An introduction to strangeness analysis and reconstruction in O2

The strangeness analysis workflow

Regular analysis flow

Converted AO2D.root files

`V0` table: {positive track ID, negative track ID, collision ID}
`cascade` table: {positive track ID, V0 ID, collision ID}

Build: Regenerate all V0 and cascade properties for analysis using cascade and V0 producer tasks [1]

Configure as filter

Cross-check/special

Converted AO2D.root files

`Fulltracks, TracksExtended` tables

Find: find V0s and cascade from scratch, generate analysis tables [2]

Configure cuts

`v0data, v0dataext, cascdata, cascdataext` tables: analysis-level info

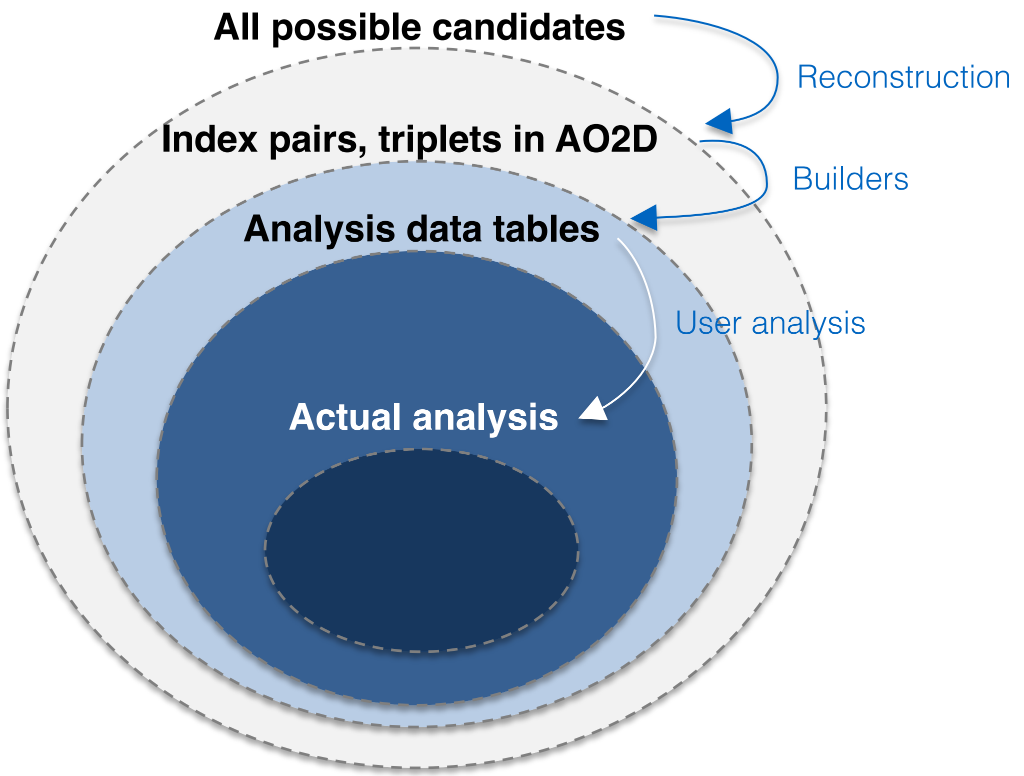
Analyse: Analyse the V0s and cascades via consumer tasks [3] (or any other analysis tasks interested!)

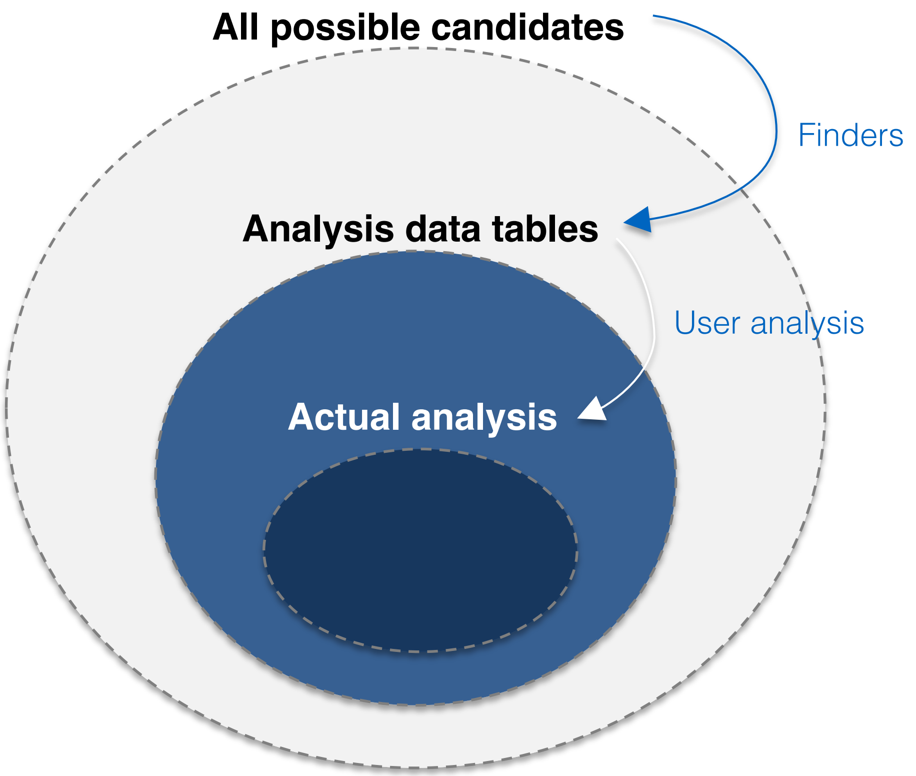
`[1] Analysis/Tasks/PWGLF/lambdakzerobuilder.cxx, cascadebuilder.cxx`

`[2] Analysis/Tasks/PWGLF/lambdakzerofinder.cxx, cascadefinder.cxx`

`[3] Analysis/Tasks/PWGLF/lambdakzeroanalysis.cxx, cascadeanalysis.cxx`

Candidate combinatorics





- Beware: regular analysis workflow does not allow for looser selections than stored indices!
- Also the case in Run 2, involves base choices
  (We will make sure they're loose enough :-))

- Replaying the finders: very heavy operation, only meant for cross-checks / benchmark for now
- Remains an option for generating derived data if strictly needed, careful use

