

# Considerations on ROOT output layout to support fine-grained event data caching.

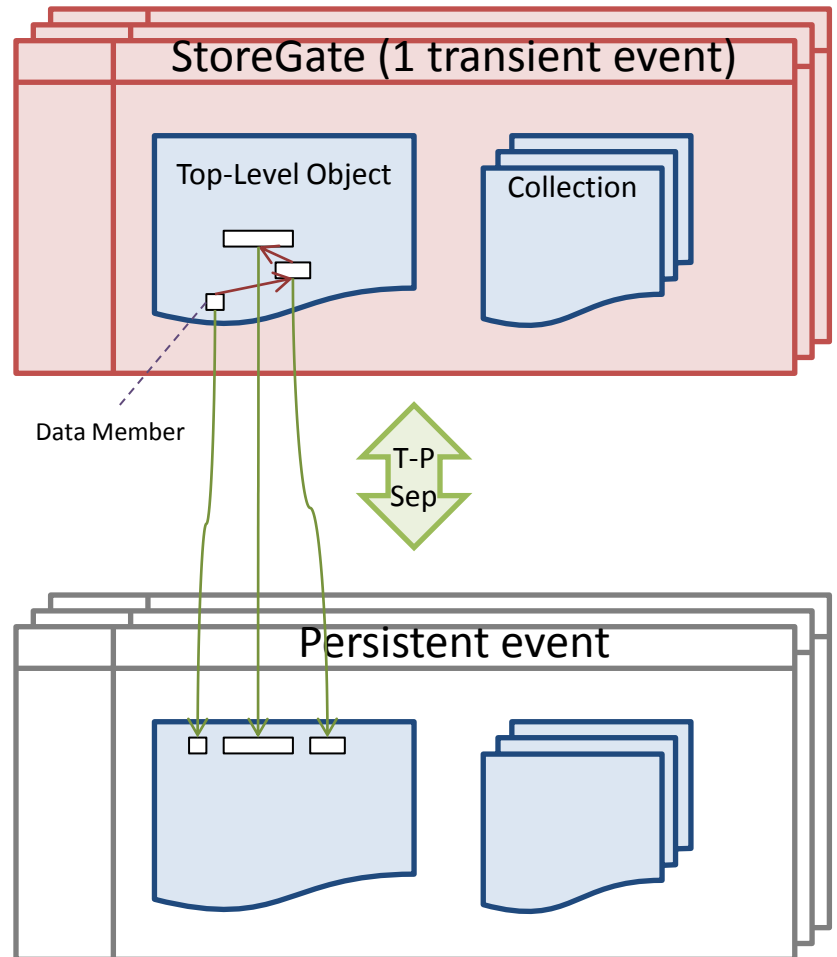
1. StoreGate vs. split ROOT TTree with 2K Baskets
2. New output settings
3. performance, especially for event selective reading
4. event caching: with  $\sim 10,000$  2K baskets, even 10MB are not sufficient to store a complete single event (150K), A 30MB optimized TTree is better, but will have  $>30$ MB or hundreds of events as caching granularity. Un-split TTrees, optimized to 10 entries should give the best performance.
5. Interaction with TTreeCache has to be studied.

