ATLAS I/O Overview

Peter van Gemmeren (ANL)

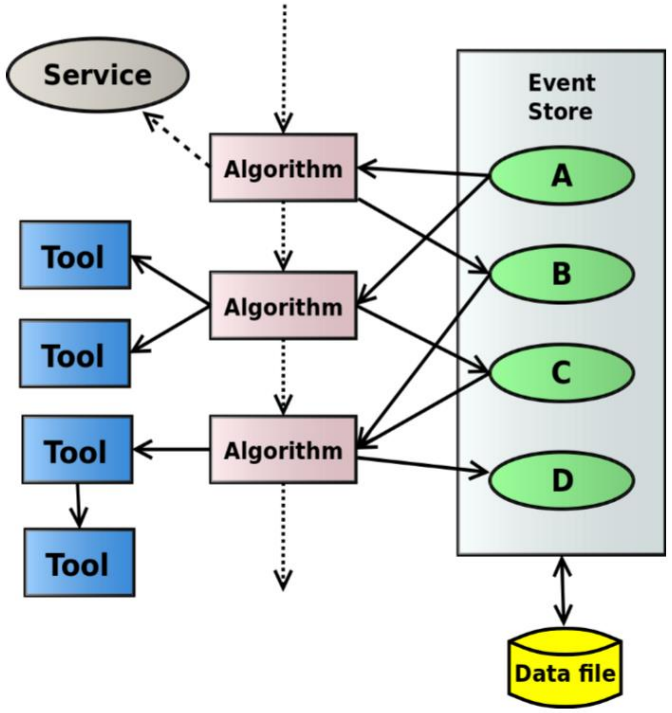
gemmeren@anl.gov for many in ATLAS

Overview

- High level overview of ATLAS Input/Output framework and data persistence.
  - Athena: The ATLAS event processing framework
  - The ATLAS event data model
  - Persistence:
    - Writing Event Data: OutputStream and OutputStreamTool
    - Reading Event Data: EventSelector and AddressProvider
    - ConversionSvc and Converter
- Timeline
  - Run 2: AthenaMP, xAOD
  - Run 3: AthenaMT
  - Run 4: Serialization, Streaming, MPI, ESP

Athena: The ATLAS event processing framework

- Simulation, reconstruction, and analysis/derivation 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:
  - 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.



Workflows

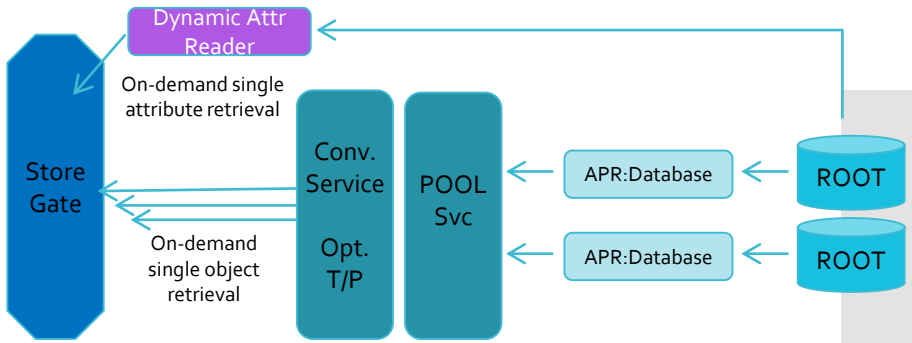
- Athena is used for different workflows in Reconstruction, Simulation and Analysis (mainly Derivation).

Step	Total Read (incl. ROOT and P->T)		Total Write (w/o compression)		ROOT compression		Total CPU evt-loop time
EVNTtoHITS	0.006	0.01%	0.017	0.02%	0.027	0.03%	91.986
HITtoRDO	1.978	5.30%	0.046	0.12%	0.288	0.77%	37.311
RDOtoRDO-Trigger	0.125	1.23%	0.153	1.51%	0.328	3.23%	10.149
RDOtoESD	0.166	1.88%	0.252	2.85%	0.444	5.02%	8.838
ESDtoAOD	0.072	23.15%	0.147	47.26%	0.049	15.79%	0.311
AODtoDAOD	0.052	5.35%	0.040	4.06%	0.071	7.24%	0.979
RAWtoALL	N/A	N/A	0.112	0.72%	0.043	0.28%	15.562

The ATLAS event data model

- The transient ATLAS event model is implemented in C++, and uses **the full power** of C++, including **pointers**, **inheritance**, **polymorphism**, **templates**, STL and Boost classes, and a variety of **external packages**.
- At any processing stage, event data consist of a **large and heterogeneous** assortment of objects, with associations among objects.
- The final production outputs are **xAOD** and **DxAOD**, which were designed for Run II and after to **simplify** the data model, and make it more **directly usable** with **ROOT**.
  - More about this later…

