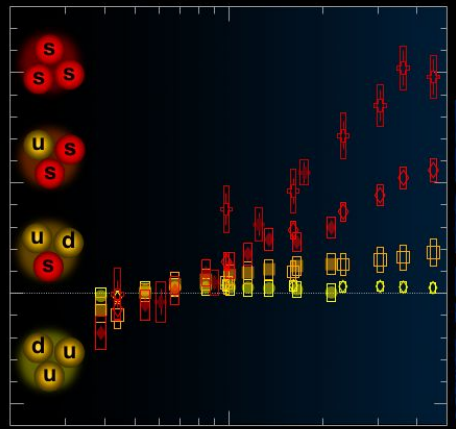
Strangeness hands-on

O2 Analysis Tutorial 12/11/2025



Romain Schotter

Austrian Academy of Sciences, Austria

romain.schotter@cern.ch

Outline

- In this hands-on session, we will focus on the analysis of cascades in OO collisions using **derived data**

- You will learn how to
  - Use standard strangeness derived data
  - Structure your analysis task
  - Reconstruct cascades
  - Apply (TPC and TOF) PID, topological and kinematic selections
  - Access MC information of the cascades

- For a similar tutorial using original data, please have a look at the back-up slides

Romain Schotter (romain.schotter@cern.ch)

Preparation: strangeness derived data

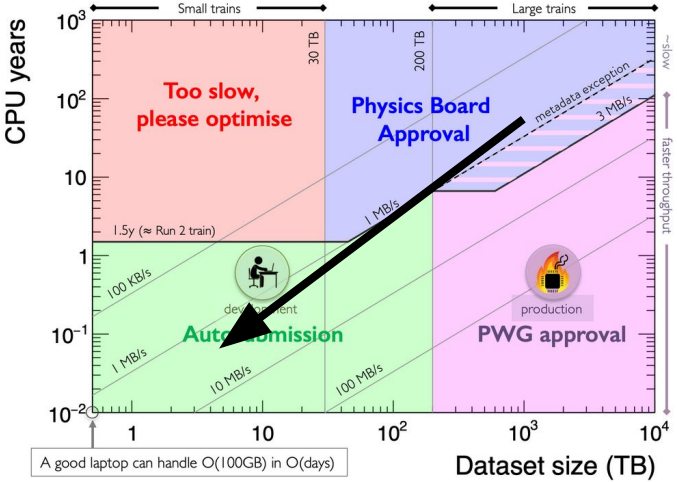
- **Derived data** are (*derived*) AO2D.root files produced with a task that creates given tables by processing other (*original*) AO2D.root files

- Only information needed for your analysis are stored in derived data
  → reduce the size of AO2D.root files
  → **speed up the execution** by skipping part of the workflows of your analysis (typically the computational intensive parts, ie track propagation, etc)

- Two types of derived data:

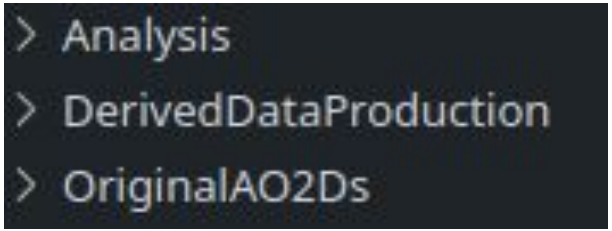
  - **Self contained:** contain all the information needed for your analysis

  - **Linked:** contain additional information wrt the original AO2D.root files and hence require access to the parent AO2D.root files



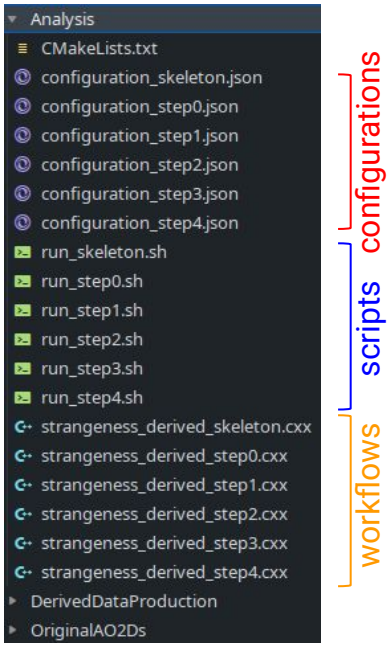
Preparation

- All the necessary scripts and workflows for this tutorial can be found in the [O2Physics/Tutorials/PWGLF/Strangeness/Derived/]
- The json files (and more!) can be found here: https://cernbox.cern.ch/s/0SFcdrxp56LkgDQ



Preparation

- All the necessary scripts and workflows for this tutorial can be found in the [O2Physics/Tutorials/PWGLF/Strangeness/Derived/]
- The json files (and more!) can be found here: https://cernbox.cern.ch/s/0SFcdrxp56LkgDQ



All files (scripts, workflows, configurations) for running the analysis task are stored in the **Analysis** folder

Preparation

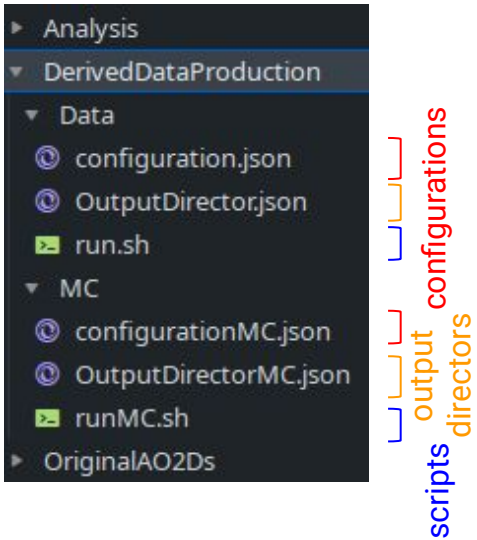
- All the necessary scripts and workflows for this tutorial can be found in the [O2Physics/Tutorials/PWGLF/Strangeness/Derived/]
- The json files (and more!) can be found here: https://cernbox.cern.ch/s/0SFcdrxp56LkgDQ

This tutorial does NOT aim at producing standard derived data (done centrally)

Twiki listing the existing strangeness derived data

Preparation

- All the necessary scripts and workflows for this tutorial can be found in the [O2Physics/Tutorials/PWGLF/Strangeness/Derived/]
- The json files (and more!) can be found here: https://cernbox.cern.ch/s/0SFcdrxp56LkgDQ



This tutorial does NOT aim at producing standard derived data (done centrally)
If you are curious:

All files (scripts, output directors, configurations) for producing the derived data are stored in the **DerivedDataProduction** folder

The original data are stored in the **OriginalAO2Ds** folder

Preparation

- All the necessary scripts and workflows for this tutorial can be found in the O2Physics/Tutorials/PWGLF/Strangeness/Derived/
- The json files (and more!) can be found here: https://cernbox.cern.ch/s/0SFcdrxp56LkgDQ

- Create a folder to store the derived data
- Download the file from cernbox (~57 MB)
- Alternatively, download the files from lxplus

```
rsync -r -u --progress USERNAME@lxplus.cern.ch:/afs/cern.ch/user/r/rschotte/cernbox/O2AT2025/Derived/DerivedDataProduction .
```

Enter your lxplus password

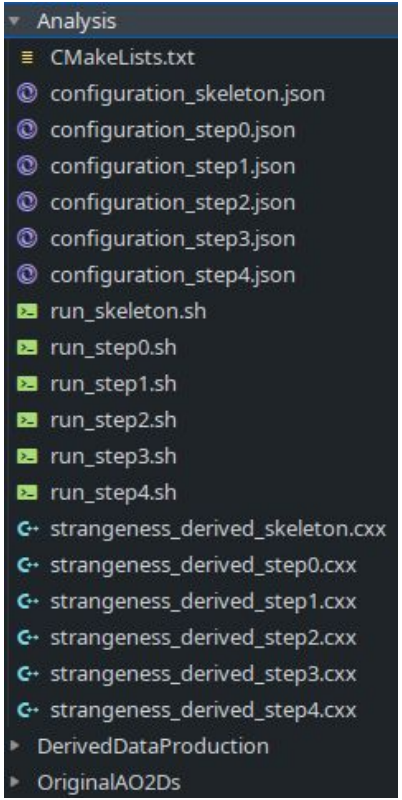
Preparation: running an analysis task

- Once the derived data have been downloaded, we can start writing the analysis task

- The idea is to start with the `strangeness_derived_skeleton.cxx` file and improve it up to step4

- Checkpoints for each step (X) are here for your convenience (`strangeness_derived_stepX.cxx`), with the running shell script (`run_stepX.sh`) and the corresponding configurations (`configuration_stepX.json`)

- Let's have a look at `strangeness_derived_skeleton.cxx`



Preparation: running an analysis task

`strangeness_derived_skeleton.cxx` loops over all events, selects the ones with |z| < 10 cm and `sel8` flag (FT0 vertex compatible with FT0-A and FT0-C timing info), fills an histogram with the *z*-vertex position

```cpp
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"
#include "Common/DataModel/EventSelection.h"
#include "PWGLF/DataModel/LFStrangenessTables.h"

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

struct strangeness_derived_tutorial {
  // Histograms are defined with HistogramRegistry
  HistogramRegistry rEventSelection{"eventSelection", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};

  // Configurable for histograms
  Configurable<int> nBins{"nBins", 100, "N bins in all histos"};

  // Configurable for event selection
  Configurable<float> cutzvertex{"cutzvertex", 10.0f, "Accepted z-vertex range (cm)"};

  void init(InitContext const&)
  {
    // Axes
    AxisSpec vertexZAxis = {nBins, -15., 15., "vrtx_{Z} [cm]"};

    // Histograms
    // Event selection
    rEventSelection.add("hVertexZRec", "hVertexZRec", {HistType::kTH1F, {vertexZAxis}});
  }
```

Include required **headers**

Histogram registries and **Configurables** are declared before the **init** function

Create axis and add needed **histograms** to the registries

Romain Schotter (romain.schotter@cern.ch)

Preparation: running an analysis task

```
47    // Defining filters for events (event selection)
48    // Processed events will be already fulfilling the event selection requirements
49    Filter eventFilter = (o2::aod::evsel::sel8 == true);
50    Filter posZFilter = (nabs(o2::aod::collision::posZ) < cutzvertex);
51
52    void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision)
53    {
54      // Fill the event counter
55      rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());
56    }
57  };
58
59  WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
60  {
61    return WorkflowSpec{
62      adaptAnalysisTask<strangeness_derived_tutorial>(cfgc)};
63  }
```

Define **Filters** for basic event selection

Subscribe to an iterator of a table joining **aod::StraCollisions** (=basic collision properties) **and aod::StraEvSels** (= all necessary event selection variables) to loop over the selected collisions

Because of the iterator, the **process()** function is called once for each selected event and fills the vertex *z*-position histogram

Romain Schotter (romain.schotter@cern.ch)

Preparation: running an analysis task

- Let's run the analysis task:

```
bash run_skeleton.sh
```

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

o2-analysistutorial-lf-strangeness-derived-skeleton ${OPTION} --aod-file @input_data.txt > "$LOGFILE" 2>&1
```

Preparation: running an analysis task

- Let's run the analysis task:

```
bash run_skeleton.sh
```

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

o2-analysistutorial-lf-strangeness-derived-skeleton ${OPTION} --aod-file @input_data.txt > "$LOGFILE" 2>&1
```

Workflow with all the helper tasks

Preparation: running an analysis task

- Let's run the analysis task:

```
bash run_skeleton.sh
```

```
#!/bin/bash
# log file where the terminal output will be saved
STEP="skeleton"
LOGFILE="log-${STEP}.txt"

#directory of this script
DIR_THIS=$PWD

OPTION="-b --configuration json://configuration_skeleton.json"

o2-analysistutorial-lf-strangeness-derived-skeleton ${OPTION} --aod-file @input_data.txt > "$LOGFILE" 2>&1
```

The whole log output is transferred into the **log-${STEP}.txt** file

Workflow with all the helper tasks

Romain Schotter (romain.schotter@cern.ch)

Preparation: running an analysis task

- Let's run the analysis task: `bash run_skeleton.sh`

```
#!/bin/bash
# log file where the terminal output will be saved
STEP="skeleton"
LOGFILE="log-${STEP}.txt"

#directory of this script
DIR_THIS=$PWD

OPTION="-b --configuration json://configuration_skeleton.json"

o2-analysistutorial-lf-strangeness-derived-skeleton ${OPTION} --aod-file @input_data.txt > "$LOGFILE" 2>&1

# report status
rc=$?
if [ $rc -eq 0 ]; then
    echo "No problems!"
    mkdir -p "${DIR_THIS}/results/${STEP}"
    mv AnalysisResults.root "${DIR_THIS}/results/${STEP}/AnalysisResults.root"
    mv dpl-config.json "${DIR_THIS}/results/${STEP}/${STEP}.json"
else
    echo "Error: Exit code ${rc}"
    echo "Check the log file ${LOGFILE}"
    exit ${rc}
fi
```

The whole log output is transferred into the **log-${STEP}.txt** file

Workflow with all the helper tasks

If the task finishes successfully, **AnalysisResults.root**, **AO2D.root**, and **json files** are moved to the `results` folder

Preparation: running an analysis task

- Let's run the analysis task: `bash run_skeleton.sh`

```
#!/bin/bash
# log file where the terminal output will be saved
STEP="skeleton"
LOGFILE="log-${STEP}.txt"

#directory of this script
DIR_THIS=$PWD

OPTION="-b --configuration json://configuration_skeleton.json"

o2-analysistutorial-lf-strangeness-derived-skeleton ${OPTION} --aod-file @input_data.txt > "$LOGFILE" 2>&1

# report status
rc=$?
if [ $rc -eq 0 ]; then
    echo "No problems!"
    mkdir -p "${DIR_THIS}/results/${STEP}"
    mv AnalysisResults.root "${DIR_THIS}/results/${STEP}/AnalysisResults.root"
    mv dpl-config.json "${DIR_THIS}/results/${STEP}/${STEP}.json"
else
    echo "Error: Exit code ${rc}"
    echo "Check the log file ${LOGFILE}"
    exit ${rc}
fi
```

The whole log output is transferred into the **log-${STEP}.txt** file

Workflow with all the helper tasks

If the task finishes successfully, **AnalysisResults.root**, **AO2D.root**, and **json files** are moved to the `results` folder

All shell running script follow the same structure in the tutorial!

Preparation: running an analysis task

- Let's run the analysis task: `bash run_skeleton.sh`

  → `No problems!` must be printed

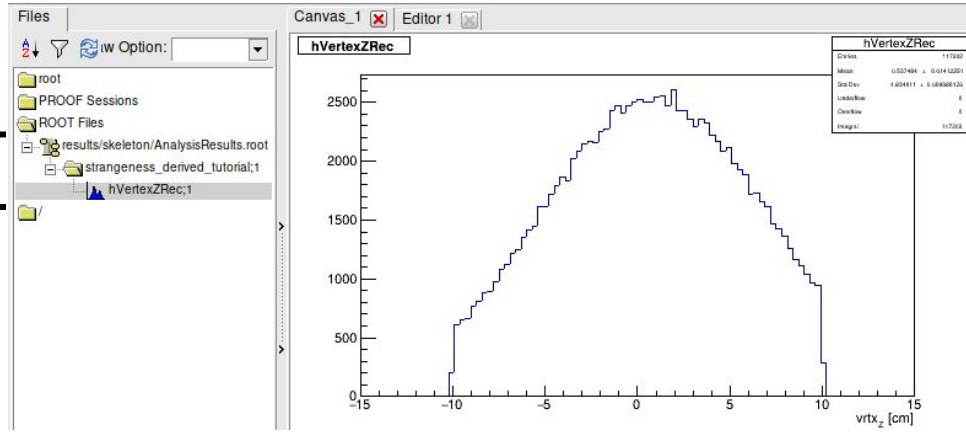
  All output are contained in the **results** folder:

  - `AnalysisResults.root`

Standard output

It contains the output of each task

Each task (including the intermediate tasks) in the workflow can create and fill **histograms**



Preparation: running an analysis task

- Let's run the analysis task: `bash run_skeleton.sh`

  → `No problems!` must be printed

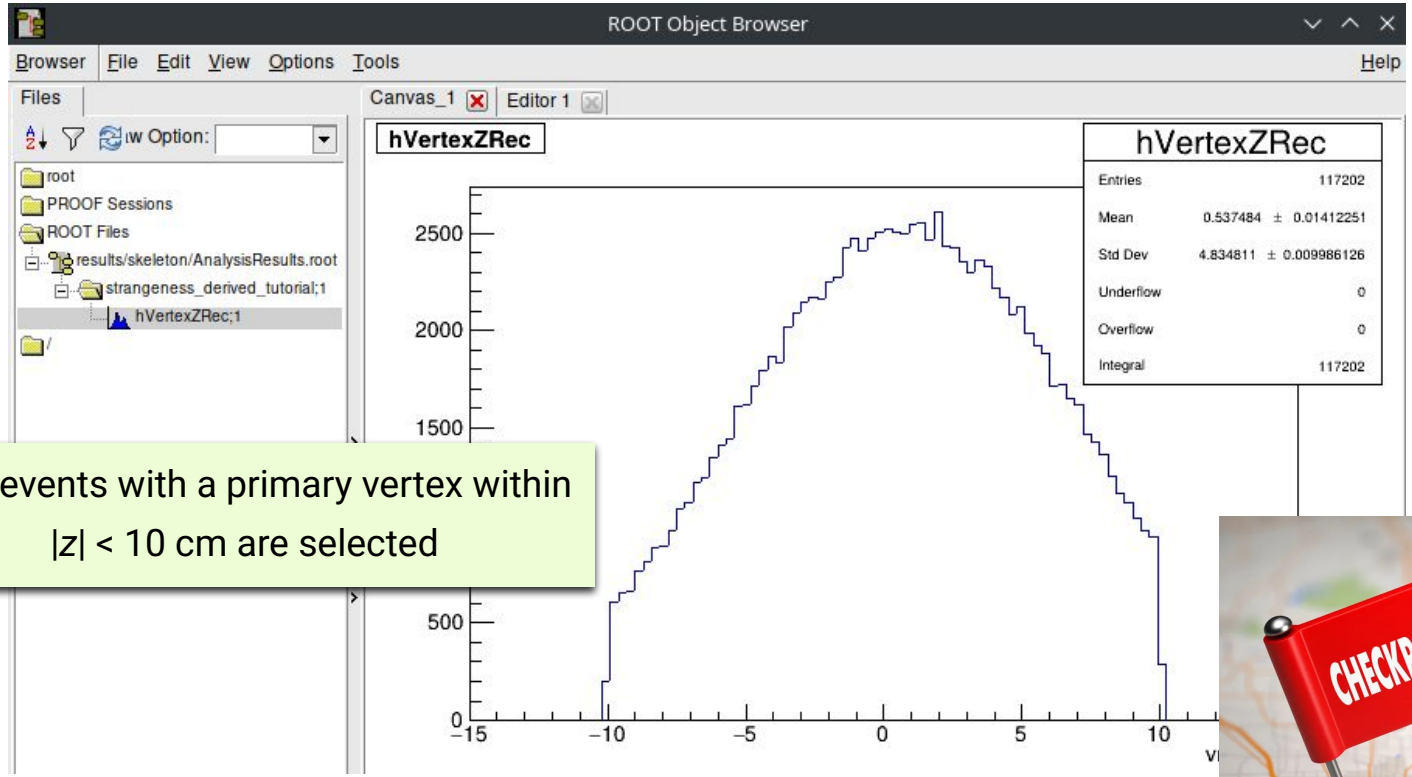
  All output are contained in the **results** folder:

  - `AnalysisResults.root`
  - `dpl-config.json`

A copy of the `configuration.json` file

Romain Schotter (romain.schotter@cern.ch)

Preparation: running an analysis task



Only events with a primary vertex within $|z| < 10$ cm are selected

Step 0: looping over cascades

- Starting from `strangeness_derived_skeleton.cxx`, we want to loop over all cascades in the selected events and fill a invariant mass histogram

```
struct strangeness_derived_tutorial {
  // Histograms are defined with HistogramRegistry
  HistogramRegistry rEventSelection{"eventSelection", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
  HistogramRegistry rXi{"xi", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
  HistogramRegistry rOmega{"omega", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
```

Create a "**xi**" and "**omega**" **HistogramRegistry**

```
  void init(InitContext const&)
  {
    // Axes
    AxisSpec XiMassAxis = {100, 1.28f, 1.36f, "#it{M}_{inv} [GeV/#it{c}^{2}]"};
    AxisSpec OmegaMassAxis = {100, 1.63f, 1.7f, "#it{M}_{inv} [GeV/#it{c}^{2}]"};
    AxisSpec vertexZAxis = {nBins, -15., 15., "vrtx_{Z} [cm]"};

    // Histograms
    // Event selection
    rEventSelection.add("hVertexZRec", "hVertexZRec", {HistType::kTH1F, {vertexZAxis}});

    // Xi/Omega reconstruction
    rXi.add("hMassXi", "hMassXi", {HistType::kTH1F, {XiMassAxis}});

    rOmega.add("hMassOmega", "hMassOmega", {HistType::kTH1F, {OmegaMassAxis}});
  }
```

Create dedicated axes for Xi and Omega

Create invariant mass histograms for Xi and Omega and store them in their corresponding **HistogramRegistry**

Step 0: looping over cascades

- Starting from `strangeness_derived_skeleton.cxx`, we want to loop over all cascades in the selected events and fill a invariant mass histogram

```
void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             aod::CascCores const& Cascades)
{
  // Fill the event counter
  rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());
```

Subscribe to the cascade tables (**aod::CascCores**)

- ***Instructions:***
  - Subscribe to the cascade tables (have a look at `LFStrangenessTables.h`)
  - For each event, loop over all cascades,
  - For each mass hypothesis, fill a histogram with the invariant mass
  - Store the histograms in the corresponding **HistogramRegistry**

- ***Hints:*** for the cascade info, look at the **aod::CascCores** table and the getters .mXi() and .mOmega()

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 0: looping over cascades

- Starting from `strangeness_derived_skeleton.cxx`, we want to loop over all cascades in the selected events and fill a invariant mass histogram

```
void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             aod::CascCores const& Cascades)
{
  // Fill the event counter
  rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());

  // Cascades
  for (const auto& casc : Cascades) {
    rXi.fill(HIST("hMassXi"), casc.mXi());
    rOmega.fill(HIST("hMassOmega"), casc.mOmega());
  }
}
```

For each event, loop over all cascades

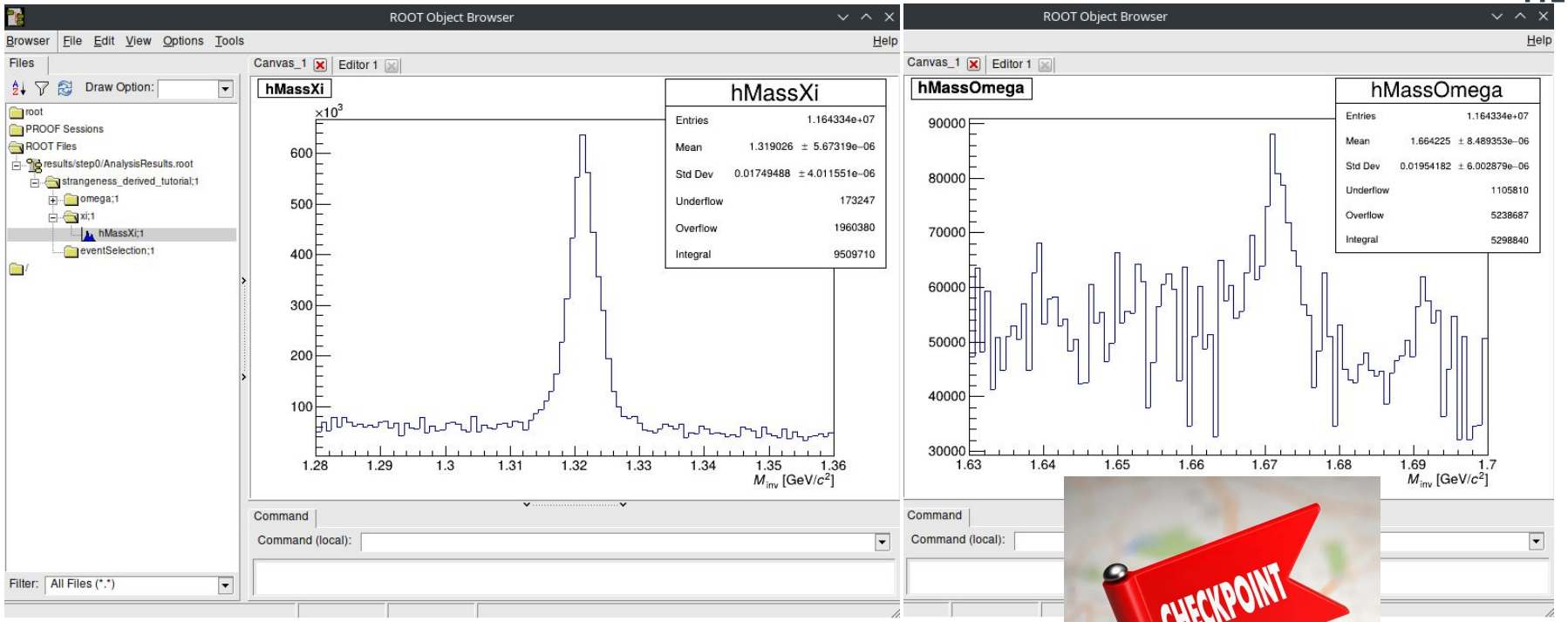
For each mass hypothesis, fill a histogram with the invariant mass

- ***Instructions:***
  - Subscribe to the cascade tables (have a look at `LFStrangenessTables.h`)
  - For each event, loop over all cascades,
  - For each mass hypothesis, fill a histogram with the invariant mass
  - Store the histograms in the corresponding **HistogramRegistry**

- ***Hints:*** for the cascade info, look at the **aod::CascCores** table and the getters .mXi() and .mOmega()

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 0: looping over cascades



Invariant mass peak is already visible for Ξ and Ω with the derived data producer **pre-selections**

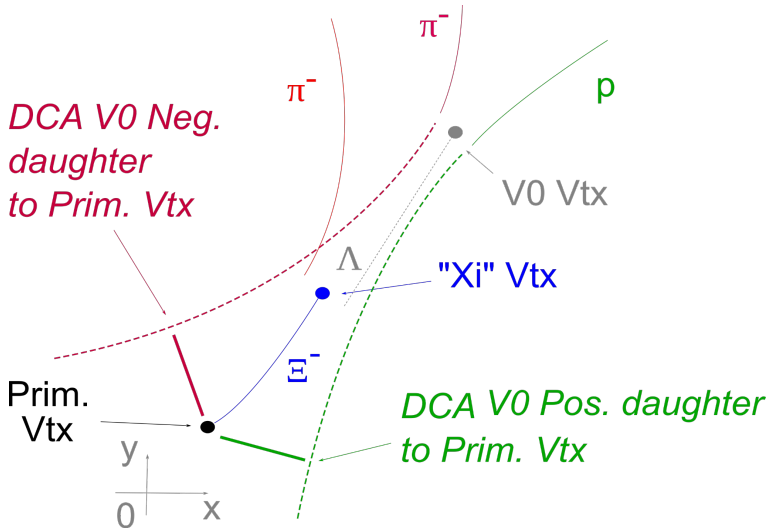
Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

$$\begin{cases} \Lambda \to \mathrm{p}\pi^- \\ \overline{\Lambda} \to \overline{\mathrm{p}}\pi^+ \end{cases}$$ c τ = 7.89 cm B.R. 63.9%

$$\begin{cases} \Xi^- \to \Lambda\pi^- \\ \overline{\Xi}^+ \to \overline{\Lambda}\pi^+ \end{cases}$$ c τ = 4.91 cm B.R. 99.9%

$$\begin{cases} \Omega^- \to \Lambda\mathrm{K}^- \\ \overline{\Omega}^+ \to \overline{\Lambda}\mathrm{K}^+ \end{cases}$$ c τ = 2.461 cm B.R. 67.8%

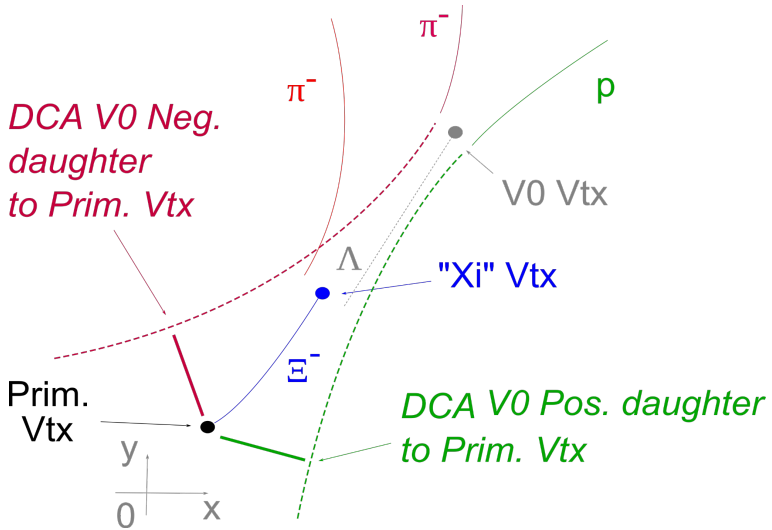


Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

V0 mass window	< 0.008 GeV/c2
DCA pos./neg. to prim. vtx	> 0.06 cm
DCA V0 to prim vtx	> 1.0 cm
DCA between V0. daughters	< 1.0 cm
V0 decay radius	> 1.2 cm
V0 cos of pointing angle	> 0.97

Competing mass rejection (only for Ω)	> 0.008 GeV/c2
DCA bach. to prim. vtx	> 0.06 cm
DCA between casc. daughters	< 1.0 cm
Casc. decay radius	> 0.5 cm
Casc. cos of pointing angle	> 0.998



Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

		Getter	Column type
V0 mass window	< 0.008 GeV/c2	mLambda () - o2::constants::physics::MassLambda	dynamic
DCA pos./neg. to prim. vtx	> 0.06 cm	dcapos/negtopv ()	standard
DCA V0 to prim vtx	> 1.0 cm	dcav0topv (*arg1, arg2, arg3*)	dynamic
DCA between V0. daughters	< 1.0 cm	dcaV0daughters ()	standard
V0 decay radius	> 1.2 cm	v0radius ()	dynamic
V0 cos of pointing angle	> 0.97	v0cospa (*arg1, arg2, arg3*)	dynamic
Competing mass rejection (only for Ω)	> 0.008 GeV/c2	mXi () - o2::constants::physics::MassXiMinus	standard
DCA bach. to prim. vtx	> 0.06 cm	dcabachtopv ()	standard
DCA between casc. daughters	< 1.0 cm	dcacascdaughters ()	standard
Casc. decay radius	> 0.5 cm	cascradius ()	dynamic
Casc. cos of pointing angle	> 0.998	casccospa (*arg1, arg2, arg3*)	dynamic

Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

```
// Configurable parameters for cascade selection
Configurable<double> cascadesetting_cospa{"cascadesetting_cospa", 0.98, "Casc CosPA"};
Configurable<double> cascadesetting_v0cospa{"cascadesetting_v0cospa", 0.97, "V0 CosPA"};
Configurable<float> cascadesetting_dcacascdau{"cascadesetting_dcacascdau", 1.0, "DCA cascade daughters"};
Configurable<float> cascadesetting_dcav0dau{"cascadesetting_dcav0dau", 1.0, "DCA v0 daughters"};
Configurable<float> cascadesetting_dcabachtopv{"cascadesetting_dcabachtopv", 0.06, "DCA bachelor to PV"};
Configurable<float> cascadesetting_dcapostopv{"cascadesetting_dcapostopv", 0.06, "DCA positive to PV"};
Configurable<float> cascadesetting_dcanegtopv{"cascadesetting_dcanegtopv", 0.06, "DCA negative to PV"};
Configurable<float> cascadesetting_mindcav0topv{"cascadesetting_mindcav0topv", 0.01, "minimum V0 DCA to PV"};
Configurable<float> cascadesetting_cascradius{"cascadesetting_cascradius", 0.5, "cascradius"};
Configurable<float> cascadesetting_v0radius{"cascadesetting_v0radius", 1.2, "v0radius"};
Configurable<float> cascadesetting_v0masswindow{"cascadesetting_v0masswindow", 0.01, "v0 mass window"};
Configurable<float> cascadesetting_competingmassrej{"cascadesetting_competingmassrej", 0.008, "Competing mass rejection"};
```

Add **Configurable** for each cascade selection

Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

```
// Configurable parameters for cascade selection
Configurable<double> cascadesetting_cospa{"cascadesetting_cospa", 0.98, "Casc CosPA"};
Configurable<double> cascadesetting_v0cospa{"cascadesetting_v0cospa", 0.97, "V0 CosPA"};
Configurable<float> cascadesetting_dcacascdau{"cascadesetting_dcacascdau", 1.0, "DCA cascade daughters"};
Configurable<float> cascadesetting_dcav0dau{"cascadesetting_dcav0dau", 1.0, "DCA v0 daughters"};
Configurable<float> cascadesetting_dcabachtopv{"cascadesetting_dcabachtopv", 0.06, "DCA bachelor to PV"};
Configurable<float> cascadesetting_dcapostopv{"cascadesetting_dcapostopv", 0.06, "DCA positive to PV"};
Configurable<float> cascadesetting_dcanegtopv{"cascadesetting_dcanegtopv", 0.06, "DCA negative to PV"};
Configurable<float> cascadesetting_mindcav0topv{"cascadesetting_mindcav0topv", 0.01, "minimum V0 DCA to PV"};
Configurable<float> cascadesetting_cascradius{"cascadesetting_cascradius", 0.5, "cascradius"};
Configurable<float> cascadesetting_v0radius{"cascadesetting_v0radius", 1.2, "v0radius"};
Configurable<float> cascadesetting_v0masswindow{"cascadesetting_v0masswindow", 0.01, "v0 mass window"};
Configurable<float> cascadesetting_competingmassrej{"cascadesetting_competingmassrej", 0.008, "Competing mass rejection"};
```

Add **Configurable** for each cascade selection

- ***Instructions:***
  - Select the candidates passing the topological selections
  - For each mass hypothesis, fill a histogram with the invariant mass after cascade selection

- ***Hints:***
  1. Have a look at the **aod::Casccores** table in `LFStrangenessTables.h` to find the columns storing the selection variables
  2. you can use **Filter** to select the candidates but it cannot cut on dynamic columns

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

```
// Configurable parameters for cascade selection
Configurable<double> cascadesetting_cospa{"cascadesetting_cospa", 0.98, "Casc CosPA"};
Configurable<double> cascadesetting_v0cospa{"cascadesetting_v0cospa", 0.97, "V0 CosPA"};
Configurable<float> cascadesetting_dcacascdau{"cascadesetting_dcacascdau", 1.0, "DCA cascade daughters"};
Configurable<float> cascadesetting_dcav0dau{"cascadesetting_dcav0dau", 1.0, "DCA v0 daughters"};
Configurable<float> cascadesetting_dcabachtopv{"cascadesetting_dcabachtopv", 0.06, "DCA bachelor to PV"};
Configurable<float> cascadesetting_dcapostopv{"cascadesetting_dcapostopv", 0.06, "DCA positive to PV"};
Configurable<float> cascadesetting_dcanegtopv{"cascadesetting_dcanegtopv", 0.06, "DCA negative to PV"};
Configurable<float> cascadesetting_mindcav0topv{"cascadesetting_mindcav0topv", 0.01, "minimum V0 DCA to PV"};
Configurable<float> cascadesetting_cascradius{"cascadesetting_cascradius", 0.5, "cascradius"};
Configurable<float> cascadesetting_v0radius{"cascadesetting_v0radius", 1.2, "v0radius"};
Configurable<float> cascadesetting_v0masswindow{"cascadesetting_v0masswindow", 0.01, "v0 mass window"};
Configurable<float> cascadesetting_competingmassrej{"cascadesetting_competingmassrej", 0.008, "Competing mass rejection"};
```

Add **Configurable** for each cascade selection

```
// Filters on Cascades
// Cannot filter on dynamic columns
Filter preFilterCascades = (aod::cascdata::dcaV0daughters < cascadesetting_dcav0dau &&
                            nabs(aod::cascdata::dcapostopv) > cascadesetting_dcapostopv &&
                            nabs(aod::cascdata::dcanegtopv) > cascadesetting_dcanegtopv &&
                            nabs(aod::cascdata::dcabachtopv) > cascadesetting_dcabachtopv &&
                            aod::cascdata::dcacascdaughters < cascadesetting_dcacascdau);

void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             soa::Filtered<aod::CascCores> const& Cascades)
```

Filter on **regular** columns

Step 1: apply topological selections

```
if (casc.casccosPA(collision.posX(), collision.posY(), collision.posZ()) < cascadesetting_cospa)
  continue;
if (casc.v0cosPA(collision.posX(), collision.posY(), collision.posZ()) < cascadesetting_v0cospa)
  continue;
if (std::abs(casc.mLambda() - o2::constants::physics::MassLambda) > cascadesetting_v0masswindow)
  continue;
if (casc.dcav0topv(collision.posX(), collision.posY(), collision.posZ()) < cascadesetting_mindcav0topv)
  continue;
if (casc.cascradius() < cascadesetting_cascradius)
  continue;
if (casc.v0radius() < cascadesetting_v0radius)
  continue;
```

Cut on **dynamic** columns in the loop

```
// Fill histograms! (if possible)
rXi.fill(HIST("hMassXiSelected"), casc.mXi());
if (std::abs(casc.mXi() - o2::constants::physics::MassXiMinus) > cascadesetting_competingmassrej) { // competing mass rejection, only in case of Omega
  rOmega.fill(HIST("hMassOmegaSelected"), casc.mOmega());
```

