Hewlett Packard
Enterprise

Enhancing RPC on Slingshot for Aurora's DAOS Storage System

Jerome Soumagne, Alexander Oganezov, Ian Ziemba, Steve Welch, John Carrier, Johann Lombardi, Mohamad Chaarawi, Zhen Liang, Scott Peirce, HPE
Philip Carns, Kevin Harms, Argonne National Laboratory

May 8, 2025

Outline

- Challenges and Approach
- Background and DAOS Overview
- Standard RPCs and Metadata Path
- Bulk RPCs and Bulk Data Path
- Security
- Conclusion

File Systems and Distributed Data Services

- Used to refer exclusively to monolithic file systems
  - (e.g., Lustre, GPFS)
  - Developed in kernel space and tuned for specific platforms
  - Communication components not accessible to users (e.g., LNet)

- New data service architectures
  - Take advantage of platforms capabilities (e.g., new storage technologies)
  - Tailored to application needs (software defined services)
  - Rely also heavily on network infrastructure
  - Developed in user space
    - User-space = lower latency
  - Client-server architectures

DAOS as a User-level Distributed Data Service

- User-level services cannot rely on MPI for communication
  - Ephemeral clients come and go (not a fixed size communicator)
  - Different messaging requirements
    - RMA semantics are not adapted (not a fixed / persistent memory window)
  - Instead use Remote Procedure Call (RPC)

- DAOS operates end-to-end in user space
  - Similar in essence as any user-level service
  - Hence faces the same challenges
    - Scalability, performance, security, deployment, resiliency, …

- Solving these challenges benefits broader community
  - File systems / resource managers / monitoring systems
  - Not all data services need or use HPC network (e.g., Kafka based services use TCP/IP protocol)

Aurora – DAOS Configuration

- 1024 DAOS server nodes
  - 2 Intel Xeon Gold Series (26 cores / 2 HW threads)
  - 16 Intel Optane PMem 200 series 512 GiB
  - 16 Samsung NVMe SSDs 15.3 TiB
  - 2 Slingshot 200 Gb NICs

Compute Node

2 Intel Xeon CPU Max Series processors; 6 Intel Data Center Max GPUs; Unified Memory Architecture; 8 fabric endpoints; RAMBO

GPU Architecture

Intel Data Center GPU Max Series; Tile-based, chiplets, HBM stack, Foveros 3D integration

On-Node Interconnect

CPU-GPU: PCIe
GPU-GPU: Xe Link

Aggregate System Memory

20.4 PB

High-Performance Storage

230 PB, 31 TB/s (DAOS)

System Interconnect

HPE Slingshot 11; Dragonfly topology with adaptive routing

Network Switch

25.6 Tb/s per switch, from 64–200 Gbs ports (25 GB/s per direction)

Programming Environment

Intel oneAPI, MPI, OpenMP, C/C++, Fortran, SYCL/DPC++

Software Stack

HPE Cray EX software stack +Intel Enhancements + Data and Learning

Platform

HPE Cray EX supercomputer

System Performance

Exascale

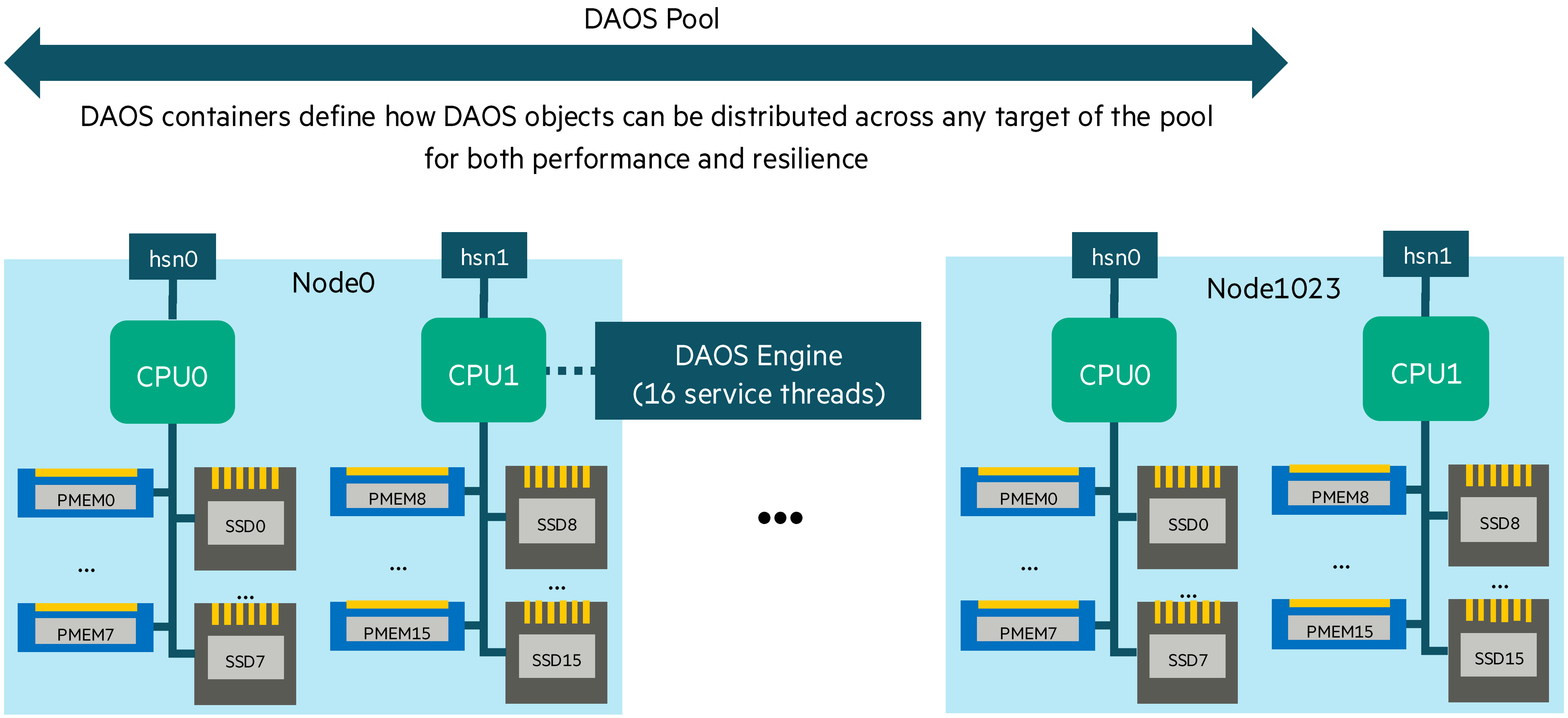
System Size

10,624 nodes

Source: https://www.alcf.anl.gov/sites/default/files/2024-07/Aurora_FactSheet_2024.pdf

Aurora – DAOS Configuration

Any client process may communicate with any target service thread!