Persistence

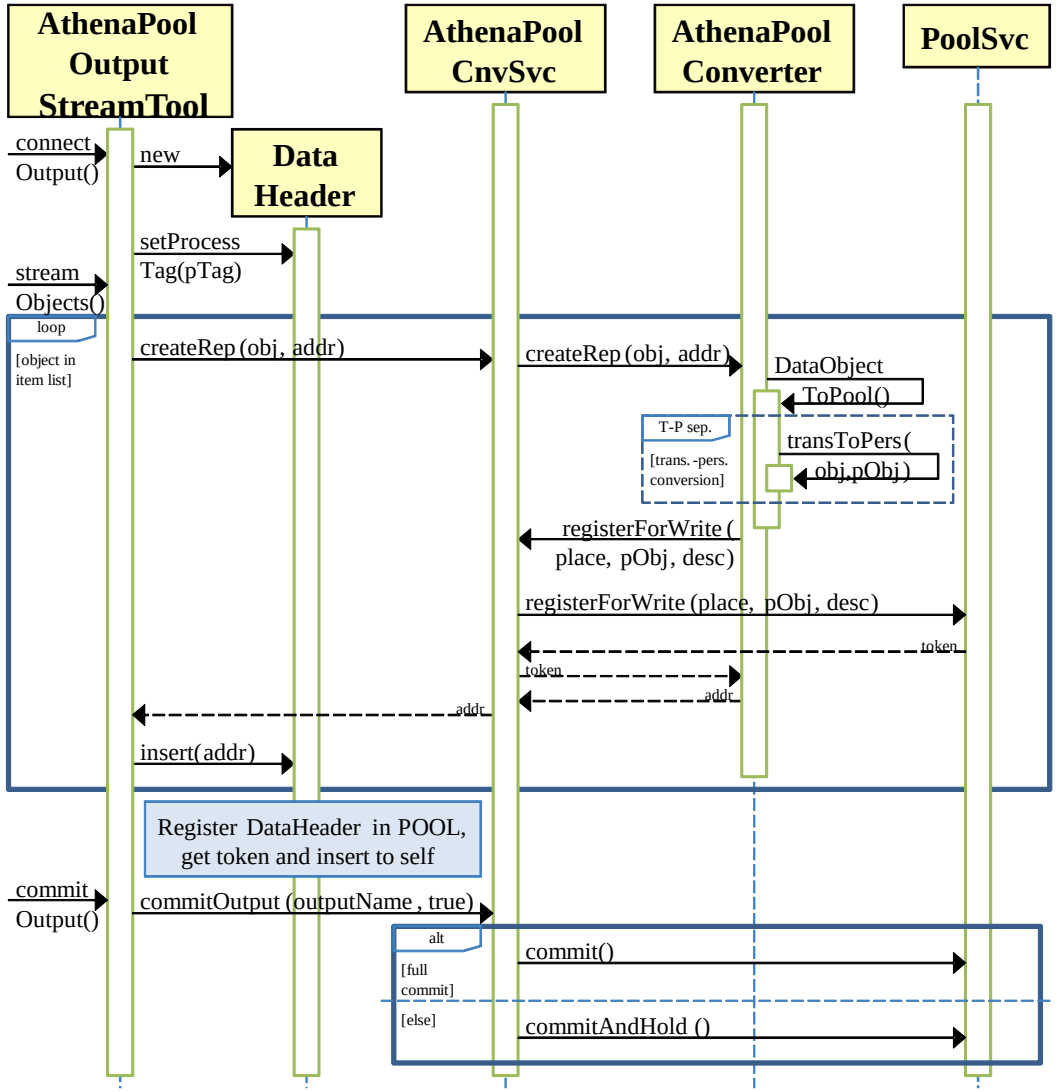


- ATLAS currently has almost **400 petabytes** of event data
  - Including replicated datasets
- ATLAS stores most of its event data using **ROOT** as its persistence technology
  - Raw readout data from the detector is in another format.

Writing Event Data

Sequence Diagram for writing Data Objects via AthenaPOOL:

The AthenaPool-OutputStreamTool is used for writing data objects into POOL/APR files and hides any persistency technology dependence from the Athena software framework.



