Data AcQuisition Overview & Online Analysis

Photon Controls Data Systems

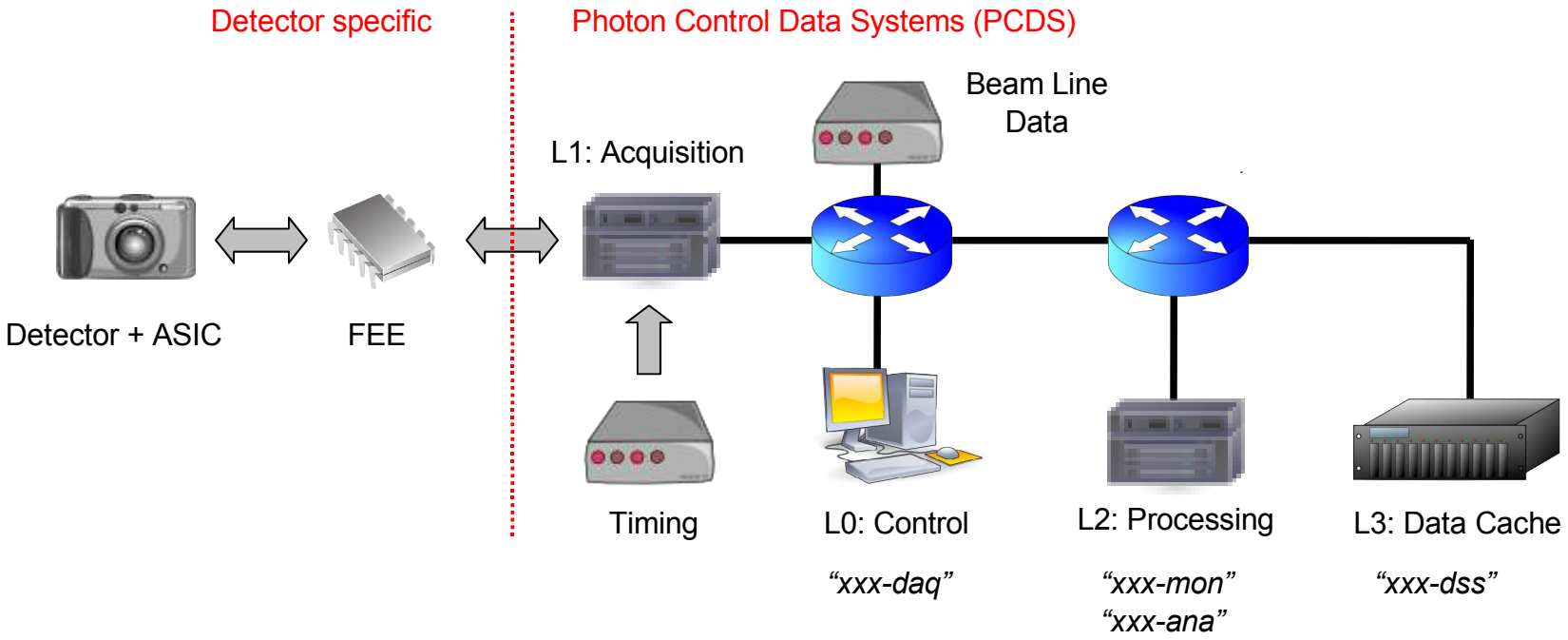
Data Acquisition Group

Chris Ford, Wilfred Ghonsalves,
Philip Hart, Chris O'Grady,
Jack Pines, Jana Thayer,
Tomy Tsai, Matt Weaver

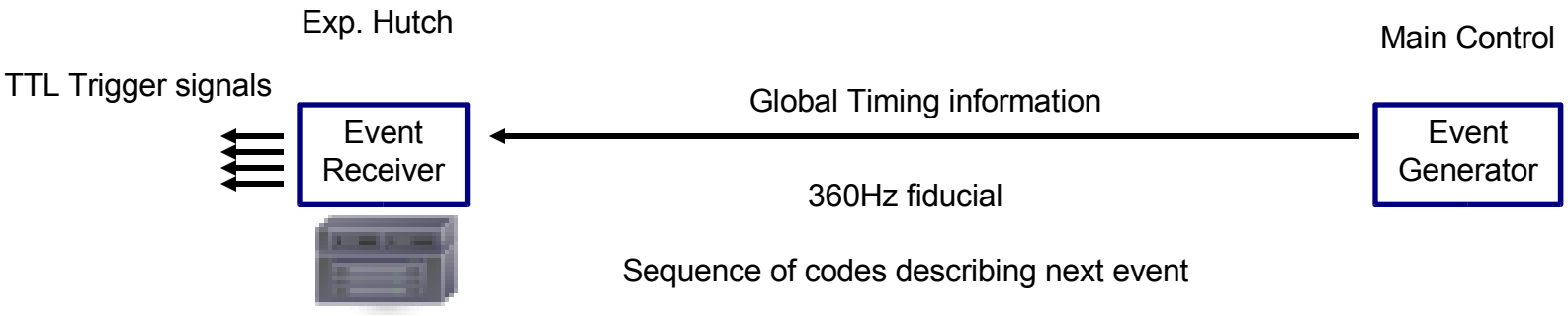
Outline

- DAQ Overview
  - Architecture
  - Operation
- Online Analysis
  - Core monitoring
  - User analysis

