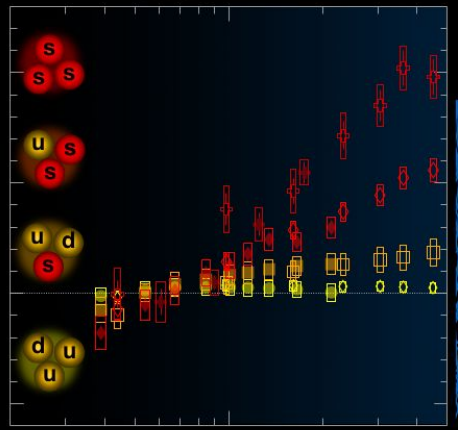
Strangeness analyses

O2 Analysis Tutorial 15/10/2024*

Romain Schotter

Austrian Academy of Sciences, Austria

romain.schotter@cern.ch



* based on the introduction session from the O2AT 07/11/2023 from Roman Nepeivoda

Strangeness reconstruction with ALICE

The (multi-)strange hadrons reconstruction relies on two weak decay topologies:

V ^{0} : neutral particle decaying into a pair of charged particles (V-shaped decay)

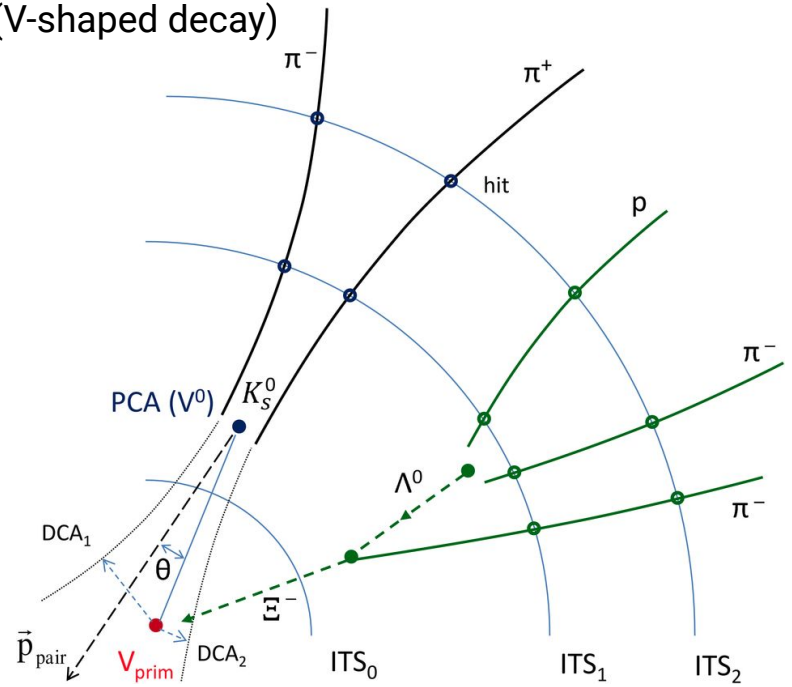
$K^0_S \rightarrow \pi^+\pi^-$ c τ = 2.6844 cm B.R. 69.2%

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

Cascade : charged particle decaying into a V ^{0} + charged particle (bachelor)

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

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



The strangeness analysis workflow

Asynchronous reconstruction (O^2)

AO2D.root

Secondary vertex reconstruction: SVertexer.cxx
Initial loose cut parameters: SVertexerParams.h

V0 table: aod::V0s
cascade table: aod::Cascades

The strangeness analysis workflow

Asynchronous reconstruction (O^2)

AO2D.root

Secondary vertex reconstruction: SVertexer.cxx
Initial loose cut parameters: SVertexerParams.h

V0 table: aod::V0s
cascade table: aod::Cascades

Run 3 V0 table (version 002)
Header file: Framework/Core/include/Framework/AnalysisDataModel.h
Is used in:

- o2::aod::V0s = o2::aod::V0s_002

Name		Getter	Type	Comment
o2::soa::Index	GI	globalIndex	int64_t	
o2::aod::v0::CollisionId	I	collisionId	int32	Collision index
o2::aod::v0::PosTrackId	I	posTrackId	int	Positive track
o2::aod::v0::NegTrackId	I	negTrackId	int	Negative track
o2::aod::v0::V0Type		v0Type	uint8_t	custom bitmap for various selections (see below)
o2::aod::v0::IsStandardV0	D	isStandardV0	bool	is standard V0
o2::aod::v0::IsPhotonV0	D	isPhotonV0	bool	is TPC-only V0 for which the photon-mass-hypothesis was good
o2::aod::v0::IsCollinearV0	D	isCollinearV0	bool	is V0 for which the photon-mass-hypothesis was good and was fitted collinearly

Run 3 cascade table
Header file: Framework/Core/include/Framework/AnalysisDataModel.h
Is used in:

- o2::aod::Cascades = o2::aod::Cascades_001

Name		Getter	Type	Comment
o2::soa::Index	GI	globalIndex	int64_t	
o2::aod::cascade::CollisionId	I	collisionId	int32	Collision index
o2::aod::cascade::V0Id	I	v0Id	int32	V0 index
o2::aod::cascade::BachelorId	I	bachelorId	int	Bachelor track index

The strangeness analysis workflow

Asynchronous reconstruction (O^2)

AO2D.root

Secondary vertex reconstruction: SVertexer.cxx
Initial loose cut parameters: SVertexerParams.h

V0 table: aod::V0s
cascade table: aod::Cascades

ONLY IDs are stored at these tables
NO topological/kinematic variables

Run 3 V0 table (version 002)
Header file: Framework/Core/include/Framework/AnalysisDataModel.h
Is used in:

- o2::aod::V0s = o2::aod::V0s_002

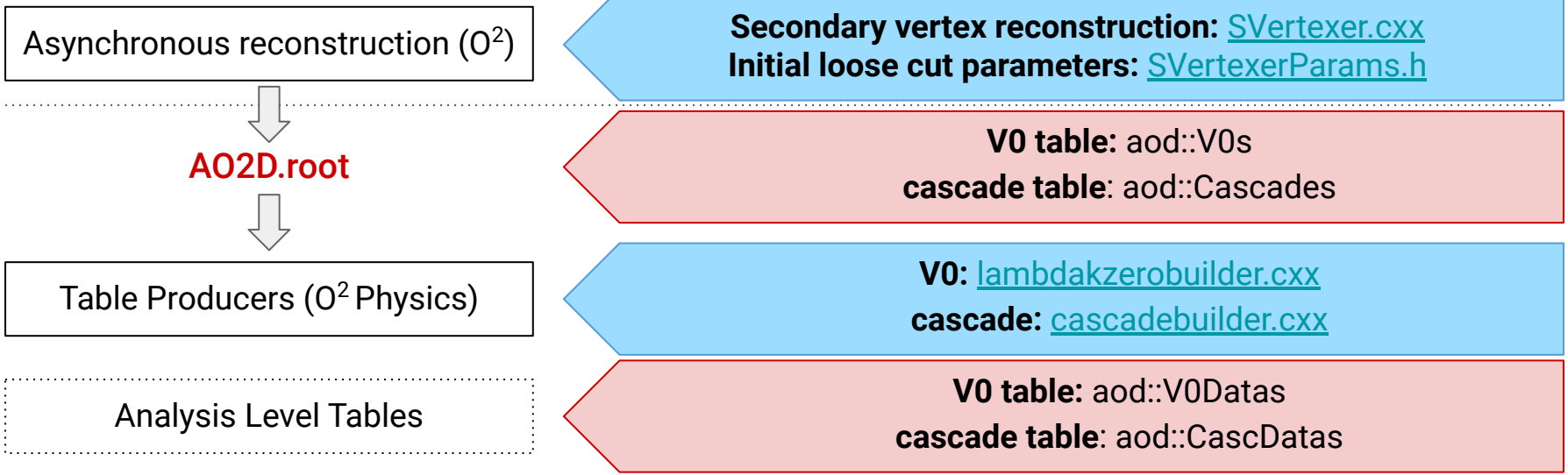
Name		Getter	Type	Comment
o2::soa::Index	GI	globalIndex	int64_t	
o2::aod::v0::CollisionId	I	collisionId	int32	Collision index
o2::aod::v0::PosTrackId	I	posTrackId	int	Positive track
o2::aod::v0::NegTrackId	I	negTrackId	int	Negative track
o2::aod::v0::V0Type		v0Type	uint8_t	custom bitmap for various selections (see below)
o2::aod::v0::IsStandardV0	D	isStandardV0	bool	is standard V0
o2::aod::v0::IsPhotonV0	D	isPhotonV0	bool	is TPC-only V0 for which the photon-mass-hypothesis was good
o2::aod::v0::IsCollinearV0	D	isCollinearV0	bool	is V0 for which the photon-mass-hypothesis was good and was fitted collinearly

Run 3 cascade table
Header file: Framework/Core/include/Framework/AnalysisDataModel.h
Is used in:

