TOWARDS A FRAMEWORK-INDEPENDENT ALGORITHM LIBRARY FOR EIC AND BEYOND

algorithms

```
using ClusteringAlgorithm = Algorithm<
    Input<edm4eic::ProtoClusterCollection,
          std::optional<edm4hep::SimCalorimeterHitCollection>>,
    Output<edm4eic::ClusterCollection,
           std::optional<edm4eic::MCRecoClusterParticleAssociation
```

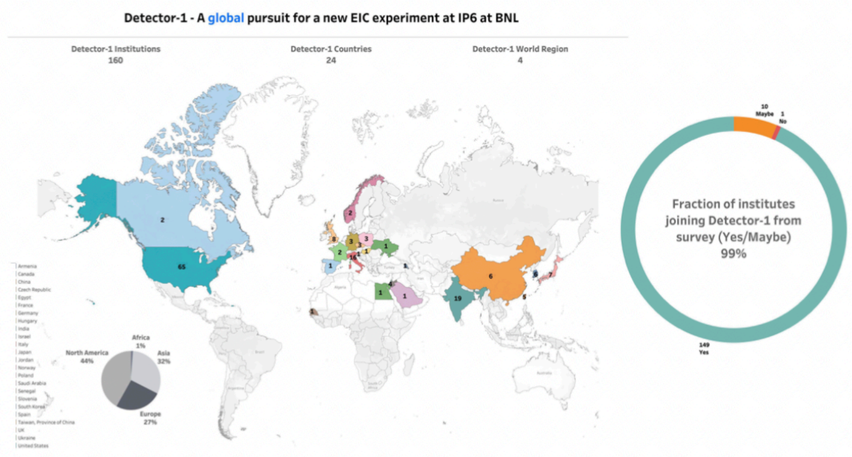
SYLVESTER JOOSTEN
sjoosten@anl.gov
on behalf of the ePIC collaboration

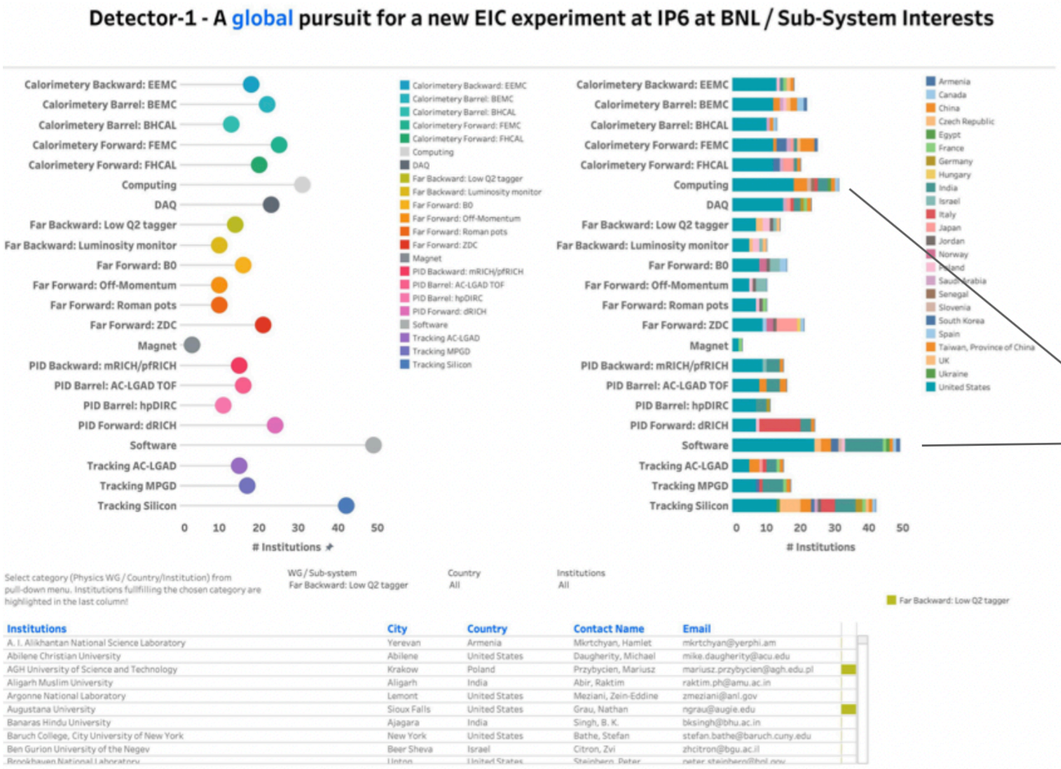
This work is supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics, under contract DE-AC02-06CH11357.

