

## EIC Simulation Software

Whitney R. Armstrong

Argonne National Laboratory

March 9, 2017

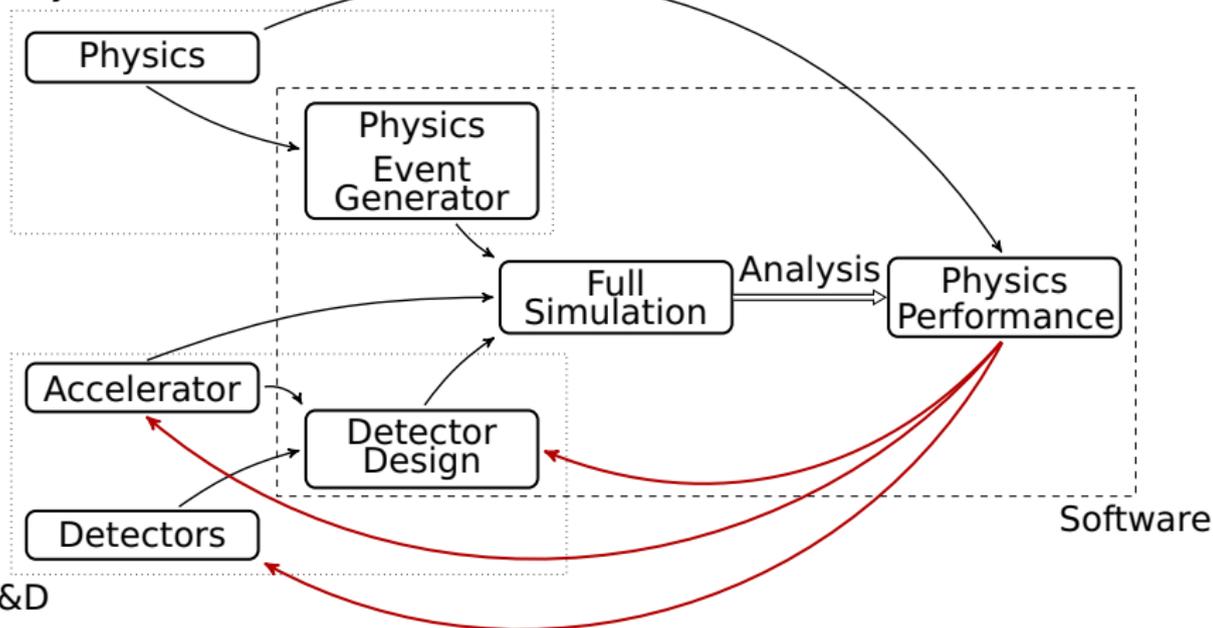


- 1 Software Overview
  - EIC Software Needs
  - Argonne's Software Effort
- 2 Detector Simulation and Analysis
  - Existing Frameworks
  - Our Software Plan
- 3 Recent ANL Progress
  - JLEIC Detector
  - SIEIC Detector
- 4 Future Work
- 5 Summary



# EIC software is very important

Theory

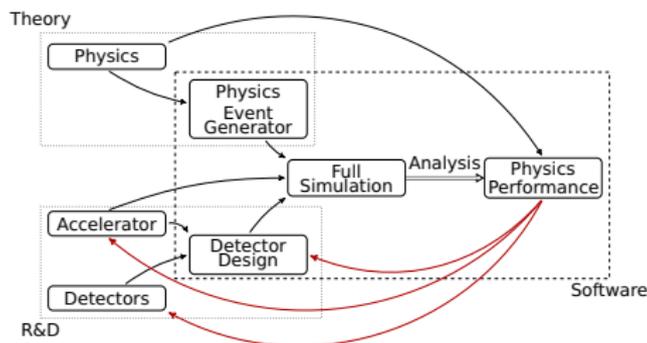


R&D

Software



# EIC Software Needs



Two broad categories:

## Event Generators

- 1 Physics input
- 2 MC event generators
- 3 Define the input for simulation
- 4 HepSim nicely provides targeted end point.

## Detector Simulation and Analysis

- 1 Simulate detectors (Geant4)
- 2 Digitize simulation
- 3 Reconstruct primary particles
- 4 Physics Analysis

# ANL software effort

- What are we doing at ANL?



## ANL software effort

- What are we doing at ANL?
- Where are we going?



# ANL software effort

- What are we doing at ANL?
- Where are we going?
- How to work together?



# ANL software effort

- What are we doing at ANL?



# Software for the future

We have recently identified the best path for future development.

## Good software is

- Maintainable and long-lasting
- Robust and Flexible
- Usable and easily extended
- not reinventing the wheel!
- always improving

I have spent a significant amount of time looking into the various HEP/NP software frameworks and tools.

It is important to reflect on what worked and what did not work.

### Framework flaws (opinion)

- eicROOT → tightly coupled, using many deprecated features
- fun4all → monolithic, experiment specific, non-starter for us
- GEMC → monolithic, tightly coupled, reinvented wheel (now wheels are square)
- slic + lcsim → unmaintainable, JAVA
- iLCSoft (pre-DD4hep) → tightly coupled
- iLCSoft (post-DD4hep) → flexible, **currently unwinding tightly coupled algorithms**

**Tightly coupled frameworks are bad!**

# Software Frameworks

## Good software

- Maintainable
- Flexible and Generic algorithms

## Main jobs of software

- 1 Simulate detectors (Geant4)
- 2 Digitize simulation
- 3 Reconstruct primary particles
- 4 Physics Analysis

