THE EUDAQ DATA ACQUISITION SYSTEM.

Introduction to Concepts and Integration

```
// A name to identify the raw data format of the events generated
// Modify this to something appropriate for your producer.
static const std::string EVENT_TYPE = "Example";

// Declare a new class that inherits from eudaq::Producer
class ExampleProducer : public eudaq::Producer {
public:

  // The constructor must call the eudaq::Producer constructor with the name
  // and the runcontrol connection string, and initialize any member variables.
  ExampleProducer(const std::string & name, const std::string & runcontrol)
    : eudaq::Producer(name, runcontrol),
      m_run(0), m_ev(0), stopping(false), done(false) {}

  // This gets called whenever the DAQ is configured
  virtual void OnConfigure(const eudaq::Configuration & config) {
    std::cout << "Configuring: " << config.Name() << std::endl;

    // Do any configuration of the hardware here
    // Configuration file values are accessible as config.Get(name, default)
    m_exampleparam = config.Get("Parameter", 0);
    std::cout << "Example Parameter = " << m_exampleparam << std::endl;
    hardware.Setup(m_exampleparam);

    // At the end, set the status that will be displayed in the Run Control.
    SetStatus(eudaq::Status::LVL_OK, "Configured (" + config.Name() + ")");
  }

  // This gets called whenever a new run is started
  // It receives the new run number as a parameter
  virtual void OnStartRun(unsigned param) {
    m_run = param;
    m_ev = 0;
    std::cout << "Start Run: " << m_run << std::endl;

    // It must send a BORE to the Data Collector
    eudaq::RawDataEvent bore(eudaq::RawDataEvent::BORE(EVENT_TYPE, m_run));
    // You can set tags on the BORE that will be saved in the data file
    // and can be used later to help decoding
    bore.SetTag("EXAMPLE", eudaq::to_string(m_exampleparam));
    // Send the event to the Data Collector
    SendEvent(bore);

    // At the end, set the status that will be displayed in the Run Control.
    SetStatus(eudaq::Status::LVL_OK, "Running");
  }
```

