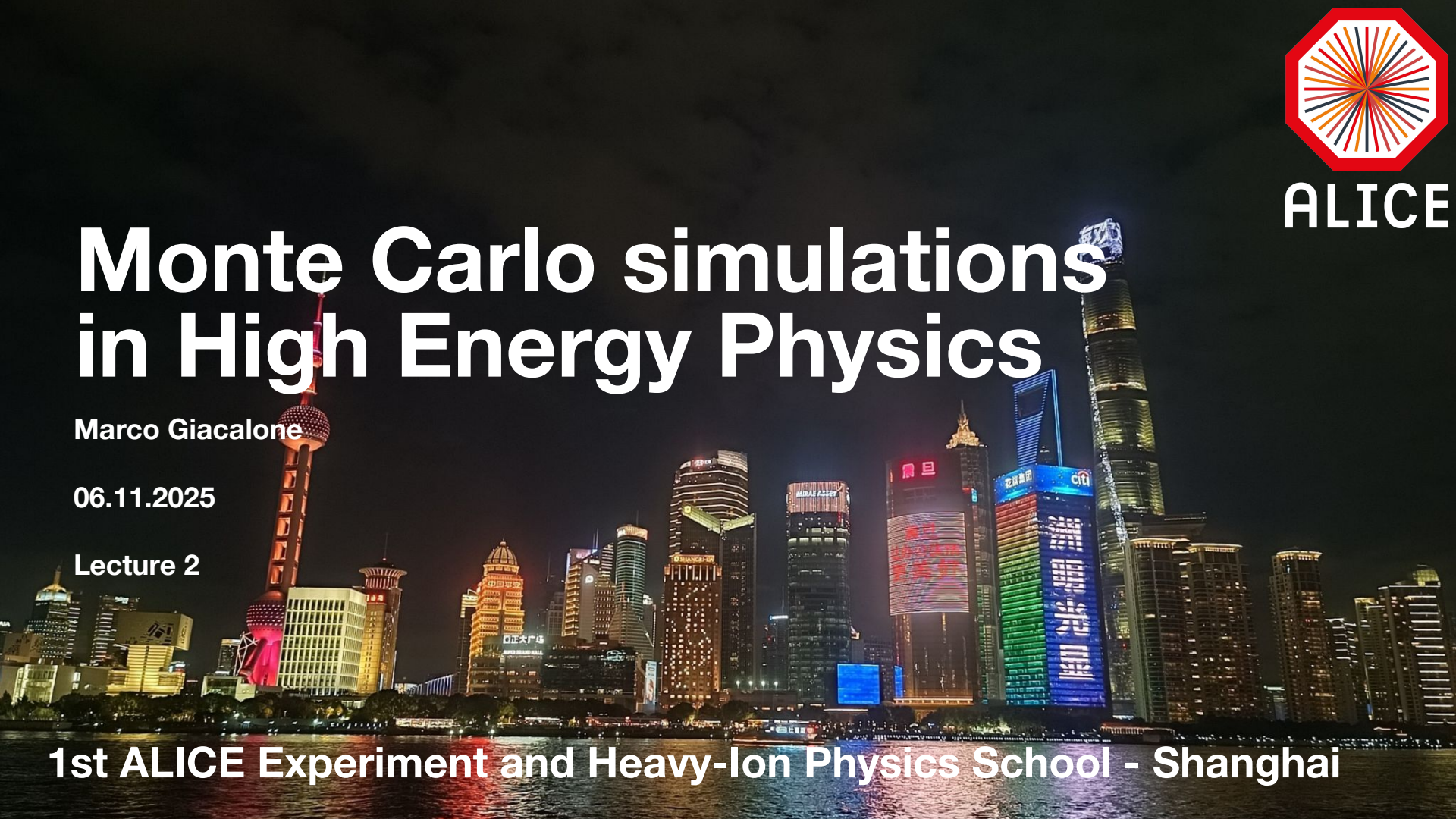


Monte Carlo simulations in High Energy Physics

Marco Giacalone

06.11.2025

Lecture 2

1st ALICE Experiment and Heavy-Ion Physics School - Shanghai

Software environment reminder

Important:

Users without an AliEn certificate will experience a limited usage of the next presented tools

simplest local build (basic generators such as PYTHIA8)

```
aliBuild build O2 O2DPG --defaults o2

alienv enter O2/latest,O2DPG/latest
```

full local build (all generators with AliGenO2, QC and O2Physics included)

```
aliBuild build O2sim --defaults o2

alienv enter O2sim/latest
```

MC stable releases → listed here

```
/cvmfs/alice.cern.ch/bin/alienv enter O2PDPSuite::MC-prod-2025-v13-1
```

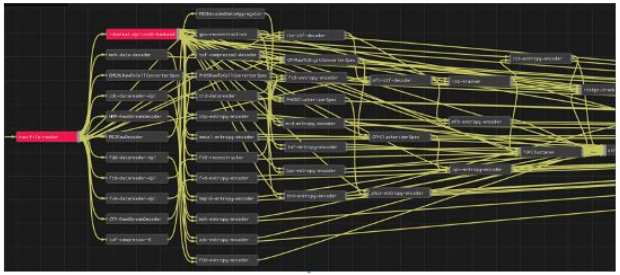
Integrated workflows: O2DPG MC

- In order to produce simulated AODs, we need to go beyond o2-sim and event generation and run the complete algorithmic pipeline including digitization and reconstruction steps

- This is a complex system, consisting of many executables or tasks, requiring consistent application and propagation of settings/configuration to work together
  - Example: full-system-test for data taking
  - hard to get right on your own ⟶ use a maintained setup !

- For ALICE Run3, the official production system targeting GRID productions is O2DPG repo (MC part)

- O2DPG also contains scripts/setup for data taking (DATA part)

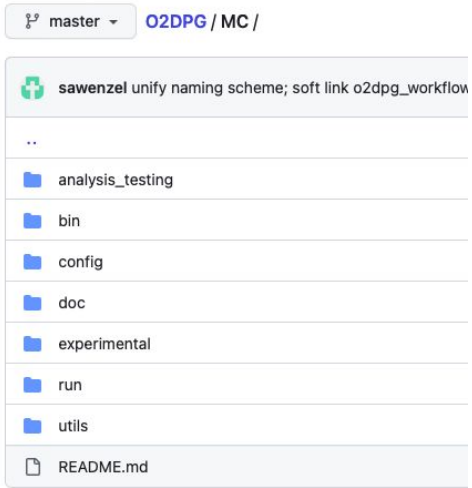


"Interplay of algorithms is a complex system (DPL topology)"

O2DPG ...

