Analysis Tutorial: Unified framework for femto analysis

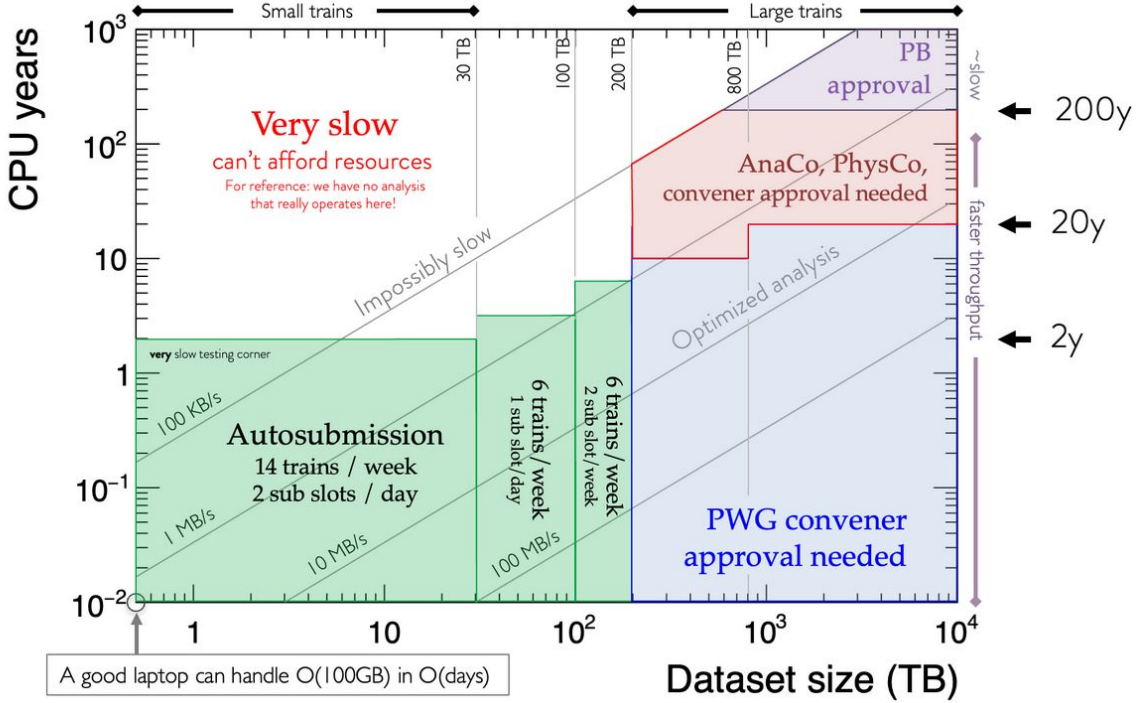
Anton Riedel
Technical University of Munich

O2 Analysis Tutorial 5.0
13.11.2025

Outline

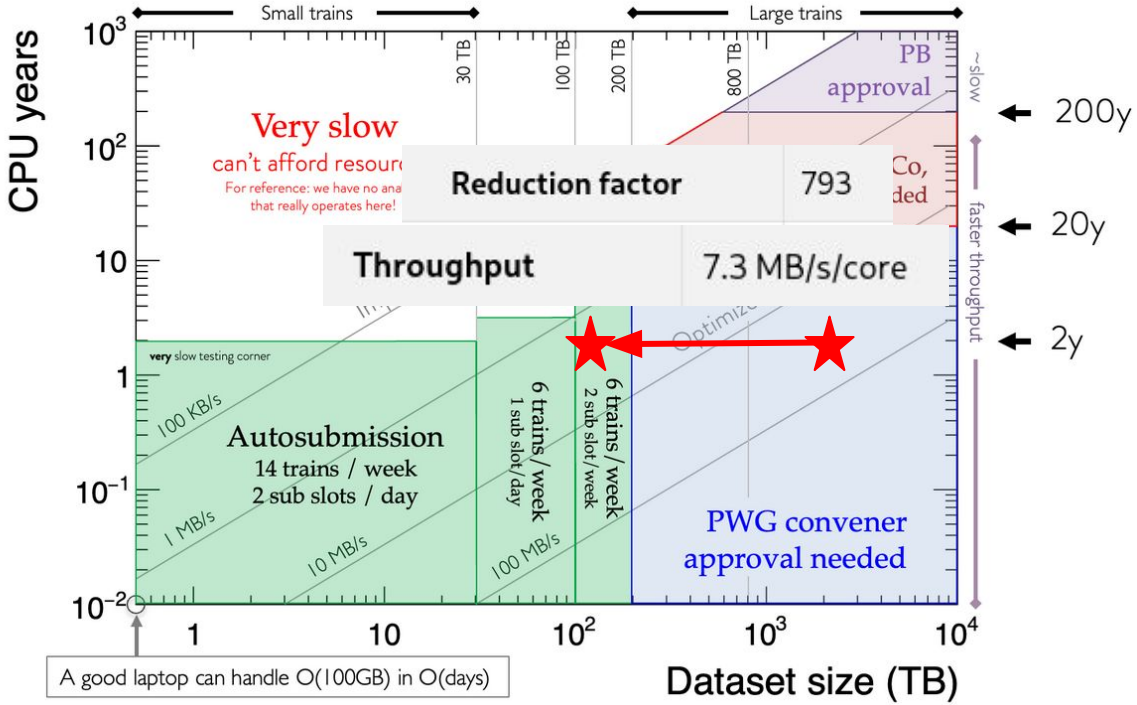
1. Motivation for a (new) Unified Framework
2. Available Tables in the Framework
3. Populating the Tables
4. Quality Assurance (QA) Tasks
5. Analysis Tasks
6. Hands on

Why do we need framework with custom data model?



