

artdaq Introduction and Status

artdaq is a toolkit for creating DAQ systems to be run on commodity servers.

- Integrated with the *art* event reconstruction and analysis framework for event filtering and data compression.

Questions that I hope to answer in these slides:

- What functions does it provide?
- What is (or will be) available when?
- What would a prospective experiment need to provide?
- What choices does it impose?

Other topics:

- Future plans
- Demo?

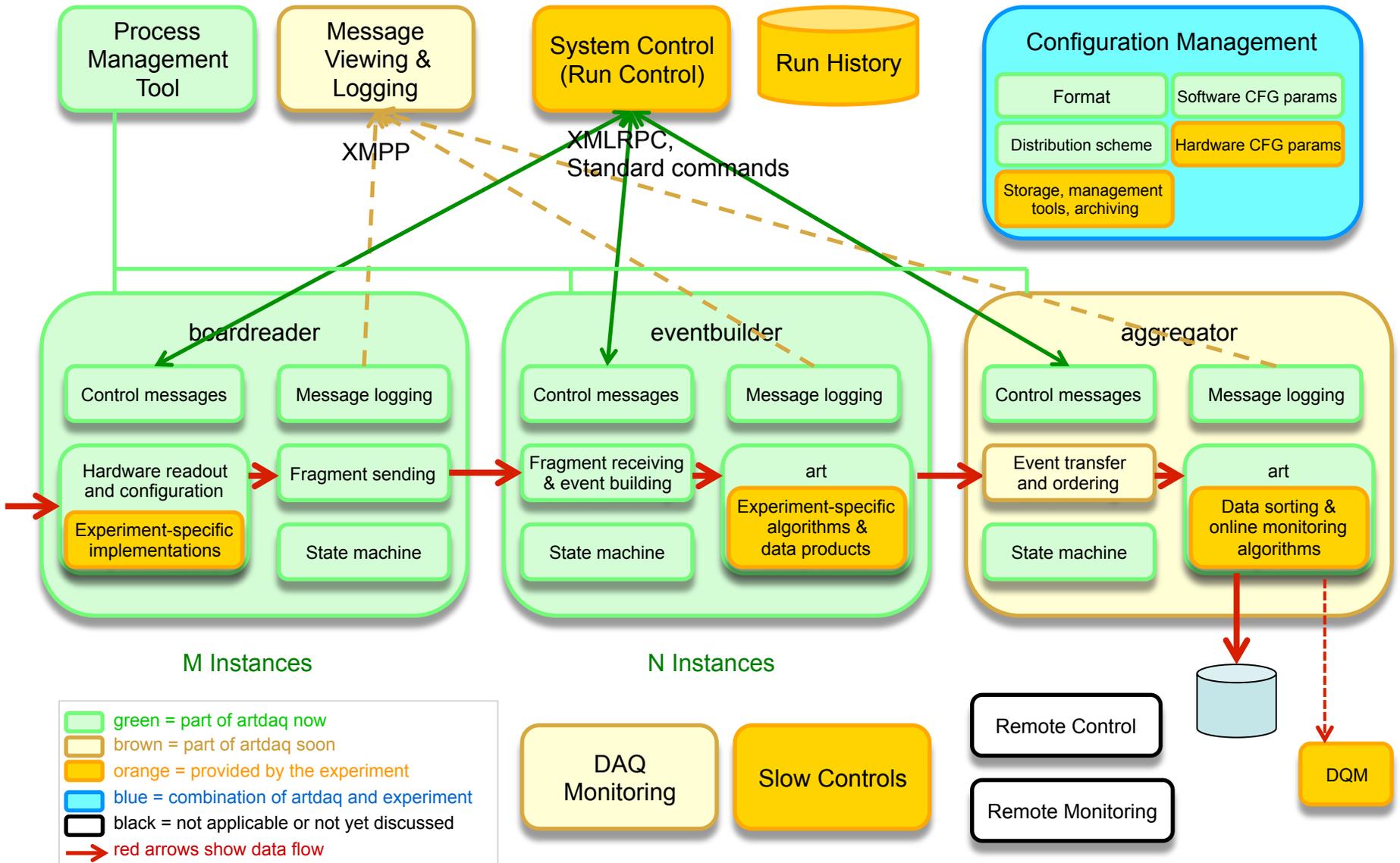
Typical DAQ functionality & components

Data movement and filtering	Control	Monitoring
<ul style="list-style-type: none">• Hardware configuration and readout• Data transfer<ul style="list-style-type: none">– Library(ies)– Transport layer• Event building• Reconstruction and filtering<ul style="list-style-type: none">– Framework– Analysis modules– Experiment-supplied data products• Data logging<ul style="list-style-type: none">– Generic output modules– Experiment-specific output modules• Process state behavior• Standard applications• Generic data components• Raw data format	<ul style="list-style-type: none">• System control<ul style="list-style-type: none">– Central application (Run Control)– System state model and/or supported control commands– Message protocol– Client library(ies)• Process management• Configuration management<ul style="list-style-type: none">– Format– Distribution scheme– Storage; management tools; archiving– Software configuration parameter definition– Hardware configuration parameter definition• Run history archiving• Slow controls• Remote control	<ul style="list-style-type: none">• Distributed message logging<ul style="list-style-type: none">– Logging libraries– Transport layer– Central viewing and logging applications• DAQ monitoring (health and performance of the DAQ system)<ul style="list-style-type: none">– Client library– Central viewing and logging facilities• Data Quality Monitoring (DQM – monitoring the quality of the data and the performance of the detector)<ul style="list-style-type: none">– Mechanics of making the data available– Analysis software– Viewers for the analysis results• Remote monitoring