CHEP 2023
Norfolk, VA - May 9, 2023

THE EPIC EXPERIMENT AT THE EIC

We are a large collaboration
(160 institutions)





49 groups indicate commitment to work on software for EIC, of which 22 also want to contribute to computing

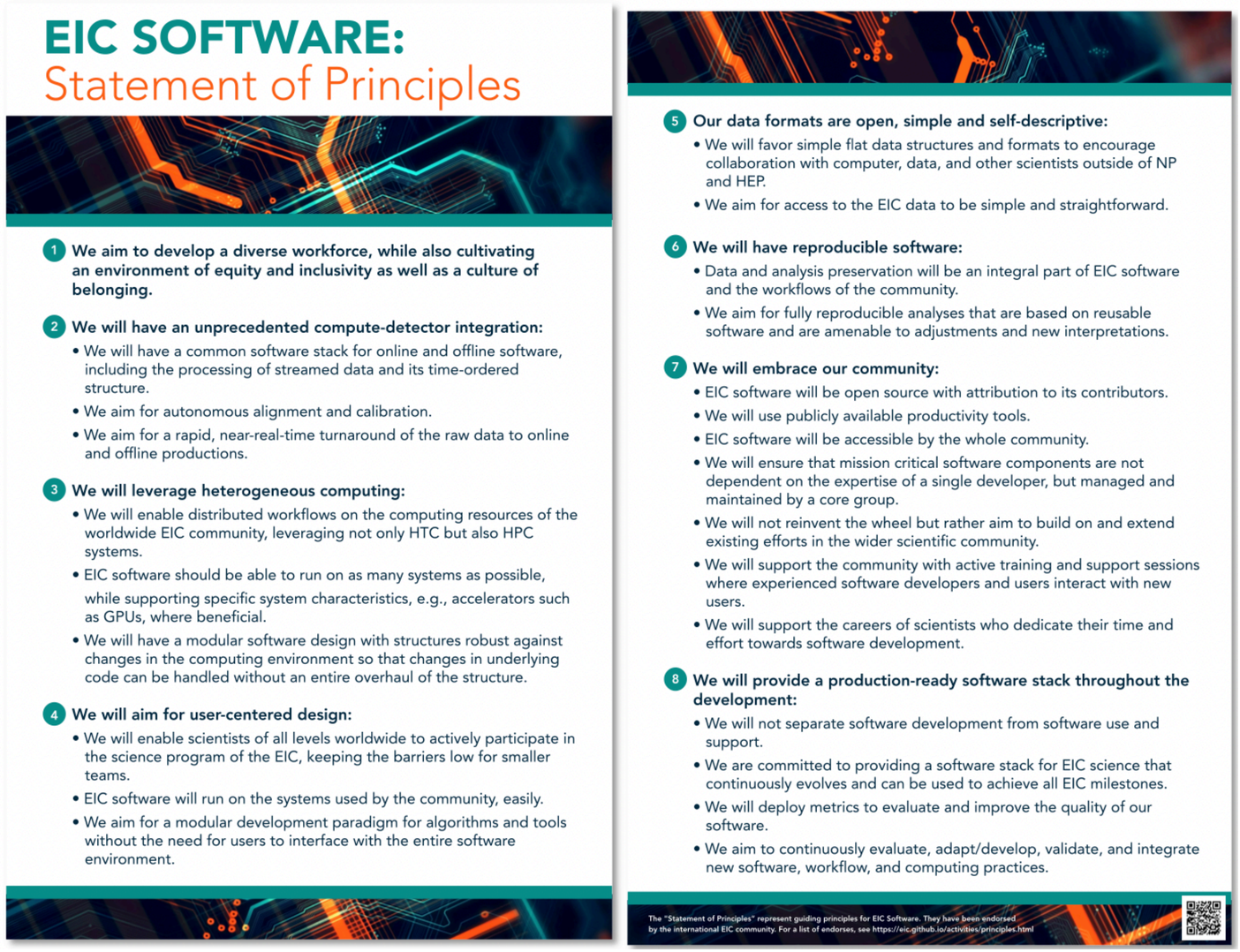
More commitments to software than *any* other topic!

