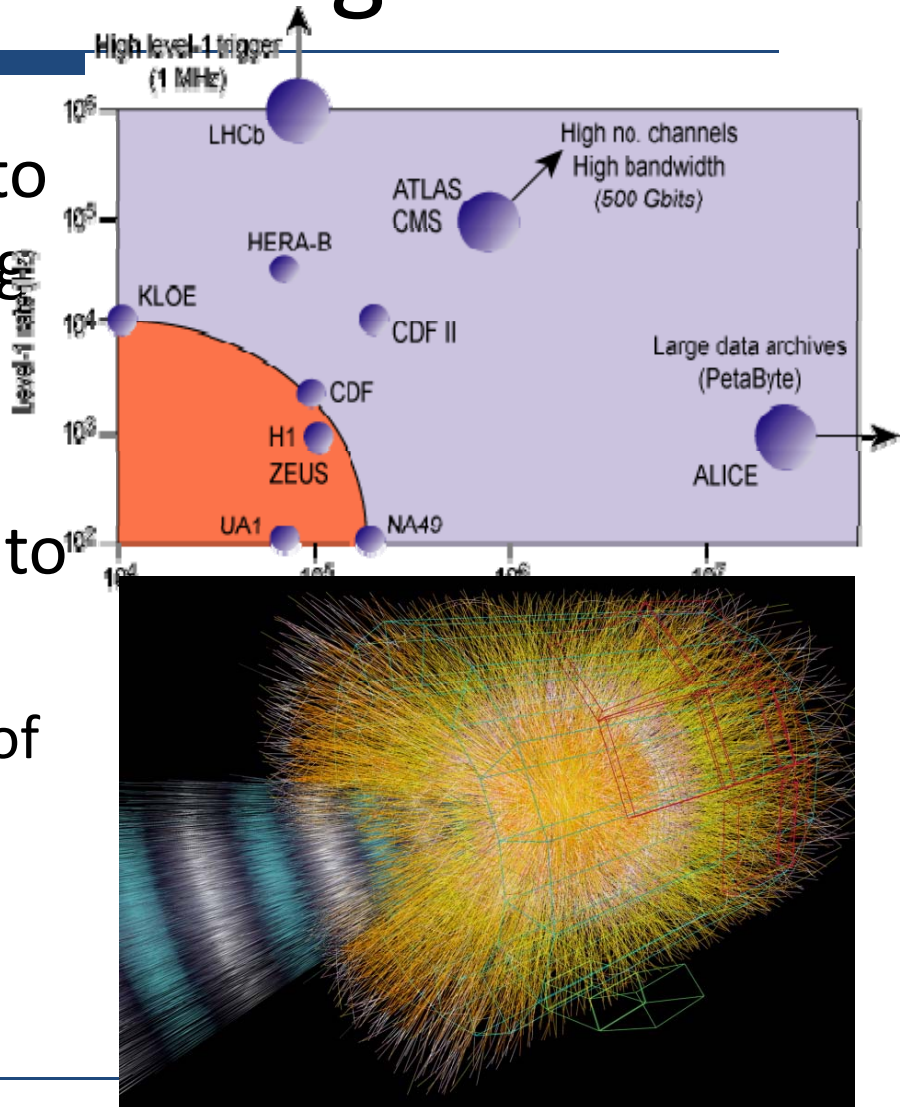
The ALICE Analysis Framework

A.Gheata for ALICE Offline Collaboration

ALICE Data Challenge

- ALICE will be producing up to 1.25 Gbyte/s of data, storing more than 1PByte/year
- We will have to deal with events ranging from kBytes to ~1Gbyte
  - Multiplicities up to few tens of thousand tracks