Table content: V0s

- Producer: makes analysis trivial task
- Consumer (typical): simple loop, subscribe only to `aod::V0DataExt`

```cpp
struct lambdakzeroanalysis {
  HistogramRegistry registry{
    "registry",
    {
      {"h3dMassK0Short", "h3dMassK0Short", {HistType::kTH3F, {{20, 0.0f, 100.0f}, {200, 0.0f, 10.0f}, {200, 0.450f, 0.550f}}}},
      {"h3dMassLambda", "h3dMassLambda", {HistType::kTH3F, {{20, 0.0f, 100.0f}, {200, 0.0f, 10.0f}, {200, 1.015f, 1.215f}}}},
      {"h3dMassAntiLambda", "h3dMassAntiLambda", {HistType::kTH3F, {{20, 0.0f, 100.0f}, {200, 0.0f, 10.0f}, {200, 1.015f, 1.215f}}}},
    },
  };

  //Selection criteria
  Configurable<double> v0cospa{"v0cospa", 0.995, "V0 CosPA"}; //double -> N.B. dcos(x)/dx = 0 at x=0)
  Configurable<float> dcav0dau{"dcav0dau", 1.0, "DCA V0 Daughters"};
  Configurable<float> dcanegtopv{"dcanegtopv", .1, "DCA Neg To PV"};
  Configurable<float> dcapostopv{"dcapostopv", .1, "DCA Pos To PV"};
  Configurable<float> v0radius{"v0radius", 5.0, "v0radius"};
  Configurable<float> rapidity{"rapidity", 0.5, "rapidity"};
  Configurable<int> saveDcaHist{"saveDcaHist", 0, "saveDcaHist"};

  Filter preFilterV0 = nabs(aod::v0data::dcapostopv) > dcapostopv&& nabs(aod::v0data::dcanegtopv) > dcanegtopv&& aod::v0data::dcaV0daughters < dcav0dau;

  void process(soa::Join<aod::Collisions, aod::EvSels, aod::Cents>::iterator const& collision, soa::Filtered<aod::V0Datas> const& fullV0s)
  {
    if (!collision.alias()[kINT7]) {
      return;
    }
    if (!collision.sel7()) {
      return;
    }

    for (auto& v0 : fullV0s) {
      //FIXME: could not find out how to filter cosPA and radius variables (dynamic columns)
      if (v0.v0radius() > v0radius && v0.v0cosPA(collision.posX(), collision.posY(), collision.posZ()) > v0cospa) {
        if (TMath::Abs(v0.yLambda()) < rapidity) {
          registry.fill(HIST("h3dMassLambda"), collision.centV0M(), v0.pt(), v0.mLambda());
          registry.fill(HIST("h3dMassAntiLambda"), collision.centV0M(), v0.pt(), v0.mAntiLambda());
        }
        if (TMath::Abs(v0.yK0Short()) < rapidity) {
          registry.fill(HIST("h3dMassK0Short"), collision.centV0M(), v0.pt(), v0.mK0Short());
        }
      }
    }
  }
};
```

Simple loop, all topological variables available

StrangenessTables.h:

converted:
```
v0::PosTrackID
v0::NegTrackID
v0::CollisionID
```

Recalculated:
```
v0data::PxPos
v0data::PyPos
v0data::PzPos
v0data::PxNeg
v0data::PyNeg
v0data::PzNeg
v0data::X
v0data::Y
v0data::Z
v0data::DCAV0Daughters
v0data::DCAPosToPV
v0data::DCANegToPV
```

On demand:
```
v0data::Pt(dynamic)
v0data::V0Radius (dynamic)
v0data::V0CosPA (dynamic)
v0data::DCAV0ToPV (dynamic)
v0data::MLambda (dynamic)
v0data::MAntiLambda (dynamic)
v0data::MK0Short (dynamic)
v0data::YK0Short (dynamic)
v0data::YLambda (dynamic)
v0data::Eta (dynamic)
```

```
v0dataext::Px (expression)
v0dataext::Py (expression)
v0dataext::Pz (expression)
```

Table content: Cascades

StrangenessTables.h:

converted

```
cascade::V0ID
cascade::BachelorID
cascade::CollisionID
```

Recalculated

```
cascdata::Charge
cascdata::PxPos
cascdata::PyPos
cascdata::PzPos
cascdata::PxNeg
cascdata::PyNeg
cascdata::PzNeg
cascdata::PxBach
cascdata::PyBach
cascdata::PzBach
cascdata::X
cascdata::Y
cascdata::Z
cascdata::Xlambda
cascdata::Ylambda
cascdata::Zlambda
cascdata::DCAV0Daughters
cascdata::DCACascDaughters
cascdata::DCAPosToPV
cascdata::DCANegToPV
cascdata::DCABachToPV
```

```
cascdata::Pt (dynamic)
cascdata::V0Radius (dynamic)
cascdata::CascRadius (dynamic)
cascdata::V0CosPA (dynamic)
cascdata::CascCosPA (dynamic)
cascdata::DCAV0ToPV (dynamic)
cascdata::DCACascToPV (dynamic)
cascdata::Mlambda (dynamic)
cascdata::Mxi (dynamic)
cascdata::Momega (dynamic)
cascdata::Yxi (dynamic)
cascdata::Yomega (dynamic)
cascdata::Eta (dynamic)
cascdataext::PxLambda (expression)
cascdataext::PyLambda (expression)
cascdataext::PzLambda (expression)
cascdataext::Px (expression)
cascdataext::Py (expression)
cascdataext::Pz (expression)
```

-> same underlying principle

Analysis example: Cascades

(after production of cascdata, cascdataext)

```
void process(soa::Join<aod::Collisions, aod::EvSels, aod::Cents>::iterator const& collision, soa::Filtered<aod::CascDataExt> const& Cascades)
{
  if (!collision.alias()[kINT7]) {
    return;
  }
  if (!collision.sel7()) {
    return;
  }
  for (auto& casc : Cascades) {
    //FIXME: dynamic columns cannot be filtered on?
    if (casc.v0radius() > v0radius &&
        casc.cascradius() > cascradius &&
        casc.v0cosPA(collision.posX(), collision.posY(), collision.posZ()) > v0cospa &&
        casc.casccosPA(collision.posX(), collision.posY(), collision.posZ()) > casccospa &&
        casc.dcav0topv(collision.posX(), collision.posY(), collision.posZ()) > dcav0topv) {
      if (casc.charge() < 0) { //FIXME: could be done better...
        if (TMath::Abs(casc.yXi()) < 0.5) {
          h3dMassXiMinus->Fill(collision.centV0M(), casc.pt(), casc.mXi());
        }
        if (TMath::Abs(casc.yOmega()) < 0.5) {
          h3dMassOmegaMinus->Fill(collision.centV0M(), casc.pt(), casc.mOmega());
        }
      } else {
        if (TMath::Abs(casc.yXi()) < 0.5) {
          h3dMassXiPlus->Fill(collision.centV0M(), casc.pt(), casc.mXi());
        }
        if (TMath::Abs(casc.yOmega()) < 0.5) {
          h3dMassOmegaPlus->Fill(collision.centV0M(), casc.pt(), casc.mOmega());
        }
      }
    }
  }
}
};
```

Non-dynamic columns can be filtered on

Note: some vars need PV position

Task only fills analysis histos

V0 finding from scratch

`Analysis/Tasks/PWGLF/lambadkzerofinder.cxx`

- A crucial task in reconstruction, but *too heavy for analysis*
- At this point: explore + test O2 performance (benchmark!)

