Screencast: OMPI OpenFabrics Protocols (v1.2 series)

Jeff Squyres

May 2008

Short Messages

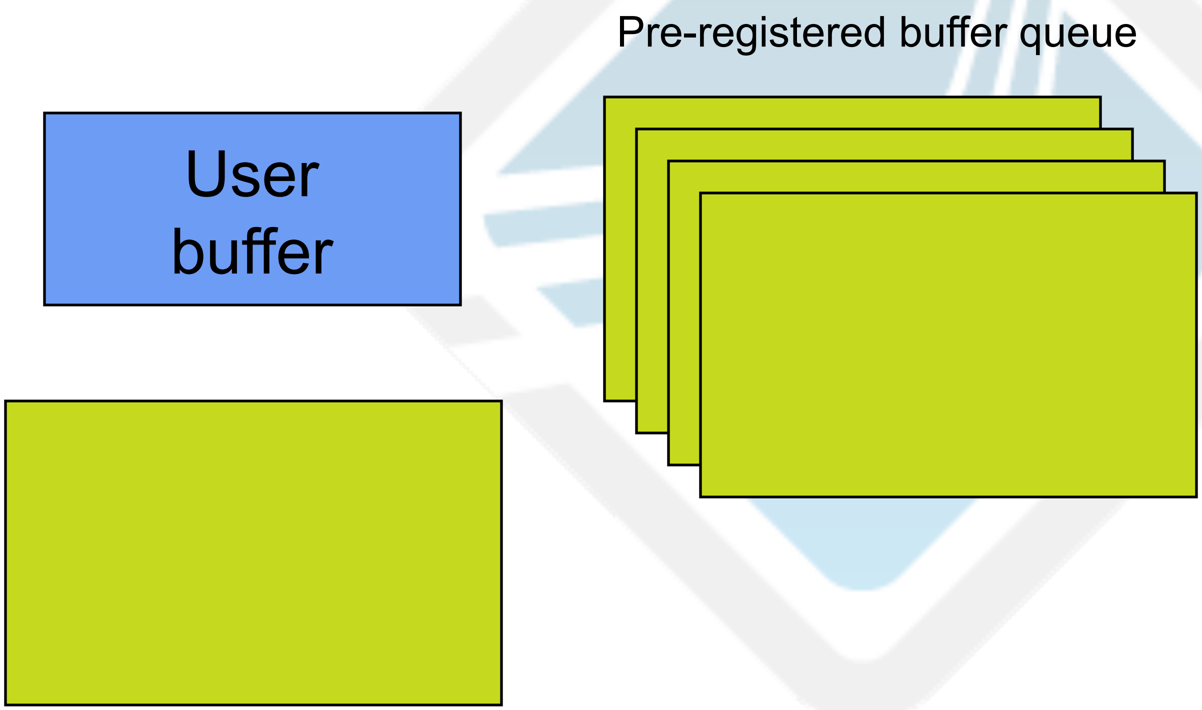
- For short messages
  - memcpy() into / out of pre-registered buffers
  - “short” = “memcpy cost does not matter”
- Once copied in, do one of two things:
  - Use “eager” RDMA
  - Use send / receive semantics

