libShower

Manual

V0.1

frank.meisel@lhep.unibe.ch

Description

- **This manual is intended to make the user familiar on the way showers are dealt with**

- **The user should not bother about details of programming**

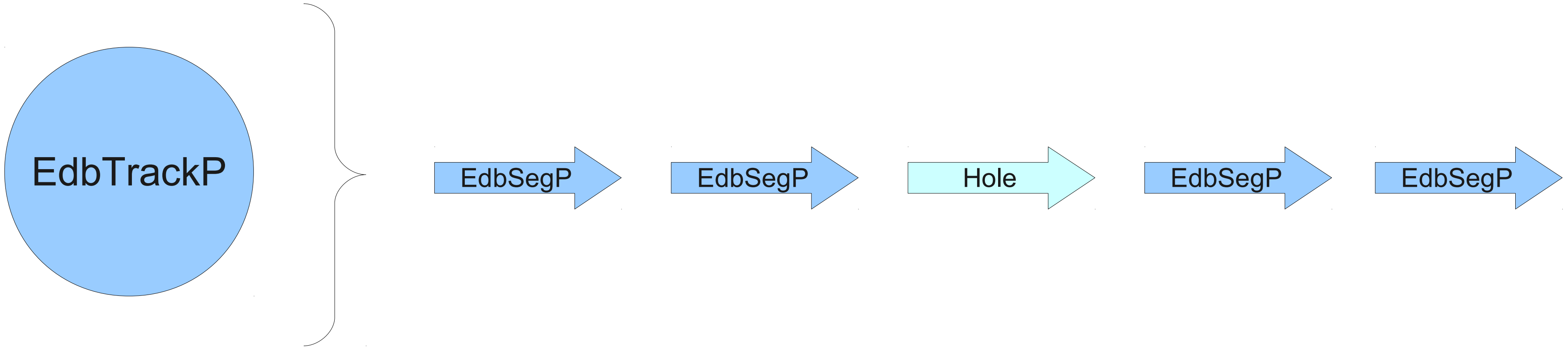
- **To increase the performance, the user may give additional information on the date to obtain a better result**

Overview

- Shower for Event classification.
- Shower characteristics
- Shower implementation in FEDRA
-

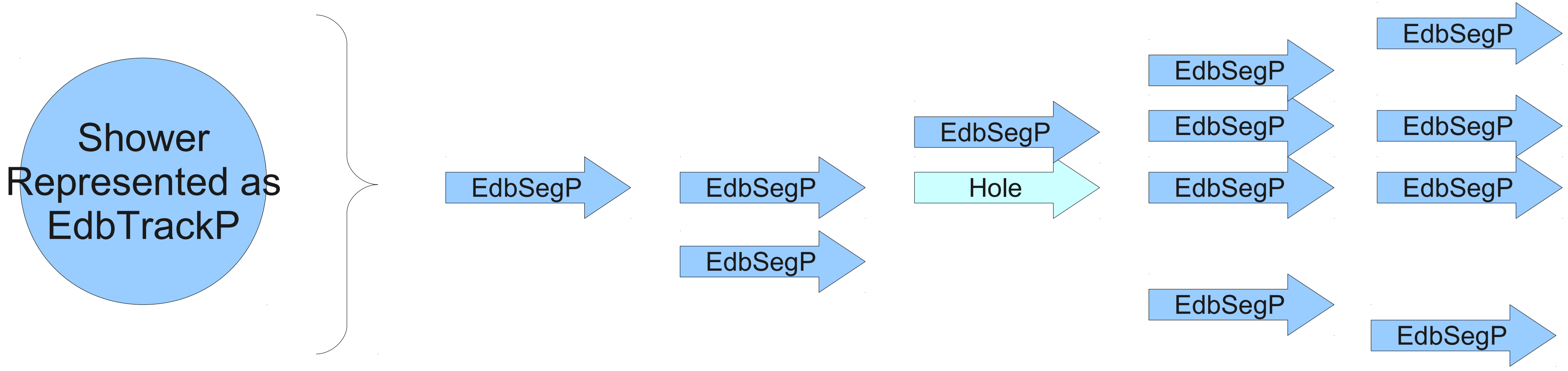
Object description

- A track in fedra
  - One Segment Per Plate
  - N(), Npl(), N0, Theta(), XYZ, P()



Object description

- A shower in fedra
  - Can have *More* than One Segment Per Plate
  - N(), Npl(), N0, Theta(), XYZ, P()
  - Plus additional information: *longitudinal&transversal* Profile!



Object description

Due to backward compabilities we have to use this way.
This is a pain in the back... :-)

- Using EdbTrackP as object class for Shower is ok, but we lose physical information.
- Compensated by infos in EdbShowerAlgE...
  - May be put later in a extra *EdbShowerP* class...
- Up to now: how is a shower represented/stored?
- We have Shower.root which uses a TTree to store the variables. Technically equivalent to do storage like EdbTrackP, but it requires complex I/O conversions to get from
  - TTree entry ↔ EdbTrackP

What is to be done for showers?