Task: **lambdakzeroprefilter**
Produces<aod::v0goodpostracks> v0GoodPosTracks;
Produces<aod::v0goodnegtracks> v0GoodNegTracks;

Single track quality:
- kTPCrefit
- $N_{\text{crossed rows}} > 70$
- DCA to PV > 0.1cm

Task: **lambdakzerofinder**
Produces<aod::v0data> v0data;

V0 selection:
- DCA V0 Daughters
- V0 Cosine PA
- V0 radius

Task: **lambdakzerofinderQA**
Generates basic QA plots and 3D mass histos

- Same structure as the V0s built with the builder -> interchangeability!

UNICAMP

ALICE

Cascade finding from scratch

`Analysis/Tasks/PWGLF/cascadefinder.cxx`

Task: **cascadeprefilter**
Produces<aod::cascgoodlambdas> cascGoodLambdas;
Produces<aod::cascgoodantilambdas> cascGoodAntiLambdas;
Produces<aod::cascgoodpostracks> cascGoodPosTracks;
Produces<aod::cascgoodnegtracks> cascGoodNegTracks;

Task: **cascadefinder**
Produces<aod::cascdata> cascdata;

Task: **cascadefinderQA**
Generates basic QA plots and 3D mass histos

- Maximize pre-filtering for performance
- Conceptually similar to V0 finding: previous slide applies!
- Same output structure as the builder -> interchangeability!

V0 quality:
- DCA V0 Dau
- DCA Neg, Pos to PV
- V0 CosPA
- DCA V0 to PV
- (Anti)lambda mass

Single track quality:
- kTPCrefit
- $N_{\text{crossed rows}} > 70$
- DCA to PV > 0.1cm

Cascade selection:
- DCA casc. daughters
- Casc. Cosine PA
- Casc. radius

Hands-on part 1

A basic V0 analysis

Setup: the data and the files

- Now we will use a converted sample of LHC15o data and get some V0 invariant mass distributions
- Download the input file:

```
alien_cp alien:///alice/data/2015/LHC15o/000244917/pass5_lowIR/PWGZZ/Run3_Conversion/148_20210304-0829_child_1/0017/AO2D.root AO2D_test.root
```

- Alternatively, use dropbox: link

- We will start from an analysis skeleton for V0s. Download the task from:
  - https://www.dropbox.com/s/65f5tlwh3816ot0/LFStrangeTut1.cxx?dl=0

UNICAMP

ALICE

The basic strangeness analysis skeleton

```
struct lfstrangetut1 {
  HistogramRegistry registry{
    "registry",
    {
      {"hMassK0Short", "hMassK0Short", {HistType::kTH1F, {{100, 0.450f, 0.550f}}}}
    },
  };

  //Selection criteria
  Configurable<double> v0cospa{"v0cospa", 0.995, "V0 CosPA"}; //double -> N.B. dcos(x)/dx = 0 at x=0)
  Configurable<float> dcav0dau{"dcav0dau", 1.0, "DCA V0 Daughters"};
  Configurable<float> dcanegtopv{"dcanegtopv", .1, "DCA Neg To PV"};
  Configurable<float> dcapostopv{"dcapostopv", .1, "DCA Pos To PV"};
  Configurable<float> v0radius{"v0radius", 5.0, "v0radius"};
  Configurable<float> rapidity{"rapidity", 0.5, "rapidity"};
  Configurable<int> saveDcaHist{"saveDcaHist", 0, "saveDcaHist"};

  Filter preFilterV0 = nabs(aod::v0data::dcapostopv) > dcapostopv&& nabs(aod::v0data::dcanegtopv) > dcanegtopv&& aod::v0data::dcaV0daughters < dcav0dau;

  void process(soa::Join<aod::Collisions, aod::EvSels, aod::Cents>::iterator const& collision, soa::Filtered<aod::V0Datas> const& fullV0s)
  {
    if (!collision.alias()[kINT7]) {
      return;
    }
    if (!collision.sel7()) {
      return;
    }
    for (auto& v0 : fullV0s) {
      if (v0.v0radius() > v0radius && v0.v0cosPA(collision.posX(), collision.posY(), collision.posZ()) > v0cospa) {
        if (TMath::Abs(v0.yK0Short()) < rapidity) {
          registry.fill(HIST("hMassK0Short"), v0.mK0Short());
        }
      }
    }
  }
};
```

Test histogram

Selection criteria

Candidate loop

The workflow: let's try it out, builder

- The standard analysis requires several components to be run correctly. A valid call is:

```
o2-analysis-timestamp --aod-file AO2D_test.root -b |
o2-analysis-multiplicity-table -b |
o2-analysis-centrality-table -b |
o2-analysis-event-selection -b |
o2-analysis-trackextension -b |
o2-analysis-weak-decay-indices -b |
o2-analysis-lambdakzerobuilder --d_bz 5 -b |
o2-analysis-LFStrangeTut1 -b
```