OutputStream and Output-StreamTool

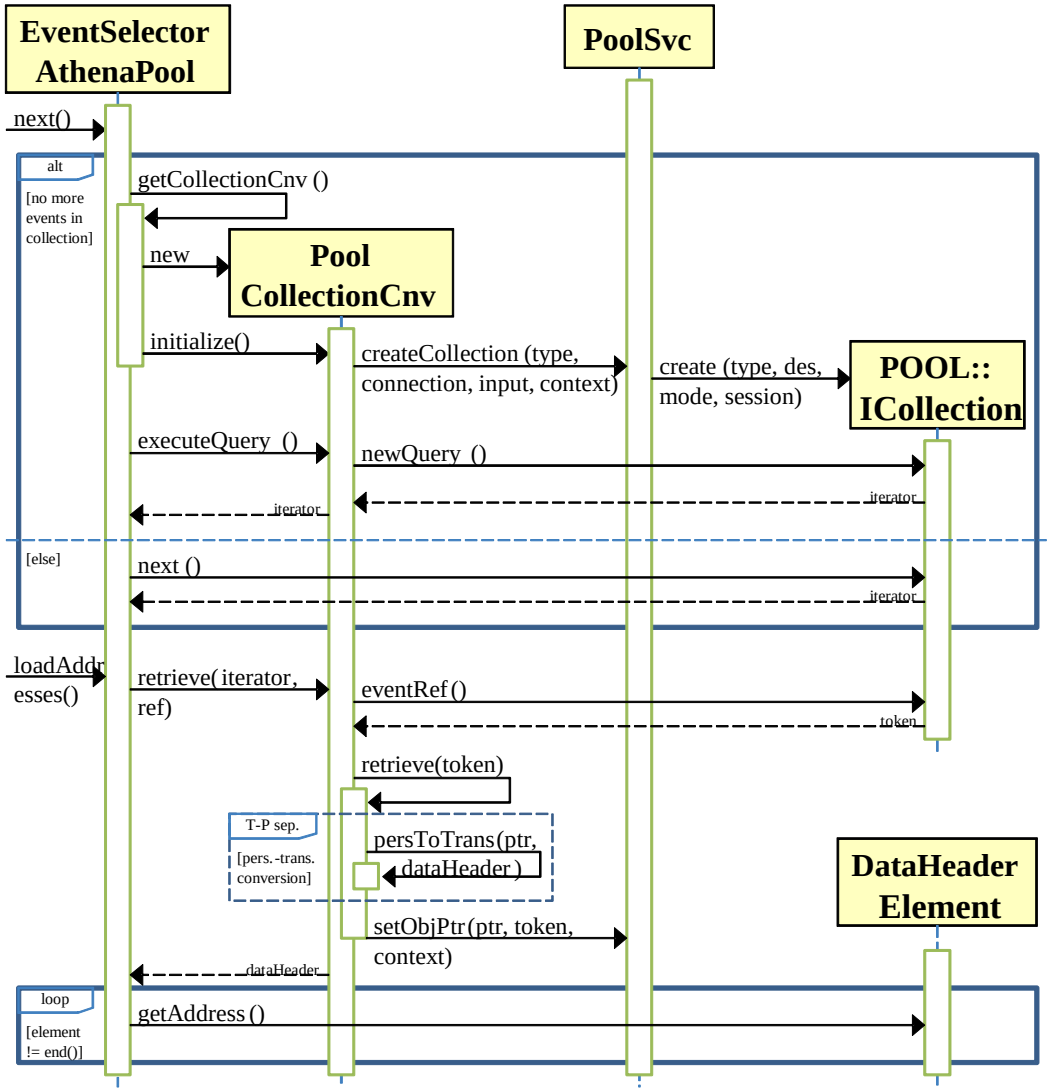
- **OutputStreams** connect a job to a data sink, usually a file (or sequence of files).
- Configured with **ItemList** for event and metadata to be written.
- Similar to Athena **algorithms**:
  - Executed once for each event
    - Can be vetoed to write filtered events
  - Can have multiple instances per job, writing to different data sinks/files
- **OutputStreamTools** are used to interface the OutputStream to a **ConversionSvc** and its **Converter** which depend on the persistent technology.

Reading Event Data

Sequence Diagram for reading Data Objects via AthenaPOOL:

An EventSelector is used to access selected events by iterating over the input DataHeaders.

An Address-Provider preloads proxies for the data objects in the current input event into StoreGate.

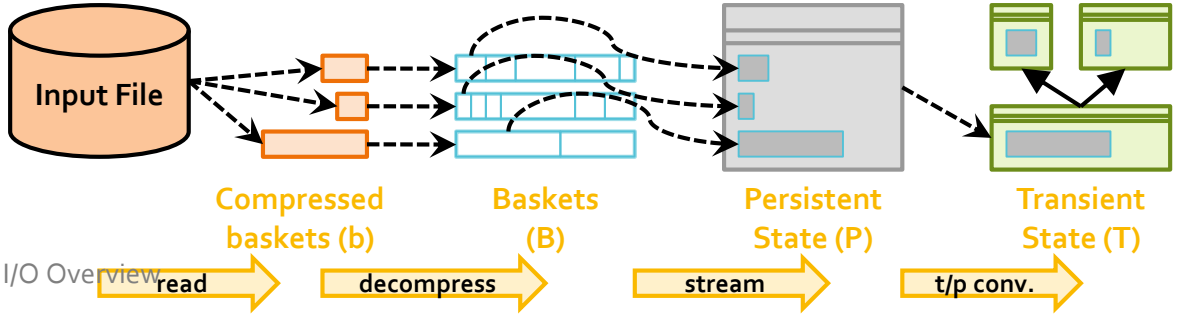


EventSelector and Address-Provider

- The **EventSelector** connect a job to a data sink, usually a file (or sequence of files).
- For event processing it implements the next() function that provides the persistent reference to the **DataHeader**.
  - The DataHeader stores persistent references and StoreGate state for all data objects in the event.
- It also has other functionality, such as handling file boundaries for e.g. metadata processing.
- An **AddressProvider** is called automatically, if an object retrieved from StoreGate has not been read.
- AddressProvider interact with **ConversionSvc** and **Converter**

ConversionSvc and Converter

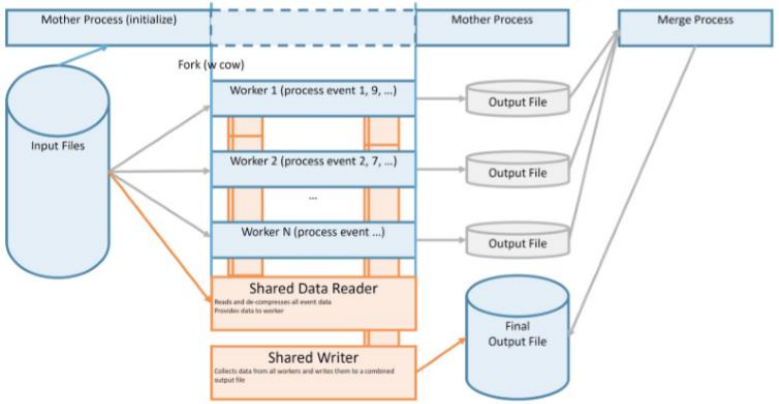
- The role of conversion services and their converters is to provide a means to write C++ data objects to storage and read them back.
- Each storage technology is implemented via a ConversionSvc and Converter.
  - ATLAS uses **ROOT** via **POOL/APR** that is implemented via Athena/Pool Conversion
    - APR implements ROOT **TKey** and **TTree** technologies.
  - Converter dispatching done by **type**.
- Converters can do (optional) **Transient/Persistent** mappings and handle schema evolution.
- When writing, Converter return an **externalizable reference**.