- In case we have scanback done, vertex found, scanforth done (hadr. re-interaction), we need for the full description of the event some other information: showers?
- Why are showers for the event classification important?
- Electrons: nu_e from beam; tau->e;
- Photons: present in almost any event, mainly through the decay of:
- Pi0: Find two photons, match them in case of more

Finding showers

- Showers are either attached to vertex directly or indirectly:
  - Electrons: start to shower direct from the vertex without flight length; IP to vertex rather small; DeltaZ to vertex very small.
  - Photons: fly – then showering happens; therefore IP to vertex bigger than attached tracks; DeltaZ to vertex can be large (mean flight length ca10plates).
- Finding a electron shower can be easier than photon shower (directly attached track or BT)

Properties of Showers

- We need to be sure of the following things.
- We have to find the shower (start)
- We have to reconstruct (collect all the tracks) it having as much of it reconstructed as possibles (lowest loss as possible).
- After shower(s) found and reconstructed we need the main properties:
  - Energy
  - ID
  - 1ry-2ry vtx attachment

Properties of the `Shower.root` file.
a) basic quantities

Name in `Shower.root`	Description	Analogy in EdbTrackP
number_eventb/I	MC event number	->MCEvt()
sizeb	Number of Basetracks	N()
nfilmb[]	Plate ID of BT[i]	??(to be looked up)
lengthfilmb	Number of Plates Crossed	Npl()
x,y,z -b[]	X,Y,Z Coordinates of BT[i]	->X(),Y(),Z()
tx,ty -b[]	TX,TY Coordinates of BT[i]	->TX(),TY()
plateb -b[]	??(to be looked up)	??(to be looked up)
Numbereventb[]	Numbereventb[]	Numbereventb[]
showerID/I	showerID/I	showerID/I
isizeb/I	isizeb/I	isizeb/I
ntrace1simu[sizeb]/I	ntrace1simu[sizeb]/I	ntrace1simu[sizeb]/I
ntrace2simu[sizeb]/I	ntrace2simu[sizeb]/I	ntrace2simu[sizeb]/I
ntrace3simu[sizeb]/I	ntrace3simu[sizeb]/I	ntrace3simu[sizeb]/I
Ntrace4simu[sizeb]/I	Ntrace4simu[sizeb]/I	Ntrace4simu[sizeb]/I
chi2btkb[sizeb]/F	chi2btkb[sizeb]/F	chi2btkb[sizeb]/F

Properties of the `Shower.root` file.
b) derived quantities

Name in `Shower.root`	**Description**	**Analogy in EdbTrackP**
number_eventb/I	number_eventb/I	number_eventb/I
sizeb	sizeb	sizeb
nfilmb[]	nfilmb[]	nfilmb[]
lengthfilmb	lengthfilmb	lengthfilmb
x,y,z -b[]	x,y,z -b[]	x,y,z -b[]
tx,ty -b[]	tx,ty -b[]	tx,ty -b[]
plateb -b[]	plateb -b[]	plateb -b[]
Numbereventb[]	Numbereventb[]	Numbereventb[]
showerID/I	showerID/I	showerID/I
isizeb/I	isizeb/I	isizeb/I
ntrace1simu[sizeb]/I	ntrace1simu[sizeb]/I	ntrace1simu[sizeb]/I
ntrace2simu[sizeb]/I	ntrace2simu[sizeb]/I	ntrace2simu[sizeb]/I
ntrace3simu[sizeb]/I	ntrace3simu[sizeb]/I	ntrace3simu[sizeb]/I
Ntrace4simu[sizeb]/I	Ntrace4simu[sizeb]/I	Ntrace4simu[sizeb]/I
chi2btkb[sizeb]/F	chi2btkb[sizeb]/F	chi2btkb[sizeb]/F

Reconstruction: Mode A

- Choose Initiator Basetracks:

```
ShowerInstance = new EdbShowerRec();
ShowerInstance-> SetInBTArray(TObjArray* InBTArray);
```

- Set the PVR object on which the reconstruction shall operate:

```
ShowerInstance-> SetEdbPVRec(EdbPVRec* Ali);
```

- Start Reconstruction

```
ShowerInstance-> Execute();
```

- Retrieve Reconstructed Shower Array

```
ShowerInstance-> GetRecoShowerArray();
```

Reconstruction: Mode B Frederics Algo Implementation

- To be filled and explained...:
- void recdown(int num, int MAXPLATE, int DATA, int Ncand, double* x0, double* y0, double* z0, double* tx0, double* ty0, double* chi20, int* W0, double* P0, int* Flag0, int* plate0, int* id0, int* TRid, double* Esim, int piece2, int piece2par)

Shower Reconstruction Algorithms

- There have been tested several algorithms to find out optimal shower reconstruction.
- Some are similar to each other, some are different.
- They can be grouped into two types:
  - Local Basetrack Cut Algorithms (CT, CA, OI, RC, BW)
  - Global Basetrack Cut Algorithms (CL, TC, GS)
- Pseudocodes of all algorithms can be found in thesis, code for OI and GS algo are given in next two slides:

Algorithm Shower reconstruction: CT (ConeTube)

Require: BT_{IN}, Volume of plates containing basetracks.
Ensure: PARA0: ConeAngle, PARA1: TubeRadius,
Ensure: PARA2: Connection ΔR, PARA3: Connection $\Delta\hat{\Theta}$.

```
Create small volume around BT_IN with ±ΔX = ±ΔY = 1200μm
plate = plate(BT_IN)
while plate != last plate do
  for i = 0 to N_BT(Pl(i)) do
    Check BT for criteria: ConeTube:
    if ∋ConeTube(PARA0,PARA1) BT(i,IN) then
      CONTINUE
    end if
    for j = 0 to N_BT(Shower)) do
      Check BT for criteria: Connection:
      if plate(BT_IN) != plate(BT_i) then
        if ΔR BT(i,j) >PARA2 or ΔΘ̂ BT(i,j) >PARA3 or Δplate BT(i,j) > 3 then
          CONTINUE
        end if
      end if
      if plate(BT_IN) == plate(BT_i) then
        if ΔR BT(i,IN) > 50μm or ΔΘ̂ BT(i,IN) > 80mrad then
          CONTINUE
        end if
      end if
    end for
  end for
  Add BT(i) to Shower
  Next (Ascending Z) Plate
end while
return Collection of BTs = Shower
```

