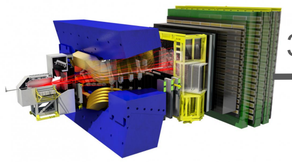
Introduction to selections in the Run 3 data flow: HLT2, sprucing and friends

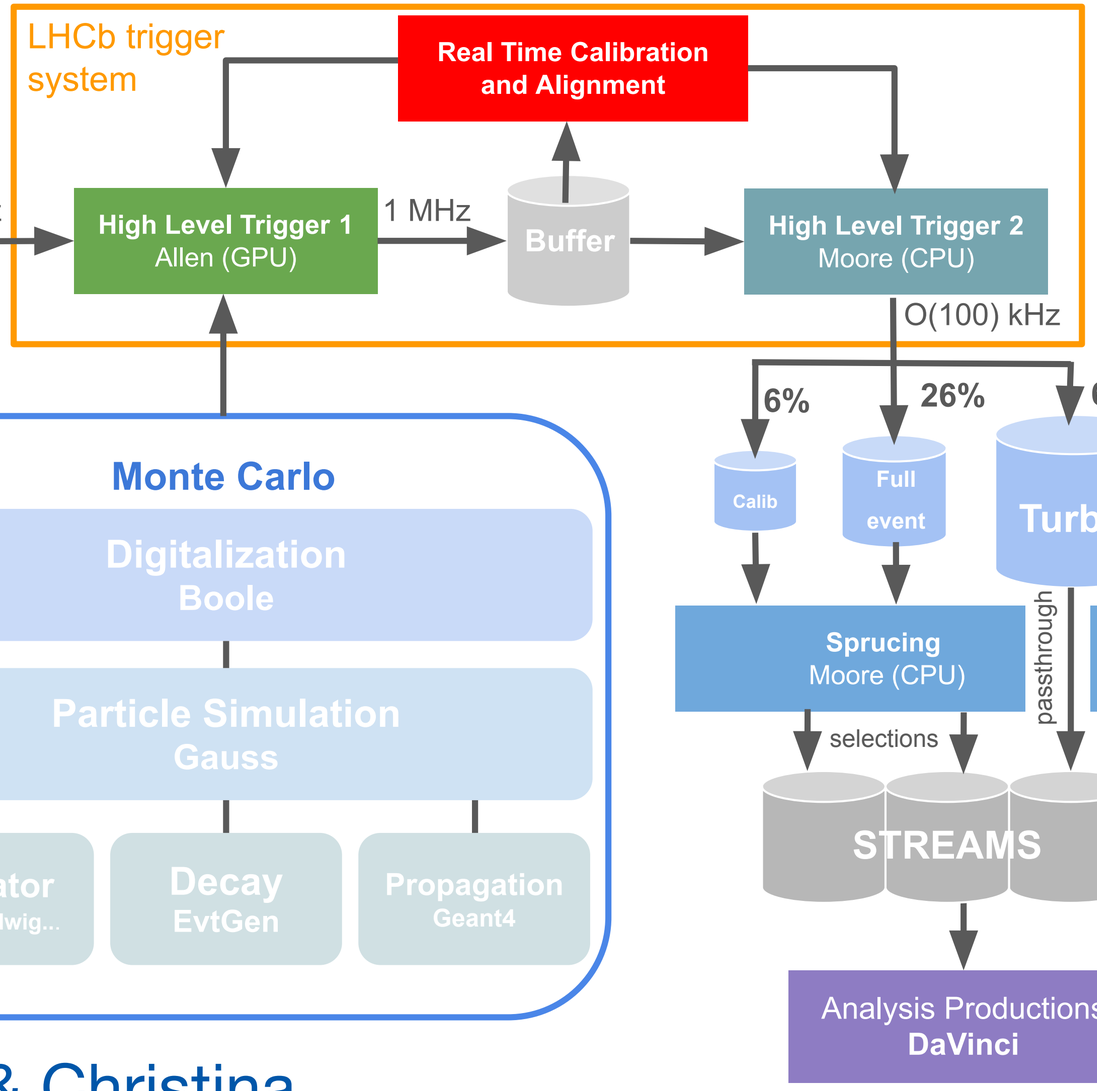
Lukas Calefice[1],[2]

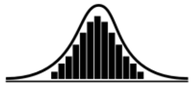
[1] Experimentelle Physik V, Technische Universität Dortmund
[2] Sorbonne Université/LPNHE, CNRS/IN2P3, Paris

LHCb Starterkit
26.11.2021

Recap: Run 3 data flow







No hardware trigger

→ triggerless detector readout at 30 MHz

→ fully software based

HLT1 (Allen)

→ has to run at 30MHz

→ partial event reco

→ HLT1 selection lines

HLT2 (Moore)

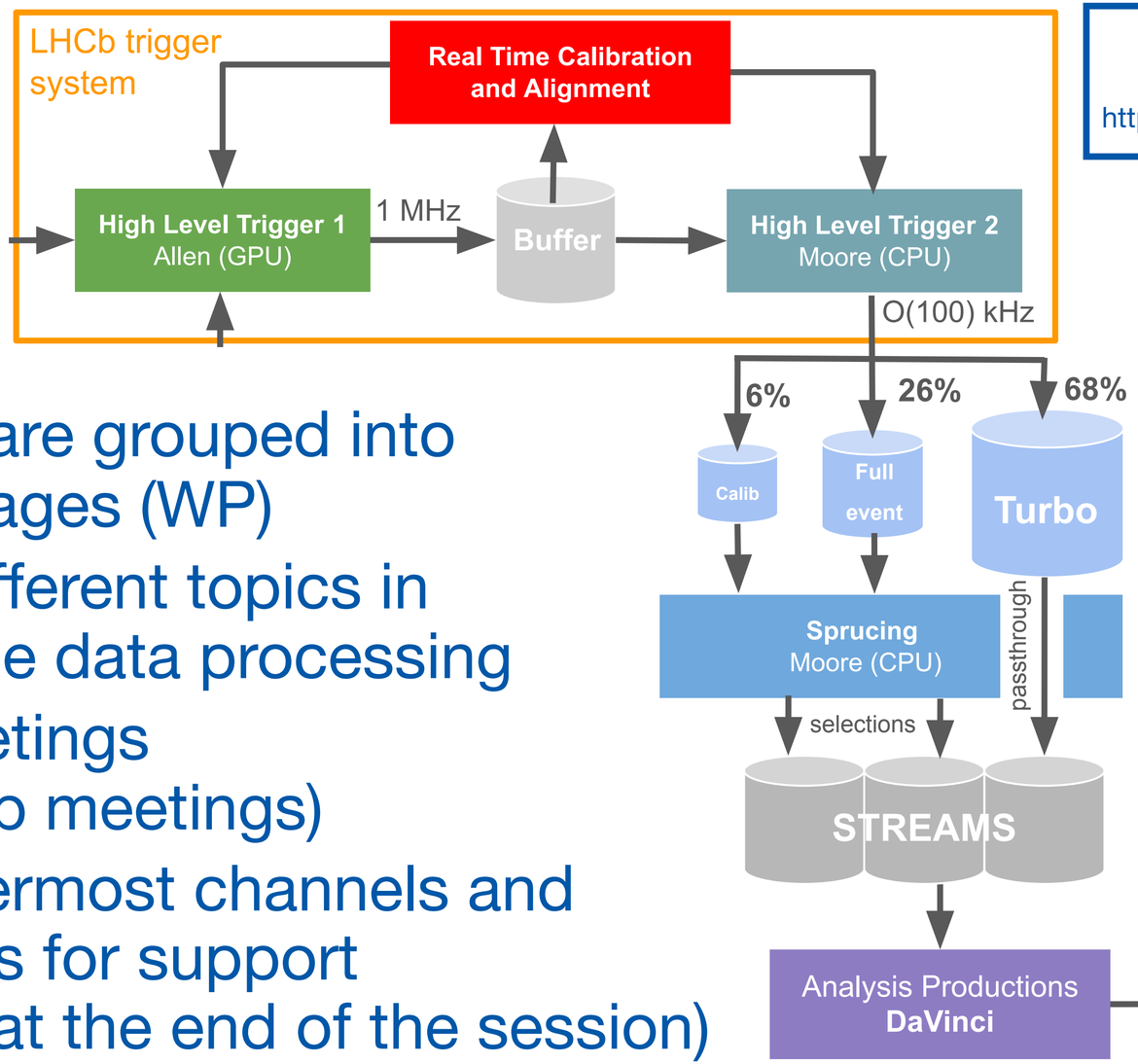
→ full event reco

→ HLT2 selections

→ different levels of persistence

Credits to Lera & Christina

RTA & DPA projects



Real-Time Analysis (RTA)

https://twiki.cern.ch/twiki/bin/view/LHCb/RealTimeAnalysis

Coordination

- PL & Deputy/ies
- IB chair (ex-officio)
- Work package coordinators

WP1 Data Structures	WP3 Selections	WP5 QA
WP2 Reconstruction	WP4 Align & Calib	WP6 R&D

Data Processing & Analysis (DPA)

https://lhcb-dpa.web.cern.ch/lhcb-dpa/

- WP1 - Sprucing
- WP2 - Analysis Productions
- WP3 - Offline Analysis Tools
- WP4 - Innovative Analysis Techniques
- WP5 - Legacy Software & Data
- WP6 - Analysis Preservation & Open Data

Both projects are grouped into **W**orking **P**ackages (WP)

→ handling different topics in online/offline data processing

→ regular meetings (check LHCb meetings)

→ many mattermost channels and mailing lists for support

(see overview at the end of the session)

What will be covered in this lesson?

