ALICE Offline Tutorial

Alice Core Offline

3 March, 2010

Part I
AliRoot

References

- AliRoot "Offline Bible"

  http://aliceinfo.cern.ch/export/download/OfflineDownload/OfflineBible.pdf

- Tutorial page

  http://aliceinfo.cern.ch/Offline/AliRoot/Manual.html#tutorial

Outline

- The ALICE Offline Web Page
- Simulation
  - Generators
  - Configuration (Config.C)
- Reconstruction
- Classes for analysis
  - ESD classes
  - AOD classes
- Practical examples

The ALICE Offline Web Page

ject - Mozilla Firefox

Tools Help

/aliceinfo.cern.ch/Offline

RedHat INFN ROOT CERN AliceOff MLTest MLProd dict Eng CERNph TW_Off DAQLB spires WeatherGE Indico

ALICE Offline

search Offline www

Home | 2Print

General Information
- Meetings
- User Support
- Project Organization
- Offline Policy
- ARTS

AliRoot
- Documentation
- Installation
- Release
- Code
- Macros
- Code Development
- Night builds
- SVN
- Report a Bug

Activities
- Simulation
- Reconstruction
- Visualisation
- Geometry
- Raw Data
- Condition Database
- Alignment
- The Shuttle
- Analysis
- QA
- RecoParams
- Trigger

Detectors
- FMD
- Dimuon
- ITS
- PHOS
- TOF
- TPC
- TRD
- VZERO
- EMCAL
- TZERO
- ZDC
- PMD
- HMPID
- ACORDE

User: Guest (?)

login

Introduction

Welcome to the home page of the ALICE Off-line Project.

This page and the following contain the description of the features of ALICE Off-line environment.

The ALICE Off-line Project has started developing the current framework in 1998. The decision was taken at the time to build the simulation tool for the Technical Design Reports of the ALICE detector using the OO programming technique and C++ as an implementation language.

This lead us to the choice of ROOT as framework and GEANT 3.21 as simulation code. A prototype was quickly built and put in production. The experience with this was positive, and in November 1998 the ALICE Off-line project adopted ROOT as the official framework of ALICE Off-line.

AliRoot is the name ALICE Off-line framework for simulation, reconstruction and analysis. It uses the **ROOT** system as a foundation on which the framework and all applications are built.

Except for large existing libraries, such as GEANT3.21 and Jetset, and some remaining legacy code, this framework is based on the Object Oriented programming paradigm, and it is written in C++.

IMPORTANT! Recently the AliRoot code was transfered to a Subversion (SVN) repository.

HowTO for Subversion repository

Related Links
- ROOT documentation can be accessed through its home page.
- GEANT3 Manual could help you a lot.
- GEANT3 Tutorial to start having fun.
- FLUKA is also accesible.

News
- ALICE main pages are full of information.

Analysis News
- Latest new
  Published on: October 30, 2009
- See all Analysis News

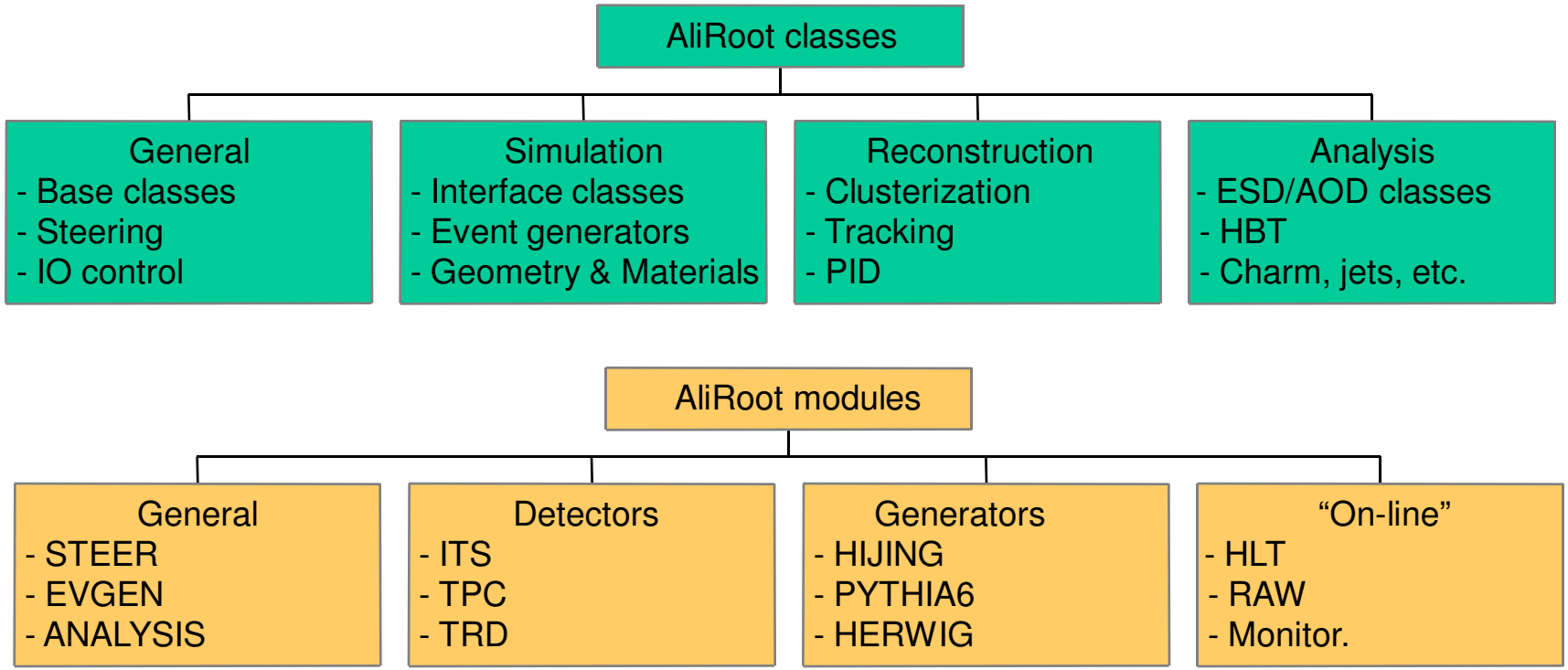
For Page Edition
- Edition rules for OpenCMS users.

Last modified on 6/22/09 by ROOT ROOT (Admin)

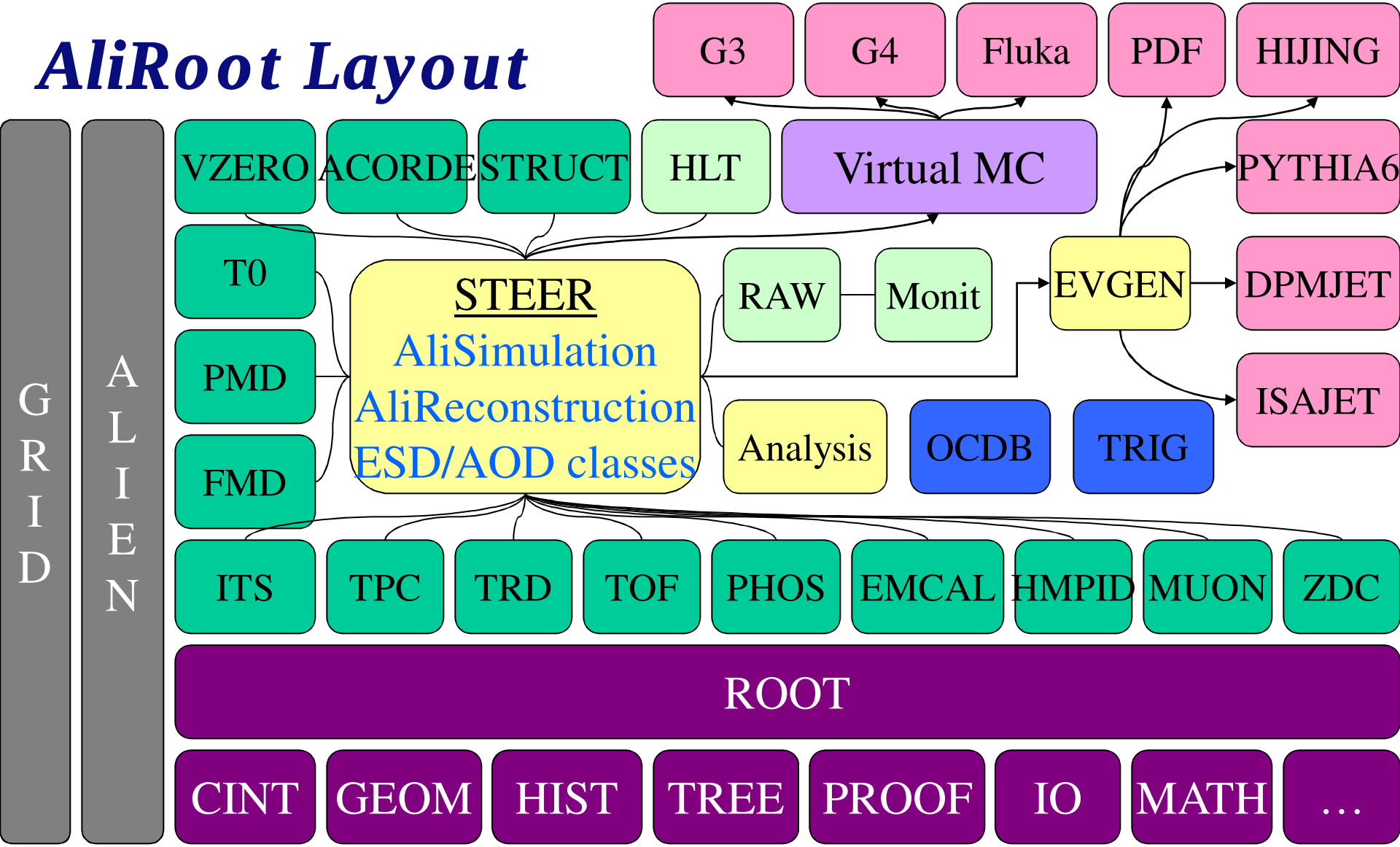
For comments or questions regarding this web page, please contact Alice Webmaster Group. CERN Copyright 2005 - ALICE EXPERIMENT

