Analysis framework

Andrei Gheata

ALICE offline week, 5 October ‘06

Purpose

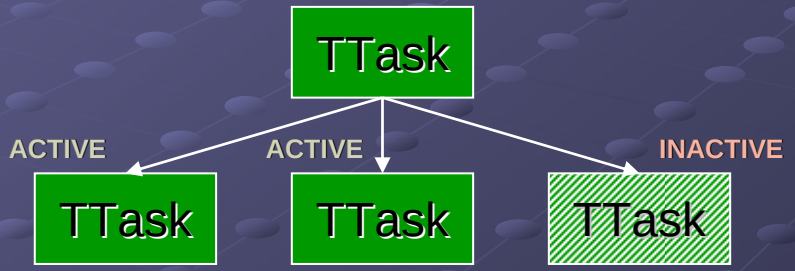
- Provide easy-to-use tools to allow data analysis in a coherent way
- Suitable for analysis ranging from simple to very complex tasks in a distributed environment
- Allow splitting complex analysis tasks in independent functional blocks possibly usable by other analysis

Functionality

- Basic ideas described at the last offline week
- Data-oriented model composed of independent tasks
  - Task execution triggered by data readiness
- Parallel execution and event loop done via ***TSelector*** functionality
- Analysis execution performed on event-by-event basis

Structure

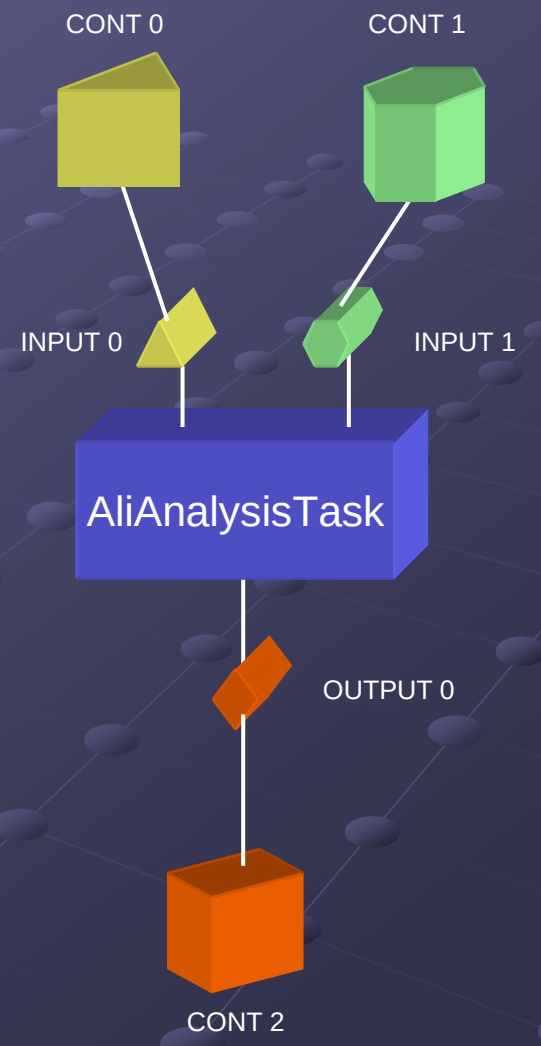
- Analysis may be split in functional modules
  - At least one
  - Deriving from TTask



