ALICE Physics Week
May 2013 – Padova –

Event mixing: basics and how-to

Andrea Rossi (CERN)
Sandro Bjelogrlić (Utrecht)

APW - Analysis Tutorial session, 24/05/2013

Outline

- Introduction
- Biases from detector acceptance, dead zones....
  - *Bias from finite pseudorapidity acceptance*
  - *Bias from vertex position*
  - *Bias from inhomogenuous azimuthal acceptance*
- Event mixing in the Aliroot framework
  - *Taking the minimum information*
  - *The event pool*
  - *Event mixing in your task*
  - *The offline correction*
  - *Example from di-hadron correlations*
- Summary

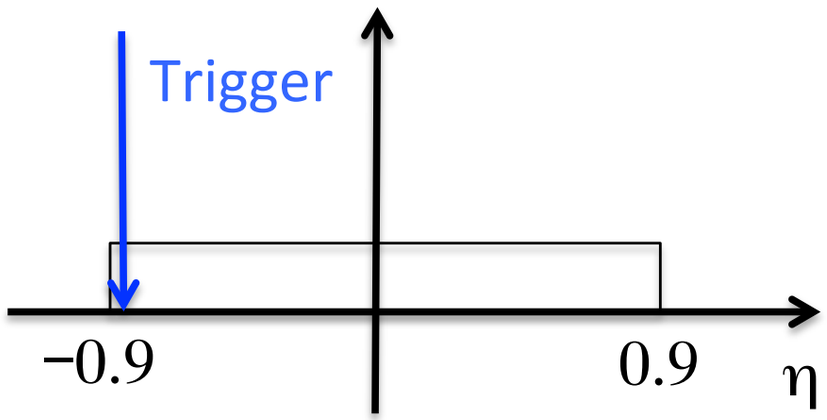
Introduction

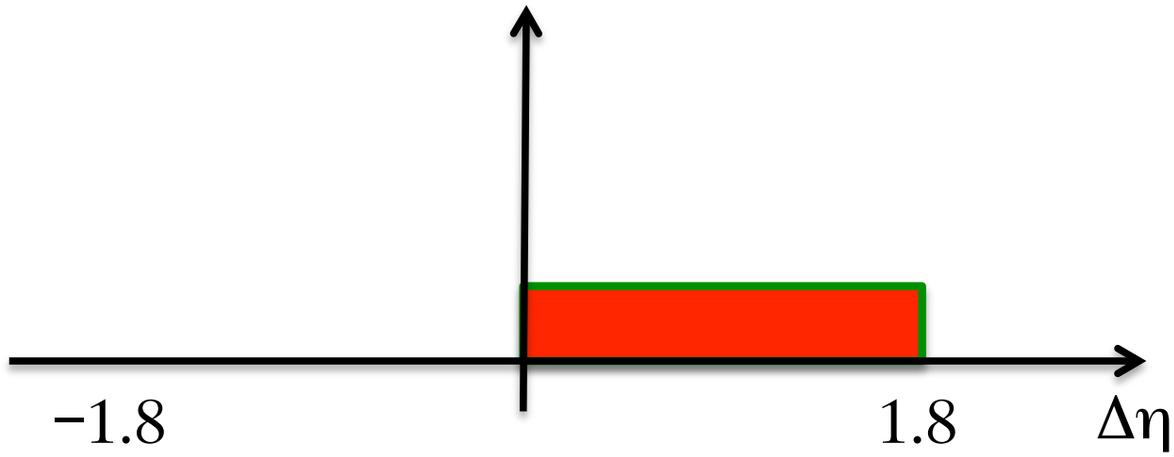
- **Event mixing - why?**
  - ***Studying distribution from event mixing is an excellent tool for estimating (and correcting for) the effects of non physical distributions to the physics we want to study***

- **Event mixing - when?**
  - ***Correlation analyses: correct for the biases of non physical correlations: two-particle acceptance, dead areas of the detector***
  - ***Combinatorial background in invariant mass analyses***
  - ***Flow analyses?***

Bias from finite η acceptance

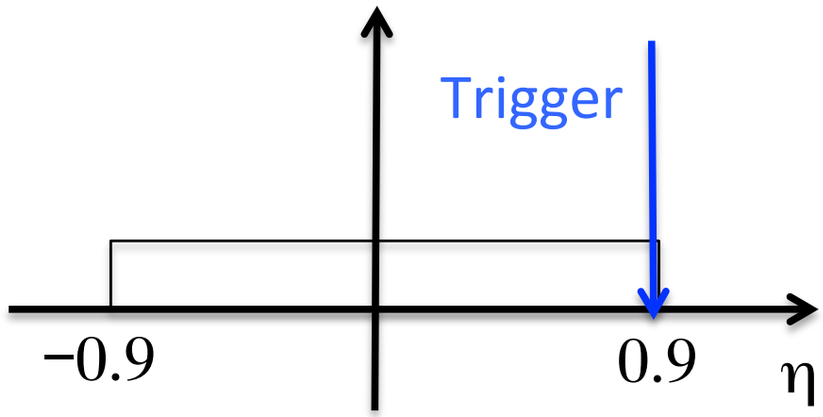
Assume a (not realistic) flat η distribution of the particles in the event over a very wide (ideally infinite) range
Pairing trigger and associated particles, a flat Δη distribution is obtained
Finite detector acceptance in pseudorapidity η (e.g. 0.9<η<0.9)

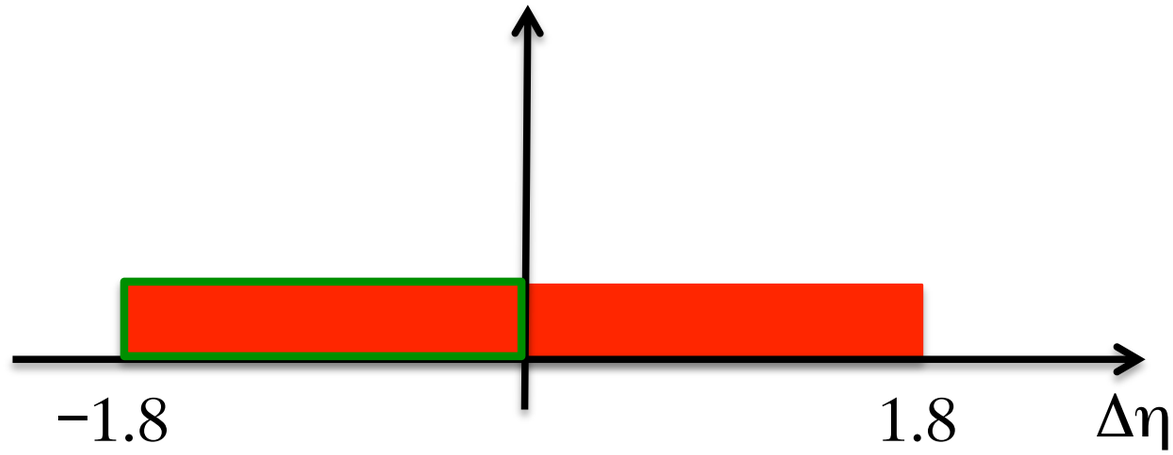




Bias from finite η acceptance

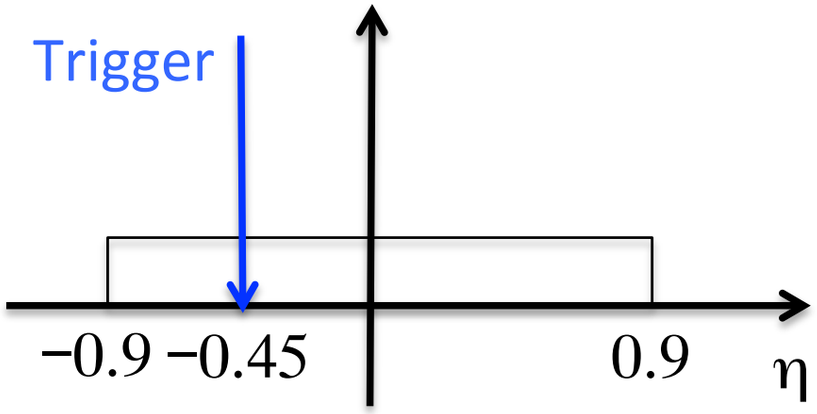
Assume a (not realistic) flat η distribution of the particles in the event over a very wide (ideally infinite) range
Pairing trigger and associated particles, a flat Δη distribution is obtained
Finite detector acceptance in pseudorapidity η (e.g. 0.9<η<0.9)

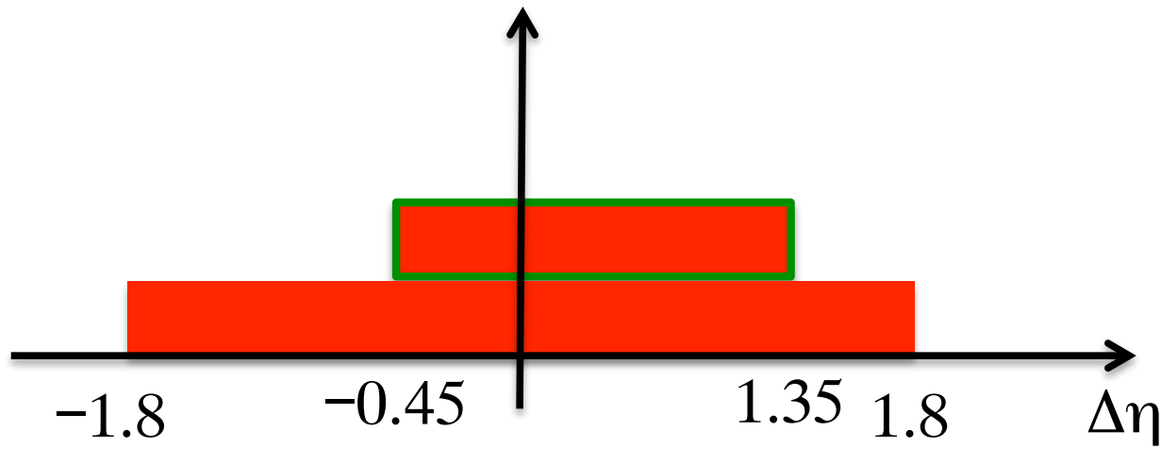




Bias from finite η acceptance

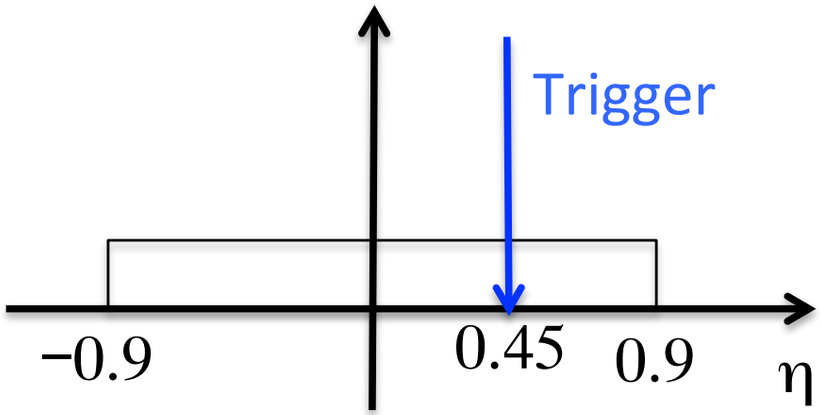
Assume a (not realistic) flat η distribution of the particles in the event over a very wide (ideally infinite) range
Pairing trigger and associated particles, a flat Δη distribution is obtained
Finite detector acceptance in pseudorapidity η (e.g. 0.9<η<0.9)

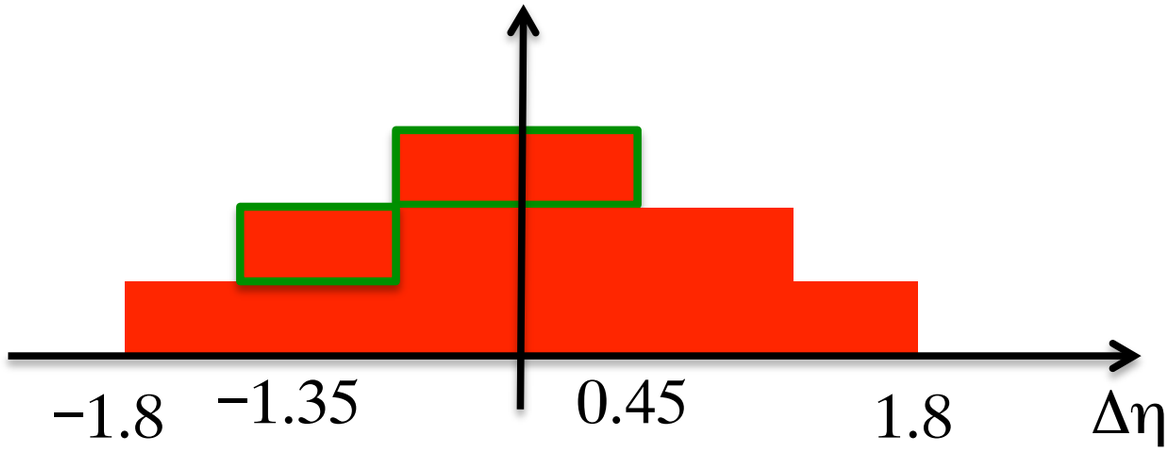




Bias from finite η acceptance

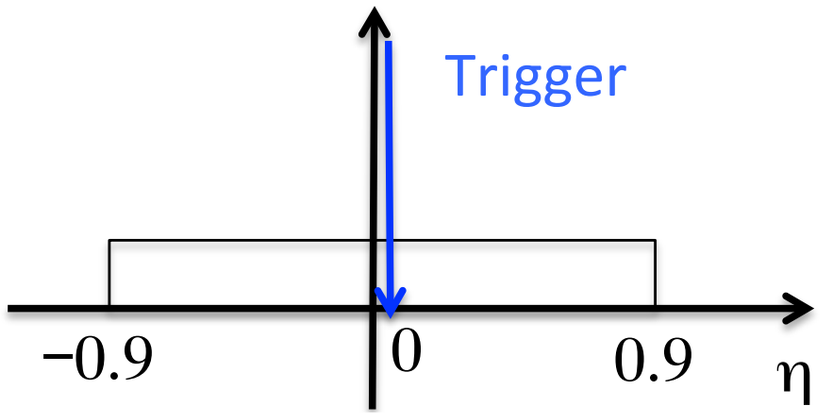
Assume a (not realistic) flat η distribution of the particles in the event over a very wide (ideally infinite) range
Pairing trigger and associated particles, a flat Δη distribution is obtained
Finite detector acceptance in pseudorapidity η (e.g. 0.9<η<0.9)

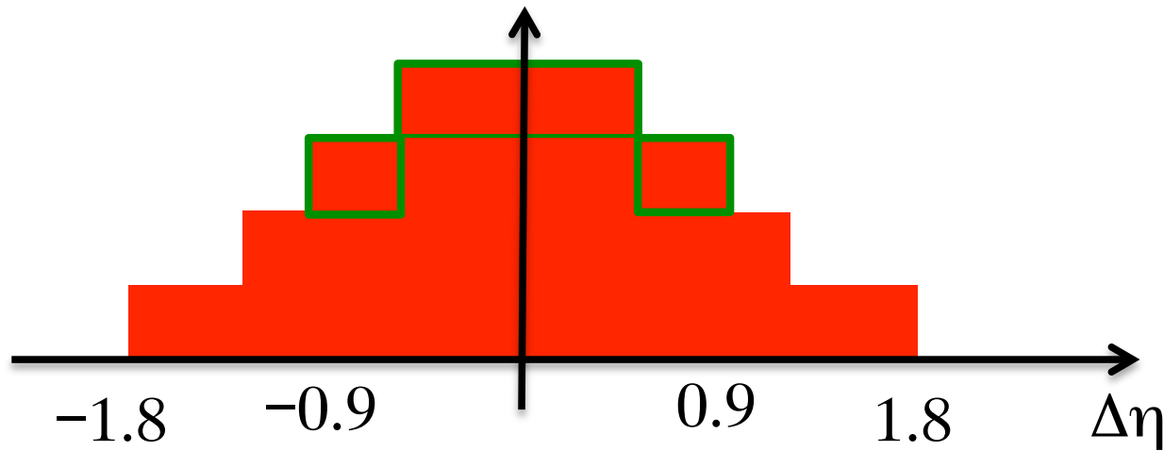




Bias from finite η acceptance

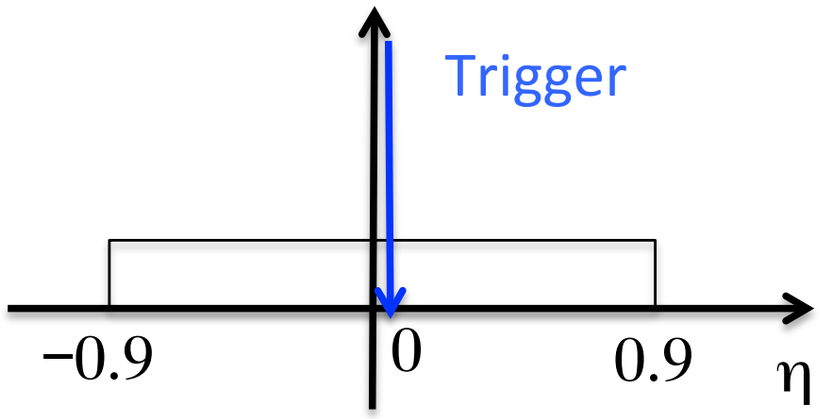
Assume a (not realistic) flat η distribution of the particles in the event over a very wide (ideally infinite) range
Pairing trigger and associated particles, a flat Δη distribution is obtained
Finite detector acceptance in pseudorapidity η (e.g. 0.9<η<0.9)