DAQ Architecture



Timing System : Generating Triggers



Sequence of codes describing next event

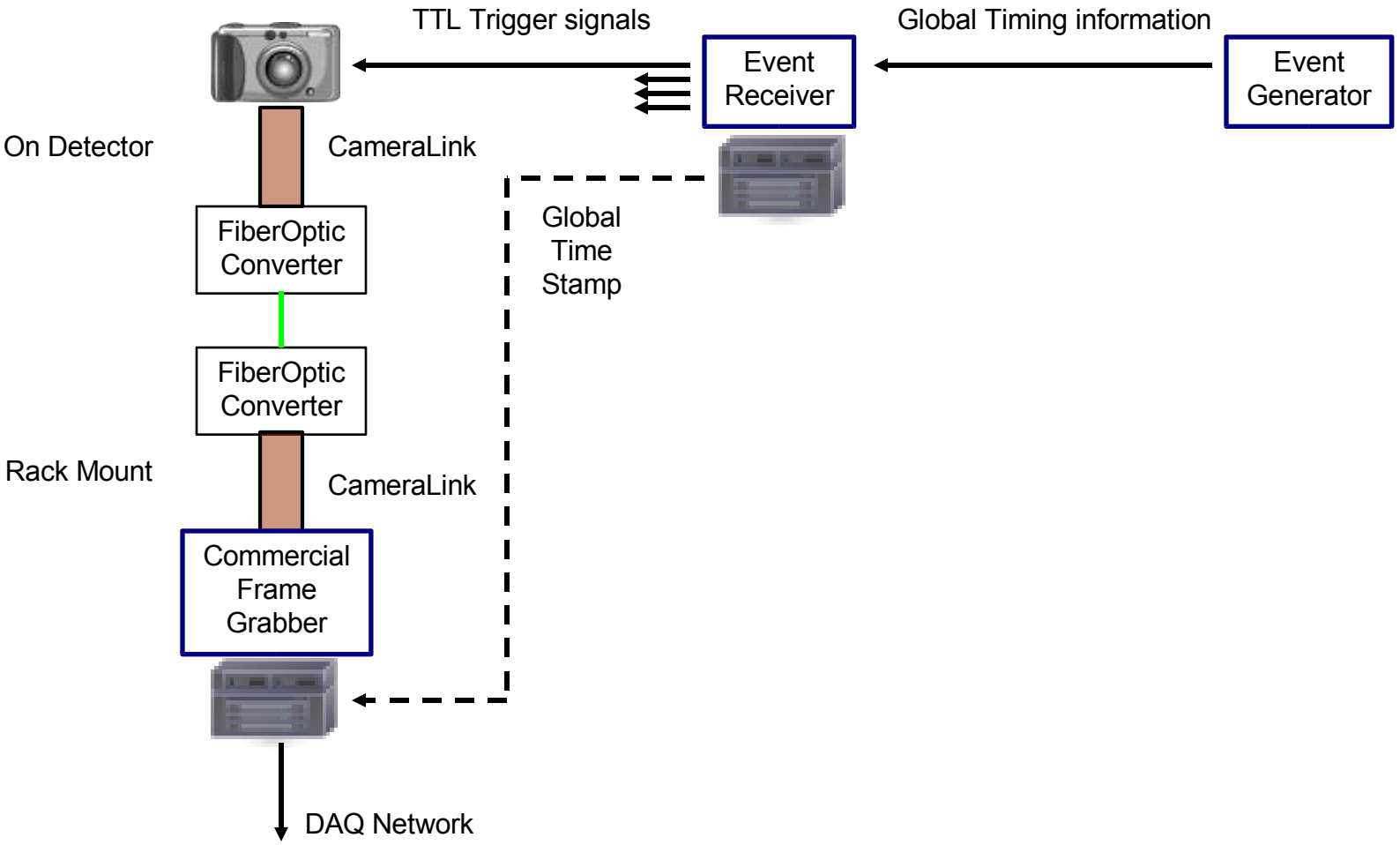
Each code occurs at a fixed 119 MHz timeslot with respect to the fiducial

Delivery of the beam is similarly synchronized to the fiducial

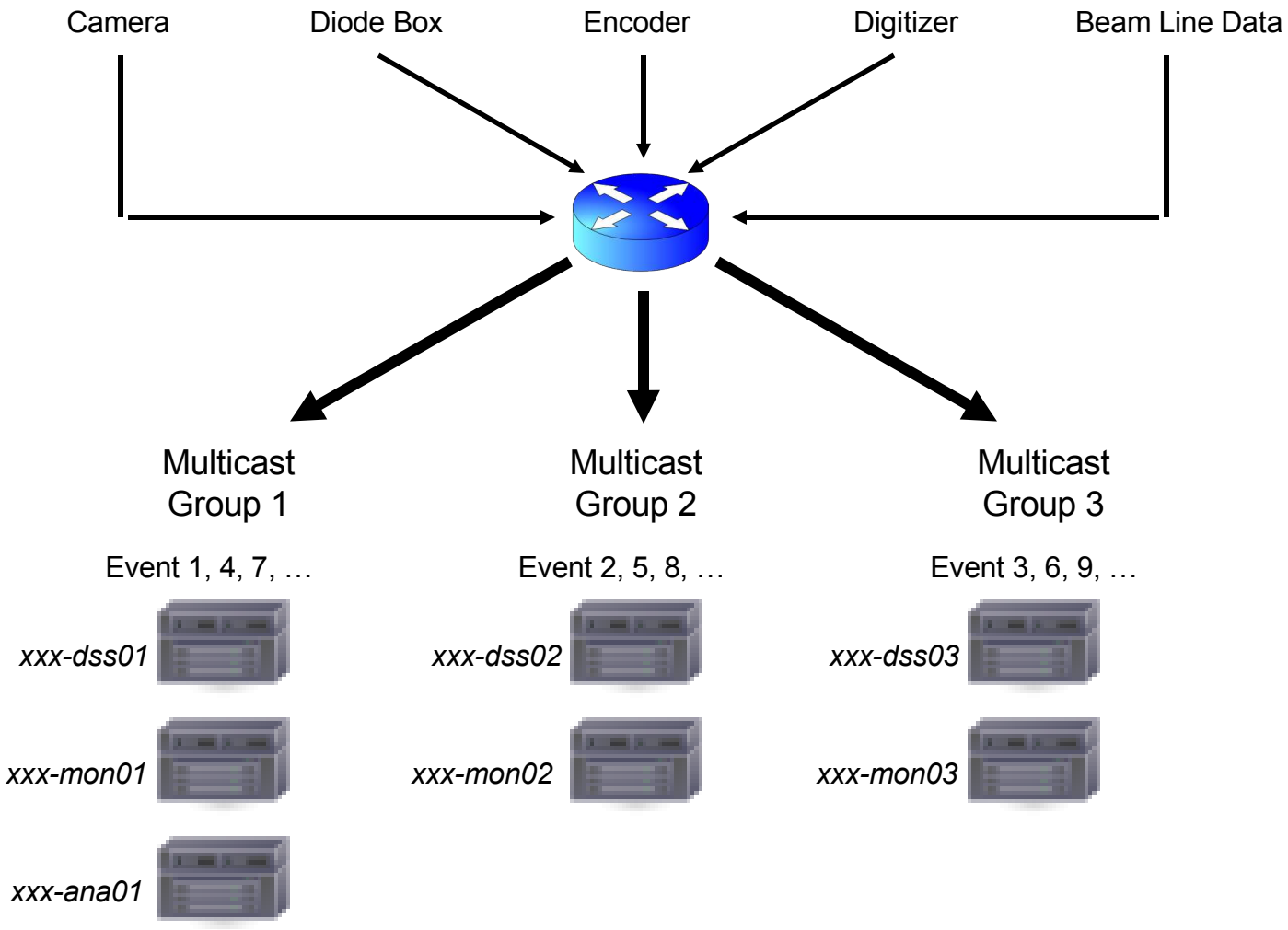
Includes a 17-bit fiducial identifier

Event codes coordinate control activities with the arrival of the beam

Example Detector Readout : Commercial Cameras



DAQ Network : Event Building



DAQ Control : Launching a "Run"

L0: Control

"xxx-daq"

Configure Triggers and Detector Readout

Go!

Watch Progress

DAQ Control : Configuring Triggers

L0: Control

"xxx-daq"

Enable	Event Code	Pulse Polarity	Pulse Delay (sec)	Pulse Width (sec)	CSPAD-1	CSPAD-2	SB1-IPM-1	SB2-IPM-1	END-OPAL-1	END-OPAL-2
☑	140	Pos	0.00089	1e-05	☑	☑	☐	☐	☐	☐
☑	140	Pos	0.000905	1.0084e-07	☐	☐	☑	☑	☐	☐
☑	140	Pos	0.0007	0.0003	☐	☐	☐	☐	☑	☑

Detector readout is triggered by arrival of an event code
(140 = beam on this fiducial)
to gate the sampling/digitization with a TTL signal

DAQ Control : Configuring Readout

L0: Control

"xxx-daq"

Most detector readout configurations are simple.
Few parameters require changing.