Algorithm Shower reconstruction: GS (GammaSearch) — !!! PSEUDOCODE !!! —

```
Require: BT_IN, Vertex VTX, Volume of plates containing basetracks.
Ensure: PARA0: Connection ΔR_IP, PARA1: Connection ΔR_minDist, PARA2: Connection ΔR_spatial,
Ensure: PARA3: Connection ΔZ, PARA4: ΔΘ, PARA5: Plate Difference ΔN_pl,
  Create small volume around BT_IN with ±ΔX = ±ΔY = 1200μm
  plate = plate(BT_IN)
  while plate != last plate do
    for i = 0 to N_BT(Pl(i)) do
      Check BT for criteria: IP Distance:
      if Min(IP(BT(i),VTX),IP(BT(IN),VTX)) >PARA0 then
        CONTINUE
      end if
      Check BT for criteria: Distance:
      if ΔR_{i,IN;backward}(BT(i,IN)) >PARA2 then
        CONTINUE
      end if
      Check BT for criteria: Minimum Distance:
      if ΔR_minDist(BT(i,IN)) >PARA1 then
        CONTINUE
      end if
      Check BT for criteria: Angular Distance:
      if ΔΘ(BT(i,IN)) >PARA4 then
        CONTINUE
      end if
      Check BT for criteria: Z Distance:
      if ΔZ(BT(i,VTX)) >PARA3 then
        CONTINUE
      end if
      Check BT for criteria: Plate Difference:
      if Abs(plate(BT_IN − plate(BT_i)) >PARA5 then
        CONTINUE
      end if
    end for
    Add BT(i) to Shower
    Next (Ascending Z) Plate
  end while
  return Collection of BTs = Shower
```

Results for Shower Reco Algs:

- OI
- GS

Shower Energy Measurement

See extra file...

Shower ID

- Problem:
  - One single Shower can originate from either photons, electrons or ($_{\text{charged}}$) pions

- Solution:
  - Shower ID class, which does a differentiation!

- Technicalities:
  - The ID class determines Shower from its originating particle

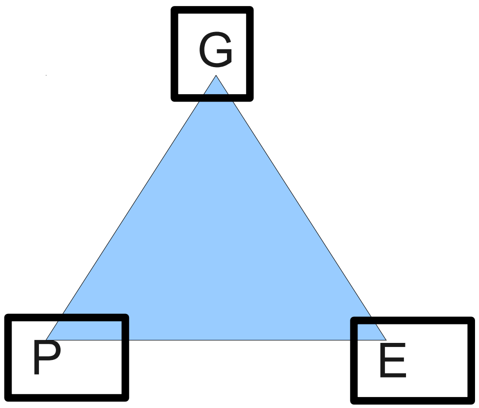
Shower ID

- Instead of assigning the ANN one number for
  - Gamma, Electron, Pion(+-)
- We do decions based on „X vs Y“, i.e.
  - Gamma vs Electron
  - Electron vs Pion
- Because the ANN is better in discriminating two-by-two variable discrimination.
- In continuation with former implementations, we save these results in eProb1,eProb90_X_vs_Y

