From LHC to FCC-ee: Guiding I/O and Data Storage Software of Future Multi-decade Experiments with Lessons Learnt at ATLAS

Alaettin Serhan Mete

Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Second Annual U.S. Future Circular Collider (FCC) Workshop 2024
25-27 March 2024

Argonne NATIONAL LABORATORY

A brief introduction to ATLAS and Athena

- **ATLAS is a general-purpose detector at the Large Hadron Collider (LHC)**
- **Athena is the open-source software framework of ATLAS**
  - Based on the Gaudi framework, jointly managed by the ATLAS and the LHCb experiments
  - It consists of about 4 (1.5) million lines of C++ (python) code
    - CMake for *building*, python for *job configuration*, and C++ for *the framework and the algorithms*

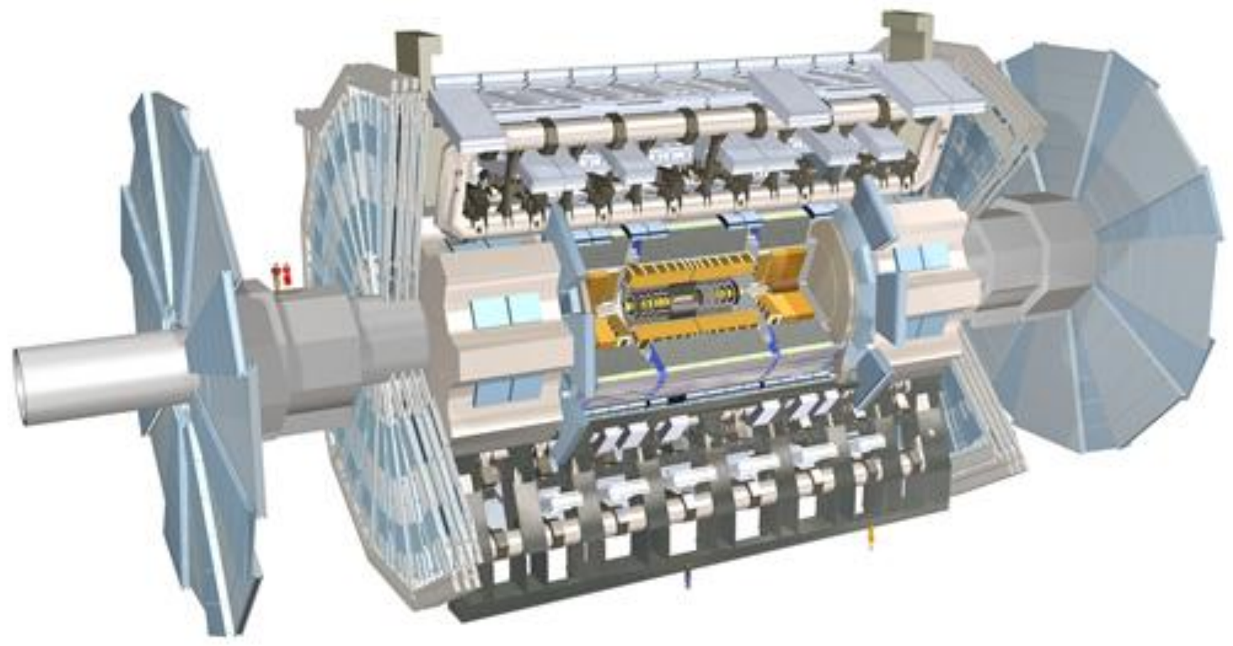
Typical ATLAS Data Processing Chain

Name
(Output Format)

Monte Carlo

Event Generation (EVNT) → Simulation (HITS) → Digitization/ Overlay (RDO)

Collision Data



(RAW)

