FairMQ Status

Data Transport for Online & Offline Processing

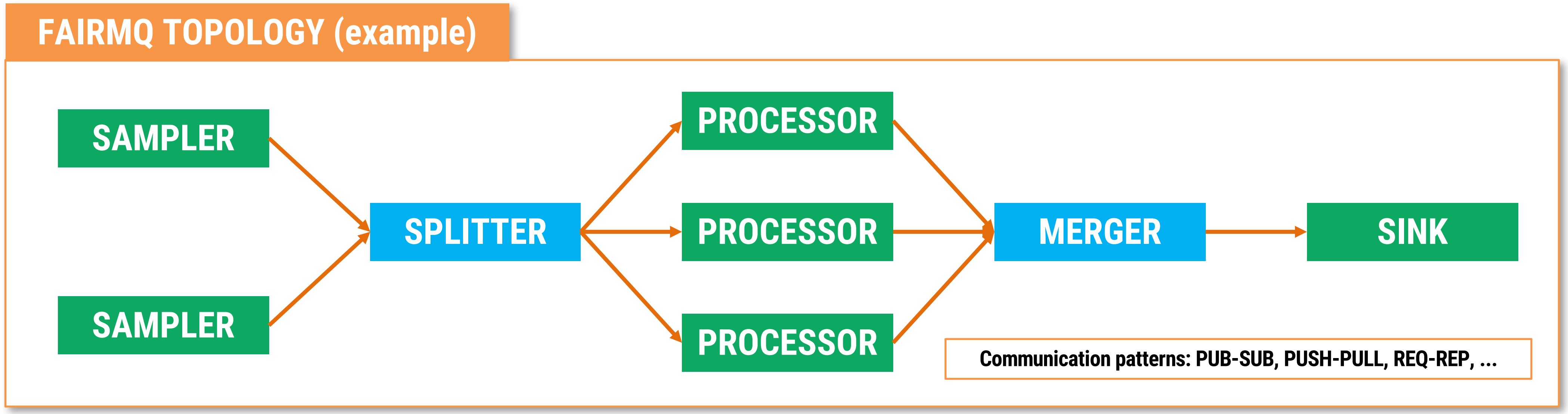
Alexey Rybalchenko
(GSI Darmstadt, FairRoot group)

PANDA Collaboration Meeting
GSI, December 6, 2016

What is FairMQ?

Organize processing tasks in topologies, consisting of independent processes (Devices), that communicate via *asynchronous message queues* over network or inter-process.

Ethernet, InfiniBand (IP-over-IB)



Communication patterns: PUB-SUB, PUSH-PULL, REQ-REP, ...

Ready to use devices are provided for typical scenarios.
User-defined devices can be implemented by inheriting from `FairMQDevice`.

FairMQ Status

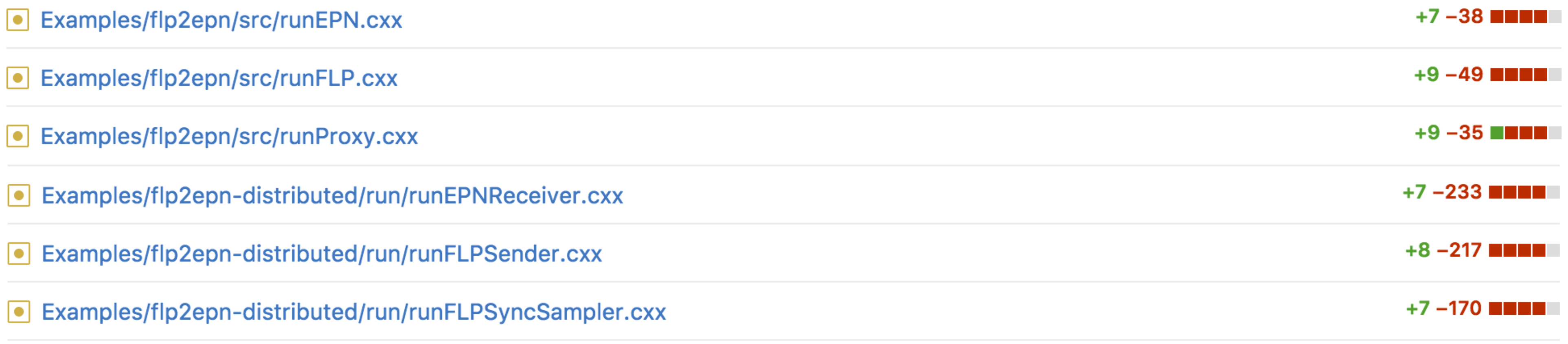
- Reducing the amount of boilerplate code needed to create a device executable and simpler API to implement communication logic.
- Configuration subscriptions.
- Plugin system for device configuration and control.

