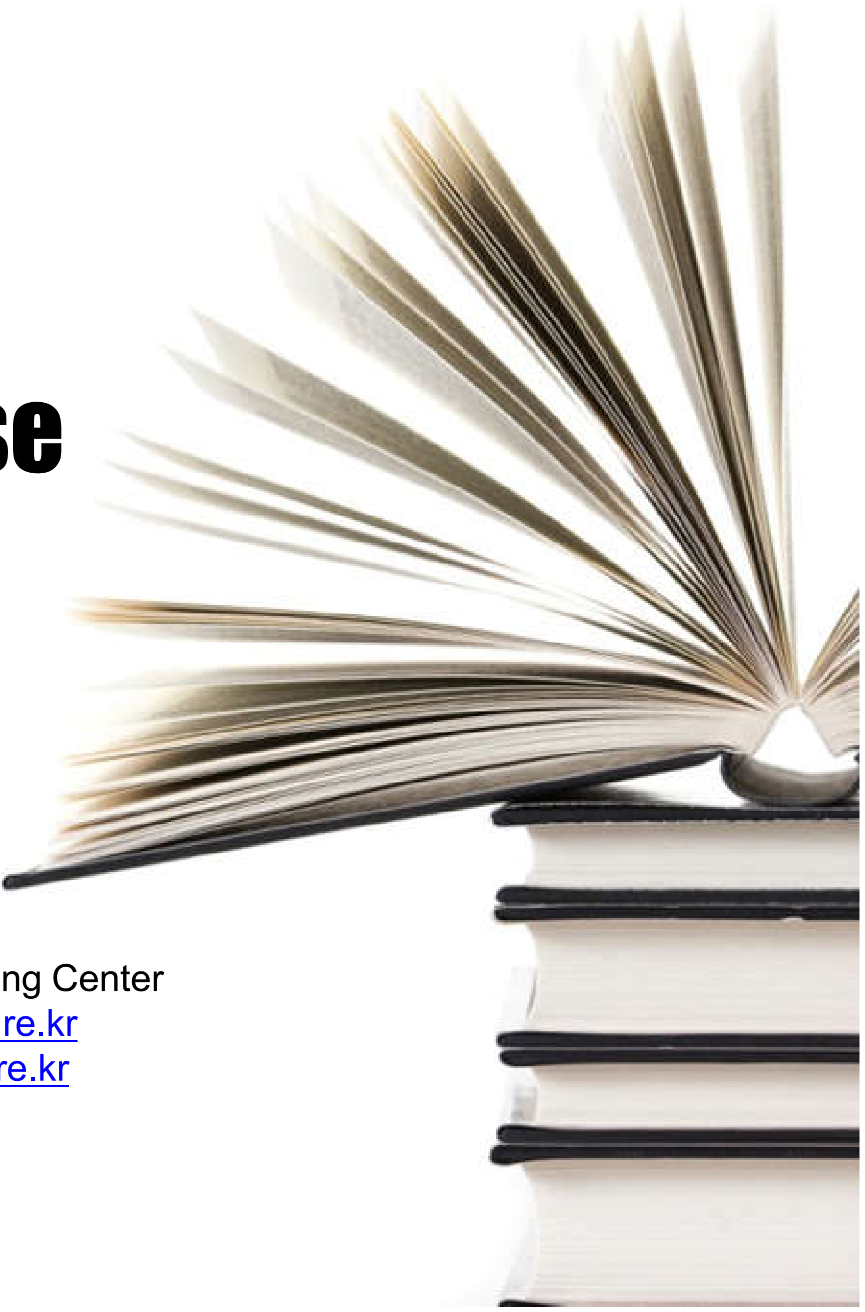
MPI Training Course in PNU



2016. Aug

KISTI Supercomputing Center
http://www.ksc.re.kr
http://edu.ksc.re.kr

Objectives

- **After successfully learning the tutorial in this module, you will be able to**

  - ✓ Understand parallel program concept
  - ✓ Compile and run MPI programs using the MPI implementation
  - ✓ Write MPI codes using the core library

Syllabus

09:30	-	10:30	▪ **MPI Introduction and Concepts**
10:30	-	10:40	▪ Break
10:40	-	12:00	▪ **P2P Communication**
12:00	-	13:00	▪ Lunch
13:00	-	14:20	▪ **Collective Communication**
14:20	-	14:30	▪ Break
14:30	-	15:50	▪ **Derived Data Type**
15:50	-	16:00	▪ Break
16:00	-	17:00	▪ How to Parallelize / Summary

Reference & Useful Site

1. MPI Tutorial

<MPI 온라인 강좌>

- **http://www.citutor.org/login.php**

<MPI Tutorial>

- **https://computing.llnl.gov/tutorials/mpi/**
- **http://mpitutorial.com/**

<Domain Decomposition 강좌>

- **http://www.nccs.nasa.gov/tutorials/mpi_tutorial2/mpi_II_tutorial.html**

<MPI 한글 레퍼런스>

- **http://incredible.egloos.com/3755171**

<SP Parallel Programming Workgroup>

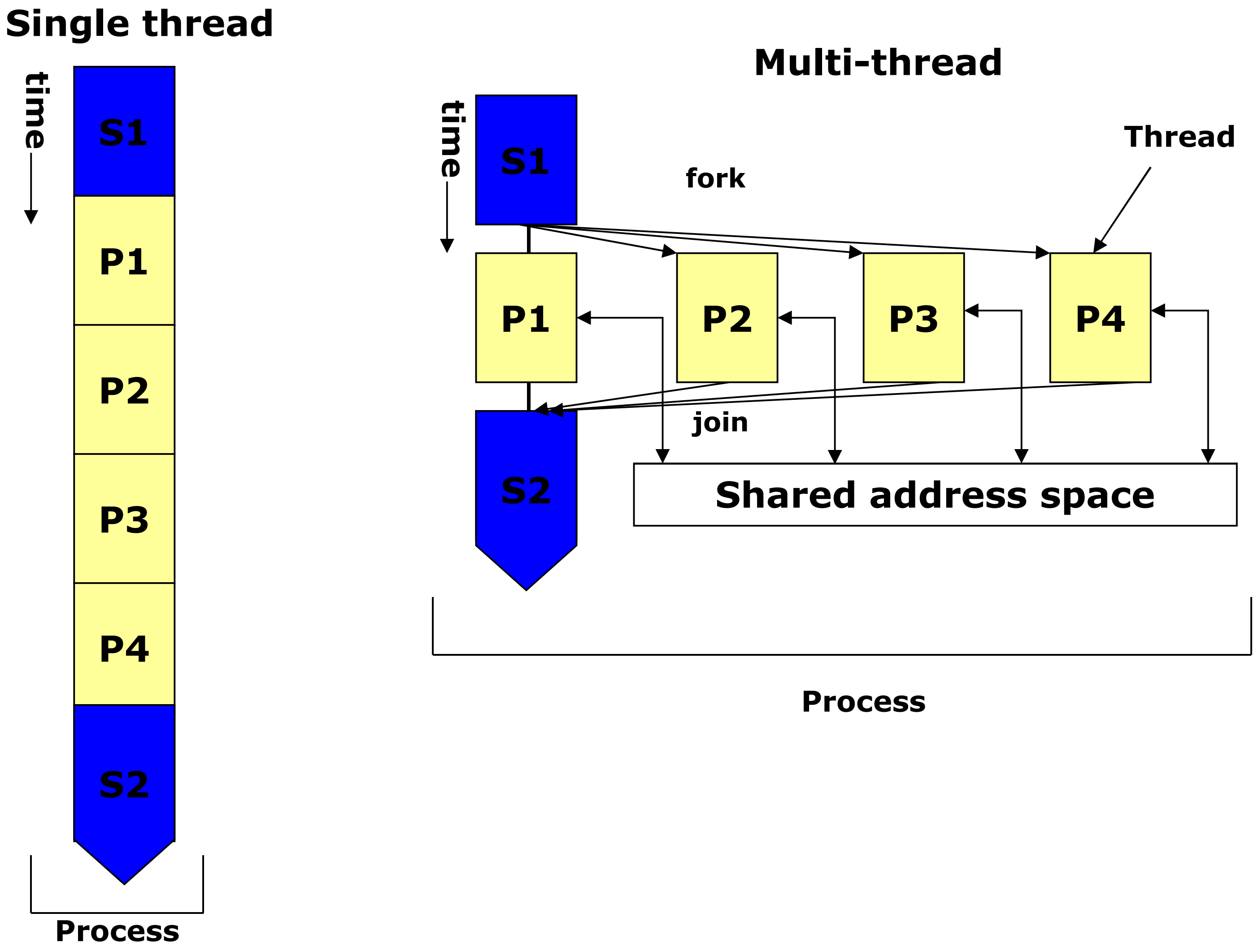
- ~~**http://www.itservices.hku.hk/sp2/workshop/html/samples/exercises.html**~~

