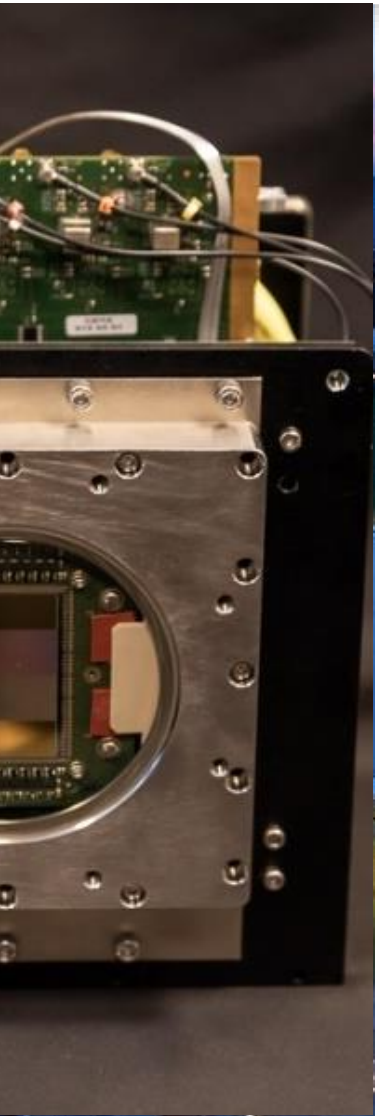

ESRF
The European Synchrotron

ESRF | The European Synchrotron



# Development of an advanced data acquisition framework for 2D X-Ray Detectors

**27/10/2020**

**Aurélien BIDEAUD** and **Samuel DEBIONNE**

on behalf of many ESRF staff

EIROforum serving european science

Big Data

# PRESENTATION OUTLINE

# HIGH PERFORMANCE DETECTORS

**Increase X-Ray source brilliance**

**High frame rate: 1 - 10 KHz**

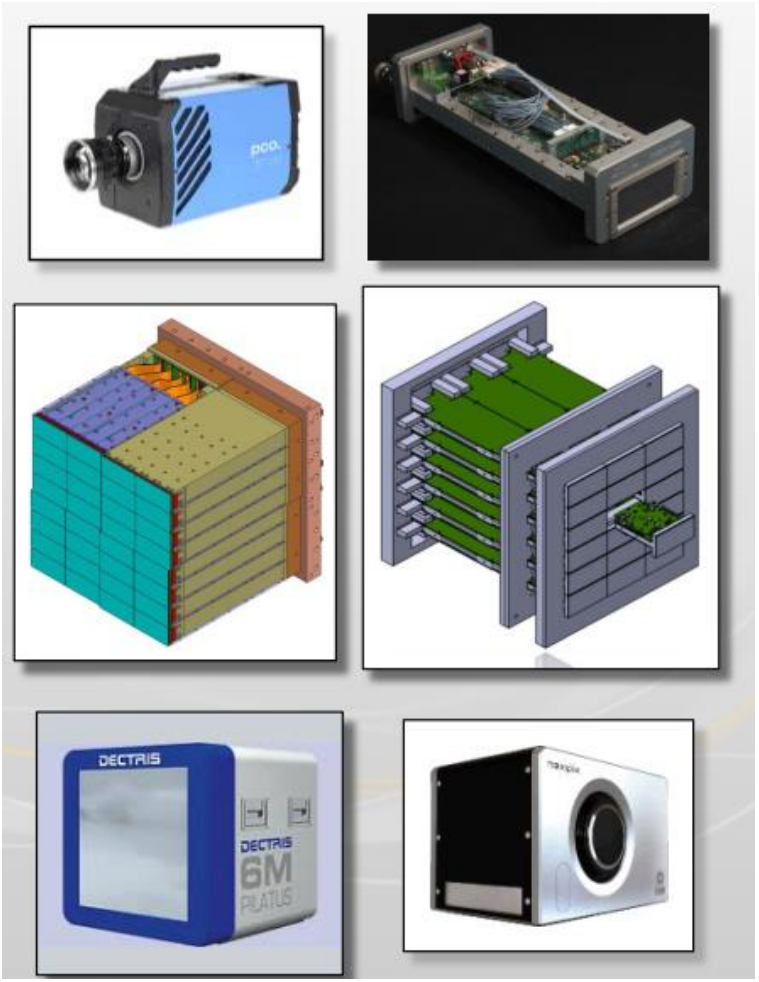
**Large detectors**

**Pixel detectors**

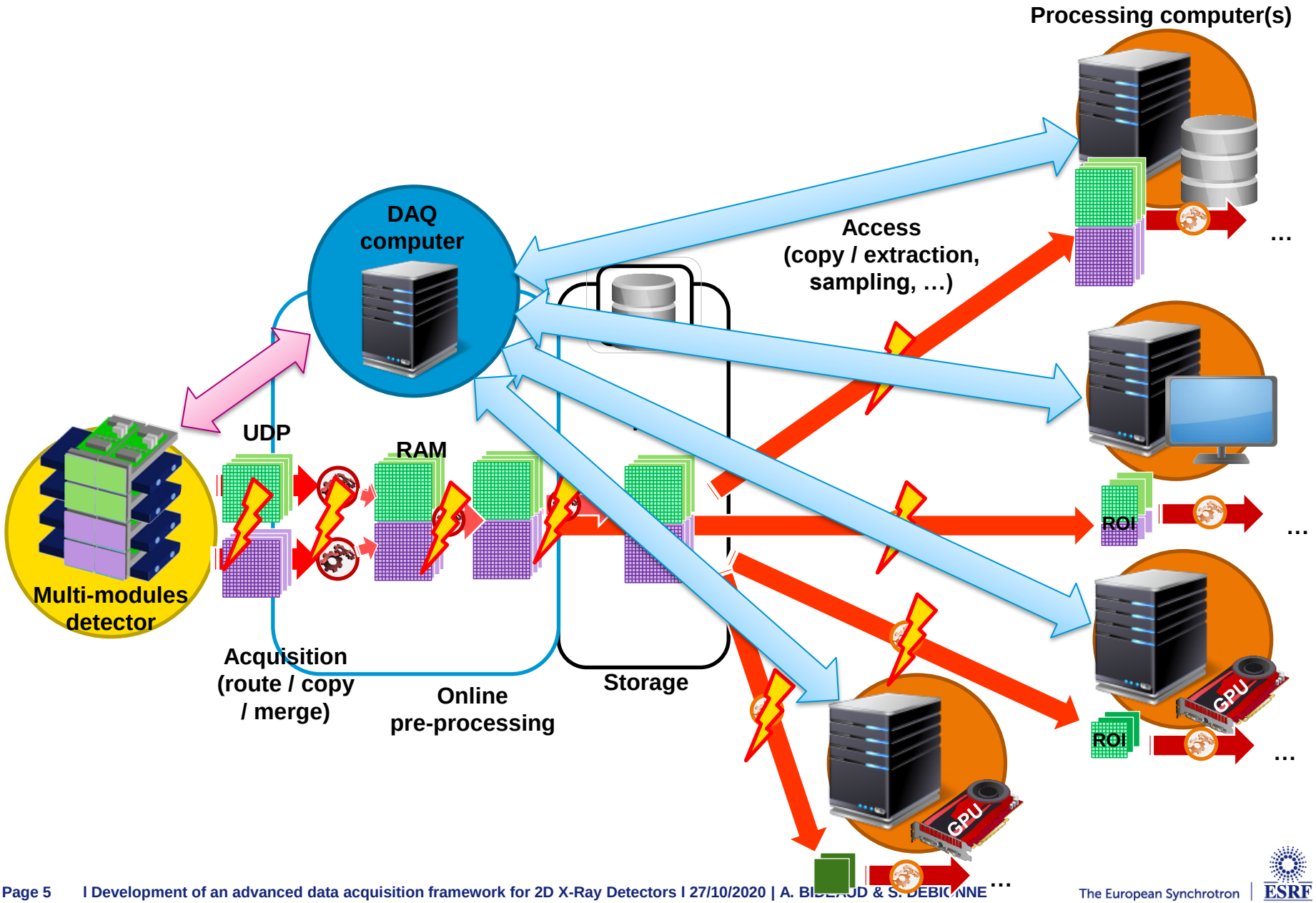
- Single chips: **100 MB/s**
- Basic modules: **0.5 – 1 GB/s**
- Multi-modules: **10+ GB/s**

