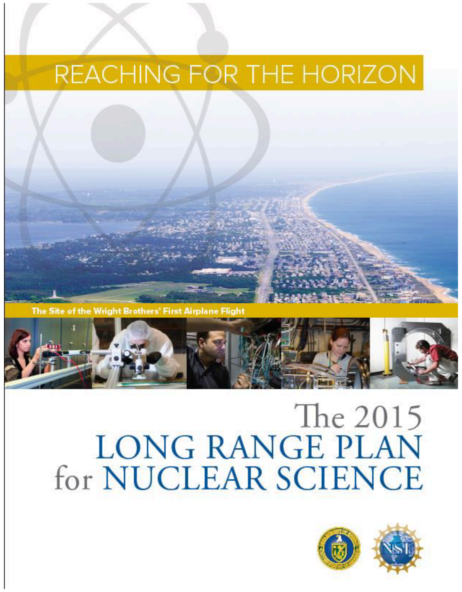
EIC software

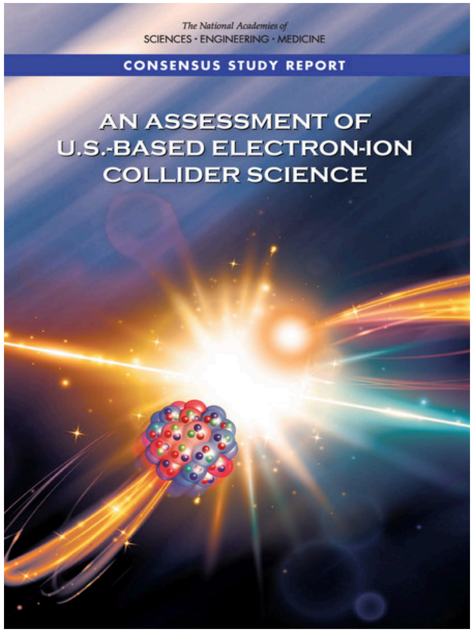
Alexander Kiselev

NPPS Group Meeting   July,3 2019

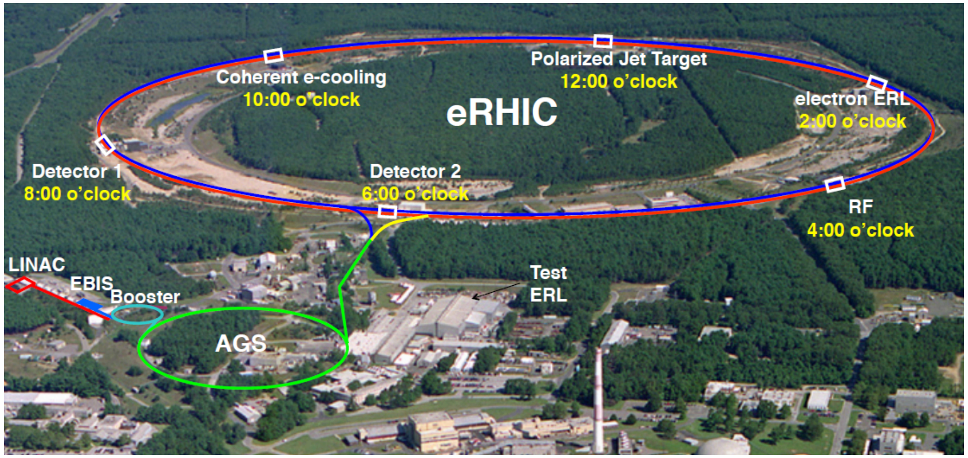
EIC timelines

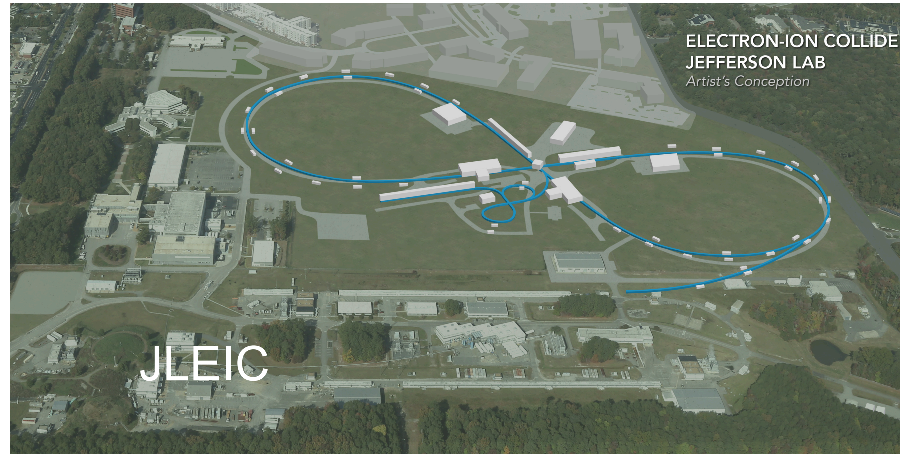
- **2015 NSAC (NP) Long-Range Plan:**
  - “We recommend a high-energy high-luminosity polarized EIC as the highest priority for new facility construction.”
- **2018 NAS review:**
  - “The committee finds that the science that can be addressed by an EIC is compelling, fundamental and timely.”
- **President’s budget request for FY2020:**
  - Critical Decision-0, Approve Mission Need, is planned for FY2019

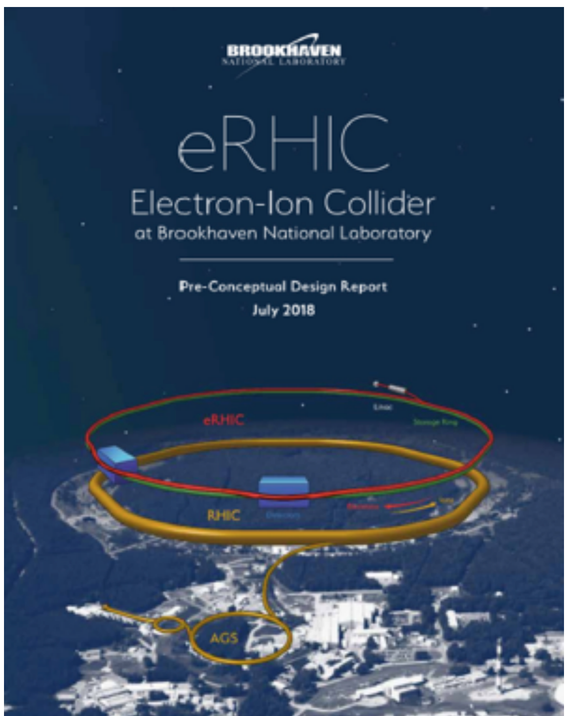




Machine requirements & EIC realization



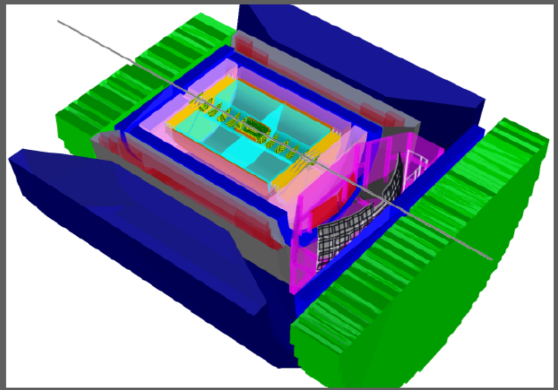




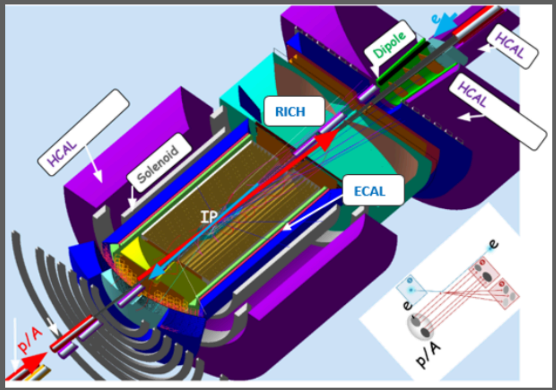
- Wide kinematic range: $\sqrt{s}$ from ~20 to 100 GeV, upgradable to 140 GeV
- Luminosity ~$10^{33\text{-}34}$ $cm^{-2}s^{-1}$
- Polarized protons, electrons and light ions
- Heavy ion beams up to U

EIC detector concepts

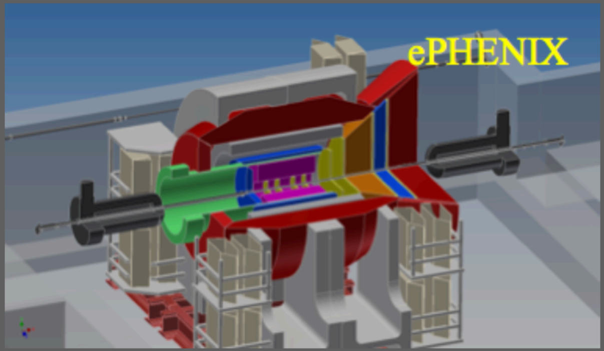
Brookhaven concept: BEAST



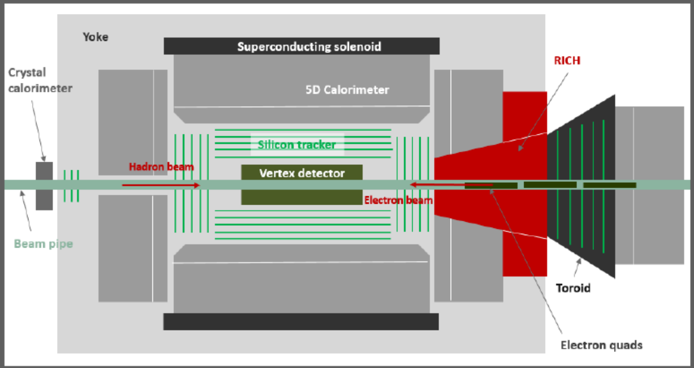
Jefferson lab concept: JLEIC



sPhenix → ePhenix



Argonne concept: TOPSiDE



-> software frameworks are strictly bound to the detector concepts

Contents of this talk

- Fast simulation tool: eic-smear
- Software frameworks
  - GEMC
  - fun4all
  - EicRoot
  - Argonne EIC software initiative
- PID consortium GEANT4 software (one slide)
- Omit all the other small custom pieces of EIC MC codes
- Near-term future trend(s)

eic-smear

by Tom Burton (BNL TF group)

Overview

- **C++** code, runs in **ROOT**
- Build with **configure/Make** or **CMake**
- `libeicsmear.so` to load in ROOT

MC generator ASCII output → **Tree code**: Build ROOT tree containing events → **Smearer**: Perform fast detector smearing

PYTHIA
DPMJet
PEPSI
Milou
gmc_trans
Rapgap
LEPTO
Djangoh

Large number of EIC Monte Carlo generators with standard ASCII format

Smearing

"**Smearer**" defines some element of performance + acceptance

- Built-in standard smearers provided with eic-smear
- Users can define own smearers using inheritance

NOT a "physical detector": represents the **overall performance** in measuring a quantity.

