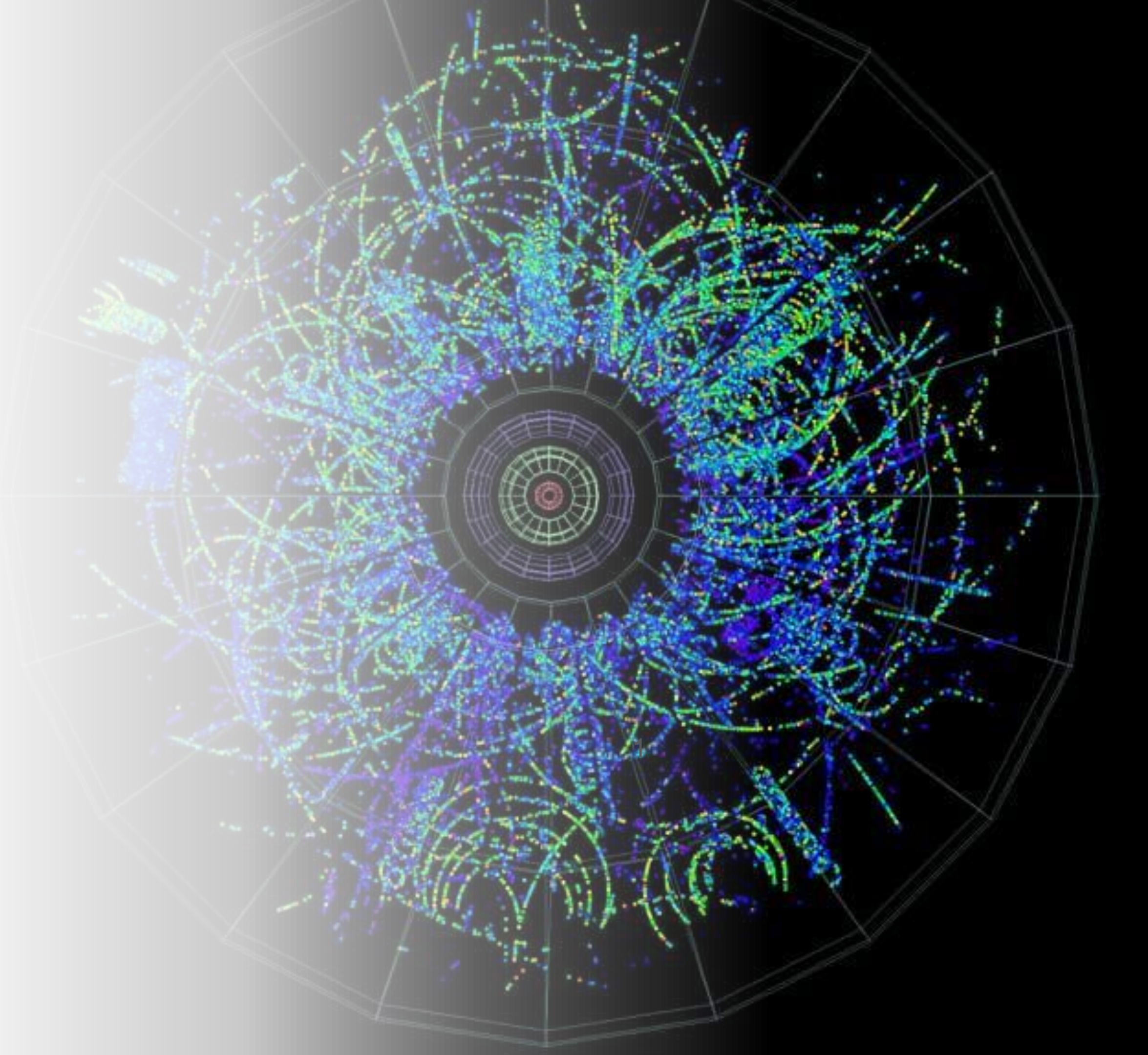
O2AT PWGLF Resonance Tutorial

Veronika, Adrian, Dukhishyam



People involved in this presentation

Veronika
PhD student

PAG Coordinators

Adrian

Dukhishyam

People involved in this presentation

Veronika
PhD student

Online support
(zoom, mattermost)

PAG Coordinators

Adrian

Dukhishyam

In person
support

Outline of this session

- General introduction to resonance analyses

- Hands-on coding session!
  - Important core tasks and configuration
  - Event and Track QA, basic Nch pT spectra
  - Phi invariant mass analysis
  - PID selection
  - Background estimation and reduction

Outline of this session

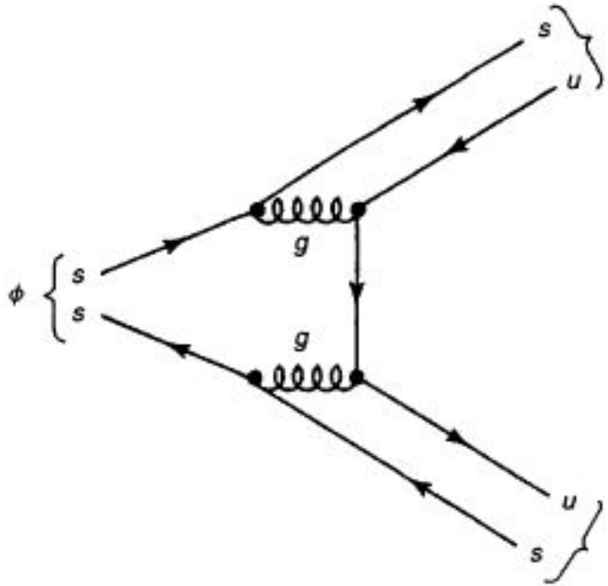
- General introduction to resonance analyses

- Hands-on coding session!
  - Important core tasks and configuration
  - Event and Track QA, basic Nch pT spectra
  - Phi invariant mass analysis
  - PID selection
  - Background estimation and reduction

Resonance basics

In a vacuum (pp):

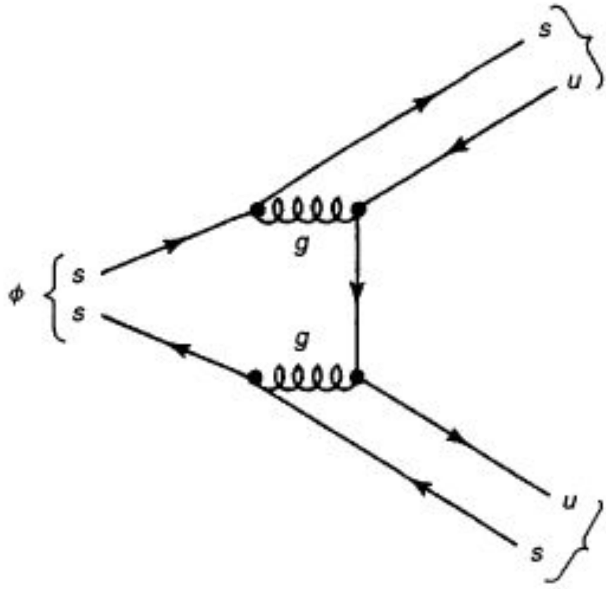
- Hadronic resonances are formed during the parton shower processes



Resonance basics

In a vacuum (pp):

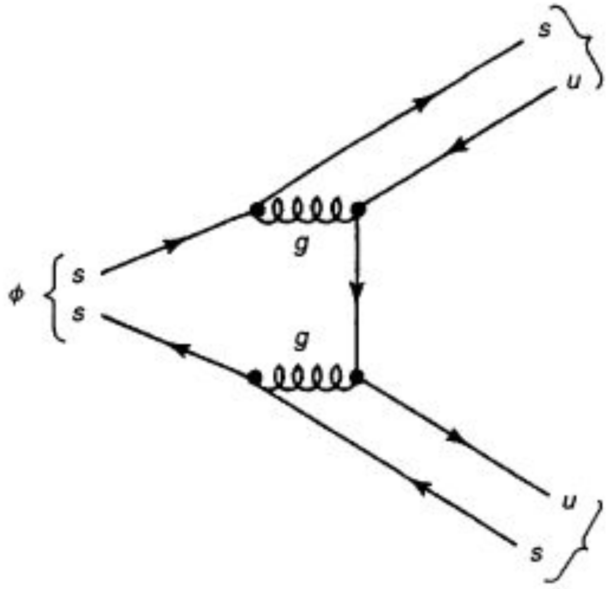
- Hadronic resonances are formed during the parton shower processes
- These particles are strongly decaying
  - Extremely short lifetimes
    - Λ^0 ~ 10^-10 s   lorentz boosted decay length ~ 8 cm
    - ϕ ~ 10^-22 s   lorentz boosted decay length ~ 46.4 fm



Resonance basics

In a vacuum (pp):

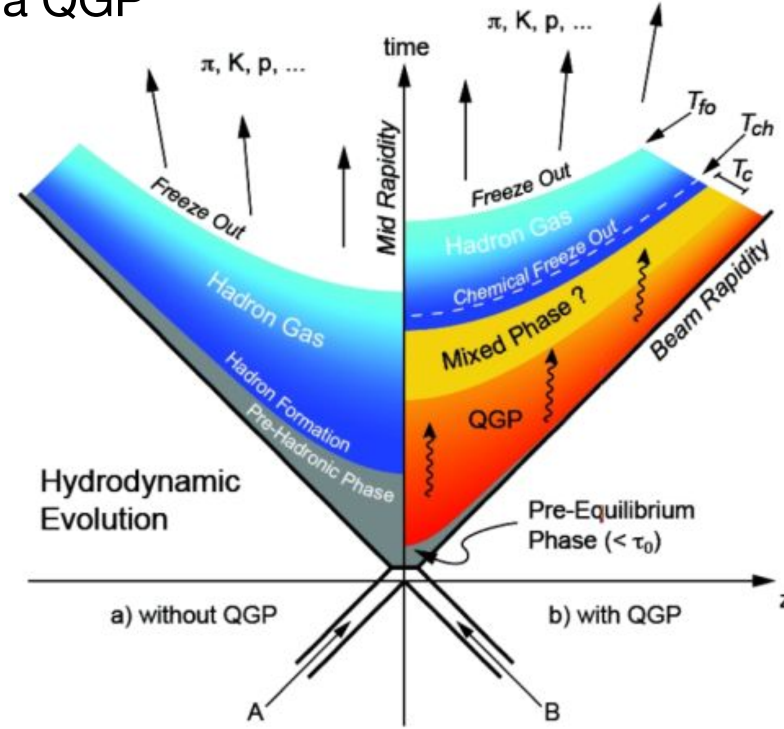
- Hadronic resonances are formed during the parton shower processes
- These particles are strongly decaying
    - Extremely short lifetimes
        - Λ^0 ~ 10^-10 s   lorentz boosted decay length ~ 8 cm
        - ϕ ~ 10^-22 s   lorentz boosted decay length ~ 46.4 fm
    - Cannot be accessed directly; can only be found through the invariant mass of its decay daughters
        - However, with such short lifetimes, experimentally indistinguishable from primary particles



