

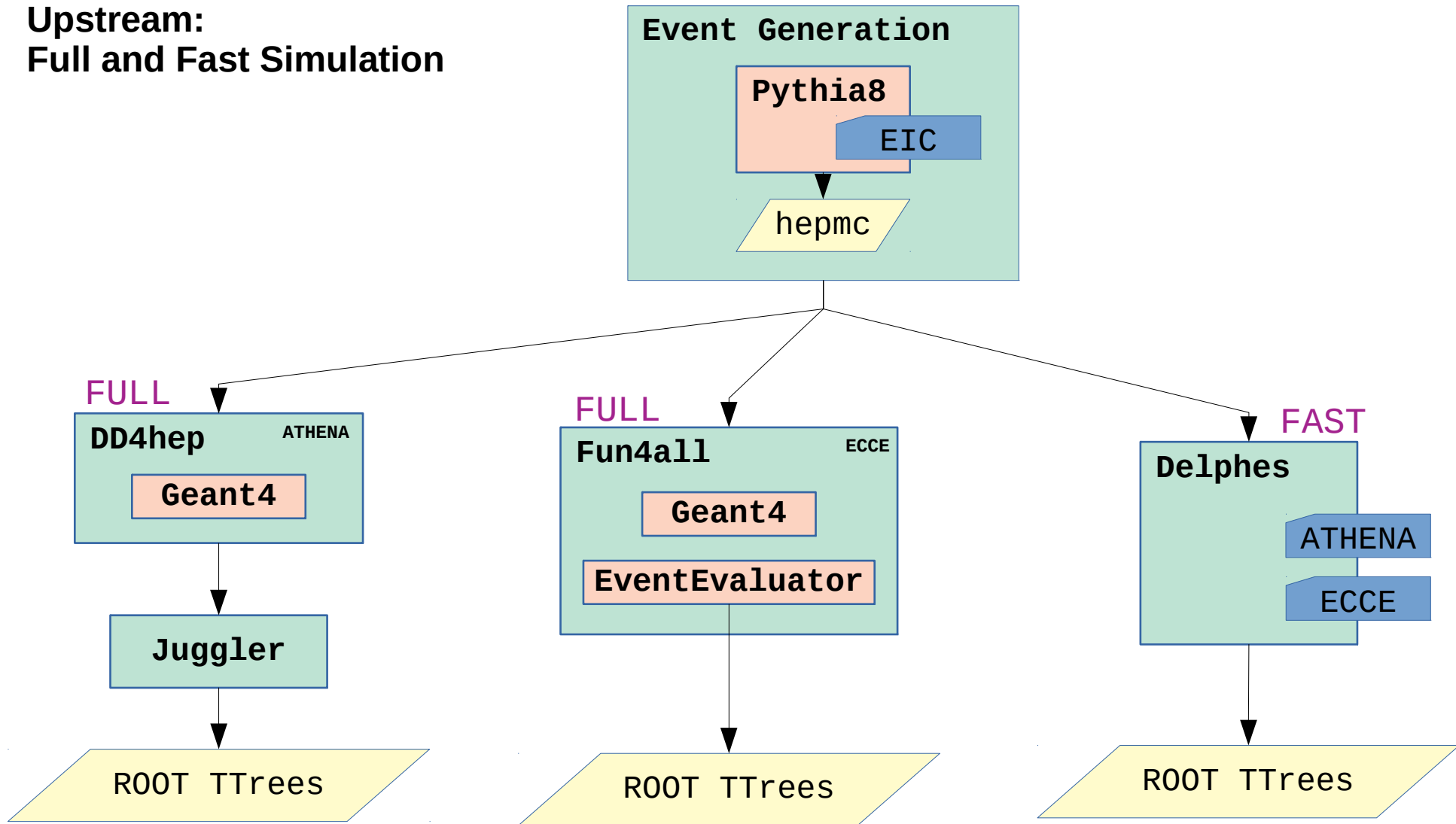
SIDIS-EIC

Common Analysis Framework for SIDIS

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

Christopher Dilks
24 May 2022

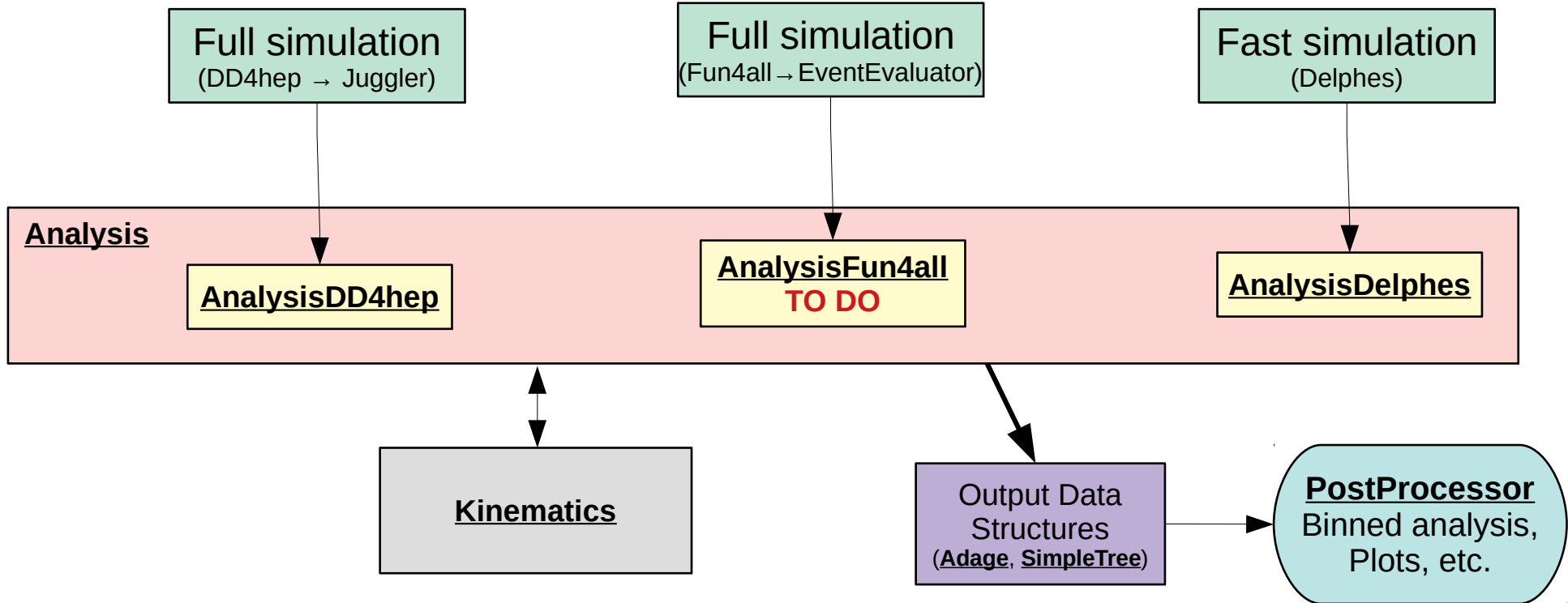
Upstream: Full and Fast Simulation



SIDIS-EIC Common Analysis Framework

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

Classes are underlined



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
 - 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 trees
 - Execute(): the main method to perform the analysis
 - Analysis::Prepare()
 - Event loop (with sub-loops over tracks, jets, ...)
 - Read input TTree variables
 - 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 (→)
 - CalculateHadronicKinematics(): single hadron SIDIS variables
 - CalculateJetsKinematics(): jet variables
 - Uses fastjet
 - Implemented only for **AnalysisDelphes**
 - **TODO: implement in AnalysisDD4hep & AnalysisFun4all, or separately (AnalysisJets)?**
- Output variables:
 - DIS: Q^2 , x , y , W , ...
 - HFS variables: Σ , ...
 - SIDIS Hadron: p , p_T , z , ϕ_h , ...
 - Jets: z , p_T , q_T , ...
- Includes boost functions

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

DETAILS:

<https://github.com/c-dilks/sidis-eic/blob/docker-base-jug-xl/doc/kinematics.md>

[to be merged to main branch soon]

