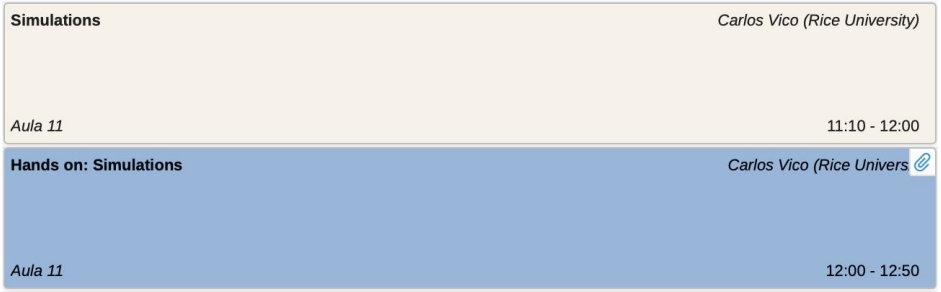


4th COMCHA school

MC simulations

Carlos Vico

Thanks



- **Thanks a lot for inviting me today to give a seminar on the usage of MC simulations in High Energy Physics!**
  - We experimentalists often rely too much on the "deliveries" of the experiments.
  - Someone in the collaboration produces an ingredient that I use in my analysis… depending on how much work is there to do that week, I might not give it too much thought on how that particular ingredient has been created ;)

Well, when it comes to simulations, we might want to be aware at all times of what we are doing → in the end, it's pure physics!

Introduction

- **So what are we going to discuss in this seminar?**
  - Mostly we are going to give an overview of the different state-of-the-art MC generators available in the market.
  - Even though there are two sessions:
    - One hour session that should be mostly theory-driven
    - One hour session that should be mostly hands-on driven
  - I (personally) believe that MC simulations should be mostly hands-on.
  - I'm going to avoid relying (too) much on slides → and instead I've prepared several notebooks that we can go through along the time of the seminar.
- **Everyone would benefit from active discussion and questions!**
  - And, any of you are welcome to play around with the notebooks (in the end, that is kind of the purpose of knowing how to do MC simulations!).
- **I will however try to dedicate at least 40-50 minutes to an exercise that will be detailed after.**

Warning

Tips

- **Several ways to run a notebook:**
  - **from lxplus + vscode (recommended)**: internally built (that's how I do it). You have to download the notebooks and the code.tar.xz file attached to the indico to your lxplus area.
  - **from swan** (recommended): CERN-IT supported platform for running notebooks. You have to upload the notebooks and the code.tar.xz file attached to the indico.
  - **from Google Collab**: same as SWAN, but with Google support.
  - **standalone run:** the jupyter notebooks are not really needed in this exercises, as most of the cells are bash scripts. One can simply open the notebooks and try to run the code from a terminal (by copy-pasting).
    - It might be better than swan or google collab (sometimes they lose connection or run out of space).

Tips

- **Unfortunately, doing MC simulations relies very much on external libraries. It is difficult to prepare against all odds of this working for everyone.**

- **I would invite you to download the notebooks in the following minutes (while I explain some basic ideas on MC) and try to execute the first cell of the "`setup.ipynb`" notebook; so that we can have some work done by the time we need to play around with the notebooks.**

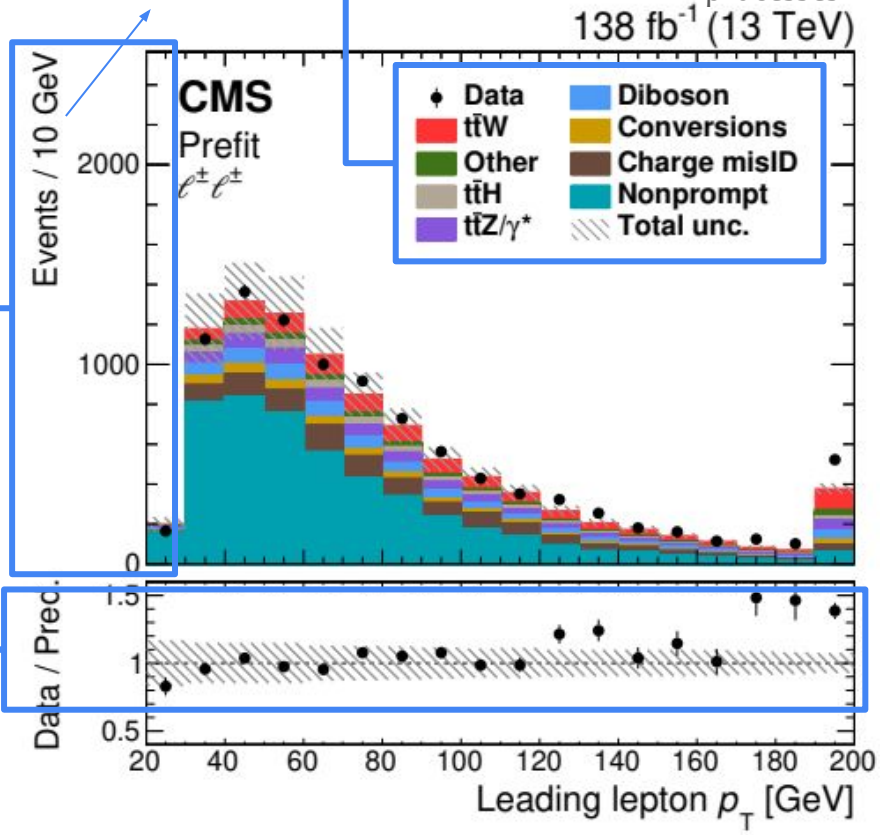
- **Alternatively, if lxplus is an option, you should be able to make use of my own installation by adding my public /eos/:**
  - let me know of your CERN user for this.

Let’s get started

Introduction

Considered signal (ttW) and background processes.

Number of recorded/simulated events that enter our phase space.

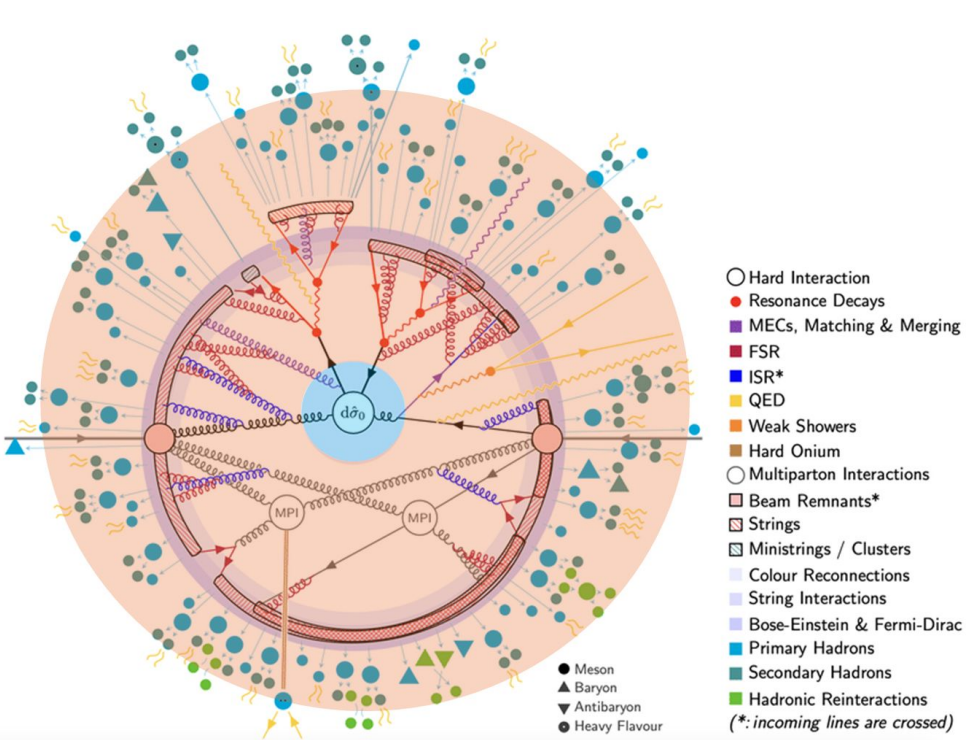


Ratio between Data and total prediction (signal + bakcground)

Thanks to simulating events we are able to directly test the theory against the data.

Observable that we are looking at

Introduction



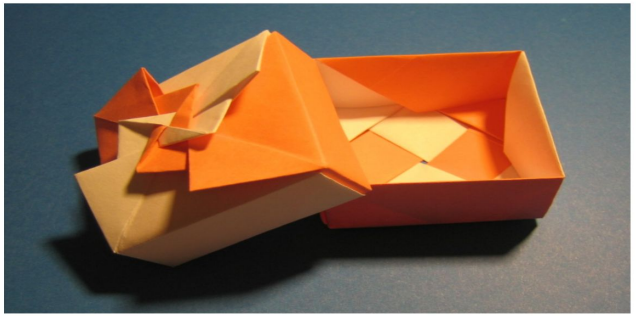
- In order to simulate one data event we need to simulate all these processes.
- What is highlighted in blue is what we call *hard scattering*. And it's simulated with softwares like `Madgraph5_aMC@NLO` or `Powheg`. The physics of the hard scatter is typically more precise and slightly less complicated.
- The orange part is what we call *parton showering*, and it's typically done *with* `Pythia8` o `Herwig`. **Very complicated physics with less accuracy.**
- But we have to simulate everything → because all of this is happening in the real collisions!

Once we have simulated all these processes, we can replicate the process of reconstruction that we apply to our data.

