Argonne NATIONAL LABORATORY

Persistent Data Layout and Infrastructure for Efficient Selective Retrieval of Event Data in ATLAS

•Peter van Gemmeren (ANL)

David Malon (ANL)

Meeting of the Division of Particles and Fields of the American Physical Society

U.S. DEPARTMENT OF ENERGY

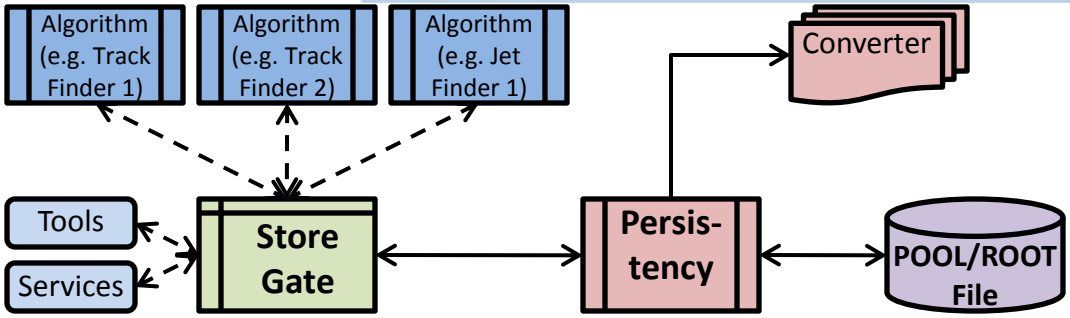
Outline:

- ATLAS Software & Event Data Model
- POOL / ROOT Persistency Framework
- ROOT Object Streaming & Splitting
- History of Storage Layout
- Mismatch of Transient vs. Persistent Event Data Layout
- Changes:
  - No single attribute retrieval
  - Almost event level retrieval
- Performance Measurements:
  - Reading all events sequentially
  - Selective reading 1% of events
- Summary

Abstract

- The ATLAS detector at CERN has completed its first full year of recording **collisions at 7 TeV**, resulting in **billions of events** and **petabytes of data**.
  - At these scales, physicists must have the capability to read only the data of interest to their analyses.
- ATLAS has developed a **sophisticated event-level metadata infrastructure** and **supporting I/O framework**.
  - The ultimate success of such a system, however, depends significantly upon the efficiency of selective event retrieval.
  - Supporting such retrieval can be challenging, as ATLAS stores its event data in **column-wise orientation** using **ROOT TTrees** for a number of reasons, including compression considerations, histogramming use cases, and more.
- For 2011 data, ATLAS will utilize **new capabilities in ROOT** to tune the persistent storage layout of event data, and to significantly speed up **selective event reading**.

ATLAS Software & Event Data Model

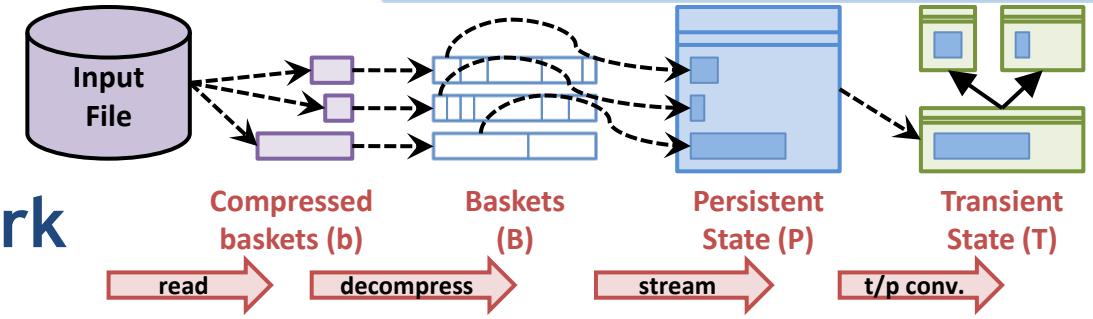


- **Simulation**, **reconstruction**, and **analysis** are run as part of the **athena framework**:
  - Using the most current (transient) version of the **event data model**
- Athena software architecture belongs to the **blackboard family**:
- **StoreGate** is the Athena implementation of the blackboard:
  - Allows a module to use **transparently** a data object created by an upstream module or read from disk.
  - A proxy defines and **hides the cache-fault mechanism**:
    - Upon request, a missing data object instance can be created and added to the transient data store, retrieving it from persistent storage on demand.
  - Support for **object identification** via data **type** and **key** string:
    - Base-class and derived-class retrieval, key aliases, versioning, and inter-object references.

