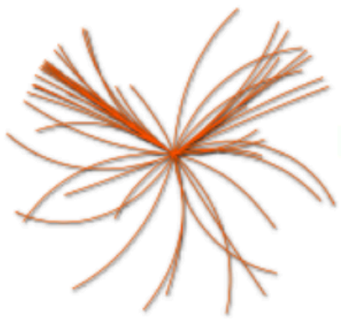
DELPHES status update

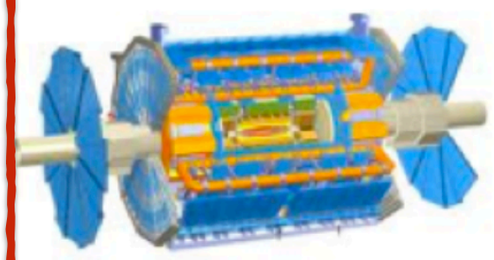
Michele Selvaggi

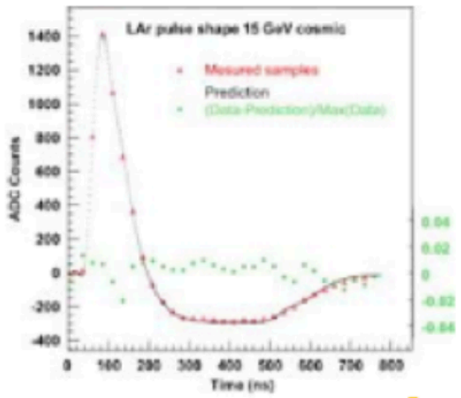
CERN

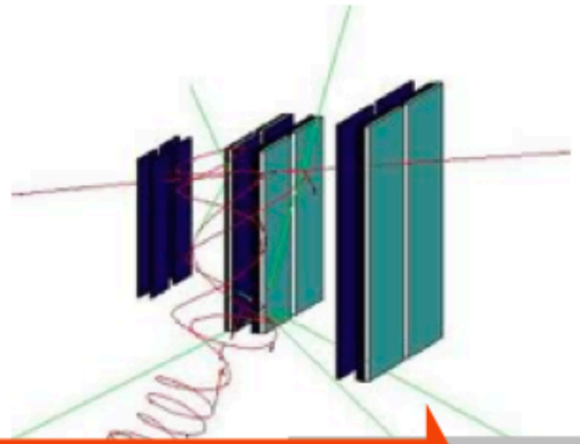
MG meeting - 18/09/2023

MonteCarlo EvGen









Event Generation → Detector Simulation → Digitization → Reconstruction → Rootification

Delphes FastSim

Detector Simulation

- Full simulation (GEANT):
  - **simulates** all **particle-detector interaction** (e.m/hadron showers, nuclear interaction, brem, conversions)

10^2-10^3 s/ev

- Experiment Fast Simulation (ATLAS, CMS ..)
  - **simplify** geometry, smear at the level of detector hits, frozen showers

10-10^2 s/ev

- Parametric simulation (**Delphes**, PGS):
  - **parameterise detector response** at the particle level(efficiency, resolution on tracks, calorimeter objects)
  - reconstruct **complex objects** and observables(use particle-flow, jets, missing ET, pile-up ..)

10^{-2} - 10^{-1} s/ev

- Ultra Fast:
  - from parton to detector object (smearing/lookup tables/ML)

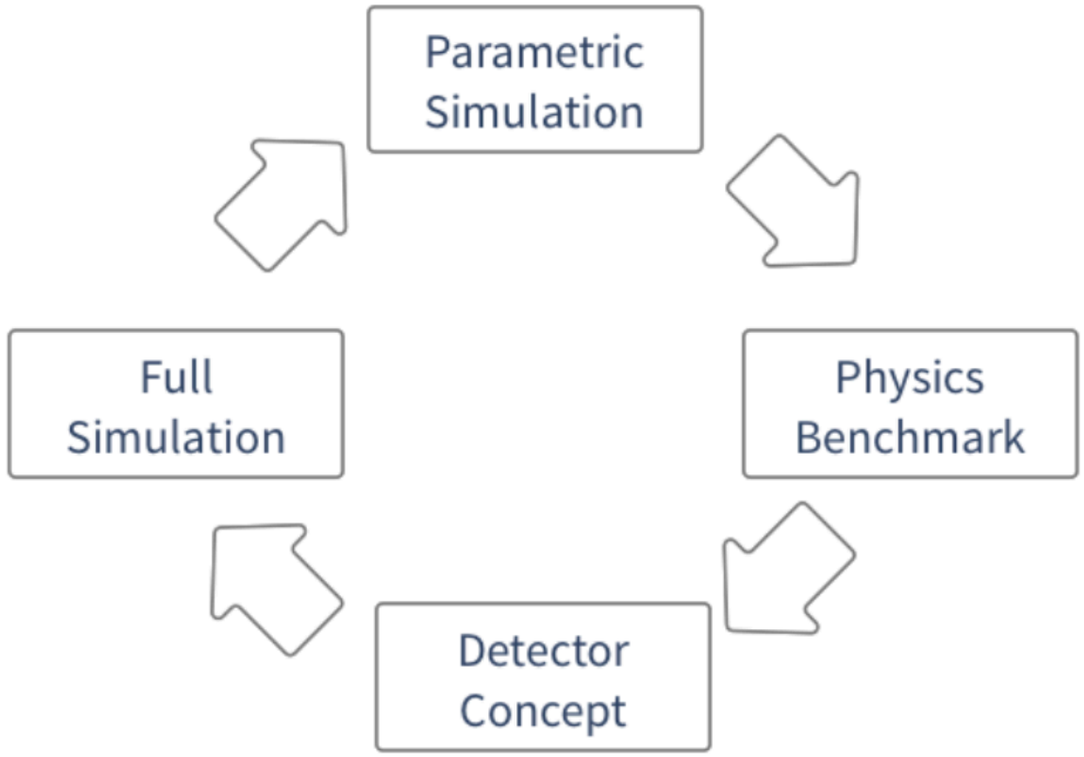
Paradigm

For past/present detectors:

- recast/reinterpret
- reconstruction algorithms dev.
  - (Boosted) jet tagging

Future colliders?

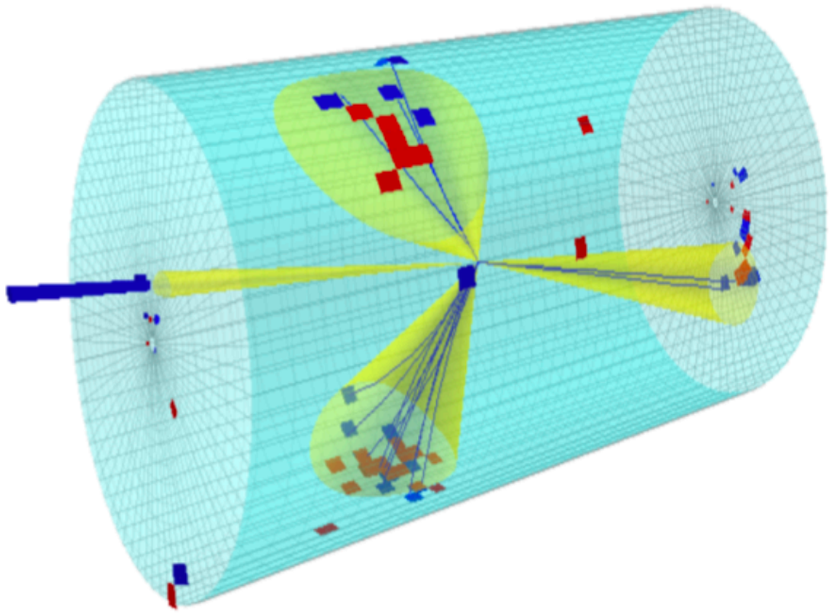
- Easily scan detector parameters
- Reverse engineer detector that maximises performance
- Preliminary sensitivity studies for key physics benchmarks



Delphes in a nutshell

- **Delphes** is a modular framework that simulates the response of a **multipurpose detector** in a parameterised fashion

- Includes:
  - pile-up
  - charged particle propagation in B field
  - EM/Had calorimeters
  - particle-flow



- Provides:
  - leptons, photons, neutral hadrons
  - jets, missing energy
  - heavy flavour tagging

- designed to deal with hadronic environment
- well-suited also for e+e- studies
- detector cards for: CMS (current/PhaseII) - ATLAS - LHCb - FCC-hh - ILD - CEPC - FCCee (IDEA/CLD) - MuCol

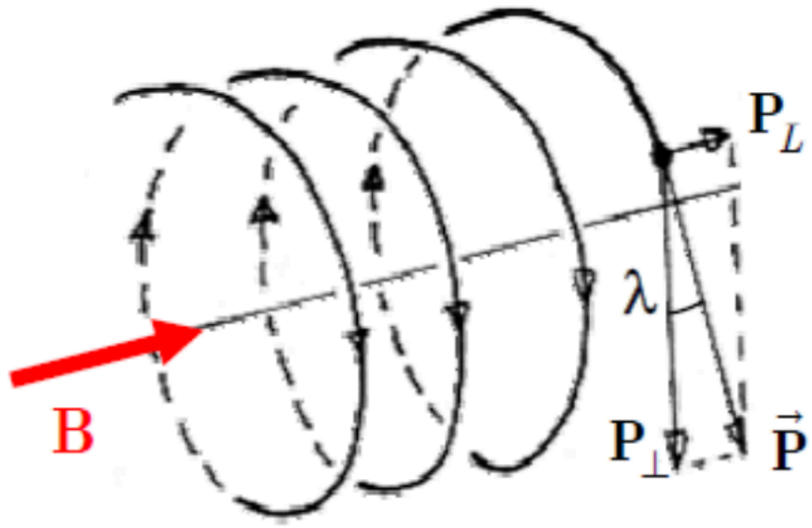
Charged Particle parameterisation

DELPHES
fast simulation

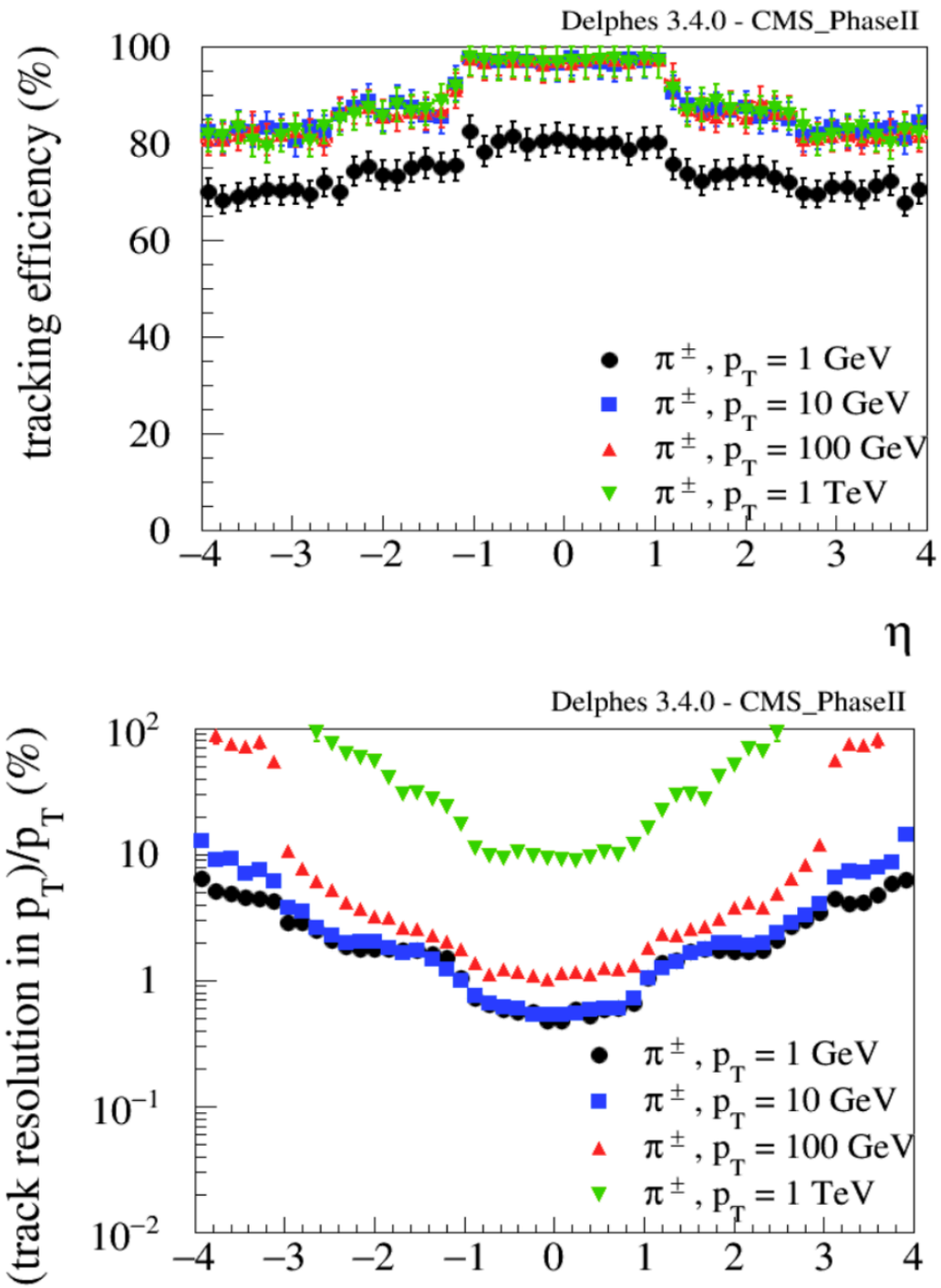
- Charged and neutral particles are propagated in B field until they reach calorimeters

