How to implement your analysis tasks in the current O2 framework for Run3

or: How I Learned to Stop Worrying and Love the Bomb

Your tutors for you today:
Alexander Kalweit
David Chinellato
Luca Barioglio
Nicolò Jacazio

Big disclaimer

- O2 installation is not supported here
- We are not the experts, we are the LF avantgarde
- This is the “experimental” approach
- These slides depict the current situation and you should refer to the official guide and twiki
- O2 framework is still under development and the instructions given here are only valid contextually to the workshop and with the recent O2 version
- Refer to the WP4+14 meeting on Mondays to follow the development: follow up on indico

Don’t panic!

- During this presentation focus on the topics discussed and on your questions/doubts
- This tutorial is recorded so no rush!
- Detailed instructions on how to write your task and practical examples will be given
- Plenty of documentation, examples already in the O2, support from the O2 community
- If you don’t have O2 installed you can still profit from this so don’t go away!

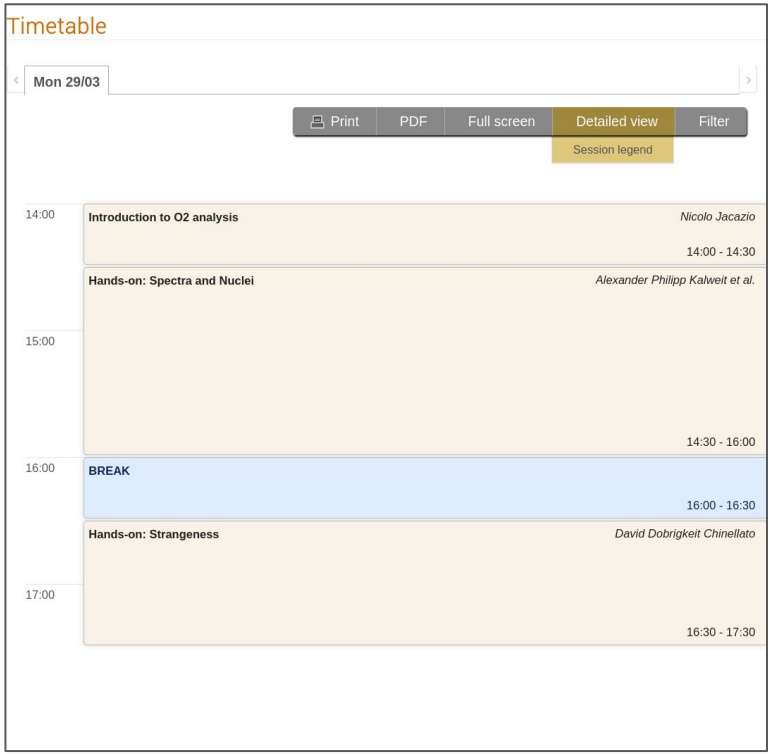
Outline

In this introduction:

- Rationale of the O2 analysis framework
- How to write your own task
- Few advanced topics

In the hands on session:

- Implementing your own task!



References

- Documentation:

https://aliceo2group.github.io/analysis-framework/
https://pbuehler.github.io/documentation/docs/tutorials/analysistask.html

- Tutorial tasks:

https://github.com/pbuehler/documentation/tree/main/docs/tutorials/code

- Detailed session at the last SeC days:

https://indico.cern.ch/event/1014566/#b-412202-session-2-run-3-data

- LF presentations:

https://indico.cern.ch/event/984699/

- Twiki:

https://twiki.cern.ch/twiki/bin/viewauth/ALICE/AliceO2WP14AF

- Monday WP4+14 meetings:

https://indico.cern.ch/event/1022816/

Rationale

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

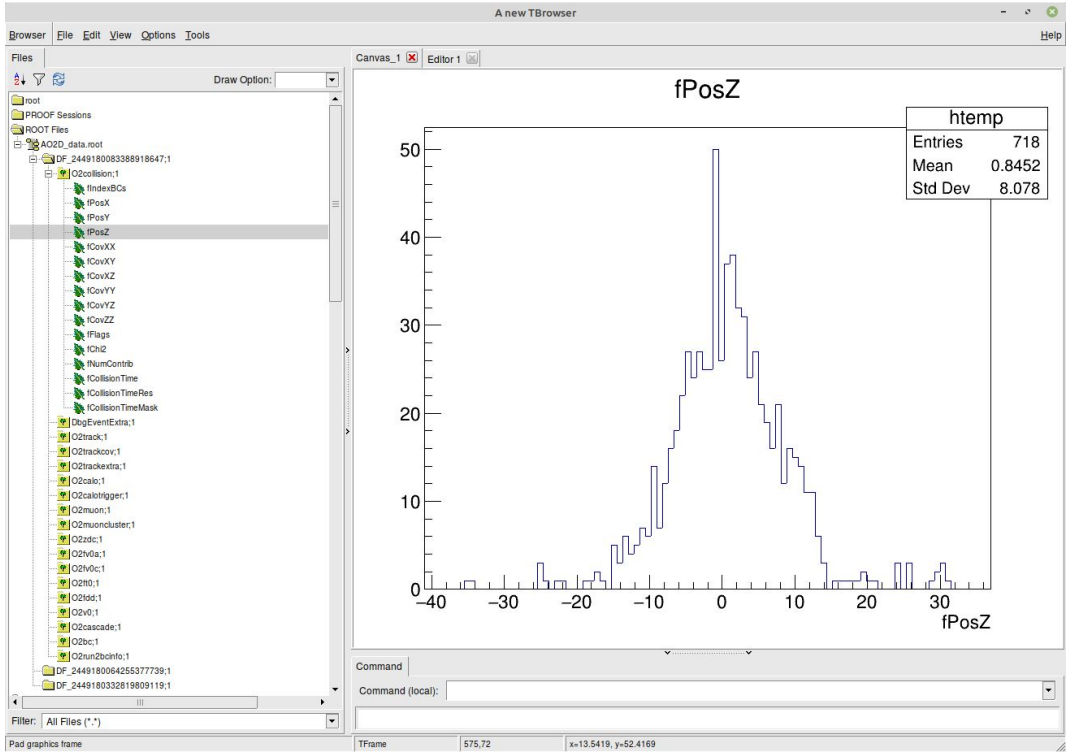
- Using arrow tables https://arrow.apache.org/
- Cross-language development platform for in-memory data
- Columnar memory layout
- Native vectorized optimization of analytical data processing
- Supports zero-copy reads for lightning-fast data access
- Implementation in O2 developed by Giulio et al.

Input data

- Input data is defined depending on the single task needs
- Tasks only access the information they need (the one they subscribe to)
- No overhead
- Depending on needs, tasks can run on collisions only, on single tracks, on all tracks in a data frame and so on
- Input data is organized into split (but linked) tables

Tables, but what do you mean?

- Root files are still the I/O backend of the new analysis framework
- Data is stored in TTrees in TFiles
- O2 AODs (AO2Ds) can be inspected with the browser as old ESDs/AODs
- Tables are read from columnar trees during the analysis



- Data chunked in Data Frames

Input data (1)

Each column is a feature of the primary vertex:

- Vertex position
- Covariance matrix
- $\chi 2$
- Number of contributors
- T0 info
- ...

Table of **Collisions**

	V0M (%)	Z_{vtx} (cm)	...
Row 1	10	1.02	...
Row 2	54	-0.04	...
Row 3	74	4.6	...
...	...	...	...

Input data (2)

Each column is a basic feature of the track:

- Momentum
- Track parameters
- Track type
- ...

Table of **Collisions**

	V0M (%)	Z_{vtx} (cm)	...
Row 1	10	1.02	...
Row 2	54	-0.04	...
Row 3	74	4.6	...
...	...	...	...

From one to the other via indices

	p_T (GeV/c)	φ	...
Row 1	-1.5	0.12	...
Row 2	0.8	2.43	...
Row 3	2.3	-1.65	...
...	...	...	...

Table of **Tracks**

Input data (3)

Each column is an extended feature of the track:

- $\chi 2$
- TPC/TOF/HMPID signal
- ...

Table of **Collisions**

	V0M (%)	Z_{vtx} (cm)	...
Row 1	10	1.02	...
Row 2	54	-0.04	...
Row 3	74	4.6	...
...	...	...	...

From one to the other via indices

	p_T (GeV/c)	φ	...
Row 1	-1.5	0.12	...
Row 2	0.8	2.43	...
Row 3	2.3	-1.65	...
...	...	...	...

Table of **Tracks**

	$\chi 2$	# ITS clusters	...
Row 1	1.4	3	...
Row 2	3	1	...
Row 3	5.4	4	...
...	...	...	...

Table of **TracksExtra**

Writing our first task

- Writing a first ”hello word” task that produces a simple plot is very easy
- a) give your task a memorable name and create a file in `O2/Analysis/Tasks/PWGLF`

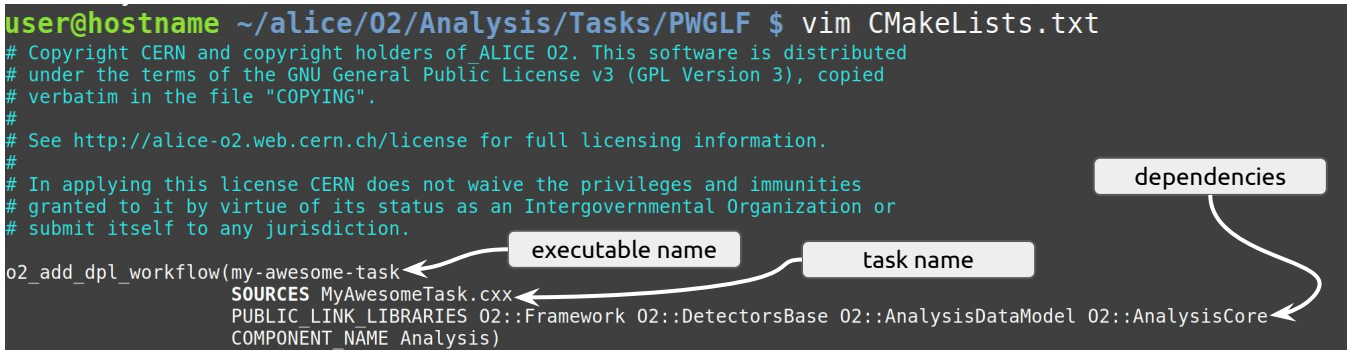
```
user@hostname ~/alice/O2/Analysis/Tasks/PWGLF $ vim CMakeLists.txt
```

- Several examples already in the PWGLF directory

```
user@hostname ~/alice/O2/Analysis/Tasks/PWGLF $ ls
cascadeanalysis.cxx       mcspectraefficiency.cxx   spectraTPC_split.cxx
cascadebuilder.cxx        NucleiSpectraTask.cxx     spectraTPCsplitPiKaPr.cxx
cascadefinder.cxx         raacharged.cxx            spectraTPC_tiny.cxx
CMakeLists.txt            spectraTOF.cxx            spectraTPCtinyPiKaPr.cxx
lambdakzeroanalysis.cxx   spectraTOF_split.cxx     trackchecks.cxx
lambdakzerobuilder.cxx    spectraTOF_tiny.cxx
lambdakzerofinder.cxx     spectraTPC.cxx
```

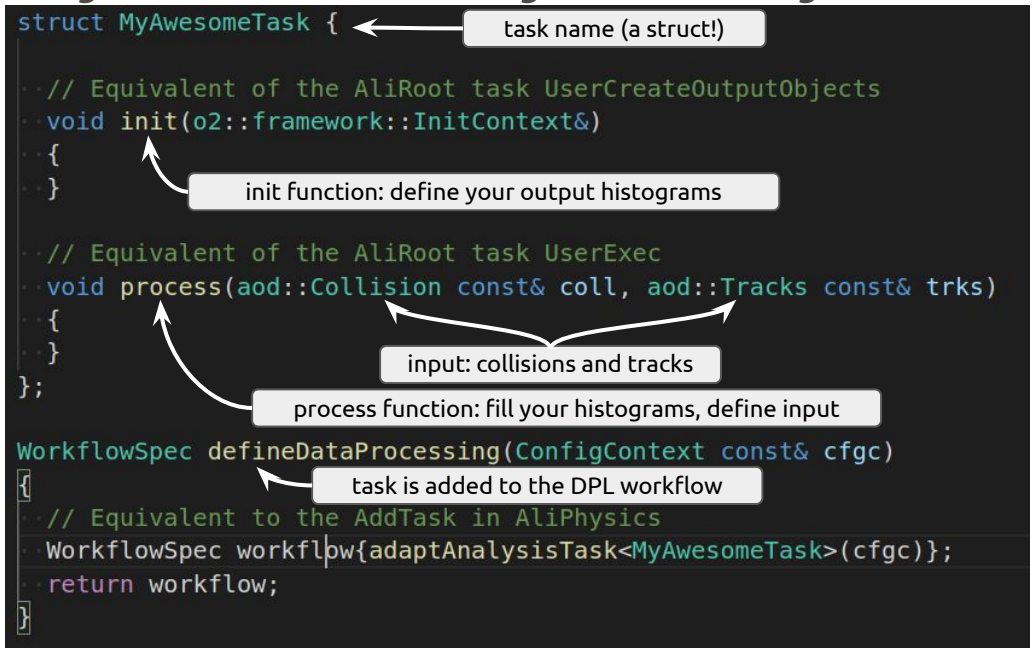
Writing our first task

- Writing a first ”hello word” task that produces a simple plot is very easy
- b) add your task to the Cmake file `O2/Analysis/Tasks/PWGLF/CMakeLists.txt` and define there the workflow name



Writing our first task

- Writing a first ”hello word” task that produces a simple plot is very easy
- c) implement your task with your analysis



