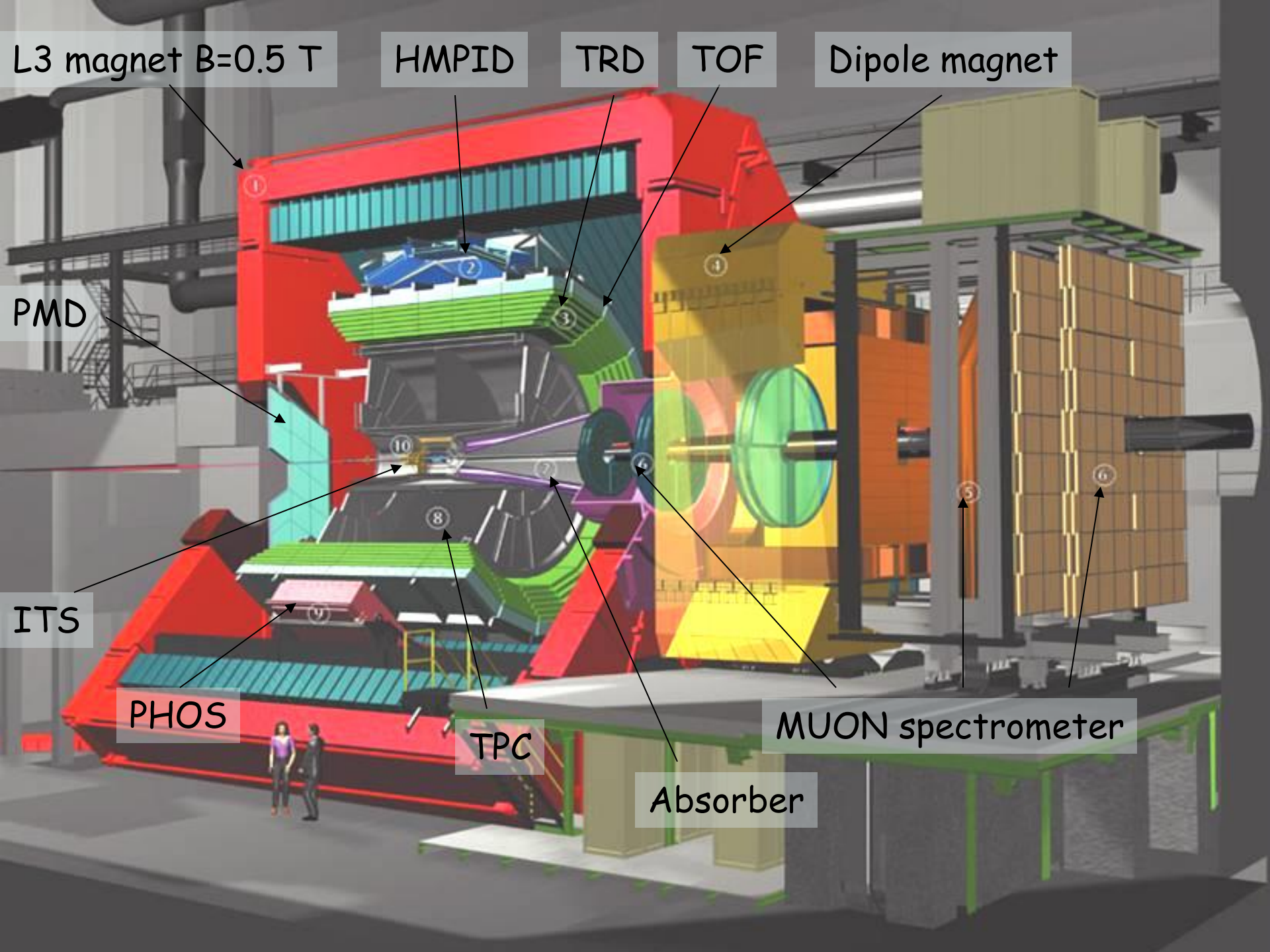


ALICE Production and Analysis Software

P.Hristov
05/03/2014



L3 magnet B=0.5 T

HMPID

TRD

TOF

Dipole magnet

PMD

ITS

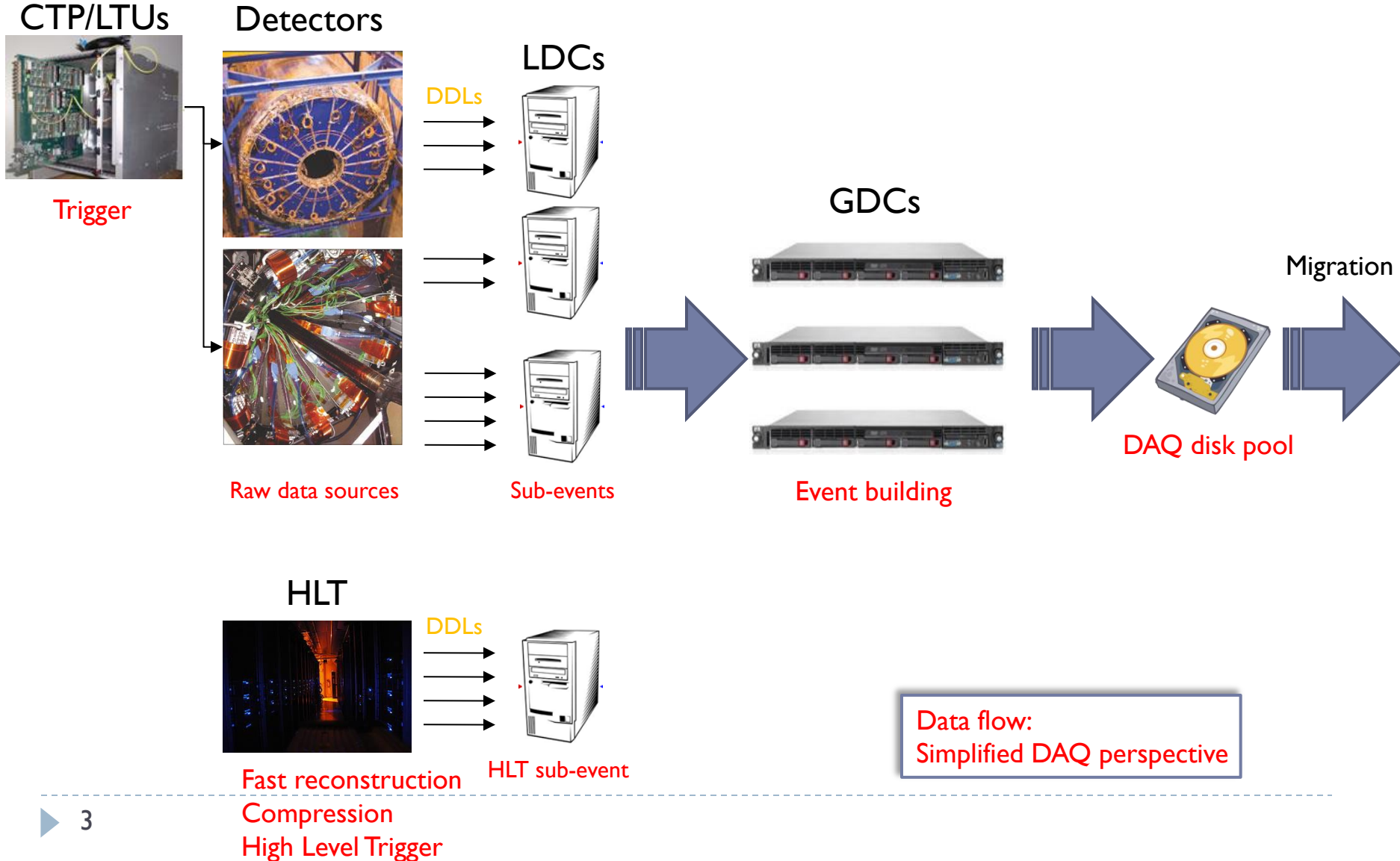
PHOS

TPC

Absorber

MUON spectrometer

Raw data flow

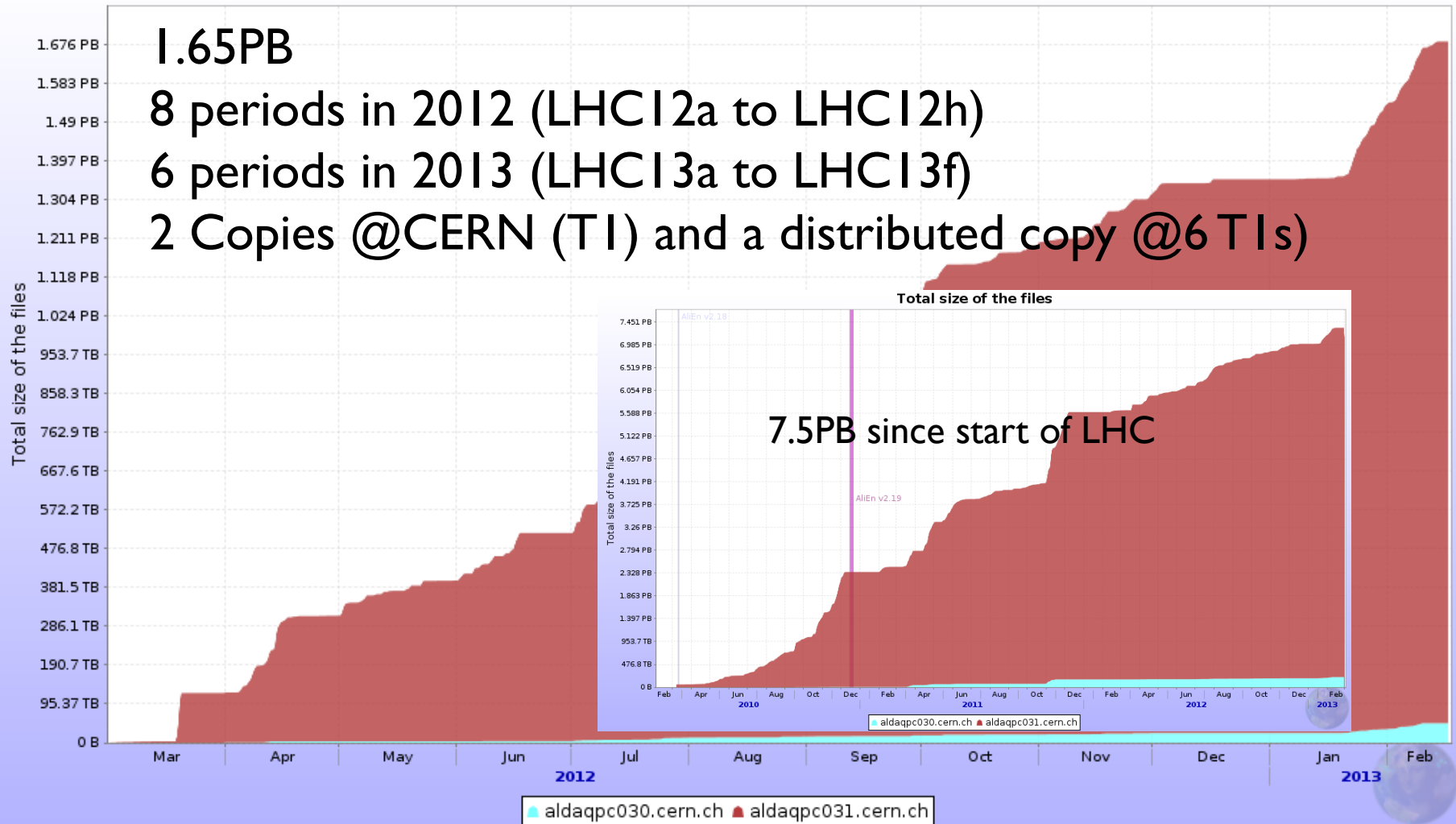


ALICE raw data

- ▶ **proton - proton**
 - ▶ many small events, pileup
 - ▶ typical run of 10 months produces ~ 1 Pb of raw data
- ▶ **PbPb**
 - ▶ from very big central events to very small ultra-peripheral ones
 - ▶ typical run of 1 month produces ~ 1 Pb of raw data
- ▶ **pPb (usually replaces PbPb)**
 - ▶ the events look like “high multiplicity pp”
 - ▶ typical run ~ 1.5 months produces ~ 0.3 Pb
- ▶ **Variety of running conditions for each period**
 - ▶ Different trigger mixtures
 - ▶ HLT compression
 - ▶ Different behavior of detectors (HV, efficiency, noise, etc.)

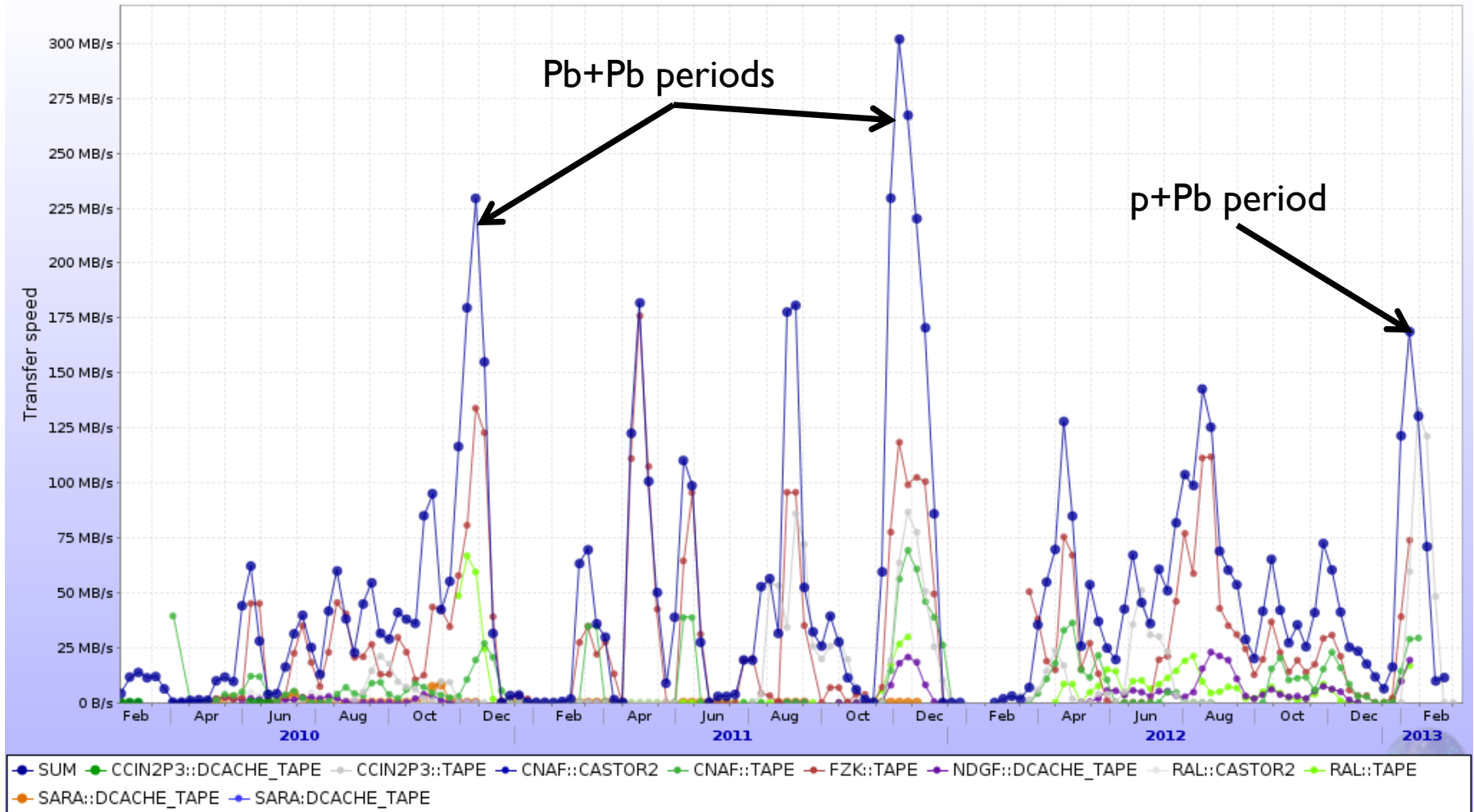
Raw data in 2012/2013

Total size of the files



RAW data transfer

SEs average transfer rates



ALICE Offline Framework

ALICE offline framework: AliRoot

- ▶ AliRoot in development since 1998
 - ▶ Directly based on [ROOT](#)
 - ▶ Used since the detector TDR's for all ALICE studies
- ▶ Few packages to install: ROOT, AliRoot, [Geant3](#), [AliEn](#) client
 - ▶ Optionally use [Geant4](#) or [Fluka](#) instead of Geant3 => additional [VMC](#) package
 - ▶ Optionally DATE + AMORE for DA and DQM development
 - ▶ Optionally [Fastjet](#) + [Boost](#) + [CGAL](#) for jet studies
- ▶ Ported on most common architectures/OS: Linux & Mac OS
- ▶ Distributed development
 - ▶ Over 180 developers and a single [Git repository](#)
 - ▶ analysis:~1.3M SLOC; simulation, reconstruction, calibration, alignment, visualization ~1.4M SLOC, external Fortran code ~0.9M SLOC
- ▶ Integration with DAQ (data recorder) and HLT (same code-base)
- ▶ Abstract interfaces and “restricted” subset of C++ used for maximum portability
- ▶ Used for simulation, reconstruction, calibration (detector algorithms), alignment, quality assurance, monitoring, and analysis