Process track indices

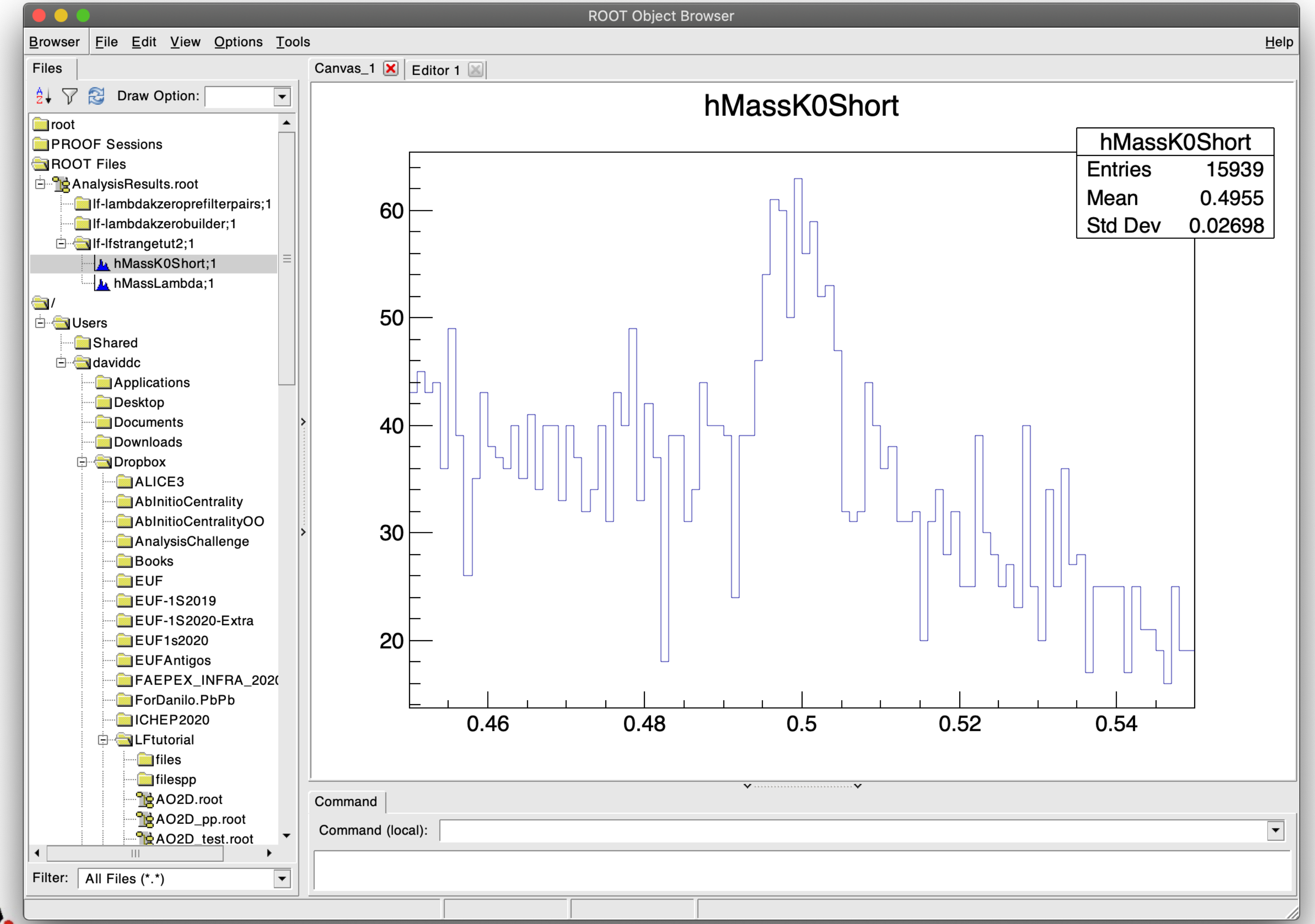
Build V0 information, do not forget magnetic field!

Your analysis!

UNICAMP

ALICE

Step 1: The output



Still not much data… Let's loop over further AO2D files!

…and we will then slowly build up…

Step 2: Downloading more files

- Use a bash script to download new AO2D files to run over! Here is an example: multidownload.sh

In your working directory, simply do:

```
[0] mkdir files
[1] source multidownload.sh
```

And you will download 10 files!
(No token? No certificate? In panic? Okay, check out this dropbox link and download from there)

Then, call:

```
[2] ls files/AO2D_*.root > l.txt
```

This will give you a list with all files you just downloaded. You can run on that!

But before that, let's add a simple Λ invariant mass histogram…

UNICAMP

ALICE

Step 2: Adding a Λ invariant mass histogram

```cpp
  HistogramRegistry registry{
    "registry",
    {
      {"hMassK0Short", "hMassK0Short", {HistType::kTH1F, {{100, 0.450f, 0.550f}}}},
      {"hMassLambda", "hMassLambda", {HistType::kTH1F, {{200, 1.016f, 1.216f}}}},
    },
  };
```

```cpp
        }
        if (TMath::Abs(v0.yLambda()) < rapidity && v0.pt() > minpt ) {
          registry.fill(HIST("hMassLambda"), v0.mLambda());
        }
```

This is a simple modification: just follow the lead of the hMassK0Short histogram!

Step 2: Adapting the workflow: adding selection criteria

- Now we can run over several files! We can use:

```
o2-analysis-timestamp --aod-file @l.txt-b --shm-segment-size 16000000000 |
o2-analysis-multiplicity-table -b |
o2-analysis-centrality-table -b |
o2-analysis-event-selection -b |
o2-analysis-trackextension -b |
o2-analysis-weak-decay-indices -b |
o2-analysis-lambdakzerobuilder --d_bz 5 -b |
o2-analysis-LFStrangeTut1 -b --v0cospa 0.999
```