Event Selection using TAGs

- TAGs are **event-level metadata** records.
- TAGs contain event attributes chosen to support efficient identification and selection of events of interest to a given analysis.
  - E.g.: Trigger, event properties (like missing ET), particle data (jets, electrons, muons, photons…).
- TAG data can be stored in **ROOT files** and uploaded into a **relational database**.
- TAGs allow jobs to process only events that satisfy a given predicate.
  - Less complex predicates can be applied as SQL-style queries on the TAG database using a web browser interface or on TAG files directly.
  - For more complex criteria, C++ modules can be written using the TAG attributes for event selection.
- No payload data is retrieved for unselected events.
  - Data files containing only unselected event are not accessed.

POOL / ROOT Persistency Framework



- ATLAS offline software uses **ROOT I/O** via the **POOL persistency** framework, which provides **high-performance** and **highly scalable** object serialization to **self-describing**, **schema-evolvable**, **random-access** files.
- POOL software is based on a **hybrid approach**: Combines **two technology types** into a **single consistent API** and storage system:
  1. **Object streaming** (e.g., **ROOT I/O**), addresses **persistency for complex C++ objects** such as high energy physics event data.
     - In practice, **ROOT I/O** is the only technology currently supported by POOL for object streaming into files.
  2. **Relational Database** (e.g., MySQL), provides **distributed**, **transactionally consistent**, **concurrent** access to data that may need to be updated.

ROOT Object Streaming & Splitting

- State of the objects is captured by special methods, **streamers**:
  - A streamer **decomposes** composite data objects into their member data and calls the streamers of all base classes and for all object data members.
    - Ultimately, **only simple data types** are written containing all data of an object.
- When ROOT writes objects to TTrees, **splitting of data members into separate TBranches** can be controlled by setting the split-level.

  Split-level 0: causes the whole object to be written entirely to a single TBranch.

  Split-level 1: will assign a branch to each object data member.

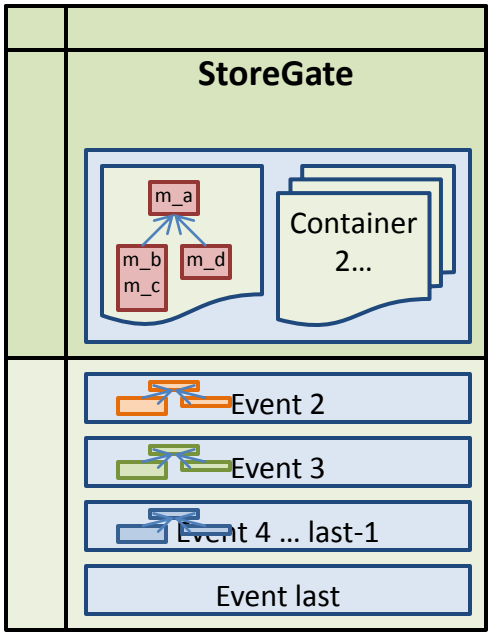
  Split-level 2+: composite data members are also split.
- When splitting TTrees, each TBranch will have its own buffer that can be **compressed individually**.
  - A split TTree has smaller **size on disk**, can be **faster** to read, but requires **more memory**.
  - Data members contained via pointer cannot be split.

History: Storage Layout

- When ATLAS started using POOL/ROOT, we wrote a **separate TTree** for each **StoreGate Collection** and did **not use splitting**.
  - At that time, POOL did work only for Container matching to TTree.
  - Before Transient-Persistent Separation, data was stored in DataVectors, which contain Pointer to Objects, which couldn't be split by ROOT.
- After **Transient-Persistent Separation**, ATLAS designed persistent objects that could be split and used the **default split-level of 99**.
- When POOL learned how to create Container as **TBranches** of a TTree, ATLAS decided to assemble most of our **event data in a single TTree**.
- **Basket sizes** were left up to POOL/ROOT to use a POOL **default of 16KB**, until **memory constrains** forced ATLAS to lower it to **2KB** for all baskets.
  - For 2011 data so far ATLAS is using a new ROOT feature to optimize Baskets in the main event **TTree to share 30MB**.

