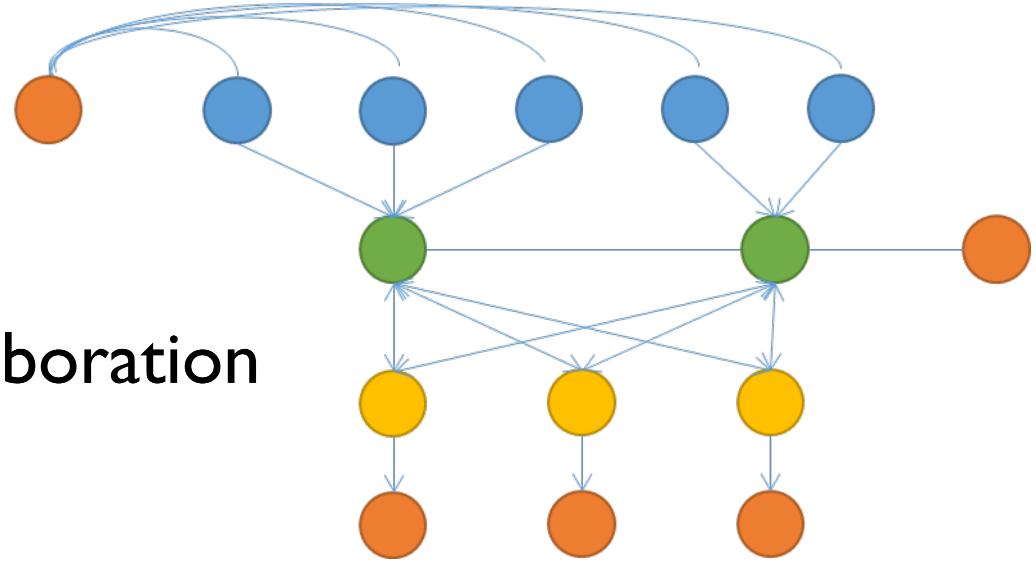
Utilizing HPC Network Technologies in High Energy Physics Experiments

Jörn Schumacher, CERN
Jorn.Schumacher@cern.ch



On behalf of the ATLAS Collaboration

25th Annual Symposium on High-Performance Interconnects
Santa Clara, 2017

High Energy Physics and Interconnects

In this presentation we will talk about…

… why High Energy Physics (HEP) **needs HPC Interconnects**

… why the current state-of-the art HPC network APIs are **not useful for HEP**

… what we can **do about it** (spoiler: we build our own API)

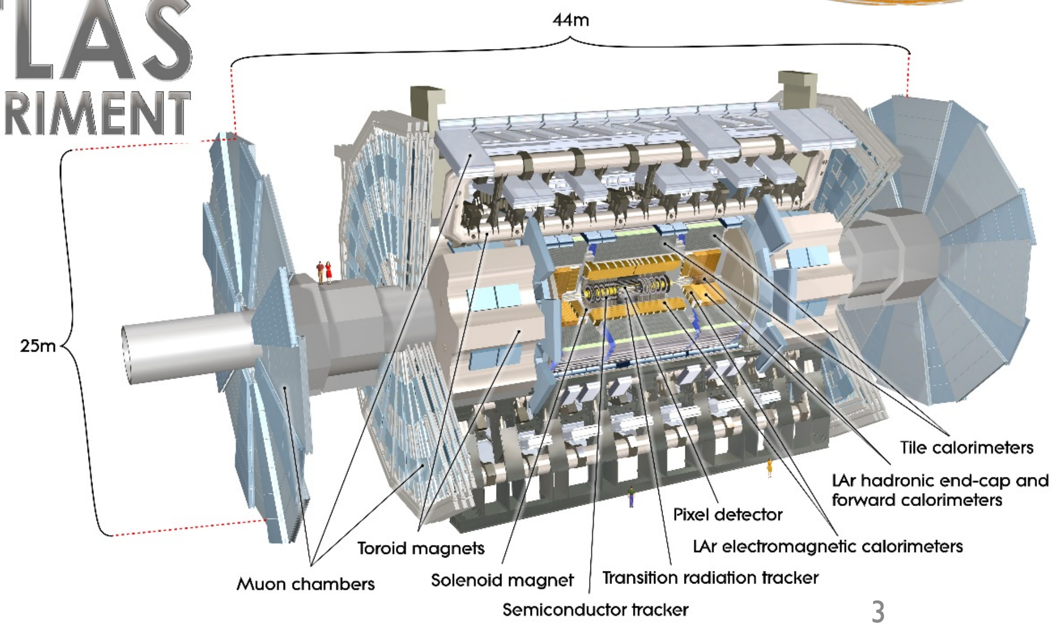
High Energy Physics

Large Hadron Collider (LHC)
27km circular collider in Geneva, Switzerland

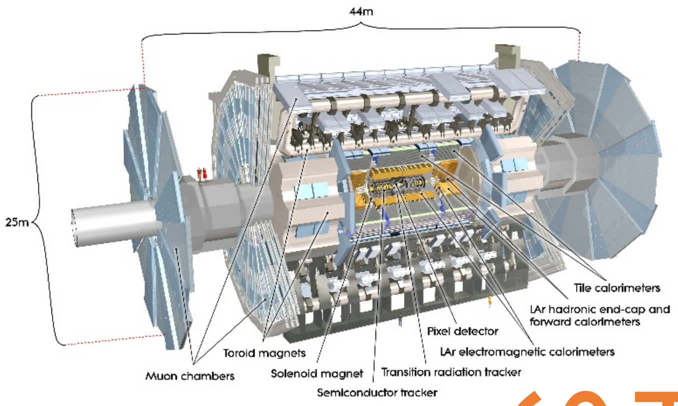
Particle colliders used in HEP study physics processes on a microscopic scale

Length (m)	46
Diameter (m)	25
Weight (t)	7000
Number of electronic channels	$100 \cdot 10^6$

ATLAS EXPERIMENT



Data Acquisition



60 TB/s

Custom electronics and a server farm with 40,000 cores

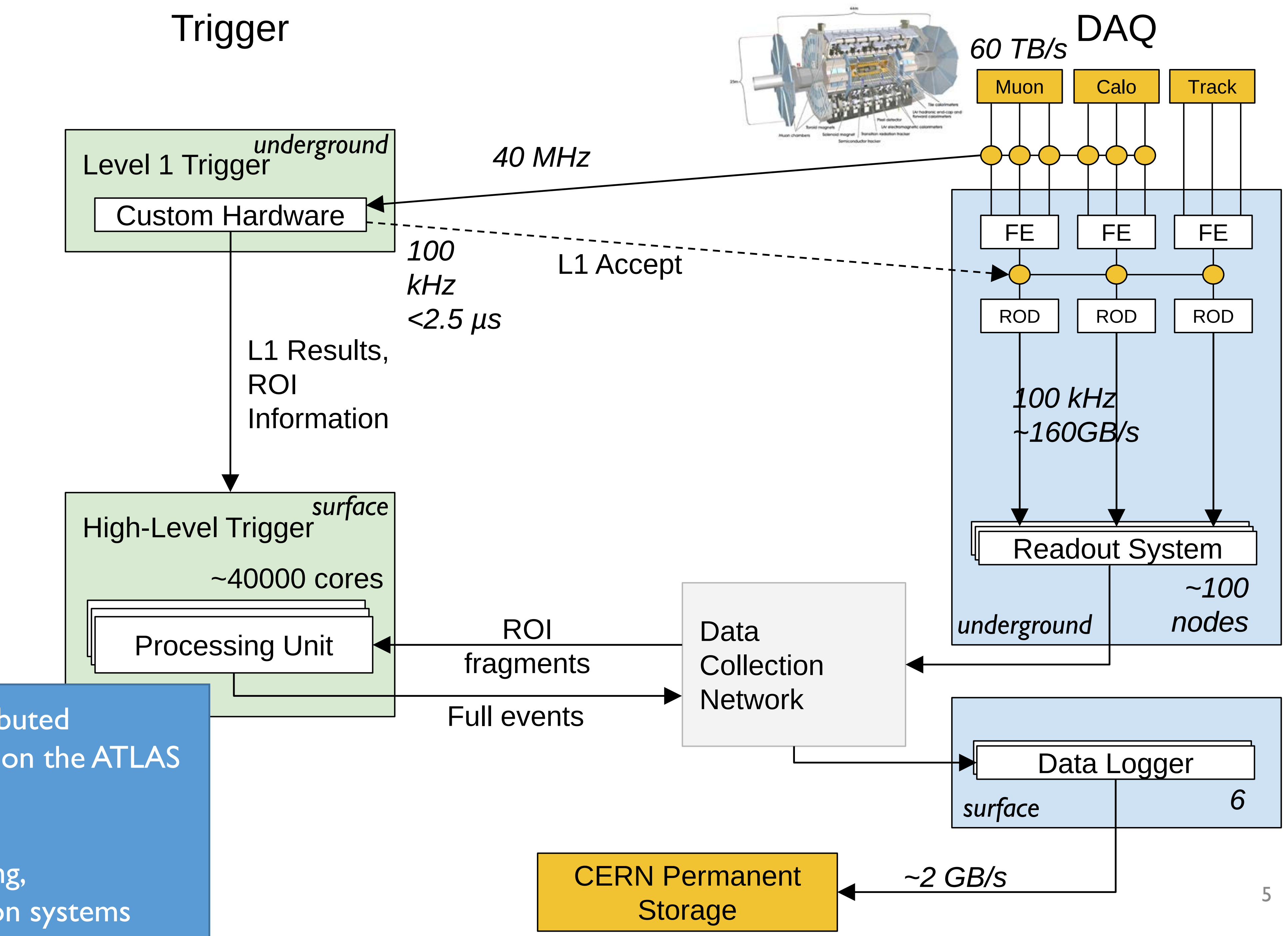
Data Filtering: Order of 10000x reduction in **real-time**

2 GB/s

Needle in a haystack: Looking for extremely rare events with a probability of 10^{-13}

Need **High Performance Networks** to move data at high rates under real-time conditions