**Data transfer can limit sensor speed**

**Proprietary solutions**

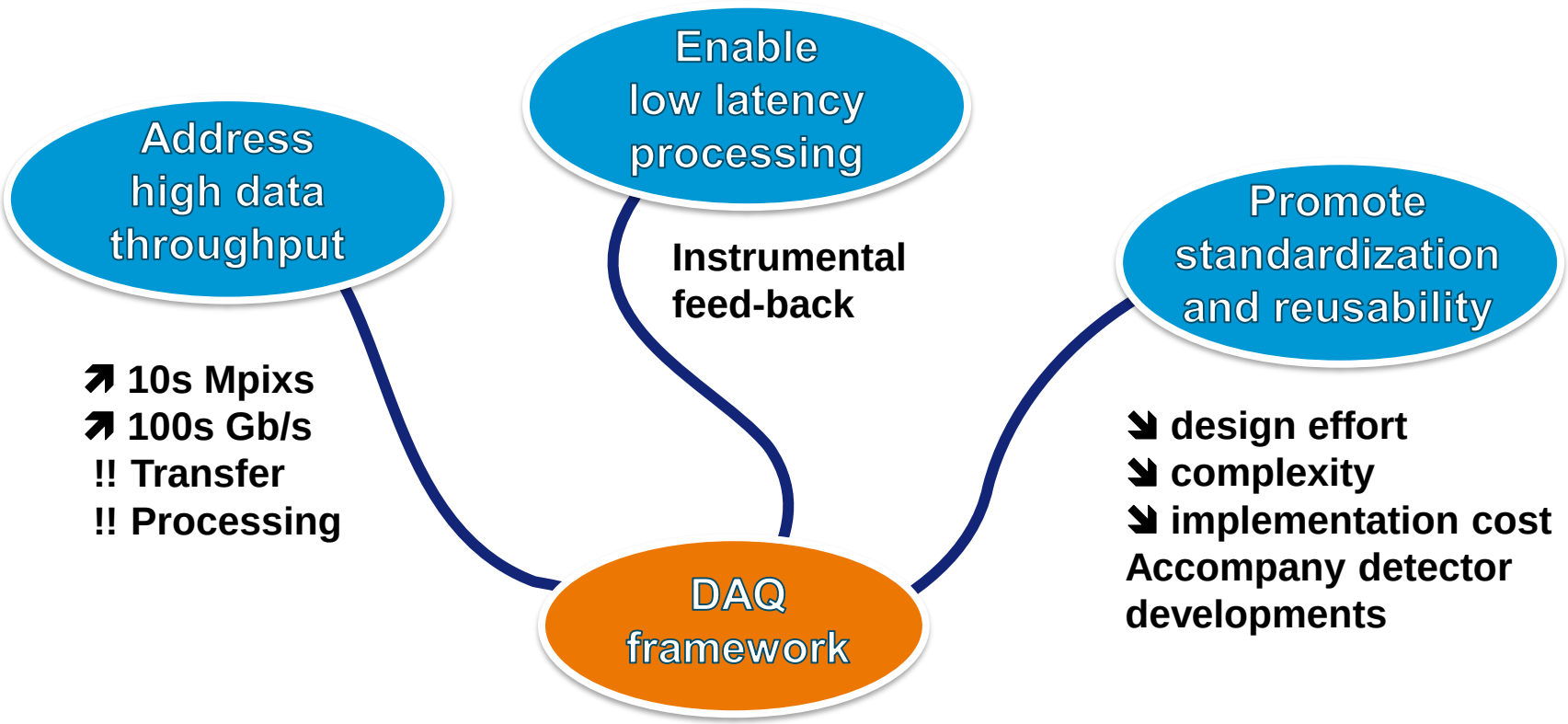


# ARCHITECTURE OF THE CURRENT DAQ FRAMEWORK @ ESRF



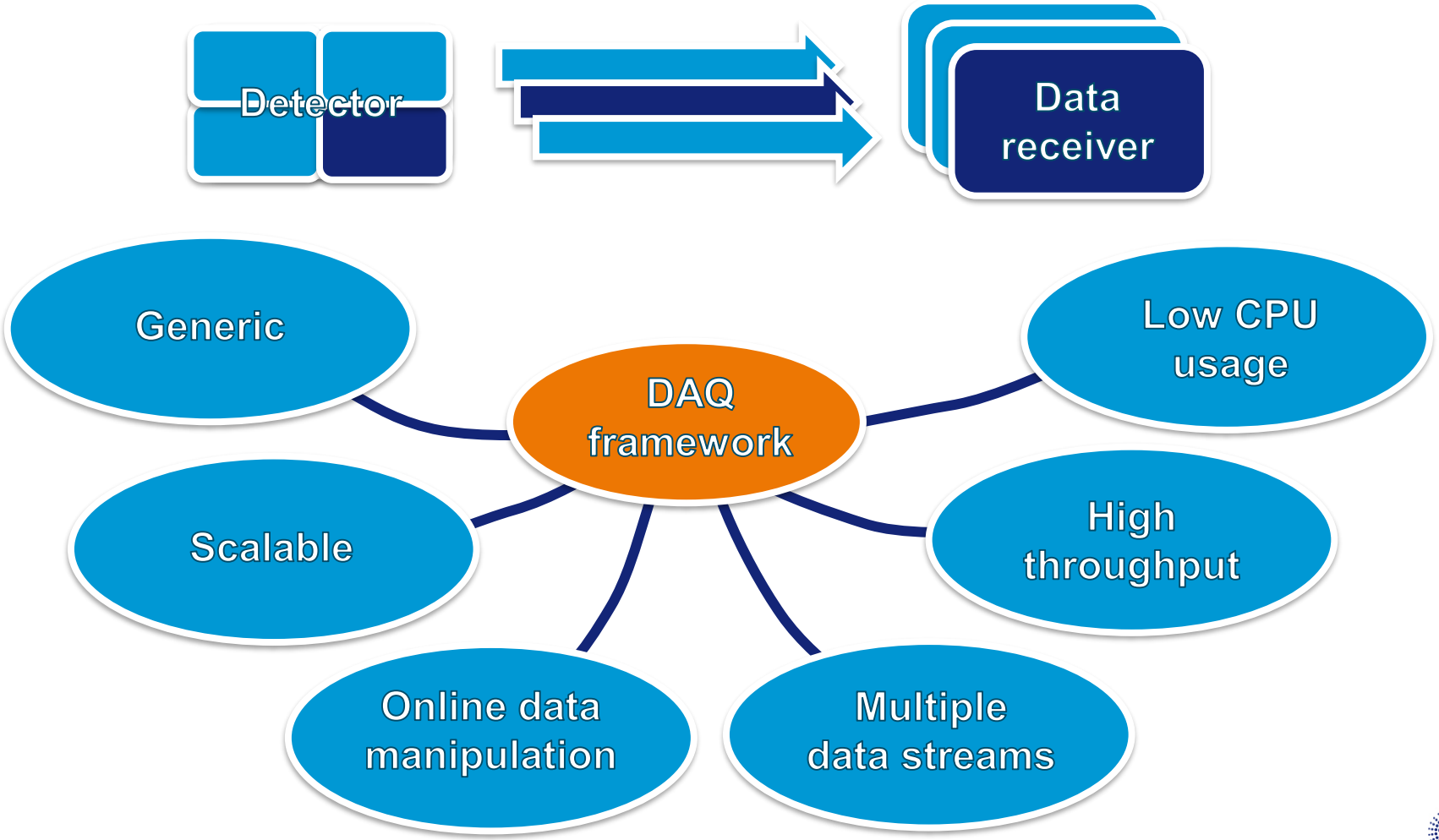
# PRESENTATION OUTLINE

# RASHPA INTRODUCTION – MOTIVATIONS



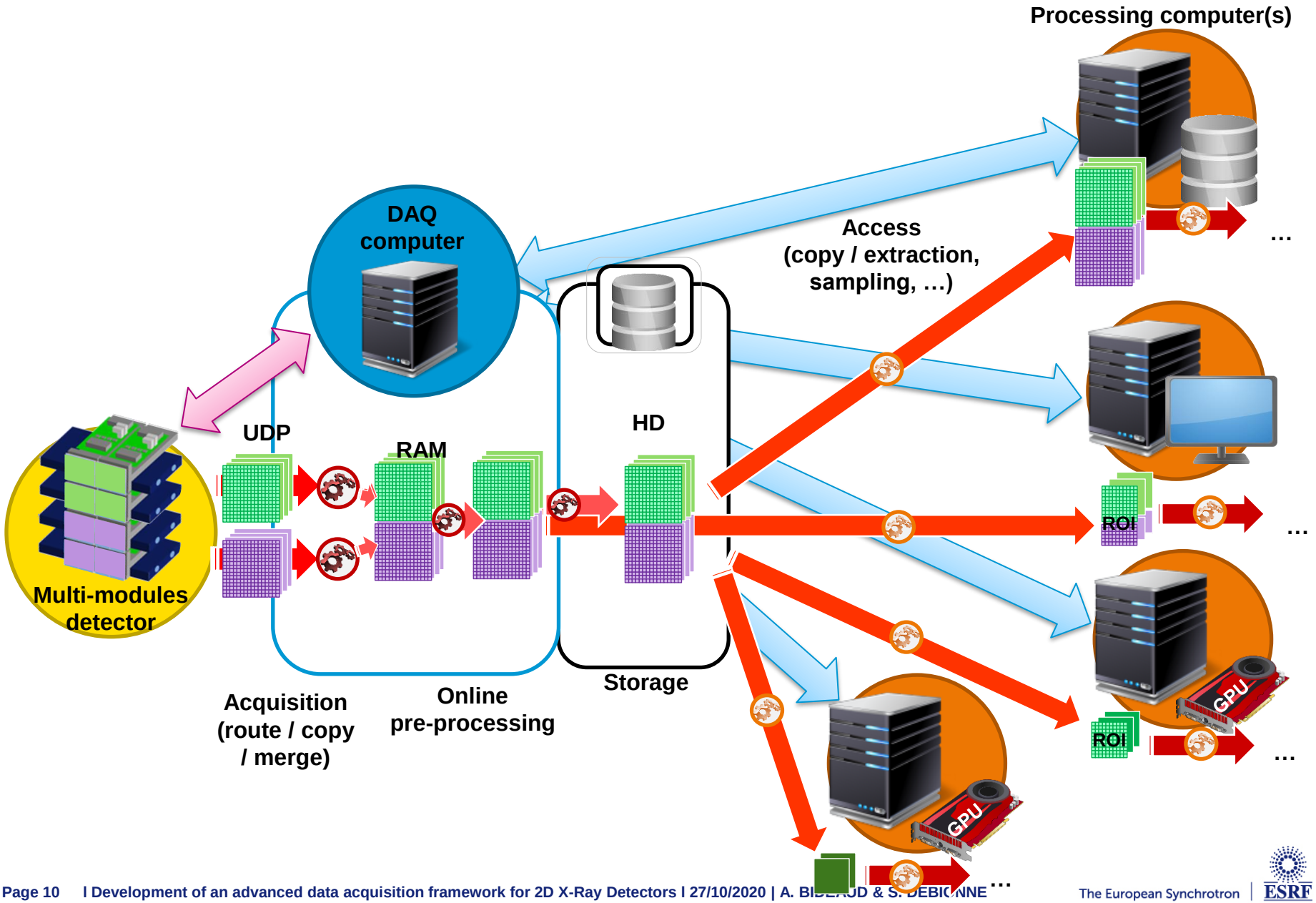
## RDMA-based Acquisition System for High Performance Applications