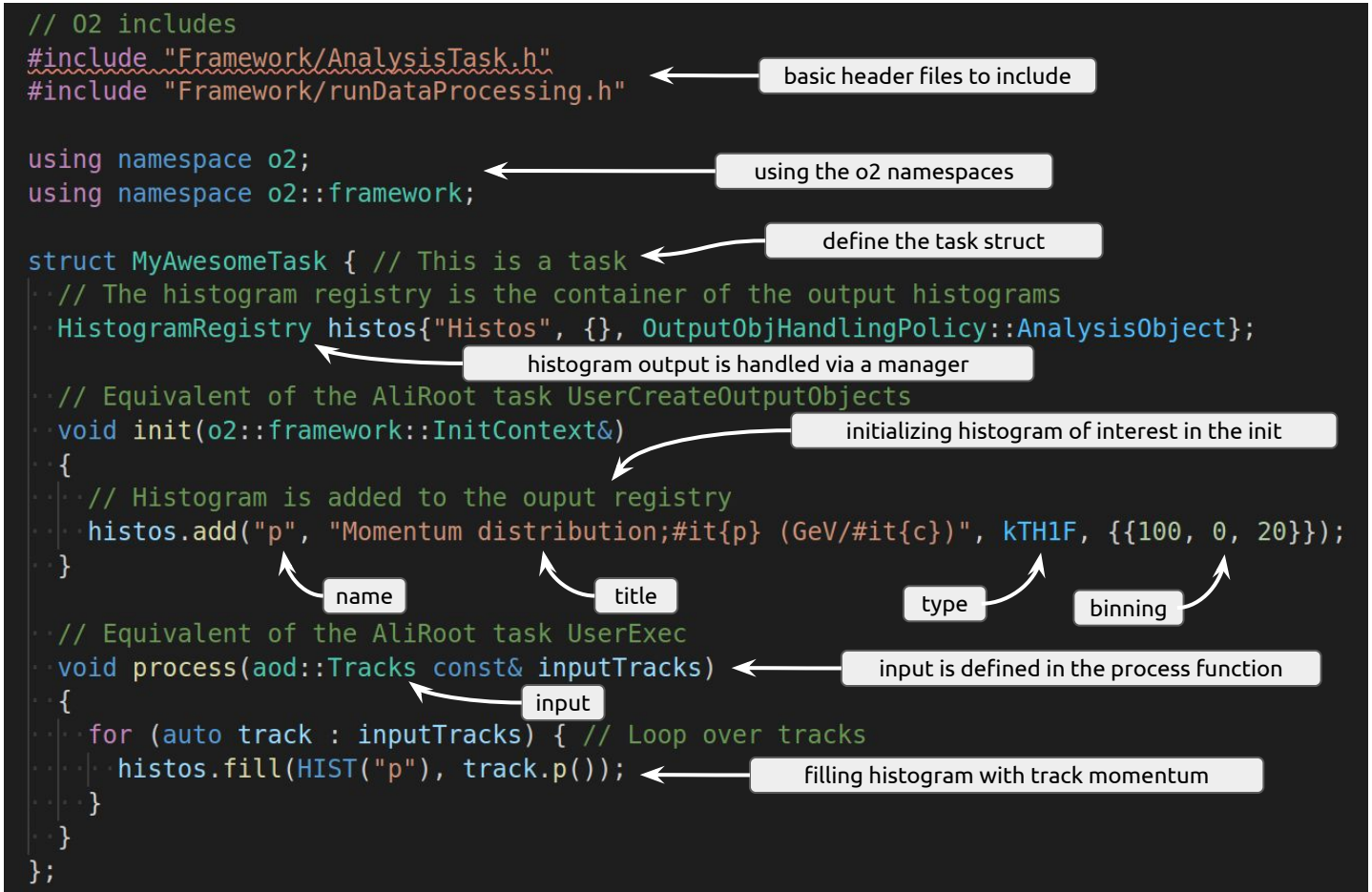
Writing our first task

- Writing a first ”hello word” task that produces a simple plot is very easy
- d) compile O2, enter the O2 environment and enjoy your new analysis workflow!

```
user@hostname ~/alice $ aliBuild build O2 --defaults o2
user@hostname ~/alice $ alienv enter O2/latest
[O2/latest] ~/alice $ o2-analysis-my-awesome-task
```

  - Of course this is the basic example!
  - This task does nothing and needs input data!

Writing our first (real) task



Side-notes on tasks

process method defines the input the task subscribes to

```
// Subscribing to collisions
void process(aod::Collisions const& inputCollisions)

// Subscribing to tracks
void process(aod::Tracks const& inputTracks)

// Subscribing to collisions and tracks
void process(aod::Collisions const& inputCollisions, aod::Tracks const& inputTracks)
```

– Information in separate tables (e.g. Tracks and TracksExtra) can be joined to analyse “full tracks”

```
// Subscribing to tracks and tracksextra
void process(soa::Join<aod::Tracks, aod::TracksExtra> const& inputTracks)
```

Information available in each table: https://github.com/AliceO2Group/AliceO2/blob/dev/Framework/Core/include/Framework/AnalysisDataModel.h

Filtering tables

- The ABC of every analysis is the event/track/candidate selection
- This can be done upfront via filters: filtered data is fed to the task → high performance gain
- All filters are defined as a function returning true or false, and they stack!

```
o2::framework::expressions::Filter pMinFilter = (aod::track::p > 1);
o2::framework::expressions::Filter pMaxFilter = (aod::track::p < 5);

// Equivalent of the AliRoot task UserExec
void process(soa::Filtered<soa::Join<aod::Tracks, aod::TracksExtra>> const& inputTracks)
{
  for (auto track : inputTracks) { // Loop over tracks
    histos.fill(HIST("p"), track.p());
  }
}
```

Configuring your task

- Task parameters can be configured from the outside
- Similar to the parameters of the AddTask in AliPhysics

```
o2::framework::Configurable<float> minP{"minP", 1.0f, "Minimum accepted P"};
o2::framework::Configurable<float> maxP{"maxP", 1.0f, "Maximum accepted P"};

o2::framework::expressions::Filter pMinFilter = (aod::track::p > minP);
o2::framework::expressions::Filter pMaxFilter = (aod::track::p < maxP);
```

- Configurable parameters can be used in filters or in the init/process functions

```
o2-analysis-my-awesome-task -h full
```

→

```
Data processor options: my-awesome-task:
  --minP arg (=1)                       Minimum accepted P
  --maxP arg (=1)                       Maximum accepted P
```

To configure:

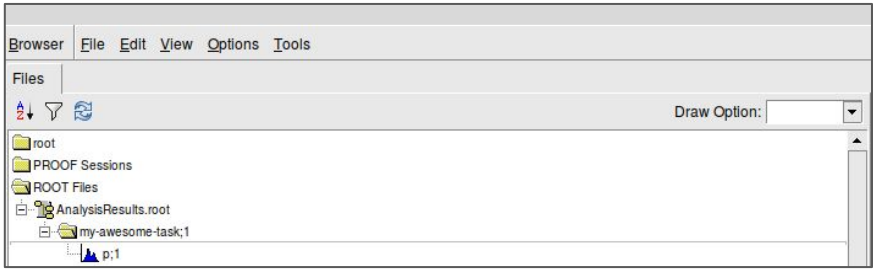
```
o2-analysis-my-awesome-task --minP 2 --maxP 3 --aod-file AO2D.root
```

Adding our task to the workflow

```
WorkflowSpec defineDataProcessing(ConfigContext const& cfgc) // This puts your task in the DPL workflow
{
  // Equivalent to the AddTask in AliPhysics
  WorkflowSpec workflow{adaptAnalysisTask<MyAwesomeTask>(cfgc)};
  return workflow;
}
```

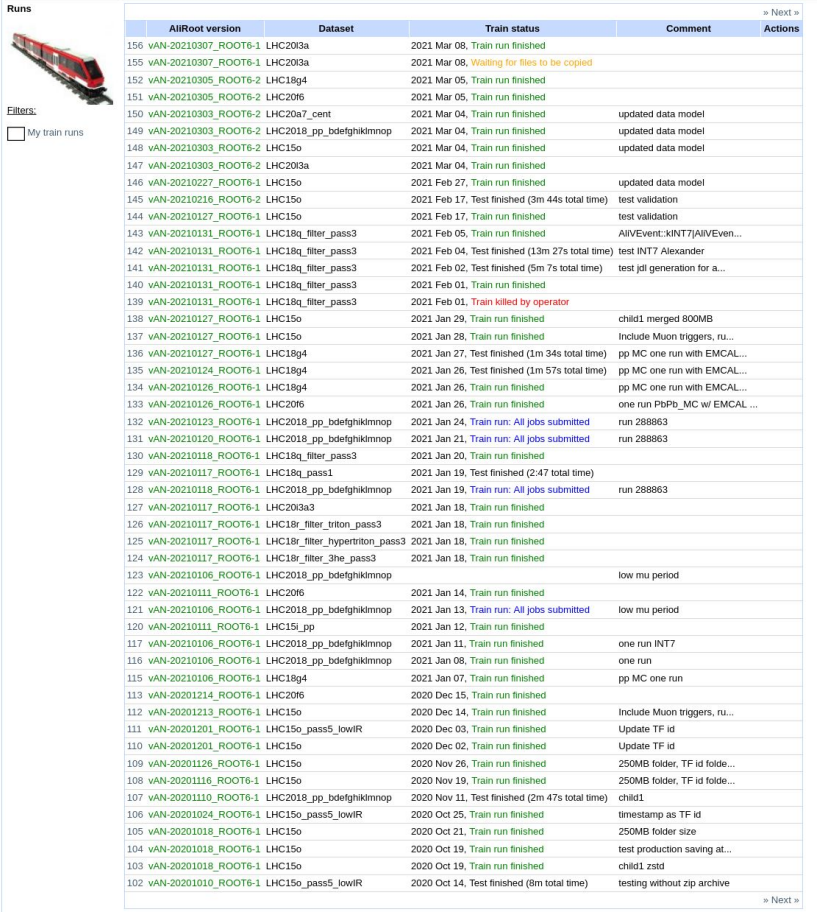
the task is added to the workflow

- Worfklows are the equivalent of the RunTask and the *adaptAnalysisTask* is the equivalent of the *AddTask* in AliPhysics
- Task output is saved in the *AnalysisResults.root* file under the name of the task



Getting data in the O2 format

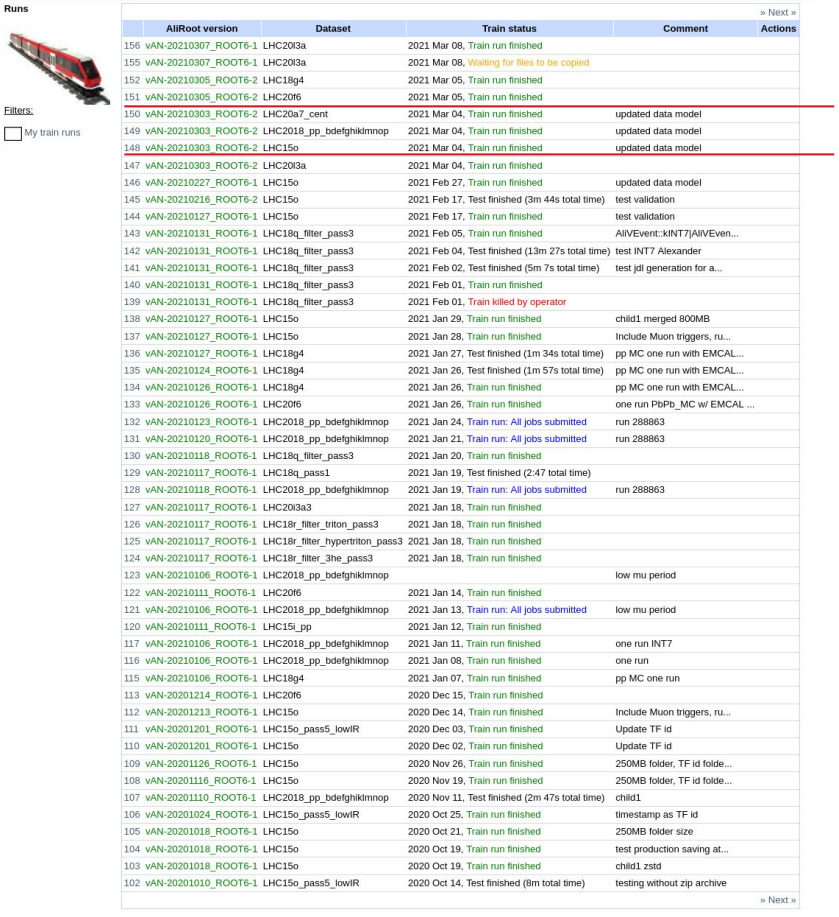
https://alimonitor.cern.ch/trains/train.jsp?train_id=132#datasets



- Data in the O2 data format is (so far) obtained by converting Run2 data (MC and real data)
- Done centrally via trains: [alimonitor Run3 conversion page]
- ESDs can also be converted locally by the user
- Not a 1:1 conversion!

Getting data in the O2 format

https://alimonitor.cern.ch/trains/train.jsp?train_id=132#datasets



Runs

Filters:

My train runs

» Next »

