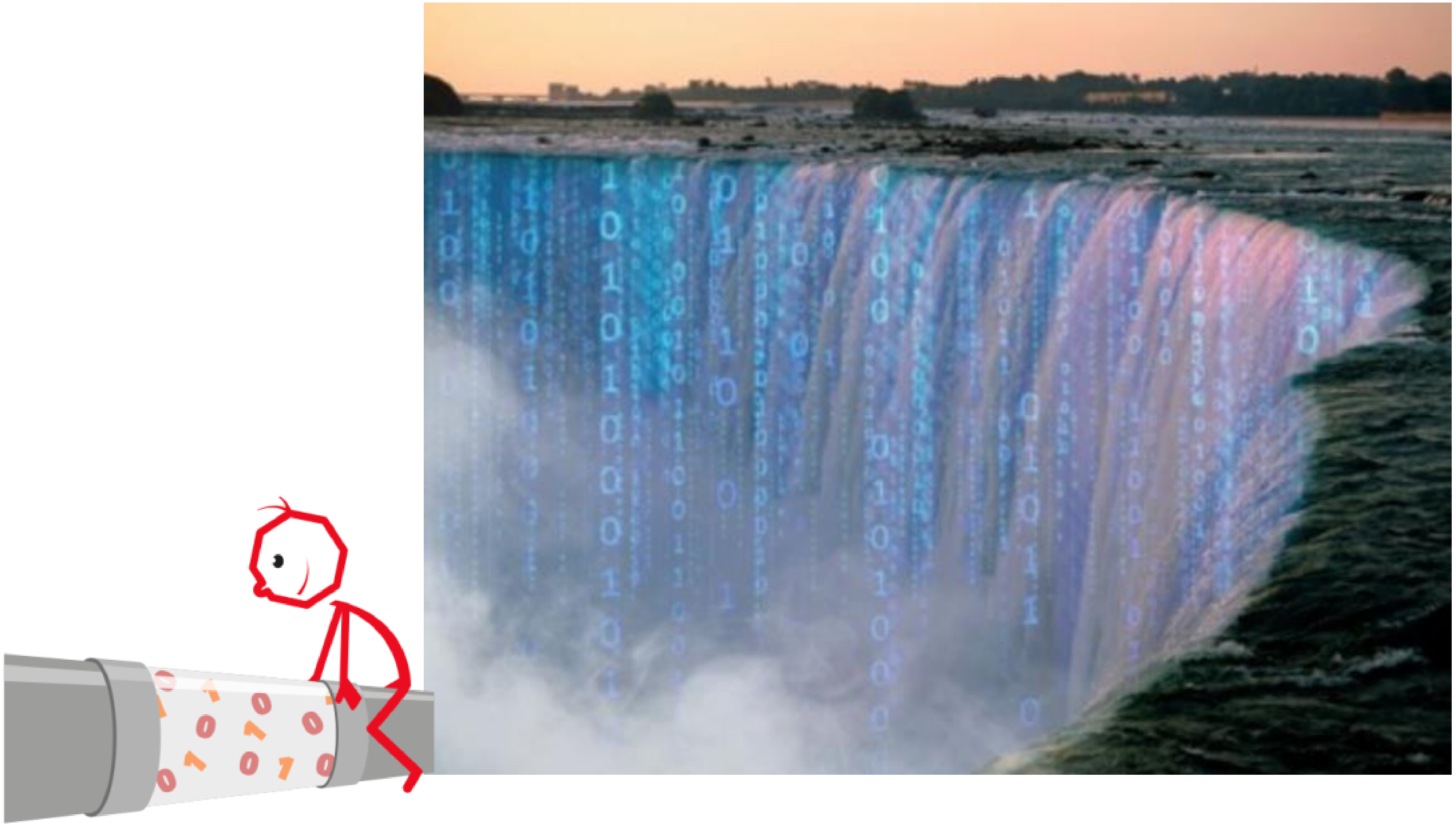
ALFA: A framework for building distributed applications

Mohammad Al-Turany, Alexey Rybalchenko, Dennis Klein, Matthias Kretz, Dmytro Kresan, Radoslaw Karabowicz, Andrey Lebedev, Anar Manafov, Thorsten Kollegger and Florian Uhlig

Developed in common by FairRoot Group (GSI), FAIR experiments and ALICE

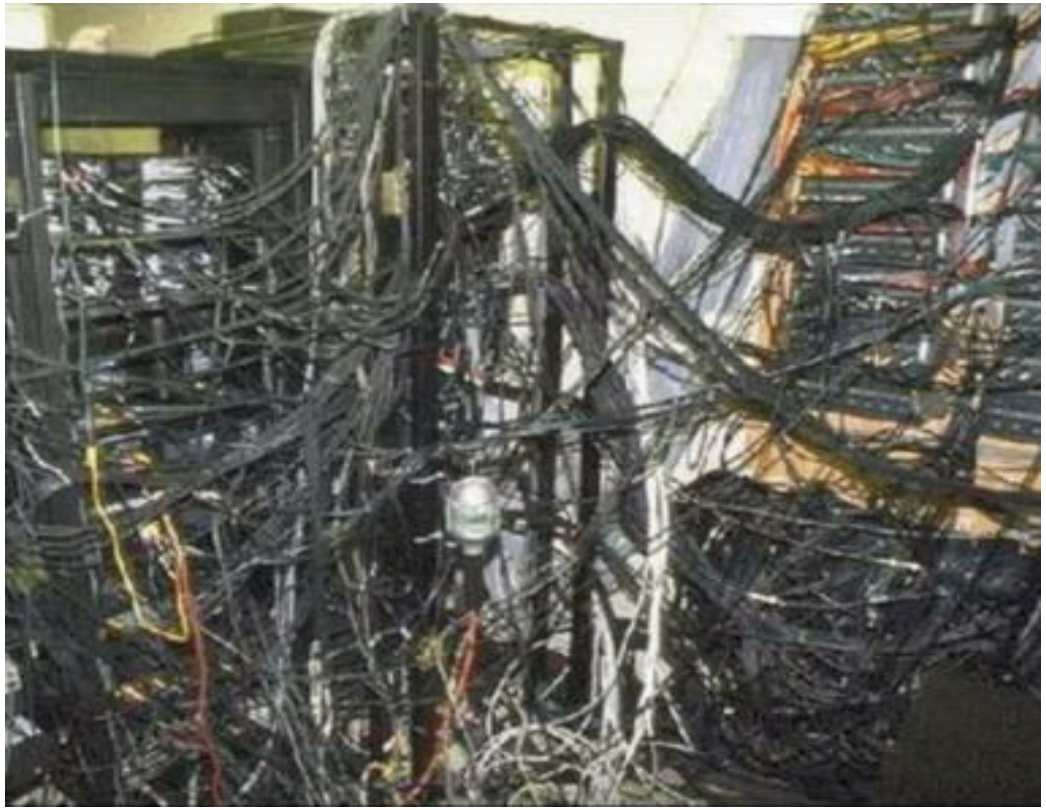
ALFA has a data-flow based model:



Message Queues based multi-processing

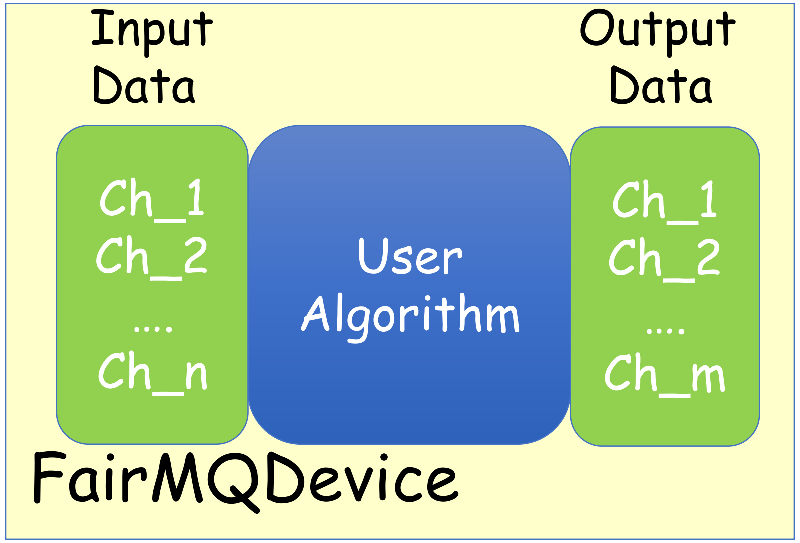
Works locally and across most networks!

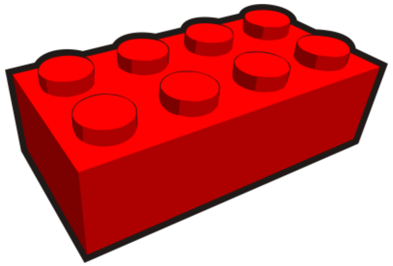
- Ethernet
  - ZMQ, nanomsg
- InfiniBand (IPoverIB, RDMA)
  - ZMQ, nanomsg, OFI
- Shared Memory Transport
  - Boost



ALFA building block (FairMQ Devices)

- Device takes/passes ownership of data
- Framework user sees only the callback to his algorithm
- Different channels can use different transport engines





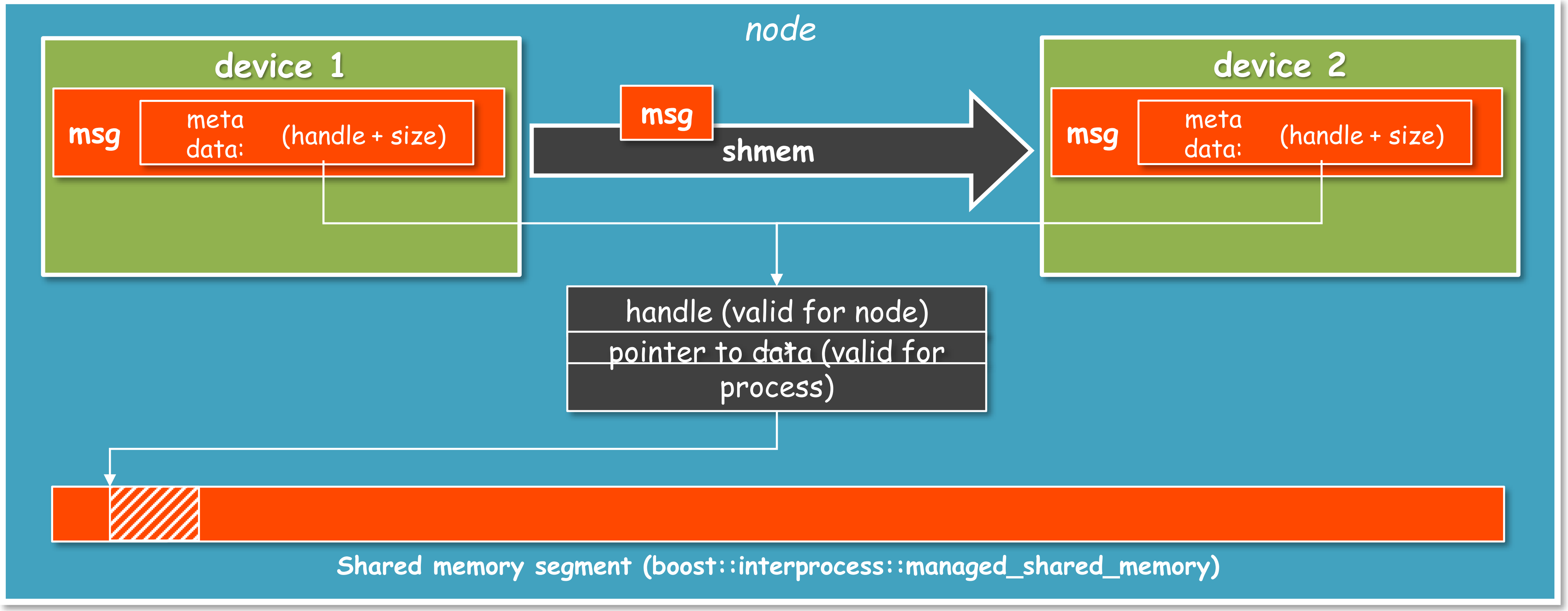
FairMQ Transport: General concepts:

- Hide all transport-specific details from the user.
- Clean, unified interface to different data transports.
- Combinations of different transport in one device in a transparent way.
- Transport switch via configuration only, without modifying device/user code -> same API for all transports.

FairMQ Transport: Ownership

- Message owns data.
- Sender device (user code) passes ownership of data to framework with send call.
- Framework transfers to next device, passes ownership to receiver (no physical copy of the data with shared memory transport).
- No sharing of ownership between different devices – if the same message is needed by more than one receiver it is copied.

FairMQ Shared Memory Transport



FairMQ Shared Memory Transport

Implementation

- boost::interprocess library for management and allocation of shared memory - cross-platform shared memory implementation with many features such as different allocation algorithms, shmem STL-like containers, shmem smart pointers, message queues and many more.

- ZeroMQ library for transfer of the meta information associated with the memory – allows us to reuse communication patterns of ZeroMQ (PUSH/PULL, PAIR, REQ/REP) and offers higher performance than boost::interprocess::message_queue.

FairMQ Shared Memory Transport

Features

- PAIR, PUSH/PULL, REQ/REP communication patterns
- Support for multipart messages
- Managed shared memory that is completely transparent for the user.

Example: Time frame in Alice O2 data model

*Headers defines the type of data. Different header types can be stacked to store extra metadata (mimicking a Type hierarchy structure). Headers and payloads are usable in a **message passing** environment.*