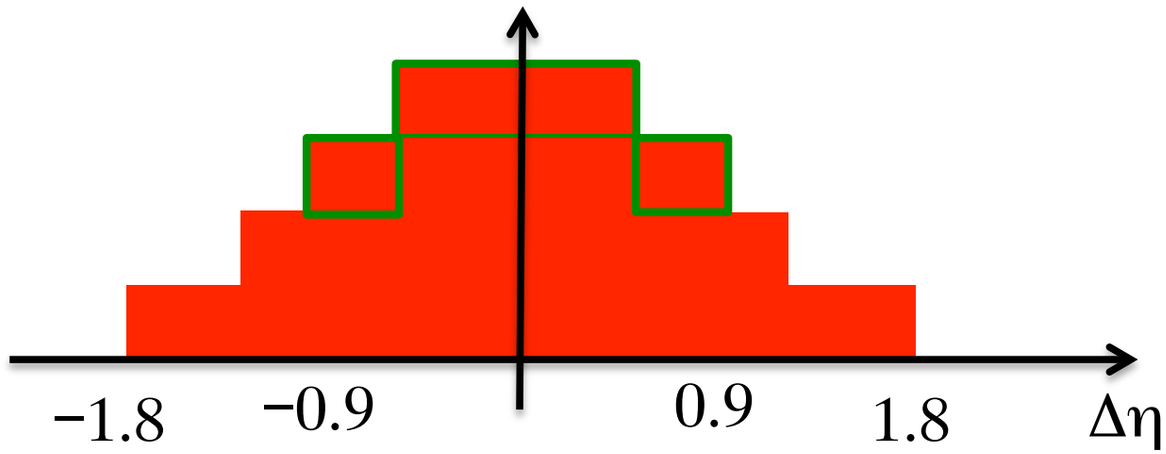




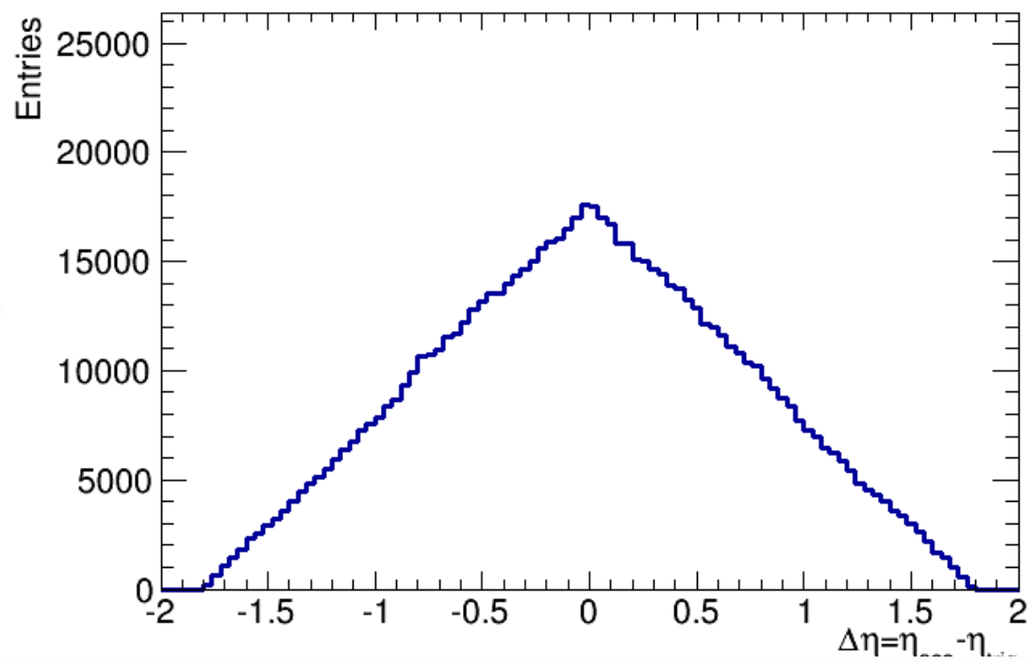
Bias from finite η acceptance

Assume a (not realistic) flat η distribution of the particles in the event over a very wide (ideally infinite) range
Pairing trigger and associated particles, a flat Δη distribution is obtained
Finite detector acceptance in pseudorapidity η (e.g. 0.9<η<0.9)





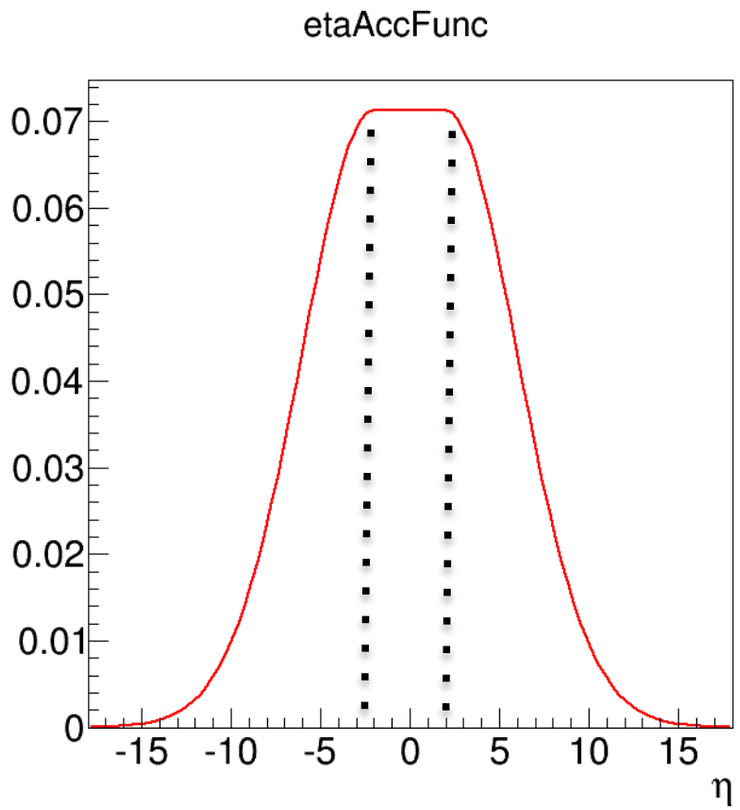
Not in steps (with a toy model)

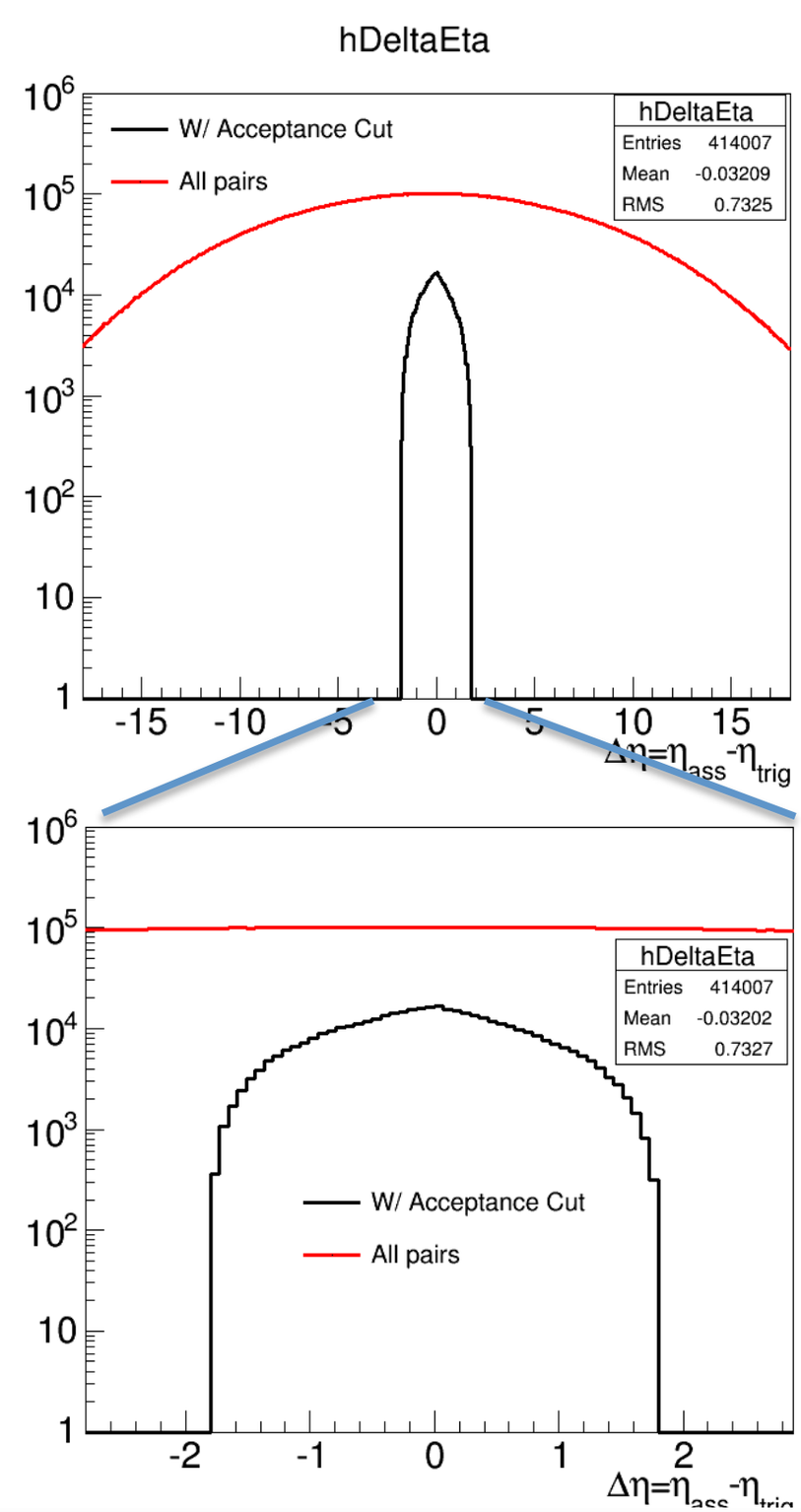


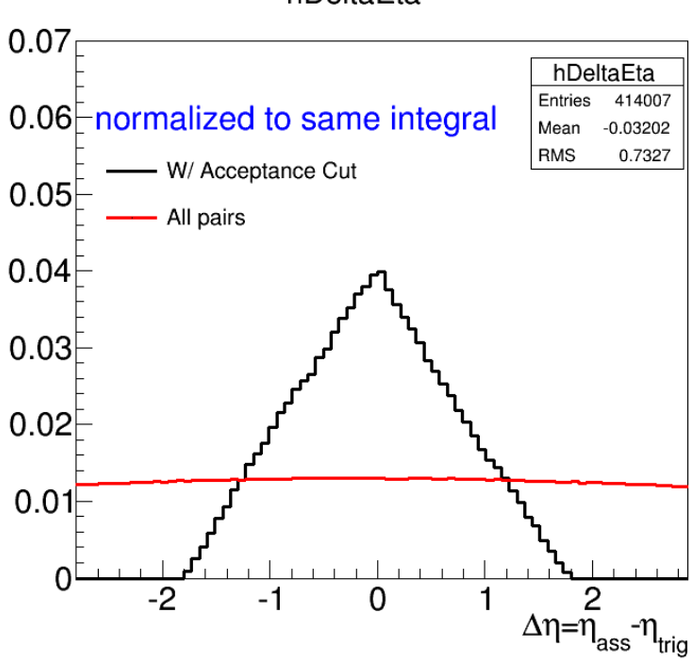
Slightly more realistic distribution

Particle eta distribution: const(-2,2)+Gaussian tails with σ=4

Acceptance: -0.9,0.9



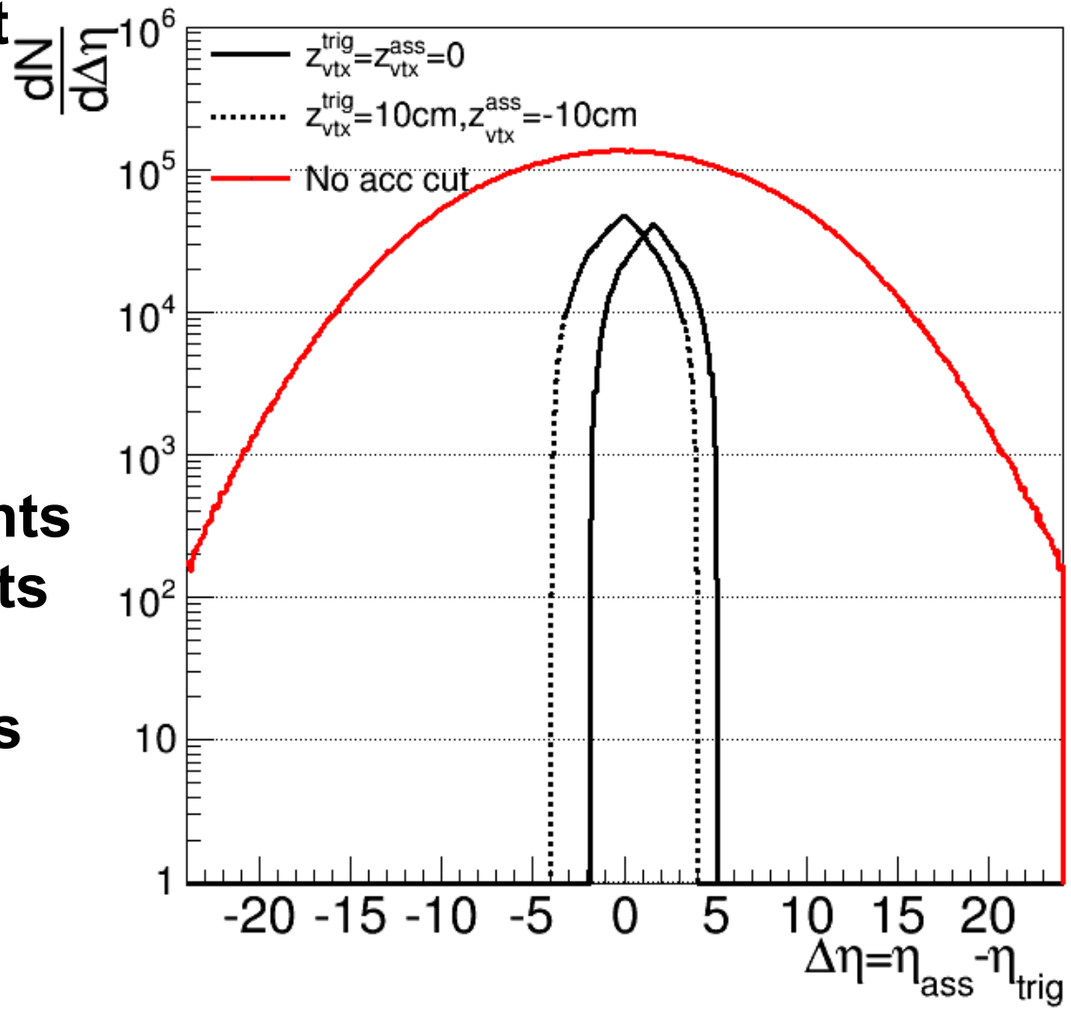




Andrea Rossi,Sandro Bjelogrlić

Bias from vertex position

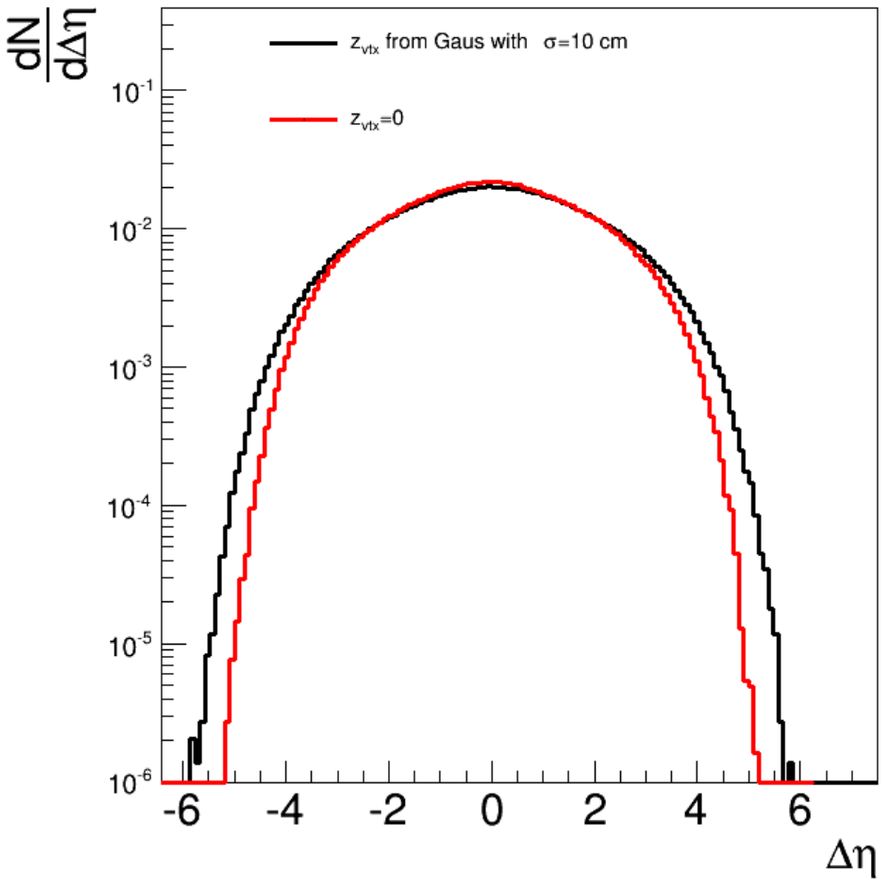
Event vertex position in z (along beam line) slightly change the acceptance window (not that much for TPC tracks and typical vtx z values)
Effect more dramatic if we consider SPD acceptance:
(z_{vtx}=0 -> |η|<1.95, z_{vtx}=10cm -> -2.5<η<0.9)

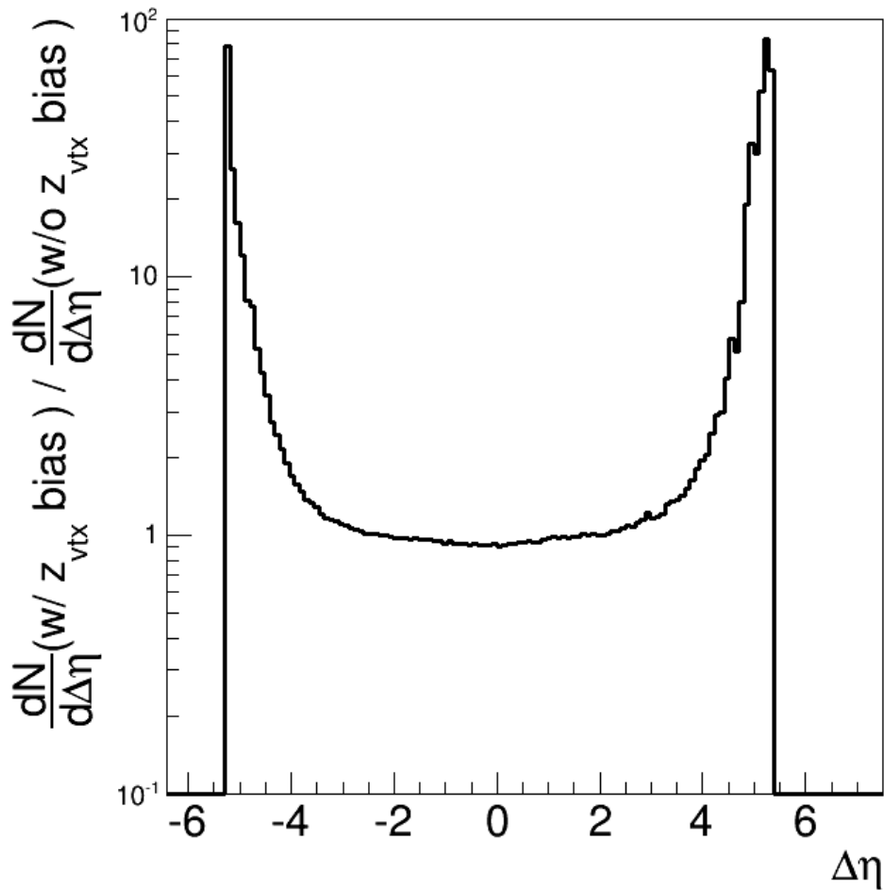


We can try to correlate (trigger) tracks from events with z_{vtx}=10 cm to (associated) tracks from events with z_{vtx}=-10 cm to see how this affects the Δη distribution we would recover from Mixed events (this is an extreme scenario and using SPD acceptance is also not a realistic case)