- provides **authoritative setup for official MC productions** for ALICE-Run3 and a runtime to execute MC jobs on GRID
- **integrates all relevant processing tasks** into a **coherent and consistent environment** to have a working pipeline from event generation to AOD and beyond
- **maintains PWG generator configurations** as versioned code
- performs testing / CI on PWG generator configurations

https://github.com/AliceO2Group/O2DPG



Important directories:

- MC/bin (workflow creation/execution)
- MC/run (PWG specific run scripts)
- MC/config (PWG specific generator configs)

Fundamentals of O2DPG-MC

- Running a MC job, is a two-fold process to decouple configuration logic from execution logic

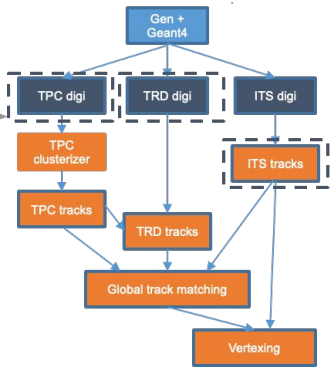
    1. Create a valid/configured description of a MC job == “workflow”

Workflow creator

Creates a coherent, integrated MC workflow in directed-acyclic-graph (DAG) form, described as a JSON, modelling the dependency of tasks

Configures the MC workflow as function of important user parameters:

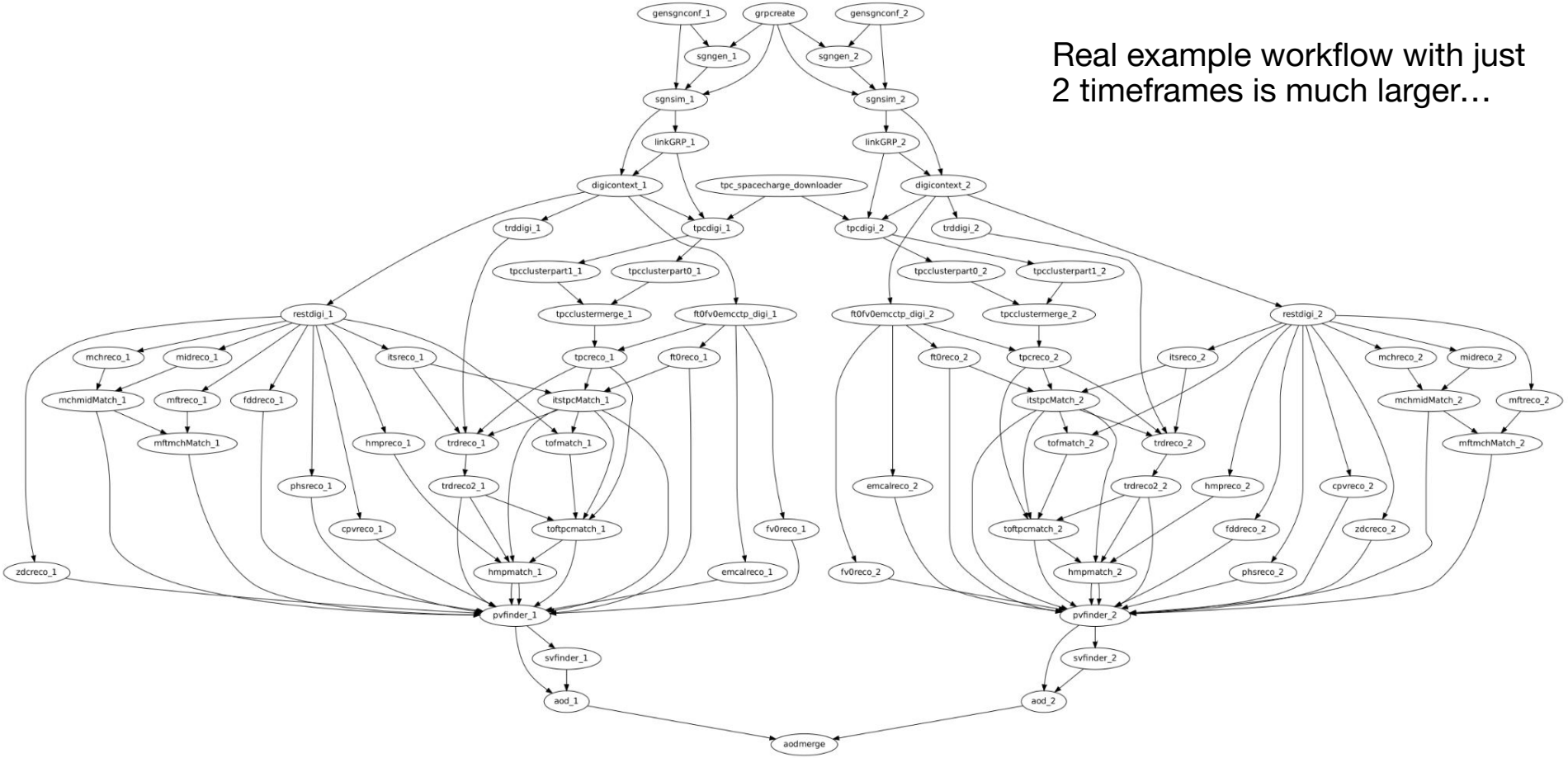
- collision system
- generators
- interaction rate
- number of timeframes



workflow.json

Simple example…

Fundamentals of O2DPG-MC

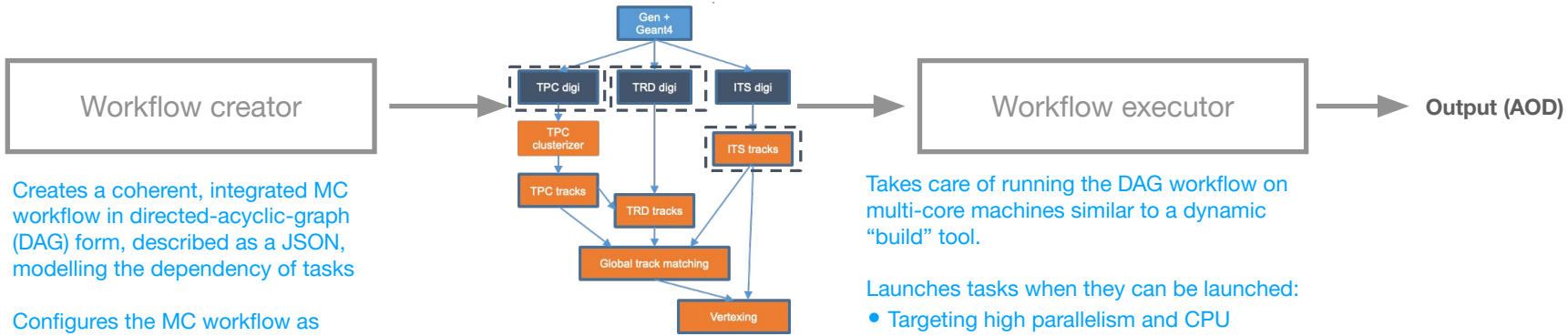


Real example workflow with just 2 timeframes is much larger…

Fundamentals of O2DPG-MC

- Running a MC job, is a two-fold process to decouple configuration logic from execution logic

