

# Non-Blocking Communication

- Non-Blocking communications
- Non-Blocking send
- Non-Blocking receive
- Handles
- Non-Blocking synchronous send
- Non-Blocking receive
- Blocking and non-blocking
- Routine names
- Completion tests
- Wait/Test routines
- Multiple communications
- Testing multiple non-blocking communications
- Class Exercise: Calculating Ring



# Non-Blocking communications

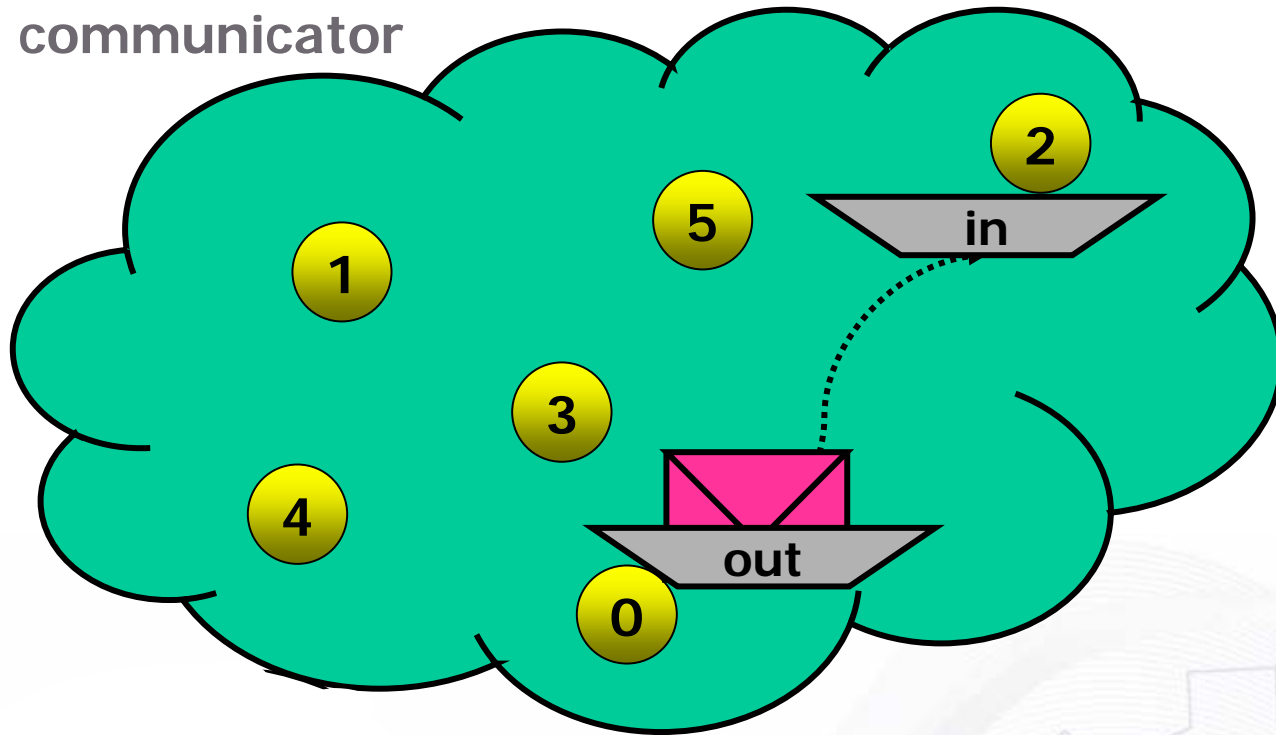
Separate communication into three phases:

1. Initiate non-blocking communication (“post” a send or receive)
2. Do some other work not involving the data in transfer
  - Overlap calculation and communication
  - Latency hiding
3. Wait for non-blocking communication to complete

