An Initial Implementation of Libfabric Conduit for OpenSHMEM-X

Subhadeep Bhattacharya*
Shaeke Salman*
Manjunath Gorentla Venkata†
Harsh Kundnani*
Neena Imam†
Weikuan Yu*

*Florida State University
†Oak Ridge National Laboratory

Outline

- Background and Motivation
- Design of Libfabric Conduit
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion and Future Works

OpenSHMEM and it's implementations

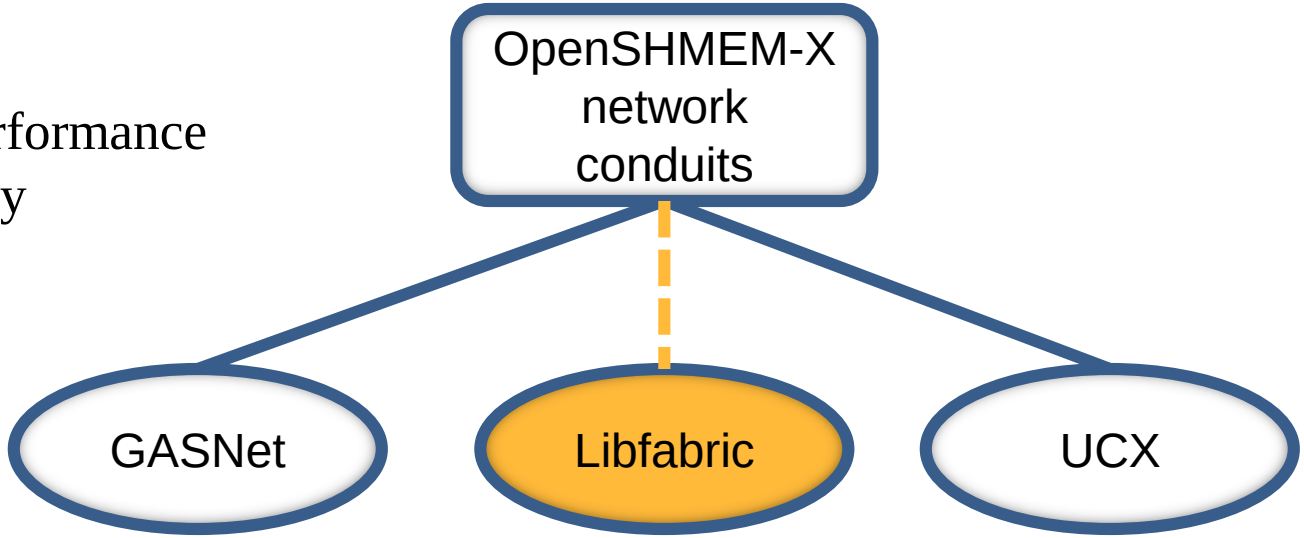
- OpenSHMEM
  - standardize collection of programming libraries
  - provide parallel processing capabilities.
  - representative of Partitioned Global Address Space Model

- Some of the basic functionalities include:
  - Point-to-Point operations
  - Atomic operations
  - Collective Routines

- OpenSHMEM-X is implemented by Oak Ridge National Laboratory and it follows the latest OpenSHMEM standard.

Role of communication layer

- OpenSHMEM takes advantage of one sided communication using high end interconnects.
- Enable the communication functionalities between different processing elements.
- Objective:
  - Portability
  - Improved performance and scalability



OpenSHMEM-X + Libfabric

- Libfabric:
  - set of network libraries to work with different providers.
  - implemented by OpenFabrics Interface (OFI) working group.
  - Optimized for various providers

- Goals:
  - high-bandwidth
  - low-latency
  - high scalability
  - portable network implementation

Outline

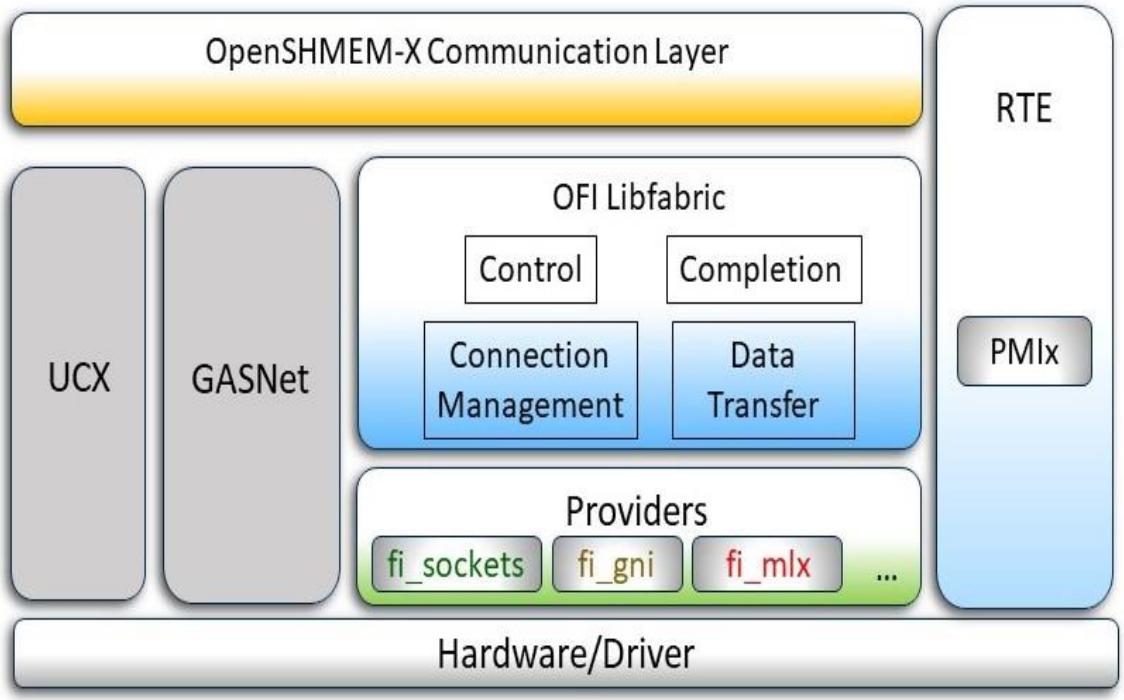
- Background and Motivation
- Design of Libfabric Conduit
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion and Future Works