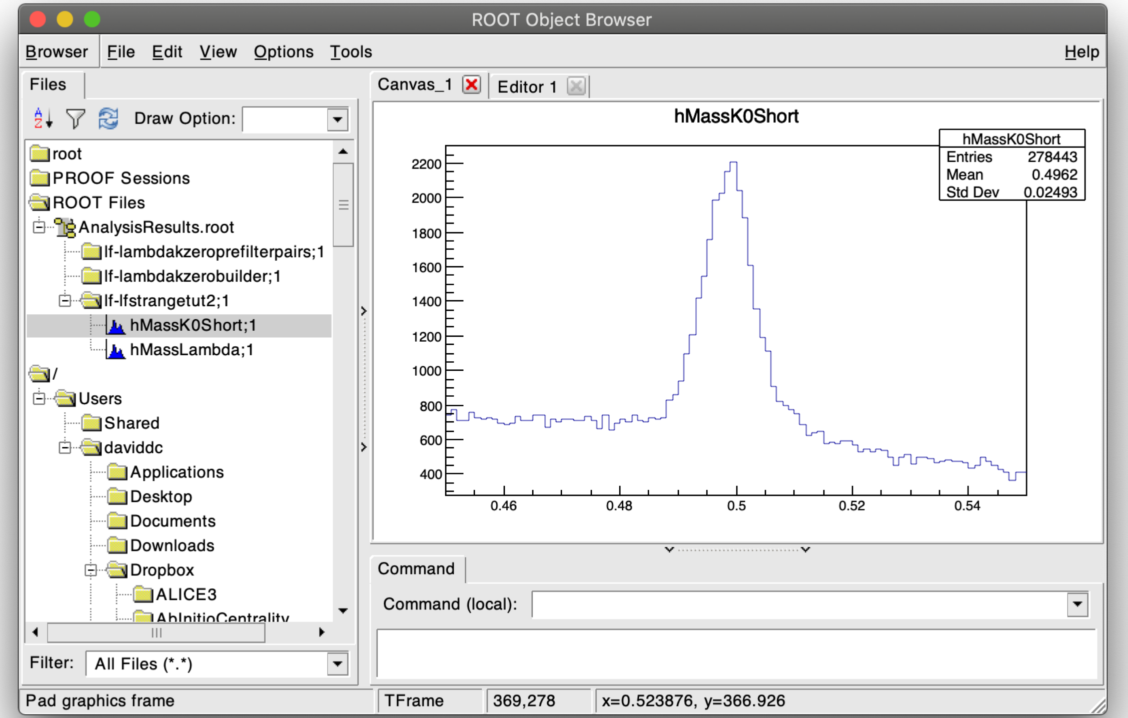
Ensure enough memory

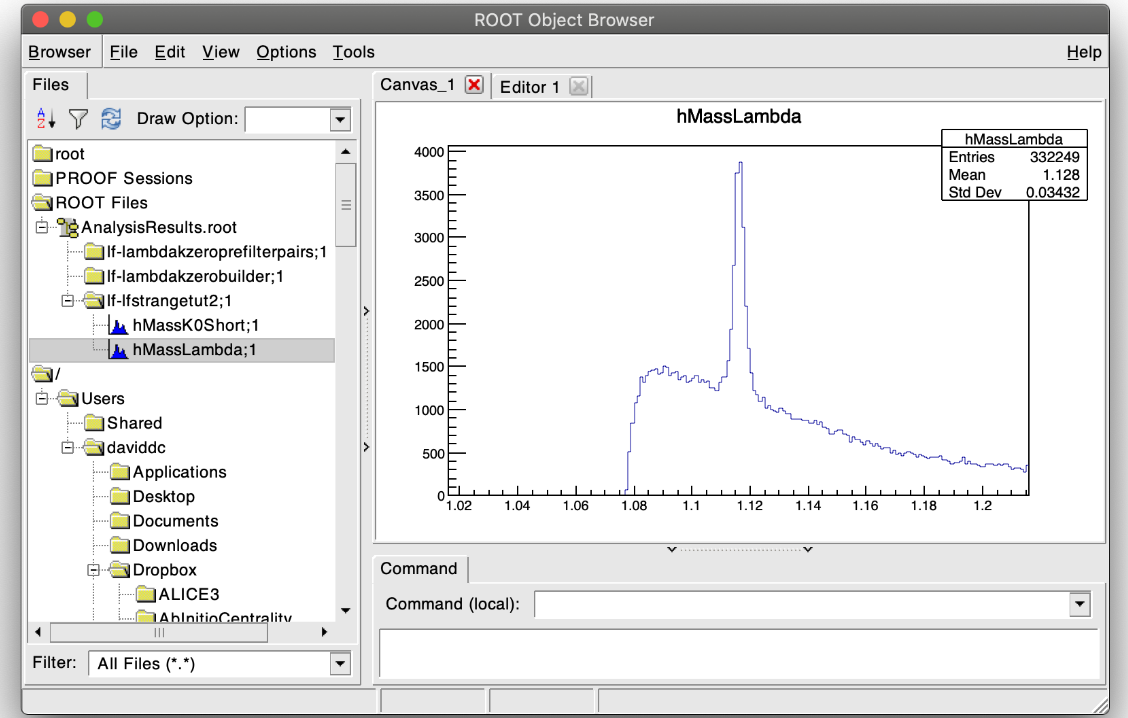
Extra selection on V0
cosine of pointing angle

UNICAMP

ALICE

Step 2: The new output: Λ invariant mass





Much better!

- Still needed: TPC dE/dx!
- Let's now combine information
- from the previous tutorial!

The code after this step: https://www.dropbox.com/s/93qvpr16eeauq0e/LFStrangeTut2.cxx?dl=0

Step 3: Adding a TPC dE/dx selection on daughter tracks

- Add PID response header

```
#include "AnalysisDataModel/PID/PIDResponse.h"      // Here PID response tables are defined
```

- Add subscription to tracks with PID information

```
using DaughterTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::pidRespTPC>;
```

```
  void process(soa::Join<aod::Collisions, aod::EvSels, aod::Cents>::iterator const& collision, soa::Filtered<aod::V0Datas> const& fullV0s, DaughterTracks const& tracks)
```

- Use selection

```
if (TMath::Abs(v0.yLambda()) < rapidity && v0.pt() > minpt ) {
  //Do proper selection on TPC dE/dx, please
  registry.fill(HIST("hMassLambda"), v0.mLambda());
  if(fabs(v0.posTrack_as<DaughterTracks>().tpcNSigmaPr()) < 3.0 &&
     fabs(v0.negTrack_as<DaughterTracks>().tpcNSigmaPi()) < 3.0){
    registry.fill(HIST("hMassLambdaWithdEdx"), v0.mLambda());
  }
}
```

Note: _as<>

Step 3: The _as<> functionality

```
if (TMath::Abs(v0.yLambda()) < rapidity && v0.pt() > minpt ) {
  //Do proper selection on TPC dE/dx, please
  registry.fill(HIST("hMassLambda"), v0.mLambda());
  if(fabs(v0.posTrack_as<DaughterTracks>().tpcNSigmaPr()) < 3.0 &&
     fabs(v0.negTrack_as<DaughterTracks>().tpcNSigmaPi()) < 3.0){
    registry.fill(HIST("hMassLambdaWithdEdx"), v0.mLambda());
  }
}
```

