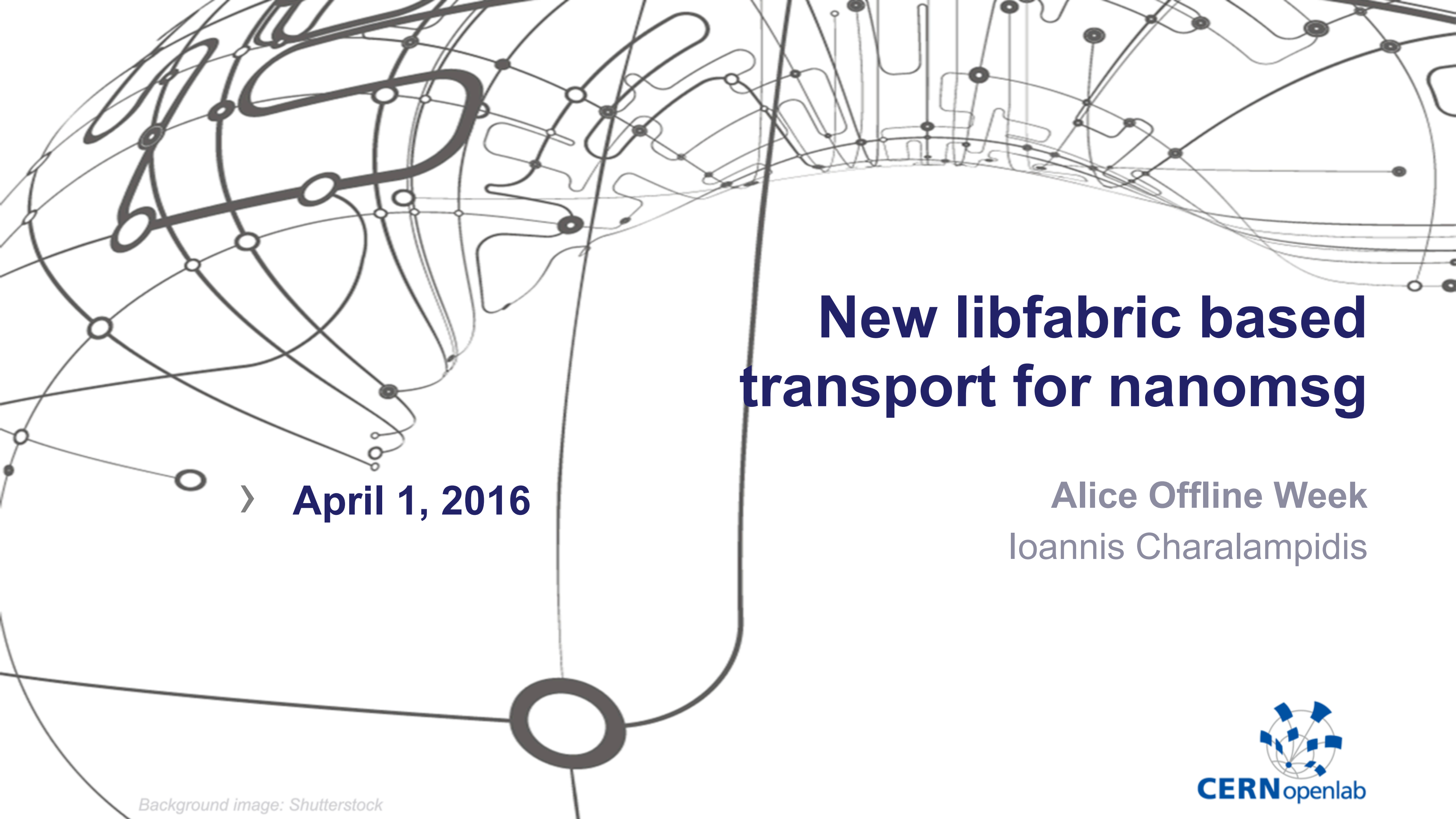


Background image: Shutterstock

New libfabric based transport for nanomsg

› **April 1, 2016**

Alice Offline Week

Ioannis Charalampidis

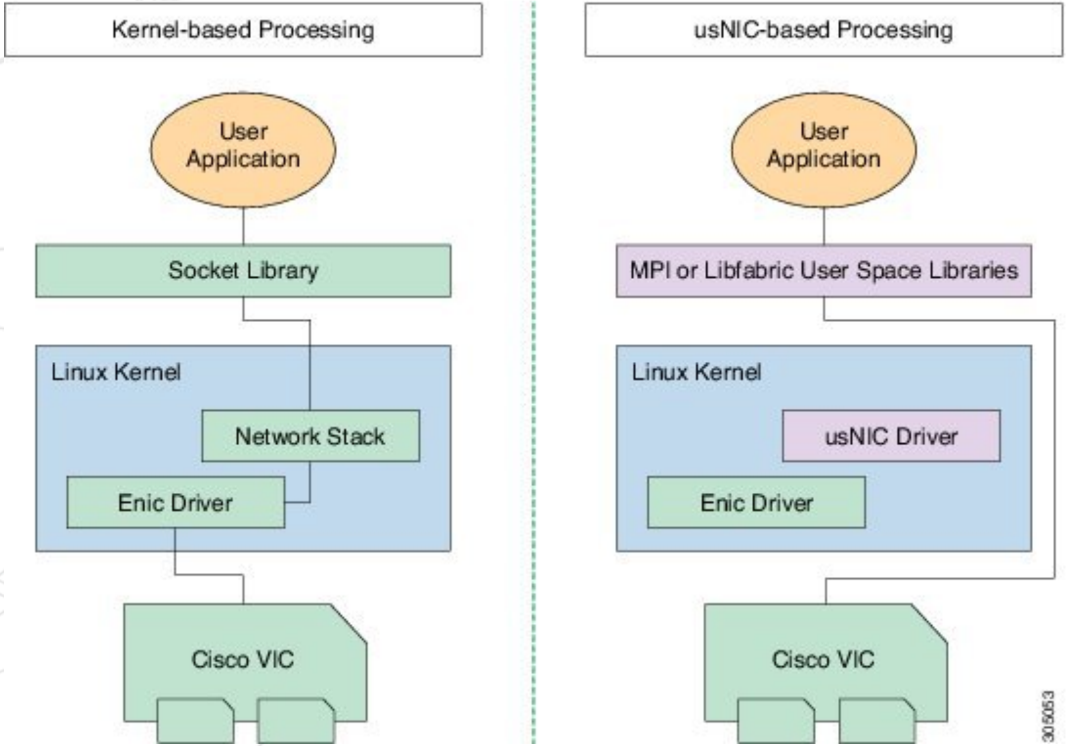
CERNopenlab

A bit of Background

- **Openlab collaboration with CISCO**
  - Interface CISCO’s user-space NICs (`usNIC`) to ALICE experimental software
  - Benchmark performance
  - Decide on further use-cases
- **An interesting by-product**
  - A new transport for the `nanomsg` library
  - https://github.com/wavesoft/nanomsg-transport-ofi/