Bias from vertex position

Now we can see the Δη distribution obtained by taking trigger and associated particles from events with z_{vtx} distributed according to a Gaussian with σ=10 cm

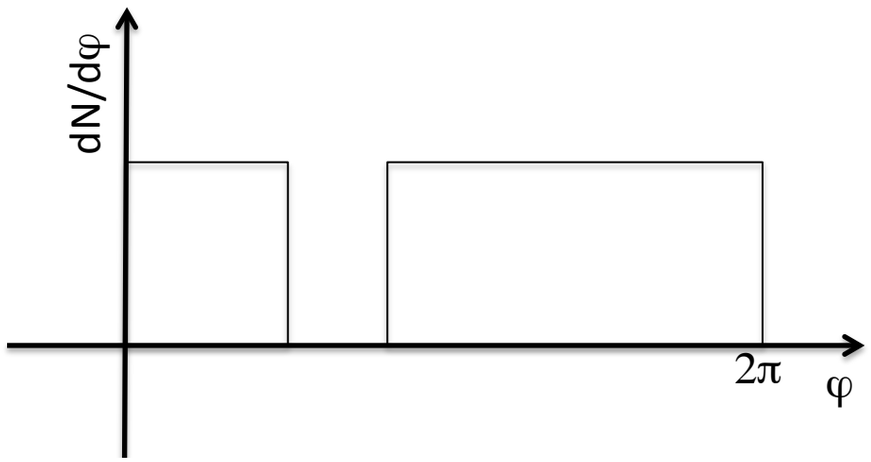




If we want to correct for limited detector acceptance with Δη distribution recovered from mixed events we have to mix events with same(similar) z_{vtx}

Bias from azimuthal inhomogeneity

Suppose that we have a hole in the detector in phi at e.g. phi=45 degrees.
And we assume a flat azimuthal distribution of the particles in the event



Let's try to make pairs

If we can sample the Dj distribution obtained by moving the trigger particle along j

Bias from azimuthal inhomogeneity

Assume a flat azimuthal distribution of the particles in the event.
-> flat Δϕ distribution expected if full and homogeneous azimuthal coverage.
Suppose that we have a "hole" in the detector in (e.g.) π/4<ϕ<π/2: the track ϕ distribution is depleted. The resulting distribution can be viewed as a flat distribution subtracted of a "hole".

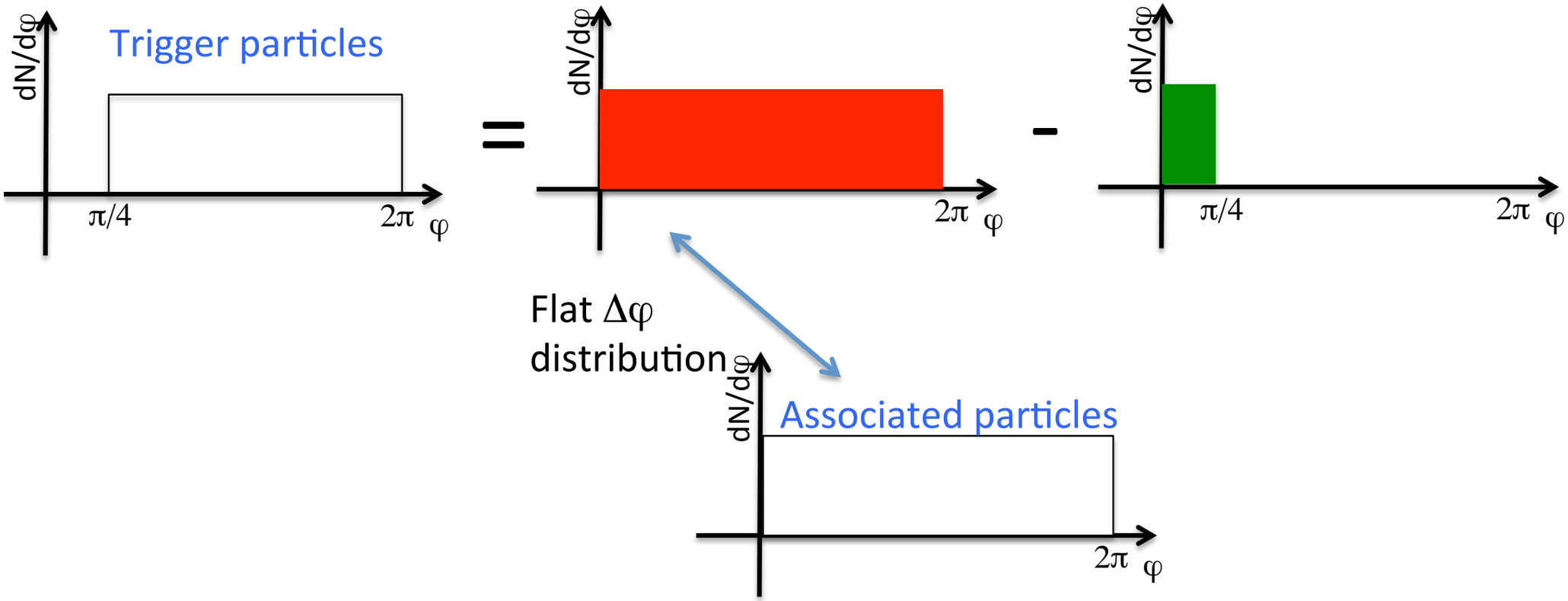
We can look at expected Δϕ distribution by spanning different trigger track ϕ values.
First, with a change of coordinate (ϕ'=ϕ−π/4) we have

Trivially, the single hole case is equivalent also to that in which we have a limited acceptance (small Δφ coverage)

Andrea Rossi,Sandro Bjelogrlić

Bias from azimuthal inhomogeneity

Let's do all pairs **assuming that only trigger particles have a limited ϕ acceptance**, while associated particles have a full azimuthal coverage (e.g. trigger particles from EMCAL, associated from TPC). We can look at it as if the "real" result can be obtained by sampling the trigger particles from the red distribution and then subtract what obtained by sampling them from the green (hole).

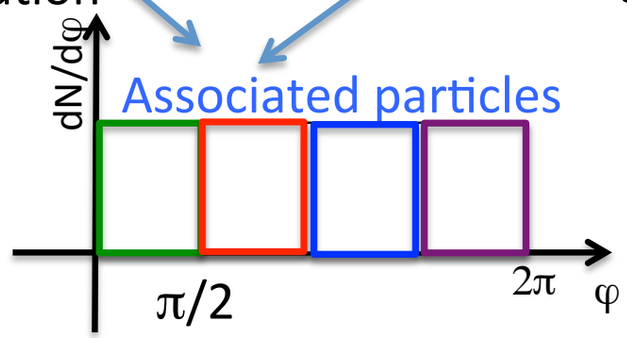


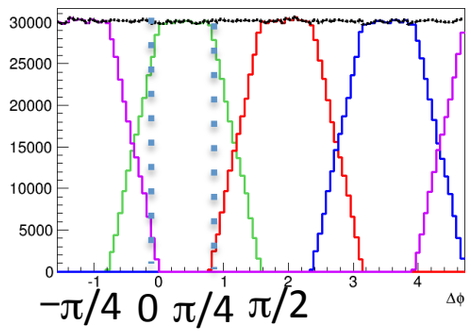
Bias from azimuthal inhomogeneity

Let's do all pairs **assuming that only trigger particles have a limited ϕ acceptance**, while associated particles have a full azimuthal coverage (e.g. trigger particles from EMCAL, associated from TPC). We can look at it as if the "real" result can be obtained by sampling the trigger particles from the red distribution and then subtract what obtained by sampling them from the green (hole).

Flat Δφ distribution

Also gives a flat Δφ distribution





⇨ **Flat Δφ distribution**

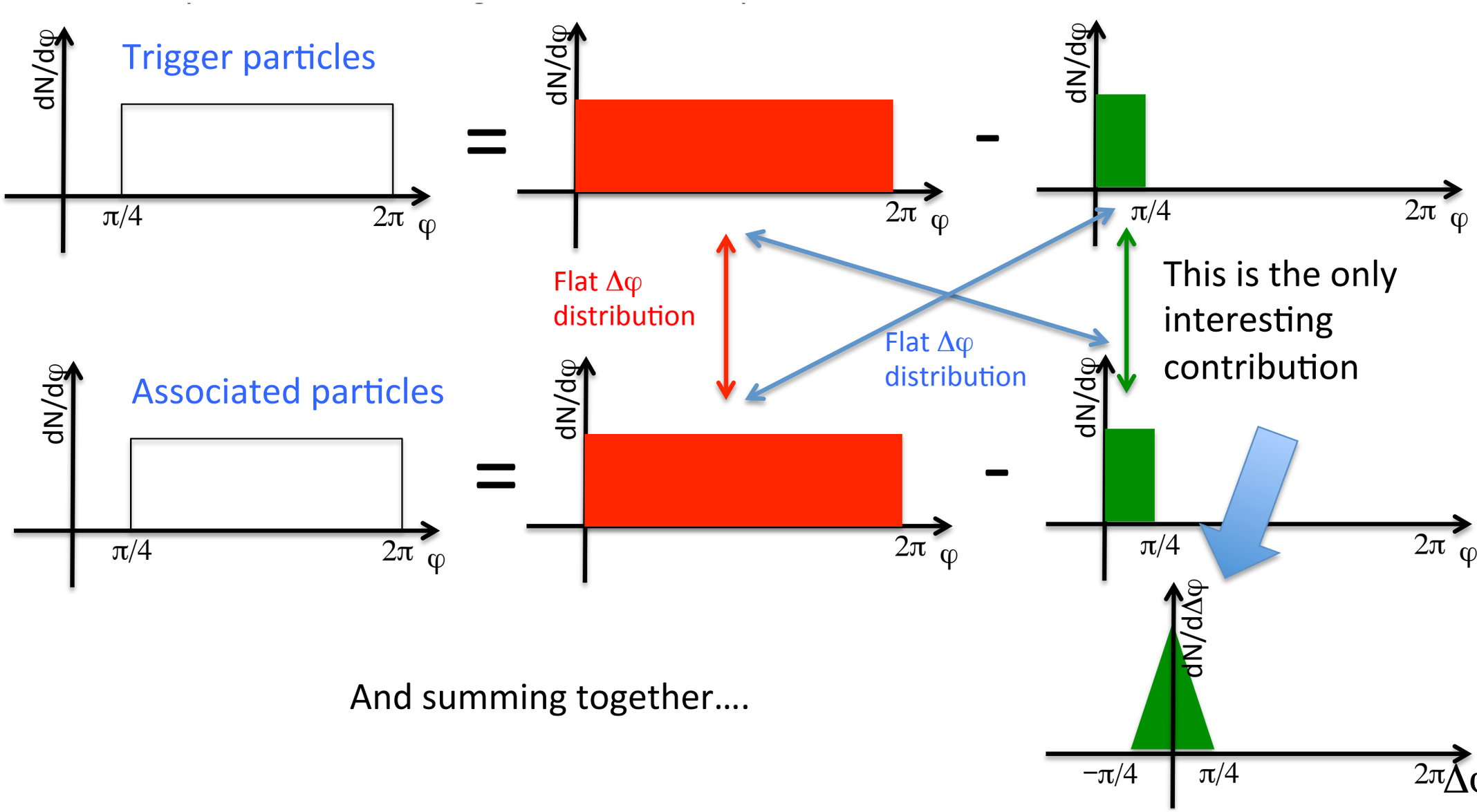
(as trivially expected from symmetry, there is not any preferential Δφ region)

The same if the "hole" is on the associated particle only

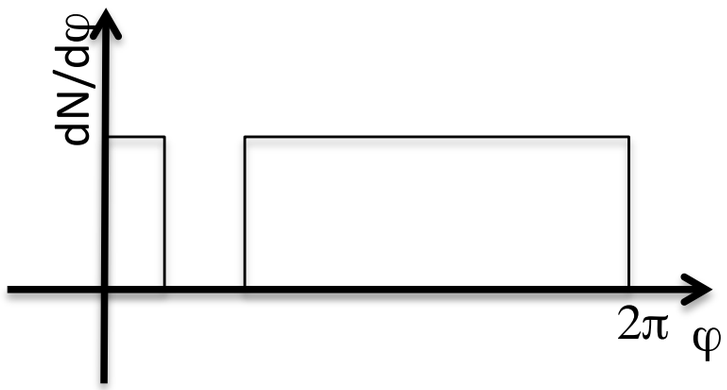
Andrea Rossi,Sandro Bjelogrlić

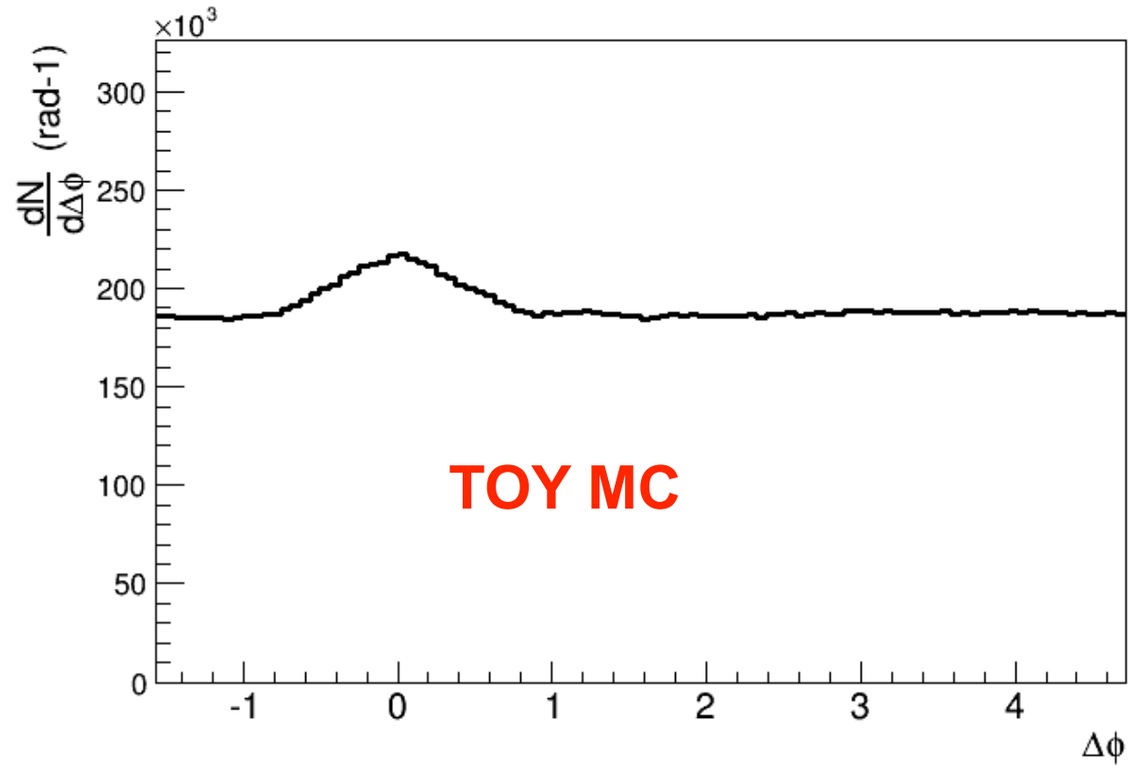
Bias from azimuthal inhomogeneity

2) Now we can consider the case in which the trigger and associated particles have the same acceptance. Proceeding in the same way as before:



Bias from azimuthal inhomogeneity

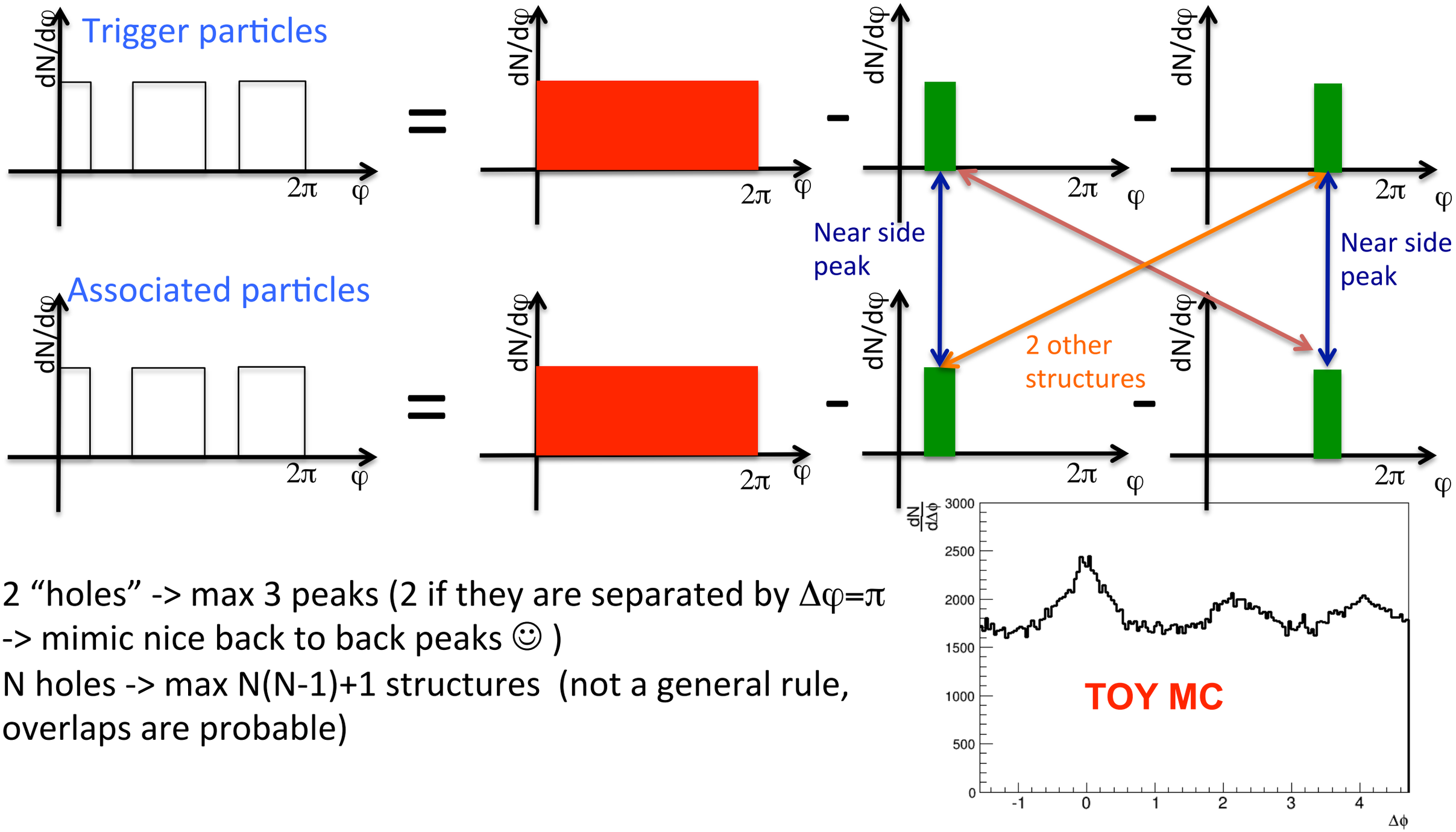




A “hole” in ϕ produces a structure in the near side
Holes always enhance the near side!

