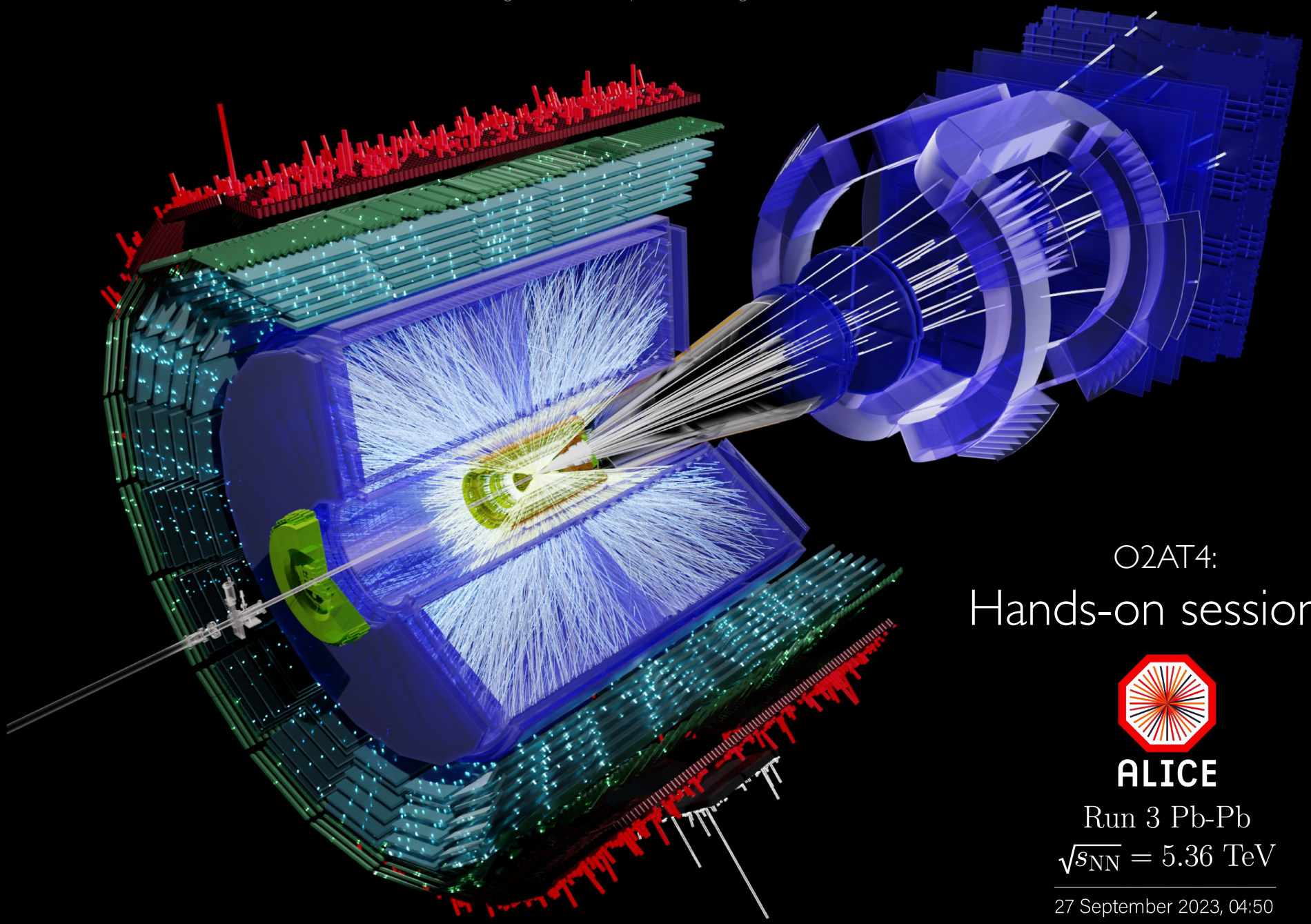
Image reference: https://zenodo.org/records/13732384



O2AT4:

Hands-on session 1

ALICE

Run 3 Pb-Pb

$\sqrt{s_{\mathrm{NN}}} = 5.36$ TeV

27 September 2023, 04:50

Data for this exercise

- Single AO2D file from Pb-Pb pass4 reconstruction (brand new!)
  - Download via alien_cp (alien shell: it's useful!):

```
$> alien_cp
/alice/data/2023/LHC23zzh/544116/apass4/0140/o2_ctf_run00544116_orbit0
034426688_tf0000000251_epn132/001/AO2D.root file:./AO2D.root
```