AliRoot: General Layout

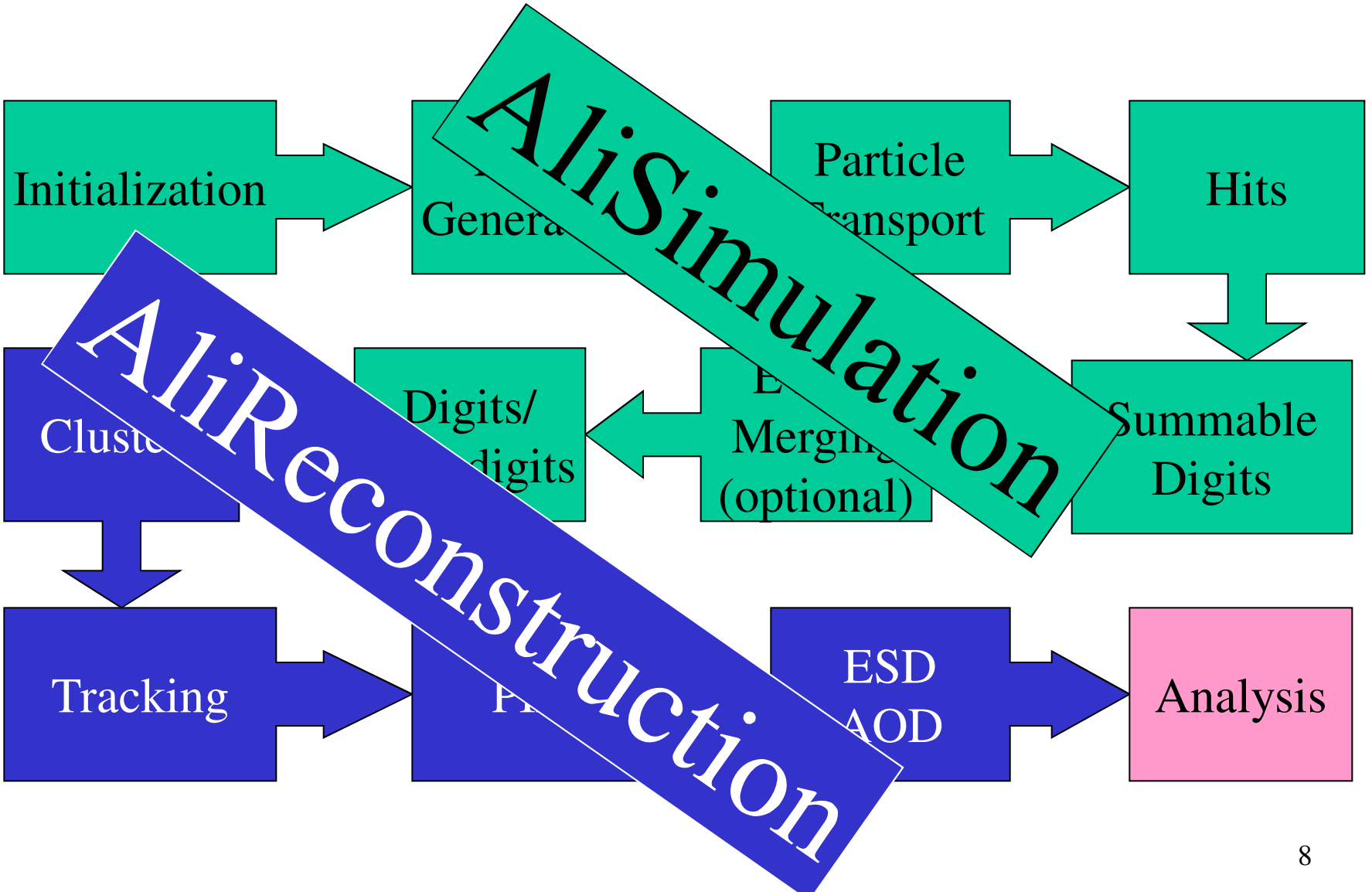
- Root v5-26-00b
- Geant3 v1-11
- AliRoot v4-18-Release



AliRoot Layout



AliRoot: Execution Flow



Config.C: Steering the Simulation

- Sets random seed
- Creates transporter
- Creates RunLoader
- Add MC particle decay model (Pythia6)
- Set up transporter
- Creates and sets up event simulator
- Defines ALICE Magnetic Field
- Defines All materials and geometries/detectors

```
Void Config(){
gRandom->SetSeed(123456789);

new TGeant3TGeo("Transporter");

AliRunLoader *rl =
      AliRunLoader::Open("galice.root",defaultFileNames,"recreat
      ");
gAlice->SetRunLoader(rl);

TVirtualMCDecayer *dec = AliDecayerPythia();
dec->Init();


gMC->SetExternalDecayer(dec);
…
gMC->SetCut("CUTGAM",1.e-3);
…
AliGenHIJINGpara *gen = new AliGenHIJINGpara(100);
gen->Init(); // Registers its self to gAlice

gAlice->SetField(new AliMagFMaps(…);


AliBody *BODY = new AliBODY("BODY","Alice envelope");
// Registers itself to gAlice
```

External Generators: HIJING

- HIJING
  - HIJING (Heavy Ion Jet INteraction Generator) combines
    - A QCD-inspired model of jet production with the Lund model for jet fragmentation
    - Hard or semi-hard parton scatterings with transverse momenta of a few GeV are expected to dominate high energy heavy ion collisions
    - The HIJING model has been developed with special emphasis on the role of mini jets in pp, pA and AA reactions at collider energies

HIJING

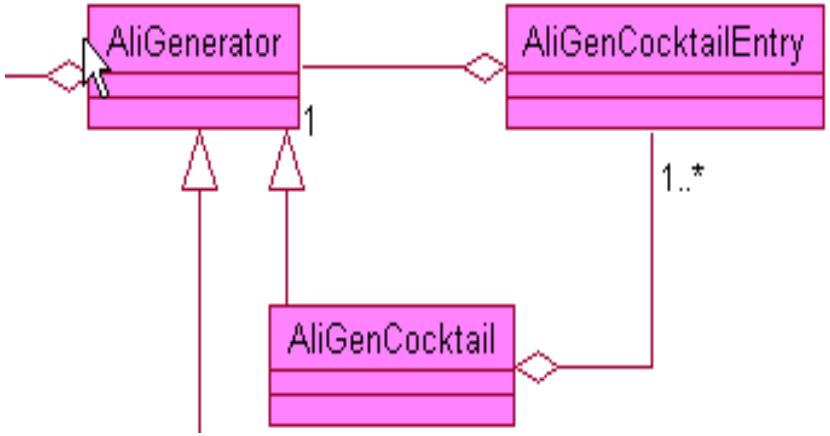
- Hijing used as
  - Underlying event in HI
    - Realistic fluctuations (N,E) from mini-jets
    - Pessimistic multiplicity (dN/dy ~ 6000)
  - Particle Correlation studies
    - Inclusive
    - And in reconstructed jets
  - Nuclear effects
    - Shadowing
    - Quenching (parton energy loss)

pp

- Minimum Bias
  - Pythia, Herwig, Phojet
  - Pythia with ATLAS (or newer) Tuning
- Hard Probes
  - Pythia tuned to NLO (MNR)
    - NLO topology
  - Nuclear modification of structure functions via EKS in LHAPDF
- Pythia preconfigured processes
  - See $ALICE_ROOT/PYTHIA6/PythiaProcesses.h

