

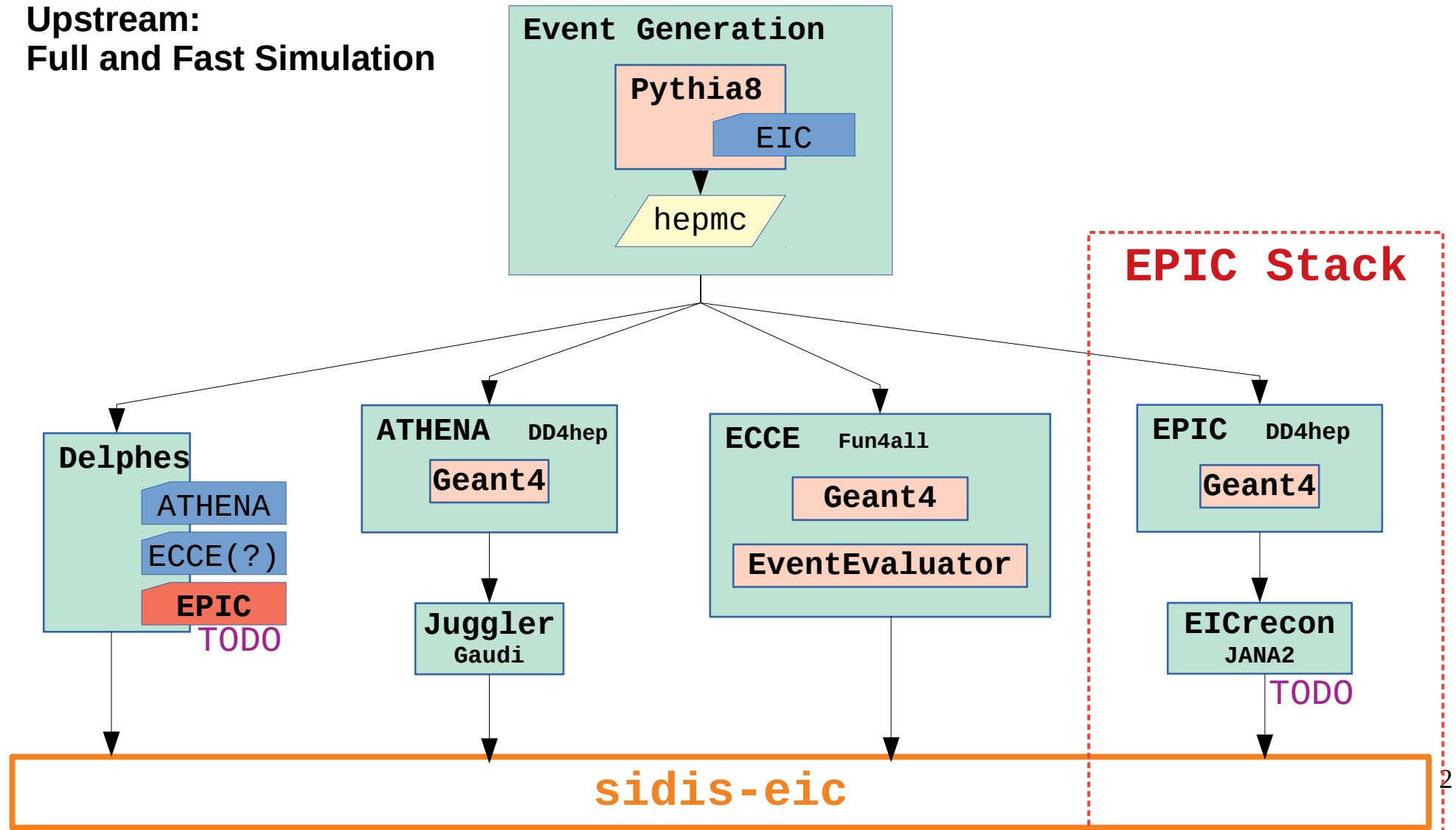
# SIDIS-EIC

Common Analysis Framework for SIDIS

<https://github.com/eic/sidis-eic>

Christopher Dilks  
7 September 2022

Upstream:  
Full and Fast Simulation





# Current Development Status

## ■ AnalysisDelphes

- Working, see tutorials and documentation to get started
- Uses configuration from ATHENA studies

## ■ AnalysisEpic

- Under development, working but not ready for common usage
- Tested only on physics benchmarks output

## ■ AnalysisEcce

- Compatible with ECCE (last tested with release `prop.5/prop.5.1/AI`)
- Hopefully compatible with the newest production; not tested, since not yet on S3)

## ■ AnalysisAthena

- Working

## Analysis

- Base class **Analysis** provides common functionality
  - Prepare() reads input and initializes:
    - Output data structures
    - Instances of **Kinematics**: one for truth and another for reconstructed
  - Finish() writes to output
  - Contains numerous configuration settings – specified externally by macros
    - Binning scheme
    - Reconstruction method
    - Final State (single hadrons, jets, ... )
  - Includes methods to fill output data structures, called by derived classes
- Derived classes **AnalysisDelphes**, etc. tuned to read respective ROOT files
  - Execute(): the main method to perform the analysis
    - Analysis::Prepare()
    - Event loop (with sub-loops over tracks, jets, ... )
      - Read input dataa
      - Set input variables of **Kinematics**
      - Call **Kinematics** calculation methods
      - Fill output data structures
    - Analysis::Finish()

## Kinematics

- 2 Instances: reconstructed and generated
- Input variables:
  - Beam momenta
  - Scattered electron
  - Hadronic Final State (HFS)
  - Single hadrons (SIDIS)
  - Jets
- Calculations:
  - CalculateDIS(): various reconstruction methods available (  $\rightarrow$  )
  - CalculateHadronicKinematics(): single hadron SIDIS variables
  - CalculateJetsKinematics(): jet variables
    - Uses fastjet, anti- $k_T$
    - Implemented only for **AnalysisDelphes**
- Output variables:
  - DIS:  $Q^2$ ,  $x$ ,  $y$ ,  $W$ , ...
  - HFS variables:  $\Sigma$ , ...
  - SIDIS Hadron:  $p$ ,  $p_T$ ,  $z$ ,  $\phi_h$ , ...
  - Jets:  $z$ ,  $p_T$ ,  $q_T$ , ...
- Includes boost functions

```
// reconstruction methods
void CalculateDISbyElectron();
void CalculateDISbyJB();
void CalculateDISbyDA();
void CalculateDISbyMixed();
void CalculateDISbySigma();
void CalculateDISbyeSigma();
```

# Current jet implementation in AnalysisDelphes

- Jet clustering using fast jet and Delphes energy flow objects with  $p_T > 0.1$  GeV
  - EFlowTracks, EFlowPhotons, EFlowNeutralHadrons
  - four-momenta from MC particle bank also clustered to get true jets
- Currently, semi-inclusive jets using anti-kT ( $R=0.8$ ) algorithm clustered, then used to calculate other variables/fill histograms
  - $z_h, j_\perp, q_T$  etc.