FairMQ Shared Memory Transport

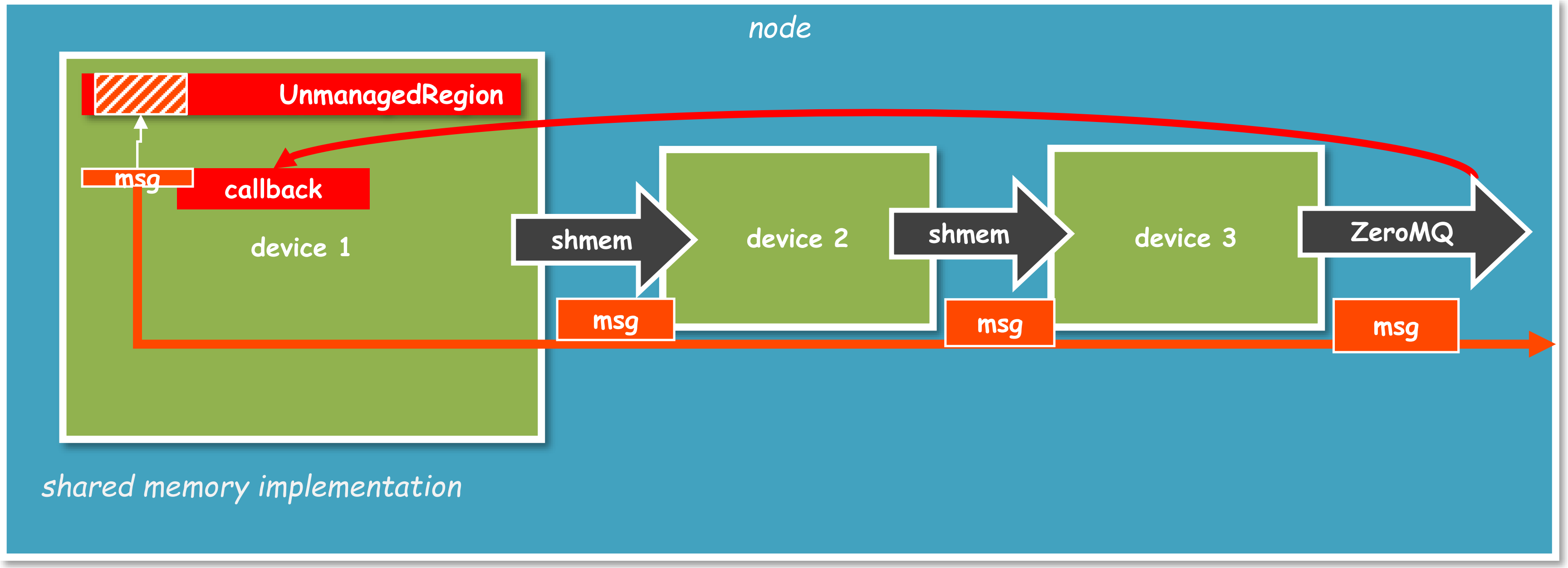
Features

- Automatic cleanup of orphan shared memory in case of device crashes. Optionally a cleanup/monitoring/debug tool provided for more control.
- Seamless integration with other transports – no copies of data between different transports (for transports that allow adopting foreign data buffer)
- Very high performance – transfer rates in high kHz/low MHz range, low CPU usage.

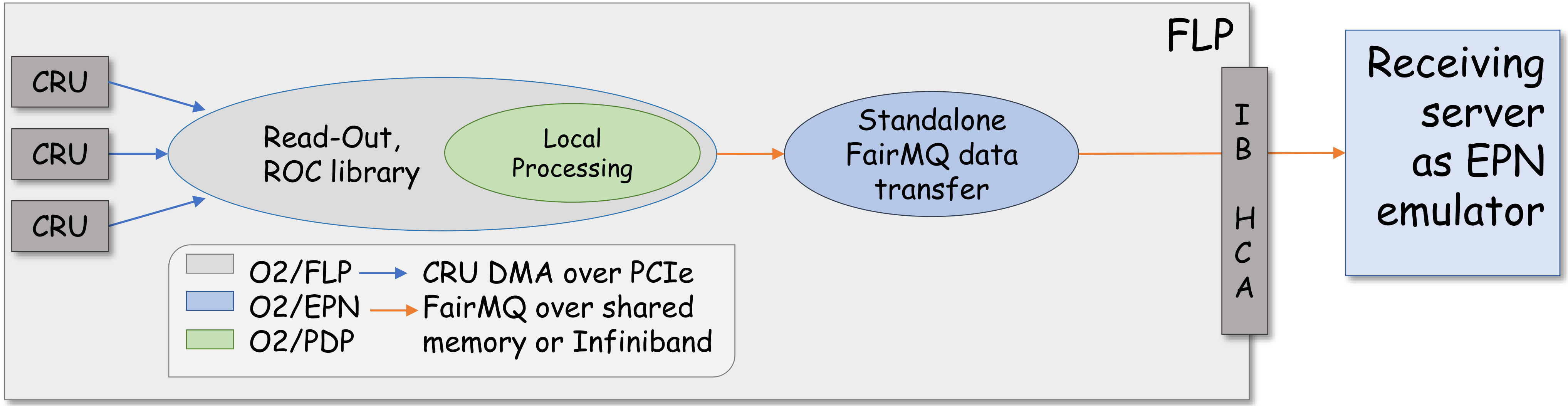
FairMQ Shared Memory Transport

Features

- Unmanaged shared memory regions for fine-grained control of buffer location and handling.



FairMQ for ReadOut in ALICE



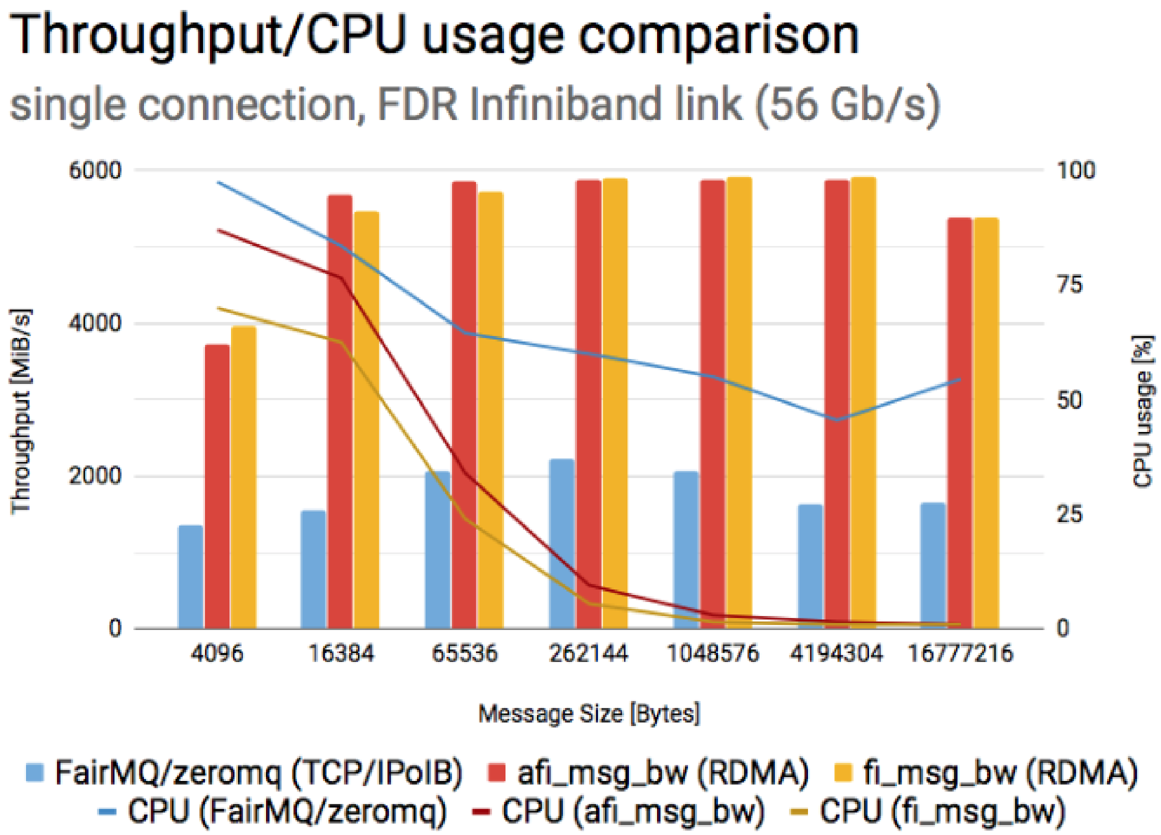
- CRU test data, TPC decoder algorithm integrated in Readout
- Demonstrate usage of available CPU resources at target data throughput