- o2::aod::Cascades = o2::aod::Cascades_001

Name		Getter	Type	Comment
o2::soa::Index	GI	globalIndex	int64_t	
o2::aod::cascade::CollisionId	I	collisionId	int32	Collision index
o2::aod::cascade::V0Id	I	v0Id	int32	V0 index
o2::aod::cascade::BachelorId	I	bachelorId	int	Bachelor track index

The strangeness analysis workflow



Analysis Level Tables: aod::V0Data(Ext)

#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
Calculated	v0data::PxPos	pxpos
	v0data::PyPos	pypos
	v0data::PzPos	pzpos
	v0data::PxNeg	pxneg
	v0data::PyNeg	pyneg
	v0data::PzNeg	pxneg
	v0data::X	x
	v0data::Y	y
	v0data::Z	z
	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
	...	...
Derived (expression)	v0dataext::Px	px
	v0dataext::Py	py
	v0dataext::Pz	pz

Analysis Level Tables: aod::V0Data(Ext)

#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	v0data::X	x
Decay vertex position	v0data::Y	y
	v0data::Z	z
Topological variables	v0data::DCAV0Daughters	dcav0daughters
	v0data::DCAPosToPV	dcapostopv
	v0data::DCANegToPV	dcanegtopv

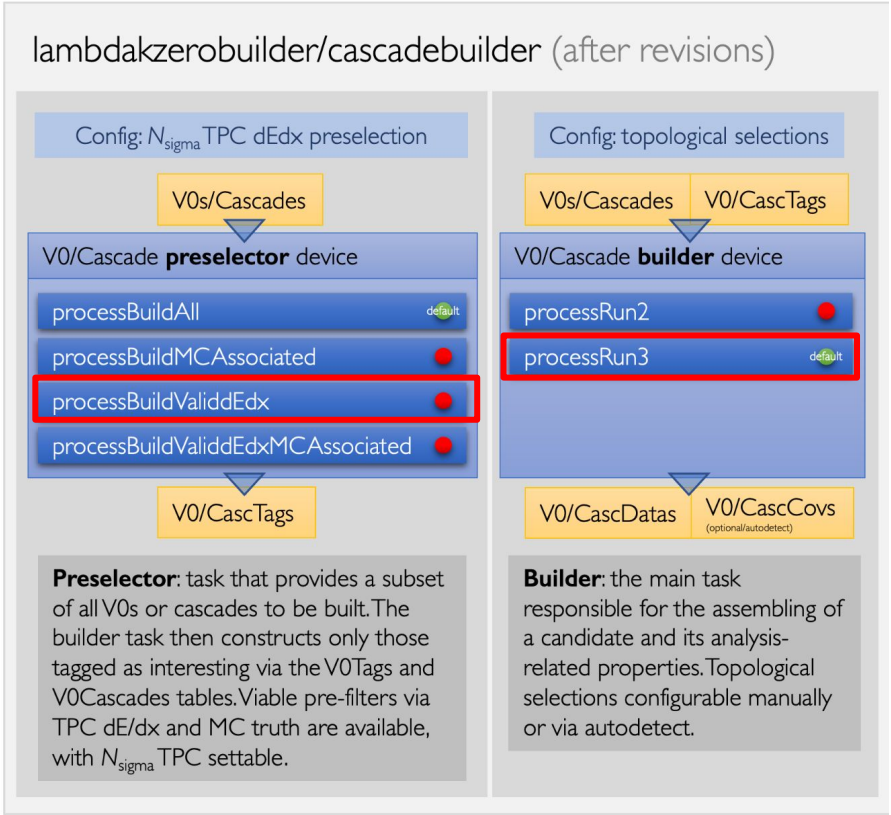
Derived (dynamic)	v0data::Pt	pt
	v0data::V0Radius	v0radius
	v0data::V0CosPA	v0cospa
	v0data::DCAV0ToPV	dcav0topv
	v0data::MLambda	mLambda
	v0data::MAntiLambda	mAntiLambda
	v0data::MK0Short	mK0Short
	v0data::YK0Short	yK0Short
	v0data::YLambda	yLambda
	v0data::Eta	eta
	v0data::Phi	phi
	...	...
Derived (expression)	v0dataext::Px	px
	v0dataext::Py	py
	v0dataext::Pz	pz

Analysis Level Tables: aod::CascData(Ext)

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

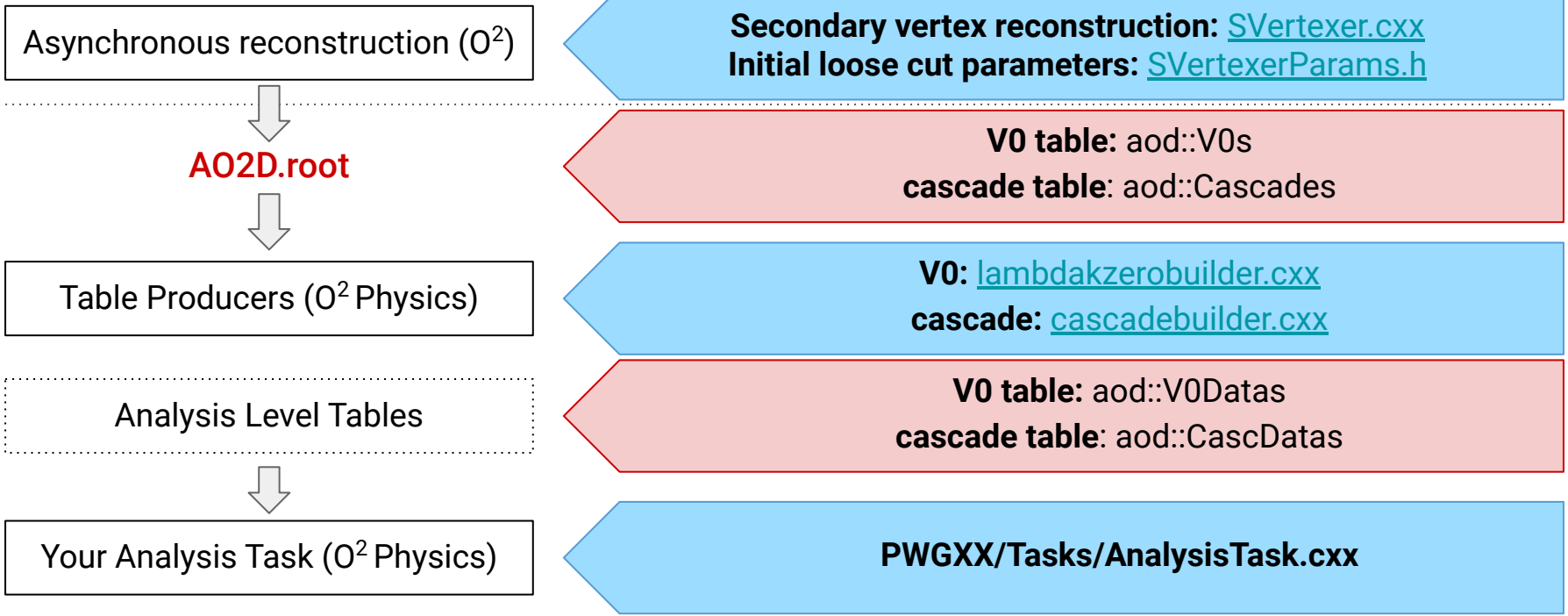
	Name	Getter
Bound	cascade::V0Id	v0Id
	cascade::BachelorId	bachelorId
	cascade::CollisionId	collisionId
Momenta of daughter tracks	cascdata::Sign	sign
	cascdata::PxPos(Neg)	pxpos(neg)
	cascdata::PyPos(Neg)	pypos(neg)
	cascdata::PzPos(Neg)	pzpos(neg)
	cascdata::PxBach	pxbach
	cascdata::PyBach	pybach
	cascdata::PzBach	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
Derived (dynamic)	cascdata::DCANegToPV	dcanegtopv
	cascdata::DCABachToPV	dcabachtopv
	cascdata::Pt	pt
	...	...
	cascdata::V0Radius	v0radius
	cascdata::CascRadius	cascradius
	cascdata::V0CosPA	v0cosPA
	cascdata::CascCosPA	casccosPA
	cascdata::MLambda	mLambda
	cascdata::MXi	mXi
	cascdata::MOmega	mOmega
	cascdata::YXi	yXi
	...	...
Derived (expression)	cascdataext::Pt	pt
	cascdataext::P	p
	cascdataext::PxLambda	pxlambda
	...	...

