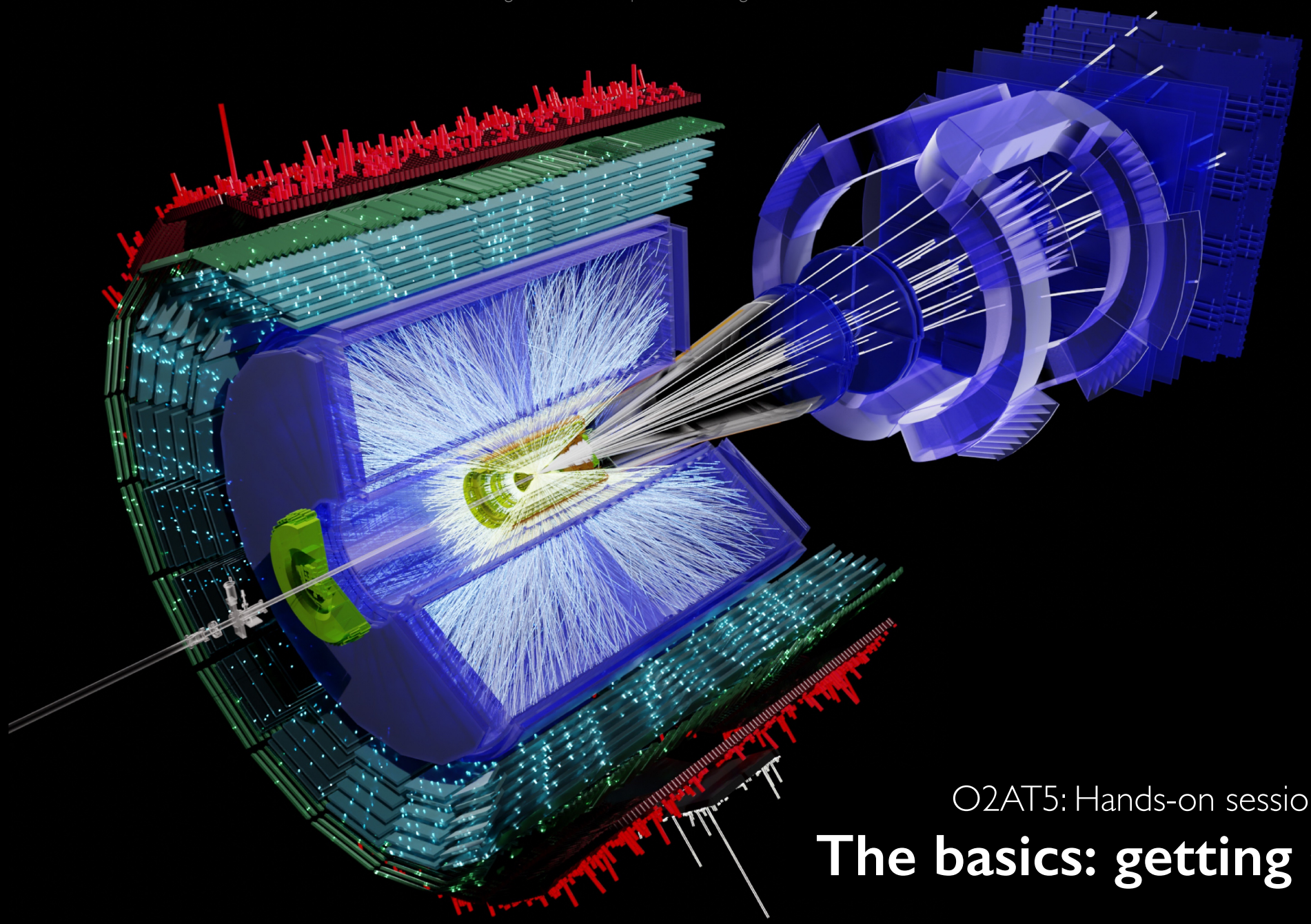
Image reference: https://zenodo.org/records/13732384



