Status of the Analysis Framework

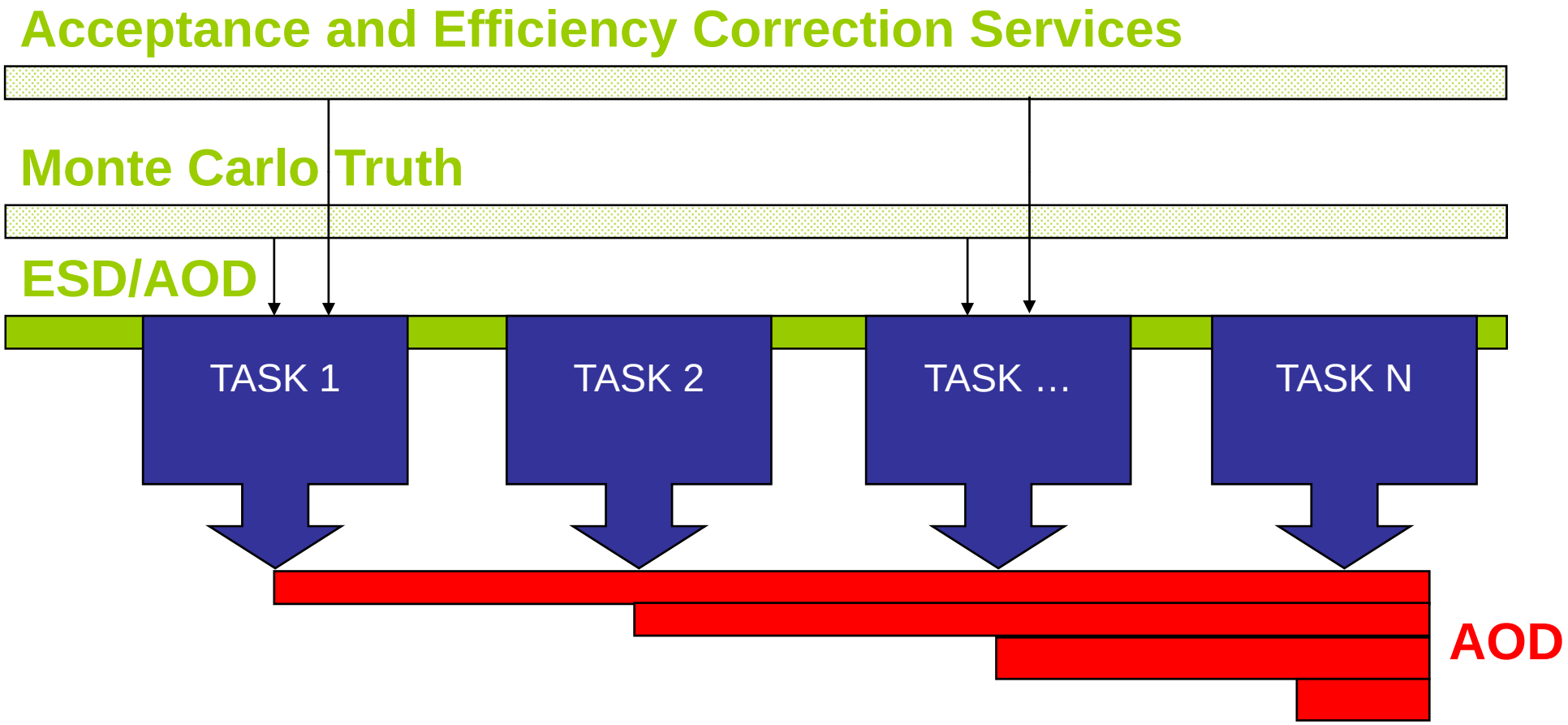
Andreas Morsch

ALICE Offline Week

10/10/2007

Why Organized Analysis ?

- 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
- But also
  - Helps to develop a common well tested framework for analysis.
  - Develops common knowledge base and terminology.
  - Helps documenting the analysis procedure and makes results reproducible.


