August 20-22, 2025

Libfabric: API tutorial

Rajalaxmi Angadi

Alexia Ingerson

Agenda

01 Overview

02 Recap

03 Peer API / LNX

04 UEC and Libfabric

05 Conclusion

Agenda

01 Overview

02 Recap

03 Peer API / LNX

04 UEC and Libfabric

05 Conclusion

Why do we need Libfabric

Applications evolve

- Application – HPC/AI
- Parallel Programming Models / middlewares
- Network Software
- Network Hardware

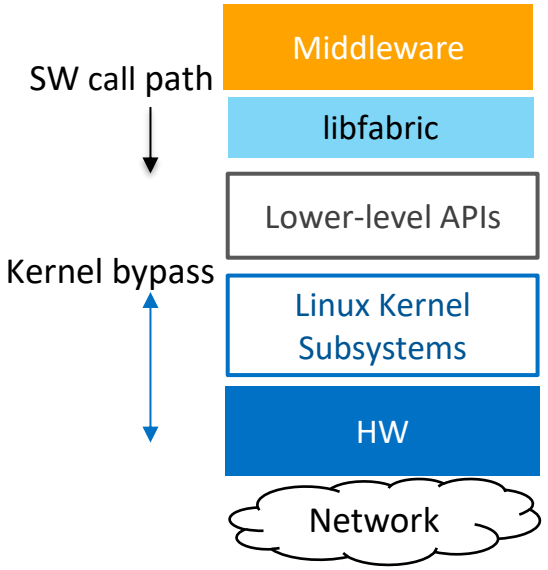
FUJITSU Mellanox TECHNOLOGIES MYRICOM Network Products
Bull atos technologies CRAY IBM intel

Hardware evolve

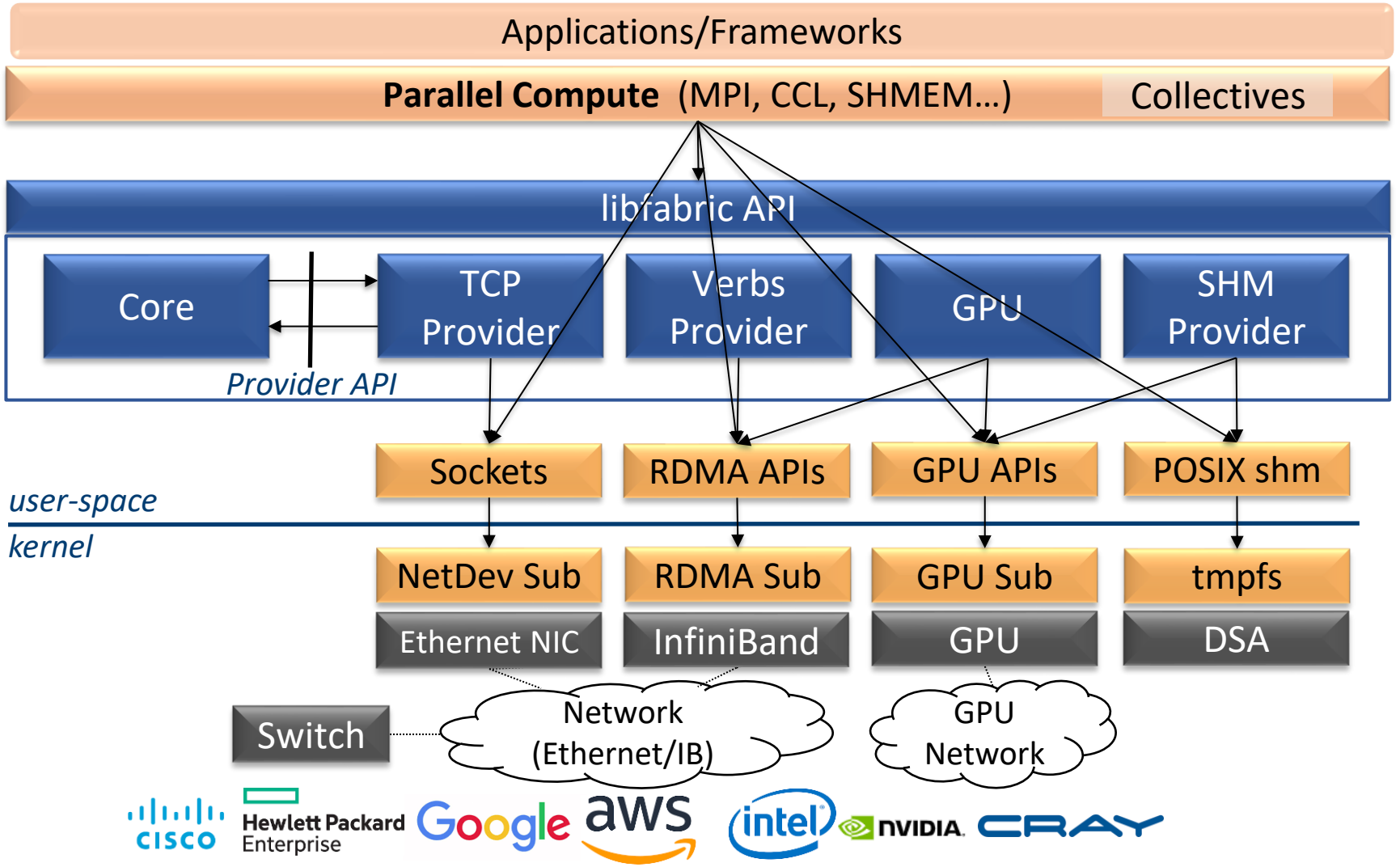
Need a standard way for developers

Libfabric

- Libfabric (OFI – Open Fabrics Interface) defines a communication API for high-performance parallel and distributed applications
- Enables a tight semantic map between applications and underlying fabric services
- Designed for performance and scalability requirements of applications running in a tightly coupled network environment scaling to thousands of nodes
- The key components of OFI: application interfaces, provider libraries, kernel services, daemons, and test applications
- Targets supported: Linux, Free BSD, Windows, and OS X



Libfabric


Applications/Frameworks
Parallel Compute (MPI, CCL, SHMEM…)
Collectives
libfabric API
Core
TCP Provider
Verbs Provider
GPU
SHM Provider
Provider API
Sockets
RDMA APIs
GPU APIs
POSIX shm
user-space
kernel
NetDev Sub
RDMA Sub
GPU Sub
tmpfs
Ethernet NIC
InfiniBand
GPU
DSA
Switch
Network (Ethernet/IB)
GPU Network
cisco
Hewlett Packard Enterprise
Google
aws
intel
NVIDIA
CRAY

Libfabric

MPI | SHMEM | Libfabric-enabled Applications | DAOS | CCL

Libfabric API

Control
- Discovery

Communication
- Connection Management
- Address Vectors

Completion
- Event Queues
- Completion Queues
- Counters

Data Transfer
- Messaging
- Atomics
- RMA
- Collectives
- Tag Matching

Utility providers – RxM, RxD, Lnx, Hook, ...

OFI provider

Verbs (InfiniBand)	TCP (sockets)	OPX (Cornelis Omni-Path)	UCX
EFA (AWS)	SHM (shared memory)	CXI (Cray Slingshot)	UET