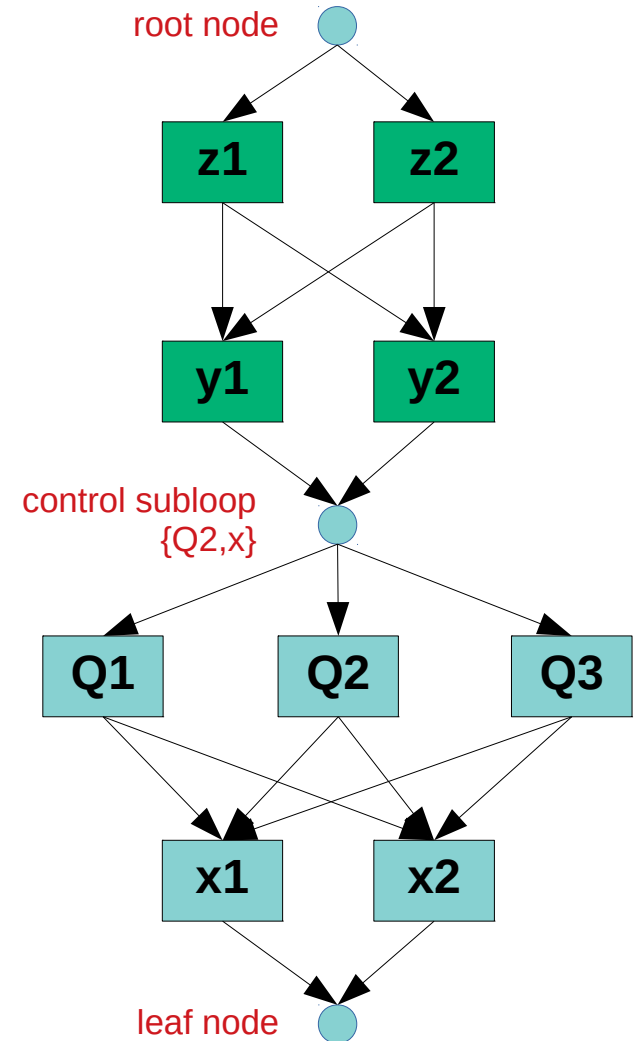
# Output Data Structures

**Adage:** Directed Acyclic Graph (DAG) that stores:

- **Data**, in arbitrary multi-dimensional bins and cuts
- **Algorithms**, executable during graph traversal
  - No nested for loops needed to analyze multi-dim. binned data
  - Allows for “binning agnostic” code
- Prototype developed within SIDIS-EIC

<https://github.com/c-dilks/adage>

4D Binning in  $(z,y,Q^2,x)$



























# Output Data Structures

- **SimpleTree** – flat TTree, useful for quick tests etc.
  - Reconstructed SIDIS variables
  - Has been used for SIDIS single-hadron asymmetries
  - Straightforward to connect to other analysis libraries and add more variables
- **Support for Custom Data Structures and Algorithms**
  - Existing data structures may not suit our future needs
  - Implement custom data structures
  - **TODO**: add plugin support
    - Class methods Prepare(), Fill(), Finish() = before all events, for each event, after all events
    - Support usage with Adage

## SimpleTree

 QSq	 Depol2
 X	 Depol3
 Y	 Depol4
 Z	 HadPID
 W	 Spin_idx
 MX	 SpinL_idx
 PhPerp	 Weight
 PhiH	
 PhiS	
 TruePhiH	
 TruePhiS	
 PolT	
 PolL	
 PolB	
 Depol1	

# How to Add a new Analysis: Macros

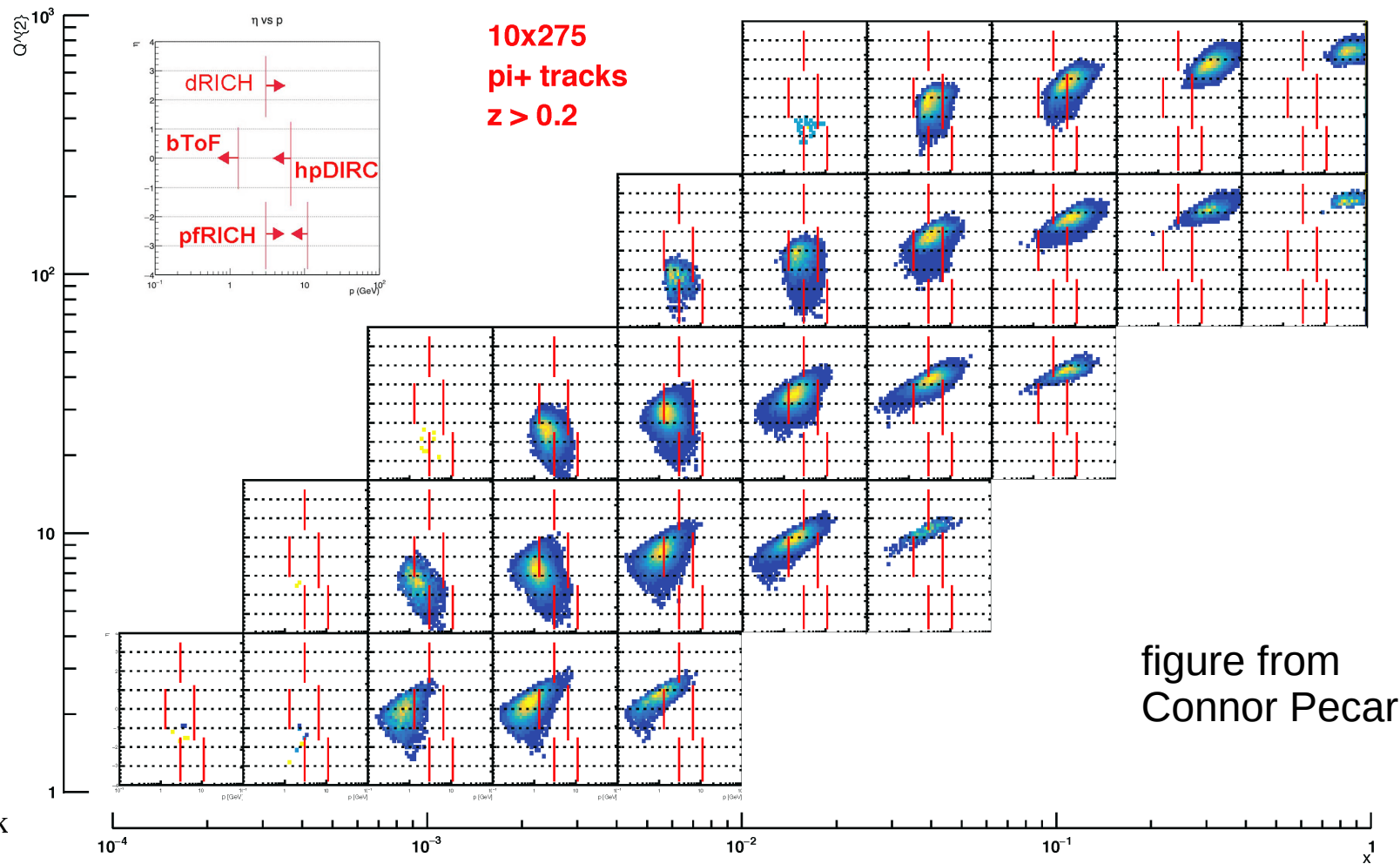
## ● Analysis Macro

- Choose AnalysisDelphes, AnalysisEcce, ...
- Configure
  - Reconstruction Method
  - Final states (single hadron, jets, ... )
  - Cuts
  - Binning Schemes (construct Adage)
- Call Execute()

## ● Post-Processing Macro (if using Adage)

- Define algorithms (lambdas), such as:
  - Draw histograms
  - Manipulate histograms (statistics, math, ...)
  - Common algorithms available in PostProcessor
- Configure Adage graph traversal
  - That is, *when* the algorithms should execute
- Call Execute()

# Example coverage plot: $\eta$ vs. $p$ in $(x, Q^2)$ bins, with PID limits



**Example benchmark plot:** pion  $z_{\text{rec}} - z_{\text{gen}}$ , from fast and full simulations, in  $(x, Q^2)$  bins