Background image: Shutterstock

“User-Space” NIC



Bypass the Linux Kernel and communicate directly with the NIC from User Space

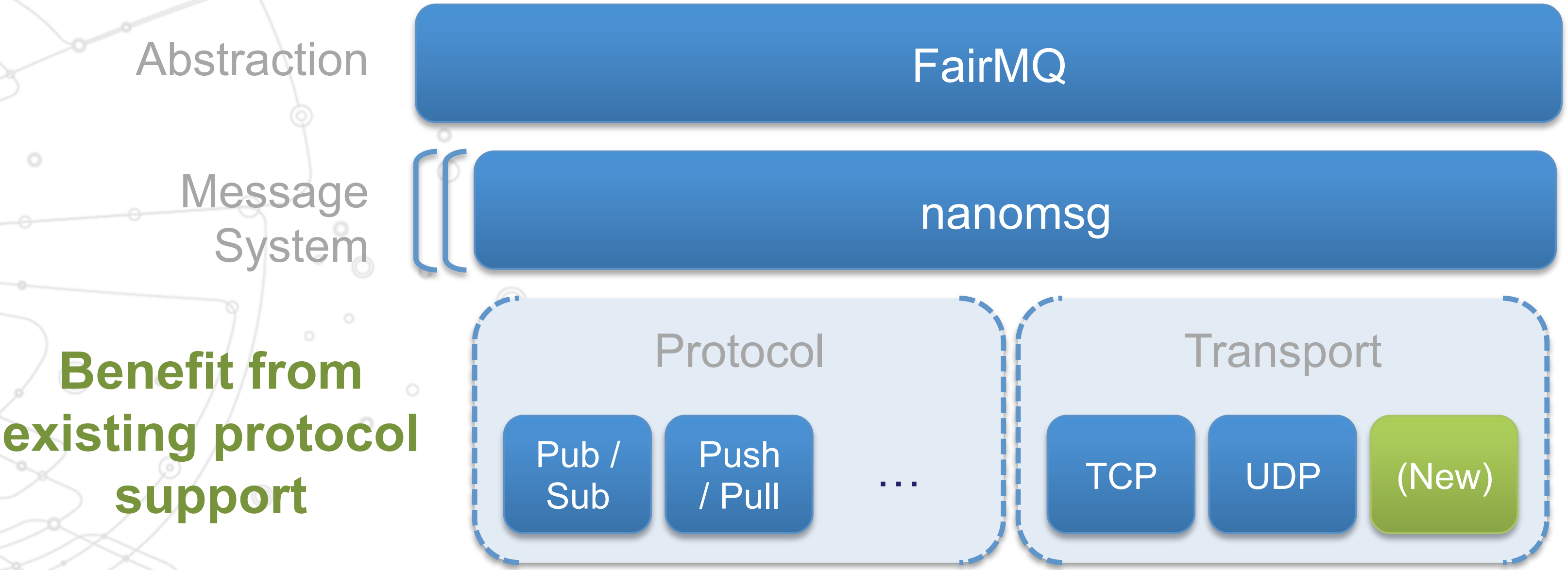
Background image: Shutterstock

ALICE Software & usNIC

- **Requirements**
  - It must interface with current ALICE software without any modification in them
- **Extend FairMQ**
  - It's a lightweight wrapper around `ØMQ` / `nanomsg`
  - We have to extend `nanomsg` or `ØMQ`
  - We decided to go with **`nanomsg`**, because of cleaner and easily extensible API