- Propagation parameters:
  - magnetic field B
  - Radius and half-length (R_{max}, z_{max})

- **Efficiency** and **resolution** depends on:
  - particle ID (electron, muon or charged hadron)
  - particle 4-momentum



Tracking parameterisation



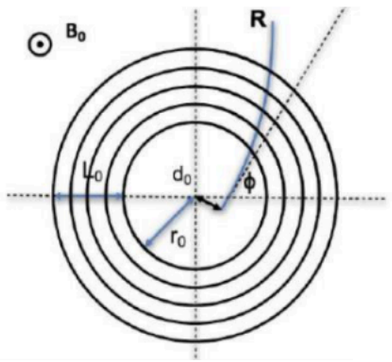

```
#####################################
# Charged hadron tracking efficiency
#####################################

module Efficiency ChargedHadronTrackingEfficiency {
  ## particles after propagation
  set InputArray  ParticlePropagator/chargedHadrons
  set OutputArray chargedHadrons
  # tracking efficiency formula for charged hadrons
  set EfficiencyFormula {
      (pt <= 0.2) * (0.00) + \
          (abs(eta) <= 1.2) * (pt > 0.2 && pt <= 1.0) * (pt * 0.96) + \
          (abs(eta) <= 1.2) * (pt > 1.0) * (0.97) + \
          (abs(eta) > 1.2 && abs(eta) <= 2.5) * (pt > 0.2 && pt <= 1.0) * (pt*0.85) + \
          (abs(eta) > 1.2 && abs(eta) <= 2.5) * (pt > 1.0) * (0.87) + \
          (abs(eta) > 2.5 && abs(eta) <= 4.0) * (pt > 0.2 && pt <= 1.0) * (pt*0.8) + \
          (abs(eta) > 2.5 && abs(eta) <= 4.0) * (pt > 1.0) * (0.82) + \
          (abs(eta) > 4.0) * (0.00)
  }
}
```

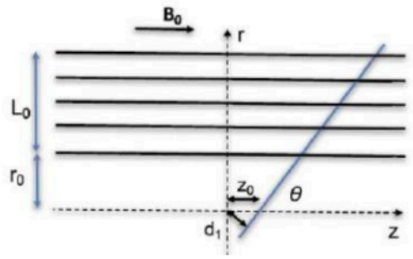


```
set ResolutionFormula {    (abs(eta) >= 0.0000 && abs(eta) < 0.2000) * (pt >= 0.0000 && pt < 1.0000) * (0.00457888) + \
   (abs(eta) >= 0.0000 && abs(eta) < 0.2000) * (pt >= 1.0000 && pt < 10.0000) * (0.004579 + (pt-1.000000)* 0.000045) + \
   (abs(eta) >= 0.0000 && abs(eta) < 0.2000) * (pt >= 10.0000 && pt < 100.0000) * (0.004983 + (pt-10.000000)* 0.000047) + \
   (abs(eta) >= 0.0000 && abs(eta) < 0.2000) * (pt >= 100.0000) * (0.009244*pt/100.000000) + \
   (abs(eta) >= 0.2000 && abs(eta) < 0.4000) * (pt >= 0.0000 && pt < 1.0000) * (0.00505011) + \
   (abs(eta) >= 0.2000 && abs(eta) < 0.4000) * (pt >= 1.0000 && pt < 10.0000) * (0.005050 + (pt-1.000000)* 0.000033) + \
   (abs(eta) >= 0.2000 && abs(eta) < 0.4000) * (pt >= 10.0000 && pt < 100.0000) * (0.005343 + (pt-10.000000)* 0.000043) + \
   (abs(eta) >= 0.2000 && abs(eta) < 0.4000) * (pt >= 100.0000) * (0.009172*pt/100.000000) + \
   (abs(eta) >= 0.4000 && abs(eta) < 0.6000) * (pt >= 0.0000 && pt < 1.0000) * (0.00510573) + \
   (abs(eta) >= 0.4000 && abs(eta) < 0.6000) * (pt >= 1.0000 && pt < 10.0000) * (0.005106 + (pt-1.000000)* 0.000023) + \
   (abs(eta) >= 0.4000 && abs(eta) < 0.6000) * (pt >= 10.0000 && pt < 100.0000) * (0.005317 + (pt-10.000000)* 0.000042) + \
   (abs(eta) >= 0.4000 && abs(eta) < 0.6000) * (pt >= 100.0000) * (0.009077*pt/100.000000) + \
   (abs(eta) >= 0.6000 && abs(eta) < 0.8000) * (pt >= 0.0000 && pt < 1.0000) * (0.00578020) + \
   (abs(eta) >= 0.6000 && abs(eta) < 0.8000) * (pt >= 1.0000 && pt < 10.0000) * (0.005780 + (pt-1.000000)* -0.000000) + \
   (abs(eta) >= 0.6000 && abs(eta) < 0.8000) * (pt >= 10.0000 && pt < 100.0000) * (0.005779 + (pt-10.000000)* 0.000038) + \
   (abs(eta) >= 0.6000 && abs(eta) < 0.8000) * (pt >= 100.0000) * (0.009177*pt/100.000000) + \
   (abs(eta) >= 0.8000 && abs(eta) < 1.0000) * (pt >= 0.0000 && pt < 1.0000) * (0.00728723) + \
   (abs(eta) >= 0.8000 && abs(eta) < 1.0000) * (pt >= 1.0000 && pt < 10.0000) * (0.007287 + (pt-1.000000)* -0.000031) + \
   (abs(eta) >= 0.8000 && abs(eta) < 1.0000) * (pt >= 10.0000 && pt < 100.0000) * (0.007011 + (pt-10.000000)* 0.000038) + \
   (abs(eta) >= 0.8000 && abs(eta) < 1.0000) * (pt >= 100.0000) * (0.010429*pt/100.000000) + \
   (abs(eta) >= 1.0000 && abs(eta) < 1.2000) * (pt >= 0.0000 && pt < 1.0000) * (0.01045117) + \
   (abs(eta) >= 1.0000 && abs(eta) < 1.2000) * (pt >= 1.0000 && pt < 10.0000) * (0.010451 + (pt-1.000000)* -0.000051) + \
   (abs(eta) >= 1.0000 && abs(eta) < 1.2000) * (pt >= 10.0000 && pt < 100.0000) * (0.009989 + (pt-10.000000)* 0.000043) + \
```

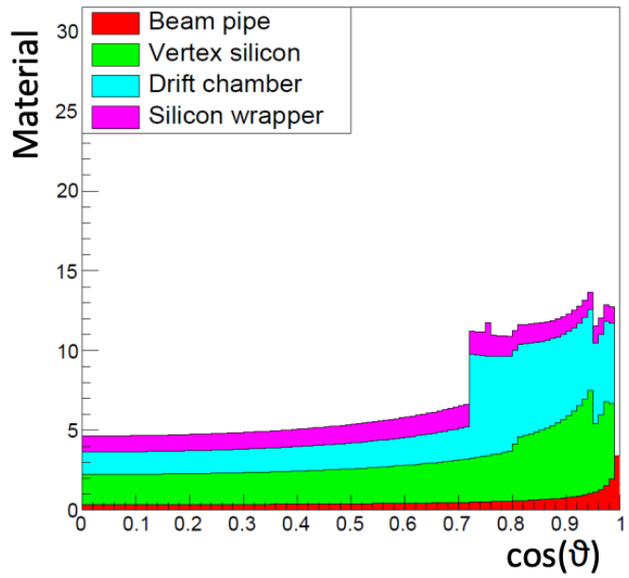
(New) Predictive Tracking

DELPHES
fast simulation



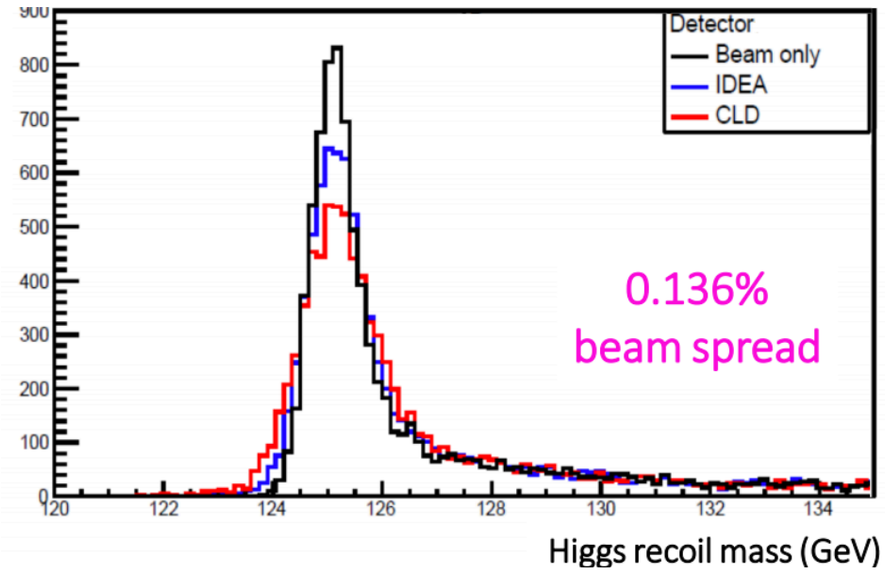


$$\vec{\alpha} = (D,\, \varphi_0,\, C,\, z_0,\, \lambda)$$



Track Smearing

- Simple tracker geometry implementation, **including material**
- Computes **full covariance matrix** (in present Delphes we have "diagonal" smearing in the 5 tracking parameters)
- Can be used for studying impact of material and realistic **HF tagging** simulation



