ALICE Event Data Model

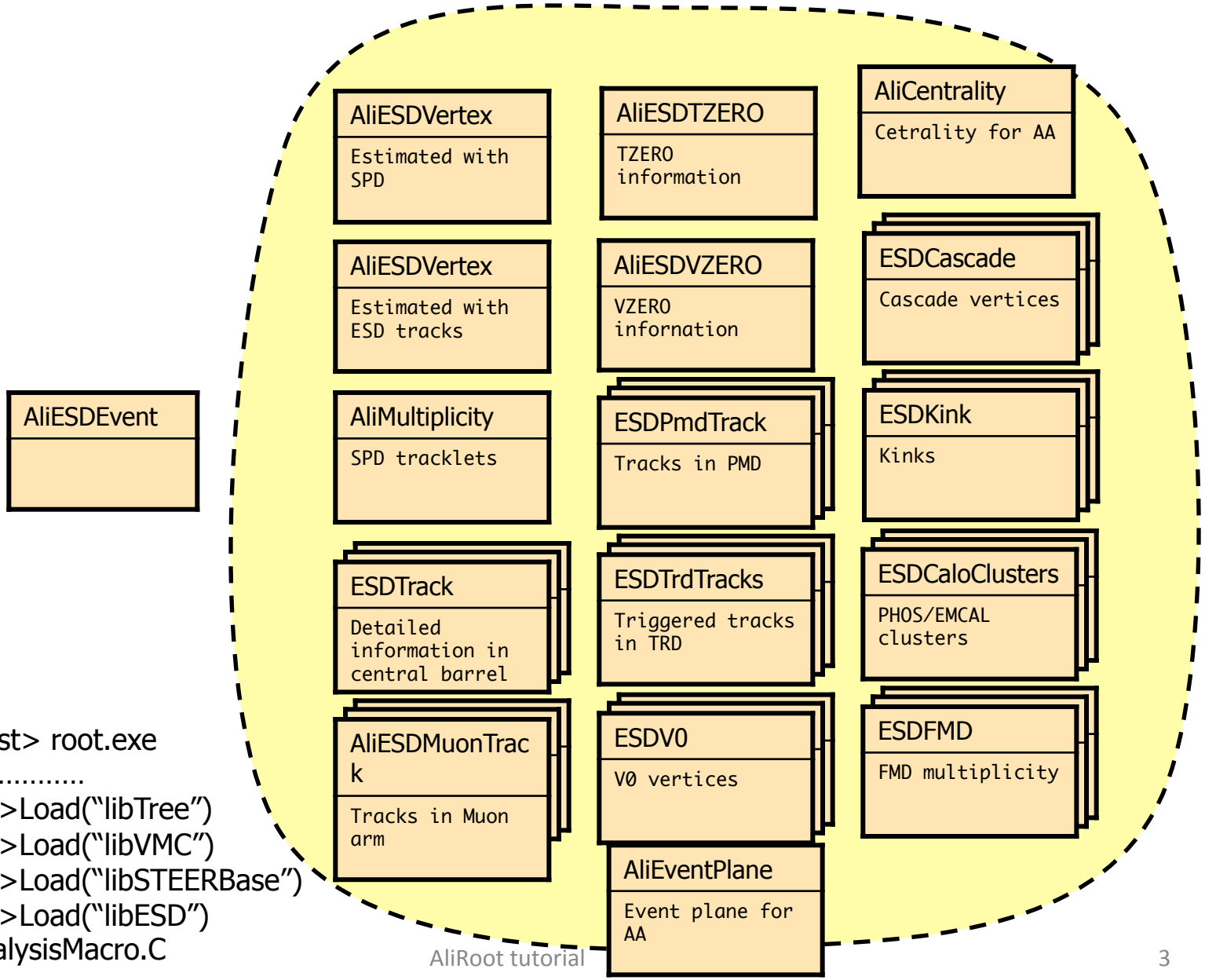
P.Hristov

08/07/2018

What can we learn from the past?

RUN1 & RUN2 EDM

The ESD



Ideally: user@host> root.exe

.....

root[0] gSystem->Load("libTree")
root[1] gSystem->Load("libVMC")
root[2] gSystem->Load("libSTEERBase")
root[3] gSystem->Load("libESD")
root[4] .x AnyAnalysisMacro.C