NIC | GPU | CPU

Agenda

01 Overview

02 Recap

03 Peer API / LNX

04 UEC and Libfabric

05 Conclusion

Libfabric – Build and Install

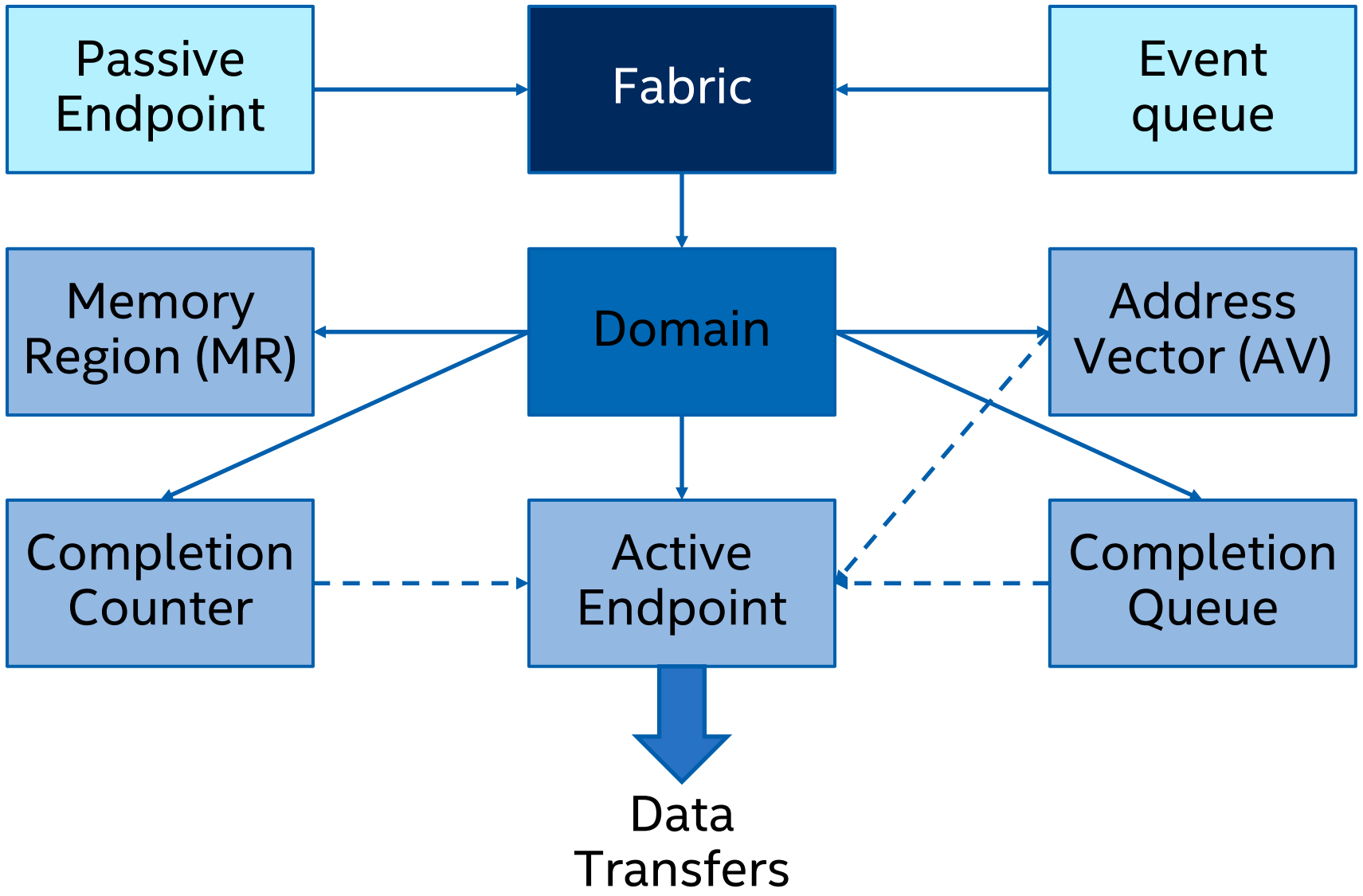
- Available in major distros but recommended to build from source to get latest updates
- Releases found on Github: https://github.com/ofiwg/libfabric/releases
  - Latest release v2.2.0
- Uses autotools for building

```
./autogen.sh
./configure --prefix=<install_dir>
make -j 32 install
```

- Optional: install included fabtests/ for testing and validation:

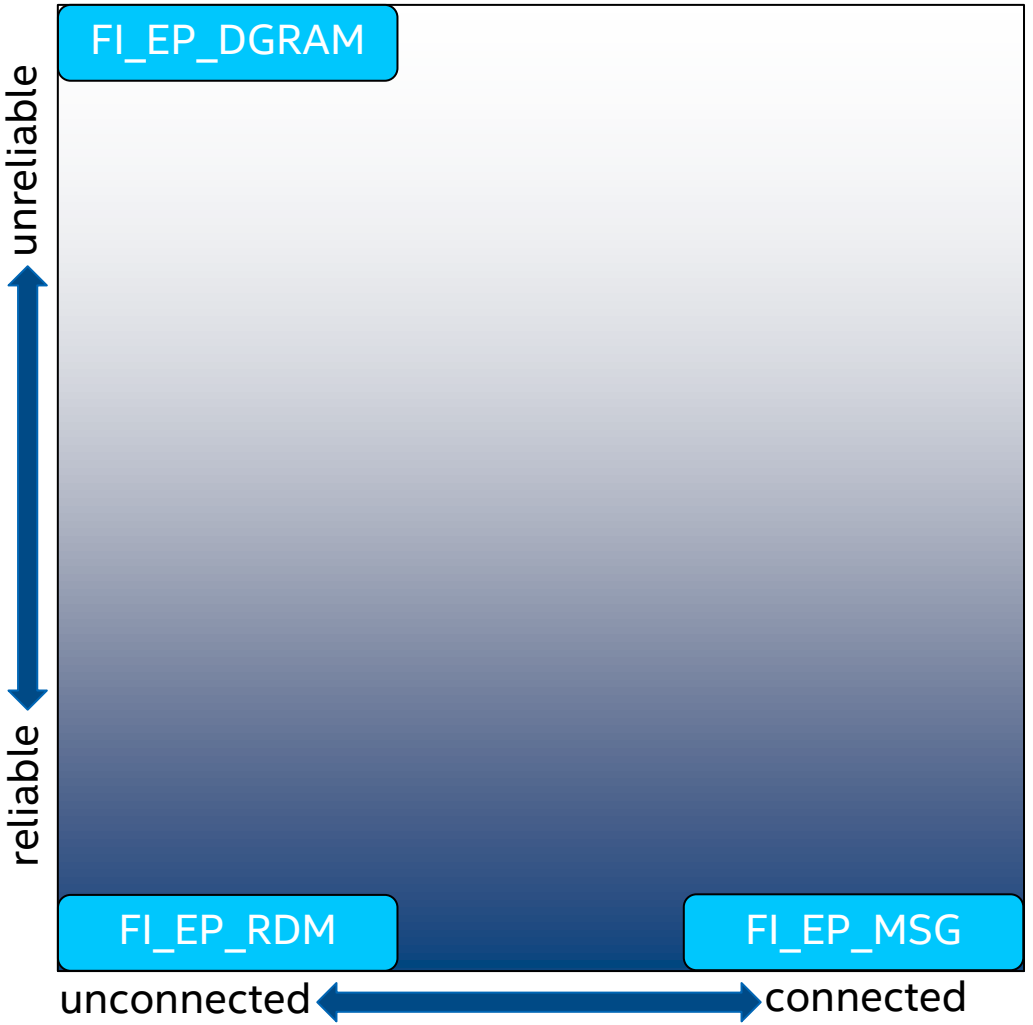
```
./autogen.sh
./configure --prefix=<install_dir> --with-libfabric=<libfabric_install_dir>
make -j 32 install
```

Libfabric


Passive Endpoint
Fabric
Event queue
Memory Region (MR)
Domain
Address Vector (AV)
Completion Counter
Active Endpoint
Completion Queue
Data Transfers

Libfabric – EP types

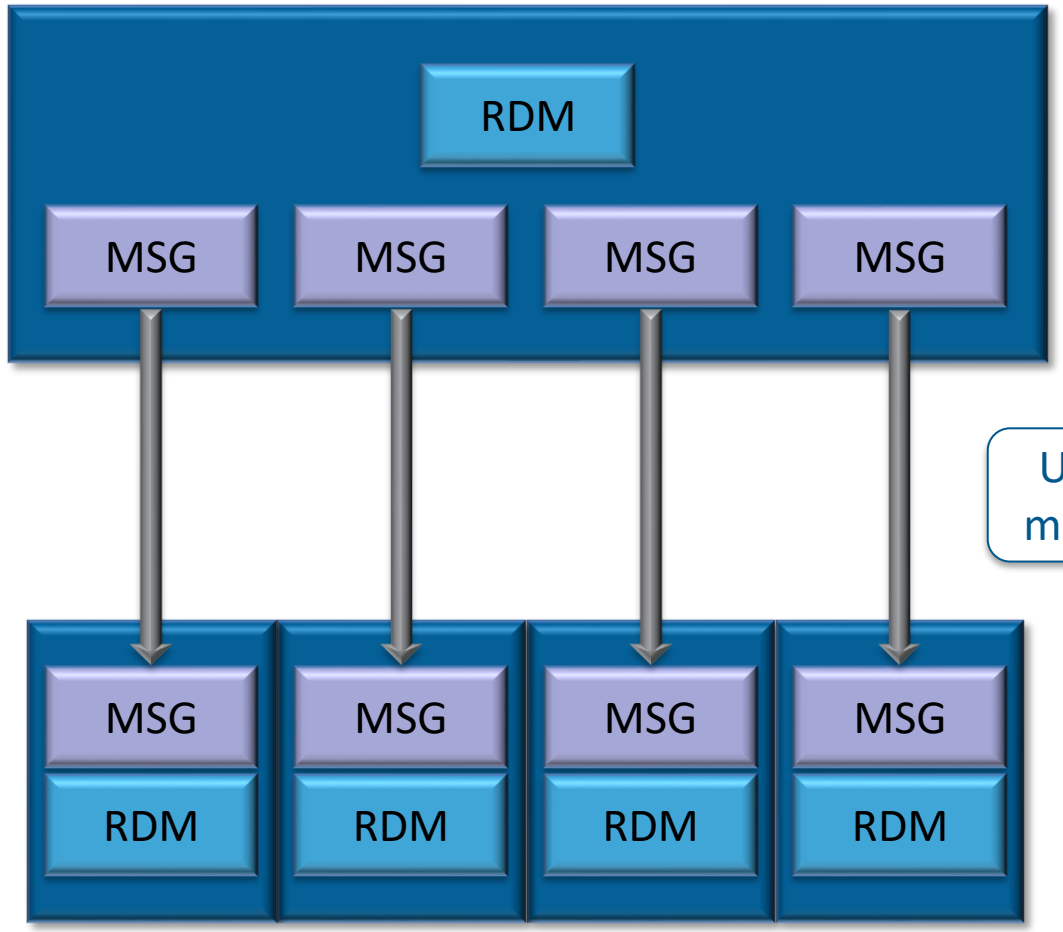
```
fi_av_insert(…)
    ->in: addr
    ->out: fiaddr
```




```
fi_listen(…)
fi_connect(…)
fi_accept(…)
fi_reject(…)
```