Short Message Protocol

- **`MPI_SEND(short_message, …)`**

Short Message Protocol

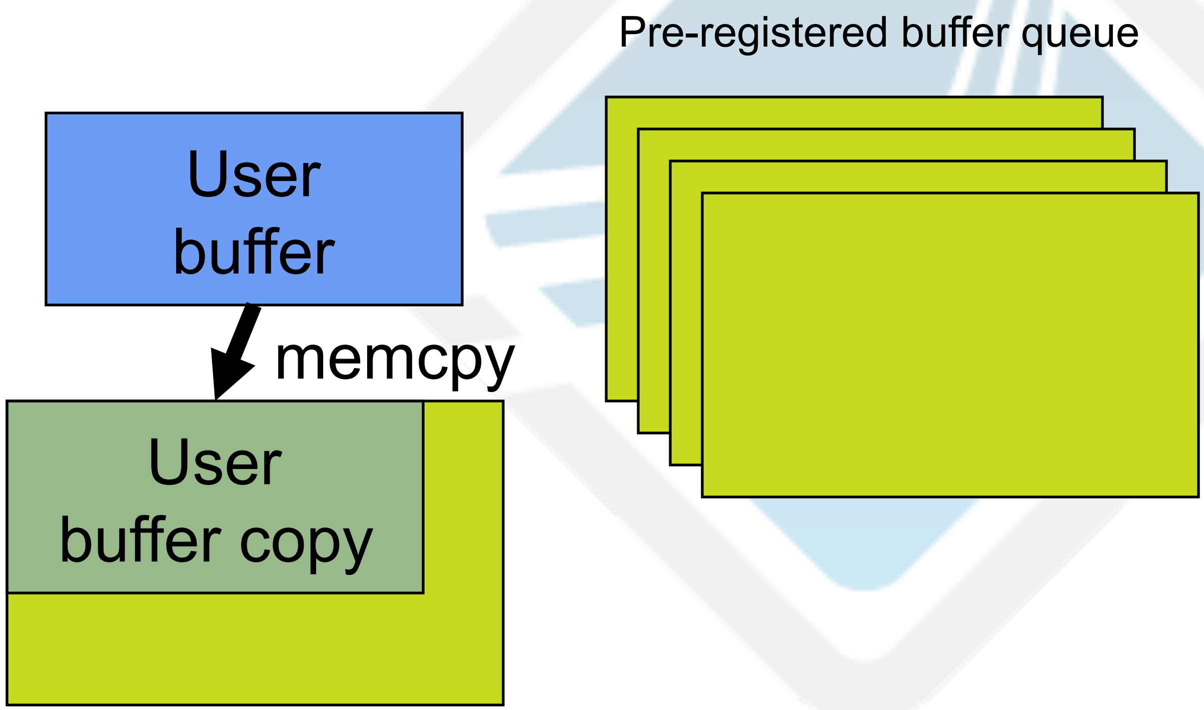
- **`MPI_SEND(short_message, …)`**



Dequeue a pre-registered buffer

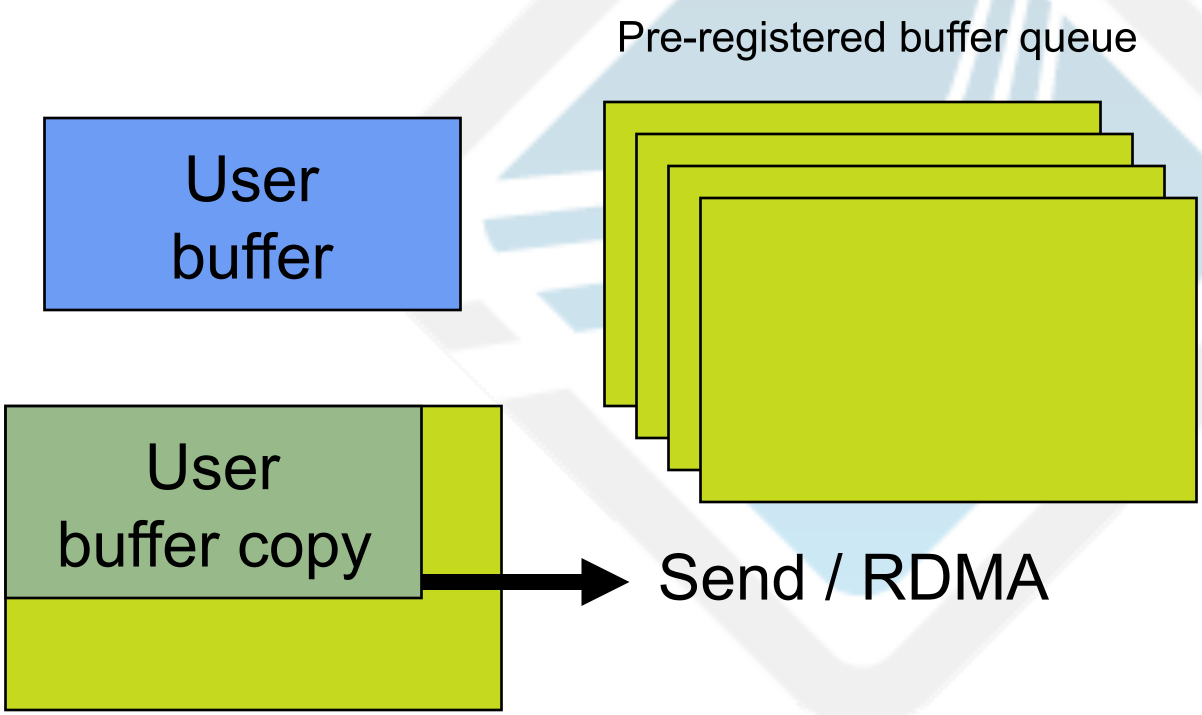
Short Message Protocol

- **`MPI_SEND(short_message, …)`**



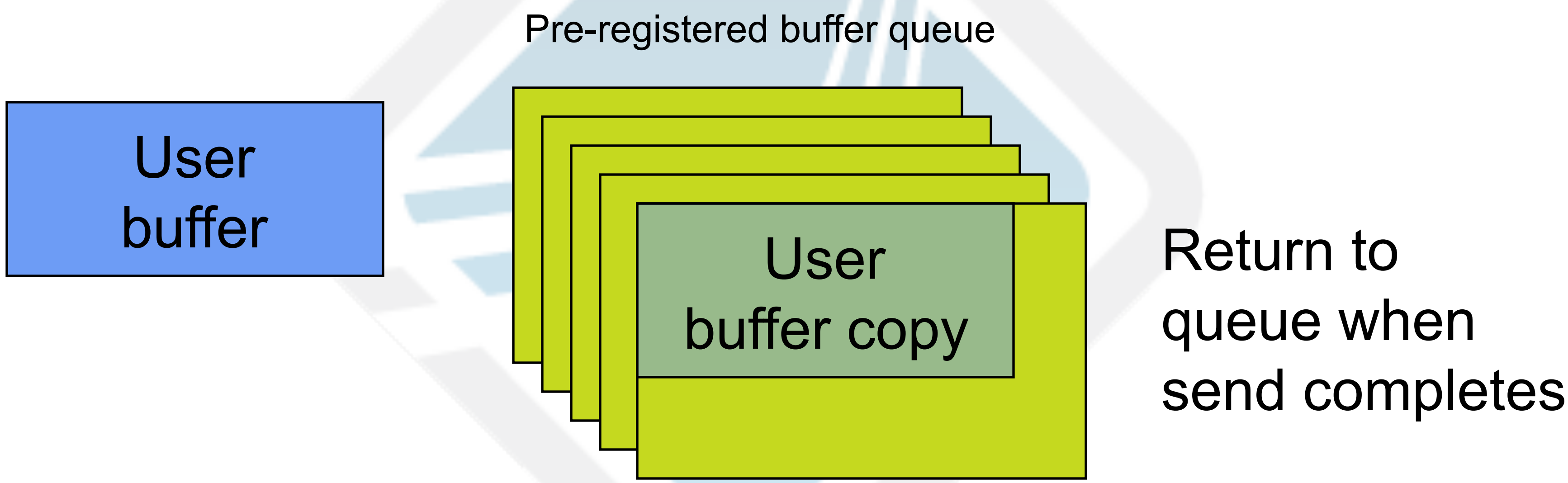
Short Message Protocol

- `MPI_SEND(short_message, …)`



Short Message Protocol

- **`MPI_SEND(short_message, …)`**

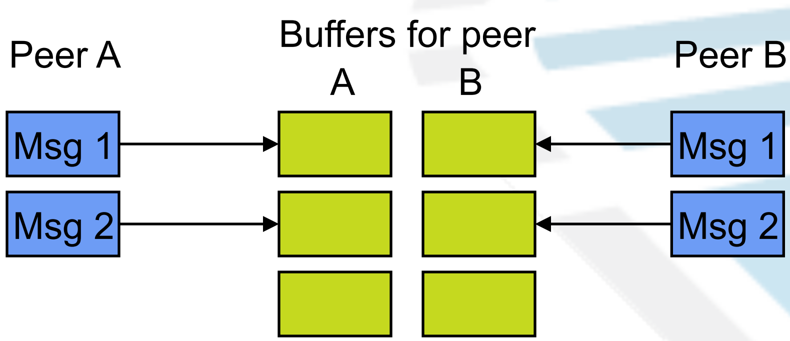


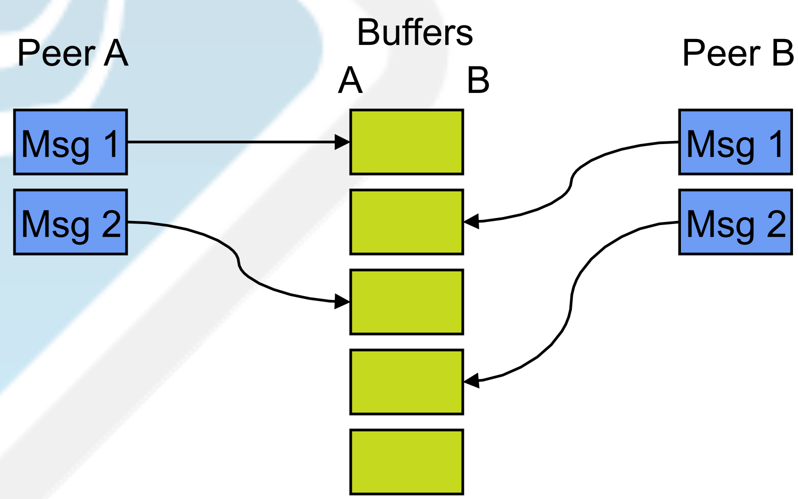
Short: RDMA vs. Send

