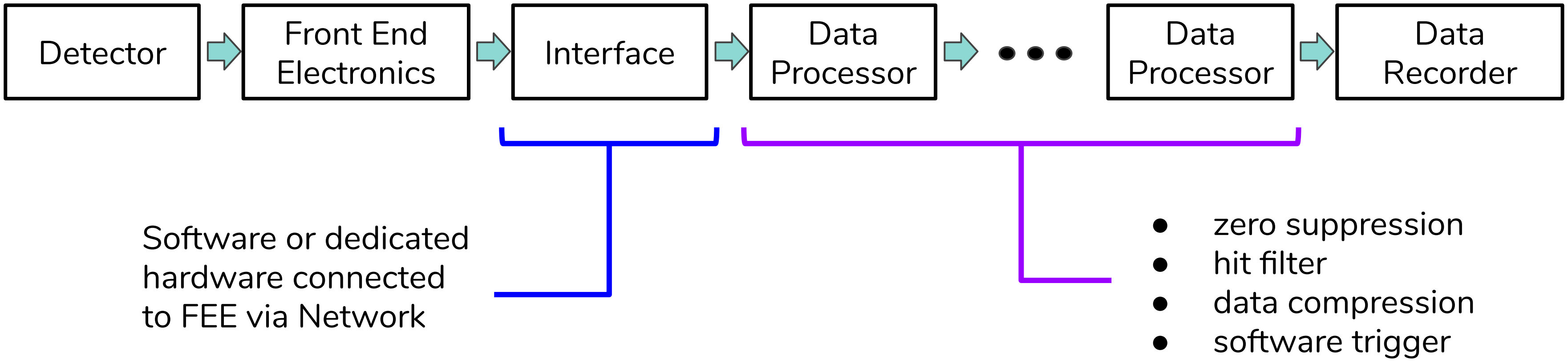
JANA2

David Lawrence, Nathan Brei
JLab EPSCI group

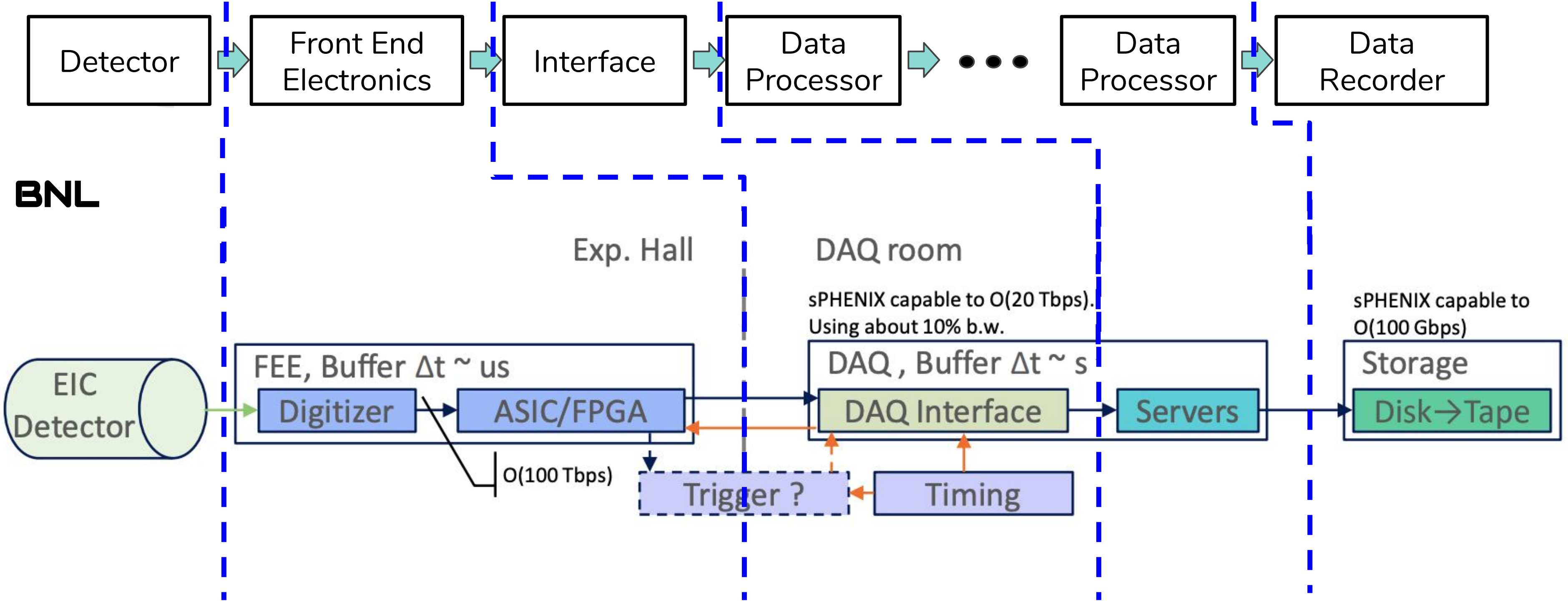
May 24, 2020 Streaming Readout VI

Basic Components of an SRO System



Jefferson Lab

EPSCI



[Jin Huang Talk](https://indico.bnl.gov/event/5807/contributions/26937/attachments/21875/30184/EIC_DAQ_Streaming_Meeting.pdf)

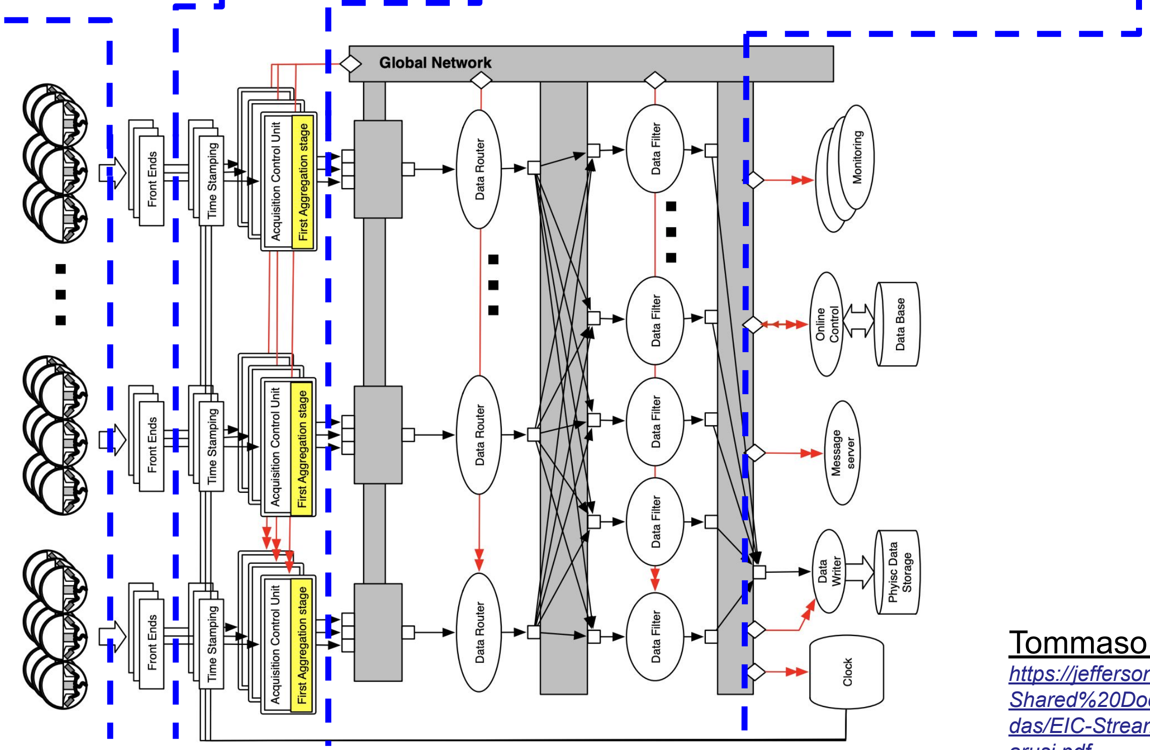
https://indico.bnl.gov/event/5807/contributions/26937/attachments/21875/30184/EIC_DAQ_Streaming_Meeting.pdf

