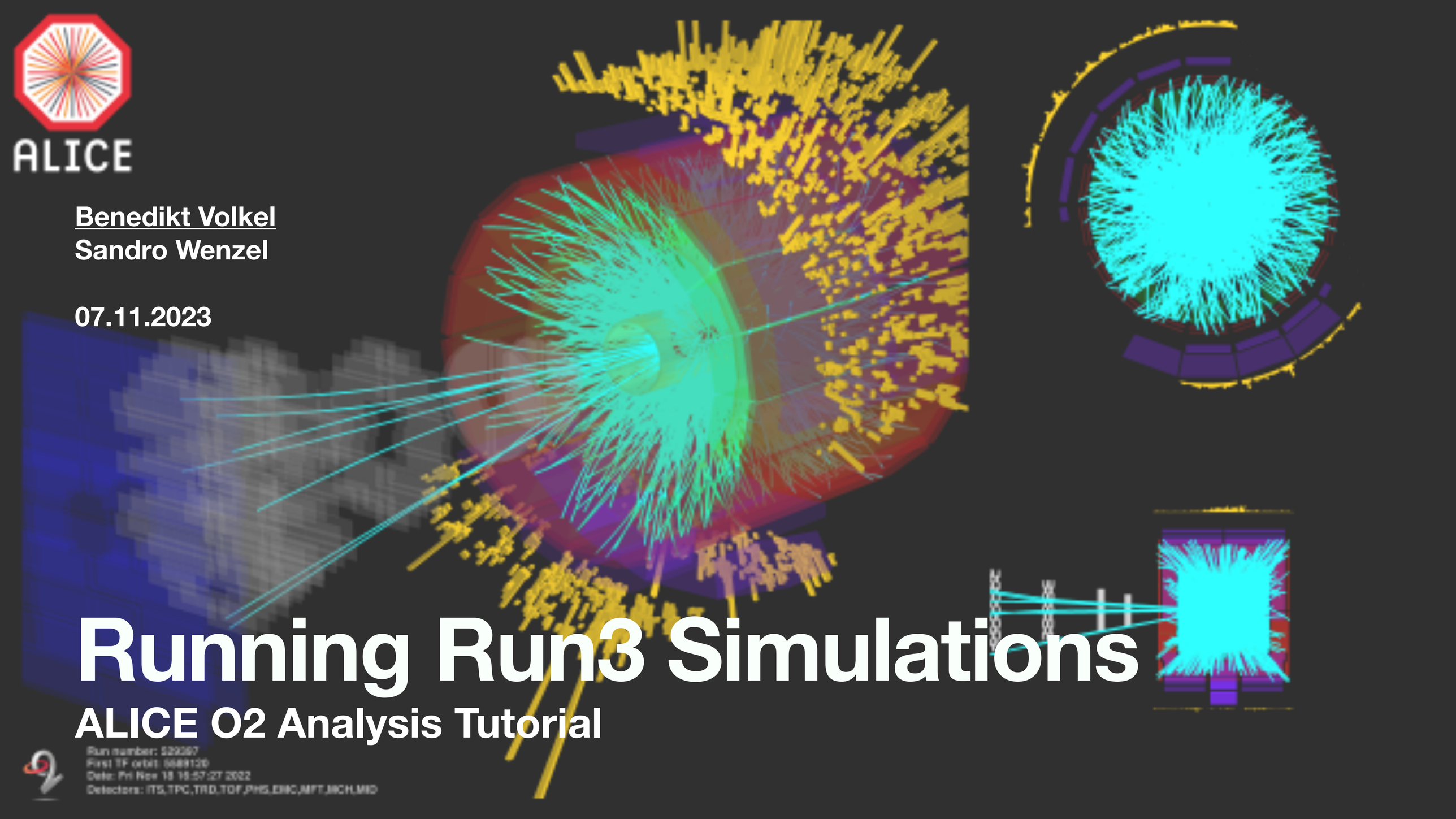


Benedikt Volkel
Sandro Wenzel

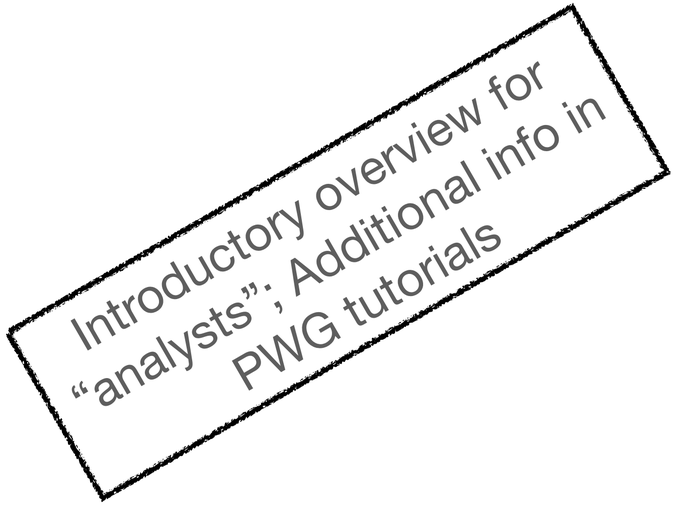
07.11.2023

Running Run3 Simulations

ALICE O2 Analysis Tutorial

Agenda

- Get an overview of ALICE Run3 ecosystem and know what to run
- To get to know fundamental uses of o2-sim (Event generation/transport simulation)
- Basic event generator use and configuration
- Be introduced to O2DPG simulation workflows
  —> Official integrated MC production pipeline from event generation to AOD and analysis



