JANA2: Multi-threaded Event Reconstruction

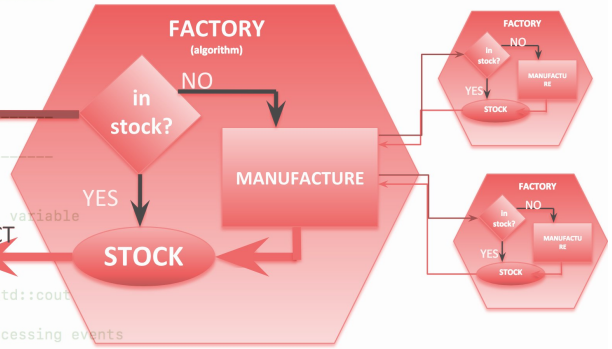
Amber Boehnlein, Nathan Brei, **David Lawrence**, Dmitry Romanov
Jefferson Lab

May 21, 2019

EIC Software Meeting

Trieste, Italy

```
//----------------------------------
// IsJoined
//----------------------------------
bool JThread::IsJoined(void)
{
   return _isjoined;
}

//----------------------------------
// Loop
//----------------------------------
void JThread::Loop(void)
{
   // Set thread_local global variable
   JTHREAD = this;

   //Set logger
   mLogger = new JLog(0); //std::cout

   /// Loop continuously, processing events
   try{
      while( mRunStateTarget != kRUN_STATE_ENDED )
      {
         // If specified, go into idle state
         if( mRunStateTarget == kRUN_STATE_IDLE ) mRunState = kRUN_STATE_IDLE;

         // If not running, sleep and loop again
         if(mRunState != kRUN_STATE_RUNNING)
         {
            std::this_thread::sleep_for(mSleepTime); //Sleep a minimal amount.
            continue;
         }

         //Check if not enough event-tasks queued
         if(CheckEventQueue())
         {
            //Process-event task is submitted, redo the loop in case we want to buffer
            continue;
         }
```



Jefferson Lab

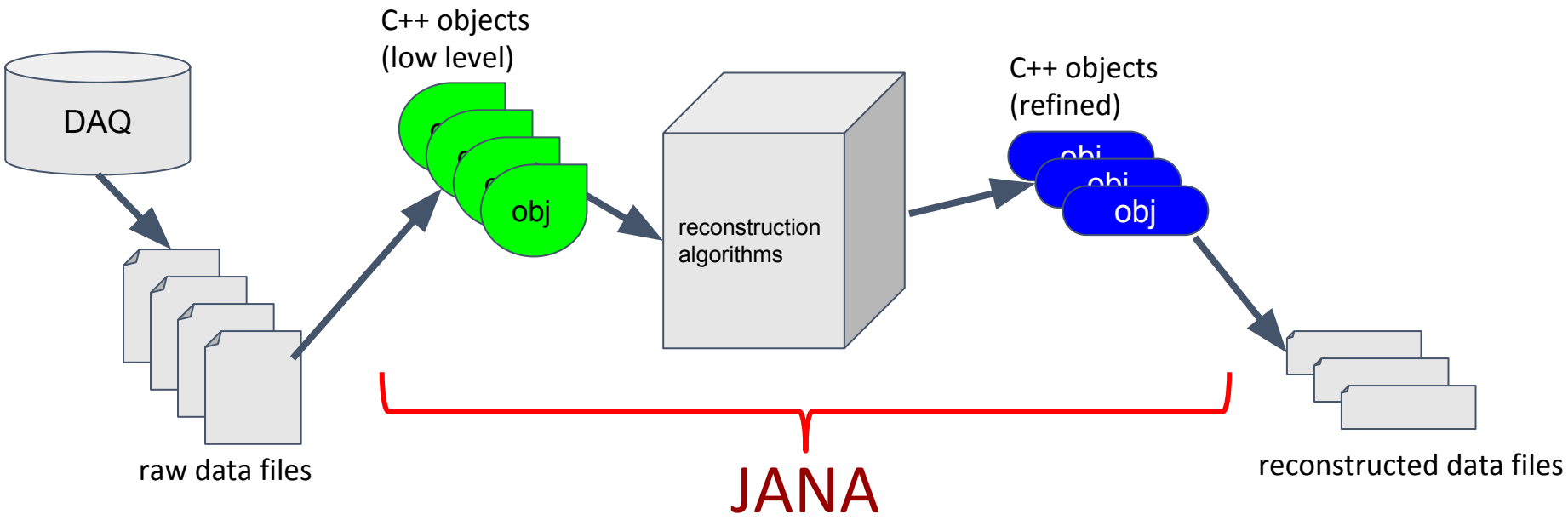
U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

“Event” Reconstruction

- Physics requires studying a single reaction at a time
- High speed (=high statistics) leads to overlapping reactions in time
- “Event” here really means a slice of time
  - Traditional electronic trigger = single reaction
  - Streaming readout = potentially many reactions