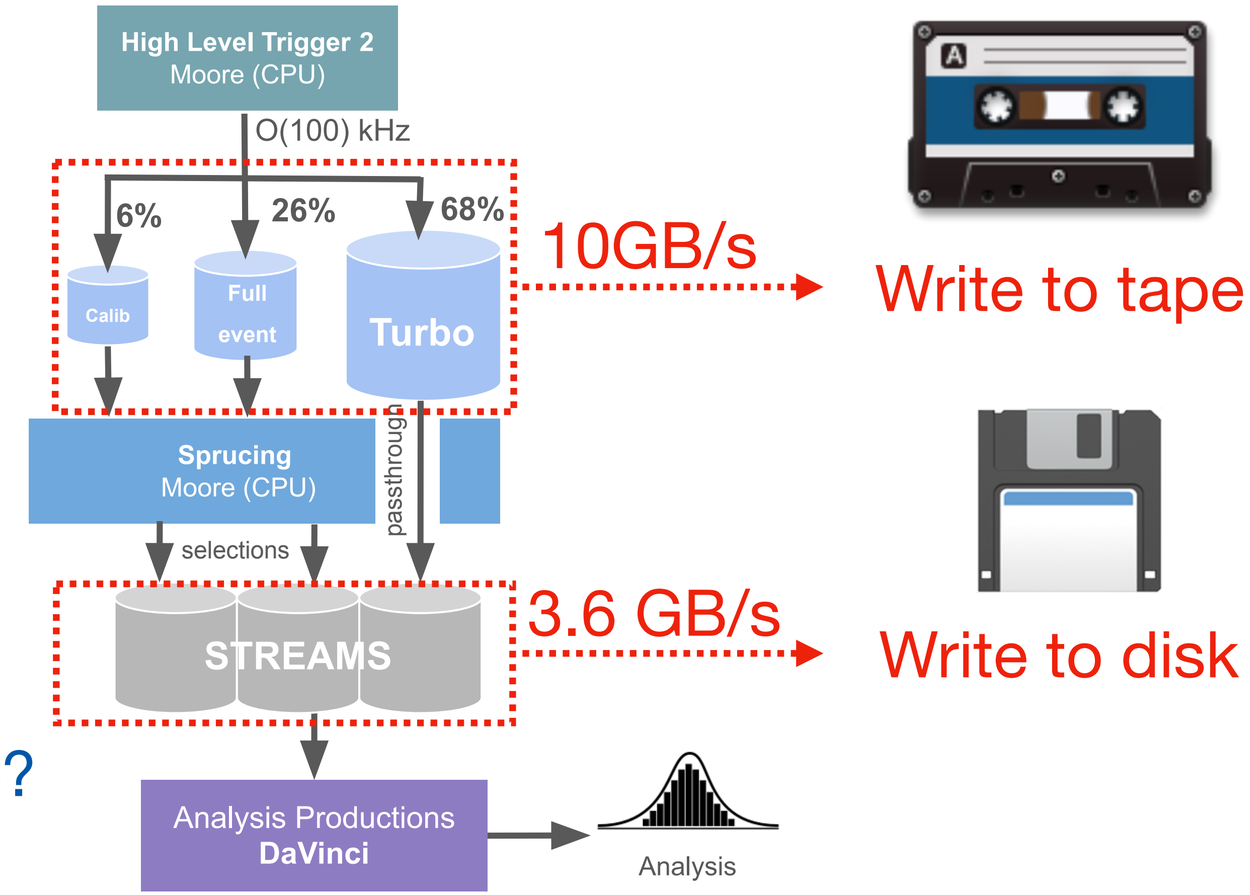
→ Talking about concepts in Run3 data flow & data flow specific jargon

→ Getting the basics on how to develop HTL2 lines

→ Understand the different between HTL2 and sprucing

→ What are ThOr functors and why do we need them?

→ What will DaVinci look like in Run 3?



Where to start?

- Moore documentation: https://lhcbdoc.web.cern.ch/lhcbdoc/moore/master/index.html
  → Writing HLT2 lines
  → Converting an HLT2 line to ThOr functors
  → Studying HLT2 efficiencies and rates
  → ThOr functors reference
- DPA documentation: https://lhcb-dpa.web.cern.ch/lhcb-dpa/#
  → WP1 - Sprucing
  → WP3 - Offline Analysis Tools
- FunTuple example: https://gitlab.cern.ch/lhcb/DaVinci/-/blob/master/DaVinciExamples/python/DaVinciExamples/tupling/example-tupling-basic.py
- Gitlab snippets: https://gitlab.cern.ch/explore/snippets

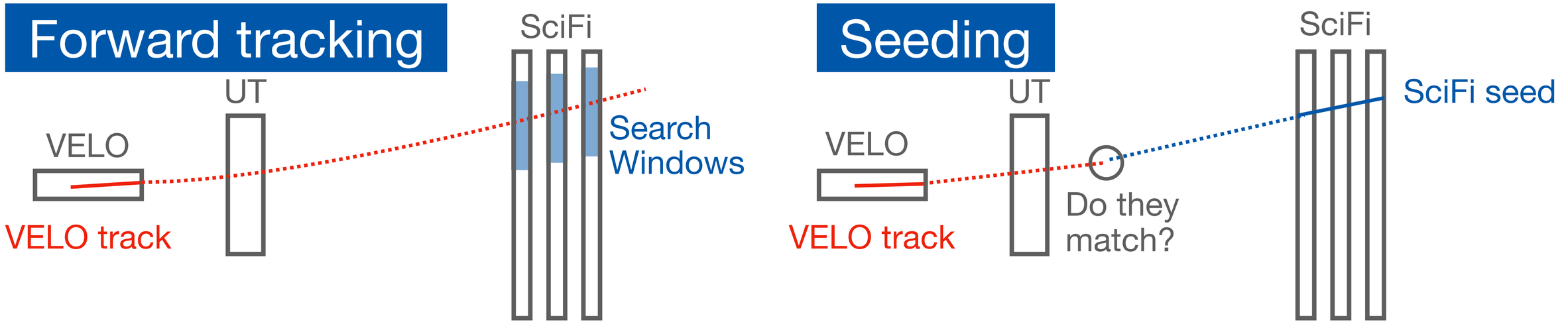
Reconstruction in HLT2

- Full event reconstruction will be run Moore
  - always online in HLT2
  - re-running offline possible if enough information is persisted
  - for Run 3 Brunel will be removed from the data flow

- Full vs. partial event reconstruction:
  - tracking algorithms are more advanced
  - ECAL clustering algorithms more advanced
  - RICH information fully available

Example: Tracking in HLT2

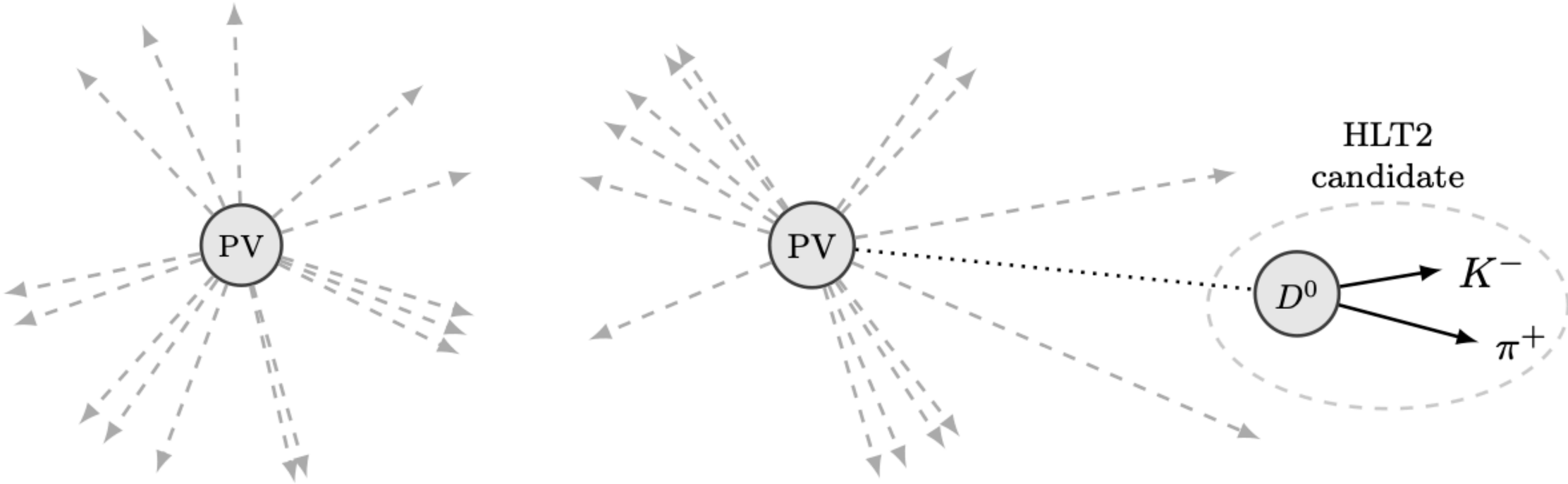
- Larger search windows in forward tracking with respect to HLT1
- Downstream tracks are reconstructed
- Reconstruct tracks that HLT1 failed to reconstruct
- Redundancy: increase tracking efficiency by a few percent