artdaq for DarkSide-50

Data movement and filtering	Control	Monitoring
<ul style="list-style-type: none">• Hardware configuration and readout (ADC, trigger, TDC)• Data transfer<ul style="list-style-type: none">– Library(ies)– Transport layer (MPI)• Event building• Reconstruction and filtering<ul style="list-style-type: none">– Framework (art)– Analysis modules (compression)– Experiment-supplied data products• Data logging<ul style="list-style-type: none">– Generic output modules (ROOT)– Experiment-specific output modules• Process state behavior (SMC)• Standard applications (boardreader, eventbuilder, aggregator)• Generic data components (RawEvent, Fragment)• Raw data format	<ul style="list-style-type: none">• System control<ul style="list-style-type: none">– Central application (Run Control)– System state model and/or supported control commands– Message protocol (XMLRPC)– Client library(ies)• Process management (PMT – uses mpirun – easy configuration)• Configuration management<ul style="list-style-type: none">– Format (FHICL)– Distribution scheme (sent with control commands)– Storage; management tools; archiving– Software configuration parameter definition– Hardware configuration parameter definition• Run history archiving• Slow controls• Remote control	<ul style="list-style-type: none">• Distributed message logging<ul style="list-style-type: none">– Logging libraries (MsgFac)– Transport layer (XMPP)– Central viewing and logging applications (MsgFac)• DAQ monitoring (health and performance of the DAQ system)<ul style="list-style-type: none">– Client library– Central viewing and logging facilities (Ganglia)– Custom metrics• Data Quality Monitoring (DQM – monitoring the quality of the data and the performance of the detector)<ul style="list-style-type: none">– Mechanics of making the data available (art with special input/output modules)– Analysis software– Viewers for the analysis results• Remote monitoring
Key: part of artdaq: green; provided by experiment: orange; combination of artdaq and experiment: blue; not applicable or not yet discussed:black.		