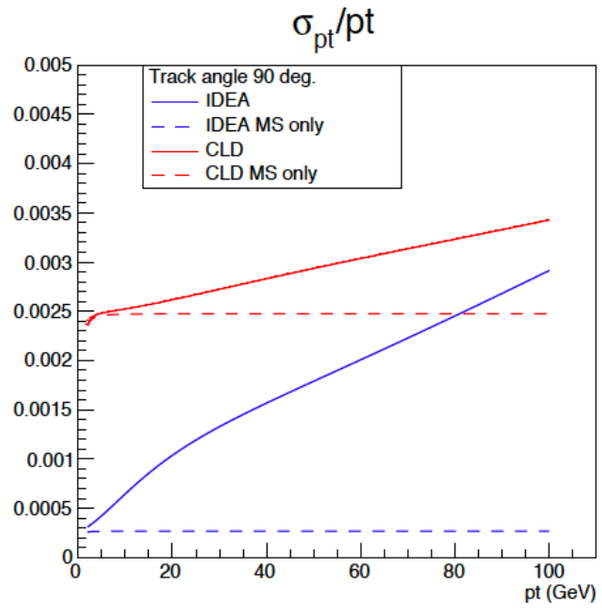
Bedeschi, Gouskos, MS, [2202.03285]

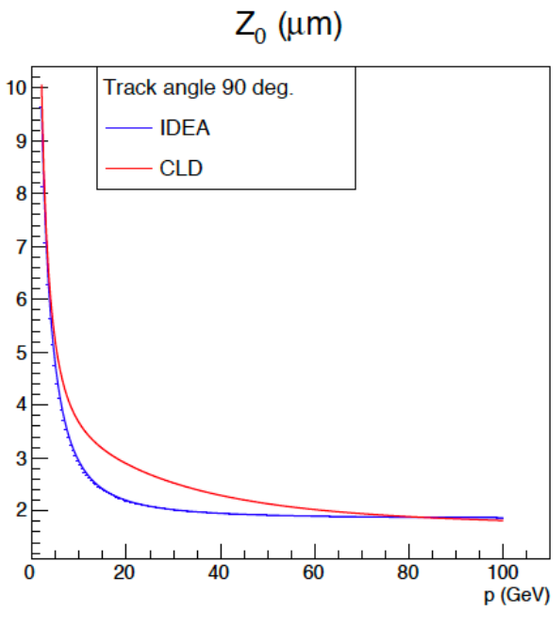
TrackCovariance

DELPHES
fast simulation

```
########################################
# Smearing for charged tracks
########################################

module TrackCovariance TrackSmearing {

    set InputArray TrackMergerPre/tracks
    set OutputArray tracks

    ## minimum number of hits to accept a track
    set NMinHits 6

    ## magnetic field
    set Bz $B

    ## uses https://raw.githubusercontent.com/selvaggi/FastTrackCovariance/master/GeoIDEA_BASE.txt
    set DetectorGeometry {
```

# barrel	name	zmin	zmax	r	w (m)	X0	n_meas	th_up (rad)	th_down (rad)	reso_up (m)	reso_down (m)	flag
1	PIPE	-100	100	0.015	0.001655	0.2805	0	0	0	0	0	0
1	VTXLOW	-0.12	0.12	0.017	0.00028	0.0937	2	0	1.5708	3e-006	3e-006	1
1	VTXLOW	-0.16	0.16	0.023	0.00028	0.0937	2	0	1.5708	3e-006	3e-006	1
1	VTXLOW	-0.16	0.16	0.031	0.00028	0.0937	2	0	1.5708	3e-006	3e-006	1
1	VTXHIGH	-1	1	0.32	0.00047	0.0937	2	0	1.5708	7e-006	7e-006	1
1	VTXHIGH	-1.05	1.05	0.34	0.00047	0.0937	2	0	1.5708	7e-006	7e-006	1



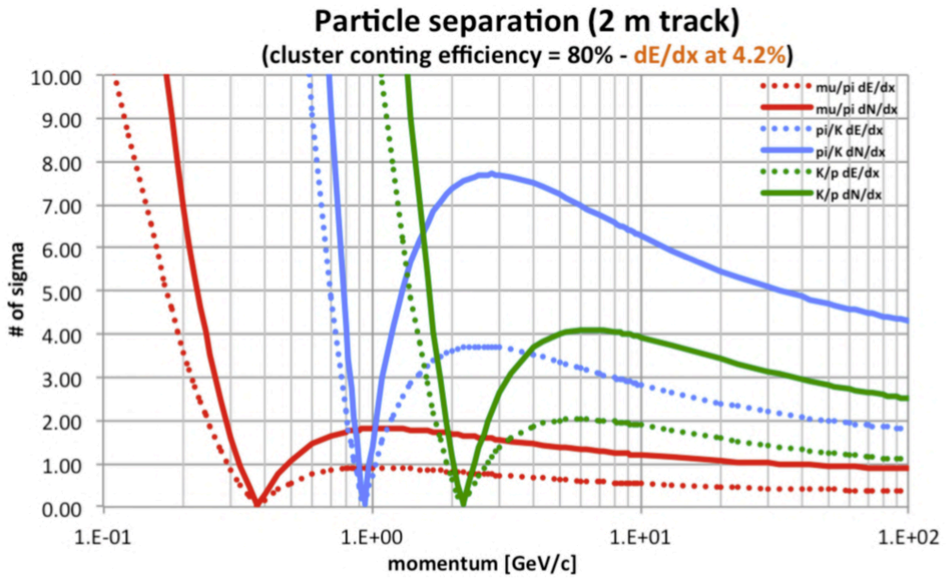


TrackCovariance module

- **Requires:**
  - Geometry input
    - cylinder coaxial
    - planar disks
  - Magnetic field

Particle Identification : Ionisation energy

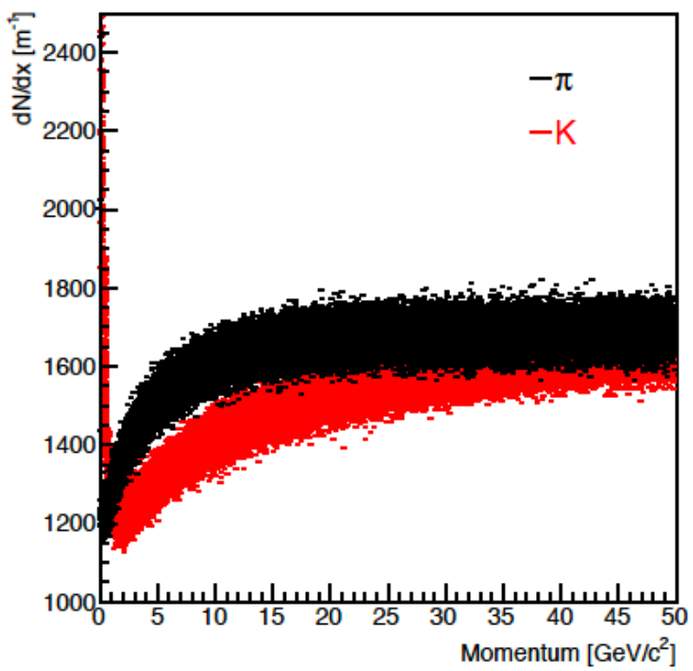
DELPHES
fast simulation




```
##################
# Cluster Counting
##################

module ClusterCounting ClusterCounting {

  add InputArray TrackSmearing/tracks
  set OutputArray tracks

  set Bz $B

  ## check that these are consistent with DCHCANI/DCHNANO parameters in TrackCovariance module
  set Rmin $DCHRMIN
  set Rmax $DCHRMAX
  set Zmin $DCHZMIN
  set Zmax $DCHZMAX

  # gas mix option:
  # 0:  Helium 90% - Isobutane 10%
  # 1:  Helium 100%
  # 2:  Argon 50% - Ethane 50%
  # 3:  Argon 100%

  set GasOption 0

}
```

PID:

- dN/dx method:
  - param. from Garfield (vs βɣ)
- 4 gas mixes implemented
- parameterisation

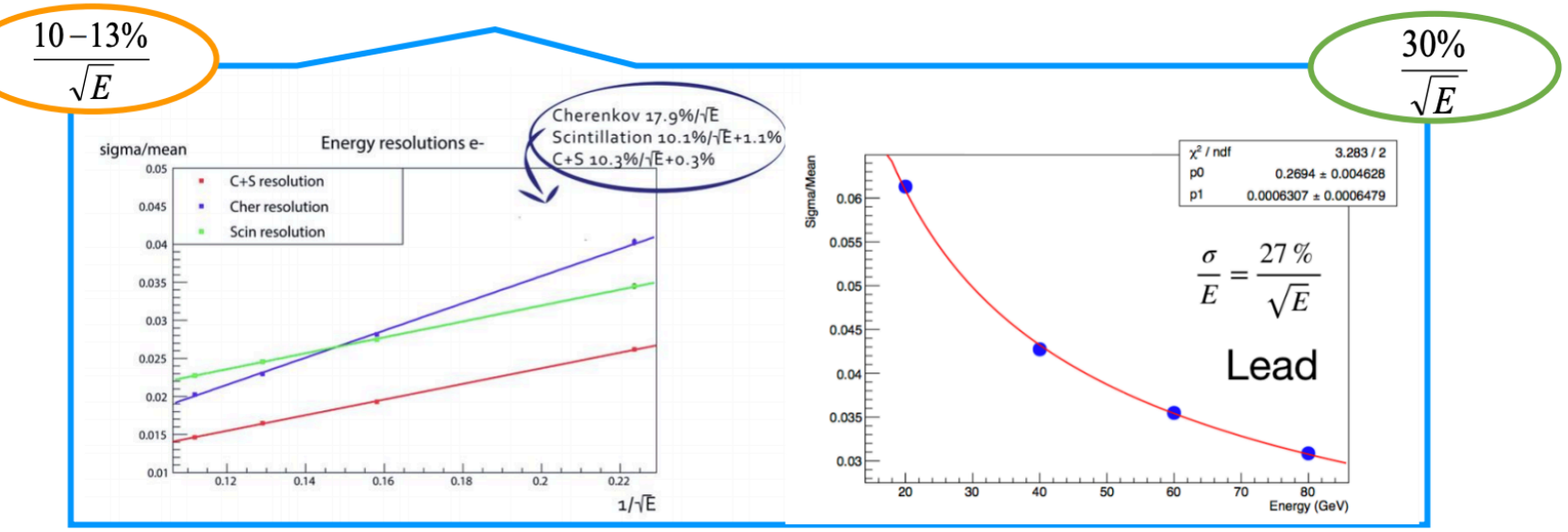


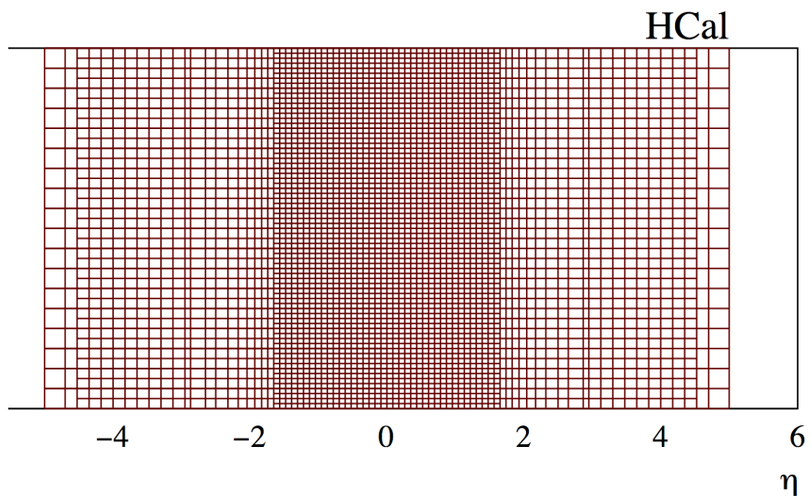
Dual Readout Calorimetry

DELPHES fast simulation

- Calorimeter **segmentation** specified in (η,ϕ) coordinates
- Particles that reach calorimeters **deposits fixed fraction of energy** in f_{EM} (f_{HAD}) in ECAL(HCAL)
- Particle **energy** and **position** is **smeared** according to the calorimeter it reaches

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{S(\eta)}{\sqrt{E}}\right)^2 + \left(\frac{N(\eta)}{E}\right)^2 + C(\eta)^2$$