<PDC Center for HPC>

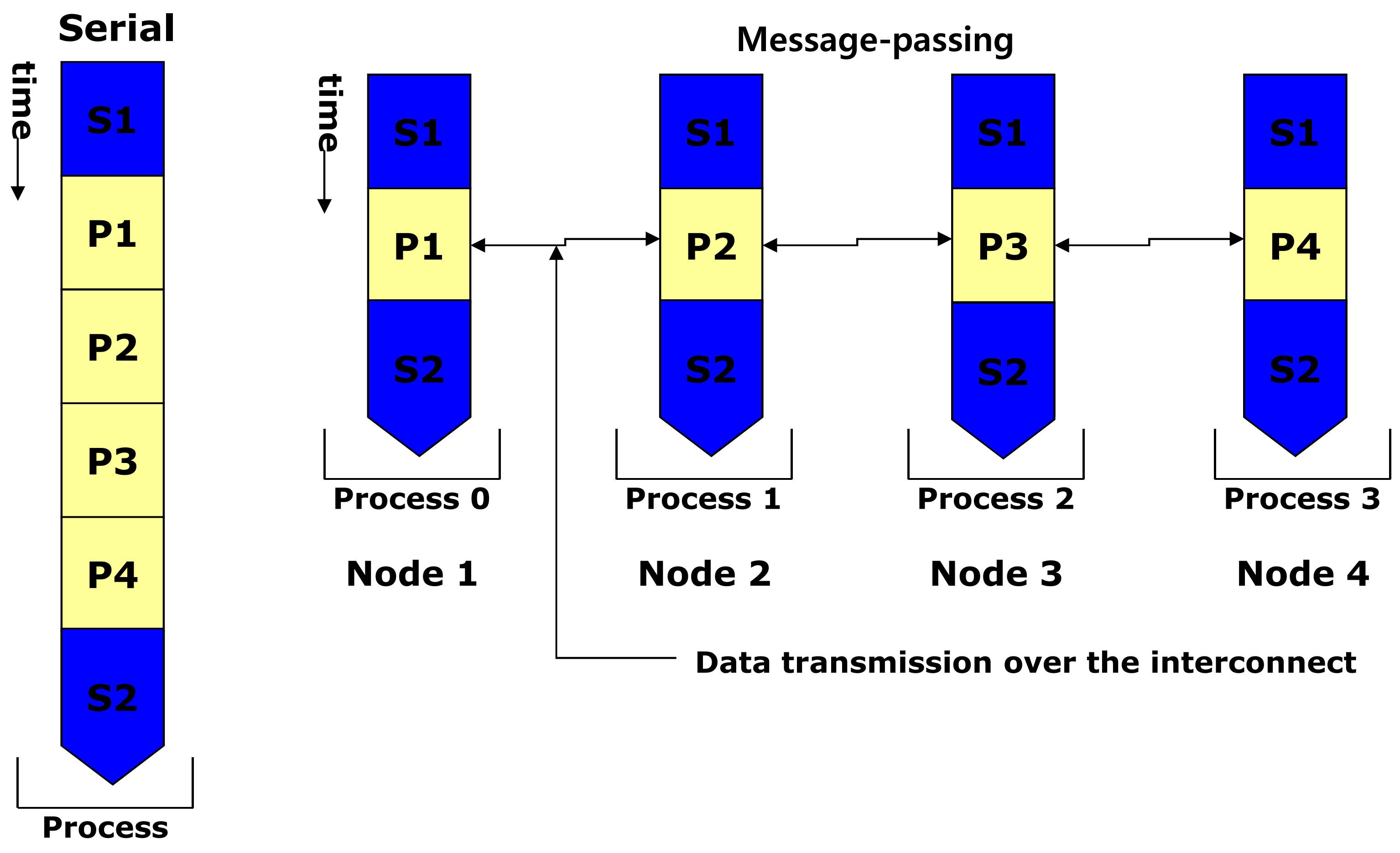
- **http://www.pdc.kth.se/education/tutorials/summer-school/mpi-exercises/mpi-labs/mpi-lab-1-program-structure-and-point-to-point-communication-in-mpi/mpi-lab-1-program-structure-and-point-to-point-communication-in-mpi**

MPI Introduction and Concepts

Shared Memory Programming Model



Message Passing Programming Model



What is MPI?

- **Message Passing Interface**
- **MPI is a library, not a language**
- **It is a library for inter-process communication and data exchange**
- **Use for Distributed Memory**

- **History**
  - MPI-1 Standard (MPI Forum) : 1994
    - http://www.mcs.anl.gov/mpi/index.html
    - MPI-1.1(1995), MPI-1.2(1997)
  - MPI-2 Announce : 1997
    - http://www.mpi-forum.org/docs/docs.html
    - MPI-2.1(2008), MPI-2.2(2009)
  - MPI-3 Announce : 2012
    - http://www.mpi-forum.org/docs/docs.html

Common MPI Implementations

- **MPICH(Argonne National Laboratory)**
  - Most common MPI implementation
  - Derivatives
    - MPICH GM – Myrinet support (available from Myricom)
    - MVAPICH – infiniband support (available from Ohio State University)
    - Intel MPI – Version tuned to Intel Architecture systems
- **Open MPI(Indiana University/LANL)**
  - Contains many MPI 2.0 features
  - FT-MPI: University of Tennessee (Data types, process fault tolerance, high performance)
  - LA-MPI: Los Alamos (Pt-2-Pt, data fault-tolerance, high performance, thread safety)
  - LAM/MPI: Indiana University (Component architecture, dynamic processes)
  - PACX-MPI: HLRS - Stuttgart (dynamic processes, distributed environments, collectives)
- **Scali MPI Connect**
  - Provides native support for most high-end interconnects
- **MPI/Pro (MPI Software Technology)**

Checking Environment

➢ **Making MPI hosts file**

– Use familiar editor : vi, emacs, gedit, etc…

```
$ cat  hosts
s0001
s0002
s0003
s0004
```

How To Run MPI

MPI Basic Steps

- **Writing a program**
  - using "mpi.h" and some essential function calls
- **Compiling your program**
  - using a compilation script
- **Specify the machine file**

"Hello, World" in MPI - C

- Demonstrates how to create, compile and run a simple MPI program on the lab cluster using the Intel MPI implementation

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

int main (int argc, char* argv[])
{
   /* Initialize the library */
   MPI_Init(&argc, &argv);

   printf(″Hello world\n″);

   /* Wrap it up. */
   MPI_Finalize();
}
```

Initialize MPI Library

Do some work!

Return the resources

```
$ mpicc -o hello.x hello.c
$ mpirun -np 4 -hostfile hosts ./hello.x
```

"Hello, World" in MPI - Fortran

- Demonstrates how to create, compile and run a simple MPI program on the lab cluster using the Intel MPI implementation

```
PROGRAM hello
INCLUDE 'mpif.h'

        INTEGER iErr

        CALL MPI_Init(iErr)

        WRITE (*, *) 'Hello, World'

        CALL MPI_Finalize(iErr)
END
```

Initialize MPI Library

Do some work!

Return the resources

```
$ mpif90 -o hello.x hello.f90
$ mpirun -np 4 -hostfile hosts ./hello.x
```

Compiling an MPI Program

- **Most MPI implementations supply compilation scripts, eg.**

Language	Command Used to Compile
Fortran 77	mpif77 mpi_prog.f
Fortran 90	mpif90 mpi_prog.f90
C	mpicc mpi_prog.c
C++	mpiCC or mpicxx mpi_prog.C

- **Manual compilation/linking also possible**
  – *Extremely complex*

```
$ gcc –o hello.x -L/applic/compilers/gcc/4.1.2/mpi/openmpi/1.4.2/lib64
  -I/applic/compilers/gcc/4.1.2/mpi/openmpi/1.4.2/include  hello.c  -lmpi
```

MPI Machine File

- A text file telling MPI where to launch processes
- Put separate host name on each line
- Example

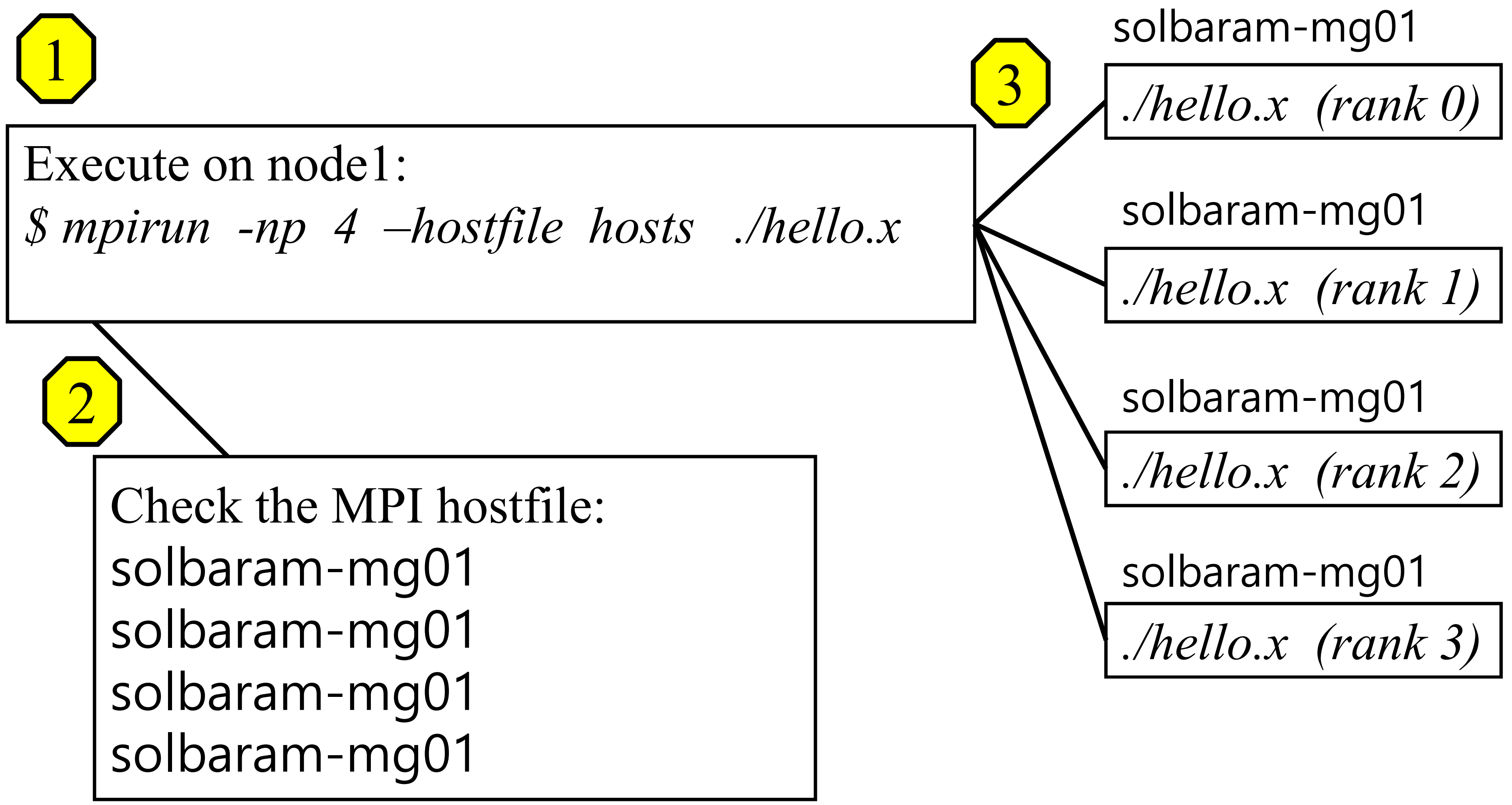
```
node01
node02
node03
node04
```

Solbaram

