Reconstruction Algorithms for ePIC

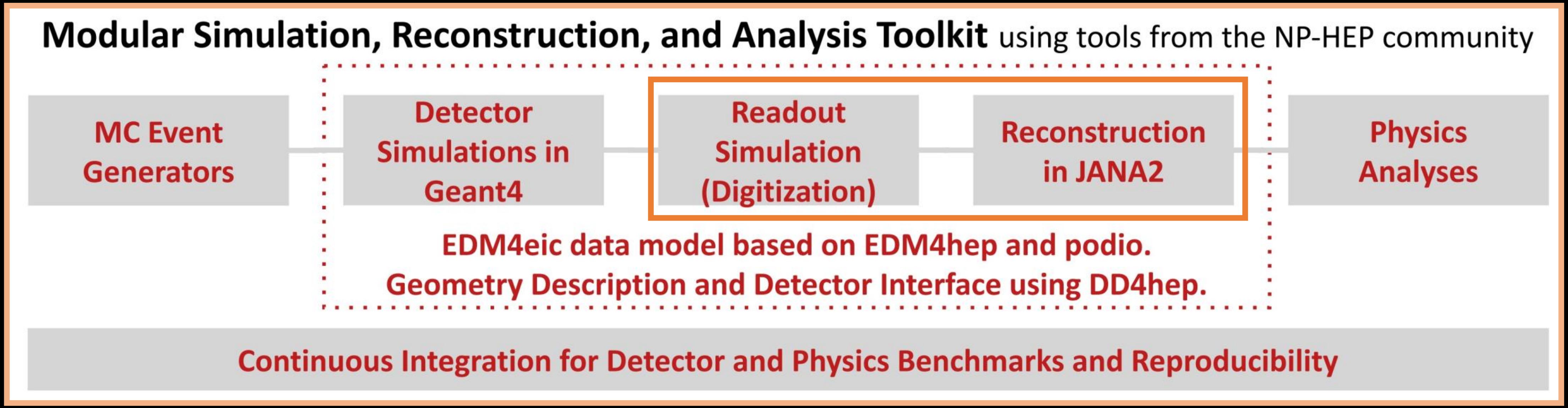
Derek Anderson (ISU)

For the ePIC Collaboration

With material from Sylvester, Markus, and others

Reconstruction | software overview

Image by Markus Diefenthaler



- **So far:** we've discussed
  - An overview of the software effort (M. Diefenthaler)
  - Event generators & detector simulation (K. Kauder)
  - Simulation production strategy (T. Britton)

- **Now:** let's discuss reconstruction (and digitization)
  - ☞ **Handled by EICrecon**

Reconstruction | reconstruction principles & design

3 We will leverage heterogeneous computing:

- We will enable distributed workflows on the computing resources of the worldwide EIC community, leveraging not only HTC but also HPC systems.
- EIC software should be able to run on as many systems as possible, while supporting specific system characteristics, e.g., accelerators such as GPUs, where beneficial.
- We will have a modular software design with structures robust against changes in the computing environment so that changes in underlying code can be handled without an entire overhaul of the structure.

4 We will aim for user-centered design:

- We will enable scientists of all levels worldwide to actively participate in the science program of the EIC, keeping the barriers low for smaller teams.
- EIC software will run on the systems used by the community, easily.
- We aim for a modular development paradigm for algorithms and tools without the need for users to interface with the entire software environment.

6 We will have reproducible software:

- Data and analysis preservation will be an integral part of EIC software and the workflows of the community.
- We aim for fully reproducible analyses that are based on reusable software and are amenable to adjustments and new interpretations.

8 We will provide a production-ready software stack throughout the development:

- We will not separate software development from software use and support.
- We are committed to providing a software stack for EIC science that continuously evolves and can be used to achieve all EIC milestones.
- We will deploy metrics to evaluate and improve the quality of our software.
- We aim to continuously evaluate, adapt/develop, validate, and integrate new software, workflow, and computing practices.

