ATLAS: ROOT I/O for multithreaded Athena

ROOT I/O Workshop, June 20th 2018

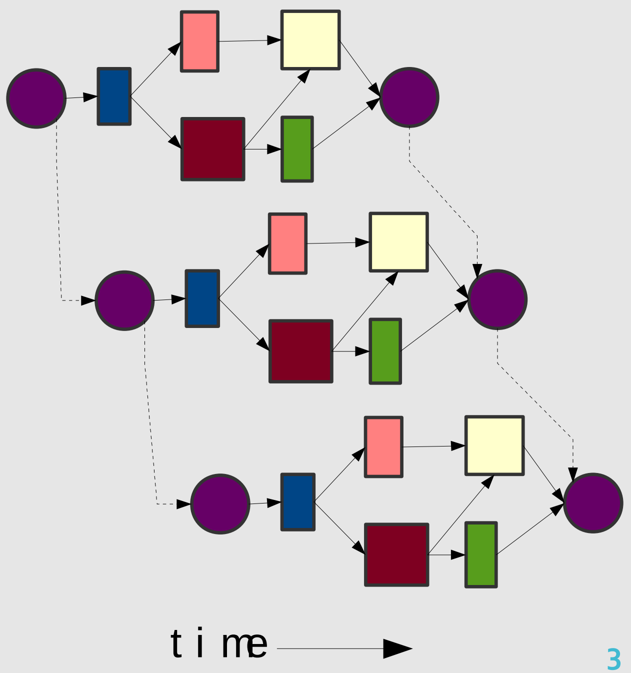
Peter van Gemmeren (ANL), Marcin Nowak (BNL)

Outline

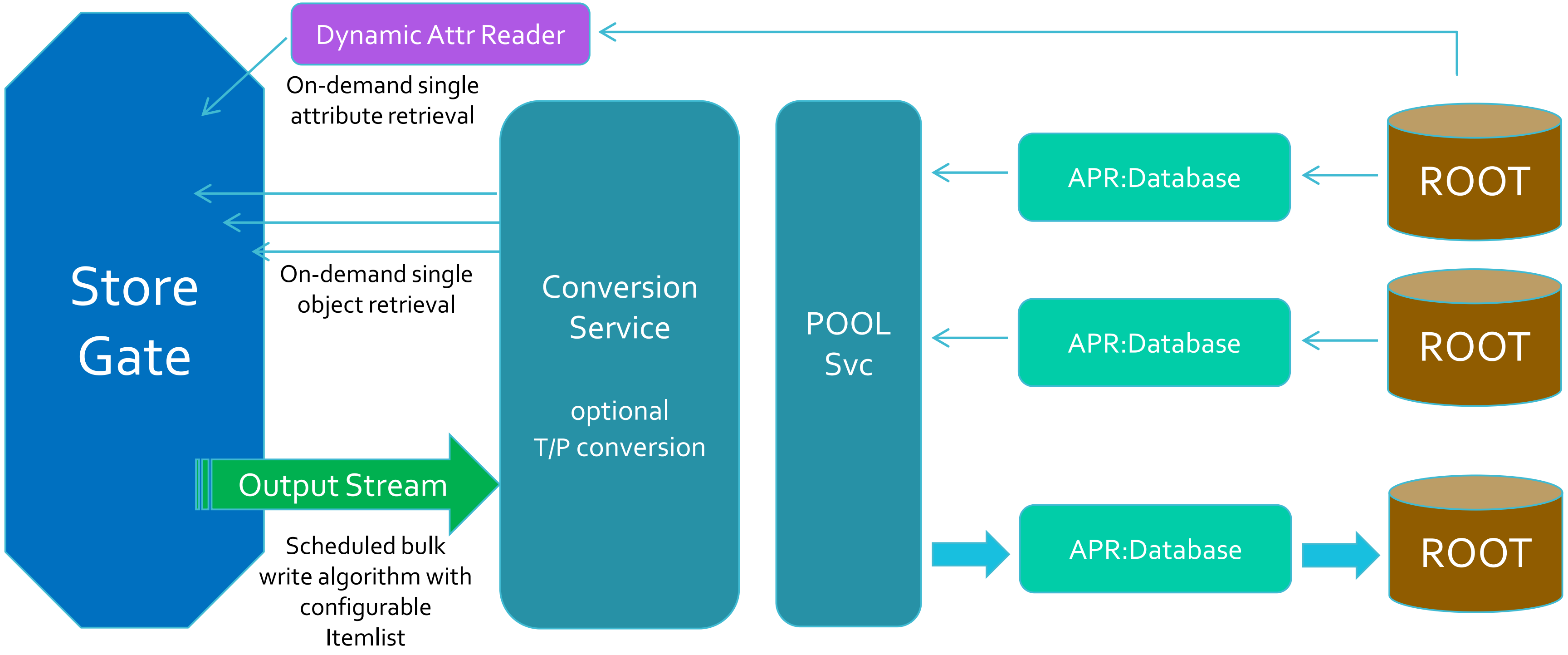
- AthenaMT
- Athena Framework I/O Components
- Input
    - Read Mutex
- Output
    - Write Mutex
- xAOD and dynamic attributes
- TTreeCache improvements
- TTree entry number and object references
- Summary