Data Processing

- Monitor node processing *("xxx-mon")*
  - Nodes register for event data multicasts
  - Receive each detector's data and assemble complete events
  - Copy events to shared memory and push event pointer into a queue for application consumption
  - If the queue is full (processing bottleneck), events are dropped
- Data cache node processing *("xxx-dss")*
  - Nodes register for event data multicasts
  - Receive each detector's data and assemble complete events
  - Record each event to online disk cache
  - Report event statistics to user console (#, size, health)
- Offline transfer
  - Run data is transferred from data cache nodes to offline storage
  - Transfer can start during recording as resources allow

Data Formats

- XTC (eXtended Tagged Container)
  - Online format : custom
  - Event-based (data belonging to same event are contiguous)
  - Serial access (but could be indexed)
  - Defined in “pdsdata” C++ package
- HDF5
  - Translated from XTC after the offline transfer
  - Random access
  - Standard

Online Analysis

- Core Online Monitoring
- User Shared Memory Application
- User Disk-Based Application

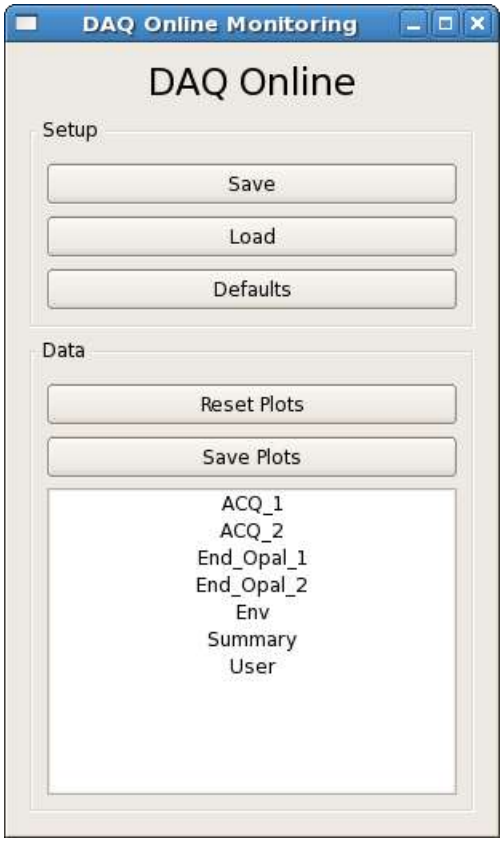
Core Online Monitoring

- Graphical display of raw data
  - Scalar data { diodes, encoders, beamline data }
  - Waveform data { sampling digitizers }
  - Image data { commercial and custom cameras }
- User configuration of fast, useful processing
  - Background subtraction
  - Sample/range selection
  - Projection
  - Filtering
  - Detector correlations
- Perform even more processing
  - User coding of specializations
  - Dynamic linking

Core Online Monitoring

Archive display setup (many plots)

List of primary event displays



Reset/Capture data for all plots (text format)

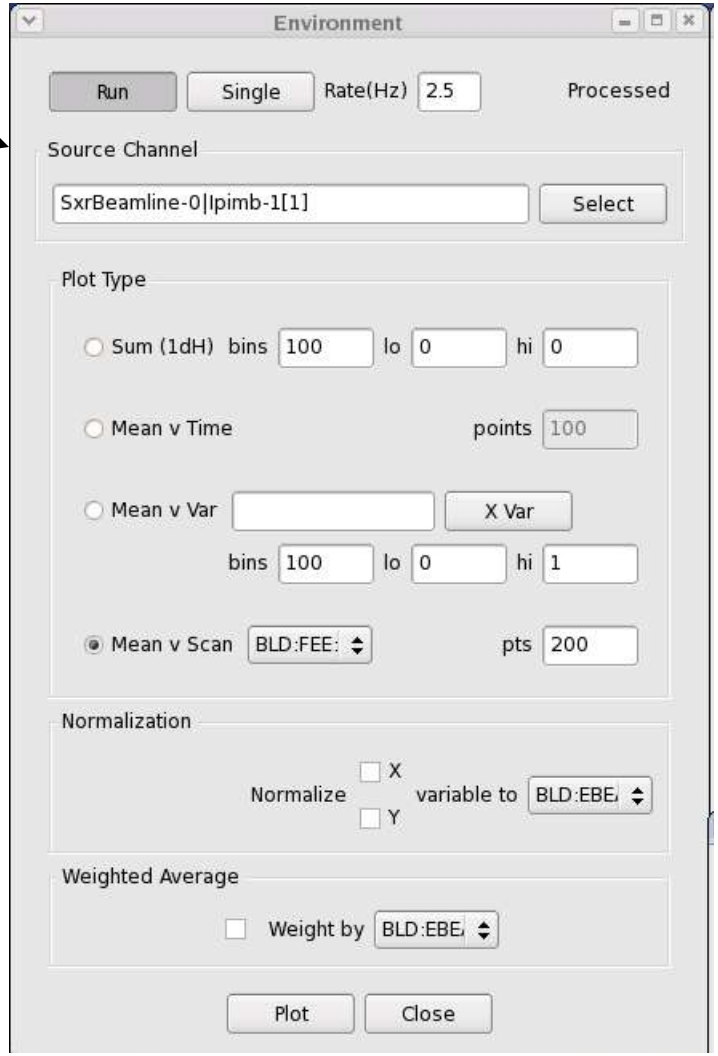
"Env" Display (scalars)

