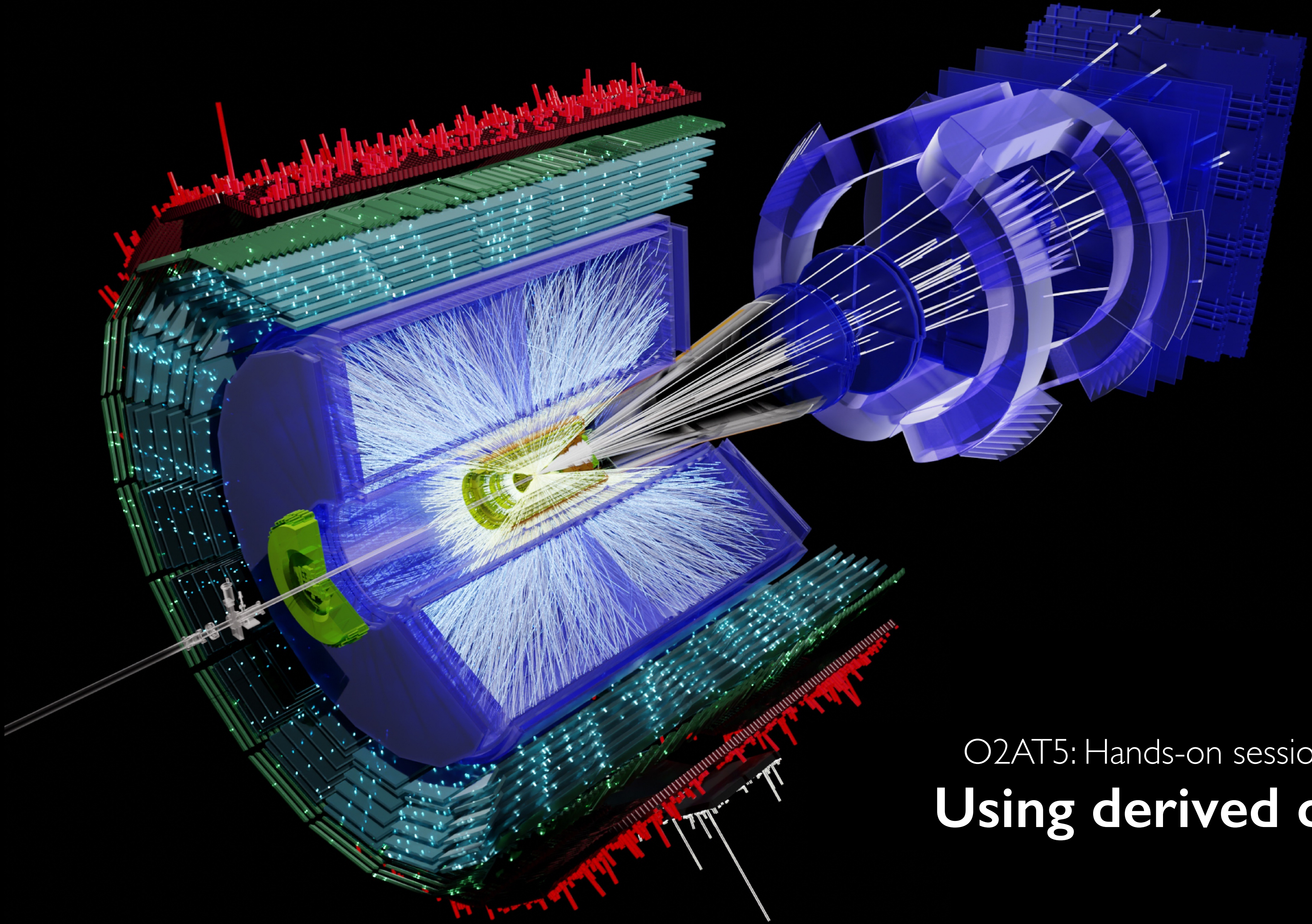
Image reference: https://zenodo.org/records/13732384



O2AT5: Hands-on session III

Using derived data

Where we stopped last time: a simple analysis of tracks per collision

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

void init(InitContext const&)
{
      // define axes you want to use
      const AxisSpec axisCounter{1, 0, 1, "events"};
      const AxisSpec axisEta{30, -1.5, +1.5, "#eta"};
      const AxisSpec axisPt{nBinsPt, 0, 10, "p_{T} (GeV/c)"};