How to model the hard scatter

Hard scatter generators

The POWHEG BOX

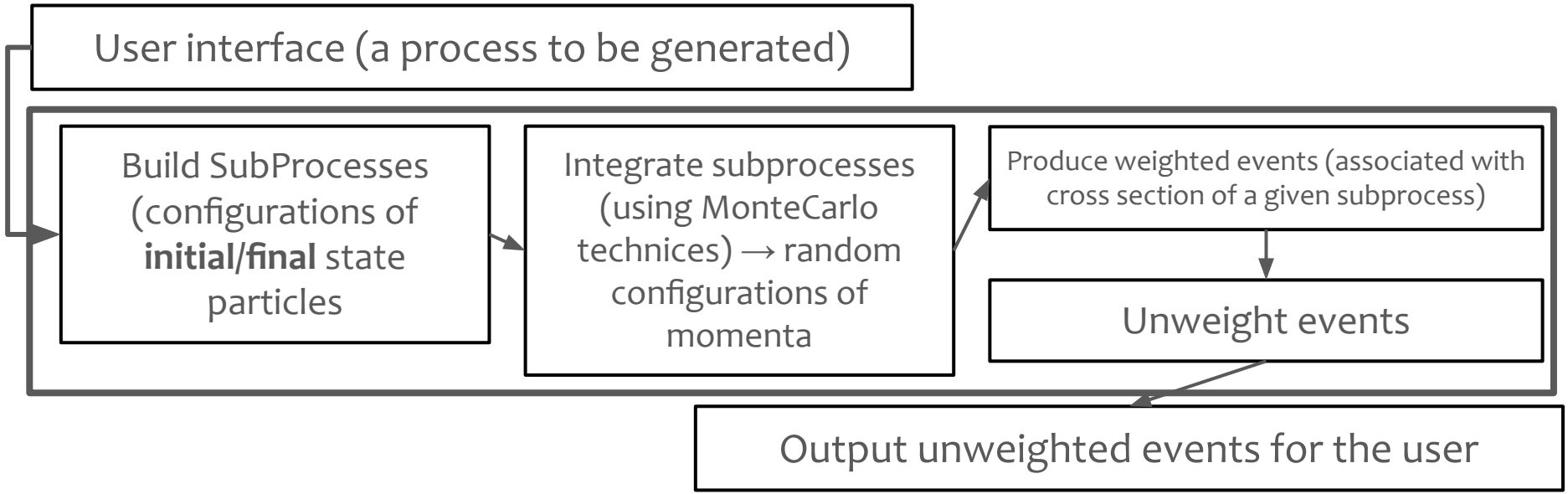


⚠ Important Notice: We are migrating from SVN to Git by the end of June 2025.
Starting July 1st 2025, the SVN repository will become read-only (until further notice).
The Git repository is already available at GitLab.
Please visit this page for more details and instructions on migrating from SVN to Git.

- **Madgraph5_aMC@NLO and POWHEG are two of the most widely used matrix-element generators used by the different collaborations of the LHC (it is particularly important in CMS and ATLAS)**
  - Their main feature is that they can perform calculations up to Next-to-leading order in perturtbative QCD.
  - Sometimes even Next-to-next-to leading order (NNLO) in QCD, and even NLO in EW (this is mostly in POWHEG).

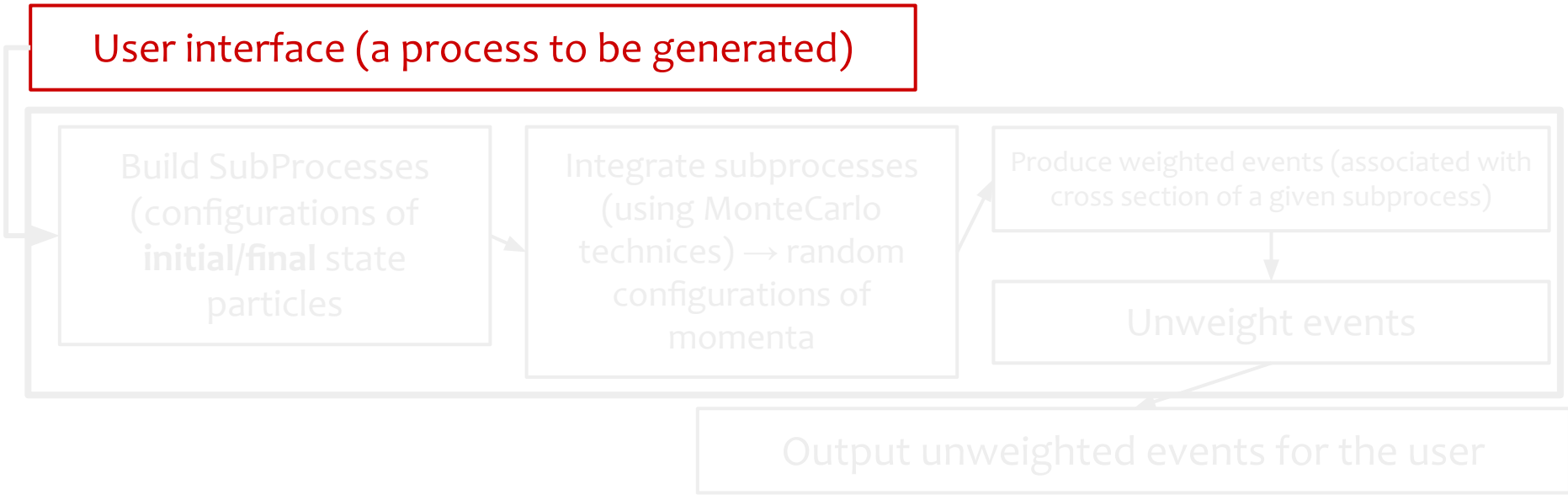
Madgraph's procedure for MC generation

- **Note: feel free to open the `01_Madgraph_Generation.ipynb` notebook**
- **MadGraph automates the calculation of Feynman diagrams**
  - It computes tree-level matrix elements (MEs) for any specified process
  - Think of it as a sophisticated calculator that exhaustively enumerates every possible way particles can interact at the most fundamental level.



Madgraph's procedure for MC generation

- **Note: feel free to open the `01_Madgraph_Generation.ipynb` notebook**
- **MadGraph automates the calculation of Feynman diagrams**
  - It computes tree-level matrix elements (MEs) for any specified process
  - Think of it as a sophisticated calculator that exhaustively enumerates every possible way particles can interact at the most fundamental level.



Madgraph's process card

- **Being an automated framework, MadGraph relies on a minimal set of inputs, and off loads the whole building of Feynman diagrams, feynman rules, and integration of the matrix elements to internal libraries**
  - Instead of actually implementing these calculations ourselves: we communicate with MadGraph by creating configuration files

cards > madgraph > WtoLNu_lo > WtoLNu_lo_proc_card.dat

```
1  import model sm
2  generate p p > w+ > l+ vl
3  add process p p > w+ > l+ vl
4  output WtoLNu_lo
```

Model to be used (MadGraph internal)

Process to be integrated

Folder with the outputs

The card from above will instruct madgraph to generate **onshell** W bosons, that subsequently decay into leptons. **This is the most important card!**

Madgraph's run card

cards > madgraph > WtoLNu_lo > WtoLNu_lo_run_card.dat

```
#*********************************************************************
#
#*********************************************************************
# Tag name for the run (one word)                                    *
#*********************************************************************
  tag_1     = run_tag ! name of the run
#*********************************************************************
# Number of events and rnd seed                                      *
# Warning: Do not generate more than 1M events in a single run       *
#*********************************************************************
  3000 = nevents ! Number of unweighted events requested
  0    = iseed   ! rnd seed (0=assigned automatically=default))
#*********************************************************************
# Collider type and energy                                           *
# lpp: 0=No PDF, 1=proton, -1=antiproton, 2=photon from proton,      *
#                3=photon from electron, 4=photon from muon          *
#*********************************************************************
      1        = lpp1    ! beam 1 type
      1        = lpp2    ! beam 2 type
      6500     = ebeam1  ! beam 1 total energy in GeV
      6500     = ebeam2  ! beam 2 total energy in GeV
# To see polarised beam options: type "update beam_pol"

#*********************************************************************
# PDF CHOICE: this automatically fixes also alpha_s and its evol.    *
#*********************************************************************
      'lhapdf'    = pdlabel     ! PDF set
      325300 = lhaid     ! if pdlabel=lhapdf, this is the lhapdf number
# To see heavy ion options: type "update ion_pdf"
```

The **run card** is actually used to control parameters of the generation:

- Number of events.
- Center-of-mass energy.
- Proton pdfs.
- Generator level cuts on output particles.

```
# BW cutoff (M+/-bwcutoff*Gamma) ! Define on/off-shell for "$" and decay
#*********************************************************************
  15.0  = bwcutoff      ! (M+/-bwcutoff*Gamma)
#*********************************************************************
# Standard Cuts                                                      *
#*********************************************************************
# Minimum and maximum pt's (for max, -1 means no cut)                *
#*********************************************************************
 0.0  = pta        ! minimum pt for the photons
 0.0  = ptj        ! minimum pt for the jets
 0.0   = ptl       ! minimum pt for the charged leptons
 -1.0  = ptamax    ! maximum pt for the photons
 -1.0  = ptjmax    ! maximum pt for the jets
 -1.0  = ptlmax    ! maximum pt for the charged leptons
 {} = pt_min_pdg ! pt cut for other particles (use pdg code). Applied on particle and anti-particle
 {} = pt_max_pdg ! pt cut for other particles (syntax e.g. {6: 100, 25: 50})
#
# For display option for energy cut in the partonic center of mass frame type 'update ecut'
#
#*********************************************************************
# Maximum and minimum absolute rapidity (for max, -1 means no cut)   *
#*********************************************************************
 -1.0 = etaa    ! max rap for the photons
 -1.0 = etaj    ! max rap for the jets
 -1.0 = etal    ! max rap for the charged leptons
 0.0  = etalmin ! main rap for the charged leptons
 {} = eta_min_pdg ! rap cut for other particles (use pdg code). Applied on particle and anti-particle
 {} = eta_max_pdg ! rap cut for other particles (syntax e.g. {6: 2.5, 23: 5})
#*********************************************************************
# Minimum and maximum DeltaR distance                                *
```

Madgraph's process card

- **Madgraph's process cards have a very specific syntax that allows one to make calculations at many different levels**
  - Leading order (LO):
    - this is the example from the previous slide
  - Next-to-LO (NLO):
    - By adding the [QCD] tag at the end of the processes, madgraph is instructed to include NLO corrections in the matrix elements. This includes "virtual" corrections.
  - LO + additional jets:
    - By repeating the process lines and adding a "j" at the end, madgraph gets instructed to include "real" emission corrections in the matrix elements.
  - NLO + additional jets
    - Includes both "virtual" and "real" corrections in the matrix elements. As accurate as MadGraph can be.

Check out the notebook to see how to implement these

The madgraph notebook

Madgraph's notebook

MadGraph Event Generation

This notebook walks through the **hard-process event generation** stage of the MC tuning pipeline: from writing MadGraph cards to producing compressed LHE files that will be passed to PYTHIA8 for parton shower.

Physics process

We simulate the **Single W process** at the LHC ($\sqrt{s} = 13$ TeV):

$$pp \to W^{\pm} \to \mu^{\pm}\nu$$

We are going to see the different features that one can integrate within the MadGraph framework to achieve a different level of accuracy in describing relevant observables for W bosons produced at the LHC.

The usage of cards

As mentioned during the lectures, MadGraph is an automated framework that relies on the definition of a set of inputs (known as `cards`) to build up the different feynman diagramas contributing to this process. Once the different contributions have been constructed, it automatically integrates all the feynman diagrams (this process is known as *matrix-element integration*), we can reuse those calculations to generate more events at different regions of the integrated phase space.

