EIC Software Infrastructure Review

EPIC

JANA2 Reconstruction Framework

David Lawrence

On behalf of the EPIC Collaboration

Outline

1. Framework Decision
2. Requirements
3. Convener Summary
4. JANA Usage
5. Current Software Development
   a. JANA2 development
   b. EICrecon Development
   c. Algorithm Porting Status
6. Schedule

Reconstruction Framework Decision

Two options presented:

- JANA2
- Gaudi

Decision Document:
https://docs.google.com/document/d/1lomak02ztchkwQB2d_f-58gabBOQF9WaPaQhf8kTvfY/edit#heading=h.si54k0yjzea0

Overview, ***Requirements****, options/speakers, discussion,* ***Convener Summary***

Meetings spread over two weeks:

- June 29 : https://indico.bnl.gov/event/15644/
  - JANA2
  - Gaudi
  - Key4hep
- July 6 : https://indico.bnl.gov/event/15645/
  - ~~ERSAP+TriDAS+JANA2~~ (speaker unavailable)
  - JANA2
  - Gaudi

Live Notes document:
https://docs.google.com/document/d/1IdlQ63PxfIDsGdOlkik0OE76EzIpHCt00Br583hOJxI/edit?usp=sharing

Requirements

- *The reconstruction framework must be able to run on both simulated events and real data. Even if there may be algorithms that use truth information (or even require truth information, initially), the reconstruction framework itself should allow for running without truth information.*
  JANA's ***factory tag*** mechanism can be used to tag "TRUTH" versions of objects. The tagged versions of objects may be requested programmatically or on a global scale at runtime via configuration parameters. Both the TRUTH tagged and the un-tagged versions of the objects may coexist.

- *The reconstruction framework must be able to take advantage of heterogeneous computing resources (multiple cores, GPUs, etc).*
  Sub-tasks were added in JANA2 specifically to add additional heterogeneous support.

- *The reconstruction framework must encourage modular approaches to algorithm development, using defined interface layers.*
  JANA has a set of base classes that define the interface. Design emphasizes a factory having one primary class of object as its output encouraging users to implement a more modular design. e.g. Track seeds can be produced in one factory and fully fit tracks in another allowing the seed finding algorithm to be easily swapped. The framework also allows for both types of objects to be produced in a single factory, but the current JANA2 design encourages the developer to break that up into smaller modules instead.

Requirements

- *Algorithms must be implemented using the selected data model, and ensure that data (event data, geometry description, and algorithm parameters) are kept separate from the algorithm itself.*
  The algorithm parameters (*Configuration Parameters* in JANA) can be set via config file or command line argument and are managed at the framework level. Geometry is provided by a service (dd4hep). The event data is managed by the framework.

- *Algorithms must be implemented in the framework independently from any scheduling strategies; an algorithm must not need to know how it is orchestrated, whether it is running in parallel, in single or multithreaded mode, concurrent or not, in online or offline analysis mode.*
  JANA algorithms are ignorant of this type of information which is handled at the framework level.

- *The reconstruction framework must be open source, accessible to the entire community, and managed by a sustainable core team.*
  The source is freely available from GitHub. Any GitHub user in the world is able to submit issues and PRs to the JANA2 repository. JLab has committed to support JANA throughout the EIC project as a full partner lab.

Requirements

- *The reconstruction framework must be able to pass (and add) metadata and so-called slow control information to the output files, so input files are not needed and output files can stand on their own.*
  JANA allows objects of any type to be inserted into an event. Writing output files necessarily relies on tools that interface with the Data Model and so are not explicitly part of the framework itself.

- *The reconstruction framework must be able to run in streaming readout mode, that is:*
  - *with access to only parts of an event (single detector, single sector),*
  - *with events (or parts of events) appearing out of sequence,*
  - *individual algorithms must not rely on an algorithm-specific internal state to be able to make sense of disconnected parts of events.*

  Recent Streaming Readout beam tests with JANA2 have been performed. Please see: https://arxiv.org/abs/2202.03085. JANA2 supports streaming at multiple levels. The on-demand design naturally supports processing of partial events. This is an extremely common exercise in GlueX.

Additional assessment criteria

- *Amount of 'boilerplate' code that must be written by algorithm developers.*
  The ***jana-generate.py*** script generates boilerplate code. This includes making a complete stand-alone plugin including a CMakeLists.txt file.

- *Ability by the framework to avoid e.g. memory errors through interface enforcement mechanisms (e.g. const passing).*
  JANA passes pointers to const objects between factories and processors. This is required for reproducibility should the order of factory calls be changed between program invocations.

- *Ability for shared algorithm development between the two EIC detector collaborations (and/or outside of the EIC).*
  Algorithms are currently being ported from the EIC detector proposal effort into a generic form that can be used by either JANA or Gaudi. This will allow sharing of algorithms even if a second detector chooses to use a different framework. For experiments using JANA the plugin mechanism in JANA allows a plugin to provide one or more factories(algorithms) to any JANA executable. Thus, a single pre-compiled plugin can be used in multiple experiments*.

- *Use of modern and sustainable coding practices, including in the code written by algorithm developers and other contributors.*
  JANA is maintained in a Github repository. The issues, pull requests, and release mechanisms are used to maintain the code. Automated builds and unit tests on multiple platforms are initiated by pull requests. Coding style is posted.

- *Demonstration of performance in production environments.*
  GlueX.

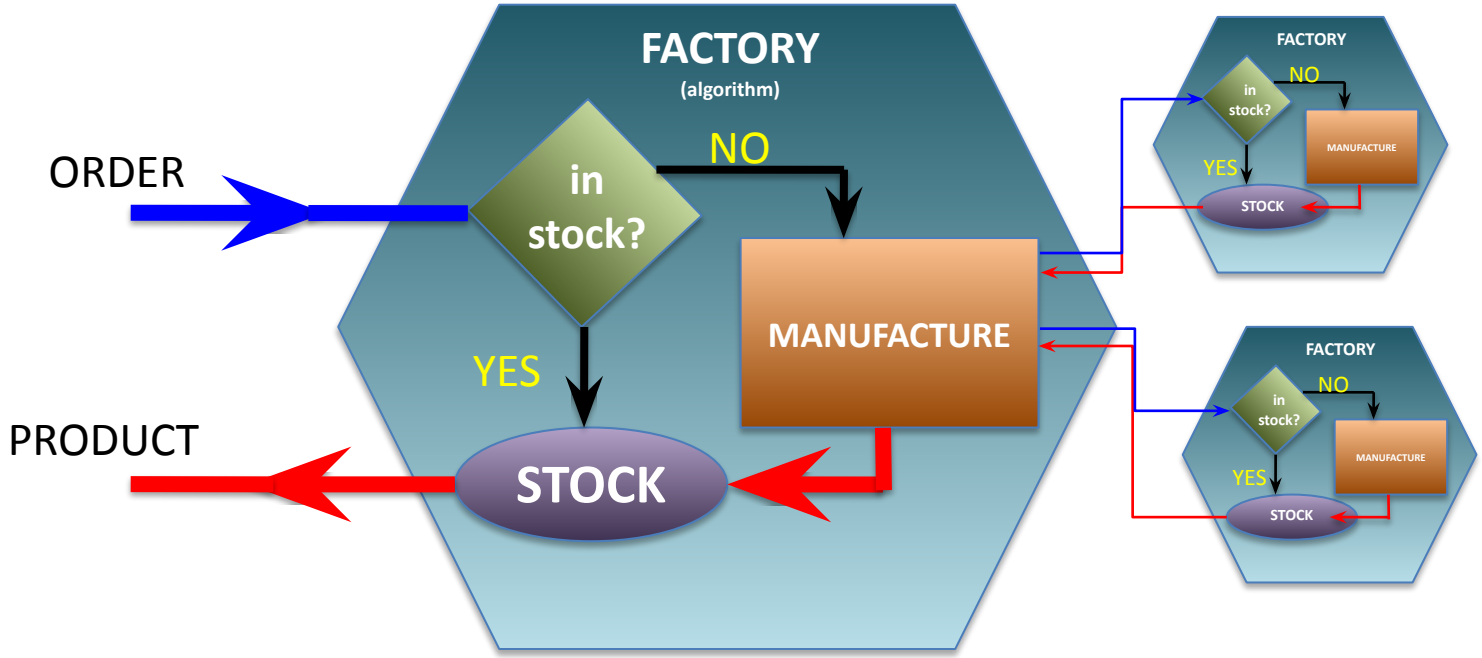
**input and output classes must be available*

Convener Summary

"The working group conveners recommend JANA2 as the reconstruction framework. …"

- JLab committed 1 FTE to support JANA2 and EIC software while multiple instances of attempting to get support from Gaudi developers failed during ATHENA development.
- Algorithms developed during detector proposal process can be ported into JANA2 with relatively minor effort. October timeline will likely to be met.
- Continued engagement with broader HENP software community.

Factory Model



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

Stock = Don't do it twice

Conservation of CPU cycles!

Basic data access with JANA

```
auto tracks = jevent->Get<DTrack>();

for(auto t : tracks){

  // ... do something with const DTrack* t

}
```

n.b. `std::vector<const DTrack*> tracks;`

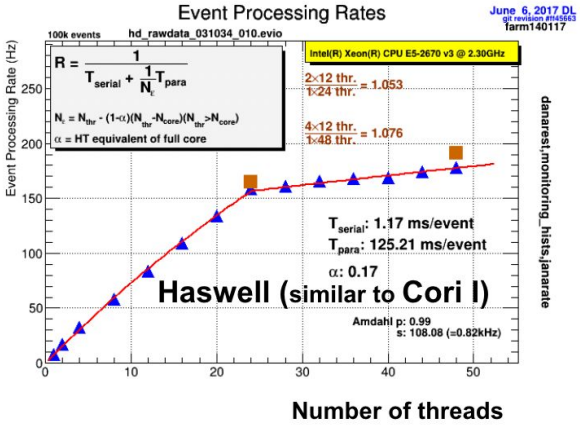
JANA at NERSC

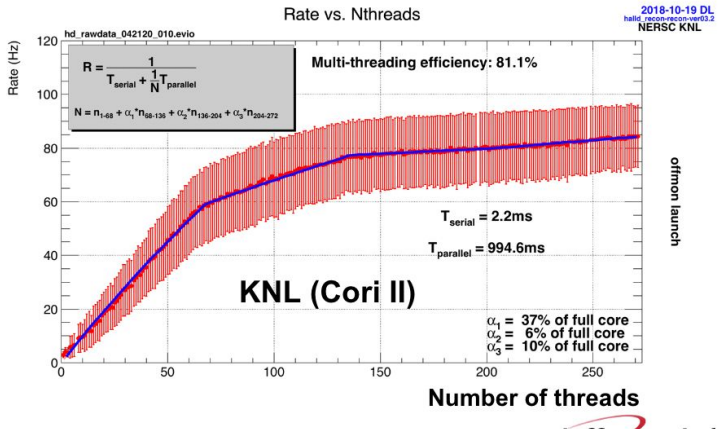
GlueX raw data volumes

2017	0.93PB
2018	3.13PB
2019	0.43PB
2020	4.86PB
2021	1.40PB
2022	1.05PB

GlueX @ NERSC

GlueX Allocation AY2019	58.5M NERSC Units
Input file size	20GB
Wall Time/file on Cori I (Haswell)	3 hours
Wall Time/file on Cori II (KNL)	6.9 hours





8

Offsite Data Processing for the GlueX Experiment - D. Lawrence(JLab) CHEP19, Adelaide, AU

Jefferson Lab

JANA2 v2.0.6

release created August 1st, 2022

EPSCI
EXPERIMENTAL PHYSICS SOFTWARE AND COMPUTING INFRASTRUCTURE

v2.0.6 Latest

Compare

nathanwbrei released this 16 days ago · 13 commits to master since this release · v2.0.6 · 8eacbea

Fixes issues uncovered while porting EICRecon.