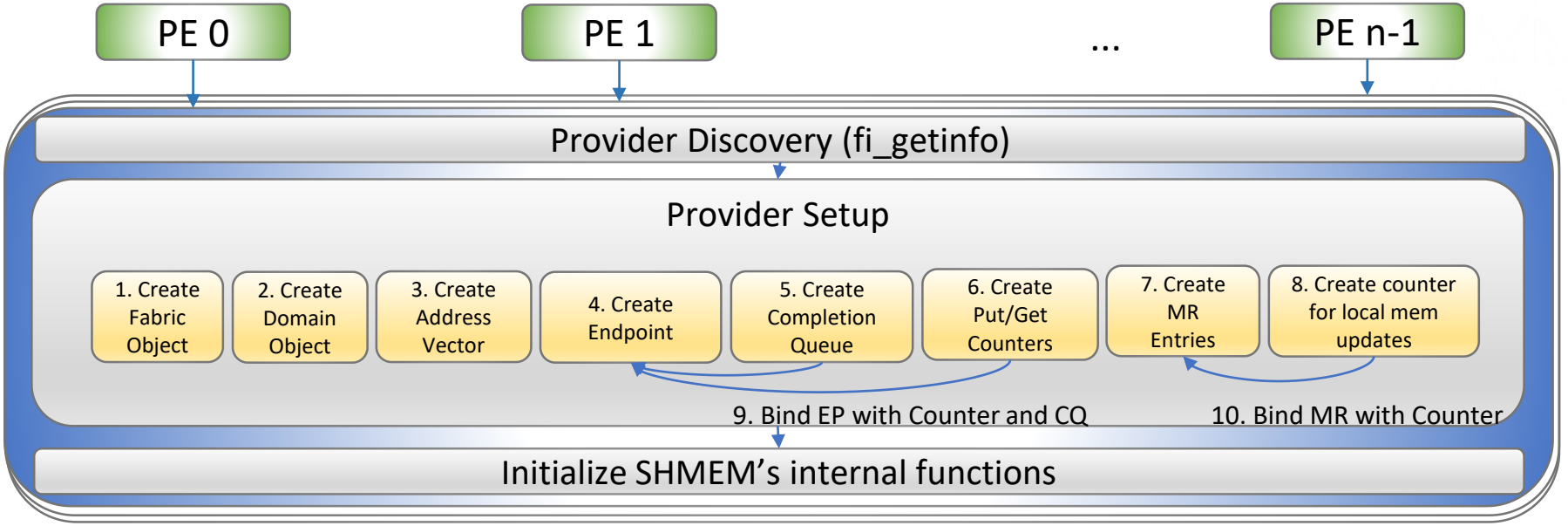
Design Overview

- Existing communication conduits:
  - GASNet
  - UCX
- Our implementation introduces
  - **OFI Libfabric**

Provider: **Sockets**

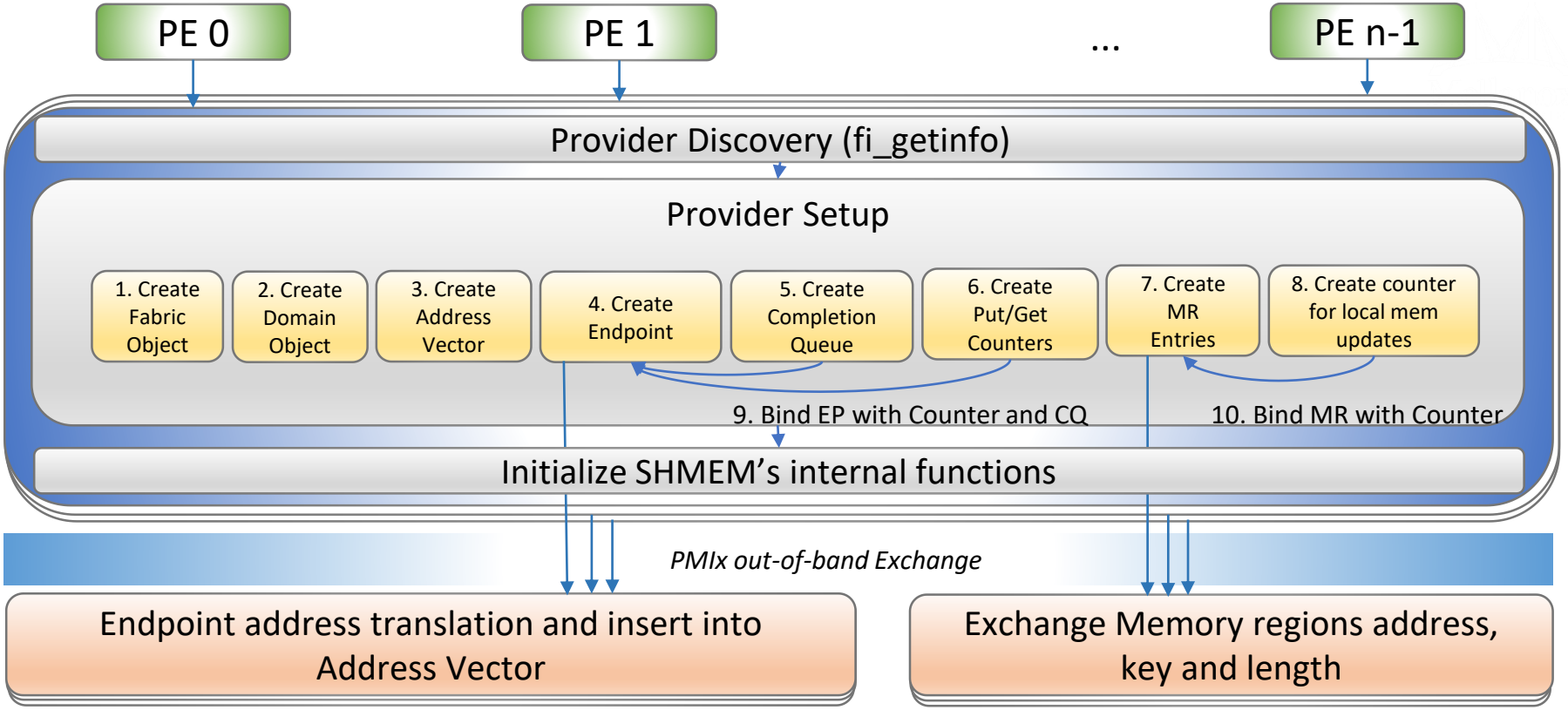
Out-of-band channel: **P**rocess **M**anagement **I**nterface E**x**ascale