- RDMA semantics
  - Sender specifies [remote] target buffer address
  - Requires N pre-registered buffers **for each peer**
  - Quickly becomes non-scalable
- Send / receive semantics
  - Receiver specifies target buffer address
  - Can use common pool of pre-registered buffers

Short: RDMA vs. Send

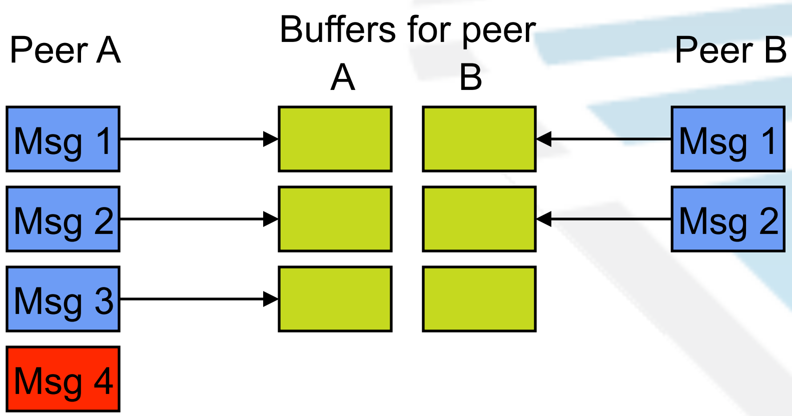
- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software

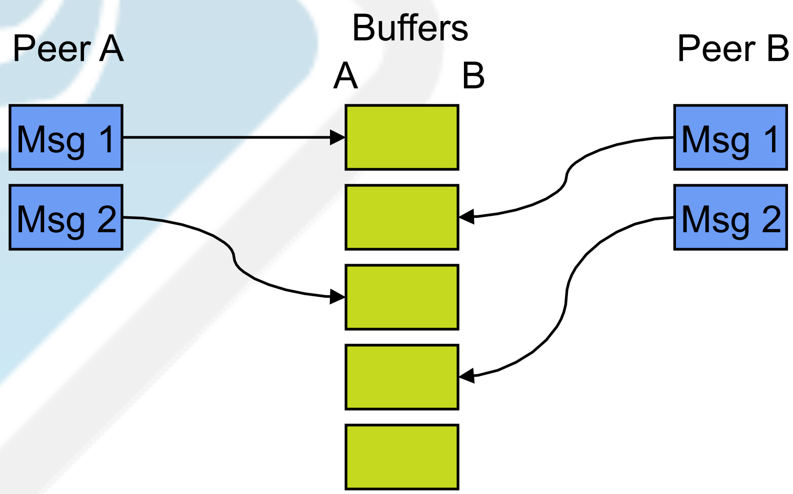




Short: RDMA vs. Send

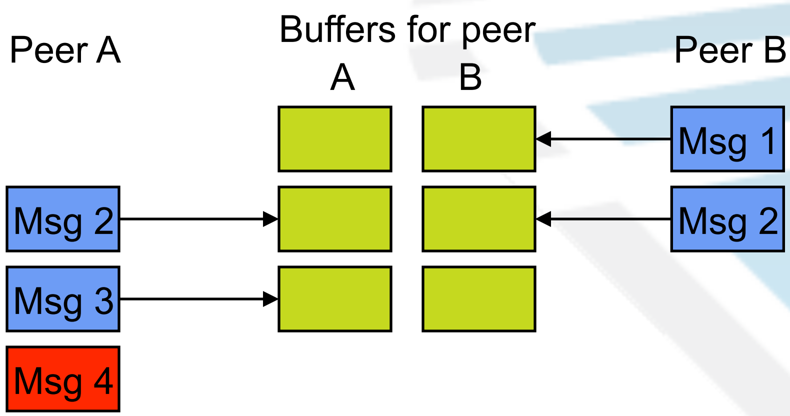
- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software