  1. Create a valid/configured description of a MC job == “workflow”
  2. Run the MC job with a dynamic graph scheduler



Creates a coherent, integrated MC workflow in directed-acyclic-graph (DAG) form, described as a JSON, modelling the dependency of tasks

Configures the MC workflow as function of important user parameters:

- collision system
- generators
- interaction rate
- number of timeframes

workflow.json

Takes care of running the DAG workflow on multi-core machines similar to a dynamic “build” tool.

Launches tasks when they can be launched:

- Targeting high parallelism and CPU utilisation
- Respect resource constraints (tries not to overload the system)

In principle a universal tool, not specific to MC

O2DPG-MC workflows: Requirements

- Valid AliEn-tokens are required to run (to access CCDB objects)

The Condition and Calibration Data Base (**CCDB**) is a ROOT file used to store calibration and alignment data (centrality, TPC splines, calibration maps for space-charge distortions, …)

O2DPG-MC workflows: Requirements

- Valid AliEn-tokens are required to run (to access CCDB objects)
  - Experts may circumvent by using CCDB snapshots
- The O2DPG MC workflows are supposed to run in an 8-CPU core with 16GB RAM environment reflecting the default resources on the GRID
- This is also the requirement that you should fulfill when running locally on your laptop
- This translates into some defaults which are put in the workflow creation / execution
  - Transport simulation will use 8 workers
  - TPC + TRD digitisation 8 threads
  - The workflow runner will assume to have 8-cores available
- In turn, O2DPG MC workloads may lead to problems when run on hardware with less resources
  - But with a bit a tuning/adjustment it might be possible to run

👉 Some examples here; 👉 docs here

O2DPG-MC step 1: workflow creation

- ALICE Run3 MC workflow creation done by script **O2DPG/MC/bin/o2dpg_sim_workflow.py**
- Configures the MC workflow as function of important (user) parameters (collision system, generators, interaction rate, number of timeframes, transport engine, etc.)
  - `**o2dpg_sim_workflow.py --help**` →documentation with all available options

```
${O2DPG_ROOT}/MC/bin/o2dpg_sim_workflow.py -eCM 14000 -col pp
                                           -gen pythia8 -proc cdiff
                                           -tf 5 —ns 2000
                                           -interactionRate 500000
                                           -run 302000
```

"Generate an ALICE-Run3 Monte Carlo workflow for a 5 timeframe simulation, with 2000 events per timeframe, at interaction rate of 500kHz …. for 14TeV pp collisions using Pythia8 that has special process cdiff enabled…"

☝ **Important options:**

-gen, -tf, -n, -eCM, -interactionRate, -run, -col
Optionally: -field, -seed, -proc

Workflow creation: Run numbers

- The use of a run number is **mandatory** as it will be used to determine a timestamp needed to fetch conditions from CCDB
- So run numbers should be used even for non-data-taking anchored simulations
- A list of pre-defined run numbers for MC has been documented here: https://twiki.cern.ch/twiki/bin/view/ALICE/O2DPGMCSamplingSchema
- For example, for a PbPb simulation with field -0.5T, a run number of 310000 can be used
  - Should in principle fetch CCDB objects good for PbPb

Run numbers assignment for unanchored MC

Type	Collision System	Energy	Magnetic field	Run no. Range	Time range (in EPOCH seconds)	JIRA Ticket
Local testing				300000-300099	1546300800-1546343770	
Run 5 ALICE3	Pb-Pb pp	5.5 TeV 14 TeV	0.5T 2T	300100-300999	1546343800-1546730770	O2-2572 (LHC21d9[x])
Run 3	pp	900 GeV	0.2T	301000-301499	1546730800-1546945770	LHC21i1_nightly LHC21i1[a-c] LHC21i3[b, d-g]
			-0.2T	301500-301599	1546945800-1546988770	
			unassigned	*301600-301999*	*1546988800-1547160770*	
Run 3	pp	13.6 TeV	-0.5T	302000-302999	1547160800-1547590770	O2-2679 (LHC21k6) LHC21i3[a, c]
			0.5T	303000-303999	1547590800-1548020770	
			unassigned	*304000-309999*	*1548020800-1550600770*	
Run 3	Pb-Pb	5.02 TeV	-0.5T	310000-310999	1550600800-1551030770	O2-2773 (LHC22b2) O2-2779 (LHC22b6)
			0.5T	311000-311999	1551030800-1551460770	
			unassigned	*312000-319999*	*1551460800-1554900770*	

Recommended way

- Documentation
- Official configuration folder

Workflow creation: Generator configuration

- Custom configurations can be specified to the generation workflow thanks to .ini files

```
o2dpg_sim_workflow.py -gen pythia8 -ini <path/to/config.ini>
```

- They contain different sections for generator configurations but additional triggers for the produced particles can be added
- Official configurations can be found by default in

```
O2DPG/MC/config/<PWG>/ini/<config>.ini
```

  and they are tested by a CI when modifications are requested via PR or new configurations are added
- Configurations folder is linked to the O2DPG_MC_CONFIG_ROOT environment variable

  Local configurations can be used, but also newer configurations can be tested with older O2DPG build and viceversa

Snippet from PWGDQ configuration

```
[GeneratorPythia8]
config = ${O2DPG_MC_CONFIG_ROOT}/MC/config/common/pythia8/generator/pythia8_hf.cfg
hooksFileName = ${O2DPG_MC_CONFIG_ROOT}/MC/config/PWGHF/pythia8/hooks/pythia8_userhooks_qqbar.C
hooksFuncName = pythia8_userhooks_ccbar(-4.3,-2.3)
```

O2DPG-MC step 2: workflow execution

- Workflow runner/executor evaluates/builds a DAG workflow on a compute node
- Minimally, it takes the workflow file and a target as input

```
${O2DPG_ROOT}/MC/bin/o2dpg_workflow_runner.py -f workflow.json -tt aod
```

"Execute workflow up to aod task (assuming 8-core CPU config)"

Good to know

- Checkpointing and incremental build

```
o2dpg_workflow_runner.py -f workflow.json -tt digi
o2dpg_workflow_runner.py -f workflow.json -tt aod
```

"First execute until digitization … and then continue until AOD (not doing tasks again which are already finished!)