In Dual Readout, if hadron and EM hit same cell, assume hadronic resolution

Particle-Flow with Dual Readout

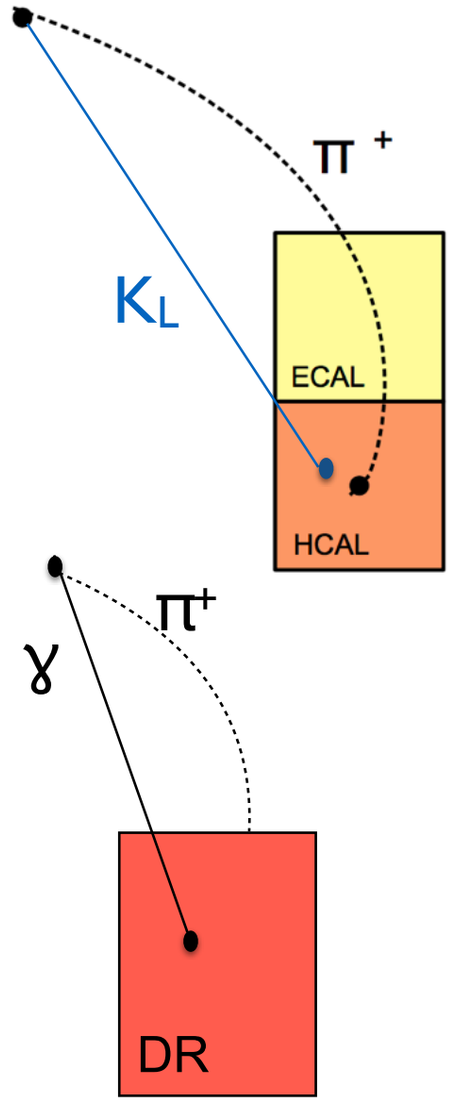
- Given **charged track** hitting **calorimeter cell**:
  - is deposit more compatible with **charged only** or **charged + neutral** hypothesis?
  - how to **assign momenta** to resulting components?

- We have two measurements (E_{trk}, σ_{trk}) and (E_{calo}, σ_{calo})
- Define $E_{Neutral} = E_{calo} - E_{trk}$

<u>Algorithm:</u>

- If $E_{neutral}/\sqrt{(\sigma_{calo}^2 + \sigma_{trk}^2)} > S$:
  → create PF-neutral particle + PF-track

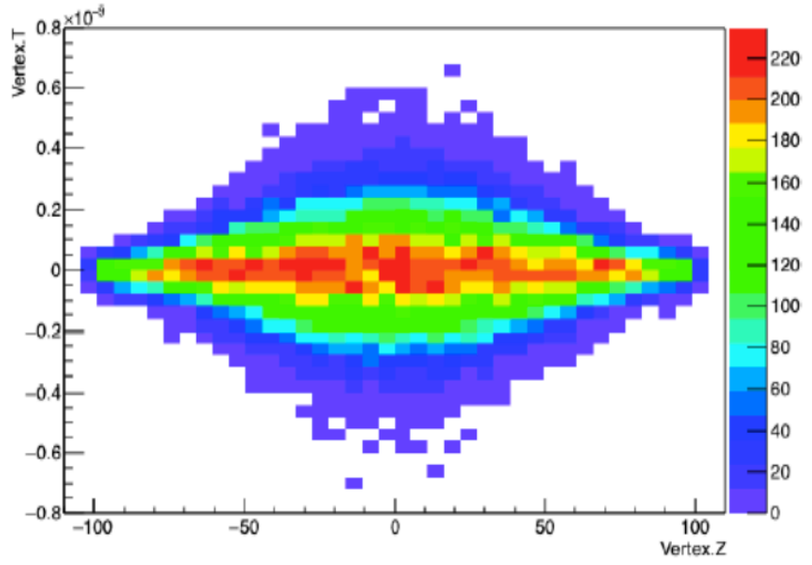
- else:
  create PF-track and **rescale momentum** by:
  combined resolution ~ track resolution

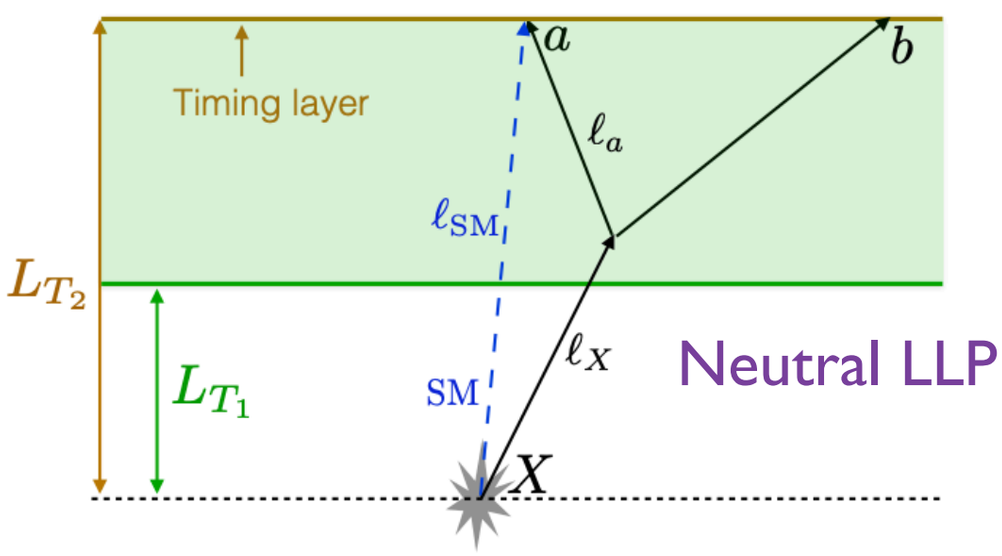


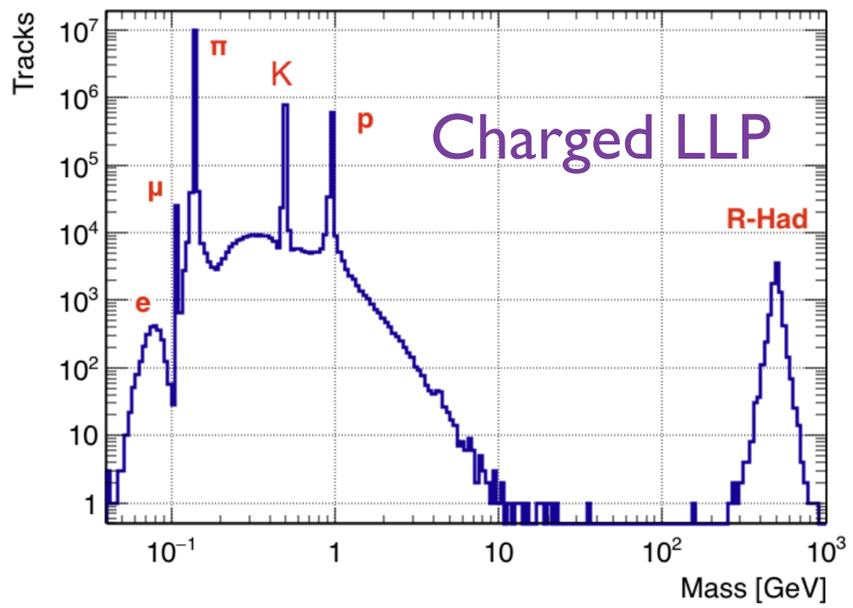
- EM (had) deposit 100% in ECAL (HCAL)
- No propagation in calorimeters
- No clustering (topological) clustering, exploiting pre-defined grid

Timing detectors

Timing can be used to measure TOF, and hence for particle ID (either SM or BSM long lived particles)







PID: Time of flight

DELPHES
fast simulation

- Time-of-flight PID computed by a sequence of modules:

  - **ParticlePropagator** computes path length and final time (L, t_F (MC))
  - **TrackCovariance** (Calorimeter) computes the momentum (energy)
  - **TimeSmearing** module computes measured time (with given time resolution)
  - **TruthVertexFinder** calculates position and time of MC truth vertices
  - **TimeOfFlight** module calculates initial time t_I:
    - $t_I = t_I$ (MC) — optimistic , default
    - $t_I = 0$ — pessimistic, for neutrals
    - $t_I = |x_{vtx}\,(MC)| / \beta_{vtx}\,(MC)$ (naive attempt to realistically account for displacement)

- User has to compute $m_{t.o.f}$ on output tracks using :

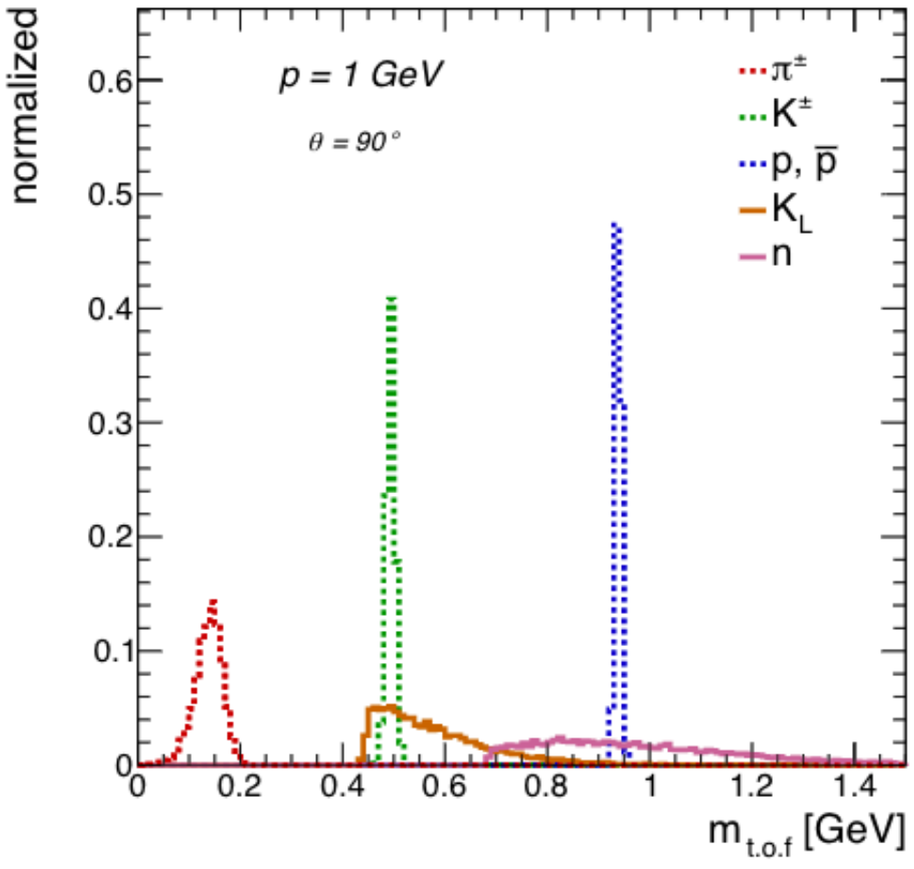
$$t_{\text{flight}} \equiv t_{\text{F}} - t_{\text{V}} = \frac{L}{\beta} = \frac{L\sqrt{p^2+m^2}}{p} = \frac{LE}{\sqrt{E^2-m^2}}$$

charged $\quad m^{(c)}_{\text{t.o.f.}} = p\sqrt{(\frac{t_{\text{flight}}}{L})^2 - 1}$

neutral $\quad m^{(n)}_{\text{t.o.f.}} = E\sqrt{1-(\frac{L}{t_{\text{flight}}})^2}$

PID: Time of flight

- Implemented in the IDEA card for testing for both charged and neutrals:

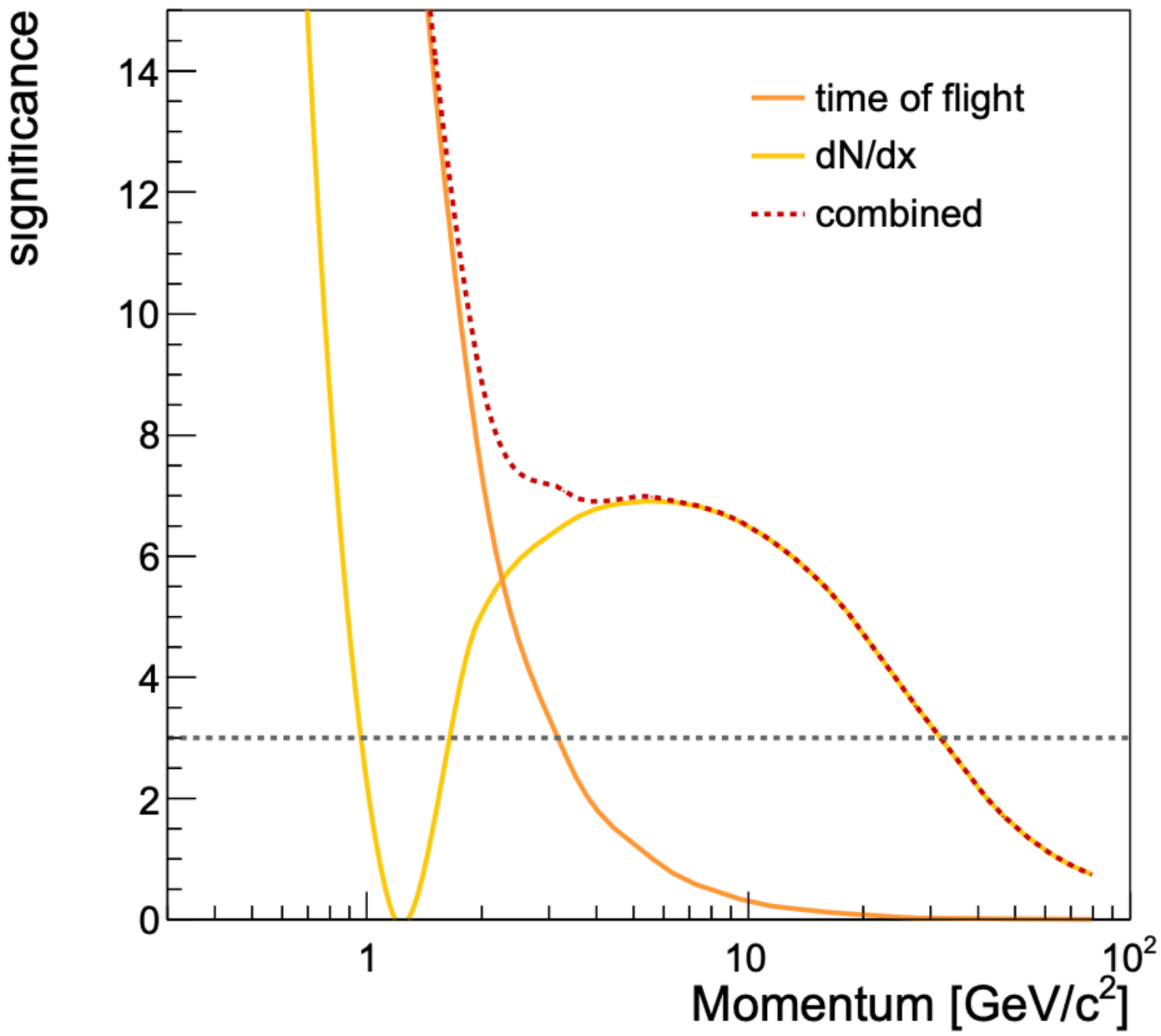



```
########################################
#   Time Smearing MIP
########################################

module TimeSmearing TimeSmearing {
  set InputArray ClusterCounting/tracks
  set OutputArray tracks

  # assume constant 30 ps resolution for now
  set TimeResolution {
                        (abs(eta) > 0.0 && abs(eta) <= 3.0)* 30E-12
                      }
}
```

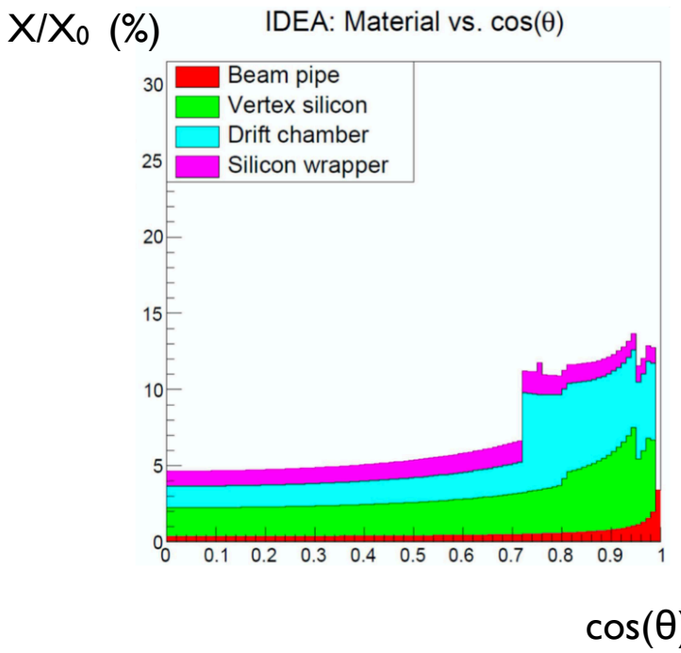
```
########################################
#   Time Of Flight Measurement
########################################

module TimeOfFlight TimeOfFlight {
  set InputArray TimeSmearing/tracks
  set VertexInputArray TruthVertexFinder/vertices

  set OutputArray tracks

  # 0: assume vertex time tV from MC Truth (ideal case)
  # 1: assume vertex time tV = 0
  # 2: calculate vertex time as vertex TOF, assuming tPV=0

  set VertexTimeMode 0
}
```

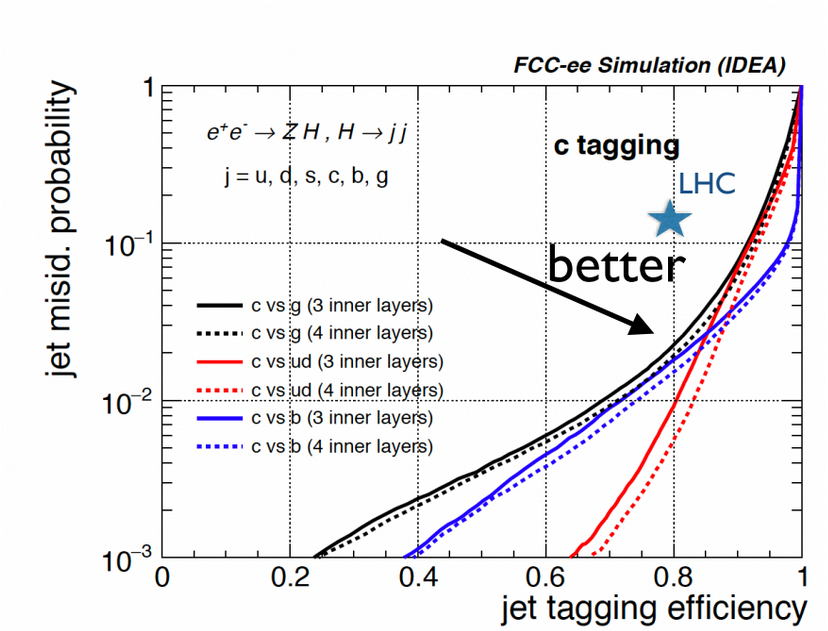
Particle ID (combined)

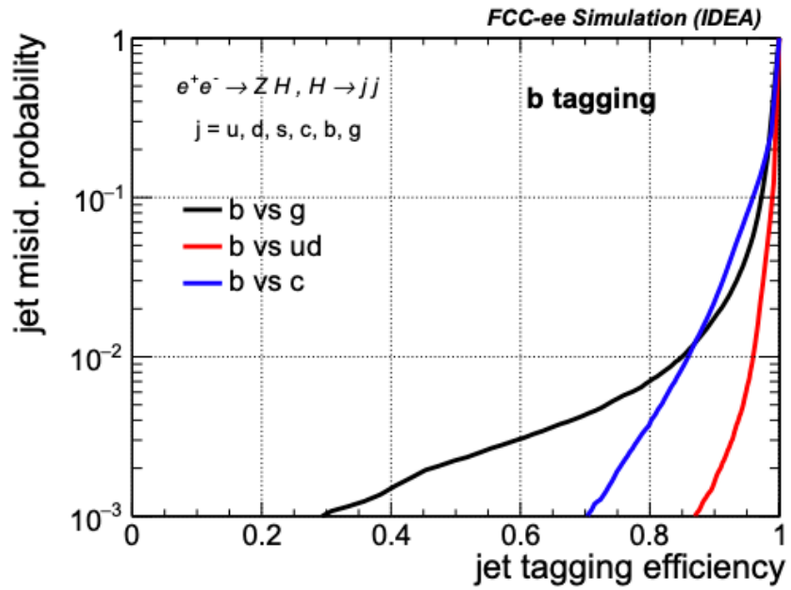
K-π separation power



Flavor tagging (bottom charm)



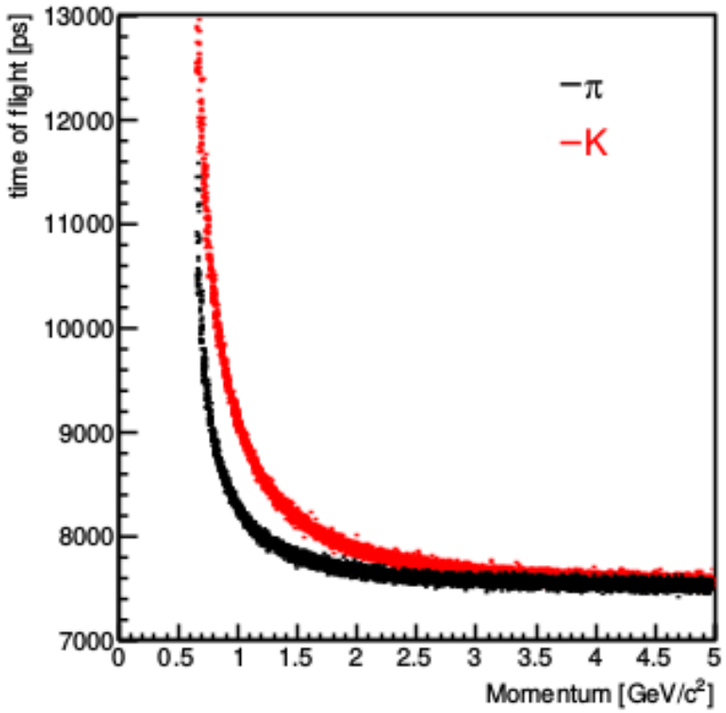


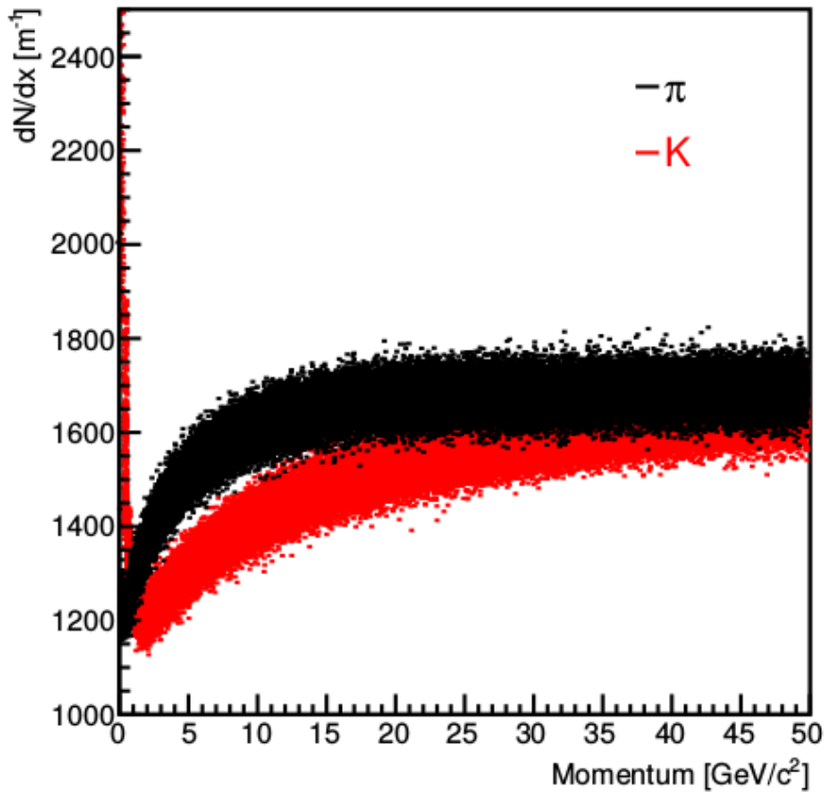


Light tracker, first measurement layer close to IP:

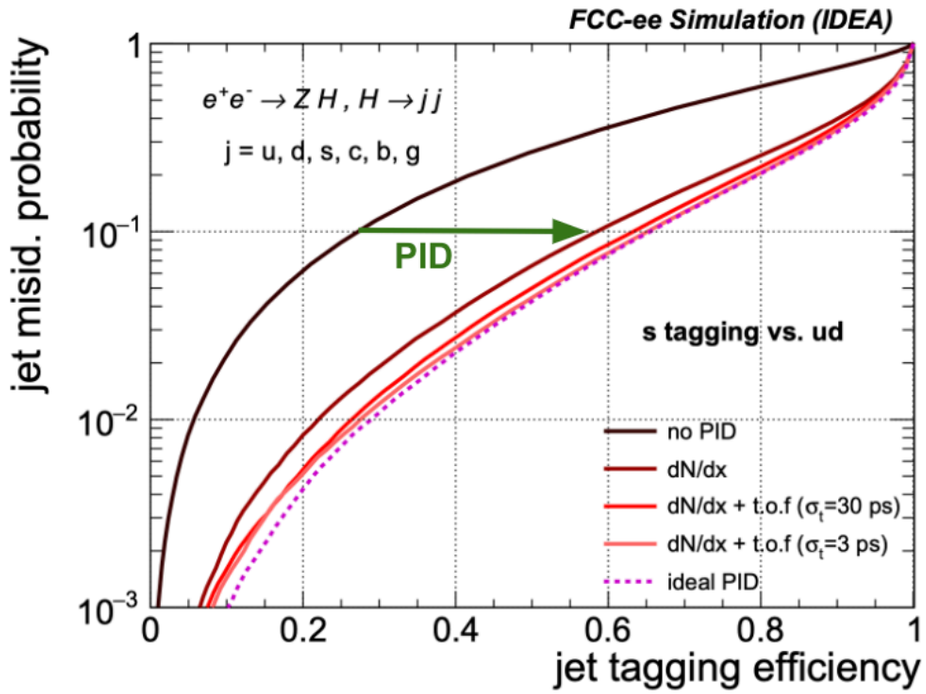
- excellent b/c-tagging performance
  - crucial to measure and to isolate clean H→bb/cc/gg samples

Strange tagging





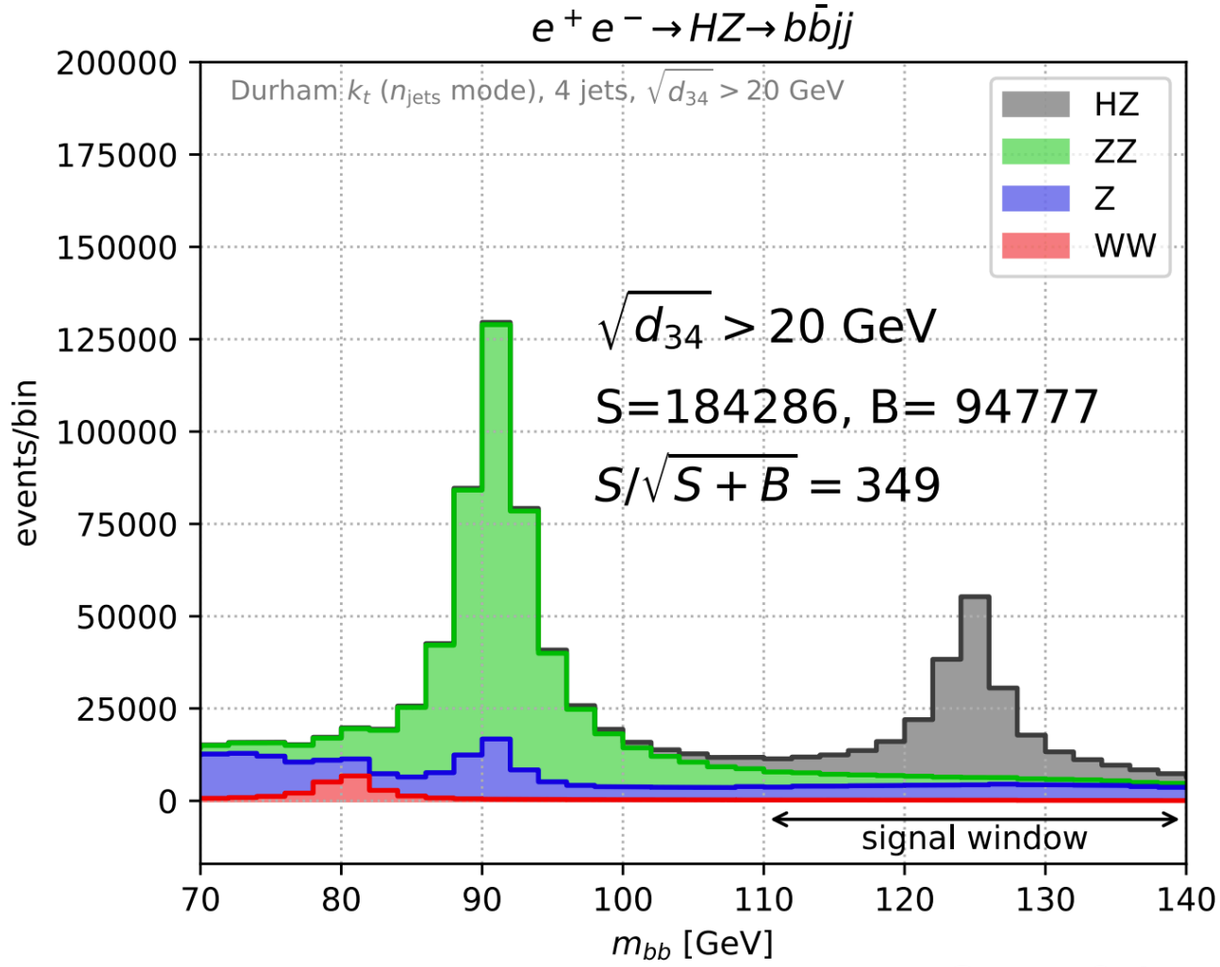
- Particle Id for **strange** jet identification:
  - ToF at low momenta
  - dN/dX at high momenta
- Possible to measure strange Yukawa at FCC-ee ?



(Exclusive) Jet Clustering

- Implemented Durham **inclusive/exclusive** clustering in both “dcut” and “njet” mode
- "Valencia” algorithm

```
##################################
# Jet finder Durham exclusive
##################################

module FastJetFinder FastJetFinderDurhamN2 {
#  set InputArray Calorimeter/towers
  set InputArray EFlowMerger/eflow

  set OutputArray jets

  # algorithm: 11 ee-durham kT algorithm
  # ref: https://indico.cern.ch/event/1173562/contributions/4929025/a
  # to run exclusive njet mode set NJets to int
  # to run exclusive dcut mode set DCut to float
  # if DCut > 0 will run in dcut mode

  set JetAlgorithm 11
  set ExclusiveClustering true
  set NJets 2
  # set DCut 10.0

}
```

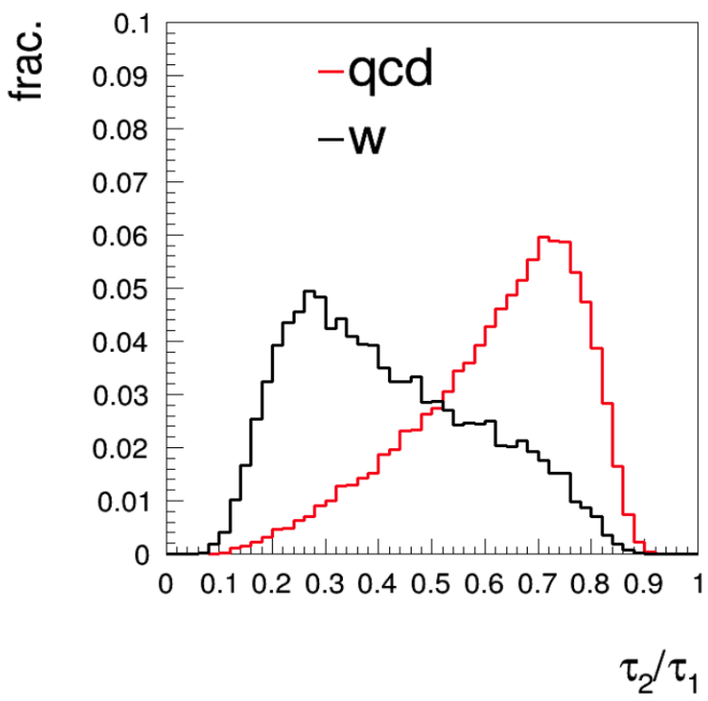


Gavin Salam

Jets and Substructure

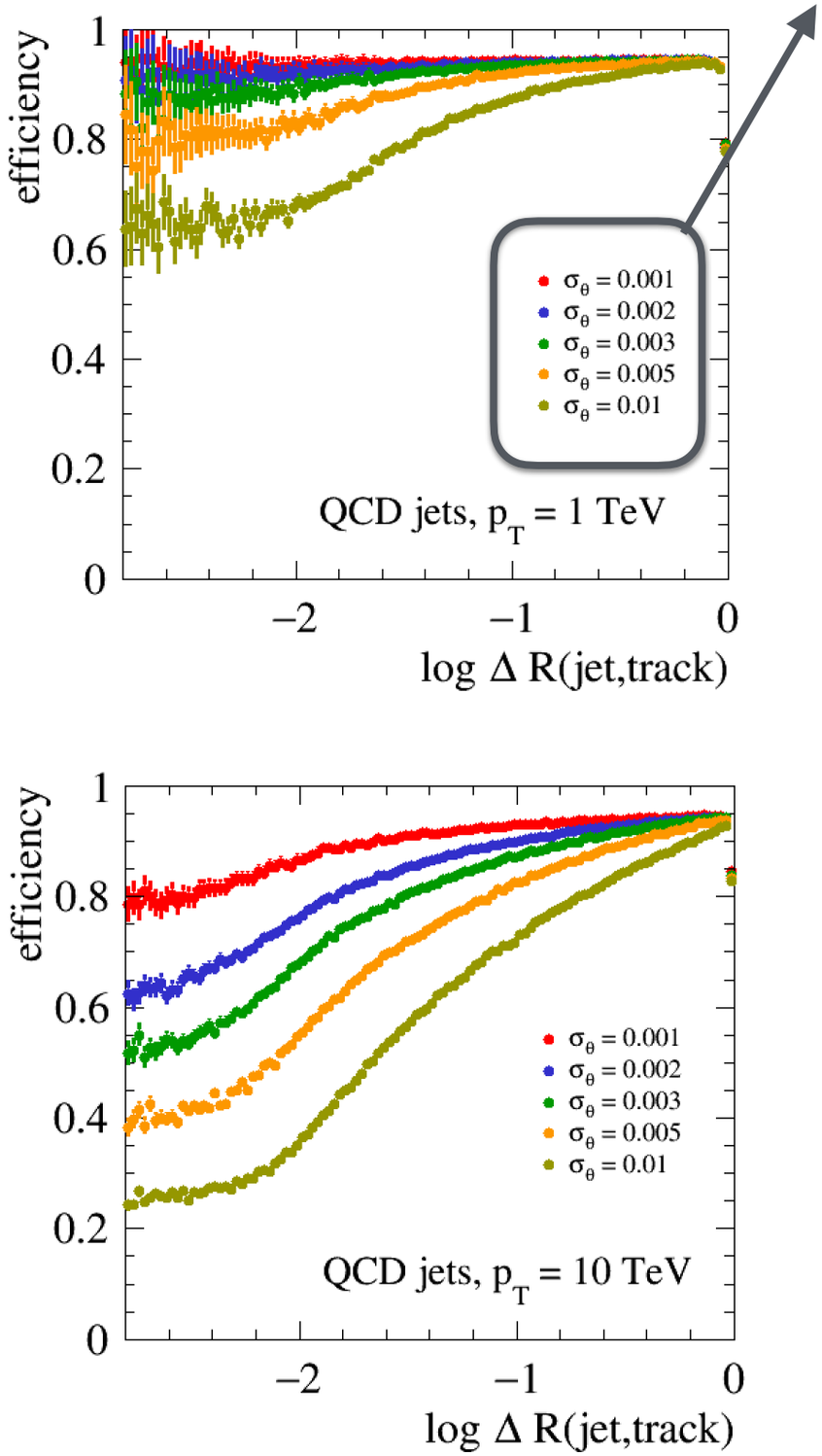
- FastJet performs **jet clustering** via the FastJetFinder module
- Most used Jet substructure algorithms are included (N-subjettiness, SoftDrop, Trimming, Pruning …)
- Delphes can also be used as a library for producing detector 4-vector objects: tracks, calo-towers or particle-flow candidates (see info here)

