Trigger ESD/AOD

Simon George (RHUL)
Ricardo Goncalo (RHUL)
Monika Wielers (RAL)

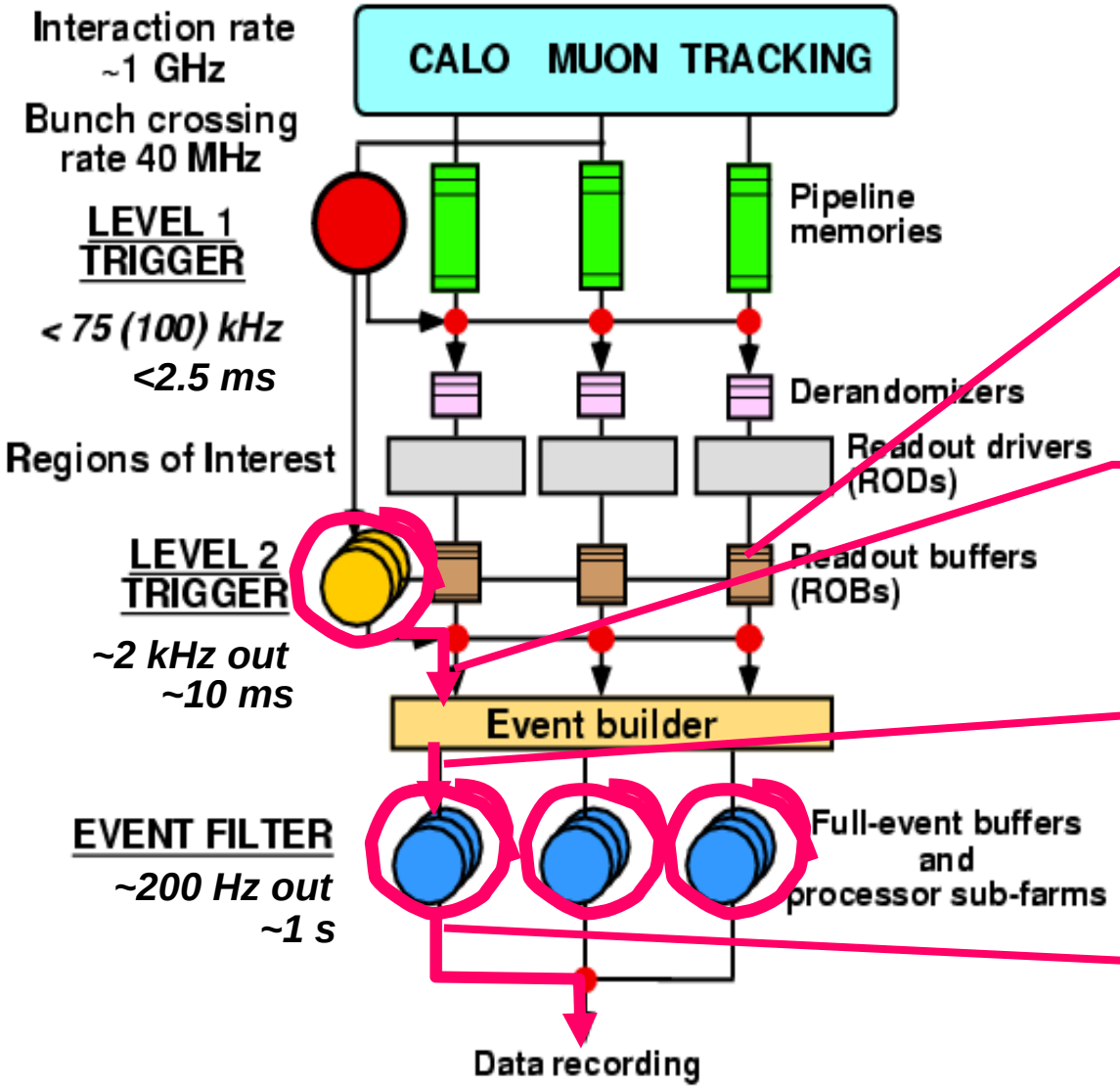
Contributions from many people.

