IRELAND’S ADVANCED COMPUTING CENTRE

MPI Point-to-Point Communication

ICHEC

Irish Centre for High-End Computing

An Roinn Post, Fiontar agus Nuálaíochta
Department of Jobs, Enterprise and Innovation

sfi
science foundation ireland
fondúireacht eolaíochta éireann

An Roinn
Oideachais agus Scileanna
Department of
Education and Skills

HEA
Higher Education Authority
An tÚdarás um Ard-Oideachas

www.ichec.ie

Messages

- A message contains a number of elements of some particular datatype.
- MPI datatypes:
  - Basic datatype.
  - Derived datatypes

- C types are different from Fortran types.
- Datatype handles are used to describe the type of the data in the memory.

Example: message with 5 integers

2345	654	96574	-12	7676

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

int arr[5]
count =5
datatype=MPI_INT

2345	654	96574	-12	7676

MPI Datatype	Fortran datatype
MPI_INTEGER	integer
MPI_INTEGERX	integer*X X=1,2,4,8
MPI_REAL	real
MPI_REALX	real*X
MPI_DOUBLE_PRECISION	double precision
MPI_COMPLEX	complex
MPI_COMPLEXY	complex*Y
MPI_ LOGICAL	logical
MPI_CHARACTER	character(1)
MPI_BYTE	
MPI_PACKED	

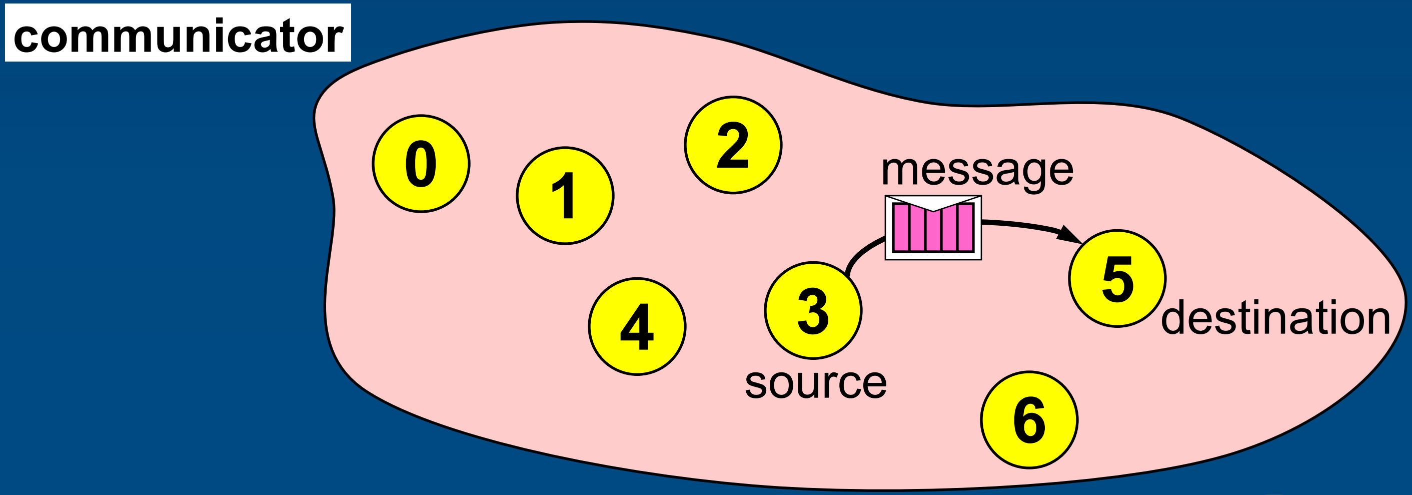
2345	**654**	**96574**	**-12**	**7676**

count=5
datatype=MPI_INTEGER

integer :: arr(5)

Point-to-Point Communication

- Communication between two processes.
- Source process sends message to destination process.
- Communication takes place within a communicator, e.g., MPI_COMM_WORLD.
- Processes are identified by their ranks in the communicator.



