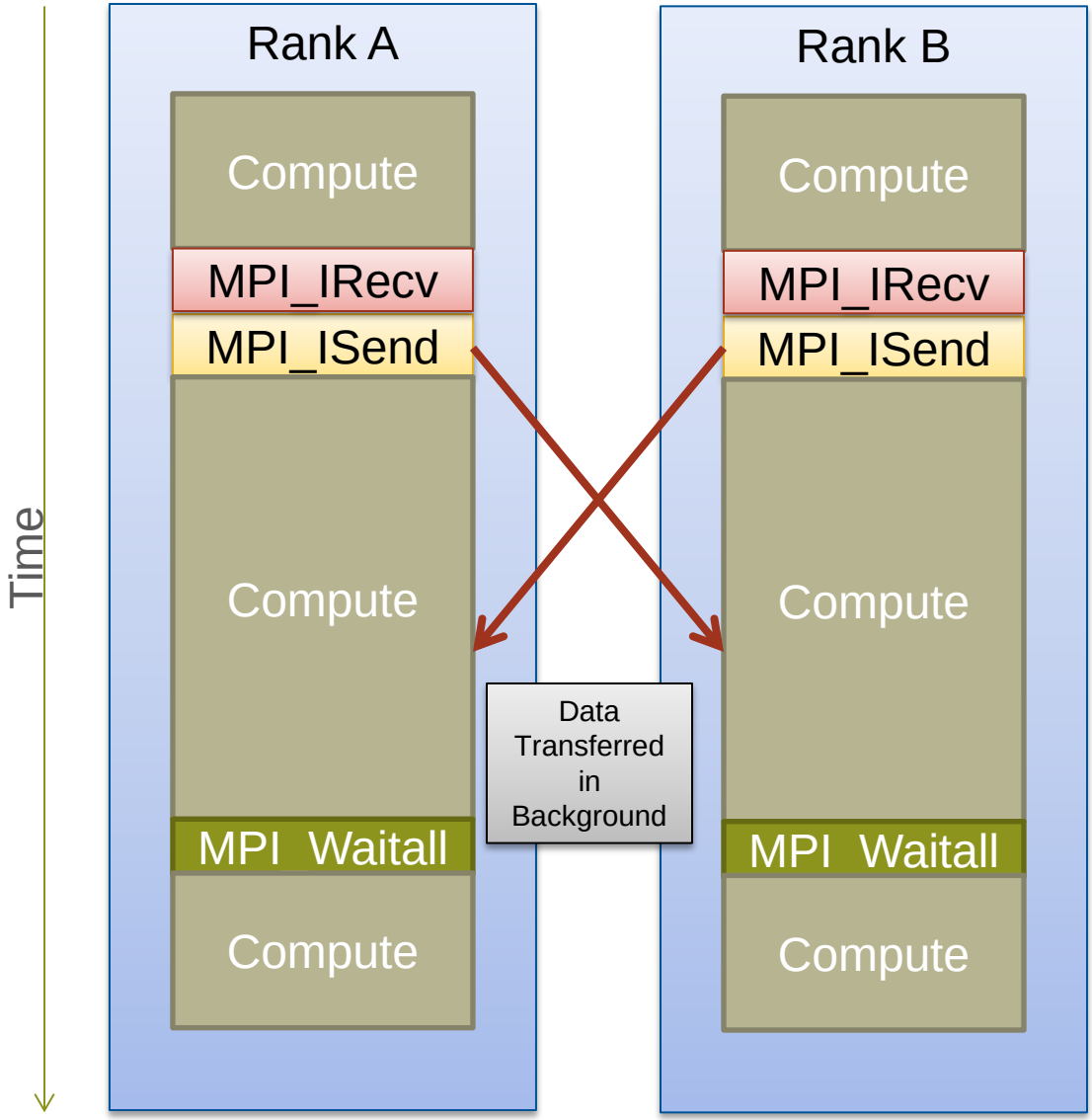
Understanding MPI on Cray XE6

MPICH2 and Cray MPT

- **Cray MPI uses MPICH2 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 MPICH2**
  - Error messages can be from MPICH2 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?