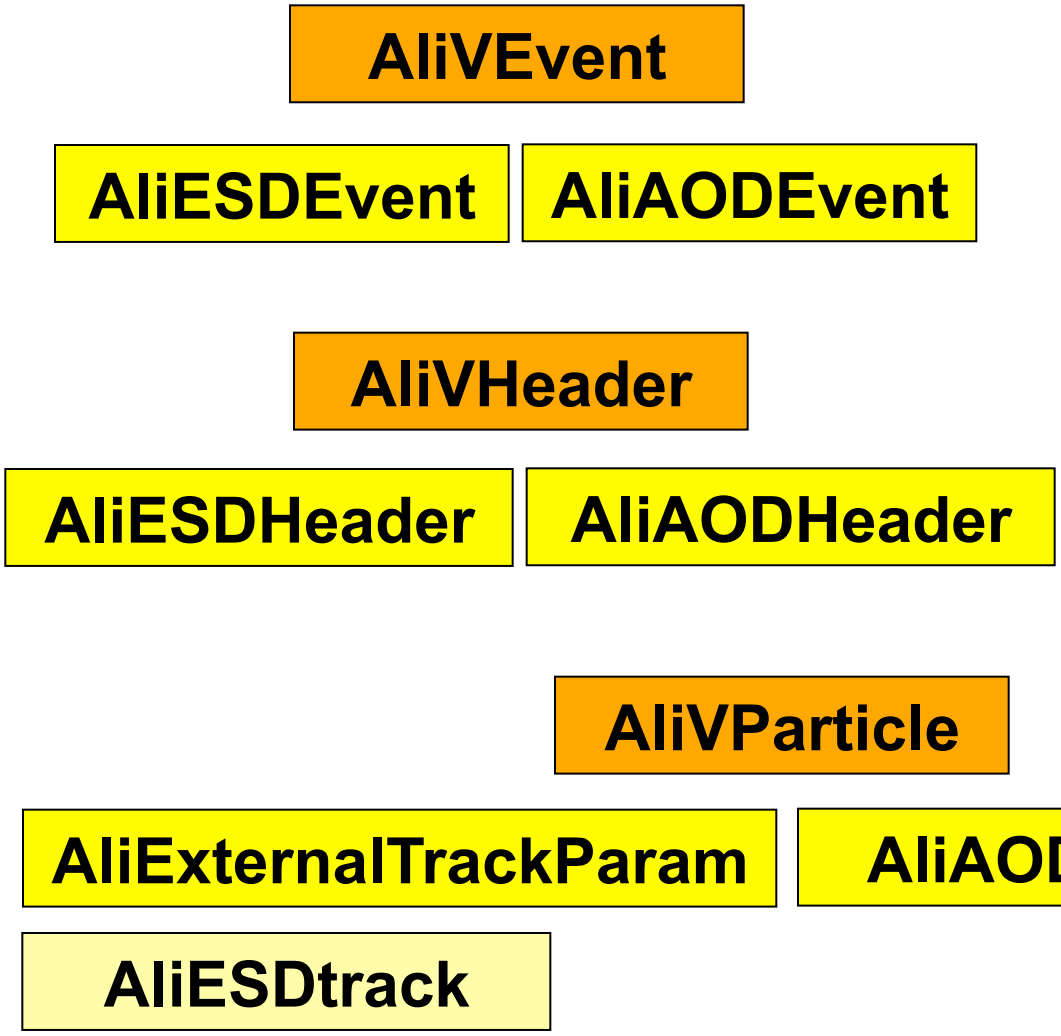
AliESDEvent and AliESDtrack classes

- Accumulation and exchange of tracking information among the barrel detectors
- Contained in the ESD and used for physics analysis

Class AliESDtrack : public AliExternalTrackParam

- final params
- reconstruction status flags
- length, time, combined PID
- vertex constrained params
- impact parameters & cov.matrix
- params at the outer TPC wall
- params at the inner TPC wall
- …
- detector specific info (chi2, num.of clusters, PID…)

Common base classes for ESDs and AODs



- standard access to containers
- common getters and setters
- some differences in the interface (to be "cured"!)