Contact information 📞

- How to get in touch with the simulation developers
  - Simulation e-group (for meeting announcements) + WP12 meetings
  - Collaborative Mattermost channels (preferred over private email): O2-simulation + O2DPG
  - JIRA tickets for feature requests/bug reports (components simulation or O2DPG)

- Where to find information about simulation
  - New documentation project: https://aliceo2group.github.io/simulation/
  - Prev. docu in AliceO2: DetectorSimulation.md
  - Some info in O2DPG: WorkflowRunner.md
  - Various examples at O2/SimExamples or nightly-tests

👈 still early stage: give feedback; ask questions; contribute !

Software environment reminder

local build (simulation —> AO2D, basic generators such as Pythia8)

```
aliBuild build O2 O2DPG --defaults o2

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

local build (also including QC, O2Physics and more generators)

```
aliBuild build O2sim --defaults o2

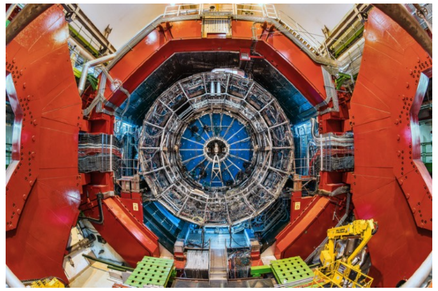
alienv enter O2sim/latest
```

nightly precompiled builds (CentOS)

