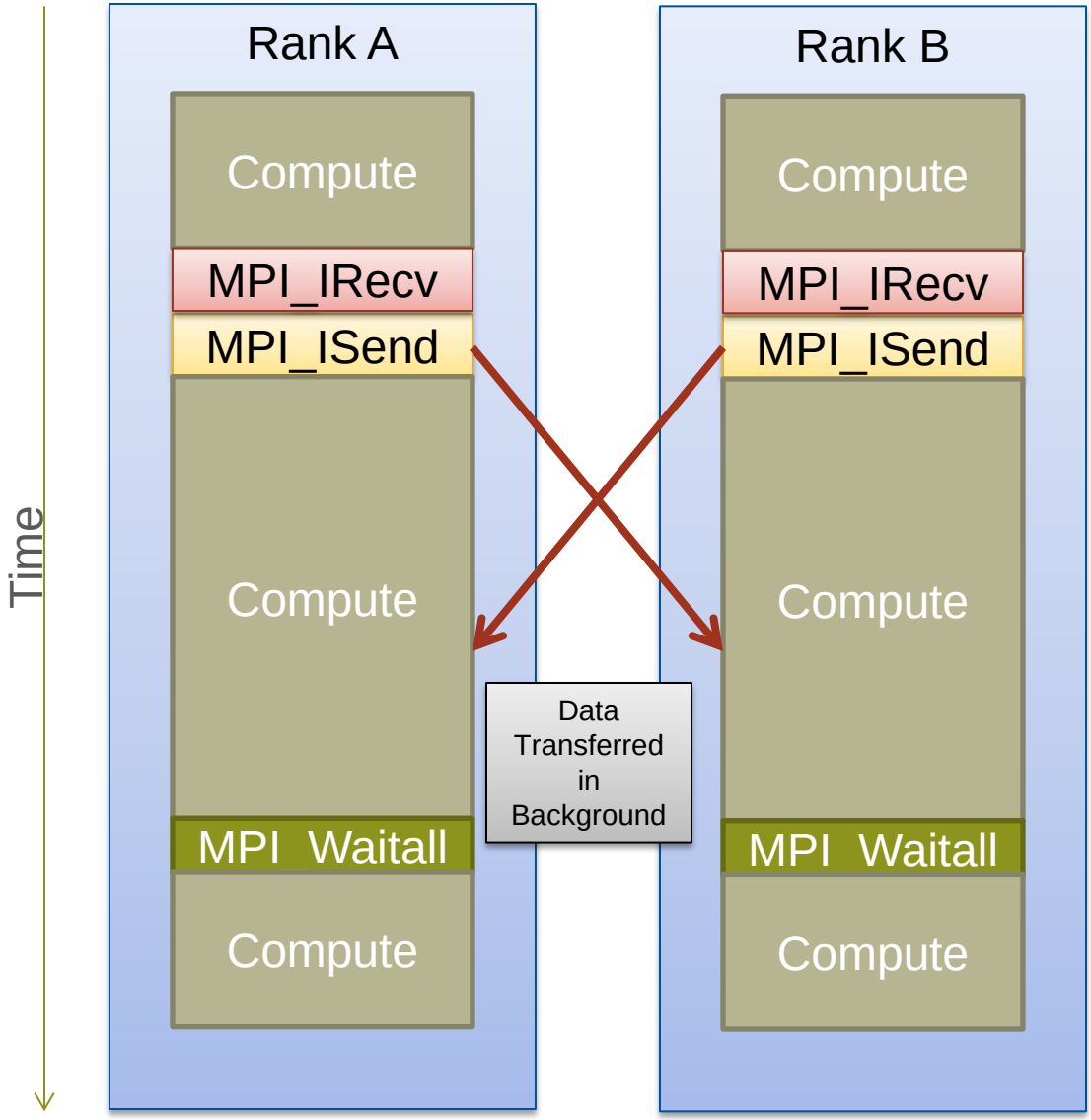
Understanding MPI on Cray XC30

MPICH3 and Cray MPT

- **Cray MPI uses MPICH3 distribution from Argonne**
  - Provides a good, robust and feature rich MPI
  - Cray provides enhancements on top of this:
    - low level communication libraries
    - Point to point tuning
    - Collective tuning
    - Shared memory device is built on top of Cray XPMEM
- **Many layers are straight from MPICH3**
  - Error messages can be from MPICH3 or Cray Libraries.

Overlapping Communication and Computation



The MPI API provides many functions that allow point-to-point messages (and with MPI-3, collectives) to be performed asynchronously.

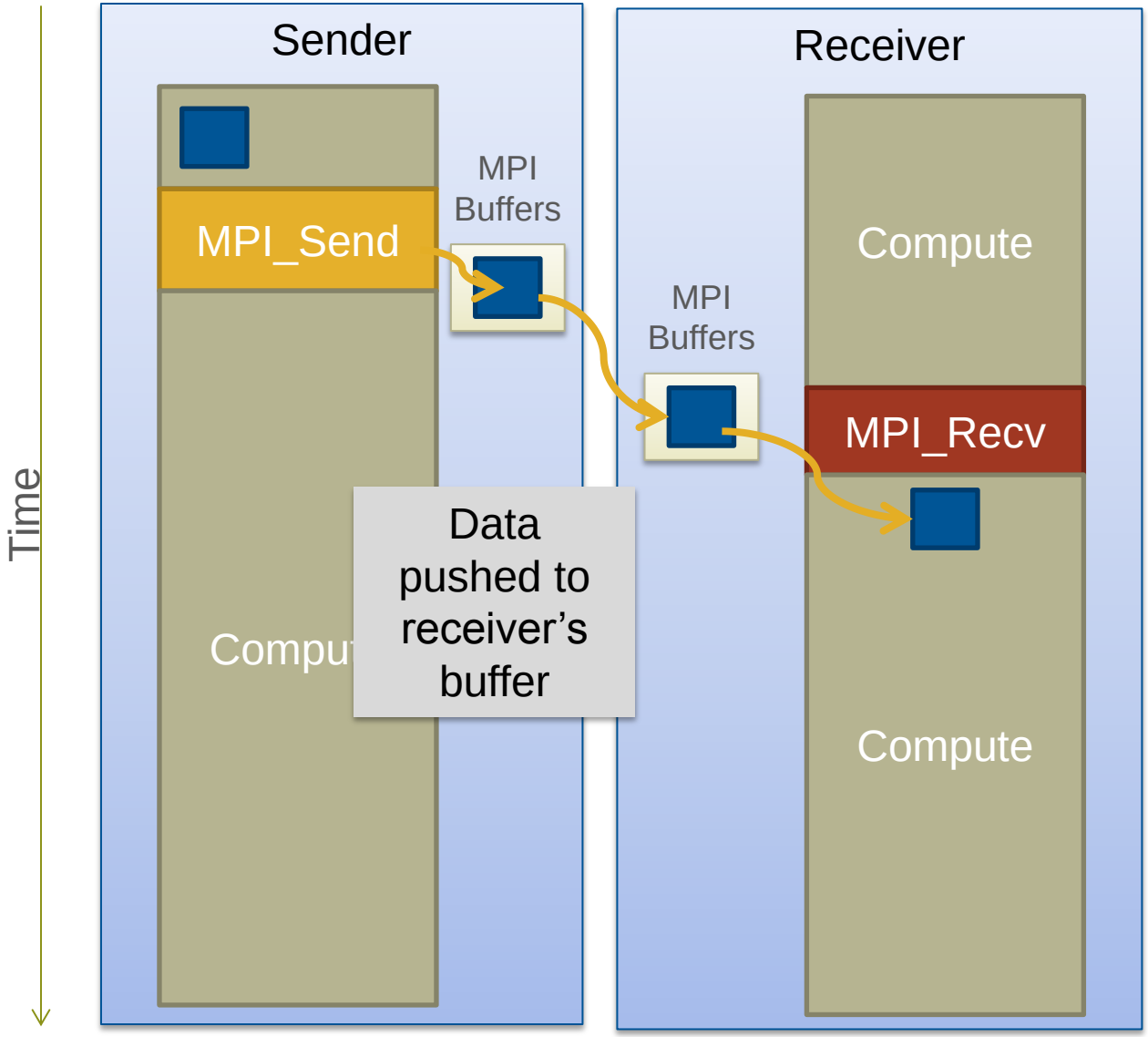
Ideally applications would be able to overlap communication and computation, hiding all data transfer behind useful computation.