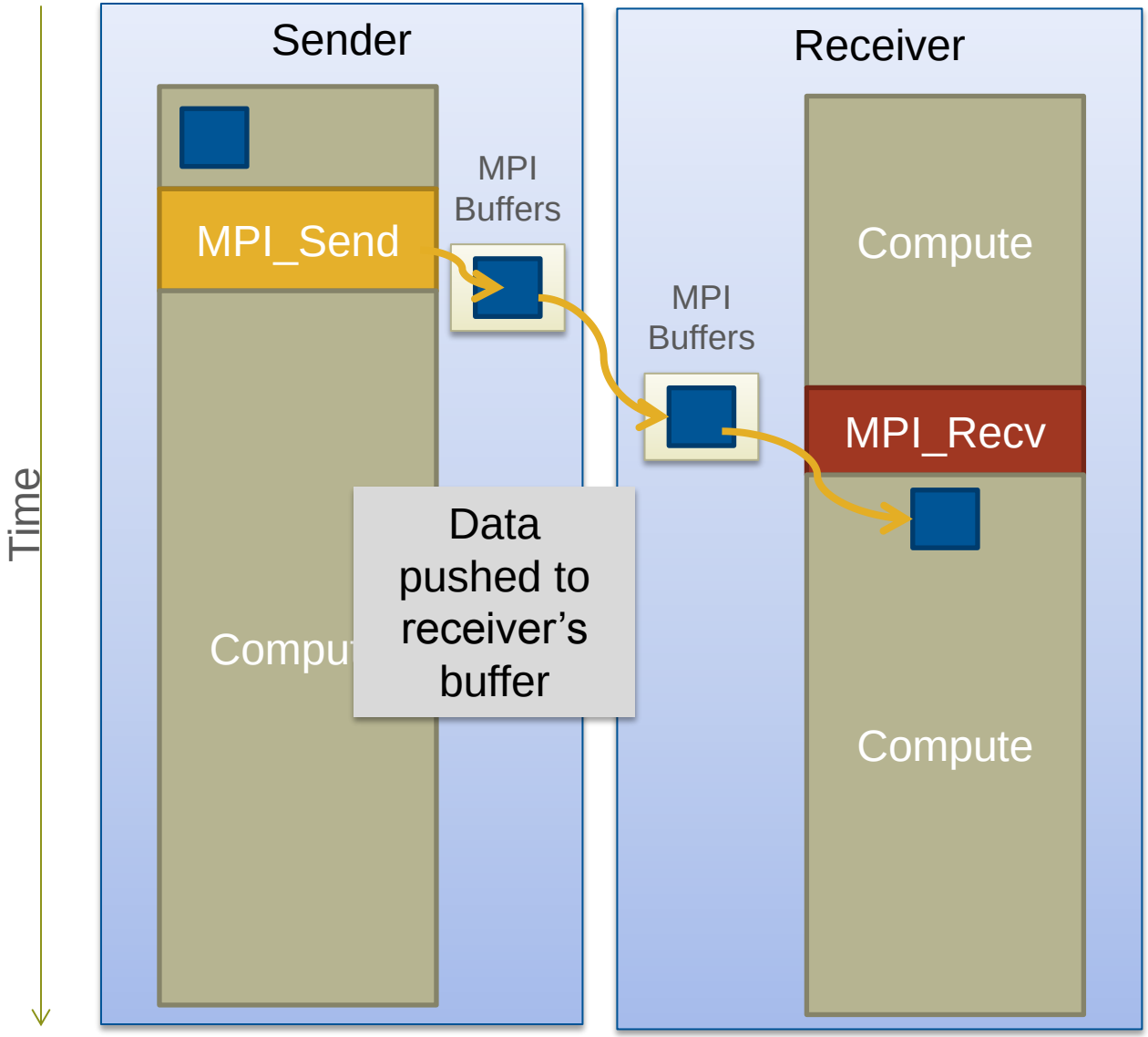
- **Even though the library has asynchronous API calls, overlap of computation and communication is not always possible**
- **This is usually because the sending process does not know where to put messages on the destination as this is part of the MPI_Recv, not MPI_Send.**
- **Also on Gemini and Aries, 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.