The ALICE GRID Map

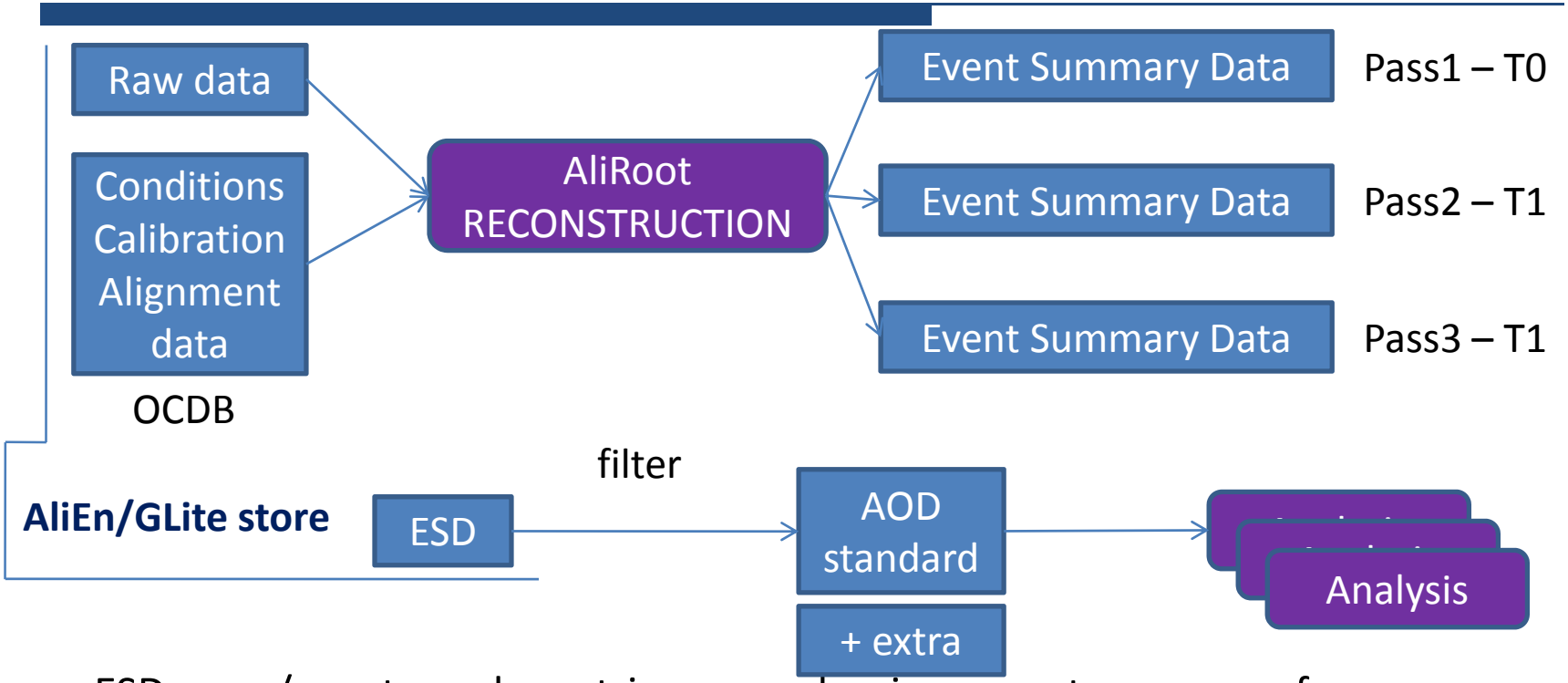
- Data to be analyzed spread over widely distributed resources

Live picture here

The Computing Model

- T0: First pass reconstruction, storage of RAW, calibration data and first-pass ESD's
  - Central PROOF cluster for pilot analysis, fast feedback calibration and reconstruction
- T1: Subsequent reconstructions and scheduled analysis, storage of a collective copy of RAW, disk replicas of ESD's and AOD's
- T2: Simulation and end-user analysis, disk replicas of ESD's and AOD's
- For analysis, hybrid flavors of PROOF/batch computing resources pop-up as the SW we are using takes more and more advantage of multi-core architectures.

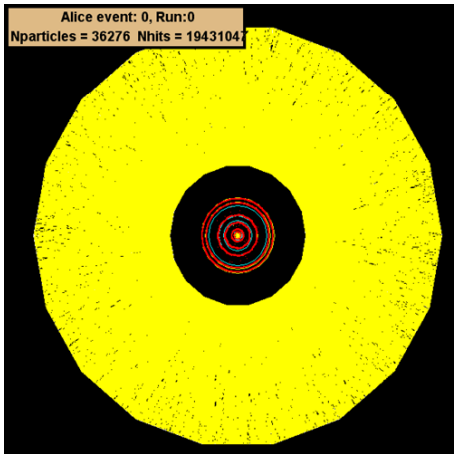
The Data Flow

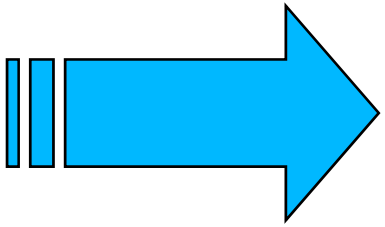


- ESD – run/event numbers, trigger word, primary vertex, arrays of tracks/vertices, detector info
- AOD standard – cleaned-up ESD's, reducing the size O(10) times
  - Can be extended on user demand with extra information
- ESD and AOD inheriting from the same base class (keep same event interface)

