OPENFABRICS ALLIANCE

2024 OFA Virtual Workshop

OFI INTEGRATED SHARED MEMORY OFFLOAD

Alexia Ingerson, Intel

Shi Jin, AWS

Amir Shehata, ORNL

intel

aws

Oak Ridge National Laboratory

U.S. DEPARTMENT OF ENERGY

OVERVIEW

What is the peer provider and how does it work?

What has changed since last year?

How did AWS use it in its efa provider?

What issues did they have and how did they solve them?

How did using shm as a peer help efa?

What is the link provider?

What does a provider need in order to leverage LINKx support?

What is the current status and direction of the provider?

What are future extensions?

- **Peer provider architecture provides a way for sharing resources between two or more providers**
  - Target use case is for integrated shm offload
- **AWS efa provider was using shm provider deep within efa protocols to offload local communication but moved to peer provider infrastructure**
- **ORNL has been developing a new “link” provider (LINKx) to allow any provider to offload to shm without having to manage two providers**

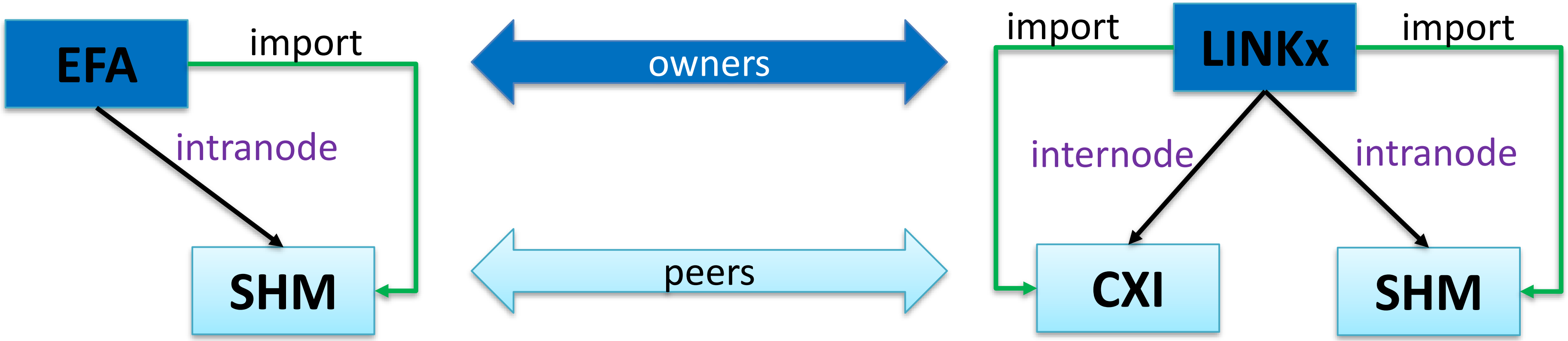
PEER PROVIDER OVERVIEW

- **Expose one endpoint to app while using two providers**
  - One for external, internode communication (verbs, tcp, efa, cxi, etc)
  - One for internal, intranode communication (shm)
- **Share provider resources**
  - Write to same CQ
  - Update same counters
  - Get receive buffers from the same receive context (SRX)
  - Share addressing (e.g. fi_addr)
- **All sharing and coordination is done internally, no application changes necessary**
- **“Owner” vs “peer”**
  - Owner owns resource and exports it for use by a peer
  - Peer cannot directly access owner resource – has to use imported ops

PEER PROVIDER EXAMPLES

EFA offloads to shm for local communication

Link provider handles coordination, providers only need to import



EFA redirects intranode transfers to shm

LINKx picks provider based on target address

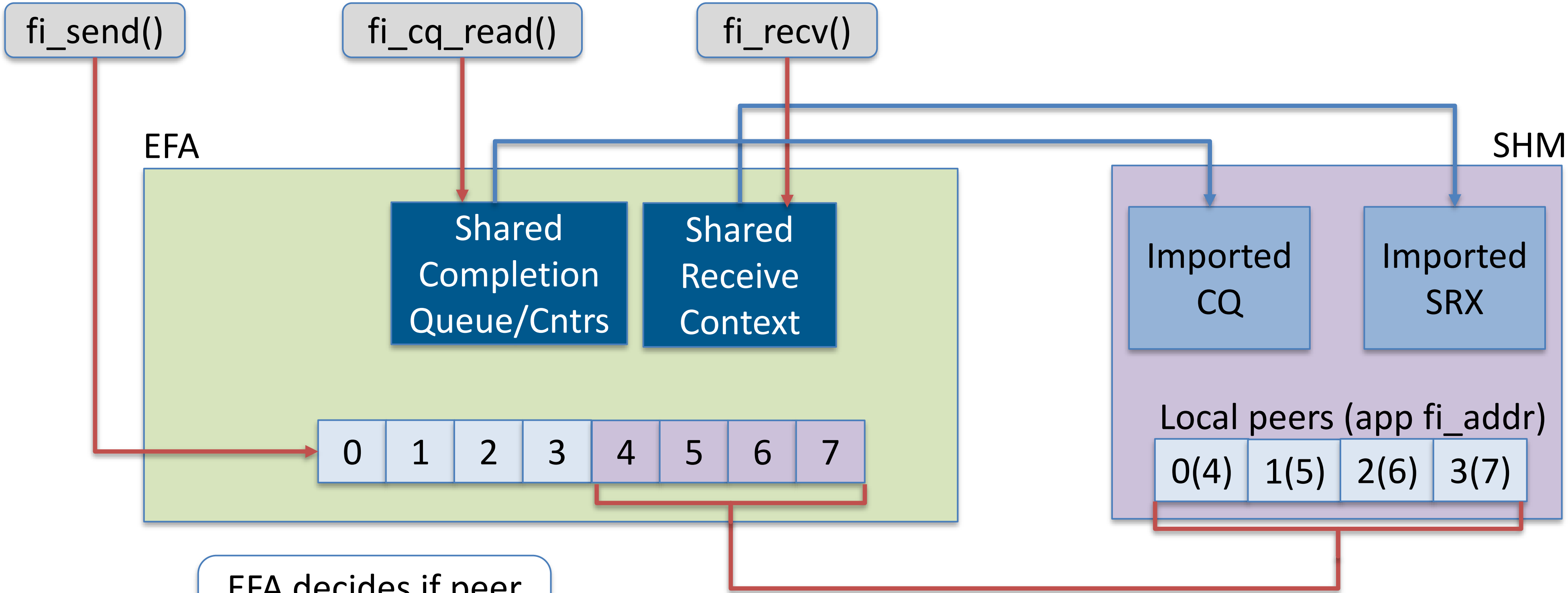
EFA owns CQ, cntr, and SRX

LINKx owns CQ, cntr, and SRX

SHM accesses EFA-owned resources

SHM and CXI access LINKx resources

EXAMPLE - OWNER: EFA

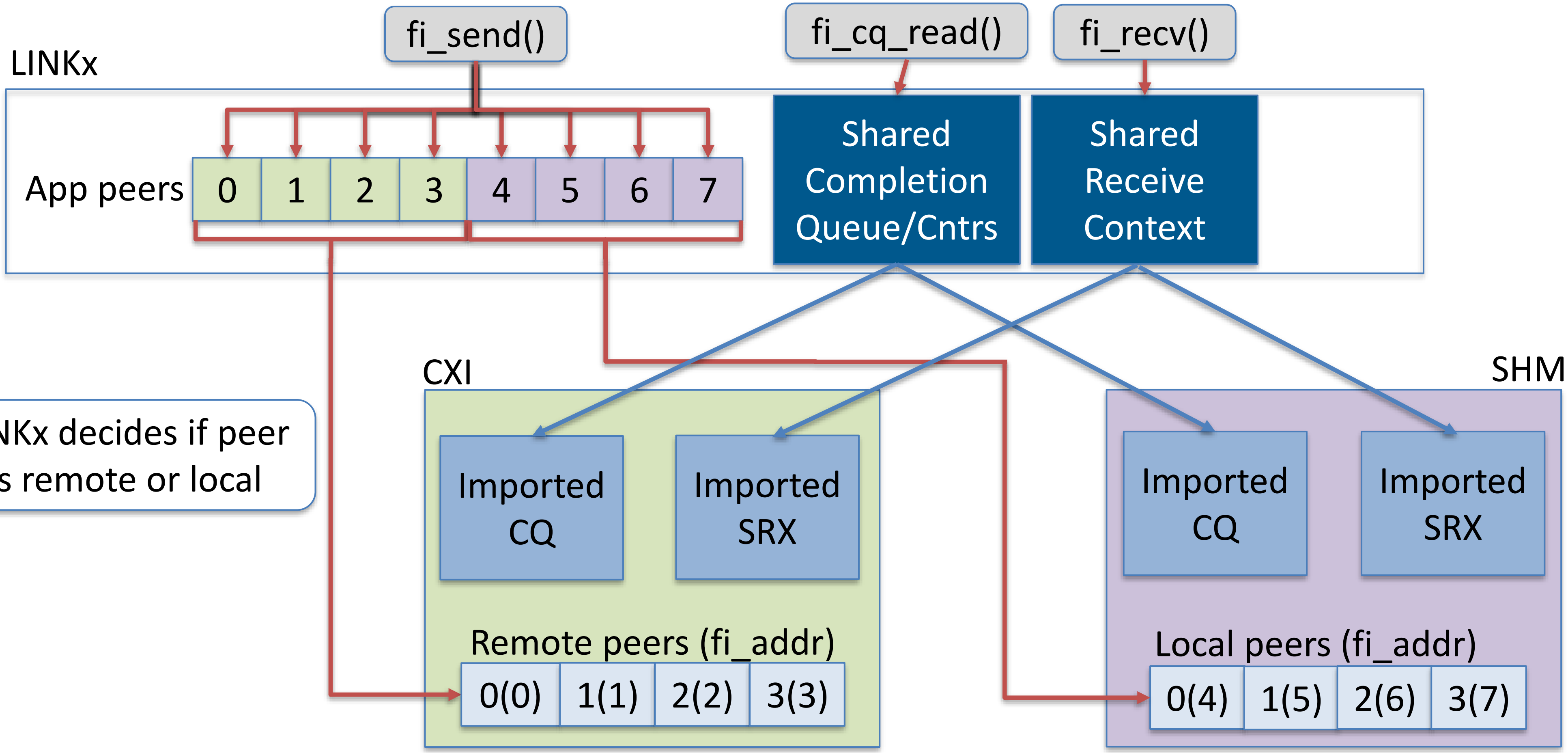


SHM writes completions directly into EFA-owned CQ and gets receive buffers from EFA-managed SRX

