CRIS daq

CRISTAL – CRIS Tuning, Acquisition and Logging

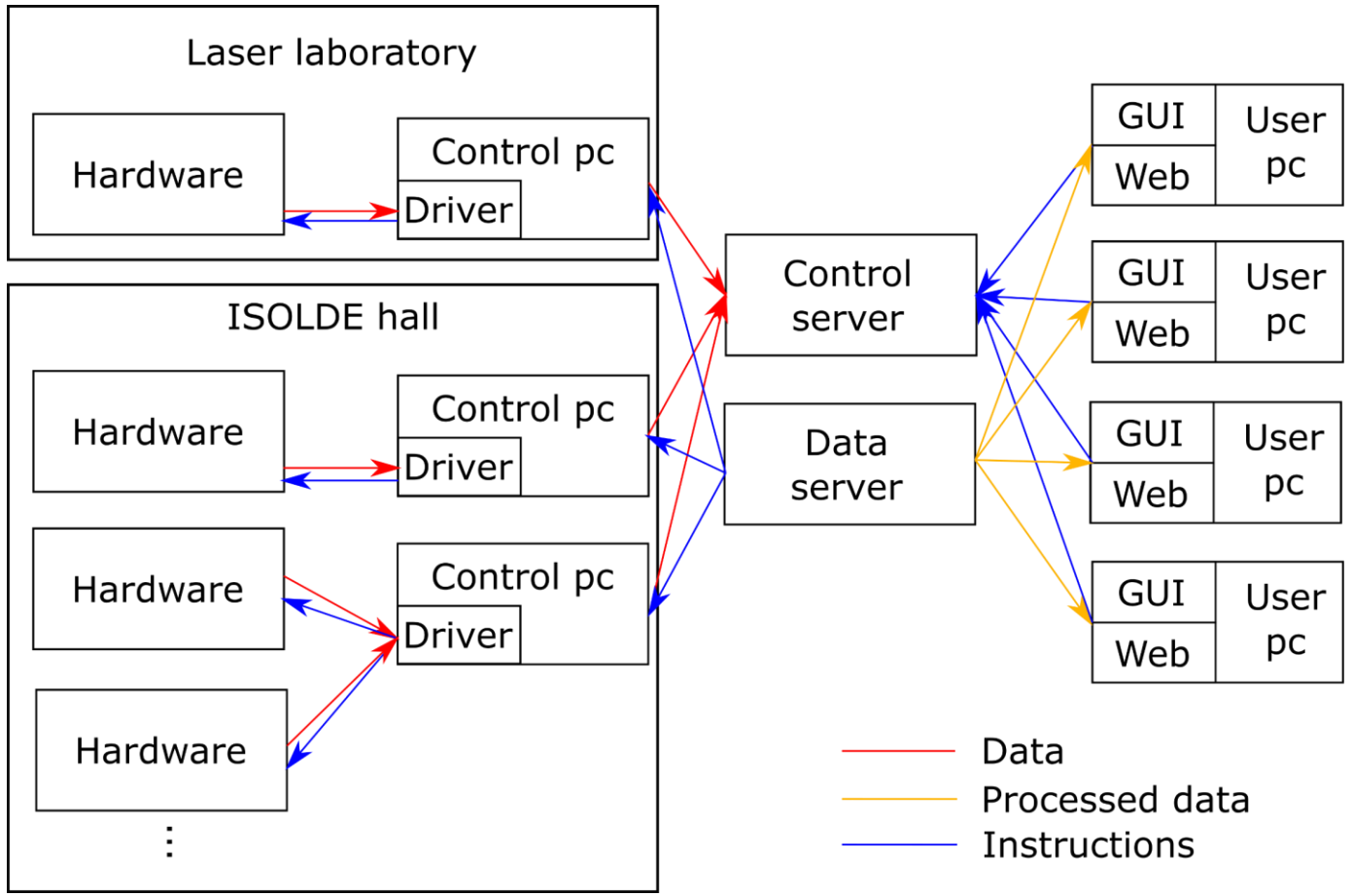
General features

CRIS collaboration signaled a need for

- a centralized control system to monitor and control hardware, to initiate measurement cycles, record data,
- That is accessible from many locations,
- Can run 24/7, 365 days a year,
- And furthermore user-friendly
- Written in an easily maintained language (Python)
  - Mostly standard library, some additional libraries for high-performance data manipulation and fast data plotting