Acceptance and Efficiency Correction Services
Monte Carlo Truth
ESD/AOD
TASK 1
TASK 2
TASK …
TASK N
AOD

Design Goals

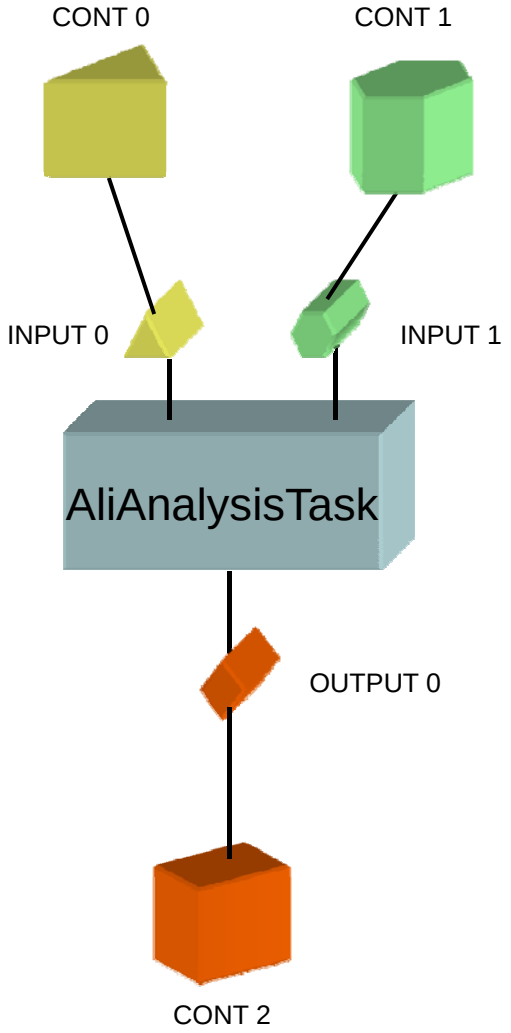
- Flexible task and data container structure
- User code independent of computing schema (interactive: local, PROOF or batch: GRID)
- Input data: ESD, AOD, MC Truth
  - Access using common interface
- Output data:
  - AOD
  - But also user histograms, containers for efficiency calculations
  - Transparent handling of memory resident and file resident data in distributed environment

Implementation

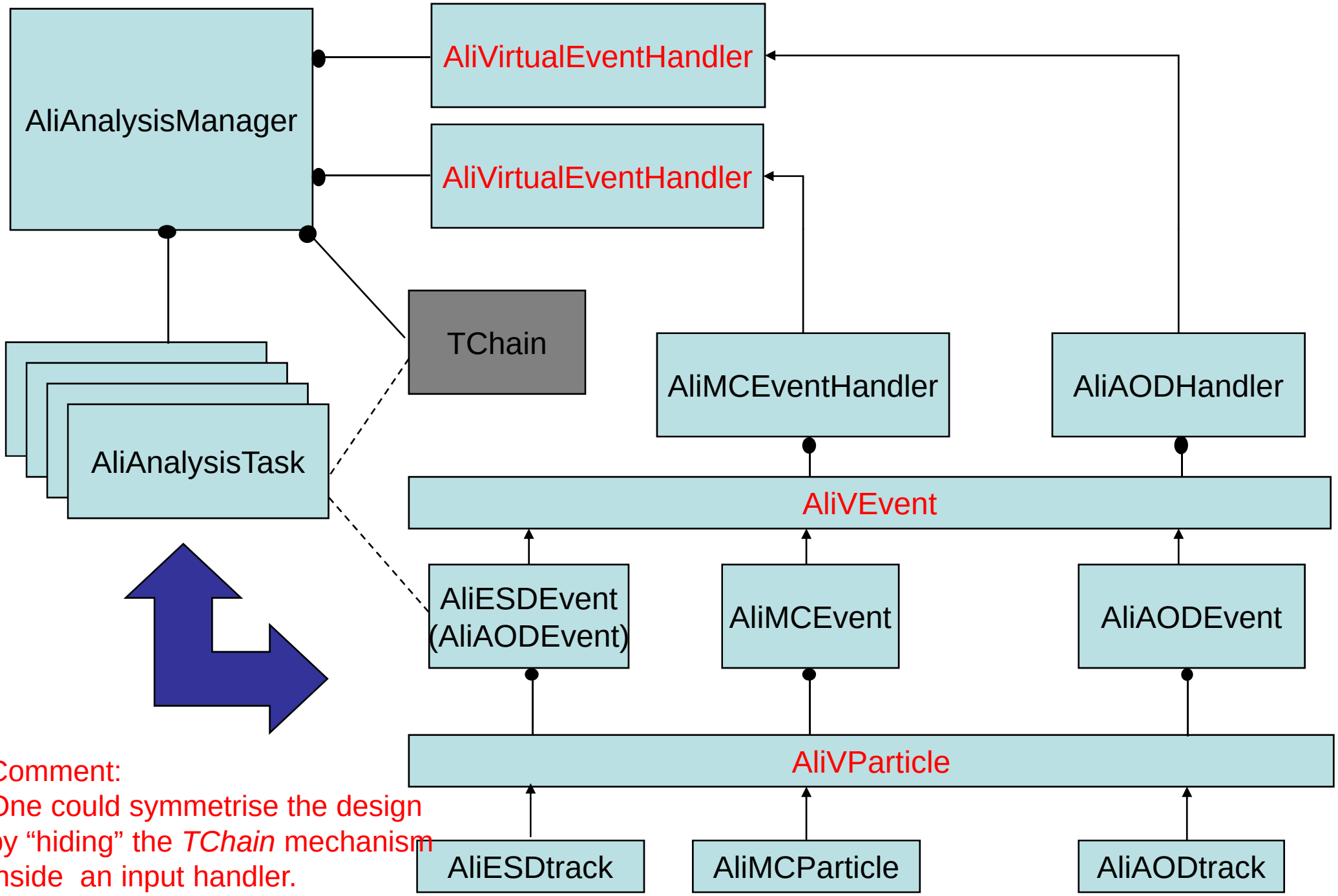
- Analysis train/taxi similar to PHENIX
- Based on the existing *AliAnalysisManager/Task* framework (A. and M. Gheata)
- *AliVEventHandler* interface for transparent optional additional event loop managements
- *AliVEvent*, *AliVParticle*, … for transparent data access

