Part II
Analysis framework

A.Gheata, A.Morsch, C-K. Boesing

v2.1: 22.03.2013

These slides + examples: https://indico.cern.ch/conferenceDisplay.py?confId=242509

Prerequisites

- Login to your AFS account on LXPLUS

> ssh -X user@lxplus.cern.ch

- Enable AliRoot v5-XX-Release

> source /afs/cern.ch/alice/caf/caf-lxplus.sh v5-XX-Release

- Copy the tutorial tarball locally:

> cp /afs/cern.ch/user/a/agheata/public/analysis-tutorial.tgz

> tar xvzf analysis-tutorial.tgz

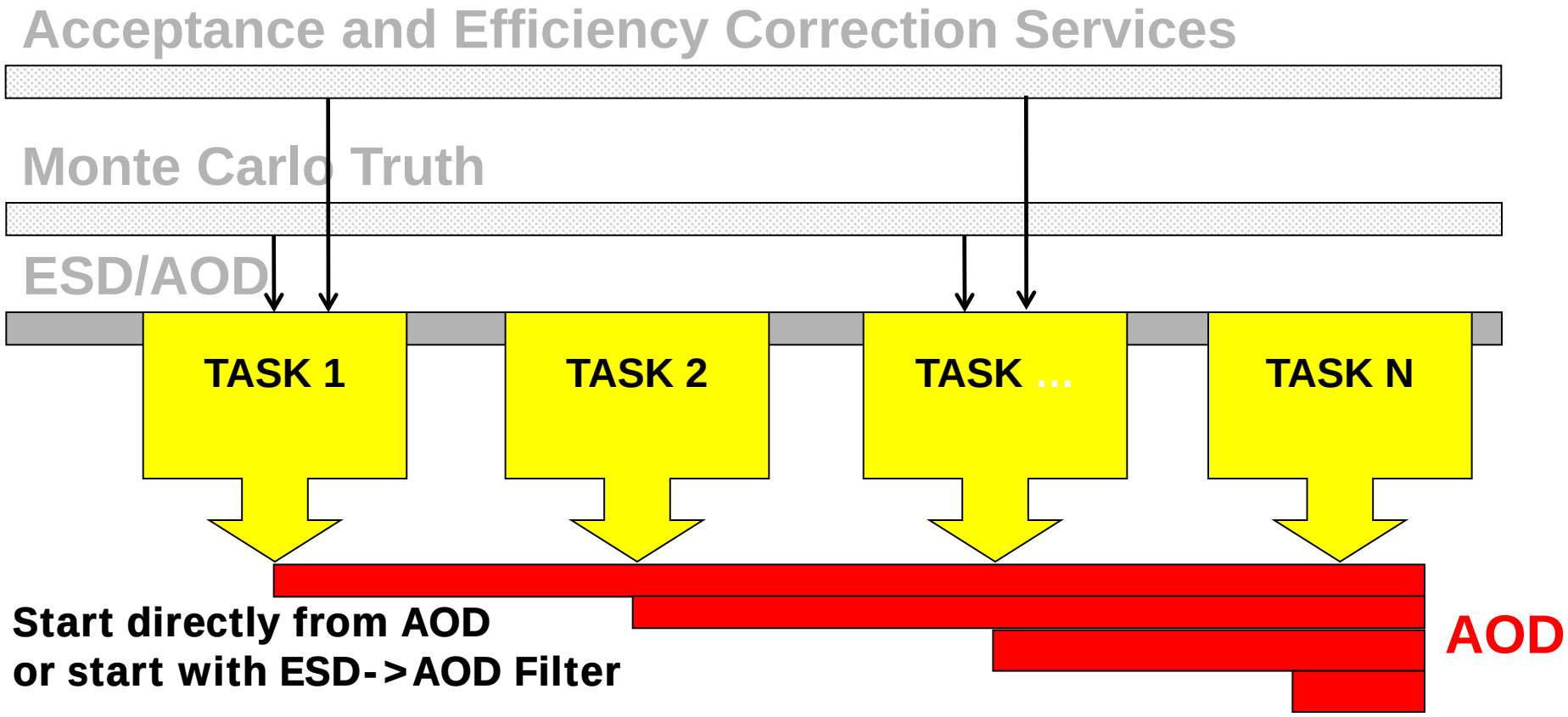
Analysis

- Software
  - AliRoot
    - **Specialized ROOT for ALICE**
    - **AliRoot = ROOT + ALICE libraries**
  - Your code
- What is the data
  - Usually ESD, AOD or Monte Carlo kinematics (MC truth)
- Where does your analysis code run?
  - Local = On your machine
  - In PROOF ("Parallel ROOT Facility")
    - **Parallel analysis on a cluster**
    - **Not related to the Grid**
  - In the AliEn ("Alice Environment") Grid
    - **AliEn is the software of ALICE to access the Grid**
    - **As a user job or in an organized analysis**

What is organized analysis

- Centrally coordinated analysis “train”
  - Collected analysis tasks (“train-wagons”) pass over the data
  - No chaotic request of data
    - Most efficient way for many analysis tasks to read and process the full data set, in particular if resources are sparse
  - Optimise CPU/IO ratio
    - User can rely on previous “service” and PWG tasks
  - Check for data integrity

Analysis trains



Possibility to write DeltaAODs

Example: PWG trains

The LEGO framework

ALICE Analysis Trains

Welcome **agheata** - Help

PWG	Train name	I'm in	Last run	Description	Train operator(s)
CF	CF_PbPb		15 Mar 12	Train for data PbPb running	jgrosseo, miweber
CF	CF_PbPb_MC		14 Mar 12		jgrosseo, miweber
CF	CF_PbPb_MC_AOD		16 Mar 12		jgrosseo, miweber
CF	CF_pp		24 Feb 12	Train for AOD pp correlation analyses	esicking, jgrosseo
CF	CF_pp_MC		18 Feb 12	Train for AOD MC pp correlation analyses	esicking, jgrosseo
GA	GA_PbPb		07 Mar 12		mcosenti
GA	GA_PbPb_MC				mcosenti
GA	GA_pp		16 Mar 12		mcosenti
GA	GA_pp_MC				mcosenti
HF	D2H_PbPb		21 Mar 12	D2H train for PbPb data analysis	jgrosseo, zconesa
HF	D2H_pp			D2H train for pp data analysis	jgrosseo, zconesa
HF	Electrons_PbPb			train for HFE PbPb	ssakai
HF	Electrons_pp				jgrosseo, sma
HF	Muons_PbPb				cheshkov
JE	Jets_PbPb		22 Mar 12	Jet analysis train for 2010 PbPb data	jgrosseo, kleinb, mverweij
JE	Jets_PbPb_2011		22 Mar 12	Jet Train for PbPb 2011	kleinb, mverweij
JE	Jets_PbPb_AOD			Train for Jet Analysis on AODs	kleinb, mverweij
JE	Jets_pp		09 Mar 12	Train for jets in pp	kleinb, vajzerm
JE	Jets_pp_MC		11 Mar 12	Jet train on pp simulated data	jgrosseo, kleinb, vajzerm
LF	LF_PbPb				janielsk
LF	LF_PbPb_AOD		19 Mar 12		delia, mnicassi
LF	LF_PbPb_MC				janielsk
LF	LF_PbPb_MC_AOD		23 Mar 12		delia, jgrosseo, mnicassi
PP	QATrain				mgheata
ZZ	Devel_1			Development testing train #1	jgrosseo
ZZ	Devel_2		21 Oct 11	Development testing train #2	jgrosseo
ZZ	Devel_3			Development testing train #3	