- Convert DAG to simple shell script which could be run standalone

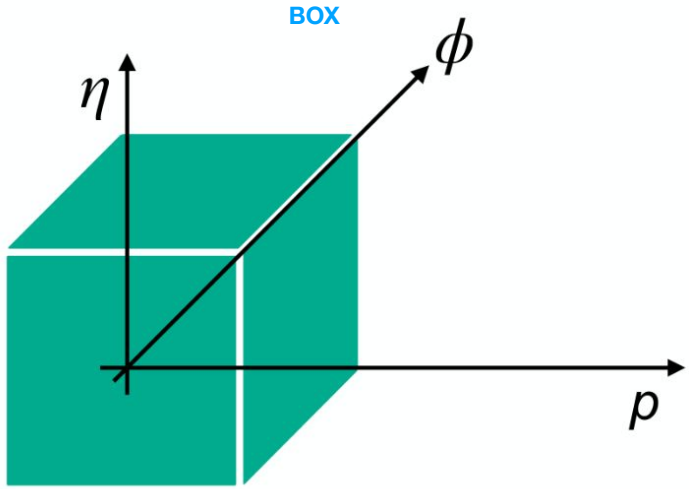
```
o2dpg_workflow_runner.py -f workflow.json -tt aod
--produce-script my_script.sh
```

"Create a simple shell script which can run everything sequentially up to AOD stage"

- … many more useful features
  - As usual, o2dpg_workflow_runner.py --help, lists possible options

BoxGenerator

- Simple mono-PDG particle generator implemented in O2
- Produces particles with uniform distributions in (p, η, ϕ) → 3D box shape



Very basic example

```
o2-sim --noGeant -j 8 -o boxtest -n 100 --seed 196485619 -g boxgen
--configKeyValues "BoxGun.number=1"
```

100 events with 1 pion each will be generated using default limits for (p, η, ϕ) →

```
int pdg = 211;
int number = 10;
double eta[2] = {-1, 1};
double prange[2] = {0.1, 5};
double phirange[2] = {0., 360.};
```

Configuration parameters

PYTHIA 8 in O2

- PYTHIA 8 is the most integrated and recommended generator in our framework
- It can be fully configured via a special text file and the GeneratorPythia8 parameter
  - valid settings can be found in the PYTHIA 8 reference manual
- The mkpy8cfg.py tool in O2DPG can be used to simplify the creation of the config file → all options available using the *--help* flag

pythia8.cfg

```
### random
Random:setSeed = on
Random:seed = 130145275

### beams
Beams:idA = 1000822080
Beams:idB = 1000822080
Beams:eCM = 5020.000000

### processes

### heavy-ion settings (valid for Pb-Pb 5520 only)
HeavyIon:SigFitNGen = 0
HeavyIon:SigFitDefPar = 13.88,1.84,0.22,0.0,0.0,0.0,0.0,0.0
HeavyIon:bWidth = 14.48

### decays
ParticleDecays:limitTau0 = on
ParticleDecays:tau0Max = 10.

### phase space cuts
PhaseSpace:pTHatMin = 0.000000
PhaseSpace:pTHatMax = -1.000000
```

run with this config

o2-sim -n 10 **-g pythia8 --configKeyValues "GeneratorPythia8.config=pythia8.cfg"**

mkpy8cfg.py usage example

```
${O2DPG_ROOT}/MC/config/common/pythia8/utils/mkpy8cfg.py --output=pythia8.cfg --idA=2212 --idB=2212 --eCM=13600 --process=none
```

👉 Example: run/SimExamples/HepMC

👉 Example: run/SimExamples/HepMC_STARlight

HepMC generation

- Many generators allow by default to output HepMC formatted data → universal and convenient way of storing information from MC event generators

- o2-sim is capable of reading HepMC files out-of-the box → data from FIFOs can be read as well
  - HepMC3 is the default, but HepMC2.06 data are compatible as well
- An additional feature of the tool is to spawn event generators using the cmd parameter of GeneratorHepMC → generators printing data in the stdout can be set to feed data automatically to o2-sim → no need for large local .hepmc files

Generation with local HepMC file or FIFO

```
o2-sim -n 10 -g hepmc --configKeyValues
“HepMC.fileName=/path_to/file.hepmc"
```

```
… --configKeyValues 'HepMC.fileName=/path_to/file.hepmc;HepMC.version=2"
```

Generation with automatic FIFOs using run/SimExamples/HepMC_EPOS4

```
o2-sim -n 100 -g hepmc --seed 12345 --configKeyValues
"GeneratorFileOrCmd.cmd=epos.sh;GeneratorFileOrCmd.bMax
Switch=none;HepMC.version=2"
```

More info <HERE>

👉 Example: Run/SimExamples/AliRoot_AMPT

👉 Example: Run/SimExamples/AliRoot_Hijing

External Generators

- Apart from Pythia, direct (compiled) integration of specific generators is small in O2 in order to decouple PWG specific generator code and configs from data-taking
  - avoid recompile
- Rather, “external” generators can be interfaced in o2-sim by using just-in-time ROOT macros which implement, e.g., a **GeneratorTGenerator** class
  - setup generator at “use-time” in C++
  - generator setup becomes “configuration problem”
- This method is used to setup PWG specific generation in the O2DPG production system
  - e.g. PWGDQ cocktail generator

“call o2-sim with -g external option and reference the external file and function name”

```
o2-sim -n 10 -g external --configKeyValues
'GeneratorExternal.fileName=myGen.C;GeneratorExternal.funcName="gen(5020)"'
```

“stub content of ROOT macro file myGen.C”

```
// my fully custom generator
class MyGen : o2::generator::GeneratorTGenerator {
  void Init() override;
  bool generateEvent() override;
};

FairGenerator* gen(double energy) {
  return new MyGen(energy);
}
```

👉 More info in documentation

O2DPG generators configurations

- Custom O2DPG generators and triggers user configurations listed in the documentation