RXM Provider

RDM EP with full msg, tagged, RMA, and atomic functionality



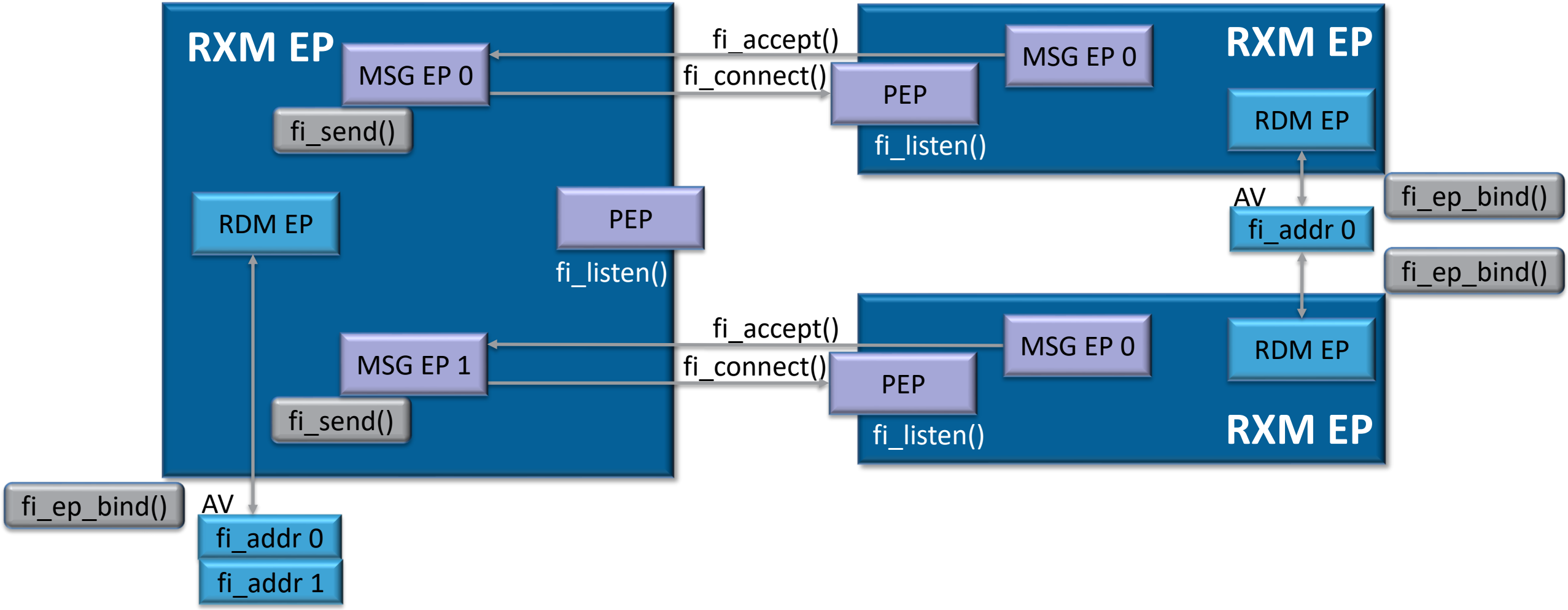
Targets verbs, tcp, and propriety hardware with reliable-connected semantics

Uses core provider messaging and RMA

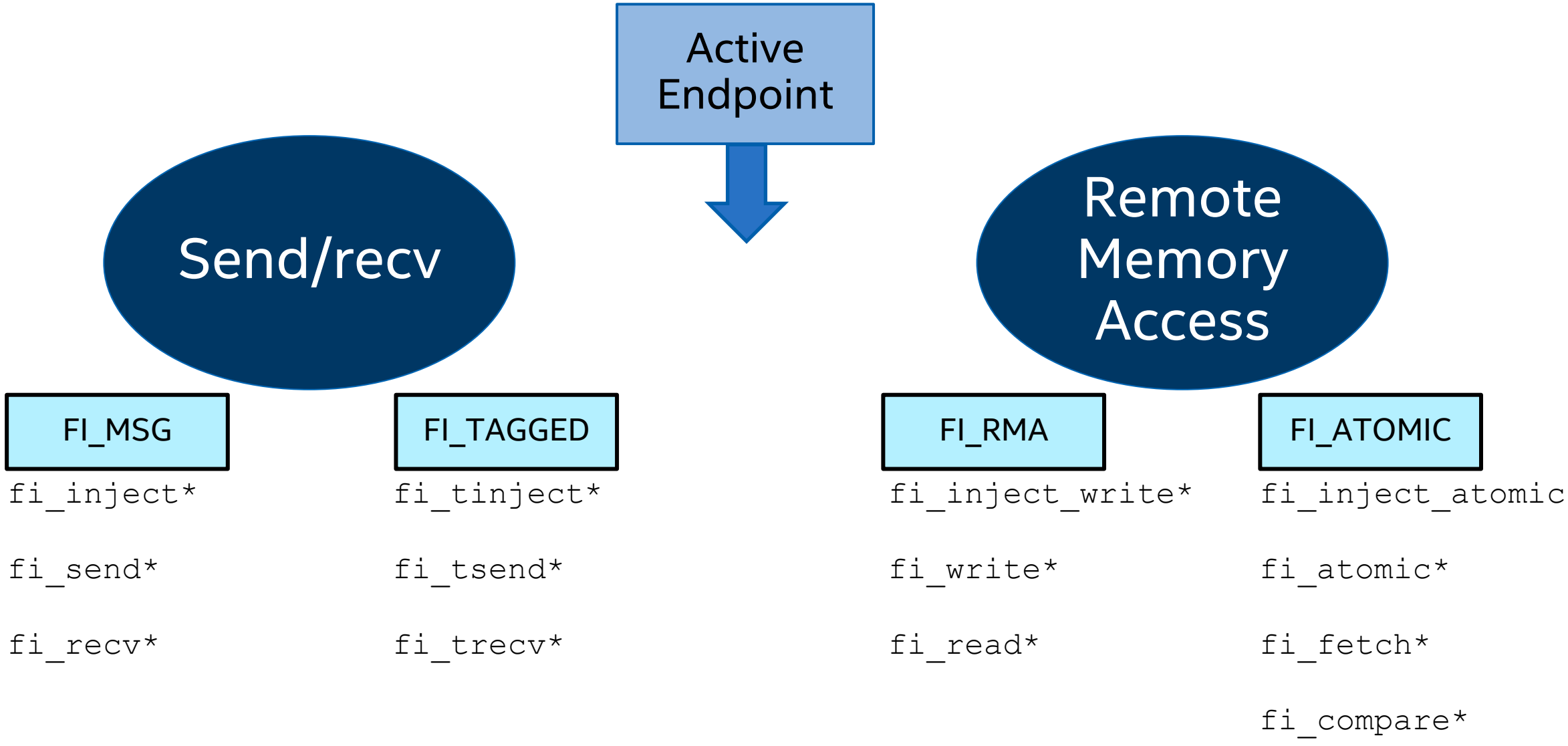
Handles connection management

RXM Provider

On-demand connections:



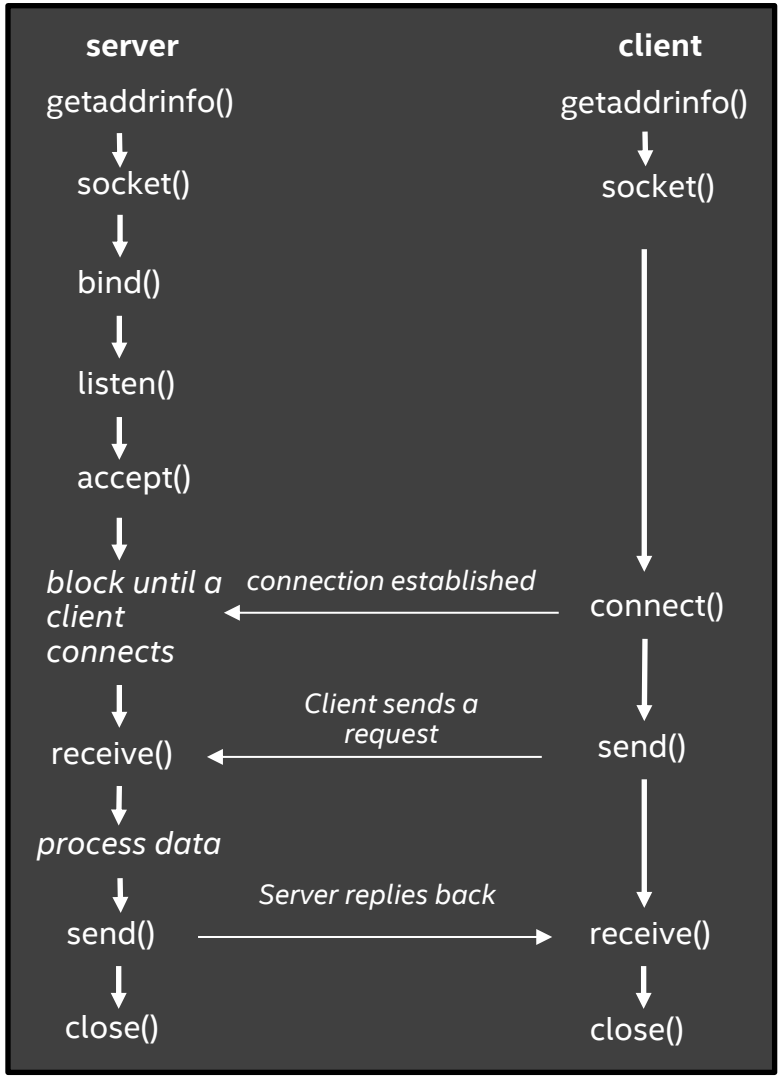
Libfabric – Data Transfers

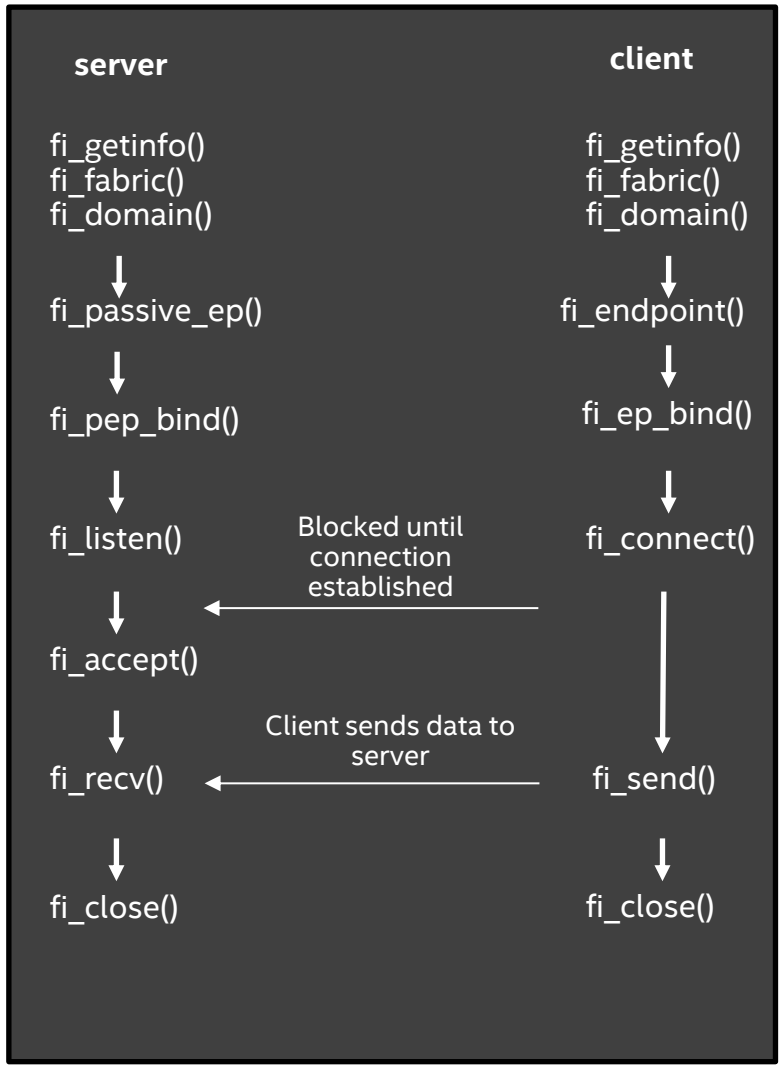


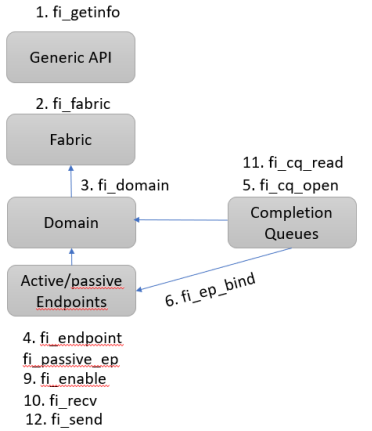
Recap

- Socket Vs libfabric
  - Msg and socket example
- Features and resources of libfabric
  - Simple program flows for msg, rdm_rma, and rdm_tagged
- Flow with an example of collective operation
- Profiling

Socket Vs Libfabric example







- 3 types of endpoints:

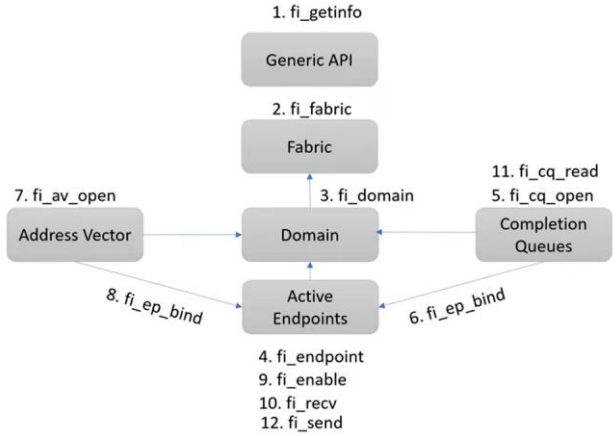
FI_EP_MSG -> connected and reliable
FI_EP_RDM
FI_EP_DGRAM

- Transfer types:

FI_MSG – send/recv
FI_TAGGED
FI_RMA
FI_ATOMIC
FI_COLLECTIVE

FI_RDM EP with send/recv

server	client
fi_getinfo(), fi_fabric(), fi_domain()	fi_getinfo(), fi_fabric(), fi_domain()
FI_EP_RDM	FI_EP_RDM
fi_endpoint()	fi_endpoint()
FI_CQ_FORMAT_MSG fi_cq_open() fi_ep_bind()	FI_CQ_FORMAT_MSG fi_cq_open() fi_ep_bind()
fi_av_open() fi_ep_bind() fi_av_insert()*	fi_av_open() fi_ep_bind() fi_av_insert()
fi_enable()	fi_enable()
fi_recv() fi_cq_read()	fi_send() fi_cq_read()
fi_close() ep, av, cq, domain, fabric	fi_close() ep, av, cq, domain, fabric



- **3 types of endpoints:**

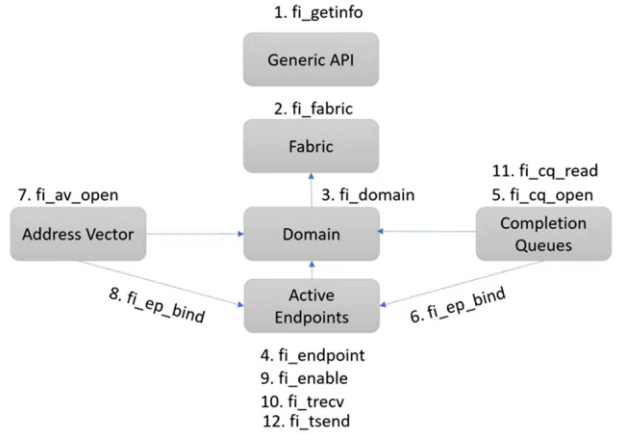
FI_EP_MSG
FI_EP_RDM – Unconnected & Reliable
FI_EP_DGRAM

- **Transfer types:**

FI_MSG – send/recv
FI_TAGGED
FI_RMA
FI_ATOMIC
FI_COLLECTIVE

FI_RDM EP with Tagged send/recv

server	client
fi_getinfo(), fi_fabric(), fi_domain()	fi_getinfo(), fi_fabric(), fi_domain()
FI_EP_RDM fi_endpoint()	**FI_EP_RDM** fi_endpoint()
FI_CQ_FORMAT_TAGGED fi_cq_open() fi_ep_bind()	**FI_CQ_FORMAT_TAGGED** fi_cq_open() fi_ep_bind()
fi_av_open() fi_ep_bind() fi_av_insert()*	fi_av_open() fi_ep_bind() fi_av_insert()*
fi_enable()	fi_enable()
fi_trecv() fi_cq_read()	fi_tsend() fi_cq_read()
fi_close() ep, av, cq, domain, fabric	fi_close() ep, av, cq, domain, fabric



- **3 types of endpoints:**

FI_EP_MSG
FI_EP_RDM – Unconnected & Reliable
FI_EP_DGRAM

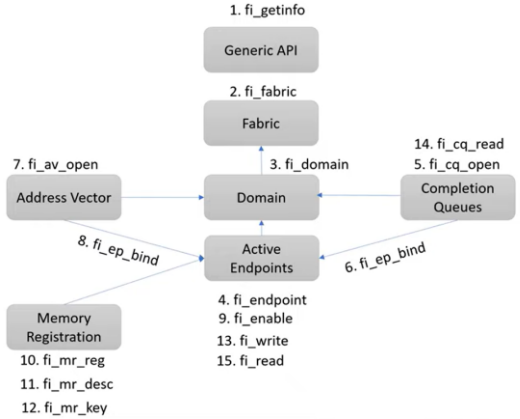
- **Transfer types:**

FI_MSG
FI_TAGGED – send/recv
FI_RMA
FI_ATOMIC
FI_COLLECTIVE

FI_RDM EP with RMA

server	client
fi_getinfo(), fi_fabric(), fi_domain()	fi_getinfo(), fi_fabric(), fi_domain()
FI_EP_RDM fi_endpoint()	**FI_EP_RDM** fi_endpoint()
FI_CQ_FORMAT_MSG fi_cq_open() fi_ep_bind()	**FI_CQ_FORMAT_MSG** fi_cq_open() fi_ep_bind()
fi_av_open() fi_ep_bind() fi_av_insert()	fi_av_open() fi_ep_bind() fi_av_insert()
fi_enable()	fi_enable()
fi_mr_reg() fi_mr_desc() fi_mr_key()	fi_mr_reg() fi_mr_desc() fi_mr_key()
fi_read() fi_cq_read()	**fi_write()** fi_cq_read()
fi_close() ep, av, cq, domain, fabric	fi_close() ep, av, cq, domain, fabric