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

void process(aod::Collision const& collision, soa::Join<aod::Tracks, aod::TracksExtra,
aod::TracksDCA> const& tracks)
{
      histos.fill(HIST("hEventCounter"), 0.5);
      for (auto& track : tracks) {
            if( track.tpcNClsCrossedRows() < 70 ) continue;
            if( fabs(track.dcaXY()) > 0.2) continue;
            histos.fill(HIST("etaHistogram"), track.eta());
            histos.fill(HIST("ptHistogram"), track.pt());
      }
}
```

- A lot of CPU spent in preparing tracks, calculating DCAs …
- What if we wanted to use the (processed) output in a more efficient way? Derived data!

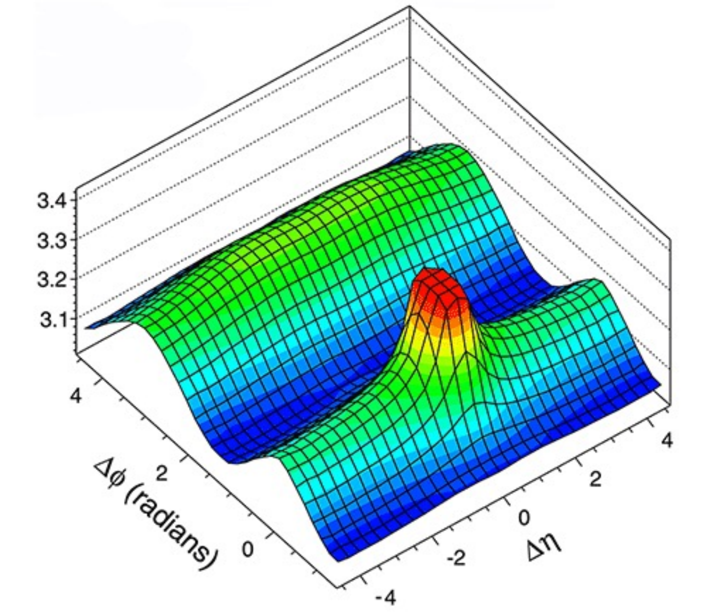
Using derived data

- For this part of the tutorial we will now use derived data
  - This example is still **Pb-Pb 2023 pass5**, but we filtered only eta, phi and pT of tracks above 4 GeV/c
  - This isn't a standard, complete AO2D anymore!

Now:

- I will walk you through creating a two-particle correlation analysis…
- Derived data subscription advantage: no supporting task needed anymore!
  - All support tasks already did what they had to: no track propagation, etc
  - In general: dramatic reduction in CPU time!

- **Parenthesis!**
  - Two-particle correlation functions are a staple of heavy-ion physics! You might have seen results from them in the past. (more reading)



The data we will be using

- In order to showcase the use of derived data and current Pb-Pb 2023 datataking:
- Derived files based on Pb-Pb pass5 have been generated for you!
  - Small file (**~900 KB**): cernbox dropbox
  - Large file (**~600 MB**, equivalent to **6×10^7 events**): cernbox dropbox
    - Pretty cool: 60M events is 5x larger than all Pb-Pb data taken in 2010!
    - Thanks to derived data use, this works very well even locally!

- Do not worry about how this was generated for now.
  - But if you are anyway curious, the code that generated this is also in "Tutorials" [1], and we'll discuss in the next slides
  - See tomorrow's talk on derived data by Victor Gonzalez for more!

- My suggestion: start developing with the small file, but download the large file now (while you code)!
  - This way, once your work is done, you can run the final code over the large file
  - Later on, you can use pipelining to loop over the large file!

[1] https://github.com/AliceO2Group/O2Physics/blob/master/Tutorials/Skimming/derivedBasicProvider.cxx

A brand new subscription for our derived data

Tutorials/Skimming/**DataModel**/DerivedExampleTable.h