	AliRoot version	Dataset	Train status	Comment	Actions
156	vAN-20210307_ROOT6-1	LHC20l3a	2021 Mar 08, Train run finished		
155	vAN-20210307_ROOT6-1	LHC20l3a	2021 Mar 08, Waiting for files to be copied		
152	vAN-20210305_ROOT6-2	LHC18g4	2021 Mar 05, Train run finished		
151	vAN-20210305_ROOT6-2	LHC20f6	2021 Mar 05, Train run finished		
150	vAN-20210303_ROOT6-2	LHC20a7_cent	2021 Mar 04, Train run finished	updated data model	
149	vAN-20210303_ROOT6-2	LHC2018_pp_bdefghiklmnop	2021 Mar 04, Train run finished	updated data model	
148	vAN-20210303_ROOT6-2	LHC15o	2021 Mar 04, Train run finished	updated data model	
147	vAN-20210303_ROOT6-2	LHC20l3a	2021 Mar 04, Train run finished		
146	vAN-20210227_ROOT6-1	LHC15o	2021 Feb 27, Train run finished	updated data model	
145	vAN-20210216_ROOT6-2	LHC15o	2021 Feb 17, Test finished (3m 44s total time)	test validation	
144	vAN-20210127_ROOT6-1	LHC15o	2021 Feb 17, Train run finished	test validation	
143	vAN-20210131_ROOT6-1	LHC18q_filter_pass3	2021 Feb 05, Train run finished	AliVEvent::kINT7|AliVEven...	
142	vAN-20210131_ROOT6-1	LHC18q_filter_pass3	2021 Feb 04, Test finished (13m 27s total time)	test INT7 Alexander	
141	vAN-20210131_ROOT6-1	LHC18q_filter_pass3	2021 Feb 02, Test finished (5m 7s total time)	test jdl generation for a...	
140	vAN-20210131_ROOT6-1	LHC18q_filter_pass3	2021 Feb 01, Train run finished		
139	vAN-20210131_ROOT6-1	LHC18q_filter_pass3	2021 Feb 01, Train killed by operator		
138	vAN-20210127_ROOT6-1	LHC15o	2021 Jan 29, Train run finished	child1 merged 800MB	
137	vAN-20210127_ROOT6-1	LHC15o	2021 Jan 28, Train run finished	Include Muon triggers, ru...	
136	vAN-20210127_ROOT6-1	LHC18g4	2021 Jan 27, Test finished (1m 34s total time)	pp MC one run with EMCAL...	
135	vAN-20210124_ROOT6-1	LHC18g4	2021 Jan 26, Test finished (1m 57s total time)	pp MC one run with EMCAL...	
134	vAN-20210126_ROOT6-1	LHC18g4	2021 Jan 26, Train run finished	pp MC one run with EMCAL...	
133	vAN-20210126_ROOT6-1	LHC20f6	2021 Jan 26, Train run finished	one run PbPb_MC w/ EMCAL ...	
132	vAN-20210123_ROOT6-1	LHC2018_pp_bdefghiklmnop	2021 Jan 24, Train run: All jobs submitted	run 288863	
131	vAN-20210120_ROOT6-1	LHC2018_pp_bdefghiklmnop	2021 Jan 21, Train run: All jobs submitted	run 288863	
130	vAN-20210118_ROOT6-1	LHC18q_filter_pass3	2021 Jan 20, Train run finished		
129	vAN-20210117_ROOT6-1	LHC18q_pass1	2021 Jan 19, Test finished (2:47 total time)		
128	vAN-20210118_ROOT6-1	LHC2018_pp_bdefghiklmnop	2021 Jan 19, Train run: All jobs submitted	run 288863	
127	vAN-20210117_ROOT6-1	LHC20l3a3	2021 Jan 18, Train run finished		
126	vAN-20210117_ROOT6-1	LHC18r_filter_triton_pass3	2021 Jan 18, Train run finished		
125	vAN-20210117_ROOT6-1	LHC18r_filter_hypertriton_pass3	2021 Jan 18, Train run finished		
124	vAN-20210117_ROOT6-1	LHC18r_filter_3he_pass3	2021 Jan 18, Train run finished		
123	vAN-20210106_ROOT6-1	LHC2018_pp_bdefghiklmnop		low mu period	
122	vAN-20210111_ROOT6-1	LHC20f6	2021 Jan 14, Train run finished		
121	vAN-20210106_ROOT6-1	LHC2018_pp_bdefghiklmnop	2021 Jan 13, Train run: All jobs submitted	low mu period	
120	vAN-20210111_ROOT6-1	LHC15i_pp	2021 Jan 12, Train run finished		
117	vAN-20210106_ROOT6-1	LHC2018_pp_bdefghiklmnop	2021 Jan 11, Train run finished	one run INT7	
116	vAN-20210106_ROOT6-1	LHC2018_pp_bdefghiklmnop	2021 Jan 08, Train run finished	one run	
115	vAN-20210106_ROOT6-1	LHC18g4	2021 Jan 07, Train run finished	pp MC one run	
113	vAN-20201214_ROOT6-1	LHC20f6	2020 Dec 15, Train run finished		
112	vAN-20201213_ROOT6-1	LHC15o	2020 Dec 14, Train run finished	Include Muon triggers, ru...	
111	vAN-20201201_ROOT6-1	LHC15o_pass5_lowIR	2020 Dec 03, Train run finished	Update TF id	
110	vAN-20201201_ROOT6-1	LHC15o	2020 Dec 02, Train run finished	Update TF id	
109	vAN-20201126_ROOT6-1	LHC15o	2020 Nov 26, Train run finished	250MB folder, TF id folde...	
108	vAN-20201116_ROOT6-1	LHC15o	2020 Nov 19, Train run finished	250MB folder, TF id folde...	
107	vAN-20201110_ROOT6-1	LHC2018_pp_bdefghiklmnop	2020 Nov 11, Test finished (2m 47s total time)	child1	
106	vAN-20201024_ROOT6-1	LHC15o_pass5_lowIR	2020 Oct 25, Train run finished	timestamp as TF id	
105	vAN-20201018_ROOT6-1	LHC15o	2020 Oct 21, Train run finished	250MB folder size	
104	vAN-20201018_ROOT6-1	LHC15o	2020 Oct 19, Train run finished	test production saving at...	
103	vAN-20201018_ROOT6-1	LHC15o	2020 Oct 19, Train run finished	child1 zstd	
102	vAN-20201010_ROOT6-1	LHC15o_pass5_lowIR	2020 Oct 14, Test finished (8m total time)	testing without zip archive	

» Next »

- This tutorial will rely on PbPb converted data
- Data: train 148 LHC15o
- MC: train 151 LHC20f6 (anchored to LHC15o)
- **N.B.** older trains might be inconsistent with respect to the latest version of the data format

Getting data and running the task

- Getting data:

```
alien_cp alien:///alice/data/2015/LHC15o/000244918/pass5_lowIR/PWGZZ/Run3_Conversion/148_20210304-0829_child_1/0017/AO2D.root /tmp/AO2D_data.root
```

- Getting MC:

```
alien_cp alien:///alice/sim/2020/LHC20f6/244918/PWGZZ/Run3_Conversion/151_20210305-2339/0017/AO2D.root /tmp/AO2D_mc.root
```

- Running the task on data

```
o2-analysis-my-awesome-task --aod-file /tmp/AO2D_data.root
```

- Running the task on MC

```
o2-analysis-my-awesome-task --aod-file /tmp/AO2D_mc.root
```

Either of the two should give an AnalysisResults.root file as output

Getting data and running the task

- Getting data:

```
alien_cp alien:///alice/data/2015/LHC15o/000244918/pass5_lowIR/PWGZZ/Run3_Conversion/148_20210304-0829_child_1/0017/AO2D.root /tmp/AO2D_data.root
```

- Getting MC:

```
alien_cp alien:///alice/sim/2020/LHC20f6/244918/PWGZZ/Run3_Conversion/151_20210305-2339/0017/AO2D.root /tmp/AO2D_mc.root
```

- Running the task on data

```
o2-analysis-my-awesome-task --aod-file /tmp/AO2D_data.root
```

Defining input file

- Running the task on MC

```
o2-analysis-my-awesome-task --aod-file /tmp/AO2D_mc.root
```

Either of the two should give an AnalysisResults.root file as output

Getting data and running the task

- Getting data:

```
alien_cp alien:///alice/data/2015/LHC15o/000244918/pass5_lowIR/PWGZZ/Run3_Conversion/148_20210304-0829_child_1/0017/AO2D.root /tmp/AO2D_data.root
```

- Getting MC:

```
alien_cp alien:///alice/sim/2020/LHC20f6/244918/PWGZZ/Run3_Conversion/151_20210305-2339/0017/AO2D.root /tmp/AO2D_mc.root
```

Defined in O2/Analysis/Tasks/PWGLF/CmakeLists.txt

```
o2_add_dpl_workflow(my-awesome-task
                    SOURCES MyAwesomeTask.cxx
                    PUBLIC_LINK_LIBRARIES O2::Framework O2::DetectorsBase O2::AnalysisDataModel O2::AnalysisCore
                    COMPONENT_NAME Analysis)
```

- Running the task on data

```
o2-analysis-my-awesome-task --aod-file /tmp/AO2D_data.root
```

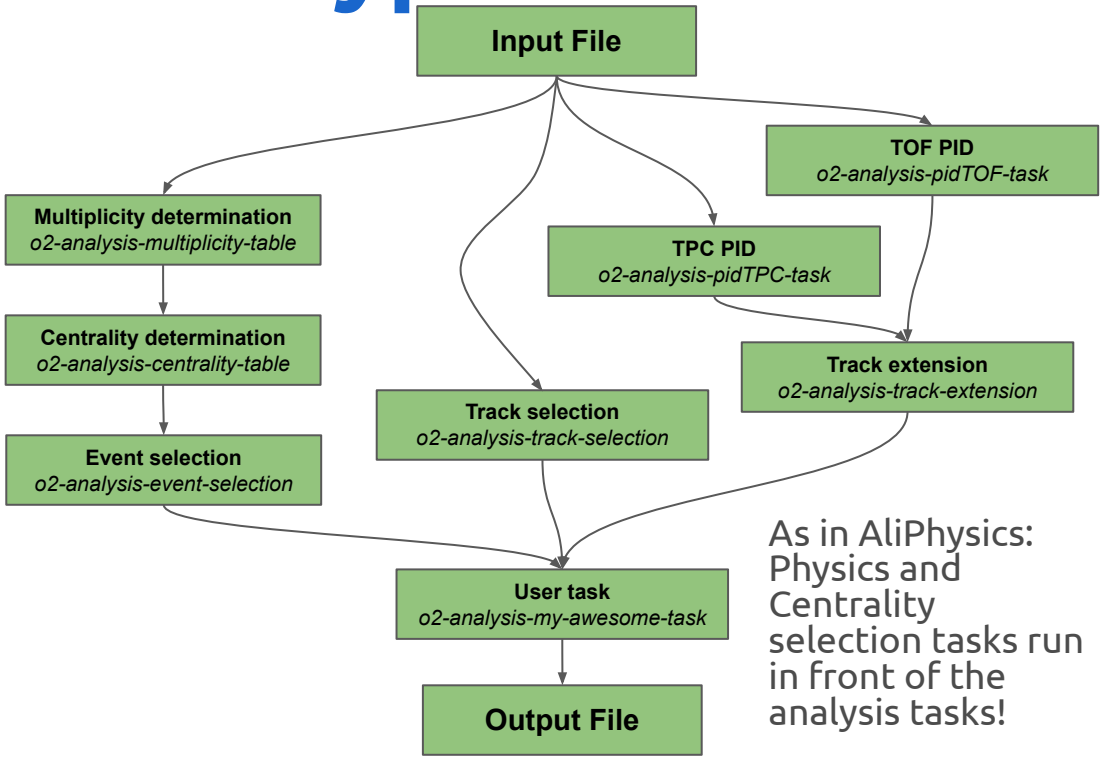
- Running the task on MC

```
o2-analysis-my-awesome-task --aod-file /tmp/AO2D_mc.root
```

Either of the two should give an AnalysisResults.root file as output

Merging workflows: typical use case

- Usually your analysis task at the end of a long chain of precursor tasks
- Tasks can produce tables as output (e.g. for PID Response) this can be used as input in another
- The order of each element is executed according to its dependencies and to the tables

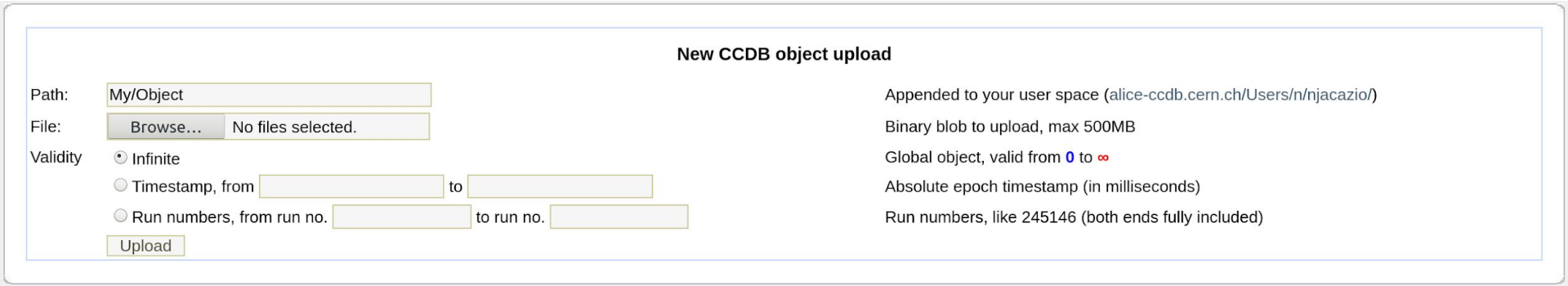


As in AliPhysics: Physics and Centrality selection tasks run in front of the analysis tasks!

```
user@hostname /tmp $ o2-analysis-my-awesome-task --aod-file AO2D.root |
o2-analysis-trackselection  | o2-analysis-trackextension  |
o2-analysis-pid-tpc | o2-analysis-pid-tof |
o2-analysis-centrality-table | o2-analysis-event-selection |
o2-analysis-multiplicity-table | o2-analysis-timestamp
```

Run time access to objects on grid

- Access to external input e.g. finer calibration, corrections is given through the CCDB (the OCDB successor)
- User objects needed in the analysis are uploaded to the CCDB: https://alimonitor.cern.ch/ccdb/upload.jsp



Example user task: https://github.com/AliceO2Group/AliceO2/blob/dev/Analysis/Tutorials/src/efficiencyPerRun.cxx

Access to CCDB from a task

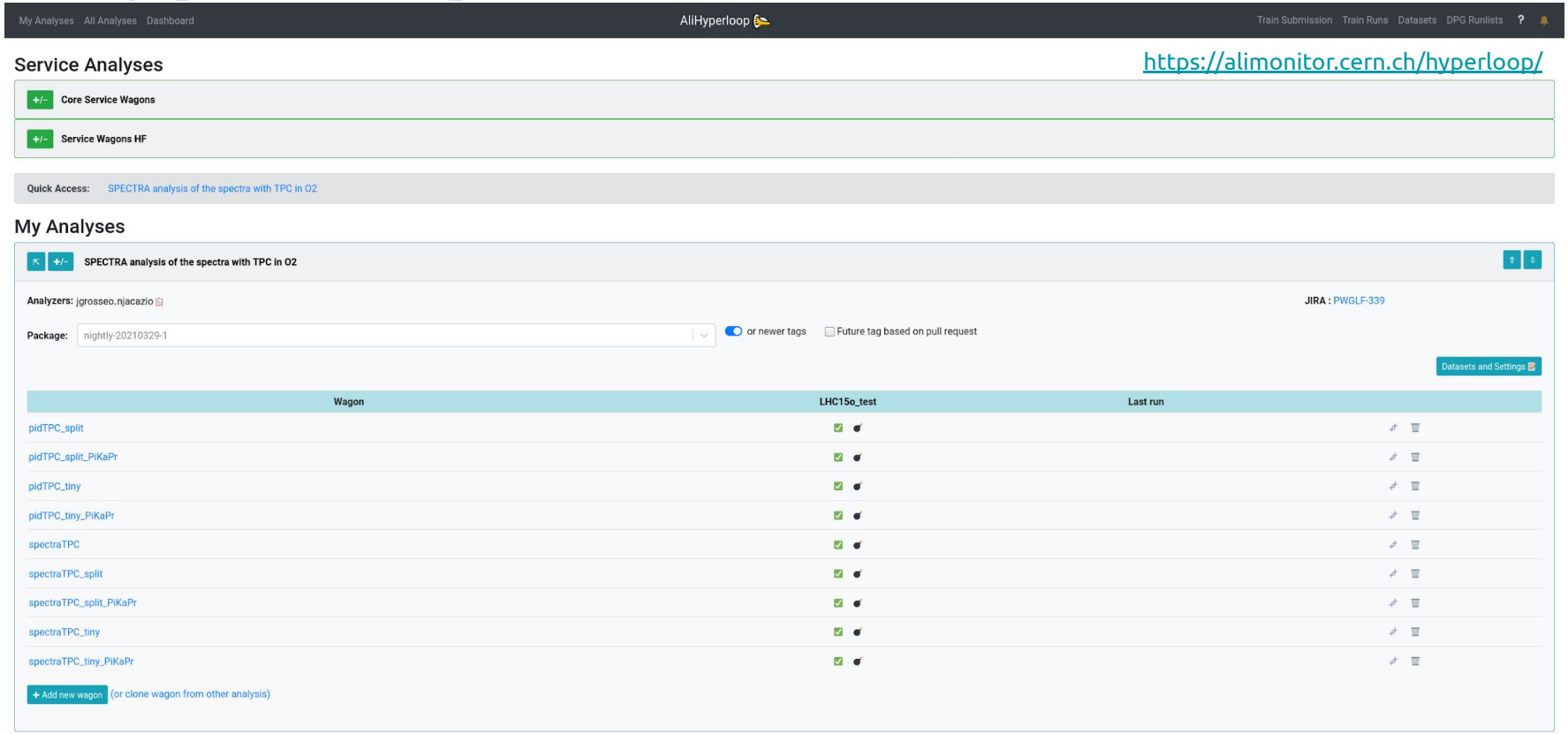
- Example of task using efficiency provided as tutorial