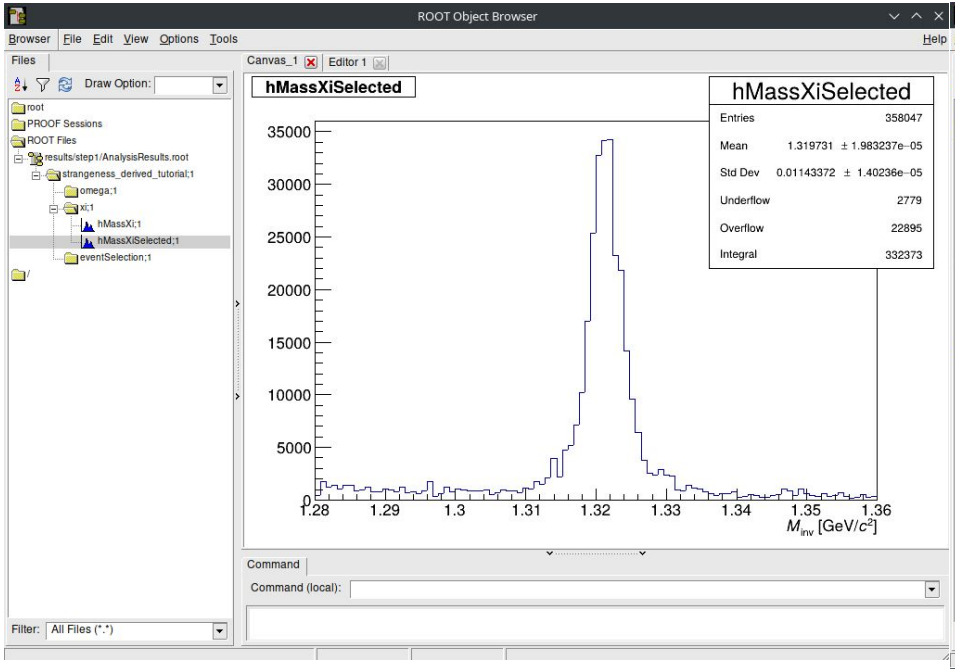
Fill invariant mass histograms <u>after</u> the topological selections

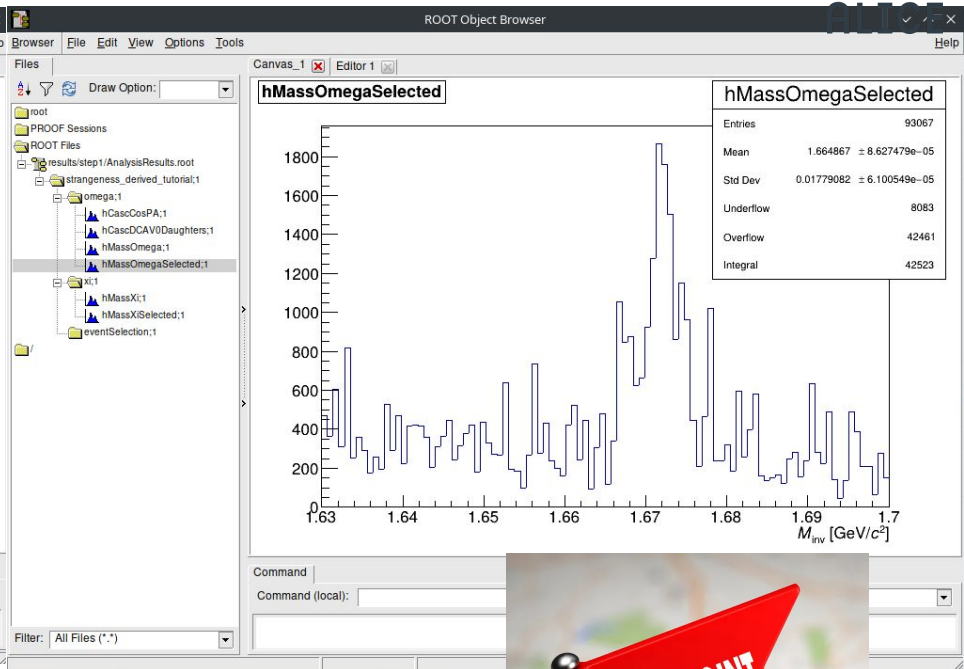
```
"strangeness_derived_tutorial": {
  "nBins": "100",
  "cutzvertex": "10",
  "cascadesetting_cospa": "0.998",
  "cascadesetting_v0cospa": "0.97",
  "cascadesetting_dcacascdau": "1",
  "cascadesetting_dcav0dau": "1",
  "cascadesetting_dcabachtopv": "0.06",
  "cascadesetting_dcapostopv": "0.06",
  "cascadesetting_dcanegtopv": "0.06",
  "cascadesetting_mindcav0topv": "0.01",
  "cascadesetting_cascradius": "0.5",
  "cascadesetting_v0radius": "1.2",
  "cascadesetting_v0masswindow": "0.008",
  "cascadesetting_competingmassrej": "0.008"
},
```

Provide the appropriate cuts in the **json configuration file**

The variables must have the same name as their corresponding configurable!

Step 1: apply topological selections





- Background reduces after applying topological selections

Step 2: apply TPC PID selections

- We would like now to apply TPC particle identification (PID) selections to our cascade candidates

Mean energy loss is parametrized by Bethe-Bloch formula:

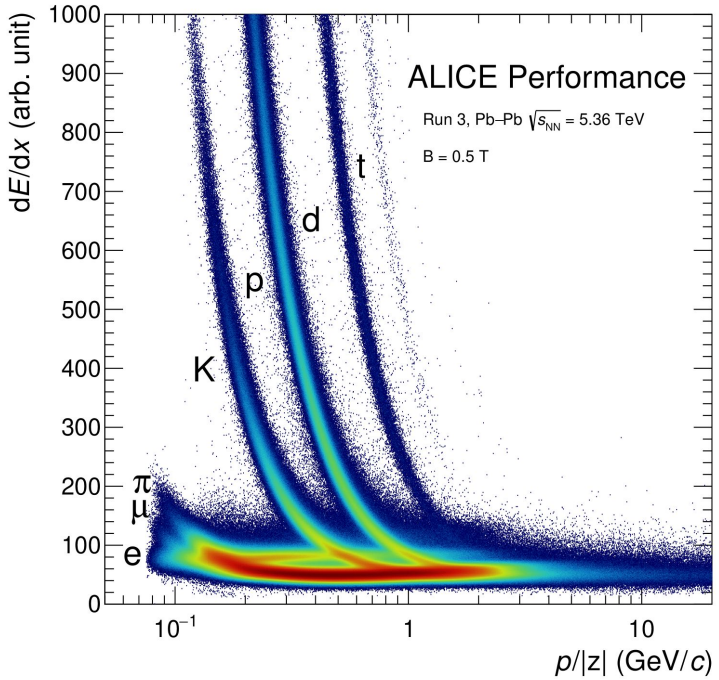
$$\begin{cases} \langle -\frac{dE}{dx} \rangle = K z^2 \frac{Z}{A} \frac{1}{\beta^2} \left[\frac{1}{2} \ln \frac{2 m_e c^2 \beta^2 \gamma^2 T_{\max}}{I} - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right] \\ \beta\gamma = \frac{p}{Mc} \end{cases}$$

The nature of the cascade decay daughter can be determined using the following PID estimator:

$$n_\sigma = \frac{\langle \mathrm{d}E/\mathrm{d}x \rangle_{\mathrm{meas.}} - \langle \mathrm{d}E/\mathrm{d}x \rangle_{\mathrm{exp.},i}}{\sigma_{\mathrm{TPC}}}$$

with

$$i = \mathrm{e},\ \mu,\ \pi,\ \mathrm{K},\ \mathrm{p},\ \mathrm{d},\ {}^3\mathrm{He},\ {}^4\mathrm{He}$$



Step 2: apply TPC PID selections

- V0/cascade tables reference the track table allowing access information about daughter tracks → if you try to check the PID of daughter tracks with reference to that table the code will fail!

```
float nsigma_bach = cascade.bachTrackExtra.tpcNSigmaPi();
float nsigma_pos = cascade.posTrackExtra.tpcNSigmaPr();
float nsigma_neg = cascade.negTrackExtra.tpcNSigmaPi();
```

Fails!

- You need to **de-reference the track** via a "***_as<>***" call, this allows you to look at the same index position in the more complete track table

```
using daughterTracks = soa::Join<aod::DauTrackExtras, aod::DauTrackTPCPIDs>

void process(soa::Join<aod::CascCores, aod::CascExtras> const& cascades,
             daughterTracks const&)
 {
   for (const auto& cascade : cascades) {
     float nsigma_bach = cascade.bachTrackExtra_as<daughterTracks>().tpcNSigmaPi();
     float nsigma_pos = cascade.posTrackExtra_as<daughterTracks>().tpcNSigmaPr();
     float nsigma_neg = cascade.negTrackExtra_as<daughterTracks>().tpcNSigmaPi();
   }
 }
```

Works!

Romain Schotter (romain.schotter@cern.ch)

Step 2: apply TPC PID selections

- We would like now to apply TPC particle identification (PID) selections to our cascade candidates

```
#include "PWGLF/DataModel/LFStrangenessPIDTables.h"
```

Header needed
`LFStrangenessPIDTables.h`

```
// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
Configurable<float> NSigmaTPCKaon{"NSigmaTPCKaon", 4, "NSigmaTPCKaon"};
Configurable<float> NSigmaTPCProton{"NSigmaTPCProton", 4, "NSigmaTPCProton"};
```

Add **Configurable** for each PID selection

```
using dauTracks = soa::Join<aod::DauTrackExtras, aod::DauTrackTPCPIDs>;

void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::CascCores, aod::CascExtras>> const& Cascades,
             dauTracks const&)
```

Subscribe to **aod::CascExtras** and **aod::DauTrackTPCPIDs** table

Romain Schotter (romain.schotter@cern.ch)

Step 2: apply TPC PID selections

- We would like now to apply TPC particle identification (PID) selections to our cascade candidates

```
#include "PWGLF/DataModel/LFStrangenessPIDTables.h"
```

Header needed
`LFStrangenessPIDTables.h`

```
// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
Configurable<float> NSigmaTPCKaon{"NSigmaTPCKaon", 4, "NSigmaTPCKaon"};
Configurable<float> NSigmaTPCProton{"NSigmaTPCProton", 4, "NSigmaTPCProton"};
```

Add **Configurable** for each PID selection

```
using dauTracks = soa::Join<aod::DauTrackExtras, aod::DauTrackTPCPIDs>;

void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::CascCores, aod::CascExtras>> const& Cascades,
             dauTracks const&)
```

Subscribe to **aod::CascExtras** and **aod::DauTrackTPCPIDs** table

- ***Instructions:***
  - De-reference (= typecast cascade to **dauTracks**) each cascade daughter tracks
  - Apply TPC PID selections (**n = 3σ**) on each decay daughters
  - For each mass hypothesis, fill a histogram with the invariant mass after cascade+PID selections

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 2: apply TPC PID selections

```
// Cascades
for (const auto& casc : Cascades) {
  const auto& bachDaughterTrackCasc = casc.bachTrackExtra_as<dauTracks>();
  const auto& posDaughterTrackCasc = casc.posTrackExtra_as<dauTracks>();
  const auto& negDaughterTrackCasc = casc.negTrackExtra_as<dauTracks>();
```

De-reference each cascade daughter tracks

Apply TPC PID selections

Romain Schotter (romain.schotter@cern.ch)

Step 2: apply TPC PID selections

```
// Cascades
for (const auto& casc : Cascades) {
  const auto& bachDaughterTrackCasc = casc.bachTrackExtra_as<dauTracks>();
  const auto& posDaughterTrackCasc = casc.posTrackExtra_as<dauTracks>();
  const auto& negDaughterTrackCasc = casc.negTrackExtra_as<dauTracks>();
```

De-reference each cascade daughter tracks

```
// PID selection
if (casc.sign() < 0) {
  if (std::abs(posDaughterTrackCasc.tpcNSigmaPr()) > NSigmaTPCProton) {
    continue;
  }
  if (std::abs(negDaughterTrackCasc.tpcNSigmaPi()) > NSigmaTPCPion) {
    continue;
  }
} else {
  if (std::abs(negDaughterTrackCasc.tpcNSigmaPr()) > NSigmaTPCProton) {
    continue;
  }
  if (std::abs(posDaughterTrackCasc.tpcNSigmaPi()) > NSigmaTPCPion) {
    continue;
  }
}
```

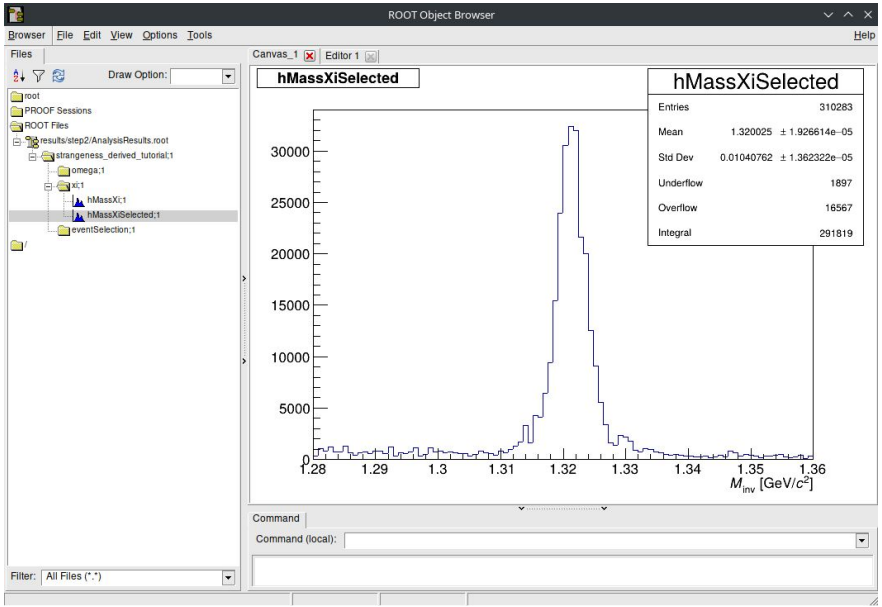
Apply TPC PID selections

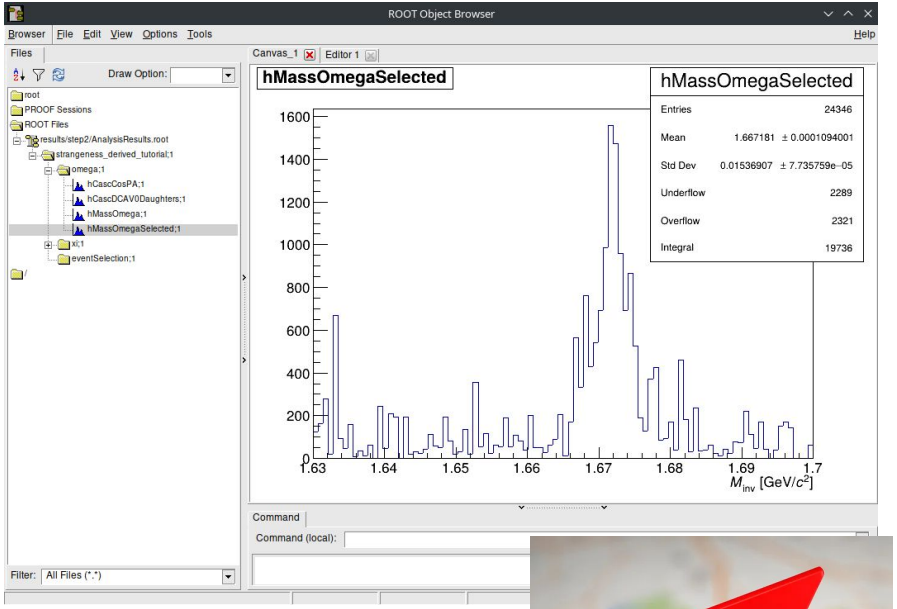
```
// Fill histograms! (if possible)
if (std::abs(bachDaughterTrackCasc.tpcNSigmaPi()) < NSigmaTPCPion) { // Xi case
  rXi.fill(HIST("hMassXiSelected"), casc.mXi());
```

```
if (std::abs(bachDaughterTrackCasc.tpcNSigmaKa()) < NSigmaTPCKaon) {              // Omega case
  if (std::abs(casc.mXi() - o2::constants::physics::MassXiMinus) > cascadesetting_competingmassrej) { // co
    rOmega.fill(HIST("hMassOmegaSelected"), casc.mOmega());
```


Romain Schotter (romain.schotter@cern.ch)

Step 2: apply TPC PID selections





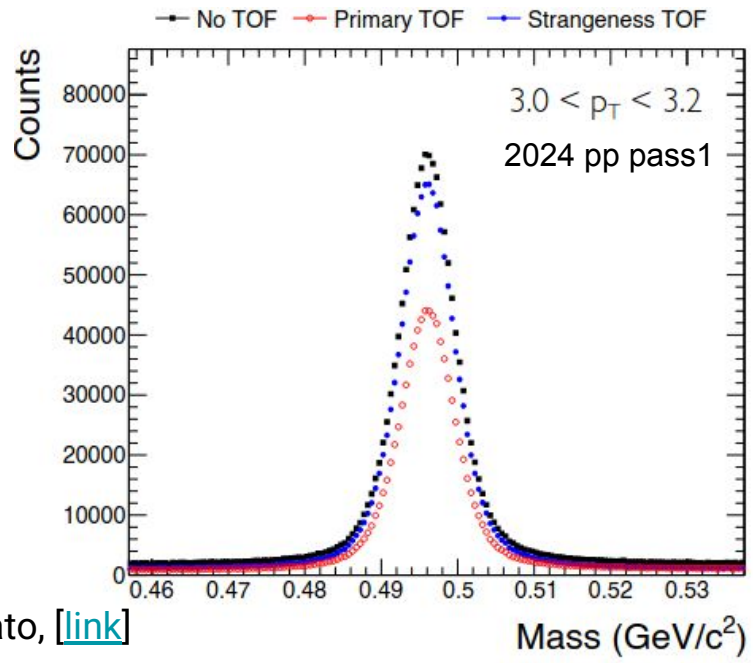
- Background is further reduced using TPC PID (**n = 3σ**) selections

Step 3: apply TOF PID selections

- We would like now to apply TOF PID selections on each decay daughters, but only if they do have signal in the TOF

- Use strangeness-based TOF PID: strangenesstofpid.cxx, **optimized for secondary particles**
  - **Collision association** may be incorrect, biasing the event time (t_0)
    → recalculated with V0/cascade collision t_0
  - **Travel time before decay** not accounted for
    → correct for travel before decay

- In high interaction rate environments,
  → primary TOF leads large to signal loss
  → while strangeness-based TOF preserves more than 95% of signal



D. Chinellato, [link]

Step 3: apply TOF PID selections

- We would like now to apply TOF PID selections on each decay daughters, but only if they do have signal in the TOF

```
// Configurable parameters for TOF PID selection
Configurable<float> NSigmaTOFPion{"NSigmaTOFPion", 3, "NSigmaTOFPion"};
Configurable<float> NSigmaTOFKaon{"NSigmaTOFKaon", 3, "NSigmaTOFKaon"};
Configurable<float> NSigmaTOFProton{"NSigmaTOFProton", 3, "NSigmaTOFProton"};
```