```
namespace o2::aod
{
DECLARE_SOA_TABLE(DrCollisions, "AOD", "DRCOLLISION", o2::soa::Index<>,
o2::aod::collision::PosZ);
using DrCollision = DrCollisions::iterator;

namespace exampleTrackSpace
{
DECLARE_SOA_INDEX_COLUMN(DrCollision, drCollision);
DECLARE_SOA_COLUMN(Pt, pt, float);
DECLARE_SOA_COLUMN(Eta, eta, float);
DECLARE_SOA_COLUMN(Phi, phi, float);
} // namespace exampleTrackSpace
DECLARE_SOA_TABLE(DrTracks, "AOD", "DRTRACK", o2::soa::Index<>,
exampleTrackSpace::DrCollisionId,
exampleTrackSpace::Pt, exampleTrackSpace::Eta, exampleTrackSpace::Phi);
using DrTrack = DrTracks::iterator;
} // namespace o2::aod
```

- First task: how should I add eta and pT histograms here?
- Hint: consult data model …
- …which is very simple for this example!

Tutorials/Skimming/**derivedBasicConsumer**.cxx

```
void init(InitContext const&)
{
     // define axes you want to use
     const AxisSpec axisCounter{1, 0, +1, ""};
     histos.add("eventCounter", "eventCounter", kTH1F, {axisCounter});
}

void process(aod::DrCollision const& collision)
{
     histos.fill(HIST("eventCounter"), 0.5);
}
```

You should edit this file!

Executable name:
`o2-analysistutorial-derived-basic-consumer`

Bulk operations via filtering!

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

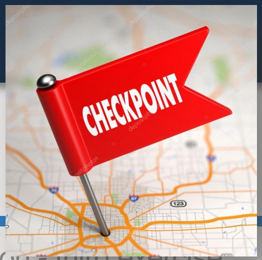
```
Filter collZfilter = nabs(aod::collision::posZ) < 10.0f;
```

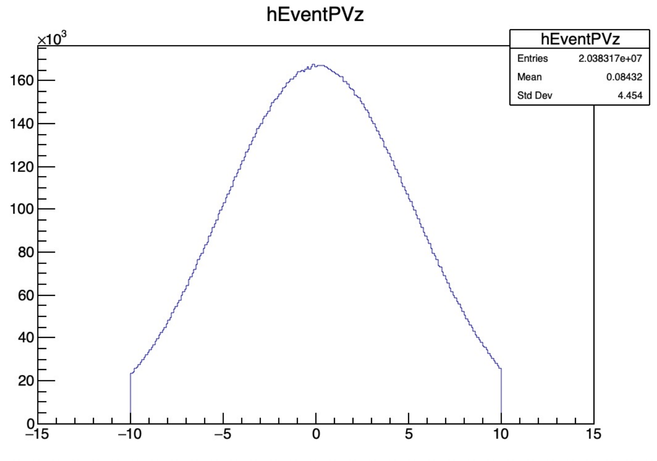
- This needs to be declared **inside the task struct (but outside process) – be careful with the location!**
- Then you need to change your subscription to reflect the filtering:

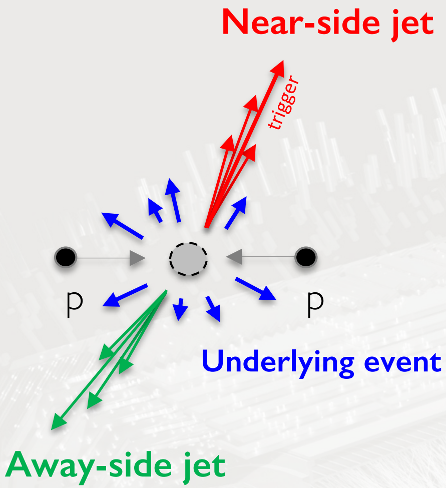
```
void process(soa::Filtered<aod::DrCollisions>::iterator const& collision, aod::DrTracks const& tracks)
```

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

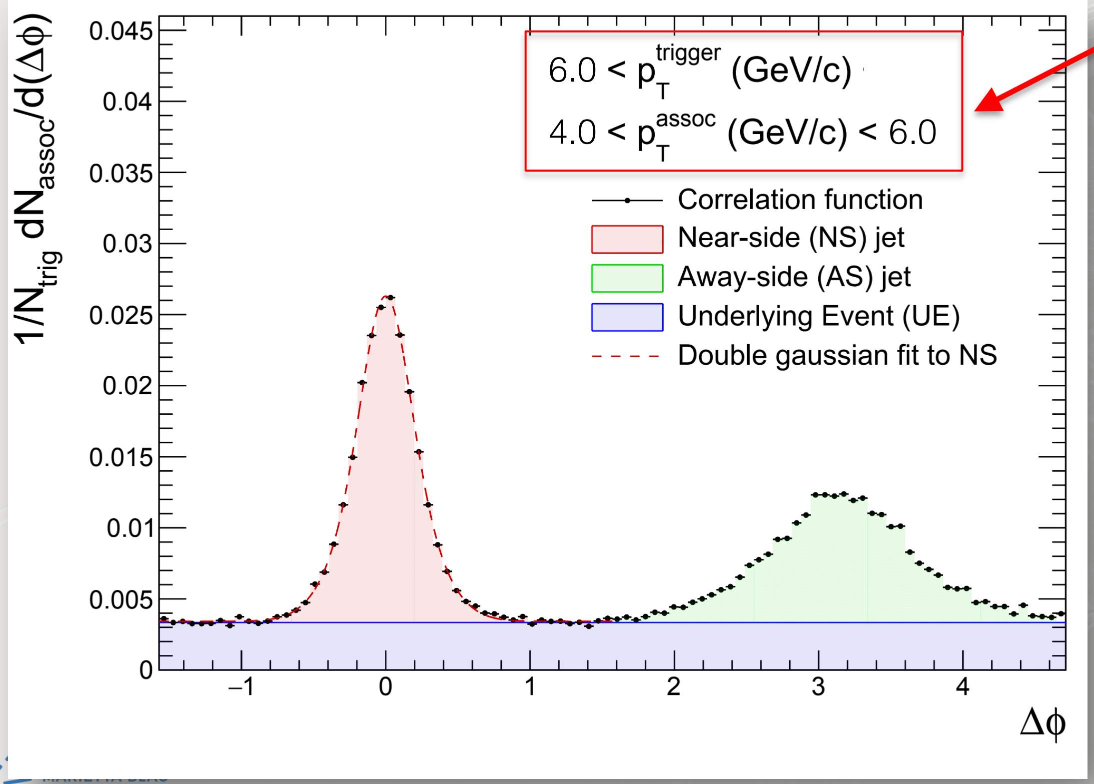
```
// Warning: the Filter declaration requires a new header! Add this:
#include "Framework/ASoAHelpers.h"
// You should also add this:
using namespace o2::framework::expressions;
```







Towards correlation functions: the logic



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

Bulk operations via filtering and partitioning!

- You'll need to add a SliceCache to your task struct by doing:

```
SliceCache cache;
```

- Now, still within the task struct, let's define two partitions: associated and trigger momentum ranges!

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

- You can (should!) also make the suggested pt cuts configurable: **associatedMinPt, associatedMaxPt, triggerMinPt**
- IMPORTANT: partitions are not grouped by default. You have to group them inside your process function:

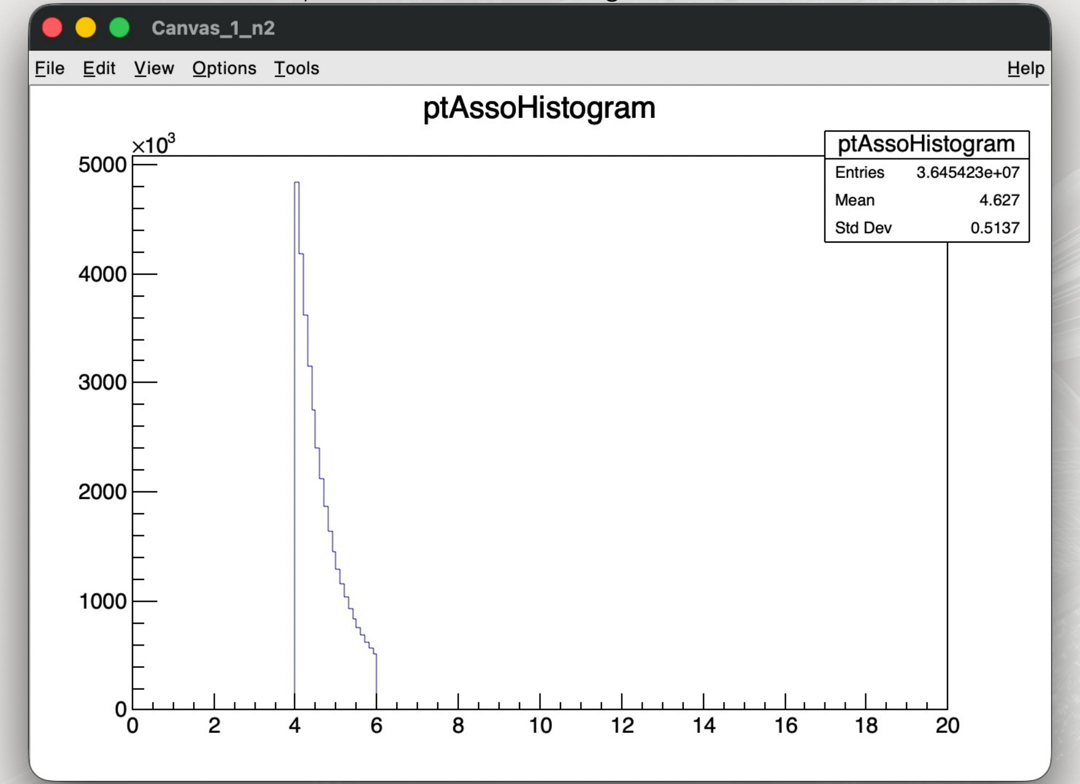
```
//partitions are not grouped by default!
auto assoTracksThisCollision = associatedTracks->sliceByCached(aod::exampleTrackSpace::drCollisionId, collision.globalIndex(), cache);
auto trigTracksThisCollision = triggerTracks->sliceByCached(aod::exampleTrackSpace::drCollisionId, collision.globalIndex(),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