https://alimonitor.cern.ch/trains/

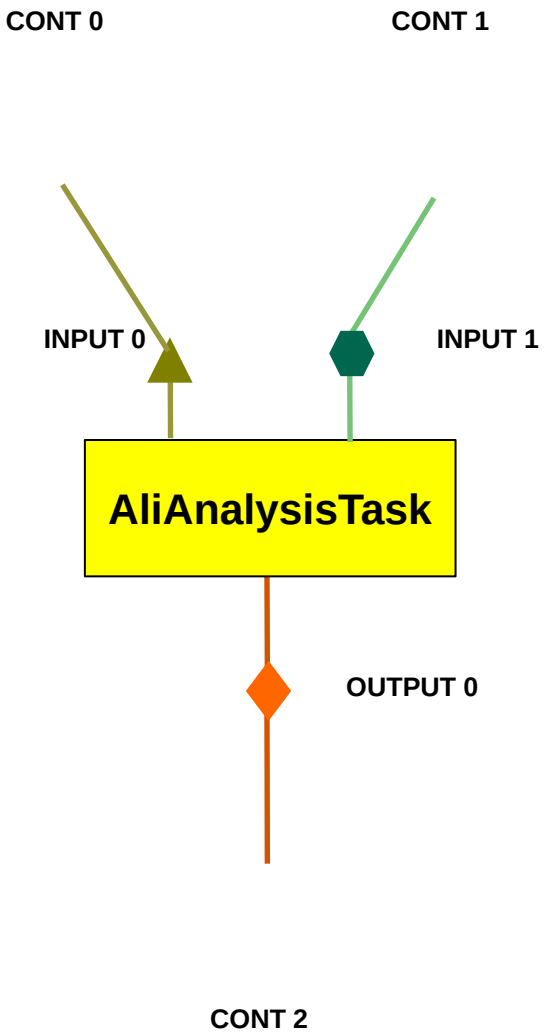
What the Analysis framework does in ALICE

- Transparent access to all resources with the same code
  - **Usage: Local, AliEn grid, CAF/PROOF**
- Transparent access to different inputs
  - **ESD, AOD, Kinematics tree (MC truth)**
- Allow for „scheduled“ analysis
  - **Common and well tested environment to run several tasks**
- Defines a common terminology

N.B.: The analysis framework itself has a very general design, not bound to ALICE software

The single task view

- AliAnalyisTask
  - User provided code
- Input data
  - Provided via numbered slots
  - Each slot connected to a data container of the corresponding type at run time
  - Content can be any TObject
  - “Handlers” handle data specific operations
- Output data
  - Communicated via one or more slots
  - Handlers e.g. for AOD output
  - Simpler output e.g. histograms
  - Output can be disk resident (file) or only memory resident (transient data)
- Several of these tasks can be collected in the manager

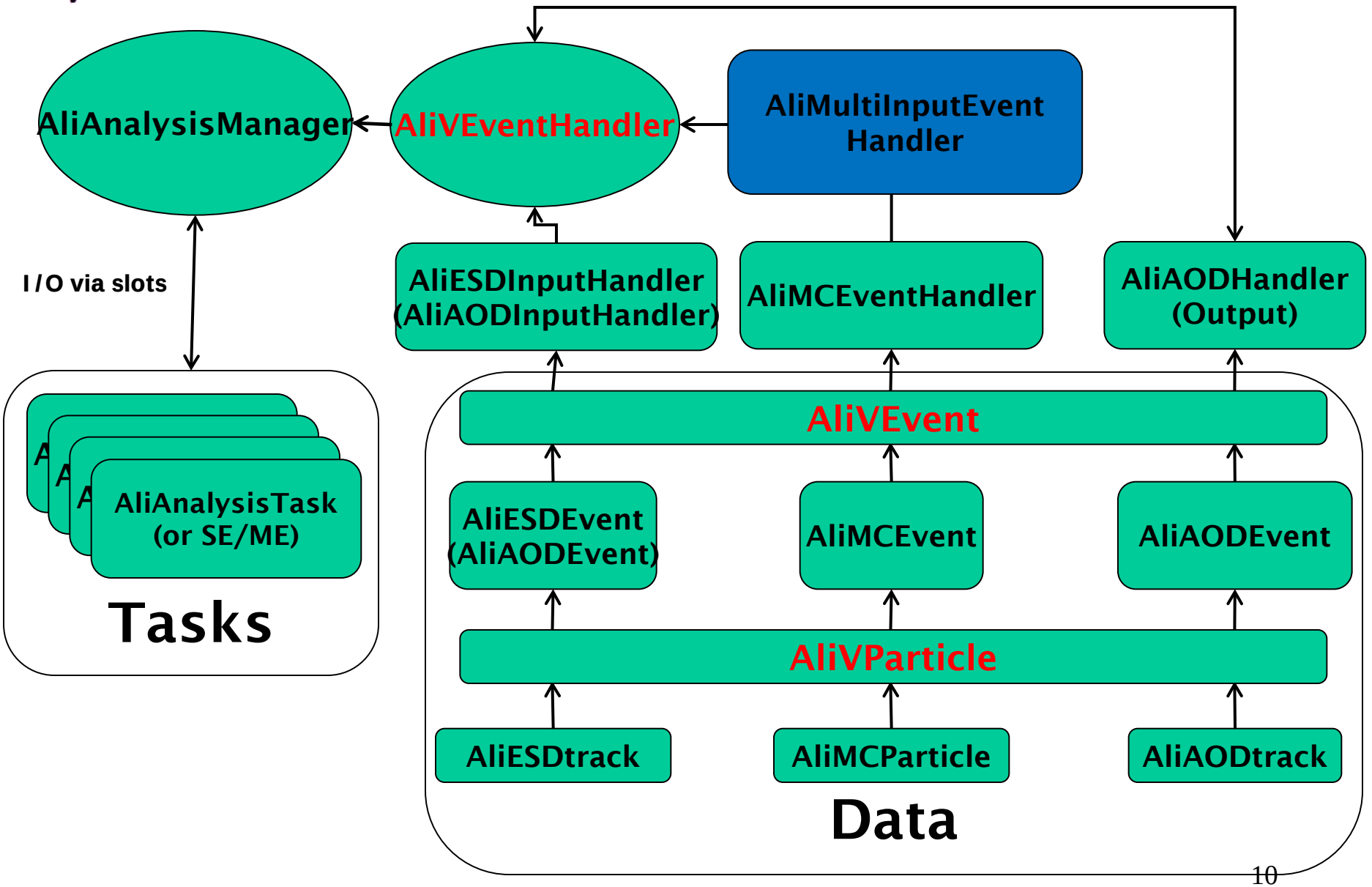


N:B.: AliAnalysisTask is a general Task
AliAnalysisTaskSE and ME are ALICE specific

Analysis manager

- Several analysis tasks registered to an analysis manager
- Top data container(s) storing the initial input data chain (ESD, AOD, kinematics tree, ...)
- Primary tasks feeding from top containers, executed in serial mode for each event
- Possibly secondary tasks feeding from data produced by parent tasks
  - e.g. filtered AOD tracks, jets etc.
- Handles initialization, execution and termination of all registered tasks