Eprob1: bg rejection at 1%

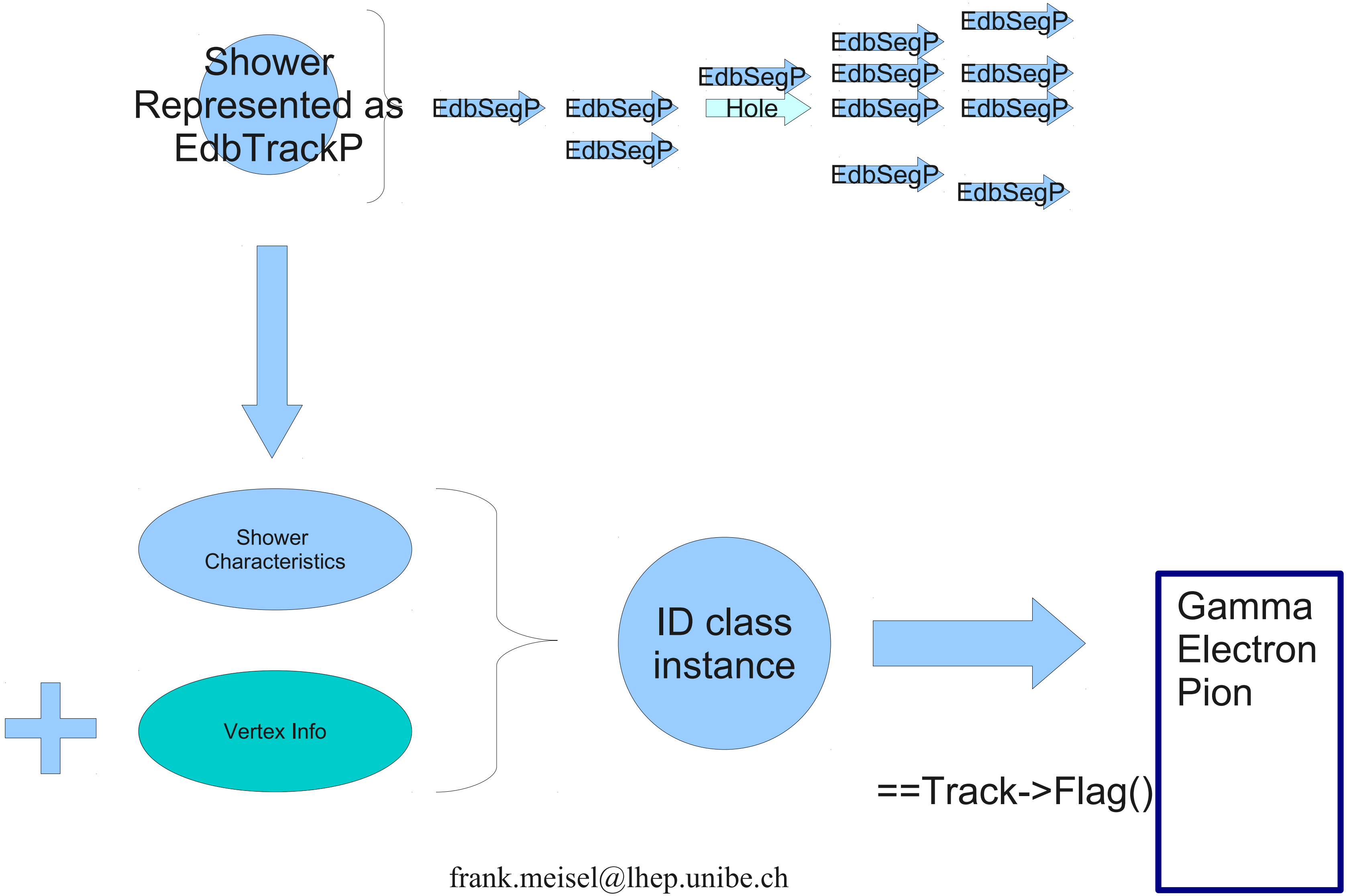
Eprob90: signal efficiency 90%

Shower ID



- Two type of variables:
  - SHID: 0(G),1(E),2(P)
  - SPID: 0,1,2
  - So we can assign a unique
  - ID for each combination:
  - Example:
    - gamma_vs_e: SHID=0,SPID=1
    - e_vs_pi: SHID=1,SPID=2
  - Always put 0 if it belongs to Type 0 (SHID)
  - Always put 1 if it belongs to Type 1 (SPID)

Block diagram: ID



More here...

The Cleaning of High BG level...

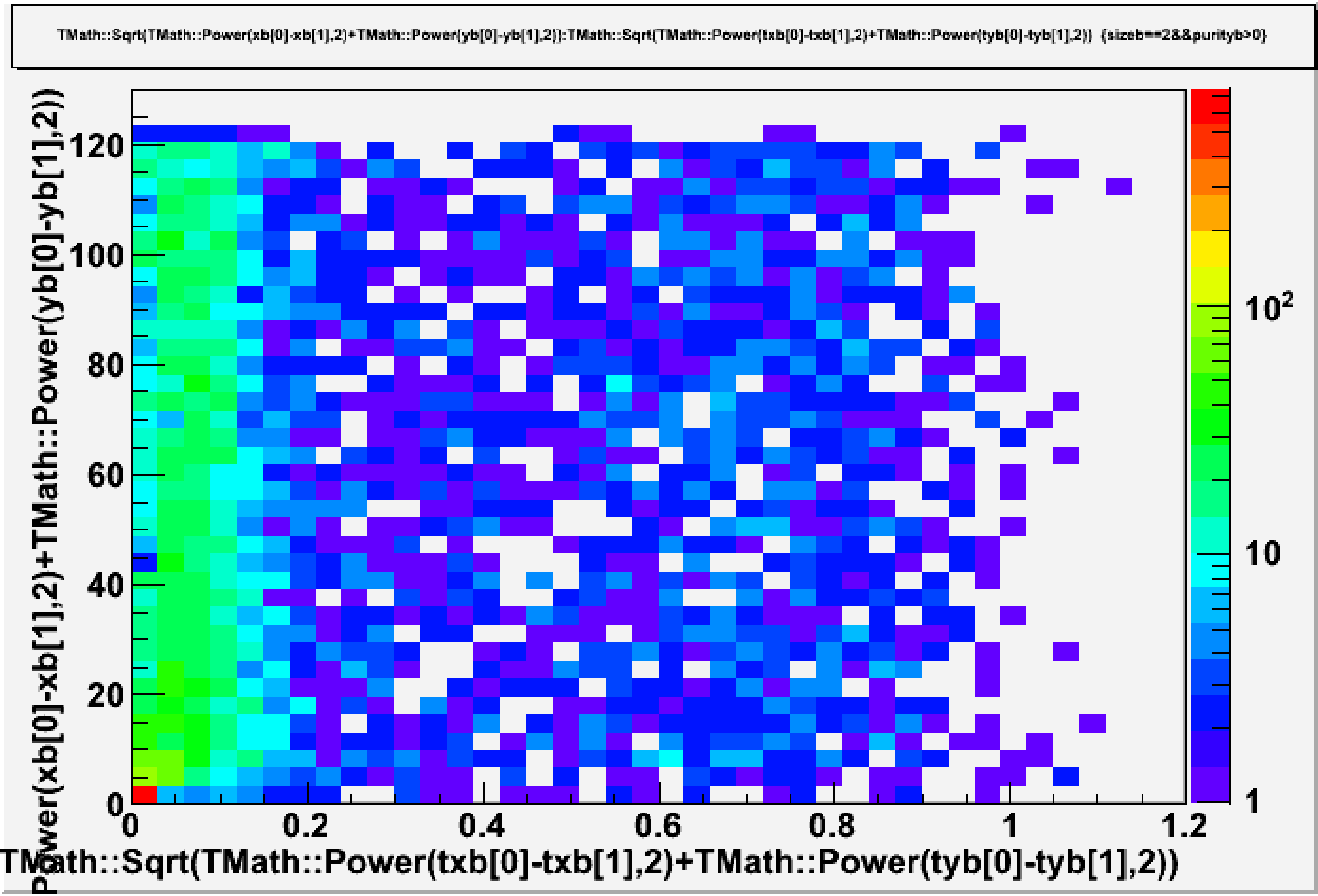
- We try to keep a constant BT density level of not more than 40BT/mm2
- To get this, we have the EdbPVRec patterns object, which we use for checks:
- We calculate BT density plate by plate and can adapt the Quality Cut plate by plate accordingly.