Custom configuration	Description	Main Folder	Generators Used
basic.ini	Basic generator configuration.	common	Pythia8
GeneratorLoopersFlatGas.ini	TPC loopers with flat gas distribution using WGAN neural network models for pair and Compton electron generation.	common	Custom (`TPCLoopers.C`)
GeneratorTPCloopers.ini	TPC loopers with Poisson-distributed pairs and Gaussian-distributed Compton electrons using WGAN neural network models.	common	Custom (`TPCLoopers.C`)
GeneratorTPCloopers_fixNPairs.ini	TPC loopers generator with a fixed number of pairs and Compton electrons using neural network models.	common	Custom (`TPCLoopers.C`)
pythia8_NeNe_536.ini	Pythia8 for NeNe collisions at 5.36 TeV.	common	Pythia8
pythia8_OO_536.ini	Pythia8 for OO collisions at 5.36 TeV.	common	Pythia8
pythia8_pO_961.ini	Pythia8 for pO collisions at 9.61 TeV.	common	Pythia8
GeneratorHF_bbbar_PsiAndJpsi_fwdy_triggerGap.ini	$b\bar{b}$, Psi and J/psi at forward rapidity with trigger gap.	PWGDQ	Pythia8
GeneratorHF_bbbar_PsiAndJpsi_midy_triggerGap.ini	$b\bar{b}$, Psi and J/psi at mid-rapidity with trigger gap.	PWGDQ	Pythia8

…

- Sorted in folders based on PWG interests: 275 configurations
  - Common: 7
  - PWGDQ: 32
  - PWGHF: 63
  - PWGEM: 55
  - PWGGAJE: 33
  - PWGLF: 47
  - ALICE3: 29
  - PWGUD: 2
  - examples: 7

EPOS4 and EPOS4HQ

- Both generators are available in their latest version
- Users can run standalone simulations by loading the AliGenO2 package
  → no need to use the ALICE simulation framework

However…

- External custom configuration in O2DPG (loading O2sim) allows to run simulation with an easier setup using our framework → example: GeneratorEPOS4.ini
- ROOT data format is available as output in standalone generator, but TTree is much less readable than output from o2-sim (4.0.0 documentation here)
- Not a lot of usage in our collaboration but:
  - O2DPG development makes EPOS4 easier to use
  - Soft-QCD observables, collective effects, and heavy-ion physics should be described well thanks to hydrodynamic system evolution, core-corona mechanism and a independent treatment of MPIs
- Available collision systems are hardcoded ⇒ will change in next release

Herwig 7

- It's the last generator included in AliGenO2 → 28/10/2025
- Standalone simulations are possible
  → Herwig ecosystem tutorial available in the website (to be adapted for ALICE environment)

Run with ALICE

- The generator can be interfaced directly to O2 using HepMC files/fifos → example provided in this folder
- The recommended way is to use the external O2DPG generator ⇒ generator_Herwig.C
  → no HepMC middle files and simplified configuration → ThePEG interface used
- Configuration options are provided by .in files → all available parameters in documentation and examples provided in Herwig7 source code
- Advanced users can feed the external gen with a .run file (locally built) and custom seed
  → reproducibility and testing purposes

ADVANCED

👉 Example: Run/SimExamples/Selective_Transport_pi0

Generators: Triggering

- Event filtering or triggering is also flexibly supported on the generator level
  - e.g., only produce and simulate events of a certain property
- A user-configurable “external” trigger follows the “external” generator mechanism
  - one implements a trigger function in a separate ROOT macro and pass it to o2-sim with the `-t external` option
  - the trigger function inspects the vector of all generator particles
- Advanced: DeepTriggers allow to trigger on the collection of the primaries and further internal information of the underlying generator

“call o2-sim with pythia8pp generator put pass forward only events that satisfy the trigger condition given in file Trigger.C”

```
o2-sim -n 10 -g pythia8pp -t external --configKeyValues
'TriggerExternal.fileName=myTrigger.C;TriggerExternal.funcName="trigger"'
```

“stub content of ROOT macro file myTrigger.C”

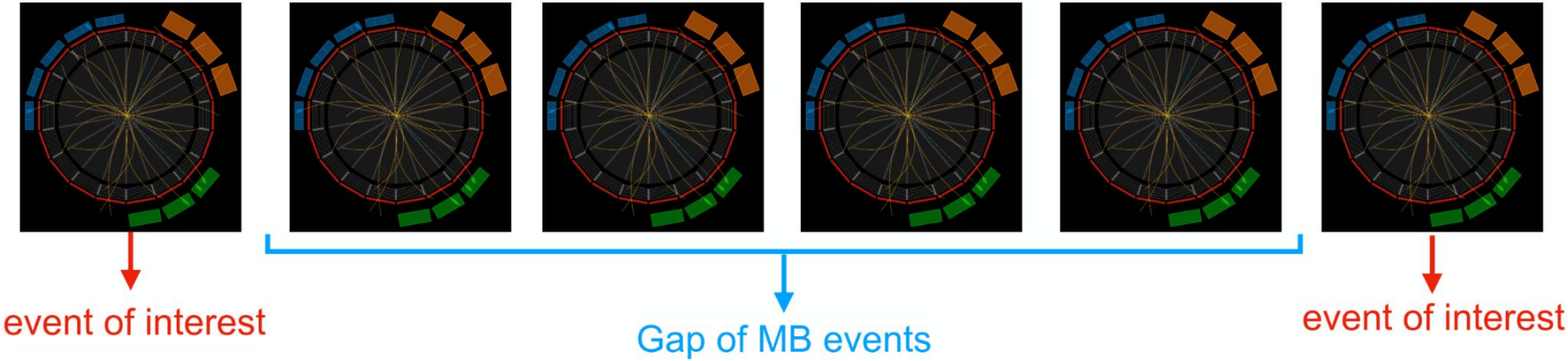
```
// returns fully custom event trigger function
o2::eventgen::Trigger trigger()
{
  return [ ](const std::vector<TParticle>& particles) -> bool {
    return true; // triggered
  }
}
```

ADVANCED

👉 Example external conf.: GeneratorHFTrigger_Xi_Omega_C.ini

Gap triggered generators

- The event of interest (EOI) is selected only every n events → gaps are filled with min. bias events



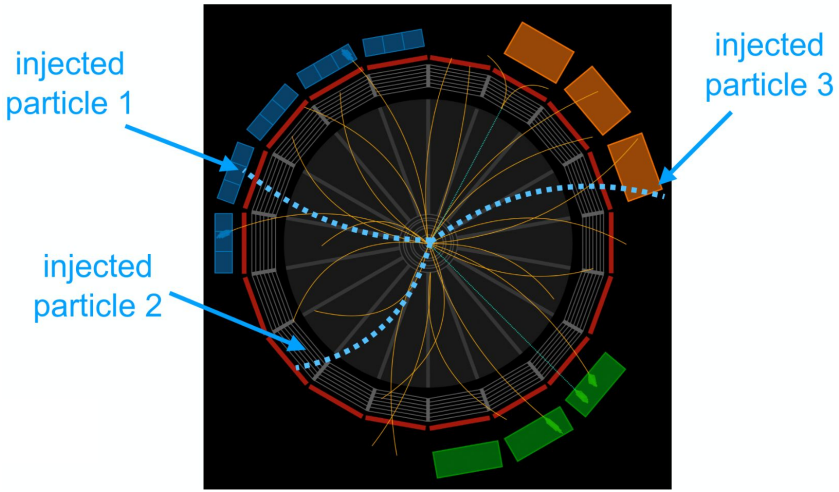
- MB events and EOI identified by sub-generator IDs:
  - mcCollision::getSubGeneratorId()=0 EOI
  - mcCollision::getSubGeneratorId()=1 gap (MB event)