Simon Spannagel

EUTelescope Workshop

DESY Hamburg, March 25-27 2013

HELMHOLTZ ASSOCIATION

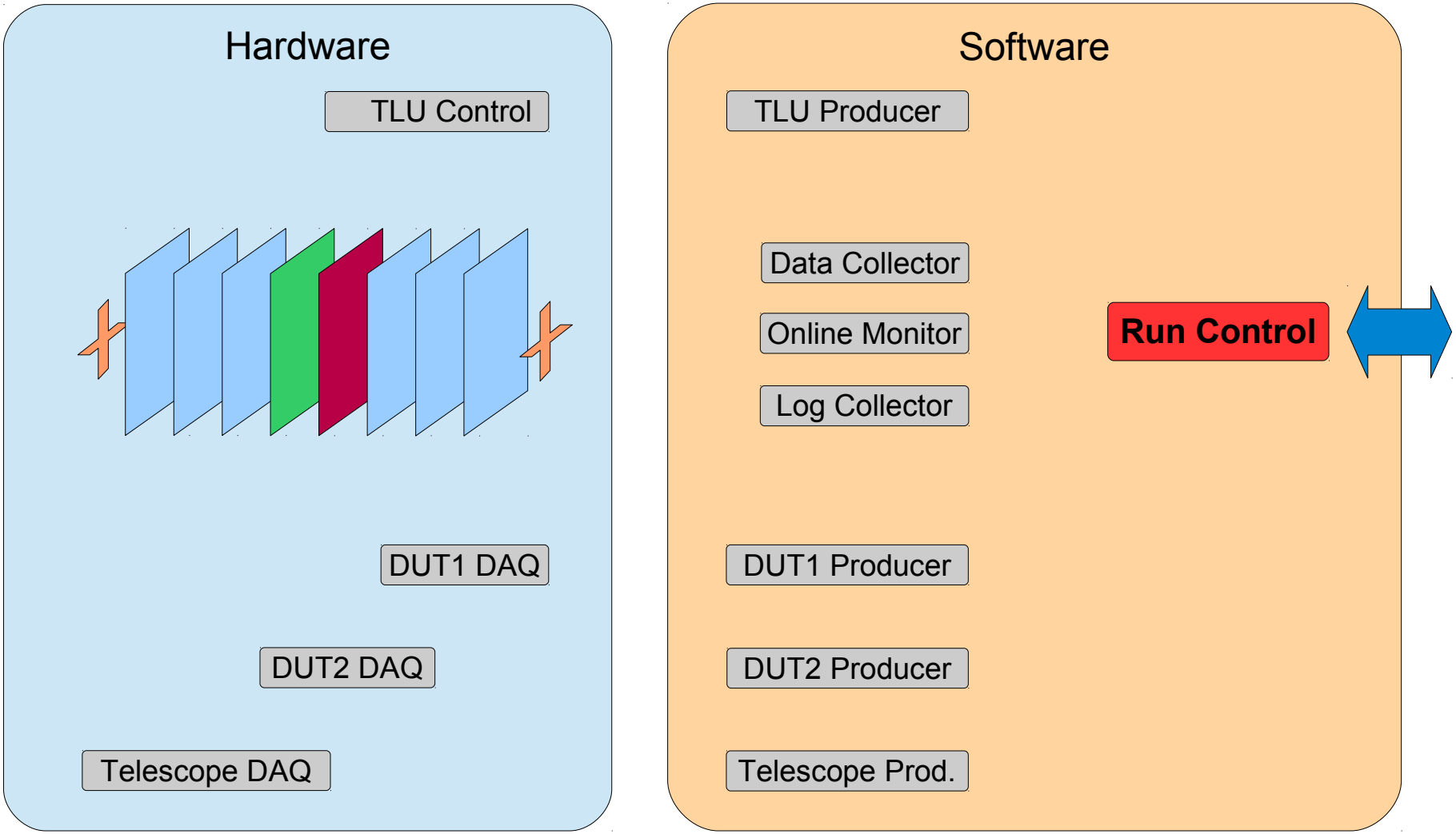
DESY

Features of the EUDAQ Data Acquisition System

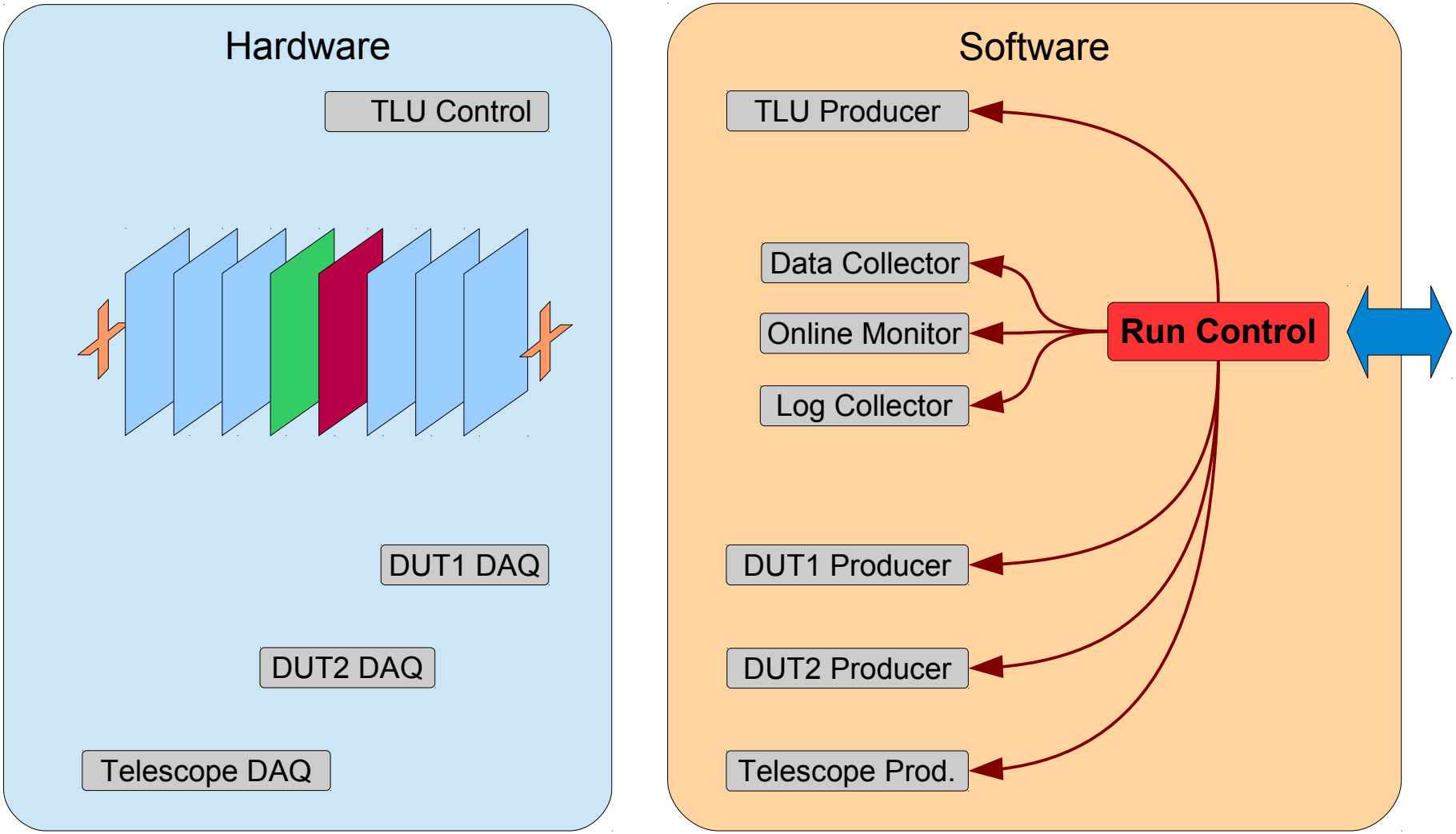
- Generic framework for data acquisition
- OS independent: Linux, Mac OSX, (Windows)
- Allows full integration of device under test (DUT) independent of its technology including pre-existing DAQ systems
- Modular and flexible design
- DAQ control via GUI, but CLI interface also available
- Online DQM using the OnlineMonitor
- All hardware communication done by “Producers” with equal rights
  - e.g. “NiProducer” for the MIMOSA26 sensors via National Instruments Crate
- Used by many groups:
  - Altro (Bonn), APIX (Atlas Pixels), Atlas (TRT), CMS Pixel (DESY), DEPFET (Bonn), FORTIS/SPIDER (Bristol), MimoRoma (INFN), MVD (DESY), PixelMan (Freiburg), SITRA (Santander), Taki (Mannheim), Timepix (Bonn)
  - and more (NA62, Alfa, Alice, etc.) ...

