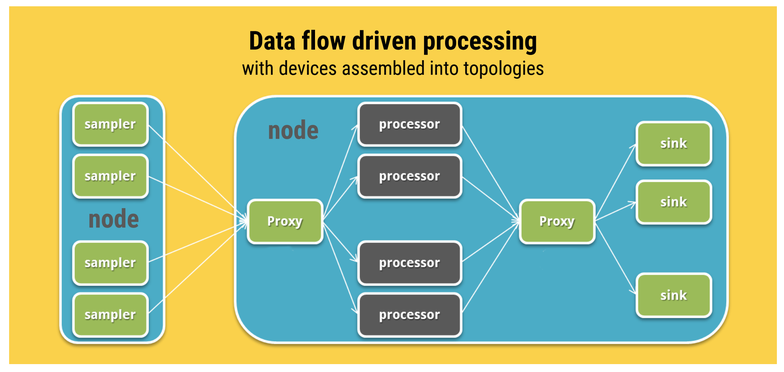
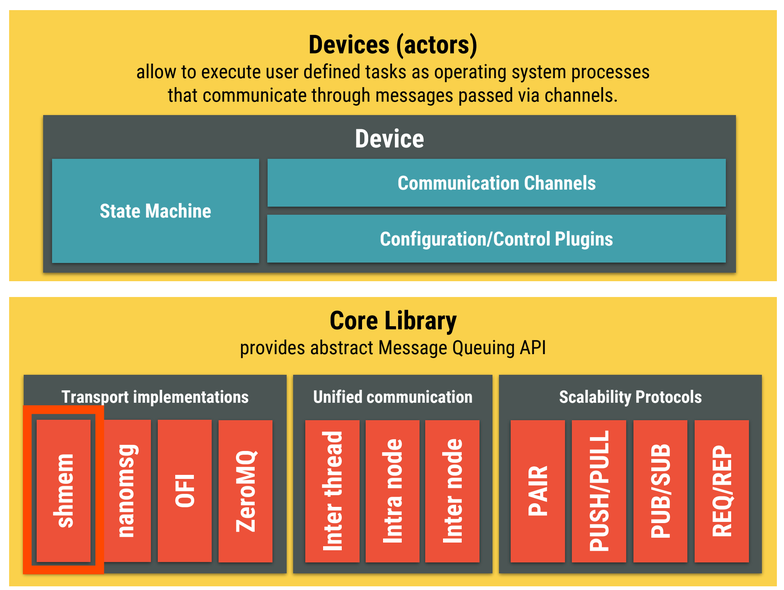
Shared Memory Transport for ALFA

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

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 upgraded ALICE detector or the FAIR experiments.

FairMQ[3] is a C++ Message Queuing Framework that integrates standard industry data transport technologies and provides building blocks for simple 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.





Concepts

Shared memory transport follows the core FairMQ concepts:

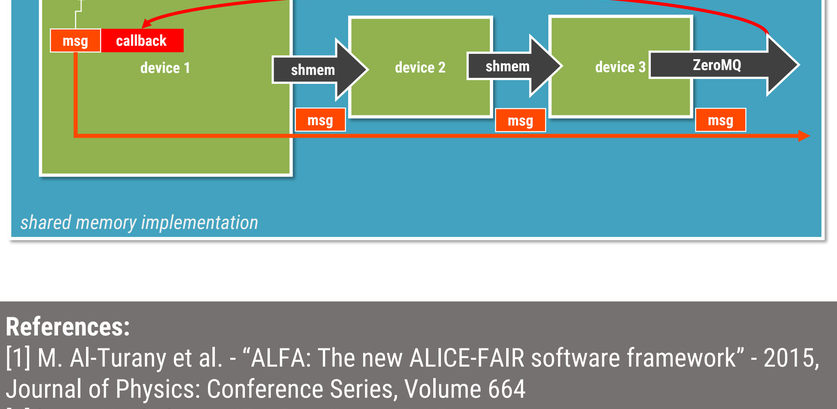
General concepts:
- Hide all transport-specific details from the user.
- Clean, unified interface to different data transports.
- Combinations of different transport in one device in a transparent way.
- Transport switch via configuration only, without modifying device/user code -> same API for all transports.

Ownership concepts:
- 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.

Unmanaged Region

The default message creation in FairMQ hides all the memory allocation/management details from the user and simply provides a ready to use buffer for every message. However, sometimes user might have very specific memory layout requirements – typically where hardware needs to write to that memory (e.g. detector readout). Ideally this memory should still be usable with no/minimal copy by further devices in the pipeline.

- UnmanagedRegion component for full memory layout control.
- Zero-copy for shared memory transport.
- Allocates memory via the transport allocator and provides it to the user to manage.
- Message creation out of subset(s) of this region.
- Framework cleans only the entire region, not separate messages – this is in user hands.
- Callback system to notify creator when a buffer is no longer needed by transport.



FairMQ Shared Memory Transport
Motivation

Provide faster inter-process transport for large messages.
Network libraries involve at least one copy of the data buffer.

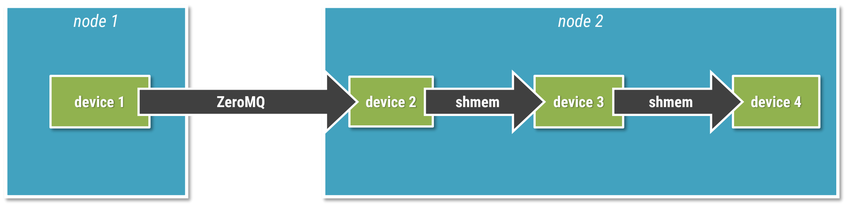
node	process	address format	ZeroMQ[5]	nanomsg[6]	shmem	OFI[4]
intra-	intra-	inproc://endpoint	✓	✓	N.A.	N.A.
intra-	inter-	ipc://endpoint	✓	✓	✓	X
		tcp://host:port	✓	✓	✓	✓
inter-	inter-	tcp://host:port	✓	✓	N.A.	✓
		verbs://host:port	X	X	N.A.	✓

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

Particle physics experiments, such as ALICE at CERN and future FAIR experiments at GSI, will produce more than a terabyte of data per second. Avoiding additional copies of the data on the same node has huge benefits for performance and cost of data processing in such scenarios.