Add **Configurable** for each TOF PID selection

```
void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::CascCores, aod::CascExtras, aod::CascTOFNSigmas, aod::CascTOFPIDs>> const& Cascades,
             dauTracks const&)
```

Subscribe to **aod::CascTOFPIDs** and **aod::CascTOFNSigmas** table

- ***Instructions:***
  - Apply TOF PID selections **(n = 3σ)** on each decay daughters, **if** it has a matched hit in the TOF
  - For each mass hypothesis, fill a histogram with the invariant mass after cascade+PID selections

- ***Hint:*** Check the getters .[bachelor,positive,negative]HasTOF() in `LFStrangenessPIDTables.h`

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 3: apply TOF PID selections

```
// TOF PID check
bool xiPassTOFSelection = true;
bool omegaPassTOFSelection = true;
if (casc.sign() < 0) {
  if (casc.positiveHasTOF()) {
    if (std::abs(casc.tofNSigmaXiLaPr()) > NSigmaTOFProton) {
      xiPassTOFSelection &= false;
    }
    if (std::abs(casc.tofNSigmaOmLaPr()) > NSigmaTOFProton) {
      omegaPassTOFSelection &= false;
    }
  }
  if (casc.negativeHasTOF()) {
    if (std::abs(casc.tofNSigmaXiLaPi()) > NSigmaTOFPion) {
      xiPassTOFSelection &= false;
    }
    if (std::abs(casc.tofNSigmaOmLaPi()) > NSigmaTOFPion) {
      omegaPassTOFSelection &= false;
    }
  }
} else {
  if (casc.positiveHasTOF()) {
    if (std::abs(casc.tofNSigmaXiLaPi()) > NSigmaTOFPion) {
      xiPassTOFSelection &= false;
    }
    if (std::abs(casc.tofNSigmaOmLaPi()) > NSigmaTOFPion) {
      omegaPassTOFSelection &= false;
    }
  }
  if (casc.negativeHasTOF()) {
    if (std::abs(casc.tofNSigmaXiLaPr()) > NSigmaTOFProton) {
      xiPassTOFSelection &= false;
    }
    if (std::abs(casc.tofNSigmaOmLaPr()) > NSigmaTOFProton) {
      omegaPassTOFSelection &= false;
    }
  }
}

if (casc.bachelorHasTOF()) {
  if (std::abs(casc.tofNSigmaXiPi()) > NSigmaTOFPion) {
    xiPassTOFSelection &= false;
  }
  if (std::abs(casc.tofNSigmaOmKa()) > NSigmaTOFKaon) {
    omegaPassTOFSelection &= false;
  }
}
```

Apply TOF PID selections **if** the track has a hit in TOF

```
xiPassTOFSelection =  casc.tofXiCompatibility (NSigmaTOF);
omegaPassTOFSelection = casc. tofOmegaCompatibility (NSigmaTOF);
```

Fill invariant mass histograms <u>after</u> selections

```
// Fill histograms! (if possible)
if (std::abs(bachDaughterTrackCasc.tpcNSigmaPi()) < NSigmaTPCPion) { // Xi case
  rXi.fill(HIST("hMassXiSelected"), casc.mXi());
  if (xiPassTOFSelection)
    rXi.fill(HIST("hMassXiSelectedWithTOF"), casc.mXi());
```

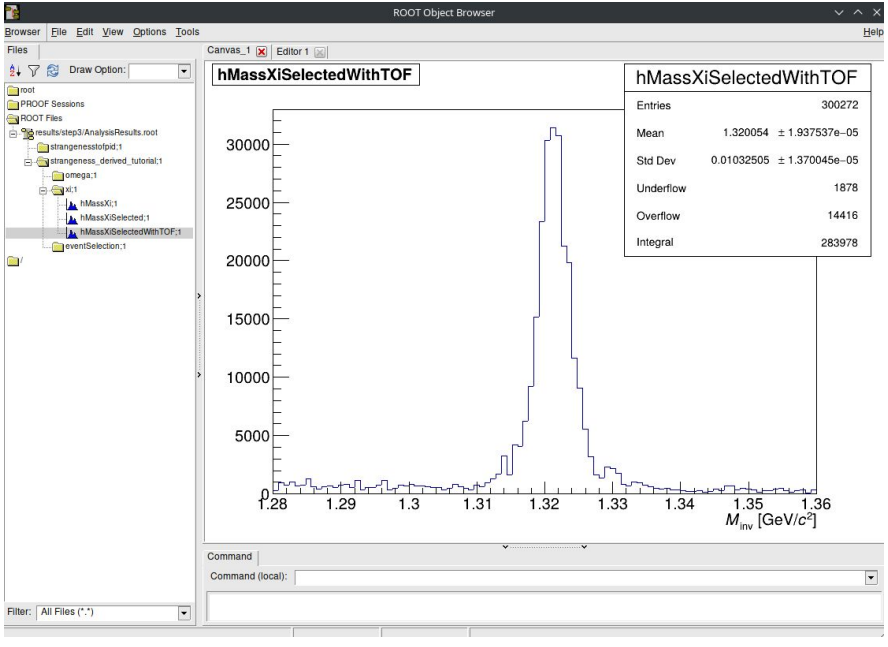
```
if (std::abs(bachDaughterTrackCasc.tpcNSigmaKa()) < NSigmaTPCKaon) {              // Omega case
  if (std::abs(casc.mXi() - o2::constants::physics::MassXiMinus) > cascadesetting_competingmassrej)
    rOmega.fill(HIST("hMassOmegaSelected"), casc.mOmega());
    if (omegaPassTOFSelection) {
      rOmega.fill(HIST("hMassOmegaSelectedWithTOF"), casc.mOmega());
    }
```

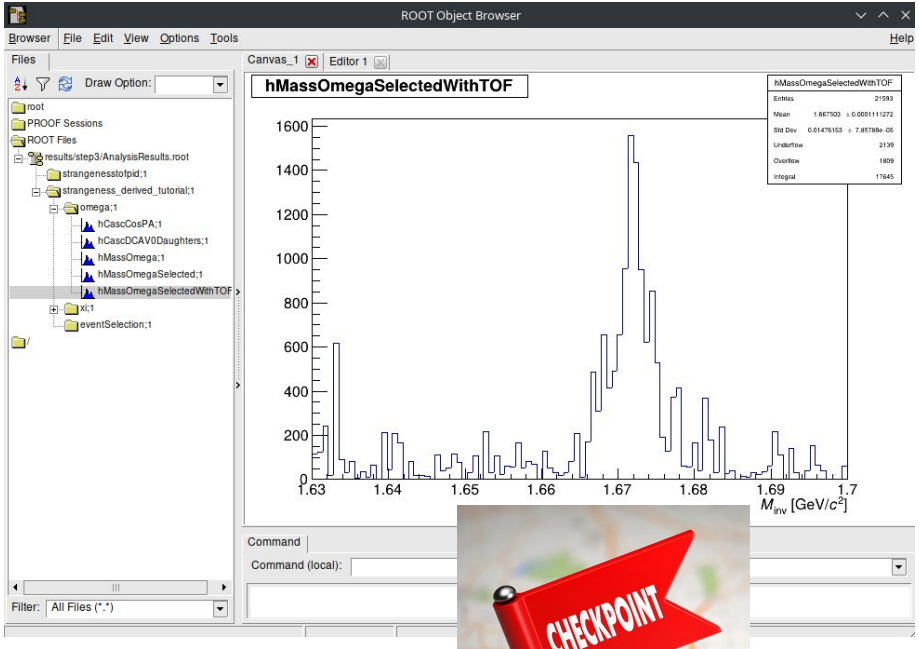

Romain Schotter (romain.schotter@cern.ch)

Step 3: apply TOF PID selections

```
OPTION="-b --configuration json://configuration_step3.json"

o2-analysis-lf-strangenesstofpid ${OPTION} | o2-analysistutorial-lf-strangeness-derived-step3 ${OPTION} --
```

Extra task needed
`strangenesstofpid`





- TOF (**n = 3σ**) selections help further suppressing combinatorial background

Step 4: access MC information of cascades

- From the previous steps, one can reconstruct the Xi and Omega with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true cascade as a function of transverse momentum

aod::CascCores

→ **reconstructed** info about cascades

Romain Schotter (romain.schotter@cern.ch)

Step 4: access MC information of cascades

- From the previous steps, one can reconstruct the Xi and Omega with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true cascade as a function of transverse momentum

aod::CascCores

→ **reconstructed** info about cascades

aod::CascMCCores

→ **generated** info about cascades

Romain Schotter (romain.schotter@cern.ch)

Step 4: access MC information of cascades

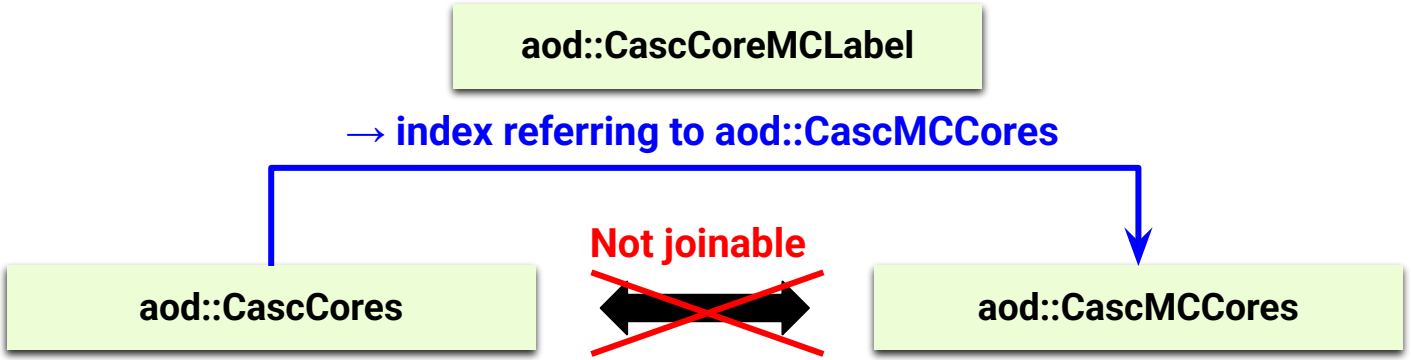
- From the previous steps, one can reconstruct the Xi and Omega with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true cascade as a function of transverse momentum

→ **reconstructed** info about cascades

→ **generated** info about cascades

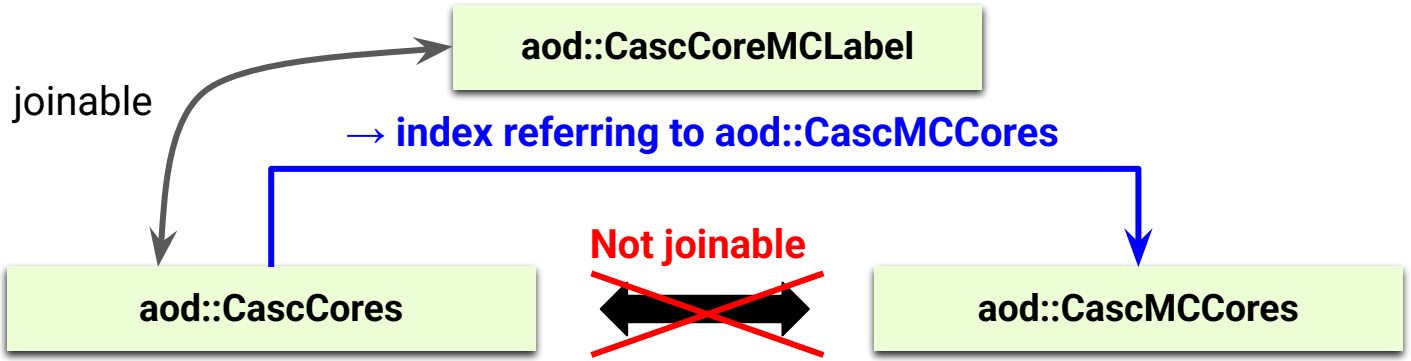
Step 4: access MC information of cascades

- From the previous steps, one can reconstruct the Xi and Omega with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true cascade as a function of transverse momentum



Step 4: access MC information of cascades

- From the previous steps, one can reconstruct the Xi and Omega with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true cascade as a function of transverse momentum



→ **reconstructed** info about cascades

→ **generated** info about cascades

Step 4: access MC information of cascades

```
void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::CascCores, aod::CascExtras, aod::CascTOFNSigmas, aod::CascTOFPIDs, aod::CascCoreMCLabels>> const& Cascades,
             dauTracks const&,
             aod::CascMCCores const& /*cascmccores*/)
```

Subscribe to **aod::CascCoreMCLabels** and **aod::CascMCCores** tables

```
// MC truth info
if(!casc.has_cascMCCore()) {
  continue;
}
auto cascmccore = casc.cascMCCore_as<aod::CascMCCores>();
```

Check that cascades come from a gen. particles via **has_cascMCCore()** and recover the gen. info

- ***Instructions:***
  - Download the derived MC data from cernbox
  - For each mass hypothesis, fill a histogram with the transverse momentum of the true reconstructed Ξ or Ω

- ***Hints:***
  1. Have a look at the getter .pdgCode() in `LFStrangenessTable.h`
  2. Use the PDG code from ROOT: **PDG_t::kXiMinus** for Ξ and **PDG_t::kOmegaMinus** for Ω
  3. For the binning in pT, use the following binning: 100 bins, from 0 to 10 GeV/*c*

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 4: access MC information of cascades

```
void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::CascCores, aod::CascExtras, aod::CascTOFNSigmas, aod::CascCoreMCLabels>> const& Cascades,
             dauTracks const&,
             aod::CascMCCores const& /*cascmccores*/)
```

```
// MC truth info
if (!casc.has_cascMCCore()) {
  continue;
}
auto cascmccore = casc.cascMCCore_as<aod::CascMCCores>();

// Checking that the cascade is a true Xi
if (std::abs(cascmccore.pdgCode()) == PDG_t::kXiMinus) {
  if (std::abs(bachDaughterTrackCasc.tpcNSigmaPi()) < NSigmaTPCPion) { // Xi case
    rXi.fill(HIST("hMassXiTrueRec"), casc.mXi());
    rXi.fill(HIST("hPtXiTrueRec"), casc.pt());
    if (xiPassTOFSelection) {
      rXi.fill(HIST("hMassXiTrueRecWithTOF"), casc.mXi());
      rXi.fill(HIST("hPtXiTrueRecWithTOF"), casc.pt());
    }
  }
}
if (std::abs(cascmccore.pdgCode()) == PDG_t::kOmegaMinus) {
  if (std::abs(bachDaughterTrackCasc.tpcNSigmaKa()) < NSigmaTPCKaon) {              // Omega case
    if (std::abs(casc.mXi() - o2::constants::physics::MassXiMinus) > cascadesetting_competingmassrej) {
      rOmega.fill(HIST("hMassOmegaTrueRec"), casc.mOmega());
      rOmega.fill(HIST("hPtOmegaTrueRec"), casc.pt());
      if (omegaPassTOFSelection) {
        rOmega.fill(HIST("hMassOmegaTrueRecWithTOF"), casc.mOmega());
        rOmega.fill(HIST("hPtOmegaTrueRecWithTOF"), casc.pt());
      }
    }
  }
}
```

Subscribe to **aod::CascCoreMCLabels** and **aod::CascMCCores** tables

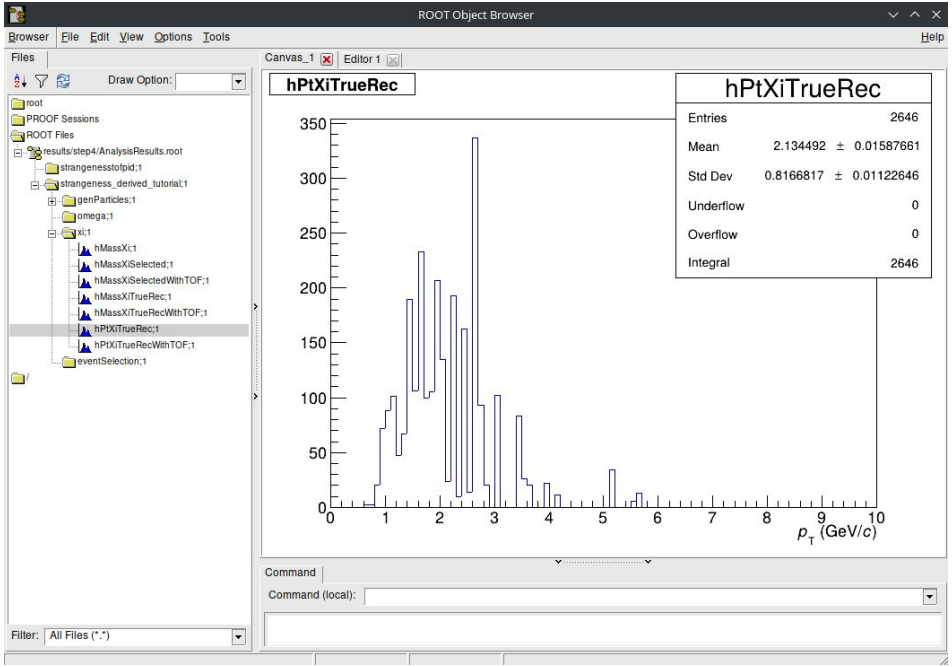
Check that cascades come from a gen. particles via **has_cascMCCore()** and recover the gen. info

+

De-reference the corresponding entry in the **aod::CascMCCores** table

Check PDG code of gen. cascades via **pdgCode()**

Step 4: access MC information of cascades



- The number of reconstructed Ξ has been extracted as a function of the transverse momentum
- Not enough statistics for the Ω (note this is done only with 1 derived data file)

Step 4: loop over generated events

- We have the number of reconstructed true cascades as a function of transverse momentum
- Now, we need the **number of generated cascades as a function of transverse momentum**
  → requires a new process function that iterates over MC collisions

```
void processRecMC(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision,
                  soa::Filtered<soa::Join<aod::CascCores, aod::CascExtras, aod::CascTOFNSigmas, aod::CascTOFPIDs, aod::CascCoreMCLabels>> const& Cascades,
                  dauTracks const&,
                  aod::CascMCCores const& /*cascmccores*/)
```

Rename the previous **process()** function in **processRecMC()**

```
void processGenMC(soa::Filtered<aod::StraMCCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<aod::StraCollisions, o2::aod::StraEvSels, aod::StraCollLabels>>& collisions,
                  const soa::SmallGroups<soa::Join<aod::CascMCCores, aod::CascMCCollRefs>>& cascMC)
```

Create an additional function **processGenMC()**

Use **soa::SmallGroups** to select reconstructed collisions and gen. cascades associated to this MC collision

```
PROCESS_SWITCH(strangeness_derived_tutorial, processRecMC, "Process Run 3 mc, reconstructed", true);
PROCESS_SWITCH(strangeness_derived_tutorial, processGenMC, "Process Run 3 mc, generated", true);
```