Contents

- General interest:
  - Event data from the trigger
    - TDAQ data flow
    - Online constraints
    - Use cases
  - Inventory of ESD & AOD classes
    - LVL1, LVL2 & EF
    - Current and planned
  - How to access trigger information in Rome data

- For PAT:
  - Size estimates
  - Persistency technologies
    - Generic serializer
  - Trigger menu configuration as conditions data

Sources of event data from the trigger



TDAQ data flow

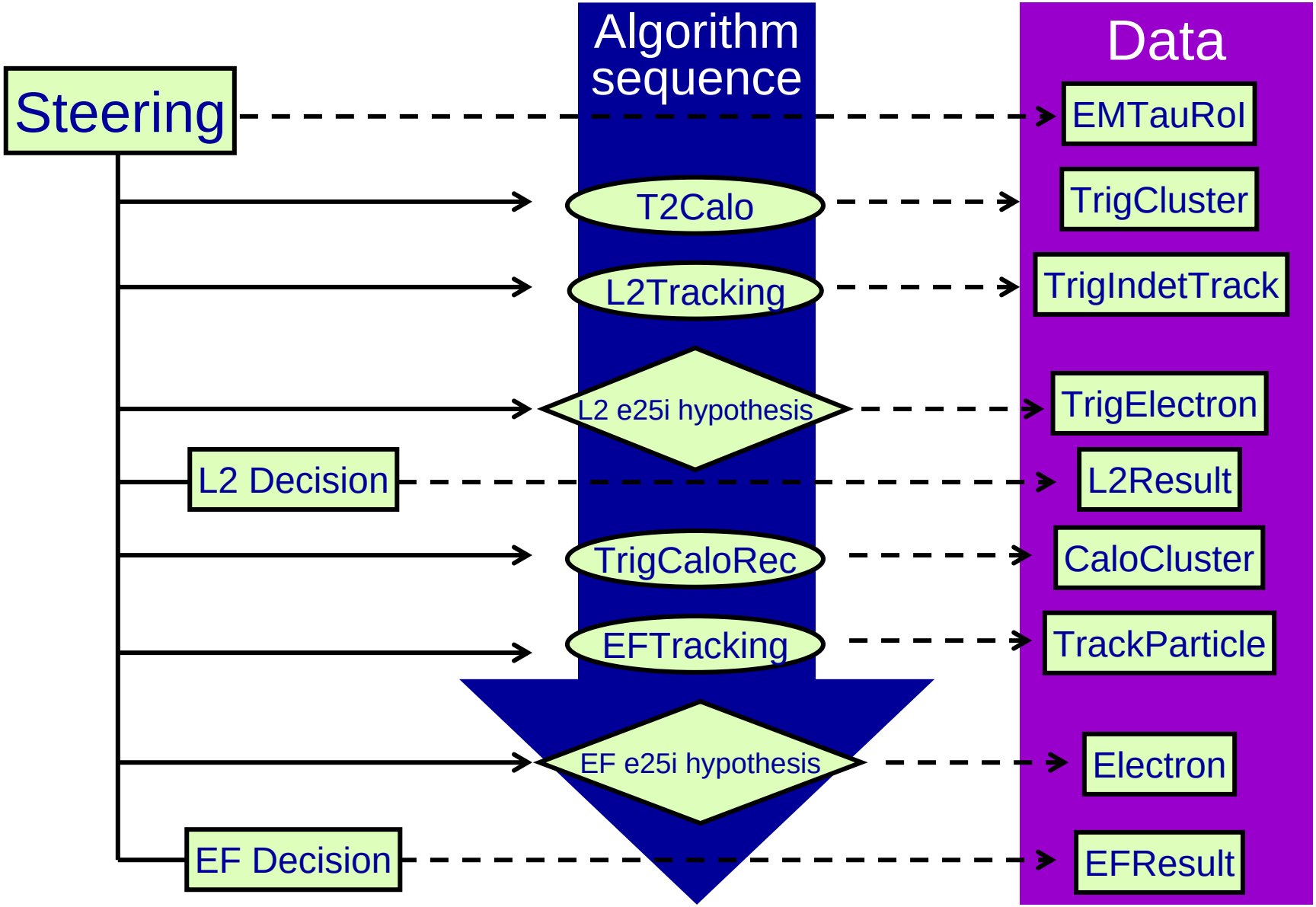
Detector data - raw event fragments

Level 2 data appended to raw event

Event passed to EF, uses LVL2 data to seed

EF data appended to raw event

HLT chain: data sources in detail – e/γ example



HLT persistency: use cases

- Online
  - L2 Result appended to raw event for purpose of seeding EF
  - Extended L2 Result appended to raw event for monitoring or debugging
  - Extended EF Result with output for calibration appended to raw event
  - L2 & EF Results appended to raw event for use offline