- Feature mimics real data-taking conditions with continuous readout → gap configuration must be studied to find optimal solution (proper EOI/MB ratio)

ADVANCED

Signal embedding

- Particles can be injected (embedded) on top of an underlying event → the entire event is then called *cocktail*
- Injected particles and particles from the underlying event can have a different source ID:
  - mcCollision::getSourceId()=0 injected particles
  - mcCollision::getSourceId()=1 bkg particles



A word of advice…

Properties of real events containing triggered signal ⟷ (uncertain equality) properties MB + injected signal

A bias might be introduced → study real event properties, quantify potential bias and tune injection/properties

The Hybrid generator

- Configuring a custom simulation in ALICE framework can be cumbersome:
  - Long configuration keys to o2-sim in command line:

```
… -confKey
'GeneratorExternal.fileName=${O2DPG_MC_CONFIG_ROOT}/MC/config/PWGGAJE/external/generator/generator_pythia8_powheg.C;GeneratorExternal
.funcName=getGeneratorJEPythia8POWHEG(\"${PWD}/../powheg.input\",\"${PWD}/../pythia8_powheg_final.cfg\",2,10);'
```

  - Limitations in embedding and triggering ⇒ only one configuration could be parsed via command line
  - Generators combination was sequential only

- The hybrid generator has been introduced as a more configurable alternative for generators combination ⇒ it will become the default configuration method in the future
  - Generators are configured using JSON file
  - Simulation order is configurable:
    - Sequential
    - Uniformly distributed
    - Randomisation based on fractions
  - Possibility to run generators in parallel
  - Easier cocktail simulations
  - Triggers combination using and/or logic

The Hybrid generator: configuration

- A script is provided to create a JSON template for the configuration ⇒ [$O2DPG_ROOT/MC/bin/o2_hybrid_gen.py]

```
$O2DPG_ROOT/MC/bin/o2_hybrid_gen.py --gen pythia8hf --iniFile $PWD/test.ini --trigger --output config.json
```

- All generators in O2 are compatible with hybrid:
  - pythia8
  - pythia8hf,pythia8pp
  - evtpool
  - …
- Triggers can be configured with three options: and, or, off
  - Multiple macros can be provided with their own functions
- External generators can be configured both using fileName and funcName, or more easily directly with the iniFile →
- Few parameters are available to configure the generator behaviour:
  - configFile ⇒ to get the JSON configuration
  - randomize ⇒ if true randomisation will be enabled
  - num_workers ⇒ number of threads available for parallel event generation

```
“GeneratorHybrid.configFile=${PWD}/config.json;GeneratorHybrid.randomize=true”
```

config.json


```
{
  "mode": "sequential",
  "generators": [
    {
      "name": "pythia8hf",
      "config": "",
      "triggers": {
        "mode": "off",
        "specs": [
          {
            "macro": "",
            "function": ""
          }
        ]
      }
    },
    {
      "name": "external",
      "config": {
        "fileName": "",
        "funcName": "",
        "iniFile": "/home/test/test.ini"
      },
      "triggers": {
        "mode": "off",
        "specs": [
          {
            "macro": "",
            "function": ""
          }
        ]
      }
    }
  ],
  "fractions": [
    1,
    1
  ]
}
```

The Hybrid generator: cocktails

- Cocktail simulations were possible thanks to external custom generators:
  - Not very versatile
  - Requires prior knowledge on O2DPG generators development
  - Time consuming
- Hybrid generator allows now to combine multiple generators in a cocktail by simply specifying them in the JSON configuration
- Each event contains a sequence of outputs from each generator
- Very useful when users want to inject single particles species:
  - Example: Pythia8 + J/psi using box generator

config.json

```
{
  "generators": [
        {
          "cocktail" : [
          {
            "name": "pythia8",
            "config": {
              "config": "${PWD}/../pythia8_hi.cfg",
              "hooksFileName": "",
              "hooksFuncName": "",
              "includePartonEvent": false,
              "particleFilter": "",
              "verbose": 0
            }
          },
          {
            "name": "boxgen",
            "config": {
              "pdg": 443,
              "number": 100,
              "eta": [
                -5,
                5
              ],
              "prange": [
                0.1,
                10
              ],
              "phirange": [
                0,
                360
              ]
            }
          }
        ]
      }
  ],
  "fractions": [
    1
  ]
}
```

Important info in next slide for BoxGen injections

The Hybrid generator: cocktails

Important reminder:

- If particles injected with Box generator are not decayed by default
  ⇒ PYTHIA 8 decayer can be called during transport with GEANT4

`SimUserDecay.pdglist=443 421…` or with physics decay list `SimUserDecay.pdglist=443;DecayerPythia8.config[1]=~/jpsidie.cfg"`

- Multiple decay lists can be provided

jpsidie.cfg

```
443:onMode = off ### turn off all J/psi decays
443:onIfMatch = 11 -11 ### turn on only J/psi -> e+ e-
```

- Full decay options documentation is available in the official PYTHIA8 page

Event Pools: creation

- Some rare events or heavy simulations require a long computation time:
  - e.g. Ω_c^0 or Ξ_c^0, EPOS4 simulation

↓

Events can be stored and reused = lots of saved CPU time

- Event pools can be created using a simple simulation workflow that stops at the MC generation step without particles transport
  → a merging step takes care of multiple timeframes simulations
  - --make-evtpool for ${O2DPG_ROOT}/MC/bin/o2dpg_sim_workflow.py
  - -tt pool for ${O2DPG_ROOT}/MC/bin/o2dpg_workflow_runner.py

- Hybrid generator JSON is used to configure the creation of event pools:
  - Multiple threads can be used to speed up the simulation

```
{
  "mode": "parallel",
  "generators": [
    {
      "name": "external",
      "config": {
        "fileName": "",
        "funcName": "",
        "iniFile": "/home/test/iniFile.ini"
      }
    },
    {
      "name": "external",
      "config": {
        "fileName": "",
        "funcName": "",
        "iniFile": "/home/test/iniFile.ini"
      }
    },
    {
      "name": "external",
      "config": {
        "fileName": "",
        "funcName": "",
        "iniFile": "/home/test/iniFile.ini"
      }
    },
    {
      "name": "external",
      "config": {
        "fileName": "",
        "funcName": "",
        "iniFile": "/home/test/iniFile.ini"
      }
    }
  ],
  "fractions": [
    1,
    1,
    1,
    1
  ]
}
```

example.json

Event Pools: usage

- Event pools are saved as ‘evtpool.root’ files
- *extKinO2* is the base generator to work with single pools, but the recently developed *evtpool* generator implements additional logic in file management