The overall picture



Library structure

- ANALYSIS classes split
  - ALICE independent: libANALYSIS.so
  - ALICE specific: libANALAYSISalice.so
  - Load both in your steering macro
- Additional layer of inheritance
  - YourTask:AliAnalyisTaskSE(:AliAnalysisTask)
  - Does some work for you specific for “Single Event” analysis

A new layer AliAnalyisTaskSE

- Already provides the access to input ESD/AOD/Kinematics and define AOD output

```
AliVEvent*    fInputEvent;   //! VEvent Input
AliAODEvent*  fOutputAOD;    //! AOD out
AliMCEvent*   fMCEvent;      //! MC
```

- Input event can be cast as needed

```
AliESDEvent *esd =
dynamic_cast<AliESDEvent*>fInputEvent;
```

- You need to implement at minimum:
  - virtual void UserCreateOutputObjects(){;}
  - virtual void UserExec(){;}
- Have a look at:
  - analysis-tutorial.tgz#analysis-tutorial/TaskSE/
- Mandatory to use to profit from centralized utilities (trigger-based event selection)

ANALYSIS FRAMEWORK … IN PRACTICE

Evolution of your analysis code

- In practice
  - Develop your analysis code as AliAnalysisTaskSE and test locally on a few files
  - Most debugging done here
- When the code works locally, submit to PROOF or AliEn Grid
  - PROOF
    - Fast response, fast turnaround
    - Limited number of files
  - AliEn Grid: Access to all files
- Add to the organized analysis (LEGO trains)
  - Several trains per PWG

Analysis: Component by Component

- What is needed
  - The manager: AliAnalysisManager
  - The input handler: ESD/AOD supported
  - The output handler: AliAODHandler
- Optional
  - MC Truth handler: AliMCEventHandler
- Your Task(s) AliAnalysisTask(SE)
- A small execution macro
  - Load libraries
  - Collect input files (TChain), connect everything to the manager
  - Run

AliAnalysisManager - basics

- AddTask(AliAnalysisTask *pTask)
  - At least 1 task per analysis (top task)
- CreateContainer(name, data_type, container_type, file_name)
  - Data can be optionally connected to a file
- ConnectInput/Output(pTask, islot, pContainer)
  - Mandatory for all data slots defined by used analysis modules
- InitAnalysis()
  - Performs a check for data type consistency and signal any illegal circular dependencies between modules
- StartAnalysis(const char *mode)
  - Starts the analysis in “local”, “proof” or “grid” mode

Analysis macro
Load libs and create manager

see analysis_tutorial.tgz#TaskSE/jetana.C

```
// Load libs (more needed when running with root instead of
   aliroot)
// Minimum need to load libANALYSISalice
  gSystem->Load("libANALYSISalice.so");

// Load the task
// AliAnalysisTaskJets is in libJETAN
   gSystem->Load("libJETAN.so");
// User tasks usually compiled on the fly at the beginning,
   not needed here
//  gROOT->LoadMacro(”AliAnalysisMyTaskXYZ.cxx+g”);

// Create a Chain of input files ESDs here
// External file list is used
   gROOT-
  >LoadMacro("$ALICE_ROOT/PWGUD/macros/CreateESDChain.C");
   TChain *chain = CreateESDChain("filelist.txt");
// or Manual chaining:
// TChain *chain = new TChain("esdTree");
// chain->Add(”SomePath/AliESDs.root");

// Create the Analysis manager
  AliAnalysisManager *mgr  =
    new AliAnalysisManager("My Manager", "My Analysis");
```

Analysis macro
Create Input/Output handler

```
// Define Input Event Handler
  AliESDInputHandler* esdHandler = new AliESDInputHandler();

// Define MC Truth Event Handler (optional)
  AliMCEventHandler* mcHandler = new AliMCEventHandler();

// Add input handlers to the Task Manager via a multi handler
  AliMultiInputEventHandler *mH = new
                                  AliMultiInputEventHandler();
  mH->AddInputEventHandler(esdHandler);
  mH->AddInputEventHandler(mcHandler);
  mgr->SetInputEventHandler  (mH);

// Define Output Event Handler (only if filtering specific
   information in form of a tree to be reprocessed later)
  AliAODHandler* aodHandler = new AliAODHandler();
    aodHandler->SetOutputFileName("aod.root");
  mgr->SetOutputEventHandler (aodHandler);


// Be sure you are told what you are doing
  mgr->SetDebugLevel(3);
```

AOD or Kinematics Analysis?

- Same schema works for AOD analysis
  - TChain contains AOD files
  - User retrieves AliAODEvent directly from the task (fInputEvent)
  - More efficient (smaller size)
- ... and even for Kinematics
  - Add galice.root files to TChain
  - This “triggers” correct loop over files
  - Obtain AliMCEvent from the task (fMCEvent) combining:
    - Kinematics tree
    - TreeE (Event Headers)
    - Track references

Analysis macro: Define Input/Output

```
// Declare Common Input TChain
  AliAnalysisDataContainer *cinput1 =
mgr->CreateContainer("Chain",TChain::Class(),
    AliAnalysisManager::kInputContainer);

// Common Output Tree in common ‘default’ output file
aod.root (ONLY NEEDED IF YOUR TASK WRITES AN AOD!)
  AliAnalysisDataContainer *coutput1 =
mgr->CreateContainer("tree", TTree::Class(),
    AliAnalysisManager::kOutputContainer, "default");

// Private output objects to write to a file
  AliAnalysisDataContainer *coutput2 =
mgr->CreateContainer("histos", TList::Class(),
    AliAnalysisManager::kOutputContainer, "histos.root");
```

AliAnalysisDataContainer

- Normally a class to be used ‘as is’
  - Enforcing a data type deriving from TObject
    - Type e.g. given by TChain::Class()
- Three types of data containers
  - Input – containing input data provided by AliAnalysisManager
  - Exchange – containing data transmitted between tasks
  - Output – containing final output data of an analysis chain, eventually written to files.
- One can specify the output file name in the format: file.root:folder

Analysis Macro
Add an Analysis Task and run

```
// Create Jet Finder Task task
  AliAnalysisTask *jetana = new
                AliAnalysisTaskJets("JetAnalysis");
  jetana->SetDebugLevel(10);

// Add task to the manager
  mgr->AddTask(jetana);

// Connect I/O to the task
  mgr->ConnectInput (jetana, 0, cinput1);
  mgr->ConnectOutput(jetana, 0, coutput1);
  mgr->ConnectOutput(jetana, 1, coutput2);

// Run the task
  mgr->InitAnalysis();
  mgr->PrintStatus();
  mgr->StartAnalysis("local",chain);
```

For jet analyis task see: $ALICE_ROOT/JETAN
AliAnalysisJets.{cxx,h}

Task to be added to a train

- Few extra things to be considered when running in a train
  - Write the following parts of the analysis macro to a separate file (named: AddTask(...).C
    - Creation of the task + possible configuration parameters
    - Creation of containers and connection of slots
      - Use: mgr->GetCommonFileName() as output file and append the outputs of your task to a specific folder

```
TString file = mgr->GetCommonFileName();
file += ":myFolder";
mgr->CreateContainer("histos", TList::Class(),
    AliAnalysisManager::kOutputContainer, file);
```