Reducing Boilerplate

Recent FairMQ developments and user feedback allows us to significantly simplify executable creation and device code.

- Extract the main() function to a common header file.
- Provide simple callback-based API for handling data from the communication channels.
- Simplify configuration access and its extension.

- Extract dynamic configuration & control (via DDS) into a plugin system.



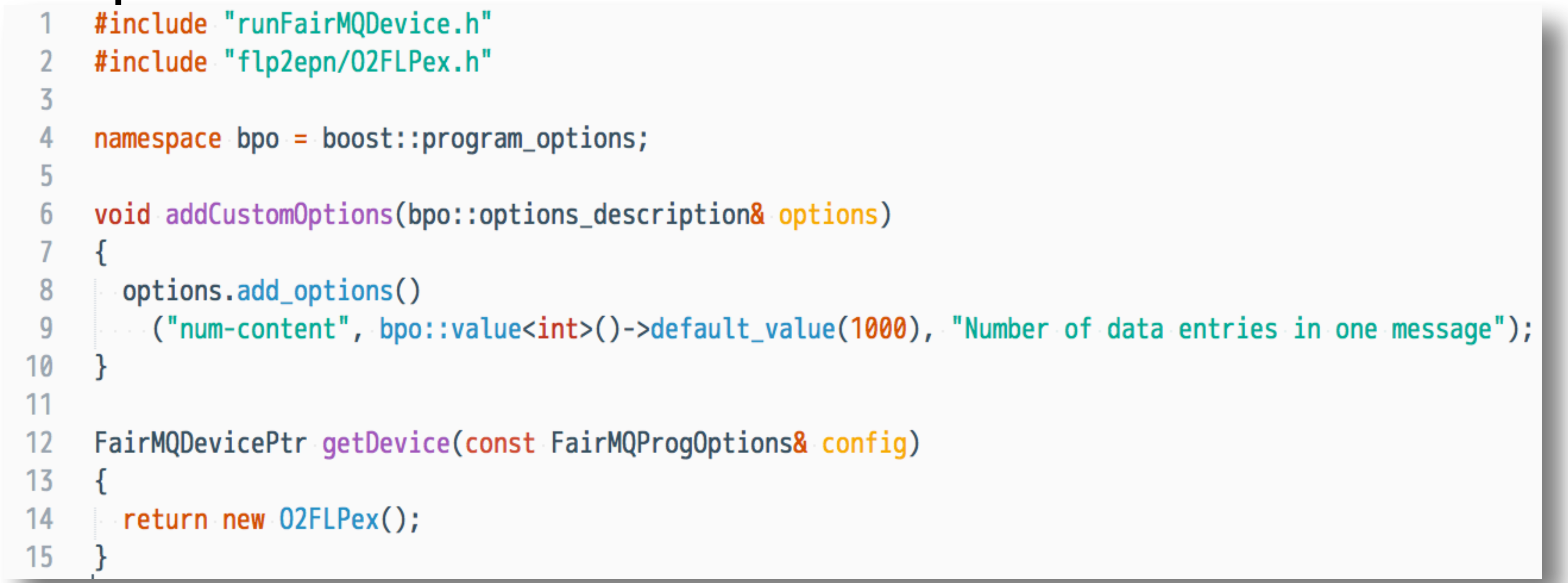
Examples/flp2epn/src/runEPN.cxx	+7 −38
Examples/flp2epn/src/runFLP.cxx	+9 −49
Examples/flp2epn/src/runProxy.cxx	+9 −35
Examples/flp2epn-distributed/run/runEPNReceiver.cxx	+7 −233
Examples/flp2epn-distributed/run/runFLPSender.cxx	+8 −217
Examples/flp2epn-distributed/run/runFLPSyncSampler.cxx	+7 −170

➢ <u>Provide a simple recommended way to write and run a device which does not exclude any complex use cases.</u>

Common main()

- **Extract the `main()` function to a common header file to minimize amount of boilerplate code.**
- **Allows us to hide more internal details from the user and easily extend/change them in the future.**
- **User has to implement two functions:**
  - `addCustomOptions(...)` **: add custom command line options (optional), accessible from the device via `fConfig`.**
  - `getDevice(...)` **: return the device to run.**

