INTRODUCTION TO O²PHYSICS

O² ANALYSIS TUTORIAL

USEFUL LINKS:

Documentation

Mattermost O2

Mattermost O2 tutorial

REFERENCES:

https://indico.cern.ch/event/1326201/timetable/

O^2 Introduction

Here we try to explain what the general idea behind the framework is, and walk you through what you need to do in order to write an analysis task, its parts and how you get stuff done.

So what is it that we mean when we do analysis in O^2 and O^2Physics and what do we actually use?How do we deal with data? how do we deal with operations on the data?

How do we we write specifically an analysis task and make it run? that's the whole point

In the past you could think of it as storing your events in objects in C++

- AliPhysics: object-oriented, arrays of structures-based
  - ultimately: restrictive for analysis speed!

You had an event and within the event you had particles and that had certain hierarchy
This is nice except SPEED WISE, different size. Hard in memory acces

NEW!

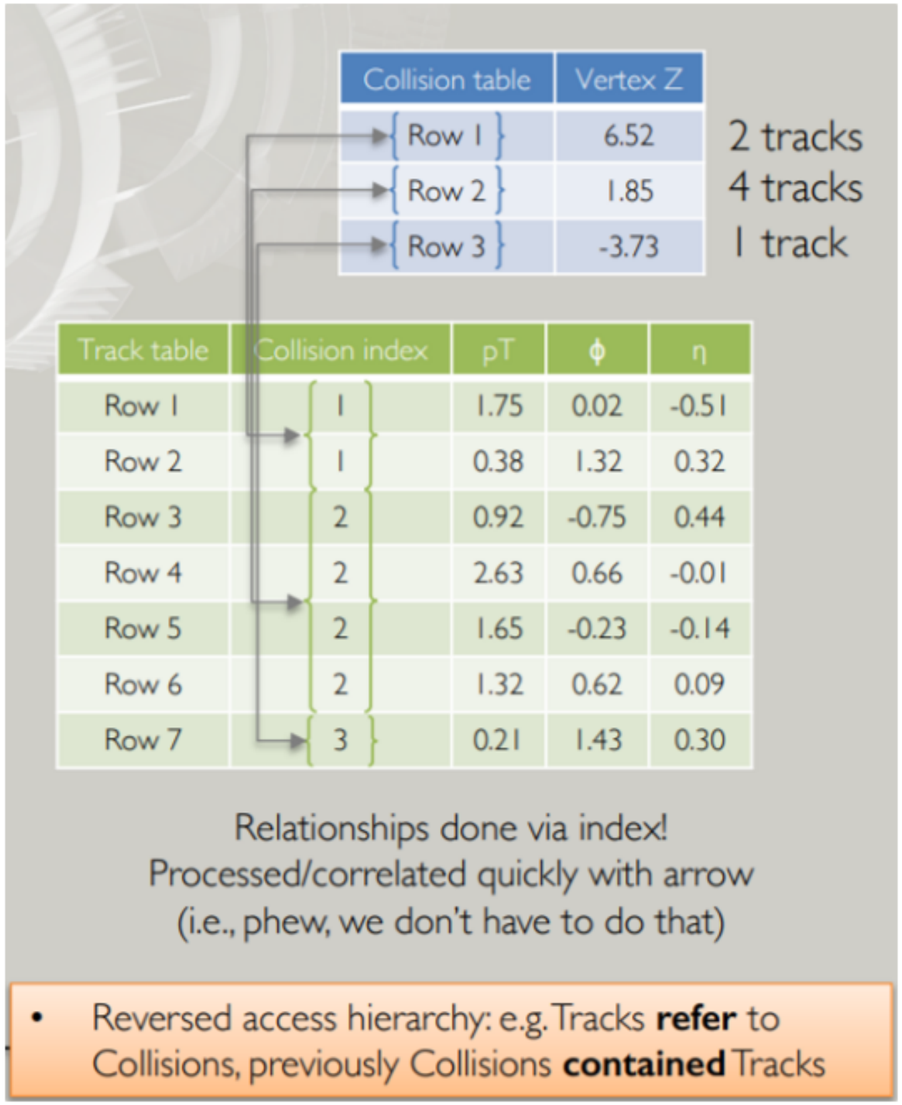
- Major switch to **arrow tables** https://arrow.apache.org/
  - structures of arrays-based, enormous processing speed unlocked!

Instead of objects we now have arrays which are correlated so you can link between them. Same size.
This leads to vectorization, run in parallel, operations at the same time.

Root files are still the I/O back-end 🙂
T Trees —O^2–> table

O^2 AODs (AO2Ds) can be inspected with the browser

TABLES:



- You can still use **for,** but there are better ways to do a loop
- Because they are tables you can use splitting instead of an **if**

Looping over a table: using iterators

• Every table has an automatically defined iterator: soa::Table::iterator
• An iterator is used to access table content:

```
for (auto const& track : tracks) {
histogram.fill(track.pt());
}
```

- **const**: helps with compiler optimizations
- **&**: use the iterator by reference → faster!

• An iterator can be used to access individual columns by calling corresponding getters

– roughly equivalent to an object in the old framework - collision, track, etc.
– The iterator can be incremented and decremented,
– The iterator can be moved to a certain row
– The iterator can be copied and compared to other iterators (via !=)

• Tables that contain a soa::Index<> column have access to enumerating methods:

– .index(), .globalIndex(), .filteredIndex()

Writing an analysis task! basics

What an analysis task looks like? There's a standard form of writing an analysis task:

```
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"
using namespace o2;
using namespace o2::framework;

struct ATask { //declare as a structure
    HistogramRegistry histos{"histos", {}};
    //2 pieces
    init() {}; //← initialize: configure, create specifics
    process(inform framework on what data you want to run) {
          //process data: where everything is suppose to happen
    };

};
//definitions for the framework know what you're doing
WorkflowSpec defineDataProcessing(ConfigContext const& cfgc) {
return WorkflowSpec{adaptAnalysisTask<ATask>(cfgc)};
}
```

Basically, the argument inside process is extremely flexible,

```
process(inform framework on what data you want to run)
```

you can tell the framework:

- I want to look over tracks
- I want to look over collisions and for each one give me the associated tracks
- Just one to look at a specific detector signal
- etc …

This are "subscriptions", you SUBSCRIBE to the table you want to access. You SHOULD BE minimalistic

Looping over all tracks

Simple example in which we loop over all tracks:

```
struct ATask {
  HistogramRegistry histos{"histos", {}}; //Way to store your output
  init() {
    histos.add(”hPhi", " hPhi ", kTH1F, {{100, 0., 2. * M_PI}}}); //initialize hist
  };
  process(aod::Tracks const& tracks) { //process tracks
    for (auto track : tracks) {
    histos.fill(HIST("hPhi"), track.phi()); //fill histogram for each track
    }
  };
};
```

- This loops over all tracks, literally, doesn't check the index.
- We have no idea which track associates with which collision, ridiculous FAST!

Ex. ϕ distribution

What if i want to know which events that ϕ belongs to? what if you want to do correlations (Ex. $\Delta\phi$) I want to do stuff in the SAME collision, so i need to GROUP the tracks according to the collision index.

What if i want to know which events that ϕ belongs to? what if you want to do correlations (Ex. $\Delta\phi$) I want to do stuff in the SAME collision, so i need to GROUP the tracks according to the collision index.

I want to loop over one collision, look at only the tracks associated with it. SO we write a subscription like this one:

```
process(o2::aod::Collision const& collision, aod::Tracks const& tracks)
```

Crucial thing: collision (singular), tracks (plural)

*collision is an iterator

```
struct ATask {
  HistogramRegistry histos{"histos", {}}; //Way to store your output
  init() {
    histos.add(”hPhi", " hPhi ", kTH1F, {{100, 0., 2. * M_PI}}}); //initialize hist
    histos.add(”hEvCount", "hEvCount", kTH1F, {{1, 0., 1.}}});
  };
  process(o2::aod::Collision const& collision, aod::Tracks const& tracks) { //proce
    histos.fill(HIST(" hEvCount"), 0.5);
    for (auto track : tracks) { //you are looping only the corresponding tracks to
    histos.fill(HIST("hPhi"), track.phi()); //fill histogram for each track
    }
  };
};
```

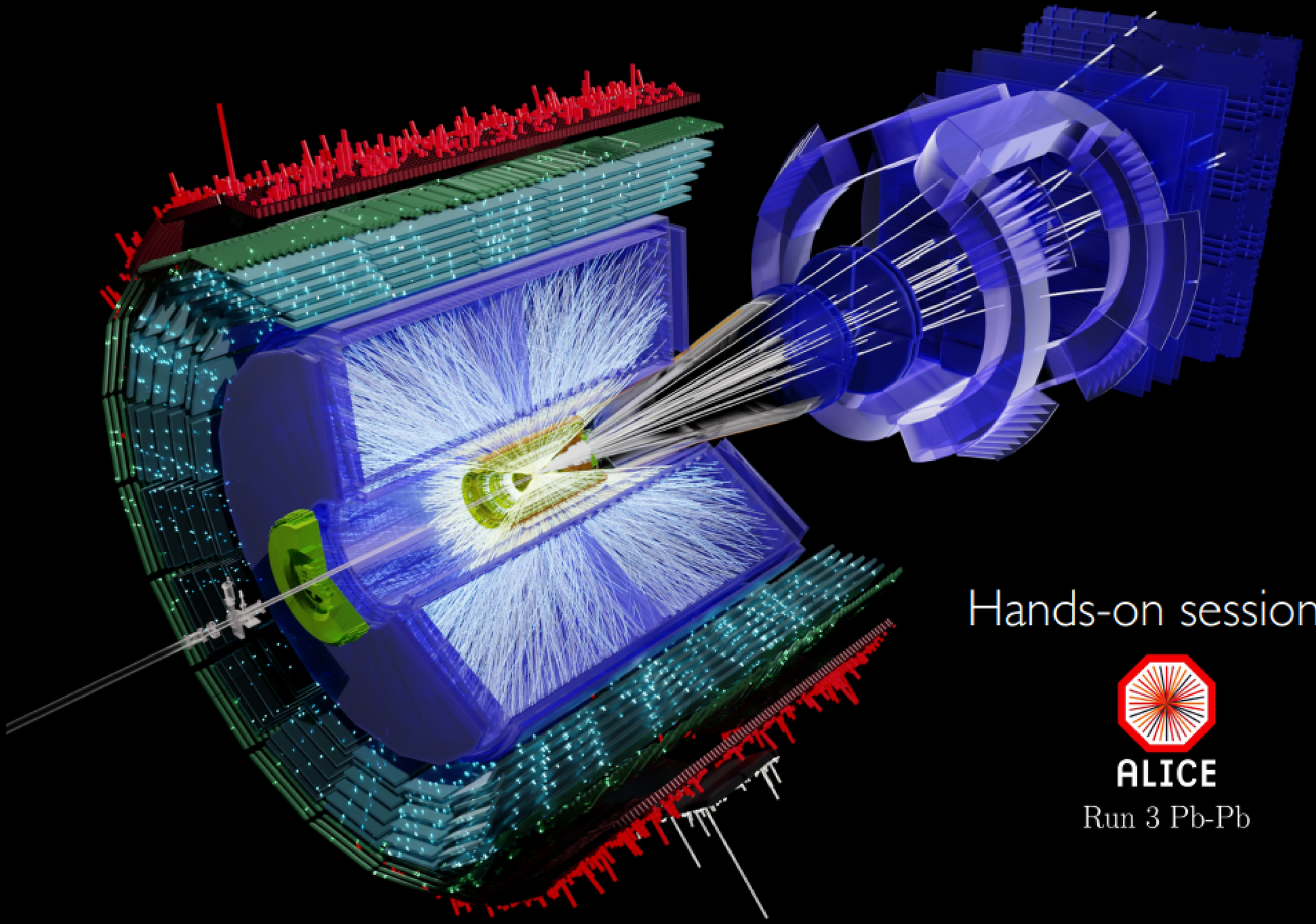
Defining axes

```
struct ATask {
  HistogramRegistry histos{"histos", {}};

  init() {
    AxisSpec phiAxis = {100, 0., 2. * M_PI};
    histos.add("hEvCount", " hEvCount", {HistType::kTH1F, {{1, 0.0f, 1.0f}}});
    histos.add("phi", " phi", {HistType::kTH1F, {phiAxis}});
  };
  process(o2::aod::Collision const& collision, o2::aod::Tracks &tracks) {
    registry.fill(HIST("hCandidateCounter"), 0.5);
    for (auto track : tracks) {
      registry.fill(HIST("phi"), track.phi());
    }
  };
};
```

What if I wanted to configure the output somehow? Number of bins, for instance?

What if I wanted to configure the output somehow? Number of bins, for instance?

```
struct ATask {
  Configurable<int> nBinsPhi{"nBinsPhi", 100, "N bins in phi histo"}

  HistogramRegistry registry{
    "registry", {},
  };

  init() {
    AxisSpec ptAxis = {200, 0.0f, 10.0f, "it{p}_{T} (GeV/c)"};
    registry.add("hEvCount", " hEvCount", {HistType::kTH1F, {{1, 0.0f, 1.0f}}});
    registry.add("phi", " phi", {HistType::kTH1F, {{nBinsPhi, 0., 2. * M_PI}}});
  };
  process(o2::aod::Collision const& collision, o2::aod::Tracks &tracks) {
    registry.fill(HIST("hCandidateCounter"), 0.5);
    for (auto track : tracks) {
    registry.fill(HIST("phi"), track.phi());
    }
  };
};
```



Hands-on session I

ALICE

Run 3 Pb-Pb

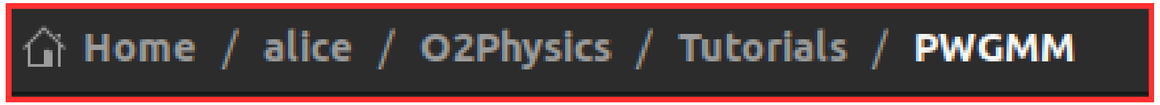
Data for this exercise

• Single AO2D:

– CERNBox

– Dropbox

*Agregar el AO2D que descargaste en tu carpeta de ALICE en este directorio



myExampleTask.cxx:

```cpp
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"

using namespace o2;
using namespace o2::framework;

struct myExampleTask {
  // Histogram registry: an object to hold your histograms
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(InitContext const&)
  {
    // define axes you want to use
    const AxisSpec axisEta{30, -1.5, +1.5, "#eta"};

    // create histograms
    histos.add("etaHistogram", "etaHistogram", kTH1F, {axisEta});
  }

  void process(aod::TracksIU const& tracks)
  {
    for (auto& track : tracks) {
      histos.fill(HIST("etaHistogram"), track.eta());
    }
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
{
  return WorkflowSpec{
    adaptAnalysisTask<myExampleTask>(cfgc)};
}
```

The executable name: **o2-analysistutorial-mm-my-example-task**

So, to run :

```
o2-analysistutorial-mm-my-example-task --aod-file AO2D.root
```

myExampleTask.cxx:

```cpp
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"

using namespace o2;
using namespace o2::framework;

struct myExampleTask {
  // Histogram registry: an object to hold your histograms
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(InitContext const&)
  {
    // define axes you want to use
    const AxisSpec axisEta{30, -1.5, +1.5, "#eta"};

    // create histograms
    histos.add("etaHistogram", "etaHistogram", kTH1F, {axisEta});
  }

  void process(
  {
    for
      h
    }
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
{
  return WorkflowSpec{
    adaptAnalysisTask<myExampleTask>(cfgc)};
}
```

The executable name: **o2-analysistutorial-mm-my-example-task**

So, to run :

o2-analysistutorial-mm-my-example-task --aod-file AO2D.root

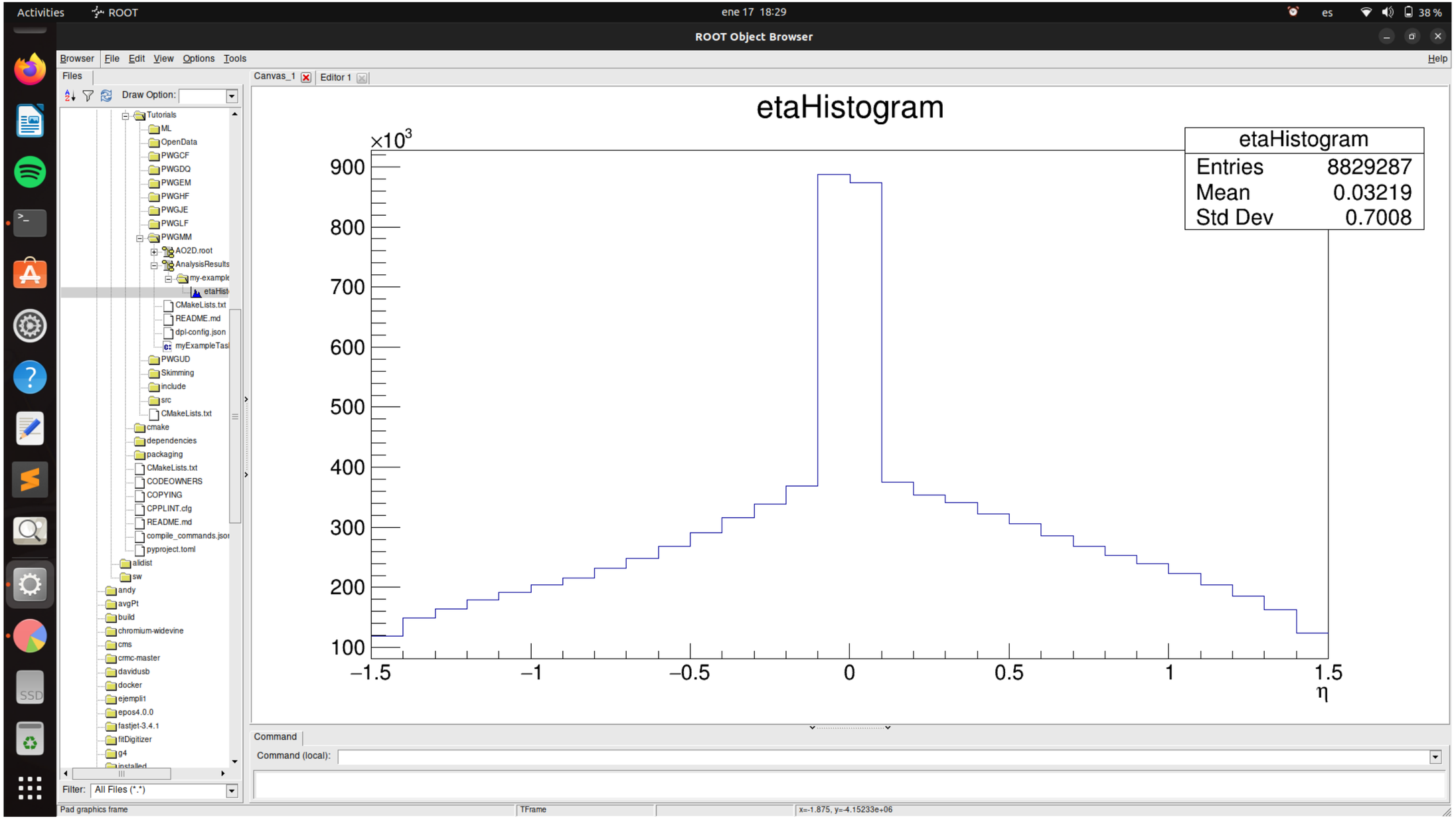
If you get this error message:

```
[O2Physics/latest-master-o2] ~/alice/O2Physics/Tutorials/PWGMM $> o2-analysistutorial-mm-my-example-task --aod-file AO2D.root
: CommandLine Error: Option 'use-dbg-addr' registered more than once!
LLVM ERROR: inconsistency in registered CommandLine options
Aborted (core dumped)
```

add -b:

o2-analysistutorial-mm-my-example-task -b --aod-file AO2D.root

```
[74762:internal-dpl-clock]: [18:21:31][STATE] RESETTING TASK ---> DEVICE READY
[74762:internal-dpl-clock]: [18:21:31][STATE] DEVICE READY ---> RESETTING DEVICE
[74766:internal-dpl-aod-global-analysis-file-sink]: [18:21:31][STATE] RESETTING DEVICE ---> IDLE
[74766:internal-dpl-aod-global-analysis-file-sink]: [18:21:31][STATE] IDLE ---> EXITING
[74766:internal-dpl-aod-global-analysis-file-sink]: [18:21:31][STATE] Exiting FairMQ state machine
[74762:internal-dpl-clock]: [18:21:31][STATE] RESETTING DEVICE ---> IDLE
[74762:internal-dpl-clock]: [18:21:31][STATE] IDLE ---> EXITING
[74762:internal-dpl-clock]: [18:21:31][STATE] Exiting FairMQ state machine
[74765:my-example-task]: [18:21:31][STATE] RESETTING DEVICE ---> IDLE
[74765:my-example-task]: [18:21:31][STATE] IDLE ---> EXITING
[74765:my-example-task]: [18:21:31][STATE] Exiting FairMQ state machine
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Received device shutdown request (signal 15).
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Waiting for graceful device shutdown. Hit Ctrl-C again to abort immediately.
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] New state requested. No timeout set, quitting immediately as per --completion-policy
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] RUNNING ---> READY
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] READY ---> RESETTING TASK
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] RESETTING TASK ---> DEVICE READY
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] DEVICE READY ---> RESETTING DEVICE
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Cleaning up for shared memory id 'c58d9607'...
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Found 0 unmanaged regions...
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Found 1 managed segments...
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Successfully removed 'fmq_c58d9607_m_0'.
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][INFO] Successfully removed 'fmq_c58d9607_mng'.
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] RESETTING DEVICE ---> IDLE
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] IDLE ---> EXITING
[74767:internal-dpl-injected-dummy-sink]: [18:21:31][STATE] Exiting FairMQ state machine
[74764:internal-dpl-aod-spawner]: [18:21:31][STATE] RESETTING DEVICE ---> IDLE
[74764:internal-dpl-aod-spawner]: [18:21:31][STATE] IDLE ---> EXITING
[74764:internal-dpl-aod-spawner]: [18:21:31][STATE] Exiting FairMQ state machine
[74763:internal-dpl-aod-reader]: [18:21:31][STATE] RESETTING DEVICE ---> IDLE
[74763:internal-dpl-aod-reader]: [18:21:31][STATE] IDLE ---> EXITING
[74763:internal-dpl-aod-reader]: [18:21:31][STATE] Exiting FairMQ state machine
[INFO] ## Processes completed. Run summary:
[INFO] ### Devices started: 6
[INFO]  - Device internal-dpl-clock: pid 74762 (exit 0)
[INFO]  - Device internal-dpl-aod-reader: pid 74763 (exit 0)
[INFO]  - Device internal-dpl-aod-spawner: pid 74764 (exit 0)
[INFO]  - Device my-example-task: pid 74765 (exit 0)
[INFO]  - Device internal-dpl-aod-global-analysis-file-sink: pid 74766 (exit 0)
[INFO]  - Device internal-dpl-injected-dummy-sink: pid 74767 (exit 0)
[INFO] ## Analysis Run Summary ##
[INFO] ### Files read stats ###
[INFO] lfn=AO2D.root,size=1049011267,total_df=28,read_df=28,read_bytes=179386851,read_calls=700,io_time=0.9,wait_time=0.0,level=0
[INFO] Dumping used configuration in dpl-config.json
[O2Phys[redacted]als/PWGMM $>
```


Activities
ROOT
ene 17 18:29
es
38 %
ROOT Object Browser
Browser
File
Edit
View
Options
Tools
Help
Files
Draw Option:
Canvas_1
Editor 1
etaHistogram
×10^3
900
800
700
600
500
400
300
200
100
−1.5
−1
−0.5
0
0.5
1
1.5
η
etaHistogram
Entries 8829287
Mean 0.03219
Std Dev 0.7008
Tutorials
ML
OpenData
PWGCF
PWGDQ
PWGEM
PWGHF
PWGJE
PWGLF
PWGMM
AO2D.root
AnalysisResults
my-example
CMakeLists.txt
README.md
dpl-config.json
PWGUD
Skimming
include
src
CMakeLists.txt
cmake
dependencies
packaging
CMakeLists.txt
CODEOWNERS
COPYING
CPPLINT.cfg
README.md
pyproject.toml
alidist
sw
andy
avgPt
build
chromium-widevine
cms
crmc-master
davidusb
docker
ejemplI1
epos4.0.0
fastjet-3.4.1
fitDigitizer
g4
Command
Command (local):
Filter:
All Files (*.*)
Pad graphics frame
TFrame
x=-1.875, y=-4.15233e+06

How do you run something?