ATLAS Data Acquisition 2017

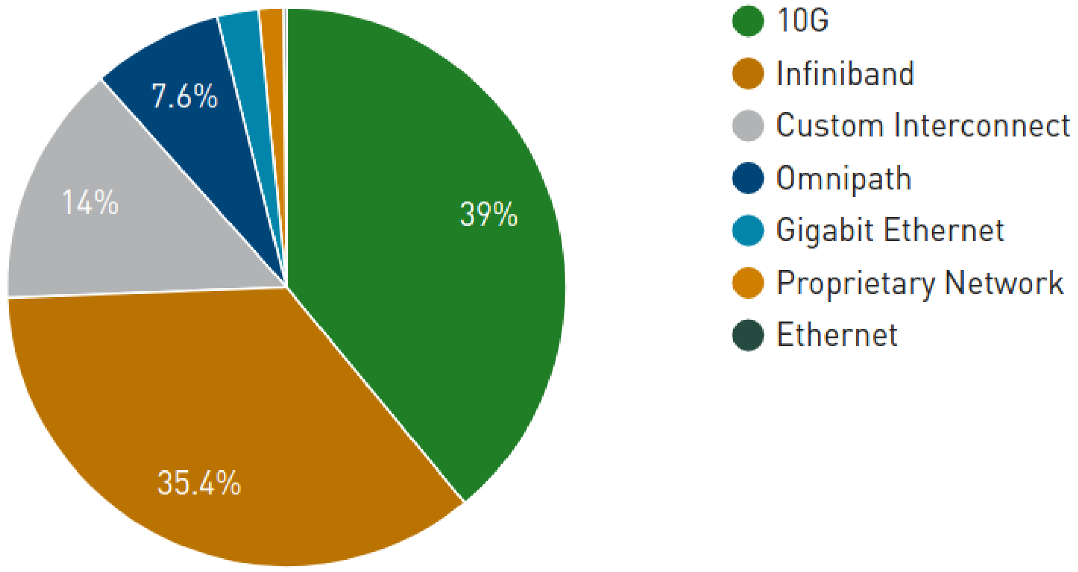


In total 10,000s of distributed applications are running on the ATLAS DAQ system

Not displayed: Monitoring, Infrastructure, Calibration systems

High Performance Networks

Interconnect Families in Top500 List in July 2017



Source: top500.org

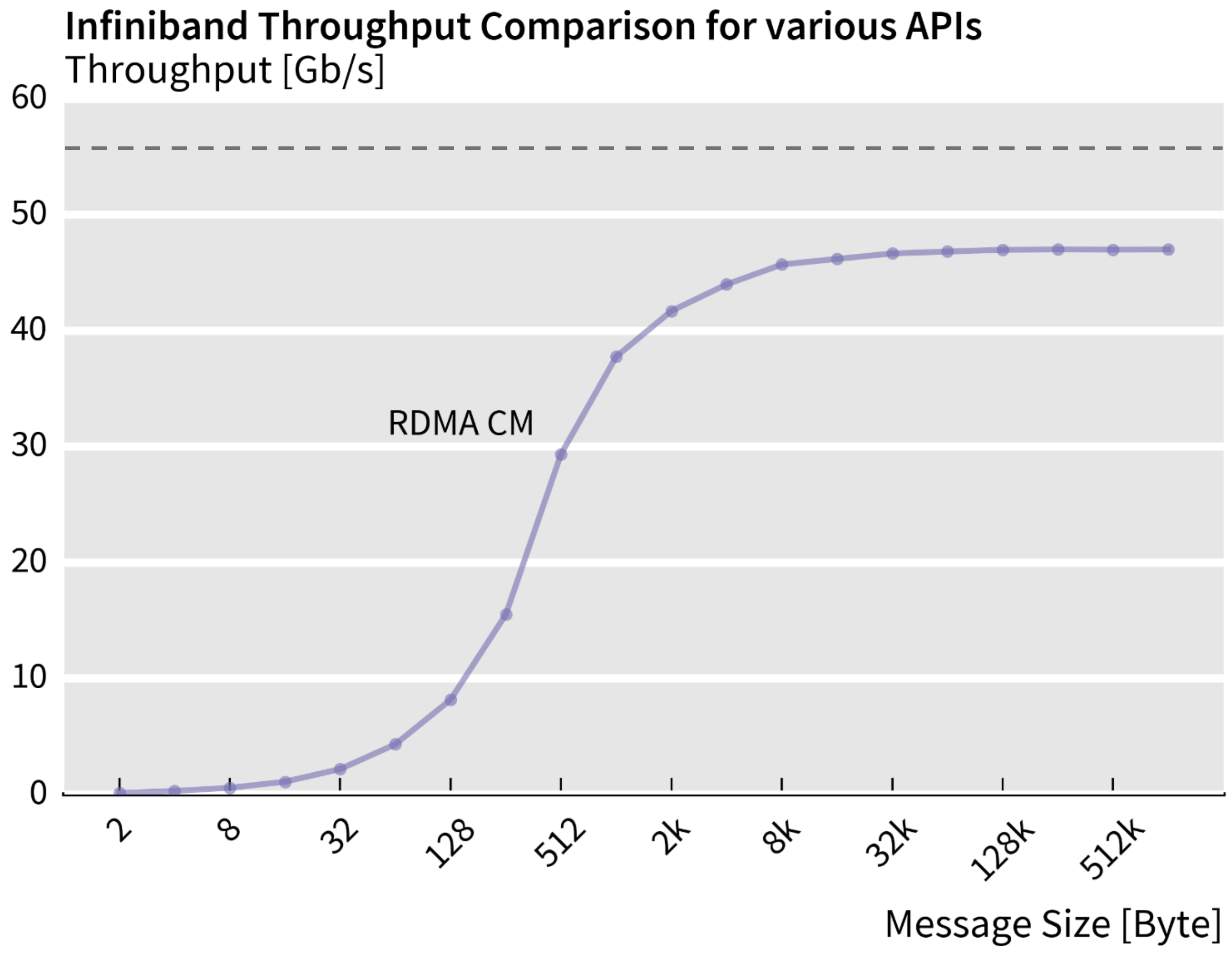
Ethernet and **Infiniband** are the two dominant technologies in the HPC market, **OmniPath** gaining share

ATLAS, ALICE, CMS and LHCb – the four LHC experiments – all use Ethernet and/or Infiniband in their DAQ systems

What API to use for HPC networks in the HEP environment?

Comparison of APIs based on performance and suitability for HEP

Infiniband API Performance



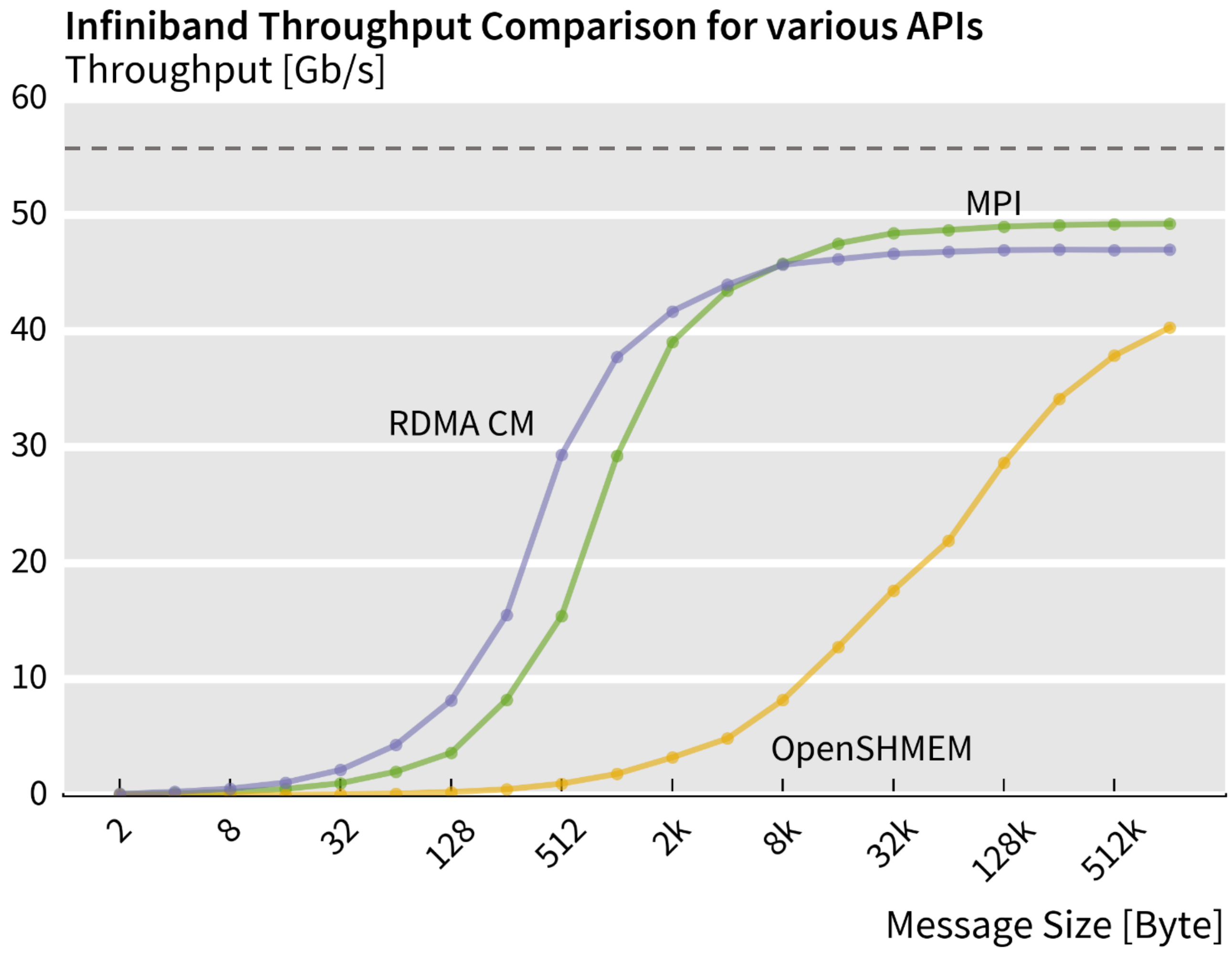
56G Infiniband FDR

Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

Native APIs like Verbs or RDMA CM: Good performance but cumbersome to use and no high-level patterns

Infiniband API Performance



56G Infiniband FDR

Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

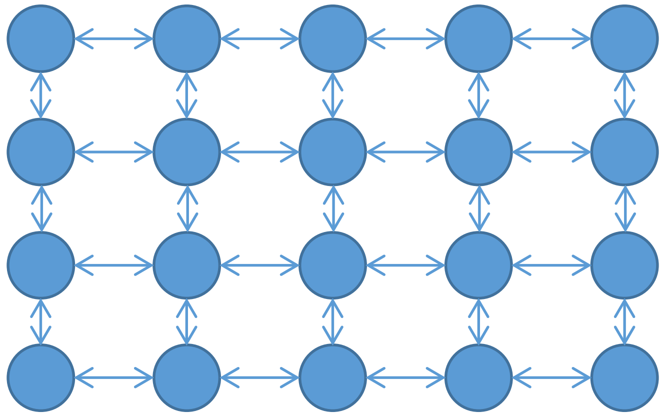
HPC APIs like MPI or PGAS/OpenSHMEM: Good performance (MPI), but paradigm does not fit the HEP use case

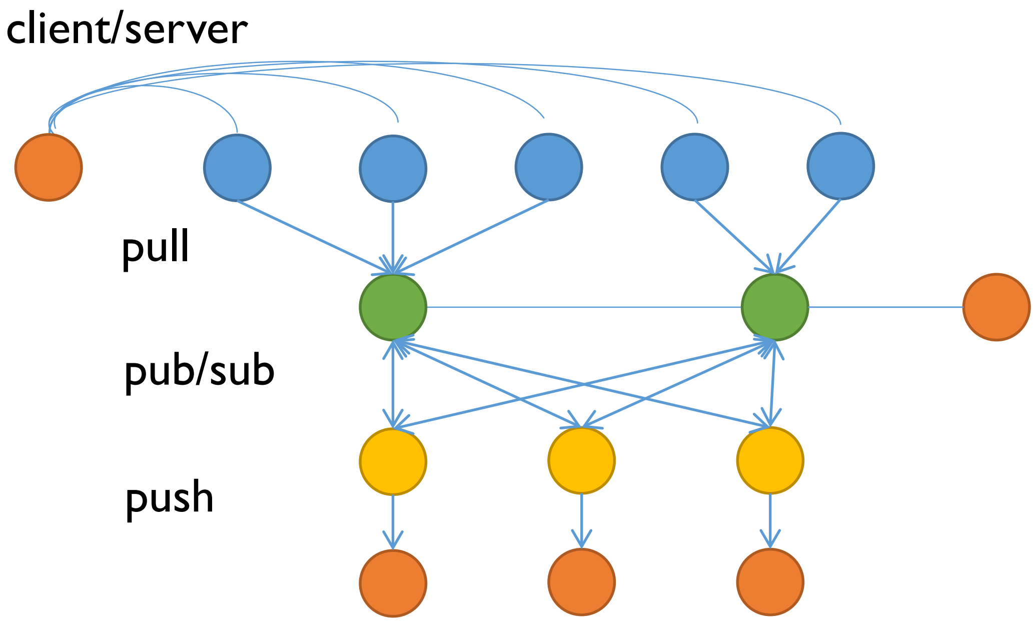
Server clusters in…

HPC

HEP

message passing



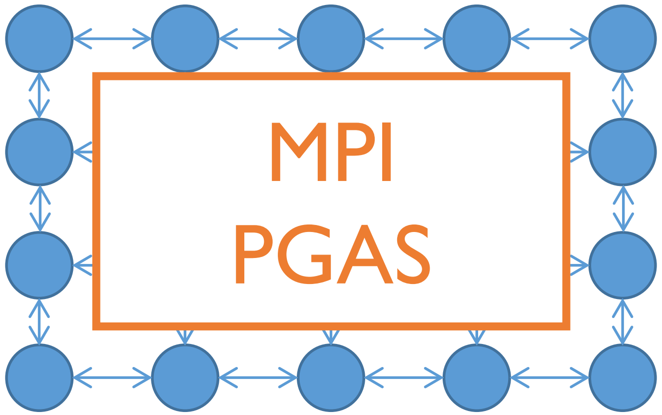


Regular topology
SPMD Pattern
No Real-Time Requirements
No Failure Tolerance
Static Resource Management

Complex topology
Complex distributed system
Real-Time Requirements
Some Failure Tolerance
Dynamic Resource Management

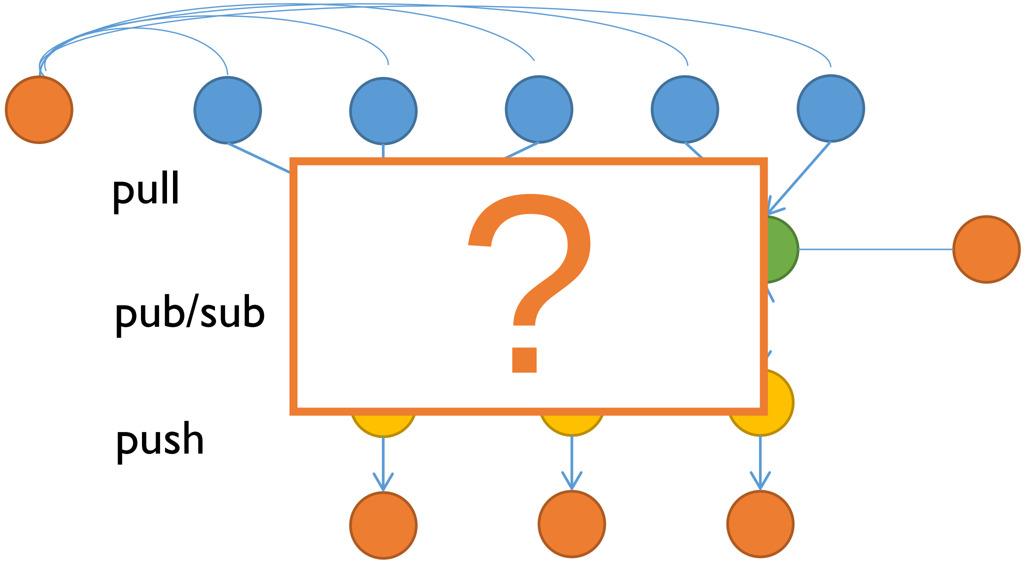
Server clusters in…

HPC



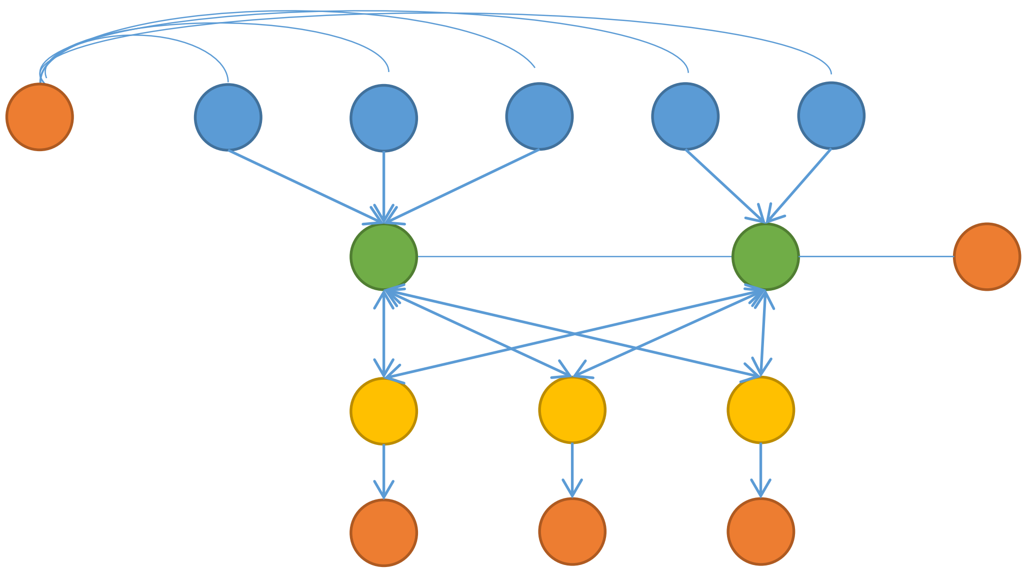
Regular topology
SPMD Pattern
No Real-Time Requirements
No Failure Tolerance
Static Resource Management

