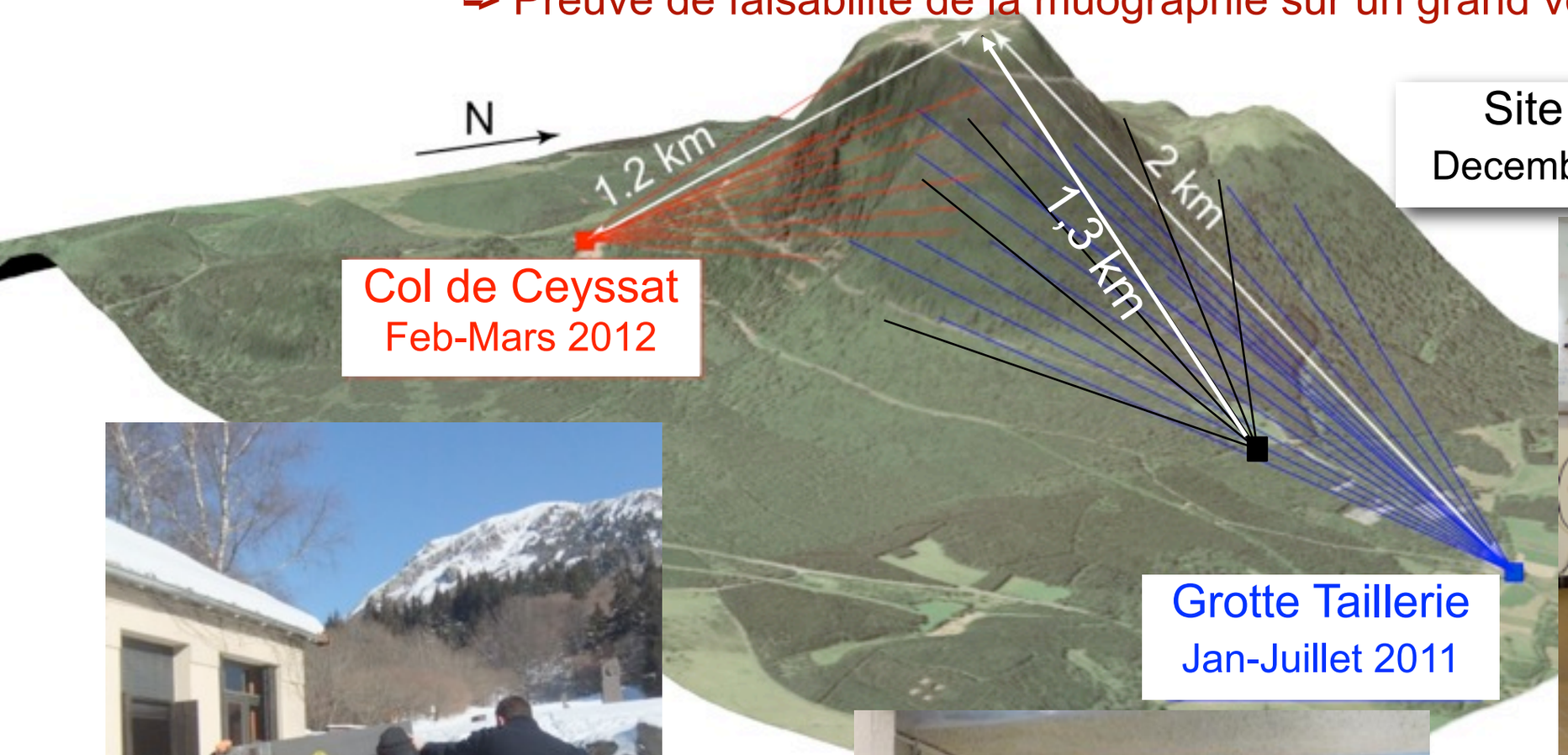
# DAQ TOMUVOL

C Cârloganu, 19.09.2014

# TOMUVOL sur le Puy de Dôme

But: la connaissance de l'historique du volcan de part sa structure pour prédire le comportement futur

➪ Preuve de faisabilité de la muographie sur un grand volcan (~2km à la base)




Site TDF
Decembre 2013





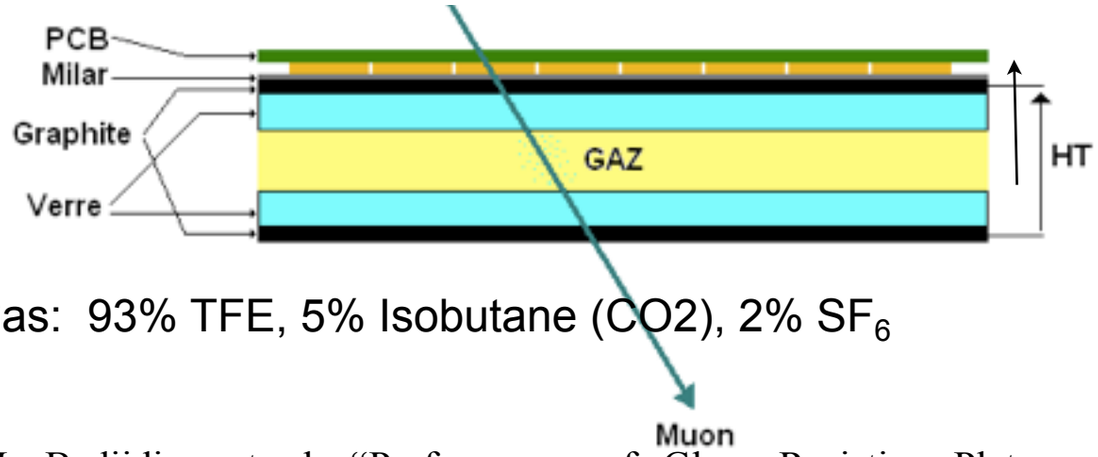
- 4 plans de 1m² x 1m² x 1 m².
- extension du télescope : 1 m .
- site en surface.
- 5cm de plomb -> 100 MeV
- résolution angulaire: ~5 mrad

- 4 plans de 1m² x 1m² x 0.66 m².
- extension du télescope : 1 m.
- site dans une auberge
- 30 cm de beton-> 140 MeV pour le muon
- résolution angulaire: ~5 mrad

- 3 plans de 1m² x 1m² x 0.16 m².
- extension du télescope : 0.5 m .
- site enterré (seuil variable, au dela du GeV
- résolution angulaire: ~10 mrad

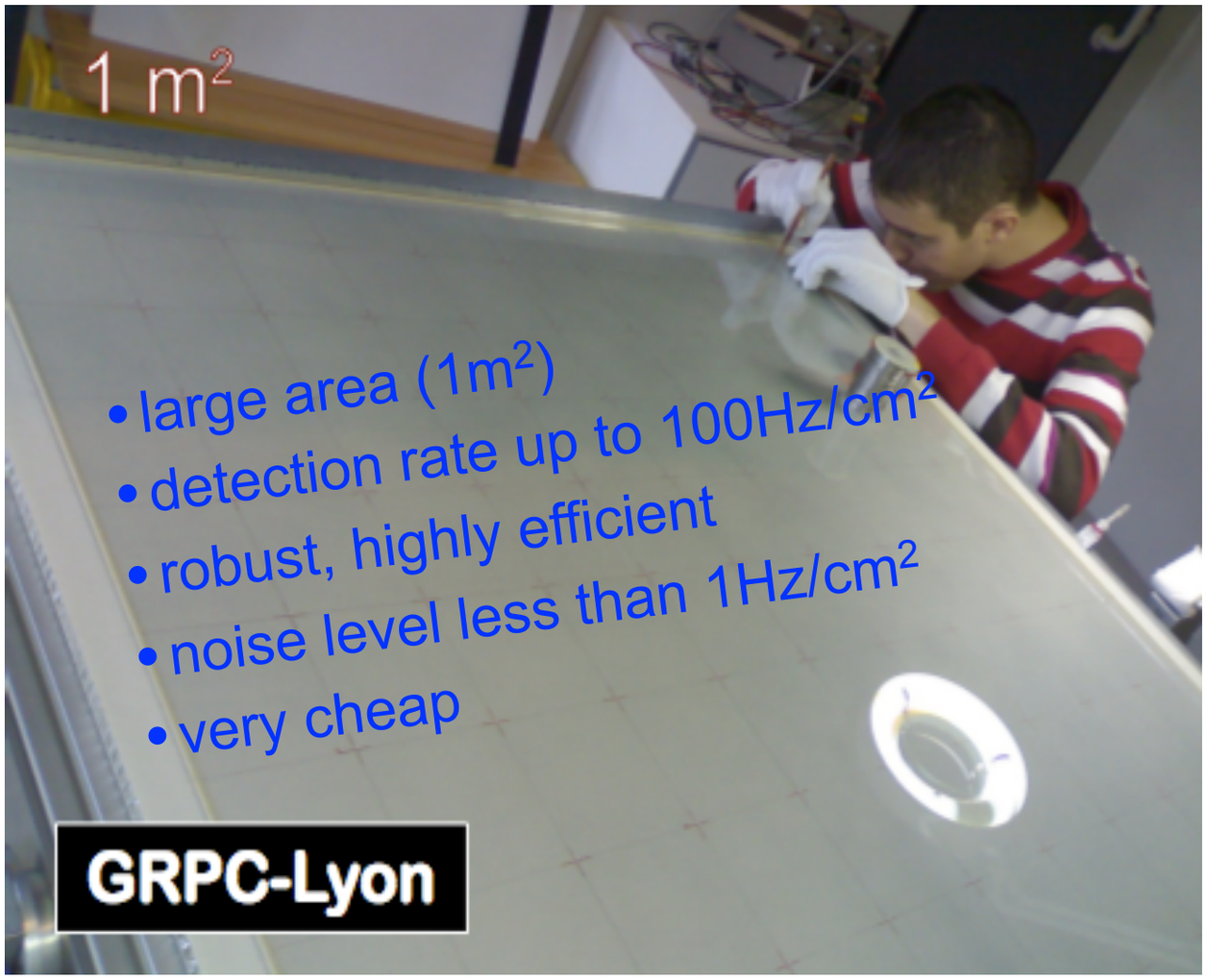
4 layers of 1 $m^2$ each with modular transportable design and improved timing.

Avalanche mode: total mean MIP charge 2.6pC, RMS: 1.6pC



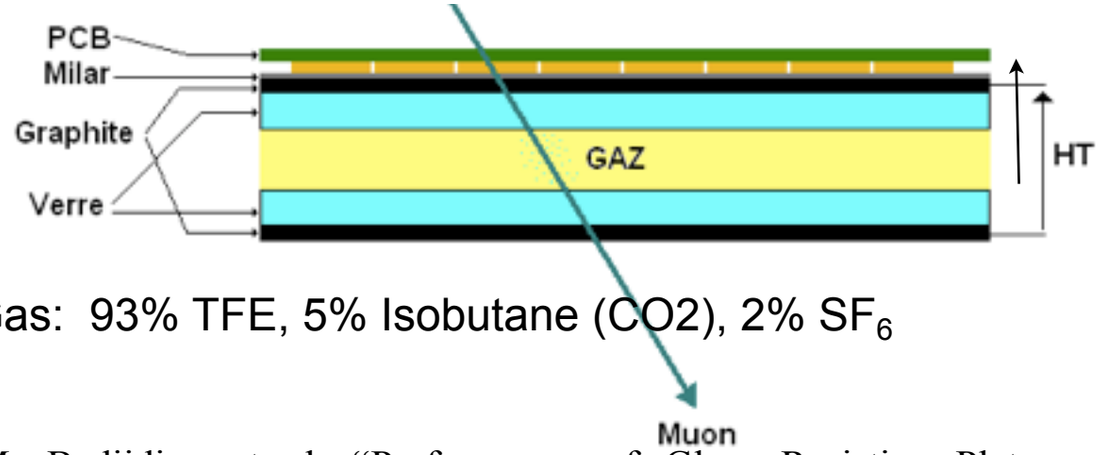

Gas: 93% TFE, 5% Isobutane ($CO_2$), 2% $SF_6$

M. Bedjidian et al, "Performance of Glass Resistive Plate Chambers for a high granularity semi-digital calorimeter", JINST 6:P02001,2011