V0/cascade table producer task

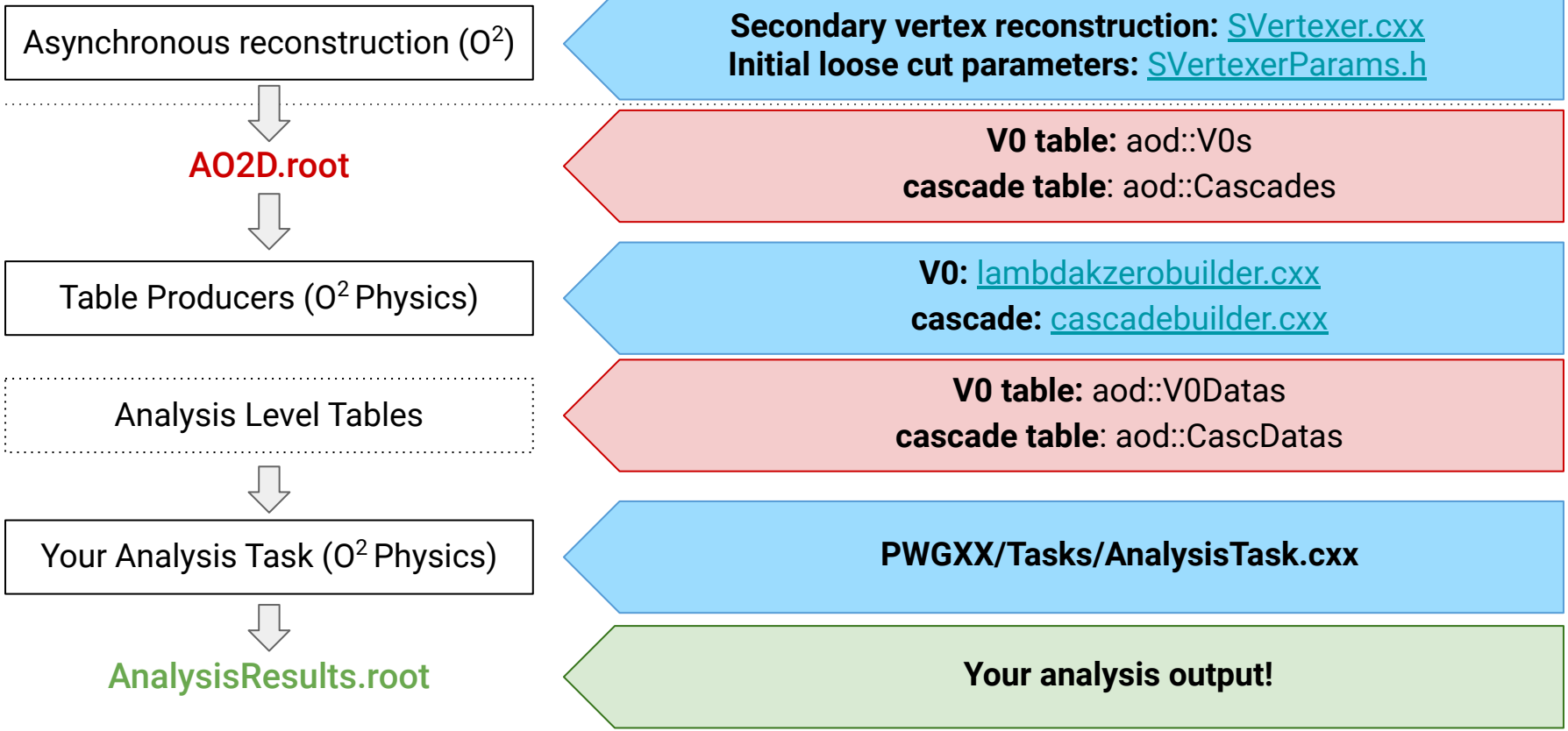


- V0/cascade tables produced in the `lambdakzerobuilder.cxx` and `cascadebuilder.cxx`
- **Pre-selector** device:
  - **processBuild** options → select V0s/casc. based on TPC N σ selections
  - **Skipper** options to optionally skip V0s not used in casc. / casc. not used in casc. tracking
- **Builder** device:
  - Look at candidates tagged as interesting → **buildStrangenessTables**
- Output:
  - **V0Datas**: regular V0 analysis table
  - **CascDatas**: regular cascade analysis table
  - **TraCascDatas**: tracked cascade analysis table
  - **KFCascDatas**: KF cascade analysis table

The strangeness analysis workflow



The strangeness analysis workflow



The strangeness analysis workflow

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

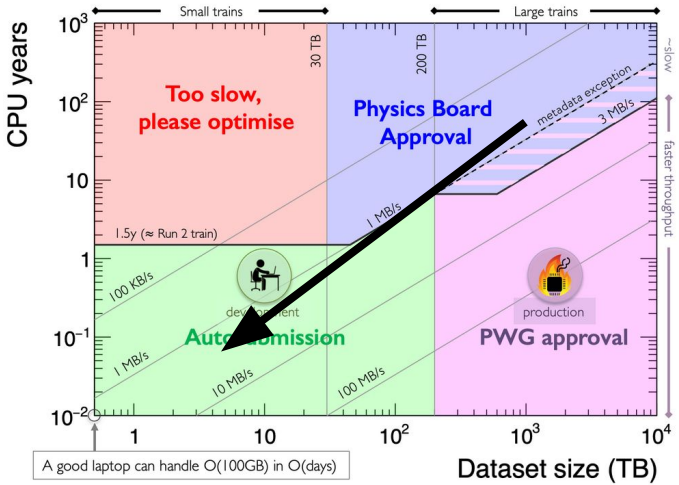
- Only information needed for your analysis are stored in derived data

  → 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)

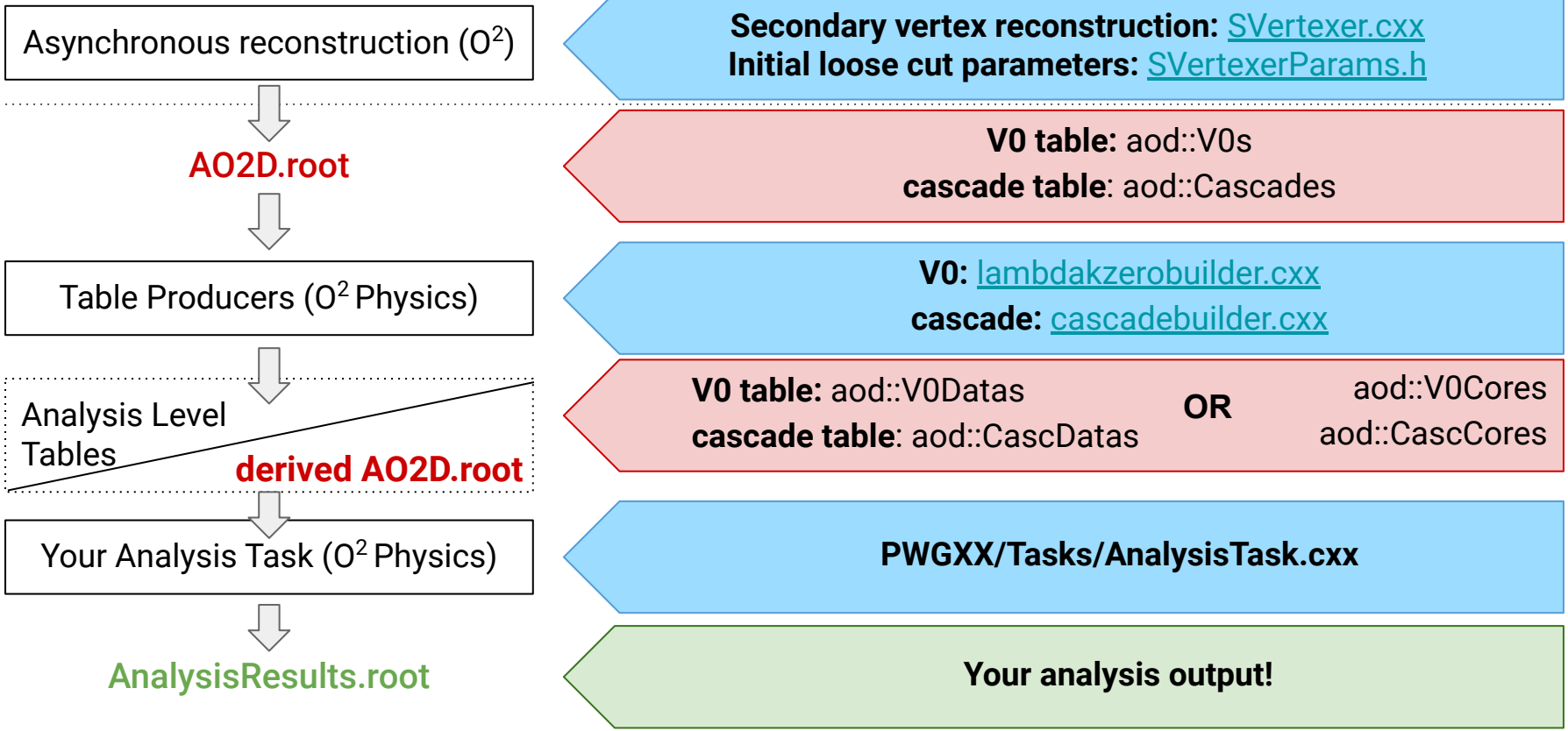
- Two types of derived data:

  - **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



The strangeness analysis workflow



Strangeness Tutorial Task

- Dedicated to the basics of the interaction with the strangeness workflow

- This tutorial focuses on an analysis of Pb-Pb collisions using derived data*
- You will learn how to
  - Produce derived tables
  - Structure your analysis task
  - Reconstruct cascades
  - Apply (TPC and TOF) PID, topological and kinematic selections
  - Access MC information of the cascades

- Tutorial task is organised in the form of 5 sequential steps
- Code is already present in your **O2Physics**
  please update the folder before the HANDS-ON session

*for an analysis on pp collisions, have a look at the O2 AT 08/11/23

Preparation: how to produce derived data?