    - Adding the task to the manager
- Check for an example: $ALICE_ROOT/ANALYSIS/macros/AddTaskPIDResponse.C

Hands on: exercise

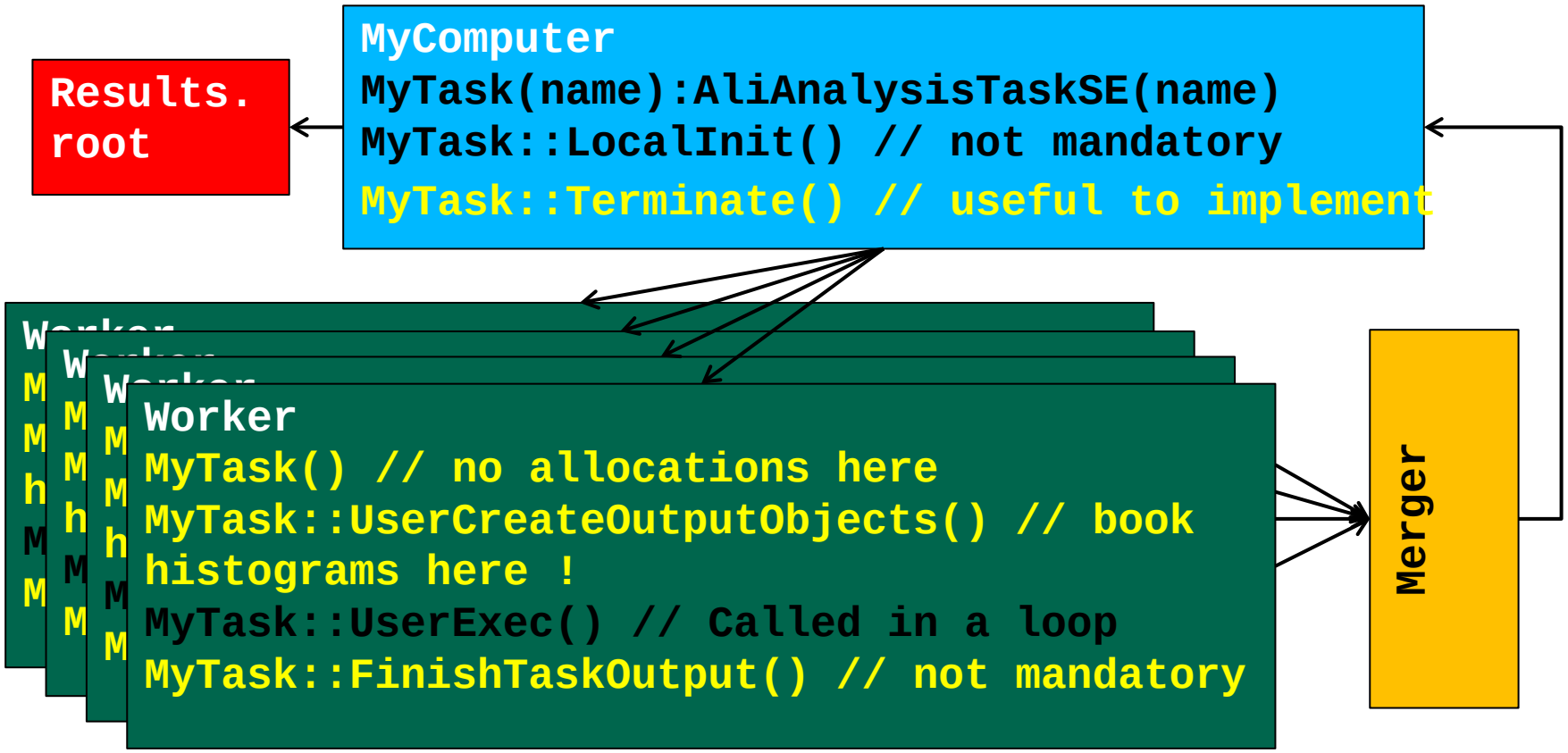
- Download/copy:
  - ...analysis-tutorial.tgz
- Unpack the file:
  - tar xvzf analysis_tutorial.tgz
- Look at TaskSE/jetana.C
  - Do you recognize everything?
- Edit jetana.C
  - Put a return; after the mgr->PrintStatus();
  - Run the macro
  - Look at the output on the screen
- Remove the return; and re-run adding 200 as last argument of StartAnalysis
  - Which files are created?
  - Have a look in a TBrowser()

Implement task: Method by Method

- We have a framework that calls an analysis task with inputs and outputs connected
- How do we implement our own analysis ask?
  - "Constructor" and "Destructor"
    - like any C++ class
  - UserCreateOutputObjects()
    - Create Histograms
  - UserExec()
    - The event loop
  - Terminate()
    - Called at the end, can draw (or fit) e.g. a histogram
- We cover here the case for AliAnalysisTaskSE
  - Recommended to use TaskSE (services provided)
  - Examples for AliAnalysisTask are in the tarball (Task/) for reference

AliAnalysisTaskSE

- Classes derived from AliAnalysisTaskSE can run locally, in PROOF and in AliEn



Named constructor

```
AliAnalysisTaskJets::AliAnalysisTaskJets(const char* name):
    AliAnalysisTaskSE(name),
    fConfigFile("ConfigJetAnalysis.C"),
    fNonStdBranch(""),
    fJetFinder(0x0),
    fHistos(0x0),
    fListOfHistos(0x0)
{
  DefineOutput(1, TList::Class()); // 0 slots assigned in parent class
}
```

Called in the macro via `new AliAnalysisTaskJets("JetAnalysis")`

```
AliAnalysisTaskSE::AliAnalysisTaskSE(const char* name):…
{
   DefineInput (0,TChain::Class());
   DefineOutput(0,  TTree::Class());
}
```

Default constructor:

```
AliAnalysisTaskJets::AliAnalysisTaskJets():
  AliAnalysisTaskSE(),
  fConfigFile("ConfigJetAnalysis.C"),
  fNonStdBranch(""),
  fJetFinder(0x0),
  fHistos(0x0),
  fListOfHistos(0x0)
{
  // Default constructor
}
```

N.B.: No DefineInput/DefineOutput in default c'tor (important for PROOF case)

UserCreateOutputObjects()

```
// Open Histograms
  OpenFile(1);

  …
  fHisto = new TH1F(“fHisto”, “My Histo”, 100, 0., 10.);
  …….
 // Several histograms are more conveniently managed in a TList
  fListOfHistos = new TList();
  fListOfHistos->Add(fHisto);
```

UserExec()

```
void AliAnalysisTaskJets::UserExec(Option_t */*option*/)
{
  // Execute analysis for current event
  //


  // Jet finding is delegated access to input outpout and
  // MC given by TaskSE
  fJetFinder->GetReader()->SetInputEvent(InputEvent(), AODEvent(),
MCEvent());
  fJetFinder->ProcessEvent();
  ….
  fHisto->Fill(pt);
  …
  // Post the data (it will be written automatically)
  PostData(1, fListOfHistos);
}
```

Called for each event

UserExec()

- virtual void UserExec(Option_t *option)
  - Mandatory to implement in the derived class
  - This actually implements how the analysis module processes the current event from input data
  - End with PostData(slot, data) – will notify all tasks depending on the output that data is ready

AliAnalysisTaskJets

- This task is already quite elaborated
  - Fills an AOD output
  - Uses a TList to manage histograms
  - Passes data to a AliJetFinder configured by an external macro
- Let‘s turn to something simpler...