```
solbaram-mg01
solbaram-mg01
solbaram-mg01
solbaram-mg01
```

- Check implementation for multi-processor node formats
- Default file found in MPI installation

Starting an MPI Program



Six-function MPI

The Whole Library

- **MPI is large and complex**
  - MPI 1.0 have 125 functions
  - MPI 2.0 is even larger

- **But, many MPI features are rarely used**
  - Inter-communicators
  - Topologies
  - Persistent communication
  - Functions designed for library developers

The Absolute Minimum

- **Six MPI functions**
  - Many parallel algorithms can be implemented efficiently with only these functions

Fortran	C
`MPI_INIT()`	`MPI_Init()`
`MPI_COMM_SIZE()`	`MPI_Comm_size()`
`MPI_COMM_RANK()`	`MPI_Comm_rank()`
`MPI_SEND()`	`MPI_Send()`
`MPI_RECV()`	`MPI_Recv()`
`MPI_FINALIZE()`	`MPI_Finalize()`

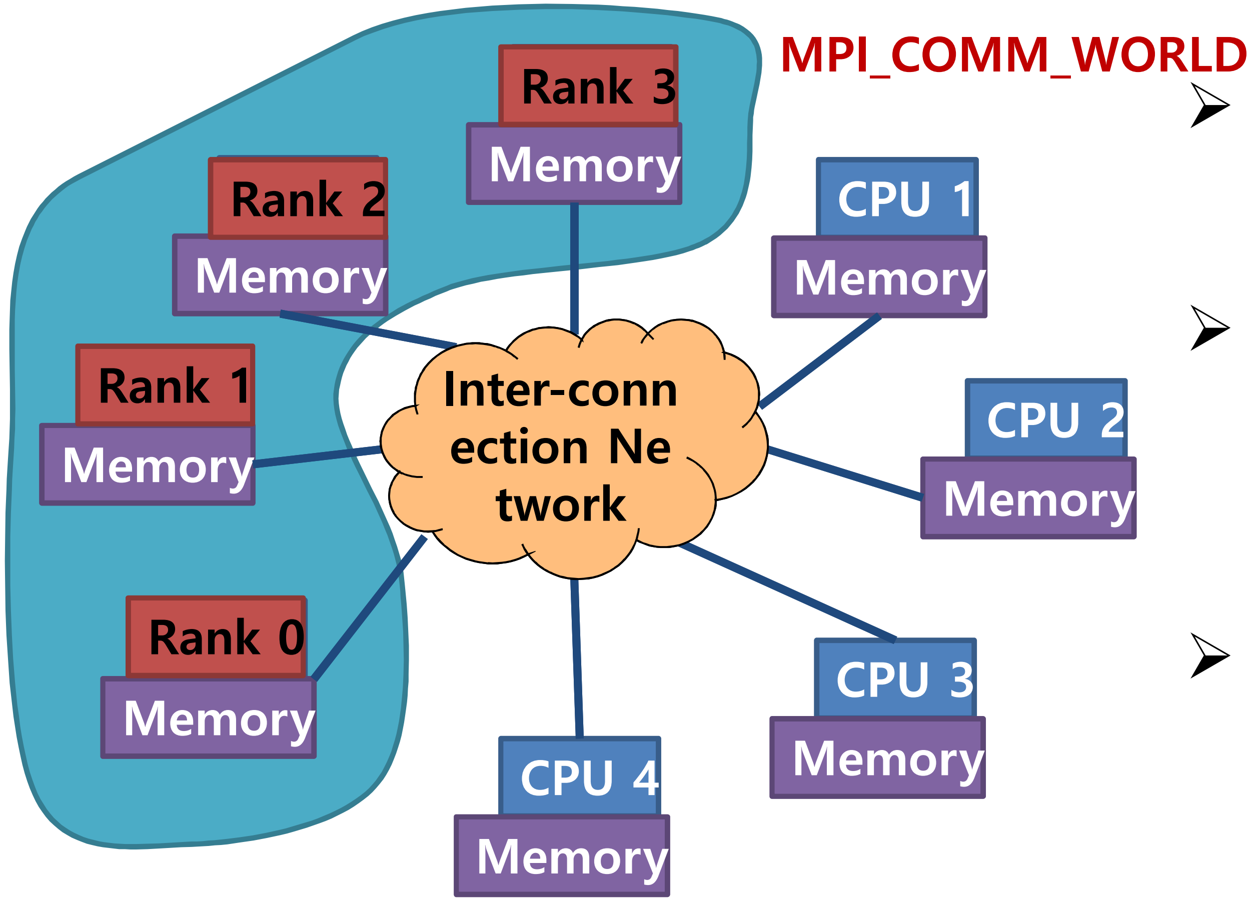
The Absolute Minimum

- **Six MPI functions**
  - Many parallel algorithms can be implemented efficiently with only these functions

Fortran	C
`MPI_INIT()`	`MPI_Init()`
`MPI_COMM_SIZE()`	`MPI_Comm_size()`
`MPI_COMM_RANK()`	`MPI_Comm_rank()`
`MPI_SEND()`	`MPI_Send()`
`MPI_RECV()`	`MPI_Recv()`
`MPI_FINALIZE()`	`MPI_Finalize()`

Starting MPI

Fortran	C
`CALL MPI_INIT(ierr)`	`int MPI_Init(&argc, &argv)`



- **`MPI_Init` prepares the system for MPI execution**
- **Call to `MPI_Init` may update arguments in C**
  - Implementation dependent
- **No MPI functions may be called before `MPI_Init`**

Exiting MPI

Fortran	C
`CALL MPI_FINALIZE(ierr)`	`int MPI_Finalize();`

- **MPI_Finalize frees any memory allocated by the MPI library**
- **No MPI function may be called after calling MPI_Finalize**

If any one process does not reach the finalization statement, the program will appear to hang

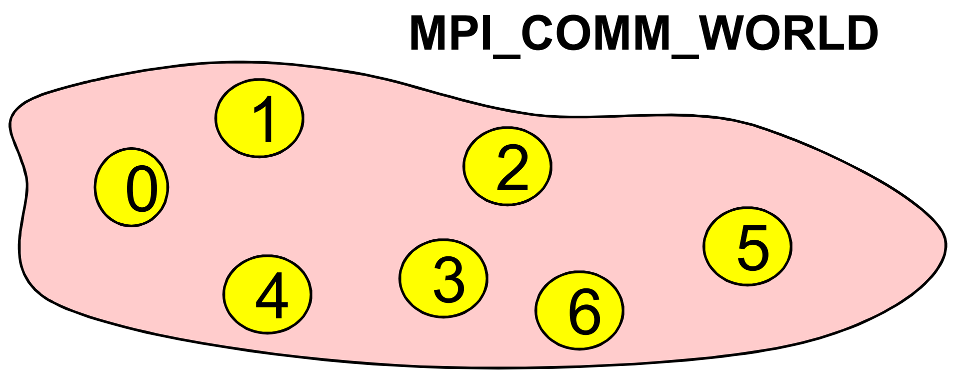
The Absolute Minimum

- Six MPI functions

Fortran	C
`MPI_INIT()`	`MPI_Init()`
`MPI_COMM_SIZE()`	`MPI_Comm_size()`
`MPI_COMM_RANK()`	`MPI_Comm_rank()`
`MPI_SEND()`	`MPI_Send()`
`MPI_RECV()`	`MPI_Recv()`
`MPI_FINALIZE()`	`MPI_Finalize()`

MPI Communicator

- **A handle representing a group of processes that can communicate with each other(more about communicators later)**
- **All MPI communication calls have a communicator argument**
- **Most often you will use MPI_COMM_WORLD**
  - Defined when you call MPI_Init
  - It is all of your processors.



Communicator Size

- How many processes are contained within a communicator?

Fortran	`CALL MPI_COMM_SIZE(comm, size, ierr)`
C	`int MPI_Comm_size(MPI_Comm comm, int *size)`

- MPI_Comm_size returns the number of processes in the specified communicator
- The communicator structure, MPI_Comm, is defined in mpi.h

Processor Rank

- Process ID number within communicator

Fortran	`CALL MPI_COMM_RANK(comm, rank)`
C	`int MPI_Comm_rank(MPI_Comm com, int *rank)`

- MPI_Comm_rank returns the rank of calling process within the specified communicator
- Processes are numbered from *0 to N-1*

bones.c