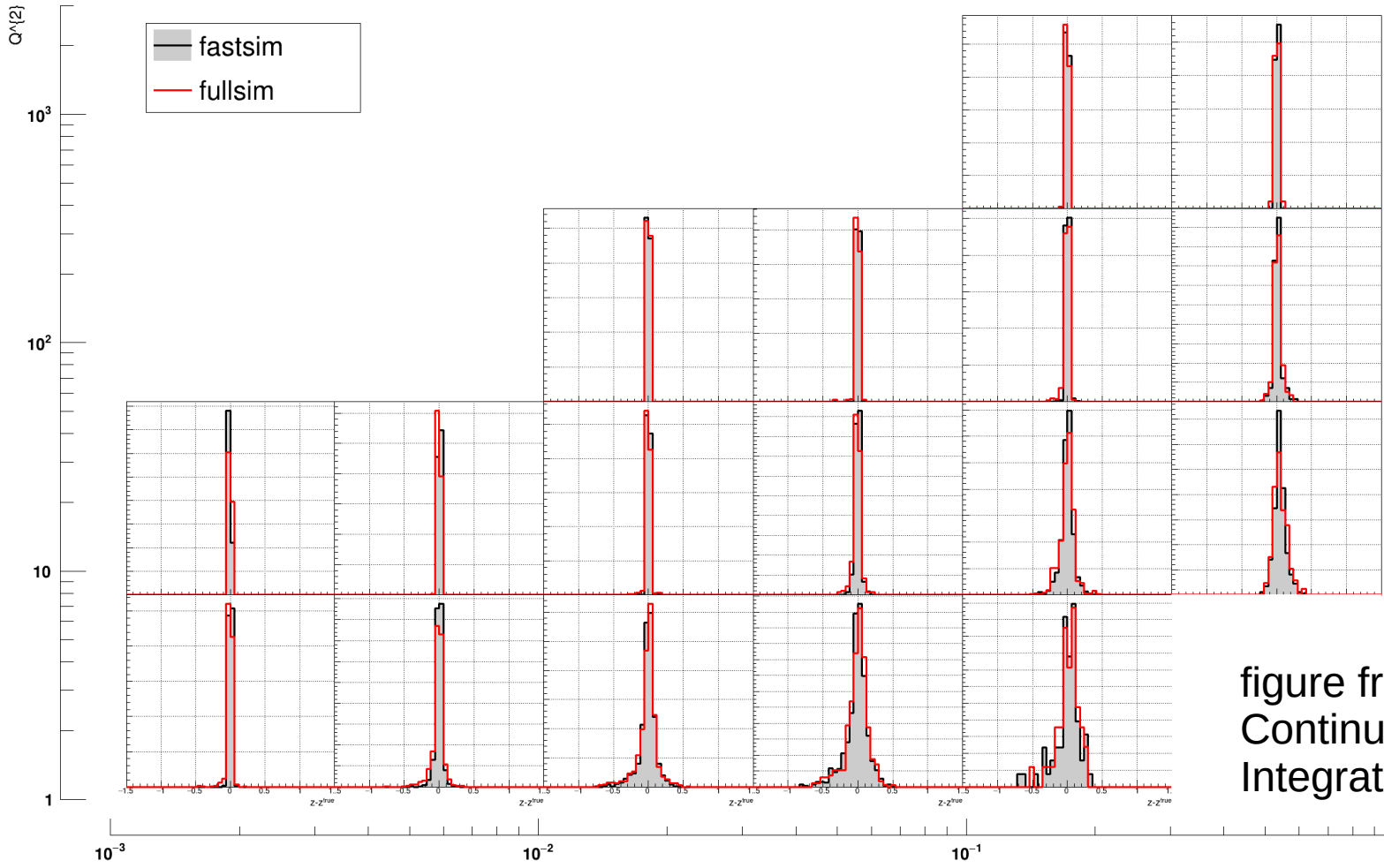
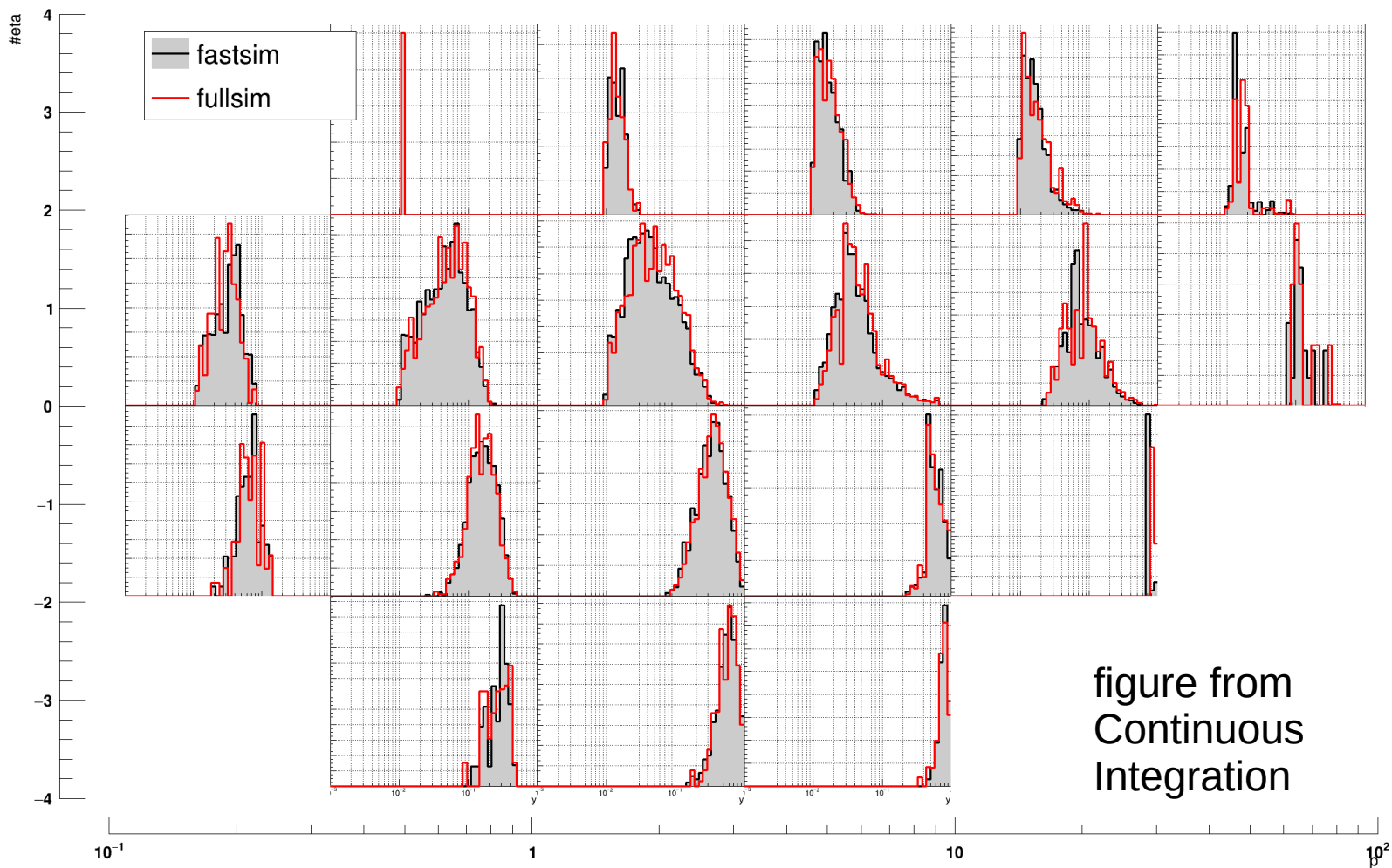


