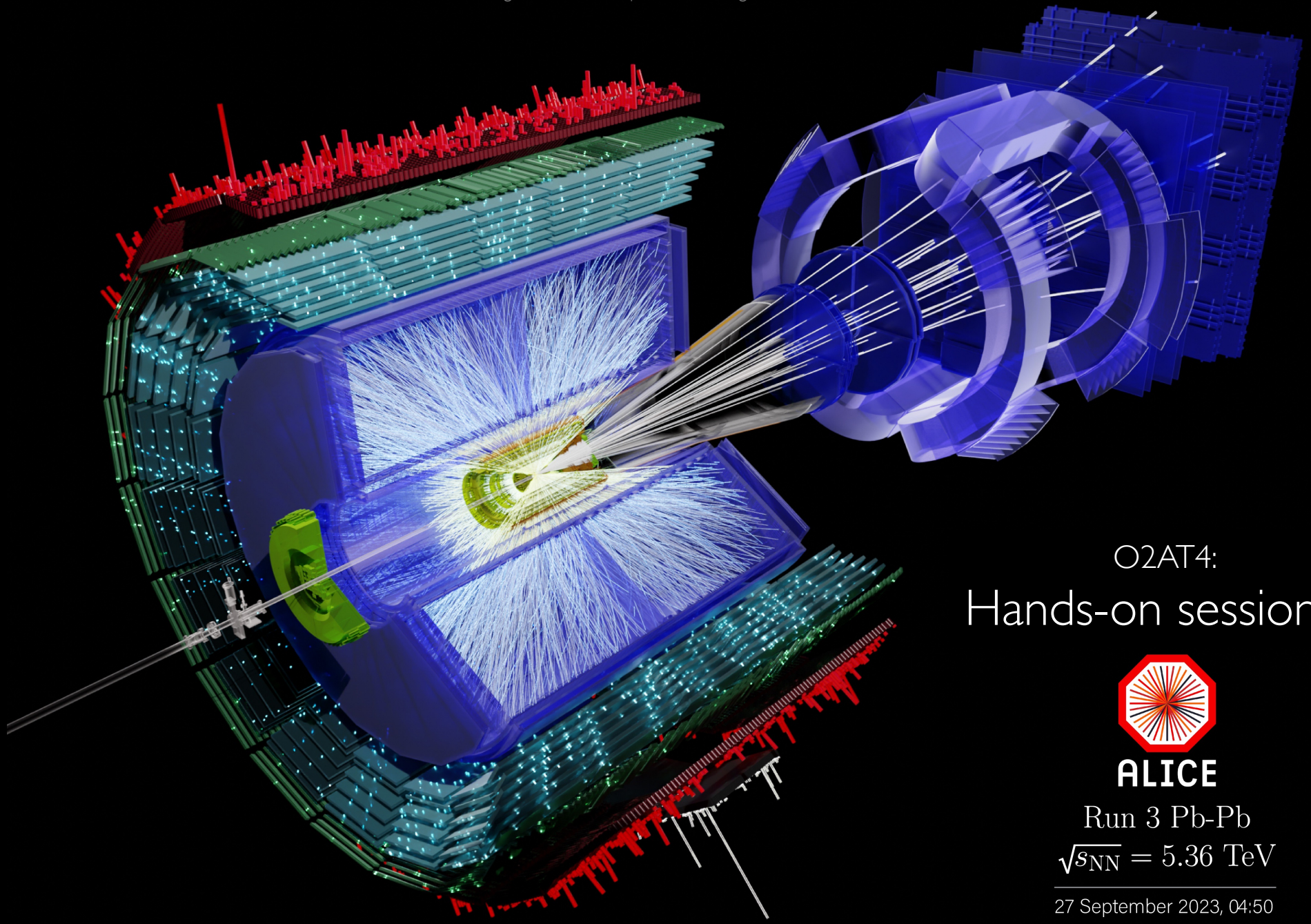
Image reference: https://zenodo.org/records/13732384