Run 2

Multi Process: AthenaMP

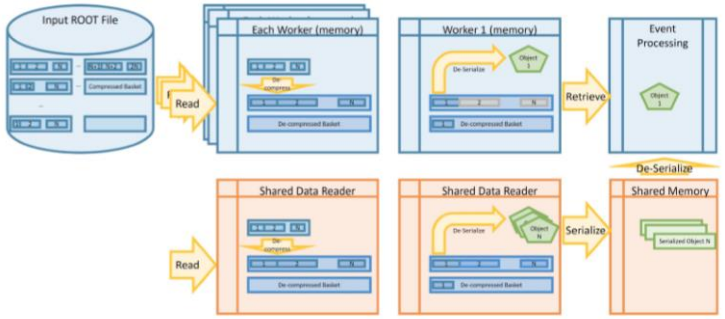
- Since Run II, ATLAS has deployed **AthenaMP**, the multi-process version of Athena.
  - Starts up and initializes as single (mother) process.
    - Optionally processes events
  - Forks of (worker) processes that do the event processing in parallel.
    - Utilizes **Copy On Write**, thereby saving large amounts of memory.
    - Each worker has its **own address space**, no sharing of event data.
  - In default mode, workers are independent of each others for I/O: Read their own data directly from file and write their own output to a (temporary) file.
    - Input may be non-optimal as worker have to de-compress the same buffers to process different subsections of events -> cluster dispatching
    - output from different workers needs to be merged, which can create a bottleneck -> deployment of **SharedWriter**



AthenaMP: Shared I/O components

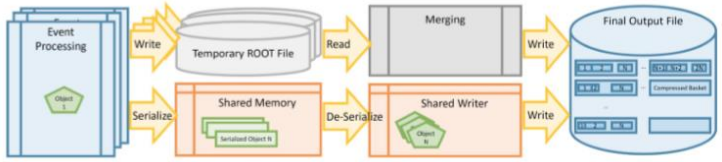
SharedReader

- The Shared Data Reader reads, de-compresses and de-serializes the data for all workers and therefore provides a single location to store the decompressed data and serve as caching layer.



SharedWriter

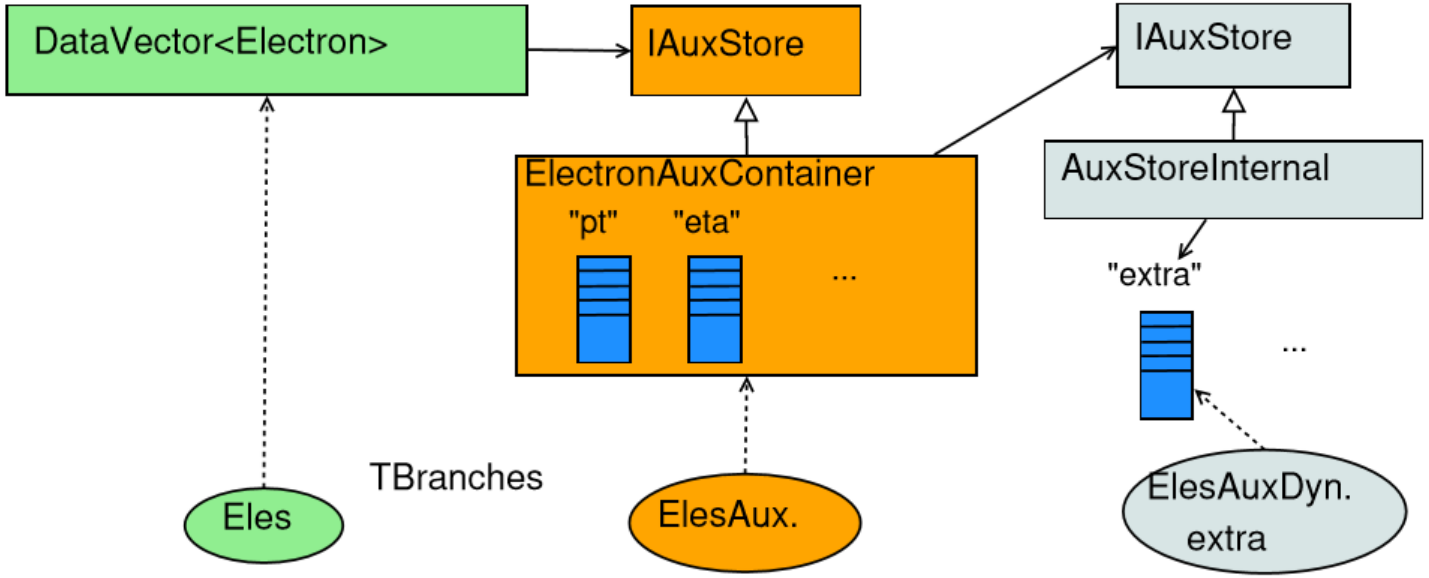
- The Shared Writer collects output data objects from all AthenaMP workers via shared memory and writes them to a single output file.
  - This helps to avoid a separate merge step in AthenaMP processing.



Also Run 2

xAOD Data Model

- Each **xAOD** container has an associated data store object (called **Auxiliary Store**).
  - Both are recorded in StoreGate.
- The key for the aux store should be the same as the data object with 'Aux.' appended.
- The xAOD aux store object contains the '**static**' aux variables.
- It also holds a SG::AuxStoreInternal object which manages any additional '**dynamic**' variables.