# Our Software Plan

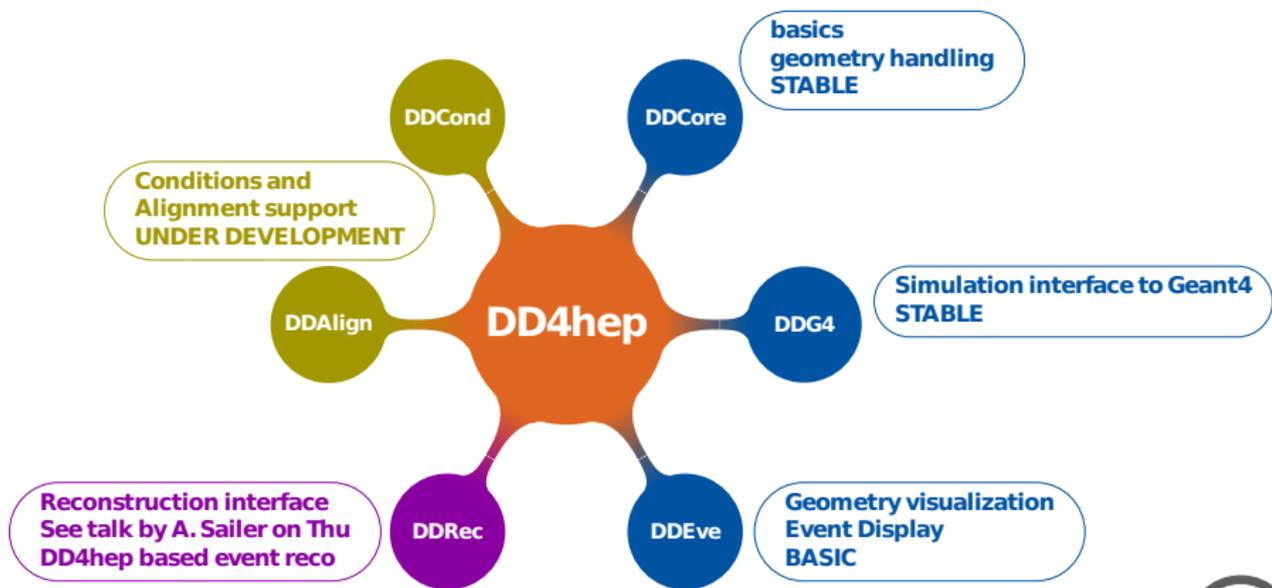
## Critical tools:

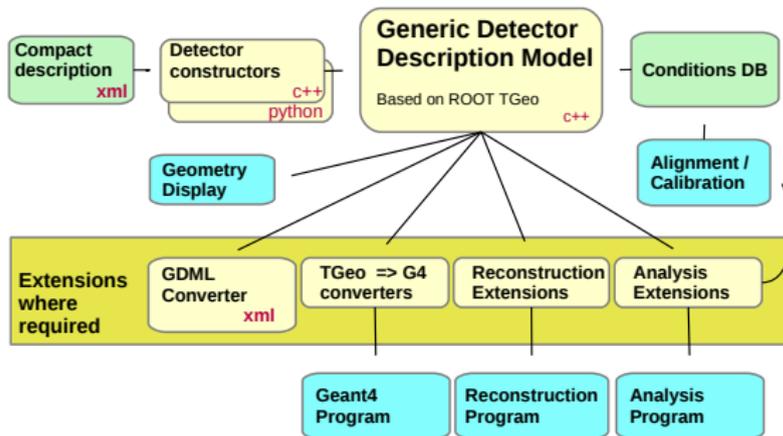
- Geant4
- ROOT
- DD4hep
- Marlin
- Collection of many marlin processors
- podio (future)
-

# DD4hep

The solution to the geometry problem

## Structure and packages

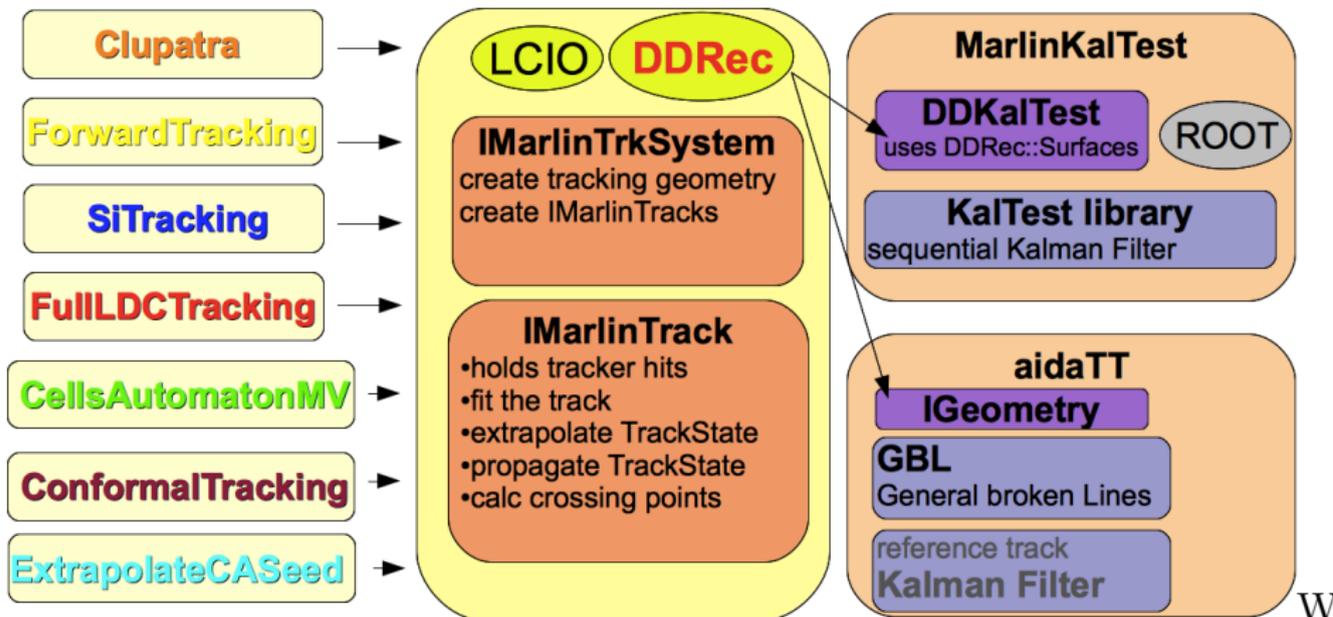




- Provides single source of geometry
- Geometry can be flexibly parameterized
- Was designed with long term application and use with real experiments

Only problem is convincing EIC/NP community → DD4NP?

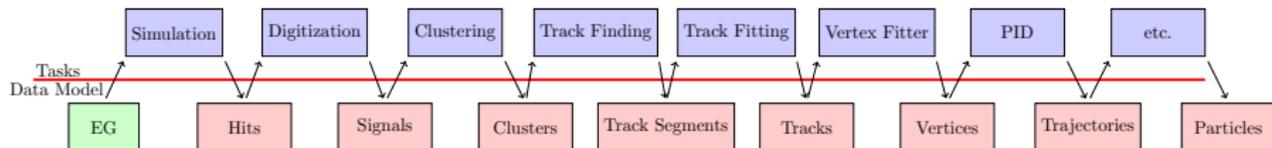
# Marlin



will use Marlin for organizing our post-simulation processing.  
In contact with lead developers planning how to improve Marlin.