```
/* program skeleton*/
#include “mpi.h”
int main(int argc, char *argv[]){
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* … your code here … */

    MPI_Finalize();
}
```

bones.f90

```
PROGRAM skeleton
INCLUDE 'mpif.h'
INTEGER ierr, rank, size
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank,
  ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, size,
  ierr)

! … your code here …

CALL MPI_FINALIZE(ierr)
END
```

Lab #1

Lab 2: “Hello, World” with hostname - C

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

int main(int argc, char *argv[])
{
    int       nRank, nProcs;
    char      procName[MPI_MAX_PROCESSOR_NAME];
    int       nNameLen;

    MPI_Init(&argc, &argv);                          // MPI Start
    MPI_Comm_rank(MPI_COMM_WORLD, &nRank);           // Get current processor rank id
    MPI_Comm_size(MPI_COMM_WORLD, &nProcs);          // Get number of processors

    MPI_Get_processor_name(procName, &nNameLen);

    printf("Hello World. (Process name = %s, nRank = %d, nProcs = %d)\n", procName, nRank, nProcs);

    MPI_Finalize();                                  // MPI End
    return 0;
}
```

```
$ mpicc  -o  hello_host  –Wall  hello_host.c
$ mpirun  -np  4  –hostfile  hosts  ./hello_host
```

Lab 2: “Hello, World” with hostname - C

```
$ cat hosts
s0001
s0002
s0003
s0004

$ mpicc  -o  hello_host.x  hello_host.c

$ mpirun  -np  4  -hostfile  hosts  ./hello_host.x
Hello World. (Process name = s0003, nRank = 2, nProcs = 4)
Hello World. (Process name = s0002, nRank = 1, nProcs = 4)
Hello World. (Process name = s0004, nRank = 3, nProcs = 4)
Hello World. (Process name = s0001, nRank = 0, nProcs = 4)
```

Lab 2: "Hello, World" with hostname - Fortran

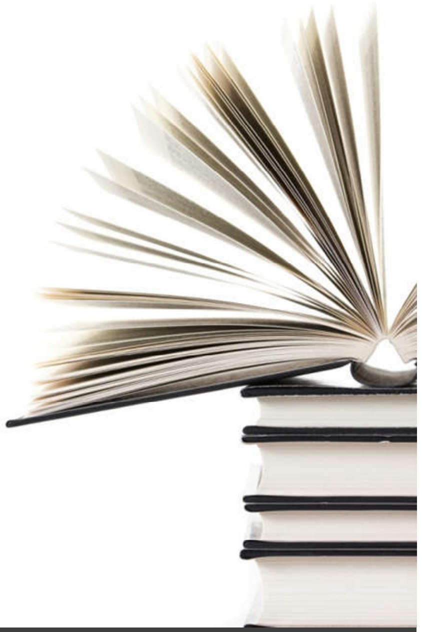
```
PROGRAM  hello
IMPLICIT  NONE
INCLUDE  'mpif.h'

   INTEGER  nRank, nProcs, nNameLen, iErr
   CHARACTER(10)  procName
   INTEGER  myar(5, 5)

   CALL  MPI_Init(iErr)
   CALL  MPI_Comm_size(MPI_COMM_WORLD, nProcs, iErr)
   CALL  MPI_Comm_rank(MPI_COMM_WORLD, nRank, iErr)

   CALL  MPI_Get_processor_name(procName, nNameLen, iErr)

   WRITE (*, *) 'Hello World. (Process name = ', procName, ', nRank = ', nRank, ', nProcs = ',
                  nProcs, ')'

   CALL  MPI_FINALIZE(iErr)

END
```

```
$ mpif90  -o  hello_host.x  –Wall  hello_host.f90
$ mpirun  -np  4  –hostfile  hosts  ./hello_host.x
```

Lab 2: "Hello, World" with hostname - Fortran

```
$ cat hosts
s0001
s0002
s0003
s0004

$ mpif90  -o  hello_host.x  hello_host.f90

$ mpirun  -np  4  -hostfile  hosts  ./hello_host.x
Hello World. (Process name = s0003, nRank = 2, nProcs = 4)
Hello World. (Process name = s0002, nRank = 1, nProcs = 4)
Hello World. (Process name = s0004, nRank = 3, nProcs = 4)
Hello World. (Process name = s0001, nRank = 0, nProcs = 4)
```

Basic Communication



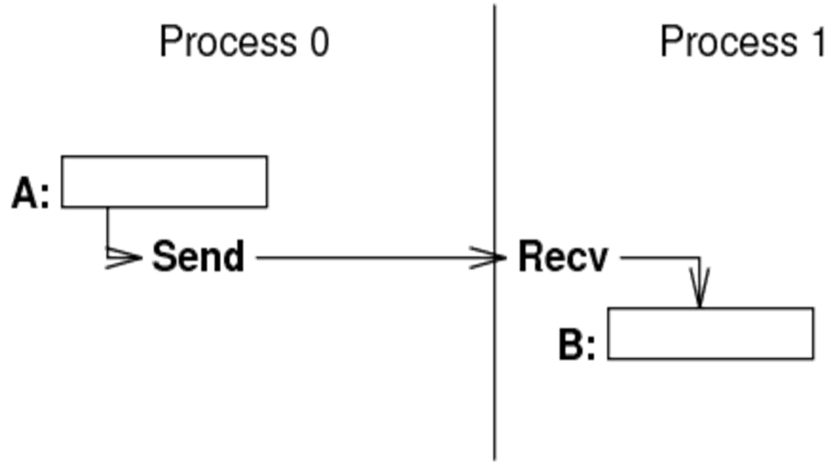
The Absolute Minimum

➢ Six MPI functions

Fortran	C
`MPI_INIT()`	`MPI_Init()`
`MPI_COMM_SIZE()`	`MPI_Comm_size()`
`MPI_COMM_RANK()`	`MPI_Comm_rank()`
`MPI_SEND()`	`MPI_Send()`
`MPI_RECV()`	`MPI_Recv()`
`MPI_FINALIZE()`	`MPI_Finalize()`

Sending and Receiving Message

➢ Basic Message Passing Process



- Where to send
- What to send
- How many to send

- Where to receive
- What to receive
- How many to receive

Message Organization in MPI

- **Message is divided into data and envelope**
  - Data
    - buffer
    - count
    - data type
  - Envelope
    - process identifier (source/destination rank)
    - message tag
    - communicator

MPI Data Types (1/2)

MPI Data Type	C Data Type
MPI_CHAR – 1 Byte character	signed char
MPI_SHORT – 2 Byte integer	signed short int
MPI_INT – 4 Byte integer	signed int
MPI_LONG – 4 Byte integer	signed long int
MPI_UNSIGNED_CHAR – 1 Byte u char	unsigned char
MPI_UNSIGNED_SHORT – 2 Byte u int	unsigned short int
MPI_UNSIGNED – 4 Byte u int	unsigned int
MPI_UNSIGNED_LONG– 4 Byte u int	unsigned long int
MPI_FLOAT – 4 Byte float point	float
MPI_DOUBLE – 8 Byte float point	double
MPI_LONG_DOUBLE- – 8 Byte float point	long double

MPI Data Types (2/2)

MPI Data Type	Fortran Data Type
MPI_INTEGER – 4 Byte Integer	INTEGER
MPI_REAL – 4 Byte floating point	REAL
MPI_DOUBLE_PRECISION – 8 Byte	DOUBLE PRECISION
MPI_COMPLEX – 4 Byte float real	COMPLEX
MPI_LOGICAL – 4 Byte logical	LOGICAL
MPI_CHARACTER – 1 Byte character	CHARACTER(1)

