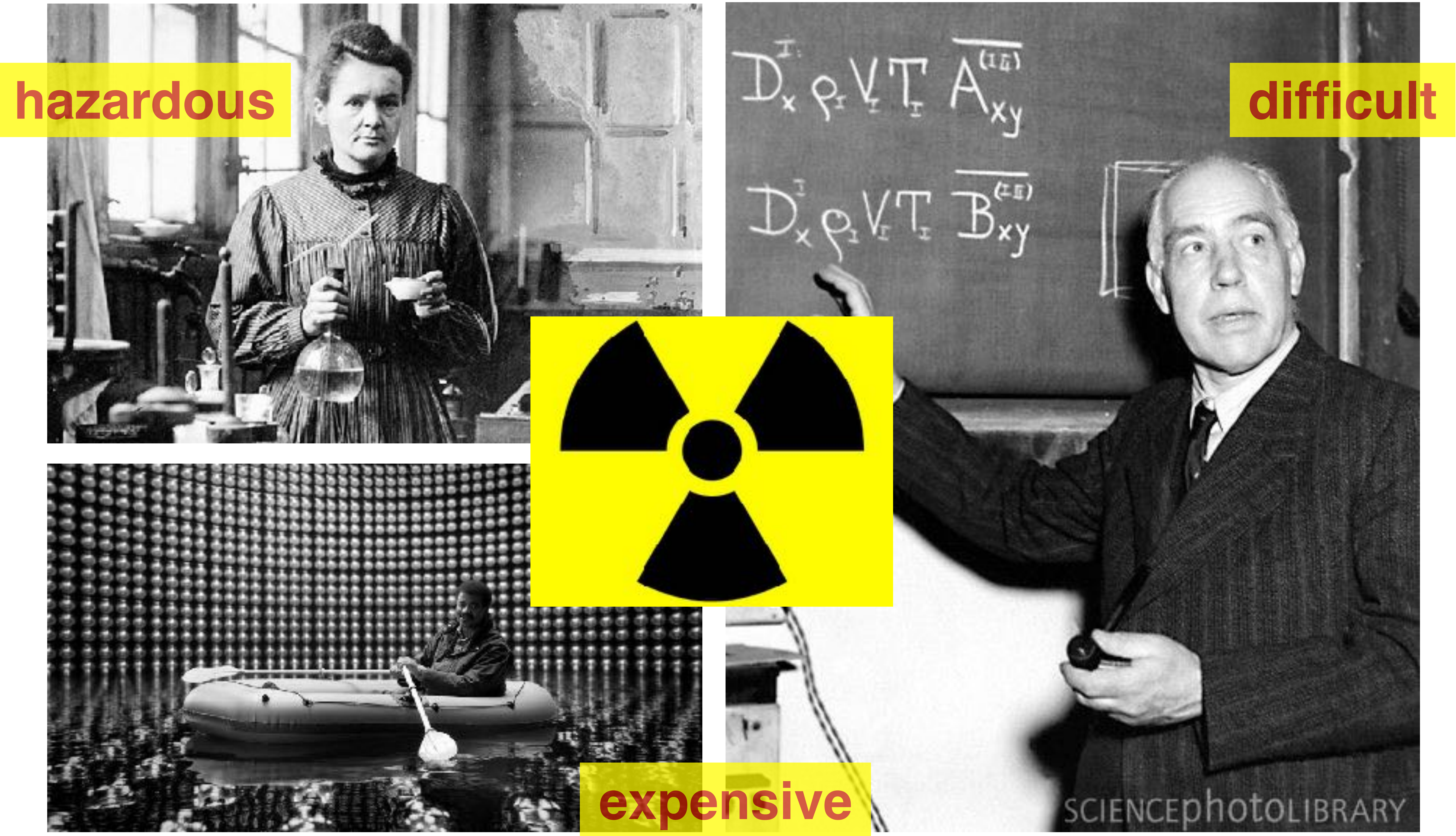
Online Processing p.1 Introduction

Radoslaw Karabowicz
Denis Klein
FairRoot Group, GSI

Nuclear experiments

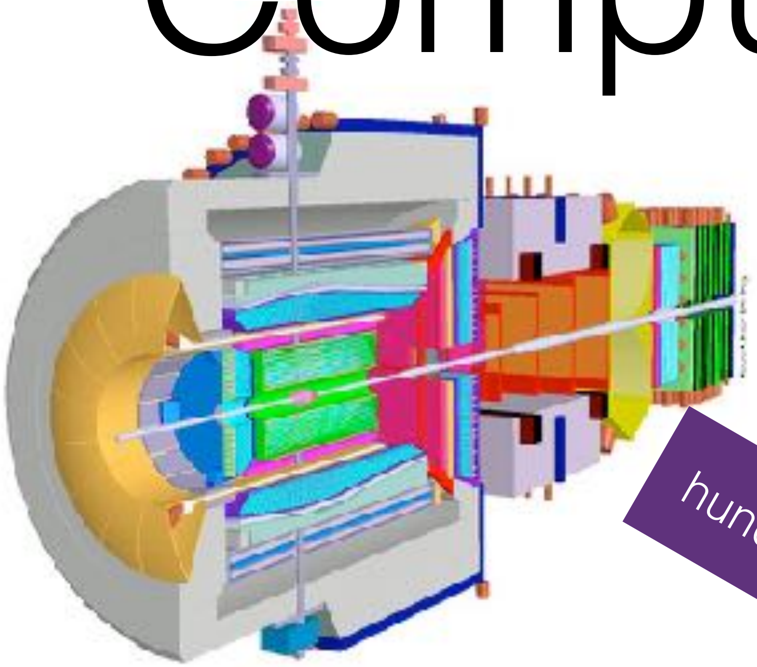


hazardous

difficult

expensive

Computing challenges



new HEP experiments producs huge amounts of data

hundreds GB/s

traditional trigger systems are extended by heterogenous (CPU, GPU, FPGA, …) online computing farms to reduce the data flow

couple of GB/s

the stored data are repeatedly being analyzed in the offline computing facilities (farms, grid, cloud)