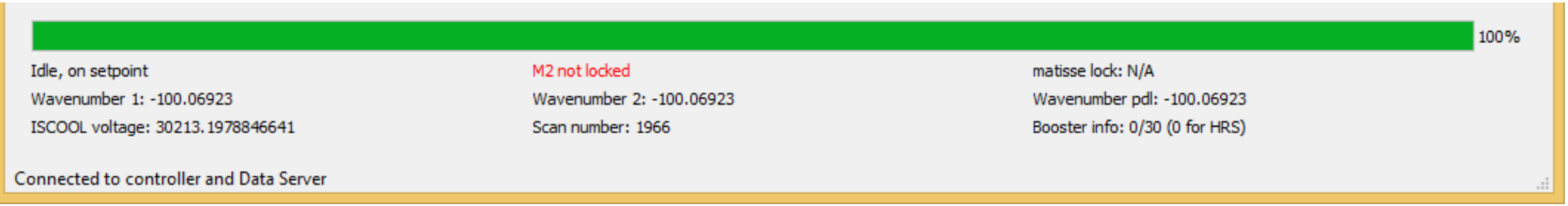
Architecture

- Centralized control and data servers
- Communications via custom TCP-IP protocol
- Every piece of hardware is polled as fast as possible, data is pushed to the servers
  - Data server timestamps all incoming data
  - Synchronization ~ms

