LVL2 ID ESD/AOD classes

Status and plans

ESD/AOD

- 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

Trigger requirements

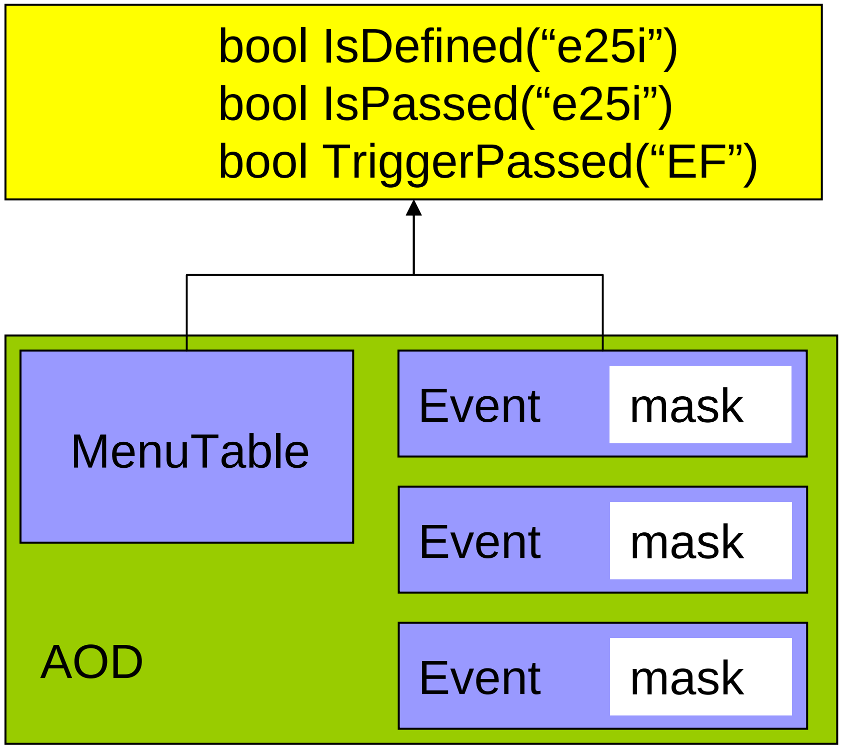
- Typical requirements on trigger data objects must include:
  - Speed
  - Size
  - Robustness
  - Maintainability
- Different uses of level 2 data classes:
  - Online: trigger processing; signatures…
  - Online: communication between LVL2 and EF
  - Offline: debugging and tuning of hypotheses
  - Offline: efficiency/rate studies
  - Offline: trigger algorithm development
- Offline uses mean storing information in Pool (ESD/AOD) and serializing information (LVL2->EF)
- It is important to maintain enough flexibility:
  - Not much information needs 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