AliAnalysis… Framework

- Data-oriented model composed of independent tasks
  - Task execution triggered by data readiness
- Parallel execution and event loop done via *TSelector* functionality
  - Mandatory for usage with PROOF
- Analysis execution performed on event-by-event basis.
  - Optional post event loop execution.



AliAnalysis + Optional Data Services



Comment:
One could symmetrise the design by “hiding” the *TChain* mechanism inside an input handler.

Common ESD Access Handling

```
void AliAnalysisTaskXYZ::ConnectInputData(Option_t* option)
 {
    // Connect the input data
   fChain = (TChain*) GetInputData(0);
   fESD = new AliESDEvent();
   fESD->ReadFromTree(fChain);
       ..
       ...
}

void AliAnalysisTaskXYZ::Exec(Option_t* option)
 {
   // For data produced without AliESDEvent
   AliESD* old = fESD->GetAliESDOld();
   if (old) fESD->CopyFromOldESD();
 }
```

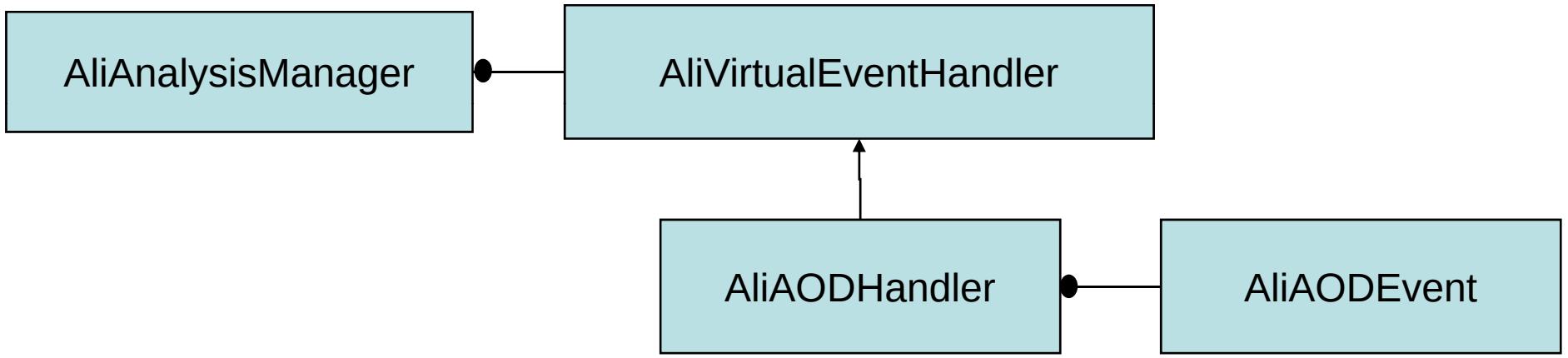
Attention: - PDC06 Data (v4-04) needs specially patched ESD.par or libESD.so
- FMD branch has to be switched off

What about AOD or Kinematics Analysis ?