- Offline:
  - Store L2 & EF Results produced by HLT simulation in ESD/AOD
  - Physics analysis: get trigger decision for an event from AOD
  - Trigger performance tuning or more detailed physics analysis: re-run HLT hypotheses & decision on AOD.
  - Detailed trigger performance studies: re-run algorithms on AOD.

- The same code is run online & offline, so would like to write the same objects in both cases, but with different “persistency technology”, byte stream and POOL respectively.

Online constraints

- Size
  - Average L2Result size within 2kB/event
  - Max L2Result 64kB/event
  - Classes must be designed to convey minimum necessary data for use case.
    - E.g. use float rather than double if sufficient, bit-pack bools.
- Format
  - Objects are written out in raw event format (byte stream), not ROOT
  - LVL2 and EF may **append** a small amount of data to the **raw event**
  - This is used to pass RoI seeding information from LVL2 to EF
  - It could also be used (within constraints) to extract information for debugging, monitoring and calibration
  - Simple, generic serializer turns objects into vector<int>
- Class design
  - Serializer limits class design
    - E.g. only support float, double, int, pointer
    - Do not intend to support full offline EDM e.g. ElementLinks
    - Complex inheritance structures would cause problems

LVL1 classes in AOD/ESD (already in Rome)

- RoI classes (in ESD and AOD)
  - EMTau_ROI, JetET_ROI, JETET_ROI, EnergySum_ROI, MUON_ROI
  - ‘Hardware like’
  - Contain bit pattern of which threshold passed, eta/phi
  - Small objects: few uint32, double
- Reconstruction objects (only in ESD: also in AOD from rel.11.0.0)
  - L1EMTauObject, L1EMJetObject, L1ETmissObject
  - ‘Software like’
  - Contain energies, isolation, eta/phi quantities which are used for optimisations
  - Small objects: 4-8 doubles per class
- Future plans:
  - MUON_ROI fine, no further plans
  - Need single consolidated Calo class, which contains cluster/isolation sums + thresholds passed

LVL1 classes only in AOD/ESD

- CTPDecision (already in Rome ESD&AOD)
  - ‘Hardware like’
  - Contains word with trigger decisions
  - Contains just few words
  - Final hardware not yet decided upon (should be very soon), thus might need revision

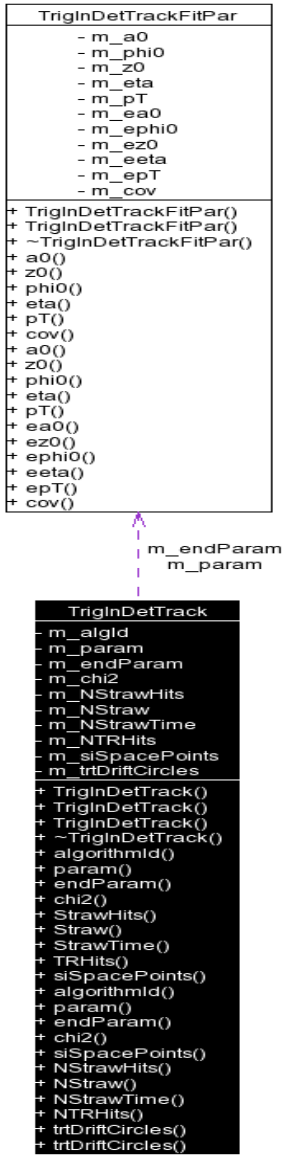
LVL1 classes only in ESD (New from rel.11.0.0)