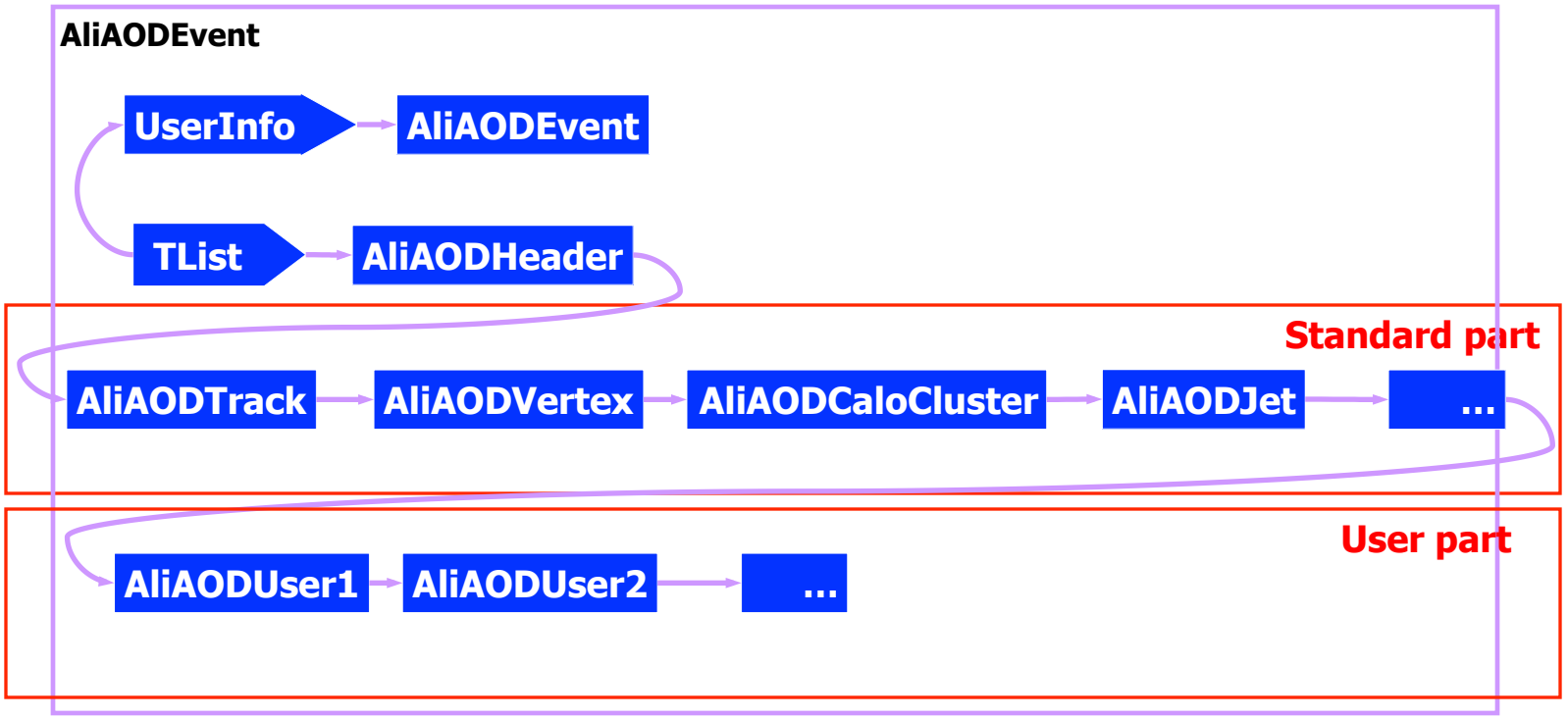
Current content of the standard AOD

AliAODEvent — contains an (extendable) TList

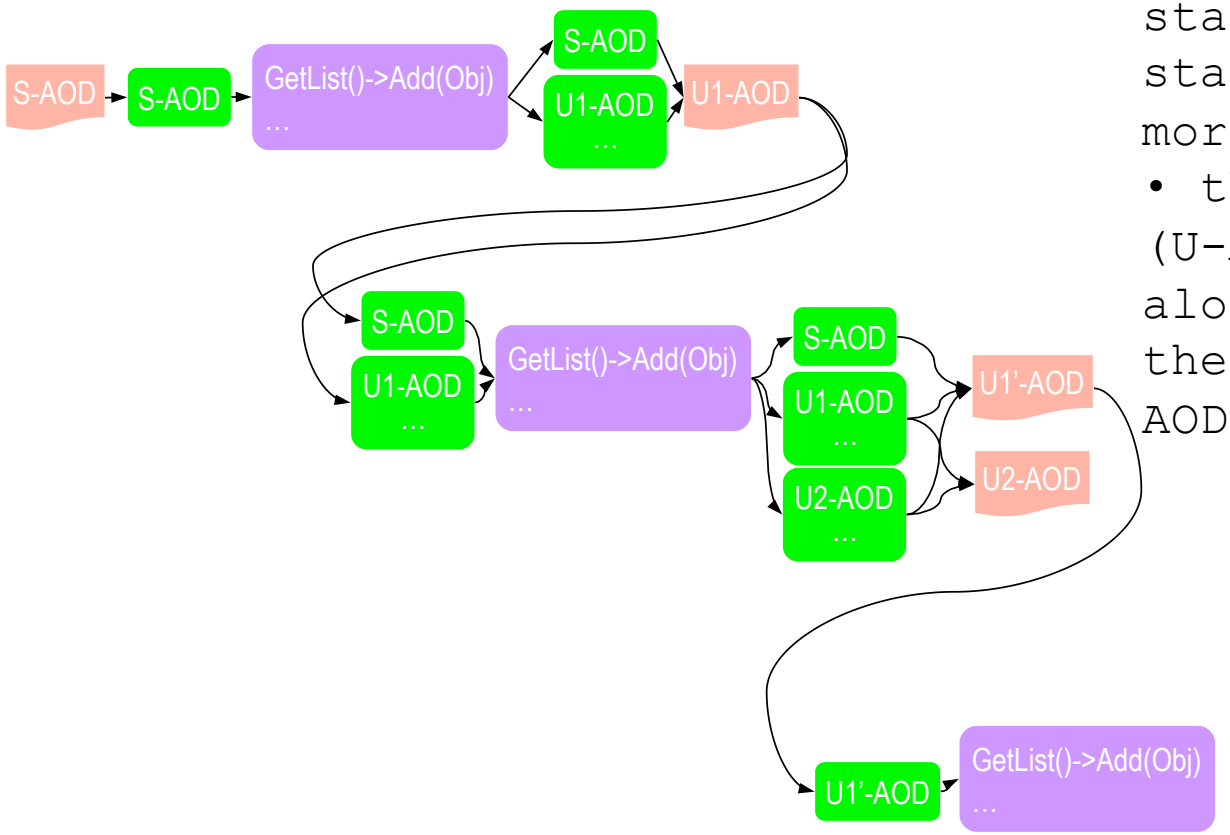
- **AliAODHeader** — event information
- **AliAODTrack** — TClonesArray of tracks
- **AliAODVertex** — TClonesArray of vertices
- **AliAODJet** — TClonesArray of jets
- **AliAODTracklets** — Container for SPD tracklets

General idea: ESD & AOD

- Very simple… use a list



Extending the AOD



Flexible and Extendable

- users can just use the standard AOD (S-AOD) or start from it to obtain more detailed results
- this new information (U-AOD) can be stored alongside the S-AOD in the TList (= in the same AOD)

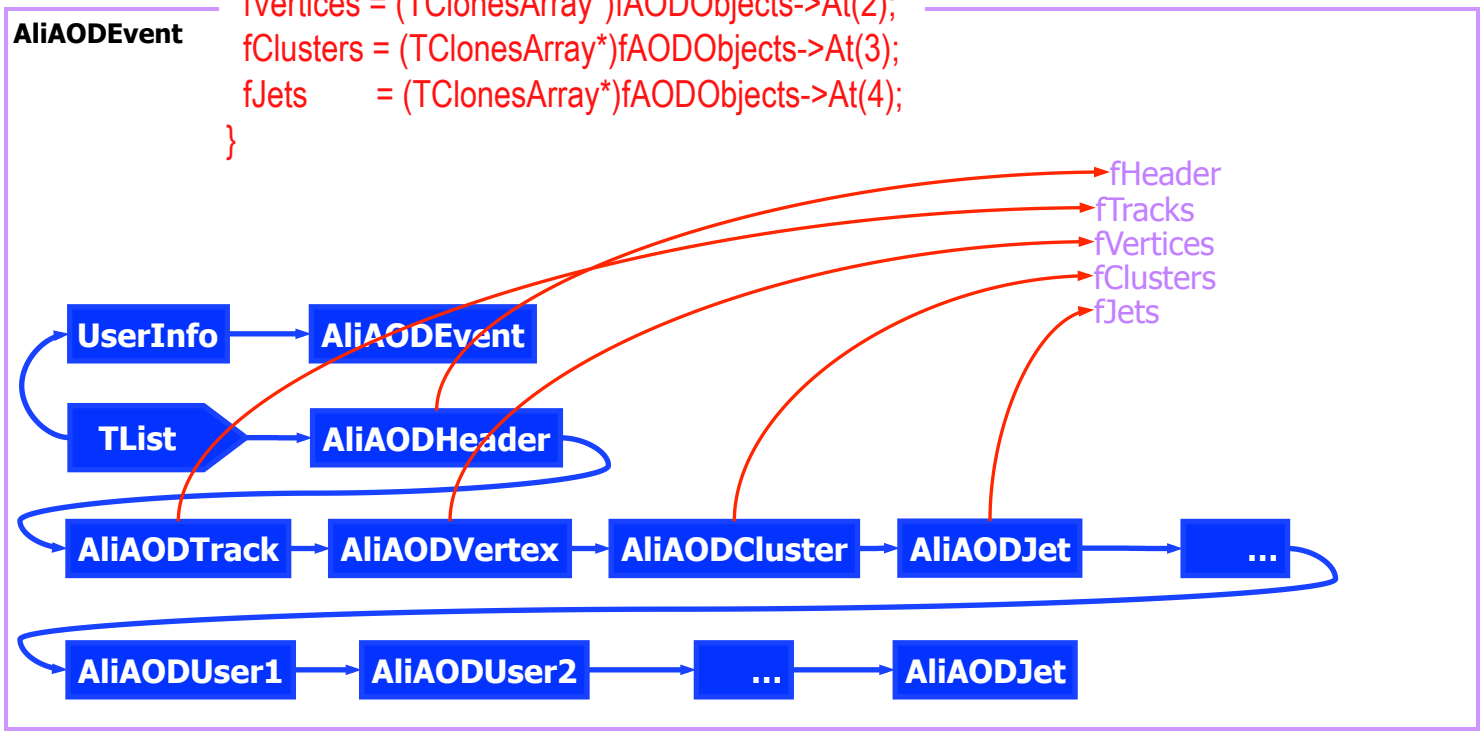
- The idea is that every user can extend the AODs with "non-standard" objects

Reading back

```
void AliAODEvent::GetStdContent() const
{
  // set pointers for standard content

  fHeader  = (AliAODHeader*)fAODObjects->At(0);
  fTracks   = (TClonesArray*)fAODObjects->At(1);
  fVertices = (TClonesArray*)fAODObjects->At(2);
  fClusters = (TClonesArray*)fAODObjects->At(3);
  fJets       = (TClonesArray*)fAODObjects->At(4);
}
```

```
AliAODTrack* AliAODEvent::GetTrack(Int_t i) const
{ return (AliAODTrack*) fTracks->At(i); }
```



- …and the same user also can access the “non-standard” objects from the previous slide

Run1 & Run2 EDM: Summary

- Fully based on ROOT
- Event-oriented data model: trees of ESD & AOD/delta AOD, but also kinematics, ESD friends, track references, tags
  - Several ROOT files
  - Access to the different data via handlers
- Deep inheritance chains
- Containers of containers of objects

- Performance @ O2 prototype facility, GSI
  - Shared file system (Lustre) + xrootd client plugin (0.6PB)
  - Job scheduler: Slurm, max. 1000 job slots
  - Possibility for local or GRID jobs
- For one process
  - Simple copy: ~1.2 GB/s
  - Unzipping: ~100 MB/s
  - P_T analysis: ~20 MB/s

Run1 & Run2: What did we learn?

