Data Acquisition Backbone Core Framework - Interfacing readout hardware

Jörn Adamczewski-Musch, Hans G.Essel, Sergey Linev
GSI, Experiment Electronics: Data Processing group

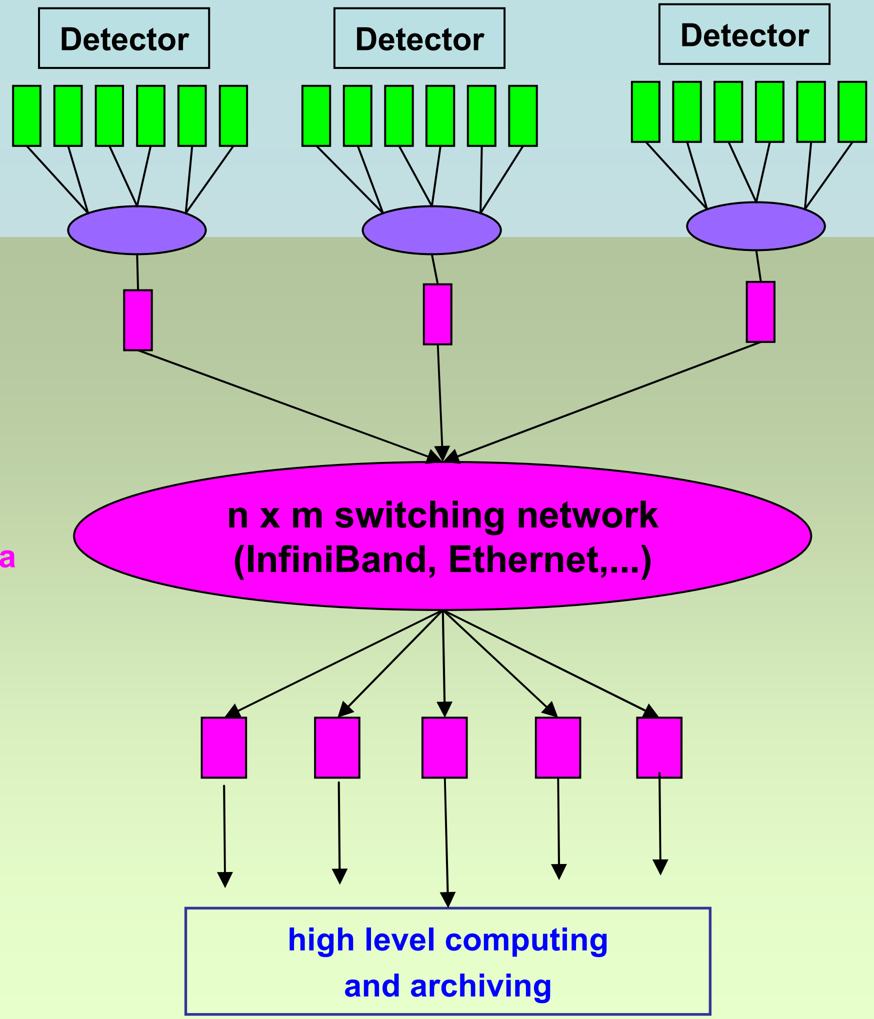
Outline

- DABC Framework overview
- Device and Transport interface
- Usage and Configuration
- Implementation Example 1:

  PEXOR (PCIe optical receiver)
- Implementation Example 2:

  CBM „Active Buffer Board“ (PCIe)
- Summary and Outlook

Use case: DAQ with fast switching network



FEE
time stamped
or triggered data

data collectors

n subdetector collector nodes

BNet
sort time stamped data
dispatch data

DABC

m event processing nodes

FLES / SLES
First / Second level
event selection

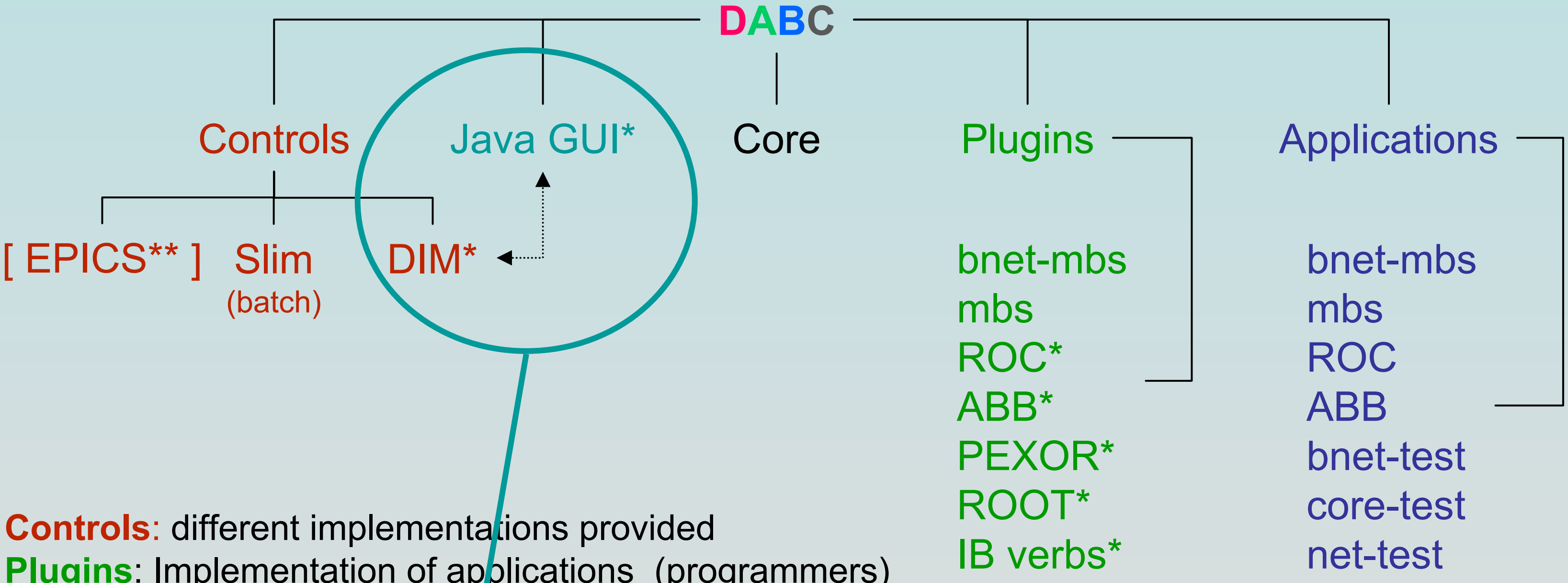
Linux PC cluster

DABC Framework

- „General purpose“ DAQ software framework
- Goal: collect and process data over fast networks
  triggered or time-stamped front-ends
- Environment: PC with Linux
- Plain C++ based core
- (user) plug ins for
  - data formats and processing
  - data input and output
  - control system (DIM, Java GUI)
- Supports established GSI production DAQ system MBS
  - data links to MBS readout nodes (Lynx OS)
  - file I/O with MBS *.lmd formats
  - can emulate MBS data servers -> online analysis

DABC release v1.0.x

Download via http://dabc.gsi.de



DABC

Controls

- [EPICS**]
- Slim (batch)
- DIM*

Java GUI*

Core

Plugins

- bnet-mbs
- mbs
- ROC*
- ABB*
- PEXOR*
- ROOT*
- IB verbs*

Applications

- bnet-mbs
- mbs
- ROC
- ABB
- bnet-test
- core-test
- net-test

Controls: different implementations provided
Plugins: Implementation of applications (programmers)
Applications: Mainly setup or testing programs (users)

* external packages needed
** under construction

ROC: ReadOutController board (UDP)
ABB: ActiveBufferBoard (PCIe)
PEXOR: PCI Express Optical Receiver
IB: InfiniBand
mbs: MultiBranchSystem (GSI DAQ)

See Poster by H.G.Essel:
A DIM Based Communication Protocol to Build Generic Control Clients