- Same schema works for AOD analysis
  - *TChain* contains AOD files
  - User connects *AliAODEvent* to chain
- … and even for Kinematics
  - Add galice.root files to *TChain*
  - This “triggers” correct loop over files
  - Obtain *AliMCEvent* from the manager as usual.

(Ch. Klein-Bösing)

Common AOD Access Handling



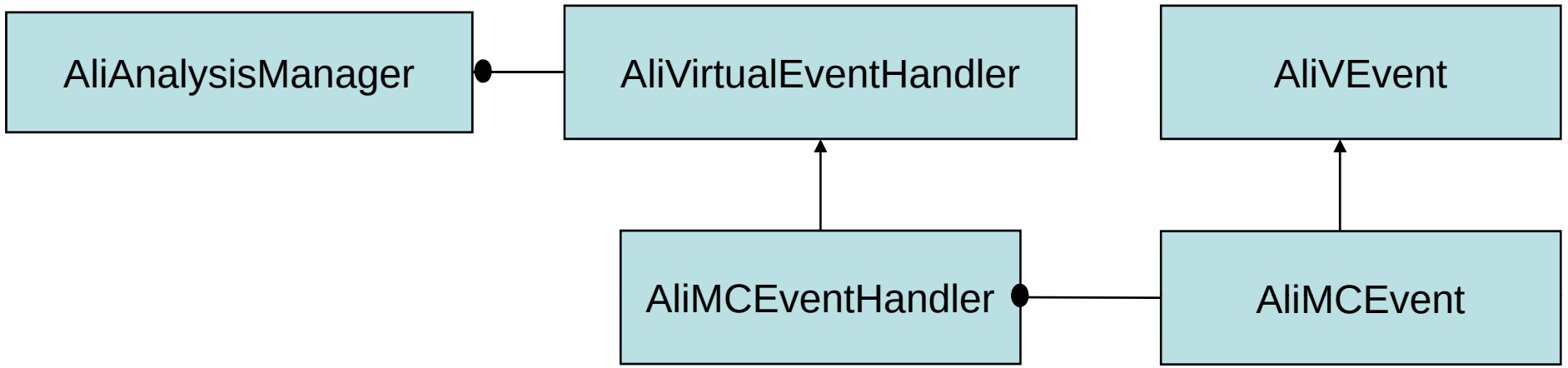

```
AliAODHandler* aodHandler = new AliAODHandler();
 aodHandler->SetOutputFileName("aod.root");

 AliAnalysisManager *mgr
    = new AliAnalysisManager('Analysis Train', 'Test');
 mgr->SetEventHandler(aodHandler);

 AliAnalysisDataContainer *coutput1 = mgr->CreateContainer('AODTree',
    TTree::Class(),
    AliAnalysisManager::kOutputContainer, "default");
```

User Analysis Code: Output Data

```
void AliAnalysisTaskXYZ::CreateOutputObjects()
{
// Create the output container
//
//  Default AOD
    AliAODHandler* handler = (AliAODHandler*)
((AliAnalysisManager::GetAnalysisManager())-
>GetEventHandler());
    fAOD   = handler->GetAOD();
    fTreeA = handler->GetTree();
    fJetFinder->ConnectAOD(fAOD);
}
```

Common Kinematics Input

- Before via class *AliAnalysisTaskRL*
  - Many dependences outside analysis
  - Requires implementation of specific MC analysis tasks.
- Now
  - Transparent usage of MC information via *AliMCEvent* combining
    - Kinematics Tree
    - TreeE (Event Headers)
    - Track References




```
AliMCEventHandler* mcHandler = new AliMCEventHandler();

AliAnalysisManager *mgr
    = new AliAnalysisManager('Analysis Train', 'Test');
 mgr->SetMCtruthEventHandler(mcHandler);
```