Software levels

- Data transport
- Farm configuration
- Farm control & management
- Data reconstruction
- Physics analysis
- Parameter accessibility
- Parameter updates

Framework requierements

- Extend traditional single-process FairRoot to allow **simulation and reconstruction of streaming data**.
- High **performance**, **flexibility** and **scalability** are required!
- Separate running of tasks in **multiple processes** (that can be multi-threaded or use GPU) to increase stability (process crashes do not immediately affect other components and allow easier recovery).
- Communicate via **platform-independent messages** (simplify integration of different hardware and programming languages).

FairMQ

- Project started in 2012.
- People involved: Mohammad Al-Turany, Dennis Klein, Alexey Rybalchenko, Nicolas Winckler, Dennis Klein.
- Message Queue (MQ) - transport messages between processes using queues.
- FairMQ - data transport framework to organize processing tasks in topologies, consisting of independent processes (devices), communicating via asynchronous message queues over network or inter-process channels.

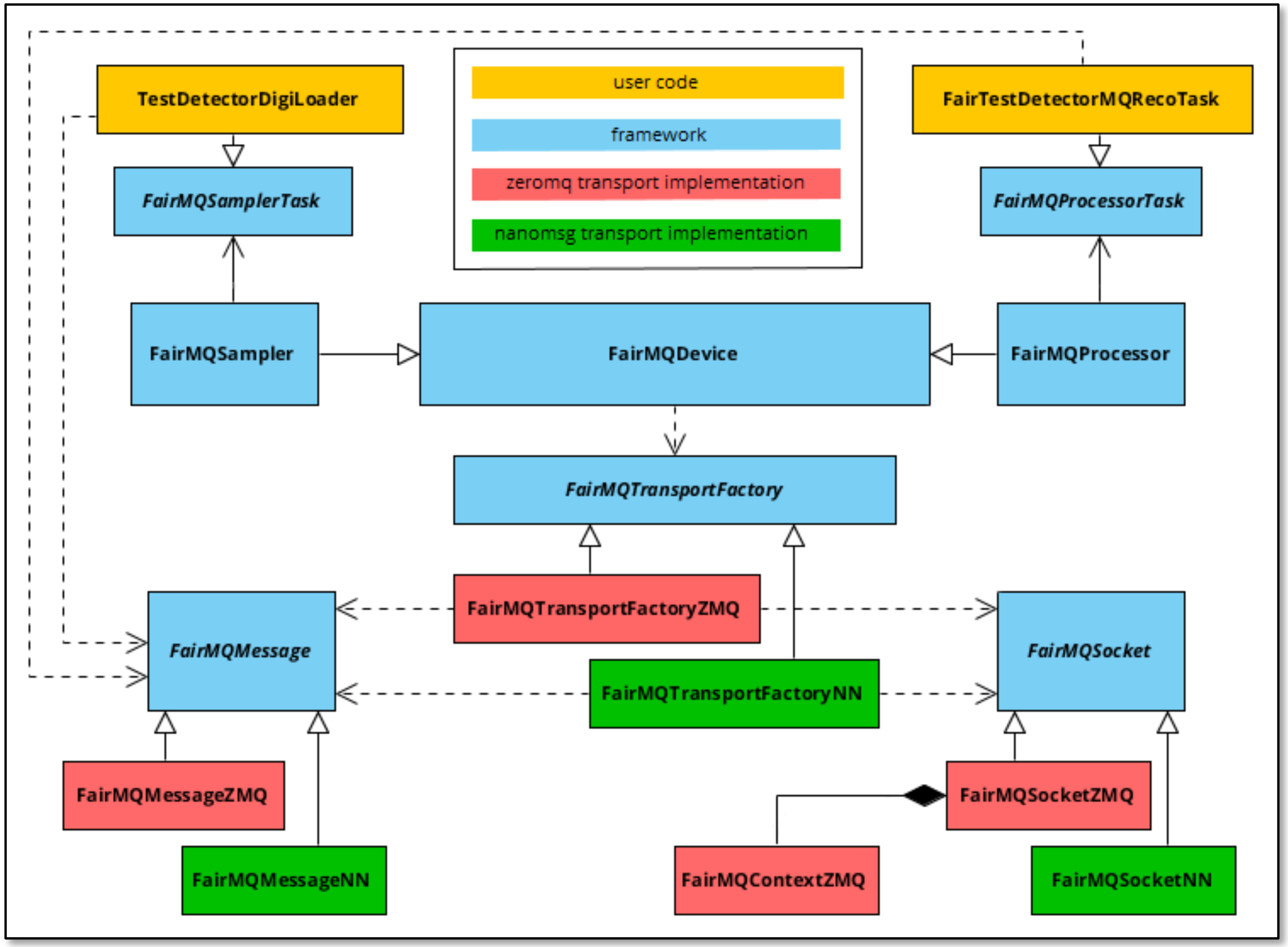
FairMQ Transport Interface

FairMQ transport interface keeps the device and task code independent of the transport implementation (implementing Send/Receive/Polling methods, (Multipart) messages, patterns).