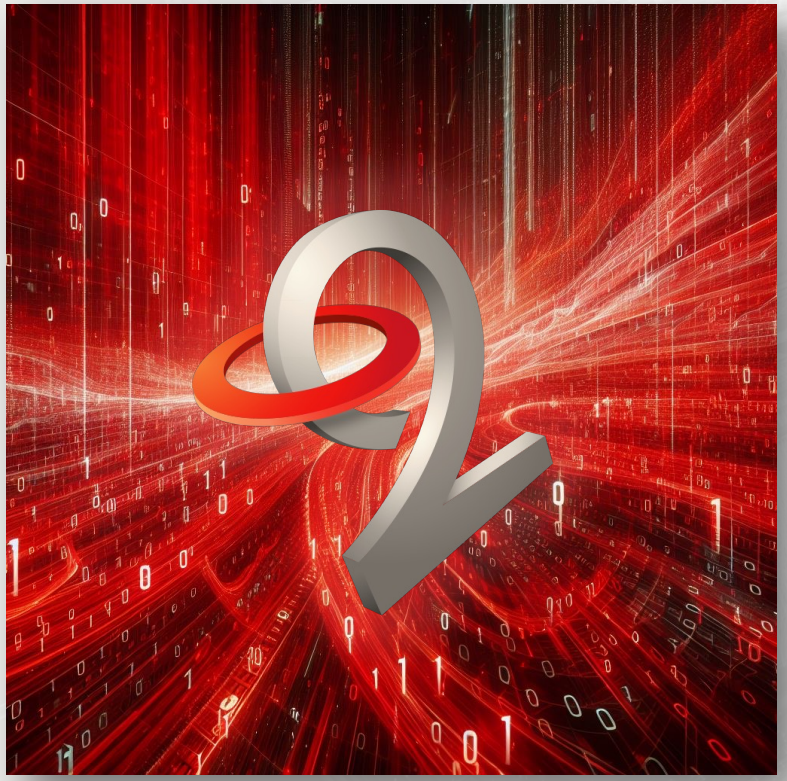
O2AT5: Hands-on session 1

The basics: getting started

The plan for this afternoon

- **First exercise: Getting started**
  - In this exercise, we'll make very simple modifications to a simple example task and learn **how to run an analysis task over real Pb-Pb data** and handle helper tasks
  - Rationale: this should be very simple and is **meant to be a 'hello world' session'**
- **Second exercise: Calculation of an efficiency**
  - In this part, we'll explore **simulated Pb-Pb data** and will introduce the concept of links between tables.
  - **Target: calculation of an efficiency for pions, kaons and protons**
  - Sets the stage for the PID talks on Tuesday morning
- **Third exercise: Two-particle correlation analysis**
  - As a last point, we will elaborate on **how to write a data model** and **how to use derived data**
  - **Target: first steps of a two-particle correlation analysis**
  - This exercise will contain an example derived data file as well as an illustration of the large gains due to that
- **For tomorrow / fourth exercise: Compile-time polymorphism**
  - Here, we'll teach you **good coding tips to design a clean and fast analysis task** without code duplication
  - This session is new and strongly recommended: suggested by users!

Data for this exercise



- Single AO2D file from Pb-Pb pass5 reconstruction
  - Download via alien_cp (alien shell: it's useful!):

```
$> alien_cp
alien:///alice/data/2023/LHC23zzh/544122/apass5/0500/o2_ctf_run0054412
2_orbit0167097184_tf0000971442_epn101/001/AO2D.root file:./AO2D-
data.root
```

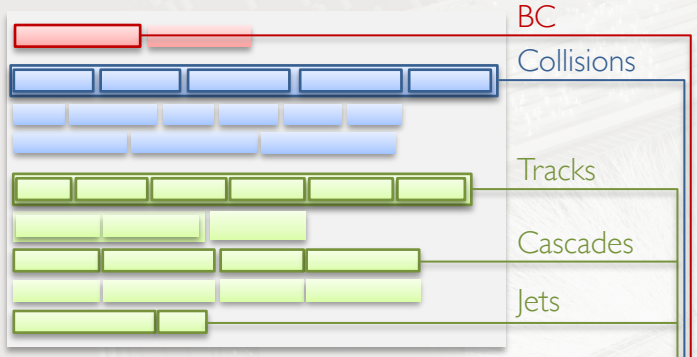
  - Download via dropbox:
  - https://www.dropbox.com/scl/fi/nob64vy35cq09bsyz0mei/AO2D-data.root?rlkey=6mg9sknm63snu7bpdatdm5exw&dl=0

Please start the download now! I will now explain the exercise in full until the end and then we do the hands-on

Enter environment: alienv enter O2Physics/latest

In a nutshell: the general analysis task structure

Data Model



Your analysis task!

struct yourAwesomeTask — Basic task definition

- **produces**<something> name; — Declares tables that may be created by this task
- **Partition** declarations — Declares partitions: new tables based on selection criteria
- **Filter** declarations — Declares filters: selection criteria
- **Output object** declarations — Declares output objects: Histograms or HistogramRegistry
- **Configurable** declarations — Declares configurables: values that can be set by user
- **init()** — Set up before processing data
- **process(●●●●●)** — Subscribe (connect to input) and process data
- **defineDataProcessing()** — Information for task -> DPL processor conversion

Examples:

- loop over tracks in each event
- loop over cascades in each event
- Correlate cascades with jets
  → do **physics**!

●●●●● = tells the framework which tables the user is interested in and which to merge / relate to one another

Very theoretical → now we will go practical! Let's run and customize our own task