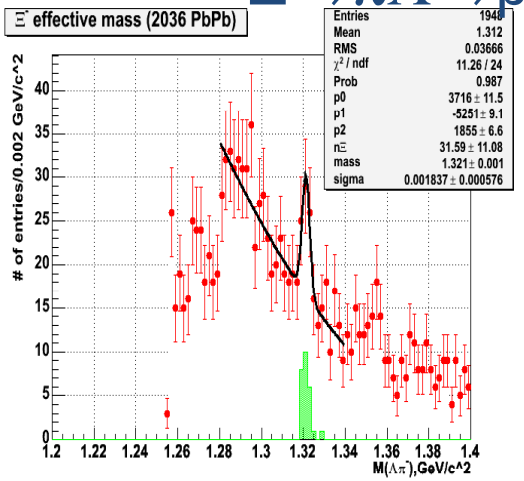
Analysis models

- Three main analysis models
  - Prompt data processing (calibration, alignment, reconstruction, analysis) @CERN with PROOF
  - Pilot analysis with local PROOF clusters for fast feedback
  - Batch Analysis on the GRID infrastructure





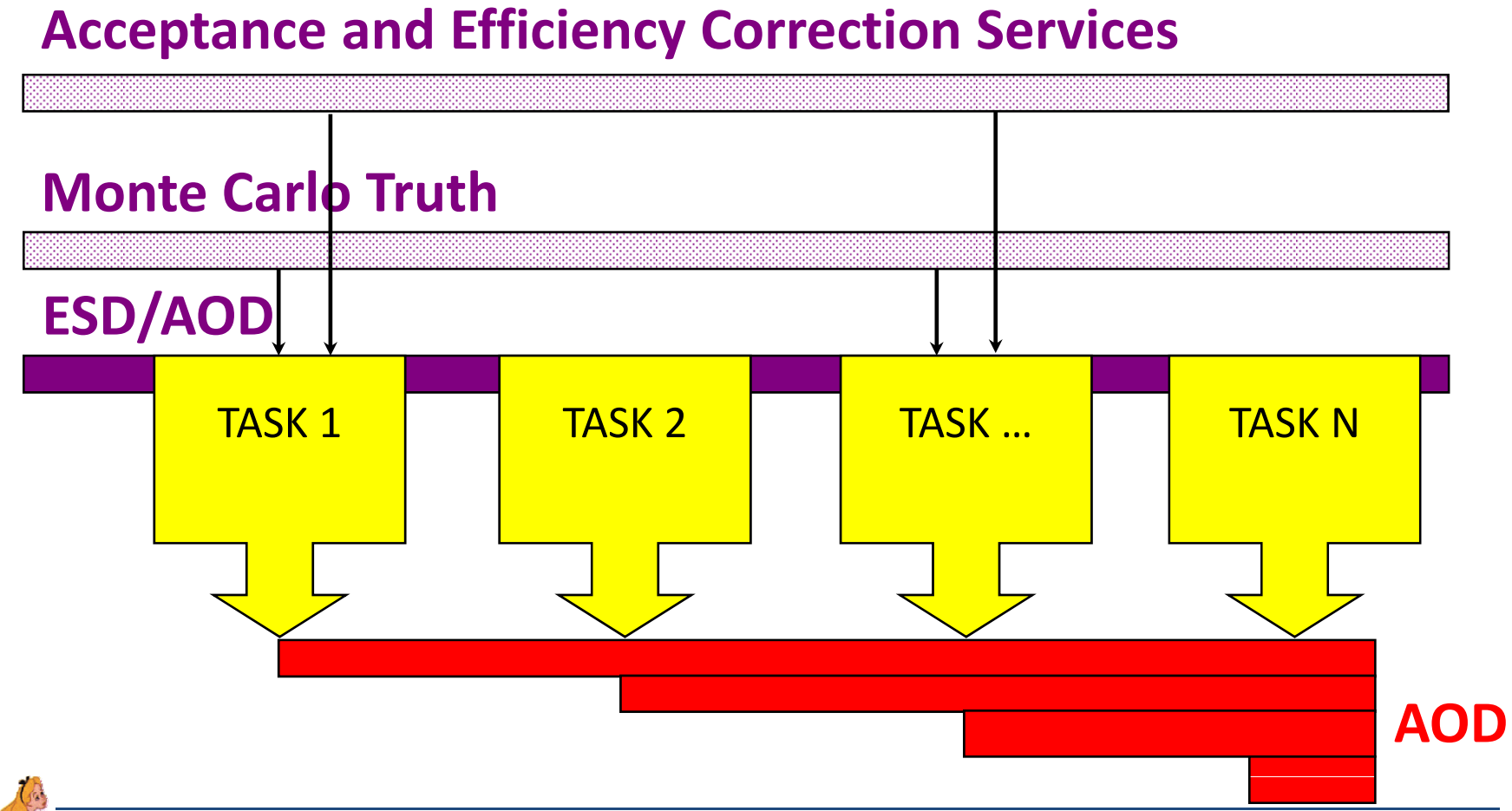
$\Xi \rightarrow \pi\Lambda \rightarrow p\pi$