Jefferson Lab

EPSCI

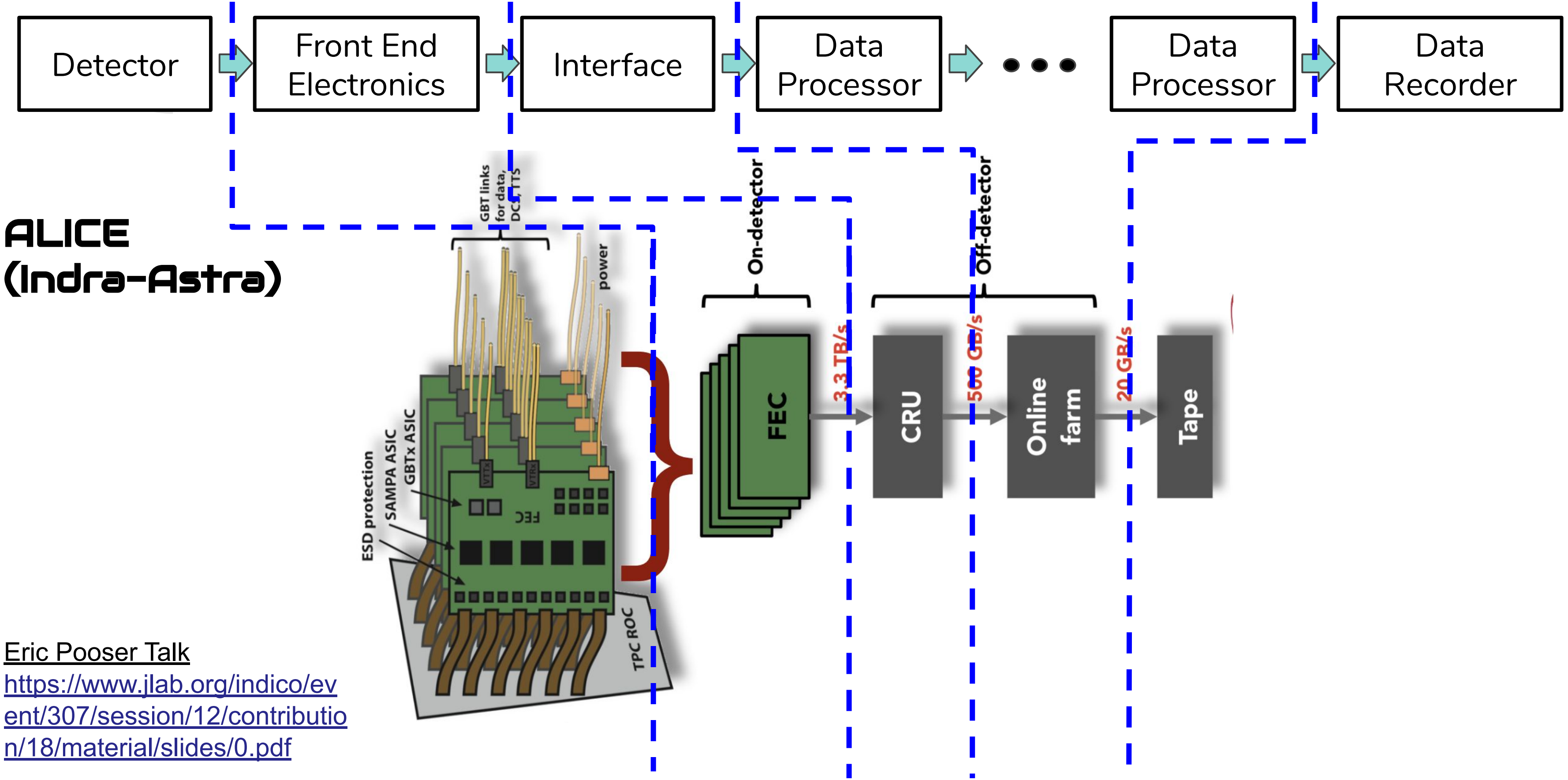
Detector → Front End Electronics → Interface → Data Processor → • • • → Data Processor → Data Recorder