https://github.com/AliceO2Group/AliceO2/blob/dev/Analysis/Tutorials/src/efficiencyPerRun.cxx

```
Service<ccdb::BasicCCDBManager> ccdb;
Configurable<std::string> path{"ccdb-path", "Users/j/jgrosseo/tutorial/efficiency/simple", "base path to the ccdb object"};
Configurable<std::string> url{"ccdb-url", "http://alice-ccdb.cern.ch", "url of the ccdb repository"};
Configurable<long> nolaterthan{"ccdb-no-later-than", std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().
time_since_epoch()).count(), "latest acceptable timestamp of creation for the object"};

OutputObj<TH1F> pt{TH1F("pt", "pt", 20, 0., 10.)};
void process(aod::Collision const& collision, aod::BCsWithTimestamps const&, aod::Tracks const& tracks)
{
  auto bc = collision.bc_as<aod::BCsWithTimestamps>();
  LOGF(info, "Getting object %s for run number %i from timestamp=%llu", path.value.data(), bc.runNumber(), bc.timestamp());
  auto efficiency = ccdb->getForTimeStamp<TH1F>(path.value, bc.timestamp());
  if (!efficiency) {
    LOGF(fatal, "Efficiency object not found!");
  }
}
```

Hyperloop is the new LEGO trains



– Instructions to get your analysis on hyperloop: https://aliceo2group.github.io/analysis-framework/hyperloop.html#allanalyses

Conclusions

- If you want to get involved in the O2 Analysis Framework
- Participate to the meetings on Monday morning to get the latest news
- Try to implement your task from an existing example
- Get in touch with the experts
- This will be the tool that we will be using in the next years, so give your feedback!

Get in touch here!

- Monday meetings at 9h30:
  https://indico.cern.ch/event/1022816/
- Mattermost channel:
  https://mattermost.web.cern.ch/alice/channels/o2-analysis-challenge
- Twiki:
  https://twiki.cern.ch/twiki/bin/viewauth/ALICE/AliceO2WP14AF

Let’s get started!

Hands on instructions:
https://github.com/njacazio/AliceO2/blob/LF-workshop/Analysis/Tasks/LFTutorial/instructions.md

If you don’t have O2 locally installed:

https://github.com/njacazio/AliceO2/blob/LF-workshop/Analysis/Tasks/LFTutorial/instructions_l.md

Additional info!

Key files in AliPhysics

- Converter for Run2 data to the Run3 data format

AliPhysics/RUN3/AliAnalysisTaskAO2Dconverter.cxx

- The development for the Analysis Framework involves both O2 and AliPhysics!
- So far no expected times are available in the converted data
- Only one collision time is available (average of the 10)

Key files in O2

- Definition of basic columns (collisions, tracks, detector signals)

O2/Framework/Core/include/Framework/AnalysisDataModel.h

- This is the equivalent of the current AliESDEvent, AliESDtrack, detector response and so on

Test-bed for Run3 software

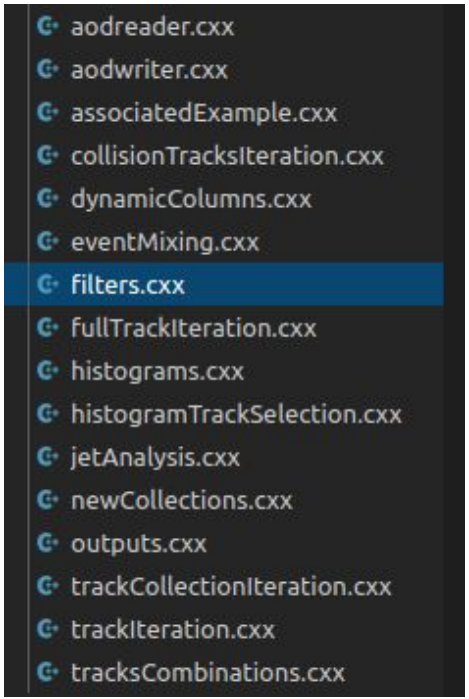
- https://github.com/njacazio/Run3Analysisvalidation
- https://github.com/njacazio/Run3Analysisvalidation/blob/master/codeLF/runtestLF.sh
- Script to convert Run2 data, run Run3 analysis, run Run2 analysis and compare the results
- In the repository are implemented examples for both LF and HF

Tutorials

- Rich list of analysis tutorials and examples are already implemented

O2/Analysis/Tutorials/src/

- This is very useful to kick-start your AF skills and start analyzing data in the Run3 format



Imperative vs declarative

- Analyses are organised into *tasks*. Tasks are automatically arranged in a workflow via DPL.

Tasks are divided in two parts:

- **Imperative**:
  - » focuses on what the program should accomplish e.g. fill a histogram
- **Declarative**:
  - » focuses on how the program should achieve the result e.g. loop on all tracks and compute invariant masses

https://www.differencebetween.com/difference-between-declarative-and-vs-imperative-programming/

Declarative

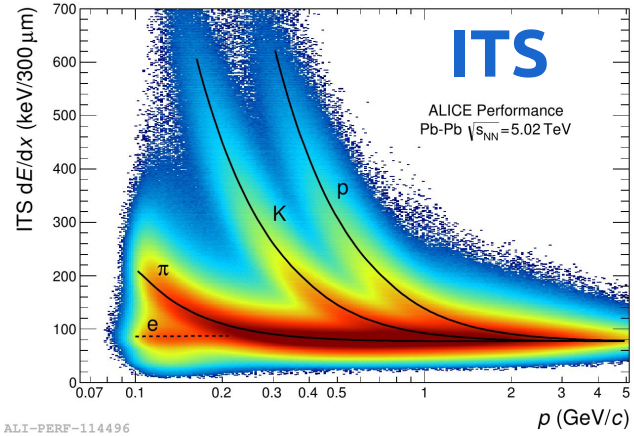
- Here are defined:
  - Outputs objects, filters (on events/tracks), partitions (of tables), simple mapping operations
- This is somehow the equivalent/extension of the ***UserCreateOutputObjects*** in AliPhysics Analysis Tasks where one can create output (TH1, TTree), cuts (AliESDtrackCuts) et cetera
- The more you can describe your analysis as declarative, the better optimisation chances

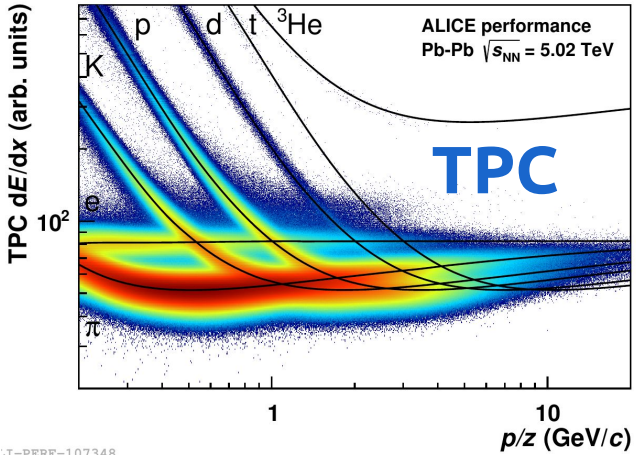
Imperative!

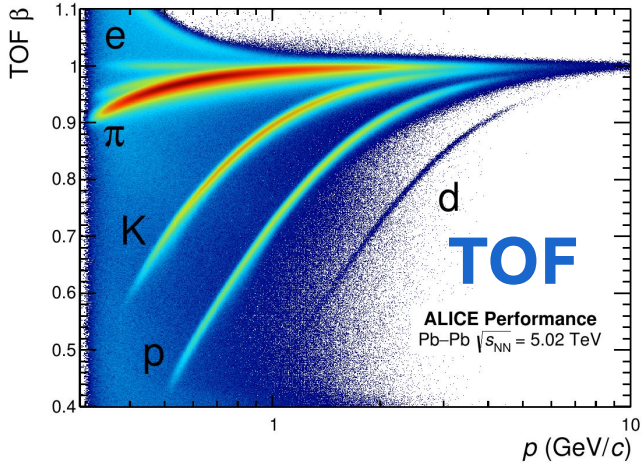
- Here are defined
  - The instructions to obtain the needed observables
- This is somehow the equivalent of the ***UserExec*** in AliPhysics Analysis Tasks
- This part is meant to simplify porting from AliPhysics and in general to provide a more Object Oriented look and feel to the user

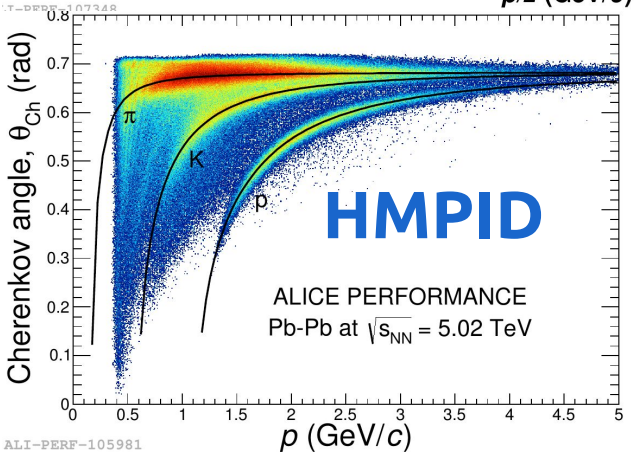
Previously on the Run2

PID in Run2


ITS
ALICE Performance
Pb-Pb $\sqrt{s_{NN}}$=5.02 TeV
ITS dE/dx (keV/300 μm)
p (GeV/c)
π
K
p
e
ALI-PERF-114496


TPC
ALICE performance
Pb-Pb $\sqrt{s_{NN}}$ = 5.02 TeV
TPC dE/dx (arb. units)
p/z (GeV/c)
p
d
t
^{3}He
K
e
π
10^2


TOF
ALICE Performance
Pb–Pb $\sqrt{s_{NN}}$ = 5.02 TeV
TOF β
p (GeV/c)
e
π
K
p
d


HMPID
ALICE PERFORMANCE
Pb-Pb at $\sqrt{s_{NN}}$ = 5.02 TeV
Cherenkov angle, θ_{Ch} (rad)
p (GeV/c)
π
K
p
ALI-PERF-105981

PID in Run2: where

In Run2 the access to the PID information was handled via the PID response

- The AliPIDResponse task is the main PID handler (access expected values, resolutions, Nsigmas)
- Every detector has the implementation of its response in a dedicated task e.g. AliITSPIDResponse, these contains all the available parametrizations
- Parameters of the response taken from the OADB: run by run parametrization with different passes (e.g. different parametrization pass1 vs pass2)

PID in Run2: workflow

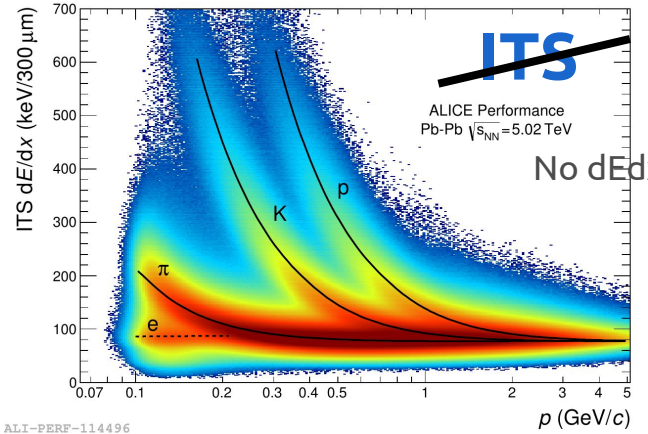
The PID task AliAnalysisTaskPIDResponse has to be run in front of the analysis tasks to prepare the response values for later usage

```
// PID Response Configuration
gROOT->LoadMacro("$ALICE_ROOT/ANALYSIS/macros/AddTaskPIDResponse.C");
// AliAnalysisTaskPIDResponse *AddTaskPIDResponse(Bool_t isMC=kFALSE,
//             Bool_t autoMCesd=kTRUE,
//             Bool_t tuneOnData=kTRUE, TString recoPass="2",
//             Bool_t cachePID=kFALSE, TString detResponse="",
//             Bool_t useTPCEtaCorrection = kTRUE,/*Please u
se default value! Otherwise splines can be off*/
//             Bool_t useTPCMultiplicityCorrection = kTRUE,/*Please use default value! Otherwise splines can be off*/
//             Int_t  recoDataPass = -1)
gROOT->ProcessLine("AddTaskPIDResponse(kFALSE, kTRUE, kTRUE, \"1\")");
```

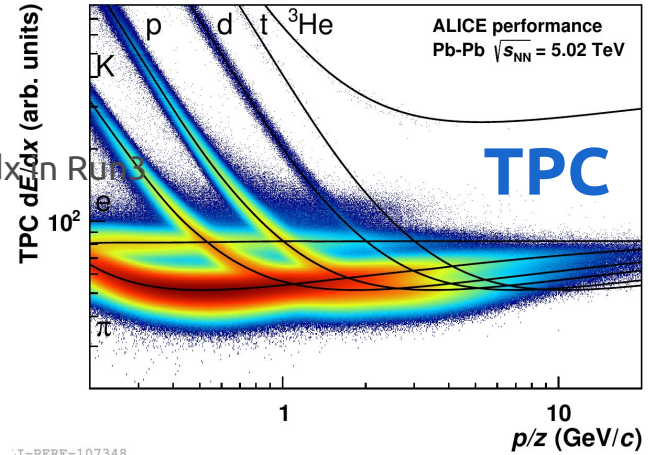
The task configuration has several handles that can be configured by the user