Madgraph's notebook

Card structure

In this project, each process requires three cards under `cards/<process-name>/`:

Card file	Purpose
`<name>_proc_card.dat`	Defines the process and selects the model (SM)
`<name>_run_card.dat`	Sets $\sqrt{s}$, PDF set, jet p_T threshold (`xqcut`), $m_{\ell\ell}$ cut, number of events, etc.
`<name>_customizecards.dat`	Optional overrides of model parameters (e.g. m_Z, Γ_Z)

The usage of gridpacks

Running MadGraph interactively for every event is impractical. Instead, LHC experiments use **gridpacks**: a self-contained tarball that includes:

- The compiled matrix-element code (Fortran, auto-generated by MadGraph).
- All configuration cards (`proc_card`, `run_card`, `customizecards`).
- A thin shell wrapper (`run.sh`) that accepts a seed and event count.

Once the gridpack is built, producing events is fast, parallelisable, and fully reproducible: just unpack and run with a different random seed per job.

What you will do in this notebook

1. **Inspect** the reference cards in `cards/W/` and understand the relevant settings.
2. **Prepare** your own set of cards for the target process under `cards/`.
3. **Build** the MadGraph gridpack using the `create_gridpack` helper.
4. **Run** the gridpack locally to produce a compressed LHE file (`batch<N>.lhe.gz`).
5. **Inspect** the LHE output to verify the event content before moving to the shower step.

Madgraph's notebook

Setup

First of all, let's make sure we have all the required environments loaded by executing the following cell.

In [21]:
```
%%bash
if [[ ! -d bin/ ]]; then tar -xvf code.tar.xz; fi
source setup.sh
source setup.sh madgraph
pip install six
```

```
Usage: bash [install]
MadGraph MG5_aMC_v2.9.27 is already installed at /eos/home-i04/c/cvicovil/SWAN_projects/COMCHA/libraries/MG5_aMC/
```

Preparing a basic W card

We are going to now create the three cards that we need in order to generate the madgraph cards for W production. To do that, we start with the process card. We can write a simple python snippet that dumps the process card in the cards directory using the following function.

Madgraph's notebook

In [22]:
```
import os
def prepare_process(
    sample_name,
    output_dir,
    processes,
    order ):

    # Create the output directory if it doesn't exist
    output_dir = os.path.join( output_dir, sample_name )
    os.makedirs( output_dir, exist_ok=True )

    model = "sm" if order == "lo" else "loop_sm"

    proc_card = [
        "import model {}".format(model),
    ]

    proc_card.append(
        "generate {} {}".format( processes[0], "" if order == "lo" else " [QCD]" )
    )
    for proc in processes[1:]:
        proc_card.append(
            "add process {} {}".format( proc, "" if order == "lo" else " [QCD]" )
        )

    proc_card.append("output {}".format( sample_name ) )

    with open( os.path.join( output_dir, "{}_proc_card.dat".format(sample_name) ), "w" ) as f:
        f.write( "\n".join(proc_card) )

    # For the customize and run cards, we can just copy the templates
    # and edit them later if needed
    os.system( "cp templates/customize_card.dat {output_dir}/{sample_name}_customize_card.dat".format(order=order, output_dir=output_dir, sample_name=sample_name) )
    os.system( "cp templates/{order}_run_card.dat {output_dir}/{sample_name}_run_card.dat".format(order=order, output_dir=output_dir, sample_name=sample_name) )


# Let's start with W production at LO decaying into muons
processes = [
    "p p > w+ > mu+ vm",
    "p p > w- > mu- vm~"
]

prepare_process(
    sample_name = "WtoLNu_lo",
    output_dir = "cards/madgraph/",
    processes = processes,
    order = "lo"
)
```

Madgraph's notebook

```
[cvicovil@lxplus917 COMCHA]$ tree cards/madgraph/WtoLNu_lo/
cards/madgraph/WtoLNu_lo/
├── WtoLNu_lo_customize_card.dat
├── WtoLNu_lo_proc_card.dat
└── WtoLNu_lo_run_card.dat
```

```
import model sm
generate p p > w+ > mu+ vm
add process p p > w- > mu- vm~
output WtoLNu_lo
```

LO

Madgraph's process card

Other available modes

```
import model loop_sm
generate p p > w+ > mu+ vm  [QCD]
add process p p > w- > mu- vm~  [QCD]
output WtoLNu_nlo
```

NLO

```
import model sm
generate p p > w+ > mu+ vm
add process p p > w+ > mu+ vm j
add process p p > w- > mu- vm~
add process p p > w- > mu- vm~ j
output WtoLNu_lo_with1Jet
```

LO+1j

```
import model loop_sm
generate p p > w+ > mu+ vm  [QCD]
add process p p > w+ > mu+ vm j  [QCD]
add process p p > w- > mu- vm~  [QCD]
add process p p > w- > mu- vm~ j  [QCD]
output WtoLNu_nlo_with1Jet
```

NLO+1j

Madgraph's notebook

Now we can create a gridpack

Inspect the cards from the last cell execution:

- How can you tell this is a Leading order calculation?

Execute the next cell to create a gridpack for this process.

In [*]:
```
%%bash
# Load helper functions (create_gridpack) and environment variables
source setup.sh
# Define here the name of your cards folder (relative to the cards/ directory)
MY_FOLDER_NAME="WtoLNu_lo"

source .venv/bin/activate
pip3 install six

# Actually run the gridpack
create_gridpack "cards/madgraph/${MY_FOLDER_NAME}"
```

```
Usage: bash [install]
Requirement already satisfied: six in ./.venv/lib/python3.9/site-packages (1.17.0)
Detected generator: madgraph
MadGraph order: lo
MadGraph LO - compiling process in /eos/home-i04/c/cvicovil/SWAN_projects/COMCHA/PROCESSES/WtoLNu_lo

WARNING:root:Support for Python3.9 (and below) has been dropped since end of 2025. Please consider update your version of Python.

************************************************************
*                                                          *
*                    W E L C O M E to                      *
*              M A D G R A P H 5 _ a M C @ N L O           *
*                                                          *
*                                                          *
*                *                   *                     *
*                  *       * *       *                     *
*                    * * * * 5 * * * *                       *
*                  *       * *       *                     *
*                *                   *                     *
*                                                          *
*         VERSION 2.9.27              2026-01-05           *
```

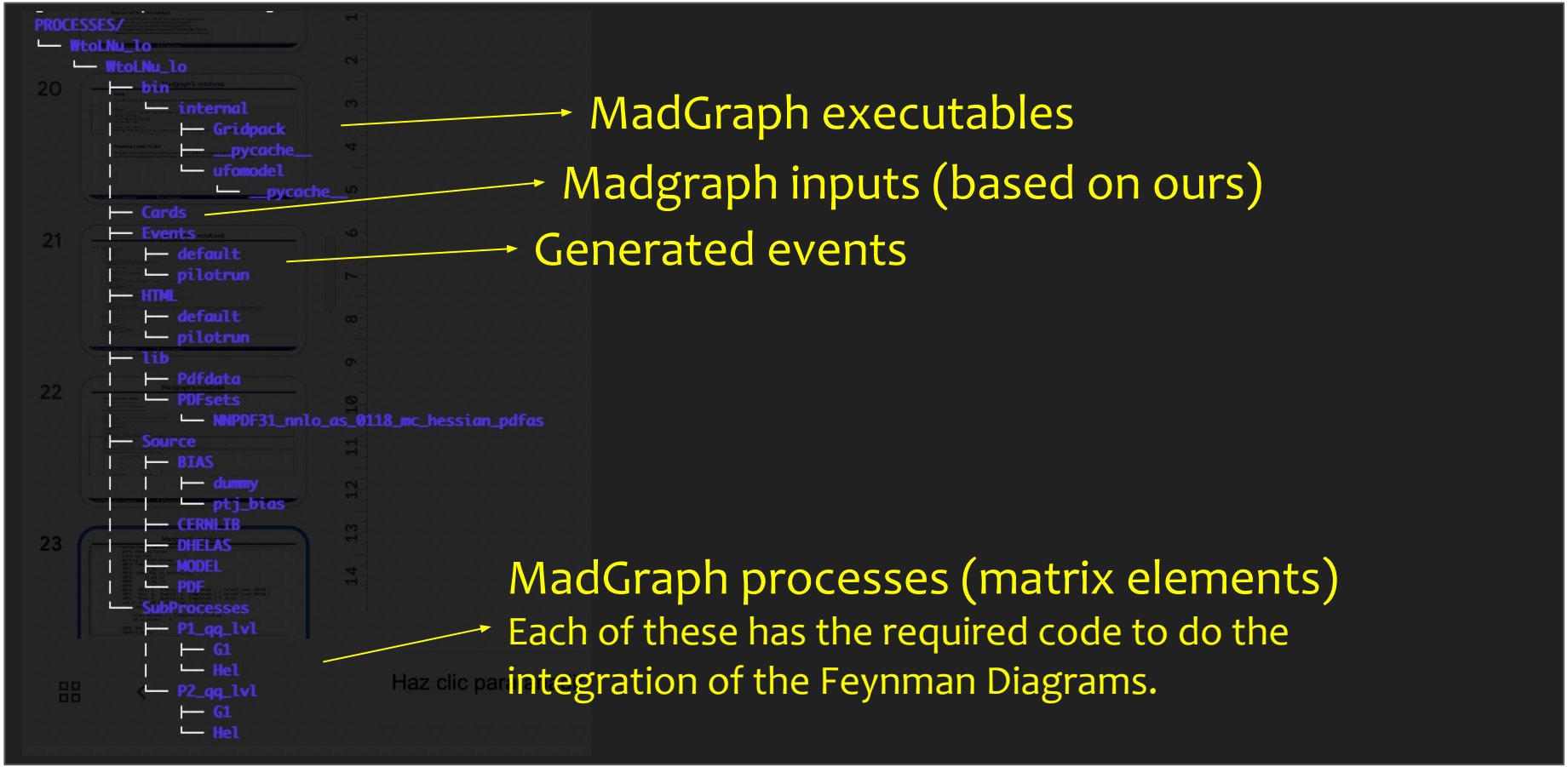
Madgraph's notebook

```
Using random number seed offset = 21
INFO: Running Survey
Creating Jobs
Working on SubProcesses
INFO: Compiling for process 1/2.
INFO:     P1_qq_lvl
INFO: Compiling for process 2/2.
INFO:     P2_qq_lvl
INFO:     P1_qq_lvl
INFO:     P2_qq_lvl
INFO:  Idle: 1,  Running: 0,  Completed: 1 [ current time: 09h28 ]
INFO:  Idle: 0,  Running: 1,  Completed: 1 [ current time: 09h28 ]
INFO:  Idle: 0,  Running: 0,  Completed: 2 [  0.72s  ]
sum of cpu time of last step: 1 seconds
  === Results Summary for run: pilotrun tag: tag_1 ===

     Cross-section :   1.814e+04 +- 57.76 pb
     Nb of events :  0

INFO: End survey
combine_events
```