Background image: Shutterstock

Extending `nanomsg`

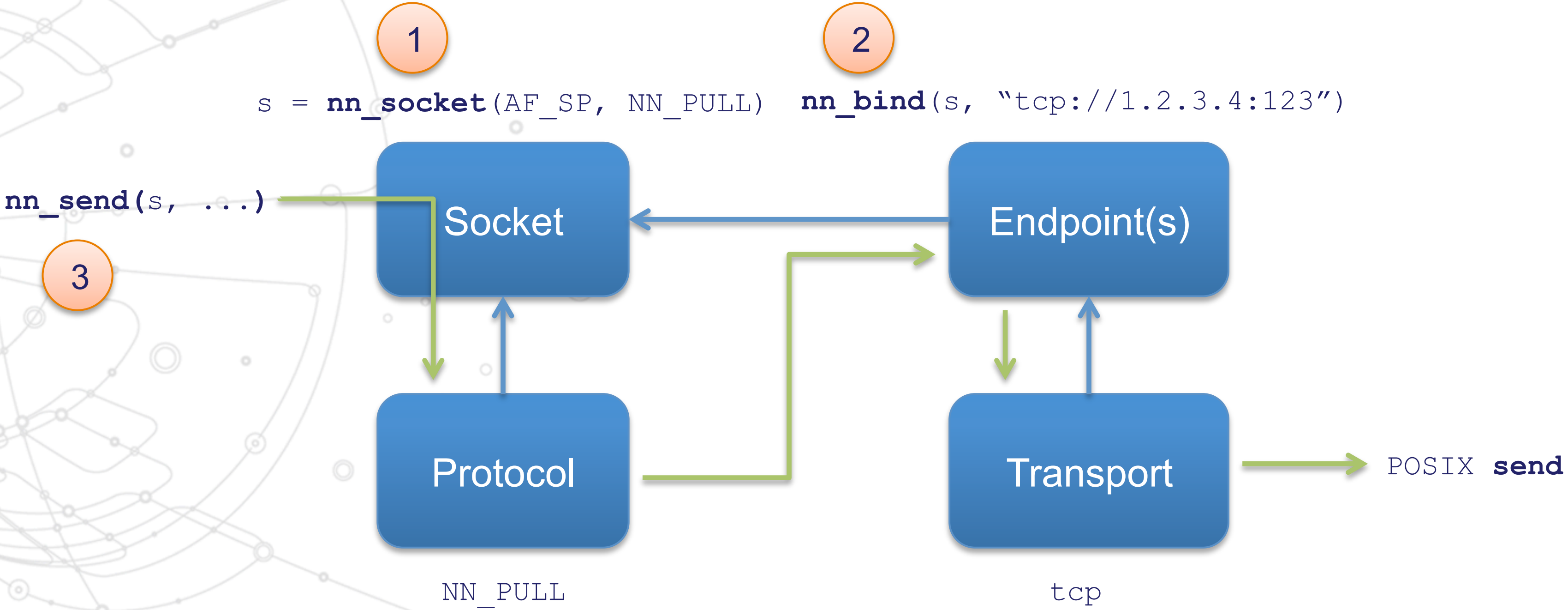


Benefit from existing protocol support

Focus on the transport

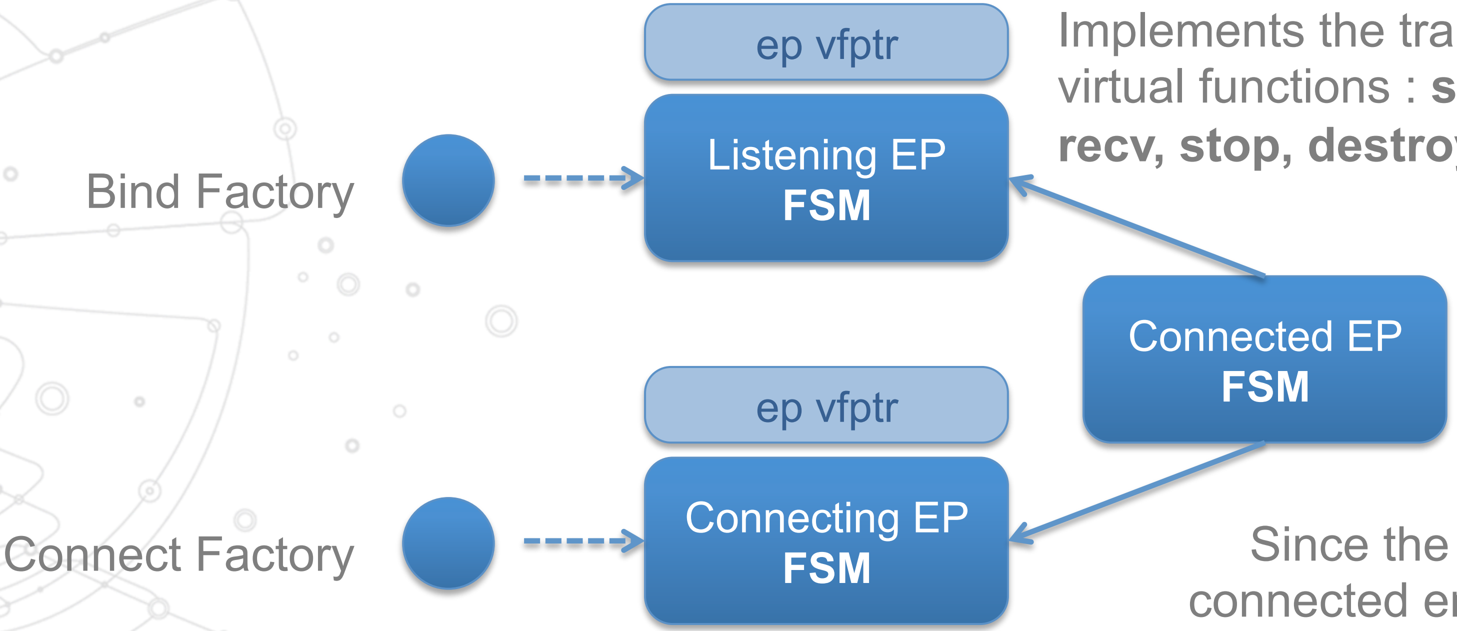
Background image: Shutterstock

Components of nanomsg



Background image: Shutterstock

Transport in `nanomsg`



Implements the transport virtual functions : **send, recv, stop, destroy**

