AOD/ESD plans

Status and plans focusing on LVL2 e/γ and some items for discussion

On behalf of: J.Baines, P.Casado, G.Comune, A.DiMattia, S.George, R.Goncalo, N.Konstantinidis, C.Santamarina, M.Sutton, M.Wielers, E.Wöhrling

The problem : trigger use cases

- We will need some (redundant) trigger information available offline for debugging if (when) things go wrong
  - This includes navigation information (re-design ongoing)
  - One possibility is to re-run the trigger on Raw Data: need to store enough information to compare online and “re-run” result
- Need ways of communicating between LVL2 and EF (through serialization into bytestream)
- Need to make it easier to develop tracking/calo/etc algorithms inside Athena: persistify spacepoints/calo cells in ESD
- This means storing information in Pool (ESD/AOD) and serializing information (LVL2->EF)
- It is important to maintain enough flexibility:
  - Not much information would need to be passed between LVL2 and EF in normal running, but potentially every LVL2 data object should be kept for a subsample of events
  - MC poses different constraints than online running: more information to persistify

The problem : physics groups use cases

- More and more interest from physics groups on trigger issues (if you don't trigger on it, you can't analyse it!)
- Need to provide ways for trigger information to be available for physics analyses (i.e. in the AODs)
- This may mean several different things:
  1) "Yes/No" result of hypothesis algorithms only: limited use; probably good enough for a normal physics analysis; would generate valuable feedback from physics groups
  2) Enough information to allow some tuning of cuts in hypothesis algorithms: more info than previous case; must include some navigation information; even more valuable feedback from physics groups; allow development of new trigger menus
  3) Everything (…this means running trigger from RDOs; not feasible for physics analysis)
- Not much time left:
  - We should be thinking in terms of what will exist in data taking
  - After a first iteration we should have a close to final product
  - Should have first prototype in rel.11 to have time to iterate

Status of ESD/AOD trigger info

- Various LVL1/LVL2 classes persistified (mainly ESD) for Rome production:
  - LVL1 : EMTauRoI, JetRoI, EtMiss
  - LVL2 : EMShowerMinimal, TrackParticles (converted from TrigInDetTracks)

http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/PESA/egamma/rome/ESDcontents.html

- This was a valid first attempt but should be revisited to find objects that are better suited to online environment: most notably with tracks
- This also spawned more realistic signatures (in custom AOD creation for now)
- Current LVL2 objects are data objects only, no navigation or hypotheses are stored in standard production
- Navigation being redesigned at the moment: some navigation information should be stored offline to allow debugging, monitoring and algorithm tuning