Example:



```cpp
#include "runFairMQDevice.h"
#include "flp2epn/O2FLPex.h"

namespace bpo = boost::program_options;

void addCustomOptions(bpo::options_description& options)
{
  options.add_options()
    ("num-content", bpo::value<int>()->default_value(1000), "Number of data entries in one message");
}

FairMQDevicePtr getDevice(const FairMQProgOptions& config)
{
  return new O2FLPex();
}
```

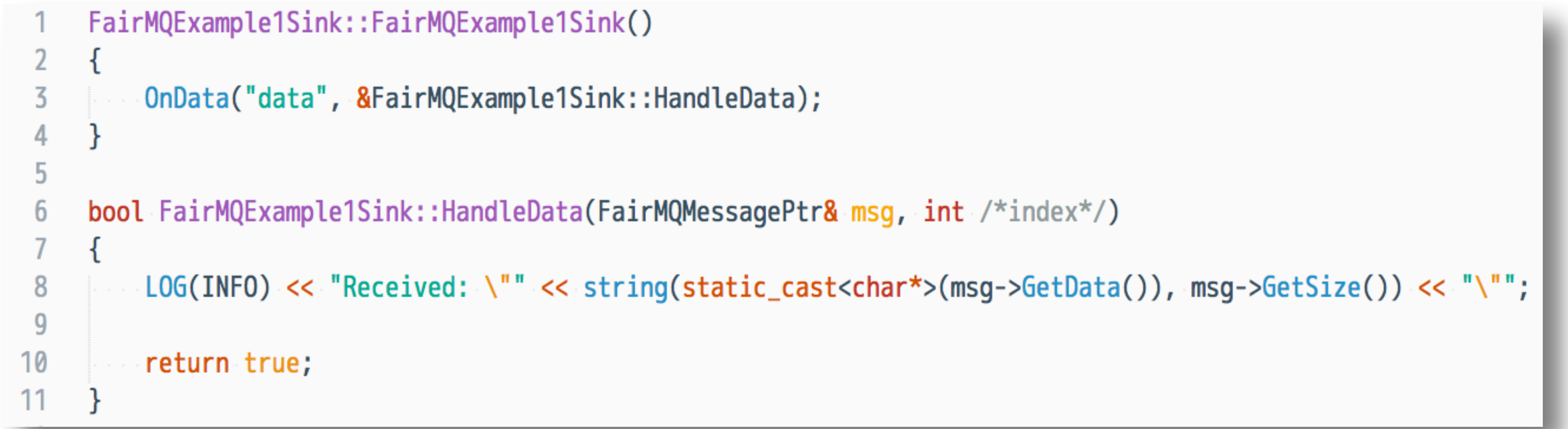
https://github.com/FairRootGroup/FairRoot/tree/dev/examples/MQ

Callback API

- Allow the user to register callback(s) for specific communication channels, that are called when data arrives on them:

```
OnData("channelA", [](FairMQMessagePtr& msg){ /* use data */ });
OnData("channelB", [](FairMQParts& parts){ /* use data */ });
```

- Hides the state check and the loop iterating over `Receive()`.
- Set in the constructor of the device or in `InitTask()`.
- Callbacks are called only when the device is in the RUNNING state.
- If a callback returns `false`, the device goes into IDLE state.

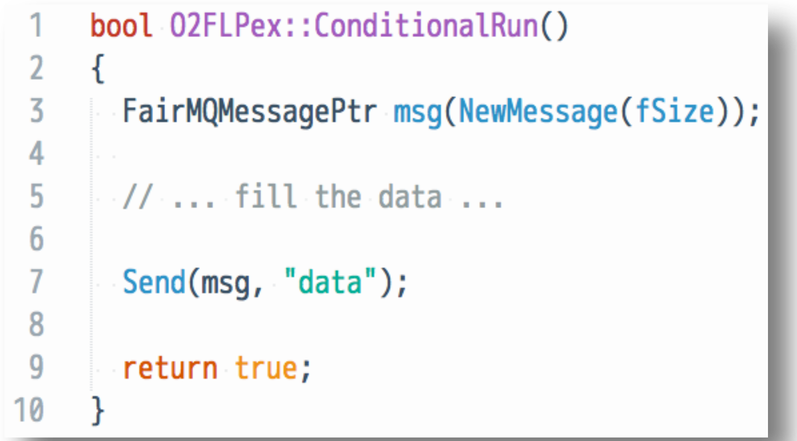


```
FairMQExample1Sink::FairMQExample1Sink()
{
    OnData("data", &FairMQExample1Sink::HandleData);
}

bool FairMQExample1Sink::HandleData(FairMQMessagePtr& msg, int /*index*/)
{
    LOG(INFO) << "Received: \"" << string(static_cast<char*>(msg->GetData()), msg->GetSize()) << "\"";

    return true;
}
```