- **Locally,** run the workflows to produce the derived tables and specify the ones you want to save in the `OutputDirector.json` file

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

o2-analysis-pid-tof-base ${OPTION} |
o2-analysis-event-selection ${OPTION} |
o2-analysis-lf-lambdakzerobuilder ${OPTION} |
o2-analysis-lf-cascadebuilder ${OPTION} |
o2-analysis-multiplicity-table ${OPTION} |
o2-analysis-centrality-table ${OPTION} |
o2-analysis-lf-epvector ${OPTION} |
o2-analysis-centrality-qa ${OPTION} |
o2-analysis-ud-sgcand-producer ${OPTION} |
o2-analysis-timestamp ${OPTION} |
o2-analysis-ft0-corrected-table ${OPTION} |
o2-analysis-track-propagation ${OPTION} |
o2-analysis-pid-tpc-base ${OPTION} |
o2-analysis-pid-tpc ${OPTION} |
o2-analysis-trackselection ${OPTION} |
o2-analysis-pid-tof-full ${OPTION} |
o2-analysis-pid-tof-beta ${OPTION} |
o2-analysis-lf-strangederivedbuilder ${OPTION} --aod-file @input_data.txt --aod-writer-json OutputDirector.json > "$LOGFILE" 2>&1
```

Text file containing the path to the original AO2D.root file

JSON file specifying which table to save on disk

Preparation: how to produce derived data?

- **Locally,** run the workflows to produce the derived tables and specify the ones you want to save in the `OutputDirector.json` file

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

o2-analysis-pid-tof-base ${OPTION} |
o2-analysis-event-selection ${OPTION} |
o2-analysis-lf-lambdakzerobuilder ${OPTION} |
o2-analysis-lf-cascadebuilder ${OPTION} |
o2-analysis-multiplicity-table ${OPTION} |
o2-analysis-centrality-table ${OPTION} |
o2-analysis-lf-epvector ${OPTION} |
o2-analysis-centrality-qa ${OPTION} |
o2-analysis-ud-sgcand-producer ${OPTION} |
o2-analysis-timestamp ${OPTION} |
o2-analysis-ft0-corrected-table ${OPTION} |
o2-analysis-track-propagation ${OPTION} |
o2-analysis-pid-tpc-base ${OPTION} |
o2-analysis-pid-tpc ${OPTION} |
o2-analysis-trackselection ${OPTION} |
o2-analysis-pid-tof-full ${OPTION} |
o2-analysis-pid-tof-beta ${OPTION} |
o2-analysis-lf-strangederivedbuilder ${OPTION} --aod-file @input_data.txt --aod-writer-json OutputDirector.json > "$LOGFILE" 2>&1
```

Text file containing the path to the original AO2D.root file

JSON file specifying which table to save on disk

Workflow with all the helper tasks

Preparation: how to produce derived data?

- **Locally,** run the workflows to produce the derived tables and specify the ones you want to save in the `OutputDirector.json` file

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

#directory of this script
DIR_THIS=$PWD

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

o2-analysis-pid-tof-base ${OPTION} |
o2-analysis-event-selection ${OPTION} |
o2-analysis-lf-lambdakzerobuilder ${OPTION} |
o2-analysis-lf-cascadebuilder ${OPTION} |
o2-analysis-multiplicity-table ${OPTION} |
o2-analysis-centrality-table ${OPTION} |
o2-analysis-lf-epvector ${OPTION} |
o2-analysis-centrality-qa ${OPTION} |
o2-analysis-ud-sgcand-producer ${OPTION} |
o2-analysis-timestamp ${OPTION} |
o2-analysis-ft0-corrected-table ${OPTION} |
o2-analysis-track-propagation ${OPTION} |
o2-analysis-pid-tpc-base ${OPTION} |
o2-analysis-pid-tpc ${OPTION} |
o2-analysis-trackselection ${OPTION} |
o2-analysis-pid-tof-full ${OPTION} |
o2-analysis-pid-tof-beta ${OPTION} |
o2-analysis-lf-strangederivedbuilder ${OPTION} --aod-file @input_data.txt --aod-writer-json OutputDirector.json > "$LOGFILE" 2>&1
```

Text file containing the path to the original AO2D.root file

JSON file specifying which table to save on disk

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

Workflow with all the helper tasks

Preparation: how to produce derived data?

- **Locally,** run the workflows to produce the derived tables and specify the ones you want to save in the `OutputDirector.json` file

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

#directory of this script
DIR_THIS=$PWD

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

o2-analysis-pid-tof-base ${OPTION} |
o2-analysis-event-selection ${OPTION} |
o2-analysis-lf-lambdakzerobuilder ${OPTION} |
o2-analysis-lf-cascadebuilder ${OPTION} |
o2-analysis-multiplicity-table ${OPTION} |
o2-analysis-centrality-table ${OPTION} |
o2-analysis-lf-epvector ${OPTION} |
o2-analysis-centrality-qa ${OPTION} |
o2-analysis-ud-sgcand-producer ${OPTION} |
o2-analysis-timestamp ${OPTION} |
o2-analysis-ft0-corrected-table ${OPTION} |
o2-analysis-track-propagation ${OPTION} |
o2-analysis-pid-tpc-base ${OPTION} |
o2-analysis-pid-tpc ${OPTION} |
o2-analysis-trackselection ${OPTION} |
o2-analysis-pid-tof-full ${OPTION} |
o2-analysis-pid-tof-beta ${OPTION} |
o2-analysis-lf-strangederivedbuilder ${OPTION} --aod-file @input_data.txt --aod-writer-json OutputDirector.json > "$LOGFILE" 2>&1

# report status
rc=$?
if [ $rc -eq ]; then
    echo "No problems!"
    mkdir -p ${DIR_THIS}/results/${STEP}
    mv AnalysisResults.root ${DIR_THIS}/results/${STEP}/AnalysisResults.root
    mv AO2D.root ${DIR_THIS}/results/${STEP}/AO2D.root
    mv dpl-config.json ${DIR_THIS}/jsonConfigs/${STEP}.json
else
    echo "Error: Exit code ${rc}"
    echo "Check the log file ${LOGFILE}"
    exit rc
fi
```

Text file containing the path to the original AO2D.root file

JSON file specifying which table to save on disk

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

`strangeness_pbpb_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"
#include "Framework/O2DatabasePDGPlugin.h"

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

struct strangeness_pbpb_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)"};

  // PDG data base
  Service<o2::framework::O2DatabasePDG> pdgDB;

  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

Preparation: running an analysis task

```
51
52    // Defining filters for events (event selection)
53    // Processed events will be already fulfilling the event selection requirements
54    Filter eventFilter = (o2::aod::evsel::sel8 == true);
55    Filter posZFilter = (nabs(o2::aod::collision::posZ) < cutzvertex);
56
57    void process(soa::Filtered<soa::Join<aod::StraCollisions, aod::StraEvSels>>::iterator const& collision)
58    {
59      // Fill the event counter
60      rEventSelection.fill(HIST("hVertexZRec"), collision.posZ());
61    }
62  };
63
64  WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
65  {
66    return WorkflowSpec{
67      adaptAnalysisTask<strangeness_pbpb_tutorial>(cfgc)};
68  }
```