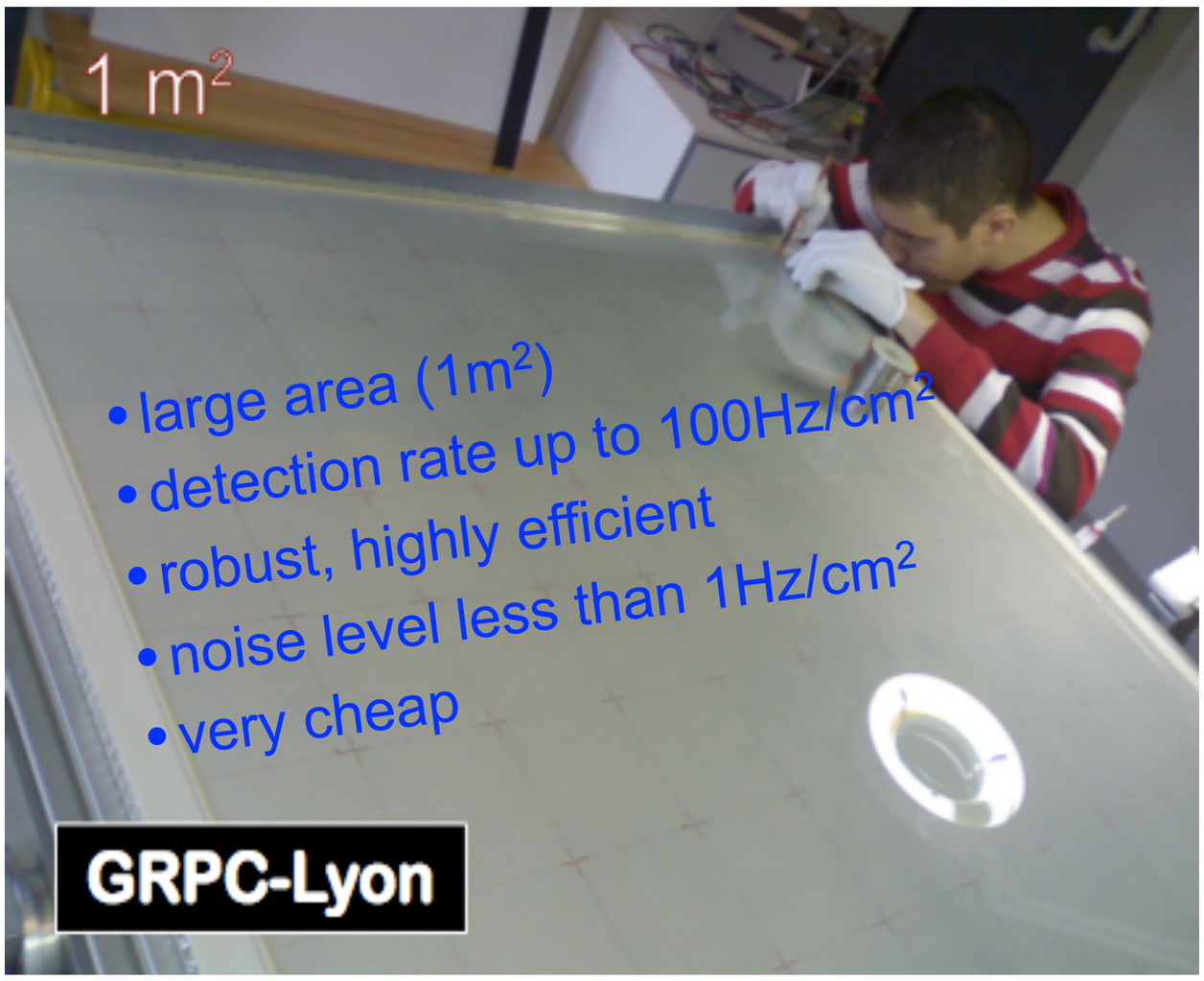
# CALICE GRPC's

Avalanche mode: total mean MIP charge 2.6pC, RMS: 1.6pC

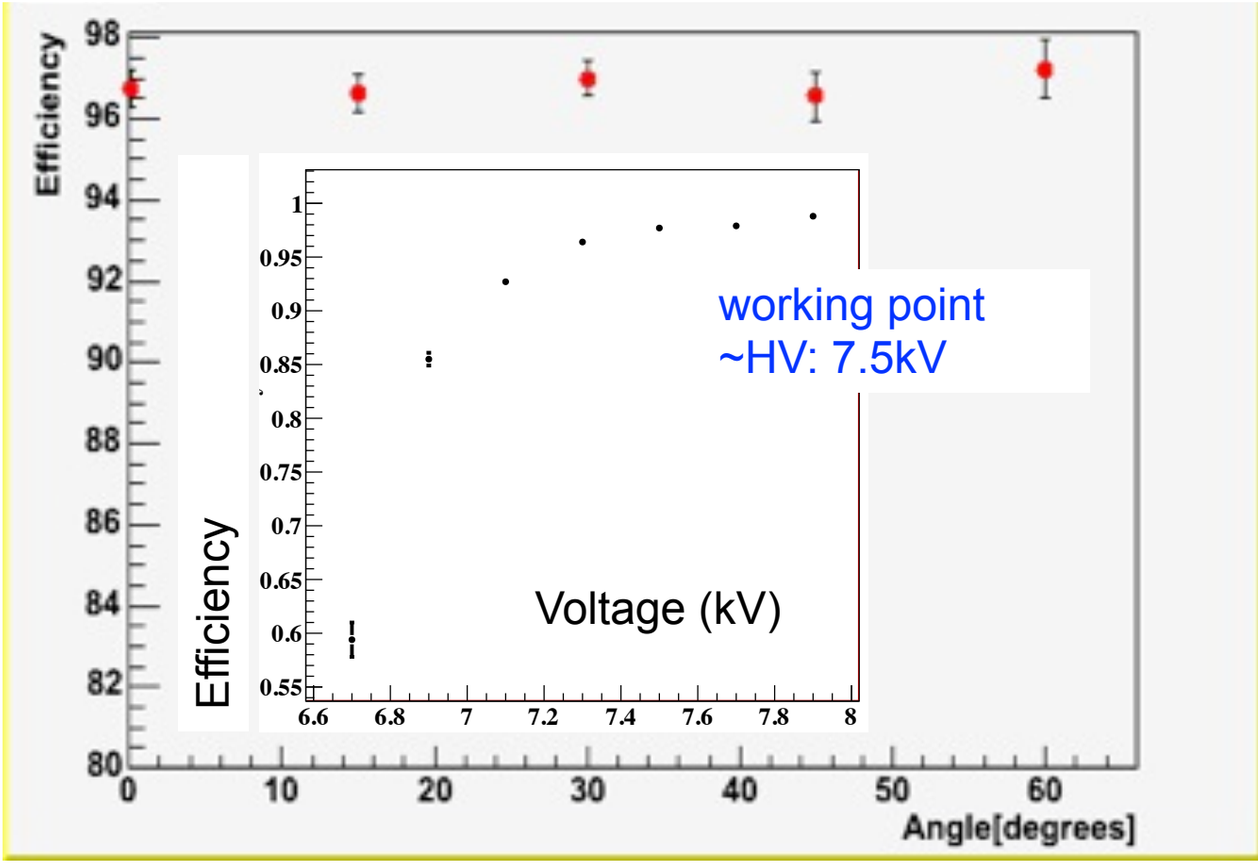



Gas: 93% TFE, 5% Isobutane ($CO_2$), 2% $SF_6$

M. Bedjidian et al, "Performance of Glass Resistive Plate Chambers for a high granularity semi-digital calorimeter", JINST 6:P02001,2011