- Support configuration parameters which are vectors of primitives (issue #114)
- Expose all configuration parameters from JParameterManager (issue #120)
- Expose ticker and timer state from JApplication (issue #112)
- JEventProcessor::Finalize was being called prematurely when stopping a run via Ctrl-C. (issue #119, issue #87)
- Support factory default tag overrides (issue #128)
- Support JEventProcessors that don't require users to manage locks (pull request #118)

Commits on Aug 1, 2022

Fix 'easy' warnings on gcc11
nathanwbrei committed 16 days ago

Suppress remaining unused parameter warnings
nathanwbrei committed 16 days ago

Commits on Aug 4, 2022

fix: JEventGroupManager is final ···
wdconinc committed 13 days ago ✓

Commits on Aug 6, 2022

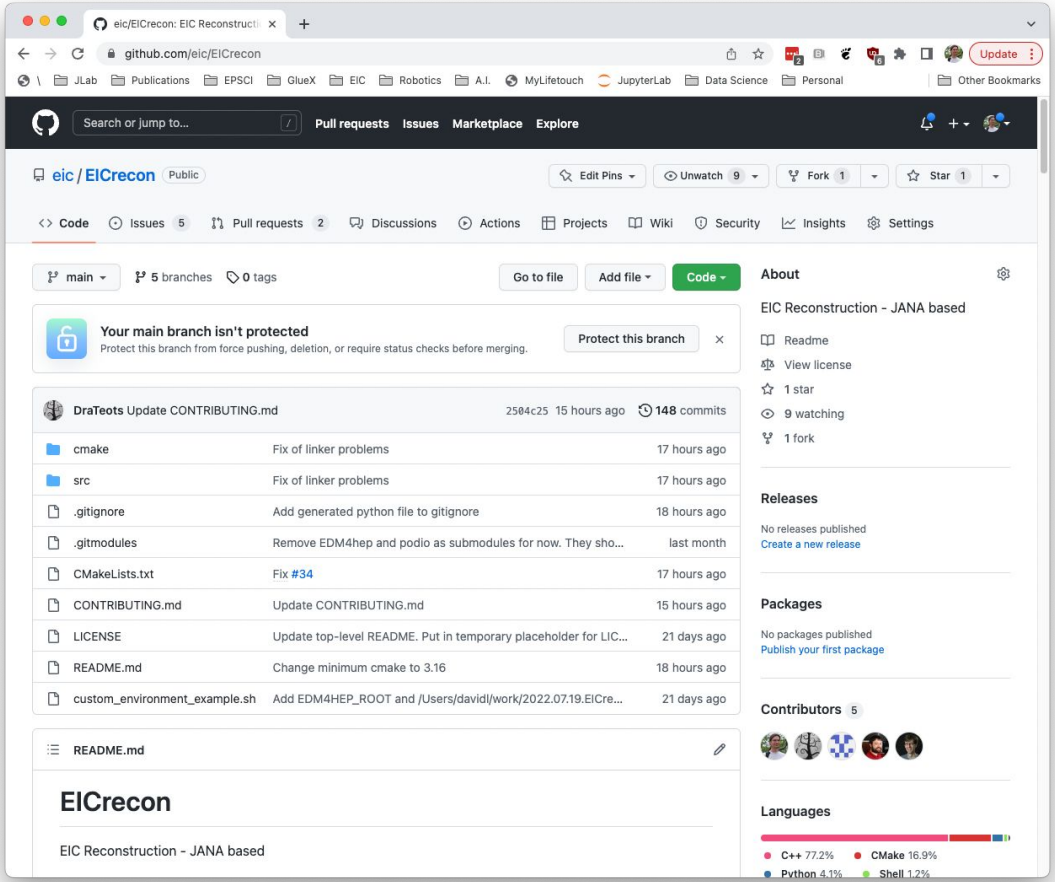
JANAConfig.cmake.in: rework to use the cmake targets ···
veprbl committed 11 days ago ✓

Contributions from non-JLab developers

JEventProcessorSequentialRoot class added to allow users to deal with synchronizing root access in a more familiar way to some

```
115
116
117  #include <JANA/JEventProcessorSequentialRoot.h>
118
119  class DaveTestProcessor: public JEventProcessorSequentialRoot {
120
121    public:
122
123      // These declare object types that should be automatically fetched
124      // from the event before ProcessSequential is called.
125      PrefetchT<Hit>     hits     = {this};
126      PrefetchT<Cluster> clusters = {this, "MyTag"};
127
128      // This will be run sequentially
129       void ProcessSequential(const std::shared_ptr<const JEvent>& event) override {
130
131          // The hits and clusters objects will already be filled by calls
132          // to event->Get(). Just use them here via their operator().
133          for( auto h : hits() ){
134              // h is const Hit*
135          }
136       }
137
138      // Boilerplate stuff
139      DaveTestProcessor() { SetTypeName(NAME_OF_THIS); }
140      void InitWithGlobalRootLock() override {}
141      void FinishWithGlobalRootLock() override {}
142  };
```

EICRecon Github Repository



Top-level README has some extensive build instructions for all necessary dependencies.
(stop-gap until best practices implemented)

README.md

EICrecon

EIC Reconstruction - JANA based

Build Instructions

These are temporary build instructions as the build system and environment setup system needs to be identified. These instructions include building all of the dependencies manually.

Start by setting the EICTOPDIR environment variable. This makes it easier to reference directories in the instructions below. Set this to a directory where you want to build and keep the software. If you wish to use your current directory then just do this:

```
export EICTOPDIR=${PWD}
```

Python Environment

PODIO requires that the python packages *pyyaml* and *jinja2* be installed. If these are not already installed on your system then you can do so either at a system level (requires sudo privilege) or just create a virtual environment:

```
mkdir -p ${EICTOPDIR}/python/virtual_environments
python3 -m venv ${EICTOPDIR}/python/virtual_environments/venv
source ${EICTOPDIR}/python/virtual_environments/venv/bin/activate
pip install pyyaml jinja2
```

boost

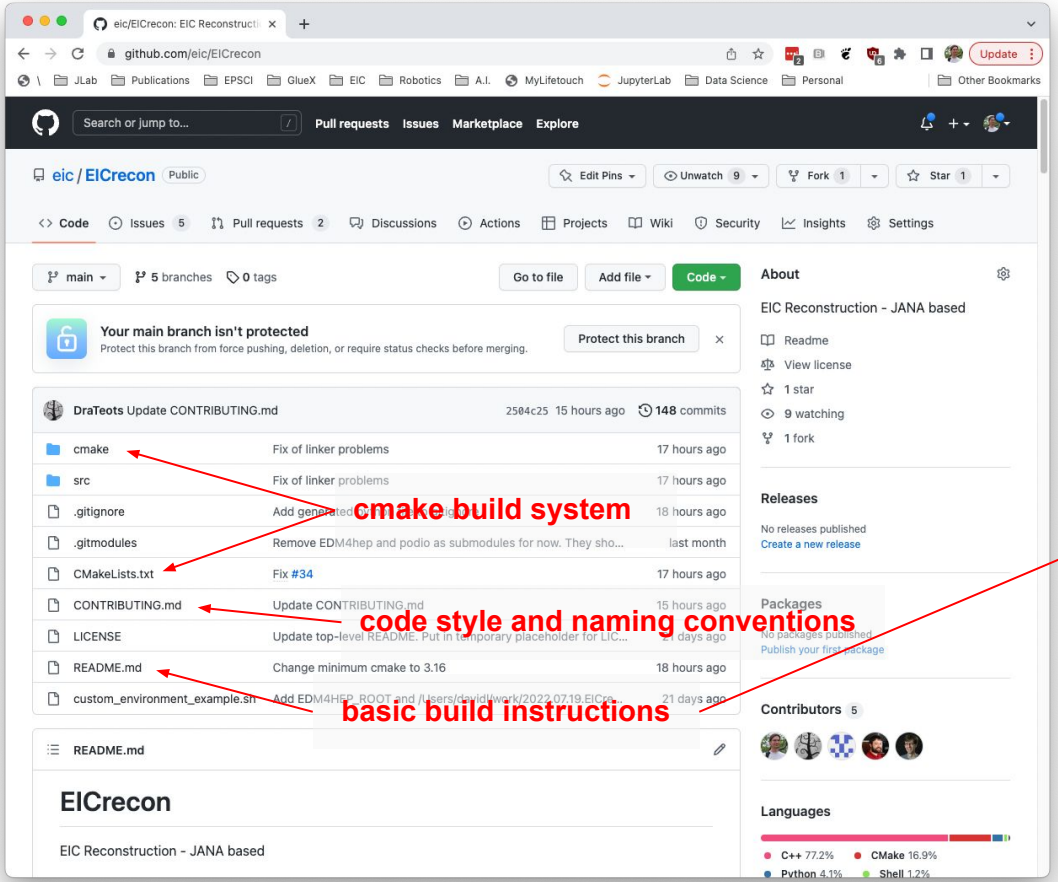
Make sure boost is installed (needed for DD4hep). On macosx 12.4 I did this with:

```
brew install boost
```

On RHEL 7.9 the version installed via yum was too old so I did it from source like this

EICRecon Github Repository

Top-level README has some extensive build instructions for all necessary dependencies.
(stop-gap until best practices implemented)



README.md

EICrecon

EIC Reconstruction - JANA based

Build Instructions

These are temporary build instructions as the build system and environment setup system needs to be identified. These instructions include building all of the dependencies manually.

Start by setting the EICTOPDIR environment variable. This makes it easier to reference directories in the instructions below. Set this to a directory where you want to build and keep the software. If you wish to use your current directory then just do this:

```
export EICTOPDIR=${PWD}
```

Python Environment

PODIO requires that the python packages *pyyaml* and *jinja2* be installed. If these are not already installed on your system then you can do so either at a system level (requires sudo privilege) or just create a virtual environment:

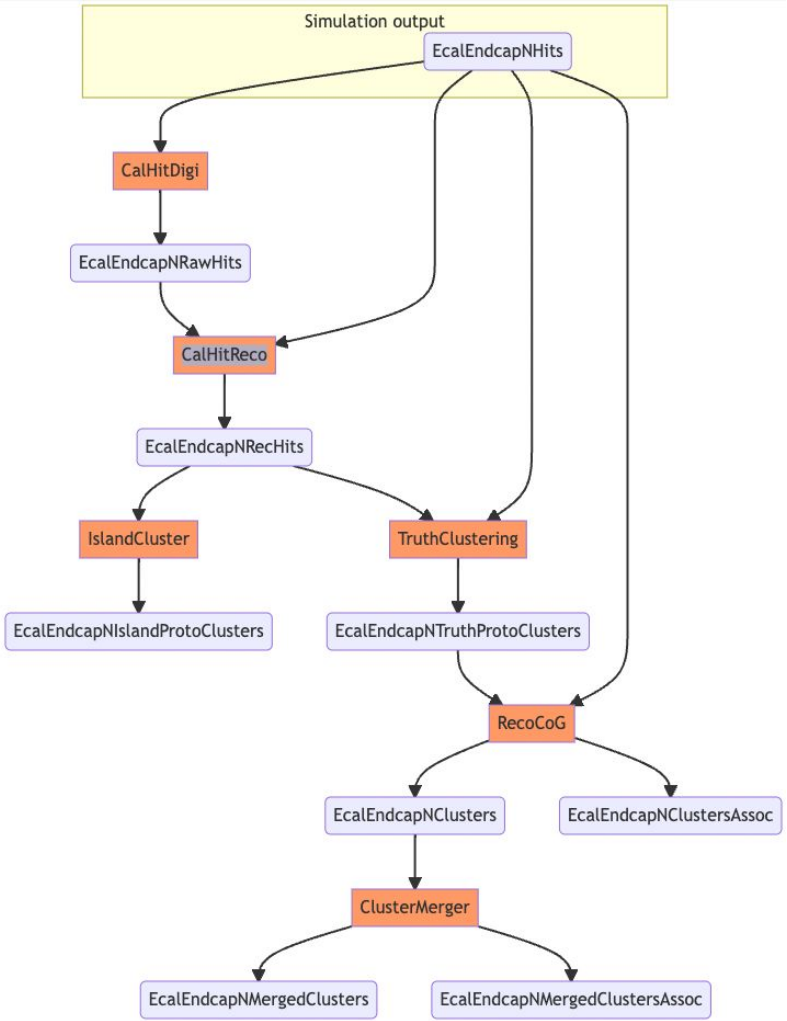
```
mkdir -p ${EICTOPDIR}/python/virtual_environments
python3 -m venv ${EICTOPDIR}/python/virtual_environments/venv
source ${EICTOPDIR}/python/virtual_environments/venv/bin/activate
pip install pyyaml jinja2
```

boost

Make sure boost is installed (needed for DD4hep). On macosx 12.4 I did this with:

```
brew install boost
```