AliRoot installation: short reference

- ▶ [Installation instructions](#) prepared by Dario Berzano
- ▶ [Installation instructions](#) prepared by Christian Holm Christensen
- ▶ Main steps if you compile on a new system (suppose you do it in \$HOME)

- ▶ Download alien installer and install alien (requires valid certificate in .globus)

```
wget http://alien.cern.ch/alien-installer
```

```
chmod +x alien-installer; ./alien-installer
```

- ▶ Make sure that all system packages [required by Root](#) are installed (+libxm2-dev)
- ▶ Download Root from the Git repository and checkout the needed version

```
git clone http://root.cern.ch/git/root.git
```

```
cd root; git checkout v5-34-00-patches
```

- ▶ Configure and compile root

```
./configure --with-pythia6-uscore=SINGLE --with-f77=gfortran --with-alien incdir=<...>/alien/api/include --with-alien-libdir=<...>/alien/api/lib --with-xrootd-incdir=<...>/alien/api/include/xrootd --with-xrootd-libdir=<...>/alien/api/lib
```

```
make -j4
```

- ▶ Set up root, compile and run test

```
source bin/thisroot.sh; cd test; make -j4; ./stress
```

- ▶ Go back to \$HOME, download Geant3 and compile it

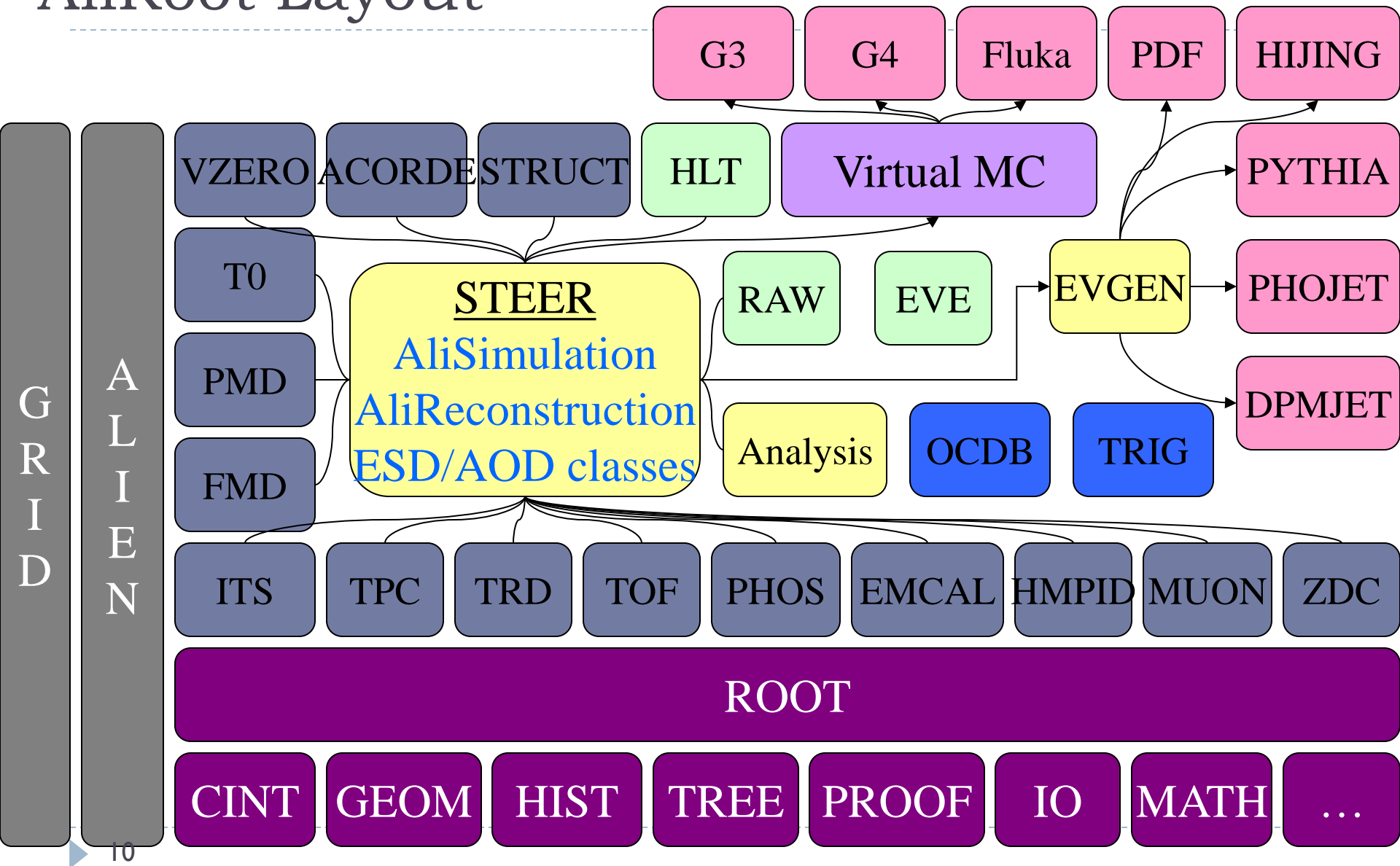
```
cd ~; git clone http://root.cern.ch/git/geant3.git; cd geant3; make -j4
```

- ▶ Download and compile AliRoot. set ALICE_ROOT. Update PATH and LD_LIBRARY_PATH

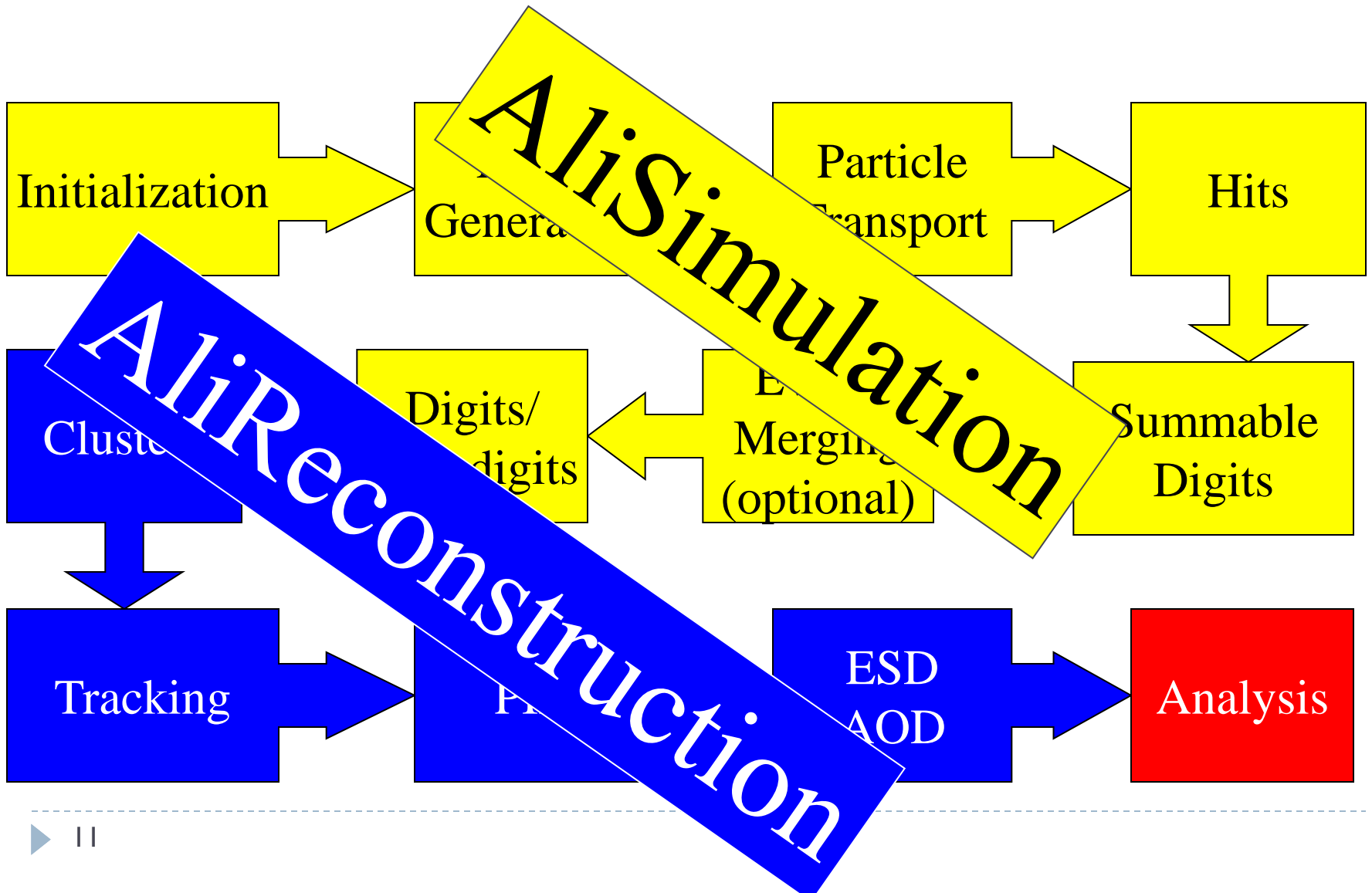
```
git clone http://git.cern.ch/pub/AliRoot.git; export ALICE_ROOT=<...>/AliRoot
```

```
mkdir build; cd build; cmake ../AliRoot; make -j4
```

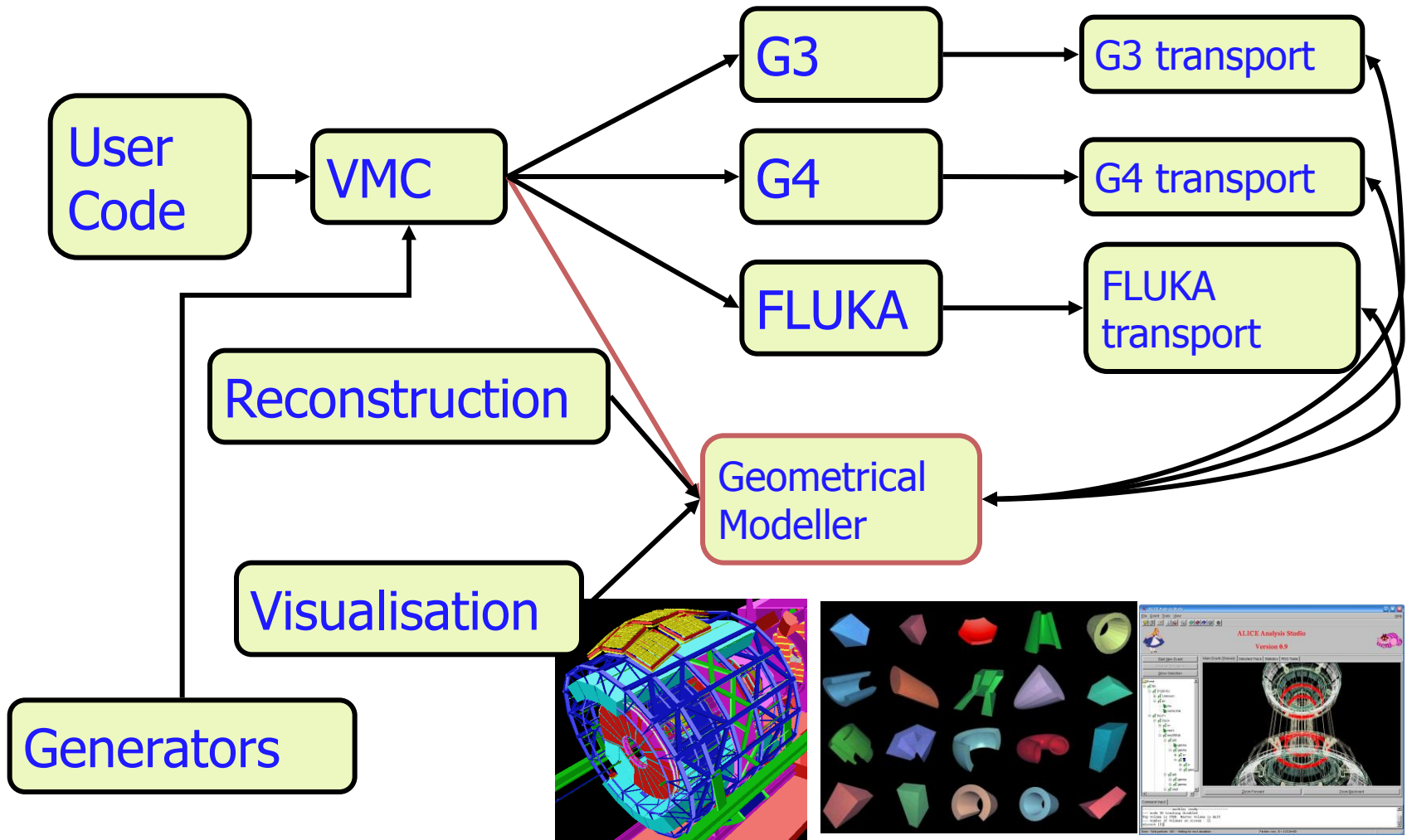
AliRoot Layout



AliRoot: Simulation + Reconstruction



Particle transport



S
I
M
U
L
A
T
I
O
N

Event Generation:
Kinematics

Particle Transport:
Hits (energy
deposition at given
Point, MC label)

Detector Response:
Summable Digits
(low ADC threshold,
no noise, MC label)

Detector Response:
Digits
(noise + normal
Threshold, MC label)

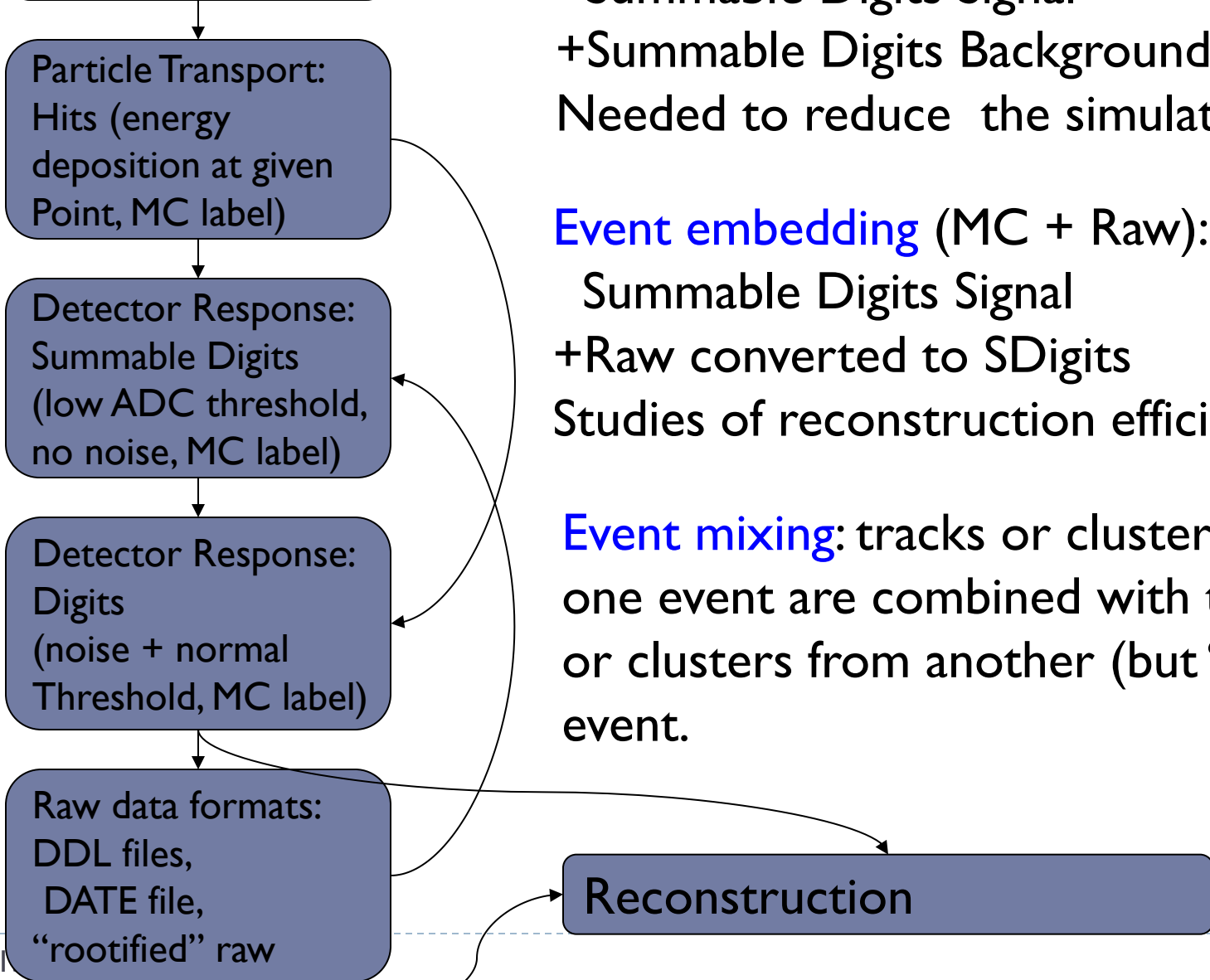
Raw data formats:
DDL files,
DATE file,
"rootified" raw