Resonance basics

In a ~~vacuum~~ medium (PbPb):

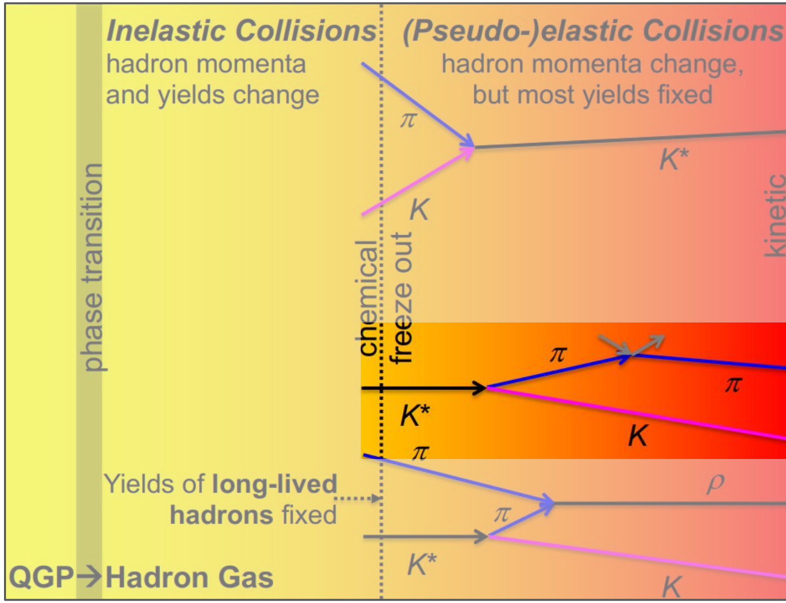
- Can decay inside the hadronic gas following a QGP
  - Allows for interactions between decay daughters
    hadron (resonance) gas following the expansion



Resonance basics

In a ~~vacuum~~ medium (PbPb):

- Can decay inside the hadronic gas following a QGP
  - Allows for interactions between decay daughters hadron (resonance) gas following the expansion
  - Rescattering
    - After chemical freeze-out, the mean-free path in the hadronic phase allows for scatterings



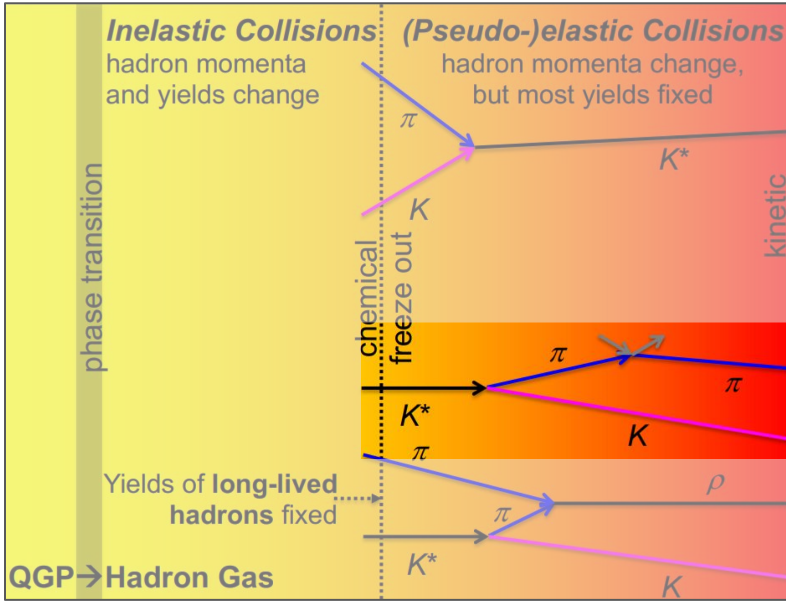
Resonance basics

In a ~~vacuum~~ medium (PbPb):

$$M = \sqrt{(E_1 + E_2)^2 - |\vec{p}_1 + \vec{p}_2|^2}$$

Changes in p
-> loss of signal

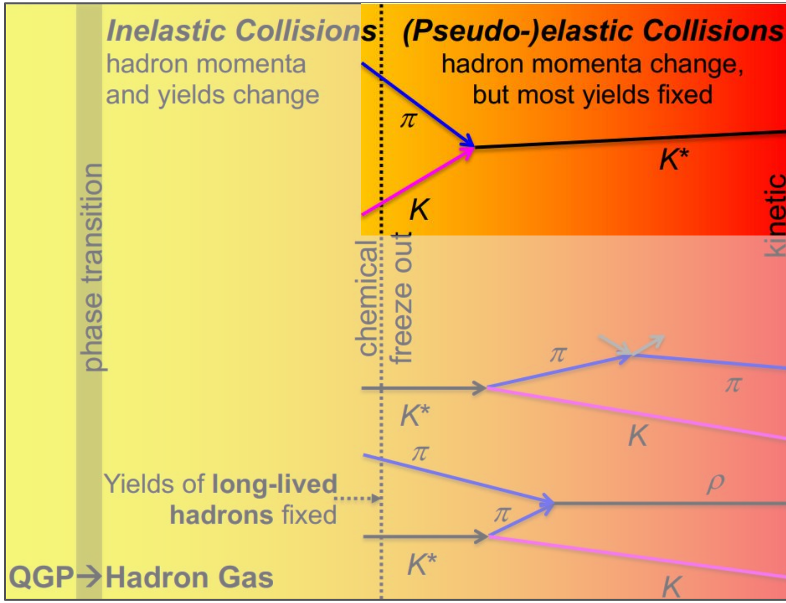
- Can decay inside the hadronic gas following a QGP
  - Allows for interactions between decay daughters hadron (resonance) gas following the expansion
  - Rescattering
    - After chemical freeze-out, the mean-free path in the hadronic phase allows for scatterings



Resonance basics

In a ~~vacuum~~ medium (PbPb):

- Can decay inside the hadronic gas following a QGP
    - Allows for interactions between decay daughters hadron (resonance) gas following the expansion
    - Rescattering
        - After chemical freeze-out, the mean-free path in the hadronic phase allows for scatterings
    - Regeneration
        - Transition of chemical equilibrium after chemical freezeout
            - Allows for reforming of resonances

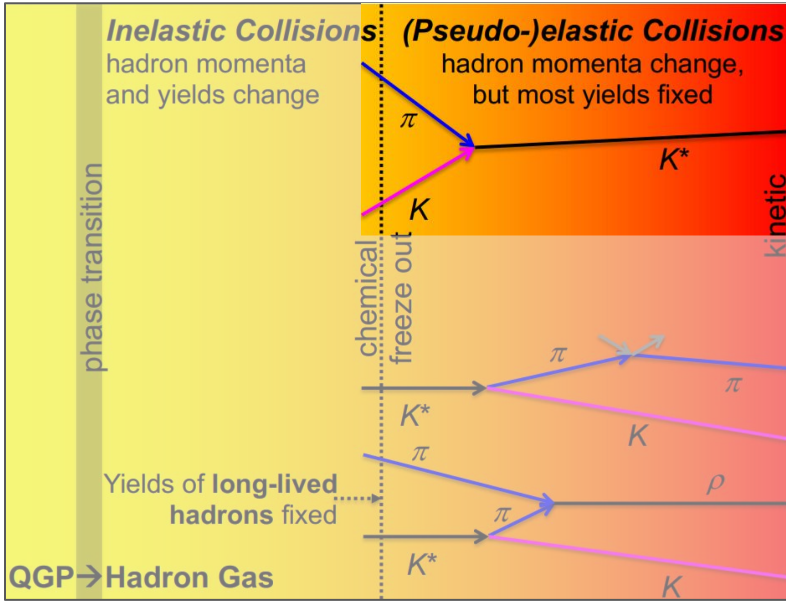


Resonance basics

In a ~~vacuum~~ medium (PbPb):

- Can decay inside the hadronic gas following a QGP
  - Allows for interactions between decay daughters hadron (resonance) gas following the expansion
  - Rescattering
    - After chemical freeze-out, the mean-free path in the hadronic phase allows for scatterings
  - Regeneration
    - Transition of chemical equilibrium after chemical freezeout
      - Allows for reforming of resonances

Gain of more resonances -> gain of signal

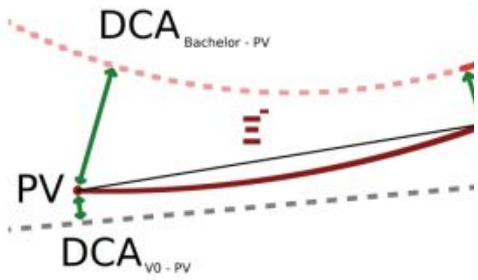


Resonance measurements

- Resonances are extremely difficult to resolve experimentally
    - Short lifetimes -> no topological pre-selection

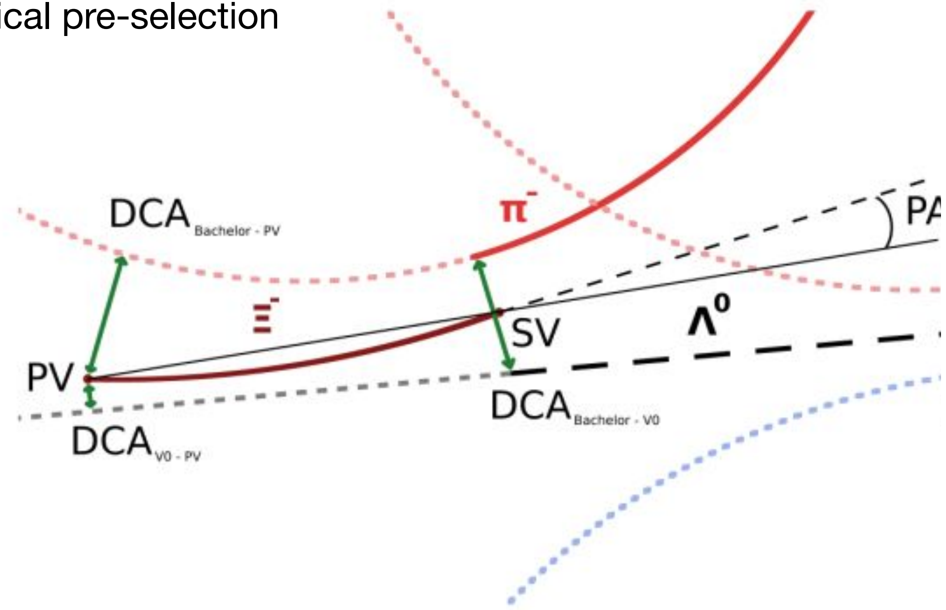
Resonance measurements

- Resonances are extremely difficult to resolve experimentally
  - Short lifetimes -> no topological pre-selection
    - Let’s compare with a Xi