Event Generator Interfaces

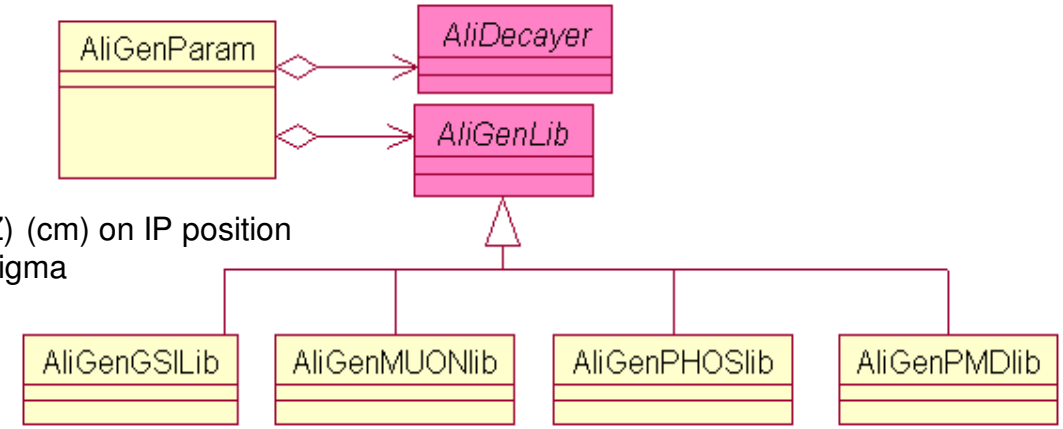
- Cocktail class to assemble events, for example:
  - Underlying event + hard process
  - Different muon sources
  - pA + slow nucleons



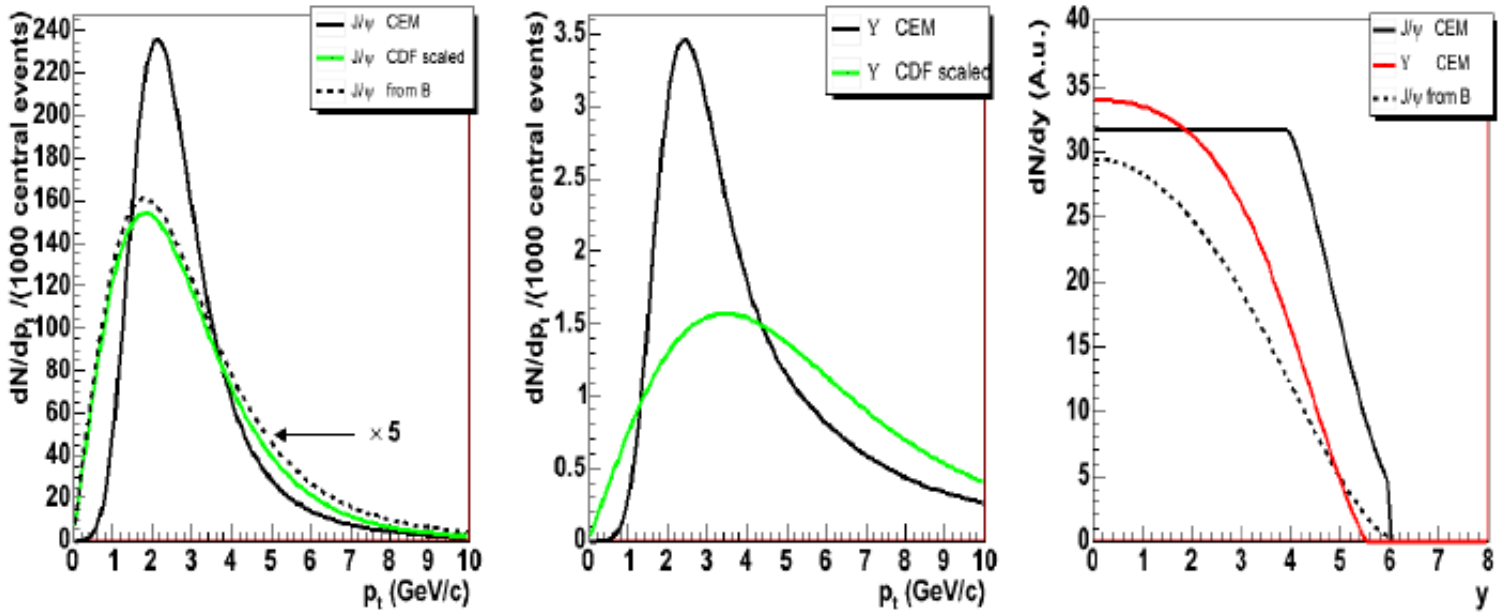
Event Generator Interfaces: Parameterizations

More examples in $ALICE_ROOT/macros/Config_PDC06_MUON.C

```
// The cocktail generator
AliGenCocktail *gener = new AliGenCocktail();

// Phi meson (10 particles)
AliGenParam *phi = new AliGenParam(10,new AliGenMUONlib(),AliGenMUONlib::kPhi,"Vogt PbPb");
phi->SetPtRange(0, 100);
phi->SetYRange(-1., +1.);
phi->SetForceDecay(kDiElectron);

// Omega meson (10 particles)
AliGenParam *omega = new AliGenParam(10,new AliGenMUONlib(),AliGenMUONlib::kOmega,"Vogt PbPb");
omega->SetPtRange(0, 100);
omega->SetYRange(-1., +1.);
omega->SetForceDecay(kDiElectron);

// Adding all the components of the cocktail
gener ->AddGenerator(phi,"Phi",1);
gener ->AddGenerator(omega,"Omega",1);

// Settings, common for all components
gener ->SetOrigin(0, 0, 0);          // vertex position
gener ->SetSigma(0, 0, 5.3);         // Sigma in (X,Y,Z) (cm) on IP position
gener ->SetCutVertexZ(1.);           // Truncate at 1 sigma
gener ->SetVertexSmear(kPerEvent);
gener ->SetTrackingFlag(1);
gener >Init();
```



Example: MUON Library



Parameterisations (pt and y of given particle):

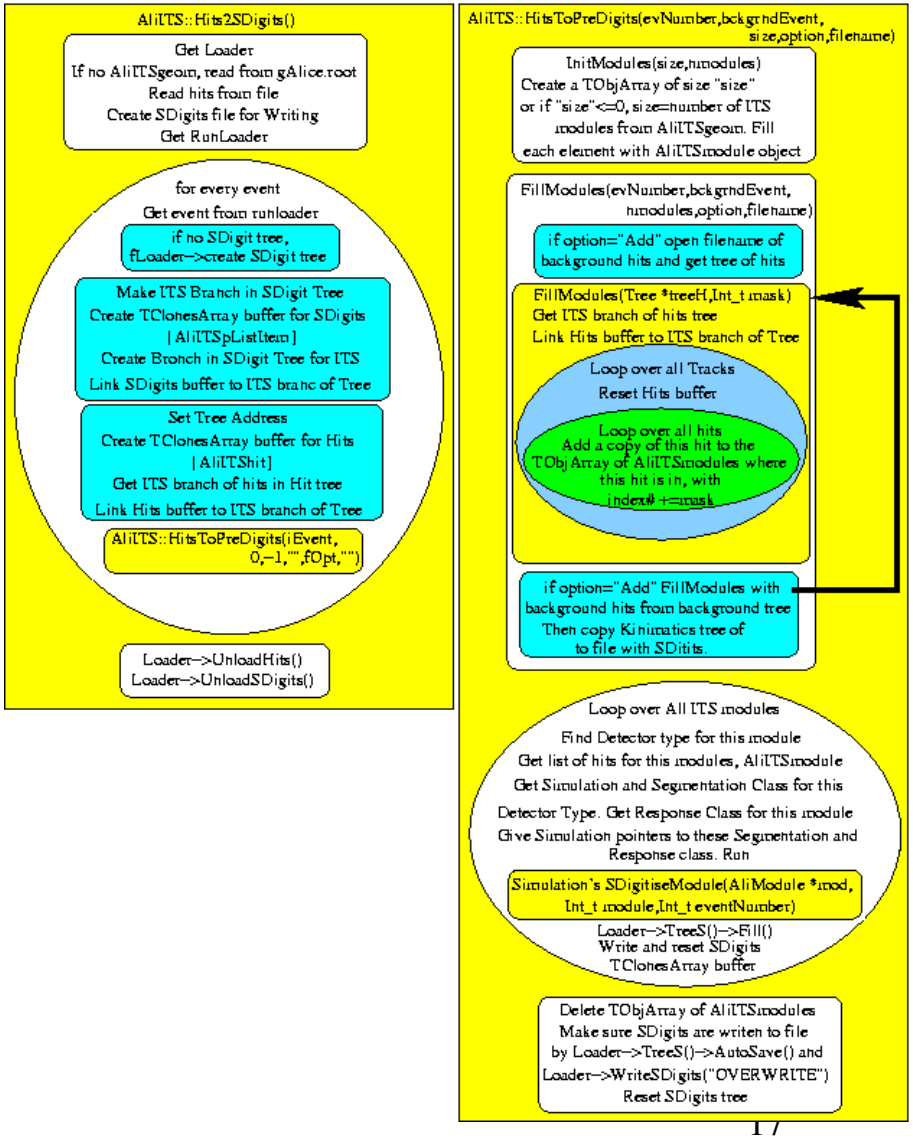
kPhi, kOmega, kEta,
kJpsi, kJpsiFamily, kPsiP, kJpsiFromB,
kUpsilon, kUpsilonFamily, kUpsilonPP,
kCharm, kBeauty,
kPion, kKaon