Unfortunately this is not always possible at the application and not always possible at the implementation level.

What prevents Overlap?

- **Even though the library has asynchronous API calls, overlap of computation and communication is not always possible**
- **This is usually because the sending process does not know where to put messages on the destination as this is part of the MPI_Recv, not MPI_Send.**
- **Also on Gemini and Aries, complex tasks like matching message tags with the sender and receiver are performed by the host CPU. This means:**
  + Gemini and Aries chips can have higher clock speed and so lower latency and better bandwidth
  + Message matching is always performed by a one fast CPU per rank.
  - Messages can usually only be “progressed” when the program is inside an MPI function or subroutine.

EAGER Messaging – Buffering Small Messages



Smaller messages can avoid this problem using the eager protocol.

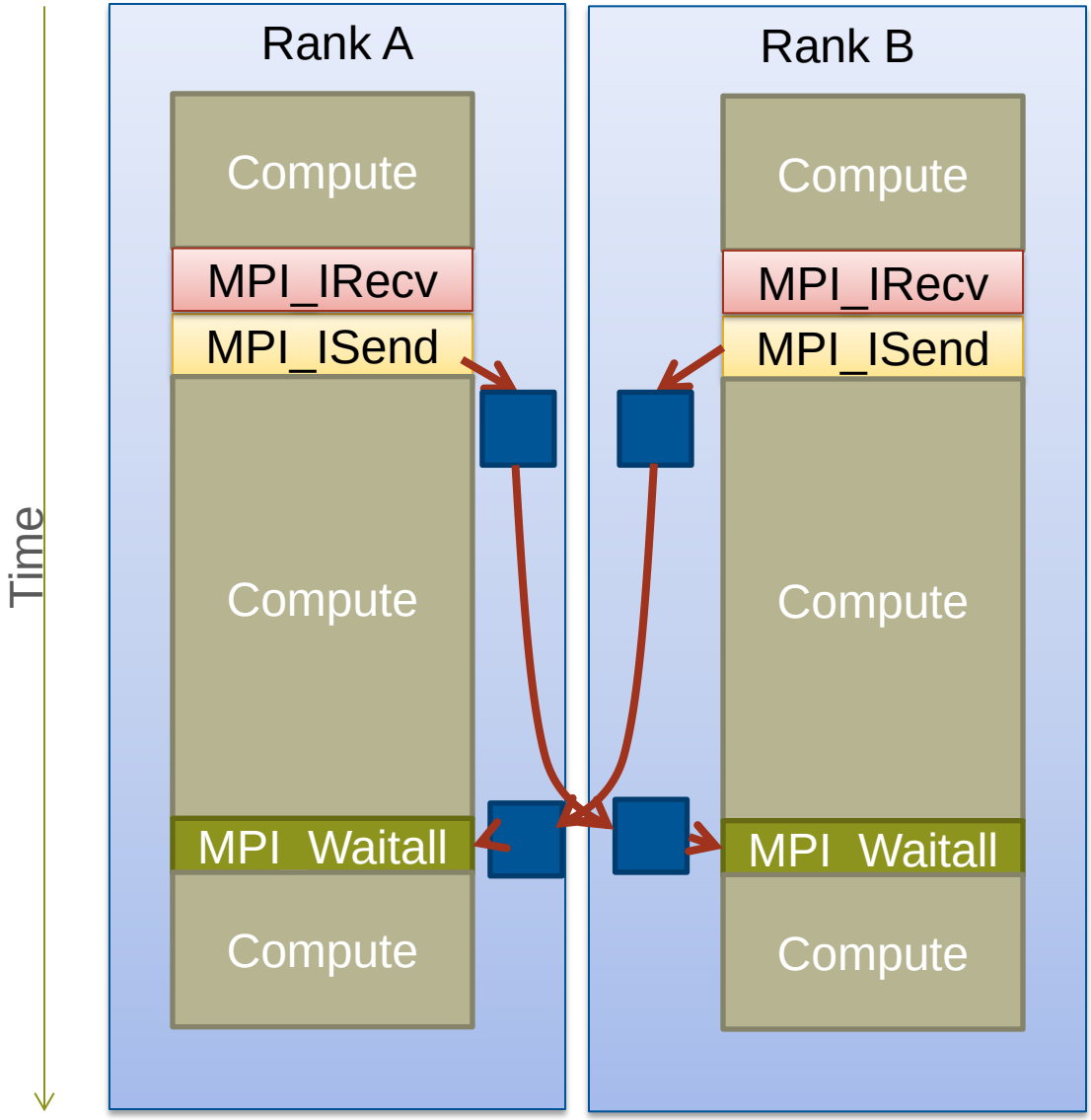
If the sender does not know where to put a message it can be buffered until the sender is ready to take it.

When MPI Recv is called the library fetches the message data from the remote buffer and into the appropriate location (or potentially local buffer)

Sender can proceed as soon as data has been copied to the buffer.

