Towards AODs for Run3

Mikolaj Krzewicki
FIAS

Overview

- Part I
  - The use of ESDs in online analysis of data in a distributed system.
- Part II
  - Basic recap of AOD related discussions and work done in the context of work package 1 (data model).

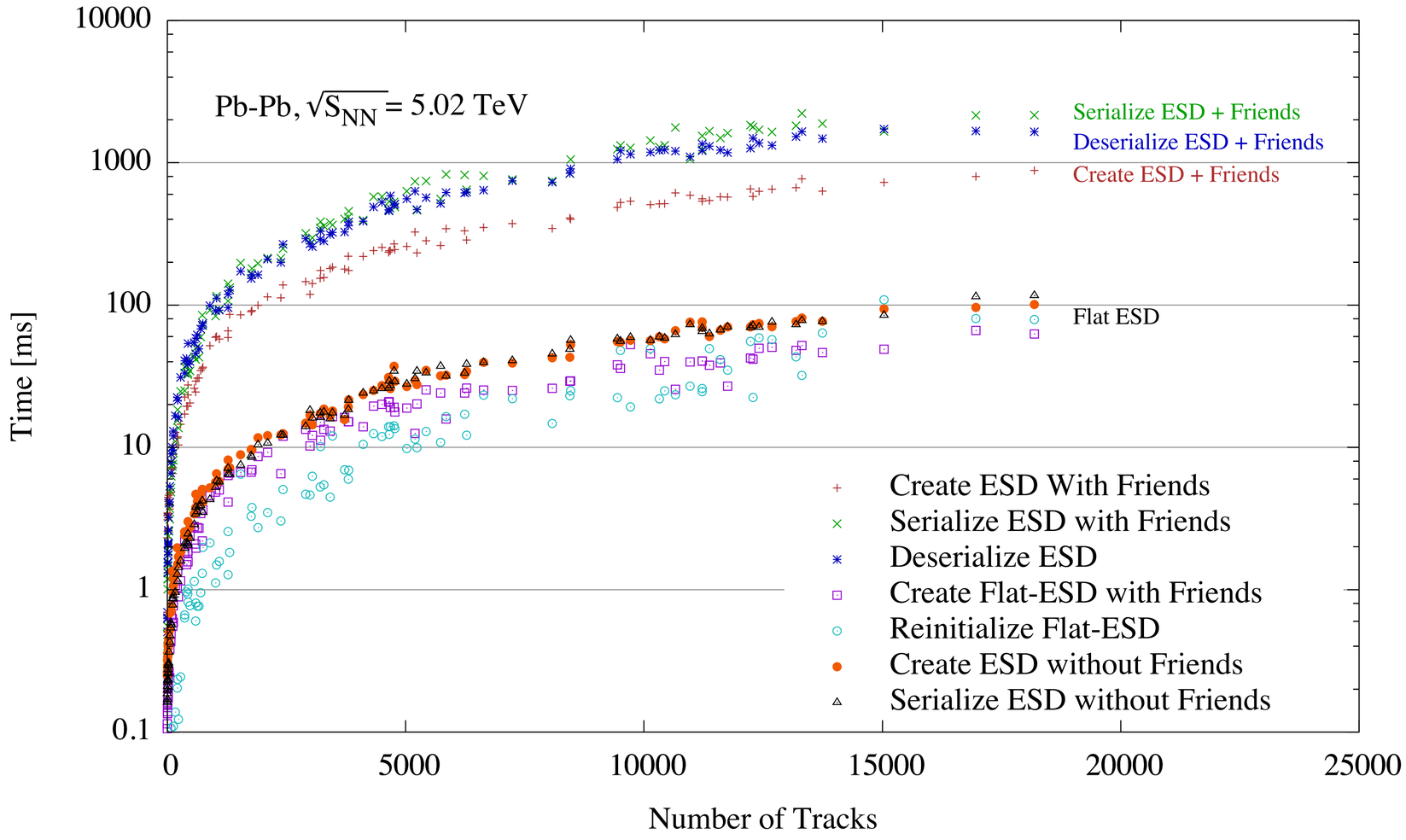
ESDs in the HLT

- HLT is a distributed system: separate processes exchange data via messaging queues, everything is in-memory.
  - -> O2 builds on the same concept(s).
- Internal data representation is mostly flat
  - no serialization/deserialization overhead.
- Many different data types, reconstructed tracks, clusters, etc....
- At the very end of processing an AliESDEvent is created from the internal data (and sent to DAQ for saving in the raw file).