artdaq for DarkSide-50



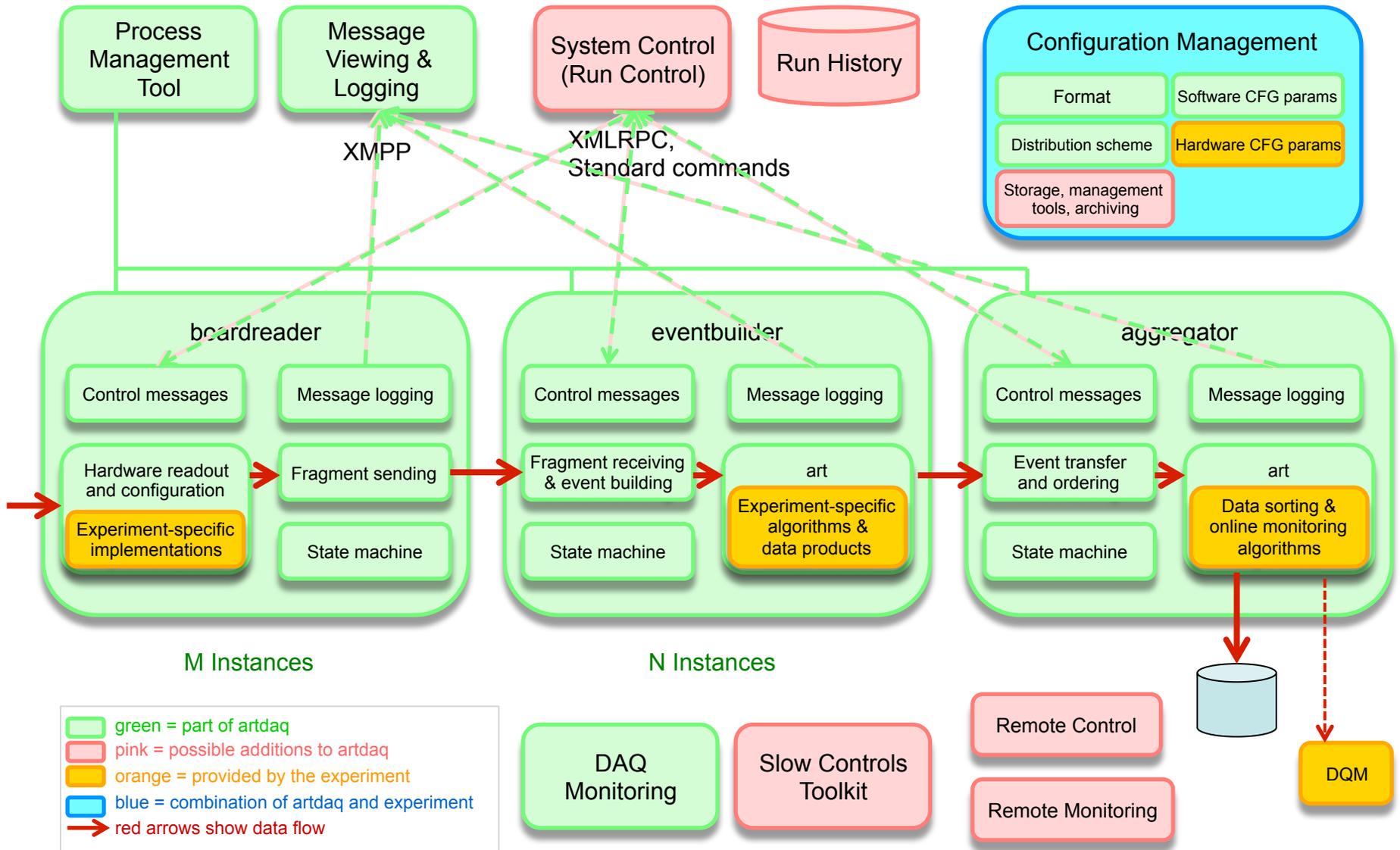
General *artdaq* – available soon

Data movement and filtering	Control	Monitoring
<ul style="list-style-type: none">• Hardware configuration and readout (sample provided)• Data transfer<ul style="list-style-type: none">– Library(ies)– Transport layer (MPI)• Event building• Reconstruction and filtering<ul style="list-style-type: none">– Framework (art)– Analysis modules (sample compression algorithm supplied)– Experiment-supplied data products• Data logging<ul style="list-style-type: none">– Generic output modules (ROOT)– Experiment-specific output modules• Process state behavior (SMC)• Standard applications (boardreader, eventbuilder, aggregator)• Generic data components (RawEvent, Fragment)• Raw data format	<ul style="list-style-type: none">• System control<ul style="list-style-type: none">– Central application (Run Control)– System state model and/or supported control commands– Message protocol (XMLRPC)– Client library(ies)• Process management (PMT – uses mpirun – easy configuration)• Configuration management<ul style="list-style-type: none">– Format (FHICL)– Distribution scheme (sent with control commands)– Storage; management tools; archiving– Software configuration parameter definition– Hardware configuration parameter definition• Run history archiving• Slow controls• Remote control	<ul style="list-style-type: none">• Distributed message logging<ul style="list-style-type: none">– Logging libraries (MsgFac)– Transport layer (XMPP)– Central viewing and logging applications (MsgFac)• DAQ monitoring (health and performance of the DAQ system)<ul style="list-style-type: none">– Client library– Central viewing and logging facilities (Ganglia)– Custom metrics• Data Quality Monitoring (DQM – monitoring the quality of the data and the performance of the detector)<ul style="list-style-type: none">– Mechanics of making the data available (art with special input/output modules)– Analysis software– Viewers for the analysis results• Remote monitoring
<p>Key: part of artdaq “now”: green; part of artdaq “soon”: gold; provided by experiment: orange; combination of artdaq and experiment: blue; not applicable or not yet discussed:black.</p>		

Future *artdaq* Ideas

Data movement and filtering	Control	Monitoring
<ul style="list-style-type: none"> • Hardware configuration and readout (sample provided) • Data transfer <ul style="list-style-type: none"> – Library(ies) – Transport layer (MPI) • Event building • Reconstruction and filtering <ul style="list-style-type: none"> – Framework (art) – Analysis modules (sample compression algorithm supplied) – Experiment-supplied data products • Data logging <ul style="list-style-type: none"> – Generic output modules (ROOT) – Experiment-specific output modules • Process state behavior (SMC) • Standard applications (boardreader, eventbuilder, aggregator) • Generic data components (RawEvent, Fragment) • Raw data format 	<ul style="list-style-type: none"> • System control <ul style="list-style-type: none"> – Central application (Run Control) – System state model and/or supported control commands – Message protocol (XMLRPC, other options) – Client library(ies) • Process management (PMT – uses mpirun – easy configuration) • Configuration management <ul style="list-style-type: none"> – Format (FHICL) – Distribution scheme (sent with control commands, other options) – Storage; management tools; archiving – Software configuration parameter definition – Hardware configuration parameter definition • Run history archiving • Slow control toolkit • Remote control (VNC instructions) 	<ul style="list-style-type: none"> • Distributed message logging <ul style="list-style-type: none"> – Logging libraries (MsgFac) – Transport layer (XMPP, other options) – Central viewing and logging applications (MsgFac) • DAQ monitoring (health and performance of the DAQ system) <ul style="list-style-type: none"> – Client library – Central viewing and logging facilities (Ganglia) – Custom metrics • Data Quality Monitoring (DQM – monitoring the quality of the data and the performance of the detector) <ul style="list-style-type: none"> – Mechanics of making the data available (art with special input/output modules) – Analysis and display toolkit • Remote monitoring (Screen Snapshot Service instructions)
<p>Key: part of artdaq “now/soon”: green; possible additions to artdaq: red; provided by experiment: orange; combination of artdaq and experiment: blue; not applicable or not yet discussed:black.</p>		

Future *artdaq* Ideas



Existing choices

Control commands are executed synchronously – a configurable timeout value keeps them from running forever.