Smearer / **Smearer** / **Smearer** / **Smearer** / **Smearer** → **"Detector"**

- Apply all smearers to an MC event
- Yield smeared event
- Optionally recalculate derived values e.g x, Q^2

How to use it

Simple "Device" smearers define σ(X) via text string

- Write a ROOT script:

```
Smear::Detector createDetector() {
// Resolution in momentum, sigma(P).
// sigma(P) = 0.4%P + 0.3%P^2.
Smear::Device tracking("P", "0.004 * P + 0.003 * pow(P, 2)");
  // Add devices to a Detector.
  Smear::Detector detector;
  detector.AddDevice(tracking);
  return detector;
}
```

Handles event loop, file I/O

- Smear your ROOT tree:

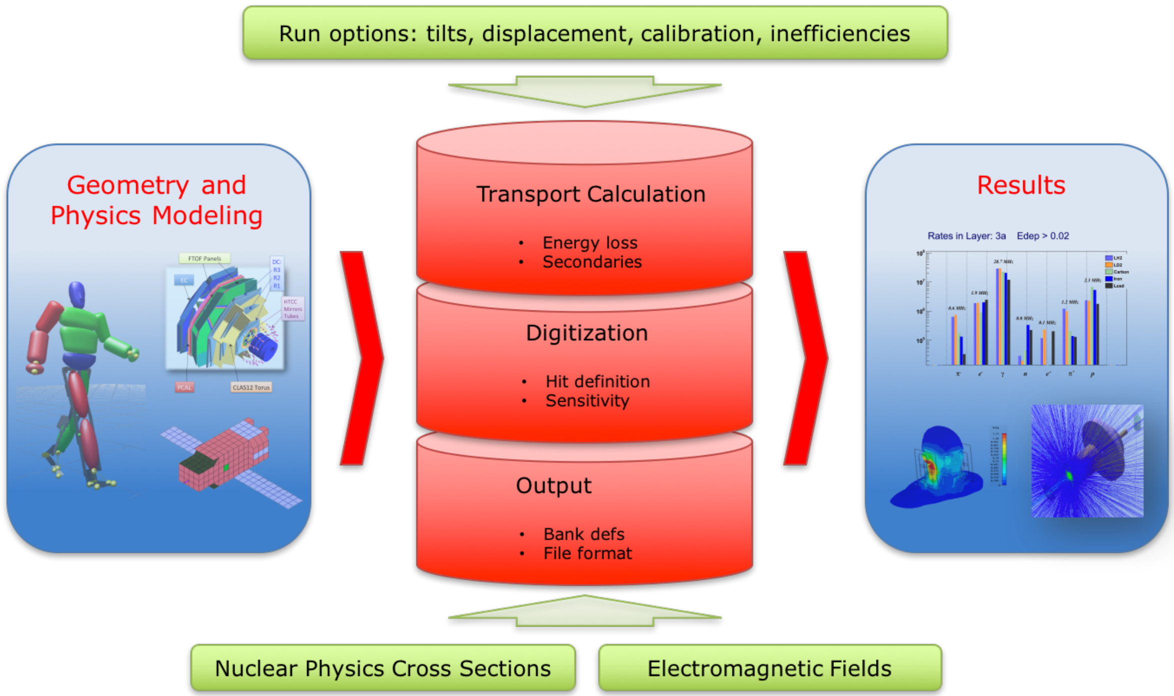
```
root[0] SmearTree(createDetector(), "mc.root", "smeared.root");
```

- "Standard" detector descriptions (like STAR or BeAST) exist

GEMC

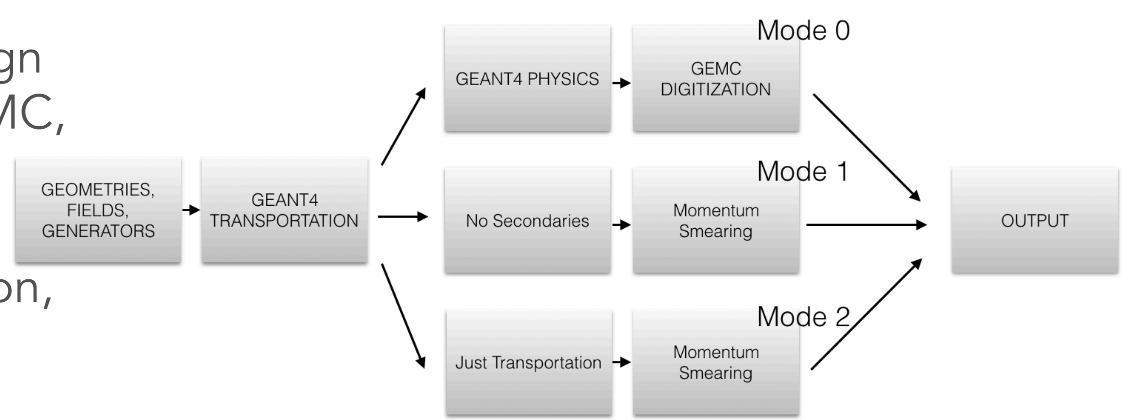
by Maurizio Ungaro (JLab)

GEant4 MonteCarlo Architecture



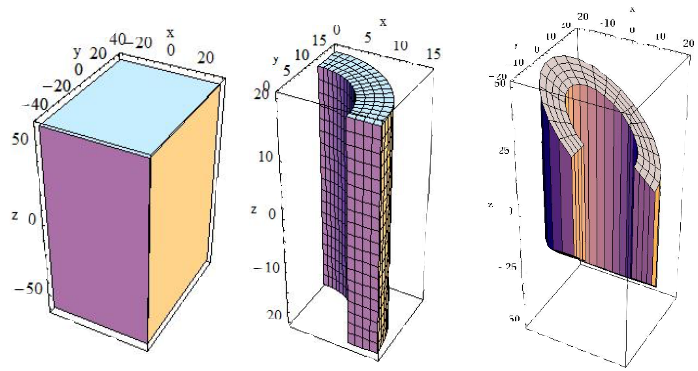
- Application independent geometry/digitization/fields: definitions stored in databases
- Realistic hits treatment: electronic time window, voltage versus time signals.
- Sensitive attributes assigned at run time: real calibration, survey tilts and displacements.
- Plugins for generator formats (LUND, BEAGLE, easy expansion)
- Plugins for output formats (TXT, CODA, JSon, easy expansion)
- Realistic signal treatment allows for background rate studies, including pile-up effects

- Application for detector simulations based on Geant4
- Macro language for detector design
- Various geometry definitions: GEMC, gdml, CAD
- Data card (XML) to steer application, all Geant4 macro commands supported by design



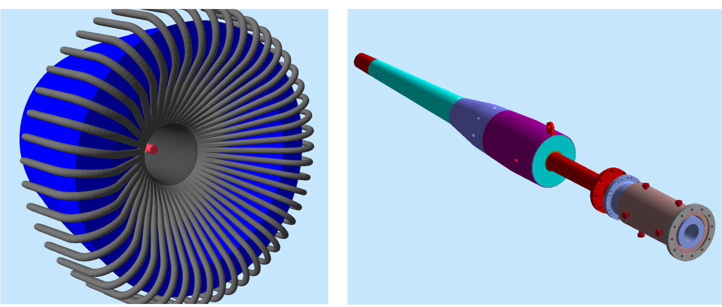
Geometry

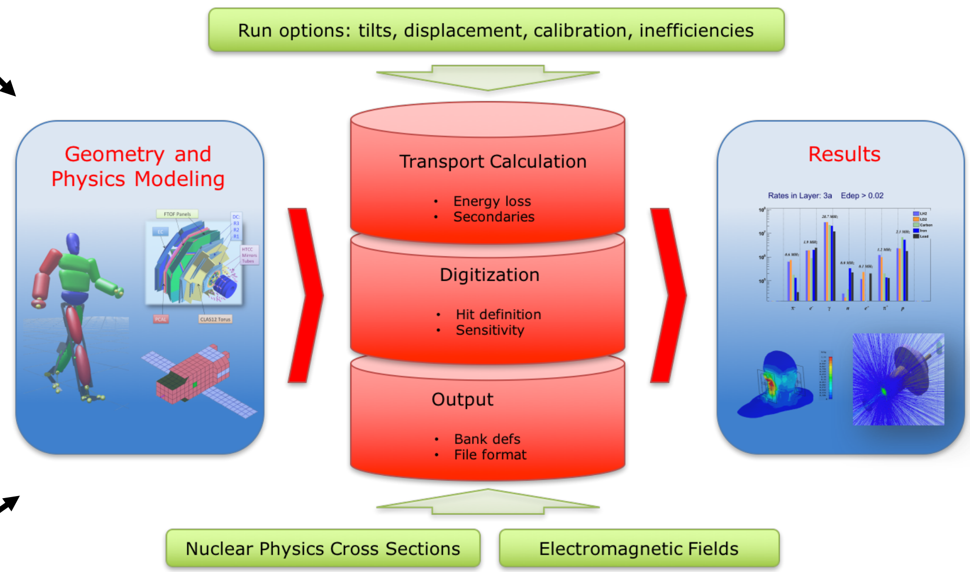
Native



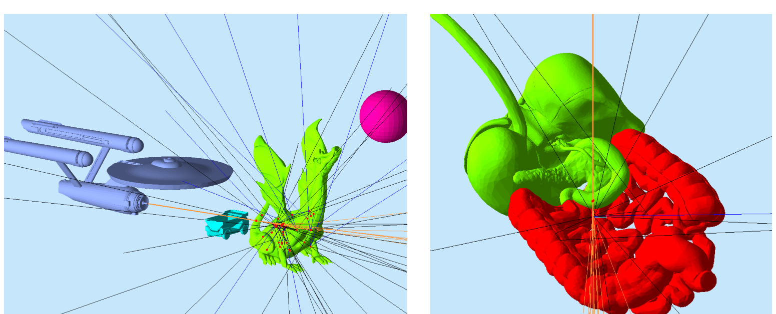
Input: Native, CAD, GDML. Arbitrary hierarchy, can be mixed and matched. Materials, sensitivity assigned at run-time.