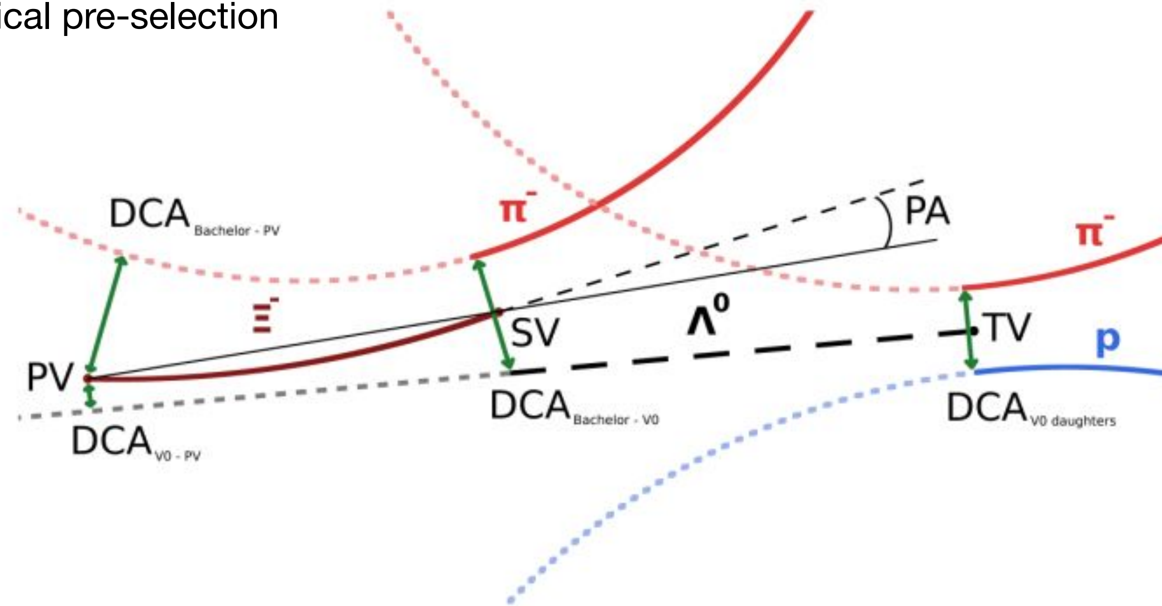
Resonance measurements

- Resonances are extremely difficult to resolve experimentally
  - Short lifetimes -> no topological pre-selection
    - Let’s compare with a Xi



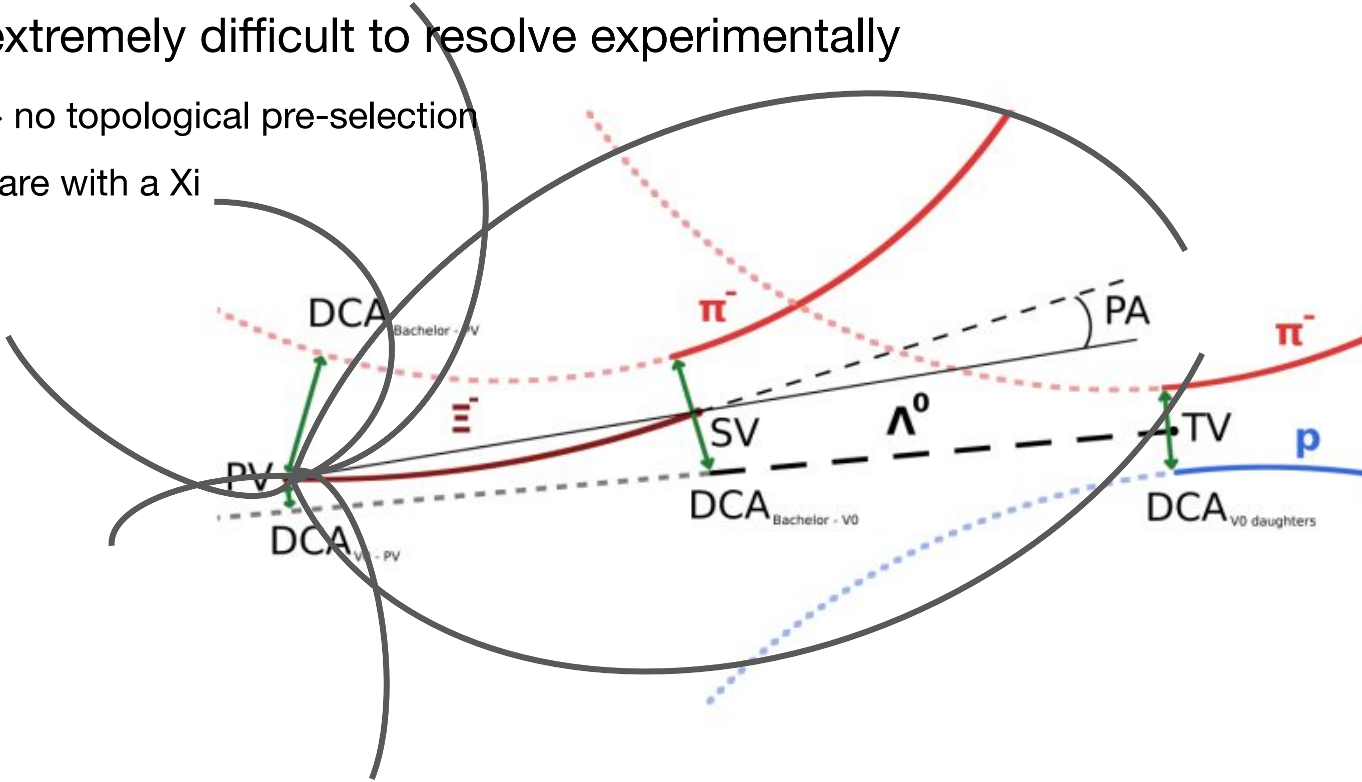
Resonance measurements

- Resonances are extremely difficult to resolve experimentally
  - Short lifetimes -> no topological pre-selection
    - Let’s compare with a Xi



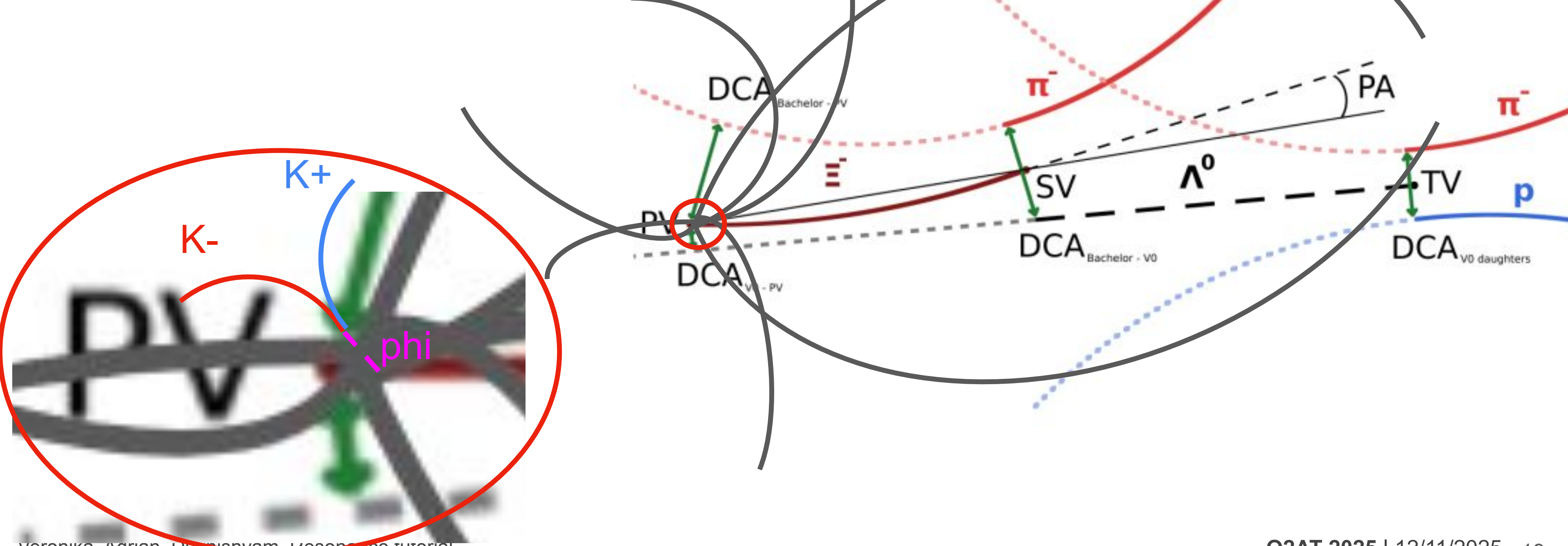
Resonance measurements

- Resonances are extremely difficult to resolve experimentally
  - Short lifetimes -> no topological pre-selection
    - Let’s compare with a Xi



Resonance measurements

- Resonances are extremely difficult to resolve experimentally
  - Short lifetimes -> no topological pre-selection
    - Let’s compare with a Xi



Resonance measurements

- Reconstruction of resonances are extremely challenges in experimentally
  - Short lifetimes & broad widths
  - Large combinatorial background
  - Particle misidentification
  - Overlapping and interfering resonances
- Our inability to pre-select based on topological structures means that

  EVERY PAIR of PRIMARY particles have to be considered as signal candidates!!
- Implications:
  - To improve signal/background ratios
  - Understand sources and shape of combinatorial background distributions
  - Choice of signal extraction fit method and fit functions

We will now try to explore on how to deal with these issues!!!!

Prerequisite for hands-on-session

ALICE

- In case you are using local O2Physics without latest tag (after daily-20251111-0000), download tutorial files here: cernbox-link
- The folder contains:
  - **Tutorial** folder with:
    - different steps of tutorial and final file `phitutorial.cxx`,
    - minimalistic configuration file `my-config.json`,
    - script to run the analysis `run.sh`
  - **AnalysisResults** folder with `AnalysisResults.root` outputs
  - **Data sample** `AO2D.root` file (in case you would like to have one locally)
- Copy files from tutorial folder to: `/your/path/O2Physics/PWGLF/Tasks/Resonances/`

Prerequisite for hands-on-session

- To build tutorial files you need to update the end of the `CMakeLists.txt` located at:
  `/your/path/O2Physics/PWGLF/Tasks/Resonances/CMakeLists.txt`
- Add lines for each step according to example:

```
o2physics_add_dpl_workflow(phitutorial-step0
        SOURCES phitutorial_step0.cxx
        PUBLIC_LINK_LIBRARIES
        O2Physics::AnalysisCore
        COMPONENT_NAME Analysis)
```

- Full content which needs to be added in cmake can be found at `README.md`
- Then rebuild O2Physics by: `aliBuild build O2Physics --debug`
  (DO NOT pull other changes to avoid building whole O2Physics!!)