Persistence

Turbo model

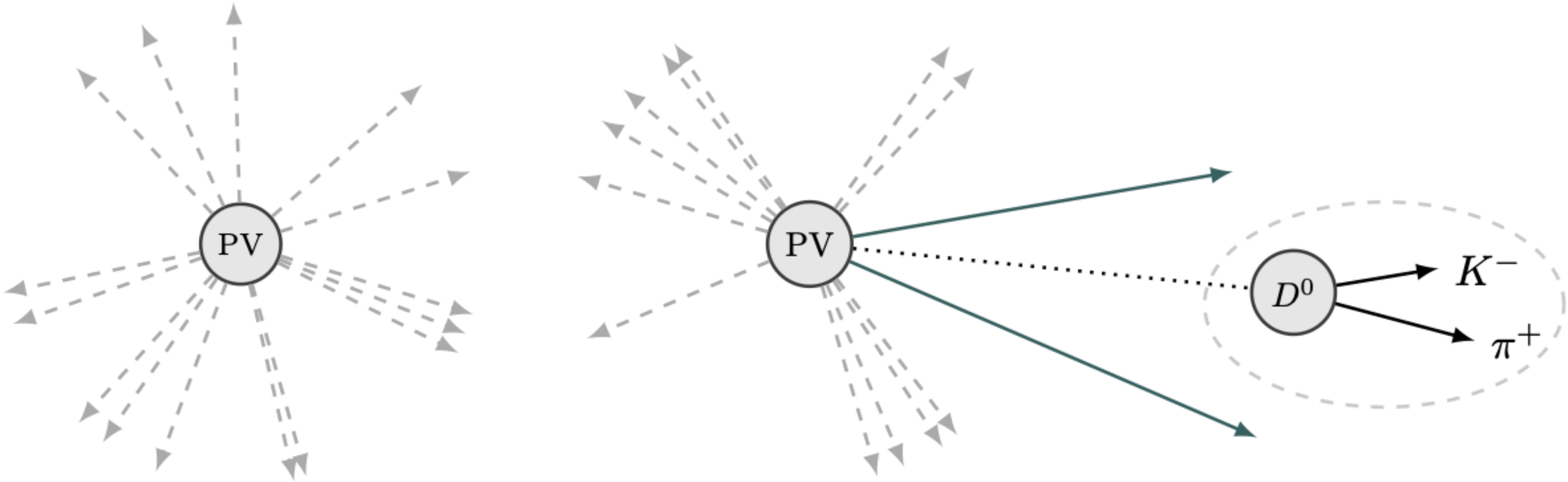
- Save raw and reconstructed data only from signal candidate that passed a HLT2 line selection



Sketch from arxiv:1903.01360

Selective persistence

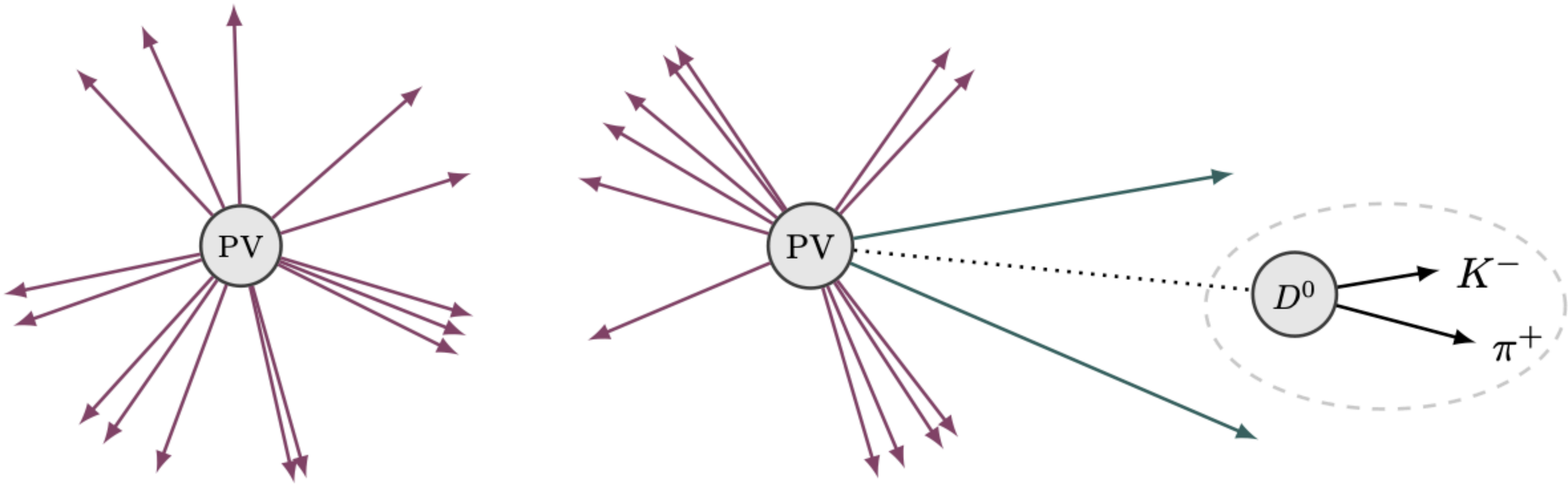
- Persist additional raw or reconstructed data *selectively*: e.g. save all reconstructed pion tracks



Sketch from arxiv:1903.01360

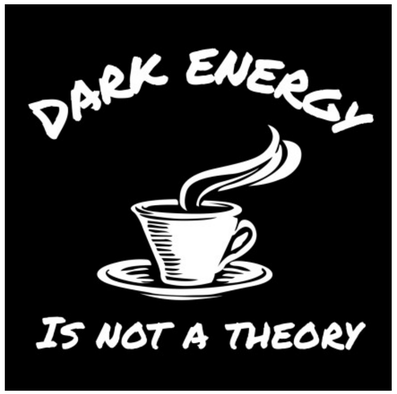
Full stream

- Persist raw and reconstructed data of the full event: enables to run the reconstruction fully offline



Sketch from arxiv:1903.01360

Persistence: What does it imply?



Turbo

Full stream

Persistence: What does it imply?

- Having extra information offline is of course very attracting! 🤩

BUT 🤔

- limited bandwidths for the data that we can write to tape (10GB/s, after HLT2) and disk (3.6GB/s, after Sprucing)
  → We cannot save the full information for every event
  → For full stream further selection needed in Sprucing
- more information = larger event size
  → O(10kB) for Turbo vs. O(100kB) for full stream
  → Smaller events: We can save more data!

Is it really relevant? vs. Do we have all necessary information?

Some examples

- Retraining of jet reconstruction algorithms
- Calibration of flavour tagging algorithms
- Training of isolation variables
- Optimization of calorimeter clustering algorithms
- Spectroscopy

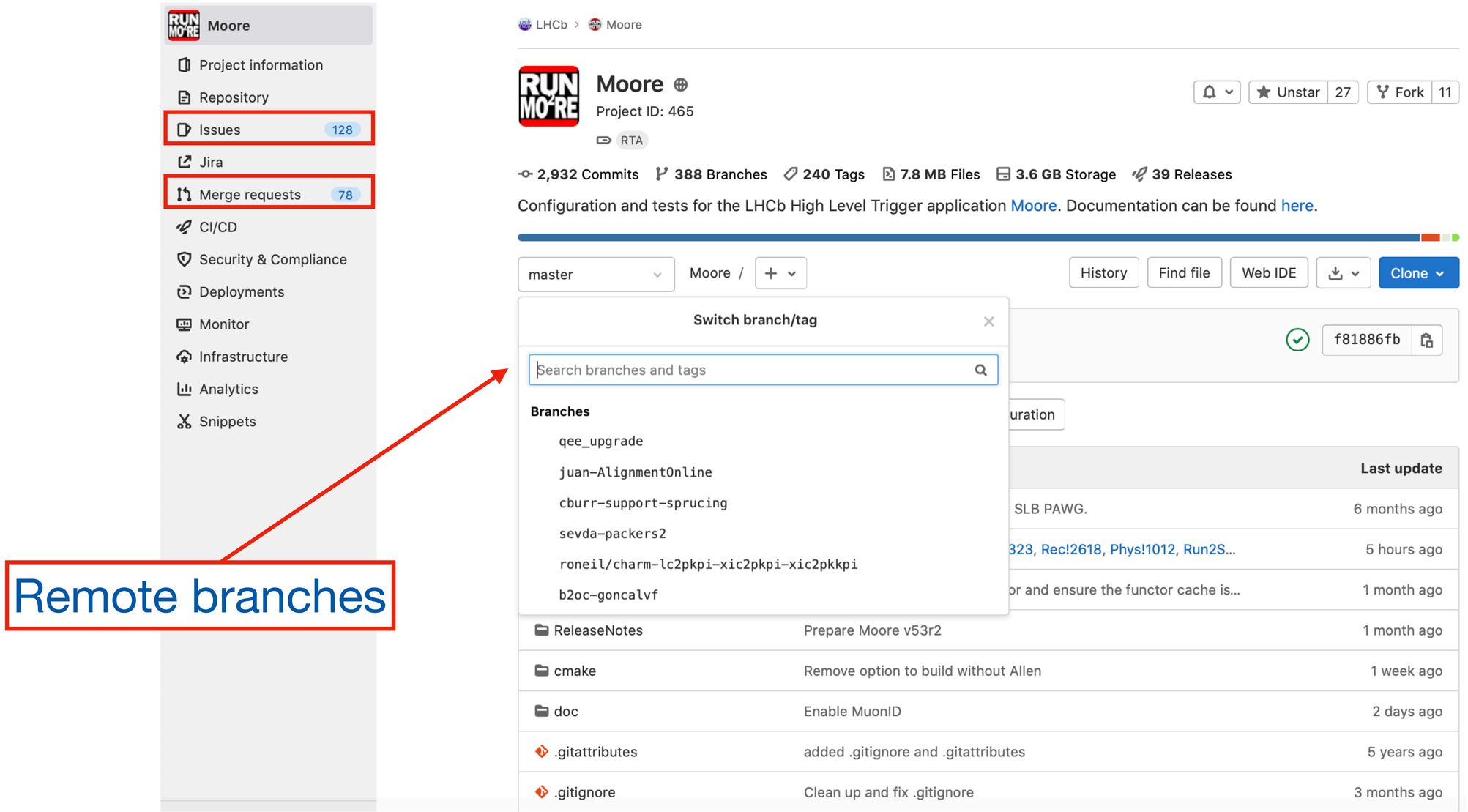
Developing HLT2 selections

Getting started

- Join upgrade HLT2 mattermost channel: https://mattermost.web.cern.ch/lhcb/channels/upgrade-hlt2
- Check if your Physics Analysis Working Group (PAWG) has extra communication channels and organisation
  - there is many Run3 “Migration Task Forces”
  - some WGs have dedicated mattermost channels
  - WG specific Moore branches
- Setup the LHCb stack and build Moore from there: https://gitlab.cern.ch/rmatev/lb-stack-setup#configuration-settings