See M. Diefenthaler's talk from yesterday (Track 5)

https://eic.github.io/activities/principles.html

EIC SOFTWARE STATEMENT OF PRINCIPLES

- As part of the "Lessons Learned" process, the entire EIC community came together to create a community document to define our aspirations for software and computing for the EIC
- Meant to form a sound foundation to design our software stack
- This document was spread to the entire EIC community through several rounds of open suggestions and endorsement to ensure this is truly a community document
  - Endorsed by a large group representing the international EIC community.
  - 100% of responses were positive!



EIC SOFTWARE:
Statement of Principles

1 We aim to develop a diverse workforce, while also cultivating an environment of equity and inclusivity as well as a culture of belonging.

2 We will have an unprecedented compute-detector integration:
- We will have a common software stack for online and offline software, including the processing of streamed data and its time-ordered structure.
- We aim for autonomous alignment and calibration.
- We aim for a rapid, near-real-time turnaround of the raw data to online and offline productions.

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.

5 Our data formats are open, simple and self-descriptive:
- We will favor simple flat data structures and formats to encourage collaboration with computer, data, and other scientists outside of NP and HEP.
- We aim for access to the EIC data to be simple and straightforward.

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.

7 We will embrace our community:
- EIC software will be open source with attribution to its contributors.
- We will use publicly available productivity tools.
- EIC software will be accessible by the whole community.
- We will ensure that mission critical software components are not dependent on the expertise of a single developer, but managed and maintained by a core group.
- We will not reinvent the wheel but rather aim to build on and extend existing efforts in the wider scientific community.
- We will support the community with active training and support sessions where experienced software developers and users interact with new users.
- We will support the careers of scientists who dedicate their time and effort towards software development.

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.

The "Statement of Principles" represent guiding principles for EIC Software. They have been endorsed by the international EIC community. For a list of endorses, see https://eic.github.io/activities/principles.html

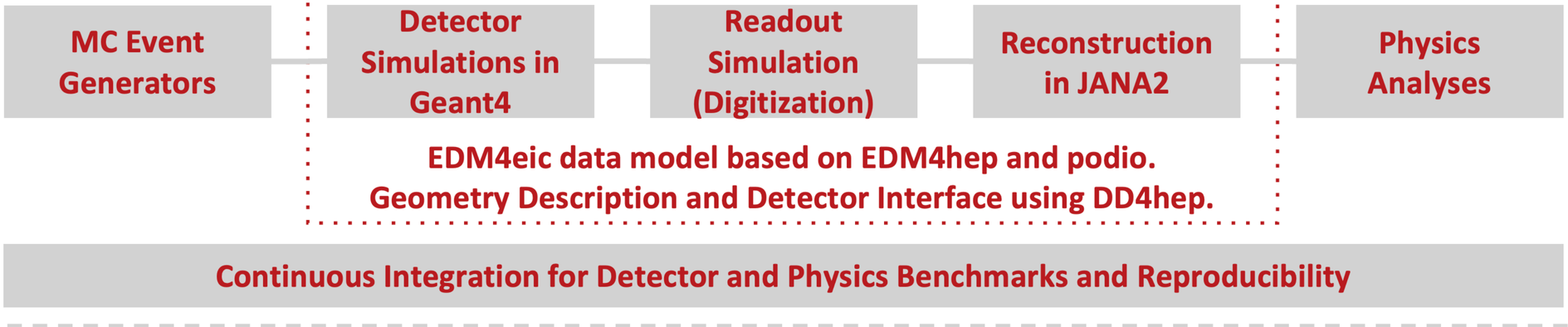
See D. Lawrence's talk from yesterday (Track 3)