EFA decides if peer is remote or local

EFA adds addresses into SHM's AV with FI_AV_USER_ID which allows SHM to report the application fi_addrs in the CQ

EXAMPLE - OWNER: LINKX



LINKx decides if peer is remote or local

CXI and SHM write completions directly into LINKx-owned CQ and get receive buffers from LINKx-managed SRX

LINKx adds addresses into SHM and CXI's AVs with FI_AV_USER_ID which allows peer providers to report the application fi_addrs in the CQ

SHARED COMPLETION QUEUE API

1. Owner allocates a peer cq and defines peer CQ write ops

```
struct fid_peer_cq {
    struct fid fid;
    struct fi_ops_cq_owner *owner_ops;
};
        struct fi_ops_cq_owner {
            ssize_t (*write)();
            ssize_t (*writeerr)();
        };
```

3. Peer calls imported peer_cq->owner_ops in order to write an entry to the shared CQ

2. Owner calls fi_cq_open, passing in the peer_cq via context indicating a peer with attr->flags | FI_PEER

```
fi_cq_open(peer_domain, &attr, &peer_cq, peer_context);

                            struct fi_peer_cq_context {
                                struct fid_peer_cq *cq;
                            };
```

SHARED COUNTER API

1. Owner allocates a peer cntr and defines peer cntr write ops

```
struct fid_peer_cntr {
    struct fid fid;
    struct fi_ops_cntr_owner *owner_ops;
};
        struct fi_ops_cntr_owner {
            ssize_t (*inc)(…);
            ssize_t (*incerr)(…);
        };
```

3. Peer calls imported peer_cntr->owner_ops in order to increment the shared counter

2. Owner calls fi_cntr_open, passing in the peer_cntr via context indicating a peer with attr->flags | FI_PEER

```
fi_cntr_open(peer_domain, &attr, &peer_cntr, peer_context);

                          struct fi_peer_cntr_context {
                              struct fid_peer_cntr *cntr;
                          };
```

SHARED RECEIVE CONTEXT

```
struct fi_peer_srx_context {
     struct fid_peer_srx *srx;
};

struct fid_peer_srx {
     struct fid_ep ep_fid;
     struct fi_ops_srx_owner *owner_ops;
     struct fi_ops_srx_peer *peer_ops;
};

struct fi_ops_srx_owner {
     int (*get_msg)(…);
     int (*get_tag)(…);
     int (*queue_msg)(…);
     int (*queue_tag)(…);
     void (*foreach_unspec_addr)(…);
     void (*free_entry)(…);
};

struct fi_ops_srx_peer {
     int (*start_msg)(…);
     int (*start_tag)(…);
     int (*discard_msg)(…);
     int (*discard_tag)(…);
};
```

1. Owner creates peer_srx_context and sets owner ops

2. Owner exports SRX into peer by calling fi_srx_context passing in the peer_srx via context indicating a peer with attr->flags | FI_PEER. Peer sets peer_ops

```
fi_srx_context(peer_domain, &attr, &srx_fid, peer_srx_context);
```

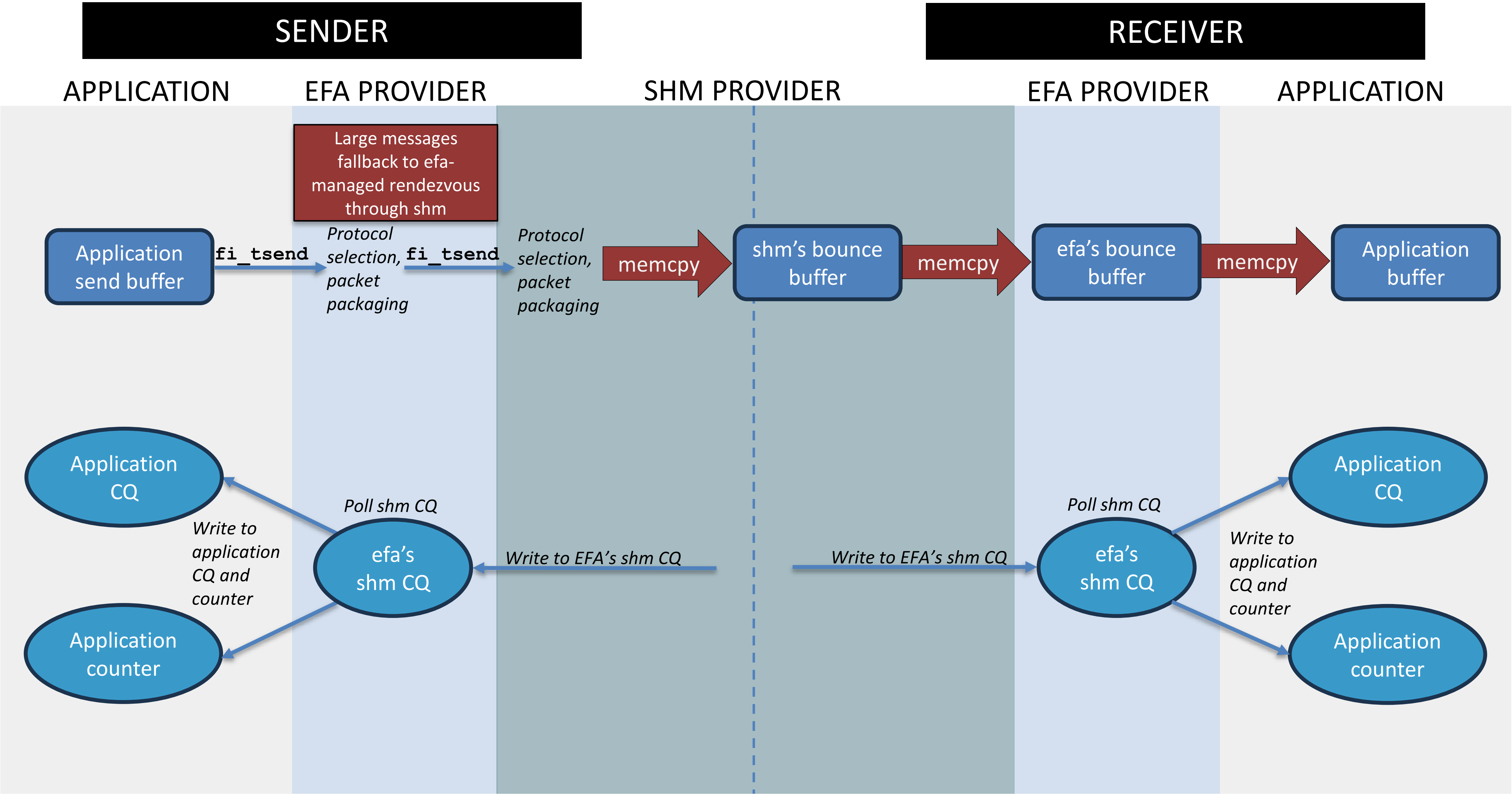
Peer calls owner ops to get, queue, and free messages

New owner op added to notify owner that AV update has occurred and unexpected messages from an unknown source might need to be updated

