FairMQ application example

Radoslaw Karabowicz, GSI

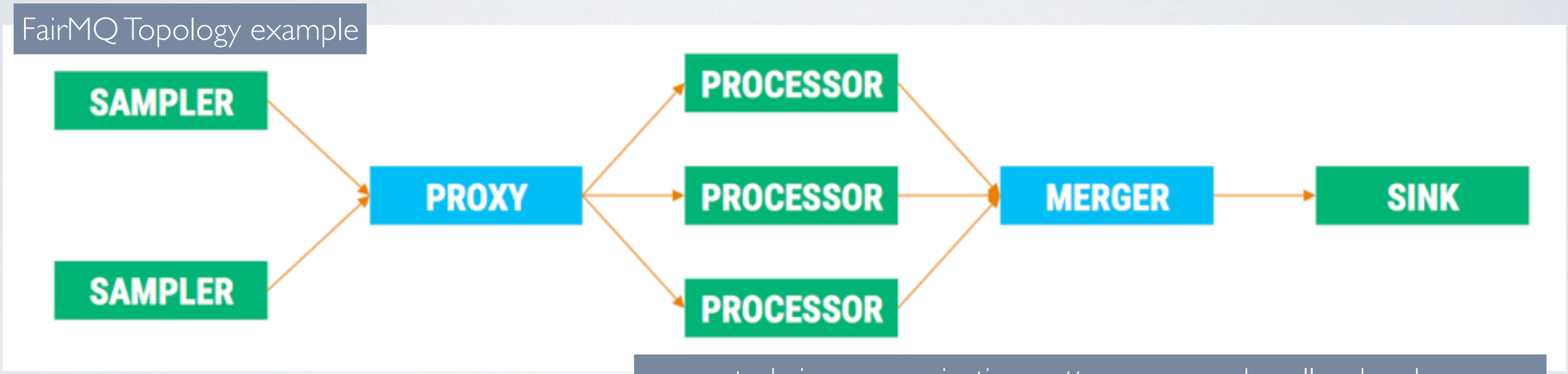
ALFA Framework

- FairMQ is developed as a part of a larger project, the ALFA framework.
- ALFA is developed in a collaboration between the FairRoot Group, experiments at FAIR and ALICE.
- The data-flow based model is grounded on the Message Queues based multi-processing (FairMQ).
- Provides configuration, process management and monitoring tools.
- Provides unified access to configuration parameters and databases.


Experiment control (ALICE)
Offline teams (ALICE, CBM, PANDA, R3B, …)
Online (ALICE, PANDA)
QA (ALICE)
ALFA
DAQ (ALICE)
Cluster (GSI, ALICE)
Industry
Software developers (FairRoot Group)
CISCO
CERN openlab
intel

What is FairMQ?

- A data transport framework to organize processing tasks in topologies, consisting of independent processes (Devices), communication via asynchronous message queues over network or inter-process channels.



- Ready to use devices are available for common scenarios.

- User-defined devices can be implemented by inheriting from `FairMQDevice` class.

What is FairMQ?

- Defines workflow for the process.
- 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 the transport details: socket settings, polling, termination, access to communication patterns.
- Takes care of the device state (change, query), termination, command interface, device configuration, logging.

motivation behind MQ/example9

- provide simple example to show how to switch from single core to multicore pipeline processing;
- show how to use generic MQ tools;
- show how to send and receive FairRoot data;
- show how to execute the FairRoot tasks;