O2AT4:

Hands-on session III

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 2024 MC anchored to pp LHC23zzh pass4 (LHC23g3)
- Alien_cp:

```
alien_cp /alice/sim/2024/LHC24g3/0/544116/AOD/001/AO2D.root file:./AO2D.root
```

- CERNBox:
- https://cernbox.cern.ch/index.php/s/JN9GVHVmEoB5i5i
- Dropbox:
- https://www.dropbox.com/scl/fi/ft59tfo6nsjoldztkgy2e/AO2D.root?rlkey=gjn5a0ogdifgjp6fkpi0t9qb2&dl=0

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

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

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

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

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

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

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

      void init(InitContext const&)
      {
            // define axes you want to use
            const AxisSpec axisCounter{1, 0, +1, ""};
            const AxisSpec axisEta{30, -1.5, +1.5, "#eta"};
            const AxisSpec axisPt{nBinsPt, 0, 10, "p_{T}"};
            // create histograms
            histos.add("eventCounter", "eventCounter", kTH1F, {axisCounter});
            histos.add("etaHistogram", "etaHistogram", kTH1F, {axisEta});
            histos.add("ptHistogram", "ptHistogram", kTH1F, {axisPt});
      }

      void process(o2::aod::Collision const& collision, myFilteredTracks const& tracks)
      {
            histos.fill(HIST("eventCounter"), 0.5);
            for (const auto& track : tracks) {
                  if( track.tpcNClsCrossedRows() < 70 ) continue; //badly tracked
                  histos.fill(HIST("etaHistogram"), track.eta());
                  histos.fill(HIST("ptHistogram"), track.pt());
            }
      }
};
```

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

- Starting point: code similar to the one used yesterday!
- The good old:

myExampleTask.cxx
+
Changes (link)

- We'll now study:
  1. How do we calculate p_T resolution?
  2. How do we calculate efficiencies?
  3. How can we understand collision reconstruction efficiency in the TF era?

- For this, we'll employ more tools!
- And fundamentally: access simulation data!

ALICE

Acessing the MC information for each track

- You will now need to de-reference the MC particle for each and every track.
- This is done via a label: that is an index column pointing to the **McParticles** table.
- This index column is the sole element in the **McTrackLabel** table – joinable with tracks!

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

- Now you have a track label that you can de-reference!
- …but to de-reference it, you also need to subscribe to the **McParticles** table (or you won't have those!)

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

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