Bias from azimuthal inhomogeneity

What matters is the width and the relative positions of the holes



2 “holes” -> max 3 peaks (2 if they are separated by $\Delta\varphi=\pi$ -> mimic nice back to back peaks ☺)

N holes -> max N(N-1)+1 structures (not a general rule, overlaps are probable)

Andrea Rossi,Sandro Bjelogrlić

Time for some code

EVENT MIXING in the AliRoot Framework

Event mixing: how to

Reducing the track to the minimum, i.e.

Create an object that will store only the necessary information

```
class MyTrack : public AliVParticle
{
public:
    MyTrack(Double_t eta, Double_t phi, Double_t pt);
       ~ MyTrack();

    // kinematics
     virtual Double_t Pt()       const { return fpT; }
     virtual Double_t Phi()      const { return fPhi; }
     virtual Double_t Eta()      const { return fEta; }
 }


MyTrack:: MyTrack(Double_t eta, Double_t phi, Double_t pt) :
fEta(eta),
fPhi(phi),
fpT(pt)
{
// default constructor
}
```

Event mixing: how to

Reducing the track to the minimum, i.e.

Create an object that will store only the necessary information

```
TObjArray *TrackArray = new TObjArray();
TrackArray->SetOwner(kTRUE);

// loop on all the tracks
for(Int_t iTrack = 0; iTrack<AOD->GetNTracks(); iTrack++){

    AliAODTrack* track = AOD->GetTrack(iTrack);

    apply your selections to the track

    TrackArray->Add(new MyTrack(track->Eta(), track->Phi(), track ->pT));
}
```

The track array will be passed to the event pool at the end of the analysis

makes sure you actually mix!

The event pool

The AliEventPoolManager: definition of the pools

AliEventPoolManager(MaxNofEvents,trackDepth,nCentralityBins,(Double_t*)centralityBins,nZvtxBins,(Double_t*) vertexBins);

The event pool

The AliEventPoolManager: definition of the pools

Maximum number of events

AliEventPoolManager(MaxNofEvents,trackDepth,nCentralityBins,(Double_t*)centralityBins,nZvtxBins,(Double_t*) vertexBins);

Maximum number of tracks

Int_t MaxNofEvents = 1000;

Int_t trackDepth = 2000;

The event pool

The AliEventPoolManager: definition of the pools

Maximum number of events

Number of centrality bins

```
AliEventPoolManager(MaxNofEvents,trackDepth,nCentralityBins,(Double_t*)centralityBins,nZvtxBins,(Double_t*) vertexBins);
```

Maximum number of tracks

Centrality bins

```
Int_t MaxNofEvents   = 1000;

Int_t trackDepth = 2000;

Int_t nCentralityBins  = 15;
Double_t centralityBins[] = { 0, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.1 };
```

The event pool

The AliEventPoolManager: definition of the pools

Maximum number of events

Number of centrality bins

Number of zVtx bins

```
AliEventPoolManager(MaxNofEvents,trackDepth,nCentralityBins,(Double_t*)centralityBins,nZvtxBins,(Double_t*) vertexBins);
```

Maximum number of tracks

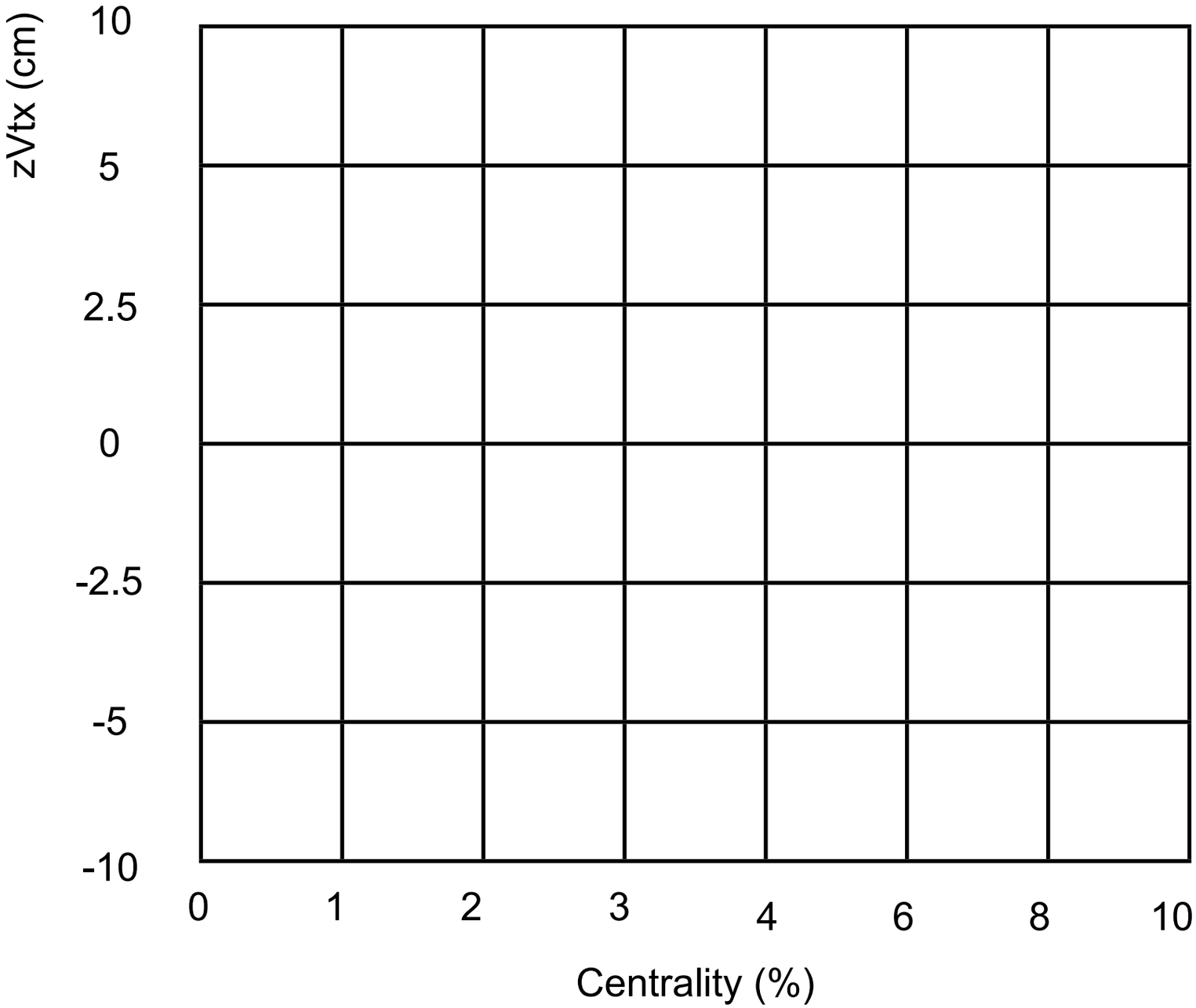
Centrality bins

zVtx bins

```
Int_t MaxNofEvents   = 1000;

Int_t trackDepth = 2000;

Int_t nCentralityBins  = 8;
Double_t centralityBins[] = { 0, 1, 2, 3, 4, 6, 8, 10 };

Int_t nZvtxBins  = 7;
Double_t vertexBins[] = { -10, -5, -2.5,0, 2.5, 5, 10};
```

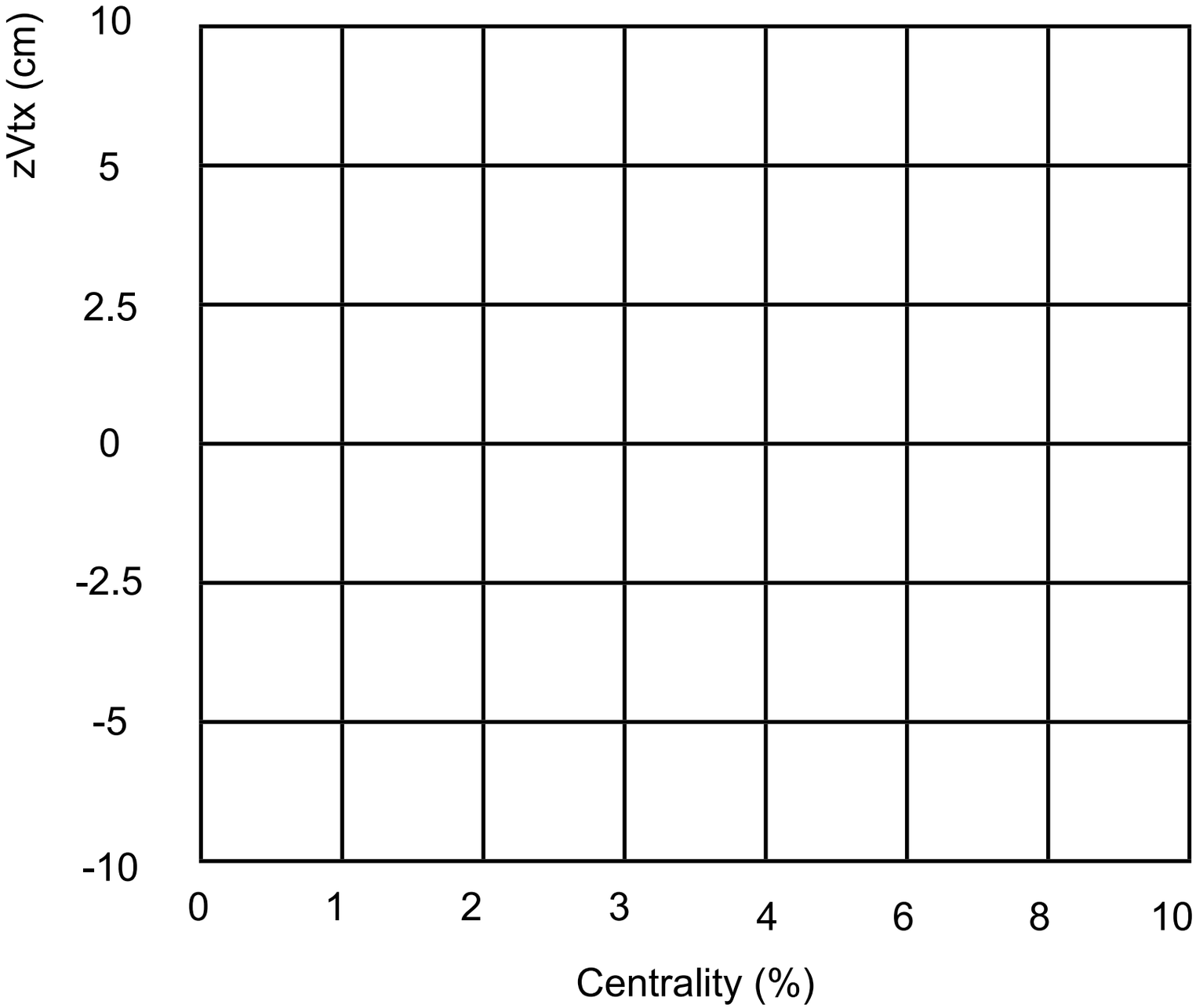
The event pool

The AliEventPoolManager: definition of the pools



The event pool

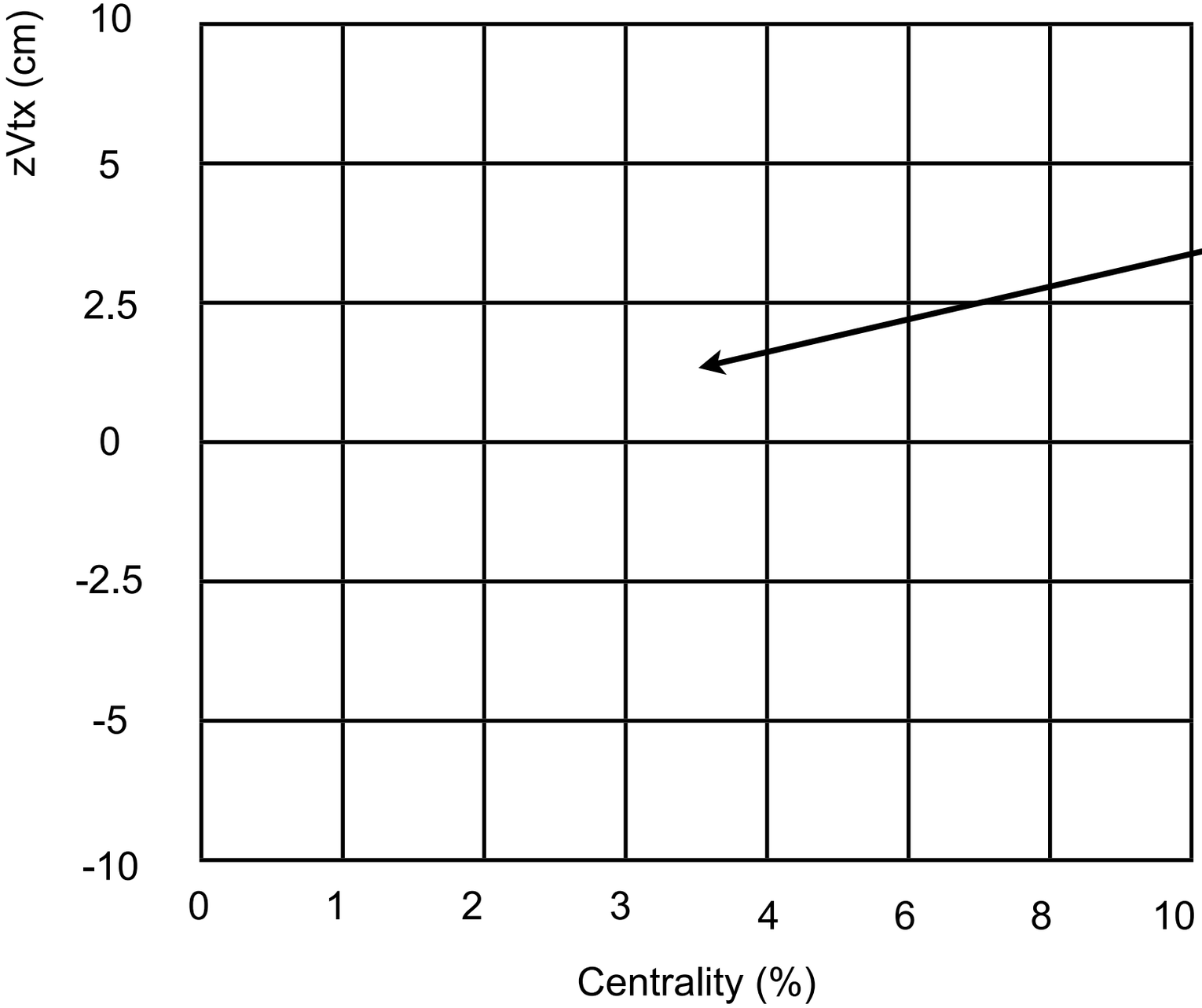
The AliEventPoolManager: definition of the pools