figure from  
Continuous  
Integration

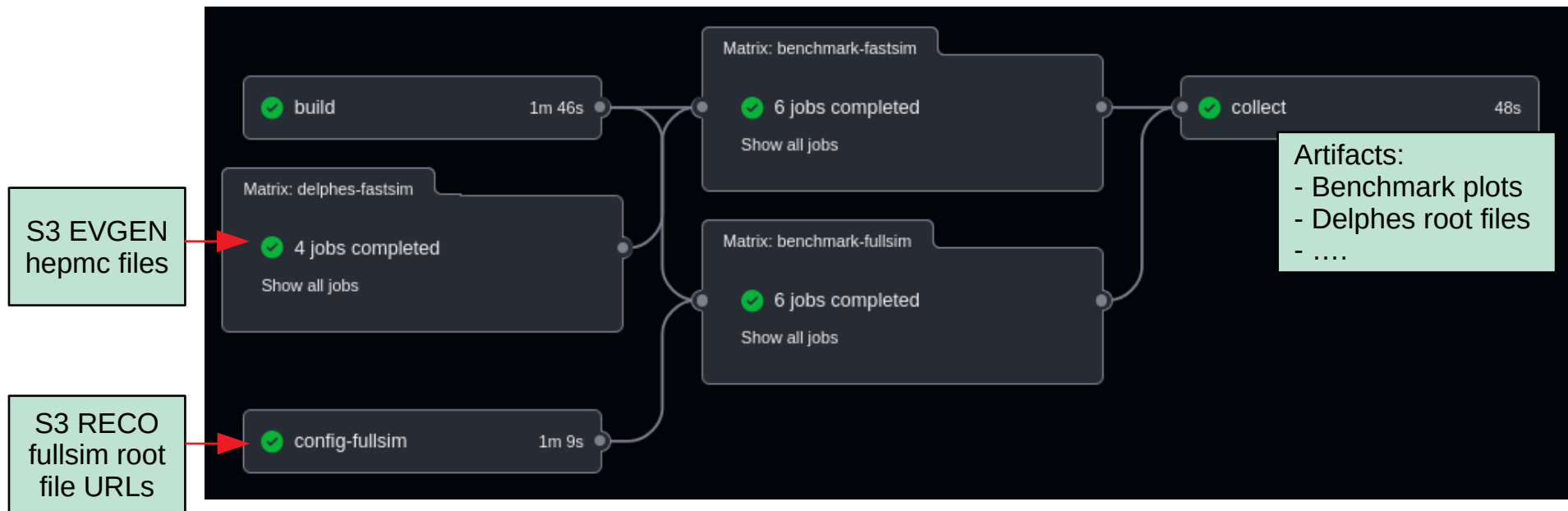
## Example benchmark plot: $y$ , from fast and full simulations, in pion ( $p, \eta$ ) bins



# Continuous Integration (CI)

Executes on every Git commit (pull request):

- Compile SIDIS-EIC
- Run Delphes → AnalysisDelphes
- Stream Full Simulation Output → Analysis DD4hep
- Output (artifacts)
  - Fast vs. Full simulation comparison plots: coverage, resolution, etc.
  - Effects of varying  $y_{\min}$  cuts
  - Add your plots, ROOT files, ... (limited to small statistics)

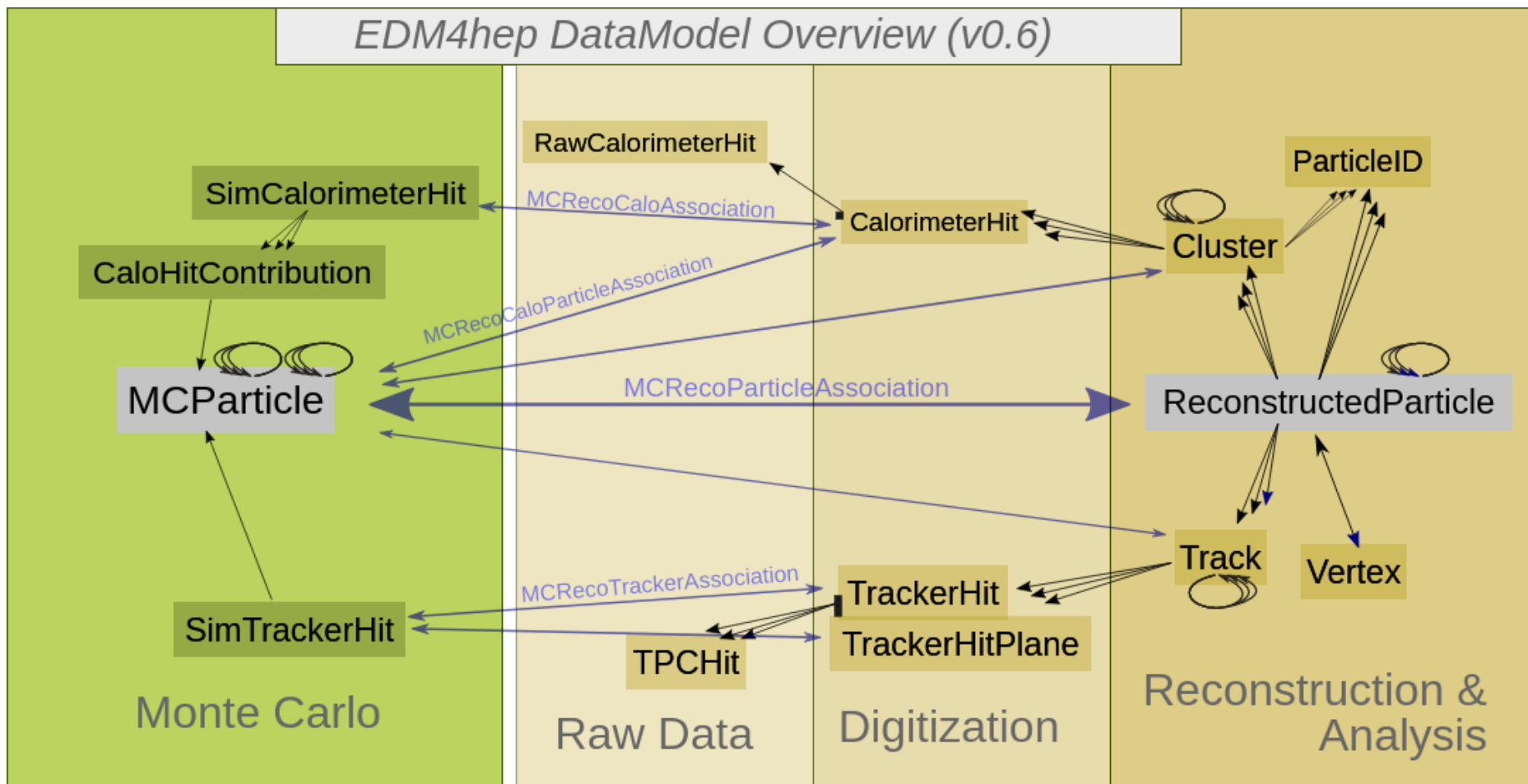


## The Upstream: EPIC Software Stack



## Data Model

- Common data model: [EDM4hep](#) + [EDM4eic](#)
- Based on [PODIO](#)





# Data Model

## YAML Specification

datatype example, defining a generated particle:

```
edm4hep::MCParticle:
  Description: "The Monte Carlo particle - based on the leio..."
  Author : "F.Gaede, DESY"
  Members:
    - int32_t PDG //PDG code of the par
    - int32_t generatorStatus //status of the parti
    - int32_t simulatorStatus //status of the parti
    - float charge //particle charge
    - float time //creation time of the pa
    - double mass //mass of the particle in
    - edm4hep::Vector3d vertex //production vertex
    - edm4hep::Vector3d endpoint //endpoint of the p
    - edm4hep::Vector3f momentum //particle 3-momen
    - edm4hep::Vector3f momentumAtEndpoint //particle 3-momen
    - edm4hep::Vector3f spin //spin (helicity)
    - edm4hep::Vector2i colorFlow //color flow a
  OneToManyRelations:
    - edm4hep::MCParticle parents // The parents of this parti
    - edm4hep::MCParticle daughters // The daughters this parti
```