- First step will be `ddsim`
- All final steps will be Marlin processors
- iLCSoft community is factorizing algorithms from existing processors to make more generic code



# Recent Progress

## Using SLIC+Icsim

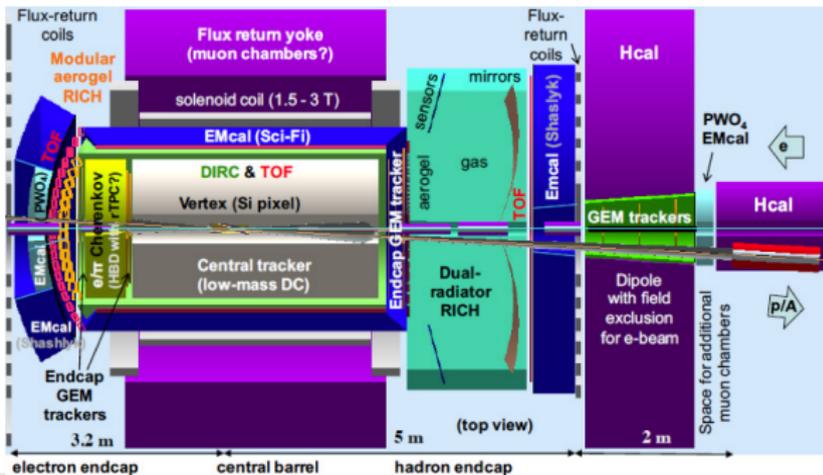
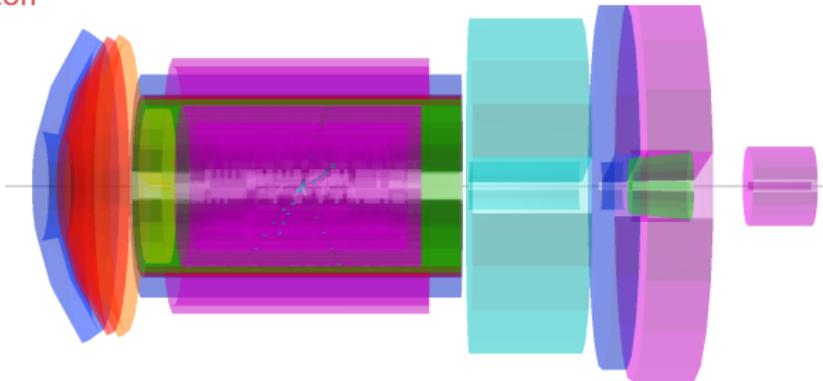
- Preliminary SiD study identifying silicon timing requirements (Repond)
- This should trigger the R&D efforts to begin in earnest.
- Full simulations of SiEIC and reconstruction on local nodes.
- Almost complete phased out slic for ddsim.

## DD4hep and reconstruction

- SiEIC and JLEIC geometries nearly completed
- New GenFit based track fitting works well.
- Developing GenFind track finder for initializing.