Scheduling the analysis

- Makes always sense when resources are limited
  - Accessing the full data set is a stringent exercise
  - There are a lot of analysis clients to be served
- One needs to maximize the profit of having data in memory and optimize CPU/IO ratio
  - Sharing the event loop for several clients ? Parallelizing ?
- Formalizing event structure is not enough, one needs to formalize also data access patterns → analysis framework. This comes with some bonuses:
  - 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

Scheduled analysis producing AOD`s

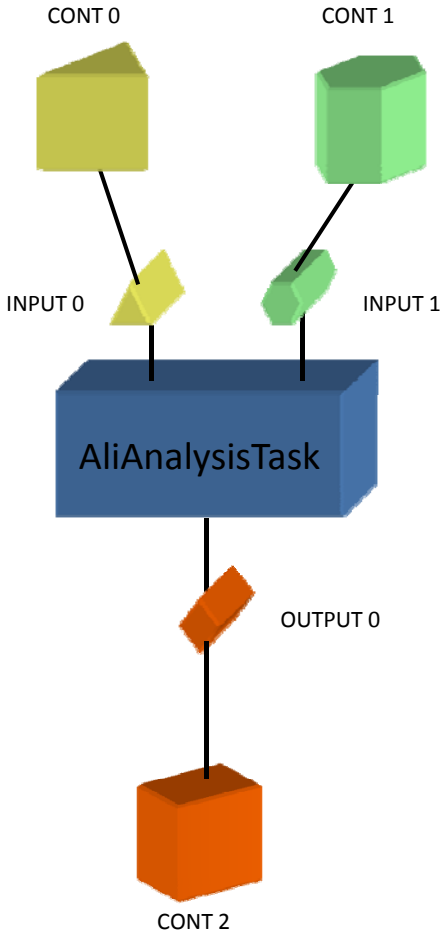


What the analysis framework does in ALICE

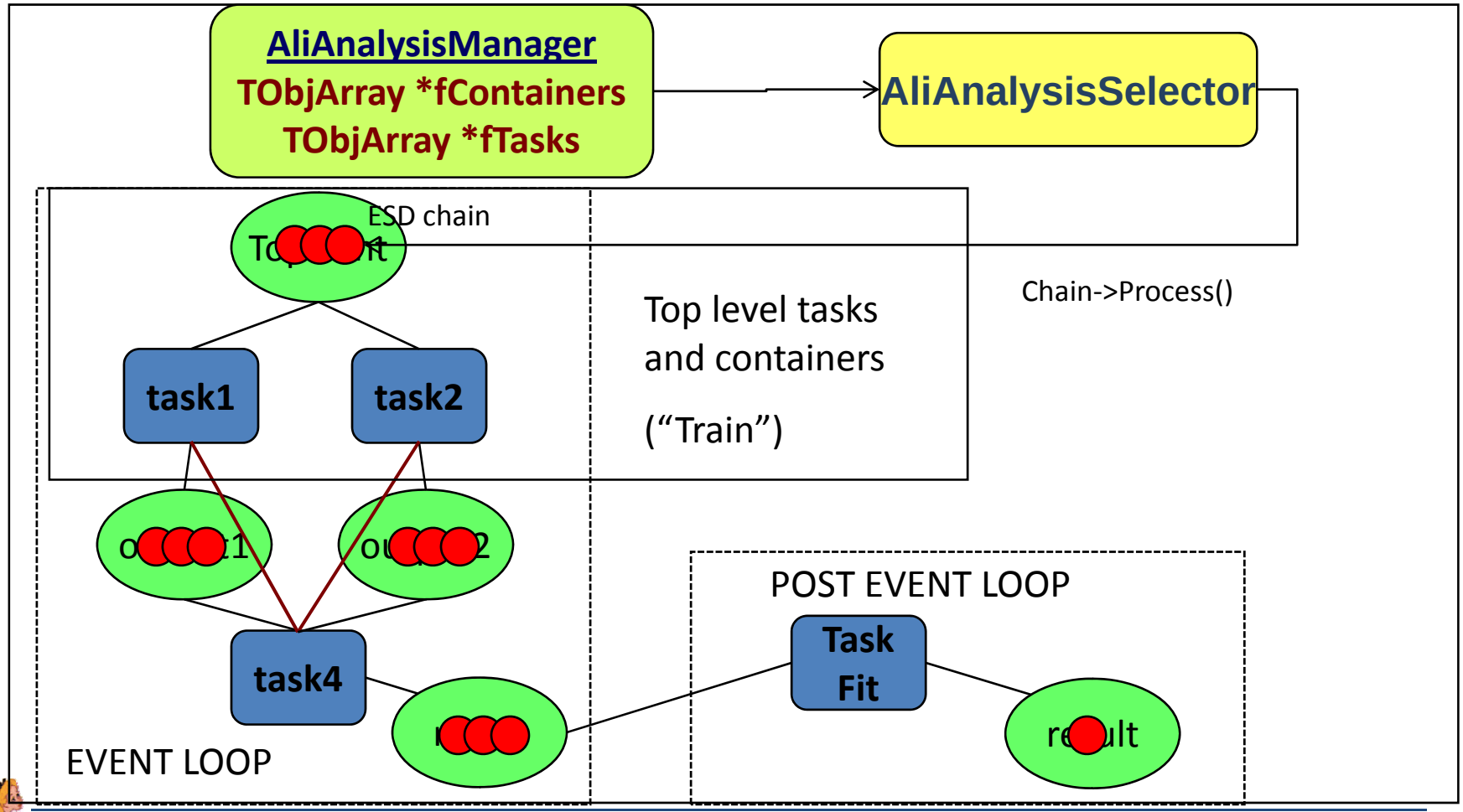
- Transparent access to all resources with the same code
  - Usage: Local, AliEn grid, CAF/PROOF
- Transparent access to different input data
  - ESD, AOD, MC truth
- Allow sharing resources by multiple analysis modules
  - Accomodate different analysis in the same session (analysis trains)
- Provides the formalism for generic inter-connectable „analysis tasks“ that share a single event loop
  - Very general design, not bound to ALICE software