TriDAS



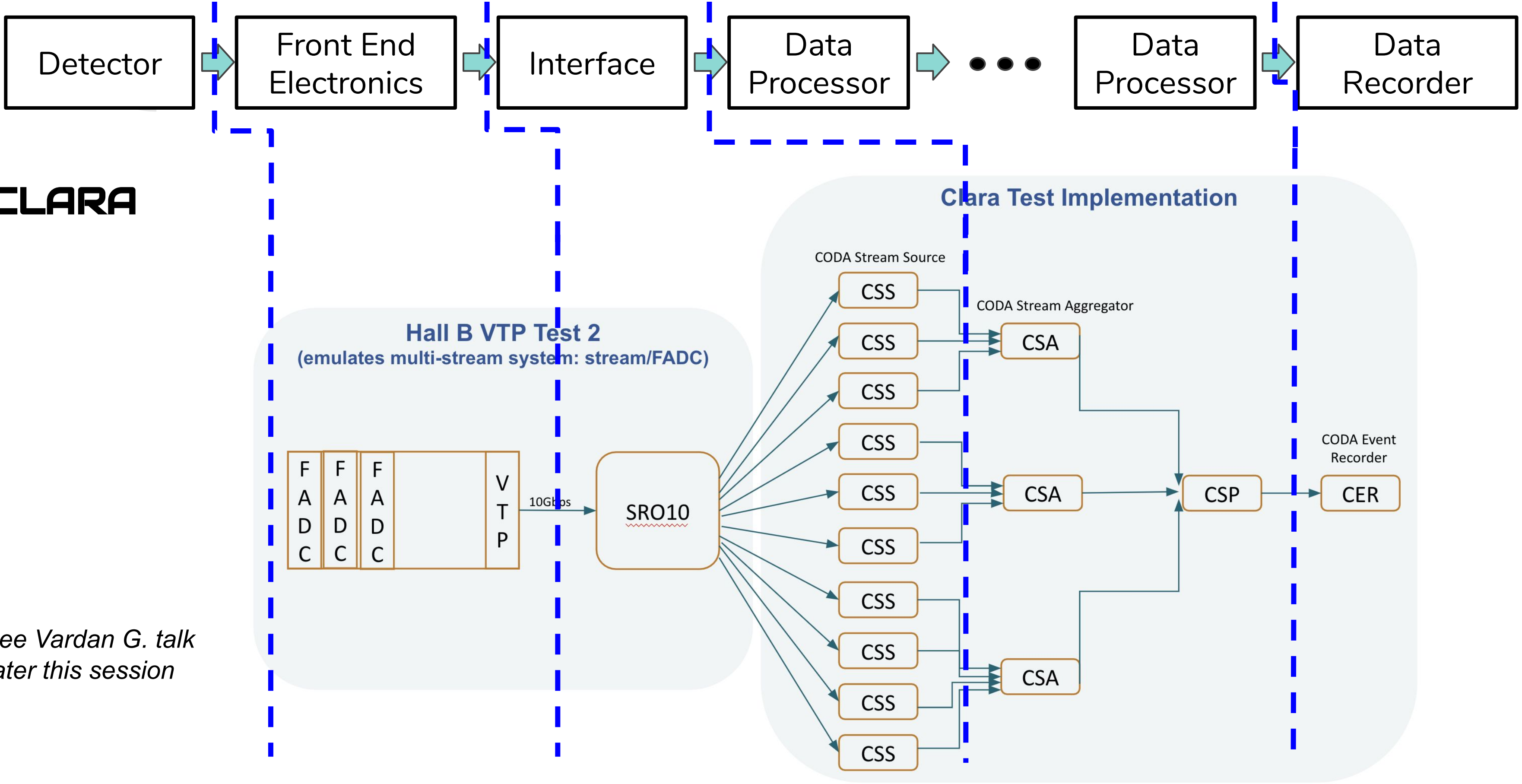
Tommaso Chiarusi Talk

https://jeffersonlab.sharepoint.com/sites/SciComp/Shared%20Documents/EPSCI/SRO/Streaming/Tridas/EIC-Stream_Readout-Camogli_20190524_chiarusi.pdf



ALICE (Indra-Astra)

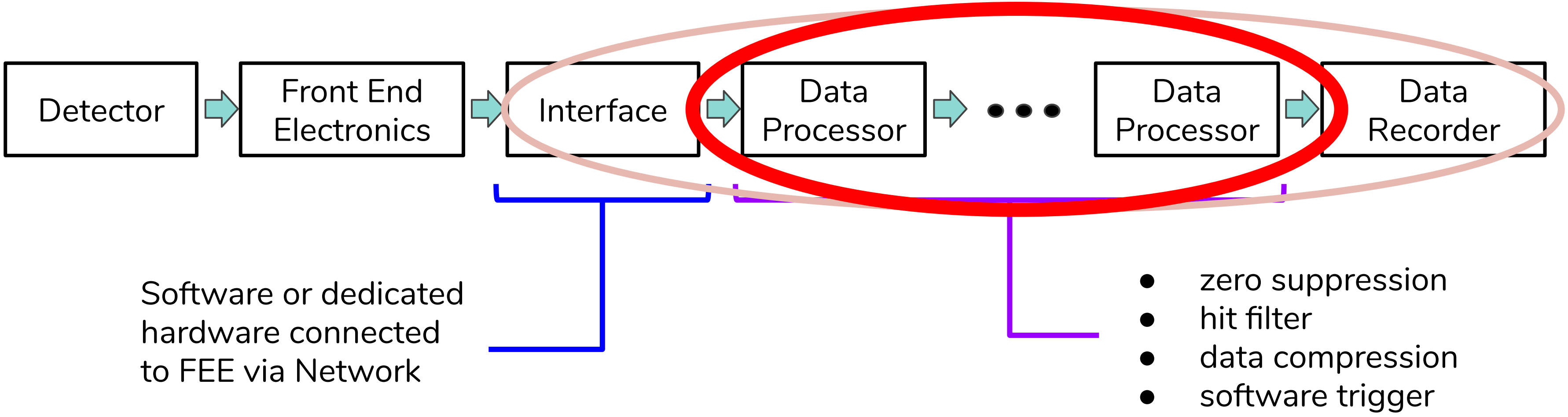
[Eric Pooser Talk](https://www.jlab.org/indico/event/307/session/12/contribution/18/material/slides/0.pdf)
https://www.jlab.org/indico/event/307/session/12/contribution/18/material/slides/0.pdf



CLARA

see Vardan G. talk later this session

JANA2’s role in SRO Systems

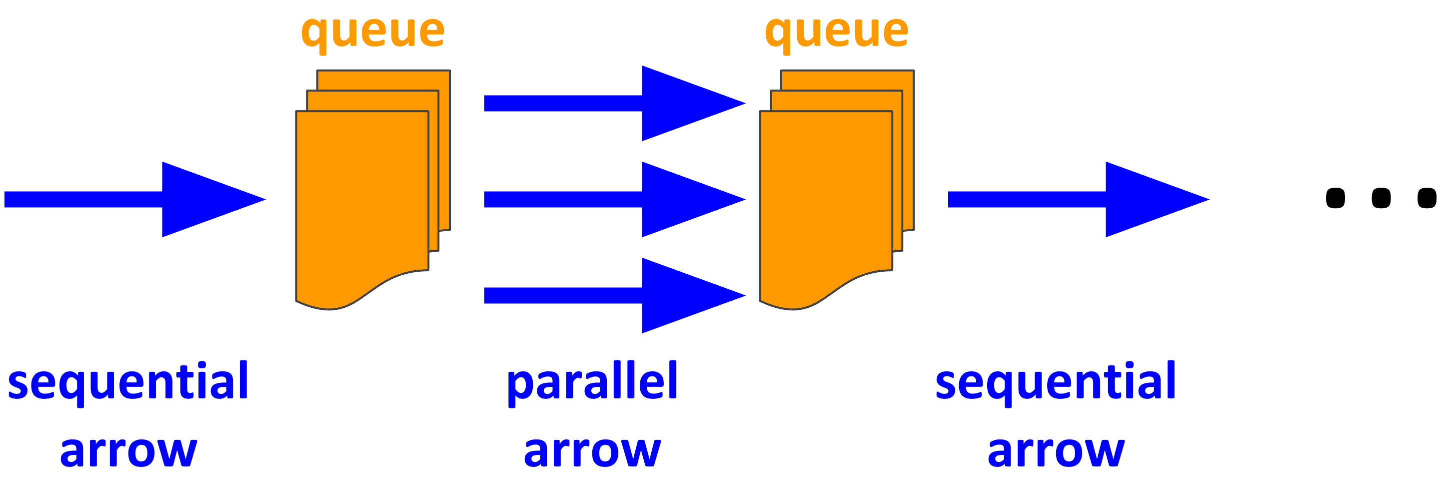


What is JANA2

- C++ Multi-threaded event/time slice processing framework
- Designed to support
  - Offline event reconstruction
  - Online Data Quality Monitoring
  - Software (aka L3) Trigger
- Used in:
  - GlueX
  - eJANA (EIC)
  - BDX
  - Multiple SRO test stands at JLab

Arrow Queue Pattern

- 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



Reactive/Dataflow Programming

- Data is presented to arrow in the form of a queue
- Arrow transforms data and places it in downstream queue
- Minimal synchronization time spent in accessing queues
- Course tasks within arrow can eliminate most or all other synchronization points

Simplest possible impl:

Legend

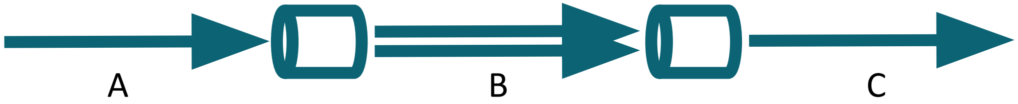
A: `event_source.get() [SEQ]`

B: `event_processor.process_parallel() [PAR]`

C: `event_processor.process_sequential() [SEQ]`

JANA1 impl:

JANA2 impl:



Idea: Structure computation as a dataflow graph.