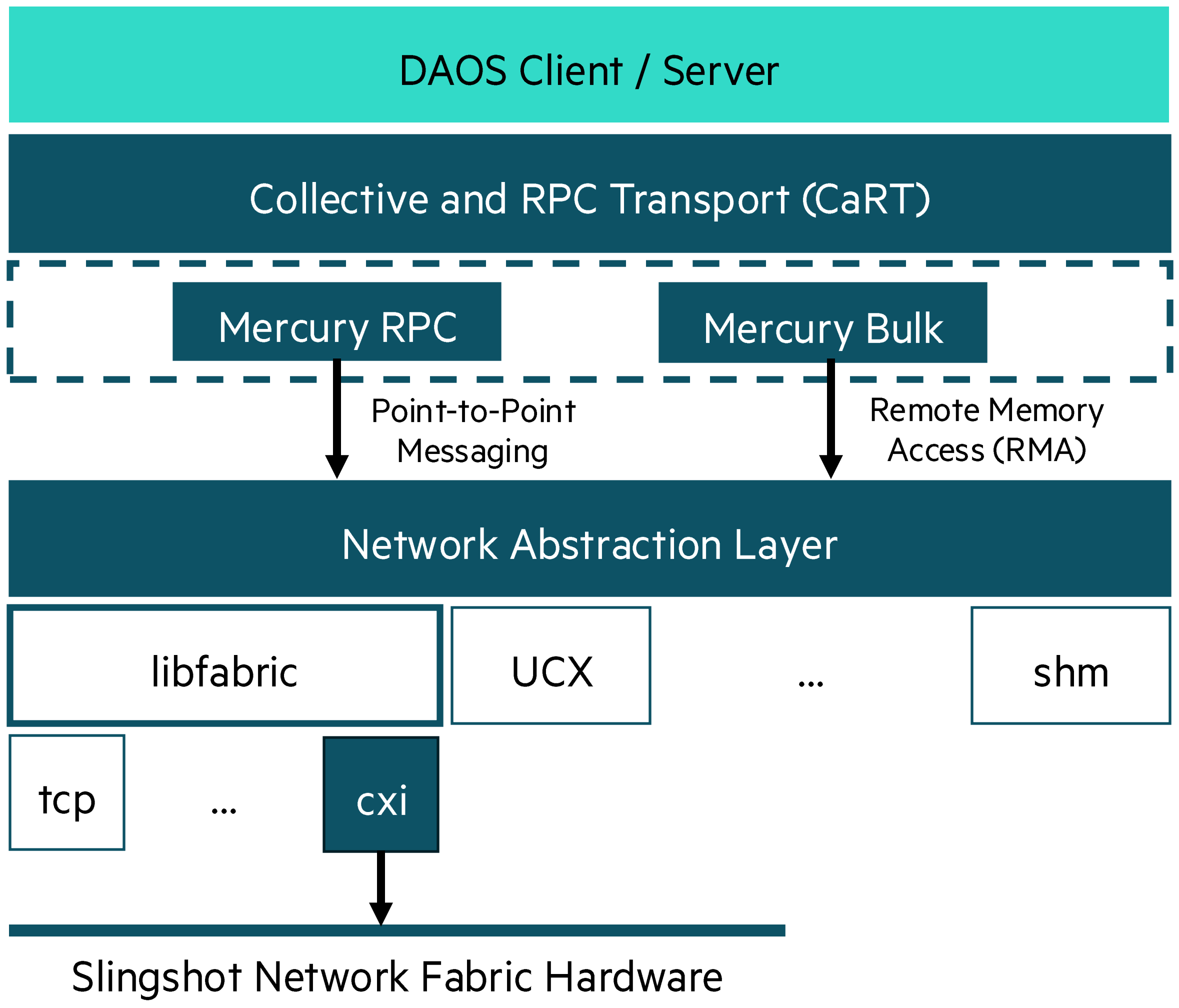


Overview of DAOS Components

- DAOS relies on 3 main network components:
  - CaRT (RPC and group membership)
  - Mercury RPC and Network Abstraction Layer
  - Libfabric and cxi provider

- RPC = request / response with header + payload
- RPC features:
  - Connectionless
  - Non-blocking

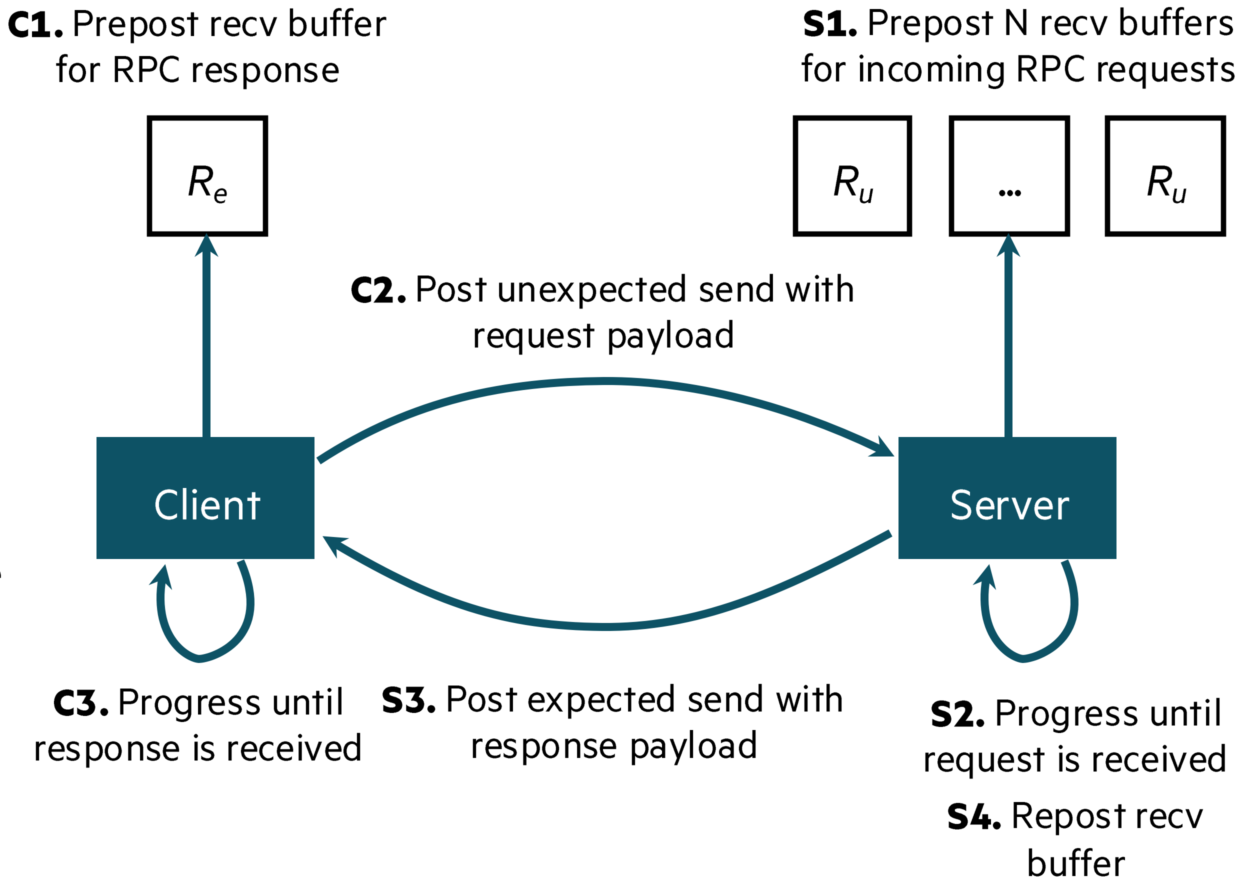
- Two types of RPCs:
  - Standard RPCs (Metadata and control messages)
    – Point-to-point messaging
  - Bulk RPCs (Bulk data)
    – Standard RPC + RMA