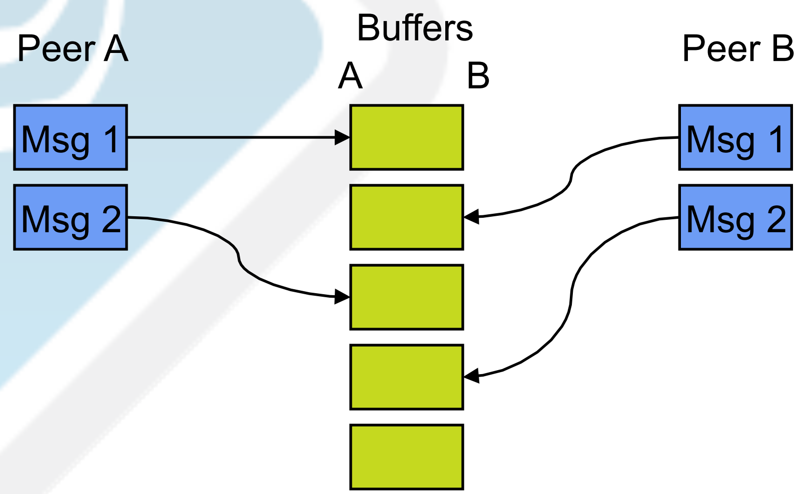




Short: RDMA vs. Send

- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software

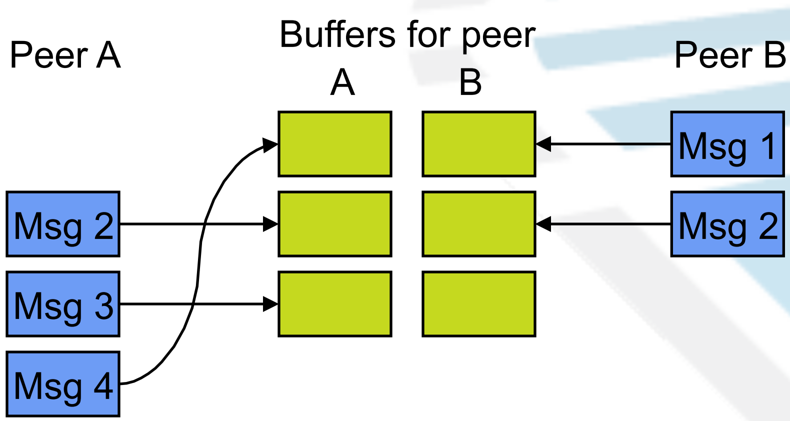


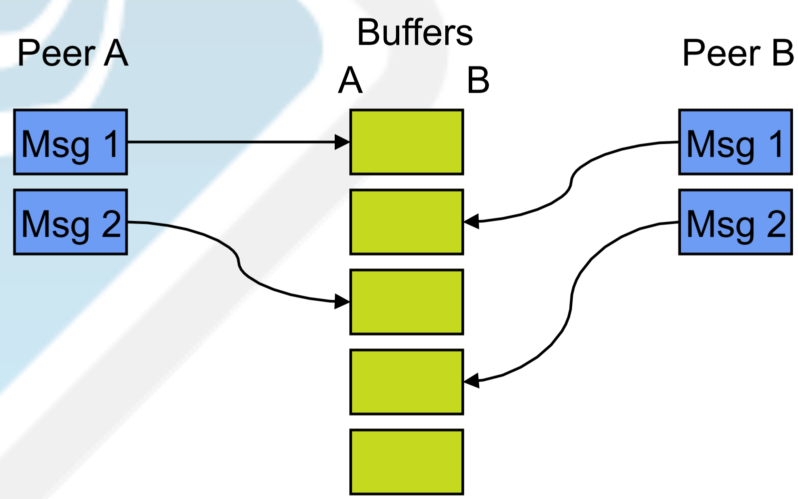


Short: RDMA vs. Send

- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven

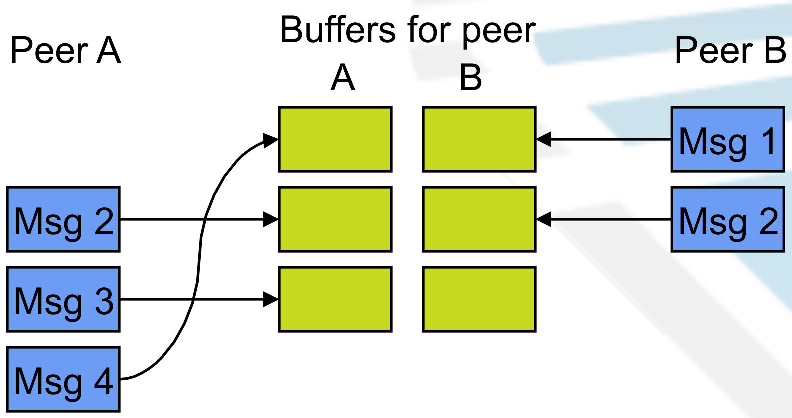
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software

