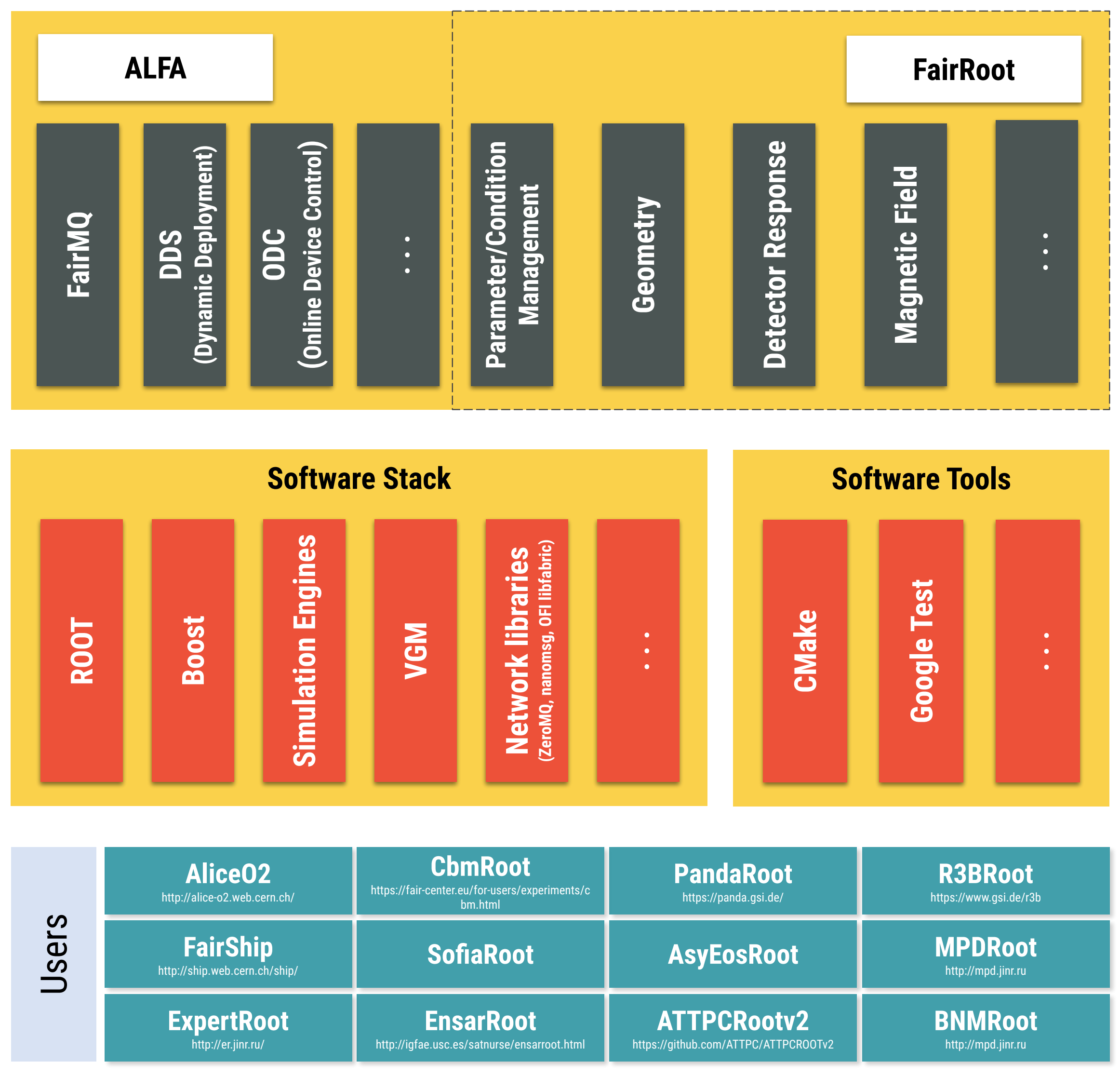
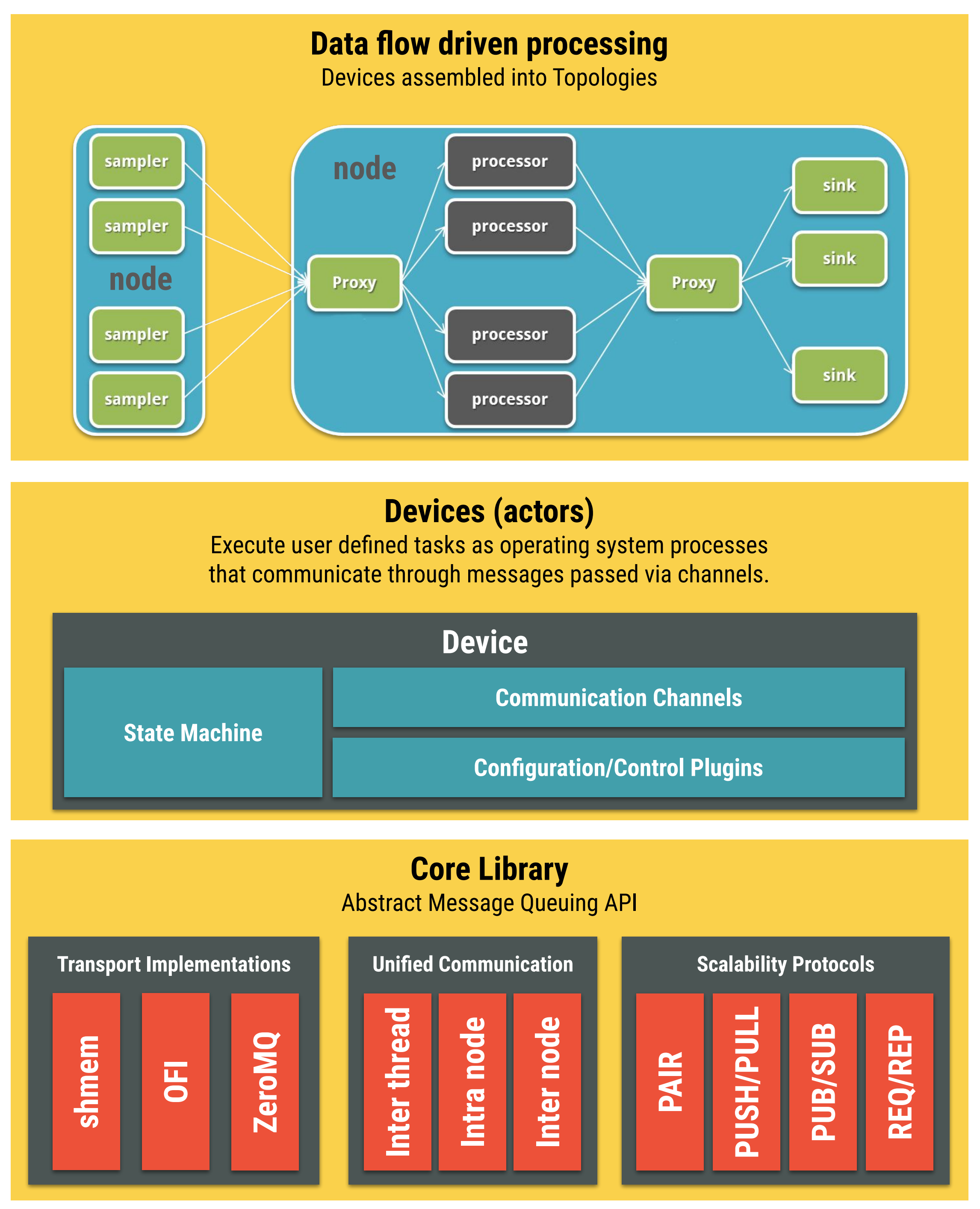
Distributed Data Processing Pipelines in ALFA