Outline of this session

- General introduction to resonance analyses

- Hands-on coding session!
  - Important core tasks and configuration

  - Event and Track QA, basic Nch pT spectra

  - Phi invariant mass analysis

  - PID selection

  - Background estimation and reduction

Core service taks and configurations

- Today, we will mainly explore the core wagons.
  - There also exists dedicated ResoTables (see previous tutorials), containing derived tables with resonance information
  - This tutorial will be compatible with said tables,
    and would simply require an adjustment of specific class getters.
    - Please see the documentation page for details.

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - `o2-analysis-event-selection-service`
    - `o2-analysis-ft0-corrected-table`
    - `O2-analysis-multcenttable`
    - `o2-analysis-propagationservice`
    - `o2-analysis-trackselection`
    - `o2-analysis-pid-tpc-service`
    - `o2-analysis-pid-tof-full`
    - `o2-analysis-pid-tof-base`
    - `o2-yourtask-goes-here`

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - `o2-analysis-event-selection-service`
    - `o2-analysis-ft0-corrected-table`
    - `O2-analysis-multcenttable`
    - `o2-analysis-propagationservice`
    - `o2-analysis-trackselection`
    - `o2-analysis-pid-tpc-service`
    - `o2-analysis-pid-tof-full`
    - `o2-analysis-pid-tof-base`
    - `o2-yourtask-goes-here`

Names of some local Helper Tasks slightly differ from names of Core Service Wagons!

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - `o2-analysis-event-selection-service`
    - `o2-analysis-propagationservice`
    - `O2-analysis-trackselection`
    - `o2-analysis-ft0-corrected-table`
    - `o2-analysis-multcenttable`
    - `o2-analysis-pid-tpc-service`
    - `o2-analysis-pid-tof-full`
    - `o2-analysis-pid-tof-base`
    - `o2-yourtask-goes-here`

Baseline event+track reconstruction

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - `o2-analysis-propagationservice`
    - `o2-analysis-event-selection-service`
    - `o2-analysis-trackselection`
    - `o2-analysis-ft0-corrected-table`
    - `o2-analysis-multcenttable`
    - `o2-analysis-pid-tpc-service`
    - `o2-analysis-pid-tof-full`
    - `o2-analysis-pid-tof-base`
    - `o2-yourtask-goes-here`

Multiplicity and Centrality Selection

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - o2-analysis-propagationservice
    - o2-analysis-event-selection-service
    - o2-analysis-trackselection
    - o2-analysis-ft0-corrected-table
    - o2-analysis-multcenttable
    - o2-analysis-pid-tpc-service
    - o2-analysis-pid-tof-full
    - o2-analysis-pid-tof-base
    - o2-yourtask-goes-here

PID information from TPC and TOF

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - `o2-analysis-propagationservice`
    - `o2-analysis-event-selection-service`
    - `o2-analysis-trackselection`
    - `o2-analysis-ft0-corrected-table`
    - `o2-analysis-multcenttable`
    - `o2-analysis-pid-tpc-service`
    - `o2-analysis-pid-tof-full`
    - `o2-analysis-pid-tof-base`
    - `o2-yourtask-goes-here`

Your actual analysis task

Core service taks and configurations

- Before working on our own task, we check which support tasks we require
  - By this point, you should be familiar with the basic “run script – config file” dynamic
  - We require the following tasks:
    - `o2-analysis-event-selection-service`
    - `o2-analysis-ft0-corrected-table`
    - `O2-analysis-multcenttable`
    - `o2-analysis-propagationservice`
    - `o2-analysis-trackselection`
    - `o2-analysis-pid-tpc-service`
    - `o2-analysis-pid-tof-full`
    - `o2-analysis-pid-tof-base`
    - `o2-yourtask-goes-here`

For more information about AO2D tables and Helper tasks see [O2 Documentation]

Core service taks and configurations

- The configuration we will not discuss in detail.
  - For each task, it is advised to pull the up-to-date configuration from the corresponding hyperloop wagon

- The script that we will use as a shell is found at:

`<you-path-for-O2Physics>/PWGLF/Tasks/Resonances/phitutorial_step0.cxx`

- In total, we have 5 files, for different steps, with `phitutorial.cxx` being the complete file
  - There will be different checkpoints! So if you somehow brick your local code, you can proceed to the next step of code.

Outline of this session

- General introduction to resonance analyses

- Hands-on coding session!
  - Important core tasks and configuration
  - Event and Track QA, basic Nch pT spectra
  - Phi invariant mass analysis
  - PID selection
  - Background estimation and reduction

Step 1 – reconstructing the phi

- Here follows a quick overview of `phitutorial_step0.cxx`

Step 1 – reconstructing the phi

- Here follows a quick overview of `phitutorial_step0.cxx`
    - `The base structure of the task:`

```cpp
// Copyright 2019-2025 CERN and copyright holders of ALICE O2.
// ...

// IMPORTANT INCLUDES
#include "Common/DataModel/EventSelection.h"
// ...

// USED NAMESPACES
using namespace o2;
// ...

// MAIN STRUCT
struct phitutorial {
// ...
// ...
// ...
};

WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
{
  return WorkflowSpec{adaptAnalysisTask<phitutorial_step0>(cfgc)};
};
```

Step 1 – reconstructing the phi

- **Inside the main struct:**

```
// MAIN STRUCT
struct phitutorial_step0 {

  SliceCache cache;
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(o2::framework::InitContext&)
  {
    const AxisSpec ptAxis = {200, 0, 20.0};
    const AxisSpec MinvAxis = {200, 0.85, 1.25};

    histos.add("Nch_pT", "Nch_pT", kTH1F, {ptAxis});
  };

  // TRACK AND EVENT CANDIDATES
  using EventCandidates = soa::Join<aod::Collisions, aod::EvSels, aod::FT0Mults, aod::CentFT0Ms>;
  using TrackCandidates = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::TrackSelection,
aod::pidTPCFullKa, aod::pidTOFFullKa>;

  double massKa = o2::constants::physics::MassKPlus;

  // HELPER FUNCTIONS
  template <typename EventType>
  bool eventSelection(const EventType event)
  {
    if (!event.sel8()) //This is required to extract good events
      return false;
    return true;
  };

  // PROCESS
  void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){

  // ...

  }
  PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
};
```

Step 1 – reconstructing the phi

ALICE

- **Inside the main struct:**
  - Slice Cache and histogram registry definition

```
// MAIN STRUCT
struct phitutorial_step0 {

  SliceCache cache;
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(o2::framework::InitContext&)
  {
    const AxisSpec ptAxis = {200, 0, 20.0};
    const AxisSpec MinvAxis = {200, 0.85, 1.25};

    histos.add("Nch_pT", "Nch_pT", kTH1F, {ptAxis});
  };

  // TRACK AND EVENT CANDIDATES
  using EventCandidates = soa::Join<aod::Collisions, aod::EvSels, aod::FT0Mults, aod::CentFT0Ms>;
  using TrackCandidates = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::TrackSelection,
aod::pidTPCFullKa, aod::pidTOFFullKa>;

  double massKa = o2::constants::physics::MassKPlus;

  // HELPER FUNCTIONS
  template <typename EventType>
  bool eventSelection(const EventType event)
  {
    if (!event.sel8()) //This is required to extract good events
      return false;
    return true;
  };

  // PROCESS
  void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){

  // ...

  }
  PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
};
```

Step 1 – reconstructing the phi

- **Inside the main struct:**
  - Slice Cache and histogram registry definition
  - **Initialization** (definition of axis specifications and histograms)

```
// MAIN STRUCT
struct phitutorial_step0 {

  SliceCache cache;
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(o2::framework::InitContext&)
  {
    const AxisSpec ptAxis = {200, 0, 20.0};
    const AxisSpec MinvAxis = {200, 0.85, 1.25};

    histos.add("Nch_pT", "Nch_pT", kTH1F, {ptAxis});
  };

  // TRACK AND EVENT CANDIDATES
  using EventCandidates = soa::Join<aod::Collisions, aod::EvSels, aod::FT0Mults, aod::CentFT0Ms>;
  using TrackCandidates = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::TrackSelection,
aod::pidTPCFullKa, aod::pidTOFFullKa>;

  double massKa = o2::constants::physics::MassKPlus;

  // HELPER FUNCTIONS
  template <typename EventType>
  bool eventSelection(const EventType event)
  {
    if (!event.sel8()) //This is required to extract good events
      return false;
    return true;
  };

  // PROCESS
  void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){

  // ...

  }
  PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
};
```

Step 1 – reconstructing the phi

- **Inside the main struct:**
  - Slice Cache and histogram registry definition
  - Initialization (definition of axis specifications and histograms)
  - Enabling tables needed for **Event and Track candidates**

```
// MAIN STRUCT
struct phitutorial_step0 {

  SliceCache cache;
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(o2::framework::InitContext&)
  {
    const AxisSpec ptAxis = {200, 0, 20.0};
    const AxisSpec MinvAxis = {200, 0.85, 1.25};

    histos.add("Nch_pT", "Nch_pT", kTH1F, {ptAxis});
  };

  // TRACK AND EVENT CANDIDATES
  using EventCandidates = soa::Join<aod::Collisions, aod::EvSels, aod::FT0Mults, aod::CentFT0Ms>;
  using TrackCandidates = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::TrackSelection,
aod::pidTPCFullKa, aod::pidTOFFullKa>;

  double massKa = o2::constants::physics::MassKPlus;

  // HELPER FUNCTIONS
  template <typename EventType>
  bool eventSelection(const EventType event)
  {
    if (!event.sel8()) //This is required to extract good events
      return false;
    return true;
  };

  // PROCESS
  void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){

  // ...

  }
  PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
};
```