Multithreaded AthenaMT

- For Run 3 ATLAS has developed a multithreaded framework called 'AthenaMT', based on GaudiHive.
  - supports processing of multiple Events concurrently
  - each Event occupies a 'slot'
  - the number of slots if chosen at runtime and remains constant
  - each slot (Event) uses a separate transient Event Store identified by an EventContext
- The framework schedules algorithms to process Event data
  - data driven algorithm scheduling
  - concurrent / non concurrent algorithms
- Framework components and services (including the I/O services) need to be able to work in multi-threaded and multi-event environment
  - need EventContext to access Event state
  - need to handle concurrent requests
    - Reentrant or using mutex locking to serialize execution



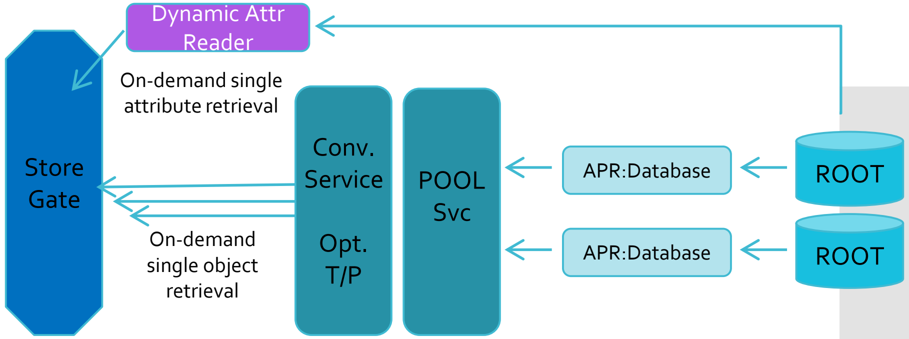
Simplified Athena I/O Components (Single Event)