```
/cvmfs/alice.cern.ch/bin/alienv enter O2sim::v20230419-1
```

Reminder of classical pipeline in a high-energy physics experiment and role of simulation

Real particle collisions



Virtual particle collisions + (detector) simulation based on physics models

~TB/s

sensor data

Reconstruction

Takes sensor data and reconstructs state of particles right after collisions

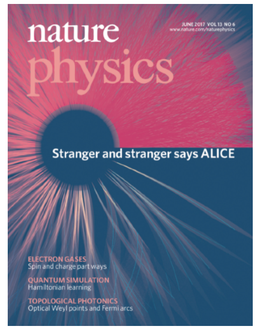
PetaBytes

AOD data

(Structured) high-level physics data be queried/ analysed

Physics Analysis

Data analysis to search for physics results and produce papers

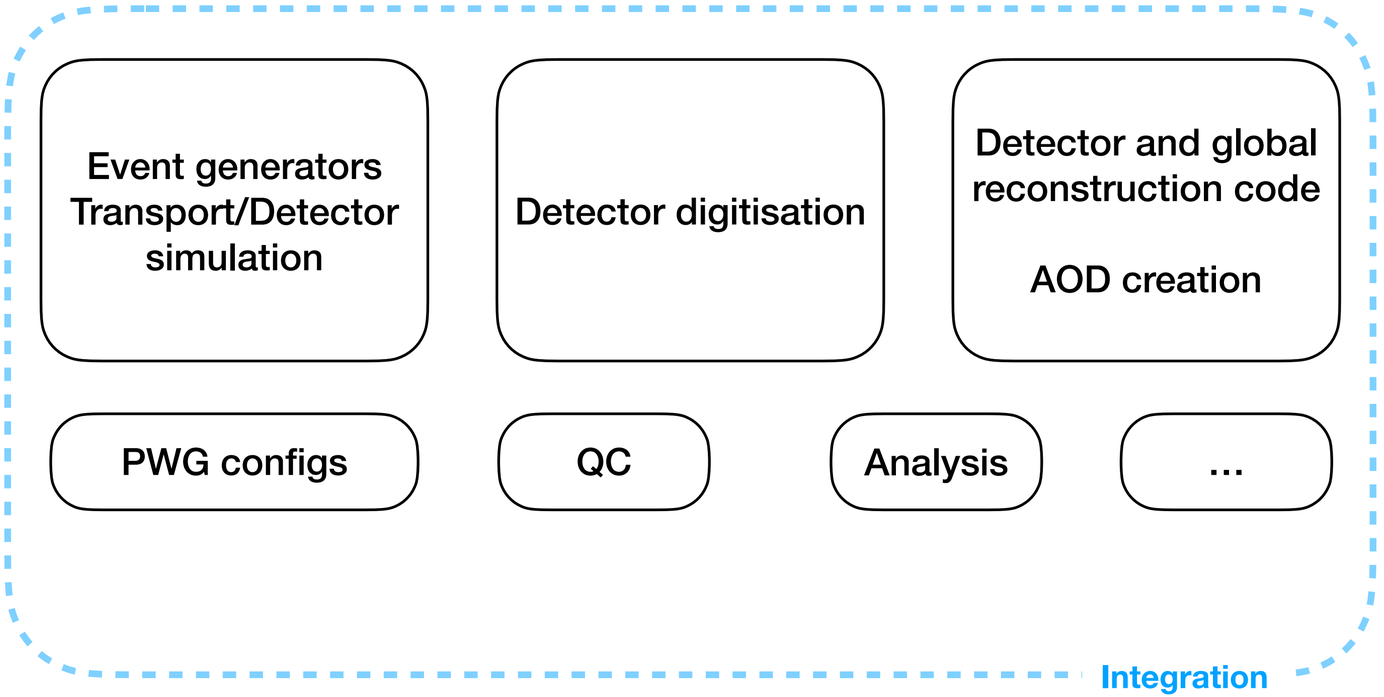


We are simulating for

- detector and systems design
- calibrating reconstruction algorithms
- knowing efficiency of reconstruction algorithms
- exercise data-taking system with synthetic data
- estimating background effects ….
- radiation studies
- etc.

Quick look into ALICE Run3 simulation ecosystem

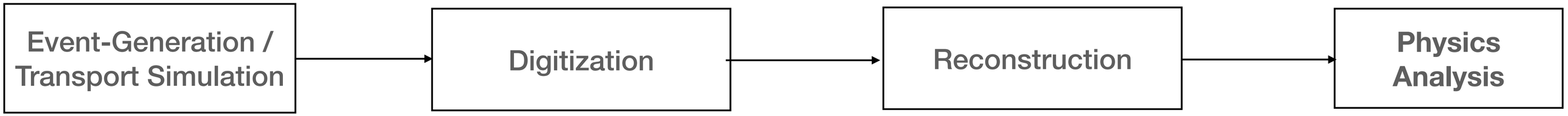
- Simulation ecosystem comprised of various components
- Core simulation part:
  - Event generation
  - Transport simulation
  - Digitization
- In addition, MC workflows may exercise all of
  - Reconstruction, QC, Analysis, etc.
- Sim -> AO2D maintained in O2
- Analyses, QC in O2Physics, QC repos



Integration and configuration of all parts into coherent workflows, done with:

- O2DPG repository (mainly for physics studies on GRID)
- full_system_test (mainly for data taking oriented simulations)

Data products in simulation pipeline



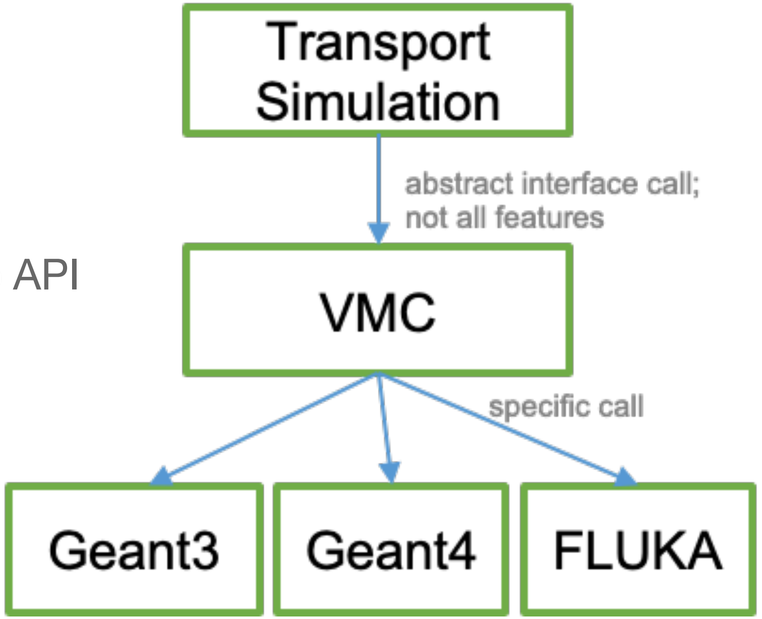
- Geometry file
- Kinematics file
- Detector response files (hits)