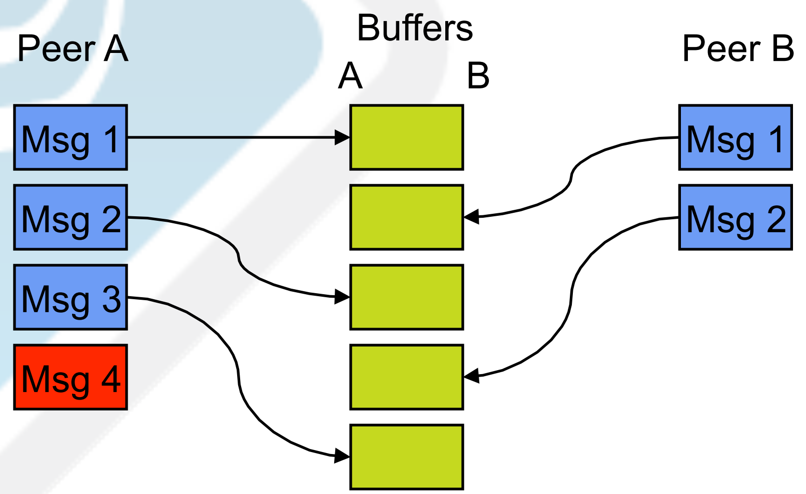




Short: RDMA vs. Send

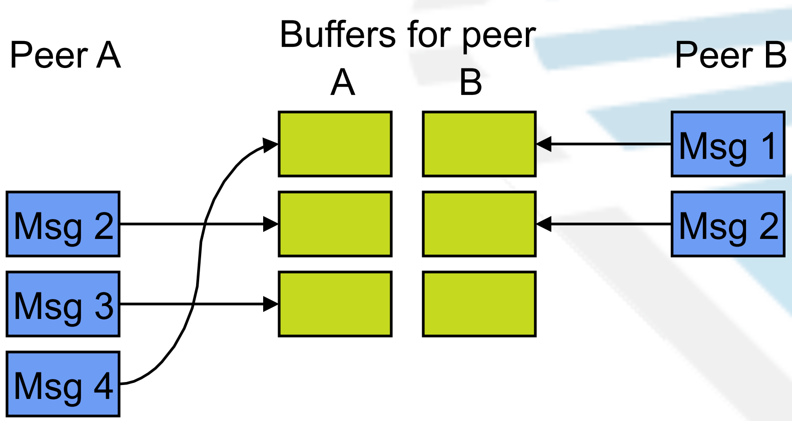
- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software

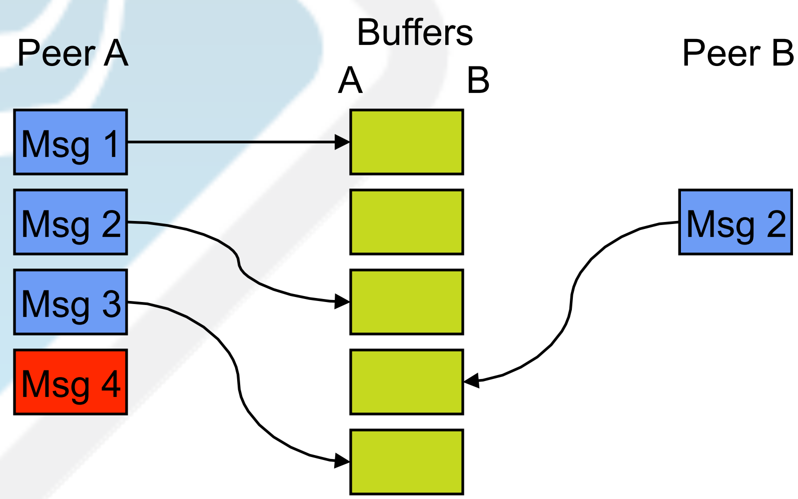




Short: RDMA vs. Send

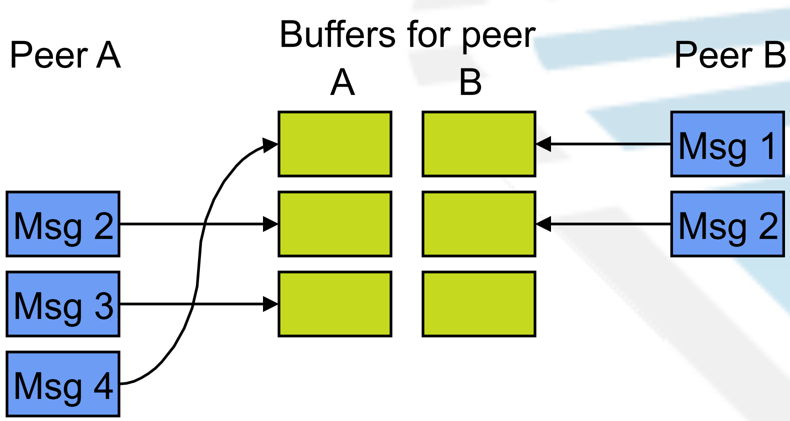
- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software

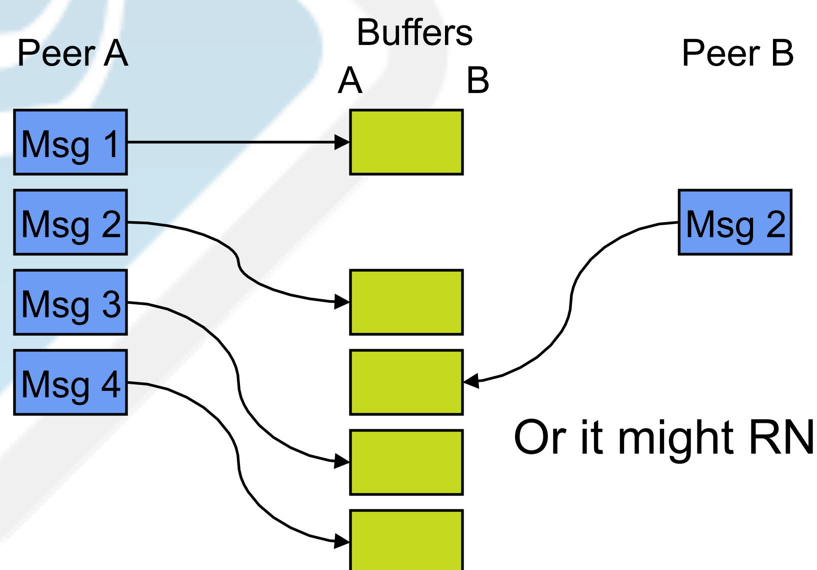




Short: RDMA vs. Send

- RDMA
  - Requires initial setup (exchange addresses)
  - Completely hardware driven
- Send / receive
  - Less initial setup
  - Driver picks buffer
  - Involves remote software