Add **PROCESS_SWITCH** to allow for 2 process functions

Step 4: loop over generated events

- The next step is to:
  - check that the MC collision have been reconstructed at least once using the **.size()**
  - Loop over all generated cascades
  - Fill a histogram with the generated transverse momentum for each mass hypothesis

```
void processGenMC(soa::Filtered<aod::StraMCCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<aod::StraCollisions, o2::aod::StraEvSels, aod::StraCollLabels>>& collisions,
                  const soa::SmallGroups<soa::Join<aod::CascMCCores, aod::CascMCCollRefs>>& cascMC)
```


Romain Schotter (romain.schotter@cern.ch)

Step 4: loop over generated events

- The next step is to:
  - check that the MC collision have been reconstructed at least once using the **.size()**
  - Loop over all generated cascades
  - Fill a histogram with the generated transverse momentum for each mass hypothesis

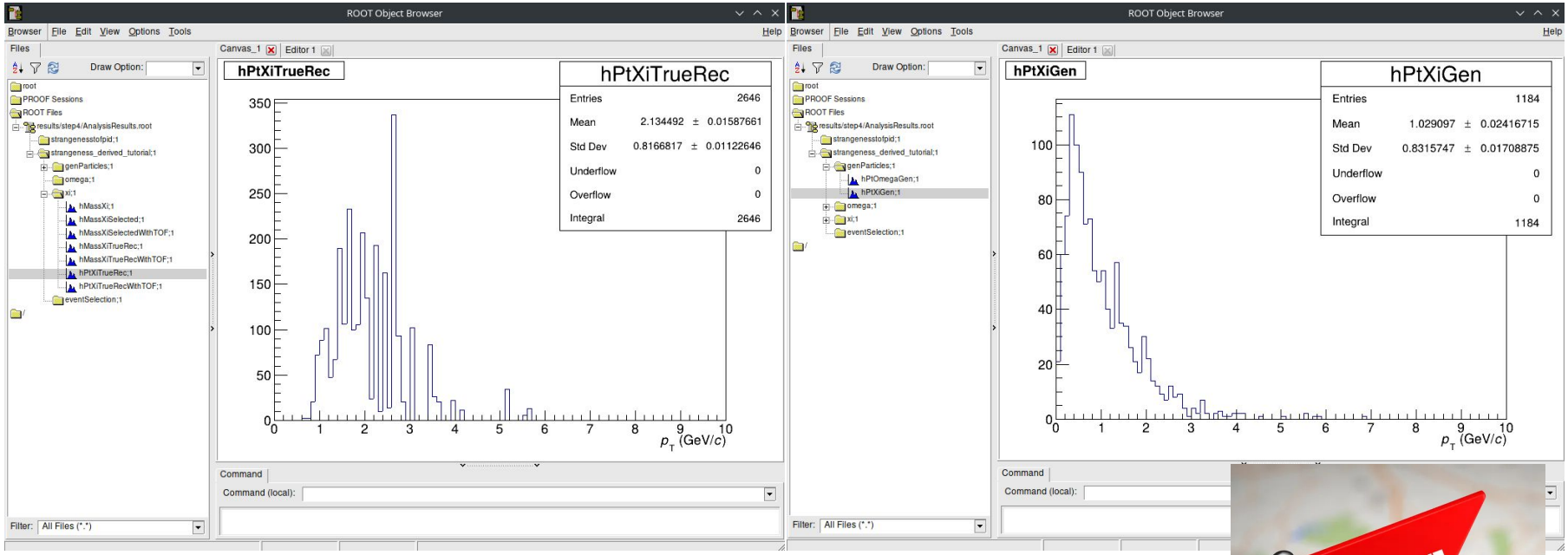
```
void processGenMC(soa::Filtered<aod::StraMCCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<aod::StraCollisions, o2::aod::StraEvSels, aod::StraCollLabels>>& collisions,
                  const soa::SmallGroups<soa::Join<aod::CascMCCores, aod::CascMCCollRefs>>& cascMC)
{
  if (collisions.size() < 1) // to process generated collisions that've been reconstructed at least once
    return;
  rEventSelection.fill(HIST("hVertexZGen"), mcCollision.posZ());

  for (const auto& cascmc : cascMC) {
    if (std::abs(cascmc.pdgCode()) == PDG_t::kXiMinus) {
      rGenParticles.fill(HIST("hPtXiGen"), cascmc.ptMC());
    }
    if (std::abs(cascmc.pdgCode()) == PDG_t::kOmegaMinus) {
      rGenParticles.fill(HIST("hPtOmegaGen"), cascmc.ptMC());
    }
  }
}
```

Loop over all generated cascades

Fill a histogram with the gen. pt

Romain Schotter (romain.schotter@cern.ch)

Step 4: loop over generated events



- The number of reconstructed and generated Ξ have been extracted as a function of the transverse momentum
- One step away from having the efficiency corrections…

Summary

In this hands-on session you learned:

- how to **subscribe** to the derived tables in your analysis
- **apply simple** topological and kinematic **selections**
- **apply TPC and TOF PID** selections
- perform an **analysis on MC**

This was just a taste of an analysis on strange hadrons!

Summary

There are many other analysis!

Feel free to contact analysers and have a peek at their task!

Decay topology / Particle	System@Energy	Main observables	People	Analysis Note
V0s	pp@900 GeV	spectra vs. multiplicity	Francesca Ercolessi, Nicolò Jacazio	
Cascades	pp@900 GeV	spectra vs. multiplicity	Chiara De Martin, Francesca Ercolessi, Roman Nepeivoda, Upasana Sharma	AN-1446
V0s	pp@13.6 TeV	spectra vs. multiplicity	Francesca Ercolessi, Nicolò Jacazio	
Cascades	pp@13.6 TeV	spectra vs. multiplicity	Chiara De Martin, Francesca Ercolessi, Roman Nepeivoda, Upasana Sharma	AN-1446
V0s and Cascades	pp@13.6 TeV	h-V0, h-Casc correlations	Kai Cui, David Chinellato, Lucia Tarasovicová, Zhong-Bao Yin, Chiara De Martin	AN-1550
Cascades	pp@13.6 TeV	correlations	Rik Spijkers	
Hyperons	pp@13.6 TeV	hyperon absorption in the beam pipe and detector material	Alberto Caliva, Francesco Mazzaschi, Maximiliano Puccio	
Hyperons	pp@13.6 TeV	elastic scattering of hyperons with nuclei in ITS	Alberto Caliva	
Cascades	pp@13.6 TeV, Pb-Pb@5.36 TeV	prompt and non-prompt cascades	Andrea Sofia Triolo, Maximiliano Puccio	
Sigma	pp@13.6 TeV	kink analysis, spectra	Paraskevi Ganoti	
Sigma	pp@13.6 TeV	Sigma -> Λ + gamma	Gianni Shigeru Setoue Liveraro, Jun Takahashi, David Dobrigkeit Chinellato	
V0s and Cascades	pp@13.6 TeV	production in jets with jet finder	Alberto Calivà, Francesca Ercolessi, Chiara De Martin	
V0s	pp@13.6 TeV	Jet fragmentation into V0	Gijs Van Weelden	
V0s	pp@13.6 TeV	yields vs charged particle flattenicity	Suraj Prasad	
V0s	Pb-Pb@5.36 TeV	spectra vs. centrality	Romain Schotter, David Dobrigkeit Chinellato	AN-1540
Cascades	Pb-Pb@5.36 TeV	spectra vs. centrality	Lucia Tarasovicová	AN-1512
Cascades	Pb-Pb@5.36 TeV	v2 (v3) and polarization	Sourav Kundu, Maximiliano Puccio, Chiara De Martin	AN-1521
Hyperons	Pb-Pb@5.36 TeV	Hyperon polarization	Junlee Kim	AN-1561
Cascades	pp@13.6 TeV	K0s - phi correlation	Stefano Cannito, Valentina Zaccolo	AN-1563

Thank you very much!

Romain Schotter (romain.schotter@cern.ch)

Additional materials

Analysis Level Tables: aod::V0Datas

#include "PWGLF/DataModel/LFStrangenessTables.h" V0Datas = soa::Join<**V0Indices**, V0TrackXs, **V0Cores**>

	Name	Getter
Bound	v0::PosTrackId	posTrackId
	v0::NegTrackId	negTrackId
	v0::CollisionId	collisionId
	v0::V0	v0Id
Momenta of daughter tracks	v0data::PxPos	pxpos
	v0data::PyPos	pypos
	v0data::PzPos	pzpos
	v0data::PxNeg	pxneg
	v0data::PyNeg	pyneg
	v0data::PzNeg	pxneg
Calculated Decay vertex position	v0data::X	x
	v0data::Y	y
	v0data::Z	z
Topological variables	v0data::DCAV0Daughters	dcav0daughters
	v0data::DCAPosToPV	dcapostopv
	v0data::DCANegToPV	dcanegtopv

Derived (dynamic)	
v0data::Pt	pt
v0data::V0Radius	v0radius
v0data::V0CosPA	v0cospa
v0data::DCAV0ToPV	dcav0topv
v0data::MLambda	mLambda
v0data::MAntiLambda	mAntiLambda
v0data::MK0Short	mK0Short
v0data::YK0Short	yK0Short
v0data::YLambda	yLambda
v0data::Eta	eta
v0data::Phi	phi
...	...
v0data::Px	px
v0data::Py	py
v0data::Pz	pz

Analysis Level Tables: aod::CascDatas

#include "PWGLF/DataModel/LFStrangenessTables.h" CascDatas = soa::Join<**CascIndices**, CascBBs, **CascCores**>

	Name	Getter
Bound	cascade::V0Id	v0Id
	cascade::BachelorId	bachelorId
	cascade::CollisionId	collisionId
	cascdata::Sign	sign
Momenta of daughter tracks	cascdata::PxPos(Neg)	pxpos(neg)
	cascdata::PyPos(Neg)	pypos(neg)
	cascdata::PzPos(Neg)	pzpos(neg)
	cascdata::PxBach	pxbach
	cascdata::PyBach	pybach
	cascdata::PzBach	pzbach
Calculated	cascdata::X	x
Decay vertex position	cascdata::Y	y
	cascdata::Z	z
Topological variables	cascdata::DCAV0Daughters	dcav0daughters
	cascdata::DCACascDaughters	dcascascdaughters
	cascdata::DCAPosToPV	dcapostopv
	cascdata::DCANegToPV	dcanegtopv
	cascdata::DCABachToPV	dcabachtopv
	cascdata::Pt	pt
	...	...
Derived (dynamic)	cascdata::V0Radius	v0radius
	cascdata::CascRadius	cascradius
	cascdata::V0CosPA	v0cosPA
	cascdata::CascCosPA	casccosPA
	cascdata::MLambda	mLambda
	cascdata::MXi	mXi
	cascdata::MOmega	mOmega
	cascdata::YXi	yXi
	...	...
	cascdata::Pt	pt
	cascdata::P	p
	cascdata::PxLambda	pxlambda
	...	...

Preparation

- All the necessary scripts and workflows for this tutorial can be found in the O2Physics/Tutorials/PWGLF/Strangeness/Original/
- The json files (and more!) can be found here: https://cernbox.cern.ch/s/O3kXLRflp7sosxh

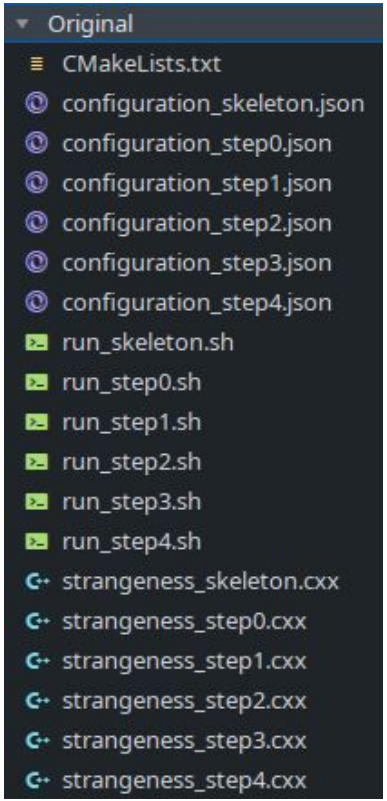
- Create a folder to store the input data
- Download the file from cernbox (~7.6 GB)
- Alternatively, download the input files from lxplus

```
rsync -r -u --progress USERNAME@lxplus.cern.ch:/afs/cern.ch/user/r/rschotte/cernbox/O2AT2025/Derived/OriginalAO2Ds .
```

Enter your lxplus password

Preparation: running an analysis task

- Once the original data have been downloaded, we can start writing the analysis task
- The idea is to start with the `strangeness_skeleton.cxx` file and improve it up to step4
- Checkpoints for each step (X) are here for your convenience (`strangeness_stepX.cxx`), with the running shell script (`run_stepX.sh`) and the corresponding configurations (`configuration_stepX.json`)
- Let's have a look at `strangeness_skeleton.cxx`



Preparation: running an analysis task

`strangeness_skeleton.cxx` loops over all events, selects the ones with |z| < 10 cm and `sel8` flag (FT0 vertex compatible with FT0-A and FT0-C timing info), fills an histogram with the *z*-vertex position

```cpp
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"
#include "Common/DataModel/EventSelection.h"
#include "PWGLF/DataModel/LFStrangenessTables.h"

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

struct strangeness_tutorial {
  // Histograms are defined with HistogramRegistry
  HistogramRegistry rEventSelection{"eventSelection", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};

  // Configurable for histograms
  Configurable<int> nBins{"nBins", 100, "N bins in all histos"};

  // Configurable for event selection
  Configurable<float> cutzvertex{"cutzvertex", 10.0f, "Accepted z-vertex range (cm)"};

  void init(InitContext const&)
  {
    // Axes
    AxisSpec vertexZAxis = {nBins, -15., 15., "vrtx_{Z} [cm]"};

    // Histograms
    // Event selection
    rEventSelection.add("hVertexZRec", "hVertexZRec", {HistType::kTH1F, {vertexZAxis}});
  }
```

Include required **headers**

Histogram registries and **Configurables** are declared before the **init** function

Create axis and add needed **histograms** to the registries

Romain Schotter (romain.schotter@cern.ch)

Preparation: running an analysis task

```
45    // Defining filters for events (event selection)
46    // Processed events will be already fulfilling the event selection requirements
47    Filter eventFilter = (o2::aod::evsel::sel8 == true);
48    Filter posZFilter = (nabs(o2::aod::collision::posZ) < cutzvertex);
49
50    void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision)
51    {
52      // Fill the event counter
53      rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());
54    }
55  };
56
57  WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
58  {
59    return WorkflowSpec{
60      adaptAnalysisTask<strangeness_tutorial>(cfgc)};
61  }
```

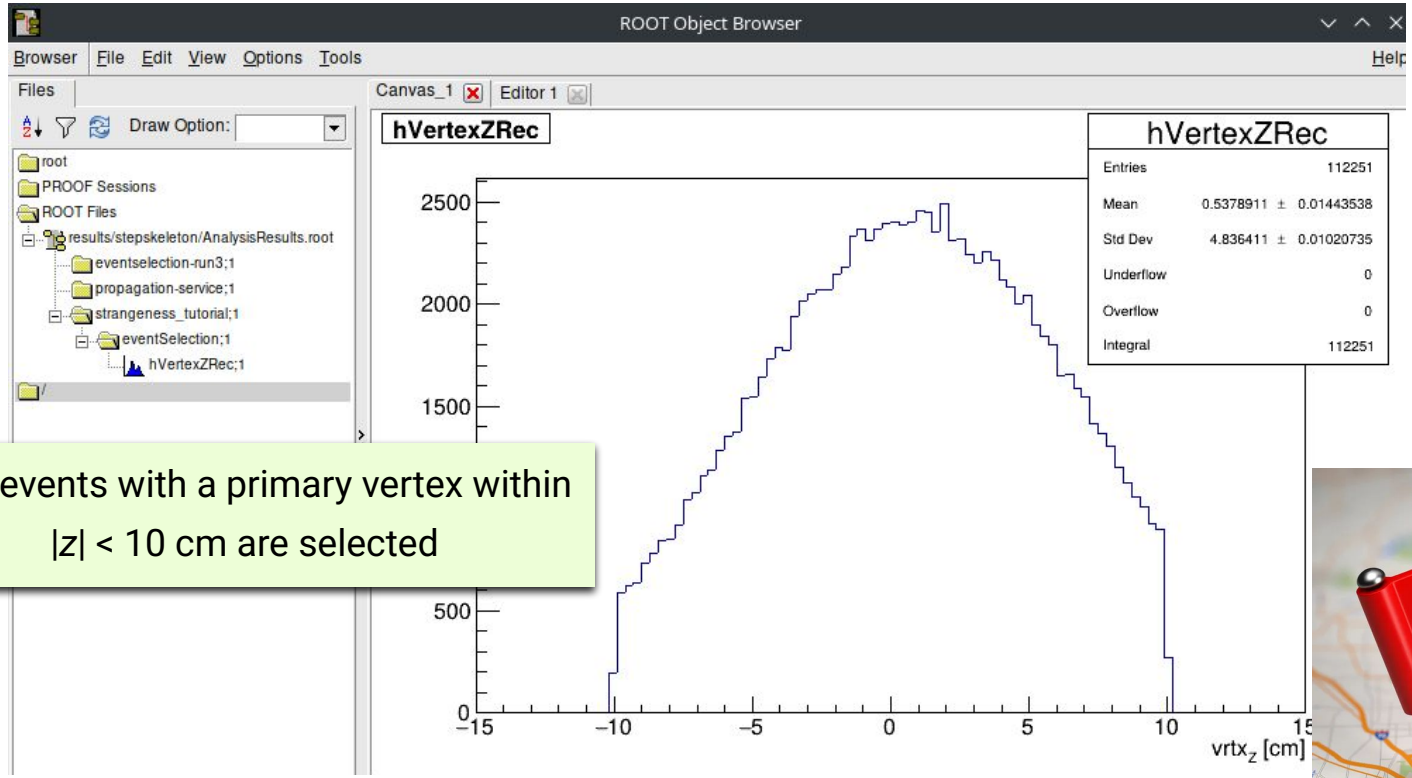
Define **Filters** for basic event selection

Subscribe to an iterator of a table joining **aod::StraCollisions** (=basic collision properties) **and aod::StraEvSels** (= all necessary event selection variables) to loop over the selected collisions

Because of the iterator, the **process()** function is called once for each selected event and fills the vertex *z*-position histogram

Romain Schotter (romain.schotter@cern.ch)

Preparation: running an analysis task



Only events with a primary vertex within $|z| < 10$ cm are selected

Step 0: looping over V0s

- Starting from `strangeness_skeleton.cxx`, we want to loop over all V0s in the selected events and fill a invariant mass histogram

```
struct strangeness_tutorial {
  // Histograms are defined with HistogramRegistry
  HistogramRegistry rEventSelection{"eventSelection", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
  HistogramRegistry rKzeroShort{"kzeroShort", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
```