Owner calls peer ops to start and discard unexpected messages

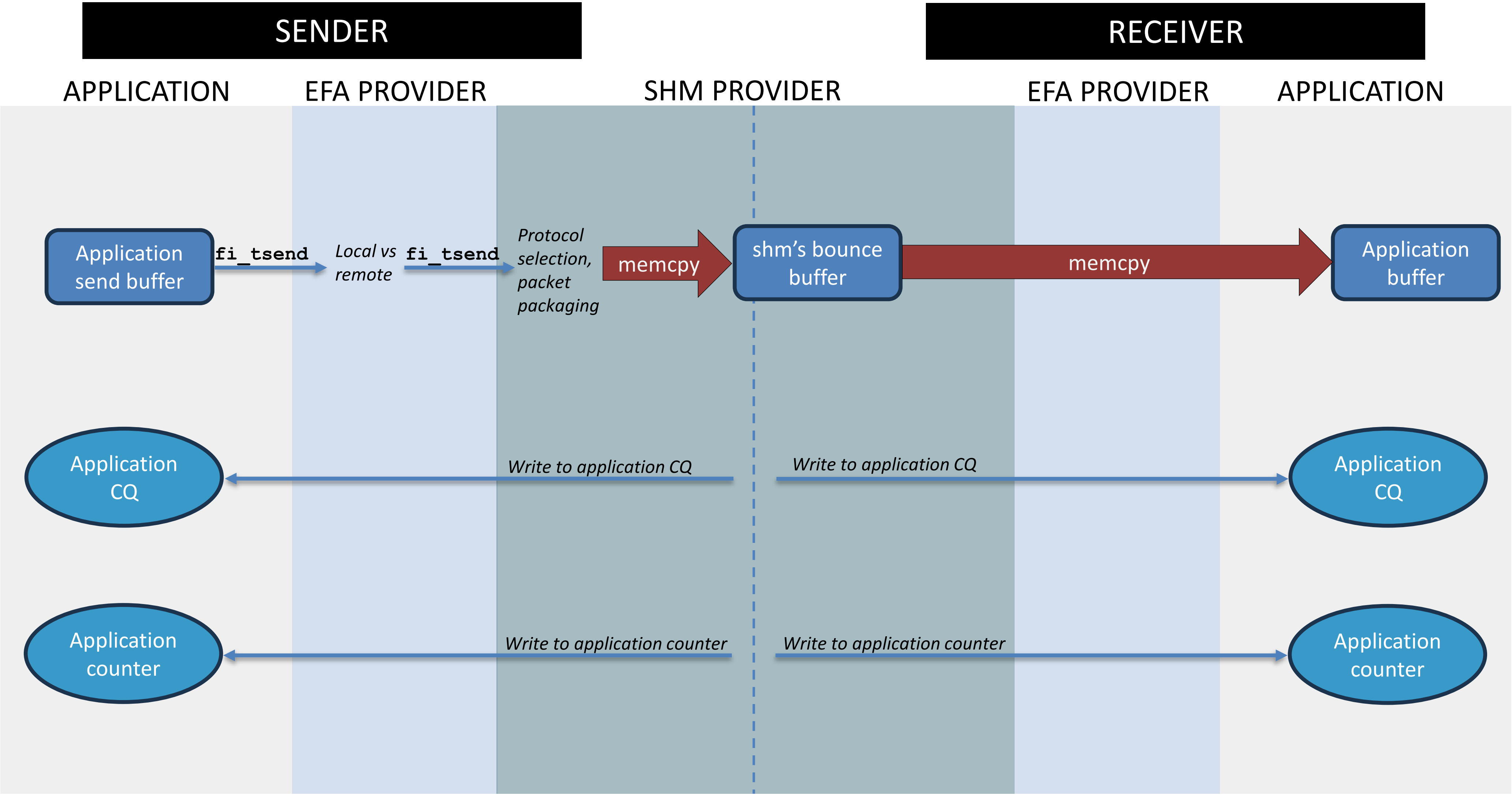
EFA SHM OFFLOAD INTEGRATION

A DAY IN LIFE OF MESSAGES THROUGH EFA + SHM PROVIDER



BEFORE USING PEER PROVIDER

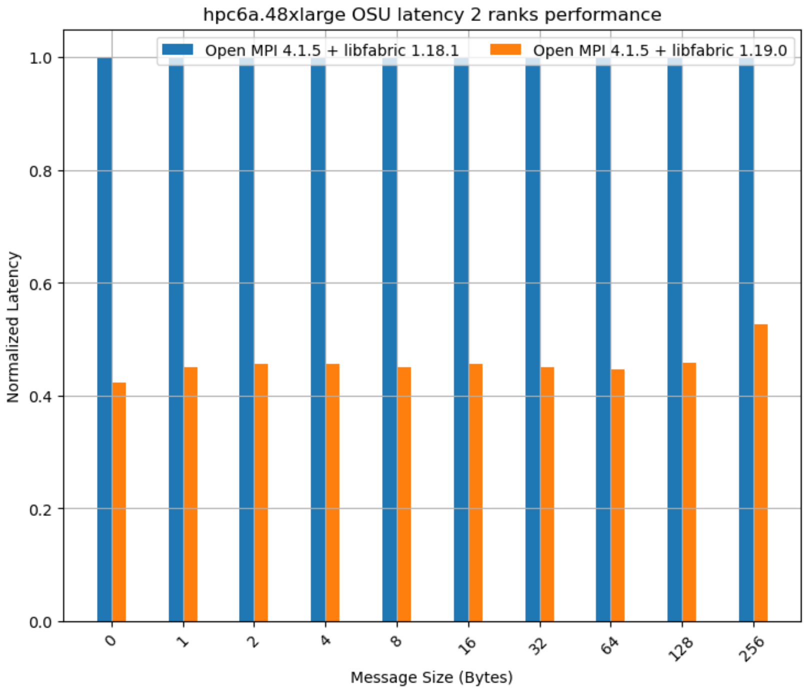
A DAY IN LIFE OF MESSAGES THROUGH EFA + SHM PROVIDER

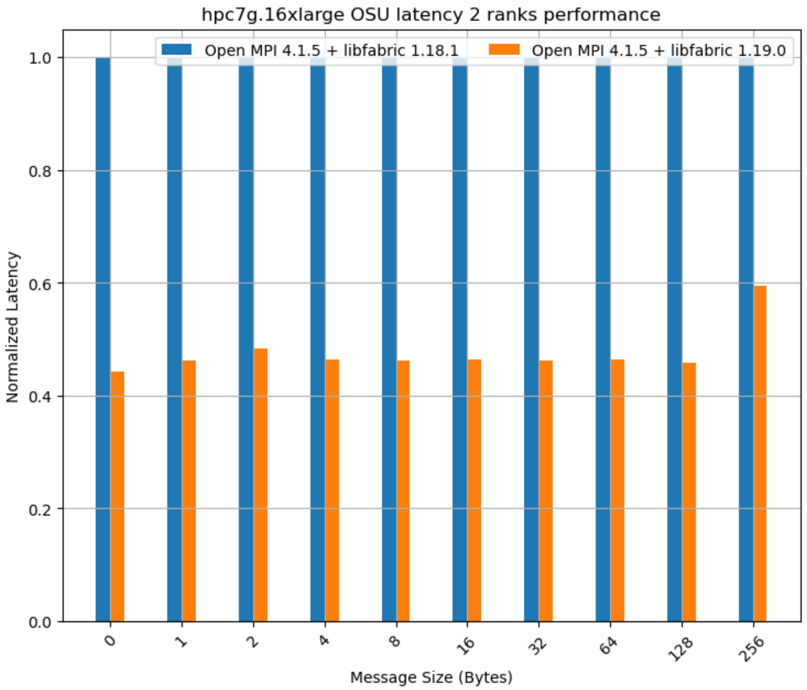


AFTER USING PEER PROVIDER

PERFORMANCE BOOST FOR EFA + SHM

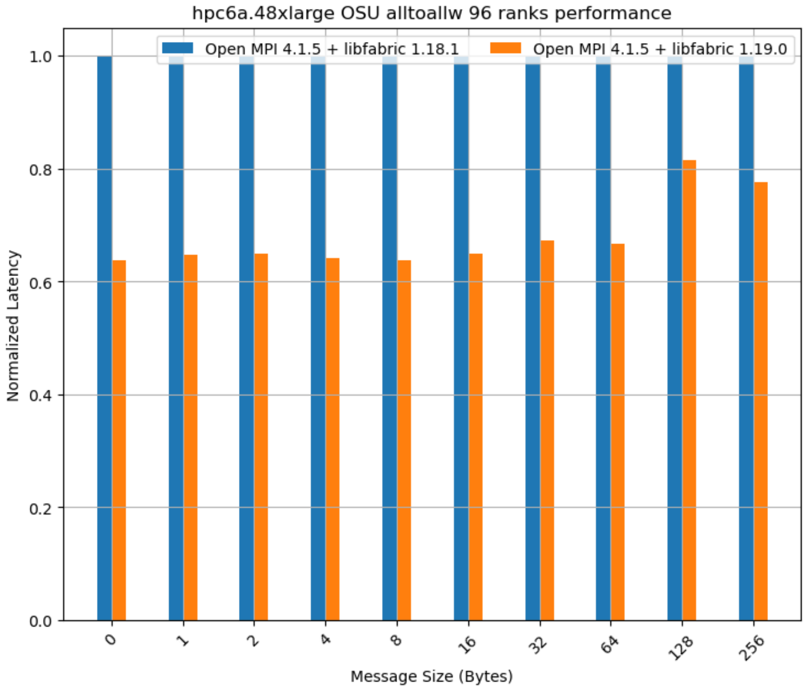
OSU latency

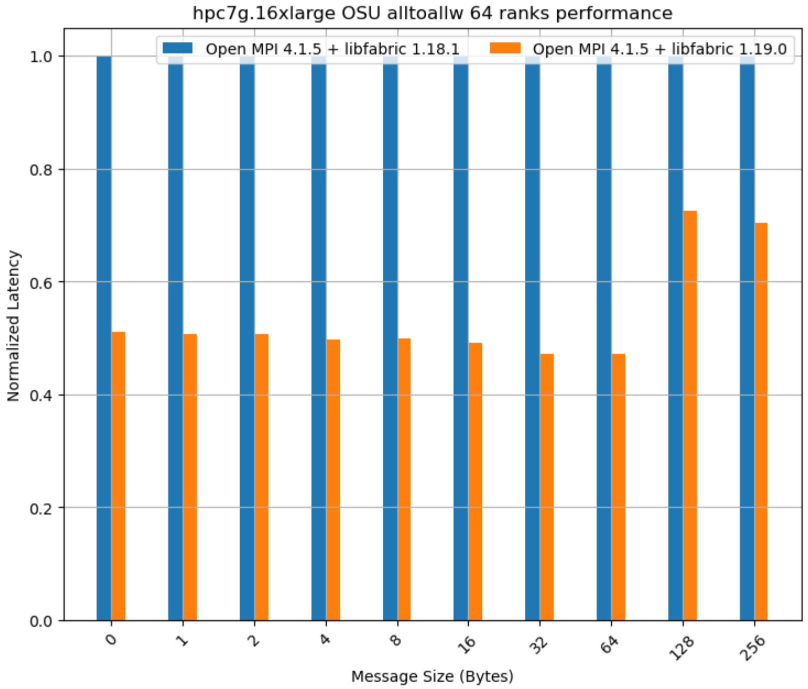

hpc6a.48xlarge OSU latency 2 ranks performance


hpc7g.16xlarge OSU latency 2 ranks performance

PERFORMANCE BOOST FOR EFA + SHM

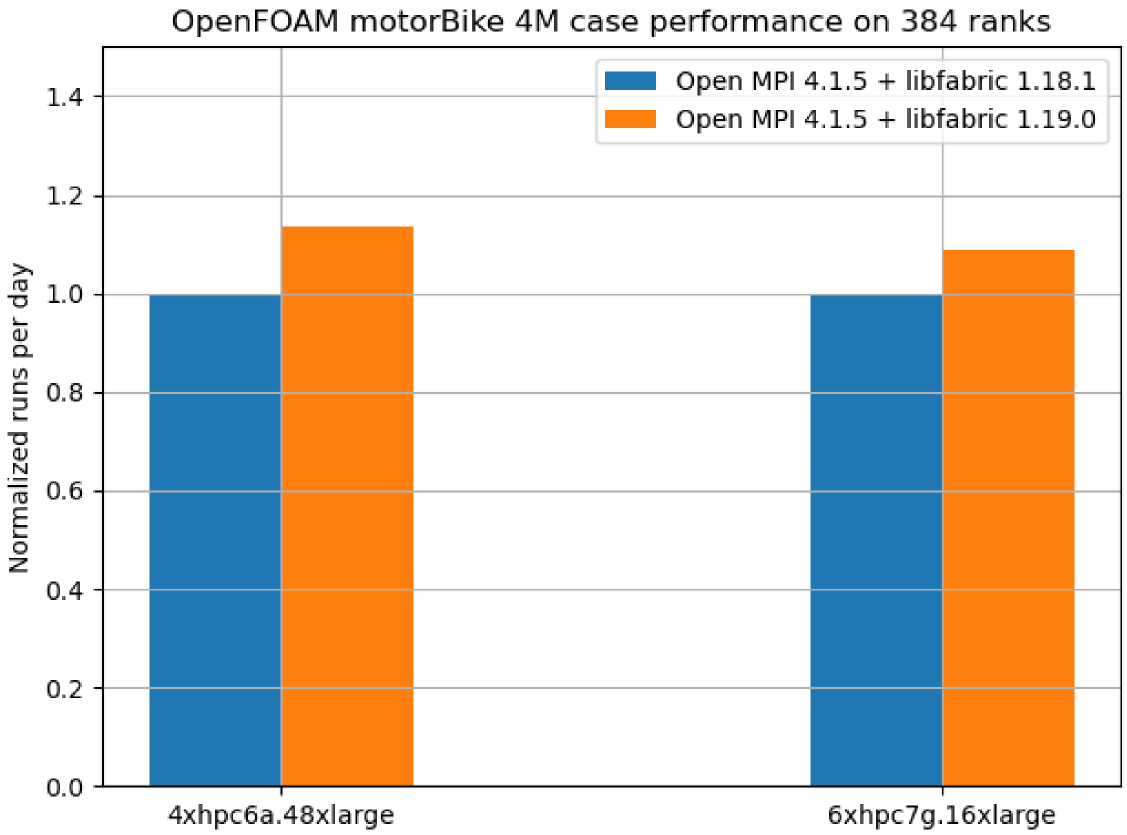
OSU alltoallw


hpc6a.48xlarge OSU alltoallw 96 ranks performance


hpc7g.16xlarge OSU alltoallw 64 ranks performance

PERFORMANCE BOOST FOR EFA + SHM

OpenFOAM MotorBike4M



CONCERNS AND FUTURE WORK

- **Discrepancy in provider’s ability to handle unexpected messages**
  - Efa provider can handle unlimited number of unexpected messages (until memory exhaustion) while shm provider’s CMA protocol (>4KB) can only handle up to rx size (1K default)
  - Before using peer provider model, efa provider handled unexpected message buffering for shm
  - After using peer provider model, unexpected messaging handed off to shm provider and exposes restriction
- **Locking strategy**
  - Need a dedicated lock to protect shared receive context resources accessed by data progress call (fi_cq_read) and transmission calls (fi_*send*)