Run chain for 8 hours, use as much CPU as possible at target data throughput

SUCCESS: # CRUs x 17.25 Gb/s with Local Processing active

Not only TCP/IP but also RDMA

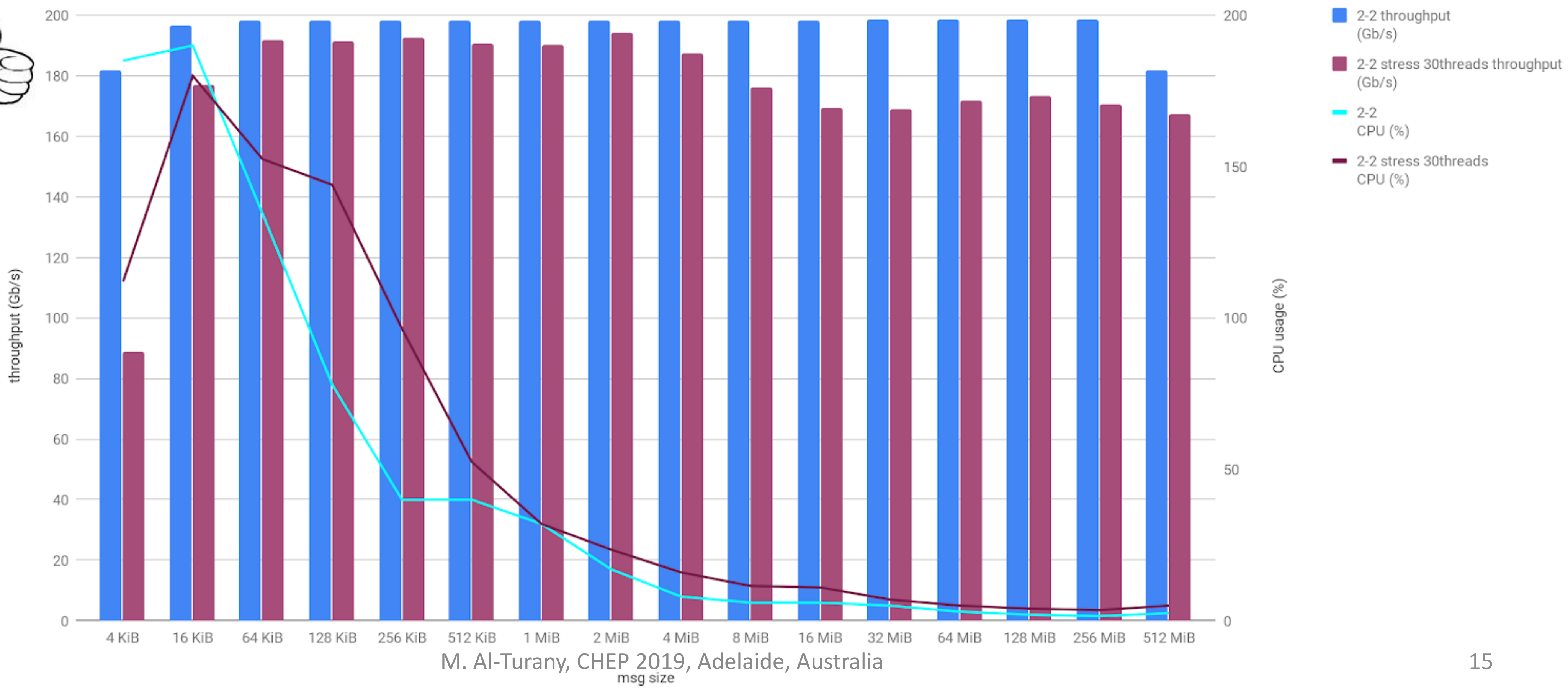
High data throughput (>90% link capacity) and significantly reduced CPU load



afi_msg_bw: Benchmark in asiofi (base for new FairMQ transport)
fi_msg_bw: Benchmark from Libfabric

Tests on 200Gb/s IB in Feb'19

- Hardware setup provided by CBM/FIAS (Mellanox engineering sample)



FairMQ OFI Transport

Features

- Available in : FairMQ v1.4.9 + asiofi v0.4.3
- About 90% of the theoretical throughput is achieved on experimental systems:
  - CBMfles: 97 of max 107 Gb/s IB
  - Alice: 60 of max 65 Gb/s RoCE
  - Alice: 80-90 of max 100 Gb/s IB

- Optimizing the implementation to utilize the last 10% of available bandwidth is ongoing

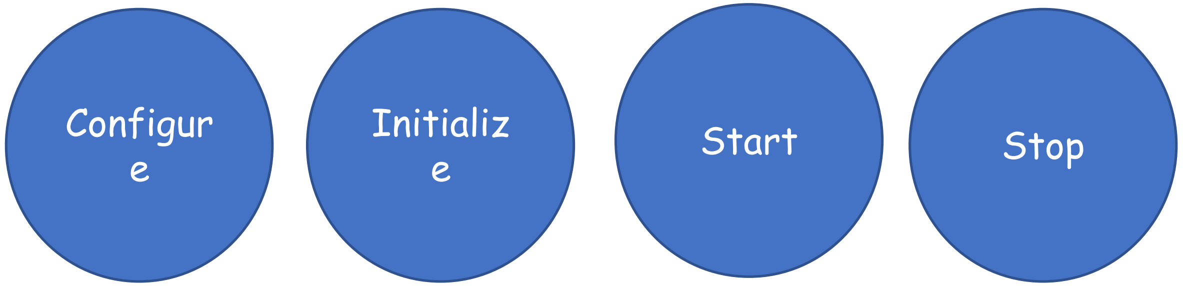
Controlling FairMQ state machine

Controlling FairMQ state machine: on one device:

- The FairMQ core library provides two device controllers
  - Static : a fixed sequence of state transitions
  - Interactive: a read-eval-print-loop which reads keyboard commands from standard input
- A device controller only knows how steer a single FairMQ device (i.e: it runs in a thread within the device process)

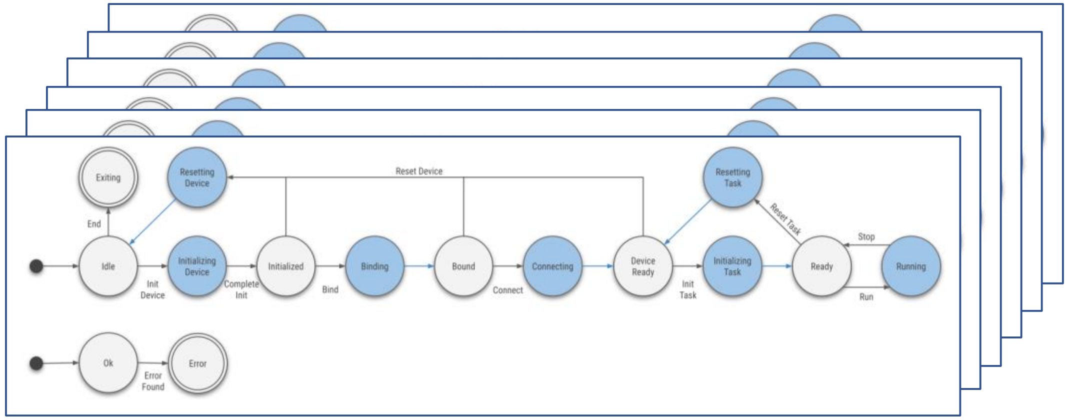
Controlling FairMQ state machine: on a processing farm