- Since ~2 years:
  - online (TPC) calibration, online QA using offline code (as AliAnalysisTask).
  - could also in principle run (some) physics analysis.
  - since we have an ESD online it would be simple, BUT:
  - (ROOT) serialization/deserialization too expensive! (both time(CPU) and memory)
    - ESD serialized once, deserialized by each component (process) that uses it.

ESDs in the HLT

- Flat ESD:
  - A flat structure with same data as ESD
  - Contents share a virtual interface with AliESDEvent:
    - (AliVEvent, AliVTrack)
  - No need for serialization, a vtable reinitialization instead of deserialization.
  - Negligible overhead



ESDs in the HLT

- Flat ESD caveats: User tasks need to use the base interfaces instead of ESD/AOD directly.
- Not all access methods could be kept compatible:
  - Some special getters to be used instead of existing ones.
- Some porting is necessary, but:
  - After porting a task runs transparently online on flatESDs and offline on ESD/AOD.
  - No performance loss observed.
- This was a major effort, many people involved.

- Aftermath:
  - Due to different data models and representations online and offline this was not trivial.
  - Still technically ESD and various AOD flavours technically not quite compatible.
  - In Run3 we should avoid this proliferation.

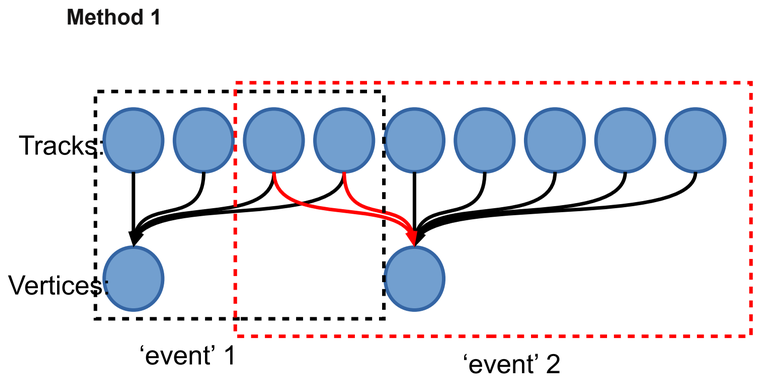
Data in O2

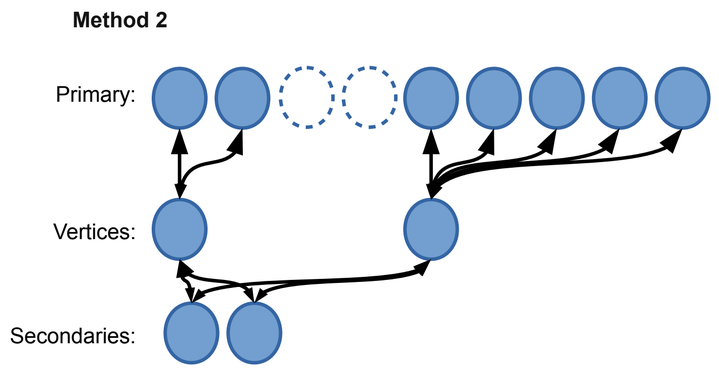
- As in the previous case (of the HLT) data is exchanged between processes in the form of messages (contiguous buffers)
- To avoid overhead the buffers contain flat data (wherever feasible)
  - structs, arrays etc., no pointers, buffers have to be self contained.
  - references between data types (track-to-vertex, track-to-clusters etc.) expressed with indices.