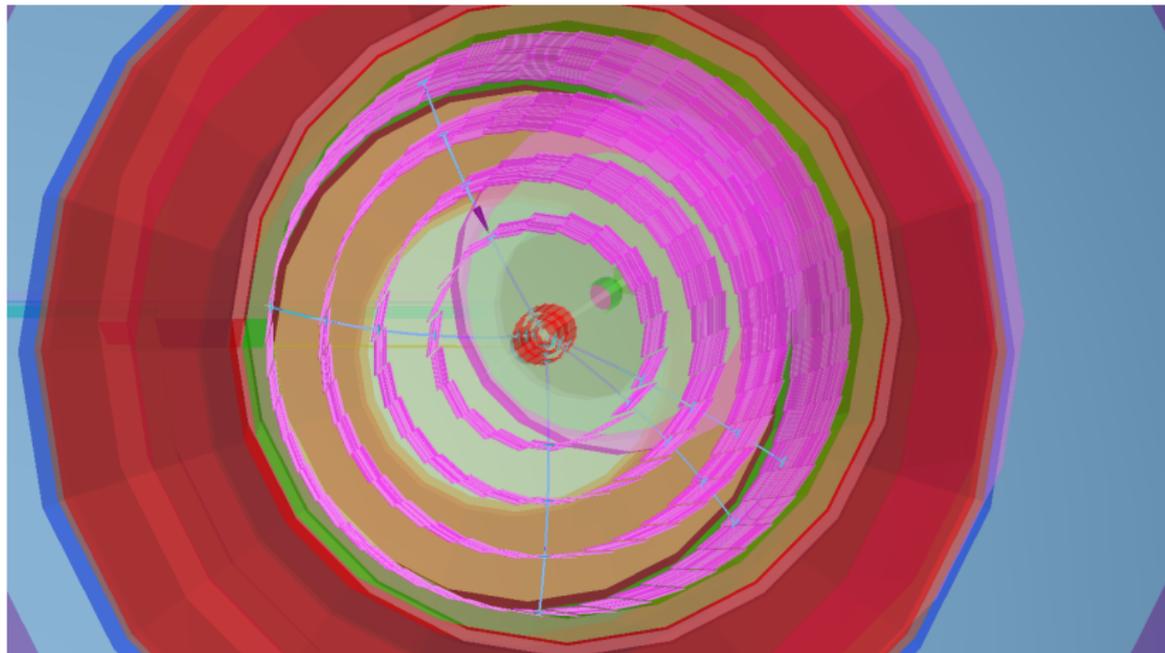
# JLEIC

Sereres Johnston



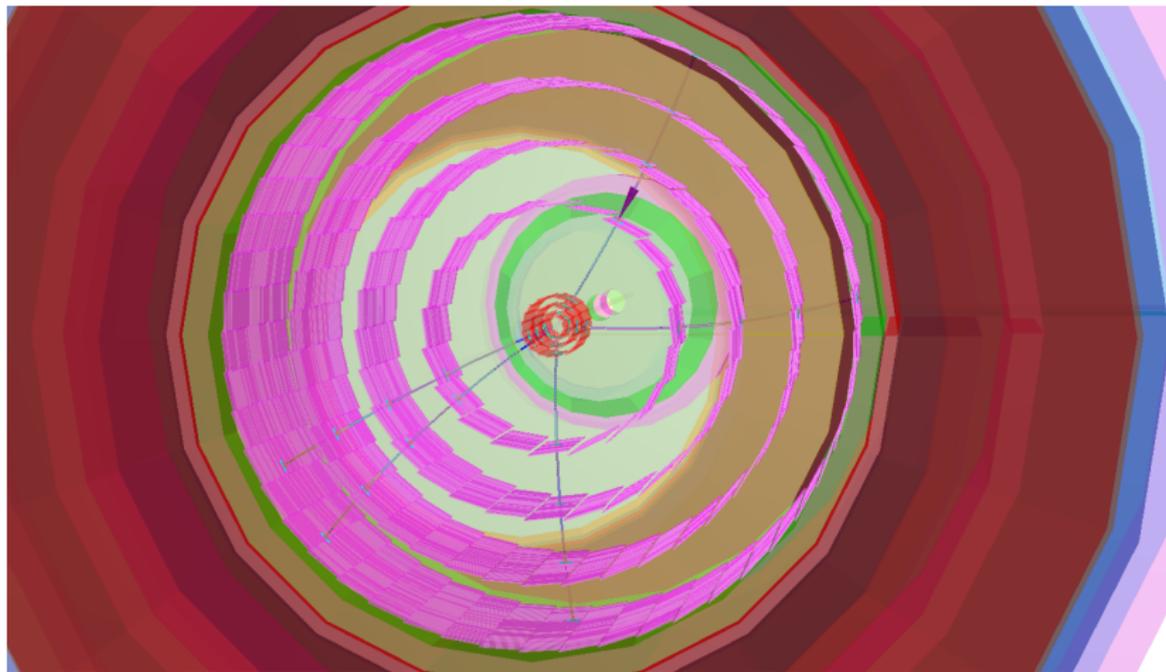
# JLEIC

## Reconstructed Tracks



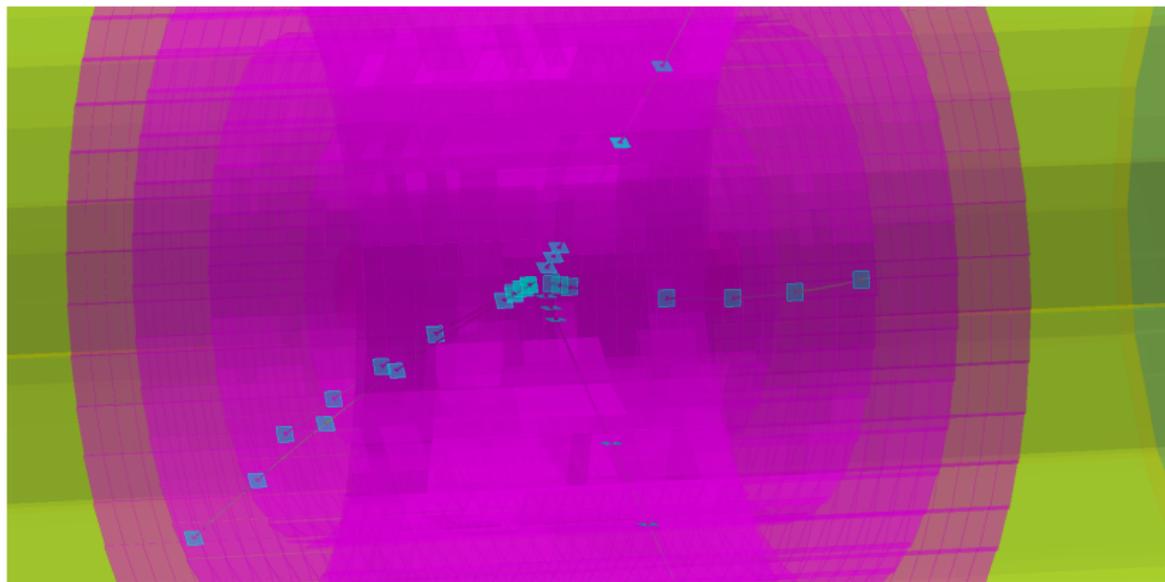
# JLEIC

## Reconstructed Tracks

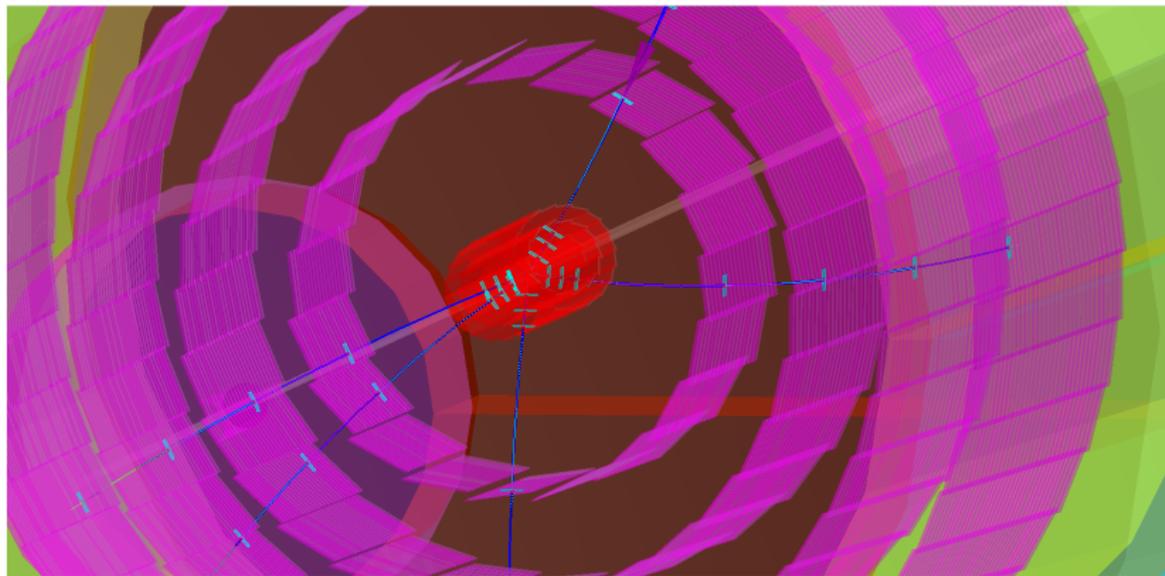


# JLEIC

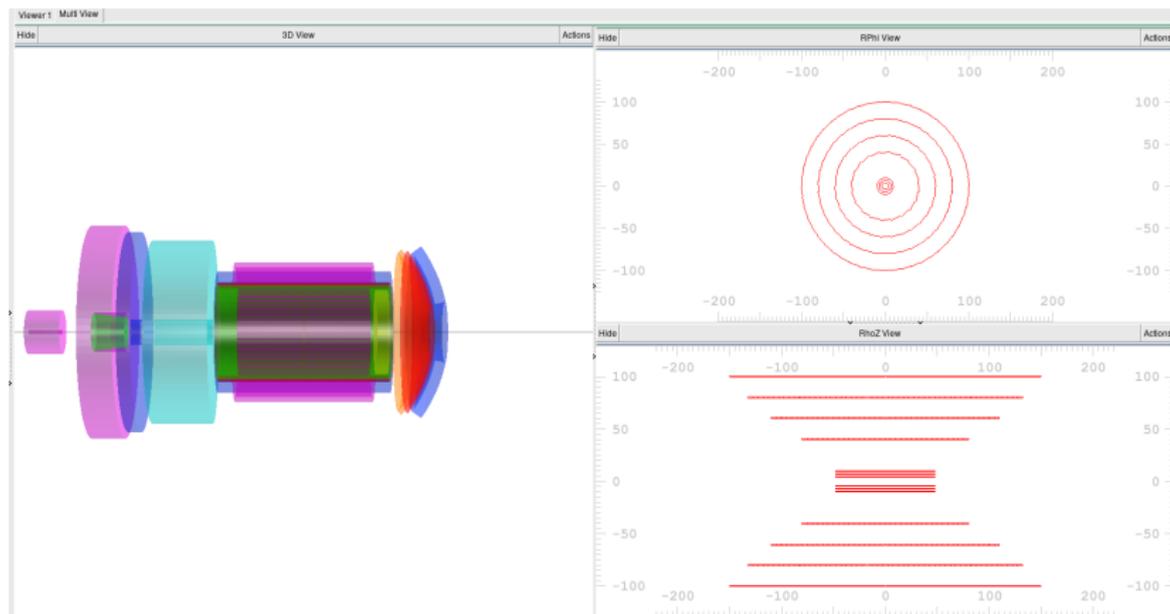
## Reconstructed Tracks



# Reconstructed Tracks

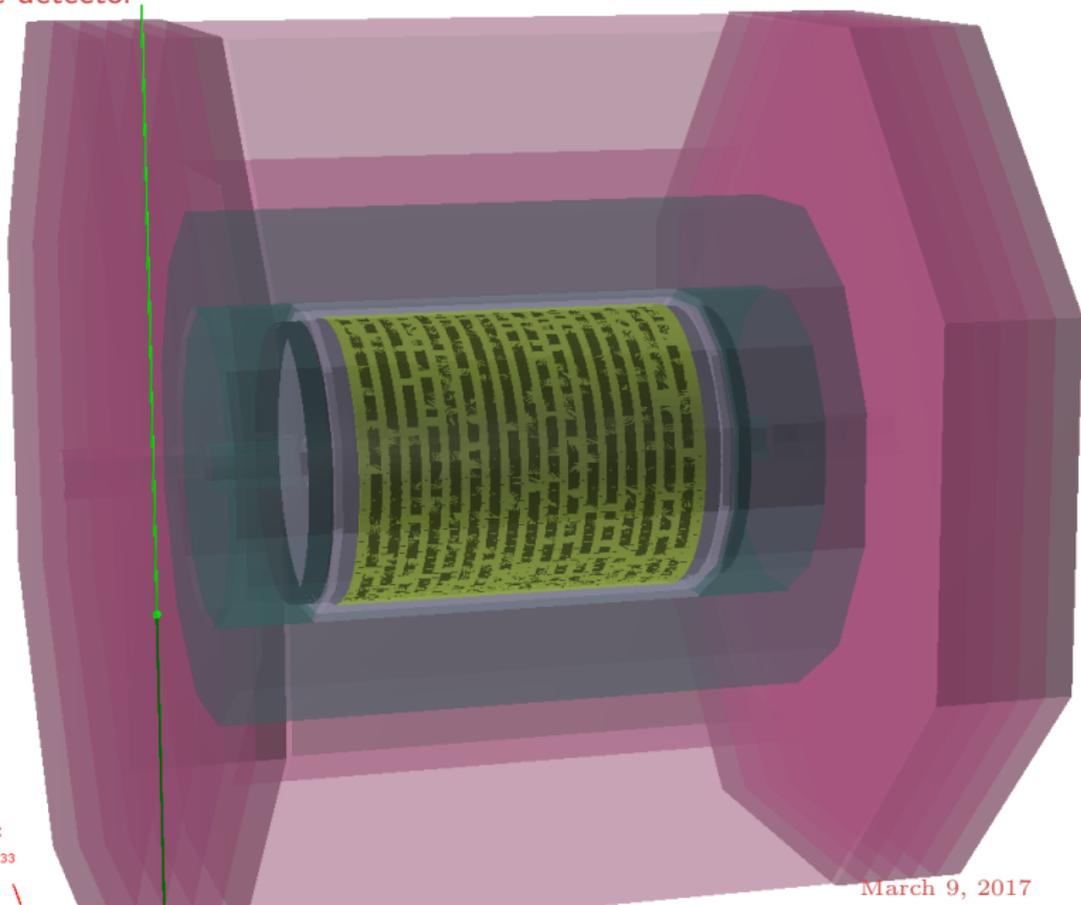