Crash course: 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-propagationservice | o2-analysis-event-selection-service
```

Example task	Input file	Helper task Propagates tracks to PV	Helper task Timestamps and event selection

- All tasks have to be provided separated with a 'pipe' character ("|")
- `--aod-file` can receive an AO2D file or you can use `--aod-file @listoffiles.txt` with a list of files!
- Typically, many helper tasks are required: we will introduce you to this in the hands-on!
- This is, among other things, a consequence of the AO2D content
  - not all table information is available in the AO2D: minimalistic!
  - Some tables and columns are generated on-the-fly to minimize data storage: a strict necessity in Run 3!

- General event (centrality/multiplicity percentile) and track properties (PID values) have to be calculated!
- And beyond that: tracks are stored at their 'innermost update' in the AO2D (TracksIU)
  - Tracks to be propagated to the primary vertices by the track propagation task
  - We'll also show you this later…

A quick overview of general helper tasks (core service wagons)

a non-exhaustive list: your favorite PWG may have many more

Executable name	Function	HY wagon
o2-analysis-event-selection-service	Provides timestamps to BCs (joinable with BCs) Provides event selection tables (joinable with Collisions)	link
o2-analysis-multcenttable	Provides multiplicity tables (joinable with Collisions) Provides centrality tables (joinable with Collisions)	link
o2-analysis-qvector-table	Provides Q Vectors (joinable with Collisions)	link
o2-analysis-track-selection	Provides standard track selection criteria	link
o2-analysis-propagationservice (Aggregates propagation CPU)	Provides Tracks (tracks at the primary vertex) Provides/builds V0 information Provides/builds Cascade / KF Cascade / Tracked Cascade information	link
o2-analysis-pid-tpc-service	Provides TPC **PID** (joinable with Tracks)	link
o2-analysis-pid-tof	Provides TOF **PID** (joinable with Tracks)	link
o2-analysis-pid-its	Provides ITS **PID** (joinable with Tracks)	link
o2-analysis-lf-strangeness-tof-pid	Provides TOF **PID** for V0s, Cascades (joinable: V0Datas, CascDatas)	link

DPG talk this morning

PID talks tomorrow

Nota bene: this has recently changed and **a lot of core service tasks were merged into fewer, much more optimized tasks**. These use **less memory and less CPU** to do the same and follow as much as possible **autodetection principles**, with a true autodetection of input data type still pending some framework developments.

Configuring tasks using json files

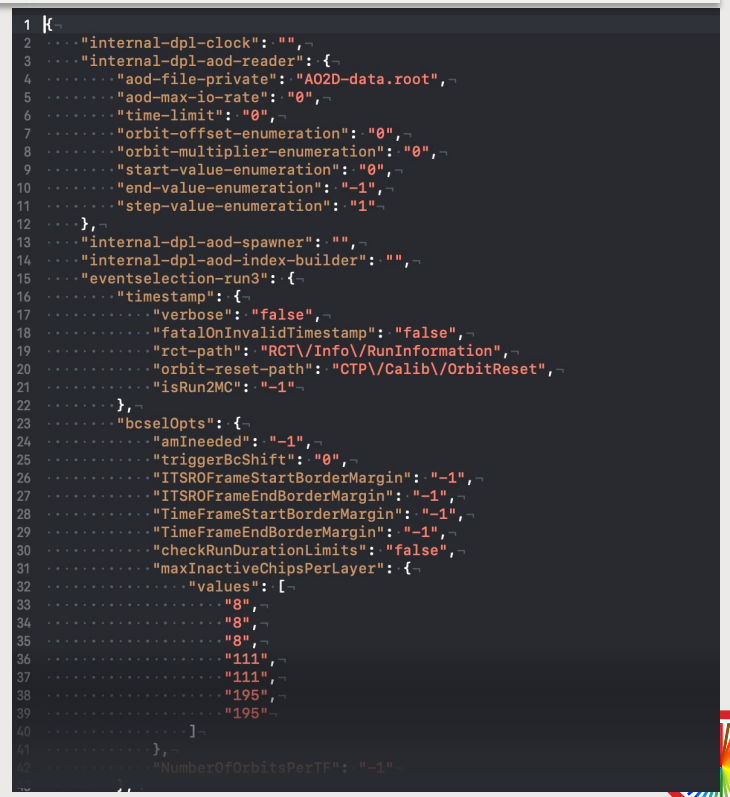
- Each analysis task is an executable → this means you can run them in the command line!

```
o2-analysis-propagationservice | o2-analysis-event-selection-service | o2-analysistutorial-histograms --aod-file AO2D.root
```

Helper task: propagates tracks (primary to PV + V0/Cascades to SV)	Helper task Timestamps and event selection	Example task	Input file

- Configurations can be passed in two ways:
  - Individually: "`--<name of variable> <value>`",
  - All together, stored in a json file

```
o2-analysistutorial-histograms --configuration json://config.json |
o2-analysis-propagationservice --configuration json://config.json |
o2-analysis-event-selection-service --configuration json://config.json
```



- Trick: to generate a json file with all configurations:
  - run the tasks without providing any json.
  - A 'dpl-config.json' file will be automatically generated.
  - Customise it and use it afterwards!

Let's get started!

- We assume you have a working O2Physics installation …
  - …if not, don't panic! Please try and arrange one for the next tutorial sessions!
  - …and don't forget: everything is being recorded for convenience!

- You're going to need a starting point!
  - Let's use our very first example task at: **`O2Physics/Tutorials/PWGMM/myExampleTask.cxx`**
  - This is already included in any current build of O2Physics → you already have a working executable!
    - This means that the CMakeLists.txt file already lists this task as a task that need to be compiled with O2Physics
    - This is convenient! If you ONLY change this task and do aliBuild build O2Physics, ONLY this task will recompile
  - The executable name: **`o2-analysistutorial-mm-my-example-task`**

The example task