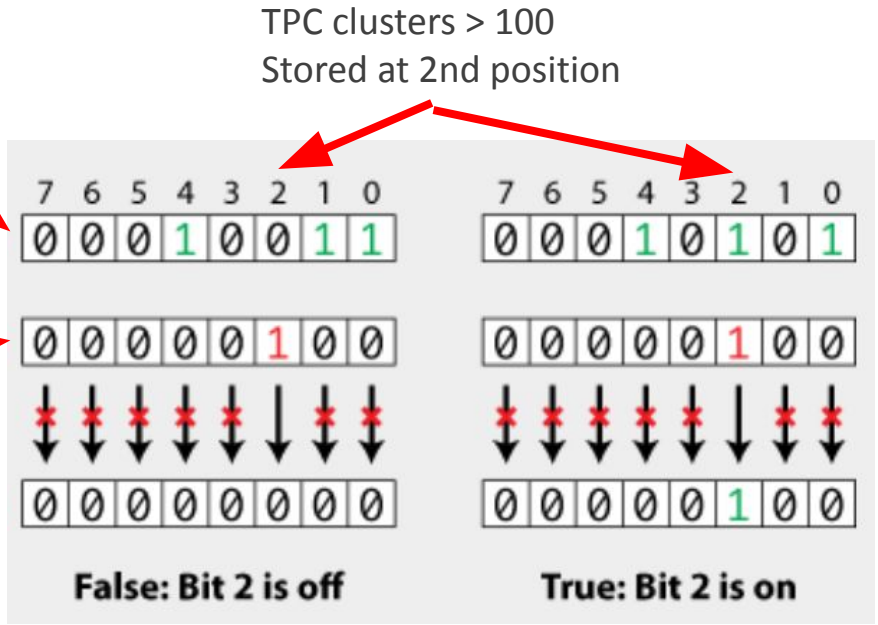
- Femto analysis are very statistics hungry, so most analysis will try to run over the largest possible dataset (e.g. 2023_thin ~ 1.5 PB)
- For femto not much information about tracks & events is needed (kinematics, zVtx, Mult, …)
- Great opportunity to produced derived data with **large compression** to allow for fast and efficient analysis

Why do we need framework with custom data model?



- Femto analysis are very statistics hungry, so most analysis will try to run over the largest possible dataset (e.g. 2023_thin ~ 1.5 PB)
- For femto not much information about tracks & events is needed (kinematics, zVtx, Mult, …)
- Great opportunity to produced derived data with **large compression** to allow for fast and efficient analysis

How do we compress information about selections?



- Compress information whether a specific cut was passed in a series of bits
- At analysis time analyzer specify which cut they want to “turn on/off” by configuring a mask
- Tables can be partitioned into particles that pass the selections **up front**, no need string endless if statements together

Track did not pass the cut, track excluded from analysis

Track did pass the cut, track excluded from analysis

How do we compress information about selections?

- Defined track partition once and it can be reused over and over

```
// standard track partition
#define MAKE_TRACK_PARTITION(selection)
  ifnode(selection.chargeSign.node() != 0, ifnode(selection.chargeSign.node() > 0, femtoba
    (nabs(selection.chargeAbs.node() * femtobase::stored::signedPt) > selection.ptMin) &&
    (nabs(selection.chargeAbs.node() * femtobase::stored::signedPt) < selection.ptMax) &&
    (femtobase::stored::eta > selection.etaMin) &&
    (femtobase::stored::eta < selection.etaMax) &&
    (femtobase::stored::phi > selection.phiMin) &&
    (femtobase::stored::phi < selection.phiMax) &&
    ifnode(nabs(selection.chargeAbs.node() * femtobase::stored::signedPt) * (nexp(femtobas
           ncheckbit(femtotracks::mask, selection.maskLowMomentum),
           ncheckbit(femtotracks::mask, selection.maskHighMomentum))
```

- PID information also compressed in bit string stored in the table, so we need to masks for low and high momentum

Why a new datamodel/framework?

```
DECLARE_SOA_TABLE(FDParticles, "AOD", "FDPARTICLE",
                  o2::soa::Index<>,
                  femtodreamparticle::FDCollisionId,
                  femtodreamparticle::Pt,
                  femtodreamparticle::Eta,
                  femtodreamparticle::Phi,
                  femtodreamparticle::PartType,
                  femtodreamparticle::Cut,
                  femtodreamparticle::PIDCut,
                  femtodreamparticle::TempFitVar,
                  femtodreamparticle::ChildrenIds,
                  femtodreamparticle::MLambda,
                  femtodreamparticle::MAntiLambda,
```

6 int32 + 6 floats + 1 int8 = 49 Bytes

Main issue: Same table format for tracks and V0s!

Unnecessary information is stored for tracks and V0

Since no information about V0 children is stored, V0 cannot be partitioned based on cuts on the children

Extension to cascades not easily possible

- Need a variable to define particle type, i.e. Track, V0, …
- PID and track quality cuts are stored in different bits
- Storing PID information for V0s
- Storing information about children ID and mass unnecessarily for tracks

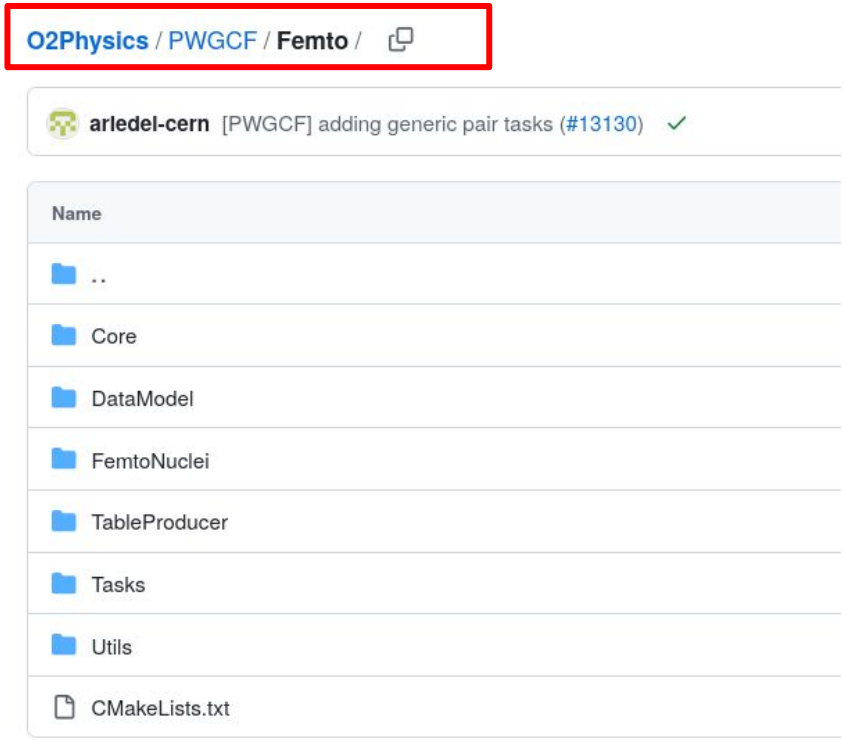
Goal:

- Use dedicated data table for tracks and V0s, cascades, resonancess
- Make data format overall more modular

This should …

- Make inclusion of new particle types easier
- Offers more flexibility in terms of track information/selections during analysis/QA
- Larger compression!