1) Get info from the Event
centrality
zVtx

The event pool

The AliEventPoolManager: definition of the pools

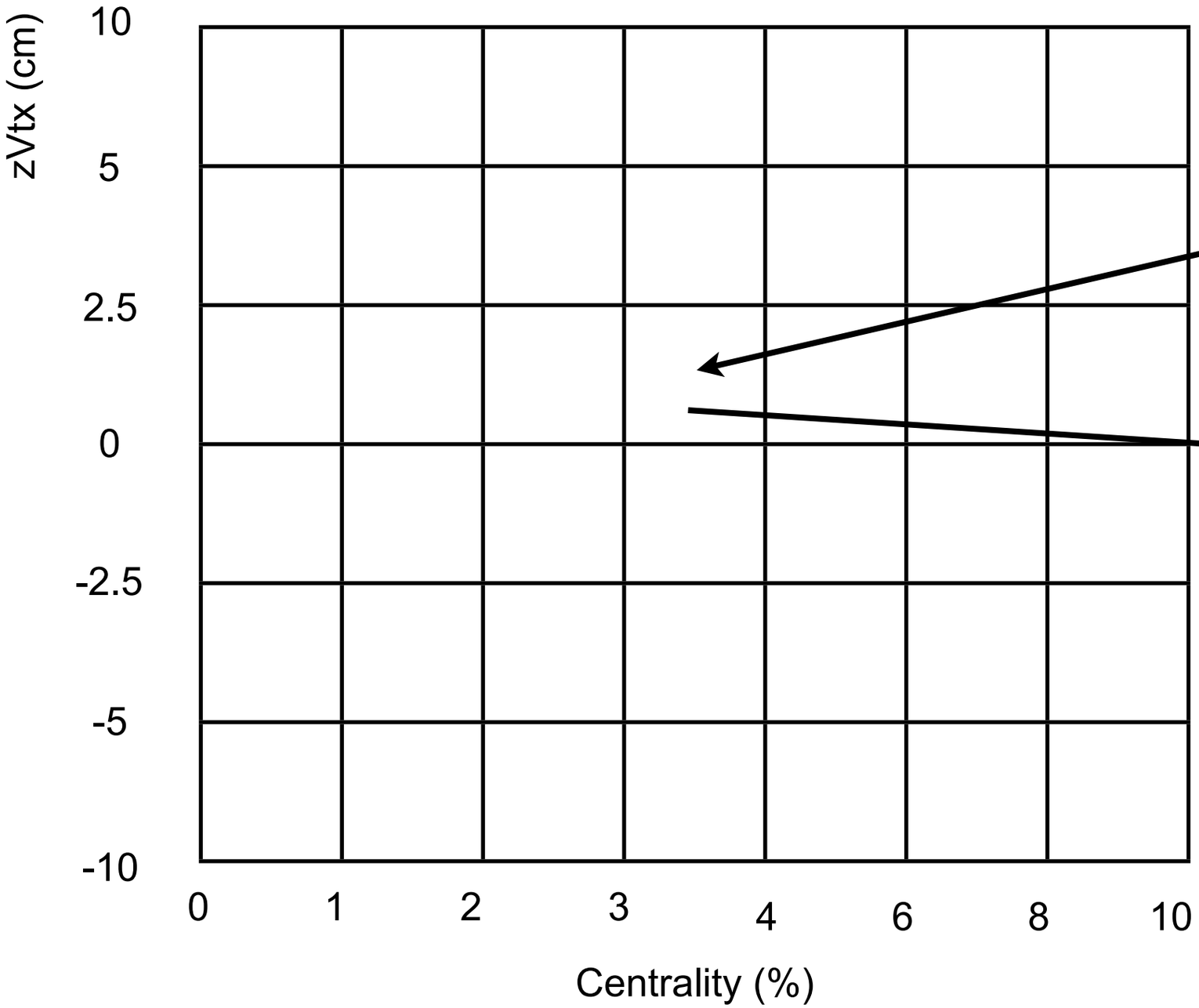


1) Get info from the Event
centrality
zVtx

example (centrality 3.5%, zVtx 1.4 cm)

The event pool

The AliEventPoolManager: definition of the pools



1) Get info from the Event
centrality
zVtx

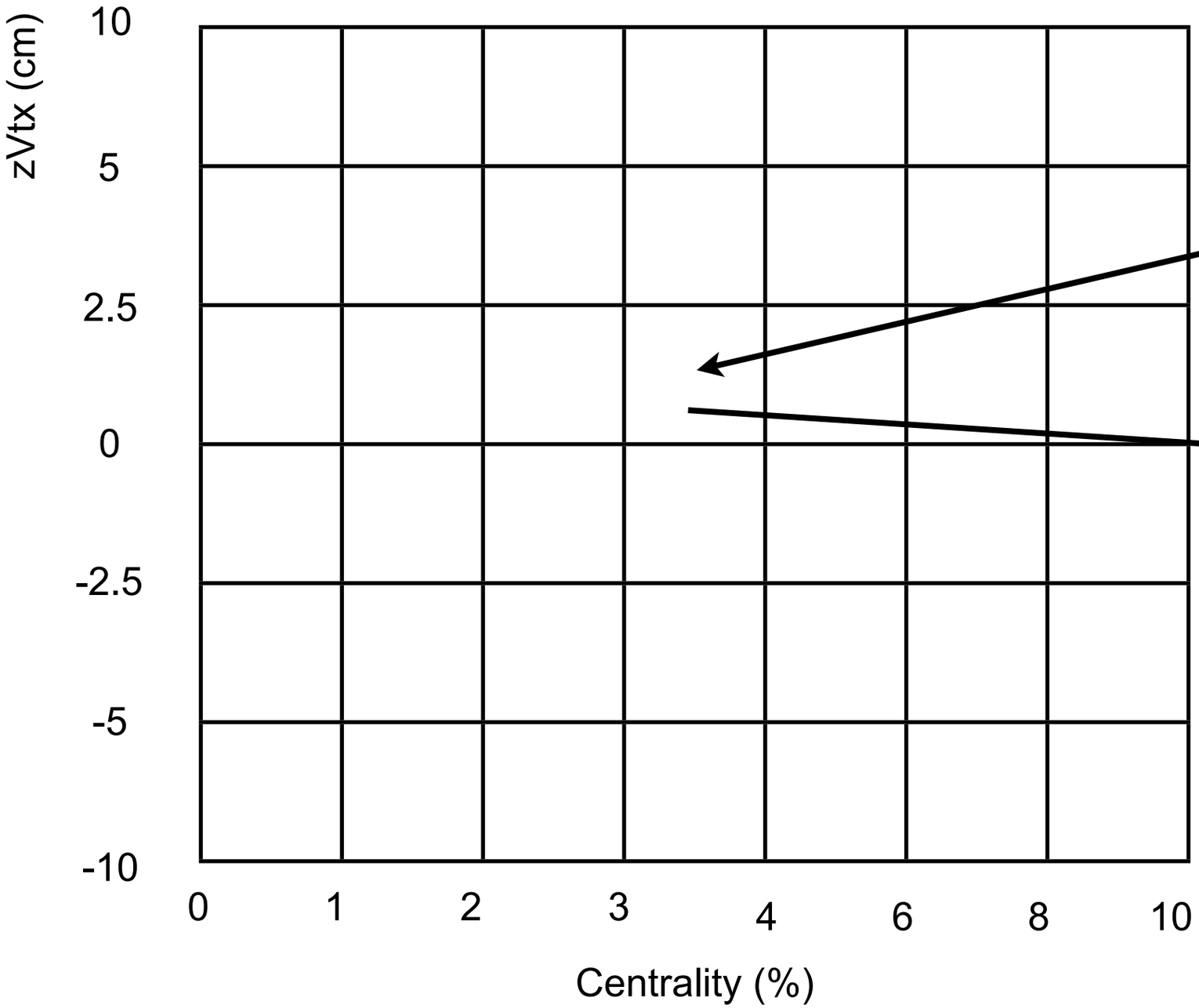
example (centrality 3.5%, zVtx 1.4 cm)

2) Get the pool to from the manager

AliEventPoolManager::GetEventPool(MultipOrCent, zvertex);

The event pool

The AliEventPoolManager: definition of the pools



1) Get info from the Event
centrality
zVtx

example (centrality 3.5%, zVtx 1.4 cm)

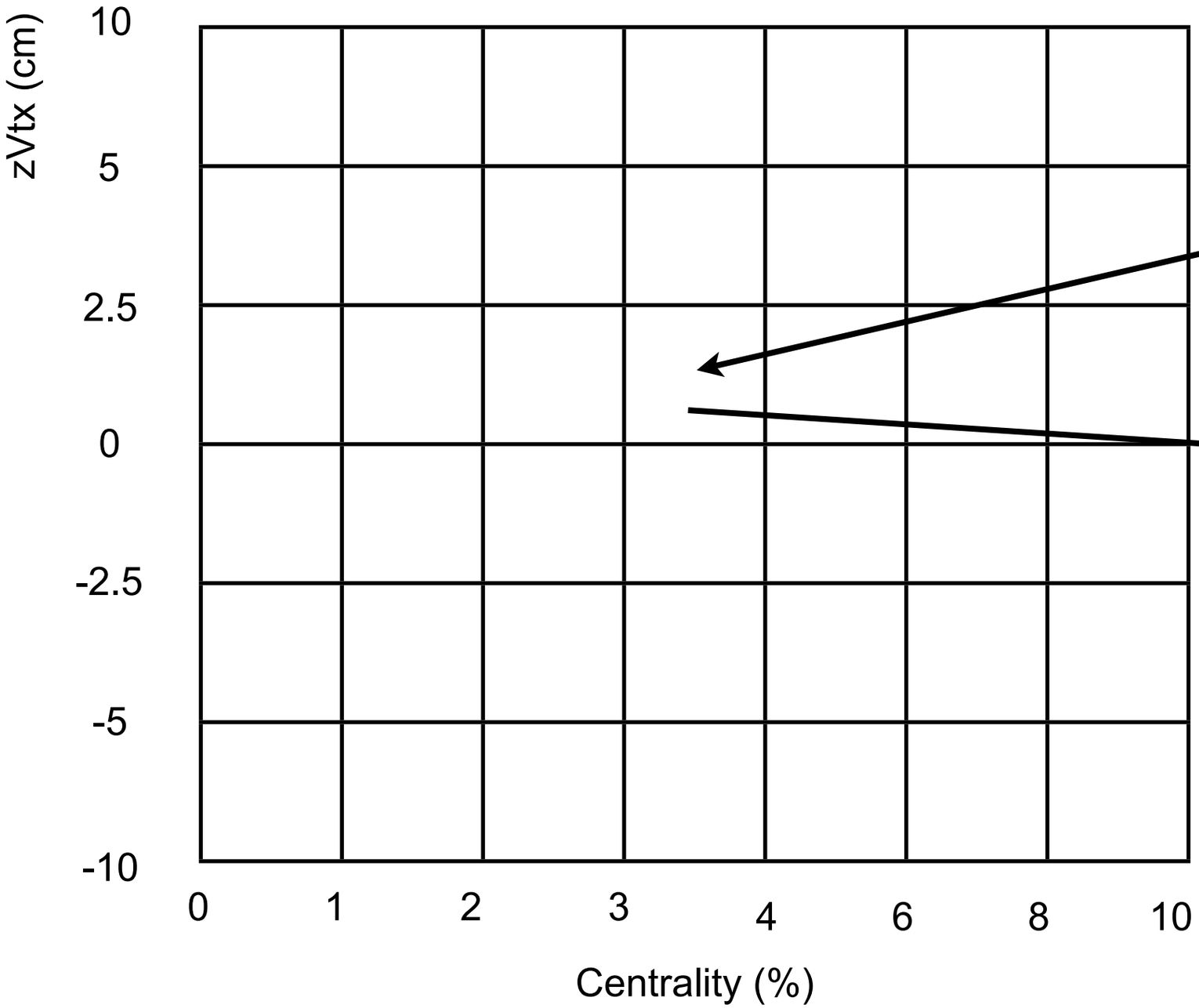
2) Get the pool to from the manager

3) Extract the tracks/events in the pool

AliEventPoolManager::GetEventPool(MultipOrCent, zvertex);

The event pool

The AliEventPoolManager: definition of the pools



1) Get info from the Event
centrality
zVtx

example (centrality 3.5%, zVtx 1.4 cm)

2) Get the pool to from the manager

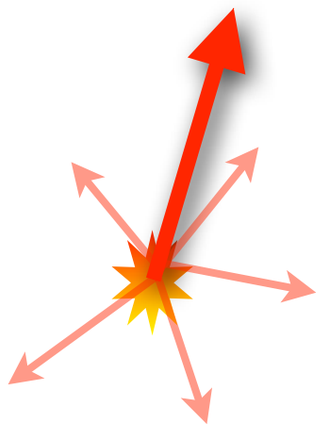
3) Extract the tracks/events in the pool

4) Run the mixing analysis from event N and the events stored in the pool

AliEventPoolManager::GetEventPool(MultipOrCent, zvertex);

Event mixing in your task

Event N



Event mixing in your task

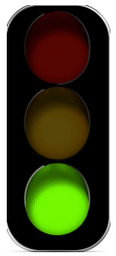
Event N - select the “trigger” particle

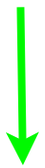
Is the event pool ready in the centrality/zVtx bin?

Event mixing in your task

Event N - select the “trigger” particle

Is the event pool ready in the centrality/zVtx bin?



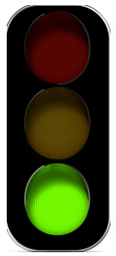


Yes

Event mixing in your task

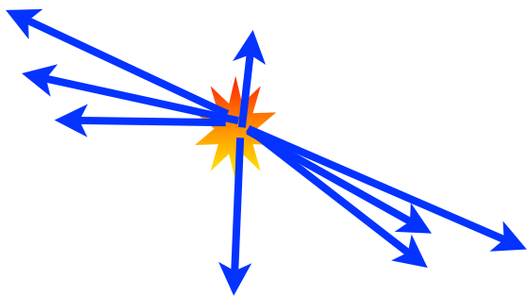
Event N - select the “trigger” particle

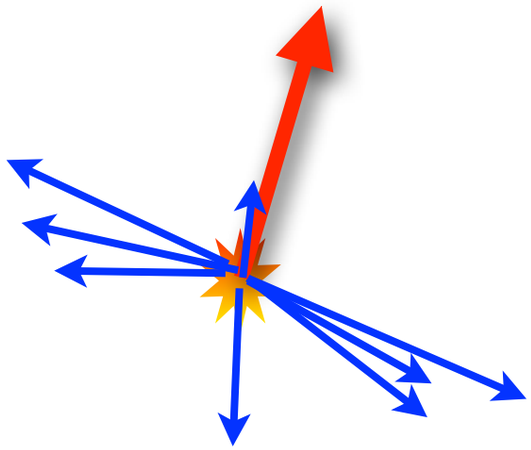
Is the event pool ready in the centrality/zVtx bin?



Yes

Get event N-1 from the pool

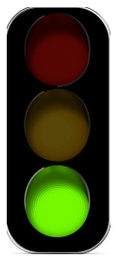




Event mixing in your task

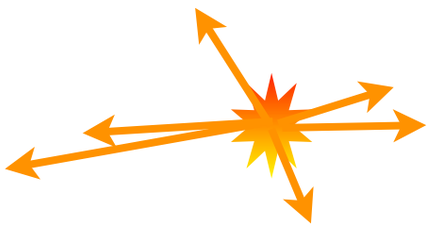
Event N - select the “trigger” particle

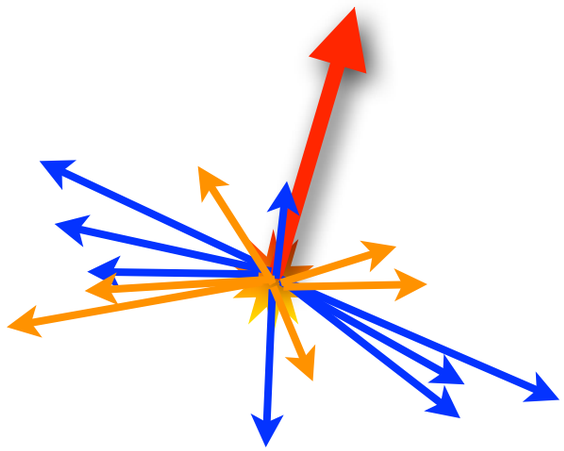
Is the event pool ready in the centrality/zVtx bin?



Yes

Get event N-2 from the pool

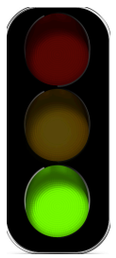




Event mixing in your task

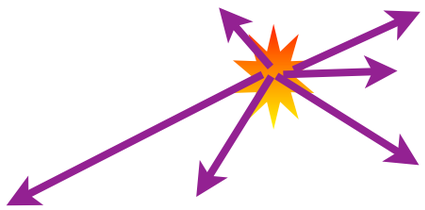
Event N - select the “trigger” particle

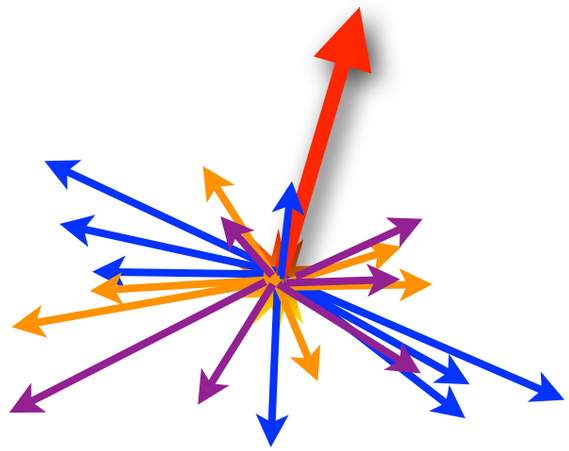
Is the event pool ready in the centrality/zVtx bin?