Generated C++ class: edm4hep/MCParticle.h

```
public:

  /// Access the PDG code of the particle
  const std::int32_t& getPDG() const;

  /// Access the status of the particle as defined by
  const std::int32_t& getGeneratorStatus() const;

  /// Access the status of the particle from the sim
  const std::int32_t& getSimulatorStatus() const;

  /// Access the particle charge
  const float& getCharge() const;
```

```
unsigned int parents_size() const;
edm4hep::MCParticle getParents(unsigned int) const;
std::vector<edm4hep::MCParticle>::const_iterator parents_begin() const;
std::vector<edm4hep::MCParticle>::const_iterator parents_end() const;
podio::RelationRange<edm4hep::MCParticle> getParents() const;
unsigned int daughters_size() const;
edm4hep::MCParticle getDaughters(unsigned int) const;
std::vector<edm4hep::MCParticle>::const_iterator daughters_begin() const;
std::vector<edm4hep::MCParticle>::const_iterator daughters_end() const;
podio::RelationRange<edm4hep::MCParticle> getDaughters() const;
```

Relations can be one-to-one or one-to-many

Extra code can be included in any datatype

# Data Model

## edm4eic::ReconstructedParticle:

**Description:** "EIC Reconstructed Particle"

**Author:** "W. Armstrong, S. Joosten, F. Gaede"

### Members:

- int32\_t type // type of reconstructed particle. Check/set collection parameters Reconstruct
- float energy // [GeV] energy of the reconstructed particle. Four momentum state is not kept
- edm4hep::Vector3f momentum // [GeV] particle momentum. Four momentum state is not kept consistent inter
- edm4hep::Vector3f referencePoint // [mm] reference, i.e. where the particle has been measured
- float charge // charge of the reconstructed particle.
- float mass // [GeV] mass of the reconstructed particle, set independently from four vect
- float goodnessOfPID // overall goodness of the PID on a scale of [0;1]
- edm4eic::Cov4f covMatrix // covariance matrix of the reconstructed particle 4vector (10 parameters).

##@TODO: deviation from EDM4hep: store explicit PDG ID here. Needs to be discussed how we  
## move forward as this could easily become unwieldy without this information here.  
## The only acceptable alternative would be to store reconstructed identified  
## particles in separate collections for the different particle types (which would  
## require some algorithmic changes but might work. Doing both might even make  
## sense. Needs some discussion, note that PID is more emphasized in NP than  
## HEP).

- int32\_t PDG // PDG code for this particle

## @TODO: Do we need timing info? Or do we rely on the start vertex time?

### OneToOneRelations:

- edm4eic::Vertex startVertex // Start vertex associated to this particle
- edm4hep::ParticleID particleIDUsed // particle ID used for the kinematics of this particle

### OneToManyRelations:

- edm4eic::Cluster clusters // Clusters used for this particle
- edm4eic::Track tracks // Tracks used for this particle
- edm4eic::ReconstructedParticle particles // Reconstructed particles that have been combined to this particle
- edm4hep::ParticleID particleIDs // All associated particle IDs for this particle (not sorted by likelihood)

### ExtraCode:

**declaration:** "  
bool isCompound() const {return particles\_size() > 0;}\n  
"

# Data Model

Association datatypes link truth to reconstructed

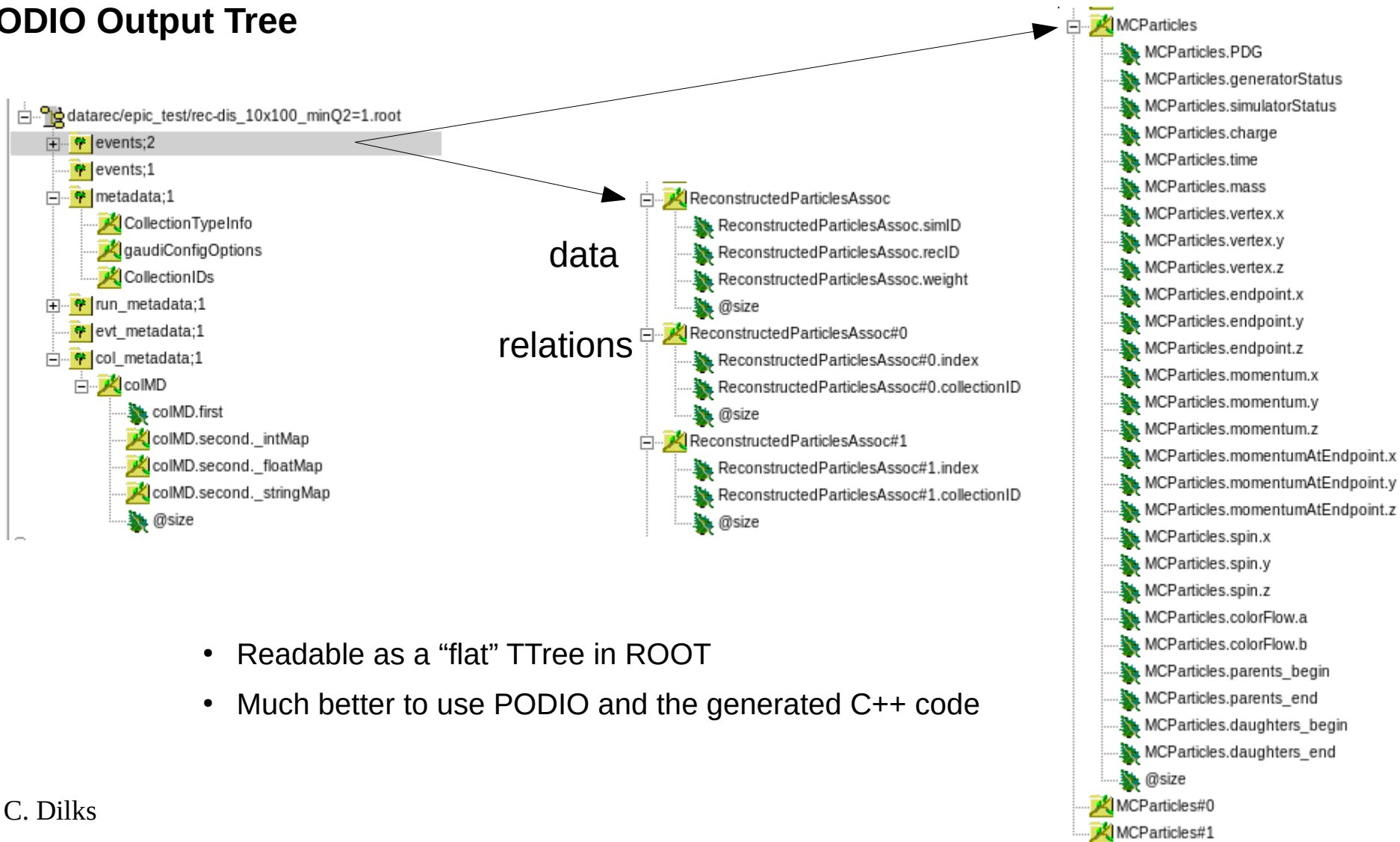
```
edm4eic::MCRecoParticleAssociation:
  Description: "Used to keep track of the correspondence between MC and reconstructed p
  Author: "S. Joosten"
  Members:
    - uint32_t      simID      // Index of corresponding MCParticle (positio
    - uint32_t      recID      // Index of corresponding ReconstructedPartic
    - float         weight     // weight of this association
  OneToOneRelations :
    - edm4eic::ReconstructedParticle rec // reference to the reconstructed particle
    - edm4hep::MCParticle sim           // reference to the Monte-Carlo particle
```