- ESD and AOD => the functionality diverges despite of the common base classes
- We have IO bottleneck because of
  - Slow storage − not EDM issue
  - Decompression − partially related to EDM
  - Deserialization − purely EDM issue
- Flexibility without discipline: AOD easily becomes “trash bit” where everybody tries to store specific objects
- Complex ESD/AOD structure with nested objects, containers of containers, pointers, etc. causes deserialization overhead
  - ROOT cannot completely recover the losses
  - The flexibility comes with performance price

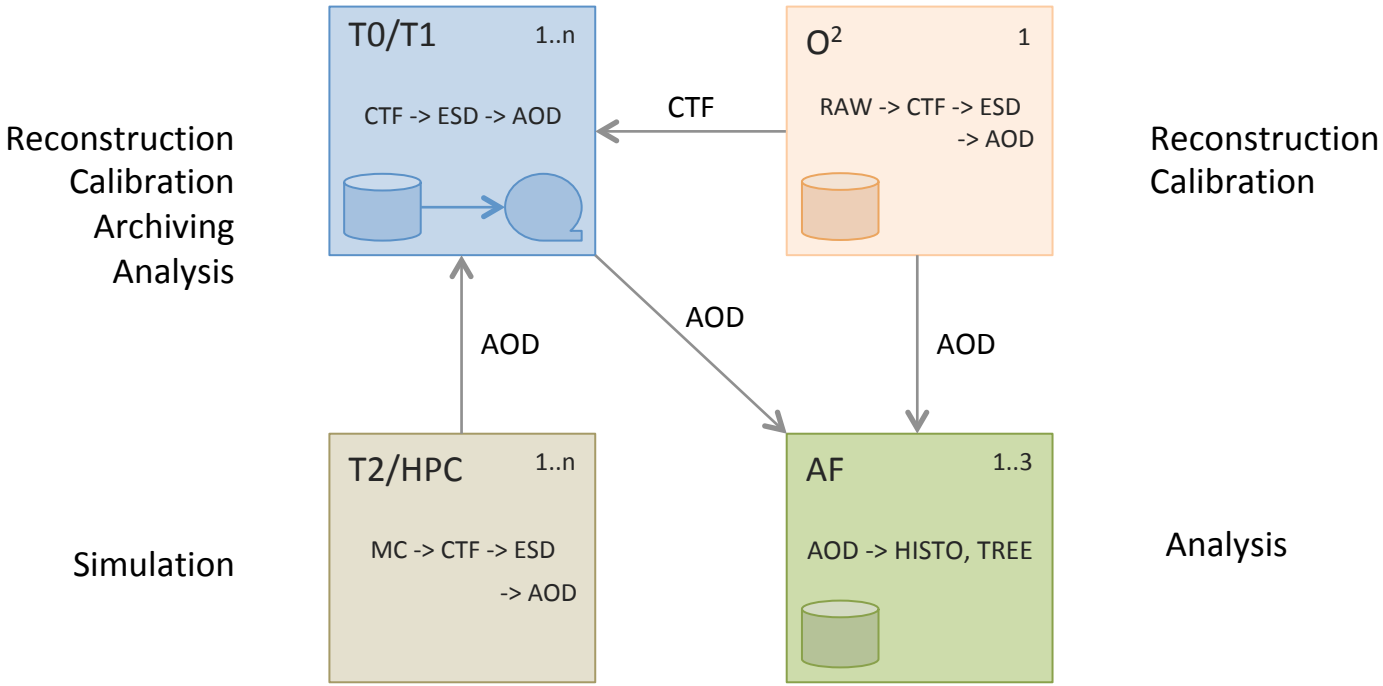
What do we plan?

RUN3 EDM

O² Analysis Framework and Facilities

Run3: Important differences

- x100 more collisions (from 0.5-1 kHz to 50 kHz Pb-Pb)
- Continuous readout => raw data in compressed time frames (23 ms, 1000 MB Pb-Pb collisions, ~2GB)
- Only AOD for analysis (transient ESD)