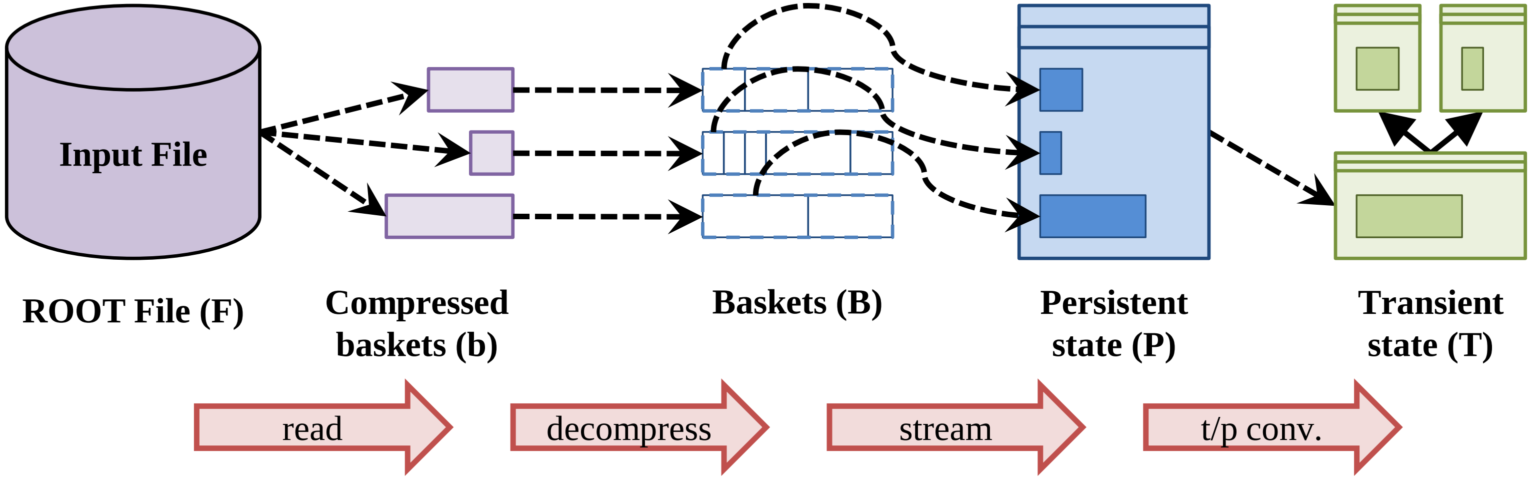


- StoreGate – transient Event Data store
- Conversion Service – Gaudi-style conversion service managing (AthenaPool) Converters
  - Converters – templates specialized by type, can be generated automatically
  - ATLAS-specific Transient/Persistent (T/P) conversion framework (for Schema Evolution)
- PoolSvc – interface layer and persistency manager for APR
- APR:Database – logical storage unit – ROOT implementation corresponds to a file
  - Event Data stored in a single TTree, every StoreGate object in a top level TBranch
    - Dynamic object attributes also use top-level branches
- Dynamic Attribute Reader – xAOD object extension for reading dynamic attributes

Input

- Read steps:
  - Single objects, on demand
  - PoolSvc + APR + ROOT
    - locate object using its Ref
      - Database, Row, Branch
    - disk read (TTreeCache)
    - branch de-compression
    - object de-serialization
    - attach Dynamic Attribute Reader to xAOD objects
  - Gaudi-style converter:
    - Persistent to Transient conversion
  - Attach transient object to its proxy in the Event Store

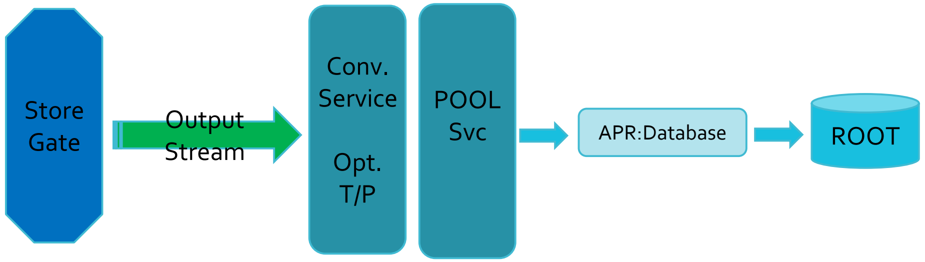




Read Mutex

- I/O Services modified (mutexed) to make Athena input multi-threaded:
  - PoolSvc - ensures that only one thread can use a particular instance of APR:Database at the time
    - Currently ATLAS uses three instances of APR:Database. One each for reading, writing, and conditions reading
      - Reading and writing happens concurrently (different files)
    - An APR:Database corresponds to a single TFile
    - One could create multiple APR:Database instances for reading a file in parallel
      - ATLAS software can create separate APR:Database instances for different data types to be read
      - This would multiply the instances of ROOT TFile, TTree and TTreeCache, but each cache would only hold a subsection of the TBranches , so memory costs are limited
        - Concurrency by event: Because data is stored in baskets, clusters of events, this could lead to multiple decompression of the same basket, so ATLAS doesn't plan to do this.
  - ConversionSvc - only one thread can use a particular instance of Converter at the time
    - Converters (Gaudi-style) - single instance per transient object type
    - T/P Converters keep a lot of internal state – easier to protect them as a whole
    - (also protecting against concurrent use of a converter for read and write)
    - as ATLAS uses many different types, lock waits should be rare