Overly Simplified View of JANA's Role



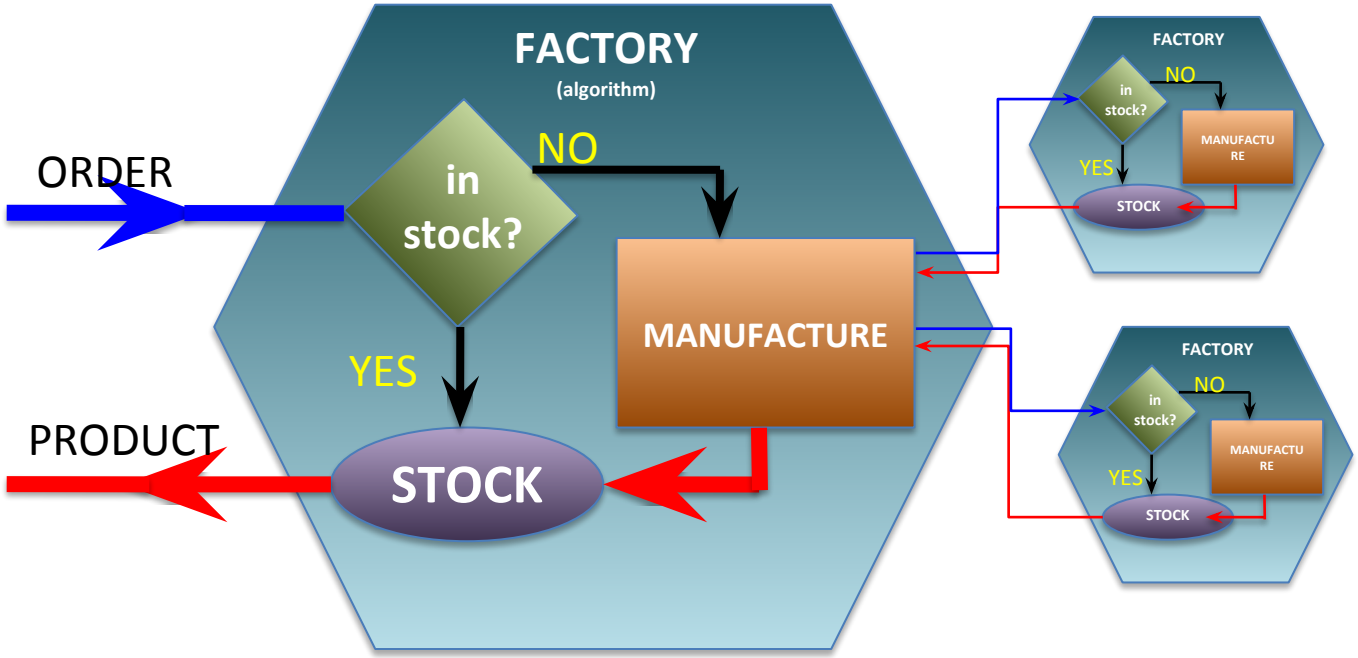
Some Goals of the JANA framework

- 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...