  - Currently this lock created as domain level lock which can cause locking contention when domain is shared by multiple EPs
- **MR sharing**
  - MR descriptors interpreted by providers differently (shm uses `struct ofi_mr *` while efa uses `struct efa_mr *`)
  - Memory needs to be registered twice for each provider and translation needed when passing descriptors between providers
  - Need better way to share MR descriptor between providers

2024 OFA Virtual Workshop

LINKX PROVIDER

Amir Shehata, Systems Engineer

Oak Ridge National Lab

LINKx Provider

Amir Shehata

Advanced Tehnology Section

Technology Integration Group

shehataa@ornl.gov

ORNL is managed by UT-Battelle LLC for the US Department of Energy

Overview

- **Provide an alternative MPI software stack using Open MPI on the Frontier supercomputer**

- Users need more choices of MPI implementations
  - Work around problems
  - Try out new features
- Vendor only provides Cray MPI on Frontier via a libfabric provider, CXI.
- CXI has no shared memory offload
- **Solution is to develop a new libfabric provider to link both CXI and SHM libfabric providers**
- Solution has been tested and deployed on Frontier

Available Solutions

- Cray supports Slingshot 11 via a new CXI libfabric provider
  - **BUT, CXI provider does not have shared memory offload**
- Two potential solutions:

  1) Use CXI provider through Open MPI’s MTL path and implement shared memory offload in libfabric

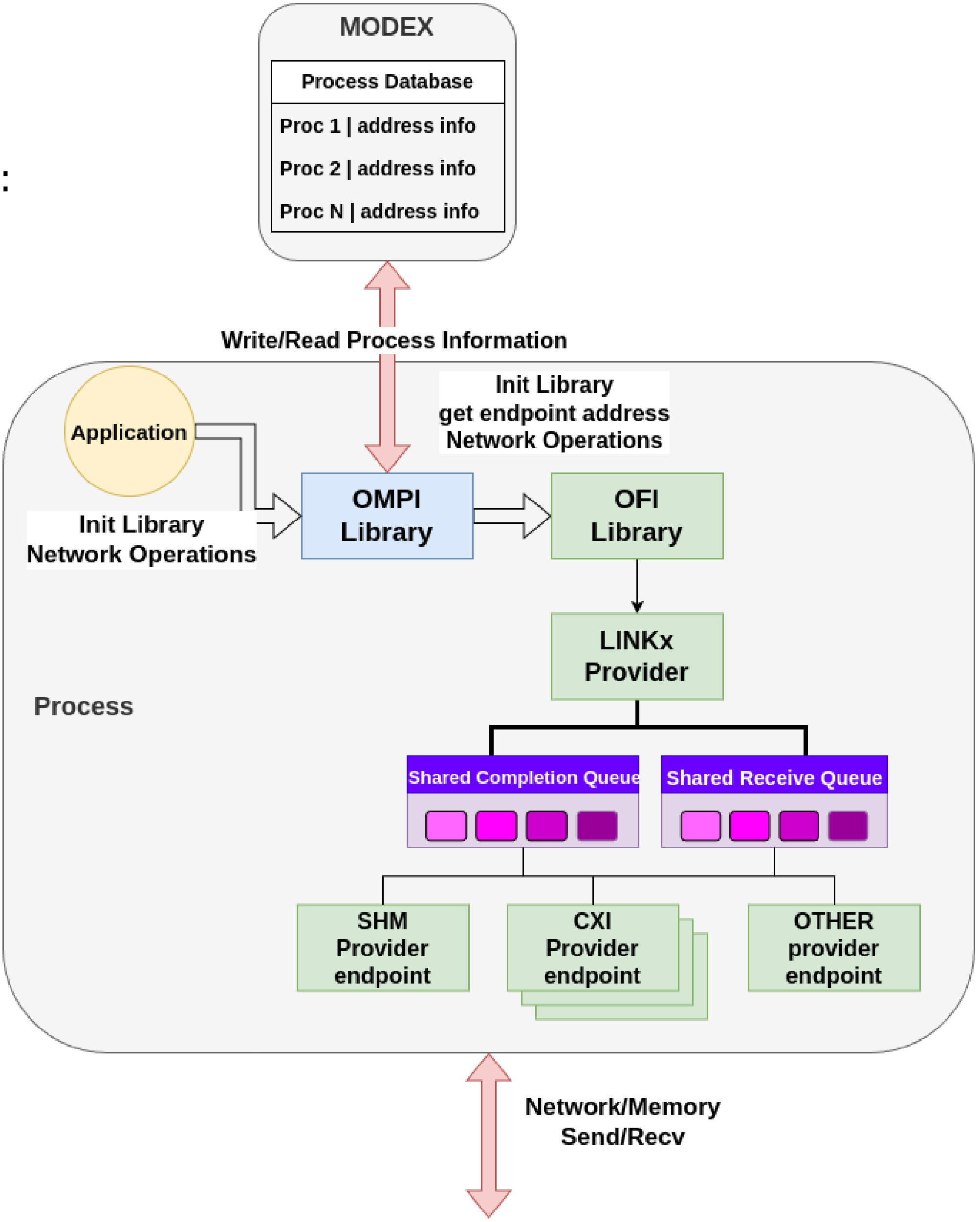
  2) Use CXI provider through Open MPI’s BTL path and use Open MPI shared memory module

Why libfabric?