HEP



Complex topology
Complex distributed system
Real-Time Requirements
Some Failure Tolerance
Dynamic Resource Management

Requirements



High Throughput (ATLAS Data Acquisition system has to transport more than100 GB/s)

Low Latency connections for detector control and calibration applications

High level communication patterns like client/server and publish/subscribe

Technology agnostic

Closest match for an API satisfying HEP requirements: ØMQ

+

High level communication patterns like publish/subscribe, client/server, push/pull

Simple, clean API

Tuned for low-latency

Can we use ØMQ for HEP purposes? It is already in use, see

Middleware trends and market leaders 2011
A. Dworak, F. Ehm, W. Sliwinski, M. Sobczak, CERN, Geneva, Switzerland

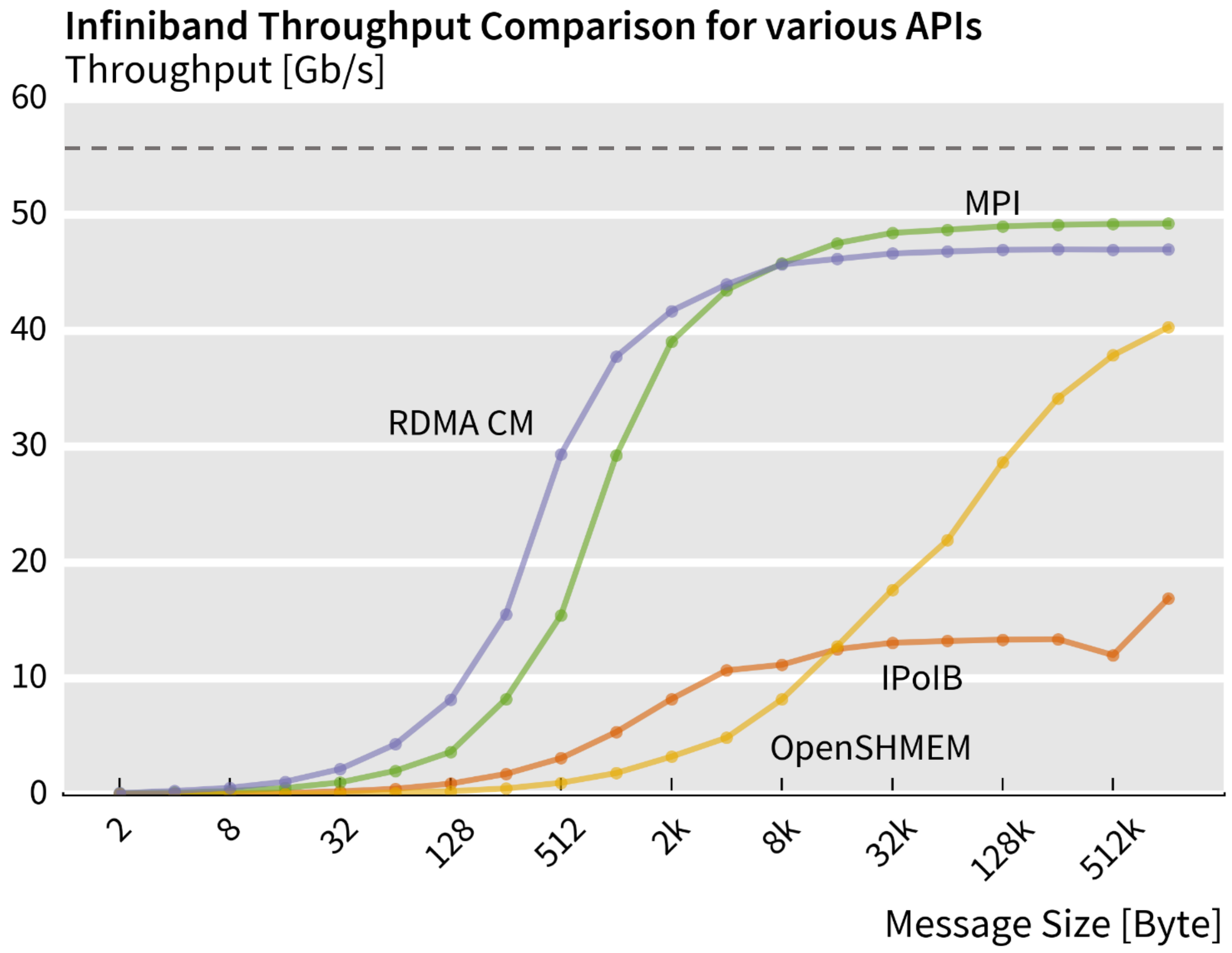
But how does it hold up in a data acquisition context?

-

NO native support for HPC interconnects

NO high-throughput mode

Infiniband API Performance



ØMQ on Infiniband needs to be run on an emulation layer

IPoIB or SDP

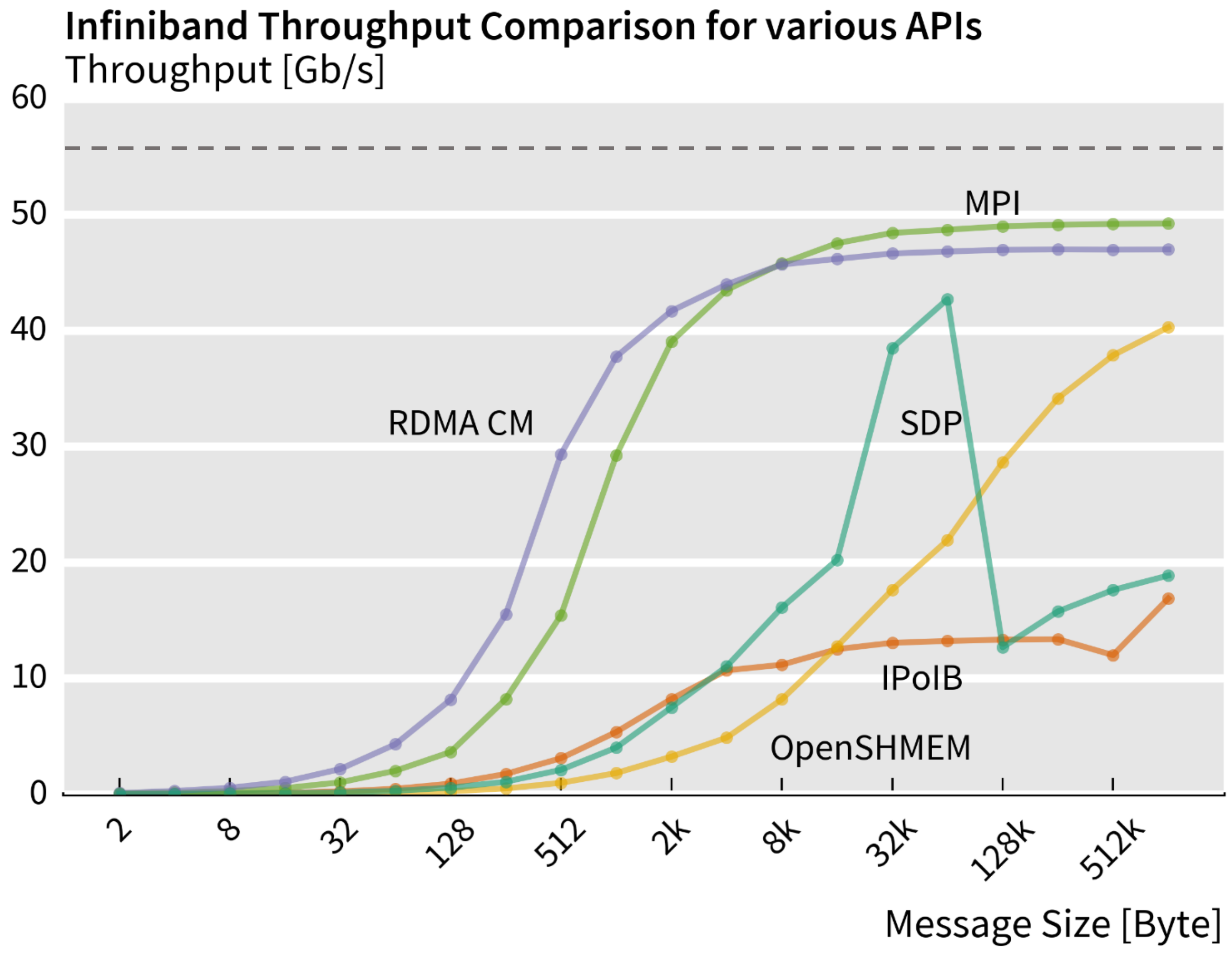
56G Infiniband FDR

Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

IPoIB:
Low performance

Infiniband API Performance



ØMQ on Infiniband needs to be run on an emulation layer

IPoIB or SDP

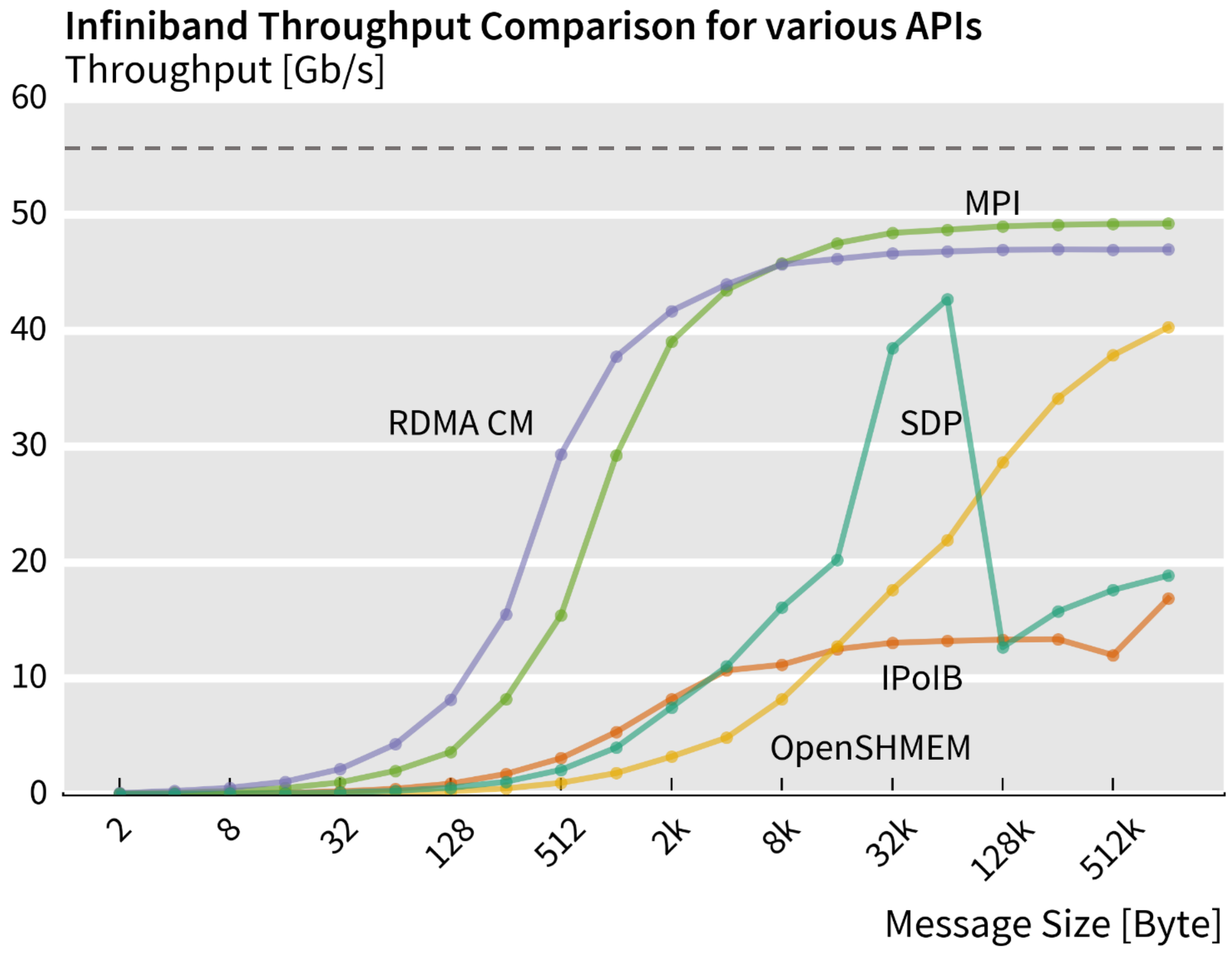
56G Infiniband FDR

Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

Socket Direct Protocol:
Better than IPoIB, but not nearly as good as native APIs

Infiniband API Performance



ØMQ on Infiniband needs to be run on an emulation layer

IPoIB or SDP

56G Infiniband FDR

Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

Emulation layers have a significant performance penalty

~~ØMQ~~

State of the art high-level APIs

Looking at the problem from the other side: What High-Level APIs are offered for us?

