FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



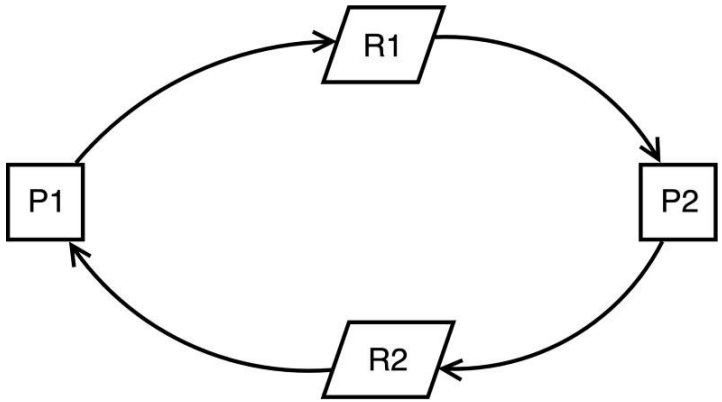
Lecture 4.
Non-Blocking Communication

Dr. Bazargul Matkerim, Professor-Researcher

Outline

1. **Deadlock**
   why blocking communication can stop the program
2. **Non-blocking send and receive**
   initiate, do other work, wait
3. **Request handles and completion**
   MPI_Wait, MPI_Test, multiple requests
4. **Exercise: rotating information around a ring**

   non-blocking ring with MPI_Issend



Source: Process deadlock, commons.wikimedia.org (FAL)

Deadlock

Code in each MPI process:

MPI_Ssend(…, right_rank, …)

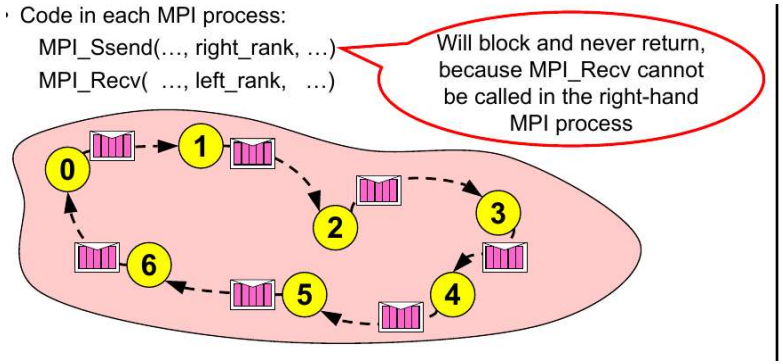
MPI_Recv(…, left_rank, …)

Will **block and never return**

MPI_Recv cannot be called in the right-hand process

Same problem with standard send MPI_Send

if MPI chooses the synchronous protocol

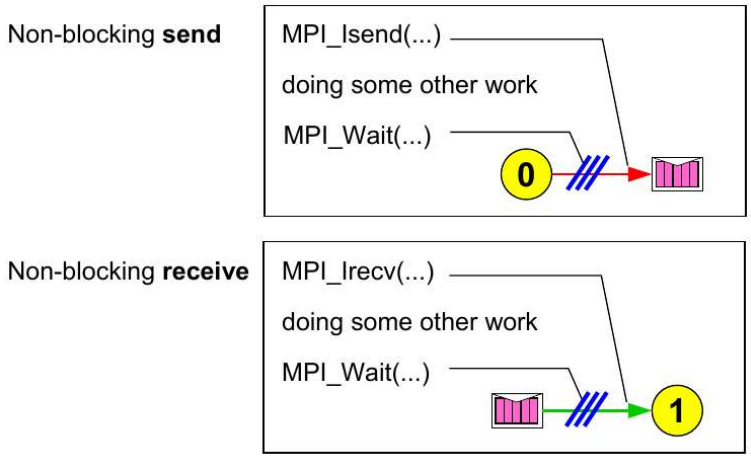


Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 63

Non-Blocking Communications

Three phases:

1. **Initiate** non-blocking communication
   returns immediately
   routine name starts with MPI_I…
2. **Do some work**
   perhaps involving other communications
3. **Wait** for the communication to complete