- Now you can use these partitions to do any loop you like. For instance, we can fill two extra QA histograms:

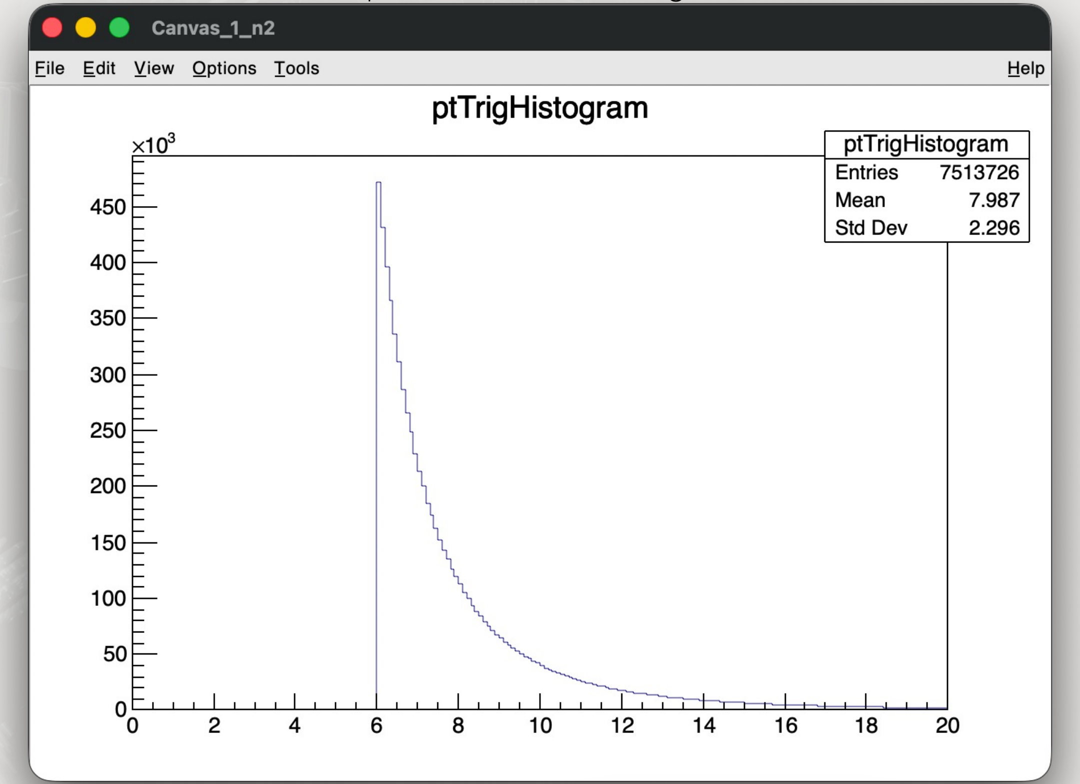
```
for (auto& track : assoTracksThisCollision)
  histos.fill(HIST("ptAssoHistogram"), track.pt());
for (auto& track : trigTracksThisCollision)
  histos.fill(HIST("ptTrigHistogram"), track.pt());
```