Idea: For each critical section, replace lock with a sequential arrow. Mediate handover of events between arrows via a queue.

Idea: Structure code identically to the corresponding work-time/work-span analysis

Advantages:

- Generalizable, which enables support for event blocks, subevents, software triggers, event building, etc

- Trivial parallel bottleneck/efficiency analysis
- Control of memory use via backpressure
- Control of memory locality
- Control of parallelism granularity

Problem: If events are too large, we may run out of memory before we run out of cores.

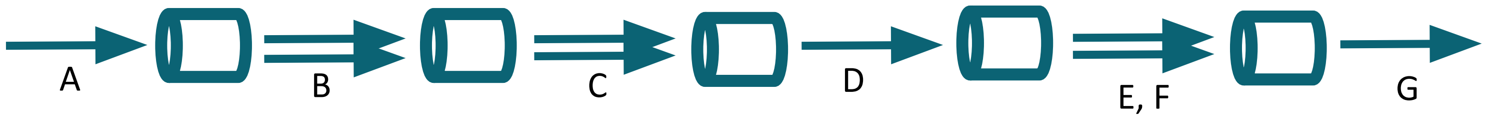
Problem: If we are using a GPU or TPU, we may prefer to submit work in batches of (e.g.) 256 equally sized tasks.

Solution: Split/merge pattern

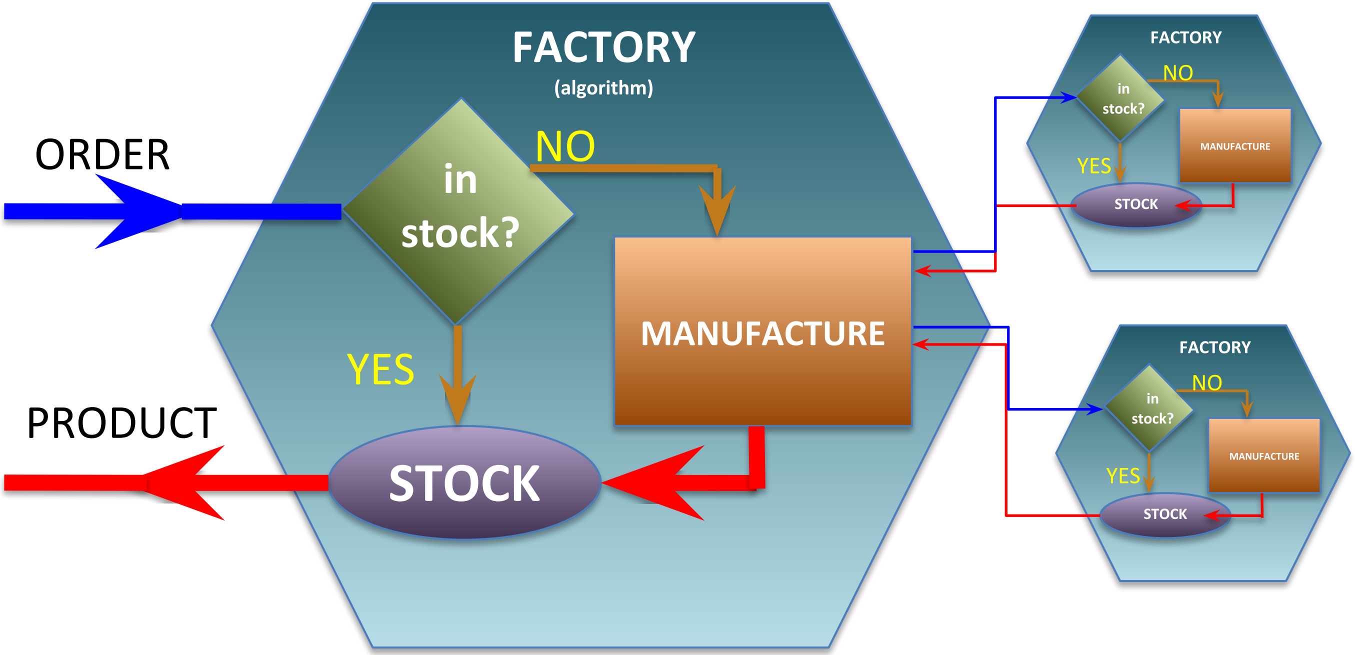
```
subevent_processor.split :: event -> [T]
subevent_processor.process :: T -> U
subevent_processor.merge :: [U], event -> event
```

Legend

A: `event_source.get() [SEQ]`
B: `subevent_processor.split() [PAR]`
C: `subevent_processor.process() [PAR]`
D: `group_by() [SEQ]`
E: `subevent_processor.merge() [PAR]`
F: `event_processor.process_parallel() [PAR]`
G: `event_processor.process_sequential() [SEQ]`