- TriggerTower (available in release 11)
  - 'Hardware like'
  - contains for towers above threshold (in total ~7200 tower, typically only 100-200 after zero-suppression)
    - EM and had energies (final calibrated 8-bit ET values) per tower
    - Raw energy, digits, filter output per tower
    - eta/phi
  - Currently contains more data than actually read-out
    - Hardware will only give digits and final energies + eta/phi
  - Future plans:
    - further re-writing/re-design:
    - separate raw tower for internal use + stripped down calibrated tower for persistency.
- JetElement (same as TriggerTower – rel.11.0.0)
  - 'Hardware like'
  - Similar to TT but coarser granularity for jet trigger (~1k tower in total, again zero-suppressed)
  - Contains EM/had energy, eta/phi

What was available for Rome from LVL2/EF

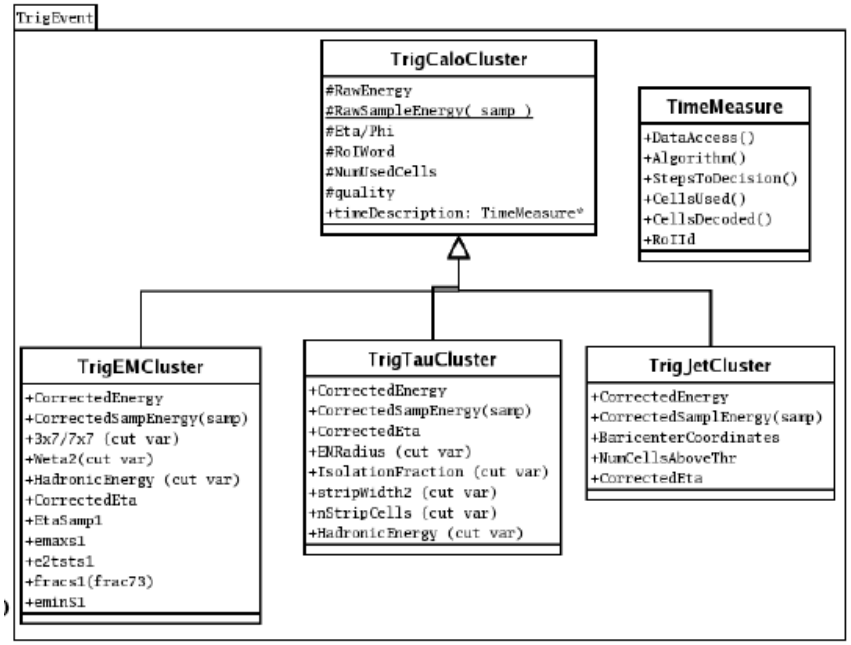
- EMShowerMinimal (ESD only):
  - Output from T2Calo
  - Contains relevant shower shapes and pointer to CaloCluster (also stored in ESD)
- TrackParticle (ESD and AOD…by accident)
  - Output from several LVL2 tracking algorithms
  - Obtained by conversion from TrigInDetTracks
  - Contains a few doubles, an ElementLink to Trk::Track, and pointers to RecVertex, MeasuredPerigee, TrackSummary, FitQuality…
- No L2Result!
- CaloCluster (only in ESD)
  - Produced by TrigCaloRec
- No other Event Filter reconstruction, so objects from offline reconstruction used instead (in ESD, AOD)
- No EFResult!

Planned LVL2 classes in AOD/ESD - I



- TrigInDetTrack
  - Inner detector track quantities
  - 21 doubles and 5 int per track
  - Plan to optionally include space points for special trigger studies
- MuonFeature
  - Muon track quantities
  - 1 int, 7 float