RDMA vs. Send

- RDMA
  - Buffers for each peer: must be (MxN)
  - Exchange addresses
  - MPI maintains accounting/flow control
  - MPI must notice new received messages
  - Unordered
  - "One-sided"
- More work for MPI

- Send / receive
  - Pool of buffers -- can be less than (MxN)
  - Network maintains accounting / [some] flow control
  - Network notifies of new received messages
  - Ordered
  - Two-sided (ACK'ed)
- Less work for MPI

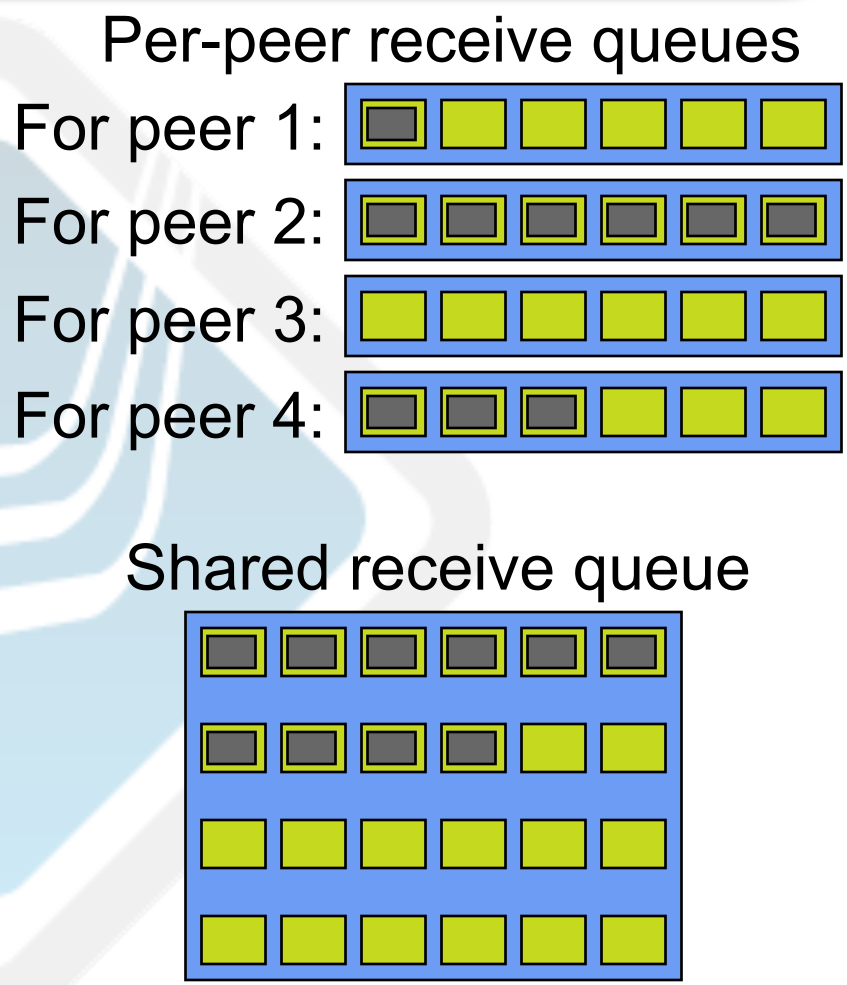
Limiting Short RDMA

- Open MPI allows N “short” RDMA peers
  - (N+1)th peer will use send/receive for short
  - Receiver’s choice
- If M buffers posted for each RDMA peer
  - Total registered memory for short RDMA

    M x N x short_size

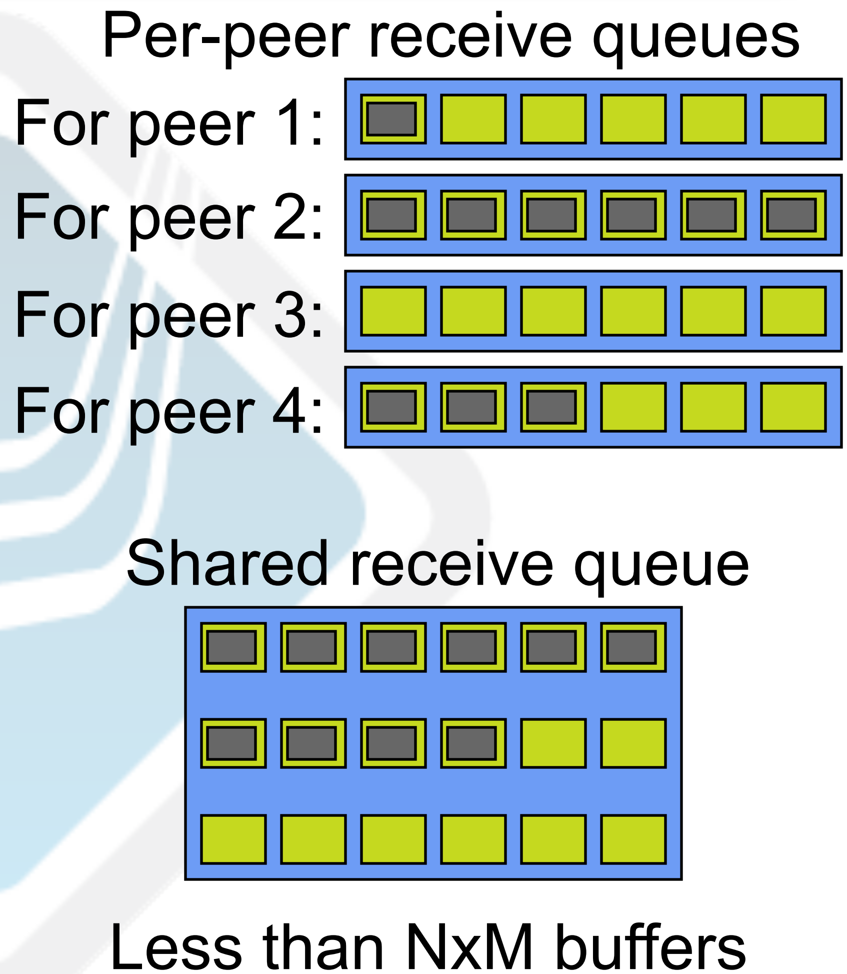
[Shared] Receive Queue

- How to receive messages?
- Per-peer resources
  - Flow control
  - Never error
- Pooled resources
  - No flow control
  - Better utilization
  - Can play stats game
    - Potential for retransmits