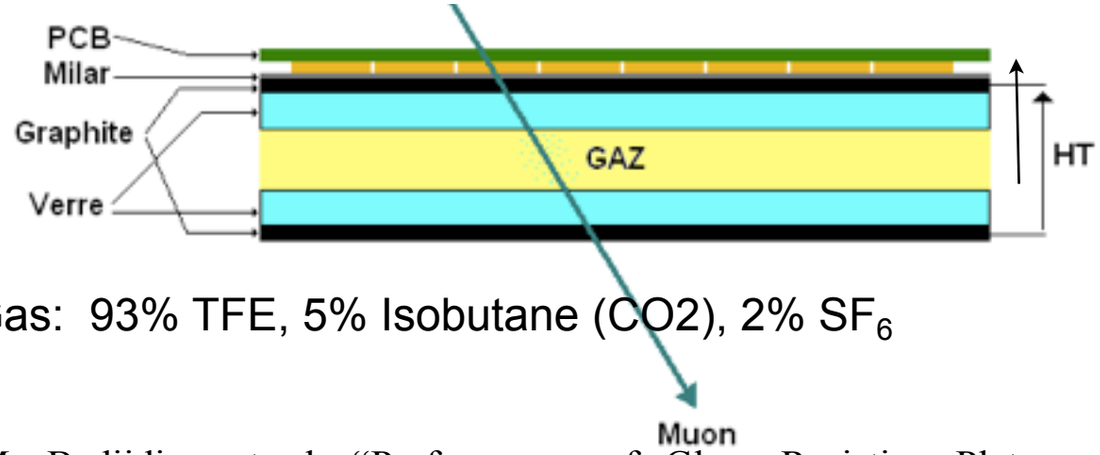



## Efficiency vs. HV & track incident angle



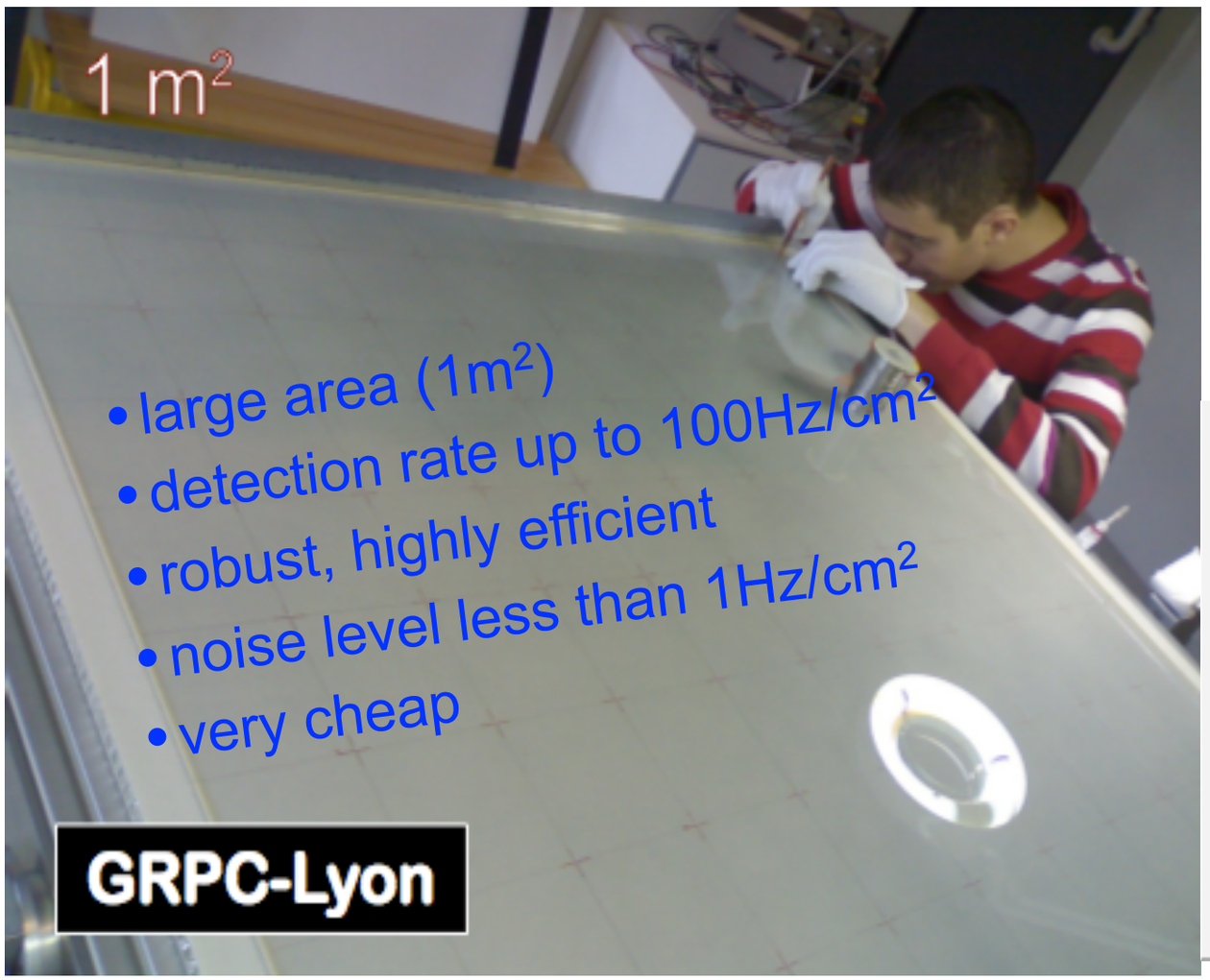
# CALICE GRPC's

Avalanche mode: total mean MIP charge 2.6pC, RMS: 1.6pC

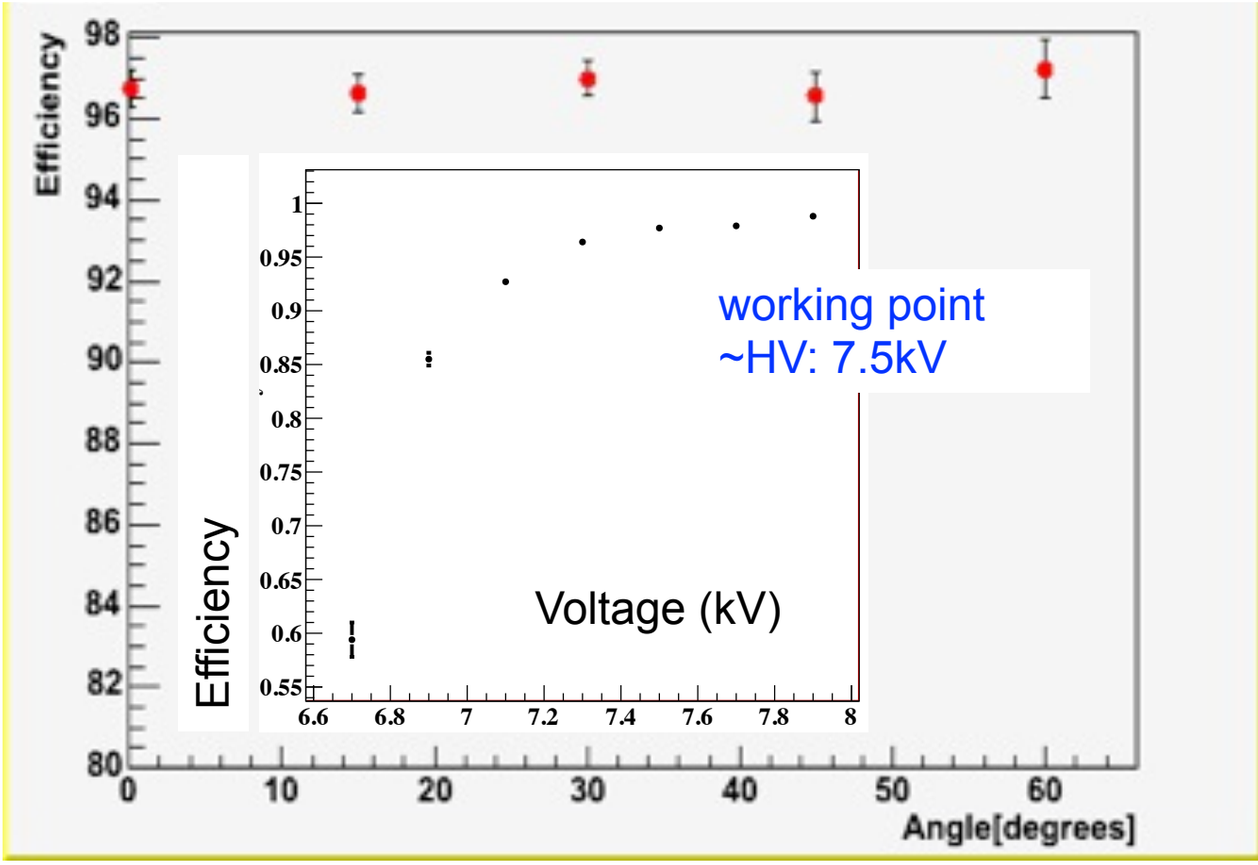



Gas: 93% TFE, 5% Isobutane ($CO_2$), 2% $SF_6$

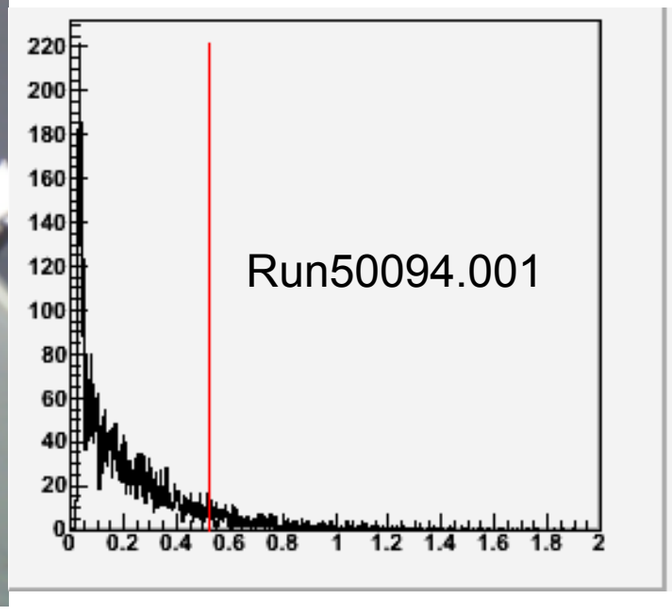
M. Bedjidian et al, "Performance of Glass Resistive Plate Chambers for a high granularity semi-digital calorimeter", JINST 6:P02001,2011