Run MC: Particle Transport

- Particles are transported through geometry
- At each step, a call to StepManager of the class whose volume the particle is in
- Example: AliITSvPPRasymmFMD::StepManager
  - If not sensitive volume return
  - If not charged return
  - Record Hit, particle Position (start & end of this step), Momentum, Energy lost during last step, Sub-detector in, Time of Flight, Status, Charge, and Track Number
  - In the ITS, hits can also be "merged"
- Hits might be deleted after SDigitization

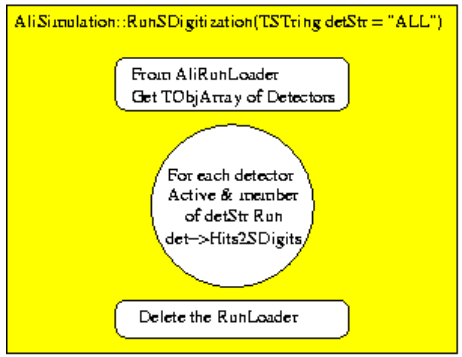
Simulation: Summable Digits

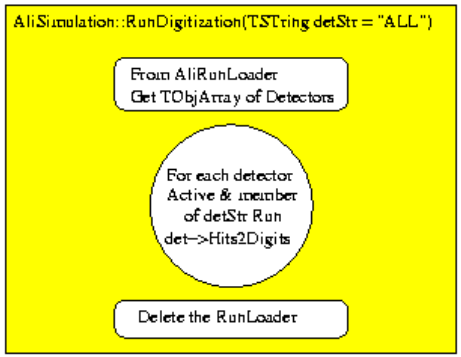
- Apply all detector response simulation which allows results to be "merged"
  - Do not add noise
  - Do not convert AtD
  - Do not apply thresholds
- Some detectors use hits as SDigits
  - For PHOS, EMCAL the hits are already summed
  - HMPID saves rings/photons

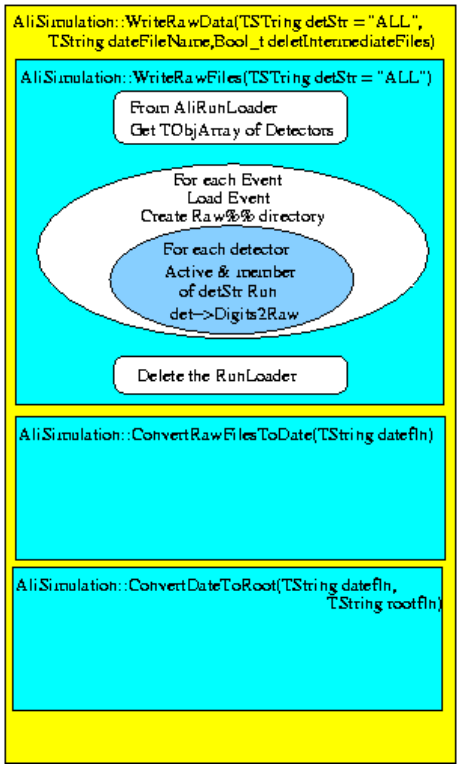


Digitization

- Adds noise
  - Random for SPD, SSD
  - Correlated for SDD
- Applies threshold
  - Simple threshold for SPD,SSD
  - 2 level threshold for SDD
- Applies ADC-ing
  - 10 bit for SDD, SSD
  - 10⇒8 conversion for SDD
- Zero suppression
  - 2 integer coordinates, 1 integer signal
  - Simulation + info by detector type