Plans for level 2 ESD/AOD

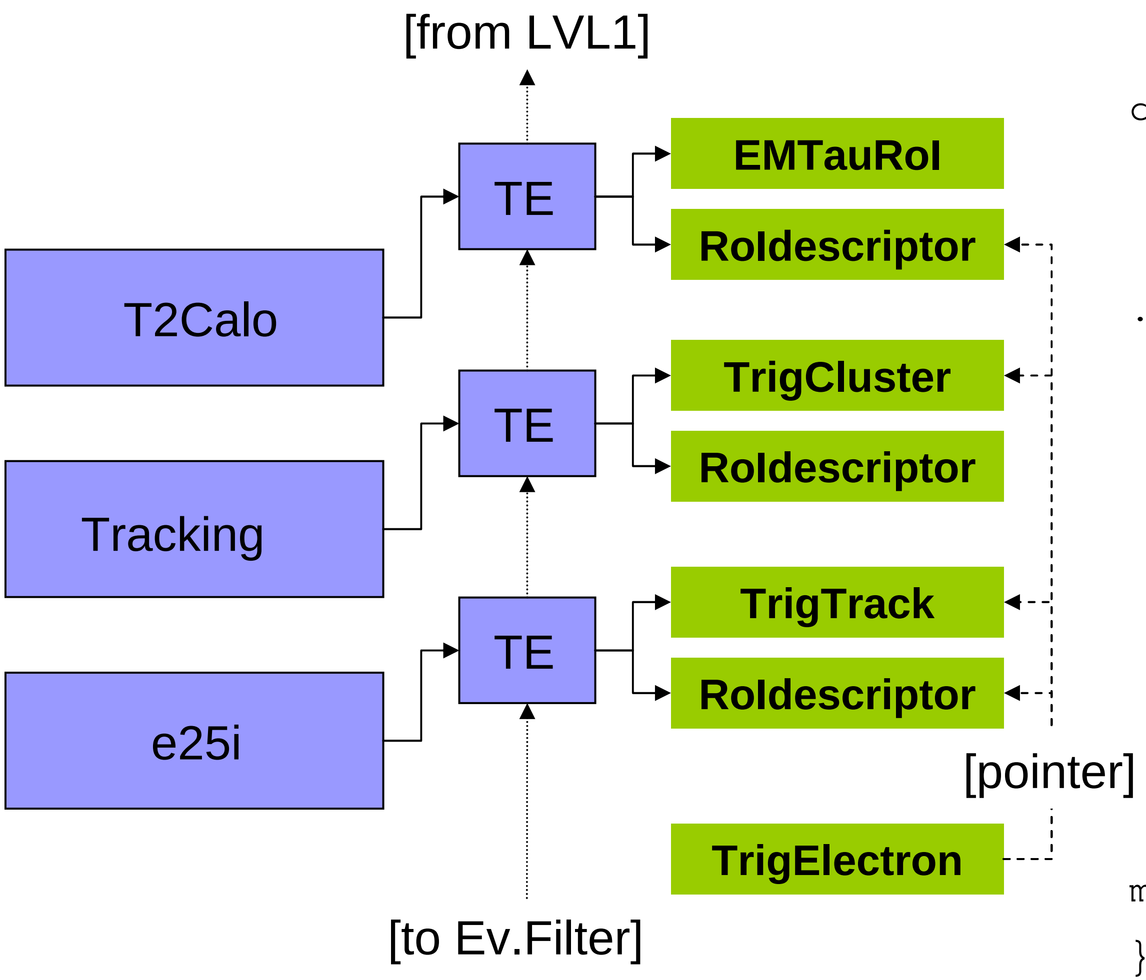
- See: `http://agenda.cern.ch/fullAgenda.php?ida=a053774#2005-07-14`
- Provide trigger result summaries in AOD ("Yes/No" result. More?)
  - These could be the menu table stored in the run store/conditions DB plus trigger masks stored for each event
  - Methods would be provided such as:
    - `bool IsDefined("e25i")`
    - `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)
  - 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

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 few signatures which were implemented
  - Long-term solution: menu table will be in conditions DB as it is part of the trigger configuration



Hypothesis result



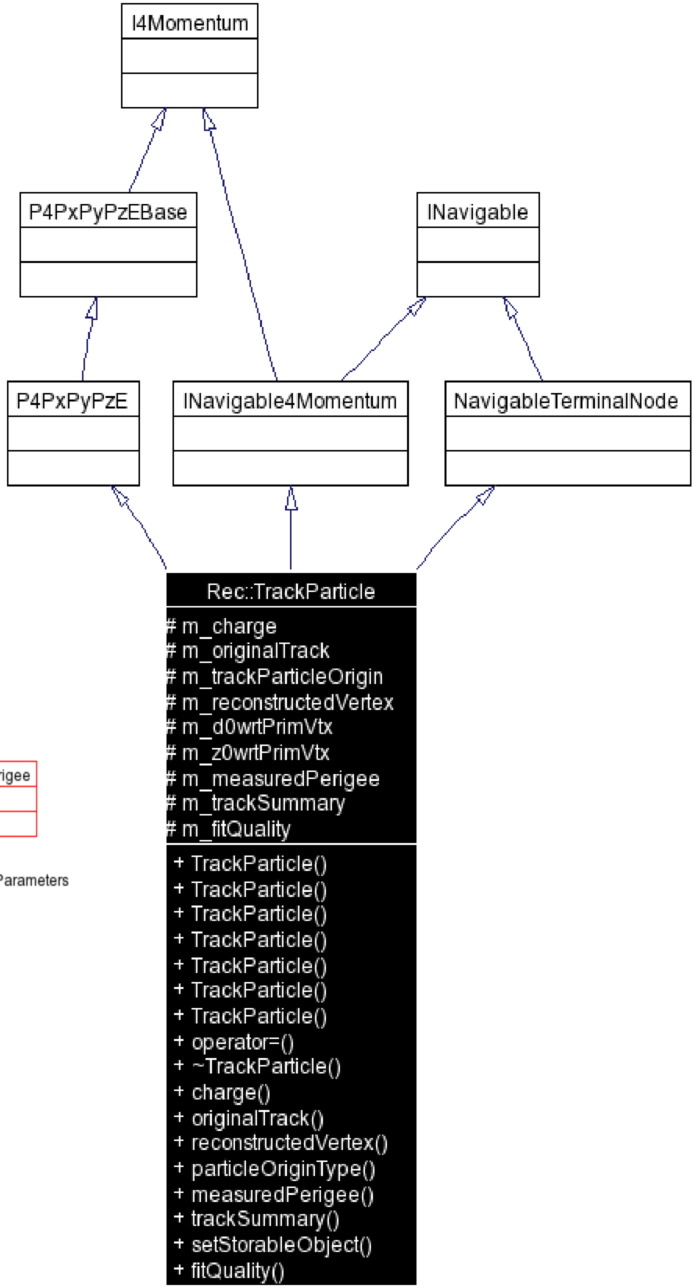

```
class TrigElectron {
 public:
    TrigElectron();
...
    int nrTracks();
    TrigTrack* GetTrack(int i);
    TrigCluster* GetCluster();
    RoIdescriptor* GetRoI();
 private:
    RoIdescriptor* m_roi;
    TrigCluster*   m_cluster;
    std::vector<TrigTrack*>
m_trk;
};
```