Output of track partition QA histograms

(These plots are from the large size file)



(These plots are from the large size file)



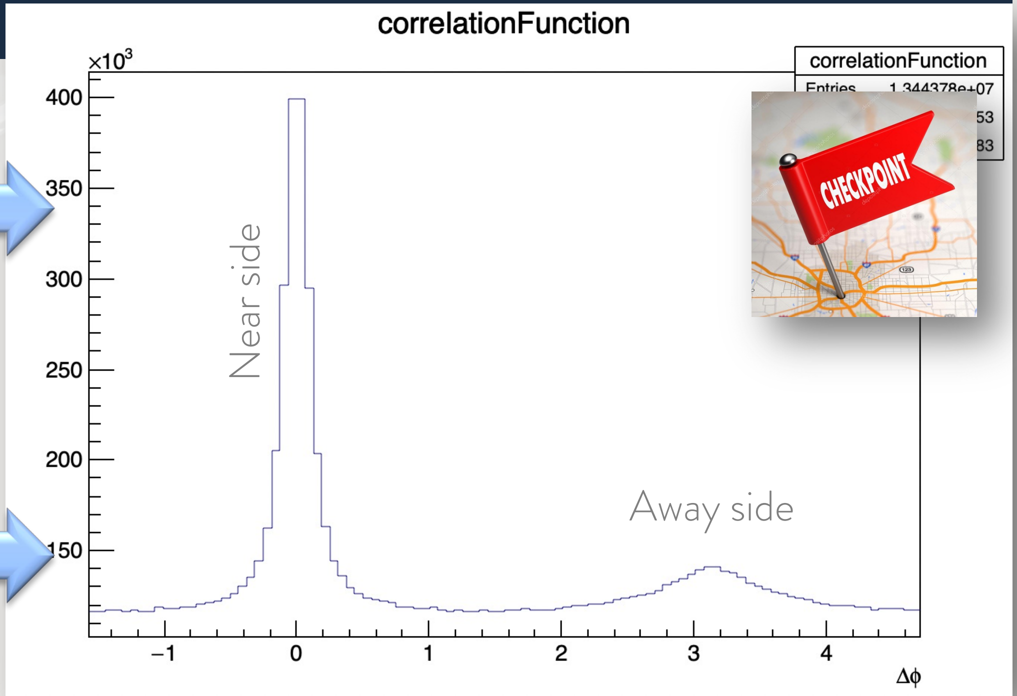
Nota bene: If you use filtering AND partitioning, parts are defined within the filtered sample!

Time to do correlations!

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

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

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



(This plot is from the large size file)

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

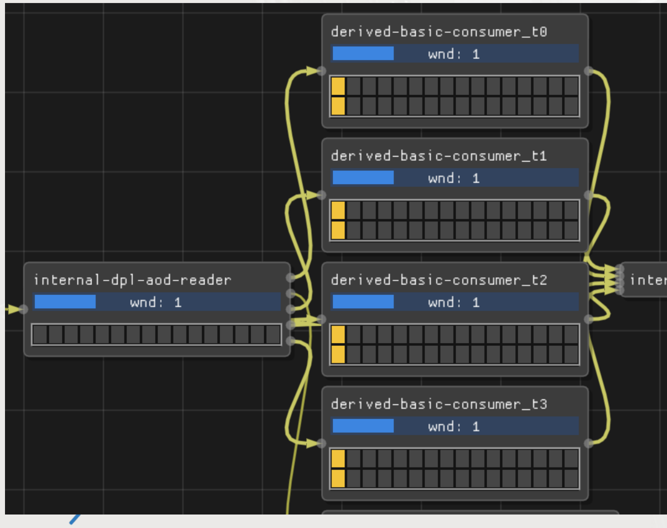
```
    for (auto const& [trigger, associated] :
combinations(o2::soa::CombinationsFullIndexPolicy(trigTracksThisCollision, assoTracksThisCollision))) {
      histos.fill(HIST("correlationFunction"), ComputeDeltaPhi(trigger.phi(),associated.phi()));
    }
```