Framework now part of O2Physics!



- Framework available in PWGF/Femto
- Old files were moved to FemtoNuclei folder
  - Moving the other frameworks as a subfolder into this folder?
- Most important files are the
  - PWGCF/Femto/DataModel/FemtoTables.h - One file containing all table definitions
  - PWGCF/Femto/TableProducer/femtoProducer.cxx - One file producing all tables
- Under task there are the QA tasks for QAing single particles and generic pair tasks that can be used for analysis

Table for Collisions

```
// table for basic collision information
DECLARE_SOA_TABLE_STAGED_VERSIONED(FCols_001, "FCOL", 1, //! femto collisions
                                   o2::soa::Index<>,
                                   femtocollisions::PosZ,
                                   femtocollisions::Mult,
                                   femtocollisions::Cent,
                                   femtocollisions::Sphericity,
                                   femtocollisions::MagField);
using FCols = FCols_001;
using StoredFCols = StoredFCols_001;

// table for collisions selections
DECLARE_SOA_TABLE_STAGED_VERSIONED(FColMasks_001, "FCOLMASK", 1, //! track masks
                                   femtocollisions::CollisionMask);
using FColMasks = FColMasks_001;
using StoredFColMasks = StoredFColMasks_001;
```

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

- Storing additional information for collision selection in dedicated bitmask
- Maybe adding the cut on sphericity and magnetic field to the mask?

RCTFlag and trigger selections

- For analysis on skimmed data, only events for a specific trigger can be selected
- Event selection is internally handled by Zorro

```
struct ConfCollisionTriggers : o2::framework::ConfigurableGroup {
  std::string prefix = std::string("CollisionTriggers");
  o2::framework::Configurable<bool> useTrigger{"useTrigger", false, "Set to true to only selected triggered collisic
  o2::framework::Configurable<std::string> ccdbPath{"ccdbPath", std::string("EventFiltering/Zorro/"), "CCDB path for
  o2::framework::Configurable<std::string> triggers{"triggers", std::string("fPPP,fPPL"), "Comma seperated list of a
};

struct ConfCollisionRctFlags : o2::framework::ConfigurableGroup {
  std::string prefix = std::string("CollisionRctFlags");
  o2::framework::Configurable<bool> useRctFlags{"useRctFlags", true, "Set to true to use RCT flags"};
  o2::framework::Configurable<std::string> label{"label", std::string("CBT_hadronPID"), "Which RCT flag to check"};
  o2::framework::Configurable<bool> useZdc{"useZdc", false, "Whether to use ZDC (only use for PbPb)"};
  o2::framework::Configurable<bool> treatLimitedAcceptanceAsBad{"treatLimitedAcceptanceAsBad", false, "Whether to tr
};
```

- DPG is providing RCTFlagChecker to have time dependent run quality
- By default using CBT_hadronPID (-> good quality from ITS, TPC and TOF) and limited acceptance is treated as NOT bad (i.e. MC reproducible)
- Event and RCTFlag cuts are implicit, so the producer only builds events which pass these cuts, but no bits are stored

Collision selection

```
struct ConfCollisionBits : o2::framework::ConfigurableGroup {
  std::string prefix = std::string("CollisionBits");
  o2::framework::Configurable<int> sel8{"sel8", 1, "Use sel8 (-1: stored in bitmaks; 0 off; 1 on)"};
  o2::framework::Configurable<int> noSameBunchPileup{"noSameBunchPileup", 0, "Reject collisions in case of pileup
  f; 1 on)"};
  o2::framework::Configurable<int> isVertexItsTpc{"isVertexItsTpc", 0, "At least one ITS-TPC track found for the v
  o2::framework::Configurable<int> isGoodZvtxFt0VsPv{"isGoodZvtxFt0VsPv", 0, "small difference between z-vertex fr
  o2::framework::Configurable<int> noCollInTimeRangeNarrow{"noCollInTimeRangeNarrow", 0, "no other collisions in s
  ; 1 on)"};
  o2::framework::Configurable<int> noCollInTimeRangeStrict{"noCollInTimeRangeStr                    ns in s
  o2::framework::Configurable<int> noCollInTimeRangeStandard{"noCollInTimeRangeS                    isions
   (-1: stored in bitmaks; 0 off; 1 on)"};
  o2::framework::Configurable<int> noCollInRofStrict{"noCollInRofStrict", 0, "no                    eadout
  o2::framework::Configurable<int> noCollInRofStandard{"noCollInRofStandard", 0, "no other collisions in this Read
  n bitmaks; 0 off; 1 on)"};
  o2::framework::Configurable<int> noHighMultCollInPrevRof{"noHighMultCollInPrevRof", 0, "veto an event if FT0C am
  0 off; 1 on)"};
  o2::framework::Configurable<int> isGoodItsLayer3{"isGoodItsLayer3", 0, "number of inactive chips on ITS layer 3
  o2::framework::Configurable<int> isGoodItsLayer0123{"isGoodItsLayer0123", 0, "                    tive chips on ITS l
  f; 1 on)"};
  o2::framework::Configurable<int> isGoodItsLayersAll{"isGoodItsLayersAll", 0, "                    tive chips on all I
  f; 1 on)"};
  o2::framework::Configurable<std::vector<float>> occupancyMin{"occupancyMin", {                    "};
  o2::framework::Configurable<std::vector<float>> occupancyMax{"occupancyMax", {                    "};
};
```

General event quality and pile up rejection

ITS layers

Occupancy cut

- All cuts are available for pp and PbPb
- Cuts on event quality can be either off, implicit (no bit stored) or explicit (with a bit stored)