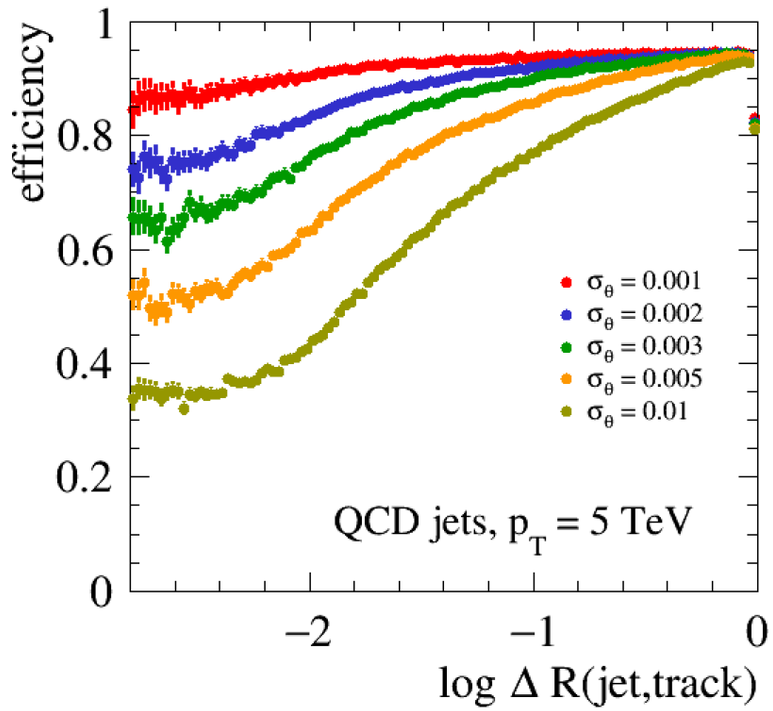
```
############
# Jet finder
############

module FastJetFinder FatJetFinder {
#  set InputArray TowerMerger/towers
  set InputArray EFlowMerger/eflow

  set OutputArray jets

  set JetAlgorithm 5
  set ParameterR 0.8

  set ComputeNsubjettiness 1
  set Beta 1.0
  set AxisMode 4

  set ComputeTrimming 1
  set RTrim 0.2
  set PtFracTrim 0.05

  set ComputePruning 1
  set ZcutPrun 0.1
  set RcutPrun 0.5
  set RPrun 0.8

  set ComputeSoftDrop 1
  set BetaSoftDrop 0.0
  set SymmetryCutSoftDrop 0.1
  set R0SoftDrop 0.8

  set JetPTMin 200.0
}
```

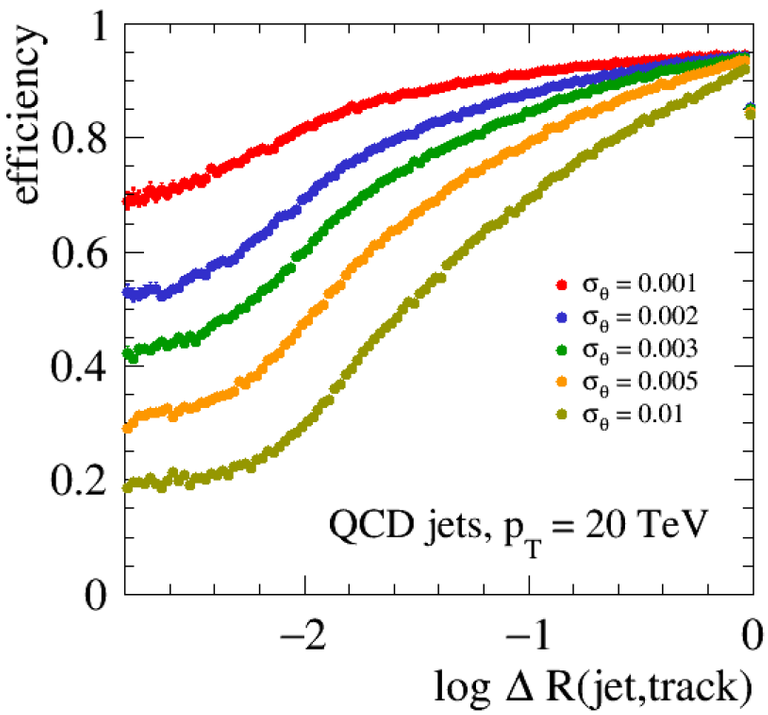


Jet substructure in Dense environment

Instrinsic tracking angular resolution







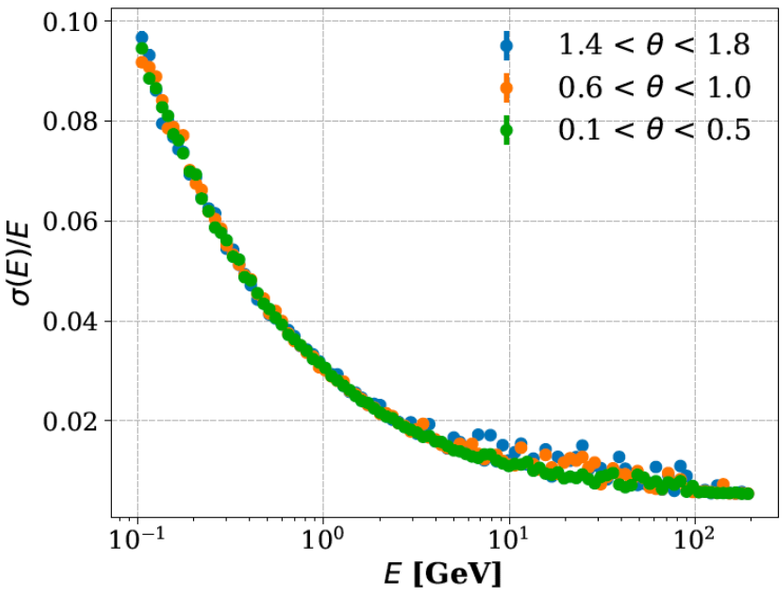
Validation report

- define a configuration file:
  - observables, binning
- configurations for IDEA and FCC-hh scenario I exist
- simple to run:

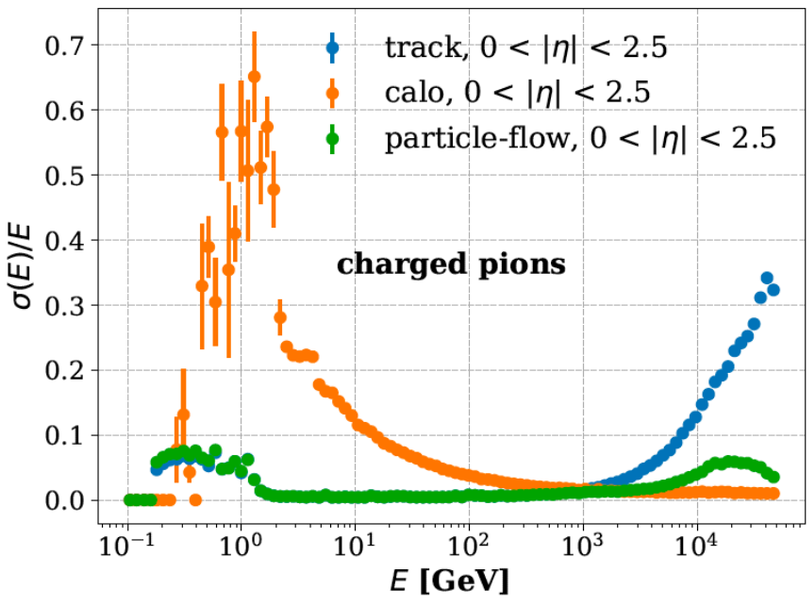
```
python submit.py  --config config/cfg_idea.py launch_local
```

- produces a large report on pdf with validation plots for all objects/efficiencies/resolutions

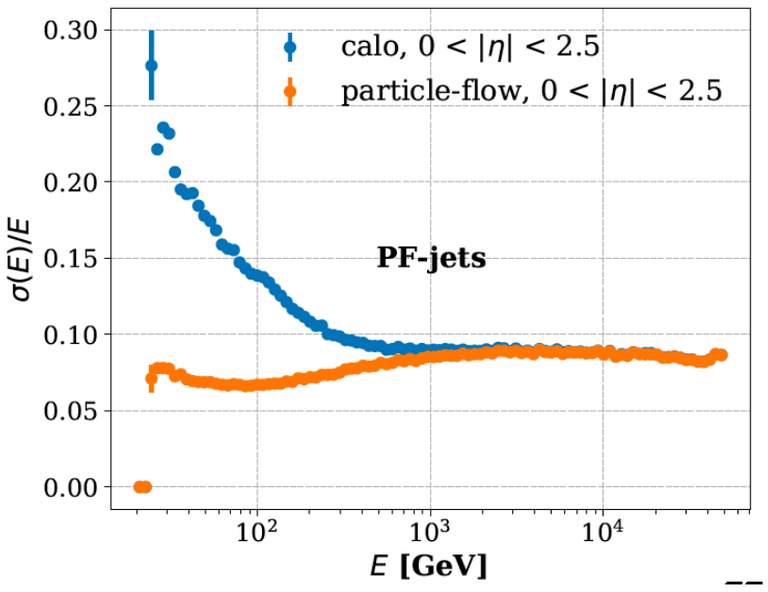
e/gamma energy resolution



PF validation single charged pion



PF validation jets



Comments

- Delphes can hardly predict object low level object reco. efficiency (e.g. tracking, calohit deposits)

BUT:

- Tracking resolution , dN/dx
- Particle Flow
- Jet
- Missing energy
- HF-tagging

Performance can be predicted (with all the caveats of a fast simulation model) …

Conclusion

- Delphes provides a **simple, highly modular framework** for performing fast detector simulation

- New features include:
  - **tracking**
  - Particle ID tools (dNdx/timing)
  - Dual Readout calorimeter and Particle Flow
  - Detector configuration validation suite
  - New cards — future (FCC-ee/μCol)

- Can be used and configured for:
  - quick **phenomenological** studies
  - explore new detector geometries, and iterate over detector design
  - as an **alternative for full-sim** if accurately tuned