Reconstruction (ESD, AOD) → Derivation (DAOD) → Analysis

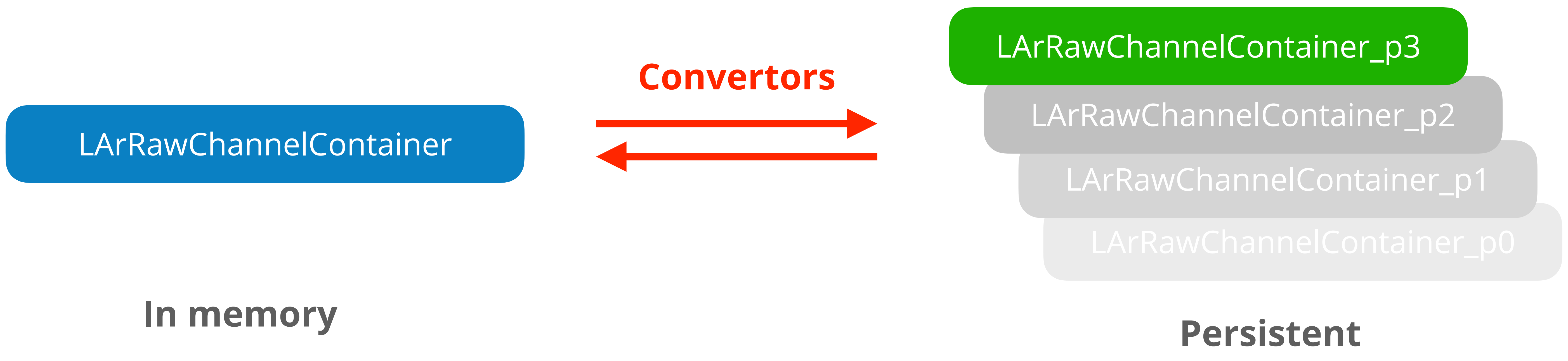
As one moves down the processing chain the complexity and event sizes go down

What is (the ATLAS) Event Data Model (EDM)?

- **HEP detectors/experiments (ATLAS, CMS, future ones at FCC ...) are typically highly complex**
  - They require advanced data model features to efficiently process sophisticated methods transiently
- **From the software side, it's extremely important to provide common data objects and interfaces that can be used across the entire experiment**
  - Same objects/concepts can be used by, e.g., simulation, reconstruction, physics analysis etc.
- **In a nutshell, Event Data Model (EDM) is a collection of -*insert language*- (e.g., C++) concepts (e.g., classes in OOP) that define a common set of detector/physics objects**
  - E.g., **Tracks** (from silicon hits), **Clusters** (from calorimeter cells), **Electrons**, **Muons**, etc.
- **All in all a good EDM:**
  - Allows efficient processing of highly complex algorithms that maximize throughput
    - Typically achieved by using advanced language concepts/features/data structures
  - Is isolated from the specific storage technology that is being used by the experiment at any time
    - For multi-decade experiments, this is extremely important!
- **Moreover, experiments have to read/write data for multiple decades!**
  - During such a long period of time a lot can (and will) change; processing methods, detectors, etc.
  - One way to deal with this is to separate the processing complexity (transient) from persistent data layout

Transient - Persistent EDM seperation

- **EDM can be separated into in-memory (transient) and on-disk (persistent)**
- **There are multiple reasons for adopting T/P seperation:**
  - One can better optimize compute efficiency (transient) and storage efficiency (persistent)
    - Usually not all transient information needs to (or can) be stored permanently
    - Transient EDM can get arbitrarily complex to ensure efficient processing of data
    - Persistent EDM can be much simpler and doing so will save storage space
  - One can support schema evolution by maintaining multiple persistent versions
    - Objects/definitions will unavoidably change over the course of the experiment!
- **ATLAS uses a hybrid EDM that partly adopts T/P separation (primarily for upstream data)**



- **The main UPSIDE:**
  - Flexibility and performance
- **The main DOWNSIDE:**
  - More code to write and maintain

(The ATLAS) Input/Output (I/O) system

- **Reading/writing data needs a robust Input/Output (I/O) system:**
  - Being able to accomplish this throughout the lifetime of the experiment (multiple decades)
  - Being able to adopt schema changes throughout the lifetime of the experiment
    - Backward, and sometimes forward, compatibility, i.e., read data you wrote 10-20 years ago!
  - Being able to do all these in the most compute/storage efficient way possible
    - Compute cycles are not cheap, neither is storage!
- **Although experiments typically use one main storage technology, it is ideal to:**
  - Adopt an abstraction layer that allows changing the storage backend relatively easily
    - Potentially adopt a second/third technology even if it is experimental
  - Avoid hardcoded specific technology API/features on the EDM/framework side
- **ATLAS' I/O system is based on the primary LCG POOL concepts**
  - The data storage broken down into a structured hierarchy:

  - Most importantly, this approach serves as an abstraction layer separating storage and EDM

(The ATLAS) Input/Output (I/O) system (cont’d)