# Reconstructed Tracks



# SiEIC

SiD style detector

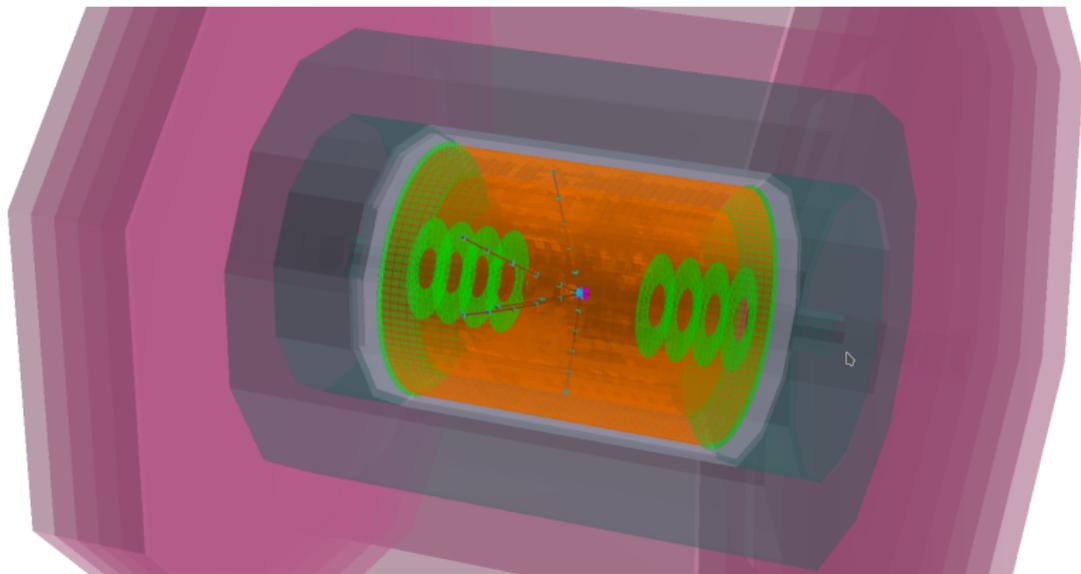


X  
533



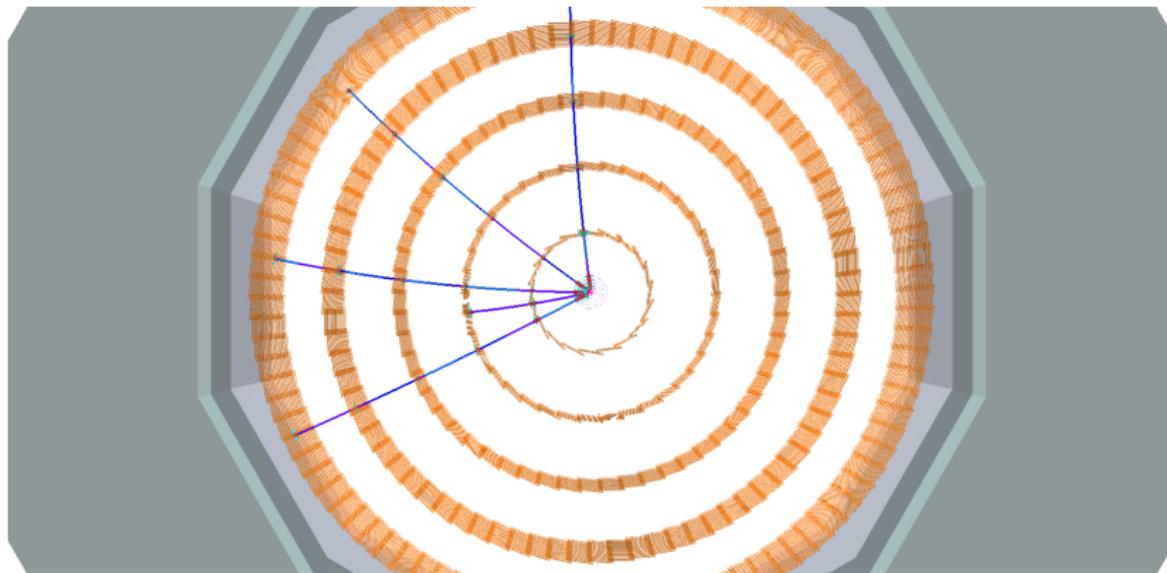
# SiEIC

## Reconstructed Tracks

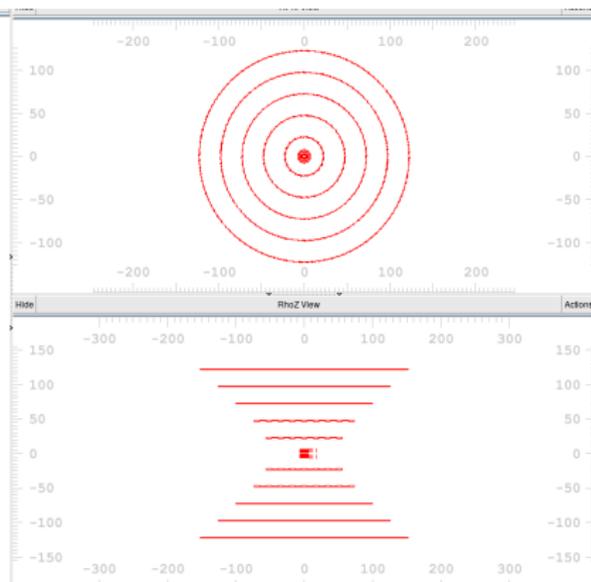
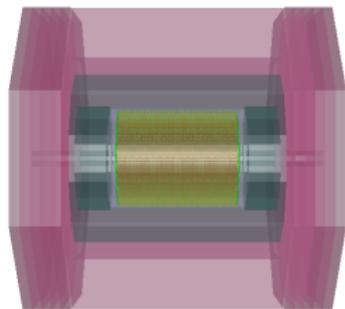


# SiEIC

## Reconstructed Tracks



# Reconstructed Tracks



# Future Work

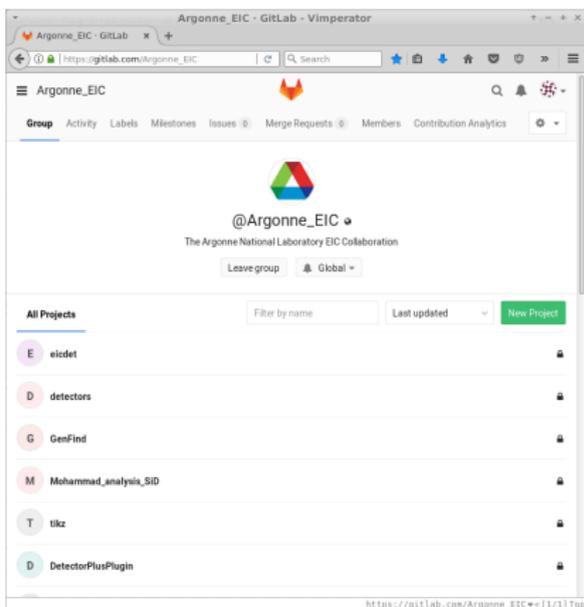
## Near-Term

- 1 Finish GenFind and write generic processor: GenFind + GenFit
- 2 Continue to improve on JLEIC geometry
- 3 Setup first common detector benchmarks (compare JLEIC to SiEIC)
- 4 Build software on jlab farm/CUE

## Long-Term

- 1 Simulate eRHIC detector designs
- 2 Adopt data model

# Summary



A lot of progress on simulation software and we look forward to collaborating with JLEIC and the broader EIC community.