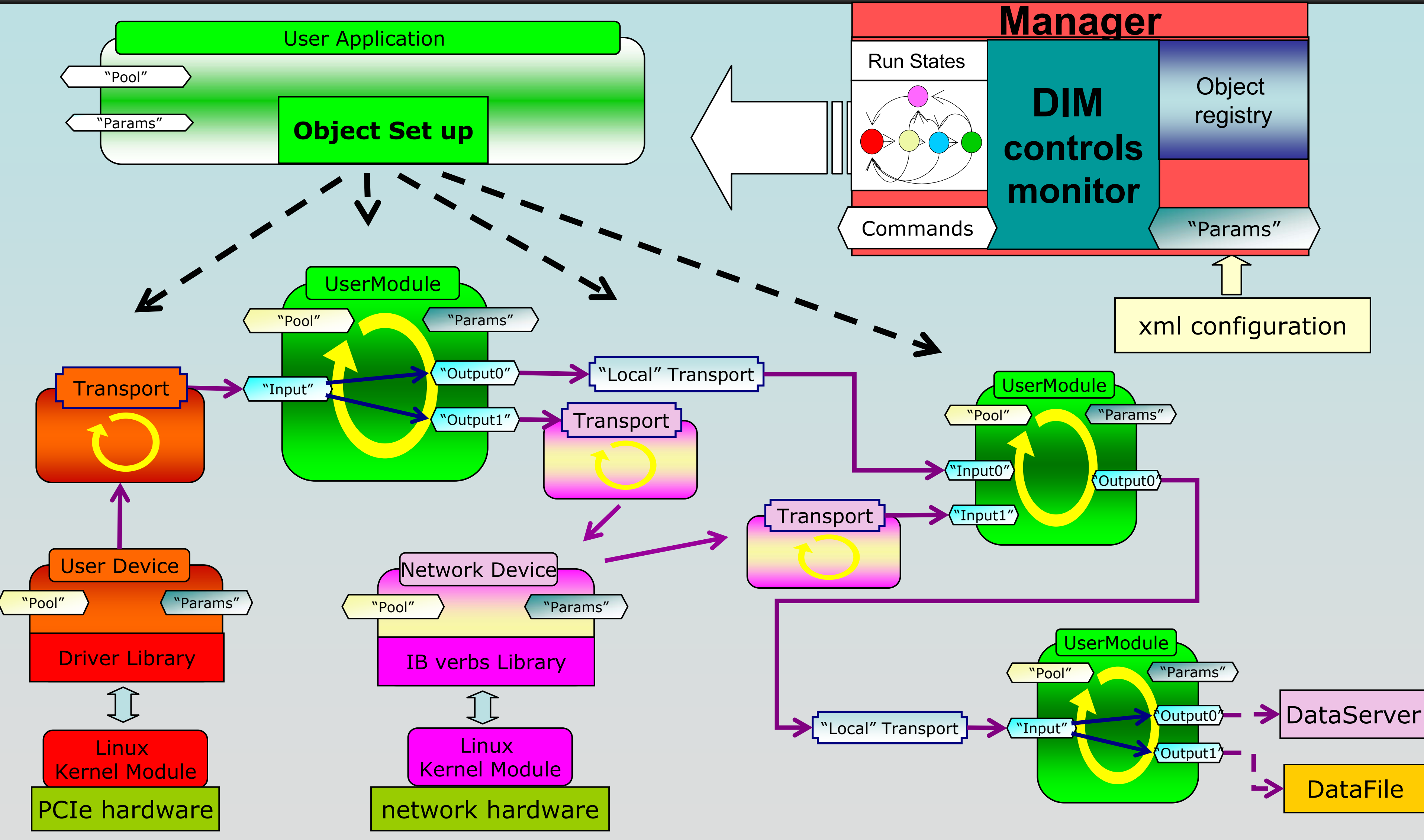
DABC Software Features 1

- Runtime environment:
  - Worker objects with (optional) shared threads
    => avoid wait times in mutex, condition, or context switches
  - Command objects executed within Worker context
  - Commands and event signals dispatched via queues
- Memory pools and Buffer management
- Data processing code in Module objects
- I/O connections in Device objects
- Dataflow via Transport connections
  between Ports at Modules and Devices

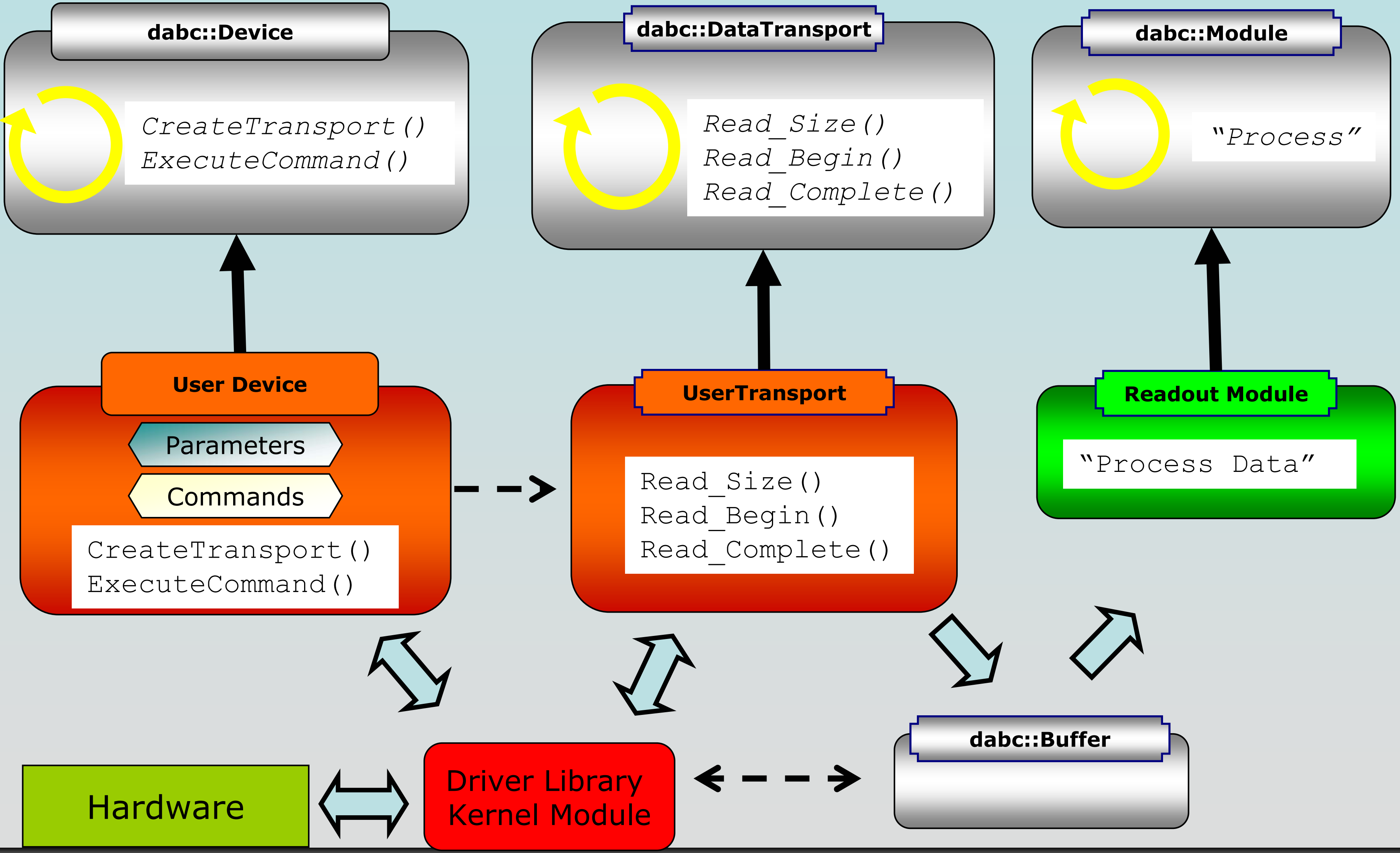
DABC Software Features 2

- DAQ node is set up by Application singleton
  - Implements initialization methods for Modules, Devices, connections, memory pool
  - Defines Parameter objects, values assignable from XML configuration file
  - Re-Implements control state transitions (optional)
- Manager singleton:
  - Object management
  - Defines run control state machine
  - Implements control system (simple, DIM,...)
  - Dispatches Commands to Processor instances