```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());
          }
     }
};
```

- A very basic example task!
- Produces an eta histogram

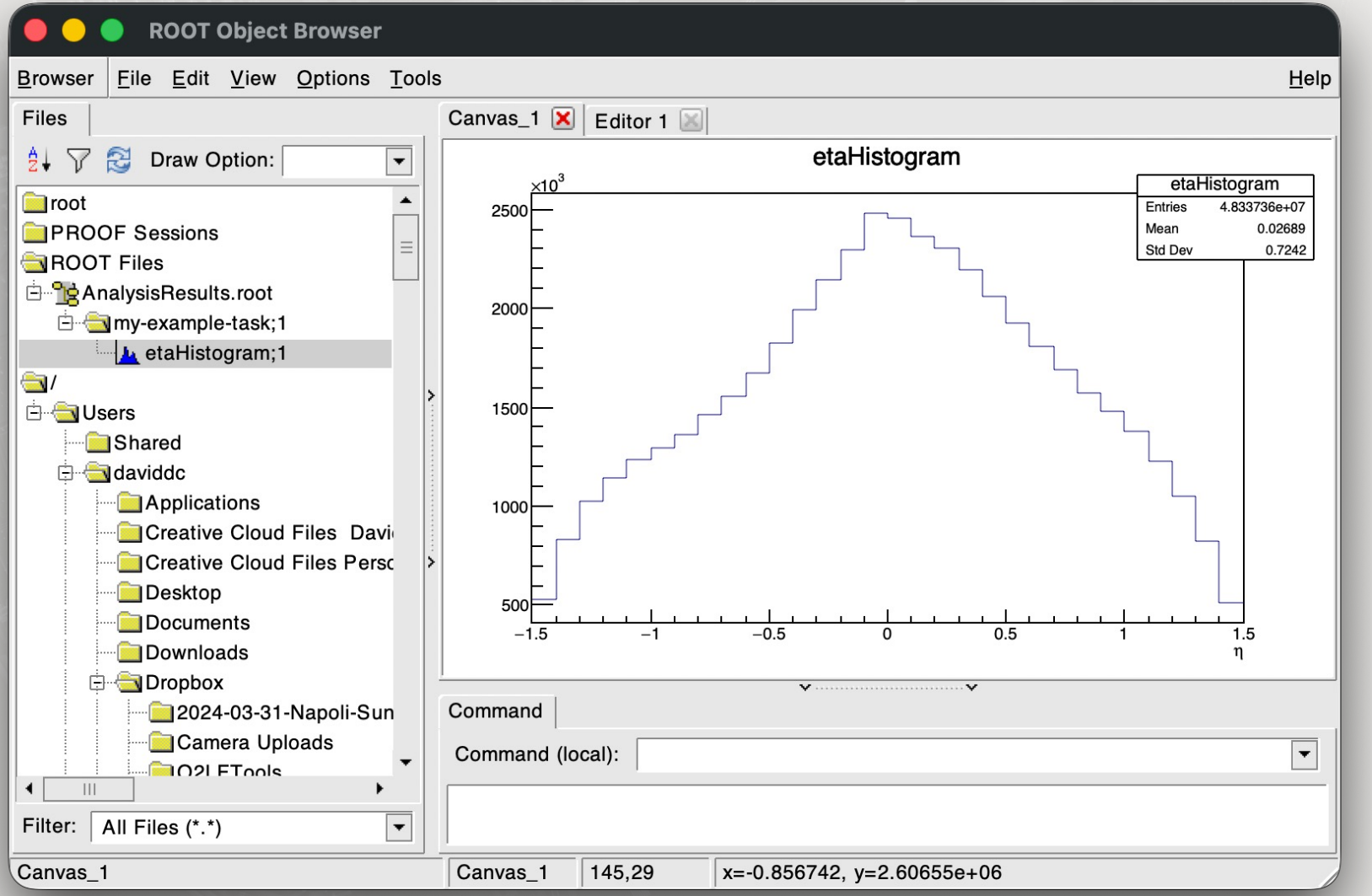
- In a moment, we will add more functionality to it…

- …but for now, let's just run it as is!

From a shell in which you have loaded O2Physics and are in the same directory as the downloaded AO2D.root file, do:

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

Output of our first task execution!



Your first changes to O2Physics !

- Now, we will propose that you make a few changes, step-by-step, to the task.
- Warning! Only modify **`O2Physics/Tutorials/PWGMM/myExampleTask.cxx`**
- Once modified, you can redo **`aliBuild build O2Physics`** and only this task will recompile
- **Or far better, use ninja: see backup slides for instructions!**

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!

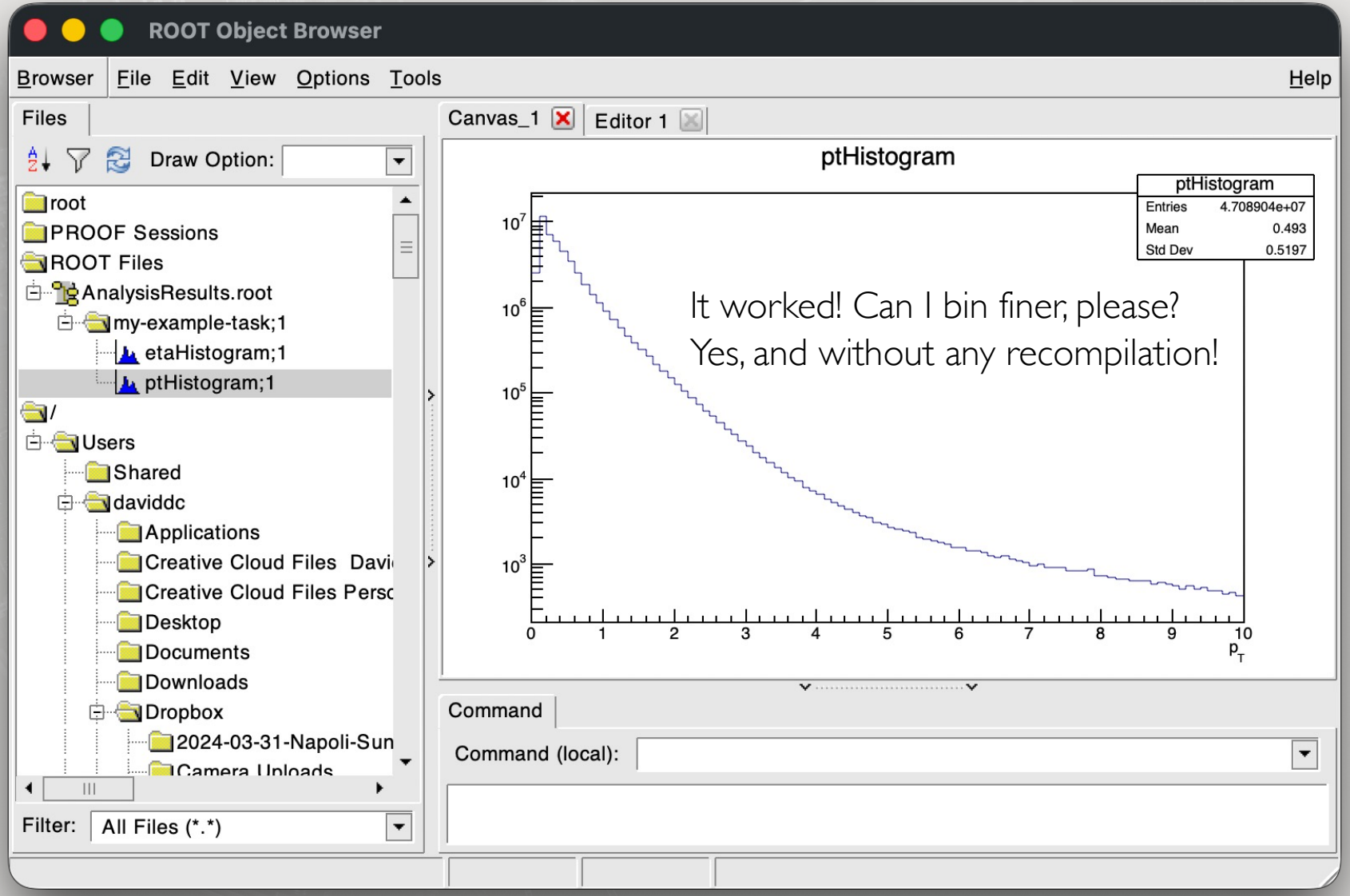
Name		Getter	Type	Comment
o2::aod::track::Pt	E	pt	float	Transverse momentum of the track in GeV/c

Use: **`track.pt();`**

2. Let's make also the number of bins in p_T 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`

Output of the revised task



Dealing with jsons: handling configurations

- 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 that json file! The contents are:

```
[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-private": "AO2D-data.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-private": "AO2D-data.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 $>
```

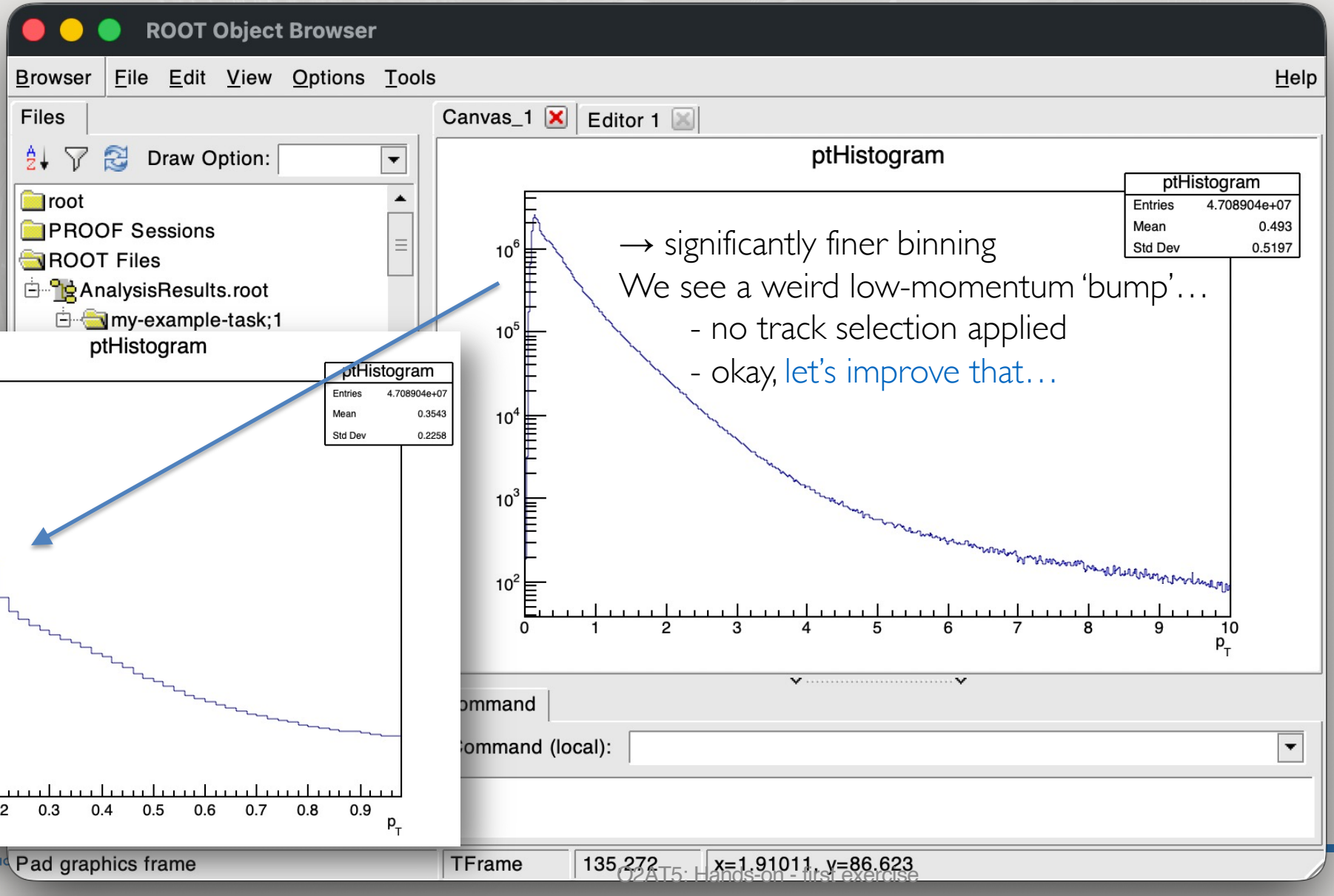
- Now, let's copy the dpl-config json into something like myConfig.json and pass that as argument!
  - No need to provide AO2D-data.root as the aod-file anymore: this is already done inside the new json!

Or: `--nBinsPt 500`

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


ALICE

Output of the revised, reconfigured task



But before that, a parenthesis:

ConfigurableAxis an improved version of what you just did

- It is clearly interesting to have an axis in a histogram be massively configurable also in hyperloop!
- But it's very cumbersome to have to do this fully manually. ConfigurableAxis helps with that!