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

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

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

```
    for (auto const& [trigger, associated] :
combinations(o2::soa::CombinationsFullIndexPolicy(trigTracksThisCollision, assoTracksThisCollision))) {
      histos.fill(HIST("correlationFunction"), ComputeDeltaPhi(trigger.phi(),associated.phi()));
      histos.fill(HIST("correlationFunction2d"), ComputeDeltaPhi(trigger.phi(),associated.phi()),
trigger.eta() - associated.eta());
    }
```



(if you use the gui: example with 4 consumers)

- This is already a fundamental step in a correlation analysis! (For more: CF)
- In reality, event mixing corrections are necessary to account for the triangle-like shape (which is due to **acceptance**)
- If you want to analyse quickly, use pipelining!
  - **This is a great use case: threre's no other task except the consumer!**
- Add the option: `--pipeline=derived-basic-consumer:4`
  - This will use 4 CPUs by spawning 4 parallel copies of the task!
  - You can adjust as necessary, and you can use this for other studies too

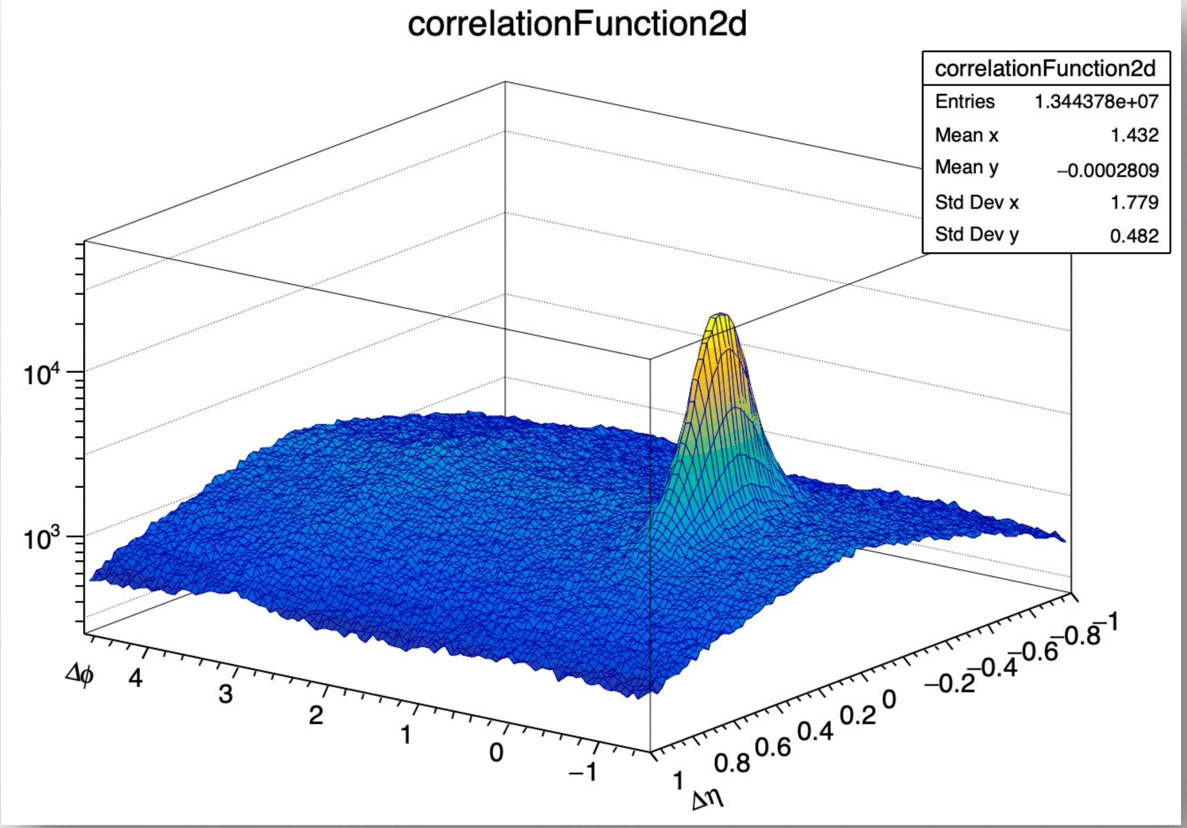
Two-dimensional correlation function with larger statistics

