Argonne NATIONAL LABORATORY

ATLAS COMPUTING

I/O in the ATLAS multithreaded framework

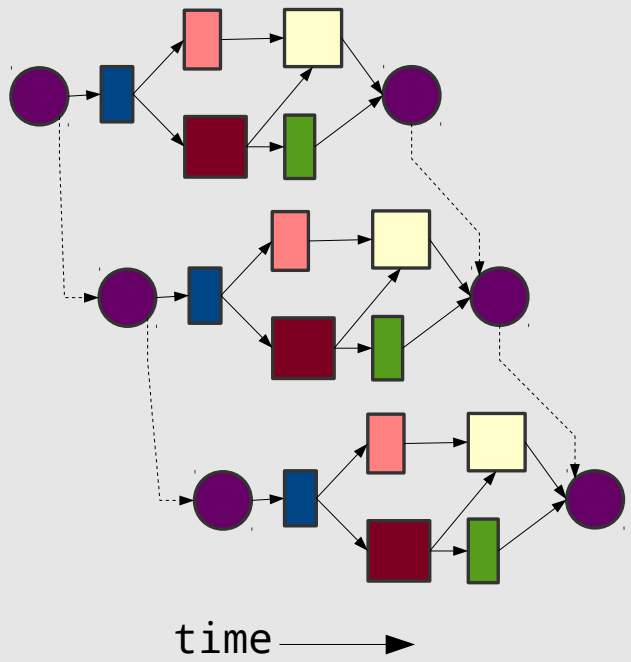
Jack Cranshaw, David Malon, Marcin Nowak, and Peter van Gemmeren