The state behavior is part of the core implementation...

- A finite set of control commands is supported
- Process state behavior is fixed (part of the artdaq)

Supported external commands

Commands which affect the state of the process:

- **init**(string ParameterSet) - initializes (configures) the process (and any associated hardware); returns a success or failure report
- **start**(integer runNumber) - begins a run; returns a success or failure report
- **stop**() - ends a run; returns a success or failure report
- **pause**() - pauses data taking during a run; returns a success or failure report
- **resume**() - resumes data taking during a run; returns a success or failure report
- **shutdown**() - prepares the process to be stopped (un-initialize); returns a success or failure report
- **soft_init**(string ParameterSet) - initializes (configures) some fraction of the software components; returns a success or failure report
- **reinit**(string ParameterSet) - re-initializes parts of the software (or the hardware) during a run; returns a success or failure report

Commands which gather information:

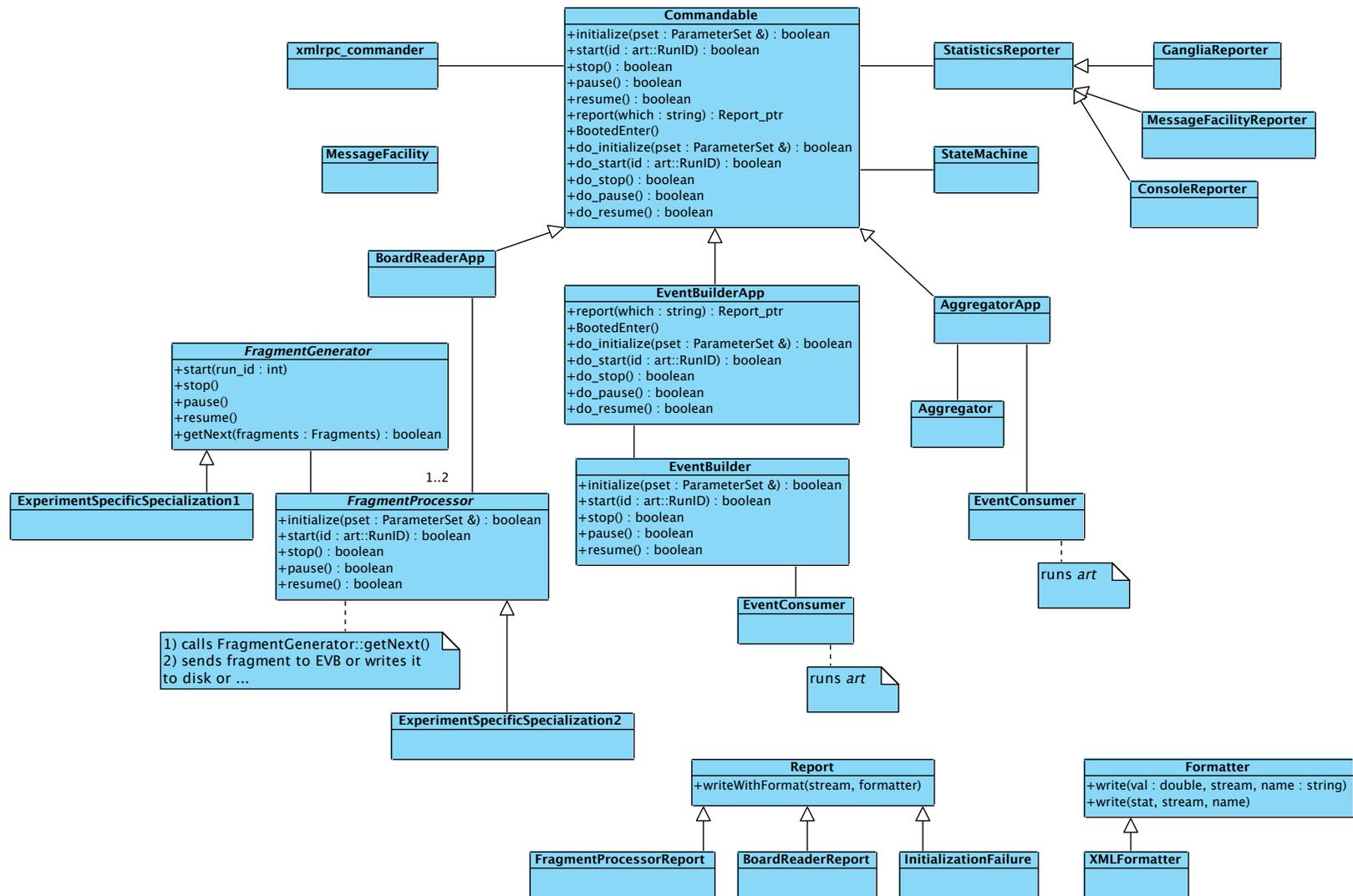
- **report**(string which) - returns statistics or error reports from some or all of the components in the process. The "which" argument specifies which statistics to report.
- **status**() - returns the current externally visible "state"
- **reset_stats**(string which) - resets some or all of the statistics in the process. The "which" argument specifies which statistics to reset. Returns a success or failure report.
- **legal_commands**() - returns the subset of the external commands which are currently legal given the state of the process