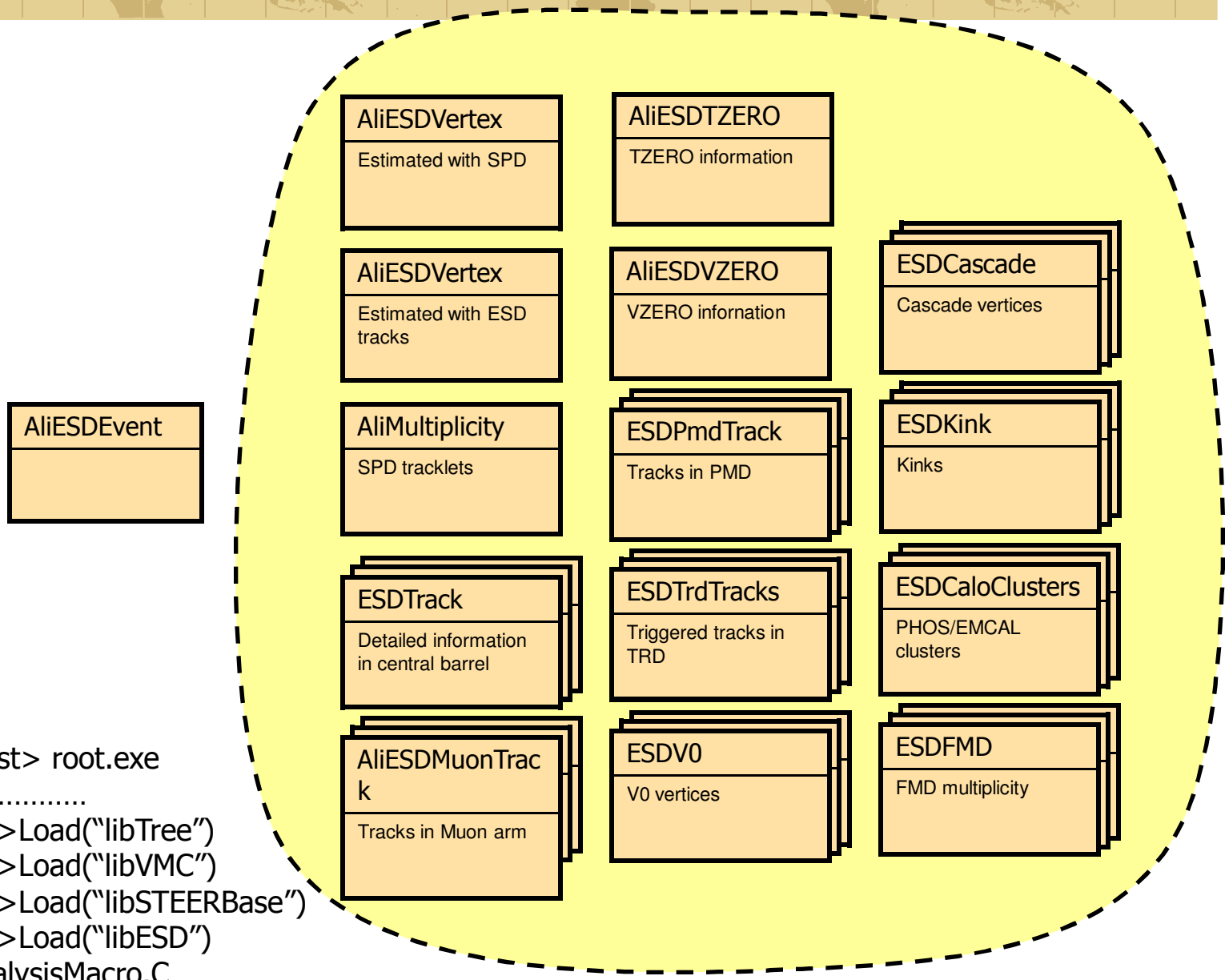




Reconstruction

- Possible inputs
  - DATE DDL files (only for test)
  - RAW DATE file (only for test)
  - RAW rootified file (standard format)
  - MC/Digit files (standard for simulated data)
- Local/Detector reconstruction (Files <DET>.RecPoints.root)
  - Calibration
  - Clusterisation
  - Cluster splitting…
- Vertex finder: Fills ESD
  - Primary vertex (Z coordinate) found in SPD, and/or T0.
- Tracking (HLT and/or Barrel), filling of ESD
  - Gives final vertex from tracks and secondary vertexes.
  - HLT uses Conformal mapping (or similar) or a fast Kalman filter
  - Final tracking is a full Kalman filter
    - In: {TPC→ITS}→ Out: {ITS→TPC →[TRD→TOF →(EMCAL|HMPID|PHOS)]}→ Refit: {TOF→TRD→TPC→ITS}
    - MUON.
- Combined (Bayesian) PID: Fills ESD

The ESD



Ideally: user@host> root.exe

.....

```
root[0] gSystem->Load("libTree")
root[1] gSystem->Load("libVMC")
root[2] gSystem->Load("libSTEERBase")
root[3] gSystem->Load("libESD")
root[4] .x AnyAnalysisMacro.C
```

Common base classes for ESDs and AODs

AliVEvent

AliESDEvent **AliAODEvent**

AliVHeader

AliESDHeader **AliAODHeader**

AliVParticle

AliExternalTrackParam **AliAODTrack** **…**

AliESDtrack

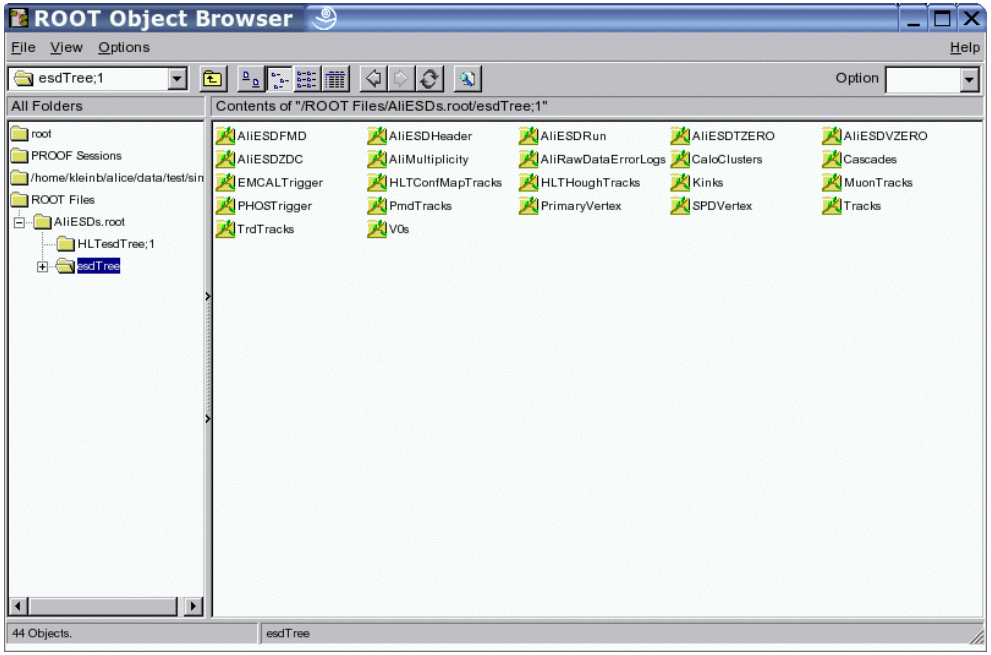
- standard access to containers
- common getters and setters

Current content of the standard AOD

AliAODEvent	contains an (extendable) TList
AliAODHeader	event information
AliAODTrack	TClonesArray of tracks
AliAODVertex	TClonesArray of vertices
AliAODJet	TClonesArray of jets
AliAODTracklets	Container for SPD tracklets

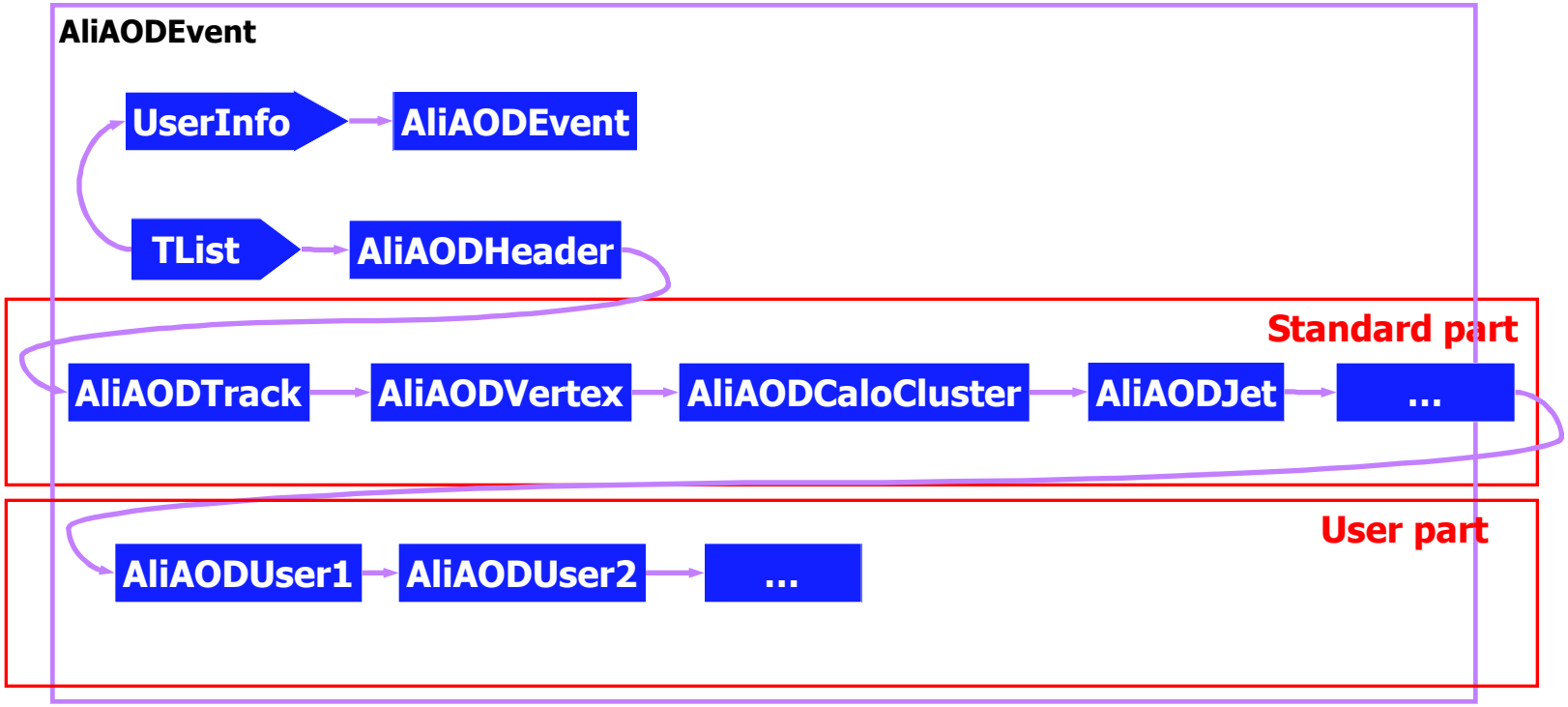
The ESD Layout

- AliESDEvent()
  - Defines the interface to the ESD tree content
    - Mainly contains a TList of pointers
      - E.g. to TClonesArray of tracks
    - Access to standard contents
      - ...ReadFromTree(...)
      - ...GetEntry(Int_t)
      - AliESDEvent::GetTrack(Int_t)
  - Similar structure in the AODs