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

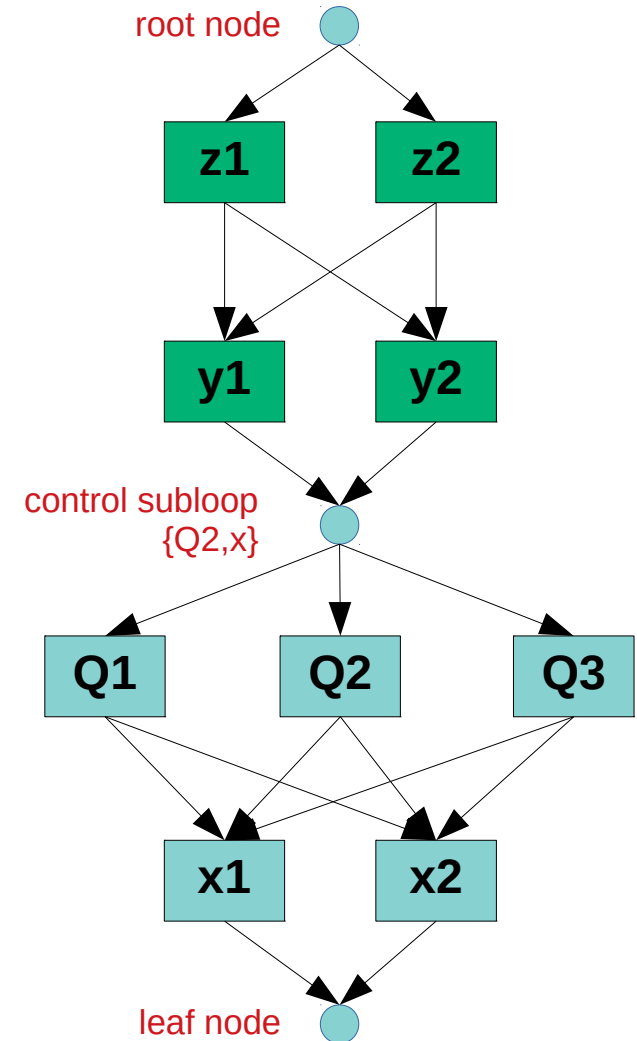
Adage: Directed Acyclic Graph (DAG) that stores:

- **Data**, in arbitrary multi-dimensional bins and cuts
 - 1 “layer” of nodes = 1 variable’s bins or cuts
 - Layers are fully connected to adjacent layers
 - 1 multi-dimensional bin = 1 full graph path from root node to leaf node
 - Stores associated set histograms (**Histos**, **HistosDAG**)
 - **TODO**: generalize to store anything
- **Algorithms**, executable during graph traversal (no nested for loops!)
 - Run “payload algorithm” on every bin or on any subset of bins
 - Graph layers can be re-ordered (switches inner and outer “loops”)
 - Allows for “binning agnostic” code
- Prototype developed within SIDIS-EIC

In practice:























- 1) Define your bins
- 2) Define your algorithms
- 3) Run

4D Binning in (z,y,Q²,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

<u>SimpleTree</u>	
 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

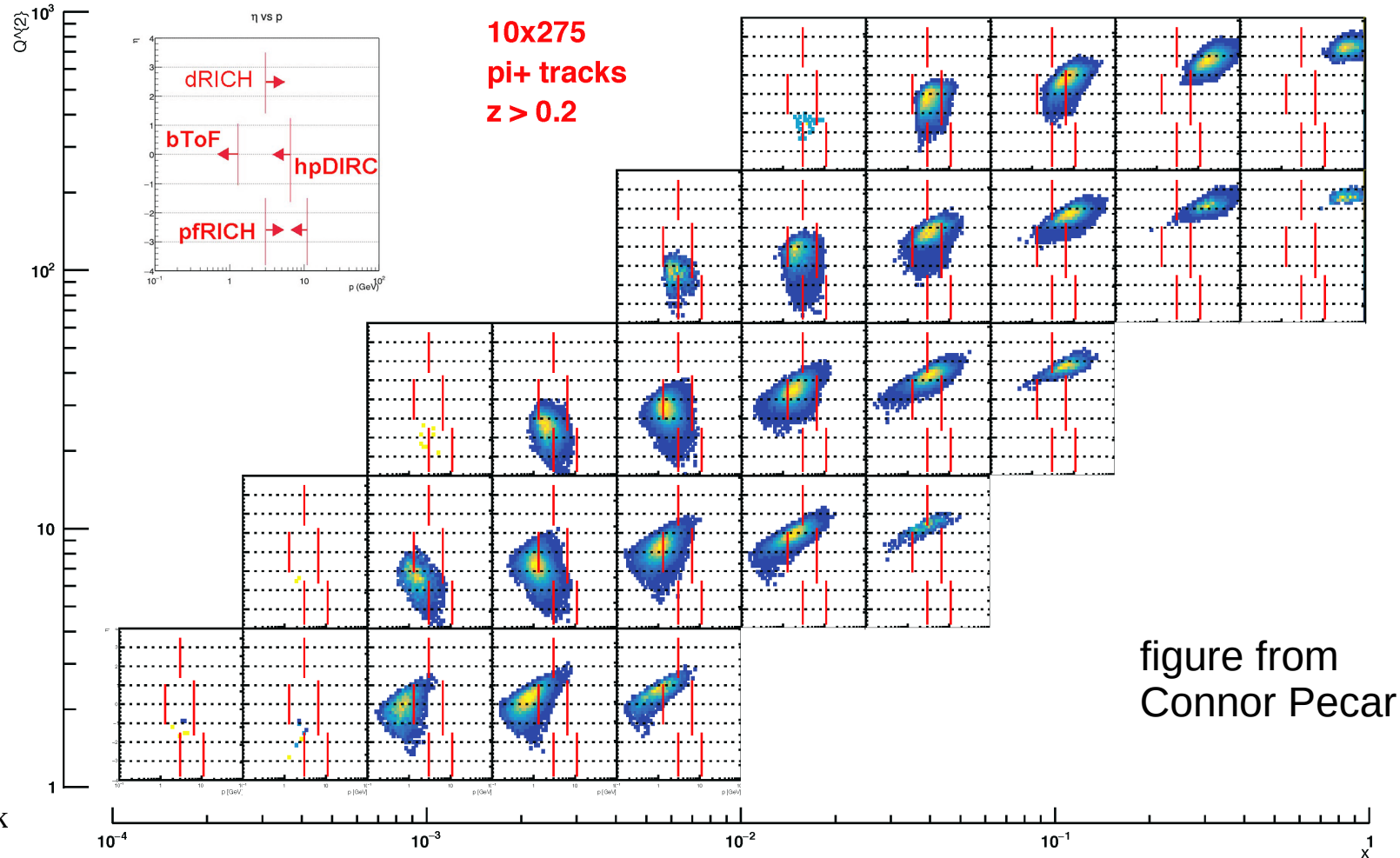
● Analysis Macro