- Each analysis task is an executable → this means you can run them in the command line!

```
o2-analysistutorial-histograms --aod-file AO2D.root | o2-analysis-track-propagation | o2-analysis-timestamp
```

Example task — Input file — Helper task: Propagates tracks to PV — Helper task: Provides timestamps

- All tasks have to be provided separated with a 'pipe' character ("|")
- Typically, many helper tasks are required: we will introduce you to this in the first task
- General event (centrality/multiplicity percentile) and track properties (PID values) have to be calculated!

➢ **Helper task** are for carrying out advance analysis.

Your first changes to O2Physics !

- Warning! Only modify O2Physics/Tutorials/PWGMM/myExampleTask.cxx
- Once modified, you can redo aliBuild build O2Physics and only this task will recompile

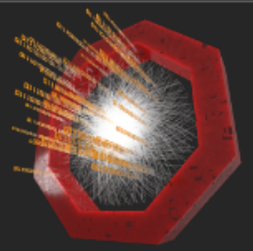
1. By copying the same logic as the etaHistogram, create a ptHistogram that stores the pt of the track.

   a) Create an AxisSpec for storing the desired momentum ranges
   b) Use the new AxisSpec when adding the ptHistogram to the ‘histos’ histogram registry
   c) When filling out the new histogram, how should you know the getter? Check our documentation!

Name		Getter	Type	Comment
o2::aod::track::Pt	E	pt	float	Transverse momentum of the track in GeV/c

Use: track.pt()

2. Let’s make also the number of bins in pT configurable!
   a) Create an int configurable: Configurable<int> nBinsPt{"nBinsPt", 100, "N bins in pT histo"};
   b) Use this integer when defining the AxisSpec, i.e. const AxisSpec axisPt{nBinsPt, 0, 10, "p_{T}"}
   Then: o2-analysistutorial-mm-my-example-task --aod-file AO2D.root



Building partially for development using ninja

This requires that the O2Physics build succeeded. Enter the environment as explained in the previous step specifying in addition the ninja package:

```
alienv enter O2Physics/latest ninja/latest
```

Go to the build directory

```
cd ~/alice/sw/BUILD/O2Physics-latest/O2Physics
```

You can now rebuild and install entire O2Physics with

```
ninja install
```

*if you dont want to use aliBuild

or just a specific directory with

```
ninja <directory>/install
```

For example:

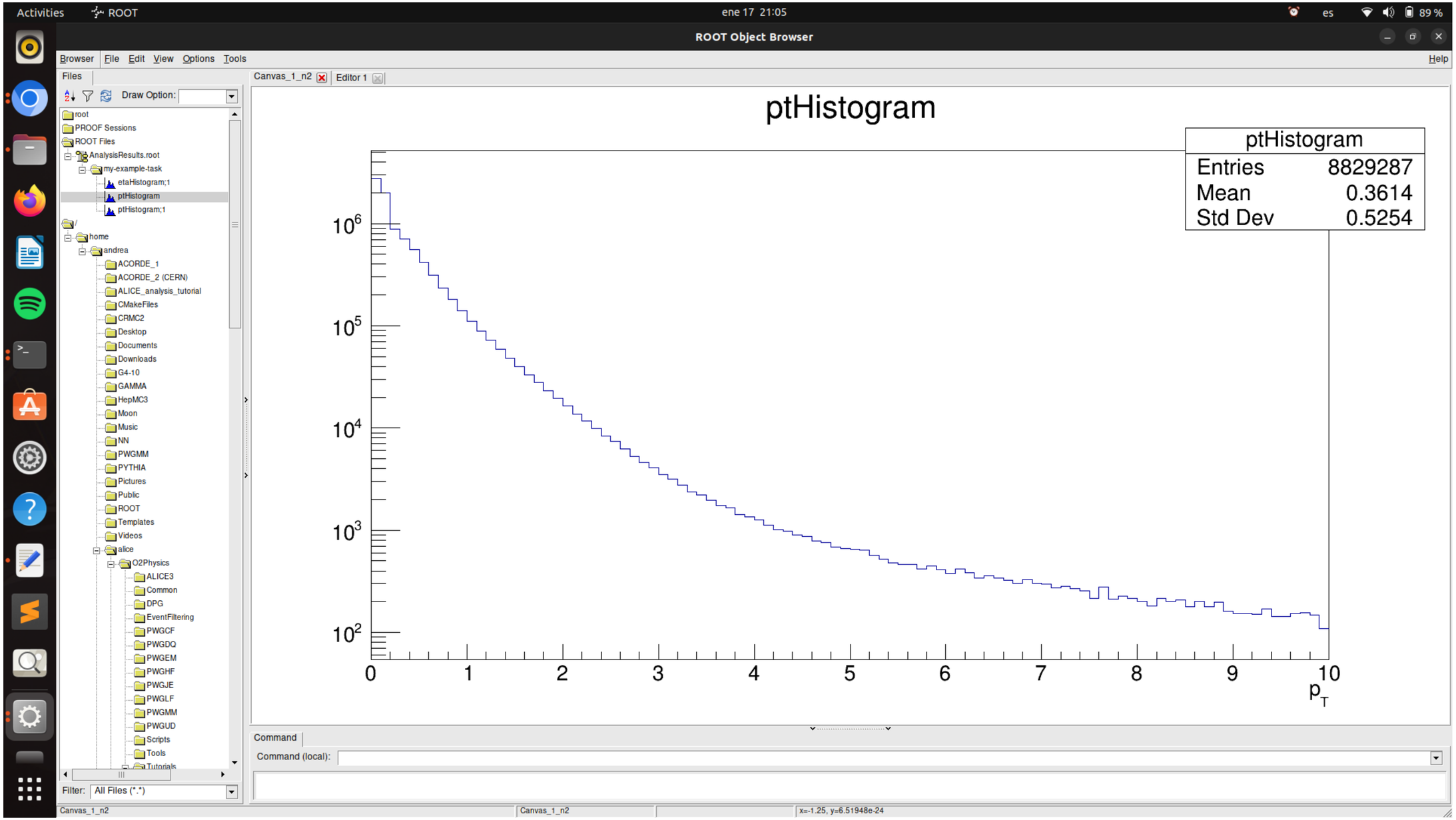
```
ninja PWGCF/Tasks/install
```

You can redirect the terminal output to the standard aliBuild log file and see whether the build succeeded:

```
ninja install > ../log 2>&1 && echo "Good" || echo "Bad"
```

A specific executable can be built in the staging directory `stage/bin` with

```
ninja stage/bin/<target>
```

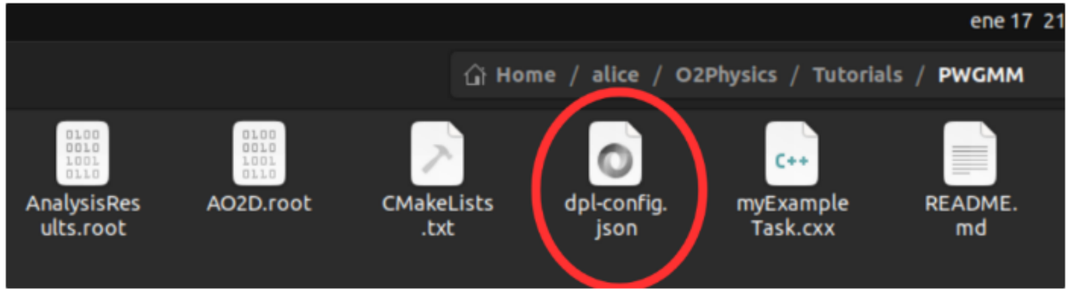

Activities
ROOT
ene 17 21:05
es
89 %
ROOT Object Browser
Browser File Edit View Options Tools Help
Files
Draw Option:
root
PROOF Sessions
ROOT Files
AnalysisResults.root
my-example-task
etaHistogram;1
ptHistogram
ptHistogram;1
home
andrea
ACORDE_1
ACORDE_2 (CERN)
ALICE_analysis_tutorial
CMakeFiles
CRMC2
Desktop
Documents
Downloads
G4-10
GAMMA
HepMC3
Moon
Music
NN
PWGMM
PYTHIA
Pictures
Public
ROOT
Templates
Videos
alice
O2Physics
ALICE3
Common
DPG
EventFiltering
PWGCF
PWGDQ
PWGEM
PWGHF
PWGJE
PWGLF
PWGMM
PWGUD
Scripts
Tools
Tutorials
Filter: All Files (*.*)
Canvas_1_n2
Editor 1
ptHistogram
ptHistogram
Entries 8829287
Mean 0.3614
Std Dev 0.5254
10^6
10^5
10^4
10^3
10^2
0 1 2 3 4 5 6 7 8 9 10
p_T
Command
Command (local):
Canvas_1_n2
Canvas_1_n2
x=-1.25, y=6.51948e-24

Dealing with jsons

- Your last execution ended with a message such as:

[INFO] Dumping used configuration in dpl-config.json

This means that the configuration you used was dumped in a json already.



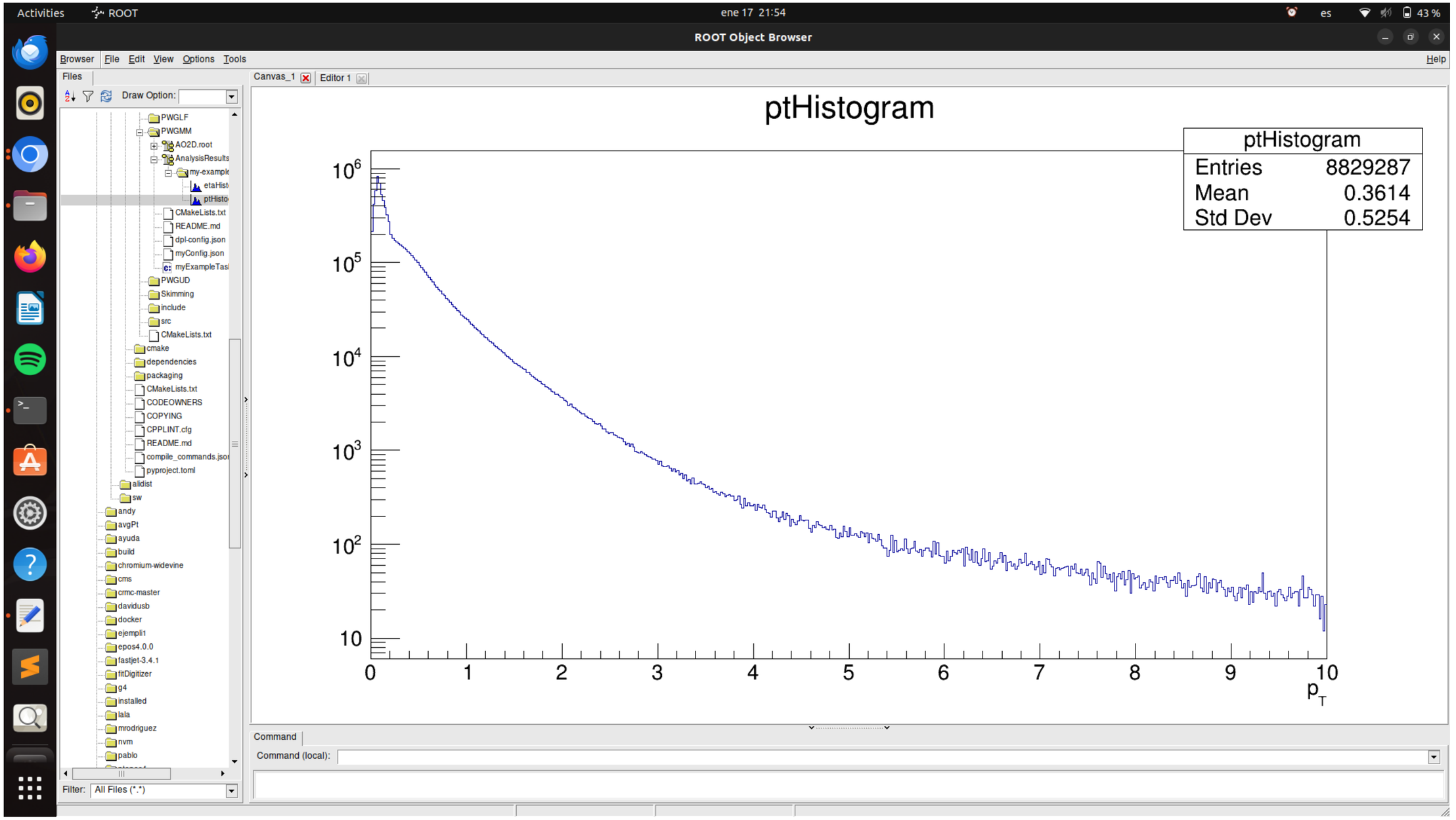
Let's use that to our advantage and just modify it:

```
[O2Physics/latest] ~/O2AT $> cat dpl-config.json
{
    "internal-dpl-clock": "",
    "internal-dpl-aod-reader": {
        "time-limit": "0",
        "orbit-offset-enumeration": "0",
        "orbit-multiplier-enumeration": "0",
        "start-value-enumeration": "0",
        "end-value-enumeration": "-1",
        "step-value-enumeration": "1",
        "aod-file": "AO2D.root"
    },
    "internal-dpl-aod-spawner": "",
    "my-example-task": {
        "nBinsPt": "100"
    },
    "internal-dpl-injected-dummy-sink": "",
    "internal-dpl-aod-global-analysis-file-sink": ""
}
[O2Physics/latest] ~/O2AT $>
```

```
[O2Physics/latest] ~/O2AT $> cat dpl-config.json
{
    "internal-dpl-clock": "",
    "internal-dpl-aod-reader": {
        "time-limit": "0",
        "orbit-offset-enumeration": "0",
        "orbit-multiplier-enumeration": "0",
        "start-value-enumeration": "0",
        "end-value-enumeration": "-1",
        "step-value-enumeration": "1",
        "aod-file": "AO2D.root"
    },
    "internal-dpl-aod-spawner": "",
    "my-example-task": {
        "nBinsPt": "500"
    },
    "internal-dpl-injected-dummy-sink": "",
    "internal-dpl-aod-global-analysis-file-sink": ""
}
[O2Physics/latest] ~/O2AT $>
```

Now, let's copy the dpl-config json into something like myConfig.json and pass that as argument!
– No need to provide AO2D.root as the aod-file anymore: this is already done inside the new json!

`o2-analysistutorial-mm-my-example-task --configuration json://myConfig.json`


Activities
ROOT
ene 17 21:54
es
43 %
ROOT Object Browser
Browser
File
Edit
View
Options
Tools
Help
Files
Draw Option:
Canvas_1
Editor 1
ptHistogram
ptHistogram
Entries 8829287
Mean 0.3614
Std Dev 0.5254
10^6
10^5
10^4
10^3
10^2
10
0 1 2 3 4 5 6 7 8 9 10
p_T
PWGLF
PWGMM
AO2D.root
AnalysisResults
my-example
etaHist
ptHisto
CMakeLists.txt
README.md
dpl-config.json
myConfig.json
myExampleTask
PWGUD
Skimming
include
src
CMakeLists.txt
cmake
dependencies
packaging
CMakeLists.txt
CODEOWNERS
COPYING
CPPLINT.cfg
README.md
compile_commands.json
pyproject.toml
alidist
sw
andy
avgPt
ayuda
build
chromium-widevine
cms
crmc-master
davidusb
docker
ejemplo1
epos4.0.0
fastjet-3.4.1
fitDigitizer
g4
installed
lala
mrodriguez
nvm
pablo
Filter:
All Files (*.*)
Command
Command (local):

It looks even uglier. We need an extra selection on the tracks

Track table only contains basic tracks, you need the tracks EXTRA TABLE cause that contains all the information of the detectors that were used.

For example, we can select tracks that points only to the primary vertex, for that we use the trackDCA TABLE.

This means we'll have to change our subcription

```
process(aod::Collision const& collision, soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA> const& tracks)
```

Tracks will be provided grouped per collision: the first argument is the collision iterator.
Remember the lecture in the morning…

- Before the track loop, add a fill command to an hEventCounter histogram with only one bin:

```
histos.fill(HIST("hEventCounter"), 0.5); // remember to add this histogram to histos!
```

- Inside the track loop, we can now use new getters defined in TracksExtra and TracksDCA:

```
tpcNClsCrossedRows(); and .dcaXY();
```

- The simplest (but not the best) approach is to use "if" inside your track loop:

```
if( track.tpcNClsCrossedRows() < 70 ) continue; //badly tracked
if( fabs(track.dcaXY()) > 0.2) continue; //doesn't point to primary vertex
```

Warning: the dcaXY information requires a new header! Add this to the includes:
#include "Common/DataModel/TrackSelectionTables.h"

Running with this additional information

• Now it will not be enough to run only:

`o2-analysistutorial-mm-my-example-task --configuration json://myConfig.json`

• If you try, you will get a message that the 'Tracks' information is missing. It will be like this:

```
[1304982:internal-dpl-aod-reader]: [17:02:59][ERROR] Exception caught: Couldn't
get TTree "DF_1/O2track" from "AO2D.root". Please check https://aliceo2group.git
hub.io/analysis-framework/docs/troubleshooting/#tree-not-found for more informat
ion.
```

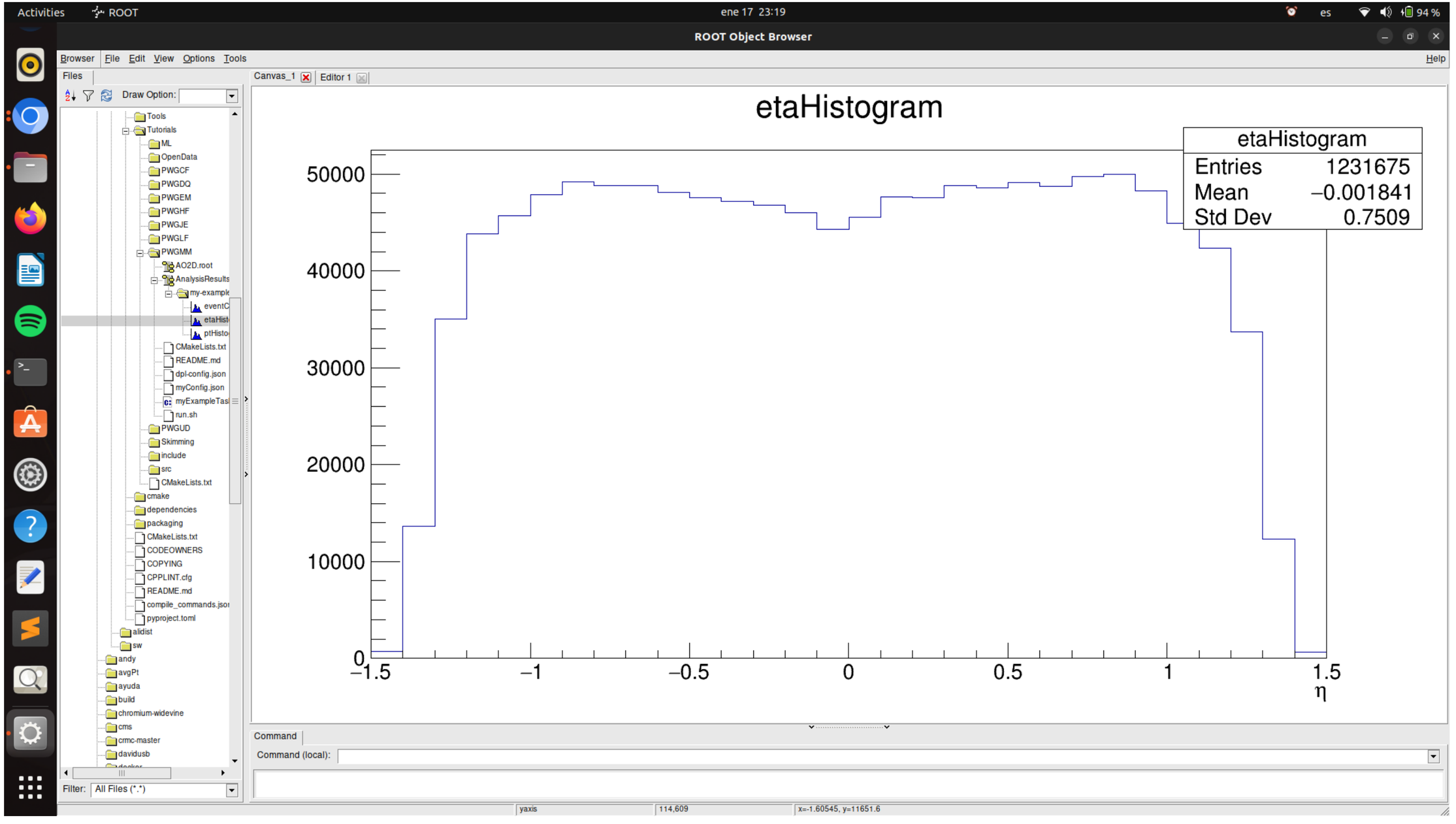
• In this case, you need the track propagation task to generate the tracks table (no IU!) and the dcaXY information for you. This means you will need an extended command line that is:

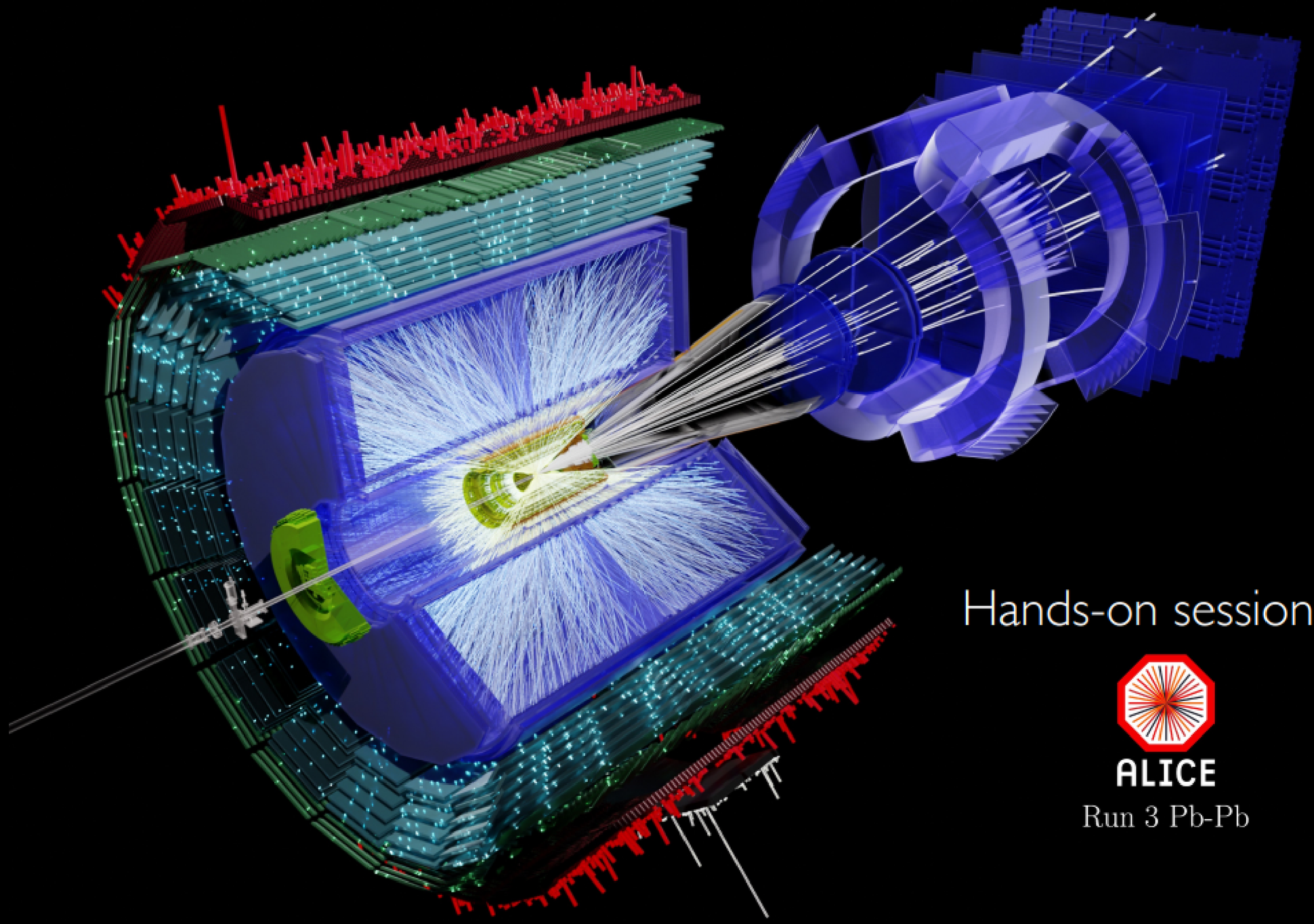
```
o2-analysistutorial-mm-my-example-task --configuration json://myConfig.json
o2-analysistutorial-mm-my-example-task --configuration json://myConfig.json |
o2-analysis-track-propagation --configuration json://myConfig.json |
o2-analysis-timestamp --configuration json://myConfig.json |
o2-analysis-tracks-extra-converter --configuration json://myConfig.json
```

• The time stamp information is needed by the track propagation task to find out the magnetic field.
• At this stage, this command is getting a bit large! It helps to transform this into a shell script…

OR A simple shell script to aggregate what you need! (run.sh)

```
OPTION="-b --configuration json://myConfig.json"

o2-analysistutorial-mm-my-example-task ${OPTION} | o2-analysis-track-propagation
${OPTION} | o2-analysis-timestamp ${OPTION} | o2-analysis-tracks-extra-converter
${OPTION}
```


Activities
ROOT
ene 17 23:19
es
94 %
ROOT Object Browser
Browser
File
Edit
View
Options
Tools
Help
Files
Draw Option:
Canvas_1
Editor 1
etaHistogram
etaHistogram
Entries 1231675
Mean −0.001841
Std Dev 0.7509
50000
40000
30000
20000
10000
0
−1.5
−1
−0.5
0
0.5
1
1.5
η
Tools
Tutorials
ML
OpenData
PWGCF
PWGDQ
PWGEM
PWGHF
PWGJE
PWGLF
PWGMM
AO2D.root
AnalysisResults
CMakeLists.txt
README.md
dpl-config.json
myConfig.json
run.sh
PWGUD
Skimming
include
src
CMakeLists.txt
cmake
dependencies
packaging
CMakeLists.txt
CODEOWNERS
COPYING
CPPLINT.cfg
README.md
pyproject.toml
alidist
sw
andy
avgPt
ayuda
build
chromium-widevine
cms
crmc-master
davidusb
Command
Command (local):
Filter:
All Files (*.*)
yaxis
114,609
x=-1.60545, y=11651.6



Hands-on session II

ALICE

Run 3 Pb-Pb

Where we stopped last time:

```
-// Histogram registry: an object to hold your histograms
HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};
Configurable<int> nBinsPt{"nBinsPt", 100, "N bins in pT histo"};

void init(InitContext const&)
{
    // define axes you want to use
    const AxisSpec axisCounter{1, 0, 1, "events"};
    const AxisSpec axisEta{30, -1.5, +1.5, "#eta"};
    const AxisSpec axisPt{nBinsPt, 0, 10, "p_{T} (GeV/c)"};
    // create histograms
    histos.add("hEventCounter", "hEventCounter", kTH1F, {axisCounter});
    histos.add("etaHistogram", "etaHistogram", kTH1F, {axisEta});
    histos.add("ptHistogram", "ptHistogram", kTH1F, {axisPt});
}
void process(aod::Collision const& collision, soa::Join<aod::Tracks, aod::Tracks
Extra, aod::TracksDCA> const& tracks)
{
    histos.fill(HIST("hEventCounter"), 0.5);
    for (auto& track : tracks) {
        if( track.tpcNClsCrossedRows() < 70 ) continue;
        if( fabs(track.dcaXY()) > 0.2) continue;
        histos.fill(HIST("etaHistogram"), track.eta());
        histos.fill(HIST("ptHistogram"), track.pt());
    }
}
```

erv

- A lot of CPU spent in preparing tracks, calculating DCAs …

For this part of the tutorial we will now use derived data:

– filtered: only eta, phi and pT of tracks above 4 GeV/c
– This isn't a standard, complete AO2D anymore!
– Derived data subscription advantage: no supporting task needed anymore!

In general: dramatic reduction in CPU time

THE DATA:

– Small file (~5MB): dropbox
– Medium file (~115MB): dropbox
– Large file (~367MB, equivalent to 6x107 events): dropbox

Tutorials/Skimming/derivedBasicConsumer.cxx :

```
void init(InitContext const&)
{
// define axes you want to use
const AxisSpec axisCounter{1, 0, +1, ""};
histos.add("eventCounter", "eventCounter", kTH1F, {axisCounter});
}
void process(aod::DrCollision const& collision)
{
histos.fill(HIST("eventCounter"), 0.5);
  }
```

Bulk operations via filtering!

• Now, let's define a filter to select on collision Z position!

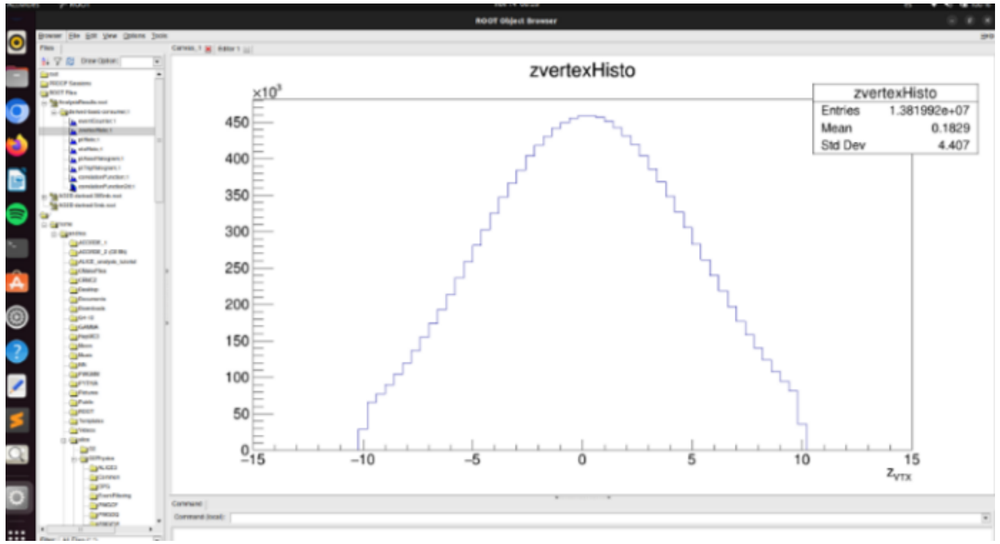
```
Filter collZfilter = nabs(aod::collision::posZ) < 10.0f;
```

• This needs to be declared inside the task struct (but outside process)
• Then you need to change your subscription to reflect the filtering:

```
void process(soa::Filteredaod::DrCollisions::iterator const& collision, aod::DrT
racks const& tracks)
```

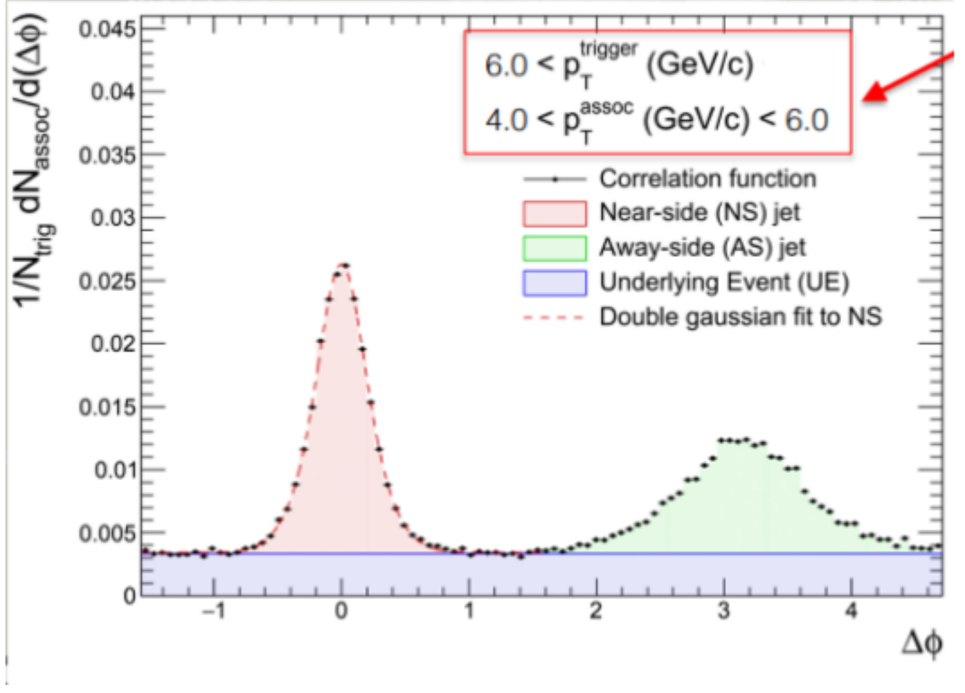
• And that's it!. The table handling will do the operations in bulk for selecting collisions.
• you can also create a histogram with vertex Z positions to be sure this is working as intended!

```
// Warning: the Filter declaration requires a new header! Add this:
#include "Framework/ASoAHelpers.h"
// You should also add this:
using namespace o2::framework::expressions;
```



Towards correlation functions: the logic

- Divide into “trigger” and “associated”
  – Trigger: the “Important” particles (high pT)
  • This example: let’s use above 6 GeV/c
  –Associated: check if other particles are related to the trigger in phase space
  – This example: 4.0 < pT < 6.0 GeV/c
  –A good task for partitioning!
- Calculate two-particle correlation function
  – Loop over triggers + loop over associated
  – Fill correlation function in nested for loop
  –Use something better than nested for loops?



Bulk operations via filtering and partitioning!

• You’ll need to add a SliceCache to your task struct by doing:

```
SliceCache cache;
```

Now, still within the task struct, let’s define two partitions: positive and negative eta!

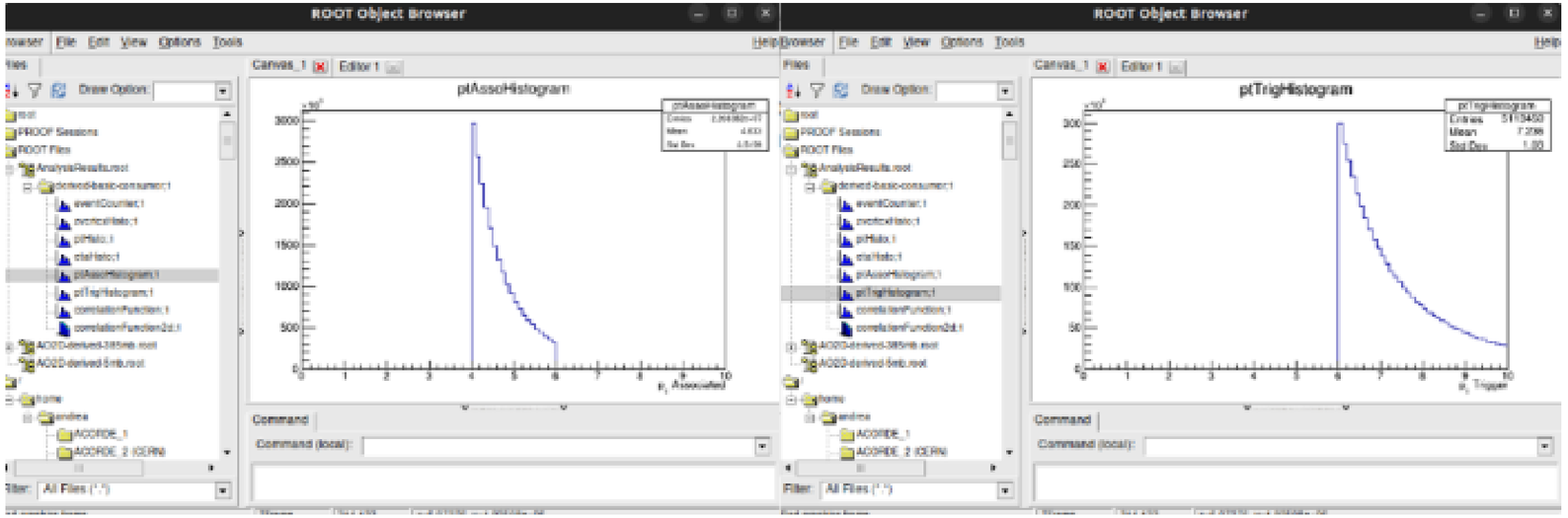
```
Partitionaod::DrTracks associatedTracks = aod::exampleTrackSpace::pt < 6.0f && a
od::exampleTrackSpace::pt > 4.0f;
Partitionaod::DrTracks triggerTracks = aod::exampleTrackSpace::pt > 6.0f;
```

• IMPORTANT: partitions are not grouped by default. You have to group them inside your process function:

```
//partitions are not grouped by default!
auto assoTracksThisCollision = associatedTracks->sliceByCached(aod::exampleTrack
Space::drCollisionId, collision.globalIndex(), cache);
auto trigTracksThisCollision = triggerTracks->sliceByCached(aod::exampleTrackSpa
ce::drCollisionId, collision.globalIndex(), cache);
```

• Now, still within the task struct, let’s define two partitions: positive and negative eta!
SliceCache cache;
• Now you can use these partitions to do any loop you like. For instance, we can fill two extra QA histograms:

```
for (auto& track : assoTracksThisCollision)
histos.fill(HIST("ptAssoHistogram"), track.pt());
for (auto& track : trigTracksThisCollision)
histos.fill(HIST("ptTrigHistogram"), track.pt());
```



Time to do correlations!

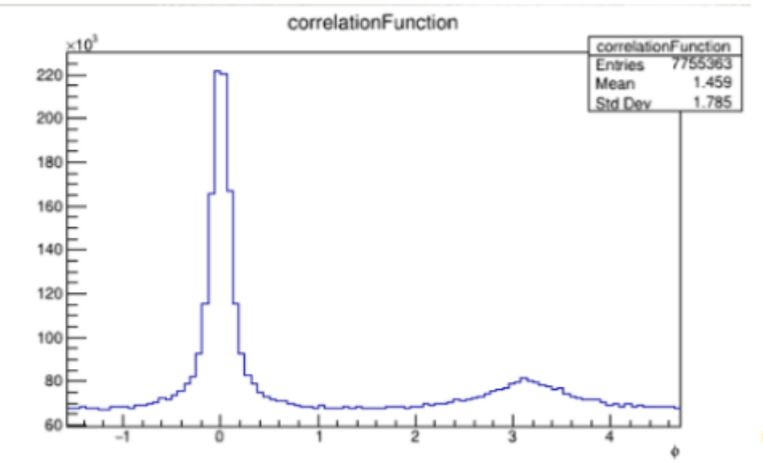
```
const AxisSpec axisDeltaPhi{100, -0.5*TMath::Pi(), +1.5*TMath::Pi(), "#phi"};
```

```
histos.add("correlationFunction", "correlationFunction", kTH1F, {axisDeltaPhi});
```

```
for (auto& trigger : trigTracksThisCollision){
for (auto& associated : assoTracksThisCollision){
histos.fill(HIST("correlationFunction"), ComputeDeltaPhi(trigger.phi(),associate
d.phi()));
}
}
```

- Note: ComputeDeltaPhi has been defined in derivedBasicConsumer!
  –No need to reinvent the wheel!
  • But… do we really need a nested loop? No!
  –Much faster to use framework funcionality!
  –More info on combinations: documentation

```
for (auto& [trigger, associated] :
combinations(o2::soa::CombinationsFullIndexPolicy(trigTracksThisCollision, assoT
racksThisCollision))) {
histos.fill(HIST("correlationFunction"), ComputeDeltaPhi(trigger.phi(),associate
d.phi()));
}
```

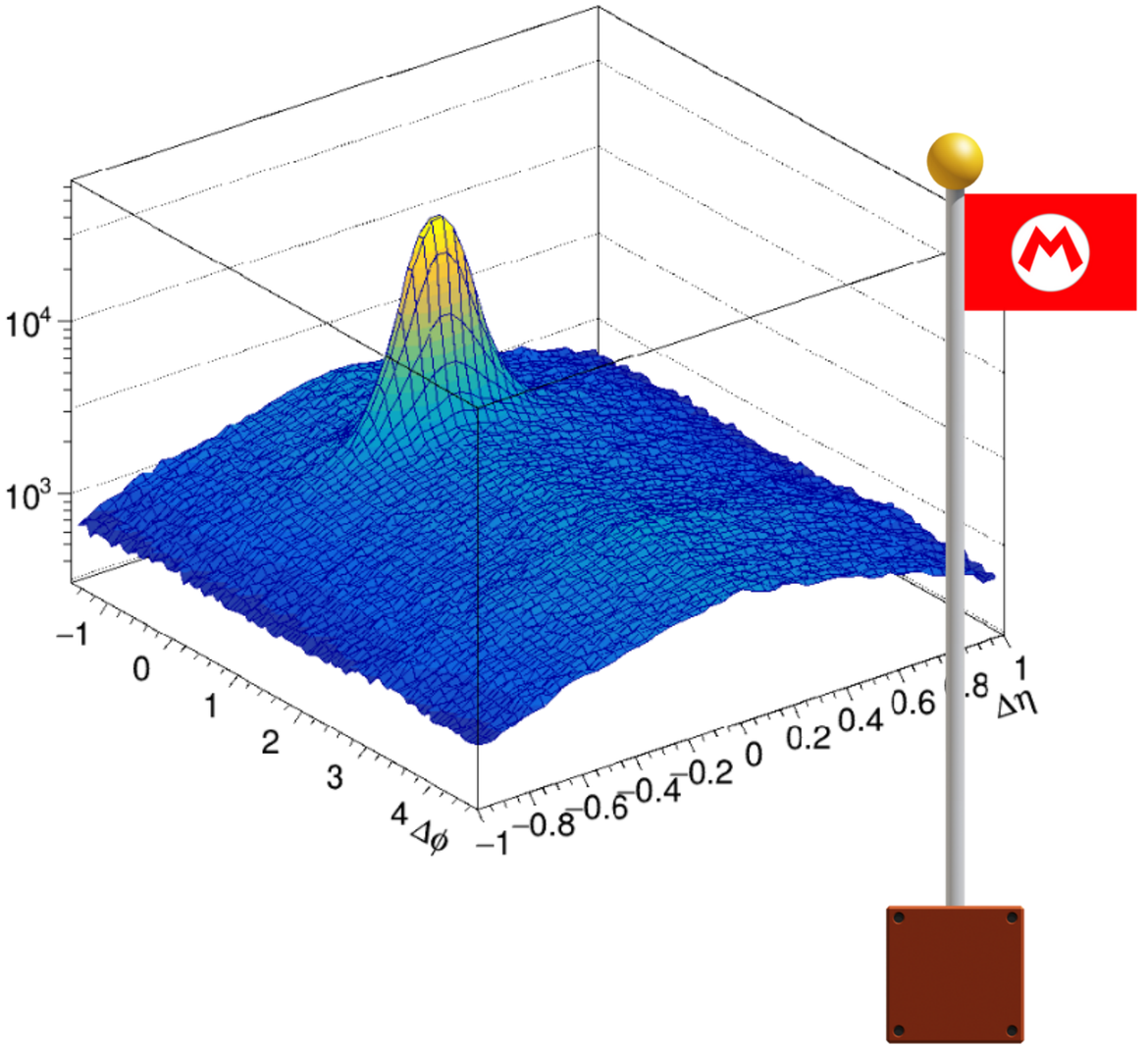


Bonus: two-dimensional correlations in $(\Delta\eta, \Delta\phi)$

```
const AxisSpec axisDeltaPhi{100, -0.5TMath::Pi(), +1.5TMath::Pi(), "#Delta#ph
i"};
const AxisSpec axisDeltaEta{100, -1.0, +1.0, "#Delta#eta"};
```