Currently two implementations:

- **ZeroMQ** (TCP, IPC, INPROC, PGM, TIPC, VMCI);
- **nanomsg** (TCP, IPC, INPROC, WebSockets).

Open for implementation with future technologies. Has to fit in the message passing model (or interface it).



Radoslaw Karabowicz, GSI

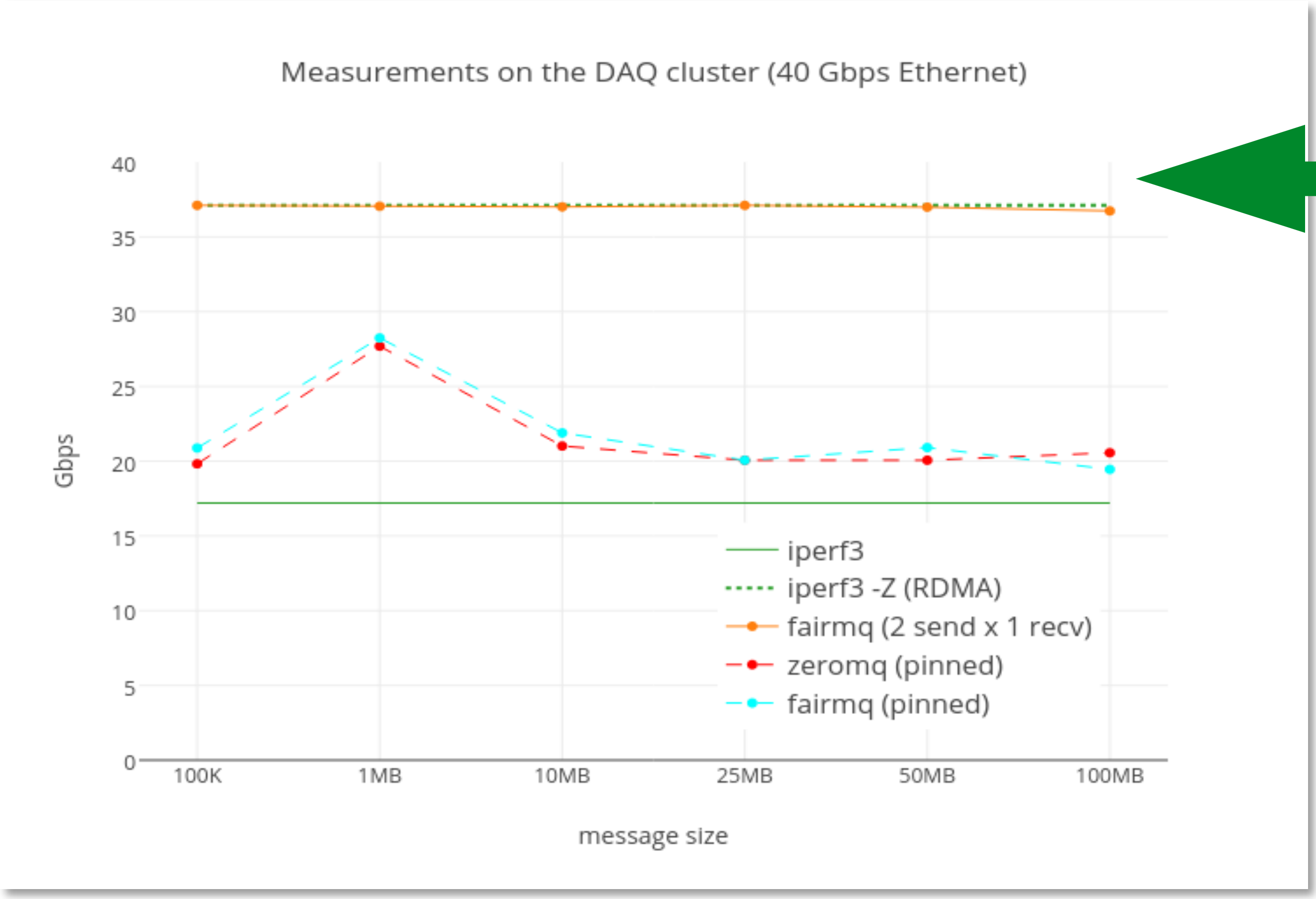
FairMQ

Transport based on:

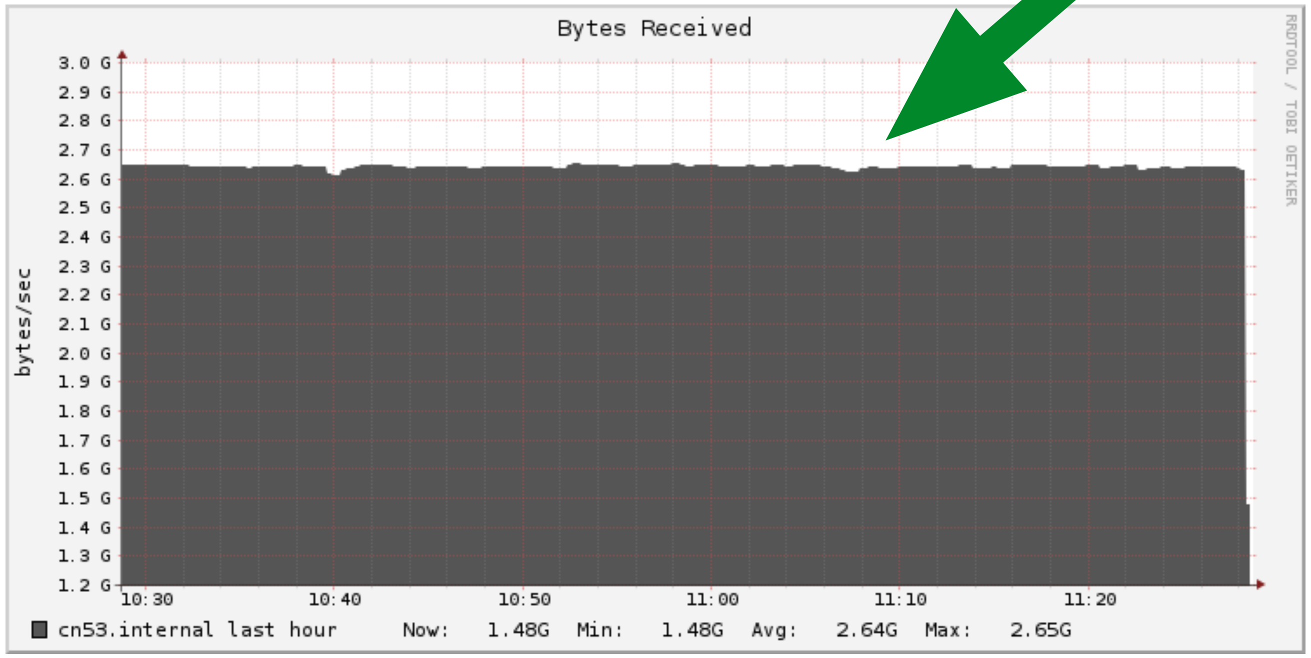
ZeroMQ http://zeromq.org/ // production quality, large community, active development, extensive documentation, examples and patterns

nanomsg http://nanomsg.org/ // still beta, several architectural improvements, cleaner internal API (eg. for adding new transport technologies)

Low overhead abstractions, staying close to theoretical performance limit of the protocol/network technology



Sustained high performance on Ethernet, InfiniBand (IPoIB)



**2x Intel Xeon E5-2650 @ 2.60GHz (8 cores, 16 threads) (dual socket), 40 Gbps Ethernet,
3.10.x kernel**

**2 x Intel Xeon E5520 @ 2.27GHz (4 cores each, 8 threads) (dual socket), QDR InfiniBand (IPoIB),
3 Processes on 3.17.x kernel.**

Radoslaw Karabowicz, GSI

FairMQ general

- Promotes separation of concerns to increase maintainability and reduce coupling: simple single-responsibility devices.

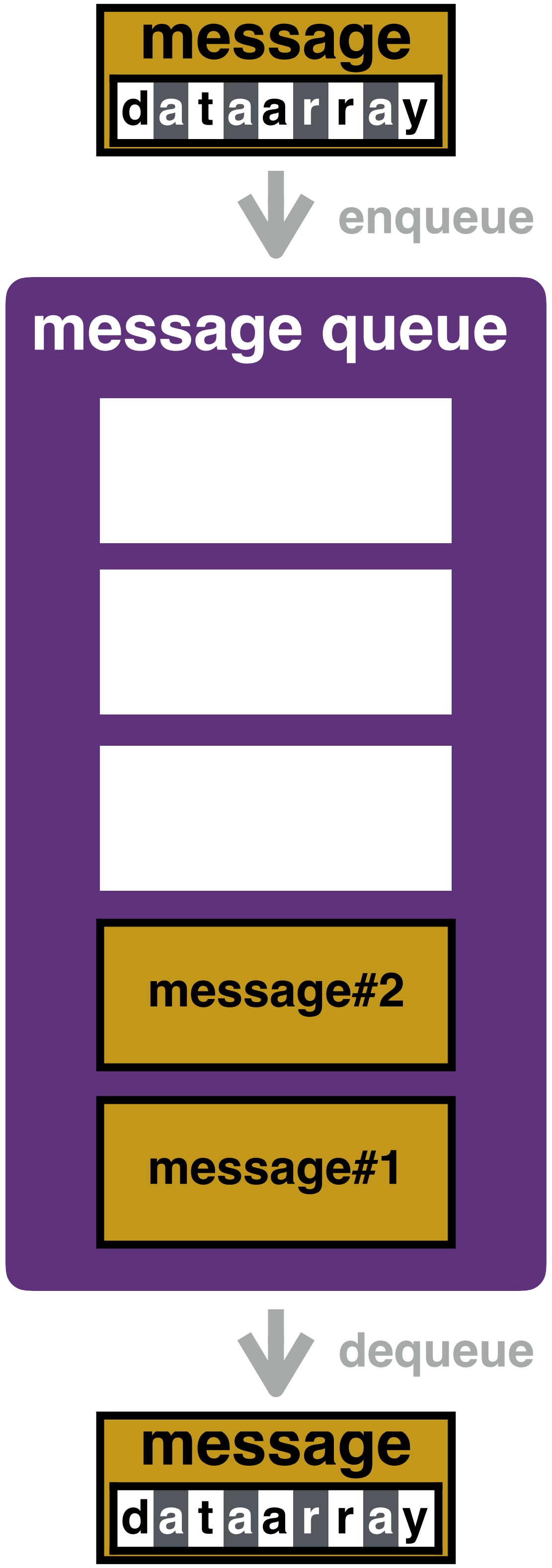
- Keeps the user and transport code separate, allowing them to evolve independently: abstract transport interface, user task separation.