Software for the Realization of the ePIC Experiment

Our software design is based on **lessons learned in the worldwide NP and HEP community** and a **decision-making process** involving the whole community. We are guided by our Software Statement of Principles:
https://eic.github.io/activities/principles.html

We will continue to work with the worldwide NP and HEP community.

Modular Simulation, Reconstruction, and Analysis Toolkit using tools from the NP-HEP community



We are providing a production-ready software stack throughout the development:

- **Milestone**: Software enabled first large-scale simulation campaign for ePIC.

We have a good foundation to meet the near-term and long-term software needs for ePIC.

slide by Markus Diefenthaler

CHANGING FRAMEWORKS IS EXPENSIVE!

The case for framework-agnostic algorithms

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:

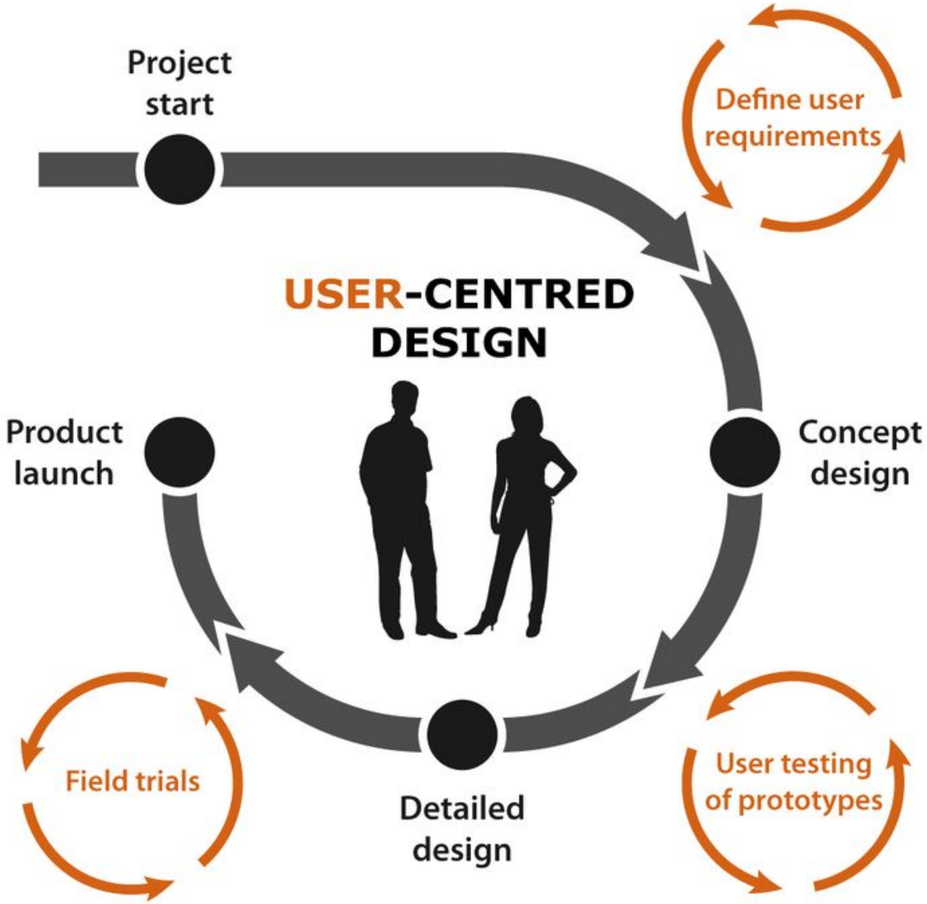
- We will enable scientists of all levels worldwide to actively participate in the science program of the EIC, keeping the barriers low for smaller teams.
- EIC software will run on the systems used by the community, easily.
- We aim for a modular development paradigm for algorithms and tools without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