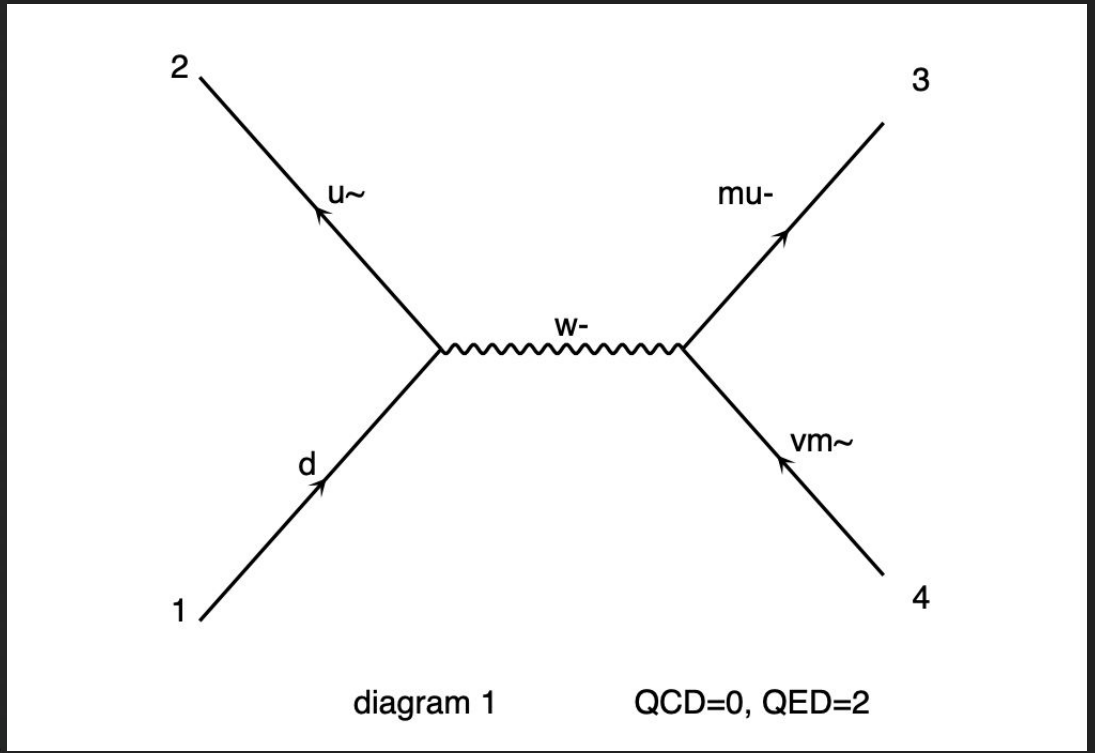
Madgraph's notebook



Madgraph's notebook

```
PROCESSES/
└─ WtoLNu_lo
   └─ WtoLNu_lo
      ├─ bin
      │  └─ internal
      │     ├─ Gridpack
      │     ├─ __pycache__
      │     └─ ufomodel
      │        └─ __pycache__
      ├─ Cards
      ├─ Events
      │  ├─ default
      │  └─ pilotrun
      ├─ HTML
      │  ├─ default
      │  └─ pilotrun
      ├─ lib
      │  ├─ Pdfdata
      │  └─ PDFsets
      │     └─ NNPDF31_nnlo_as_0118_mc_hessian_pdfas
      ├─ Source
      │  ├─ BIAS
      │  │  ├─ dummy
      │  │  └─ ptj_bias
      │  ├─ CERNLIB
      │  ├─ DHELAS
      │  ├─ MODEL
      │  └─ PDF
      └─ SubProcesses
         ├─ P1_qq_lvl
         │  ├─ G1
         │  └─ Hel
         └─ P2_qq_lvl
            ├─ G1
            └─ Hel
```

Haz clic para añadir



Madgraph's notebook

```
%%bash
source setup.sh
cd ${HERE}

source .venv/bin/activate
pip3 install six

function run_events() {
    mode=$1
    gridpack=$2
    nevents=$3
    seed=$4
    batchID=$5

    # Validate inputs
    if [[ ! -f "${gridpack}" ]]; then
        echo "ERROR: Gridpack not found at ${gridpack}"
        exit 1
    fi

    # Create a temporary directory for this run
    full_gridpack="${HERE}/PROCESSES/${my_gridpack}/${my_gridpack}_gridpack.tar.gz"
    run_dir="${HERE}/PROCESSES/${my_gridpack}/run_events/"
    output_folder="${HERE}/my_events/${my_gridpack}/"

    mkdir -p "${run_dir}" "${output_folder}"
    # Extract the gridpack into the run directory
    pushd "${run_dir}" >> /dev/null

    # make sure you remove previous runs beforehand
    find . -name "temp_run" -type d -exec rm -rf {} \;

    run_batch_job ${mode} ${gridpack} ${nevents} ${seed} ${batchID}

    mv batch${batchID}.lhe.gz "${output_folder}/"

    popd >> /dev/null
    echo "==> Done. Output: ${output_folder}/batch${batchID}.lhe.gz"
}

# ── Configuration ──────────────────────────────────────────────
mode="mg-lo"              # Generation mode: 'mg-nlo' (aMC@NLO) or 'mg-lo' (MadGraph LO)
my_gridpack="WtoLNu_lo"   # Name of the gridpack (must match a folder under PROCESSES/)
nevents=1000               # Number of events to generate
seed=3                    # Random seed for reproducibility
# ──────────────────────────────────────────────────────────────

# Let's span over 3 different seeds to generate 10 different LHE files
for iseed in $(seq 1 ${seed}); do
    run_events \
        ${mode} \
        "${HERE}/PROCESSES/${my_gridpack}/${my_gridpack}_gridpack.tar.gz" \
        ${nevents} \
        ${iseed} \
        ${iseed} # we use the iseed as batchID for simplicity
done
```

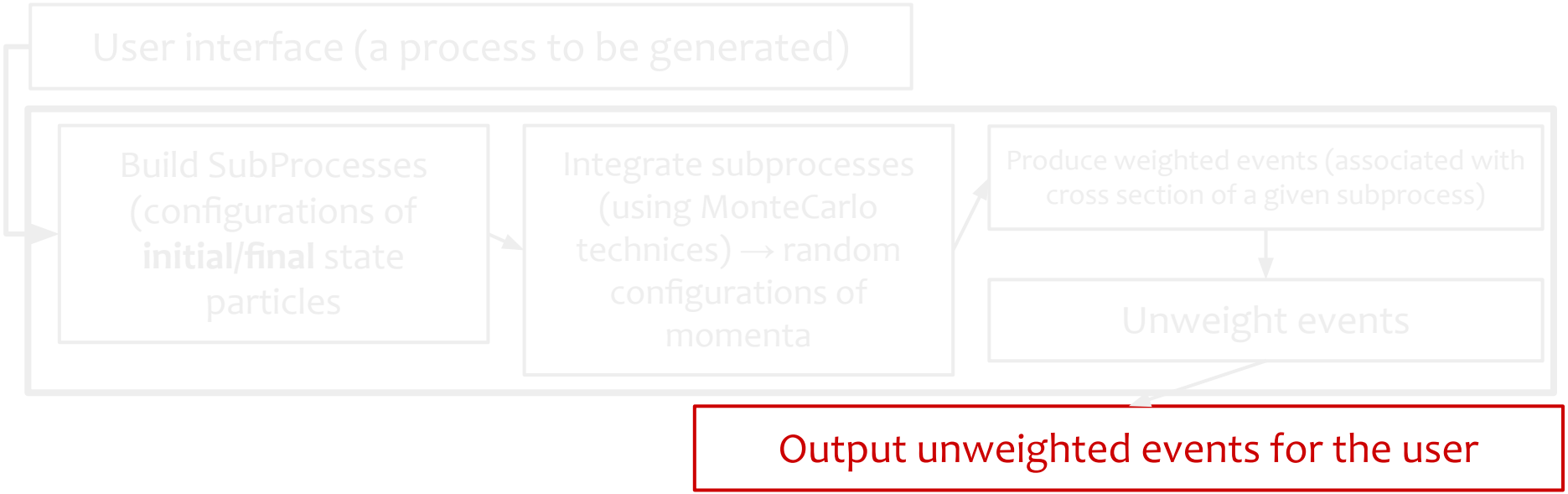
```
[cvicovil@lxplus917 COMCHA]$ tree my_events/
my_events/
└── WtoLNu_lo
    └── batch1.lhe.gz
```

```
                pz = lep.pz + nu.pz
                m2 = E*E - (px*px + py*py + pz*pz)
                m_lnu.append(np.sqrt(max(m2, 0.0)))

m_lnu = np.array(m_lnu)
```

Madgraph's procedure for MC generation

- **Note: feel free to open the `01_Madgraph_Generation.ipynb` notebook**
- **MadGraph automates the calculation of Feynman diagrams**
  - It computes tree-level matrix elements (MEs) for any specified process
  - Think of it as a sophisticated calculator that exhaustively enumerates every possible way particles can interact at the most fundamental level.



Madgraph's procedure for MC generation

- **Remember: we want to make use of our simulations to compare against recorded events in data**
  - Therefore the output of the generation must have an specific event-like format.
  - This is the so-called **Les Houches Event** (LHE) data format.

A standard format for Les Houches Event Files

J. Alwall, A. Ballestrero, P. Bartalini, S. Belov, E. Boos, A. Buckley, J.M. Butterworth, L. Dudko, S. Frixione, L. Garren, S. Gieseke, A. Gusev, I. Hinchliffe, J. Huston, B. Kersevan, F. Krauss, N. Lavesson, L. Lönnblad, E. Maina, F. Maltoni, M.L. Mangano, F. Moortgat, S. Mrenna, C.G. Papadopoulos, R. Pittau, P. Richardson, M.H. Seymour, A. Sherstnev, T. Sjöstrand, P. Skands, S.R. Slabospitsky, Z. Wcas, B.R. Webber, M. Worek, D. Zeppenfeld

A standard file format is proposed to store process and event information, primarily output from parton-level event generators for further use by general-purpose ones. The information content is identical with what was already defined by the Les Houches Accord five years ago, but then in terms of Fortran commonblocks. This information is embedded in a minimal XML-style structure, for clarity and to simplify parsing.