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

Device without input channels: ConditionalRun()

- **For devices without incoming data channels** `OnData()` **cannot be used.**
- **Instead the device implements** `ConditionalRun()` **method.**
- **It is called repeatedly until it returns** `false`**, after which the device goes into IDLE state.**
- **As with** `OnData()`**, this method hides the state check loop.**



```cpp
bool O2FLPex::ConditionalRun()
{
    FairMQMessagePtr msg(NewMessage(fSize));

    // ... fill the data ...

    Send(msg, "data");

    return true;
}
```

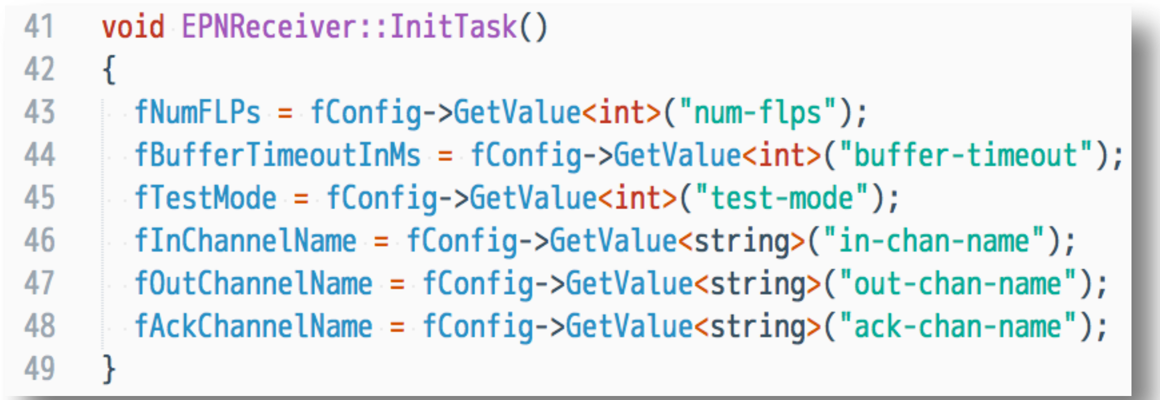
https://github.com/FairRootGroup/FairRoot/tree/dev/examples/MQ

Configuration Access and Extension

The old `GetProperty()/SetProperty()` methods in FairMQ device are very verbose to extend in the derived devices, especially when support for custom options descriptions and different types is needed.

Now the device has direct access to the FairMQProgOptions class via `fConfig` member. It allows easy access to all the provided command line options and other configuration parameters (e.g., from the JSON file).

Here is an example of its use to set some device members from cmd options:



```
void EPNReceiver::InitTask()
{
  fNumFLPs = fConfig->GetValue<int>("num-flps");
  fBufferTimeoutInMs = fConfig->GetValue<int>("buffer-timeout");
  fTestMode = fConfig->GetValue<int>("test-mode");
  fInChannelName = fConfig->GetValue<string>("in-chan-name");
  fOutChannelName = fConfig->GetValue<string>("out-chan-name");
  fAckChannelName = fConfig->GetValue<string>("ack-chan-name");
}
```

Configuration Subscriptions

The FairMQProgOptions class also allows the user to subscribe for configuration updates.

Nicolas Winckler

This can be used, for example, to react when an external tool updates the configuration (to change device state and/or reinitialize).