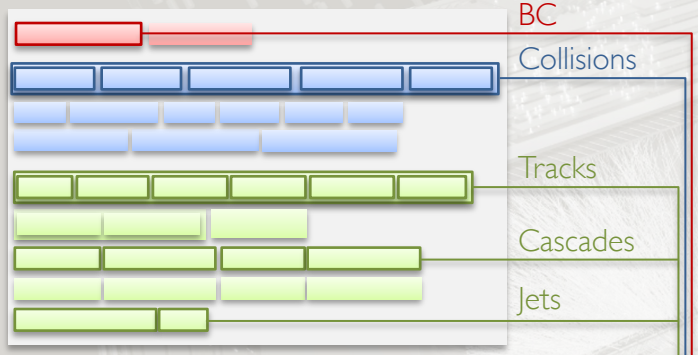
  - Download via CERNBox:
  - https://cernbox.cern.ch/index.php/s/10kHA1hlHVL5Tad

  - Download via dropbox:
  - https://www.dropbox.com/scl/fi/fqkuuxmc0pt2v7rc8y9vp/AO2D.root?rlkey=qk3migmerxke0tjm3axxhzbur&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

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-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 ("|")
- `--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…

The helper tasks

More on that as we go!

- o2-analysis-timestamp: provides time stamp columns to events
- o2-analysis-event-selection: provides event selection columns to events
- o2-analysis-multiplicity-table: provides multiplicity columns to events
- o2-analysis-centrality-table: provides centrality percentile columns to events
- o2-analysis-track-propagation: propagates tracks to primary vertices, provides $DCA_{xy/z}$ columns to tracks
- o2-analysis-pid-tpc: provides TPC pid columns to tracks
- o2-analysis-pid-tof: provides TOF pid columns to tracks
- o2-analysis-lf-lambdakzerobuilder: provides the V0 data table with all V0 candidate information
- o2-analysis-lf-cascadebuilder: provides the cascade data table with all cascade candidate information

N.B.: Your PWG might have specific Table producers for specific tasks!

- This will be explained in the dedicated sessions!

Configuring tasks using json files

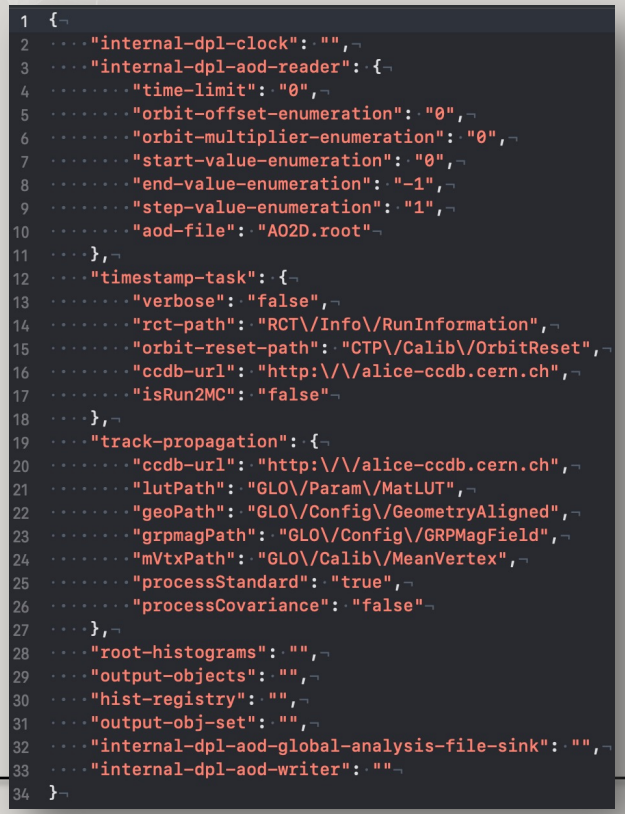
- Each analysis task is an executable → this means you can run them in the command line!

```
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

- Configurations can be passed in two ways:
  - Individually: --nBinsPhi 200 in the previous case
  - All together, stored in a json file

```
o2-analysistutorial-histograms --configuration json://config.json |
o2-analysis-track-propagation --configuration json://config.json |
o2-analysis-timestamp --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