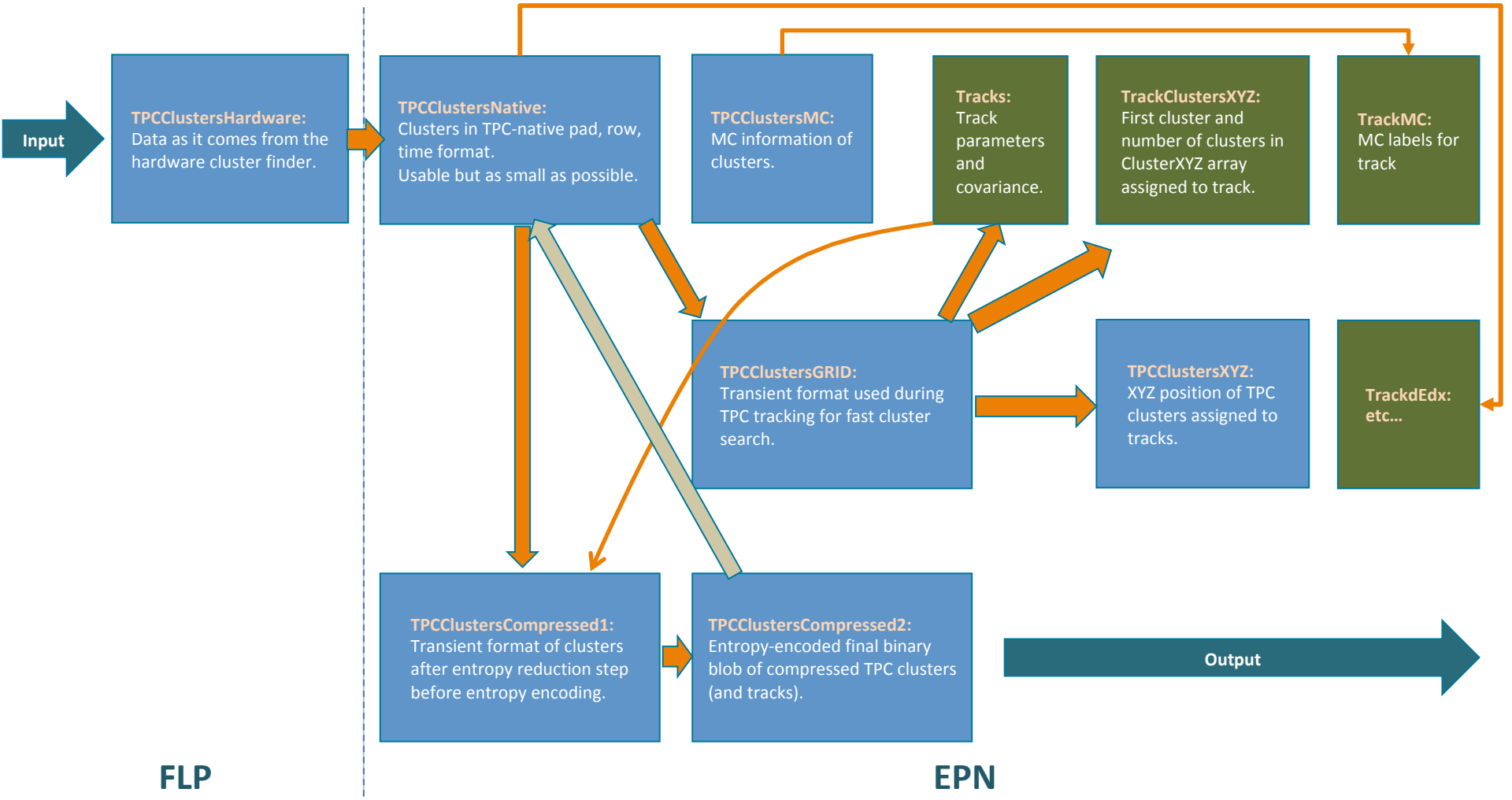


Run3 – important differences wrt Run1/2

- No “events” in general – continuous readout without trigger
- The collision is defined by its primary vertex
  - It is an association (index) of primary tracks
  - The secondary objects: vertexes, kinks, tracks, etc. are associated with the primary vertex, but can be re-associated
- Many collisions in the same time frame
- Some collisions are triggered (they are “events” in Run1/2 sense)

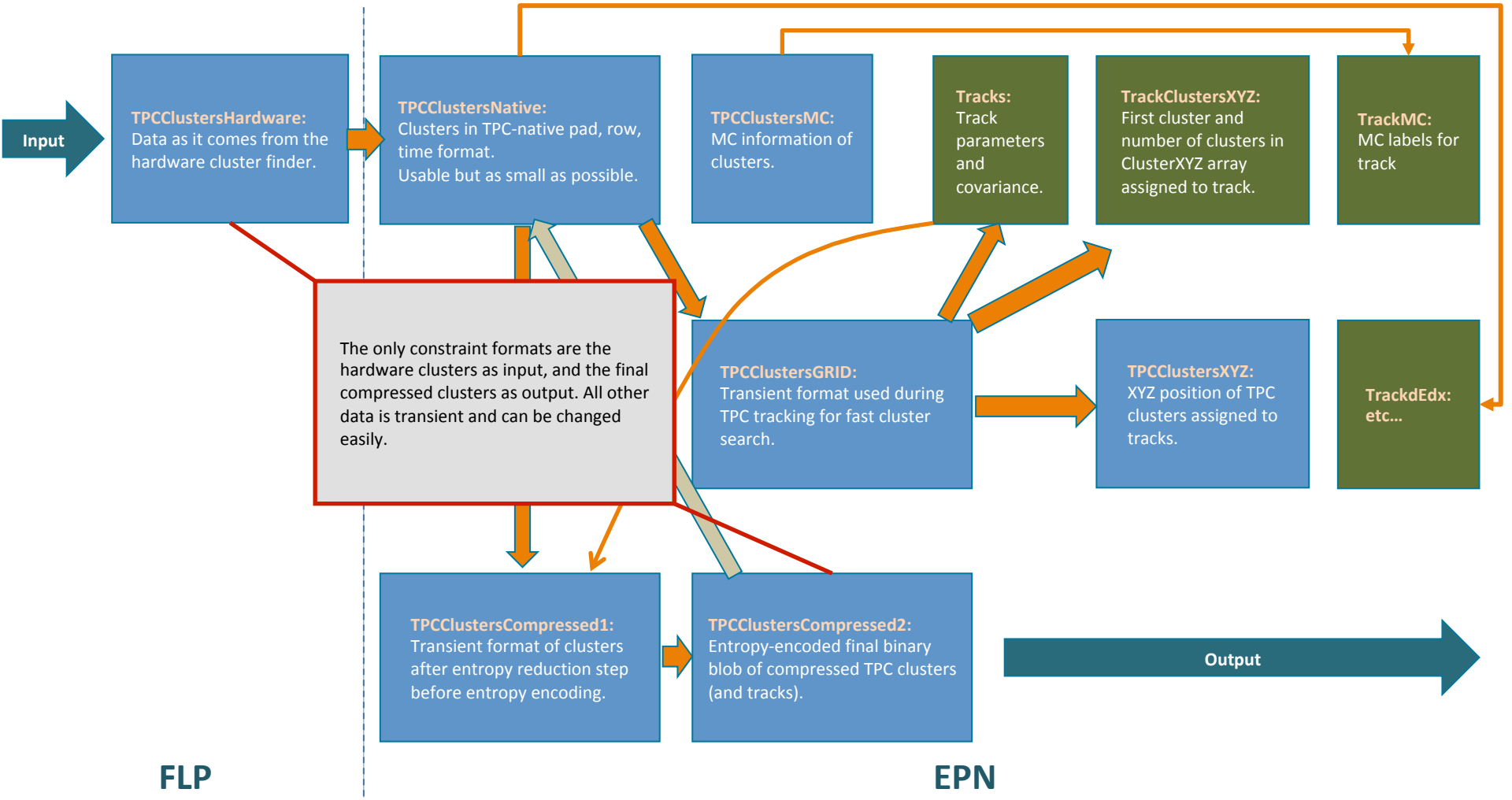
Draft of data types for TPC clustering / tracking / compression in O2

- **TPC produces the bulk of data, so a format that minimizes memory consumption is needed.**
  - **We should avoid data duplication, but some copy steps seem to be necessary.**