Since the logic of a connected endpoint is the same, it's isolated in a separate FSM

Background image: Shutterstock

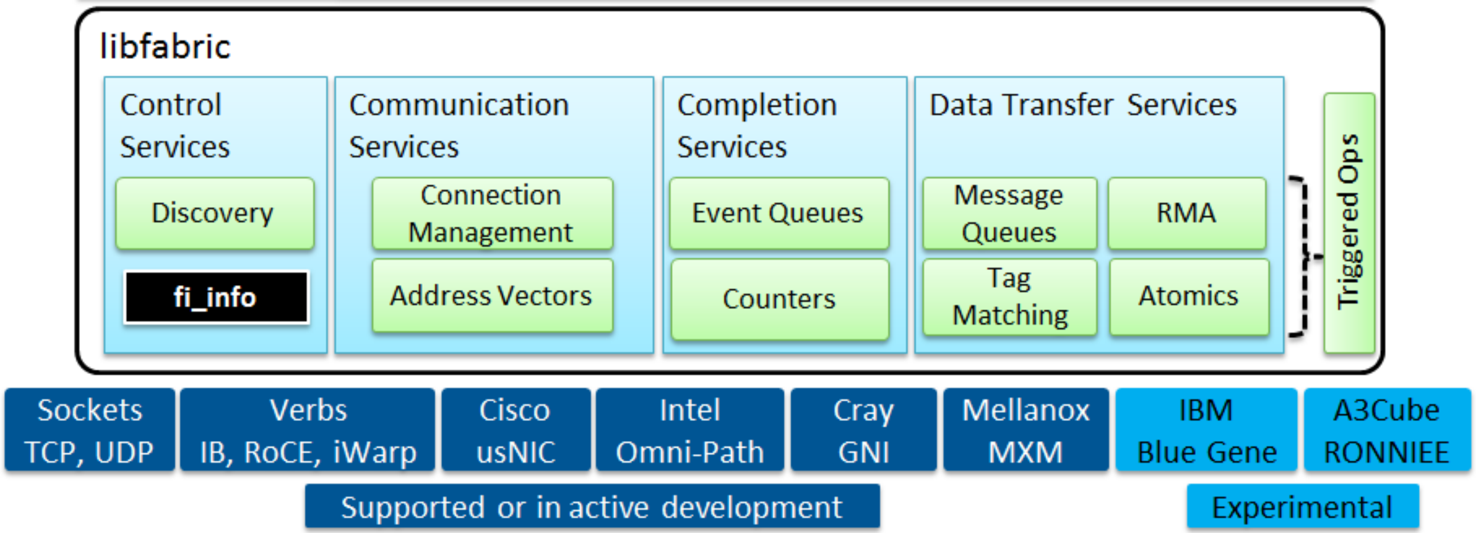
Interfacing to usNIC

- **Difficulties**
  - The core of ØMQ or NanoMsg is designed around the UNIX sockets
  - usNIC API is closer to MPI or RDMA
- **We are using `libfabric`**
  - It is somewhere in the middle

Background image: Shutterstock

OpenFabrics Interfaces (OFI)