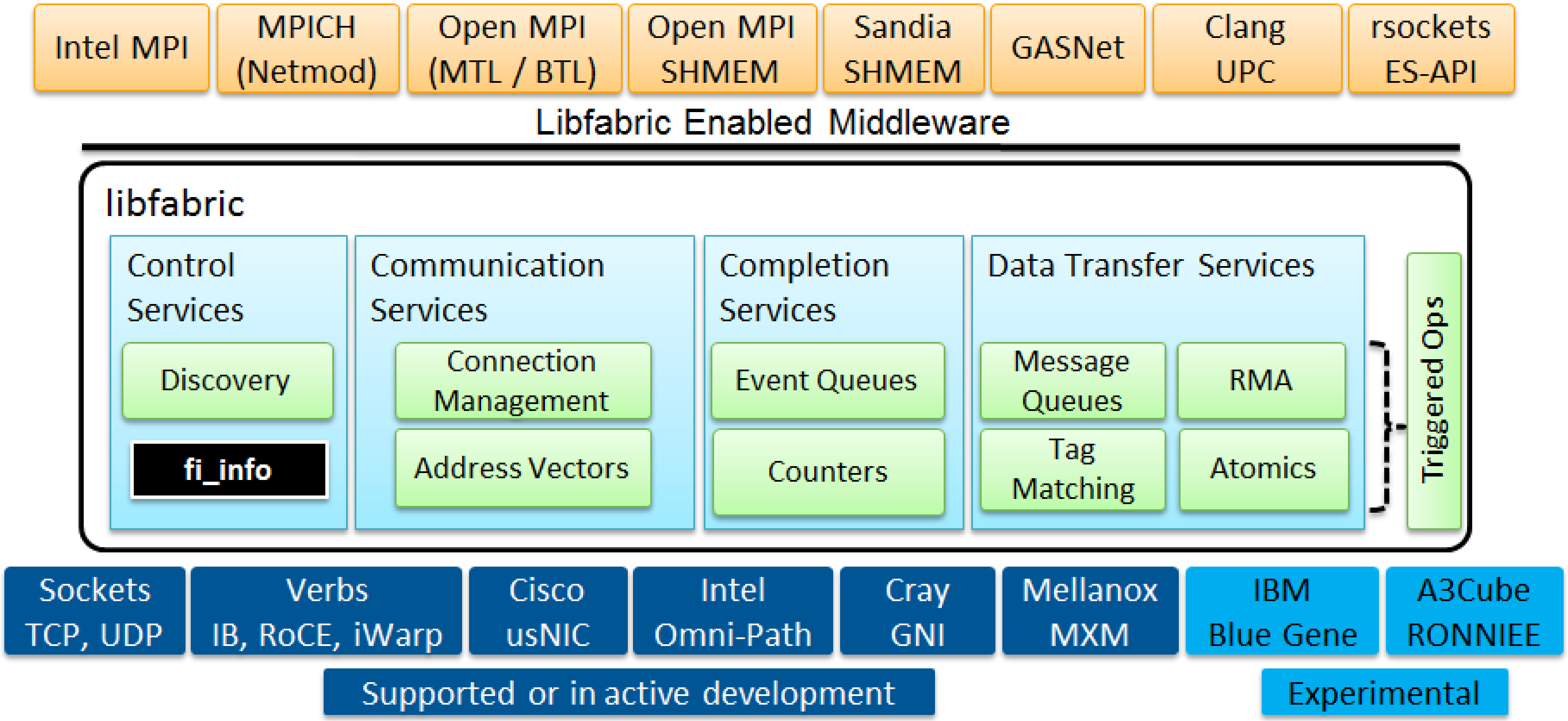


Image source: libfabric manual

State of the art high-level APIs

Looking at the problem from the other side: What High-Level APIs are offered for us?

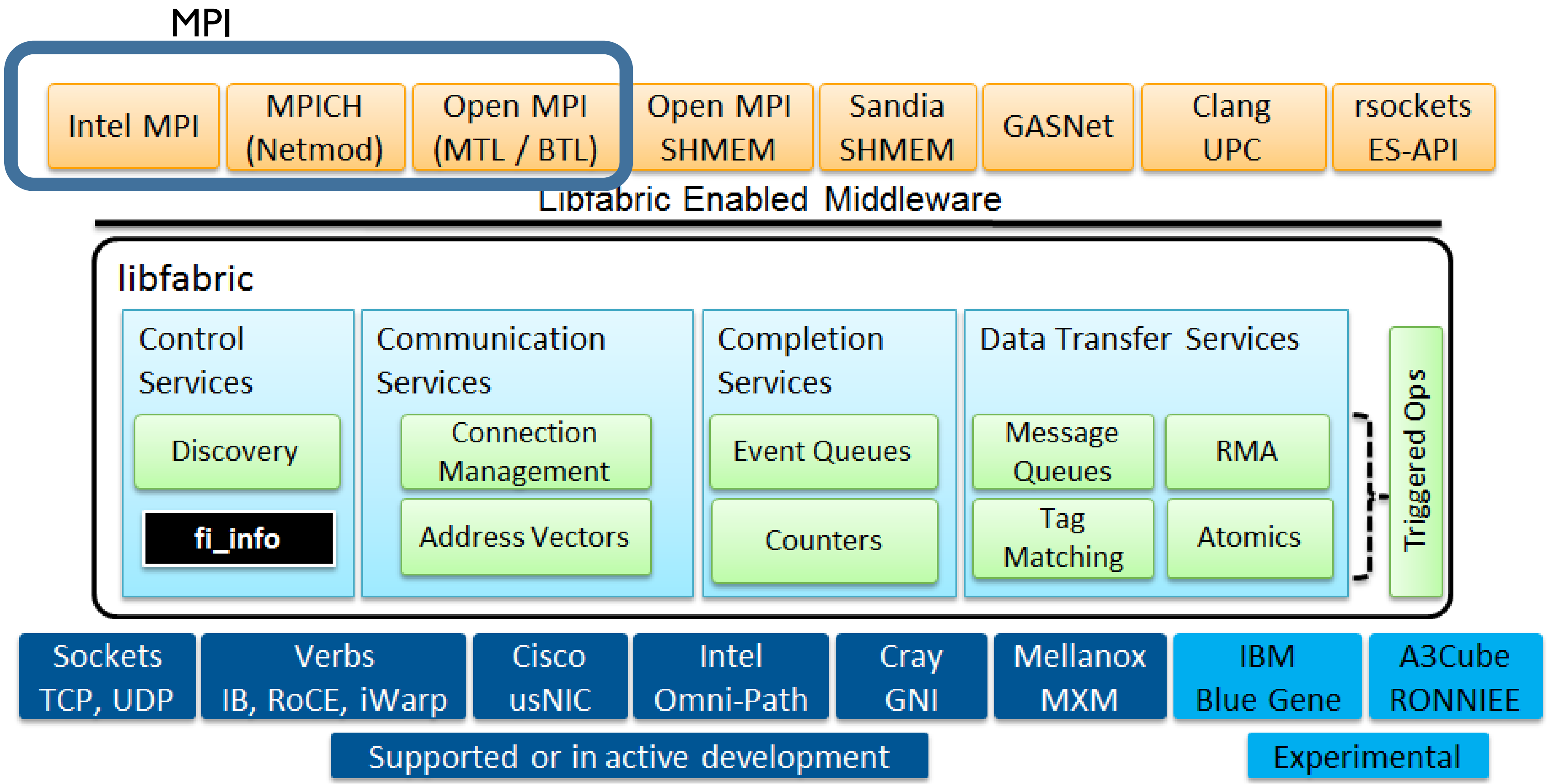


Image source: libfabric manual

State of the art high-level APIs

Looking at the problem from the other side: What High-Level APIs are offered for us?

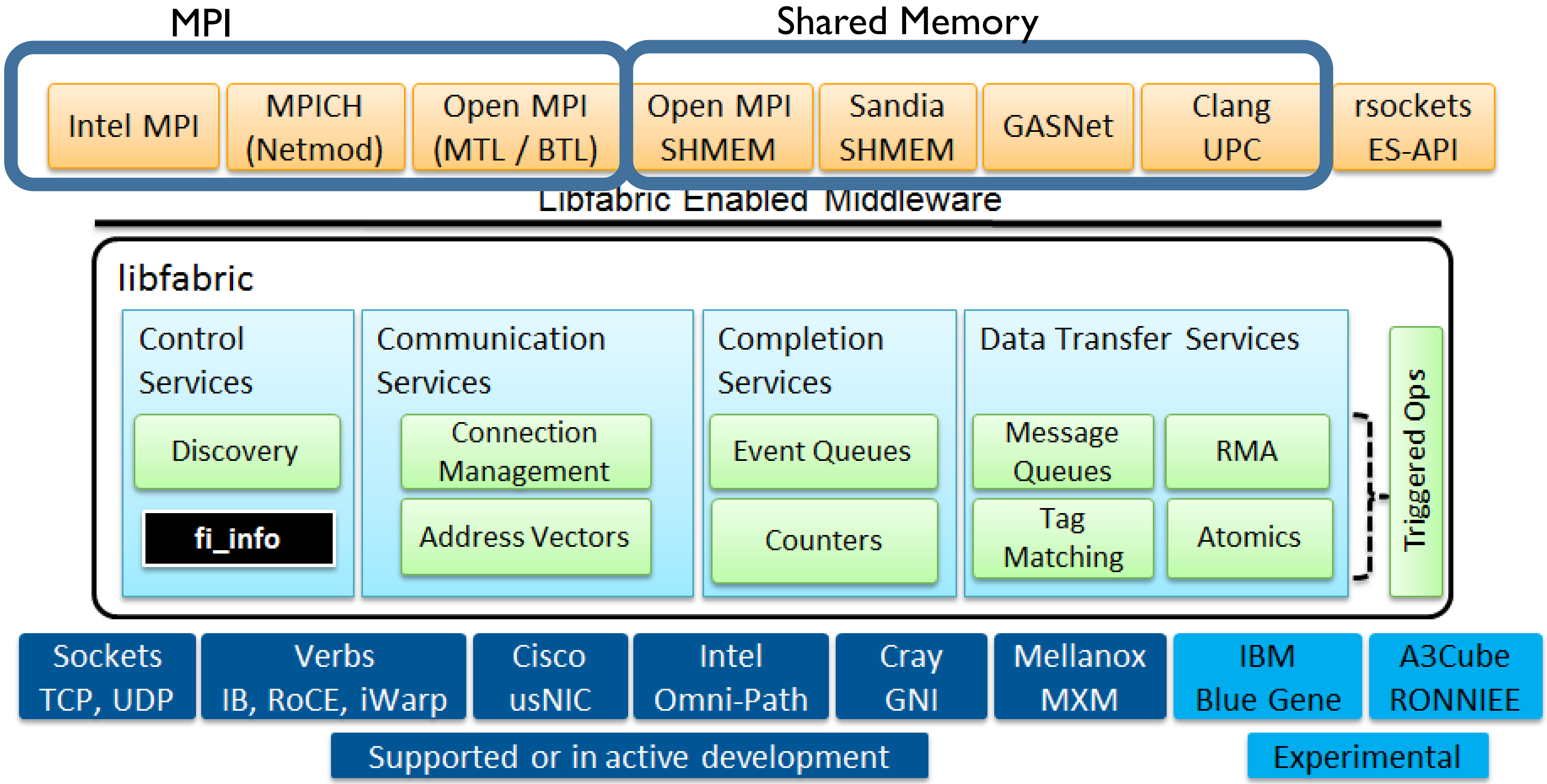


Image source: libfabric manual

State of the art high-level APIs

Looking at the problem from the other side: What High-Level APIs are offered for us?

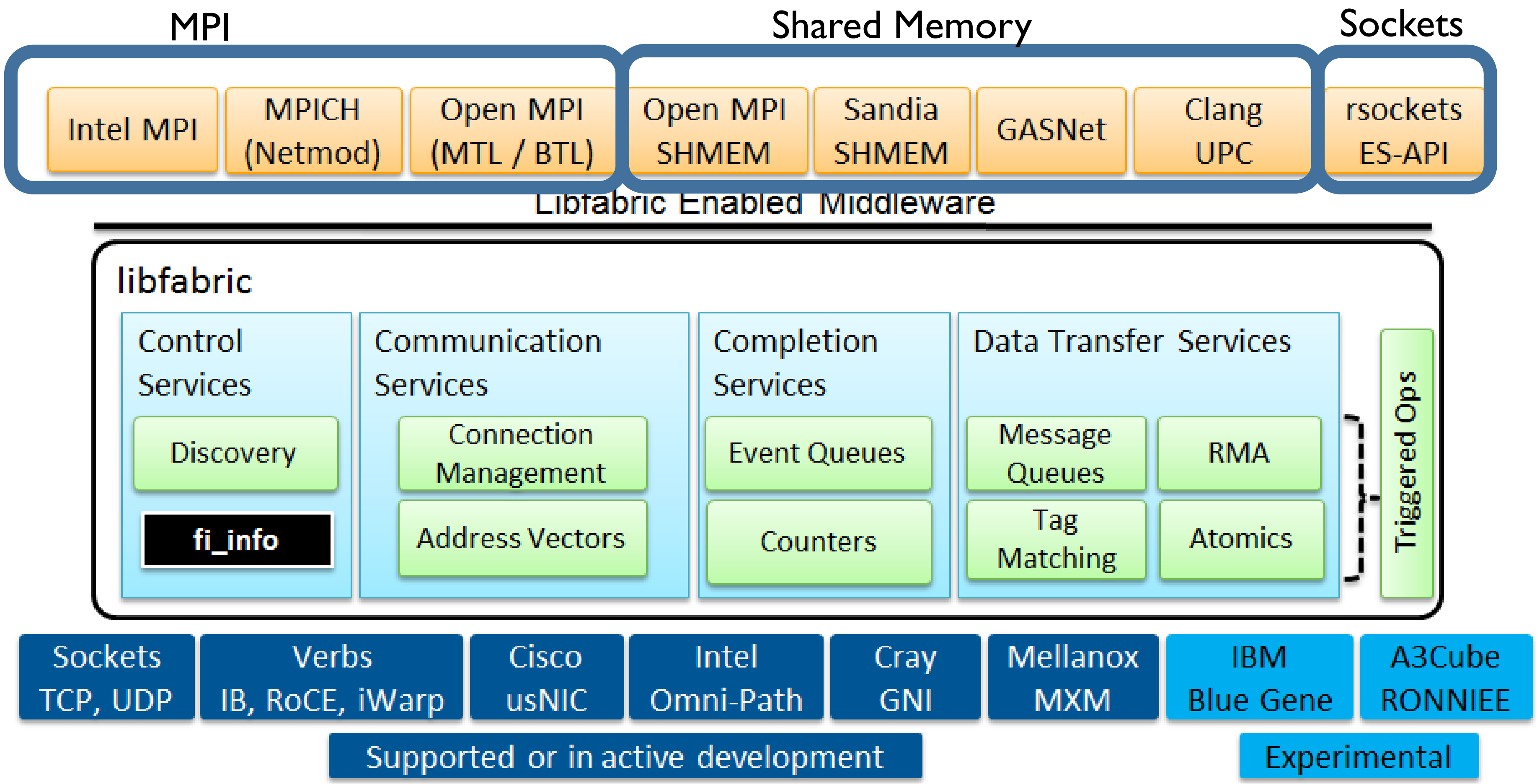


Image source: libfabric manual

State of the art high-level APIs

Looking at the problem from the other side: What High-Level APIs are offered for us?