- One has to make the entire cluster state available for the experiment control system and not single process one

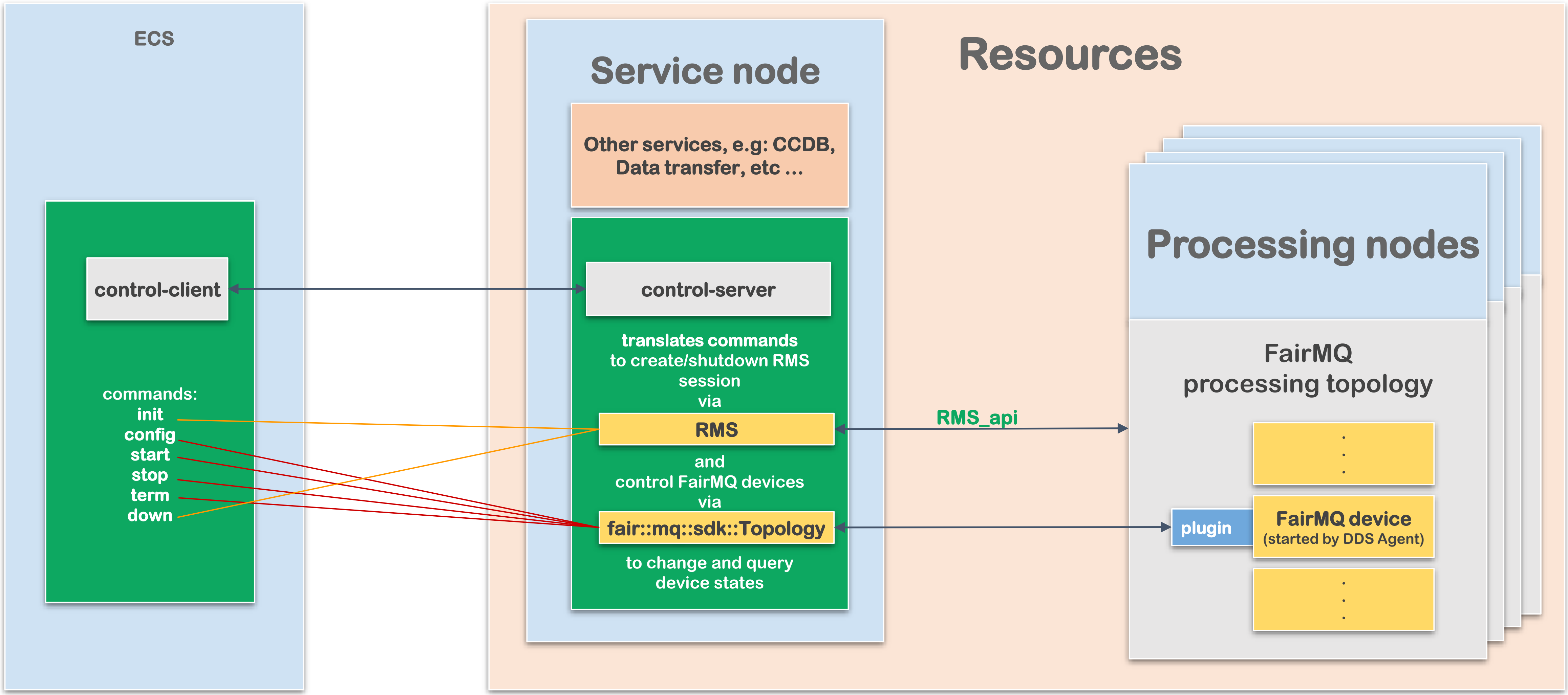


Exported cluster state machine

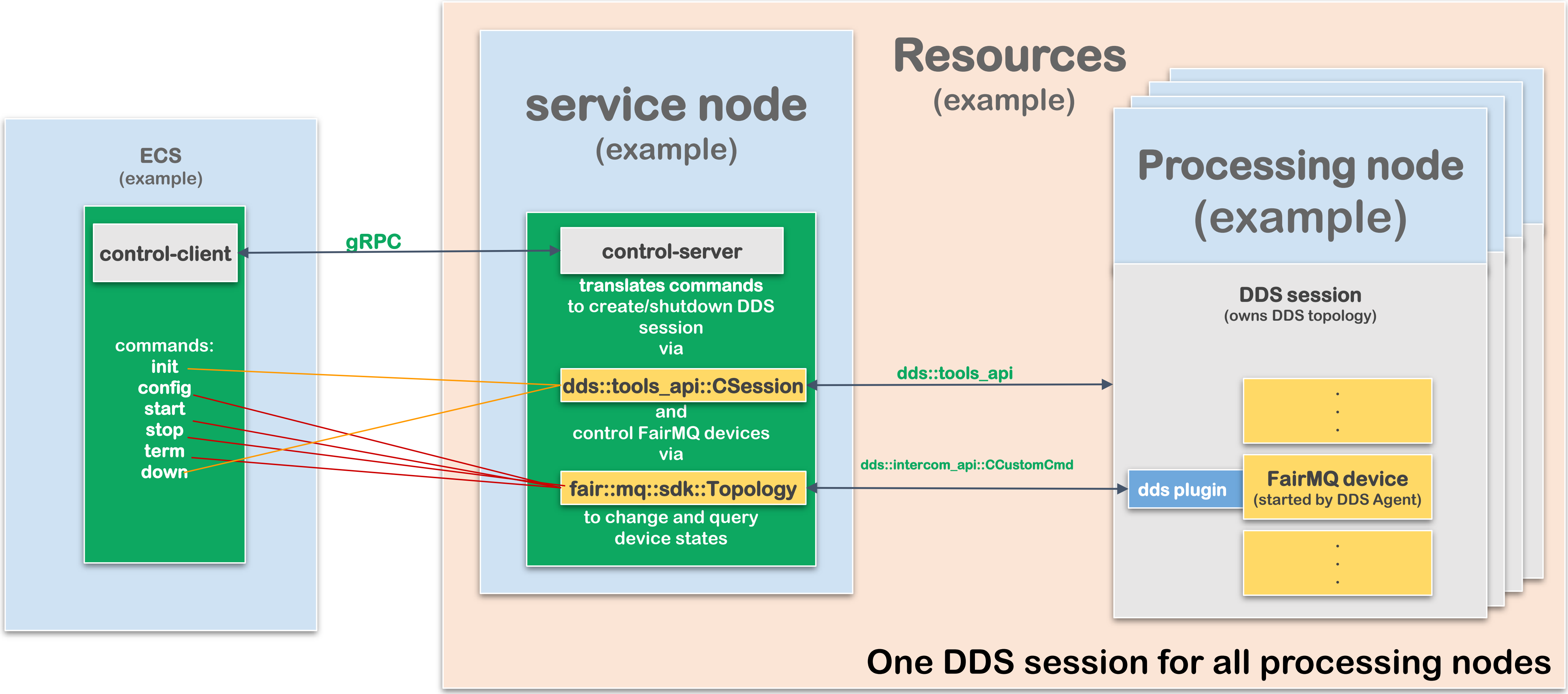
EPNs internal state machine (FairMQ)