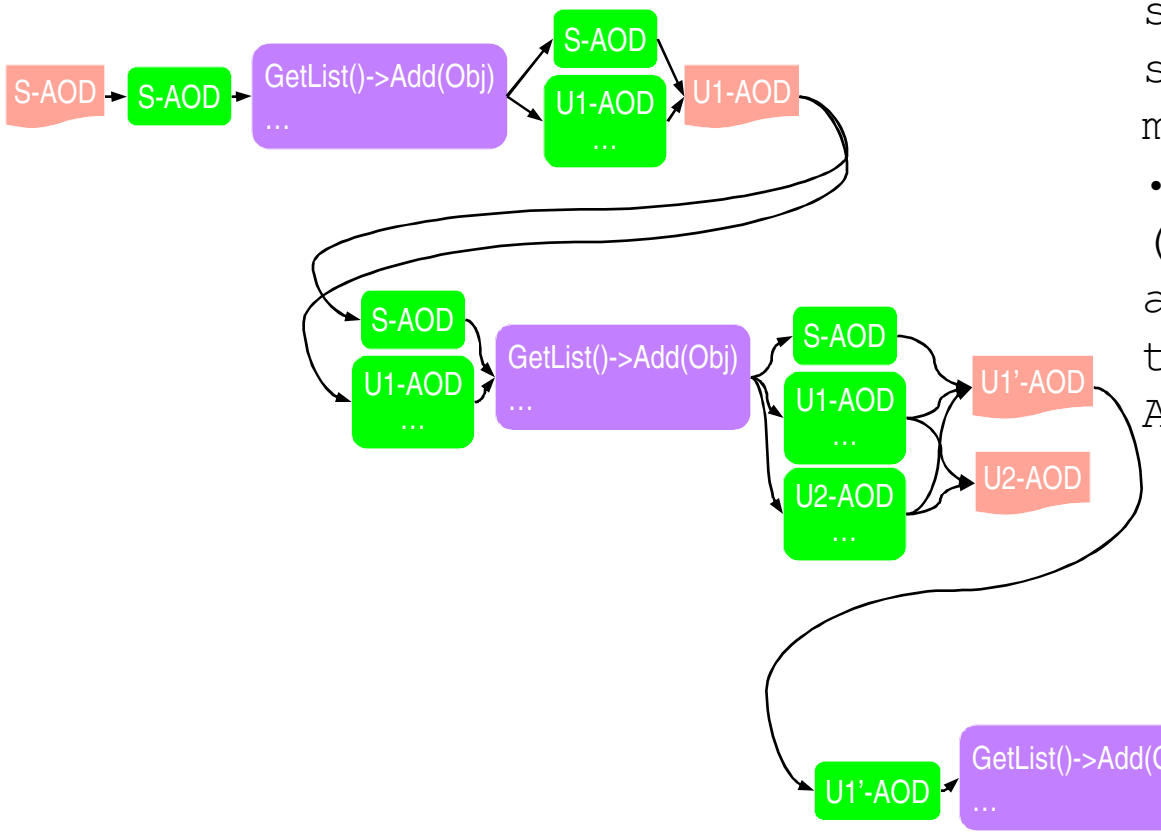


General idea: ESD & AOD

- Very simple… use a list



Extending the AOD



Flexible and Extendable

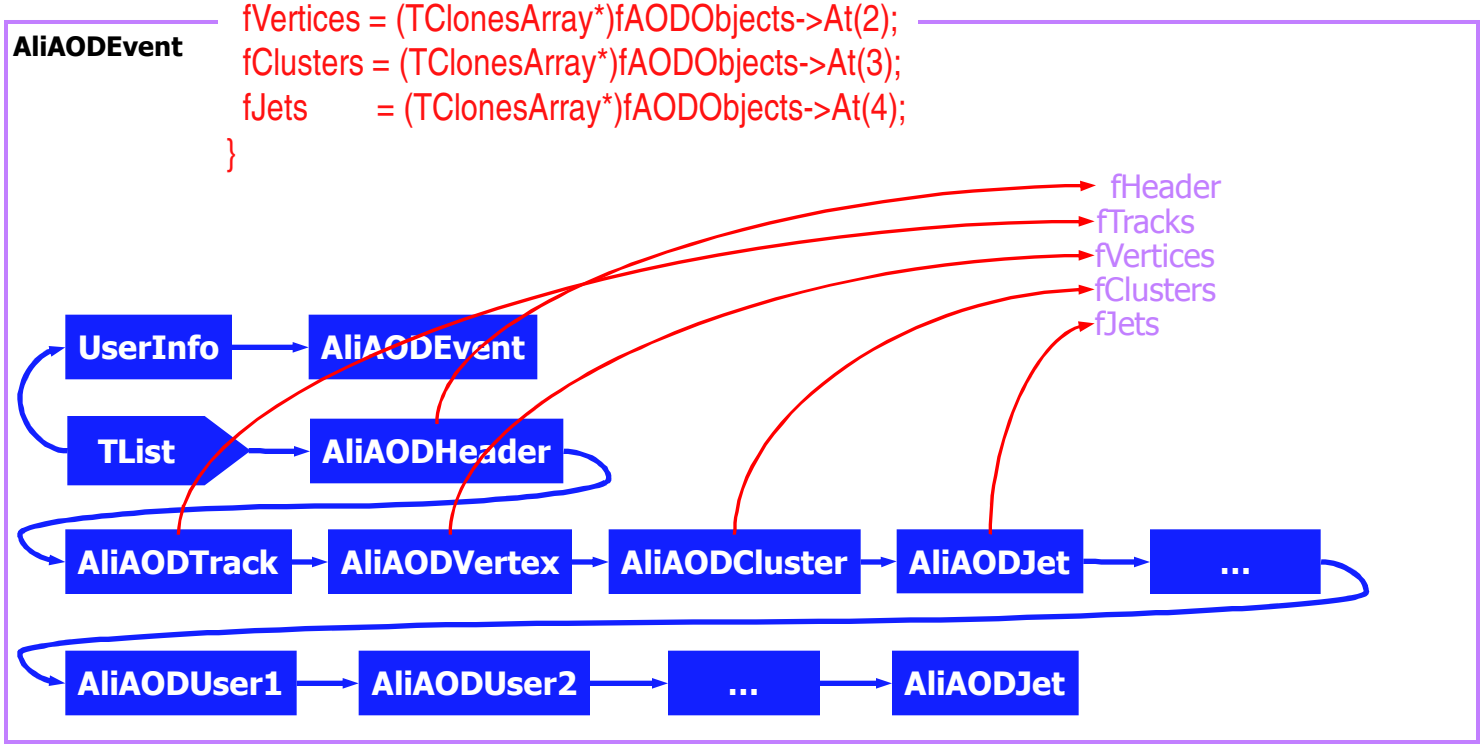
- users can just use the standard AOD (S-AOD) or start from it to obtain more detailed results
- this new information (U-AOD) can be stored alongside the S-AOD in the TList (= in the same AOD)

- The idea is that every user can extend the AODs with "non-standard" objects

Reading back

```
void AliAODEvent::GetStdContent() const
{
  // set pointers for standard content

  fHeader  = (AliAODHeader*)fAODObjects->At(0);
  fTracks   = (TClonesArray*)fAODObjects->At(1);
  fVertices = (TClonesArray*)fAODObjects->At(2);
  fClusters = (TClonesArray*)fAODObjects->At(3);
  fJets       = (TClonesArray*)fAODObjects->At(4);
}
```

```
AliAODTrack* AliAODEvent::GetTrack(Int_t i) const
{ return (AliAODTrack*) fTracks->At(i); }
```



- …and the same user also can access the “non-standard” objects from the previous slide

AliESDEvent and AliESDtrack classes

- Accumulation and exchange of tracking information among the barrel detectors
- Contained in the ESD and used for physics analysis

Class AliESDtrack : public AliExternalTrackParam

- final params
- reconstruction status flags
- length, time, combined PID
- vertex constrained params
- impact parameters & cov.matrix
- params at the outer TPC wall
- params at the inner TPC wall
- …
- detector specific info (chi2, num.of clusters, PID…)

ESD Example: Loop on the tracks

```
void test1(const char * fname ="AliESDs.root") {
  TFile * file = TFile::Open(fname);
  TTree * tree = (TTree*)file->Get("esdTree");

  AliESDEvent * esd = new AliESDEvent();      // The signal ESD object is put here
  esd->ReadFromTree(tree);

  Int_t nev = tree->GetEntries();
  for (Int_t iev=0; iev<nev; iev++) {

    tree->GetEntry(iev); // Get ESD

    Int_t ntrk = esd->GetNumberOfTracks();
    for(Int_t irec=0; irec<ntrk; irec++) {
      AliESDtrack * track = esd->GetTrack(irec);
      cout << "Pt: " << track->Pt() << endl;
    }

  }

  file->Close();
}
```