Event Merging (MC + MC):
Summable Digits Signal
+Summable Digits Background
Needed to reduce the simulation time

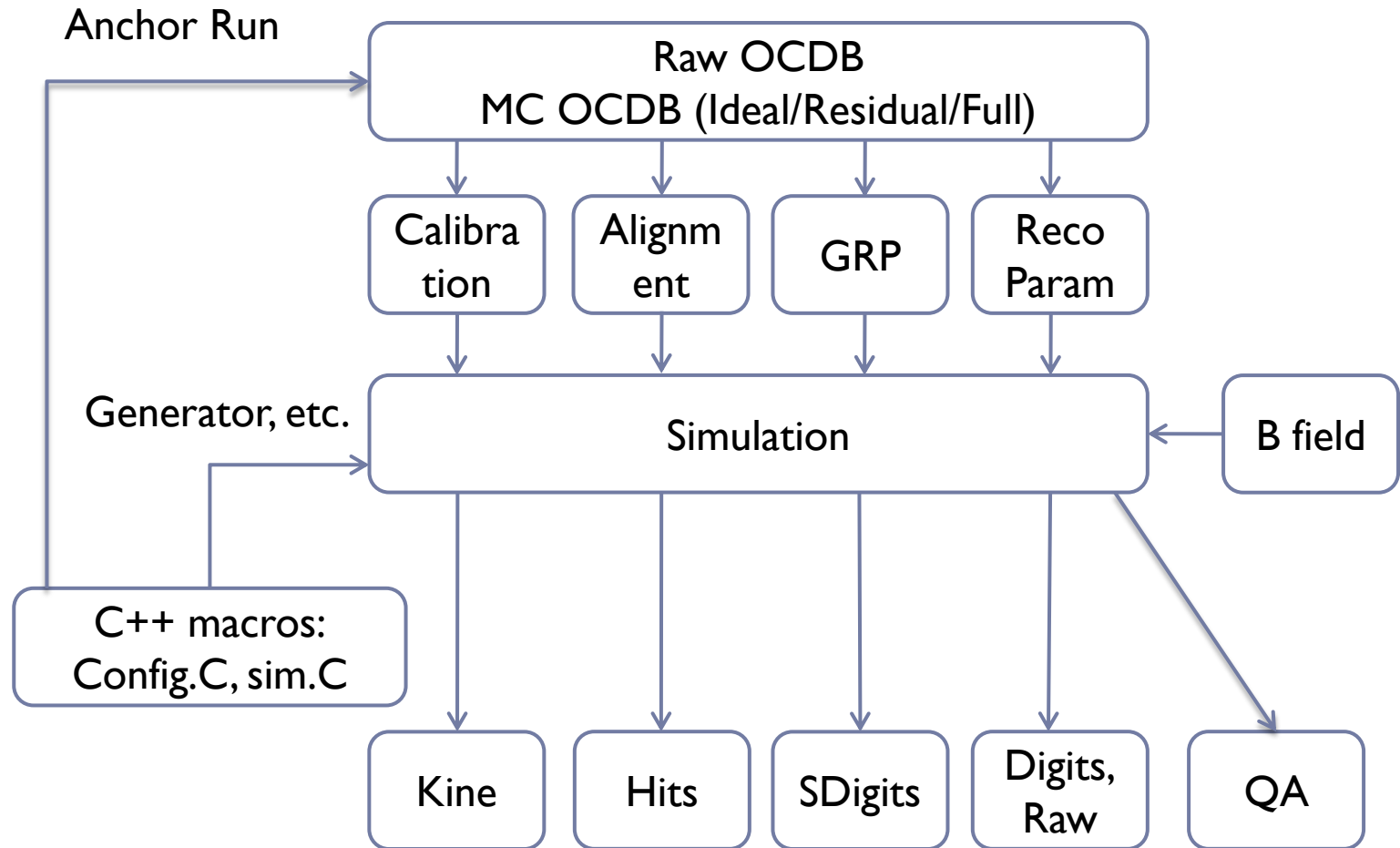
Event embedding (MC + Raw):
Summable Digits Signal
+Raw converted to SDigits
Studies of reconstruction efficiency

Event mixing: tracks or clusters from
one event are combined with tracks
or clusters from another (but "similar")
event.

Reconstruction



Simulation: Data flow

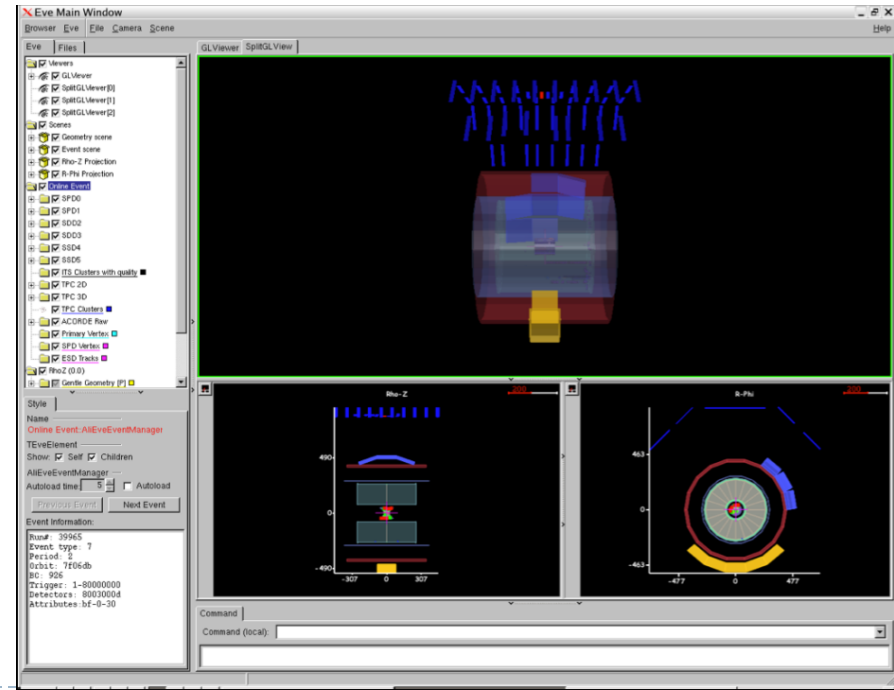
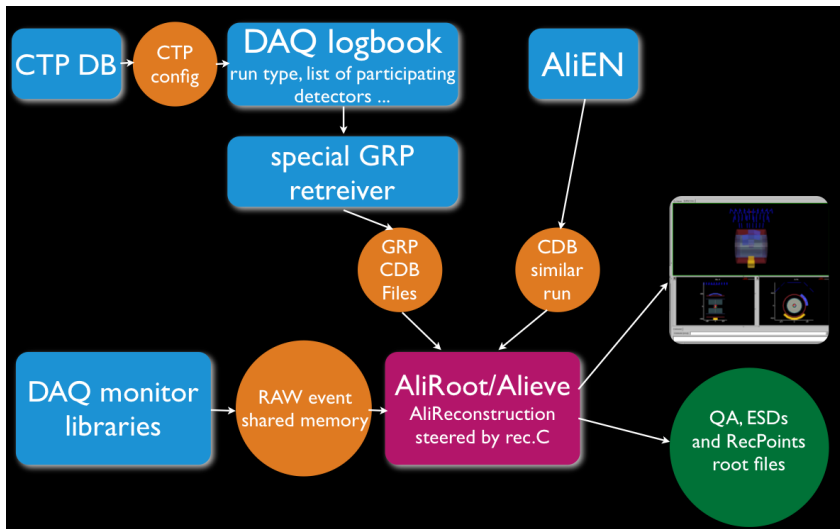
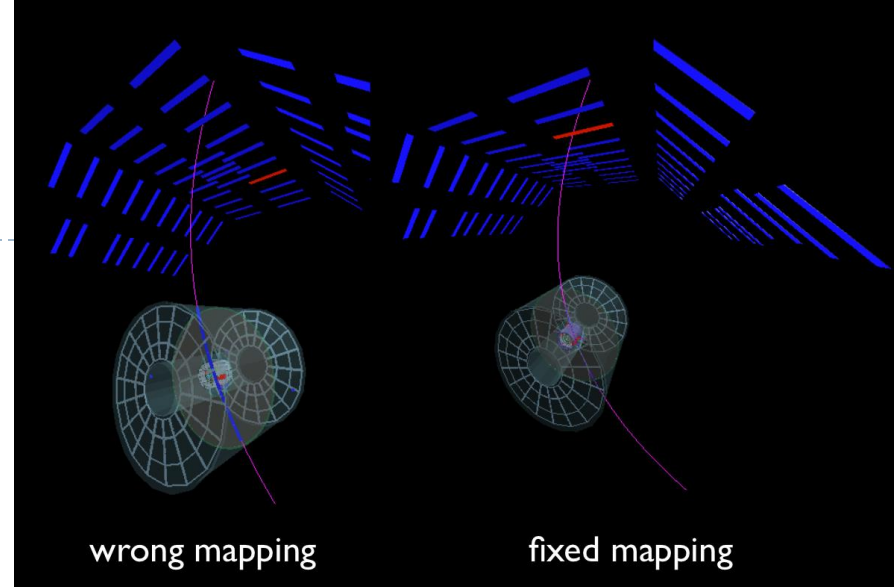


Preparation of simulation request

- ▶ Define the anchor runs
- ▶ Decide which generator you need
 - ▶ proton-proton MB: Pythia6, Phojet, Pythia8
 - ▶ Pb-Pb MB: Hijing
 - ▶ p-Pb MB: DPMJET, Hijing
- ▶ Decide which signals you want to add, i.e. heavy flavors
- ▶ Prepare Config.C, sim.C, rec.C. Make sure that you use the correct OCDB objects from the specific storage
- ▶ Run local test
- ▶ Prepare the JDL file: software versions, input/output files, resources...
- ▶ Run GRID test and estimate the size of the output and required CPU time
- ▶ Get your request approved by the PWG and Physics Board

Reconstruction

- Getting the original physics event information (tracks, particles) from detector signals (digitized information)
- Prompt reconstruction
 - Integrated with the AliEVE event display
 - Full Offline code sampling events directly from DAQ memory
- Reconstruction is subject to refinement
 - Subsequent passes after improved calibration
- Reconstruction produces **events** as input for analysis



RECONSTRUCTION

BARREL TRACKING

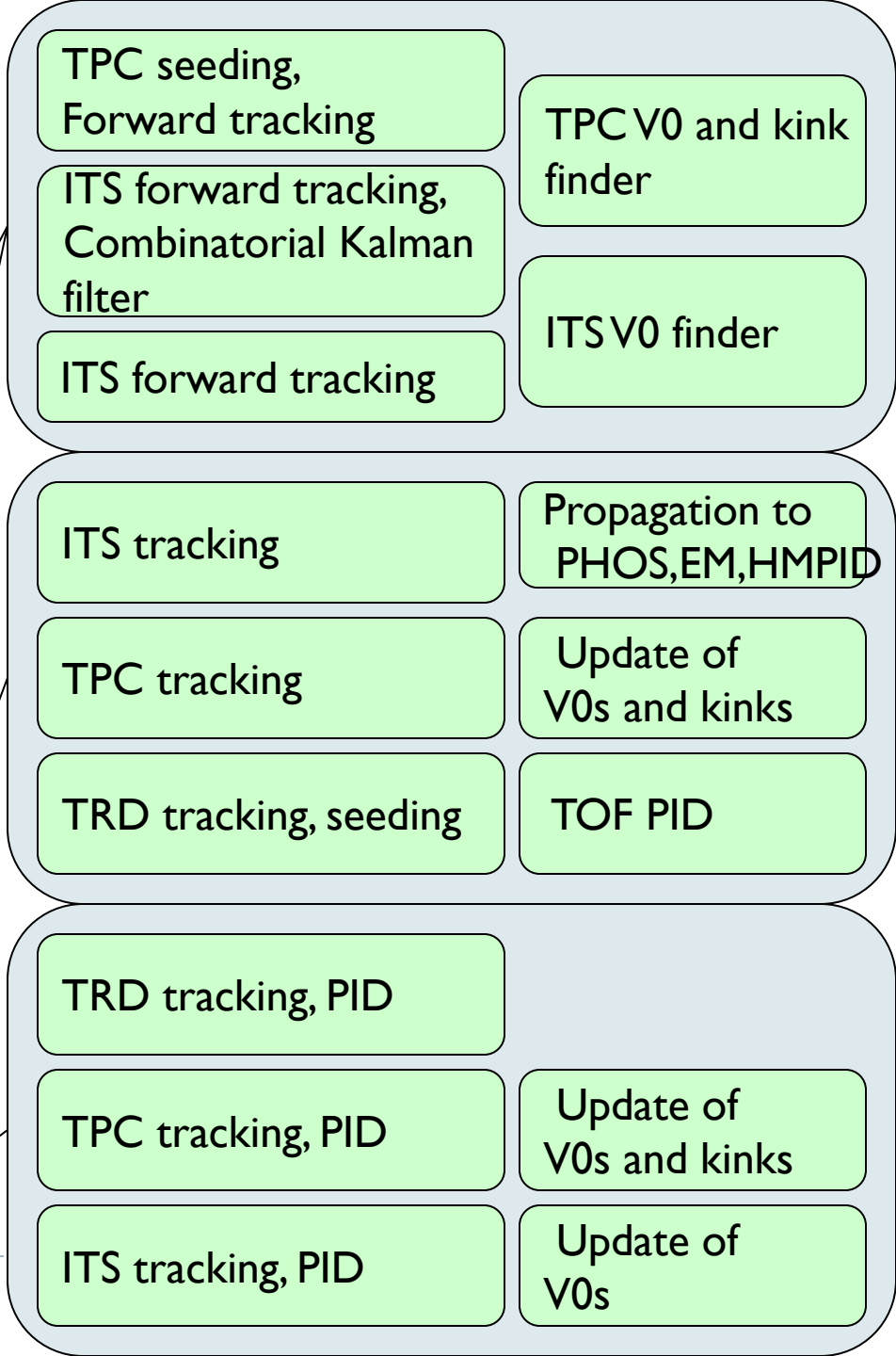
Local Detector
Reconstruction:
Clusterization, vertex