Step 1 – reconstructing the phi

- **Inside the main struct:**
  - Slice Cache and histogram registry definition
  - Initialization (definition of axis specifications and histograms)
  - Enabling tables needed for Event and Track candidates
  - Declaration of **helper functions** (for clarity of code)

```
// MAIN STRUCT
struct phitutorial_step0 {

  SliceCache cache;
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(o2::framework::InitContext&)
  {
    const AxisSpec ptAxis = {200, 0, 20.0};
    const AxisSpec MinvAxis = {200, 0.85, 1.25};

    histos.add("Nch_pT", "Nch_pT", kTH1F, {ptAxis});
  };

  // TRACK AND EVENT CANDIDATES
  using EventCandidates = soa::Join<aod::Collisions, aod::EvSels, aod::FT0Mults, aod::CentFT0Ms>;
  using TrackCandidates = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::TrackSelection,
aod::pidTPCFullKa, aod::pidTOFFullKa>;

  double massKa = o2::constants::physics::MassKPlus;

  // HELPER FUNCTIONS
  template <typename EventType>
  bool eventSelection(const EventType event)
  {
    if (!event.sel8()) //This is required to extract good events
      return false;
    return true;
  };

  // PROCESS
  void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){

  // ...

  }
  PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
};
```

Step 1 – reconstructing the phi

- **Inside the main struct:**
  - Slice Cache and histogram registry definition
  - Initialization (definition of axis specifications and histograms)
  - Enabling tables needed for Event and Track candidates
  - Declaration of helper functions (for clarity of code)
  - Declaration of **process function**

```
// MAIN STRUCT
struct phitutorial_step0 {

  SliceCache cache;
  HistogramRegistry histos{"histos", {}, OutputObjHandlingPolicy::AnalysisObject};

  void init(o2::framework::InitContext&)
  {
    const AxisSpec ptAxis = {200, 0, 20.0};
    const AxisSpec MinvAxis = {200, 0.85, 1.25};

    histos.add("Nch_pT", "Nch_pT", kTH1F, {ptAxis});
  };

  // TRACK AND EVENT CANDIDATES
  using EventCandidates = soa::Join<aod::Collisions, aod::EvSels, aod::FT0Mults, aod::CentFT0Ms>;
  using TrackCandidates = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::TrackSelection,
aod::pidTPCFullKa, aod::pidTOFFullKa>;

  double massKa = o2::constants::physics::MassKPlus;

  // HELPER FUNCTIONS
  template <typename EventType>
  bool eventSelection(const EventType event)
  {
    if (!event.sel8()) //This is required to extract good events
      return false;
    return true;
  };

  // PROCESS
  void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){

  // ...

  }
  PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
};
```

Step 1 – reconstructing the phi

- **Process function**
    - Basic task, which creates a pT spectra for charged tracks

Using Event and Track candidates defined earlier

iterator (no need to loop over collisions)

Our predefined helper function

Filling the histogram with pT

```
int nEvents = 0;
void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks) {
  nEvents++;
  if ((nEvents + 1) % 10000 == 0) {
    std::cout << "Processed Data Events: " << nEvents << std::endl;
  }

  if (!eventSelection(collision))
    return;

  for (const auto& track : tracks) {
    histos.fill(HIST("Nch_pT"), track.pt());
  }
}
PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
```

Step 1 – reconstructing the phi

- You can run the analysis by:

1. Directly from command line:

```
o2-analysis-propagationservice -b  --configuration json://path/to/my/config/my-config.json  | o2-analysis-trackselection -b  --configuration json://path/to/my/config/my-config.json  | o2-analysis-tracks-extra-v002-converter -b --configuration json://path/to/my/config/my-config.json | o2-analysis-ft0-corrected-table -b  --configuration json://path/to/my/config/my-config.json | o2-analysis-pid-tpc-service -b  --configuration json://path/to/my/config/my-config.json | o2-analysis-event-selection-service -b  --configuration json://path/to/my/config/my-config.json| o2-analysis-multcenttable -b  --configuration json://path/to/my/config/my-config.json  | o2-analysis-pid-tof-full -b  --configuration json://path/to/my/config/my-config.json | o2-analysis-pid-tof-base -b  --configuration json://path/to/my/config/my-config.json | o2-analysistutorial-lf-phitutorial-step0 -b  --configuration json://path/to/my/config/my-config.json
```

2. Creating and running a bash script `run.sh` containing:

```
MY_CONFIG="/path/to/my/config/my-config.json"

o2-analysis-propagationservice -b  --configuration json://$MY_CONFIG  | \
o2-analysis-trackselection -b  --configuration json://$MY_CONFIG  | \
o2-analysis-tracks-extra-v002-converter -b --configuration json://$MY_CONFIG  | \
o2-analysis-ft0-corrected-table -b  --configuration json://$MY_CONFIG  | \
o2-analysis-pid-tpc-service -b  --configuration json://$MY_CONFIG  | \
o2-analysis-event-selection-service -b  --configuration json://$MY_CONFIG | \
o2-analysis-multcenttable -b  --configuration json://$MY_CONFIG  | \
o2-analysis-pid-tof-full -b  --configuration json://$MY_CONFIG  | \
o2-analysis-pid-tof-base -b  --configuration json://$MY_CONFIG  | \
o2-analysistutorial-lf-phitutorial-step0 -b  --configuration json://$MY_CONFIG
```

Outline of this session

- General introduction to resonance analyses
- Hands-on coding session!
  - Important core tasks and configuration
  - Event and Track QA, basic Nch pT spectra
  - Phi invariant mass analysis
  - PID selection
  - Background estimation and reduction

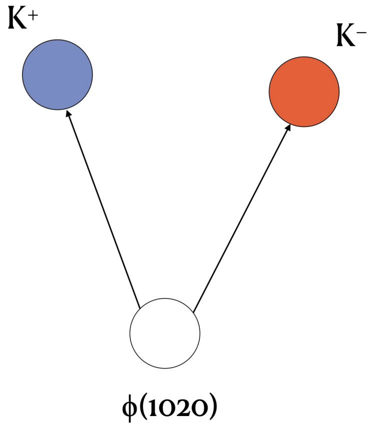
Step 1 – reconstructing the phi

- **Process function**
  - Basic task, which creates a pT spectra for charged tracks
  - Now we have to add logic for **phi invariant mass reconstruction**

```
int nEvents = 0;
void processDataSameEvent(EventCandidates::iterator const& collision, TrackCandidates const& tracks){
  nEvents++;
  if ((nEvents + 1) % 10000 == 0) {
    std::cout << "Processed Data Events: " << nEvents << std::endl;
  }

  if (!eventSelection(collision))
    return;

  for (const auto& track : tracks) {
    histos.fill(HIST("Nch_pT"), track.pt());
  }

  // ...
  // Place for your implementation
  // ...

}
PROCESS_SWITCH(phitutorial_step0, processDataSameEvent, "process Data Same Event", false);
```

Step 1 – reconstructing the phi

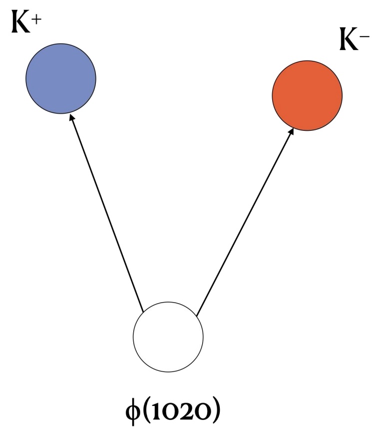
- **Process function**
  - Basic task, which creates a pT spectra for charged tracks
  - Now we have to add logic for **phi invariant mass reconstruction**
  - We want to study phi decay via: $\Phi \rightarrow K^+K^-$



Step 1 – reconstructing the phi

- **Process function**
  - Basic task, which creates a pT spectra for charged tracks
  - Now we have to add logic for **phi invariant mass reconstruction**
  - We want to study phi decay via: Φ→K+K-
  - Invariant mass formula:

$$M^2_{K^+K^-} = m^2_{K^+} + m^2_{K^-} + 2(E_{K^+}E_{K^-} - \overrightarrow{p_{K^+}} . \overrightarrow{p_{K^-}})$$

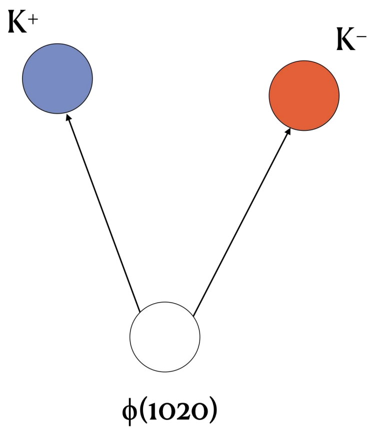


Step 1 – reconstructing the phi

- **Process function**
  - Basic task, which creates a pT spectra for charged tracks
  - Now we have to add logic for **phi invariant mass reconstruction**
  - We want to study phi decay via: Φ→K+K-
  - Invariant mass formula:

$$M^2_{K^+K^-} = m^2_{K^+} + m^2_{K^-} + 2(E_{K^+}E_{K^-} - \overrightarrow{p_{K^+}} \cdot \overrightarrow{p_{K^-}})$$

  - There are multiple ways of solving this problem!
    - Check the comments in `phitutorial_step0.cxx` for hints!

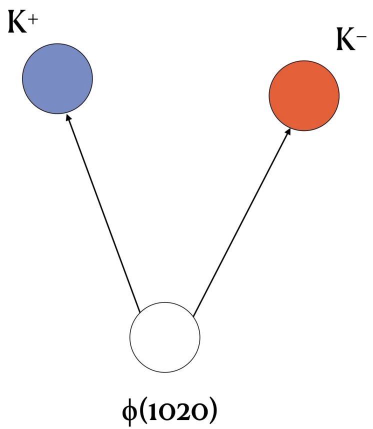


Step 1 – reconstructing the phi

- **Process function**
  - Basic task, which creates a pT spectra for charged tracks
  - Now we have to add logic for **phi invariant mass reconstruction**
  - We want to study phi decay via: Φ→K+K-
  - Invariant mass formula:

$$M^2_{K^+K^-} = m^2_{K^+} + m^2_{K^-} + 2(E_{K^+}E_{K^-} - \overrightarrow{p_{K^+}} . \overrightarrow{p_{K^-}})$$

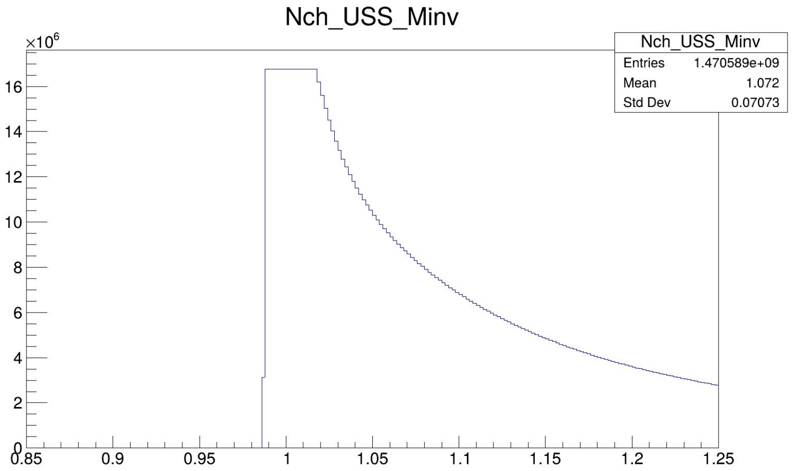
  - There are multiple ways of solving this problem!
    - Check the comments in `phitutorial_step0.cxx` for hints!