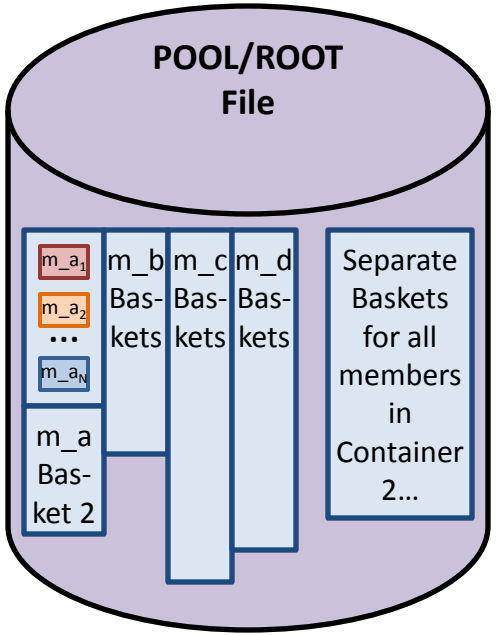
Mismatch:
Transient vs. Persistent Event Data Layout

- **Event-wise oriented**.
  - DataObject / Container retrieval on demand.
    - No partial object or data member retrieval.
- **Column-wise oriented**.
  - Size of column depends on split-level:
    - Split-level 99, split object down to individual member.



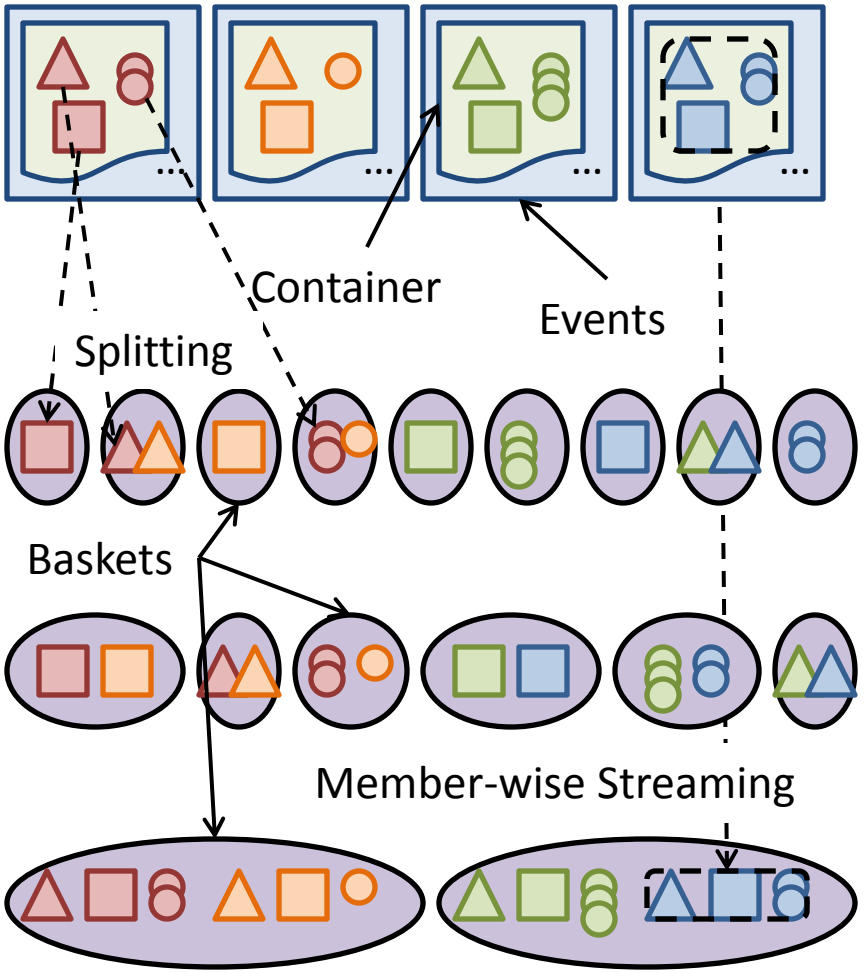
← **StoreGate EventStore holds one event only.**
  - Cleared after each event is processed.

→ **ROOT compresses data for many events into common baskets**.
  - Completely read and uncompressed when single event is retrieved.



1st Change: No single attribute retrieval

- Switched **splitting off** while retaining **member-wise streaming**.
  - Each collection (or POOL container) will be stored in a single Basket.
  - Except for largest container:
    - New feature in POOL allows custom split-level.
- This decreases the number of baskets from ~10,000 to ~800, and therefore increases their average size >x10.
  - Therefore lowers the number of total reads.



2nd Change: Almost event level retrieval

- **Current**: With 30 MB optimized TTrees, **~50 – 200 entries** (events) will share the same basket.
  - For selective reading of a single event, 30 MB of data have to be read from disk and uncompressed.
  - **Past**: For fixed sized 2 KB baskets, the number of entries varies to a large degree. To read a single event, >10K baskets -> ~5-10 MB had to be read from disk and uncompressed.
- Use **automatic basket optimization** to write **fewer events per basket**, by flushing every **5 events for ESD** and every **10 events for AOD**.
  - Increase size for baskets outside the main event tree to 32K (ROOT default):
  - Switch off basket optimization for all auxiliary container.