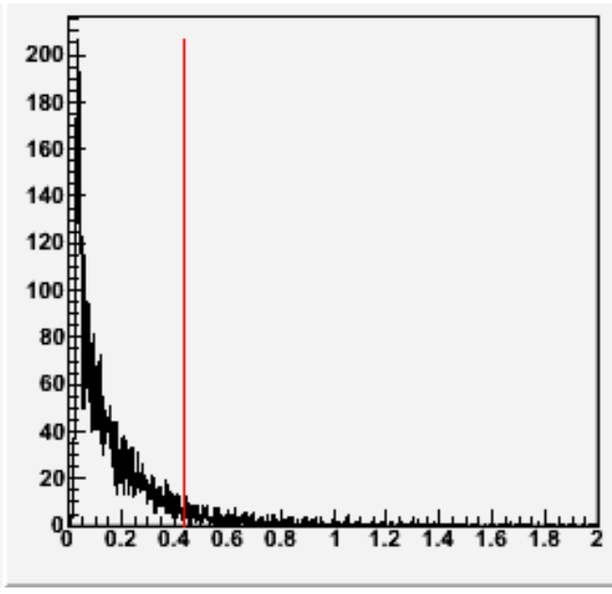



## Efficiency vs. HV & track incident angle

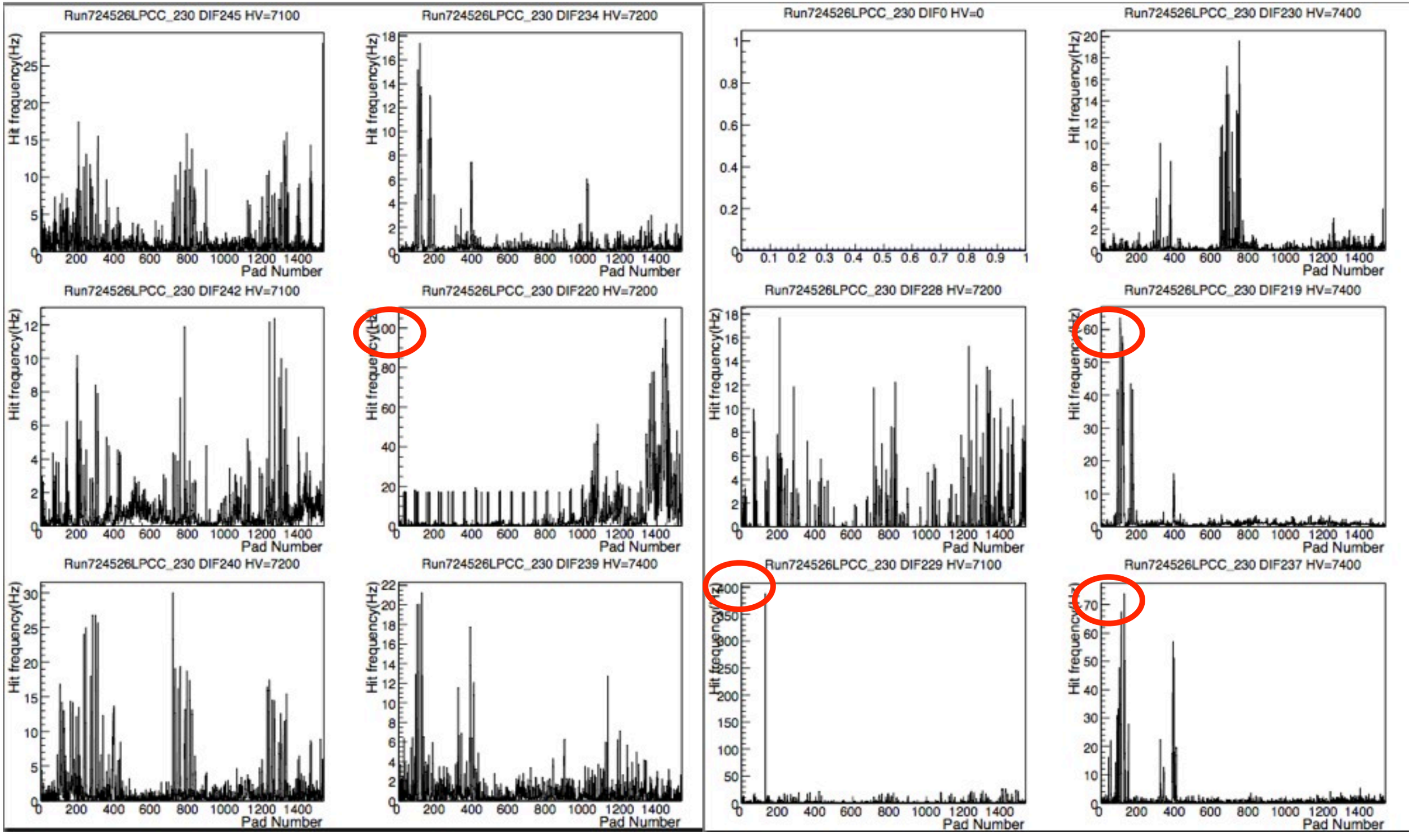



## Noise rate (Hz)

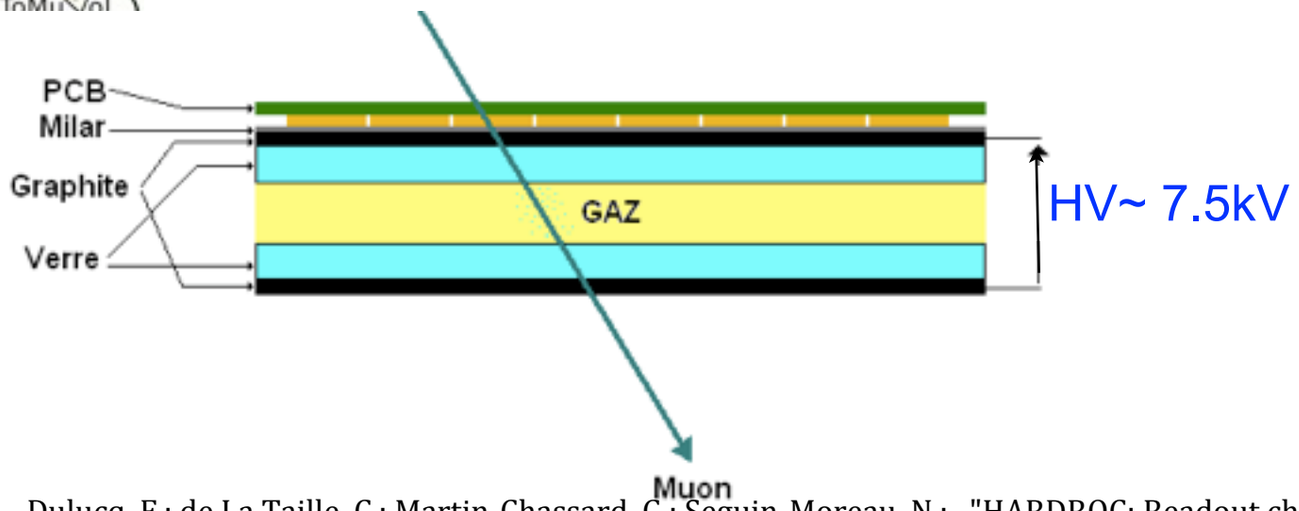


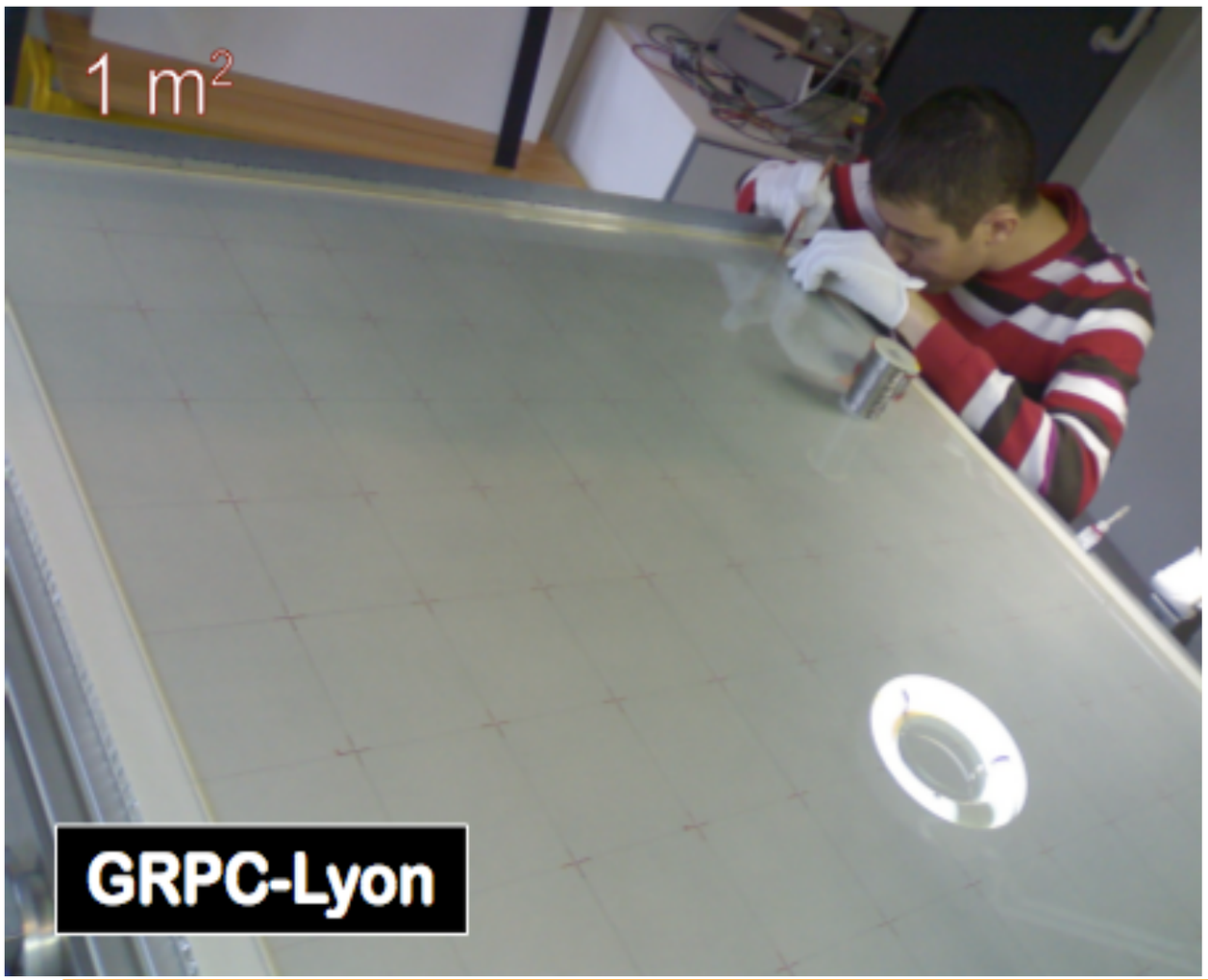
# Snapshot à l'allumage le 12.09.2014



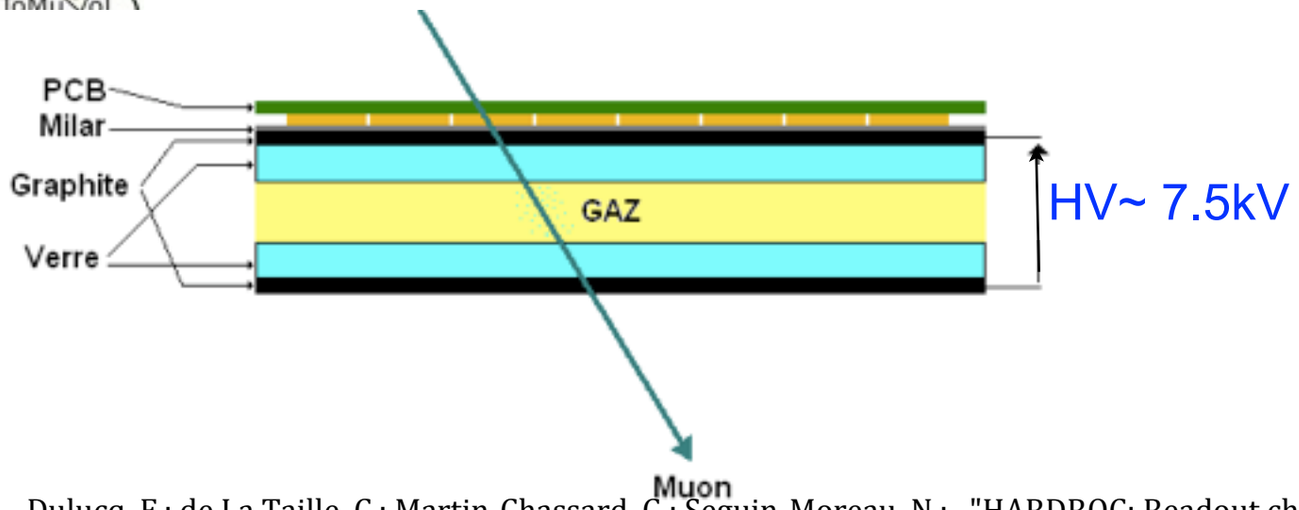
# Muon Tracker : CALICE Electronics



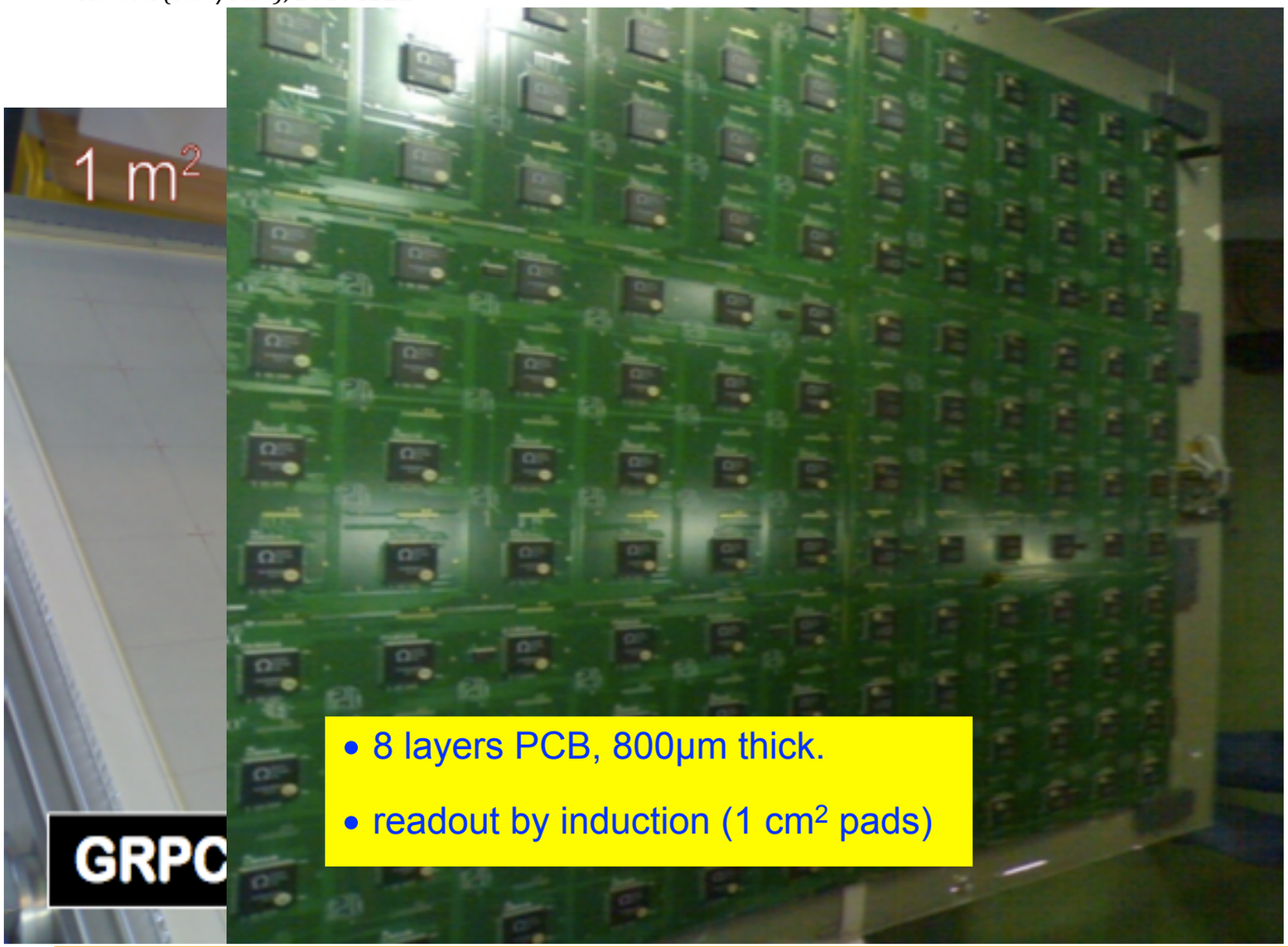

Dulucq, F.; de La Taille, C.; Martin-Chassard, G.; Seguin-Moreau, N.; , "HARDROC: Readout chip for CALICE/EUDET Digital Hadronic Calorimeter," *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*



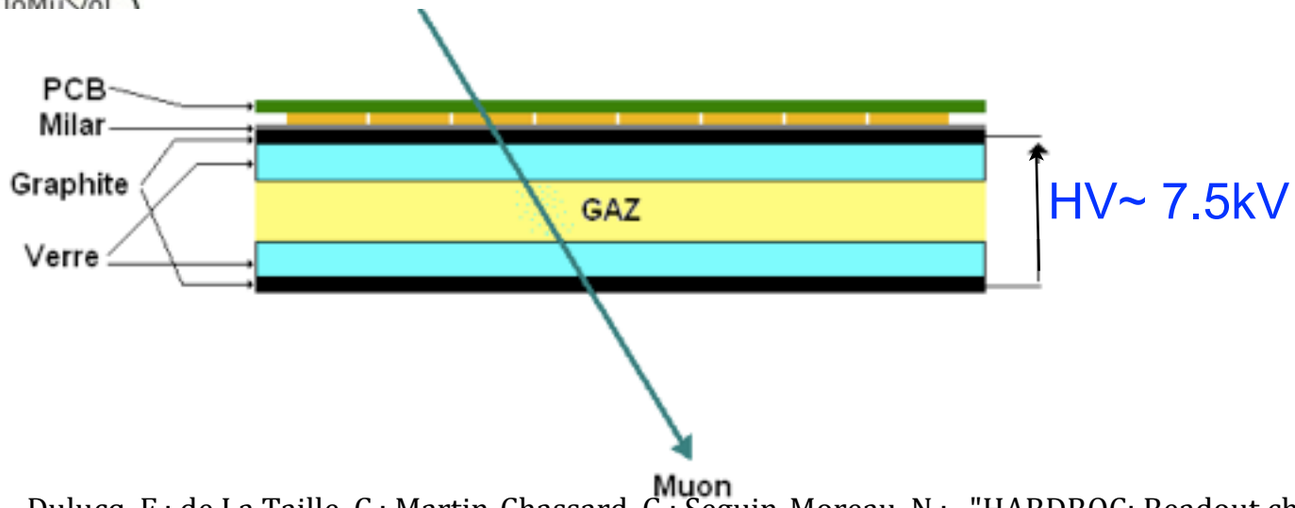
# Muon Tracker : CALICE Electronics



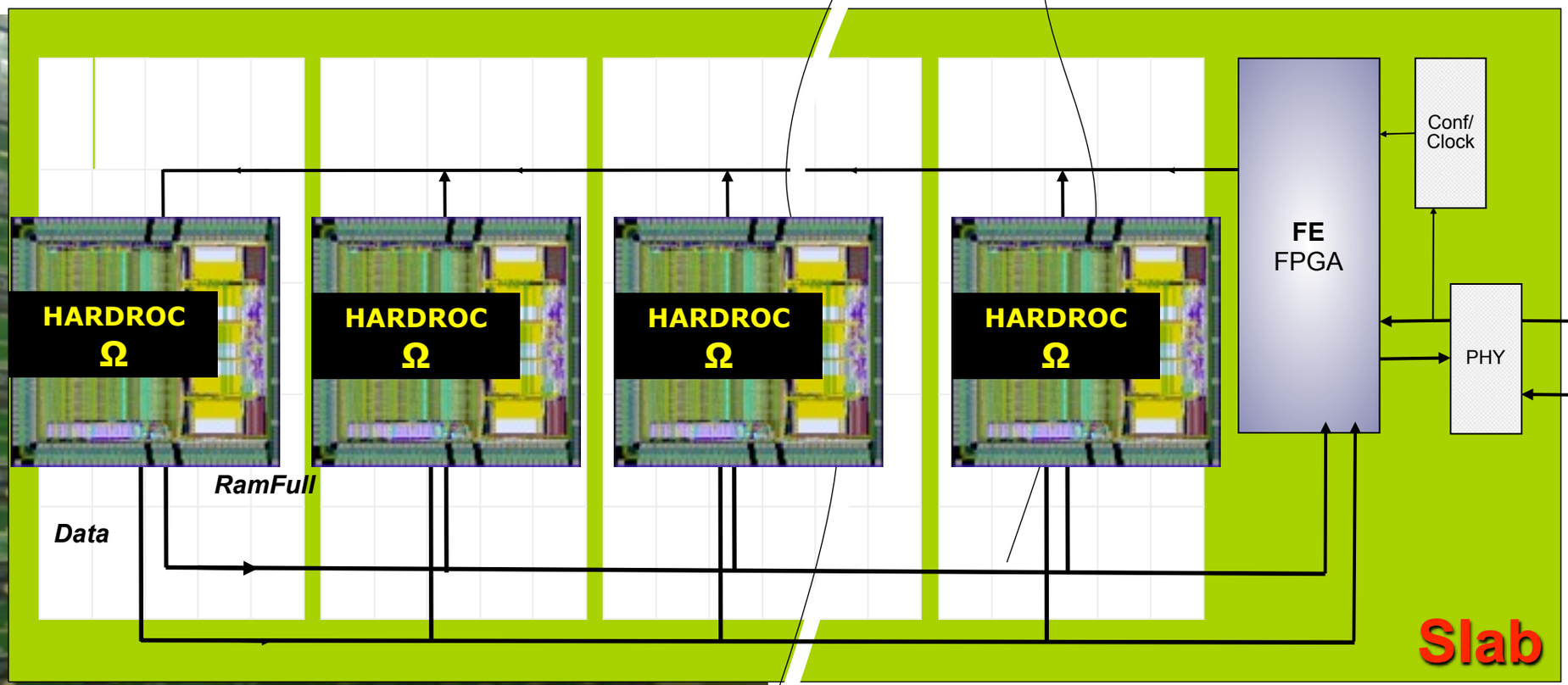

Dulucq, F.; de La Taille, C.; Martin-Chassard, G.; Seguin-Moreau, N.; , "HARDROC: Readout chip for CALICE/EUDET Digital Hadronic Calorimeter," *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*



# Muon Tracker : CALICE Electronics



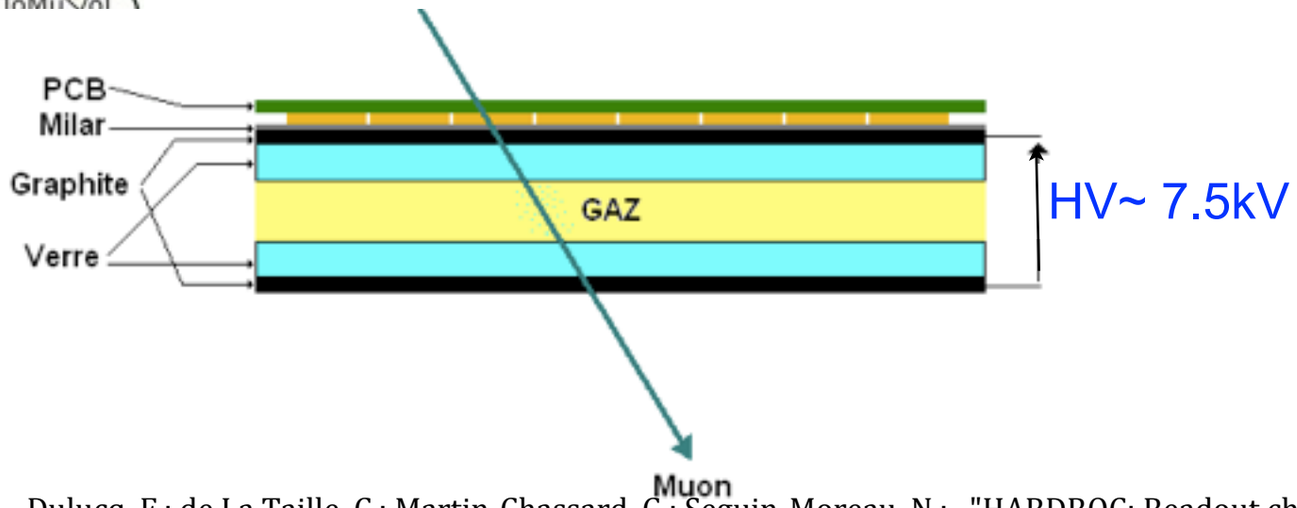

Dulucq, F.; de La Taille, C.; Martin-Chassard, G.; Seguin-Moreau, N.; , "HARDROC: Readout chip for CALICE/EUDET Digital Hadronic Calorimeter," *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*