**Remote Direct Memory Access**

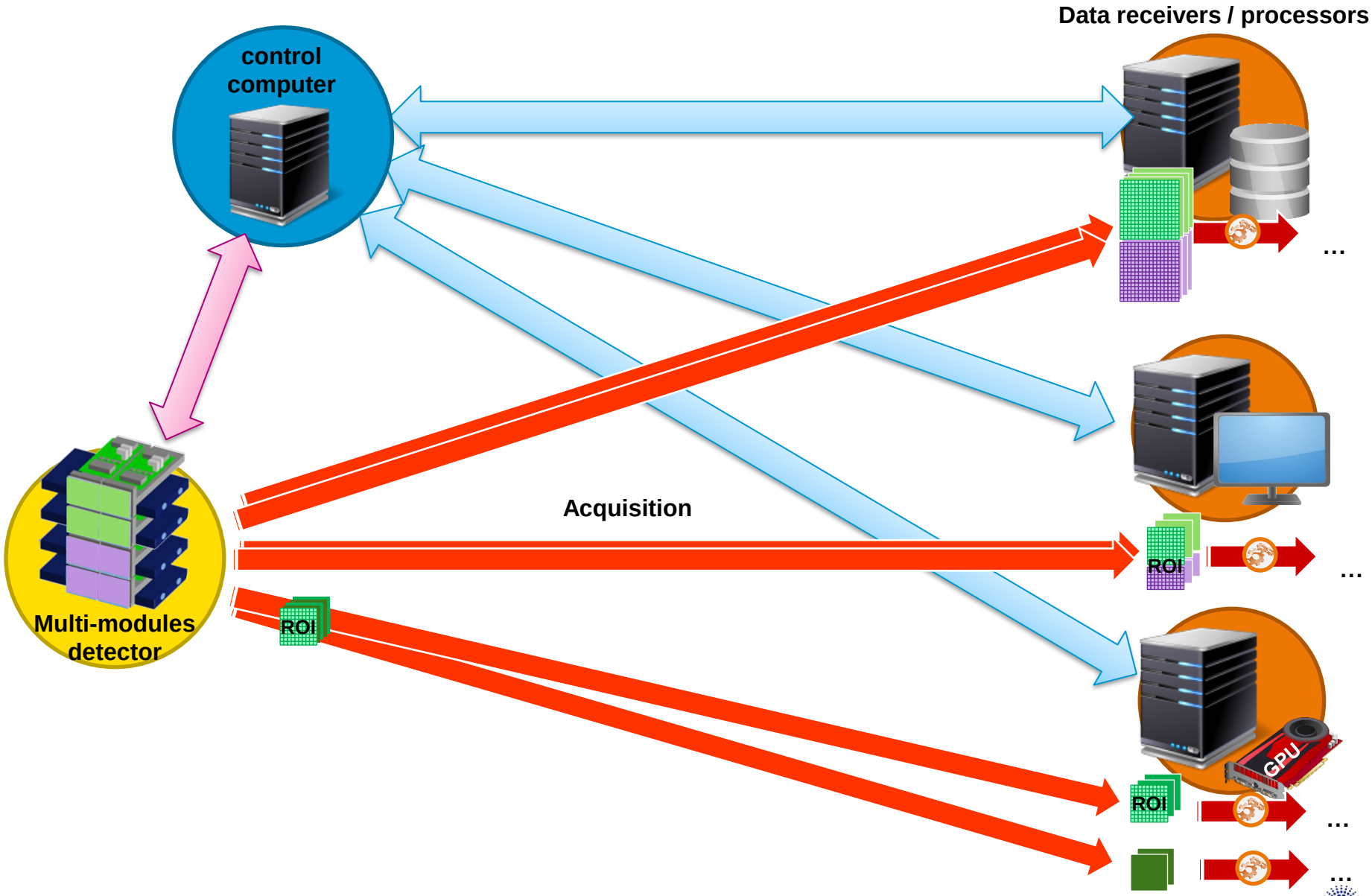


# PRESENTATION OUTLINE

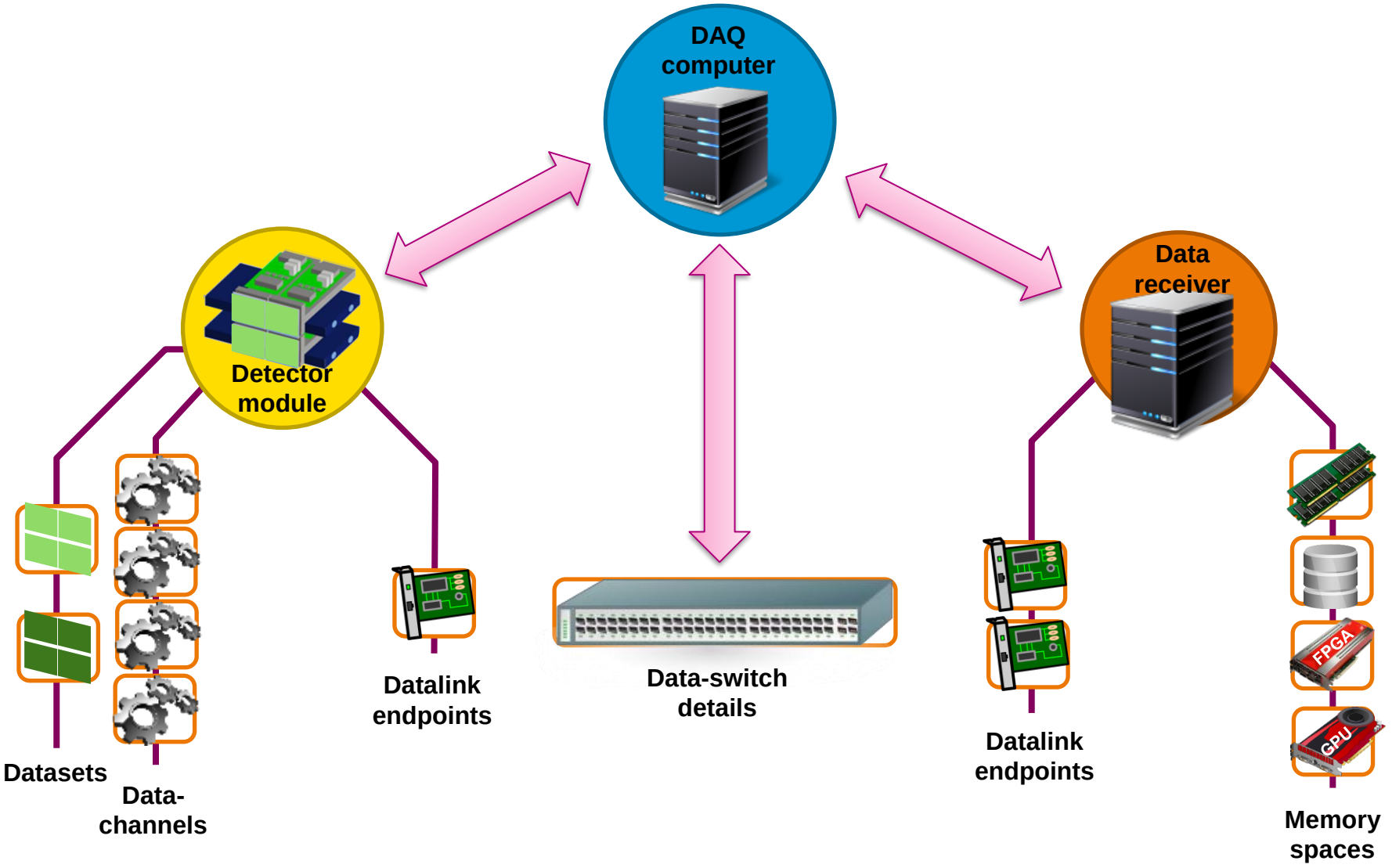
# RASHPA ARCHITECTURE – CURRENT DAQ FRAMEWORK @ ESRF



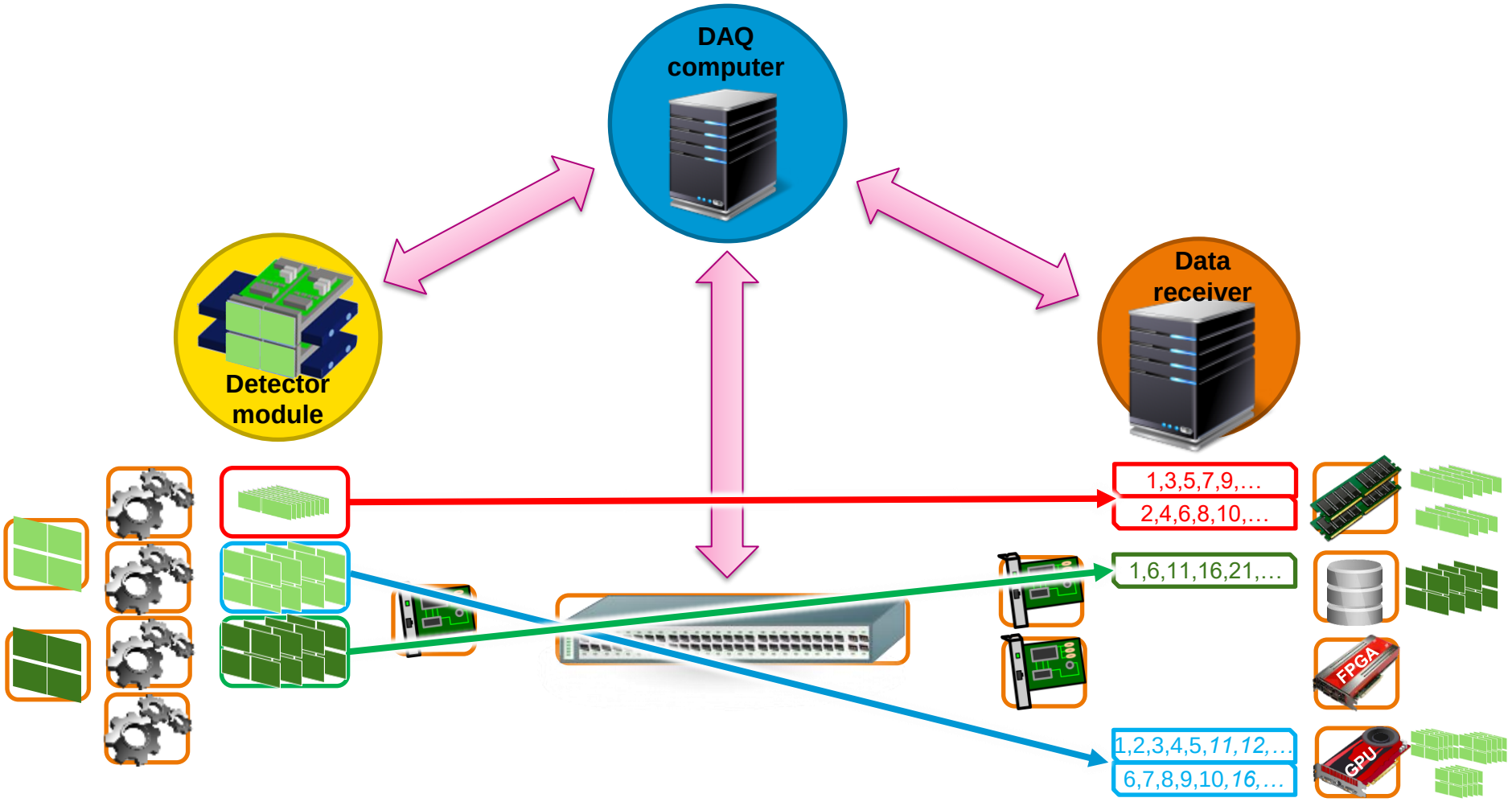
# RASHPA ARCHITECTURE – QUICK VIEW



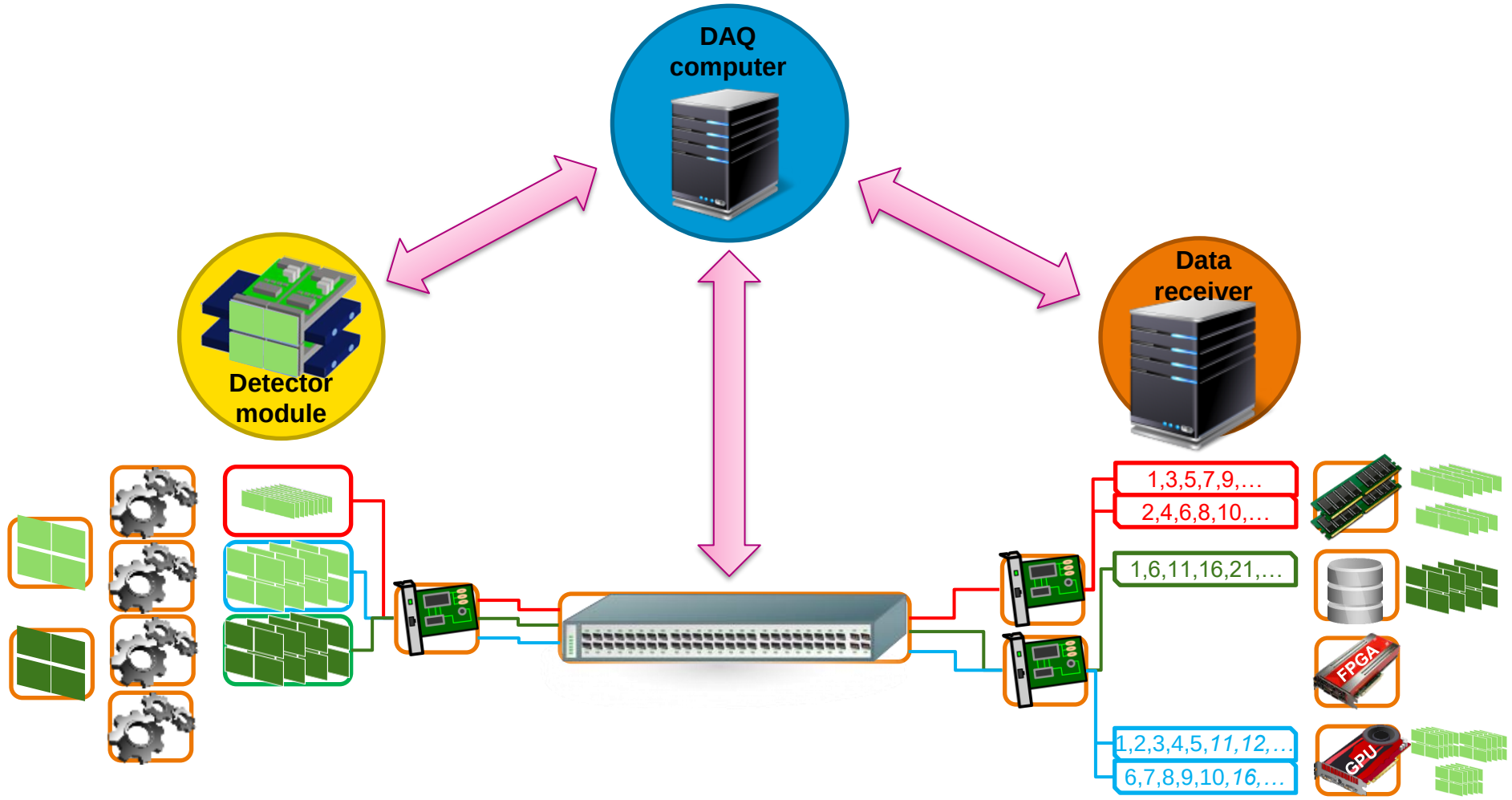
# RASHPA CONFIGURATION – CAPABILITIES RETRIEVAL



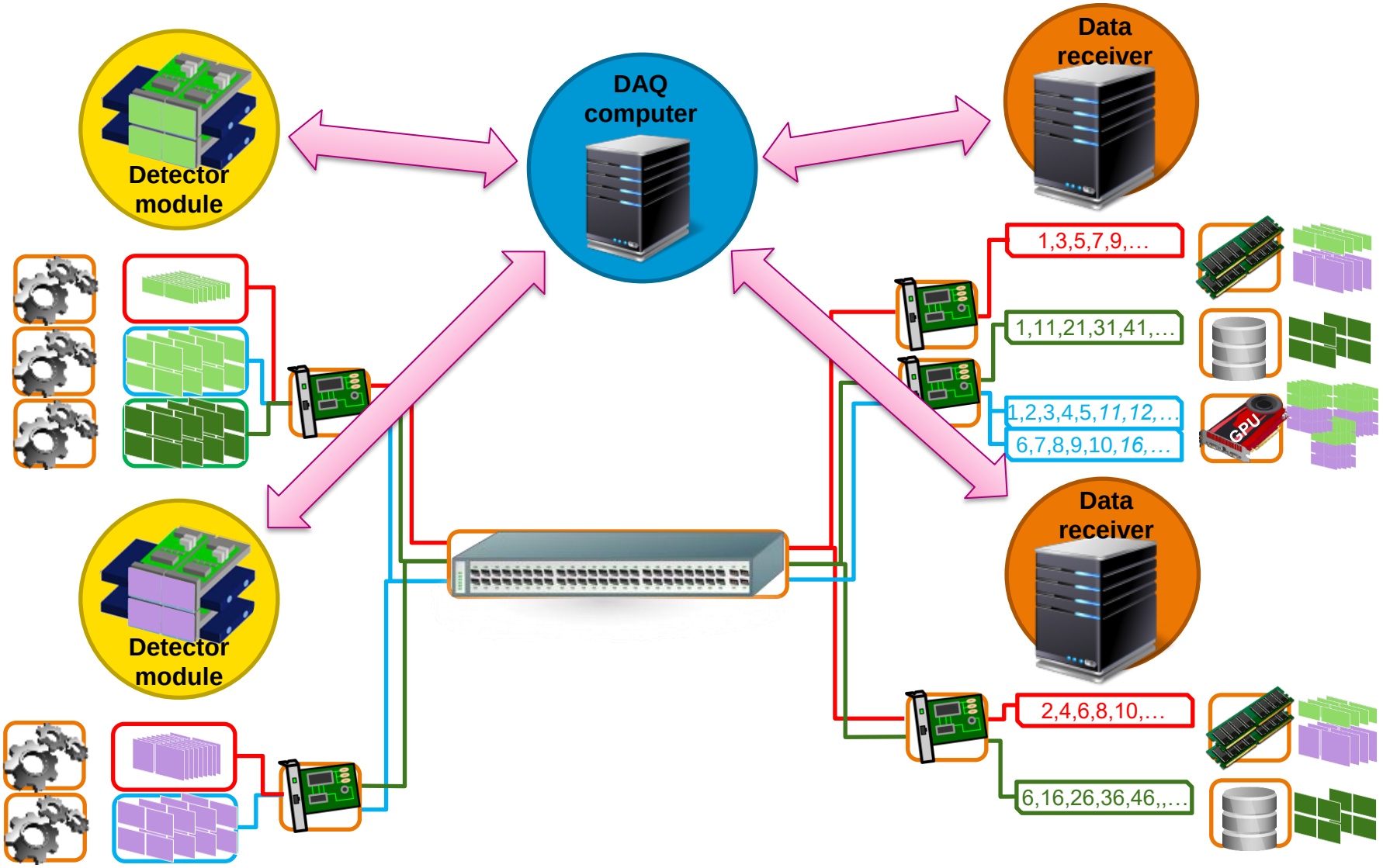
# RASHPA CONFIGURATION – DATA FLOWS DEFINTION



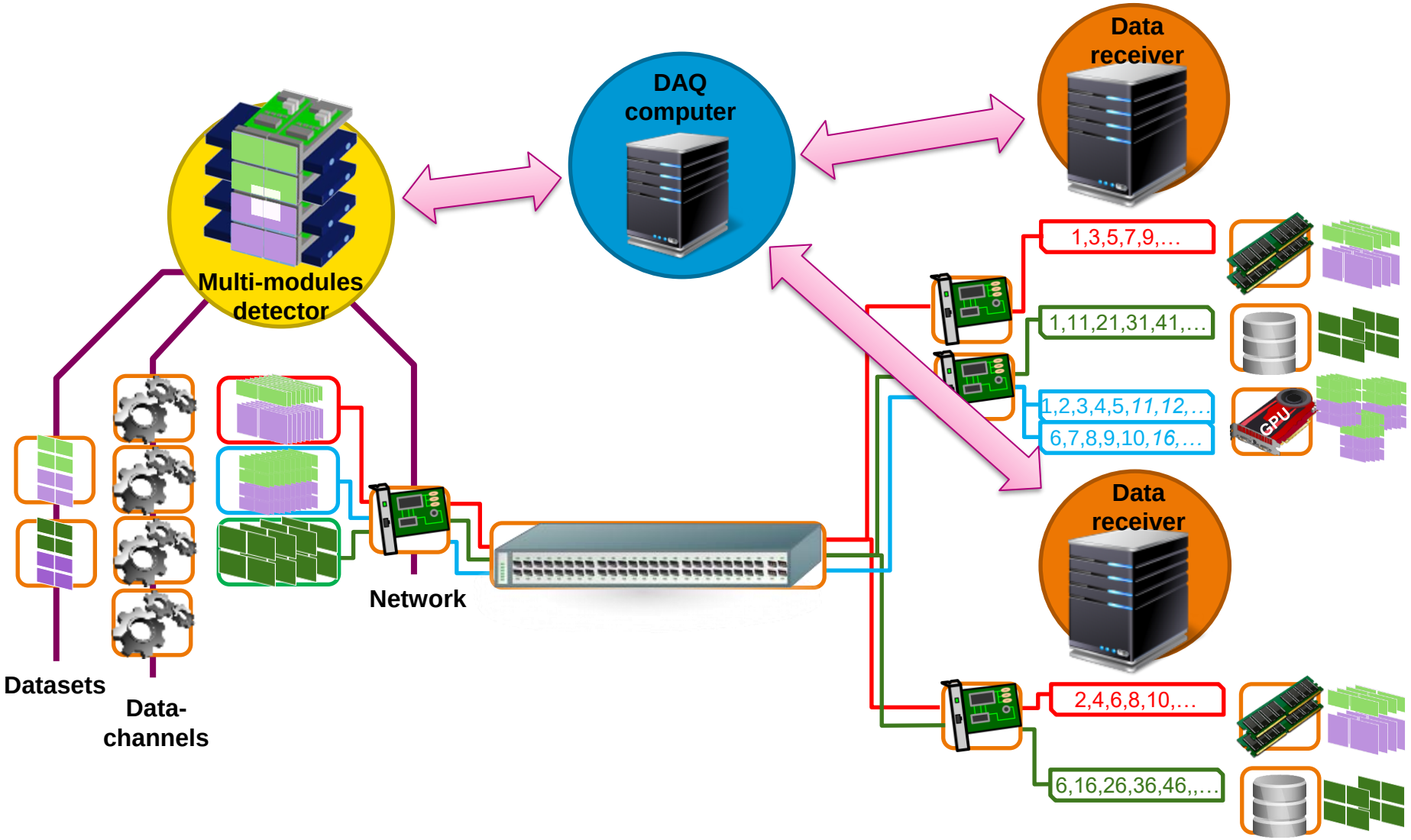