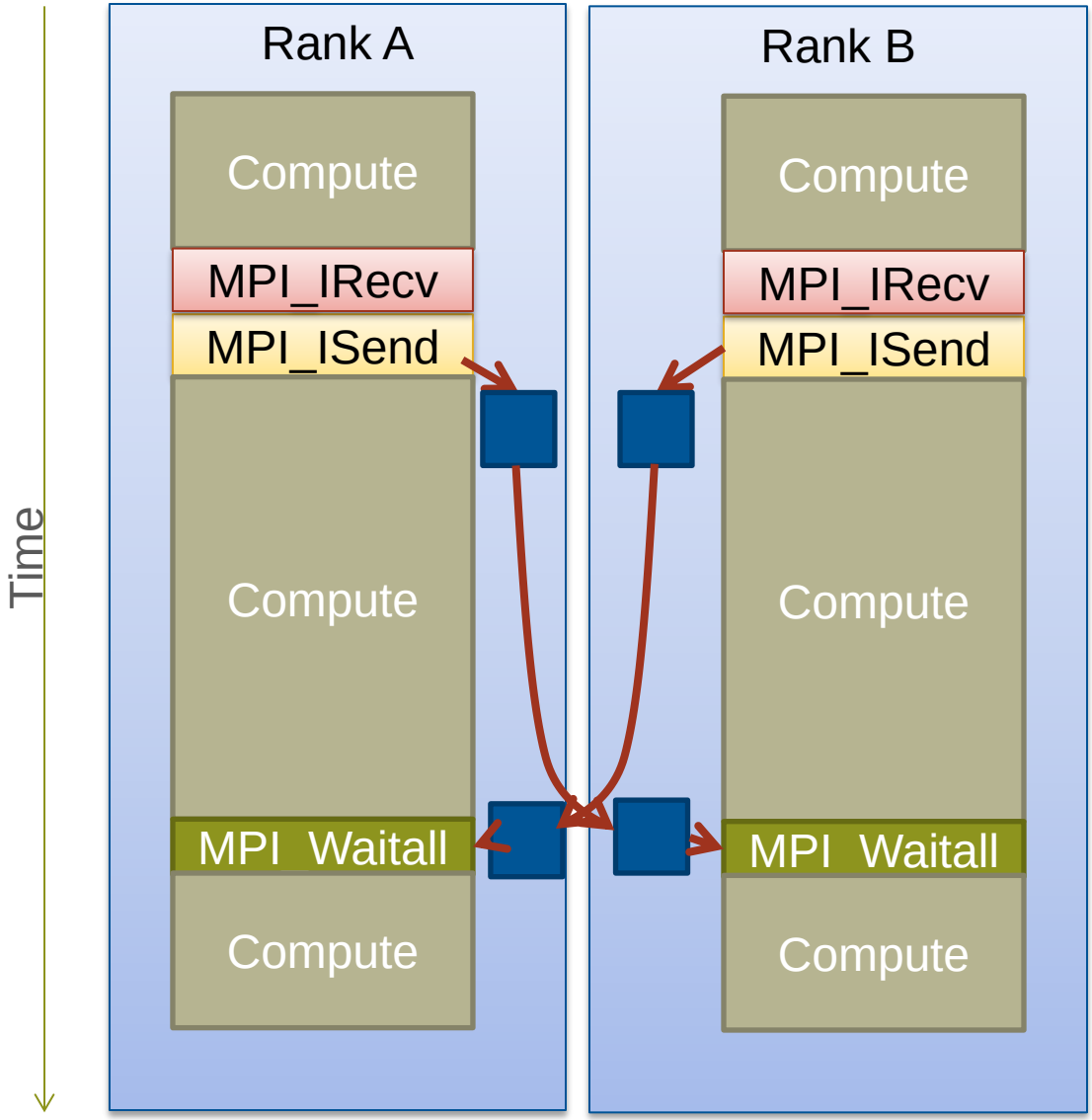
If the sender does not know where to put a message it can be buffered until the sender is ready to take it.