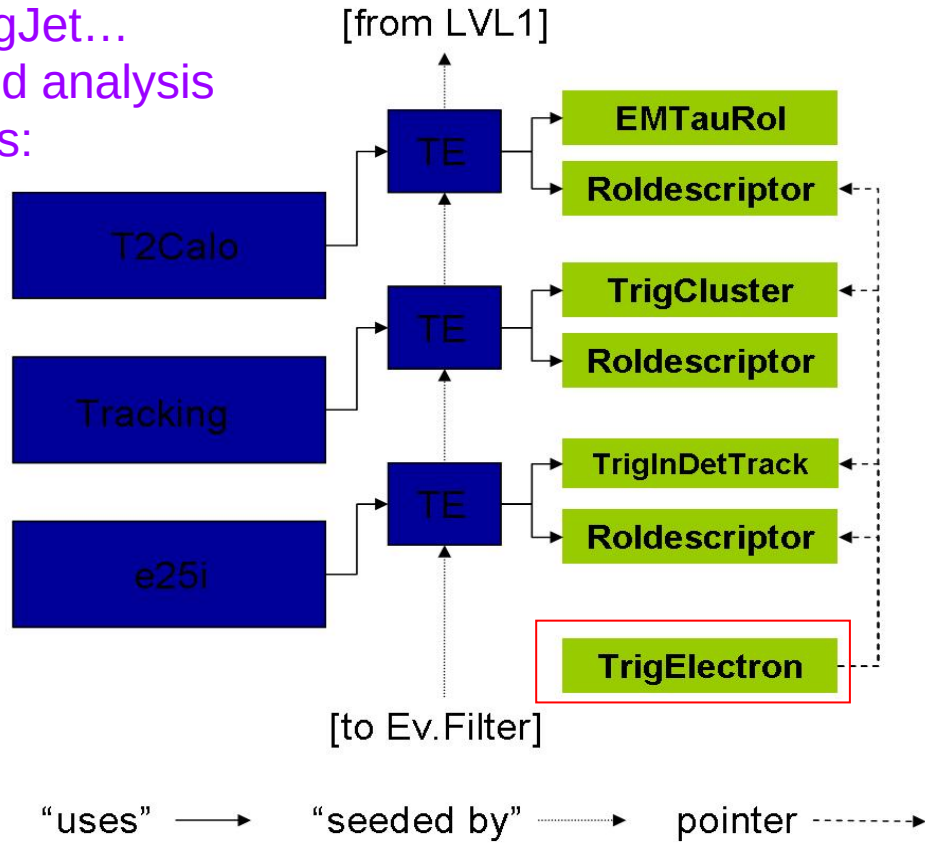
- Calorimeter classes



- Discussed last LArg week
- Classes to be implemented very soon after rel. 11, change algorithms to use new classes in 2nd step

Planned LVL2 classes in AOD/ESD - II

- TrigParticle
  - TrigElectron, TrigMuon, TrigTau, TrigJet…
  - Summary data to use for seeding and analysis
  - Example: TrigElectron data members:
    - Roi_Id
    - eta, phi
    - Z vertex
    - p_T, E_T
    - pointer to track
    - Pointer to cluster
  - Variables filled by hypothesis algo with “best values”
  - Pointers can be 0 when track & cluster not needed
  - Small object
  - Aim to have prototype in release 11



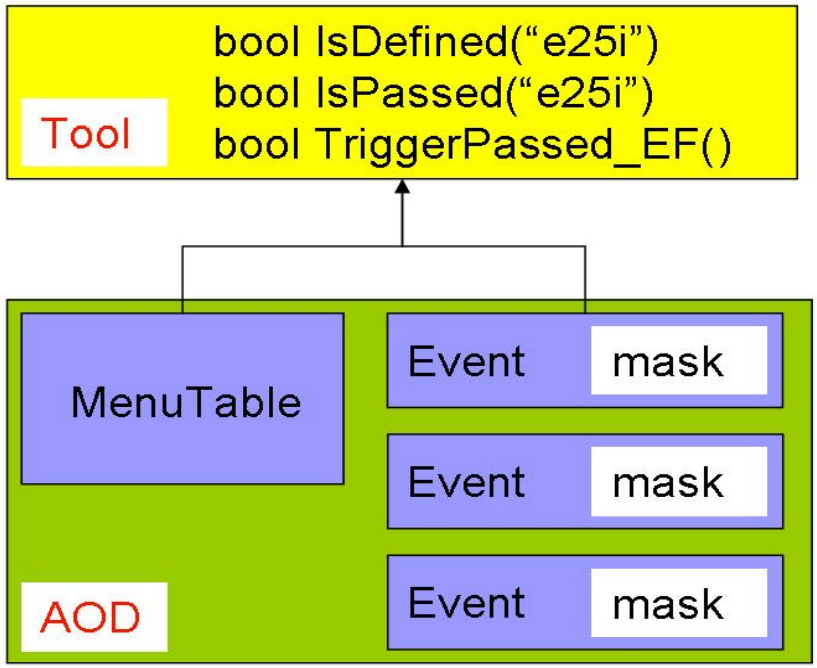
- Other classes will be added as required
  - Reflect hypothesis algorithms in the trigger
  - E.g. J/Psi, Z, di-muon