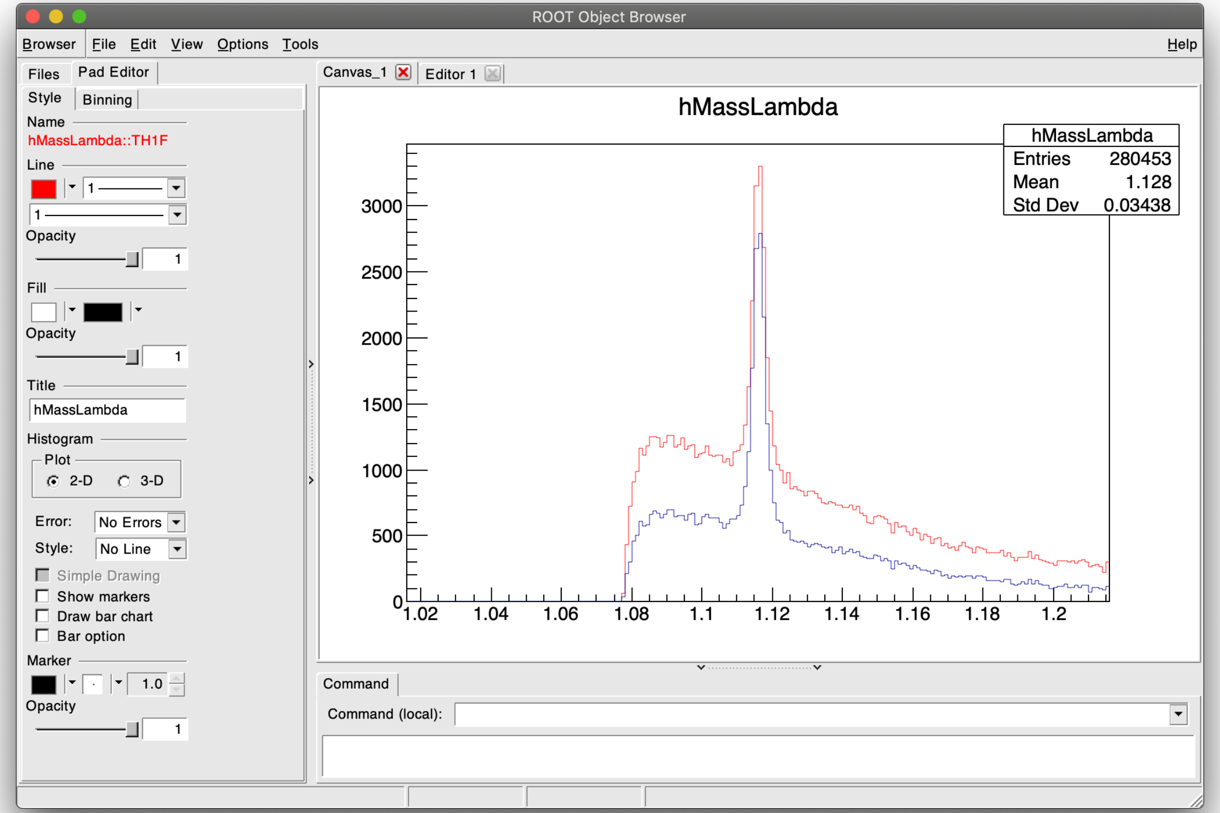
OK!

```
if (TMath::Abs(v0.yLambda()) < rapidity && v0.pt() > minpt ) {
  //Do proper selection on TPC dE/dx, please
  registry.fill(HIST("hMassLambda"), v0.mLambda());
  if(fabs(v0.posTrack.tpcNSigmaPr()) < 3.0 &&
     fabs(v0.negTrack.tpcNSigmaPi()) < 3.0){
    registry.fill(HIST("hMassLambdaWithdEdx"), v0.mLambda());
  }
}
```

Crashes!

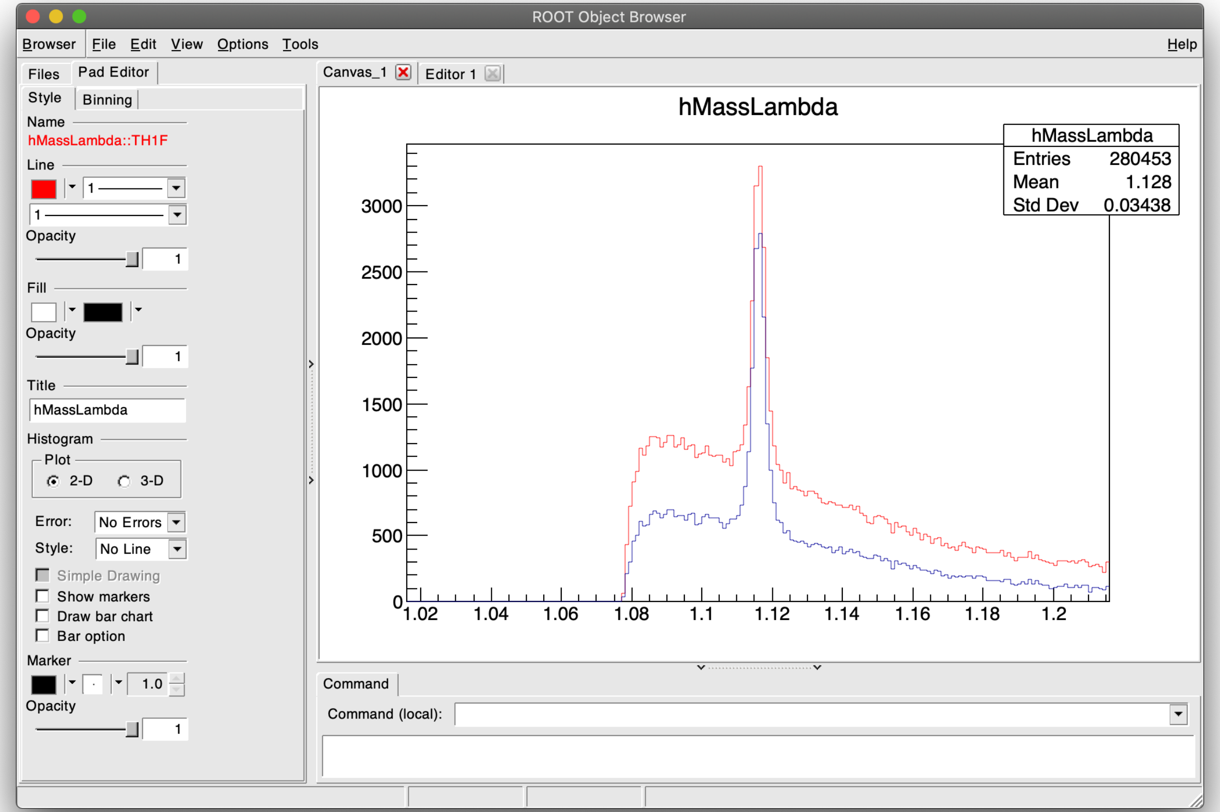
- The v0 table references a track table that does not contain PID.
- Trying to check PID information with a reference to that table will fail! It's not a simple index
- You need to de-reference the track via a _as<> call: will look at the same position in the more complete table!
- Valid in any similar situation, very useful!