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

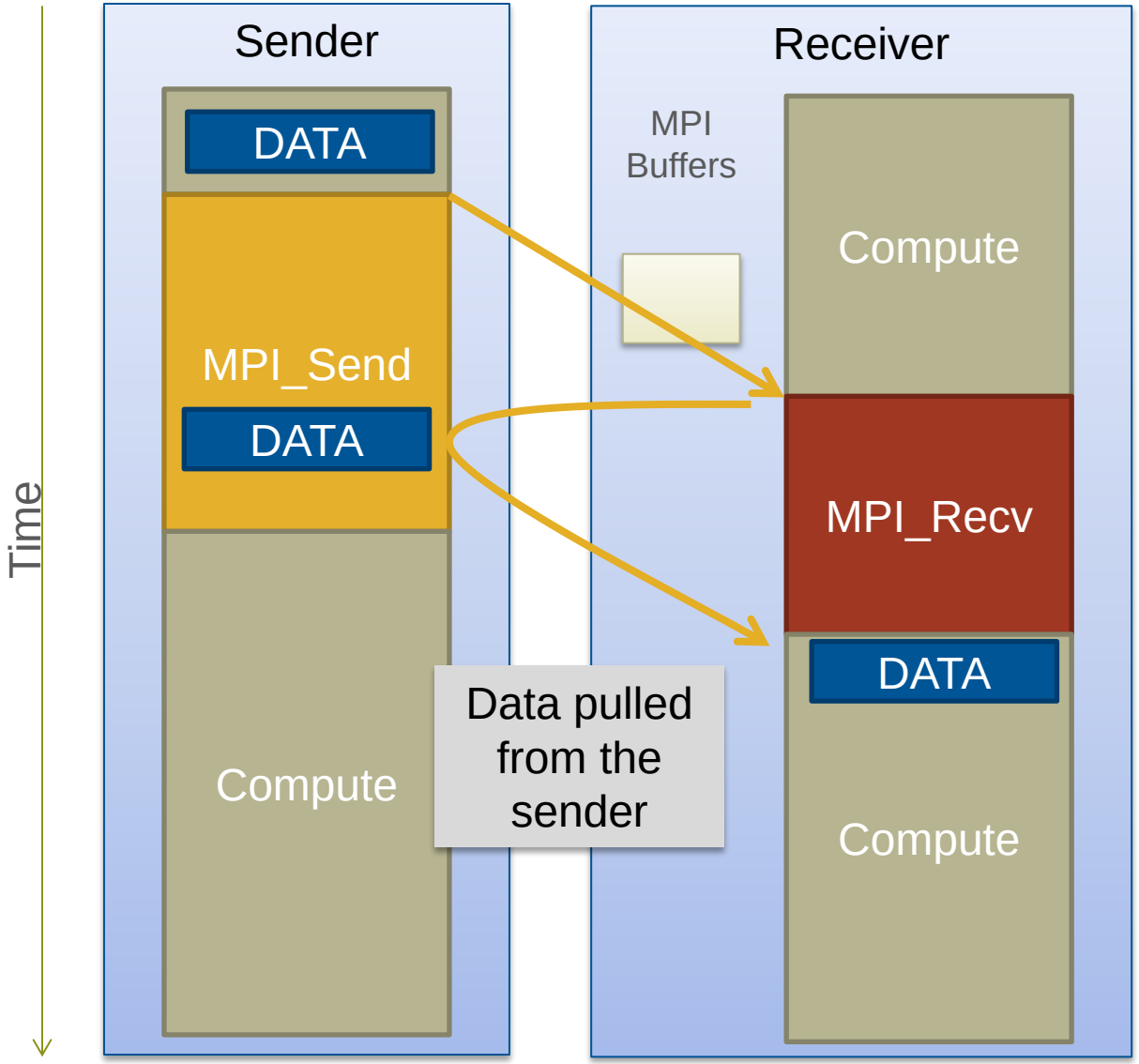


Data is pushed into an empty buffer(s) on the remote processor.