- Digits == detector sub-timeframes
- Comparable or close to raw detector output

- Global reconstructed tracks
- Primary + Secondary Vertexes
- etc.
- **AOD (analysis object data)**

o2-sim: The ALICE Run3 transport simulator

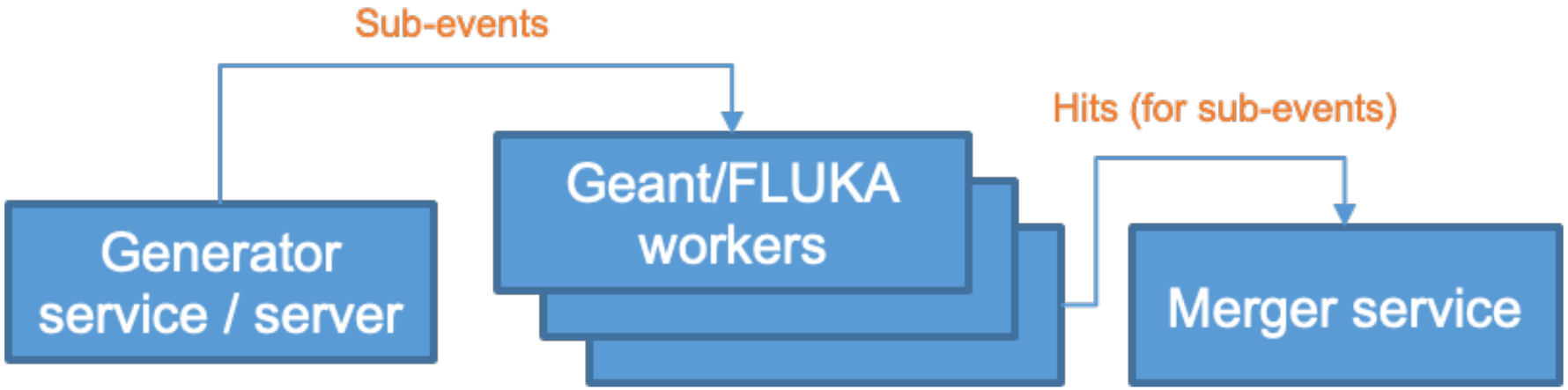
- **o2-sim is the particle-detector simulator** for ALICE Run3
- Implements ALICE detector on top of well known particle-transport engines that implement actual physics models and particle transport
  - Geant4, Geant3 and FLUKA interchangeably through use of Virtual Monte Carlo API
- Main tasks of o2-sim:
  - ALICE geometry creation
  - Event generation (primary particle generation)
  - Simulation of physics interaction of particles with detector material (secondary creation, etc.) and transport of particles until they exit detector or stop
  - Creation of hits (energy deposits) as a pre-stage of detector response after particle passage



o2-sim: The ALICE Run3 transport simulator

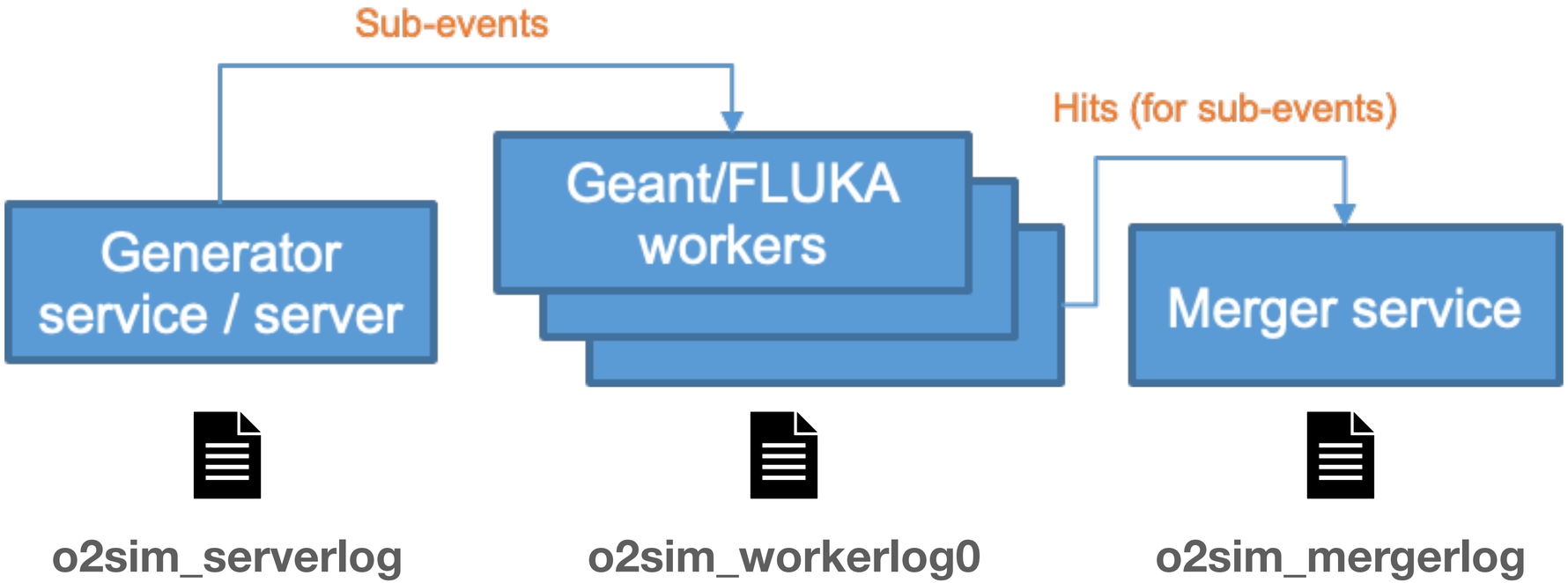
- New in Run3: **scalable multi-core simulation with sub-event parallelism**; allows to use big servers and obtain results for individual large events quickly

- Important: o2-sim treats **events in complete isolation** - **no timeframe concept** (enters during digitization)Geant4, Geant3 and FLUKA interchangeably through use of Virtual Monte Carlo API



o2-sim: The ALICE Run3 transport simulator

- o2-sim produces 3 internal log files (one from each micro-service), which is good to know about when you would like to see what is going on or to trouble-shoot



Basic usage of o2-sim in examples

- some examples how to use o2-sim …