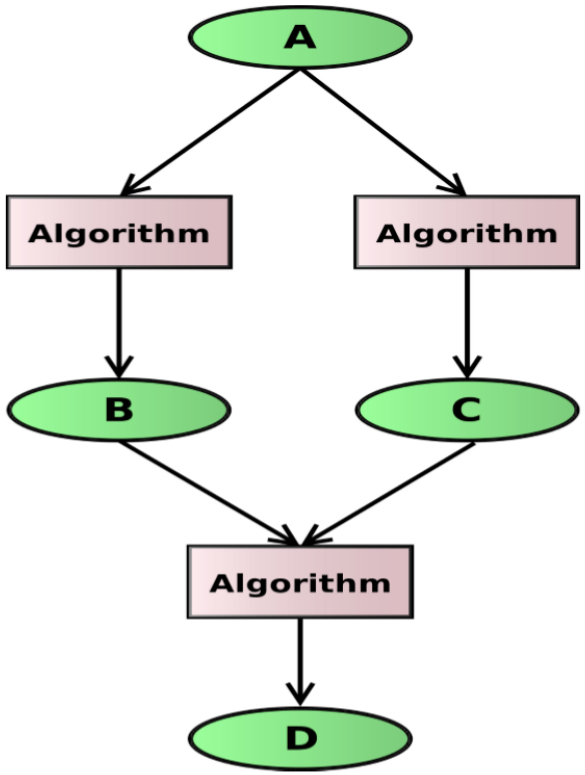
xAOD: Auxiliary data

- Most xAOD object data are not stored in the xAOD objects themselves, but in a separate auxiliary store.
- Object data stored as vectors of values.
  - ("Structure of arrays" versus "array of structures.")
- Allows for better interaction with root, partial reading of objects, and user extension of objects.
- Opens up opportunities for more vectorization and better use of compute accelerators.

Run 3

Multi Thread: AthenaMT

- Task scheduling based on the Intel Thread Building Blocks library with a custom graph scheduler.
- Data Flow scheduling:
  - Algorithms declare their inputs and outputs. Scheduler finds an algorithm with all inputs available and runs it as a task.
    - Algorithm data dependencies declared via special properties (HandleKey).
    - Dependencies of tools will be propagated up to their owning algorithms.
  - Flexible parallelism within an event.
- Can still declare sequences of algorithms that must execute in fixed order ("control flow").
- Number of **simultaneous events** in flight is configurable

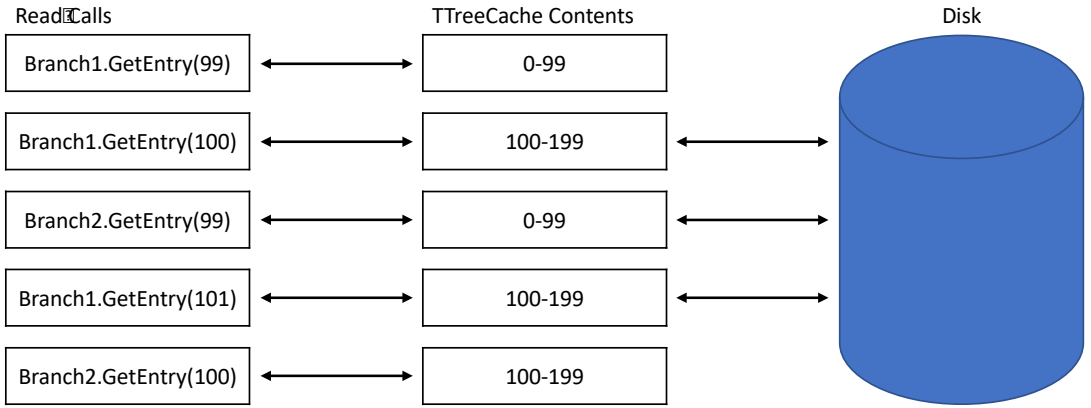


Things about ROOT

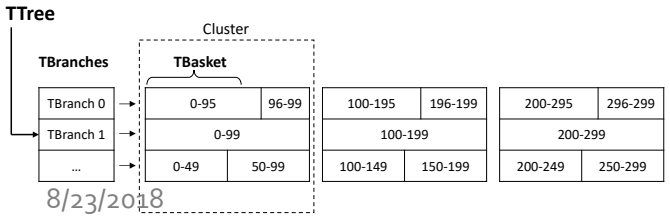
- ROOT is solidly thread safe:
  - After calling ROOT::EnableThreadSafety() switches ROOT into MT-safe mode (done in PoolSvc).
  - As long as one doesn't use the same TFile/TTree pointer to read an object
  - Can't write to the same file
- In addition, ROOT uses implicit Multi-Threading
  - E.g., when reading/writing entries of a TTree
    - After calling ROOT::EnableImplicitMT(<NThreads>) (new! in PoolSvc).
    - Very preliminary test show Calorimeter Reconstruction (very fast) with 8 threads gain 70 - 100% in CPU utilization
- However, that doesn't mean that multi-threaded workflows can't provide new challenges to ROOT
  - ATLAS Example on the next slides

Enhanced data caching for multi-threaded workflows

- On demand data reading (even dynamic aux store) and multi-threaded workflow can lead to non-sequential branch access, which can cause thrashing of TTreeCache.