DAOS Client / Server
Collective and RPC Transport (CaRT)
Mercury RPC
Mercury Bulk
Point-to-Point Messaging
Remote Memory Access (RMA)
Network Abstraction Layer
libfabric
UCX
...
shm
tcp
...
cxi
Slingshot Network Fabric Hardware

Metadata Path – Standard Receives

- Most natural way of implementing RPC
  - Pre-post buffers
  - 1 buffer for each RPC request

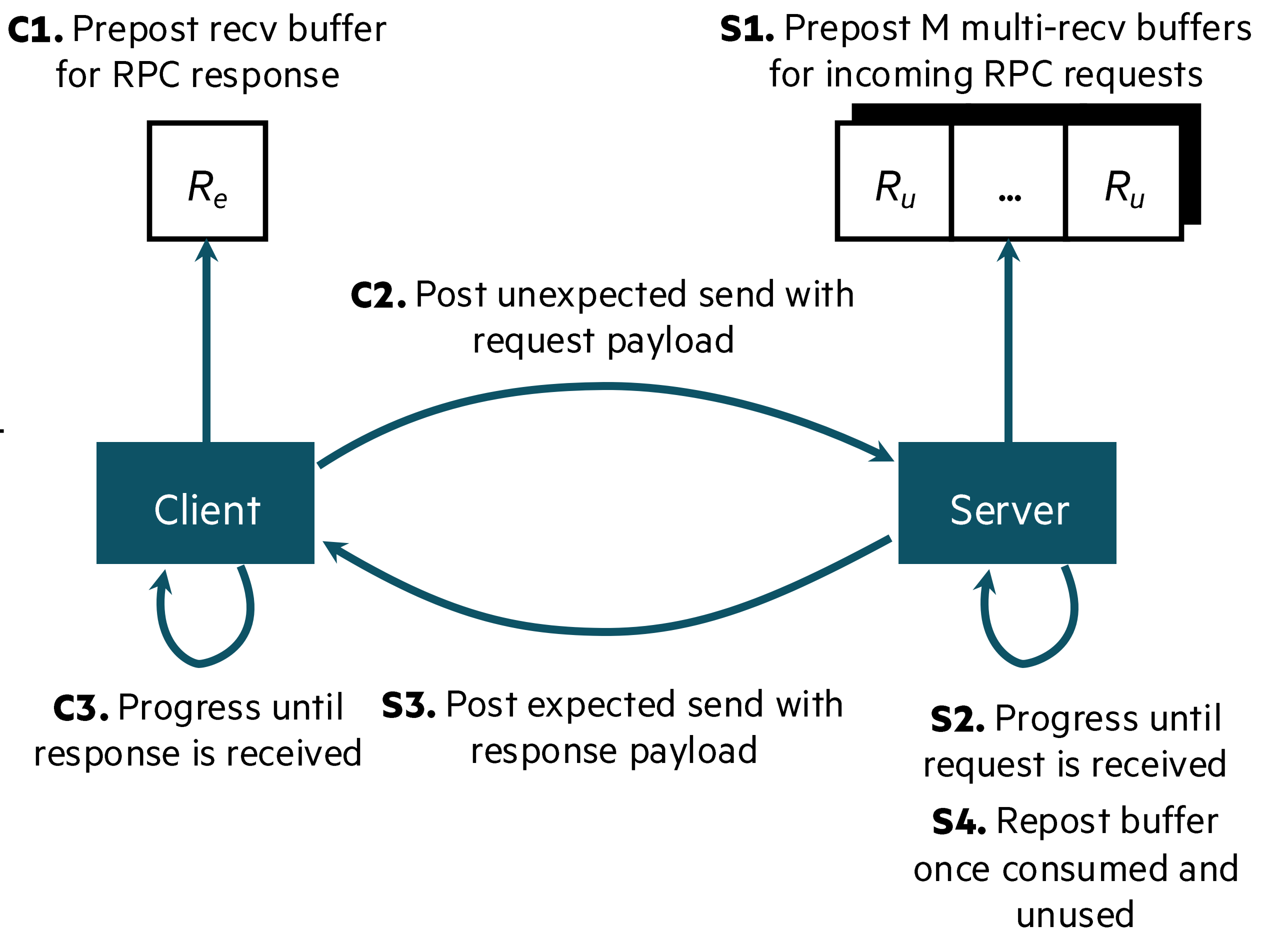
- Constraints
  - Buffer is consumed regardless of message size
  - Must always have pre-posted N buffers with N > number of client processes
  - For each buffer posted we consume hardware resources → easy to reach hardware limits
    - Slingshot: **`RX_MATCH_MODE`** from hardware to software increases latency


C1. Prepost recv buffer for RPC response
S1. Prepost N recv buffers for incoming RPC requests
R_e
R_u
...
R_u
C2. Post unexpected send with request payload
Client
Server
C3. Progress until response is received
S3. Post expected send with response payload
S2. Progress until request is received
S4. Repost recv buffer

Metadata Path – Multi-Receives

- Multi-receive
  - Feature exposed by libfabric
  - Post a single buffer that is used until consumed
    - Most advantageous when message size is small
  - Drastically reduces the amount of HW resources used
  - Reduces the number of recv calls
    - Number of HW commands reduced
  - Much easier to remain in scale with number of client processes