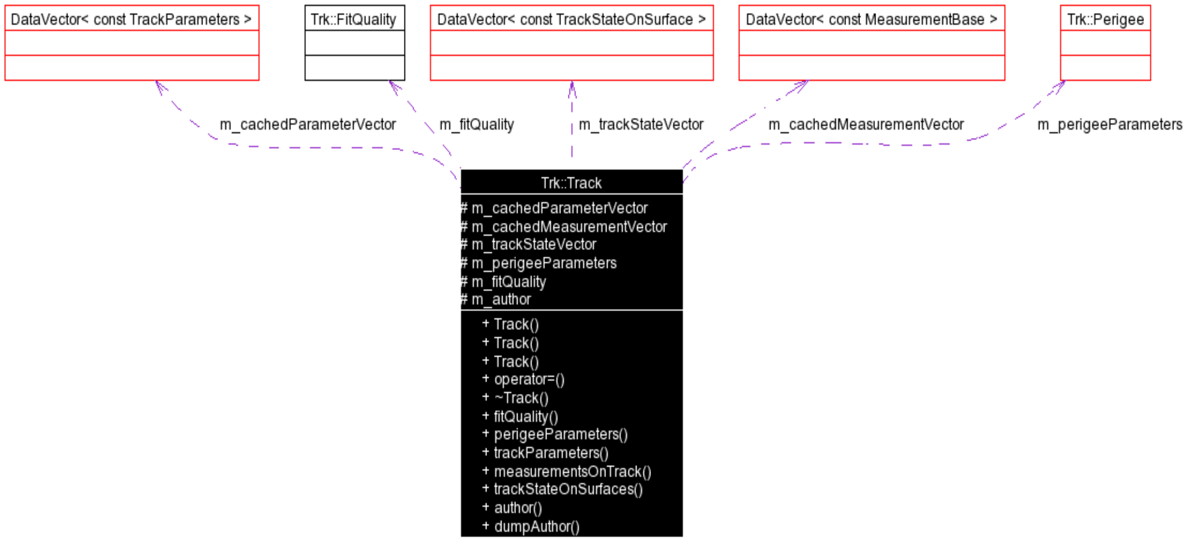
Status of ESD/AOD trigger info

- Various LVL1/LVL2 classes persistified for Rome production:
  - see : http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/PESA/egamma/rome/ESDcontents.html
  - LVL1 : EMTauRoI, JetRoI, EtMiss
  - LVL2 : EMShowerMinimal, TrackParticles
- TrackParticles are stored in ESD and AOD
- Persistency mechanisms already in place
- Converted from TrigInDetTrack
  - Using TrigT.ParticleCreator TrigToTrkTrackTool and ParticleCreatorTool
- This means creating several objects:
  - Trk::Track, Trk::TrackSummary
  - Trk::MeasuredPerigee, Trk::TrackStateOnSurface
  - Trk::TrackParameters, Trk::RIO_OnTrack, HepLorentzVector, etc
- This was a valid first attempt but should be revisited to find objects that are better suited to online environment
- Ideally aim to store same objects as are used in trigger
  - This would allow re-running of hypotheses algorithms offline