On RHEL 7.9 the version installed via yum was too old so I did it from source like this



Algorithm relationship/data flow diagrams produced from earlier proposal development
(Dmitry Romanov)

Helps guide focus on single algorithm chains for conversion

See EICrecon issue 11:
https://github.com/eic/EICrecon/issues/11

charge(s): 3e

Algorithm Porting

	A	B	C	D
1	**Base package**	**Path**	**Algorithm Name**	**Assignee**
2	Juggler	JugDigi	CalorimeterBirksCorr.cpp	
3	Juggler	JugDigi	CalorimeterHitDigi.cpp	David
4	Juggler	JugDigi	PhotoMultiplierDigi.cpp	Thomas
5	Juggler	JugDigi	SiliconTrackerDigi.cpp	Dmitry
6	Juggler	JugDigi	SimTrackerHitsCollector.cpp	Dmitry
7	Juggler	JugPID	FuzzyKClusters.cpp	
8	Juggler	JugPID	PhotoRingClusters.cpp	
9	Juggler	JugReco	CalorimeterHitReco.cpp	Torri
10	Juggler	JugReco	CalorimeterHitsEtaPhiProjector.cpp	
11	Juggler	JugReco	CalorimeterHitsMerger.cpp	
12	Juggler	JugReco	CalorimeterIslandCluster.cpp	Thomas
13	Juggler	JugReco	ClusterRecoCoG.cpp	
14	Juggler	JugReco	EnergyPositionClusterMerger.cpp	
15	Juggler	JugReco	FarForwardParticles.cpp	Dmitry
16	Juggler	JugReco	FarForwardParticlesOMD.cpp	

All Algorithms | Ecalendcap | Truth level kinematics | Roman Pots | Tracking | Event Building | DRICH

- Porting of O(100) existing algorithms into JANA2 is being coordinated centrally.
- Efforts are being made to port algorithms into generic, framework-agnostic form first.
- JANA factories utilize these generic algorithms

link available on CompSW wiki page

EIC Single Software Stack

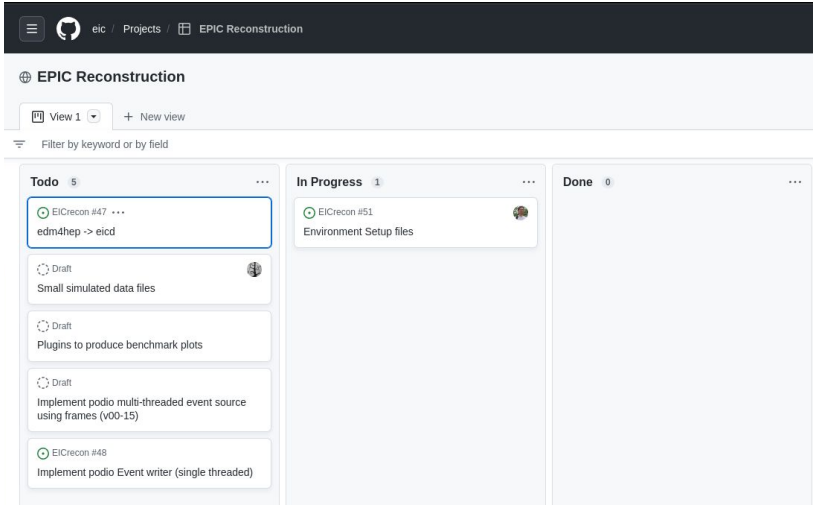
- EIC Single Software Stack
- EICrecon Development
- Algorithm Port List

charge(s): 1b,1e,3e

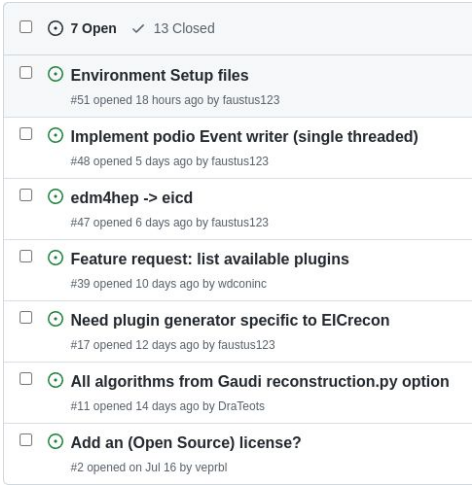
Active use of GitHub Development Tools

EPSCI
EXPERIMENTAL PHYSICS SOFTWARE AND COMPUTING INFRASTRUCTURE

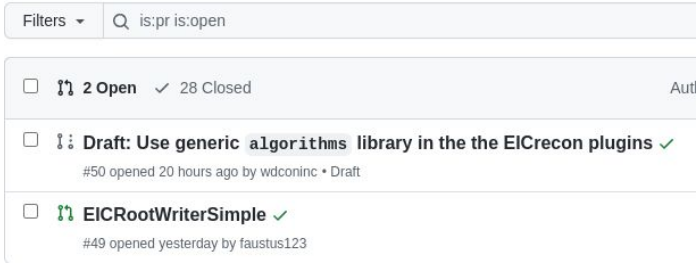
EIC Reconstruction Project



GitHub Issues



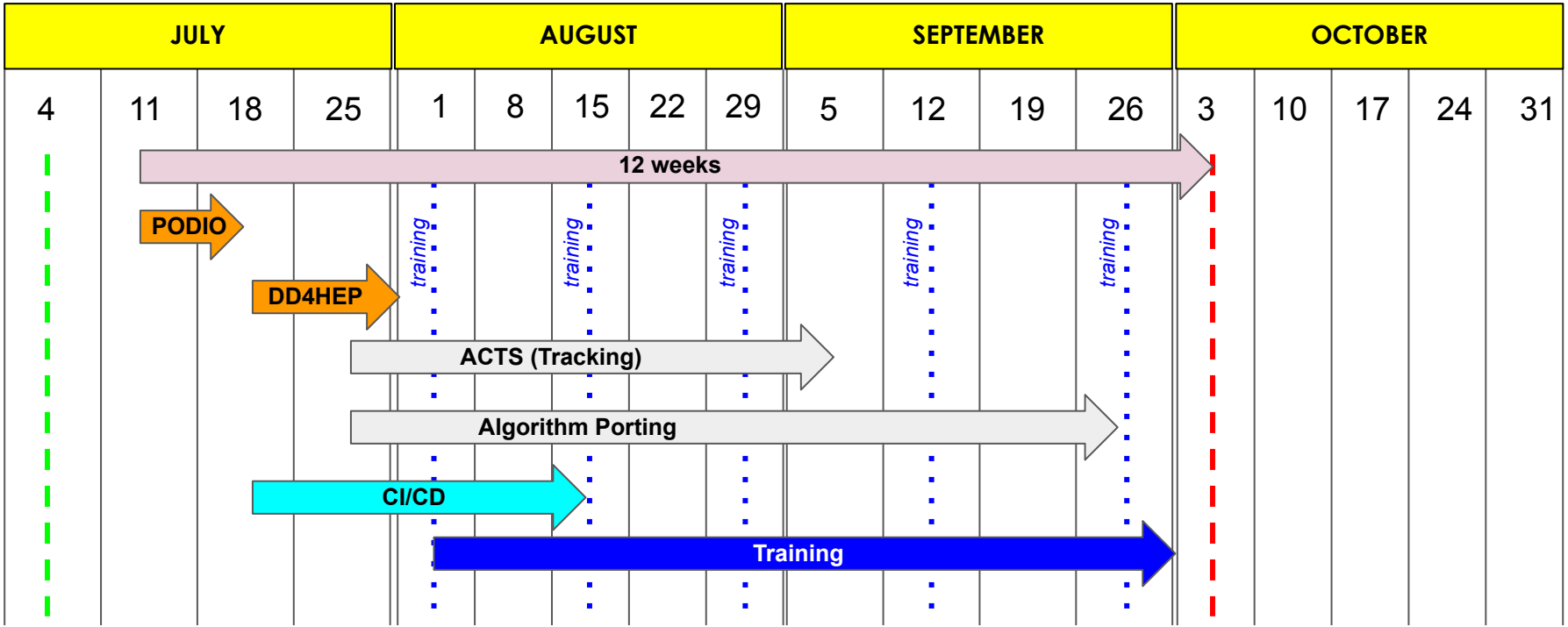
GitHub Pull Requests



charge(s): 3e

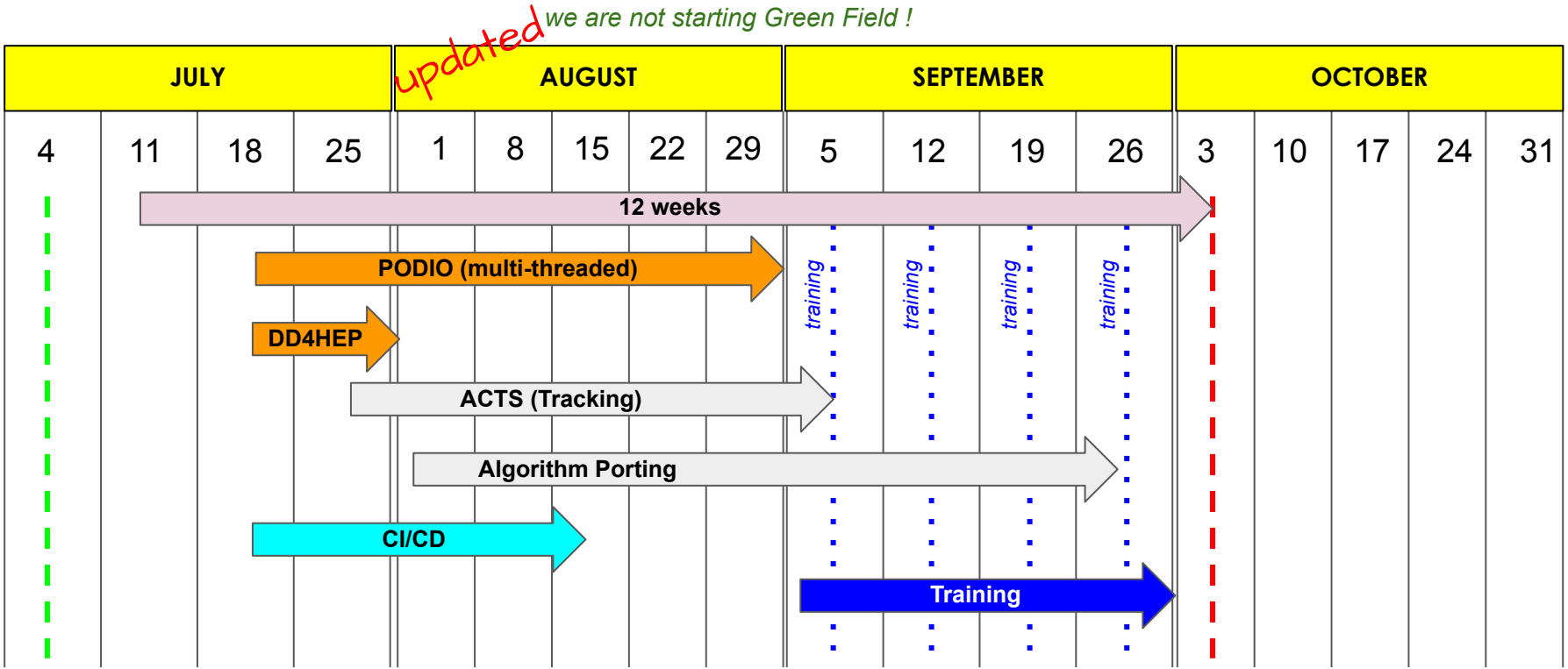
Reconstruction Software Schedule for JANA2

we are not starting Green Field !



charge(s): 3e

Reconstruction Software Schedule for JANA2



3. Institutional Considerations

a. What are the budgetary implications for the host labs if any? What is the cost of both implementing a software solution and maintaining it for the lifetime of the experiment?
JLab has committed 1 FTE/yr to maintain JANA2 and contribute to the development and maintenance of EIC reconstruction software

b. What are the risks incurred by the host Labs with the proposed solution?
If software/hardware technologies shift significantly over the next decade in such a way that 1 FTE is insufficient to respond, additional staffing will be required.

c. If external institutions are involved, is there a reasonable expectation of an institutional commitment that includes succession planning? Is this an acceptable contribution to the experiment?
The framework itself is expected to have primary institutional responsibility with JLab. This will be true even if a significant contribution to the framework itself is made by an external institution who then becomes unable to maintain that feature.

d. Is the proposed solution compatible with a potential second detector? (This is a requirement)
Yes.

e. What is the mechanism for reviewing and monitoring timeline for deliverables and evolution of functionalities of the software solution?
Development of algorithms using JANA2 is already underway and there are no pending features in the framework itself that are delaying that work. Regarding the reconstruction software itself (e.g. algorithms): this has yet to be determined. The immediate timeline is known regarding TDR development in preparation for CD2/3. In addition to an internal schedule that will be better developed after the formal formation of the EPIC collaboration, we anticipate a regular review schedule from the Project.