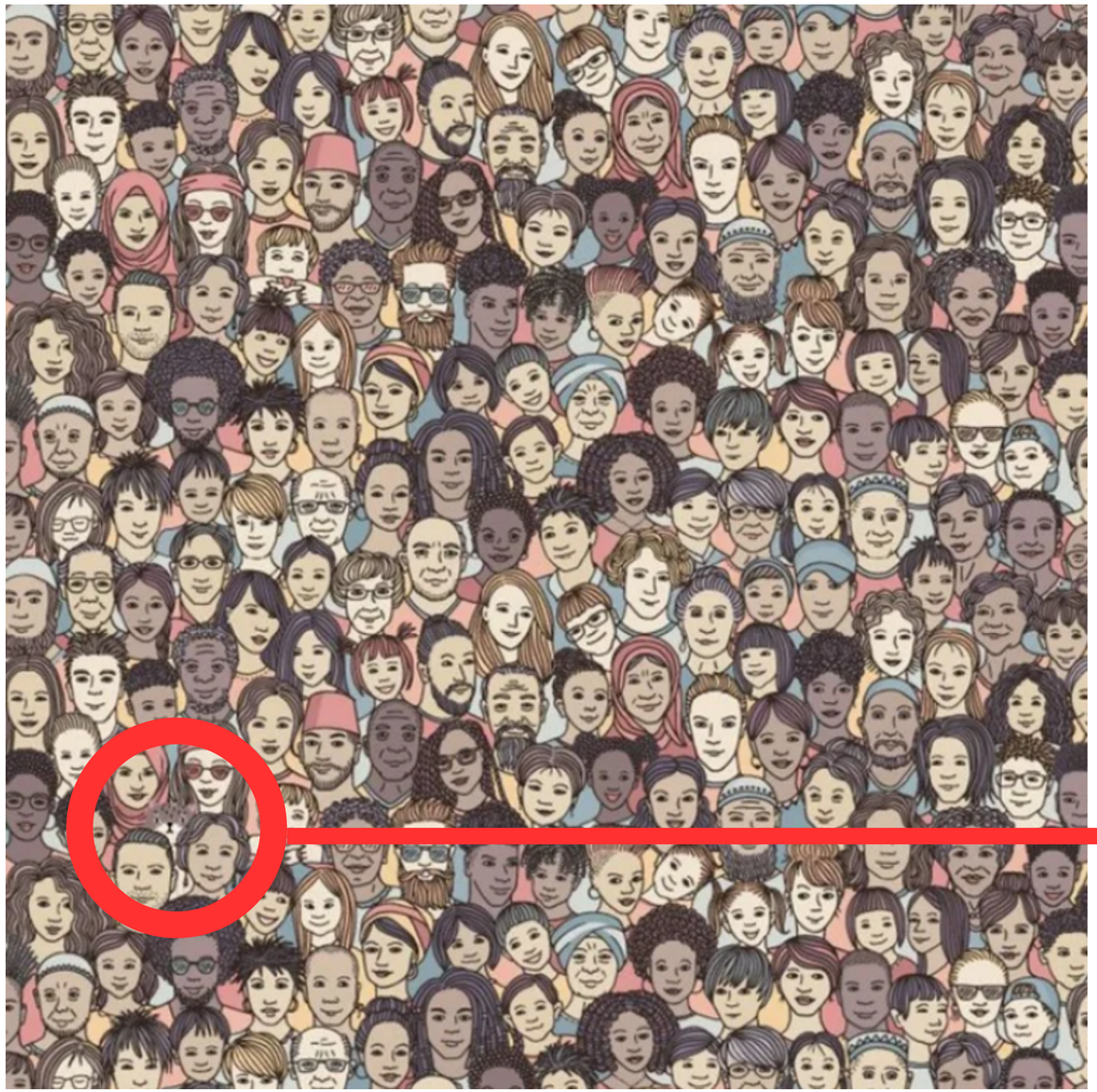
```
histos.add("correlationFunction", "correlationFunction", kTH1F, {axisDeltaPhi});
histos.add("correlationFunction2d", "correlationFunction2d", kTH2F, {axisDeltaPh
i, axisDeltaEta});
```

```
for (auto& [trigger, associated] :
combinations(o2::soa::CombinationsFullIndexPolicy(trigTracksThisCollision, assoT
racksThisCollision))) {
histos.fill(HIST("correlationFunction"), ComputeDeltaPhi(trigger.phi(),associate
d.phi()));
histos.fill(HIST("correlationFunction2d"), ComputeDeltaPhi(trigger.phi(),associa
ted.phi()),
trigger.eta() - associated.eta());
}
```



Particle identification

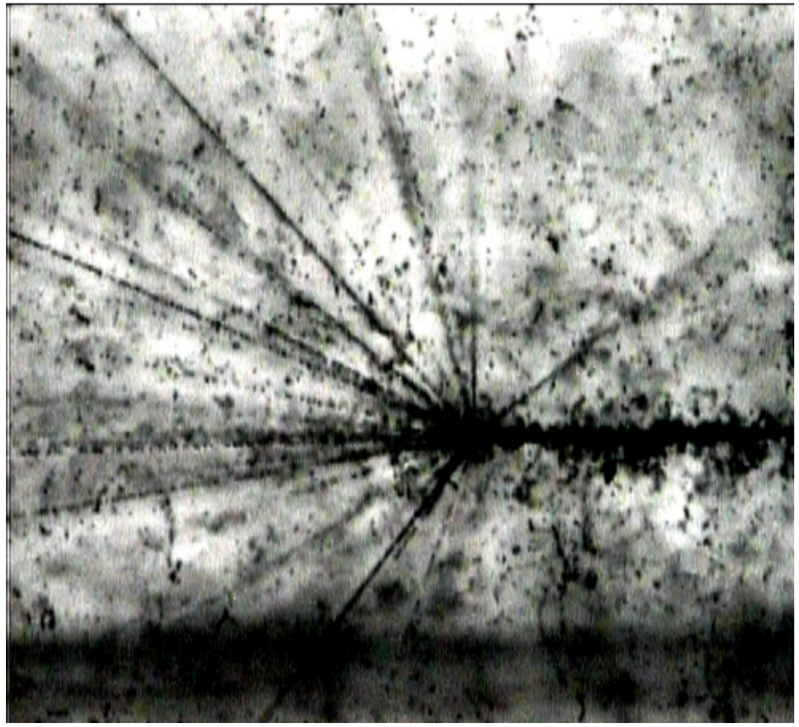
Can you find the cat?

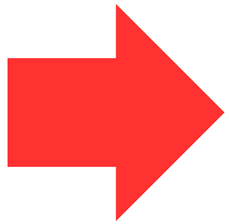


Particle identification

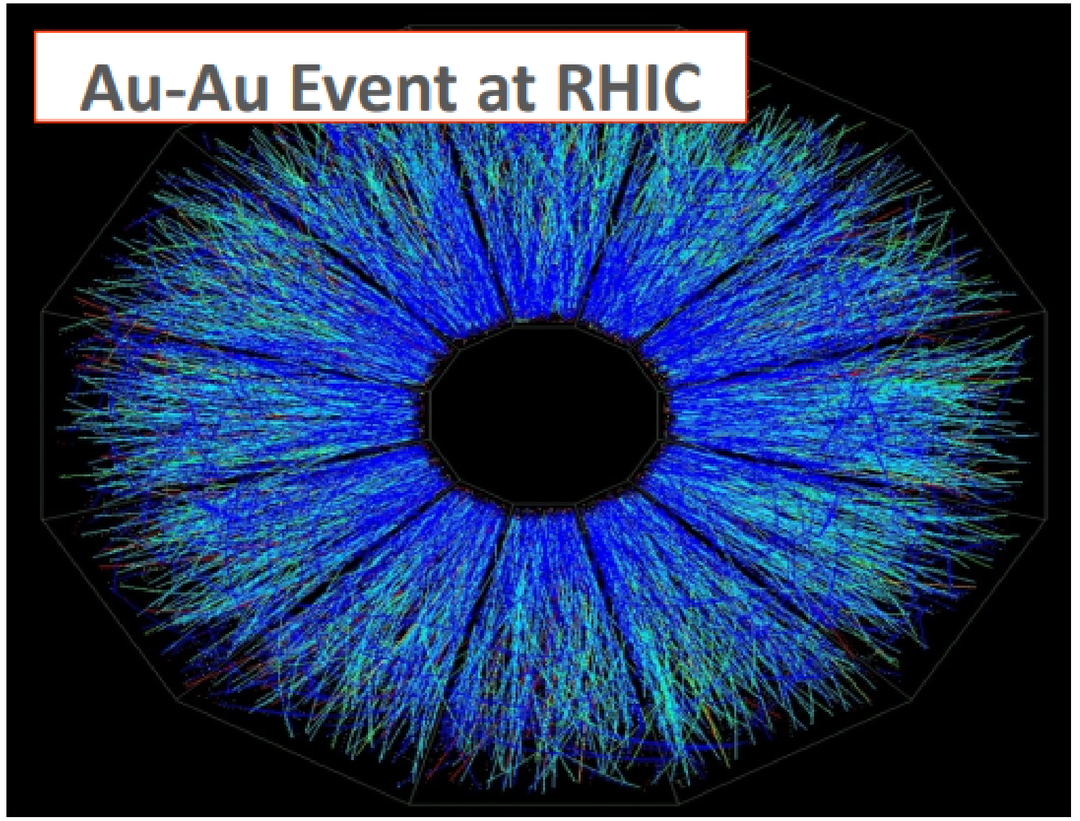
Can you find the cat?

Central Kr-Ag/Br event at 950 MeV/A



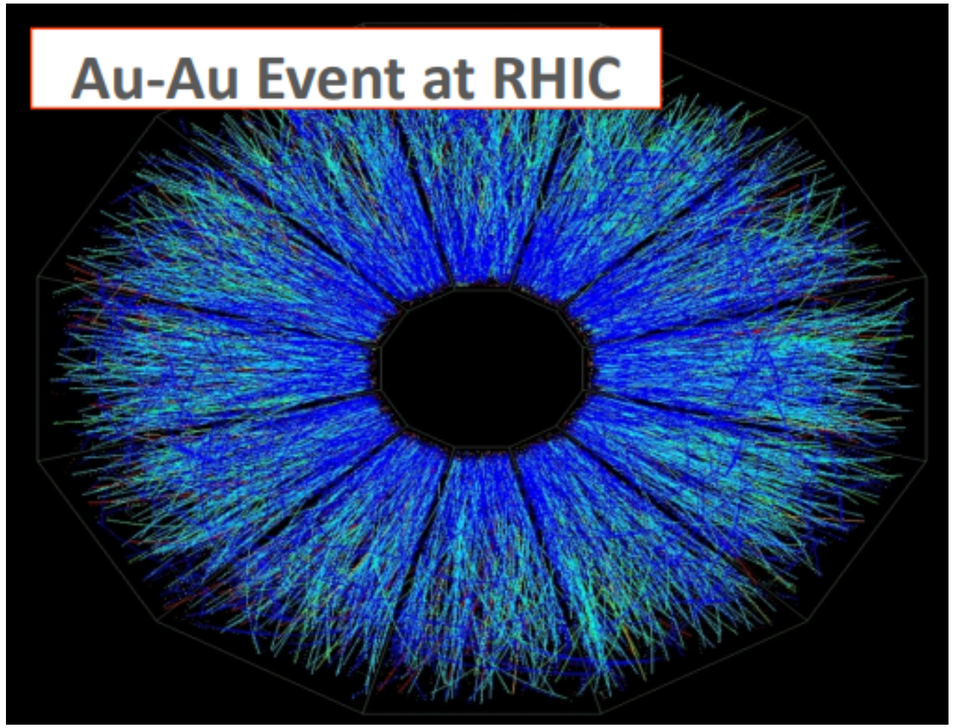


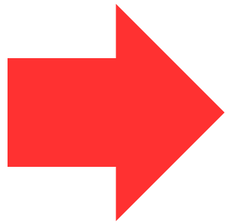
Au-Au Event at RHIC

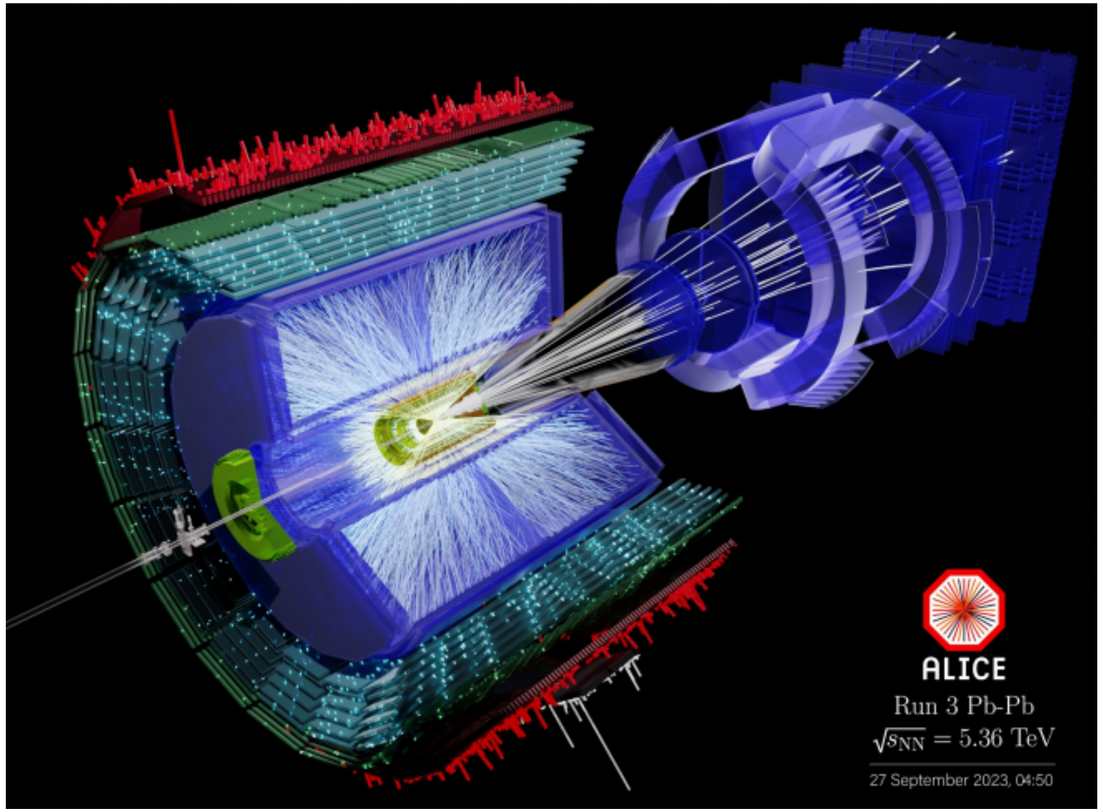


Few particles only

Thousands of particles!



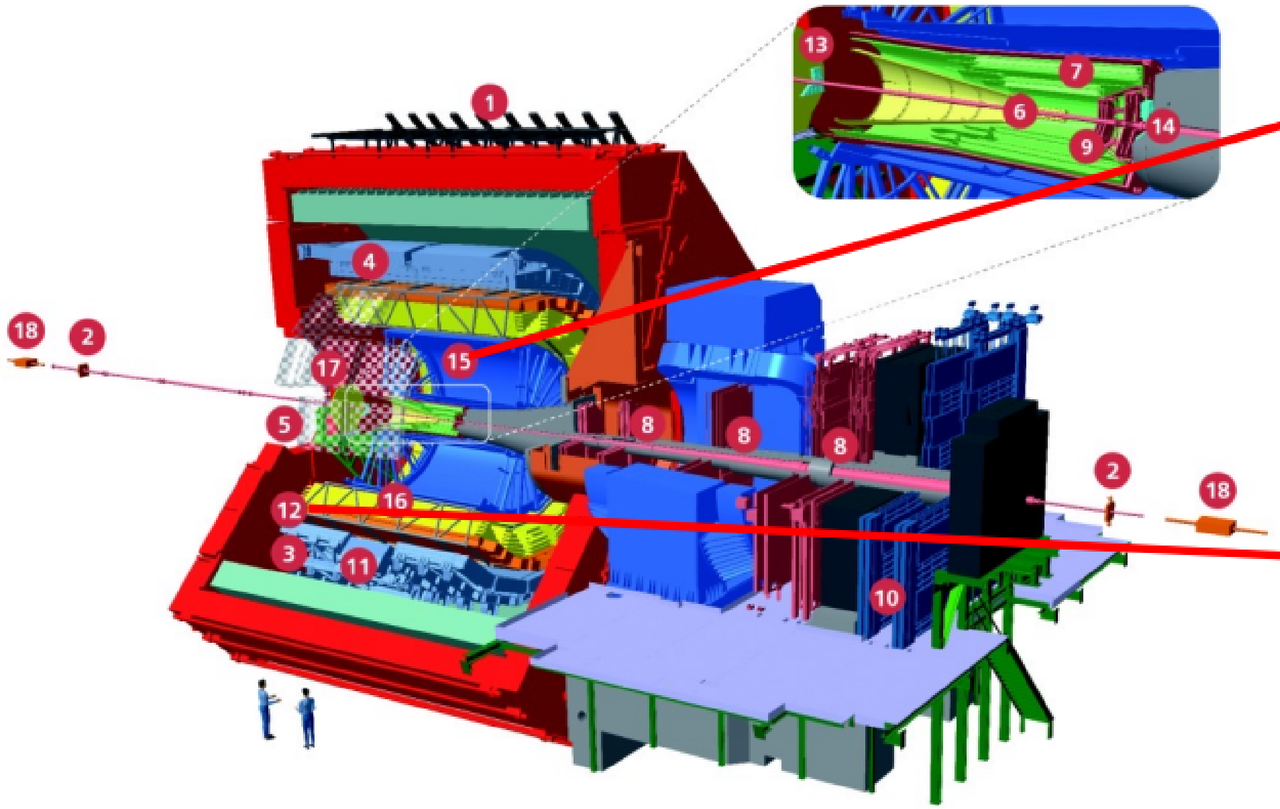




Thousands of particles!

Out of a particle crowd we need to identify **concerned particles** as the cat in the picture

Particle identification detectors



1. **ACORDE** | ALICE Cosmic Rays Detector
2. **AD** | ALICE Diffractive Detector
3. **DCal** | Di-jet Calorimeter
4. **EMCal** | Electromagnetic Calorimeter
5. **HMPID** | High Momentum Particle Identification Detector
6. **ITS-IB** | Inner Tracking System - Inner Barrel
7. **ITS-OB** | Inner Tracking System - Outer Barrel
8. **MCH** | Muon Tracking Chambers
9. **MFT** | Muon Forward Tracker
10. **MID** | Muon Identifier
11. **PHOS / CPV** | Photon Spectrometer
12. **TOF** | Time Of Flight
13. **T0-A** | Tzero + A
14. **T0+C** | Tzero + C
15. **TPC** | Time Projection Chamber
16. **TRD** | Transition Radiation Detector
17. **V0+** | Vzero + Detector
18. **ZDC** | Zero Degree Calorimeter

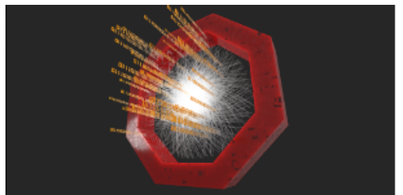
TPC

- Use energy loss value given by **Bethe-Bloch** for PID.

TOF

- Use particle **time-of-flight** from collision vertex to TOF for PID

Helper tasks for PID



ALICE O2 Analysis framework
DOCUMENTATION

Search docs...

DOCUMENTATION HOME
Getting started
Writing an analysis task
Running an analysis
1. Downloading AO2D samples
2. Setting up an execution
3. Using json configuration files
4. Helper tasks
Timestamp
Event selection
Multiplicity and centrality selection in O2
Particle identification (PID)
Track Selection
Track Propagation
5. Saving tables to a file
6. Reading tables from files
7. Options for running analyses
ALICE O2 documentation master

Particle identification (PID)

Table of contents:

Here are described the working principles of Particle Identification (PID) in O2 and how to get PID information (expected values, nSigma separation *et cetera*) in your analysis tasks if you plan to identify particles.

Introduction

PID is handled in analysis by filling helper tables that can be joined to tracks (propagated or not). The parameterization of the expected detector response (e.g. signal, resolution, separation) is used in the PID tasks to fill the PID tables. These parameterizations are detector specific and handled by the detector experts; usually, they are shipped to the PID helper tasks from the CCDB to match the data-taking conditions. The interface between the detector and the Analysis Framework (i.e. your tracks) is fully enclosed in `PIDResponse.h`. Here are the defined tables for the PID information for all the detectors.

The filling of the PID tables is delegated to dedicated tasks in `Common/TableProducer/PID/` Examples of these tasks can be found in `pidTOF.cxx` and `pidTPC.cxx` for TOF and TPC tables, respectively.

Usage in user tasks

Tables for PID values in O2 are defined in `PIDResponse.h`. You can include it in your task with:

```
#include "Common/DataModel/PIDResponse.h"
...
```

In the process functions, you can join the table to add the PID (per particle mass hypothesis) information to the track. In this case, we are using the mass hypothesis of the electron, but tables for nine (9) stable particle species are produced (`El` , `Mu` , `Pi` , `Ka` , `Pr` , `De` , `Tr` , `He` , `Al`).

- For the **TOF** PID as:

```
void process(soa::Join<aod::Tracks, aod::pidTOFEl>::iterator const& track) {
  track.tofNSigmaEl();
}
```

- For the **TPC** PID as:

1. **Monte Carlo Simulation:**
   - Before conducting experiments in HEP, scientists often use Monte Carlo simulations to generate simulated data based on theoretical models.
   - Monte Carlo simulations take into account various factors such as the initial conditions of the experiment, the properties of the particles involved, and the response of the detectors.
   - The outcome of these simulations is a set of simulated events that mimic what might be observed in the real experiment.
2. **Reconstruction Process:**
   - After conducting an actual experiment, researchers obtain raw data from the detectors. This raw data needs to be processed and analyzed to reconstruct the properties of the particles produced in the experiment.
   - The reconstruction process involves calibrating the detectors, correcting for experimental conditions, and applying algorithms to interpret the recorded signals in terms of particle properties.
   - The goal is to extract information such as the types of particles, their momenta, trajectories, and interaction points.

3. **Checking If Monte Carlo Particles Have Been Reconstructed:**
   - When researchers say they need to check if Monte Carlo particles have been reconstructed, they are referring to comparing the results of the Monte Carlo simulations with the actual reconstructed data.
   - This comparison helps validate the performance of the reconstruction algorithms. It ensures that the reconstruction process accurately reflects the expected outcomes based on theoretical models.
   - Researchers may check whether the reconstructed data matches the simulated data in terms of particle identification, energy deposition, and other relevant properties.
4. **Validation and Optimization:**
   - The agreement between the Monte Carlo simulations and the reconstructed data is crucial for validating the experimental setup, the detector response model, and the analysis methods.
   - If there are discrepancies, researchers may need to revisit and optimize their reconstruction algorithms, calibration procedures, or other aspects of the analysis to better align with the expected outcomes from simulations.

In summary, checking if Monte Carlo particles have been reconstructed involves a validation step to ensure that the experimental data aligns with the simulated expectations. This process helps researchers refine their analysis techniques and gain confidence in the accuracy of their results.