Controller Design

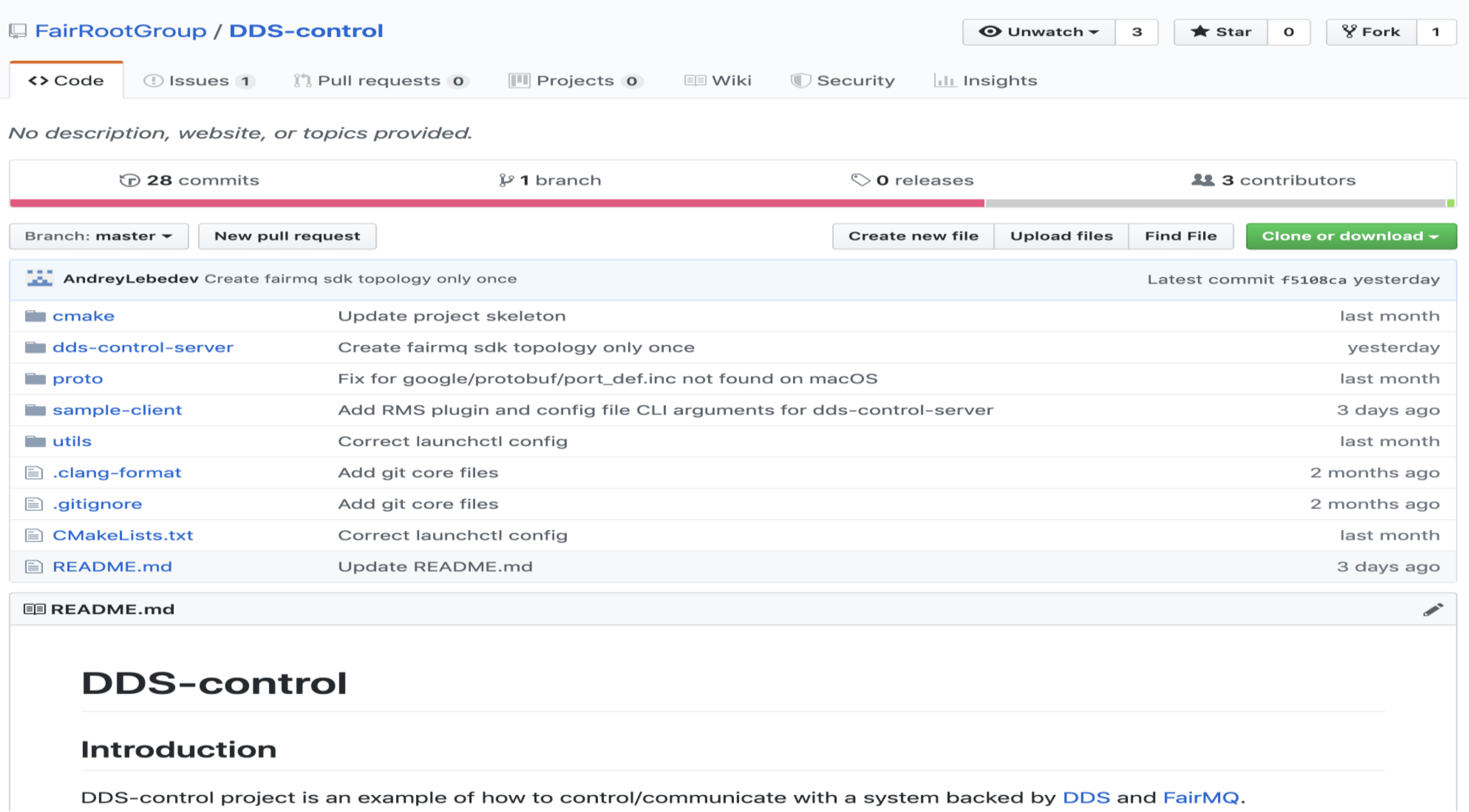


Controller example (DDS based)



DDS-control

An example of how to control/communicate with a system backed by DDS and FairMQ.



https:// github.com/ FairRootGroup/ DDS-control

```
--   DEPENDENCY FOUND     VERSION                    PREFIX
--   Boost                1.69 (>= 1.67)             /usr
--   DDS                  2.5.5.g3aa26d5 (>= 2.4)    /home/dklein/projects/DDS/build/install
--   FairLogger           1.5.0 (>= 1.2.0)           /home/dklein/projects/FairLogger/build/install
--
--   COMPONENT          BUILT?  INFO
--   fairmq               NO    (enable with -DBUILD_FAIRMQ=ON)
--   tests                NO    (enable with -DBUILD_TESTING=ON)
--   nanomsg_transport    NO    (default, enable with -DBUILD_NANOMSG_TRANSPORT=ON)
--   ofi_transport        NO    EXPERIMENTAL (requires C++14) (default, enable with -DBUILD_OFI_TRANSPORT=ON)
--   dds_plugin           NO    (default, enable with -DBUILD_DDS_PLUGIN=ON)
--   pmix_plugin          NO    (default, enable with -DBUILD_PMIX_PLUGIN=ON)
--   examples             NO    (enable with -DBUILD_EXAMPLES=ON)
--   docs                 NO    (default, enable with -DBUILD_DOCS=ON)
--   sdk                 YES    (disable with -DBUILD_SDK=OFF)
```