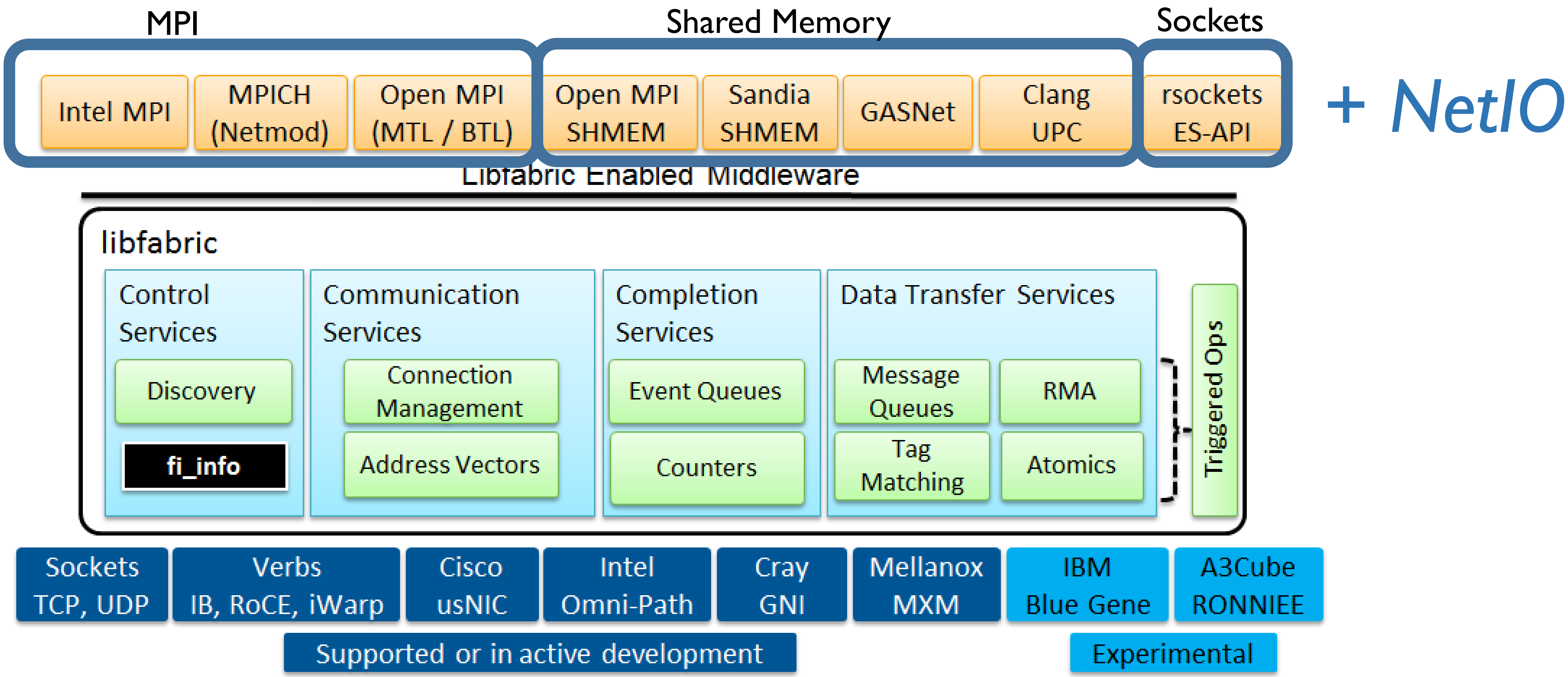


Image source: libfabric manual

NetIO

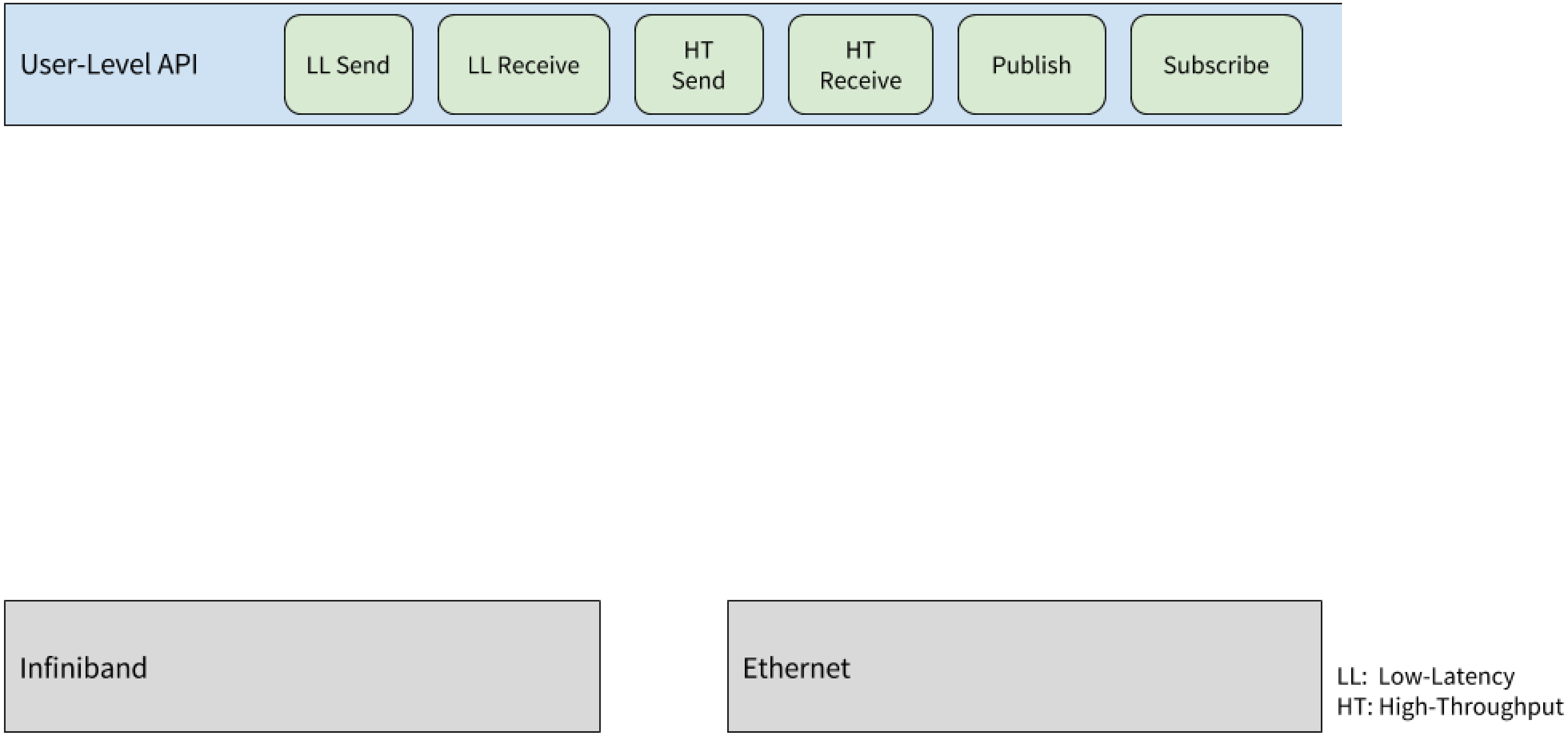
A high-level, general-purpose API for HPC networks

NetIO was designed with High Energy Physics experiments in mind, but it is not restricted to this use case

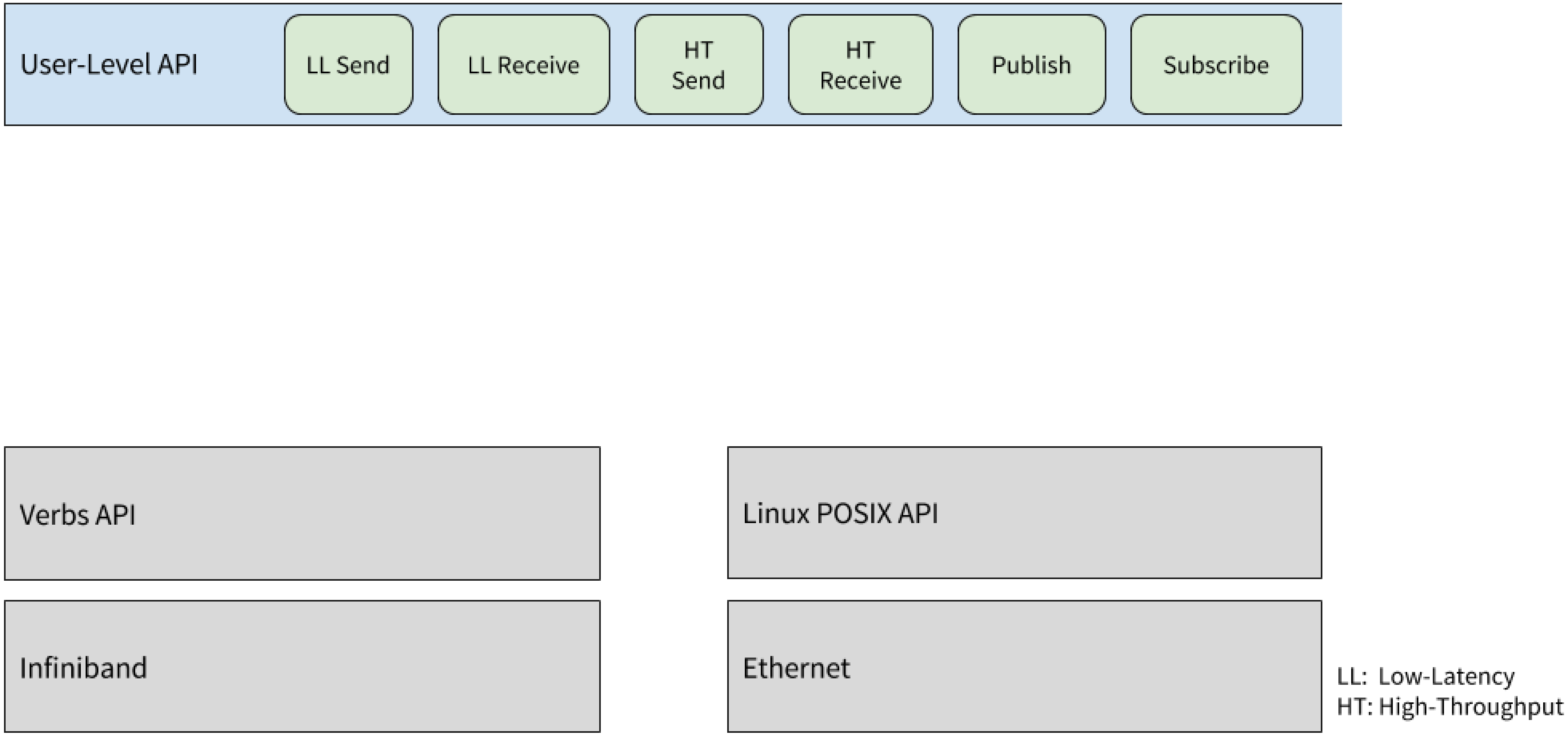
Design Goals:

- Native support for HPC interconnects via a back-end system
- Different operation modes tuned for high-throughput communication or low-latency communication
- High-level communication patterns including publish/subscribe