DESY

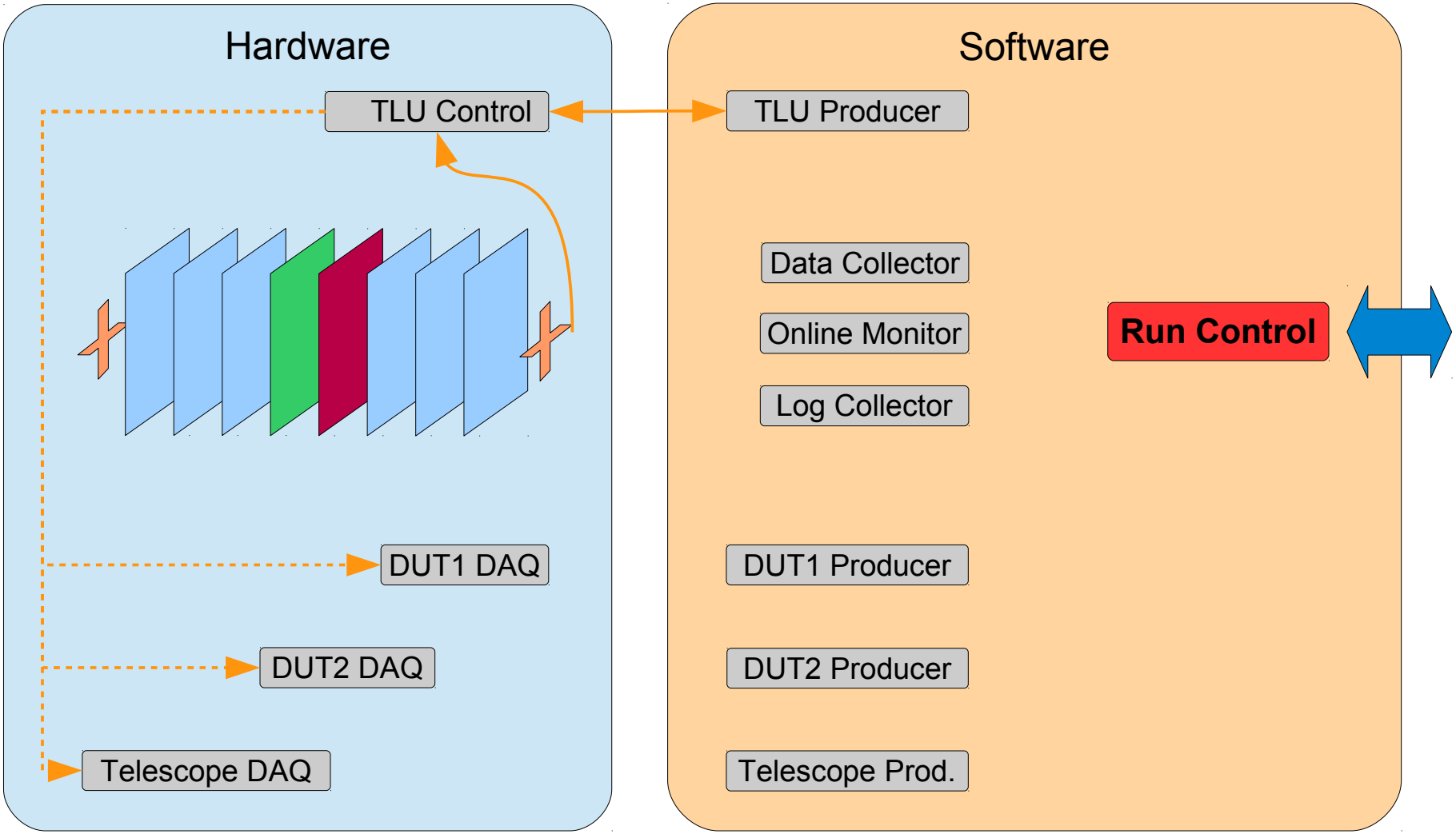
The EUDAQ System Architecture



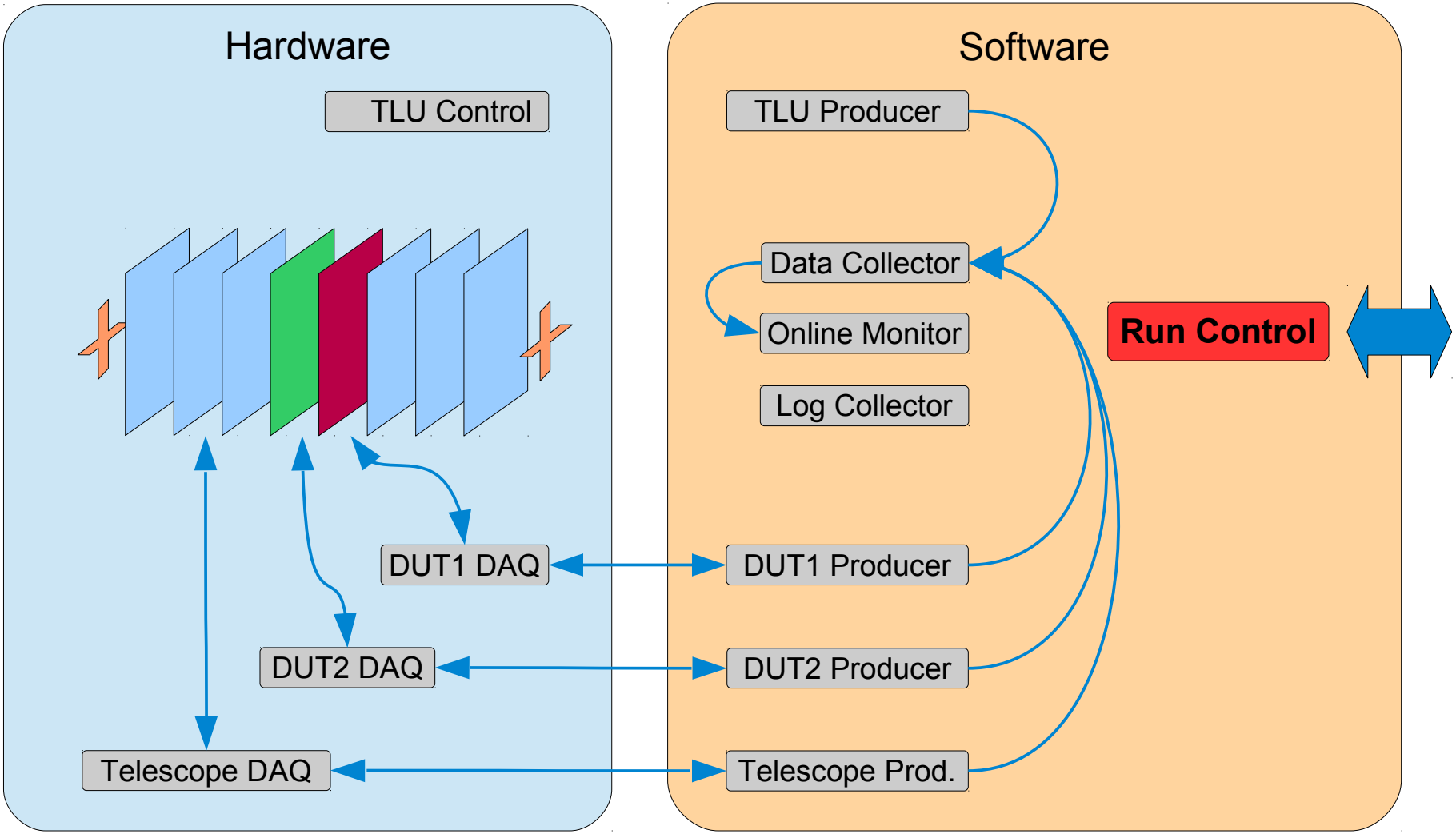
The EUDAQ System Architecture



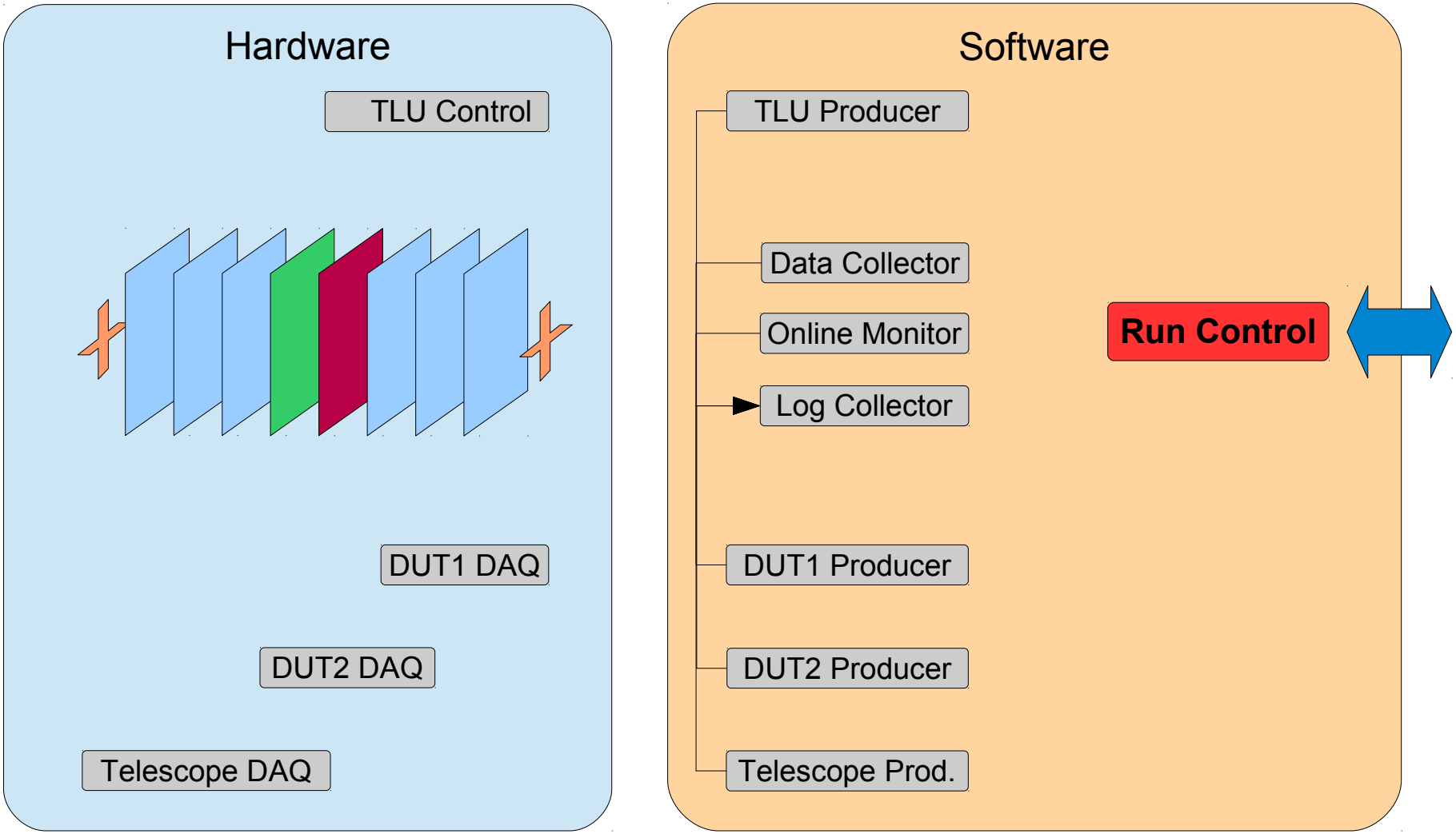
The EUDAQ System Architecture



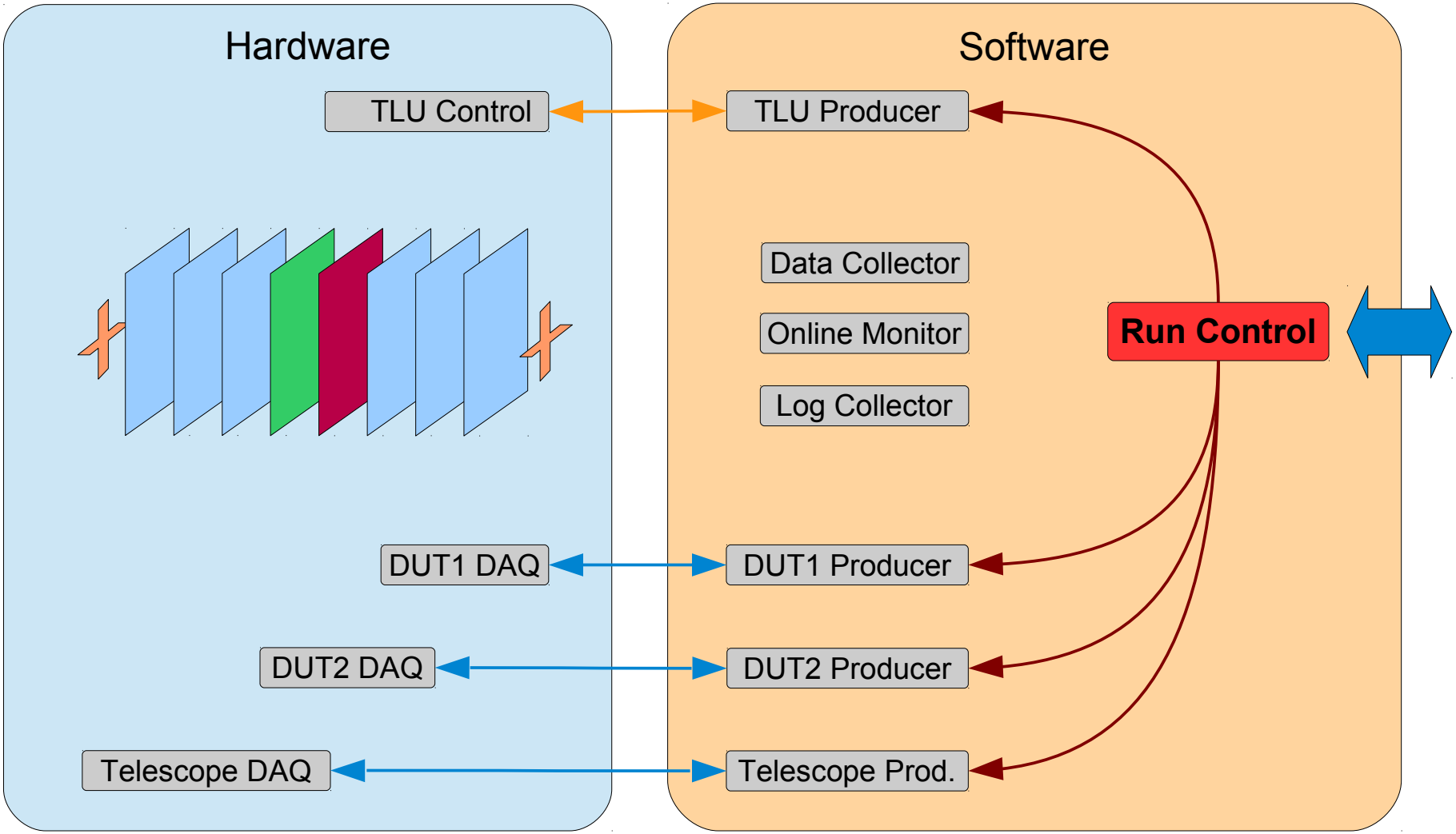
The EUDAQ System Architecture



The EUDAQ System Architecture



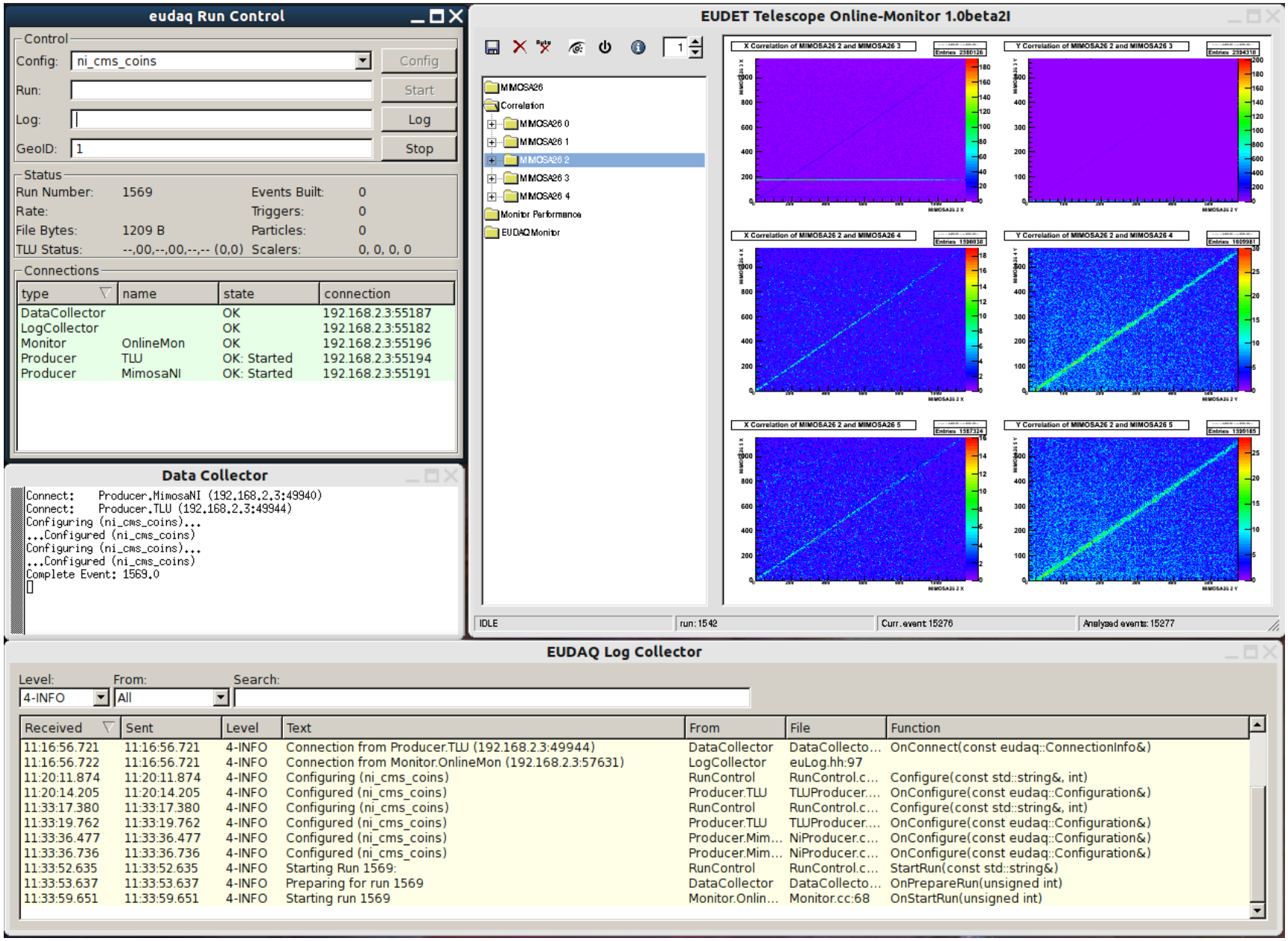
The EUDAQ System Architecture



Running EUDAQ

- All components communicate via TCP stack
  - can be run on different networked machines
- User interaction via Run Control