Externally visible states

The externally visible states are currently supported are listed here along with the external commands that are allowed in each state:

- Booted
 - init(pset)
- Ready
 - init(pset)
 - soft_init(pset)
 - start(runNumber)
 - shutdown()
- Running
 - pause()
 - stop()
 - init(pset)
 - soft_init(pset)
 - reinit(pset)
- Paused
 - resume()
 - stop()
 - init(pset)
 - soft_init(pset)
 - reinit(pset)
- Error
 - init(pset)

Preliminary *artdaq* class diagram



Details

We will be creating an initial release of the ds50daq code soon. It will be packaged and deployed as a relocatable UPS product, similar to how *artdaq* and *art* are packaged and deployed.

We could move the reusable pieces of ds50daq into *artdaq* with a couple of weeks of notice and provide that to users as a UPS product.

In addition, we would like to create an *artdaq-sample* UPS product (and git repository) that would demonstrate how a new experiment would create the experiment-specific pieces of a system that uses *artdaq*.

Current and future users

NOvA Data-Driven Trigger

- *art* and a few classes from *artdaq* are being used to run trigger algorithms in the buffer farm.

DarkSide-50:

- *artdaq* will be used, as shown in an earlier slide.

uBooNE:

- At the moment, individual classes from the *artdaq* library are being used in the data transfer.

Mu2e:

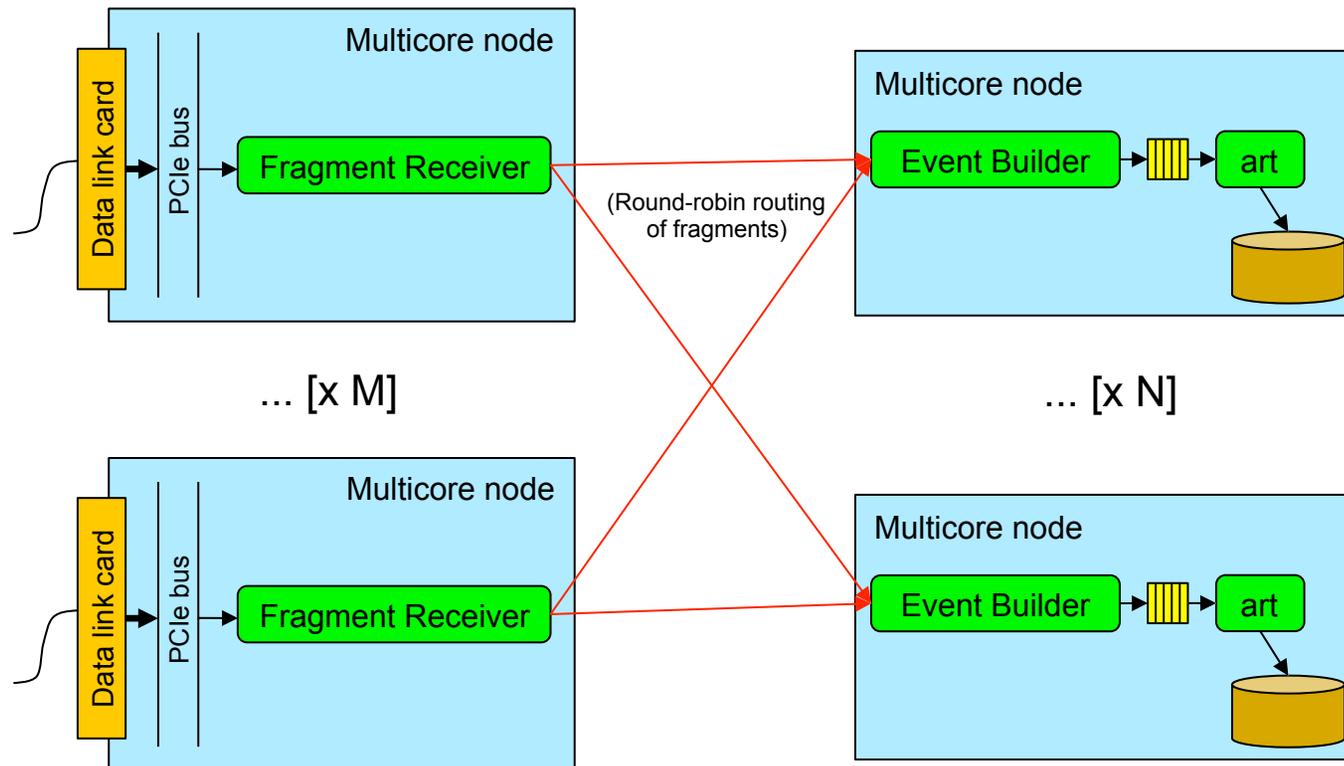
- *artdaq* will be used for initial DAQ development and testing with an eye toward using it in the full DAQ. (The current baseline design uses FPGAs to do the event building, but *artdaq* may provide a more cost-effective solution.)