**Work in progress...**



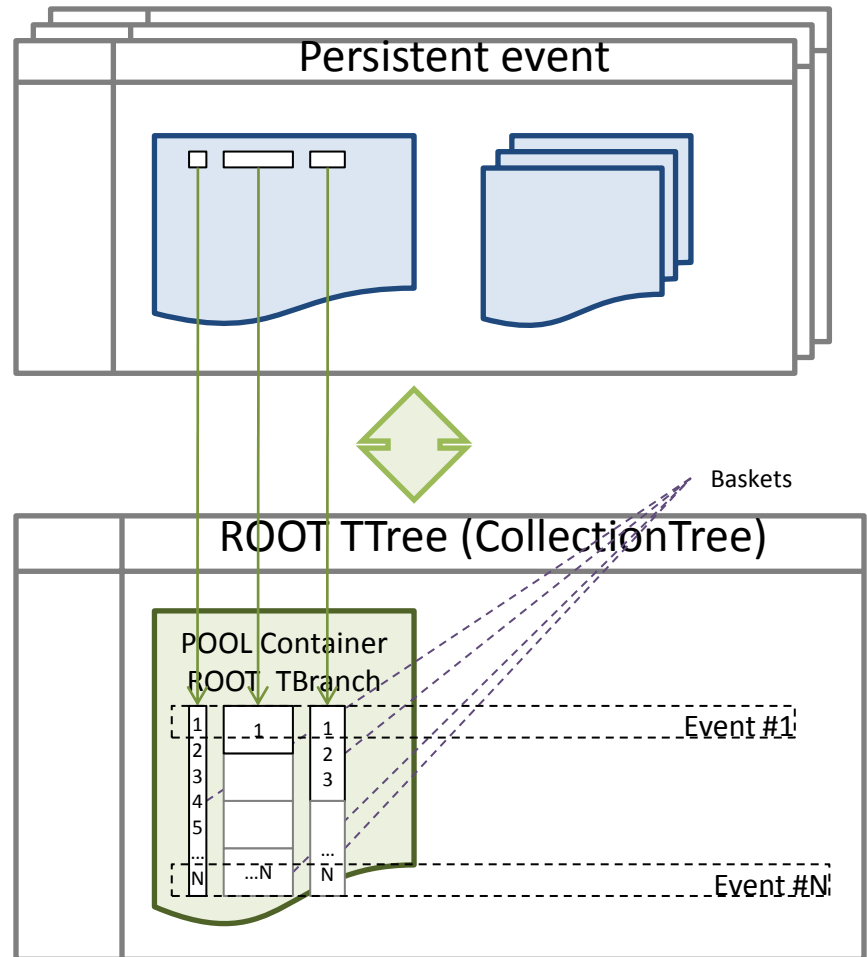
# Transient event data layout

- Event-wise oriented.
- DataObject / Container retrieval on demand.
- No partial object or data member retrieval.
  - E.g.: One cannot just retrieve the event number, but has to retrieve complete EventInfo object.
- Transient Objects are converted to Persistent Representation before writing to POOL/ROOT.
  - Significantly less complex C++ structure optimized for persistency:
    - Few methods, no pointers , less inheritance.