- 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

Data-oriented model

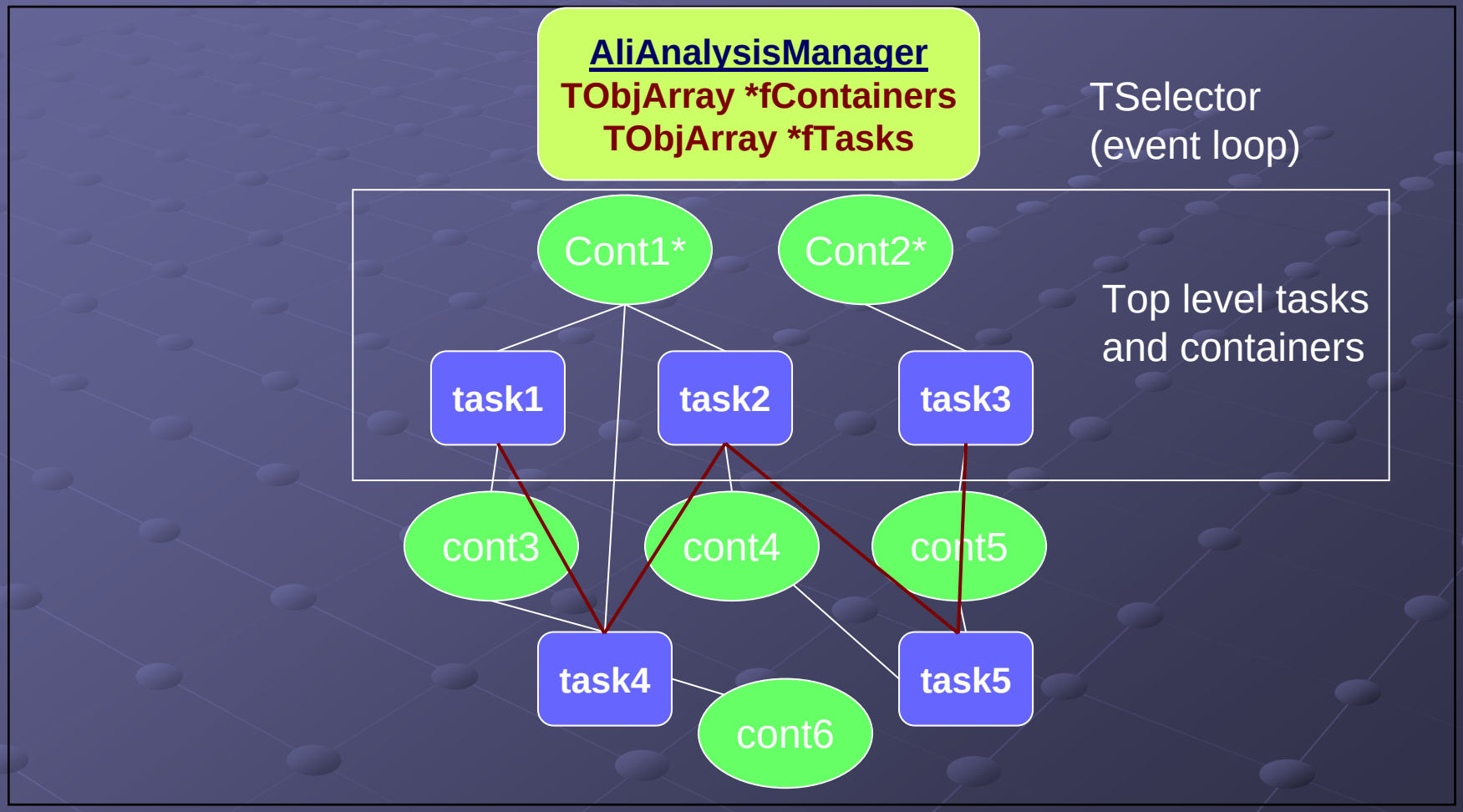
- Data type formalized by ***TClass*** usage
- Any module declares a number of input data slots
  - Each slot must be connected to a data container of the corresponding type at run time
- Modules provide data at one or more output slots



Management

- Analysis modules managed by a ***TSelector***-derived class
  - Provides access to initial input data (ESD's, kinematics, whatever…) for the top-level containers
  - Initiates the main event loop over the entries of the input trees, calling the Exec() method for the top-level tasks
- Input data is generally a TChain, but the framework can manage other data types
  - Retreival by event tags mechanism (see talk from Panos) – to be interfaced
- Parallelizing analysis execution
  - Functionality provided by TSelector@PROOF (see talk from Jan Fiete)

Data flow structure



Implementation

- Code in AliRoot
  - Inside **ANALYSIS** module
  - Classes: AliAnalysysManager, AliAnalysisTask, AliAnalysisDataContainer, AliAnalysisDataSlot, AliAnalysisContainerRL
  - Besides the last class, no dependency to AliRoot
- Separate library to be loaded
  - libANALYSIS_NEW
- Demo for package usage: testAna.C inside ANALYSIS folder

AliAnalysisManager : public TSelector

- CreateContainer(const char *name, TClass *data_type, EAliAnalysisContType cont_type)
  - Mandatory to define all data containers that will assembly the analysis
  - Container types:
    - kInputContainer – minimum 1 input container needed
    - kNormalContainer – containers used for communication between task modules
    - kOutputContainer – minimum 1 output container