- **Since the beginning of data taking, ATLAS has used ROOT’s TTree**
  - This basically means having a ROOT storage service that contains and implements:
    - RootDatabase, i.e., ROOT file-level operations, opening/closing TFile etc.
    - TreeContainer, i.e., ROOT TTree-level operations, creating, filling TTree/TBranch etc.
- **For LHC Run 4 (2029), ROOT’s primary I/O sub-system will be RNTuple**
  - A more modern, compute and storage efficient technology compare to TTree
    - It has a codified specifications (that is not yet finalized but getting there)
  - However, one important point is that it is **not a drop-in replacement of TTree**
    - For example, **it does not support raw pointers, polymorphism**, etc.
- **ATLAS has been working on adopting RNTuple for the last 1+ year(s) or so**
  - ATLAS development went in parallel to the ROOT development (feedback loop)
  - We’ve learned many invaluable lessons adopting this brand new storage backend

(ATLAS) Requirements from the storage backend

- **ATLAS needs a set features:**
  - Plain Old Data (POD), STL vectors (nested), user defined classes/enums
    - As an extension, we need some stdlib types, e.g., std::map etc.
  - User-defined collection proxies and late model extensions
    - Collection proxies hide various EDM complexities from the storage backend, i.e., ROOT I/O
    - Late model extensions accommodate data that arrives at any point during processing
  - Type-based user code execution when reading data a.k.a. Read Rules
    - This feature is needed for initializing (some) data for transient objects and schema evolution
  - A void* based interface to bind the I/O layer with the rest of the framework
- **The storage backend should ideally support all these core features**
  - Otherwise various compromises/adjustments need to be made
- **(Most of) These requirements apply to many (future) HEP experiments**

What have we learnt from RNTuple migration?

- **ATLAS can read/write all applicable data formats in RNTuple!**
  - Adoption in production still needs a lot of important work and testing/optimization/validation
  - Having said that, we're well within (event ahead of) the planned schedule
- **A few key earlier design choices eased our work immensely**
  - Having a simplified EDM when T/P seperation is not adopted (reconstruction to analysis)
  - Having T/P seperation for the more complex parts of the EDM (upstream data)
  - Most importantly, keeping EDM and storage backend completely separated
- **Most of our efforts were focused on collaborating with the ROOT team**
  - To make sure all features needed by ATLAS are in place
- **The separation between the EDM and the storage services meant**
  - We focused on adopting the new API in a few isolated storage services
  - No changes to the EDM and/or any of the client code were needed
    - The user doesn't know what the storage backend is and we actually use TTree/RNTuple simultaneously!

What should we keep in mind for future experiments?

- **Hardware landscape is ever evolving and we should be mindful of that**
  - Next gen. experiments will need to adopt a more heterogeneous computing model
  - When designing a brand new EDM one should keep portability in mind
    - When possible benefit/learn from experiment agnostic models such as EDM4Hep of Key4Hep
    - Having the simplest EDM that'll get the job done is typically the best option
- **Software landscape is also ever evolving and that should be kept in mind**
  - Specifically for I/O, pick the "best" backend for the use case but expect changes
    - Adopting abstraction layers and T/P separation can significantly ease switching the backend
    - Separating the EDM and specific storage backends provide the most flexibility
- **Avoid premature optimizations but never underestimate optimizations**
  - Compute cycles as well as storage are never cheap!
  - Be open-minded to lossless/lossy-smart data compressions when applicable
  - Embrace the fact that optimization is a never ending process that pays off!

Conclusions and outlook

- **Any new generation experiment should learn from predecessors**
  - In case of FCC-ee, the LHC experiments are of immense importance!
- **Data are the most important assets of any scientific endeavor**
  - Without a robust way to read/write data, any detector/experiment is practically useless
- **A good EDM and I/O system work in unison and:**
  - Allow efficient data processing while minimizing the storage footprint
  - Embrace emerging technologies and hardware/software developments
  - Adopt a healthy dose of separation to ease adopting different storage backends
- **In this talk we've looked at ATLAS as a use-case but the idea is universal**
  - Other large-scale experiments such as CMS also stick to comparable ideas
- **We have a deep knowledge and experience from current experiments**
  - We should certainly take advantage of this!


Argonne
NATIONAL LABORATORY
DEPARTMENT OF ENERGY
UNITED STATES OF AMERICA
U.S. DEPARTMENT OF
ENERGY