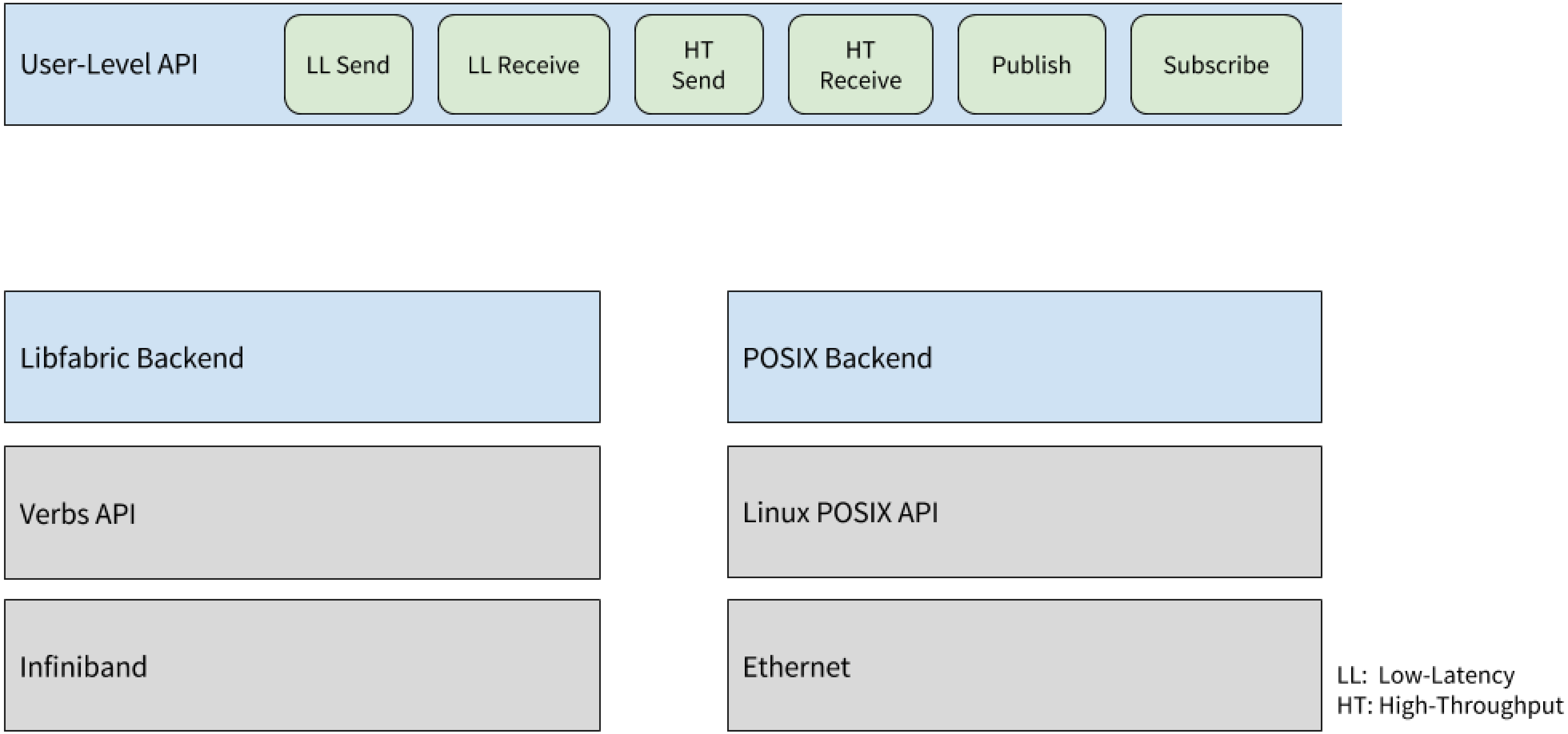
NetIO Architecture



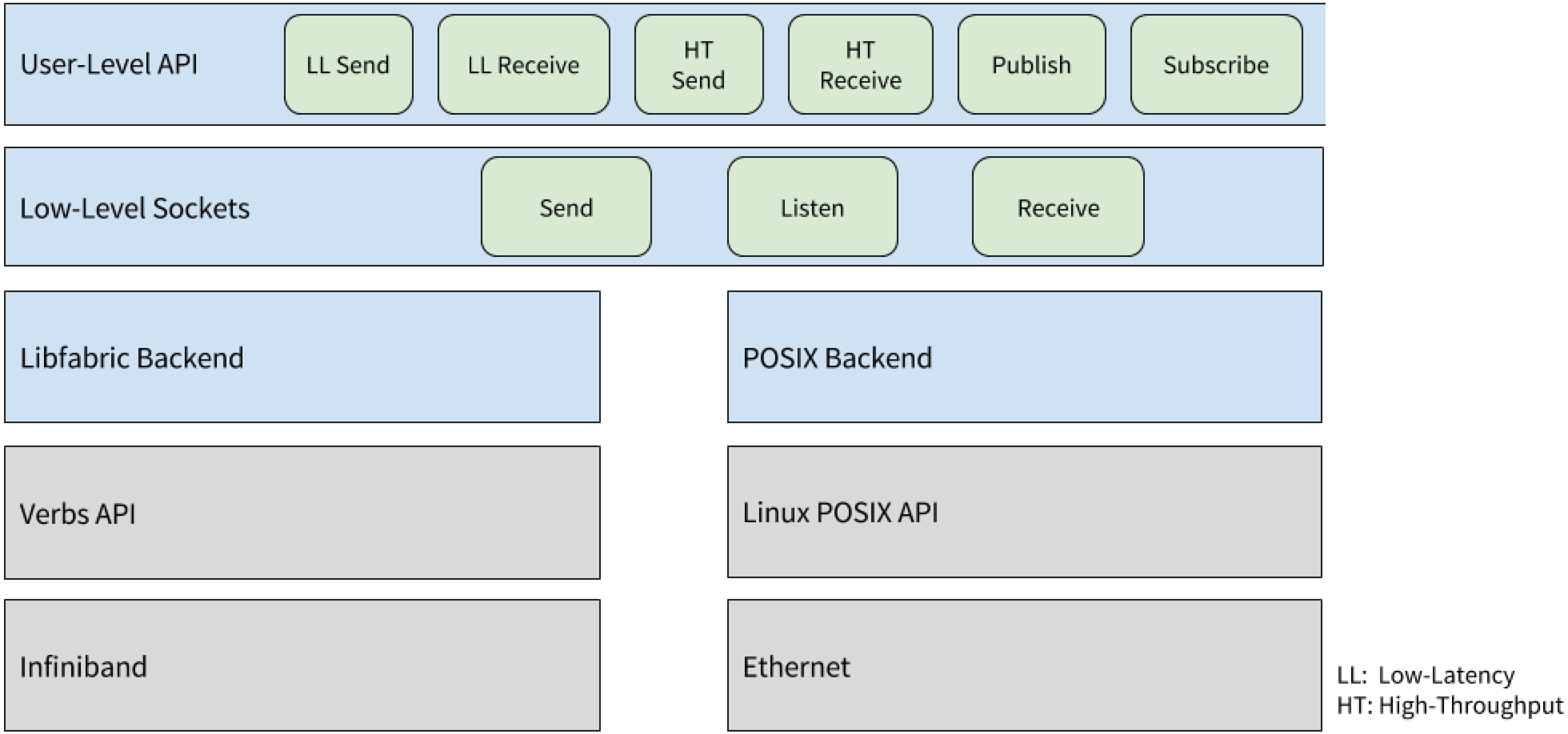
NetIO Architecture



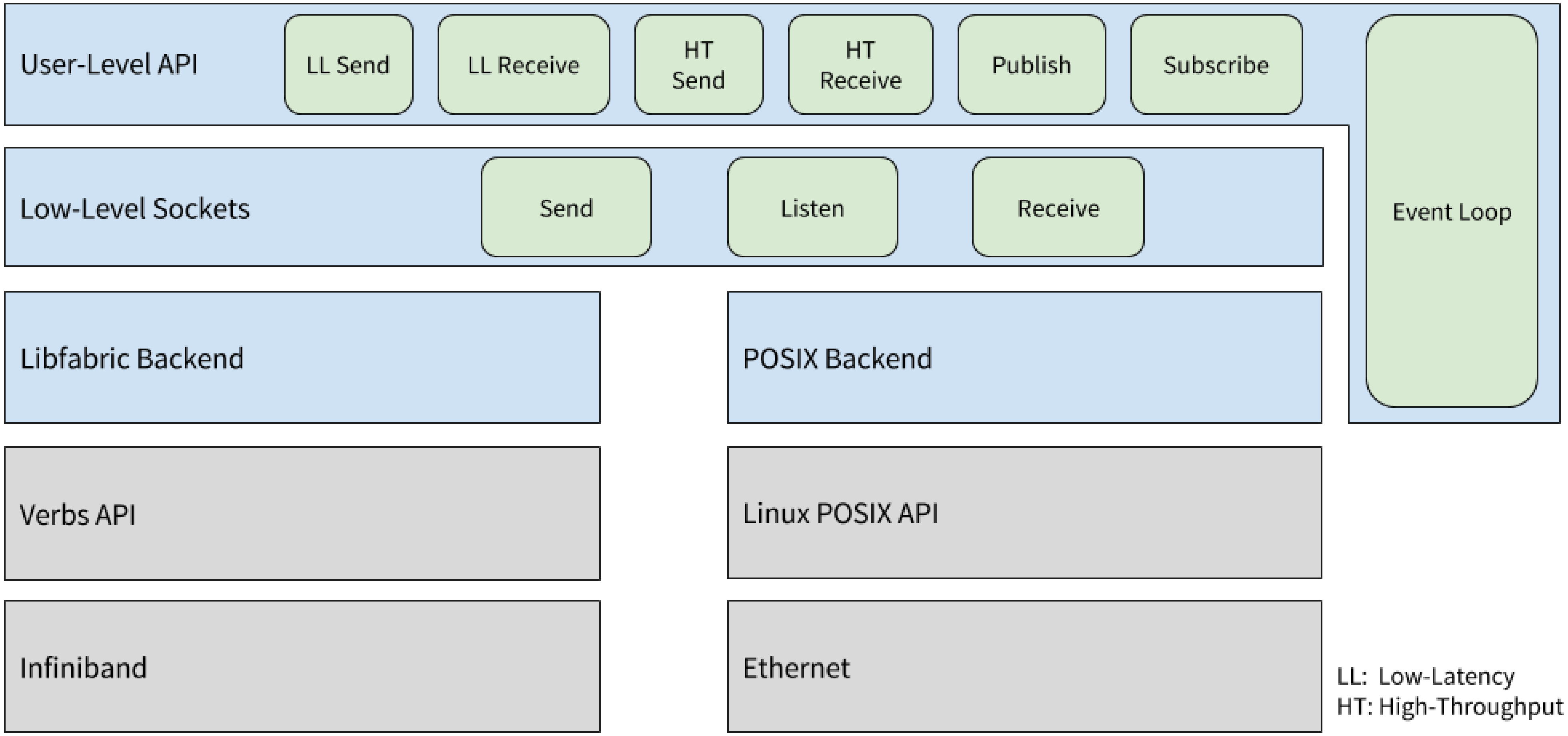
NetIO Architecture



NetIO Architecture



NetIO Architecture



Memory Management

Messages are packed into **pages** (buffering for higher efficiency)

Typical max. page size is 1 MB

Event loop drives a timeout to send out partial pages and avoid connection starvatation

NetIO maintains a list of pre-allocated, free pages per connection

Default: up to 256 pages per connection

Pages are recycled after having been processed (i.e. fully sent or received)

Low-level sockets

Uniform interface used by user-level sockets

Abstract interface that is implemented by back-ends

Pages: Buffers that contain one or multiple messages

Listen Sockets: Listen to incoming connection requests, create receive sockets

Receive Sockets: Receive pages from remote endpoints, deserialize into messages

Send Sockets: Send pages to remote endpoint.

No distinction between high-throughput and low-latency communication (this is done in the user-level sockets)

Configuration interface to enable fine-tuning of connection parameters

User-level sockets

Provide a simple interface for users

High-level communication patterns:

- Send/Receive
- Publish/Subscribe

Come in two flavors:

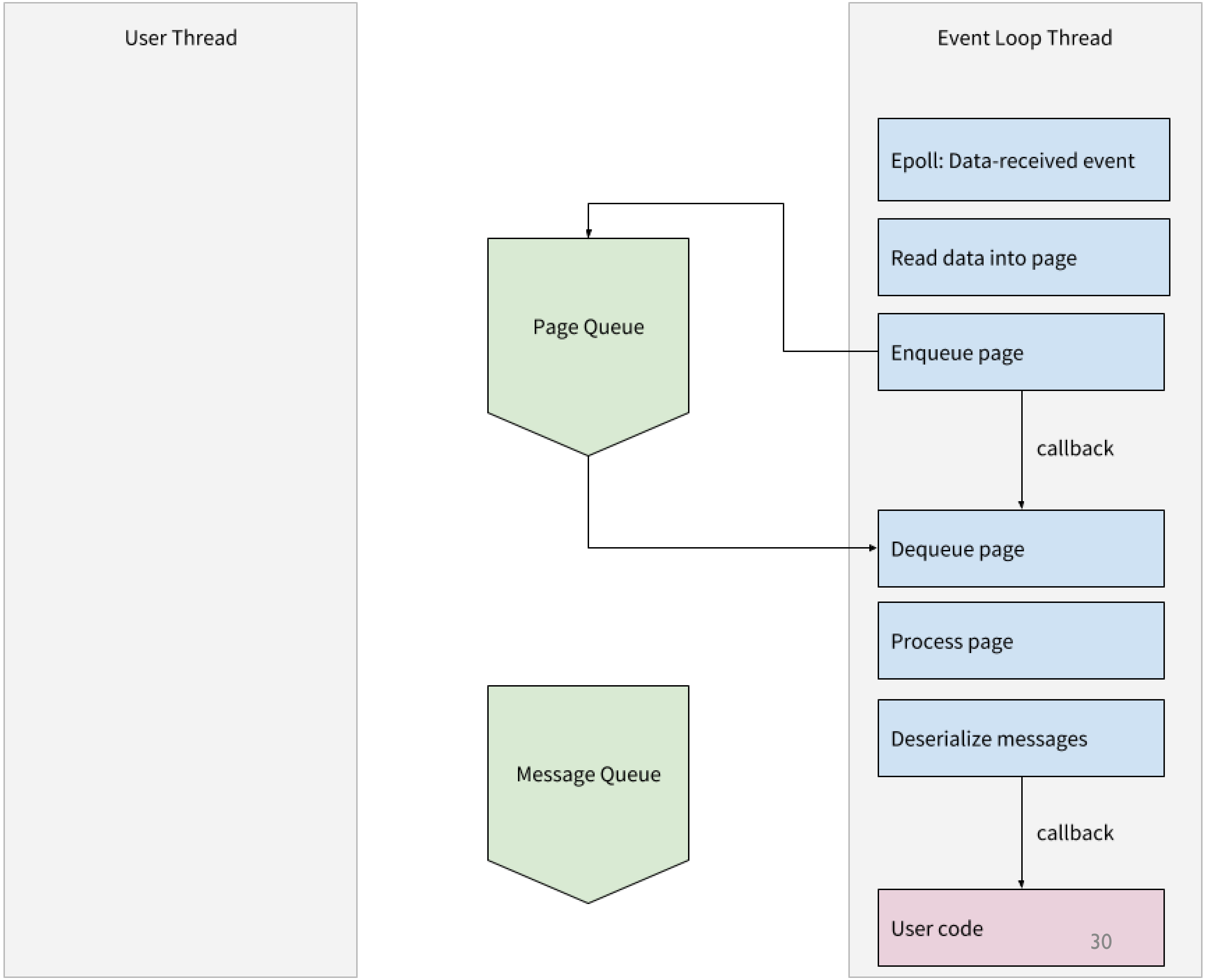
- Low-latency
- High-Throughput

Addressing based on IPv4 or IPv6

Low-Latency • **Callback-based** • **No buffering**	**High-Throughput** • **Queue-based** • **Buffering**
LL Send socket	HT Send socket
LL Receive socket	HT Receive socket
LL Subscribe socket	HT Subscribe socket
Publish socket *Both high-throughput and low-latency subscribe sockets can connect*	