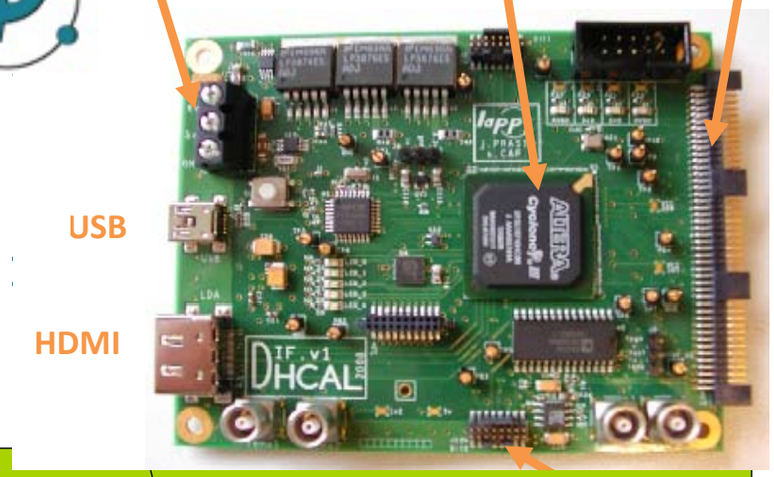

- 8 layers PCB, 800µm thick.
- readout by induction (1 cm² pads)

- 64 channels, 16 mm²
- digital output (3 adjustable thrs)
- low power consumption (1.5 mW/ch)
- large gain range
- xtalk <2%
- ajustable gain for each channel

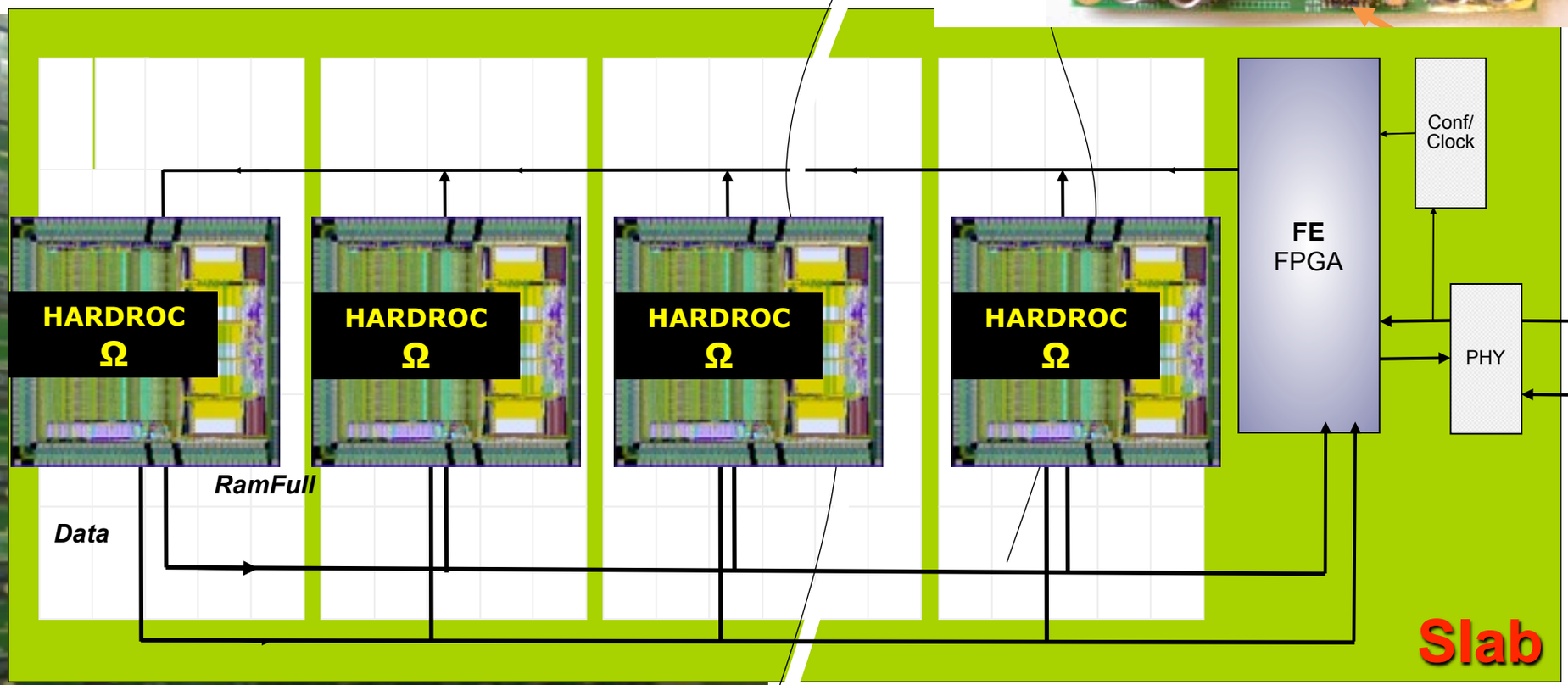
# Muon Tracker : CALICE Electronics




Dulucq, F.; de La Taille, C.; Martin-Chassard, G.; Seguin-Moreau, N.; , "HARDROC: Readout chip for CALICE/EUDET Digital Hadronic Calorimeter," *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*



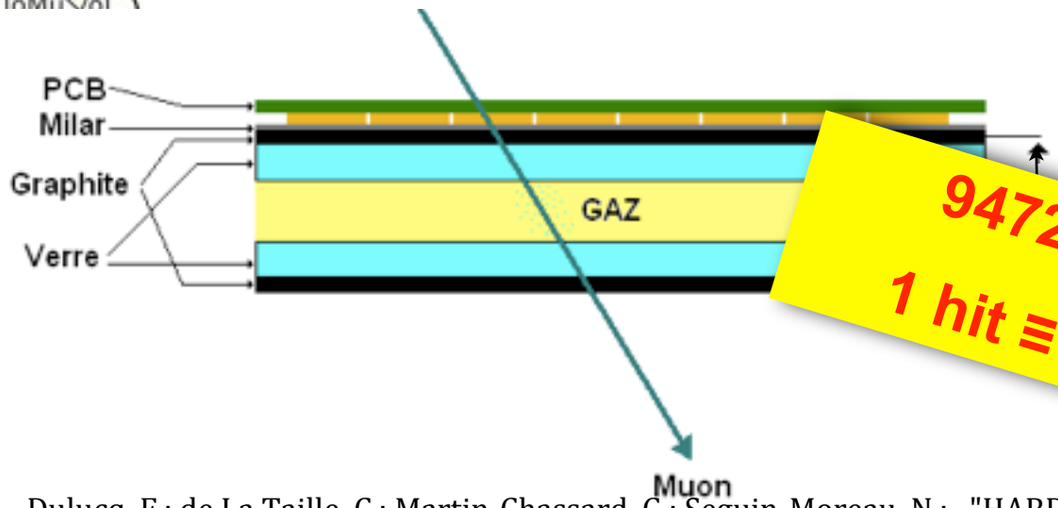





- 8 layers PCB, 800µm thick.
- readout by induction (1 cm² pads)

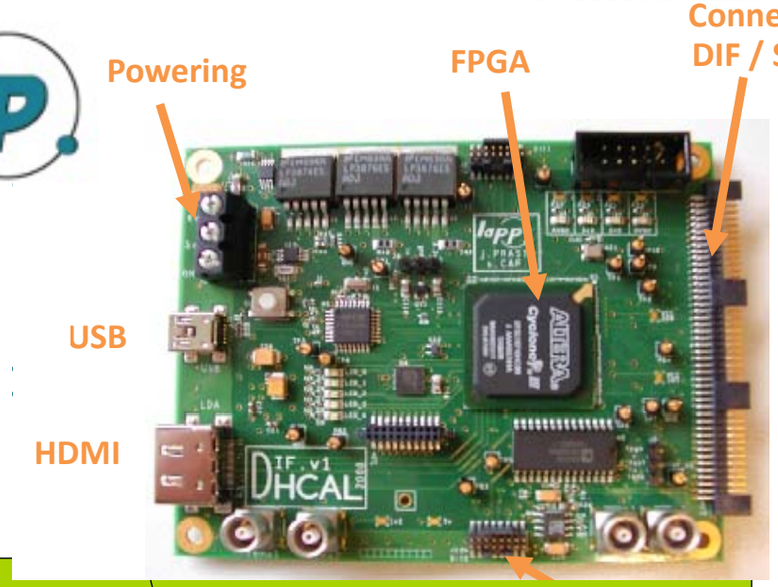
- 64 channels, 16 mm²
- digital output (3 adjustable thrs)
- low power consumption (1.5 mW/ch)
- large gain range
- xtalk <2%
- ajustable gain for each channel

# Muon Tracker : CALICE Electronics

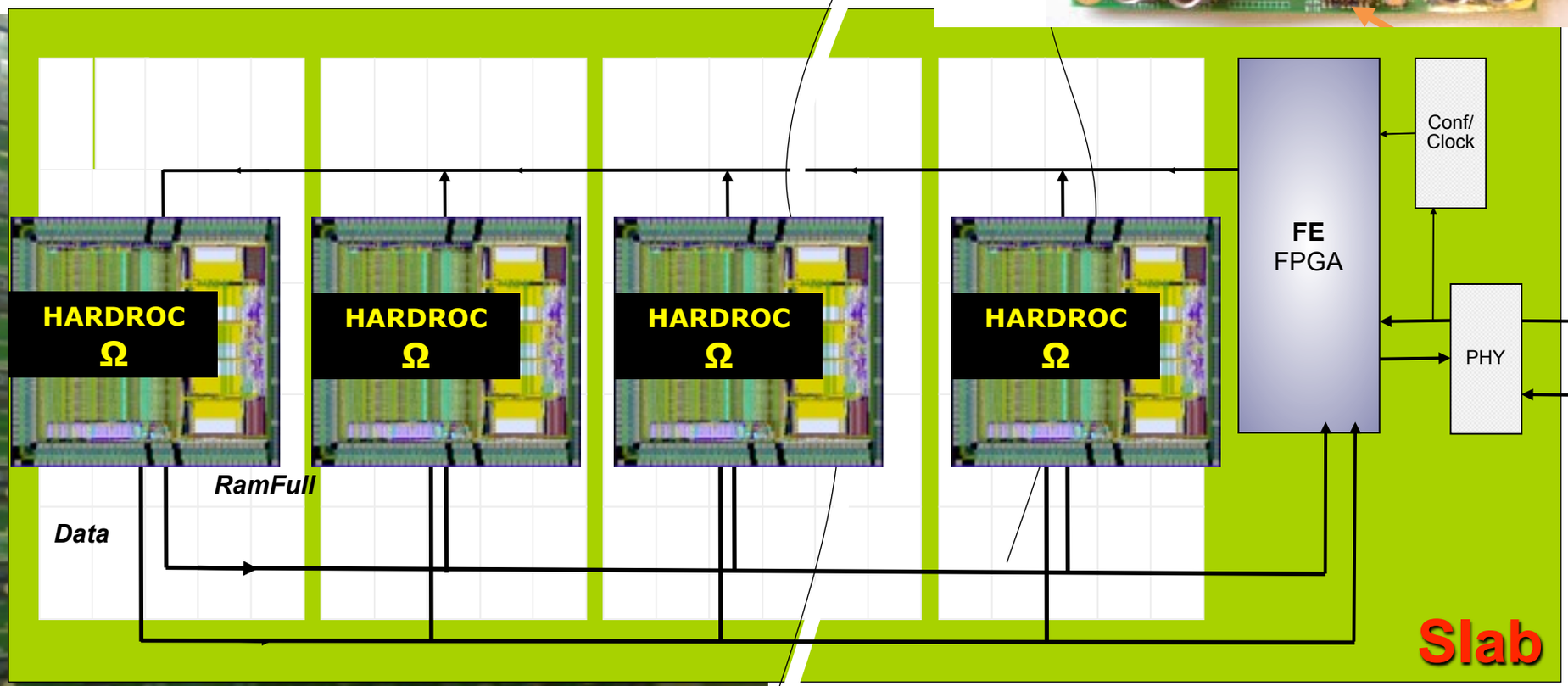



**9472 channels/m2**
**1 hit ≡ time + thresh**

Dulucq, F.; de La Taille, C.; Martin-Chassard, G.; Seguin-Moreau, N.; , "HARDROC: Readout chip for CALICE/EUDET Digital Hadronic Calorimeter," *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*



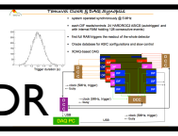


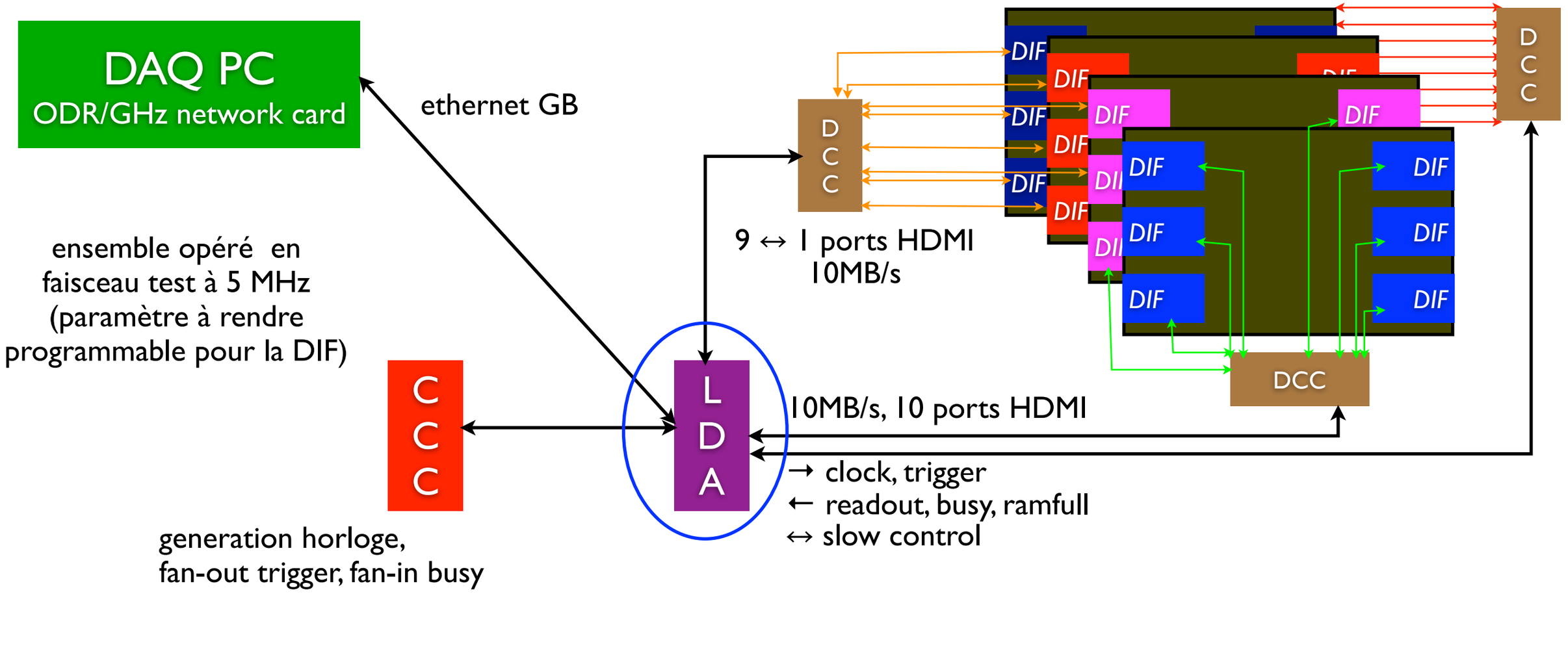



- 8 layers PCB, 800µm thick.
- readout by induction (1 cm² pads)