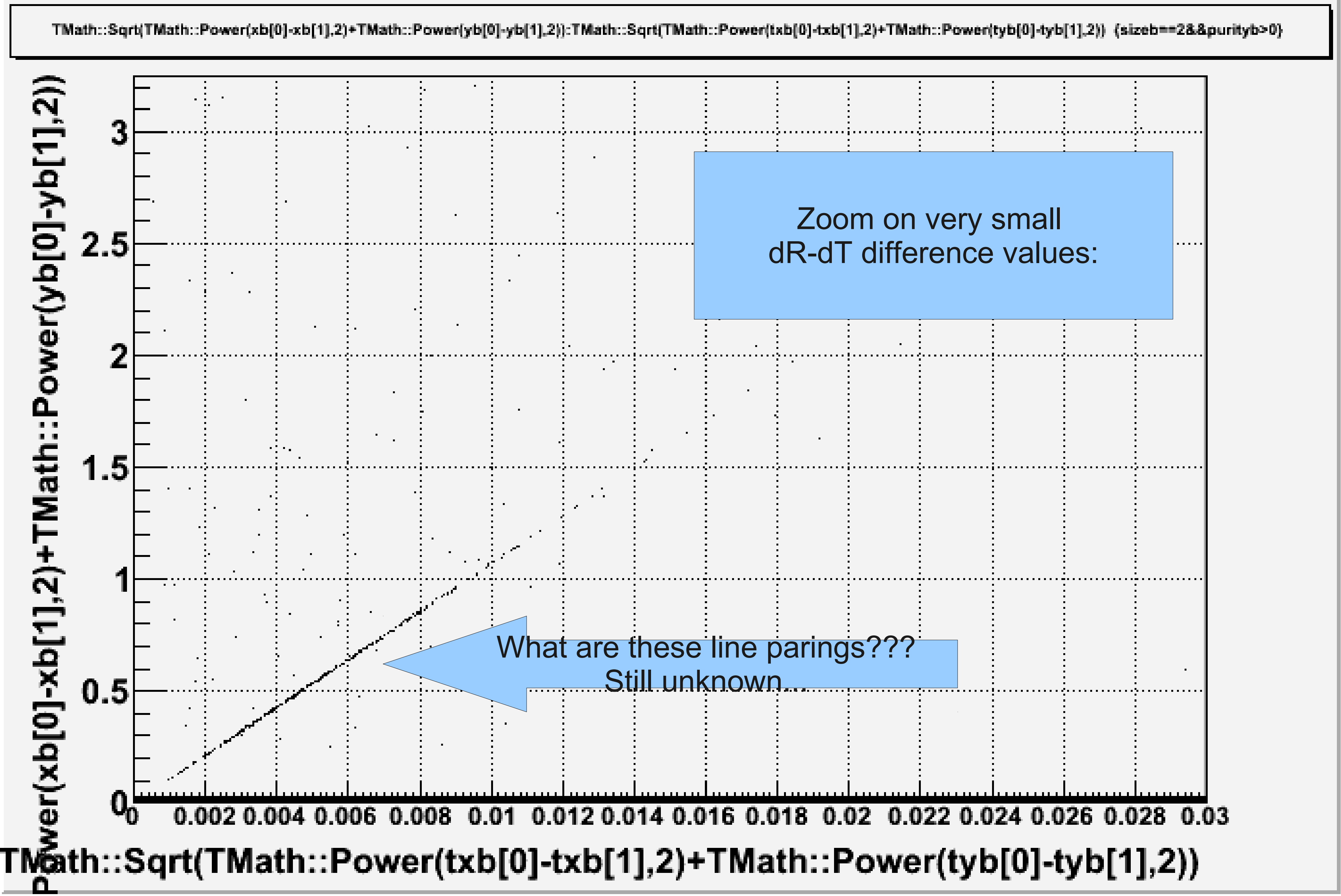
EdbPVRQuality

- This class does BG cleaning... How it works:
- EdbPVRQuality* Qualityobject = new EdbPVRQuality(gAli);
  - gAli is the filled EdbPVRec object. Usually this implicates the data coming from cp.root files.
  - LinkedTracks.root EdbPVRec object usually is already very „clean“, i.e. that linked tracks are often „good“ (physically) objects.
- Qualityobject->Print();
- Qualityobject->Help();