- **Approach should be flexible to link any libfabric provider**
- BTL option restricts the solution to Open MPI
- **By pushing the shared memory offload to libfabric, then any application using libfabric may benefit from this feature**
- Having a separate provider, LINKx, avoids the need to implement the shared memory offload in every provider which needs SHM
- Solution should not be restricted to linking SHM, but be flexible to link any provider which supports the peer infrastructure
- This opens the potential for the following features:
  - Supporting heterogeneous interfaces
  - Supporting binding multiple interfaces (Multi-Rail)

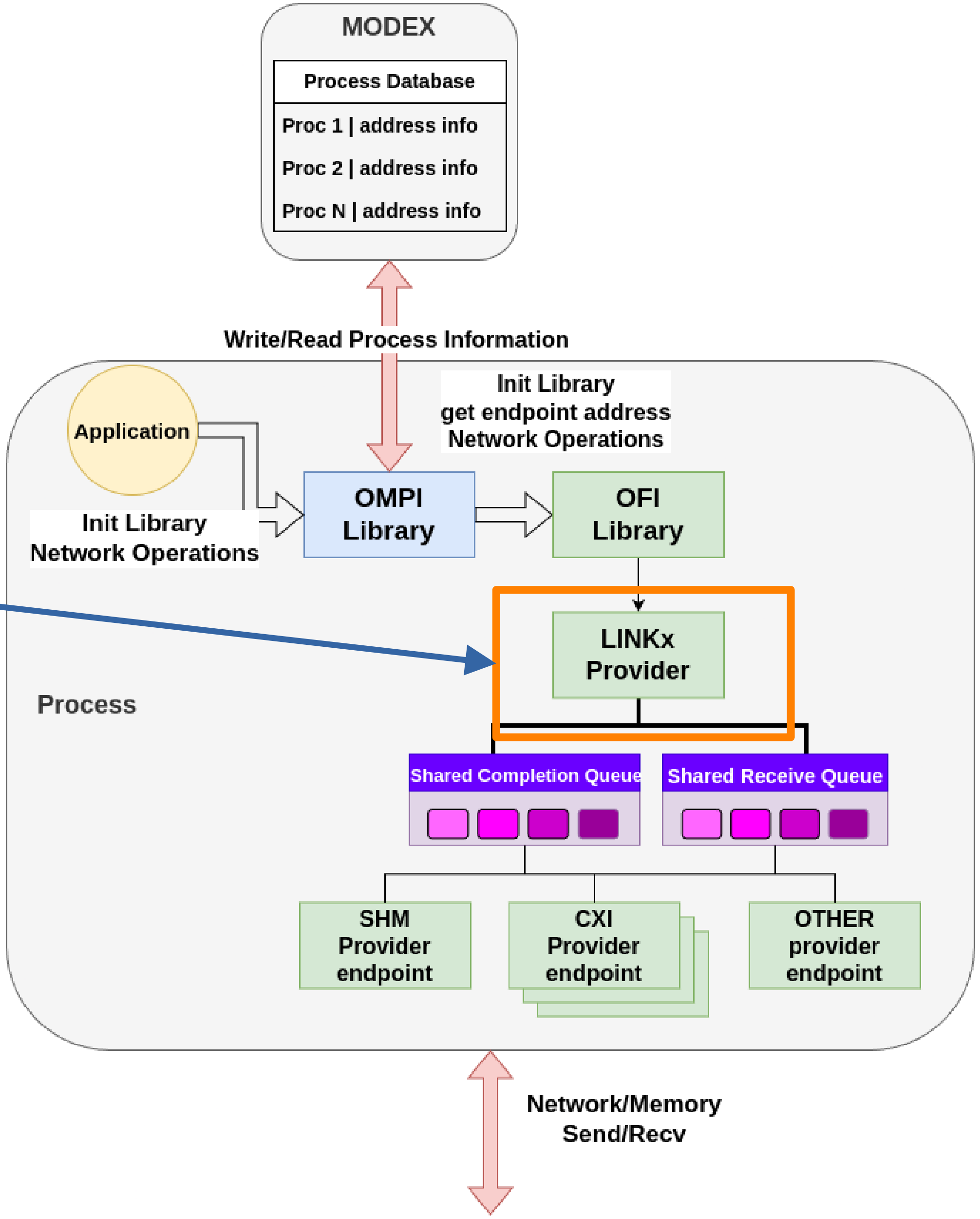
Solution Architecture

Full architectural overivew presented last year:
https://www.openfabrics.org/2023-ofa-virtual-workshop-agenda/