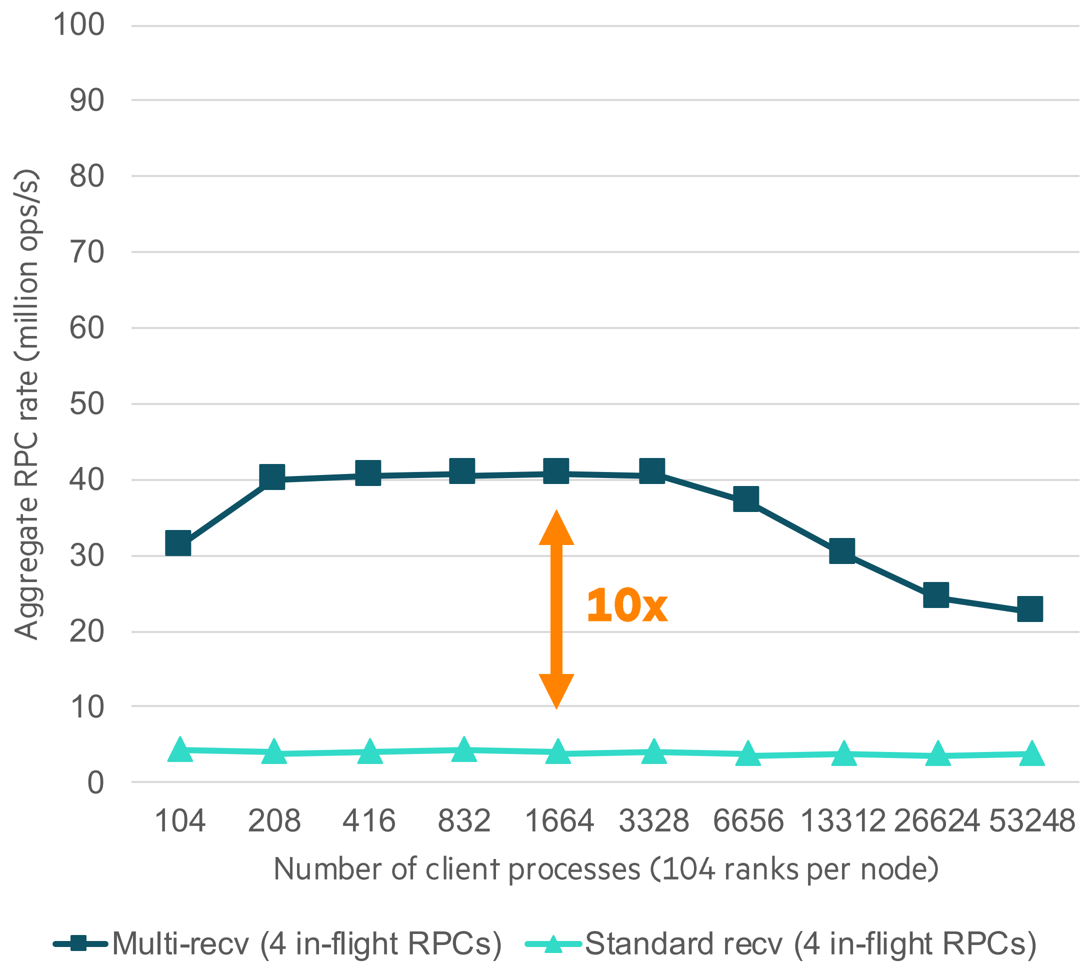
- Constraint:
  - Buffer can only be re-posted when no longer in use
  - Must implement a message copy fallback if all buffers are full



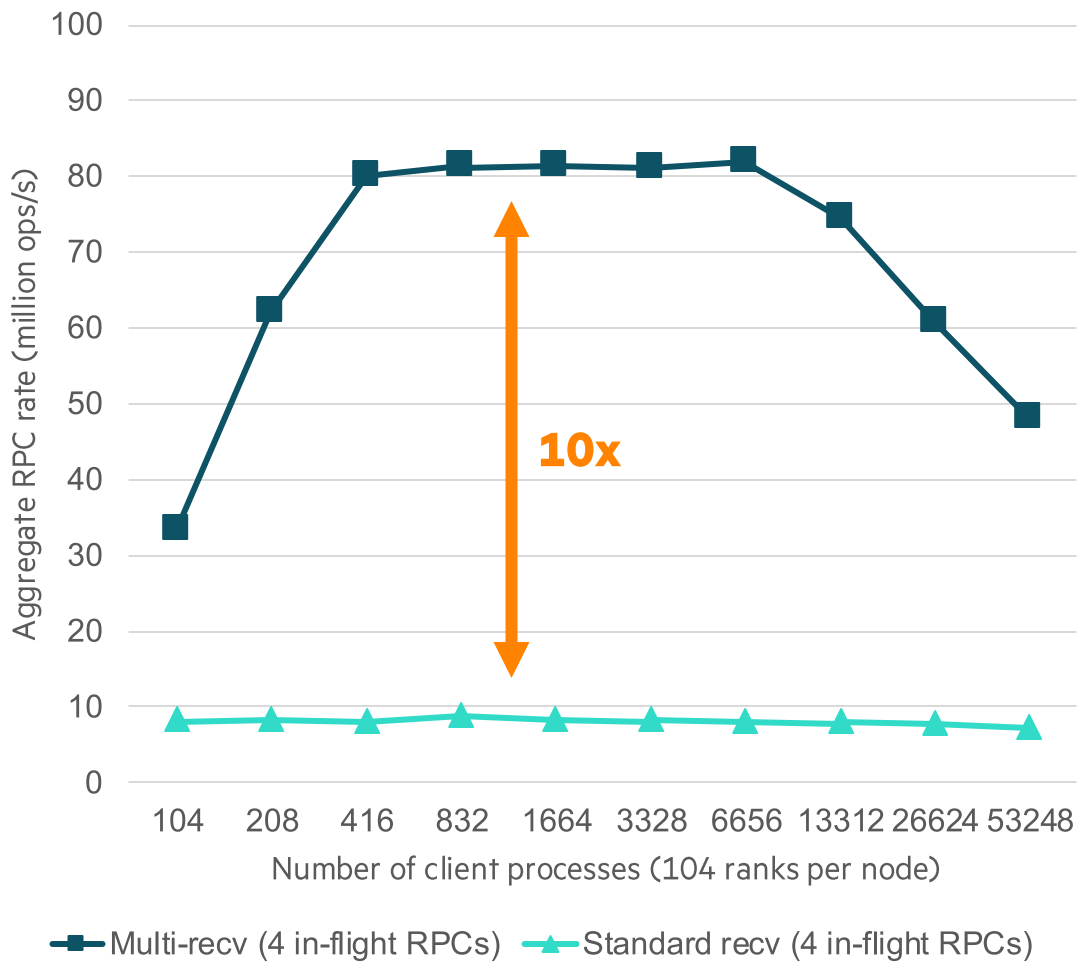
C1. Prepost recv buffer for RPC response
S1. Prepost M multi-recv buffers for incoming RPC requests
R_e
R_u
...
R_u
C2. Post unexpected send with request payload
Client
Server
C3. Progress until response is received
S3. Post expected send with response payload
S2. Progress until request is received
S4. Repost buffer once consumed and unused