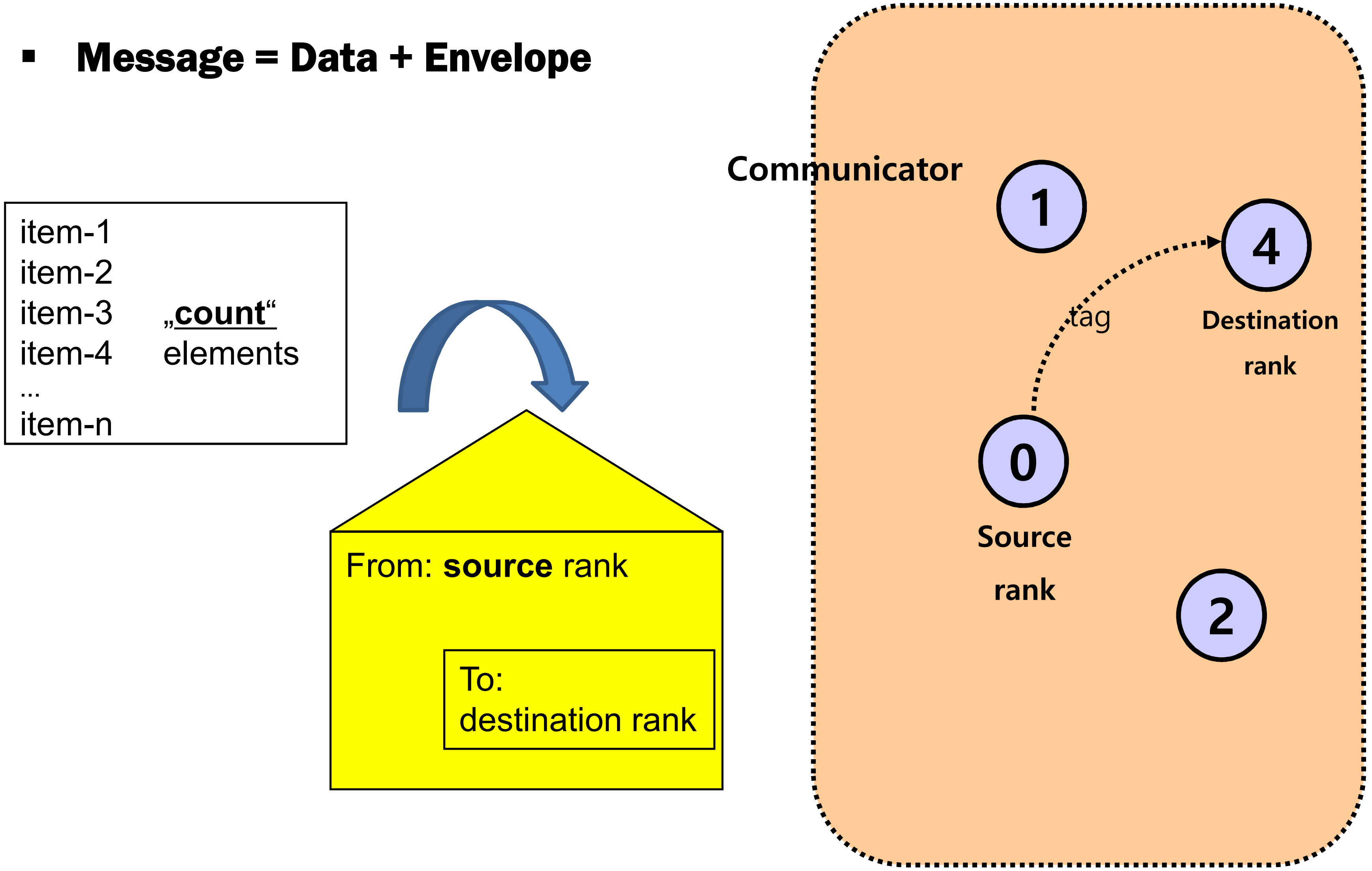
2345	654	96574	-12	7676

count=5
datatype=MPI_INTEGER

INTEGER arr(5)

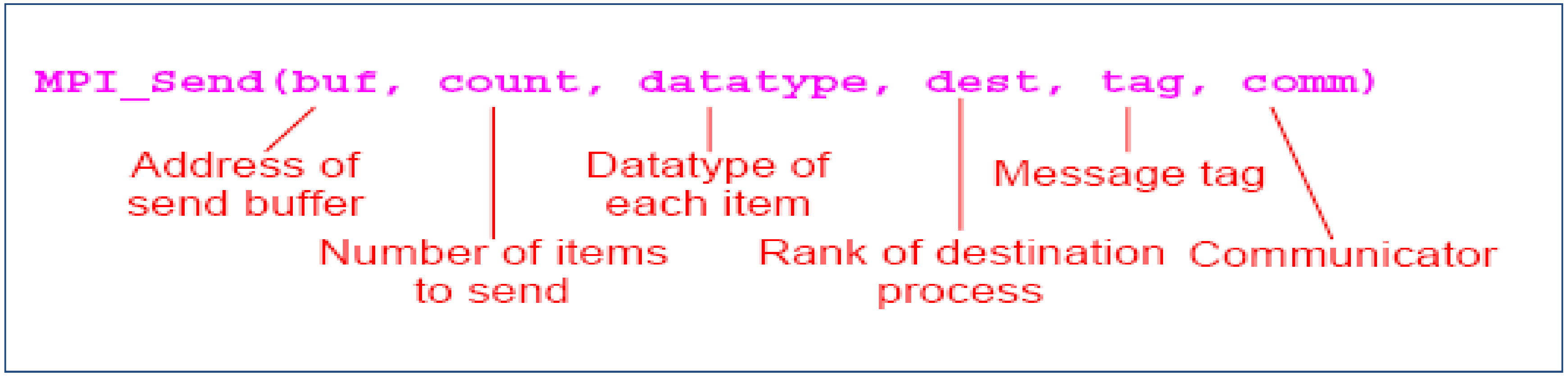
Communication Envelope

- **Message = Data + Envelope**

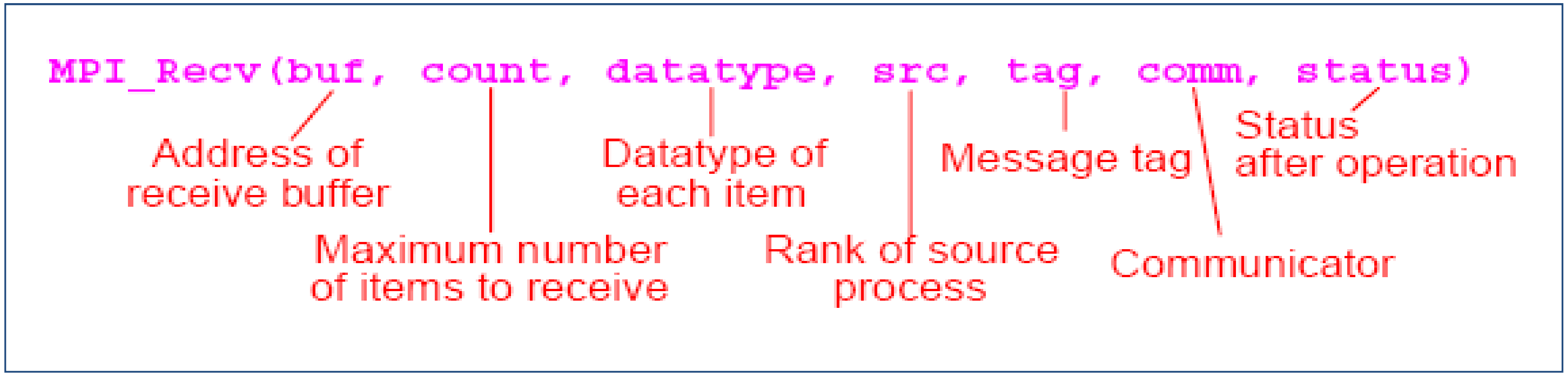


MPI Blocking Send & Receive

➢ **MPI_Send() vs MPI_Recv()**



```
MPI_Send(buf, count, datatype, dest, tag, comm)
```



```
MPI_Recv(buf, count, datatype, src, tag, comm, status)
```

Blocking Recv - Status

➢ Status Information

– Send Process (Rank)
– Tag
– Data size : MPI_GET_COUNT

Information	Fortran	C
source	status(MPI_SOURCE)	status.MPI_SOURCE
tag	status(MPI_TAG)	status.MPI_TAG
Error	status(MPI_ERROR)	status.MPI_ERROR
count	MPI_GET_COUNT()	MPI_Get_count()

Blocking Recv - Status

- **It is possible to use Wild Card in the MPI_Recv.**
  - MPI_ANY_SOURCE
  - MPI_ANY_TAG

MPI_Recv(a, 50, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, status)

MPI Send & Receive Example

- To send an integer x from process 0 to process 1

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank); /* find rank */

if (myrank == 0) {
   int x;
   MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x, 1, MPI_INT,
   0,msgtag,MPI_COMM_WORLD,status);
}
```

Lab #2

SEND RECV Example: C

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
   int rank, i, count;
   float data[100],value[200];
   MPI_Status status;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   if(rank==1) {
      for(i=0;i<100;++i) data[i]=i;
      MPI_Send(data,100,MPI_FLOAT,0,55,MPI_COMM_WORLD);
   }
else {
MPI_Recv(value,200,MPI_FLOAT,MPI_ANY_SOURCE,55,MPI_COMM_WORLD,&status);
 printf("P:%d Got data from processor %d \n",rank, status.MPI_SOURCE);
      MPI_Get_count(&status,MPI_FLOAT,&count);
      printf("P:%d Got %d elements \n",rank,count);
      printf("P:%d value[5]=%f \n",rank,value[5]);
   }
   MPI_Finalize();
}
```

SEND RECV Example: Fortran

```
PROGRAM isend
INCLUDE 'mpif.h'
INTEGER err, rank, size, count
REAL data(100), value(200)
INTEGER status(MPI_STATUS_SIZE)
CALL MPI_INIT(err)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
IF (rank.eq.0) THEN
    data=3.0
    CALL MPI_SEND(data,100,MPI_REAL,1,55,MPI_COMM_WORLD,err)
ELSEIF (rank .eq. 1) THEN
    CALL MPI_RECV(value,200,MPI_REAL,MPI_ANY_SOURCE,55, &
    MPI_COMM_WORLD,status,err)
    PRINT *, "P:",rank," got data from processor ", &
    status(MPI_SOURCE)
    CALL MPI_GET_COUNT(status,MPI_REAL,count,err)
    PRINT *, "P:",rank," got ",count," elements"
    PRINT *, "P:",rank," value(5)=",value(5)
ENDIF
CALL MPI_FINALIZE(err)
END
```

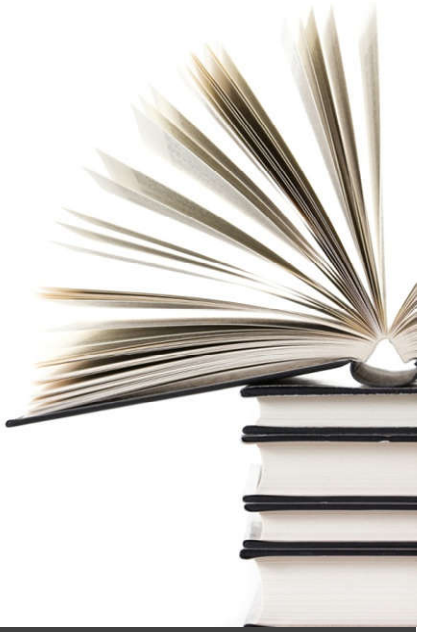
The Absolute Minimum

- Six MPI functions

Fortran	C
`MPI_INIT()`	`MPI_Init()`
`MPI_COMM_SIZE()`	`MPI_Comm_size()`
`MPI_COMM_RANK()`	`MPI_Comm_rank()`
`MPI_SEND()`	`MPI_Send()`
`MPI_RECV()`	`MPI_Recv()`
`MPI_FINALIZE()`	`MPI_Finalize()`

Break!!