Low Latency Mode

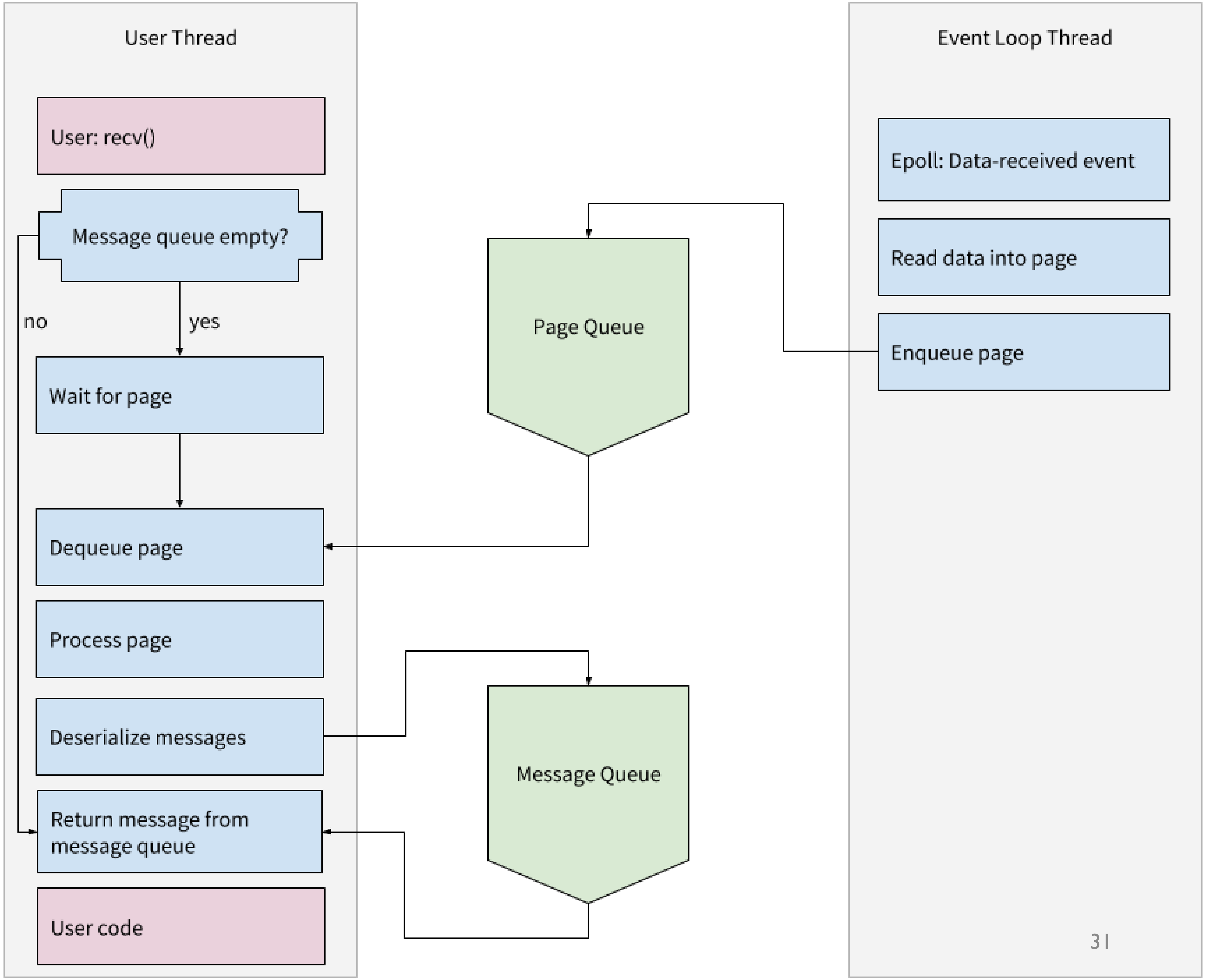
Low latency
- No thread synchronization
- No buffering, pages contain a single message
- Skipping message queue
- Immediate handling of messages via callbacks
- In the future: also skip page queue



High Throughput Mode

High Throughput

- Minimal work in the event loop so it can return to process incoming pages
- Buffering: Multiple messages per page
- Event-loop drives buffer timeout to avoid connection starvation
- Explicit user call to retrieve messages



Back-ends

POSIX

Uses POSIX stream sockets (TCP), which translates naturally into the low-level socket API

Nagle's algorithm disabled
(buffering in user-level sockets)

Simple integration with epoll event loop

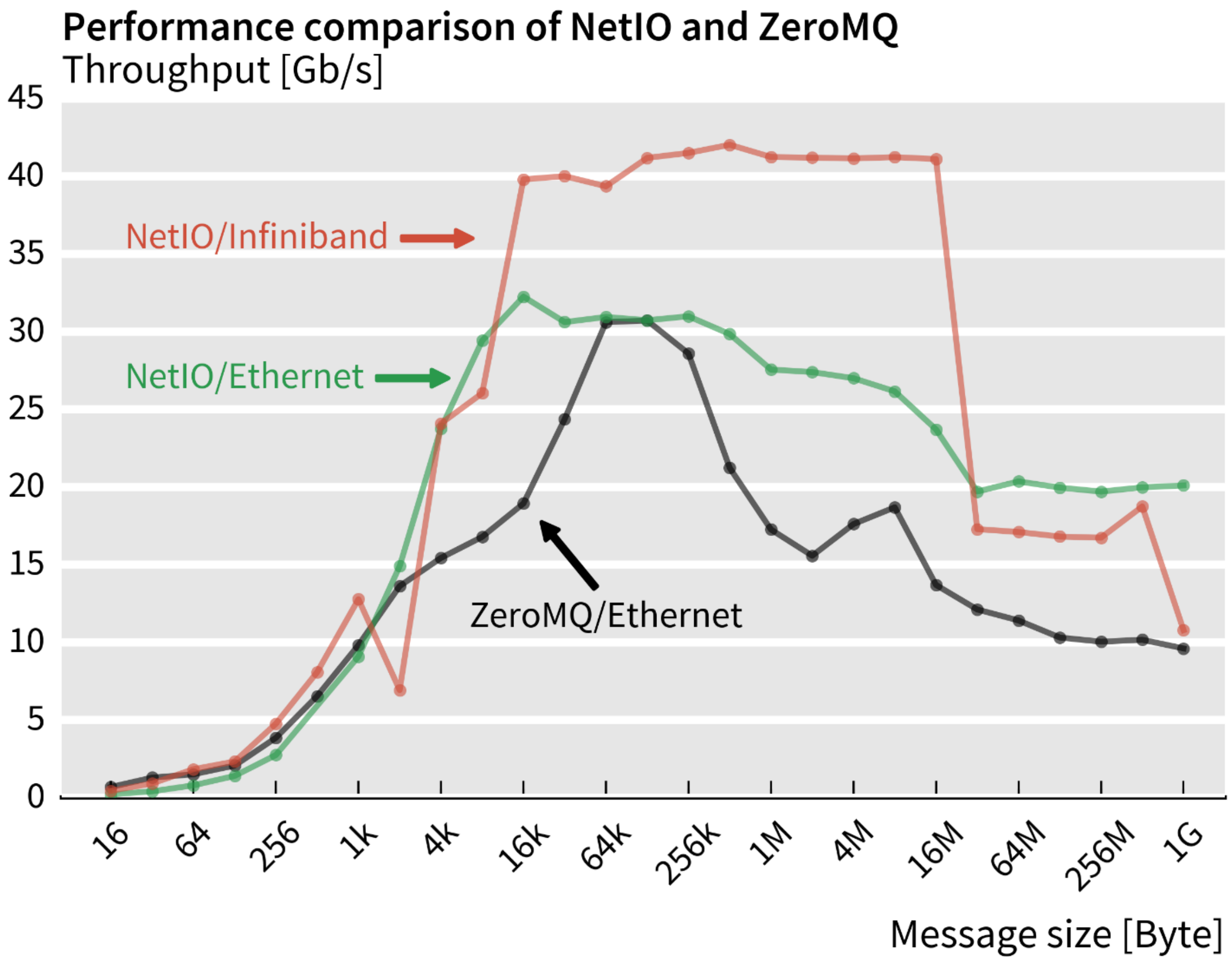
Libfabric

Uses libfabric Reliable Connection (RC)

RDM mode is not supported – ordering is needed to ensure proper deserialization
(That means currently RDM-based libfabric providers are not supported, for example the PSM provider for OmniPath. OmniPath is instead supported by the Verbs provider)
Send windows used to control data-flow for higher throughput

Uses file descriptors for asynchronous completion management – can be integrated in the epoll event loop

NetIO Throughput: Push/Pull



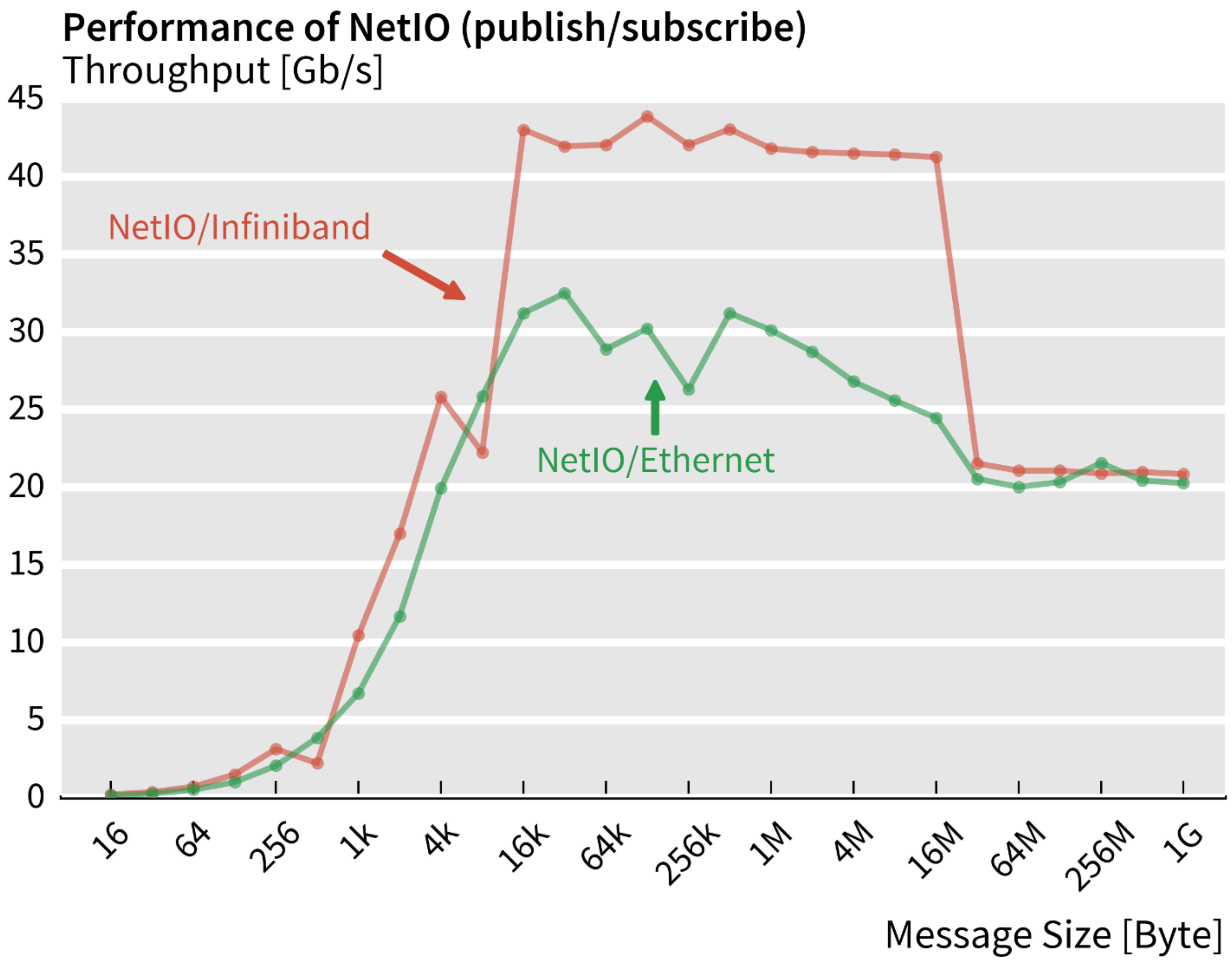
Push/Pull benchmarks

56G FDR Infiniband
40G Ethernet
1 MB pagesize

NetIO outperforms ZeroMQ in nearly all uses cases

Using the Infiniband mode of the underlying hardware allows a performance boost that we can leverage with NetIO – without changing our software