Step 3: Λ invariant mass with extra dE/dx selection

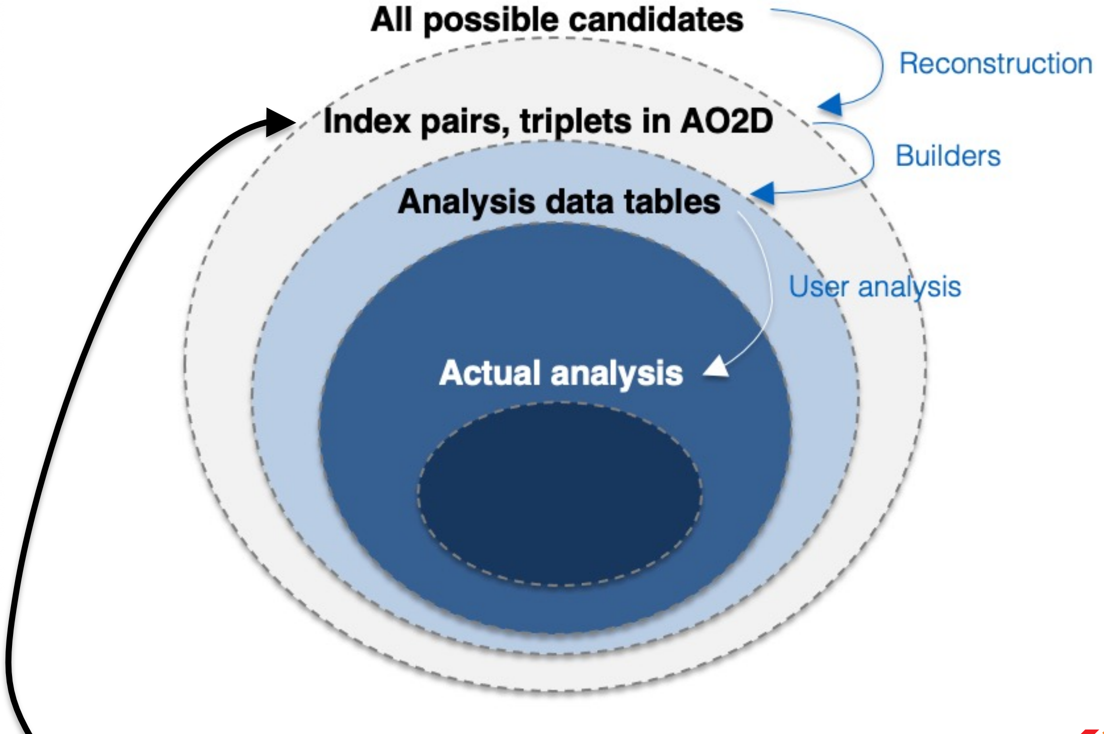


- Okay, this worked! But… is the gain so small with dE/dx?

Step 3: Λ invariant mass with extra dE/dx selection



- Okay, this worked! But… is the gain so small with dE/dx?
  - Actually, no! Remember this:



There's a dE/dx pre-selection applied here!

The code after this step:
https://www.dropbox.com/s/vib3h1mngn8v9zf/LFStrangeTut3.cxx?dl=0

Hands-on part 2 (‘bonus’)

Creating custom tables as desired

Step 4: An important feature: creating custom tables

- Tables are the cornerstone data container in O2.
  …but they are not only those we provided!
- You can have your own, fully customized, skimmed / derived table!
- This is useful for data reduction and optimizing your algorithm!
- Here, we'll walk you through a very basic example:

Let's say you're now happy with your dE/dx-selected Lambda candidates and you want to save their invariant masses for posterior analysis in a new table. It's less data! So let's do it…

Step 4: Declaring a custom table

- You can declare a custom table even inside your own task cxx file! See:

```cpp
namespace o2::aod
{
namespace skimmeddata
{
DECLARE_SOA_INDEX_COLUMN(Collision, collision);
DECLARE_SOA_COLUMN(LambdaMass, lambdaMass, float);
} // namespace skimmeddata
DECLARE_SOA_TABLE(SkimmedData, "AOD", "SKIMMEDDATA", o2::soa::Index<>, skimmeddata::CollisionId, skimmeddata::LambdaMass);
} // namespace o2::aod

struct lfstrangetut4 {
  Produces<aod::SkimmedData> skim;
  HistogramRegistry registry{
    "registry",
```

Collision index: allows for collision grouping

Your favorite information

Declaring the table

Statement to the framework: this task produces this table! Important for automatic task ordering

Step 4: Filling a custom table

```cpp
for (auto& v0 : fullV0s) {
  registry.fill(HIST("hSkimmedData"), 0.5);
  if (v0.v0radius() > v0radius && v0.v0cosPA(collision.posX(), collision.posY(), collision.posZ()) > v0cospa) {
    if (TMath::Abs(v0.yK0Short()) < rapidity && v0.pt() > minpt ) {
      registry.fill(HIST("hMassK0Short"), v0.mK0Short());
    }
    if (TMath::Abs(v0.yLambda()) < rapidity && v0.pt() > minpt ) {
      //Do proper selection on TPC dE/dx, please
      registry.fill(HIST("hMassLambda"), v0.mLambda());
      if(fabs(v0.posTrack_as<DaughterTracks>().tpcNSigmaPr()) < 3.0 &&
          fabs(v0.negTrack_as<DaughterTracks>().tpcNSigmaPi()) < 3.0){
        registry.fill(HIST("hMassLambdaWithdEdx"), v0.mLambda());
        if(fabs(v0.mLambda()-1.116)<0.075){
          registry.fill(HIST("hSkimmedData"), 1.5); //All V0s considered in loop
          skim(v0.globalIndex(), v0.mLambda());
        }
      }
    }
  }
}
```

Bookkeeping of skim fraction (for info!)

Fill the skimmed information table: provide data elements in order!