- **Above:** some of our software principles
  - Will emphasize a few in context of reconstruction
- **In particular:**
  - Modularity
  - Accessibility
  - Reproducibility
  - Mutuability

Reconstruction | reconstruction principles & design

⇒ How do we achieve those 4 principles?
- ☞ **Clear separation of components!**
- – Minimize friction by reducing scope of each component
  - › Meet people where they are
  - › Easy onboarding
  - › Etc.

- ○ **Right:** design goal of EICrecon
  - – 3 major components
    - › Framework
    - › Services (Resources)
    - › **Algorithms**
- ○ **In particular:** let's discuss algorithms…

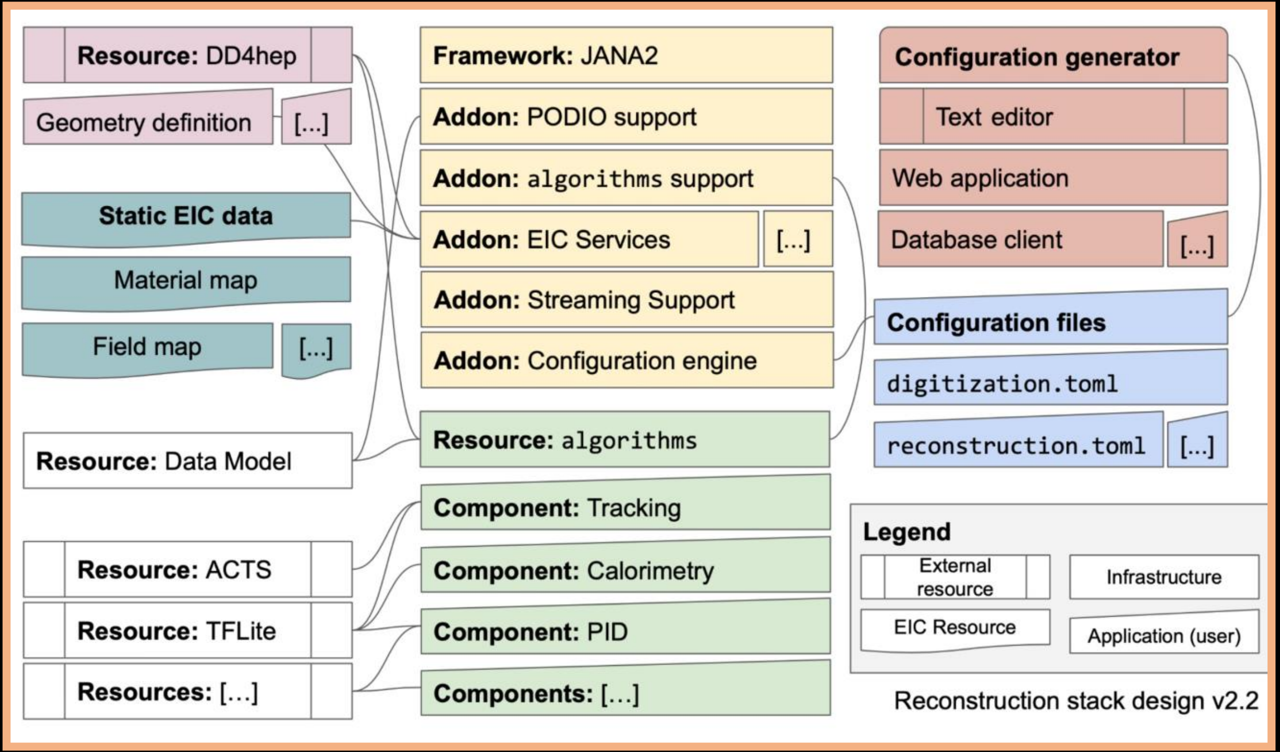


Image by Sylvester Joosten

Algorithms

- Multiple different detectors may use the same algorithm but with different parameters
  - Same for physics objects (e.g. jets)

∴ **Algorithms in EICrecon should be generic, abstract**
  ⇒ Then realized in multiple concrete detector (etc.) instances
  ☞ Nathan and co. preparing "best practices" guide