## Collaborators Needed

- 1 We are using gitlab
- 2 Collaborators welcome
- 3 Public group soon...

## Useful links

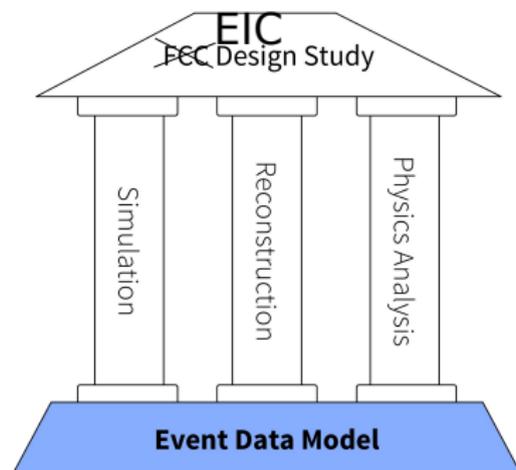
- [https://gitlab.com/Argonne\\_EIC](https://gitlab.com/Argonne_EIC)



# Backup Slides



# Common Data Model



from A. Zaborowska

- 1 A standardized data model is a great idea
- 2 Unfortunately not picked up by ESC → problem will only get worse with time

# Goal 1

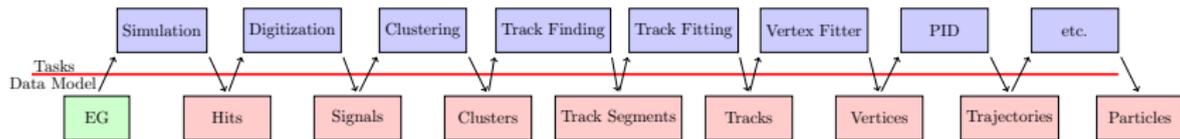
Is worth adopting a data model?

Yes!

## Motivation

Development of unified geometry, detector, tracking, and reconstruction tools for an EIC.

- The data model exists at the boundaries of each software task.



# Goal 1

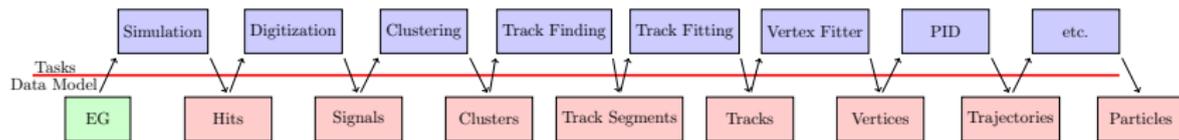
Is worth adopting a data model?

Yes!

## Motivation

Development of unified geometry, detector, tracking, and reconstruction tools for an EIC.

- The data model exists at the boundaries of each software task.



Example: I want to make an Event Display

- The Data Model are the **red boxes**
- I know exactly which kind of hit/cluster/tracks and how to read them
- Now I can just worry about the actual challenge of building an elegant and useful event display.

This principle holds for all tasks manipulating (input and output) data model objects.

Hopefully I have convinced you that a standardized data model is a  
good idea.  
OK, but what tool?



## Goal 2

What data model should we use?

### What are the requirements for a good data model?

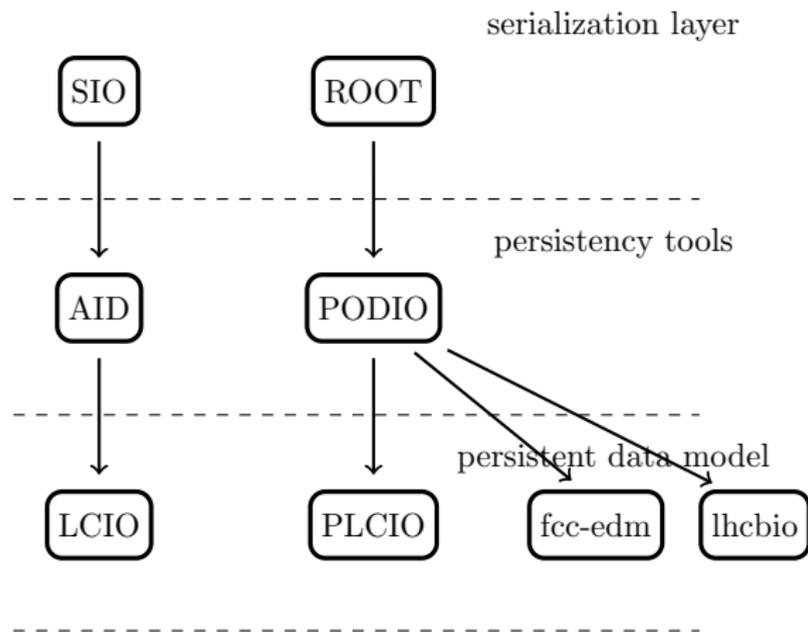
- Should be relatively static (ie, we don't want to create a new "Hit" class for every new detector/algorithm).
- ROOT compatible
- Maximum compatibility with existing libraries (frameworks?)
- What else?