Software objects in one DABC node



DABC interface for new user hardware



dabc::Device class interface

- **Set up by Parameter objects**
- **Factory method *CreateTransport()***
  - **Defines corresponding Transport implementation**
  - **used by framework to connect any Module to this Device**
- **Method *ExecuteCommand()***
  - **can implement callbacks for user Commands (Device control)**
  - **runs in dedicated Device thread**
- **Method *DoDeviceCleanup()***
  invoked by framework at cleanup time

dabc::DataTransport class interface

- Method *unsigned int Read_Size()*
  - Invoked before each buffer transfer
  - Returns size of data to be filled
- Optional: Method *Read_Start(dabc::Buffer*)*
  - Invoked at begin of each buffer transfer
  - Passes target buffer of requested size
  - May initiate transfer to buffer
  - Must not wait for transfer completion

Asynchronous mode
(device DMA,
DABC double buffers)

- Method *Read_Complete(dabc::Buffer*)*
  - Invoked at the end of each buffer transfer
  - Passes target buffer of requested size
  - Fills target buffer, or waits until filling is complete

DABC Node configuration

DABC Parameter objects:

- Registered in Application or Modules
- Values read from XML config
- Used at (Device) initialization
- Can be monitored by control system
- XML syntax with name wildcards (*)
  => simplifies set up of many nodes

```
<?xml version="1.0"?>
<dabc version="1">
 <Context name="Pexor-Readout">
  <Run>
            <lib value="libDabcMbs.so"/>
            <lib value="x86_64/lib/libDabcPexor.so"/>
            <lib value="libpexor.so"/>
            <logfile value="ReadoutPexor.log"/>
 </Run>
<Application class="pexorplugin::ReadoutApplication">
            <PexorID value="0"/>
            <PexorNumSlaves_0 value="0"/>
            <PexorNumSlaves_1 value="2"/>
            <PexorNumSlaves_2 value="0"/>
            <PexorNumSlaves_3 value="0"/>
            <PexorDMALen value="65536"/>
            <PexorDMABuffers value="30"/>
            <ExploderSubmemSize value="2048"/>
            <PexorFormatMbs value="true"/>
            <PexorOutFile value=""/>
            <MbsServerKind value="Stream"/>
            <MbsFileSizeLimit value="110"/>
            <BufferSize value="65536"/>
            <NumBuffers value="100"/>
            <PexorModuleName value="PexorReadout"/>
            <PexorModuleThread value="ReadoutThread"/>
            <PexorDeviceName value="PEXOR2"/>
            <PexorDeviceThread value="DeviceThread" />
</Application>
<Module name="PexorReadout">
             <Ratemeter name="*" debug="true" interval="3" width="5" prec="2"/>
</Module> </Context>
</dabc>
```