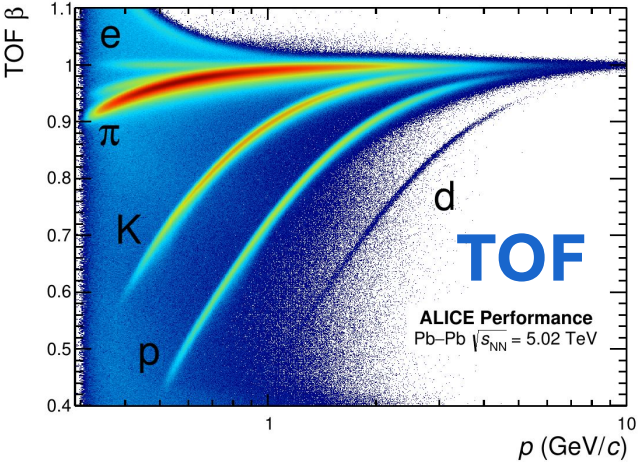
In Run3

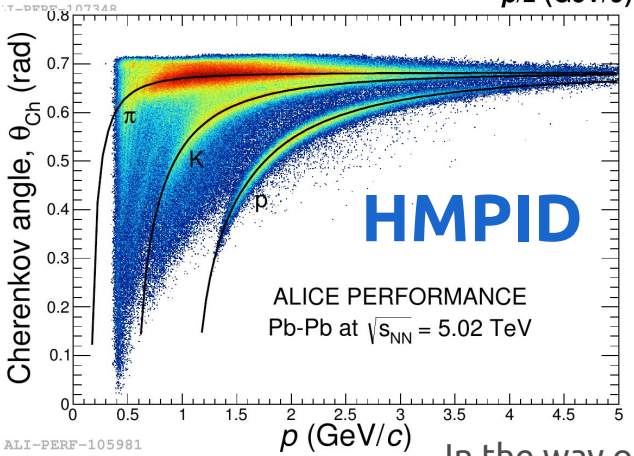
PID in O2: status



No dEdx in Run 3







In the way of being implemented

Status of PID in O2

The PID in O2 is already in production for TPC and TOF with some differences with respect to Run2

- dE/dx parametrization in TPC relies on a simple Bethe Bloch parametrization and not on the splines
- The TOF response is based on the expected times computed under the pion hypothesis and the collision time is unique in O2 (*p*-dependent in Run2)
- Caveat! A 1:1 comparison of Run2 and O2 performance is by design not straightforward as values are recomputed for the sake of performance

Analyses in O2

Analyses in O2 rely on the information available in tables: each column is a feature ->AnalysisDataModel.h

- Collisions: *PosX*, *PosY*, *PosZ*, *CollisionTime*, ...
- Tracks: *Signed1Pt*, *TPCSignal*, *TOFSignal*, …

```
200
201   void process(const o2::aod::Collision& collision, const o2::aod::Tracks& tracks)
202   {
203     eventCount->Fill(0);
204     histograms.fill("collision/collisionX", collision.posX());
205     histograms.fill("collision/collisionY", collision.posY());
206     histograms.fill("collision/collisionZ", collision.posZ());
207
208     int nTracks(0);
209     for (const auto& track : tracks) {
210       nTracks++;
211     }
```

The number of tables that a particular task needs are defined at compile time, tasks see only the information they subscribe to

What's in O2

Usage https://aliceo2group.github.io/analysis-framework/pid.html

A structure similar to the one in AliRoot can be found in O2

- PIDResponse.h (similar to AliPIDResponse) contains the definitions of the columns for each detector e.g. expected values, expected resolutions, Nsigma, TOF beta et cetera
- Tasks that need the PID information should include it to be able to use the PID response
- The values in the PID tables are filled via dedicated tasks (one per detector or per method) e.g. pidTOF.cxx, pidTPC.cxx

Parametrization and customization

The main difference with respect to Run2 is the parametrization of the response

- The parametrization transforms track features to PID features i.e. expected values, expected resolutions and separation (Nsigma)
- Tasks are unaware of the parametrization and they receive it from the CCDB database
- In principle the user can produce his own parametrization and use it in his task (instead of using some post calibration)

Structure in O2

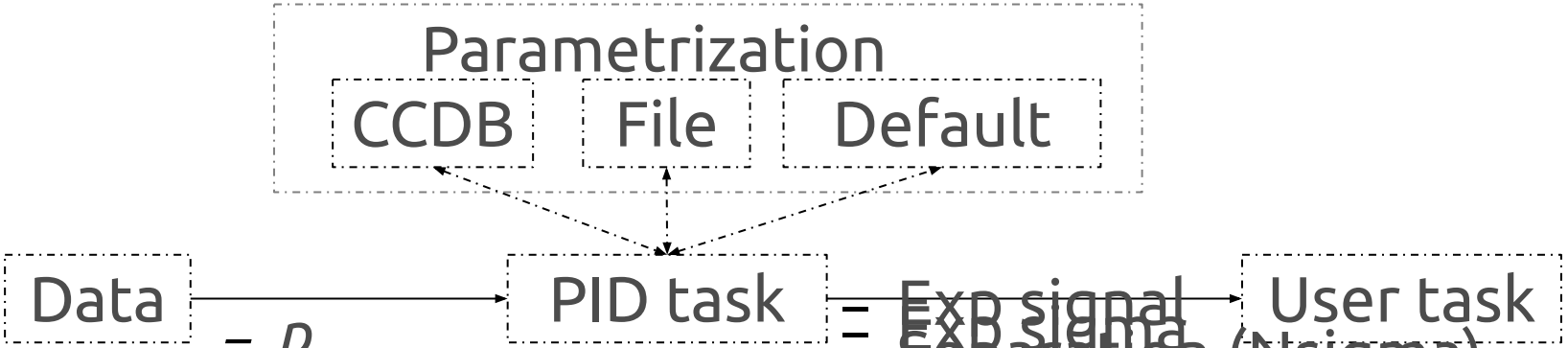
Parametrizations are handled in selfcontained classes

- Independent objects that can be saved in root files
- The correct parametrization object is selected based on the timestamp of the collision (w.r.t Run2 a finer granularity is possible!)

The source of the parametrization can be decided by the user when configuring the task

- Ad hoc parametrizations specific to particles of interest can be used
- Not limited by the version of the software

The PID workflow



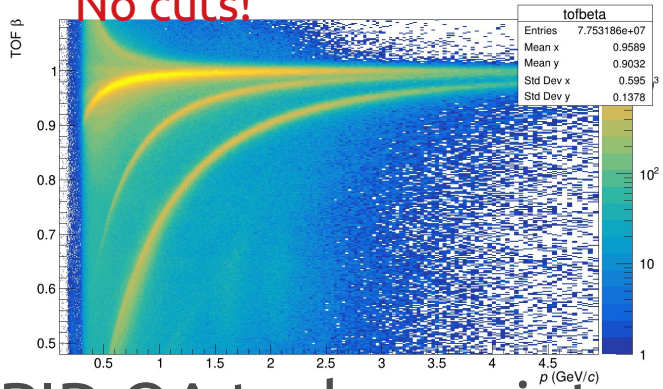
- p
- dE/dx
- TOF
- Event time
- ...

In practice:

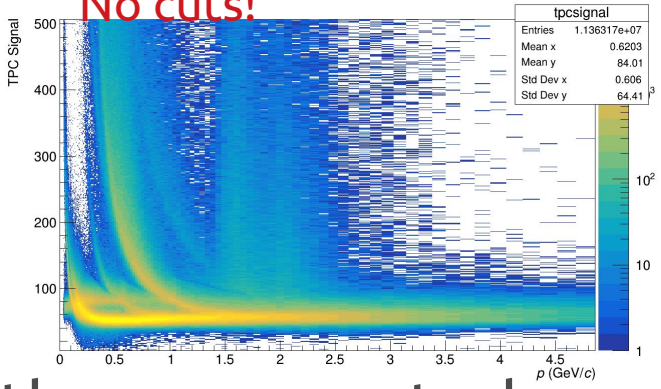
o2-analysis-your-task -b --aod-file AO2D.root |
o2-analysis-pid-tof -b

PID QA integrated in the response

No cuts!



No cuts!



PID QA tasks are integrated in the response tasks

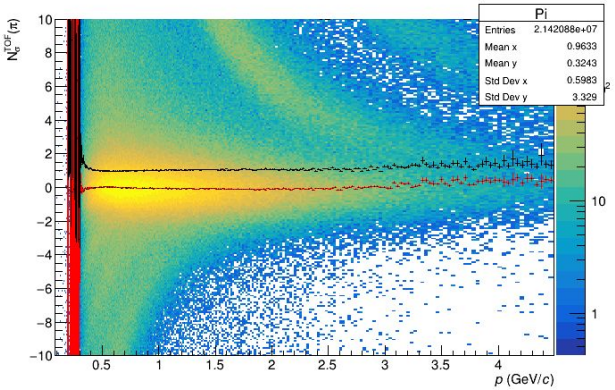
o2-analysis-pid-tof -b --aod-file AO2D.root --add-qa 1

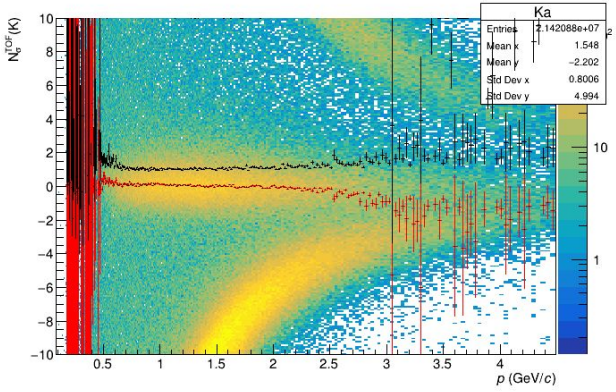
o2-analysis-pid-tpc -b --aod-file AO2D.root --add-qa 1

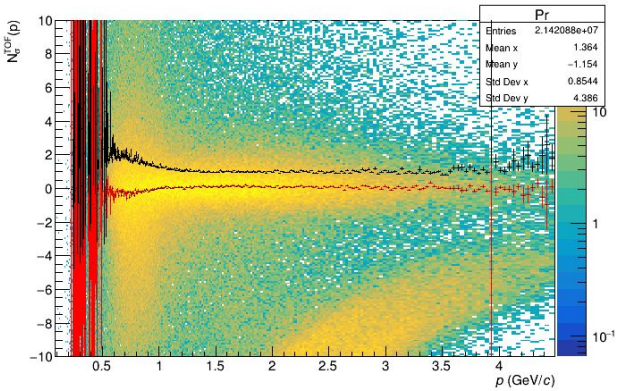
These produce plots for all particle species (from electrons to alpha particles)

PID QA integrated in the response

No cuts are (yet) used in the selection of tracks!

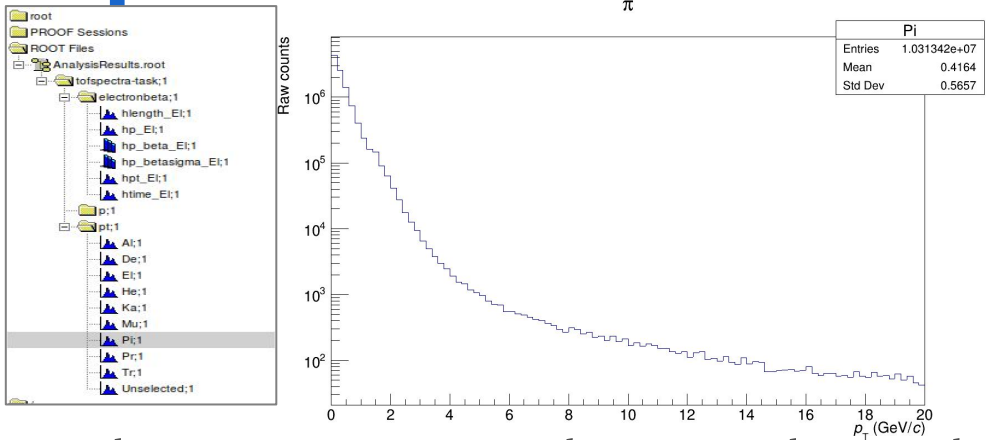






From converted Pb-Pb 5.02 TeV

The spectra tasks



Spectra tasks are in O2: selection based on the standard Nsigma cut

o2-analysis-spectra-tof -b --aod-file AO2D.root | o2-analysis-pid-tof -b |o2-analysis-trackselection -b | o2-analysis-trackextension -b

The same is available for TOF

Establishing the correction

The correction procedure is still to be sorted out within the framework

Efficiency correction can be used directly at runtime in the task by getting it via CCDB

A task handling the MC information is also in O2:

o2-analysis-mc-spectra-efficiency --aod-file AO2D.root -b|o2-analysis-trackselection -b|o2-analysis-trackextension -b

Summary

- This is a very quick (and partial!) overview of the PID
- Development discussed on Monday mornings at the WP4 + WP14 Meeting
- It’s the right time to move your analysis to O2 as feedback is fundamental
- The PWG-LF is in a privileged position for what concerns the input for the requirements of the PID
- Feedback is useful to shape the future of the framework and avoid shortcomings
- Get in touch!

More info!