Metadata Path – Performance Evaluation

1 server node (2 x 16 threads per node)



2 server nodes (2 x 16 threads per node)

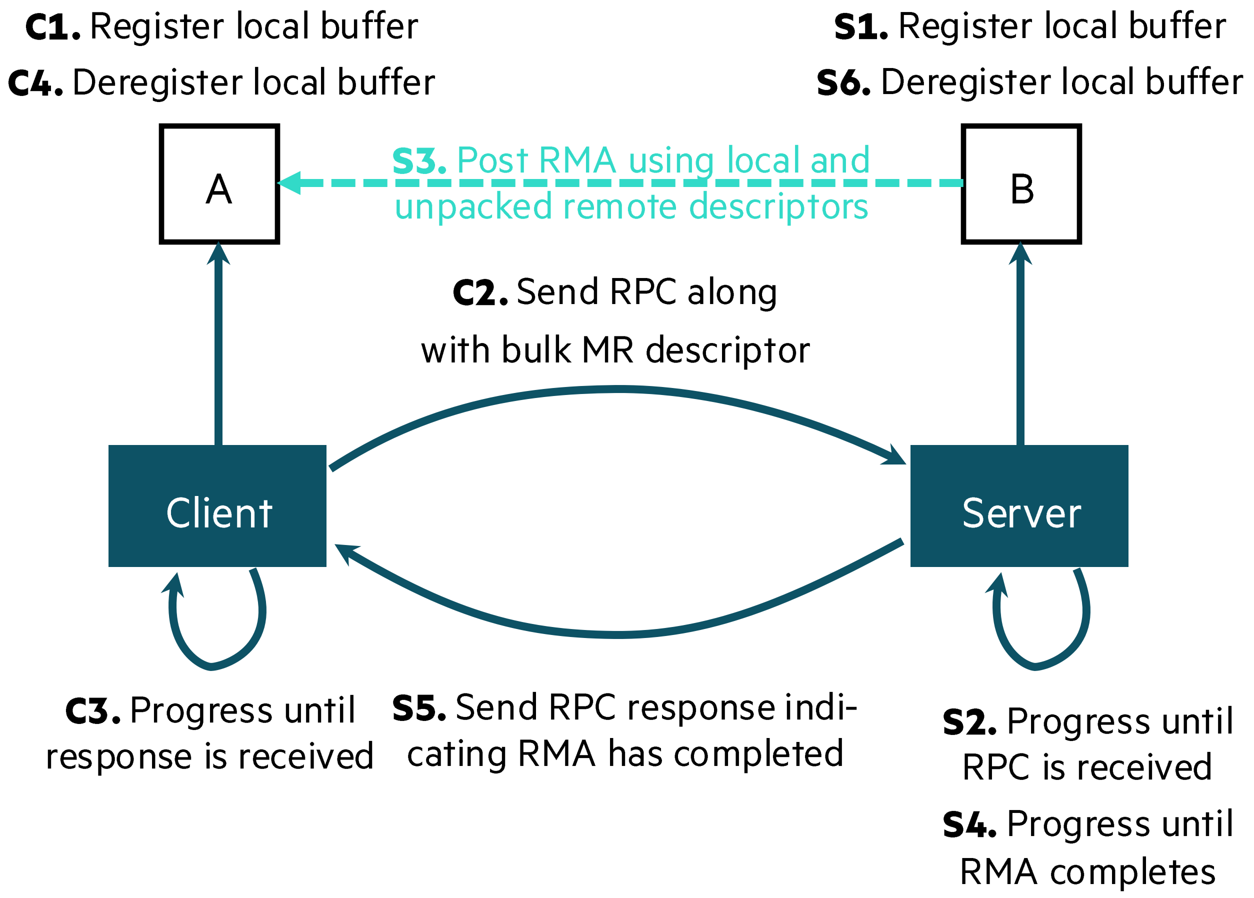


Bulk Data Path – Bulk RPCs

- Used for the transfer of large data that does not fit into regular message
- Uses RDMA for zero-copy transfers

- RDMA requires
  - Registration of buffer / memory regions (MR)
  - Exchange of MR descriptors
  - Client memory to remain valid while RMA is performed

- Typically, applications pass own buffers through DAOS APIs
  - Have to register / deregister buffers in a hot code path, expensive!


C1. Register local buffer
C4. Deregister local buffer
S1. Register local buffer
S6. Deregister local buffer
A
B
S3. Post RMA using local and unpacked remote descriptors
C2. Send RPC along with bulk MR descriptor
Client
Server
C3. Progress until response is received
S5. Send RPC response indicating RMA has completed
S2. Progress until RPC is received
S4. Progress until RMA completes

Bulk Data Path – MR Caching

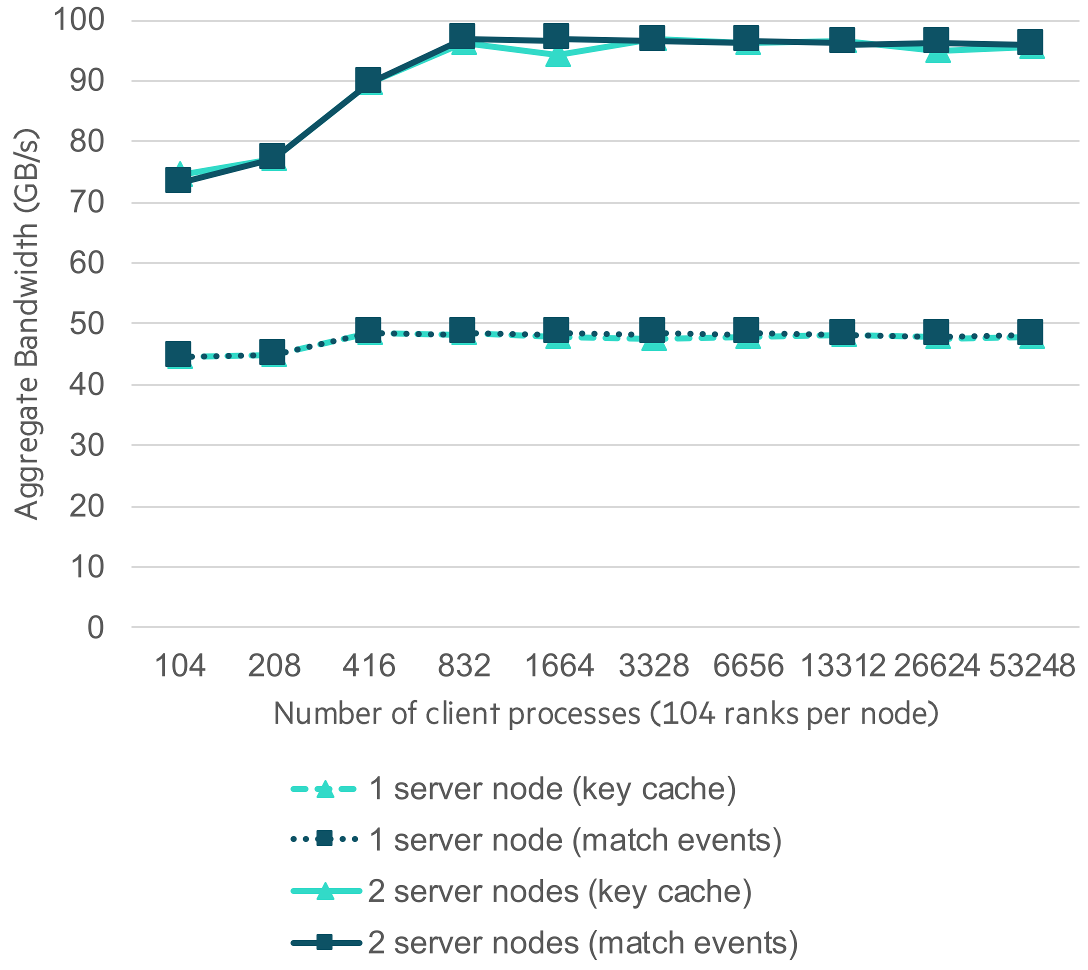
- MR registration / deregistration in libfabric is performed by:
  - `fi_mr_reg()` / `fi_mr_bind()` / `fi_mr_enable()`
  - `fi_close()`
- Slingshot provides two types of MRs:
  - **Optimized MRs**: best suited for persistent memory windows
    - Allocation / deallocation require kernel calls
    - Limited in count (0-99)
  - **Standard MRs**: lower operation rate
    - Deregistration more expensive and also require kernel calls
    - Not limited in count (up to HW limits)