NetIO Throughput: Publish/Subscribe



Publish/Subscribe benchmarks

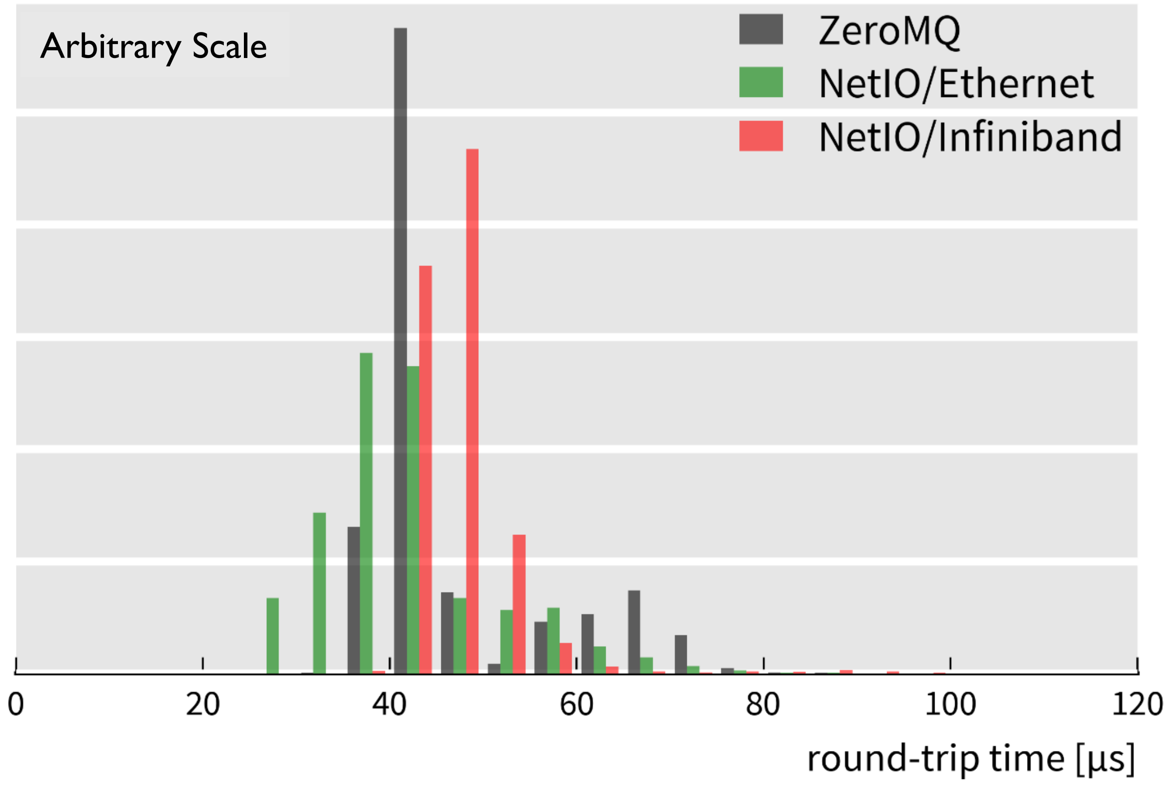
56G FDR Infiniband
40G Ethernet
1 MB pagesize

Similar to the push/pull benchmarks, using the Infiniband mode of the hardware yields a performance boost

ZeroMQ already discarded due to limited performance

NetIO Latency

Round-Trip Time (RTT) Comparison



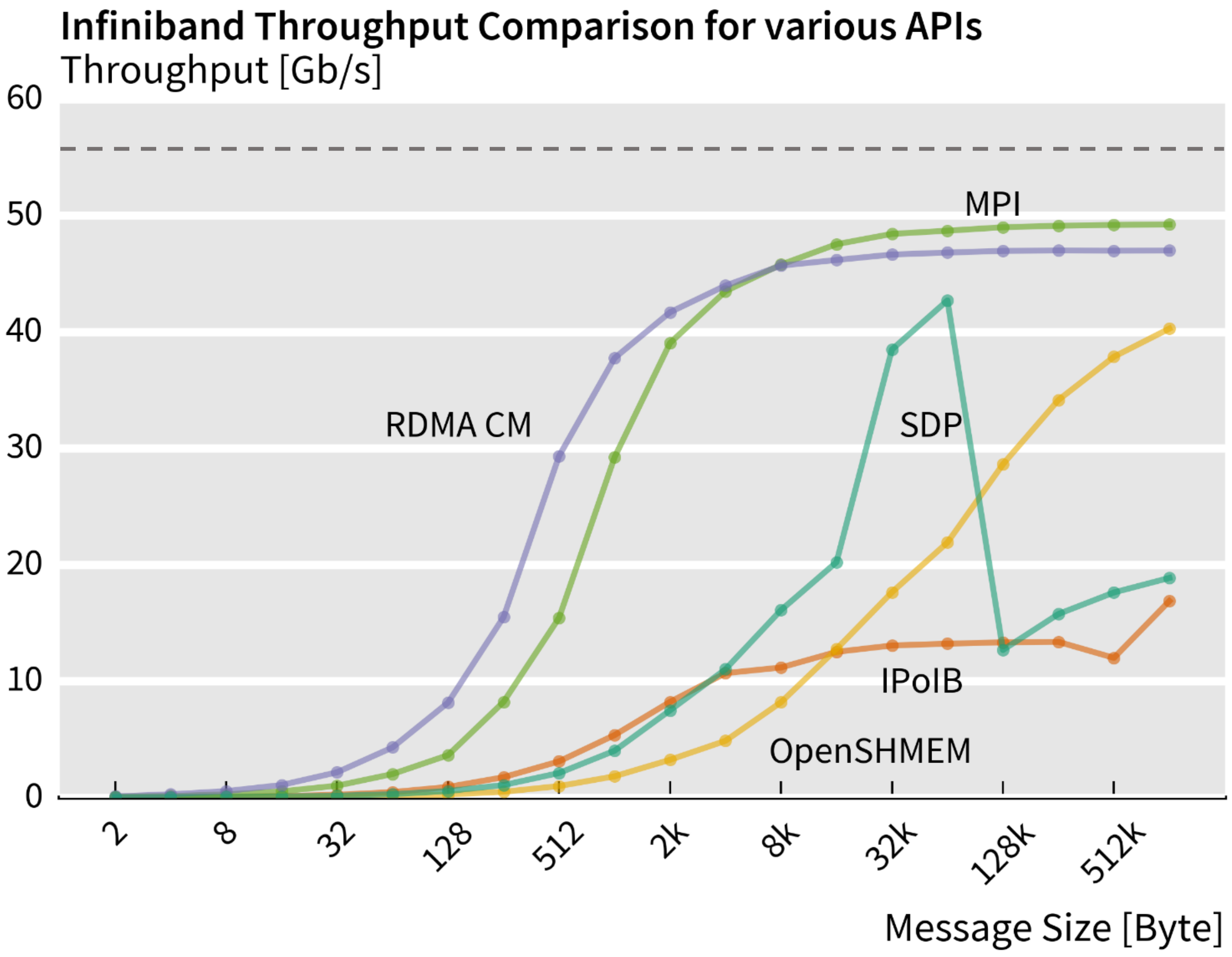
Point-to-Point benchmarks
No additional load on switch

56G FDR Infiniband
40G Ethernet

Latency is very similar for ZeroMQ and NetIO.

Lower latency expected for NetIO/Infiniband: room for improvement

NetIO compared to other Infiniband APIs



56G Infiniband FDR

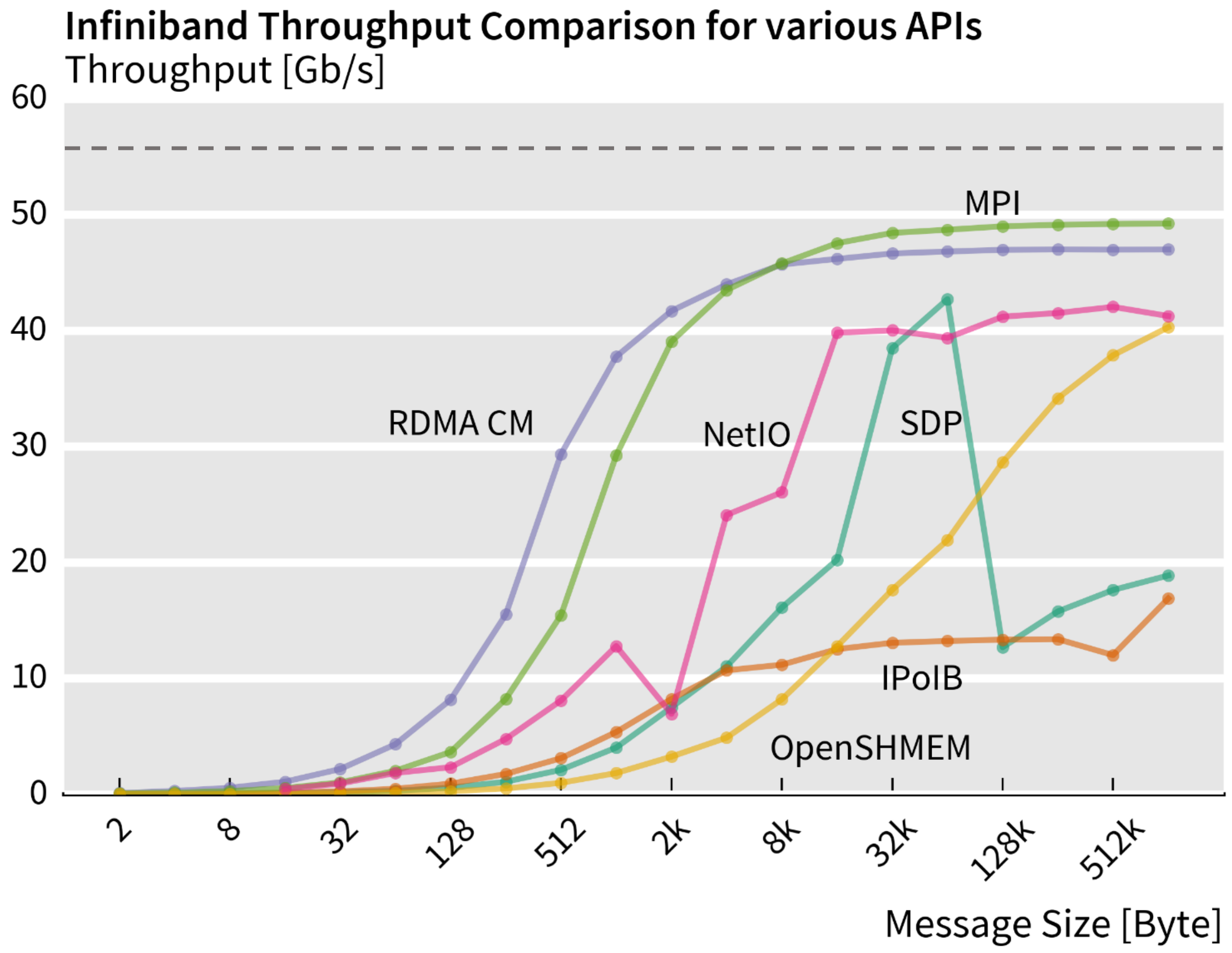
Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

NetIO performance exceeds the performance of emulation layers

Still some room for improvement compared to MPI/native APIs

NetIO compared to other Infiniband APIs



56G Infiniband FDR

Data-transfer between two nodes (Intel Haswell, 8-core and 10-core systems)

Connected via a single switch

NetIO performance exceeds the performance of emulation layers

Still some room for improvement compared to MPI/native APIs

NetIO Status & Outlook

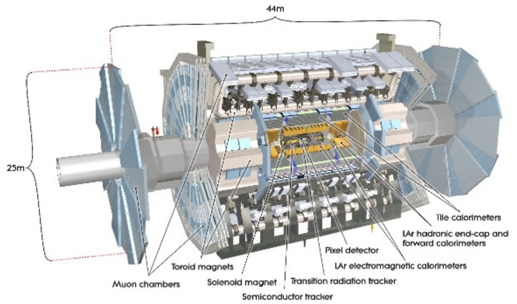
Some performance improvements planned

- New ZeroCopy mode
- Improved queuing scheme

Status

- Small functional improvements needed
- Ongoing parameter studies
- User documentation being written
- OpenSource release planned this year
- NetIO going to be used in ATLAS data-taking beginning 2019

Conclusion

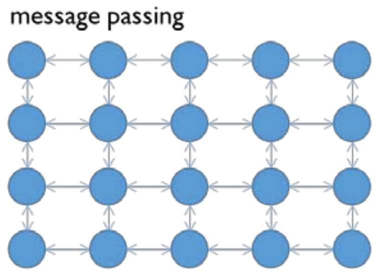


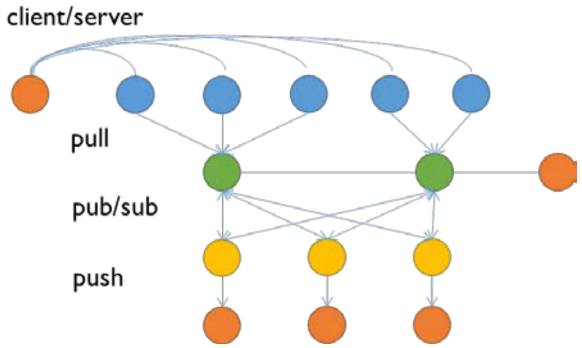
HPC

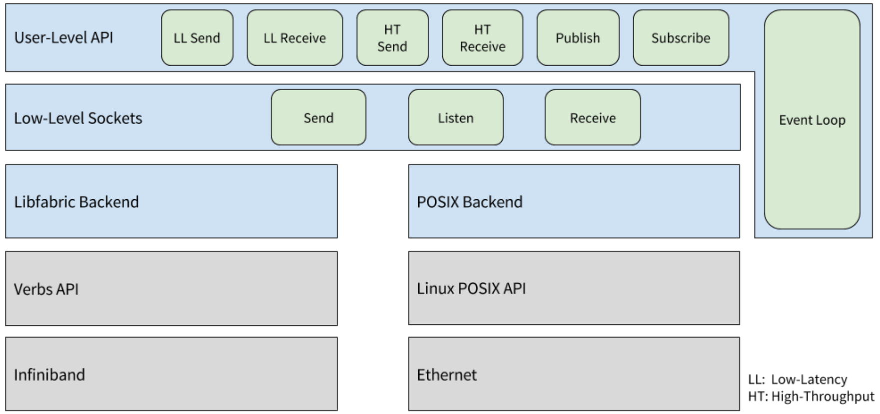
HEP

HPC interconnects are interesting technologies for HEP

HPC is fundamentally different from HEP computing and different network APIs are required







NetIO is network API with high-level communication patterns and native support for Ethernet and HPC interconnects via a pluggable back-end system

To be used in ATLAS readout from 2019 on