Sender will block if there are no free buffers

EAGER potentially allows overlapping

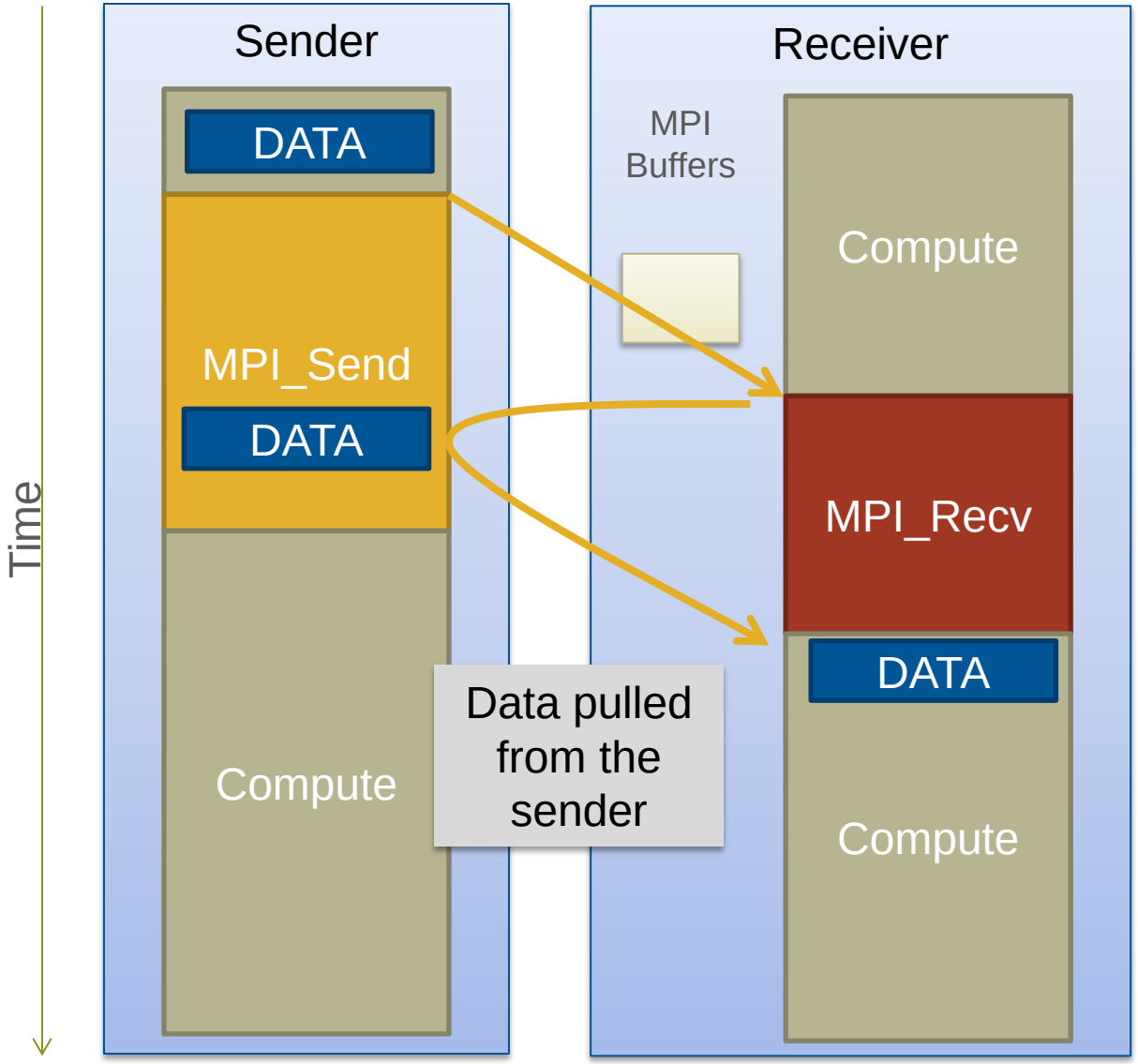


Data is pushed into an empty buffer(s) on the remote processor.

Data is copied from the buffer into the real receive destination when the wait or waitall is called.

Involves an extra memcopy, but much greater opportunity for overlap of computation and communication.

RENDEZVOUS Messaging – Larger Messages



Larger messages (that are too big to fit in the buffers) are sent via the **rendezvous** protocol

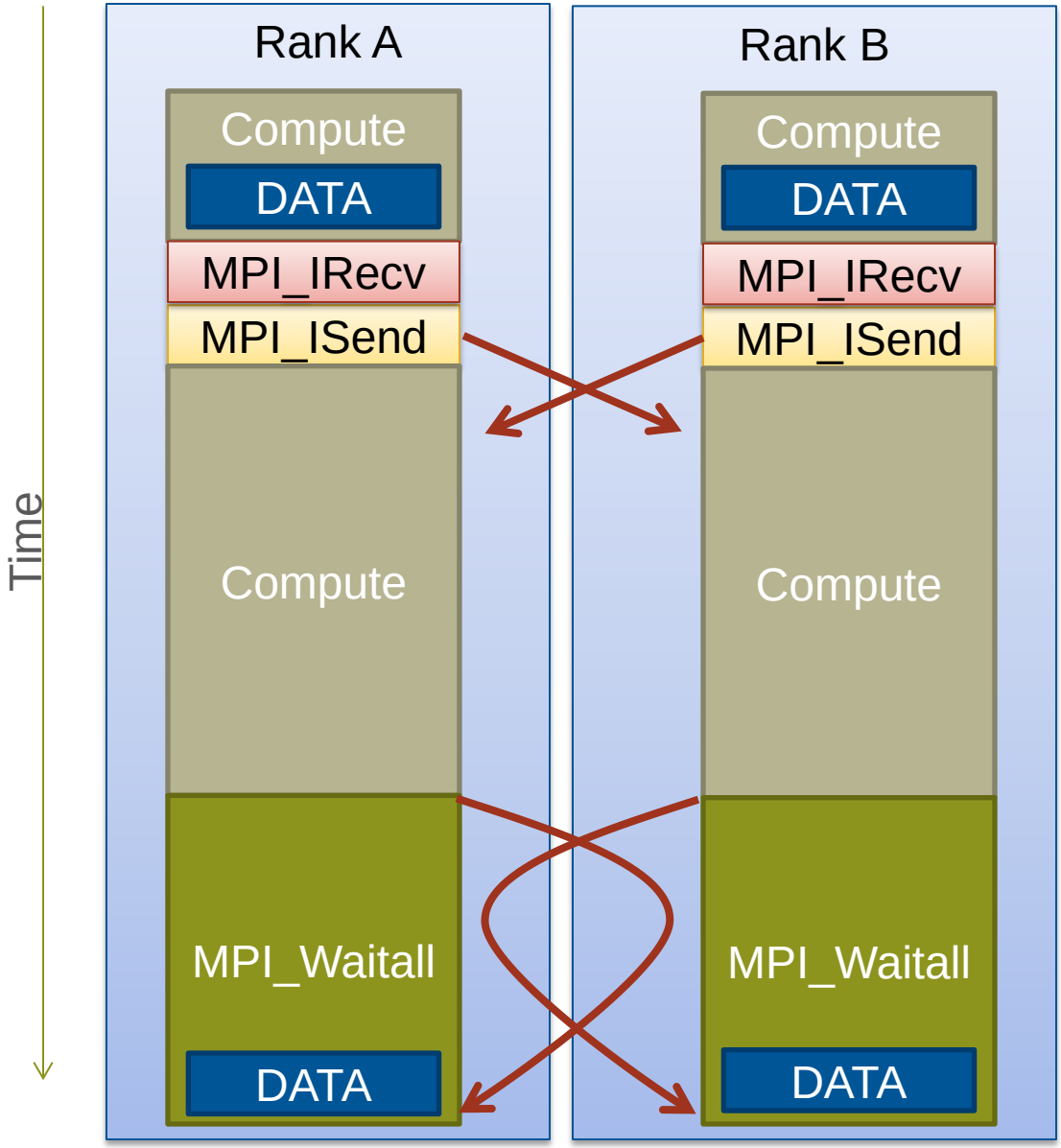
Messages cannot begin transfer until MPI_Recv called by the receiver.