- Simplifies and unifies handling of transport details: socket settings, polling, termination, access to communication patterns.

- Takes care of the device state (change, query), termination, command interface, device configuration, logging.

Message Queue

- Message - data to be transported between two processes, essentially it is an array of bytes (defined by the address to the first byte and the size of the array) and a header:
  - FairMQMessage - simple message;
  - FarMQParts - message composed of multiple parts.
- Queue - array of messages waiting to be processed, arranged in a FIFO pipeline.



FairMQ task

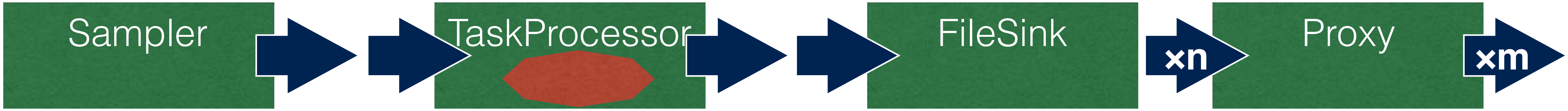
- Task contains an algorithm to address (hopefully) one specific physics problem (f.e. cluster finding, track fitting, PID assignment, etc.)

ClusterFinder TrackFinder TrackFitter TrackMerger PID

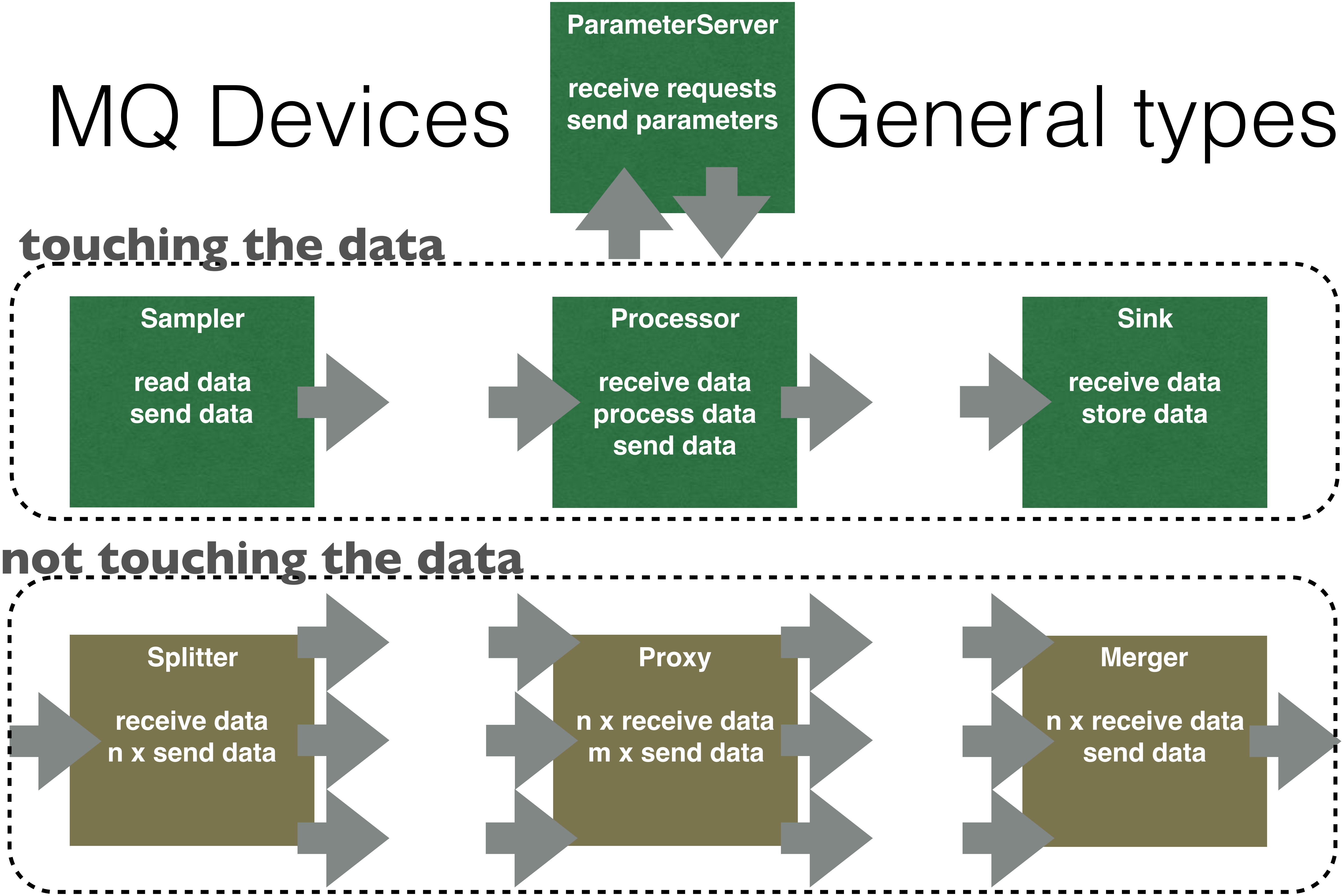
- It relates directly to FairTask.
- But, contrary to running under FairRoot, in FairMQ the task has no direct access to FairRunAna, FairRootManager, FairRuntimeDb