- 64 channels, 16 mm²
- digital output (3 adjustable thrs)
- low power consumption (1.5 mW/ch)
- large gain range
- xtalk <2%
- ajustable gain for each channel

# CALICE DAQ appliquée à TOMUVOL ?

| Acronyms | | Origin |
|---|---|---|
| DIF | Detector InterFace | LAPP |
| LDA | Link/Data Aggregator<br>fans in/out DIFs and drives links to ODR  | UCL/LLR |
| GDCC | 10 entrées/sorties HDMI, remplace la LDA, même I/Os | LLR |
| CCC | Clock and Control Card | UCL/Mainz |
| ODR | Off-Detector Receiver ➙ remp avec carte GHz | |
| DCC | Data Concentrator Card | LLR |



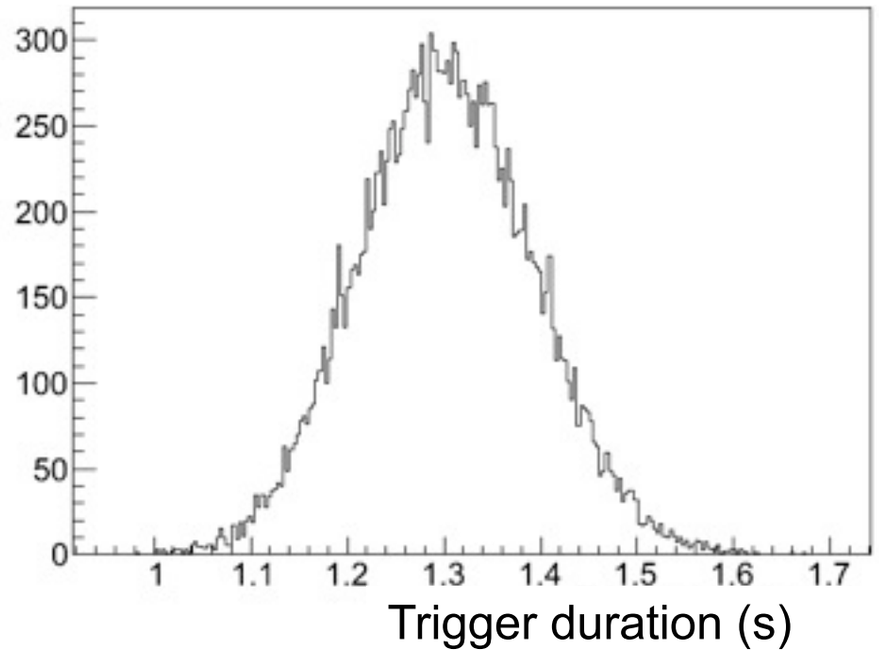
# Acquisition and control system

- Scalable
  - Tree architecture
  - Successive concentration of data
- Standard
  - Rely on standard protocols, data coding
    - Ethernet, optical links,
    - Serial 8b10B
  - Cables and connectors
    - Hdmi
    - CAT5, RJ45
    - Optical
  - Front end interface unified among the detectors (CALICE standard)
- but some limitations
  - Low speed access to a chain of 10-100 very front end chips (1k-10k channels) : 1-5 Mbit/s (daisy chain and reduced number of connections for compacity, length of PCB traces, low power consumption)
  - Therefore assume : auto trigger, zero suppression at VFE level
  - TFC interleaved with SC, DAQ using 8b/10n protocol : limited timing precision (>10 ns)
  - Events build at VFE level, inter-bunch train read out : limited buffering capacity (10-100 evt)

Low occupancy of the detectors is assumed (<0.5%/cell/bunch)

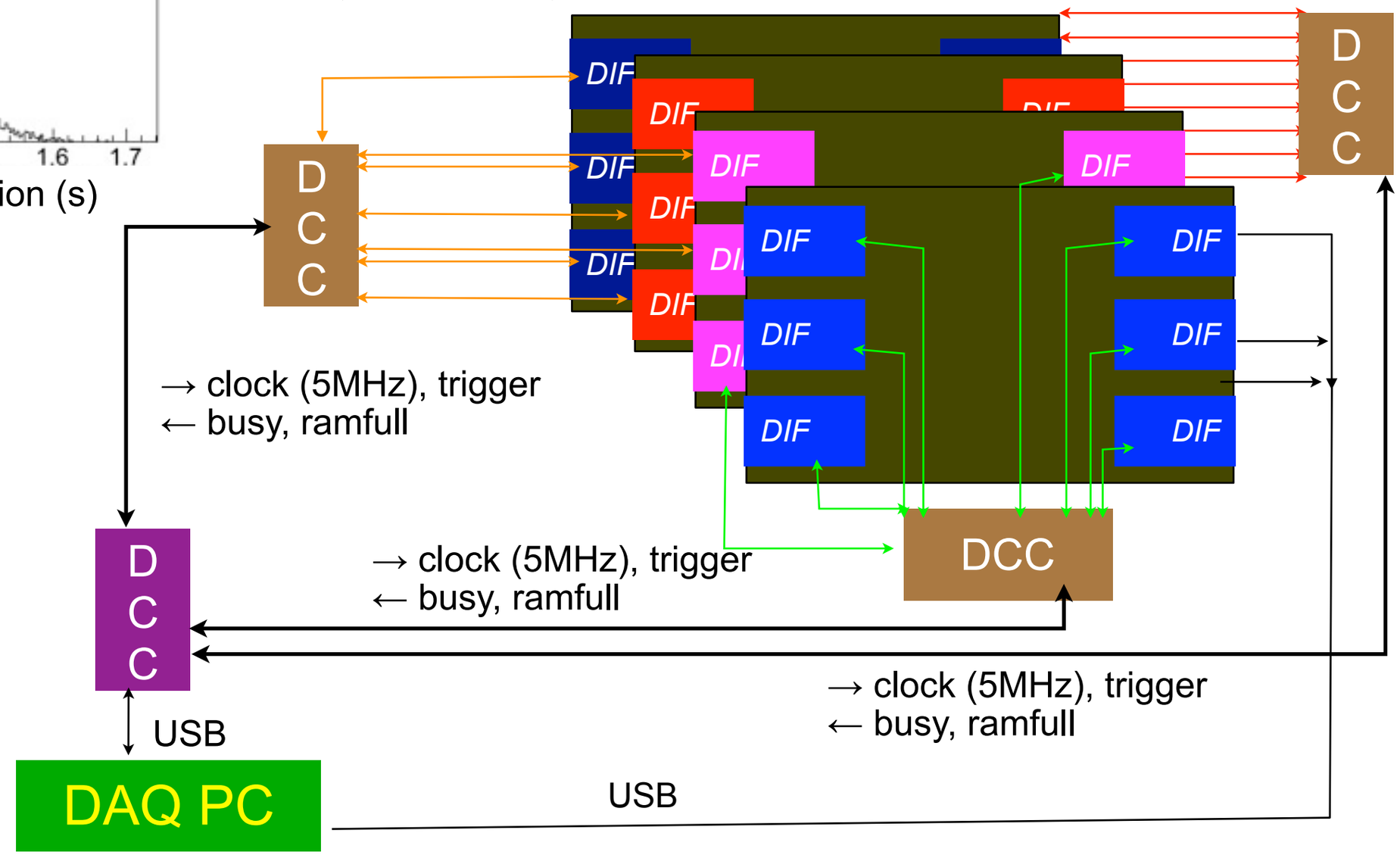
- Compact
  - Serial links used everywhere
  - “one cable for everything” : DAQ, TFC, SC
  - Front end components
- Flexible
  - (re)programmable parts (fpga)
  - Routing, switching, buffering of data packets

# Tomuvol Clock & DAQ Synopsis

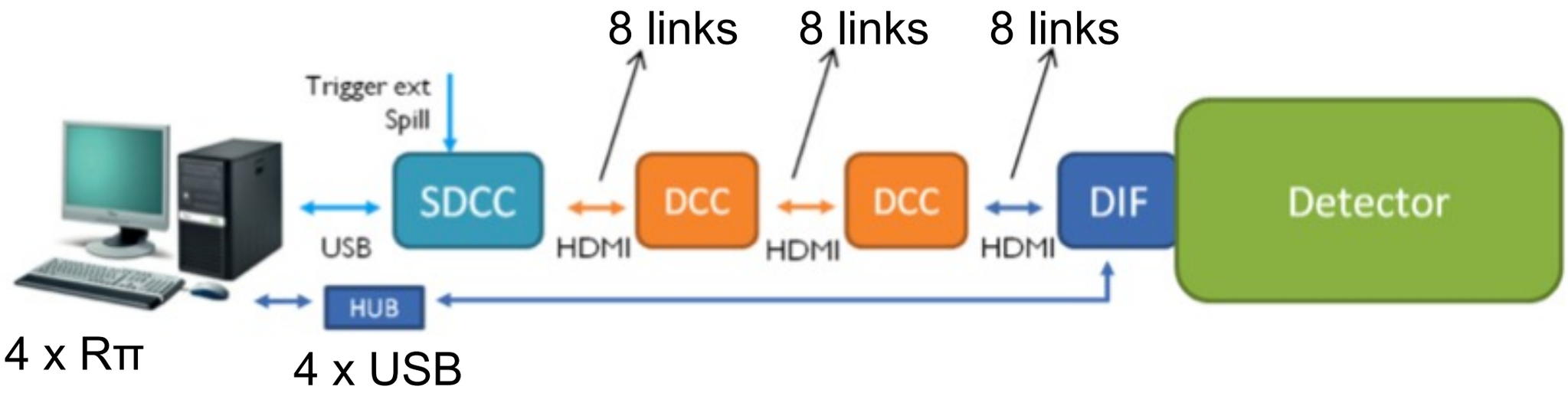



- system operated synchronously @ 5 MHz
- each DIF reads/controls 24 HARDROC2 ASICS (autotrigged and with internal RAM holding 128 consecutive events)
- first full RAM triggers the readout of the whole detector
- Oracle database for ASIC configurations and slow control
- XDAQ-based DAQ

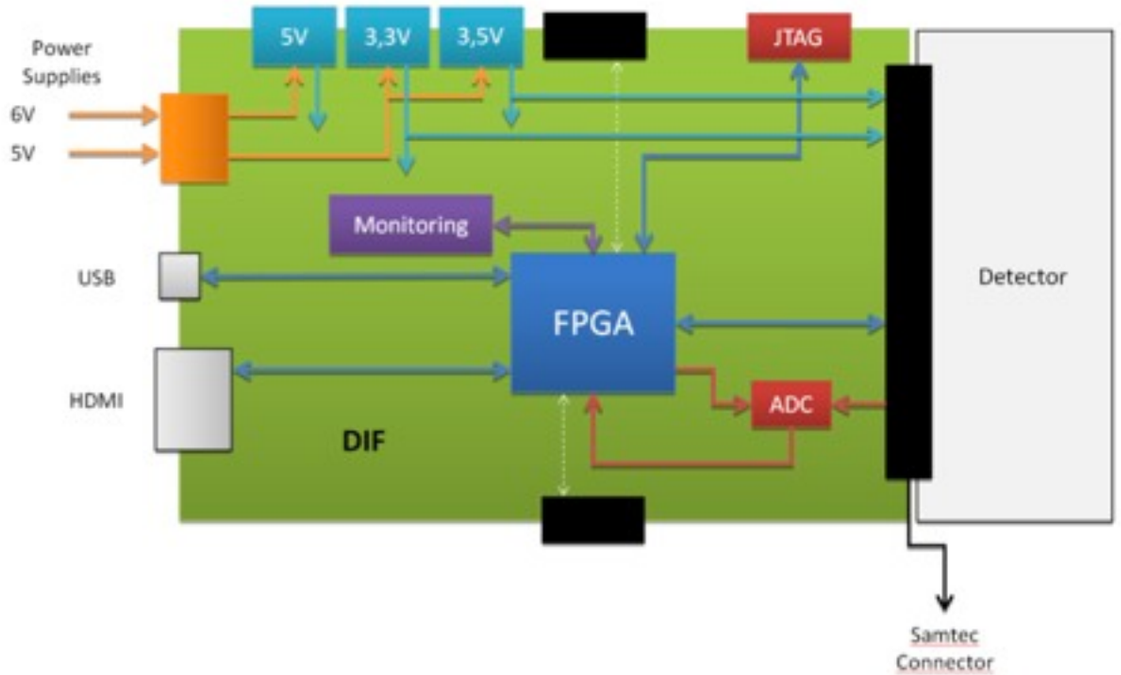
The protocol used for the HDMI commands is a homemade protocol which includes a header, the command number and a checksum. Almost all the HDMI signals are real LVDS signals



# Architecture générale de la DAQ




The DIF architecture




For every readout operation, the ASIC data are encapsulated by the DIF FPGA with a header, a trailer and the following information before they are sent to the computer:

1. DIF Trigger Counter (DTC): Coded in 32 bits, it counts the number of readouts. It is reset at the first acquisition of each run.
2. Information Counter (IC) : Coded in 32 bits. Bits 23 to 0 are used to count the dead time. This occurs when ASIC are not acquiring. It is reset every acquisition. Bits 31 to 24 are used for BCID counter overflow. It is reset at the first acquisition of each run.
3. Global Trigger Counter (GTC) : Coded in 32 bits. It counts the number of triggers received by the DIF when the Trigger mode is run and counts the number of readouts in the Triggerless mode. It is reset at the first acquisition of each run.
4. Absolute BCID : Coded in 48 bits. It is incremented with the 5 MHz clock received from the SDCC. It is reset at the first acquisition of each run.