Evtpool is the recommended choice when working with a large amount of files

An AliEN path can be provided → generator will take care of getting a random file among the ones in the folder

- Events in the pools can be randomised
- Events can also be reused ⇒ round-robin schedule or tracks ϕ rotation

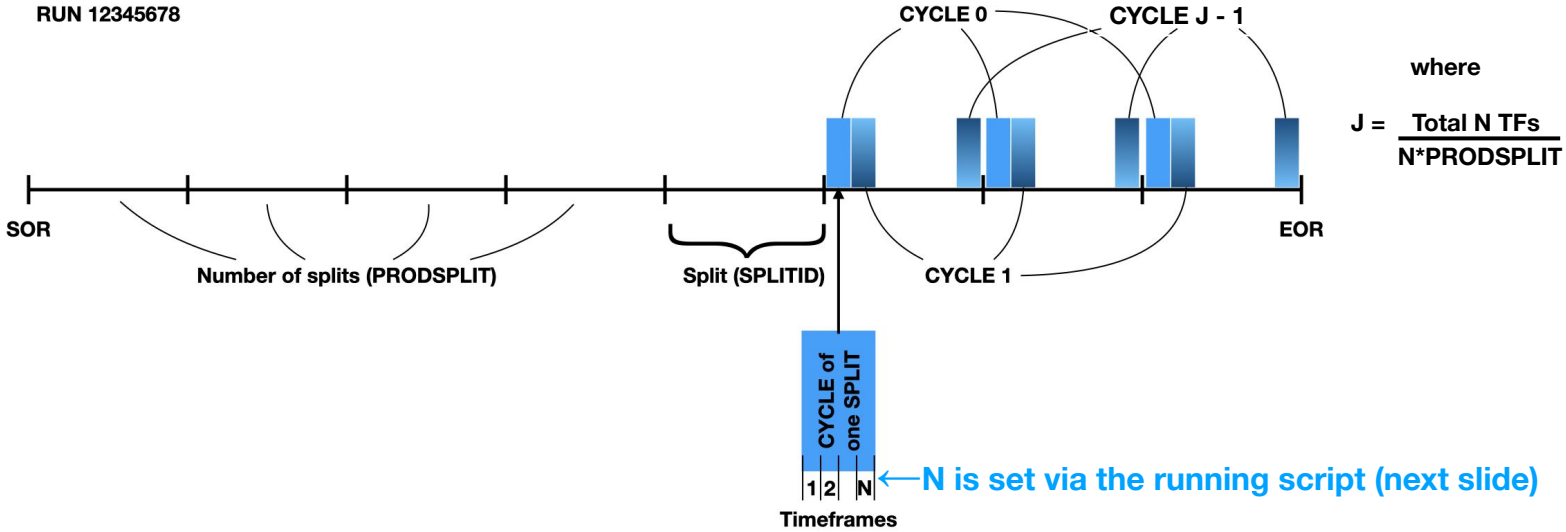
```
{
  "mode": "sequential",
  "generators": [
    {
      "name": "evtpool",
      "config": {
        "eventPoolPath": "alien:///alice/cern.ch/user/path",
        "skipNonTrackable": true,
        "roundRobin": false,
        "randomize": true,
        "rngseed": 0,
        "randomphi": false
      }
    }
  ],
  "fractions": [
    1
  ]
}
```

evtpool.json

Anchored MC productions

Documentation

- Simulations in which conditions are set to match those during a real data taking run → LHC filling scheme, included ALICE detectors, dead channels, alignment, interaction rate etc.
- These productions are crucial for physics analyses to have realistic simulated samples
- One anchored MC run corresponds to one specific CYCLE of one SPLITID containing N timeframes of the total→ full RUN covered when all CYCLEs are produced for all SPLITIDs



Anchored MC productions

Documentation

Example script for Anchored MC simulation →

```
export ALIEN_JDL_LPMANCHORPASSNAME=apass2
export ALIEN_JDL_MCANCHOR=apass2
export ALIEN_JDL_CPULIMIT=8
export ALIEN_JDL_LPMRUNNUMBER=535069
export ALIEN_JDL_LPMPRODUCTIONTYPE=MC
export ALIEN_JDL_LPMINTERACTIONTYPE=pp
export ALIEN_JDL_LPMPRODUCTIONTAG=LHC24a2
export ALIEN_JDL_LPMANCHORRUN=535069
export ALIEN_JDL_LPMANCHORPRODUCTION=LHC23f
export ALIEN_JDL_LPMANCHORYEAR=2023

export NTIMEFRAMES=1
export NSIGEVENTS=50
export SPLITID=100
export PRODSPLIT=153
export CYCLE=0

export SEED=5
export NWORKERS=2

${O2DPG_ROOT}/MC/run/ANCHOR/anchorMC.sh
```

One last topic

Rivet: a MC validation tool

Website

RIVET is a Python and C++ based framework

↓

Easy comparisons between MC simulations (in HepMC format) and experimental Data

↓

Published results pulled directly from HEPData website using YODA files

↓

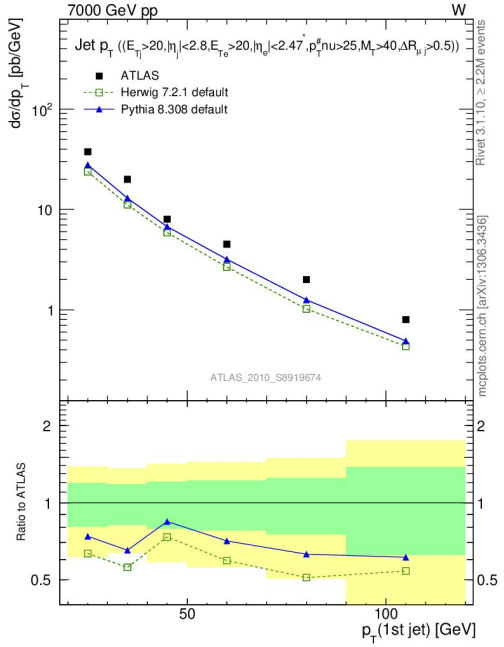
Custom YODAs can be easily generated by the user