Common calculations: inclusive kinematics

The datatype is generic and usable by all inclusive reconstruction algorithms

```
edm4eic::InclusiveKinematics:
  Description: "Kinematic variables for DIS events"
  Author: "S. Joosten, W. Deconinck"
  Members:
    - float      x      // Bjorken x (Q2/2P.q)
    - float      Q2      // Four-momentum transfer squared [GeV^2]
    - float      W      // Invariant mass of final state [GeV]
    - float      y      // Inelasticity (P.q/P.k)
    - float      nu      // Energy transfer P.q/M [GeV]
  OneToOneRelations:
    - edm4eic::ReconstructedParticle scat // Associated scattered electron (if identified)
    ## @TODO: Spin state?
    ## - phi_S?
```

# PODIO Output Tree



- Readable as a “flat” TTree in ROOT
- Much better to use PODIO and the generated C++ code

```
1 // create PODIO event store
2 std::vector<std::string> infiles = { "file1.root", "file2.root", "file3.root" };
3 podioReader.openFiles(infiles);
4 evStore.setReader(&podioReader);
5
6 // event loop =====
7 for(unsigned e=0; e<podioReader.getEntries(); e++) {
8
9     // next event
10    if(e>0) { evStore.clear(); podioReader.endOfEvent(); };
11
12    // read collection of generated particles
13    const auto& simParts = evStore.get<edm4hep::MCParticleCollection>("MCParticles");
14
15    // loop over generated particles
16    for(const auto& simPart : simParts) {
17        auto simPDG = simPart.getPDG();
18        //...
19    }
20
21    // loop over associations of reconstructed and generated particles
22    for(const auto& assoc : evStore.get<edm4eic::MCRecoParticleAssociationCollection>("ReconstructedParticlesAssoc")) {
23        auto simPart = assoc.getSim(); // truth
24        auto recPart = assoc.getRec(); // reconstructed
25        // ...
26    }
27 }
28 podioReader.closeFile();
```

Currently testing on possibly-incomplete data from benchmarks:

```
// repo:      physics_benchmarks, from https://eicweb.phy.anl.gov/EIC/benchmarks/physics_benchmarks
// CI stage:   finish
// CI job:     summary
// CI artifact: results/dis/10on100/minQ2=1/rec-dis_10x100_minQ2=1.root
```

Cross check with upstream kinematics calculations in  
InclusiveKinematics Juggler algorithm look promising!!

AnalysisEpic output:

```
KINEMATICS, calculated from upstream: -----
      x      Q2      W      y      nu
Truth    0.0544  1.6932  5.5034  0.0078  16.5729
Electron -0.0617  1.7230 -nan      -0.0070 -14.8865
DA        0.0554  1.7002  5.4667  0.0077  16.3627
JB        0.0409  0.9290  4.7615  0.0057  12.1075
Sigma     0.0776  1.7425  4.6480  0.0056  11.9721
KINEMATICS, calculated locally in SIDIS-EIC, with method "Ele": -----
      x      Q2      W      y      nu
Truth    0.0544  1.6932  5.5034  0.0078  16.5729
Reconstructed -0.0617  1.7230 -5.3644 -0.0070 -14.8858
DIFFERENCE: upstream(Electron) - local(Ele): -----
      x      Q2      W      y      nu
Truth    -0.0000  -0.0000  -0.0001  0.0000  0.0000
Reconstructed 0.0000  0.0000 -nan      0.0000 -0.0007
```

# Algorithms

EPIC Stack Development:

- Algorithms – framework-independent, general algorithms
- EICRecon – reconstruction framework (JANA2)

- An **algorithm** transforms **datatype(s)** to **datatype(s)**
  - For example:
    - Digitization : simulated hits → raw hits
    - Reconstruction : raw hits → reconstructed objects
- Algorithms *can be* independent of reconstruction framework
- They can be applied anywhere, even after reconstruction

## dRICH\*:

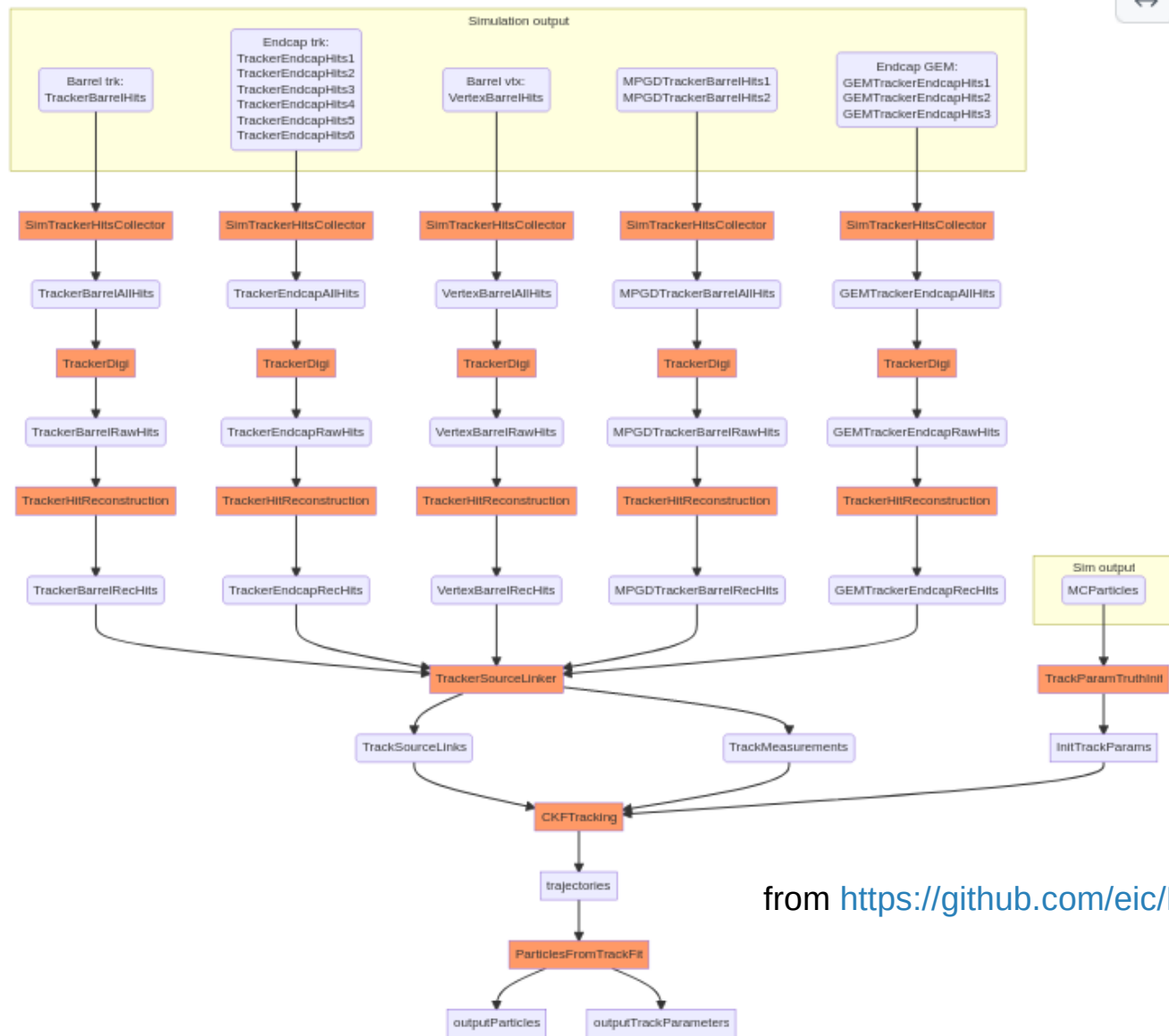
\*under re-development (IRT)