- Solution: use MR caching (`PROV_KEY_CACHE`)
  - Drawback: lets the MR exposed on the network
    - Even after a call to `fi_close()`
  - Problem: a client timing out and cancelling an RPC could still see its memory being accessed by a DAOS server
    - Memory can be re-used etc
- New solution: match events (`MR_MATCH_EVENTS`)
  - Track number of operations pending and completed against a remote MR to skip kernel calls
  - Specific to Slingshot

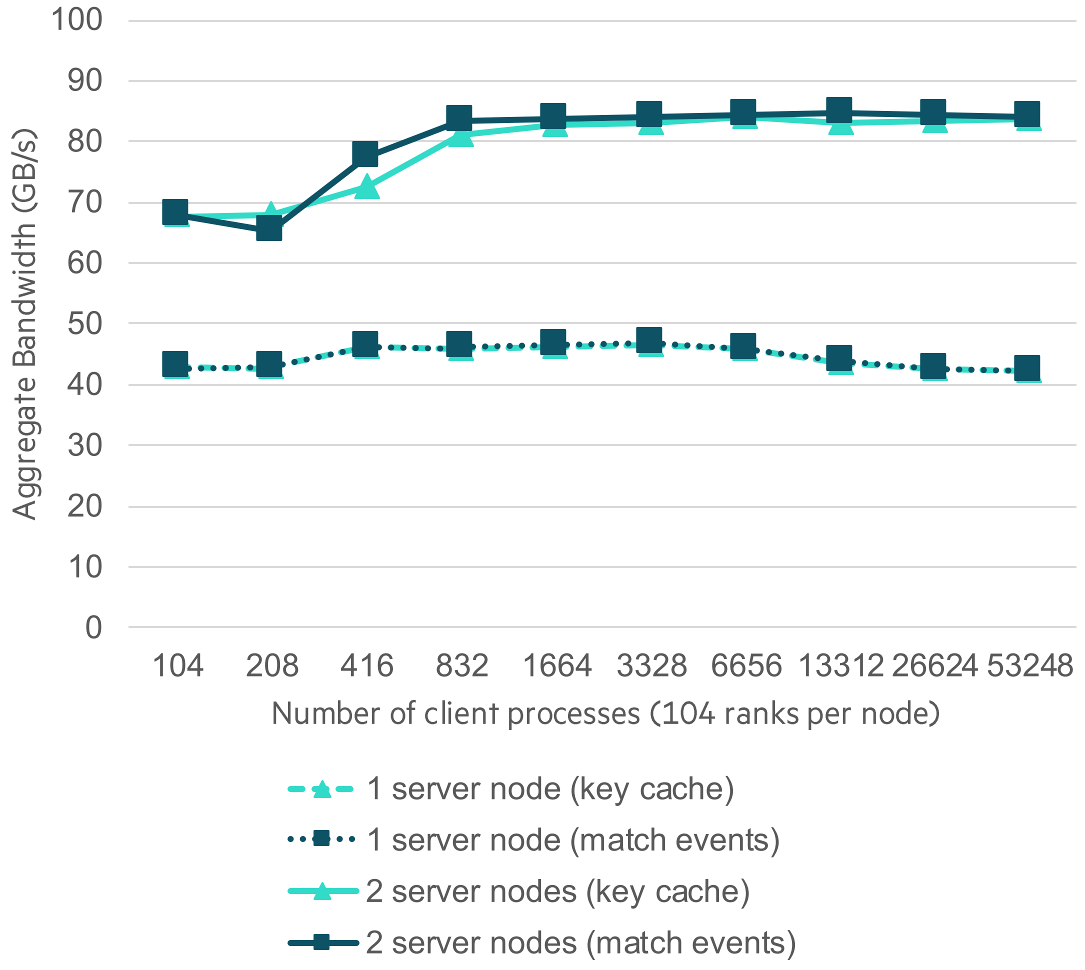
- Other solution considered: keep caching but combine it with transfer deadlines