Factory Model Embedded in Arrow



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

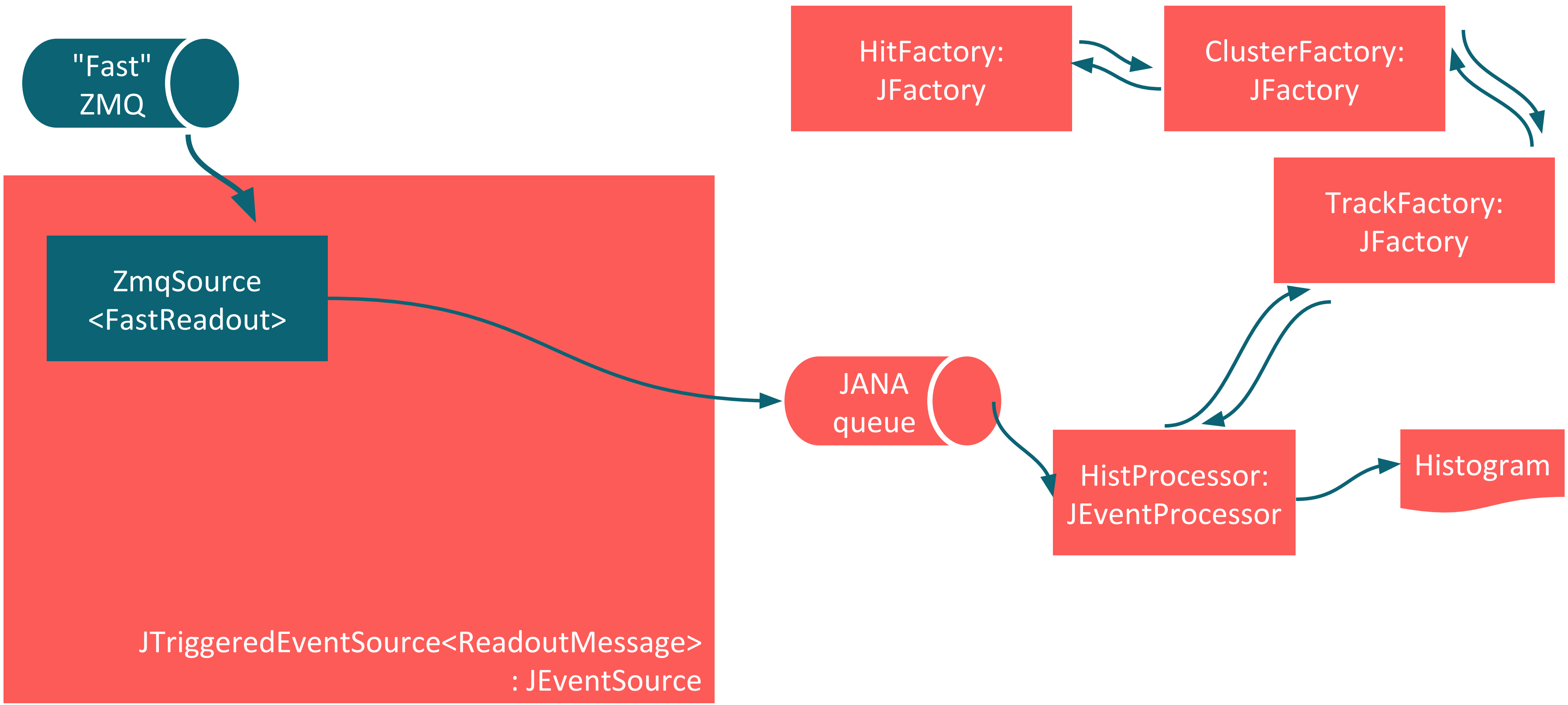
Conservation of CPU cycles!