- Need more rigorous separation of different domains:
  - ➡ Framework
  - ➡ Algorithms
  - ➡ Configuration
  - ➡ Resources
  - ➡ User workflow
  - ➡ …
- This will enhance user experience, improve maintainability, increase flexibility against future changes, reduce scope of developer responsibility (everyone is the ruler of their own realm)

FROM A USER'S PERSPECTIVE

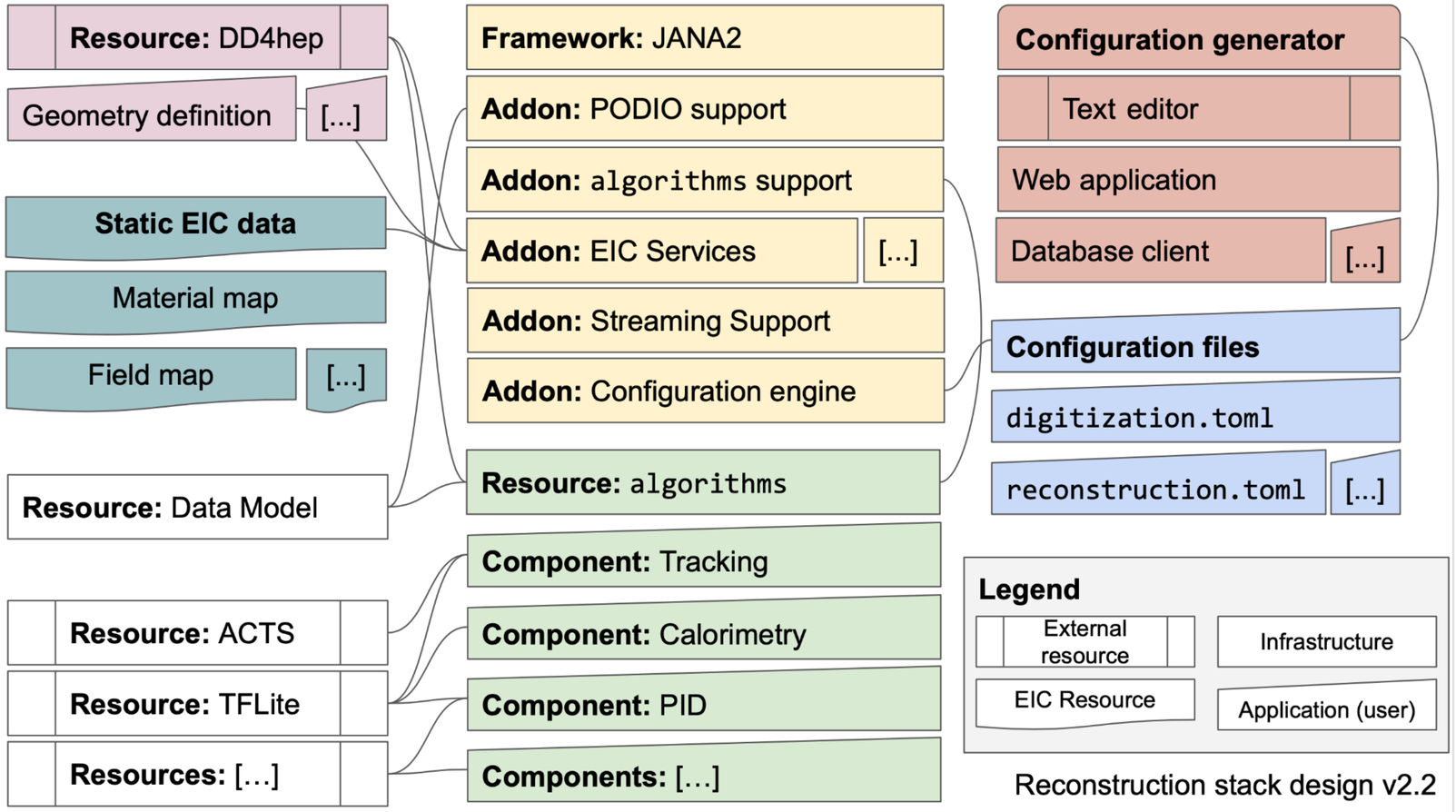
Close tie-in of algorithms within their frameworks creates friction