from <https://github.com/eic/ElCrecon/issues/11>

# Algorithms

## Tracking:

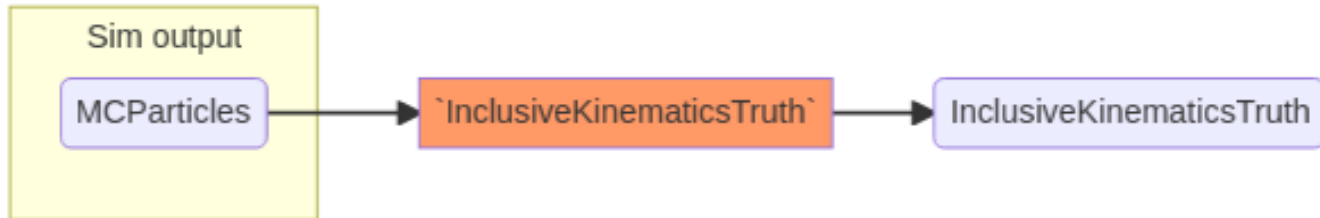


from <https://github.com/eic/ElCrecon/issues/11>



# Algorithms

## Inclusive Kinematics:



+ others for reconstructed kinematics, with various methods

Electron

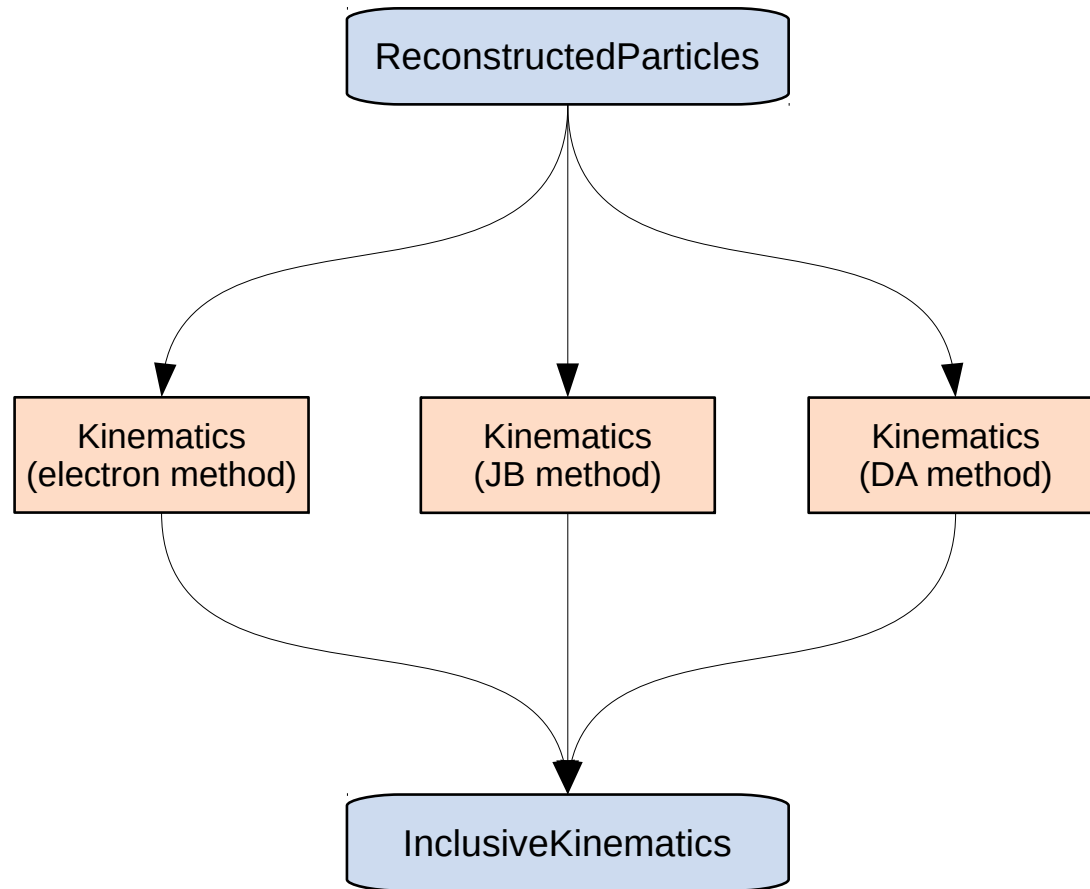
JB

Double Angle

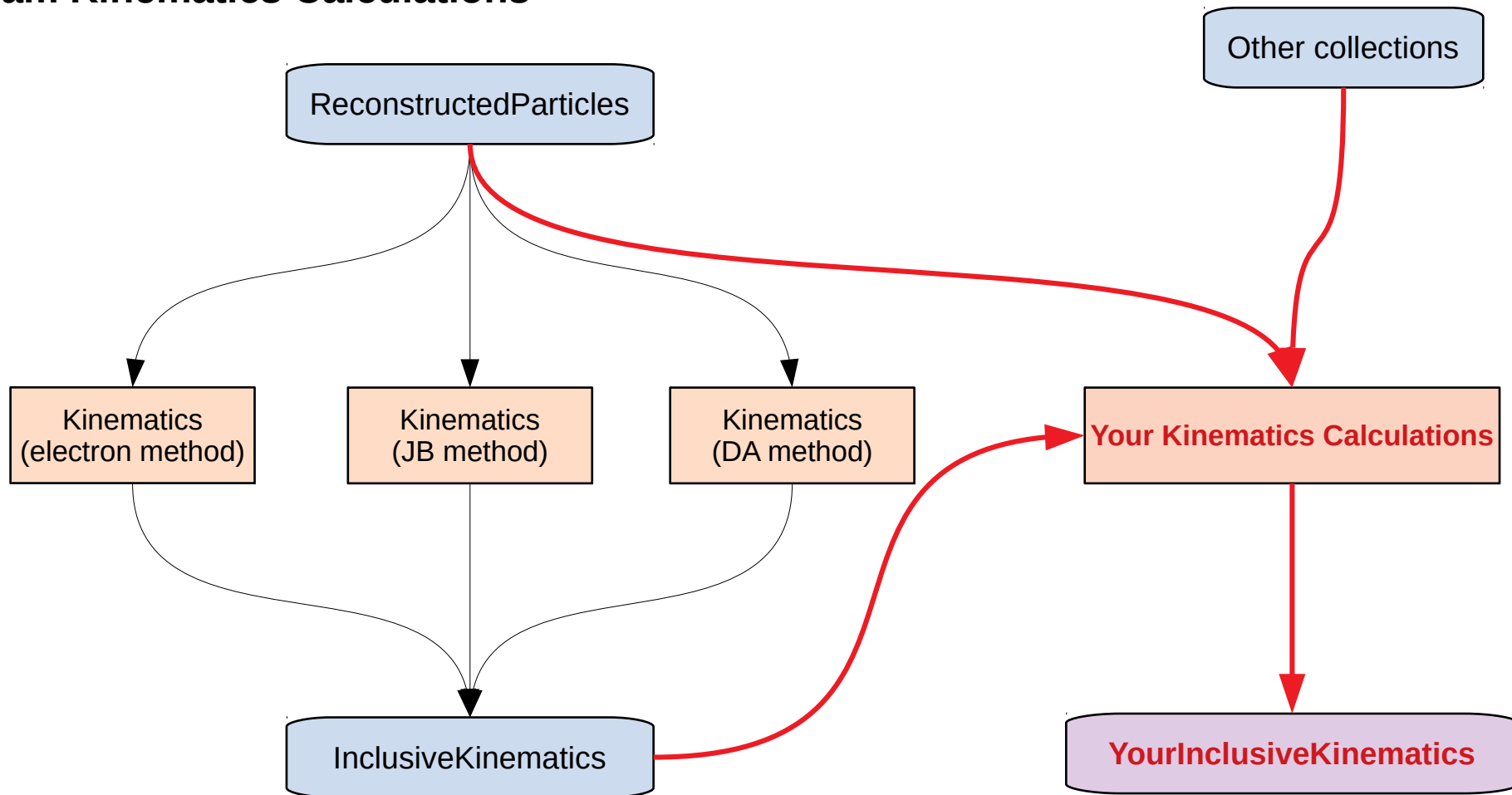
Sigma

from <https://github.com/eic/ElCrecon/issues/11>

# Upstream Kinematics Calculations



# Upstream Kinematics Calculations



- Pushing kinematics calculations upstream does not stop development of custom kinematics calculations
- Common anything should be pushed as far upstream as possible