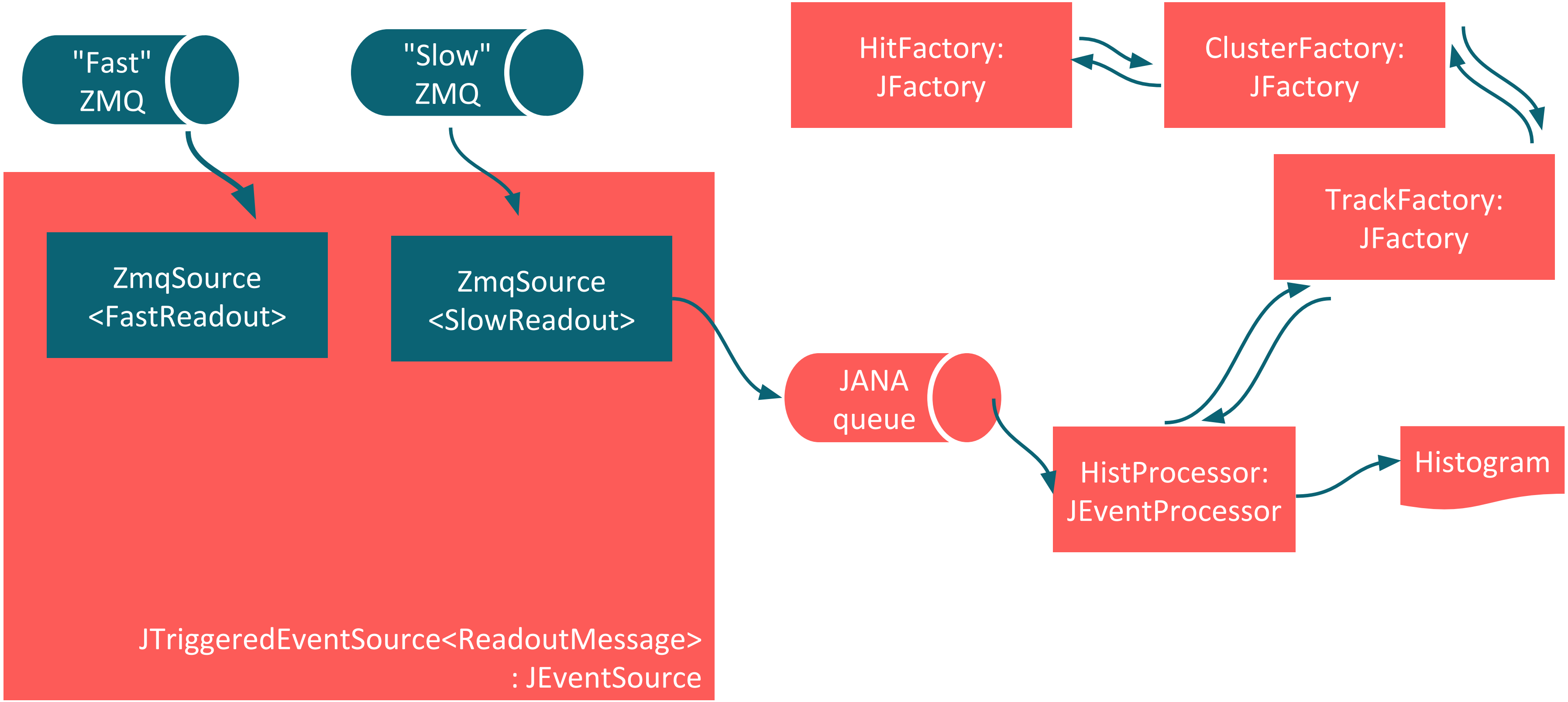
Data on Demand => Software Trigger

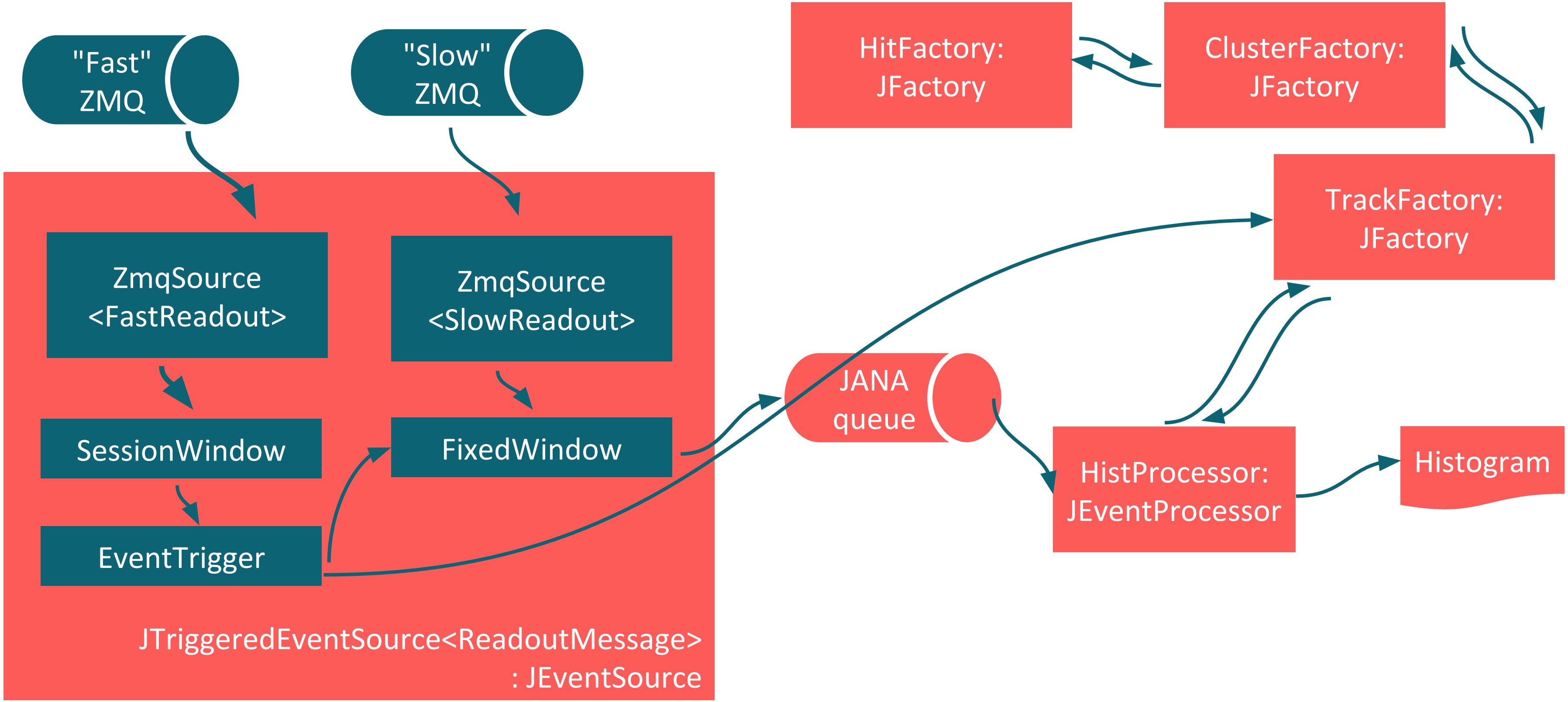
Event by event decision on whether to activate a factory:

Software triggers may have multiple "keep" or "discard" conditions that may be probed in order of CPU cost

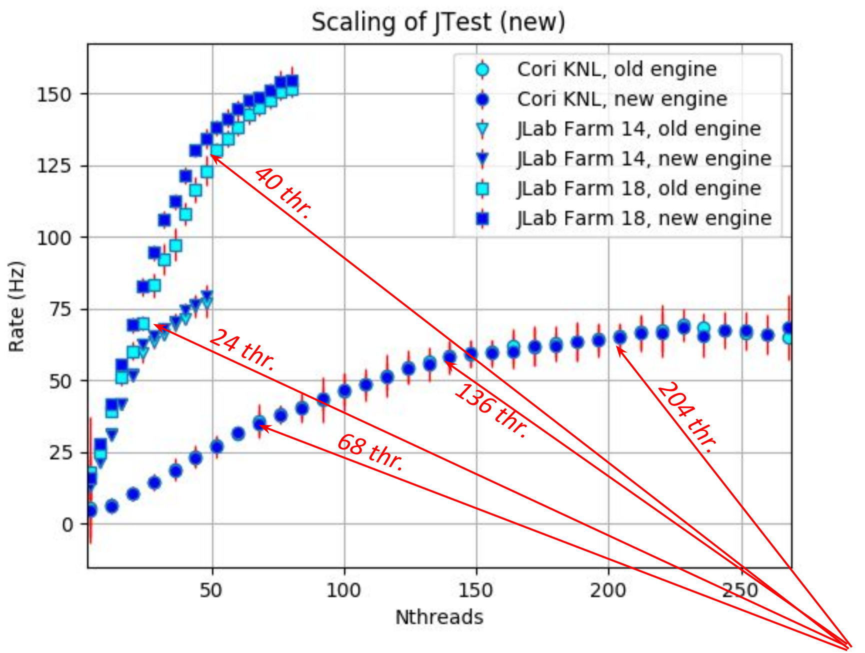
```
// Getting hit objects is cheap so we check that first
auto NcaloHits = jevent->Get<CaloHit>().size();
if( NcaloHits>minCaloHits ){

    keep_event = true;

// Tracks factory only activated if not already keeping event
}else if( jevent->Get<Tracks>().size() > minTrackHits ) {

    keep_event = true;

}
```

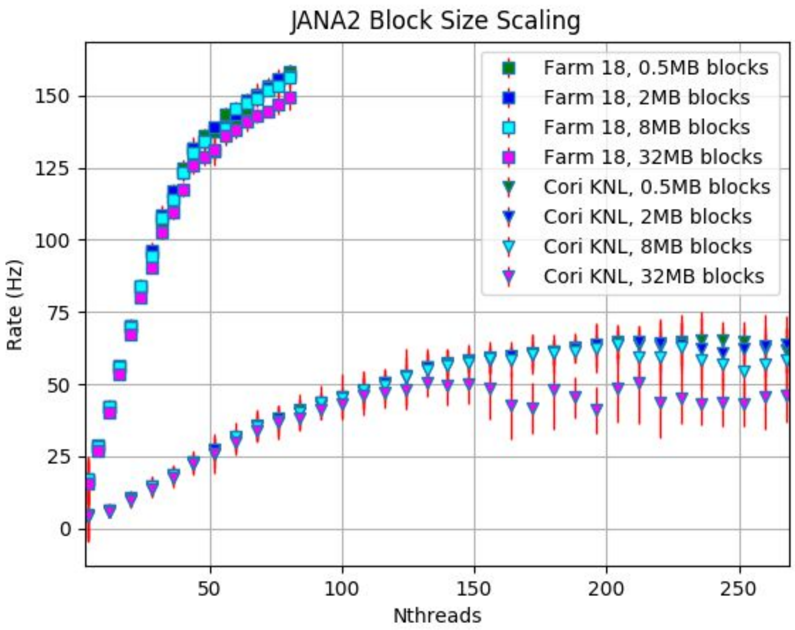

"Fast" ZMQ
ZmqSource
<FastReadout>
JTriggeredEventSource<ReadoutMessage>
: JEventSource
HitFactory:
JFactory
ClusterFactory:
JFactory
TrackFactory:
JFactory
JANA
queue
HistProcessor:
JEventProcessor
Histogram


"Fast" ZMQ
"Slow" ZMQ
HitFactory: JFactory
ClusterFactory: JFactory
TrackFactory: JFactory
ZmqSource <FastReadout>
ZmqSource <SlowReadout>
JANA queue
HistProcessor: JEventProcessor
Histogram
JTriggeredEventSource<ReadoutMessage>
: JEventSource


"Fast"
ZMQ
"Slow"
ZMQ
ZmqSource
<FastReadout>
ZmqSource
<SlowReadout>
SessionWindow
FixedWindow
EventTrigger
JTriggeredEventSource<ReadoutMessage>
: JEventSource
HitFactory:
JFactory
ClusterFactory:
JFactory
TrackFactory:
JFactory
JANA
queue
HistProcessor:
JEventProcessor
Histogram