User Analysis Code: MC truth

```
void AliAnalysisTaskXYZ::Exec(Option_t* option )
{
// During Analysis
AliMCEvent* mc = mgr->GetMCEventHandler()->MCEvent();
Int_t ntrack = mc->GetNumberOfTracks();
for (Int_t i = 0; i < ntrack; i++)
{
      AliVParticle* particle = mc->GetTrack(i);
      Double_t pt = particle->Pt();
}
}
```

Some words on TrackReferences

- *TParticle* written in TreeK carries only limited information about the transport MC truth.
  - Only properties at production point
  - *Some MC truth is not stored*
- On the other hand *AliHit* contains MC truth but also depends on detector acceptance and response
  - *Some MC truth is lost*
- Solution *AliTrackReference* in tree TreeTR
  - Particle information at user defined reference plane crossings
  - Used in ITS, TPC, TRD, TOF, MUON, FRAME

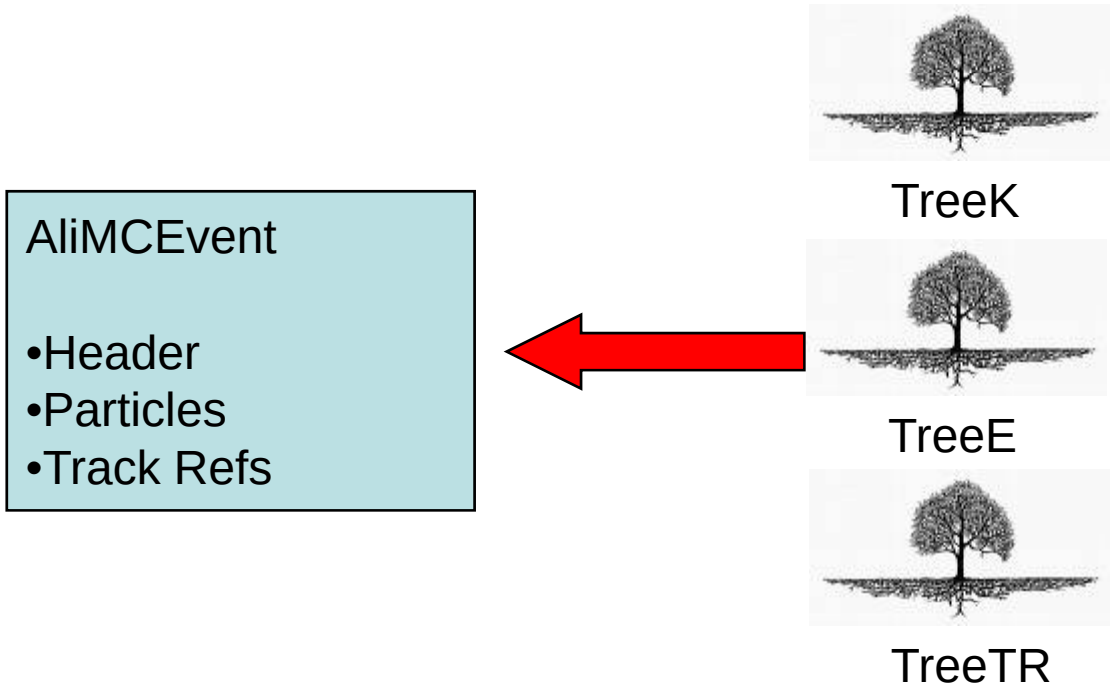
To be discussed:
Are the present Track References useful for efficiency and acceptance studies ??

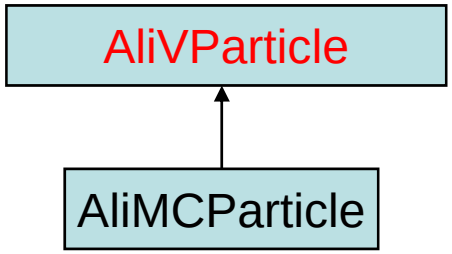
Problems with the previous implementation

- Track reference information spread over branches
  - One branch per detector
- TreeTR has structure different from TreeK
  - TreeK: one entry per particle
    - Primaries and secondaries
  - TreeTR: one entry per primary
- Information about one particle has to be collected from several branches.