*Key exchange happen out of band in this example
* Data transfer calls here are not paired



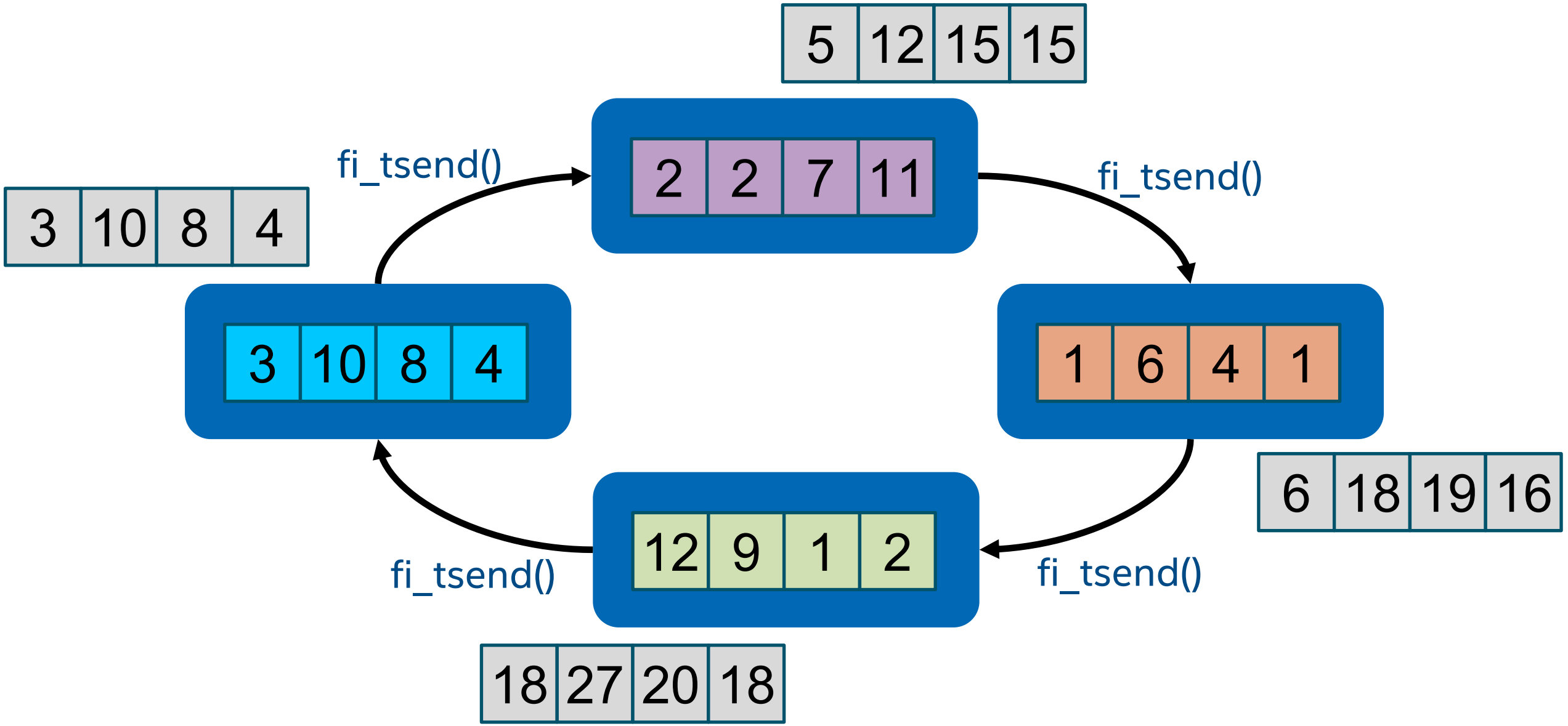
- **3 types of endpoints:**

FI_EP_MSG
FI_EP_RDM – Unconnected & Reliable
FI_EP_DGRAM

- **Transfer types:**

FI_MSG
FI_TAGGED
FI_RMA – Write/read
FI_ATOMIC
FI_COLLECTIVE

Flow with Collective: reduce (sum)



Collectives

Collective example API flow

fi_getinfo(),
fi_fabric(),
fi_domain()

FI_EP_RDM
fi_endpoint()

fi_cq_open()
fi_ep_bind()

fi_av_open()
fi_ep_bind()

fi_enable()

```
fi_query_collective()
fi_av_set()
fi_join_collective
fi_eq_sread()
fi_allreduce()
```

fi_close()
ep, av, cq,
domain, fabric

➢**3 types of endpoints:**
FI_EP_MSG
FI_EP_RDM endpoint
FI_EP_DGRAM endpoint

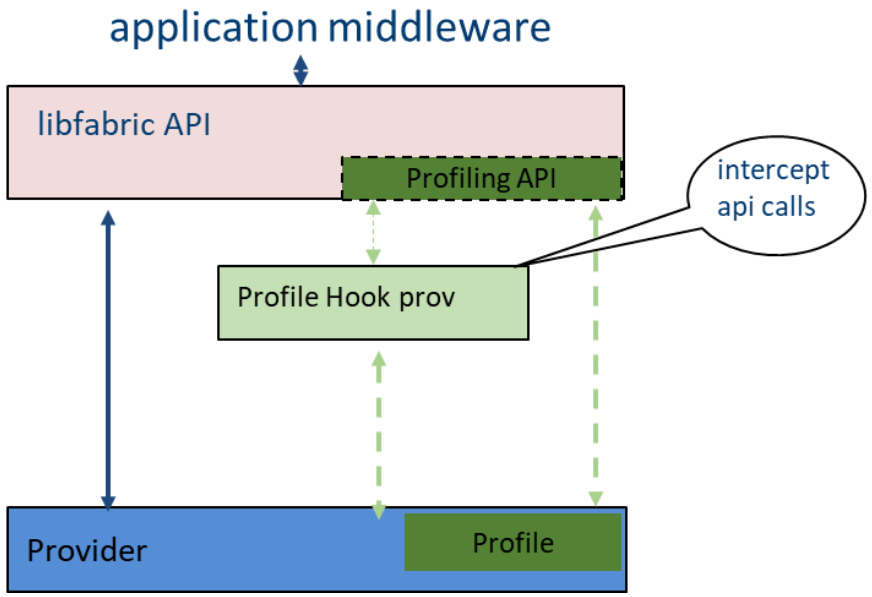
➢**Transfer types:**
FI_MSG – send/recv
FI_TAGGED – send/recv
FI_RMA – Write/read
FI_ATOMIC
FI_COLLECTIVE

Profiling

- Ability to debug and do performance analysis
- Gain insight into providers implementation
- Analyze how libfabric is being used by applications/middlewares

Profiling is done via

- Profiling hooking provider
  - Hooking API calls
  - Aggregate API calls
- Fabric Profiling Interface (FPI)
  - Variables
  - Events
  - APIs



- fi_profile_query_vars()
- fi_profile_query_events()
- fi_profile_read_u64()
- fi_profile_register_callback()
- fi_profile_start_reads()
- fi_profile_end_reads()

Agenda

01 Overview

02 Recap

03 Peer API / LNX

04 UEC and Libfabric

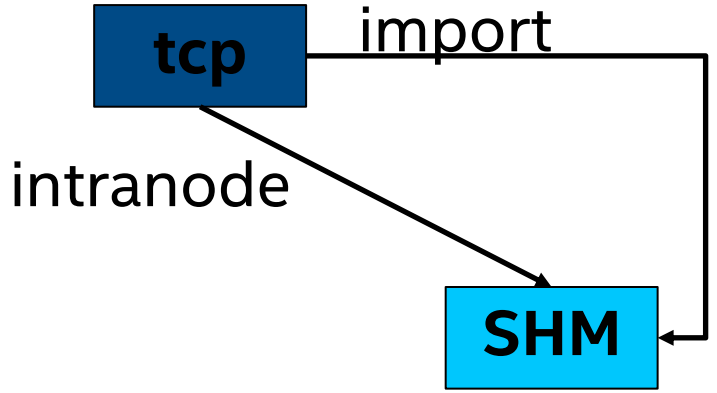
05 Conclusion

Peer API/LNX

- Use shared memory provider (shm) to optimize intranode communication
- Expose one endpoint to app while using two providers
  - One for external, internode communication
  - One for internal, intranode communication
- Share provider resources
  - Write to same CQ/counter
  - Get receive buffers from the same receive context (SRX)
  - Share addressing (e.g. fi_addr)