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

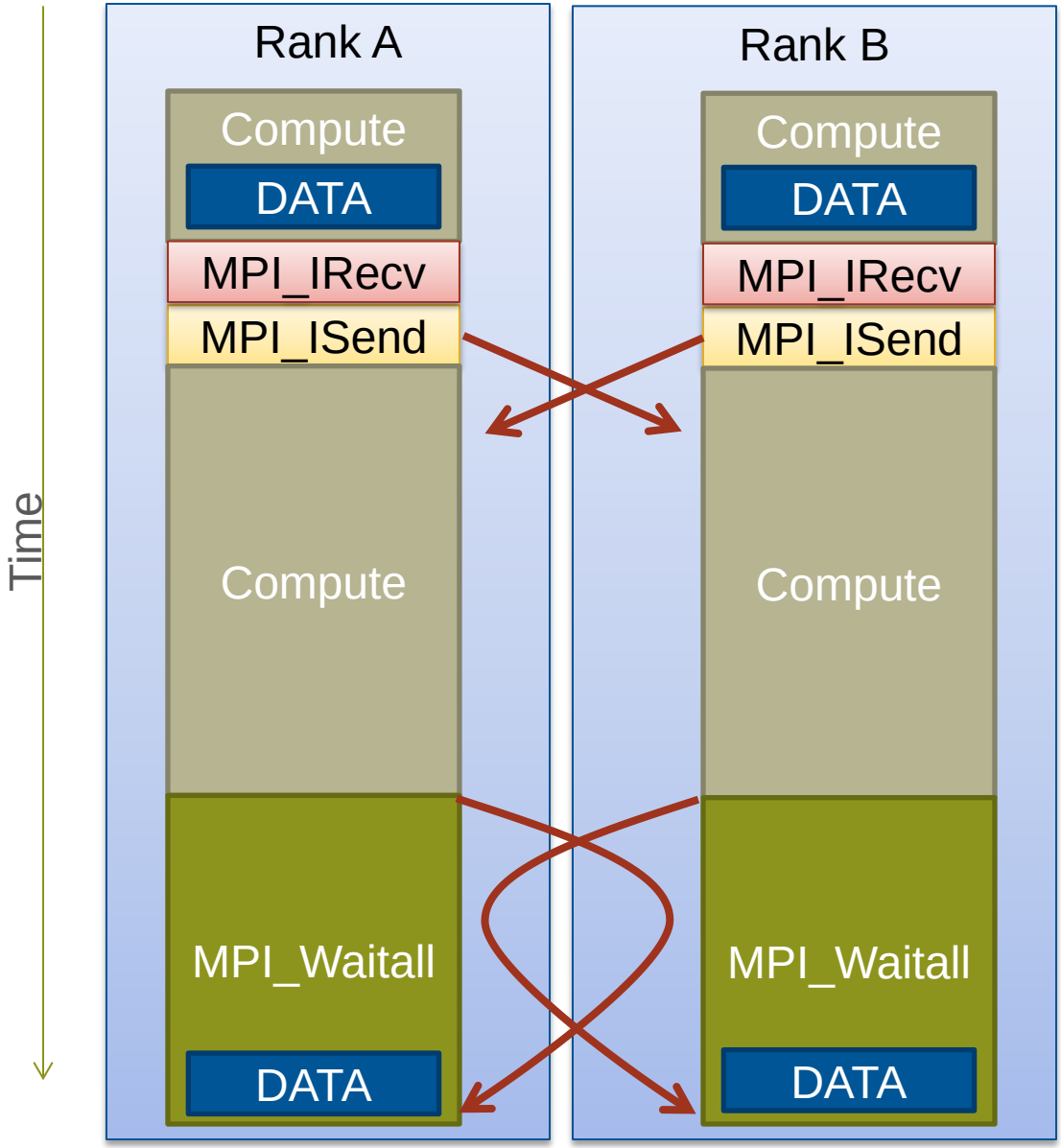
Messages cannot begin transfer until MPI_Recv called by the receiver.

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.