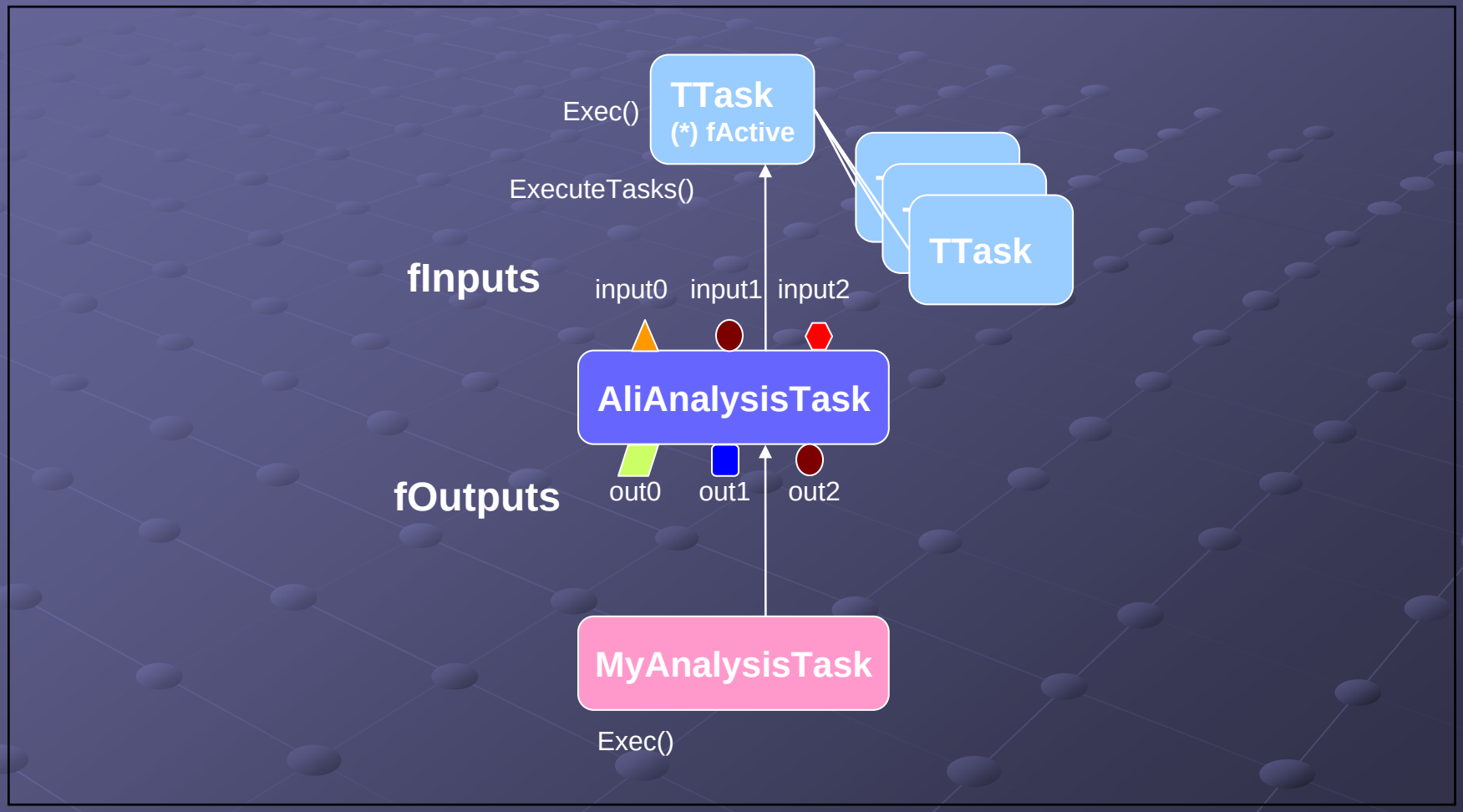
AliAnalysisManager (continued)

- AddTask(AliAnalysisTask *task)
  - At least 1 task per analysis (top task)
- ConnectInput(pTask, islot, pContainer)
- ConnectOutput(pTask, islot, pContainer)
  - Mandatory for all data slots defined by used analysis modules
- InitAnalysis()
  - Performs a check for data type cosistency and signal any illegal circular dependencies between modules
  - To be called by TSelector::Init()
- ExecAnalysis()
  - Starts the analysis
  - To be called by TSelector::Process()

AliAnalysisTask : public TTask

- User analysis module MUST subclass this
- DefineInput/Output(Int_t islot, TClass *type)
  - Mandatory at least 1 input & 1 output
  - Usually declared in the class constructor
- virtual void Exec(Option_t *option) = 0
  - Manadatory to implement in the derived class
  - This actually implements how the analysis module processes input data

Analysis module (task)