Madgraph's procedure for MC generation

```
<init>
2212 2212 6.500000e+03 6.500000e+03 0 0 325300 325300 -4 2
1.070361e+05 4.416387e+02 1.256751e+05 0
1.863897e+04 8.227356e+02 1.256751e+05 1
<generator name='MadGraph5_aMC@NLO' version='2.9.27'>please cite 1405.0301 </generator>
</init>
```

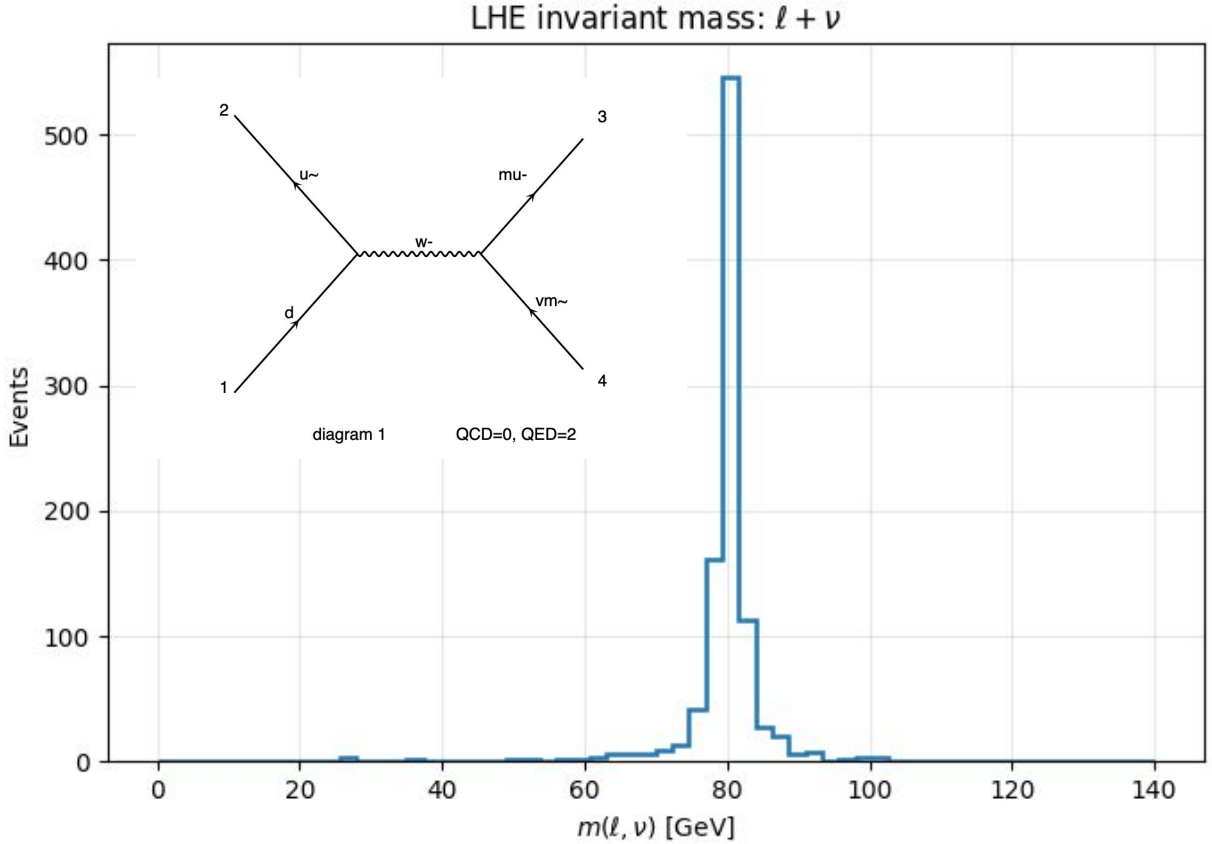
- **2212, 2212:** PDG IDs of the incoming beams
  - 2212 = proton
- **6.500000e+03, 6.500000e+03:** Beam energies (in GeV)
- **0, 0:** PDF group (0 means default)
- **325300, 325300:** PDF set used for parton modeling (see lhapdf)
- **-4:** Weighting strategy
  - -4 is what usually madgraph does (events can be negative or positive)
- **2:** Number of subprocesses included

Madgraph's procedure for MC generation

```
<event>
5       1 +1.8131200e+04 8.07407300e+01 7.54677100e-03 1.20213800e-01
        2 -1    0    0  501    0 +0.0000000000e+00 +0.0000000000e+00 +2.7914970914e+00 2.7914970914e+00 0.0000000000e+00 0.0000e+00 -1.0000e+00
       -1 -1    0    0    0  501 -0.0000000000e+00 -0.0000000000e+00 -5.8383232467e+02 5.8383232467e+02 0.0000000000e+00 0.0000e+00 1.0000e+00
       24  2    1    2    0    0 +0.0000000000e+00 +0.0000000000e+00 -5.8104082757e+02 5.8662382176e+02 8.0740726680e+01 0.0000e+00 9.0000e+00
      -13  1    3    3    0    0 +1.2444688738e+01 -1.1189582325e+01 -5.5744214222e+02 5.5769330187e+02 0.0000000000e+00 0.0000e+00 1.0000e+00
       14  1    3    3    0    0 -1.2444688738e+01 +1.1189582325e+01 -2.3598685358e+01 2.8930519889e+01 0.0000000000e+00 0.0000e+00 -1.0000e+00
```

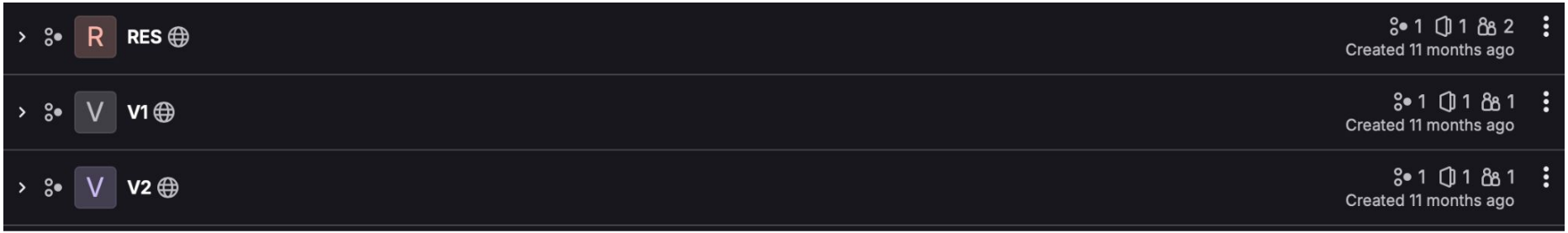
- **Each line corresponds to one generated particle**
- **Each column is:**
  - IDUP: PDG particle ID (e.g. 11 = e^-, 13 = μ^-, etc.)
  - ISTUP: Status code (- -1 = incoming, +1 = outgoing, +2 = intermediate resonance)
  - MOTHUP(1), MOTHUP(2): Indices of the first and last mother particles
  - ICOLUP(1), ICOLUP(2): Color flow indices (used to track QCD color connections)
  - PUP(1–5): 4-momentum and mass.
  - VTIMUP: Proper lifetime ($c\tau$)
  - SPINUP: spin information (cosine of angle between spin and momentum, or 9 if unknown)

Madgraph's procedure for MC generation

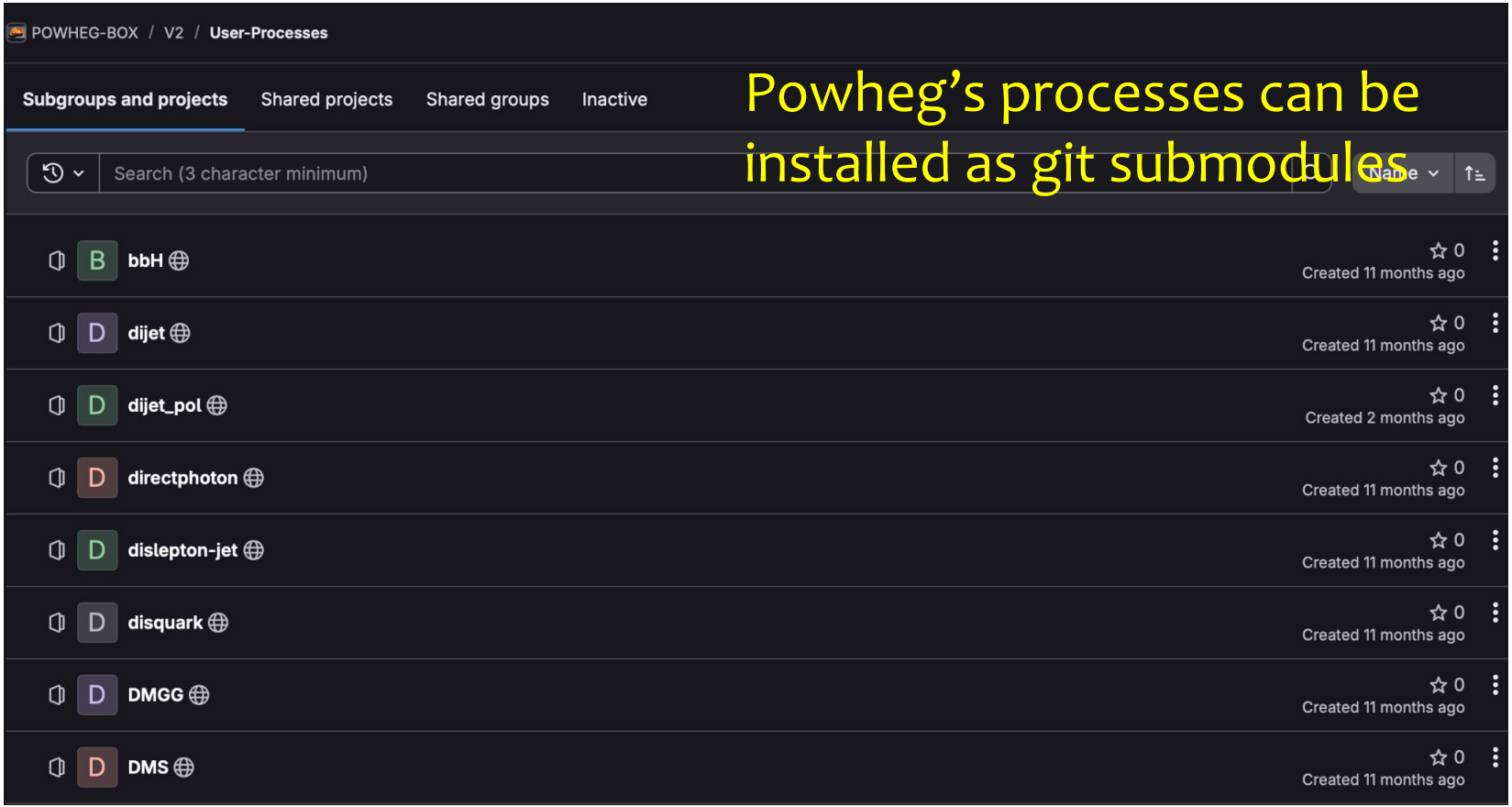


Powheg's procedure for MC generation

- **Note: feel free to open the `02_POWHEG_Generation.ipynb` notebook**
- **While MadGraph does automatic implementation of any physics process**
  - Powheg specializes in a wide variety of them.
  - Much more restricted procedure (less automated) – but arguably much more control on the implementation of the process.
- **Three flavours for POWHEG**
  - V1: first version of powheg
  - V2: currently most widely used version
  - RES: from **resonant aware** introduces improvements in the way resonances are treated when matching ME to Parton Shower (more on this later)