- Choose AnalysisDelphes, AnalysisDD4hep, AnalysisFun4all
- Configure
 - Reconstruction Method
 - Final states (single hadron, jets, ...)
 - Cuts
 - Binning Schemes (construct Adage)
- Call Execute()

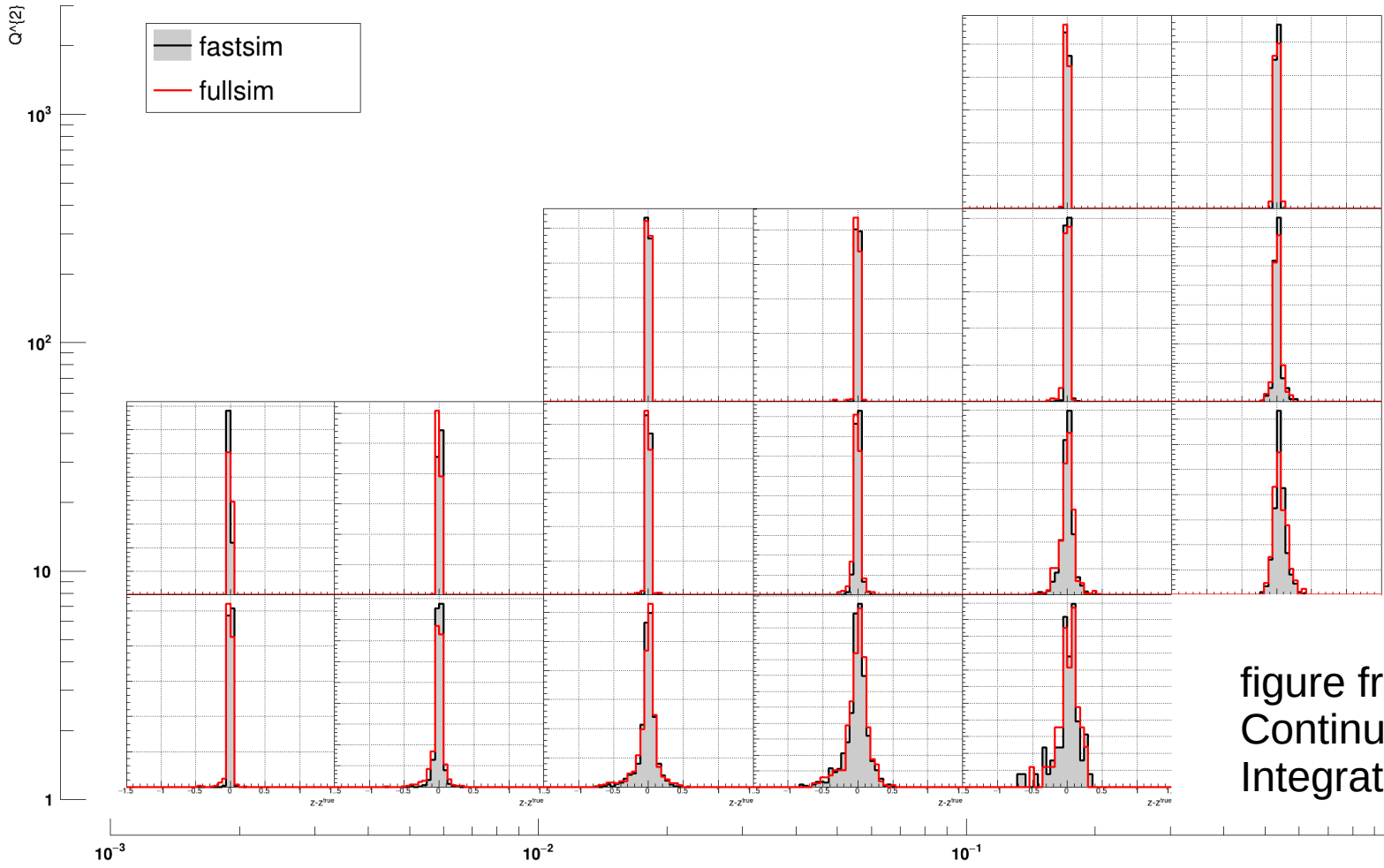
● Post-Processing Macro (if using Adage)