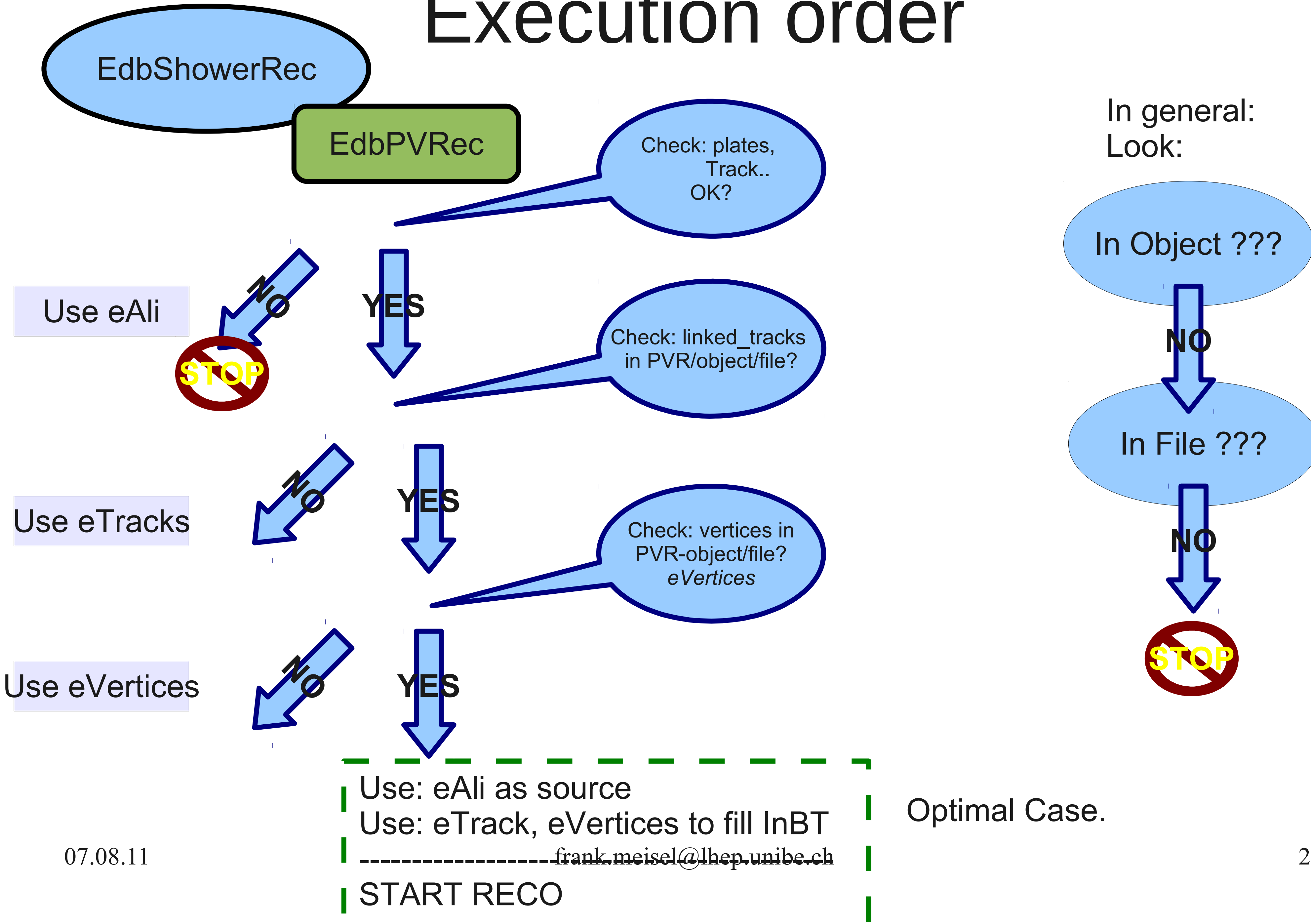
EdbPVRQuality Remove FakeDoublets.

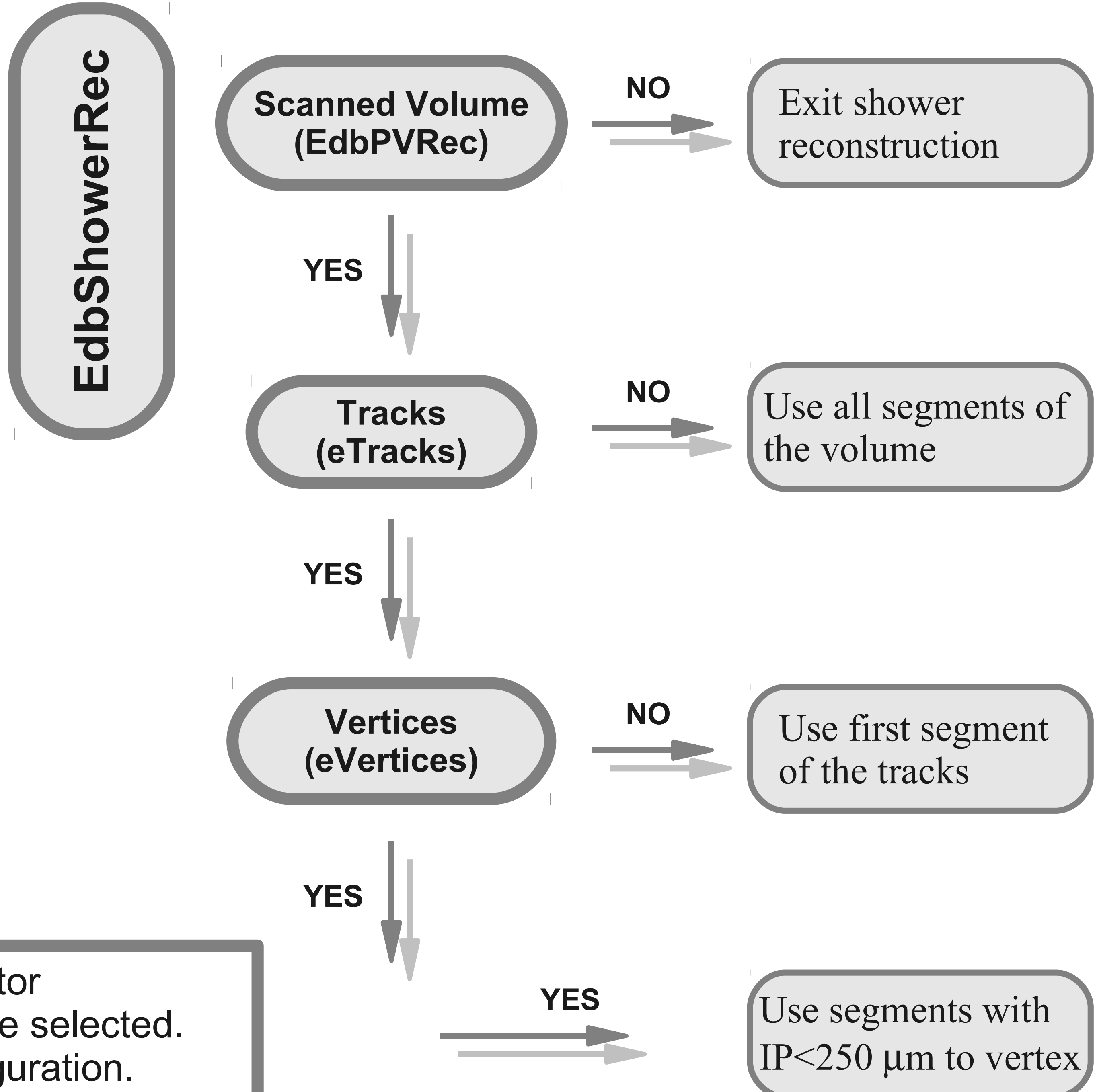
- From Scanning we know that by view overlap of the microscope there is fake double Bts:
  - One BT scanned in two views appears (due to Sysal bending correction function) as two slightly shifted different Bts!
  - This we dont want, since several reasons. One of them is for the GammaPairSearch: The pair algo might take these as a e+e—Pair.
- Those duplicated tracks show a distinct behaviour in the dR-dT plane.
- Plot: photon pairs and instrumental BG doublets in ther dR-dT plane
-
- EdbPVRQuality* Qualityobject = new EdbPVRQuality(gAli);
- Qualityobject->Remove_DoubleBT();
- Returns a new EdbPVRec Object, where these doublets are eliminated.