on behalf of the ATLAS Collaboration

10 July 2018

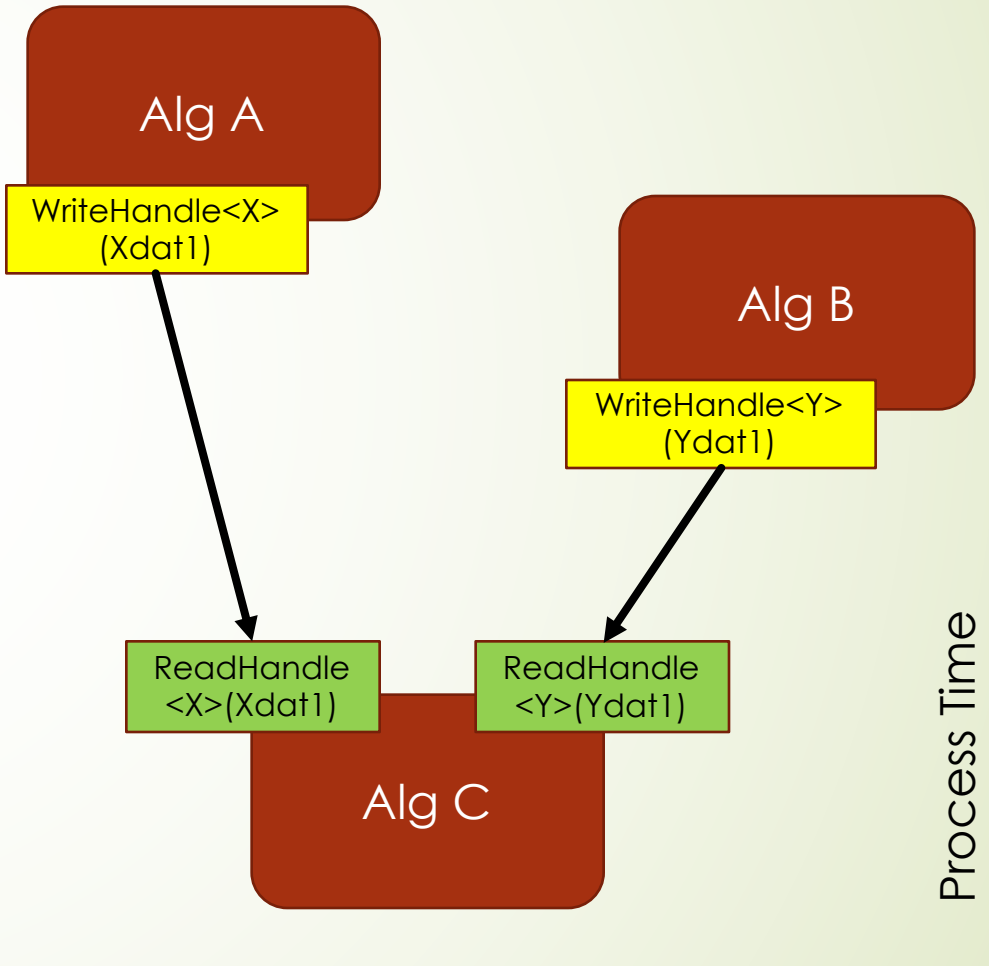
AthenaMT

- For Run 3 ATLAS has developed a ‘multithreaded' framework called AthenaMT:
  - Based on GaudiHive.
    - Multiple events in flight. Concurrent events are processed in parallel in “*slots*”.
    - A fixed number of slots is chosen at runtime, and a scheduler dispatches events to these slots.
    - Slots use separate transient event stores. Each of these slots has a specific *EventContext*.
  - The services which support the processing (including the I/O services) are outside of the event loop. These services must be one of the following:
    - stateless,
    - ‘slot-safe’ such that they can return the correct result if given an *EventContext*.