Moore project

https://gitlab.cern.ch/lhcb/Moore

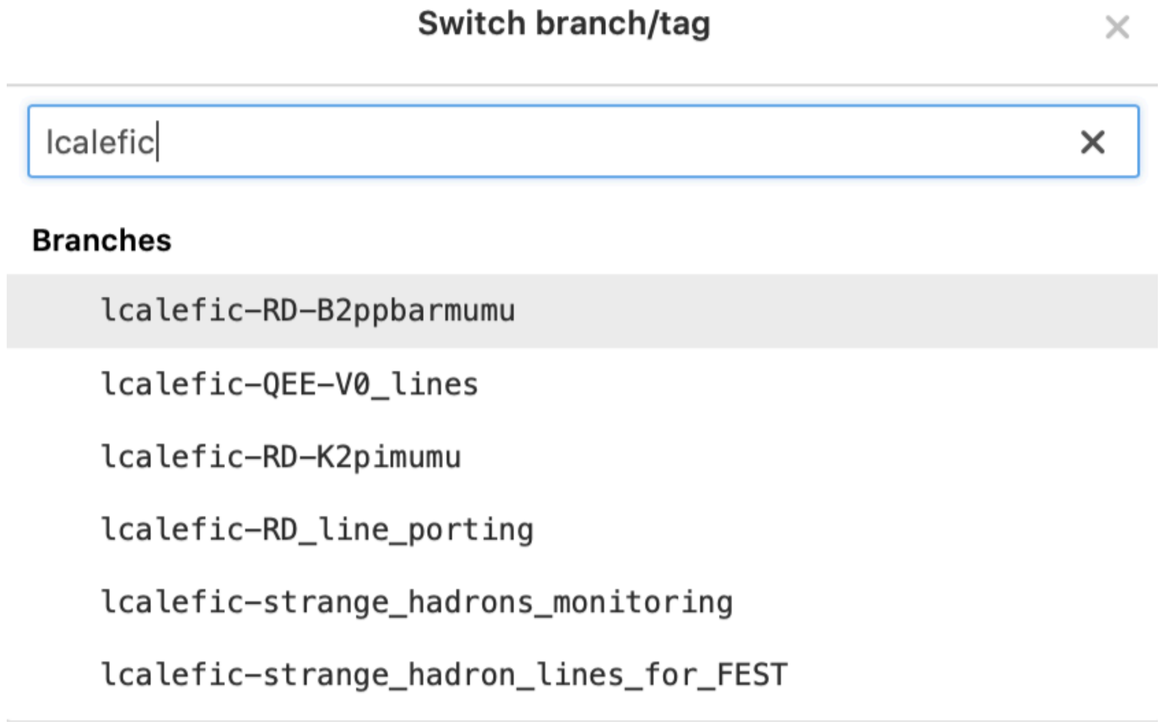


Remote branches

Moore project

https://gitlab.cern.ch/lhcb/Moore

- Create a new branch in Moore with your name and the name of the line(s)
- Once you have a v0 of the HLT2 line(s) open a Merge Request (MR) Draft

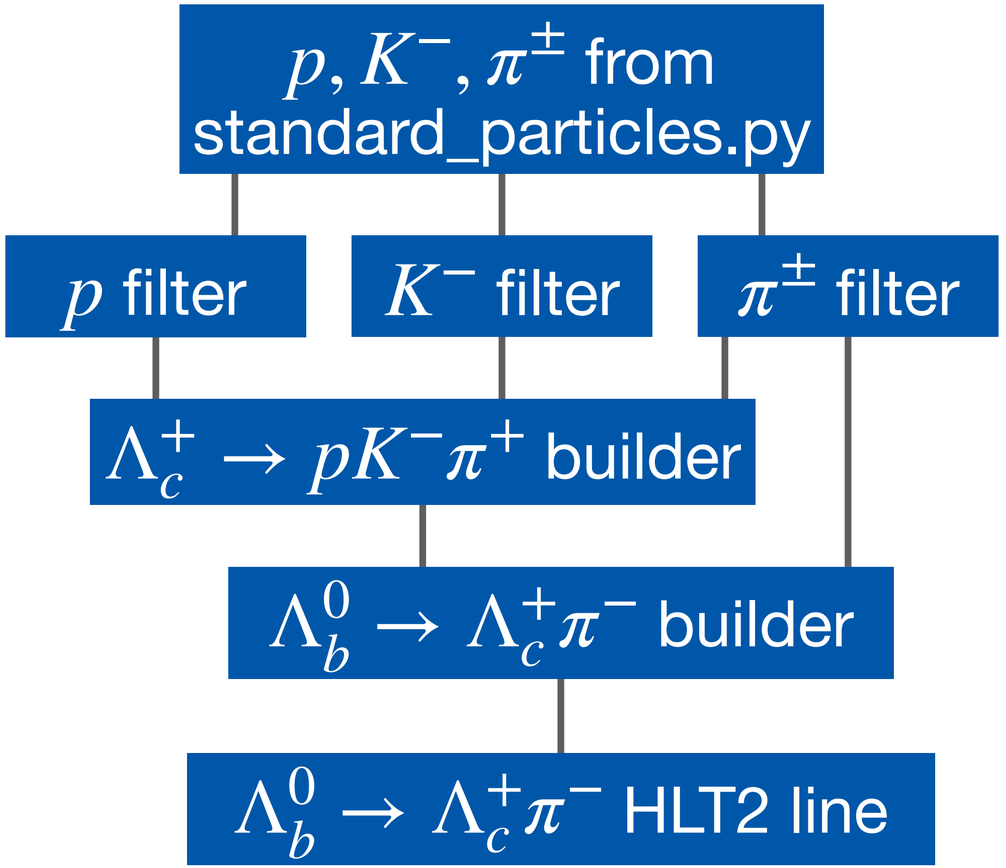


Structure of HLT2 lines

- We always start from final-state particles and filter them with selection requirements

$$\Lambda_b^0 \to \Lambda_c^+(\to pK^-\pi^+)\pi^-$$

- Create builders for intermediate particles by combining final-state particles
- Main builder of the decay mode
- Line declaration that calls all the builders and filters



Selection Algorithms

- Moore/Hlt/Hlt2Conf/python/Hlt2Conf/
  → algorithms.py and algorithms_thor.py
  → these contain the python configurations of **ParticleFilter**s and **ParticleCombiner**s
- At the moment Moore works as a hybrid of algorithm backends in python that are designed to work with LoKi or ThOr functors
  → Algorithms can be mixed within HLT2 lines
  → Functors cannot be mixed inside a filter or combiner

ThOr functors

ThOr functors

- Functors are function-like objects composable with other functors without knowing the input value
- LHCb selection framework separates configuration (in python) and execution (in C++) processes

  → configure selection requirements as complex functor compositions in a python options file (e.g in DaVinci or Moore)

  → functor implementation in C++

ThOr functors

- ThOr = Throughput Oriented
  - Goal is to be much faster than in Run 2 to match the timing constraints in HLT2!
  - profit from vectorised event model: Structure of Arrays (SoA) instead of AoS (Array of Structures), speeds up by x4 when using ThOr functors
- Conversion of LoKi functors to ThOr functors is ongoing!
  - right now we are in a transition period!
  - at some point LoKi functors will be fully removed for Run 3

ThOr functors

- For available ThOr functors check: https://lhcbdoc.web.cern.ch/lhcbdoc/moore/master/selection/thor_functors_reference.html
- Also Functor translation tables for conversion from LoKi: https://lhcbdoc.web.cern.ch/lhcbdoc/moore/master/selection/thor_functors.html#functor-translation-tables

`ETA`	`F.ETA`	✓
`ISMUON`	`F.ISMUON`	✓
`MIPCHI2DV(PRIMARY)`	`F.MINIPCHI2(pvs)`	✓
`MIPDV(PRIMARY)`	`F.MINIP(pvs)`	✓
`M`	`F.MASS`	✓

← Tables are intended to keep track of whether the behaviour of functors is validated

Contributions in validation and documentation are very welcome!

Missing functors

- Functor writing campaign: https://gitlab.cern.ch/lhcb-dpa/project/-/issues/61
- Writing ThOr functors: Talk by Niklas during 22nd hackathon

- [x] ProbNNx, MR lhcb/Rec!2471 (merged)
- [ ] Tagging decisions: @emgabrie is working on this.
- [ ] Functor that delegates to LoKi, issue lhcb/Rec#169
- [ ] `CHILD` : Need to think about how to implement this, issue lhcb/Rec#221
- [ ] `INTREE` , `MINTREE` , 'MAXTREE`... Need to think about how to implement this: lhcb/Rec#221
- [ ] `ADAMASS` @ngrieser is working on it, issue lhcb/Moore#289
- [x] Functors for DecayTreeFitted candidates, issue #117 (closed)
  - [ ] Version of DecayTreeFitterAlg that allows for vertex constraining https://gitlab.cern.ch/lhcb-dpa/project /-/issues/172
  - [ ] Access to decay tree fit chi2
  - [ ] Get DecayTreeFitterAlg to return the Fitter object
- [ ] `FOURMOMENTUM` , `PX` , `PY` , `PZ` , `PE` : MR lhcb/Rec!2619
- [ ] `HASMUON` and `INMUON` : lhcb/Rec#233
- [ ] `POINTINGMASS`
- [ ] Correlation matrices. Will be useful to test lhcb/Analysis!819 (merged).
- [ ] Trigger decisions
- [ ] One functor returning all trigger decisions in a map (useful for finding which line triggered the signal).
- [ ] MTDOCACHI2, issue lhcb/Moore#345.
- [ ] MC truth access. #145
  - [ ] BackgroundCategory lhcb/Rec!2621
- [ ] Functors accessing a refitted PV
- [ ] `ID` and `ABSID` : issue lhcb/Rec#215