Channel selection

BeamLine Data
{ gas det.,
e- bunch msmt }

Detector Scalars
{ diodes, encoders }

Slow controls
{ motor positions,
voltages, … }



Plot Types

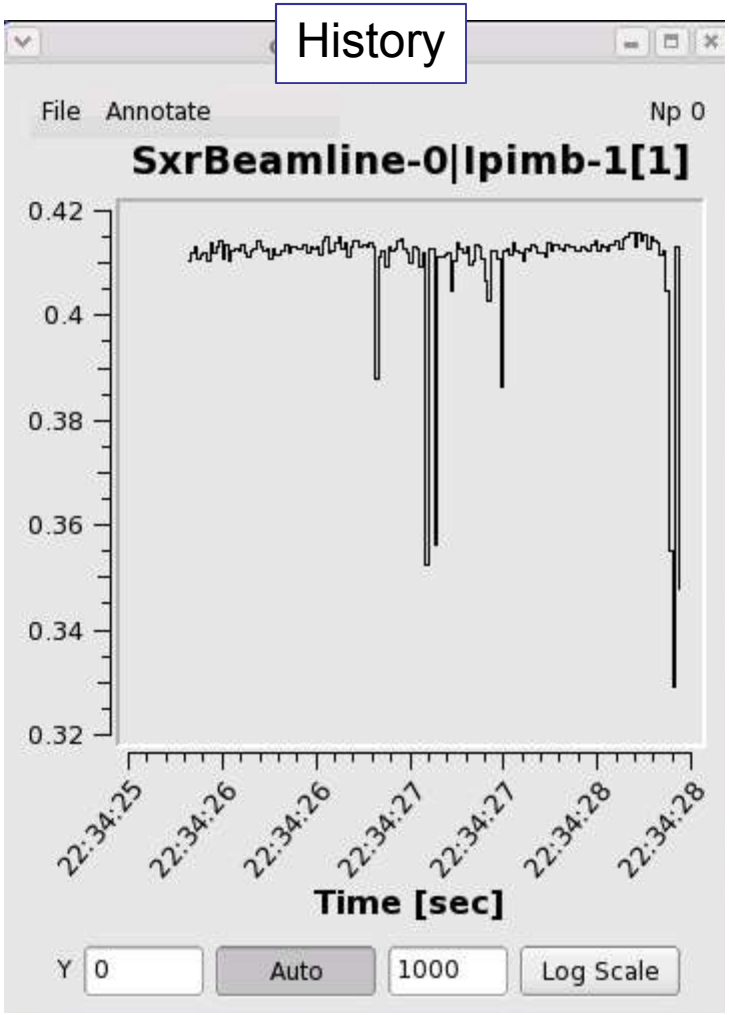
Frequency histogram

History stripchart

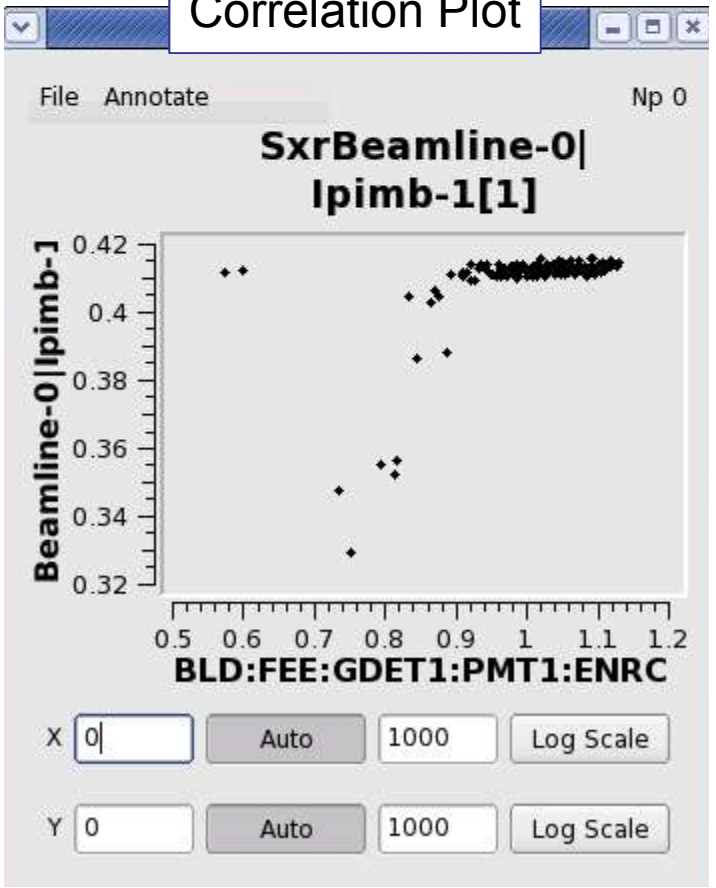
Correlation plot
(vs other scalars)

"Env" Display (scalars)

History



Correlation Plot



Waveform Display (Acqiris digitizer)

Multiple channels

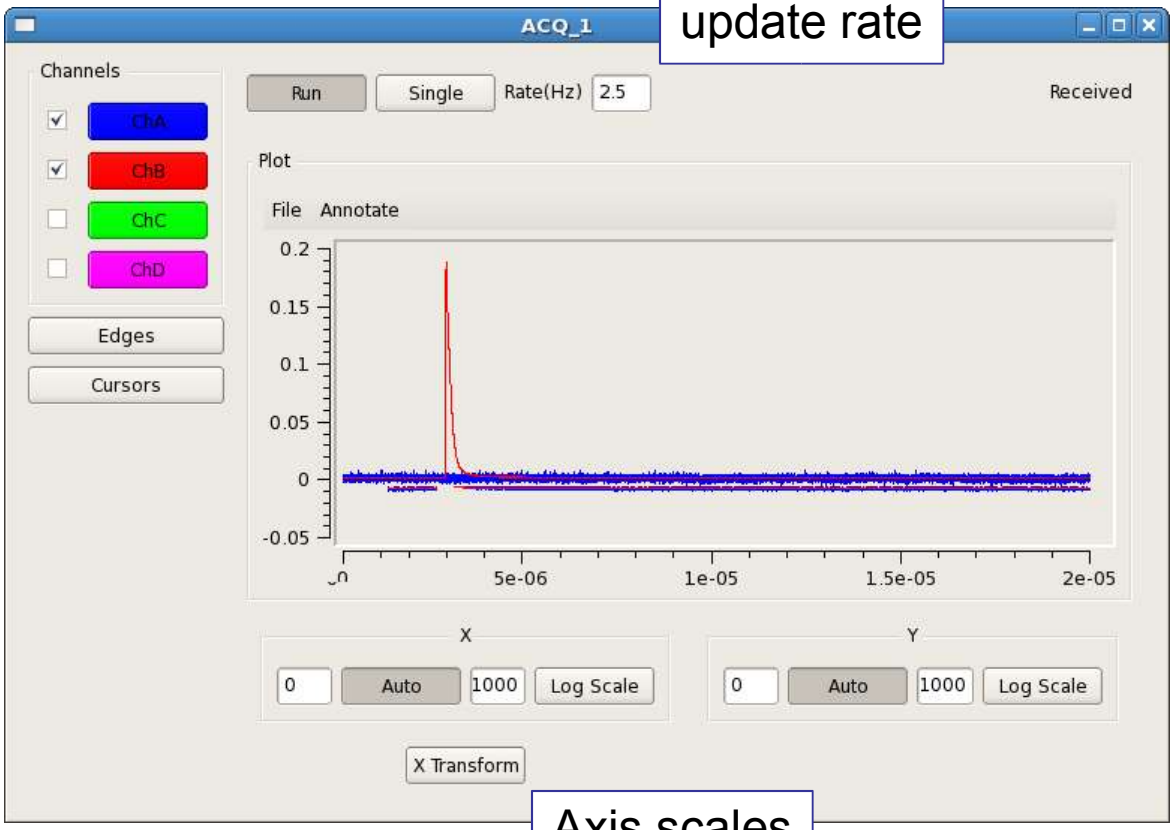
Superimpose reference, average, channel math, filter