```
o2-sim -n 10 -g pythia8pp
```

“Generate 10 default Pythia8 pp events and transport them through the complete ALICE detector”

```
o2-sim -n 10 -g pythia8pp -j 8 \
--skipModules ZDC --field 2 \
-e TGeant3
```

“Generate 10 default Pythia8 pp events and transport them with 8 Geant3 workers through everything but ZDC and use an L3-field of 2kGauss”

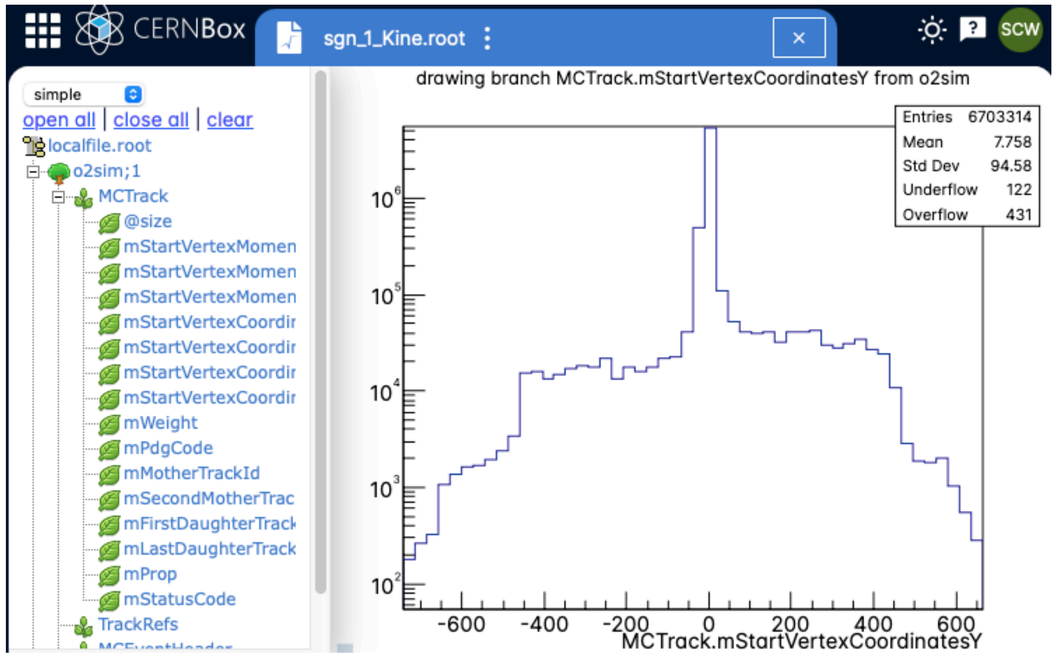
```
o2-sim -n 10 -g pythia8pp \
       --noGeant
```

“Just generate 10 default Pythia8 pp events and do nothing else (pure generator output)”

- `o2-sim --help` lists main options and shows defaults

o2-sim: Kinematics output

- Kinematics output (default file o2sim_**Kine.root**) from transport simulation likely most interesting for physics analysis
  - contains creation vertices, momenta, … of **primary (generator)** and **secondary (transport) particles** created in simulation
  - information on physics creation process, provenance (mother-daughter), etc.
  - Based on `o2::MCTrack` class, which is basically a more lightweight TParticle
- For each event, there is one entry of vector<MCTracks> in a TTree
- By default, kinematics is pruned (only relevant particles kept)
- In addition, event-level meta-information about each generated event is available in a separate file (o2sim_**MCHeader.root**)
  - for instance impact parameter of PbPb collision



"histogram of production vertex-y of all MCtracks (primary and secondary)"

👉 Example: Run/SimExamples/Jet_Embedding_Pythia8

Important helper classes to access MC kinematics

- Reading and navigating manually through kinematics can be cumbersome (“ROOT-IO boilerplate”)
- Offer 2 main utility classes making this easy for user
  - **MCKinematicsReader** - Class to easily read and retrieve tracks for given event or a Monte Carlo label
  - **MCTrackNavigator** - Class to navigate through mother-daughter tree of MC tracks and to query physics properties

```
using o2::steer;
using o2;

// access kinematics file with simulation prefix o2sim
MCKinematicsReader reader("o2sim",MCKinematicsReader::Mode::kMCKine);

// get all Monte Carlo tracks for this event
std::vector<MCTrack> const& tracks = reader.getTracks(event);

for (auto& t : tracks) {
   // analyse tracks; fetch mother track of each track (in the pool of all
tracks)
   auto mother = o2::mcutil::MCTrackNavigator::getMother(t, tracks);
   if (mother) {
      std::cout << "This track has a mother\n";
   }
   // fetch the (backward first) primary particle from which this track
derives
    auto primary = o2::mcutil::MCTrackNavigator::getFirstPrimary(t, tracks);
}
```

“Read all Monte Carlo tracks from stored kinematics file for event id 1. Then loop over all tracks and determine the direct mother particle and the primary ancestor in each case.”

Generators: Basic

- o2-sim has a couple of pre-defined generators (select with -g option)
  - phythia8pp (pre-configured Pythia8 for pp)
  - phythia8hi (pre-configured Pythia8 for PbPb)
  - pythia8powheg (pre-configured Pythia8 with POWHEG)
  - boxgen (a simple mono-PDG generator)
  - extkinO2 (use external kinematics file, e.g. generated in pre-step)
  - hepmc (take events from HepMC file)

👉 Example: run/SimExamples/JustPrimaryKinematics

👉 Example: run/SimExamples/HepMC_STARlight

```
o2-sim -g [ pythia8pp | pythia8hi | pythia8powheg | boxgen | extkinO2 | hepmc ] …
```

👉 Example: Run/SimExamples/Jet_Embedding_Pythia8

Generators: Phythia8

- Pythia8 is more deeply integrated in O2 (compared to others) and it is recommended to use Pythia8 whenever possible
- When Pythia8 is used, it can be fully configured via a special text file and the GeneratorPythia8 parameter
  - valid settings can be found in the Pythia8 reference manual
- we also provide a tool mkpy8cfg.py to help with the creation of the config file

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”
```

👉 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
  - link to 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;GeneratorExterna
l.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);
}
```

👉 Much more information in the documentation.

👉 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"'
```