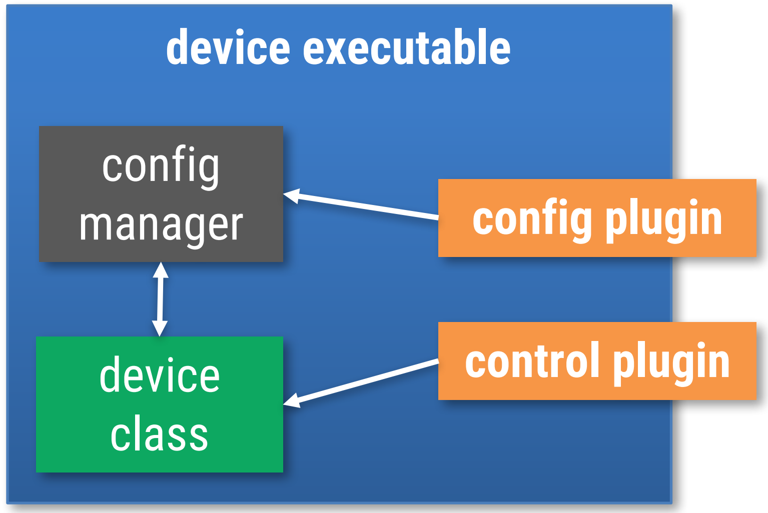
<u>API:</u>

```
config.Subscribe<string>("data.0.address", [&device](const string& key, const string& value)
{
    LOG(INFO) << ”Address configuration has changed: " << key << " = " << value;
    device.fChannels.at("data").at(0).UpdateAddress(value);
});

config.UpdateValue<string>("data.0.address", "tcp://localhost:4321"); // callback will be called after this
```

Example: https://github.com/FairRootGroup/FairRoot/blob/dev/fairmq/options/runConfigEx.cxx

Device Plugins for Configuration/Control

- **Provide a plugin system for:**
  - **Updating device configuration:**
    configure connection parameters with values from runtime environment, configuration files, deployment system.
  - **Controlling the state of the device:**
    accepting commands from an external control system and translating these into device state changes: init, run, stop, etc.



Device Configuration Plugin: DDS

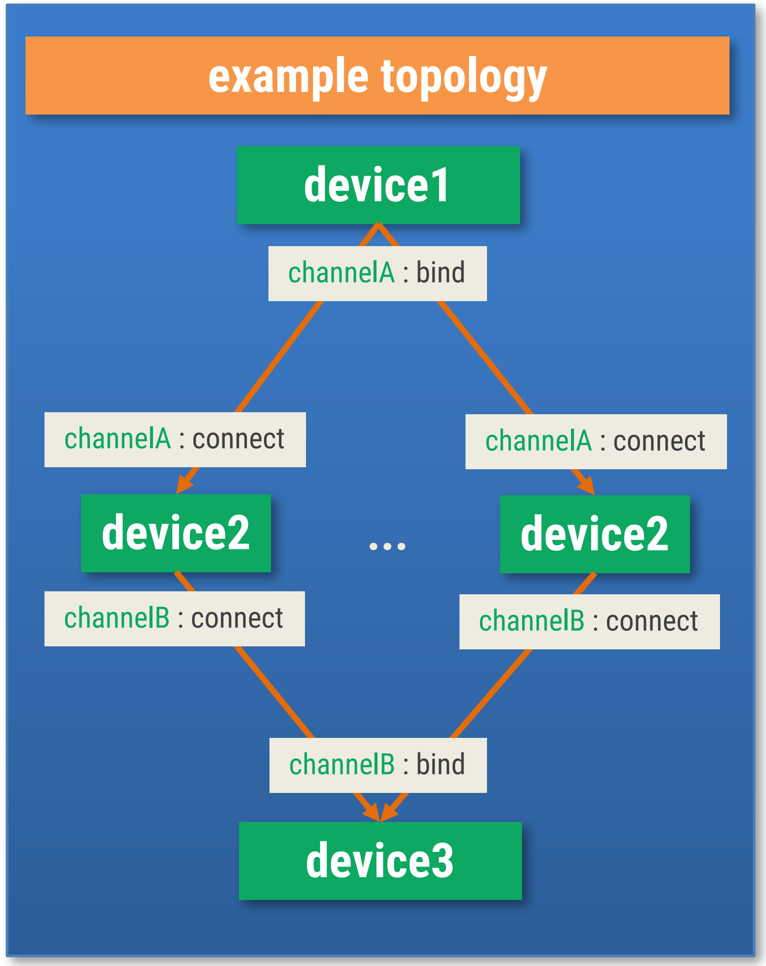
- **Configure a running topology of devices dynamically by exchanging connection information via DDS.**
- **Plugin can be applied to any device without changing its code via command line option:**

```
--config libFairMQDDSConfigPlugin.so
(default is 'static')
```

- **The mapping is done via channel names corresponding to DDS property names in the DDS XML topology file:**
  - Binding channels write DDS properties.
  - Connecting channels read DDS properties.
  - E.g.:

```
"channel": {
    "name": "channelA",
    "method": "bind",
    ...
}
```

```
<properties>
    <id access="write">channelA</id>
</properties>
```