AliAnalysis… framework

- Data-oriented model composed of independent tasks
  - Task execution triggered by data readiness
  - Tasks can be inter-connected via data containers with matching type (graph-like)
  - Tasks have to implement virtual methods that are called in different processing stages
- Event loop steered by ROOT TSelector functionality
  - Special embedded loops used in event mixing also supported
- Tasks are owned by a manager class (AliAnalysisManager)
  - Used to hide computing scheme dependent code and steer event processing in the task graph



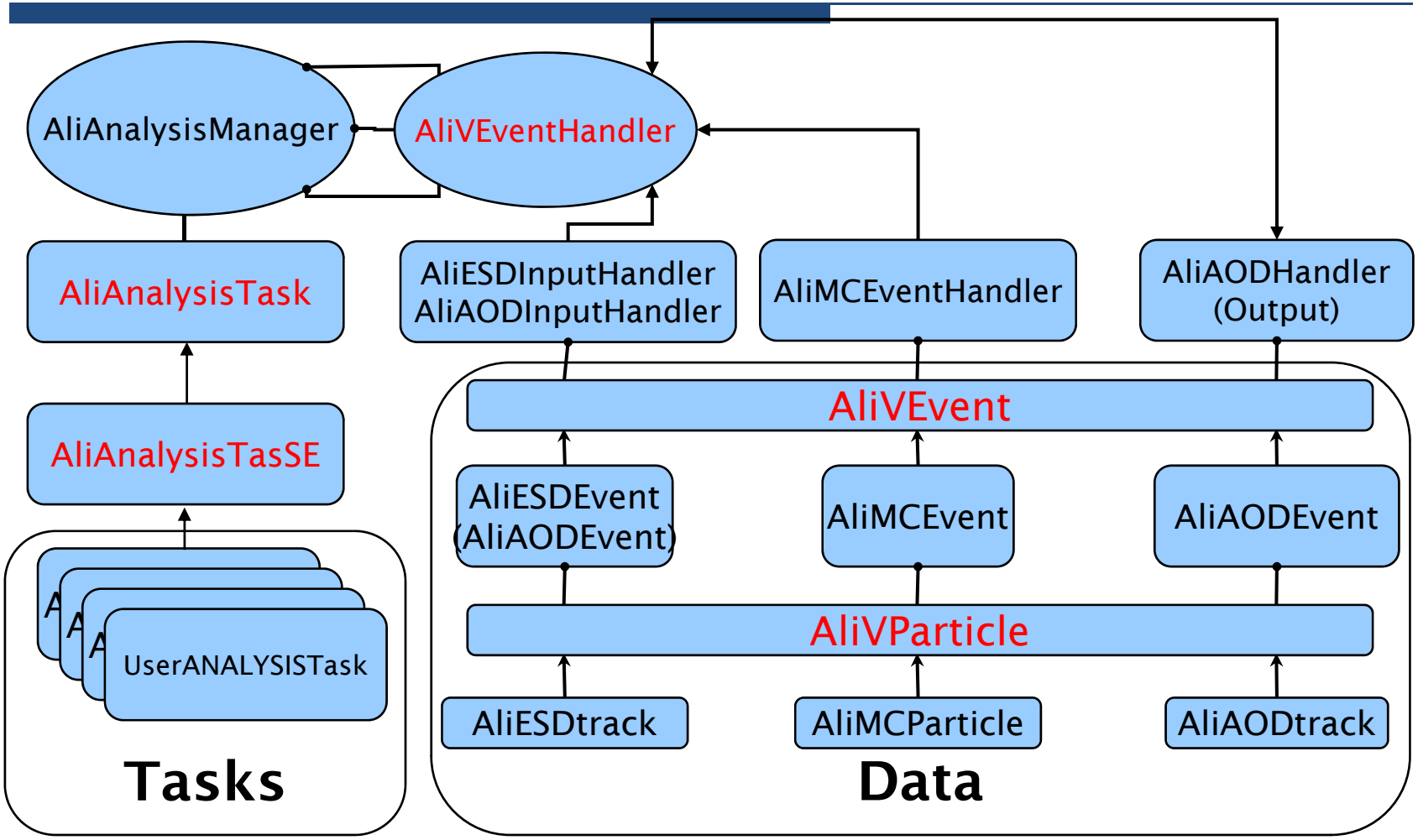
How it works


AliAnalysisManager
TObjArray *fContainers
TObjArray *fTasks
AliAnalysisSelector
ESD chain
Chain->Process()
Top level tasks
and containers
("Train")
task1
task2
task4
POST EVENT LOOP
Task
Fit
EVENT LOOP

The user task structure

- CreateOutputObjects (called once per session on the worker node)
  - Self describing, allows booking histograms and trees
- ConnectInputData (called at any change of input tree)
  - Allows connecting to the current event via analysis manager
- Exec (called for every event)
  - Allows processing current event and publishing the output data
- Terminate (called once on the client)
  - Allows finalizing the analysis (e.g. draw/fit histograms)