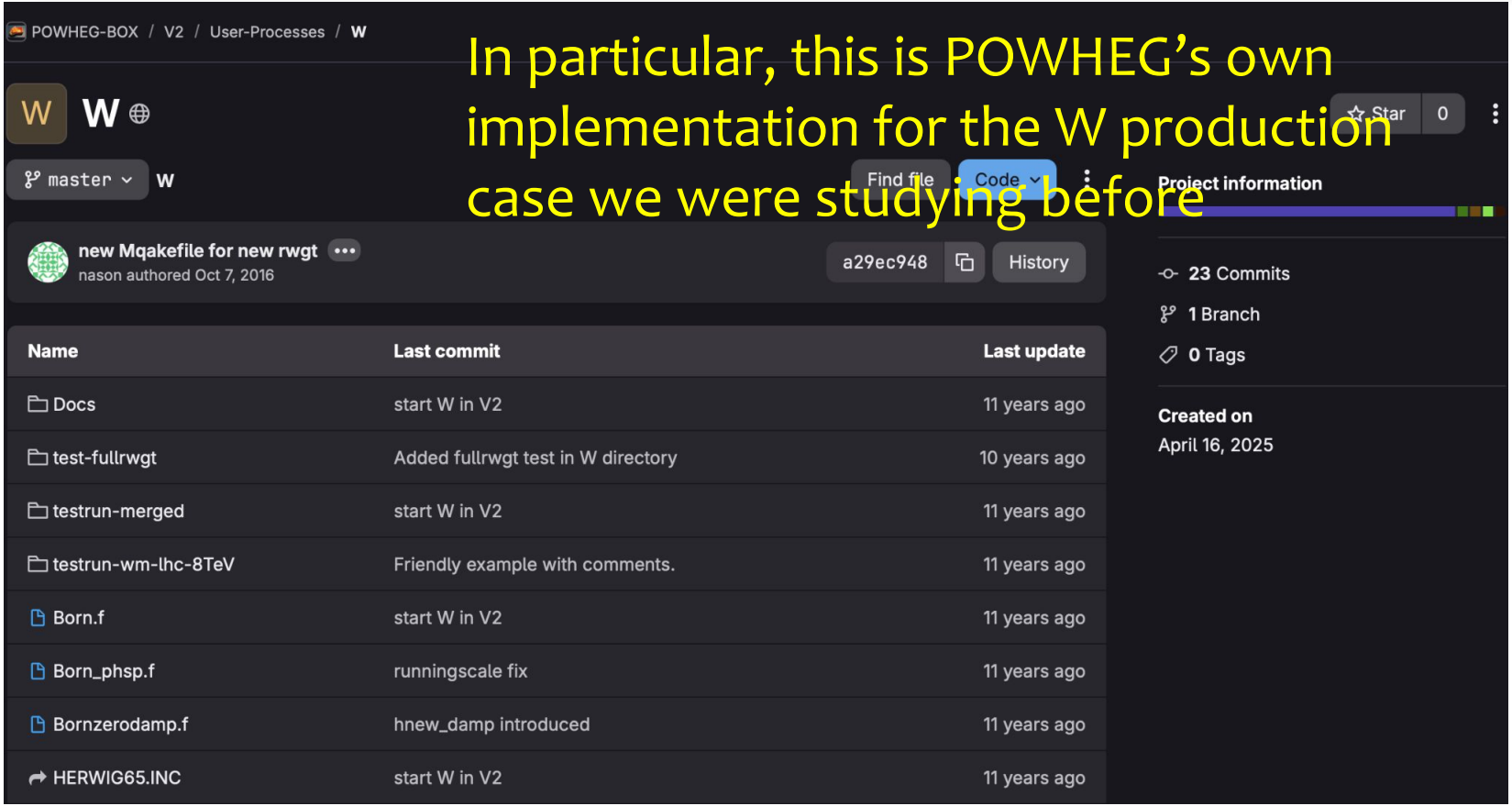


Powheg's procedure for MC generation



Powheg's processes can be installed as git submodules

Powheg's procedure for MC generation



In particular, this is POWHEG's own implementation for the W production case we were studying before

Main difference between Powheg and MadGraph

- **Here's a breakdown of the major differences:**
  - Madgraph:
    - Does automatic computation at any order (up to NLO)
    - Allows control of how real/virtual corrections are included in the predictions.
  - Powheg:
    - Only performs (in the majority of cases) at NLO (some even at Next-to-NLO)
    - It **always** includes one virtual emission at the very beginning of the computation → this is one of the most important features of POWHEG
    - **Why? —> Interfaced parton showers!**

How to model the soft interactions → parton showers

Parton shower algorithms

Welcome to PYTHIA

PYTHIA is a program for the generation of high-energy physics collision events, i.e. for the description of collisions at high energies between electrons, protons, photons and heavy nuclei. It contains theory and models for a number of physics aspects, including hard and soft interactions, parton distributions, initial- and final-state parton showers, multiparton interactions, fragmentation and decay. It is largely based on original research, but also borrows many formulae and other knowledge from the literature. As such it is categorized as a **general-purpose Monte Carlo event generator**.

Download and install PYTHIA 8.317

- **Parton shower algorithms are needed in order to complete the picture.**
  - Madgraph or powheg only handle the hard scatter
  - We need to correct for many other effects (hadronization, radiation, fragmentation, soft QCD…)
  - Parton shower (PS) softwares such as pythia or herwig do these kinds of operations up to a given order of accuracy.
  - **Unshowered events are not physical!**

Herwig 7

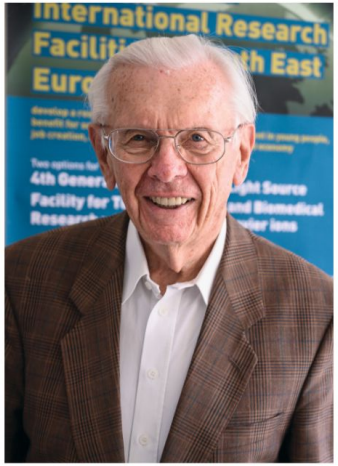
- Downloads | Tickets
- Tutorials | Plots
- Browse source | Doxygen
- Contact
- Talks and publications
- Developer area

ThePEG

- Browse source | Doxygen

Old Herwig

In Memory of Herwig Schopper



Key differences between PYTHIA8 and HERWIG7

- **Both pythia8 and herwig7 are one solution to the same issue**
  - How to properly model soft effects in MC generated events

Pythia 8 — Lund String Model

- The force between quarks is modeled like a **stretching string**.
- As the quarks move apart, the string gets longer and stores more energy.
  - When it stretches too much, it **breaks**, creating new quark–antiquark pairs that turn into hadrons.
  - The **string tension** determines how much energy is stored as it stretches.
- The process repeats, breaking the string step by step into many hadrons.

Herwig 7— Cluster Model

- All gluons are split into quark–antiquark pairs.
- Nearby color-connected quarks form **color-neutral clusters** that decay into pairs of hadrons based on phase space.
  - Commonly referred to as “angular shower”
  - Focuses on distance between coloured particles, rather than energy.
- Better “QCD coherence” of radiation

Parton shower algorithms

- **Although both PYTHIA8 and HERWIG7 can do the same task as MadGraph and Powheg**
  - Generating hard scatter events from matrix elements.
  - They do this at much lower accuracy in the QCD expansion.
  - Feel free to check `03_PYTHIA8.ipynb`.
- **So, what we often do at experiments is:**
  - We generate the matrix elements up to NLO with Madgraph/Powheg
  - We **interface** these predictions with a **parton shower software**
  - The overall result is complete up to NLO in QCD + PS (NLOPS accuracy)
  - There are **multiple** ways of doing this, and these are generator dependant
    - Interfacing PYTHIA8 to MadGraph is different than interface Herwig7 to MadGraph
    - Interfacing PYTHIA8 to MadGraph is different than interface PYTHIA8 to Powheg
- **So each software has its own ways of handling this interface.**

Parton shower algorithms

Setup	Accuracy	Extra jets handled by	Matching/Merging
LO	Leading Order	PYTHIA shower	none
LO MLM	LO multi-jet	MadGraph + PYTHIA	MLM matching
NLO	Next-to-Leading Order	PYTHIA shower	MC@NLO matching
NLO FxFx	NLO multi-jet	MadGraph + PYTHIA	FxFx merging

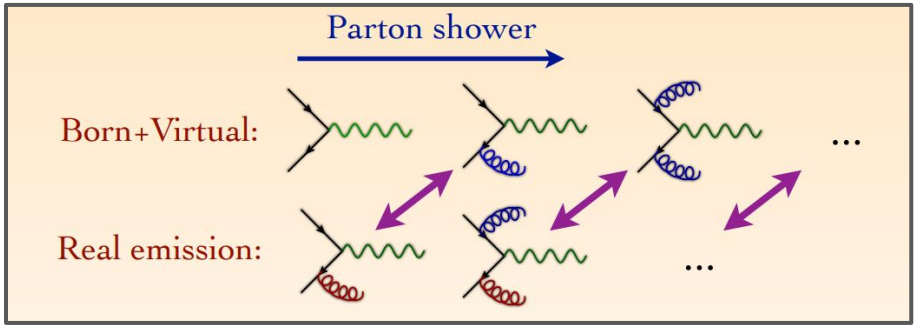
- **For instance, MadGraph has four different modes in which LHE events can be interfaced with PYTHIA8.**
- **POWHEG has its own matching too.**
  - Actually leading to one of the major distinction between POWHEG and Madgraph → positive and negative weights!