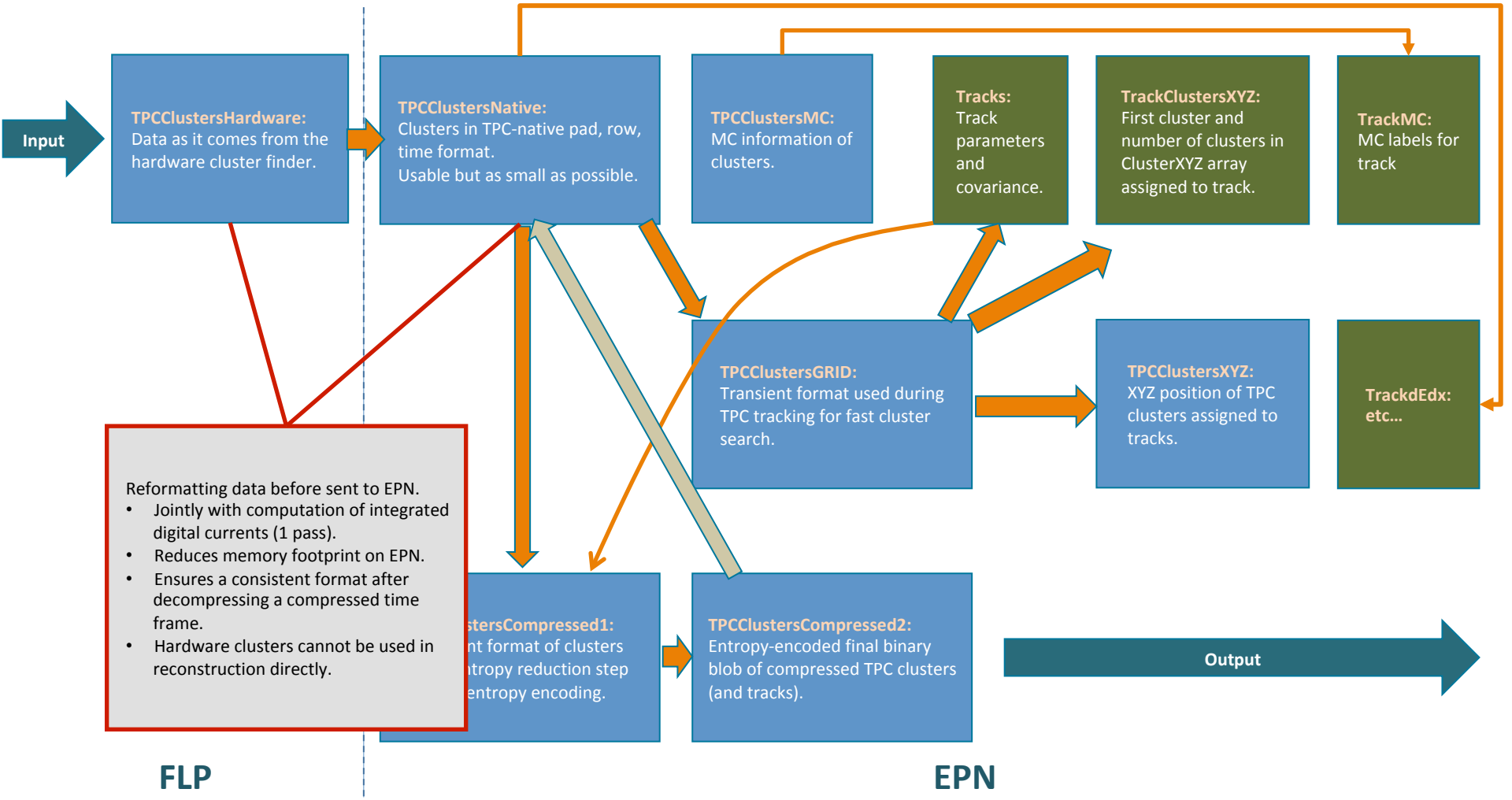
Draft of data types for TPC clustering / tracking / compression in O2

- **TPC produces the bulk of data, so a format that minimizes memory consumption is needed.**
  - **We should avoid data duplication, but some copy steps seem to be necessary.**



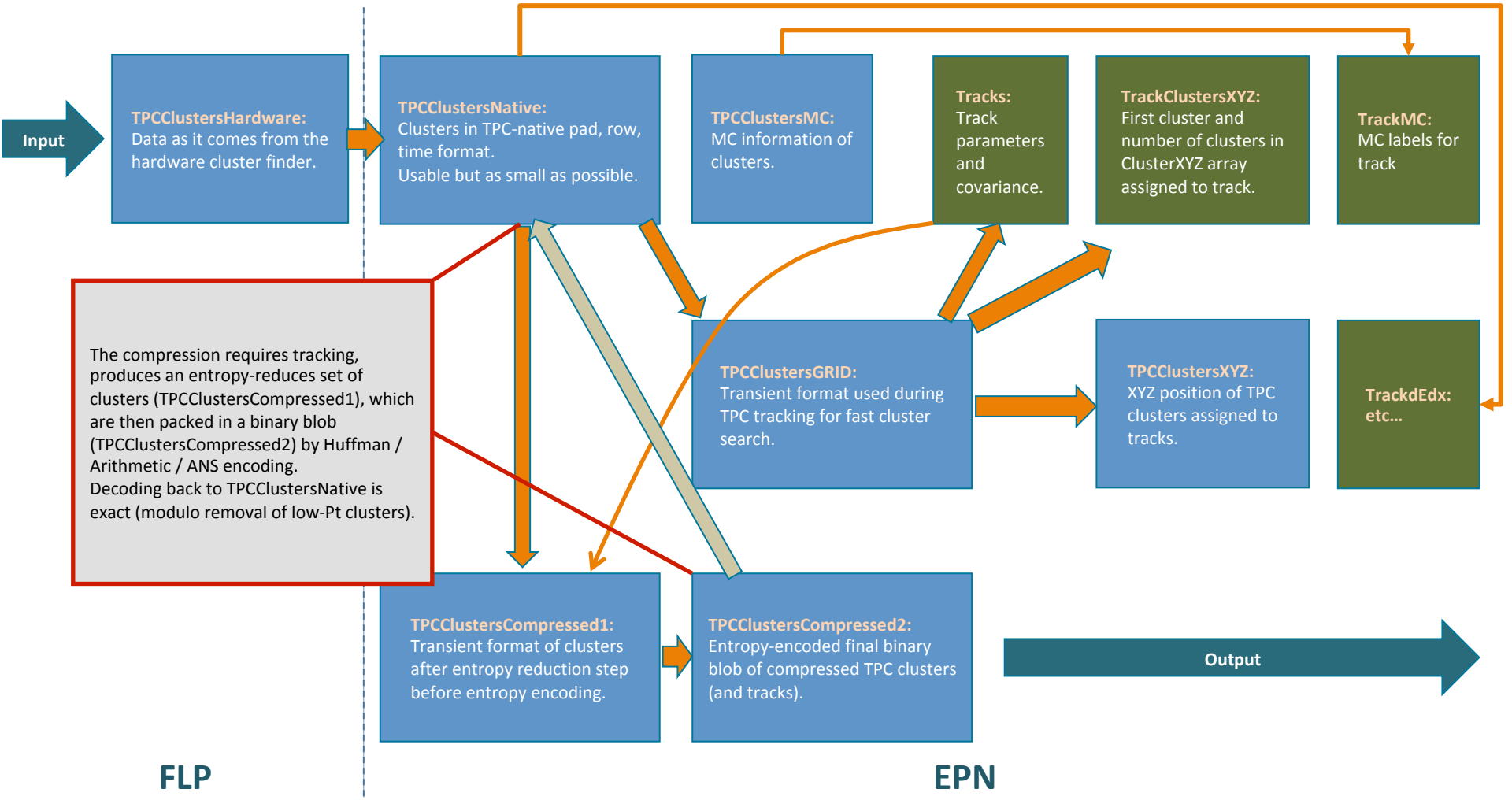
Draft of data types for TPC clustering / tracking / compression in O2