PE 0
PE 1
...
PE n-1
Provider Discovery (fi_getinfo)
Provider Setup
1. Create Fabric Object
2. Create Domain Object
3. Create Address Vector
4. Create Endpoint
5. Create Completion Queue
6. Create Put/Get Counters
7. Create MR Entries
8. Create counter for local mem updates
9. Bind EP with Counter and CQ
10. Bind MR with Counter
Initialize SHMEM's internal functions

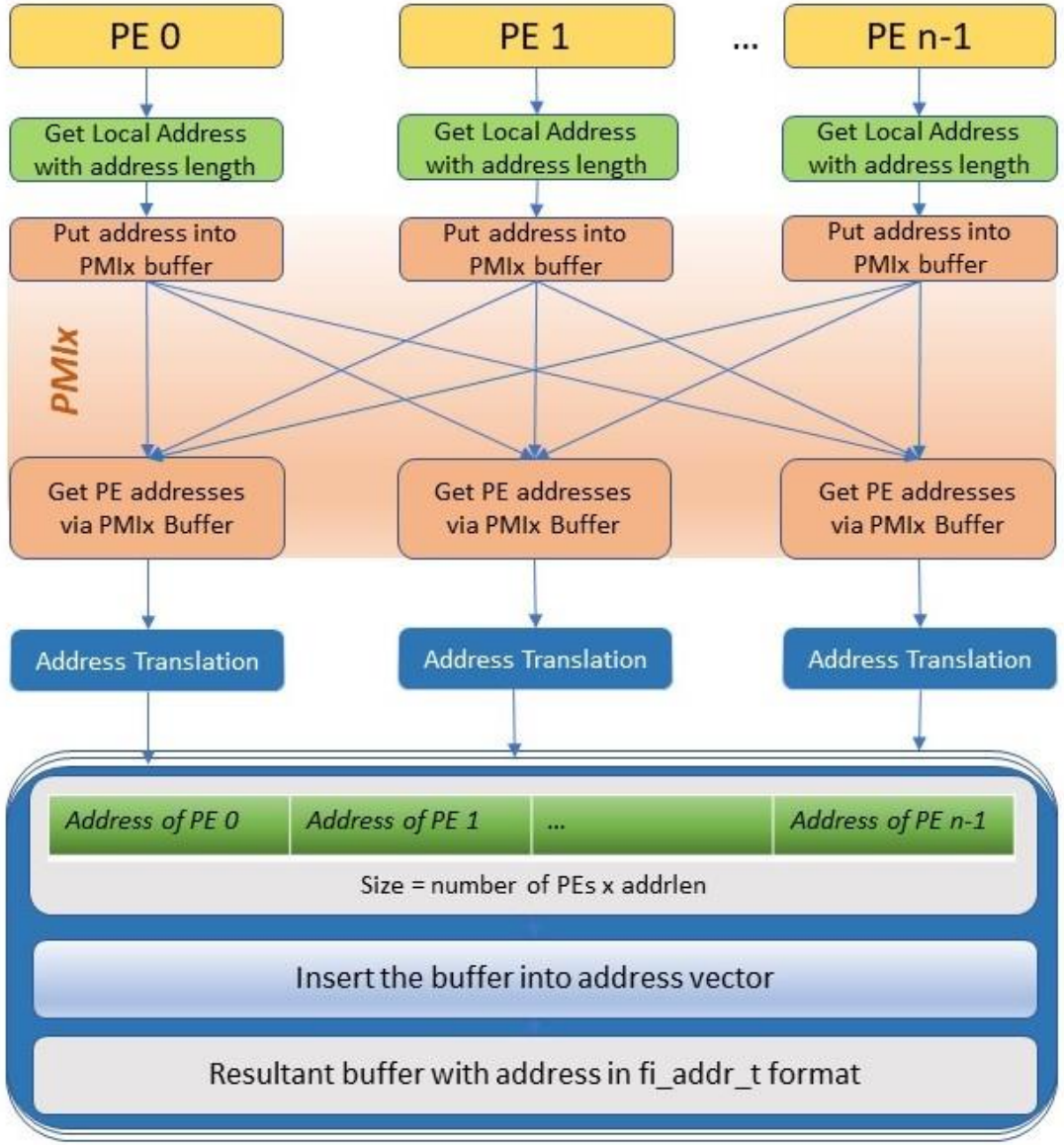
Initialization of Libfabric conduit

- Provider discovery: fi_getinfo
- Create fabric and domain object
- Endpoint creation
- Address vector initialization
- Create MR(Memory Region) entries for:
  - Data Segment
  - BSS Segment
  - Heap Segment
- Bind completion queue and put/get counters to the endpoint for completion event
- Bind a counter to the MR to keep track of local memory updates