Writing ThOr functors

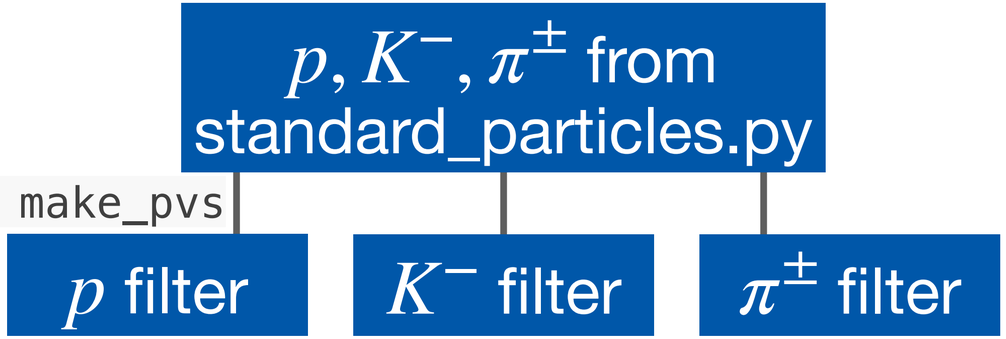
- Functor is implemented in C++ and has a python representation as front-end
- Example: F.PT
  - Rec/Phys/FunctorCore/python/Functors/__init__.py
  - Rec/Phys/FunctorCore/include/Functors/TrackLike.h

```
/** @brief Transverse momentum, as defined by the pt() accessor.
 */
struct TransverseMomentum : public Function {
  template <typename Data>
  auto operator()( Data const& d ) const {
    if constexpr ( Sel::Utils::has_tracklike_API<Data> )
      return d.pt( Sel::Utils::get_kinematic_state( d ) );
    else
      return Sel::Utils::deref_if_ptr( d ).pt();
  }
};
```

```
PT = Functor('PT', "Track::TransverseMomentum", "TrackLike.h",
             "Transverse momentum.")
```

Back to the line development

Line development - Filters




```python
from GaudiKernel.SystemOfUnits import GeV, MeV, mm
from RecoConf.reconstruction_objects import make_pvs_v2 as make_pvs
import Functors as F
from Functors.math import in_range
from ..algorithms_thor import (ParticleFilter,
                               ParticleCombiner,
                               require_all)
```

```python
def filter_protons(particles, pvs, pt_min=0.5 * GeV, mipchi2_min=9, dllp_min=5):
    cut = require_all(
        F.PT > pt_min,
        F.MINIPCHI2(pvs) > mipchi2_min,
        F.PID_P > dllp_min,
    )
    return ParticleFilter(particles, F.FILTER(cut))
```

require_all(**cuts*) [source]

Return a functor expression requiring all arguments.

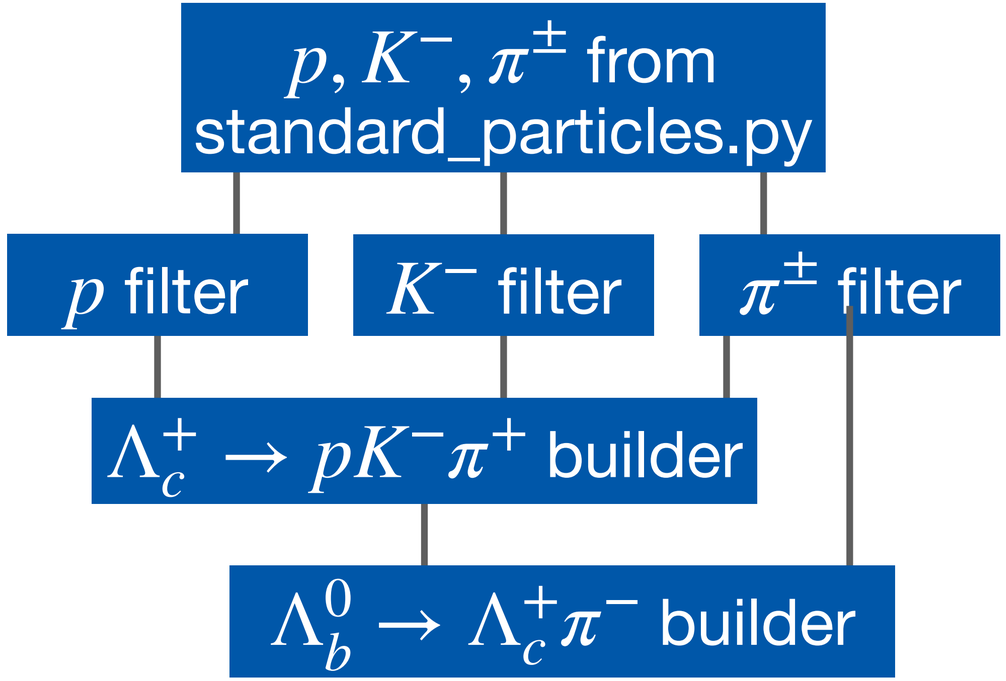
Example

```
>>> import Functors as F
>>> functor = require_all(F.PT > 1, F.PID_K < 0)
>>> functor.code_repr()
'( ( PT > 1 ) & ( PID_K < 0 ) )'
```

Line development - Builders

```
def make_lambdacs(protons,
                  kaons,
                  pions,
                  pvs,
                  two_body_comb_maxdocachi2=9.0,
                  comb_m_min=2080 * MeV,
                  comb_m_max=2480 * MeV,
                  comb_pt_min=2000 * MeV,
                  comb_maxdoca=0.1 * mm,
                  vchi2pdof_max=10,
                  bpvfdchi2_min=25):
    two_body_combination_code = F.MAXDOCACHI2CUT(two_body_comb_maxdocachi2)
    combination_code = require_all(
        in_range(comb_m_min,  F.MASS, comb_m_max),
        F.SUM(F.PT) > comb_pt_min,
        F.MAXDOCACUT(comb_maxdoca),
    )
    vertex_code = require_all(
        F.CHI2DOF < vchi2pdof_max,
        F.BPVFDCHI2(pvs) > bpvfdchi2_min,
    )
    return ParticleCombiner(
        [protons, kaons, pions],
        DecayDescriptor="[Lambda_c+ -> p+ K- pi+]cc",
        Combination12Cut=two_body_combination_code,
        CombinationCut=combination_code,
        CompositeCut=vertex_code,
    )
```

Renaming with respect to docs!



1-1 matching between the list of particles and the DecayDescriptor necessary!

For allowed particle names in Decay Descriptor: https://gitlab.cern.ch/lhcb-conddb/DDDB/-/blob/master/param/ParticleTable.txt

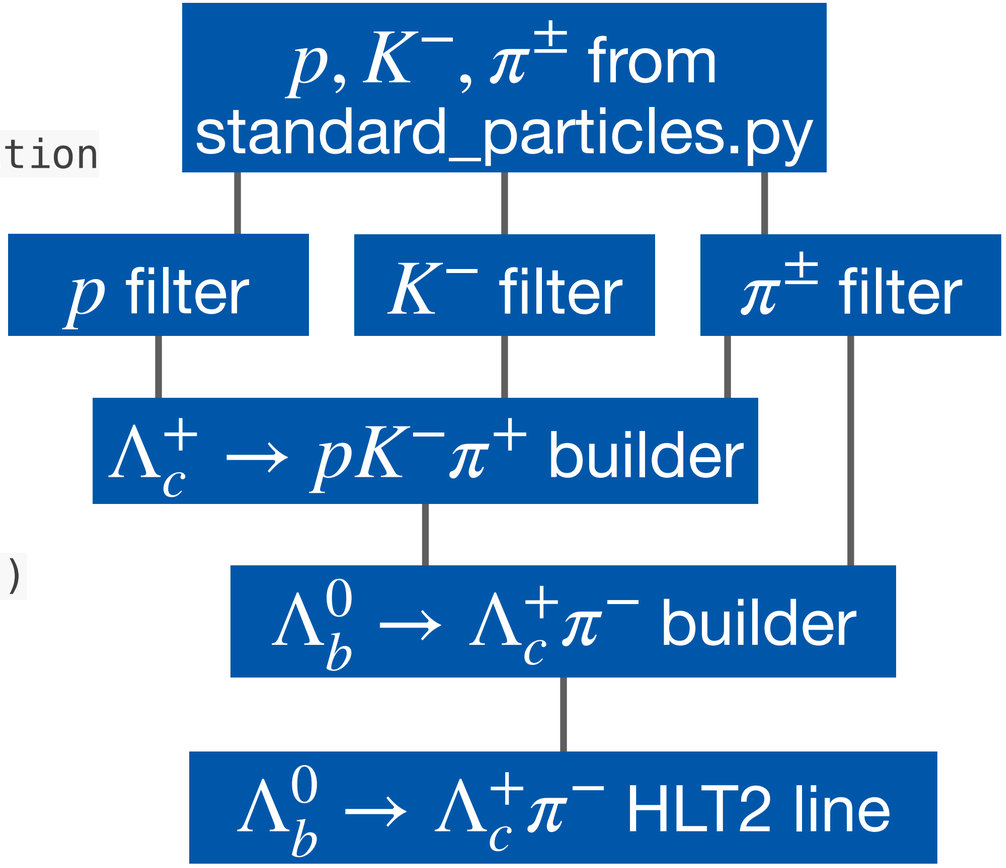
Line development - Line Declaration

```python
from Moore.config import register_line_builder
from Moore.lines import Hlt2Line
from RecoConf.reconstruction_objects import upfront_reconstruction

all_lines = {}

@register_line_builder(all_lines)
def lbtolcpi_lctopkpi_line(
             name="Hlt2LbToLcpPim_LcToPpKmPipLine",
             prescale=1):
    pvs = make_pvs()
    protons = filter_protons(make_has_rich_long_protons(), pvs)
    kaons = filter_kaons(make_has_rich_long_kaons(), pvs)
    pions = filter_pions(make_has_rich_long_pions(), pvs)
    lcs = make_lambdacs(protons, kaons, pions, pvs)
    lbs = make_lambdabs(lcs, pions, pvs)

    return Hlt2Line(
        name=name,
        algs=upfront_reconstruction()+[lbs],
        prescale=prescale,
     )
```