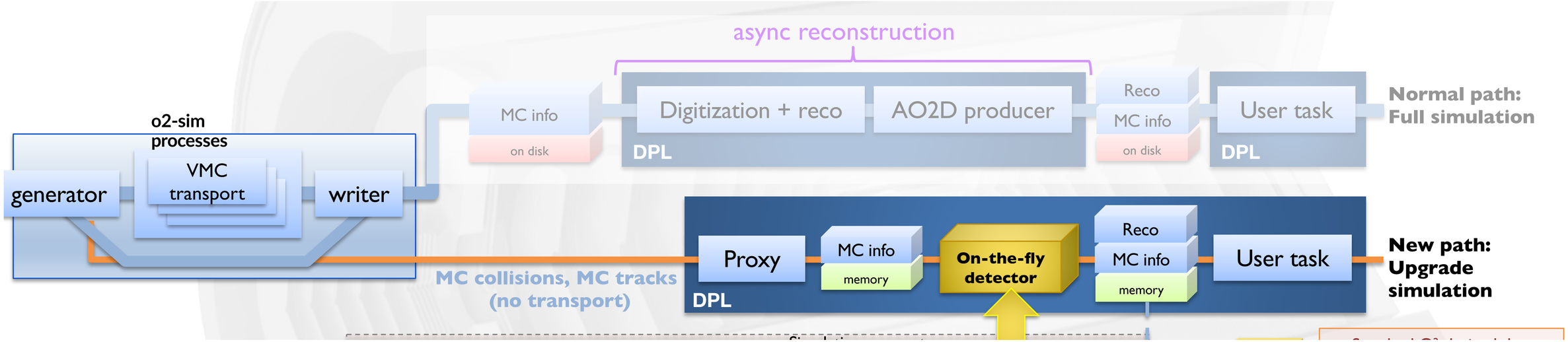
“stub content of ROOT macro file myTrigger.C”

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

👉 Example: Run/SimExamples/MCTrackToDPL

o2-sim as on-the-fly event generator for Analysis

- o2-sim can be used as generator service to inject events into a DPL (analysis) topology without intermediate storage
- useful for fast-simulation studies within analysis framework or for primary-only analysis tasks
- will be used/covered in tutorial of PWG-EM



@D. Chinellato

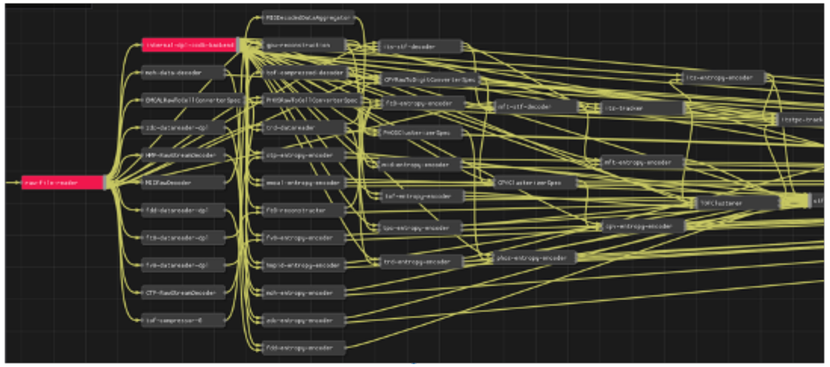
Integrated workflows: O2DPG MC

- 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)

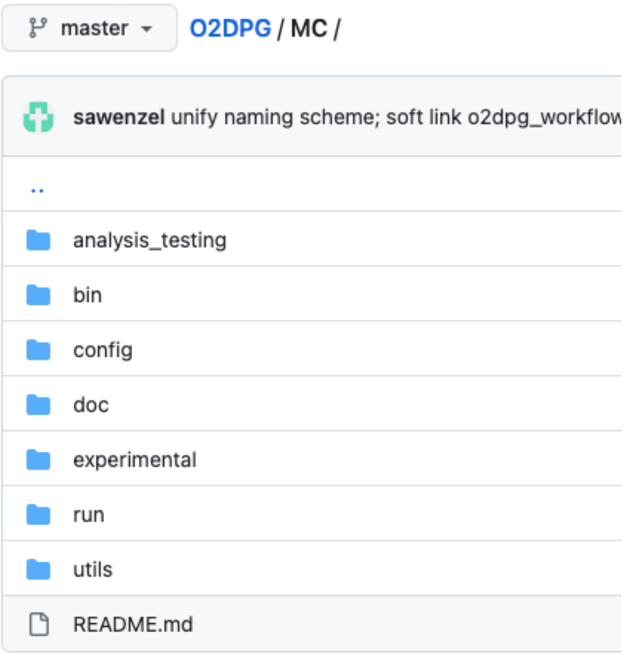


"Interplay of algorithms is a complex system (DPL topology)"

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



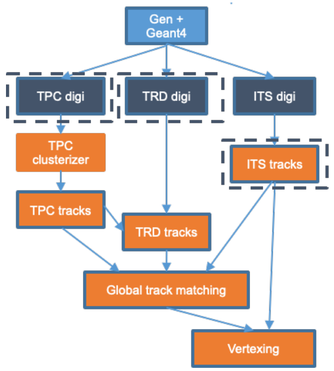
Important directories:

- 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
- Step One: Create a valid/configured description of a MC job == “workflow”
- Step Two: Run the MC job with a dynamic graph scheduler