Steps Involved in OpenSHMEM-X Libfabric Conduit

Address Translation

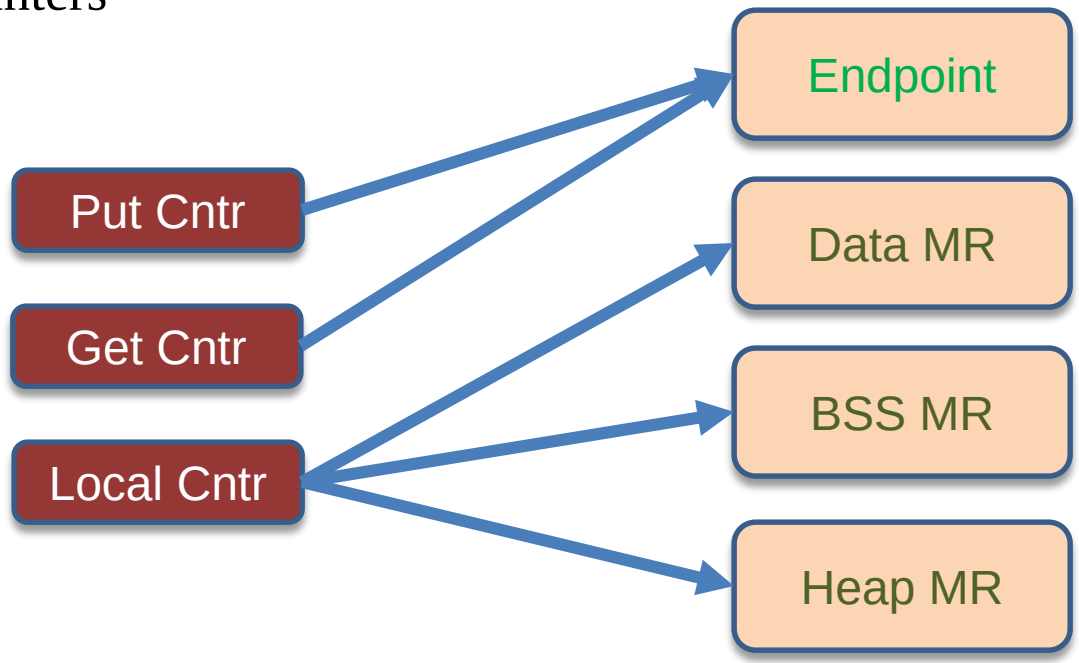


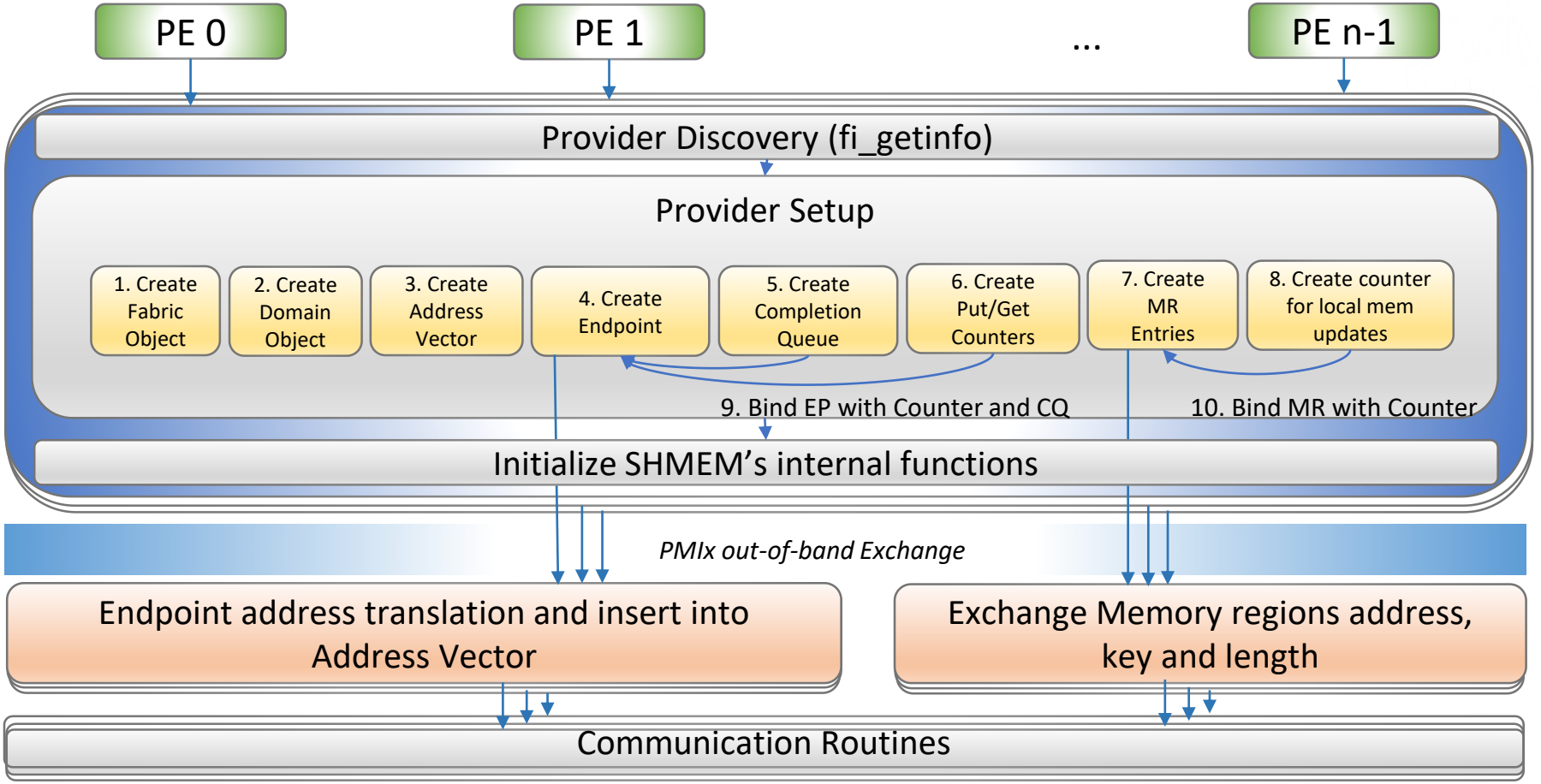
Memory Management

- Register memory regions using the base address and length of **data**, **BSS** and **heap** section.
  - Scheme used: FI_MR_BASIC
- Exchange memory segment information among other PEs using PMIx as out-of-band channel.
  - fi_rma_iov is used as a container for the base address, length and key.
- Maintain a buffer with memory segment information of all PEs in each PE.