Data is pulled from the sender by the receiver.

Sender must wait for data to be copied to reciever before continuing.

Sender and Receiver block until communication is finished

RENDEZVOUS does not usually overlap



With rendezvous data transfer is often only occurs during the Wait or Waitall statement.

When the message arrives at the destination, the host CPU is busy doing computation, so is unable to do any message matching.

Control only returns to the library when MPI_Waitall occurs and does not return until all data is transferred.

There has been no overlap of computation and communication.

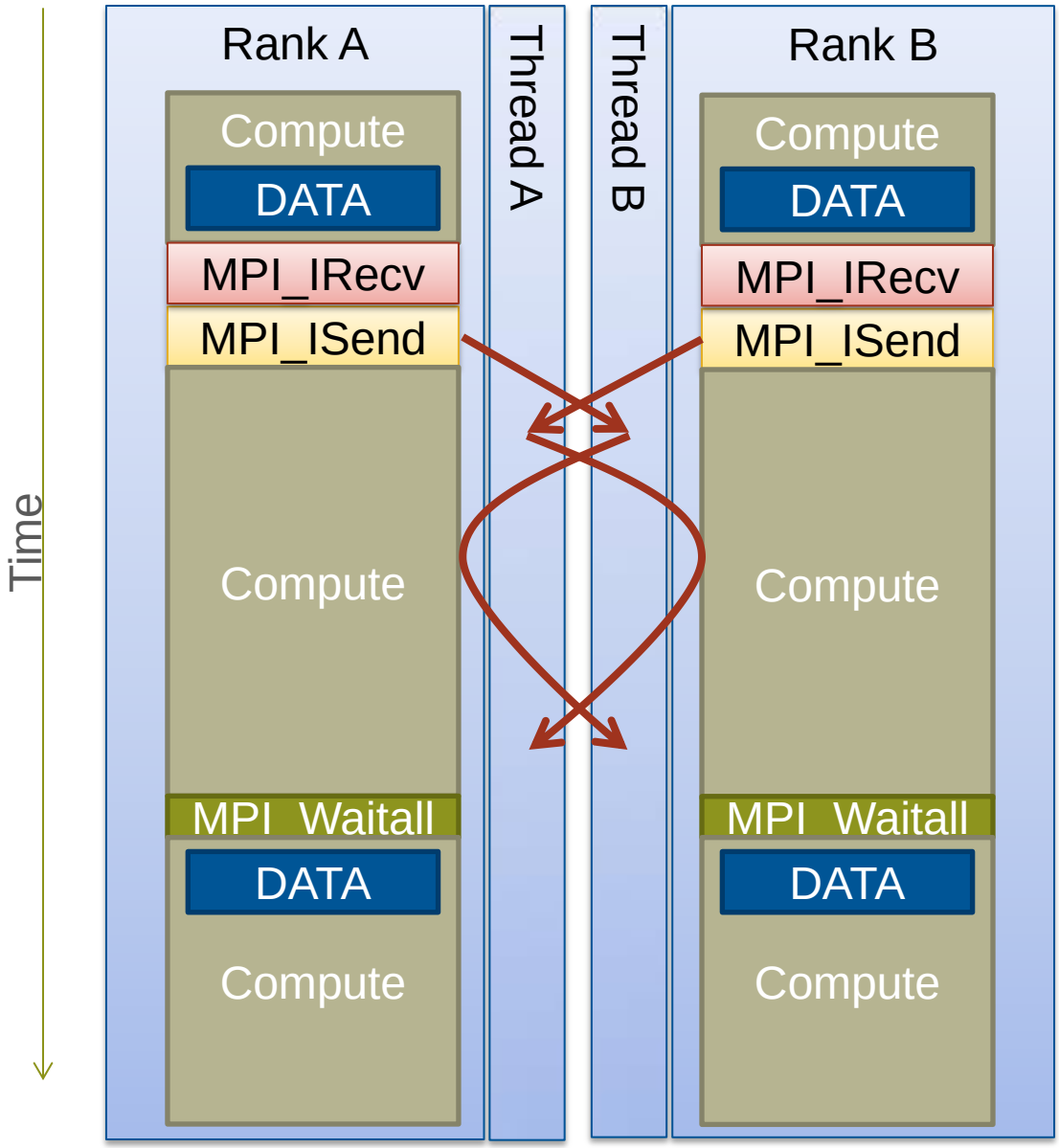
Making more messages EAGER

- One way to improve performance is to send more messages using the eager protocol.
- This can be done by raising the value of the eager threshold, by setting environment variable:
  `export MPICH_GNI_MAX_EAGER_MSG_SIZE=X`
- Values are in bytes, the default is 8192 bytes. Maximum size is 131072 bytes (128KB).
- Try to post MPI_IRecv calls before the MPI_ISend call to avoid unecessary buffer copies.

Consequences of more EAGER messages

- Sending more messages via EAGER places more demands on buffers on receiver.
- If the buffers are full, transfer will wait until space is available or until the Wait.
- Buffer size can be increased using:
  `export MPICH_GNI_NUM_BUFS=X`
- Buffers are 32KB each and default number is 64 (total of 2MB).
- Buffer memory space is competing with application memory, so we recommend only moderate increases.

Progress threads help overlap



Cray's MPT library can spawn additional threads that allow progress of messages while computation occurs in the background.

Thread performs message matching and intiatiates the transfer.

Data has already arrived by the time Waitall is called, so overlap between compute and communication.