Fundamentals of O2DPG-MC

Workflow creator → 

→ Workflow executor → Output (AOD)

workflow.json

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

☝ **Good to know:**
- DAG workflow is similar to a DPL topology
- however, DAG workflow executed in stages with intermediate files on disc
- DPL topology would not fit in GRID memory

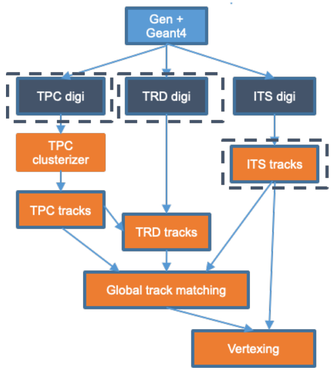
Takes care of running the DAG workflow on multi-core machines similar to a dynamic “build” tool

Launches tasks when they can be launched:
- Targeting high parallelism and CPU utilisation
- Respect resource constraints (tries not overload the system)

In principle a universal tool, not specific to MC

Fundamentals of O2DPG-MC

Workflow creator → workflow.json → Workflow executor → Output (AOD)



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

☝ **Good to know:**

- DAG workflow is similar to a DPL topology
- however, DAG workflow executed in stages with intermediate files on disc
- DPL topology would not fit in GRID memory

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 overload the system)

In principle a universal tool, not specific to MC

👉 Some examples here; 👉 docs here

MC workflow creation: Useful basic features

- 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.)
  - As usual, `o2dpg_sim_workflow.py --help`, lists possible 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
```

Important options are: -gen, -tf, -n, -eCM, -interactionRate, -run, -col

Optionally: -field, -seed, -proc

“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…”

Workflow creation: Run numbers

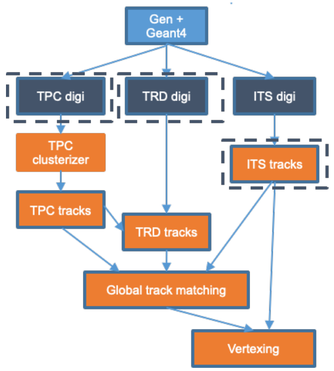
- The use of a run number is mandatory; as it will be used to determine a timestamp needed to fetch conditions from CCDB
- So run numbers should be used even for non-data-taking anchored simulations
- A list of pre-defined run numbers for MC has been documented here: https://twiki.cern.ch/twiki/bin/view/ALICE/O2DPGMCSamplingSchema
- 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*	

Fundamentals of O2DPG-MC

Workflow creator



workflow.json

Workflow executor

Output (AOD)

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

☝ **Good to know:**

- DAG workflow is similar to a DPL topology
- however, DAG workflow executed in stages with intermediate files on disc
- DPL topology would not fit in GRID memory

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 overload the system)

In principle a universal tool, not specific to MC

MC workflow execution: basic features

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

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

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

Good to know

- Checkpointing and incremental build

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

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

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

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

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

- … many more useful features

```
o2dpg_workflow_runner.py --help
```

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; 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

Tips and tricks: CCDB snapshots

- Multiple CCDB objects are required throughout the MC pipeline
- O2DPG MC workflow fetches each object **only once** and caches them as snapshots [avoid overhead of potentially requesting the same objects multiple times]
  - Default paths ${WORKDIR}/ccdb/<path>/<in>/<ccdb>/snapshot.root
- The path can be set manually by exporting `ALICEO2_CCDB_LOCALCACHE=/<your>/<path>`
- Allows for usage of
  - existing cache (get one from somebody else, run a reproducer etc.)
  - Pre-fetched specific objects or even user-constructed objects with custom settings

Tips and tricks: CCDB snapshots

- ```
  ${O2_ROOT}/bin/o2-ccdb-downloadccdbfile \
  --host http://alice-ccdb.cern.ch -p TPC/Calib/CorrectionMapRef \
  --timestamp <timestamp> --created-not-after 3385078236000 \
  -d ${YOURPATH}
  ```
- `ALICEO2_CCDB_LOCALCACHE=${YOURPATH} o2_dpg_workflow_runner.py …`
- …
  ```

Tips and tricks: O2DPG developments

- Developments in O2DPG can be done without any aliBuild re-build.
- O2DPG is “simply” a collection of scripts, macros and configuration files.

```
$> alienv enter O2/latest
$> export O2DPG_ROOT=/<path>/<to>/<your>/O2DPG
$> # now you can run, change something in /<path>/<to>/<your>/O2DPG and run again and again
```