Peer API/LNX

tcp offloads to shm for local communication

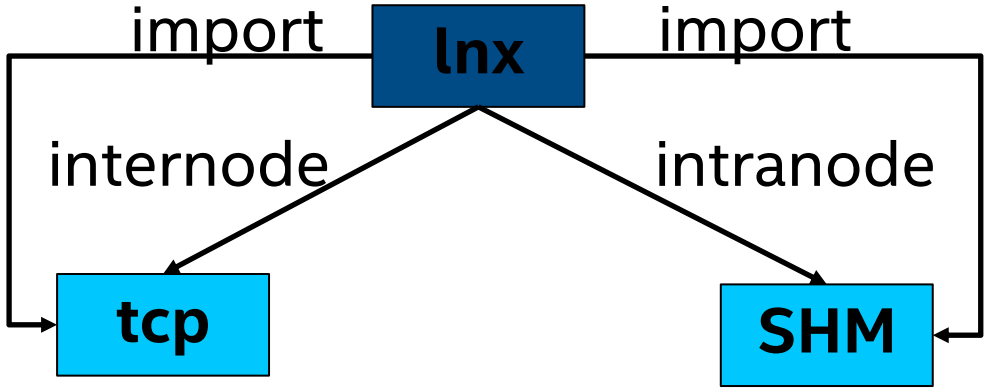


tcp redirects intranode transfers to shm

tcp owns CQ and SRX

shm directly accesses tcp CQ and SRX

link provider handles coordination, providers only to import

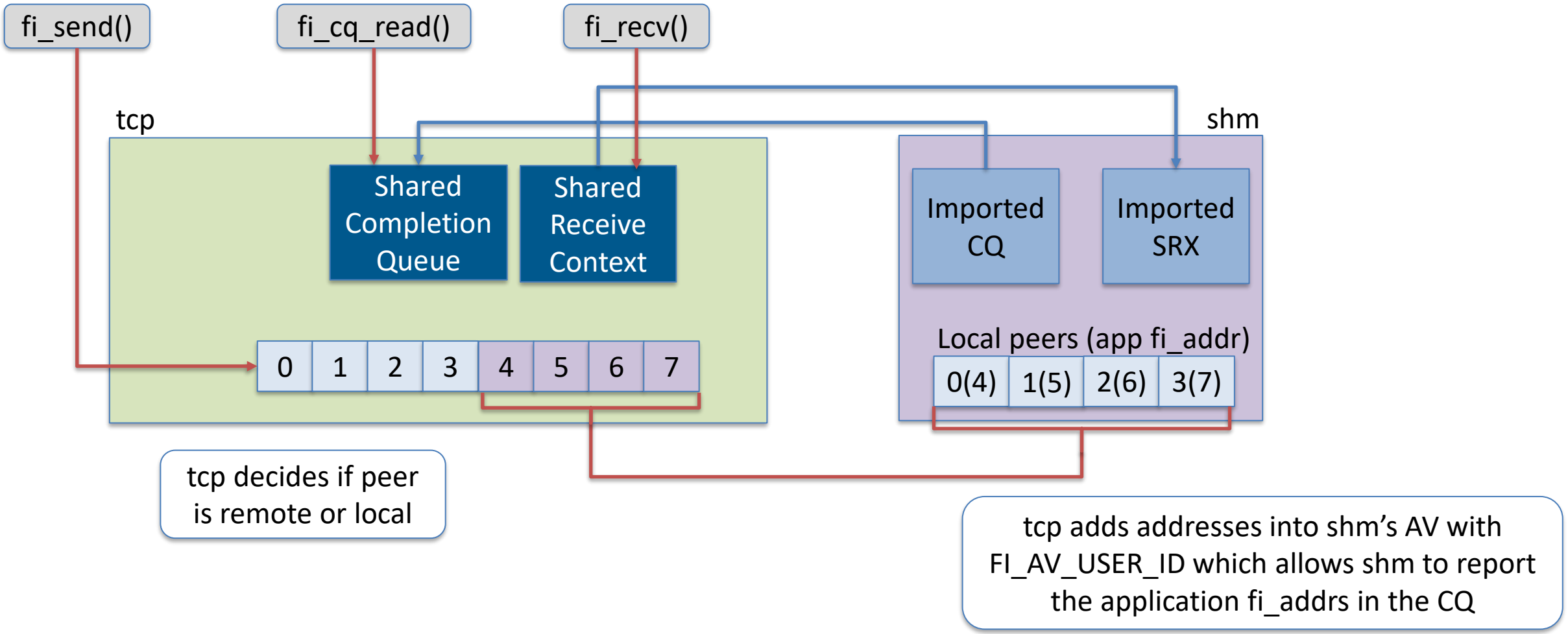


lnx picks provider based on target address

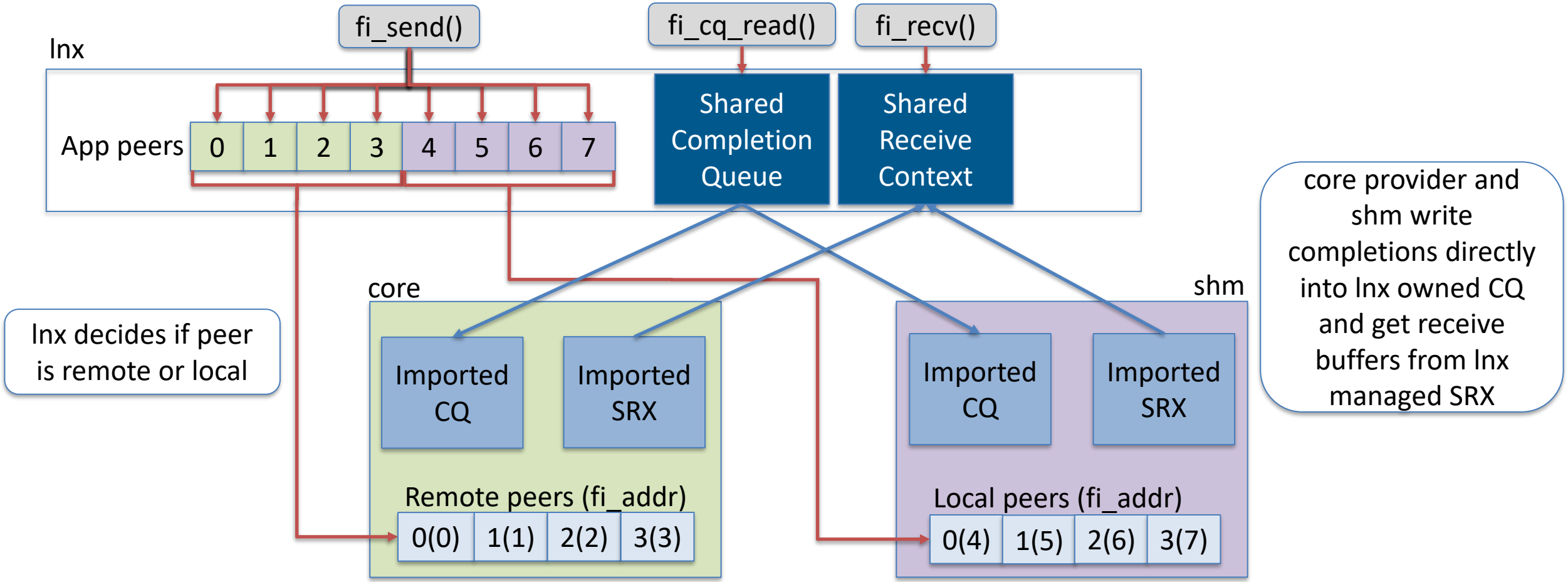
Generic lnx owns CQ and SRX

shm and tcp directly access lnx queues

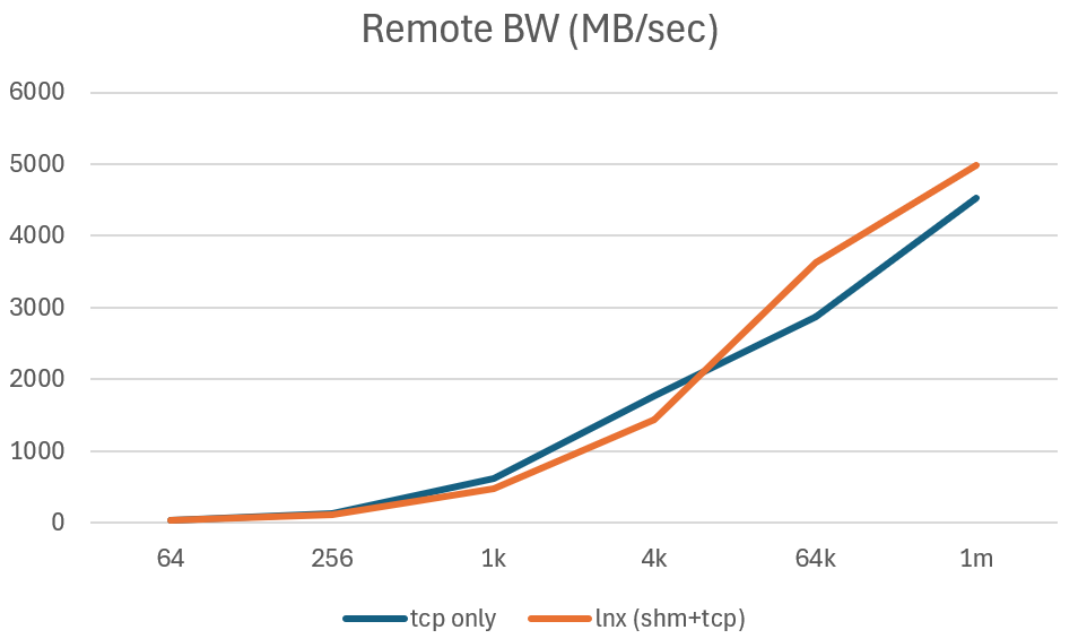
Peer API/LNX – TCP Owner

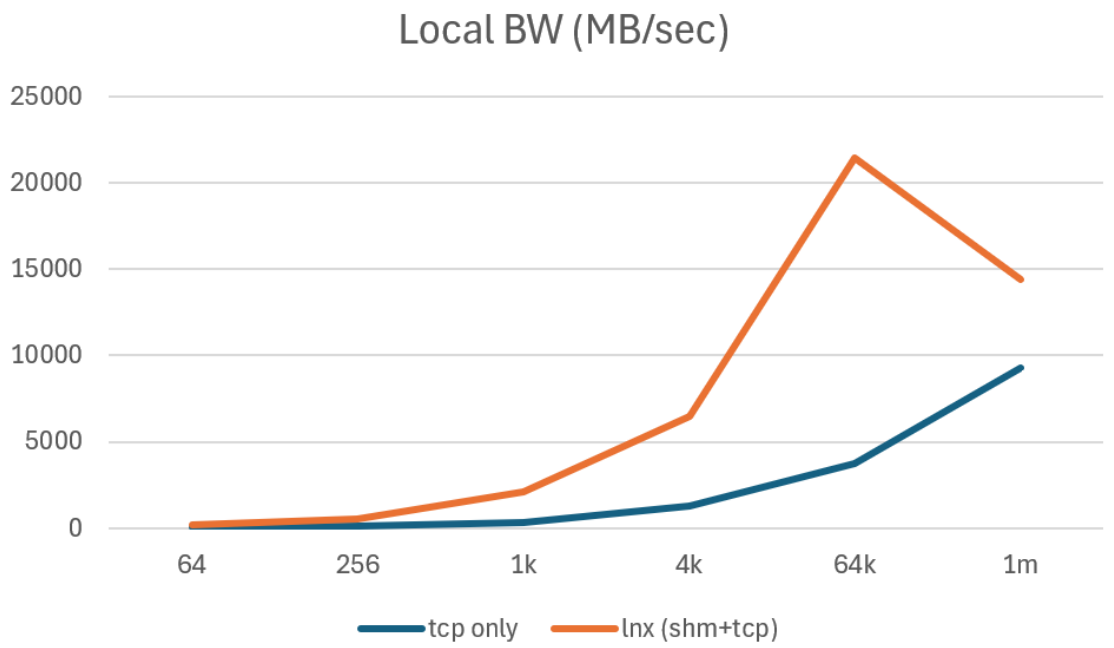


Peer API/LNX – LNX Owner



Peer API/LNX


Remote BW (MB/sec)
6000
5000
4000
3000
2000
1000
0
64
256
1k
4k
64k
1m
tcp only
lnx (shm+tcp)


Local BW (MB/sec)
25000
20000
15000
10000
5000
0
64
256
1k
4k
64k
1m
tcp only
lnx (shm+tcp)

Peer API/LNX

```
export FI_TCP_IFACE=eth0
export FI_PROVIDER=tcp
fi_rdm_tagged_bw -E
```

```
export FI_LNX_PROV_LINKS="shm+tcp:eth0“
export FI_PROVIDER=lnx
fi_rdm_tagged_bw -E
```

Agenda

01 Overview

02 Recap

03 Peer API / LNX

04 UEC and Libfabric

05 Conclusion

UEC and Libfabric

Modern Transport and RDMA Services Needs for AI and HPC

Requirement	UEC Transport	Legacy RDMA	UEC Advantage
Multi-Pathing	Packet spraying	Flow-level multi-pathing	Higher network utilization