[Shared] Receive Queue

- How to receive messages?
- Per-peer resources
  - Flow control
  - Never error
- Pooled resources
  - No flow control
  - Better utilization
  - Can play stats game
    - Potential for retransmits



Long Messages

- For long messages
  - Pipelined protocol
  - Fragment the message, event-driven queue
    - Register
    - Send / receive
    - Unregister
  - (…skipping many details…)
- More complicated if message is not contiguous (not described here)

Long Message Protocol

- `MPI_SEND(long_message, …)`

Long message

- Sent in 3 phases:
  - Eager / match data
  - Send / receive data
  - RDMA data

Long Message Protocol

- `MPI_SEND(long_message, …)`

- 1st phase: Find receiver match
  - Use (copy + send/receive) for first fragment
  - Only send enough to confirm receiver match; do not overwhelm receiver resources
  - Typical rendezvous protocol

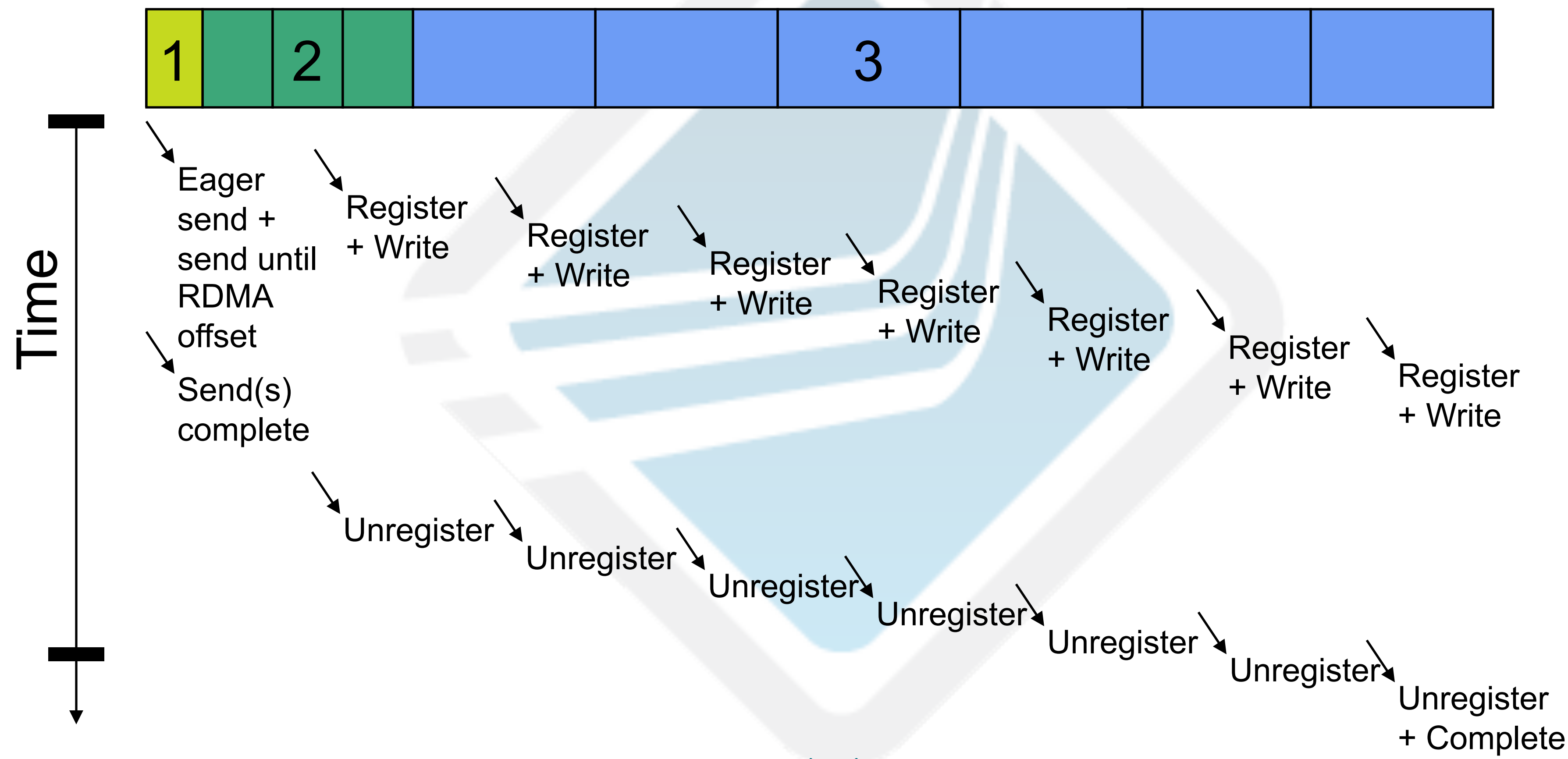
Long Message Protocol

- **`MPI_SEND(long_message, …)`**

- 2nd phase: Hide receiver register latency
  - Use (copy + send/receive) for next few frags
  - Allows overlap of memory registration
- Pipeline subject to max depth setting
  - Conserve resources

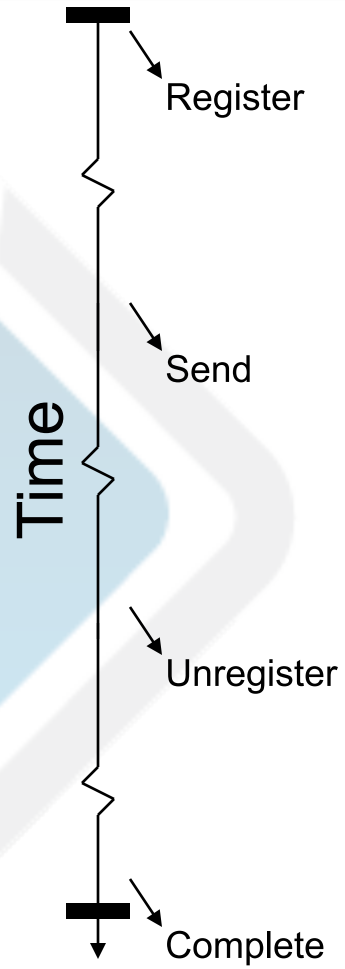
Long Message Protocol

- `MPI_SEND(long_message, …)`



Why So Complex?