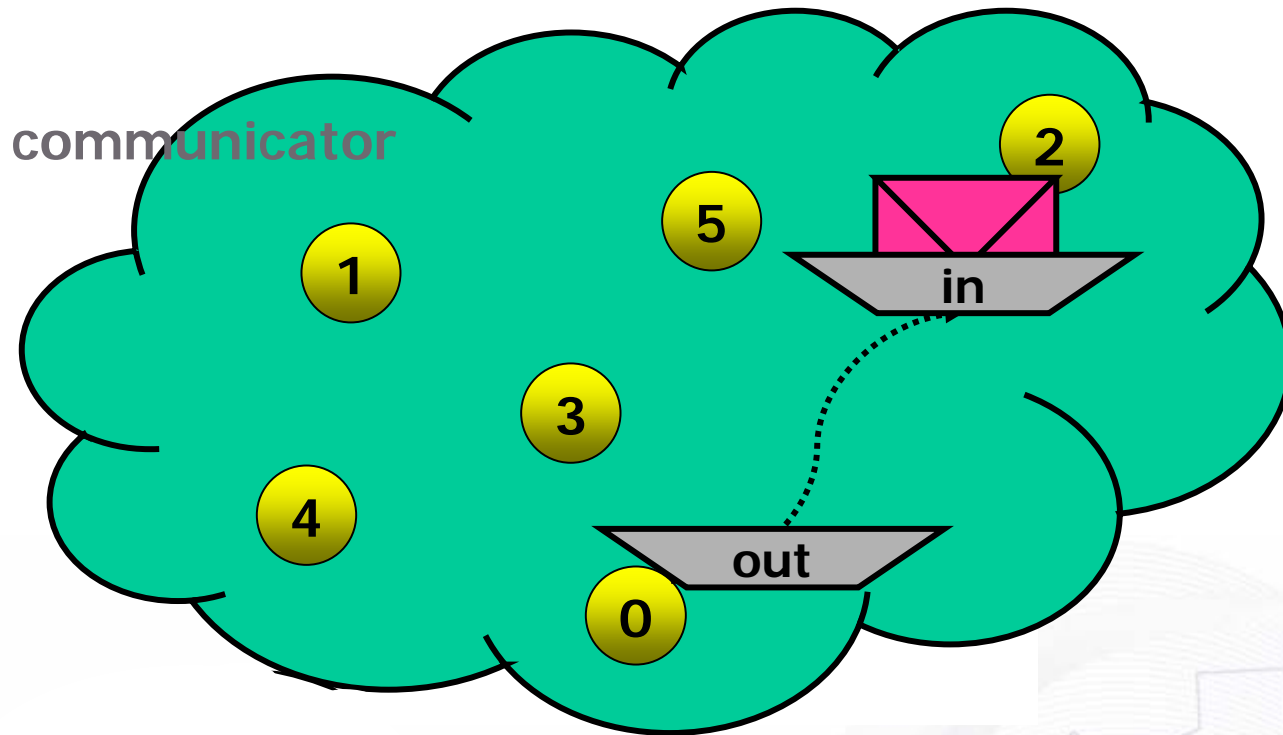


# Non-Blocking send

communicator



# Non-Blocking receive



## Handles used for non-blocking communication

Datatype	Same as for blocking (MPI_Datatype or INTEGER)
Communicator	Same as for blocking (MPI_Comm or INTEGER)
Request	MPI_Request or INTEGER

- A request handle is allocated when a non-blocking communication is initiated
- The request handle is used for testing if a specific communication has completed



# Non-Blocking synchronous send

**C:**

```
int MPI_Issend(void *buf, int count, MPI_Datatype datatype,  
              int dest, int tag, MPI_Comm comm, MPI_Request *request)
```

**Fortran:**

```
CALL  
  MPI_ISEND( BUF , COUNT , DATATYPE , DEST , TAG , COMM , REQUEST , IERROR )
```

```
<type> BUF( * )  
INTEGER COUNT , DATATYPE , DEST , TAG , COMM  
INTEGER REQUEST , IERROR
```

“\_I” for “Immediate” because they return more or less immediately



# Non-Blocking receive

**C:**

```
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype,  
             int source, int tag, MPI_Comm comm, MPI_Request  
             *request)
```

**Fortran:**

```
CALL  
  MPI_IRecv(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IER  
  ROR)
```

```
<type> BUF(*)  
INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM  
INTEGER REQUEST, IERROR
```

**system-defined, not  
directly accessible by  
programmer**

Note: no STATUS argument



# 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, e.g.,

MPI\_Isend  MPI\_Recv

- Non-blocking sends can use any mode - synchronous, buffered, standard or ready

Note: there is no advantage for buffered or ready modes



# Routine names

<b>Non-Blocking Operation</b>	<b>MPI Call</b>
Standard send	MPI_ISEND
Synchronous send	MPI_ISSEND
Buffered send	MPI_IBSEND
Ready send	MPI_IRSEND
Receive	MPI_IRECV

“I” : Immediate



# Completion tests

- Waiting vs. Testing
- **Wait:** routine does not return until completion finished
- **Test:** routine returns a TRUE or FALSE (0) value depending on whether the communication has completed



# Wait/Test routines

**C:**

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)  
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
```

**Fortran:**

```
CALL MPI_WAIT(REQUEST, STATUS, IERR)  
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERR  
  
CALL MPI_TEST(REQUEST, FLAG, STATUS, IERR)  
LOGICAL FLAG  
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERR
```

Here is where `STATUS` appears. It is a three-element structure in C and an integer array of size `MPI_STATUS_SIZE` (defined in `mpif.h`) in Fortran.



# Multiple communications

- Test or wait for completion of one (and only one) message:
  - int MPI\_Waitany(...)
  - int MPI\_Testany(...)
- Test or wait for completion of all messages:
  - int MPI\_Waitall(...)
  - int MPI\_Testall(...)
- Test or wait for completion of as many messages as possible:
  - int MPI\_Waitsome(...)
  - int MPI\_Testsome(...)



# Class Exercise: Calculating Ring

- Repeat the Processor Ring exercise, this time using non-blocking communication routines
- In addition, each processor should calculate the sum of all the ranks as it receives them