- Need to support workflows actually needed by the Users
- Create, test, and run a new reconstruction algorithm with minimal work, support new stand-alone plugins with minimal friction
- Evaluate changes in geometry by changing only the geometry definition and relevant configuration file (no need to change/recompile everything) - again, minimize friction
- Get reproducible (and easily altered) reconstruction configurations without needing to do any additional work (zero-friction reproducibility)
- Provide domains of responsibility where Users of all experience levels can make meaningful contributions
- Distinct domains of responsibility also make clear who to talk to, no more single persons supporting everything at once.



EVOLVING OF THE EPIC RECONSTRUCTION STACK DESIGN

- Strictly modular approach reduces scope of each component
- Easier to onboard new users in any singular piece of the stack
- Every user can find their place based on experience and needs
- Better maintainability and more resilient against changing software needs
- Baked-in reproducibility by enforcing configuration files in every workflow



Reconstruction stack design v2.2

DESIGN GOALS AND CHALLENGES

Towards a first prototype for `algorithms`

DESIGN GOALS

- Enable algorithm sharing across experiments and even communities
- Framework agnostic algorithms
- Main dependencies: EDM4hep/ EDM4eic and DD4hep
- Showcase independence through both Gaudi and JANA2 integration
- No duplication of definitions
- “Zero-line” holistic framework integration