FairMQ devices

- Simple single-responsibility computer programm able to receive and send messages over various channels.
- Messages may contain data, runtime parameters, configuration or communication data.
- It may encapsulate a task.
- There are also devices not connected to tasks, for example to provide IO access, to read and send parameters, to split or merge data streams.



MQ Devices General types



ParameterServer

receive requests
send parameters

touching the data

Sampler

read data
send data

Processor

receive data
process data
send data

Sink

receive data
store data

not touching the data

Splitter

receive data
n x send data

Proxy

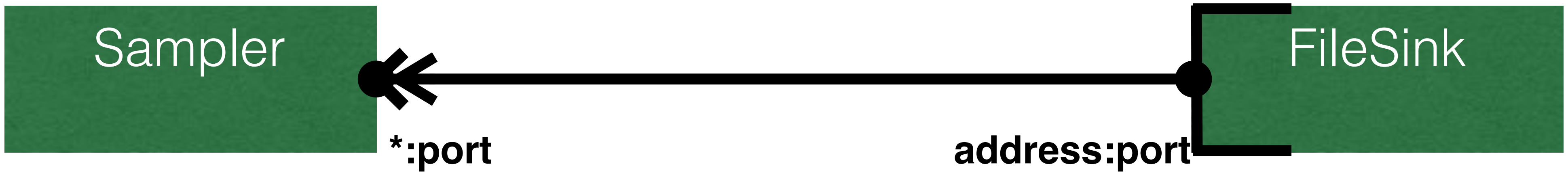
n x receive data
m x send data

Merger

n x receive data
send data

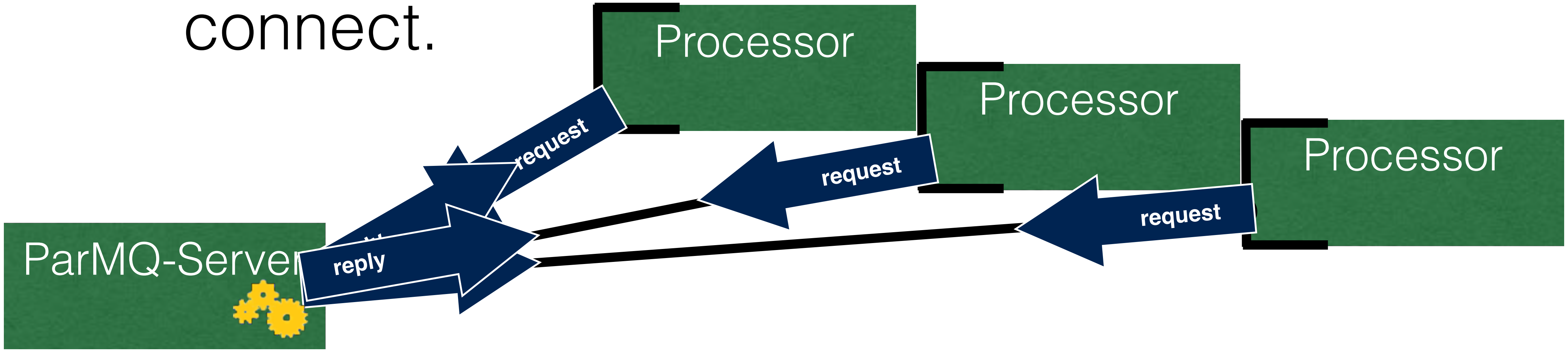
FairMQ channel

- Channel connects devices.
- There is always one device that binds a specific port on the host machine.
- Another devices connect to that port at a given address.
- The messaging pattern defines the way of message exchange.



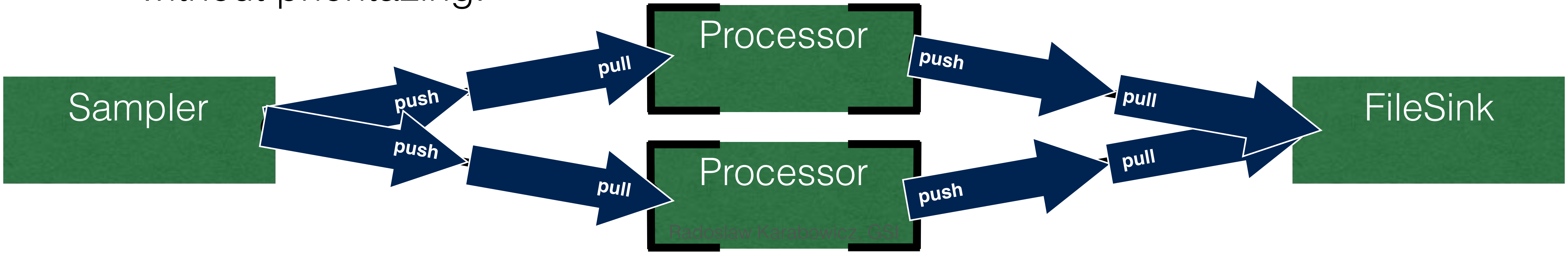
FairMQ req-rep

- the commom example of request-reply pattern is the comunication between a server and clients;
- the clients **request** a specific data or action;
- the server **replies** by sending the data or an answer;
- in this pattern the server has to bind, the clients connect.