Offline

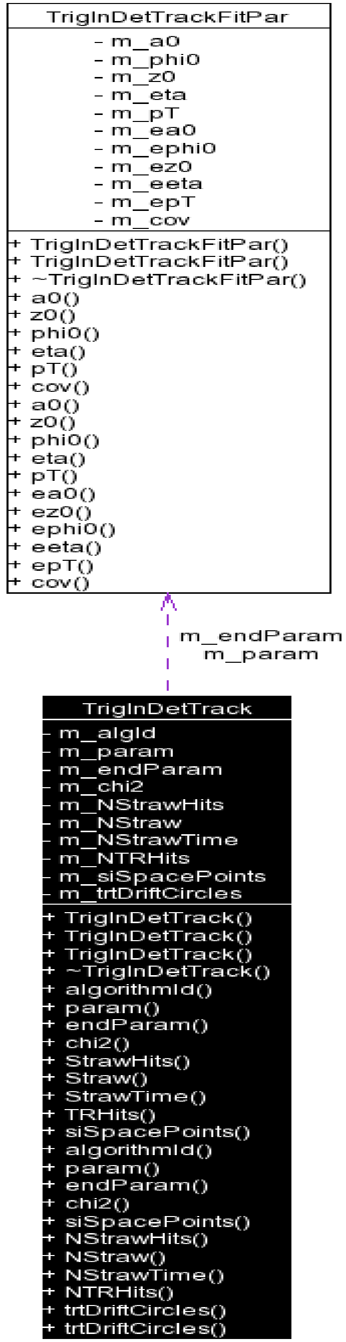
- See talk by Markus Elsing
- Offline tracking classes are flexible and can accomodate enormous complexity
- May contain only a small subset of constituent objects
- But at the cost of code complexity, large dependencies and some overheads
- E.g. a few objects pointed to by TrackParticles must be copied from constituent Trk::Tracks to maintain object ownership and avoid dangling pointers





TrigInDetTrack

- Currently produced by LVL2 tracking algorithms and used in trigger code and hypotheses algorithms
- Not great flexibility (parameters at 2 surfaces only)
- But: Simple
- No inheritance
- Few (direct) dependencies:
  - std::vector<>
  - TrigSiSpacePoint
  - TRT_DriftCircle
- Extrapolation to 2nd surface should be done by tracking algorithms
- Size: ~21 `double` + 5 `int` + 6 pointers + 30 `double` (cov. matrix) - could be even smaller if no 2nd surface
- May increase to:
  - + N*sizeof(TRT_DriftCircle) ->(more complicated)
  - + M*sizeof(SiSpacePoint) ->(~10 `double` +1 `int`)
- Should be "easy" to persistify and serialize: this would be more appropriate for LVL2 ESD/AOD



Design : constraints

- 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?)
  - This assumes small numbers 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, LAr cluster, RoIdescriptor
  - This avoids duplication of data objects and problems from ElementLinks
  - This could be redesigned when navigation information is available and persistent/serializeable (being re-designed)
- Mainly e/gamma and tau objects currently being defined: probably equal needs for other triggers

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)
  - Tested in simple case and works “out of the box”
- 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; should investigate serialisation of STL containers

Conclusions and outlook

- Design should proceed with online running in mind as well as trigger signature development, debugging, etc
- It seems a good idea to minimise complexity and dependencies to improve maintainability and ease persistency/serialisation
- Classes to be serialised need to be simple
- Ideally store same classes that are used in trigger hypotheses
- What could be stored in POOL for algorithm development ?
  - This is very important and would mean faster development and improved algorithms
  - But it must be balanced against how much we can store
  - ESD? New, lighter data structure just for this?
- Prototype “hypothesis” classes could be done soon
- Same subjects also under discussion in muon community : common solutions should be explored whenever possible
- New ESD/AOD classes should be available and validated in release 11 to allow time for redesign