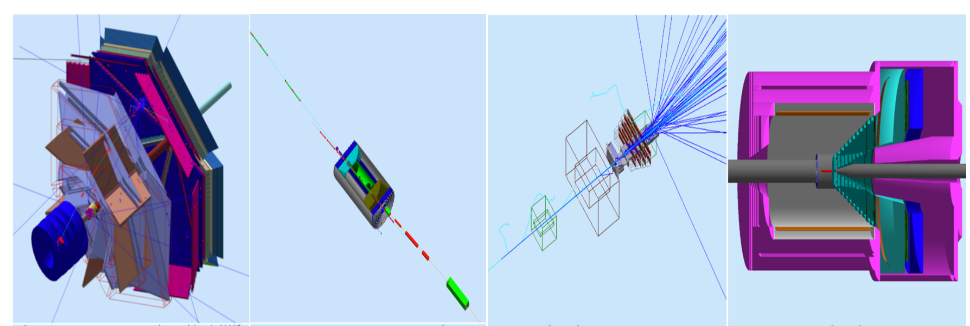
CAD





GDML

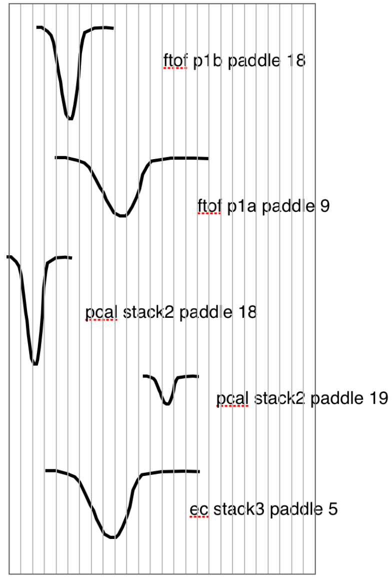


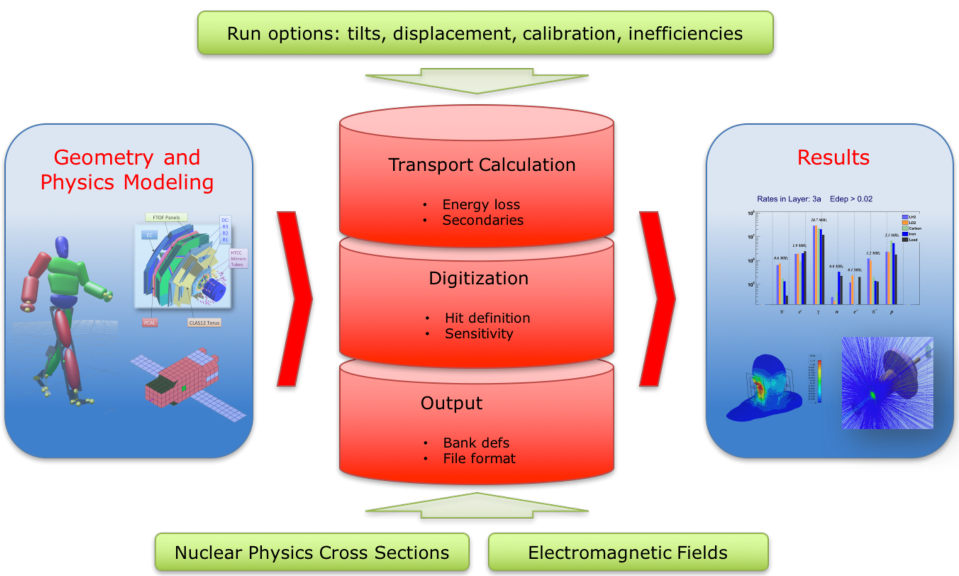


Experiments using the GEMC Framework: CLAS12 (Hall-B), EIC Beamline and detectors, HPS, Solid

Digitization, Output

- Single ADC/TDC over electronic time window.
- Voltage vs time signal.
- FADC output (4ns intervals or integratal mode)
- Automatic true information
- All g4 steps in the output

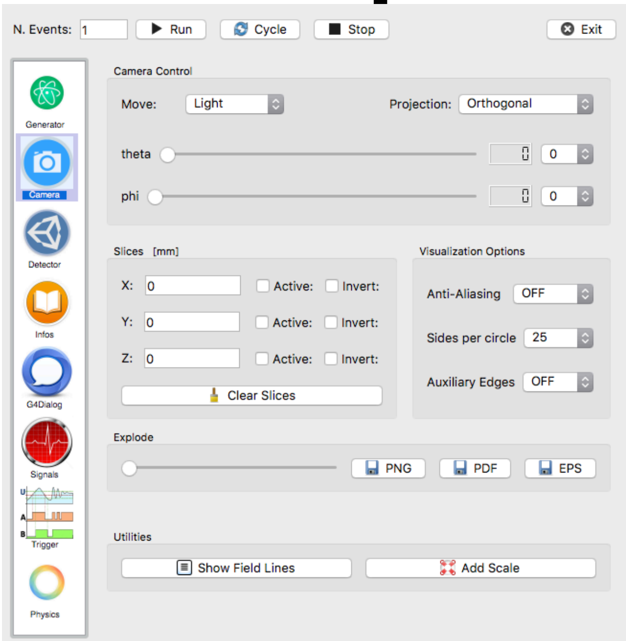


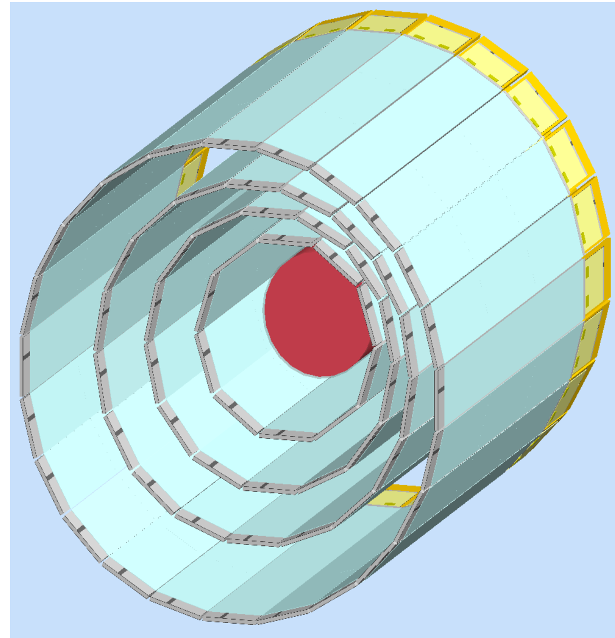



```
> BST

  > True Step by Step infos   (101, 0)
     - Edep                   (101, 1)
     - Pid                    (101, 2)
     - positions              (101, 3)

  > Dgtz Step by Step infos   (102, 0)
     - ADCL                   (102, 1)
     - ADCR                   (102, 2)

  > True Integrated infos     (103, 0)
     - Edep                   (103, 1)
     - Pid                    (103, 2)
     - positions              (103, 3)

  > Dgtz Integrated infos     (104, 0)
     - ADCL                   (104, 1)
     - ADCR                   (104, 2)

  > Voltage as a function of time (105, 0)
     - Identifier                 (105, 1)
     - Time                       (105, 2)
     - Voltage                    (105, 3)

 > Trigger Bank               (106, 0)
     - Identifier             (106, 1)
     - Time                   (106, 2)
     - Voltage                (106, 3)
```

Graphical Interface

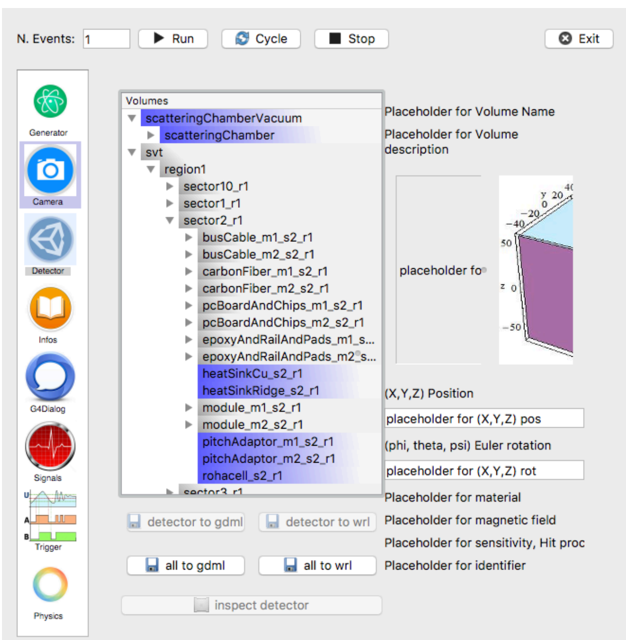
- Generator
- Event time window
- Background beams
- Camera views slices.
- Axis, Scale, Show field.

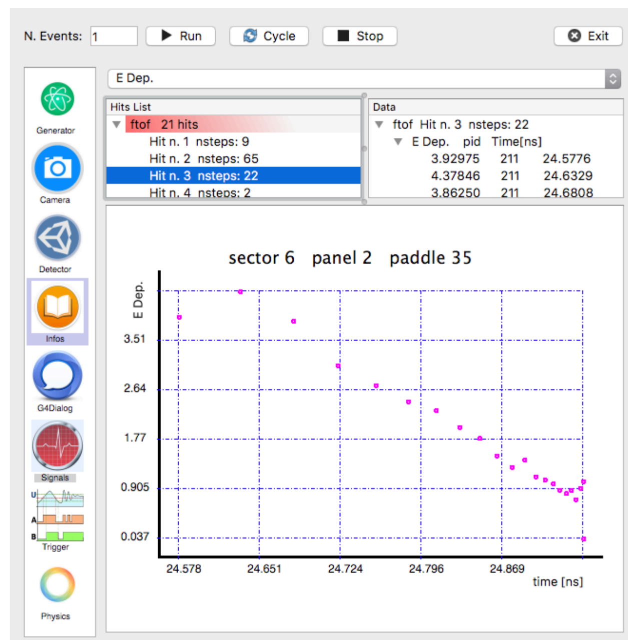




- Geant4 OpenGL View for the whole detector.
- Can inspect and open a view on single volumes.