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

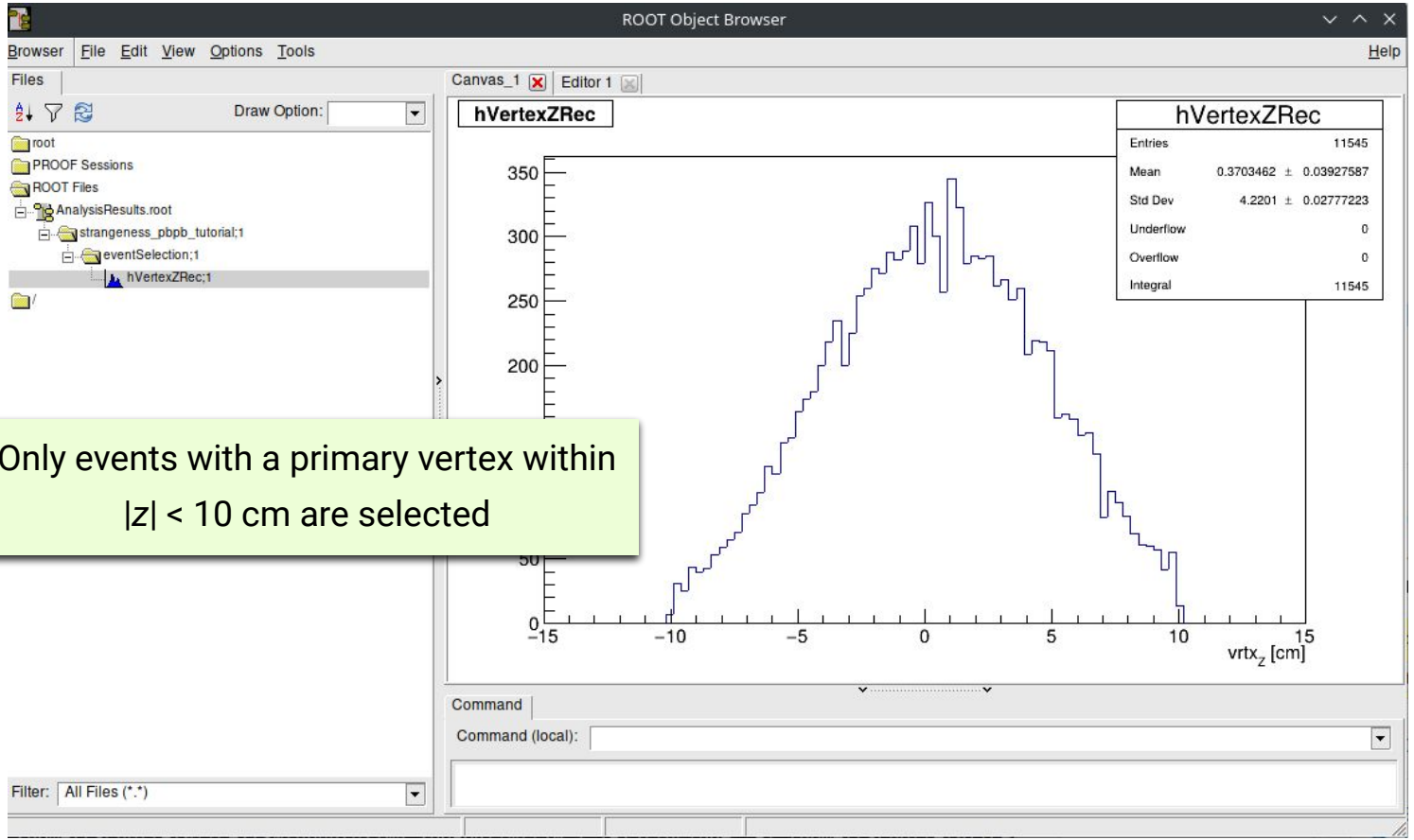
Because of the iterator, the **process()** function is called once for each selected event and fills the vertex *z*-position histogram

Let's run the analysis task:

```
bash run_skeleton.sh
```

```
OPTION="-b --configuration json://configuration_skeleton.json"
o2-analysistutorial-lf-strangeness-pbpb-skeleton ${OPTION} --aod-file @input_data.txt
```

Preparation: running an analysis task



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

Step 0: looping over cascades

```
struct strangeness_pbpb_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

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

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

For each event, loop over all cascades

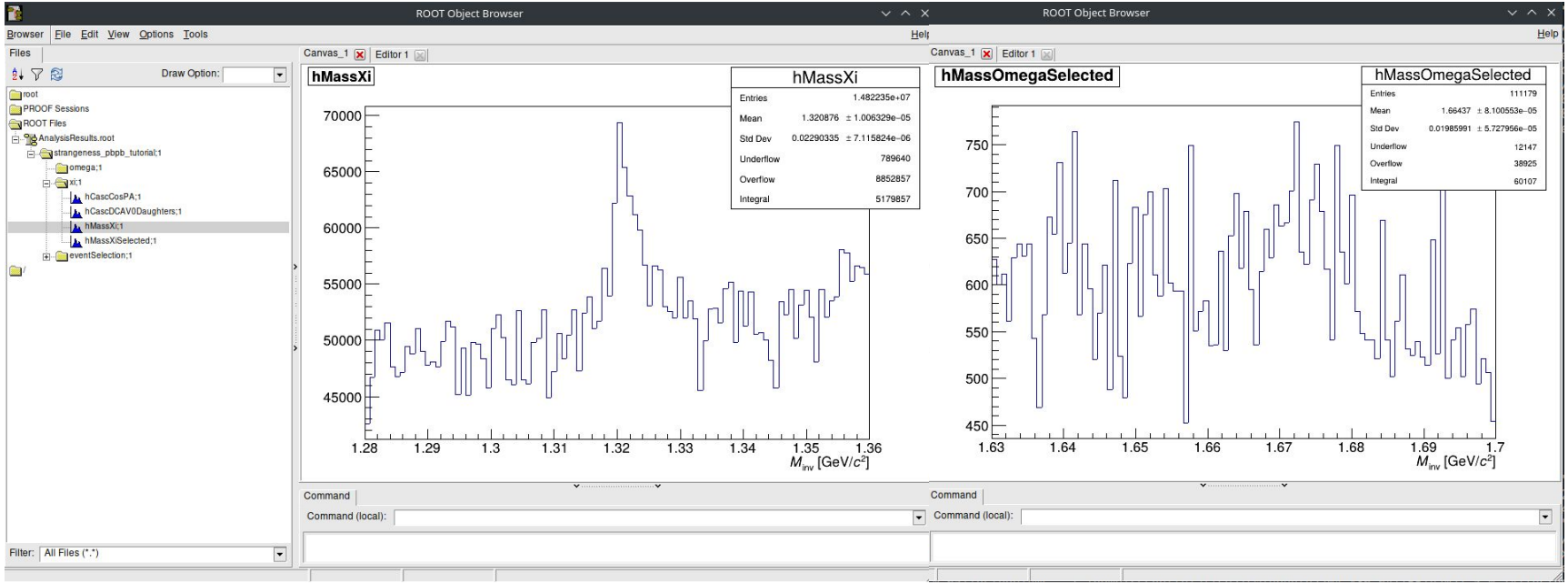
For each mass hypothesis, fill a histogram with the invariant mass

Require an extra task to enable the use of table with extended columns like **aod::CascCores**

```
OPTION="-b --configuration json://configuration_step0.json"

o2-analysis-lf-cascadespawner ${OPTION} | o2-analysistutorial-lf-strangeness-pbpb-step0 ${OPTION} --aod-file @input_data.txt
```

```
// extended table with expression columns that can be used as arguments of dynamic columns
DECLARE_SOA_EXTENDED_TABLE_USER(CascCores, StoredCascCores, "CascDATAEXT", //!
                                cascdataext::PxLambda, cascdataext::PyLambda, cascdataext::PzLambda, cascdataext::Pt, cascdataext::P, cascdataext::Eta,
                                cascdataext::Phi);
```

Step 0: looping over cascades



Invariant mass peak is already visible for Ξ with the derived data producer **pre-selections**, while no peak for Ω

Step 1: apply topological selections

```
// 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

```
// Cut on dynamic columns
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 (TMath::Abs(casc.mLambda() - pdgDB->Mass(3122)) > 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 (TMath::Abs(casc.mXi() - pdgDB->Mass(3312)) > cascadesetting_competingmassrej) { // competing mass rejection, only in case of Omega
  rOmega.fill(HIST("hMassOmegaSelected"), casc.mOmega());
```