Prescale: fraction of randomly kept events that were triggered by this HLT2 line

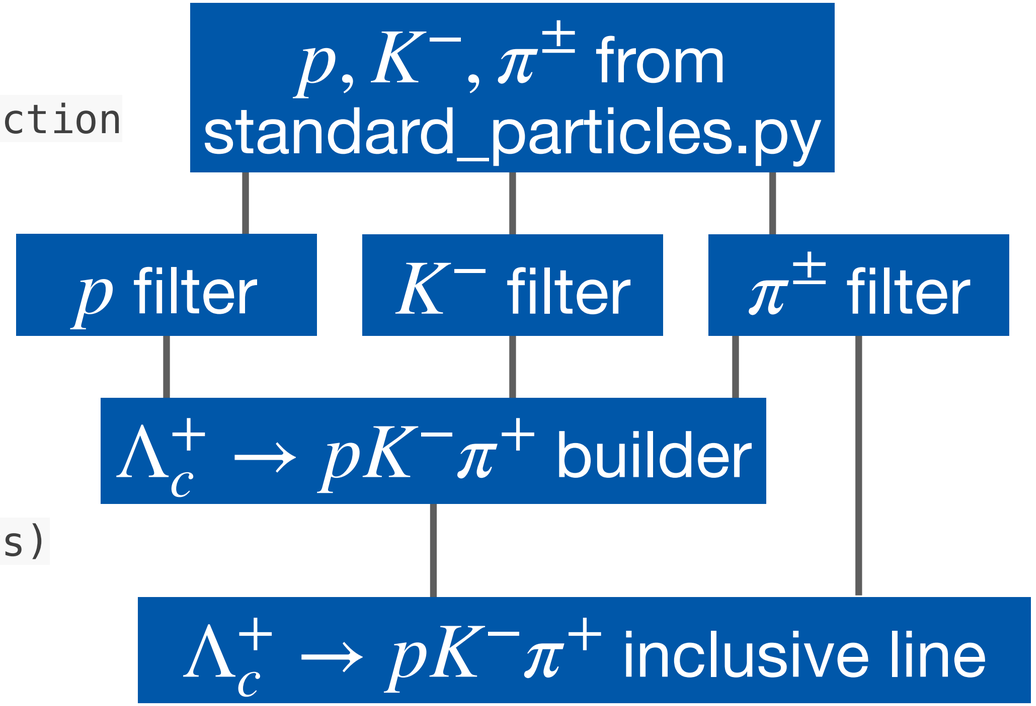
Line development - persistence

```python
from Moore.config import register_line_builder
from Moore.lines import Hlt2Line
from RecoConf.reconstruction_objects import upfront_reconstruction

all_lines = {}

@register_line_builder(all_lines)
def lbtolcpi_lctopkpi_line(
            name="Hlt2LbToLcpPim_LcToPpKmPipLine",
            prescale=1):
    pvs = make_pvs()
    protons = filter_protons(make_has_rich_long_protons(), pvs)
    kaons = filter_kaons(make_has_rich_long_kaons(), pvs)
    pions = filter_pions(make_has_rich_long_pions(), pvs)
    lcs = make_lambdacs(protons, kaons, pions, pvs)
    lbs = make_lambdabs(lcs, pions, pvs)

    return Hlt2Line(
        name=name,
        algs=upfront_reconstruction()+[lbs],
        prescale=prescale,
        extra_outputs = [("pions", pions)]
     )
```



Persisting all filtered pions!

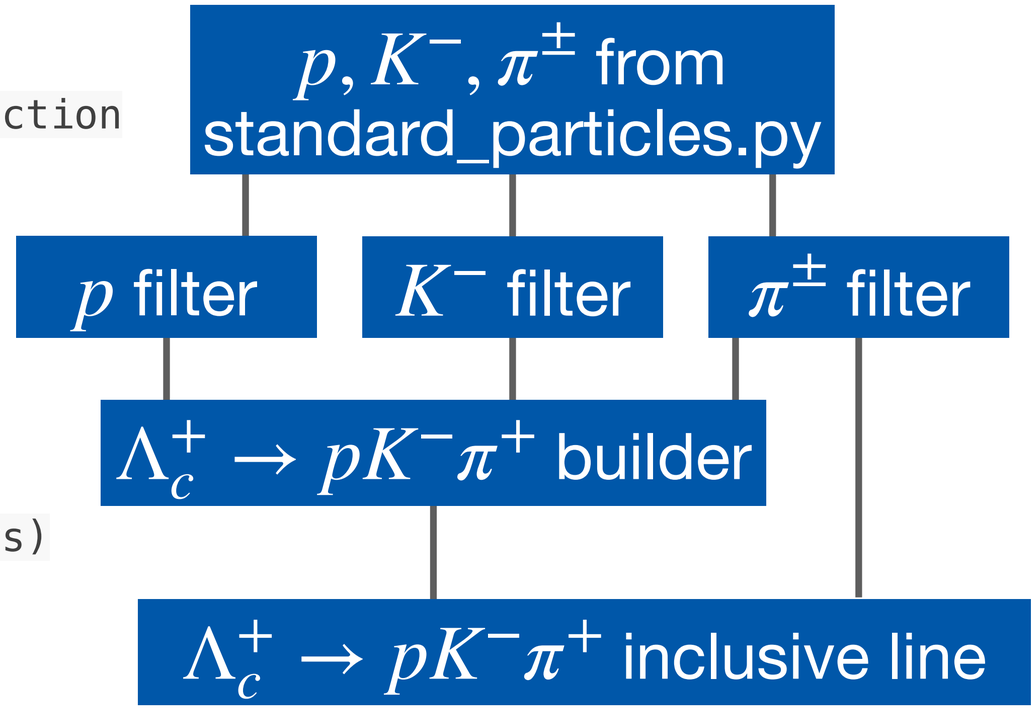
Line development - persistence

```python
from Moore.config import register_line_builder
from Moore.lines import Hlt2Line
from RecoConf.reconstruction_objects import upfront_reconstruction

all_lines = {}

@register_line_builder(all_lines)
def lbtolcpi_lctopkpi_line(
            name="Hlt2LbToLcpPim_LcToPpKmPipLine",
            prescale=1):
    pvs = make_pvs()
    protons = filter_protons(make_has_rich_long_protons(), pvs)
    kaons = filter_kaons(make_has_rich_long_kaons(), pvs)
    pions = filter_pions(make_has_rich_long_pions(), pvs)
    lcs = make_lambdacs(protons, kaons, pions, pvs)
    lbs = make_lambdabs(lcs, pions, pvs)

    return Hlt2Line(
        name=name,
        algs=upfront_reconstruction()+[lbs],
        prescale=prescale,
        persistreco=True
     )
```



Persisting the full raw
+ reconstructed data of the event!

Good practices - Common builders

- Use *common* builders and filters as much as possible
  → simplified maintenance, central bug fixing
  → avoid duplicated code

- *common* can be:
  → framework-wide, for very fundamental particles:
  Moore/Hlt/Hlt2Conf/python/Hlt2Conf/standard_particles.py
  e.g. final state particles, common intermediate states
  → WG or sub-WG specific
  e.g. Moore/Hlt/Hlt2Conf/python/Hlt2Conf/line/b_to_open_charm/builders/

Good practices - Common builders

- Very important to check if builder can fit your needs and does not bias your measurement later on
- Don’t hesitate to introduce new builders if you think there is no builder for your use case!
- Validation is very important!

Good practices - Naming conventions

https://gitlab.cern.ch/lhcb/Moore/-/issues/60

- Start with `Hlt{module}`, end with `Line`, and contain only alphanumeric characters and underscores (already enforced).
  - We currently do not have module structure, e.g. `CharmHad`, but will likely introduce one soon.
- The first letter of a particle name is always capitalised, the remaining are lowercase
  - `Jpsi` not `JPsi`
  - `Pi` not `pi`
- Separate parent and children in a decay with `To` (not `2`)
  - `D0ToKK`
- Particle charges can be specified for clarity
  - `D0ToKmPip` and `D0ToKpPim`
  - `DpToKmPiPip`
  - `BuToJpsiKp` or `BuToJpsiK`
- Cascade decays can be separated with underscores
  - `XiccppToLcpPip_LcpToppKmPip`
- 'Standard' decay modes can be left unspecified
  - `BsToJpsiPhi`, where `JpsiToMuMu` and `PhiToKK` are implied
  - `D0ToKS0PipPim`, where `KS0ToPipPim` is implied
- Qualifiers can be prepended to `Line`
  - `Detached`, `Incl`, `LL`

Decay Descriptors

1. Identical particle IDs *must* appear consecutively in the decay descriptor
   - OK: `D_s+ -> pi+ pi+ pi-`, `D_s+ -> pi- pi+ pi+`
   - Not OK: `D_s+ -> pi+ pi- pi+`
2. Identical particle IDs *must* have the same input container
   - OK: `[pions, pions, tight_pions]` for `D_s+ -> pi+ pi+ pi-`
   - Not OK: `[pions, tight_pions, tight_pions]` for `D_s+ -> pi+ pi+ pi-`
3. Objects *should* appear in order of rarity in the decay descriptor
   - Typically optimal: `B0 -> D0 pi+ pi-`
   - Typically sub-optimal: `B0 -> pi+ pi- D0`

Running Moore

- We need to create a Moore options file: hlt2_options.py

```
from Moore import options, run_moore
from RecoConf.global_tools import stateProvider_with_simplified_geom
from RecoConf.reconstruction_objects import reconstruction
public_tools = [stateProvider_with_simplified_geom()]

from Hlt2Conf.lines import lbtolcpi_lctopkpi_line

def all_lines():
    return [lbtolcpi_lctopkpi_line()]

options.input_raw_format = 0.3
options.input_file = config['input_files']
options.evt_max = n_events
options.simulation = simulation
options.output_type = "ROOT"

options.conddb_tag = config['conddb_tag']
options.dddb_tag = config['dddb_tag']

options.output_file = f"/eos/lhcb/user/l/lcalefic/hlt2_{decay_mode}_{n_events}.dst"
with reconstruction.bind(from_file=False):
    config = run_moore(options, all_lines, public_tools)
```

Can put configuration into yaml file

Tell Moore to run the reconstruction!