- **TPC produces the bulk of data, so a format that minimizes memory consumption is needed.**
  - **We should avoid data duplication, but some copy steps seem to be necessary.**



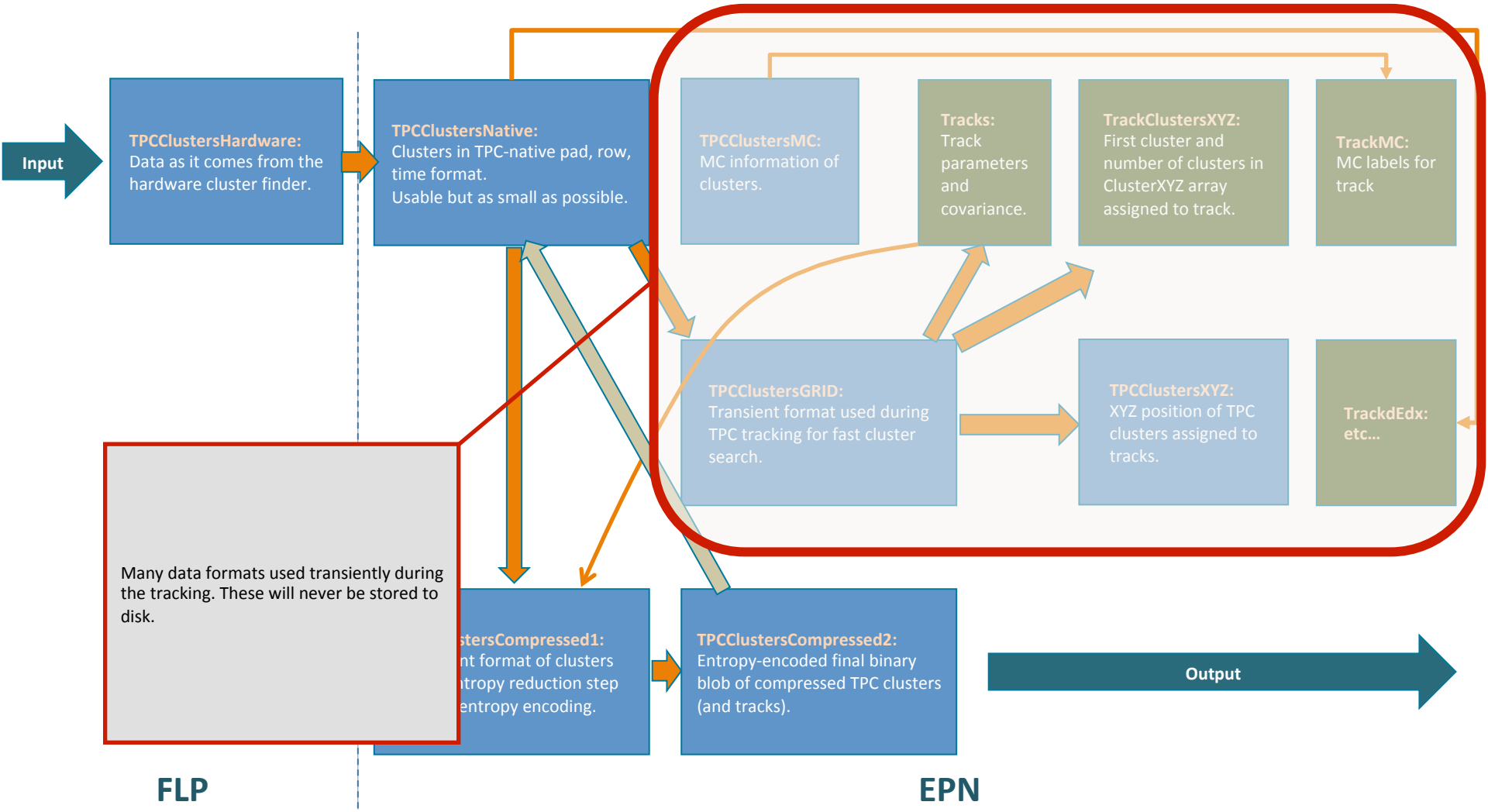
Draft of data types for TPC clustering / tracking / compression in O2

- **TPC produces the bulk of data, so a format that minimizes memory consumption is needed.**
  - **We should avoid data duplication, but some copy steps seem to be necessary.**



Draft of data types for TPC clustering / tracking / compression in O2

- **TPC produces the bulk of data, so a format that minimizes memory consumption is needed.**
  - **We should avoid data duplication, but some copy steps seem to be necessary.**



Changes in the software concept

- From sequential ROOT-based processing and format to message based multiprocessing (see the ALFA talk @ CHEP2018)
- The message serialization is very important
  - Multiple languages are allowed
  - Heterogeneous architectures (different OS, CPU, GPU, FPGA,…)
- We are investigating different options also for analysis

Analysis data format: requirements

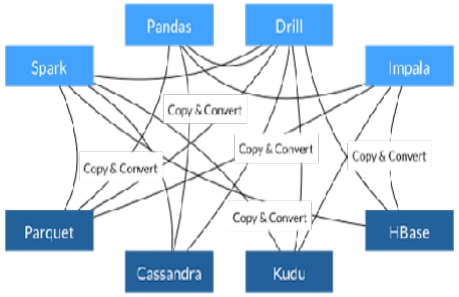
New data format should reduce as much as possible the cost of deserialization: some generality will be lost for the sake of improved speed

- **Simple, flat:** numbers only (no classes), use tables, cross-reference via numeric indices
- **Columnar:** SoA in-memory structure for better growing/shrinking and vectorization
- **Extensible:** base format is immutable, but easily extensible because it's SoA
- **Chunked:** a single timeframe can be divided in smaller units processable in parallel
- **Zero size for null objects:** filtered-out fields do not use RAM memory
- **Recompute, don't store:** do not store everything because recomputing may be cheaper
- **No data restructuring:** disk → memory → network should use similar representations

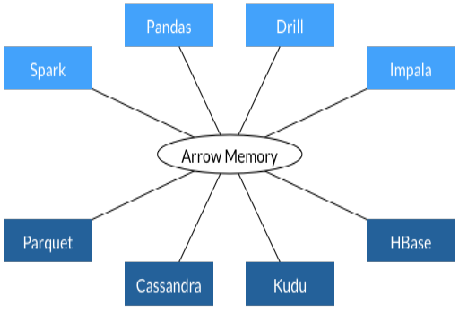
Apache Arrow Development

We have been experimenting with Apache Arrow: in-memory, language independent columnar data format. Has an on-disk companion too (Apache Parquet)

- **Leverages vectorization** and fits our other requirements
- **Units:** data organized in Tables, made of immutable Columns. Columns shared among tables (no copy)
- **Memory management:** Columns backed by Buffers, which allow for custom Memory Pools
- **Meant for interoperability:** allows for data exchange within the Apache ecosystem and outside, widely supported
- **Fits the ALICE Run 3 data model** based on message passing