The aMC@NLO formula vs POWHEG

- The issue of negative event weights primarily arises at next-to-leading order (NLO).
- In **MadGraph**, this occurs because the overlap between **MEs** and the **PS** (e.g. PYTHIA) must be removed.
  - This subtraction is implemented by introducing compensating negative contributions that cancel the phase-space regions later filled by the parton shower.
  - As a result, some events acquire negative weights.
- In **POWHEG**, this problem is avoided by construction.
  - In this approach, the hardest (first) radiation is always generated by POWHEG itself, and this emission is always kept.
  - All subsequent radiation is then produced by the parton shower (e.g. PYTHIA).
  - Since this procedure avoids double counting without requiring event-by-event subtractions, all generated events have positive weights.

Parton shower algorithms: matching

- **We will not go in detail about each mode, but let's focus on the most interesting one: the FXFX.**
- The algorithm consists on matching jets (defined in PS) to partons (defined in ME).
- How events are rejected or not depends on how many jets match to partons, and the k_T hardness of the jets.



- The hardness is defined through a matching scale (μ_Q).
  - At p_T(radiation) > μ_Q → **our definition of** hard and non-collinear → pick from **Madgraph5_aMC@NLO**.
  - At p_T(radiation) < μ_Q → **our definition of** soft and collinear → pick from **pythia8**.
  - Jets are clustered as long as the the k_T hardness is above μ_Q.
- The gain is that **Madgraph5_aMC@NLO models radiation kinematics at NLO** while **pythia8** is **less accurate**.

End product

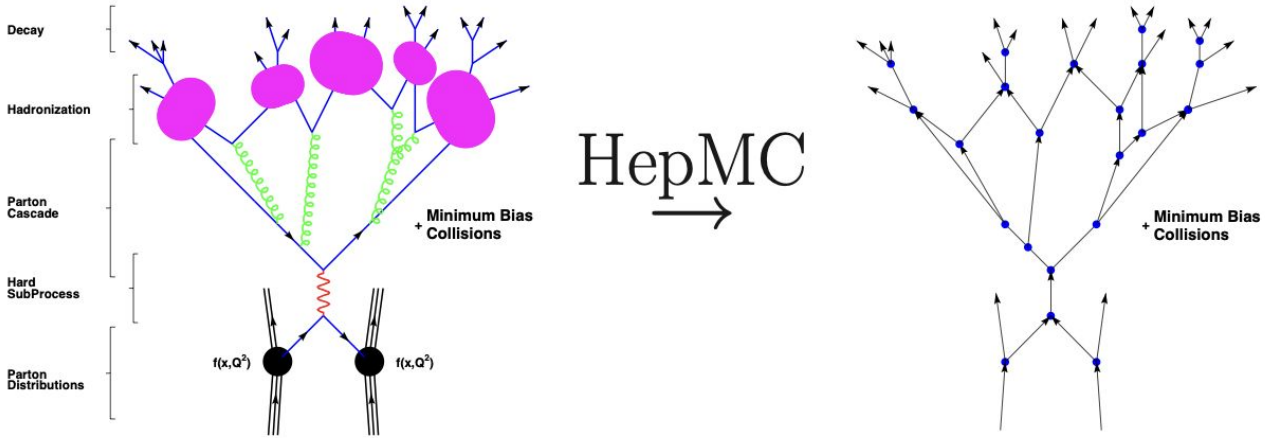
- **Once we have interfaced the matrix element predictions with PS matching, we obtain the final generated event that we need**
  - Format: typically `hepmc`.

The HepMC
C++ Monte Carlo Event Record
for High Energy Physics ⋆

Matt Dobbs, [a] Jørgen Beck Hansen [b]

[a] *University of Victoria, Department of Physics and Astronomy, P.O. Box 3055, Victoria, Canada V8W 3P6* [1]

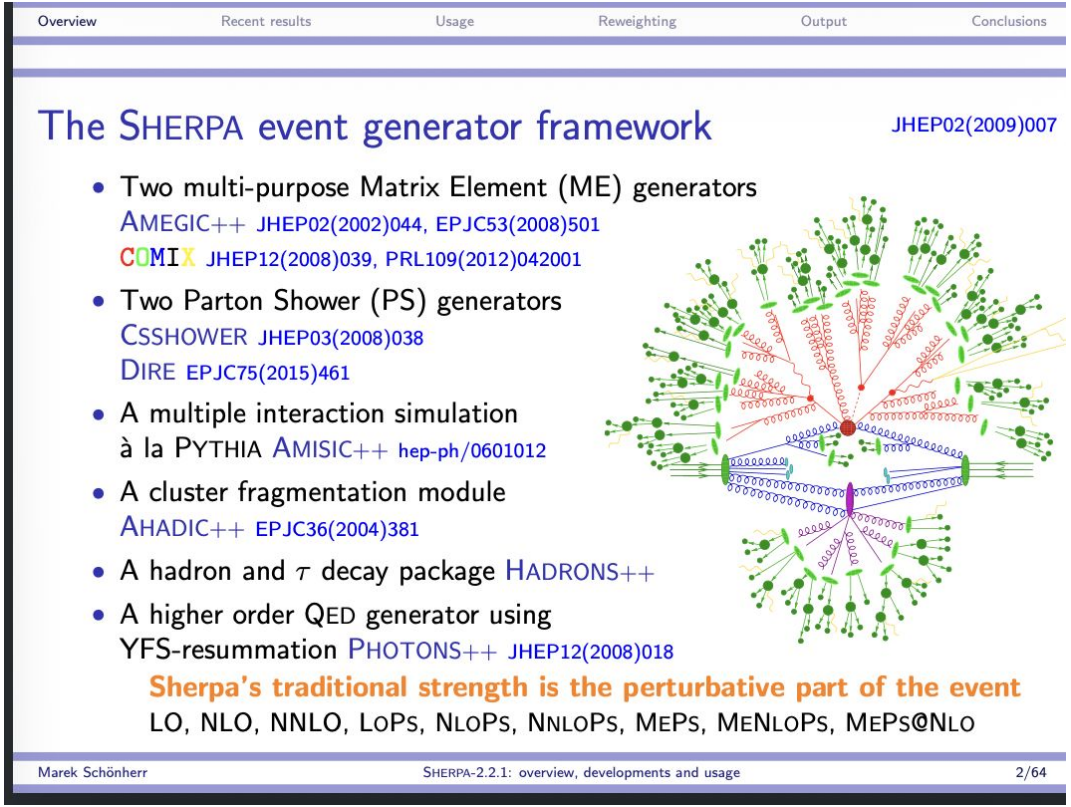
[b] *European Laboratory for Particle Physics (CERN), CH-1211 Geneva 23, Switzerland* [2]



One remark: sherpa

MC simulation with Sherpa

- **Sherpa is a software package (widely used in ATLAS) that is self contained:**
  - It allows for generating NLO accurate matrix elements
  - And interface them with its own Parton Shower algorithm
- **It has proven to be a very powerful generator in describing e.g. TOP properties.**
  - A minimal standalone setup is provided in `04_sherpa.ipynb`.
  - We will not focus on this generator for now.



MC tuning

MC tuning exercise

MC Tuning Exercise

In this exercise, you will consolidate the main concepts covered in the first part of the course through a guided, end-to-end hands-on project in Monte Carlo (MC) generator tuning.

Physics motivation

You will study the **Drell-Yan process** $pp \to Z/\gamma^* \to \mu^+\mu^-$ at the LHC. Differential distributions of the Z boson (e.g. transverse momentum p_T^Z, rapidity, ϕ^*) are highly sensitive to soft QCD effects — in particular to **multi-parton interactions (MPI)**, **initial-state radiation (ISR)** and **final-state radiation (FSR)**. The reference data come from CMS measurements CMS-FSQ-16-008 where detector effects have been fully unfolded to particle level, making them directly comparable to generator-level predictions.

What is MC tuning?

MC generators like PYTHIA8 contain many phenomenological parameters that cannot be calculated from first principles (e.g. the strong coupling α_s used in space- and time-like showers, or MPI core/energy-scaling parameters). **Tuning** is the process of finding the values of these parameters that best reproduce a set of reference measurements, typically quantified by a χ^2 metric.

MC tuning exercise

In this exercise, the six parameters being tuned are:

PYTHIA8 parameter	Physical meaning
`SpaceShower:alphaSvalue`	α_s for initial-state (space-like) radiation
`TimeShower:alphaSvalue`	α_s for final-state (time-like) radiation
`SigmaProcess:alphaSvalue`	α_s used in the hard-process cross section
`MultipartonInteractions:coreRadius`	Hadronic core radius in the MPI model
`MultipartonInteractions:coreFraction`	Fraction of matter in the hadronic core
`MultipartonInteractions:ecmPow`	Centre-of-mass energy scaling exponent for MPI activity

The kinematics of the Z boson are in sensitivy to the choice of these parameters, particularly in the low p_T region, where soft radiation corrections (integrated within PYTHIA8) become more relevant.

MC tuning exercise

Overview of the pipeline

The full simulation chain you will work through consists of four stages:

Stage	Tool	Output
Hard-process generation	**MadGraph5_aMC@NLO**	Gridpack → LHE events
Parton shower + hadronisation	**PYTHIA8**	HepMC2 events
Analysis & comparison to data	**RIVET**	χ^2/ndof per observable
Model optimisation	**Graph Neural Network** + GP	Optimal PYTHIA8 parameters

To jump straight to the GNN training, you can just these data files.

- Make sure to download them into a folder named “data”

MC tuning exercise: GNN

Based on the ParticleNet architecture used in CMS for b tagging

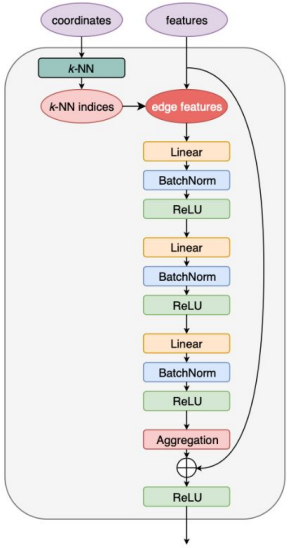
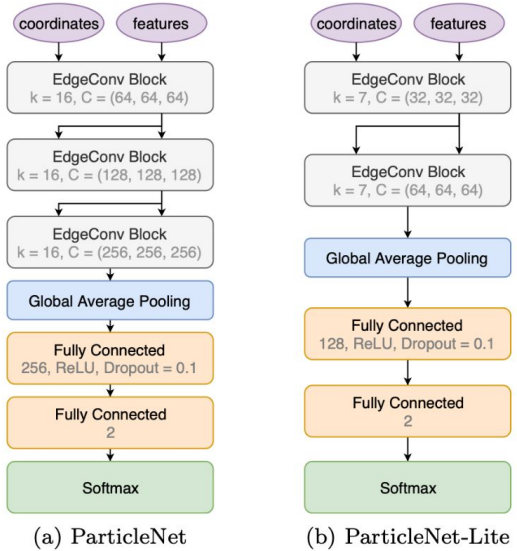


FIG. 1: The structure of the EdgeConv block.