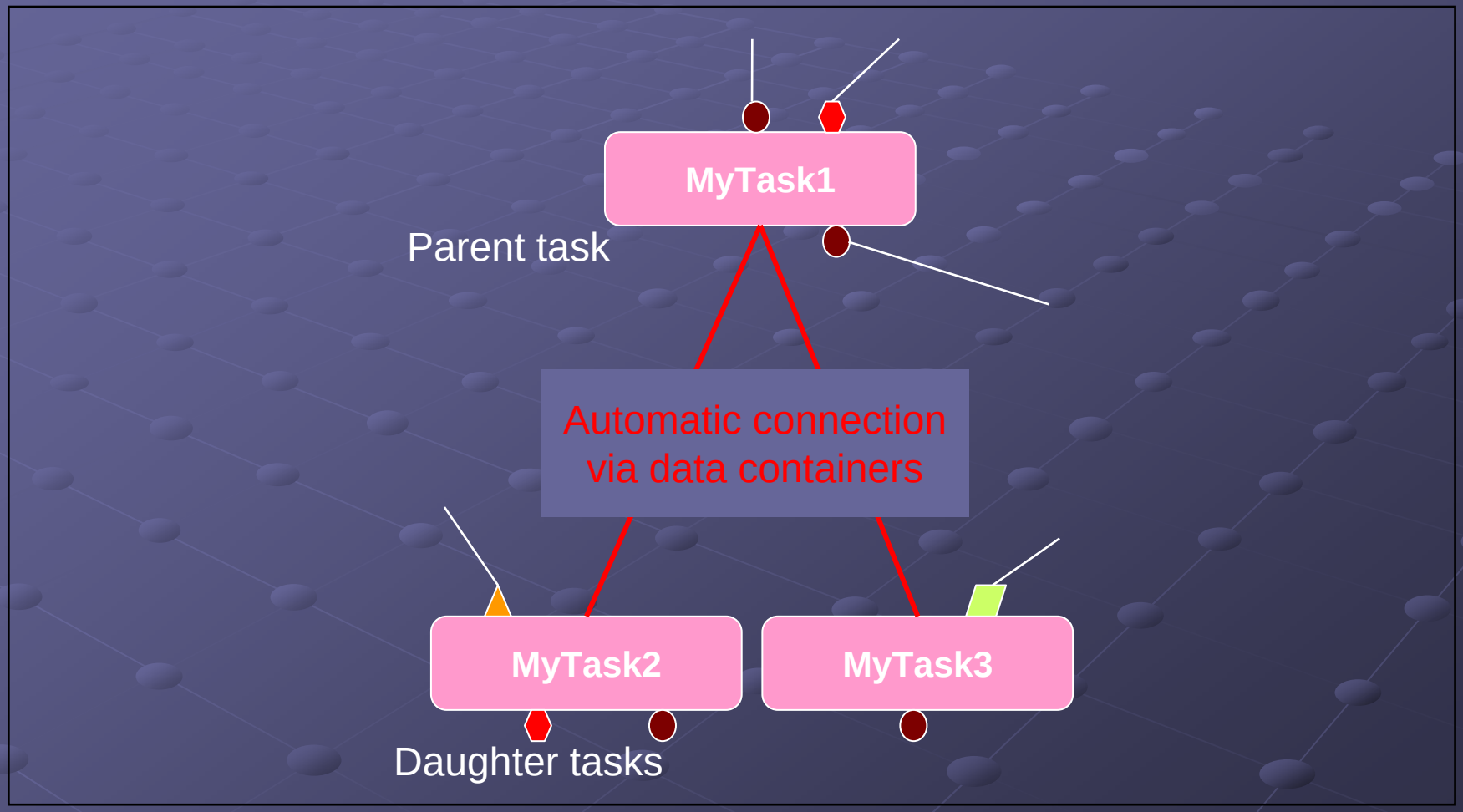
How to implement Exec()

- Accesing data from input slots
  - When Exec() is called, data will be always available at all declared inputs
  - Use: MyClass *data = (MyClass*)GetInputData(islot)
- Processing input data
  - In case of events, organize track loop
- Publishing the result at output
  - Mandatory to be done at the end of event processing
  - Use: PostData(Int_t islot, TObject *result, Option_t *option)
    - Will notify the container connected to output and all dependent daughter tasks that data is ready
    - Subtasks activated when all inputs are ready, executed by the last provider
    - Option – specifies if data should be written to a file

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
- ***AliAnalysisContainerRL*** – special container using ***AliRunLoader*** to access specific data
  - To be moved in a separate library

