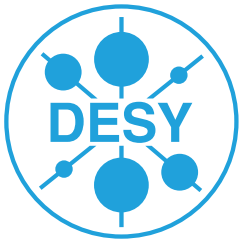


EUDAQ 2.0 as DAQ Software for the AIDA Telescope

Ulf Behrens, Alan Campbell, Francesco Crescioli, David Cussans, Moritz Kiehn,
•Hanno Perrey, Richard Peschke, Igor Rubinskiy, Simon Spannagel

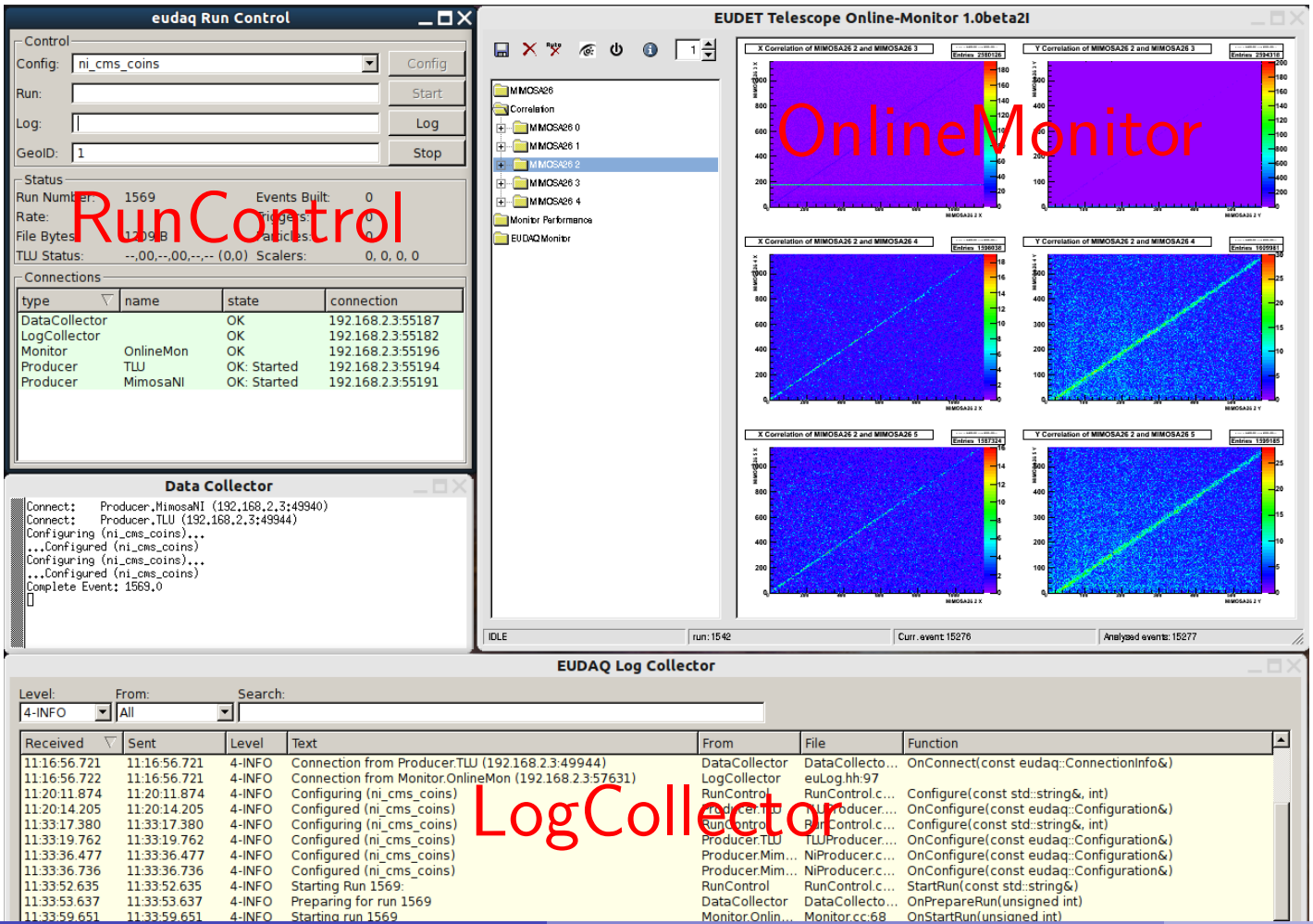
1. Short Introduction to EUDAQ
2. EUDAQ 2.0 (aka AIDAQ) for the AIDA Beam Telescope
3. Status of Development

Reminder: EUDAQ Philosophy and Features

- Generic framework for data acquisition
- OS independent: Linux, Mac OSX, Windows
- Modular and flexible design
- Provides central DAQ control, data handling and storage, log collection, online monitoring
- Components communicate via TCP/IP and can run on different networked machines
- Focused on easy and flexible integration of the device under test including pre-existing DAQs
- All hardware communication done by "Producers" with equal rights