https://github.com/FairRootGroup/FairRoot/tree/dev

/examples/MQ/9-PixelDetector

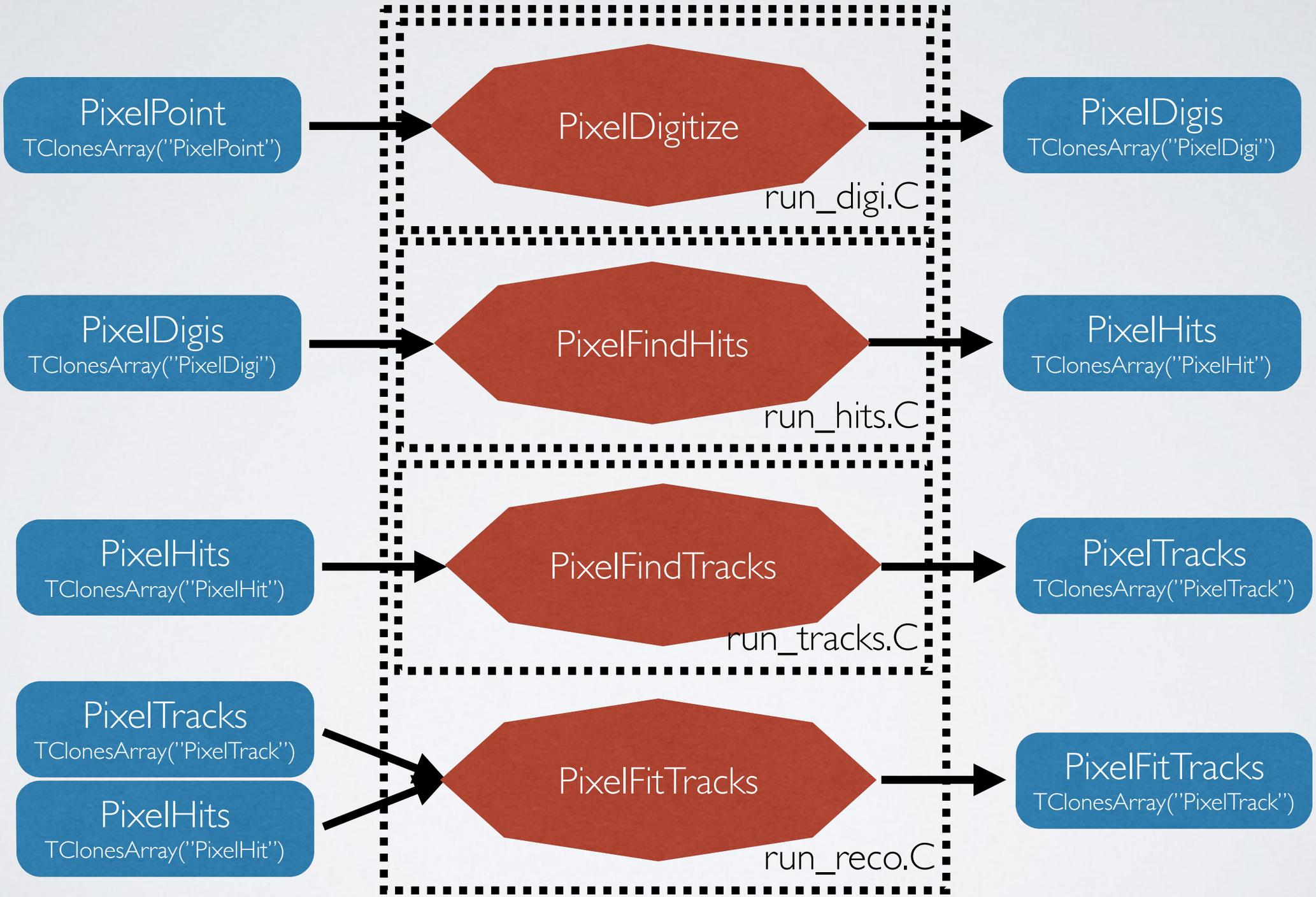
- 3 stations with 4 rectangular sensor each:
  - size: 5x 5cm^2, inner hole: 1x1cm^2, at z = 5cm;
  - size: 10x10cm^2, inner hole: 1x1cm^2, at z = 10cm;
  - size: 20x20cm^2, inner hole: 2x2cm^2, at z = 20cm;
- each sensor divided into pixels (0.01x0.01cm^2), that are grouped into FE modules (110 pixels x 116 pixels)

FEs numbering on one sensor

…					
FE 5	…				
FE 4	FE 68	…			
FE 3	FE 67	FE 131	…		
FE 2	FE 66	FE 130	FE 194	…	
FE 1	FE 65	FE 129	FE 193	FE 257	…

```
[INFO   ] ------------ Pixel Digitizer : Summary --------
[INFO   ]  Events:         100000
[INFO   ]  MC Points:      944127    ( 9.44127 per event )
[INFO   ]  Digis:          940281    ( 9.40281 per event )
[INFO   ] ---------------------------------------------
[INFO   ] ------------ Pixel Hit Finder : Summary -------
[INFO   ]  Events:         100000
[INFO   ]  Digis:          940281    ( 9.40281 per event )
[INFO   ]  Hits:           940281    ( 9.40281 per event )
[INFO   ] ---------------------------------------------
[INFO   ] ------------ Pixel Track Finder : Summary -----
[INFO   ]  Events:         100000
[INFO   ]  Hits:           940281    ( 9.40281 per event )
[INFO   ]  Tracks:         281431    ( 2.81431 per event )
[INFO   ] ---------------------------------------------
[INFO   ] ------------ Pixel Track Fitter : Summary -----
[INFO   ]  Events:         100000
[INFO   ]  Tracks:         281431    ( 2.81431 per event )
[INFO   ]  Fitted Tracks: 281431    ( 2.81431 per event )
[INFO   ] ---------------------------------------------
```

data classes, tasks and macros



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

MQ concepts

- **FairMQParts** - multi-part message that is sent between devices;

- **TMessage** - a ROOT wrapper for any TObject that can be added to FairMQParts (another options: binary, protobuf, boost);

MQ Devices for FairRoot data processing

- **FairMQEx9Sampler** reads data branch names specified by AddInputBranchName(string) from the input file(s) set by AddInputFileName(string);

- **template <typename T>**
  FairMQEx9TaskProcessor runs a task of **class T**. The class T needs to have the following functions:

```
void GetParList (TList* parList);
void InitMQ     (TList* parList);
void ExecMQ     (TList* inputList, TList* outputList);
```

- **FairMQEx9FileSink** creates output file set by SetOutputFileName(string) and tree, with the branches specified by AddOutputBranch(string className, string branchName).
  Currently only two classes (FairEventHeader and TClonesArray) are allowed:

```
fileSink.AddOutputBranch("FairEventHeader","EventHeader.");
fileSink.AddOutputBranch("TClonesArray(anyclassname)","branchname");
```

- **ParameterMQServer** creates instance of the FairRuntimeDb, which reads the parameters from ROOT or ASCII files.