Unified **`libfabric`** API



libfabric

Control Services

Discovery

fi_info

Communication Services

Connection Management

Address Vectors

Completion Services

Event Queues

Counters

Data Transfer Services

Message Queues

RMA

Tag Matching

Atomics

Triggered Ops

Sockets TCP, UDP

Verbs IB, RoCE, iWarp

Cisco usNIC

Intel Omni-Path

Cray GNI

Mellanox MXM

IBM Blue Gene

A3Cube RONNIEE

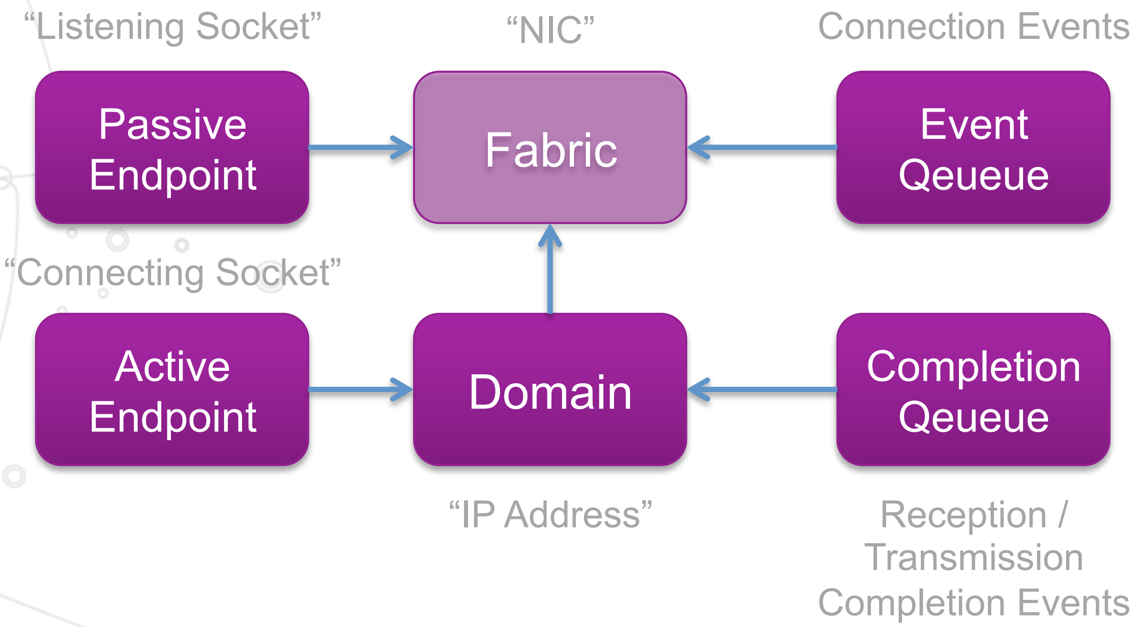
Supported or in active development

Experimental

Different Low-Lattency, High-Performance Fabric Hardware

Background image: Shutterstock

libfabric Terminology

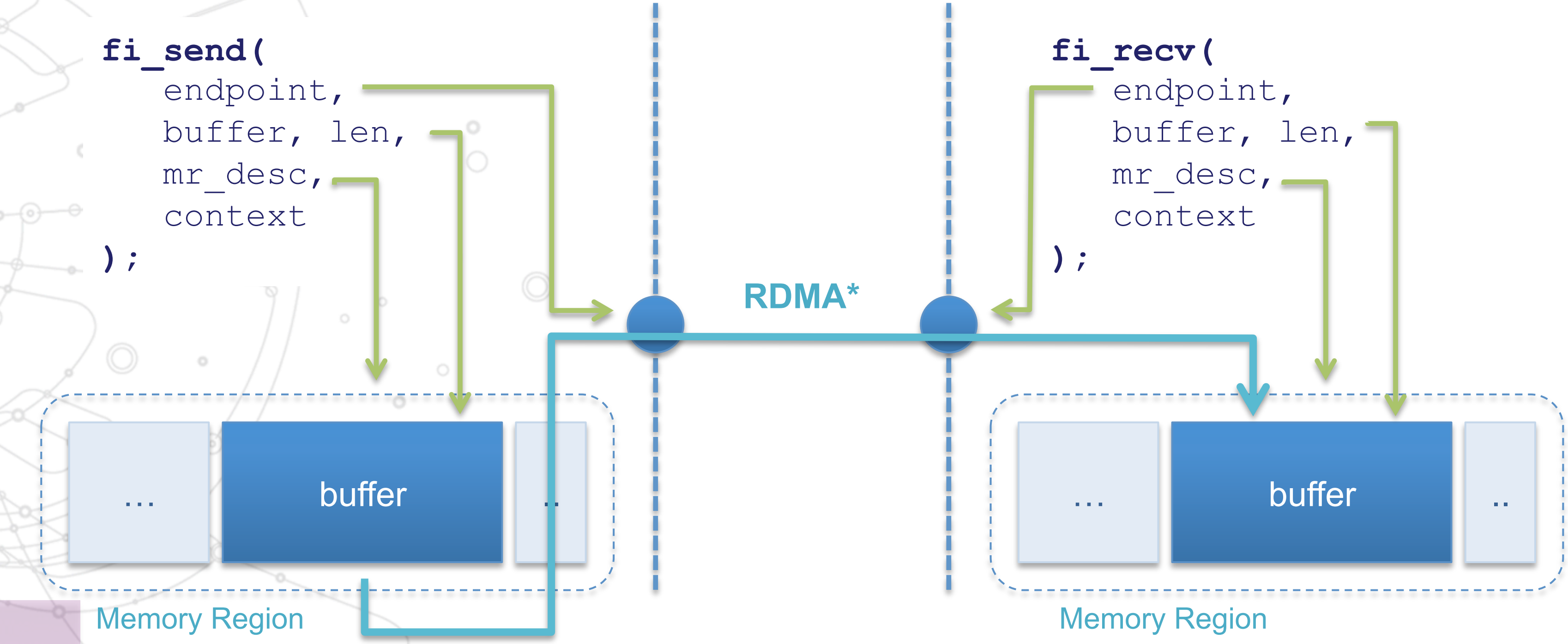


Background image: Shutterstock

`libfabric` Features

- **Active Endpoint Types**
  - `FI_DGRAM` – Unreliable Datagrams (ex. UDP)
  - `FI_RDM` – Reliable Datagrams (ex. RDMA)
  - `FI_MSG` – Connection-aware message passing (ex. TCP)