Alexey Rybalchenko, Dennis Klein, Mohammad Al-Turany
GSI Helmholtz Centre for Heavy Ion Research GmbH, Darmstadt, Germany


ALFA
FairRoot
FairMQ
DDS (Dynamic Deployment)
ODC (Online Device Control)
...
Parameter/Condition Management
Geometry
Detector Response
Magnetic Field
...
Software Stack
ROOT
Boost
Simulation Engines
VGM
Network libraries (ZeroMQ, nanomsg, OFI libfabric)
...
Software Tools
CMake
Google Test
...
Users
AliceO2 http://alice-o2.web.cern.ch/
CbmRoot https://fair-center.eu/for-users/experiments/cbm.html
PandaRoot https://panda.gsi.de/
R3BRoot https://www.gsi.de/r3b
FairShip http://ship.web.cern.ch/ship/
SofiaRoot
AsyEosRoot
MPDRoot http://mpd.jinr.ru
ExpertRoot http://er.jinr.ru/
EnsarRoot http://igfae.usc.es/satnurse/ensarroot.html
ATTPCRootv2 https://github.com/ATTPC/ATTPCROOTv2
BNMRoot http://mpd.jinr.ru

ALFA[1] is a modern C++ software framework for simulation, reconstruction and analysis of particle physics experiments. ALFA extends FairRoot[2] to provide building blocks for highly parallelized and data flow driven processing pipelines required by the next generation of experiments, such as the ALICE experiment at CERN or the FAIR experiments.

FairMQ[3] is a C++ message queuing framework and library that integrates standard industry data transport technologies and provides building blocks for creation of data flow actors and pipelines. FairMQ hides transport details behind an abstract interface and ensures best utilization of the underlying transports (zero-copy, high throughput). The framework does not impose any format on the messages.



Core FairMQ Concepts

Library concepts:

- Main library concepts are **Message** and **Channel** which provide a **message queuing APIs** (very similar to ZeroMQ) for *inter/intra node/process* communication.

Framework concepts:

- Main framework concept is a **Device** which composes **State Machine**, **Program Options**, and **Command/Configuration Plugins**.
- Devices are composed into **Topologies** that belong to a **Session** which isolates them.
- The framework part is a thin and opinionated compositional layer. However, a user could as well **pick and compose any subset** of available FairMQ concepts and create his framework entity that represents a actor (e.g. task, process) in the online processing graph.

Ownership concepts:

- A Message and its content has **single ownership** by the process it is created/received in.
- Transport may choose not to physically copy the buffer, but to share across the messages. Modifying the buffer after a call to the copy is undefined behaviour.

General goals:

- Unified API to different data transports.
- Hide transport-specific details from the user.
- Allow to transparently combine different transport in one device.
- Transport switch via configuration, without modifying user code.

Adoption Example

ALICE Experiment

The upgraded ALICE experiment at CERN deploys FairMQ both on the level of First Level Processors (FLPs), which include Readout and local data processing, as well as on the level of Event Processing Nodes (EPNs), where data aggregation and synchronous global reconstruction take place.

Deployments on EPNs involve up to ~70000 running FairMQ devices per session, on ~200 nodes. Single node handling ~350 FairMQ devices communicating via shared memory. Topologies are deployed and controlled via ODC through DDS and Slurm.

For the final output data rates of the upgraded experiment, this is expected to increase to 1500 nodes.

FairMQ Transports

FairMQ provides three transport implementations:

- **TCP Network transport based on ZeroMQ[4].**
- **RDMA Network transport based on OFI[5].**
- **Shared memory transport for interprocess communication[6], based on ZeroMQ and boost::interprocess library[7].**

node	process	address format	ZeroMQ	shmem	OFI
intra-	intra-	`inproc://endpoint`	✓	n/a	n/a
intra-	inter-	`ipc://endpoint`	✓	✓	X
		`tcp://host:port`	✓	✓	✓
inter-	inter-	`tcp://host:port`	✓	n/a	✓
		`verbs://host:port`	X	n/a	✓