Collision selection

```
[23:39:21][INFO] Initialize femto collision builder...
[23:39:21][INFO] Enabled femto table (auto): FCols_001
[23:39:21][INFO] Enabled femto table (auto): FColMasks_001
[23:39:21][INFO] Enabled femto table (auto): FColPos_001
[23:39:21][INFO] Printing Configuration of Collision Selection Object
[23:39:21][INFO] Observable: Is good zvtx FT0 vs PV (index 3)
[23:39:21][INFO]   Limit type           : Equal
[23:39:21][INFO]   Minimal cut          : no
[23:39:21][INFO]   Skip most permissive : no
[23:39:21][INFO]   Bitmask shift        : 1
[23:39:21][INFO]   Selections           :
[23:39:21][INFO]     1.000000            -> bitmask: 1
[23:39:21][INFO]
[23:39:21][INFO] Observable: No same bunch pileup (index 1)
[23:39:21][INFO]   Limit type           : Equal
[23:39:21][INFO]   Minimal cut          : no
[23:39:21][INFO]   Skip most permissive : no
[23:39:21][INFO]   Bitmask shift        : 1
[23:39:21][INFO]   Selections           :
[23:39:21][INFO]     1.000000            -> bitmask: 2
[23:39:21][INFO]
[23:39:21][INFO] Observable: Sel8 (index 0)
[23:39:21][INFO]   Limit type           : Equal
[23:39:21][INFO]   Minimal cut          : yes
[23:39:21][INFO]   Skip most permissive : yes
[23:39:21][INFO]   Bitmask shift        : 0
[23:39:21][INFO]   Selections           :
[23:39:21][INFO]     1.000000            -> loosest minimal selection, no bit saved
[23:39:21][INFO]
[23:39:21][INFO] Printing done
[23:39:21][INFO] Initialization done...
```

- femtoProduer is very verbose during initialization
- Example for configuration of collsionBuilder:
  - Explicit cut on **IsGoodVtxFT0VsPv** and **noSameBunchPileUp** (-> collision are stored, result of the cut is stored in bitmask
  - Implicity cut on **Sel8** (-> collision only stored of Sel8 is fulfilled, not bit stored)

To activate all cuts:

```
  0b0001  (1)
+ 0b0010  (2)
----------
  0b0011  (3)
```

Table for Tracks

```
// table for basic track information
DECLARE_SOA_TABLE_STAGED_VERSIONED(FTracks_001, "FTRACK", 1, //! femto tracks
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId,
                                   femtobase::stored::SignedPt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::dynamic::Sign<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::Pt<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::P<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);

using FTracks = FTracks_001;
using StoredFTracks = StoredFTracks_001;
```

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

- **Use signed Pt of the track to determine its charge instead of reserving a special bit in the bitmask**
- Pt and other kinetic variables are still accessible via dynamic columns
- “Stored” tables are produced by second stage derived data production

Table for Tracks

- Trackmask has a dedicated table

```
// table for track selections and PID selections
DECLARE_SOA_TABLE_STAGED_VERSIONED(FTrackMasks_001, "FTRACKMASK", 1, //! track masks
                                   femtotracks::TrackMask);
using FTrackMasks = FTrackMasks_001;
using StoredFTrackMasks = StoredFTrackMasks_001;
```

- Can be joined with the track table for selections

```
using Tracks = o2::soa::Join<FUTracks, FUTrackMasks>;
```

2 int32 + 3 floats + 1 int64 = 28 Bytes

- Previously stored track quality and PID in separate masks (32bit each), now combined into one mask of size 64bit

```
using TrackMaskType = uint64_t;
```

- Extended columns for debugging and QA also available, staged and versioned

```
using Tracks = o2::soa::Join<FTracks, FTrackMasks, FTrackDcas, FTrackExtras, FTrackPids>;
```

Example: Bitmask for tracks

```
Observable: Min. fraction of TPC clusters over TPC crossed rows (index 2)
  Limit type           : Lower Limit
  Minimal cut          : yes
  Skip most permissive : yes
  Bitmask shift        : 2
  Selections           :
    0.830000            -> loosest minimal selection, no bit saved
    0.900000            -> bitmask: 65536
    0.950000            -> bitmask: 131072

Observable: Min. number of crossed TPC rows (index 1)
  Limit type           : Lower Limit
  Minimal cut          : yes
  Skip most permissive : yes
  Bitmask shift        : 2
  Selections           :
    80.000000           -> loosest minimal selection, no bit saved
    90.000000           -> bitmask: 262144
    100.000000          -> bitmask: 524288

Observable: Min. number of TPC clusters (index 0)
  Limit type           : Lower Limit
  Minimal cut          : yes
  Skip most permissive : yes
  Bitmask shift        : 2
  Selections           :
    90.000000           -> loosest minimal selection, no bit saved
    100.000000          -> bitmask: 1048576
    110.000000          -> bitmask: 2097152
```

- Here is a cutout from producer output
- To compute the bitmask, select the cuts you want and just add the associated bitmasks
- Here:

Fraction cluster/shared > 0.95 -> 131072
Crossed Rows > 100 -> 524288
Min. Cluster > 110 -> 2097152

```
  0b00001000000000000000      (131072)
+ 0b00100000000000000000      (524288)
+ 0b10000000000000000000      (2097152)
-----------------------------------------
  0b10101000000000000000      (2752512)
```

Bitmask for Tracks and more

```
template <typename T, typename BitmaskType>
class SelectionContainer
```

- SelectionContainer is storing all possible values for one specific cut and can generate the mask for a given value

```
private:
 std::vector<T> mSelectionValues = {};                          ///< Values used for the selection
 std::vector<TF1> mSelectionFunctions = {};                     ///< Function used for the selection
 limits::LimitType mLimitType;                                  ///< Limit type of selection
 std::bitset<sizeof(BitmaskType) * CHAR_BIT> mBitmask = {}; ///< bitmask for the observable
 bool mSkipMostPermissiveBit = false;                           ///< whether to skip the last bit or not
 bool mIsMinimalCut = false;                                    ///< whether to use this observable for minimal selection or not
};
```

- Instead of values, also functions can be given (needed for DCA selections)
- In case of, e.g., track quality cuts, they all need to be passed simultaneously for a track to be accepted, so we can mark them as “Minimal Cut”
- Similarly for track quality, if they need to be passed for the track to be accepted at all, we do not need to bit for the loosest selection, so this bit can be skipped