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 65

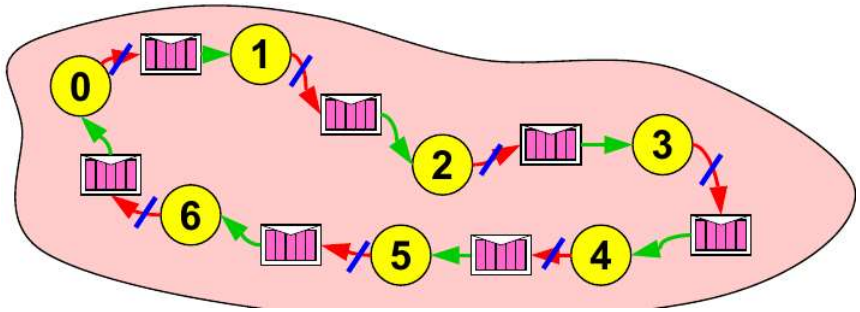
Non-Blocking Send

Initiate non-blocking send
    ring: send to the right neighbour

Do some work
    ring: receive from the left neighbour

Now the message transfer can be completed

Wait for the non-blocking send to complete



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 66

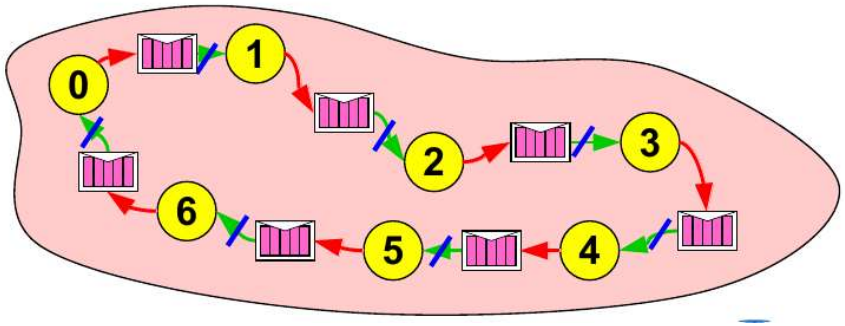
Non-Blocking Receive

Initiate non-blocking receive
    ring: receive from the left neighbour

Do some work
    ring: send to the right neighbour

Now the message transfer can be completed

Wait for the non-blocking receive to complete



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 67

Request Handles

Used for **non-blocking** communication

Must be stored in local variables:

- C: MPI_Request
- Fortran: INTEGER

The value:

- is generated by a non-blocking routine
- is used (and freed) in MPI_Wait

Other handles: communicator MPI_Comm, datatype MPI_Datatype

Example in C

```
MPI_Request request;
MPI_Status  status;

MPI_Isend(buf, n, MPI_INT, dest, tag,
          MPI_COMM_WORLD, &request);
/* ... other work ... */
MPI_Wait(&request, &status);
```

Non-Blocking Synchronous Send and Receive

Send

```
MPI_Issend(buf, count, datatype, dest, tag,
           comm, &request_handle);
MPI_Wait(&request_handle, &status);
```

Receive

```
MPI_Irecv(buf, count, datatype, source, tag,
          comm, &request_handle);
MPI_Wait(&request_handle, &status);
```

buf must **not be used** between Issend/Irecv and Wait

Issend + Wait directly after = **blocking Ssend**

status is not used in Issend, only in Wait

Remember

Every non-blocking operation needs a matching MPI_Wait or MPI_Test.

Completion

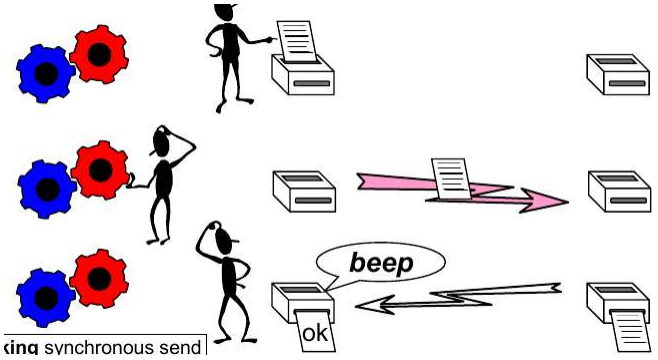
```
MPI_Wait(&request_handle, &status);
MPI_Test(&request_handle, &flag, &status);
```

One must:

- call WAIT, or
- loop with TEST until flag == 1

Several requests:

- one of them: MPI_Waitany / MPI_Testany
- all of them: MPI_Waitall / MPI_Testall
- some of them: MPI_Waitsome / MPI_Testsome



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 22

Blocking and Non-Blocking

Send and receive can be blocking or non-blocking

A blocking send can be used with a non-blocking receive, and vice versa

Synchronous mode affects **completion** (Wait/Test), not initiation

Non-blocking + matching Wait = **blocking operation**

Mode	Non-blocking routine
standard	MPI_Isend
synchronous	MPI_Issend
buffered	MPI_Ibsend
ready	MPI_Irsend

Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 74

Exercise: Rotating Information around a Ring

Processes are arranged in a ring

Each process stores its rank in snd_buf

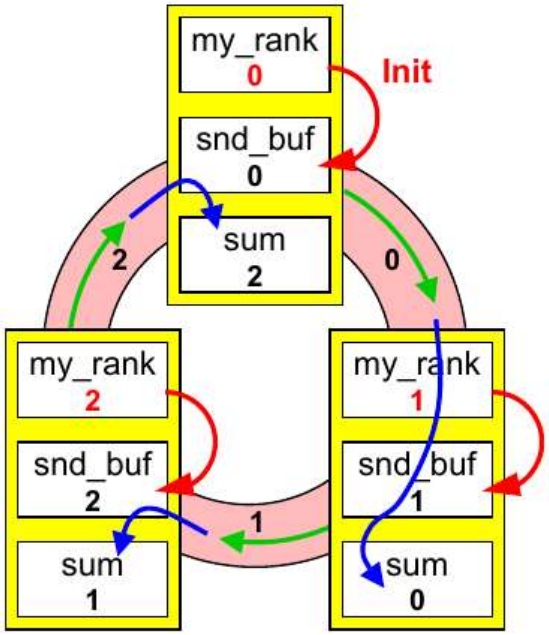
Passes it to the right neighbour and adds what it receives

Use non-blocking MPI_Issend
- to avoid deadlocks
- to verify the correctness

$$dest = (my_rank + 1)\ mod size$$

$$source = (my_rank - 1 + size)\ mod size$$

$$sum = \sum_{i=0}^{size-1} i = \frac{size(size - 1)}{2}$$



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 77

Pitfalls with Non-Blocking Calls

Register optimization (Fortran)
- compiler may keep buf in a register
- old data is written instead of received data
- workaround: MPI_ADDRESS(buf, ...) after MPI_WAIT

Strided sub-arrays
- copied to a temporary array, released on return
- do not use non-contiguous sub-arrays in non-blocking calls

```
MPI_IRECV(buf, ..., request_handle, ierror)
MPI_WAIT(request_handle, status, ierror)
write (*,*) buf   ! may print old data
```

Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 72