```
const AxisSpec axisDeltaPt{100, -1.0, +1.0, "#Delta(p_{T})"};
(…)
histos.add("ptResolution", "ptResolution", kTH2F, {axisPt, axisDeltaPt});
```

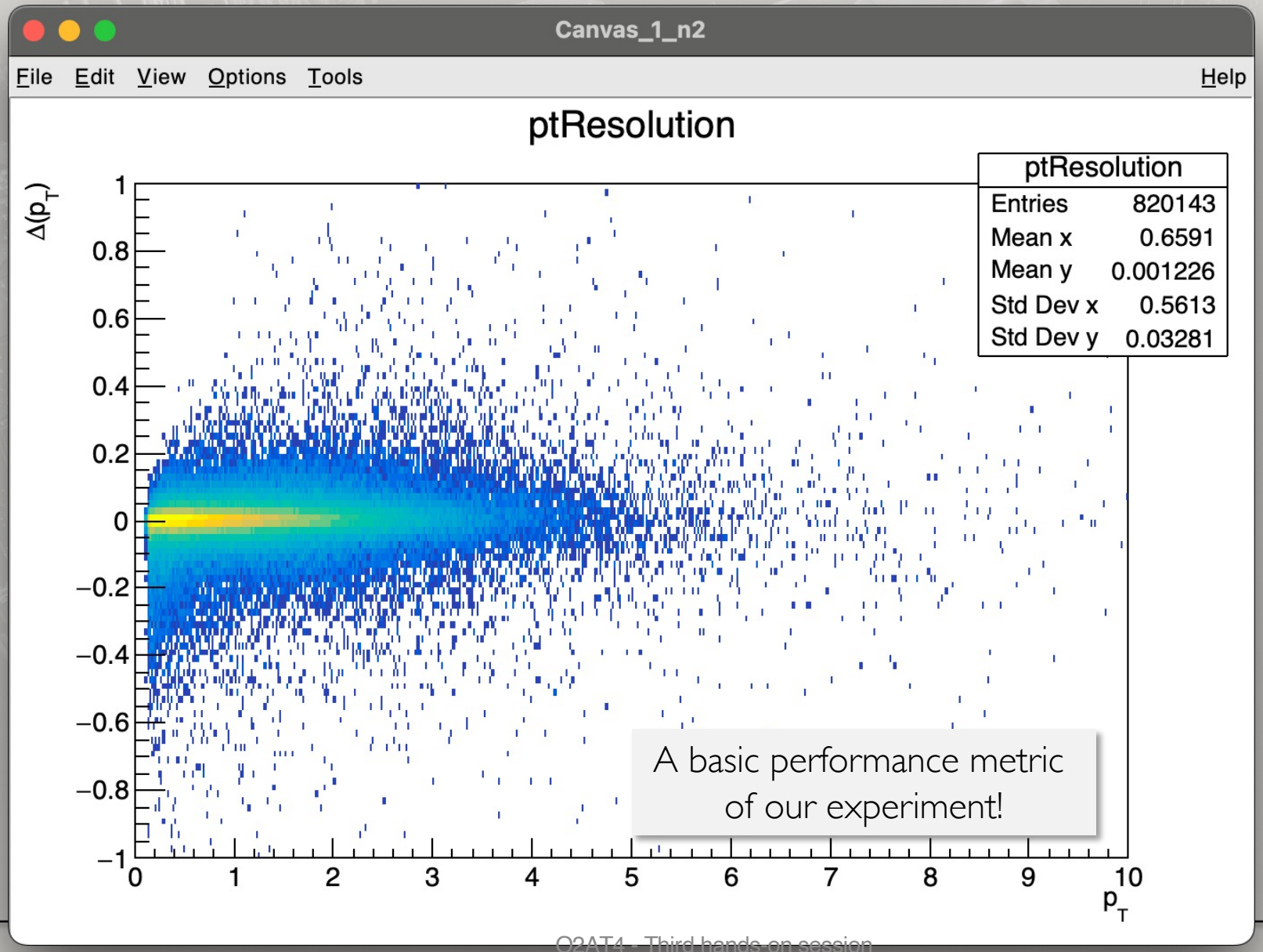
- Now comes the magic: the immediate de-referencing!

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

And that's it!

For running, you can still use the same run.sh you used yesterday throughout this session!
But: add o2-analysis-mccollision-converter to the mix (McCollisions got a new version recently)