Summary

- A structured decision process was used to select JANA2 as the reconstruction framework based on a carefully formed set of requirements reviewed by the EIC software community
  - A commitment by JLab to support JANA2 throughout the lifetime of the EIC was a key factor in the decision
- JANA2 is maintained on GitHub and uses modern software tools for development
  - *Cmake, GitHub Issues, Pull Requests, formal releases, multi-platform C.I. builds, …*
- Development of the EICrecon software is well underway starting with porting of the relevant algorithms already developed for EIC detector proposals
  - Currently on schedule to meet Oct. deadline for first major simulation campaign with the Single Software Stack using JANA2
- GlueX is the highest volume NP experiment ever run at JLab to date and has been successfully using JANA for several years.
  - Fast turnaround to publications using JANA

Backups

Requirements

- *The reconstruction framework must be able to run on both simulated events and real data. Even if there may be algorithms that use truth information (or even require truth information, initially), the reconstruction framework itself should allow for running without truth information.*
  JANA's ***factory tag*** mechanism can be used to tag "TRUTH" versions of objects. The tagged versions of objects may be requested programmatically or on a global scale at runtime via configuration parameters. Both the TRUTH tagged and the un-tagged versions of the objects may coexist.

- *The reconstruction framework must be able to take advantage of heterogeneous computing resources (multiple cores, GPUs, etc).*
  JANA's main purpose for existence was to provide multi-threaded event reconstruction and the entire design of the framework grows from that. Sub-tasks were added in JANA2 specifically to add additional heterogeneous support.

- *The reconstruction framework must encourage modular approaches to algorithm development, using defined interface layers.*
  JANA has a set of base classes that define the interface. Design emphasizes a factory having one primary class of object as its output encouraging users to implement a more modular design. e.g. Track seeds can be produced in one factory and fully fit tracks in another allowing the seed finding algorithm to be easily swapped. The framework also allows for both types of objects to be produced in a single factory, but the current JANA2 design encourages the developer to break that up into smaller modules instead.

Requirements

- *Algorithms must be implemented using the selected data model, and ensure that data (event data, geometry description, and algorithm parameters) are kept separate from the algorithm itself.*
  JANA supports this style of programming. The algorithm parameters (formally *Configuration Parameters* in JANA) can be set via config file or command line argument and a centrally available to all factories. Furthermore, the implementation allows new configuration parameters to be easily deep in a factory's user code, yet still be accessible to all JANA objects.

  The event data is managed by the framework. Geometry description is provided by a JANA Service that gives access to the underlying geometry package (e.g. DD4hep).

- *Algorithms must be implemented in the framework independently from any scheduling strategies; an algorithm must not need to know how it is orchestrated, whether it is running in parallel, in single or multithreaded mode, concurrent or not, in online or offline analysis mode.*
  JANA algorithms are ignorant of this type of information which is handled at the framework level.

- *The reconstruction framework must be open source, accessible to the entire community, and managed by a sustainable core team.*
  JLab has committed to support JANA throughout the EIC project as a full partner lab. The source is freely available from GitHub. Any Github user in the world is able to submit issues and PRs to the JANA2 repository.

Requirements

- *The reconstruction framework must be able to pass (and add) metadata and so-called slow control information to the output files, so input files are not needed and output files can stand on their own.*
  JANA allows objects of any type to be inserted into an event. Any output file writing would need to rely on tools that interface with the Data Model and so are not explicitly part of the framework itself.

- *The reconstruction framework must be able to run in streaming readout mode, that is:*
  - *with access to only parts of an event (single detector, single sector),*
  - *with events (or parts of events) appearing out of sequence,*
  - *individual algorithms must not rely on an algorithm-specific internal state to be able to make sense of disconnected parts of events.*

  JANA's Queue/Arrow architecture supports streaming at multiple levels. In particular, it can support one to many, or reordering algorithms in a natural way. The on demand design naturally supports processing of partial events. This is an extremely common exercise in GlueX.

Additional assessment criteria

- *Amount of 'boilerplate' code that must be written by algorithm developers.*
  The ***jana-generate.py*** script generates the boilerplate code based on single or a few inputs. This includes making a complete stand-alone plugin with CMakeLists.txt file. This makes it very easy to add new components quickly.

- *Ability by the framework to avoid e.g. memory errors through interface enforcement mechanisms (e.g. const passing).*
  JANA passes pointers to const objects between factories and processors. This is required for reproducibility should the order of factory calls be changed between program invocations.

- *Ability for shared algorithm development between the two EIC detector collaborations (and/or outside of the EIC).*
  JANA factories are self contained in that they request objects and publish objects via the framework. Any detector collaboration using the same input and output classes will be portable/sharable. Furthermore, the plugin mechanism allows a plugin to provide one or more factories to any JANA executable. Thus, a single pre-compiled plugin can be used in multiple experiments.

- *Use of modern and sustainable coding practices, including in the code written by algorithm developers and other contributors.*
  JANA is maintained in a Github repository. The issues, pull requests, and release mechanisms are used to maintain the code. Automated builds and unit tests on multiple platforms are initiated by pull requests.

- *Demonstration of performance in production environments.*
  GlueX.

Convener Summary

The working group conveners recommend JANA2 as the reconstruction framework.

Although both Gaudi and JANA2 are technically able to meet the requirements, there is too much risk in depending on Gaudi which is not focused on a community outside of LHCb. The efforts invested in the already written juggler algorithms will be able to be reused with relatively minor effort in JANA2 algorithms. Likewise, the efforts invested in fun4all algorithms may be able to be reused.

The translation of the relevant Gaudi/juggler and fun4all algorithms will be completed by the Jefferson Lab EPSCI group by October 2022, aided by the 1 FTE-year per year committed by Jefferson Lab to the support of JANA2 for EIC.

The working group conveners point out that continued engagement with the key4HEP project through the development of modular reconstruction algorithms and functional programming approaches is desirable.

The Jefferson Lab EPSCI group is encouraged to develop JANA2 into a community project where developers from outside Jefferson Lab are valued at all stages of software development.

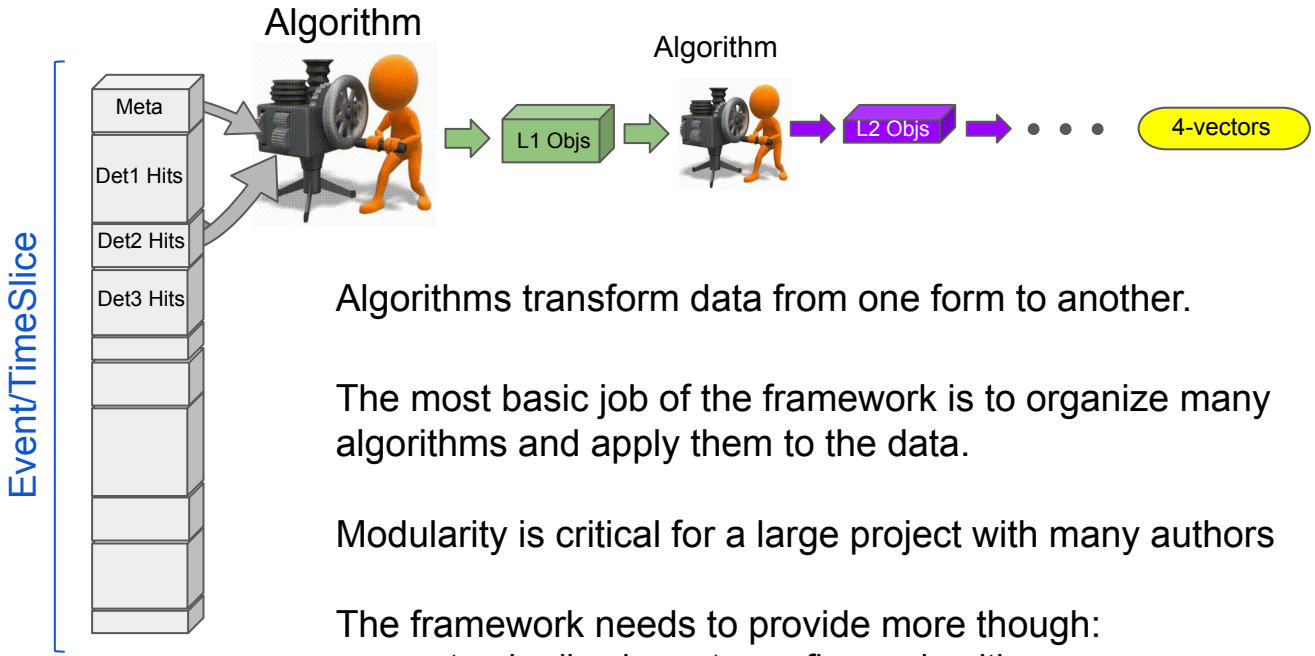
First example Gaudi to JANA algorithm port

https://eicweb.phy.anl.gov/EIC/juggler/-/blob/master/JugDigi/src/components/CalorimeterHitDigi.cpp

https://github.com/eic/EICrecon/blob/main/src/detectors/BEMC/CalorimeterHitDigi.cc

- Core algorithm simply cut and pasted
- Input/Output objects and configuration parameters converted from Gaudi-speak to JANA-speak
- *CalorimeterHitDigi* class is itself JANA agnostic
  - Specialization done by *JFactory_BEMCRawCalorimeterHit* class which can use CalorimeterHitDigi in two ways:
    - Inherited (isA relation)
    - Member (hasA relation)

Purpose of the “framework”



Algorithms transform data from one form to another.

The most basic job of the framework is to organize many algorithms and apply them to the data.