Flexible Ordering	Out-of-order packet delivery with in-order message delivery	N/A	Matches application requirements, lower tail latency
AI and HPC Congestion Control	Workload-optimized, configuration free, lower latency, programmable	DCQCN: configuration required, brittle, signaling requires additional round trip	Incast reduction, faster response, future-proofing
E2E Telemetry	Sender or Receiver	ECN	Faster congestion resolution, better visibility
Simplified RDMA	Streamlined API, native workload interaction, minimal endpoint state	Based on IBTA Verbs	App-level performance, lower cost implementation
Security	Scalable, 1st class citizen	Not addressed, external to spec	High scale, modern security
Large Scale with Stability and Reliability	Targeting 1M endpoints	Typically, a few thousand simultaneous end points	Current and future-proof scale

Ultra Ethernet

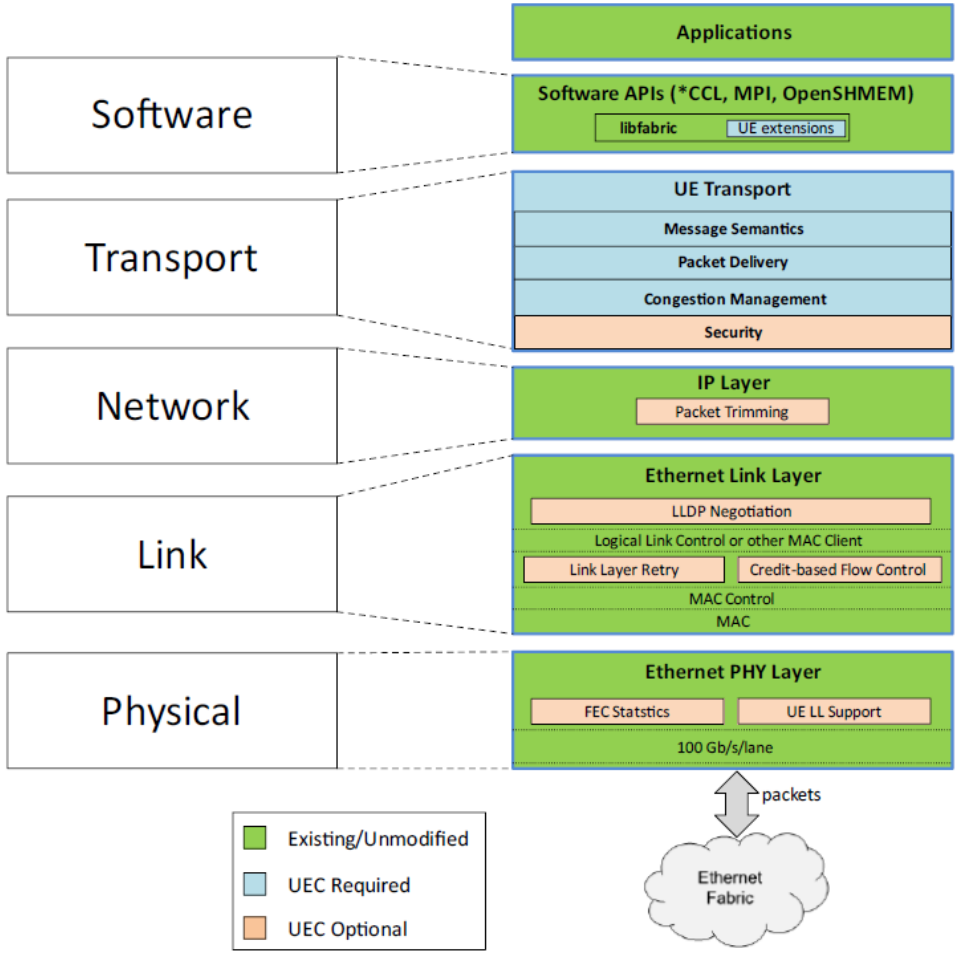
UEC and Libfabric

UET is focused on enabling the workloads and use cases for HPC and AI (AIT and AII)

RDMA service and attempts to provide the best, modernized, and highly optimized transport service for carrying RDMA in AI and HPC workloads

Meet the demand for hyperscale computing and AI with 1M simultaneous endpoints

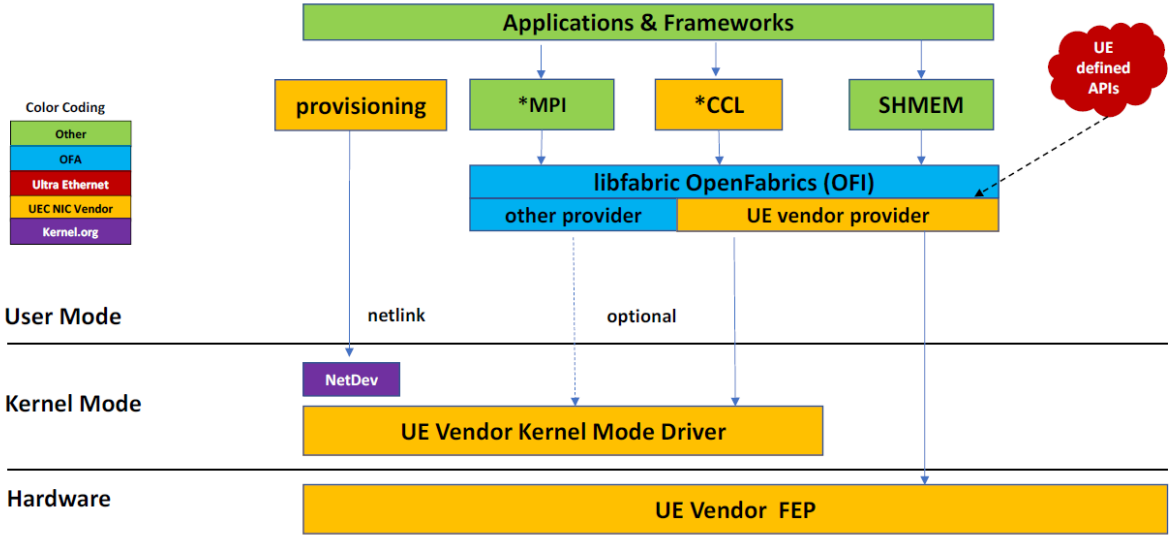
- Multi-Pathing and Packet Spraying
- Flexible ordering
- Optimized Congestion Control
- End to End Telemetry