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
```

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

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

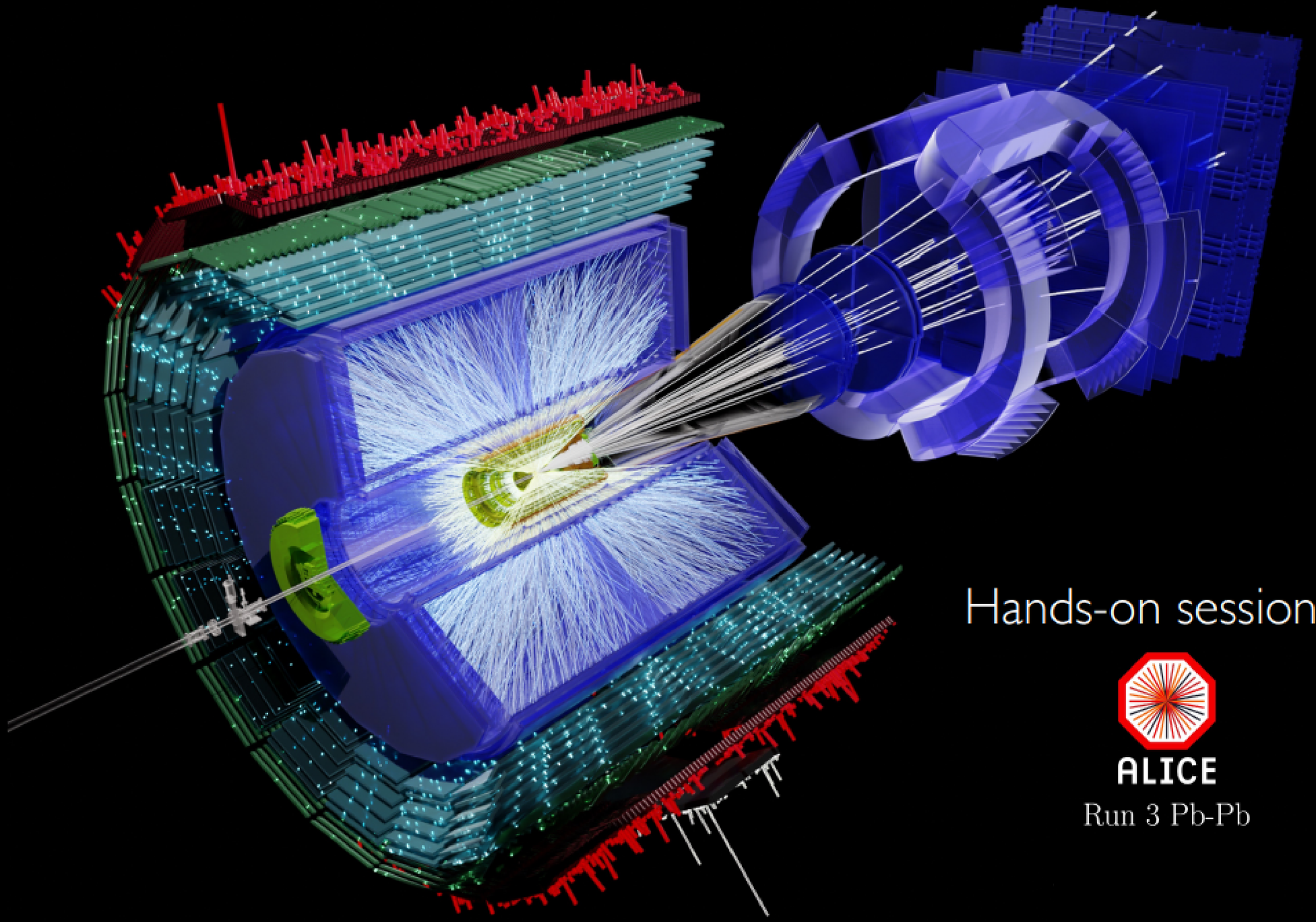
- Pythia8 is more deeply integrated in O2 (compared to others) and it is recommended to use Pythia8 whenever possible
- When Pythia8 is used, it can be fully configured via a special text file and the GeneratorPythia8 parameter
  - valid settings can be found in the Pythia8 reference manual
- we also 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”
```



Hands-on session III

ALICE

Run 3 Pb-Pb

notion: hands-on III

Hands-on session III

Data for this exercise:

• Single AO2D file from 2023 MC anchored to pp LHC22o pass4 (LHC23f4b2)

- CERNBox

- Dropbox

The starting point: a simple task to do a collision loop

Starting point: code similar to the one used yesterday!
• The good old:
myExampleTask.cxx + Changes (dropbox)

```
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"
#include "Common/DataModel/TrackSelectionTables.h"
#include "Framework/ASoAHelpers.h"
```

```
using namespace o2;
using namespace o2::framework;
using namespace o2::framework::expressions;

struct myExampleTask {
  // Histogram registry: an object to hold your histograms
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObjec
t};

  Configurable<int> nBinsPt{"nBinsPt", 100, "N bins in pT histo"};

  Filter trackDCA = nabs(aod::track::dcaXY) < 0.2f;

  //This is an example of a convenient declaration of "using"
  using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksD
CA>;
  using myFilteredTracks = soa::Filtered<myCompleteTracks>;

  void init(InitContext const&)
  {
    // define axes you want to use
    const AxisSpec axisCounter{1, 0, +1, ""};
    const AxisSpec axisEta{30, -1.5, +1.5, "#eta"};
    const AxisSpec axisPt{nBinsPt, 0, 10, "p_{T}"};
    // create histograms
    histos.add("eventCounter", "eventCounter", kTH1F, {axisCounter});
    histos.add("etaHistogram", "etaHistogram", kTH1F, {axisEta});
    histos.add("ptHistogram", "ptHistogram", kTH1F, {axisPt});
  }
  void process(o2::aod::Collision const& collision, myFilteredTracks const& trac
ks)
  {
    histos.fill(HIST("eventCounter"), 0.5);
    for (const auto& track : tracks) {
      if( track.tpcNClsCrossedRows() < 70 ) continue; //badly tracked
      histos.fill(HIST("etaHistogram"), track.eta());
      histos.fill(HIST("ptHistogram"), track.pt());
  }
  }
};
```

Accessing the MC information for each track

• You will now need to de-reference the MC particle for each and every track.

• This is done via a label: that is an index column pointing to the McParticles table.

• This index column is the sole element in the McTrackLabel table – joinable with tracks!

```
//This is an example of a convenient declaration of "using"
using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDC
A, aod::McTrackLabels>;
```

• Now you have a track label that you can de-reference!

• …but to de-reference it, you also need to subscribe to the McParticles table (or you won't have those!)

```
void process(aod::Collision const& collision, myFilteredTracks const& tracks, ao
d::McParticles const&)
```

• Let's also create a histogram to store the momentum resolution – this time, a two-dimensional one – in **Init():**

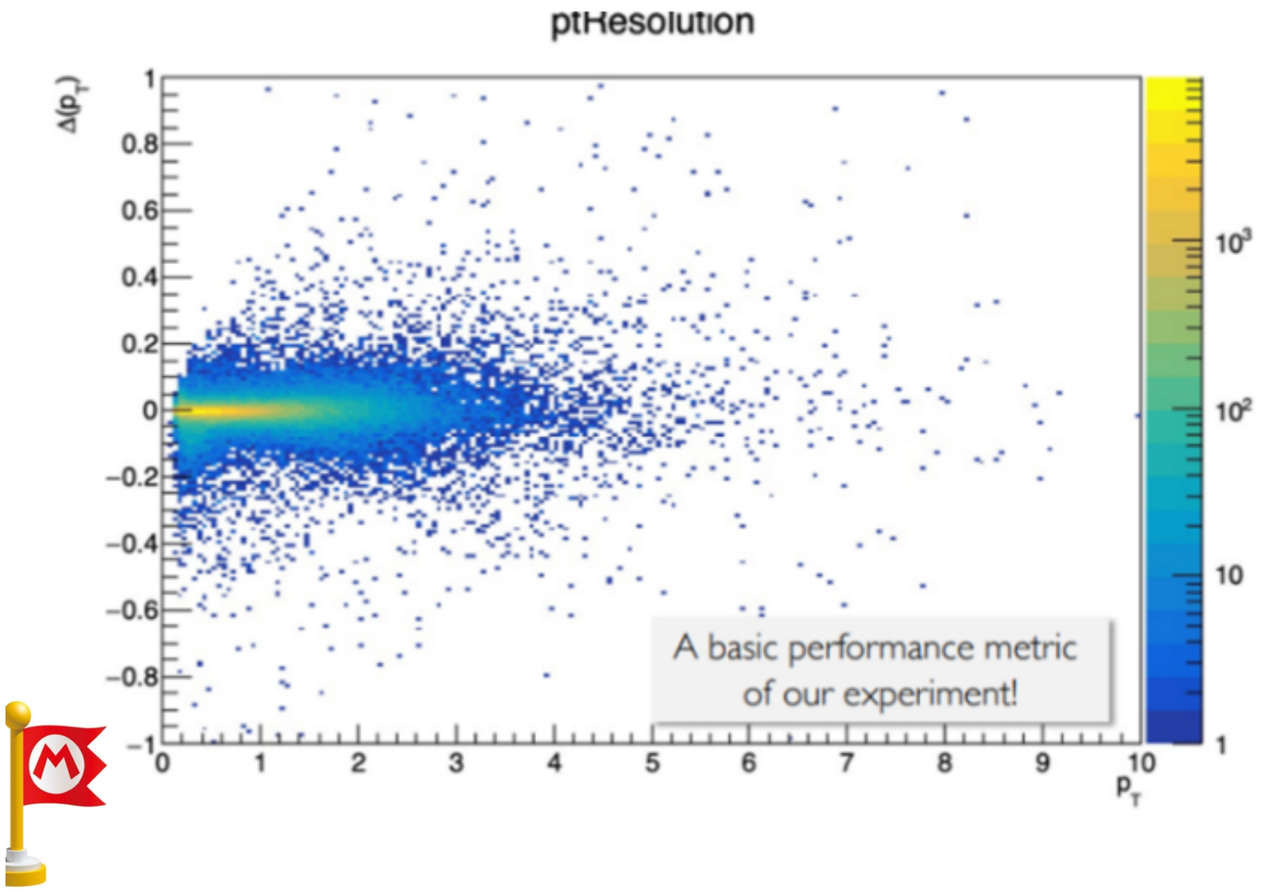
```
const AxisSpec axisDeltaPt{100, -1.0, +1.0, "#Delta(p_{T})"};
(…)

histos.add("ptResolution", "ptResolution", kTH2F, {axisPt, axisDeltaPt});
```

• Now comes the magic: the immediate de-referencing!

```
if(track.has_mcParticle()){
  auto mcParticle = track.mcParticle();
  histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
}
```

For running, you can still use the same run.sh you used yesterday throughout this session!
…but you need an o2-analysis-bc-converter added (dropbox)

The outcome: momentum resolution as a function of transverse momentum



Getting the reconstructed pion, kaon and proton spectra

- Yes! This will eventually turn into an efficiency…
- Let's declare three more histograms: a pion, a kaon and a proton pT histogram!

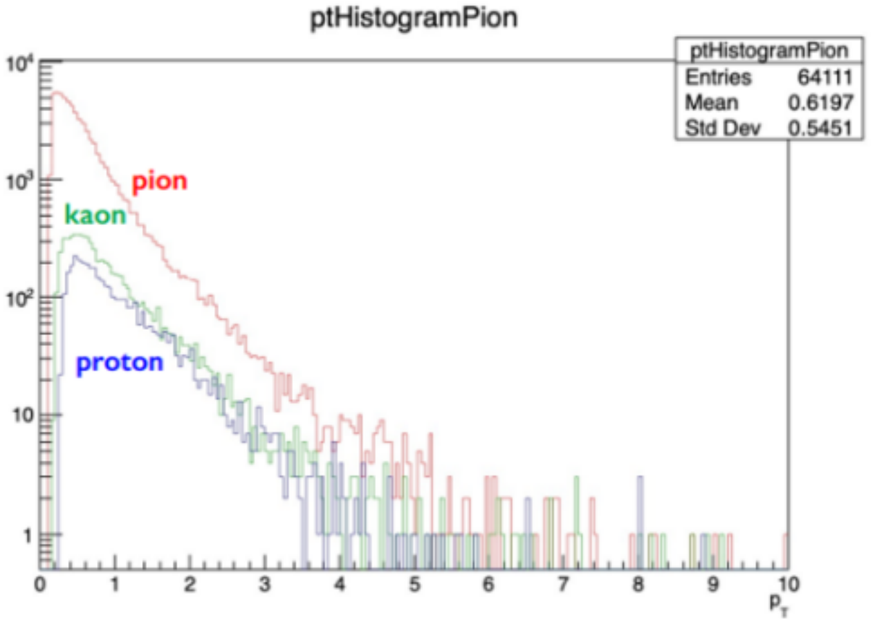
```
histos.add("ptHistogramPion", "ptHistogramPion", kTH1F, {axisPt});
histos.add("ptHistogramKaon", "ptHistogramKaon", kTH1F, {axisPt});
histos.add("ptHistogramProton", "ptHistogramProton", kTH1F, {axisPt});
```

- How can we find those pions, kaons and protons? Can we restrict to midrapidity?
- And more: how can we know they actually come from a primary collision and not from material?
- Hint: check the documentation again! This time, for McParticles:

Name		Getter	Type	Comment
o2::soa::Index	GI	globalIndex	int64_t	
o2::aod::mcparticle::Y	E	y	float	Particle rapidity, conditionally defined to avoid FPEs
o2::aod::mcparticle::PdgCode		pdgCode	int	PDG code
o2::aod::mcparticle::IsPhysicalPrimary	D	isPhysicalPrimary	bool	True if particle is considered a physical primary according to the ALICE definition

```
if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){ // do this in th
e context of the track ! (context matters!!!)
  if(abs(mcParticle.pdgCode())==211) histos.fill(HIST("ptHistogramPion"), mcPart
icle.pt());
  if(abs(mcParticle.pdgCode())==321) histos.fill(HIST("ptHistogramKaon"), mcPart
icle.pt());
  if(abs(mcParticle.pdgCode())==2212) histos.fill(HIST("ptHistogramProton"), mcP
article.pt());

}
```

The outcome: three histograms with reconstructed pion/kaon/proton spectra



Processing pure generation information

• Let's say you want to get the generated particle spectra
• It is much more convenient to do it inside a separate process function: the subscription will be very different!
• Let's now resort to process switches within the same task: you saw it in the lecture, but practice is nicer!
• Before we do that, let's add generated pT histograms to the init function:

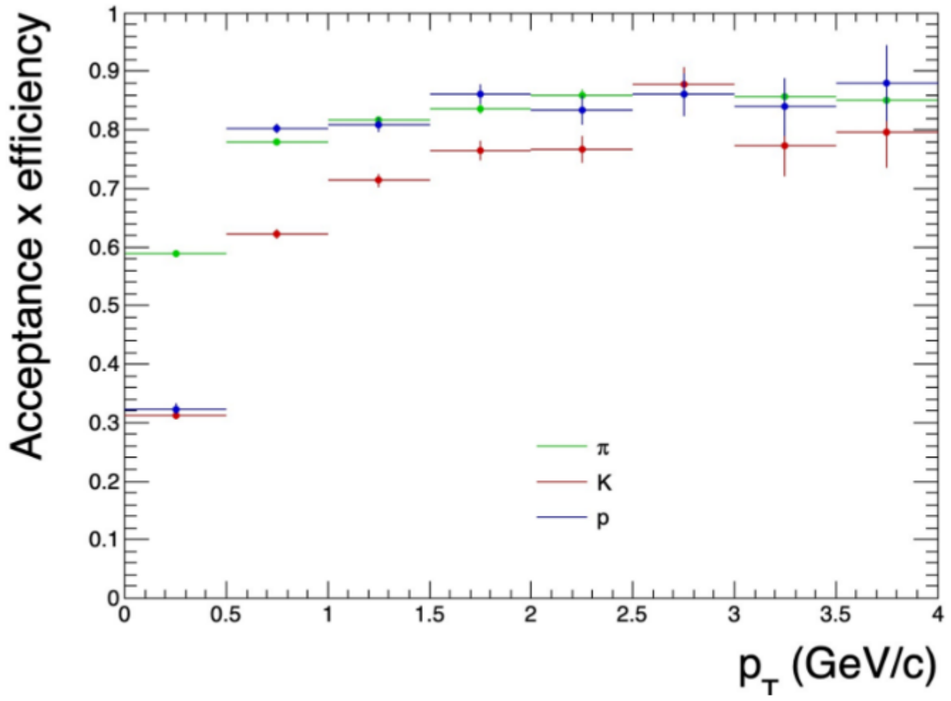
```
histos.add("ptGeneratedPion", "ptGeneratedPion", kTH1F, {axisPt});
histos.add("ptGeneratedKaon", "ptGeneratedKaon", kTH1F, {axisPt});
histos.add("ptGeneratedProton", "ptGeneratedProton", kTH1F, {axisPt});
```

• Now let's add another process function!

```
void processReco(aod::Collision const& collision, myFilteredTracks const& track
s, aod::McParticles const&)
{
  (…)
}
PROCESS_SWITCH(myExampleTask, processReco, "process reconstructed information",
true);
void processSim(aod::McParticles const& mcParticles)
{
  for (const auto& mcParticle : mcParticles) {
    if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){ // watch out
for context!!!
      if(abs(mcParticle.pdgCode())==211) histos.fill(HIST("ptGeneratedPion"), mc
Particle.pt());
      if(abs(mcParticle.pdgCode())==321) histos.fill(HIST("ptGeneratedKaon"), mc
Particle.pt());
      if(abs(mcParticle.pdgCode())==2212) histos.fill(HIST("ptGeneratedProton"),
mcParticle.pt());
    }
  }
}
PROCESS_SWITCH(myExampleTask, processSim, "process pure simulation information",
true);
```

The outcome: efficiencies of pion, kaon and proton

disclaimer: this is a far cry from something real! In practice, you also need PID (as you saw in the morning…)
disclaimer 2: this also specifically makes no distinction about particles being associated to wrong collisions, etc etc



For this, you'll need a macro that divides the reconstructed and generated histograms with ROOT function **Divide** :

```
hRecoPion->Divide(hRecoPion, hGenPion, 1.0, 1.0, "B");
hRecoKaon->Divide(hRecoKaon, hGenKaon, 1.0, 1.0, "B");
hRecoProton->Divide(hRecoProton, hGenProton, 1.0, 1.0, "B");
```

I provide the macro later in the main page.

Moving flexibly around from MC to reconstruction: going beyond!

- What about the number of times a MC collision was reconstructed?
- What about an analysis starting from the MC side and “extending” to the reconstruction?
- This can be done with sliceBy, but it can also be done with a very careful subscription!

```
void processSim(aod::McCollision const& mcCollision, soa::SmallGroups<soa::Join<
aod::McCollisionLabels,
aod::Collisions>> const& collisions, aod::McParticles const& mcParticles, myFilt
eredTracks const& tracks)
```

- A soa::SmallGroups subscription essentially does grouping for you based on the collisionId column!

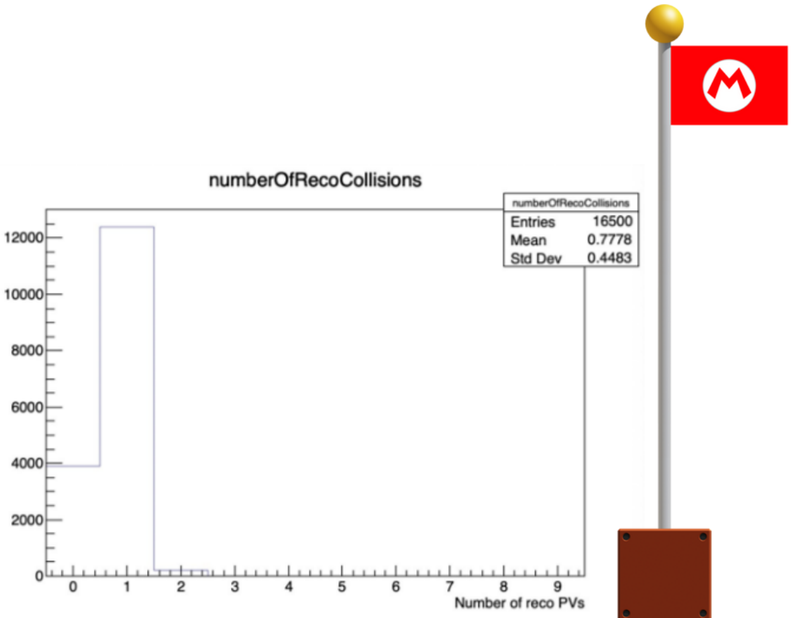
```
histos.fill(HIST("numberOfRecoCollisions"), collisions.size()); // number of tim
es coll was reco-ed
```

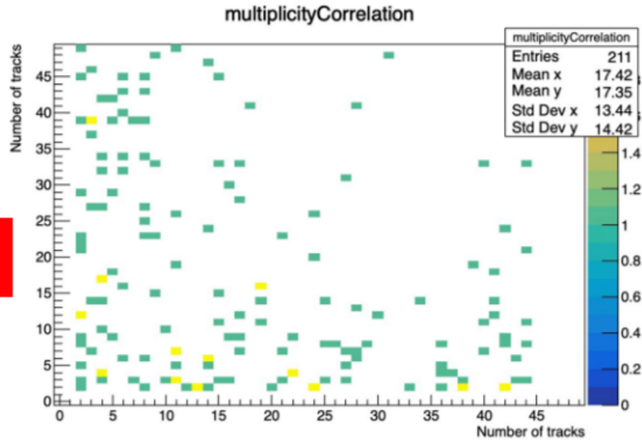
- Then one can immediately find out the number of times a MC collision appears by just checking the size of collisions!
- …and we can go beyond! Can we find out how two reco-ed collisions of the same MC collision look like?

```
Preslice<aod::Tracks> perCollision = aod::track::collisionId; // add this to you
r struct (outside process!)
```

```
//inside processSim: now loop over each time this collision has been reconstruct
ed and aggregate tracks
std::vector<int> numberOfTracks;
for (auto& collision : collisions) {
auto groupedTracks = tracks.sliceBy(perCollision, collision.globalIndex());
// size of grouped tracks may help in understanding why event was split!
numberOfTracks.emplace_back(groupedTracks.size());
}
if( collisions.size() == 2 ) histos.fill(HIST("multiplicityCorrelation"), number
OfTracks[0], numberOfTracks[1]);
```

Final outcome: number of times a MC collision was reconstructed + properties





Options for running analyses

Local execution

• Run on **local** data (i.e. some input files that are stored on your laptop)

> **Please note**
>
> For practical purposes it is (nearly) impossible to process a whole dataset locally on your laptop, but launching a local test provides you with an environment to test your analysis task!

To run a local analysis, we need to

- Compile our code, and
- Launch our analysis

Running macros:

To run your analysis, it will now suffice to type in a terminal (after sourcing the O2Physics environment):

```
root nameofyourmacro.C
```

You can try by running this macro in your Hands-on III directory:

macro for hands-on III

Grid execution (standalone)

• Run on **GRID** (either in test or full mode) yourself

The Worldwide LHC Computing Grid, usually simply referred to as Grid, is a global collaboration of computer centers. It has been up and running since 2002, to provide resources to store, distribute and analyze the petabytes of data that are generated by the LHC.

On the picture below, from http://alimonitor.cern.ch/map.jsp , you can see the distribution of computing centers that are working for the ALICE collaboration. We will the Grid to run your analysis on a larger scale than your laptop can provide.

Hyperloop system

• The Hyperloop train system is used to submit your analysis in a train to the Grid.

Previous to Hyperloop: Lego train system (RUN2)

By now, your code is running well on Grid, you're submitting and merging jobs like a seasoned professional. However, do you really want to

- Stay up all night to resubmit jobs?
- Risk losing your code because it's not part of AliPhysics and your hard drive crashes?

To avoid these headaches, users are encouraged to run their jobs in a more automated and efficient way by using the LEGO framework

ℹ Why LEGO trains are crucial

A train can easily contain multiple instances of your analysis, meaning you can run your default configuration and in addition multiple cut variations and other cross-checks. The success rate of the jobs is EXACTLY the same for each of your configurations, as they are all handled in parallel. This means that the all the configurations handle the exact same set of data.

This is in contrast to submitting your jobs one after each other (possibly running into quota problems) and having random job crashes, resulting in a situation where your default configuration and the variations did not handle exactly the same set of data.

If you already know the LEGO system just see what has changed : *Run 2 LEGO train expert*

Hyperloop Preconditions

- Login to Mattermost and (if not already done) join the restricted team ALICE (top menu, select join restricted teams). Then join the support channels on Mattermost: O2 Analysis and O2 Hyperloop Operation.

- Make sure that you have a valid AliEn certificate installed in your browser. If you can access https://alimonitor.cern.ch/hyperloop/ without getting "access denied", then everything is fine.
- Check if your AliEn username corresponds to the CERN login. The latter is for example the username you use for logging into the mail server or into lxplus.cern.ch. You can check your AliEn username by going to the file catalog browser here. Check the folder path or the string "Welcome ".