TMath::Sqrt(TMath::Power(xb[0]-xb[1],2)+TMath::Power(yb[0]-yb[1],2)):TMath::Sqrt(TMath::Power(txb[0]-txb[1],2)+TMath::Power(tyb[0]-tyb[1],2)) {sizeb==2&&purityb>0}
120
100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
1.2
10^2
10
1
TMath::Sqrt(TMath::Power(txb[0]-txb[1],2)+TMath::Power(tyb[0]-tyb[1],2))
Power(xb[0]-xb[1],2)+TMath::Power(yb[0]-yb[1],2))

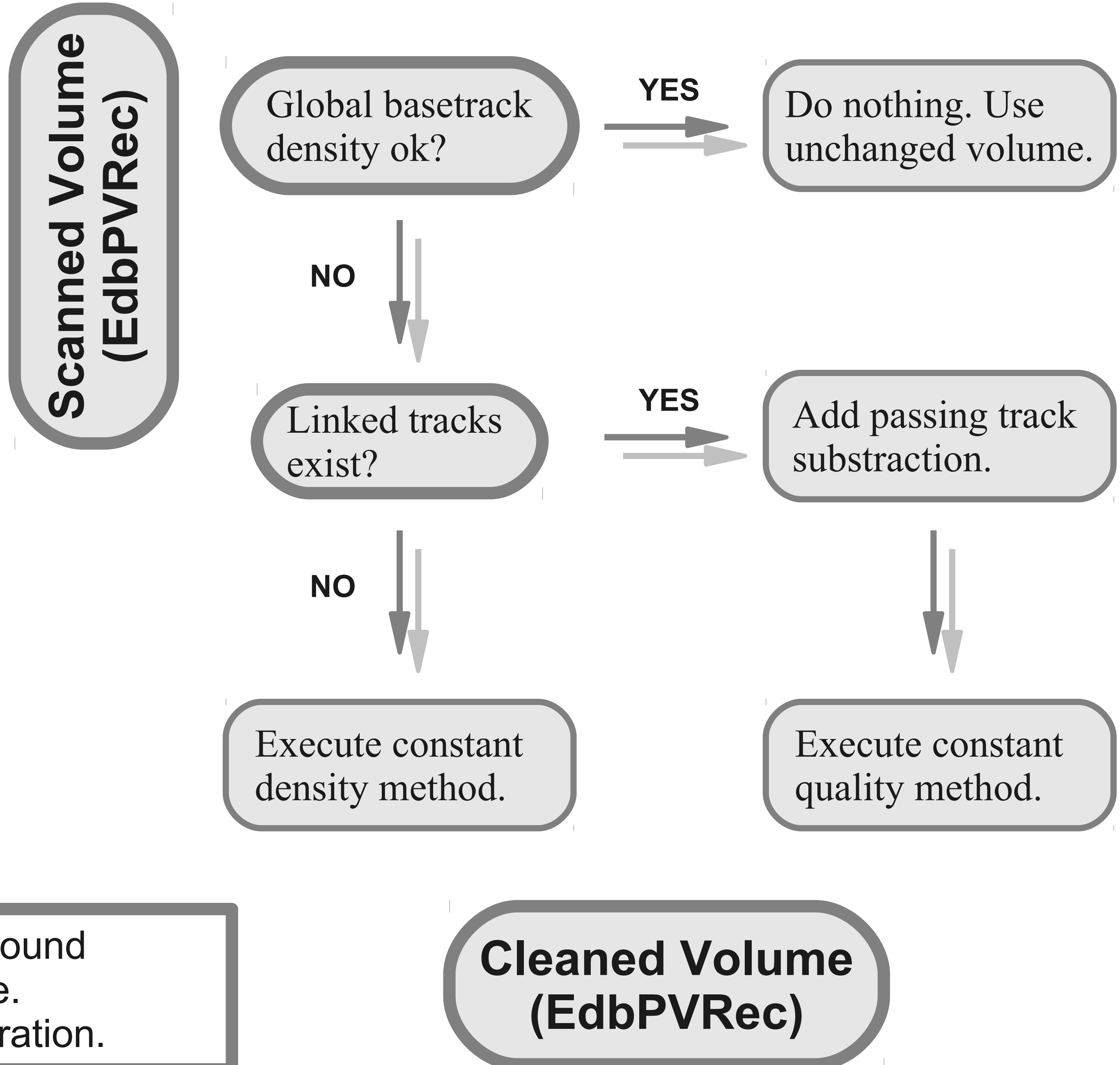

TMath::Sqrt(TMath::Power(xb[0]-xb[1],2)+TMath::Power(yb[0]-yb[1],2)):TMath::Sqrt(TMath::Power(txb[0]-txb[1],2)+TMath::Power(tyb[0]-tyb[1],2)) {sizeb==2&&purityb>0}
Zoom on very small
dR-dT difference values:
What are these line parings???
Still unknown...
TMath::Sqrt(TMath::Power(txb[0]-txb[1],2)+TMath::Power(tyb[0]-tyb[1],2))
Power(xb[0]-xb[1],2)+TMath::Power(yb[0]-yb[1],2))
0
0.5
1
1.5
2
2.5
3
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
0.022
0.024
0.026
0.028
0.03

Execution order



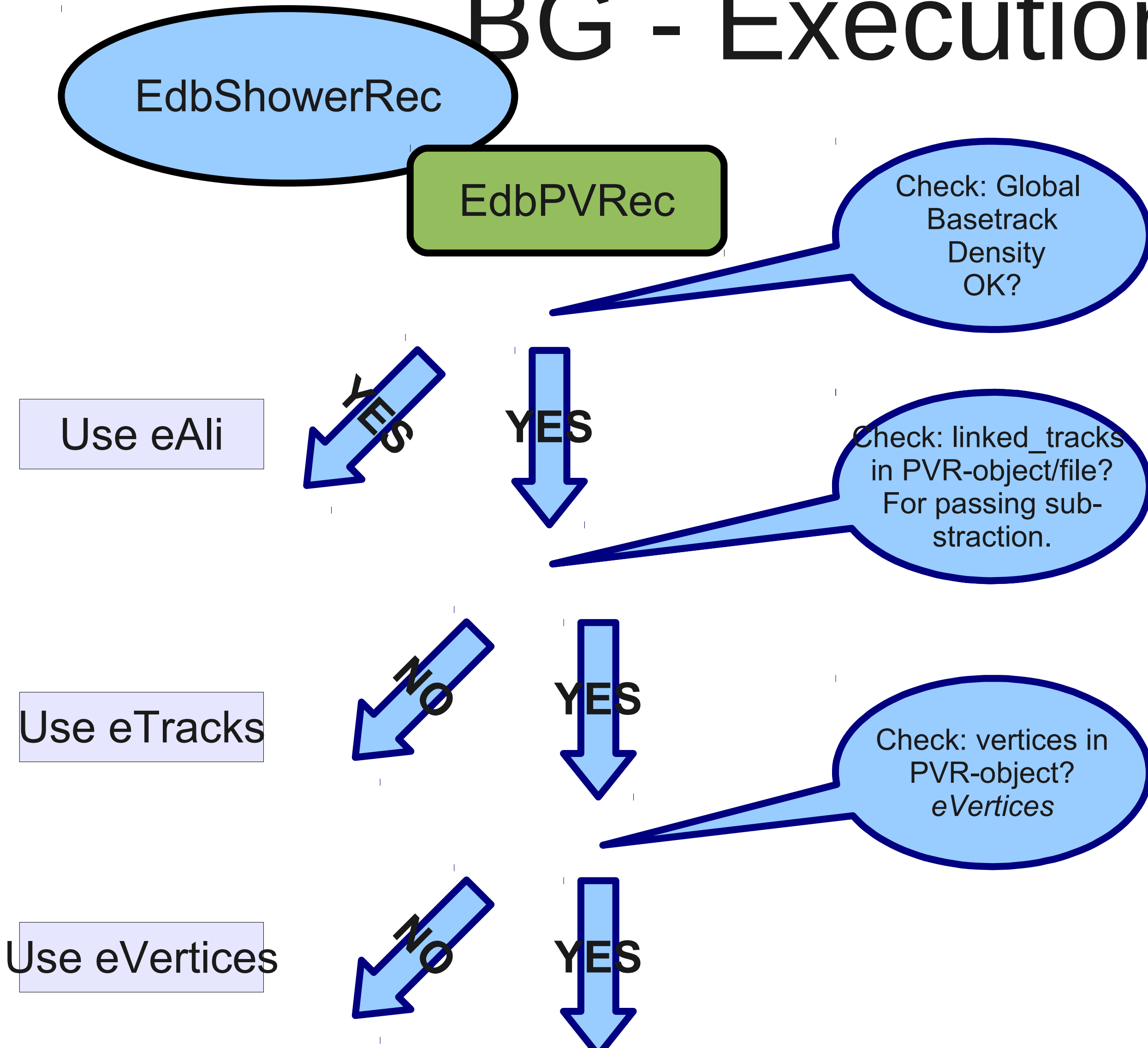


How the Initiator Basetracks are selected. Default Configuration.



How the background cleaning is done.
Default Configuration.

BG - Execution order



Optimal Case.

Use: Constant QualityCut
Use: Passing Track Substraction, Double BT substraction.

SET NEW EdbPVRec FOR RECONSTRUCTION