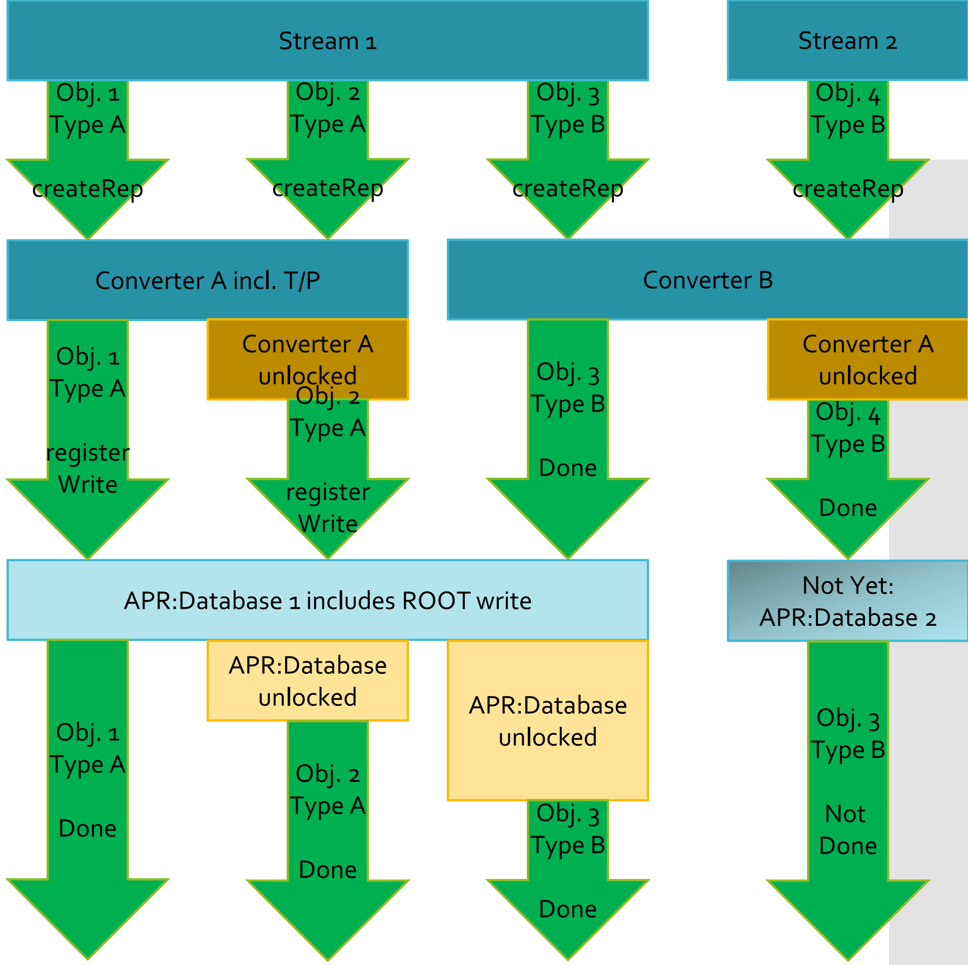
Output



- Write steps:
  - OutputStream loops over output-list objects in a given transient Event Store (slot):
    - Gaudi-style converter:
      - Persistent to Transient conversion
    - PoolSvc using ROOT:
      - Object serialization
    - xAOD objects – write all selected Dynamic Attributes
  - OutputStream is an algorithm – writes a complete Event (TTree row)
    - Can only write a single Event to a given APR:Database
  - At the end of the OutputStream execution TTree::Fill() is called
    - TTree compression
    - Disk write
      - using AutoFlush

Write Mutex

- Even though, OutputStream loops over objects, converter of different type can be dispatched concurrently
  - Possible to convert Type A for event N concurrently with converting Type B for event M.
- Currently, ATLAS uses a single, mutexed APR:Database for writing:
  - Allowing ROOT implicit multithreading.