Runtime Libraries,
Logfile

Hardware set up

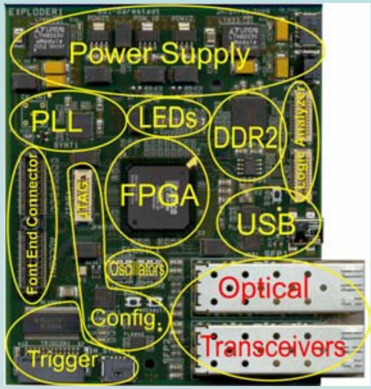
Application
Parameters

Module
Parameters

Example 1: PEXOR board and FEBs

Developed by GSI EE: J.Hoffmann, N.Kurz, S.Minami, W.Ott, S.Voltz

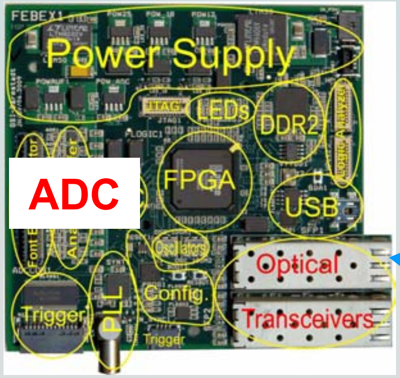
Front End Board



EXPLODER

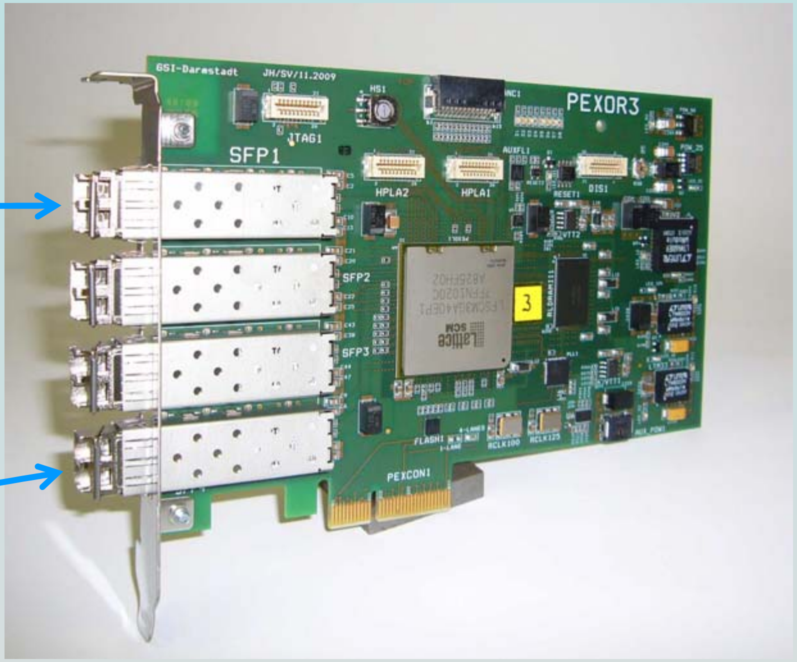
chain

Detector electronics



FEBEX

chain



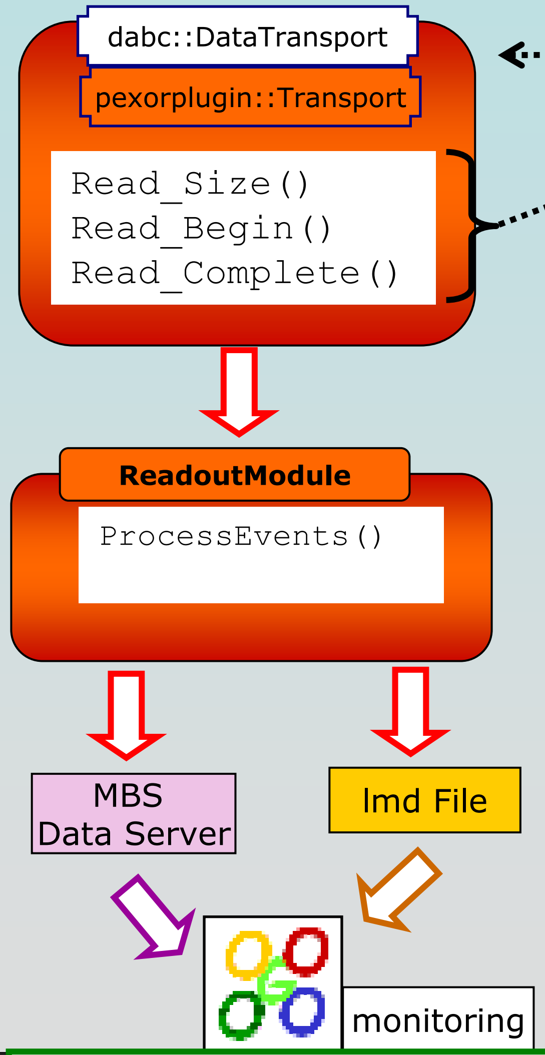
***PEXOR**: **P**ci-**EX**press **O**ptical **R**eceiver*

- Lattice FPGA
- 4 lane PCIe
- 4 high speed optical connectors (SFP)

See Poster by S.Minami:
Design and Implementation of a Data Transfer Protocol via Optical Fibre