Create a “**kzeroShort**” **HistogramRegistry**

```
  void init(InitContext const&)
  {
    // Axes
    AxisSpec K0ShortMassAxis = {200, 0.45f, 0.55f, "#it{M}_{inv} [GeV/#it{c}^{2}]"};
    AxisSpec vertexZAxis = {nBins, -15., 15., "vrtx_{Z} [cm]"};

    // Histograms
    // Event selection
    rEventSelection.add("hVertexZRec", "hVertexZRec", {HistType::kTH1F, {vertexZAxis}});

    // K0s reconstruction
    rKzeroShort.add("hMassK0Short", "hMassK0Short", {HistType::kTH1F, {K0ShortMassAxis}});
  }
```

Create dedicated axis for K^0_S

Create an invariant mass histogram for K^0_S and store it in its corresponding **HistogramRegistry**

Step 0: looping over V0s

- Starting from `strangeness_skeleton.cxx`, we want to loop over all V0s in the selected events and fill a invariant mass histogram

```
void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             aod::V0Datas const& V0s)
{
  // Fill the event counter
  rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());
```

Subscribe to the V0 tables (**aod::V0Datas**)

- ***Instructions:***
  - Subscribe to the V0 tables (have a look at `LFStrangenessTables.h`)
  - For each event, loop over all V0s,
  - Fill a histogram with the invariant mass of K^0_S
  - Store the histogram in the corresponding **HistogramRegistry**

- ***Hints:*** for the V0 info, look at the **aod::V0Datas** table and the getters .mK0Short()

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 0: looping over V0s

- Starting from `strangeness_skeleton.cxx`, we want to loop over all V0s in the selected events and fill a invariant mass histogram

```
void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             aod::V0Datas const& V0s)
{
  // Fill the event counter
  rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());

  for (const auto& v0 : V0s) {
    rKzeroShort.fill(HIST("hMassK0Short"), v0.mK0Short());
  }
}
```

For each event, loop over all V0s

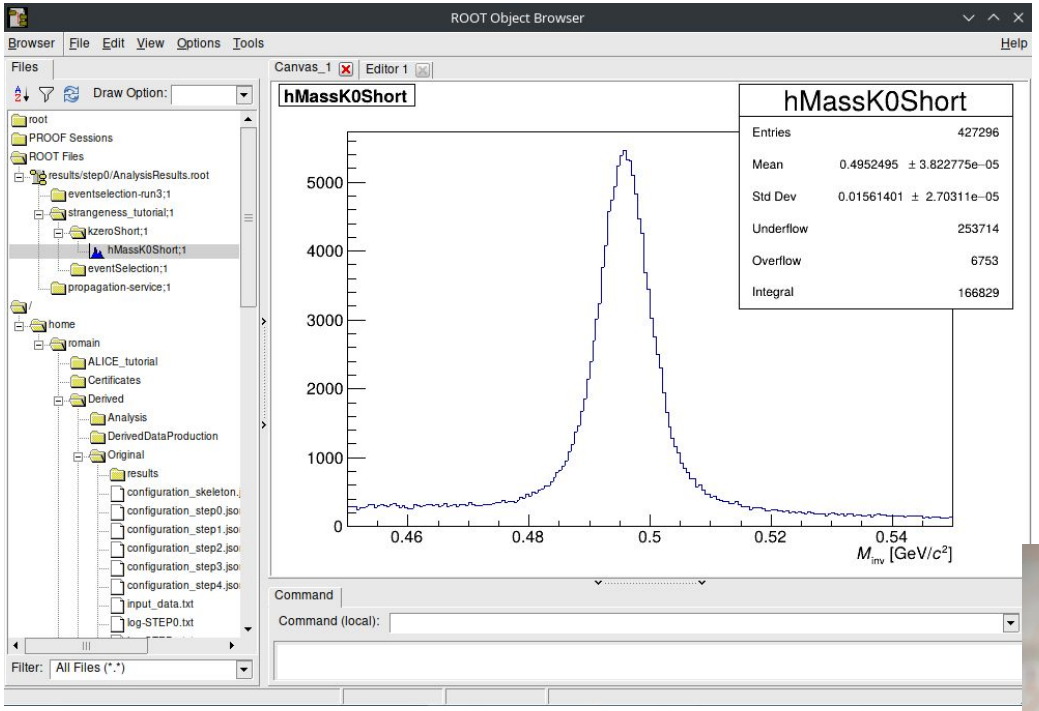
Fill a histogram with the invariant mass

- ***Instructions:***
  - Subscribe to the V0 tables (have a look at `LFStrangenessTables.h`)
  - For each event, loop over all V0s,
  - Fill a histogram with the invariant mass of K^0_S
  - Store the histogram in the corresponding **HistogramRegistry**

- ***Hints:*** for the V0 info, look at the **aod::V0Datas** table and the getters .mK0Short()

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 0: looping over V0s

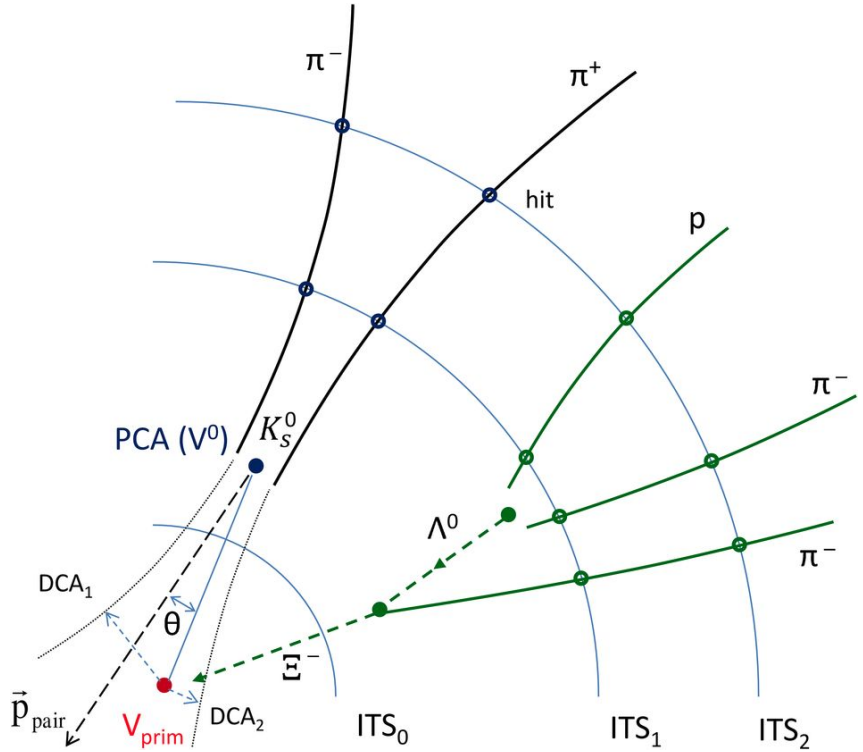


Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

$K^0_S \rightarrow \pi^+\pi^-$ cτ = 2.6844 cm B.R. 69.2%

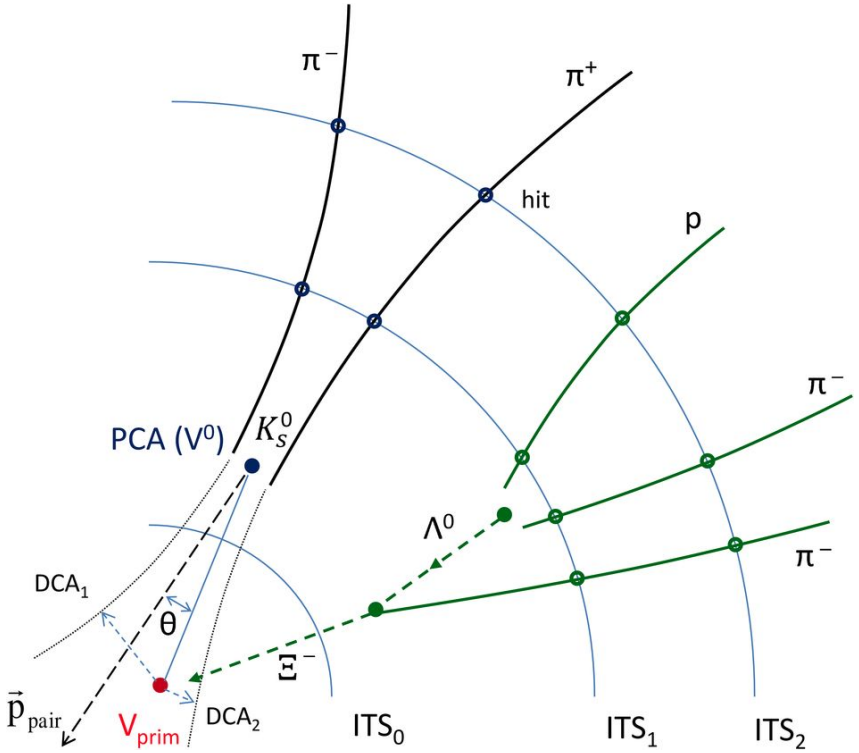
$\begin{cases} \Lambda \rightarrow p\pi^- \\ \overline{\Lambda} \rightarrow \overline{p}\pi^+ \end{cases}$ cτ = 7.89 cm B.R. 63.9%



Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

DCA pos./neg. to prim. vtx	> 0.06 cm
DCA between V0. daughters	< 1.0 cm
V0 decay radius	> 1.2 cm
V0 cos of pointing angle	> 0.998



Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

		Getter	Column type
DCA pos./neg. to prim. vtx	> 0.06 cm	dcapos/negtopv ()	standard
DCA between V0. daughters	< 1.0 cm	dcaV0daughters ()	standard
V0 decay radius	> 1.2 cm	v0radius ()	dynamic
V0 cos of pointing angle	> 0.998	v0cospa (*arg1, arg2, arg3*)	dynamic

Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

```
// Configurable parameters for V0 selection
Configurable<float> v0setting_dcav0dau{"v0setting_dcav0dau", 1, "DCA V0 Daughters"};
Configurable<float> v0setting_dcapostopv{"v0setting_dcapostopv", 0.06, "DCA Pos To PV"};
Configurable<float> v0setting_dcanegtopv{"v0setting_dcanegtopv", 0.06, "DCA Neg To PV"};
Configurable<double> v0setting_cospa{"v0setting_cospa", 0.98, "V0 CosPA"}; // double -> N.B. dcos(x)/dx = 0 at
Configurable<float> v0setting_radius{"v0setting_radius", 0.5, "v0radius"};
```

Add **Configurable** for each V0 selection

Romain Schotter (romain.schotter@cern.ch)

Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

```
// Configurable parameters for V0 selection
Configurable<float> v0setting_dcav0dau{"v0setting_dcav0dau", 1, "DCA V0 Daughters"};
Configurable<float> v0setting_dcapostopv{"v0setting_dcapostopv", 0.06, "DCA Pos To PV"};
Configurable<float> v0setting_dcanegtopv{"v0setting_dcanegtopv", 0.06, "DCA Neg To PV"};
Configurable<double> v0setting_cospa{"v0setting_cospa", 0.98, "V0 CosPA"}; // double -> N.B. dcos(x)/dx = 0 at
Configurable<float> v0setting_radius{"v0setting_radius", 0.5, "v0radius"};
```

Add **Configurable** for each V0 selection

- ***Instructions:***
  - Select the candidates passing the topological selections
  - Fill a histogram with the invariant mass after V0 selections

- ***Hints:***
  1. Have a look at the **aod::V0Datas** table in `LFStrangenessTables.h` to find the columns storing the selection variables
  2. you can use **Filter** to select the candidates but it cannot cut on dynamic columns

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 1: apply topological selections

- We would like now to select more finely our candidates based on their decay kinematics and topology

```
// Configurable parameters for V0 selection
Configurable<float> v0setting_dcav0dau{"v0setting_dcav0dau", 1, "DCA V0 Daughters"};
Configurable<float> v0setting_dcapostopv{"v0setting_dcapostopv", 0.06, "DCA Pos To PV"};
Configurable<float> v0setting_dcanegtopv{"v0setting_dcanegtopv", 0.06, "DCA Neg To PV"};
Configurable<double> v0setting_cospa{"v0setting_cospa", 0.98, "V0 CosPA"}; // double -> N.B. dcos(x)/dx = 0 at x
Configurable<float> v0setting_radius{"v0setting_radius", 0.5, "v0radius"};
```

Add **Configurable** for each V0 selection

```
// Filters on V0s
// Cannot filter on dynamic columns, so we cut on DCA to PV and DCA between daughters only
Filter preFilterV0 = (nabs(aod::v0data::dcapostopv) > v0setting_dcapostopv &&
                      nabs(aod::v0data::dcanegtopv) > v0setting_dcanegtopv &&
                      aod::v0data::dcaV0daughters < v0setting_dcav0dau);

void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<aod::V0Datas> const& V0s)
```

Filter on **regular** columns

Step 1: apply topological selections

```cpp
for (const auto& v0 : V0s) {
  rKzeroShort.fill(HIST("hMassK0Short"), v0.mK0Short());

  // Cut on dynamic columns
  if (v0.v0cosPA() < v0setting_cospa)
    continue;
  if (v0.v0radius() < v0setting_radius)
    continue;

  rKzeroShort.fill(HIST("hMassK0ShortSelected"), v0.mK0Short());
```

Cut on **dynamic** columns in the loop

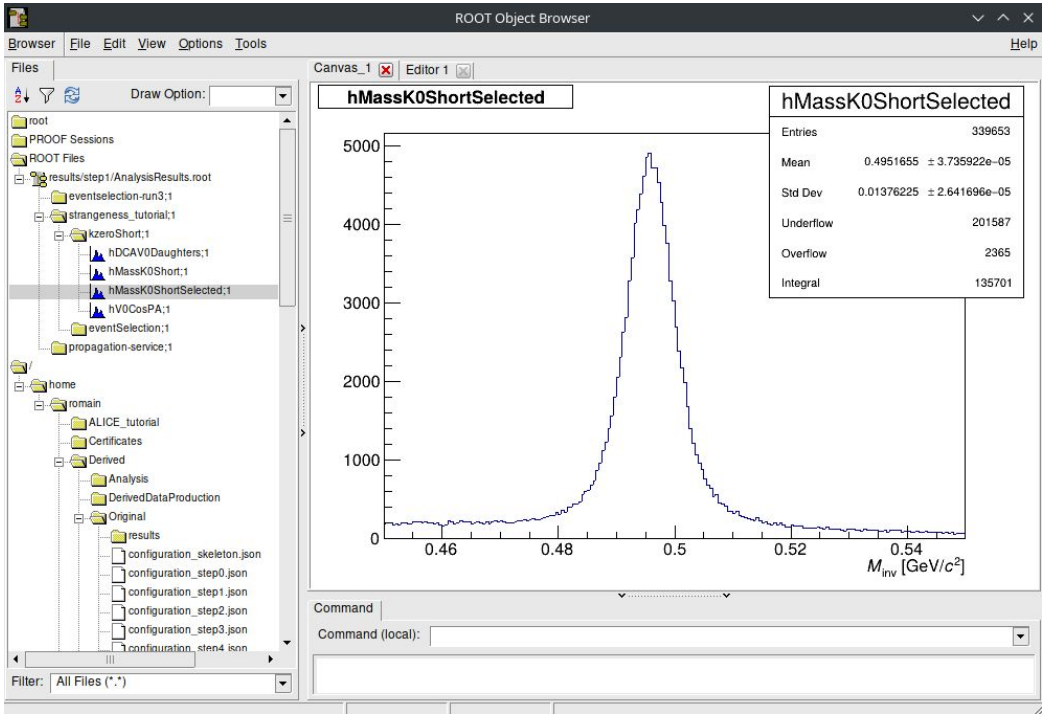
Fill invariant mass histograms <u>after</u> the topological selections

```
"strangeness_tutorial": {
    "nBins": "100",
    "cutzvertex": "10",
    "v0setting_dcav0dau": "1",
    "v0setting_dcapostopv": "0.06",
    "v0setting_dcanegtopv": "0.06",
    "v0setting_cospa": "0.998",
    "v0setting_radius": "1.2"
}
```

Provide the appropriate cuts in the **json configuration file**

The variables must have the same name as their corresponding configurable!

Step 1: apply topological selections



- Background reduces after applying topological selections

Step 2: apply TPC PID selections

- We would like now to apply TPC particle identification (PID) selections to our V0 candidates

Mean energy loss is parametrized by Bethe-Bloch formula:

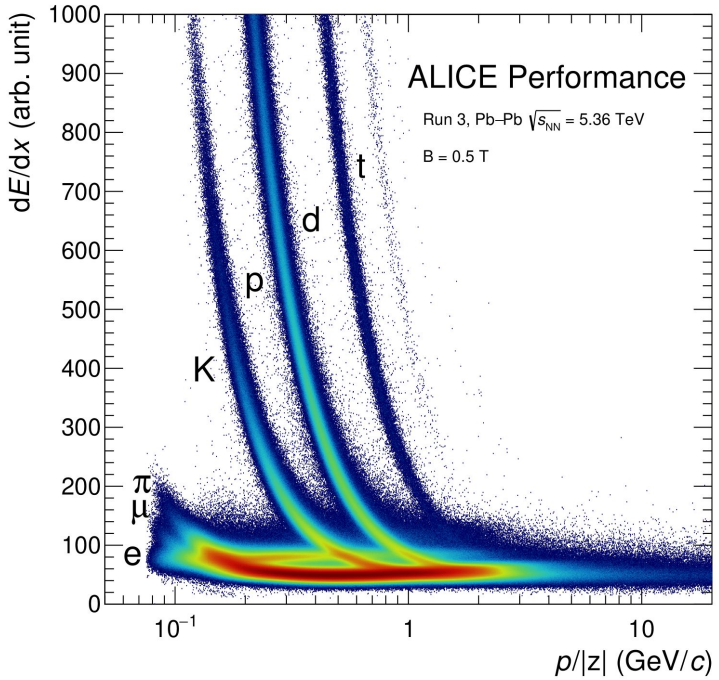
$$\begin{cases} \langle -\frac{dE}{dx} \rangle = K z^2 \frac{Z}{A} \frac{1}{\beta^2} \left[\frac{1}{2} \ln \frac{2 m_e c^2 \beta^2 \gamma^2 T_{\max}}{I} - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right] \\ \beta\gamma = \frac{p}{Mc} \end{cases}$$

The nature of the V0 decay daughter can be determined using the following PID estimator:

$$n_\sigma = \frac{\langle \mathrm{d}E/\mathrm{d}x \rangle_{\text{meas.}} - \langle \mathrm{d}E/\mathrm{d}x \rangle_{\text{exp.},i}}{\sigma_{\text{TPC}}}$$

with

$$i = \mathrm{e},\ \mu,\ \pi,\ \mathrm{K},\ \mathrm{p},\ \mathrm{d},\ {}^3\mathrm{He},\ {}^4\mathrm{He}$$



Step 2: apply TPC PID selections

- V0/cascade tables reference the track table allowing access information about daughter tracks → if you try to check the PID of daughter tracks with reference to that table the code will fail!

```
float nsigma_bach = cascade.bachelor.tpcNSigmaPi();
float nsigma_pos = cascade.posTrack.tpcNSigmaPr();
float nsigma_neg = cascade.negTrack.tpcNSigmaPi();
```

Fails!