Completion Queues and Counters

- All Libfabric communication operations are non-blocking in nature
  - Polling is required for blocking operations of OpenSHMEM.
- Two mechanisms
  - Completion Queue
  - Counters





Steps Involved in OpenSHMEM-X Libfabric Conduit

Remote Memory Access routines

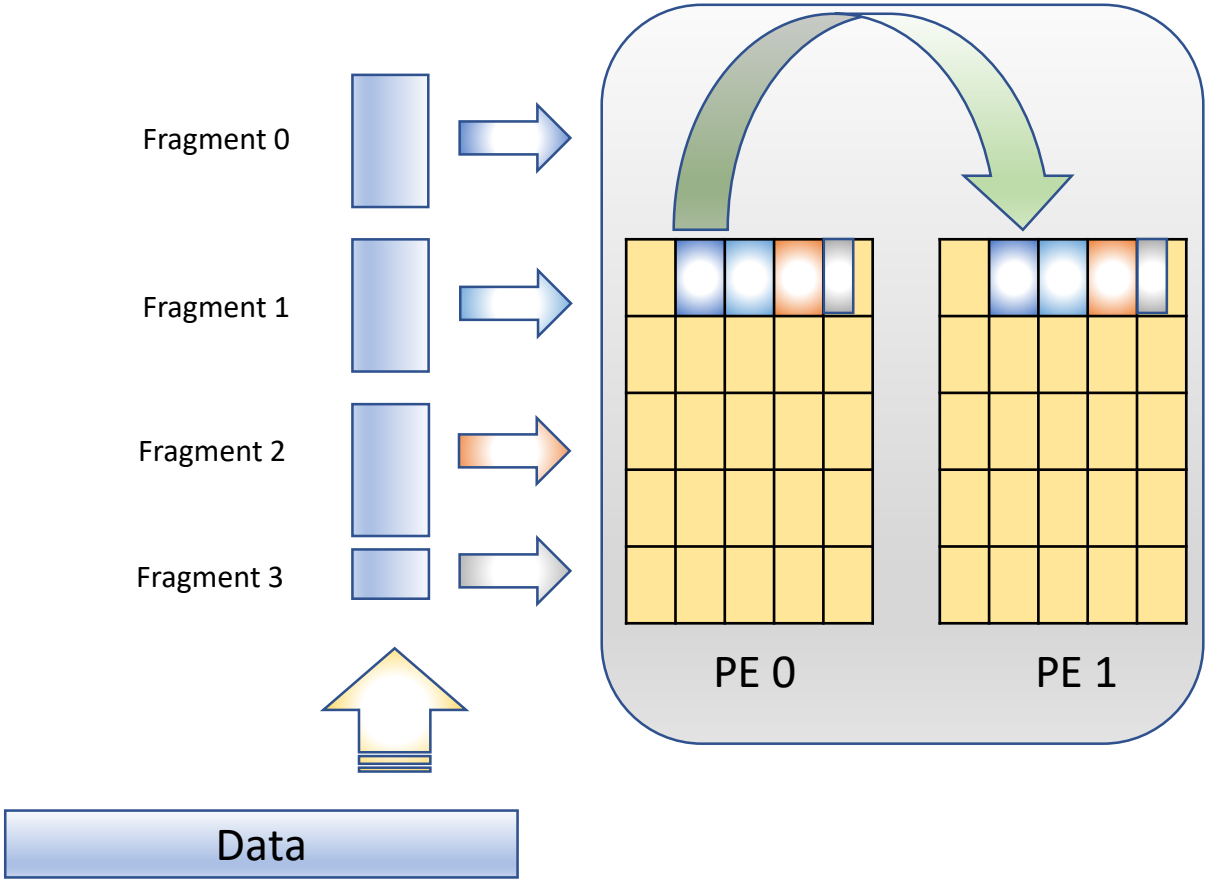
Communication Routines

Get details of registered memory region

- Find correct segment
- Check address range
- Adjust offset for remote address

Break the data into fragments based on max buffer size of the endpoint

Communication operations (put/get/atomic)