- Volumes hierarchies and properties
- Output to GDML





- Graphical analysis of steps in a hit.
- Can choose variable to display.

fun4all

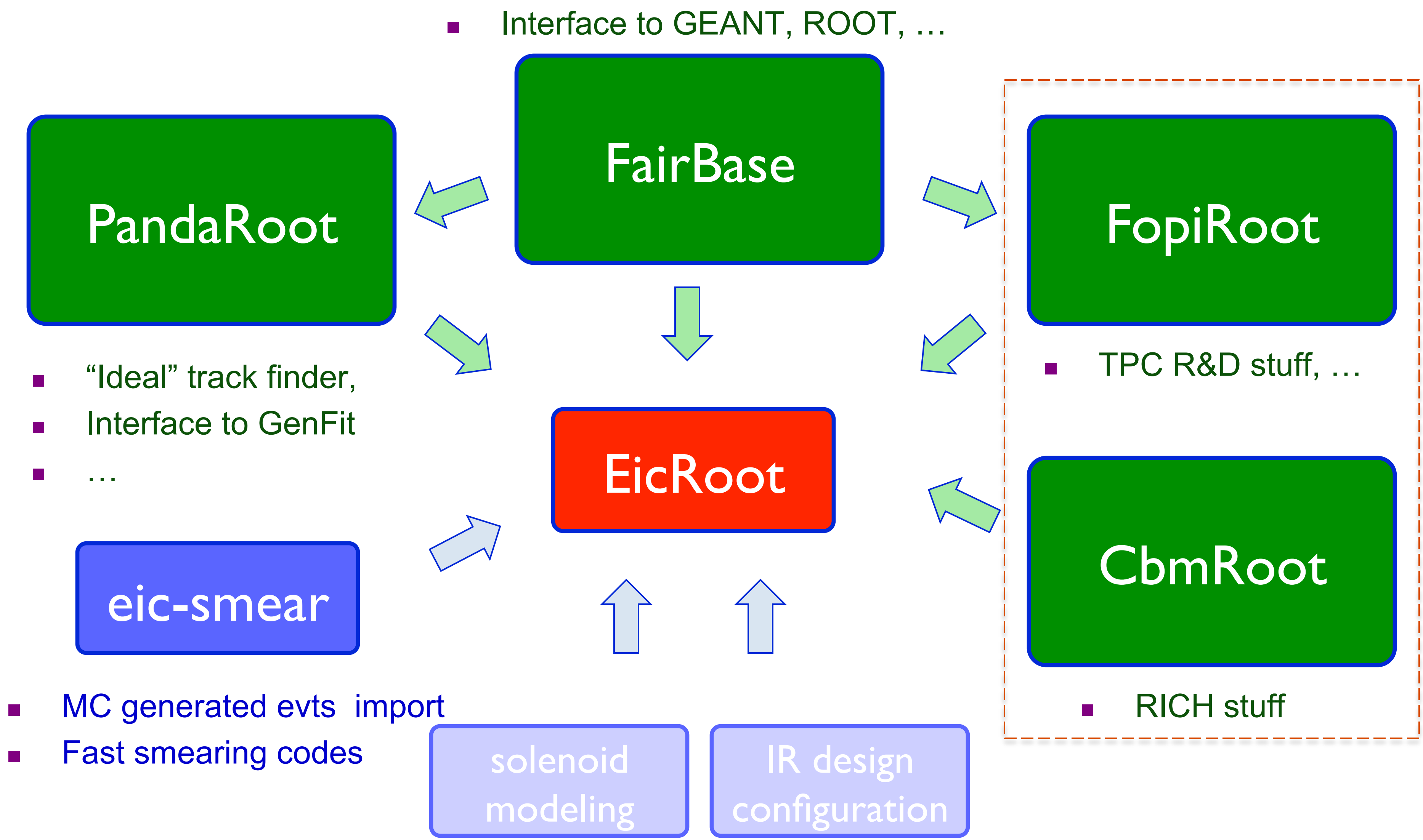
by Chris Pinkenburg (BNL)

-> see the previous talk

EicRoot

by AK (BNL)

EicRoot framework building blocks



-> basically a yet another FairRoot software clone

End user view

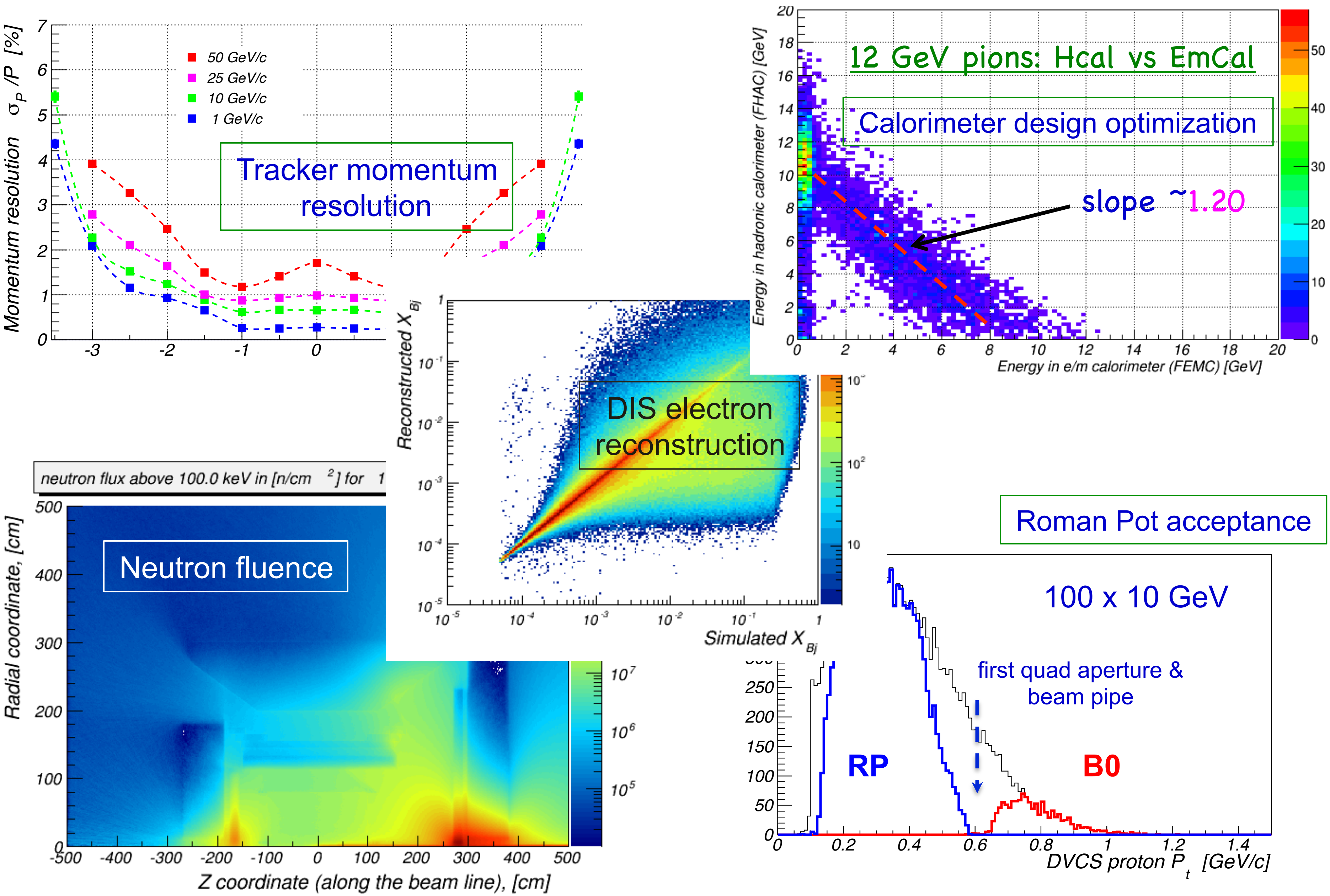
- No executable (steering through ROOT macro scripts)

simulation → **digitization** → **reconstruction** → **PID; assembly**

-> MC points | -> Hits | -> Tracks, clusters | -> Events

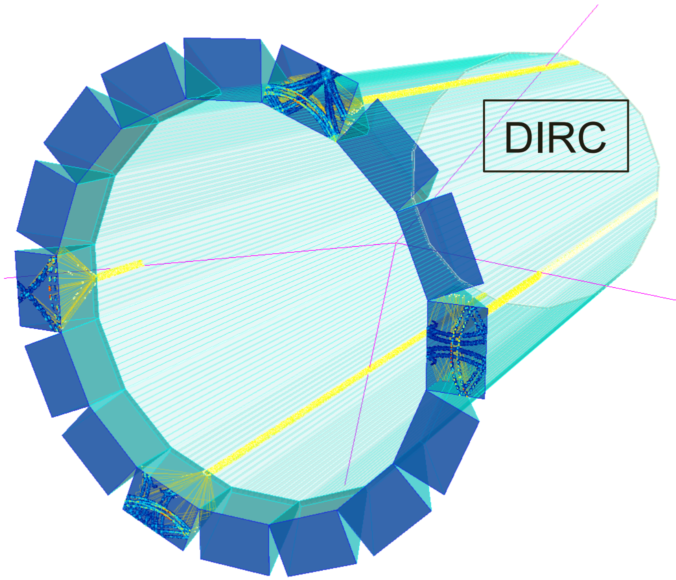
- ROOT files for analysis available after each step
- C++ class structure is well defined at each I/O stage