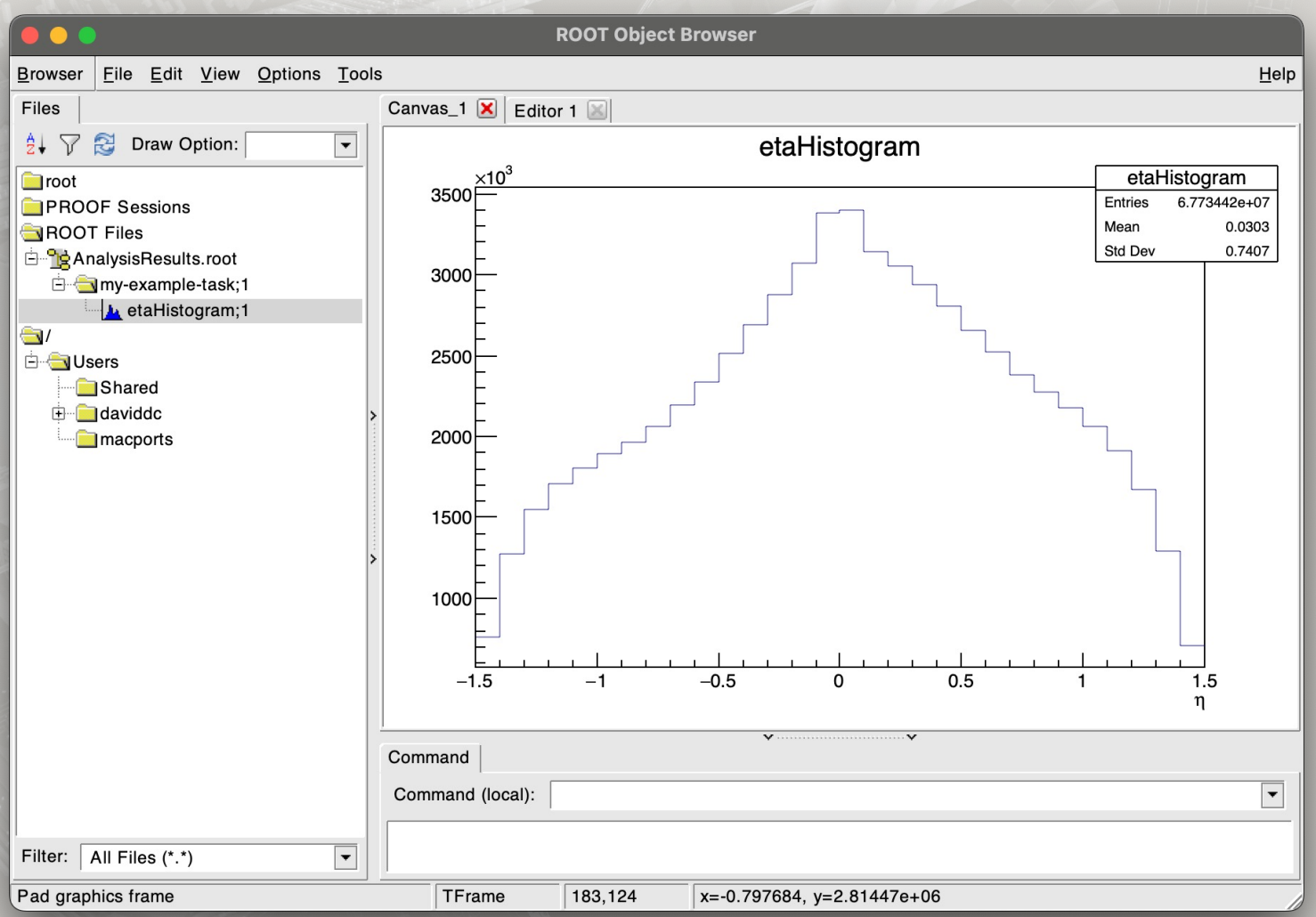
- In a moment, we will add more functionality to it…

- …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.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!