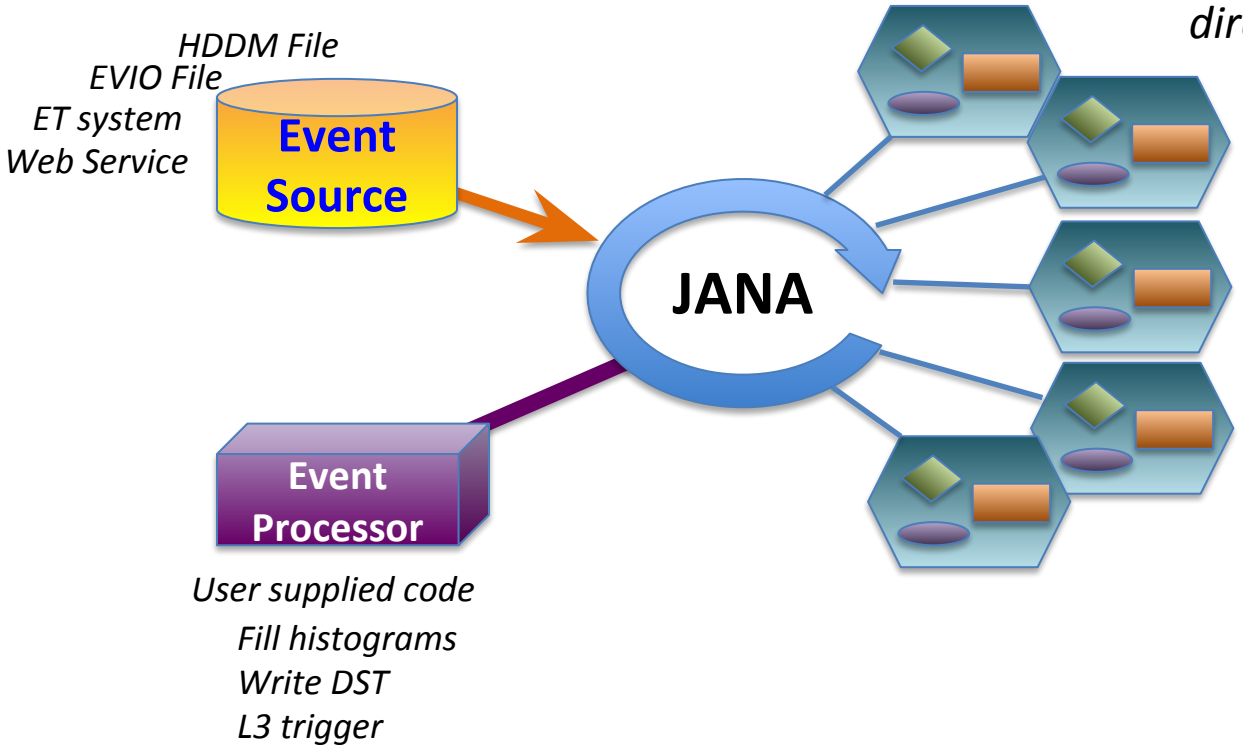
Factory Model



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



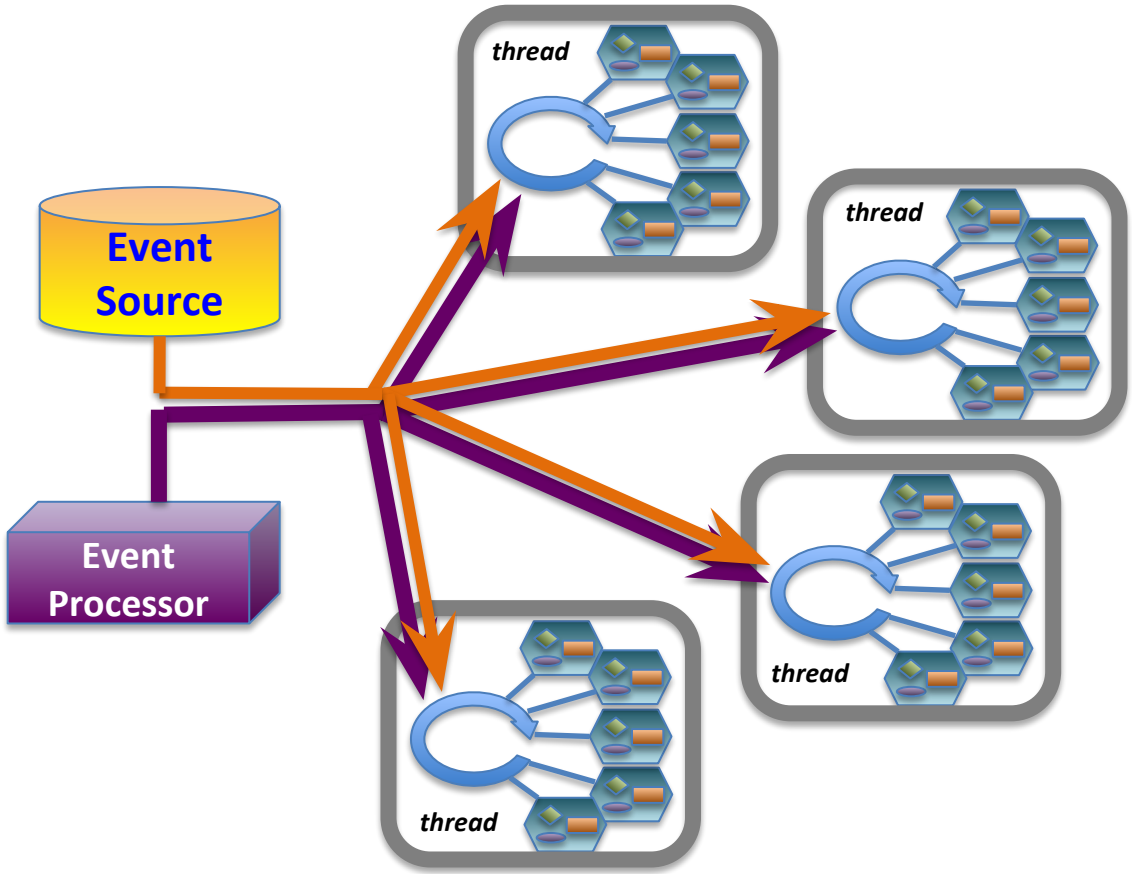
Framework has a layer that directs object requests to the factory that completes it

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

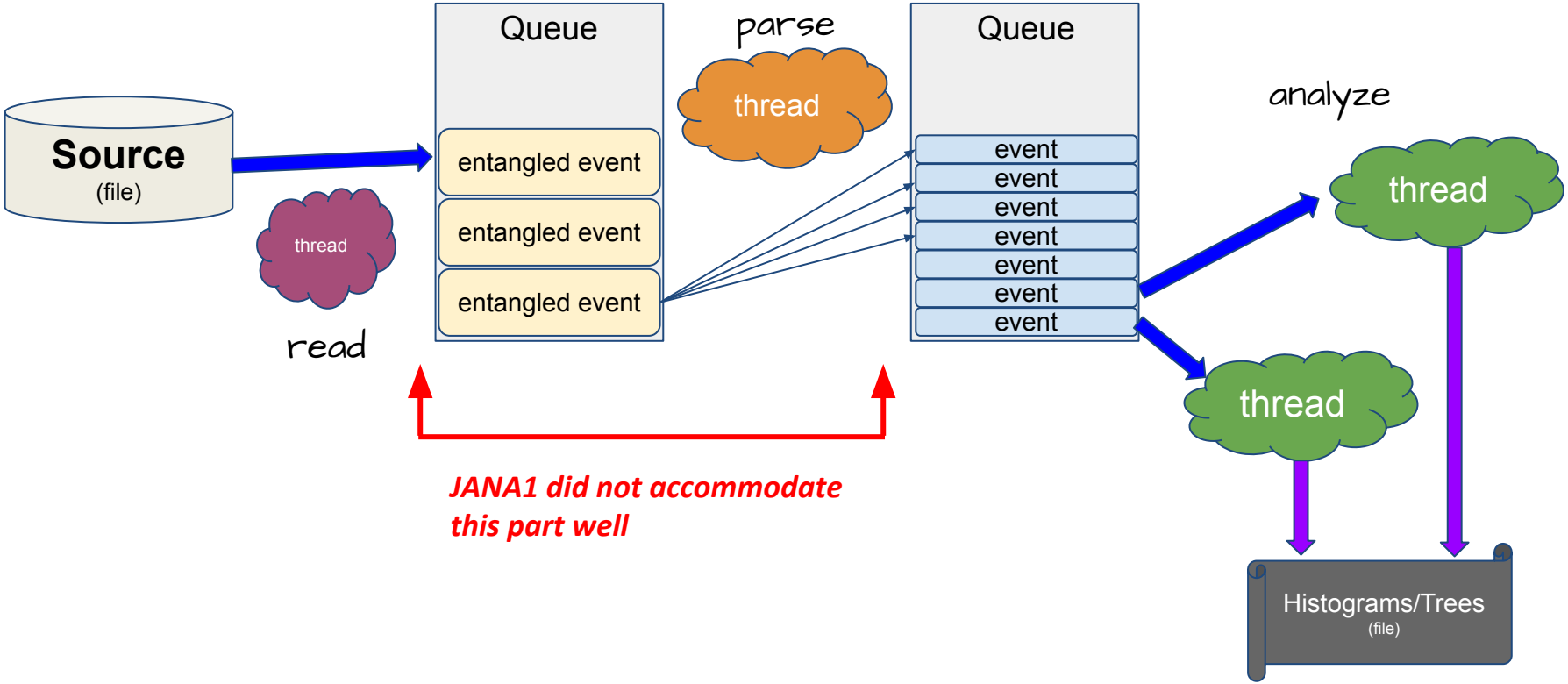
This allows the framework to easily redirect requests to alternate algorithms specified by the user at run time