# RASHPA CONFIGURATION – DATA-FLOW ROUTING



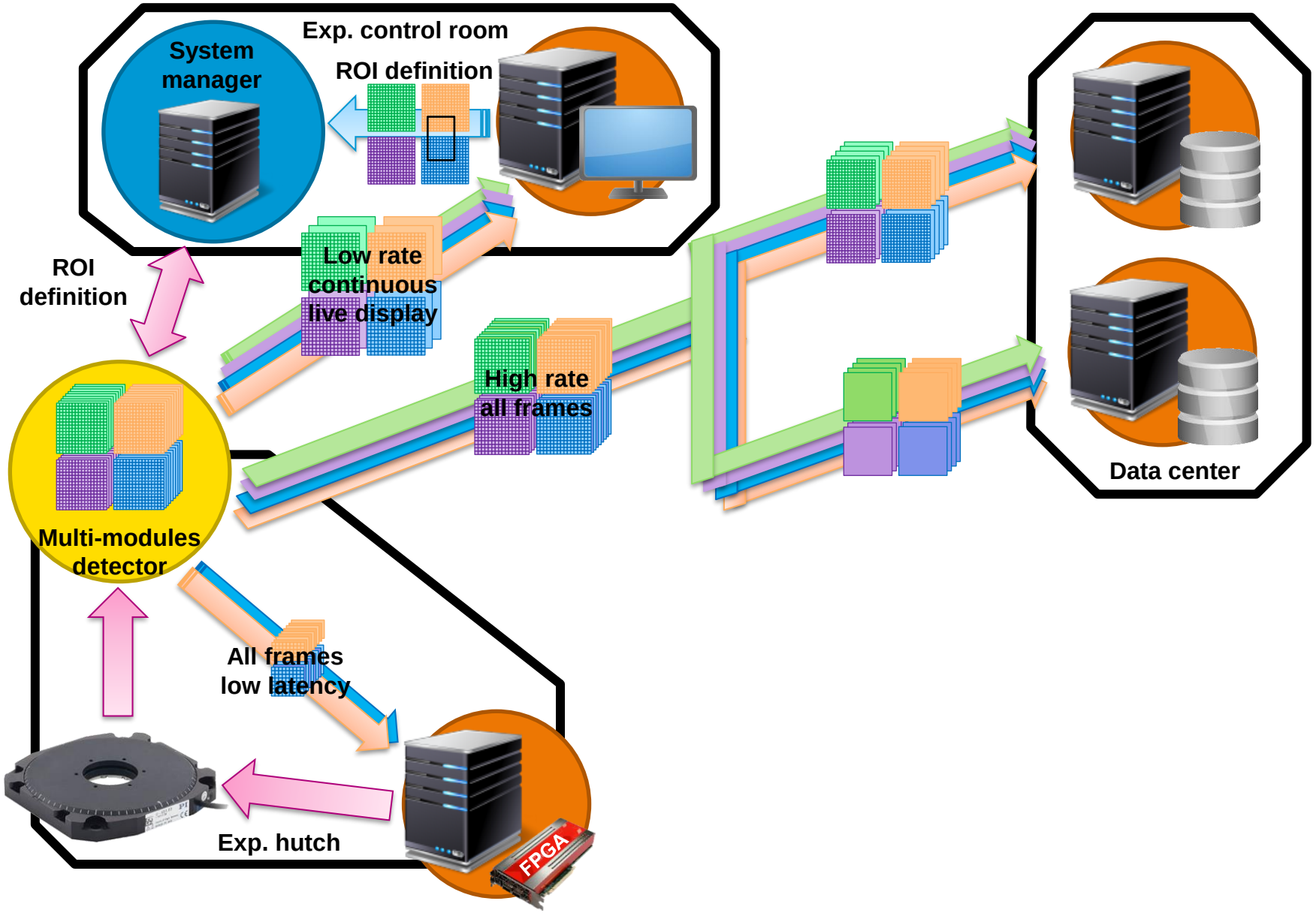
# RASHPA CONFIGURATION – MULTI-NODES



# RASHPA CONFIGURATION – MULTI-NODES



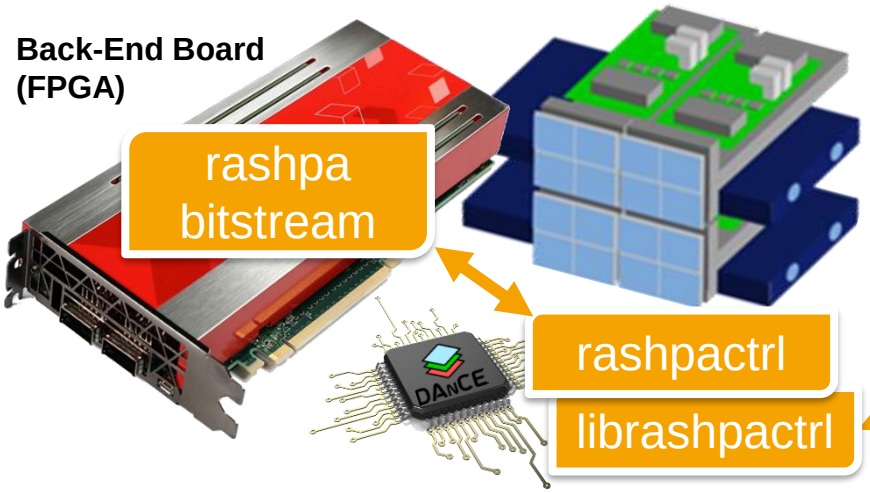
# RASHPA CONFIGURATION – EXAMPLE OF USE



# PRESENTATION OUTLINE

# RASHPA IMPLEMENTATION – A SPECIFIC SHOW CASE

## Detector: SMARTPIX




## Client application: LIMA2


## SMARTPIX RASHPA capabilities:

- 8 simultaneous data streams.
- PCIe data transfer.
- Gbethernet data transfer in progress.
- Extraction, reconstruction.