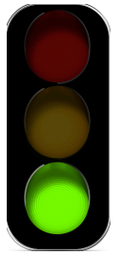
Yes

Get event N-3 from the pool

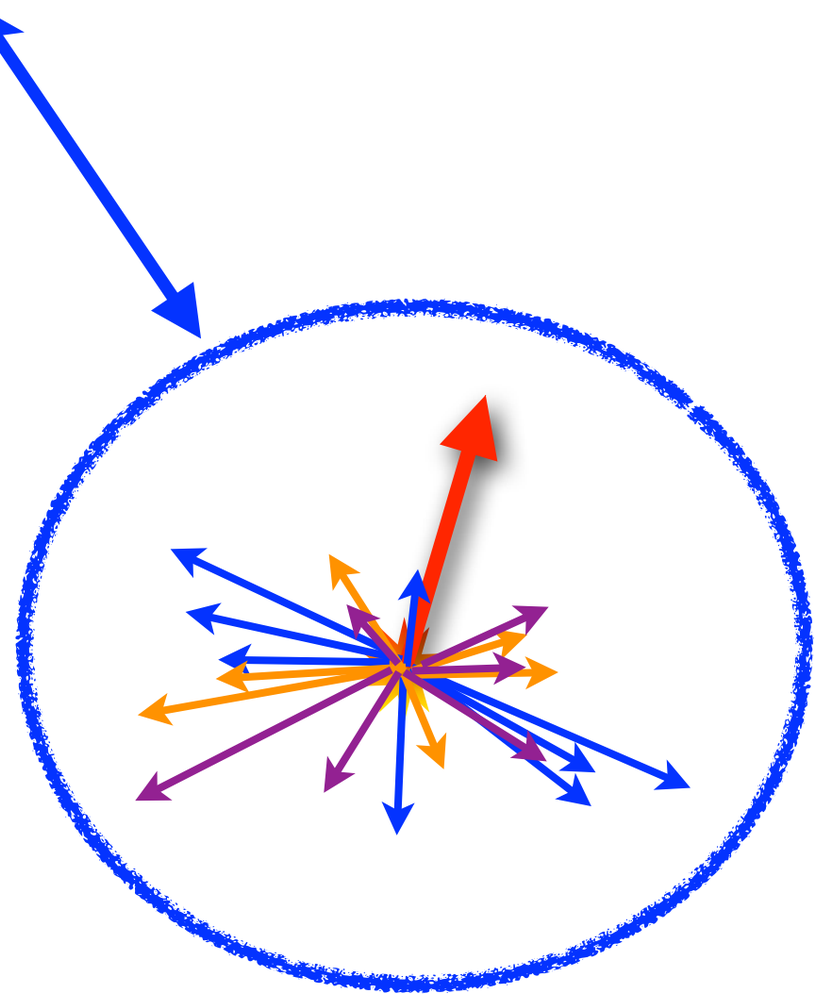




Event mixing in your task

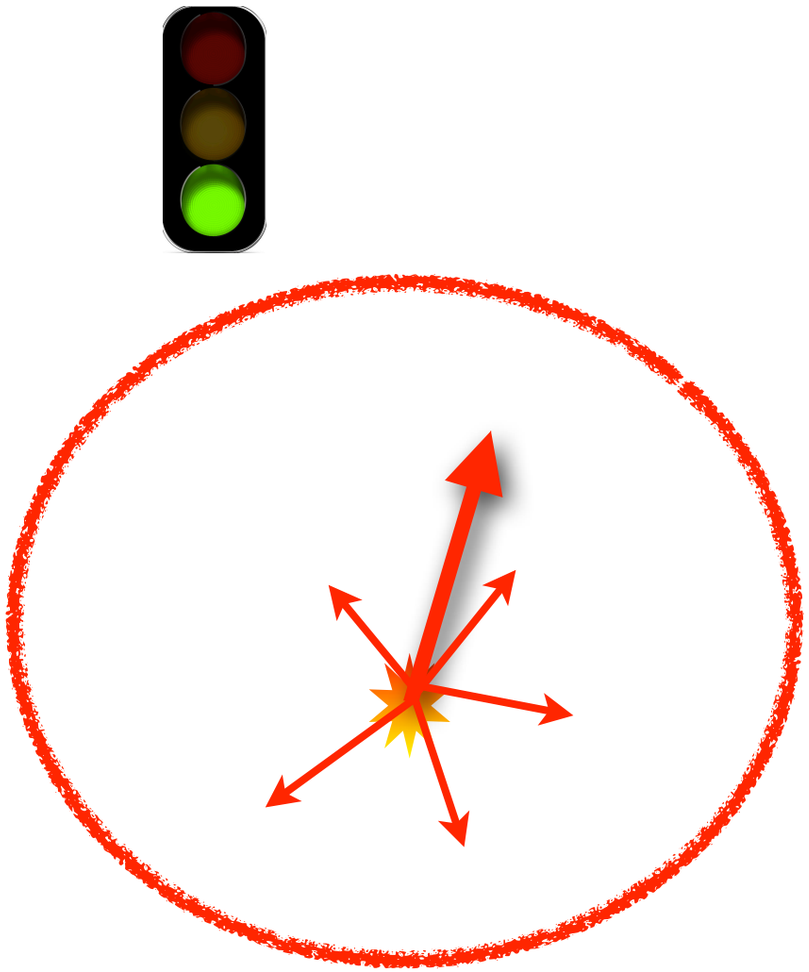


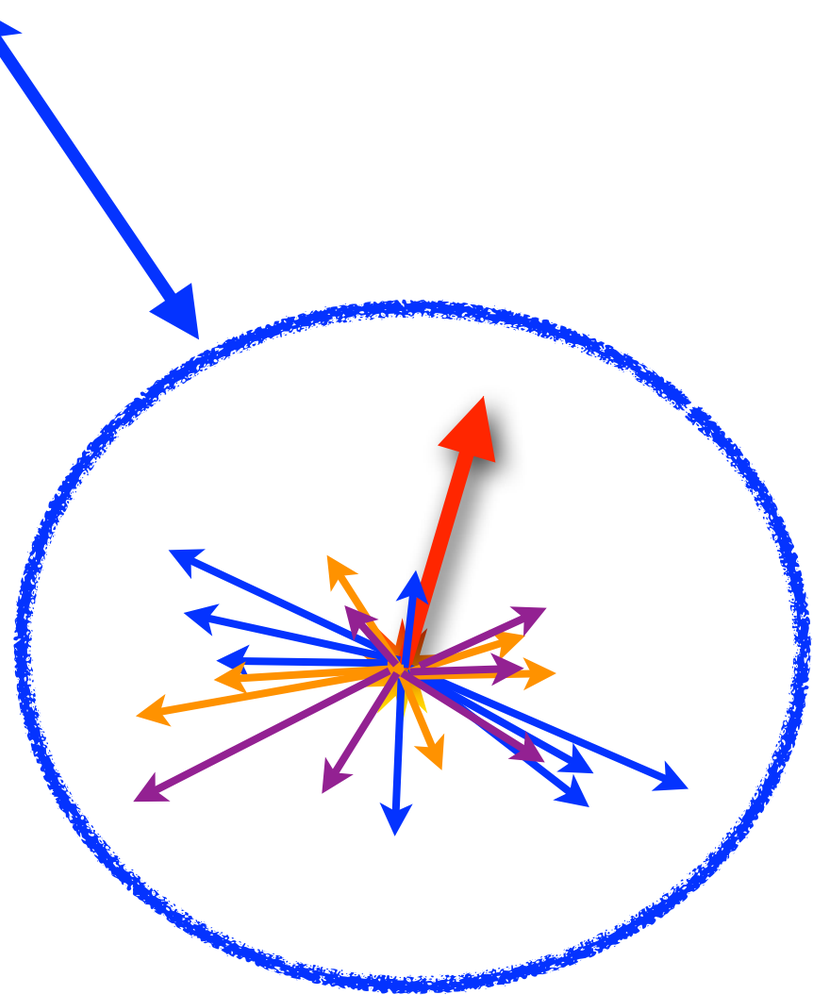
1) Study the correlation distributions



Event mixing in your task

1) Study the correlation distributions





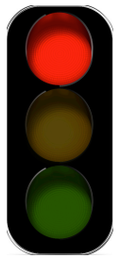
2)Store the event N in the pool

Andrea Rossi,Sandro Bjelogrlić

Event mixing in your task

Event N - select the “trigger” particle

Is the event pool ready in the centrality/zVtx bin?

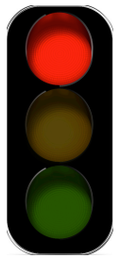


No

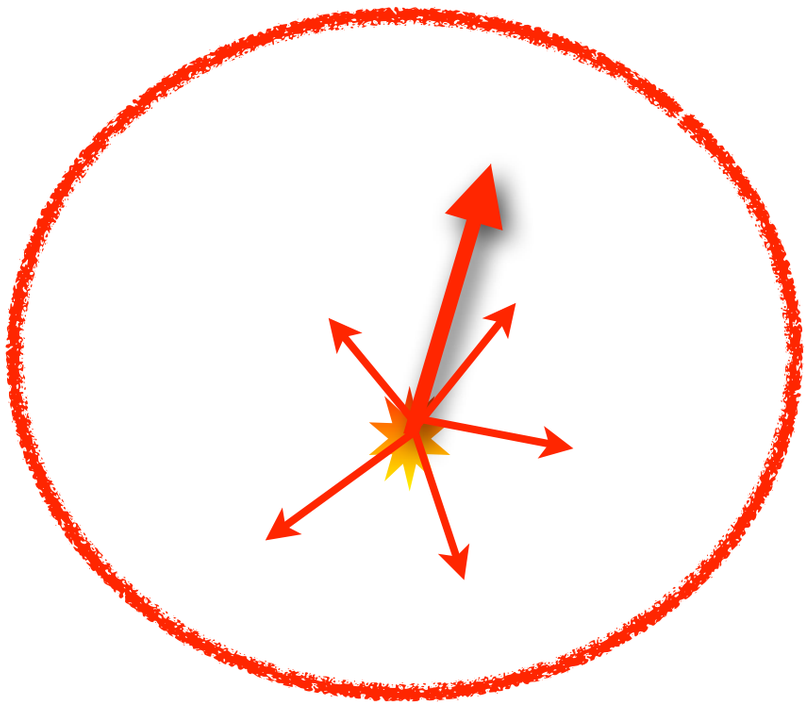
Event mixing in your task

Event N - select the “trigger” particle

Is the event pool ready in the centrality/zVtx bin?



No

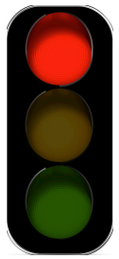


Store the event N in the pool

Event mixing in your task

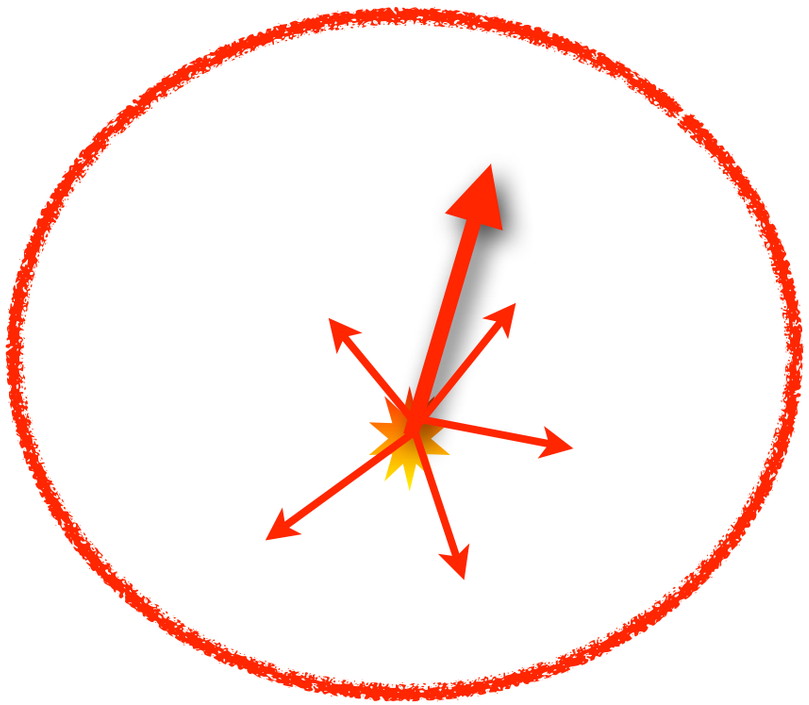
Event N - select the “trigger” particle

Is the event pool ready in the centrality/zVtx bin?



No

Go to event N+1



2)Store the event N in the pool

Event mixing: how to

The AliEventPoolManager: Get the pool for your event

Extract the event pool from the manager using the functions

```
// First uses bin indices, second uses the variables themselves.
(AliEventPool *) AliEventPoolManager:: GetEventPool(Int_t iMult, Int_t iZvtx, Int_t iPsi=0);
(AliEventPool *) AliEventPoolManager:: GetEventPool(Int_t centVal, Double_t zvtxVal, Double_t psiVal=0.);
(AliEventPool *) AliEventPoolManager:: GetEventPool(Double_t centVal, Double_t zvtxVal, Double_t psiVal=0.);
```

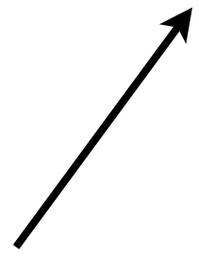
```
Double_t centralityObj = AODEvent->GetHeader()->GetCentralityP();
MultipOrCent = centralityObj->GetCentralityPercentileUnchecked("V0M");
```

```
AliAODVertex *vtx = AODEvent->GetPrimaryVertex();
Double_t zvertex = vtx->GetZ(); // zvertex
```

```
AliEventPool * Pool = PoolMgr->GetEventPool(MultipOrCent, zvertex);

Pool->SetTargetEvents(10); // set the minimum number of events to mix
```

Event mixing: how to

Checks that the pool for the selected event is ready (has a minimum number of tracks and events set previously)



```
Bool_t AliEventPool::IsReady();

if(!Pool->IsReady()) return;
```

Event mixing: how to

Define a TObjArray

```
TObjArray *AssociatedTracks = new TObjArray();
```

Loop on the events present in the event pool

```
For(Int_t k=0; k<Pool->GetCurrentNEvents(); k++){
AssociatedTracks = fPool->GetEvent(k);
 for loop over the tracks stored in the event k (explained in the next slide)

}
```

Event mixing: how to

```
For(Int_t k=0; k<Pool->GetCurrentNEvents(); k++){

AssociatedTracks = fPool->GetEvent(k);

  for loop over the tracks stored in the event k

}
```

```
Int_t NofTracks = AssociatedTracks->GetEntriesFast();

for(Int_t iTrack = 0; iTrack<NofTracks; iTrack++){

MyTrack * MixedTrack = (MyTrack*) AssociatedTracks->At(iTrack);

Double_t phimixed = MixedTrack->Phi();
...
// calculate correlations
}
```

Event mixing: update the pool

!! At the end of your analysis !!

Update the event pool → store the event N that will be used at the next step for mixing

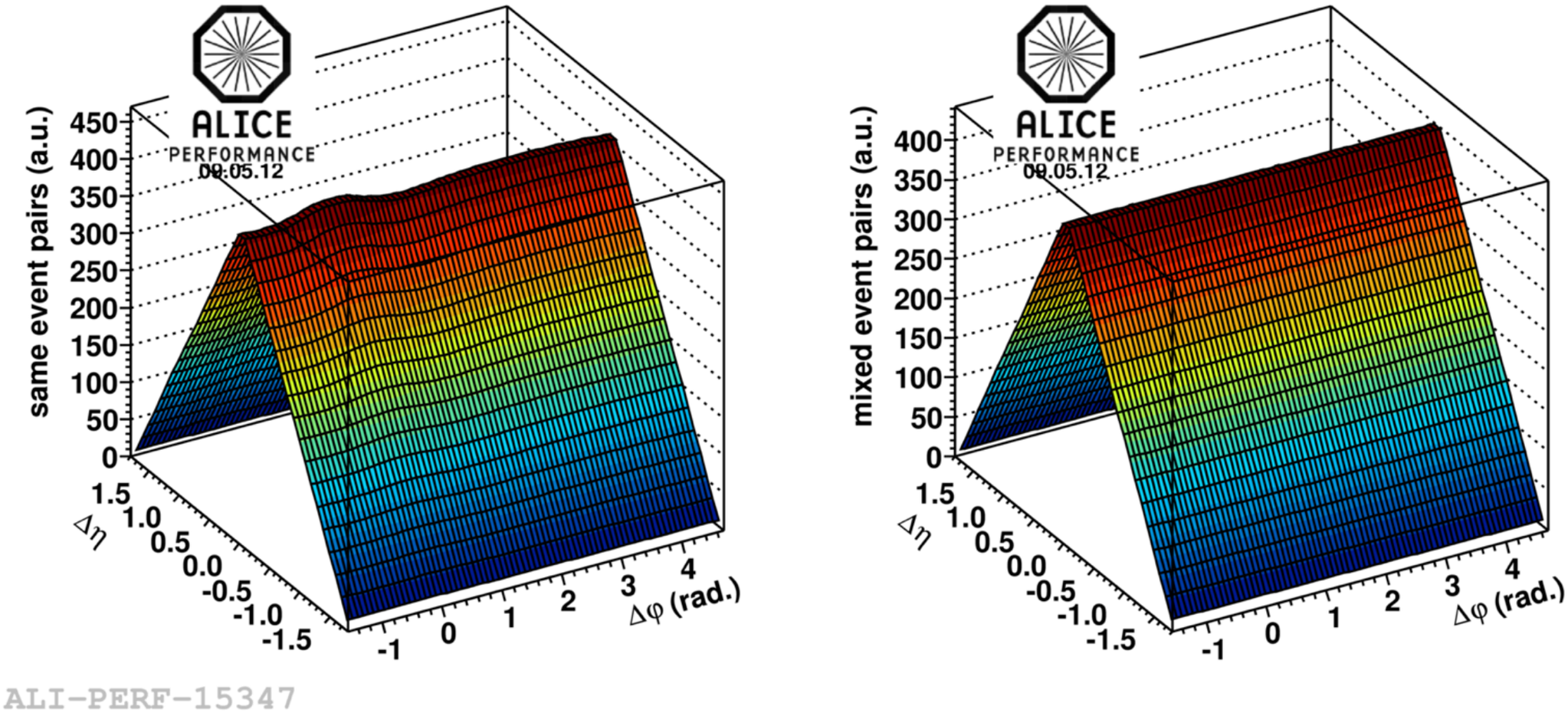
```
TObjArray *TrackArray = new TObjArray();
TrackArray->SetOwner(kTRUE);

// loop on all the tracks
for(Int_t iTrack = 0; iTrack<AOD->GetNTracks(); iTrack++){
 AliAODTrack* track = AOD->GetTrack(iTrack);
```

apply your selections to the track

```
TrackArray->Add(new MyTrack(track->Eta(), track->Phi(), track ->pT));
}
```