MPI - Async Progress Engine Support

- Used to improve communication/computation overlap
  - Each MPI rank starts a “helper thread” during MPI_Init
- Helper threads progress MPI engine while application computes
- Only inter-node messages that use Rendezvous Path are progressed (relies on BTE for data motion)
- To enable on XC when using 1 stream per core:
  - `export MPICH_NEMESIS_ASYNC_PROGRESS=1`
  - `export MPICH_MAX_THREAD_SAFETY=multiple`
  - `export MPICH_GNI_USE_UNASSIGNED_CPUS=enabled`
  - Run application: `aprun –n XX a.out`
- To enable on XC when using 2 streams per core recommend running with the corespec option:
  - `export MPICH_NEMESIS_ASYNC_PROGRESS=1`
  - `export MPICH_MAX_THREAD_SAFETY=multiple`
  - Run application with corespec: `aprun –n XX -r [1-2] a.out`
- 10% or more performance improvements with some apps

Other Techniques - Collectives

- `MPICH_COLL_OPT_OFF=<collective name>` switches off the Cray optimized algorithm for a given collective and uses the original MPICH algorithm
  - E.g. `MPICH_COLL_OPT_OFF=mpi_allgather`
- The algorithm selection for all-to-all routines (allgather(v), alltoall(v)) is based on the number of ranks on the calling communicator and the message sizes.
- This can be adjusted with `MPICH_XXXX_VSHORT_MSG` environment variable, where XXXX=collective, e.g. ALLGATHER.

Why use Huge Pages

- **The Aries performs better with HUGE pages than with 4K pages. The Aries can map more pages using fewer resources meaning communications may be faster.**

- **Huge Pages will also affect TLB performance:**
  - Your code may run with fewer TLB misses (hence faster)
  - However, your code may load extra data and so run slower
  - Only way to know is by experimentation.

- **Use modules to change default page sizes (`man intro_hugepages`):**
  - `e.g. module load craype-hugepages#`
    - `craype-hugepages128K`
    - `craype-hugepages512K`
    - **`craype-hugepages2M`**
    - **`craype-hugepages8M`**
    - `craype-hugepages16M`
    - `craype-hugepages64M`

Most commonly successfully on Cray XC

MPICH_GNI_DYNAMIC_CONN

- Enabled by default
- Normally want to leave enabled so mailbox resources (memory, NIC resources) are allocated only when the application needs them
- If application does all-to-all or many-to-one/few, may as well disable dynamic connections. This will result in significant startup/shutdown costs though.
- Syntax for disabling:

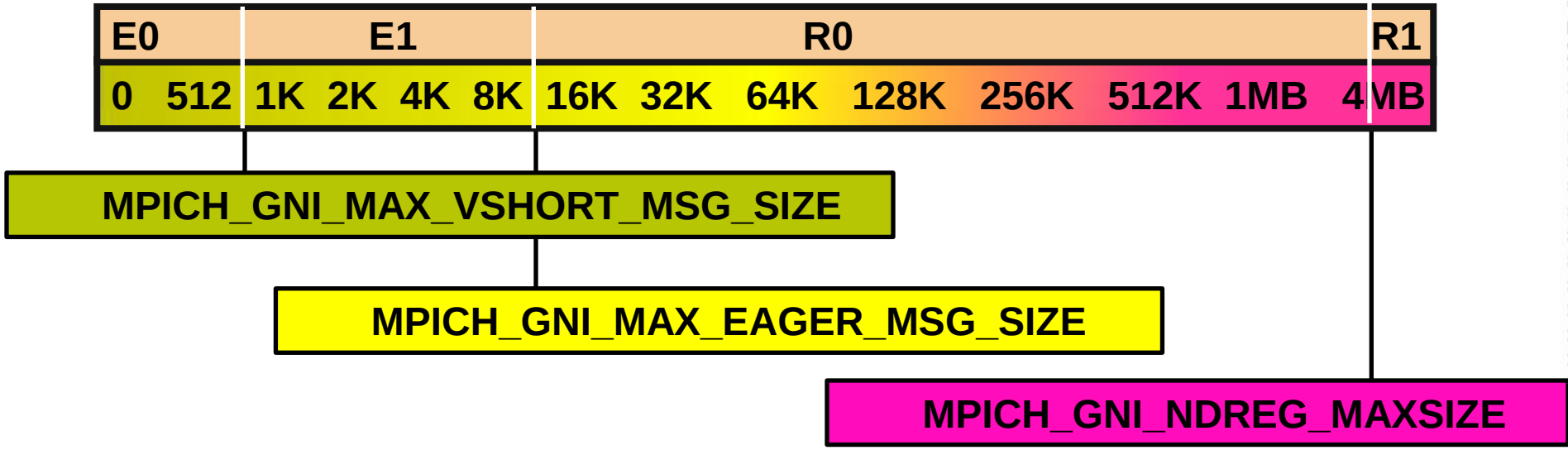
  export MPICH_GNI_DYNAMIC_CONN=disabled

More detail.

- In reality, there are two different Eager and Rendezvous protocols in use with Cray's MPI.
- The following slides show more detail on the implementation and possible techniques for tuning them.