Operations

Constant fraction discriminator

Sample/range selection

Display update rate



Axis scales

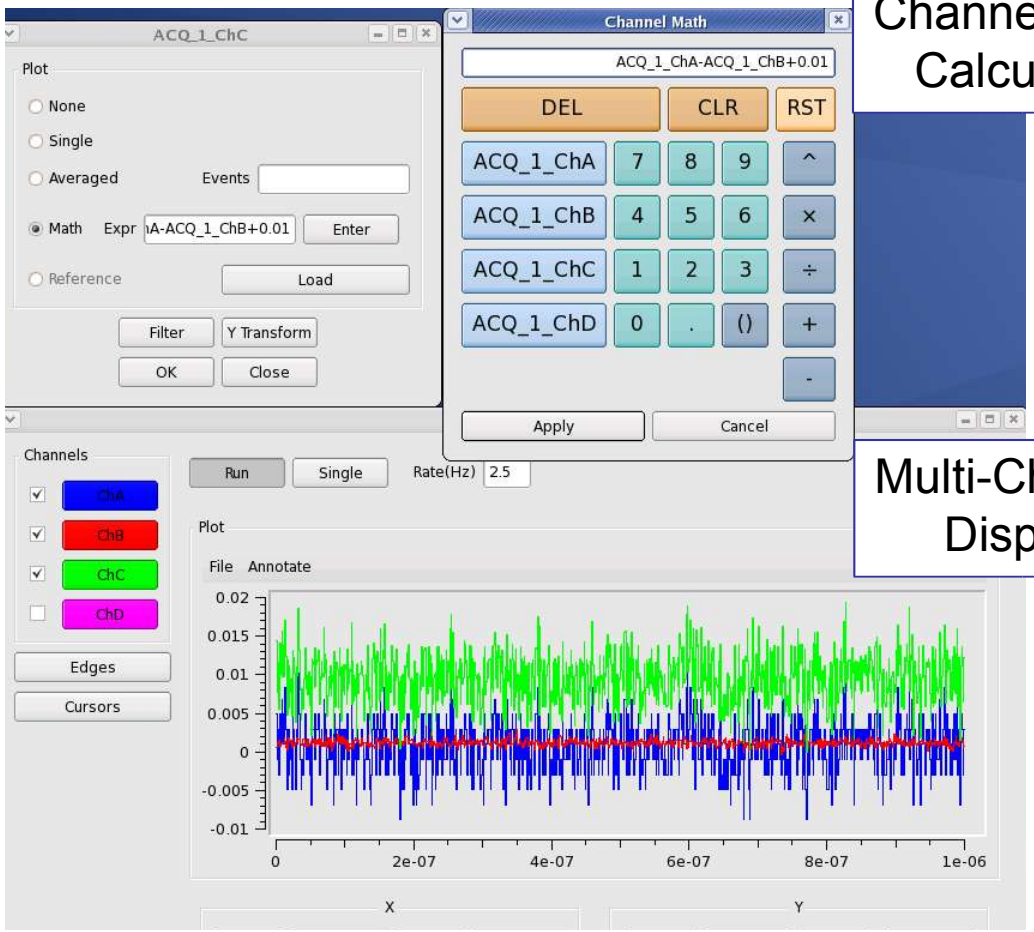
Channel Definitions

Single event

Average last N events

Arithmetic expr.

Reference wf



Channel Math Calculator

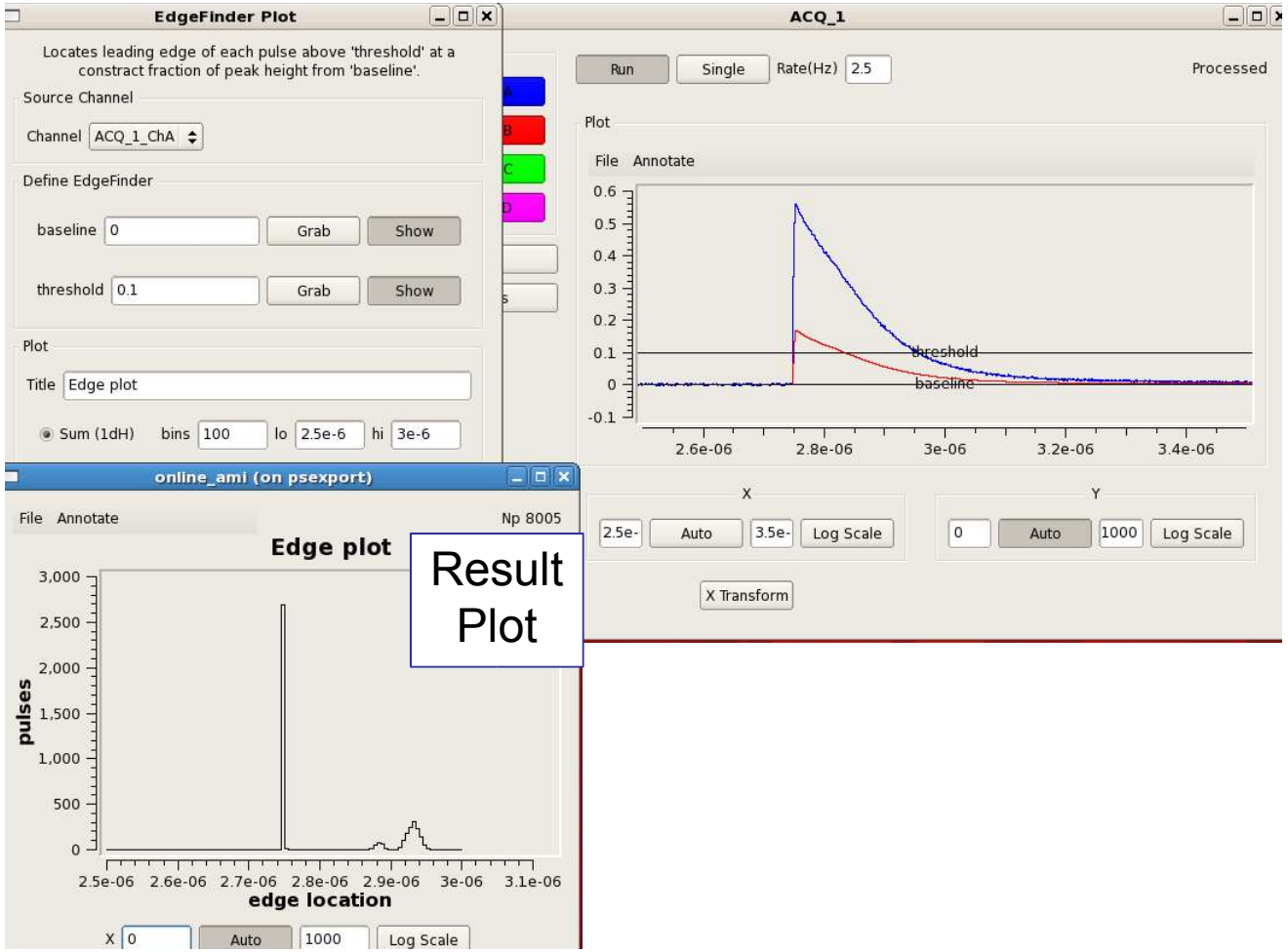
Multi-Channel Display

Operations – Constant Fraction Discr.