```
  ConfigurableAxis axisPtQA{"axisPtQA", {VARIABLE_WIDTH, 0.0f, 0.1f, 0.2f, 0.3f, 0.4f, 0.5f,
0.6f, 0.7f, 0.8f, 0.9f, 1.0f, 1.1f, 1.2f, 1.3f, 1.4f, 1.5f, 1.6f, 1.7f, 1.8f, 1.9f, 2.0f, 2.2f,
2.4f, 2.6f, 2.8f, 3.0f, 3.2f, 3.4f, 3.6f, 3.8f, 4.0f, 4.4f, 4.8f, 5.2f, 5.6f, 6.0f, 6.5f, 7.0f,
7.5f, 8.0f, 9.0f, 10.0f, 11.0f, 12.0f, 13.0f, 14.0f, 15.0f, 17.0f, 19.0f, 21.0f, 23.0f, 25.0f,
30.0f, 35.0f, 40.0f, 50.0f}, "pt axis for QA histograms"};
```

```
    histos.add("ptHistogram", "ptHistogram", kTH1F, {axisPtQA});
```

- It will be more difficult to configure this via the command line directly, but you can use jsons
- But, using hyperloop, this will be a lifesaver: you can then change the binning scheme from fixed-width to variable and adjust to your liking using the web interface!
- Try it out: axisPtQA defined above can be used as the 'AxisSpec' before very conveniently!

Adding extra information: changing subscriptions

- Using track quality selections requires an extra table: **`TracksExtra`**
- Using track DCA to primary vertex requires yet another table: **`TracksDCA`**
- This means we'll have to change our subcription.
- Let's also switch to **`Tracks`** instead of **`TracksIU`**: tracks at their point closest to their PV.
- Let's now count events with an event counter! Remember the lecture in the morning…
- The new process function:

Tracks will be provided grouped per collision: the first argument is the collision iterator!

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

- Before the track loop, add a fill command for an event counter (also possible: event selection criteria!):
  - `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() < minCrossedRows ) continue; //badly tracked
if( fabs(track.dcaXY()) > maxDCAxy) continue; //doesn't point to primary vertex
```

Forgetting this is a common mistake!!!

Make these configurable! Defaults to test: 70 rows and 0.2mm

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.github.io/analysis-framework/docs/troubleshooting/#tree-not-found for more information.
```

- 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-analysis-propagationservice --configuration json://myConfig.json |
  o2-analysis-event-selection-service --configuration json://myConfig.json
```

- The event selection service 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…

A simple shell script to aggregate what you need!