The outcome: momentum resolution as a function of transverse momentum



Getting the reconstructed pion, kaon and proton spectra

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

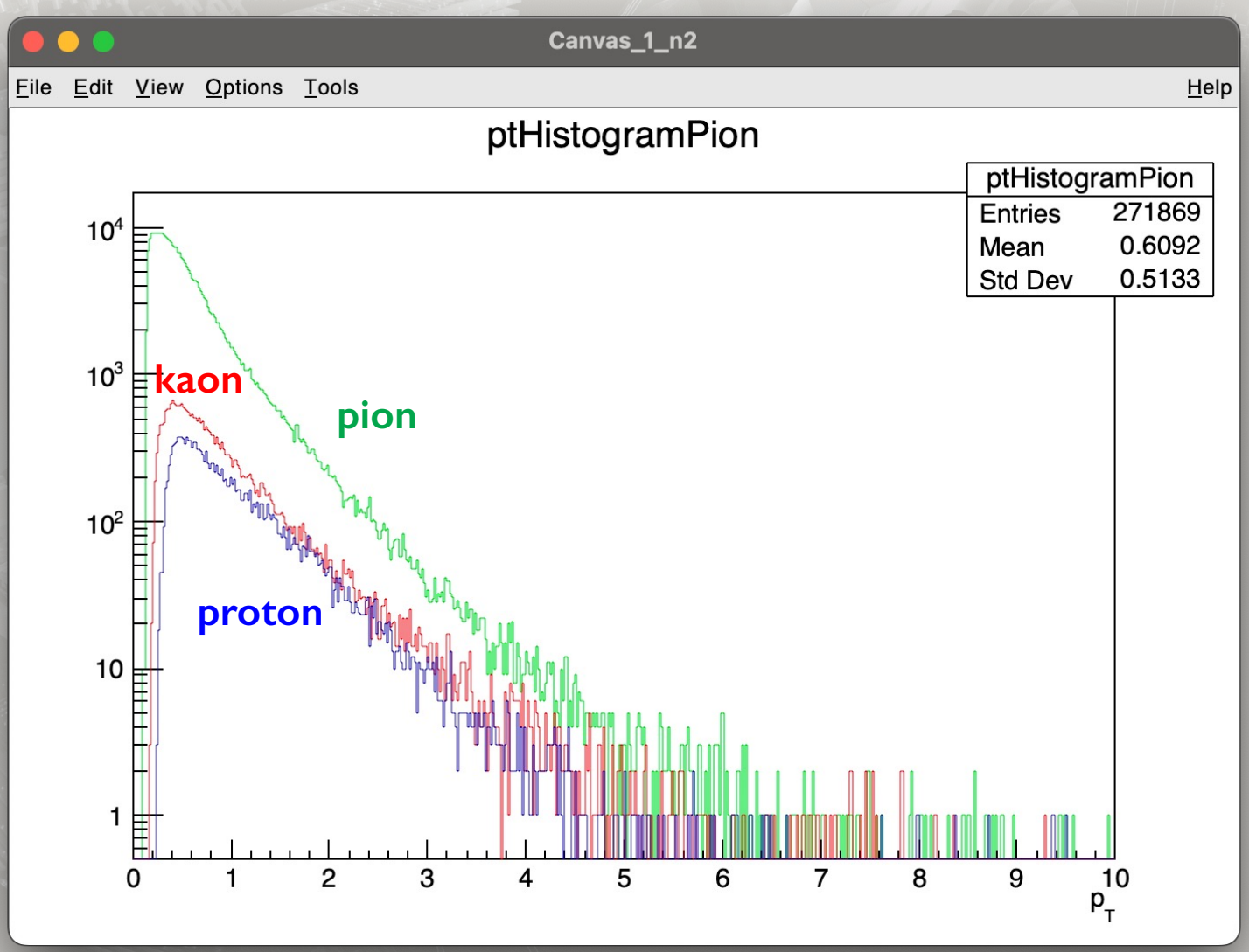
```
histos.add("ptHistogramPion", "ptHistogramPion", kTH1F, {axisPt});
histos.add("ptHistogramKaon", "ptHistogramKaon", kTH1F, {axisPt});
histos.add("ptHistogramProton", "ptHistogramProton", kTH1F, {axisPt});