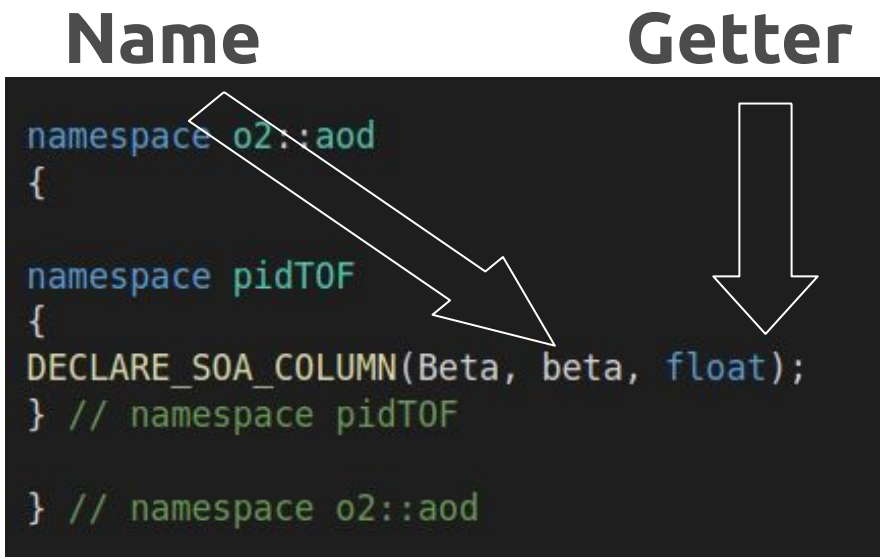
Defining new tables

- In AliPhysics tasks can produce output used by other tasks (centrality, physics selection et cetera)
- This can also be done in the new AF
- New tables that are analysis specific can be defined e.g. HF decay vertices, PID response
- Such task produces new tables and these can be used as input in later tasks

```
WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>("mytask"),
    adaptAnalysisTask<MyTask>("mytask2")
    };
}
```

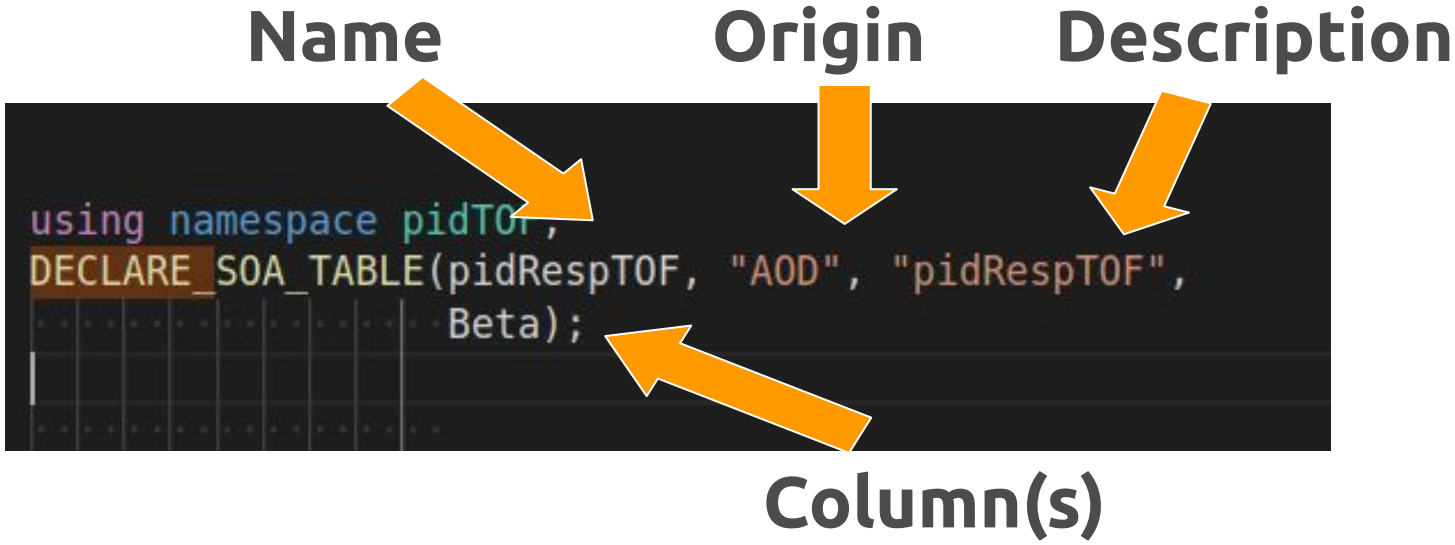
Defining new tables

- **First:** define the column properties
- Give it a namespace for better management



Defining new tables

- **Second:** define the table columns
- Give it a namespace for better management



Defining new tables

- **Third:** fill the table in the producer task
- Give it a namespace for better management

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

struct pidTOFTask {
  Produces<aod::pidRespTOF> tofpid;

  void process(aod::Collision const& collision, soa::Join<aod::Tracks, aod::TracksExtra> const& tracks)
  {
    for (auto i : tracks) {
      tofpid(i.length() / (i.tofSignal() - collision.collisionTime()) / kCSPEED);
    }
  }
};
```

Using new tables

- Now the new table can be used in your task!

```
struct MyTask {
  OutputObj<TH2F> hbeta{TH1F("hbeta", "beta", 100, 0., 2., 100, 0., 2.)};

  void process(soa::Join<aod::Tracks, aod::pidRespTOF> const& tracks)
  {
    for (auto i : tracks) {
      hbeta->Fill(i.p(), i.beta());
    }
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<pidTOFTask>("pidTOF-task"),
    adaptAnalysisTask<MyTask>("mytask")};
}
```

Rationale

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

- Using arrow tables https://arrow.apache.org/
- Cross-language development platform for in-memory data
- Columnar memory layout
- Native vectorized optimization of analytical data processing
- Supports zero-copy reads for lightning-fast data access
- Implementation in O2 developed by Giulio and others

Key files in AliPhysics

- Converter for Run2 data to the Run3 data format

AliPhysics/RUN3/AliAnalysisTaskAO2Dconverter.cxx

- The development for the Analysis Framework involves both O2 and AliPhysics!
- So far no expected times are available in the converted data
- Only one collision time is available (average of the 10)

Key files in O2

- Definition of basic columns (collisions, tracks, detector signals)

O2/Framework/Core/include/Framework/AnalysisDataModel.h

- This is the equivalent of the current AliESDEvent, AliESDtrack, detector response and so on

Test-bed for Run3 software

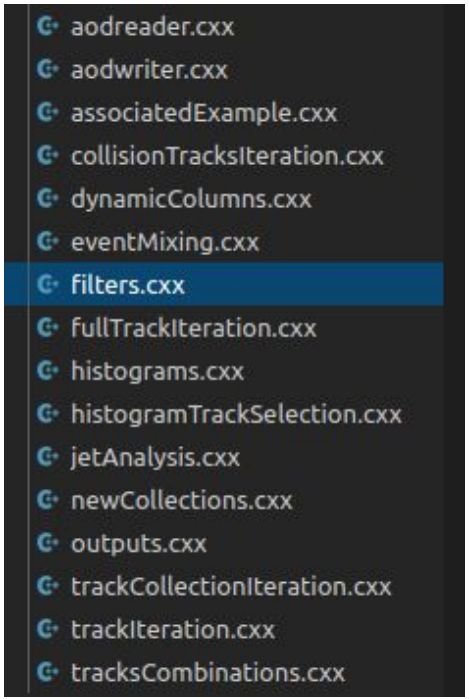
- https://github.com/njacazio/Run3Analysisvalidation
- https://github.com/njacazio/Run3Analysisvalidation/blob/master/codeLF/runtestLF.sh
- Script to convert Run2 data, run Run3 analysis, run Run2 analysis and compare the results
- In the repository are implemented examples for both LF and HF

Tutorials

- Rich list of analysis tutorials and examples are already implemented

O2/Analysis/Tutorials/src/

- This is very useful to kick-start your AF skills and start analyzing data in the Run3 format



Imperative vs declarative

- Analyses are organised into *tasks*. Tasks are automatically arranged in a workflow via DPL.

Tasks are divided in two parts:

- **Imperative**:
  - » focuses on what the program should accomplish e.g. fill a histogram
- **Declarative**:
  - » focuses on how the program should achieve the result e.g. loop on all tracks and compute invariant masses

https://www.differencebetween.com/difference-between-declarative-and-vs-imperative-programming/

Declarative

- Here are defined:
  - » Outputs objects, filters (on events/tracks), partitions (of tables), simple mapping operations
- This is somehow the equivalent/extension of the ***UserCreateOutputObjects*** in AliPhysics Analysis Tasks where one can create output (TH1, TTree), cuts (AliESDtrackCuts) et cetera
- The more you can describe your analysis as declarative, the better optimization chances

Imperative!

- Here are defined
  - The instructions to obtain the needed observables
- This is somehow the equivalent of the ***UserExec*** in AliPhysics Analysis Tasks
- This part is meant to simplify porting from AliPhysics and in general to provide a more Object Oriented look and feel to the user

Input data

- Input data is automatically defined depending the single task needs
- Task only access the information they need (*divide et impera*)
- No overhead
- Tasks can run on collisions only, on single tracks, on all tracks in a timeframe and so on
- Input data is organized into split (but linked) tables

Input data (1)

Each column is an vertex property:

- Vertex position
- Covariance matrix
- Chi2
- Contributors
- T0 info

Table of collisions

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Input data (2)

Each column is an track property:

- Momentum
- Track parameters
- Track type
- …

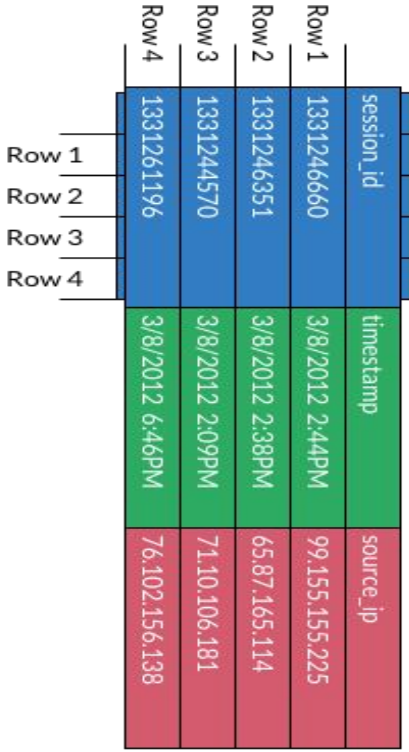


Table of **Collisions**

timestamp	source_ip
3/8/2012 2:44PM	99.155.155.225
3/8/2012 2:38PM	65.87.165.114
3/8/2012 2:09PM	71.10.106.181
3/8/2012 6:46PM	76.102.156.138

Table of **Tracks** ”1 per row”

Input data (3)

Each column is an track property:

- Chi2
- TPC/TOF signal
- …

Table of **Collisions**

	timestamp	source_ip
Row 1	3/8/2012 2:44PM	99.155.155.225
Row 2	3/8/2012 2:38PM	65.87.165.114
Row 3	3/8/2012 2:09PM	71.10.106.181
Row 4	3/8/2012 6:46PM	76.102.156.138

Table of **Tracks** ”1 per row”

	session_id	timestamp
Row 1	1331246660	3/8/2012 2:44PM
Row 2	1331246351	3/8/2012 2:38PM
Row 3	1331244570	3/8/2012 2:09PM
Row 4	1331261196	3/8/2012 6:46PM

Table of **TracksExtra**

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Writing our first task

- Writing a first ”hello word” task that produces a simple plot is very easy
- a) give your task a memorable name and create a file in O2/Analysis/Tasks/

Writing our first task

- Writing a first ”hello word” task that produces a simple plot is very easy
- a) give your task a memorable name and create a file in O2/Analysis/Tasks/
- b) add your task to the Cmake file O2/Analysis/Tasks/CmakeLists.txt and define there the workflow name
- c) implement your task
- d) enjoy your new plot

https://is.gd/sQzWGn

Writing our first task

- Disclaimer: No includes are shown here refer to other tasks in the directory

```cpp
using namespace o2;
using namespace o2::framework;

struct MyTask {
  // primary vertex position
  OutputObj<TH1F> hvtxp_x_out{TH1F("hvertexx", "x primary vtx", 100, -10., 10.)};

  void process(aod::Collision const& collision)
  {
    hvtxp_x_out->Fill(collision.posX());
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>("mytask")};
}
```

task

workflow

Writing our first task

- Disclaimer: No includes are shown here refer to other tasks in the directory

```cpp
using namespace o2;
using namespace o2::framework;

struct MyTask {
  // primary vertex position
  OutputObj<TH1F> hvtxp_x_out{TH1F("hvertexx", "x primary vtx", 100, -10., 10.)};

  void process(aod::Collision const& collision)
  {
    hvtxp_x_out->Fill(collision.posX());
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>("mytask")};
}
```

Writing our first task

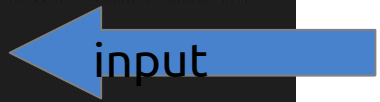
- Disclaimer: No includes are shown here refer to other tasks in the directory

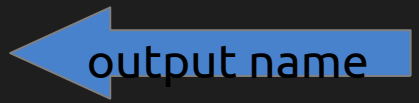
```cpp
using namespace o2;
using namespace o2::framework;

struct MyTask {
  // primary vertex position
  OutputObj<TH1F> hvtxp_x_out{TH1F("hvertexx", "x primary vtx", 100, -10., 10.)};

  void process(aod::Collision const& collision)
  {
    hvtxp_x_out->Fill(collision.posX());
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>("mytask")};
}
```





Adding our first task and run it

- Now that we wrote the task we need to include it in the building process
- In O2/Analysis/Tasks/CmakeLists.txt
- If your task is ”MyTask.cxx”

- And rebuild O2
- Then you should be able to run your workflow

Side-notes (1)

- Tasks “produce” objects e.g. TH1F saved in the *AnalysisResults.root* file under their output name
- Worfklows are the equivalent of the RunTask and the *adaptAnalysisTask* is the equivalent of the *AddTask* in AliPhysics
- As in the current RunTask Workflows can handle several tasks that are run in cascade (as for the physics selection b

```
WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>("mytask"),
    adaptAnalysisTask<MyTask>("mytask2")
    };
}
```

Side-notes (2)

- ***process*** methods can handle several input

- Tables can be joined if they have the same length
- Joined tables are equivalent to standard tables, just with more columns!
- This is to analyze several input and mix separate columns while keeping the processed and unused information to the bare bone

Side-notes (3)

- You can convert your own data from ESDs using
- https://github.com/njacazio/Run3Analysisvalidation/blob/master/codeHF/convertAO2D.C
- To see it at work refer to the script:
- https://github.com/njacazio/Run3Analysisvalidation/blob/a29f31d07925424e65e5f2ce04c96fb4d0d253de/codeLF/runtestLF.sh#L18-L30

Defining new tables

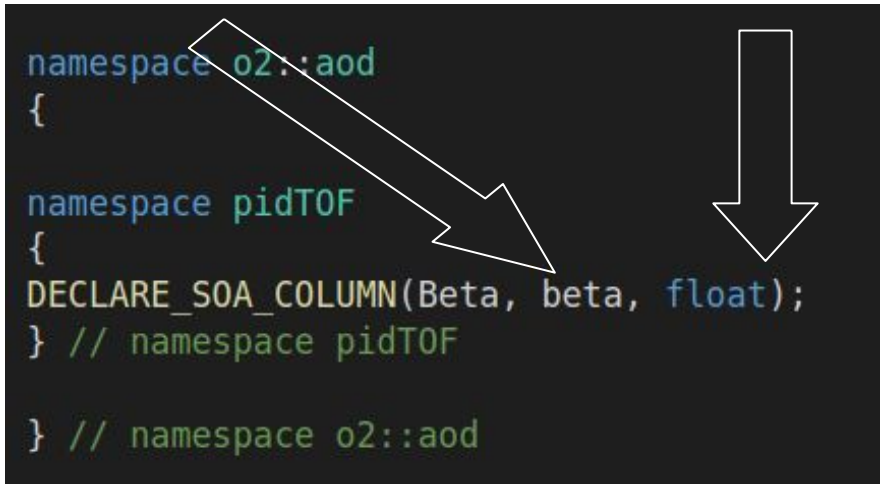
- In AliPhysics tasks can produce output used by other tasks (centrality, physics selection et cetera)
- This can also be done in the new AF
- New tables that are analysis specific can be defined e.g. HF decay vertices, PID response
- Such task produces new tables and these can be used as input in