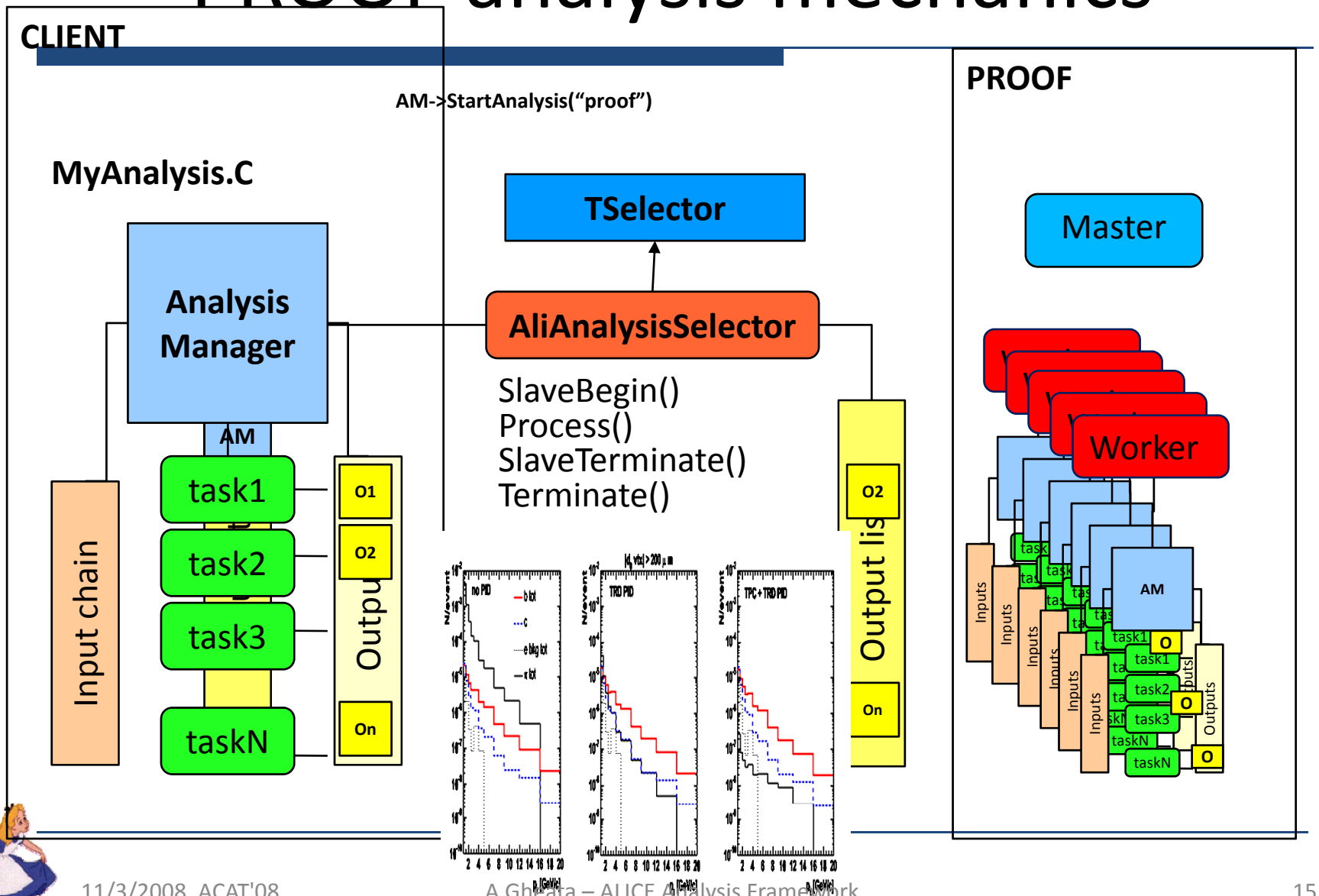
The overall picture



PROOF Analysis

- PROOF stands for Parallel ROOT Facility
  - Using trivial event based parallelism
  - Forces user implementation of a TSelector – derived class
- The framework comes with a special selector streaming a full analysis session to the PROOF cluster
  - Completely transparent for users
- The same local analysis can be run in PROOF with minor changes
  - Connect to PROOF cluster and upload/enable the user code
  - StartAnalysis(“proof”)
- This mode is in production in ALICE CAF (CERN Analysis Facility)
  - Very good scalability for reasonable cluster load
  - Heavily used by ALICE users due to the fast response

PROOF analysis mechanics



GRID Analysis

- Submitting batch analysis in GRID is somehow more difficult because user has to:
  - Create dataset(s), write fully customized JDL, write executable and validation scripts, copy all dependency files in AliEn FC, handle merging …
- An AliEn plugin for ALICE analysis framework was developed to:
  - Keep user at ROOT prompt, allowing straightforward customization via API
  - Automate all interactions with AliEn, generate all needed files, submit the job and collect the results.
  - Everything done via AliEn API using ROOT TGrid interface

AliEn analysis plugin

PROOF SETUP

TProof::Open("user@lxb6046")
gProof->UploadPackage("pack.par")
gProof->EnablePackage("pack")
....

+

MY MACHINE

MyAnalysis.C

+

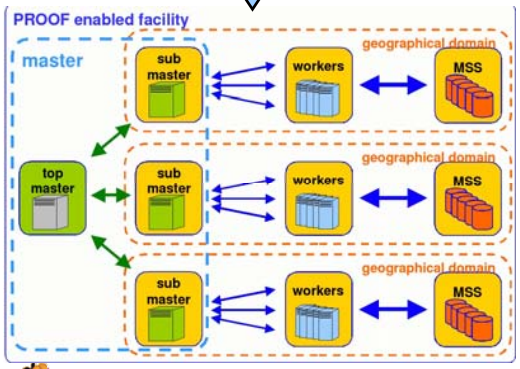
AliEn SETUP

CREATE + CONFIGURE GRID PLUGIN

StartAnalysis("proof")

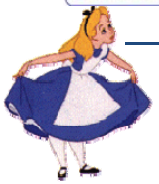
StartAnalysis("local")

StartAnalysis("grid")