Seeding:
2 clusters in TPC
+ primary vertex

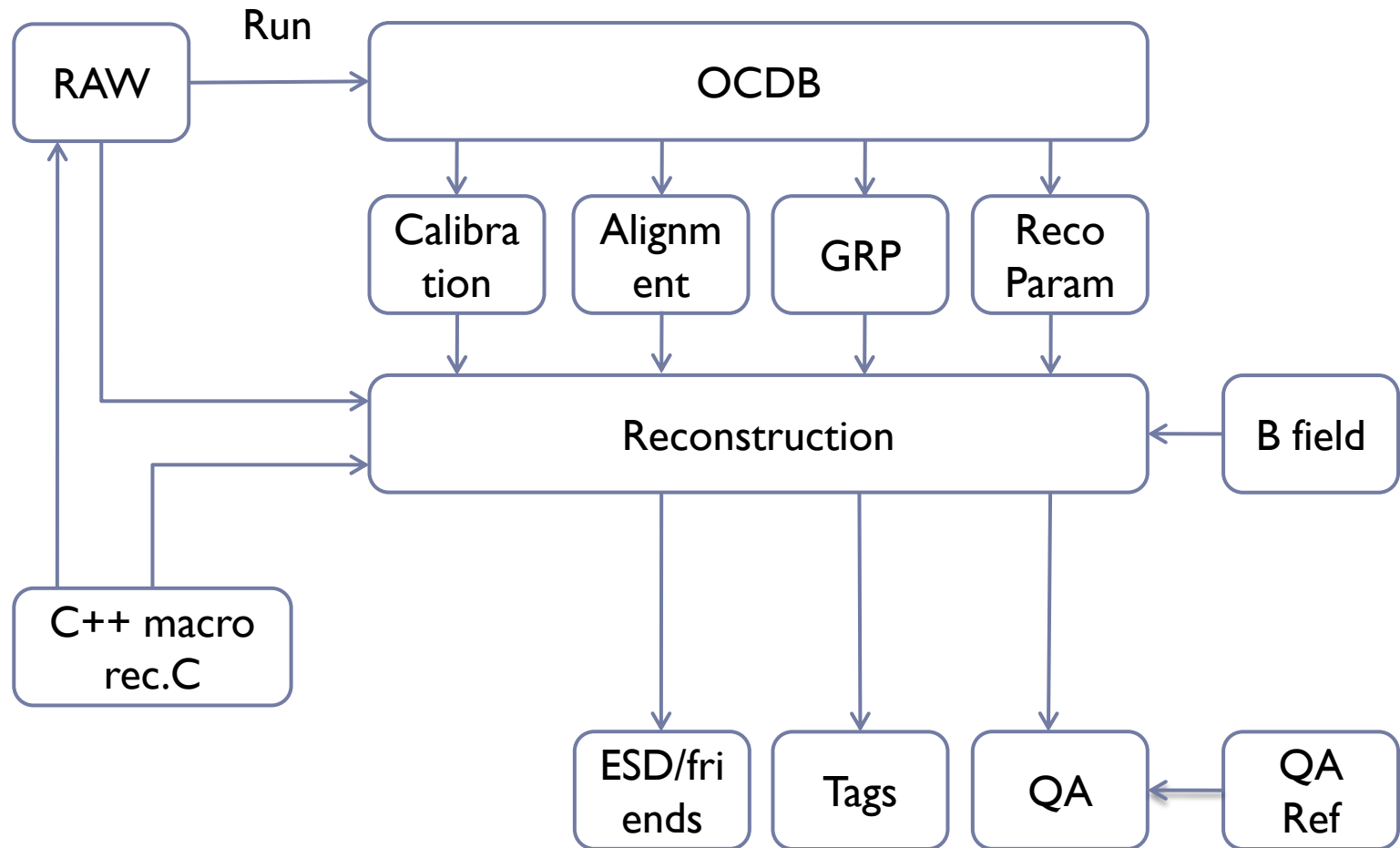
Kalman Filter:
Forward propagation
TPC→ITS

Kalman Filter:
Backward propagation
ITS→TPC→TRD→TOF

Kalman Filter:
Refit inward
TRD→TPC→ITS



Reconstruction: Data flow



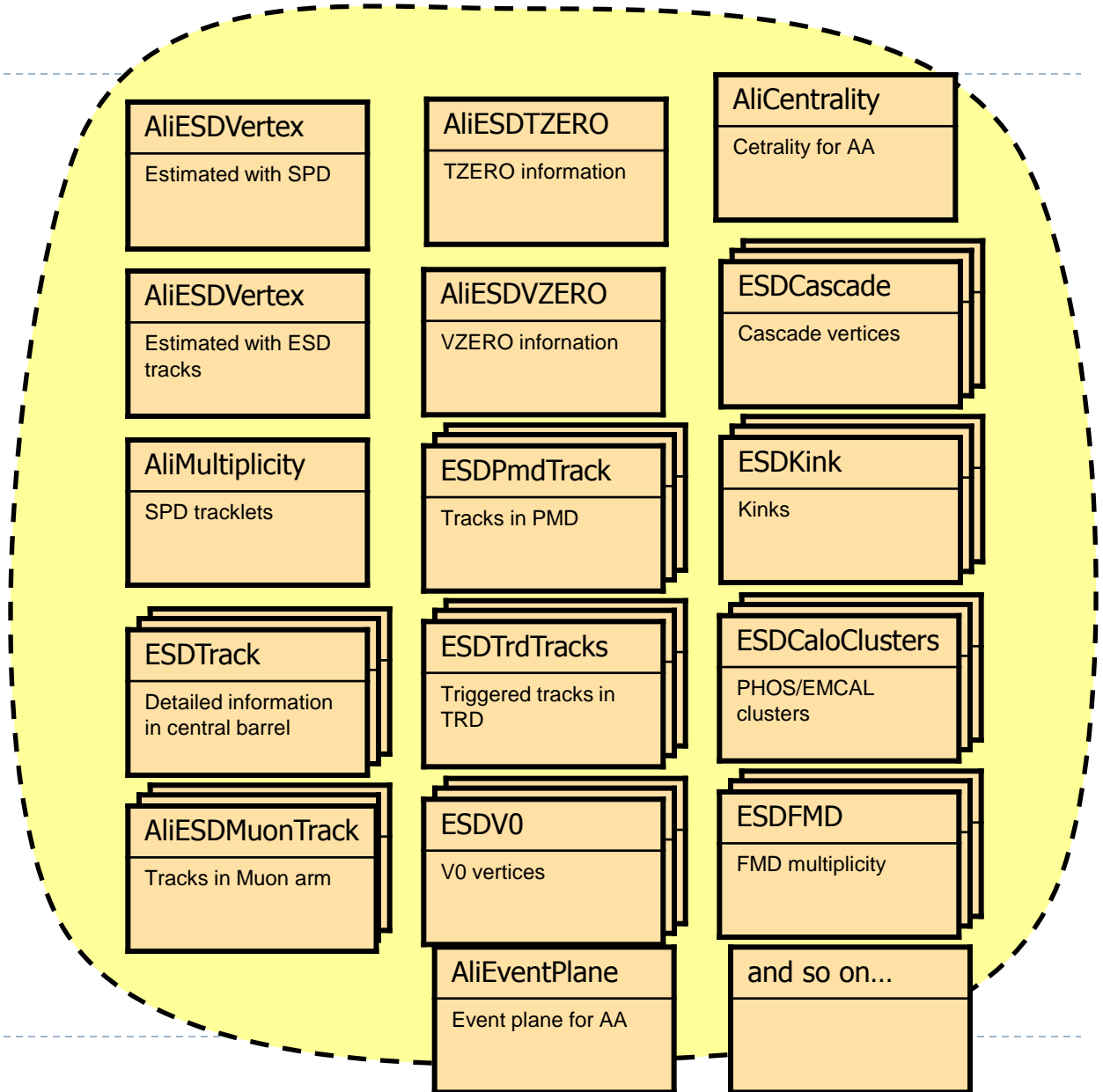
Raw data processing

- ▶ Calibration (reconstruction with special settings + QA + analysis) => OCDB update
 - ▶ CPass0 (mainly for barrel detectors, i.e.TPC)
 - ▶ CPass1 (all detectors, i.e.TOF)
 - ▶ Manual calibration
- ▶ Validation (reconstruction of 10% raw sample, standard settings + QA trains)
 - ▶ VPass
- ▶ Production (reconstruction of all collected raw data + AOD filtering)
 - ▶ PPass

Result of reconstruction: ESD

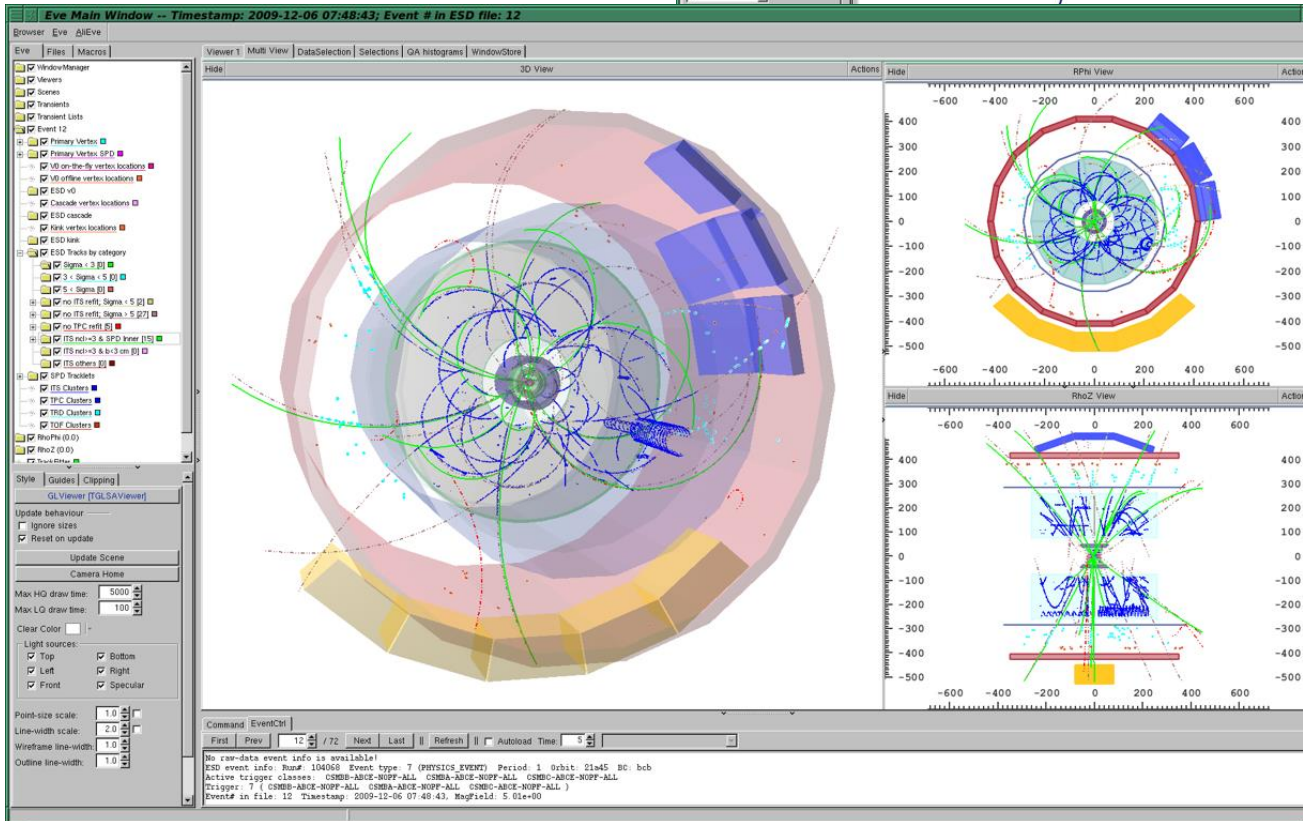
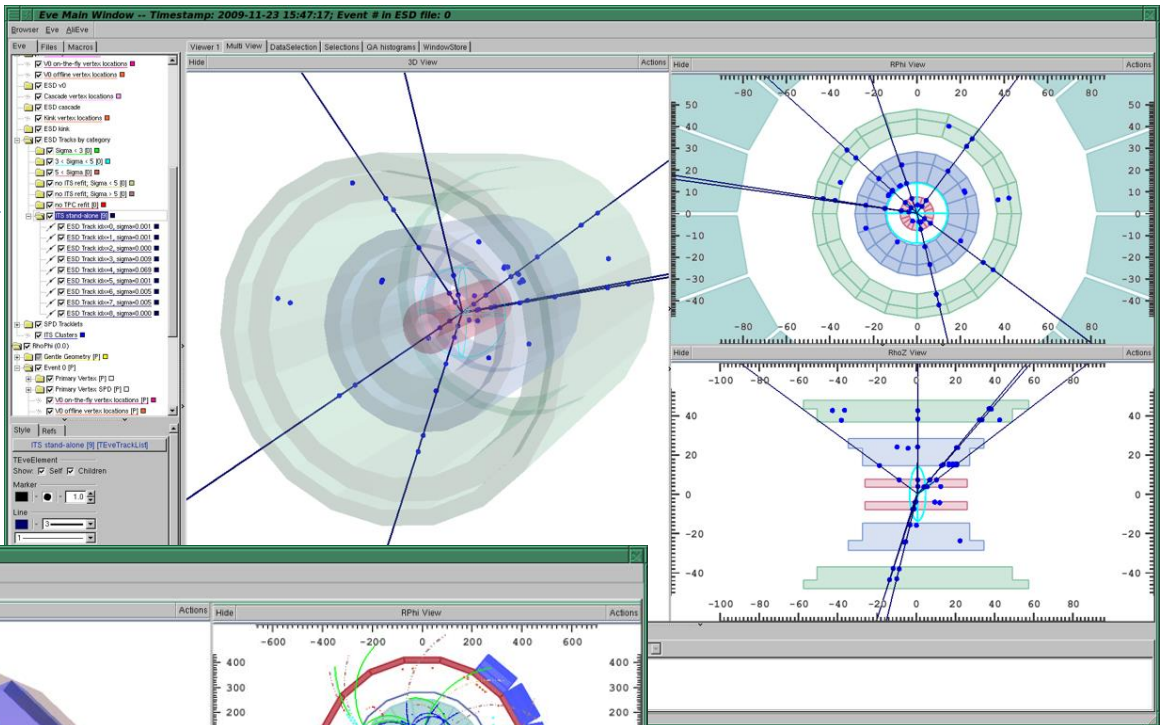
- ▶ **ESD – Event Summary Data**
 - ▶ Contains all the information needed for analysis
 - ▶ Large size, a lot of I/O => should be avoided
- ▶ **Filtering procedure**, removing most detector information, useless tracks -> new format for analysis: **AOD – Analysis Object Data**
 - ▶ General purpose AOD: contain only what is really needed for the analysis
 - ▶ Analysis-specific AOD: contain the additional information for a given analysis, i.e. for heavy flavor analysis
 - ▶ **Physicists are strongly encouraged to use AOD**
 - ▶ smaller size, analysis specific information => efficient IO and CPU use

The ESD



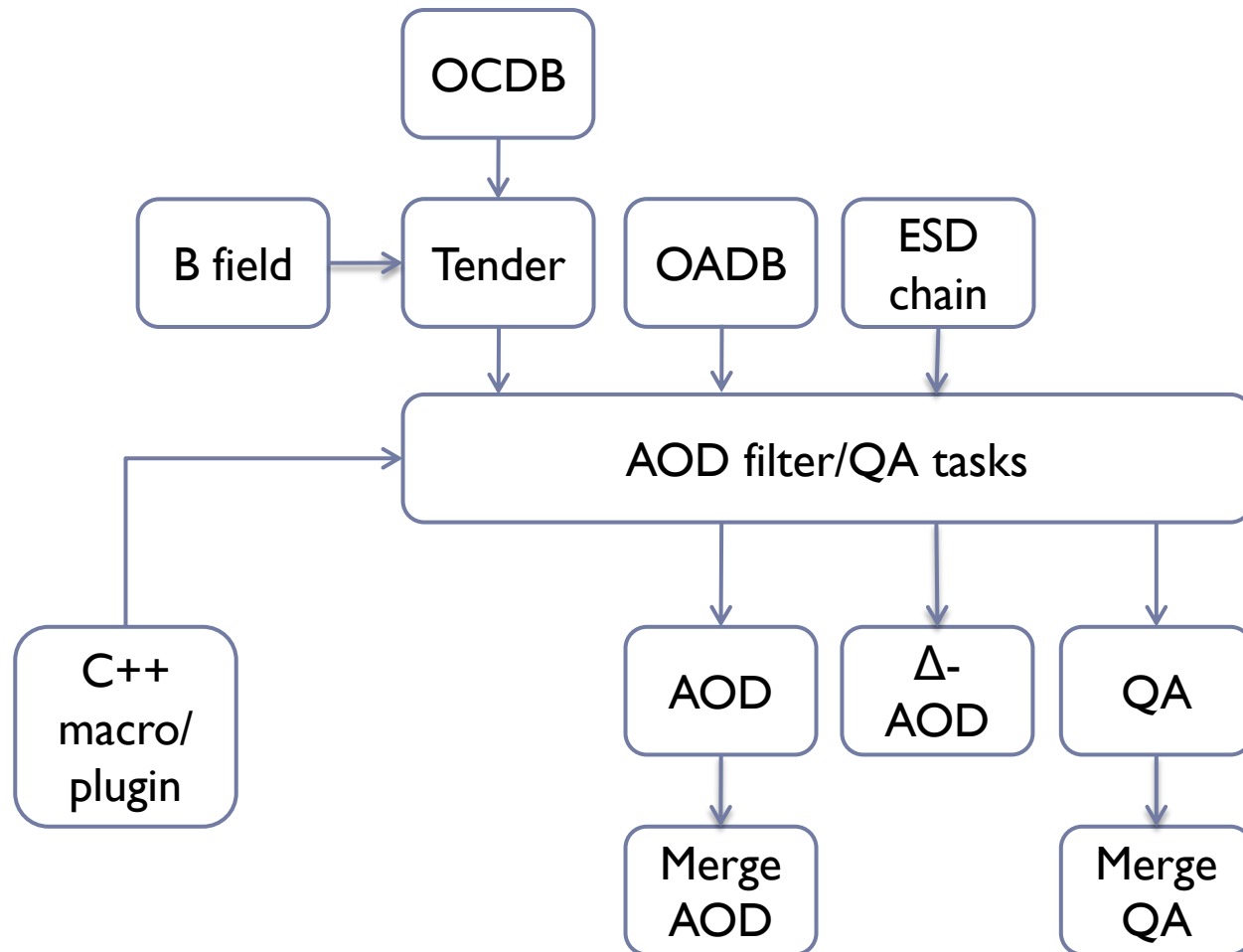
- AliESDEvent has a TList of objects and containers (TClonesArrays) "to be extendable"
- Non-persistent pointers to the "standard" content
- Heavy object
- Complex IO (de-serialization)
- Long inheritance chains for the content of containers. Example
 - AliESDTrack
 - AliExternalTrackParam
 - AliVTrack
 - AliVParticle
 - TObject