libFairMQ_SDK.so

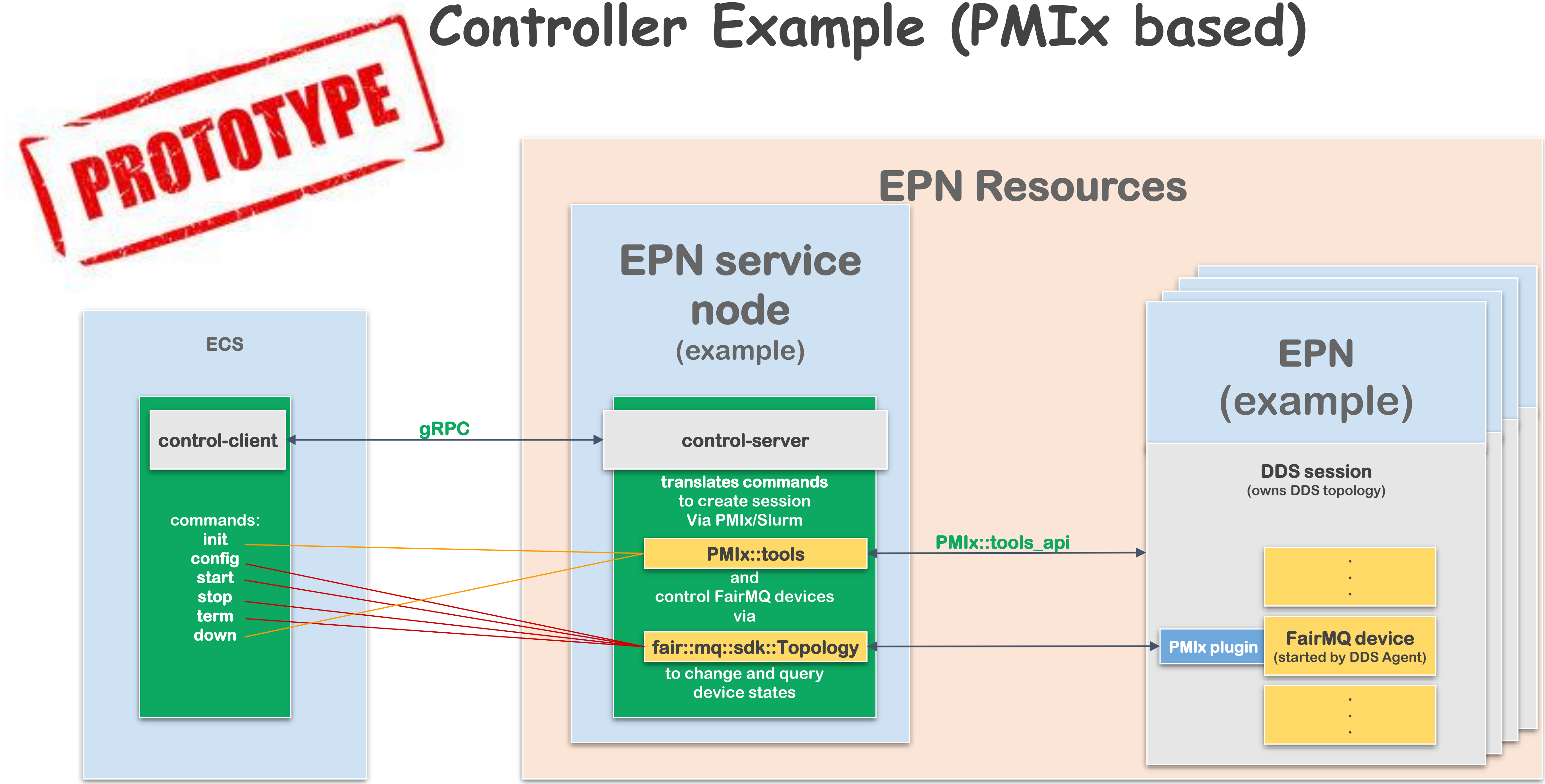
find_package(FairMQ COMPONENTS sdk)

Then link against

FairMQ::SDK

#include <fairmq/SDK.h>

Controller Example (PMIx based)



Summary

- *ALFA allows developers to write their specific code in whatever language they choose as long as that language can send and receive data through message queues.*
- allows non-expert to write messaged based code without going into the details of the transport or the system below
- offers a **clean** and **maintainable** and **extendable** interface to the existing different data transport (ZMQ, nanomsg, shared Memory, OFI, ..etc)
- provides utilities to deploy and control topologies on computing clusters, online clusters as well as on a laptop

Backup

asiofi (C++ Boost.Asio language bindings for OFI libfabric)

- The asiofi library provides a C++ Boost.Asio interface to OpenFabric Interface's libfabric and is used to implement the FairMQ OFI transport.

https://github.com/FairRootGroup/asiofi

PMIx (Process Management Interface for Exascale)

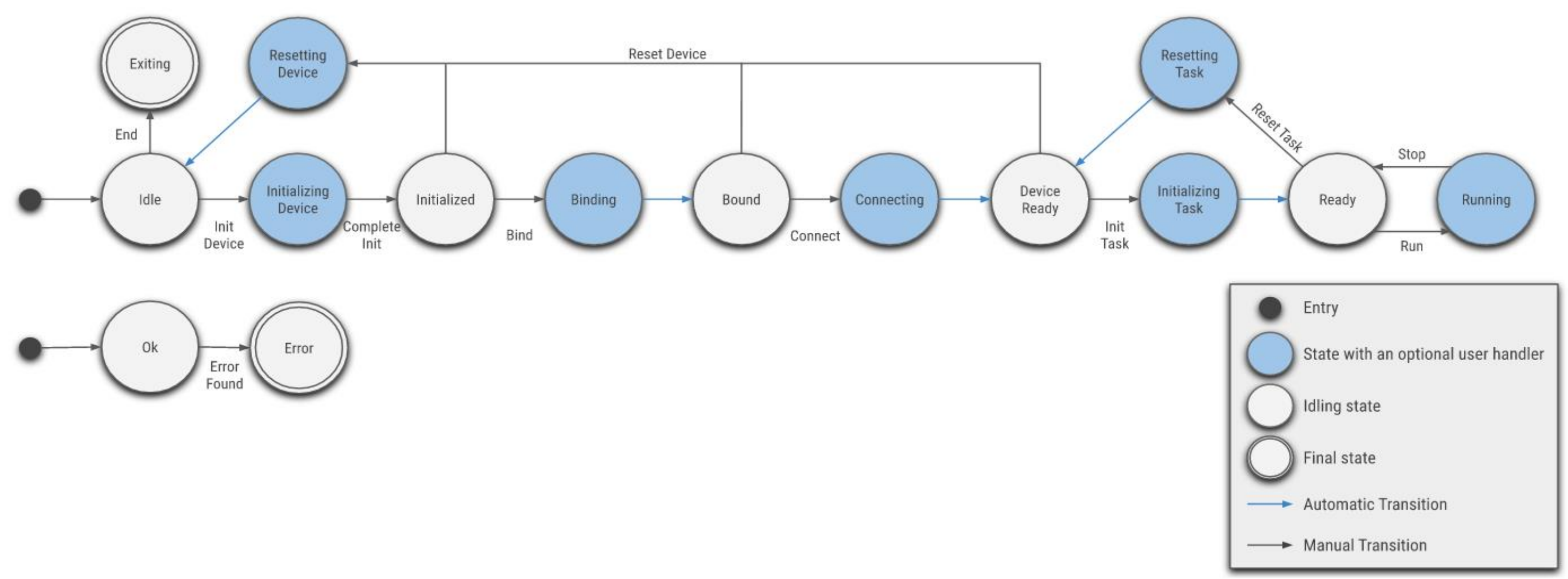
- Originally developed and distributed as part of MPICH, has historically been used as a means of exchanging wireup information needed for interprocess communication and deployment of processes
  - Distributed key/value store for data exchange
  - Asynchronous events for coordination
  - Enable interactions with the resource manager
- PMIx also covers: Resource allocation, process/job mgmt (creation/deletion/monitoring), system information, error notifications