NOW WE TRY TO SOLVE THIS FOR 10-15 MINUTES

Step 1 – reconstructing the phi

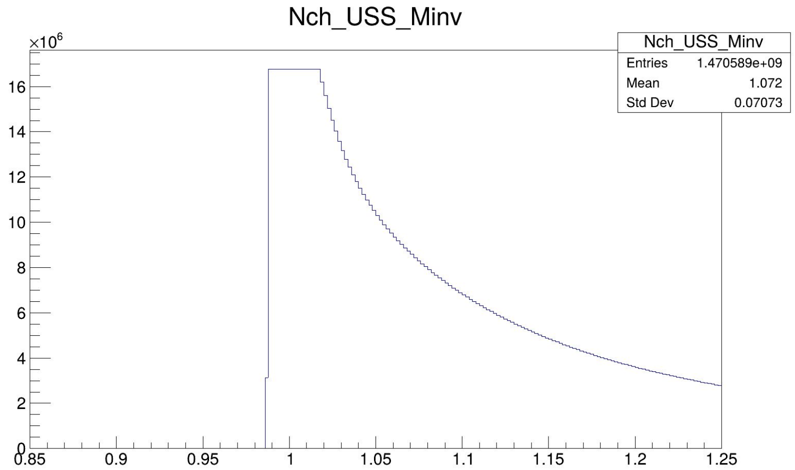
- If you've done everything correctly, you should obtain something like this:
  - There are many paths to this:



Step 1 – reconstructing the phi

- If you've done everything correctly, you should obtain something like this:
  - There are many paths to this:
    1. Double **for loop** through all tracks and manually calculating invariant mass **via formula**

```
for(track1:tracks){
    for(track2:tracks){
     pt1 = track1.pt();
     pz1 = pt1*sinh(track1.eta());
     p1 = TMath::sqrt(pt1*pt1 +pz1*pz1)
     E1 = sqrt(p1*p1 + mK*mK)
     …
     M_phi = TMath::sqrt((E1+E2)*(E1+E2)-((p1+p2)(p1+p2)))
    }
}
```



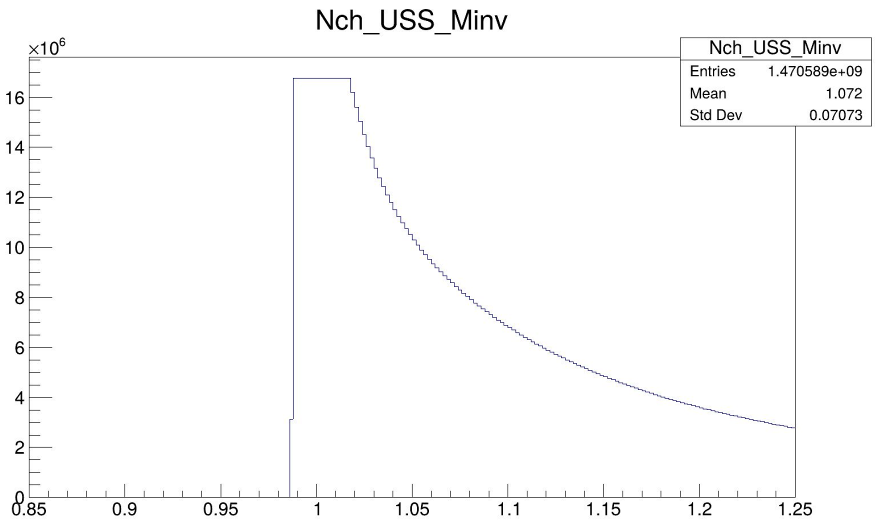
Step 1 – reconstructing the phi

- If you've done everything correctly, you should obtain something like this:
  - There are many paths to this:
    1. Double for loop through all tracks and manually calculating invariant mass via formula
    2. Using helper functions from `ASoAHelpers.h` to get **combinations of tracks** (For more info see O2 Documentation) and `ROOT::TMath::PxPyPzMVector()` for invariant mass calculation (ROOT Documentation)

```
for (const auto& [trk1, trk2] : combinations(o2::soa::CombinationsStrictlyUpperIndexPolicy(tracks, tracks))) {

    ROOT::Math::PxPyPzMVector lDecayDaughter1, lDecayDaughter2, lResonance;
    lDecayDaughter1 = ROOT::Math::PxPyPzMVector(trk1.px(), trk1.py(), trk1.pz(), massKa);
    lDecayDaughter2 = ROOT::Math::PxPyPzMVector(trk2.px(), trk2.py(), trk2.pz(), massKa);

    lResonance = lDecayDaughter1 + lDecayDaughter2;
    double conjugate = trk1.sign() * trk2.sign();
    if (conjugate < 0) {
      histos.fill(HIST("Nch_USS_Minv"), lResonance.M());
    }
  }
```

Step 1 – reconstructing the phi

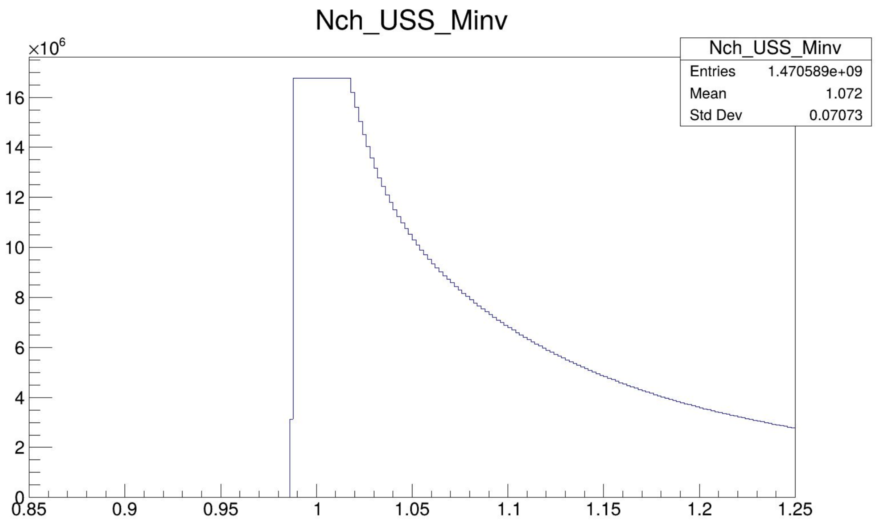
- If you've done everything correctly, you should obtain something like this:
  - There are many paths to this.
- Regardless of how you ended up doing this, this does not look that good…
  - There's a slope, not a peak!
  - How do we improve this?



Step 1 – reconstructing the phi

- If you’ve done everything correctly, you should obtain something like this:
  - There are many paths to this.
- Regardless of how you ended up doing this, this does not look that good…
  - There’s a slope, not a peak!
  - How do we improve this?
  - We have to improve our track quality!

For now with no PID



Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`

Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`
  - We want to accept only tracks which pass our quality checks:

Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`
  - We want to accept only tracks which pass our quality checks:
  - For example:
    - Globally good track quality → **track.isGlobalTrack() == true**

      kGlobalTrack = kQualityTracks | kPrimaryTracks | kInAcceptanceTracks

      kQualityTracks = kTrackType | kQualityTracksITS | kQualityTracksTPC

      kPrimaryTracks = kGoldenChi2 | kDCAxy | kDCAz;

      kInAcceptanceTracks = kPtRange | kEtaRange;

      (More here)

Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`
  - We want to accept only tracks which pass our quality checks:
  - For example:
    - Globally good track quality → **track.isGlobalTrack() == true**

      kGlobalTrack = kQualityTracks | kPrimaryTracks | kInAcceptanceTracks

      kQualityTracks = kTrackType | kQualityTracksITS | kQualityTracksTPC

      kPrimaryTracks = kGoldenChi2 | kDCAxy | kDCAz;

      kInAcceptanceTracks = kPtRange | kEtaRange;

      (More here)
    - To remove lowest momentum tracks → **track.pt() > 0.15f**

Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`
  - We want to accept only tracks which pass our quality checks:
  - For example:
    - Globally good track quality → **track.isGlobalTrack() == true**

      kGlobalTrack = kQualityTracks | kPrimaryTracks | kInAcceptanceTracks

      kQualityTracks = kTrackType | kQualityTracksITS | kQualityTracksTPC

      kPrimaryTracks = kGoldenChi2 | kDCAxy | kDCAz;

      kInAcceptanceTracks = kPtRange | kEtaRange;

      (More here)
    - To remove lowest momentum tracks → **track.pt() > 0.15f**
    - Cut on η to keep only tracks with good TPC efficiency → **std::abs(tacks.eta()) > 0.8**

Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`
  - We want to accept only tracks which pass our quality checks:
  - For example:
    - Globally good track quality → **track.isGlobalTrack() == true**

      kGlobalTrack = kQualityTracks | kPrimaryTracks | kInAcceptanceTracks

      kQualityTracks = kTrackType | kQualityTracksITS | kQualityTracksTPC

      kPrimaryTracks = kGoldenChi2 | kDCAxy | kDCAz;

      kInAcceptanceTracks = kPtRange | kEtaRange;

      (More here)
    - To remove lowest momentum tracks → **track.pt() > 0.15f**
    - Cut on η to keep only tracks with good TPC efficiency → **std::abs(tacks.eta()) > 0.8**
  - And others, but we can work with these three for now

Step 2 – Track selection

- You can continue in `phitutorial_step1.cxx`
  - We want to accept only tracks which pass our quality checks:
  - For example:
    - Globally good track quality → **track.isGlobalTrack() == true**

      kGlobalTrack = kQualityTracks | kPrimaryTracks | kInAcceptanceTracks

      kQualityTracks = kTrackType | kQualityTracksITS | kQualityTracksTPC

      kPrimaryTracks = kGoldenChi2 | kDCAxy | kDCAz;