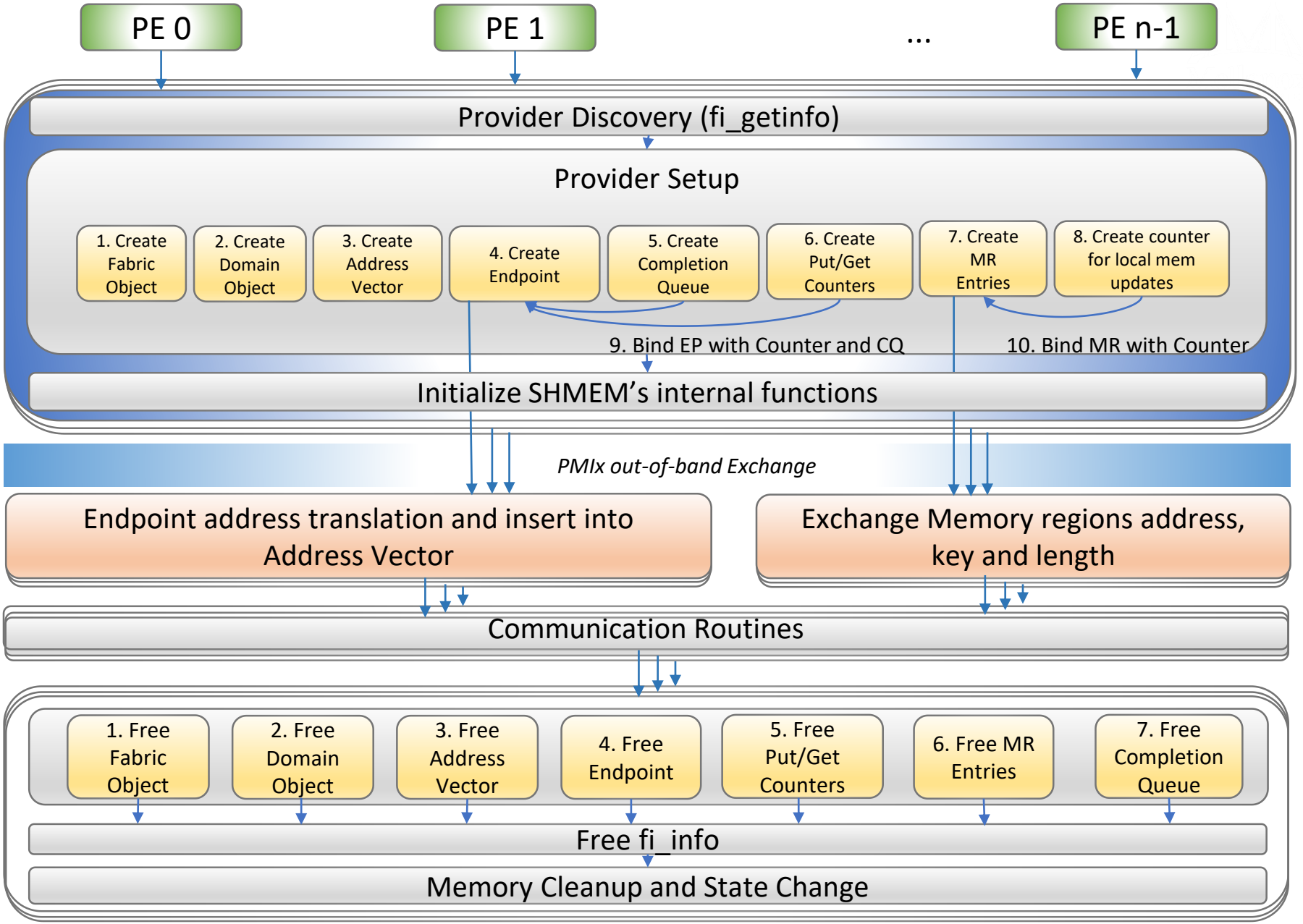


Atomic Operations

- 32 and 64 bit atomic operation support

Atomic Operation	Libfabric Function	Libfabric Datatype	Libfabric Operation
fadd (Fetch and Add)	fi_fetch_atomic	FI_INT32 FI_INT64	FI_SUM
finc (Fetch and Increment)	fi_fetch_atomic	FI_INT32 FI_INT64	FI_SUM
add (Add)	fi_inject_atomic	FI_INT32 FI_INT64	FI_SUM
inc (Increment)	fi_inject_atomic	FI_INT32 FI_INT64	FI_SUM
cswap (Compare and Swap)	fi_compare_atomic	FI_INT32 FI_INT64	FI_CSWAP_NE
swap (Swap)	fi_fetch_atomic	FI_DOUBLE FI_FLOAT FI_INT32 FI_INT64	FI_ATOMIC_WRITE

Mapping of OpenSHMEM-X atomic operations with Libfabric



Steps Involved in OpenSHMEM-X Libfabric Conduit

Outline

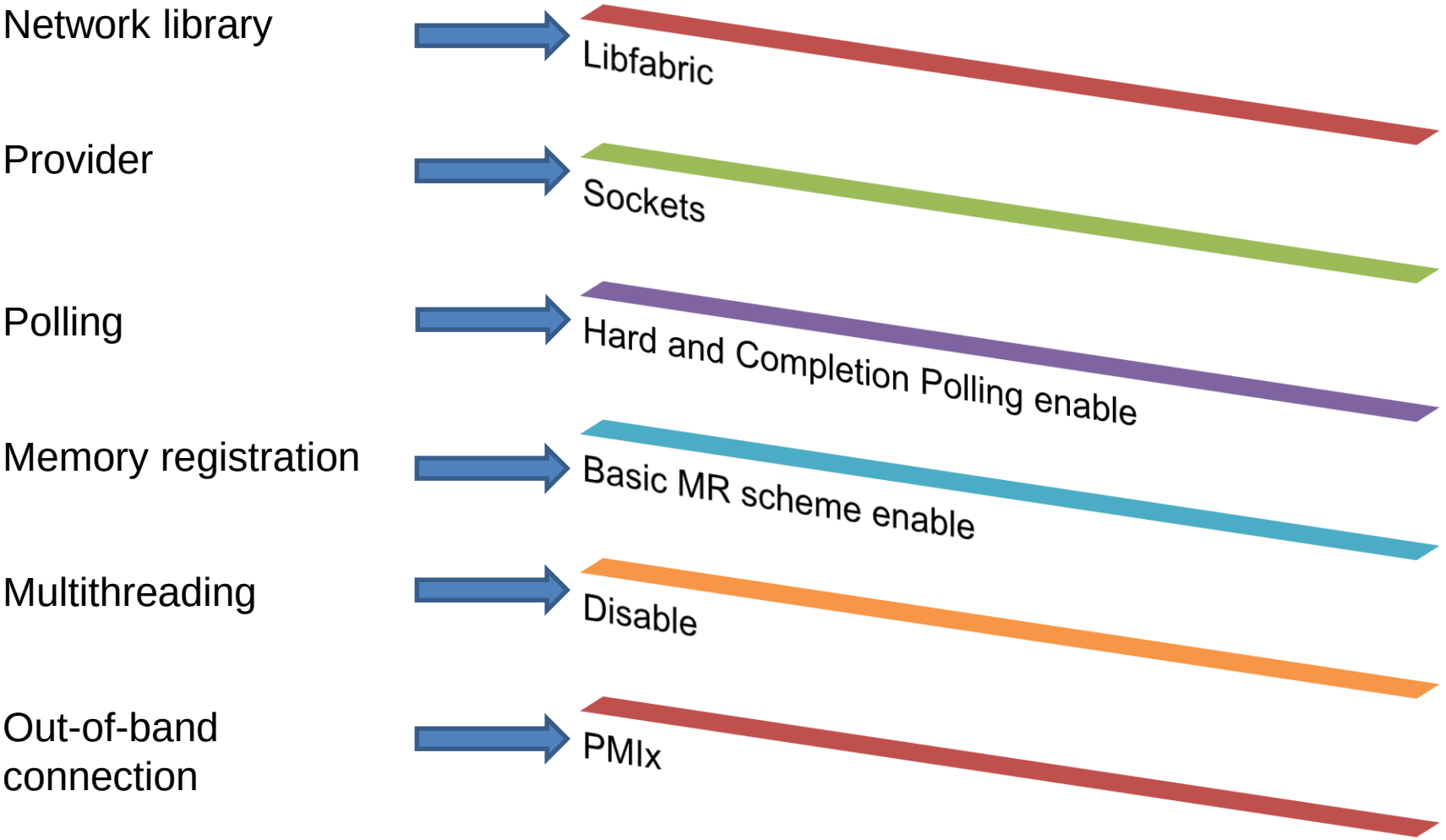
- Background and Motivation
- Design of Libfabric Conduit
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion and Future Works