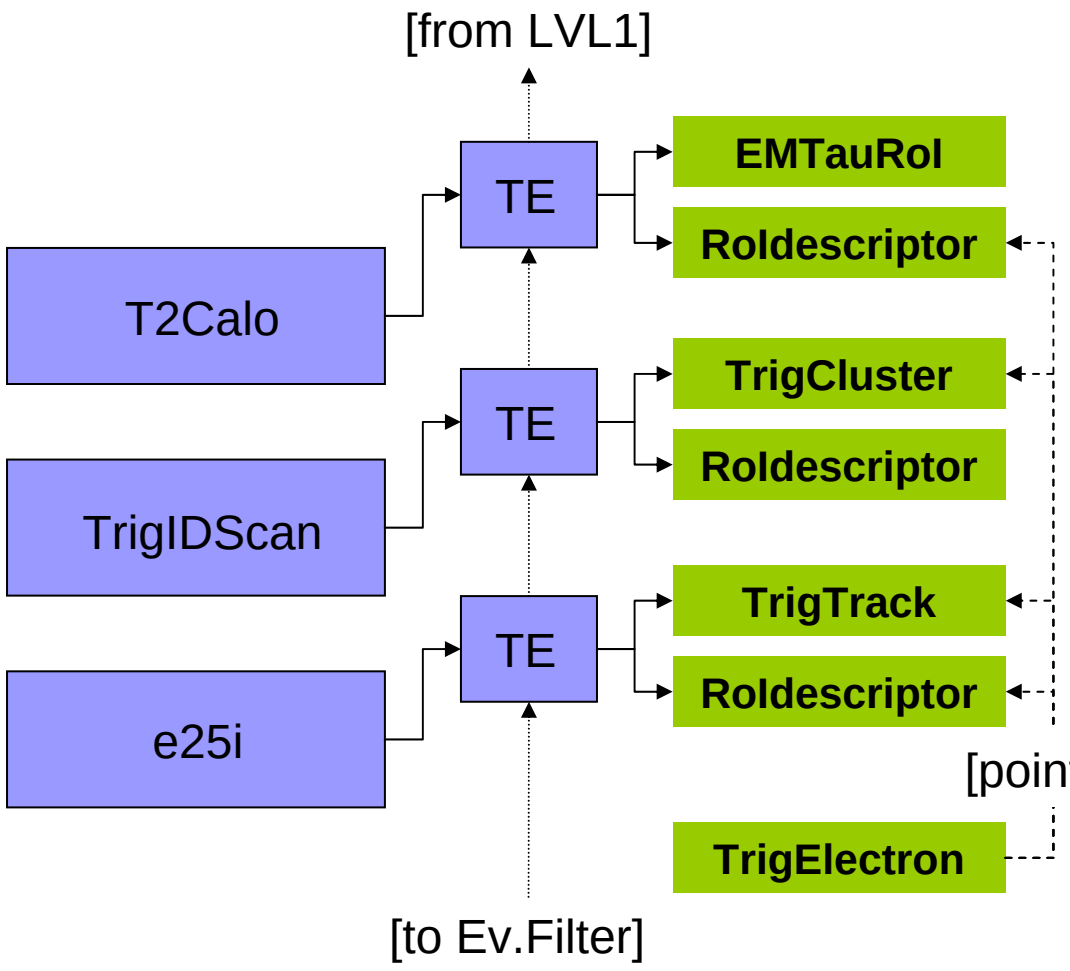
Proposal

- Provide trigger result summaries in AOD (“Yes/No” result? More?)
  - These could be the menu table stored in the run store plus trigger masks stored for each event
  - Methods would be provided such as:
    - `bool IsDefined(“e25i”)`
    - `bool IsPassed(“e25i”)`
    - `bool TriggerPassed(“L1/L2/EF”)`
- Provide “slimmmed-down” data classes produced by tracking/calorimetry/… algorithms
  - LVL1 RoI types
  - LVL2 tracks/clusters (redesign/slim down current ESD objects)
  - EF offline data classes already persistent; any new ones needed?
  - These would allow the possibility of re-running hypothesis algorithms
- Provide new objects as the result of hypothesis algorithms
  - TrigElectron, TrigTau, TrigMu…
  - These would group together tracks/clusters/RoIdescriptors etc
  - Would be a way of storing online information
- All/some of these should be designed with data taking in mind: size, complexity, dependencies, robustness

Design : data objects and hypothesis results

- To make persistency/serialization easier avoid:
  - ElementLinks
  - Inheritance
- Classes should be small and simple:
  - Maintainable and robust (minimise dependencies)
  - Size must be minimal to avoid problems for online running
- Data objects would be persistified (cluster / RoIdescriptor / Spacepoints?) – again, this assumes small number of objects stored for normal running but potential to store more information for debugging and efficiency studies
- "Hypothesis" classes (e.g. "TrigElectron") should have pointers to tracks, cluster, RoIdescriptor
  - This avoids duplication of data objects and problems from ElementLinks
  - This could be redesigned when navigation information is available and persistent/serializeable
- Mainly e/gamma and tau objects currently being defined: probably equal needs for other triggers

Design : example




```
class TrigElectron {
 public:
    TrigElectron();
...
    int nrTracks();
    TrigTrack* GetTrack(int i);
    TrigCluster* GetCluster();
    RoIdescriptor* GetRoI();
 private:
    RoIdescriptor* m_roi;
    TrigCluster*   m_cluster;
    std::vector<TrigTrack*>
m_trk;
};
```

"uses" ⟶ "seeded by" ⟶ pointer ⟶

Design : constraints

- Persistency - usual recipes from:

`https://uimon.cern.ch/twiki/bin/view/Atlas/WriteReadDataViaPool`

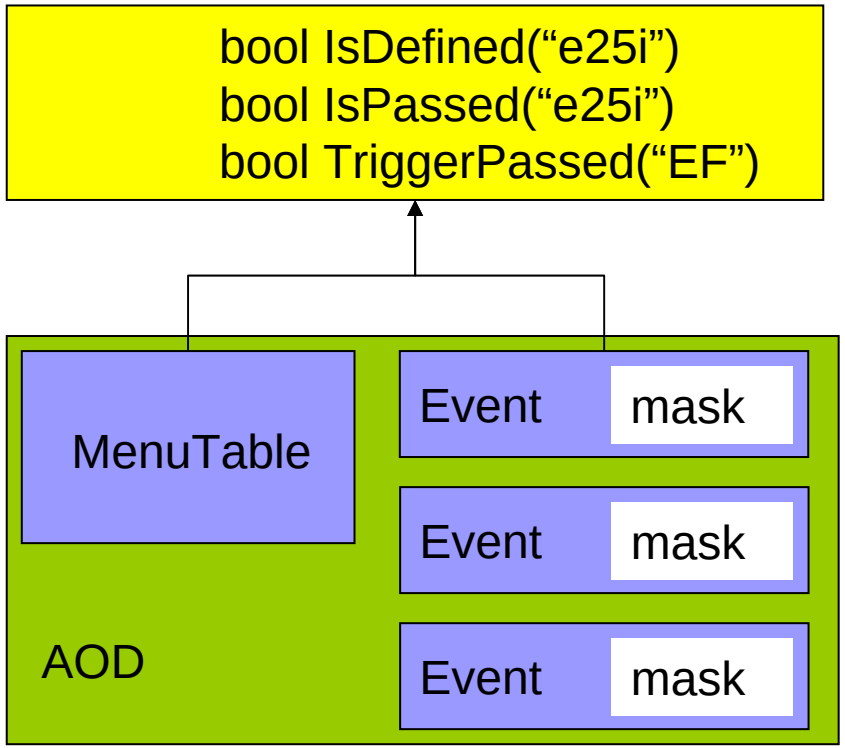
- To persistify pointers:
  - Classes should have virtual destructor (guarantee polymorphism)
  - Default constructor should initialize all data members especially pointers
  - No pointers to STL collections (not polymorphic; must be contained by value)
- To persistify classes (the usual thing):
  - Classes must have dictionary fillers: `lcgdict` pattern
  - Automatic converters must be generated: `poolcnv` pattern
- To serialize classes (Jiri Masik, LVL2):
  - Classes must have dictionary fillers as for persistency
  - Classes should contain only data members of type `int`, `float` and pointers to other classes
  - Has been demonstrated; may have to investigate serialisation of STL collections

Event Filter

- Different constraints apply for EF, as it does not seed offline reconstruction
- Only data to store is EF result
- More data could be stored for debugging and trigger studies
- EF uses offline data model: POOL converters available for data objects
- Dedicated machine or subfarm could write events to POOL files at a small rate
- This would be done for certain trigger types or at a random sampling frequency
- These events would then be transferred periodically to offline data store and used by trigger experts

Design : trigger decision

- This applies equally well to LVL1, LVL2 and EF
- Trigger decision:
  - Menu table to be stored in RunStore (may not be feasible yet)
  - Trigger masks to be stored for each event (interpreted through menu table)
  - Methods should be provided to interpret masks for each event
  - Short-term solution for Rome data would be to write methods that mimic this for the two signatures which were implemented
  - Long-term solution: menu table will be in conditions DB as it is part of the trigger configuration



Conclusions and outlook

- There's a clear need for producing a trigger AOD for both trigger and physics communities
  - Discussions started and first proposal presented here
- Design should proceed with online running in mind as well as trigger signature development, debugging, etc
- Prototype classes for TrigCluster, TrigIDTrack and TrigElectron could be done very fast
- More discussion clearly needed on storing enough information for algorithm development in POOL:
  - This is very important and would mean faster development and improved algorithms
  - But it must be balance against how much we can store
  - ESD? New, lighter data structure just for this?
- Same subjects also under discussion in muon community : common solutions should be explored when possible
- New ESD/AOD classes should be available and validated in release 11 to allow time for redesign