Sending a Message

- C: int MPI_Send(void *buf, int count, MPI_Datatype datatypeHandle,
  int dest, int tag, MPI_Comm comm)

- Fortran: MPI_Send(buf, count, datatypeHandle, dest, tag, comm, *ierror*)
  <type> buf(:)
  integer :: count, datatypeHandle, dest, tag, comm, ierror

- buf is the starting point of the message with count elements, each described with datatypeHandle.
- dest is the rank of the destination process within the communicator comm.
- tag is an additional nonnegative integer piggyback information, additionally transferred with the message.
- The tag can be used by the program to distinguish different types of messages.

Receiving a Message

- C: int MPI_Recv(void **buf*, int count, MPI_Datatype datatypeHandle,
  int source, int tag, MPI_Comm comm,
  MPI_Status **status*)

- Fortran: MPI_Recv(*buf*, count,datatypeHandle, source, tag,
  comm, *status*, *ierror*)
  <type> buf(:)
  integer :: count, datatype, source, tag, comm
  integer :: status(MPI_STATUS_SIZE), ierror

- buf/count/datatypeHandle describe the receive buffer.
- Receiving the message sent by process with rank source in comm.
- Envelope information is returned in *status*.
- Output arguments are printed *green-cursive*.
- Only messages with matching tag are received.

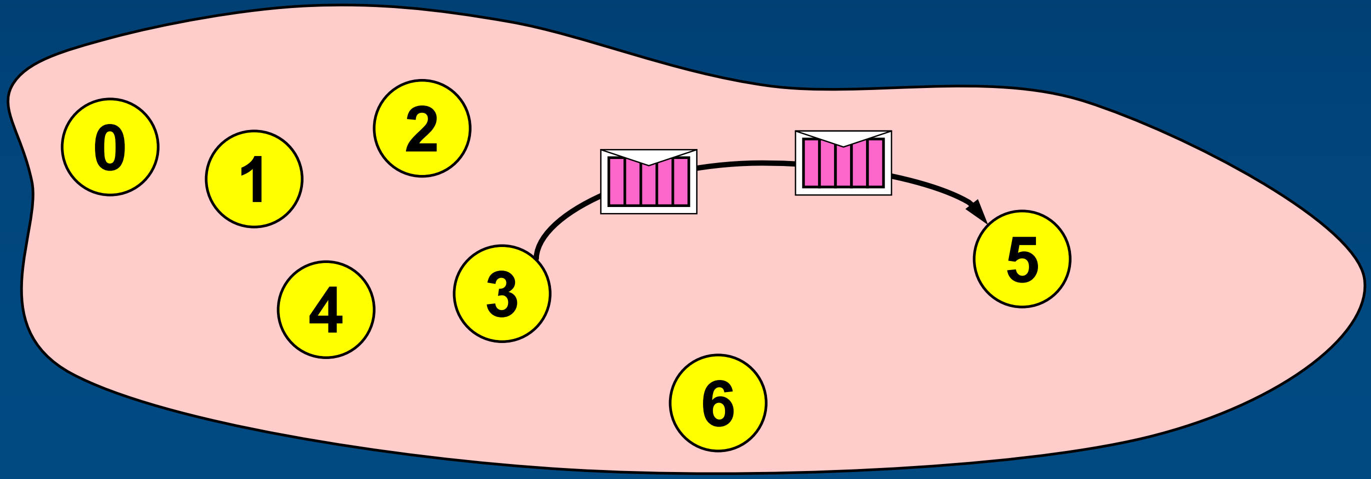
Requirements for Point-to-Point Communications

For a communication to succeed:

- Sender must specify a valid destination rank.
- Receiver must specify a valid source rank.
- The communicator must be the same.
- Tags must match.
- Message datatypes must match.
- Receiver's buffer must be large enough.

Message Order Preservation

- Rule for messages on the same connection,
  i.e., same communicator, source, and destination rank:
- **Messages do not overtake each other.**
- This is true even for non-synchronous sends.