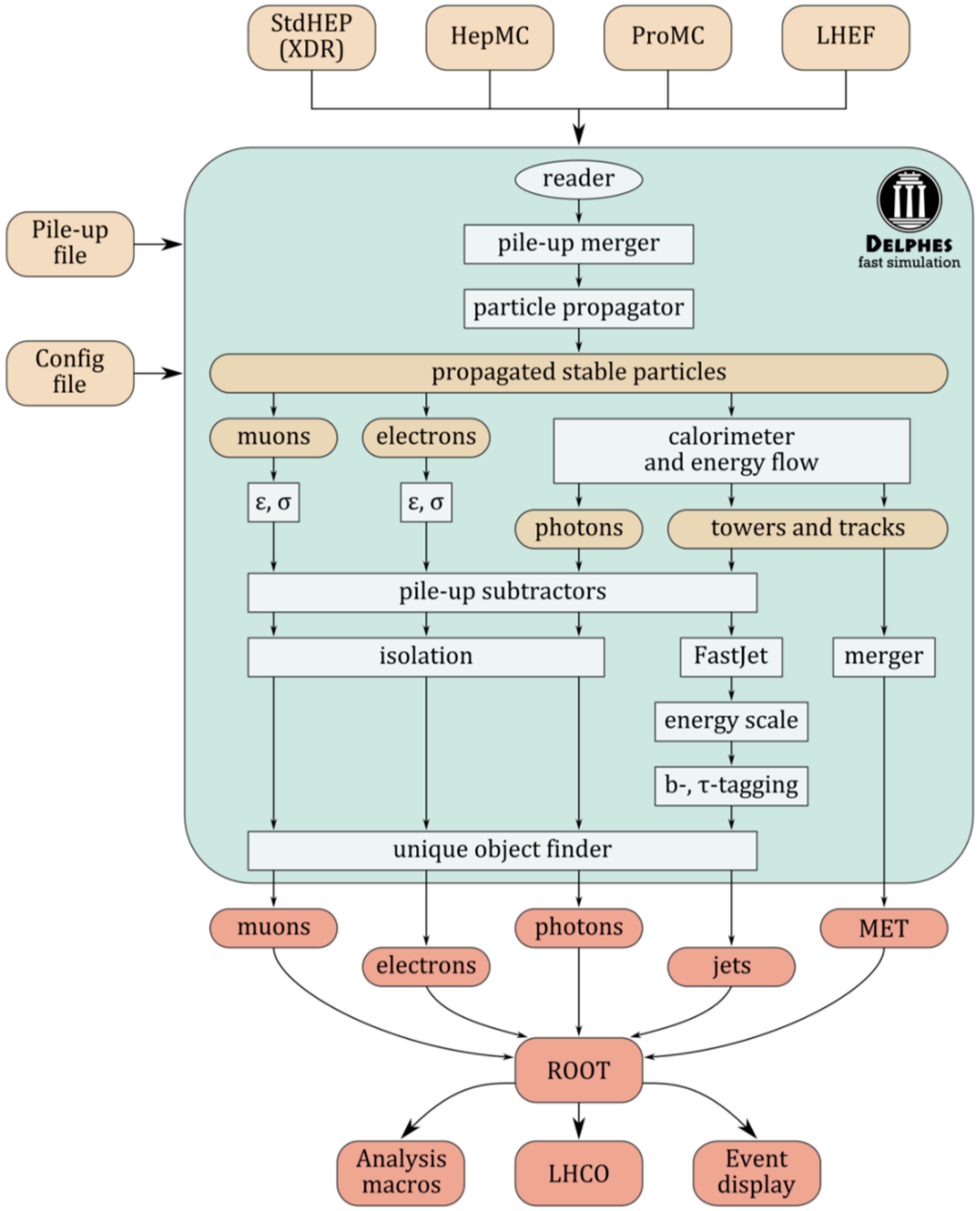
Backup

Modularity

- The modular system allows the user to configure a detector and schedule modules via a configuration file (.tcl), add modules, change data flow, alter output information

- Modules communicate entirely via exchange of collections (vectors) of universal objects (TObjArray of Candidate, 4-vector-like objects)

- Any module can access TObjArrays produced by other modules.



Run

- Install ROOT from root.cern.ch
- Clone Delphes from github.com/delphes
- Run Delphes:

```
>    ./configure
>    make
>    ./DelphesHepMC [detector_card] [output] [input(s)]
```

- Input formats: STDHEP, HepMC, ProMC, Pythia8
- Output: ROOT Tree

Configuration file

- Delphes configuration file is based on tcl scripting language
- This is where the detector parameters, the data-flow and the output content delphes root tree content are defined.
- Delphes provides tuned configurations for most existing detectors:
  - ATLAS, CMS, ILD, FCC, CEPC …

The order of execution of the various modules is configured in the execution path (usually defined at the beginning of the card):

```
set ExecutionPath {
    ParticlePropagator
    TrackEfficiency
    ...
    Calorimeter
    ...
    TreeWriter
}
```

Configuration file

```
module FastJetFinder FastJetFinder {

  set InputArray EFlowMerger/eflow
  set OutputArray jets

  # algorithm: 1 CDFJetClu, 2 MidPoint, 3 SIScone, 4 kt, 5 Cambridge/Aachen, 6 antikt
  set JetAlgorithm 5
  set ParameterR 0.8

  set ComputeNsubjettiness 1
  set Beta 1.0
  set AxisMode 4

  set ComputeTrimming 1
  set RTrim 0.2
  set PtFracTrim 0.05

  set ComputePruning 1
  set ZcutPrun 0.1
  set RcutPrun 0.5
  set RPrun 0.8

  set ComputeSoftDrop 1
  set BetaSoftDrop 0.0
  set SymmetryCutSoftDrop 0.1
  set R0SoftDrop 0.8

  set JetPTMin 20.0


}
```

Configuration file

```
module Calorimeter Calorimeter {

  set ParticleInputArray ParticlePropagator/stableParticles
  set TrackInputArray TrackMerger/tracks

  set TowerOutputArray towers
  set PhotonOutputArray photons

  set EFlowTrackOutputArray eflowTracks
  set EFlowPhotonOutputArray eflowPhotons
  set EFlowNeutralHadronOutputArray eflowNeutralHadrons

  ...

  # 10 degrees towers
  set PhiBins {}
  for {set i -18} {$i <= 18} {incr i} {
    add PhiBins [expr {$i * $pi/18.0}]
  }
  foreach eta {-3.2 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 -1 -0.9 -0.8
-0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
1.9 2 2.1 2.2 2.3 2.4 2.5 2.6 3.3} {
    add EtaPhiBins $eta $PhiBins
  }

  ...
  set ECalResolutionFormula {
            (abs(eta) <= 1.5) * (1+0.64*eta^2) * sqrt(energy^2*0.008^2 + energy*0.11^2 + 0.40^2) +
            (abs(eta) > 1.5 && abs(eta) <= 2.5) * (2.16 + 5.6*(abs(eta)-2)^2) * sqrt(energy^2*0.008^2 +
energy*0.11^2 + 0.40^2) +
            (abs(eta) > 2.5 && abs(eta) <= 5.0) * sqrt(energy^2*0.107^2 + energy*2.08^2)}
```

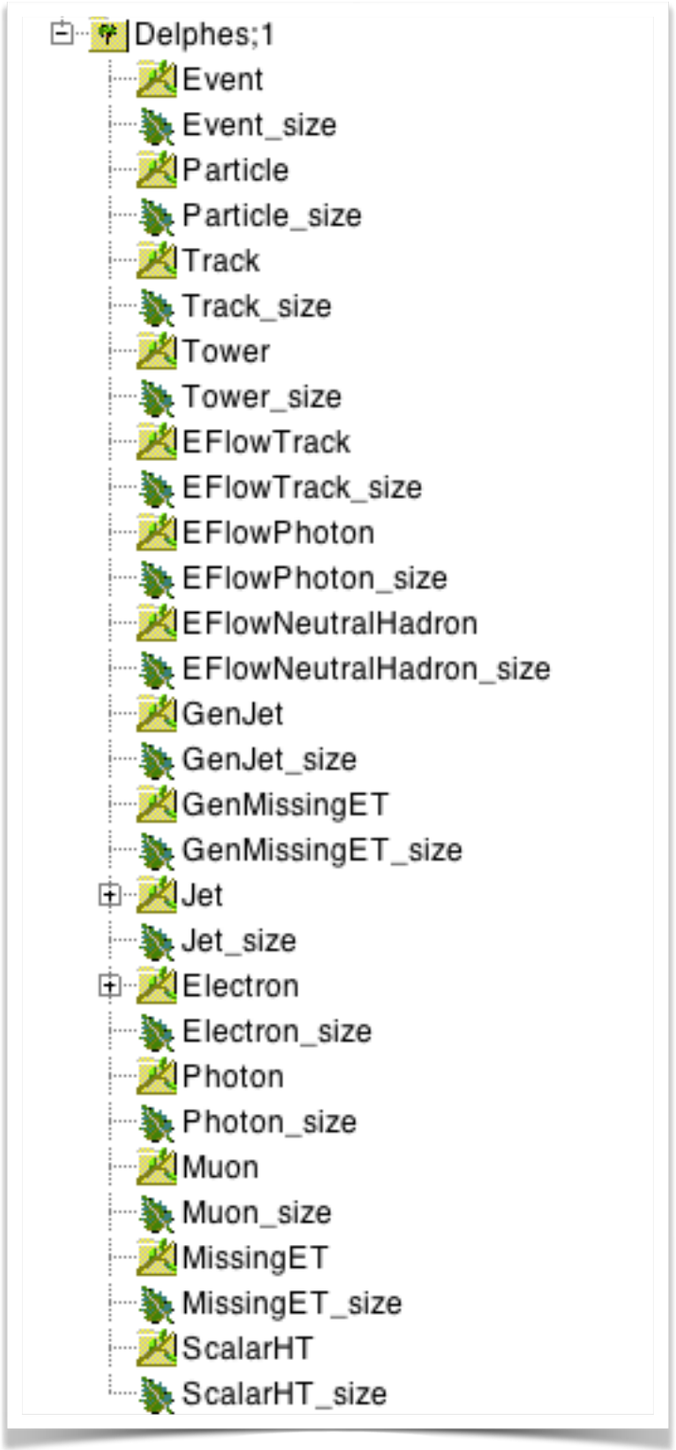
input(s) candidates

output(s) candidates

Configuration file

Output collections are configured in the TreeWriter module

```
module TreeWriter TreeWriter {
# add Branch InputArray BranchName BranchClass
  add Branch Delphes/allParticles Particle GenParticle

  add Branch TrackMerger/tracks Track Track
  add Branch Calorimeter/towers Tower Tower

  add Branch Calorimeter/eflowTracks EFlowTrack Track
  add Branch Calorimeter/eflowPhotons EFlowPhoton Tower
  add Branch Calorimeter/eflowNeutralHadrons EFlowNeutralHadron Tower

  add Branch GenJetFinder/jets GenJet Jet
  add Branch GenMissingET/momentum GenMissingET MissingET

  add Branch UniqueObjectFinder/jets Jet Jet
  add Branch UniqueObjectFinder/electrons Electron Electron
  add Branch UniqueObjectFinder/photons Photon Photon
  add Branch UniqueObjectFinder/muons Muon Muon
  add Branch MissingET/momentum MissingET MissingET
  add Branch ScalarHT/energy ScalarHT ScalarHT
}
```



Tracking in FastSim

pattern recognition
track fitting

Full Simulation

-most accurate
-least flexible
-does not allow for change of geometry

TkLayout

-standalone tool
-predicts tracking performance for a given geometry
-allows for quick turnaroud
- no further implementation needed in Delphes
-requires intermediate step

parameterisation
(d_0, d_z, p, ctg θ, ϕ)

Delphes

FastTrackCovariance

-(for now) standalone tool
-can be plugged into Delphes
-predicts performance given geometry
-allows for quick turnaroud
-not implemented in Delphes yet