Analysis framework: hands on

- Trivial example: plot the p_t of the ESD particles
  - Files in the analysis_tutorial.tgz archive (TaskSE)

AliAnalysisTaskPt.cxx
AliAnalysisTaskPt.h
esd.txt (aod.txt)
run1.C

Analysis framework: hands on

Have a look at run1.C

```
void run1()
 // load analysis framework
 gSystem->Load("libANALYSIS");
 gSystem->Load("libANALYSISalice");
 gROOT->LoadMacro("$ALICE_ROOT/PWG0/CreateESDChain.C");
 TChain* chain = CreateChain(“esdTree”,“esd.txt", 2);

 // Create the analysis manager
 AliAnalysisManager *mgr = new AliAnalysisManager("testAnalysis");
 AliVEventHandler* handler = new AliESDInputHandler();
 mgr->SetInputEventHandler(handler);
 // Create task
 gROOT->LoadMacro("AliAnalysisTaskPt.cxx+g");
 AliAnalysisTask *task = new AliAnalysisTaskPt("TaskPt");
 // Add task
 mgr->AddTask(task);
 // Create containers for input/output
 AliAnalysisDataContainer *cinput = mgr->CreateContainer("cchain", TChain::Class(),
                                AliAnalysisManager::kInputContainer);
 AliAnalysisDataContainer *coutput = mgr->CreateContainer("chist", TH1::Class(),
                    AliAnalysisManager::kOutputContainer,"Pt.ESD.1.root");
 // Connect input/output
 mgr->ConnectInput(task, 0, cinput);
 mgr->ConnectOutput(task, 0, coutput);
 // Enable debug printouts
 mgr->SetDebugLevel(2);

 if (!mgr->InitAnalysis())
   return;
 mgr->PrintStatus();
 mgr->StartAnalysis("local", chain);
}
```

Analysis framework: hands on

Have a look at AliAnalysisTaskPt.h

```
#ifndef AliAnalysisTaskPt_cxx
#define AliAnalysisTaskPt_cxx
class TH1F;

#include "AliAnalysisTaskSE.h"

class AliAnalysisTaskPt : public AliAnalysisTaskSE {
 public:
  AliAnalysisTaskPt(const char *name = "");
  virtual ~AliAnalysisTaskPt() {}


  virtual void   UserCreateOutputObjects();
  virtual void   UserExec(Option_t *option);
  virtual void   Terminate(Option_t *);

 private:
  TH1F        *fHistPt; //Pt spectrum
  ClassDef(AliAnalysisTaskPt, 1); // example of analysis
};
#endif
```

Analysis framework: hands on

```
//_____________________________________________________________________
AliAnalysisTaskPt::AliAnalysisTaskPt(const char *name)
  : AliAnalysisTaskSE(name), fHistPt(0)
{
  // Constructor
  // Define input and output slots here
  // Slot #0 works are defined in TaskSE
  // Output slot #1 writes into a TH1 container
  DefineOutput(1, TH1F::Class());
}

//_____________________________________________________________________
void AliAnalysisTaskPt::UserCreateOutputObjects()
{
  // Create histograms
  // Called once

  fHistPt = new TH1F("fHistPt", "P_{T} distribution", 15, 0.1, 3.1);
  fHistPt->GetXaxis()->SetTitle("P_{T} (GeV/c)");
  fHistPt->GetYaxis()->SetTitle("dN/dP_{T} (c/GeV)");
  fHistPt->SetMarkerStyle(kFullCircle);
}
```

Have a look at AliAnalysisTaskPt.cxx

Only in the constructor with the signature (const char *)

Analysis framework: hands on

```
void AliAnalysisTaskPt::UserExec(Option_t *)
{
  // Main loop
  // Called for each event
  if (!fInputEvent) {
     Printf("ERROR: Could not retrieve event");
     return;
  }

  if(Entry()==0){
    AliESDEvent* esd = dynamic_cast<AliESDEvent*>(event);
    AliAODEvent* aod = dynamic_cast<AliAODEvent*>(event);
    if(esd){
      Printf("We are reading from ESD");
    }
    else if(aod){
      Printf("We are reading from AOD");
    }
  }

  Printf("There are %d tracks in this event", fInputEvent->GetNumberOfTracks());
  // Track loop to fill a pT spectrum
  for (Int_t iTrack = 0; iTrack < fInputEvent->GetNumberOfTracks(); iTrack++) {
    AliVParticle *track = fInputEvent->GetTrack(iTrack);
    if (!track) {
      Printf("ERROR: Could not receive track %d", iTrack);
      continue;
    }
    fHistPt->Fill(track->Pt());
  } //track loop

  // Post output data.
  PostData(1, fHistPt);
}
```

Works for AOD and ESD Input

Side Remark: MC truth

```
void AliAnalysisTaskXYZ::UserExec(Option_t* option )
{

  // During Analysis
  AliVEvent* mc = MCEvent();
  Int_t ntrack = mc->GetNumberOfTracks();
  for (Int_t i = 0; i < ntrack; i++)
  {
    AliVParticle* particle = mc->GetTrack(i);
    Double_t pt = particle->Pt();
  }

}
```

Can also read only Kinematics (no need for ESDs), without ESDs change one line in steering macro:

chain = CreateChain(“TE”,”galice_root_list”,2);

Analysis framework: hands on

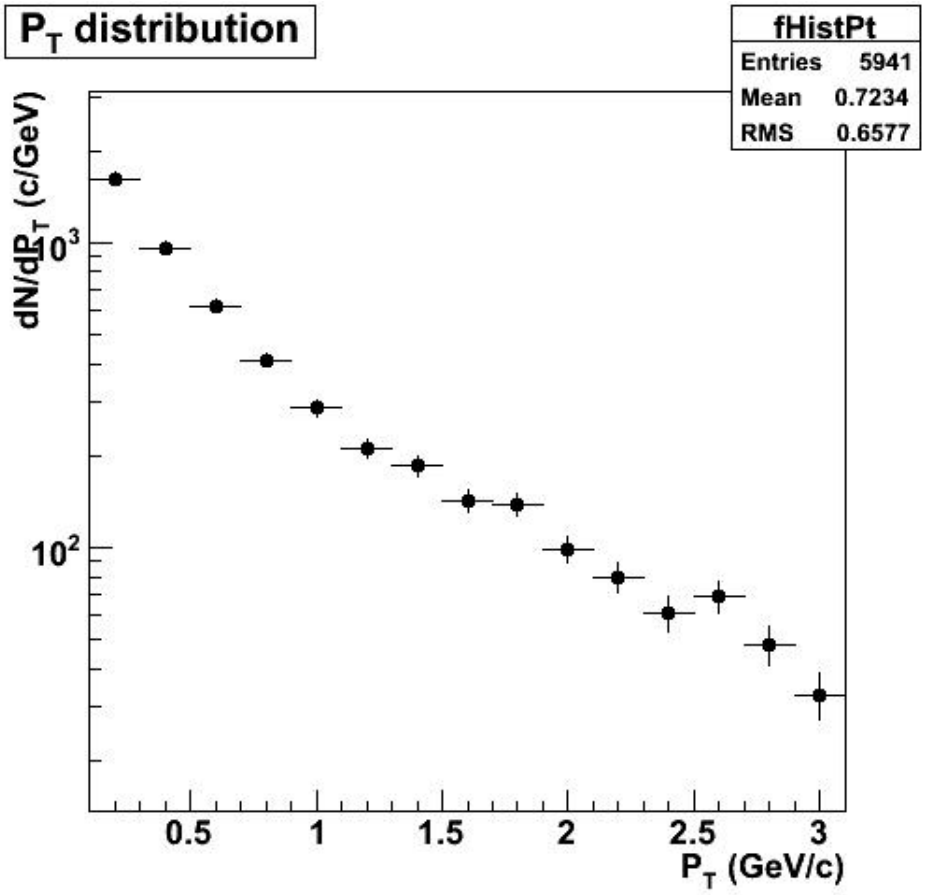
```
void AliAnalysisTaskPt::Terminate(Option_t *)
{
  // Draw result to the screen
  // Called once at the end of the query

  fHistPt = dynamic_cast<TH1F*>(GetOutputData(1));
  if (!fHistPt) {
    Printf("ERROR: fHistPt not available");
    return;
  }

  TCanvas *c1 = new
TCanvas("AliAnalysisTaskPt","Pt",10,10,510,510);
  c1->cd(1)->SetLogy();
  fHistPt->DrawCopy("E");
}
```