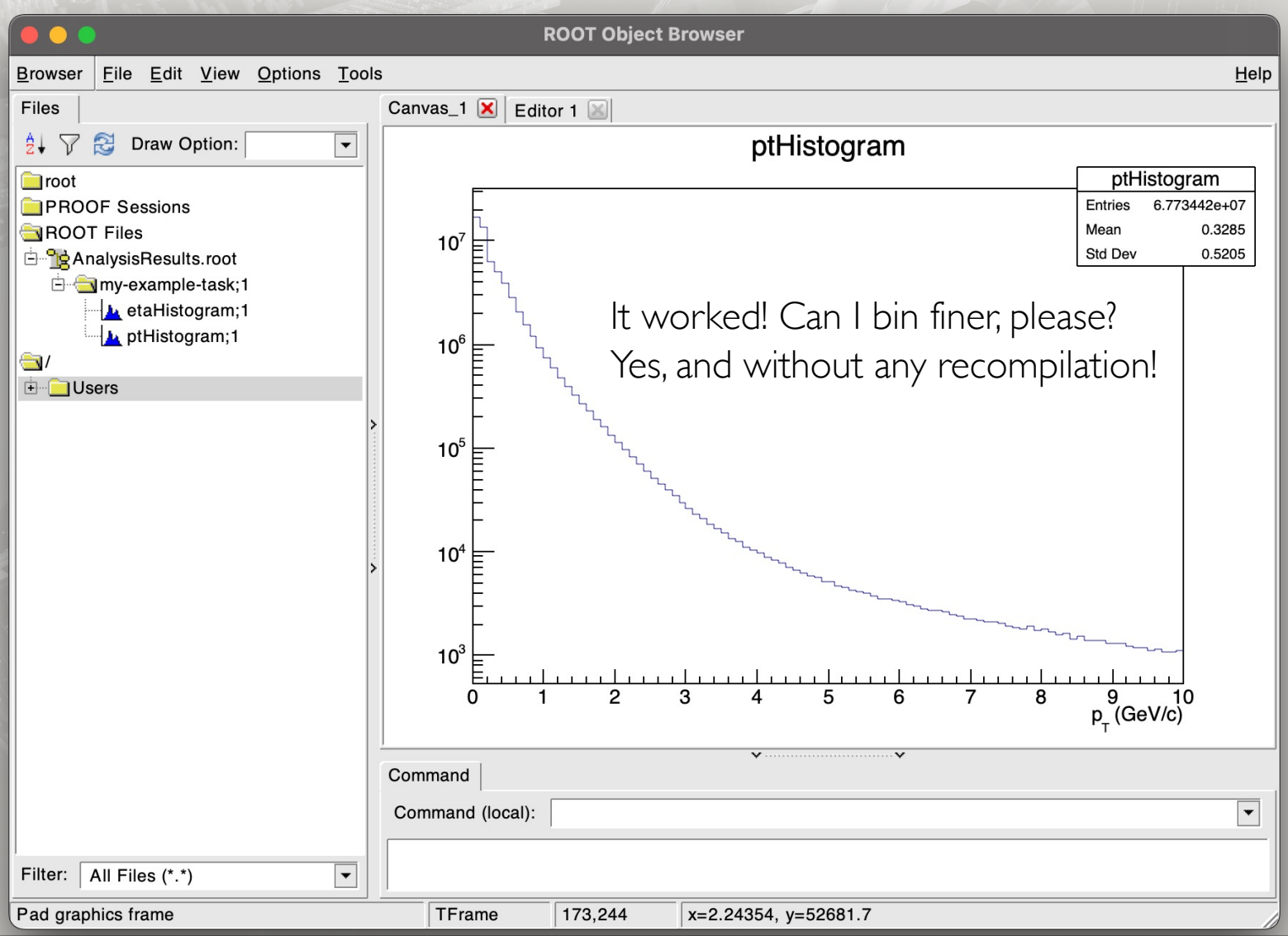
Name		Getter	Type	Comment
o2::aod::track::Pt	E	pt	float	Transverse momentum of the track in GeV/c

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:

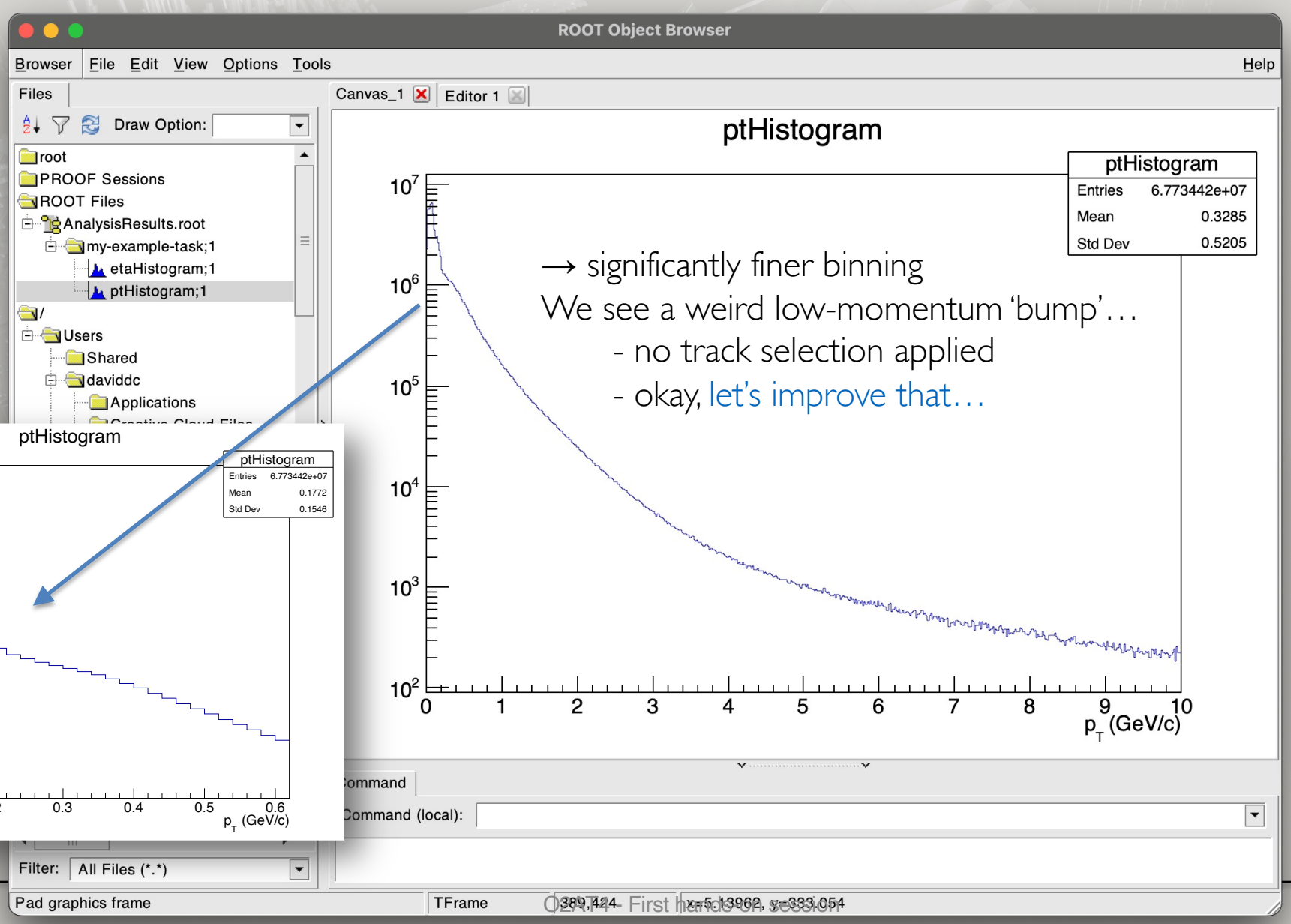
```
[O2Physics/latest] ~/O2AT $> cat dpl-config.json
{
    "internal-dpl-clock": "",
    "internal-dpl-aod-reader": {
        "time-limit": "0",
        "orbit-offset-enumeration": "0",
        "orbit-multiplier-enumeration": "0",
        "start-value-enumeration": "0",
        "end-value-enumeration": "-1",
        "step-value-enumeration": "1",
        "aod-file-private": "AO2D.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.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
```