```
WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>("mytask"),
    adaptAnalysisTask<MyTask>("mytask2")
    };
}
```

Defining new tables

- **First:** define the column properties
- Give it a namespace for better management

Name **Getter** **Type**



```
namespace o2::aod
{

namespace pidTOF
{
DECLARE_SOA_COLUMN(Beta, beta, float);
} // namespace pidTOF

} // namespace o2::aod
```

Defining new tables

- **Second:** define the table columns
- Give it a namespace for better management

Name **Origin** **Description**

Column(s)

Defining new tables

- **Third:** fill the table in the producer task
- Give it a namespace for better management

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

struct pidTOFTask {
  Produces<aod::pidRespTOF> tofpid;

  void process(aod::Collision const& collision, soa::Join<aod::Tracks, aod::TracksExtra> const& tracks)
  {
    for (auto i : tracks) {
      tofpid(i.length() / (i.tofSignal() - collision.collisionTime()) / kCSPEED);
    }
  }
};
```

Using new tables

- Now the new table can be used in your task!

```
struct MyTask {
  OutputObj<TH2F> hbeta{TH1F("hbeta", "beta", 100, 0., 2., 100, 0., 2.)};

  void process(soa::Join<aod::Tracks, aod::pidRespTOF> const& tracks)
  {
    for (auto i : tracks) {
      hbeta->Fill(i.p(), i.beta());
    }
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<pidTOFTask>("pidTOF-task"),
    adaptAnalysisTask<MyTask>("mytask")};
}
```

Dynamic and static columns

- Table columns can be dynamic and static
- **Dynamic**: transient, cached only when used
- **Static**: kept in memory
- e.g. p_T and p both carry the same information and you can obtain one from the other
- This is done at the column declaration level
- Not filled in constructor

Filters

- The ABC of every analysis is the event/track/candidate selection
- Filters fall in the declarative part → high performance gain
- Filters only work on static columns (for now)
- All filters are defined as a function returning true or

```
struct FilterTask {

  Filter ptFilterlow = aod::track::signed1Pt < 1.0f / 0.5;
  Filter ptFilterhigh = aod::track::signed1Pt > 1.0f / 1.5;

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

Types of NUCLEX analyses from a framework perspective

1. Direct spectra with TPC and TOF: antideuteron, anti-^{3}He,..
   - primary tracks with appropriate cuts
   - dca template fit for secondary rejection
   - full glory PID: TPC dE/dx and TOF
   - centrality and multiplicity selection

2. hyper-nuclei
   - needs in addition also complicated secondary vertex finding in several decays channels (ideally in *strangeness tracking* mode)

3. absorption
   - needs in addition TOF information on other matched clusters in window, track-to-hit residual

4. flow → should be simple corollary of CF efforts

5. exotica → skip for the time being

Types of NUCLEX analyses from a framework perspective

1. Direct spectra with TPC and TOF: antideuteron, anti-^{3}He,..
   - primary tracks with appropriate cuts
   - dca template fit for secondary rejection
   - full glory PID: TPC dE/dx and TOF
   - centrality and multiplicity selection

→ The plan is to implement step-by-step the needed ingredients!

2. hyper-nuclei
   - needs in addition also complicated secondary vertex finding in several decays channels (ideally in *strangeness tracking* mode)

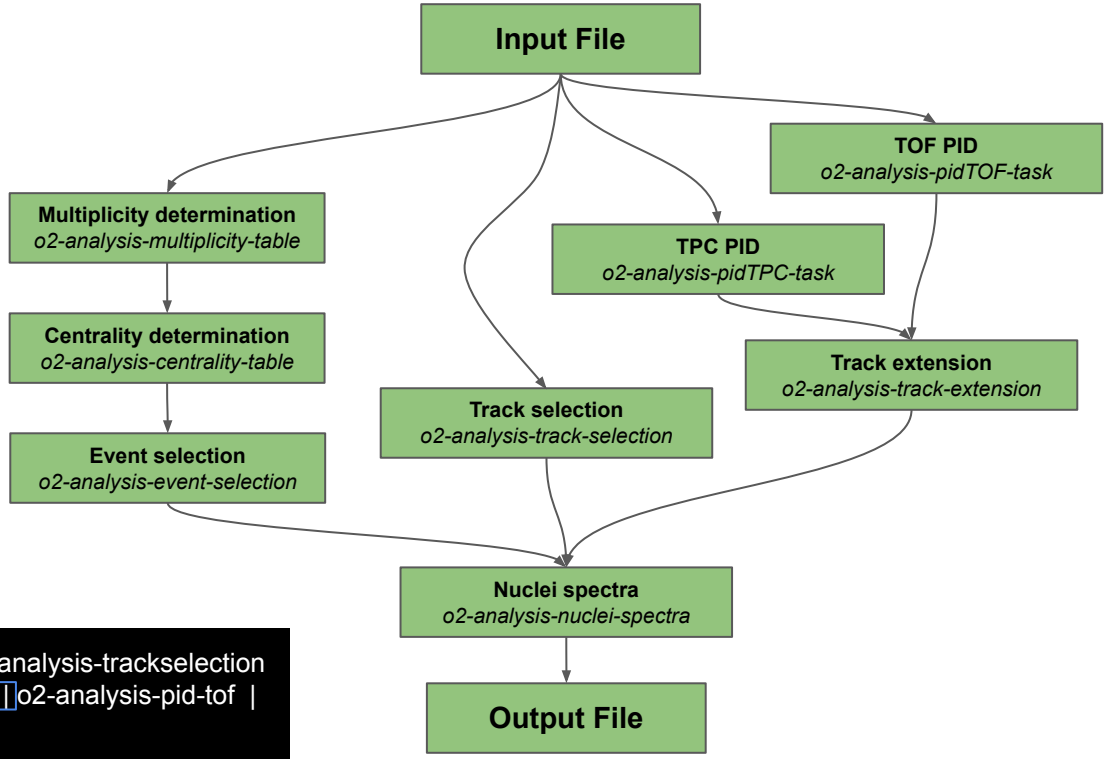
3. absorption
   - needs in addition TOF information on other matched clusters in window, track-to-hit residual

4. flow → should be simple corollary of CF efforts

5. exotica → skip for the time being

Structure of the Analysis Framework

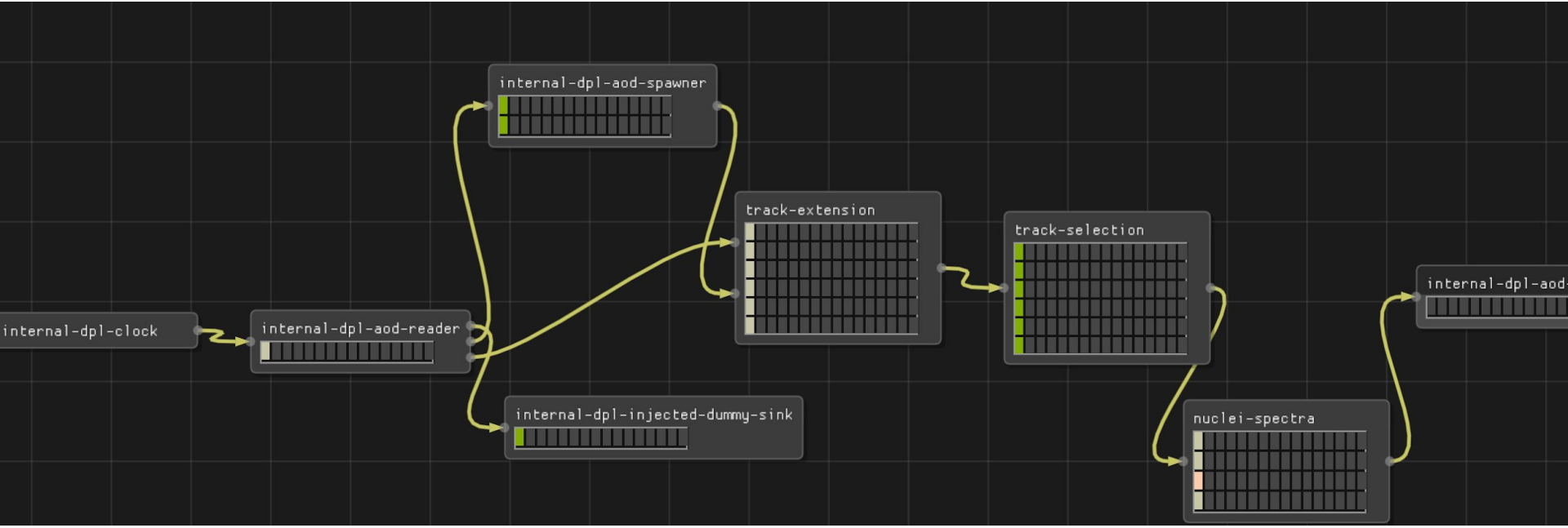
- The analysis task is part of a **Workflow**
- Data are organised into **Tables**
  - the table output of a task can be used as input from another
- The **order** of each element is executed according to its **dependencies** and to the **tables**
- This is the **command**:



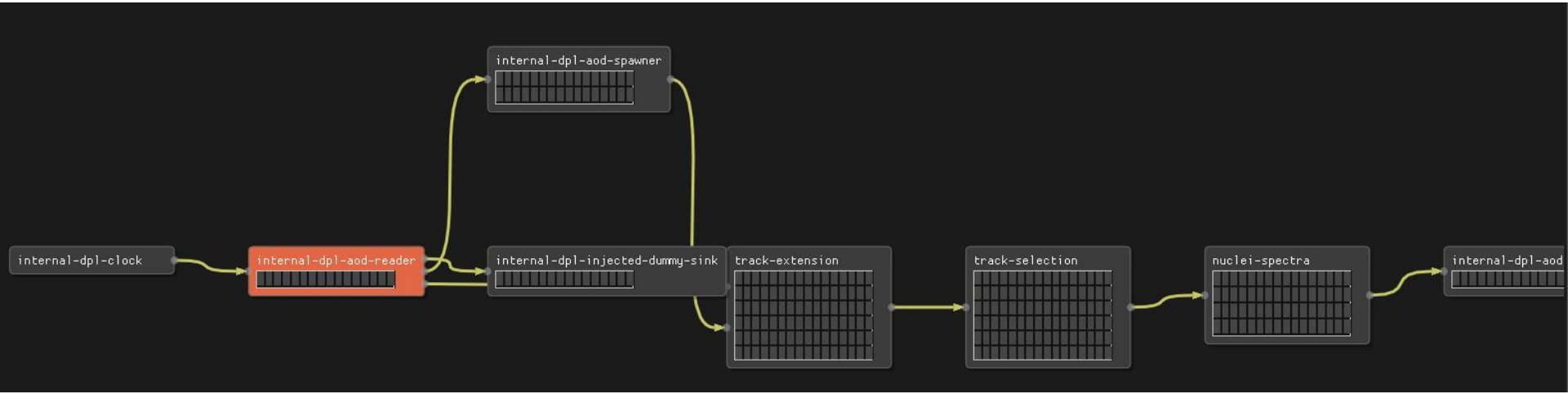

```
user dir $ o2-analysis-nuclei-spectra --aod-file pp.root | o2-analysis-trackselection
| o2-analysis-trackextension | o2-analysis-pid-tpc --add-qa 1| o2-analysis-pid-tof |
o2-analysis-centrality-table | o2-analysis-event-selection |
o2-analysis-multiplicity-table o2-analysis-timestamp
```

- The highlighted parts are used to: select the input file, add an output with QA, provide needed information to other tasks

Job execution (1)



Job execution (2)



During the last weeks, jobs were often hanging in local execution on my old MacBook.
No clear pattern what caused this (seemed erratic).

→ Could be circumvented with memory rate limit command (--aod-memory-rate-limit 1).
→ Not needed anymore with latest version (since yesterday?)!

Job execution (3)

```
[8218:internal-dpl-aod-reader]: [09:13:49][ERROR] Failed binding socket internal-dpl-aod-reader.from_internal-dpl-aod-reader_to_internal-dpl-injected-dummy-sink[0].push, reason: Interrupted system call
[8218:internal-dpl-aod-reader]: [09:13:49][ERROR] failed to attach channel from_internal-dpl-aod-reader_to_internal-dpl-injected-dummy-sink[0] (bind)
[8218:internal-dpl-aod-reader]: [09:13:49][ERROR] 1 of the binding channels could not initialize. Initial configuration incomplete.
[8218:internal-dpl-aod-reader]: [09:13:49][STATE] Exiting FairMQ state machine
[8218:internal-dpl-aod-reader]: [09:13:49][ERROR] Unhandled std::exception reached the top of main: 1 of the binding channels could not initialize. Initial configuration incomplete., device shutting down.
```

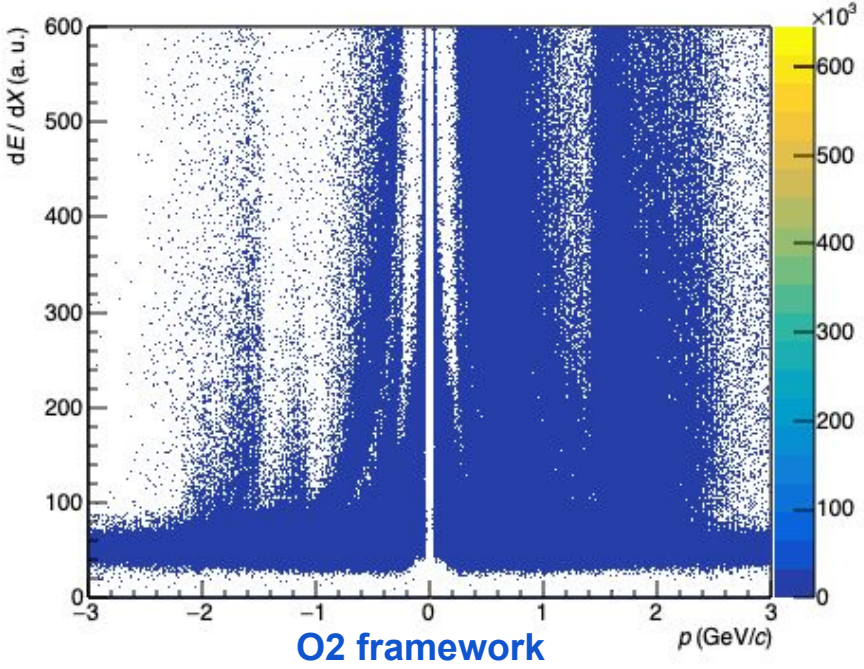
...