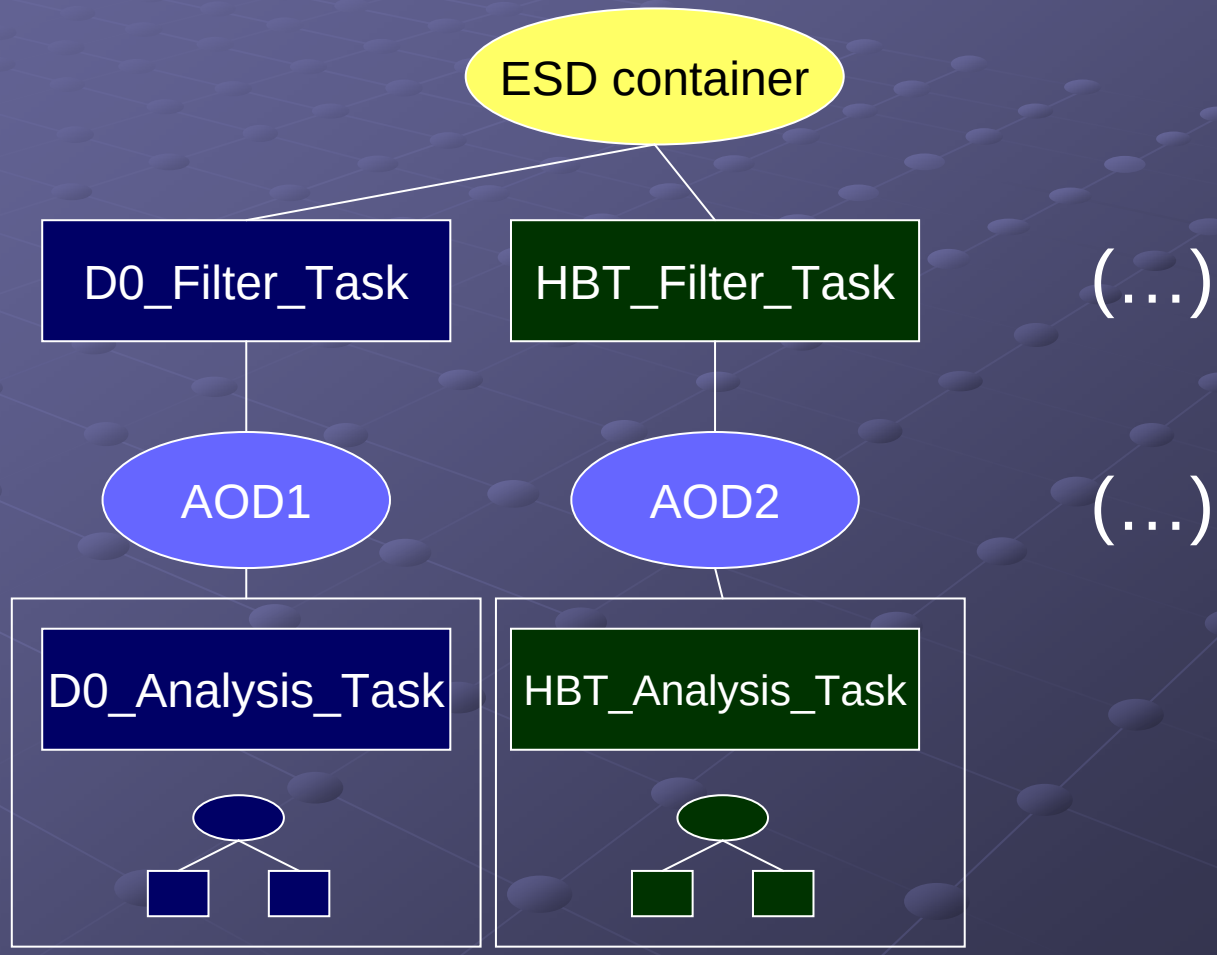
Connection via data containers



AliAnalysisDataSlot

- Input/Output task slots
- Not a class to be handled by users
  - Can be declared/created in association with a task, using methods belonging to ***AliAnalysisTask***

Example: AOD parallel production



Conclusions

- Analysis framework in AliRoot
  - Provides all needed functionality, but there are also some basic to-do's left
    - Connection to event tag mechanism
    - TSelector functionality connection
- Framework quite flexible and simple to use
  - See ANALYSIS/testAna.C macro as a simple example on how to use the framework
- Additional functionality, bug fixes, optimizations certainly needed
  - Feedback would help