AOD Example: Loop on the tracks

```
AliAODEvent *event= new AliAODEvent();          //The reconstructed events are
TTree *aodTree = …;                             //stored in TTrees  (and so can be
“chained”)
event->ReadFromTree(aodTree);                   //Sets the branch adresses for
                                                //AliAODEvent content

Int_t i=0;
while (aodTree->GetEvent(i++)) {                //loop over the reconstructed events
    …                                            //select run, event number etc…
  if (event->GetEventType() != … )  continue;   //select the event type
  AliAODVertex *primary=event->GetPrimaryVertex();
  if (/* some cuts on the primary vertex */)  continue;

  Int_t ntracks=event->GetNumberOfTracks();
  for (i=0; i<ntracks; i++) {                   //loop over ESD tracks (or kinks, V0s…)
    AliAODTrack *track=event->GetTrack(i);
    if (/* select tracks according to proper selection (quality) criteria  */) {
       …                                        //do whatever with the selected tracks
    }
  }
```

ESD Example: PID

```
AliESDEvent *event=new AliESDEvent();                    //The reconstructed events are
TTree *esdTree = …;                                      //stored in TTrees  (and so can be “chained”)
event->ReadFromTree(esdTree);                            //Sets the branch adresses for AliESDEvent content
Int_t i=0;
while (esdTree->GetEvent(i++) {                          //loop over the reconstructed events
    …                                                    //event selection…

    Double_t priors[AliPID::kSPECIES]={…}                //A set of a priori probabilities
    AliPID::SetPriors(priors);

    Int_t ntracks=event->GetNumberOfTracks();
    for (i=0; i<ntracks; i++) {                          //loop over ESD tracks (or kinks, V0s …)
        AliESDtrack *track=event->GetTrack(i);

        ULong _t status=AliESDtrack::kTPCpid | AliESDtrack::kTOFpid;
        if ((track->GetStatus()&status) != status)  continue;   //select tracks with the proper status
        if ( … )  continue;                              //some other selection (quality) criteria


        Double_t probDensity[AliPID::kSPECIES];  track->GetESDpid(probDensity);
        AliPID pid(probDensity);

        Double_t   pp=pid.GetProbablity(AliPID::kProton);   // probability to be a proton
        Double_t   pk=pid.GetProbability(AliPID::kKaon);    // probability to be a kaon
        …
        if (pp > 1./AliPID::kSPECIES) { /* this is a proton */}
    }
}
```

Pieces needed for simulation/reconstruction

- sim.C macro that runs the simulation
  - AliSimulation sim;
  - sim.Run();
- rec.C macro that runs the reconstruction
  - AliReconstruction rec;
  - rec.Run();
- Config.C macro that configures the simulation

Files produced in Sim/Rec

- Simulation
  - galice.root: Event header
  - Kinematics.root: MC particles
  - DET.Hits.root: Hits of MC particles in sensitive regions
  - DET.(S)Digits.root: Induced signal in the electronics
  - raw.root: Digits converted to the RAW data format (optional)
- Reconstruction
  - galice.root: Event header
  - DET.RecPoints.root: Clusters
  - AliESDs.root: "Event Summary Data" Reconstruction output
  - AliESDfriends.root: Reconstruction debugging information (optional)
  - AliAOD.root: "Analysis Object Data" Condensed reconstruction output (optional)

Exercises from $ALICE_ROOT/test

- Particle gun (test/gun)
- Generation from a kinematics tree (test/genkine)
- Event merging (test/merge)
- Proton-proton benchmark (test/ppbench)
- Pb-Pb benchmark (test/PbPbbench)
- Proton-proton simulation and reconstruction loading the libraries in Root (test/pploadlibs)

Particle gun: $ALICE_ROOT/test/gun

Cocktail generator

```
AliGenCocktail *gener = new AliGenCocktail();
gener->SetPhiRange(0, 360);
 …
gener->SetThetaRange(thmin,thmax);
gener->SetOrigin(0, 0, 0);  //vertex position
gener->SetSigma(0, 0, 0);   //Sigma in (X,Y,Z)
      (cm) on IP position
```

Components

```
AliGenFixed *pG1=new AliGenFixed(1);
pG1->SetPart(2212);
pG1->SetMomentum(2.5);
pG1->SetTheta(109.5-3);
pG1->SetPhi(10);
gener->AddGenerator(pG1,"g1",1);
```

Simulation

```
AliSimulation sim;
sim.SetMakeSDigits("TRD TOF PHOS HMPID EMCAL
      MUON FMD ZDC PMD T0 VZERO");
sim.SetMakeDigitsFromHits("ITS TPC");
sim.SetWriteRawData("ALL","raw.root",kTRUE)
```

Reconstruction from raw

```
AliMagFMaps* field = new
      AliMagFMaps("Maps","Maps", 2, 1.,
      10., AliMagFMaps::k5kG);
AliTracker::SetFieldMap(field,kTRUE);
AliReconstruction reco;
reco.SetUniformFieldTracking(kFALSE);
reco.SetWriteESDfriend();
reco.SetWriteAlignmentData();
AliTPCRecoParam * tpcRecoParam =
      AliTPCRecoParam::GetLowFluxParam
      ();
AliTPCReconstructor::SetRecoParam(tpcR
      ecoParam);
reco.SetInput("raw.root");
reco.SetWriteAOD();
reco.Run();
```

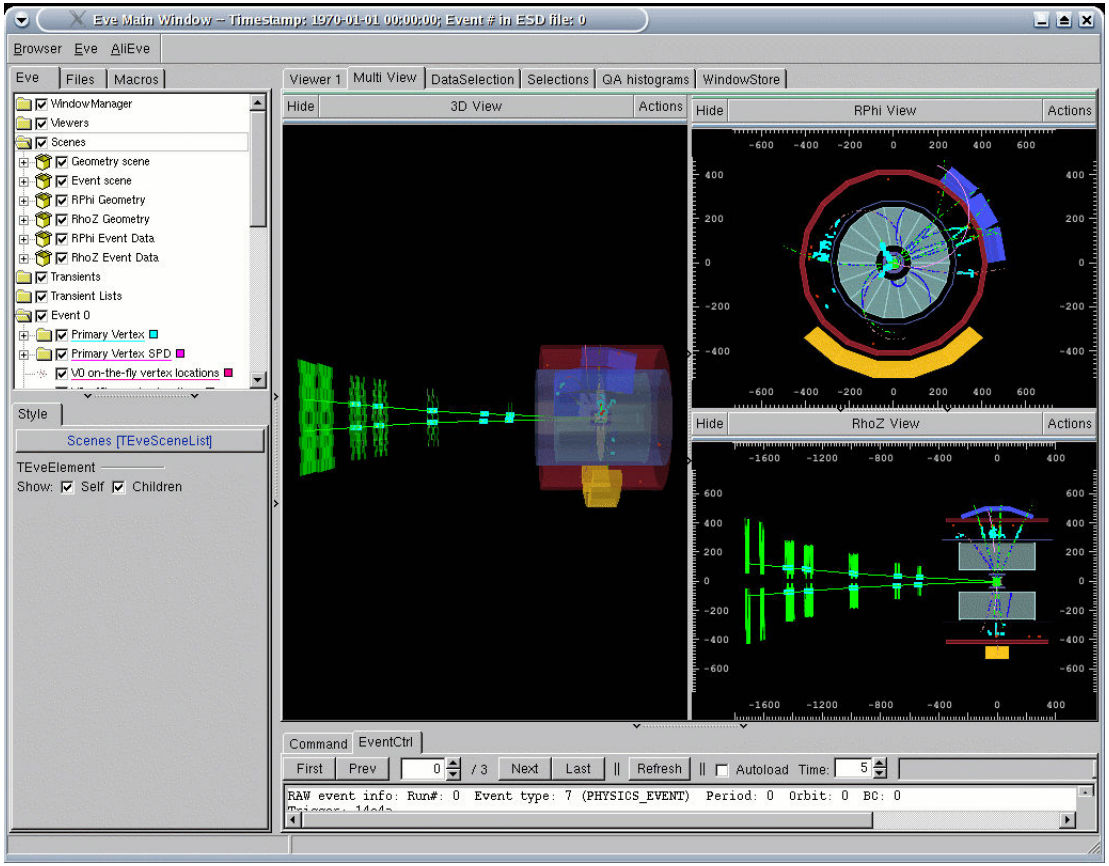
Exercise

- Modify the example to test the identification of muons in the barrel detectors
- Modify test.C to print only the muon tracks