Bitmask for Tracks and more

```cpp
for (auto const& selectionContainer : mSelectionContainers) {
  // if there are no selections for a certain observable, skip
  if (selectionContainer.isEmpty()) {
    continue;
  }
  // Shift the result to make space and add the new value
  mFinalBitmask = (mFinalBitmask << selectionContainer.getShift()) | selectionContainer.getBitmask();
}
```

Example:

- Mask for Cut1: 011
- Maks for Cut2: 0111

1. FinalBitmask = 00000000
   a. Shift by 3 00000000 << 3 = 00000000
   b. Add bits of mask1: 00000000 || 00000011 = 00000011
2. FinalBitmask = 00000011
   a. Shift by 4 00000011 << 4 = 00110000
   b. Add bits of maks2: 00110000 || 00000111 = 00110111
3. Keep going

Bitmask for Tracks and more

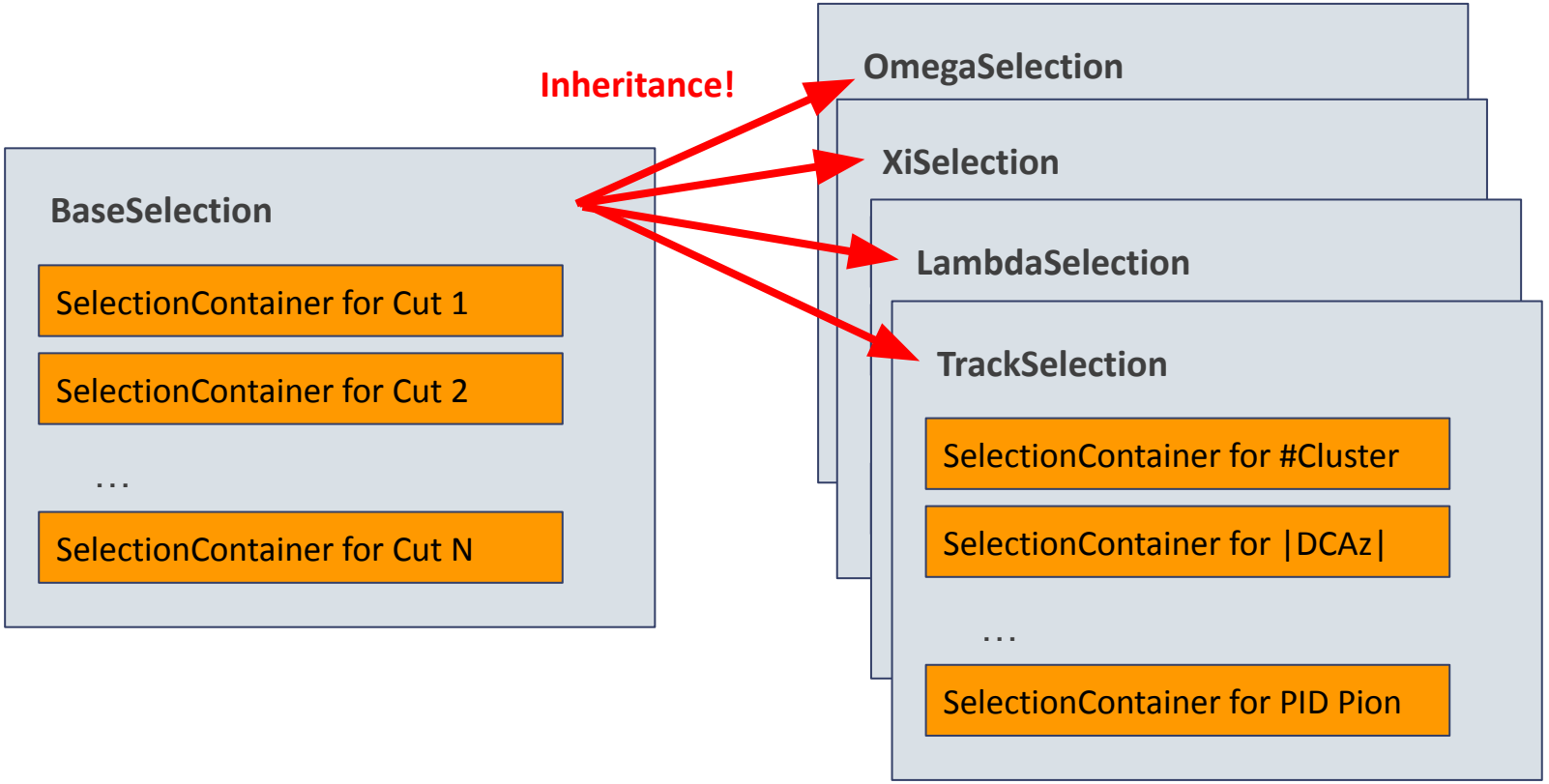
```
Observable: TPC+TOF Proton PID (index 32)
  Limit type           : Absolute Upper Limit
  Minimal cut          : no
  Skip most permissive : no
  Bitmask shift        : 1
  Selections           :
    3.000000            -> bitmask: 1

Observable: TPC Proton PID (index 18)
  Limit type           : Absolute Upper Limit
  Minimal cut          : no
  Skip most permissive : no
  Bitmask shift        : 1
  Selections           :
    3.000000            -> bitmask: 2

Observable: Max. |DCA_z| (cm) as a function of pT (index 7)
  Limit type           : Absolute Upper Function Limit
  Minimal cut          : yes
  Skip most permissive : yes
  Bitmask shift        : 0
  Selections           :
    0.004+0.013*TMath::Power(x,-1)-> loosest minimal selection, no bit saved
```

- Verbose output, mask for each cut is indicated here
- To compute the bitmask for a desired selections, just add up the numbers