- PMIx provides server, tool, and client APIs

https://github.com/pmix/pmix
https://github.com/pmix/pmix-standard

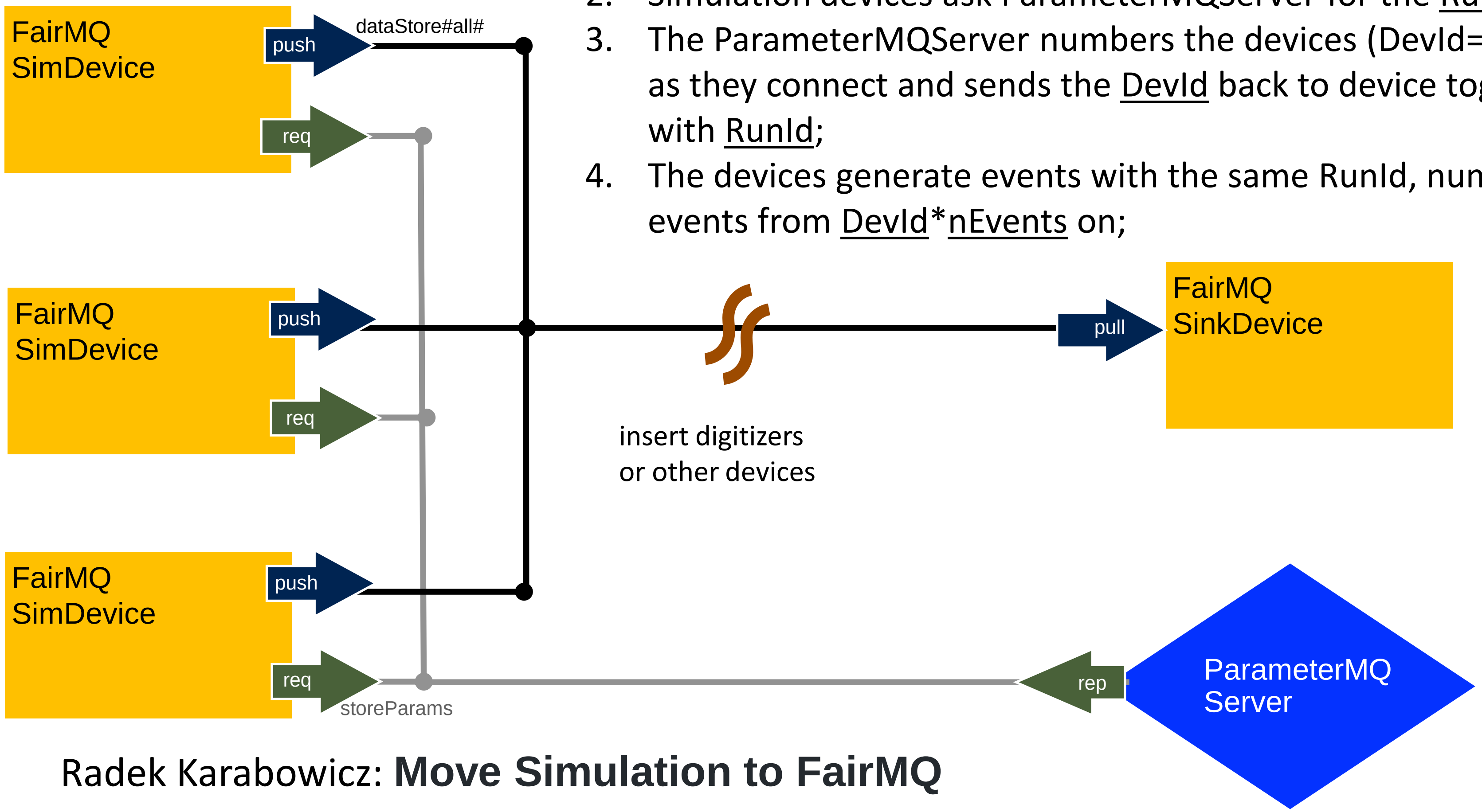
FairMQ State Machine & Example ECS Command Mapping

ECS command	DDS/FairMQ actions
init	DDS: Create session, submit agents, activate topology -> devices go in Idle state
configure	Devices: InitDevice->CompleteInit->Bind->Connect->InitTask
start	Devices: Run
stop	Devices: Stop
term	Devices: ResetTask->ResetDevice->End
down	DDS: Shutdown session



Distributed Simulation with FairMQ

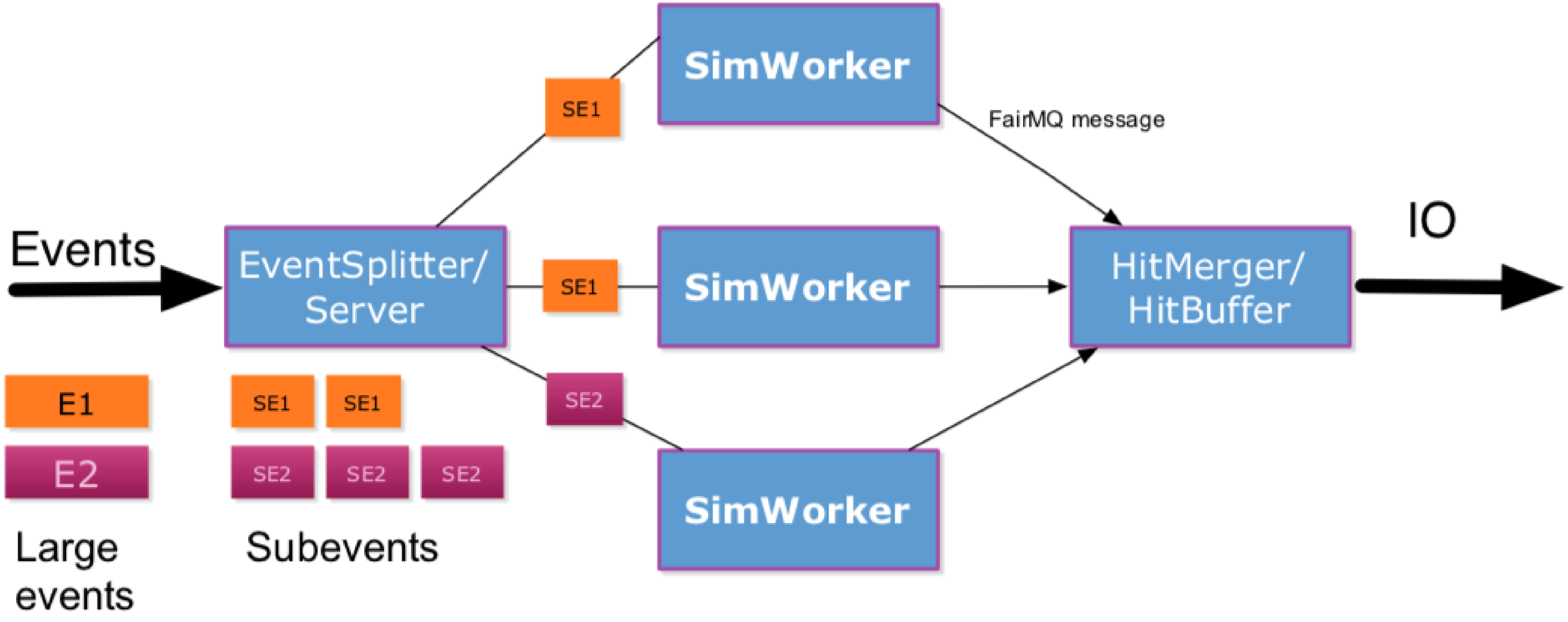
1. Each simulation device will generate nEvents;
2. Simulation devices ask ParameterMQServer for the RunId;
3. The ParameterMQServer numbers the devices (DevId=0,1,2,…) as they connect and sends the DevId back to device together with RunId;
4. The devices generate events with the same RunId, numbering events from DevId*nEvents on;



Radek Karabowicz: **Move Simulation to FairMQ**

https://github.com/FairRootGroup/FairRoot/tree/dev/examples/MQ/pixelDetector

FairMQ-based parallel simulation



Sandro Wenzel