## Specific developments:

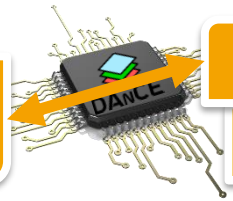
- generic plug-in for all RASHPA-compliant detectors

# RASHPA IMPLEMENTATION – A GENERIC CASE

## Emulated Detector

RASHPA EMUlator

REMU




emulated firmware

rashpactrl

librashpactrl

## Client application: LIMA2

Lima 2

librashpa

libcdma

libmetaresc

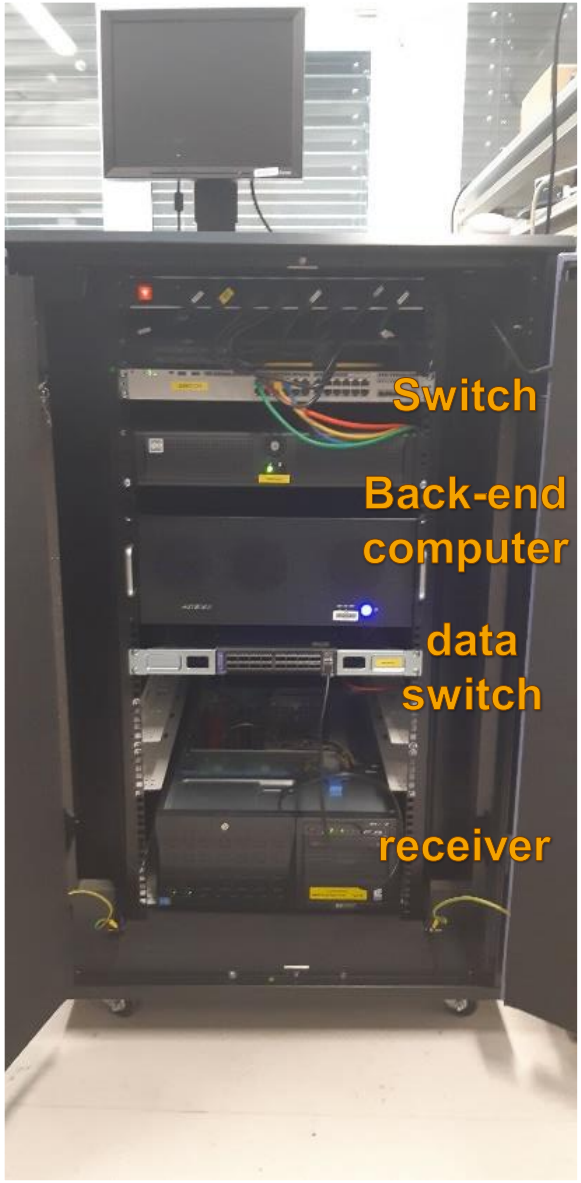
## RASHPA Capabilities:

- N simultaneous data streams.
- PCIe data transfer.
- Gbethernet data transfer in progress.
- Extraction, reconstruction and extra data manipulation possible.

## Specific developments:

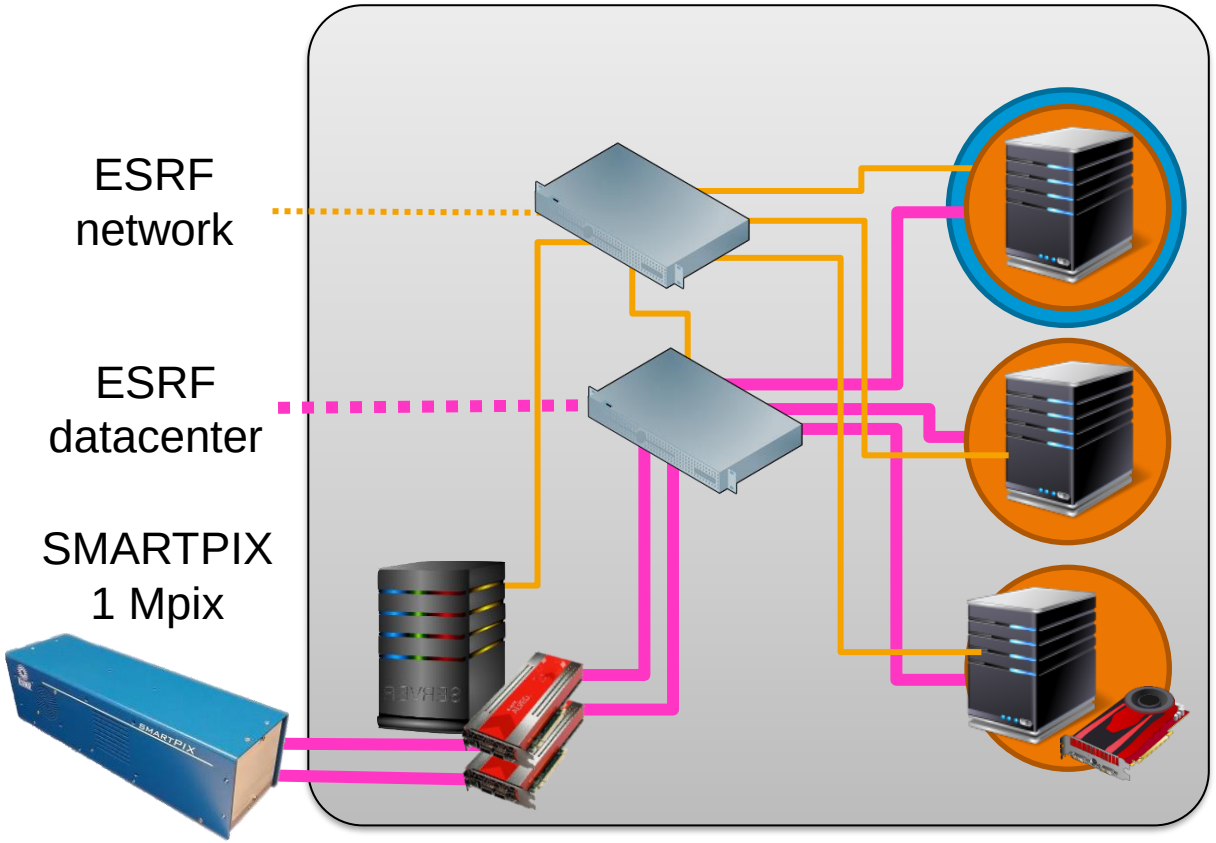
- generic plug-in for all RASHPA-compliant detectors
- REMU plug-in (for control)

# RASHPA IMPLEMENTATION – HARDWARE VIEW



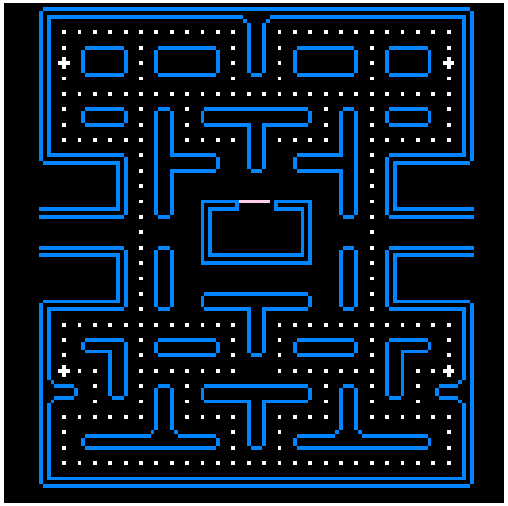

## Goal:

Demonstrate RASHPA associated to the SMARTPIX 2Gpix-detector on a beamline @ ESRF.



# PRESENTATION OUTLINE

# RASHPA EXAMPLE – CONSIDERED CASE (1/6)



Scene to image

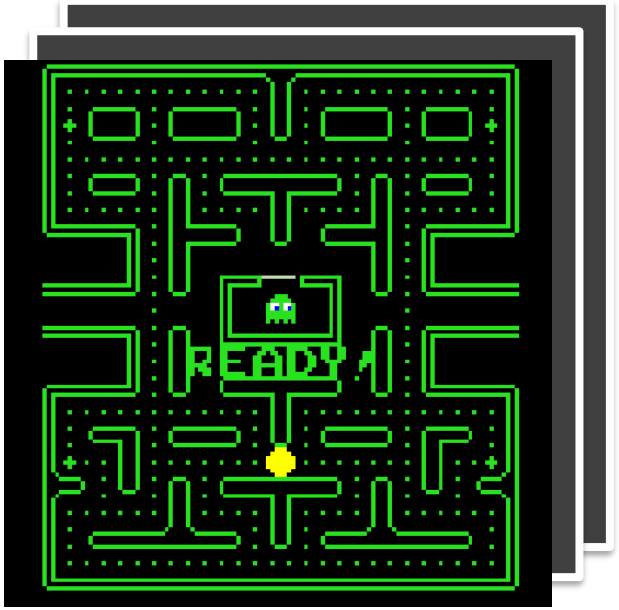
4-modules detector

2 data-sets

10 data-channels

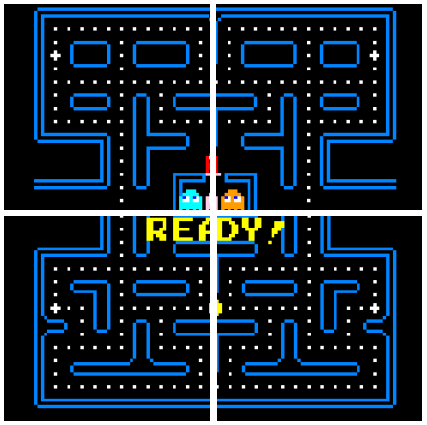
3 data receivers

# RASHPA EXAMPLE – DATA-SETS (2/6)

# RASHPA EXAMPLE – DATA-SETS (2/6)

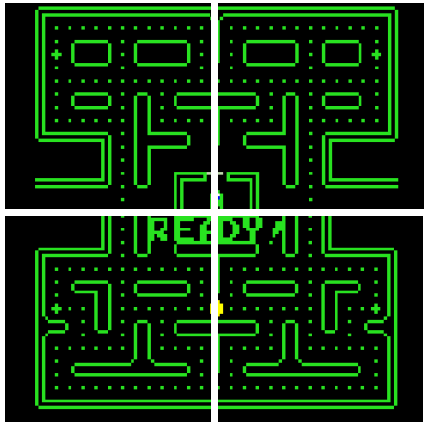





Module size : 240x240

Total size : 500 x 500

2 datasets : CTR1 & CTR2



# RASHPA EXAMPLE – DATA FLOW #1 (3/6)


Full frame

dataset #1

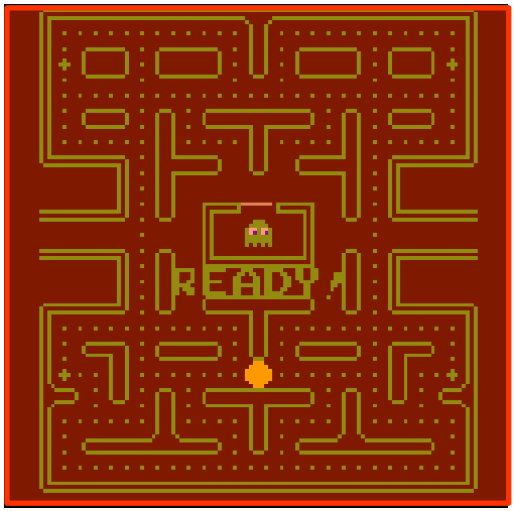
1 frame / 10

limited buffer size

```
rashpa_gdt_t *GDT = rashpaManagerCreateGDT(manager_pt, "FullDet1", "CTR1");  // GDT named "FullDet1", dataset "CTR1"
rashpaGDTSetSourceFullImage(GDT);                                            // full detector image
rashpaGDTSetAlignment(GDT, 256, 1, 0, 0);                                    // aligned on 256 bits, no header/footer
rashpaGDTSetSourceSlices(GDT, 0, 12, 10);                                    // starting at frame 0, 12 frames, 1 frame / 10
rashpaGDTSetRules(GDT, 1, 0, RASHPA_CONCATENATE | RASHPA_CIRCULAR);          // 1 block / group, starting at slot 0
rashpaGDTSetEvents(GDT, 1, RASHPA_EV_GRP);                                   // event every group of data block
```

```
ssize_t lbuff_bytes = 16*1024;                                               // 16 kiB per local buffer
rashpa_buffer_t* buffer = rashpaManagerGetGDTBuffer(manager_pt, GDT, lbuff_bytes);
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver0_pt, "dataMSpace"); // in receiver 0, in dataMSpace memspace
```