JANA2 Scaling Tests (JLab + NERSC)





kinks indicate hardware boundaries

Summary - JANA2

- Multi-threaded C++ Framework
  - Offline event reconstruction
  - Online Monitoring
  - Software (L3) trigger

- Used for multiple projects
  - GlueX
  - eJANA
  - BDX
  - Indra-Astra
  - CLAS12 SRO R&D

- Well suited for SRO applications
  - Bridges Offline/Online
  - Data on demand
  - Heterogeneous hardware support

https://github.com/JeffersonLab/JANA2

Backups

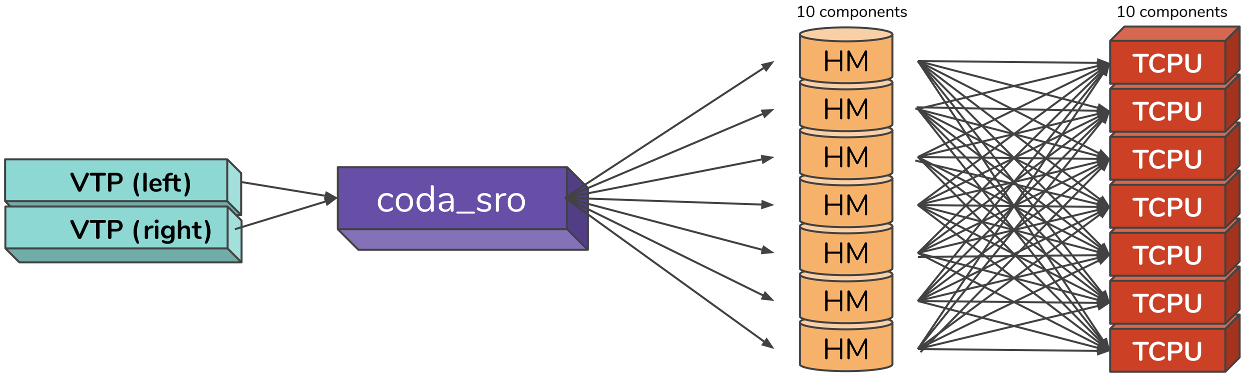
Projects Driving JLab SRO

Experiment	Conditions	Event Rate	Data Rate	Comments
Moller	Production/ integrated mode	1920Hz	130MB/s	Can be handled with the traditional DAQ.
EIC	$L=10^{34}\ cm^{-2}s^{-1}$	450-550kHz (not including background noise rates)	20-25GB/s not included vertex tracker that will generate ~240GB/s	~10kHz/μb, track multiplicity = ~5 JLAB EIC detector design will have millions of channels. Only non-vertex detectors combined will have ~1M channels plus vertex detector: estimated 20-50M channels. In total ~1000 ROC's. Control nightmare (starting stopping a run). Streaming readout has less control requirements.
TIDIS		rTPC hit rates enormous (~800KHz/pad)	4GB/s	How to match up super Bigbyte detected electrons with rTPC detected spectator protons is a big question. Conventional triggered DAQ will be challenged.
SoLID	30 sector GEM		30GB/s	30 separate DAQ's each 1GB/s? How to combine GEM readout with other detectors? Handling GEM hits sharing adjacent sectors.
CLAS12	Phase 2	100KHz	5-7GB/s	

TriDAS Testing

see: https://agenda.infn.it/event/18179/contributions/89843/attachments/63451/76396/EIC-Stream_Readout-Camogli_20190524_chiarusi.pdf

- TriDAS system testing in Experimental Halls B and D at JLab
  - Existing Flash-ADC systems using VTP module with high speed VXS interconnects
  - Multiple testbeds currently available (Sergey B.)
  - Supports multi-node, multi-process and multi-threaded scaling options
    - integrated JANA2 for triggering
  - Only preliminary testing done so far at JLab
    - expect more stress testing over coming months
- Open issues
  - System designed for deep sea neutrino experiments (how well does it scale?)
  - Overall process management

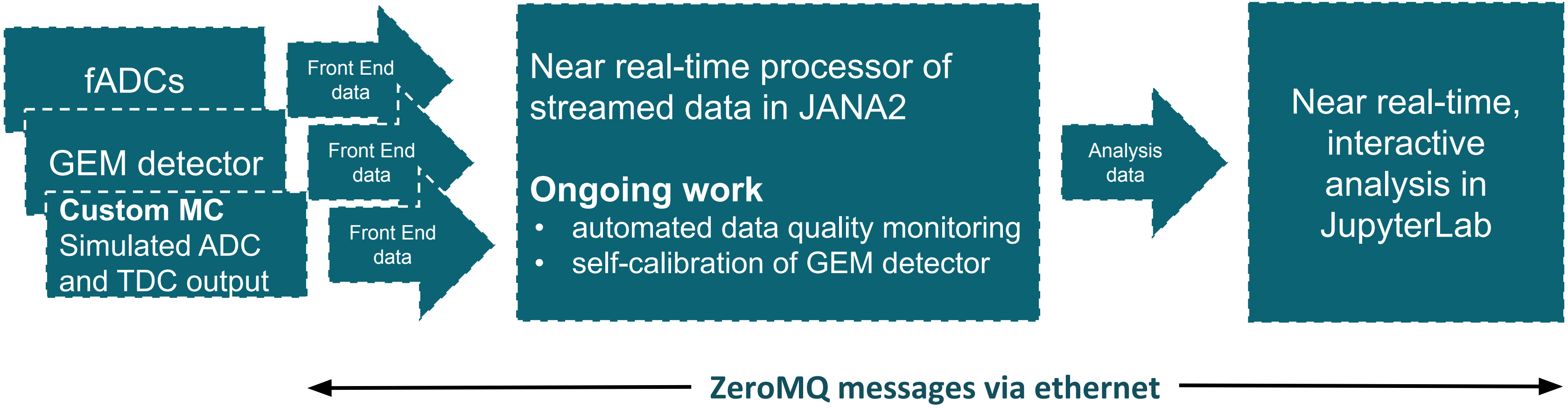


INDRA-ASTRA: Seamless integration of DAQ and analysis using AI

LDRD goal

prototype components of streaming readout at NP experiments
→ integrated start to end system from detector read out through analysis
→ comprehensive view: no problems pushed into the interfaces

prototype (near) real-time analysis of NP data
→ inform design of new NP experiments