Multi-threading

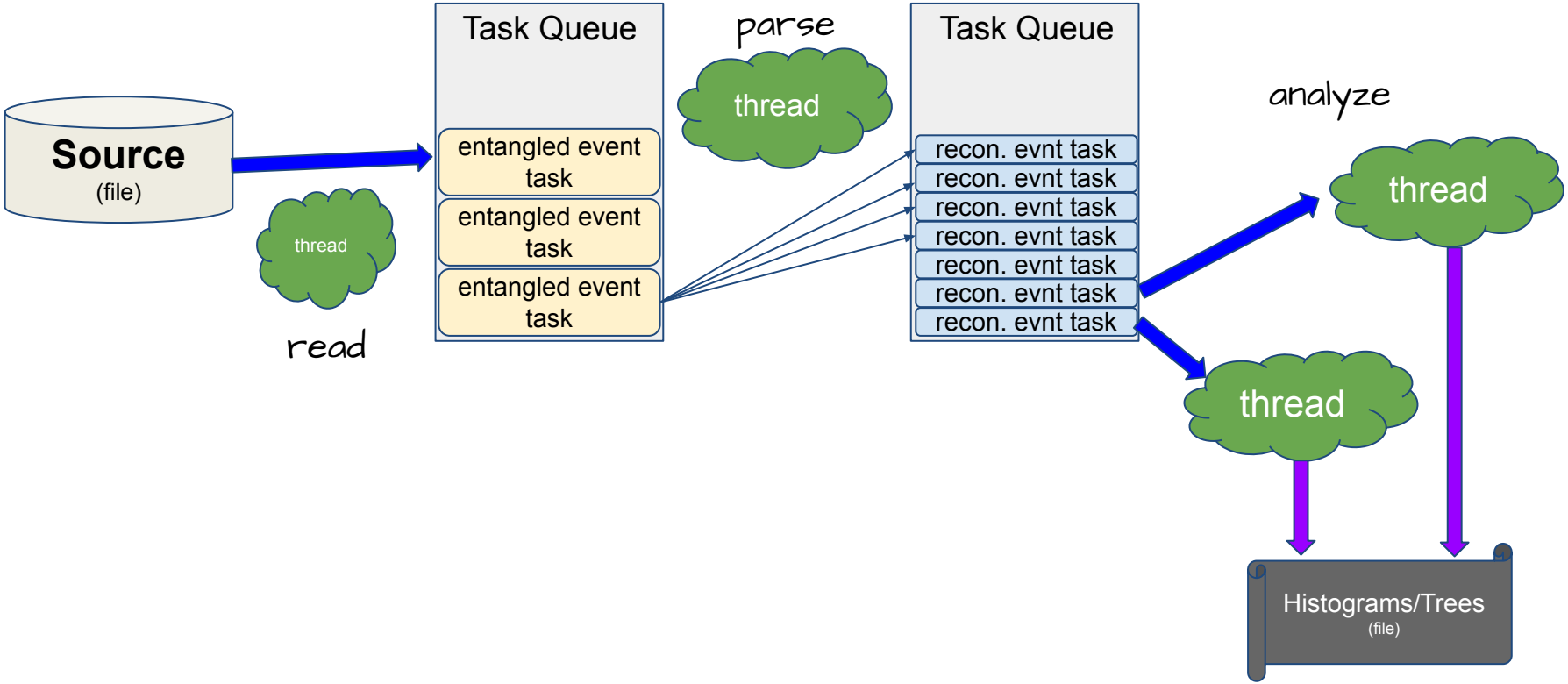
- *Each thread has a complete set of factories making it capable of completely reconstructing a single event*
- *Factories only work with other factories in the same thread eliminating the need for expensive mutex locking within the factories*
- *All events are seen by all Event Processors (multiple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



High event rate (100kHz) requires buffering in front end leading to entangled events. "Event" changes meaning as it propagates through.

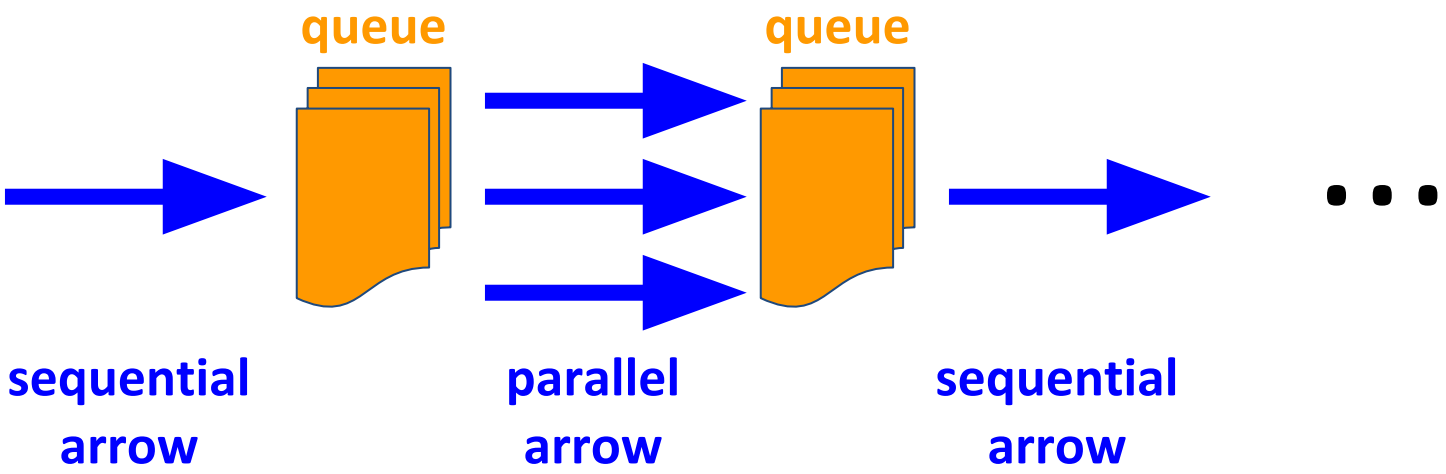


JANA2 generalizes the "event" queue to allow multiple queues. Threads are now responsiible for moving data between queues