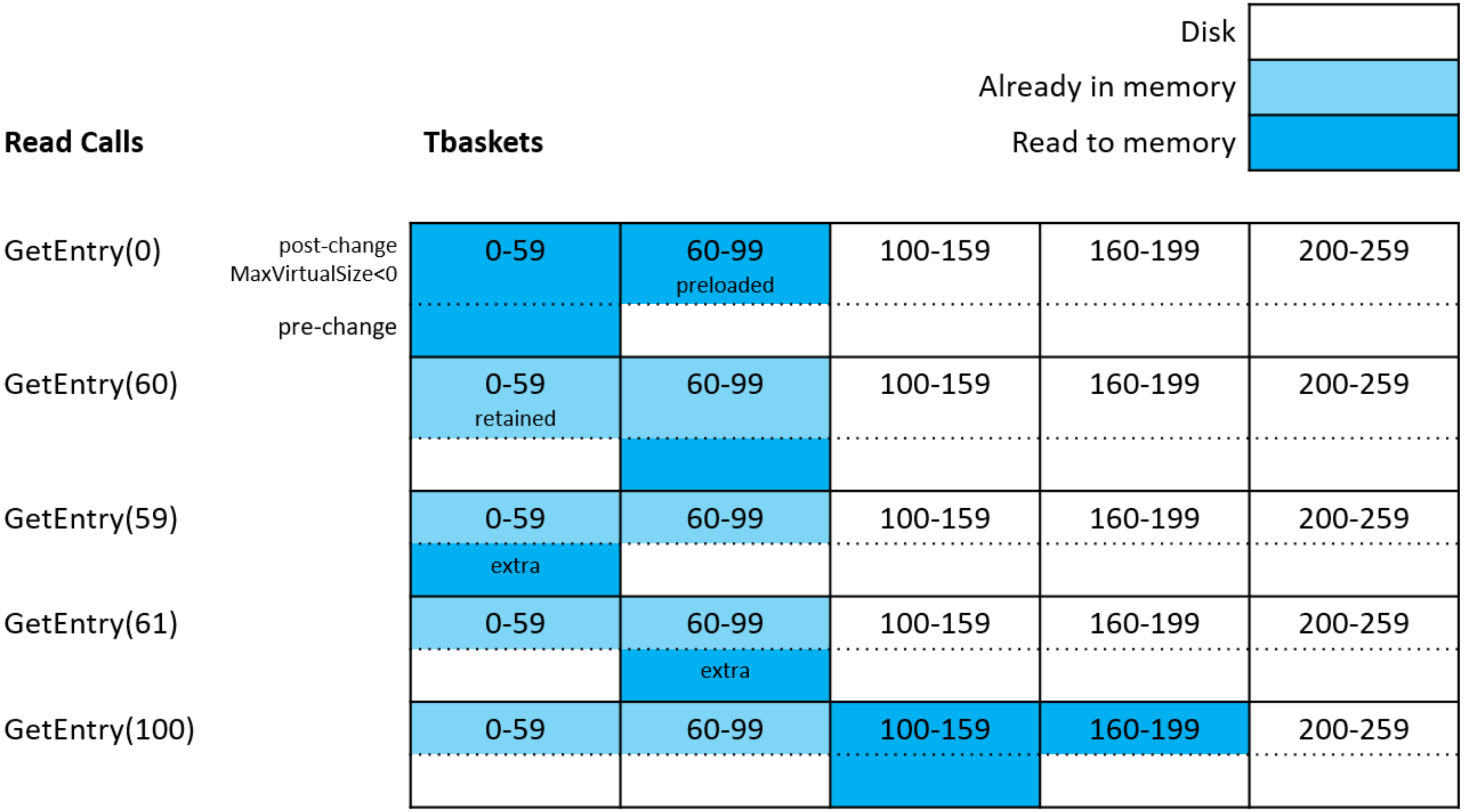
  - Possible extension would be separate APR:Database for each Stream:
    - Only for I/O intense workflows like Derivation.
      - Not clear that all production will move to AthenaMT, AthenaMP will remain important.
        - Good experience with recently deployed shared I/O for Derivation