Run the macro…

```
root[0].x run1.C
Processing run1.C...
task: TaskPt  ACTIVE=0 POST_LOOP=0
   INPUT #0: TChain <-  [cchain]
   OUTPUT #0: TH1F ->  [chist]
   Container: chist  type: TH1 POST_LOOP=0
 = Data producer: task TaskPt    = Consumer tasks: -none-
Filename: Pt.ESD.1.root
StartAnalysis: testAnalysis
===== RUNNING LOCAL ANALYSIS testAnalysis ON TREE esdTree
->AliAnalysisSelector->Init: Analysis manager restored
->AliAnalysisSelector->SlaveBegin() after Restore
->AliAnalysisManager::SlaveBegin()
->AliAnalysisManager::Init(esdTree)
<-AliAnalysisManager::Init(esdTree)
<-AliAnalysisManager::SlaveBegin()
<-AliAnalysisSelector->SlaveBegin()
AliAnalysisManager::Notify() file: AliESDs.root
->AliAnalysisSelector::Process()
== AliAnalysisManager::GetEntry()
AliAnalysisManager::ExecAnalysis
 Executing task TaskPt
…
<-AliAnalysisSelector::Process()
->AliAnalysisSelector::SlaveTerminate()
->AliAnalysisManager::PackOutput()
<-AliAnalysisManager::PackOutput: output list contains 0 containers
<-AliAnalysisSelector::SlaveTerminate()
->AliAnalysisSelector::Terminate()
->AliAnalysisManager::UnpackOutput()
   Source list contains 0 containers
<-AliAnalysisManager::UnpackOutput()
->AliAnalysisManager::Terminate()
<-AliAnalysisManager::Terminate()
<-AliAnalysisSelector::Terminate()
```



Hands on: exercises

- Try to run with root alone instead of aliroot
  - Note the libraries needed
- Try to run on input AOD data
  - The chain name is “aodTree” and the file with links to AOD’s is “aod.txt”
- Replace AliAnalysisTaskPt by AliAnalysisTaskPtMC in run1.C
  - What happens?
  - Why?
  - How to fix it?
  - Try to run AliAnalysisTaskPt and AliAnalysisTaskPtMC together
- Add a new data member fEta to AliAnalysisTaskPt
  - Fill the histogram using track->Eta() and plot it in Terminate
- Check handson/run4.C on how to get physics selection and centrality services

Some extras….

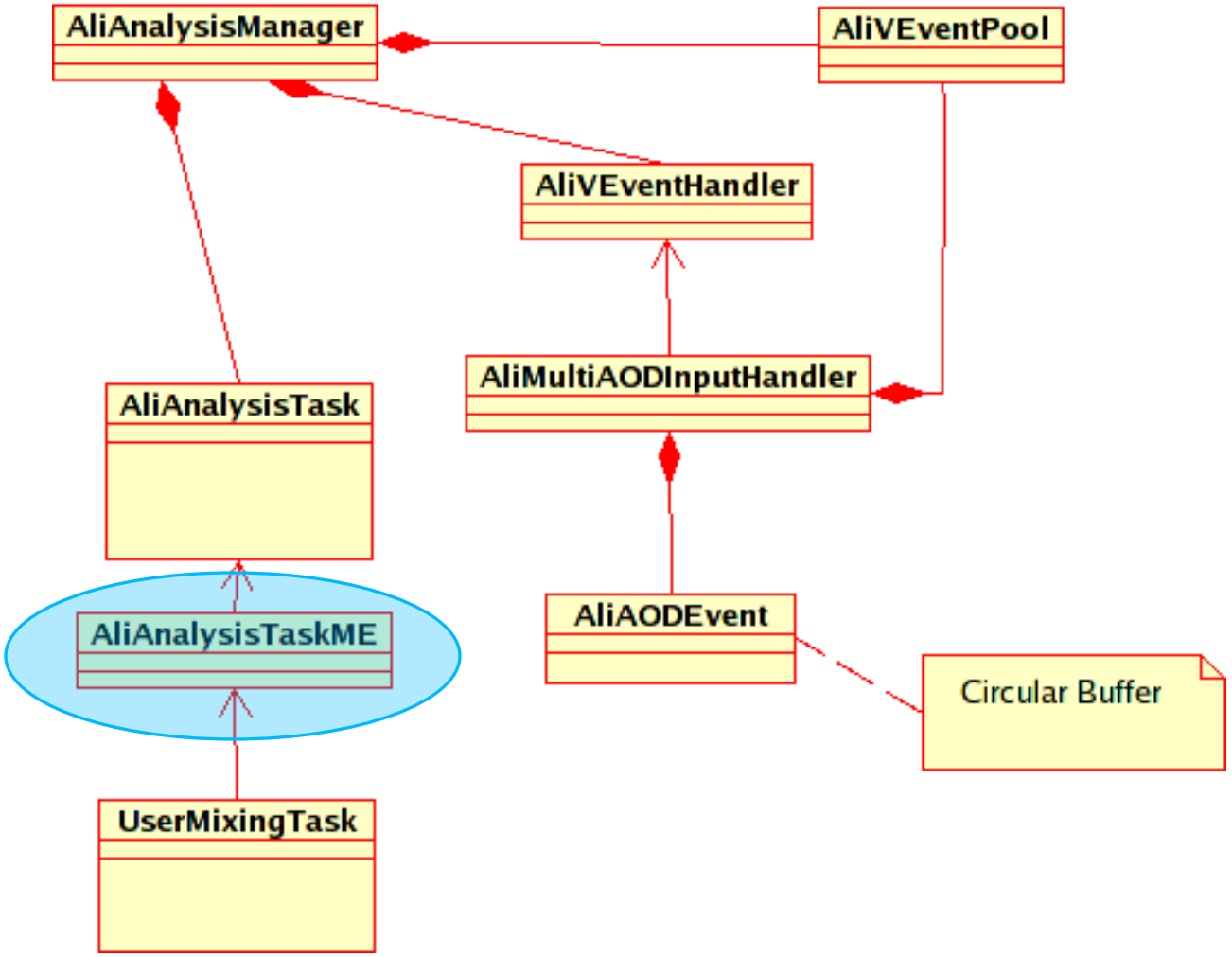
Mixed Events and the Correction Framework

Mixed Events

- Implemented since v4-13-Release
- Needed for any analysis that suffers from combinatorial background
  - e.g. photon pair combinations for π^0 -> $\gamma\gamma$ analysis
  - Inherit from `AliAnalysisTaskME`
  - Provides access to a pool of events which are “close” (e.g. in multiplicity) to the current event (tags needed for selction)
  - Pool stored independent of user requirements only once.