- **High-level API is close to socket API**
  - The provider implements fragmentation and flow control
  - Simple functions : `fi_send(), fi_recv()`
  - It uses RDMA behind the scenes!

libfabric API



```
fi_send(
    endpoint,
    buffer, len,
    mr_desc,
    context
);
```

```
fi_recv(
    endpoint,
    buffer, len,
    mr_desc,
    context
);
```

RDMA*

…

buffer

..

Memory Region

…

buffer

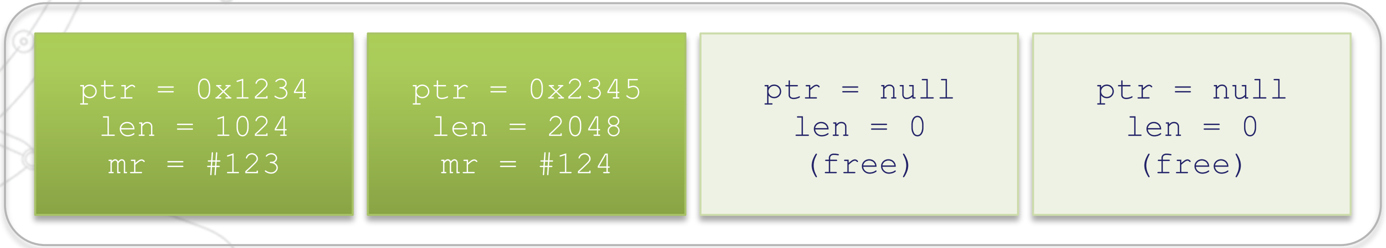
..

Memory Region

Background image: Shutterstock

Memory Registration

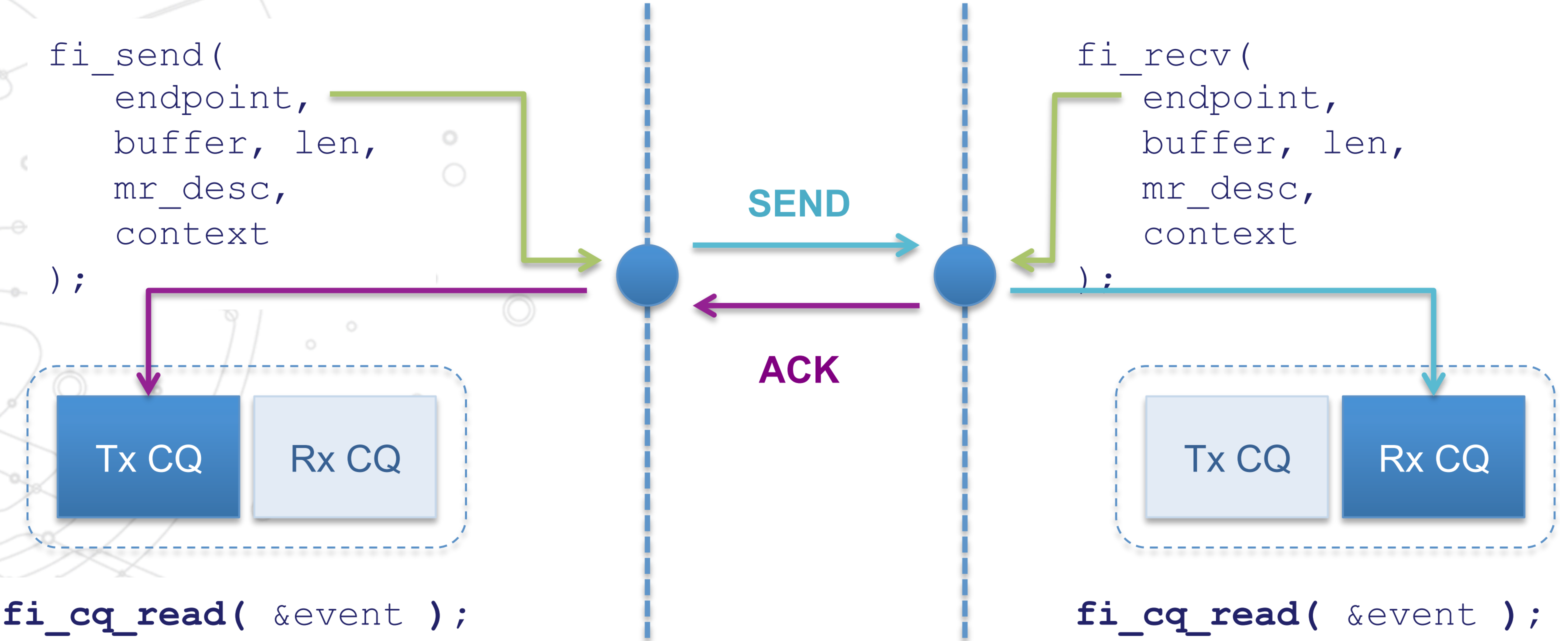
- **Register outgoing messages on-the-fly**
  - OFI transport has re-usable memory “banks”
  - If the pointer being sent belongs to a registered region, the MR description from that bank will be used
  - Otherwise the oldest bank will be de-registered and populated with the new pointer information