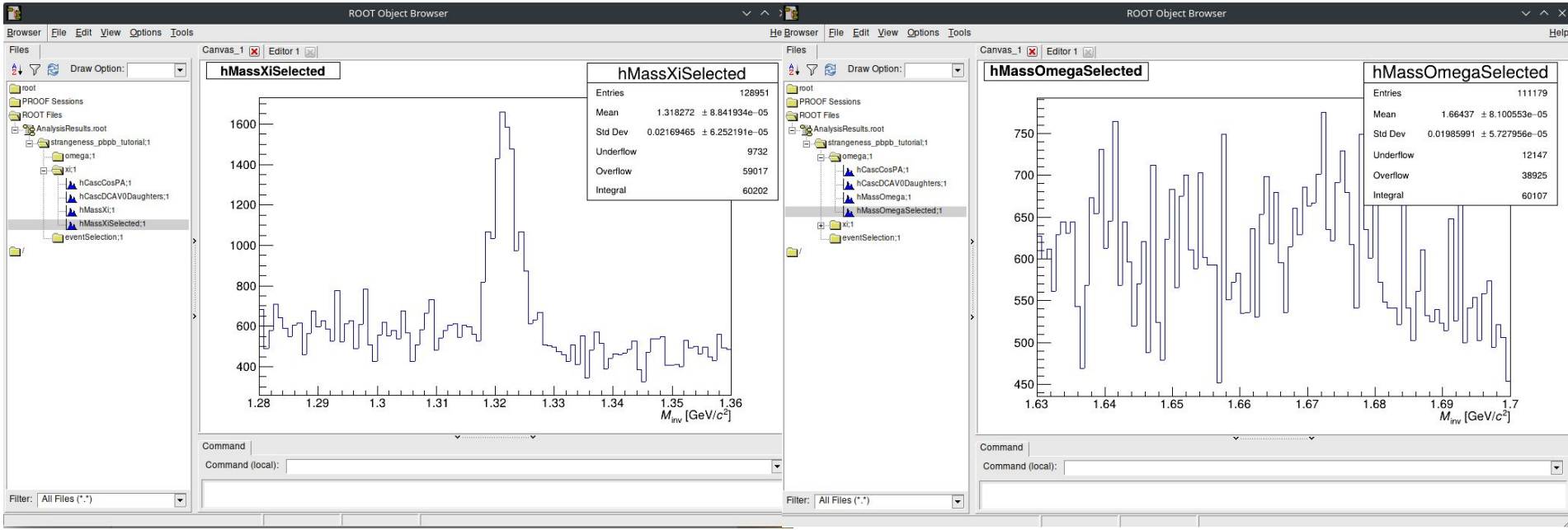
Fill invariant mass histograms <u>after</u> the topological selections

```
"strangeness_pbpb_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

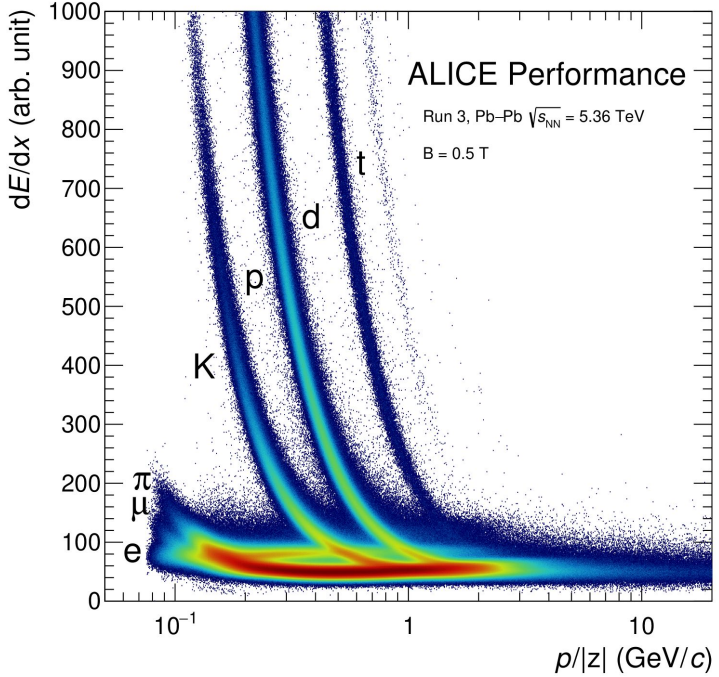
Mean energy loss is parametrized by Bethe-Bloch formula:

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

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

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

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



Step 2: apply TPC PID selections

```
#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

```
// Defining the type of the daughter tracks
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::DauTrackTPCPIDs** table

```
// 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

Step 2: apply accessing daughter tracks

- The cascade table (**aod::CascCores**) references a track table (**aod::DauTrackExtras**) that contains information about the TPC clusters, whether it has a hit in the ITS, TPC, TOF,...
  But it does NOT contain all useful information for analysis, such as PID information

  → if you try to check PID of daughter tracks with reference to that table the code will fail!

```
float nsigma_bach = casc.bachTrackExtra.tpcNSigmaPi();
float nsigma_pos = casc.posTrackExtra.tpcNSigmaPr();
float nsigma_neg = casc.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>;

float nsigma_bach = casc.bachTrackExtra_as<dauTracks>().tpcNSigmaPi();
float nsigma_pos = casc.posTrackExtra_as<dauTracks>().tpcNSigmaPr();
float nsigma_neg = casc.negTrackExtra_as<dauTracks>().tpcNSigmaPi();
```

Works!

Step 2: apply TPC PID selections

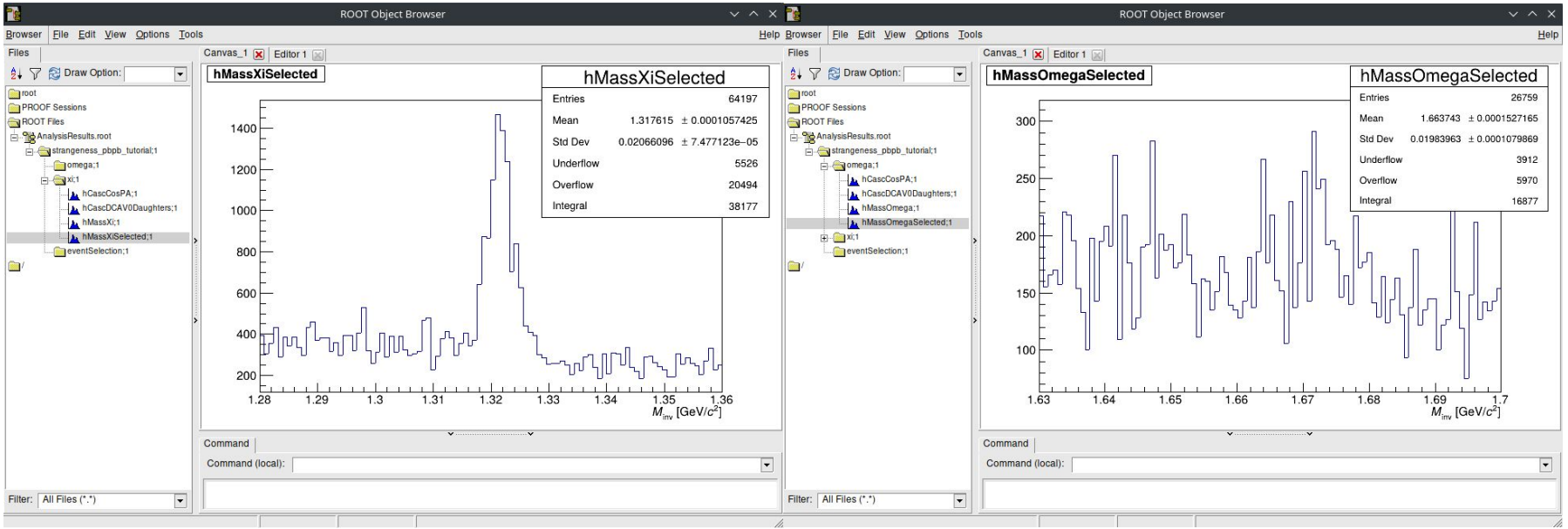
```
// PID selection
if (casc.sign() < 0) {
  if (TMath::Abs(posDaughterTrackCasc.tpcNSigmaPr()) > NSigmaTPCProton) {
    continue;
  }
  if (TMath::Abs(negDaughterTrackCasc.tpcNSigmaPi()) > NSigmaTPCPion) {
    continue;
  }
} else {
  if (TMath::Abs(negDaughterTrackCasc.tpcNSigmaPr()) > NSigmaTPCProton)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

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

```
if (TMath::Abs(bachDaughterTrackCasc.tpcNSigmaKa()) < NSigmaTPCKaon) { // Omega case
  if (TMath::Abs(casc.mXi() - pdgDB->Mass(3312)) > cascadesetting_competingmassrej) { // competing mass rejection, only in case of Omega
    rOmega.fill(HIST("hMassOmegaSelected"), casc.mOmega());
```

Apply TPC PID selections

Step 2: apply TPC PID selections



- Background is further reduced using TPC PID 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

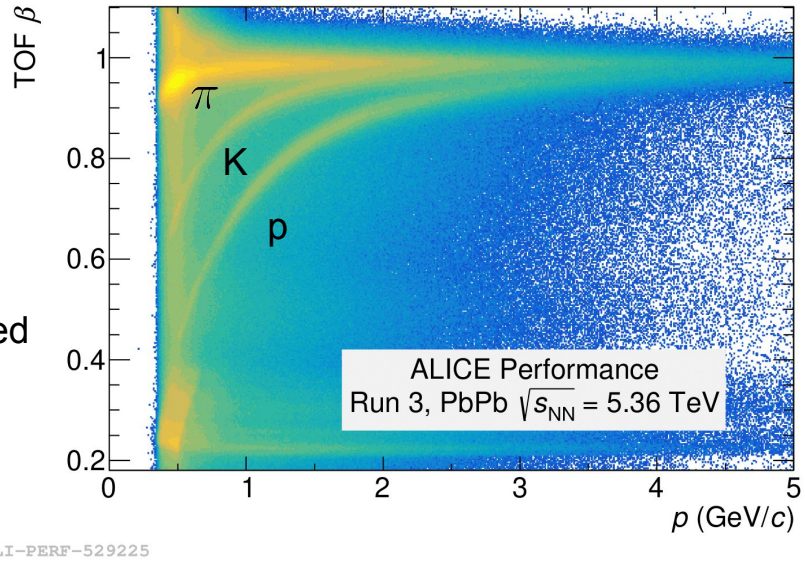
By measuring the particle velocity, the expected time-of-flight can be estimated:

$$t_{\mathrm{exp.},i} = L \frac{\sqrt{p^2 + m_i^2}}{cp}$$

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

$$n_\sigma = \frac{t_{\mathrm{meas.}} - t_{\mathrm{ev}} - t_{\mathrm{exp.},i}}{\sigma}$$

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



Step 3: apply TOF PID selections