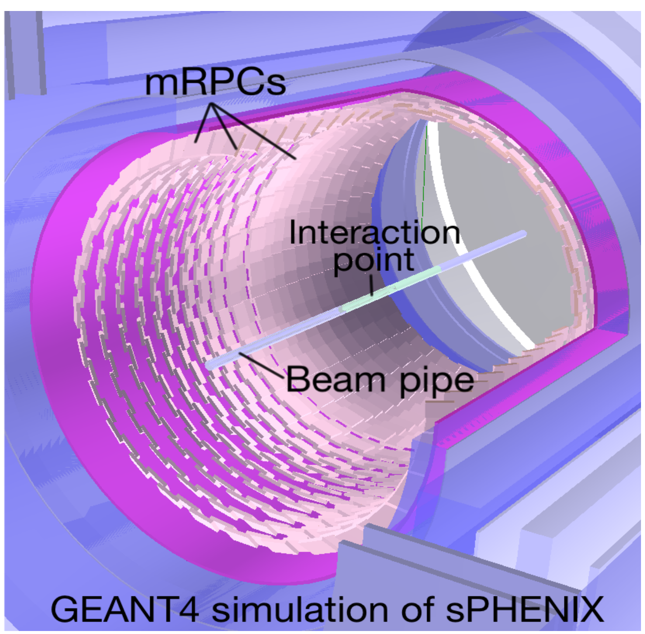
Example case studies

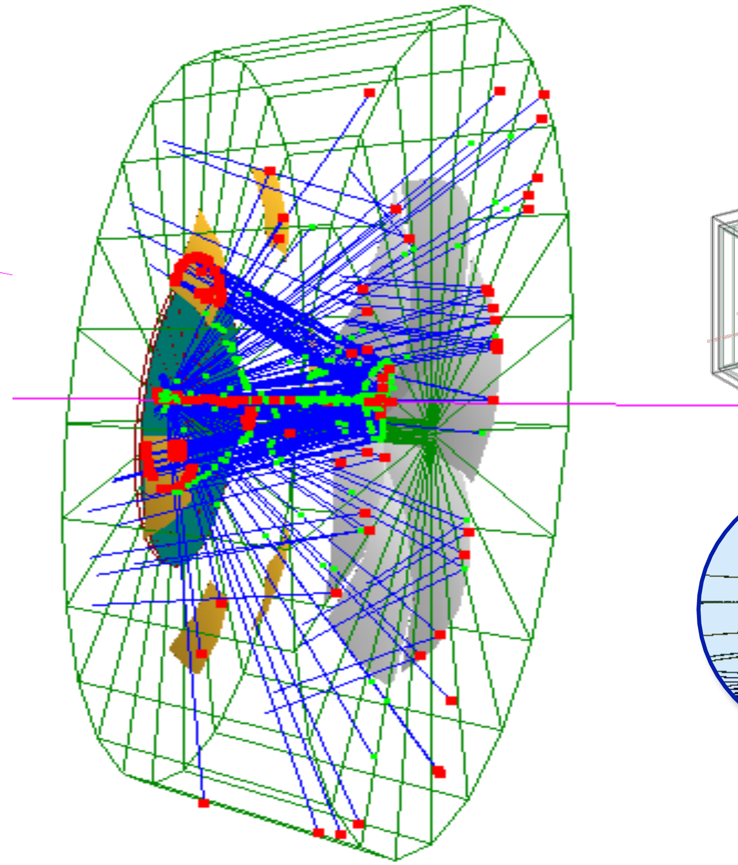


PID Consortium software

Mostly RICH & ToF applications

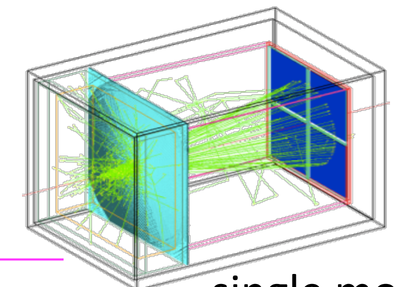




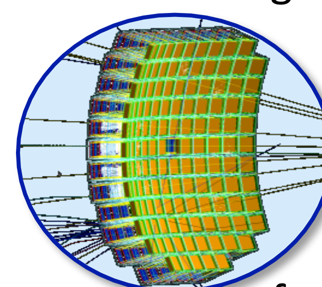


Dual radiator RICH

GEMC



single module



for JLEIC

Modular RICH

- All are custom GEANT4 codes
- No easy way to incorporate in EicRoot

Argonne EIC software

Full simulation and reconstruction chain

Data Model

Event generation

Produce the simulation input events

Detector simulation

Particle transport through detectors

Digitization

Turn energy deposits in active media into detector hits

Reconstruction of

Event vertex, charged tracks, Particle Flow Objects (PFO)

Perform analysis

Collection of benchmark analyses

Argonne Software: Overview

Legacy chain

Adaptation of the SiD (ILC) simulation and reconstruction software chain

Major parts

- SLIC (wrapper around GEANT4)
- LCSIM (digitization and event reconstruction)
- slicPandora (PFA reconstruction)

Visualization with JAS4pp

Limitations

- Only SiD subdetectors (e.g. no RICH)
- Geometry description not centralized
- Geometry constrained to be symmetric
- Some parts difficult to maintain

Full chain

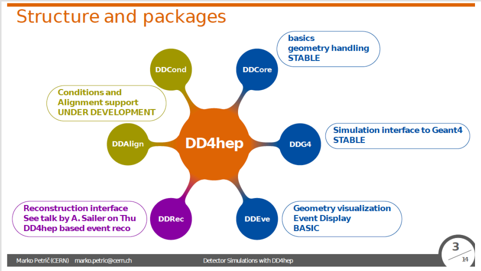
- Available
- Studies of F_2 reconstruction, timing…

Evolution chain

Evolved from the legacy chain

Geometry interface

DD4HEP



Features

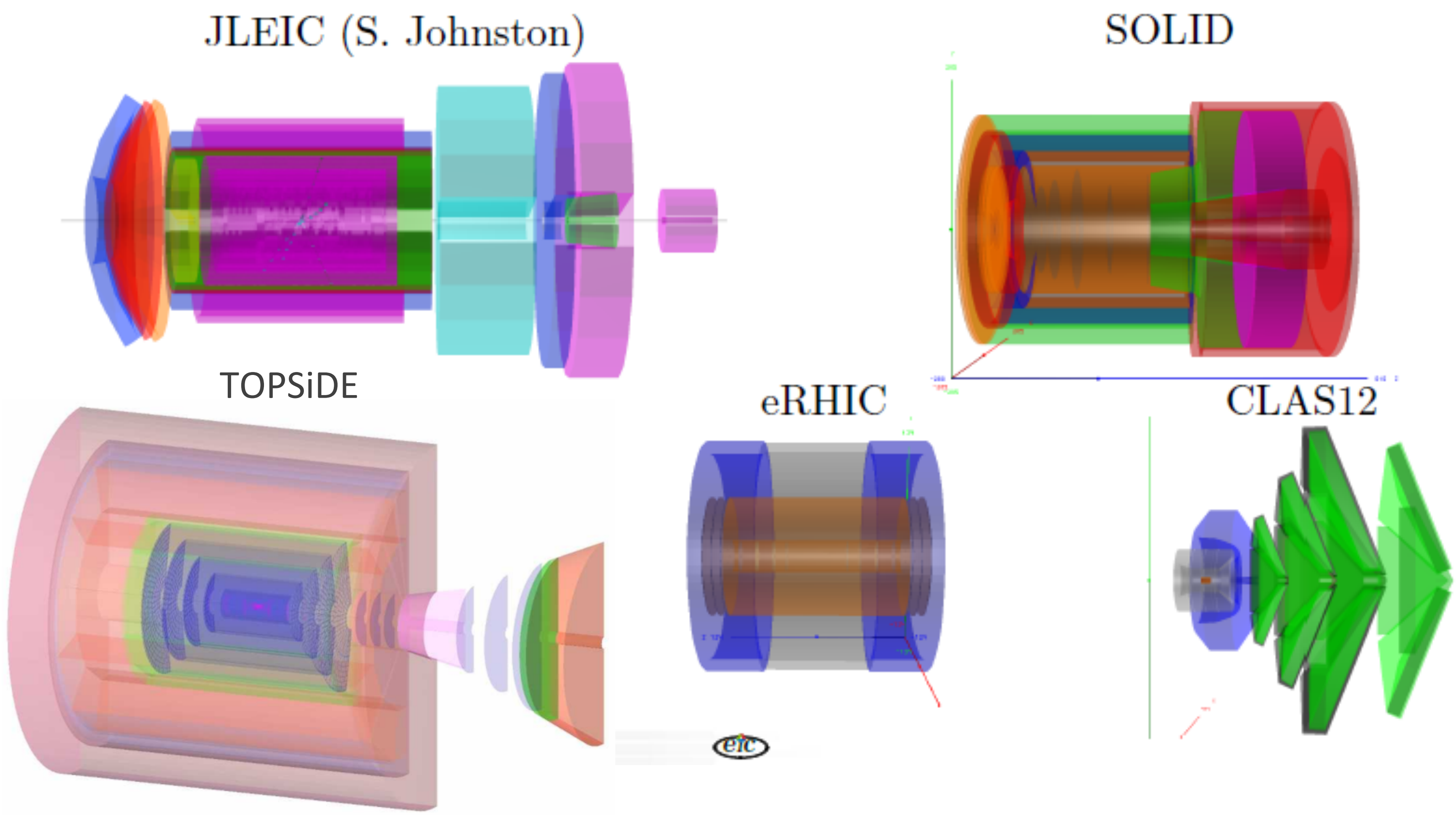
- Fully maintainable
- Geometry obtained from single source
- Geometry can be parametrized
- Geometry not constrained to be symmetric
- New subsystems can be easily implemented

Still working on

- Realistic digitization
- Generic tracking
- PFA reconstruction
- Visualization

Nuclear Physics Detector Library (NPDet)

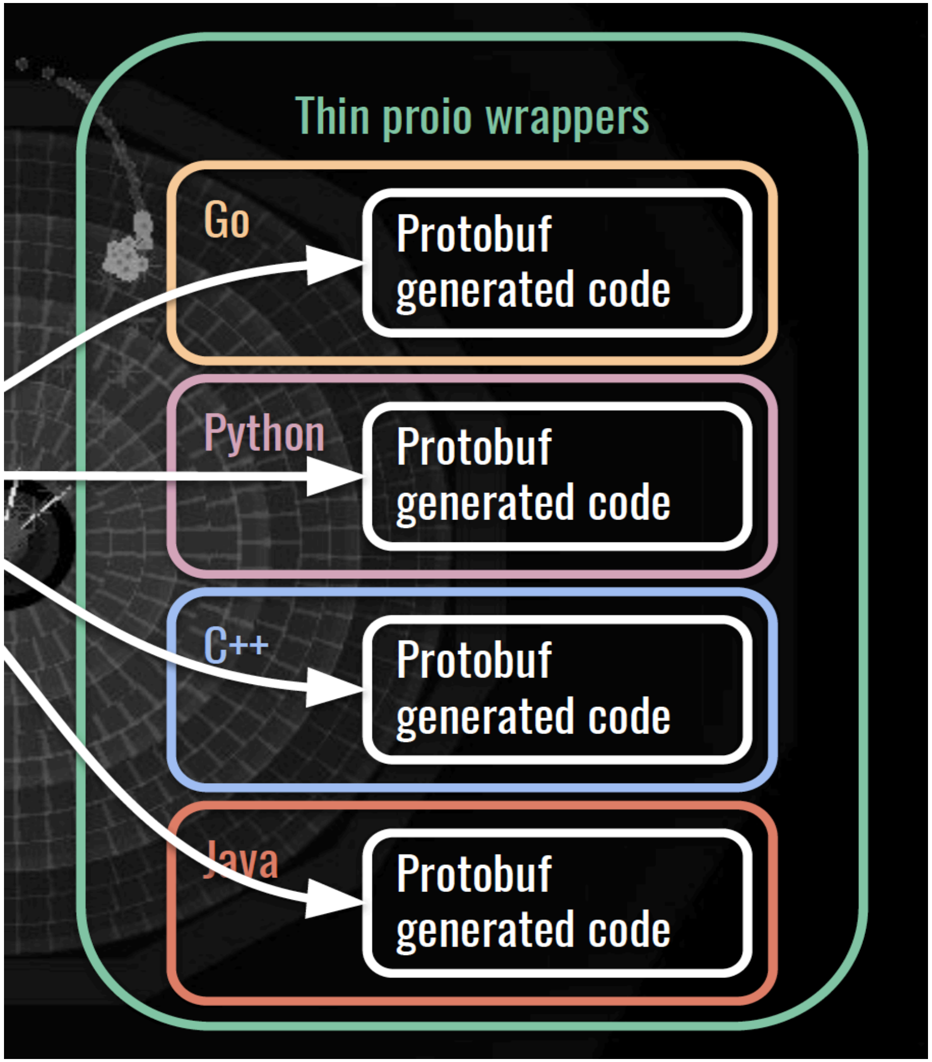
Collection of **parametrized** detectors which can be developed into full concepts



ProIO

ProIO Key Concepts

- Language-neutral I/O for streaming events
- Thin, native containers for protobuf messages, simply adding the concept of an event
- protobuf + event structure = ProIO
- Serialized output can be accessed effectively in archival file, or in a stream