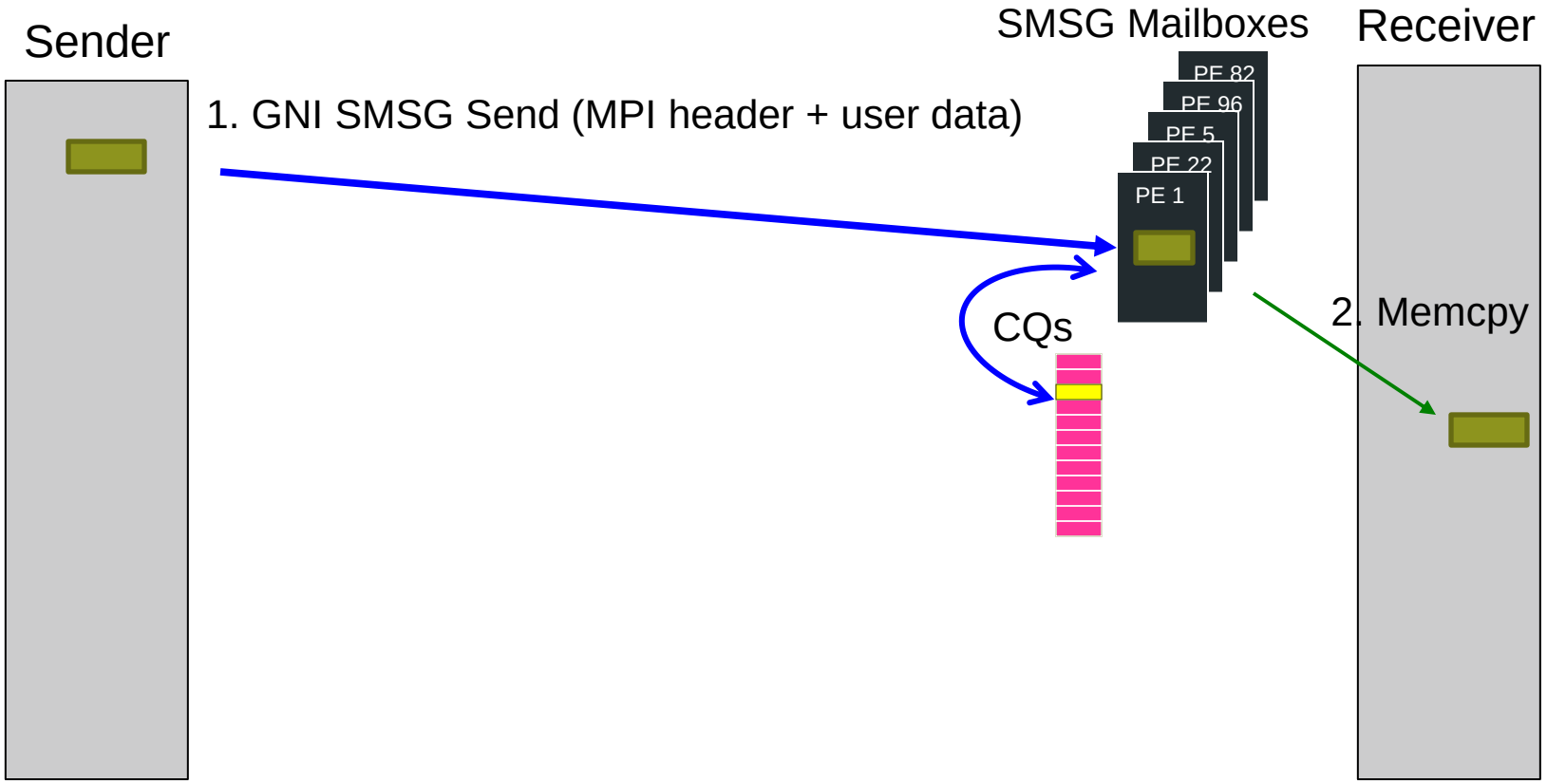
Day in the Life of an MPI Message

- **Four Main Pathways through the MPICH2 GNI NetMod**
  - Two EAGER paths (E0 and E1)
    - For a message that can fit in a GNI SMSG mailbox (E0)
    - For a message that can't fit into a mailbox but is less than MPICH_GNI_MAX_EAGER_MSG_SIZE in length (E1)
  - Two RENDEZVOUS (aka LMT) paths : R0 (RDMA get) and R1 (RDMA put)
- **Selected Pathway is based on Message Size**



Day in the Life of Message type E0

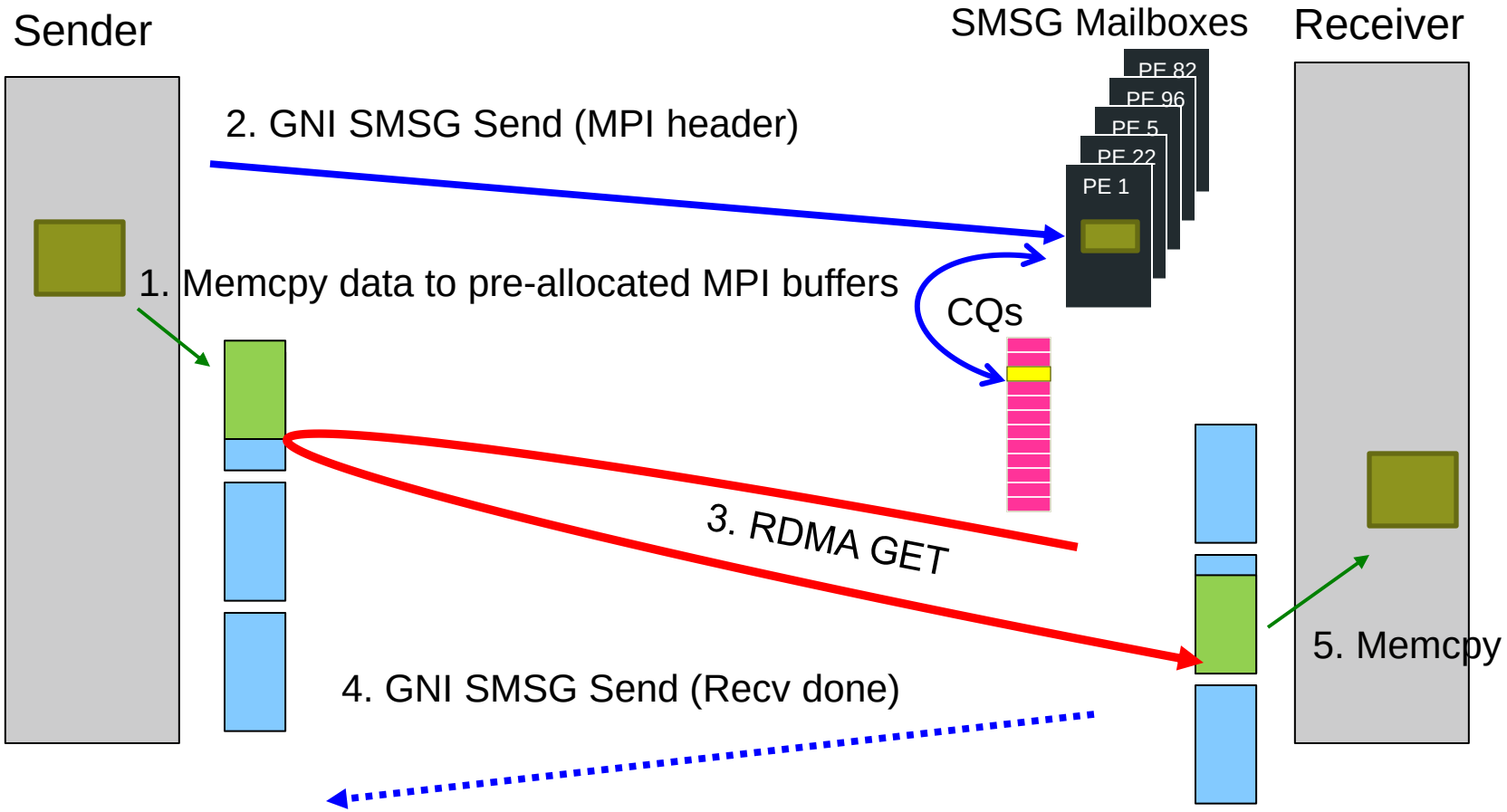
EAGER messages that fit in the GNI SMSG Mailbox



- **GNI SMSG Mailbox size changes with number of ranks in job**
- **If user data is 16 bytes or less, it is copied into the MPI header**

Day in the Life of Message type E1

EAGER messages that don't fit in the GNI SMSG Mailbox



- **User data is copied into internal MPI buffers on both send and receive side**

MPICH_GNI_NUM_BUFS
default 64 buffers, each 32K

EAGER Message Protocol

Default mailbox size varies with number of ranks in the job

- **Protocol for messages that can fit into a GNI SMSG mailbox**
- **The default varies with job size, although this can be tuned by the user to some extent**