- 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()

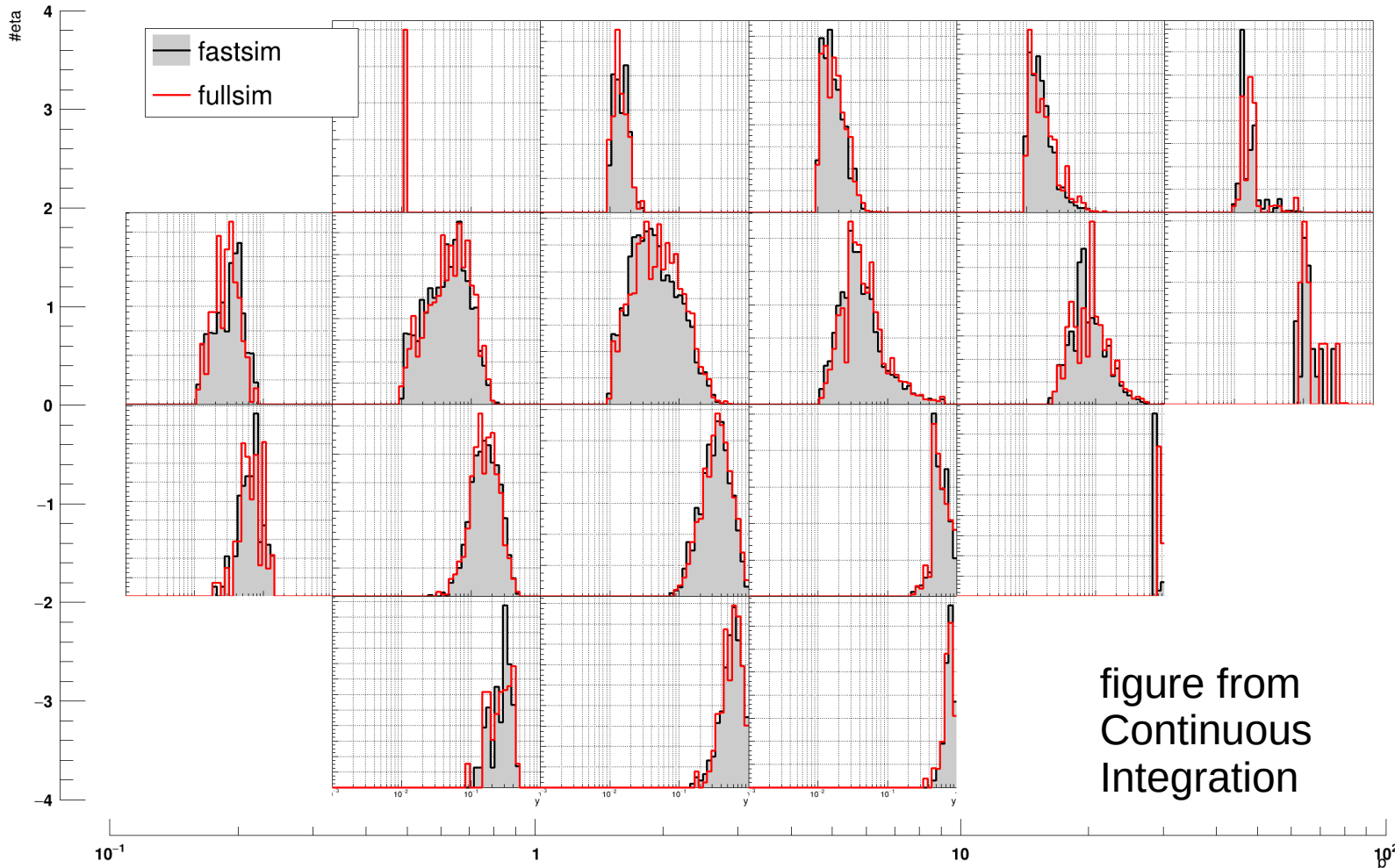
Example coverage plot: η vs. p in (x, Q^2) bins, with PID limits