- Run3 -> continuous readout
  - Concept of an “event” gone -> data unit to become timeframe that contains data from multiple collisions.
- Timeframe consists (logically) of a collection of tracks, vertices etc...
  - possibly with some preliminary track-to-vertex associations.
  - possible ambiguities in associations to collision, especially secondary tracks.

Physics data in O2

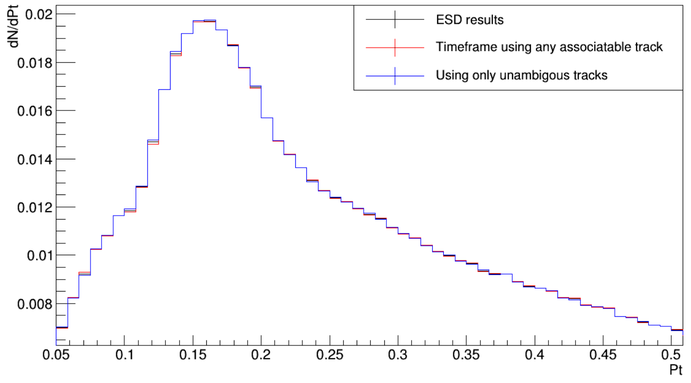
- Some preliminary studies done on the impact of ambiguous track-to-vertex associations on storage, interfaces and physics (using toy models and converted ESDs).
- Needs to be picked up and continued.





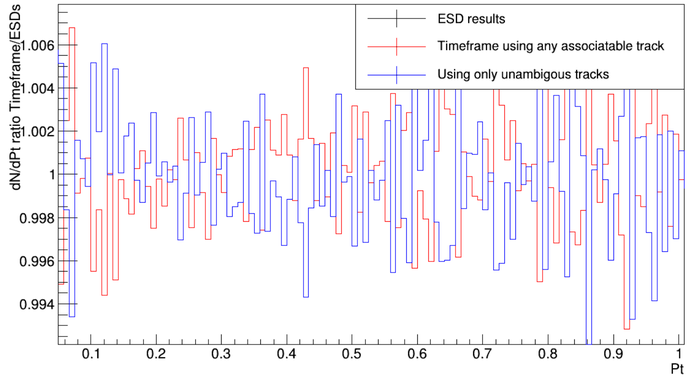
Pt Spectrum comparison.

- Normalized global tracks



Pt Spectrum comparison.

- Ratios global



https://indico.cern.ch/event/590690/contributions/2384589/attachments/1378058/2093436/pretty.pdf

Run3 AODs: general guidelines (1)

- (from: https://indico.cern.ch/event/538783/contributions/2191535/attachments/1284561/1909911/CWG4_AKal_MFloris.pdf)

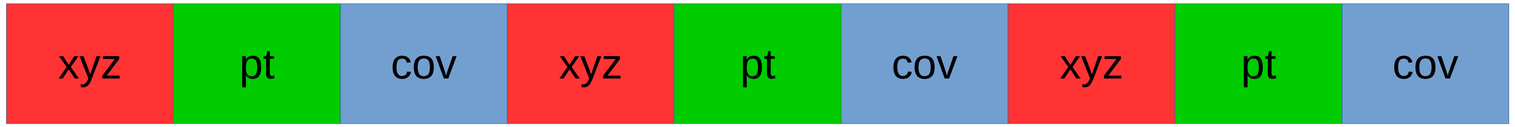
- [G1] The ideal design of the data format is strongly influenced by the workflow of the analysers and vice versa.
- [G2] The data format should allow for schema evolution: while analysing data, we might discover that vital information is missing.
- [G3] There should be one single format with a universal and abstract interface
- [G4] Classes for analysed physics objects (tracks, vertices, V0s) have to be unique (no ESD / AOD version as in current schema).
  - To be ensured also by e.g. a proper inheritance structure (e.g. ESDtrack should inherit from AODtrack in current schema)
- [G5] Analysis can produce objects. The framework should naturally allow to store them or to produce them on the fly.