- You need to **de-reference the track** via a "***_as<>***" call, this allows you to look at the same index position in the more complete track table

```
using DaughterTracks = soa::Join<aod::TracksIU, aod::TracksExtra, aod::pidTPCPi, aod::pidTPCPr>

void process(aod::CascDatas const& cascades,
             DaughterTracks const&)
 {
   for (const auto& cascade : cascades) {
     float nsigma_bach = cascade.bachelor_as<DaughterTracks>().tpcNSigmaPi();
     float nsigma_pos = cascade.posTrack_as<DaughterTracks>().tpcNSigmaPr();
     float nsigma_neg = cascade.negTrack_as<DaughterTracks>().tpcNSigmaPi();
   }
 }
```

Works!

Step 2: apply TPC PID selections

- We would like now to apply TPC particle identification (PID) selections to our V0 candidates

```
#include "PWGLF/DataModel/LFStrangenessPIDTables.h"
```

Header needed
`LFStrangenessPIDTables.h`

```
// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
```

Add **Configurable** for each PID selection

```
// Defining the type of the daughter tracks
using DaughterTracks = soa::Join<aod::TracksIU, aod::TracksExtra, aod::pidTPCPi>;

void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<aod::V0Datas> const& V0s,
             DaughterTracks const&)
```

Subscribe to **DaughterTracks** table

Romain Schotter (romain.schotter@cern.ch)

Step 2: apply TPC PID selections

- We would like now to apply TPC particle identification (PID) selections to our V0 candidates

```
#include "PWGLF/DataModel/LFStrangenessPIDTables.h"
```

Header needed
`LFStrangenessPIDTables.h`

```
// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
```

Add **Configurable** for each PID selection

```
// Defining the type of the daughter tracks
using DaughterTracks = soa::Join<aod::TracksIU, aod::TracksExtra, aod::pidTPCPi>;

void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<aod::V0Datas> const& V0s,
             DaughterTracks const&)
```

Subscribe to **DaughterTracks** table

- ***Instructions:***
  - De-reference (= typecast V0 to **DaughterTracks**) each V0 daughter tracks
  - Apply TPC PID selections (**n = 4σ**) on each decay daughters
  - Fill a histogram with the invariant mass after V0+PID selections

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 2: apply TPC PID selections

```
// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
```

Add **Configurable** for each PID selection

```
// Defining the type of the daughter tracks
using DaughterTracks = soa::Join<aod::TracksIU, aod::TracksExtra, aod::pidTPCPi>;

void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<aod::V0Datas> const& V0s,
             DaughterTracks const&)
```

Subscribe to **DaughterTracks** table

```
  const auto& posDaughterTrack = v0.posTrack_as<DaughterTracks>();
  const auto& negDaughterTrack = v0.negTrack_as<DaughterTracks>();
```

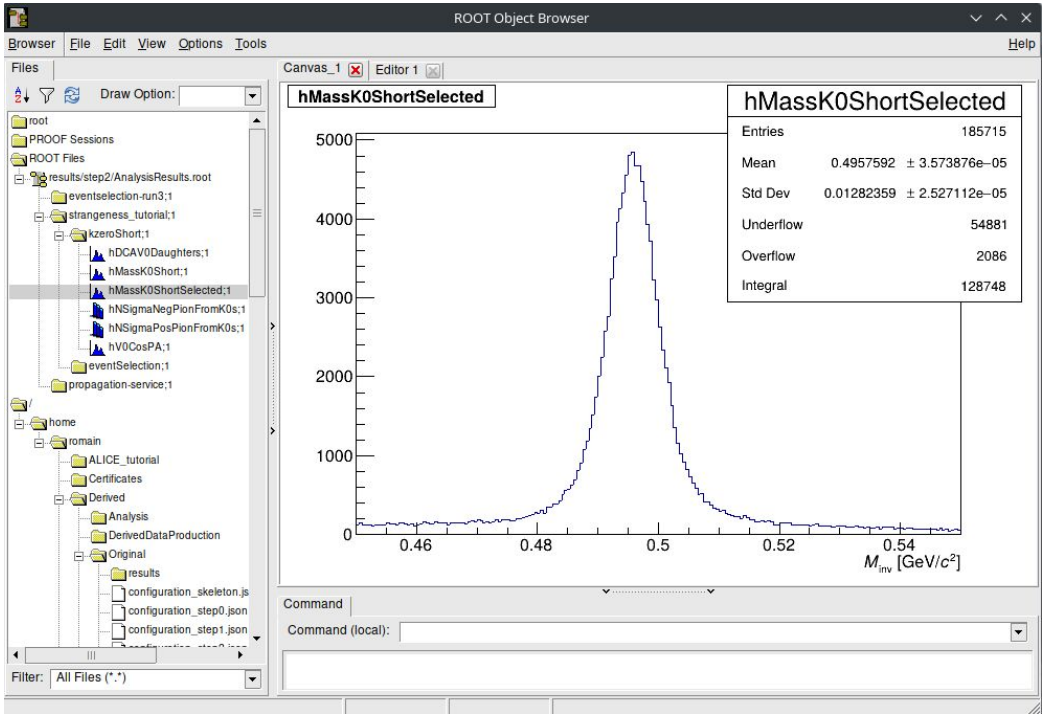
De-reference each V0 daughter tracks

```
  if (std::abs(posDaughterTrack.tpcNSigmaPi()) > NSigmaTPCPion) {
    continue;
  }
  if (std::abs(negDaughterTrack.tpcNSigmaPi()) > NSigmaTPCPion) {
    continue;
  }

  rKzeroShort.fill(HIST("hMassK0ShortSelected"), v0.mK0Short());
```

Apply TPC PID selections

Romain Schotter (romain.schotter@cern.ch)

Step 2: apply TPC PID selections



- Background is further reduced using TPC PID (**n = 4σ**) selections

Step 3: access MC information of V0s

- From the previous steps, one can reconstruct the K^0_S with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true V0s as a function of transverse momentum

→ **reconstructed** info about V0s

→ **generated** info about particles

Step 3: access MC information of V0s

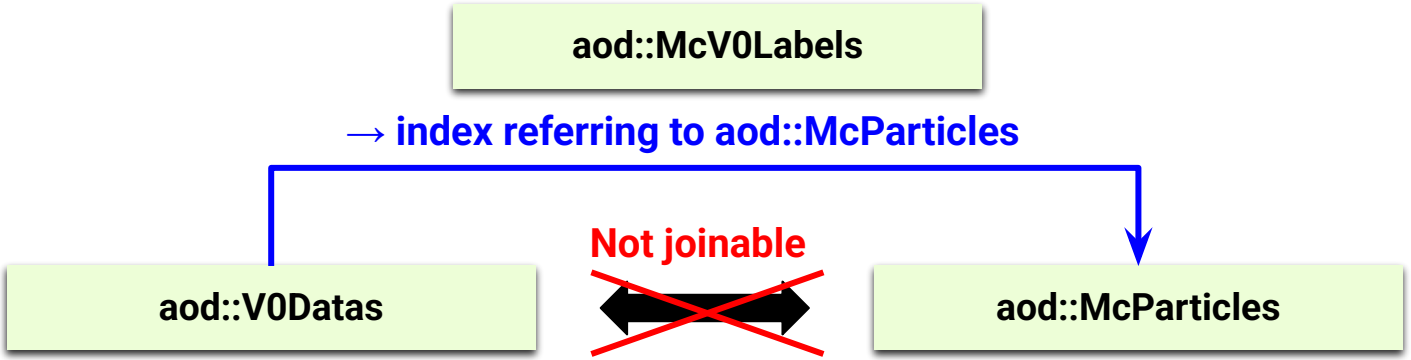
- From the previous steps, one can reconstruct the K^0_S with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true V0s as a function of transverse momentum

→ **reconstructed** info about V0s

→ **generated** info about particles

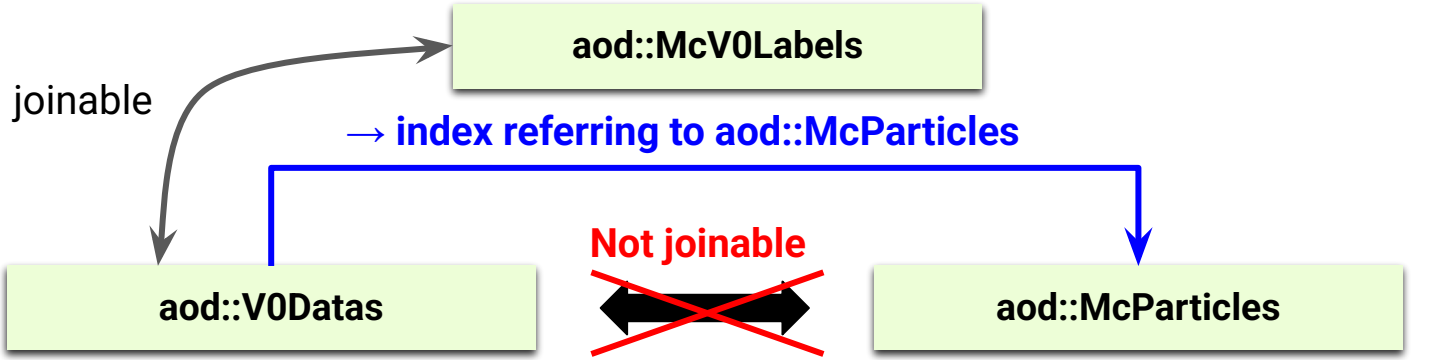
Step 3: access MC information of V0s

- From the previous steps, one can reconstruct the K^0_S with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true V0s as a function of transverse momentum



Step 3: access MC information of V0s

- From the previous steps, one can reconstruct the K^0_S with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number of reconstructed true V0s as a function of transverse momentum



Step 3: access MC information of V0s

```
void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::V0Datas, aod::McV0Labels>> const& V0s,
             DaughterTracks const&, // no need to define a variable for tracks, if we don't access them directly
             aod::McParticles const&)
```

Subscribe to **aod::McV0Labels** and **aod::McParticles** tables

```
// Checking that the V0 is a true K0s
if (v0.has_mcParticle()) {
  auto v0mcParticle = v0.mcParticle();
```

Check that cascades come from a gen. particles via **has_mcParticle()** and recover the gen. info

- ***Instructions:***
  - Download the input MC data from cernbox
  - Fill a histogram with the transverse momentum of the true reconstructed K^0_S
- ***Hints:***
  1. Have a look at the getter .pdgCode() in `LFStrangenessTable.h`
  2. Use the PDG code from ROOT: **PDG_t::kK0Short** for K^0_S
  3. For the binning in pT, use the following binning: 100 bins, from 0 to 10 GeV/*c*

In case you are facing some difficulties, please do not hesitate to ask questions!

Step 3: access MC information of V0s

```
void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::V0Datas, aod::McV0Labels>> const& V0s,
             DaughterTracks const&, // no need to define a variable for tracks, if we don't access them directly
             aod::McParticles const&)
```

```
// Checking that the V0 is a true K0s
if (v0.has_mcParticle()) {
  auto v0mcParticle = v0.mcParticle();
```

Subscribe to **aod::McV0Labels** and **aod::McParticles** tables

Check that V0s come from a gen. particles via **has_mcParticle()** and recover the gen. info

+

De-reference the corresponding entry in the **aod::McParticles** table

Step 3: access MC information of V0s

```
void process(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
             soa::Filtered<soa::Join<aod::V0Datas, aod::McV0Labels>> const& V0s,
             DaughterTracks const&, // no need to define a variable for tracks, if we don't access them directly
             aod::McParticles const&)
```

```
// Checking that the V0 is a true K0s
if (v0.has_mcParticle()) {
  auto v0mcParticle = v0.mcParticle();
  if (v0mcParticle.pdgCode() == PDG_t::kK0Short) {
    rKzeroShort.fill(HIST("hMassK0ShortTrueRec"), v0.mK0Short());
    rKzeroShort.fill(HIST("hPtK0ShortTrueRec"), v0.pt()); // To m
  }
}
```

Subscribe to **aod::McV0Labels** and **aod::McParticles** tables

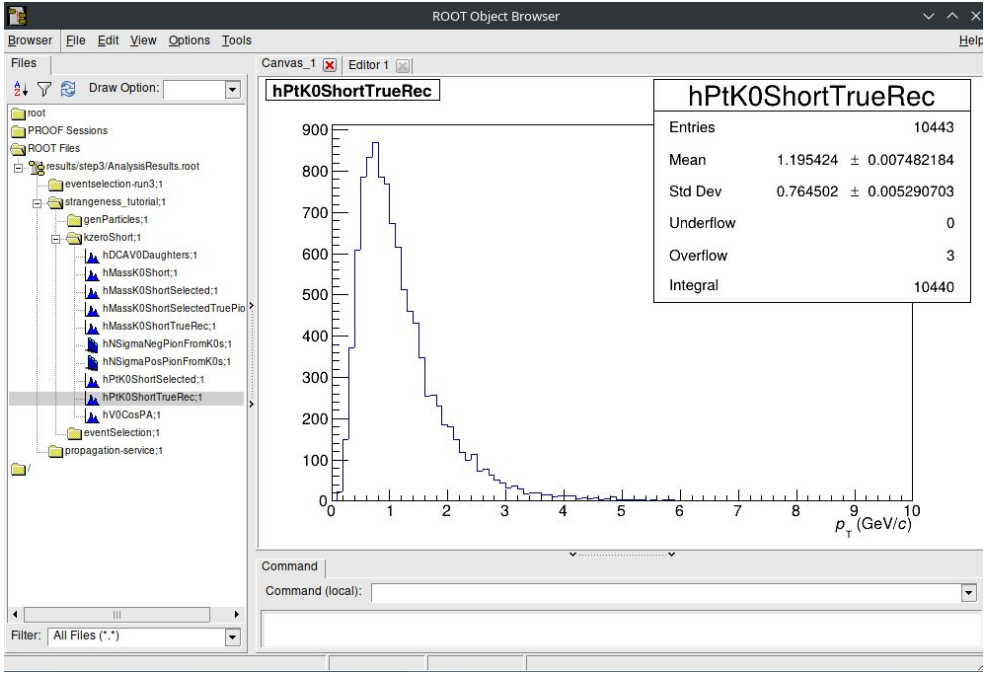
Check that V0s come from a gen. particles via **has_mcParticle()** and recover the gen. info

+

De-reference the corresponding entry in the **aod::McParticles** table

Check PDG code of gen. V0s via **pdgCode()**

Step 3: access MC information of V0s



- The number of reconstructed K^0_S has been extracted as a function of the transverse momentum

Step 3: loop over generated events

- We have the number of reconstructed true V0s as a function of transverse momentum
- Now, we need the **number of generated V0s as a function of transverse momentum**
  → requires a new process function that iterates over MC collisions

```
void processRecMC(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
                  soa::Filtered<soa::Join<aod::V0Datas, aod::McV0Labels>> const& V0s,
                  DaughterTracks const&, // no need to define a variable for tracks, if we don't acce
                  aod::McParticles const&)
```

Rename the previous **process()** function in **processRecMC()**

```
void processGenMC(soa::Filtered<aod::McCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<o2::aod::Collisions, o2::aod::McCollisionLabels, o2::aod::EvSels>>& collisions,
                  aod::McParticles const& mcParticles)
{
```

Create an additional function **processGenMC()**

Use **soa::SmallGroups** to select reconstructed collisions associated to this MC collision

```
PROCESS_SWITCH(strangeness_tutorial, processRecMC, "Process Run 3 mc, reconstructed", true);
PROCESS_SWITCH(strangeness_tutorial, processGenMC, "Process Run 3 mc, generated", true);
```

Add **PROCESS_SWITCH** to allow for 2 process functions

Romain Schotter (romain.schotter@cern.ch)

Step 3: loop over generated events

- The next step is to:
  - check that the MC collision have been reconstructed at least once using the **.size()**
  - Loop over all generated V0s
  - Fill a histogram with the generated transverse momentum for the true K^0_S

```
void processGenMC(soa::Filtered<aod::McCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<o2::aod::Collisions, o2::aod::McCollisionLabels, o2::aod::EvSels>>& collisions,
                  aod::McParticles const& mcParticles)
{
```


Romain Schotter (romain.schotter@cern.ch)

Step 3: loop over generated events

- The next step is to:
  - check that the MC collision have been reconstructed at least once using the **.size()**
  - Loop over all generated V0s
  - Fill a histogram with the generated transverse momentum for the true K^0_S

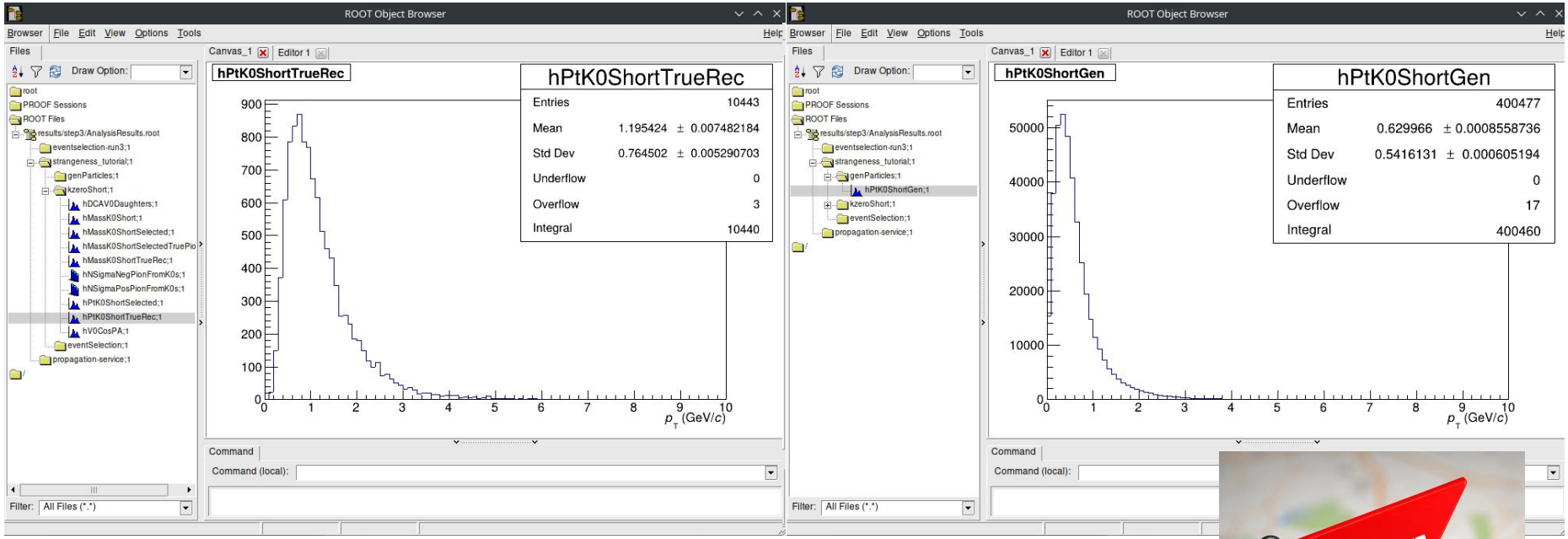
```
void processGenMC(soa::Filtered<aod::McCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<o2::aod::Collisions, o2::aod::McCollisionLabels, o2::aod::EvSels>>& collisions,
                  aod::McParticles const& mcParticles)
{
  if (collisions.size() < 1) // to process generated collisions that've been reconstructed at least once
    return;
  rEventSelection.fill(HIST("hVertexZGen"), mcCollision.posZ());
  for (const auto& mcParticle : mcParticles) {
    if (mcParticle.pdgCode() == PDG_t::kK0Short) {
      rGenParticles.fill(HIST("hPtK0ShortGen"), mcParticle.pt());
    }
  }
}
```