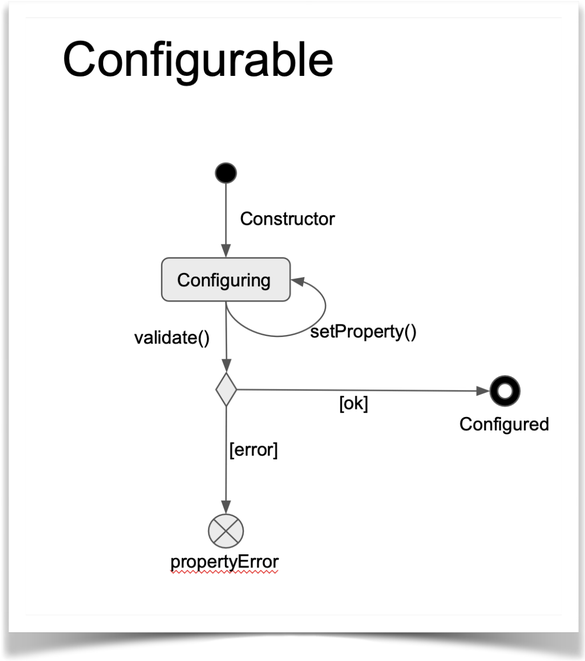
CHALLENGES

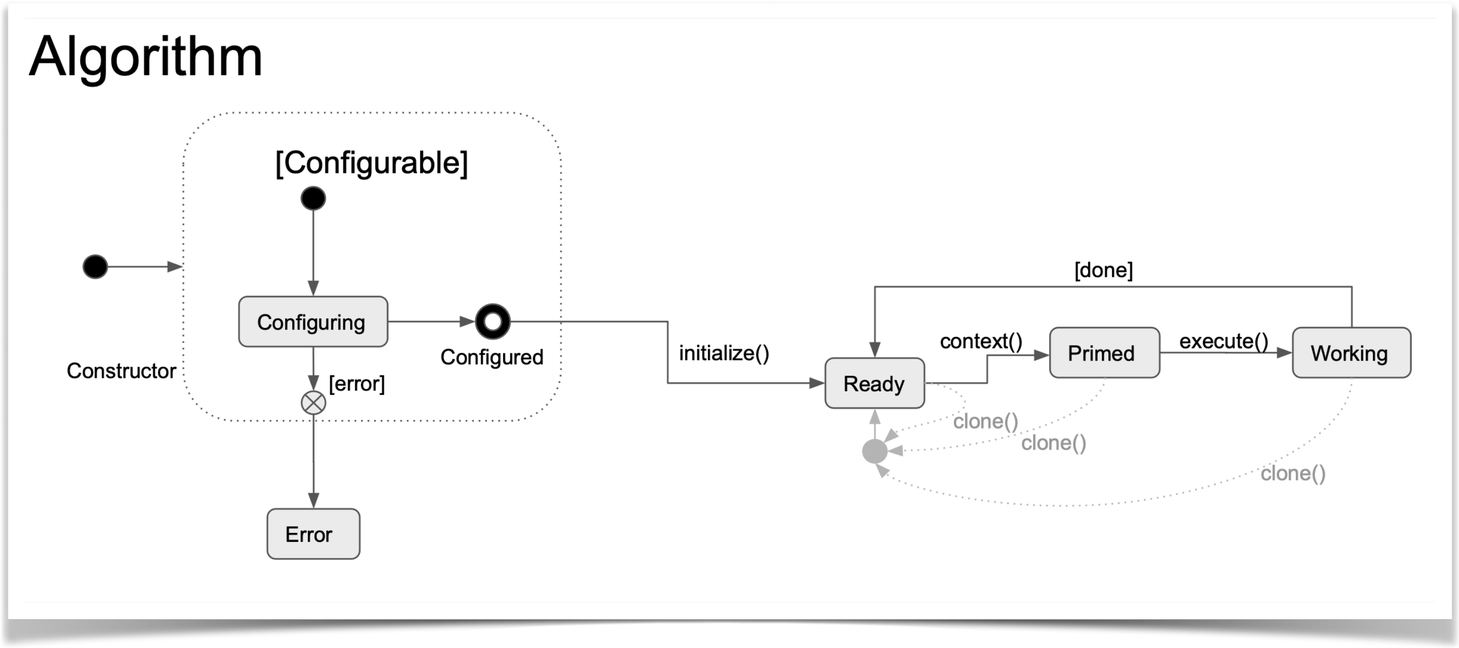
- Data store interactions
- Properties
- Service integration
- Context
- “Zero-line” generic framework integration non-trivial
- Automatic testing in a no-framework context

CURRENTLY IN API DESIGN PHASE

Framework integration requires predictable API

Preliminary state machine design; rigorous state validation desired.