UEC and Libfabric

Libfabric Mapping spec:

Goal is to extend and change libfabric as required to provide a vendor-interoperable mapping that supports all the Ultra Ethernet profiles defined by the transport working group



- Libfabric addressing
  - FI_ADDR_UET
- Discovery APIs
  - Ensure libfabric version is >= 2.0
  - Predefined service names (Generic, CCL, mpi, shmem)
- Communication and Completion APIs
- Data transfer APIs

- PDS
- Traffic Classes

UEC and Libfabric

Communication and Completion

UET providers must support FI_EP_DGRAM & FI_EP_RDM endpoints

APIs that must be supported by UET

Data transfer types	Endpoint	APIs
FI_MSG	FI_EP_DGRAM FI_EP_RDM	fi_recv, fi_recvv, fi_revcmsg fi_send, fi_sendv, fi_sendmsg, fi_inject, fi_senddata, fi_injectdata
FI_TAGGED	FI_EP_RDM	fi_trecv, fi_trecvv, fi_trecvmsg fi_tsend, fi_tsendv, fi_tsendmsg, fi_tinject, fi_tsenddata
FI_RMA	FI_EP_RDM	Fi_read, fi_readv, fi_readmsg fi_write, fi_writev, fi_writemsg, fi_inject_write, fi_writedata
FI_ATOMIC	FI_EP_RDM	fi_atomic, fi_atomicv, fi_atomicmsg, fi_inject_atomic, fi_fetch_atomic, fi_fetch_atomicv, fi_fetch_atomicmsg, fi_compare_atomic, fi_compare_atomicv, fi_compare_atomicmsg, fi_atomicvalid, fi_fetch_atomicvalid, fi_compare_atomicvalid, fi_query_atomic

Completion counter requirements

Profile	Completion Counter Requirements
AI Base	None
AI Full	FI_SEND, FI_RECV, FI_READ, FI_WRITE
HPC	FI_SEND, FI_RECV, FI_READ, FI_WRITE, FI_REMOTE_READ, FI_REMOTE_WRITE

UET and Libfabric

Packet delivery modes

Endpoint type	Delivery mode
FI_EP_DGRAM	UUD (Unreliable Unordered Delivery)
FI_EP_RDM	ROD (Reliable Ordered Delivery) RUD (Reliable Unordered Delivery) RUDI (Reliable Unordered Delivery of Idempotent Operation) (only for HPC profile)

Traffic class: The SES sublayer is aware of only data traffic classes. The libfabric application is not aware of traffic classes used by PDS. Default = Default Forwarding (DF) PHB DSCP Codepoint = '000000 '

Job ID: The JobID is part of UET addressing, is carried in the SES header and is used for authorization

- JobID assignment at job initialization time
- JobID assignment at libfabric endpoint creation time
- Fallback JobID assignment

Use FI_AV_AUTH_KEY capability to select the JobID associated with a data transfer operation When multiple JobIDs are assigned to the same {OS PID, Service Name}

Agenda

01 Overview

02 Recap

03 Peer API / LNX

04 UEC and Libfabric

05 Conclusion

Takeaways

- Libfabric is a networking abstraction to allow for more portability of programs between hardware
- We saw different types of endpoint and data transfers
- Features provided via utility providers like profiling hook, RxM and link provider
- Libfabric mapping to UET provider

Useful links

Libfabric

Libfabric man pages

Webinar #2: Coding for Libfabric – OpenFabrics Alliance (August 27, 2025 | 8:00AM-9:00AM PT)

Go to openfabrics.org to learn more and register

UEC: https://ultraethernet.org/

1.0 Spec https://ultraethernet.org/uec-1-0-spec

Thank you!

Key take-aways

01 Why do we need Libfabric

02 What is Libfabric

03 Basics

04 RDM example

05 Peer API / LNX

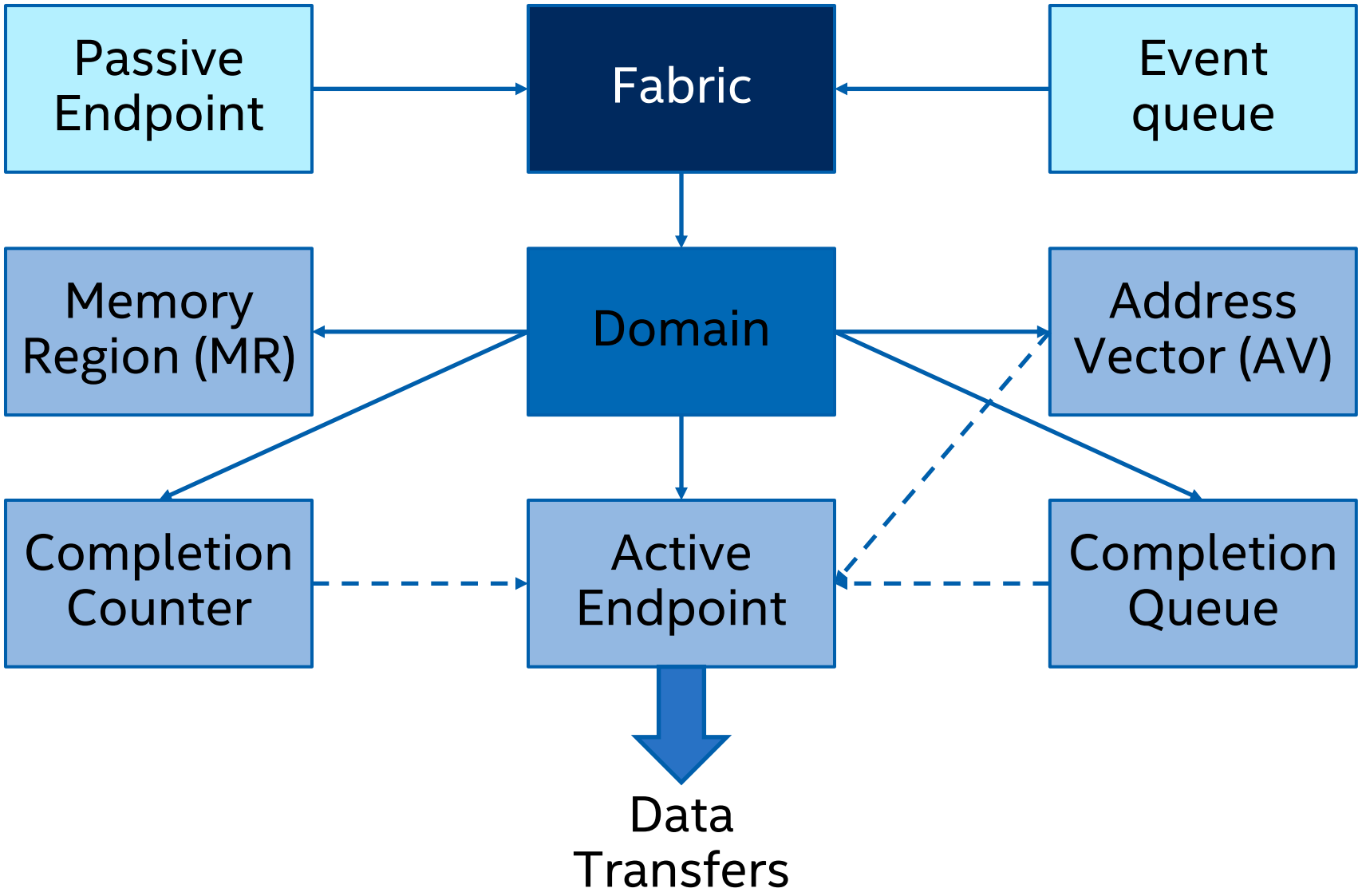
06 UEC and Libfabric

07 Conclusion and useful resources

Features and Resources

- 3 types of endpoints:
  - FI_EP_MSG -> connected and reliable (msg example)
  - FI_EP_RDM endpoint -> unconnected and reliable (rdm example)
  - FI_EP_DGRAM endpoint -> unconnected and unreliable
- Transfer types:
  - FI_MSG – send/recv
  - FI_TAGGED –send/recv with tag matching
  - FI_RMA – remote memory access (read/write)
  - FI_ATOMIC – RMA with atomic operation on data
  - FI_COLLECTIVE – simple, OFI-direct collective algorithms

RDM example


Passive Endpoint
Fabric
Event queue
Memory Region (MR)
Domain
Address Vector (AV)
Completion Counter
Active Endpoint
Completion Queue
Data Transfers

Libfabric Collectives

- Libfabric Software Abstraction APIs for Collectives
  - Collective membership (fi_av_set)
  - Setting up of communication groups (join_collective(), broadcast_send() broadcast_recvfrom())
    - OFI managed
    - Application Managed
  - Data transfers by invoking collectives (fi_allreduce, …)