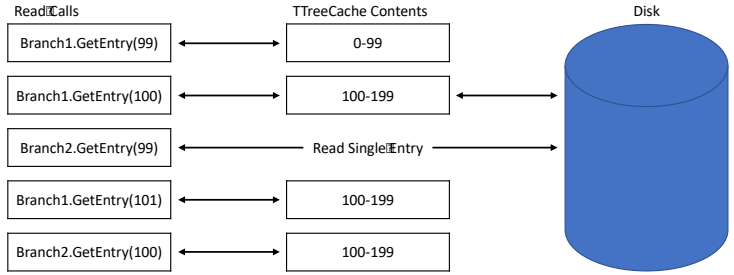


Work done by ANL SULI David Clark



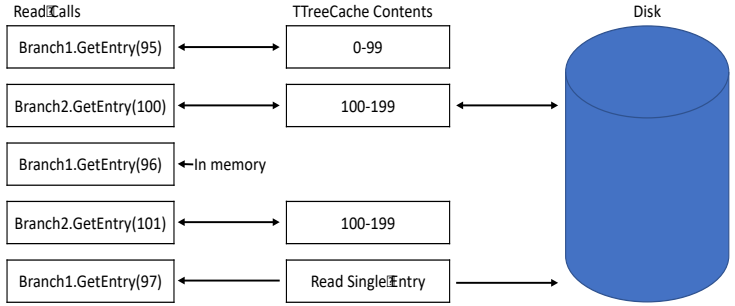
Forward Caching

- Setting the cache for leading branch will avoid invalidating it for late branch reads



Preloading and Retaining clusters

- Preloading trailing baskets can avoid reading single entries

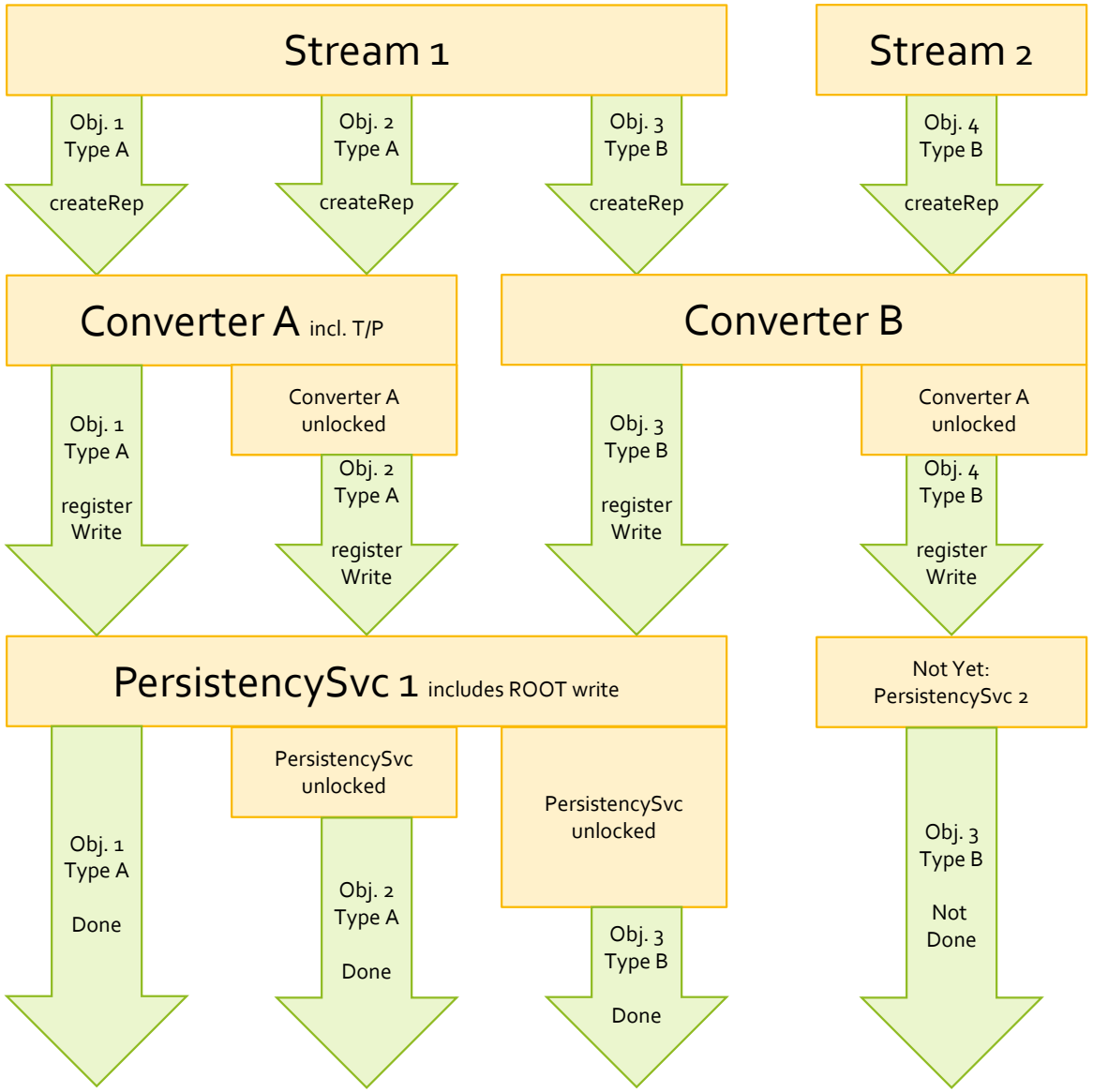


Thread Safety of Athena I/O

- The I/O layer has been adapted for multi-threaded environment
  - Conversion Service – OK
    - Serializing access to Converters for the same type, but converters for different types can operate concurrently
      - It means we can read/convert different objects types (~branches) in parallel
  - PoolSvc – OK
    - Serializing access to PersistencySvc
      - Can use multiple PersistencySvc for reading, but currently only one for writing
  - POOL/APR – OK
    - Multiple PersistencySvc can operate concurrently
      - Each has its own TFile instance with dedicated cache
  - FileCatalog – OK
  - Dynamic AuxStore I/O (reading) – OK
    - On-demand reading from the same file as other threads