EXAMPLE SERVICE INTEGRATION

Towards a first prototype for `algorithms`

```cpp
// Thread-safe lazy-evaluated minimal service system
// CRTP base class to add the instance method
// This could have been part of DEFINE_SERVICE macro, but I think it is better
// to keep the macro magic to a minimum to maximize transparency
template <class SvcType> class Service : public PropertyMixin, public NameMixin {
public:
  static SvcType& instance() {
    // This is guaranteed to be thread-safe from C++11 onwards.
    static SvcType svc;
    return svc;
  }
  // constructor for the service base class registers the service, except
  // for the ServiceSvc which is its own thing (avoid circularity)
  Service(std::string_view name) : NameMixin{name} { ServiceSvc::instance().add(name, this); }
};

} // namespace algorithms
```

```cpp
// Note: the log action is responsible for dealing with concurrent calls
//       the default LogAction is a thread-safe example
class LogSvc : public Service<LogSvc> {
public:
  using LogAction = std::function<void(LogLevel, std::string_view, std::string_view)>;
  void defaultLevel(const LogLevel l) { m_level.set(l); }
  LogLevel defaultLevel() const { return m_level; }
  void action(LogAction a) { m_action = a; }
  void report(const LogLevel l, std::string_view caller, std::string_view msg) const {
    m_action(l, caller, msg);
  }

private:
  Property<LogLevel> m_level{this, "defaultLevel", LogLevel::kInfo};
  LogAction m_action = [](const LogLevel l, std::string_view caller, std::string_view msg) {
    static std::mutex m;
    std::lock_guard<std::mutex> lock(m);
    fmt::print("{} [{}] {}\n", logLevelName(l), caller, msg);
  };
  ALGORITHMS_DEFINE_SERVICE(LogSvc)
};
```

- Services as lazy-evaluated singletons
- Support standalone minimal interface
  - Interface has usable defaults for standalone operation
  - Standalone defaults are meant to be overridden by the framework
- Prototype currently implements `LogSvc` and `GeoSvc`
- Special `ServiceSvc` provides framework with all required services, so it can handle the bindings

EXAMPLE DATA STORE INTERACTIONS

Towards a first prototype for `algorithms`

```
using ClusteringAlgorithm = Algorithm<
    Input<edm4eic::ProtoClusterCollection,
          std::optional<edm4hep::SimCalorimeterHitCollection>>,
    Output<edm4eic::ClusterCollection,
           std::optional<edm4eic::MCRecoClusterParticleAssociationCollection>>>;

class ClusterRecoCoG : public ClusteringAlgorithm {
public:
  using Input      = ClusteringAlgorithm::Input;
  using Output     = ClusteringAlgorithm::Output;
  using WeightFunc = std::function<double(double, double, double)>;

  ClusterRecoCoG(std::string_view name)
      : ClusteringAlgorithm{name,
                            {"inputProtoClusterCollection", "mcHits"},
                            {"outputClusterCollection", "outputAssociations"}} {}
```

```
void ClusterRecoCoG::process(const ClusterRecoCoG::Input& input,
                             const ClusterRecoCoG::Output& output) {
  const auto [proto, opt_simhits] = input;
  auto [clusters, opt_assoc]      = output;

  for (const auto& pcl : *proto) {
    auto cl = reconstruct(pcl);

    if (aboveDebugThreshold()) {
      debug() << cl.getNhits() << " hits: " << cl.getEnergy() / dd4hep::GeV <<
              << cl.getPosition().x / dd4hep::mm << ", " << cl.getPosition().y
              << cl.getPosition().z / dd4hep::mm << ")" << endmsg;
    }
    clusters->push_back(cl);
```

- Needed to choose between (1) providing `algorithms` with a framework allocator, (2) going with a purely functional approach, or (3) passing pointers to already existing objects
- Chose (3) (tuple of pointers) as it significantly simplifies interactions with the frameworks
- Algorithm definition takes an `Input` and an `Output` type to define the signature of the `::process` function
- Special cases for `std::vector<T>` (to handle multiple objects of the same type) and `std::optional<T>` (to handle optional data, e.g. MC truth info in reconstruction algorithms)

OUTLOOK

Towards a first prototype for `algorithms`

- ✓ Library infrastructure code ready
- 🚧 Framework-agnostic API evaluation
  - 🚧 JANA2 bindings in the design phase
- ✖ Algorithm migration within the ePIC stack will follow a successful prototyping phase
  - Seeking “radical modularity" to minimize user friction
- 🚧 Explore collaboration with Key4hep
  - 🚧 Gaudi bindings being tested