Event display: AliEve



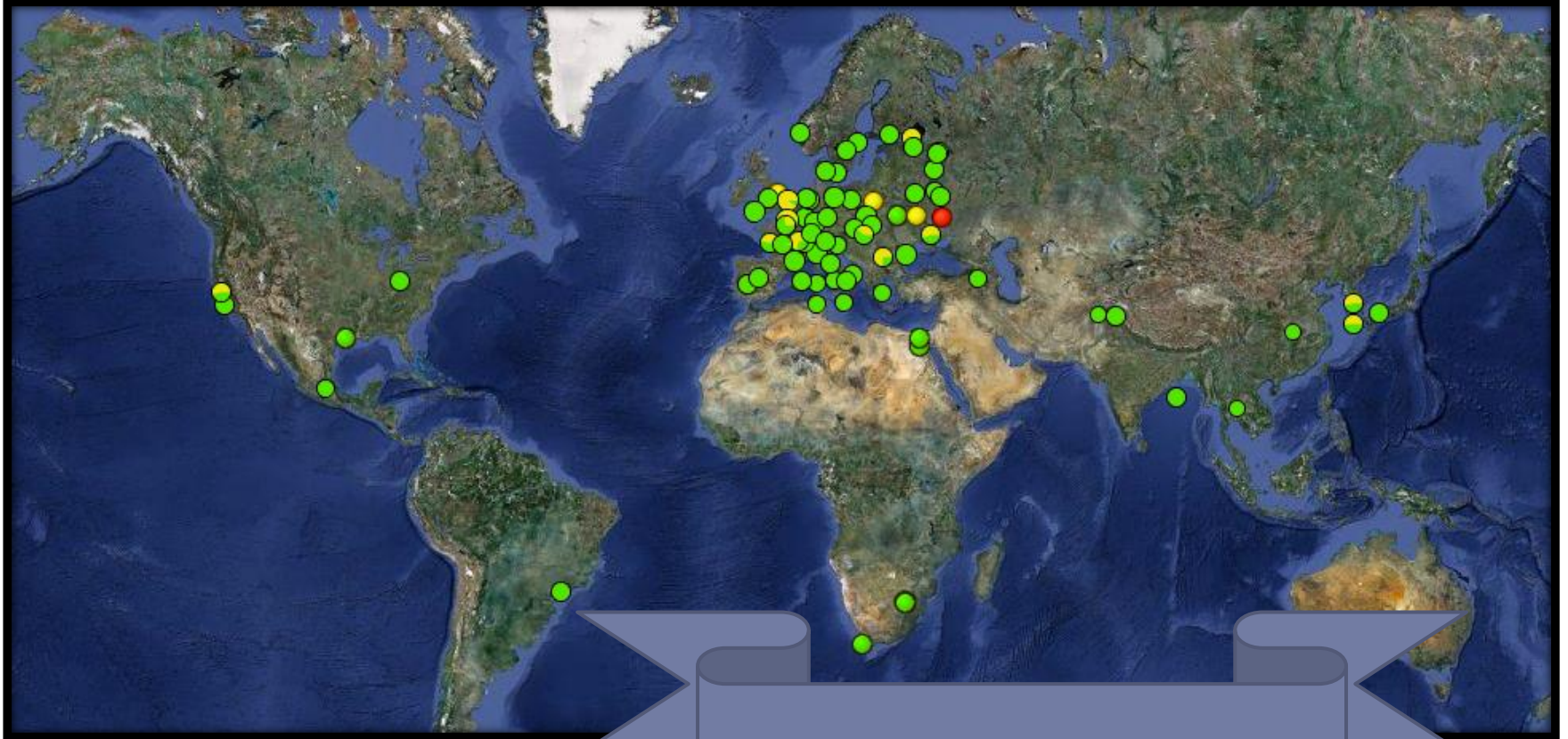
Visualization of
reconstructed
events

From ESD to AOD





Distributed data and processing



More than 80 centres

The ALICE Grid

■ AliEn working prototype in 2002

■ Single interface to distributed computing for all ALICE physicists

■ File catalogue, job submission and control, software management, user analysis

■ ~80 participating sites now

■ 1 T0 (CERN/Switzerland)

■ 6 T1s (France, Germany, Italy, The Netherlands, Nordic DataGrid Facility, UK)

■ ~73 T2s spread over 4 continents

■ ~50,000 cores, 13.5 PB of disk & 15.5PB tape

■ Resources are “pooled” together

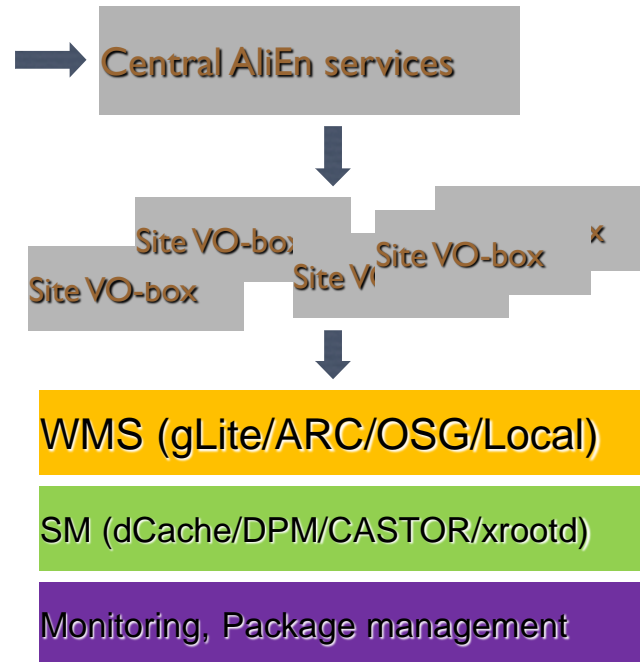
■ No localization of roles / functions

■ National resources must integrate seamlessly into the global grid to be accounted for

■ FAs contribute proportionally to the number of PhDs (M&O-A share)

■ T3s have the same role than T2s, even if they do not sign the MoU

GRID operation principle



- VO –Virtual Organization
- The VO-box system (very controversial in the beginning)
 - Has been extensively tested
 - Allows for site services scaling
 - Is a simple isolation layer for the VO in case of troubles

AliEn

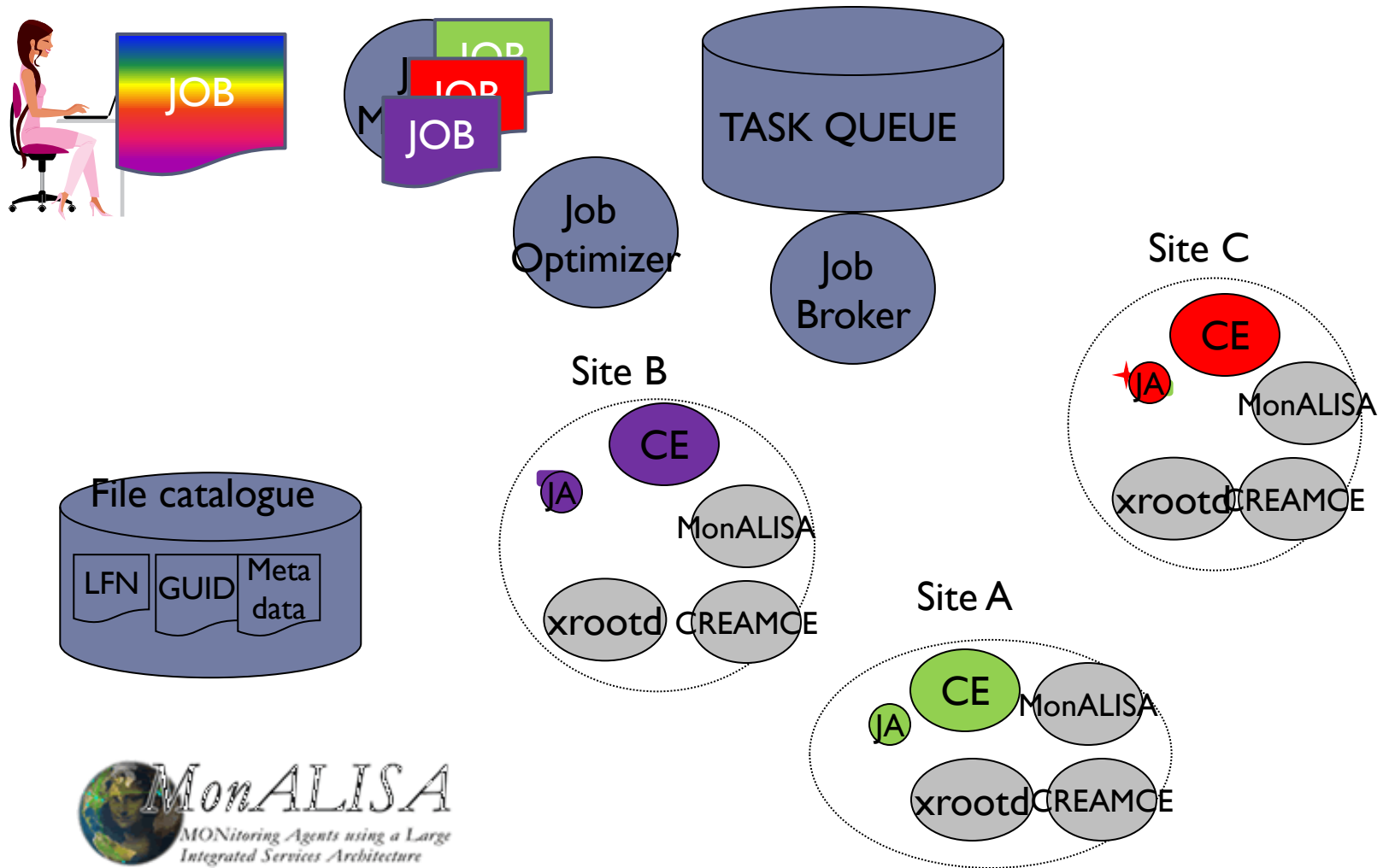
- ▶ All components to create a GRID
- ▶ File Catalogue
 - ▶ UNIX-like file system
 - ▶ Mapping to physical files
 - ▶ Metadata information
 - ▶ SE discovery
- ▶ Developed by ALICE
 - ▶ Used by several communities
- ▶ Transfer Model
 - ▶ With different plugins
- ▶ Task Queue
 - ▶ Job Agent & pull model
 - ▶ Automatic installation of software packages
 - ▶ Simulation, reconstruction, analysis...



Job Execution

- ▶ Central Task Queue with all jobs
- ▶ Optimizers: rearrange
 - ▶ Priorities & quotas on user/role level
- ▶ VO-box on sites that submit to batch system
- ▶ Job Agents (JA)
 - ▶ Install AliEn/ Software packages
 - ▶ Sanity check
 - ▶ Validate output
 - ▶ Upload output

Job execution



Summary of operations 2012/13

- ▶ 60% simulation
- ▶ 10% organized analysis
- ▶ 10% RAW data reconstruction
- ▶ 20% individual user analysis (465 users)

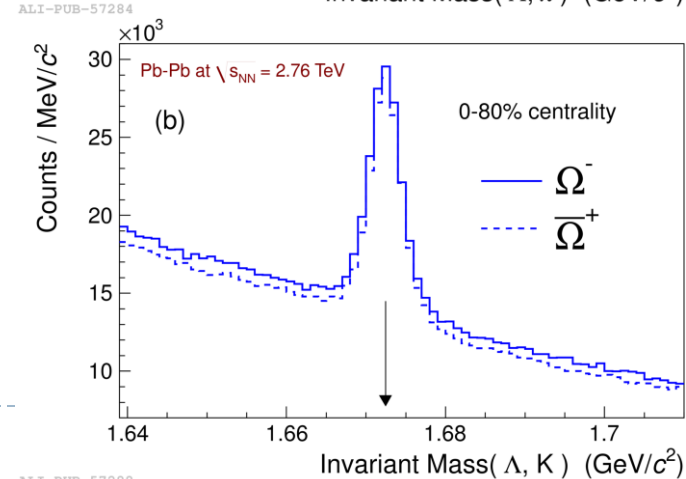
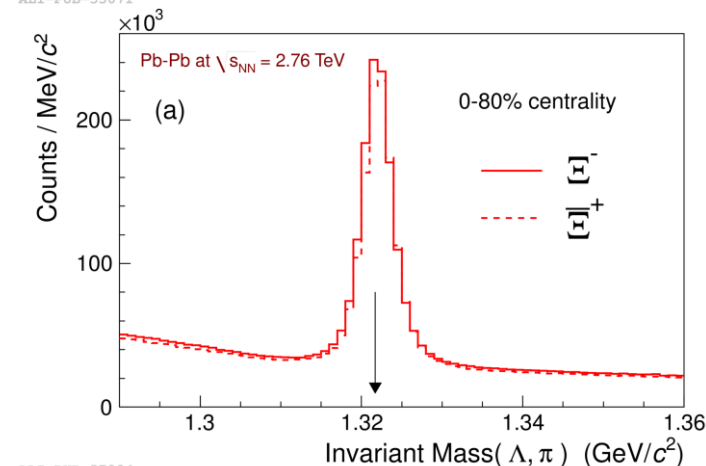
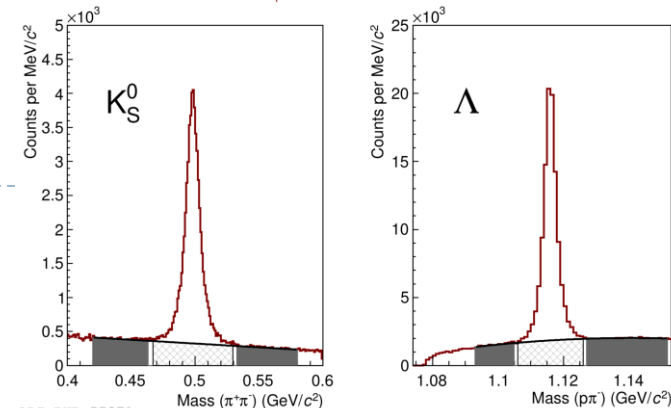
Data Analysis Tools for ALICE



ALICE Analysis Basic Concepts

- Analysis Framework
 - Generic data-driven framework + ALICE specific layer
- Supported Analysis Modes
 - Local analysis
 - Prompt data processing (calib, align, reco, analysis) @CERN with PROOF
 - Batch Analysis using GRID infrastructure
- User Interface
 - Access GRID via AliEn or ROOT UIs
 - UI provided by the analysis framework
- PROOF/ROOT
 - Enabling technology for (C)AF
 - GRID API class TAlien
- Analysis Object Data contain data needed for several particular analyses
 - Extensible with Δ -AODs
- Same user code local, on CAF and Grid

Pb-Pb at $\sqrt{s_{NN}} = 2.76$ TeV, $|y| < 0.5$
 $3.0 < p_T < 3.2$ GeV/c, 0-5% centrality



Main goals for the analysis framework in ALICE

- ◎ Provide transparent access to all resources with the same code

SERVICES

- Usage: Local, AliEn grid, CAF/PROOF

- ◎ Provide transparent access to different data types

- ESD, AOD, Kinematics tree (MC truth)

DATA STRUCTURES

- Tracks, vertices, cascades, V0's, ...