FairMQEx9Sampler

- InitTask(): creates FairFileSource* fSource with the provided input file names and activates all the needed branches (getting TObject*) in the fSource;

- Run(): calls fSource->ReadEvent(): each requested TObject* is wrapped in a TMessage and added to a multipart FairMQParts message; the result FairMQParts message is sent for each event;

template<typename T> FairMQEx9TaskProcessor

- InitTask(): creates object fFairTask of class T, asks fFairTask to fill the TList of needed parameters (by calling fFairTask->GetParList()); creates empty input and output TLists;

- Run(): receives the FairMQParts message; runs through its parts to fill the input TList; if FairEventHeader object is found and the RunId changed, the new parameters are requested from the ParameterMQServer and the fFairTask is initialised with the new parameters (by calling fFairTask->InitMQ()); calls the ExecMQ function of the fFairTask; wraps all the TObjects in the output TList in the TMessages and adds them to the output FairMQParts; sends the result output FairMQParts

FairMQEx9FileSink

- InitTask(): creates output TFile and TTree; adds requested objects to the output TTree;

- Run(): receives the FairMQParts message; runs through its parts to set the branch addresses to the received objects; fills the output tree;

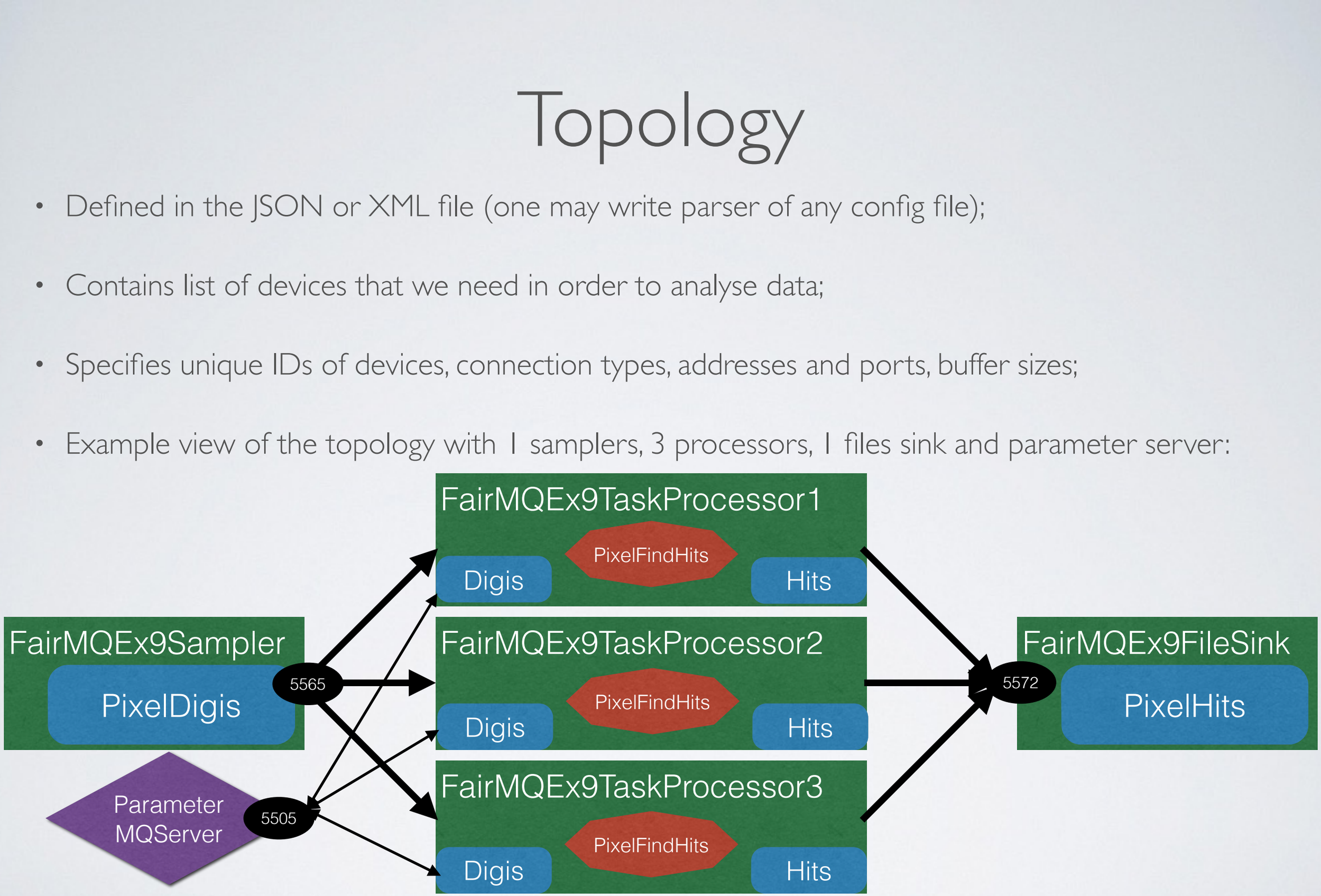
Starting MQ devices

- MQ Devices are run by c++ programs, compiled and stored in the bin folder of your build and install directories.
- Each MQ device has several settings that should be provided at startup:
  - transport protocol (zeromq, nanomsg);
  - verbosity (TRACE, DEBUG*, INFO, NOLOG);
  - unique string ID;
  - connection type (push/pull,req/rep,pub/sub) and method (bind or connect), address and port, buffer sizes (all set in a config file, JSON or XML)
  - plus additional settings of the particular MQ device implementation.
- Example command to start program containing MQ device:

```
./FairMQEx9Sampler --transport zeromq --verbose INFO --id sampler1
--mq-config ../share/fairbase/examples/MQ/9-PixelDetector/run/options/Pixel9MQConfig_Multipart.json
--file-name ../share/fairbase/examples/MQ/9-PixelDetector/macros/pixel_TGeant3.digi.root
--branch-name PixelDigis
```

Topology

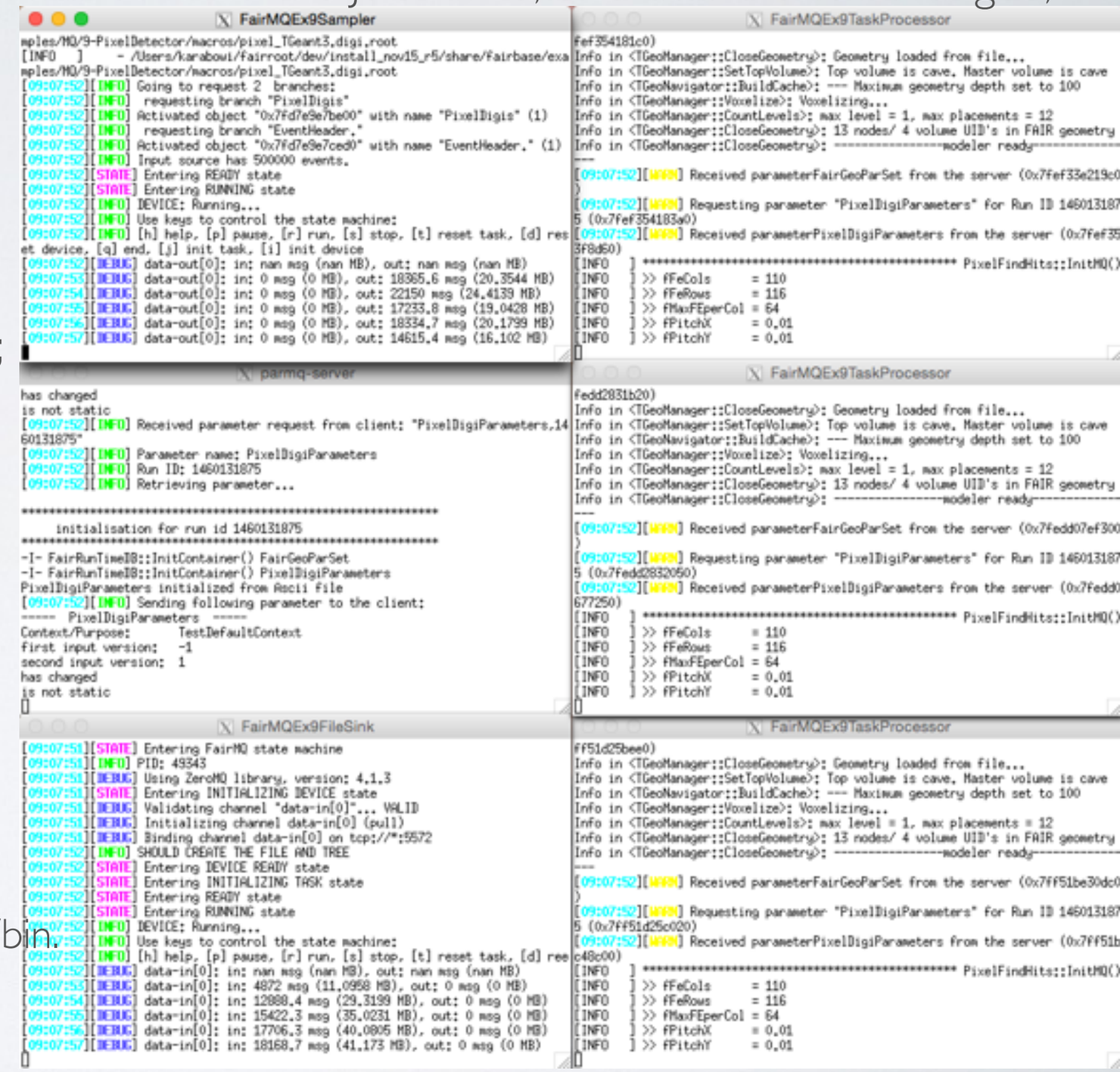
- Defined in the JSON or XML file (one may write parser of any config file);
- Contains list of devices that we need in order to analyse data;
- Specifies unique IDs of devices, connection types, addresses and ports, buffer sizes;
- Example view of the topology with 1 samplers, 3 processors, 1 files sink and parameter server:



- Several topologies provided in the 9-PixelDetector/run/options.

Shell script

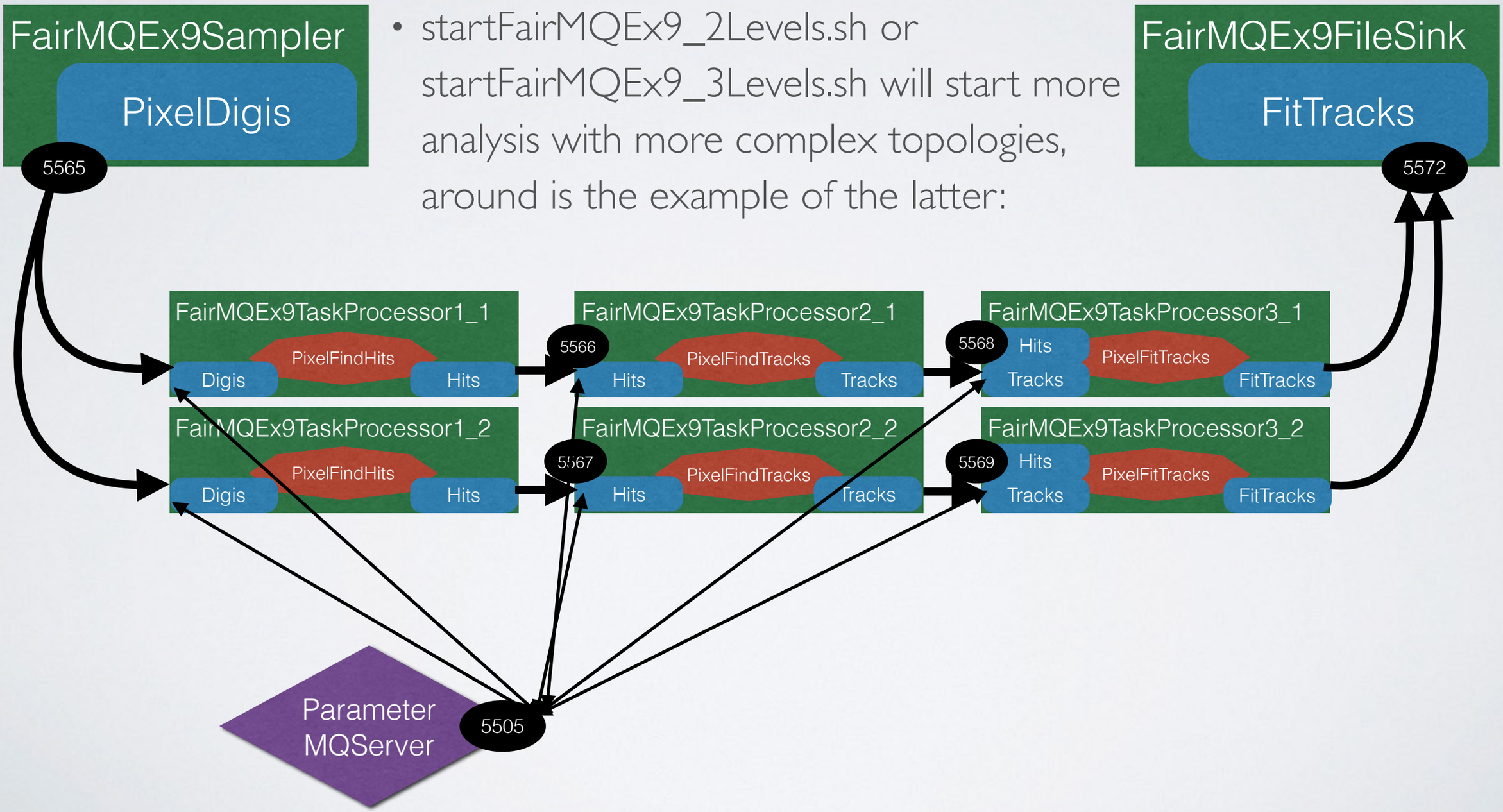
- Specifies program options that are not taken from the JSON file, or that are often changed;
- Starts all the programs from the topology at once
- Opens several xterm windows and starts one program per window;
- Example execution of the startFairMQEx9New.sh script results in:
- Several configurable shell script provided in the 9-PixelDetector/run/scripts; shell scripts located in build/bin and install/bin.