Running Moore

```
./Moore/run gaudirun.py ‘$MOOREROOT/tests/options/default_input_and_conds_hlt2.py' hlt2_options.py
```

- Which input files to use?
  - upgrade signal MC for line development and efficiencies

  → check the bookkeeping, do a MC request

  → for new files you will have a XDIGI (no reconstruction)

  → for old files there can be (L)DST (usually outdated reconstruction, change input raw format to 4.3)

  - for testing the rates of HLT2 lines we use MinBias MC

  → already in testfile database

```
options.set_input_and_conds_from_testfiledb('Upgrade_MinBias_LDST')
```

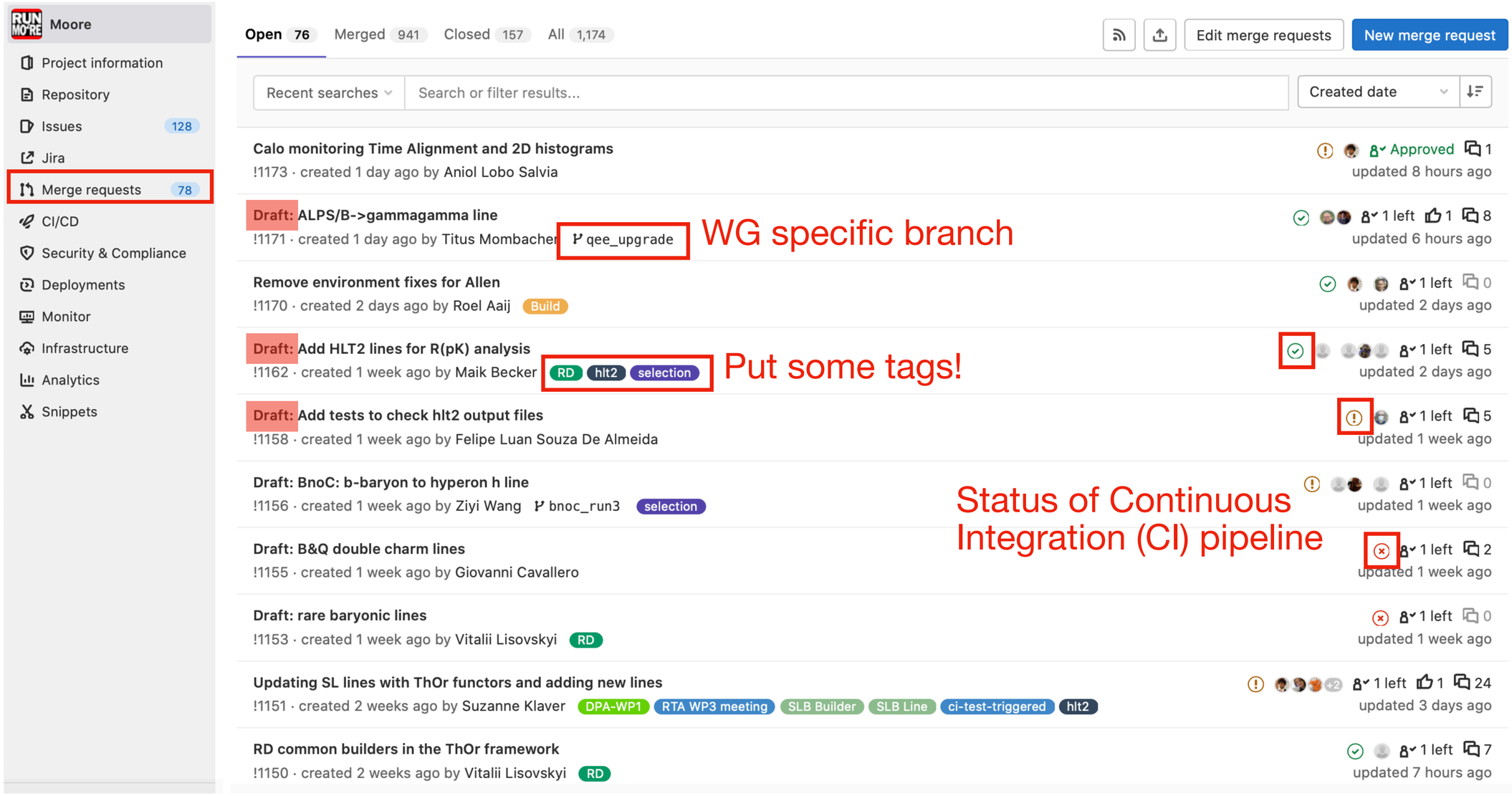
HLT efficiency checker

- A tool for you to easily run HLT lines (also HLT1) and calculate their rates and efficiencies
- Configuration via yaml files
- Provides plotting functionality: e.g. efficiency as a function of typical kinematic variables like p_{T} or η
- In order to use it you will need to build the MooreAnalysis project inside the stack: https://gitlab.cern.ch/lhcb/MooreAnalysis
- Check also: https://lhcbdoc.web.cern.ch/lhcbdoc/moore/master/tutorials/hltefficiencychecker.html