Details for Changes

- For ESD:
  - Trk::TrackCollection#Tracks
- And for AOD:
  - CaloClusterContainer#CaloCalTopoCluster
  - Rec::TrackParticleContainer#TrackParticleCandidate
- Container are very large and do not gain by streaming member-wise, therefore they remain being written in split mode.
  - Avoids file size increase.
- Part of main event tree, therefore flushed every 5/10 events.

1st Performance measurements: Reading all events sequentially (ESD and AOD)

- This use case is most often considered.

1. **Current**: Full splitting and 30 MB to optimize main event tree:
   - Total read **ESD**: **420 ms/event**
   - Total read **AOD** : **55** (+/- 3) **ms/event**
2. **New Storage Layout**: No splitting and flushing of main event tree every 10/5 events:
   - Total read **ESD**: **360 ms/event**
   - Total read **AOD**: **35** (+/- 2) **ms/event**

- Reading all events is **~15 - 30% faster**
- Fewer reads, 10-30 x less branches only ~10 x more baskets.
- Better sequencing of baskets in file.

2nd Performance measurements: Selective reading 1% of events via TAGs (AOD only)

- This use case is the main motivation for changing the storage layout.

1. **Current**: Full splitting and 30 MB to optimize main event tree:
   - Total read **AOD**: **270 ms/event**
     - Per event reading is ~5 times slower than reading all events. However, total read time is reduced by almost a factor of 20.
     - A 1% selection ends up reading and uncompressing most data.
2. **New Storage Layout**: No splitting and flushing of main event tree every 10/5 events:
   - Total read **AOD**: **60 ms/event**
     - Per event reading is 50-100% slower than reading all events (almost as fast as with the current layout). Total read time is reduced by more than a factor of 50.
     - A 1% selection ends up reading and uncompressing less than 10%.

- Reading (1%) selected events is **~4 - 5 times faster**.

Performance Details for DataObjects

- **Large** containers are not changed and do not gain (or lose) in I/O performance.
  - These container already performed well.
- **Very small** DataObjects did not see a performance gain either.
  - Small DataObjects tend to have only few data objects, so there is limited gain by streaming member-wise.
- Medium sized container show the **biggest improvements**:
  - Analysis::MuonContainer ~3 x faster
  - TrigMuonEFInfoContainer ~3 x faster
  - Trk::SegmentCollection ~3 x faster
  - egammaContainer >2 x faster

Performance Gains for Selective Reads

- When reading via TAGs, all DataObjects show great improvements.
- For this study, TAGs were only used as the mechanism to trigger selective reading.
- There are many use cases for selective reading of events or DataObjects / Containers:
  - The user may need some information only for a subset of events:
    - E.g.: A slepton search may need to use the lepton containers only if there is missing energy greater than a threshold.
    - In many of these cases using TAGs can be beneficial, as they were designed to enable efficient event selection.
      - E.g.: One can select events with a certain number of jets, without having to read the Jet container (for the rejected events).
  - When using the **multi-processor athenaMP** framework, multiple worker processes, each read only a non-sequential part of the input file.

Further Performance Results

- **File Size** is **very similar** to old and current format.
- **Virtual Memory** foot print **reduced by about 50-100 MB** for writing and reading new data:
  - Fewer baskets loaded not memory.
- **Write Speed** has **increased** by about **20%**.
  - The write speed was increased even further (almost 50%), as the compression level was relaxed.

Fall 2011 Reprocessing

- This month ATLAS started to reprocess all data taken in 2011, using release 17 software, which deploys the new storage layout:
  - No splitting and flushing of main event tree every 10/5 events
- Until end of August, ATLAS Tier 0 continues to process data using the current release 16 software:
  - Full splitting and 30 MB to optimize main event tree
- By end of the reprocessing campaign, all 2011 data will have been reprocessed using the new storage layout and I/O performance benefits will be available to reader of ESD and AOD.

Summary

- Changes to the event data storage layout:
  - No ROOT splitting, but member-wise streaming.
  - Optimize ROOT baskets to store small number of events.
- Improve I/O performance:
  - Reading is ~30% faster.
  - Selective Reading is 4-5 times faster (for 1%)
- To be deployed in fall reprocessing.

	Read all events	Read 1% of events
AOD 1 split, 30MB TTree	55 ms/ev.	270 ms/ev.
AOD 2 un-split, flush 10 evt.	35 ms/ev.	60 ms/ev.
Difference	**~30 % faster read**	**4-5 times faster read**

vs. **OLD** ROOT TTree