Concept

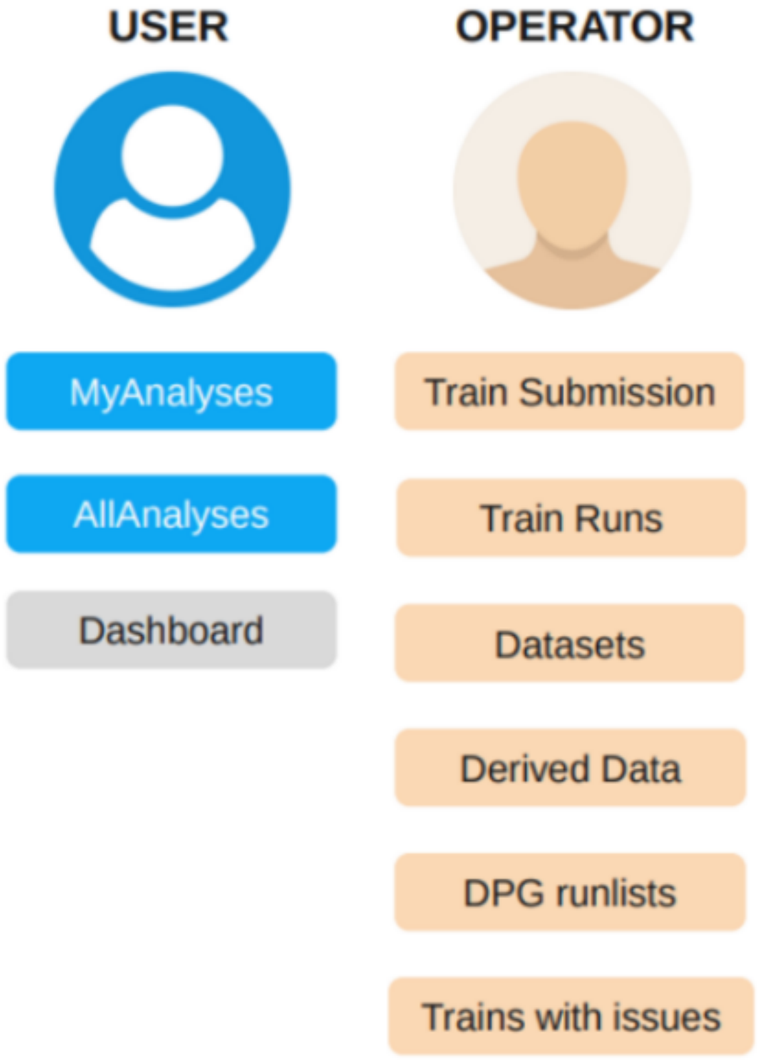
The Hyperloop framework is a tool to run and manage analysis trains on AliEn. It builds on the O2 analysis framework, O2 DPL, MonALISA and LPM.

In order to run trains on the Grid, the code has to be contained in an AliEn package. Therefore, the train uses the regularly deployed O2Physics tags. Hyperloop supports Run 3 data and converted Run 2 data.

Hyperloop provides a web interface for users and operators which allows to:

- register train wagons
- configure trains (wagons, input datasets)
- test the wagons and the train in a well-defined environment
- study the test results
- submit the train to the Grid
- study the resource consumption of the train for each wagon

So yes, we have the user and the operator:

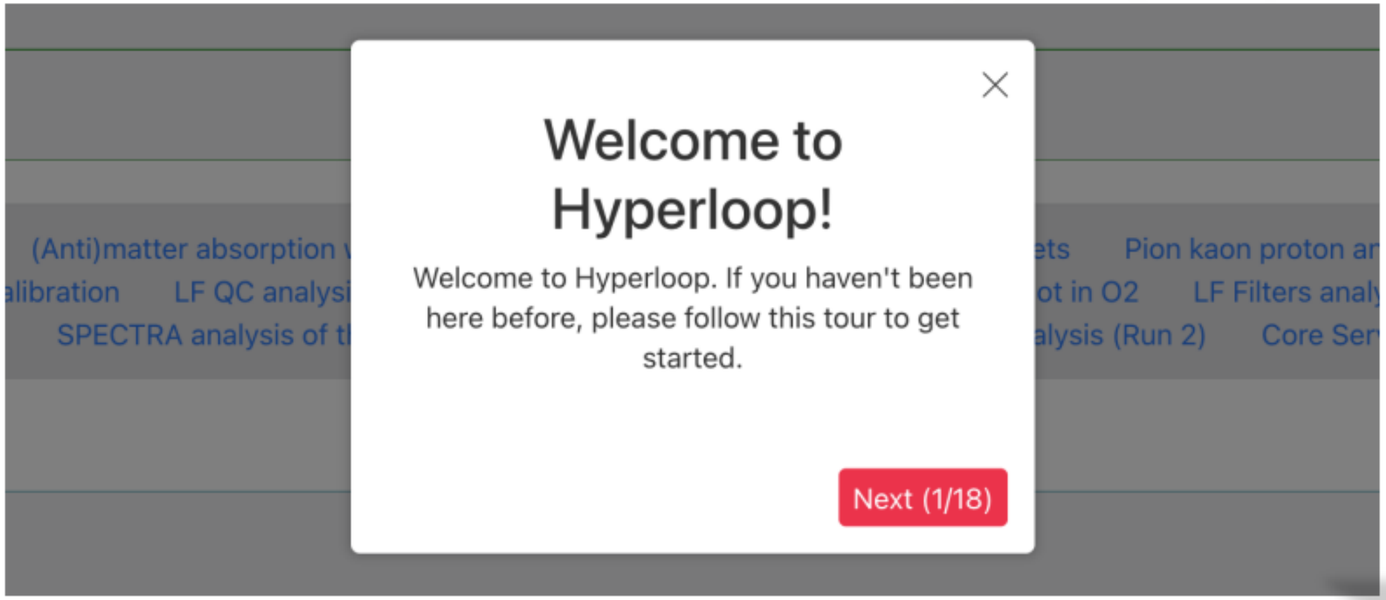


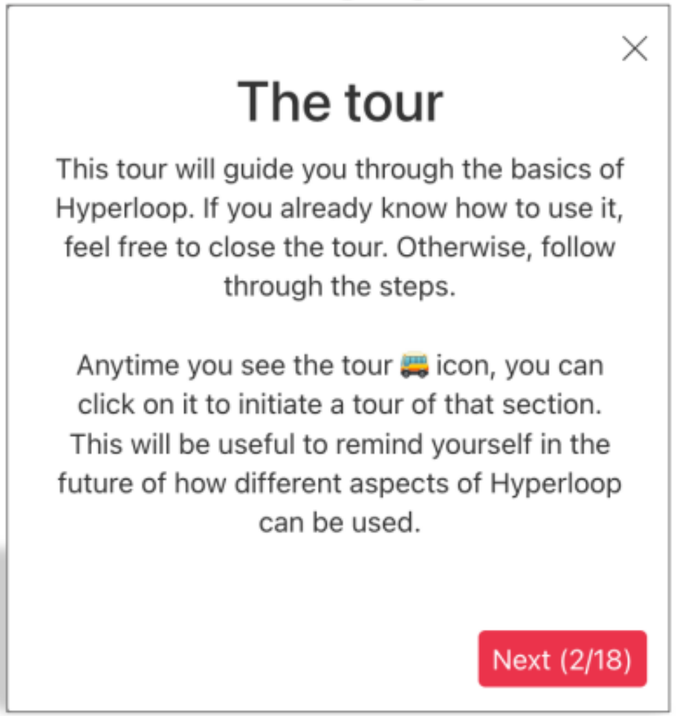
- My Analyses: Personalized webpage which displays all the analyses where the user belongs to.
- All Analyses: Displays all the analyses available in the system.
- Dashboard: Presents an overview of the current state of the system, as well as a summary from the previous week.
- Train Submission: Used by operators to compose and submit trains.
- Train Runs: Displays all the train runs in the system.
- Datasets: Displays all the datasets available and allows the operator to create, configure and delete datasets.
- DPG Runlists: Displays all the DPG runlists available in the system. This page is used by DPG experts to create, configure and delete runlists.

❑ Train support

- 24/5 Operation (different timezones)
- Institutes: 1 in Americas, 2 in Europe,1 in Asia
- Shift-type support during working hours
- Organized feedback sessions

New to Hyperloop? Take the tour!



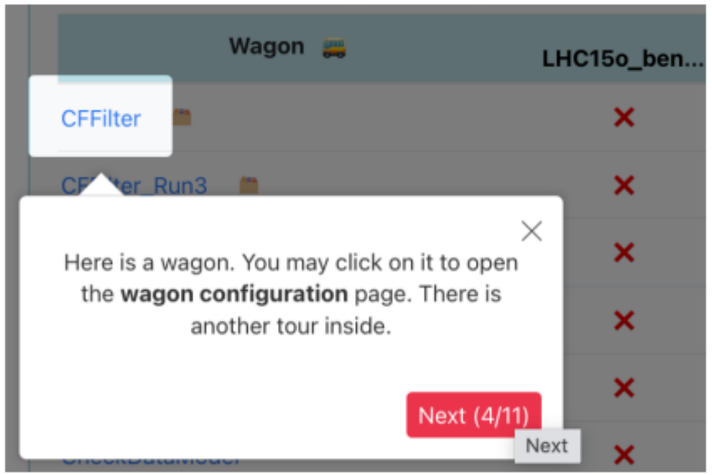


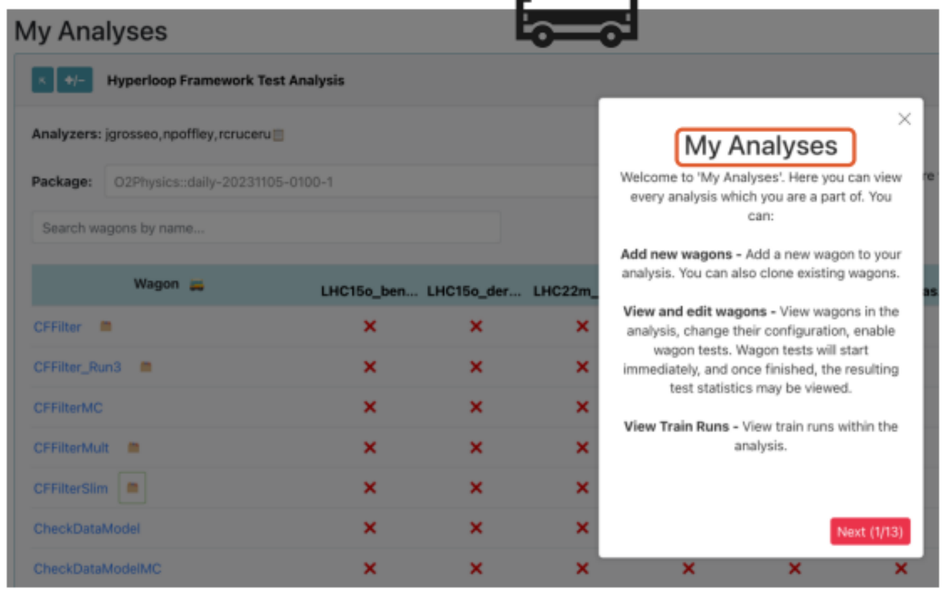
- When opening a page in Hyperloop that you did not visit before, a guided tour will explain key concepts
- Ideal for beginners, but also for refreshing knowledge ***(many tours offer direct links to relevant sections of the documentation)***
- Where appropriate, when one tour ends, the next will begin to explain the next section of Hyperloop
- Tours can be exited at any time. Once closed, they will not automatically begin on future page visits
- A tour can be retaken anytime by clicking

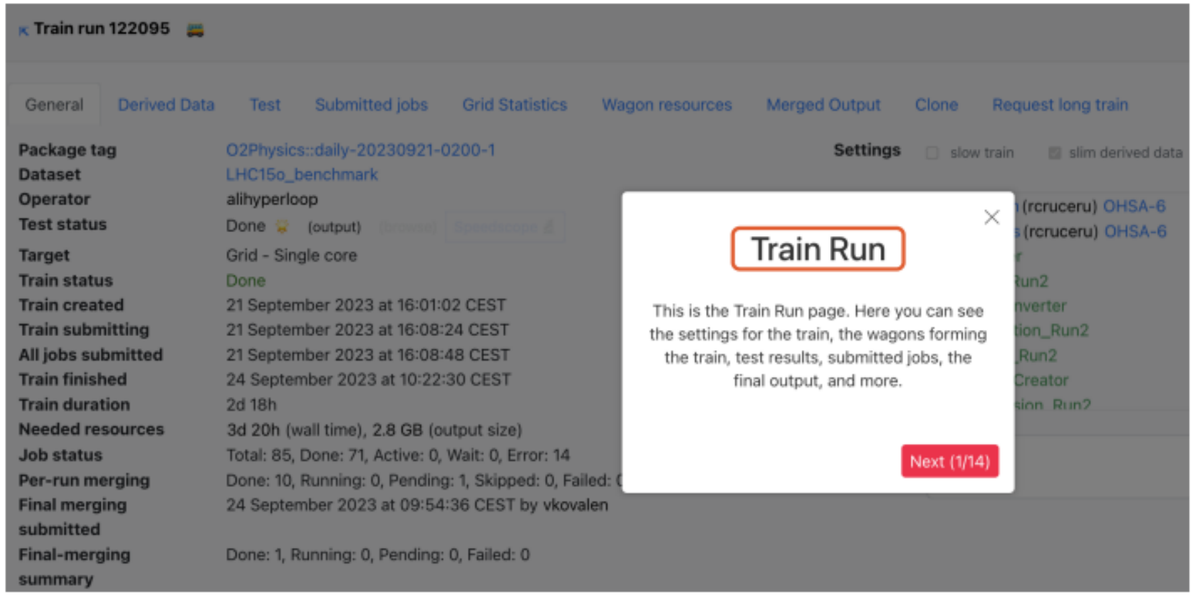
New to Hyperloop? Take the tour!

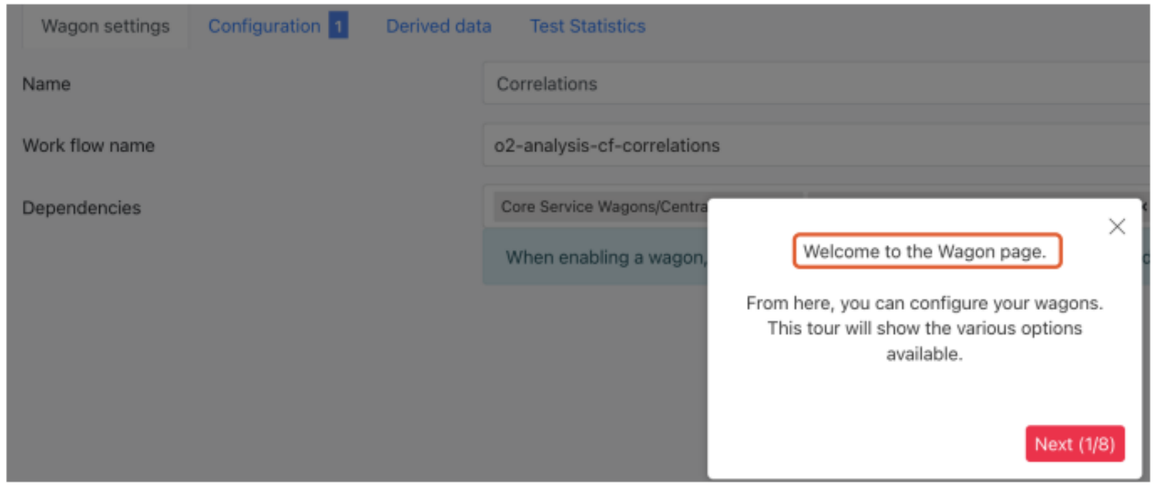
Currently available tours:

- My Analyses page
- Wagon view
- Wagon test view
- Train run view



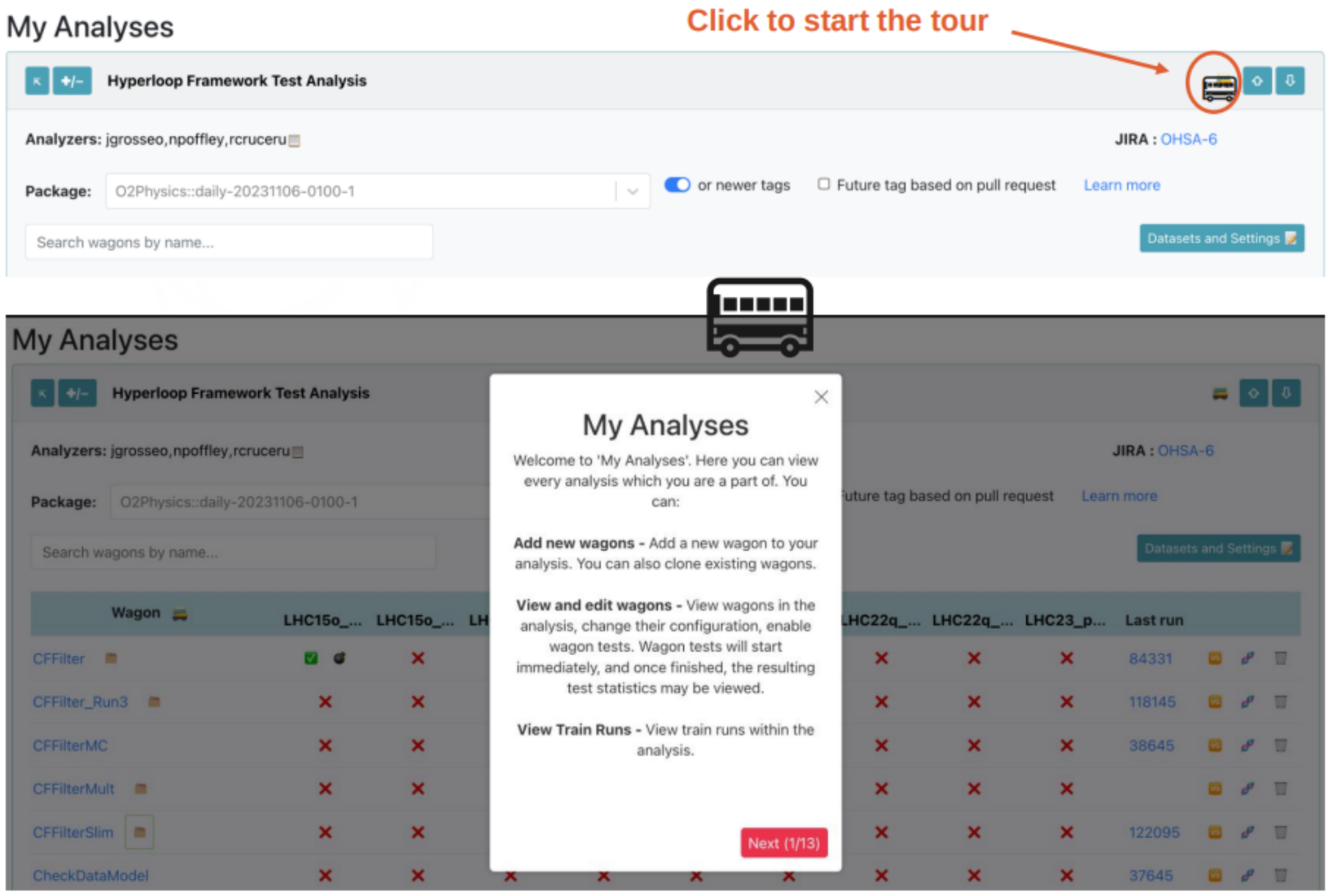






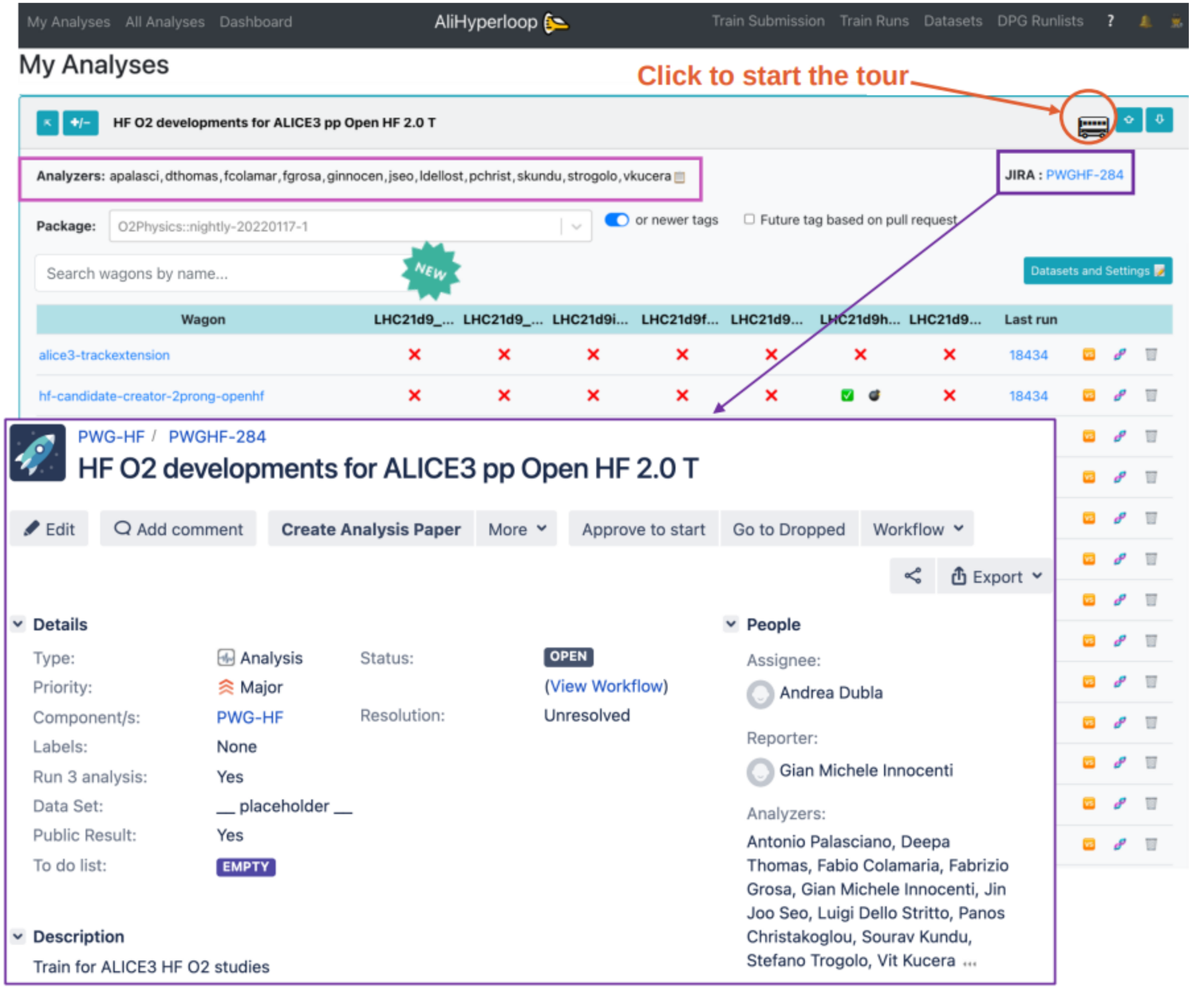
My Analyses

- Displays all the analyses the user belongs to
- The analyses are defined in JIRA of the respective PWG, along with the users that will be part of them
- Several analyzers can share an analysis
- Datasets are enabled per analysis
- The user decides the package tag