- ◎ Allow sharing resources by multiple analysis modules – efficiency & better CPU/IO

FORMALIZE USER ANALYSIS

- Framework must allow accommodating different analysis in the same session

ANALYSIS FLOW

Analysis today and tomorrow

- ▶ Input: storing ~4 PB/month data suitable for analysis
- ▶ Processing: ~20 PB/month (using 1/3 of total GRID CPU resources)
- ▶ Growth: ~20% increase/year in computing capacity
- ▶ Resource migration towards analysis: slowly growing to ~50% by LS2
- ▶ => Assess the situation today by improving monitoring tools & learn from today's mistakes...
- ▶ Large improvements needed to analyze the higher rates after LS2
 - ▶ Analysis job efficiency + time to solution
 - ▶ Cut down turnaround time in distributed analysis
 - ▶ I/O improvements (data size, transaction time, throughput, ...)
 - ▶ Low level parallelism (IPC, pipelining, vectorization)

Analysis Framework: Advantages and disadvantages

- ▶ Interfaces for uniform navigation
- ▶ Access to all resources
- ▶ Uniform analysis model and style
- ▶ Job control and traceability
- ▶ Sharing input data for many analysis in a train
- ▶ Increasing quota of organized analysis
- ▶ Flexible AOD format to accommodate any type of analysis
- ▶ Big event size
- ▶ High impact of user errors
- ▶ Insufficient control of bad usage patterns
- ▶ Uncoordinated analysis triggers inefficient use of resources

Analysis: typical steps

- ▶ Define the goals of analysis and the methods to achieve them
- ▶ Check for similar analyses or possibility to reuse existing code
 - ▶ PWG: General Code – 67 tasks
 - ▶ PWGCF: Correlations, Fluctuations and Bulk – 48 tasks
 - ▶ PWGDQ: Dileptons and Quarkonia – 12 tasks
 - ▶ PWGGA: Gamma and Pi0 – 44 tasks
 - ▶ PWGHF: Heavy Flavour – 47 tasks
 - ▶ PWGJE: Jets – 71 tasks
 - ▶ PWGLF: Light Flavour Spectra – 94 tasks
 - ▶ PWGPP: Physics Performance – 51 tasks
 - ▶ PWGUD: Ultraperipheral, Diffractive, Cosmics and pp First Physics – 18 tasks
 - ▶ Example analysis task: ANALYSIS/examples

Typical steps

- ▶ Check for useful information [ALICE TWiki Home Page](#)
i.e.
 - ▶ ALICE Physics Working Groups
 - ▶ Analysis code and technical information
 - ▶ Analysis specific pages
 - ▶ [Analysis Trains with the LEGO Framework](#)
 - ▶ Search engine, i.e. “Lego Train”
- ▶ Check the [ALICE Public/Analysis Notes](#)
- ▶ Be aware that [the Analysis](#) page contains some obsolete information (will be fixed soon)
- ▶ [ALICE Analysis User Guide \(aliensh\)](#)

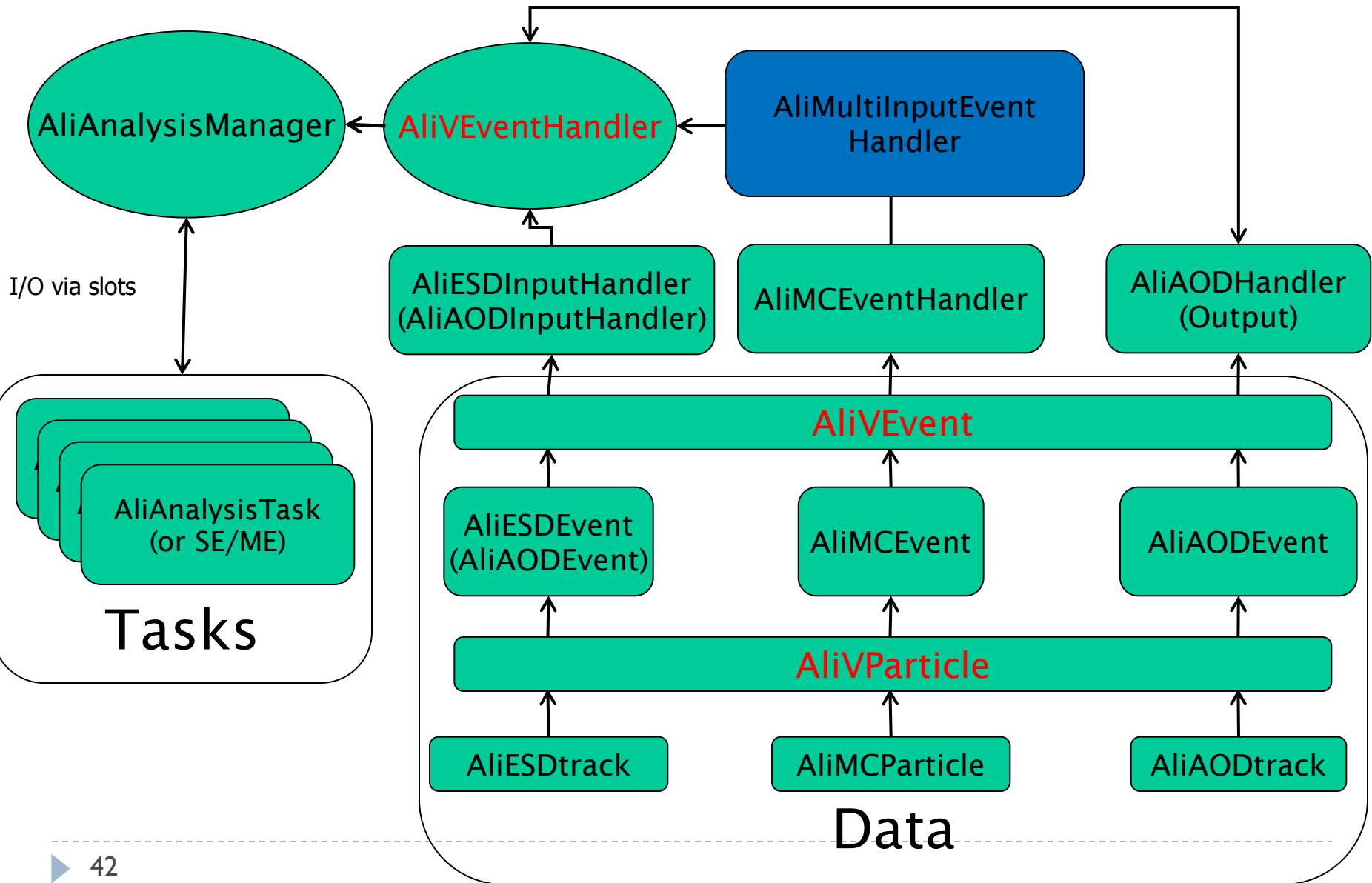
Typical steps

- ▶ Check the availability of the data
 - ▶ [Run Condition Table](#)
 - ▶ Select the run period from the menu (top left): i.e. LHC10b
 - ▶ Select the reconstruction pass in the “Quality” field: i.e. pass2
 - ▶ Select “Global quality” 1 (add 1 in the field above “Global quality” and press “Enter”)
 - ▶ Check the “Comment” field (require “ok”)
 - ▶ Make other selections (see the help “How to select runs”, top right)
 - ▶ Optionally for the selected runs see the logs in the [ALICE Electronic Logbook](#) (i.e. Run Quick Access 117112)

Selected information from the offline tutorial

- ▶ Recent version from [March 2013](#)
- ▶ Part II: Analysis framework
 - ▶ Correction framework and mixed events are not covered
- ▶ Part III: Alien
 - ▶ aliensh is not covered
- ▶ Part IV: PROOF
- ▶ LEGO trains

The overall picture



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
- ▶ Optionally, add to the organized analysis (i.e. Lego Trains)

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

- ▶ **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#Task/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);

// Make sure you get information about 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)
- ▶ ... 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 analysis 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

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 task?
 - ▶ “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
 - ▶ Examples for AliAnalysisTask are in the tarball (Task/) for reference

Notes about streaming

- ▶ A task working perfectly locally can fail in distributed mode
 - ▶ There is an intermediate step for streaming your class object to the workers
- ▶ What to check:
 - ▶ Do I initialize all data members in my constructors ?
 - ▶ I use **!!!** Comments to all histograms and objects that are created on the workers (in `UserCreateOutputObjects`) but NOT for data members that are initialized in the initial configuration phase (constructor, `LocalInit`)
 - ▶ I use **`ClassDef(MyClass, n)`** in my header file

Example: 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

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. Initialize all data members correctly. DO NOT allocate
  // objects here, since this is called by ROOT while streaming and will result in
  // memory leaks.
}
```

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



Constructor: additional

- ▶ **User analysis module MUST derive from AliAnalysisTask**
 - ▶ DefineInput/Output(Int_t islot, TClass *type)
 - ▶ Declared in the named class constructor
 - ▶ Mandatory at least 1 input & 1 output slots
- ▶ **... or AliAnalysisTaskSE**
 - ▶ Predefined input (TChain) and output (AOD) at slot #0
 - ▶ User must define at least one extra output slot

UserCreateOutputObjects()

```
// Book all custom output objects
  OpenFile(I); // optional
// This connects all booked histograms/trees to the output
// file and makes sense only for large outputs

...
fHisto = new TH1F("fHisto", "My Histo", 100, 0., 10.);
.....
// Several histograms are more conveniently managed in a
// TList
fListOfHistos = new TList();
fListOfHistos->SetOwner(); // MANDATORY! To avoid leaks in merging
fListOfHistos->Add(fHisto);
PostData(I, fListOfHistos);
// Just to avoid merging problems when the output is not
// produced
```

UserExec()

```
void AliAnalysisTaskJets::UserExec(Option_t **option*)
{
    // Execute analysis for current event
    // This actually implements how the analysis module
    // processes the current event from input data

    // Jet finding is delegated access to input output and
    // MC given by TaskSE
    fJetFinder->GetReader()->SetInputEvent(InputEvent(), AODEvent(), MCEvent());
    fJetFinder->ProcessEvent();
    ...
    fHisto->Fill(pt);
    ...
    // Post the data (it serves as notification message for
    // possible client tasks. Not posted data is also not
    // written to file.)
    PostData(1, fListOfHistos);
}
```

Called for each event



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();
  }
}
```

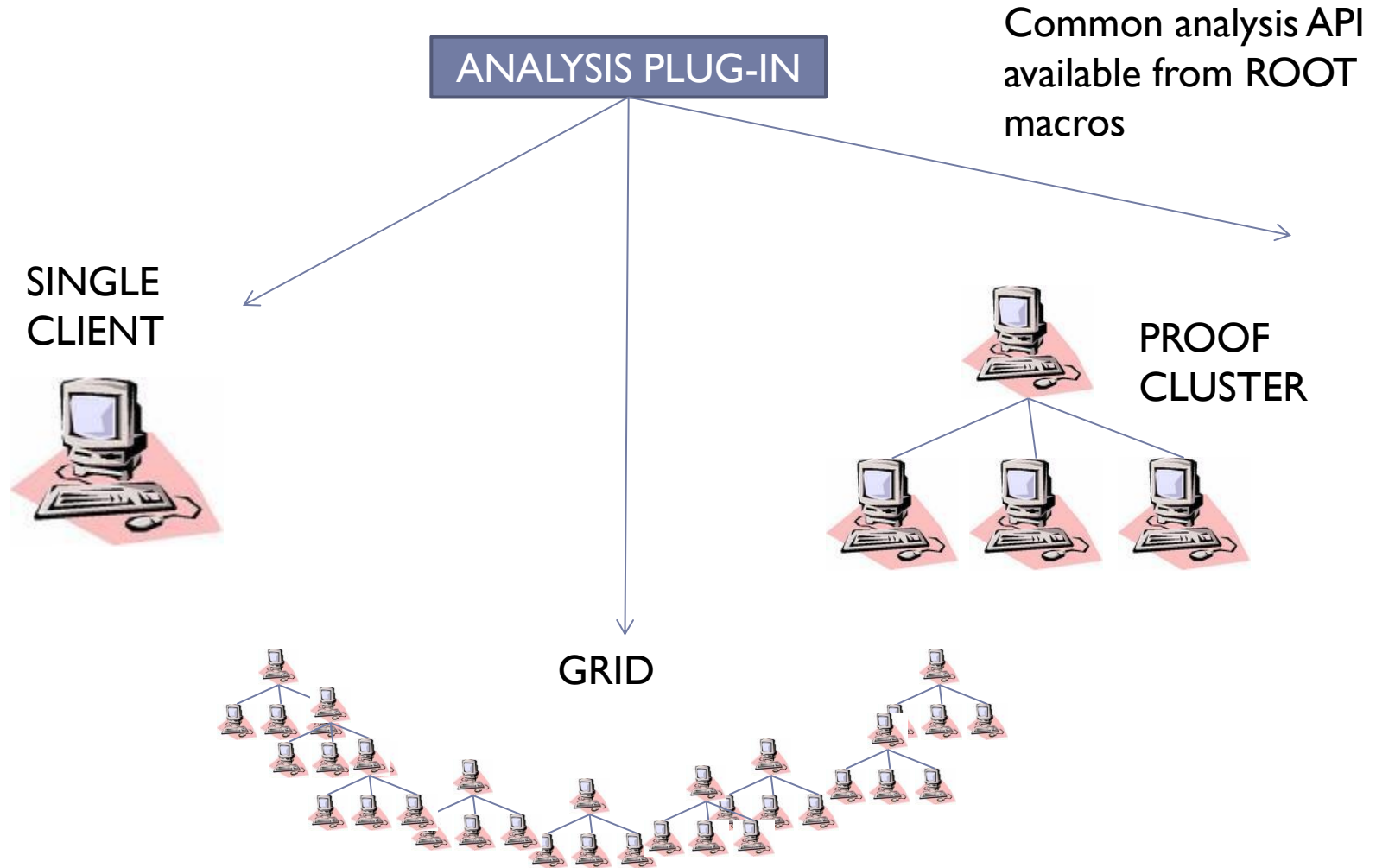
Can also read only Kinematics (no need for ESDs),
without ESDs change one line in steering macro:

```
chain = CreateChain("TE",galice_root_list,2);
```