Grand unification, yet another try

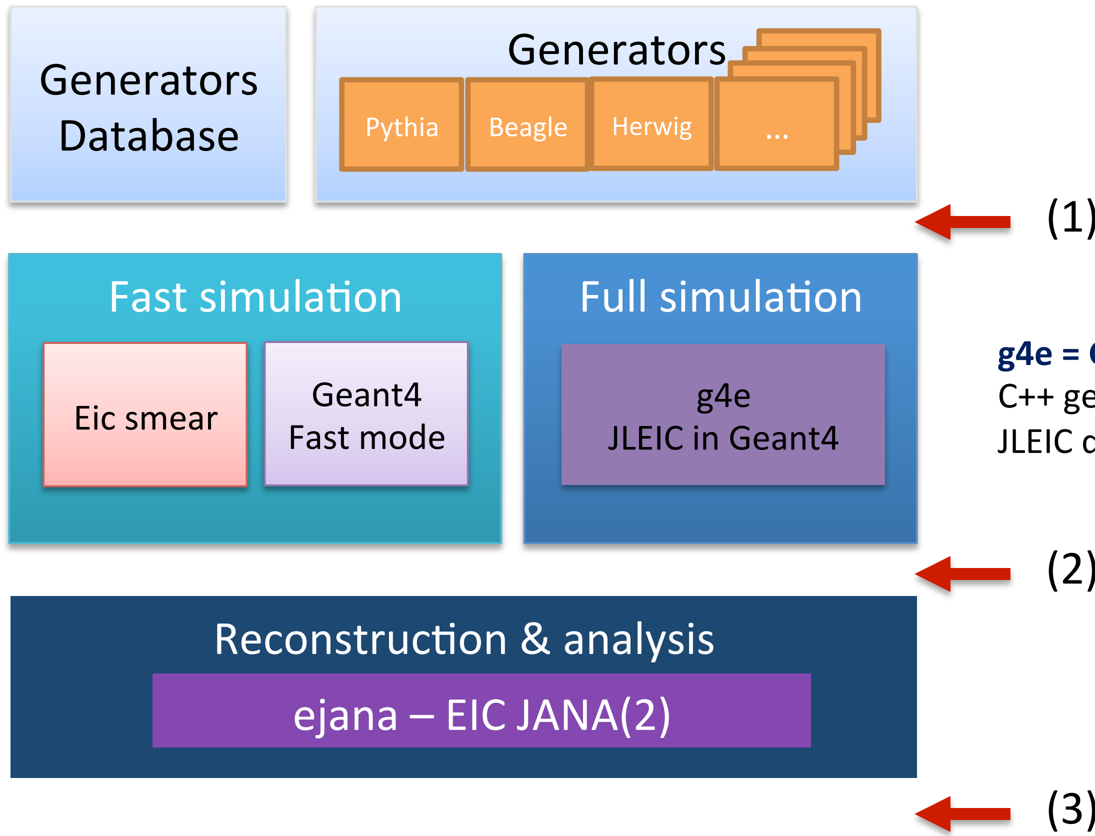
by Dmitry Romanov, David Lawrence & others (JLAB)

Overview

Database with various
MC samples

BNL & Jlab effort on
Fast detector prototyping

ejana = EIC Jana
Community reference
reconstruction



g4e = Geant 4 EIC
C++ genue GEANT4 with
JLEIC detector in it

(1) MC events
(2) Digitized hits + magnetic field + material distribution
(3) Reconstructed events

Software distribution

NO EFFORT AT ALL
Novice

Some effort
Experts

Efforts required axis

Cloud

Containers

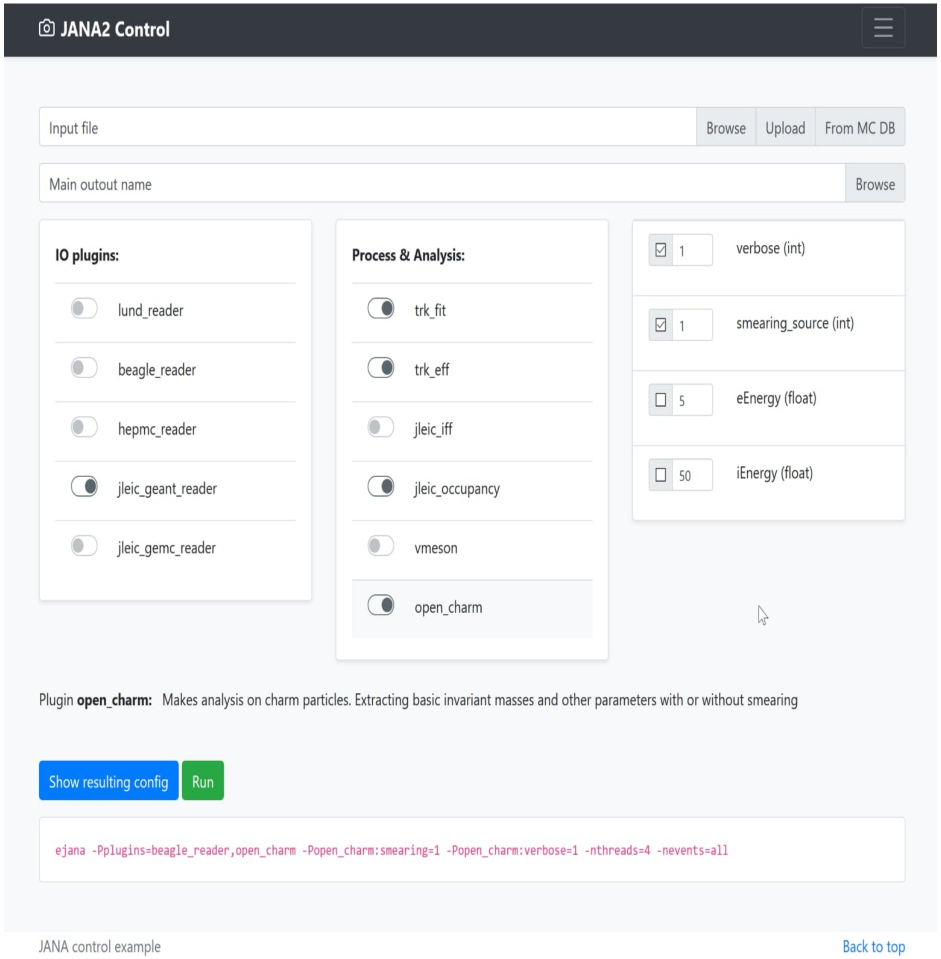
Jlab farm

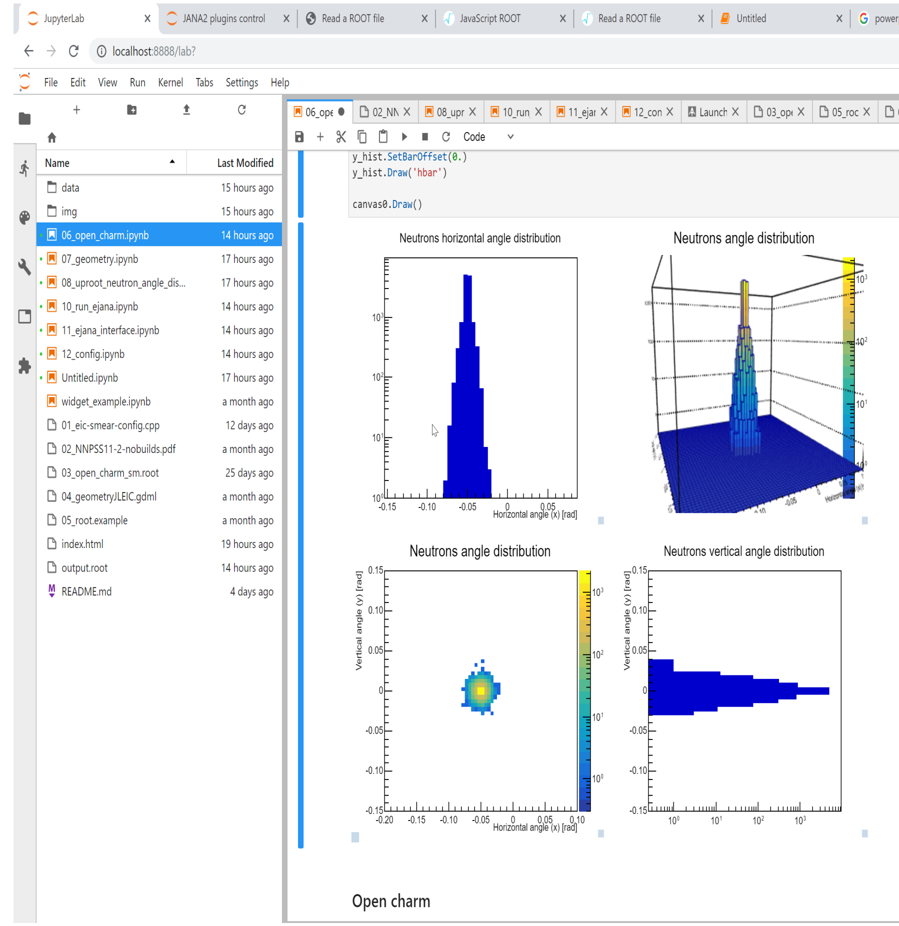
Conda

Workstation
Compilation

EJPM

User Interface





Workflow oriented interactive environment based on Jupyter

e^{JANA} - Community reference reconstruction

e^{JANA} - stands for EIC JANA

- Basic reconstruction
- Physics analysis
- **Users detector codebase integration**

Reconstruction

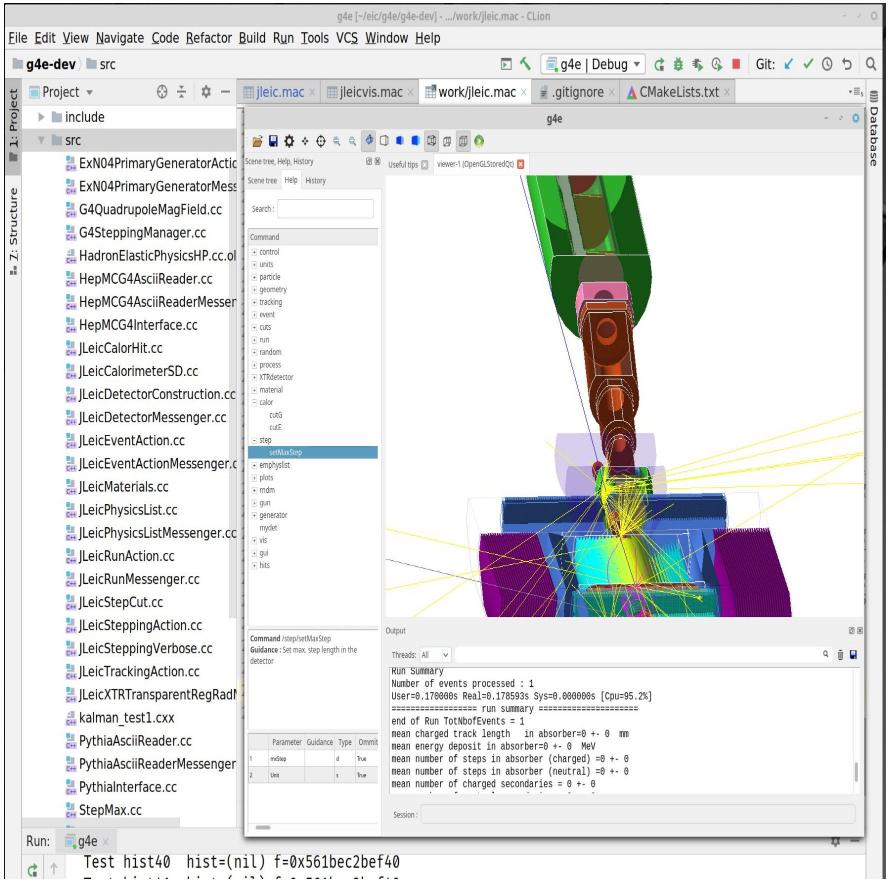
- **Tracking** - Genfit
- **Vertex finding** – Rave
- **Physical analysis**:
  - ROOT C++ or
  - Python data science tools (Jupyter, Seaborn, Pandas, etc)