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



What the user needs to know:

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

for(auto t : tracks){

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

}
```

*vector<const *DTrack> tracks*

If an alternate factory is desired:

auto tracks = jevent->Get<**DTrack**>("MyTest");

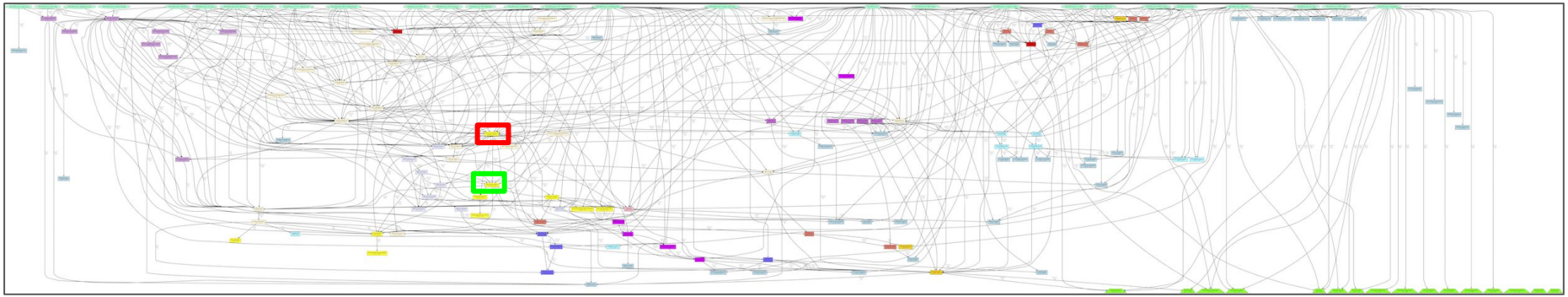
or, even better

set configuration parameter: **DTrack:DEFTAG=MyTest**

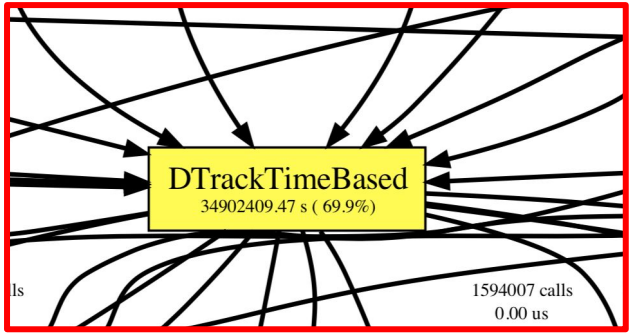
- Configuration parameters are set at run time
- NAME:DEFTAG is special and tells JANA to re-route ALL requests for objects of type NAME to the specified factory.

JANA process configured for full recon of GlueX data

Plugins

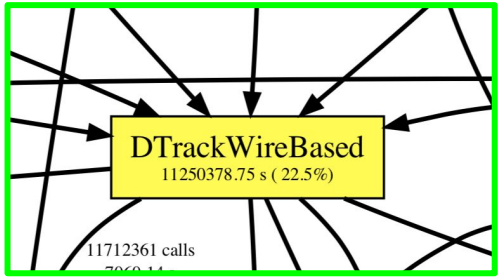


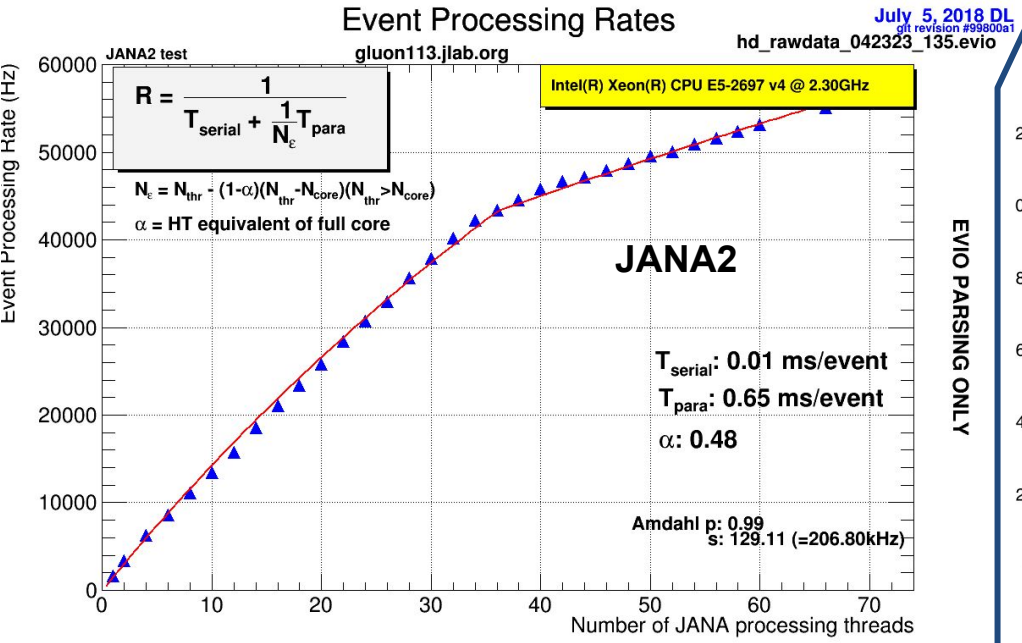
EVIO file



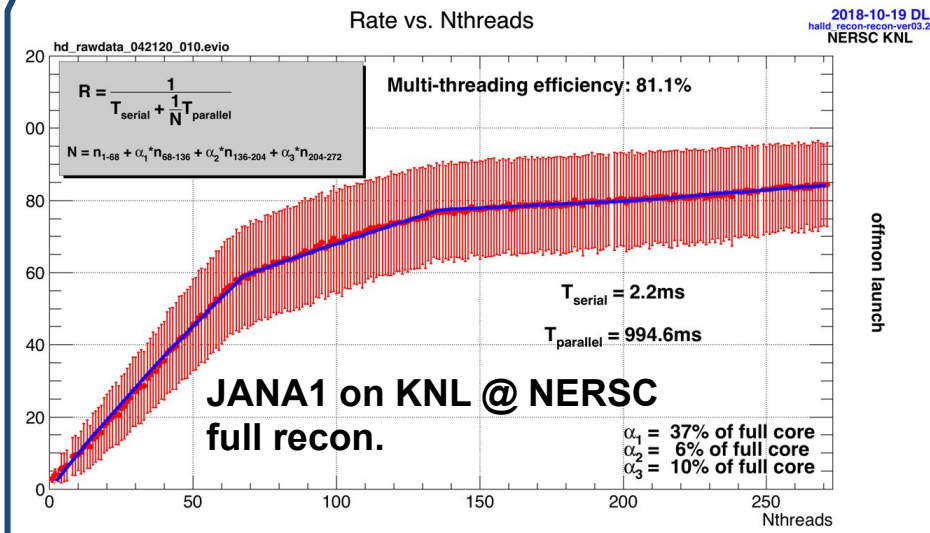
Run 42513:

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





EVIO PARSING ONLY



offmon launch

"Parsing Only" Job demonstrates more stable CPU usage with single flavor of thread