Solution Architecture

Introduce LINKx provider binds SHM and CXI

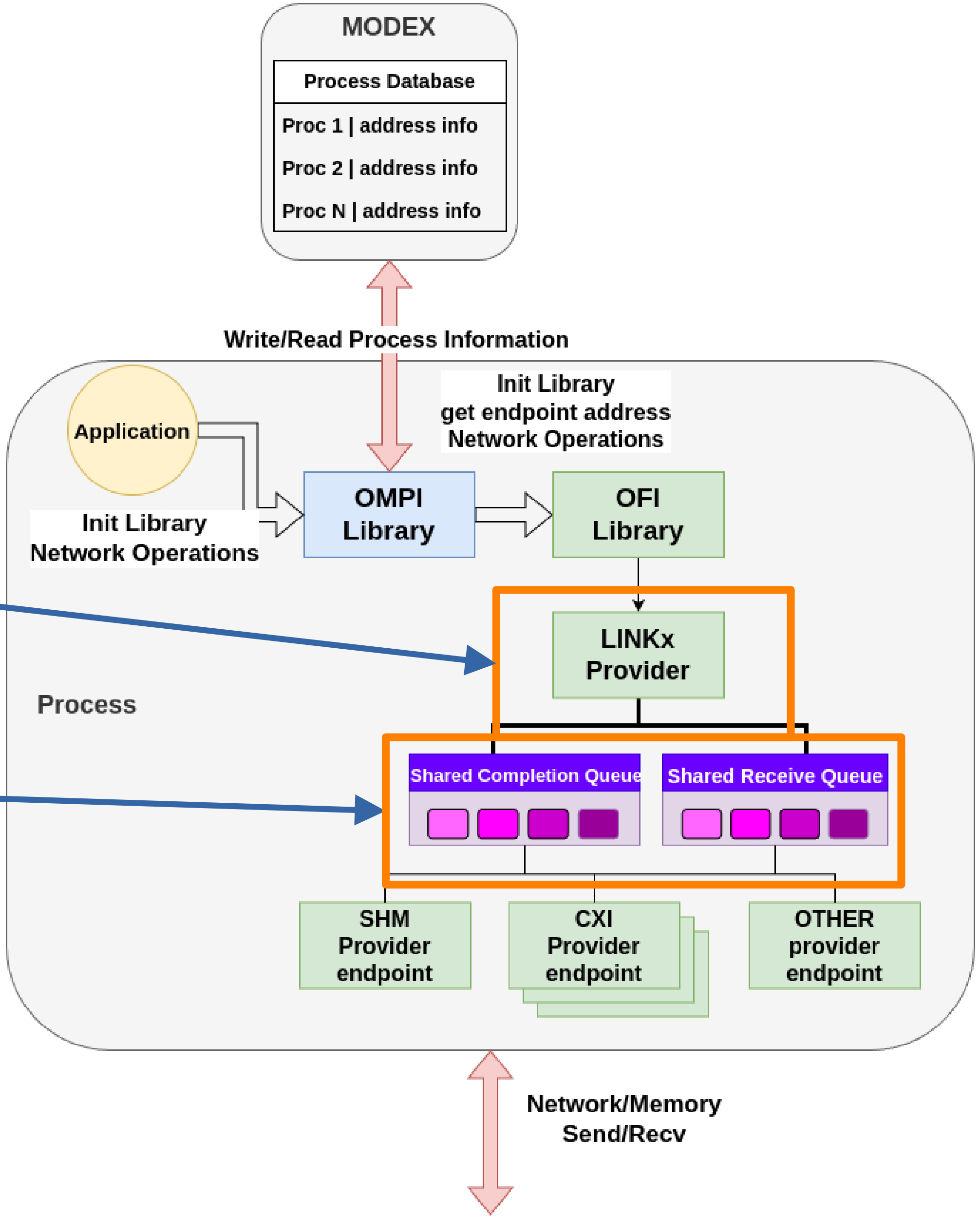


Solution Architecture

Introduce LINKx provider binds SHM and CXI

LINKx shares
- Completion Queues
- Shared Receive Queues

With core providers



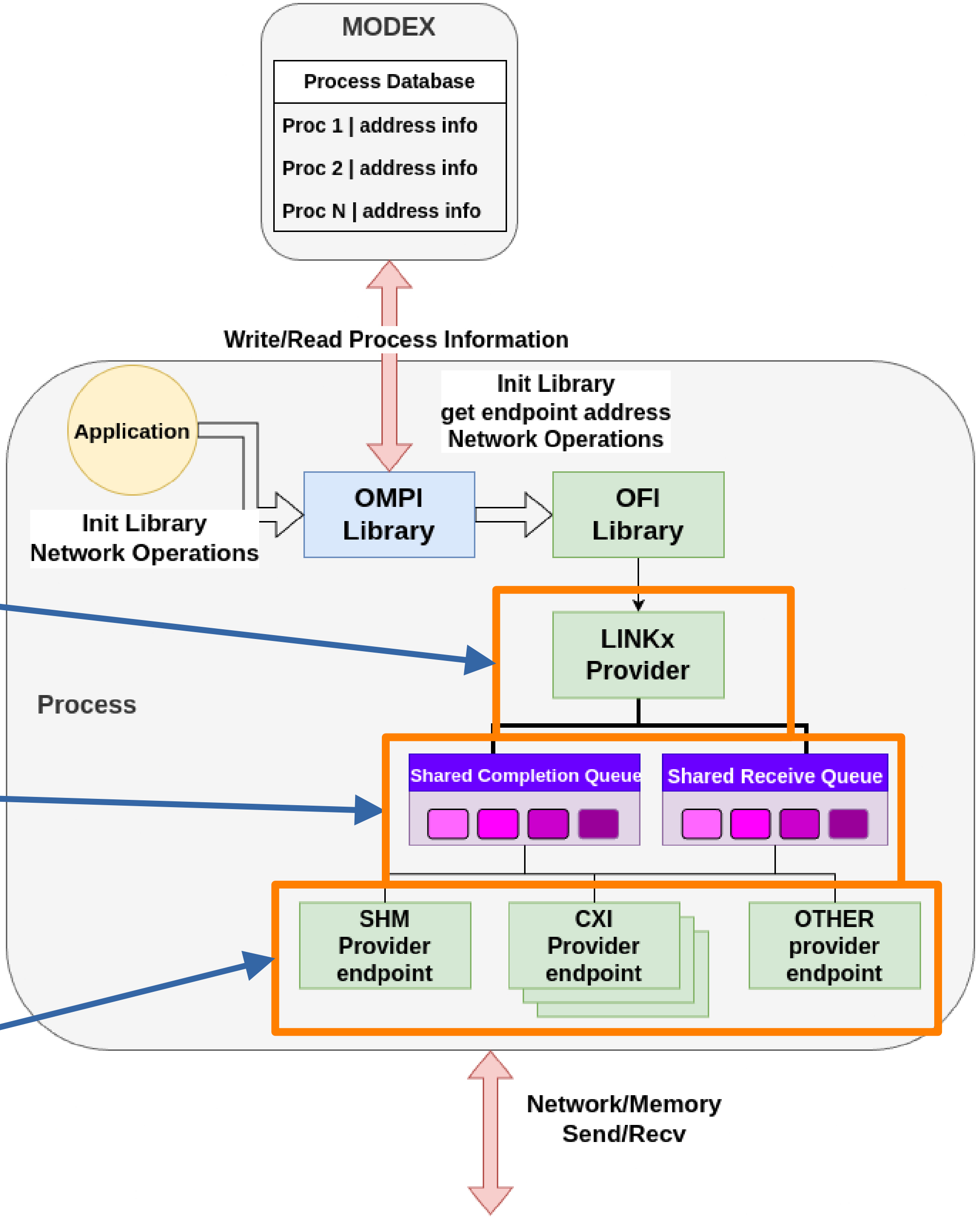
Solution Architecture

Introduce LINKx provider binds SHM and CXI

LINKx shares
- Completion Queues
- Shared Receive Queues

With core providers

LINKx selects core provider based on destination locality
- Intra-node use SHM provider
- Inter-node use CXI provider



LINKx Architecture

- **LINKx behaves both like an application and a provider**
  - Users of libfabric see LINKx as a provider they can select
  - LINKx behaves as an application in that it sets up “core” providers the same way an application would
- LINKx uses the peer infrastructure to share its:
  - Receive and unexpected queues
  - Completion queues
- Core providers pull receive requests from the shared queues and place completion events on LINKx’ completion queue.

LINKx Status

- **Currently in production on Frontier**
  - Available via module environments

- Tested Linking SHM with CXI
- Tested linking SHM with RXM
- It supports Tagged and RMA interfaces only
- It does not support counters

LINKx Usage

```
#### On Frontier
#> module load ums
#> module load ums024

#> export FI_LINKX_PROV_LINKS= “shm+tcp;ofi_rxm”
#> fi_info
...
provider: shm+tcp;ofi_rxm:linkx
    fabric: ofi_lnx_fabric
    domain: shm+hsn0:ofi_lnx_domain
    version: 120.0
    type: FI_EP_RDM
    protocol: FI_PROTO_SHM
```

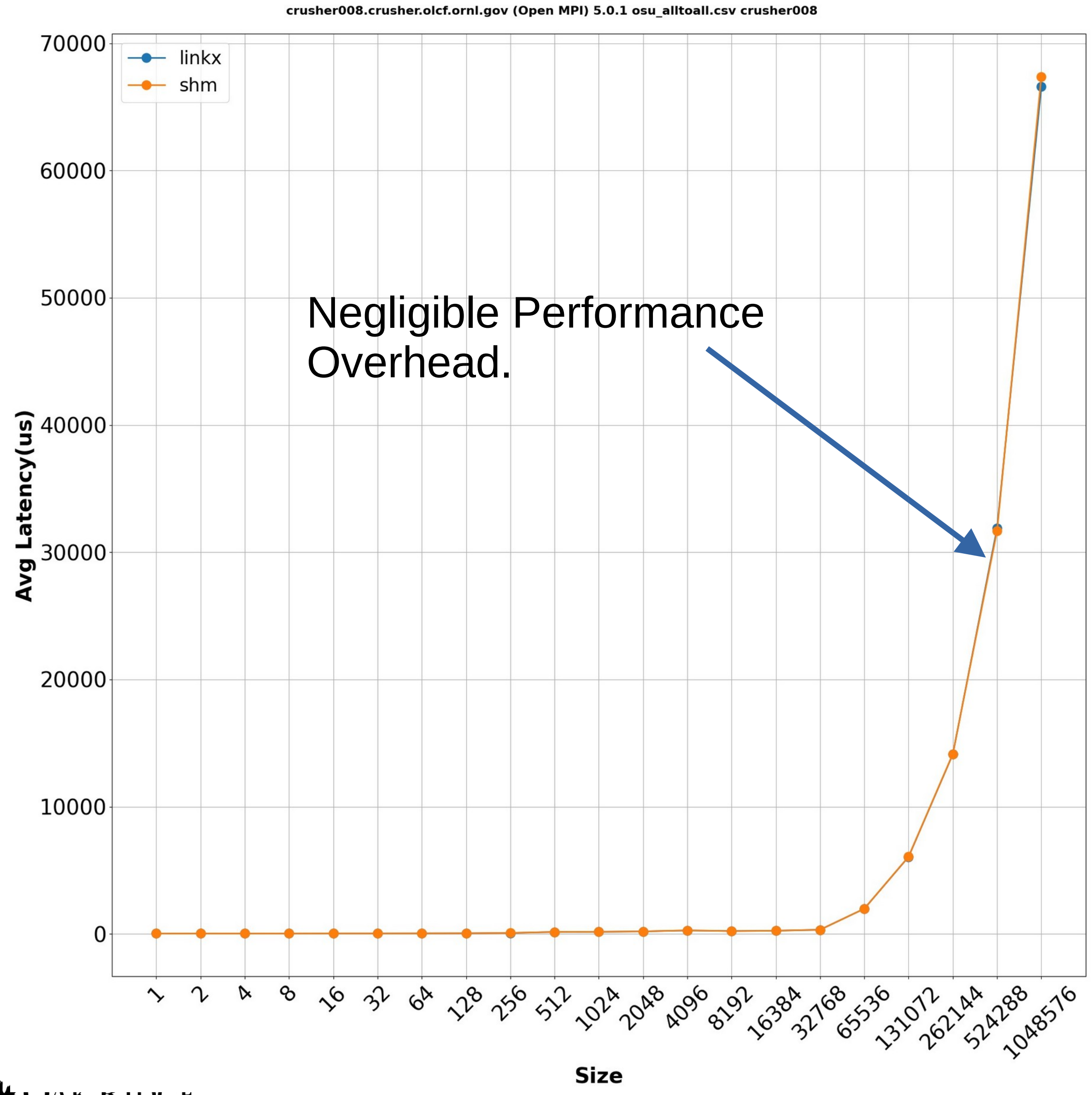
```
#### On Frontier
#> module load ums
#> module load ums024

#> export FI_LINKX_PROV_LINKS= “shm+cxi”
#> fi_info
...
provider: shm+cxi:linkx
    fabric: ofi_lnx_fabric
    domain: shm+cxi0:ofi_lnx_domain
    version: 120.0
    type: FI_EP_RDM
    protocol: FI_PROTO_SHM
```

- Application then selects linkx provider
- EX: In Open MPI the selection can be forced using mca parameter:

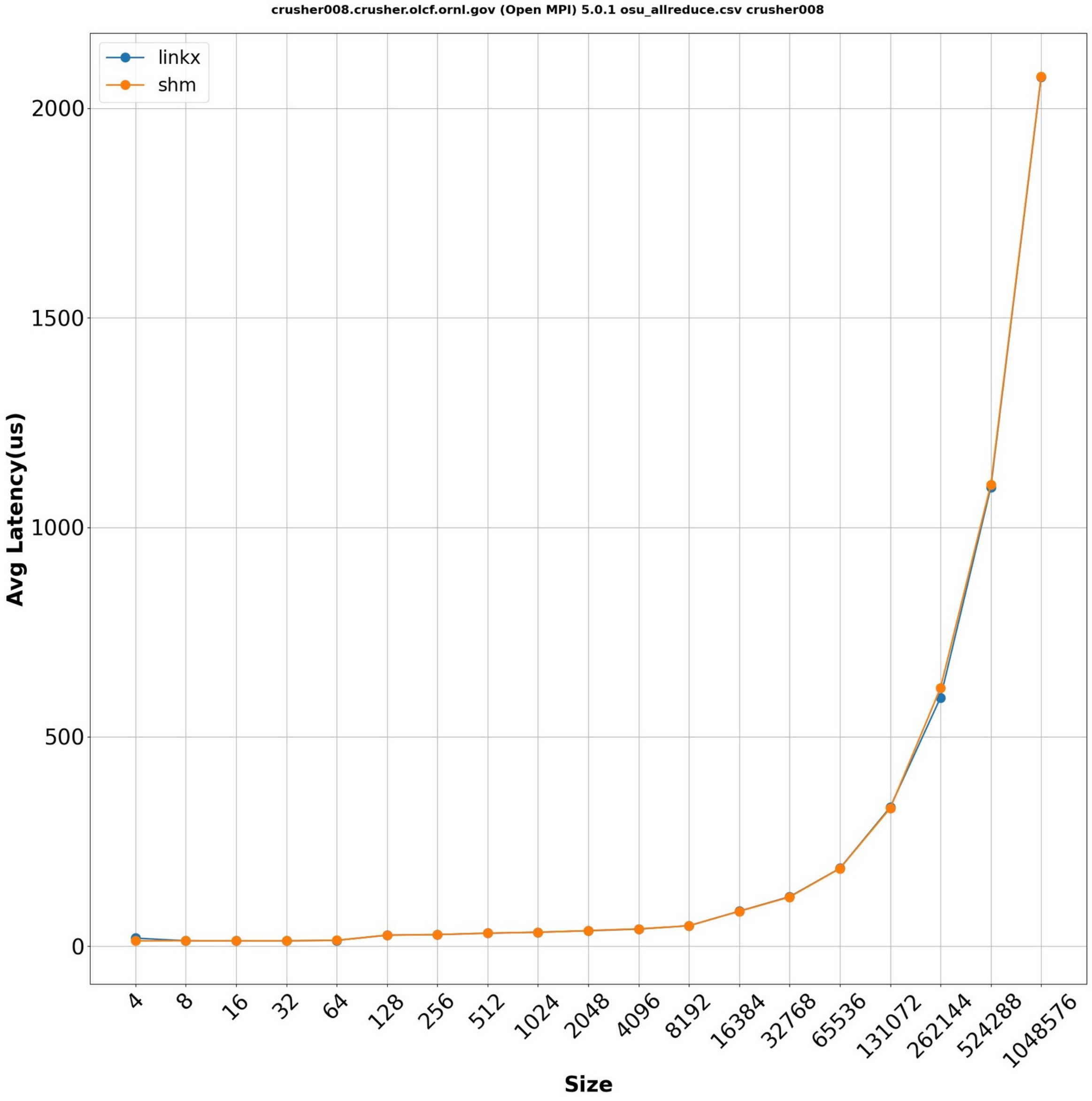
    opal_common_ofi_provider_include

Performance: LINKx vs SHM – 56 Processes



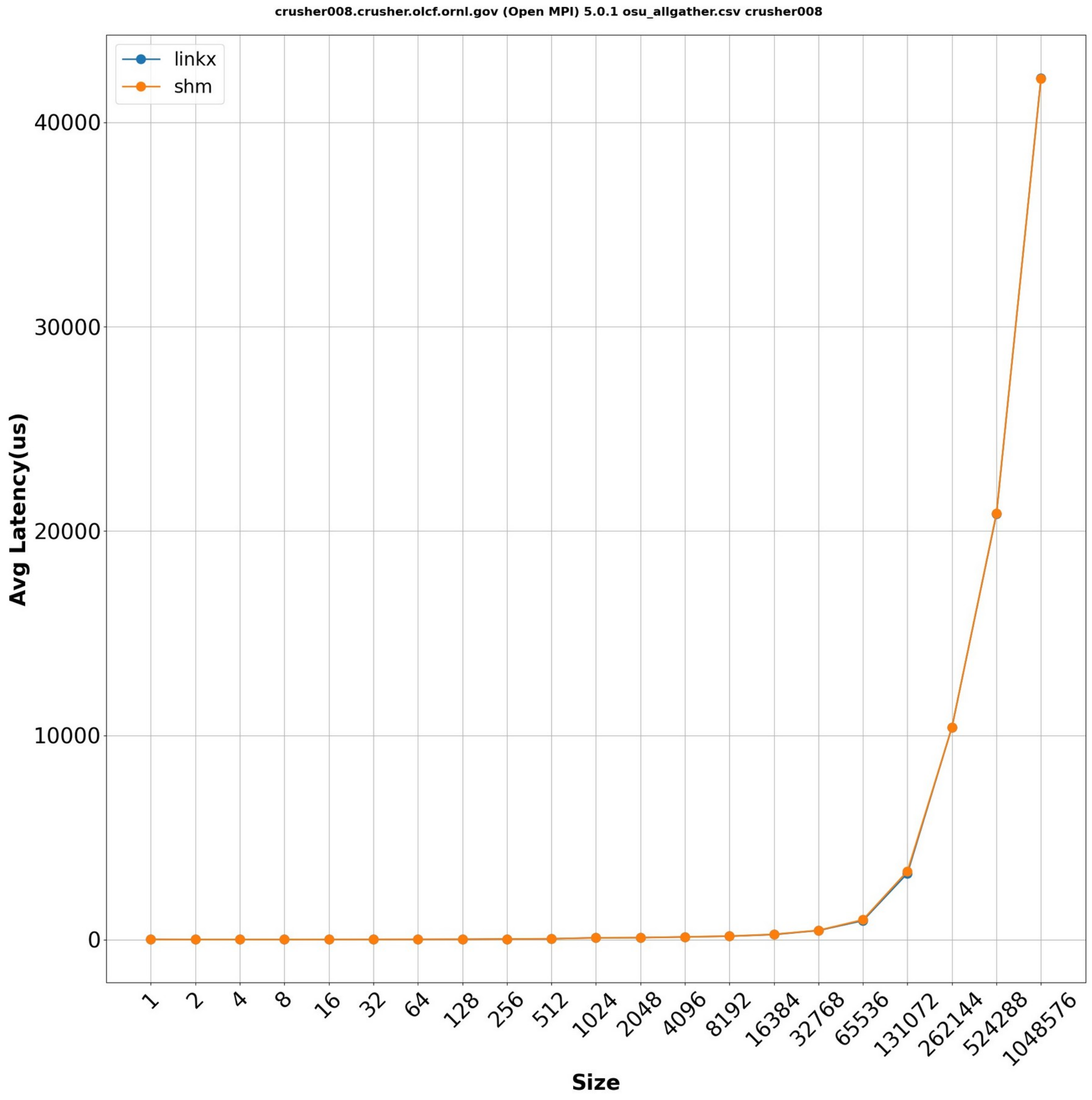
crusher008.crusher.olcf.ornl.gov (Open MPI) 5.0.1 osu_alltoall.csv crusher008

Negligible Performance Overhead.