ptr = 0x1234
len = 1024
mr = #123
	ptr = 0x2345
	len = 2048
	mr = #124
		ptr = null
		len = 0
		(free)
			ptr = null
			len = 0
			(free)

Background image: Shutterstock

libfabric Events



```
fi_send(
    endpoint,
    buffer, len,
    mr_desc,
    context
);
```

```
fi_recv(
    endpoint,
    buffer, len,
    mr_desc,
    context
);
```

```
fi_cq_read( &event );
```

```
fi_cq_read( &event );
```

* libfabric has custom event polling functions

Background image: Shutterstock

Receiving Events

- **`libfabric` API has custom polling functions**
  - We cannot re-use the existing FD-based solutions *
- **OFI Transport polls the CQs and EQs**
  - A dedicated thread polls all the currently active CQs/EQs and it forwards the events to the appropriate endpoint FSMs
  - Where supported, it uses *wait sets* to synchronously wait for an event from any source, otherwise it spins

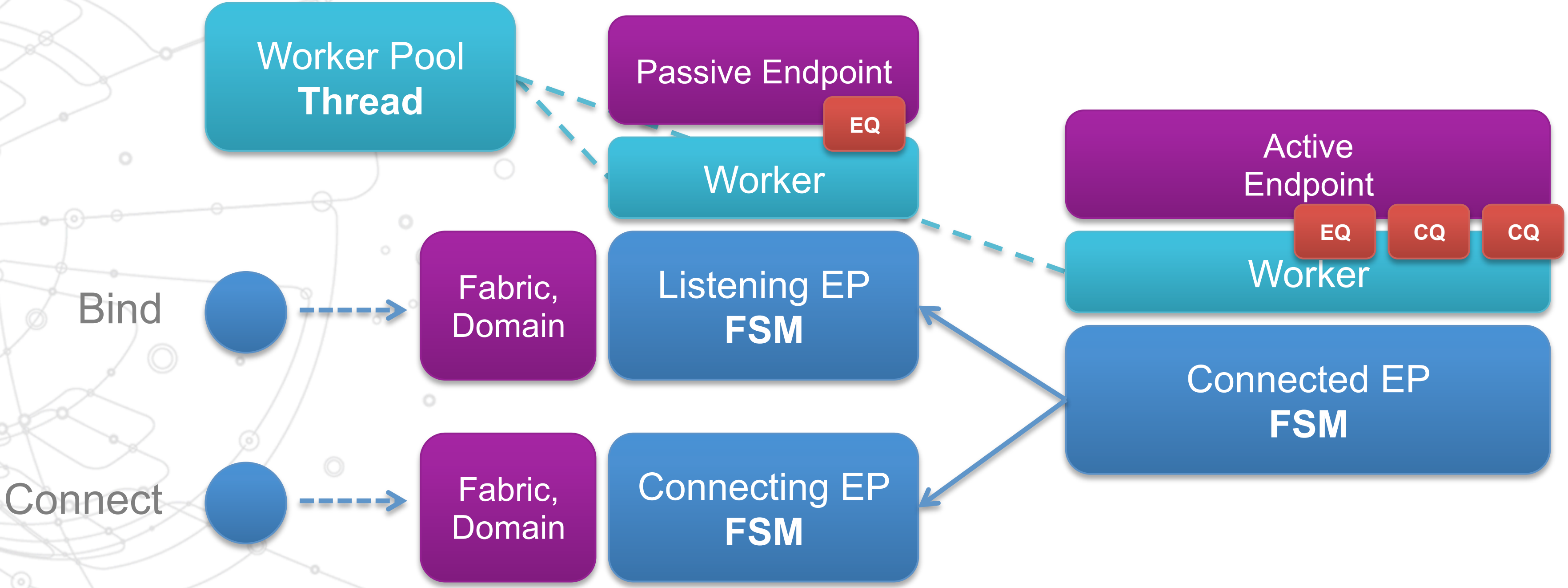
Background image: Shutterstock

Receiving Events

- **** NOTE: The libfabric specs DO support FDs***
  - *It's possible to create an EQ or CQ with an underlying 'waitable' object, such as a mutex or a file descriptor*
  - *However it's not (yet) supported by all providers*

Background image: Shutterstock

The OFI Transport



Background image: Shutterstock

The OFI Transport

- **The ‘ofi’ transport is selected with the nanomsg uri:**
  - `ofi://ip:port[@fabric[:provider]]`
  - The appropriate fabric is selected by it’s IP address and/or the fabric specifications provided
- **Seamless transition to other providers**
  - The transport is completely agnostic to the provider. The same code works the same with infiniband, omnipath, usnic etc.