Tips and tricks: O2DPG custom generators

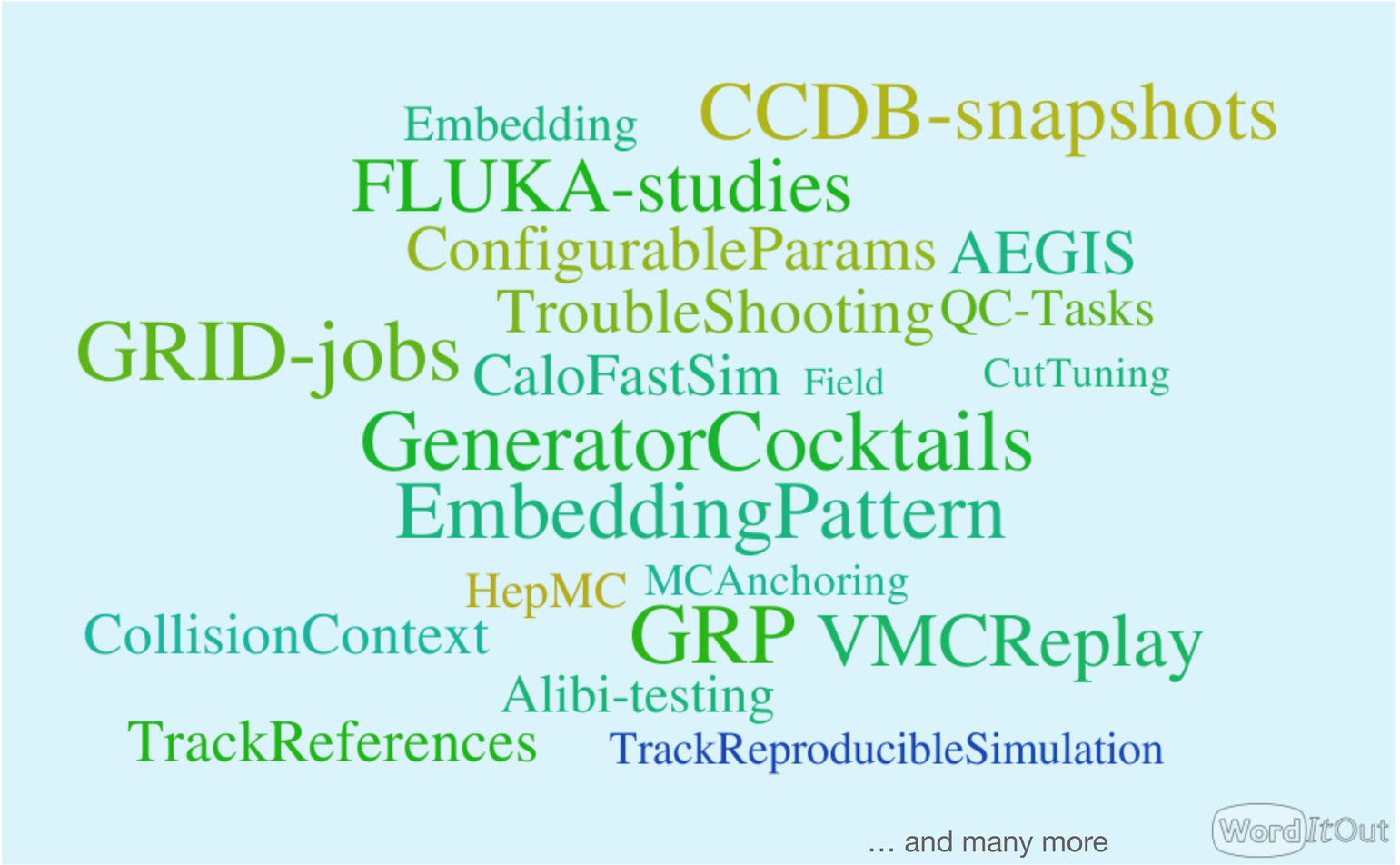
- Custom PWG generators will be tested during a PR before they can be merged
- This is to make sure that the generator and its configuration do what they are supposed to do before being used in production
- Tests can be done locally as well (also before opening PRs or to investigate CI failures)
- All necessary information in the documentation.

Tips and tricks: very custom MC workflows

- You might want to test custom/very specific configuration of the MC pipeline.
  To do so
  - Create the workflow
  - Open and customise the created workflow.json
  - Run the workflow

Beyond here: Additional keywords

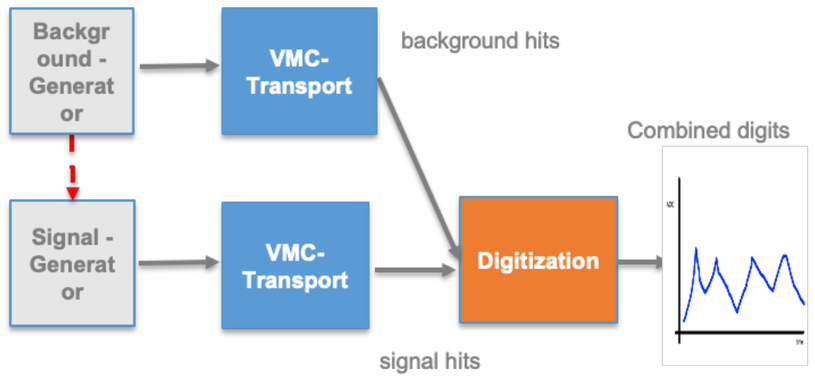
- Many more expert topics to be covered
- … going beyond this basic set of slides …
- Contact us for information !
- Help us improving the documentation which is still in an early stage !



Backup

Digitization, embedding (signal mixing)

- Digitization is likely less relevant for physics analysis purposes … yet important to have heard about it
- Fundamental task of digitization is to
  - convert simple energy deposits into detector signals (digits) which finally resemble raw detector output
  - actually put individual generated events into a timeframe collection
  - account for pileup effects and triggering
- Digitization as signal embedding / mixing framework
  - Digitization can be used as an event-mixing / event-embedding framework
  - signal-background embedding allows to inject signal events into a repeated collection of background events (saves transport simulation time)
  - Can engineer sequences of event types within a timeframe (a signal event after every n-th min-bias event)



👉 Embedding example in O2DPG: PWGHF embedding