MyResults.root

AliEn 2 @GRID



AliEn plugin configuration

```
//  source /tmp/gclient_env_$UID in the current shell.
  if (!AliAnalysisGrid::CreateToken()) return NULL;
  AliAnalysisAlien *ana = new AliAnalysisAlien();
// Set the run mode (can be "full", "test", "offline", "submit" or "terminate")
  ana->SetRunMode("full");  // VERY IMPORTANT - DECRIBED BELOW
// Set versions of used packages
  ana->SetAPIVersion("V2.4");
  ana->SetROOTVersion("v5-21-01-alice");
  ana->SetAliROOTVersion("v4-15-Rev-05");
// Declare input data to be processed.
// Method 1: Create automatically XML collections using alien 'find' command.
// Define production directory LFN
  ana->SetGridDataDir("/alice/sim/PDC_08/LHC08b1/");
// Set data search pattern
  ana->SetDataPattern("*tag.root");
// ...then add run numbers to be considered
  ana->AddRunNumber(300000);
  ana->AddRunNumber(300001);
// Method 2: Declare existing data files (raw collections, xml collections, root file)
// If no path mentioned data is supposed to be in the work directory (see SetGridWorkingDir())
// XML collections added via this method can be combined with the first method if
// the content is compatible (using or not tags)
//  ana->AddDataFile("tag.xml");
//  ana->AddDataFile("/alice/data/2008/LHC08c/000057657/raw/Run57657.Merged.RAW.tag.root");
// Define alien work directory where all files will be copied. Relative to alien $HOME.
  ana->SetGridWorkingDir("work");
// Declare alien output directory. Relative to working directory.
  ana->SetGridOutputDir("output"); // In this case will be $HOME/work/output
// Declare the analysis source files names separated by blancs. To be compiled runtime
// using ACLiC on the worker nodes.
  ana->SetAnalysisSource("AliAnalysisTaskPt.cxx");
// Declare all libraries (other than the default ones for the framework. These will be
// loaded by the generated analysis macro. Add all extra files (task .cxx/.h) here.
  ana->SetAdditionalLibs("AliAnalysisTaskPt.h AliAnalysisTaskPt.cxx");
// Declare the output file names separated by blancs.
// (can be like: file.root or file.root@ALICE::Niham::File)
  ana->SetOutputFiles("Pt.ESD.1.root");
// Optionally define the files to be archived.
//  ana->SetOutputArchive("log_archive.zip:stdout,stderr@ALICE::NIHAM::File root_archive.zip:*.root@A
```

“Official” tasks tested in the train

- Tested on CAF an AliEn data for all “official” new tasks
- Regular runs of the train in GRID will be scheduled
- Testing/running new tasks in GRID will be much easier with the new plugin

PWG0	AlidNdEtaTask	First physics	OK	OK	OK
PWG0	AlidNdEtaCorrectionTask	First physics	OK	OK	OK
PWG0	AliMultiplicityTask	First physics	OK	OK	OK
PWG1		Reconstruction	N/A	N/A	N/A
PWG2	AliAnalysisTaskProtons	p/pbar analysis	OK	OK	OK
PWG2	AliAnalysisTaskFemto	Femtoscopy	OK	OK	OK
PWG2	AliAnalysisTaskCheckV0	V0 check	OK	OK	OK
PWG2	AliAnalysisTaskESDDedx	Dedx analysis	?	?	?
PWG2	AliAnalysisTaskStrange	Strangeness	OK	OK	OK
PWG2	AliAnalysisTaskESDStrangeMC	Strangeness on MC	?	?	?
PWG2	AliAnalysisTaskScalarProduct	Flow analysis	OK	OK	OK
PWG3	AliAnalysisTaskMuonAODfromGeneral		?	?	?
PWG3	AliAnalysisTaskSingleMu		?	?	?
PWG3	AliAnalysisTaskSEVertexingHF	HF vertexing	OK	OK	OK
PWG3	AliAnalysisTaskESDMuonFilter	MUON filtering	OK	OK	OK
PWG4	AliAnalysisTaskParticleCorrelation	Particle correlations	OK	OK	OK
PWG4	AliAnalysisTaskJets	JETAN	OK	OK	OK
PWG4	AliAnaPi0	Pi0 analysis	OK	OK	OK

Experience so far

- The framework was developed during the last 2 years and fully adopted by ALICE users
  - A lot of effort done by several people which now pays back
    - Very stable and bug-free in all modes
    - Extremely handy to process widely distributed data quite fast
  - Design goals met
    - Still there are new features developed very recently
- Good user feedback via the alice-analysis-task-force mailing list
- Very good CAF experience
  - 5-10 concurrent CAF users daily
  - Triggered fixes and new developments in PROOF also

Some numbers

- Timing for several analysis tasks submitted separately versus a single train via the framework
  - Batch cluster at GSI
  - No GRID latencies

Speed Results

Different task numbering

task	real time [sec]	CPU time [sec]	fraction
task1	1564	706	0.451
task2	1627	781	0.480
task3	2339	1505	0.643
task4	2761	1999	0.724
task5	1596	727	0.456
task6	2361	1474	0.624
task7	2796	1985	0.710
task8	3029	2203	0.727
task9	1587	726	0.457
task10	1600	735	0.459
task11	2231	1422	0.637
sum	23491	14263	
TRAIN OF 11	3084	2282	0.740

Silvia Masciocchi, GSI Darmstadt — ALICE Offline Week, July 10, 2008

Summary

- ALICE Offline has developed an analysis framework that hides computing scheme dependences from the user. The same user code runs on
  - Local PC
  - CAF/PROOF
  - Grid
- Framework manages a list of independent tasks:
  - Execution triggered by data readiness
  - Sequential execution of the top level tasks (train) driven by input chain
- Common I/O is managed by event handlers
- Framework fully adopted by ALICE
  - Reaching stability but new developments ongoing
  - Very positive experience so far