PEXOR board: Driver and DABC plug-in

DABC Plug ins



dabc::DataTransport

pexorplugin::Transport

```
Read_Size()
Read_Begin()
Read_Complete()
```

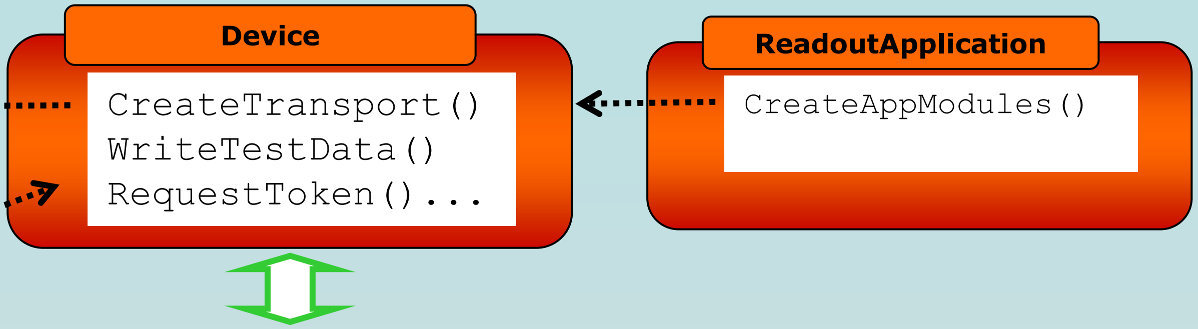
ReadoutModule

```
ProcessEvents()
```

MBS Data Server

lmd File

monitoring

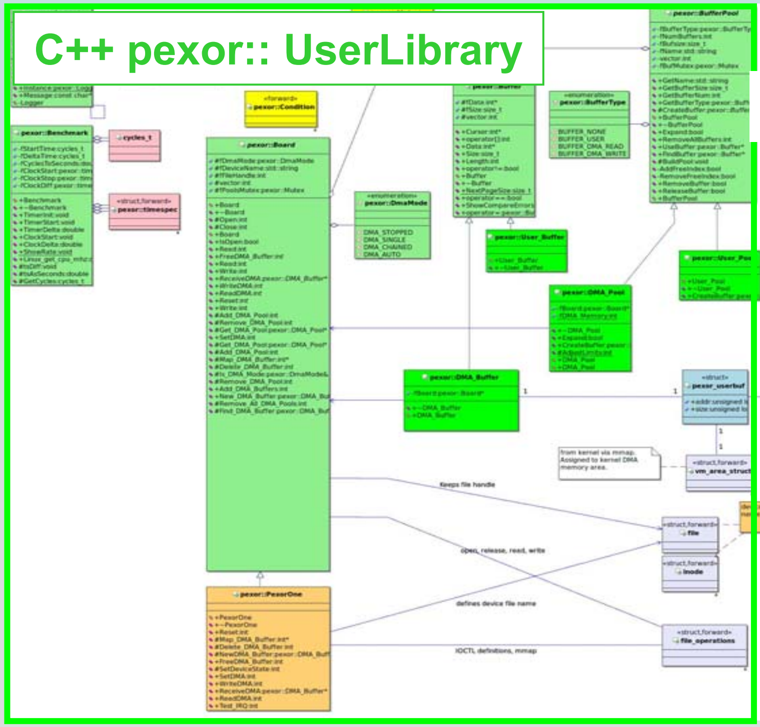


Device

```
CreateTransport()
WriteTestData()
RequestToken()...
```

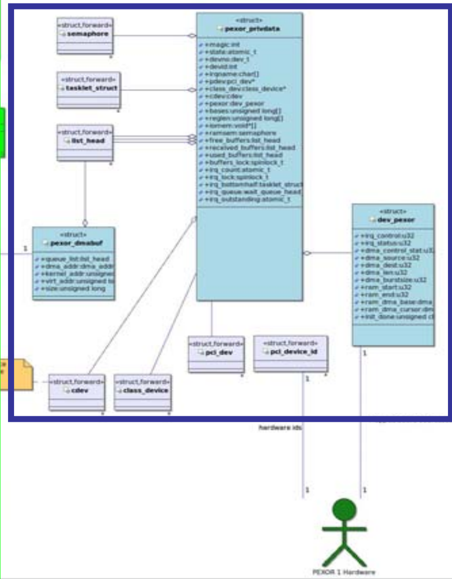
ReadoutApplication

```
CreateAppModules()
```



C++ pexor:: UserLibrary

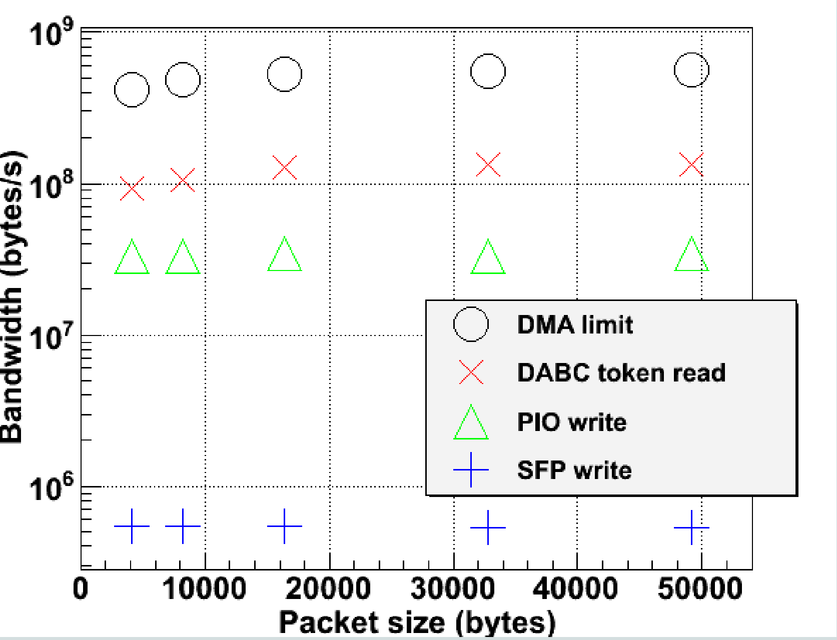
Linux Kernel module

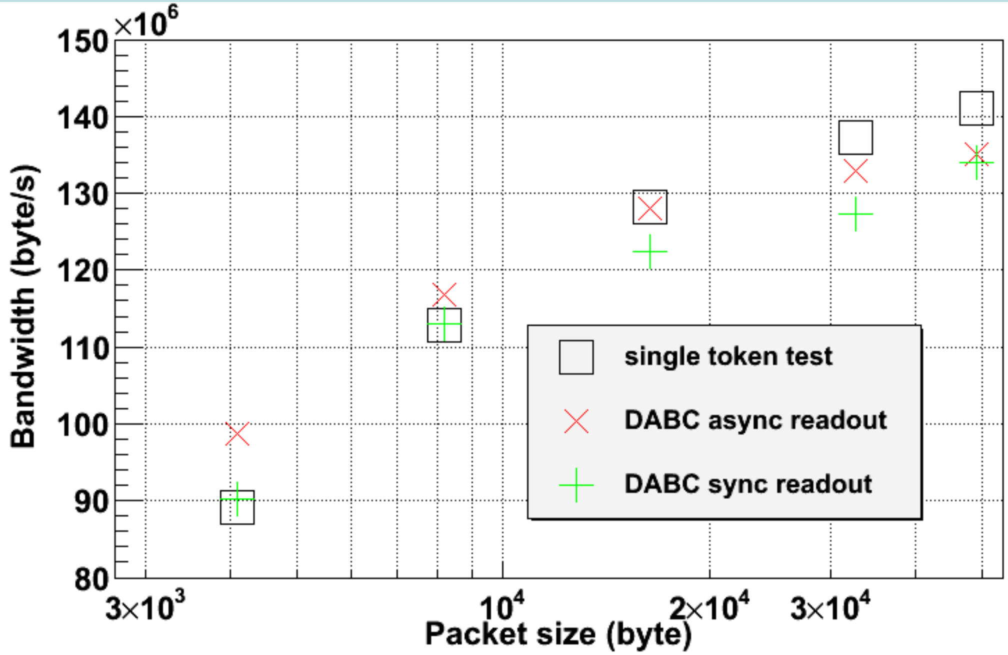


PEXOR driver release v0.95 (GSI, J.Adamczewski-Musch)