Modularity is critical for a large project with many authors

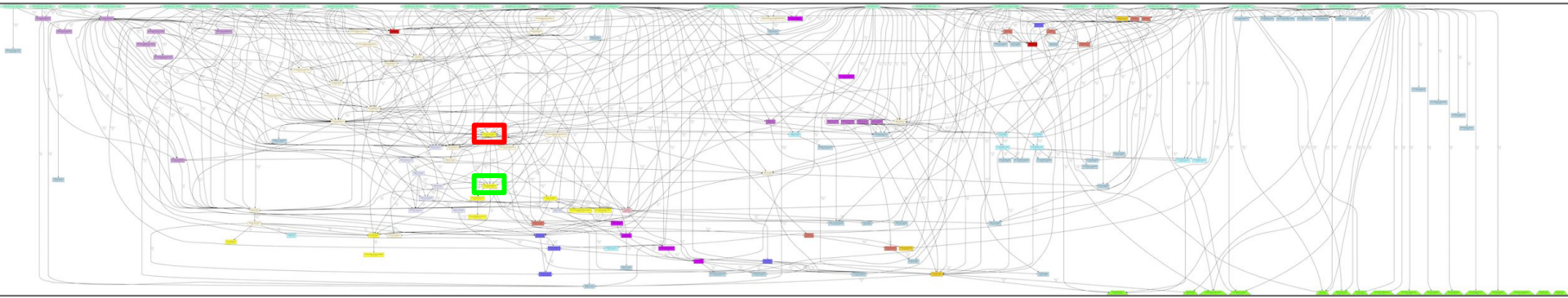
The framework needs to provide more though:

- standardized way to configure algorithms
- standardized control over local resources (CPUS, GPUS)
- Geometry, calibration, alignment, … services

Large experiments have complex call graphs

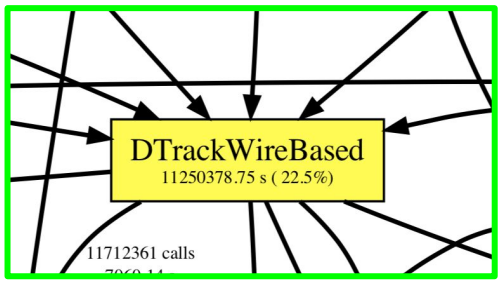
EPSCI
EXPERIMENTAL PHYSICS SOFTWARE AND COMPUTING INFRASTRUCTURE

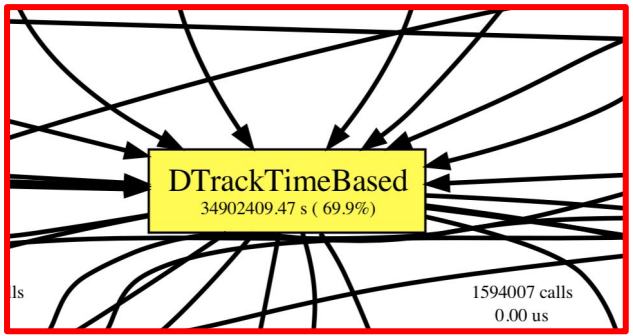
GlueX Reconstruction - automated rendering via janadot plugin



Run 42513:

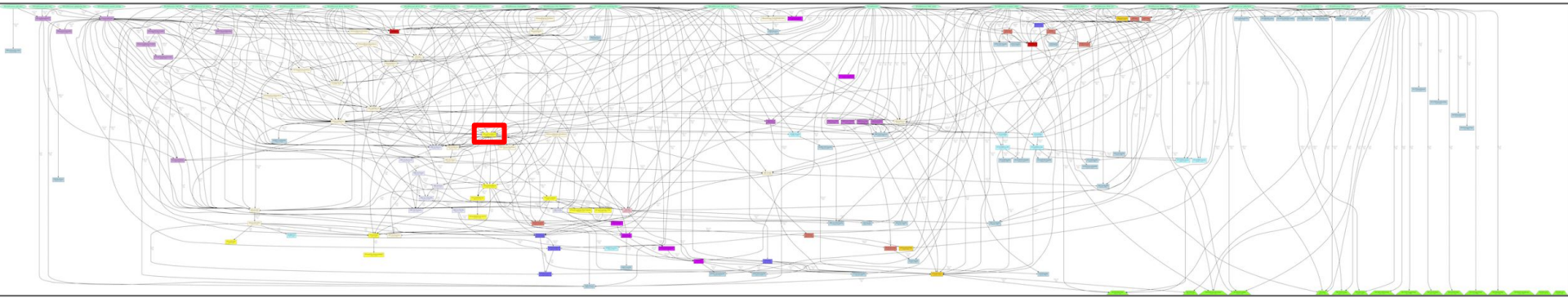
Physics Production mode Trigger: FCAL_BCAL_PS_m9.conf
setup: hd_all.tsg
0/90 PERP 90
JD70-100 58um
TPOL Be 75um
beam looks stable





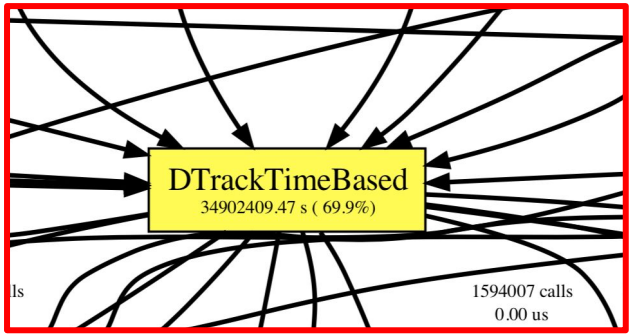
Large experiments have complex call graphs

GlueX Reconstruction - automated rendering via janadot plugin



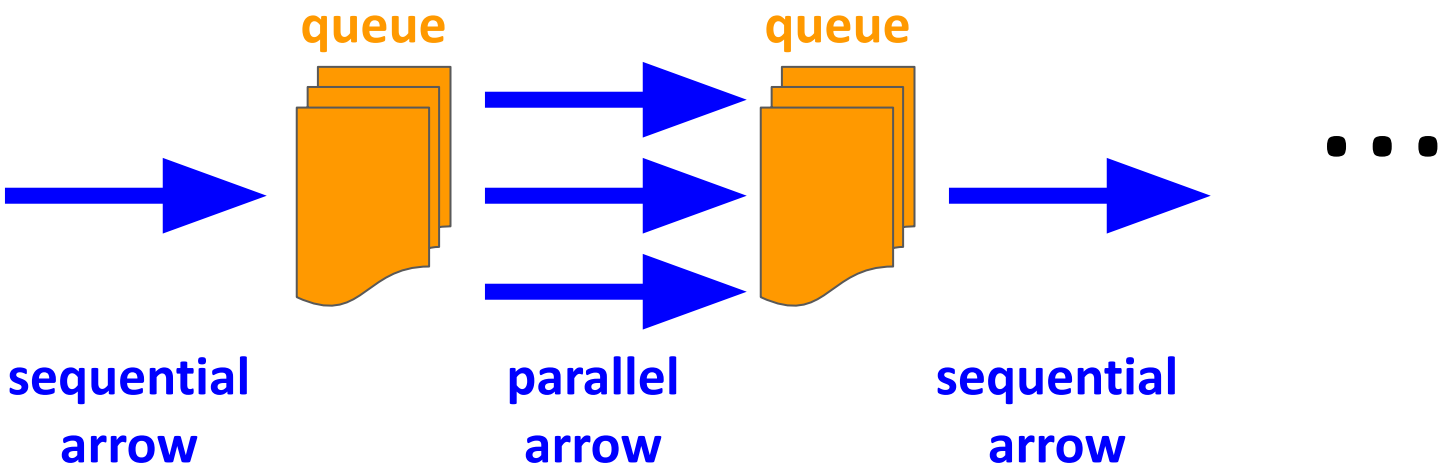
Modular design:

- Factories (algorithms) need to know what they depend on
- Factories do ***not*** need to know what depends on them
- Dependencies do ***not*** need to be specified at higher level