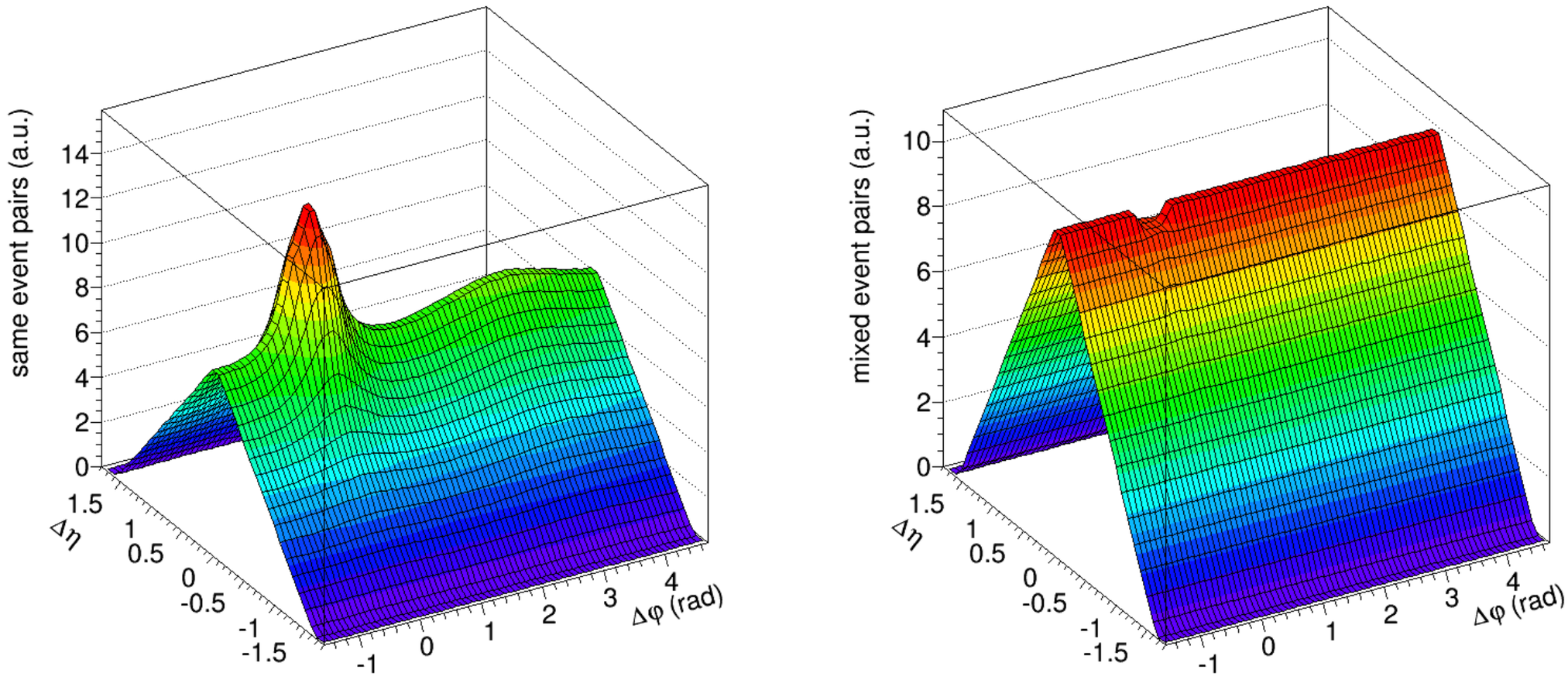
```
Pool->UpdatePool(TrackArray);
```

The offline correction



PbPb

The offline correction



pPb

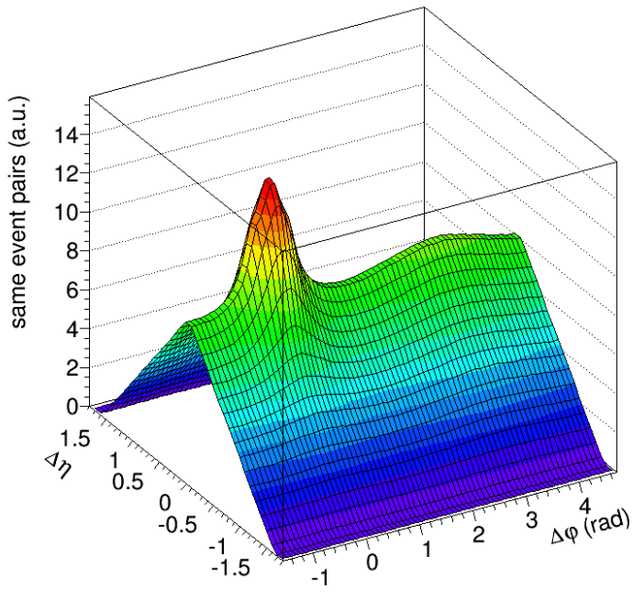
The offline correction

$$\frac{d^2N^{Corrected}(\Delta\phi,\Delta\eta)}{d\Delta\phi d\Delta\eta} = \frac{\frac{d^2N^{SE}(\Delta\phi,\Delta\eta)}{d\Delta\phi d\Delta\eta}}{\frac{d^2N^{ME}(\Delta\phi,\Delta\eta)}{d\Delta\phi d\Delta\eta}} \frac{d^2N^{ME}(0,0)}{d\Delta\phi d\Delta\eta}$$

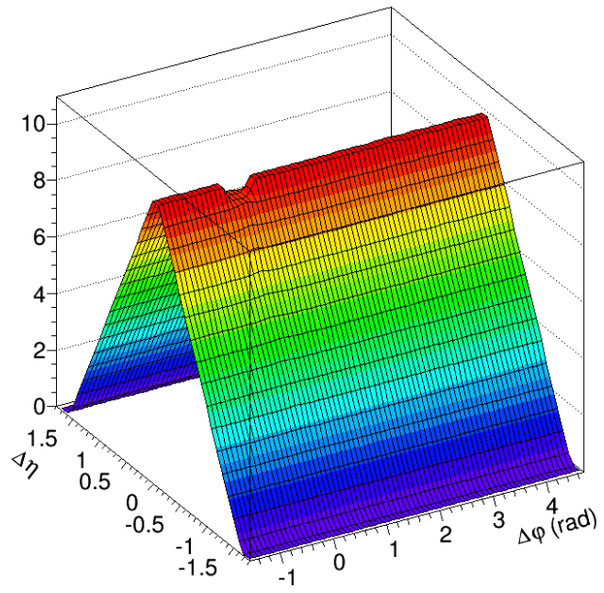
(Done for each pool bin)

- Normalize the mixed distribution to the value at (0,0)→ a track that goes in the same direction as the trigger one “feels” the same condition
- Use the mixed event as a weight: divide the single event distribution with the mixed one
- Do it for each pool bin
- Merge your outputs
- Have fun with the rest of your analysis

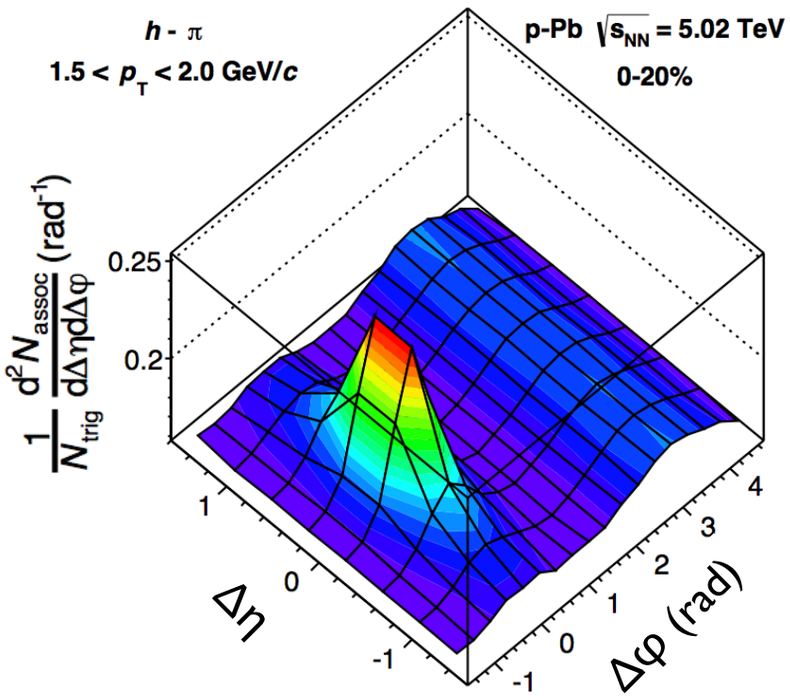
The offline correction



/



=

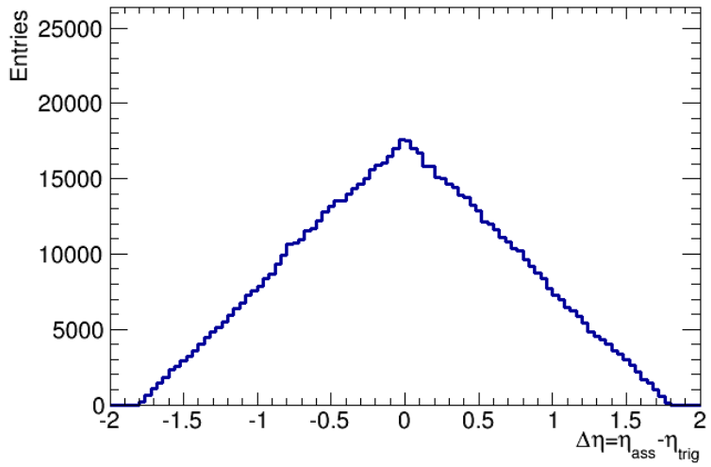


After correction physical correlation remains

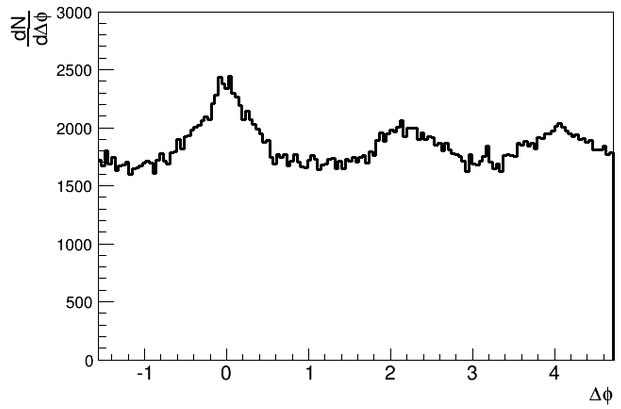
pPb

Summary

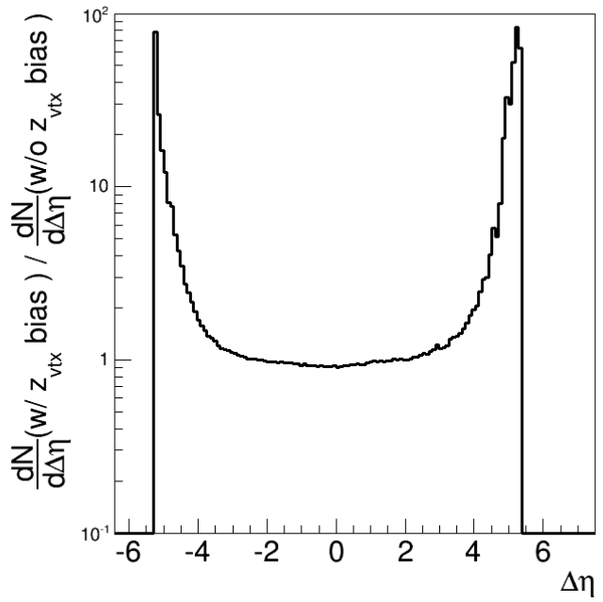
Correction for the pseudorapidity triangular shape



Correction for the azimuthal inhomogenuity



Careful with the vertex position



Summary: my task

MyTask::UserCreateOutputObject()

```
Int_t MaxNofEvents   = 1000;
Int_t trackDepth = 2000;
Int_t nCentralityBins  = 15;
Double_t centralityBins[] = { 0, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.1 };
Int_t nZvtxBins  = 7;
Double_t vertexBins[] = { -7, -5, -3, -1, 1, 3, 5, 7 };
AliEventPoolManager *manager = new AliEventPoolManager(MaxNofEvents,trackDepth,nCentralityBins,(Double_t*)centralityBins,nZvtxBins,(Double_t*) vertexBins);
```

MyTask::UserExec()

```
Double_t centralityObj = AODEvent->GetHeader()->GetCentralityP();
MultipOrCent = centralityObj->GetCentralityPercentileUnchecked("V0M");
AliAODVertex *vtx = AODEvent->GetPrimaryVertex();
Double_t zvertex = vtx->GetZ(); // zvertex
AliEventPool * Pool = manager->GetEventPool(MultipOrCent, zvertex);
if(!Pool->IsReady()) return;
TObjArray *AssociatedTracks = new TObjArray();
for (Int_t k=0; k<Pool->GetCurrentNEvents(); k++){
AssociatedTracks = fPool->GetEvent(k);
        Int_t NofTracks = AssociatedTracks->GetEntriesFast();
        for(Int_t iTrack = 0; iTrack<NofTracks; iTrack++){
                MyTrack * MixedTrack = (MyTrack*) AssociatedTracks->At(iTrack);
                Double_t phimixed = MixedTrack->Phi();
        ...
        // calculate correlations
        }
}
TObjArray *TrackArray = new TObjArray();
TrackArray->SetOwner(kTRUE);
// loop on all the tracks
for(Int_t iTrack = 0; iTrack<AOD->GetNTracks(); iTrack++){
     AliAODTrack* track = AOD->GetTrack(iTrack);
    apply your selections to the track
TrackArray->Add(new MyTrack(track->Eta(), track->Phi(), track ->pT));
}
Pool->UpdatePool(TrackArray);
```

BACKUP

EM analysis

- Developed through two classes
  - Analysis Task
  - AliHFCorrelator
- Uses the framework developed by PWGCF
- Inputs for the event mixing is passed through the associated tracks cut object

```
HFCorrelationCuts->SetMaxNEventsInPool(200);
HFCorrelationCuts->SetMinNTracksInPool(1000);
HFCorrelationCuts->SetMinEventsToMix(8);

HFCorrelationCuts->SetNofPoolBins(5,4);

Double_t MBins[]={0,20,40,60,500};
Double_t * MultiplicityBins = MBins;

Double_t ZBins[]={-10,-5,-2.5,2.5,5,10};
Double_t *ZVrtxBins = ZBins;

HFCorrelationCuts->SetPoolBins(ZVrtxBins,MultiplicityBins);
```

Task::UserCreateOutputObjects

☑ Definition of the correlator

```
fCorrelator = new AliHFCorrelator("Correlator",fCuts2,fSystem);
```

AliHFAssociatedTrackCuts contains info on the mixing

Bool to switch in between pp, AA (multiplicity/centrality)

```
fCorrelator->SetEventMixing(fmixing); //set kFALSE/kTRUE for mixing Off/On
Bool_t pooldef = fCorrelator->DefineEventPool();
```

```
//_____________________________________________________________________
Bool_t AliHFCorrelator::DefineEventPool(){
   // definition of the Pool Manager for Event Mixing
   Int_t MaxNofEvents = fhadcuts->GetMaxNEventsInPool();
   Int_t MinNofTracks = fhadcuts->GetMinNTracksInPool();
   Int_t NofCentBins = fhadcuts->GetNCentPoolBins();
   Double_t * CentBins = fhadcuts->GetCentPoolBins();
   Int_t NofZVrtxBins = fhadcuts->GetNZvtxPoolBins();
   Double_t *ZVrtxBins = fhadcuts->GetZvtxPoolBins();
   fPoolMgr = new AliEventPoolManager(MaxNofEvents, MinNofTracks, NofCentBins, CentBins, NofZVrtxBins,
ZVrtxBins);
   if(!fPoolMgr) return kFALSE;
   return kTRUE;
}
```

Roadmap of the task

```
//_____________________________________
Task::UserExec(){
   Initalization of the event i
   Physics selection
   Reading the branch of the AOD containing filtered D candidates
   for(on D candidates in the AOD: Int_t j){

      topological selection on D meson
      Invariant mass check (is in peak or sideband)

      if(!sideband && !peak) continue

         for(on the events in the pool if analysis on event mixing){
            loop on event (i-1), (i-2), (i-3) .....
            "events before the one where the D candidate is found"

            for(on associated tracks in the AOD/pool: Int_t k){

                  track quality selection (Done in the AliHFCorrelator)
                  is Track a D daughter? (if yes) continue;
                  calculate Delta Phi, Delta eta in between D(j) and track(k)

                  Filling histos/ ThNsparses
            } // end loop on tracks

         } // end loop on events in pool

      } // end loop on D candidates

The event pool for event mixing is updated with the tracks from this AOD event (i^th)
Makes sure we are actually mixing
} // end loop on events
```

Where we are in the code

```
//_____________________________________________
Task::UserExec(){

   Initalization of the event i
   Physics selection
   Reading the branch of the AOD containing filtered D candidates

    for(on D candidates in the AOD: Int_t j){

       topological selection on D meson
       Invariant mass check (is in peak or sideband)

       if(!sideband && !peak) continue

          for(on the events in the pool if analysis on event mixing){
             loop on event (i-1), (i-2), (i-3) .....
             "events before the one where the D candidate is found"

            for(on associated tracks in the AOD/pool: Int_t k){

                    track quality selection (Done in the AliHFCorrelator)
                    is Track a D daughter? (if yes) continue;
                    calculate Delta Phi, Delta eta in between D(j) and track(k)

                    Filling histos/ ThNsparses
            } // end loop on tracks

         } // end loop on events in pool

     } // end loop on D candidates

The event pool for event mixing is updated with the tracks from this AOD event (i^th)
Makes sure we are actually mixing
} // end loop on events
```

Task::UserExec (*before* loop on D)

☑ Initialization of the pool

```
Bool_t correlatorON = fCorrelator->Initialize(); //define the pool for mixing

if(!correlatorON) {
      AliInfo("AliHFCorrelator didn't initialize the pool correctly or processed a bad
event");
      return;
   }
```

AliHFCorrelator::Initialize()

```
//_____________________________________________________
Bool_t AliHFCorrelator::Initialize(){

    //  std::cout << "AliHFCorrelator::Initialize"<< std::endl;
  AliInfo("AliHFCorrelator::Initialize") ;
  if(!fAODEvent){
    AliInfo("No AOD event") ;
    return kFALSE;
  }
    //std::cout << "No AOD event" << std::endl;

    AliCentrality *centralityObj = 0;
    Int_t multiplicity = -1;
    Double_t MultipOrCent = -1;

    // initialize the pool for event mixing
    if(!fsystem){ // pp
    multiplicity = fAODEvent->GetNTracks();
           MultipOrCent = multiplicity; // convert from Int_t to Double_t
    }
    if(fsystem){ // PbPb

           centralityObj = fAODEvent->GetHeader()->GetCentralityP();
           MultipOrCent = centralityObj->GetCentralityPercentileUnchecked("V0M");
           AliInfo(Form("Centrality is %f", MultipOrCent));
    }

    AliAODVertex *vtx = fAODEvent->GetPrimaryVertex();
    Double_t zvertex = vtx->GetZ(); // zvertex
    Double_t * CentBins = fhadcuts->GetCentPoolBins();
    Double_t poolmin=CentBins[0];
    Double_t poolmax=CentBins[fhadcuts->GetNCentPoolBins()];


           if(TMath::Abs(zvertex)>=10 || MultipOrCent>poolmax || MultipOrCent < poolmin) {
           if(!fsystem)AliInfo(Form("pp Event with Zvertex = %.2f cm and multiplicity = %.0f out of pool bounds, SKIPPING",zvertex,MultipOrCent));
           if(fsystem) AliInfo(Form("PbPb Event with Zvertex = %.2f cm and centrality = %.1f  out of pool bounds, SKIPPING",zvertex,MultipOrCent));

                 return kFALSE;
           }

    fPool = fPoolMgr->GetEventPool(MultipOrCent, zvertex);

    if (!fPool){
           AliInfo(Form("No pool found for multiplicity = %f, zVtx = %f cm", MultipOrCent, zvertex));
        return kFALSE;
    }
    //fPool->PrintInfo();
    return kTRUE;
}
```

Switch pp/AA
Multiplicity or Centrality for the pools

Not the actual multiplicity, but the N of tracks in the AOD

Get the Event pool relative to the multiplicity/zvtx bin

General structure of the task

```
//_____________________________________________
Task::UserExec(){

   Initalization of the event i
   Physics selection
   Reading the branch of the AOD containing filtered D candidates

   for(on D candidates in the AOD: Int_t j){

      topological selection on D meson
      Invariant mass check (is in peak or sideband)

      if(!sideband && !peak) continue

         for(on the events in the pool if analysis on event mixing){
            loop on event (i-1), (i-2), (i-3) .....
            "events before the one where the D candidate is found"

            for(on associated tracks in the AOD/pool: Int_t k){

               track quality selection (Done in the AliHFCorrelator)
               is Track a D daughter? (if yes) continue;
               calculate Delta Phi, Delta eta in between D(j) and track(k)

               Filling histos/ ThNsparses
            } // end loop on tracks

         } // end loop on events in pool

      } // end loop on D candidates

The event pool for event mixing is updated with the tracks from this AOD event (i^th)
Makes sure we are actually mixing
} // end loop on events
```

Task::UserExec (*inside* loop on D)

If a D candidate is found - after topological and invariant mass selection - peak or sideband

☑ Processing of the pool

```
Bool_t execPool = fCorrelator->ProcessEventPool();
     if(fmixing && !execPool) {
         AliInfo("Mixed event analysis: pool is not ready");
         continue;
     }
```

```
//_____________________________________________________
Bool_t AliHFCorrelator::ProcessEventPool(){
    // analysis on Mixed Events
   //cout << "AliHFCorrelator::ProcessEventPool"<< endl;
        if(!fmixing) return kFALSE;
        if(!fPool->IsReady()) return kFALSE;
        if(fPool->GetCurrentNEvents()<fhadcuts->GetMinEventsToMix()) return kFALSE;
   //   fPool->PrintInfo();
        fPoolContent = fPool->GetCurrentNEvents();

        return kTRUE;

}
```

Checks if the event pool is ready for the mixing analysis! - satisfies the conditions we set at the beginning

Where we are in the code

```
//_____________________________________________
Task::UserExec(){

   Initalization of the event i
   Physics selection
   Reading the branch of the AOD containing filtered D candidates

    for(on D candidates in the AOD: Int_t j){

       topological selection on D meson
       Invariant mass check (is in peak or sideband)

       if(!sideband && !peak) continue

          for(on the events in the pool if analysis on event mixing){
             loop on event (i-1), (i-2), (i-3) .....
             "events before the one where the D candidate is found"

             for(on associated tracks in the AOD/pool: Int_t k){

                   track quality selection (Done in the AliHFCorrelator)
                   is Track a D daughter? (if yes) continue;
                   calculate Delta Phi, Delta eta in between D(j) and track(k)

                   Filling histos/ ThNsparses
             } // end loop on tracks

          } // end loop on events in pool

    } // end loop on D candidates

The event pool for event mixing is updated with the tracks from this AOD event (i^th)
Makes sure we are actually mixing
} // end loop on events
```

Task::UserExec (*inside* loop on D)

If the event pool is ready...