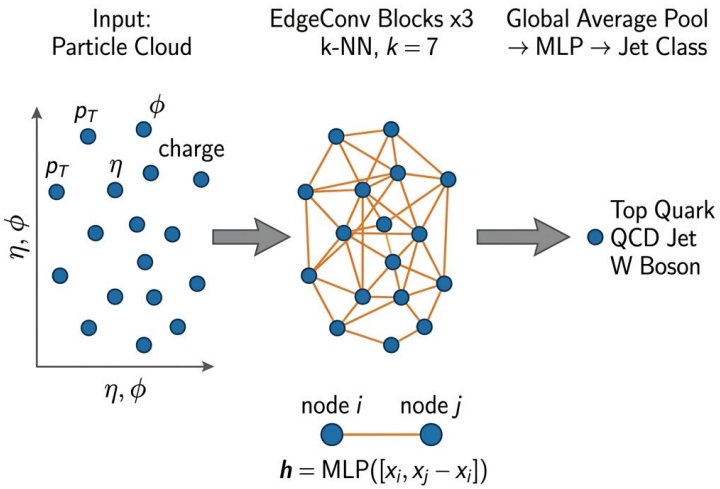


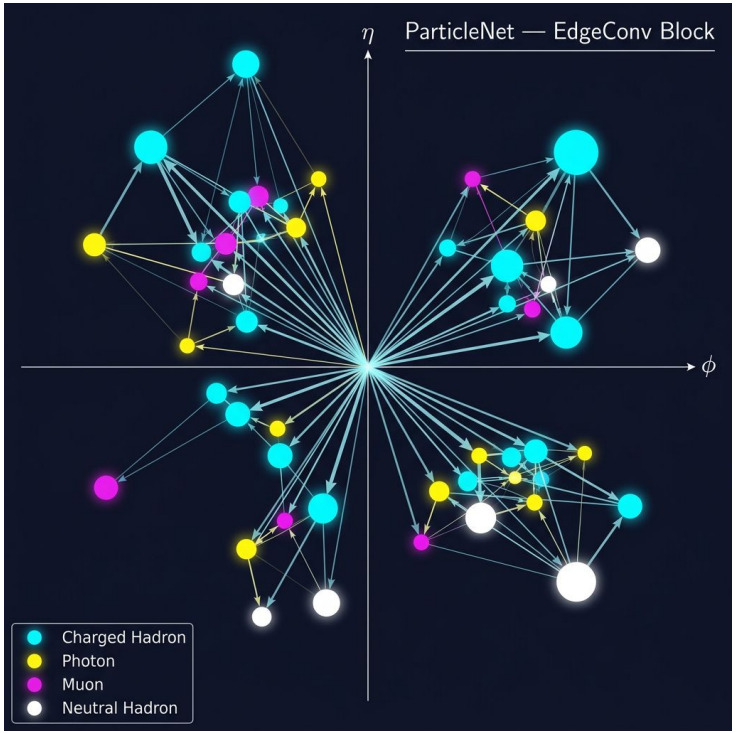
(a) ParticleNet (b) ParticleNet-Lite

FIG. 2: The architectures of the ParticleNet and the ParticleNet-Lite networks.

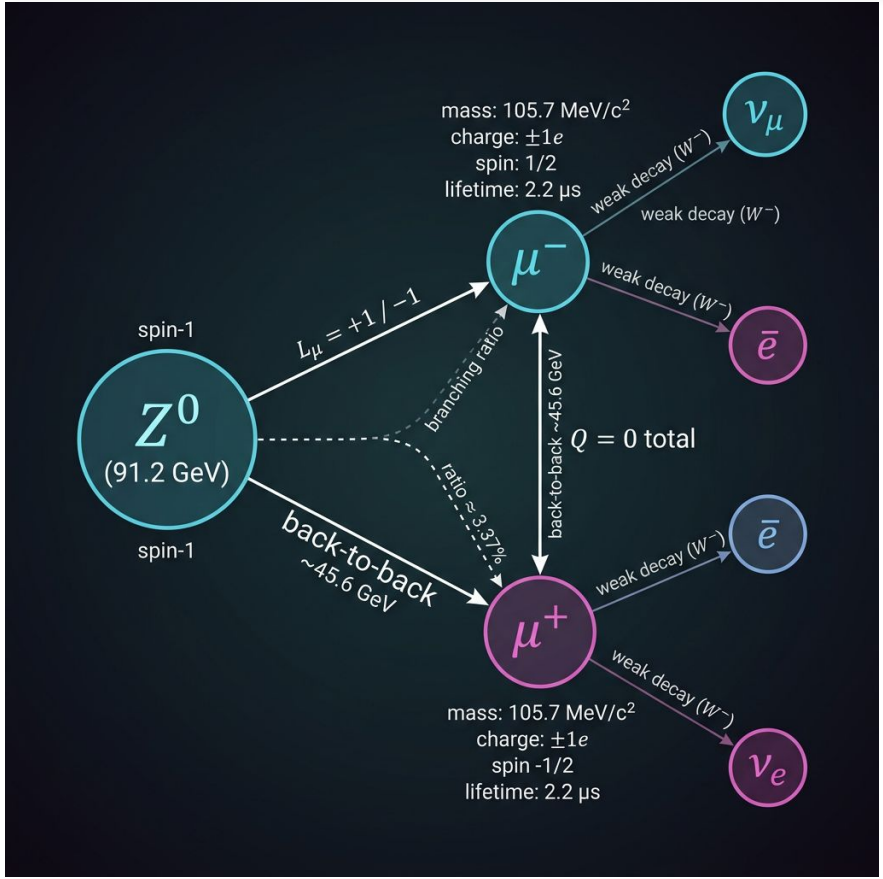
MC tuning exercise: GNN

Based on the ParticleNet architecture used in CMS for b tagging





MC tuning exercise: GNN



- **In our exercise:**
  - We are going to visualize events as “graphs”
  - For each “graph” muons are represented as nodes
    - Features: four momenta, charge, and pdgId.
  - Each node talks to other – closeby – nodes through and edge.
    - Features: distances between nodes, differences in momentum, etc…
  - The network takes as target how well a given PYTHIA8 set of parameters models the data.
  - Learns how to extrapolate the chi2 for an alternatative set of parameters.
  - We train in multiple phase spaces.
- **Then we fit the outputs to obtain the final set of parameters**

New techniques on MC generators

Developments in MC simulation

- **One has to take into account that the most important developments in MC generators must come from the theory side.**
  - Higher accuracies in perturbative expansions
  - Improved decay kinematics
  - Improved parton shower
- **However, there are many other parallel developments that heavily rely on MC simulation improvements**
  - Improved hardware architectures:
    - Acceleration in GPUs: MG@GPU
    - Acceleration in FPGAs: soon :)
  - Faster algorithms:
    - Grid integration → MadNis
    - Faster simulation → Flashim
    - **Testing the acceleration in FPGAs → Soon! (spanish work!)**
  - Improved techniques
    - Lowering the effect of negative weights in MC simulations

Developments in MC simulation

Cascade Pipeline for Leading-Order Matrix Element Evaluation on AMD Versal AI Engine Arrays

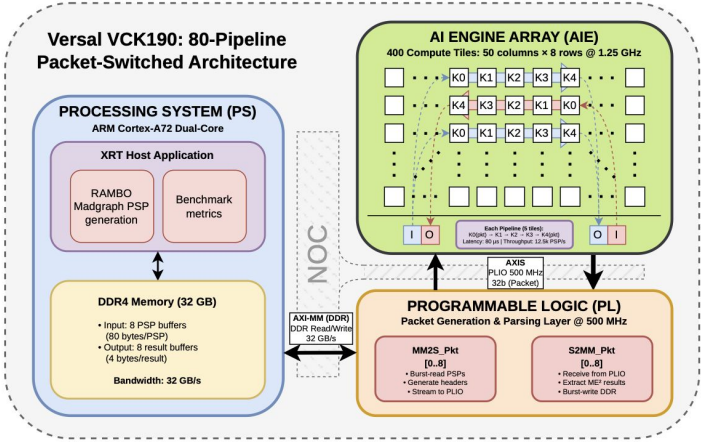
P. Leguina López[1*], J. Fernández Menéndez[1], L. Fiorini[2], S. Folgueras[1], F. Hervás Álvarez[2], H. Gutiérrez Arance[2], A. Valero[2], C. Vico Villalba[3]

[1*]Universidad de Oviedo – ICTEA, Oviedo, Spain.
[2]Instituto de Física Corpuscular (IFIC), Universitat de València – CSIC, Valencia, Spain.
[3]Rice University, Houston, Texas, USA.

*Corresponding author(s). E-mail(s): leguinapelayo@uniovi.es;

Abstract

A major computational bottleneck in modern High Energy Physics event generators arises from the integration of the matrix element, which requires repeated evaluations at different phase-space points to cover all possible initial- and final-state configurations. As the Large Hadron Collider enters its High-Luminosity phase, the demand for energy-efficient acceleration is expected to exceed the limits of conventional CPU scaling, motivating the use of highly parallel computing platforms such as graphics processing units (GPUs). In this work, we present an alternative approach based on a cascade pipeline architecture for evaluating leading-order matrix elements of the $\mathbf{gg \to t\bar{t}\,g}$ process on AMD Versal AI Engine (AI Engine) arrays. Due to the 16 KB per-tile program memory constraint, the computation is decomposed into a five-stage pipeline, with stages communicating via a wavefunction-token protocol over the on-chip cascade interface. Mapping 80 independent pipelines onto the 400 AI Engine tiles of the VCK190 platform yields a projected throughput of $\mathbf{1.0 \times 10^6}$ matrix element evaluations per second at 54.8 W, corresponding to a **34×** speedup over a single CPU core and a **7.7×** improvement in energy efficiency. Numerical agreement with the MadGraph5_aMC@NLO double-precision reference is validated at the parts-per-million level in mean relative error.



- **We study a way of accelerating the integration of a MadGraph process with an FPGA.**
  - The trade-off between process complexity seems to not fit well into the FPGA logic.
  - **However** this opens the floor for more advance attempts at trying to use hardware based accelerators on MC simulation.