Step 4: Declaring a sub-task that uses this information

```cpp
struct lfstrangetut4a {
  HistogramRegistry registry{
    "registry",
    {
      {"hMassLambda", "hMassLambda", {HistType::kTH1F, {{200, 1.016f, 1.216f}}}}
    },
  };

  void process(aod::SkimmedData const& skim)
  {
    for (auto& object : skim) {
      registry.fill(HIST("hMassLambda"), object.lambdaMass());
    }
  }
};
```

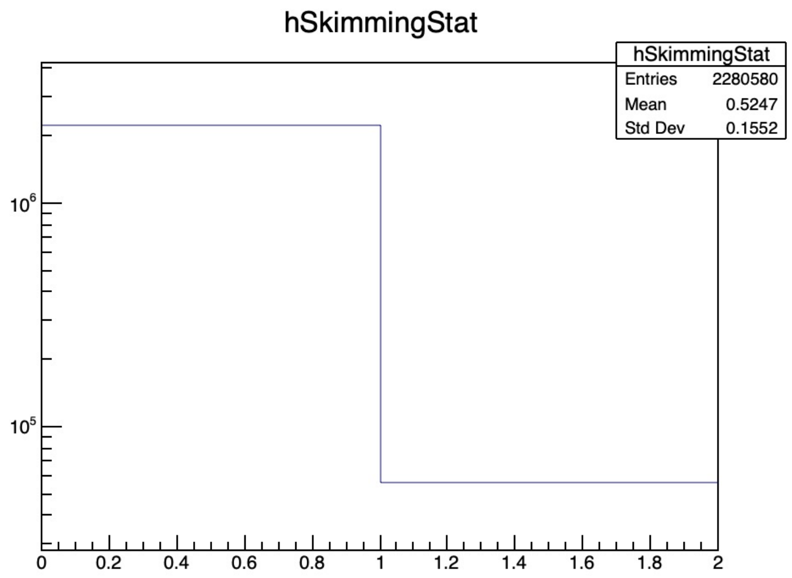
Simple and intuitive looping

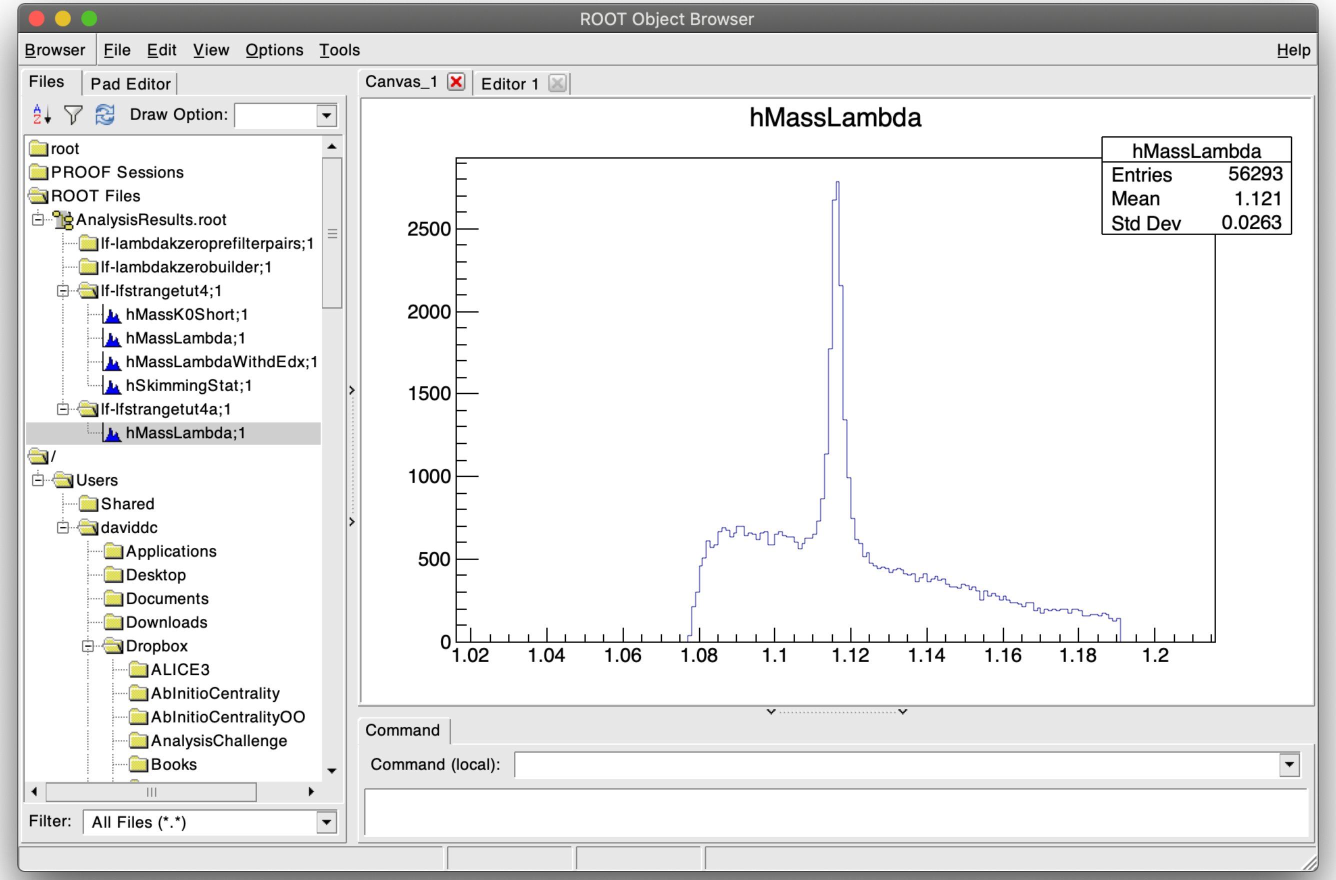
```cpp
WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
{
  return WorkflowSpec{
    adaptAnalysisTask<lfstrangetut4>(cfgc, TaskName{"lf-lfstrangetut4"}),
    adaptAnalysisTask<lfstrangetut4a>(cfgc, TaskName{"lf-lfstrangetut4a"})
  };
}
```

UNICAMP

ALICE

Step 4: The skimming example: results





2224K / 56K = **40x reduction!**

The same analysis result!

The code after this step:
https://www.dropbox.com/s/mluv518alerks76/LFStrangeTut4.cxx?dl=0

UNICAMP

ALICE

Summary

- Here, you saw:
  - an example V0 analysis
  - how to apply topological selections
  - how to apply PID information
  - how to construct a custom table with selected information for posterior analysis
- It goes way beyond: cascades, nuclei, etc!

Thank you!