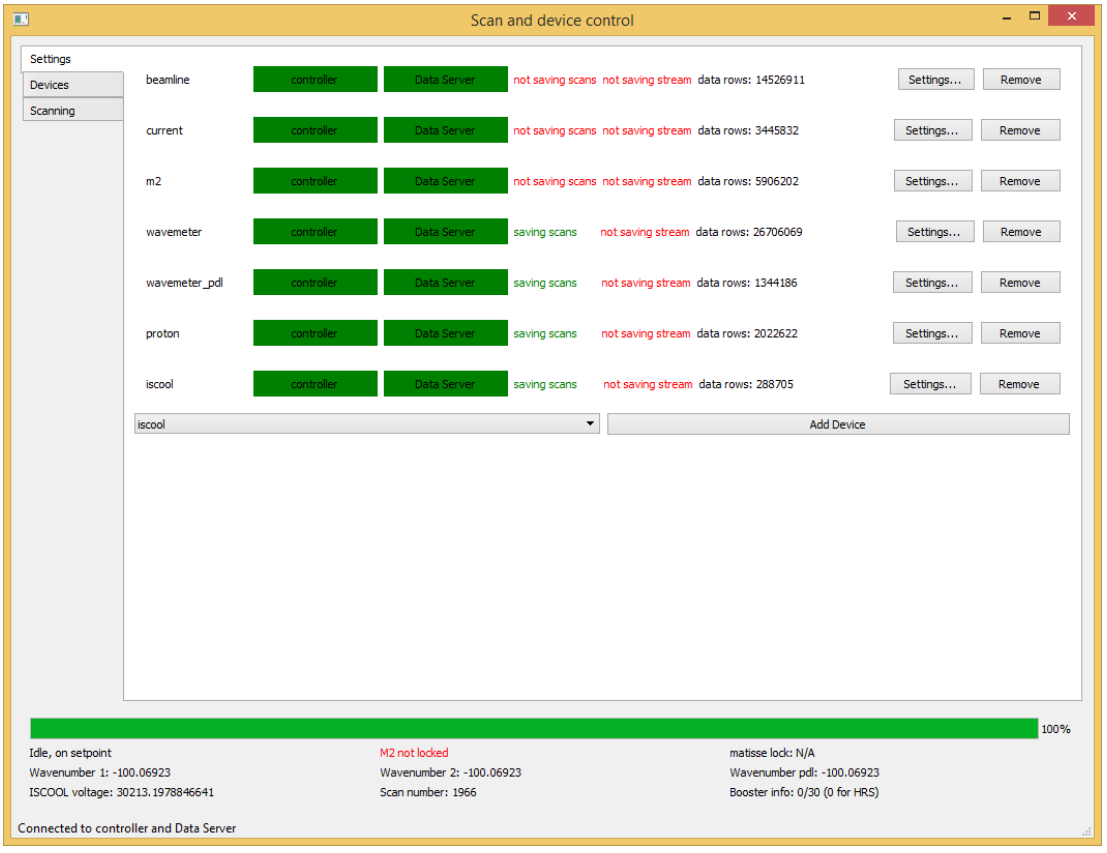


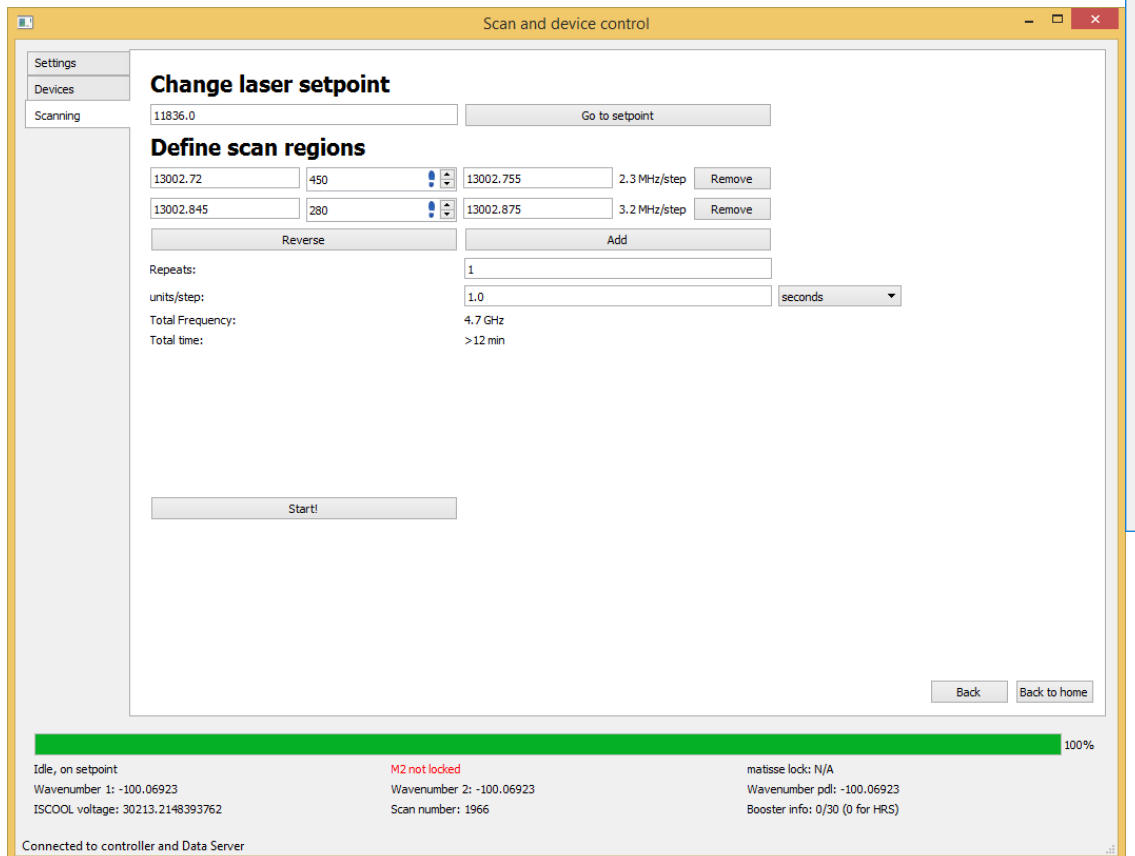
Architecture

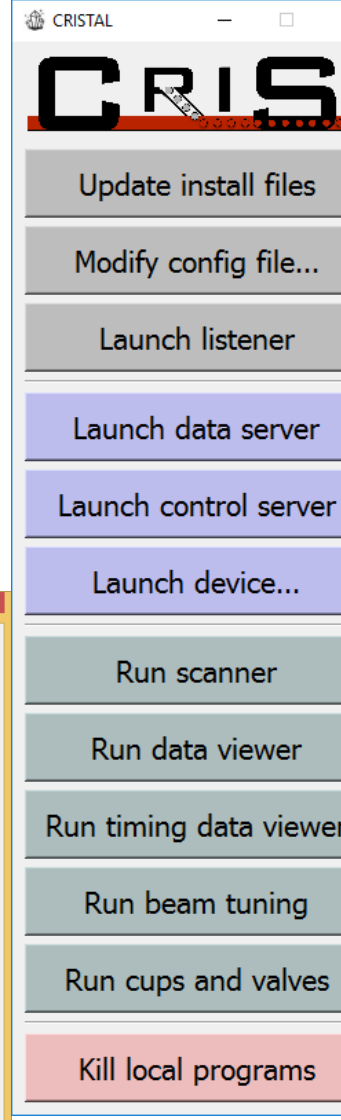
- Servers summarize information and make it available to user programs
- User programs request updates periodically
- ‘Multi-user’ – servers are multi-threaded to allow many points of interaction



- Graphical User Interface (GUI) programs – control hardware and monitor experiment from anywhere in CERN