Other candidates:

- Minerva? LBNE, ORKA...

Backup slides

Generic DAQ



Lots of variations:

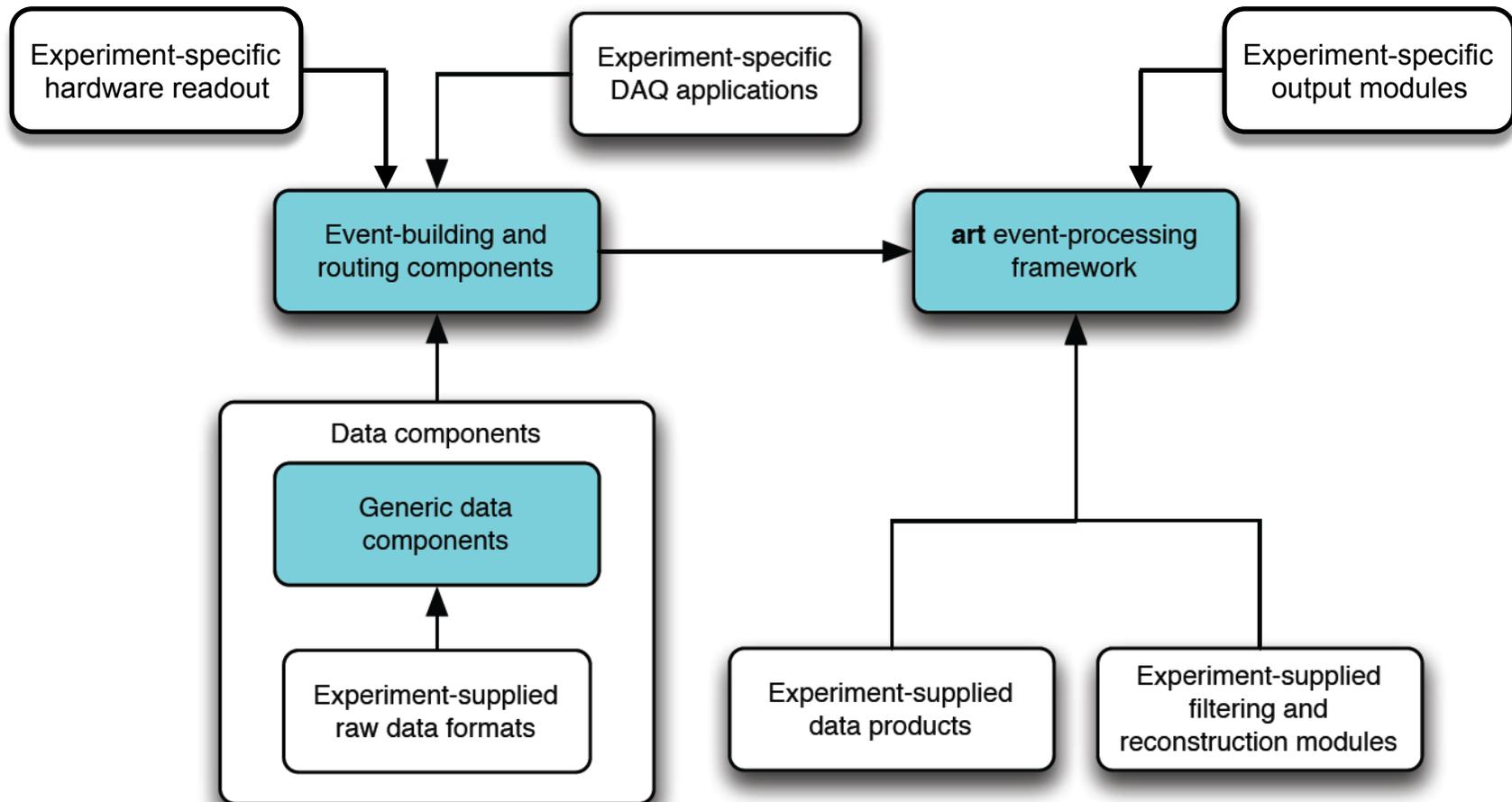
- multiple fragment receivers per front-end node
- multiple event builder/**art** process pairs per reconstruction node
- (multiple **art** processes per event builder)
- everything run on a single node

A flexible configuration process makes testing and deployment easier.

Initial *artdaq* Goals

- Support the use of commodity computers as close to the data collection as possible.
- Make efficient use of multi-core computers.
- Take advantage of high-speed networking and hardware buses.
- Support modular algorithms, enable the use of GPGPUs.
- Enable collaborators to contribute to online code development.
- Allow for concurrent processing of events to best utilize all of the cores on a node.
- Support easy system reconfiguration.
- Allow for similar or identical algorithms to be used offline and online.
- Provide an environment for R&D tasks.
- Provide a springboard for DAQ development in future experiments.