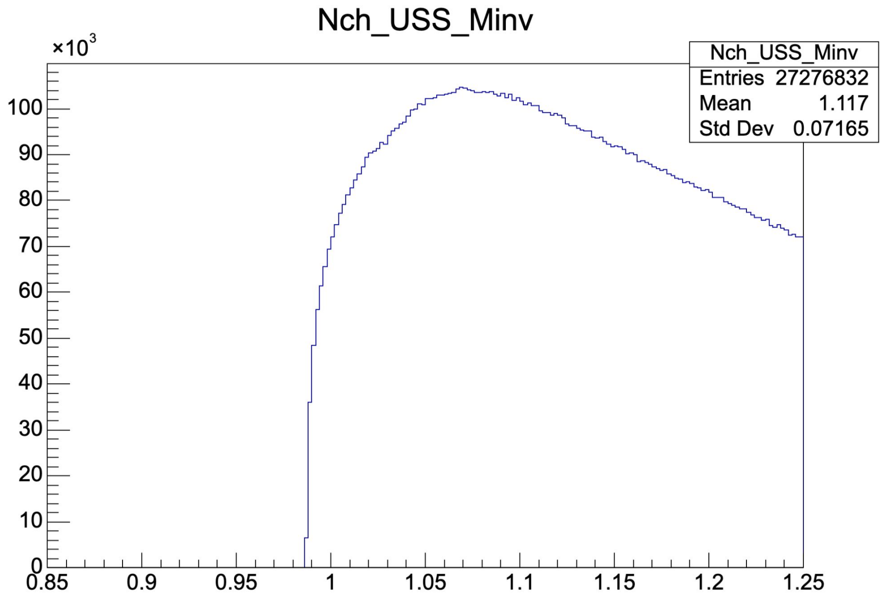
      kInAcceptanceTracks = kPtRange | kEtaRange;

      (More here)
    - To remove lowest momentum tracks → **track.pt() > 0.15f**
    - Cut on η to keep only tracks with good TPC efficiency → **std::abs(tacks.eta()) > 0.8**
  - And others, but we can work with these three for now

LET'S TRY TO ADD THIS TO YOUR TASK (5-10 MINUTES)

Step 2 – Track selection

- After applying track selection our invariant mass distribution is changing like this:
  - We observe different shape
  - But still no significantly visible peak!
- By now we were working with all charged particles within our track selections
- We were not using any particle identification (PID) to most likely select kaons → **So let's do it now!**



Outline of this session

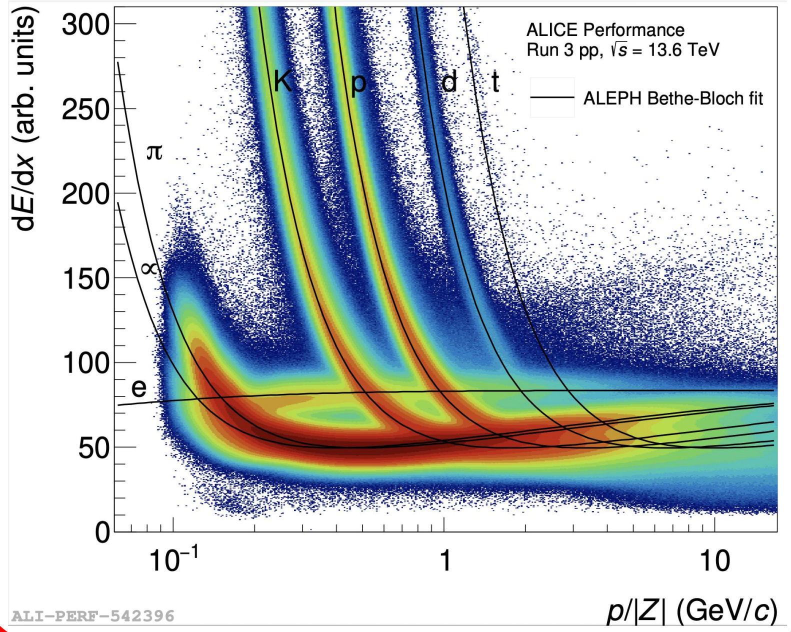
- General introduction to resonance analyses
- Hands-on coding session!
  - Important core tasks and configuration
  - Event and Track QA, basic Nch pT spectra
  - Phi invariant mass analysis
  - PID selection
  - Background estimation and reduction

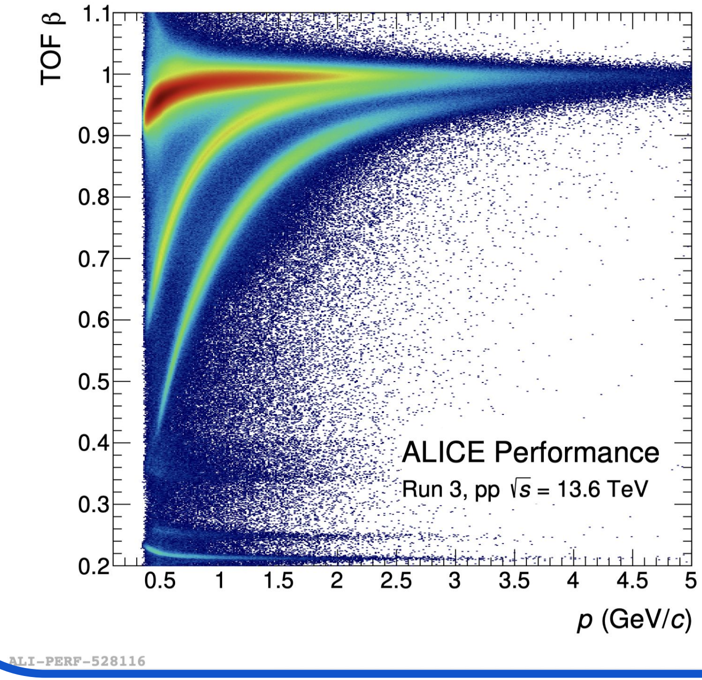
Step 2 – Particle Identification

- Hints for implementing PID can be found in `phitutorial_step2.cxx`

Step 2 – Particle Identification

- We are using information from 2 detectors for PID: **TPC** and **TOF**
- Usual range for nσ is 3 (deviation from Bethe-Bloch fit)





Step 2 – Particle Identification

- There are 3 ways of doing this:

Step 2 – Particle Identification

- There are 3 ways of doing this:
    1. Directly cut on the tracks in the looping functions

By applying these cuts for both tracks:

```
std::abs(track1.tpcNSigmaKa()) > 3

std::abs(track1.tofNSigmaKa()) > 3
```

Not the most efficient way!

Step 2 – Particle Identification

- There are 3 ways of doing this:
  1. Directly cut on the tracks in the looping functions
  2. Creating a helper function similar to track or event selection

A lot of tracks with good TPC PID does not have TOF information.

You can make a conditional cut on TOF only if track has information from TOF.

That means only if `track.hasTOF()` returns TRUE.

Than using similarly to other helper functions:

```
if (!trackPIDKaon(track1)||!trackPIDKaon(track2))
    continue;
```

```
template <typename TrackPID>
bool trackPIDKaon(const TrackPID& candidate)
{
  bool tpcPIDPassed{false}, tofPIDPassed{false};
  // TPC
  if (std::abs(candidate.tpcNSigmaKa()) < 3)
    tpcPIDPassed = true;
  // TOF
  if (candidate.hasTOF()) {
    if (std::abs(candidate.tofNSigmaKa()) < 3) {
      tofPIDPassed = true;
    }
  } else {
    tofPIDPassed = true;
  }
  // TPC & TOF
  if (tpcPIDPassed && tofPIDPassed) {
    return true;
  }
  return false;
}
```

Step 2 – Particle Identification

- There are 3 ways of doing this:
  1. Directly cut on the tracks in the looping functions
  2. Creating a helper function similar to track or event selection
  3. Partitioning your tracks with a preselection

By adding Partition declaration outside the process function:

```
Partition<TrackCandidates>kaon(nabs(aod::pidtpc::tpcNSigmaKa)<=3);
```

Than adding slicing inside your process function:

```
auto tracks = kaon->sliceByCached(aod::track::collisionId,collision.globalIndex(),cache);
```

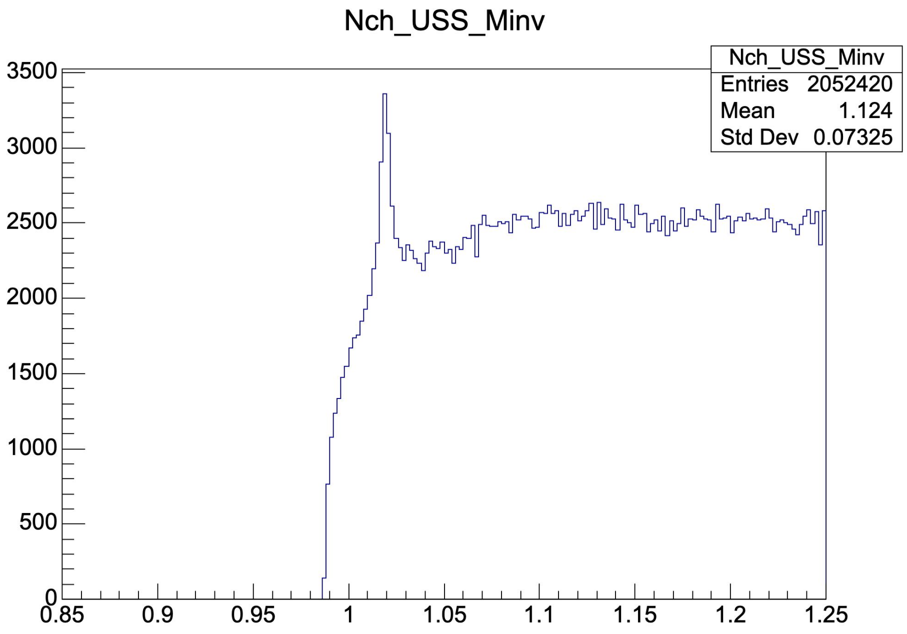
Step 2 – Particle Identification

- There are 3 ways of doing this → No matter which way we choose the result should be the same:

CHOOSE ONE APPROACH AND IMPLEMENT IT (10-15 MINUTES)

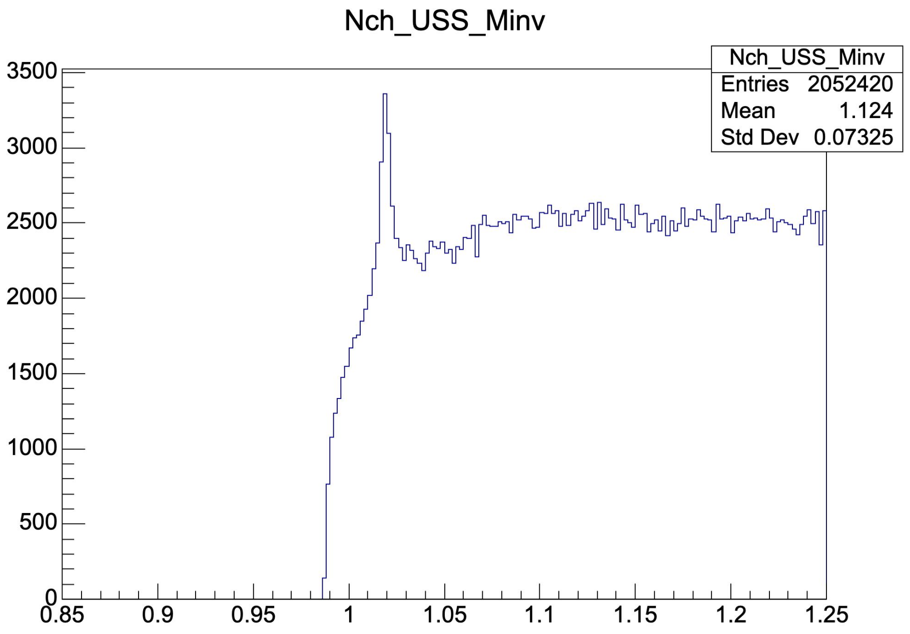
Step 2 – Particle Identification

- There are 3 ways of doing this → No matter which way we choose the result should be the same:
  - Clearly visible peak around 1.02 GeV/c2
  - The peak is sitting on high combinatorial background.