Bulk Data Path – Performance Evaluation

Client bulk RPC write / Server RMA read



Client bulk RPC read / Server RMA write

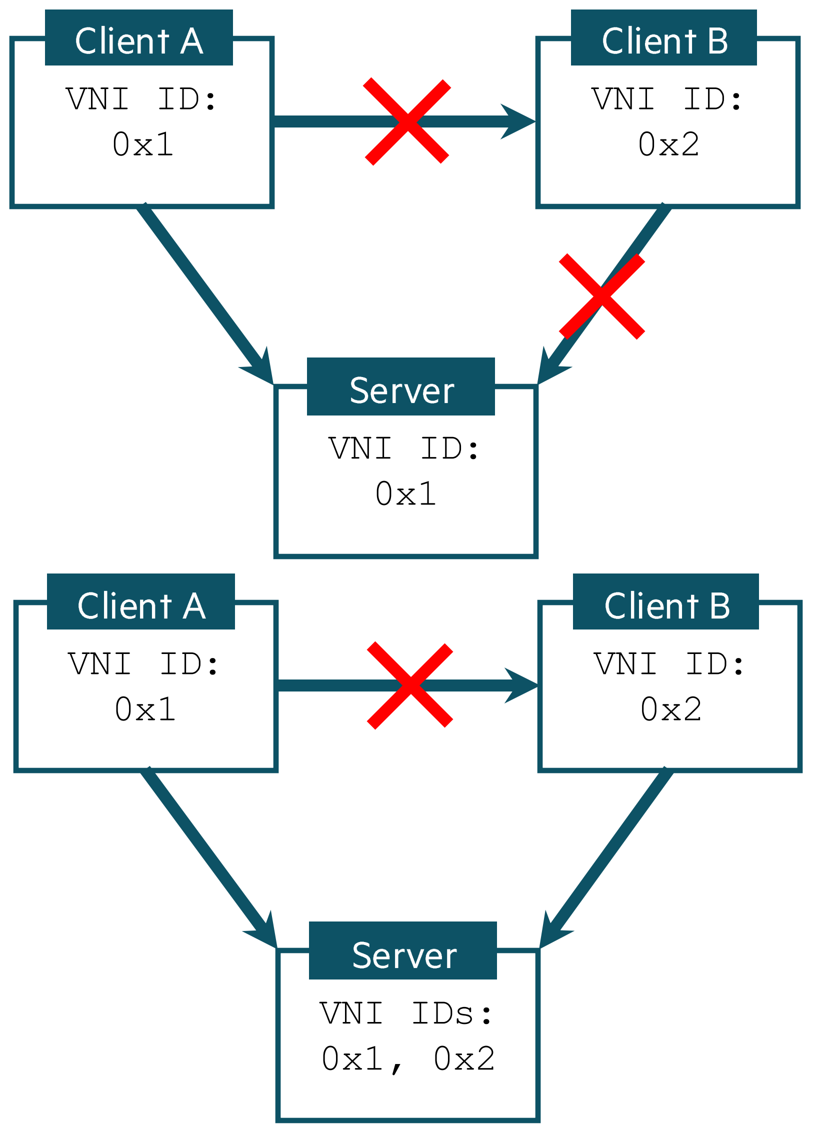


Security – VNI Restrictions

- Slingshot needs 3 components to communicate:
  - NIC address / Port ID (PID) / Virtual Network ID (VNI)
  - VNI = protection key used to provide isolation between applications

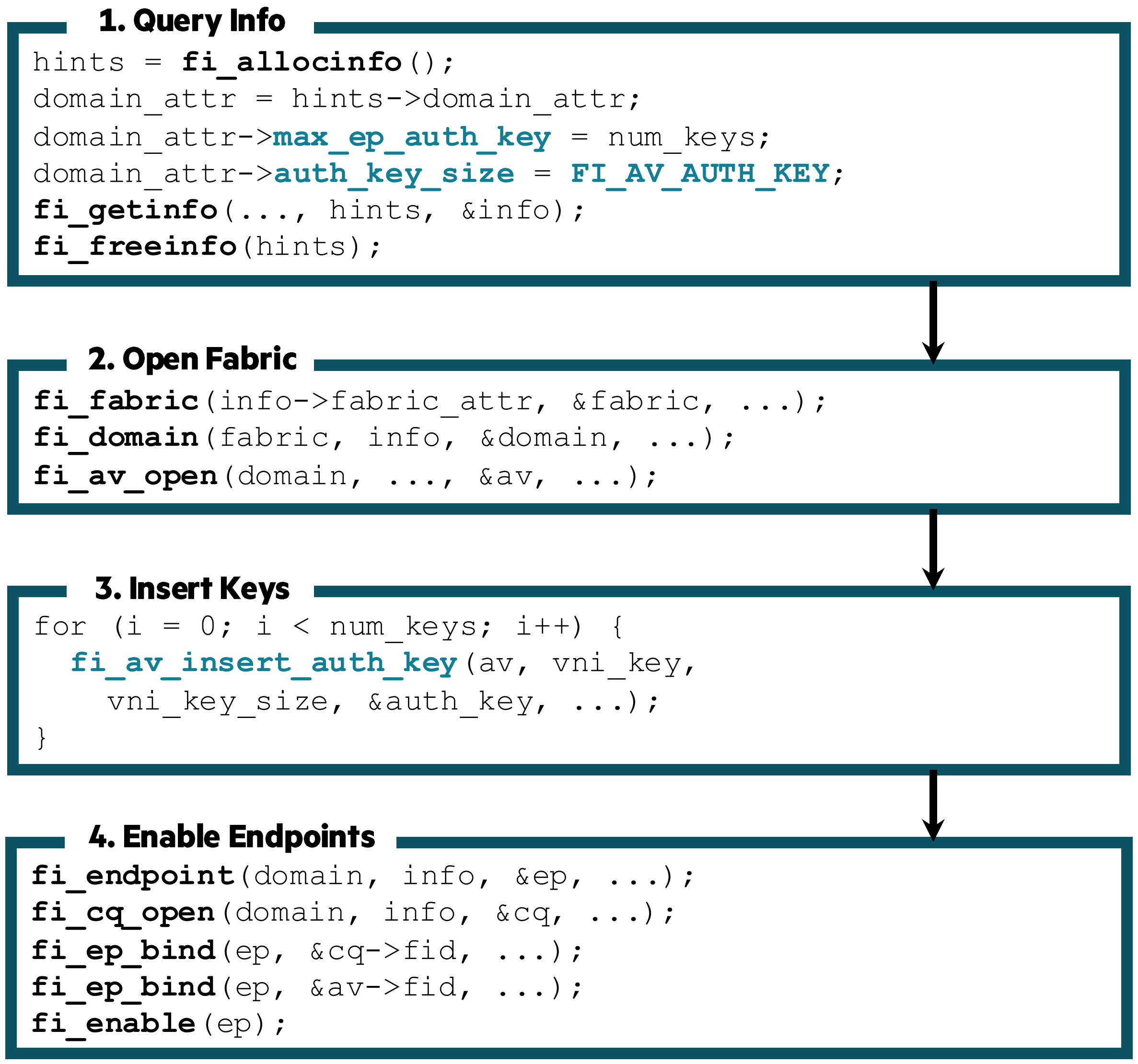
- Issue in client-server models:
  - A server needs to be able to communicate with different client applications
  - “Currently” only a single VNI per endpoint
  - DAOS has its own form of authentication and authorization mechanism
    – Not sufficient to prevent RDMA access

- Solution: augment libfabric authorization key support
  - NB. Changes are only needed on server



Security – Libfabric Authorization Keys

- Authorization key = CXI service ID + VNI
- New feature added in libfabric 1.20
  - FI_AV_AUTH_KEY
  - Define the max number of auth keys
- Libfabric address vector (AV) augmented
  - Can insert multiple authorization keys
- All keys must be inserted prior to endpoint enablement