```
// 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>> const& Cascades,
             dauTracks const&)
```

Subscribe to **aod::CascTOFNSigmas** table

```
// TOF PID check
bool xiPassTOFSelection = true;
bool omegaPassTOFSelection = true;
if (casc.sign() < 0) {
  if (posDaughterTrackCasc.hasTOF()) {
    if (TMath::Abs(casc.tofNSigmaXiLaPr()) > NSigmaTOFProton) {
      xiPassTOFSelection &= false;
    }
    if (TMath::Abs(casc.tofNSigmaOmLaPr()) > NSigmaTOFProton) {
      omegaPassTOFSelection &= false;
    }
  }
  if (negDaughterTrackCasc.hasTOF()) {
    if (TMath::Abs(casc.tofNSigmaXiLaPi()) > NSigmaTOFPion) {
      xiPassTOFSelection &= false;
    }
    if (TMath::Abs(casc.tofNSigmaOmLaPi()) > NSigmaTOFPion) {
      omegaPassTOFSelection &= false;
    }
  }
```

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

Step 3: apply TOF PID selections

```
// TOF PID check
bool xiPassTOFSelection = true;
bool omegaPassTOFSelection = true;
if (casc.sign() < 0) {
  if (posDaughterTrackCasc.hasTOF()) {
    if (TMath::Abs(casc.tofNSigmaXiLaPr()) > NSigmaTOFProton) {
      xiPassTOFSelection &= false;
    }
    if (TMath::Abs(casc.tofNSigmaOmLaPr()) > NSigmaTOFProton) {
      omegaPassTOFSelection &= false;
    }
  }
  if (negDaughterTrackCasc.hasTOF()) {
    if (TMath::Abs(casc.tofNSigmaXiLaPi()) > NSigmaTOFPion) {
      xiPassTOFSelection &= false;
    }
    if (TMath::Abs(casc.tofNSigmaOmLaPi()) > NSigmaTOFPion) {
      omegaPassTOFSelection &= false;
    }
  }
} else {
  if (posDaughterTrackCasc.hasTOF()) {
    if (TMath::Abs(casc.tofNSigmaXiLaPi()) > NSigmaTOFPion) {
      xiPassTOFSelection &= false;
    }
    if (TMath::Abs(casc.tofNSigmaOmLaPi()) > NSigmaTOFPion) {
      omegaPassTOFSelection &= false;
    }
  }
  if (negDaughterTrackCasc.hasTOF()) {
    if (TMath::Abs(casc.tofNSigmaXiLaPr()) > NSigmaTOFProton) {
      xiPassTOFSelection &= false;
    }
    if (TMath::Abs(casc.tofNSigmaOmLaPr()) > NSigmaTOFProton) {
      omegaPassTOFSelection &= false;
    }
  }
}

if (bachDaughterTrackCasc.hasTOF()) {
  if (TMath::Abs(casc.tofNSigmaXiPi()) > NSigmaTOFPion) {
    xiPassTOFSelection &= false;
  }
  if (TMath::Abs(casc.tofNSigmaOmKa()) > NSigmaTOFKaon) {
    omegaPassTOFSelection &= false;
  }
}
```

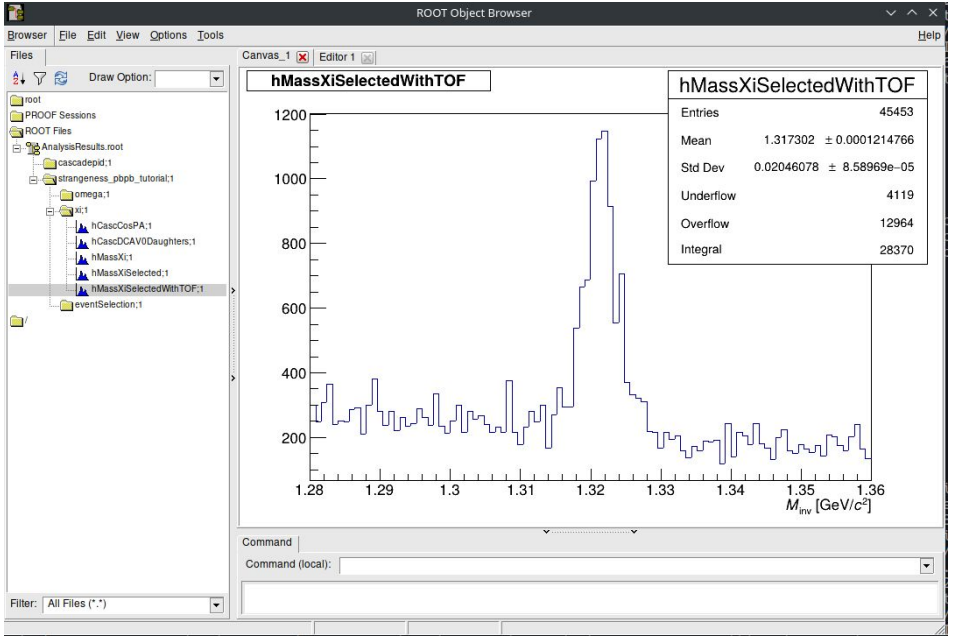
Apply TOF PID selections **if** the track has a hit in TOF

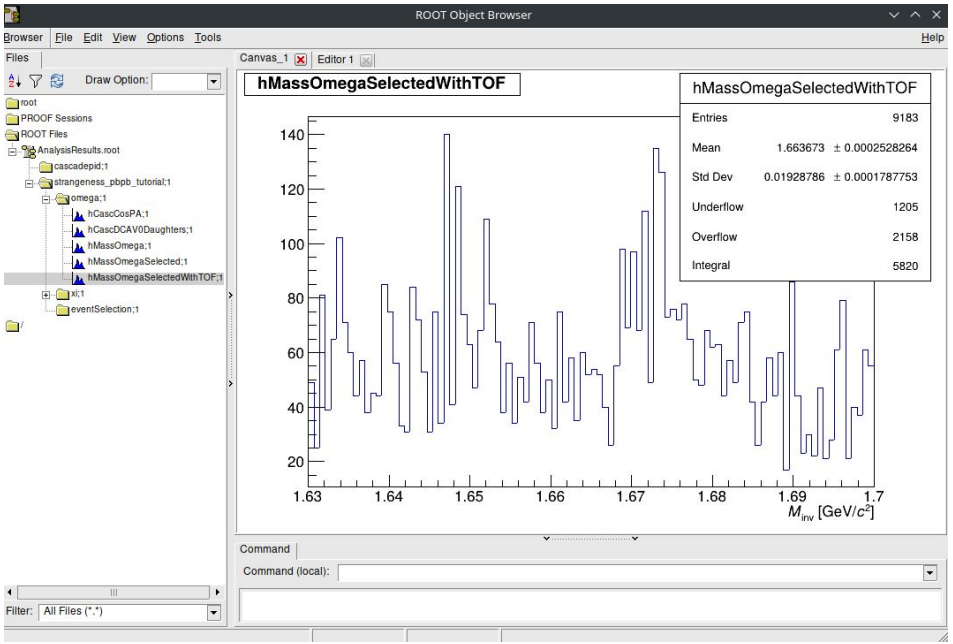
Fill invariant mass histograms <u>after</u> selections

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