☑ Get the number of events in the pool of the and loop on them

```
        Int_t NofEventsinPool = 1; // SE analysis
    if(fmixing) NofEventsinPool = fCorrelator->GetNofEventsInPool(); // ME analysis
```

Loop on the events in the pool (1 if SE)

```
for (Int_t jMix =0; jMix < NofEventsinPool; jMix++){// loop on events in the pool;
if it is SE analysis, stops at one

            Bool_t analyzetracks = fCorrelator->ProcessAssociatedTracks(jMix);
```

```
//_____________________________________________________
Bool_t AliHFCorrelator::ProcessAssociatedTracks(Int_t EventLoopIndex, const TObjArray* associatedTracks){

    fAssociatedTracks = new TObjArray();

    if(!fmixing){ // analysis on Single Event

          if(fselect==kHadron || fselect ==kKaon) fAssociatedTracks = AcceptAndReduceTracks(fAODEvent);
          if(fselect==kKZero) {fAssociatedTracks = AcceptAndReduceKZero(fAODEvent);}
          if(fselect==kElectron && associatedTracks) fAssociatedTracks=new TObjArray(*associatedTracks);

    }
    if(fmixing) { // analysis on Mixed Events
          fAssociatedTracks = fPool->GetEvent(EventLoopIndex);
    } // end if mixing

    if(!fAssociatedTracks) return kFALSE;

    fNofTracks = fAssociatedTracks->GetEntriesFast();

    return kTRUE;

}
```

Function explained at the end

TObjArray that contains the tracks from one of the previous events
(i-1) or (i-2) or (i-3),....

Where we are in the code

```
//_____________________________________________
Task::UserExec(){

   Initalization of the event i
   Physics selection
   Reading the branch of the AOD containing filtered D candidates

   for(on D candidates in the AOD: Int_t j){

      topological selection on D meson
      Invariant mass check (is in peak or sideband)

      if(!sideband && !peak) continue

         for(on the events in the pool if analysis on event mixing){
            loop on event (i-1), (i-2), (i-3) .....
            "events before the one where the D candidate is found"

            for(on associated tracks in the AOD/pool: Int_t k){

                  track quality selection (Done in the AliHFCorrelator)
                  is Track a D daughter? (if yes) continue;
                  calculate Delta Phi, Delta eta in between D(j) and track(k)

                  Filling histos/ ThNsparses
            } // end loop on tracks

         } // end loop on events in pool

   } // end loop on D candidates

The event pool for event mixing is updated with the tracks from this AOD event (i^th)
Makes sure we are actually mixing
} // end loop on events
```

Task::UserExec (*inside* loop on events in pool)

```
Int_t NofTracks = fCorrelator->GetNofTracks();
```

Get the number of tracks that are stored in the TObjArray for the given event

```
for(Int_t iTrack = 0; iTrack<NofTracks; iTrack++){ // looping on track candidates

    Bool_t runcorrelation = fCorrelator->Correlate(iTrack);
    if(!runcorrelation) continue;

    Double_t DeltaPhi = fCorrelator->GetDeltaPhi();
    Double_t DeltaEta = fCorrelator->GetDeltaEta();

    AliReducedParticle * hadron = fCorrelator->GetAssociatedParticle();

    Double_t ptHad = hadron->Pt();
    Double_t phiHad = hadron->Phi();
    Double_t etaHad = hadron->Eta();
    Int_t label = hadron->GetLabel();
    Int_t trackid = hadron->GetID();
```

```
//_____________________________________________________
Bool_t AliHFCorrelator::Correlate(Int_t loopindex){

    if(loopindex >= fNofTracks) return kFALSE;
    if(!fAssociatedTracks) return kFALSE;

    fReducedPart = (AliReducedParticle*)fAssociatedTracks->At(loopindex);


    fDeltaPhi = SetCorrectPhiRange(fPhiTrigger - fReducedPart->Phi());

    fDeltaEta = fEtaTrigger - fReducedPart->Eta();

    return kTRUE;

}
```

Gets the loopindexth track in the TObjArray

Calculation of delta phi and delta eta

Task::UserExec (*inside* loop on events on the tracks)

Filling the histos/ThNSparses

```
    arraytofill[0] = DeltaPhi;
    arraytofill[1] = DeltaEta;
    arraytofill[2] = ptDStar;
    arraytofill[3] = etaDStar;
    arraytofill[4] = ptHad;
   // arraytofill[5] = zVtxPosition;
    arraytofill[5] = MultipOrCent;


if(isInPeak)  {

            if(!fReco && TMath::Abs(etaHad)>0.9) continue;
            if(fselect==1)  ((THnSparseF*)fOutput->FindObject("CorrelationsDStarHadron"))->Fill(arraytofill);


            }
```

```
if(isInDZeroSideBand) {

            if(!fReco && TMath::Abs(etaHad)>0.9) continue;

              if(fselect==1)  ((THnSparseF*)fOutput->FindObject("DZeroBkgCorrelationsDStarHadron"))->Fill(arraytofill);

              }

 if(isInDStarSideBand) {

            if(!fReco && TMath::Abs(etaHad)>0.9) continue;
               if(fselect==1)  ((THnSparseF*)fOutput->FindObject("DStarBkgCorrelationsDStarHadron"))->Fill(arraytofill);


            }
```

Where we are in the code

```
//_____________________________________________
Task::UserExec(){

   Initalization of the event i
   Physics selection
   Reading the branch of the AOD containing filtered D candidates

   for(on D candidates in the AOD: Int_t j){

      topological selection on D meson
      Invariant mass check (is in peak or sideband)

      if(!sideband && !peak) continue

         for(on the events in the pool if analysis on event mixing){
            loop on event (i-1), (i-2), (i-3) .....
            "events before the one where the D candidate is found"

            for(on associated tracks in the AOD/pool: Int_t k){

                  track quality selection (Done in the AliHFCorrelator)
                  is Track a D daughter? (if yes) continue;
                  calculate Delta Phi, Delta eta in between D(j) and track(k)

                  Filling histos/ ThNsparses
            } // end loop on tracks

         } // end loop on events in pool

   } // end loop on D candidates
```

The event pool for event mixing is updated with the tracks from this AOD event (i^{th})
Makes sure we are actually mixing
} // end loop on events

Task::UserExec (*after* the loop on the D mesons)

```
Bool_t updated = fCorrelator->PoolUpdate();

    if(!updated) AliInfo("Pool was not updated");
```

```
//________________________________________________________
Bool_t AliHFCorrelator::PoolUpdate(const TObjArray* associatedTracks){

    if(!fmixing) return kFALSE;
    if(!fPool) return kFALSE;
    if(fmixing) { // update the pool for Event Mixing
          TObjArray* objArr = NULL;
          if(fselect==kHadron || fselect==kKaon) objArr = (TObjArray*)AcceptAndReduceTracks(fAODEvent);
          else if(fselect==kKZero) objArr = (TObjArray*)AcceptAndReduceKZero(fAODEvent);
          else if(fselect==kElectron && associatedTracks){
            objArr = new TObjArray(*associatedTracks);
          }
          else return kFALSE;
          if(objArr->GetEntriesFast()>0) fPool->UpdatePool(objArr); // updating the pool only if there are entries in the array
    }

    return kTRUE;

}
```

Function that stores an array of tracks that pass the quality selections

AliHFCorrelator::AcceptAndReduce...(...) reco

```
//_____________________________________________
TObjArray*  AliHFCorrelator::AcceptAndReduceTracks(AliAODEvent* inputEvent){

  Double_t weight=1.;
  Int_t nTracks = inputEvent->GetNTracks();
  AliAODVertex * vtx = inputEvent->GetPrimaryVertex();
  Double_t pos[3],cov[6];
  vtx->GetXYZ(pos);
  vtx->GetCovarianceMatrix(cov);
  const AliESDVertex vESD(pos,cov,100.,100);

  Double_t Bz = inputEvent->GetMagneticField();
  TObjArray* tracksClone = new TObjArray;
  tracksClone->SetOwner(kTRUE);

  //*******************************************************
  // use reconstruction
  if(fUseReco){
    for (Int_t iTrack=0; iTrack<nTracks; ++iTrack) {
      AliAODTrack* track = inputEvent->GetTrack(iTrack);
      if (!track) continue;
      if(!fhadcuts->IsHadronSelected(track,&vESD,Bz)) continue; // apply ESD level selections
      if(!fhadcuts->Charge(fDCharge,track)) continue; // apply selection on charge, if required

      Double_t pT = track->Pt();

      //compute impact parameter
      Double_t d0z0[2],covd0z0[3];
      Double_t d0=-999999.;
      if(fUseImpactParameter) track->PropagateToDCA(vtx,Bz,100,d0z0,covd0z0);
      else d0z0[0] = 1. ; // random number - be careful with the cuts you applied

      if(fUseImpactParameter==1) d0 = TMath::Abs(d0z0[0]); // use impact parameter
      if(fUseImpactParameter==2) { // use impact parameter over resolution
    if(TMath::Abs(covd0z0[0])>0.00000001) d0 = TMath::Abs(d0z0[0])/TMath::Sqrt(covd0z0[0]);
    else d0 = -1.; // if the resoultion is Zero, rejects the track - to be on the safe side

      }

      f(!fhadcuts->CheckHadronKinematic(pT,d0)) continue; // apply kinematic cuts
      Bool_t rejectsoftpi = kTRUE;// TO BE CHECKED: DO WE WANT IT TO kTRUE AS A DEFAULT?
      if(fD0cand && !fmixing) rejectsoftpi = fhadcuts->InvMassDstarRejection(fD0cand,track,fhypD0); // TO BE CHECKED: WHY NOT FOR EM?


      if(fselect ==kKaon){
    if(!fhadcuts->CheckKaonCompatibility(track,fmontecarlo,fmcArray,fPIDmode)) continue; // check if it is a Kaon - data and MC
      }
      weight=fhadcuts->GetTrackWeight(pT,track->Eta(),pos[2]);
      tracksClone->Add(new AliReducedParticle(track->Eta(), track->Phi(), pT,track->GetLabel(),track->GetID(),d0,rejectsoftpi,track->Charge(),weight));
    } // end loop on tracks
  } // end if use reconstruction kTRUE
```

Loop on the tracks

Selection done for AliESDtrackCuts class (stored in the cut object)

fhadcuts - associated track cuts / AliHFAssociatedTrackCuts class

Impact parameter of the track - computed only if the flag is set to true - saves CPU time on grid

Selection for pt and imp.par

PID for Kaons

Storing in the TObjArray

AliHFCorrelator::AcceptAndReduce...(...) MC truth

```
//_____________________________________________________
TObjArray*  AliHFCorrelator::AcceptAndReduceTracks(AliAODEvent* inputEvent){


  //use MC truth
  if(fmontecarlo && !fUseReco){

    for (Int_t iPart=0; iPart<fmcArray->GetEntriesFast(); iPart++) {
      AliAODMCParticle* mcPart = dynamic_cast<AliAODMCParticle*>(fmcArray->At(iPart));
      if (!mcPart) {
    AliWarning("MC Particle not found in tree, skipping");
    continue;
      }
      if(!mcPart->Charge()) continue; // consider only charged tracks

      Int_t PDG =TMath::Abs(mcPart->PdgCode());
if(fselect ==kHadron) {if(!((PDG==321)||(PDG==211)||(PDG==2212)||(PDG==13)||(PDG==11))) continue;} // select only if kaon, pion, proton, muon or electron
      else if(fselect ==kKaon) {if(!(PDG==321)) continue;} // select only if kaon
      else if(fselect ==kElectron) {if(!(PDG==11)) continue;} // select only if electron

      Double_t pT = mcPart->Pt();
      Double_t d0 =1; // set 1 fot the moment - no displacement calculation implemented yet
      if(!fhadcuts->CheckHadronKinematic(pT,d0)) continue; // apply kinematic cuts

      tracksClone->Add(new AliReducedParticle(mcPart->Eta(), mcPart->Phi(), pT,iPart,-1,d0,kFALSE,mcPart->Charge()));
    }

  } // end if use  MC truth


  return tracksClone;
}
```

Loop on the particles in the array

consider only final state charged particles

check pt/imp.par selection

store the array

Offline correction

☑ Taken from the D efficiency correction class - under testing

```
//_____________________________________________________________________________
TH2D * AliHFCorrelationCF::ApplyEventMixingCorrection(TH2D * SEHisto, TH2D * MEHisto){
    // function that applies the event mixing correction

    TH2D * OutputHisto;
    NormalizeToPeak(MEHisto);

    OutputHisto = Divide2DHistos(SEHisto,MEHisto);

    return OutputHisto;


}
```

Normalization to the bins at (0,0)

Division of the SE and ME histos

```
//_____________________________________________________________________________
TH2D * AliHFCorrelationCF::NormalizeToPeak(TH2D * inputHisto){
    // function that normalizes the SB distribution to the bin centered at (delta phi, delta eta) = (0,0)

    Int_t* centralbins = FindCentralBin(inputHisto);
    inputHisto->Scale(1./inputHisto->GetBinContent(centralbins[0],centralbins[1]));

    return inputHisto;
}
```

Offline correction

```
//_____________________________________________________________________________
TH2D * AliHFCorrelationCF::Divide2DHistos(TH2D * NumHisto, TH2D * DenomHisto){
    // function that divides two TH2D histos

    Int_t BinsX = NumHisto->GetNbinsX();
    Int_t BinsY = NumHisto->GetNbinsY();
    Double_t Xlowerbin = NumHisto->GetXaxis()->GetBinLowEdge(1);
    Double_t Xupperbin = NumHisto->GetXaxis()->GetBinLowEdge(BinsX+1);
    Double_t Ylowerbin = NumHisto->GetYaxis()->GetBinLowEdge(1);
    Double_t Yupperbin = NumHisto->GetYaxis()->GetBinLowEdge(BinsY+1);

    if((BinsX != DenomHisto->GetNbinsX())||((BinsY != DenomHisto->GetNbinsY())))printf(">>> Warning! Dividing two histos with different binning!");
    if((Xlowerbin != DenomHisto->GetXaxis()->GetBinLowEdge(1))||((Xupperbin != DenomHisto->GetXaxis()->GetBinLowEdge(BinsX+1)))) printf(">>> Warning!
Dividing two histos with different X axis ranges!");
    if((Ylowerbin != DenomHisto->GetYaxis()->GetBinLowEdge(1))||((Yupperbin != DenomHisto->GetYaxis()->GetBinLowEdge(BinsY+1)))) printf(">>> Warning!
Dividing two histos with different Y axis ranges!");
    TString name = NumHisto->GetName();
    name += "_ratio";
    // define the output histo
    TH2D * outputHisto = new TH2D(name.Data(),name.Data(),BinsX,Xlowerbin,Xupperbin,BinsY,Ylowerbin,Yupperbin);

    Double_t ratio = 0;   Double_t ratioerr = 0;
    Double_t numvalue = 0; Double_t denomvalue = 0;
    Double_t numvalerr = 0; Double_t denomvalerr = 0;

    for (Int_t x =1; x<BinsX+1; x++){ // loop on delta phi
           for (Int_t y=1; y<BinsY+1; y++){ // loop on delta eta

            numvalue = NumHisto->GetBinContent(x,y);
            denomvalue = DenomHisto->GetBinContent(x,y);
            numvalerr = NumHisto->GetBinError(x,y);
            denomvalerr = DenomHisto->GetBinError(x,y);

            if(!denomvalue) {

                printf("Error: Dividing by zero - cannot divide histos\n"); return NULL;}

            ratio = numvalue/denomvalue;
            ratioerr = TMath::Sqrt((numvalerr/denomvalue)*(numvalerr/denomvalue) + ratio*ratio * (denomvalerr/denomvalue)*(denomvalerr/denomvalue));

            outputHisto->SetBinContent(x,y,ratio);
            outputHisto->SetBinError(x,y,ratioerr);

        }
    }

    return outputHisto;

}
```

Loop on the bins

Error propagation:: to be checked