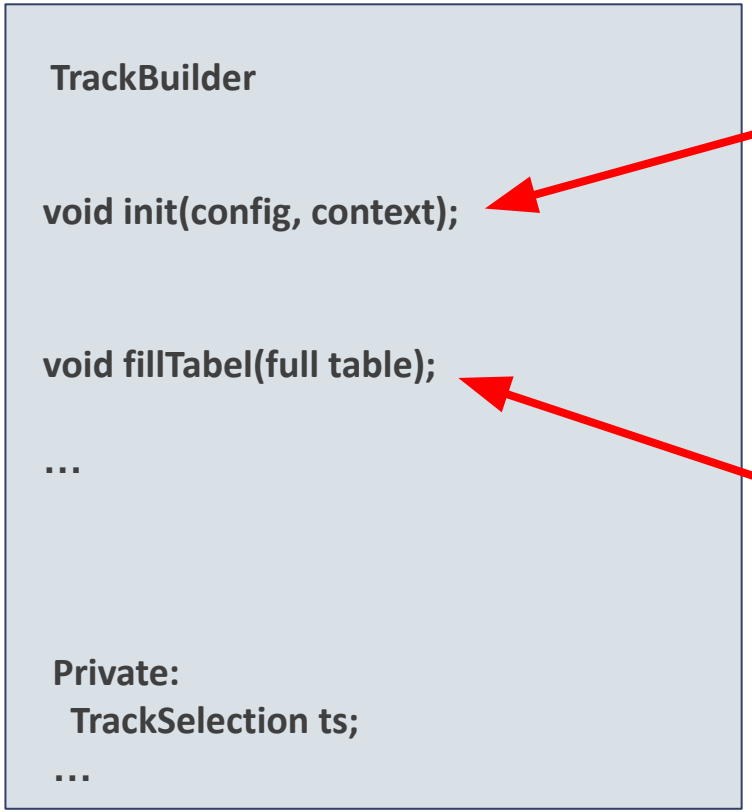
In example here, everything is a minimal cut (bit skip) except for PID, so

TPC PID Proton: Mask = 2
TPCTOF PID Proton: Mask=1

Bitmask for Tracks and more



Example: TrackBuilder



- Init function takes all configurable and the init context as input
  - From the init context the builder can deduce which tables are activated and only builds these tables (!process function needs to be correct!)
  - Configures the TrackSelection object, responsible for selection tracks and computing the bit mask
- fillTable function takes as input the original table and then fills the new femto table, depending on the configured selection and enables tables

FemtoProducer is fully modular

```
// track builder
trackbuilder::TrackBuilderProducts trackBuilderProducts;
trackbuilder::ConfTrackTables confTrackTables;
trackbuilder::TrackBuilder trackBuilder;
trackbuilder::ConfTrackBits confTrackBits;
trackbuilder::ConfTrackFilters confTrackFilters;
```

Declare

```
void init(InitContext& context)
{

  // collision selection
  collisionBuilder.init(confCollisionFilters, confCollisionBits, confCollisionRo

  // configure track builder
  trackBuilder.init(confTrackBits, confTrackFilters, confTrackTables, context);
```

Initialize

```
// Core implementation, parameterized by builders to call
template <modes::System system, typename T1, typename T2, typename T3, typename T4>
void processTracks(T1 const& col, T2 const& /* bcs*/, T3 const& tracks, T4 const& tracksWithItsPid)
{
  auto bc = col.template bc_as<T2>();
  collisionBuilder.buildCollision<system>(bc, col, tracks);
  if (!collisionBuilder.checkCollision(bc, col)) {
    return;
  }
  collisionBuilder.fillCollision<system>(collisionBuilderProducts, col);

  // tracks
  indexMapTracks.clear();
  trackBuilder.fillTracks(tracksWithItsPid, trackBuilderProducts, collisionBuilderProducts, indexMapTracks);
```

Process and build new tables

- Keep the logic for filling tables outside of the femtoProducer
- Filling of the tables for different particles is handled by dedicated builder class for each particle type
- This keeps the femtoProduer “small” and the class can be reused if needed
- Declare builder with dedicated ConfigurableGroups and ProducesGroups
- Initialize the Builder in init function
- Process and fill tables in process function

Table for V0s (Lambdas)

```
// table for basic lambda information
DECLARE_SOA_TABLE_STAGED_VERSIONED(FLambdas_001, "FLAMBDA", 1, //! femto lambdas
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId, // use sign to differentiate between lambda (
                                   femtobase::stored::SignedPt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass, // mass of the lambda/antilambda depending on the si
                                   femtov0s::PosDauId,
                                   femtov0s::NegDauId,
                                   femtobase::dynamic::Sign<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::Pt<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::P<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);
using FLambdas = FLambdas_001;
using StoredFLambdas = StoredFLambdas_001;
```

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

- **Use signed Pt of the lambda to determine whether it is Lambda (+1) or AntiLambda (-1)**
- **Daughter tracks are (always) stored in the track table and can be access via the index**
- Pt and other kinetic variables are still accessible via dynamic columns
- All selections for Lambda candidate (including track quality of daughters and PID) are stored in mask of the lambda). K0short hypothesis is rejected at producer level

Table for V0s (K0shorts)

```
// table for basic k0short information
DECLARE_SOA_TABLE_STAGED_VERSIONED(FK0shorts_001, "FK0SHORT", 1, //! femto k0shorts
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId,
                                   femtobase::stored::Pt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass,
                                   femtov0s::PosDauId,
                                   femtov0s::NegDauId,
                                   femtobase::dynamic::P<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);
using FK0shorts = FK0shorts_001;
using StoredFK0shorts = StoredFK0shorts_001;
```

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

- **No need for signed Pt since there is no antiparticle**
- **Other than that, has exactly the same structure as Lambda (mask has the same size)**
- Lambda hypothesis can be rejected at the producer level

4 int32 + 4 floats + 1 int16 = 34 Bytes (+ at most 2*28 Bytes if both daughters are not in the track table)

Table for Cascades (Xis)

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

```
DECLARE_SOA_TABLE_STAGED_VERSIONED(FXis_001, "FXI", 1, //! femto xis
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId,
                                   femtobase::stored::SignedPt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass,
                                   femtocascades::BachelorId,
                                   femtov0s::PosDauId,
                                   femtov0s::NegDauId,
                                   femtobase::dynamic::Sign<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::Pt<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::P<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);
using FXis = FXis_001;
```

- **Use signed Pt to differentiate between particle (Xi-) and antiparticle (Xi+)**
- **Add additional index for bachelor particle**
- This follows the same structure as the original strangeness table, where we do not store the index of the daughter lambda extended information about daughter lambda is stored in the extend cascade table
- Cuts on Xi, daughter lambda and all daughter tracks is stored in the mask of the Xi