Planned EF classes

- EF reconstruction is independent from offline reconstruction
  - Same output classes but produced under different conditions
    - Algorithms run from seeds and run typically in a ‘simpler’ mode
  - Therefore need to save both EF and offline reconstruction objects
  - Store information per RoI
    - ➔less space than full offline reconstruction
  - Space requirements have to be investigated

Trigger Decision

- Planned for inclusion in ESD/AOD
- LVL2 and EF Result
  - **decision**: bit for each signature
  - navigation & steering 'state',
    e.g. needed to re-run hypothesis.
- Bit pattern interpreted through MenuTable (conditions data)
- Plan to provide a Tool to give L1, L2, EF results and signature results by interpreting bit patterns in AOD
  - See PAT discussion on accessing MenuTable

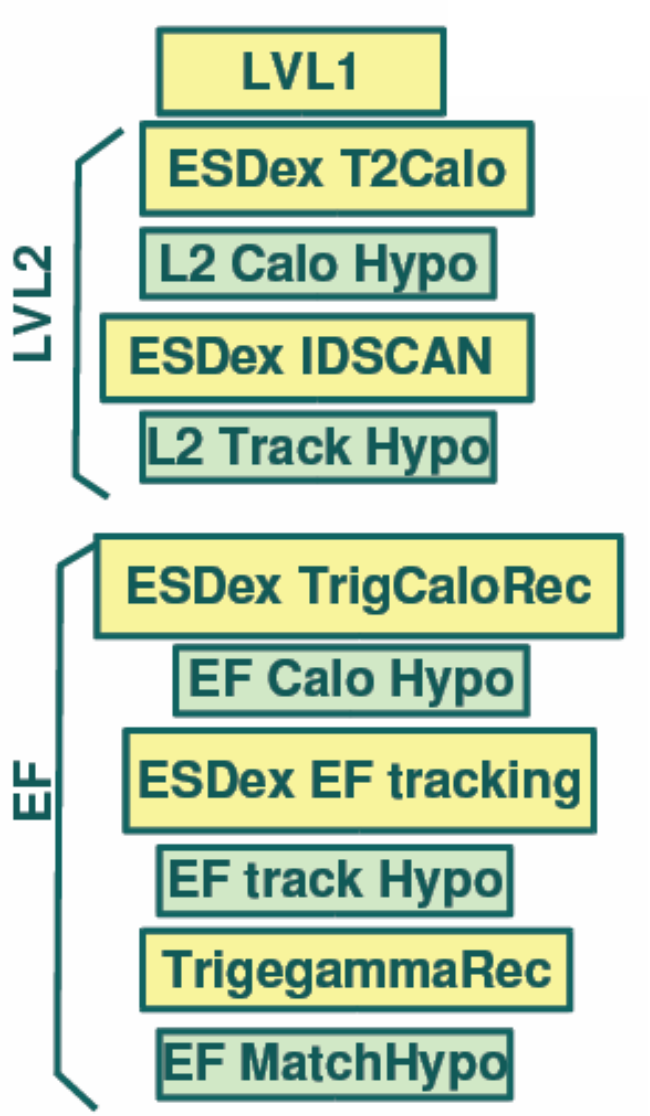


- **Short term implementation**: while there are only one or two signatures
  - First tool implementation to use this instead of MenuTable and bit pattern
  - Store map in AOD: `map<string label, bool accept>`
  - Derive trigger decisions from ESD
  - Only a few signatures - wasted AOD space by repeating labels each event is negligable
  - Aim to have prototype in release 11
- To use tool, could add map to personal AOD's made from Rome data
- **Map and bit patterns need to be included automatically in AOD in future productions**

How to use trigger decision today on Rome data

- The software described on the previous slides is not yet available. What is possible **today**?
- We currently offer 3 options to access the trigger decision when analysing Rome data:
  - Analyse ESDs
  - Create customised AODs
  - Use CBNTs and AODs
- For more information on each of the methods, look at the wiki
  - https://uimon.cern.ch/twiki/bin/view/Atlas/PesaEgamma