- **Goals:** EICrecon algorithms should be
  - "Framework agnostic"
  - Shareable across experiments and communities
  - Capable of supporting multiple workflows
  - Devoid of duplicate definitions

⇒ **Most importantly:** algorithms will make EICrecon **accessible**

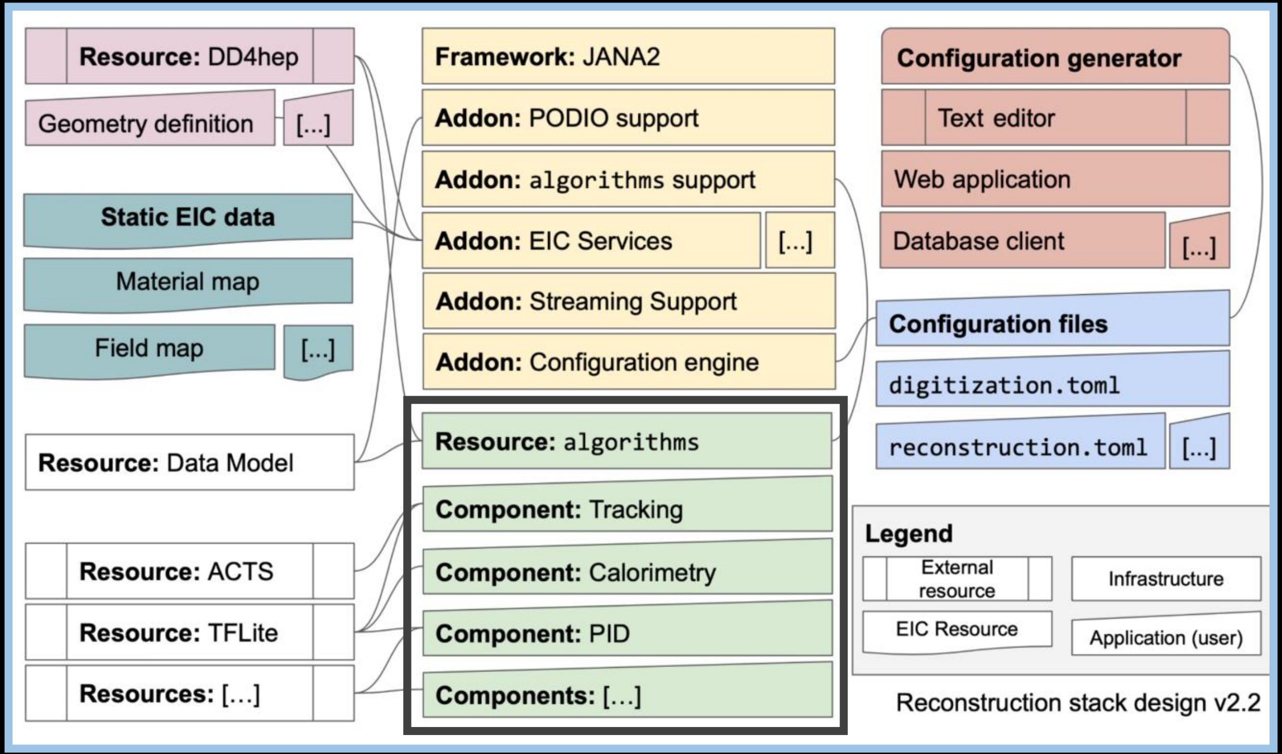


Image by Sylvester Joosten

Algorithms | example 1: jet reconstruction

- **One example:** jet reconstruction!
  - Clusters a provided list of 4-momenta into jets
    - ☞ via `execute()`
  - Algorithm realized in **2 instances:**
    - › Reconstructed Jets
    - › Generated Jets