```
if (TMath::Abs(bachDaughterTrackCasc.tpcNSigmaKa()) < NSigmaTPCKaon) { // Omega case
  if (TMath::Abs(casc.mXi() - pdgDB->Mass(3312)) > cascadesetting_competingmassrej) {
    rOmega.fill(HIST("hMassOmegaSelected"), casc.mOmega());
    if (omegaPassTOFSelection)
      rOmega.fill(HIST("hMassOmegaSelectedWithTOF"), casc.mOmega());
```

Step 3: apply TOF PID selections





- TOF selections help further suppressing combinatorial background, a peak starts to emerge close to PDG mass of the Ω

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

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

Step 4: access MC information of cascades

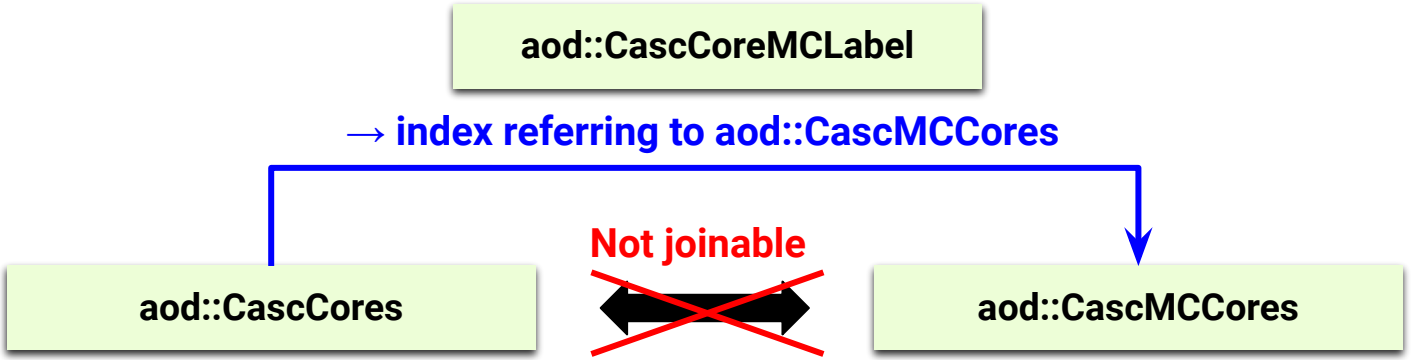
- From the previous steps, one can reconstruct the Xi and Omega with excellent purities, and evaluate their raw yields
- In order to converge towards corrected results, we would like now to move towards determining their efficiencies → **need to access MC information**
- Let's have a look at the number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

→ **reconstructed** info about cascades

→ **generated** info about cascades

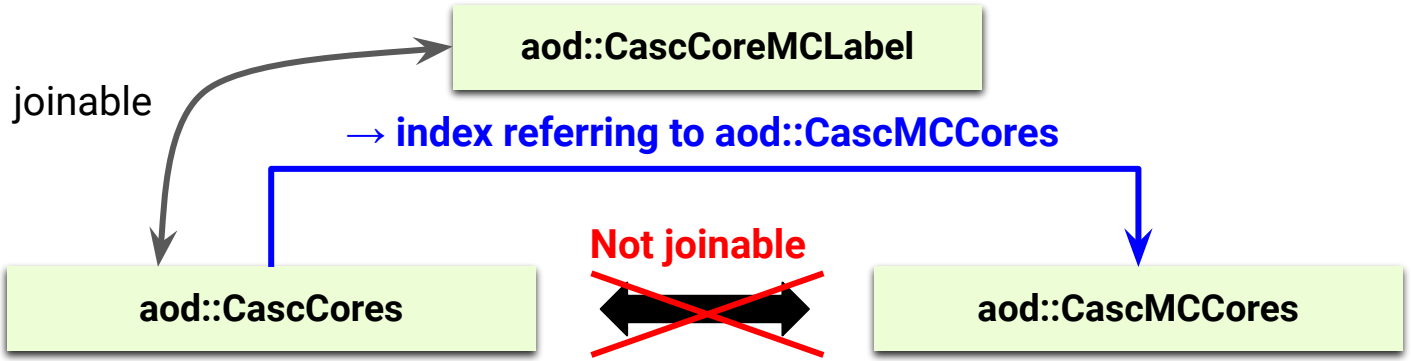
Step 4: access MC information of cascades

- 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



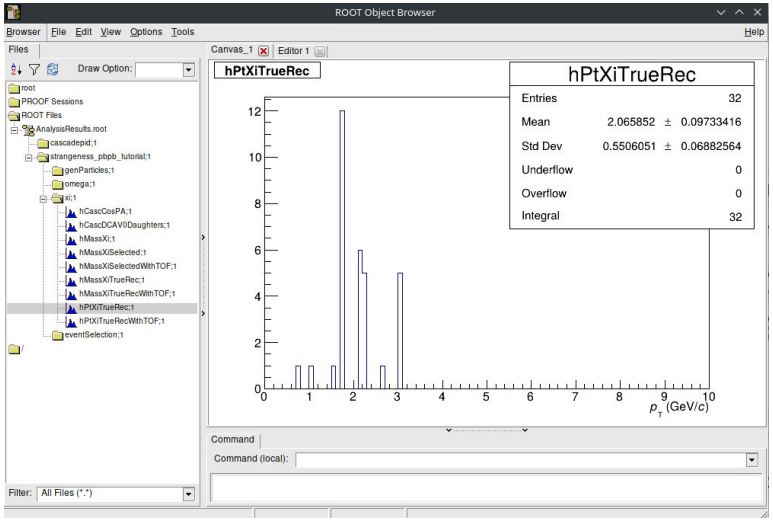
Step 4: access MC information of cascades

```
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*/)
```

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



- 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::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_pbpb_tutorial, processRecMC, "Process Run 3 mc, reconstructed", true);
PROCESS_SWITCH(strangeness_pbpb_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)
```

Step 4: loop over generated events

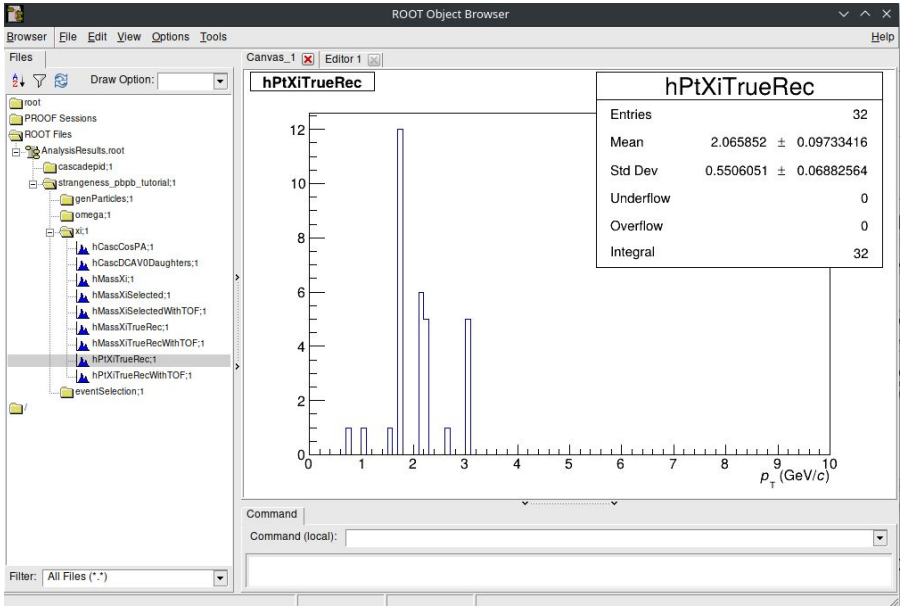
- The next step is to:
  - check that the MC collision have been reconstructed at least once using the **.size()**
  - Loop over all generated cascades
  - Fill a histogram with the generated transverse momentum for each mass hypothesis

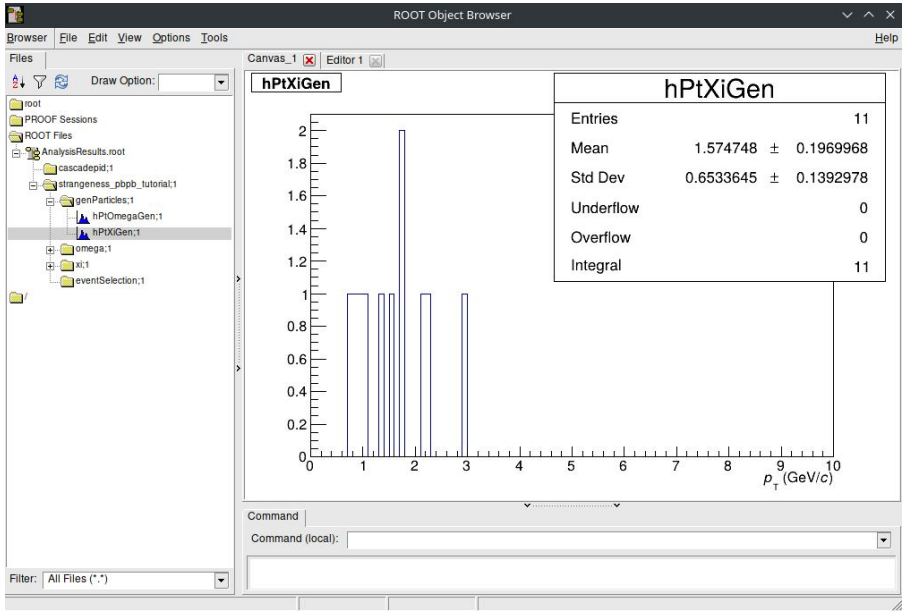
```
void processGenMC(soa::Filtered<aod::StraMCCollisions>::iterator const& mcCollision,
                  const soa::SmallGroups<soa::Join<aod::StraCollisions, o2::aod::StraEvSels, aod::StraCollLabels>>& collisions,
                  const soa::SmallGroups<soa::Join<aod::CascMCCores, aod::CascMCCollRefs>>& 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 (TMath::Abs(cascmc.pdgCode()) == 3312) {
      rGenParticles.fill(HIST("hPtXiGen"), cascmc.ptMC());
    }
    if (TMath::Abs(cascmc.pdgCode()) == 3334) {
      rGenParticles.fill(HIST("hPtOmegaGen"), cascmc.ptMC());
    }
  }
}
```

Loop over all generated cascades

Fill a histogram with the gen. pt

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 lecture you learned:

- produce strangeness derived tables
- 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

See you at the hands-on session of Wednesday!