Channel selection

Discriminator settings

Result plot definition



Operations - Cursors

Define Cursors

Generate Expression

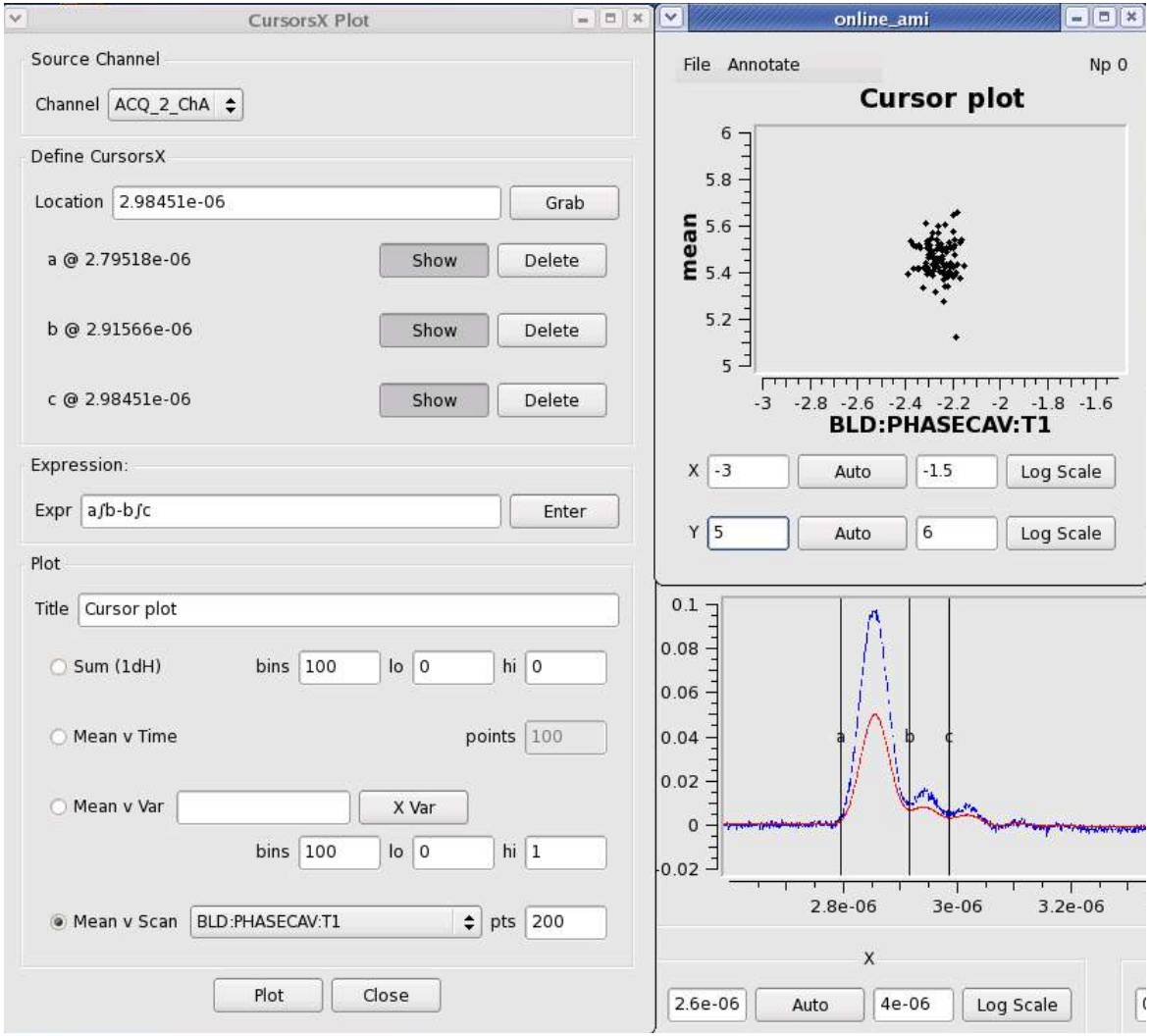
f (Value@cursor, integral btwn)

Creates a scalar result

Choose Plot

Same types as “Env” plots

Can correlate against any other “Env” value



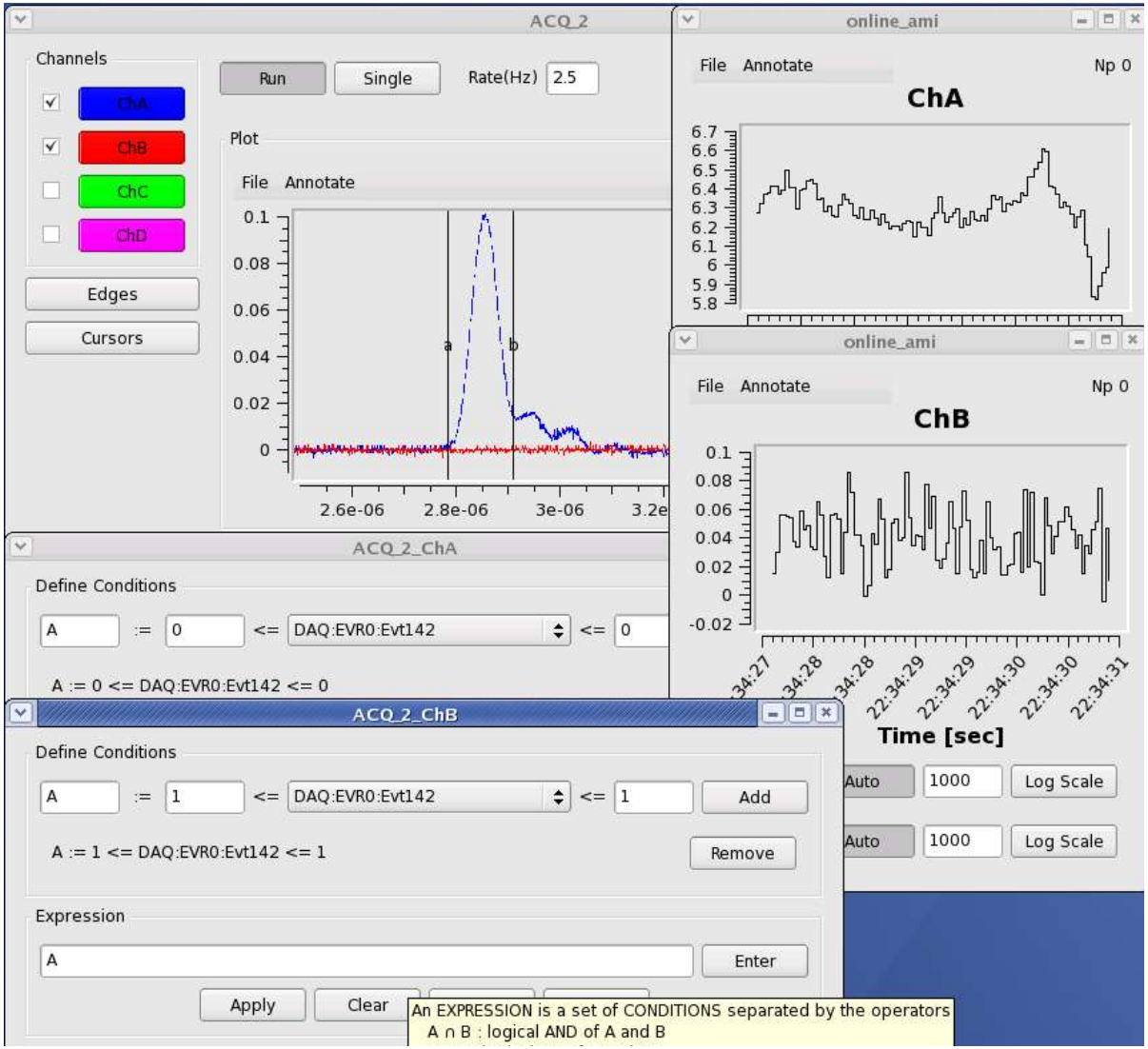
Operations – Cursors (contd)

- Scalar values which are generated by some operation can be correlated against the original set of scalar variables.
- Important application for scans.
- Appears in several places in the core monitoring interface.