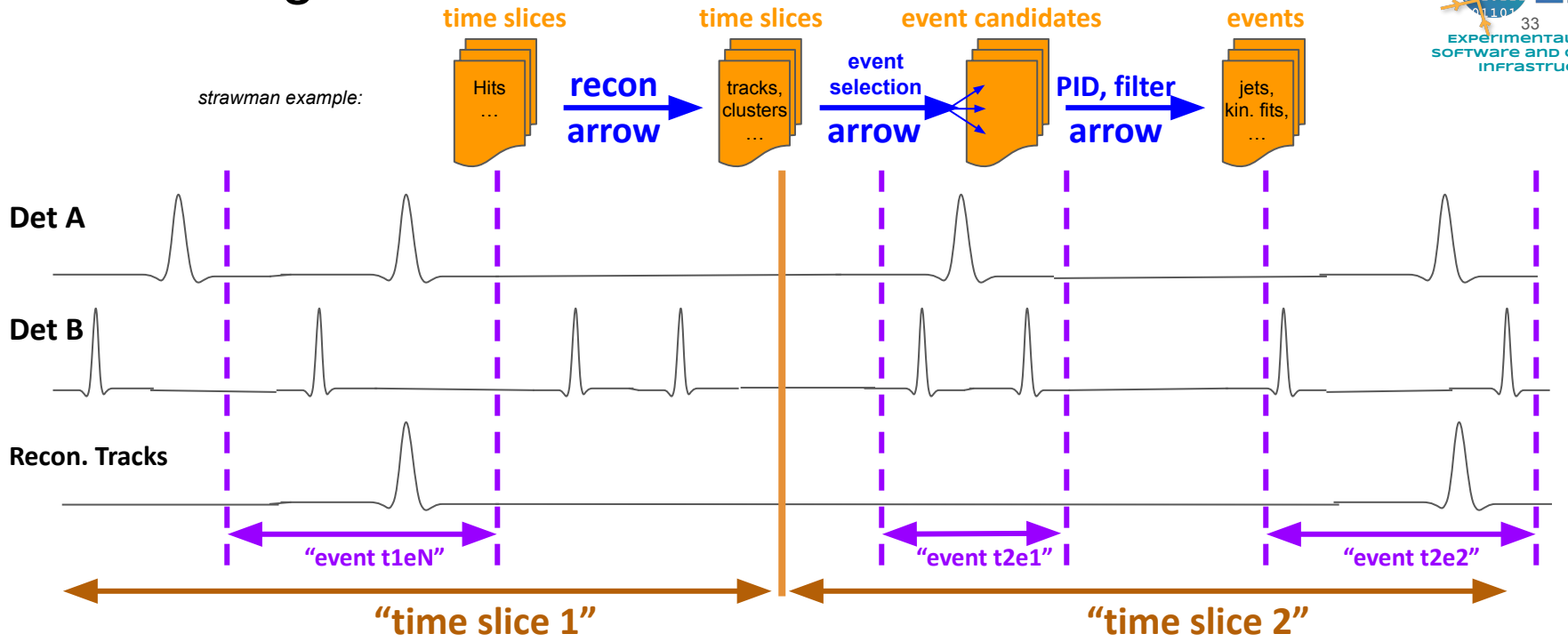


JANA2 arrows separate Sequential and Parallel tasks

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

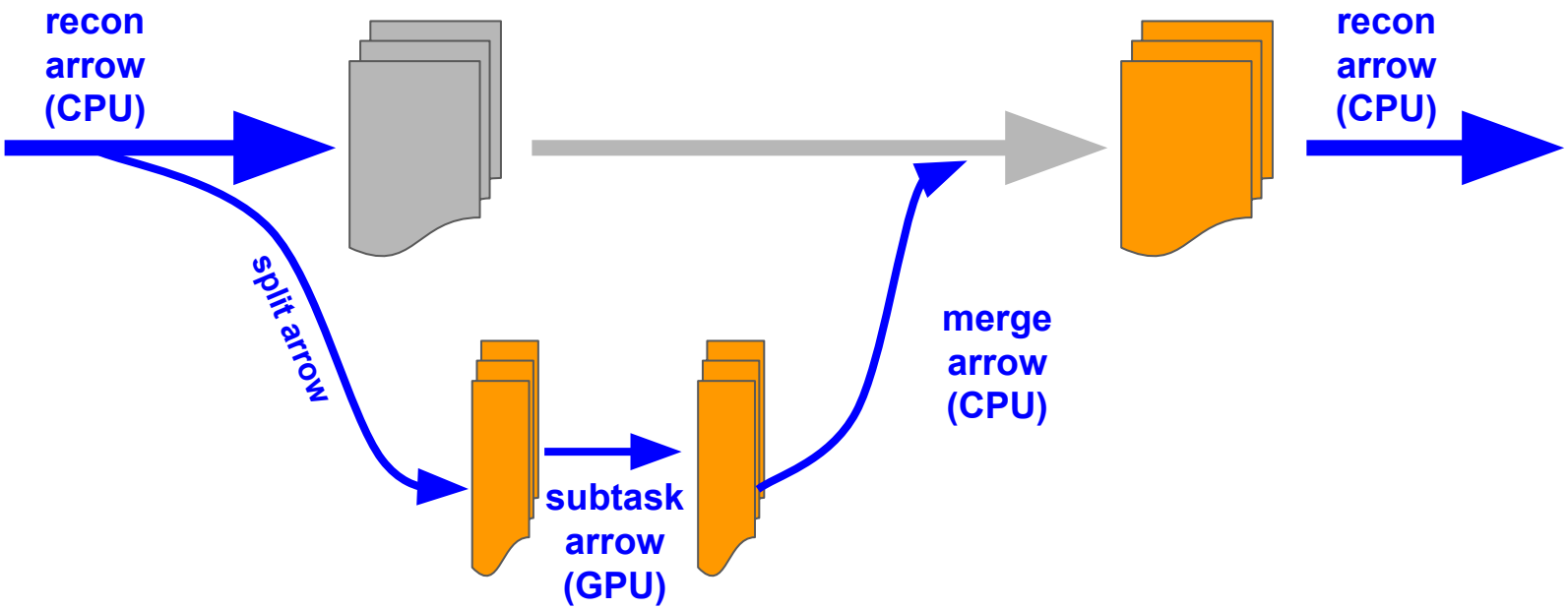


Streaming Data

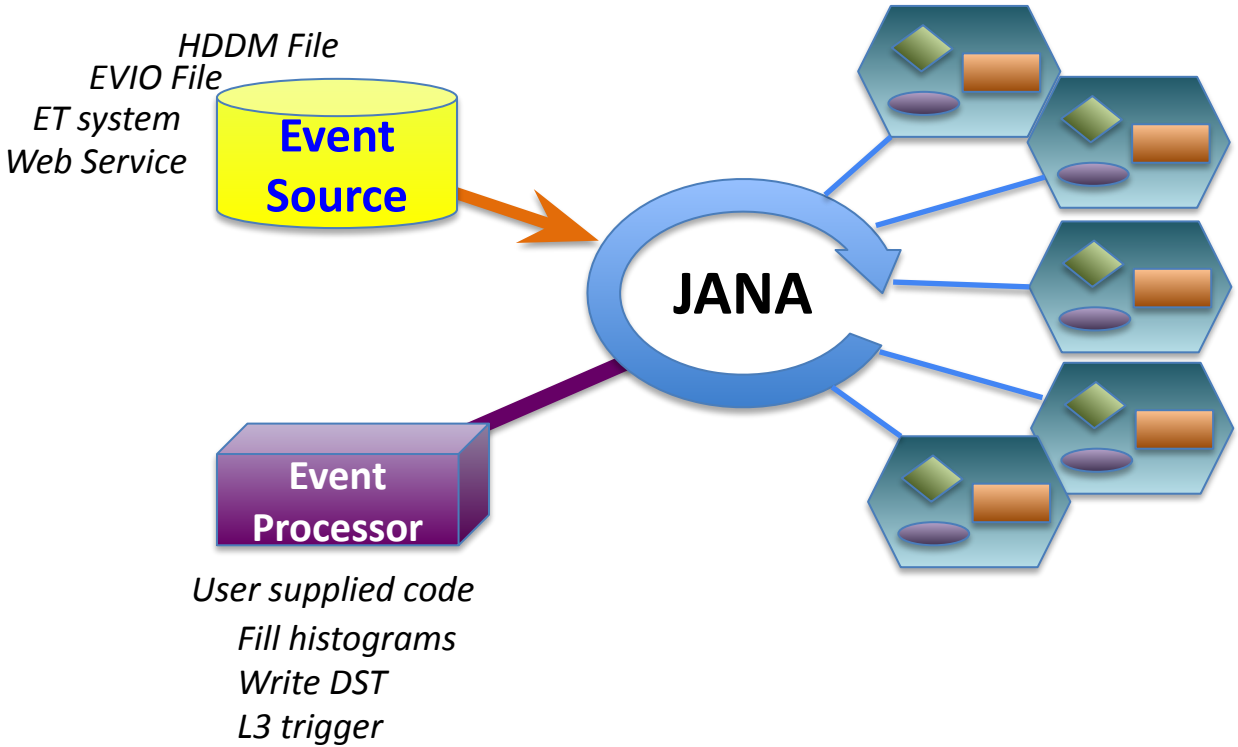


- Stream comes in the form of large time slices which may contain many events
- Arrow/queue system naturally supports one-to-many transformations
- Used in (ERSAP+) TriDAS + JANA2 system in Hall-B/Hall-D
- Used in INDRA-ASTRA AI/ML near-realtime calibration project

Heterogeneous Hardware Support



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

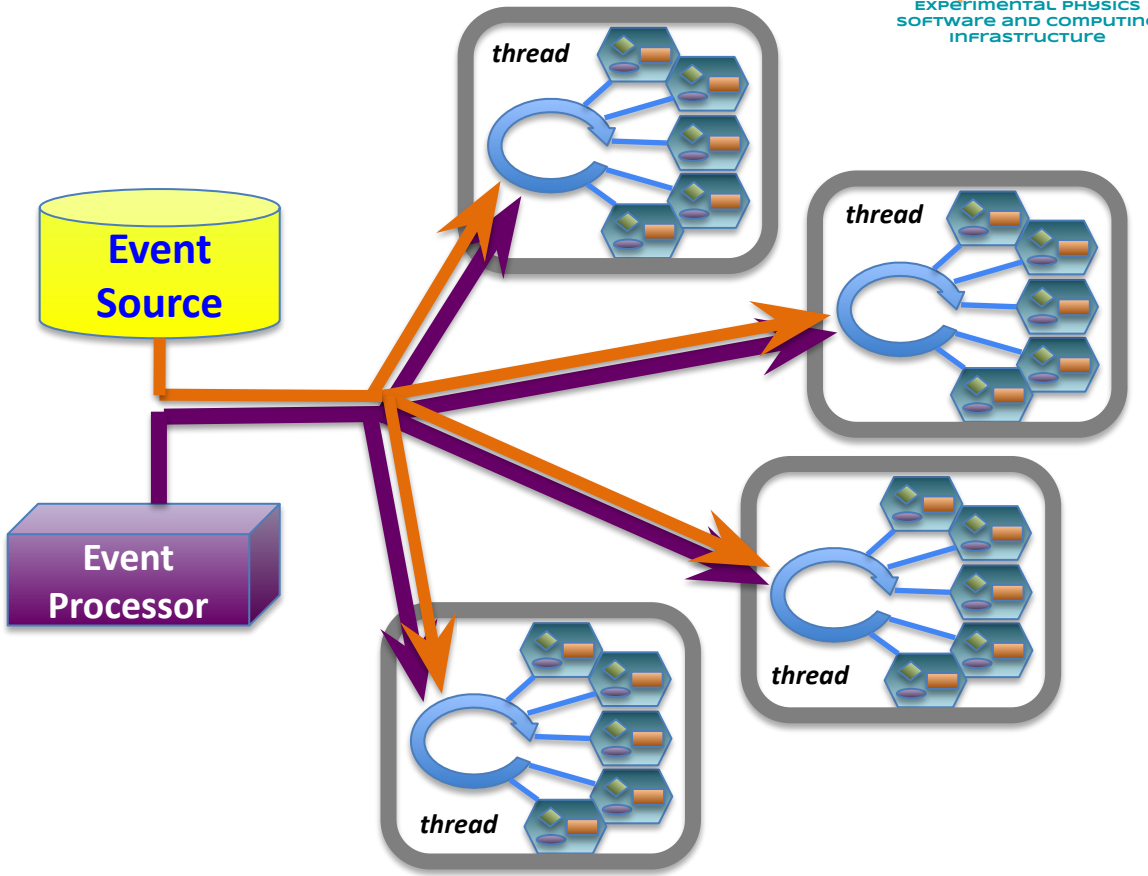
This allows the framework to easily redirect requests to alternate algorithms specified by the user at run time

Multi-threading

- *Each thread has a complete set of factories making it capable of completely reconstructing a single event/slice*

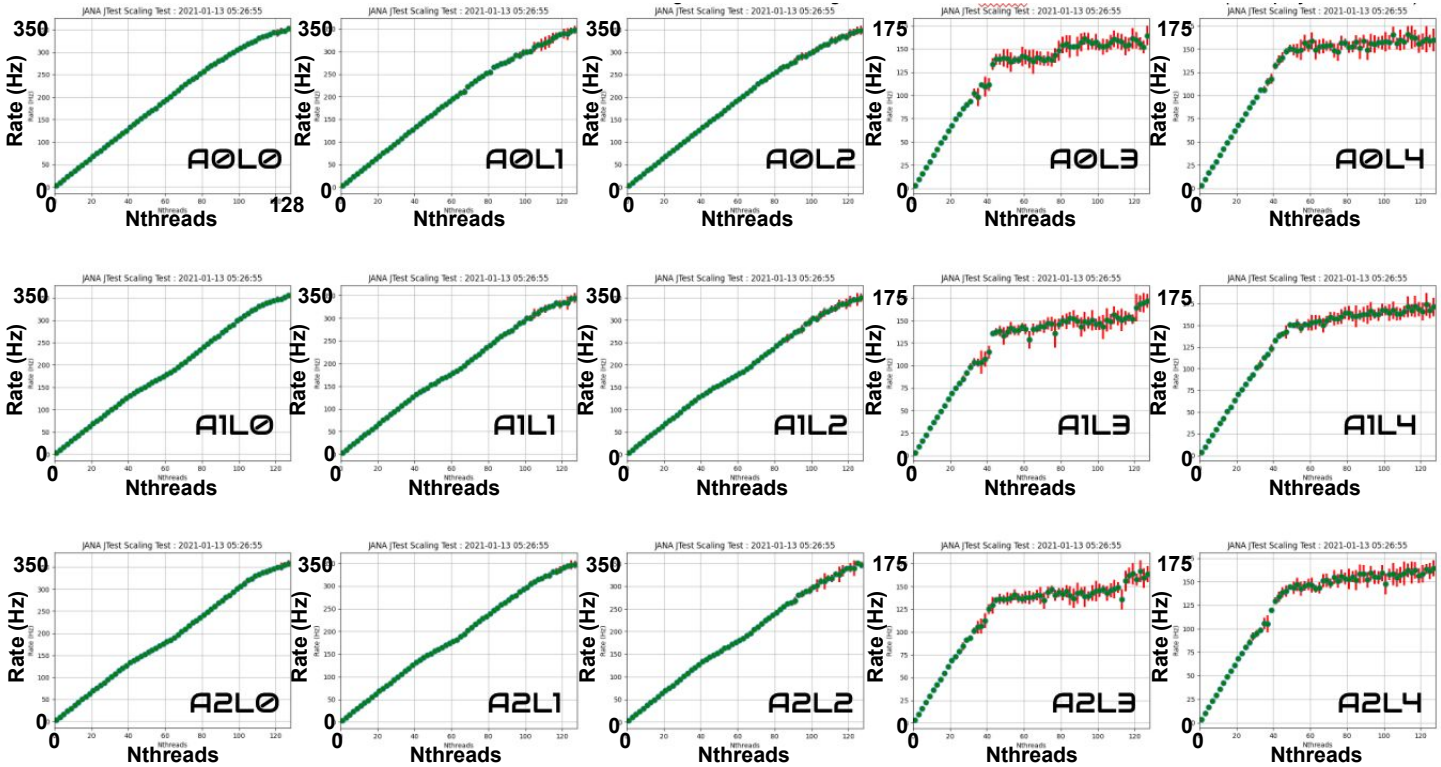
- *Factories only work with other factories in the same thread **eliminating the need for expensive mutex locking** within the factories*

- *All events are seen by all Event Processors (multiple processors can exist in a program)*



Multiple Affinity and Locality strategies

OS, chip type, memory architecture, and nature of job all can affect which model yields optimal performance