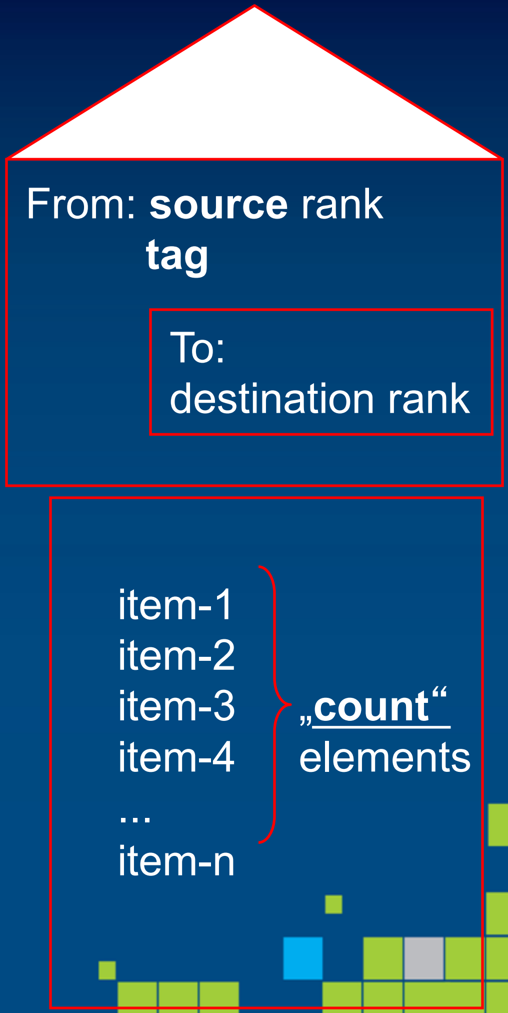
- If both receives match both messages, then the order is preserved.

Communication Envelope

- Envelope information is returned from MPI_RECV in *status*.

- C: status.MPI_SOURCE
  status.MPI_TAG
  count via MPI_Get_count()

- Fortran: status(MPI_SOURCE)
  status(MPI_TAG)
  count via MPI_Get_count()



From: **source** rank
tag

To:
destination rank

item-1
item-2
item-3
item-4
...
item-n

„**count**“
elements

Wildcards

- Receiver can wildcard.
- To receive from any source — source = MPI_ANY_SOURCE
- To receive from any tag — tag = MPI_ANY_TAG
- Actual source and tag are returned in the receiver's *status* parameter.

Communication Modes

- Send communication modes:

○ synchronous send	MPI_**S**send
○ buffered [asynchronous] send	MPI_**B**send
○ standard send	MPI_**Send**
○ Ready send	MPI_**R**send

- Receiving all modes MPI_**Recv**

Communication Modes — Definitions

Sender modes	Definition	Notes
Synchronous send MPI_Ssend	Only completes when the receive has started	risk of deadlock risk of serialization risk of waiting —> idle time
Buffered send MPI_Bsend	Always completes (unless an error occurs), irrespective of receiver	needs application-defined buffer to be declared with MPI_BUFFER_ATTACH/DETACH
Standard send MPI_Send	Standard send. Either synchronous or buffered	
Ready send MPI_Rsend	May be started only if the matching receive is already posted!	highly dangerous!
Receive MPI_Recv	Completes when a the message (data) has arrived	

C

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv){
   int myRank, uniSize, ierror, arr[5];
   MPI_Status stat;

   ierror=MPI_Init(&argc,&argv);
   ierror=MPI_Comm_rank(MPI_COMM_WORLD,&myRank);
   ierror=MPI_Comm_Size(MPI_COMM_WORLD,&uniSize);
   if (myRank == 0) {
      arr[0]=2345; arr[1]=654; arr[2]=96574; arr[3]=-12; arr[4]=7676;
      MPI_Send(arr, 5, MPI_INT, 1, 100, MPI_COMM_WORLD);
   } else if (myRank == 1) {
      MPI_Recv(arr, 5, MPI_INT, 0, 100, MPI_COMM_WORLD, &stat);
   }
   ierror=MPI_Finalize();
   return 0;
}
```

Fortran

```
program testMPI
use mpi
implicit none
integer :: myRank,uniSize,ierror
real, dimension(5) :: arr
integer stat(MPI_STATUS_SIZE)

call MPI_Init(ierror)
call MPI_Comm_rank(MPI_COMM_WORLD,myRank,ierror)
call MPI_Comm_Size(MPI_COMM_WORLD,uniSize,ierror)
if (myRank .eq. 0) then
    arr= (/2345, 654, 96574, -12, 7676/)
    call MPI_Send(arr, 5, MPI_INTEGER, 1, 100, MPI_COMM_WORLD,ierror)
else if (myRank .eq. 1) then
    call MPI_Recv(arr, 5, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, stat,ierror)
endif
call MPI_Finalize(ierror)
end program testMPI
```