1. Query Info

```
hints = fi_allocinfo();
domain_attr = hints->domain_attr;
domain_attr->max_ep_auth_key = num_keys;
domain_attr->auth_key_size = FI_AV_AUTH_KEY;
fi_getinfo(..., hints, &info);
fi_freeinfo(hints);
```

2. Open Fabric

```
fi_fabric(info->fabric_attr, &fabric, ...);
fi_domain(fabric, info, &domain, ...);
fi_av_open(domain, ..., &av, ...);
```

3. Insert Keys

```
for (i = 0; i < num_keys; i++) {
  fi_av_insert_auth_key(av, vni_key,
    vni_key_size, &auth_key, ...);
}
```

4. Enable Endpoints

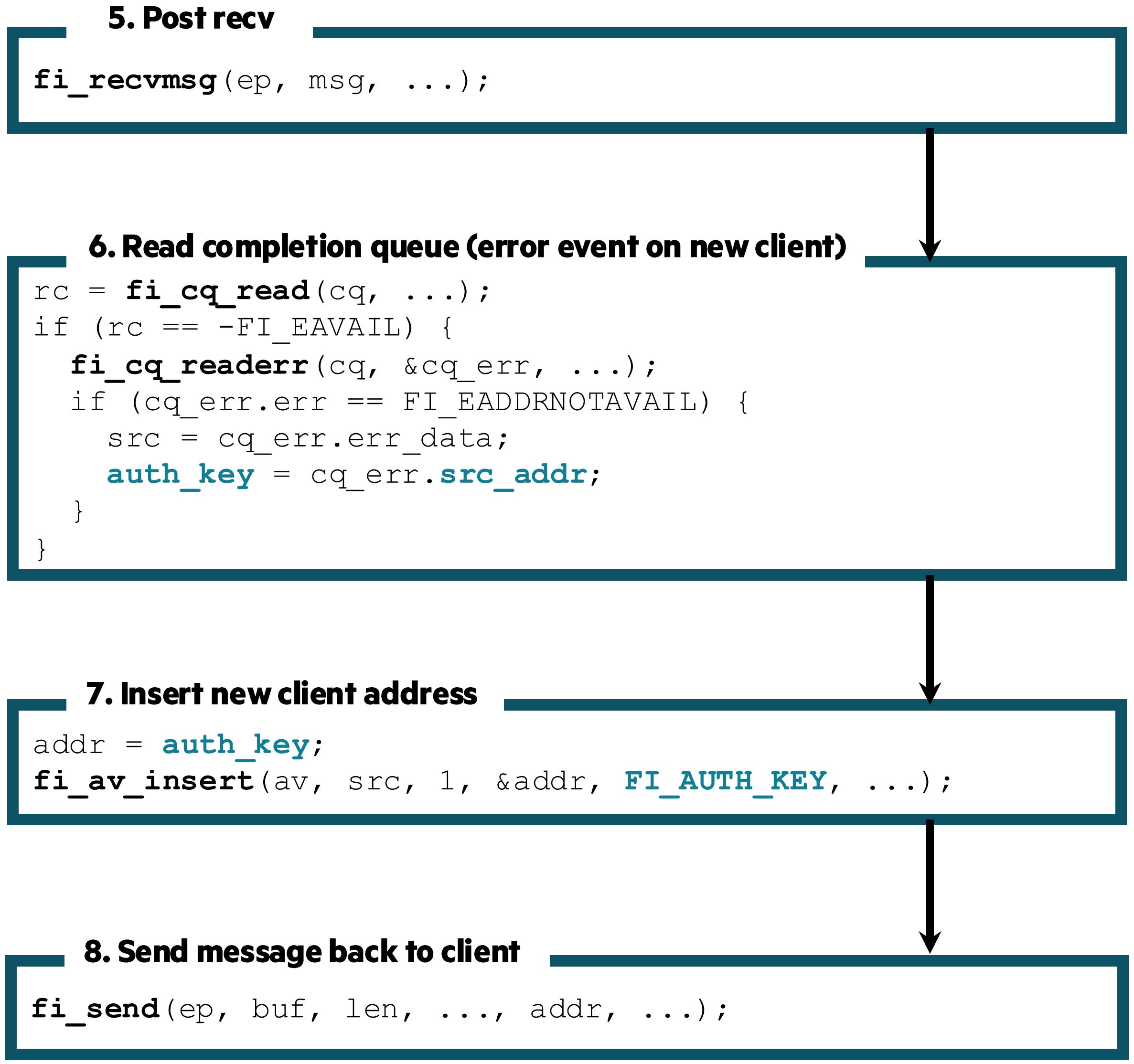
```
fi_endpoint(domain, info, &ep, ...);
fi_cq_open(domain, info, &cq, ...);
fi_ep_bind(ep, &cq->fid, ...);
fi_ep_bind(ep, &av->fid, ...);
fi_enable(ep);
```

Security – Libfabric Authorization Keys

- Receive error event when unknown client sends first message to server
  - Previously inserted key is then returned to us
  - Requires provider to support **`FI_SOURCE`** and **`FI_SOURCE_ERR`** capabilities of libfabric

- Libfabric requires peers addresses to be inserted before communication can be made
  - Added new FI_AUTH_KEY flag to associate auth_key with peer address

- We can then send a message back to client!
  - (And even RDMA to it)



5. Post recv

```
fi_recvmsg(ep, msg, ...);
```

6. Read completion queue (error event on new client)

```
rc = fi_cq_read(cq, ...);
if (rc == -FI_EAVAIL) {
  fi_cq_readerr(cq, &cq_err, ...);
  if (cq_err.err == FI_EADDRNOTAVAIL) {
    src = cq_err.err_data;
    auth_key = cq_err.src_addr;
  }
}
```

7. Insert new client address

```
addr = auth_key;
fi_av_insert(av, src, 1, &addr, FI_AUTH_KEY, ...);
```

8. Send message back to client

```
fi_send(ep, buf, len, ..., addr, ...);
```

Conclusion and Future Work

- Designing and implementing distributed data services at user-level presents unique challenges
  - Challenges are not solved by MPI implementations
    - Unique to client-server models
  - Requires fabric software to be designed with that level of support
  - User-level means that it benefits broader community

- Contributions:
  - Metadata Path augmented by making use of multi-recv
    - 10x improvement in RPC rate
  - Registration overheads
    - Multiple techniques can be used but MR caching leaves memory exposed
  - Security can be maintained by using multiple authorizations keys on server

- Future Work:
  - Remove use of tag messages
  - Make use of a new CXI protocol, Receiver-Not-Ready (RNR) protocol
    - Best for flow control

Thank you

jerome.soumagne@hpe.com