- startFairMQEx9New.sh - shell script to start sampler, few processors, sink and parameter server. The task to process may be easily changed by using --task-name PixelFindTracks or --task-name PixelFitTracks;

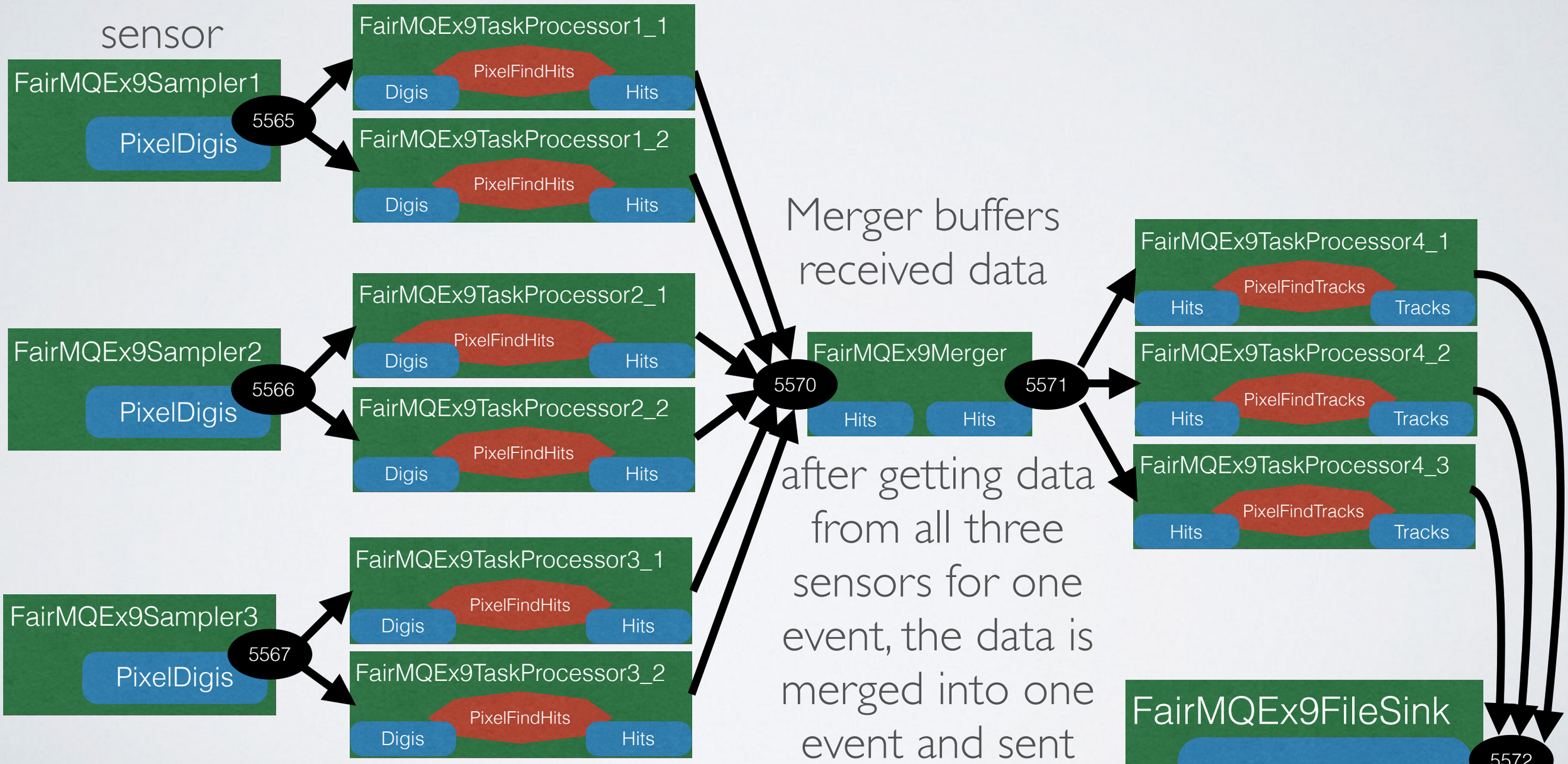
Other topologies:

- startFairMQEx9_2Levels.sh or startFairMQEx9_3Levels.sh will start more analysis with more complex topologies, around is the example of the latter:



Other topologies:

Each sampler reads from different file with data from one sensor



Merger buffers received data after getting data from all three sensors for one event, the data is merged into one event and sent

How to start?