Output of the revised, reconfigured task



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 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
```

Forgetting this is a common mistake!!!

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-track-propagation --configuration json://myConfig.json |
        o2-analysis-timestamp --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…

A simple shell script to aggregate what you need!

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

o2-analysistutorial-mm-my-example-task ${OPTION} | o2-analysis-track-propagation
${OPTION} | o2-analysis-timestamp ${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)!
- This is how one utilises "helper" tasks. There are really many of them for tasks such as PID, etc etc…

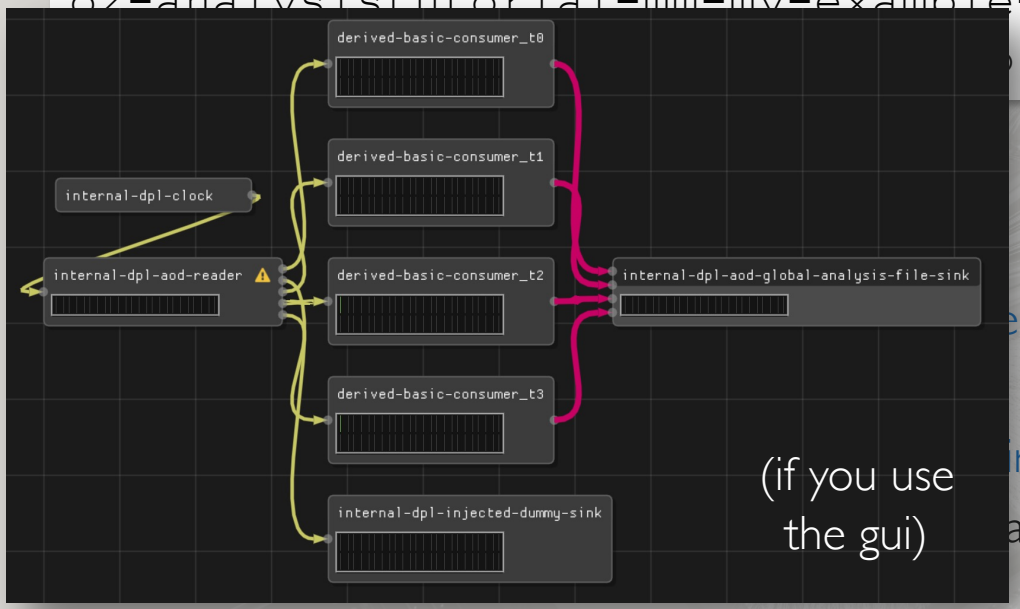
In case you had troubles with the code or would like to check,
the current version of `myExampleTask.cxx` should look like this now!
And: the run.sh script can be found here too

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 track-propagation:4"