Run3 AODs: general guidelines (2)

- [G6] The framework should allow efficient skimming, pruning, and growing (adding of objects).
- [G7] The framework should foresee a user-friendly staging of higher level analysis files at Tier-3s.
- [G8] Track parameters have to be written in the tracking reference frame (similar to the current AliExternalTrackParam).
- [G9] MC information should be smoothly integrated from the beginning (stored close to the data, kinematics in one stream that can be read independently).
- [G10] Even if not full backward compatibility can be kept, all interfaces should remain as close as possible to the current implementation.
- [G11] Whenever possible avoid using pointers and TRefs, instead rely on indices. This is more compact and facilitates skimming and IO, especially for shared objects.

Data layouts: array of structs (AoS)

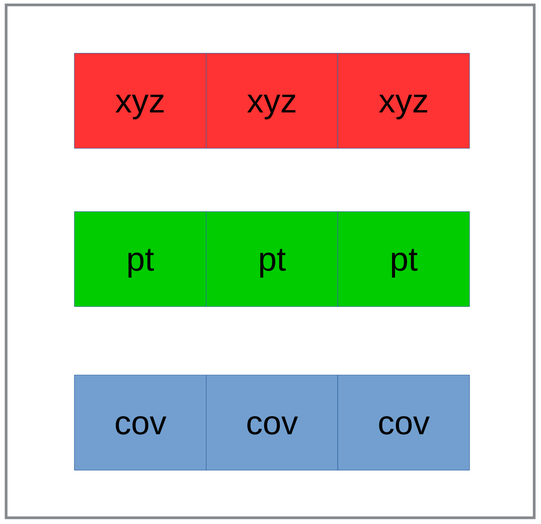
- Similar to what we have now (in memory).



- Advantages:
  - When reading all tracks in a row, produces linear reads from memory → good performance
  - Same as currently used → no need to adapt code

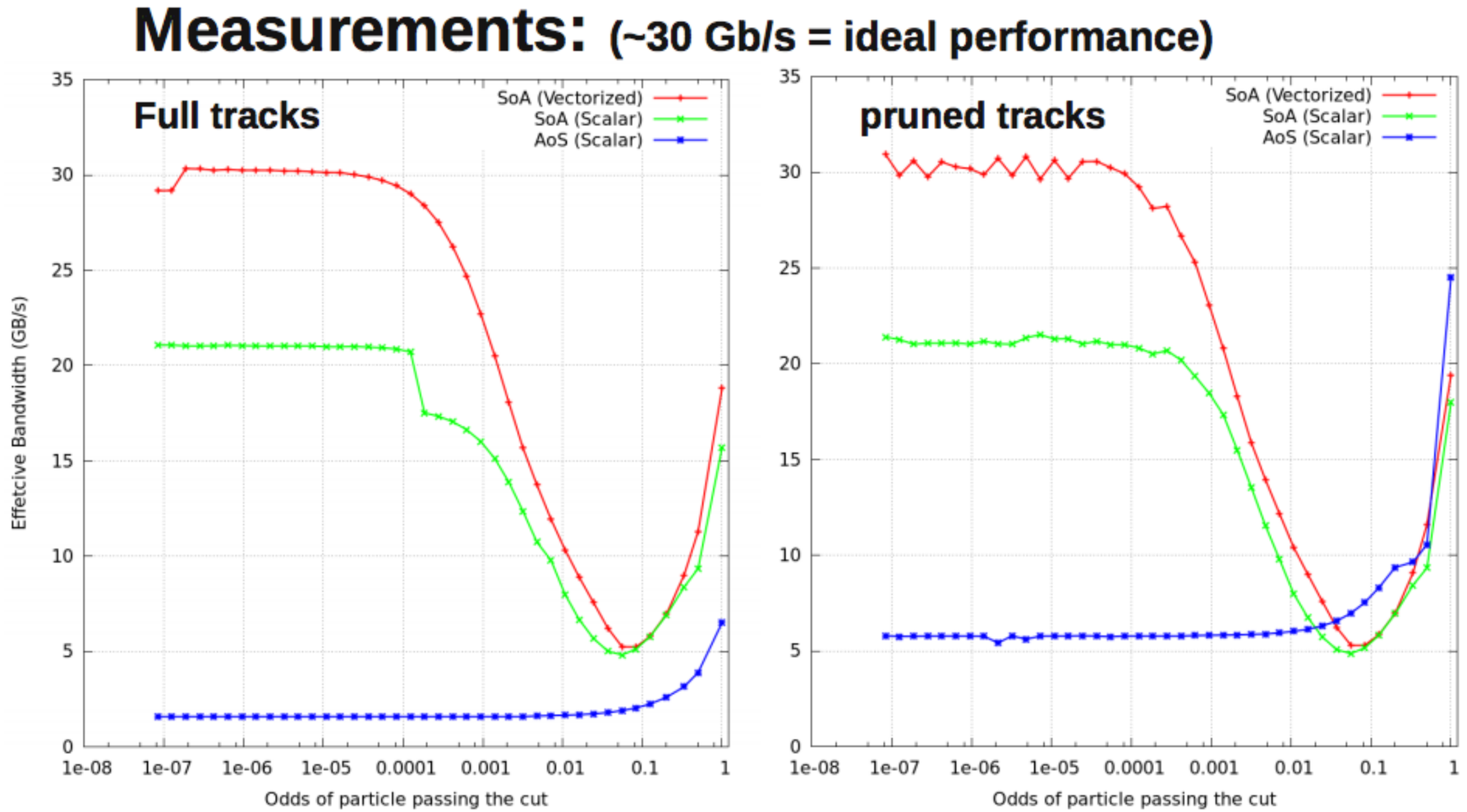
- Disadvantages:
  - Data types are interleaved → Hard to impossible to do effective pruning, difficult to vectorize.
  - Extending an existing array requires moving all elements to make room for the new data.