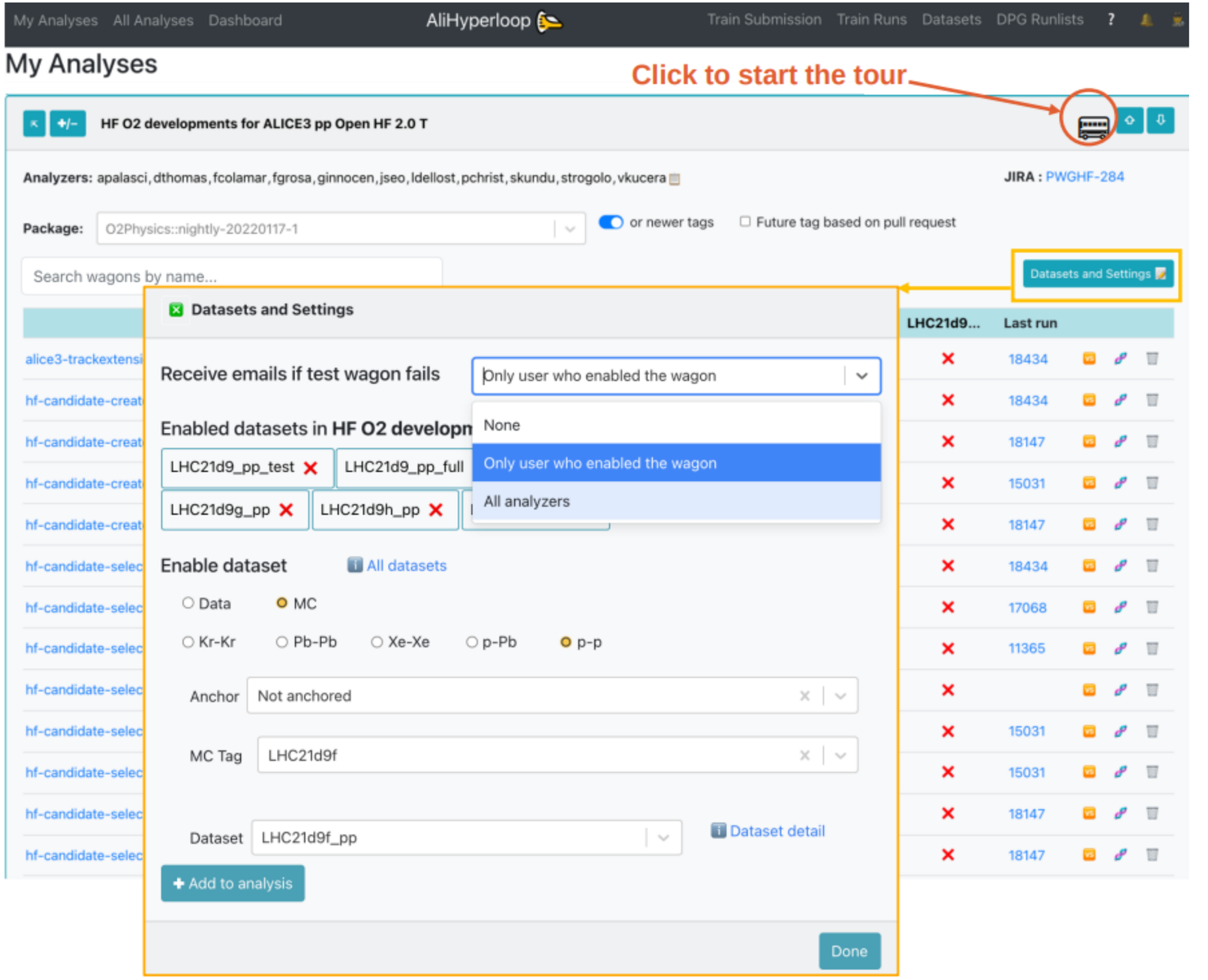
My Analyses

- **Displays all the analyses the user belongs to**
- **The analyses are defined in JIRA of the respective PWG, along with the users that will be part of them**
- **Several analyzers can share an analysis**
- Datasets are enabled per analysis
- The user decides the package tag



My Analyses

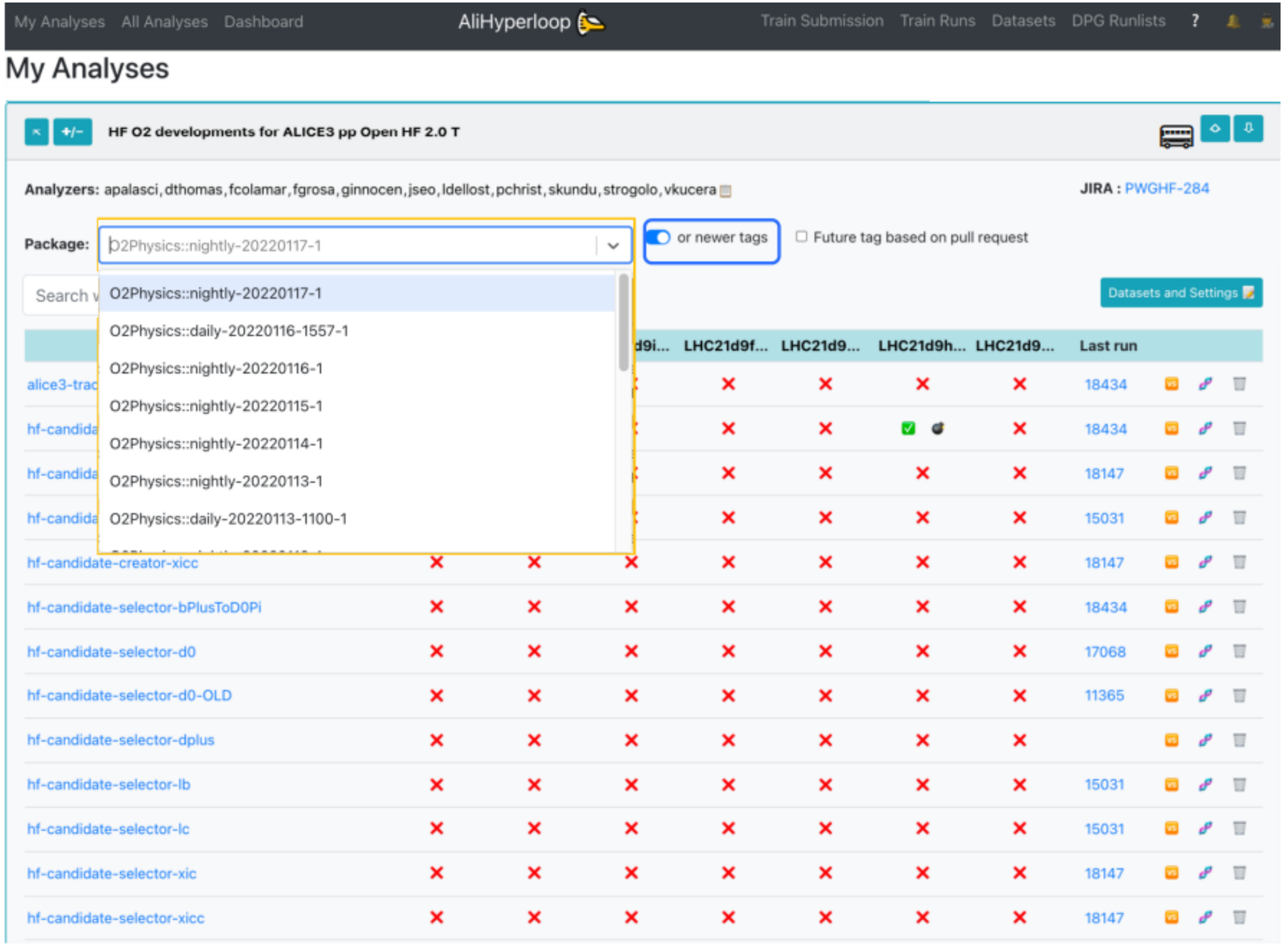
- Displays all the analyses the user belongs to
- The analyses are defined in JIRA, along with the users that will be part of them
- Several analyzers can share an analysis
- **Datasets are enabled per analysis**
- The user decides the package tag



My Analyses

Deciding on package tag:

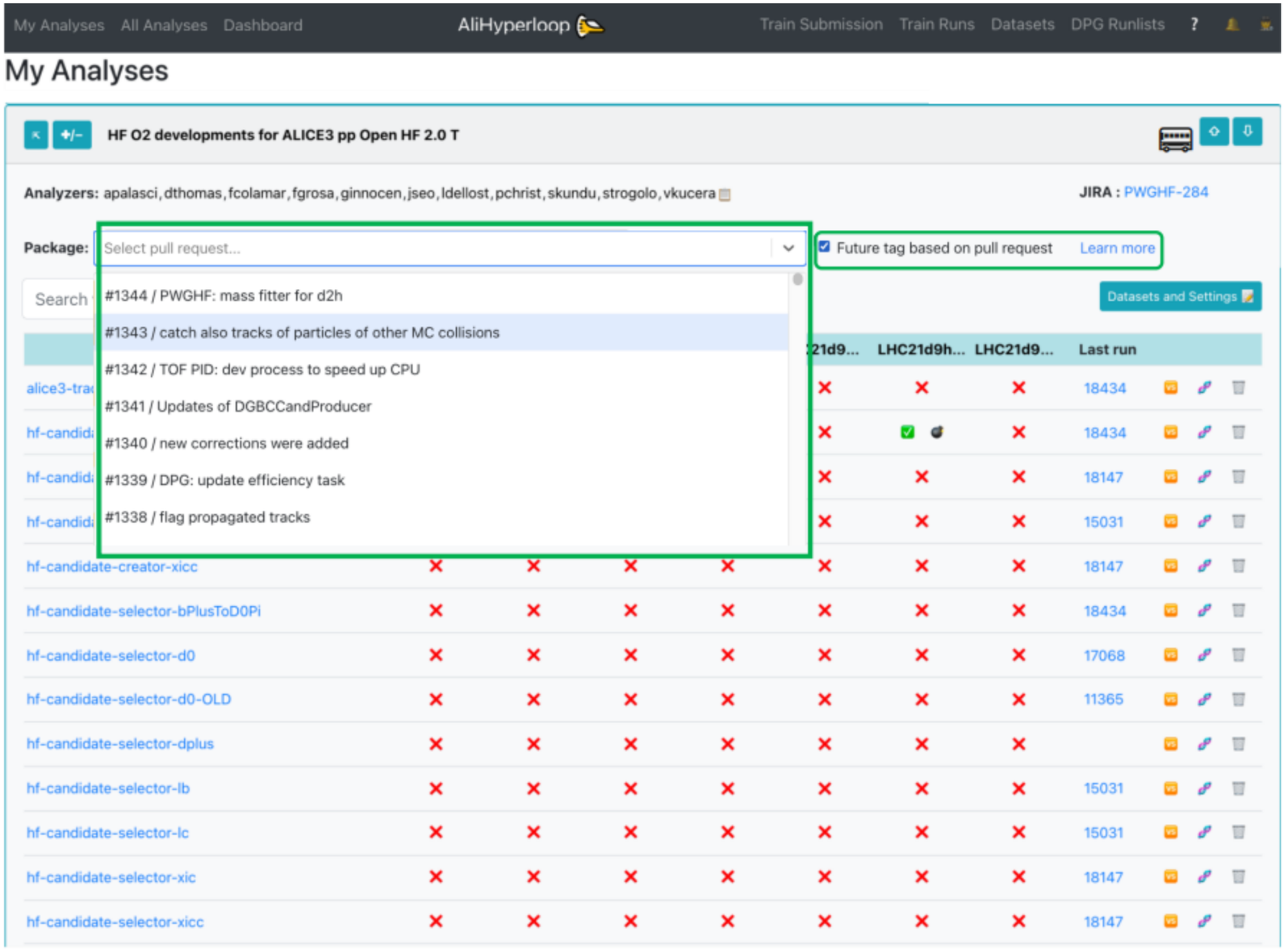
- Choose a tag from the ***dropdown*** and the wagon will be tested only with that specific tag
- Selecting ***or newer tags*** will allow the wagon to be included in a train run that has the previously selected tag or a **newer tag** *(this is useful to group analysis together in a train)*



My Analyses

Deciding on package tag:

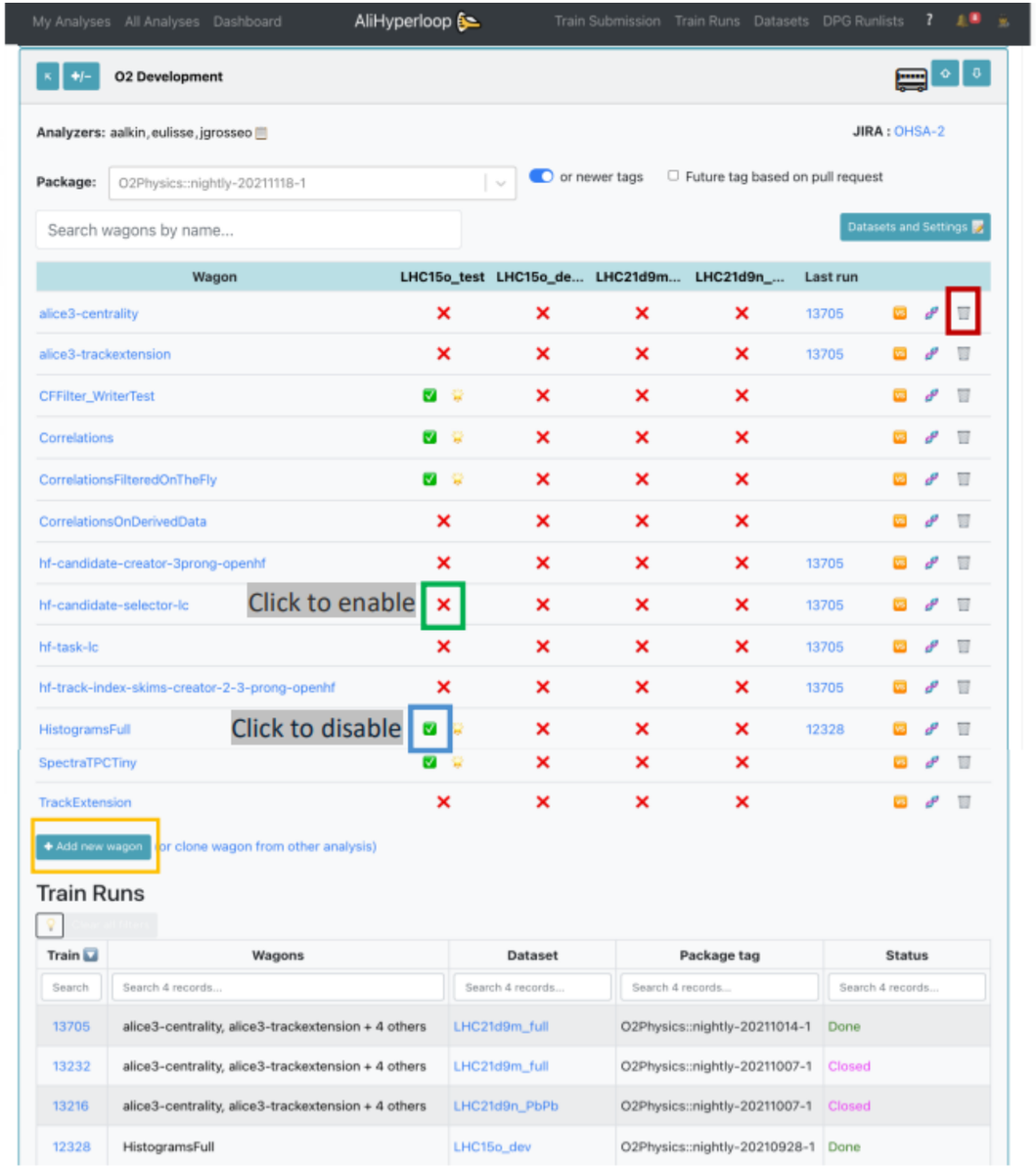
- Choose a tag from the ***dropdown*** and the wagon will be tested only with that specific tag
- Selecting ***or newer tags*** will allow the wagon to be included in a train run that has the previously selected tag or a **newer tag** *(this is useful to group analysis together in a train)*
- ***Future tag based on PR*** **:**
  - The wagon is tested and runs once the *pull request is included in a tag*
  - Once the PR is part of the tag, the wagon can be tested on that tag or a future tag



My Analyses

- Wagons defined from O^2 workflows
- Displays the list of train runs per analysis

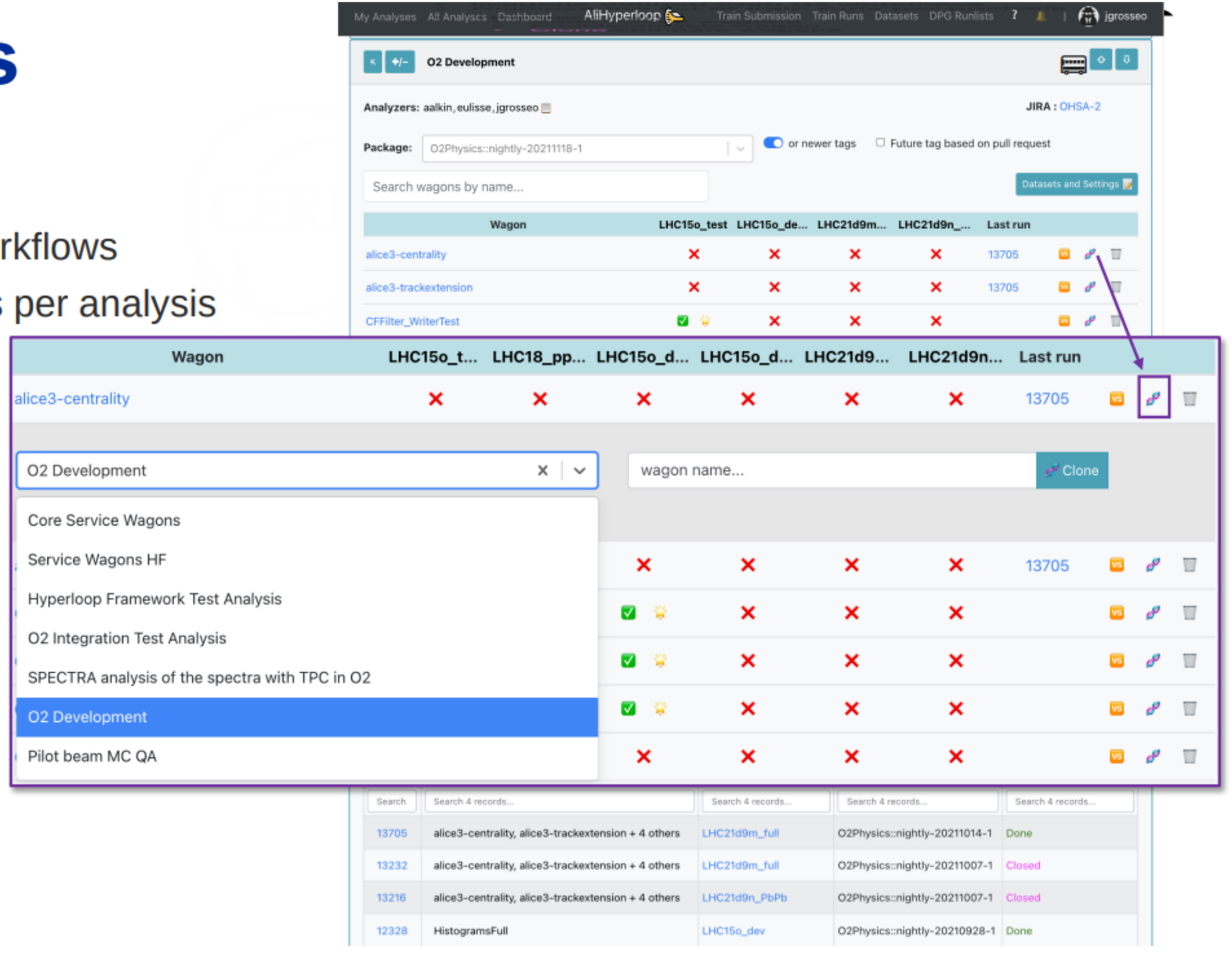
- The user can:
  - Add/remove wagons
  - Enable/disable wagon > immediate wagon test
  - Clone/compare wagons



My Analyses

- Wagons defined from O^2 workflows
- Displays the list of train runs per analysis

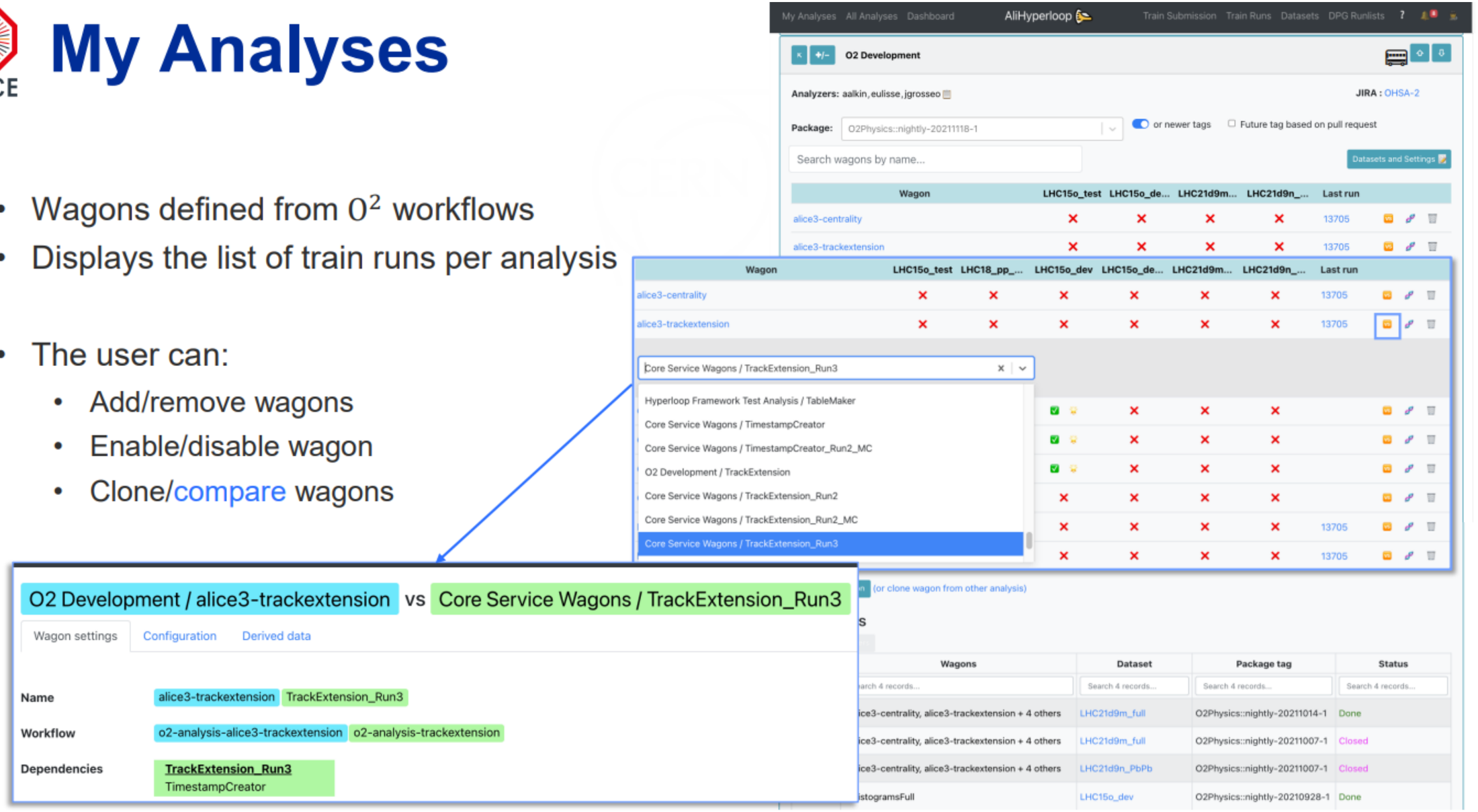
- The user can:
  - Add/remove wagons
  - Enable/disable wagon
  - Clone/compare wagons