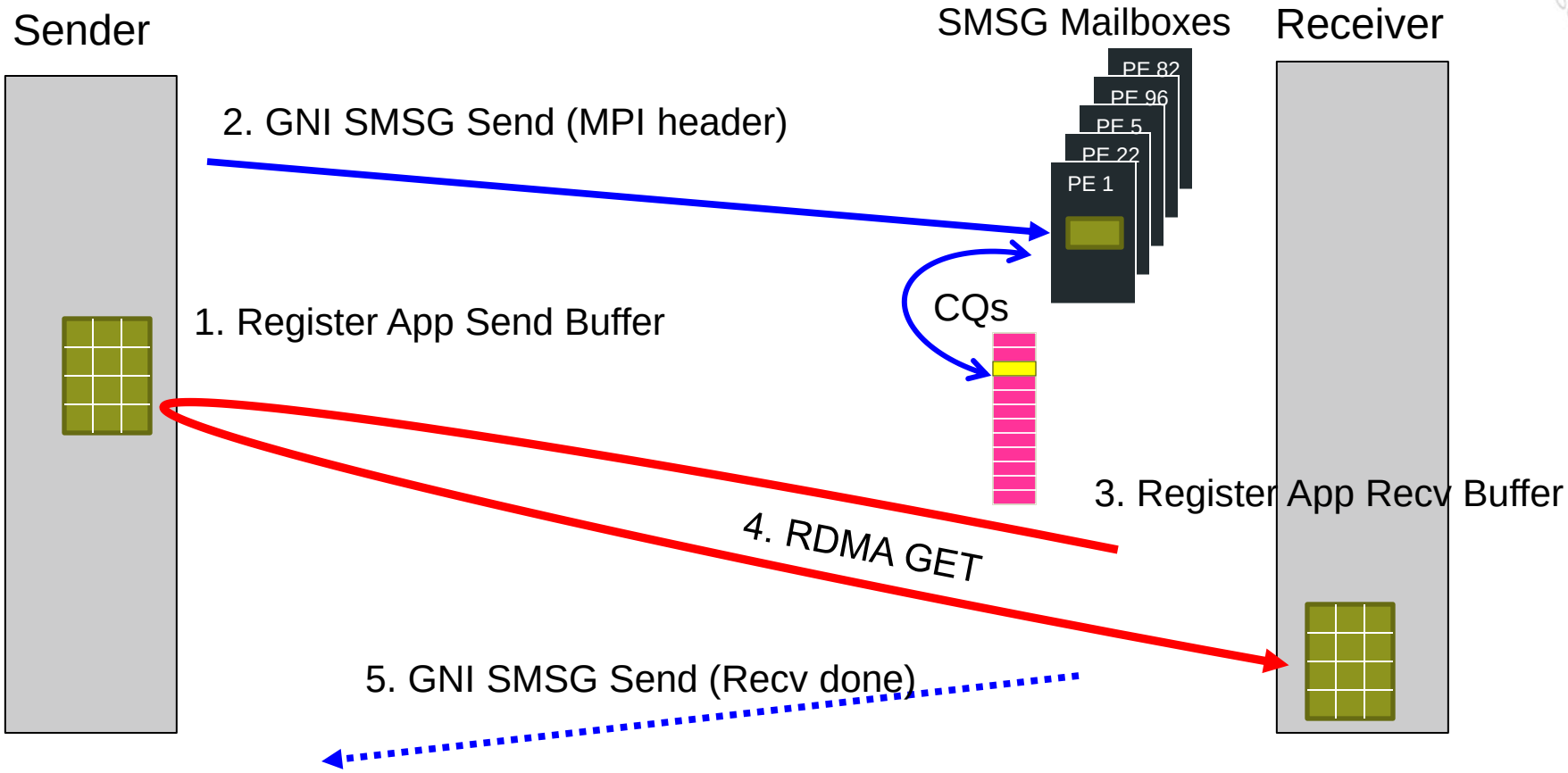
Ranks in Job	Max user data (MPT 5.3)	MPT 5.4 and later
< = 512 ranks	984 bytes	8152 bytes
> 512 and <= 1024	984 bytes	2008 bytes
> 1024 and < 16384	472 bytes	472 bytes
> 16384 ranks	216 bytes	216 bytes

MPI env variables affecting the pathway

- **MPICH_GNI_MAX_VSHORT_MSG_SIZE**
  - Controls max size for E0 Path
    Default varies with job size: 216-984 bytes
- **MPICH_GNI_MAX_EAGER_MSG_SIZE**
  - Controls max message size for E1 Path (Default is 8K bytes)
- **MPICH_GNI_NDREG_MAXSIZE**
  - Controls max message size for R0 Path (Default is 4MB bytes)
- **MPICH_GNI_LMT_PATH=disabled**
  - Can be used to Disable the entire Rendezvous (LMT) Path