Environment Used

- Eos
  - Cray® XC30™ cluster
  - Cray's Aries interconnect & Dragonfly topology
  - 736 compute nodes (Intel® Xeon® E5-2670 processor)
    - 16 cores per node
    - Total 11,776 traditional processor cores
    - 23,552 logical cores with Intel Hyper-Threading
    - 64GB memory per node

  - https://www.olcf.ornl.gov/for-users/system-user-guides/eos/system-overview/

Configuration: SOS and OpenSHMEM-X



Benchmark tools

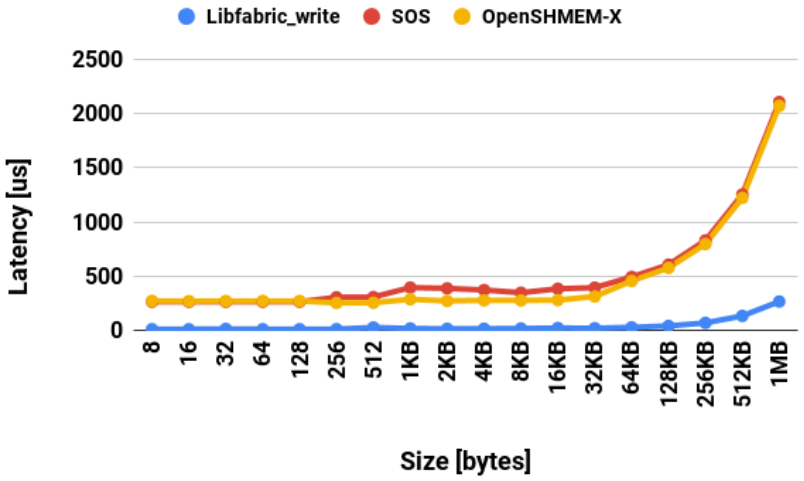
- OpenSHMEM micro-benchmarking suite is used for :
  - PUT Bandwidth and Latency
  - GET Bandwidth and Latency

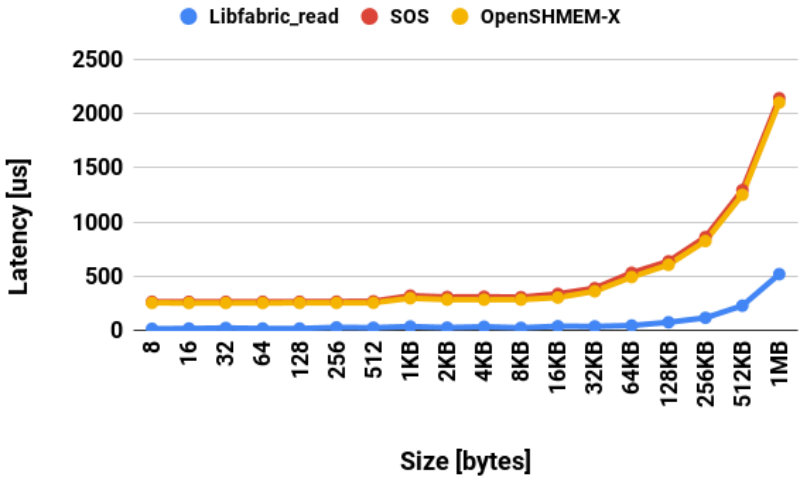
  Message Size: 8 bytes to 1 MB

- OSU Micro-Benchmarks 5.4.3 is used for :
  - FADD Latency
  - FINC Latency
  - ADD Latency
  - INC Latency
  - SWAP Latency
  - CSWAP Latency

  Message Size: 32 bit and 64 bit

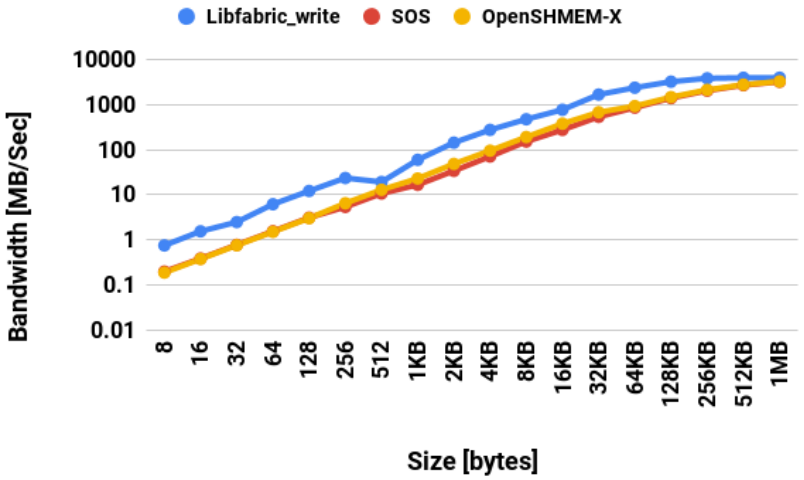
Latency (Put and Get)