### Examples of data models from other projects

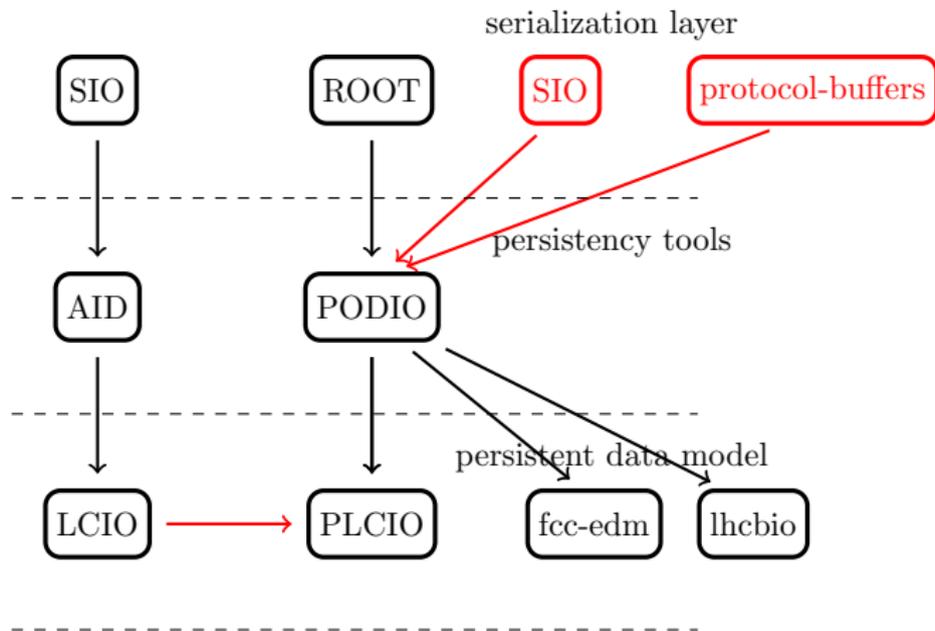
- LCIO (old)
- PLCIO
- lhcbio
- fcc-edm

What about the actual data serialization IO (file) format?

## Data Persistence for the near future



## Data Persistence for the near future



## Goal 3: Adopting the EDM



# Conclusion

- Can we do it?
- This decision has direct impact on how the tracking/geometry/detector objectives will proceed.

## Some useful links

- <https://github.com/hegner/podio>
- <https://stash.desy.de/projects/IL/repos/plcio/browse>
- <https://github.com/iLCSoft>
- <https://github.com/HEP-FCC>
- <http://ilcsoft.desy.de/v01-17-09/DD4hep/v00-15/doc/html/index.html>

# Introduction to the data persistency problem

## What is **persistent data**?

**Persistent data** denotes information that is infrequently accessed and not likely to be modified.

The opposite of this is **transactional data**.

– google



# Introduction to the data persistency problem

## What is **persistent data**?

**Persistent data** denotes information that is infrequently accessed and not likely to be modified.

The opposite of this is **transactional data**.

– google

## What is a **persistent data structure**?

**Persistent data structure** is a data structure that always preserves the previous version of itself when it is modified

– wikipedia

# Introduction to the data persistency problem

## What is **persistent data**?

**Persistent data** denotes information that is infrequently accessed and not likely to be modified.

The opposite of this is **transactional data**.

– google

## What is a **persistent data structure**?

**Persistent data structure** is a data structure that always preserves the previous version of itself when it is modified

– wikipedia

## What is data **serialization**?

**Serialization** is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment

– google

## Quick note on context

Here persistent data means the data that is used **between each** step of simulation/tracking/reconstruction. Thus it facilitates the development of simple or complex single purpose libraries.

There is a larger type of data persistence, of the data archiving type, which we are not talking about there.

We want a **quasi-persistent** data model, which from the view of the entire software chain, seems to be used in a transient way.

## Let's look at some HEP projects

- **SIO** (1999) - serial IO library
- **AID** (1999-2003?) - tool that generates code based on data model
- **LCIO** (2003) - A fixed but flexible persistent data model. Uses **AID** to define data structures and **SIO** for serialization.



## Let's look at some HEP projects

- SIO (1999) - serial IO library
- AID (1999-2003?) - tool that generates code based on data model
- LCIO (2003) - A fixed but flexible persistent data model. Uses AID to define data structures and SIO for serialization.

### LCIO is still in heavy use. Why has it successfully lasted this long?

- Is it really fast? ... Not really
- Does it have the best compression? ... No
- Are the data structures optimized for speed? ... No
- Has it been modernized with changing/improving language features? ... No

## Let's look at some HEP projects

- SIO (1999) - serial IO library
- AID (1999-2003?) - tool that generates code based on data model
- LCIO (2003) - A fixed but flexible persistent data model. Uses AID to define data structures and SIO for serialization.

### LCIO is still in heavy use. Why has it successfully lasted this long?

- Is it really fast? ... Not really
- Does it have the best compression? ... No
- Are the data structures optimized for speed? ... No
- Has it been modernized with changing/improving language features? ... No
- Has it been well maintained? ... **Yes.**
- Was it adopted by the community? ... **Yes.**
- Was it accessible with a variety of languages? ... **Yes.**

## Looking at LCIO's Success

- LCIO was successful because it was **flexible, maintained,** and **adopted by the developer community** (not the users).
- User community overwhelmingly adopted ROOT for everything (but no persistent data model).
- ROOT and LCIO do not work together!
- ROOT is clearly the tool of choice and will remain so.



## Looking at LCIO's Success

- LCIO was successful because it was **flexible, maintained, and adopted by the developer community** (not the users).
- User community overwhelmingly adopted ROOT for everything (but no persistent data model).
- ROOT and LCIO do not work together!
- ROOT is clearly the tool of choice and will remain so.

### We need a new library for the future!

It needs to

- Use ROOT IO for serialization layer.
- Develop tools for creating persistent data, making maximal use of ROOT tools as well.
- Read and write LCIO files to provide backward compatibility so we can use all the tools developed over the past 15 years.



# This is the most immediate software problem

We need to solve this problem ASAP!

- Use ROOT IO for serialization layer.
- Develop tools for creating persistent data, making maximal use ROOT tools as well.
- Read and write LCIO files to provide backward compatibility so we can use all the tools developed over the past 15 years.



# This is the most immediate software problem

## We need to solve this problem ASAP!

- Use ROOT IO for serialization layer.
- Develop tools for creating persistent data, making maximal use ROOT tools as well.
- Read and write LCIO files to provide backward compatibility so we can use all the tools developed over the past 15 years.

Fortunately, the FCC community is already working on it:

- PODIO - Plain-old-data IO (analog of AID) but uses ROOT and treats python as first class language.
- PLCIO - LCIO data model implementation with PODIO

Both of these projects are at an early stage but can be easily completed with more support from the community.



## This is the most immediate software problem

### We need to solve this problem ASAP!

- Use ROOT IO for serialization layer.
- Develop tools for creating persistent data, making maximal use ROOT tools as well.
- Read and write LCIO files to provide backward compatibility so we can use all the tools developed over the past 15 years.

Fortunately, the FCC community is already working on it:

- PODIO - Plain-old-data IO (analog of AID) but uses ROOT and treats python as first class language.
- PLCIO - LCIO data model implementation with PODIO

Both of these projects are at an early stage but can be easily completed with more support from the community.

### The real challenge...

**Getting library/toolkit/framework developers to agree to using the same event data model.**