Operations – Channel Filters

Can filter on logical combinations of “Env” scalar ranges

Example: eventcode indicating presence of optical laser



Camera Display (custom CCD)

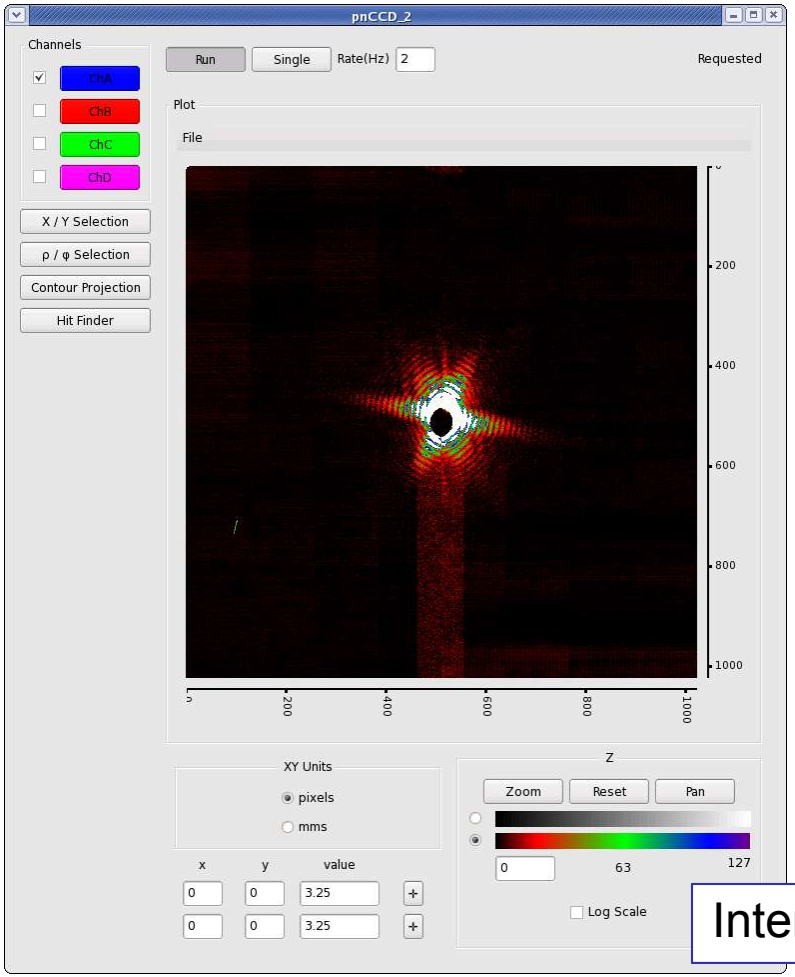
Operations

Rectangular selection/projection

Annular selection/projection

Contour projection

Hit mapping



Intensity scale

Operations – X/Y projection

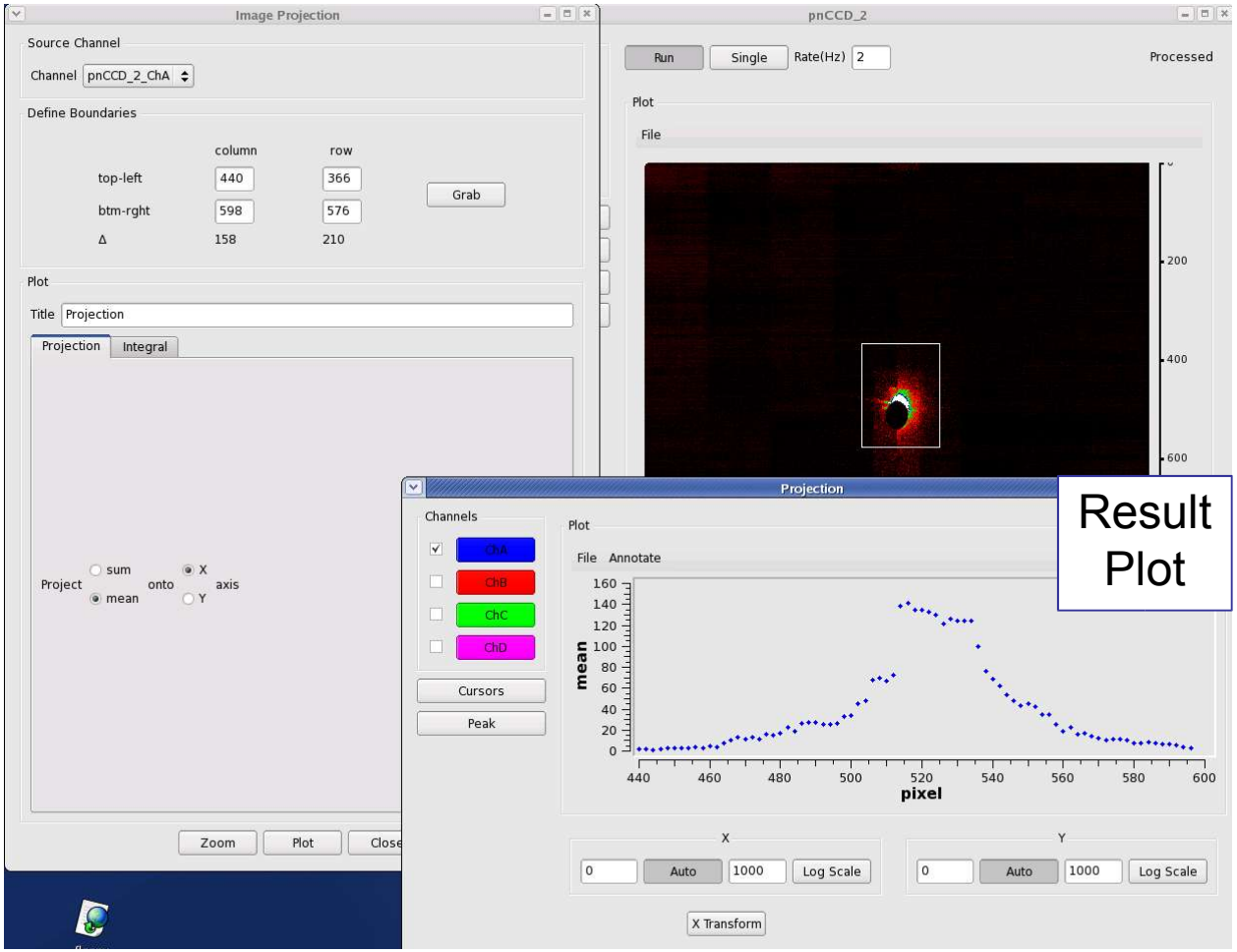
Define Region

Rectangular coordinates

Choose Projection

Direction/Axis

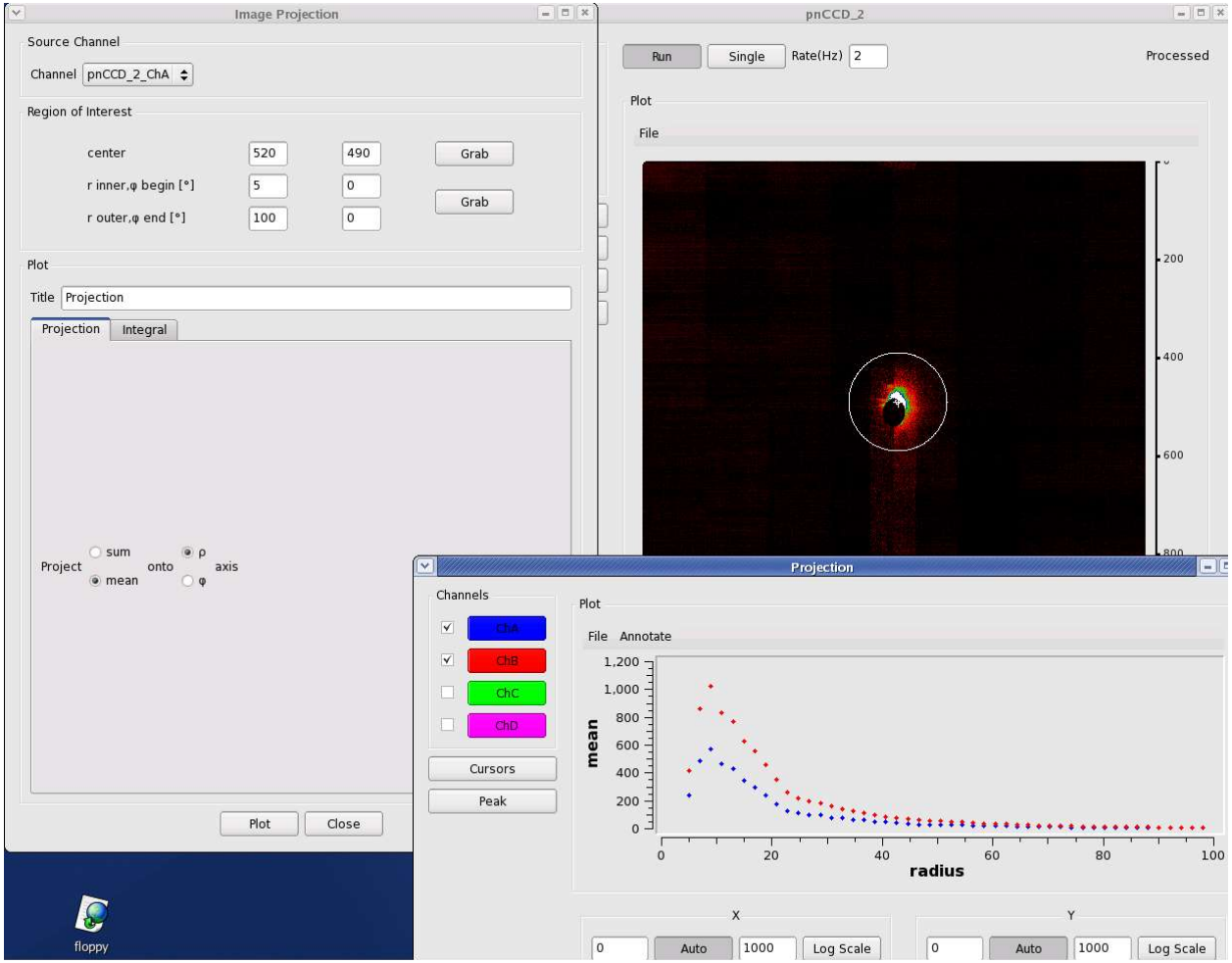
Sum or Average (normalize by area)



Operations – X/Y projection

- The resulting projections can be treated like original waveforms :
  channels {averaging, reference},
  operations {cursors, peak finder}
- Similarly for other projections
- Can also integrate over rectangular region to generate a scalar value and plot …

Operations – radial/azimuthal projection



Core Online Monitoring – User Specialization

- Simple C++ API defined for writing a plug-in module *(amiuser)*
  - *User code called for each detector event received*
  - *Defined set of plot types are available for generating displays*
  - *Some accomodation of plot arrangement on pages*
- User writes code and compiles a dynamic library
- Core monitoring application links in user library when available

Core Online Monitoring – User Specialization (2)

(https://confluence.slac.stanford.edu/display/PCDS/Adding+plots+to+the+Core+Online+Monitoring)

Copy /reg/g/pcds/package/amiuser to your area

```
cp -rf /reg/g/pcds/package/amiuser ~/.
```

Edit the ExampleAnalysis.{hh,cc} files

```
cd ~/amiuser; gedit ExampleAnalysis.cc
```

Build the libamiuser.so library

```
make
```

Copy the libamiuser.so to your experiment's home area

```
cp libamiuser.so ~xxxopr/.
```

User Shared Memory Application