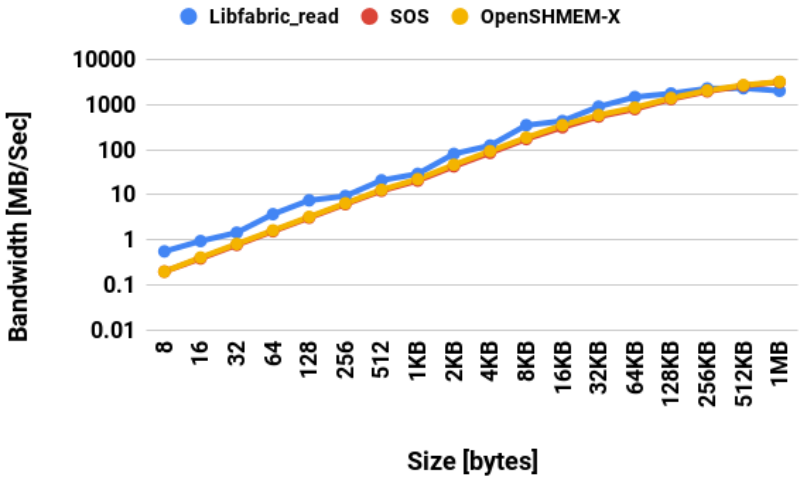




- Put Operation:
  - SOS latency performance is slightly better(around 2-5%) for message size upto 128 bytes
  - Upto 29% improvement for 2 KB message size
- Get Operation:
  - Gradual improvement in latency with 10% latency improvement for 16 KB message size

Bandwidth (Put and Get)

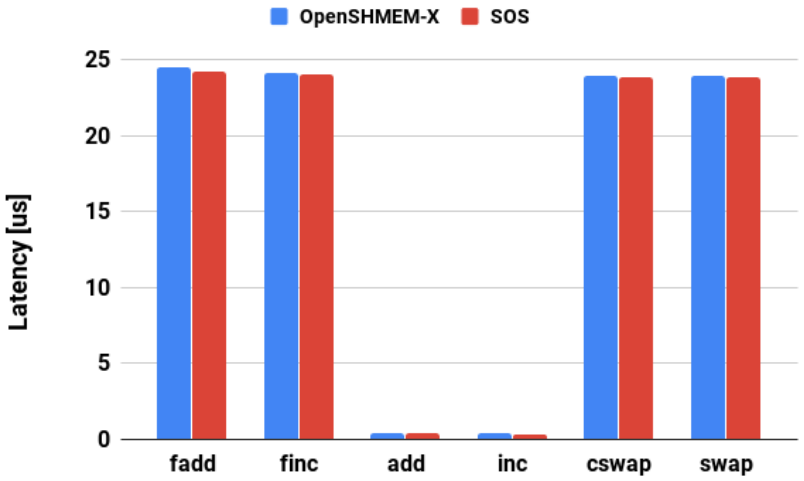




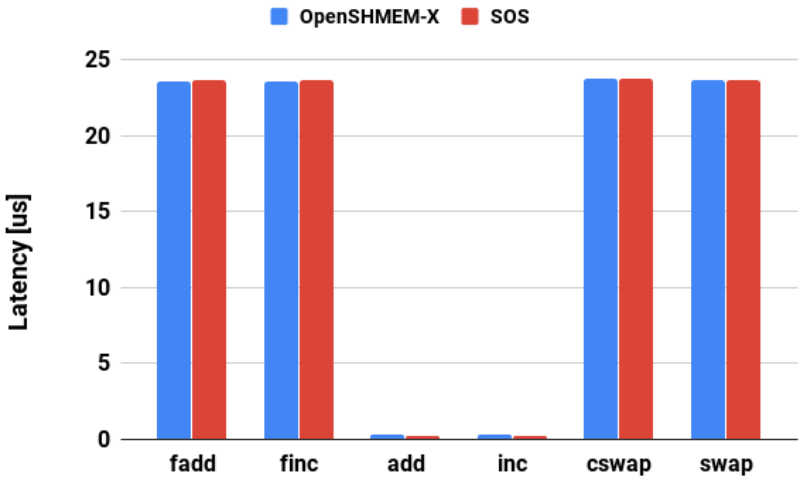
- Put Operation:
  - SOS put operation performance is 2-5% better for message sizes upto 128 bytes*
  - Up to 42% better bandwidth for 2 KB message size.
- Get Operation:
  - Performance improvement up to 11%

*SOS uses fi_inject and bounce buffer for small and medium messages respectively

Atomic Operations



32 bit atomic operation



64 bit atomic operation

- Our implementation shows similar or slightly better latency measurements than Sandia OpenSHMEM for atomic operations
- fi_inject_atomic helps to reduce the latency of add and inc operation

Outline

- Background and Motivation
- Design of Libfabric Conduit
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion and Future Works

Conclusion

- Libfabric offers a set of network library
  - minimizes the semantic gap
  - maintains application performance
  - delivers scalability

- Our initial implementation supports sockets as the provider
  - provides a basic visualization of integrating Libfabric with OpenSHMEM-X.

- Sockets provider is available on many systems.

Future Works

- Enabling other providers will enhance the portability of OpenSHMEM-X

- Currently working on enabling the support for uGNI provider to get the performance measurements using gni in Cray machines

- We are also working on additional tuning and optimizations

Acknowledgment