Initial *artdaq* architecture (this has changed)

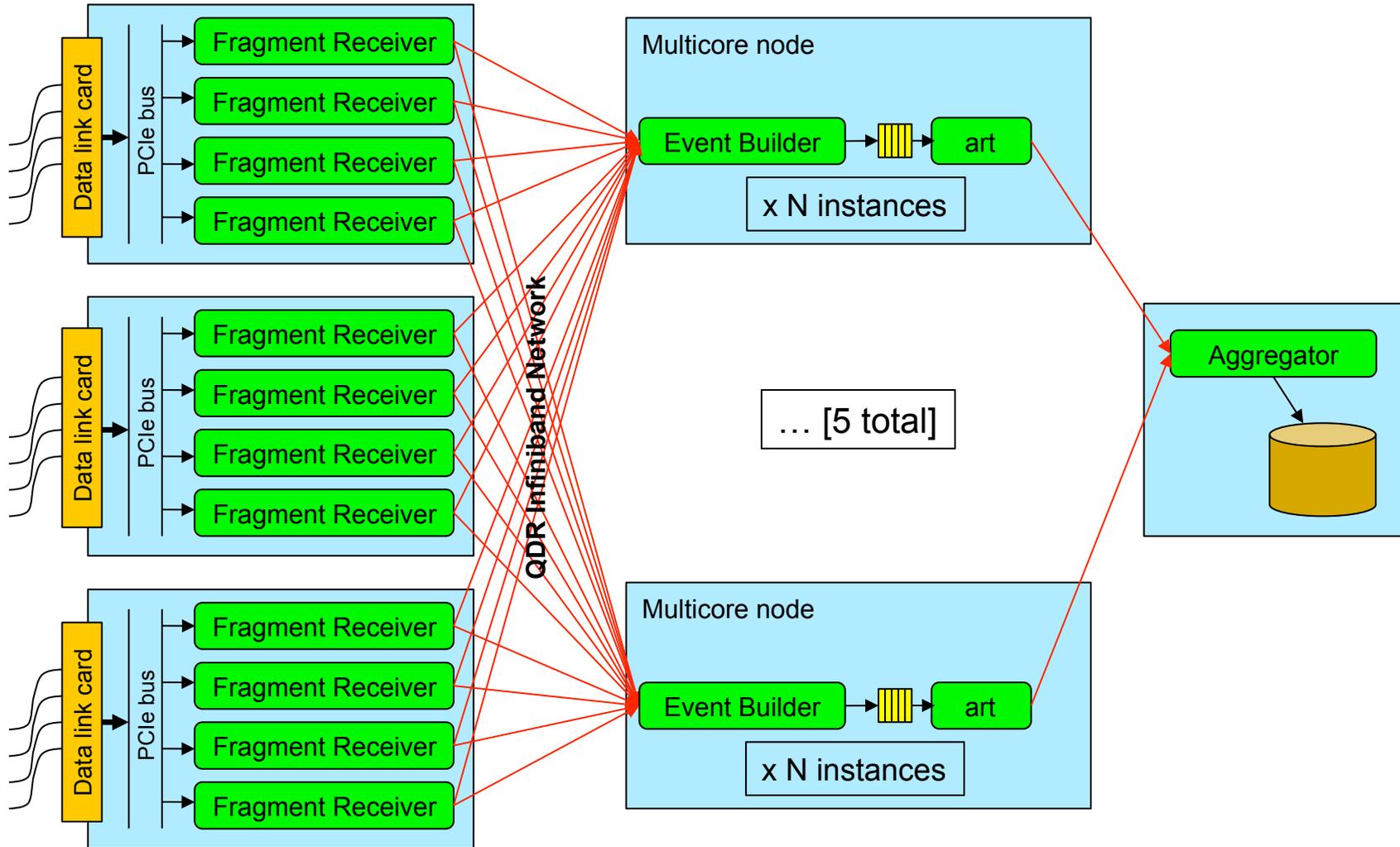


MPI and mpirun

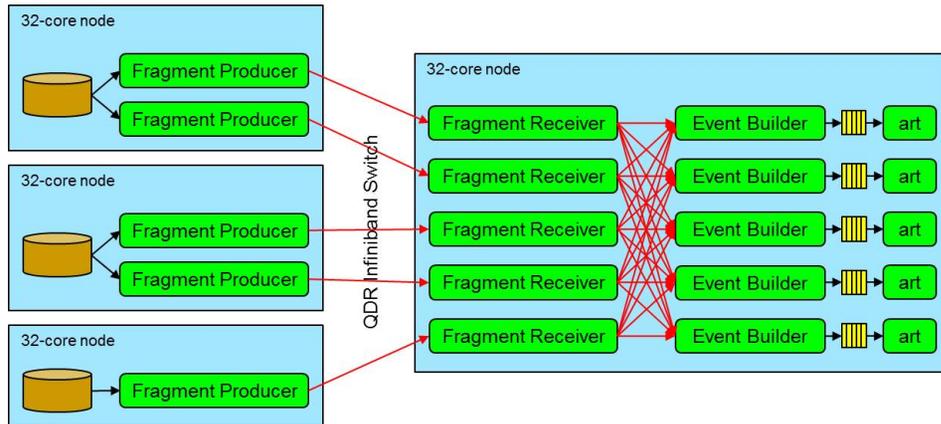
In *artdaq*, MPI (Message Passing Interface) is used to transfer data between the distributed processes and to manage the processes.