# What to put where?

## Upstream:

- Electron finder
- Inclusive kinematics reconstruction methods (electron, JB, ...)
  - Implemented EDM4eic Datatype: InclusiveKinematics (x,Q2,W,y,Nu), 1-1 relation to electron
- Jet Reconstruction algorithms
  - Possible Datatype: Jet (z,p<sub>T</sub>,...), 1-N relation to ReconstructedParticles
- SIDIS kinematics calculations
  - Possible Datatype: SIDISparticle (p<sub>T</sub>,φ<sub>h</sub>,...)
- Benchmarks (for Continuous Integration)
  - Any plots or tests that can be made with small statistics

## Common Analysis Framework (SIDIS-EIC, ...)

- Support analysis algorithms now, for the short term, while the general upstream framework is coming together
- Application of common analysis techniques
  - Cuts
  - Multidimensional binning and plotting
  - Q2 weighting
  - Automation for high statistics analysis (streaming/downloading data, job submission, ...)
- Benchmarks: plots and tests that need *larger* statistics
- Comparisons with Fast Simulations (Delphes)
  - May need to convert Delphes output to a compatible EDM4eic format to use upstream framework-independent algorithms
- Support for legacy simulation software stacks (ATHENA & ECCE)

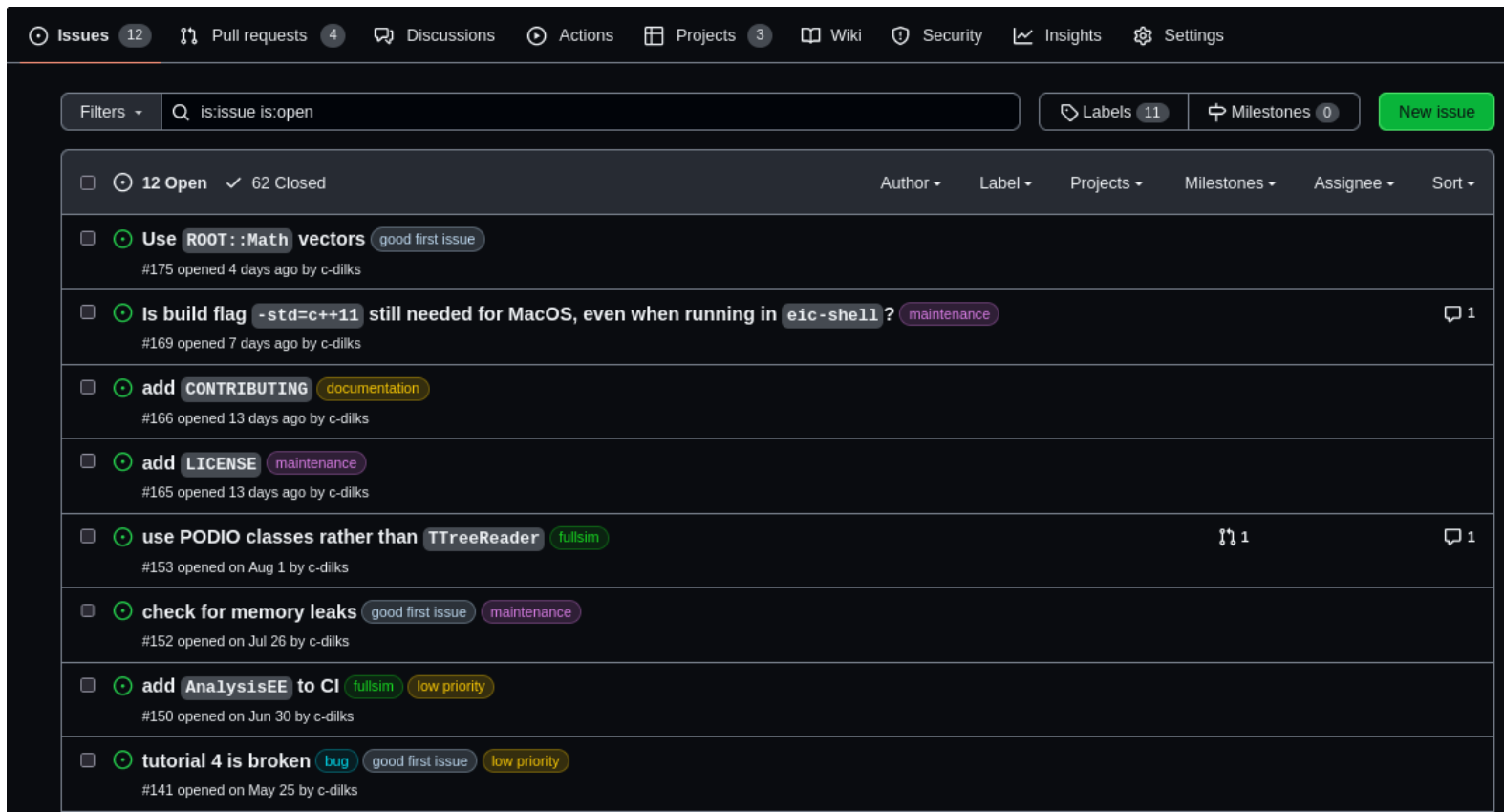
## Contributions to SIDIS-EIC, or to *any* EPIC Software

- See recent SW tutorial:
  - <https://eic.github.io/tutorial-setting-up-environment/>

### **In general:**

- Join EIC Organization: <https://github.com/eic>
  - How to join: <https://github.com/eic/.github/blob/main/profile/README.md>
- Join EPIC Devs team: <https://github.com/orgs/eic/teams/epic-devs>
  - After you are a member of EIC Organization
  - Click 'Members Tab → Request to Join'
  - Grants 'push' access to SIDIS-EIC and other EPIC repositories
- Every repository follows the usual Github development workflow:
  - Issue → Pull Request → Code Review → Approval → Merge

# Issues == TODO List



The screenshot shows a GitHub repository's 'Issues' page. The top navigation bar includes links for Issues (12), Pull requests (4), Discussions, Actions, Projects (3), Wiki, Security, Insights, and Settings. Below the navigation bar, there's a search bar with the query 'is:issue is:open'. To the right of the search bar are buttons for 'Labels' (11) and 'Milestones' (0), and a green 'New issue' button. The main content area displays a list of 12 open issues. Each issue entry includes a checkbox, a green circle icon, the issue title, labels, the issue number, the time since it was opened, the author, and a comment icon. The issues are as follows:

- ☐ **Use ROOT: :Math vectors** (good first issue) #175 opened 4 days ago by c-dilks
- ☐ **Is build flag -std=c++11 still needed for MacOS, even when running in eic-shell?** (maintenance) #169 opened 7 days ago by c-dilks
- ☐ **add CONTRIBUTING** (documentation) #166 opened 13 days ago by c-dilks
- ☐ **add LICENSE** (maintenance) #165 opened 13 days ago by c-dilks
- ☐ **use PODIO classes rather than TTreeReader** (fullsim) #153 opened on Aug 1 by c-dilks
- ☐ **check for memory leaks** (good first issue, maintenance) #152 opened on Jul 26 by c-dilks
- ☐ **add AnalysisEE to CI** (fullsim, low priority) #150 opened on Jun 30 by c-dilks
- ☐ **tutorial 4 is broken** (bug, good first issue, low priority) #141 opened on May 25 by c-dilks

Add your own if you want to work on something, or just start a pull request...