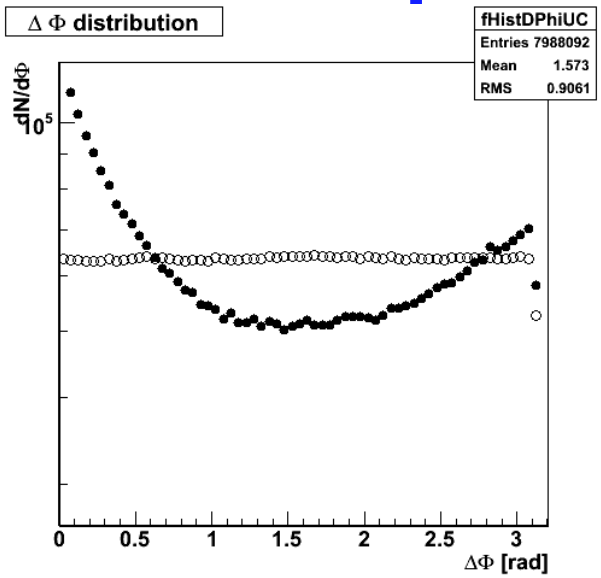
Mixed Events

Analysis With Event Mixing



Example Mixed Events

- $ALICE_ROOT/ANALYSIS/ (trunk)
  - AliAnalysisTaskPhiCorr.{cxx,h}
    - DphiAnalysis.C
    - Data from /afs/cern.ch/user/m/morsch/public/



Correction framework

- In general efficiency is:
  - Output(x_1,x_2,x_3...)/Input(x_1,x_2,x_3...)
  - (x_1,x_2,x_3...) e.g. (p_T,η,z,...)
- $ALICE_ROOT/CORRFW provides
  - Container classes
    - n-dim histograms which store distributions with
      - MC input
      - after acceptance cut (with track references)
      - reconstrcuted tracks w/wo cuts
  - Cut classes
- Basic example:
  - analysis-tutorial.tgz#analysis-tutorial/TaskSE/

Correction Framework Crash Course

Create the Cuts (see run6.C)

```
// generator level kinematic cuts
// these cuts shall select the particles of interest.
// Their efficiency shall be studied lateron.
// Here we will calculate the efficiency of charged tracks around midrapidity.
AliCFTrackKineCuts *kineCutsMC = new AliCFTrackKineCuts("kineCutsMC","kinematic cuts MC");
kineCutsMC->SetQAOn(kTRUE);
kineCutsMC->SetEtaRange(-1.,1.);
kineCutsMC->SetRequireIsCharged(kTRUE);

// cuts on reconstructed tracks
// apply the same cuts as for MC particles
// add more cuts if desired
AliCFTrackKineCuts *kineCutsRec = new AliCFTrackKineCuts("kineCutsRec","kinematic cuts rec");
kineCutsRec->SetQAOn(kTRUE);
kineCutsRec->SetEtaRange(-1.,1.);
kineCutsRec->SetRequireIsCharged(kTRUE);
kineCutsRec->SetPhiRange(0.,5.);
// apply also other kind of cuts
AliCFTrackQualityCuts *qualityCuts = new AliCFTrackQualityCuts("qualityCuts"," quality cuts");
qualityCuts->SetQAOn(kTRUE);
qualityCuts->SetMinNClusterTPC(50);
'/   qualityCuts->SetRequireTPCRefit(kTRUE);
```

Correction Framework Crash Course

Create the Containers and create the Task (see run6.C)

```
  // create the container for the efficiency calculation
  // configure it:
  // set number sensitive variables: eff = eff(pt,eta)
  const Int_t nvar = 2;
  // set binning: 6 bins in pt, 4 bins in eta
  Int_t nbin[nvar] = {6,4};
  // set number of steps: here container is filled twice
  // (1) with MC information
  // (2) with reconstructed tracks after cuts were applied
  Int_t nstep = 2;
  // set bin limits
  Double_t limitsPt[7]  = {0.,0.5,1.,1.5,2.,2.5,3.};
  Double_t limitsEta[5] = {-1.,-0.5,0.,0.5,1.};
  // create container
  AliCFContainer *aliCFContainer = new AliCFContainer("aliCFContainer","container for efficiency calculation",n
:ep,nvar,nbin);
  aliCFContainer -> SetBinLimits(0, limitsPt);
  aliCFContainer -> SetBinLimits(1, limitsEta);



  // Create task
  gROOT->LoadMacro("AliAnalysisTaskPtCF.cxx+g");
//   AliAnalysisTask *task = new AliAnalysisTaskPtCF("TaskPtCF");
  AliAnalysisTaskPtCF *task = new AliAnalysisTaskPtCF("TaskPtCF");
  // pass the correction framework objects to the task
  task->SetKineCutsMC(kineCutsMC);
  task->SetKineCutsRec(kineCutsRec);
  task->SetQualityCuts(qualityCuts);
  task->SetContainer(aliCFContainer);
  task->SetQAList(qaList);
```

Correction Framework Crash Course

Fill MC information in AliAnaylisTaskCF::UserExec() after kinematical cuts (see AliAnaylisTaskCF.cxx)

```
// fill QA histograms before and after the cut is applied
fKineCutsMC->FillHistograms(track,0);
if(!fKineCutsMC->IsSelected(track)) continue;
fKineCutsMC->FillHistograms(track,1);

// fill container, first step is MC info: Fill(...,0)
Double_t containerInput[2] ;
containerInput[0] = track->Pt();
containerInput[1] = track->Eta() ;
fAliCFContainer->Fill(containerInput,0);
```