Data layouts: struct of arrays (SoA)



- Advantages:
  - Modular, a collection of arrays.
  - Each data member put in a separate array → effective pruning,growing (also on-the-fly), vectorization
  - Easy to extend: only need to add a new array to the collection, do not have to touch existing data.
- Disadvantages:
  - In order to use effectively, requires a slightly different thinking pattern.
  - Lots of scattered reads to load a whole track.

AoS vs. SoA



https://indico.cern.ch/event/615901/contributions/2487092/attachments/1417057/2169981/presentation.pdf

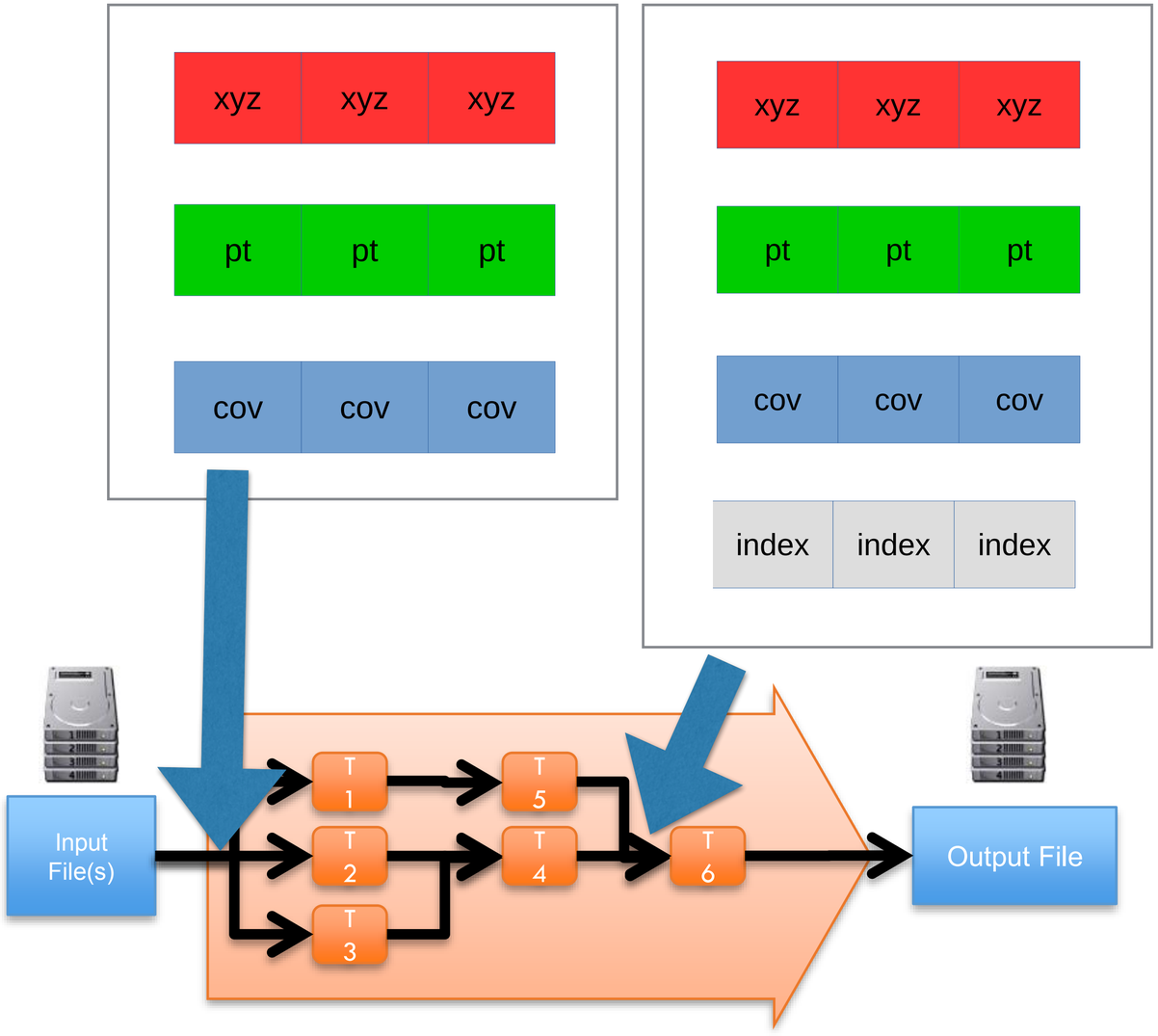
ALICE© | 2016-11-25 | Roel Deckers 16

- SoA has performance benefits, ties with AoS in worst case.

Analysis data flows

In the O2-like processing scenario:

- Constraints similar to “online” processing:
  - Data flows through a processing topology
  - Tasks (devices) get data from multiple sources
    - Main input
    - Output of other tasks (e.g jet/V0 finder, physics selection,...) as additional data parts (data is immutable!).
  - Serialization/deserialization overhead?
    - can be avoided by keeping the data simple: arrays, indices etc...
    - reminder: exchanging ROOT data is expensive in a distributed environment! *



*your mileage may vary

Interface for end-user analysis code

- Since SoA has clear benefits from both the performance and usability (effective pruning, growing) point of view, work continued in that direction (while trying to achieve (at least) conceptual backward compatibility).

- An early prototype (proof-of-concept) of an AoS-like interface to SoA data available:
  - Template based (types/code on demand) - as opposed to static inheritance based interfaces.
  - https://github.com/alice-aod-upgrade
  - Single interface to various data variants: basic, pruned, grown etc., driven by the task requirement and data contents.
  - Only minimal changes to (existing) user level logic (and code) needed.
  - Needs to be evaluated (and possibly developed further).

  - https://indico.cern.ch/event/615901/contributions/2487092/attachments/1417057/2169981/presentation.pdf

Open questions

- Define contents - maybe based on current AODs.
  - what can be recomputed on the fly? (storage vs. IO vs. CPU)
- Develop an interface to access the data - would be good to have only one...
  - Like in Run2 some analysis will run “online” as well.
- Persistent format (on-disk) to be determined.
  - All timeframe (AOD) parts in single file?
- Analysis strategy (fully distributed, other?)
  - might dictate optimal data representations/interfaces due to serialize/deserialize cycles.
- Manpower...