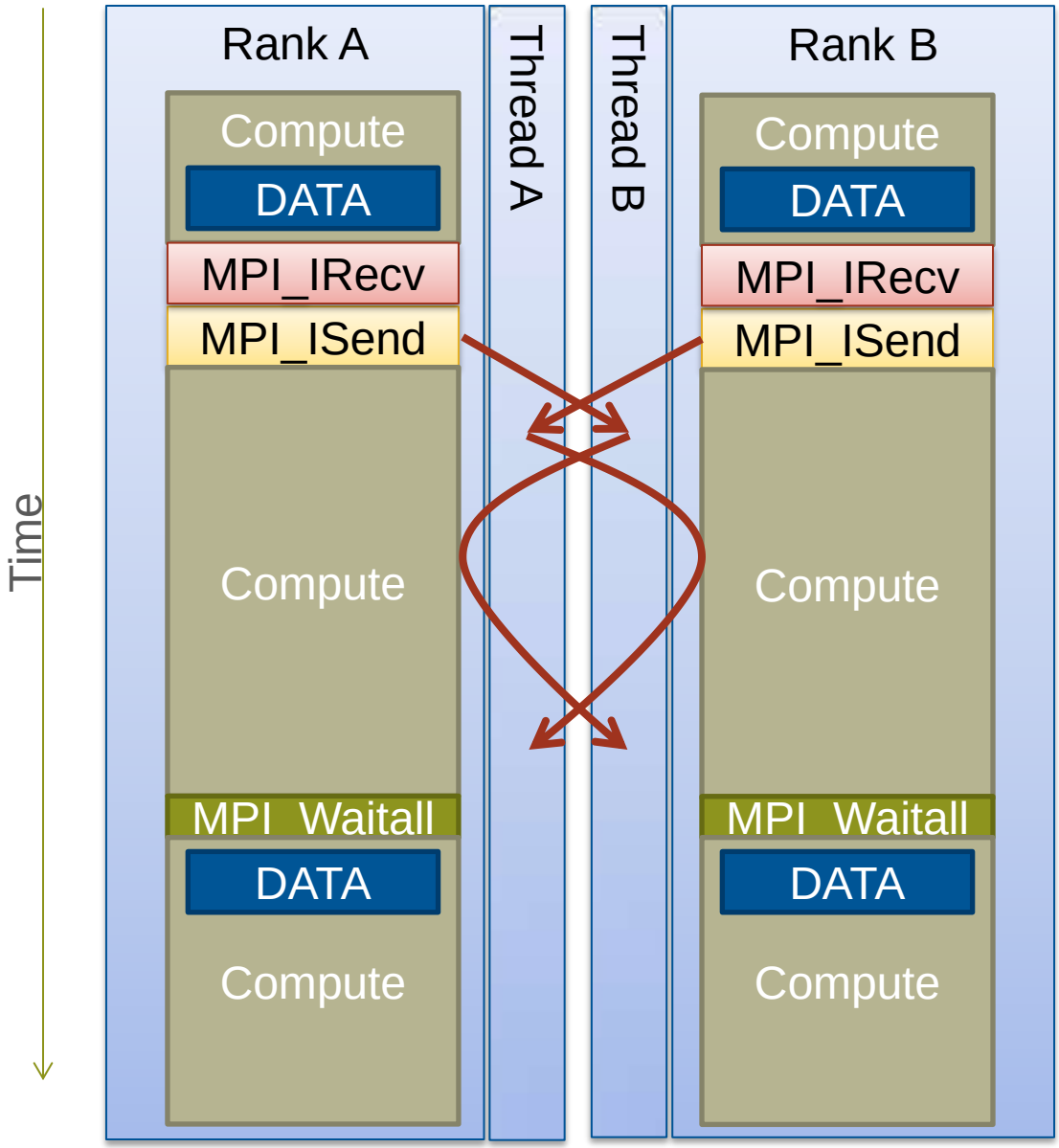
Making more messages EAGER

- 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.

Avoiding OS Noise

- **The kernel on the compute nodes is designed to be low noise and high throughput**
  - Only absolutely necessary daemons running on the nodes
  - Fewer interrupts and context switches means higher occupancy.
- **Inevitably, some kernel processes and user daemons will interrupt user processes**
  - Handling hardware interrupts
  - Flushing I/O buffers
  - Node health heart beats
- **These are not synchronised across the nodes.**
  - Effectively occurring randomly on each node
  - Can have a cumulative amplifying effect if applications synchronise at high frequency.

Core Specialisation

- **Unavoidable OS noise can potentially degrade application scaling performance if it synchronises frequently.**
  - Usually only visible at the largest scales
- **Solution, under-populate nodes slightly and move OS processes to unoccupied cores.**
  - Technique called “Core Specialisation” by Cray
- **Requires additional arguments in aprun command line:**
  - Add “-r <num cores>” to aprun command line
  - Total of –N and –r must not exceed available cores per node (32 for HECToR)

  E.g. `aprun -n 1024 -N 30 -r 2 <exe>`
- **May potentially require more nodes or fewer total ranks but overall performance improved**

Core Specialisation + Asynchronous Progress Engines

- **Problem: MPI requires heavyweight message matching**
  - Task of matching messages with tags, order and processes on Gemini is performed by Opteron processor.
  - Positive: Message matching done by fast CPU, also scales with number of MPI ranks (e.g. not a single engine per node like Seastar).
  - Negatives: CPU has to be in an MPI library for calls to progress. Limited opportunities for Comms/Compute overlap

- **Solution: Spawn an additional thread on CPU to do message matching.**
  - Allows message matching and transfer of RENDEZVOUS messages while main compute thread still working.
  - Much greater opportunity for Comms and Compute to overlap.

  - On Cray XE6 requires at least one dedicated core per node (via core specialisation). (Cray XC30 can use Intel Hyperthreads with almost no overhead).

Asynchronous Progress Engines

- Only useful if code using asynchronous MPI calls (includes MPI-3 async collectives).
- Requires library to use “Thread Multiple” safety.

```
export MPICH_NEMESIS_ASYNC_PROGRESS=1
export MPICH_MAX_THREAD_SAFETY=multiple
export MPICH_GNI_USE_UNASSIGNED_CPUS=enabled
```

Run application: `aprun -r 1 -N 31 -n XX  a.out`

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 XE

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