- MPI is “a library specification for message passing, designed for high performance on parallel machines and workstation clusters.”
- It supports point-to-point and collective messaging. It also supports parallel execution features such as synchronization between processes (e.g. all process wait until they have all reached a certain point in their execution).
- An MPI “program” contains all of the cooperating processes, and startup is handled by an agent (*mpirun*) that runs the program on a configurable set of nodes.
- In MPI-1, the same executable binary is used for all processes, and the different processes know which role to perform based on their “rank”.
- For the initial **artdaq** test system, we wrote a straightforward Python script to automate the configuration of the nodes and handle the invocation of *mpirun*.

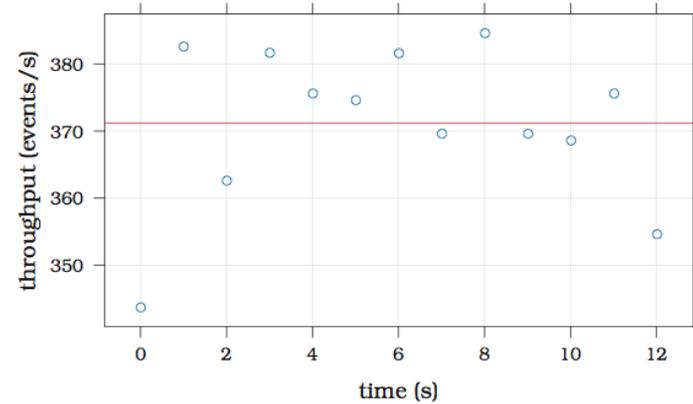
DarkSide-50 Architecture



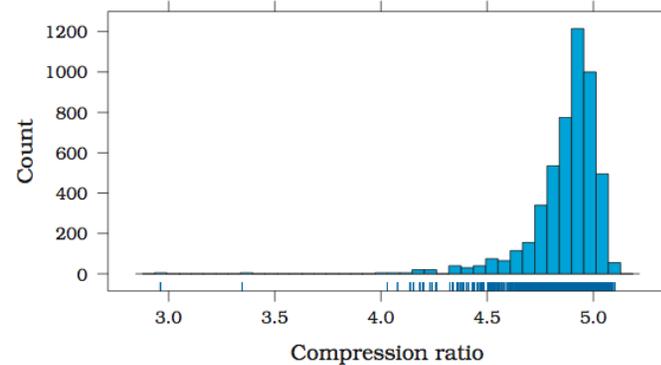
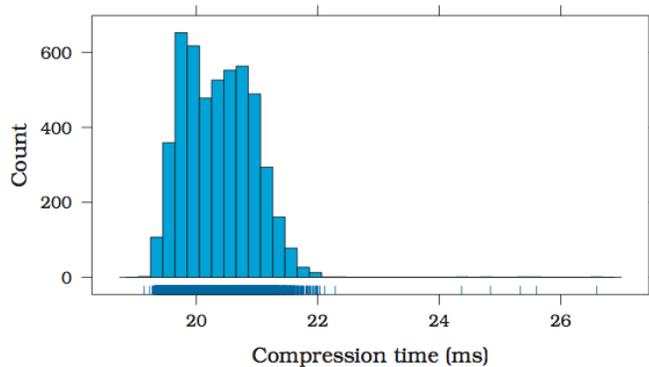
Performance Studies



To test the performance of a candidate compression algorithm for DarkSide-50, a cluster of four 32-core nodes connected by a QDR Infiniband switch was used. In this test, the detector electronics was simulated by a Linux process that read DS50 fragment data from disk. In this test, all five Fragment Receivers, all five Event Builders, and all five *art* reconstruction applications (with 5 threads – one per fragment) were run on the same host to make use of all of the 32 cores on the machine.



Throughput results, with no compression, as a function of time during the test. In the test, the event size was 6 MB, so the average throughput data rate was 2.2 GB/s.



Performance results from a test of a candidate Huffman compression algorithm. The compression time results are per event per *art* process, so the overall measured rate was ~250 events per second.

Additional Performance Results

Mu2e-like system

- In Mu2e, data will be received directly into the processor farm PCs, and the event building will be done between those PCs.
- Using a test cluster of five 32-core nodes connected by an InfiniBand QDR network, we measured the throughput of configurations where a FragmentReceiver process and an EventBuilder process ran on each node. The result was ~ 730 MB/s per node.
- To handle the 30 GB/s needed for Mu2e, this would translate into ~ 42 nodes (a reasonable number).

DarkSide-50 system

- When the DarkSide-50 DAQ machines were running here at Fermilab (before being shipped to LNGS), tests were run in which a FragmentReceiver process produced simulated data on each of 4 front-end computers. Studies were done to see how many EventBuilder processes were needed to handle the rate from the 4 FragmentReceivers. The results are shown in the graph on the right.