# RASHPA EXAMPLE – DATA FLOW #2 (4/6)




Full frame

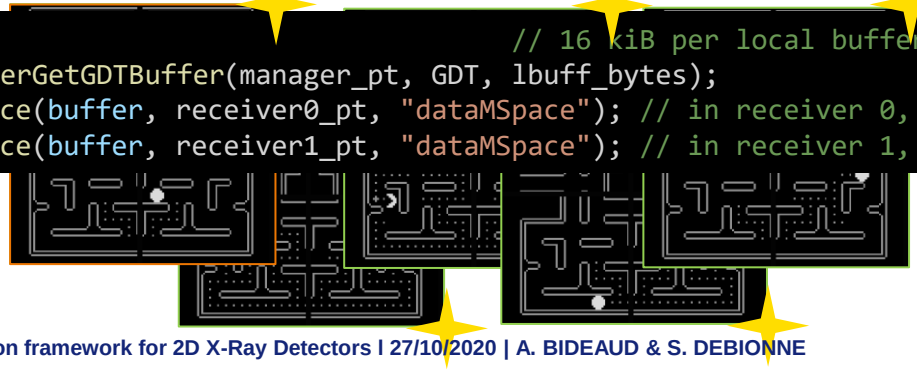
dataset #2

1 frame / 5

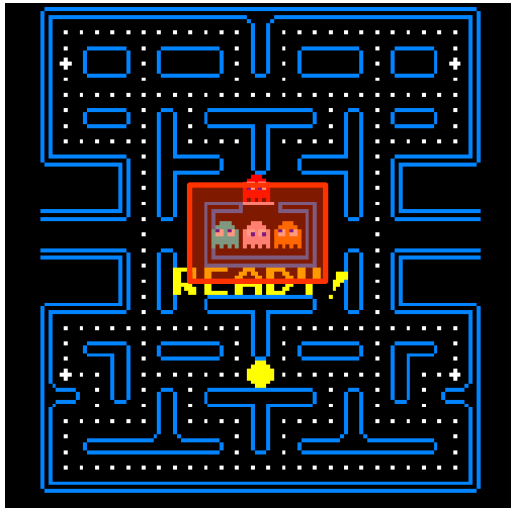
limited buffer size

```
rashpa_gdt_t *GDT = rashpaManagerCreateGDT(manager_pt, "FullDet2", "CTR2");  // GDT named "FullDet2", dataset "CTR2"
rashpaGDTSetSourceFullImage(GDT);                                            // full detector image
rashpaGDTSetAlignment(GDT, 256, 1, 0, 0);                                    // aligned on 256 bits, no header/footer
rashpaGDTSetSourceSlices(GDT, 0, 22, 5);                                     // starting at frame 0, 22 frames, 1 frame / 5
rashpaGDTSetRules(GDT, 1, 0, RASHPA_CONCATENATE | RASHPA_CIRCULAR);          // 1 block / group, starting at slot 0
rashpaGDTSetEvents(GDT, 1, RASHPA_EV_GRP);                                   // event every group of data blocks
```

```
ssize_t lbuff_bytes = 16*1024;                                               // 16 kiB per local buffer
rashpa_buffer_t* buffer = rashpaManagerGetGDTBuffer(manager_pt, GDT, lbuff_bytes);
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver0_pt, "dataMSpace"); // in receiver 0, in dataMSpace memspace
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver1_pt, "dataMSpace"); // in receiver 1, in dataMSpace memspace
```



# RASHPA EXAMPLE – DATA FLOW #3 (5/6)




ROI

dataset #1

1 frame / 5

```
rashpa_gdt_t *GDT = rashpaManagerCreateGDT(manager_pt, "GhostRoom", "CTR1");  // GDT "GhostRoom", dataset "CTR1"
rashpaGDTSetSourceImage(GDT, 190, 125, 190, 78);                              // ROI of (125 x 78) with offset (190, 78)
rashpaGDTSetAlignment(GDT, 1, 1, 0, 0);                                       // no data alignment, no header/footer
rashpaGDTSetSourceSlices(GDT, 0, 22, 5);                                      // starting at frame 0, 22 frames, 1 frame / 5
rashpaGDTSetRules(GDT, 1, 0, RASHPA_CONCATENATE | RASHPA_CIRCULAR);           // 1 block / group, starting at slot 0
rashpaGDTSetEvents(GDT, 0, RASHPA_EV_ATEND);                                  // event every data blocks
```

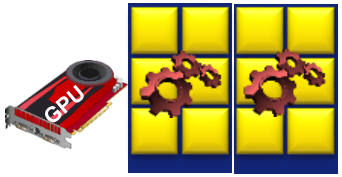
```
ssize_t lbuff_bytes = 10*(1024*1024);                                         // 10 MiB per local buffer
rashpa_buffer_t* buffer = rashpaManagerGetGDTBuffer(manager_pt, GDT, lbuff_bytes);
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver0_pt, "dataMSpace"); // in receiver 0, in dataMSpace memspace
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver1_pt, "dataMSpace"); // in receiver 1, in dataMSpace memspace
```

# RASHPA EXAMPLE – DATA FLOW #4 (6/6)


ROI

dataset #1

all frames




```
rashpa_gdt_t *GDT = rashpaManagerCreateGDT(manager_pt, "LeftCorridor", "CTR1");  // GDT LeftCorridor, dataset CTR1
rashpaGDTSetSourceImage(GDT, 35, 89, 204, 50);                                   // ROI of (89 x 50) with offset (35, 204)
rashpaGDTSetAlignment(GDT, 1, 1, 0, 0);                                          // no data alignment, no header/footer
rashpaGDTSetSourceSlices(GDT, 0, 114, 1);                                        // starting at frame 0, 114 frames, all frames
rashpaGDTSetRules(GDT, 4, 0, RASHPA_OVERWRITE | RASHPA_ONESHOT);                 // 4 blocks / group, starting at slot 0
rashpaGDTSetEvents(GDT, 1, RASHPA_EV_GRP | RASHPA_EV_ATEND);                     // event every group of data blocks
```

```
ssize_t lbuff_bytes = 10*(1024*1024);                                            // 10 MiB per local buffer
rashpa_buffer_t* buffer = rashpaManagerGetGDTBuffer(manager_pt, GDT, lbuff_bytes);
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver0_pt, "GPUMSpace"); // in receiver 0, in dataMSpace memspace
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver1_pt, "GPUMSpace"); // in receiver 1, in dataMSpace memspace
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver0_pt, "GPUMSpace"); // in receiver 0, in dataMSpace memspace
rashpaBufferCreateLocalBufferInMemSpace(buffer, receiver1_pt, "GPUMSpace"); // in receiver 1, in dataMSpace memspace
```

# PRESENTATION OUTLINE

Introduction

A. RASHPA: RDMA-based Data AcQuisition framework

I. Introduction
II. Architecture
III. Implementation
IV. Example

**B. LIMA2: Distributed Acquisition for High Performance Detectors**

**I. Motivations**
**II. Software Stack**
**III. Data transfers**
**IV. Processing**
**V. Plugins**

Conclusion & perspectives

# MOTIVATIONS

## Scalable to support the ever growing data throughput of 2D detector

➔ Distributed DAQ and Processing

## Flexible and Open-Ended to support the most exotic experiments of our users

➔ Decoupled DAQ, Data Transfer and Processing

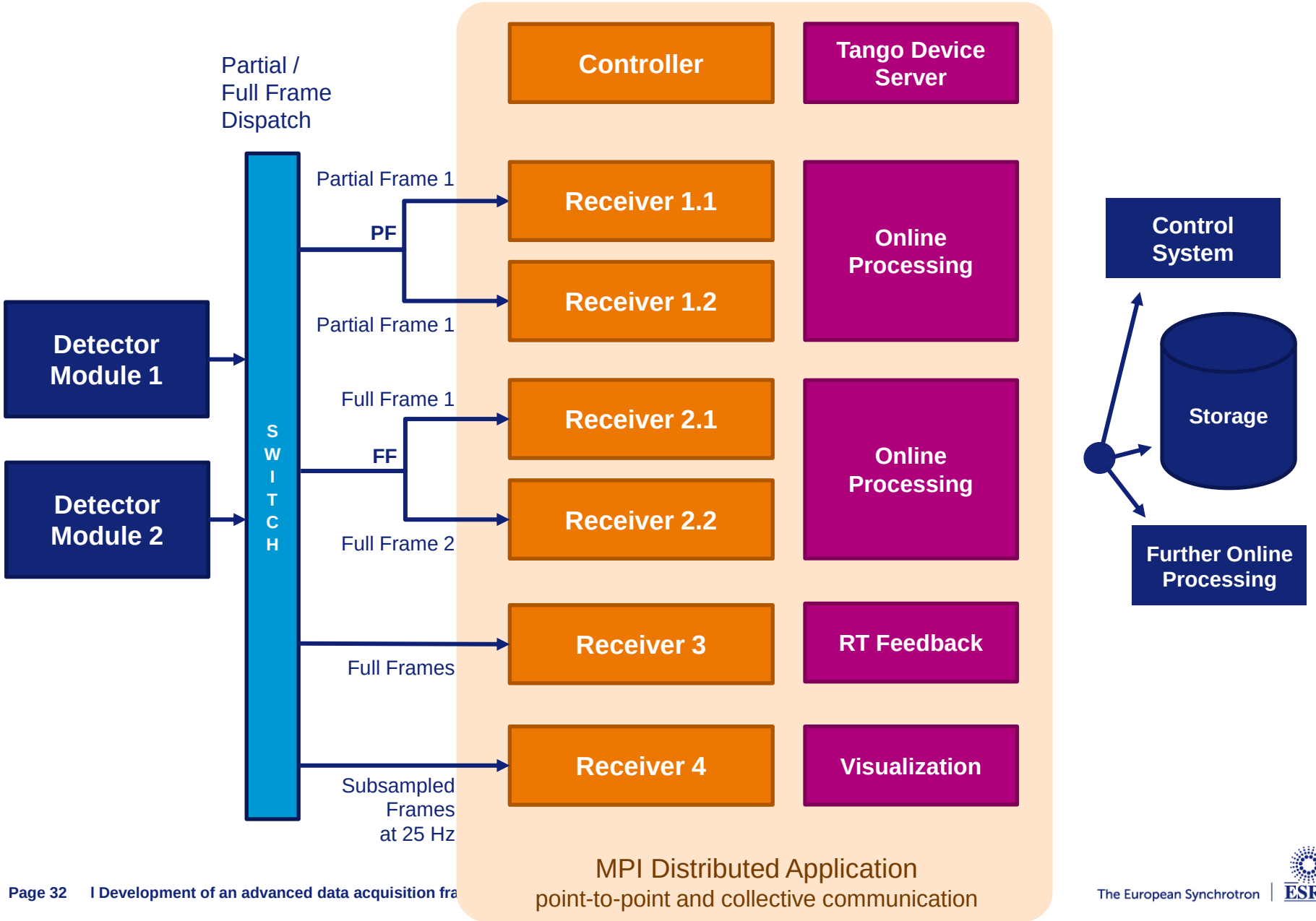
Reduce dead-time between acquisitions

Memory management and resource monitoring

Support multi-band, sparse and more data types

Re-built on solid foundation (with 3rd party libraries)

# TOPOLOGIES


Partial /
Full Frame
Dispatch
Controller
Tango Device Server
Detector Module 1
Detector Module 2
SWITCH
Partial Frame 1
PF
Partial Frame 1
Full Frame 1
FF
Full Frame 2
Full Frames
Subsampled Frames at 25 Hz
Receiver 1.1
Receiver 1.2
Receiver 2.1
Receiver 2.2
Receiver 3
Receiver 4
Online Processing
Online Processing
RT Feedback
Visualization
Control System
Storage
Further Online Processing
MPI Distributed Application
point-to-point and collective communication

# MESSAGE PASSING INTERFACE - MPI

MPI, the Message Passing Interface, is a standardized and portable **message-passing system**.