Analyse physics data on ESDs



- Run the HLT steering
  - Feature extraction algorithms (those that reconstruct objects, like tracks or clusters) are substituted by other algorithms that just read those objects from AOD
  - Run the real hypothesis algorithms so you can change cuts
  - Since there is no actual reconstruction, running is very fast and the job options are rather simple.
- Note: only the recolum01 Rome data contains the trigger information
- Instructions based on 10.0.1 can be found in
  - https://uimon.cern.ch/twiki/bin/view/Atlas/TrigChainOnESDs
    - How to run the e/γ slice
    - How to derive trigger efficiency
    - Solutions to known problems
- Recipe will be updated once release 11 is out

Analyse data using AODs

- 2nd method: create customised AODs
  - Make your own AOD from ESD
  - Add trigger classes in addition to 'default' classes into your AOD
  - Then one can run the trigger chain in the same way on the AODs as just described for the ESDs

- 3rd method: Bricolage
  - **Not recommended** - but we thought we should admit what we had to resort to, to get some of our results
  - Run the Root e/γ analysis program on CBNTs
  - Write out the event numbers that pass the trigger to a text file.
  - Read back text file during AOD analysis to get trigger decision
  - See Wiki page for details and limitations

Current size of trigger objects in ESD/AOD

- Average size per event using Z→ee files from Rome data at low luminosity (250 events)

L1 RoI+reco objects	~65 Bytes
L1 Trigger tower and JetElements	~5 kBytes
EMShowerMinimal + CaloCluster	~800 Bytes
TrigInDetTracks from all 4 L2 track algorithms	~2.5 kBytes

- Of course this is physics/sample dependant.

Persistency technologies 1/2

- Offline
  - POOL/ROOT is the obvious choice when running trigger software offline, so objects can be included in ESD and AOD
- Online
  - Only raw event format (“byte stream”)
  - LVL2 and EF may append a small amount of data to the raw event
  - This is used to pass RoI seeding information from LVL2 to EF
  - It could also be used (within constraints) to extract information for debugging, monitoring and calibration
- Technically
  - The steering must pass all data to be written out to the DataFlow layer in a single vector<int>. This is then packed into the raw event with the correct formatting.
  - So any classes to be written out online must be “serialized” as a vector of integers.

Persistency technologies 2/2

- Where does this leave EF output?
  - Simple data from steering (decision) is same as LVL2 and therefore easily serialized
  - No byte stream converters or serializers for offline reconstruction EDM so they will not be written out
  - Simple objects for monitoring and calibration could be serialized if suitably designed
  - Note that only use case for EF output is debugging.
    - Can re-run EF offline to emulate it (O(1s)/event)
    - **Open question**