Loop over all generated particles + select K^0_S based on the PDG code

Fill a histogram with the gen. pt

Romain Schotter (romain.schotter@cern.ch)

Step 3: loop over generated events



- The number of reconstructed and generated K^0_S have been extracted as a function of the transverse momentum
- One step away from having the efficiency corrections…

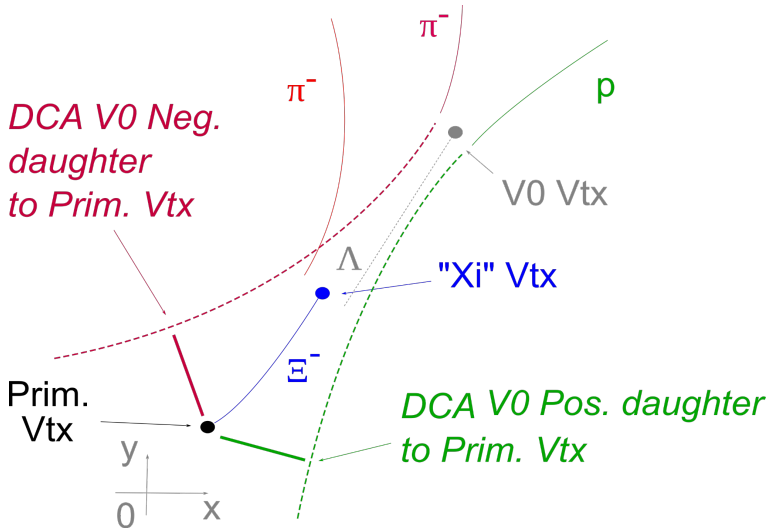
Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

$$\begin{cases} \Lambda \rightarrow \mathrm{p}\pi^- \\ \overline{\Lambda} \rightarrow \overline{\mathrm{p}}\pi^+ \end{cases}$$ c τ = 7.89 cm B.R. 63.9%

$$\begin{cases} \Xi^- \rightarrow \Lambda\pi^- \\ \overline{\Xi}^+ \rightarrow \overline{\Lambda}\pi^+ \end{cases}$$ c τ = 4.91 cm B.R. 99.9%

$$\begin{cases} \Omega^- \rightarrow \Lambda\mathrm{K}^- \\ \overline{\Omega}^+ \rightarrow \overline{\Lambda}\mathrm{K}^+ \end{cases}$$ c τ = 2.461 cm B.R. 67.8%

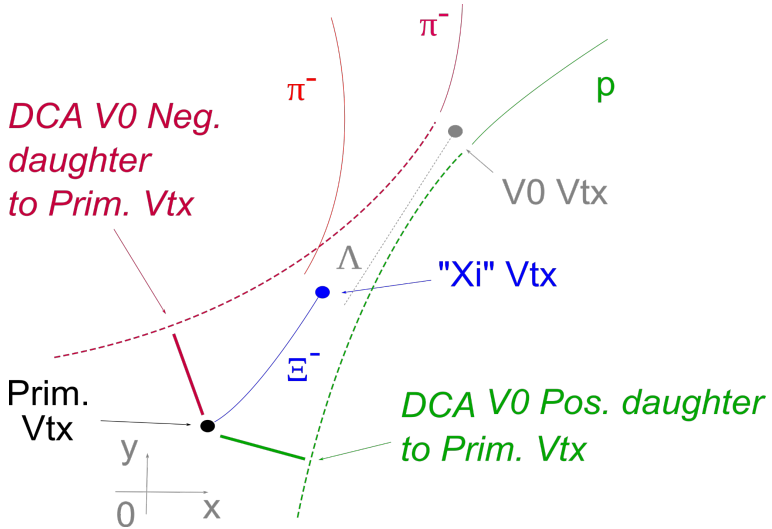


Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

V0 mass window	< 0.008 GeV/c2
DCA pos./neg. to prim. vtx	> 0.06 cm
DCA V0 to prim vtx	> 1.0 cm
DCA between V0. daughters	< 1.0 cm

DCA bach. to prim. vtx	> 0.06 cm
DCA between casc. daughters	< 1.0 cm
Casc. decay radius	> 0.5 cm
Casc. cos of pointing angle	> 0.998



Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

			Getter	Column type
V0 mass window	< 0.008 GeV/c2	→	mLambda () - o2::constants::physics::MassLambda	dynamic
DCA pos./neg. to prim. vtx	> 0.06 cm	→	dcapos/negtopv ()	standard
DCA V0 to prim vtx	> 1.0 cm	→	dcav0topv (*arg1, arg2, arg3*)	dynamic
DCA between V0. daughters	< 1.0 cm	→	dcaV0daughters ()	standard
DCA bach. to prim. vtx	> 0.06 cm	→	dcabachtopv ()	standard
DCA between casc. daughters	< 1.0 cm	→	dcacascdaughters ()	standard
Casc. decay radius	> 0.5 cm	→	cascradius ()	dynamic
Casc. cos of pointing angle	> 0.998	→	casccospa (*arg1, arg2, arg3*)	dynamic

Romain Schotter (romain.schotter@cern.ch)

Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

```
struct strangeness_tutorial {
  // Histograms are defined with HistogramRegistry
  HistogramRegistry rEventSelection{"eventSelection", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
  HistogramRegistry rKzeroShort{"kzeroShort", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
  HistogramRegistry rXi{"xi", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
  HistogramRegistry rGenParticles{"genParticles", {}, OutputObjHandlingPolicy::AnalysisObject, true, true};
```

Create a “**xi**” **HistogramRegistry**

```
void init(InitContext const&)
{
  // Axes
  AxisSpec K0ShortMassAxis = {200, 0.45f, 0.55f, "#it{M}_{inv} [GeV/#it{c}^{2}]"};
  AxisSpec XiMassAxis = {200, 1.28f, 1.36f, "#it{M}_{inv} [GeV/#it{c}^{2}]"};
  AxisSpec vertexZAxis = {nBins, -15., 15., "vrtx_{Z} [cm]"};
  AxisSpec ptAxis = {100, 0.0f, 10.0f, "#it{p}_{T} (GeV/#it{c})"};

  // Xi reconstruction
  rXi.add("hMassXi", "hMassXi", {HistType::kTH1F, {XiMassAxis}});
  rXi.add("hMassXiSelected", "hMassXiSelected", {HistType::kTH1F, {XiMassAxis}});
  rXi.add("hMassXiTrueRec", "hMassXiTrueRec", {HistType::kTH1F, {XiMassAxis}});
```

Create dedicated axis for Ξ

Create invariant mass histograms for Ξ and store them in their corresponding **HistogramRegistry**

Romain Schotter (romain.schotter@cern.ch)

Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

```
// Configurable parameters for cascade selection
Configurable<double> cascadesetting_cospa{"cascadesetting_cospa", 0.98, "Casc CosPA"}; // double -> N.B. dcos(x)/dx = 0 at x=0
Configurable<float> cascadesetting_dcacascdau{"cascadesetting_dcacascdau", 1.0, "DCA cascade daughters"};
Configurable<float> cascadesetting_dcabachtopv{"cascadesetting_dcabachtopv", 0.1, "DCA bachelor to PV"};
Configurable<float> cascadesetting_mindcav0topv{"cascadesetting_mindcav0topv", 0.01, "minimum V0 DCA to PV"};
Configurable<float> cascadesetting_cascradius{"cascadesetting_cascradius", 0.5, "cascradius"};
Configurable<float> cascadesetting_v0masswindow{"cascadesetting_v0masswindow", 0.01, "v0 mass window"};

Configurable<float> cascade_dcanegtopv{"dcanegtopv", 0.06, "DCA Neg To PV"};
Configurable<float> cascade_dcapostopv{"dcapostopv", 0.06, "DCA Pos To PV"};

// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
Configurable<float> NSigmaTPCProton{"NSigmaTPCProton", 4, "NSigmaTPCProton"};
```

Add **Configurable** for each cascade selection

Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

```
// Configurable parameters for cascade selection
Configurable<double> cascadesetting_cospa{"cascadesetting_cospa", 0.98, "Casc CosPA"}; // double -> N.B. dcos(x)/dx = 0 at x=0
Configurable<float> cascadesetting_dcacascdau{"cascadesetting_dcacascdau", 1.0, "DCA cascade daughters"};
Configurable<float> cascadesetting_dcabachtopv{"cascadesetting_dcabachtopv", 0.1, "DCA bachelor to PV"};
Configurable<float> cascadesetting_mindcav0topv{"cascadesetting_mindcav0topv", 0.01, "minimum V0 DCA to PV"};
Configurable<float> cascadesetting_cascradius{"cascadesetting_cascradius", 0.5, "cascradius"};
Configurable<float> cascadesetting_v0masswindow{"cascadesetting_v0masswindow", 0.01, "v0 mass window"};

Configurable<float> cascade_dcanegtopv{"dcanegtopv", 0.06, "DCA Neg To PV"};
Configurable<float> cascade_dcapostopv{"dcapostopv", 0.06, "DCA Pos To PV"};

// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
Configurable<float> NSigmaTPCProton{"NSigmaTPCProton", 4, "NSigmaTPCProton"};
```

Add **Configurable** for each cascade selection

- ***Instructions:***
  - Select the candidates passing the topological selections
  - Fill a histogram with the invariant mass after cascade selection

- ***Hints:***
  1. Have a look at the **aod::CascDatas** table in `LFStrangenessTables.h` to find the columns storing the selection variables
  2. you can use **Filter** to select the candidates but it cannot cut on dynamic columns

Step 4: select cascades

- We would like now to repeat the previous steps for Ξ, starting by selecting the candidates based on their decay kinematics and topology

```
// Configurable parameters for cascade selection
Configurable<double> cascadesetting_cospa{"cascadesetting_cospa", 0.98, "Casc CosPA"}; // double -> N.B. dcos(x)/dx = 0 at x=0
Configurable<float> cascadesetting_dcacascdau{"cascadesetting_dcacascdau", 1.0, "DCA cascade daughters"};
Configurable<float> cascadesetting_dcabachtopv{"cascadesetting_dcabachtopv", 0.1, "DCA bachelor to PV"};
Configurable<float> cascadesetting_mindcav0topv{"cascadesetting_mindcav0topv", 0.01, "minimum V0 DCA to PV"};
Configurable<float> cascadesetting_cascradius{"cascadesetting_cascradius", 0.5, "cascradius"};
Configurable<float> cascadesetting_v0masswindow{"cascadesetting_v0masswindow", 0.01, "v0 mass window"};

Configurable<float> cascade_dcanegtopv{"dcanegtopv", 0.06, "DCA Neg To PV"};
Configurable<float> cascade_dcapostopv{"dcapostopv", 0.06, "DCA Pos To PV"};

// Configurable parameters for PID selection
Configurable<float> NSigmaTPCPion{"NSigmaTPCPion", 4, "NSigmaTPCPion"};
Configurable<float> NSigmaTPCProton{"NSigmaTPCProton", 4, "NSigmaTPCProton"};
```

Add **Configurable** for each cascade selection

```
// Filters on Cascades
Filter preFilterCascades = (nabs(aod::cascdata::dcabachtopv) > cascadesetting_dcabachtopv &&
                            aod::cascdata::dcaV0daughters < v0setting_dcav0dau &&
                            nabs(aod::cascdata::dcapostopv) > cascade_dcapostopv &&
                            nabs(aod::cascdata::dcanegtopv) > cascade_dcanegtopv &&
                            nabs(aod::cascdata::dcabachtopv) > cascadesetting_dcabachtopv &&
                            aod::cascdata::dcacascdaughters < cascadesetting_dcacascdau);
```

```
void processRecMC(soa::Filtered<soa::Join<aod::Collisions, aod::EvSels>>::iterator const& collision,
                  soa::Filtered<soa::Join<aod::CascDatas, aod::McCascLabels>> const& Cascades,
                  soa::Filtered<soa::Join<aod::V0Datas, aod::McV0Labels>> const& V0s,
                  DaughterTracks const&,
                  aod::McParticles const&)
```

Filter on **regular** columns

Romain Schotter (romain.schotter@cern.ch)

Step 4: select cascades

```
// Cascades
for (const auto& casc : Cascades) {
  const auto& bachDaughterTrackCasc = casc.bachelor_as<DaughterTracks>();
  const auto& posDaughterTrackCasc = casc.posTrack_as<DaughterTracks>();
  const auto& negDaughterTrackCasc = casc.negTrack_as<DaughterTracks>();

  rXi.fill(HIST("hMassXi"), casc.mXi());

  // Cut on dynamic columns
  if (casc.casccosPA(collision.posX(), collision.posY(), collision.posZ()) < cascadesetting_cospa)
    continue;
  if (std::abs(casc.mLambda() - o2::constants::physics::MassLambda) > cascadesetting_v0masswindow)
    continue;
  if (casc.dcav0topv(collision.posX(), collision.posY(), collision.posZ()) < cascadesetting_mindcav0topv)
    continue;
  if (casc.cascradius() < cascadesetting_cascradius)
    continue;

  if (casc.sign() < 0) {
    if (std::abs(posDaughterTrackCasc.tpcNSigmaPr()) > NSigmaTPCProton) {
      continue;
    }
    if (std::abs(negDaughterTrackCasc.tpcNSigmaPi()) > NSigmaTPCPion) {
      continue;
    }
  } else {
    if (std::abs(negDaughterTrackCasc.tpcNSigmaPr()) > NSigmaTPCProton) {
      continue;
    }
    if (std::abs(posDaughterTrackCasc.tpcNSigmaPi()) > NSigmaTPCPion) {
      continue;
    }
  }
  if (std::abs(bachDaughterTrackCasc.tpcNSigmaPi()) > NSigmaTPCPion) {
    continue;
  }
```

De-reference each cascade daughter tracks

Cut on **dynamic** columns in the loop

Apply TPC PID selections

Step 4: access cascade MC information

```
rXi.fill(HIST("hMassXiSelected"), casc.mXi());
// Checking that the cascade is a true Xi
if (casc.has_mcParticle()) {
  const auto cascmcParticle = casc.mcParticle();
  if (std::abs(cascmcParticle.pdgCode()) == PDG_t::kXiMinus) {
    rXi.fill(HIST("hMassXiTrueRec"), casc.mXi());
  }
}
```

Fill invariant mass histograms after the topological selections

Check that cascades come from a gen. particles via **has_mcParticle()** and recover the gen. info

+

De-reference the corresponding entry in the **aod::McParticles** table

Check PDG code of gen. cascade via **pdgCode()**

Step 4: access cascade MC information

```cpp
void processGenMC(soa::Filtered<aod::McCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<o2::aod::Collisions, o2::aod::McCollisionLabels, o2::aod::EvSels>>& collisions,
                  aod::McParticles const& mcParticles)
{
  if (collisions.size() < 1) // to process generated collisions that've been reconstructed at least once
    return;
  rEventSelection.fill(HIST("hVertexZGen"), mcCollision.posZ());
  for (const auto& mcParticle : mcParticles) {
    if (mcParticle.pdgCode() == PDG_t::kK0Short) {
      rGenParticles.fill(HIST("hPtK0ShortGen"), mcParticle.pt());
    }
    if (std::abs(mcParticle.pdgCode()) == PDG_t::kXiMinus) {
      rGenParticles.fill(HIST("hPtXiGen"), mcParticle.pt());
    }
  }
}
```

Select Ξ based on the PDG code

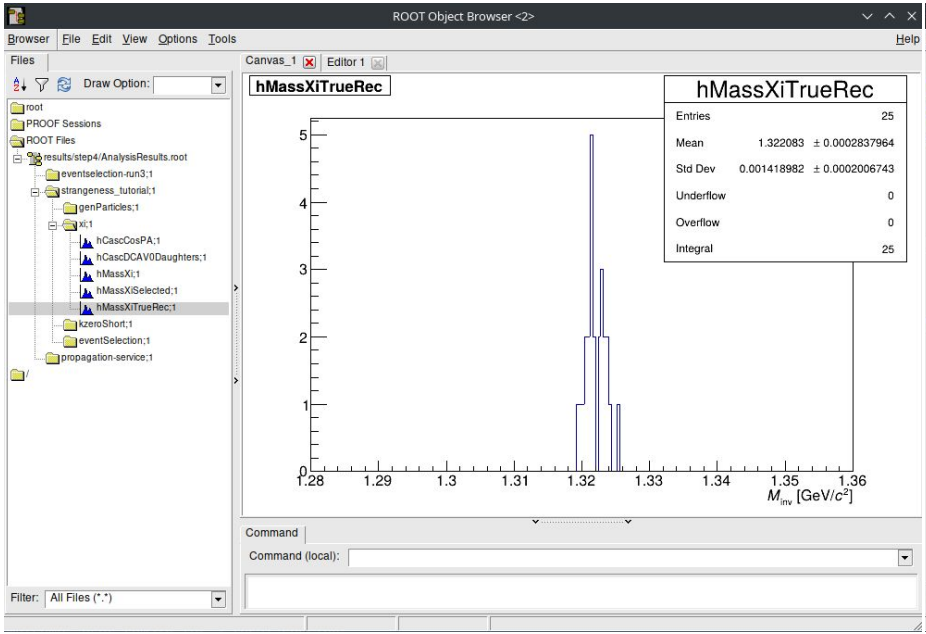
Fill a histogram with the gen. pt

```json
"strangeness_tutorial": {
    "nBins": "100",
    "cutzvertex": "10",
    "v0setting_dcav0dau": "1",
    "v0setting_dcapostopv": "0.06",
    "v0setting_dcanegtopv": "0.06",
    "v0setting_cospa": "0.998",
    "v0setting_radius": "1.2",
    "cascadesetting_cospa": "0.998",
    "cascadesetting_dcacascdau": "1",
    "cascadesetting_dcabachtopv": "0.06",
    "cascadesetting_mindcav0topv": "1",
    "cascadesetting_cascradius": "0.5",
    "cascadesetting_v0masswindow": "0.010",
    "dcanegtopv": "0.06",
    "dcapostopv": "0.06",
    "NSigmaTPCPion": "4",
    "processRecMC": "true",
    "processGenMC": "true"
}
```

Provide the appropriate cuts in the **json configuration file**

The variables must have the same name as their corresponding configurable!

Step 4: access cascade MC information



- The number of reconstructed and generated Ξ have been extracted as a function of the transverse momentum
- One step away from having the efficiency corrections…