Step 2 – Particle Identification

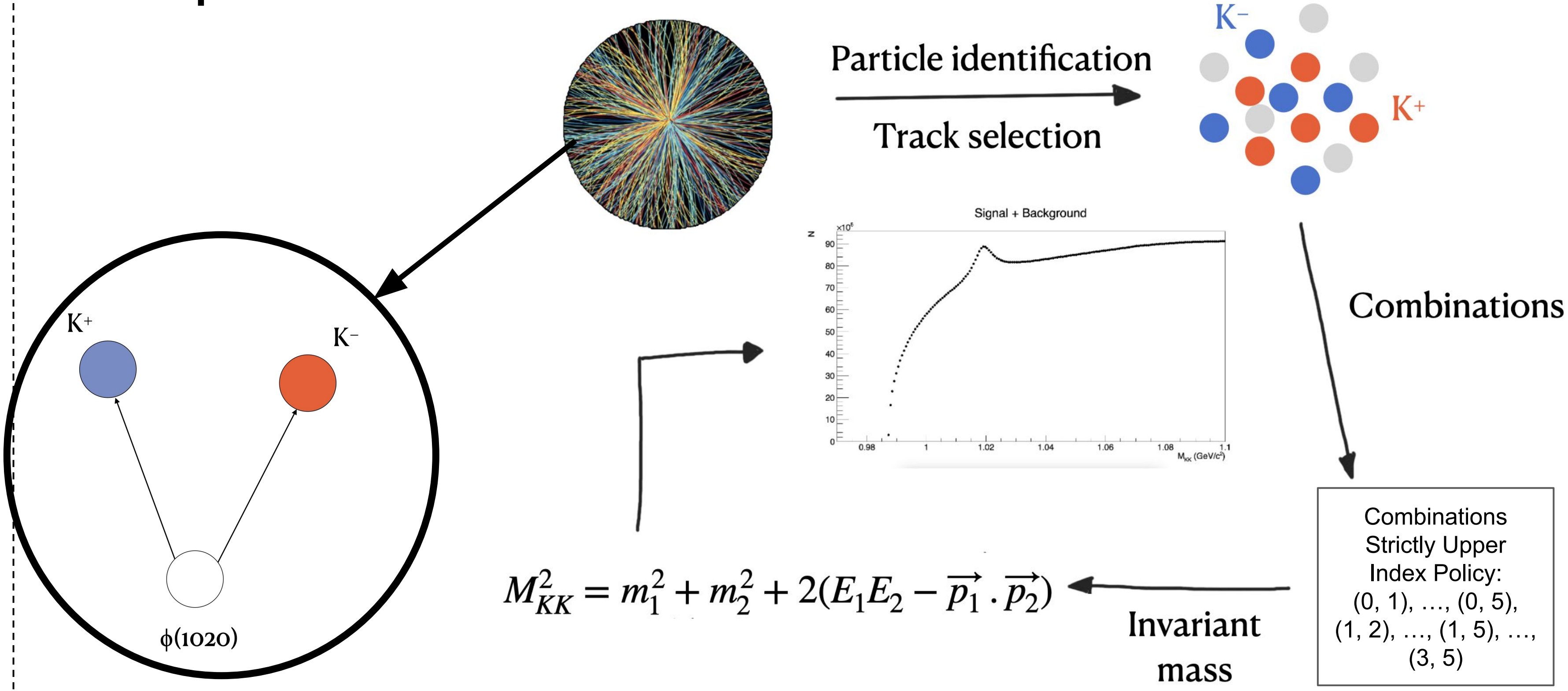
- There are 3 ways of doing this → No matter which way we choose the result should be the same:
    - Clearly visible peak around 1.02 GeV/c2
    - The peak is sitting on high combinatorial background.
- Background can be estimated by generating invariant mass distribution of pairs of kaons which we know for sure not originated from phi meson.



Move to step 3

BUT FIRST: RECAP

- **Full process function**



Outline of this session

- General introduction to resonance analyses

- Hands-on coding session!
  - Important core tasks and configuration
  - Event and Track QA, basic Nch pT spectra
  - Phi invariant mass analysis
  - PID selection
  - Background estimation and reduction

Step 3 – Background estimation

- There are multiple methods of background estimation
- In this tutorial 2 examples:

Like Sign
- Made of combinations of two K+ or two K-
- Simplest option

Event Mixing
- Combinations of kaons from different but similar events
- Similarity criteria for example:
  - Primary vertex position in z-axis
  - FT0M multiplicity

Step 3 – Background estimation

- There are multiple methods of background estimation
- In this tutorial 2 examples:

Like Sign

- Made of combinations of two K+ or two K-
- Simplest option

Event Mixing

- Combinations of kaons from different but similar events
- Similarity criteria for example:
  - Primary vertex position in z-axis
  - FT0M multiplicity

By filling the new histogram when the signs of kaons are the same,

```
track1.sign()*track2.sign() > 0
```

By implementing the separate process in which we iterate over track pairs from different collisions.

See part of the O2 documentation about Event Mixing

Step 3 – Background estimation

- There are multiple methods of background estimation
- In this tutorial 2 examples:

Like Sign
- Made of combinations of two K+ or two K-
- Simplest option

By filling the new histogram when the signs of kaons are the same,

```
track1.sign()*track2.sign() > 0
```

CREATE YOUR BACKGROUND (10-15 MINUTES)

Event Mixing
- Combinations of kaons from different but similar events
- Similarity criteria for example:
  - Primary vertex position in z-axis
  - FT0M multiplicity

By implementing the separate process in which we iterate over track pairs from different collisions.

See part of the O2 documentation about Event Mixing

Step 3 – Background estimation

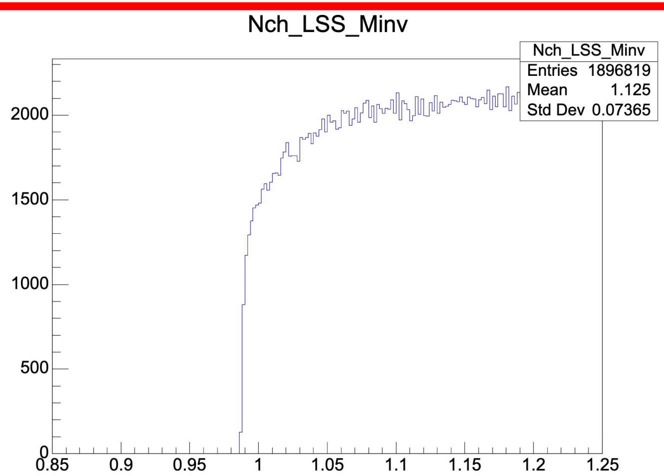
- There are multiple methods of background estimation
- In this tutorial 2 examples:

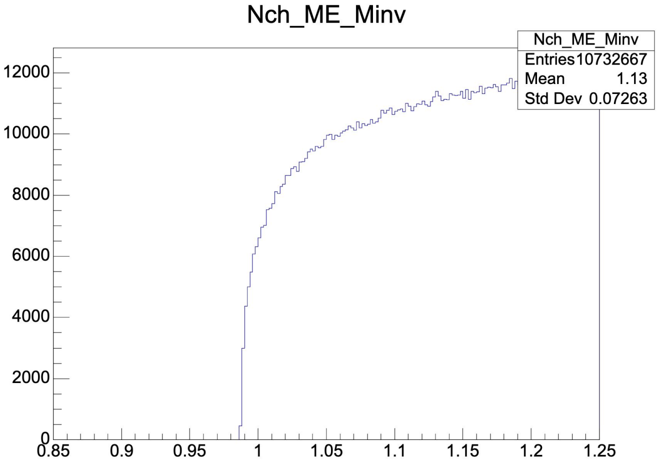
Like Sign

- Made of combinations of two K+ or two K-
- Simplest option

Event Mixing

- Combinations of kaons from different but similar events
- Similarity criteria for example:
  - Primary vertex position in z-axis
  - FT0M multiplicity





Outline of this session

- General introduction to resonance analyses

- Hands-on coding session!
    - Important core tasks and configuration
    - Event and Track QA, basic Nch pT spectra
    - Phi invariant mass analysis
    - PID selection
    - Background estimation and reduction

Congratulations,

you’ve completed the tutorial!

The real game begins now — welcome to Level 1:
Your Own Analysis.

If you have any questions, don’t hesitate to reach out – via DM on Mattermost, by email, or by joining the Resonances Mattermost channel at Resonance O2Framework.

Back up

Resonance measurements

- Reconstruction of resonances are extremely challenges in experimentally
  - Short lifetimes & broad widths
  - Large combinatorial background
  - Particle misidentification
  - Overlapping and interfering resonances
- Our inability to pre-select based on topological structures means that

  EVERY PAIR of PRIMARY particles have to be considered as signal candidates!!
- Implications:
  - To improve signal/background ratios
  - Understand sources and shape of combinatorial background distributions
  - Choice of signal extraction fit method or fit functions

Resonance measurements

- Resonances are extremely difficult to resolve experimentally
    - Short lifetimes -> no topological pre-selection

- Our inability to pre-select based on topological structures means that

  EVERY PAIR of PRIMARY particles have to be considered as signal candidates!!

- Implications:
    - Our signal/background ratios are usually quite bad
    - Difficult background structures
    - Fine-tuning of fits are required

We will now try to explore on how to deal with these issues!!!!