SDCC

• Trigger signal: In Trigger mode, the DAQ needs the trigger signal to stop the acquisition and start the readout of the detectors.
• Busy and Ramfull signals :The Busy and Ramfull signals are used to maintain the synchronization and the automation of the different processes of the DAQ system. Both signals are sent from the DIF to the SDCC. In Triggerless mode the Ramfull signal initiates the readout phase. Concerning the the Busy signal, it is is active when the ASIC are being read out. While the Busy signal is active, a new start acquisition can not be sent. But as soon as the last Busy is unset, the SDCC sends a new StartAcquisition command to the different DIF to launch a new acquisition.

# DAta Acquision Software

## Low level hardware acces

**USB readout**
DIF and SDCC FPGA are interfaced with the same USB chip (FTDI). The DIF is thus uniquely identified by its FTDI device identifier (id) stored in an EEPROM and access to a specific device id is done using either the proprietary library FTD2XX or the free version library libFTDI. A base class called UsbDeviceDriver implements a set of low level access: bytes read or write , DIF and SDCC registers access and finally predefined commands.

**DIF and SDCC readout**
An upper layer software for DIF configuration and readout. The class called BasicUsbDataHandler aggregates a pointer to a UsbDeviceDriver and to a configuration buffer handling all DIF and SC parameters. Specific methods are used to implement the DIF configuration, the ASIC configurations and a single event readout to an internal buffer. Similarly, an SDCCDataHandler implements commands associated to the SDCC.

**DIF Manager**
Eventually one DIF manager class per PC is responsible of the data taking of the DIF through the USB connected to it. It handles the DIF and ASIC configuration parameters via an interface called DIFManager. This scans and detects all connected DIF, instantiates one DIFDataHandler class per detected device and distributes configuration parameters. When data taking starts, it starts a polling thread continuously reading events on all connected DIF. Events can be directly dumped to disk in LCIO format or transfer to the central data acquisition. Figure 21 summarizes this architecture.

## Configuration database

Gives the possibility to store and retrieve all parameters needed by the DAQ system. It is hosted on an Oracle server at CC IN2P3.

Homemade C++ library to interface this SQL database with the DAQ software and to allow users to insert and query data without knowledge of SQL.

Part of the DAQ system being written in Python, we used Swig to generate a Python version of the C++ library. The system has been designed to easily allow the addition or modification of existing object parameters. It is built on 2 levels: the database itself and the C++ library.

**Database model** :
The database model is conceived to deal with extendable number of ASIC. It can also handle different kinds of ASIC using different settings of parameters to take into account addition of other sub-detectors. In this model, each ASIC has a unique entry in the ASIC table, containing its type. The actual configuration parameters are contained in 2 tables : the first is ASIC_CONFIG for the parameters common to all ASIC types and the second <ASIC_TYPE>_CONFIG for the parameters specific to a given type. For instance the configuration parameters for a HARDROC ASIC will be stored in the ASIC_CONFIG table for the common part and in the HR2_CONFIG table for the HARDROC specific parameters4. As a consequence, supporting a new type of ASIC requires only to create a new table containing the specific configuration parameters. When downloading the parameters, the system will automatically choose the accurate tables associated to the type of the concerned ASIC.

## Data collection & monitoring

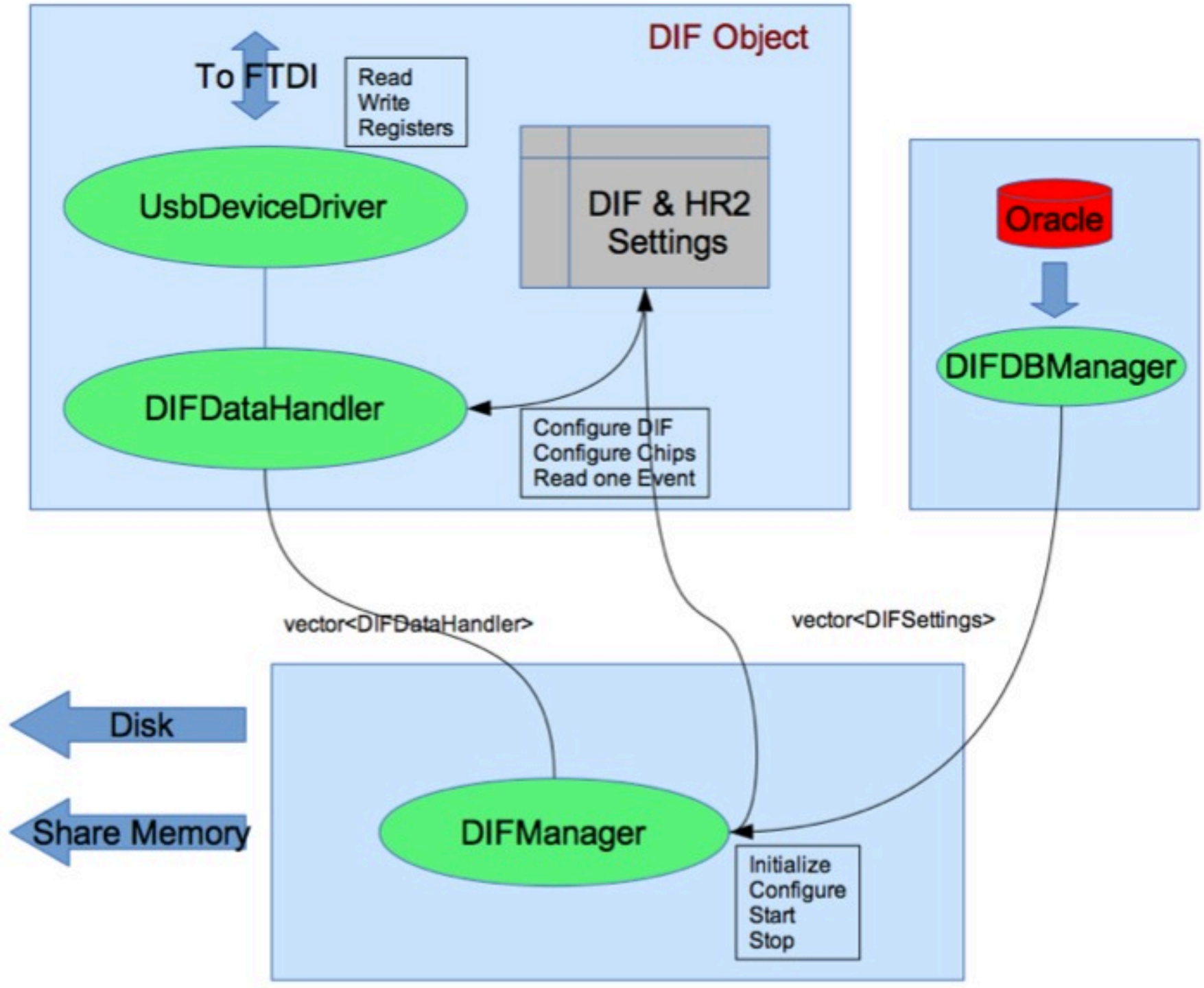
**Data acquisition access**

In order to minimize database access during data acquisition, a class called the DIFDBManager is responsible of the download of all the DIF and ASIC configuration parameters of a given setup and to cache it for fast access. Data are stored in indexed maps that provide an instantaneous access to the data needed by one specific DIF. Each separate DIFManager instantiates one instance of DIFDBManager and uses it as a configuration cache.

**Data collection**
In order to keep the coherence of collected data, DIF readout output are synchronized using the Absolute BCID. Data from different DIFs may be read out at different times but will have the same Absolute BCID for a given Trigger (or Ramfull) signal.

The logical way to keep synchronicity is to store in a BCID indexed map the buffers of all read DIF but this requires to manage memory allocation, access and cleaning. Recent Linux kernels offer the ability to use shared memory based file, ie, /dev/shm. This special file system is directly mapped in the system memory and data can be written, listed and read with extremely fast access.
Each DIF data block is written as a single file Event_BCID_DIFID in /dev/shm and an empty file / dev/shm/closed/Event_BCID_DIFID is created once the event file is closed. Standard c functions are used to list events available, to read, write and delete data. This method allows to separate the process reading data from those making data collection, writing, debugging and monitoring in a single computer without special protocol or API to be used.

# Low level software architecture



# DAta Acquision Software

## Low level hardware acces

**USB readout**
DIF and SDCC FPGA are interfaced with the same USB chip (FTDI). The DIF is thus uniquely identified by its FTDI device identifier (id) stored in an EEPROM and access to a specific device id is done using either the proprietary library FTD2XX or the free version library libFTDI. A base class called UsbDeviceDriver implements a set of low level access: bytes read or write , DIF and SDCC registers access and finally predefined commands.

**DIF and SDCC readout**
An upper layer software for DIF configuration and readout. The class called BasicUsbDataHandler aggregates a pointer to a UsbDeviceDriver and to a configuration buffer handling all DIF and SC parameters. Specific methods are used to implement the DIF configuration, the ASIC configurations and a single event readout to an internal buffer. Similarly, an SDCCDataHandler implements commands associated to the SDCC.

**DIF Manager**
Eventually one DIF manager class per PC is responsible of the data taking of the DIF through the USB connected to it. It handles the DIF and ASIC configuration parameters via an interface called DIFManager. This scans and detects all connected DIF, instantiates one DIFDataHandler class per detected device and distributes configuration parameters. When data taking starts, it starts a polling thread continuously reading events on all connected DIF. Events can be directly dumped to disk in LCIO format or transfer to the central data acquisition. Figure 21 summarizes this architecture.

## Configuration database

Gives the possibility to store and retrieve all parameters needed by the DAQ system. It is hosted on an Oracle server at CC IN2P3.

Homemade C++ library to interface this SQL database with the DAQ software and to allow users to insert and query data without knowledge of SQL.

Part of the DAQ system being written in Python, we used Swig to generate a Python version of the C++ library. The system has been designed to easily allow the addition or modification of existing object parameters. It is built on 2 levels: the database itself and the C++ library.

**Database model** :
The database model is conceived to deal with extendable number of ASIC. It can also handle different kinds of ASIC using different settings of parameters to take into account addition of other sub-detectors. In this model, each ASIC has a unique entry in the ASIC table, containing its type. The actual configuration parameters are contained in 2 tables : the first is ASIC_CONFIG for the parameters common to all ASIC types and the second <ASIC_TYPE>_CONFIG for the parameters specific to a given type. For instance the configuration parameters for a HARDROC ASIC will be stored in the ASIC_CONFIG table for the common part and in the HR2_CONFIG table for the HARDROC specific parameters4. As a consequence, supporting a new type of ASIC requires only to create a new table containing the specific configuration parameters. When downloading the parameters, the system will automatically choose the accurate tables associated to the type of the concerned ASIC.

## Data collection & monitoring

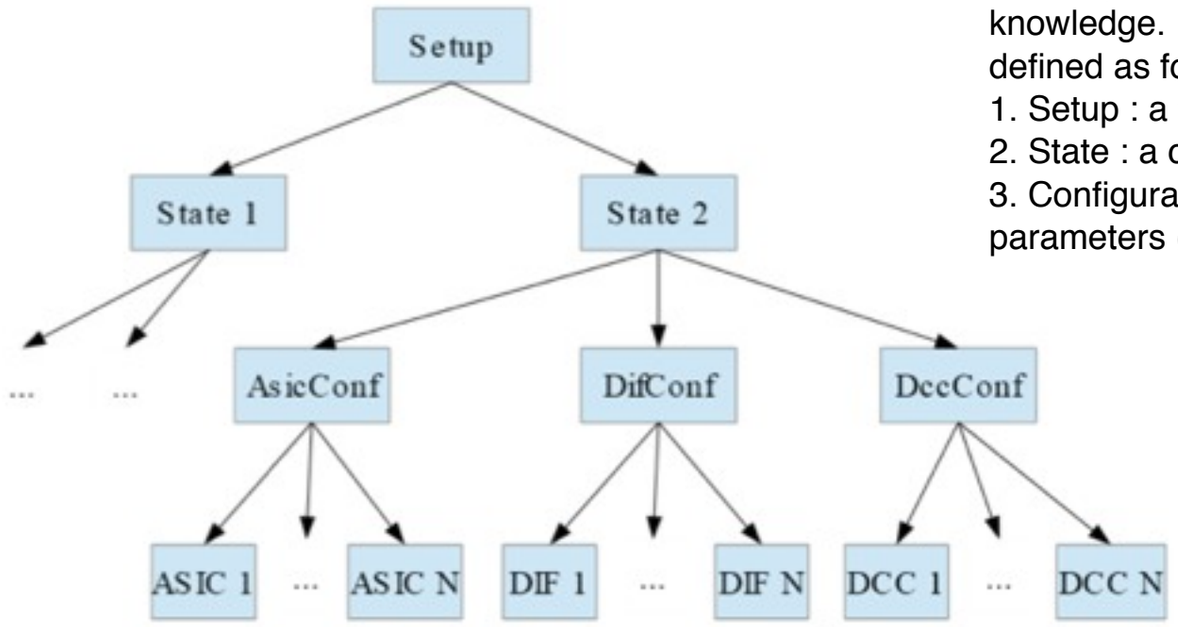
**Data acquisition access**

In order to minimize database access during data acquisition, a class called the DIFDBManager is responsible of the download of all the DIF and ASIC configuration parameters of a given setup and to cache it for fast access. Data are stored in indexed maps that provide an instantaneous access to the data needed by one specific DIF. Each separate DIFManager instantiates one instance of DIFDBManager and uses it as a configuration cache.