Any existing C++ (or even others) code can be:

- compiled as JANA plugin
- run parallelized in eJANA
- accessed by other plugins

GEANT 4 EIC

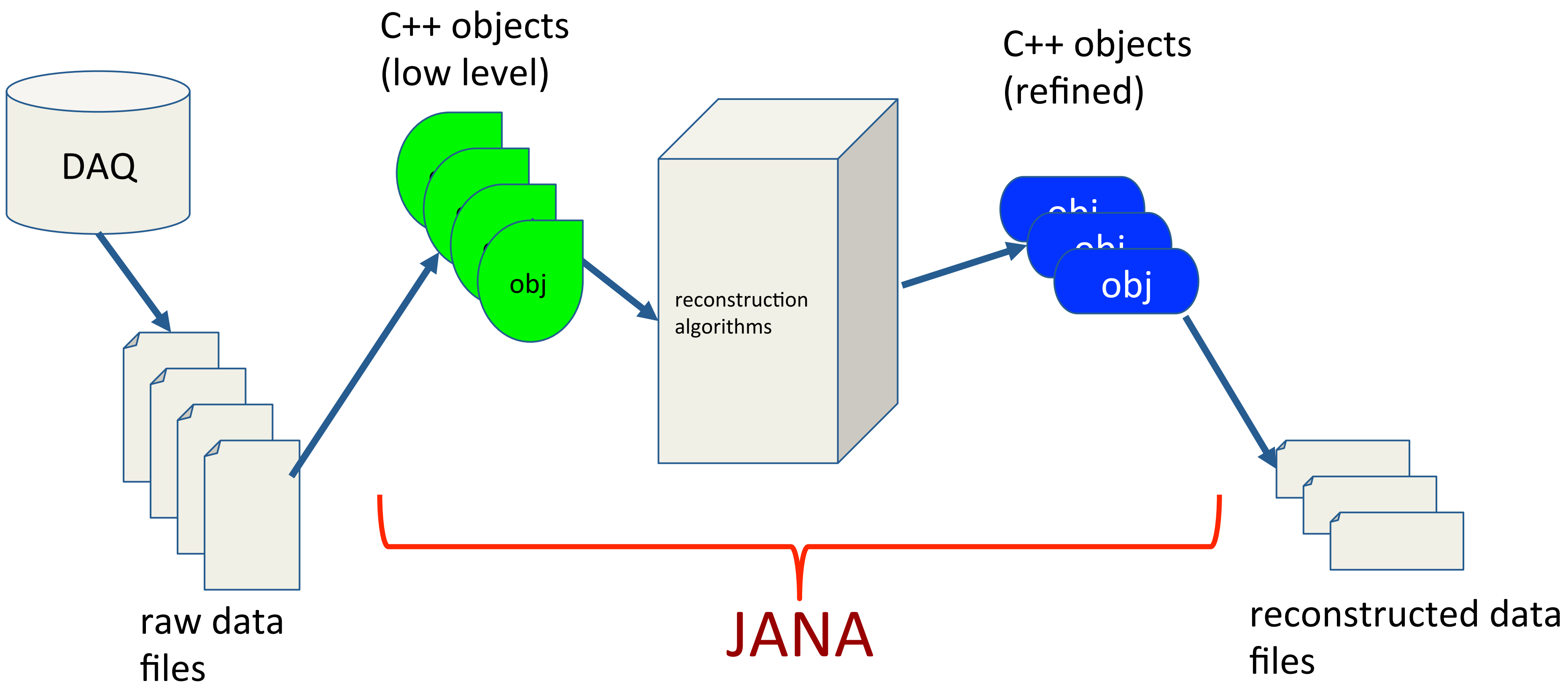
- The codename g4e – stands for **Geant 4 EIC**
- **Beta** stage
- √s 100 GeV design is implemented
- Imports CAD, accelerator group data
- Exports final Geometry in various formats
- Plain flattened analysis ready ROOT files



For those who prefer scripting over compilation
***Geant 4 python** can be used*

Backup (JANA2)

Overly Simplified View of JANA's Role



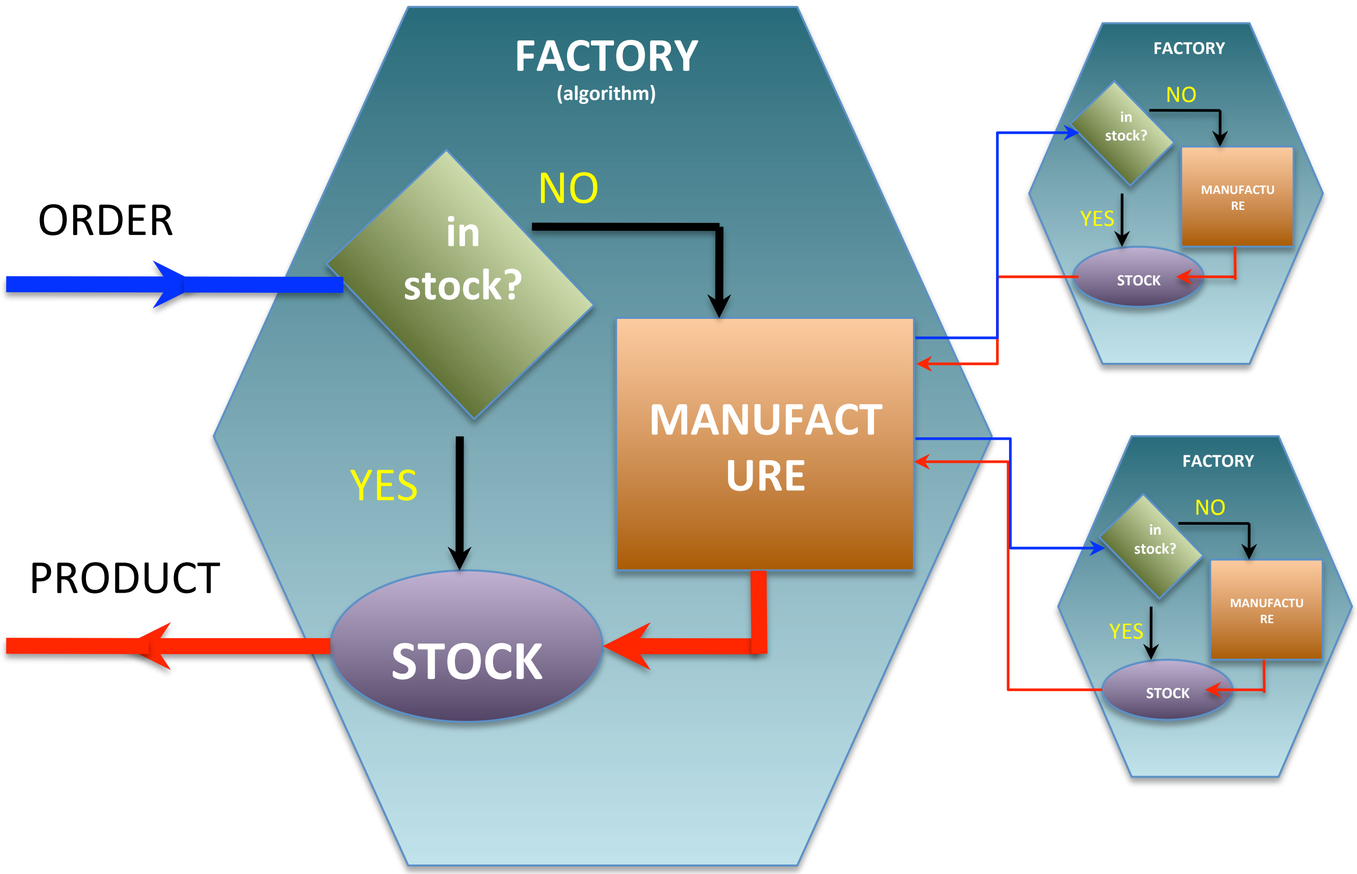
Some Goals of the JANA framework

- Provide mechanism for many physicists to contribute code to the full reconstruction program

- Implement multi-threading efficiently and external to contributed code

- Provide common mechanisms for accessing job configuration parameters, calibration constants, etc...