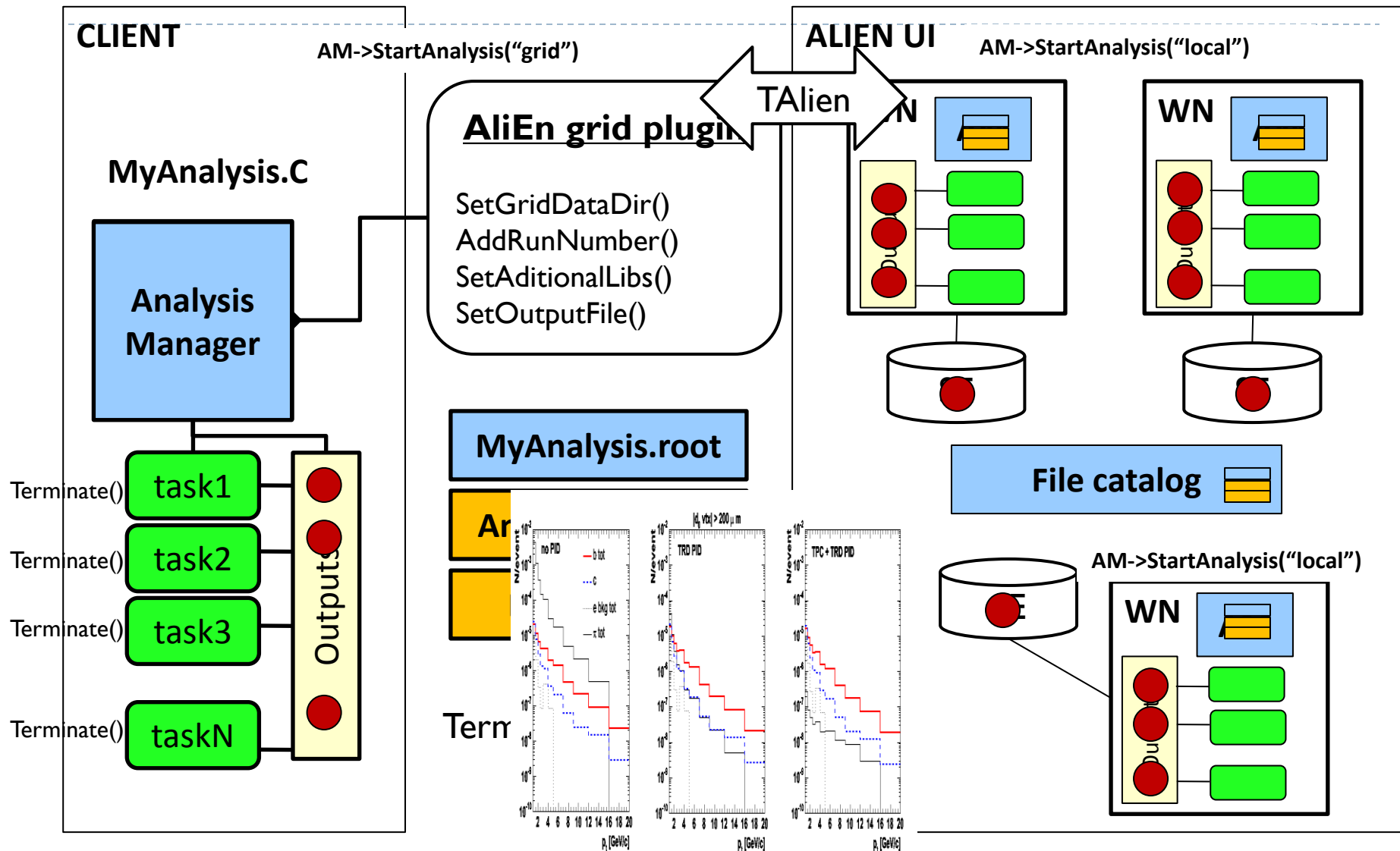
A helper for AliEn analysis

- ▶ Works as a plugin for the analysis manager (as event handlers)
 - ▶ One has to create and configure a `AliAnalysisAlien` object
 - ▶ See: <http://aliceinfo.cern.ch/Offline/Activities/Analysis/AnalysisFramework/AlienPlugin.html>
- ▶ Creates dataset, JDL, analysis macro, execution and validation scripts
- ▶ Submits your job and merges the results

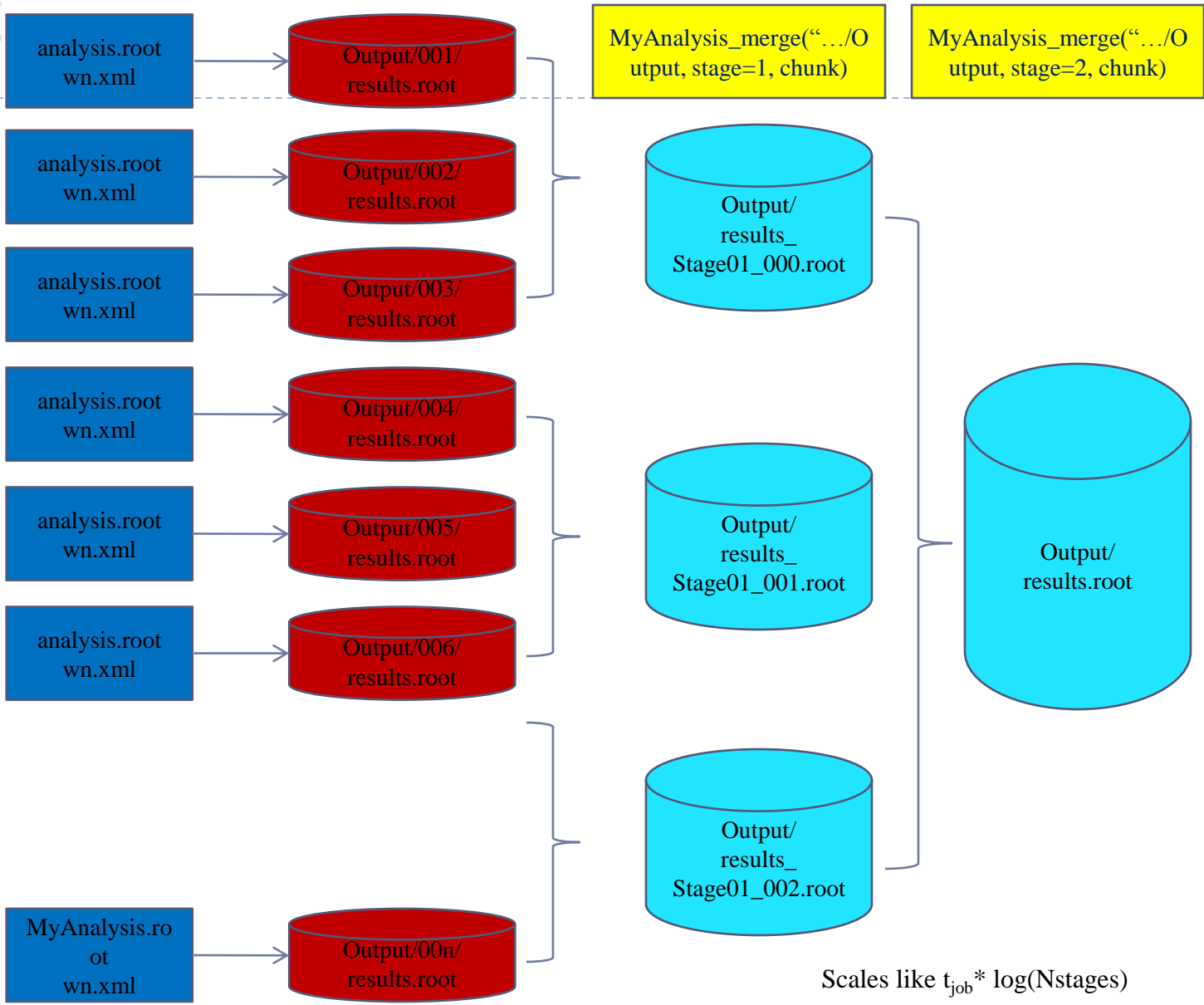
The analysis plug-in



GRID analysis via plugin



Merging the results



- 2 parameters: number of stages and maximum number of chunks to be merged per job
- Stages can be skipped if reduction is enough
- For production jobs: max_stages=5, split=20 (less for AOD merging)

Scales like $t_{job} * \log(N_{stages})$

Important plug-in settings

- ▶ ***plugin->SetRunMode(const char *mode)***
 - ▶ “full”: generate files, copy in grid, submit, merge
 - ▶ “offline”: generate files, user can change them
 - ▶ “submit”: copy files in grid, submit, merge
 - ▶ “terminate”: merge available results
 - ▶ “test”: generate files + a small dataset, run locally as a remote job
 - ▶ `plugin->SetNtestFiles(Int_t nfiles)` – default 1
- ▶ ***plugin->SetROOTVersion(const char *rootver)***
- ▶ ***plugin->SetAliRootVersion(const char *alrootver)***
- ▶ Change root and AliRoot versions when needed, according to the output of `packages` AliEn command.

Describing the input data

- ▶ ***plugin->SetGridDataDir(const char *datadir)***
 - ▶ Put here the alien path before run numbers
 - ▶ See pcalimonitor.cern.ch for relevant data paths
- ▶ ***plugin->SetDataPattern(const char *pattern)***
 - ▶ Use uniquely identifying patterns
 - ▶ i.e. ****/pass3*/AliESDs.root***
 - ▶ Plugin supports making datasets on ESD, ESD tags or AOD
- ▶ ***plugin->SetRunRange(Int_t min, Int_t max)***
 - ▶ Sets the run range to be analyzed
 - ▶ Enumeration of run numbers allowed
 - ▶ For existing data collections, use **AddDataFile()**
- ▶ ***Plugin->SetRunPrefix("000")***
 - ▶ To be used for real data

Describing the output

- ▶ **`plugin->SetGridOutputDir(const char *dir)`**
 - ▶ Can be absolute AliEn FC path (/alien/cern.ch/...) or relative to work directory (no slashes)
- ▶ **`plugin->SetOutputFiles("file1 file2 ...");`**
 - ▶ Allows a selection of files among the analysis outputs
- ▶ **`plugin->SetDefaultOutputs()`**
 - ▶ Enables all outputs of the tasks connected to the analysis manager
- ▶ **`plugin->SetOutputArchive("log_archive.zip:stderr,stdout root_archive.zip:*.root@disk=2");`**
 - ▶ Will save the standard output/error in a zip and all root files in another zip replicated in 2 storage elements
 - ▶ **Note:** If archiving the output you may want to omit declaring the output files

Other settings

- ▶ Using par files
 - ▶ *plugin->EnablePackage("package.par")*
- ▶ Using other external libraries available in AliEn
 - ▶ *plugin->AddExternalPackage("fastjet::v2.4.0")*
- ▶ Compiling single source files
 - ▶ *plugin->SetAnalysisSource("mySource.cxx")*
 - ▶ But files have to be uploaded to AliEn from current directory
 - ▶ *plugin->SetAdditionalLibs("libJETAN.so mySource.cxx mySource.h")*
 - ▶ Extra libraries to be loaded (besides AF ones) have to be enumerated in the same method.

Optional settings

- ▶ Number of files per job
 - ▶ *plugin->SetSplitMaxInputFileNumber(Int_t n);*
- ▶ Number of runs per master job
 - ▶ *plugin->SetNrunsPerMaster(Int_t n)*
- ▶ Number of files to merge in a chunk
 - ▶ *plugin->SetMaxMergeFiles(Int_t n)*
- ▶ Resubmit threshold
 - ▶ *plugin->SetMasterResubmitThreshold(Int_t percentage)*
- ▶ Process a single run per job and output to a single directory
 - ▶ *plugin->SetOutputSingleFolder(const char *folder)*

PROOF & ALICE Analysis Facilities

Terminology

▶ Client

- Your machine running a ROOT session that is connected to a PROOF master

▶ Master

- PROOF machine coordinating work between slaves

▶ Slave/Worker

- PROOF machine that processes data

▶ Query

- A job submitted from the client to the PROOF system. A query consists of a selector and a chain

▶ Selector

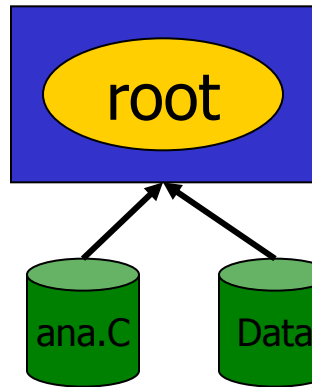
- A class containing the analysis code.
- In ALICE we use the Analysis Framework, therefore a AliAnalysisTaskSE is sufficient

▶ Chain

- A list of files (trees) to process (more details later)

How does PROOF analysis work?

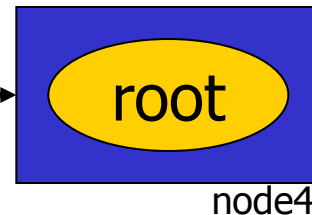
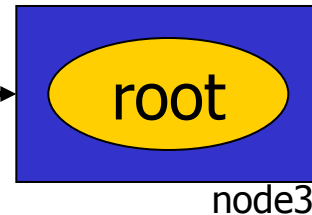
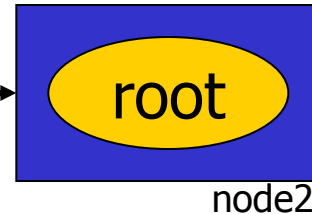
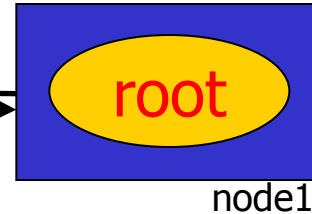
Client –
Local PC



stdout/result

ana.C

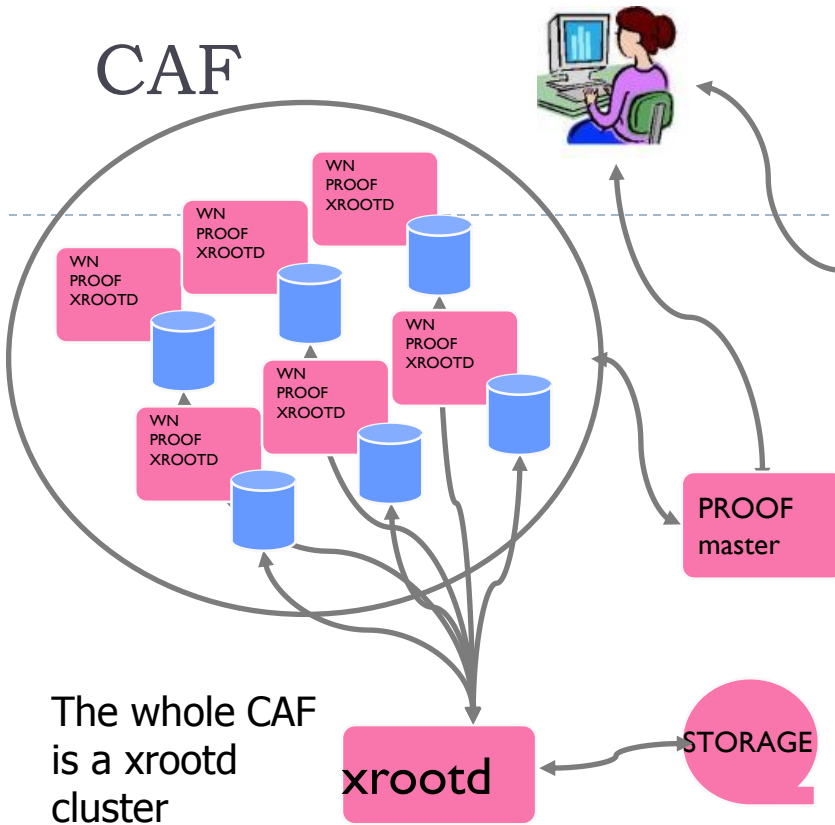
Remote PROOF Cluster



Proof master
Proof slave

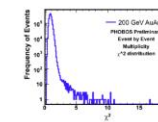


CAF

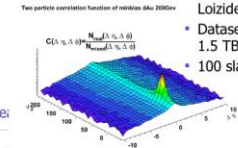


lfn	guid	{se's}
lfn	guid	{se's}
lfn	guid	{se's}
lfn	guid	{se's}
lfn	guid	{se's}

Rare high multiplicity event set



- Burak Al
- Dataset: 11k files, 4.5 TB
- 150 slaves, ~1 hour



- Wei Li, Constantin Loizides
- Dataset: 4.5k files, 1.5 TB
- 100 slaves, 75 min

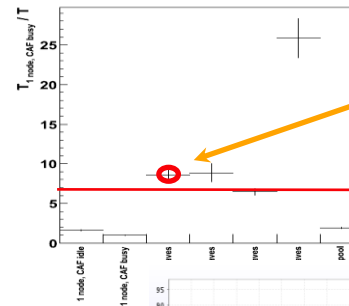
The whole CAF is a xrootd cluster

xrootd

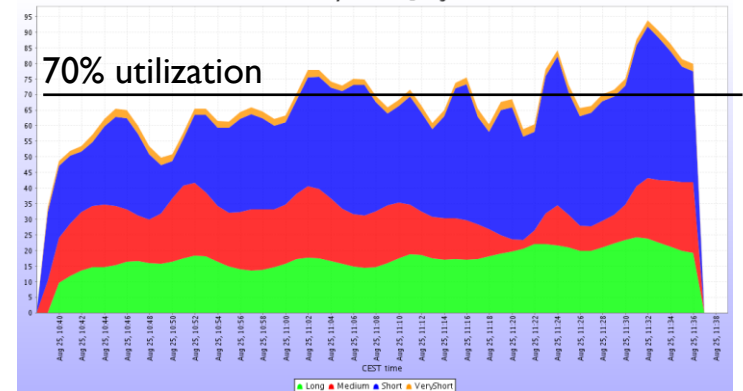
STORAGE

- Powerful and fast machinery – very popular with users
- Allows for any use pattern, however quite often leading to contention for resources