## Features

Optimal transport (shared memory, InfiniBand) of messages

- Synchronization of parallel tasks
- Sharing and moving data
- Collective operations

Process placement (affinity to NUMA node)

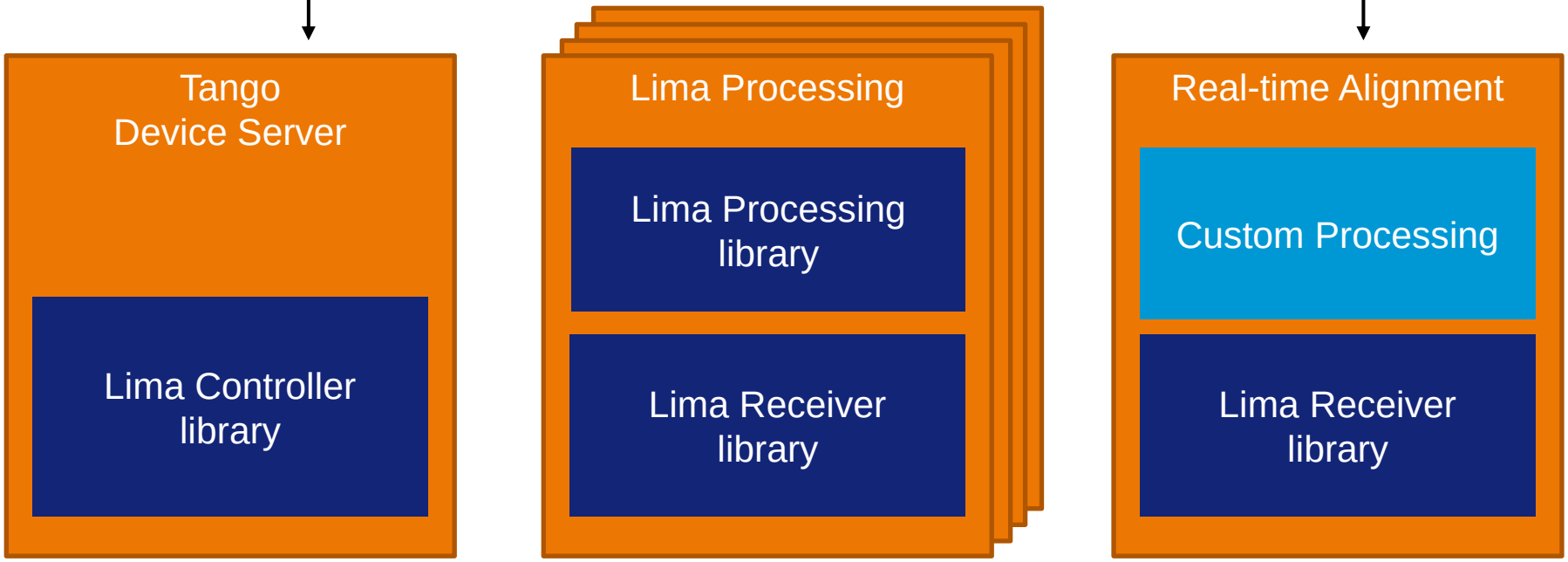
Cross platform (x86-64, POWER9), Linux, Windows…

Processes talk on *communicators* (attached to a group of process)

Communications can be *point to point* (e.g. send, recv) or *collective* (e.g. broadcast, reduce)

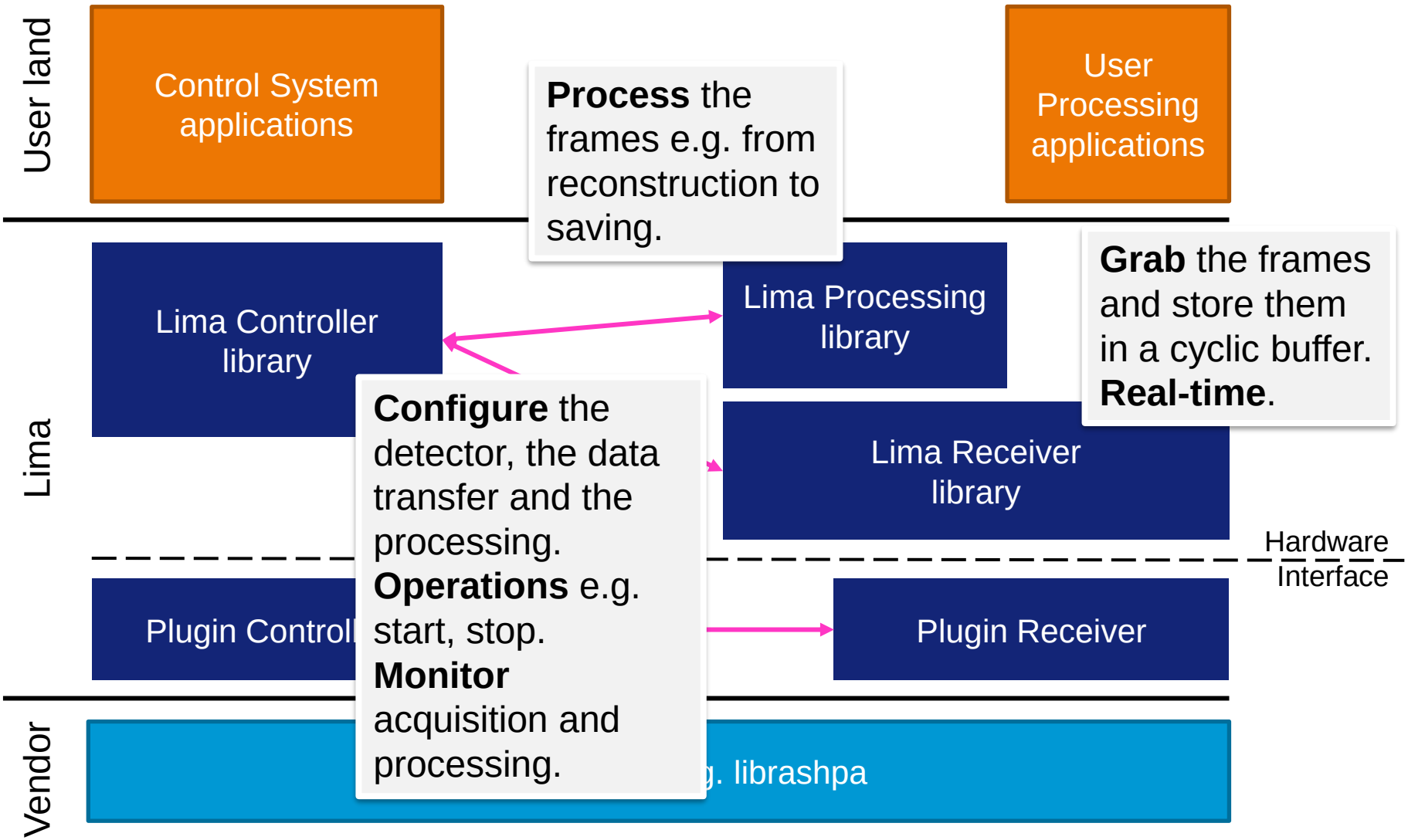
# RUNNING MPI (MPMD) APPLICATION

```
$ mpiexec -n 1 python -m lima_tds : -n 4 lima_recv : -n 1 python -m rt_align
```



# PRESENTATION OUTLINE

# SOFTWARE STACK



## Abstract Interface for Detectors


Detector static infos
- Name
- Pixel size
- Versions

Capabilities
- Data info (dimensions, bit depths …
- Binning, ROI, Flip
- Accumulation
- Shutter
- Sync / Triggers (soft, hard, gate…)
- Multiband (datasets)
- Data transfers (datachannels, reconstruction)
- Buffer (CPU/GPU, allocation, ownership..)

Configuration
```
prepare(acq_params, xfer_para
```

Operations
```
start()
stop()
reset()
```

Monitoring
```
status()
```

Acquisition
```
get_frames()
```

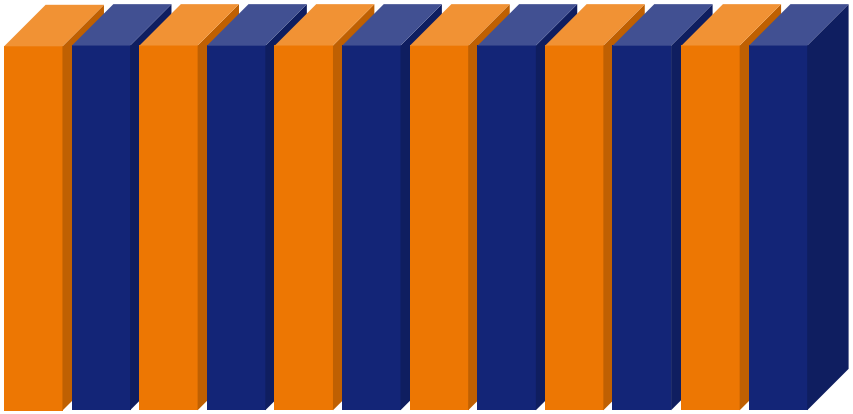
Configurat
```
prepar
```

Operations
```
start(
stop()
reset()
```

Monitoring
```
status()
nb_acq_frames()
```

# PRESENTATION OUTLINE

# DATA TRANSFER - FULL FRAMES

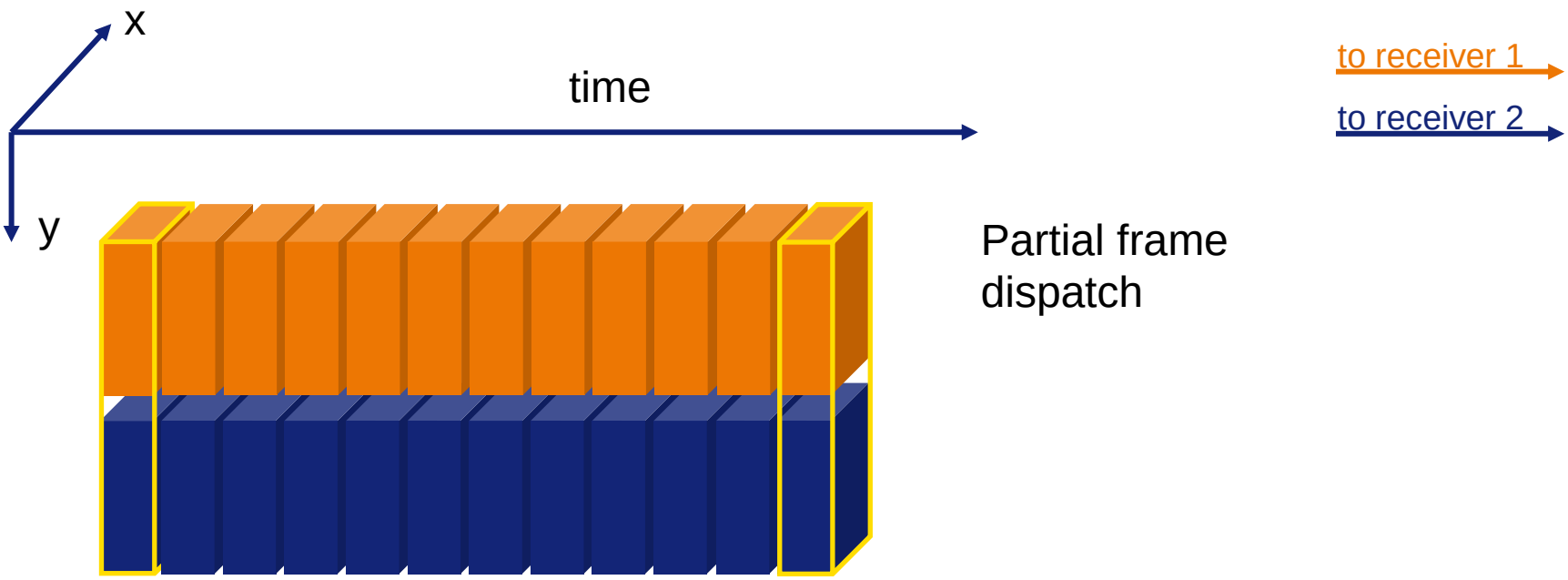



Full frame dispatch
(round robin)

Full frame chunk dispatch
(round robin)