- Processes:
  - Run Control
  - Log Collector
  - Data Collector
  - Producers for hardware communication (e.g. TLU, Telescope, DUT)
  - [Online Monitor]
- Start scripts are provided which fire up all necessary processes

Running EUDAQ



EUDAQ Data Formats

- Basic EUDAQ event data format: RawDataEvent
  - Generic container for unaltered, encapsulated detector response
  - Data input: raw block of memory or vector
  - Storage of additional information possible (custom tags, trigger numbers...)
  - Correct data decoder is chosen by unique identifier string for each producer

- Alternative: StandardEvent
  - Decoded detector data
  - Consists of StandardPlanes with set pixel dimensions of the respective detector
  - Can be read by e.g. the Online Monitor for direct processing

- Final analysis in EUTelescope: LCIO (Linear Collider I/O)
  - Usually done within EUTelescope

Integrating a DUT into EUDAQ

- DUT Producer
  - Talk to the DAQ hardware, receive events from there
  - Receive commands from Run Control:

    OnConfigure, OnStartRun, OnStopRun, Terminate
  - Send data to the Data Collector (either RawDataEvent or StandardEvent)
  - Configure itself with parameters received before data taking
  - Send log messages to the Log Collector
- [DUT DataConverterPlugin]
  - Convert the specific native detector data into StandardEvents
  - Needed e.g. for online monitoring of the DUT
  - Can be used to convert into LCIO for the final analysis using EUTelescope
- Example code showing the usage of the base classes is provided.

Summary

- EUDAQ
  - offers a modular and flexible framework for data acquisition
  - is well documented (see below)
  - is used and supported within AIDA
- Both simple and full integration of a DUT and its DAQ possible (only producer or additional DataConverterPlugin)
- Also usable for detector DAQ w/o AIDA telescopes
- Future plans: add slow control elements for beam monitoring
  - Enables e.g. the storage of beam energy settings into the respective data files

- SVN repository: http://eudaq.hepforge.org/svn/trunk
- Extensive documentation (~60p) available in SVN, build PDF using
  - cd doc/ && make manual