ATLAS xAOD and dynamic attributes

- In Run 2, ATLAS has moved to a more advanced Event Data Model – xAOD
- xAOD type objects have fixed (compile time defined) and dynamic (run time defined) data members kept in 'stores'
  - The fixed (static) stores have dictionaries
    - all data read and written in a single I/O operation
    - ROOT split-level can be configured
  - Dynamic attribute stores have no dictionaries
    - Each attribute from the dynamic store is written into its own branch "by hand"
      - Attributes themselves do have dictionaries
- Dynamic attributes are not read back at the same time as their xAOD objects, but are retrieved one by one only when they are actually accessed
    - all xAOD attributes have special accessors
  - Possibility for concurrent reads from the same TFile!
    - From concurrently running algorithms accessing dynamic attributes of different xAOD objects
- ATLAS data files can have up to several thousand top-level branches, most of them for dynamic attributes
  - Efficient Caching is very important

Recent TTreeCache improvements (David Clark, ANL SULI 2017)

- Preloading and Retaining Clusters
  - Branches will load an entire clusters into memory
  - Branches will keep the current and previous cluster in memory



Disk	□
Already in memory	▨
Read to memory	■

Read Calls		Tbaskets				
GetEntry(0)	post-change MaxVirtualSize<0 pre-change	■ 0-59 ■	■ 60-99 preloaded □	□ 100-159 □	□ 160-199 □	□ 200-259 □
GetEntry(60)		▨ 0-59 retained □	▨ 60-99 ■	□ 100-159 □	□ 160-199 □	□ 200-259 □
GetEntry(59)		▨ 0-59 ■ extra	▨ 60-99 □	□ 100-159 □	□ 160-199 □	□ 200-259 □
GetEntry(61)		▨ 0-59 □	▨ 60-99 ■ extra	□ 100-159 □	□ 160-199 □	□ 200-259 □
GetEntry(100)		▨ 0-59 □	▨ 60-99 □	■ 100-159 ■	■ 160-199 □	□ 200-259 □

Future Improvements to ATLAS TTree Navigation (Nikita Dulin, ANL SULI 2018)

- ATLAS currently makes extensive use of ROOT TTree entry number as an external reference for object retrieval
- For several newer ROOT features, mainly those using TMemFile, the entry number reported by TTree::GetEntries() may not be the same as the entry number in the physical file.
  - Prevents ATLAS from adopting these features
- ATLAS recently introduced a SharedWriter concept:
  - In multi-process (MP) Athena a dedicated process is writing, collecting output from all other processes
    - no need to perform a very costly output merging later
  - Data is passes between processes in TBuffers – adding extra serialize and de-serialize steps
- Investigate, adding an unique identifier and build TTreeIndex
  - Also limit use of entry number

Summary/ Outlook

- For Run 3, ATLAS is developing a data driven, multithreaded event processing framework AthenaMT
  - ATLAS I/O components have been adapted and are safe to use in AthenaMT
    - current solutions (serialization in some areas) do not appear to create bottlenecks
    - that can change with time as AthenaMT is used for different workflows
- AthenaMT likely will not completely replace AthenaMP and recent (and future) improvement to I/O in multi-process mode are important.
- For Run 4 (and therefore in Run 3), LHC needs to move beyond just multithreaded:
  - HEP-CCE Scalable IO Workshop: https://indico.fnal.gov/event/ANLHEP1383/