EUDAQ: Components and Interfaces

- Central components run as separate processes: RunControl, DataCollector, LogCollector, OnlineMonitor
- Different interfaces available: Qt GUI, console, Python



From EUDET to AIDA Beam Telescope

EUDET style: "Event-based"

- One trigger per slowest DAQ system in the telescope
- No triggers from TLU while at least one system is *BUSY*
- Low rates/data-taking efficiency for fast (LHC-type) DAQ systems
- Data is written in single stream and stored centrally

AIDA style: "Particle-based"

- TLU issues trigger for every particle (i.e. scintillator signal)
- Triggers stop only on *VETO* from any DAQ
- High-rates/more efficient data-taking possible
- Telescope DAQ needs to cope with high data rates (e.g. continuous readout of Mimosa28 quad-planes)

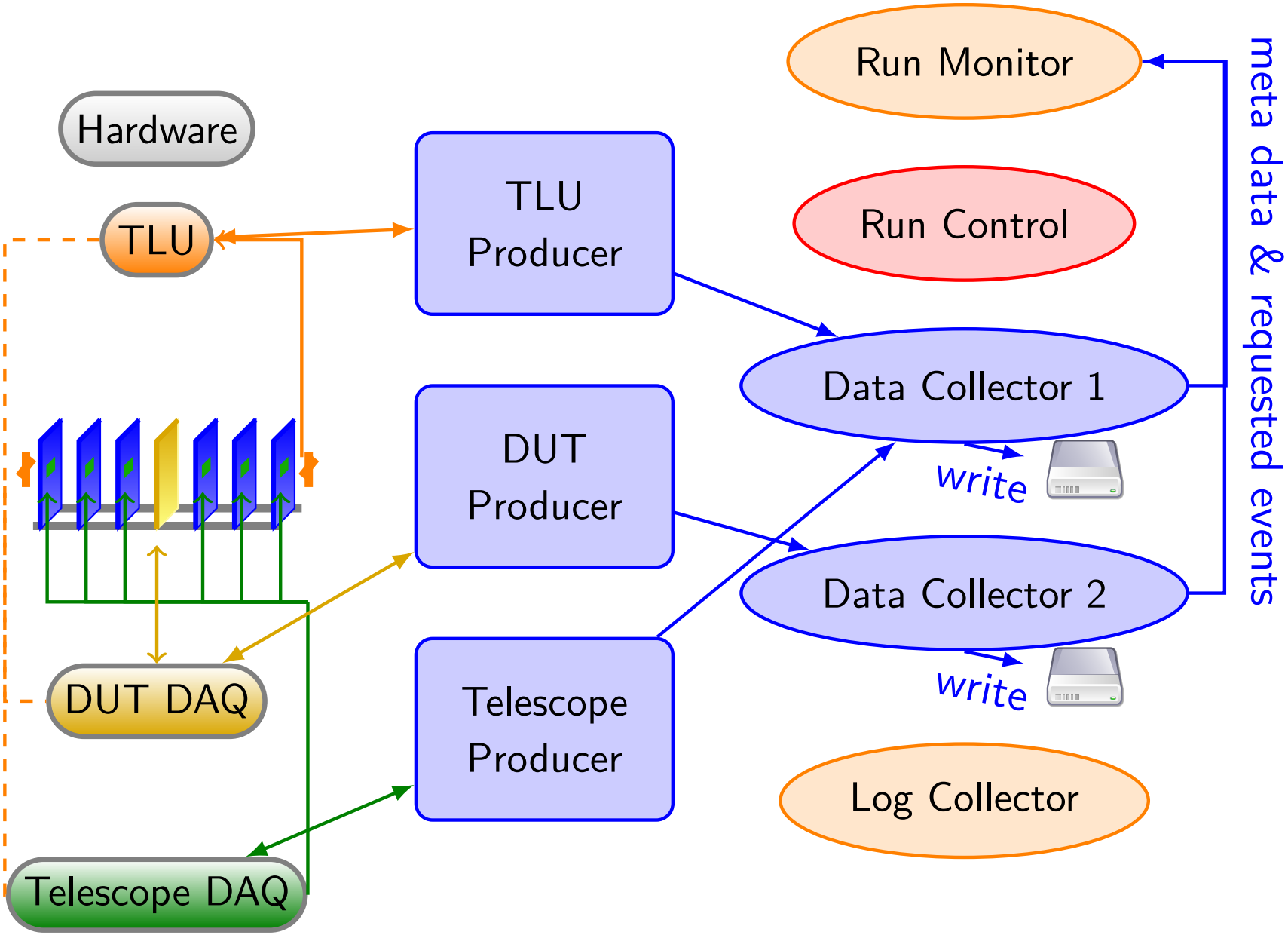
⇨ Data in asynchronous streams from various DAQs