My Analyses

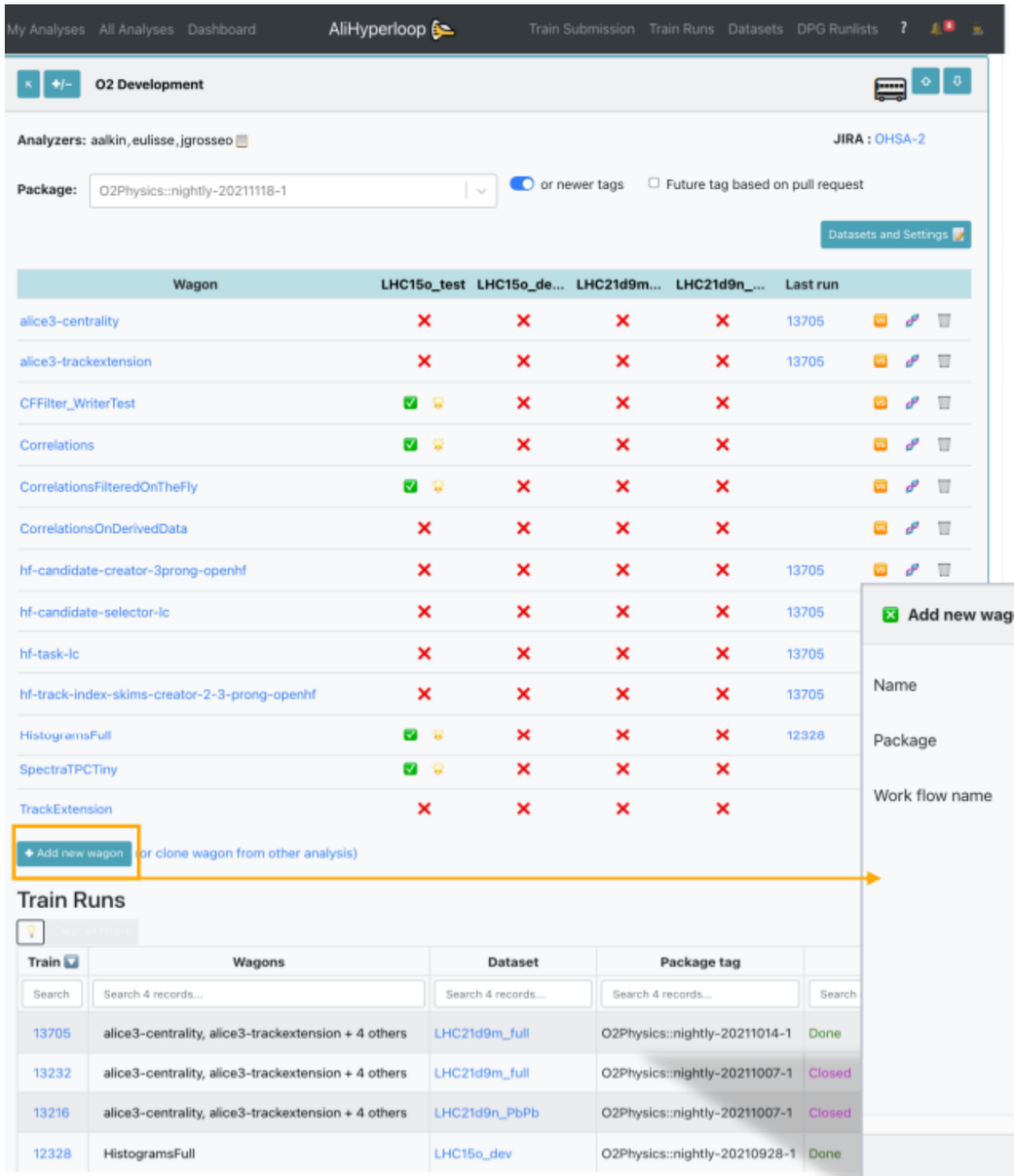
- Wagons defined from O^2 workflows
- Displays the list of train runs per analysis

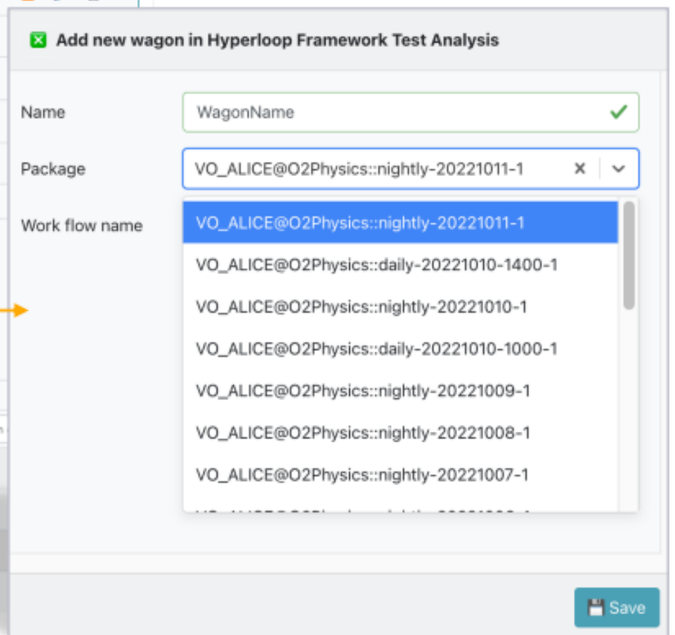
- The user can:
  - Add/remove wagons
  - Enable/disable wagon
  - Clone/compare wagons

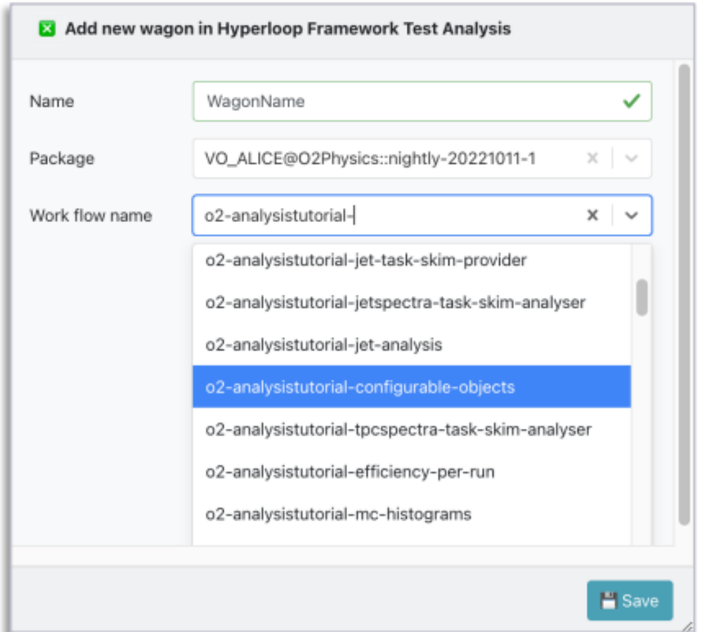


How to create a wagon

- Go to **My Analyses** page
- Click **Add new wagon** within your analysis
- This will open a pop-up window:
  - Insert wagon name
  - Choose Package tag
  - Select Workflow name
  - Click Save to create your new wagon



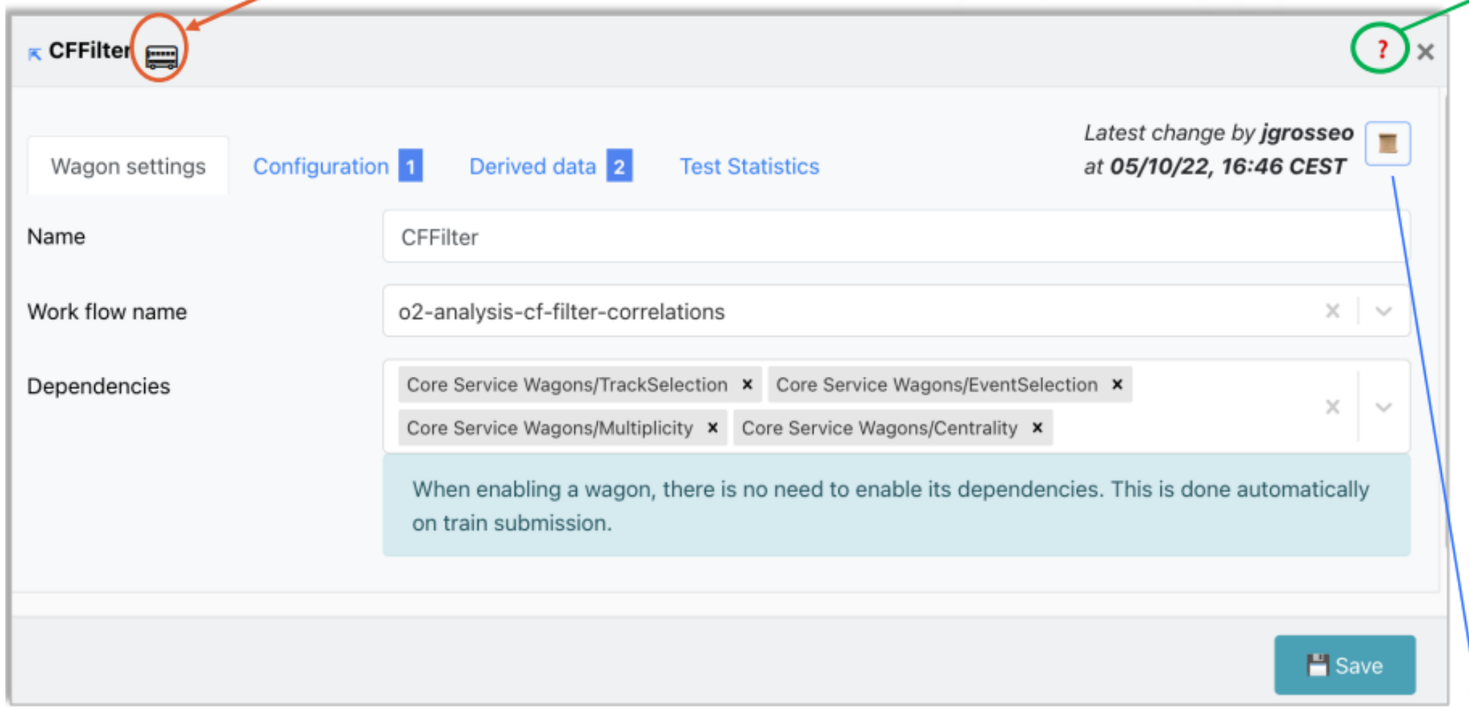




Edit Wagon

Click to start the tour

Documentation



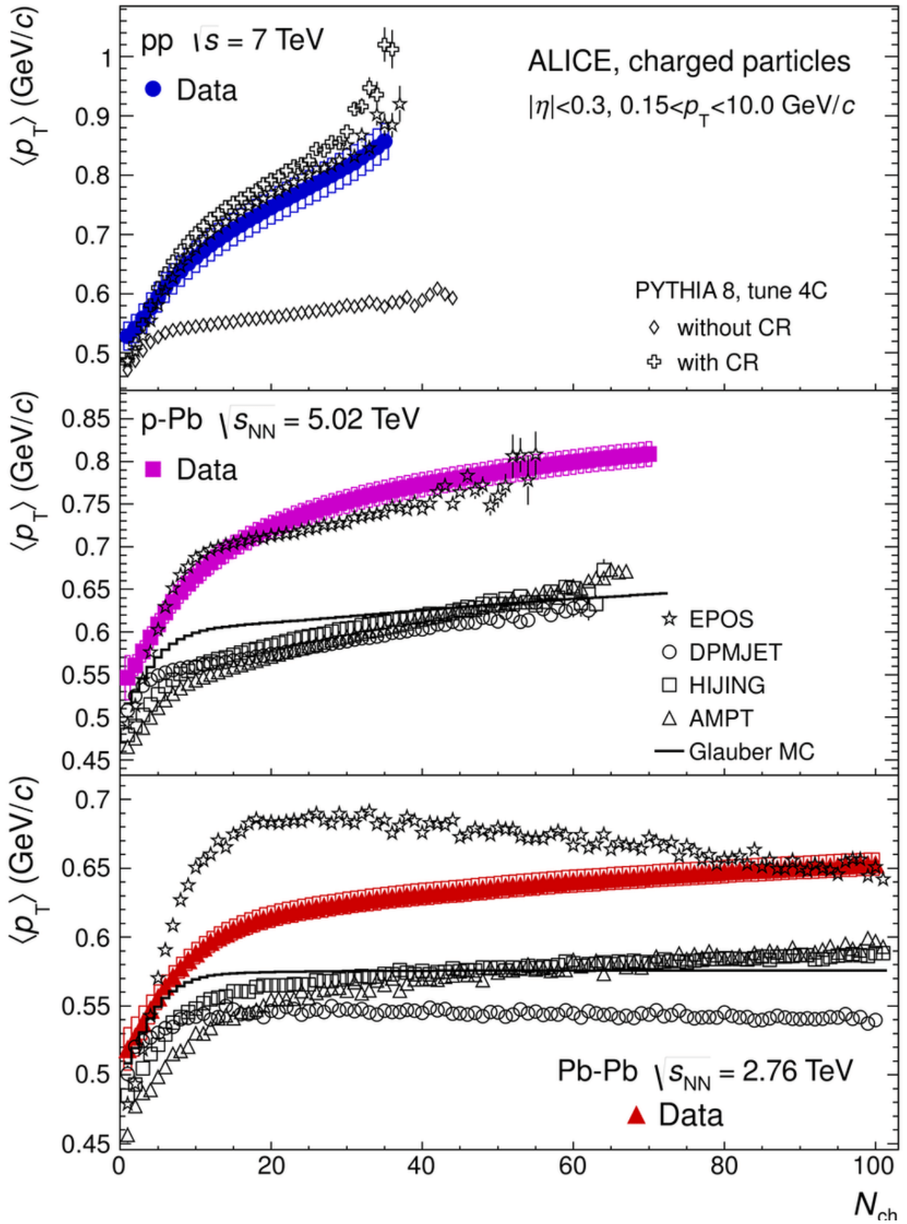
You can edit a wagon by clicking on the *Wagon* name in *My Analyses* page.

- Analyzers who are part of the analysis can add/edit/enable a wagon
- Add dependencies from service wagons or wagons from the same analysis
  - **Dependencies are automatically enabled when enabling a wagon** (this is different from the LEGO trains)
- Can open the detailed **wagon changelog** in a new tab (allows detailed comparison of wagon configuration for different timestamps)

- Direct link to a read-only view in a new tab: https://alimonitor.cern.ch/hyperloop/view-wagon/55
- Used to send to colleagues or to the support list when needed

- Available for wagons, datasets and DPG runlists, this always leads to the history page of the current element

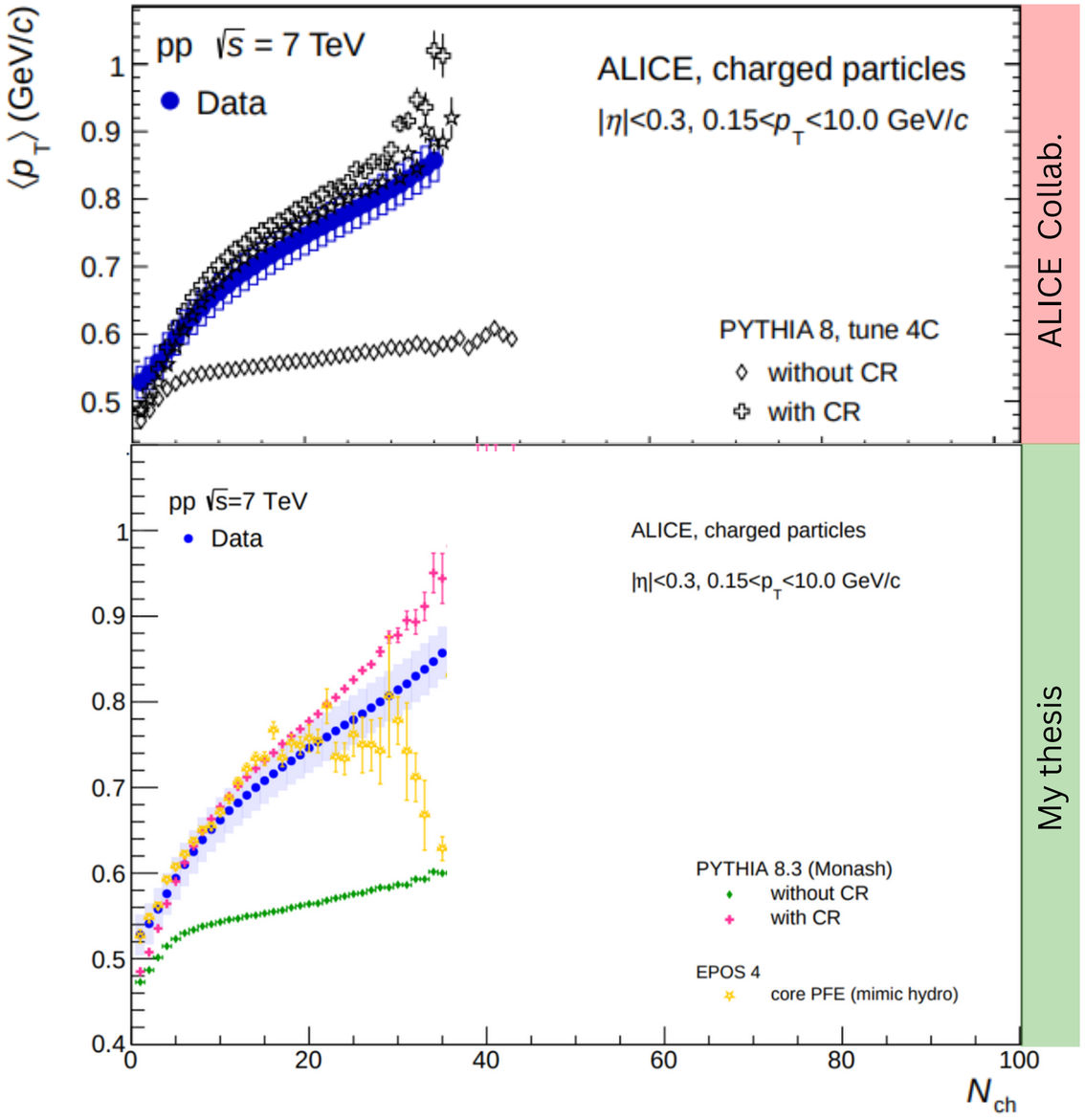
Mean transverse momentum

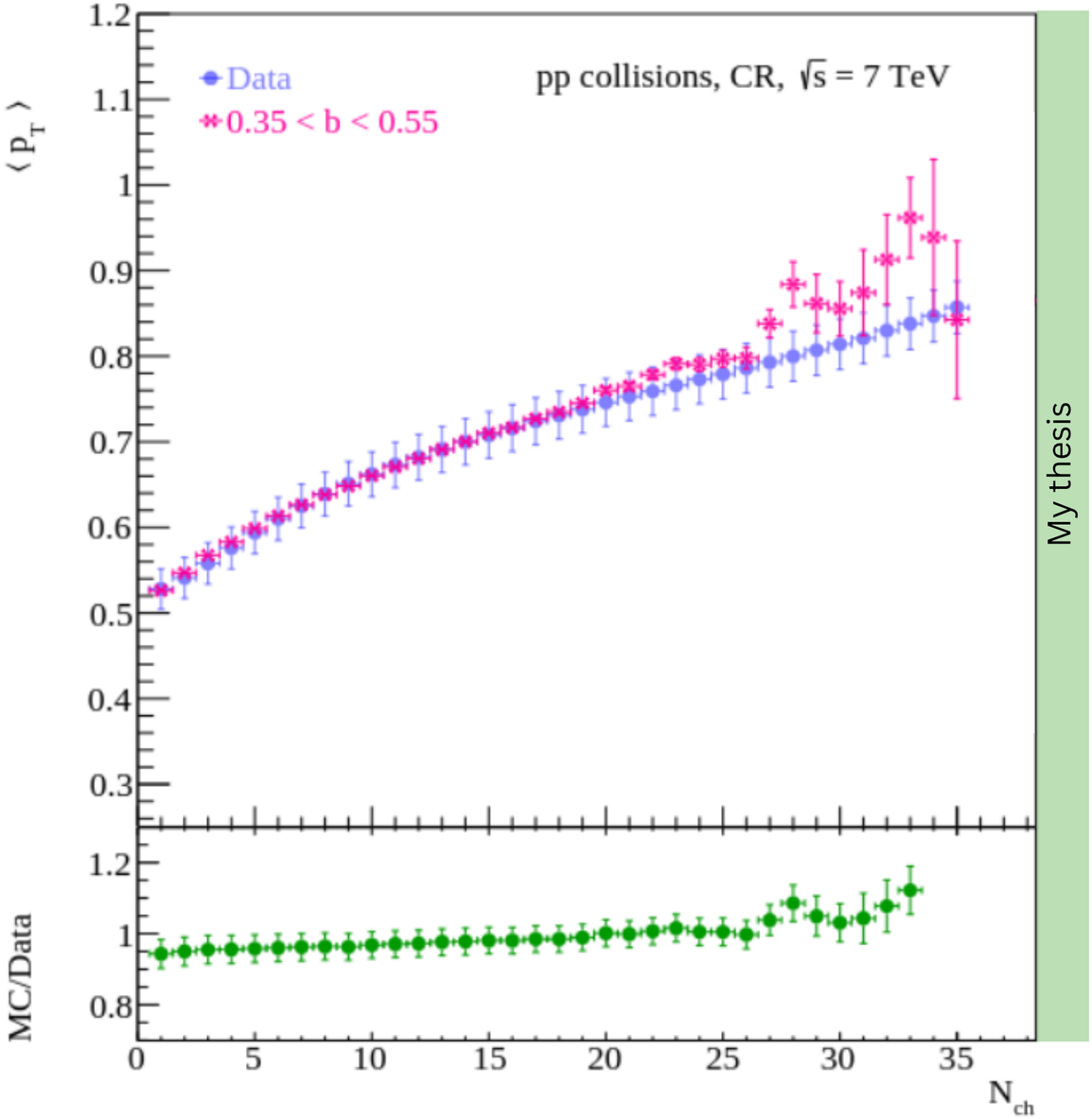


PYTHIA:

```
TH1F* nCharge = new TH1F("nCharge", "Charged Multiplicity", 200, -0.5, 199.5);
// Create a 2D histogram to store the mean transverse momentum (pT) as a function of the charged multiplicity.
TH2F* pTavg = new TH2F("pTavg", "mean pT", 200, -0.5, 199.5, 985, 0.15, 10);

// Begin event loop. Generate event. Skip if error. List first few.
for (int iEvent = 0; iEvent < nEv; ++iEvent) {
    if (!pythia.next()) continue;


    // Find and fill histograms for charged multiplicity and mean transverse momentum.
    int nCh = 0;
    double SpT = 0;

    for (int i = 0; i < pythia.event.size(); ++i) {
        if (pythia.event[i].isFinal()) {
            if (pythia.event[i].isCharged()) {

                SpT += pythia.event[i].pT();

                ++nCh;
            }
        }
    }

    // Fill the histograms with the values obtained from each event.
    nCharge->Fill(nCh);
    pTavg->Fill(nCh, SpT / double(nCh));
}
```

<pT> vs Nch


pp √s = 7 TeV
Data
ALICE, charged particles
|η|<0.3, 0.15<p_T<10.0 GeV/c
PYTHIA 8, tune 4C
without CR
with CR
ALICE Collab.
pp √s=7 TeV
Data
ALICE, charged particles
|η|<0.3, 0.15<p_T<10.0 GeV/c
PYTHIA 8.3 (Monash)
without CR
with CR
EPOS 4
core PFE (mimic hydro)
My thesis
⟨p_T⟩ (GeV/c)
N_ch


Data
0.35 < b < 0.55
pp collisions, CR, √s = 7 TeV
⟨ p_T ⟩
MC/Data
N_ch
My thesis

TASK:

<pT> vs Nch with O^2