Background image: Shutterstock

Zero-Copy in `nanomsg`

- **Buffers in `nanomsg` are organized in chunks**
  - Each chunk has a reference counter
  - Instead of copying, it increments the reference number
- **When data from a raw pointer are to be sent, they are copied in a new chunk**
  - In order to avoid this, the `nn_allocmsg` function should be called to allocate a new chunk in advance
  - Till now (v0.8-beta) it’s not possible to allocate a chunk from existing data

Background image: Shutterstock

Additions to `nanomsg`

- **1. Chainable chunk destructors**
  - To allow transports to track when a chunk is free’d in order to invalidate the memory registration.
- **2. Create `nn_msg` from user pointer**
  - Instead of letting nanomsg allocate the message body, the the function `nn_allocmsg_ptr` enables creation of a zero-copy message from existing data
- Pull request submitted :
  - https://github.com/nanomsg/nanomsg/pull/612

Background image: Shutterstock

Benchmarks

Test set-up

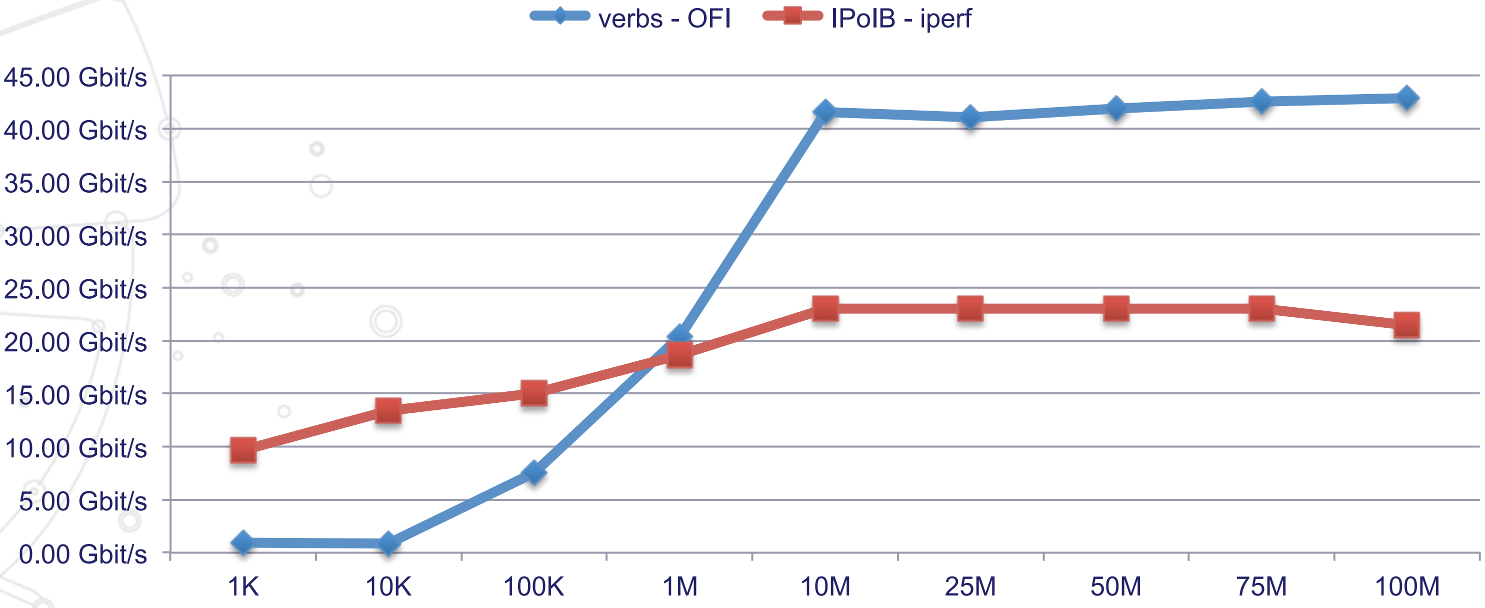
- Intel Xeon E5-2690
  - 2.9 GHz
  - 8 core (16 threads)
  - L2 8x256 KB
  - L3 or LLC (8x2.5MB)
- InfiniBand FDRx4 (56 Gb/s)
  - Mellanox MT27500 (ConntctX-3)

- CentOS 7.2.1511
  - 3.10.0-327.4.5.el7.x86_64

- nanomsg-transport-ofi 1.0.0
  - Beta version

Background image: Shutterstock

Benchmarks



***Warning!** Preliminary results with early beta version of the transport.*
More benchmarks are currently undergoing.

Background image: Shutterstock

Conclusions

- The nanomsg OFI transport enables **socket-like** interface to high performance RDMA fabrics, such as Infiniband, Omni-Path, usNIC etc.
- Even from the early development versions the **performance** measurements looks promising
- There is still lots of room for **improvement**
  - Better memory registration, stability issues, etc.

Background image: Shutterstock