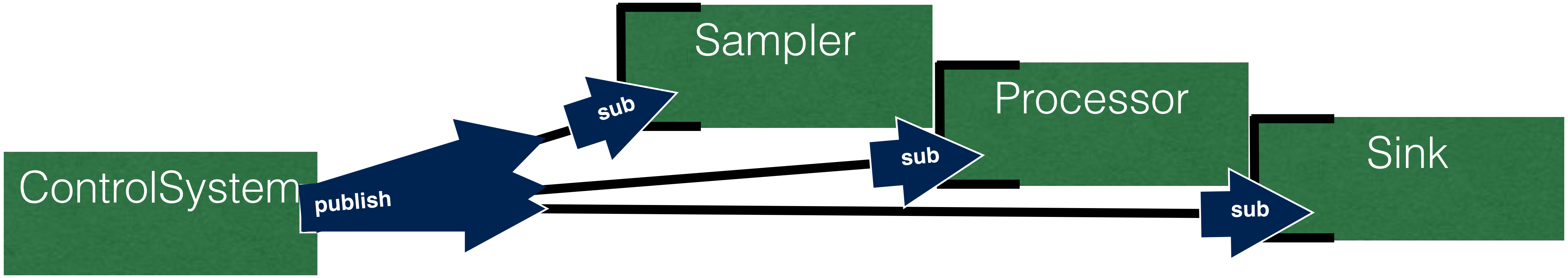
FairMQ push-pull

- the push-pull pattern works like a pipeline: the messages flow always in one direction;
- one side **pushes** the messages into the queue;
- another side **pulls** the messages from the queue;
- this pattern works either 1to1, 1toN or Nto1;
- in case of 1toN, the messages will be distributed among the N receivers evenly;
- in the case of Nto1, the receiver will receive the messages from senders without prioritazing.



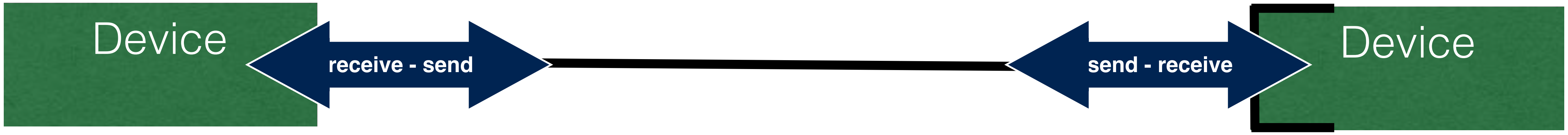
FairMQ pub-sub

- the publish-subscribe pattern could be used for controling purposes;
- the subscriber(s) gets the messages after they are published by the publisher(s);
- it works 1to1, 1toN or Nto1;
- if no subscribers connected, the messages will be lost;
- otherwise each subscriber gets each message.



FairMQ PAIR

- bidirectional communication between two nodes.