Day in the Life of Message type R0

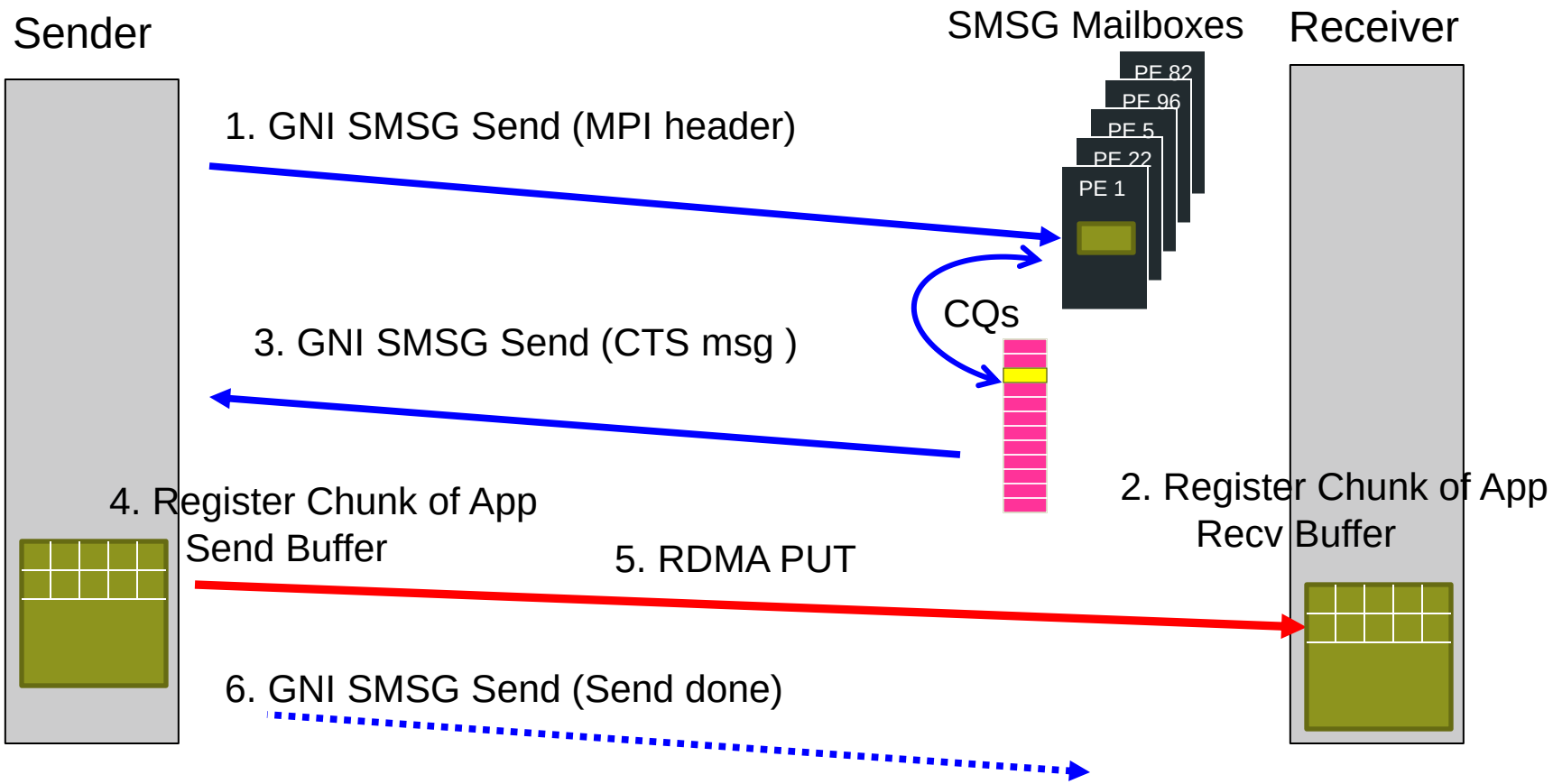
Rendezvous messages using RDMA Get



- **No extra data copies**
- **Best chance of overlapping communication with computation**

Day in the Life of Message type R1

Rendezvous messages using RDMA Put



- ***Repeat steps 2-6 until all sender data is transferred***
- **Chunksize is MPI_GNI_MAX_NDREG_SIZE (default of 4MB)**

MPICH_GNI_MAX_VSHORT_MSG_SIZE

- Can be used to control the maximum size message that can go through the private SMSG mailbox protocol (E0 *eager* path).
- Default varies with job size
- Maximum size is 8192 bytes. Minimum is 80 bytes.
- If you are trying to demonstrate an MPI_Alltoall at very high count, with smallest possible memory usage, may be good to set this as low as possible.
- If you know your app has a scalable communication pattern, and the performance drops at one of the edges shown on table on slide 18, you may want to set this environment variable.
- Pre-posting receives for this protocol avoids a potential extra memcpy at the receiver.

MPICH_GNI_MAX_EAGER_MSG_SIZE

- Default is 8192 bytes
- Maximum size message that go through the *eager* (E1) protocol
- May help for apps that are sending medium size messages, and do better when loosely coupled. Does application have a large amount of time in MPI_Waitall? Setting this environment variable higher may help.
- Maximum allowable setting is 131072 bytes
- Pre-posting receives can avoid potential double memcpy at the receiver.
- Note that a 40-byte Nemesis header is included in account for the message size.

MPICH_GNI_MBOX_PLACEMENT

- Provides a means for controlling which memories on a node are used for some SMSG mailboxes (private).
- Default is to place the mailboxes on the memory where the process is running when the memory for the mailboxes is faulted in.
- For optimal MPI message rates, better to place mailboxes on memory of die0 (where Aries is attached).
- Only applies to first 4096 mailboxes of each rank on the node.
- Syntax for enabling placement of mailboxes near the Aries:
  export MPICH_GNI_MBOX_PLACEMENT=nic

MPICH_GNI_RDMA_THRESHOLD

- Default is now 1024 bytes
- Controls the threshold at which the GNI netmod switches from using FMA for RDMA read/write operations to using the BTE.
- Since BTE is managed in the kernel, BTE initiated RDMA requests can progress even if the applications isn't in MPI.
- Owing to Opteron/HT quirks, the BTE is often better for moving data to/from memories that are farther from the Aries.
- But using the BTE may lead to more interrupts being generated

MPICH_GNI_NDREG_LAZYMEM

- Default is enabled. To disable export MPICH_GNI_NDREG_LAZYMEM=disabled
- Controls whether or not to use a lazy memory deregistration policy inside UDREG. Memory registration is expensive so this is usually a good idea.
- Only important for those applications using the LMT (large message transfer) path, i.e. messages greater than MPICH_GNI_MAX_EAGER_MSG_SIZE.
- Disabling results in a significant drop in measured bandwidth for large transfers ~40-50 %.
- If code only works with this feature being diabled => BUG

MPICH_GNI_DMAPP_INTEROP

- Only relevant for mixed MPI/SHMEM/UPC/CAF codes
- Normally want to leave enabled so MPICH2 and DMAPP can share the same memory registration cache,
- May have to disable for SHMEM codes that call *shmem_init* after *MPI_Init.*
- May want to disable if trying to add SHMEM/CAF to an MPI code and notice a big performance drop.
- Syntax:

  export MPICH_GNI_DMAPP_INTEROP=disabled

MPICH_GNI_DMAPP_INTEROP

- **May have to set to disable if one gets a traceback like this:**

```
Rank 834 Fatal error in MPI_Alltoall: Other MPI error, error stack:
MPI_Alltoall(768).....: MPI_Alltoall(sbuf=0x2aab9c301010,
scount=2596, MPI_DOUBLE, rbuf=0x2aab7ae01010, rcount=2596,
MPI_DOUBLE,
comm=0x84000004) failed
MPIR_Alltoall(469).....:
MPIC_Isend(453).....:
MPID_nem_lmt_RndvSend(102).....:
MPID_nem_gni_lmt_initiate_lmt(580)......: failure occurred while attempting to
send RTS packet
MPID_nem_gni_iStartContigMsg(869).....:
MPID_nem_gni_iSendContig_start(763).....:
MPID_nem_gni_send_conn_req(626).....:
MPID_nem_gni_progress_send_conn_req(193):
MPID_nem_gni_smsg_mbox_alloc(357).....:
MPID_nem_gni_smsg_mbox_block_alloc(268).: GNI_MemRegister
GNI_RC_ERROR_RESOURCE)
```

MPICH_GNI_NUM_BUFS

- Default is 64 32K buffers (2M total)
- Controls the number of 32KB DMA buffers available for each rank to use in the GET-based Eager protocol (E1).
- May help to modestly increase. But other resources constrain the usability of a large number of buffers, so don’t go berserk with this one.
- Syntax:

  export MPICH_GNI_NUM_BUFS=X