```

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

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

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

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



Processing pure generation information

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

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

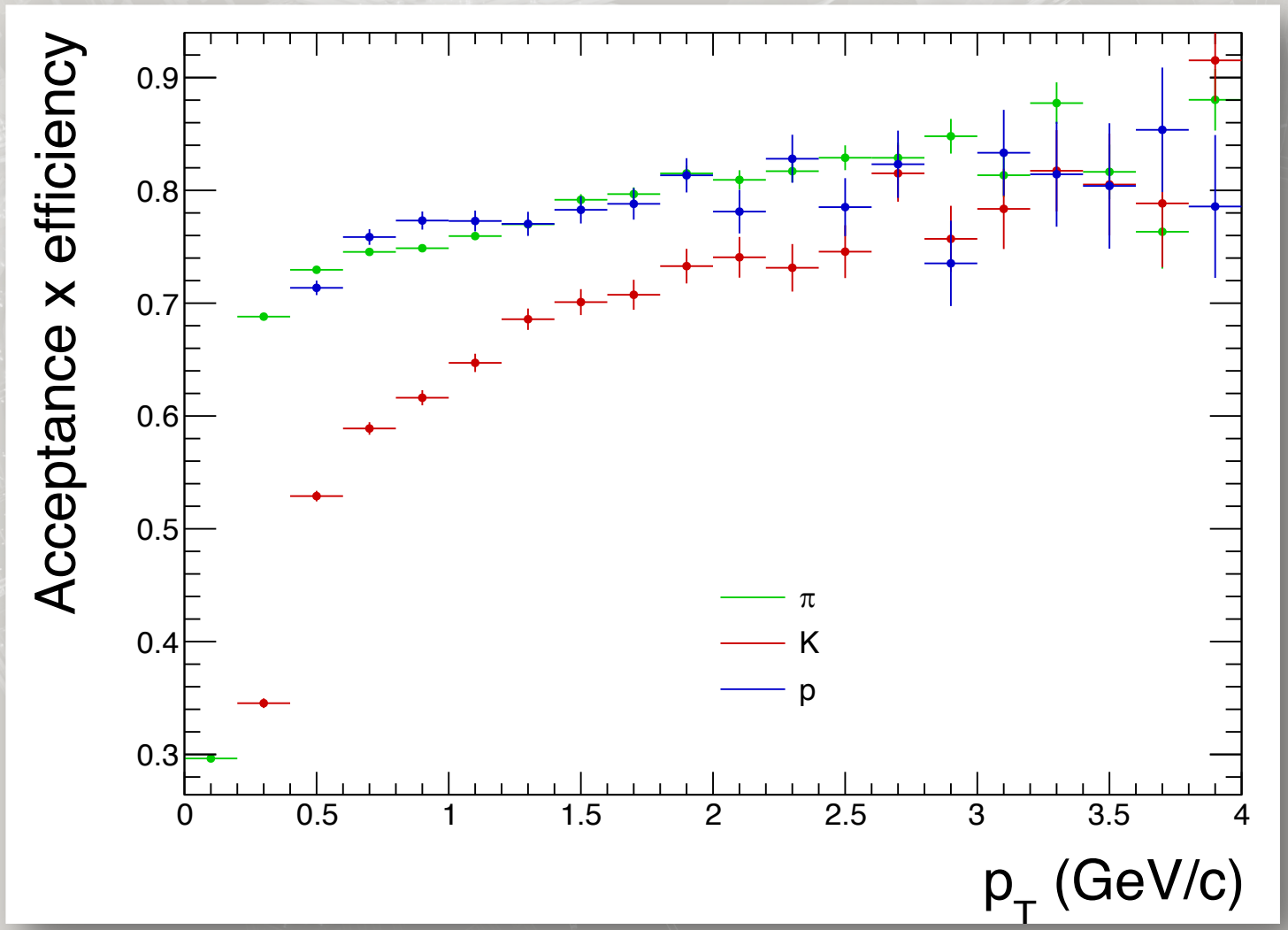
- Now let's add another process function!