- C++ API defined for receiving event data in shared memory *(pdsdata/app)*
- User writes code and builds an executable in their development environment
  - User adopts graphical display package of their choosing
  - May integrate with other analysis tools
- Application runs on monitoring nodes *{"xxx-mon"}*
- Recorded data files can also be played back through the shared memory interface
- CAMP pnCCD analysis example { XOnline, CASS }

User Disk-Based Application

- One node *("xxx-ana")* in each experiment records an extra copy of the data for prompt readback (even while writing)
  - *No risk to official data cache*
  - *File remains for brief period (1-10hrs)*
  - *First opportunity for offline-style analysis*
  - *Provides only data access when offline transfer is backlogged*
- Data format is "XTC"
- Initial analysis support provided by "myana" package
  - C++ framework (mostly simple C calls)

Getting Started with *myana*

(https://confluence.slac.stanford.edu/display/PCDS/Data+Analysis)

Nice instructions on Confluence now, but basic steps are:

```
ssh psexport                                  (login to a machine data access)
cp -r -d /reg/g/pcds/package/ana .            (copy package to your diskspace)
cd ana                                        (edit source files; e.g. myana.cc)

./comp                                        (build the executable and run)
./myana -f /reg/g/pcds/package/anatestdata/acqiris.xtc -n 100

./root                                        (view ROOT histogram output)
```

Several example uses of myana are described on Confluence.

Conclusion

The core developers try to make the system

intuitive and easy to use,

but users will benefit considerably from early preparation

of their analysis code and

monitoring plans.