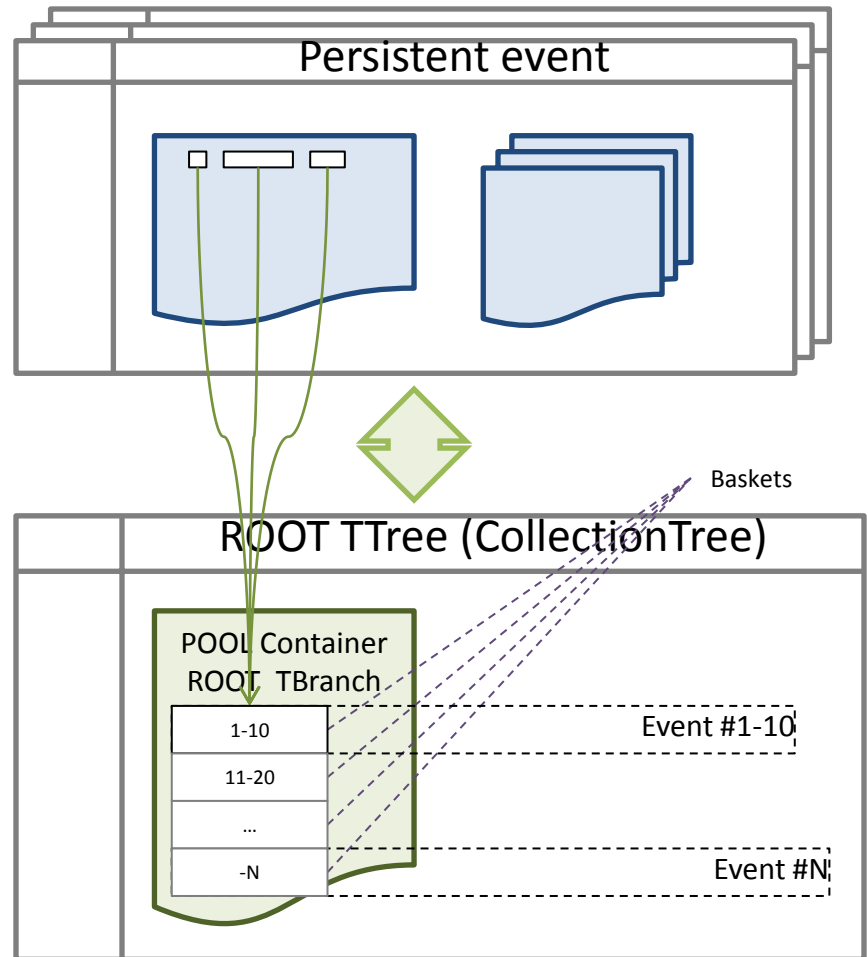
# Persistent POOL/ROOT data layout

- StoreGate Objects/Collections are written to POOL container which correspond to TBranches in ROOT.
- Column-wise oriented.
  - Size of column depends on split-level:
    - Split-level 99, split object down to individual member.
- Compressed in baskets over many events.
  - Size of Basket can be set or (new!) ROOT can optimize Basket size so that all hold about the same number of events.
- So far the basket size was limited by the available memory.
  - With split-level 99 and hundreds of container, there are ~10,000 baskets.



# New Persistent POOL/ROOT data layout to support efficient single event access and caching.

- **No single attribute** retrieval:
  - Switched **splitting off** while retaining **member-wise streaming** (another recent ROOT addition).
    - For the very largest container keep splitting to avoid file size increase
- Almost **event level** retrieval:
  - Use **automatic basket optimization** to write fewer events per basket, by **flushing every 10 events** and increase the size for baskets outside the CollectionTree to **32K**.
    - With 2 kb Baskets a varying number (average ~100) of entries are stored.
    - With a 30 MB optimized TTree 100-200 entries are in each basket.



# Event Caching and Event Selective Read (e.g. TAGs)

- When splitting all data and putting them in 2KB baskets, one would have to cache 20MB of data.
  - Still would not be guaranteed to have the complete event.
    - But lots of data for neighboring events would be cached.
- With a fully split 30 MB optimize TTree (baskets should contain ~ the same number of events), one would have to cache at least 30 MB
  - Granularity of several 100 events.
- For sub-file or event level caching, the requirements for the storage layout are similar to the event selective reading use case.

	Read all events	Read 1% of events
	all data objects ~19K AOD	all data objects
<b>AOD 1</b>	823 s	50803 ms
split, 30MB TTree	42 ms/ev.	270 ms/ev.
<b>AOD 2</b>	665 s	11233 ms
un-split, 10 events/basket	35 ms/ev.	60 ms/ev.
<b>Difference</b>	<b>20 % faster read</b>	<b>4-5 times faster read</b>
All performance number should be considered random until verified independently		

# Outlook

- Athena/POOL has framework to allow tuning persistency parameter and optimizing the POOL/ROOT layout for various use-cases.
- Output data layout is an important consideration to enable efficient sub-file level event data caching.
  - Requirements similar to event selective reads.
- ROOT has added many I/O capabilities that can be utilized by Athena.
- Studies have been done for no-split, small number of events/basket which show drastic improvements for selective read speed.
  - Should be beneficial for event caching
- Further study, including TTreeCache have to be done.
  - Work should be synchronized between ADC R&D and Core development.