Pb-Pb pass5 data, 6x10^7 events

Not corrected for acceptance

$4.0 < p_T^{assoc}$ (GeV/c) < 6.0

$6.0 < p_T^{trigger}$ (GeV/c)



Solution to this exercise:
[derivedBasicProducer.cxx]

Bonus: writing a task that creates a table in memory

```
struct DerivedBasicProvider {
  // This marks that this task produces a table in memory / can be stored to disk
  Produces<aod::DrCollisions> outputCollisions;
  Produces<aod::DrTracks> outputTracks;

  // (…)
  void process(aod::Collision const& collision, myFilteredTracks const& tracks)
  {
    histos.fill(HIST("eventCounter"), 0.5);
    if (tracks.size() < 1 && skipUninterestingEvents)
      return;
    bool interestingEvent = false;
    for (const auto& track : tracks) {
      if (track.tpcNClsCrossedRows() < minTPCNClsCrossedRows)
        continue; // remove badly tracked
      interestingEvent = true;
    }
    if (!interestingEvent && skipUninterestingEvents)
      return;
    outputCollisions(collision.posZ());
    for (const auto& track : tracks) {
      if (track.tpcNClsCrossedRows() < minTPCNClsCrossedRows)
        continue; // remove badly tracked
      histos.get<TH1>(HIST("ptHistogram"))->Fill(track.pt());
      outputTracks(outputCollisions.lastIndex(), track.pt(), track.eta(), track.phi()); // all that I need for posterior analysis!
    }
  }
};
```

This task will produce a table in memory
(it can be marked to be saved to disk!)

This command populates the table, with the order of variables matching the order of the definition of the table in the data model

Bonus: marking a certain table for saving to disk

- Any task that creates a table can be used to generate derived data!
- Let's use the derivedBasicProvider.cxx task as an example.
- One needs to mark the table to be saved as derived in the execution. The command line needs to have:

```
o2-analysistutorial-derived-basic-provider -b --configuration json://configuration.json | o2-analysis-event-selection-service -b --configuration json://configuration.json | o2-analysis-propagationservice -b --configuration json://configuration.json --aod-file @input_data.txt --aod-writer-json OutputDirector.json
```

```
{ "OutputDirector": {
  "debug_mode": true,
  "resfile": "AO2D",
  "OutputDescriptors": [
    {
      "table": "AOD/DRCOLLISION/0"
    },
    {
      "table": "AOD/DRTRACK/0"
    }
  ],
  "ntfmerge": 1
} }
```

- The OutputDirector.json file lists the tables that are to be saved to the derived AO2D
- Think of it as a 'milestone' in processing!
- Any table created by any task **can be saved or consumed in RAM**
  - Extremely useful flexibility!
- Note that derived data is useful if:
  - Only little data is saved (> 50x reduction in data volume)
  - Heavy processing can be spared

Bonus / Easter Eggs / Tips

Further tricks with indices: manual slicing

- Many operations of slicing etc can be done manually (but still using table functionality)!
- Let's do an example collision grouping by hand! For this, our task struct needs to have a Preslice declaration:

```
// For manual sliceBy
Preslice<aod::Tracks> tracksPerCollisionPreslice = o2::aod::track::collisionId;
```

- Now one can do the collision – track grouping manually:

```
void process(aod::Collisions const& collisions, myFilteredTracks const& tracks)
{
    for (const auto& collision : collisions) {
        //group tracks manually
        const uint64_t collIdx = collision.globalIndex();
        auto trackTable_thisCollision = tracks.sliceBy(tracksPerCollisionPreslice, collIdx);
        histos.fill(HIST("eventCounter"), 0.5);
        for (auto& track : trackTable_thisCollision) {
            // (…)
        }
    }
}
```

- This is a simple example, but this is very powerful as a general method!

Further tricks with axes: ConfigurableAxis

- 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});
```

- Using hyperloop, you can then change the binning scheme from fixed-width to variable and adjust to your liking using the web interface!