Present Implementation

- One branch for track references instead of several.
  - Detectors identified by new data member fDetectorId.
- Synchronize TreeK and TR
  - Reorder the tree in a post-processor after simulation of each event
  - Executed automatically on the flight when old data is read

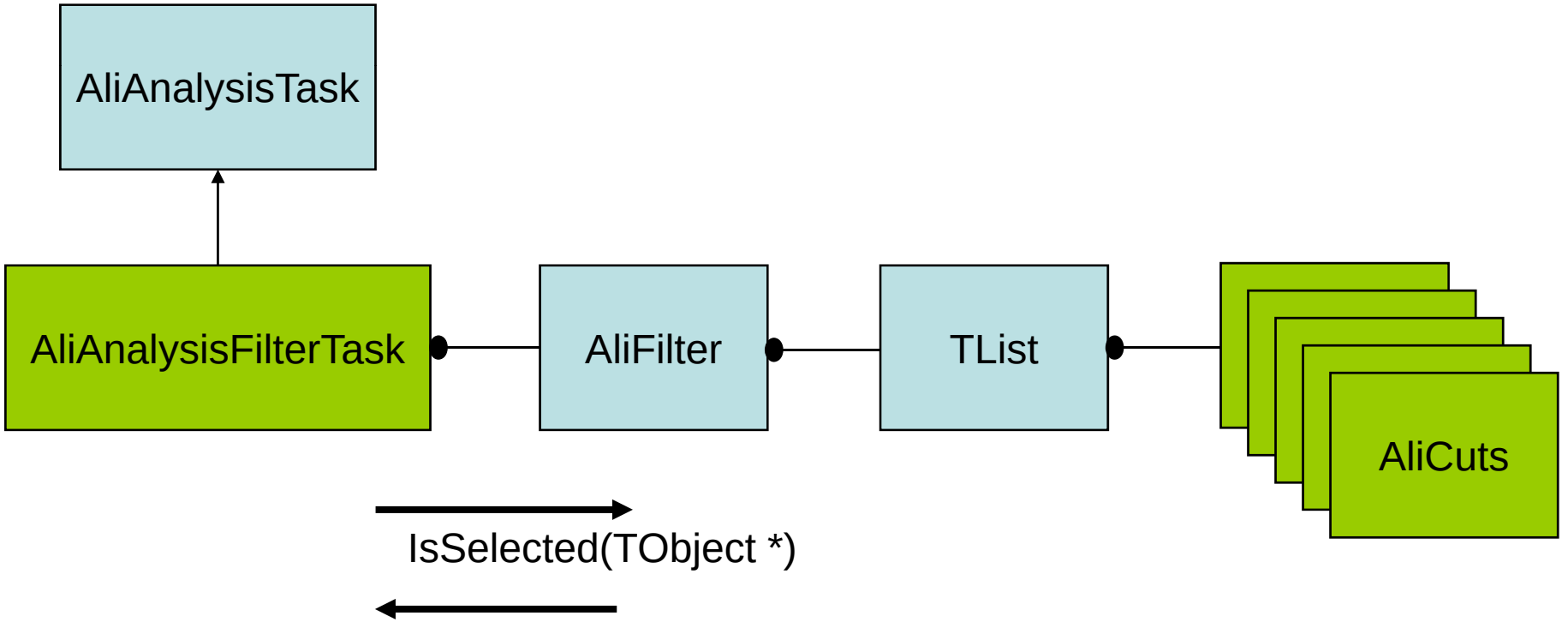




AliMCParticle

- Wraps *TParticle*
- Should also provide the TrackReference and vertex information
  - What is here the commonality with *AliESDtrack* and *AliAODtrack* ?
- Technical problem:
  - *AliMCParticle* is created on the flight and has to be buffered in *AliMCEvent*
  - No problem for *TParticle* part (*AliStack* already contains the mechanism), but what about *AliTrackReference* which is stored per particle inside a *TClonesArray* ?