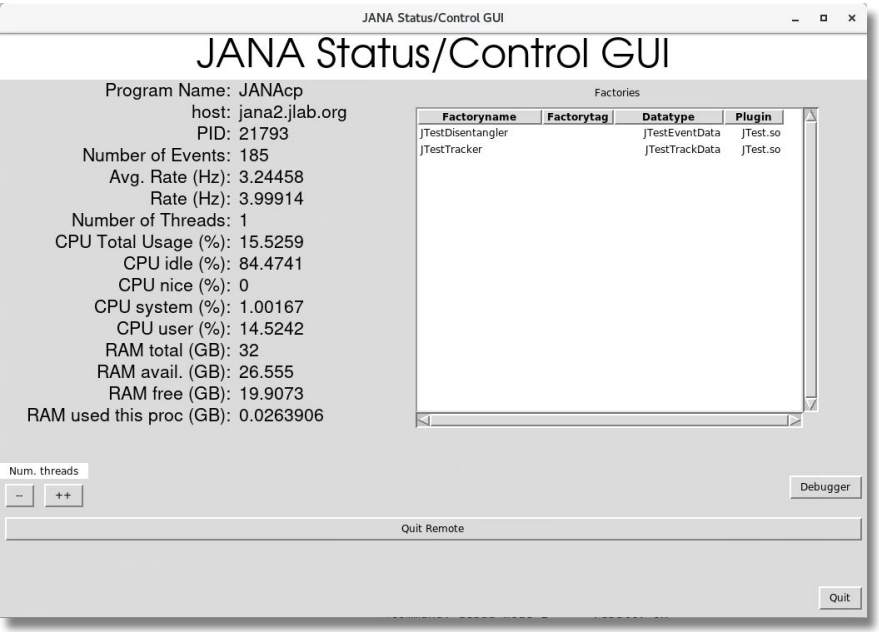

```
enum class
AffinityStrategy {
      None,
      MemoryBound,
      ComputeBound };
```

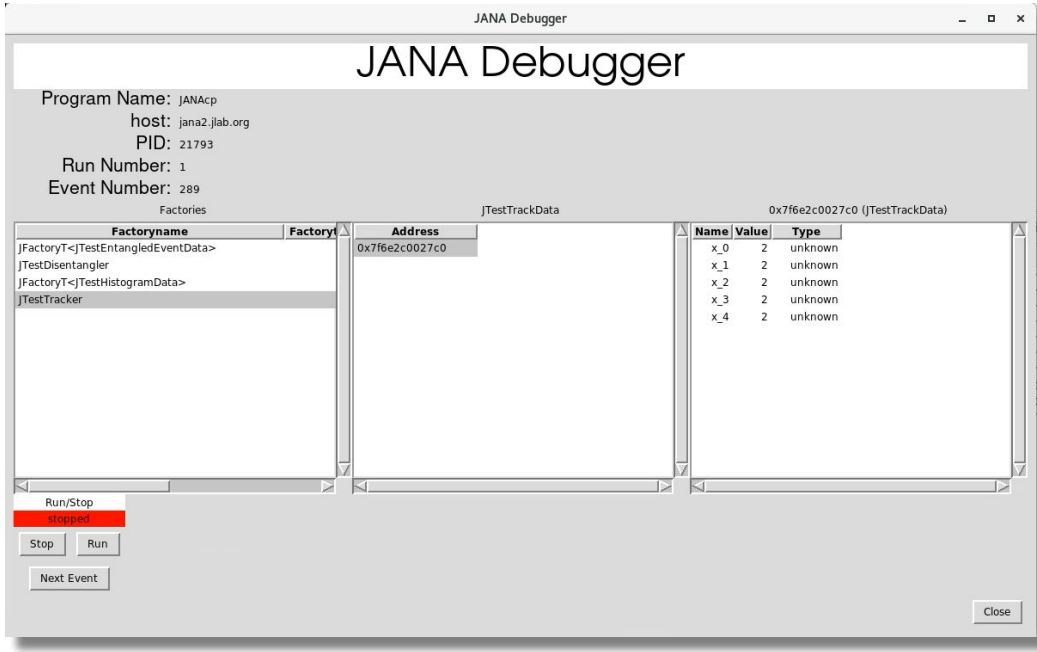
```
enum class
LocalityStrategy {
      Global,
      SocketLocal,
      NumaDomainLocal,
      CoreLocal,
      CpuLocal };
```

Configurable at run time via Config. Parameters

JANA2 Scaling test: PSC Bridges-2 RM Two AMD EPYC 7742 CPUS (128 physical cores)

Inspection Tools





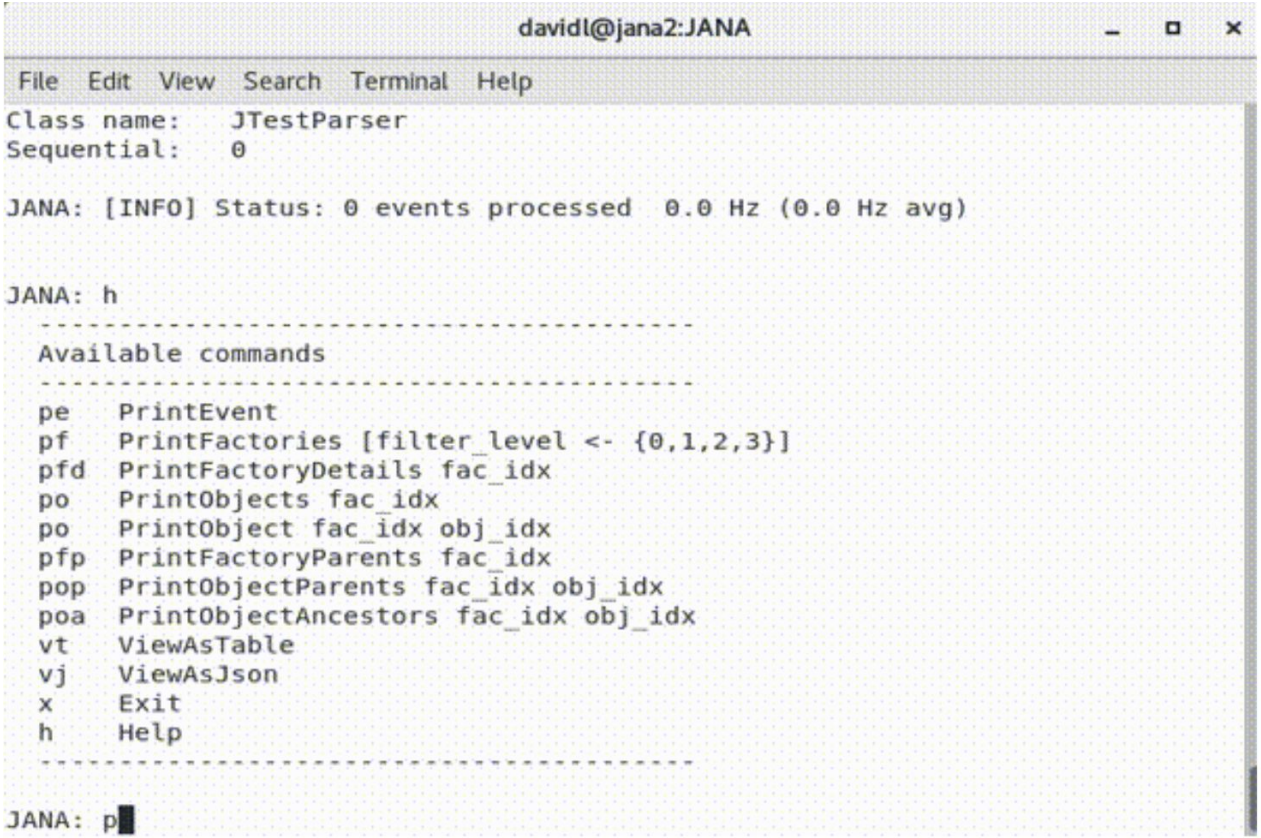

```
> jana -Pplugins=JTest,janacontrol
```

Add *janacontrol* plugin to any process

```
> jana-control.py [--host host] [--port port]
```

Run GUI from remote (or same) node

JANA Command Line Debugging w/ gdb

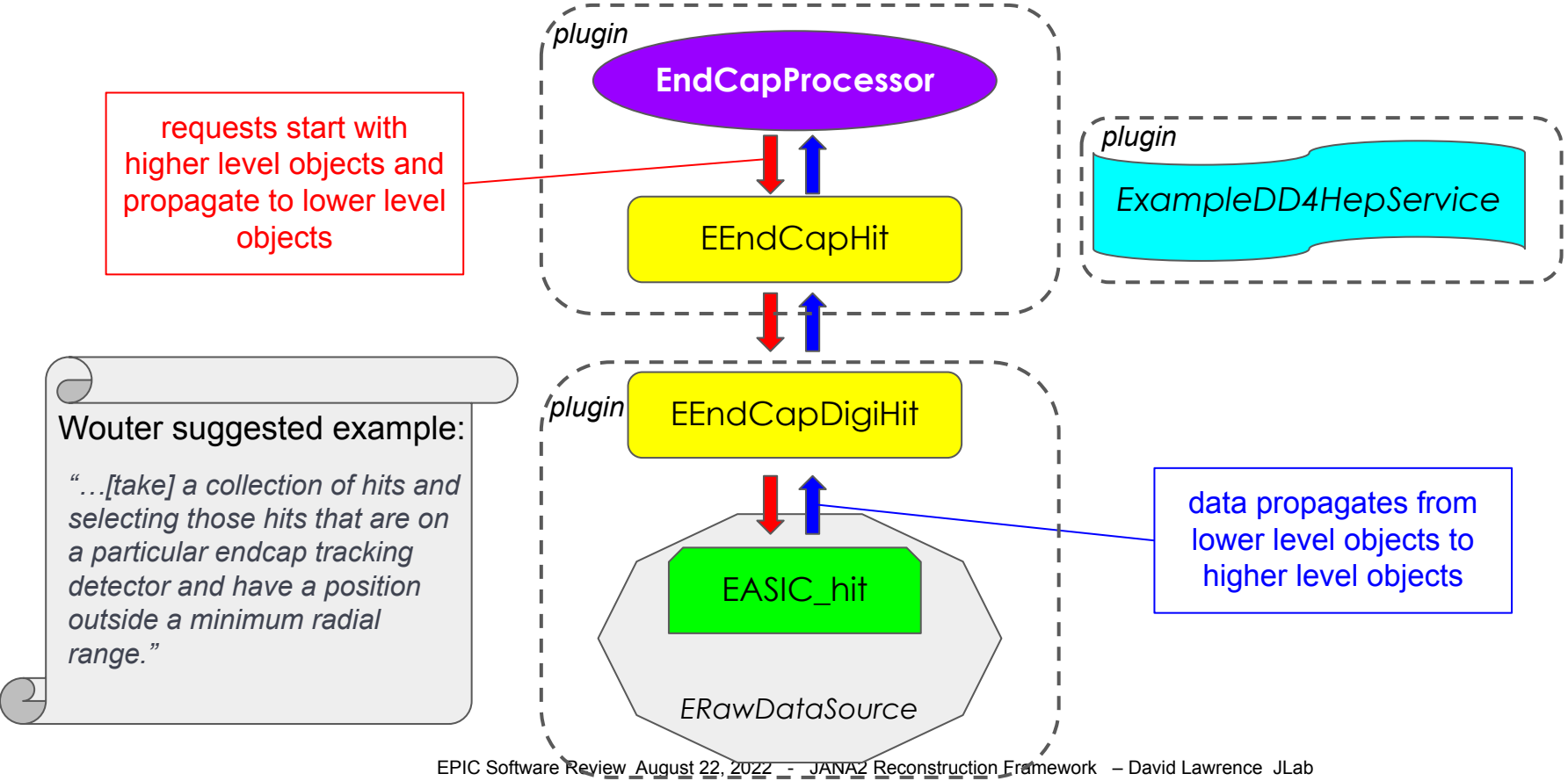


Certain JANA methods are written with the intention of being called from debugger.

This allows easier browsing from the framework point of of view.