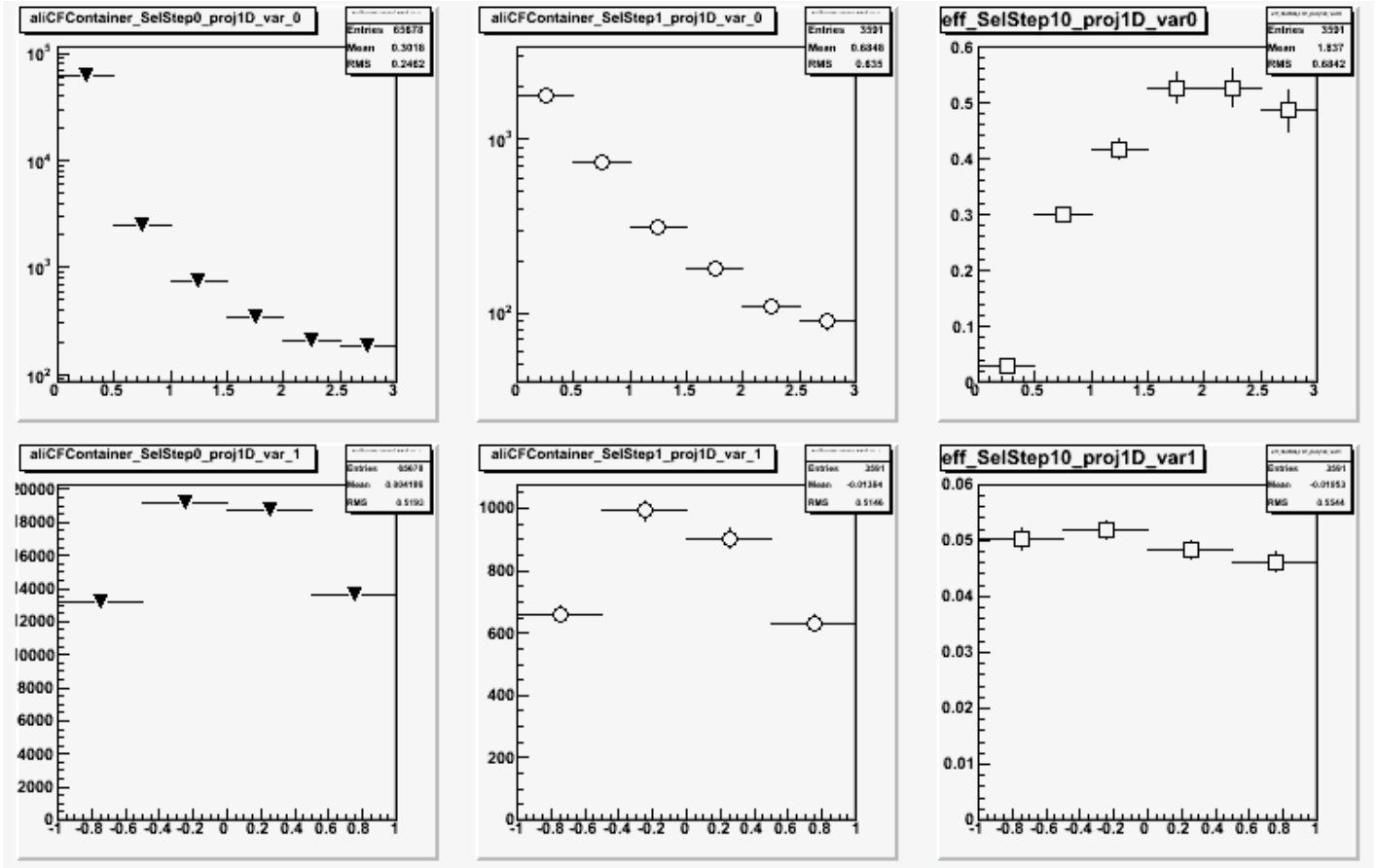
Fill reconstructed information after QA cuts

```
// fill QA histograms before and after the cut is applied
fKineCutsRec->FillHistograms(track,0);
if(!fKineCutsRec->IsSelected(track)) continue;
fKineCutsRec->FillHistograms(track,1);
fQualityCuts->FillHistograms(track,0);
if(!fQualityCuts->IsSelected(track)) continue;
fQualityCuts->FillHistograms(track,1);

// fill container, second step is after reconstruction and cuts: Fill(...,1)
Double_t containerInput[2] ;
containerInput[0] = track->Pt();
containerInput[1] = track->Eta() ;
fAliCFContainer->Fill(containerInput,1);
```

Try it Out

- run TaskCF/run6.C and TaskCF/CalcEff_run6.C

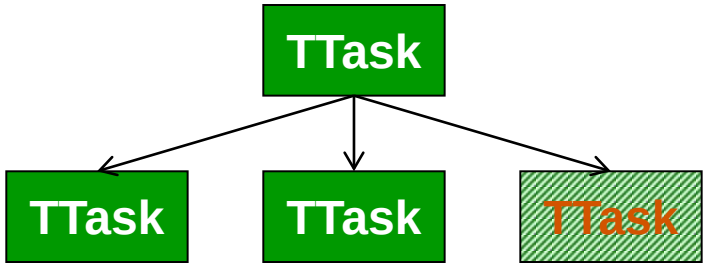


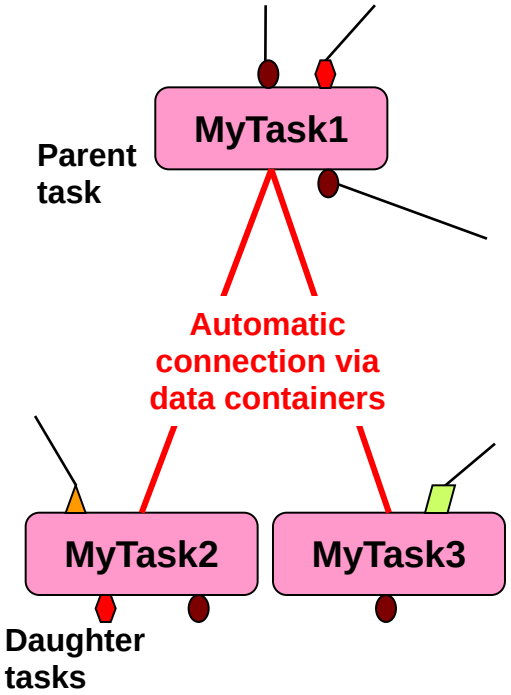
To be continued…

Analysis Framework …technicalities (Backup)

Advanced Structure

- **Analysis has to be split in functional modules**
  - At least one
  - Deriving from TTask
  - Parent task running active daughters
- **Modules are not manually inter-connected**
  - Connected just to input/output data containers
  - A data container has one provider and possibly several clients
  - A module becomes active when all input data is ready





AliAnalysisDataContainer

- **Normally a class to be used ‘as is’**
  - **Enforcing a data type deriving from TObject**
  - **For non-TObject (e.g. basic) types one can subclass and append the needed types as data members**
- **Three types of data containers**
  - **Input – containing input data provided by AliAnalysisManager**
  - **Transient – containing data transmitted between modules**
  - **Output – containing final output data of an analysis chain, eventually written to files.**
- **One can set a file name if the content is to be written**

AliAnalysisTask::ConnectInputData()

```
// Get the input handler from the manager
  AliESDInputHandler* esdH = (AliESDInputHandler*)
    ((AliAnalysisManager::GetAnalysisManager()
      ->GetInputEventHandler());

// Get pointer to esd event from input handler
  AliESDEvent* fESD = esdH->GetEvent();
```

- EsdFilter
- default constructor
  - copy from afs area
  - check corrfw
  - clean up some slides
  - which aliroot version?
  - refer to analysis train example

References

- This tutorial:
  - https://aliceinfo.cern.ch/Offline/AliRoot/Manual.html
- Analysis web pages:
  - http://aliceinfo.cern.ch/Offline/Activities/Analysis/
- Analysis framework:
  - http://aliceinfo.cern.ch/Offline/Activities/Analysis/AnalysisFramework/index.html
- Analysis train:
  - http://aliceinfo.cern.ch/Offline/Activities/Analysis/AnalysisFramework/index.html#train
- News and known problems RSS:
  - http://aliceinfo.cern.ch/Offline/Activities/Analysis/NewsAndProblems.html
- Analysis task force mailing list:
  - alice-project-analysis-task-force@cern.ch