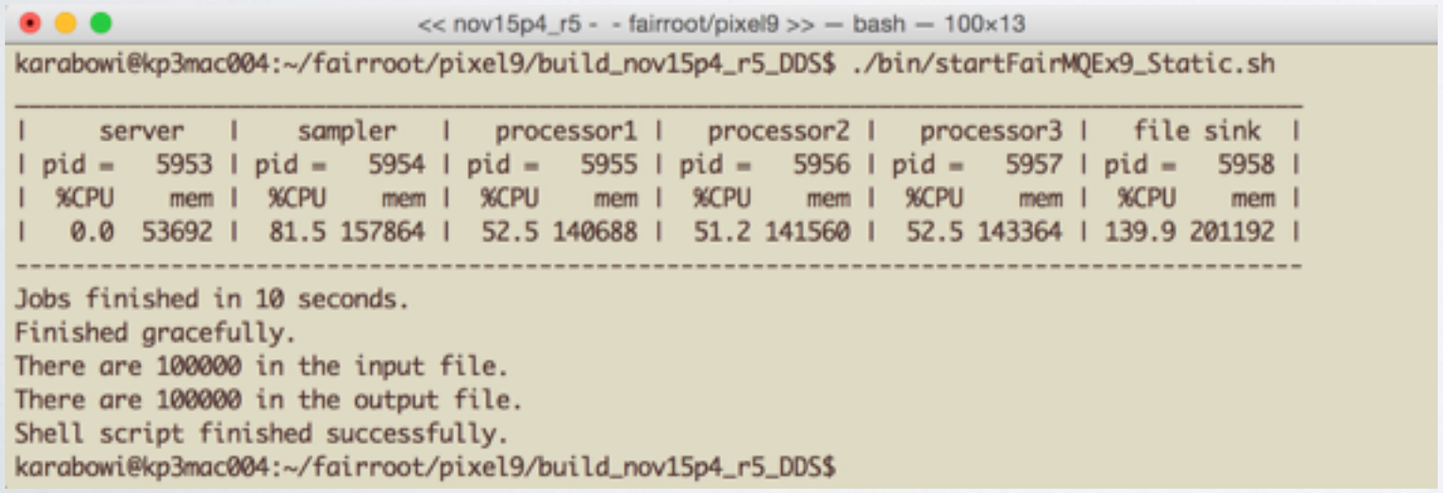
- Example available in the FairRoot dev branch:

  https://github.com/FairRootGroup/FairRoot/tree/dev

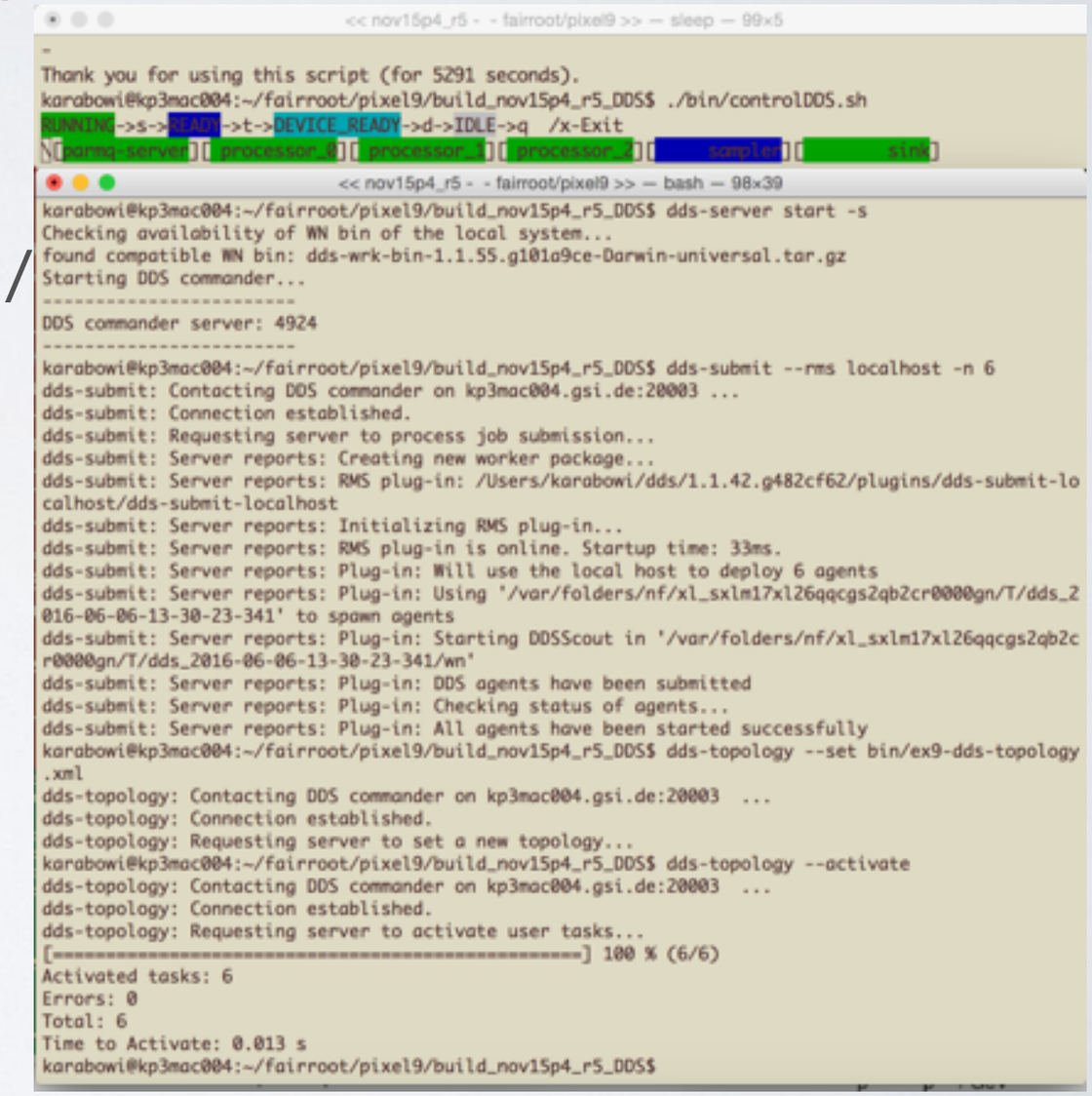
- Simulate and digitize data in

  `examples/MQ/9-PixelDetector/macros/`

- Run provided examples, f.e.:



run devices in background, monitor running, check output



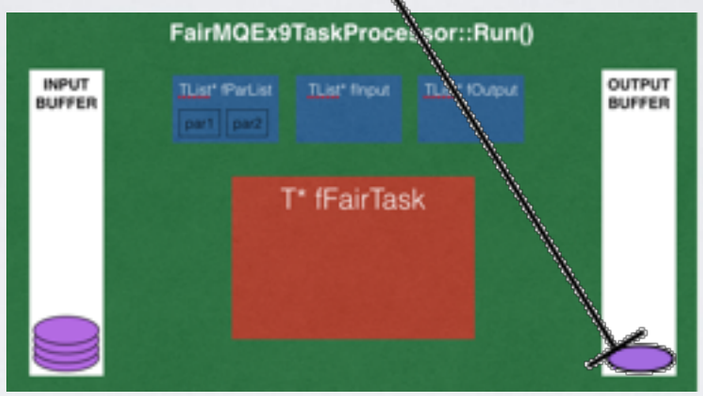
run and control devices with DDS

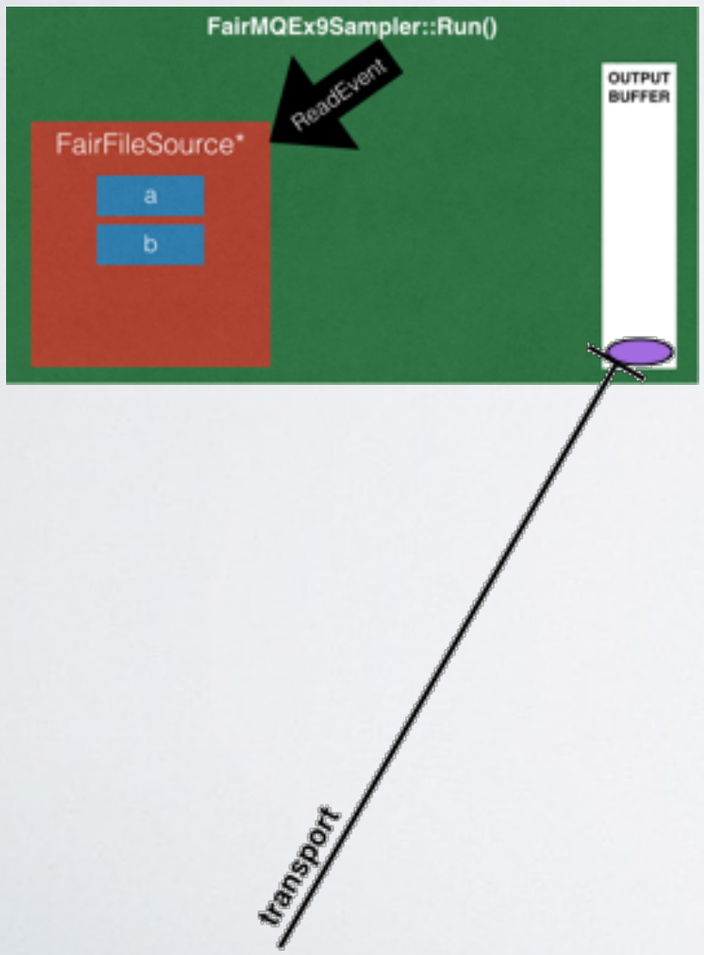
Summary

- Ready to use example implemented;
- The example devices are quite simple and yet general enough to allow processing of different tasks;
- Merger of splitted event data implemented;
- Binary transport available for selected data classes;
- The example may be run in DDS:

https://github.com/FairRootGroup/DDS

Thank you for attention





Special thanks to the **FairRoot** group:
Mohammad Al-Turany
Alexey Rybalchenko
Nicolas Winckler
Anar Manafov
Andrey Lebedev
Florian Uhlig
Dmytro Kresan
Thorsten Kollegger