Full event reconstruction using JANA1 on KNL via container at NERSC

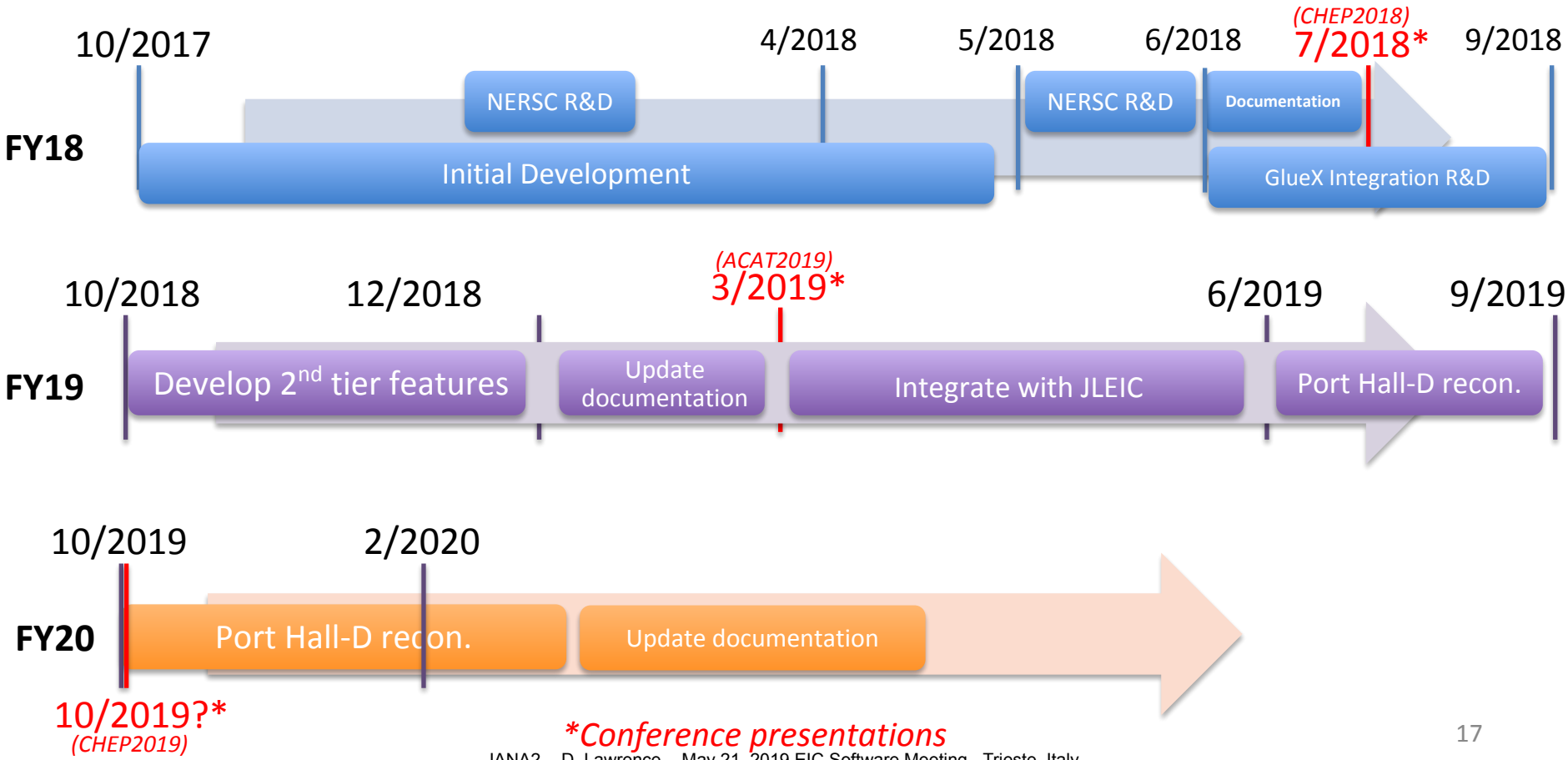
Features Added in JANA2

- Better use of “modern” C++ features
  - thread model via C++ language
  - lock guards
  - shared pointers
  - atomic variables
- Generalized use of threads (pool)
  - multiple queues
  - arrows (sequential or parallel)
- NUMA awareness
- Python API (both embedded and as an extension)

Features maintained from JANA1

- On demand interface
- Plugin support
- Rich configuration parameter feature
- Built-in profiling features
- Automated ROOT tree generation*

Schedule for JANA2 LDRD Project at JLab



*Conference presentations

Summary

- JANA2 is:
  - C++ event processing framework
  - multi-threaded
  - currently being written with >10 years experience with JANA1
  - Bigger, better, badder (gooder?)
- Python interface (embedded and extension)
- First production release by end of year

JANA2 Website on GitHub

Secure | https://jeffersonlab.github.io/JANA2/

JANA2

Multi-threaded HENP Event Reconstruction

View project on GitHub

Home | Getting Started | Download | FAQ