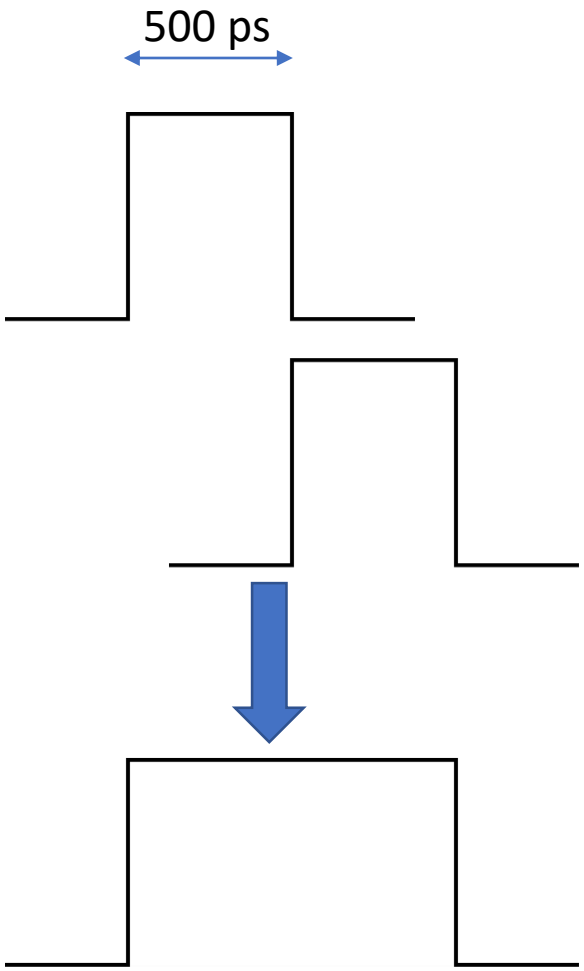




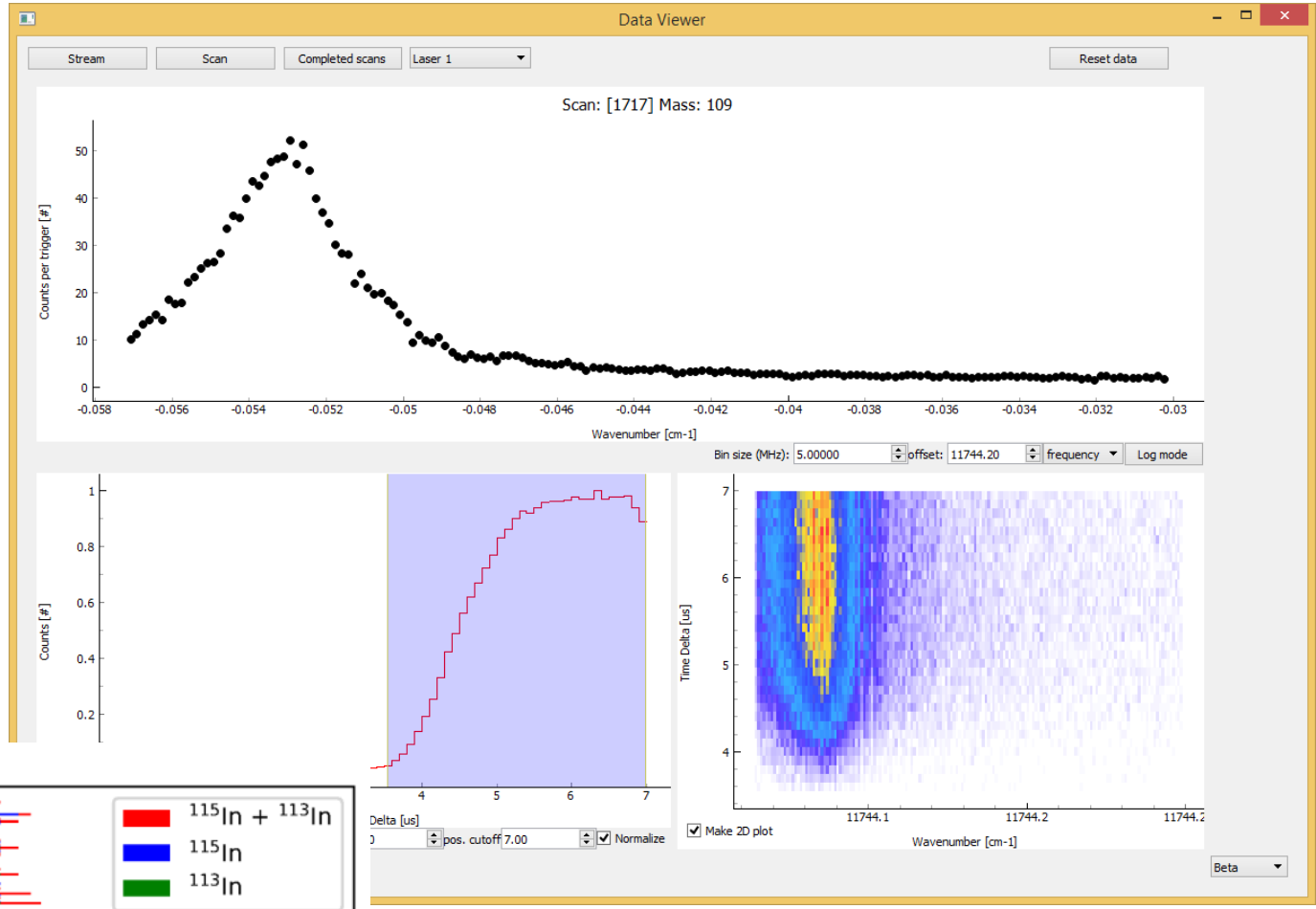


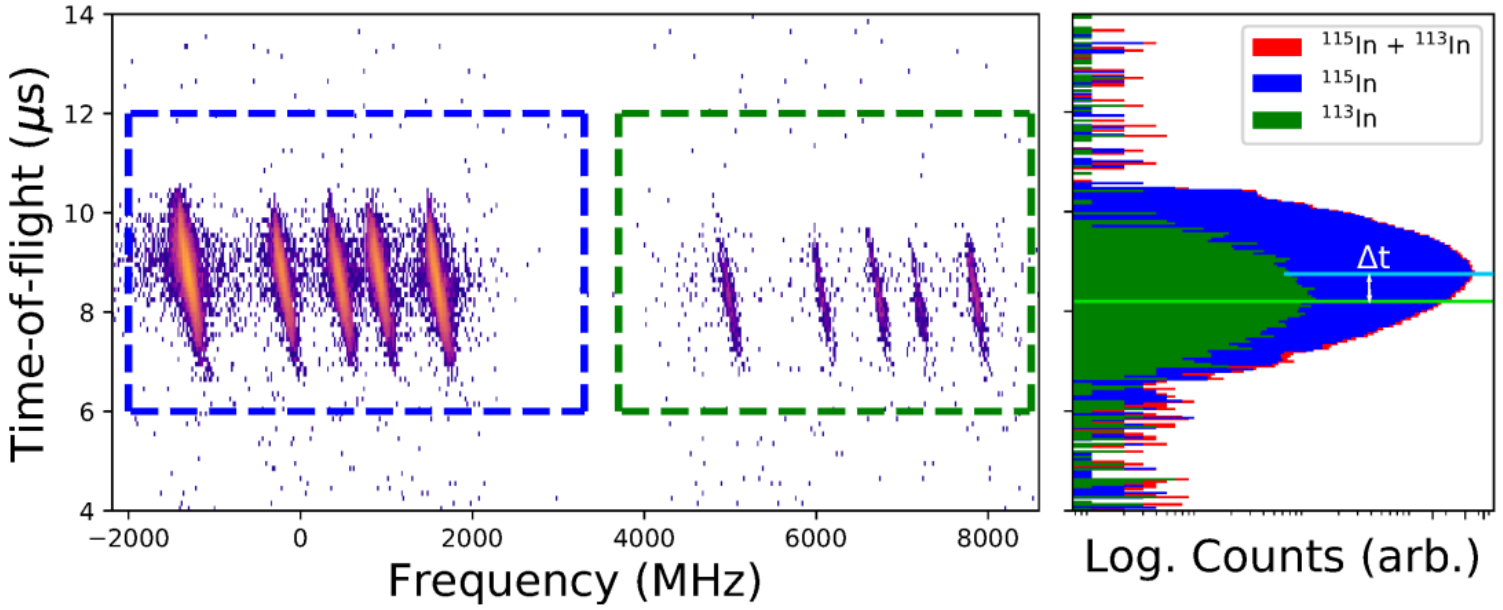
New addition in late 2017

- TDC card: timetagger 4-2G
  - 500 ps time bin
  - ‘dead time’: 1 ns double pulse resolution
  - 4 independent stop channels with software-controlled trigger levels
  - Up to 1000 events in a burst can be recorded
  - 2.5 kEUR



- Online view →


Data Viewer
Stream
Scan
Completed scans
Laser 1
Reset data
Scan: [1717] Mass: 109
Counts per trigger [#]
Wavenumber [cm-1]
Bin size (MHz): 5.00000
offset: 11744.20
frequency
Log mode
Counts [#]
Delta [us]
pos. cutoff 7.00
Normalize
Time Delta [us]
Make 2D plot
Wavenumber [cm-1]
Beta


Time-of-flight (μs)
Frequency (MHz)
115In + 113In
115In
113In
Δt
Log. Counts (arb.)

Lessons learnt

- Writing a full network protocol from scratch is very time-intensive
  - Program is easily expanded with simple features (e.g. adding a thermocouple readout, etc.)
  - But, I notice that changing the user-interface programs and working on the ‘internals’ is not something colleagues have taken up

-> I would now opt for EPICS-based networking solution for everything but high-throughput data transmission where synchronization is important

  - Experience in JYFL shows that this is a better route to take

- ‘Modular approach’ with many small programs that communicate is in my opinion the only scalable solution

- Note also: spending a bit more money on a piece of hardware with good driver support >>>>>> having a student work hard on a cheaper product