Visualization

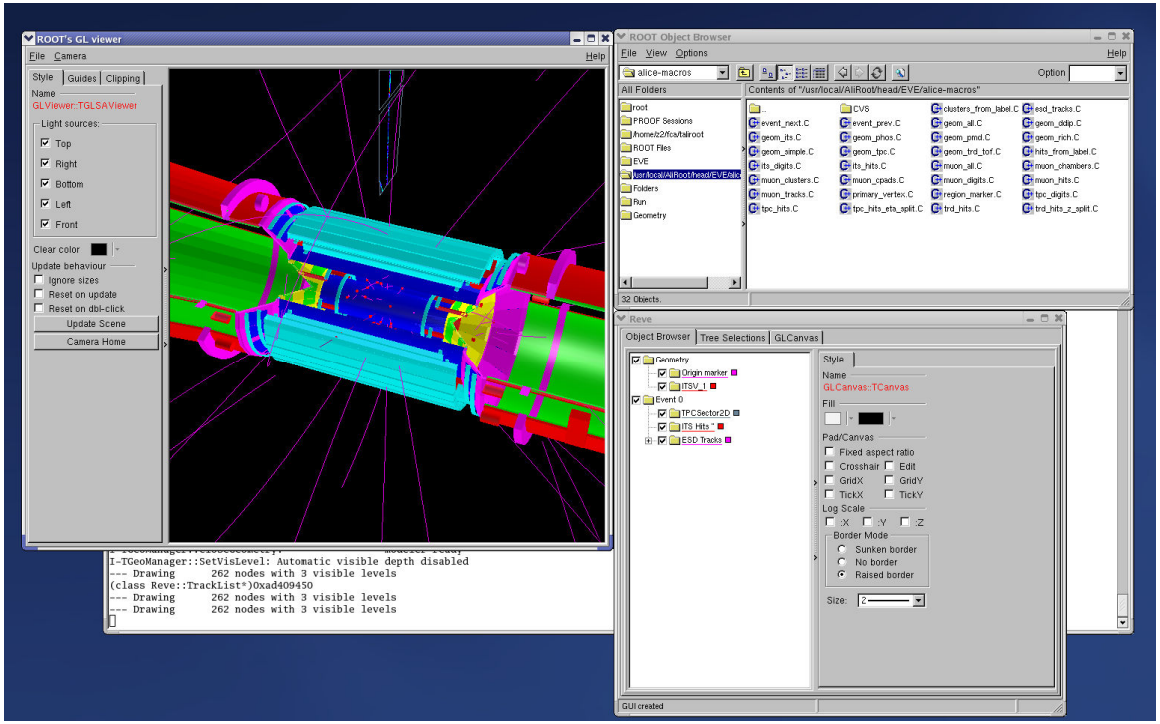
http://aliceinfo.cern.ch/Offline/Activities/Visualisation/

- Usage
  - alieve
  - .x visscan_local.C, visscan_raw.C, visscan_mcideal.C



Visualization – II

- Usage
  - alieve
  - .x alieve_init.C
  - Use then the macros in the EVE folder in TBrowser



Generation from a kinematics tree

- Creation of a kinematics tree: Pythia events containing D*→D0 π
  - See test/genkine/gen/fastSim.C
  - Selection of the specific decay modes via

```
AliPythia * py= AliPythia::Instance();
py->SetMDME(737,1,0); //forbid
    D*+->D+ + pi0
py->SetMDME(738,1,0); //forbid
    D*+->D+ + gamma
```

  - Selection of desired events in the event loop

- Simulation using the kinematic tree: see Config.C

```
AliGenExtFile *gener = new AliGenExtFile(-1);
 AliGenReaderTreeK * reader = new
      AliGenReaderTreeK();


reader->SetFileName("galice.root");
reader->AddDir("../gen");
gener->SetReader(reader);
```

- Reconstruction

Exercise

- Modify the example to select Pythia events containing the decay $\Lambda_c^+ \rightarrow pK\pi$
- Obtain the list of decays with

```
AliPythia * py = AliPythia::Instance()
py->Pylist(12); >> decays.txt
```

  - It is also in PYTHIA6/AliDecayerPythia.cxx
- Generate raw data
- Reconstruct from raw data

Event merging: $ALICE_ROOT/test/merge

- Generate & reconstruct underlying events (./backgr)
  - Simulation (full chain up to Digits)
    - AliSimulation sim;
    - sim.Run(2);
  - Reconstruction
    - AliReconstruction rec;
    - rec.Run();
- Generate, merge & reconstruct signal events (./signal)
  - Simulation (with event merging)
    - AliSimulation sim;
    - sim.MergeWith("../backr/galice.root",3);
    - sim.Run(6);
  - Reconstruction
    - AliReconstruction rec;
    - rec.Run();

Event merging: test.C

```
void test(const char * sdir ="signal",
          const char * bdir ="backgr") {

 TStopwatch timer;
 timer.Start();
 TString name;

 // Signal file, tree, and branch
 name = sdir;
 name += "/AliESDs.root";
 TFile * fSig = TFile::Open(name.Data());
 TTree * tSig = (TTree*)fSig->Get("esdTree");

 AliESDEvent * esdSig = new AliESDEvent; // The signal ESD
 esdSig->ReadFromTree(tSig);

 // Run loader (signal events)
 name = sdir;
 name += "/galice.root";
 AliRunLoader* rlSig = AliRunLoader::Open(name.Data());

 // Run loader (underlying events)
 name = bdir;
 name += "/galice.root";
 AliRunLoader* rlUnd = AliRunLoader::Open(name.Data(),"Underlying");

 // gAlice
 rlSig->LoadgAlice();
 rlUnd->LoadgAlice();
 gAlice = rlSig->GetAliRun();

 // Now load kinematics and event header
 rlSig->LoadKinematics();
 rlSig->LoadHeader();
 rlUnd->LoadKinematics();
 rlUnd->LoadHeader();

 // Loop on events: check that MC and data contain the same number of events
 Long64_t nevSig = rlSig->GetNumberOfEvents();
 Long64_t nevUnd = rlUnd->GetNumberOfEvents();
 Long64_t nSigPerUnd = nevSig/nevUnd;

 cout << nevSig << " signal events" << endl;
 cout << nevUnd << " underlying events" << endl;
 cout << nSigPerUnd << " signal events per one underlying" << endl;

 for (Int_t iev=0; iev<nevSig; iev++) {
  cout << "Signal event " << iev << endl;
  Int_t ievUnd = iev/nSigPerUnd;
  cout << "Underlying event " << ievUnd << endl;

  // Get signal ESD
  tSig->GetEntry(iev);
  // Get underlying kinematics
  rlUnd->GetEvent(ievUnd);

  // Particle stack
  AliStack * stackSig = rlSig->Stack();
  Int_t nPartSig = stackSig->GetNtrack();
  AliStack * stackUnd = rlUnd->Stack();
  Int_t nPartUnd = stackUnd->GetNtrack();

  Int_t nrec = esdSig->GetNumberOfTracks();
  cout << nrec << " reconstructed tracks" << endl;
  for(Int_t irec=0; irec<nrec; irec++) {
   AliESDtrack * track = esdSig->GetTrack(irec);
   UInt_t label = TMath::Abs(track->GetLabel());
   if (label>=10000000) {
        // Underlying event. 10000000 is the
        // value of fkMASKSTEP in AliRunDigitizer

        label %=10000000;
        if (label>=nPartUnd) continue;
        TParticle * part = stackUnd->Particle(label);

   }
   else {
        cout << " Track " << label << " from the signal event" << endl;
        if (label>=nPartSig) continue;
        TParticle * part = stackSig->Particle(label);
        if(part) part->Print();
   }

  }

 }
 fSig->Close();

 timer.Stop();
 timer.Print();
}
```

PYTHIA preconfigured processes

- Heavy Flavors (open)
  - kPyCharm, kPyBeauty
  - kPyCharmUnforced, kPyBeautyUnforced
- kPyCharmPbPbMNR, kPyD0PbPbMNR, kPyDPlusPbPbMNR, kPyBeautyPbPbMNR, kPyCharmpPbMNR, kPyD0pPbMNR, kPyDPluspPbMNR, kPyBeautypPbMNR, kPyCharmppMNR, kPyD0ppMNR, kPyDPlusppMNR, kPyBeautyppMNR
- Heavy Flavor (resonances)
  - kPyJpsi, kPyJpsiChi
- Minimum Bias
  - kPyMb, kPyMbNonDiffr
- Jets and high-pT gammas
  - kPyJets, kPyDirectGamma,
- W
  - kPyW