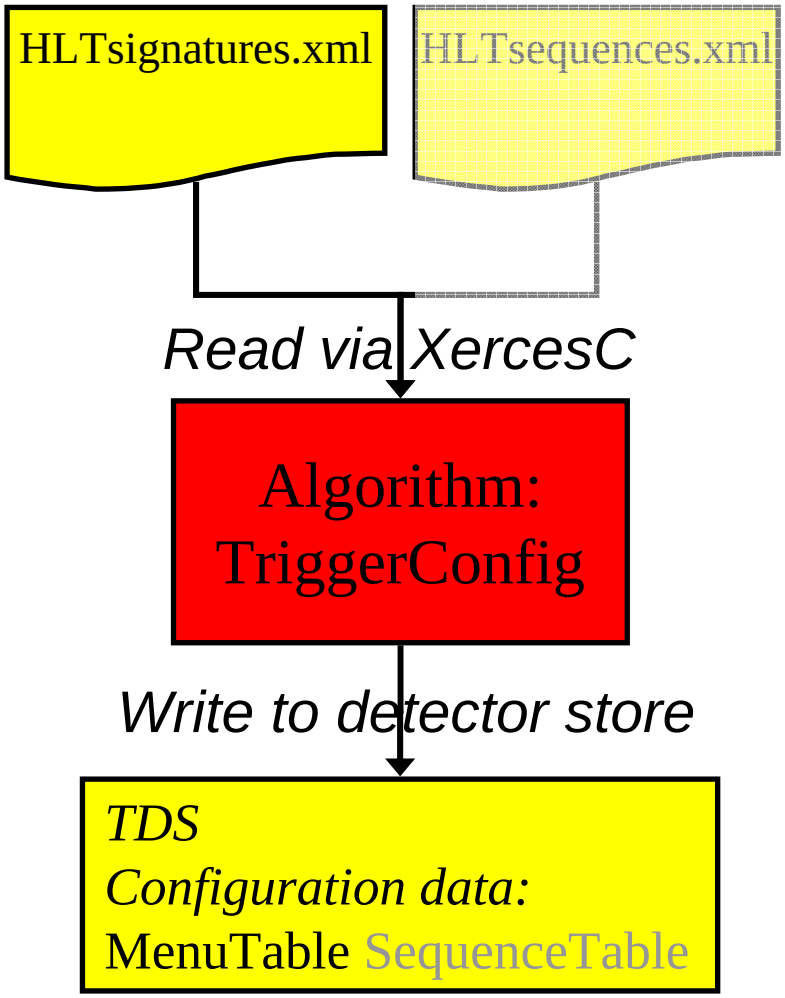
- Wrap serialised event data in POOL?
  - This is the easiest solution for the steering state and navigation data (Trigger Elements, navigation)
  - Extending this approach to the all HLT data is the easiest way to get all we need to satisfy the use case
    "Trigger performance tuning or more detailed physics analysis:
    re-run HLT hypotheses & decision on AOD."
  - Storing reconstructed objects directly in POOL poses the challenge of how to restore pointers from steering state to this data when reading back
    - E.g. Trigger element to TrigIndetTrack, TrigElectron, etc.

Generic serializer

- In previous releases, any instances of classes which inherit from ISerializable are written out.
  - Classes deriving from ISerializable must implement serialize & deserialize methods
  - Works (CTB'04) but concerned about having to write a lot of similar code and practicality of requiring inheritance for all EDM
- In release 11 we are trying a new generic serializer
- Uses SEAL Reflection
  - Same dictionaries as offline persistency
  - We are not reinventing POOL & ROOT i/o; keep it simple
  - Constraints on classes that can be serialized
    - E.g. only support float, double, int, pointers
    - Do not intend to support full offline EDM e.g. ElementLinks
    - Unlikely that complex inheritance will work
    - Select objects by following pointers
    - Select classes by class id – others will be ignored
- Short term
  - Timing study underway, initial indications are promising
  - Test with new LVL2 EDM
  - Try to support all that is needed, e.g. STL containers
- Longer term
  - Reflex migration; Schema evolution

HLT decision and associated configuration

- Decision is a bit pattern to indicate which signatures an event fulfils
- Needs configuration data to understand which signature each bit corresponds to
- Decision is created during simulation (LVL1) or reconstruction (LVL2, EF)
- At this time, configuration is read from XML files and turned into a few objects in the detector store
- Data could potentially change from run to run, by editing XML file
- Need a mechanism to access the exact same configuration when using the Decision from ESD/AOD
- **Configuration becomes conditions**



Trigger configuration persistency options

- Long term: XML files will be replaced by Trigger configuration DB
  - Any used configuration automatically referenced in conditions DB
  - Accessible through IoV mechanism
  - Assume trigger config database replicated on Grid.
- Short term: interim solution needed
- Two options have been discussed:

- 1) Save config data in POOL file, then manage by hand or with IoV Svc
  - ***Either*** keep this POOL file with the corresponding AOD file by hand, and read back in using CondProxyProviderSvc
  - Disadvantage: no IoV Svc, only works when running over AOD file(s) which use the same config data file
  - ***Or*** reference POOL file for corresponding run(s) in conditions DB
  - Either way, problem of replicating or accessing POOL files on the Grid
  - Advantage: config data is pre-computed so no time penalty when running over AOD.

- 2) Save XML as string in conditions DB
  - modify code to take XML as string instead of file
  - re-produce config data in TDS
  - disadvantage: time to reproduce config data from XML every run change, could significantly slow down running on AOD

- Thanks to Richard H, RD, David M, and A-Team for input.
  I hope I summarised it correctly.