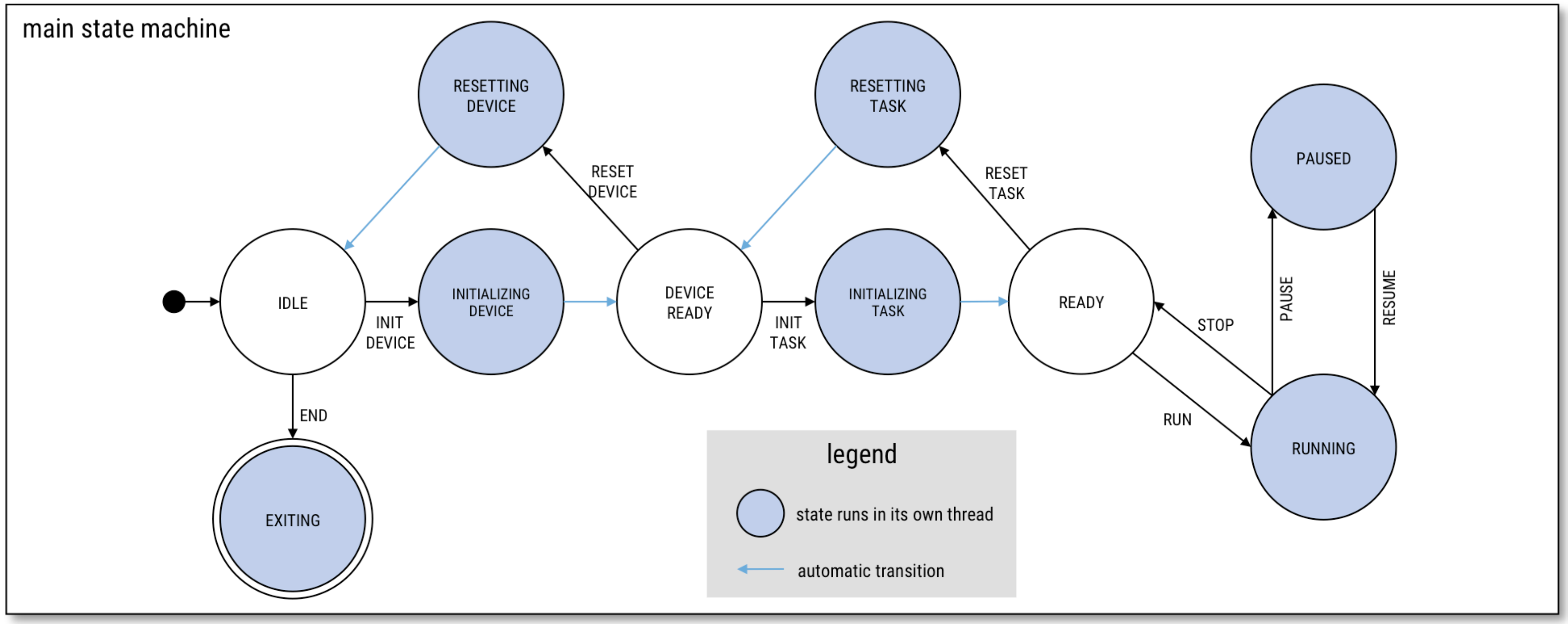


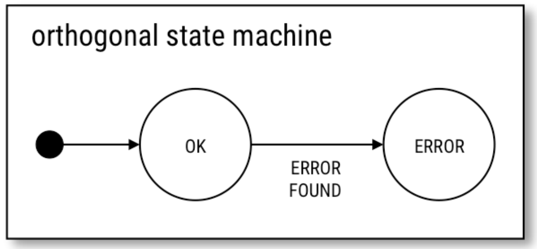
FairMQ State Machine

A complex topology may require some degree of **synchronization** between different stages of the processing pipeline.

To allow this, the state of the devices can be queried and changed in response to an external control system. This is handled by the **FairMQStateMachine**.

The **device remains responsive** regardless of the condition of the user code, by running it in a separate thread.





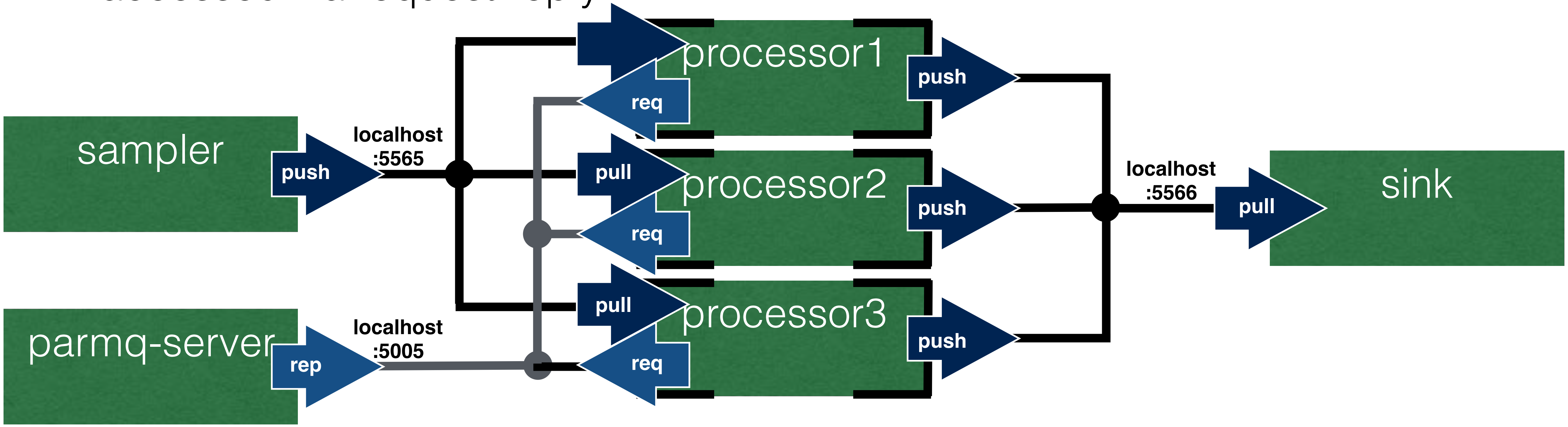
Additionally, an orthogonal OK/ERROR state reflects the device condition. If error occurs, the device remains in the ERROR state to allow debuggin.

FairMQ topology

- the different devices are used as “LEGO” blocks;
- the structure of all the blocks used for a given problem is called a **topology**;
- it lists the **devices** used, states the **connections** between them and describes the communication **patterns**;

FairMQ topology

- defined in a JSON file;
- contains the list of devices (by the unique IDs);
- specifies connection types and communication patterns, addresses and ports, buffer sizes, verbosity level;
- example view of a topology with 1 sampler (bind), 1 sink (bind), 3 processors (connect) with push-pull communication pattern + a patameter server accessed via request-reply:


processor1
req
push
sampler
push
localhost
:5565
pull
processor2
req
push
localhost
:5566
pull
sink
pull
processor3
req
push
parmq-server
rep
localhost
:5005