↓

Phenomenological studies, MC/data validation, generators development

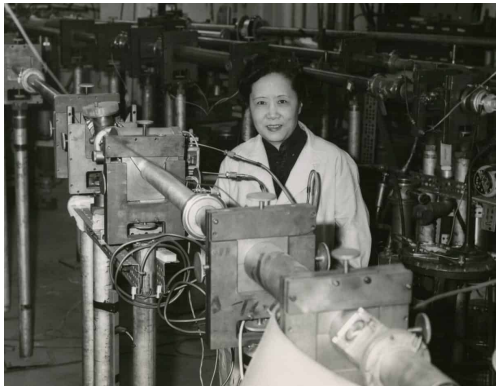
HEPData

Repository for publication-related High-Energy Physics data

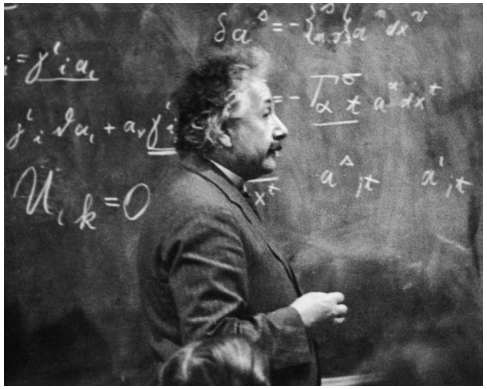


ATLAS example from MCplots repository
→ ALICE plugins will be inserted soon

Who is Rivet for?



Experimentalists
Are data coherent with simulation results?



Theorists
Can this new model describe better the experimental results?

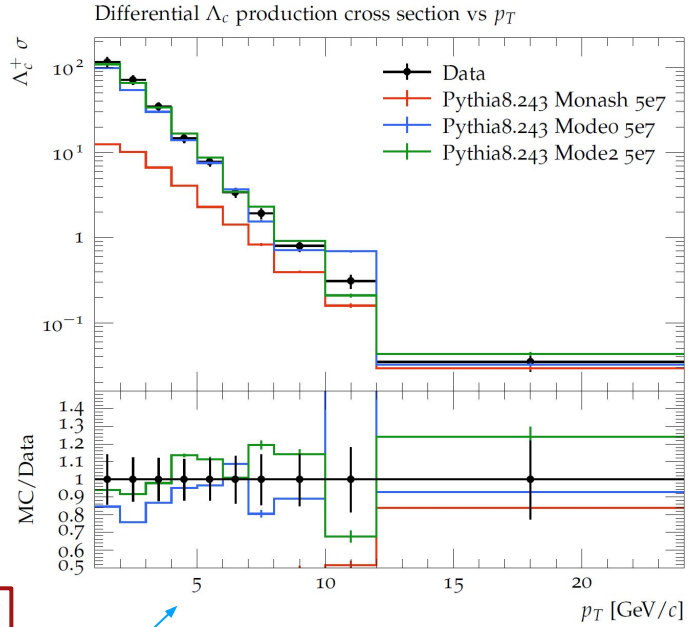
Relatively easy to use → anyone with a basic C++ understanding can use it

→ Valuable and powerful resource for students

Rivet: a MC analysis tool

RIVET is probably the most famous tool to compare experimental data to MC simulations

- Limited usage in ALICE → heavily used in other HEP experiments → Standardised MC comparison method
- Included in AliGenO2 → no additional package required
- Validation of generators and their development
- ALICE Primary particles definition is natively included in



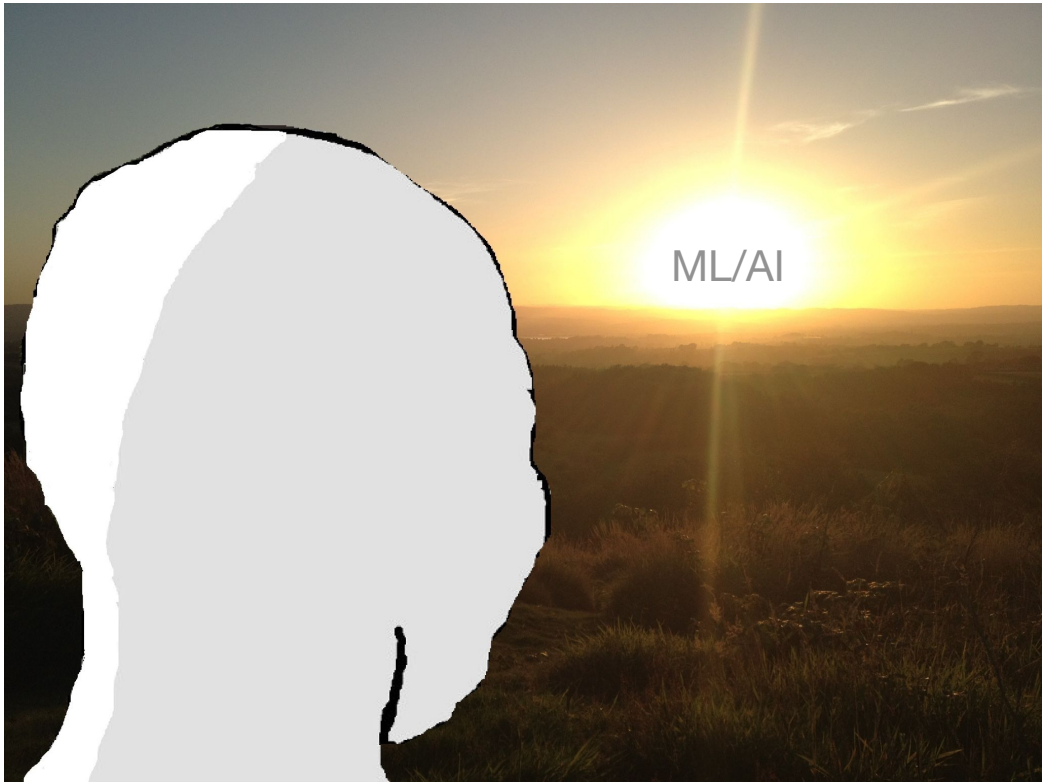
> A primary particle is a particle with a mean proper lifetime τ larger than 1 cm/c, which is either a) produced directly in the interaction, or b) from decays of particles with τ smaller than 1 cm/c, restricted to decay chains leading to the interaction.

- Great results with simple code

```
book(_h_Lc,2,1,1);
else if(p.abspid()==4122){
  _h_Lc->fill(p.pT()/GeV);
scale(_h_Lc,crossSection()/(microbarn*2*sumOfWeights()));
```

What’s next?

The future is bright…



Simulation lectures are over, but this is just a start