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**

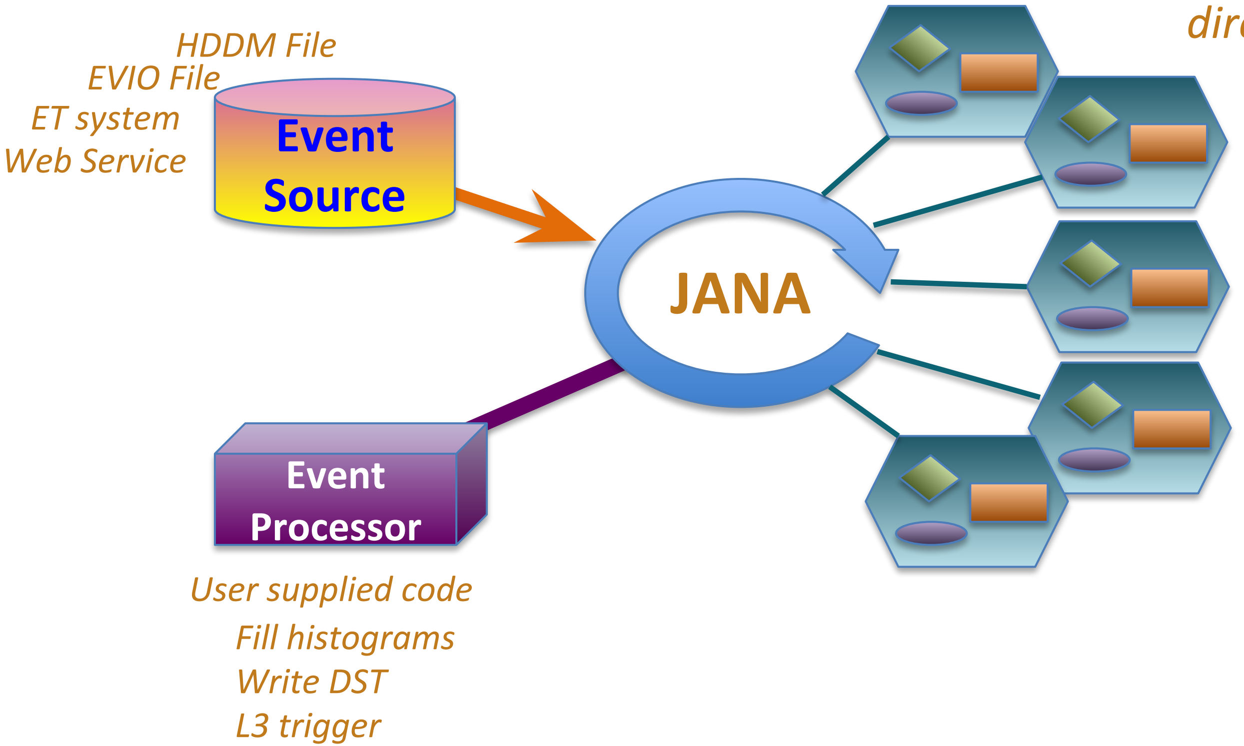
Anticipate 2018 data will be processed by end of summer 2019

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**

Event size:
12-13kB

Complete Event Reconstruction in JANA



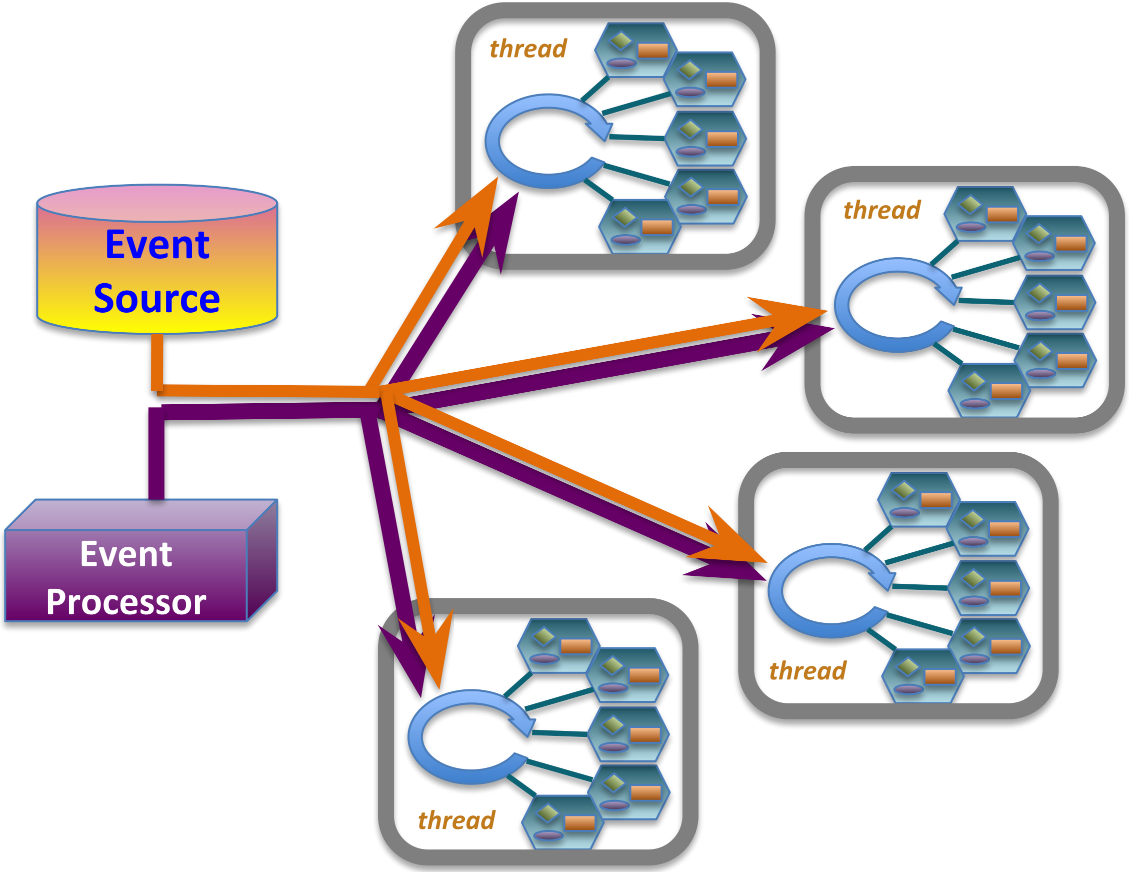
Framework has a layer that directs object requests to the factory that completes it

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

- A complete set of factories is assigned to an event giving it exclusive use while that event is processed
- 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)



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:
(i.e. algorithm)

```
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.

TOPOLOGY STATUS

Thread team size [count]:	4
Total uptime [s]:	50.09
Uptime delta [s]:	0.5002
Completed events [count]:	587
Inst throughput [Hz]:	14
Avg throughput [Hz]:	11.7
Sequential bottleneck [Hz]:	335
Parallel bottleneck [Hz]:	11.9
Efficiency [0..1]:	0.986

Name	Status	Type	Par	Threads	Chunk	Thresh	Pending	Completed
dummy_evt_src	Running	Source	F	0	16	-	-	672
processors	Running	Sink	T	4	1	500	81	587

Name	Avg latency [ms/event]	Inst latency [ms/event]	Queue latency [ms/visit]	Queue visits [count]	Queue overhead [0..1]
dummy_evt_src	2.98	1.03	0.00415	42	8.71e-05
processors	337	321	0.00883	1450	6.48e-05

ID	Last arrow name	Useful time [ms]	Retry time [ms]	Idle time [ms]	Scheduler time [ms]	Scheduler visits [count]
0	processors	623	0	0	0.000576	76
1	processors	622	0	0	0.000624	138
2	processors	668	0	0	0.000553	131
3	processors	734	0	0	0.000606	125

JANA2 now has much better built-in diagnostics compared to the original JANA.

This helps pinpoint bottlenecks, especially in more complex systems