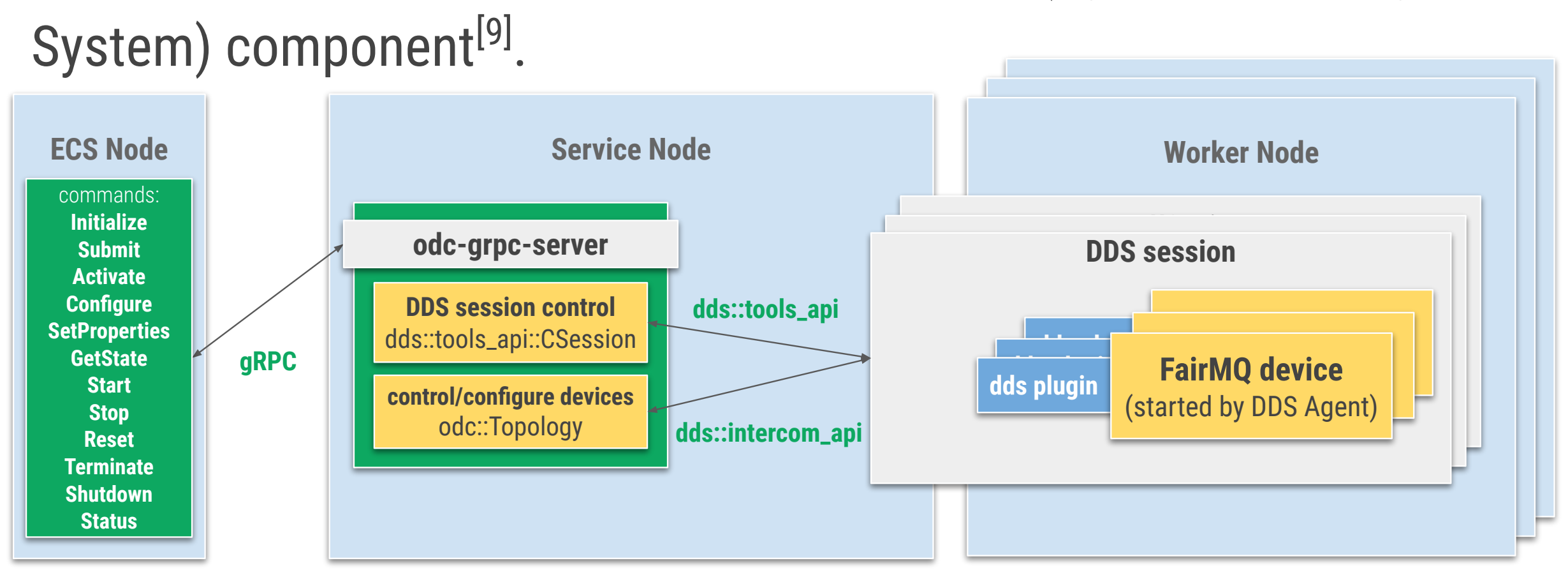
✓ - zero-copy ✓ - not zero-copy X – no support planned

Scalability protocols, inspired by ZeroMQ, provide several communication patterns:

Scalability Protocol	ZeroMQ	shmem	OFI
pair (one-to-one)	✓	✓	✓
push/pull	✓	✓	X
pub/sub	✓	X	X
req/rep	✓	✓	X

Device Control

A topology of FairMQ devices can be launched and controlled via the **ODC** (Online Device Control) component[8]. ODC acts as a command broker between an Experiment Control System and one or many Topologies of FairMQ devices. ODC shows a homogeneous topology state to the ECS.
The task of process deployment and command exchange between them is implemented via APIs of the **DDS** (Dynamic Deployment System) component[9].



ODC provides a **gRPC server** component and a sample **gRPC client**. A fixed set of commands is available to launch deployments with FairMQ topologies, control device states and configure device properties. A single server process can handle multiple DDS/FairMQ sessions in parallel and reconnect to those which are running prior to server launch.

A simple command line server is also provided that can be used for testing without gRPC.

Resource management is delegated to the resource managers supported by DDS. ALICE online deployment runs with Slurm.

New Shared Memory Features

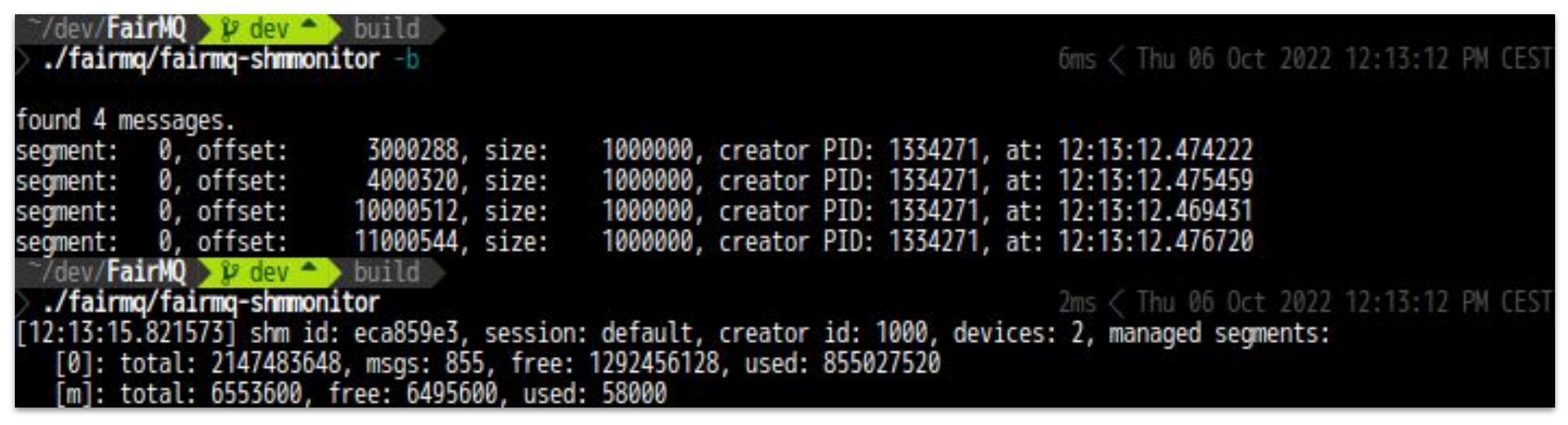
Since the introduction of the Shared Memory transport in [5] several new features have been added:

- **Shared memory event notifications**: Device can subscribe to region creation/destruction events in order to obtain region address, size and settings prior to data transfer. This can be used, for example, to register the memory areas for use with **GPU** or RDMA
- **Buffer shared ownership**: A performance optimization where calling a Copy() method of the message class would create additional message object for the same physical memory buffer. Can be used, for example, when multiple processes need to access the same buffer in parallel. However, modifying such buffers is undefined behaviour.
- **Support for custom alignment for message buffers**.
- **Externally created regions**. Creating (and on demand quickly resetting) shared memory outside of a session with a unique identifier. This can be helpful when a long memory registration process is involved, which in this case can be done only once, while keeping (and resetting) the memory between sessions. No two sessions can use it in parallel however.

Debugging & Monitoring

Shared memory transport provides a tool to monitor and debug the status of shared memory and existing messages. The tool provides data for a given FairMQ session, such as:

- Used shared memory segments
- Their total and used size
- Info on creator process
- Number of devices in the session
- List of messages in a session, with their size, creator ID, memory location, creation timestamp (available in debug mode)



```
~/dev/FairMQ  dev ^  build
> ./fairmq/fairmq-shmmonitor -b                                   6ms < Thu 06 Oct 2022 12:13:12 PM CEST

found 4 messages.
segment:   0, offset:      3000288, size:    1000000, creator PID: 1334271, at: 12:13:12.474222
segment:   0, offset:      4000320, size:    1000000, creator PID: 1334271, at: 12:13:12.475459
segment:   0, offset:     10000512, size:    1000000, creator PID: 1334271, at: 12:13:12.469431
segment:   0, offset:     11000544, size:    1000000, creator PID: 1334271, at: 12:13:12.476720
~/dev/FairMQ  dev ^  build
> ./fairmq/fairmq-shmmonitor                                      2ms < Thu 06 Oct 2022 12:13:12 PM CEST
[12:13:15.821573] shm id: eca859e3, session: default, creator id: 1000, devices: 2, managed segments:
   [0]: total: 2147483648, msgs: 855, free: 1292456128, used: 855027520
   [m]: total: 6553600, free: 6495600, used: 58000
```

These debugging features are also provided via an API for integration in user software. This can be especially interesting for message object information - knowing the data types contained in the buffers can help to extract additional debugging info, e.g. from data headers.

References:
[1] M. Al-Turany et al. - "ALFA: The new ALICE-FAIR software framework", Journal of Physics: Conference Series, Volume 664 (2015)
[2] https://github.com/FairRootGroup/FairRoot
[3] https://github.com/FairRootGroup/FairMQ
[4] http://zeromq.org/
[5] D. Klein et al. - "RDMA-accelerated data transport in ALFA", EPJ Web of Conferences 214 (2019) CHEP 2018
[6] A. Rybalchenko et al. - "Shared Memory Transport for ALFA", EPJ Web of Conferences 214 (2019) CHEP 2018
[7] https://www.boost.org/doc/libs/1_67_0/doc/html/interprocess.html
[8] https://github.com/FairRootGroup/ODC
[9] A. Lebedev and A. Manafov - "DDS: The Dynamic Deployment System", EPJ Web of Conferences 214 (2019) CHEP 2018