**Data collection**
In order to keep the coherence of collected data, DIF readout output are synchronized using the Absolute BCID. Data from different DIFs may be read out at different times but will have the same Absolute BCID for a given Trigger (or Ramfull) signal.

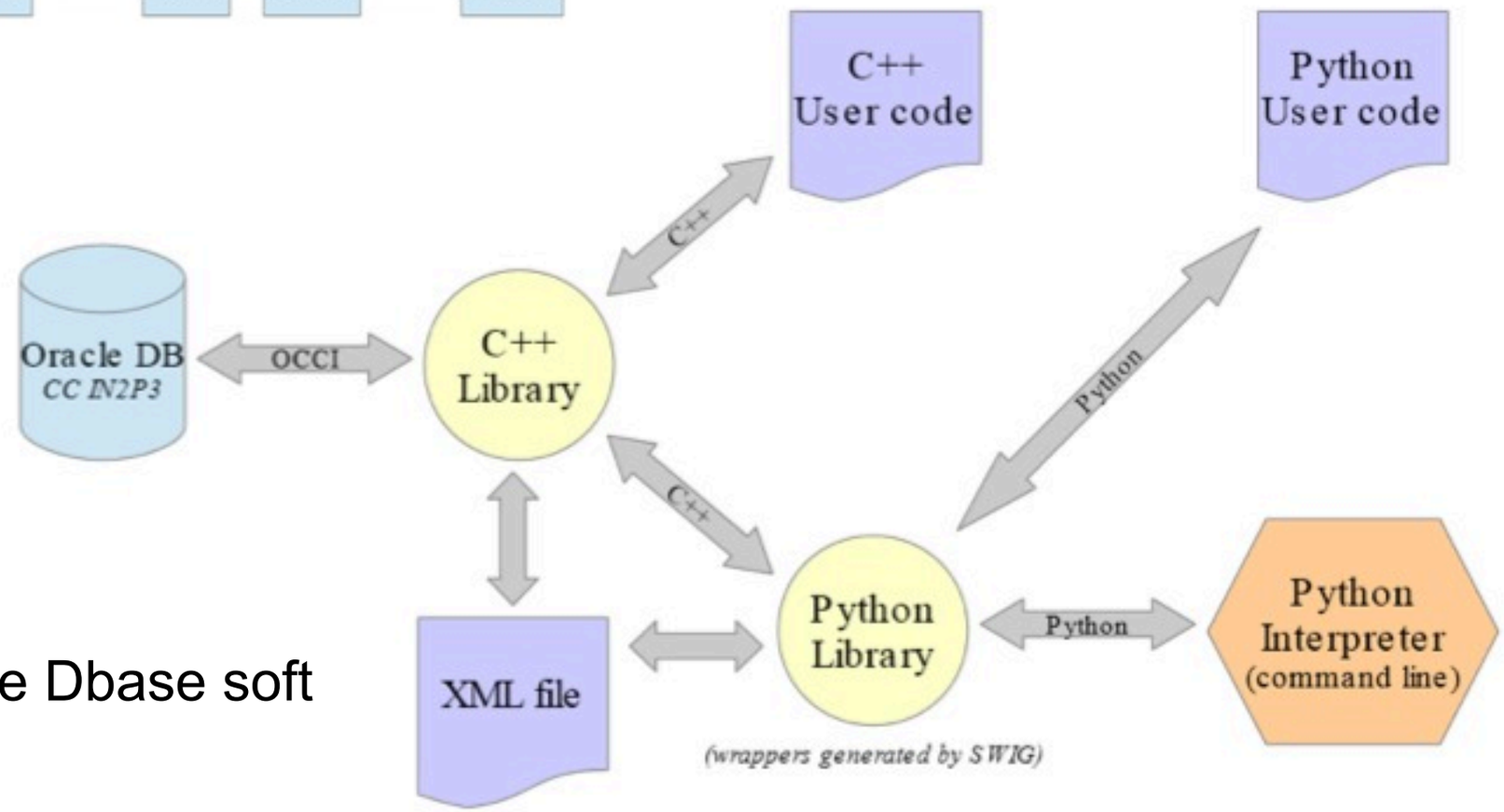
The logical way to keep synchronicity is to store in a BCID indexed map the buffers of all read DIF but this requires to manage memory allocation, access and cleaning. Recent Linux kernels offer the ability to use shared memory based file, ie, /dev/shm. This special file system is directly mapped in the system memory and data can be written, listed and read with extremely fast access.
Each DIF data block is written as a single file Event_BCID_DIFID in /dev/shm and an empty file /dev/shm/closed/Event_BCID_DIFID is created once the event file is closed. Standard c functions are used to list events available, to read, write and delete data. This method allows to separate the process reading data from those making data collection, writing, debugging and monitoring in a single computer without special protocol or API to be used.

## Database Setup Structure




The C++ library gives a high level access to the database, providing the possibility to manipu- late the concepts of ASIC, DIF or Configuration without any SQL knowledge. It allows to create/modify/upload or download a complete setup defined as follows:

1. Setup : a set of "states" (one per sub-detector)
2. State : a coherent set of "configurations"
3. Configuration : a list of basic objects Basic object : a unique object with all its parameters (ASIC, DIF, DCC, ...) .



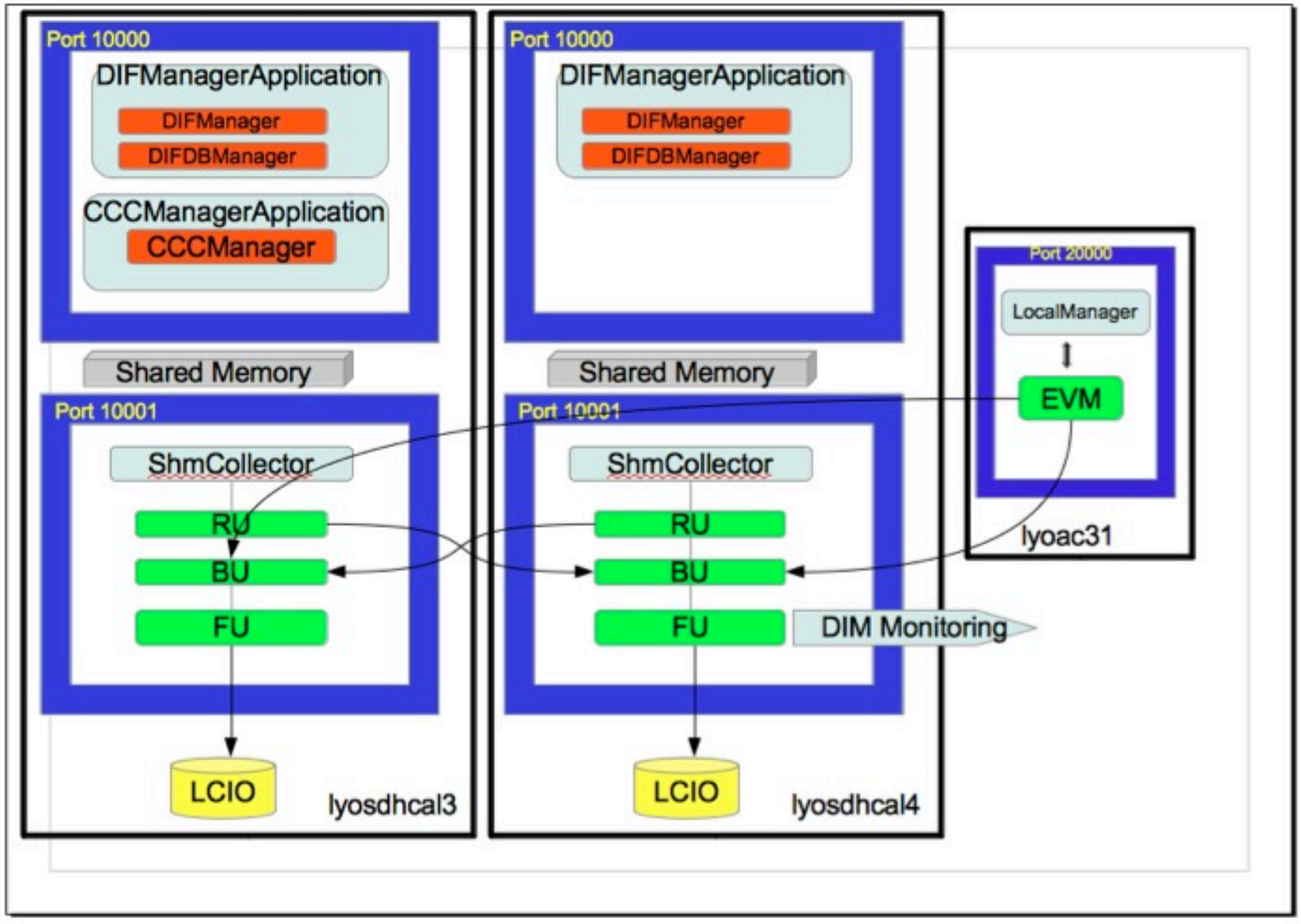

Scheme of the Dbase soft

# Data Collection Structure

Global data acquisition
Whenever several computers are involved in the data taking, a communication framework is needed -> CMS data acquisition XDAQ framework.
It provides:
• A communication with both binary and XML messages.
• An XML description of the computer and software architecture.
• A web-server implementation of all data acquisition application.
• A scalable Event builder.

Each PC handling DIFs holds:
• DIF manager XDAQ application obeying to a message driven state machine responsible for initialization ( USB scan and DB download), configuration (DIF and chips settings) and running (DIF readout and /dev/shm storage).
• A ShmSupervisor that scans the shared memory and pushes completed events to the local event builder application (Readout Unit).

The main advantage of the CMS event builder is the scalability. It is possible to add any number of collecting applications (Builder Unit) that will merge data from all Readout Units for a given trigger. Those Builder Units distribute merged events to any analysis and data writing application (Filter Unit) declared to them. In the SDHCAL case, the computing capability is handled by the PC's reading the DIF so one Readout Unit, one Builder Unit and one Filter Unit application are created on each PC as described on Figure 6.3.2 and each Filter Unit writes the associated data in a separate stream.



# DAta Acquision Software

## Low level hardware acces

**USB readout**
DIF and SDCC FPGA are interfaced with the same USB chip (FTDI). The DIF is thus uniquely identified by its FTDI device identifier (id) stored in an EEPROM and access to a specific device id is done using either the proprietary library FTD2XX or the free version library libFTDI. A base class called UsbDeviceDriver implements a set of low level access: bytes read or write , DIF and SDCC registers access and finally predefined commands.

**DIF and SDCC readout**
An upper layer software for DIF configuration and readout. The class called BasicUsbDataHandler aggregates a pointer to a UsbDeviceDriver and to a configuration buffer handling all DIF and SC parameters. Specific methods are used to implement the DIF configuration, the ASIC configurations and a single event readout to an internal buffer. Similarly, an SDCCDataHandler implements commands associated to the SDCC.

**DIF Manager**
Eventually one DIF manager class per PC is responsible of the data taking of the DIF through the USB connected to it. It handles the DIF and ASIC configuration parameters via an interface called DIFManager. This scans and detects all connected DIF, instantiates one DIFDataHandler class per detected device and distributes configuration parameters. When data taking starts, it starts a polling thread continuously reading events on all connected DIF. Events can be directly dumped to disk in LCIO format or transfer to the central data acquisition. Figure 21 summarizes this architecture.

## Configuration database

Gives the possibility to store and retrieve all parameters needed by the DAQ system. It is hosted on an Oracle server at CC IN2P3.

Homemade C++ library to interface this SQL database with the DAQ software and to allow users to insert and query data without knowledge of SQL.

Part of the DAQ system being written in Python, we used Swig to generate a Python version of the C++ library. The system has been designed to easily allow the addition or modification of existing object parameters. It is built on 2 levels: the database itself and the C++ library.

**Database model** :
The database model is conceived to deal with extendable number of ASIC. It can also handle different kinds of ASIC using different settings of parameters to take into account addition of other sub-detectors. In this model, each ASIC has a unique entry in the ASIC table, containing its type. The actual configuration parameters are contained in 2 tables : the first is ASIC_CONFIG for the parameters common to all ASIC types and the second <ASIC_TYPE>_CONFIG for the parameters specific to a given type. For instance the configuration parameters for a HARDROC ASIC will be stored in the ASIC_CONFIG table for the common part and in the HR2_CONFIG table for the HARDROC specific parameters4. As a consequence, supporting a new type of ASIC requires only to create a new table containing the specific configuration parameters. When downloading the parameters, the system will automatically choose the accurate tables associated to the type of the concerned ASIC.

## Data collection & monitoring

**Data acquisition access**

In order to minimize database access during data acquisition, a class called the DIFDBManager is responsible of the download of all the DIF and ASIC configuration parameters of a given setup and to cache it for fast access. Data are stored in indexed maps that provide an instantaneous access to the data needed by one specific DIF. Each separate DIFManager instantiates one instance of DIFDBManager and uses it as a configuration cache.

**Data collection**
In order to keep the coherence of collected data, DIF readout output are synchronized using the Absolute BCID. Data from different DIFs may be read out at different times but will have the same Absolute BCID for a given Trigger (or Ramfull) signal.

The logical way to keep synchronicity is to store in a BCID indexed map the buffers of all read DIF but this requires to manage memory allocation, access and cleaning. Recent Linux kernels offer the ability to use shared memory based file, ie, /dev/shm. This special file system is directly mapped in the system memory and data can be written, listed and read with extremely fast access.
Each DIF data block is written as a single file Event_BCID_DIFID in /dev/shm and an empty file / dev/shm/closed/Event_BCID_DIFID is created once the event file is closed. Standard c functions are used to list events available, to read, write and delete data. This method allows to separate the process reading data from those making data collection, writing, debugging and monitoring in a single computer without special protocol or API to be used.