    - mutexed/locking, i.e. not support parallel access.



Data Driven Scheduling

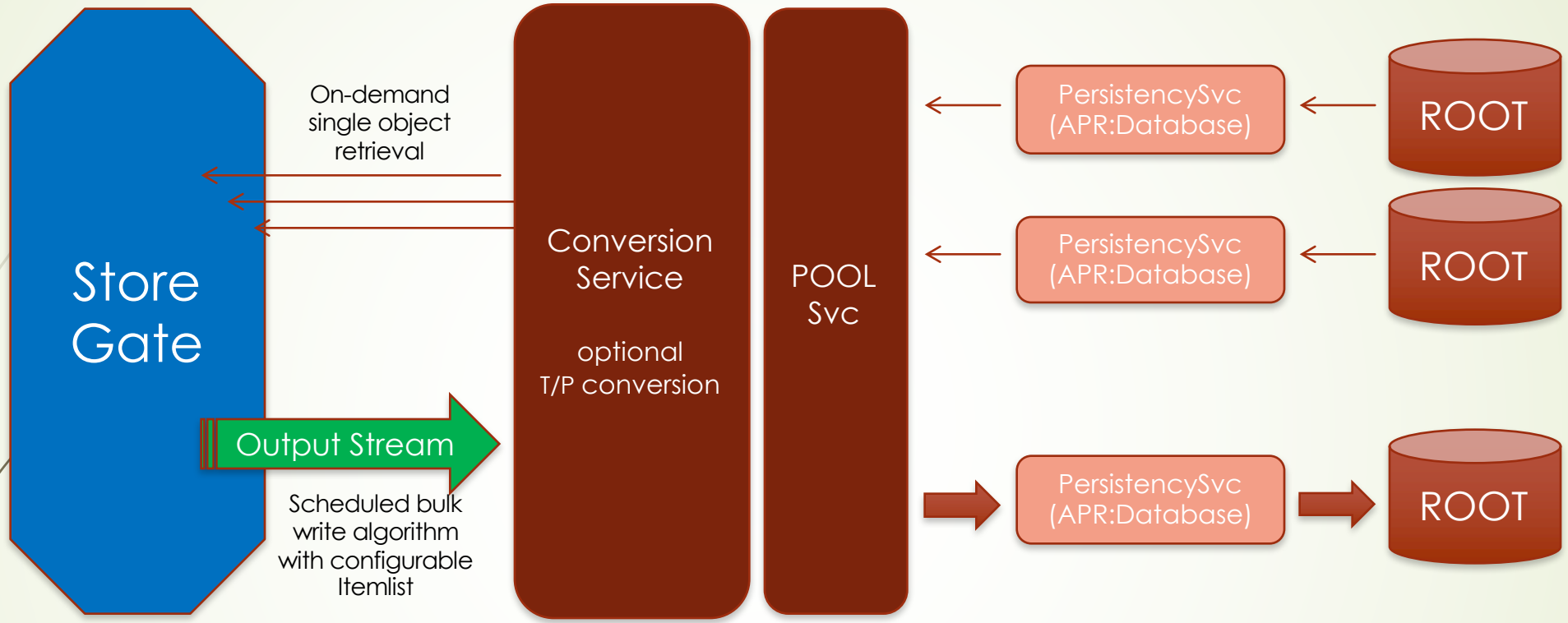
- In GaudiHive, Algorithms are scheduled based on their data dependencies.
- For this to work, the Algorithms must declare their input and output dependencies using Read/Write data handles.
  - Alg A has a WriteHandle<X>(mydat1), and Alg B has a ReadHandle<X>(mydat1)
- On input, this requires an adapter/bridge for data originating outside the process.
  - The contents of the file have to be declared to the scheduler.
  - This is currently done by attributing unmet input dependencies to the SGInputLoader Algorithm, where they are declared as WriteHandles.