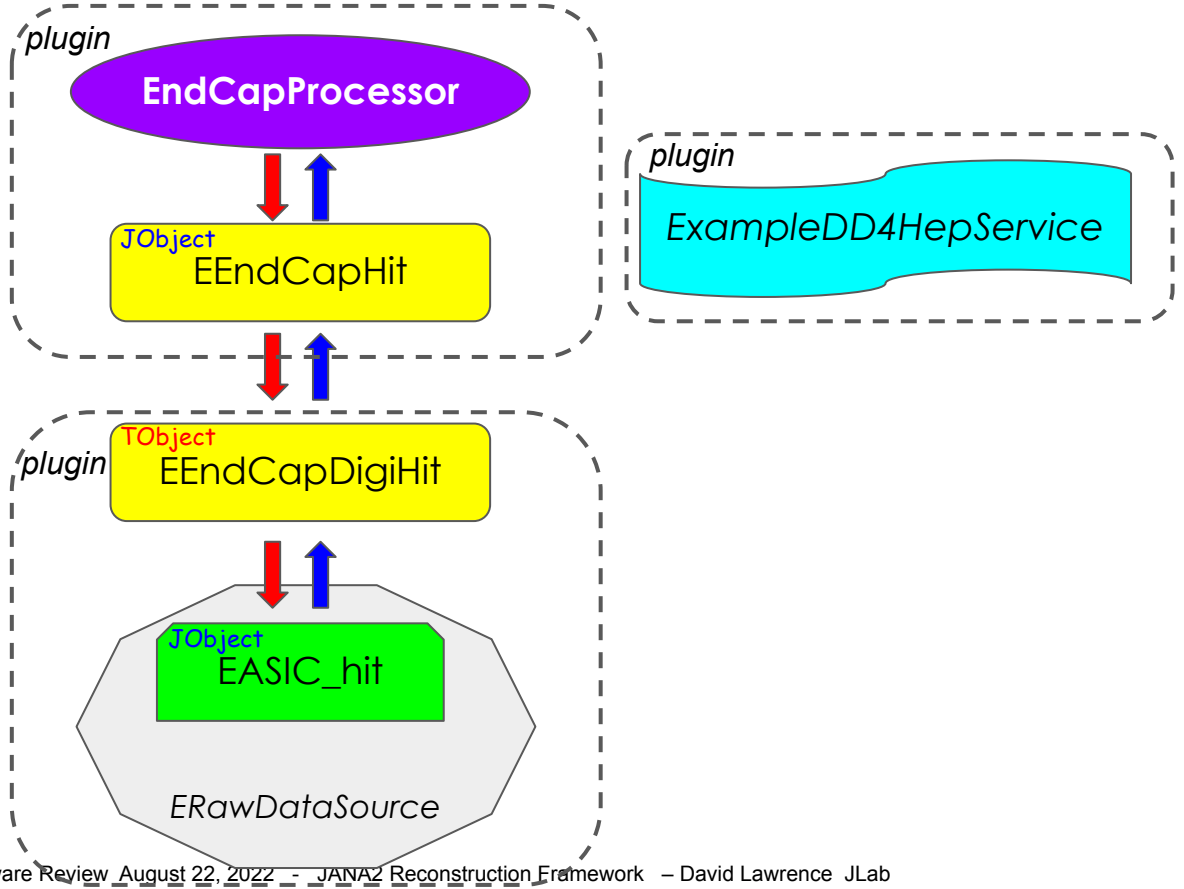
Example with Geometry Service

https://github.com/faustus123/EIC_JANA_Example



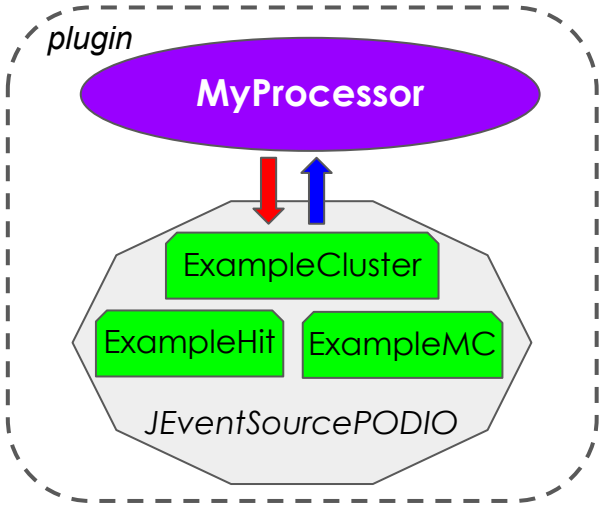
Example with Mixed TObject and JObject

https://github.com/faustus123/EIC_JANA_Example/tree/TObject_example



Example with PODIO Event Source

https://github.com/faustus123/EIC_JANA_Example/tree/PODIO_example



ALL EXAMPLES

- *detailed build instructions accompany each example*
- *extensive comments in code*

Example implements a JANA event source that can read from the *example.root* file produced by PODIO *tests/write* program.

n.b. ***objects are still owned by PODIO EventStore***

JFactory_EEndCapHit

```
2  #ifndef _JFactory_EEndCapHit_h_
3  #define _JFactory_EEndCapHit_h_
4
5  #include <JANA/JFactoryT.h>
6  #include <ExampleDD4HepService/ExampleDD4HepService.h>
7  #include "EEndCapHit.h"
8
9  class JFactory_EEndCapHit : public JFactoryT<EEndCapHit> {
10
11     // Insert any member variables here
12
13 public:
14     JFactory_EEndCapHit();
15     void Init() override;
16     void ChangeRun(const std::shared_ptr<const JEvent> &event) override;
17     void Process(const std::shared_ptr<const JEvent> &event) override;
18
19 protected:
20     double min_radius;
21
22     const ExampleDD4HepService *geomservice=nullptr;
23
24 };
25
26 #endif // _JFactory_EEndCapHit_h_
```

```
24 void JFactory_EEndCapHit::Init() {
25     auto app = GetApplication();
26
27     // Just for fun, create a configuration parameter named
28     // EndCap:min_radius so we can set the threshold at run time.
29     min_radius = 15.0;
30     app->SetDefaultParameter("EndCap:min_radius", min_radius, "The mini
31
32     /// Acquire geometry service pointer (see ExampleDD4HepService plugin)
33     geomservice = app->GetService<ExampleDD4HepService>().get();
34 }
```

boilerplate

added for this example

JFactory_EEndCapHit::Process

```
44  //----------------------
45  // Process
46  //----------------------
47  void JFactory_EEndCapHit::Process(const std::shared_ptr<const JEvent> &event) {
48
49      /// JFactories are local to a thread, so we are free to access and modify
50      /// member variables here. However, be aware that events are _scattered_ to
51      /// different JFactory instances, not _broadcast_: this means that JFactory
52      /// instances only see _some_ of the events.
53
54      // The EEndCapDigiHit objects are made by a factory in the EICRawData plugin.
55      // That factory uses the low-level EASIC_hit objects coming from the event source
56      auto endcapdigihits = event->Get<EEndCapDigiHit>();
57
58      // Loop over the EEndCapDigiHit objects and create calibrated hits
59      // objects with geometry info.
60      std::vector<EEndCapHit *> hits;
61      for( auto digihit : endcapdigihits ){
62
63          auto pos = geomservice->GetVTXPixelLocation( digihit->layer, digihit->chip, digihit->pixel );
64          auto r = pos.Perp();
65          if( r > min_radius ){
66
67              auto hit = new EEndCapHit();
68              hit->x = pos.X();
69              hit->y = pos.Y();
70              hit->z = pos.Z();
71              hit->t = ((double)digihit->t - 125.0)*2.50E-1;  // Here we would apply calibrations read from DB
72              hits.push_back(hit);
73          }
74      }
75
76      /// Publish outputs
77      Set(hits);
78
79      // n.b. if we created additional types of objects we could also add them to the event using event->Insert() )
80  }
```

Summary

- JANA is a multithreaded framework project with nearly 2 decades of experience behind it

- JANA2 is a rewrite incorporating more modern coding and CS practices and improving on the original using lessons learned
  - Streaming DAQ and Heterogeneous hardware support strongly considered in redesign

- JLab is a **partner lab** in the EIC project and is ready to commit **~1 FTE** to feature development, support and implementation in the EIC software stack
  - Nathan Brei, David Lawrence, Dmitry Romanov, + others in EPSCI/EIC
  - Very interested in elevating this project to include community involvement

Github: https://github.com/JeffersonLab/JANA2
Documentation: https://jeffersonlab.github.io/JANA2/
Example project: https://github.com/faustus123/EIC_JANA_Example

Publications:

https://arxiv.org/abs/2202.03085 *Streaming readout for next generation electron scattering experiments*
https://doi.org/10.1051/epjconf/202125104011 *Streaming Readout of the CLAS12 Forward Tagger Using TriDAS and JANA2*
https://doi.org/10.1051/epjconf/202024501022 *JANA2 Framework for Event Based and Triggerless Data Processing*
https://doi.org/10.1051/epjconf/202024507037 *Offsite Data Processing for the GlueX Experiment*
https://iopscience.iop.org/article/10.1088/1742-6596/119/4/042018 *Multi-threaded event reconstruction with JANA*
https://pos.sissa.it/070/062 *Multi-threaded event processing with JANA*
https://iopscience.iop.org/article/10.1088/1742-6596/219/4/042011 *The JANA calibrations and conditions database API*
https://iopscience.iop.org/article/10.1088/1742-6596/1525/1/012032 *JANA2: Multithreaded Event Reconstruction*

Python support in JANA2

EPSCI
EXPERIMENTAL PHYSICS SOFTWARE AND COMPUTING INFRASTRUCTURE

As pure python script

python3 jana.py

```
4  # This example JANA python script
5  import time
6  import jana
7
8  print('Hello from jana.py!!!')
9
10 # Turn off JANA's standard ticker so we can print our own updates
11 jana.SetTicker(False)
12
13 # Wait for 4 seconds before allowing processing to start
14 for i in range(1,5):
15         time.sleep(1)
16         print(" waiting ... %d" % (4-i))
17
18 # Start event processing
19 jana.Start()
20
21 # Wait for 5 seconds while processing events
22 for i in range(1,6):
23         time.sleep(1)
24         print(" running ... %d  (Nevents: %d)" % (i, jana.GetNeventsProcessed()))
25
26 # Tell program to quit gracefully
27 jana.Quit()
```

As embedded interpreter

jana -PPLUGINS=janapy -PJANA_PYTHON_FILE=myfile.py

```
4  # This is a simple example JANA python script. It shows how to add plugins
5  # and set configuration parameters. Event processing will start once this
6  # script exits.
7  import jana
8
9  jana.AddPlugin('JTest')
10 jana.SetParameterValue('jana:nevents', 200)
11
12 jana.Run()
```

EPSCI Members

group formed Feb. 2020

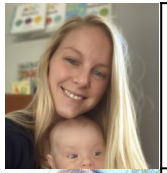
David Lawrence PhD (physics) - Group Lead
Expertise: Physics, C++, software framework, online systems

Nathan Brei BS(aerospace engineering) MS (CS)
Expertise: Programming languages, parallel processing

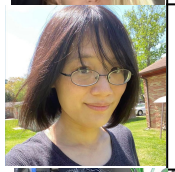
Thomas Britton PhD (physics)
Expertise: Physics, software, OSG, AI DQM

Michael Goodrich MS (physics) PhD (Computational Modeling/Simulation)
Expertise: Computational Modeling, Data Science, physics, real-time, Data Acquisition

Vardan Gyurjyan PhD (physics) MS (CS)
Expertise: Data Acquisition, Java, C++, software frameworks

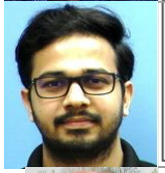


Torri Jeske PhD (physics)
Expertise: Experimental Nuclear Physics, Data Analysis, Detector Calibration



Xinxin "Cissie" Mei PhD (CS)
Expertise: Computer Science, GPU performance

****Diana McSpadden BS (math) MSDS** in progress*
Expertise: Data Science, AI/ML



****Kishan Rajput MS (CS)***
Expertise: Data Science, AI/ML

Carl Timmer PhD (physics)
Expertise: Data Acquisition, Java, file format, I/O

* Member of DS Dept.