P2P: Blocking Communications



Point-to-Point Communication

- **Simplest form of message passing.**
- **One process sends a message to another.**
- **Different types of point-to-point communication:**
  - synchronous send
  - buffered = asynchronous send

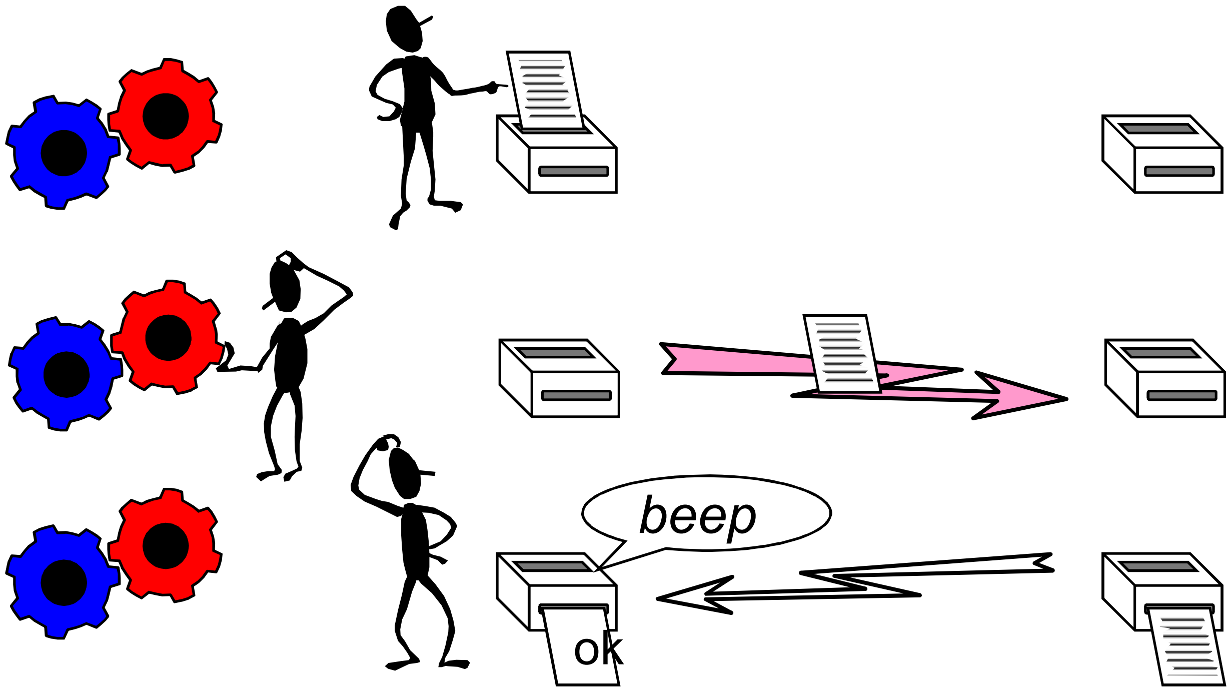
Blocking Operations

- **Some sends/receives may block until another process acts:**
  - synchronous send operation blocks until receive is issued;
  - receive operation blocks until message is sent.

- **Blocking subroutine returns only when the operation has completed.**

Non-Blocking Operations

- Non-blocking operations return immediately and allow the sub-program to perform other work



Communication Mode

Mode	MPI Call Routine	
	Blocking	Non Blocking
Synchronous	MPI_SSEND	MPI_ISSEND
Ready	MPI_RSEND	MPI_IRSEND
Buffer	MPI_BSEND	MPI_IBSEND
Standard	MPI_SEND	MPI_ISEND
Recv	MPI_RECV	MPI_IRECV

Blocking Send (standard)

C	int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
Fortran	MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)

(CHOICE) buf : initial address of send buffer (IN)

INTEGER count : number of elements in send buffer (IN)

INTEGER data type : data type of each send buffer element (IN)

INTEGER dest : rank of destination (IN)

If communication is not needed, MPI_PROC_NULL

INTEGER tag : message tag (IN)

INTEGER comm : MPI communicator (IN)

```
MPI_SEND(a, 50, MPI_REAL, 5, 1, MPI_COMM_WORLD, ierr)
```

Blocking Recv (standard)

C	int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
Fortran	MPI_RECV(buf, count, datatype, source, tag, comm, status, ierr)

(CHOICE) buf : initial address of receive buffer (OUT)

INTEGER count : number of elements in receive buffer (IN)

INTEGER datatype : datatype of each receive buffer element (IN)

INTEGER source : rand of source (IN)

If communication is not needed, MPI_PROC_NULL

INTEGER tag : message tag (IN)

INTEGER comm : MPI communicator (IN)

INTEGER status(MPI_STATUS_SIZE) : Have information of received message (OUT)

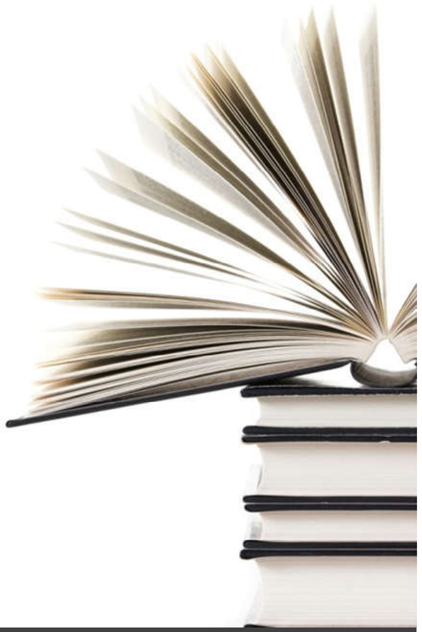
```
MPI_RECV(a,50,MPI_REAL,0,1,MPI_COMM_WORLD,status,ierr)
```

Precautions for Successful Communication

- The receiver has exact rank of sender
- The sender has exact rank of receiver
- Same communicator
- Same message tag
- Enough buffer size of receiver

```
[s0001:20135] *** An error occurred in MPI_Recv
[s0001:20135] *** on communicator MPI_COMM_WORLD
[s0001:20135] *** MPI_ERR_TRUNCATE: message truncated
[s0001:20135] *** MPI_ERRORS_ARE_FATAL (your MPI job will now abort)
```

P2P: Non-Blocking Communications



Non-Blocking Communication

- ➢ **Communication has three steps**

  1. **Initialization : Posting send or recv**
  2. **Perform other job**
     - Do communication and calculation at the same time
  3. **Completion : Waiting or Testing**

- ➢ **Easier to write dead-lock free code**
- ➢ **Reduce communication overhead**

Non-Blocking's Initialization

C	`int MPI_ISend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)`
Fortran	`MPI_ISEND(buf, count, datatype, dest, tag, comm, request, ierr)`

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

INTEGER request : communication request handle (OUT)

Non-blocking Irecv has no status argument

Non-blocking's Completion

- **Waiting or Testing**

  - Waiting
    - Process blocking until the communication is complete
    - **Non-blocking communication + Waiting = Blocking Comm**

  - Testing
    - Return true or false, depending on communication completion

MPI_WAIT

C	`int MPI_Wait(MPI_Request *request, MPI_Status *status)`
Fortran	`MPI_WAIT(request, status, ierr)`

INTEGER request : request handle (IN)

INTEGER status(MPI_STATUS_SIZE) : status object (OUT)

MPI_TEST

C	`int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)`
Fortran	`MPI_TEST(request, flag, status, ierr)`

INTEGER request : request handle (IN)

LOGICAL flag : true if operation completed, or false (OUT)

INTEGER status(MPI_STATUS_SIZE) : status object (OUT)

Unidirectional Communication (1 / 2)

- Blocking Send, Blocking Recv

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
  ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_RECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
  istatus, ierr)
ENDIF
```

- Non-Blocking Send, Blocking Recv

```
IF (myrank==0) THEN
   CALL MPI_ISEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
  ireq, ierr)
   CALL MPI_WAIT(ireq, istatus, ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_RECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
  istatus, ierr)
ENDIF
```

Unidirectional Communication (2 / 2)

- **Blocking Send, Non-Blocking Recv**

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
   ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_IRECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
   ireq, ierr)
   CALL MPI_WAIT(ireq, istatus, ierr)
ENDIF
```

- **Non-Blocking Send, Non-Blocking Recv**

```
IF (myrank==0) THEN
   CALL MPI_ISEND(sendbuf, icount, MPI_REAL, 1, itag, MPI_COMM_WORLD,
   ireq, ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_IRECV(recvbuf, icount, MPI_REAL, 0, itag, MPI_COMM_WORLD,
   ireq, ierr)
ENDIF
CALL MPI_WAIT(ireq, istatus, ierr)
```