Open a Merge Request

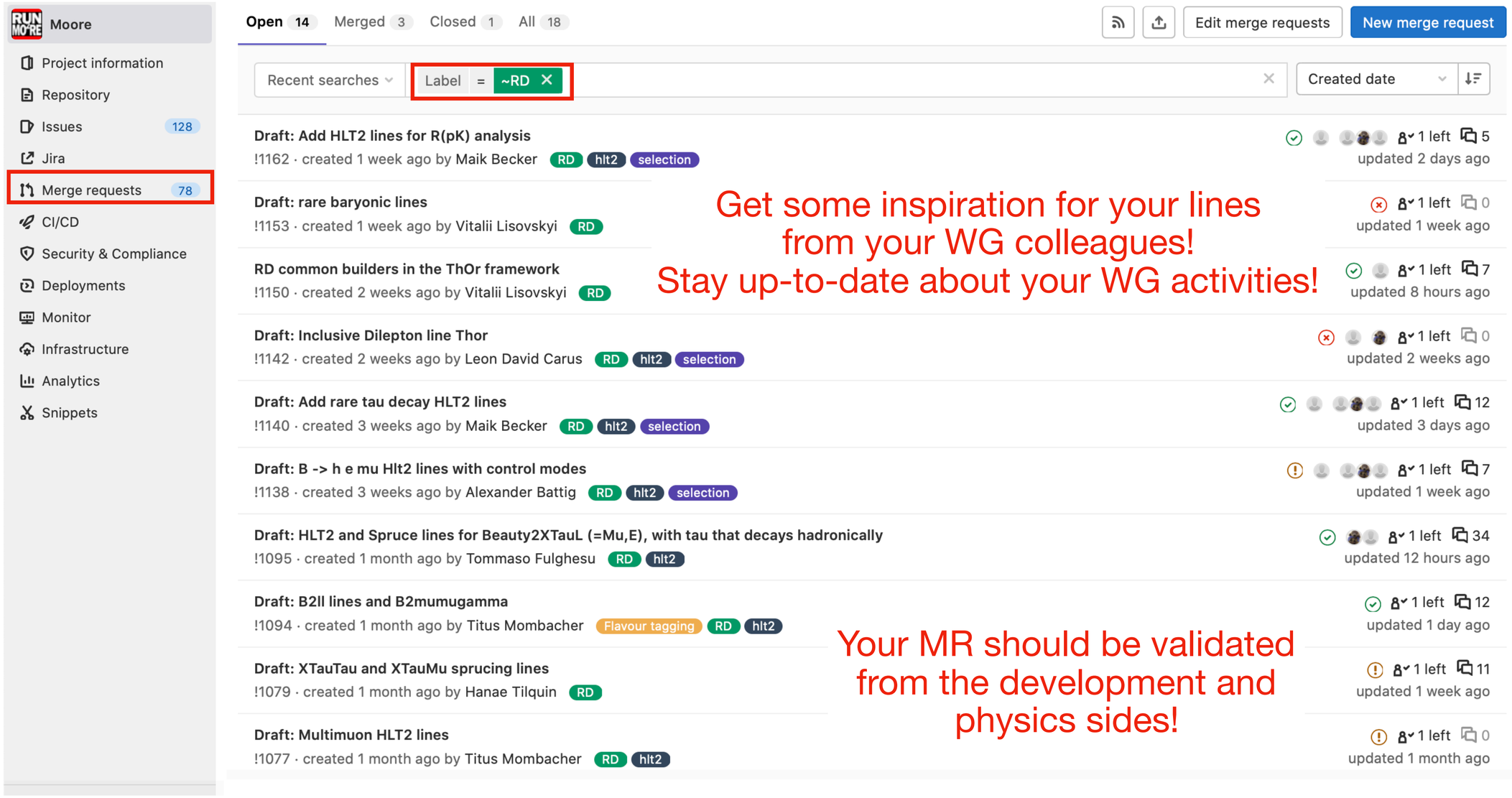
Merge requests

https://gitlab.cern.ch/lhcb/Moore



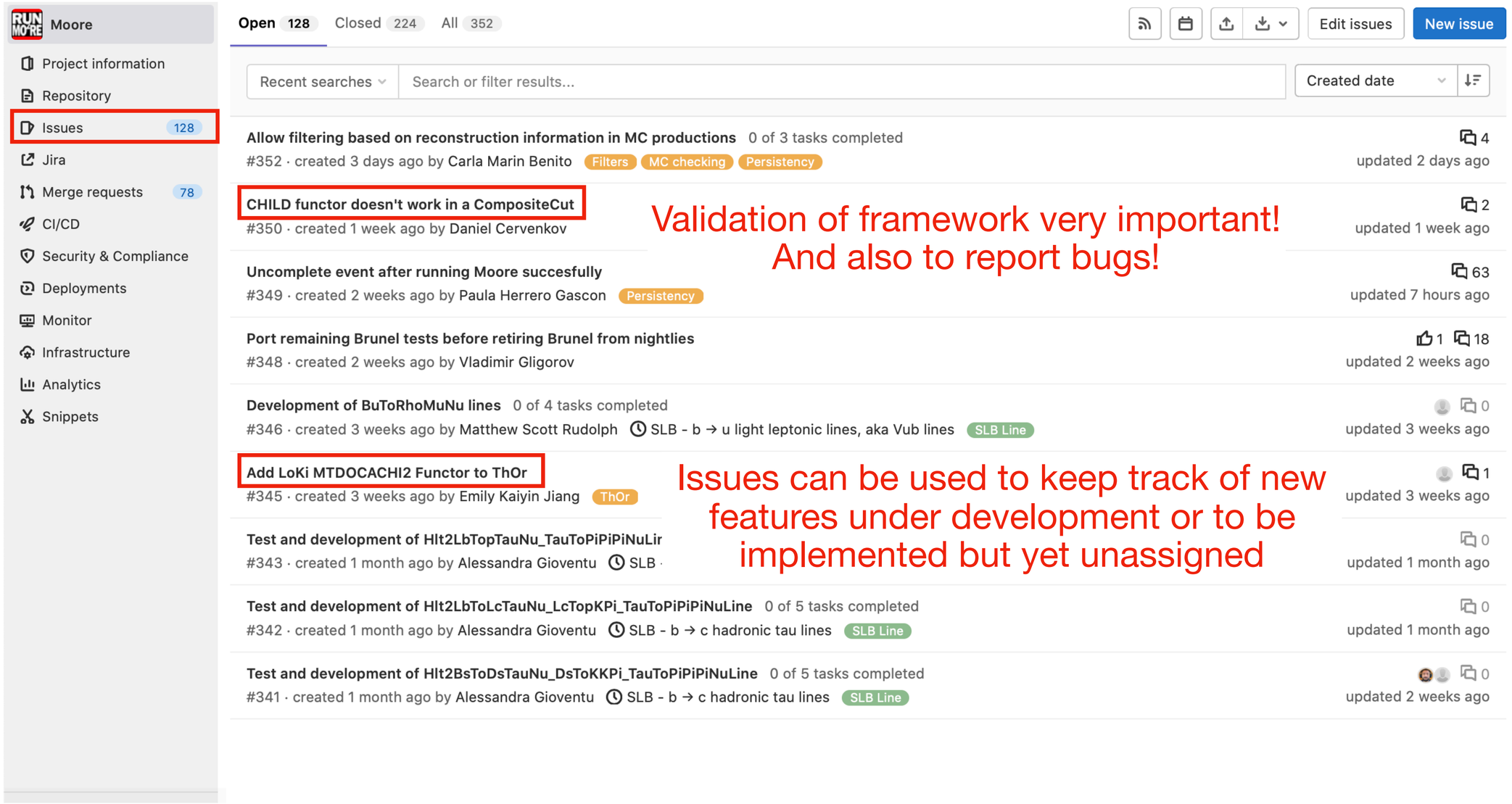
Merge requests

https://gitlab.cern.ch/lhcb/Moore



Issues

https://gitlab.cern.ch/lhcb/Moore

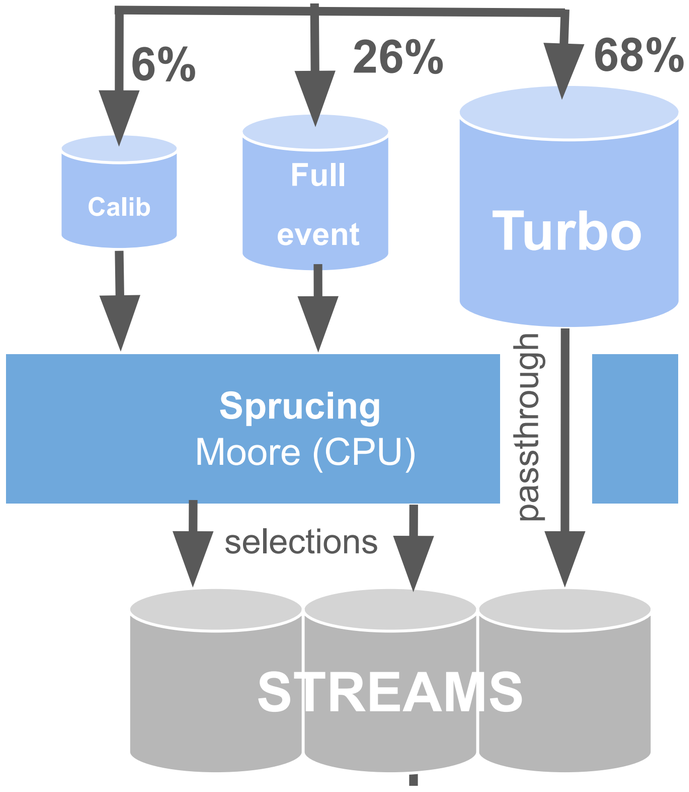


Sprucing

Sprucing

https://lhcb-dpa.web.cern.ch/lhcb-dpa/wp1/index.html

- Offline selection to further reduce amount of data that will be stored on disk when throughput of HLT2 is too large
- Similar task was performed by the Stripping before
- Turbo data will be running in passthrough mode
- Sprucing selections are defined inside Moore in the same way as HLT2 selections



Sprucing

https://lhcb-dpa.web.cern.ch/lhcb-dpa/wp1/index.html

- Sprucing and HLT2 in the same selection framework
  - As before: Maintenance much easier!
  - More flexibility in designing selections for the data taking

  → lines can easily be interchanged between HLT2 and Sprucing

  → instead of declaring a“HLT2Line” you will do a “SprucingLine”
- These are very beneficial concepts for the commissioning period of the trigger system!
- https://mattermost.web.cern.ch/lhcb/channels/dpa-wp1-sprucingtesla

Sprucing: How can it work?

Example from the B2OpenCharm (B2OC) WG

```python
@check_process
def make_BdToDsmK_DsmToHHH(process):
    if process == 'spruce':
        kaon = basic_builder.make_tight_kaons(k_pidk_min=None)
        d = d_builder.make_dsplus_to_hhh(pi_pidk_max=None, k_pidk_min=None)
    elif process == 'hlt2':
        kaon = basic_builder.make_tight_kaons()
        d = d_builder.make_dsplus_to_hhh()
    line_alg = b_builder.make_b2x(
        particles=[d, kaon], descriptors=['[B0 -> D_s- K+]cc'])
    return line_alg
```

spruce_b2oc.py

```python
PROCESS = 'spruce'
sprucing_lines = {}

##########################################
# From the BToDh_Builder
##########################################


@register_line_builder(sprucing_lines)
def BdToDsmK_DsmToHHH_sprucing_line(name='SpruceB2OC_BdToDsmK_DsmToHHH_Line',
                                    prescale=1):
    line_alg = b_to_dh.make_BdToDsmK_DsmToHHH(process=PROCESS)
    return SpruceLine(
        name=name,
        prescale=prescale,
        algs=prefilters.b2oc_prefilters() + [line_alg])
```

hlt2_b2oc.py

```python
PROCESS = 'hlt2'
hlt2_lines = {}
'''
book bare line from b_to_dh.py as hlt2 lines
'''


@register_line_builder(hlt2_lines)
def BdToDsmK_DsmToHHH_hlt2_line(name='Hlt2B2OC_BdToDsmK_DsmToHHH_Line',
                                prescale=1):
    line_alg = b_to_dh.make_BdToDsmK_DsmToHHH(process=PROCESS)
    return HltLine(
        name=name,
        prescale=prescale,
        algs=prefilters.b2oc_prefilters() + [line_alg])
```

DaVinci in Run 3

DaVinci in Run 3

- Run 2: Large processing times and huge amount of storage
  → many unused variables that were shipped by “some” TupleTool
  → sometimes you might even wonder what a variable is for
  → in practise: reading subsets of branches or removing unnecessary variables by pruning the tuples to have faster reading and processing of the relevant data
- In Run 3 we will have way more data!
- Compatibility with online selection framework desirable
  → enable ThOr functors in offline tupling

FunTuple

- Gives back the power over variables to the analyst!
- Customize collection of functors for each particle in the DecayDescriptor
- At the moment both LoKi and ThOr functors available
- TupleTools are under development

```
#FunTuple: make collection of functors for Jpsi
variables_jpsi = FunctorCollection({
    'LOKI_P':'P',
    'LOKI_MAXPT':'TRACK_MAX_PT',
    'LOKI_NTRCKS_ABV_THRSHLD':'NINTREE(ISBASIC & (PT > 15*MeV))',
    'LOKI_Muonp_PT':'CHILD(PT, 1)',
    'LOKI_Muonm_PT':'CHILD(PT, 2)',
    'THOR_PT':F.PT
})
```

Martina's talk during the 101st LHCb week
Davide's talk during the 23rd hackathon
FunTuple tupling example
WP3 section in DPA docs

Contributions are very welcome!

Stay tuned for upcoming training events

&

enjoy LHCb!