Factory Model

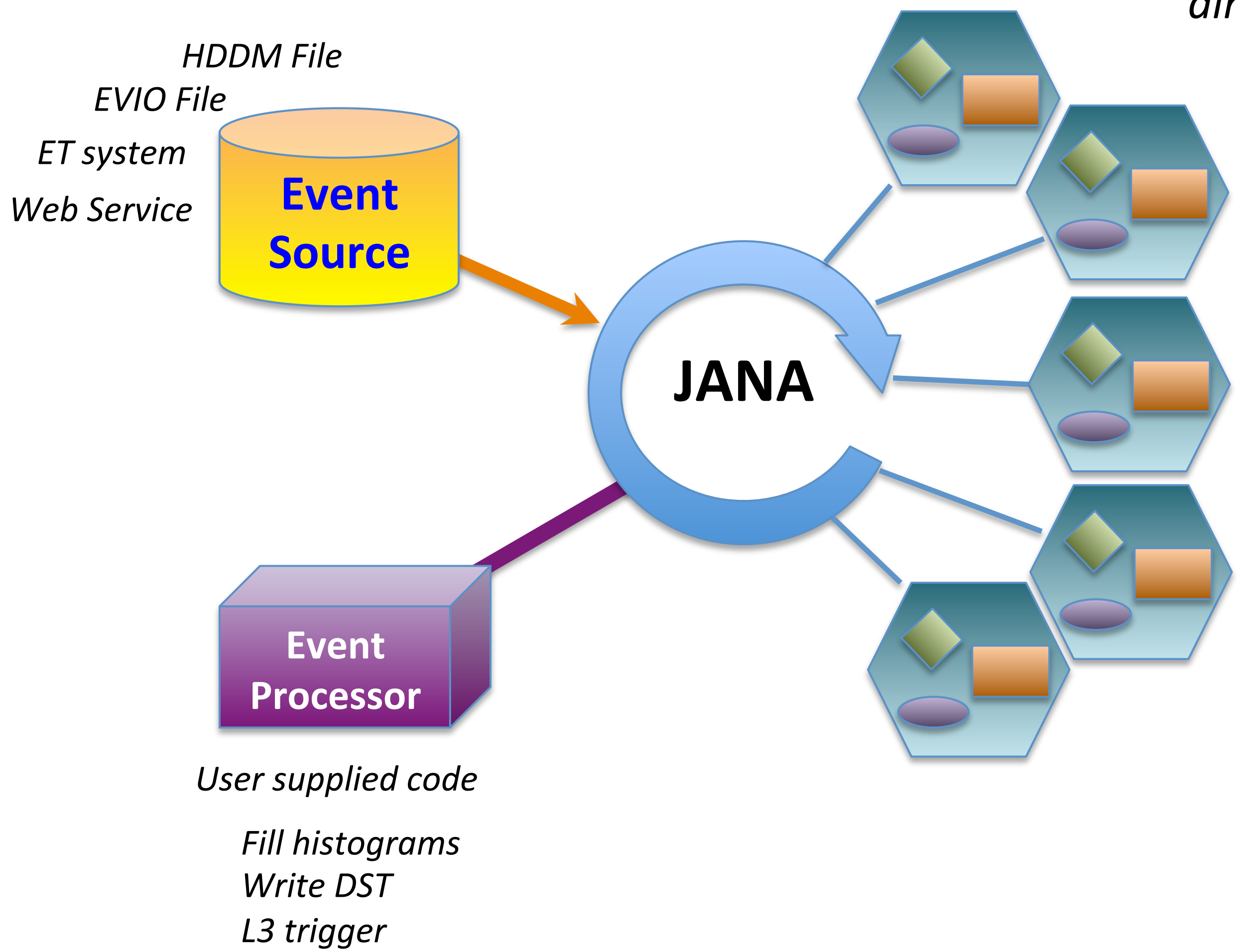


Data on demand = Don't do it unless you need it

Stock = Don't do it twice

Conservation of CPU cycles!

Complete Event Reconstruction in JANA



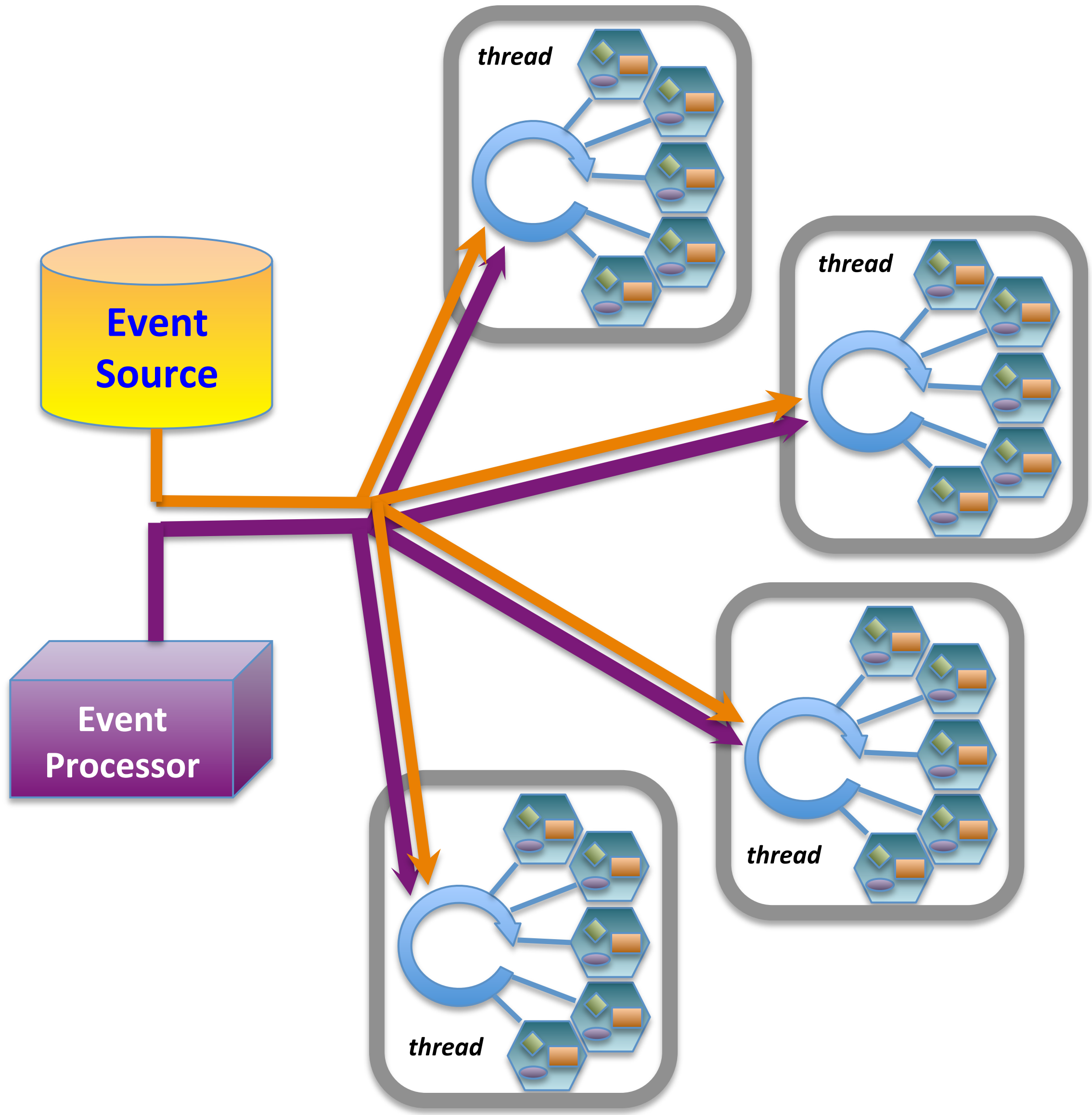
Framework has a layer that directs object requests to the factory that completes it

Multiple algorithms (factories) may exist in the same program that produce the same type of data objects

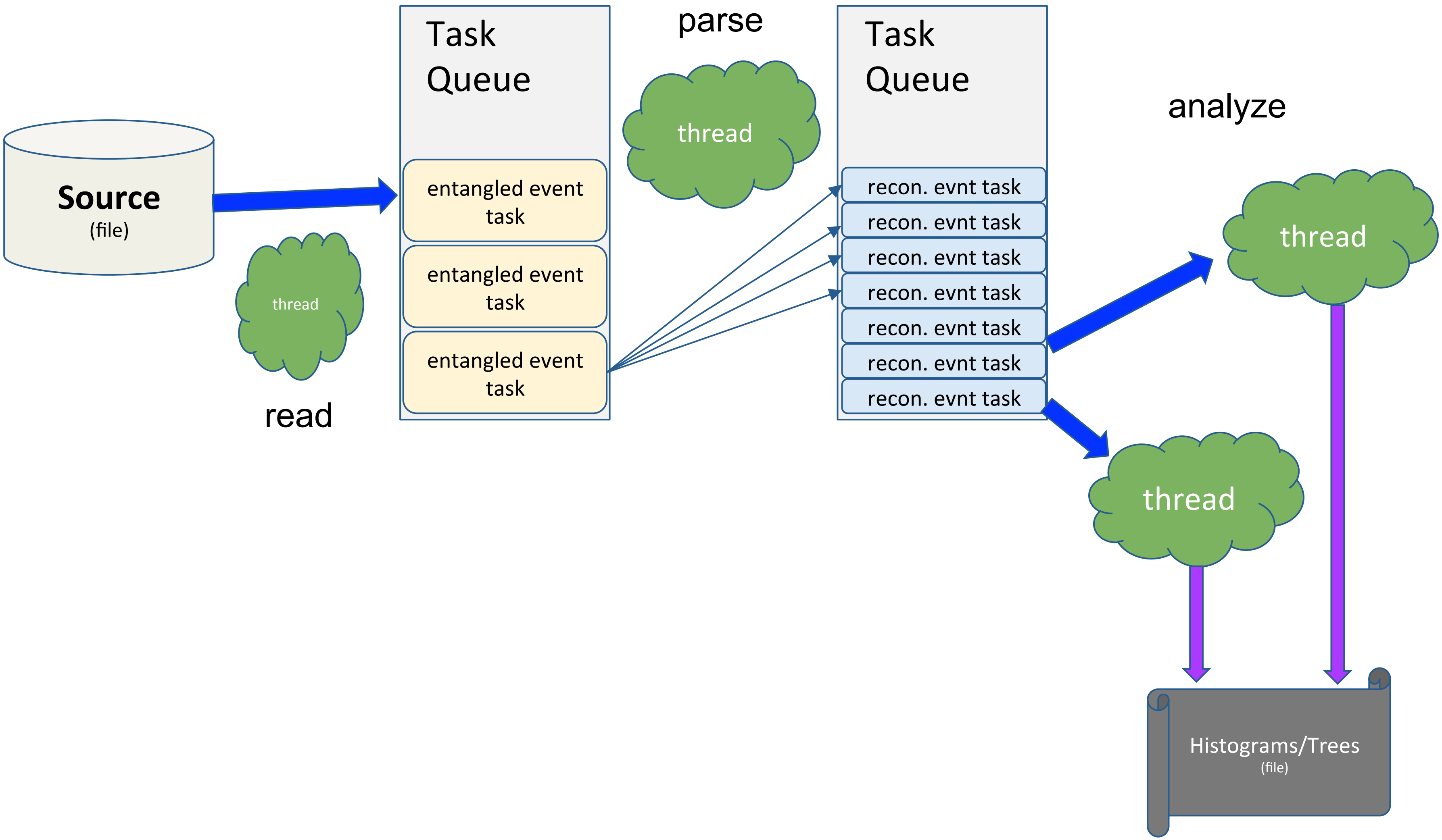
This allows the framework to easily redirect requests to alternate algorithms specified by the user at run time

Multi-threading

- *Each thread has a complete set of factories making it capable of completely reconstructing a single event*
- *Factories only work with other factories in the same thread eliminating the need for expensive mutex locking within the factories*
- *All events are seen by all Event Processors (multiple processors can exist in a program)*



JANA2 generalizes the “event” queue to allow multiple queues. Threads are now responsiible for moving data between queues



JANA2 arrows separate sequential and parallel tasks

- CPU intensive event reconstruction will be done as a parallel arrow
- Other tasks (e.g. histogram filling) can be done as a sequential arrow
- Fewer locks in user code allows framework to better optimize workflow