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

o2-analysistutorial-mm-my-example-task ${OPTION} | o2-analysis-propagationservice
${OPTION} | o2-analysis-event-selection-service ${OPTION}
```

- You can have the lines above in a run.sh script and then just call source run.sh
- The order in which the tasks appear is not relevant: DPL will connect inputs and outputs as needed automatically

- …and now you should be able to run your first workflow (including more than one task)!
- If using a shell script to aggregate executables, life becomes easier: all helper tasks will be listed conveniently there, no need to repeat them all!

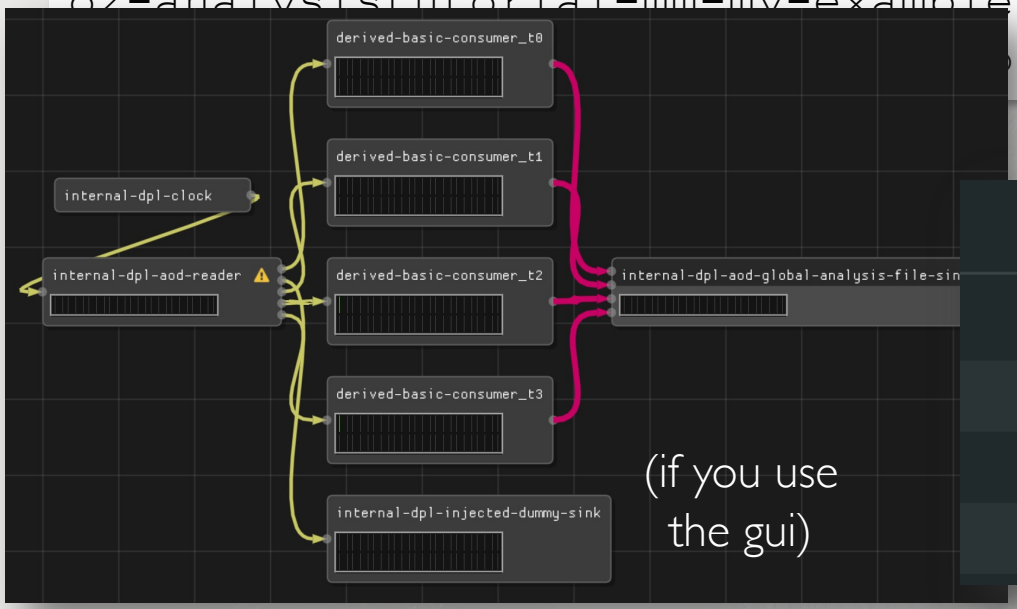
In case you had troubles with the code or would like to check,
the current version of `myExampleTask.cxx` should look like this now!
And: the run.sh script can be found here too

A simple shell script to aggregate what you need!

+ extra: example use of 'pipelining' to speed up analysis!

```
OPTION="-b --configuration json://myConfig.json --pipeline propagation-service:4"

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



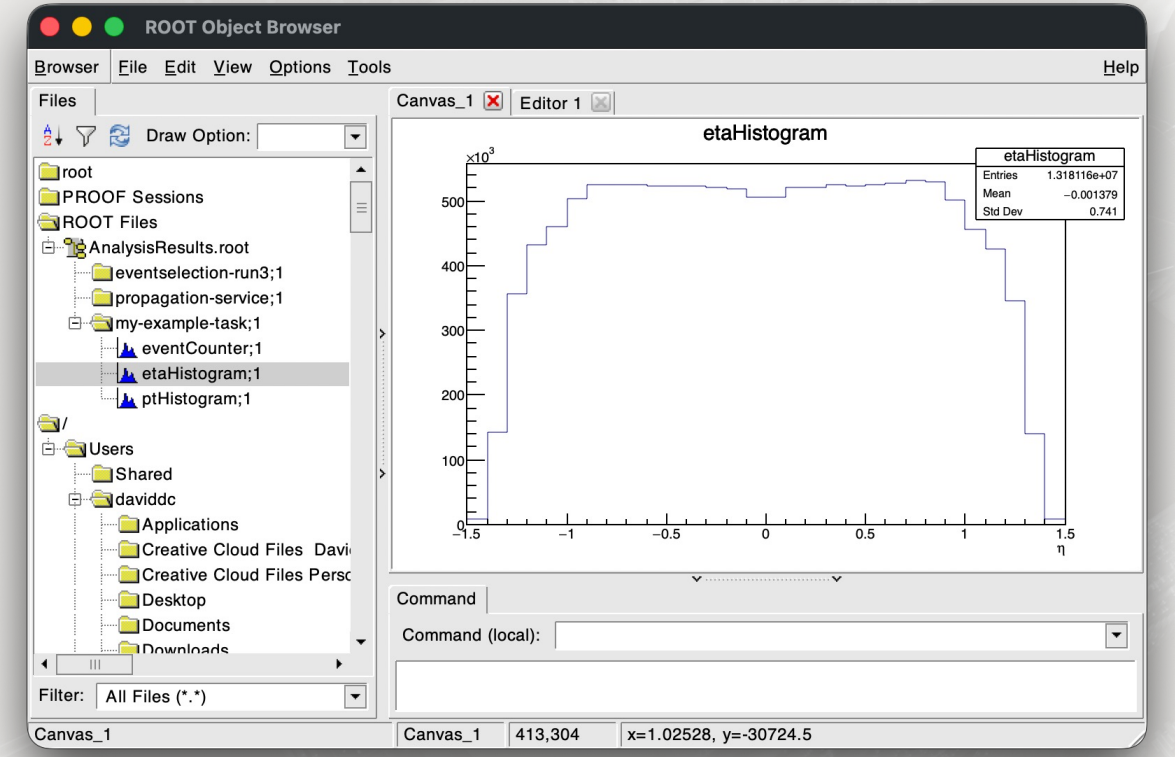
(if you use the gui)