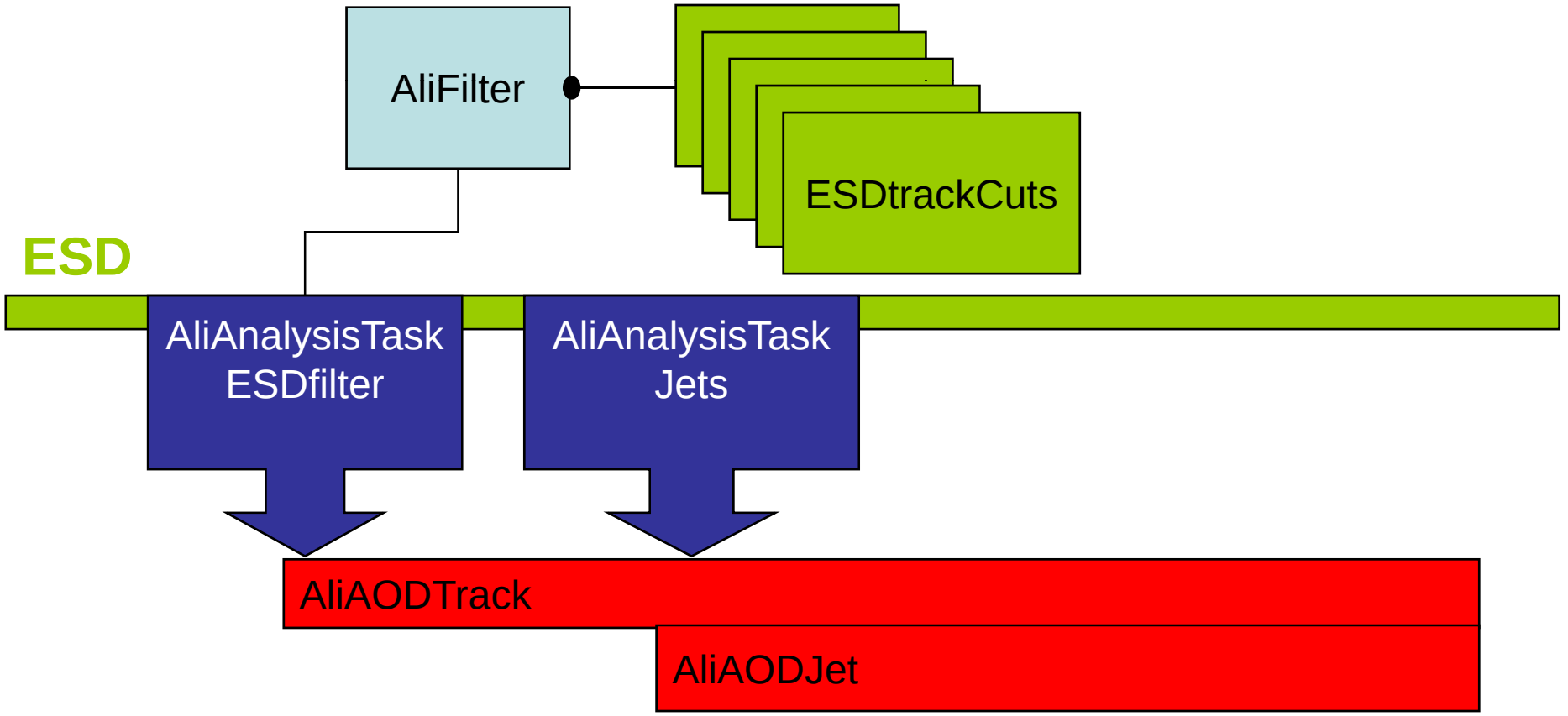
ESD Filter



UInt_t interpreted as a bit-field storing filter information
Bit n 0/1 => Filter n no/yes

Prototype ! Requirements of the Effeciency and Acceptance Framework are being discussed.

TestTrain


AliFilter
ESDtrackCuts
ESD
AliAnalysisTask
ESDfilter
AliAnalysisTask
Jets
AliAODTrack
AliAODJet

Next Steps (from July Offline Week)

- Collect, integrate, assemble and test existing analysis tasks (started but not finished)
- Collect requirements on
  - AOD (done, M. Oldenburg)
  - Cuts for filters (tracks, V0, Kinks) (ongoing, R. Vernet)
  - Number of ESD reading cycles (>1 for flow analysis)
- Define possible interactions with efficiency calculation framework (done, see Silvia's talk)
- MC information handling
  - Kinematics, reference hits (almost done)

Other requirements

- Event merging, for example Pythia+HIJING
- Event mixing
  - Both should be relatively easy to implement using the *VEventHandler*
- For PROOF, possibility to connect Trees to files and merge mechanism for file resident objects.
  - Now a show stopper. Tests are only possible on relatively small event samples.