Nonblocking Example

Fortran	C
see below	see below

Fortran

```
Example name : non_p2p.f90
PROGRAM non_p2p
INCLUDE 'mpif.h'
   INTEGER ierr, nrank, req
   INTEGER status(MPI_STATUS_SIZE)
   INTEGER :: send = -1, recv = -1, ROOT = 0

   CALL MPI_INIT(ierr)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

   IF (nrank == ROOT) THEN
      PRINT *, 'Before : nrank = ', nrank, 'send = ', send, 'recv =
', recv
      send = 7
      CALL MPI_ISEND(send, 1, MPI_INTEGER, 1, 55, MPI_COMM_WORLD,
                     req, ierr)
      PRINT *, 'Other job calculating'

      CALL MPI_WAIT(req, status, ierr)
   ELSE
      CALL MPI_RECV(recv, 1, MPI_INTEGER, ROOT, 55, MPI_COMM_WORLD,
                    status, ierr)
      PRINT *, 'After  : nrank = ', nrank, 'send = ', send, 'recv =
', recv
   ENDIF

   CALL MPI_FINALIZE(ierr)

END
```

```
$ mpif90 –o non_p2p.x non_p2p.f90
$ mpirun –np 2 –hostfile hosts ./non_p2p.x
```

C

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

int main(int argc, char *argv[])
{
   int nrank, nprocs, tag = 55, ROOT = 0;
   int send = -1, recv = -1;
   MPI_Request req;
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

   if (nrank == ROOT) {
      printf("Before : nrank(%d) send = %d, recv = %d₩n", nrank,
     send, recv);
      send = 7;
      MPI_Isend(&send, 1, MPI_INTEGER, 1, tag, MPI_COMM_WORLD,
                &req);

      printf("Other job calculating.₩n₩n");
      MPI_Wait(&req, &status);
   }
   else {
      MPI_Recv(&recv, 1, MPI_INTEGER, ROOT, tag, MPI_COMM_WORLD,
     &status);
      printf("After  : nrank(%d) send = %d, recv = %d₩n", nrank,
     send, recv);
   }

   MPI_Finalize();

   return 0;
}
```

```
$ mpicc –o non_p2p non_p2p.c
$ mpirun –np 2 –hostfile hosts ./non_p2p
```

Bidirectional Communication (1 / 9)

- 선 송신, 후 수신 **1.** : 메시지 크기에 따라 교착 가능

```
IF (myrank == 0) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank == 1) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, ...)
ENDIF
```

Bidirectional Communication (2 / 9)

➢ 선 송신, 후 수신 **2. (1.**의 경우와 동일**)**

```
 IF (myrank == 0) THEN
   CALL MPI_ISEND(sendbuf, …, ireq, …)
   CALL MPI_WAIT(ireq, …)
   CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank == 1) THEN
   CALL MPI_ISEND(sendbuf, …, ireq, …)
   CALL MPI_WAIT(ireq, …)
   CALL MPI_RECV(recvbuf, ...)
ENDIF
```

Bidirectional Communication (3 / 9)

- 선 송신, 후 수신 **3.** : 메시지 크기와 무관 하게 교착 없음

```
 IF (myrank == 0) THEN
   CALL MPI_ISEND(sendbuf, …, ireq, …)
   CALL MPI_RECV(recvbuf, ...)
   CALL MPI_WAIT(ireq, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_ISEND(sendbuf, …, ireq, …)
   CALL MPI_RECV(recvbuf, ...)
   CALL MPI_WAIT(ireq, …)
ENDIF
```

Bidirectional Communication (5 / 9)

➢ 선 수신, 후 송신 **1.** : 메시지 크기와 무관하게 교착

```
IF (myrank == 0) THEN
   CALL MPI_RECV(recvbuf, ...)
   CALL MPI_SEND(sendbuf, ...)
ELSEIF (myrank == 1) THEN
   CALL MPI_RECV(recvbuf, ...)
   CALL MPI_SEND(sendbuf, ...)
ENDIF
```

Bidirectional Communication (6 / 9)

➢ 선 수신, 후 송신 **2.** : 메시지 크기와 무관하게 교착 없음

```
IF (myrank == 0) THEN
   CALL MPI_IRECV(recvbuf, …,ireq, …)
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_WAIT(ireq, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_IRECV(recvbuf, …,ireq, …)
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_WAIT(ireq, …)
ENDIF
```

Bidirectional Communication (8 / 9)

- 한쪽은 송신부터, 다른 한쪽은 수신부터
  : 블록킹, 논블록킹 루틴의 사용과 무관하게 교착 없음

```
IF (myrank == 0) THEN
   CALL MPI_SEND(sendbuf, ...)
   CALL MPI_RECV(recvbuf, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_RECV(recvbuf, …)
   CALL MPI_SEND(sendbuf, ...)
ENDIF
```

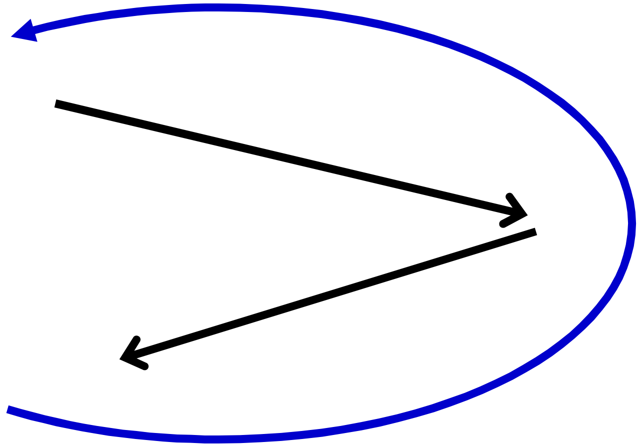
Bidirectional Communication (9 / 9)

➢ 권장 코드

```
IF (myrank == 0) THEN
   CALL MPI_ISEND(sendbuf, …, ireq1, …)
   CALL MPI_IRECV(recvbuf, …, ireq2, …)
ELSEIF (myrank == 1) THEN
   CALL MPI_ISEND(sendbuf, …, ireq1, …)
   CALL MPI_IRECV(recvbuf, …, ireq2, …)
ENDIF

CALL MPI_WAIT(ireq1, …)
CALL MPI_WAIT(ireq2, …)
```

Ping Pong Example

rank=0		rank=1
Send (dest=1)		
	(tag=55)	
		Recv (source=0)
		Send (dest=0)
	(tag=88)	
Recv (source=1)		



```
if (my_rank==0)                         /* i.e., emulated multiple program */
        MPI_Send( ... dest=1 ...)
        MPI_Recv( ... source=1 ...)
else
        MPI_Recv( ... source=0 ...)
        MPI_Send( ... dest=0 ...)
fi
```

Lab #3

deadlock_blocking.c

```
/* Example name : deadlock_blocking.c */
#include <stdio.h>
#include <mpi.h>

/* if BUF_SIZE > 4KB, deadlock */
#define BUF_SIZE    (1024)

int main(int argc, char *argv[])
{
    int nprocs, nrank, i, ROOT = 0;
    MPI_Status status;
    double a[BUF_SIZE], b[BUF_SIZE];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

    for (i=0; i<BUF_SIZE; i++)
        a[i] = i;

    if (nrank == ROOT) {
     for (i=0; i<10; i++)
 printf("before> a[%d] = %.0f, b[%d] = %.0f\n", i, a[i], i, b[i]);
    }
```

deadlock_blocking.c

```
    if (nrank == 0) {
        MPI_Send(a, BUF_SIZE, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD);
        MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD,
  &status);
    }
    else if (nrank == 1) {
        MPI_Send(a, BUF_SIZE, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD);
        MPI_Recv(b, BUF_SIZE, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD,
  &status);
    }
    printf("\n\n");
    if (nrank == ROOT) {
        for (i=0; i<10; i++)
            printf("after > a[%d] = %.0f, b[%d] = %.0f\n", i, a[i], i,
  b[i]);
    }

    MPI_Finalize();
    printf("\n");
    return 0;
}
```

deadlock_blocking.f90

```
! Example name : deadlock_blocking.f90
! if buf_size > 4KB is deadlock
! ------------------------------------------------------------
PROGRAM  deadlock_blocking
INCLUDE  'mpif.h'
    INTEGER  nrank, nprocs, i, ierr
    INTEGER,  PARAMETER :: buf_size = 1500
    DOUBLE PRECISION , DIMENSION(buf_size) :: a, b
    INTEGER status(MPI_STATUS_SIZE)

    CALL  MPI_INIT(ierr)
    CALL  MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)
    CALL  MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
```

deadlock_blocking.f90

```
IF (nrank == 0) THEN
    CALL MPI_SEND(a, buf_size, MPI_DOUBLE_PRECISION, 1, 17,
    MPI_COMM_WORLD, ierr)
    CALL MPI_RECV(b, buf_size, MPI_DOUBLE_PRECISION, 1, 19,
    MPI_COMM_WORLD, status, ierr)
ELSE IF (nrank == 1) THEN
    CALL MPI_SEND(a, buf_size, MPI_DOUBLE_PRECISION, 0, 19,
    MPI_COMM_WORLD, ierr)
    CALL MPI_RECV(b, buf_size, MPI_DOUBLE_PRECISION, 0, 17,
    MPI_COMM_WORLD, status, ierr)
ENDIF

CALL MPI_FINALIZE(ierr)