ATLAS Use Cases for Multithreading

- **High Level Trigger** (ATLAS ByteStream format)
  - More efficient use of memory to increase throughput
  - Coupled with general move to make HLT and offline algorithms interoperable.
    - Gaudi flow control can replace some of the steering code.
- **Reconstruction/Simulation** (ATLAS ROOT EDM)
  - Addresses memory limitations of existing algorithmic code and challenges of handling increased pileup.
  - Allows use of native tools that support multithreading on various hardware.
- **Derivations and Analysis** (ATLAS ROOT EDM)
  - Derivations produce skimmed, slimmed, and thinned data samples for physics analysis.
  - Memory is less of an issue. The tradeoffs between multi-process and multi-threaded will have to be investigated.
  - These are our most I/O limited jobs.

Athena I/O Infrastructure



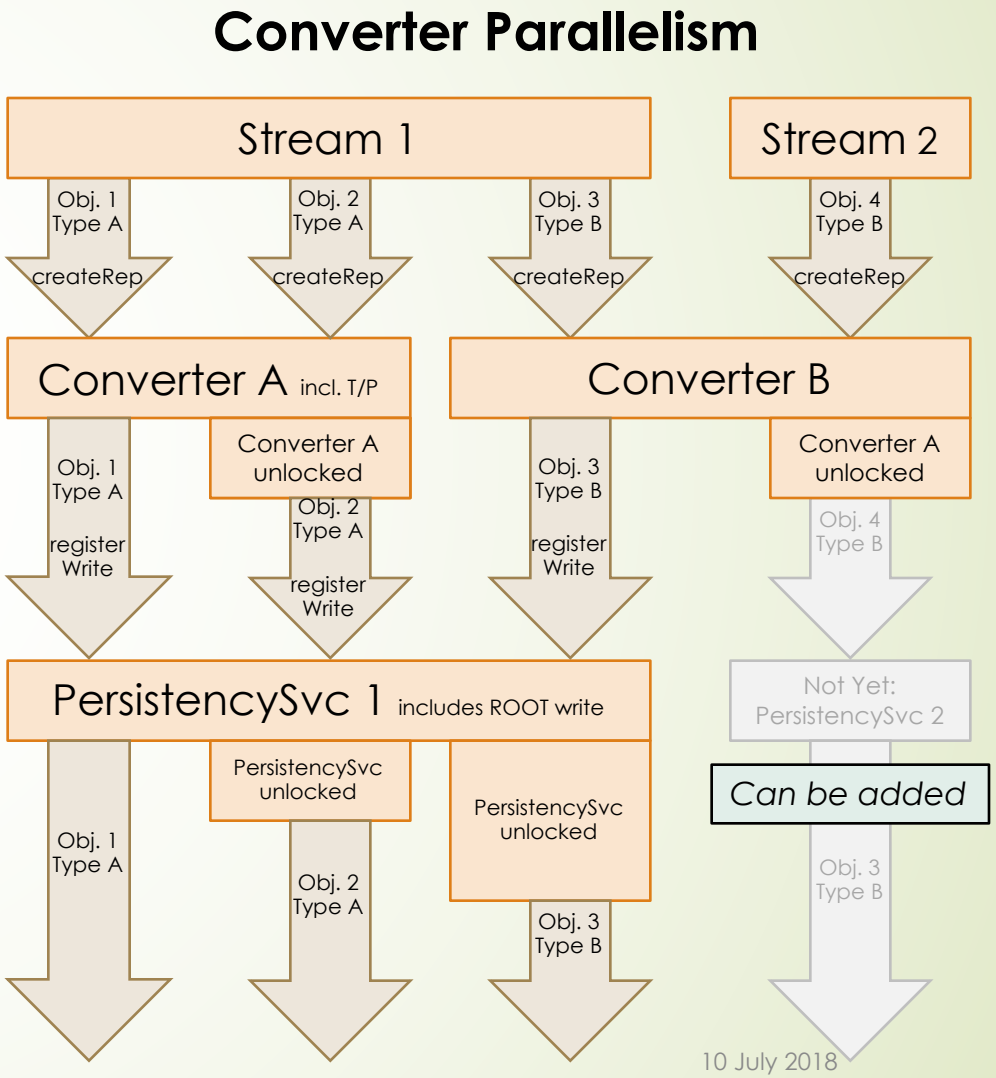
- Data is read from, and written to, StoreGate during the Athena event loop.
- Converters translate between persistent and transient class representations.
  - This is normally a simple mapping, but for some classes the storage is optimized during this stage.
- The Athena I/O services for ROOT these are shown in the diagram.
- We have looked into sharing I/O services within Gaudi, such as a RootCnvSvc, but both LHCb and ATLAS have built assumptions about their data model into their I/O services which make sharing non-trivial.

Parallel Reading and Writing (I)

- StoreGate proxies
  - When the I/O system 'reads' data it just registers the address for that data with StoreGate. The proxies data only when accessed.
- Caching
  - ByteStream:
    - For reading ByteStream, the data provider service has been upgraded to hold multiple event buffers, one for each slot. The calls to the converters now include the EventContext, so that they can access the correct buffer.
  - ROOT:
    - TTreeCache performs intelligent read-aheads and caches the data.
    - Event storage and caching have boundaries: files, baskets, TTreeCache, etc. With multiple events in flight, the I/O can encounter points where it flips back and forth across one of those boundaries. When it does, you can get a thrash condition.
      - Fixed by a summer student at Argonne last summer and incorporated into the ROOT code base.

Parallel Reading and Writing (II)

- Multiple PersistencySvc
  - Currently we use three PersistencySvc. One each for reading, writing, and conditions reading.
- Concurrent Writing
  - We have multiple serialization points during writing: ROOT, our PoolSvc, and StoreGate.
  - Converters for different types can be run in parallel(as shown in the figure), so there is limited scope for parallelism within the conversion service.
- Parallel File I/O
  - One could read create multiple TFile/TTreeCaches for parallel file reading
    - *By type*: We can do this by creating multiple PersistencySvc instances, one for each type (the grayed out portion in the figure).
    - *By event*: Because each branch is stored in baskets, clusters of events, this can lead to multiple decompression of the same basket, so we don't do this.