Implications on the feature set of EUDAQ 2.0

- Accepts data packets from DAQs covering:
  - single trigger ("classic mode")
  - list of (timestamped) triggers
  - time range (shutter/data driven devices)
  - both time range and trigger list
- Reduce disk IO/network bottlenecks: DAQs (can) store data locally by assigning separate (locally-running) DataCollectors to each Producer
- Online verification in this scheme:
  - separate on-the-fly event building (full or partial for specific triggers)
  - cross-check timestamps e.g. against TLU information
  - merged data can be used for online monitoring/immediate offline analysis
- Will be backward-compatible to old integration efforts (no more than a recompilation should be needed)

⇨ Easier integration with a wider range of DAQ concepts

Schematic Layout of EUDAQ 2.0 (preliminary)



Current status of EUDAQ development

- A lot of maintenance has been done on EUDAQ (and is ongoing):
  - changed to CMake for easier cross-platform configuration and builds
  - now runs on all major platforms (Linux, OS X, Windows)
  - moved to git/GitHub for easier collaboration and code review
  - various bug fixes
  - ...

→ just released version 1.2.0!

- Now provide a scriptable Python interface e.g. to RunControl and a generic Producer implementation (see backup)
- RunControl can now assign a different DataCollector to each Producer and the resulting files can be merged offline
- New event format specified and implementation started
- Best strategy for online (partial) event building and data validity checking currently being investigated

Our aim: testing an early version of EUDAQ2.0 (with new data format and at least minimal online validity checks) in the July testbeam

Summary and Outlook

- EUDAQ is a very flexible DAQ framework and has been used successfully for a long time with EUDET-family of telescopes
- Support one-trigger-per-particle operation of AIDA beam telescope requires extension of EUDAQ and removal of IO/network bottlenecks
- Key changes: timestamped data packages, multiple data streams
- Makes integration with EUDAQ 2.0 easier and more flexible than before
- Development on EUDAQ is ongoing, contributions are welcome!

Download the newest version and follow the development at `http://eudaq.github.io`

Overview Backup Slides

Integrating a Device into Telescope/EUDAQ 2.0

TLU

- Receive clock, trigger and/or shutter signals
- Timestamp events (trigger, shutter open/close, ...)

Producer to interact with RunControl and Hardware

- Implemented in C++, but Python interface exists (see backup)
- Receives commands from RunControl (Configure, Run Start/Stop)
- Talks to the DAQ hardware, receives data from there
- Sends its data to a DataCollector (optional)
- Gain flexibility with new event format and multiple data streams
  $\Rightarrow$ wider range of devices suitable for (full) integration

Optional: Data Converter Plugin

- Converters native device format into defined structure
- Used for conversion into LCIO and for online monitoring

The Anatomy of a *Producer*

- A Producer needs to implement command receiving methods of the `Producer` base class:
  - `OnConfigure`, `OnStartRun`, `OnStopRun`, `Terminate`
- It configures the hardware according to the config received from RunControl
- It (optionally) sends its data to the data collector:
  either in raw format or converted to a custom `StandardEvent` class
- It (optionally) logs status/error messages

Example code is provided!
→ see also shortened examples on the following slides

Example C++ (Pseudo-) Code for a EUDAQ Producer

```
class ExampleProducer : public eudaq::Producer {
  public:
    ExampleProducer(){}
    virtual void OnConfigure(const eudaq::Configuration & config) {
      // .... configure your hardware
    }
    virtual void OnStartRun(unsigned param) {
      // .... prepare for and start run
    }
    virtual void OnStopRun() {
      // .... stop your DAQ
    }
    void ReadoutLoop() {
      while (true) {
        // while running:
        // ..... get raw data, put it into RawDataEvent and send
      }
    }
};
int main(int /*argc*/, const char ** argv) {
    ExampleProducer producer();  // Create a producer
    producer.ReadoutLoop();      // And set it running...
    return 0;
}
```

Example Python Code for a EUDAQ Producer

```
#!/usr/bin/env python2
execfile('PyEUDAQWrapper.py') # load the ctypes wrapper
from time import sleep
import numpy # for data handling
# create PyProducer instance
pp = PyProducer("testproducer","tcp://localhost:44000")

# wait for CONFIGURE cmd from RunControl
while not pp.Configuring:
    sleep(1)
# .... do your config stuff here ...
pp.Configuring = True
# check for RUNSTART cmd from RunControl
while not pp.StartingRun:
    sleep(1)
# ... prepare your system for the immanent run start
pp.StartingRun = True
# starting to run main DAQ loop
while not pp.Error and not pp.StoppingRun and not pp.Terminating:
    # prepare an numpy array for raw data storage
    data = numpy.ndarray(shape=[1,3], dtype=numpy.uint64)
    # .... get your data from your hardware ...
    pp.SendEvent(data)          # send event off
```

New Python Interface to EUDAQ

- uses CTypes to provide a Python wrapper around pure C function calls
- C++ classes have to expose such C functions to their methods (usually straightforward and already done as proof-of-concept for RunControl, DataCollector, and a generic Producer)
- For example Python-side code, see earlier slides
- Will become part of main EUDAQ repository soon, see pull request 43 on GitHub (`https://github.com/eudaq/eudaq/pull/43`)