See presentation by H.G.Essel:
On-Line Object Monitoring with New Version V4.4 of Go4

PEXOR board tests





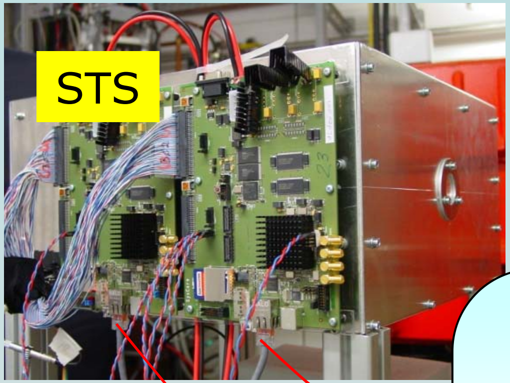
DABC Token Block Readout:

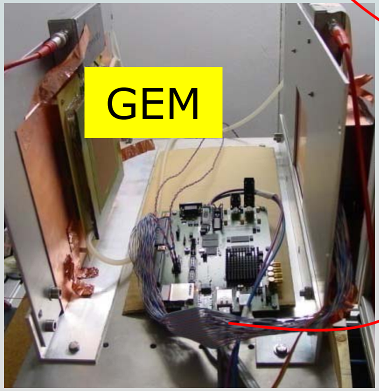
- data request from 2 frontends chained via 1 SFP and DMA to PC host
- Polling mode, no trigger IR -> maximum speed
- Data server + online monitor

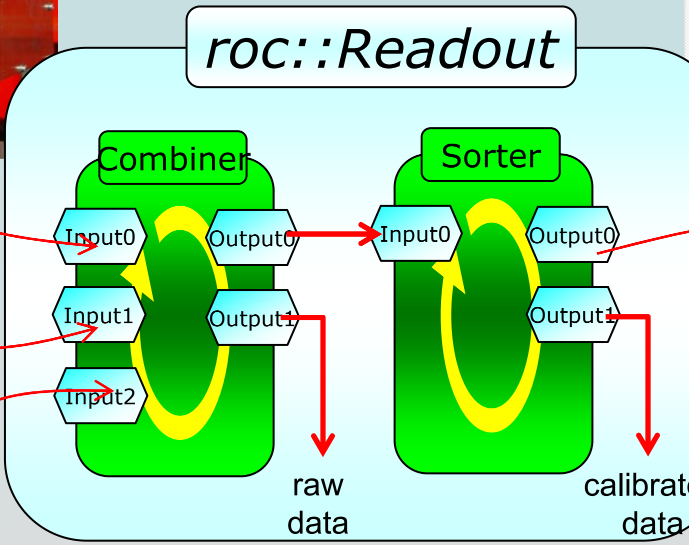
-> almost reaching plain test speed of driver library 130 MB/s
(2Gbit/s SFP connection)

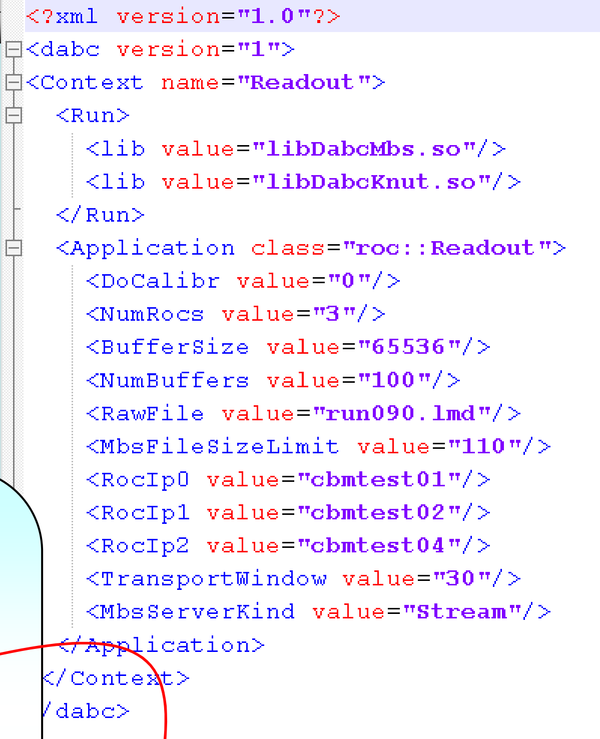
Example 2: DABC in CBM test beam

Compressed **B**aryonic **M**atter experiment for **FAIR**

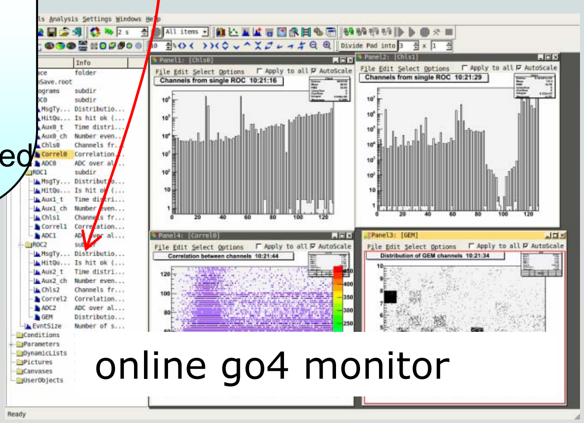








```
<?xml version="1.0"?>
<dabc version="1">
<Context name="Readout">
  <Run>
    <lib value="libDabcMbs.so"/>
    <lib value="libDabcKnut.so"/>
  </Run>
  <Application class="roc::Readout">
    <DoCalibr value="0"/>
    <NumRocs value="3"/>
    <BufferSize value="65536"/>
    <NumBuffers value="100"/>
    <RawFile value="run090.lmd"/>
    <MbsFileSizeLimit value="110"/>
    <RocIp0 value="cbmtest01"/>
    <RocIp1 value="cbmtest02"/>
    <RocIp2 value="cbmtest04"/>
    <TransportWindow value="30"/>
    <MbsServerKind value="Stream"/>
  </Application>
</Context>
</dabc>
```