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



Example benchmark plot: y , from fast and full simulations, in pion (p, η) bins

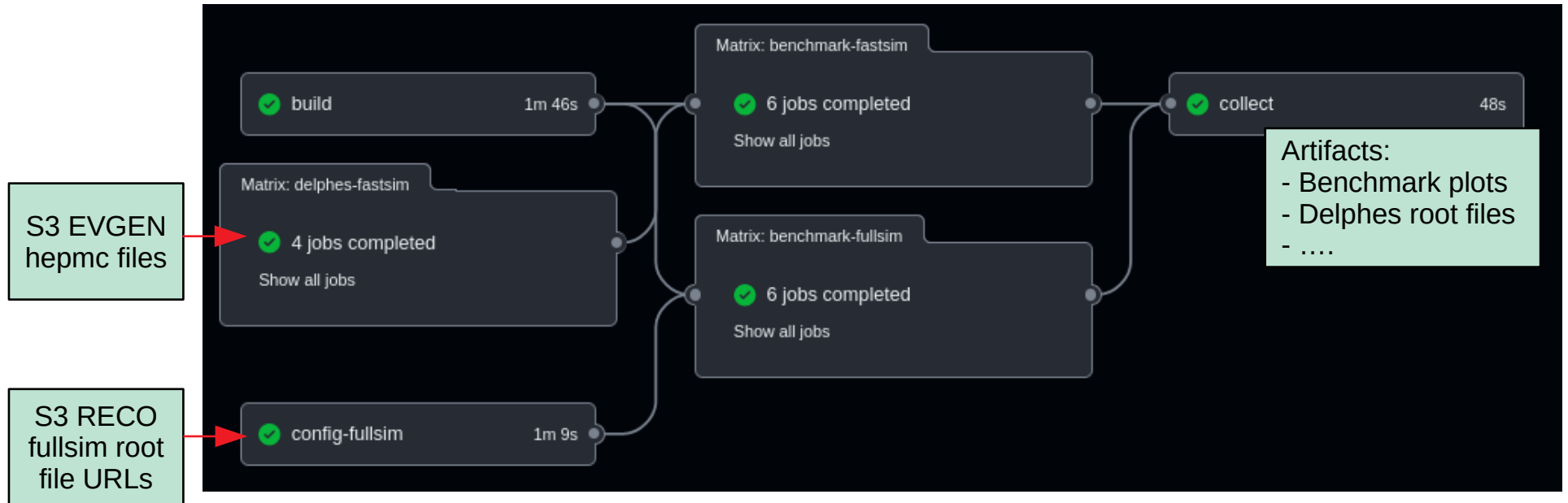


Continuous Integration (CI)

<https://github.com/c-dilks/sidis-eic/actions>

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)



Getting Started

- Setup (see README.md)
 - Use the Singularity or Docker image: based from [eicweb/jug_xl](https://github.com/eicweb/jug_xl)
 - Includes Delphes, in addition to [jug_xl](https://github.com/eicweb/jug_xl) software
 - Alternatively, local install (+dependencies)
- Follow README.md for documentation
- Follow Tutorials for example macros (tutorial/README.md)

Contributions are Welcome

- Fork SIDIS-EIC
- New branch → write code → pull request
- Github Workflow Tutorial:
<https://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project>