- **Note:** algorithm parameters will be user-configurable in near future…

```
namespace eicrecon {

  // jet reco
  class JetReconstruction {

    public:

      void init(std::shared_ptr<spdlog::logger> logger);

      edm4eic::ReconstructedParticleCollection* execute(
        const std::vector<const edm4hep::LorentzVectorE*> momenta
      );

      // input parameters
      double m_minCstPt  = 0.2  * dd4hep::GeV;  // minimum pT of objects fed to cluster sequence
      double m_maxCstPt  = 100. * dd4hep::GeV;  // maximum pT of objects fed to clsuter sequence

      // jet parameters
      float                       m_rJet         = 1.0;                                  // jet resolution  parameter
      double                      m_minJetPt     = 1.0 * dd4hep::GeV;                    // minimum jet pT
      fastjet::JetAlgorithm       m_jetAlgo      = fastjet::antikt_algorithm;            // jet finding algorithm
      fastjet::RecombinationScheme  m_recombScheme = fastjet::RecombinationScheme::E_scheme;  // particle recombination scheme

      // area parameters
      double             m_ghostMaxRap    = 3.5;                          // maximum rapidity of ghosts
      int                m_numGhostRepeat = 1;                            // number of times a ghost is reused per grid site
      double             m_ghostArea      = 0.001;                        // area per ghost
      fastjet::AreaType  m_areaType       = fastjet::AreaType::active_area;  // type of area calculated

    private:

      std::shared_ptr<spdlog::logger> m_log;

  };  // end JetReconstruction definition

}  // end eicrecon namespace
```

Algorithm Definition

Algorithms | example 1: jet reconstruction

```
namespace eicrecon {

    class ReconstructedJets_factory :
            public JChainFactoryT<edm4eic::ReconstructedParticle>,
            public SpdlogMixin<ReconstructedJets_factory> {

    public:
        explicit ReconstructedJets_factory(std::vector<std::string> default_input_tags):
            JChainFactoryT<edm4eic::ReconstructedParticle>(std::move(default_input_tags)) {
        }

        /** One time initialization **/
        void Init() override;

        /** On run change preparations **/
        void ChangeRun(const std::shared_ptr<const JEvent> &event) override;

        /** Event by event processing **/
        void Process(const std::shared_ptr<const JEvent> &event) override;

    protected:
        JetReconstruction m_jet_algo;

    };

} // eicrecon
```

```
    void ReconstructedJets_factory::Process(const std::shared_ptr<const JEvent> &event) {
        auto rc_particles = event->Get<edm4eic::ReconstructedParticle>("ReconstructedParticles");

        std::vector<const edm4hep::LorentzVectorE*> momenta;
        for (const auto& p : rc_particles) {
          // TODO: Need to exclude the scattered electron
          const auto& mom    = p -> getMomentum();
          const auto& energy = p -> getEnergy();
          momenta.push_back(new edm4hep::LorentzVectorE(mom.x, mom.y, mom.z, energy));
        }

        auto jets = m_jet_algo.execute(momenta);
        for (const auto &mom : momenta) {
          delete mom;
        }
        Set(jets);
    }
```

Algorithm Realization

- Each factory **calls** algorithm during process
  1) List of 4-vectors pulled from relevant input
  2) Passed to algorithm for clustering
  3) Algorithm returns jets, which are handed off to JANA

Algorithms | example 2: calorimeter clustering

```
class CalorimeterIslandCluster {

    // Insert any member variables here

public:
    CalorimeterIslandCluster() = default;
    virtual ~CalorimeterIslandCluster(){} // better to use smart pointer?
    virtual void AlgorithmInit(std::shared_ptr<spdlog::logger>& logger);
    virtual void AlgorithmChangeRun() ;
    virtual void AlgorithmProcess() ;

    //-------- Configuration Parameters ------------
    //instantiate new spdlog logger
    std::shared_ptr<spdlog::logger> m_log;

    std::string m_input_tag;

    // geometry service to get ids
    std::string m_geoSvcName; //{this, "geoServiceName", "GeoSvc"};
    std::string m_readout; //{this, "readoutClass", ""};
    std::string u_adjacencyMatrix; //{this, "adjacencyMatrix", ""};

    // neighbour checking distances
    double m_sectorDist;//{this, "sectorDist", 5.0 * dd4hep::cm};
    std::vector<double> u_localDistXY;//{this, "localDistXY", {}};
    std::vector<double> u_localDistXZ;//{this, "localDistXZ", {}};
    std::vector<double> u_localDistYZ;//{this, "localDistYZ", {}};
    std::vector<double> u_globalDistRPhi;//{this, "globalDistRPhi", {}};
    std::vector<double> u_globalDistEtaPhi;//{this, "globalDistEtaPhi", {}};
    std::vector<double> u_dimScaledLocalDistXY;//{this, "dimScaledLocalDistXY", {1.8, 1.8}};
    // neighbor checking function
    std::function<edm4hep::Vector2f(const CaloHit*, const CaloHit*)> hitsDist;
```