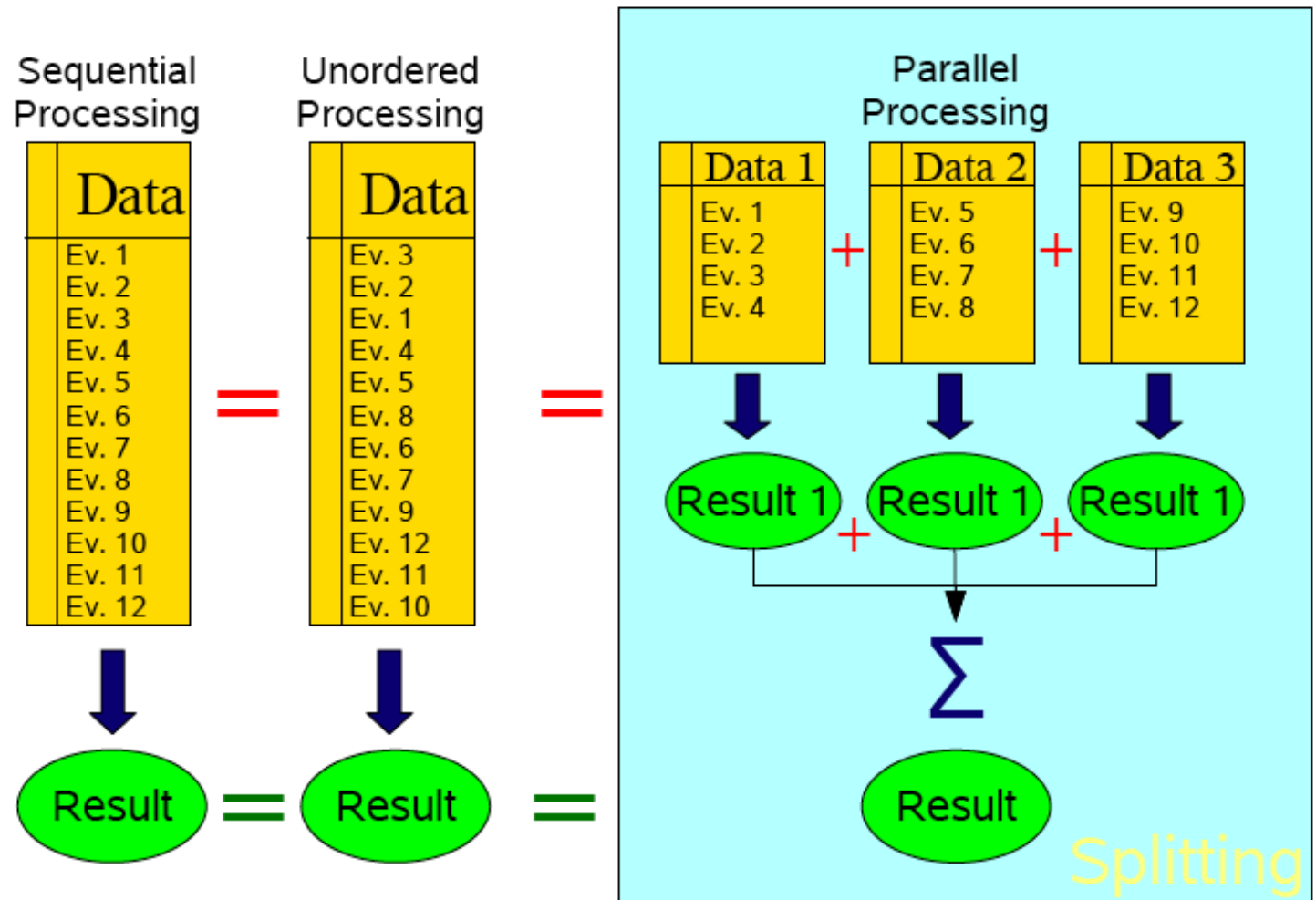
Query Short in different environments



History of cluster usage



Event based (trivial) Parallelism



How to use PROOF

- ▶ **The analysis framework is used**
 - ▶ Files to be analyzed are put into a chain → TChain.
 - ▶ Analysis written as a task (already introduced in previous tutorial) → AliAnalysisTaskSE
 - ▶ The same analysis like written previously can be used
- ▶ **If additional libraries are needed, these have to be distributed as a "package" (PAR: PRoot Archive)**



Packages

- ▶ **PAR files: PROOF ARchive. Like Java jar**
 - ▶ Gzipped tar file
 - ▶ PROOF-INF directory
 - ▶ BUILD.sh, building the package, executed per slave
 - ▶ SETUP.C, set environment, load libraries, executed per slave
- ▶ **API to manage and activate packages**
 - ▶ UploadPackage("package")
 - ▶ EnablePackage("package")

PROOF datasets

- ▶ A dataset represents a list of files (e.g. physics run X)
 - ▶ Correspondence between AliEn collection and PROOF dataset
- ▶ Users register datasets
 - ▶ The files contained in a dataset are automatically staged from AliEn (and kept available)
 - ▶ Datasets are used for processing with PROOF
 - ▶ Contain all relevant information to start processing (location of files, abstract description of content of files)
- ▶ Datasets are public for reading, common datasets are available (for data of common interest)
- ▶ In practice
 - ▶ Upload to PROOF cluster
 - ▶ `gProof->RegisterDataSet("myDataSet", proofColl);`
 - ▶ Check status
 - ▶ `gProof->ShowDataSets();`
 - ▶ <http://aaf.cern.ch> -> Datasets -> CAF

Some useful tips

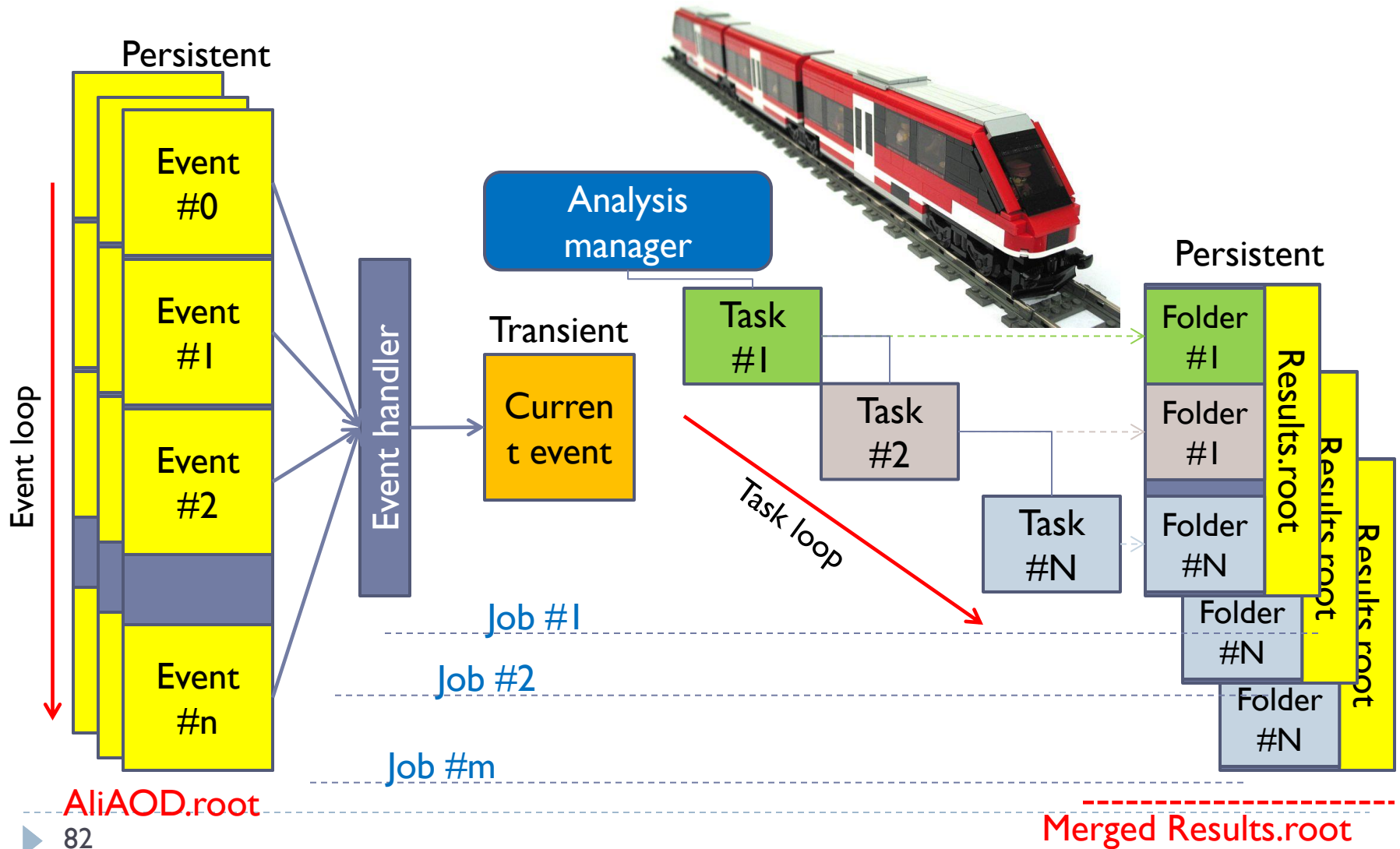
- ▶ When your task crashes
- ▶ You can access the output of the last query by clicking on the “Show Log” button in the PROOF progress window
- ▶ You can retrieve the output from any previous query
 - ▶ Open ROOT
 - ▶ Get a PROOF manager object
`mgr = TProof::Mgr("alice-caf")`
 - ▶ Get the log files from the last session
`logs = mgr->GetSessionLogs(0) // 0=last query`
 - ▶ Display them
`logs->Display()`
 - ▶ Search for a special word (e.g. segmentation violation)
`logs->Grep("segmentation violation")`
 - ▶ Save them to a file
`logs->Save("*", "logs.txt")`
- ▶ Resetting environment
 - ▶ `TProof::Reset("alicecaf")`
 - ▶ `TProof::Reset("alicecaf", kTRUE)`
- ▶ Compile with debug
 - ▶ `Load("<task>+g")`

LEGO Trains

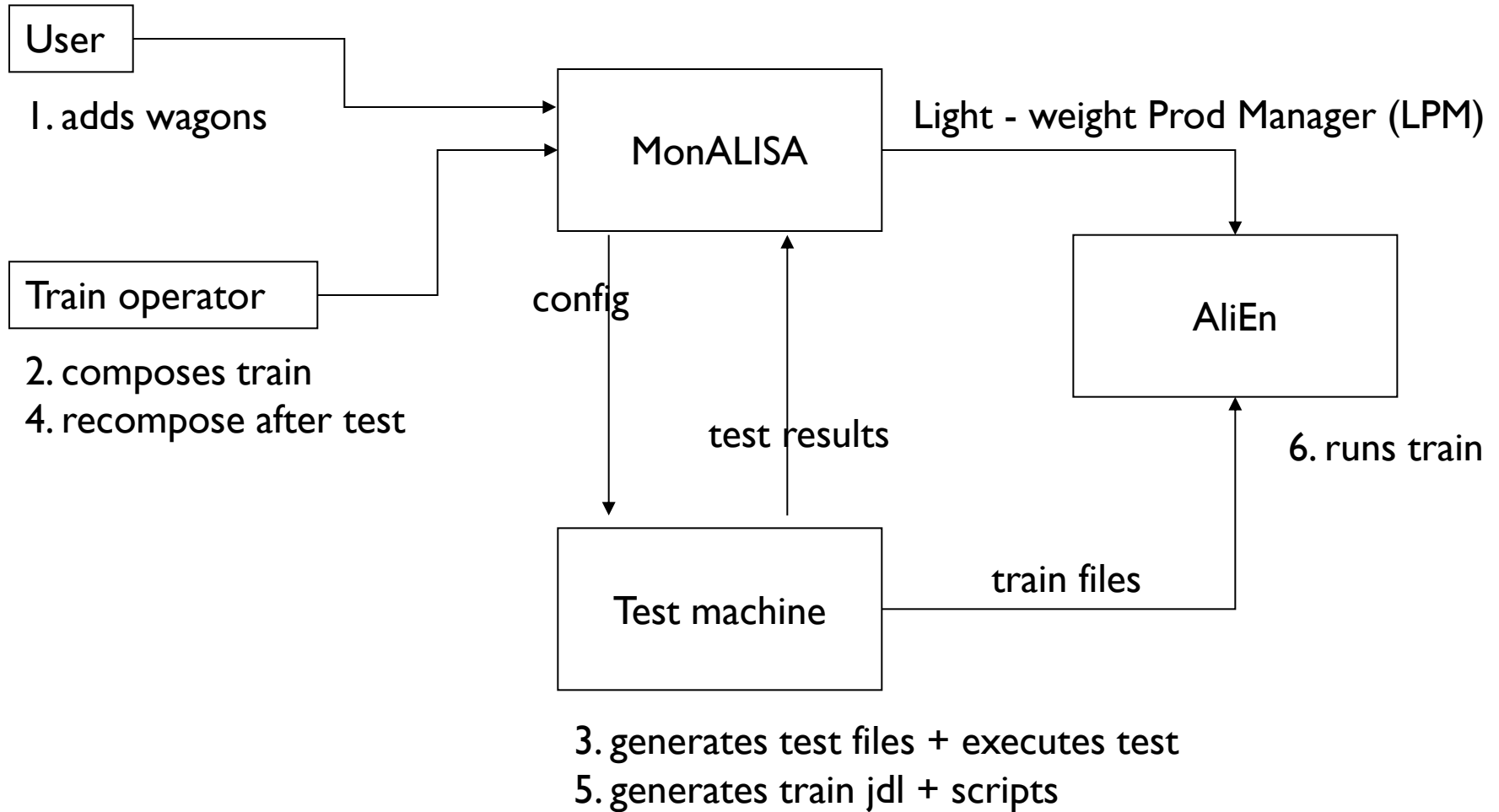
Organized analysis with LEGO trains

- ▶ Web (MonALISA) based tool
- ▶ Allows running many analyses for the same data → I/O reduction
- ▶ CPU efficiency overall behaves like the best component
- ▶ Better automatic testing of components and control of memory or “bad practices”
- ▶ [Extensive documentation](#)

The ALICE analysis train



Workflow




Organized analysis

Name Devel_1 ([train temporary file dir](#))

PWG 1

Description Development testing train #1


Handlers



Name	Macro path (parameters)	Body	Actions
AOD handler	ANALYSIS/macros/train/AddAODHandler.C (...)	handler->Dump();	Add new handler >

Handler configuration

Wagons



Name	Owner	Macro path (parameters)	Libraries	Dependencies	Enabled	Last run	Actions
PhiCorrelations	grigoras	PWG4/macros /AddTaskPhiCorrelations.C (...)	CORRFW,EMCALUtils,JETAN,PWG4JetTasks			8	
PhiCorrelationsQA	grigoras	PWG4/macros /AddTaskPhiCorrelationsQA.C (...)				2	
test wagon 2	grigoras	something (param)	l1.so	PhiCorrelations,PhiCorrelationsQA			

[Add new wagon >](#)

Wagon configuration

Periods




Period name	Reference production	Run list	Description	Last analyzed	Actions
LHC10h(2)	LHC10h(2)		LHC10h - pass2, the ...	2	
AODs_73	FILTER_Pb-Pb_073_LHC10h			8	
AODs	FILTER_Pb-Pb_049_LHC10h_Stage3	136854, 139513, 139514, 139517		3	

[Add new period >](#)

Data configuration

Runs



Run ID	AliRoot version	Testing status	Run status	Actions
8	VO_ALICE@AliRoot::v5-02-05-AN	Finished (1:47 total time)		
7	VO_ALICE@AliRoot::v5-02-04-AN			
6	VO_ALICE@AliRoot::v5-02-04-AN			
5	VO_ALICE@AliRoot::v5-02-04-AN	Finished (0m 43s total time)		
4	VO_ALICE@AliRoot::v5-02-04-AN	Finished (1d 19:40 total time)		

Testing and running status

Additional Links

- ▶ [ALICE collaboration page](#) with a lot of information and links
- ▶ Some places where you can find new generators or software that is not yet in AliRoot
 - ▶ [Generators@CERN](#)
 - ▶ [Software@hepforge](#)
- ▶ If you want to try some theoretical calculations
 - ▶ [CalcHEP](#) - a package for calculation of Feynman diagrams and integration over multi-particle phase space + references to other software
 - ▶ Symbolic calculations in C++ [GiNaC](#)
- ▶ [The Durham HepData Project](#) – i.e. try the search “re Pb Pb”

Conclusions

- ▶ This introduction does not replace the detailed offline tutorial
- ▶ The system may look complex but it is based on simple ideas and you definitely will be able to use it
- ▶ Probably the most important for you will be the analysis framework: the easiest way to start with it is to use an existing analysis class or example (we have many)
- ▶ I wish you good luck and many interesting results!

Thank you very much for your attention!