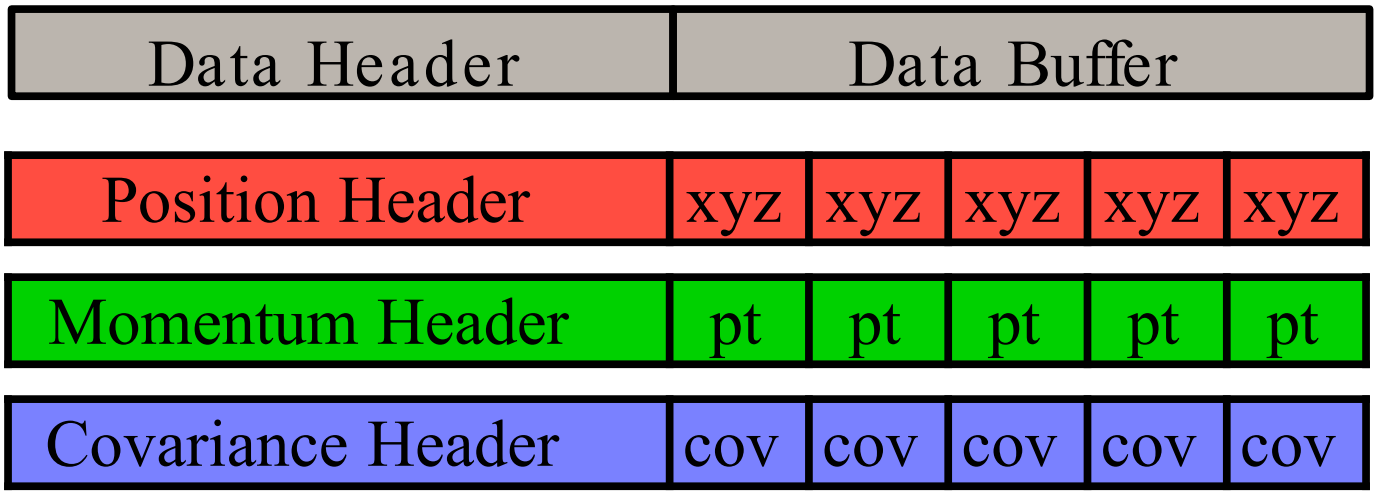
Apache Arrow
arrow.apache.org



Prototype based on Arrow: other solutions being investigated too

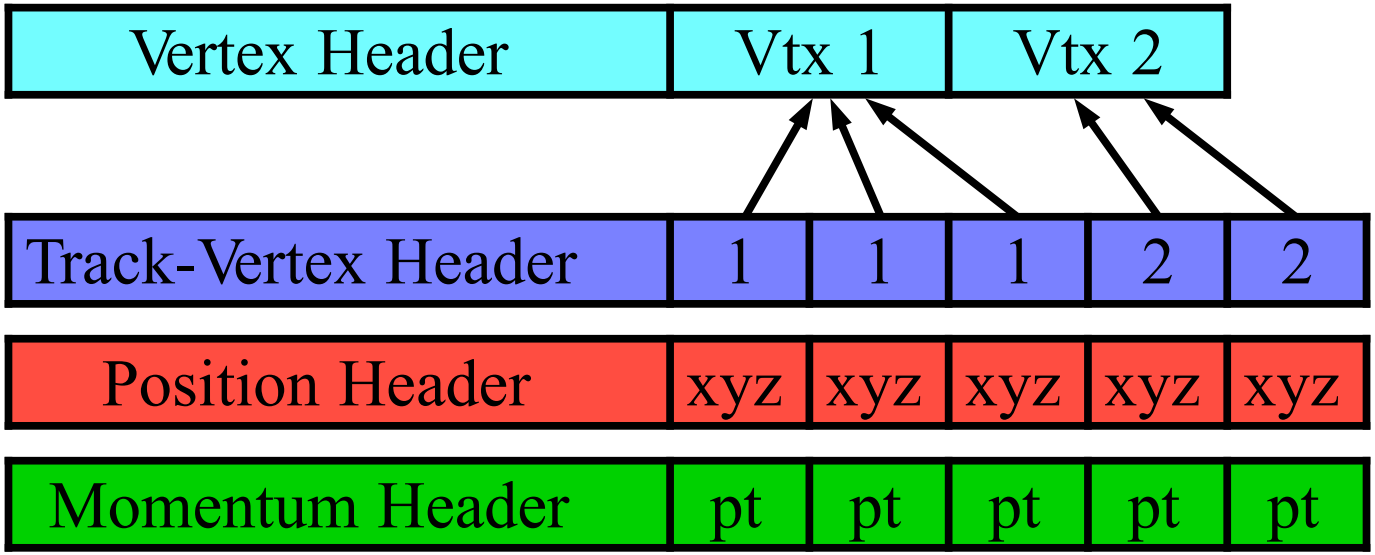
Data Message Development

- Use over the network message queuing (=buffers)
- Headers carry metadata used to uniquely identify the message payload
  - Data buffer type
  - Data buffer size
- Data element have to be flat (self-contained)
  - Any data type, arrays, structs, ...
  - No pointers



References in Data Message

- References between data buffers via the same index
- Separate data buffers can have different entry number
  - Use separate buffer for the reference
  - Track-Vertex contains the index of the vertex related to the track



More investigations: SOAContainer (LHCb)

- Code available in GitLab
- Tutorial exists
- Aims
  - exploiting modern CPUs/GPUs is hard enough
    - SOAContainer should help you try SOA code quickly, with low barrier of entry
    - code should be readable, and do what it says writable by experts and non-experts alike
    - produce reasonable code to start with (profile, optimize later)
  - will not magically solve your performance problems
    - brain not outdated yet: need good idea and right algorithm
    - may still want to look at assembly to judge if compiler needs help (simpler loops, break dependencies, …)
    - compilers may not autovectorise the way you want them to
- Use a skin to describe
  - which fields make up an SOA object
  - give the underlying tuple a nice interface gets all the methods defined in the fields
  - can define methods that access more than one field

More investigations: Libflatarray

- Who should use it?
  - Basically anyone who wants to store objects or structs in an array with 2 or more dimensions and needs to iterate through them.
- Features
  - object-oriented interface, but SoA memory layout
  - good for vectorization (SSE, AVX, CUDA)
  - zero address generation overhead
  - fast iterators, yield access to neighboring elements
  - efficient copy in/out
- Small development team (5 contributors)

Conclusions and Plans

- Variety of data formats in the quasi-online O^2 processing
- ALICE is designing a new flat event (collision) data model for the Run3 analysis
- We have gained a lot of experience from Run1/2 (positive: organized analysis in trains; negative: design of EDM)
- Several options are in different stages of investigation
- The plan is to have detailed comparison by the end of the year and finalize the Run3 EDM design