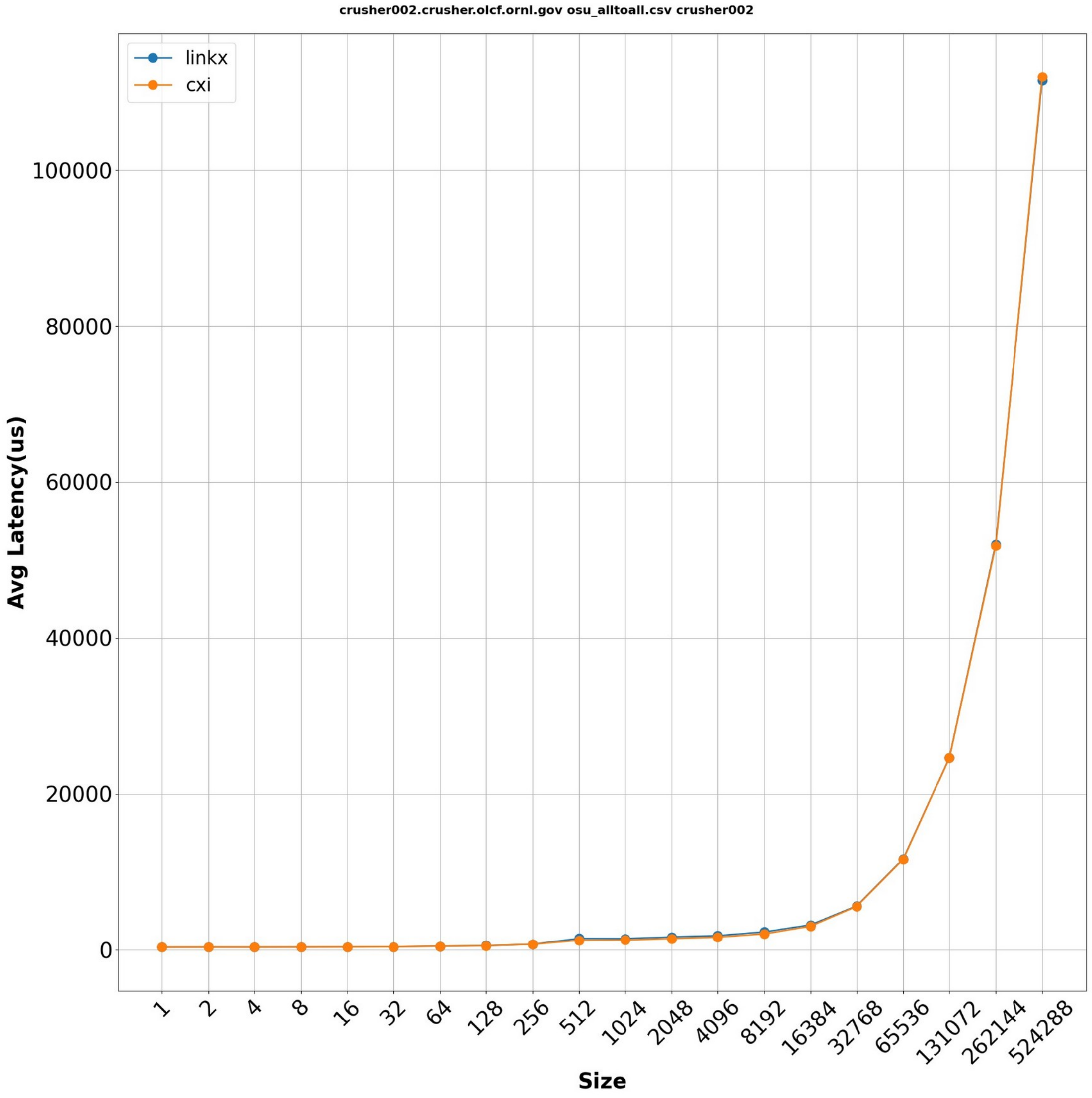


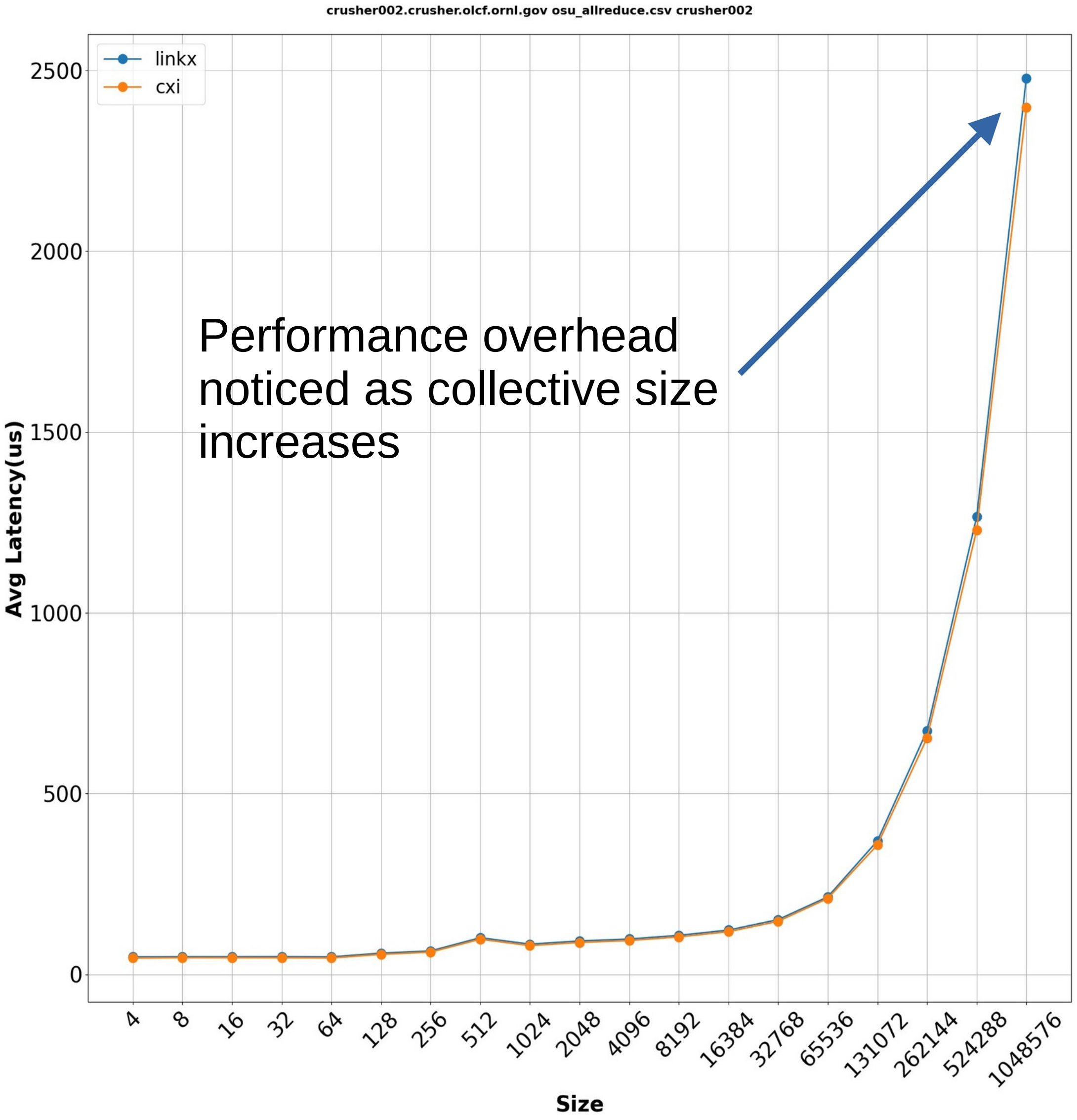
crusher008.crusher.olcf.ornl.gov (Open MPI) 5.0.1 osu_allreduce.csv crusher008

Performance: LINKx vs SHM – 56 Processes



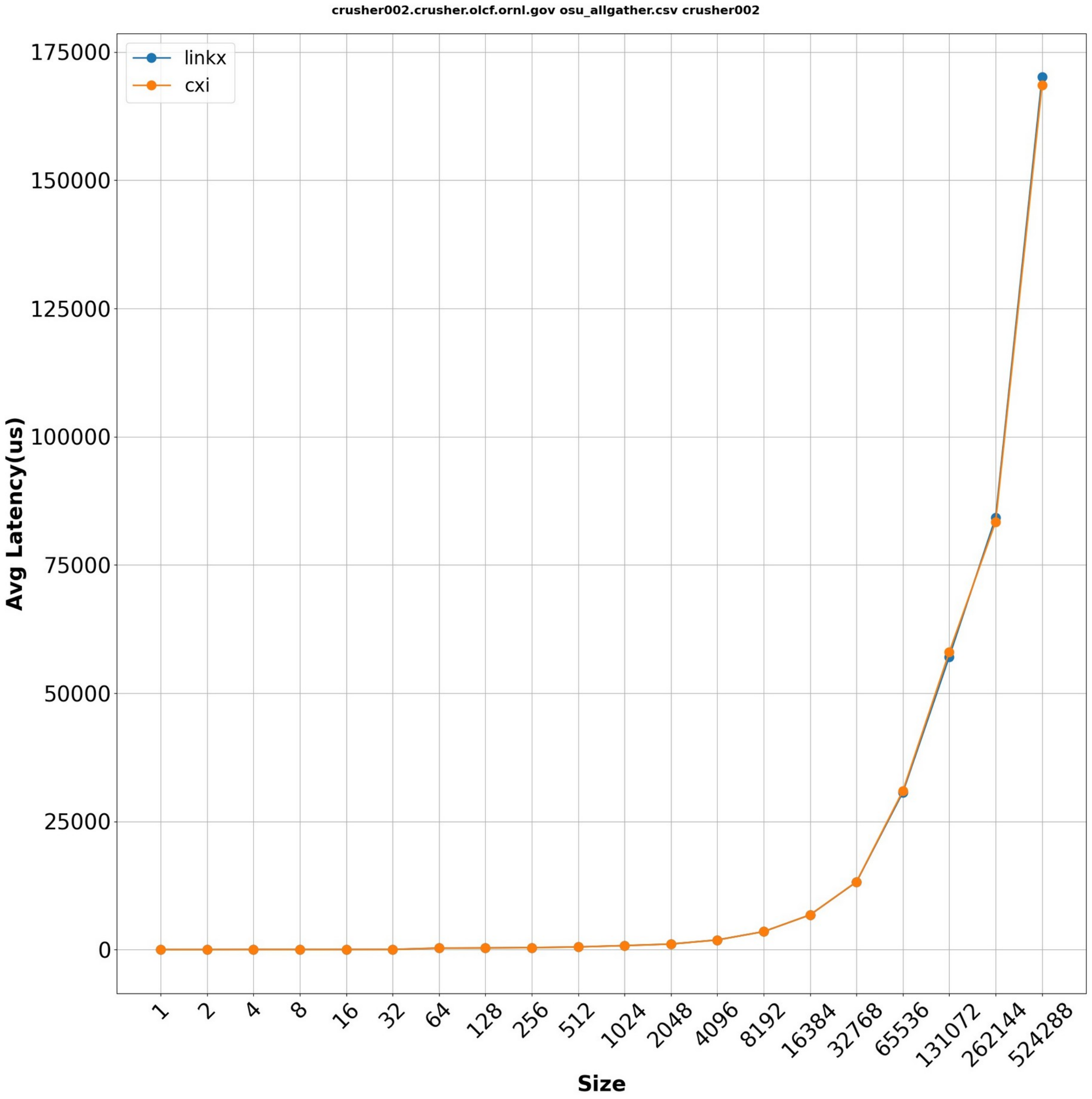
Performance: LINKx vs CXI – 1024 Processes





Performance overhead noticed as collective size increases

Performance: LINKx vs CXI – 1024 Processes



Open Questions

- Memory Registration
  - How should LINKx handle memory registration? The libfabric API assumes a single provider.
    - LINKx has no way of knowing which core provider to register memory against.
    - Currently it registers memory against all core providers.
- Hardware Offload support
  - Due to shared receive queues, HW offload, like tag matching needs to be turned off.
  - Can be turned on if application never uses FI_ADDR_UNSPEC

Future Work

- Support all libfabric APIs.
    - Currently only Tagged and RMA are supported
- Optimize LINKx to reduce the overhead as much as possible
- Better handling for memory registration
- Handle hardware offload; tag matching, stream triggering
- Support linking any number of providers
- Implement Multi-Rail

Conclusion

- **Solution is available and tested on Frontier**
- LINKx provides a portable solution which can benefit any libfabric user
- LINKx is expandable and can support different features
- More work is needed to fully optimize it
- Upstreaming work is currently underway

Questions?

OPENFABRICS ALLIANCE

2024 OFA Virtual Workshop

THANK YOU

Alexia Ingerson, Intel
Shi Jin, AWS
Amir Shehata, ORNL

intel aws OAK RIDGE National Laboratory U.S. DEPARTMENT OF ENERGY