For feeding a GPU
For frame accumulation

# DATA TRANSFER - PARTIAL FRAMES



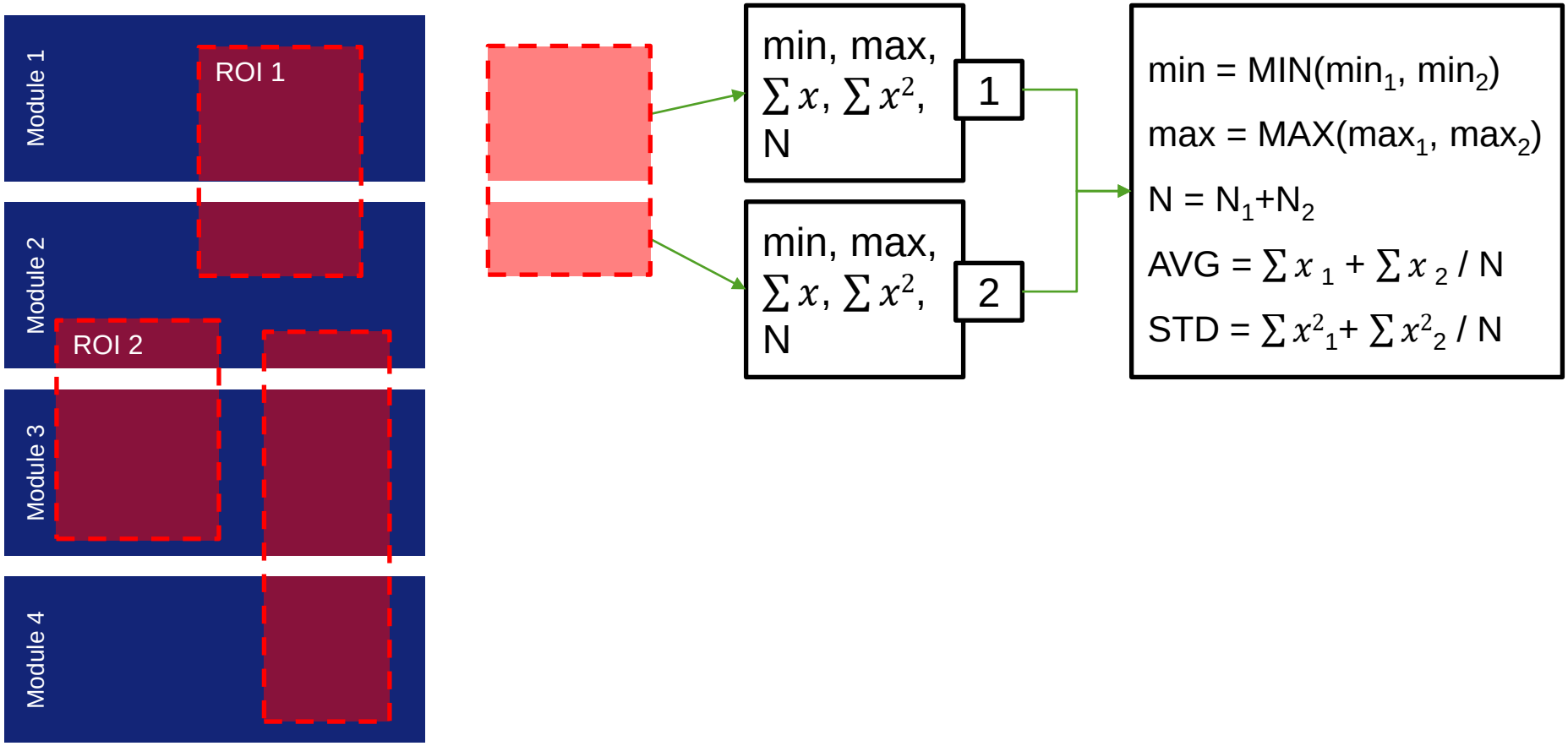

Typical data transfer for multi-module detector without Rashpa

Requires a reconstruction, depending on the use case, either

- At the beginning of the processing using collective « gather » operation - where each receivers gater the full frame in turn
- When saving to file assembling with an HDF5 Virtual Dataset

# PRESENTATION OUTLINE

# COOPERATIVE PARALLEL ALGORITHMS - ROI COUNTERS



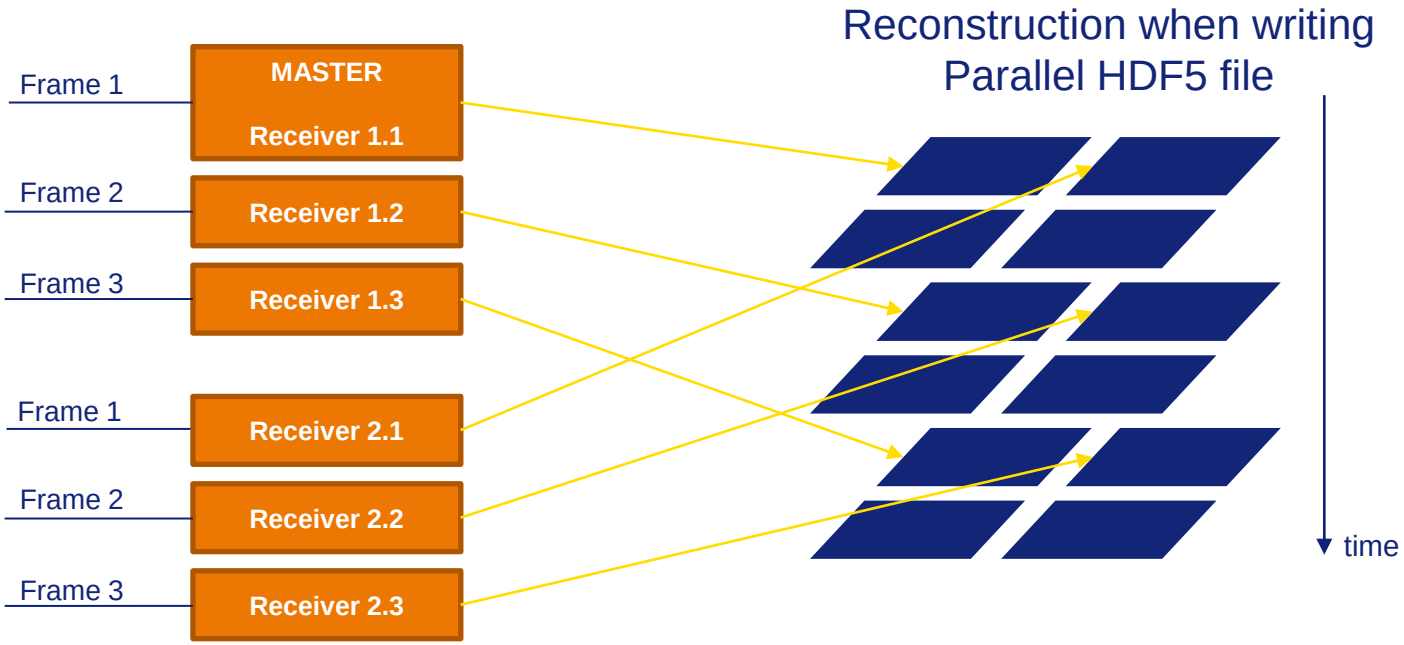
# COOPERATIVE PARALLEL I/O

## At the program level

- Concurrent reads or writes from multiple processes to a common file
- Build large blocks, so that reads/writes in I/O system will be large
  - Requests from different processes may be merged together
  - Particularly effective when the accesses of different processes are non-contiguous and interleaved

## At the system level

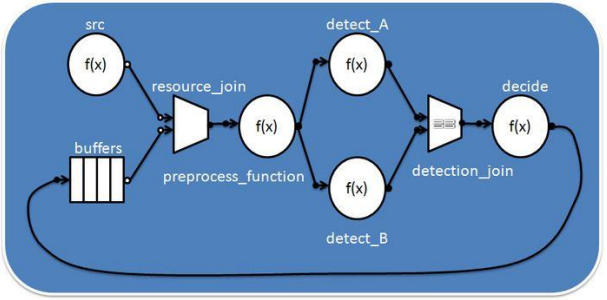
- A parallel file system and hardware that support such concurrent access (e.g. GPFS)



# PROCESSING GRAPH

## Data flow programming paradigm with TBB Flow Graph library

Expressive, Flexible, Fast and scalable




| TBB nodes | Description |
|---|---|
| `checkpoint_node` | Count frames and log idx / time |
| `io_hdf5_node` | Save frames to HDF5 format |
| `roi_flip_rot_node` | Apply geometric transformations |
| `reconstruction_node` | Reconstruct frames according to layout |
| `roi_counters_node` | Get statistics on ROIs |
| `background_sub_node` | Background subtraction |
| `flatfield_node` | Flatfield correction |
| `mask_node` | Mask defective pixels |
| `python_node` | Delegate processing to a Python interpreter |
| `mpi_node` | Delegate processing to another MPI process |

# PRESENTATION OUTLINE

Introduction

A. RASHPA: RDMA-based Data AcQuisition framework

I. Introduction
II. Architecture
III. Implementation
IV. Example

**B. LIMA2: Distributed Acquisition for High Performance Detectors**

**I. Motivations**
**II. Software Stack**
**III. Data transfers**
**IV. Processing**
**V. Plugins**

Conclusion & perspectives

# EXTENDING LIMA2

## On the Detector side

Implementing new **Detector Plugins**.

- SmartPix
- PSI Detectors (starting with JungFrau)
- REMU / Simulator

## On the Processing side

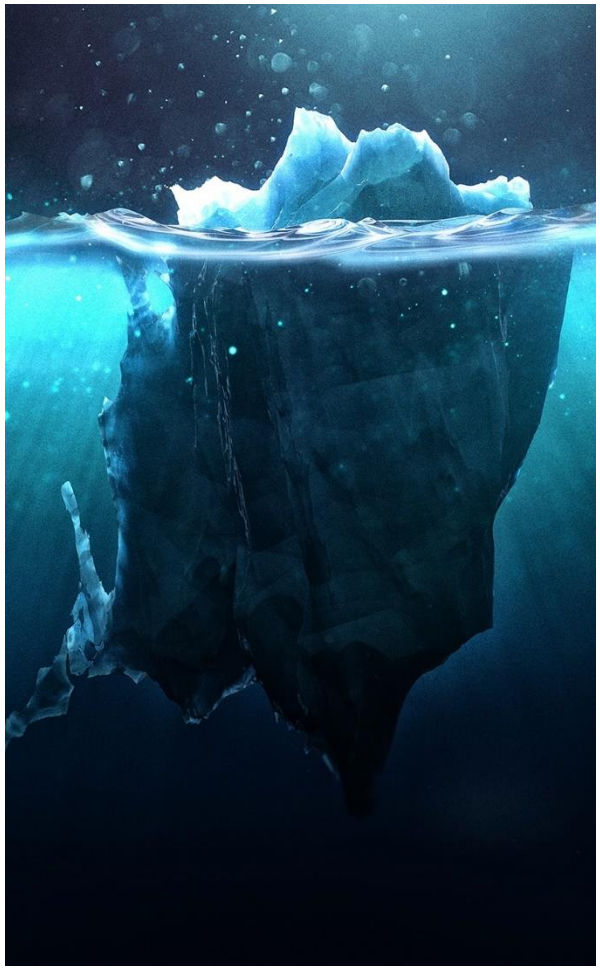
Implementing new **Processing Nodes** (including in Python)

Implementing new **Processing Graphs**

Implementing new **Processing Systems** on top of the receiver API

# PRESENTATION OUTLINE

# CONCLUSION & PERSPECTIVES



Thank you !

Lima2 is **Work in Progress**

**Open-source**, available on ESRF's Gitlab instance

Open to **Cooperation** and Contribution

Integration of SmartPix and PSI JungFrau for Q1-Q2 2021

More data transfer and processing options to be implemented

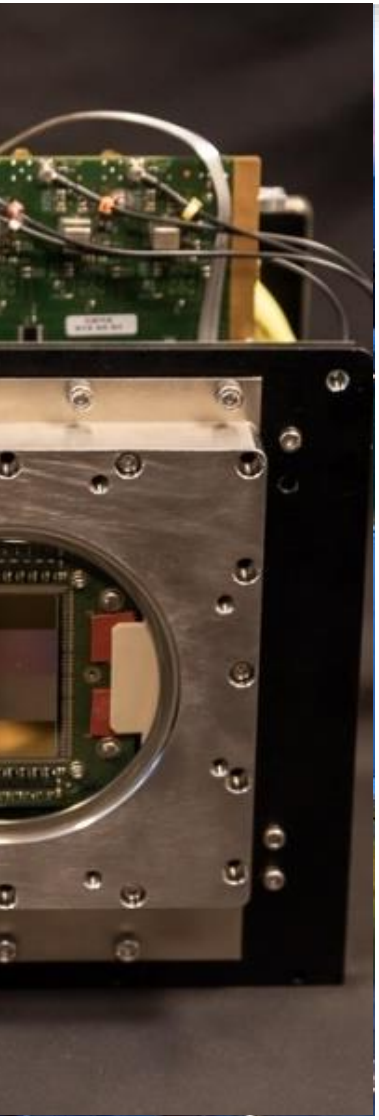
Rashpa **v1.0 release** foreseen for Q1 2021

Main DAQ framework for the SmartPix detector

Support within Lima2 under development

Demonstration on a beam-line foreseen in 2021

ESRF | The European Synchrotron



# Development of an advanced data acquisition framework for 2D X-Ray Detectors


**RASHPA team:**

Aurélien BIDEAUD, Pablo FAJARDO, Nicolas JANVIER, Wassim MANSOUR, Raphaël PONSARD

**LIMA2 team:**

Laurent CLAUSTRE, Samuel DEBIONNE, Alejandro HOMS


EIRO forum serving european science

Big Data