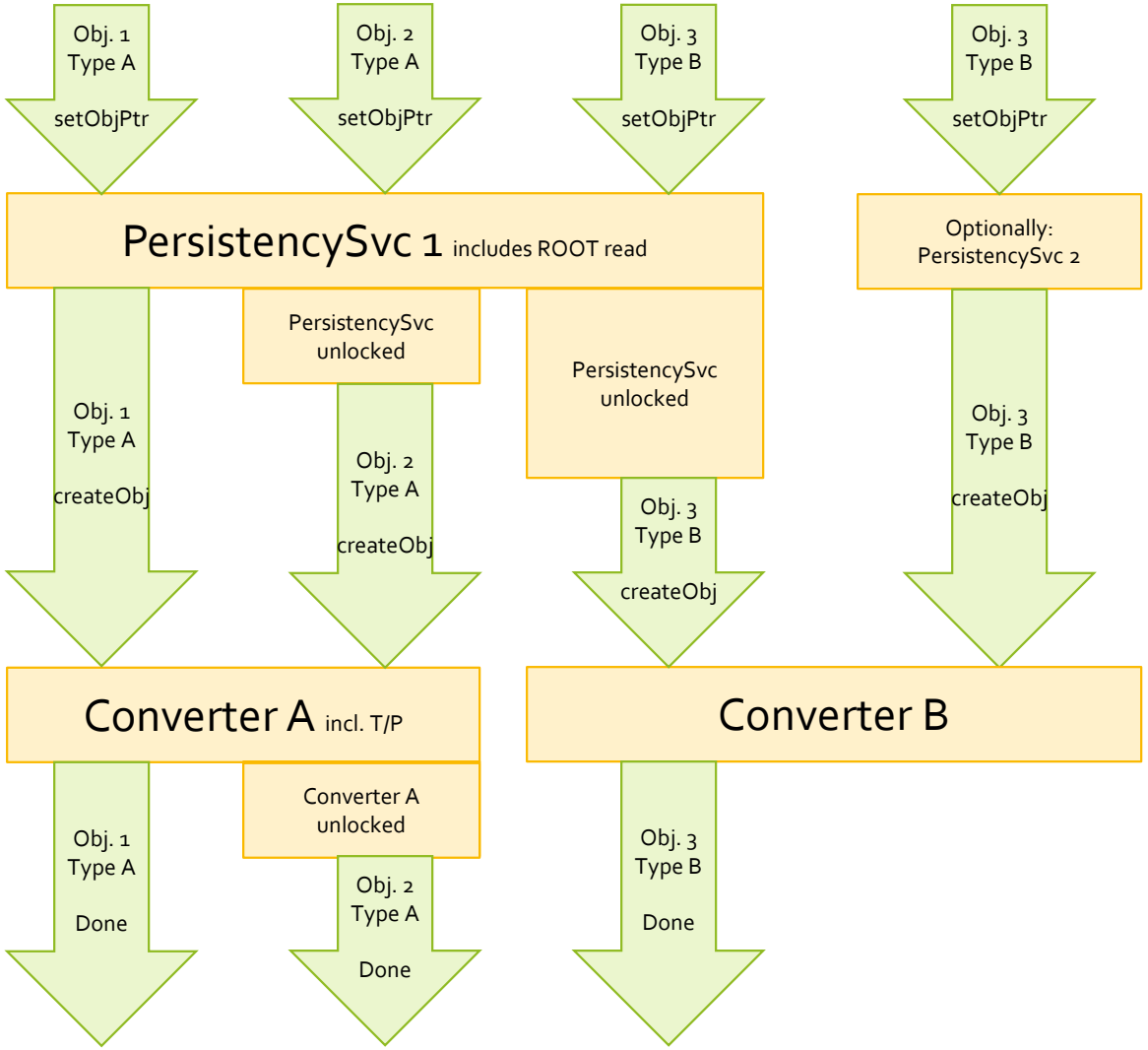
Writing:

Objects are to the same ROOT file and typically the same TTree, but there may be several streams…



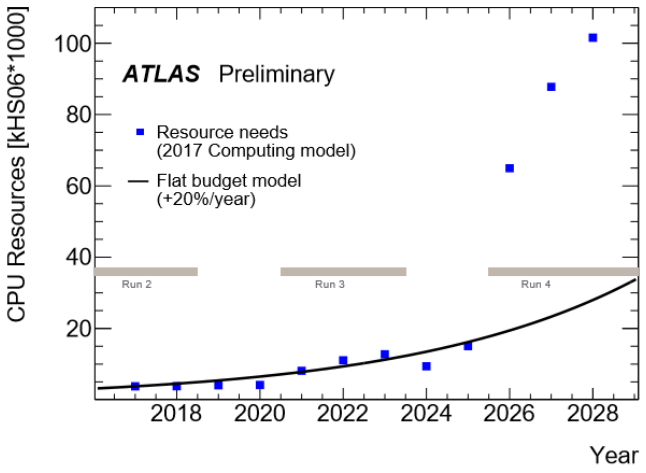
Reading:

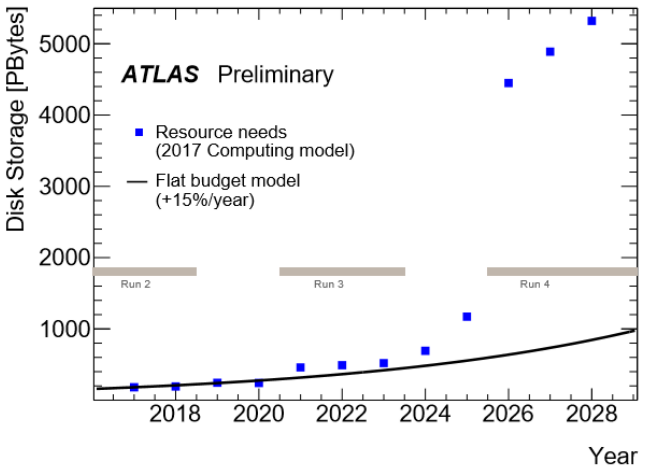
All Objects are in the same ROOT file and typically the same TTree



Challenge for LHC computing at Run 4

- Assuming ATLAS' current compute model, CPU and storage needs for Run 4 will increase to a factor of 5-10 beyond what is affordable.
- The answer on how to mitigate the shortfall is better, wider and more efficient, use of HPC:
  - ATLAS software, Athena, was written for serial workflow
    - Migrated to AthenaMP in Run 2, but still dealing with improvements.
      - Required only Core and I/O software changes.
    - In process, but behind schedule, to move to AthenaMT for Run 3.
      - Limited changes non-Core software, but clients need to adjust to new interfaces.
    - Changes to allow efficient use of heterogeneous HPC resources (including GPU/accelerators) for Run 4 will be more intrusive.





Figures taken from: arXiv:1712.06982v3

Run3 -> Run 4

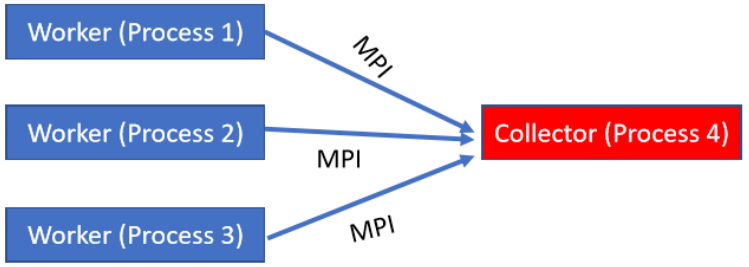
- ATLAS is currently reviewing its I/O framework and persistence infrastructure.
- Clearly efficient utilization of HPC resources will be a major ingredient for dealing with the increase of compute resource requirements in HL-LHC.
  - Getting data onto and off of a large number of HPC nodes efficiently will be essential to effective exploitation of HPC architectures.
  - SharedWriter already in production (e.g., in AthenaMP) and the I/O components already supporting multithreaded processing (AthenaMT) provide a solid foundation for such work
    - A look at integrating current ATLAS shared writer code with MPI underway at LBNL
    - Related work (TMPIFile with synchronization across MPI ranks) by a summer student at Argonne

Streaming data

- ATLAS already employs a serialization infrastructure
  - for example, to write high-level trigger (HLT) results
  - and for communication within a shared I/O implementation
- Developing a unified approach to serialization that supports, not only event streaming, but data object streaming to coprocessors, to GPUS, and to other nodes.
- ATLAS takes advantage of ROOT-based streaming.
  - An integrated, lightweight approach for streaming data directly would allow us to exploit co-processing more efficiently.
    - E.g.: Reading an Auxiliary Store variable (like vector<float> directly onto GPU (as float []).

TMPI File

- Work done by Amit Bashyal (CCE summer student, Advisor: Taylor Childers).
- TFile like Object that is derived from TMemFile and uses MPI Libraries for Parallel IO.
- Process data in parallel and write them into disk in TFile as output.
- Works with TTree cluster
  - Worker collect events, compresses and sends to collector.



Workers:
- Process Events (Populate TTrees or TH1D's)
- Send Processed Events to Collector Using MPI functionalities

Collectors:
- Receive Processed Events from Workers
- Write into disk

ALCF data and learning project for Aurora Early Science Program

- **Simulating and Learning in the ATLAS Detector at the Exascale**
  James Proudfoot, Argonne National Laboratory
  - ***Co-PI's from ANL and LBNL***
- The ATLAS experiment at the Large Hadron Collider measures particles produced in proton-proton collision as if it were an extraordinarily rapid camera. These measurements led to the discovery of the Higgs boson, but hundreds of petabytes of data still remain unexamined, and the experiment's computational needs will grow by an order of magnitude or more over the next decade. This project deploys necessary workflows and updates algorithms for exascale machines, preparing Aurora for effective use in the search for new physics.

AURORA | EARLY SCIENCE PROGRAM FOR DATA & LEARNING

Conclusion

- ATLAS has successfully used ROOT to store almost 400 Petabyte of event data.
- ATLAS will continue to rely on ROOT to support its I/O framework and data storage needs.
- Run 3 and 4 will present challenges to ATLAS that can only be solved by efficient use of HPC …
- and we need to prepare our software for this.
  - ATLAS and ROOT