Algorithm Definition

- **Another example:** clustering calorimeter cells
  - Cluster calorimeter cells into continuous distributions of energy
    - ☞ via `AlgorithmProcess()`
  - Algorithm realized in **11 instances:**
    - › (iall of the calorimeters)

Algorithms | example 2: calorimeter clustering

```
class ProtoCluster_factory_HcalBarrelIslandProtoClusters : public JChainFactoryT<edm4eic::ProtoCluster>, CalorimeterIslandCluster {

public:
    //------------------------------------------
    // Constructor
    ProtoCluster_factory_HcalBarrelIslandProtoClusters(std::vector<std::string> default_input_tags)
    : JChainFactoryT<edm4eic::ProtoCluster>(std::move(default_input_tags)) {
        m_log = japp->GetService<Log_service>()->logger(GetTag());
    }

    //------------------------------------------
    // Init
    void Init() override{
        InitDataTags(GetPluginName() + ":" + GetTag());

        auto app = GetApplication();
```

```
    //------------------------------------------
    // Process
    void Process(const std::shared_ptr<const JEvent> &event) override{
        // Prefill inputs
        hits = event->Get<edm4eic::CalorimeterHit>(GetInputTags()[0]);

        // Call Process for generic algorithm
        AlgorithmProcess();

        // Hand owner of algorithm objects over to JANA
        Set(protoClusters);
        protoClusters.clear(); // not really needed, but better to not leave dangling pointers around
    }
};
```