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



(if you use the gui)

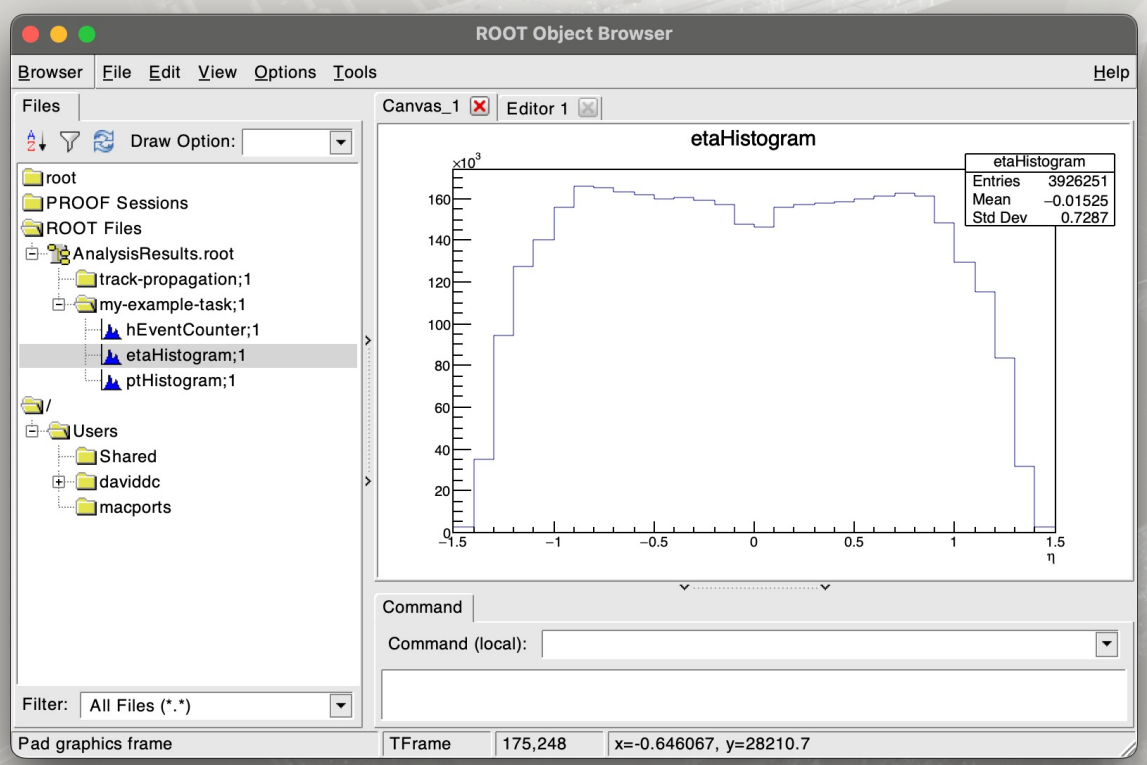
Process Name	% CPU	CPU Time
o2-analysis-track-propagation	98,9	30,89
o2-analysis-track-propagation	94,6	27,08
o2-analysis-track-propagation	63,4	25,85
o2-analysis-track-propagation	46,5	31,33

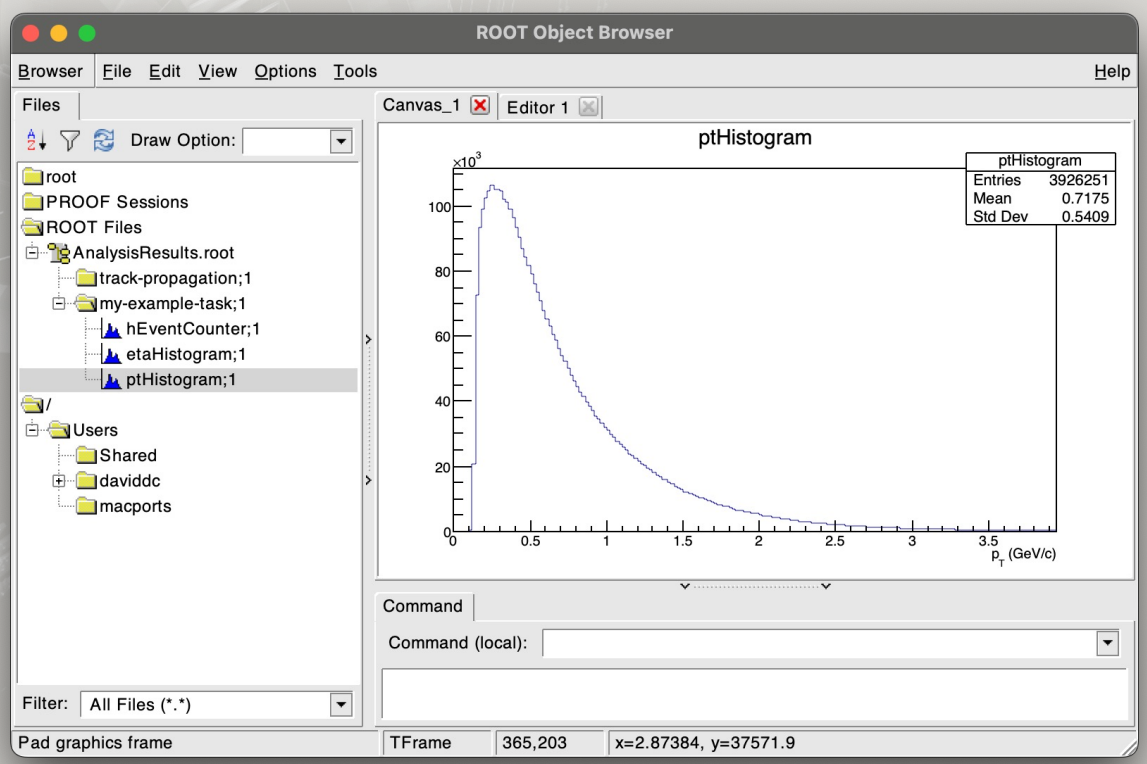
Use multi-core CPUs efficiently by choosing one or more tasks to replicate (drawback: requires some more memory)

are really many of them for tasks such as PID, etc etc…

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

The outcome: a more selective sample of particles!





…and a coffee break!

backup

Ninja to the rescue: faster compilation

- **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!

- Enter an O2Physics environment and include the ninja tool with:

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

- Go to the build directory via:

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

- 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
```