```
[INFO] Dumping used configuration in dpl-config.json
[ERROR] SEVERE: Device internal-dpl-aod-reader (8218) had at least one message above severity ERROR: Failed binding socket internal-dpl-aod-reader.from_internal-dpl-aod-reader_to_internal-dpl-injected-dummy-sink[0].push, reason: Interrupted system call
[INFO] Process 8204 is exiting.
```

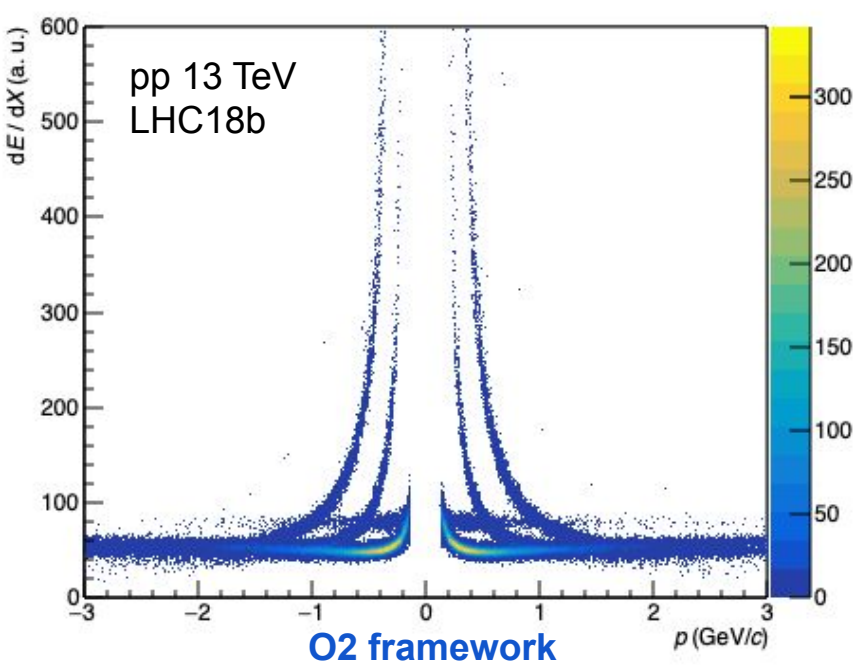
Sometimes the process breaks. This problem seems again to be erratic.

PID: dE/dx with and without default track cuts

without standard track cuts (isGlobalTrack)



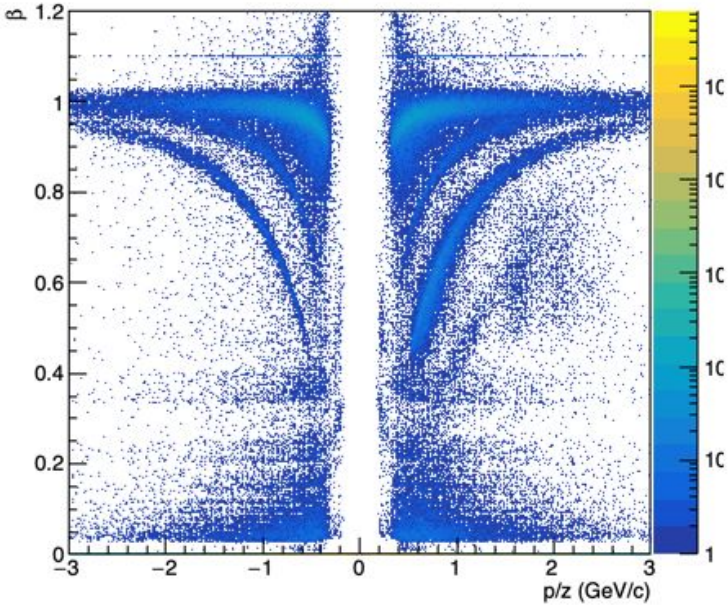
with standard track cuts (isGlobalTrack)



→ default track cuts as implemented nicely clean up the dEdx spectrum!

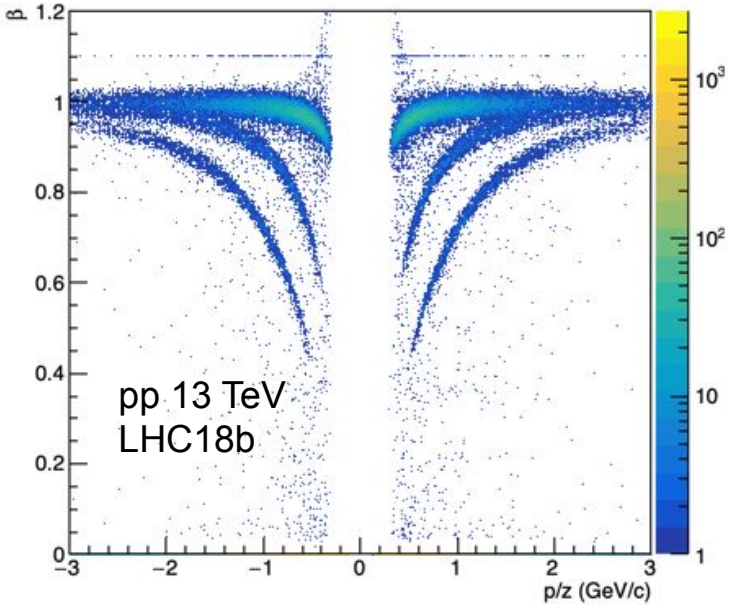
PID: TOF with and without default track cuts

without standard track cuts (isGlobalTrack)



O2 framework

with standard track cuts (isGlobalTrack)

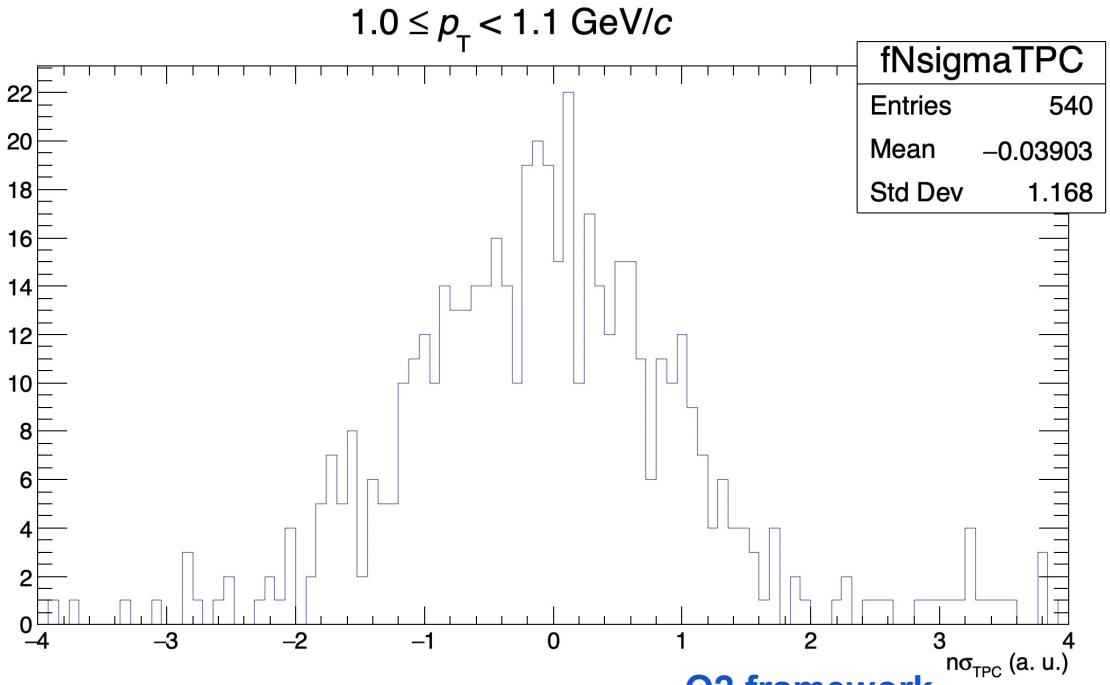


O2 framework

→ default track cuts as implemented nicely clean up the TOF spectrum!

Signal extraction

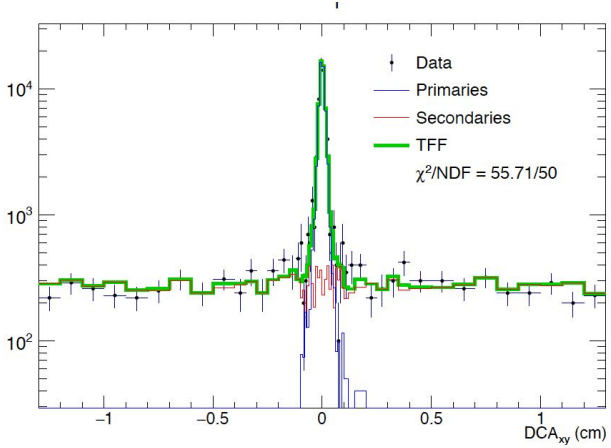
- Using extended tracks (with PID information) signal can be extracted from nσ distributions



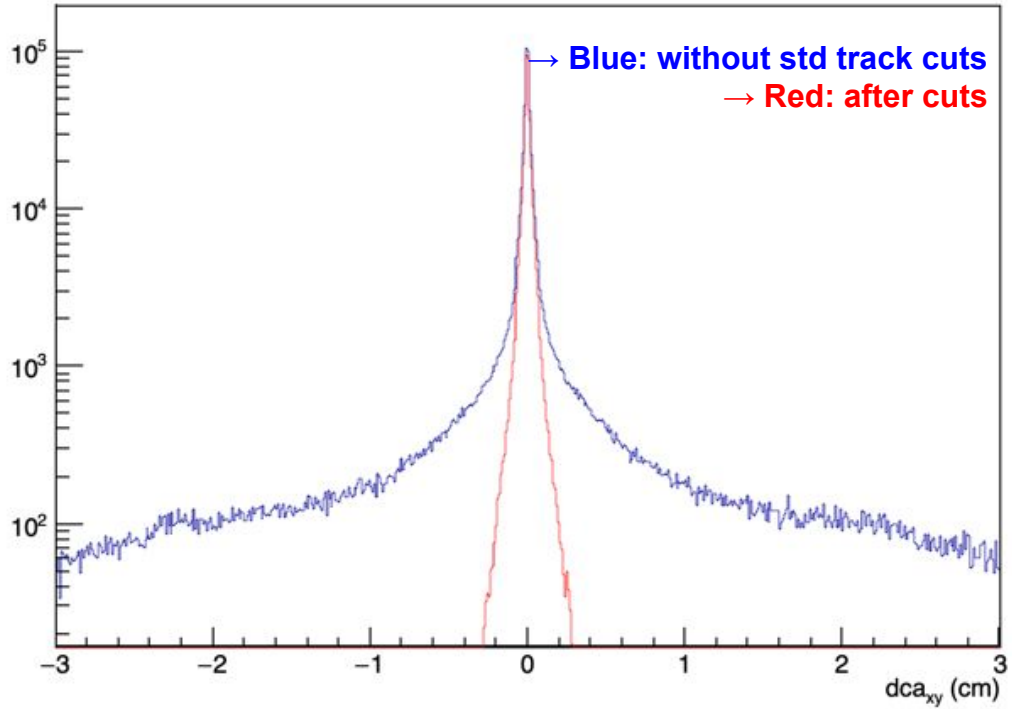
O2 framework

DCA_xy distribution

ALIPHYSICS

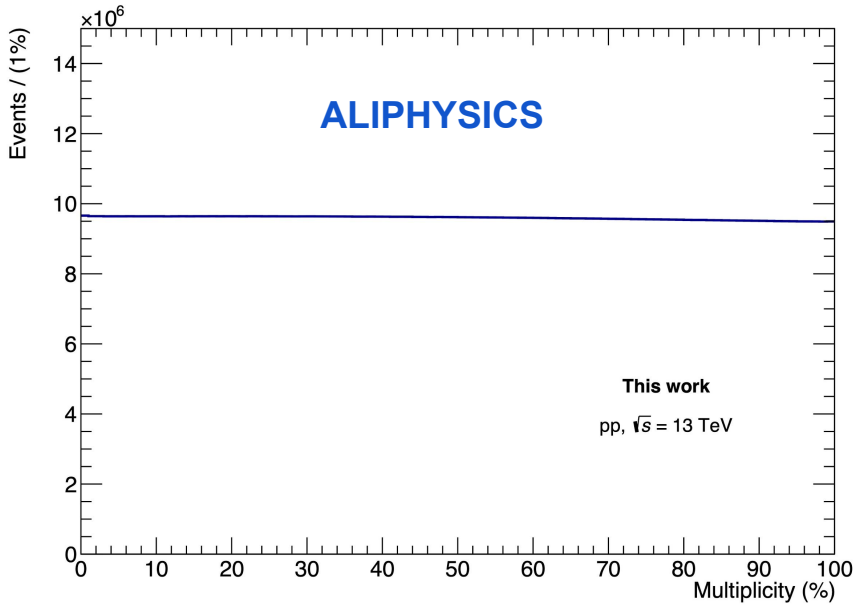


O2 framework



Centrality / Multiplicity selection

- All the tests are carried out on pp collisions @ 13 TeV
- Analyses as a function of **multiplicity**:
  1. event selection
  2. get the multiplicity value from the event
- Goal #1: get the **multiplicity distribution**
- Previously:
  1. AliMultSelectionTask
  2. AliEventCuts



Summary and next steps

→ Analysis challenge started successfully in NUCLEX (we are now also ready to give guidance to more interested analysers).

→ conversion of special productions: for LHC18qr, we have dedicated data reconstructions that contain only (anti-)3He and (anti-)Hypertriton candidates

→ It would be very good to convert these data sets:

LHC18r_filter_hypertriton_pass3
LHC18r_filter_3he_pass3
LHC18r_filter_triton_pass3

→ We should also try to refilter the corresponding MonteCarlo.

→ Add task to hyperloop?

Additional slides

Full workflow