// Welcome to JANA!

JANA is a C++ framework for multi-threaded HENP (High Energy and Nuclear Physics) event reconstruction. It is very efficient at multi-threading with a design that makes it easy for less experienced programmers to contribute pieces to the larger reconstruction project. The same JANA program can be used to easily do partial or full reconstruction, fully maximizing the available cores for the current job.

It's design strives to be extremely easy to setup when first getting started, yet have a depth of customization options that allow for more complicated reconstruction as your project grows. The intent is to make it easy to run on a laptop for local code development, but to also be highly efficent when deploying to large computing sites like NERSC.

JANA is currently undergoing a complete rewrite. The new version will be JANA 2. The code is not quite ready for actual use yet, but you are free to browse around to see how progress is going. The project is hosted on GitHub

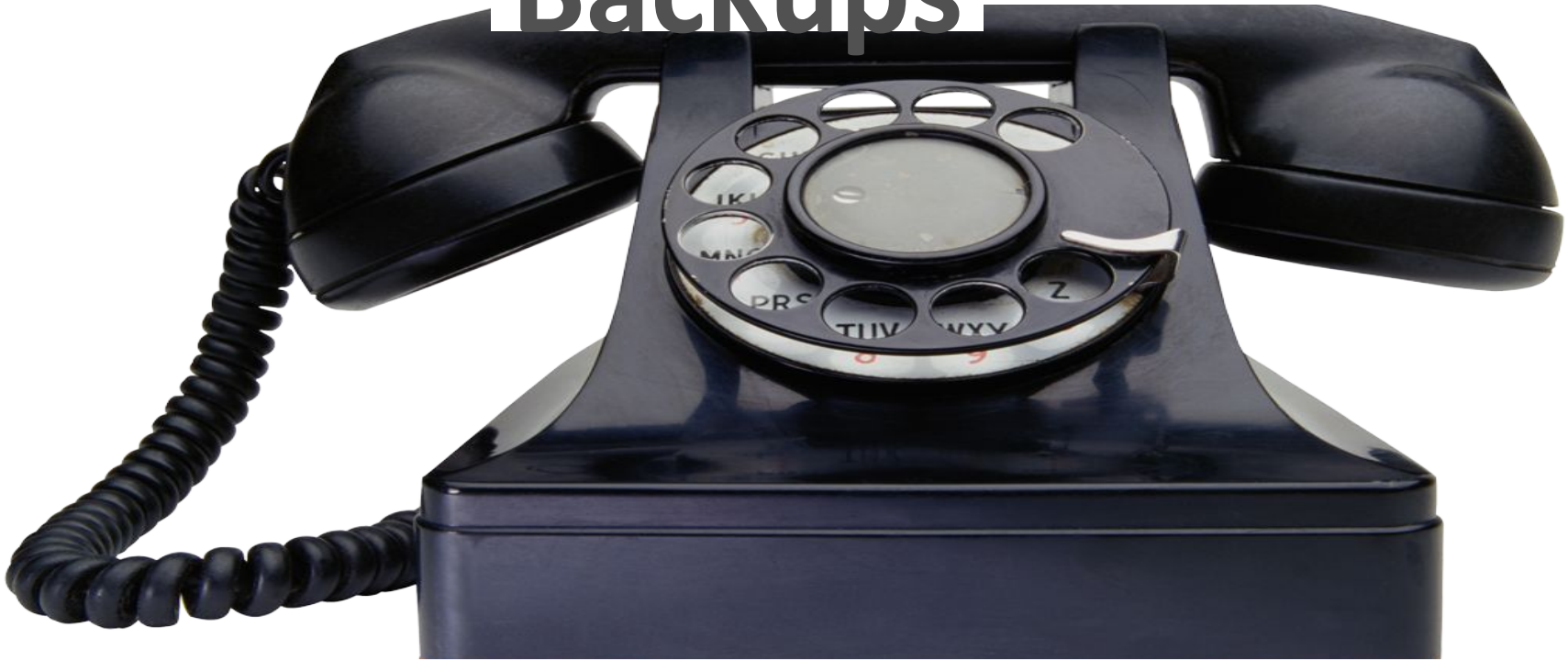
```
auto tracks = jevent->Get<DTrack>(tracks);