Connecting to Other Transports

- Efficient mechanism to connect different transports to each other. Example for this is connecting network transport to shared memory.
- No additional copy if the target transport allows it.
- Each channel can have different transport.
- No transport specific details are needed in the device code – it knows only channel names – the actual transport of a channel is decided by the configuration.



Currently shmem+ZeroMQ Send and OFI[4]+shmem Send/Receive operations are zero-copy.

Monitoring

Because shared memory segments can outlive the process, it is important to make sure no unused shared memory is left in case devices crash and fail to cleanup used resources.

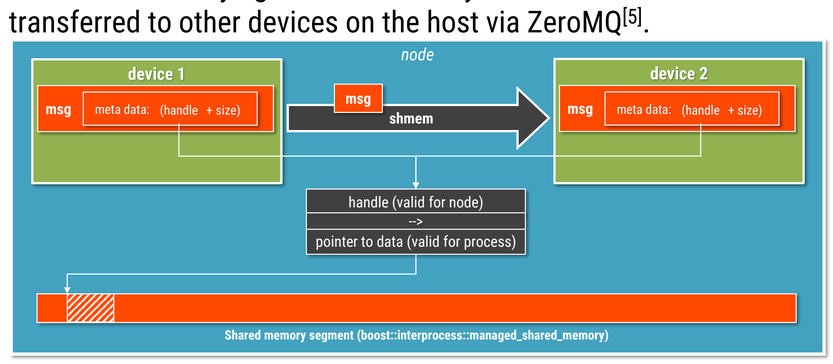
- Automatic cleanup of shared memory resources, even if all devices crash.
- Monitoring tool to debug/monitor shared memory use.

```
~/d/FairMQ  dev  build
 ./fairmq/fairmq-shmmonitor -i                          Wed 20 Jun 2018 05:58:58 PM CEST
Starting shared memory monitor for session: "default_"...
Did not remove "fmq_default__cq". Already removed?

controls: [x] close memory, [p] print queues, [h] help, [q] quit.

|              name |       size |       free | ok |  # devices |   ms since |
|  fmq_default__main | 2000000000 | 1996999632 |  1 |          2 |         32 |\
```

Implementation

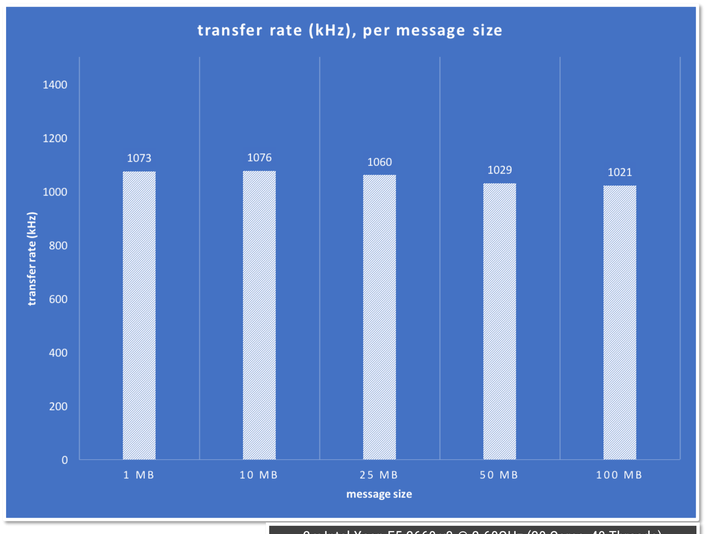
- boost::interprocess[7] library for management and allocation of shared memory - cross-platform shared memory implementation with many features such as different allocation algorithms, shmem STL-like containers, shmem smart pointers, message queues and many more.
- ZeroMQ library for transfer of the meta information associated with the memory.

The FairMQ message object holds meta information (handle + size) about the underlying shared memory. This meta information is transferred to other devices on the host via ZeroMQ[5].



Performance

Transfer Rate:
Allocation of messages with shmem transport and transfer to a second device.



2 x Intel Xeon E5-2660 v3 @ 2.60GHz (20 Cores, 40 Threads)
128 GB RAM, 60 GB shared memory segment

CPU Usage:
CPU usage measured at ~2.5 GB/s rate (one core) (1MB message size)

	ZeroMQ:TCP	shmem
sender	~ 68.5%	~ 1.1%
receiver	~ 89.1%	~ 0.9%

Run the tests:

```
$ fairmq-bsampler --id bsampler1 --mq-config config/benchmark.json --transport shmem(/zeromq)
--same-msg false --msg-rate 2500
```

```
$ fairmq-sink --id sink1 --mq-config config/benchmark.json --transport shmem(/zeromq)
```

References:
[1] M. Al-Turany et al. - "ALFA: The new ALICE-FAIR software framework" - 2015, Journal of Physics: Conference Series, Volume 664
[2] https://github.com/FairRootGroup/FairRoot
[3] https://github.com/FairRootGroup/FairMQ
[4] CHEP 2018, Dennis Klein, "RDMA-accelerated data transport in ALFA"
[5] http://zeromq.org/
[6] https://nanomsg.org/
[7] https://www.boost.org/doc/libs/1_67_0/doc/html/interprocess.html