- Why not a single register / send / deregister?
  - Each time is directly proportional to buffer length
- Total time is longer
- Many details skipped
  - See paper on www.open-mpi.org



fork() Support

- fork() and registered memory problematic
  - Must differentiate between parent / child registered memory
- Not properly supported until:
  - OFED v1.2
  - Open MPI v1.2.1
  - Kernel 2.6.16 or later (but some Linux distros have backported, e.g., RHEL4U6)
- Query MCA param:
  - btl_openib_have_fork_support

Striping

- Automatic:
  - Short messages round robin across lowest latency BTL module
  - Long message fragments round robin across all available BTL modules
  - Size of long message fragments proportional to network capacity
- Developers investigating more complex scheduling scenarios

How to Activate Striping?

- Do nothing (!)

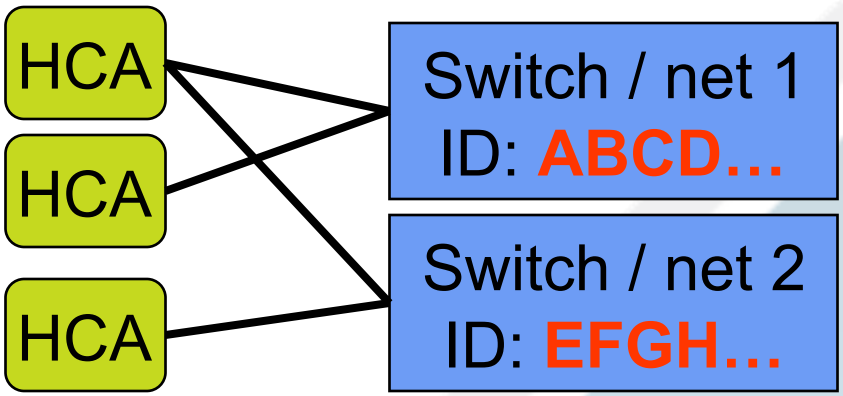
```
mpirun -np 4 a.out
```

- If Open MPI detects multiple active OF ports, it will use them all
  - Assuming each peer port is “reachable”
  - Determined by subnet ID

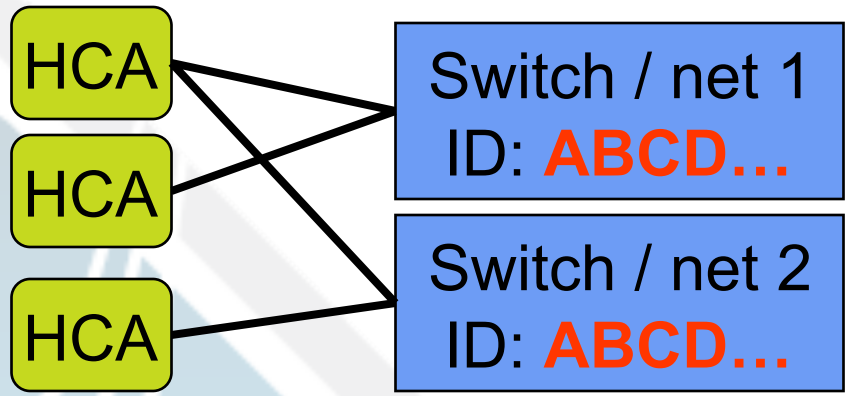
Warning: Default Subnet ID

- Subnet ID uniquely identifies an IB network
  - All Cisco IB switches ship with the default ID
  - FE:80:00:00…
- To compute peer process reachability, Open MPI *must* have different subnet IDs
  - But still, ambiguities exist

Warning: Default Subnet ID

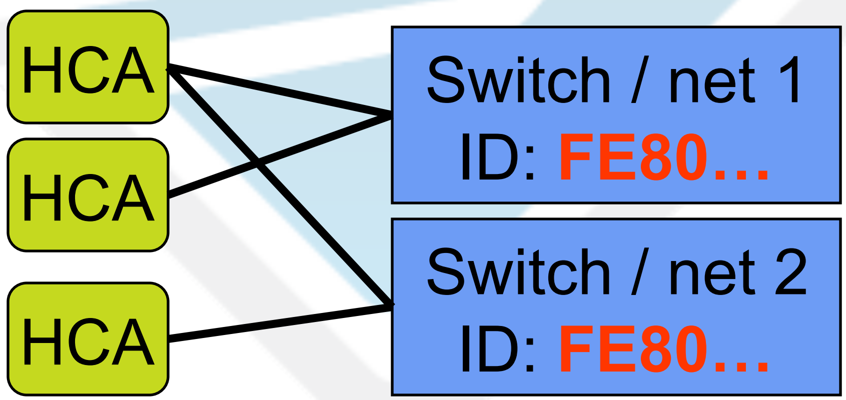


Good



Bad -- OMPI can't compute reachability

Bad -- OMPI can't compute reachability, but will warn



More Information

- Open MPI FAQ
  - General tuning

    http://www.open-mpi.org/faq/?category=tuning
  - InfiniBand / OpenFabrics tuning

    http://www.open-mpi.org/faq/?category=openfabrics

CISCO