```
void processReco(aod::Collision const& collision, myFilteredTracks const& tracks, aod::McParticles const&)
{
     (…)
}
PROCESS_SWITCH(myExampleTask, processReco, "process reconstructed information", true);

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

The outcome: efficiencies of pion, kaon and proton

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



Example drawing macro can be found here

Moving flexibly around from MC to reconstruction: going beyond!

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

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

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

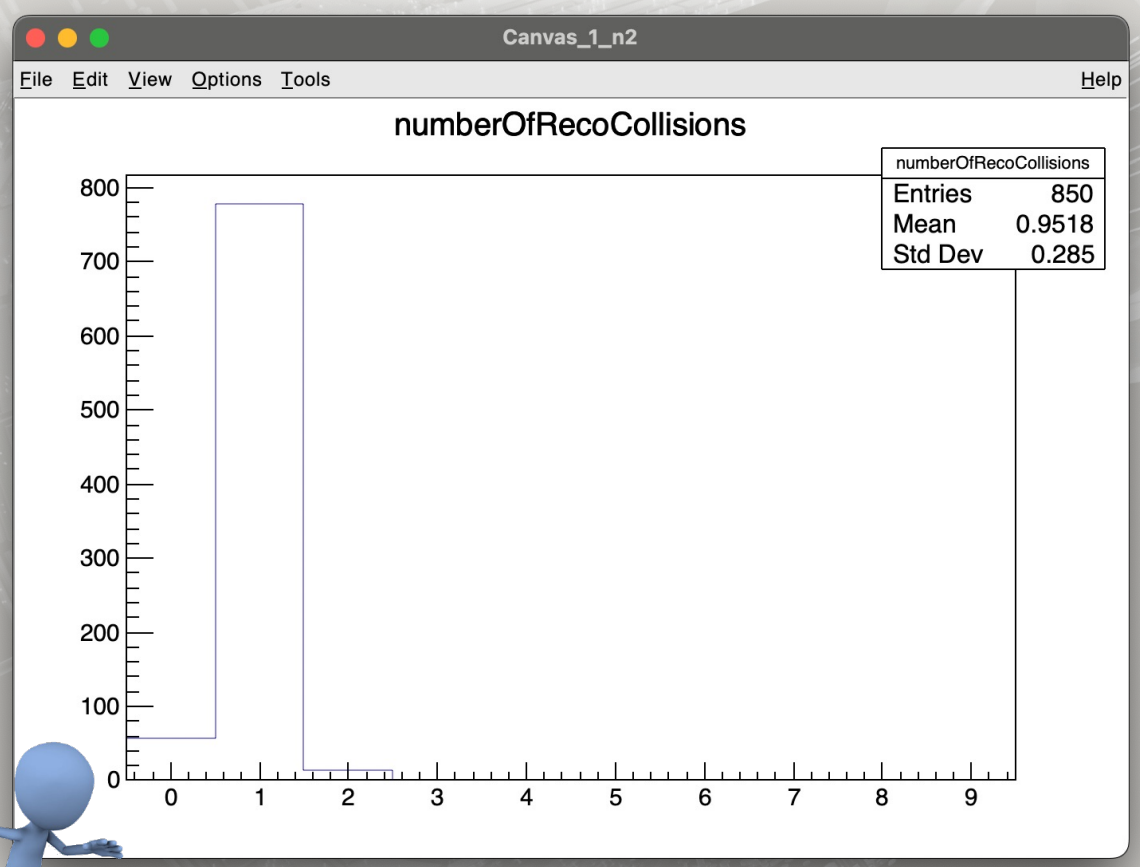
```
histos.fill(HIST("numberOfRecoCollisions"), collisions.size()); // number of times coll was reco-ed
```

- Then one can immediately find out the number of times a MC collision appears by just checking the size of collisions!

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

```
//inside processSim: now loop over each time this collision has been reconstructed and aggregate tracks
std::vector<int> numberOfTracks;
for (auto& collision : collisions) {
     auto groupedTracks = tracks.sliceBy(perCollision, collision.globalIndex());
     // size of grouped tracks may help in understanding why event was split!
     numberOfTracks.emplace_back(groupedTracks.size());
}
```

Final outcome: number of times a MC collision was reconstructed



- Homework bonus: how does this
- Change with centrality?

- Can you implement a centrality selection based on the content of Mattia's talk yeterday?