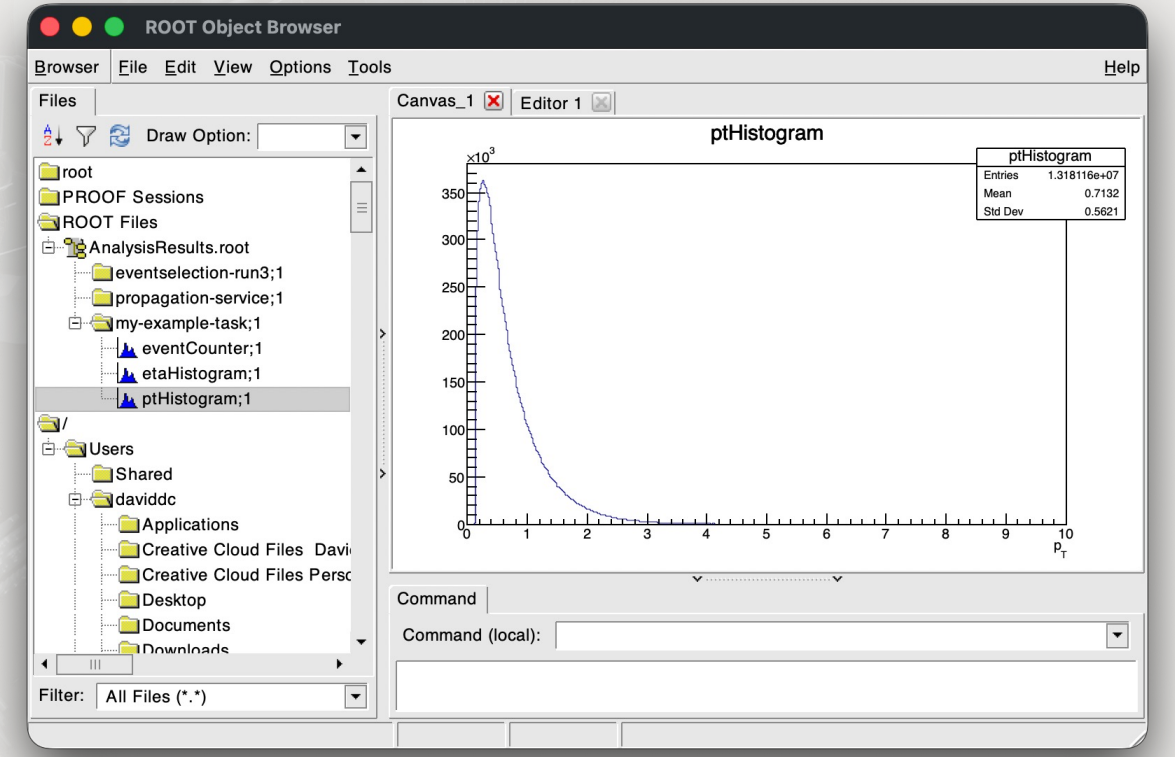
Process Name	% CPU	CPU Time
o2-analysis-propagationservice	99,2	39,22
o2-analysis-propagationservice	93,8	41,54
o2-analysis-propagationservice	85,4	37,43
o2-analysis-propagationservice	76,1	42,63

Use multi-core CPUs efficiently by choosing one or more tasks to replicate (drawback: requires some more memory)

In case you had troubles with the code or would like to check,
the current version of `myExampleTask.cxx` should look like this now!
And: the run.sh script can be found here + myConfig.json here

The outcome: a more selective sample of particles!





backup

Surgical compilation: ninja to the rescue

- **If you have a compiled O2Physics installation** and just want to recompile one subdirectory or even one particular executable, this can be done using a tool called “ninja” – it will help you be faster!

- Go to your build directory using:

Bear in mind: different path needed for any non-standard installation path

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

- Once in that directory, load the build environment doing: (important: you should not have called alienv before!)

```
direnv allow
```

- You can then rebuild only one specific subdirectory of O2Physics:

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

```
# Example: ninja PWGCF/Tasks/install
```

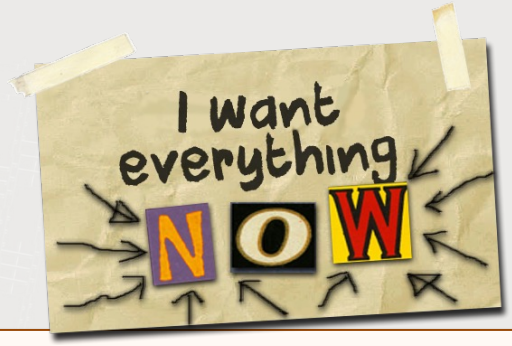
- You can even rebuild only one single executable:

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

```
# Example: ninja stage/bin/o2-analysis-cf-correlations
```

Instant gratification:

The recommended installation for the tutorial




```
mkdir -p ~/alice
cd ~/alice
```
```
# creates an "alice"
# directory in your home
```

```
aliBuild init O2@dev
aliBuild init O2Physics@master
```
```
# initializes O2 (if interested in core dev)
# initializes O2Physics
```

```
export O2PHYSICS_COMPONENTS="Tutorials/install Common/install"
aliBuild build O2Physics
```
```
# small compilation
# compile!
```

- IMPORTANT: do this before the actual tutorial and especially before the Hands-on!
  - Compilation takes time! Make sure it finished successfully so you're set
- **If you're following PWG hands-on session(s),** add also "PWGxx/install" to O2PHYSICS_COMPONENTS
- Add also "-j N" after "aliBuild" if you do not have a lot of memory
  - The parameter N should be at most (available RAM) / 4GB (mac) or (available RAM) / 6GB (linux)

(for completeness: this assumes memory is more limiting than N_{cores}; ideally, N should be min[(available RAM) / 6GB, N_{cores}]