Table for Cascades (Omegas)

```
DECLARE_SOA_TABLE_STAGED_VERSIONED(FOmegas_001, "FOMEGA", 1, //! femto omegas
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId,
                                   femtobase::stored::SignedPt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass,
                                   femtocascades::BachelorId,
                                   femtov0s::PosDauId,
                                   femtov0s::NegDauId,
                                   femtobase::dynamic::Sign<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::Pt<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::P<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);
using FOmegas = FOmegas_001;
```

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

5 int32 + 4 floats + 1 int16 = 38 Bytes (+ at most 3*28 Bytes if bachelor/daughters are not in the track table)

Table for TwoTrackResonances (Phi)

```
DECLARE_SOA_TABLE_STAGED_VERSIONED(FPhis_001, "FPHI", 1, //! femto phis
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId,
                                   femtobase::stored::Pt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass,
                                   femtotwotrackresonances::PosDauId,
                                   femtotwotrackresonances::NegDauId,
                                   femtobase::dynamic::P<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::Pt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);

using FPhis = FPhis_001;
```

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

- **In contrast to V0s, daughters with high pt need TOF information, so PID for daughters is no longer minimal cut and we need information about the momentum of the daughters**
- Store special bit about daughter momentum in the bitmask
- No topological cuts for resonance, so mask only contains selections on daughter properties

5 int32 + 4 floats = 36 Bytes (+ at most 3*28 Bytes if bachelor/daughters are not in the track table)

Table for TwoTrackResonances (Phi)

```cpp
template <typename T1, typename T2, typename T3, typename T4, typename T5, typename T6>
void fillResonance(T1& collisionProducts, T2& trackProducts, T3& resonanceProducts, T4 const& posDaughter, T4 const& negDaughter, T5& trackBuilder, T6& indexMap)
{

  mTwoTrackResonanceSelection.applySelections(posDaughter, negDaughter); // for resonances selection are only applied to daughter tracks
  if (!mTwoTrackResonanceSelection.hasTofAboveThreshold(posDaughter, negDaughter) || !mTwoTrackResonanceSelection.passesAllRequiredSelections()) {
    return;
  }
  mTwoTrackResonanceSelection.reconstructResonance(posDaughter, negDaughter);
  if (!mTwoTrackResonanceSelection.checkFilters() || !mTwoTrackResonanceSelection.checkHypothesis()) {
    return;
  }
  int64_t posDaughterIndex = 0;
  int64_t negDaughterIndex = 0;
  posDaughterIndex = trackBuilder.template getDaughterIndex<modes::Track::kResonanceDaughter>(posDaughter, trackProducts, collisionProducts, indexMap);
  negDaughterIndex = trackBuilder.template getDaughterIndex<modes::Track::kResonanceDaughter>(negDaughter, trackProducts, collisionProducts, indexMap);

  if constexpr (modes::isEqual(resoType, modes::TwoTrackResonance::kRho0)) {
    if (mProduceRho0s) {
      resonanceProducts.producedRhos(
        collisionProducts.producedCollision.lastIndex(),
        mTwoTrackResonanceSelection.getPt(),
        mTwoTrackResonanceSelection.getEta(),
        mTwoTrackResonanceSelection.getPhi(),
        mTwoTrackResonanceSelection.getMass(),
        posDaughterIndex,
        negDaughterIndex);
    }
    if (mProduceRho0Masks) {
      resonanceProducts.producedRhoMasks(mTwoTrackResonanceSelection.getBitmask());
    }
  }
```

- Pairing negative and positive tracks and check for resonance hypothesis
- Check track quality and PID of daughter tracks first, then reconstruct and check the mass. If all passes, add the resonance to the table

Table for TwoTrackResonances (K*0)

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

```
DECLARE_SOA_TABLE_STAGED_VERSIONED(FKstar0s_001, "FKSTAR0", 1, //! femto k0star
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId,
                                   femtobase::stored::SignedPt, //! +1 for k0star and -1 for k0starbar
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass,
                                   femtotwotrackresonances::PosDauId,
                                   femtotwotrackresonances::NegDauId,
                                   femtobase::dynamic::Sign<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::Pt<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::P<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);
using FKstar0s = FKstar0s_001;
```

- Compared to Phi and Rho, K*0 is not its own antiparticle
- Use same trick as before, store signedPt for particle (+1) and antiparticle (-1)

5 int32 + 4 floats = 36 Bytes (+ at most 3*28 Bytes if bachelor/daughters are not in the track table)

Table for Kinks (charged sigmas)

Staged -> table can be consumed and reproduced in same task
Versioned -> old derived data can be updated on the fly to the new version

```
// table for basic sigma minus information
DECLARE_SOA_TABLE_STAGED_VERSIONED(FSigmas_001, "FSIGMA", 1,
                                   o2::soa::Index<>,
                                   femtobase::stored::CollisionId, // use sign to differentiate between sigma mi
                                   femtobase::stored::SignedPt,
                                   femtobase::stored::Eta,
                                   femtobase::stored::Phi,
                                   femtobase::stored::Mass,
                                   femtokinks::ChaDauId,
                                   femtobase::dynamic::Sign<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::Pt<femtobase::stored::SignedPt>,
                                   femtobase::dynamic::P<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Px<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Py<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Pz<femtobase::stored::SignedPt, femtobase::stored::Eta>,
                                   femtobase::dynamic::Theta<femtobase::stored::Eta>);

using FSigmas = FSigmas_001;
```

- **Use signed Pt of the sigma to determine whether it is SigmaMinus (-1) or AntiSigmaMinus (+1)**
- **Daughter track is (always) stored in the track table and can be access via the index**
- Pt and other kinetic variables are still accessible via dynamic columns
- All selections for Sigma candidate (including track quality of daughters and PID) are stored in mask