Multiple Output Streams

- ATLAS makes use of parallel output streams at multiple stages in its processing.
  - Reconstruction: filtered streams for calibration or trigger studies. (~10)
  - Derivation: filtered streams for physics analyses. (~10)
- Derivation uses resources very differently from reconstruction.
  - For example, we will have to compare the performance of multi-process (MP) and multi-threaded (MT) approaches. Using a separate process for each stream may work better than a single multi-threaded process.
- Changes to event filtering.
  - Filtering of output streams is done using filter algorithms within the event loop and applying a logic inside the *DecisionSvc* which provides a boolean result to the AthenaOutputStream (an Algorithm in the event loop).
  - The *DecisionSvc* has been updated to be 'slot'-safe, and works in AthenaMT. Much of the functionality has been subsumed into the Gaudi framework itself.
  - This entire process has been rendered somewhat redundant by Gaudi's new, advanced control flow features. Some users of simulation have already abandoned the *DecisionSvc*.
    - This needs to be addressed at a more architectural level as it affects bookkeeping.
- Currently the scheduling of output streams is the same in AthenaMT as it is in serial Athena. They are placed in a sequence which is executed after the event processing algorithms.
  - One could think of using data driven scheduling to run the output stream as soon as all of its objects are ready, but its unclear whether the increase in complexity warrants this.

Metadata and Conditions

- Event data are processed within the event loop and have an event scope defined by the *EventContext* and the event store for that slot.
- Non-event data are also accessed during event processing. These include both **conditions** (calibrations, alignments, etc.) and **metadata** (simulation parameters, dataset information, bookkeeping, etc.).
  - *Condition* contexts are defined by an interval of validity (*IOV*) scope.
    - This is associated with the conditions data by creating container objects in StoreGate which function as maps of *IOV* to data. Conditions handles take an *EventContext* as argument and return the correct data object from the conditions store by comparing the *EventContext* to the *IOV's* in the map.
  - *Conditions* data are primarily accessed from a database (Oracle, squid, sqlite,…), but the data is stored in ROOT objects (thus the third PersistencySvc)
  - *Metadata* context is defined by the source of the metadata.
    - Metadata uses a similar system of handles and containers to conditions, but with a different key. This system is still under development and testing.
  - For historical reasons, a large fraction of the metadata used by a job is stored and accessed from the data file rather than from a database.
    - ATLAS is reviewing this system as part of building an I/O roadmap.

Data Chunking and Bookkeeping

- There are two data chunks which impact bookkeeping during the job.
  - *Luminosity*: luminosity is calculated for chunks of events called luminosity blocks, typically around 1 minute of data taking. Physics analyses need to know they have seen all the events from a given luminosity block to integrate the total luminosity correctly.
  - *Input Events*: Both for luminosity and for other normalizations, it is important to know how many events have been read and whether a file has been completely read.
- Because bookkeeping is done during data processing, it has been stored in the data files. The sum of events and sum of events per lumiblock are saved and marked as either complete (input fully read) or incomplete (input partially read).
  - This system must evolve as our data processing workflow becomes more flexible. A single process doesn't have enough information to do it properly.
- Two ideas are being developed for Run 3 to make this more robust.
  - External accounting: Propagating the event counts to a database which then has information about multiple jobs.
  - Deferred accounting: Keeping a separate list of events which can be summed after the file is closed or merged.

Conclusion and Outlook

- Athena I/O services have been upgraded to support multithreaded athena, AthenaMT.
  - Caching is enabled for both RAW data (multiple buffers) and ROOT data (TTreeCache).
  - Parallelism has been improved and expanded within the Athena I/O services.
  - We maximize our use of ROOT multithreading where possible, and we will leverage (and contribute) to improvements in ROOT I/O.
- A system for handling conditions and metadata is in the later stages of development, but it is also being re-evaluated as part of an ATLAS review of I/O strategies for Run 3/4.
- This means that ATLAS I/O now supports event processing on multi-core architectures for both multithreaded processing (AthenaMT) and multiprocess processing (AthenaMP).