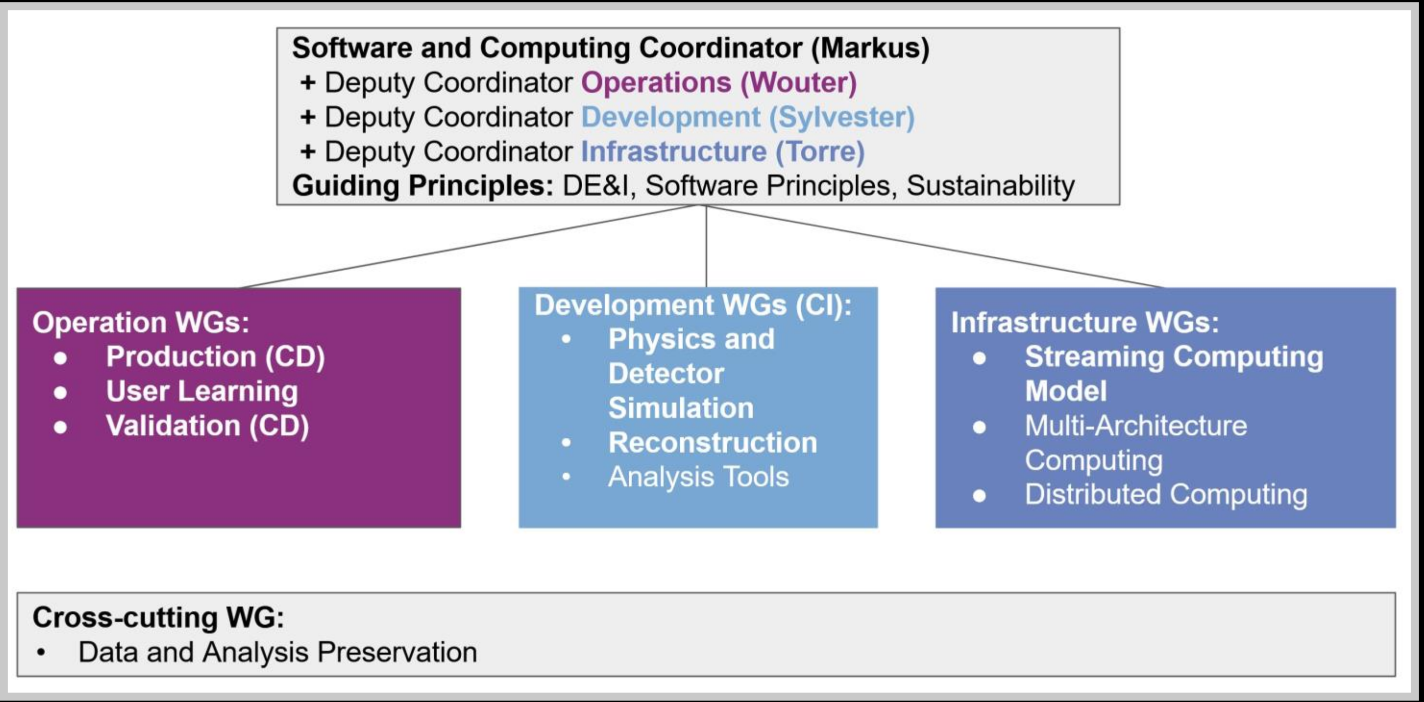
Algorithm Realization

- Each factory **inherits** algorithm, and then runs algorithm during process
  1) Input calorimeter hits grabbed
  2) Algorithm is run to produce clusters
  3) Clusters handed off to JANA

The Reco Working Group

Conveners: Derek Anderson and Shujie Li

- **Charge**: Development of a holistic and modular reconstruction for the integrated ePIC detector.
- **Priorities for 2023**:
  - Enforce modularity for clear separation between development of reconstruction algorithms and development of framework and its services.
  - Embrace algorithmic development that utilizes the holistic information from detector components or the entire detector.
  - Integrate far-forward and far-backward detectors in reconstruction.
  - Implement a web-based event display.



Slide by Markus Diefenthaler

The Reco Working Group | activity squadrons

- Organized 4 “squadrons” organized around priorities identified by Physics Analysis & C/S Coordinators
- See the following talks for more details:
  - Analysis Coordinator Report (Fri., 3:30 pm)
  - Electron-Finding and Particle Flow (Fri., 4:50 pm)

Electron Finder:
- **Coordinator:** Daniel Brandenburg (brandenberg.89@osu.edu)
- **Charge:** develop an efficient and accurate algorithm for identifying electrons and identifying scattered e^- in DIS.

Vertexing/Tracking
- **Coordinator:** Shujie Li (shujieli@lbl.gov)
- **Charge:** enhance vertexing capabilities and PID techniques to study heavy flavor physics.

Particle Flow
- **Coordinator:** Derek Anderson (dmawxc@iastate.edu)
- **Charge:** improve jet reconstruction using particle flow information.

Low-Q^2 Tagger
- **Coordinator:** Simon Gardner (simon.gardner@glasgow.ac.uk)
- **Charge:** integrate low-Q2 tagger into reco. framework for precise measurements of photoproduction and vector mesons.

The Reco Working Group | how to get involved

- **Help wanted!**
  - Labor-power is currently very limited…
  - Helping hands are **always** appreciated!
- **Don't hesitate to reach out!**
  - Derek Anderson (dmawx@iastate.edu)
  - Shuji Li (shujieli@lbl.gov)
  - Sylvester Joosten (sjoosten@anl.gov)
  - Markus Diefenthaler (mdiefent@jlab.org)

Some available tasks:

- Validation of existing cluster splitting
- Validation of MC-cluster associations
- Implementation of PF algorithm + factories
- Enable user-configured jet-finding parameters
- Enable proper PODIO jet-cst. Associations

(And many more…)

Key:

Purple = particle flow
Green = jets

Thanks!