FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 3.
Messages and
Point-to-Point Communication

Dr. Bazargul Matkerim, Professor-Researcher

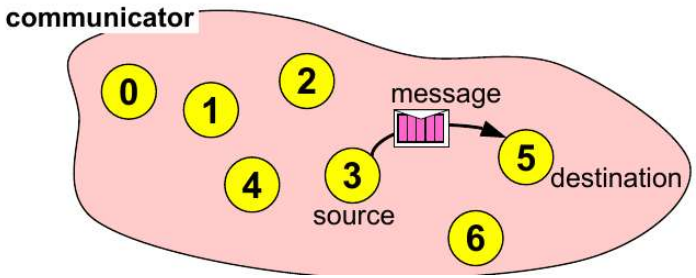
Outline

1. **Messages and datatypes**
   elements, basic MPI datatypes

2. **Sending and receiving a message**
   MPI_Send, MPI_Recv, envelope, wildcards

3. **Communication modes**
   synchronous, buffered, standard, ready

4. **Message order and ping pong**
   order preservation, latency, bandwidth



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 48

Messages

A message contains a number of **elements** of some datatype

MPI datatypes:

- basic datatypes
- derived datatypes (built from basic ones)

C types are different from Fortran types

Datatype **handles** describe the type of data in memory



Example: message with 5 integers. Source: R. Rabenseifner, HLRS, slide 45

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED	unsigned int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI basic datatypes in C. Source: R. Rabenseifner, HLRS, slide 46

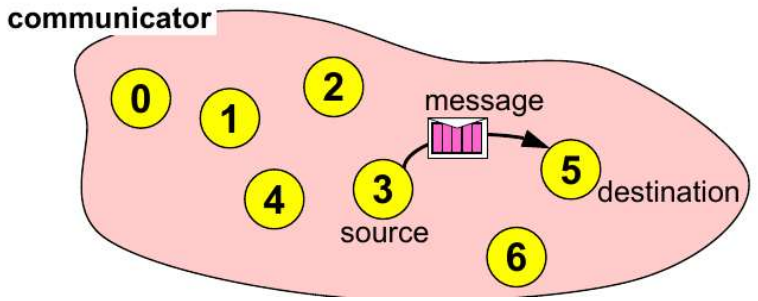
Point-to-Point Communication

Communication between **two** processes

Source process sends a message to **destination** process

Takes place within a **communicator**
e.g. MPI_COMM_WORLD

Processes are identified by their **ranks**



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 48

Sending a Message

```
int MPI_Send(void *buf, int count,
             MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)
```

buf: starting point of the message with count elements
each element described with datatype

dest: **rank** of the destination process in comm

tag: additional non-negative integer
used to distinguish different types of messages

Example

```
int x[5];
/* send 5 integers
   to rank 1, tag 17 */
MPI_Send(x, 5, MPI_INT, 1, 17,
         MPI_COMM_WORLD);
```

Receiving a Message

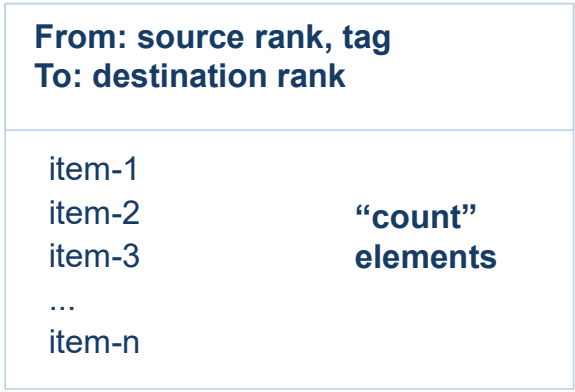
```
int MPI_Recv(void *buf, int count,
             MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status)
```

buf/count/datatype describe the **receive buffer**

Receives the message from rank source in comm

Only messages with **matching tag** are received

Envelope information is returned in status
- status.MPI_SOURCE, status.MPI_TAG
- count via MPI_Get_count()



Communication envelope (after Lecture 5 material, slide 52)

Requirements and Wildcarding

For a communication to succeed:

Sender specifies a valid destination rank

Receiver specifies a valid source rank

The **communicator** must be the same

Tags must match

Message **datatypes** must match

Receiver's buffer must be **large enough**

Wildcarding (receiver only)

Any source: MPI_ANY_SOURCE

Any tag: MPI_ANY_TAG

Actual source and tag are in status

Example

```
MPI_Status status;
MPI_Recv(buf, n, MPI_INT,
         MPI_ANY_SOURCE, MPI_ANY_TAG,
         MPI_COMM_WORLD, &status);
printf("from %d, tag %d\n",
  status.MPI_SOURCE, status.MPI_TAG);
```

Communication Modes

Synchronous: MPI_Ssend

Buffered: MPI_Bsend

Standard: MPI_Send

Ready: MPI_Rsend

Receive (all modes): MPI_Recv



Sender mode	Definition	Notes
Synchronous send **MPI_SSEND**	Only completes when the receive has started	
Buffered send **MPI_BSEND**	Always completes (unless an error occurs), irrespective of receiver	needs application-defined buffer to be declared with MPI_BUFFER_ATTACH
Standard send **MPI_SEND**	Either synchronous or buffered	uses an internal buffer
Ready send **MPI_RSEND**	May be started **only** if the matching receive is already posted!	highly dangerous!
Receive **MPI_RECV**	Completes when a message has arrived	same routine for all communication modes

Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 56

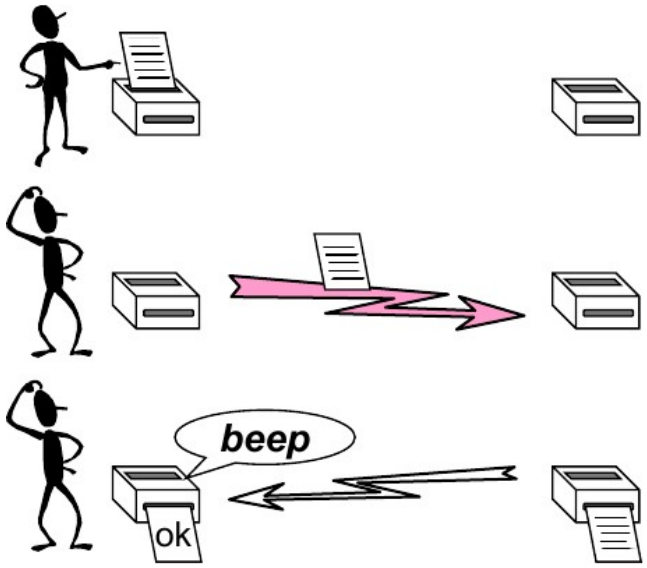
Rules for the Communication Modes

Standard send (MPI_Send)
- minimal transfer time
- may block due to synchronous mode

Synchronous send (MPI_Ssend)
- risk of deadlock, serialization and waiting (idle time)
- high latency, best bandwidth

Buffered send (MPI_Bsend)
- low latency, bad bandwidth

Ready send (MPI_Rsend): use never, unless 200 % sure



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 19

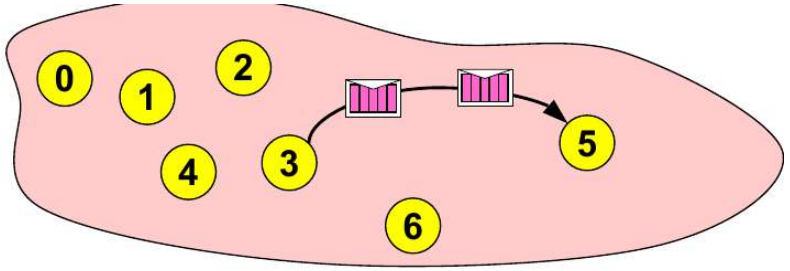
Message Order Preservation

Rule for messages on the same connection:
- same communicator, source and destination rank

Messages **do not overtake** each other

True even for non-synchronous sends

If both receives match both messages, the order is preserved



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 58

Exercise: Ping Pong

Process 0 sends to process 1 (**ping**)

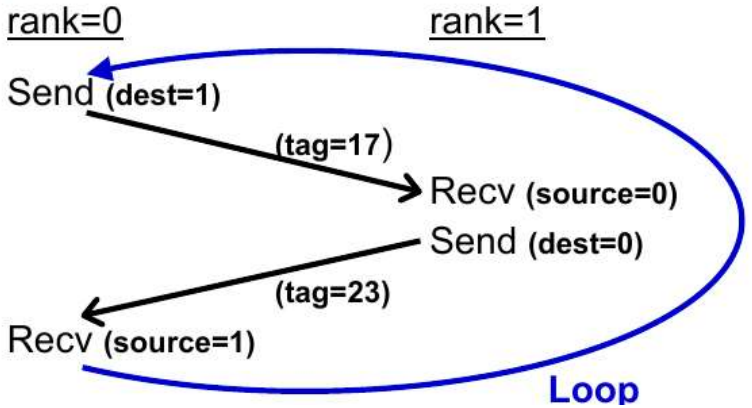
Process 1 sends back to process 0 (**pong**)

Repeat in a loop of length 50

Timing with MPI_Wtime()

```
if (my_rank == 0) {
  MPI_Send(... dest=1 ...);
  MPI_Recv(... source=1 ...);
} else {
  MPI_Recv(... source=0 ...);
  MPI_Send(... dest=0 ...);
}
```

$$t_{msg} = \frac{t_{end} - t_{start}}{2 \cdot 50}$$



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 60

Latency and Bandwidth

Definitions

Latency = transfer time for short messages

$$bandwidth = \frac{messagesize\ (bytes)}{transfertime}$$

Measure for:

- standard send (MPI_Send)
- synchronous send (MPI_Ssend)

Message sizes:

- 8 B, 512 B, 32 kB, 2 MB

Message size	Meaning
8 B	one double value
512 B	8 · 64 bytes
32 kB	8 · 64² bytes
2 MB	8 · 64³ bytes

Message sizes of the advanced exercise. Source: R. Rabenseifner, HLRS, slide 61