for(auto t : tracks){
  // ... do something with a track
}
```

JANA2 is maintained by JeffersonLab.

This page was generated by GitHub Pages.

This work supported by Jefferson Lab LDRD program project LDRD1908

Backups



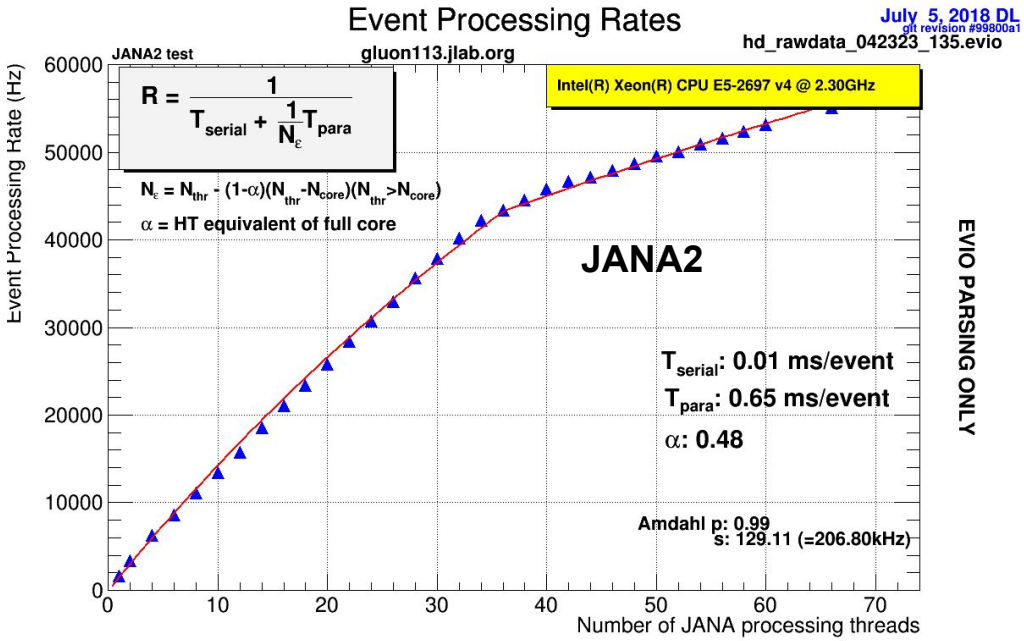
GlueX Computing Needs

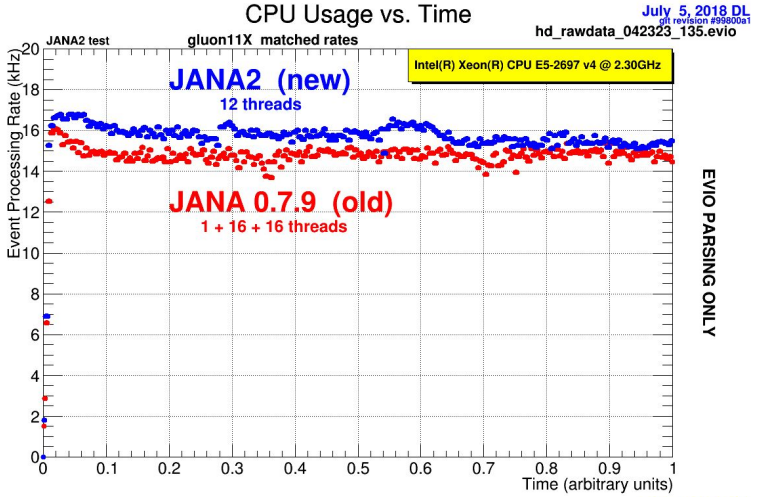
	2017 (low intensity GlueX)	2018 (low intensity GlueX)	2019 (PrimEx)	2019 (high intensity GlueX)
Real Data	1.2PB	6.3PB	1.3PB	3.1PB
MC Data	0.1PB	0.38PB	0.16PB	0.3PB
Total Data	**1.3PB**	**6.6PB**	**1.4PB**	**3.4PB**
Real Data CPU	21.3Mhr	67.2Mhr	6.4Mhr	39.6Mhr
MC CPU	3.0Mhr	11.3MHr	1.2Mhr	8.0Mhr
Total CPU	**24.3PB**	**78.4Mhr**	**7.6Mhr**	**47.5Mhr**

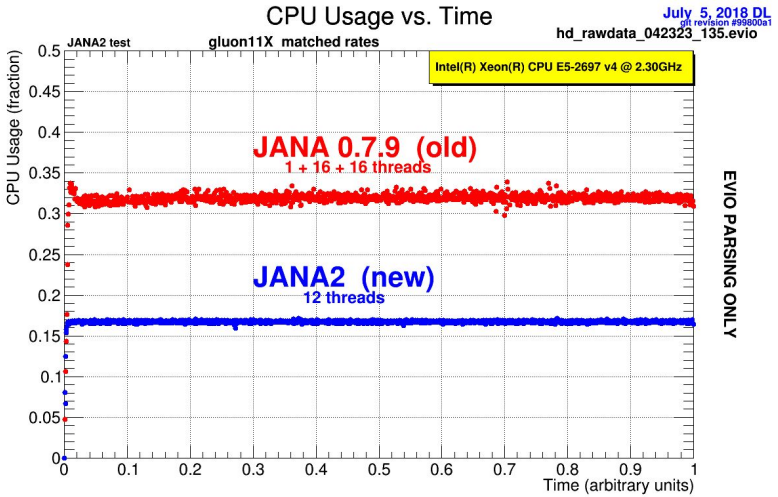
Anticipate 2018 data will be processed by end of summer 2019

Projection for out-years of GlueX High Intensity running at 32 weeks/year

	Out - years (high intensity GlueX)
Real Data	16.2PB
MC Data	1.4PB
Total Data	**17.6PB**
Real Data CPU	125.6Mhr
MC CPU	36.5Mhr
Total CPU	**162.1Mhr**







"Parsing Only" Job demonstrates more stable CPU usage with single flavor of thread