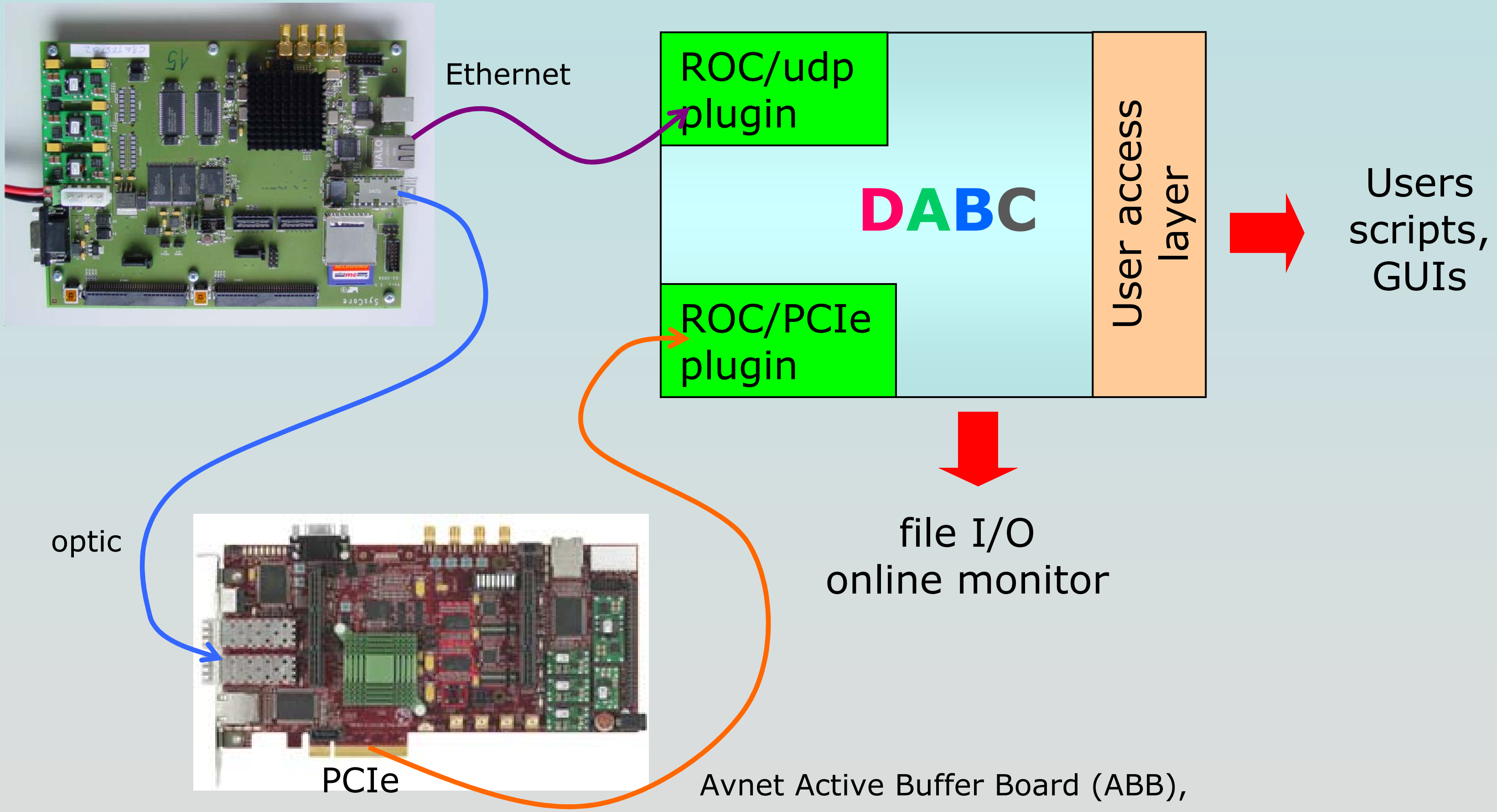
online go4 monitor

Presented by S.Linev, RT 2009

DABC as access layer to CBM ROC

CBM Readout controller (ROC),

Kirchhoff Institut für Physik, Heidelberg



Ethernet

ROC/udp plugin

DABC

User access layer

Users scripts, GUIs

optic

ROC/PCIe plugin

file I/O
online monitor

PCIe

Avnet Active Buffer Board (ABB),

ZITI, Heidelberg

Summary

DABC is modular C++ framework for DAQ processing on Linux

Supports distributed event building in network clusters

Custom hardware for data input can be implemented
by simple interface (Device and DataTransport)

PEXOR board is ready to use with DABC,
but driver and plug-in are still under further development
(trigger interrupt mode,…)

CBM experiment uses DABC as production system for test beam times
(next: June 2010)