Implemented by Henrik Fribert

Generice Qa tasks

```cpp
// setup tracks
trackbuilder::ConfTrackSelection1 trackSelections;
trackhistmanager::ConfTrackBinning1 confTrackBinning;
trackhistmanager::ConfTrackQaBinning1 confTrackQaBinning;
trackhistmanager::TrackHistManager<trackhistmanager::PrefixTrackQa, modes::Mode::kAnalysis_QA> trackHistManager;

Partition<Tracks> trackPartition = MAKE_TRACK_PARTITION(trackSelections);
Preslice<Tracks> perColReco = aod::femtobase::stored::collisionId;

HistogramRegistry hRegistry{"FemtoTrackQA", {}, OutputObjHandlingPolicy::AnalysisObject};

void init(InitContext&)
{
  // create a map for histogram specs
  auto colHistSpec = colhistmanager::makeColHistSpecMap(confCollisionBinning);
  colHistManager.init(&hRegistry, colHistSpec);
  auto trackHistSpec = trackhistmanager::makeTrackQaHistSpecMap(confTrackBinning, confTrackQaBinning);
  trackHistManager.init(&hRegistry, trackHistSpec, trackSelections.charge.value, confTrackQaBinning.momentumType.value);
};

void process(FilteredCollision const& col, Tracks const& /*tracks*/)
{
  colHistManager.fill(col);
  auto trackSlice = trackPartition->sliceByCached(femtobase::stored::collisionId, col.globalIndex(), cache);
  for (auto const& track : trackSlice) {
    trackHistManager.fill(track);
  }
}
```

Declare

Initialize

Process particle by particle

- Generate partition of particles and then fill them one by one

Generice Pair tasks

Declare

```
pairbuilder::PairTrackTrackBuilder<
  trackhistmanager::PrefixTrack1,
  trackhistmanager::PrefixTrack2,
  pairhistmanager::PrefixTrackTrackSe,
  pairhistmanager::PrefixTrackTrackMe,
  closepairrejection::PrefixTrackTrackSe,
  closepairrejection::PrefixTrackTrackMe,
  modes::Mode::kAnalysis>
  pairTrackTrackBuilder;
```

Initialize

```
pairTrackTrackBuilder.init(&hRegistry, trackSelections1, trackSelections2,
```

- Same goals as femtoProducer; keep logic in dedicate class and only declare, initalize and process inside the actuall task
- PairBuilder manages the single particle histograms, the pair histograms (so far only k* distributions), close pair rejection (here also generic class) and pair cleaner

Process same/mixed event and build particle pairs

```
void processSameEvent(FilteredCollision const& col, Tracks const& tracks)
{
  pairTrackTrackBuilder.processSameEvent(col, tracks, trackPartition1, trackPartition2, cache);
}
PROCESS_SWITCH(FemtoPairTrackTrack, processSameEvent, "Enable processing same event processing", true);
```

Generice Pair cleaner

Track Track cleaner
(trivial, does same as base class)

```cpp
class TrackTrackPairCleaner : public BasePairCleaner
{
 public:
  TrackTrackPairCleaner() = default;
  template <typename T>
  bool isCleanPair(const T& track1, const T& track2) const
  {
    return this->isCleanTrackPair(track1, track2);
  }
};
```

Base class

```cpp
class BasePairCleaner
{
 public:
  BasePairCleaner() = default;
  virtual ~BasePairCleaner() = default;

 protected:
  template <typename T1, typename T2>
  bool isCleanTrackPair(const T1& track1, const T2& track2) const
  {
    return track1.globalIndex() != track2.globalIndex();
  };
};
```

Track V0 cleaner
(a bit more involved, check index of the track vs V0 daughters)

```cpp
class TrackV0PairCleaner : public BasePairCleaner // also works for particles decaying into a positive
{
 public:
  TrackV0PairCleaner() = default;
  template <typename T1, typename T2, typename T3>
  bool isCleanPair(const T1& track, const T2& v0, const T3& /*trackTable */) const
  {
    auto posDaughter = v0.template posDau_as<T3>();
    auto negDaughter = v0.template negDau_as<T3>();
    return (this->isCleanTrackPair(posDaughter, track) && this->isCleanTrackPair(negDaughter, track));
  }
};
```

Base class is very simple, it only checks the global indices of two tracks

Generice Close Pair rejection

Track Track CPR
(does same as base CPR)

```cpp
template <const char* prefix>
class CloseTrackRejection
{
 public:
  CloseTrackRejection() = default;
  virtual ~CloseTrackRejection() = default;

 bool isClosePair() const
 {
   return std::hypot(mAverageDphistar / mDphistarMax, mDeta / mDetaMax) < 1.f;
 }
```

```cpp
template <const char* prefix>
class ClosePairRejectionTrackTrack
{
 public:

 private:
  CloseTrackRejection<prefix> mCtr;
  bool mIsActivated = true;
};
```

Track V0 CPR
(check whether the track and the same charged daughter are too close)

```cpp
template <typename T1, typename T2, typename T3>
void setPair(const T1& track, const T2& v0, const T3 /*trackTable*/)
{
  if (mChargeTrack > 0) {
    auto daughter = v0.template posDau_as<T3>();
    mCtr.compute(track, daughter);
  } else if (mChargeTrack < 0) {
    auto daughter = v0.template negDau_as<T3>();
    mCtr.compute(track, daughter);
  } else {
    LOG(fatal) << "CPR Track-V0: Sign of the track is 0!";
  }
}
```

Generic class which checks whether two tracks are too close or not (also does the histogramming)
No inheritance here, but class is initialized as a data member of the specialized CPR class for different particle types

Overview

Available:

- Implemented/updated format for collisions, tracks, V0s (Lambda & K0short), Resonances (Phi, Rho0, K*0) and cascades (Xi, Omega)
- New tables are modular and versioned
- Selection and histograming class for each particle type
- Qa task for each particle type
- Most generic Pairing tasks are implemented

Pending:

- Tables for MC are missing (there is a standard procedure on how to handle MC)
- Utility for computation of bitmask for different particle types
  - Verbose information is provided by the producer
  - At least for Qa task, traditional method of supplying cuts should work
- Second stage derived producer and event tagging (old CollisionMasker)

Hands-on Part:

- Part 1: Using the framework
  - Download the provided files and run the femtoProducer task by yourself!
  - Take your favourite particle type and configure and run the corresponding QA task
  - Take your favourite particle type and configure and run the corresponding pair task pairing it with a track
  - Run femtoproducer task standalone and produce derived data
  - Run QA/analysis task with derived data as input
  - Generate 2nd stage derived data for individual analysis
- Part 2: Adding to the framework
  - Download the provided files and implement step by step a simplified version of the pair track track task

backup