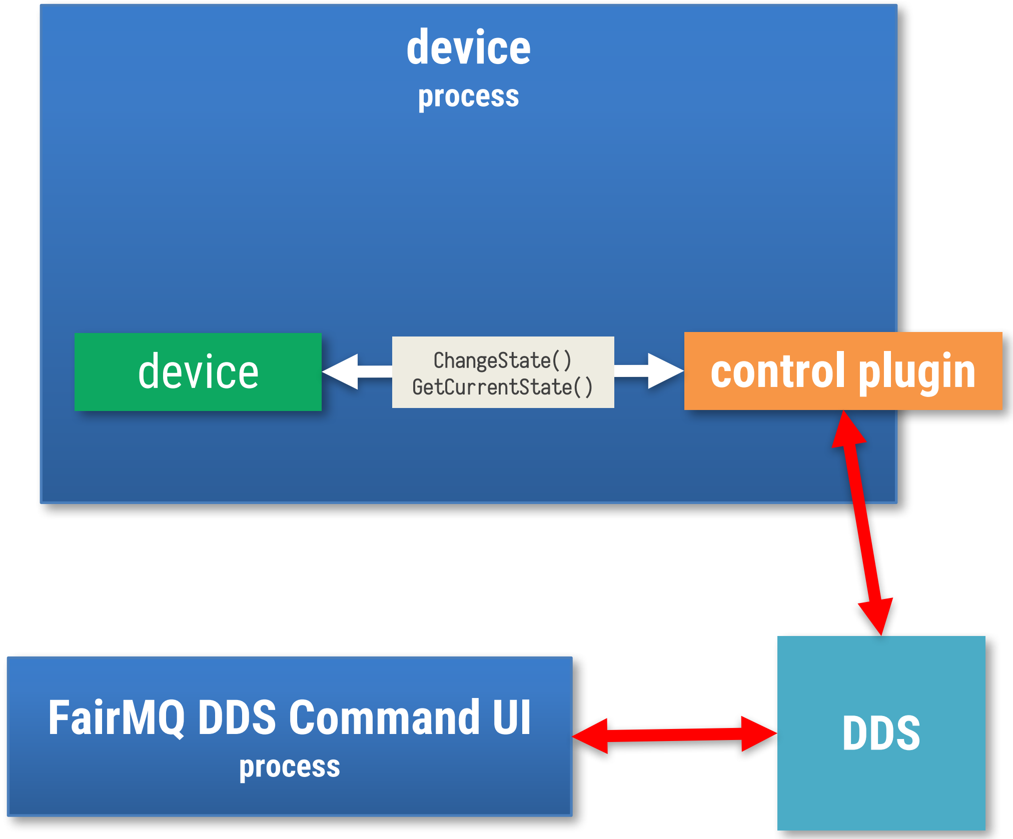
Example: https://github.com/FairRootGroup/FairRoot/tree/dev/examples/MQ/3-dds

Device Control Plugin: DDS

- **FairMQDDSControlPlugin uses DDS CCustomCmd interface to listen to incoming commands.**
- **The commands are sent by a simple reference utility**
  https://github.com/FairRootGroup/FairRoot/blob/dev/fairmq/run/runDDSCommandUI.cxx
  (to run: `$FAIRROOTPATH/bin/fairmq-dds-command-ui` **(when dds-server is running))**
- **Plugin can be applied to any device without changing its code via command line option:**

```
--control libFairMQDDSControlPlugin.so
(other options are 'static' or 'interactive' (default))
```

- **The reference utility allows querying and changing the device state.**
- **DDS CCustomCmd interface allows to target which tasks to send the commands to (via topology path).**



Example: https://github.com/FairRootGroup/FairRoot/tree/dev/examples/MQ/3-dds

Miscellaneous

- `--log-to-file <filename>` **to log to a file instead of console.**
- `NewSimpleMessage()` **for simpler creation of small messages (copying!).**
- **Configurable logging interval per channel (“**`rateLogging”: “<seconds>”`**).**
- **FairMQMultiplier device – send same incoming message to multiple receivers with PUSH-PULL (without an additional copy).**
- **Simpler API for send/receive timeouts:** `Send(msg, “data”, 0, <milliseconds>)`.

Work in Progress:

- **Extension of the configuration plugin to support configuration updates (e.g. topology updates).**
- **Integration of the shared memory transport prototype as a third transport (beside zmq/nanomsg).**