END
```

Break!!

Collective Communications

81

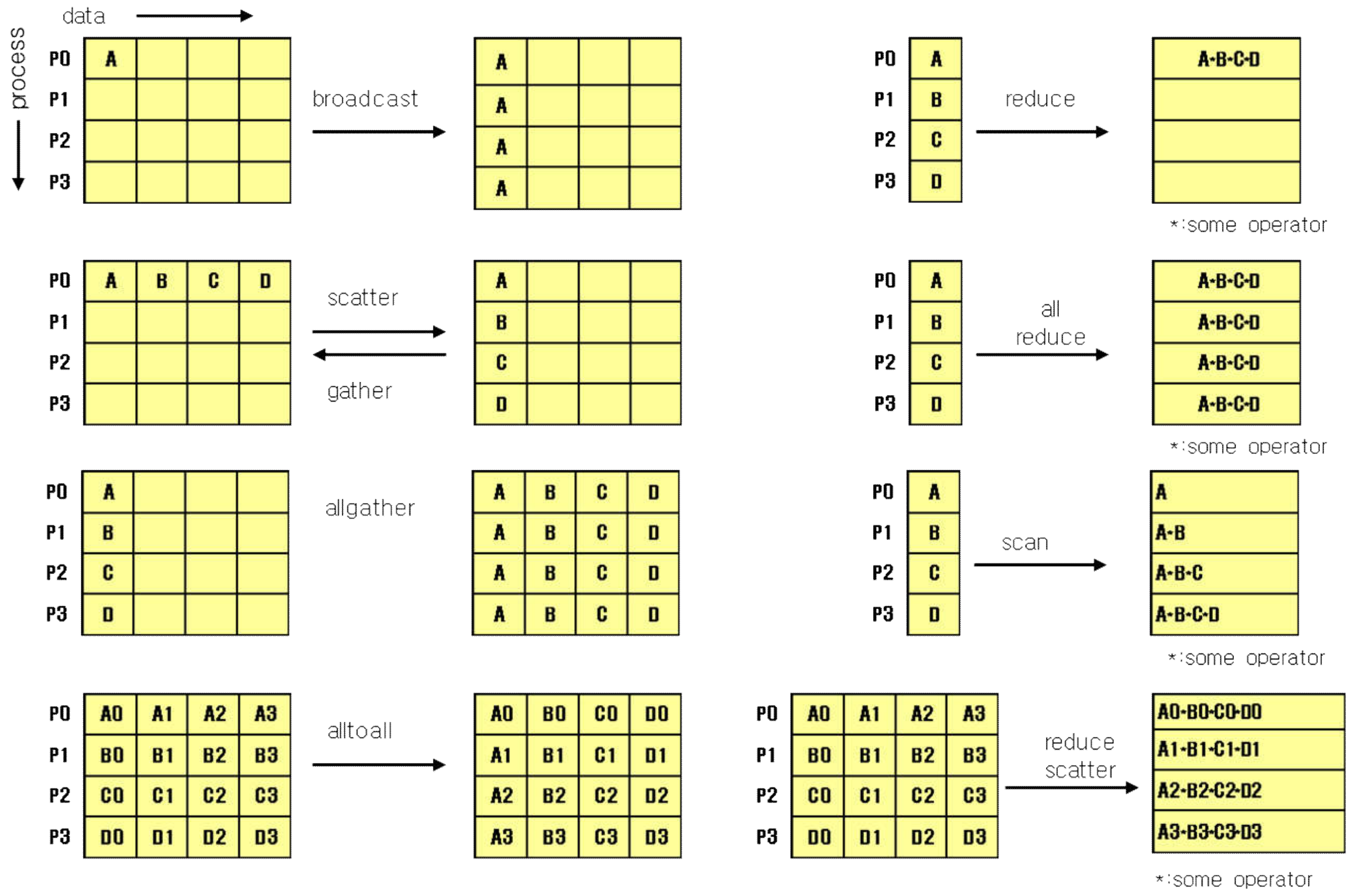
Collective Communications (1/3)

- **A group of processes participate in the communication**
- **Based on Point to Point communication**
- **More efficient, better performance than P2P Communications**
- **Special feature**
  - All processes in the communicator group must be called
  - All collective operations are blocking
  - No message tag

Collective Communications (2/3)

Category	Subroutines
One buffer	**MPI_BCAST**
One send buffer and one receive buffer	**MPI_GATHER**, **MPI_SCATTER**, **MPI_ALLGATHER**, MPI_ALLTOALL, MPI_GATHERV, MPI_SCATTERV, MPI_ALLGATHERV, MPI_ALLTOALLV
Reduction	**MPI_REDUCE**, **MPI_ALLREDUCE**, MPI_SCAN, MPI_REDUCE_SCATTER
Others	**MPI_BARRIER**, MPI_OP_CREATE, MPI_OP_FREE

Collective Communications (3/3)



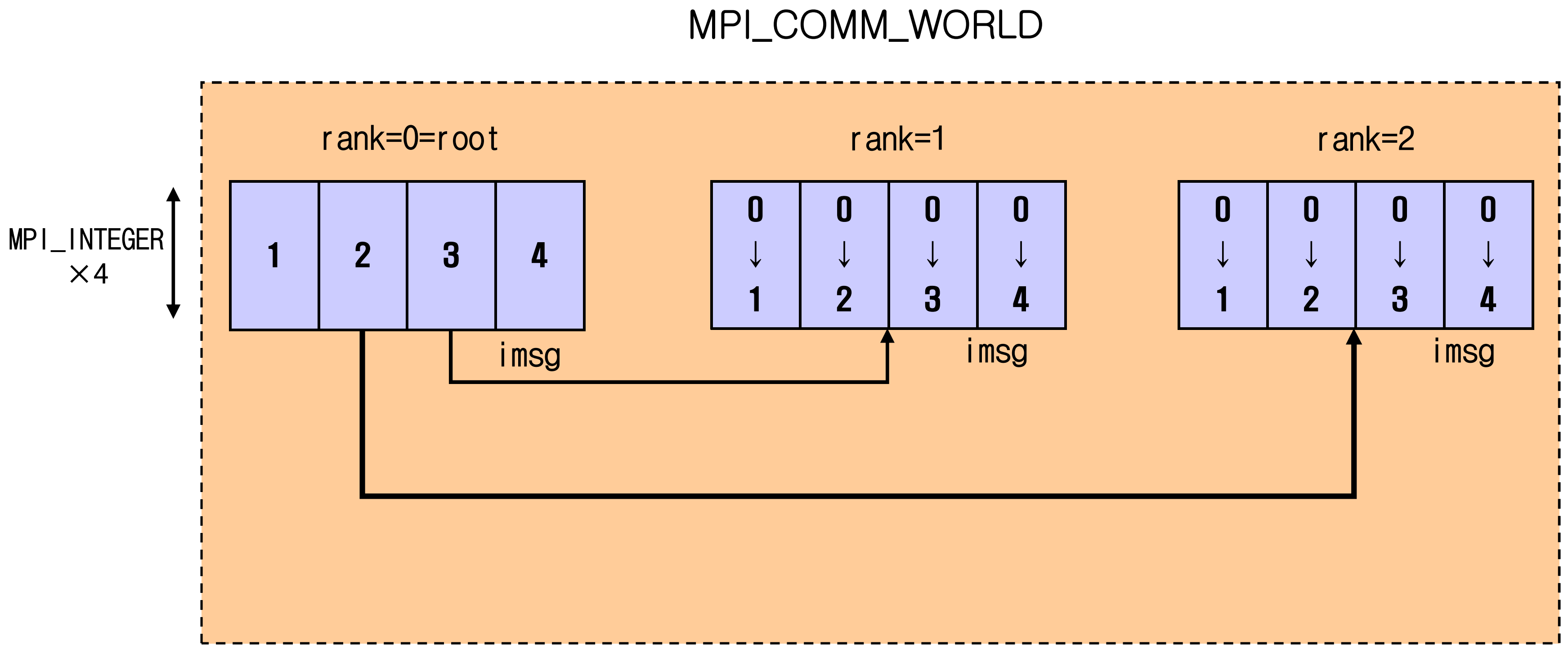
MPI_BCAST

C	`int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)`
Fortran	`MPI_BCAST(buffer, count, datatype, root, comm, ierr)`

(CHOICE) buffer : starting address of buffer (INOUT)
INTEGER count : number of elements in buffer (IN)
INTEGER datatype : mpi data type of buffer (IN)
INTEGER root : rank of broadcast root (IN)
INTEGER comm : communicator (IN)

➢ **MPI_BCAST send messages from one process to all others**

MPI_BCAST


MPI_COMM_WORLD
rank=0=root
rank=1
rank=2
MPI_INTEGER ×4
1
2
3
4
imsg
0 ↓ 1
0 ↓ 2
0 ↓ 3
0 ↓ 4
imsg
0 ↓ 1
0 ↓ 2
0 ↓ 3
0 ↓ 4
imsg

Lab #4

MPI_BCAST Example: C

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
   int i, nrank, nprocs, ROOT = 0;
   int buf[4] = { 0, 0, 0, 0 };

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

   if (nrank == ROOT) {
      buf[0] = 5; buf[1] = 6; buf[2] = 7; buf[3] = 8;
   }

   printf("rank (%d) :  Before :  ", nrank);
   for (i=0; i<4; i++)  printf(" %d", buf[i]);
   printf("\n");

   MPI_Bcast(buf, 4, MPI_INT, ROOT, MPI_COMM_WORLD);

   printf("rank (%d) :  After  :  ", nrank);
   for (i=0; i<4; i++)  printf(" %d", buf[i]);
   printf("\n");

   MPI_Finalize();
   return 0;
}
```

MPI_BCAST Example: Fortran

```
PROGRAM bcast
INCLUDE "mpif.h"

   INTEGER buf(4), nprocs, nrank, ierr
   INTEGER :: ROOT = 0

   DATA buf/0, 0, 0, 0/

   CALL MPI_INIT(ierr)
   CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

   IF (nrank == ROOT) THEN
      buf(1) = 5; buf(2) = 6; buf(3) = 7; buf(4) = 8
   END IF

   print *, 'rank = ', nrank, ' Before :', buf

   CALL MPI_BCAST(buf, 4, MPI_INTEGER, ROOT,
                  MPI_COMM_WORLD, ierr)

   print *, 'rank = ', nrank, ' After  :', buf

   CALL MPI_FINALIZE(ierr)
END
```

MPI_GATHER

C	`int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_GATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)`

(CHOICE)sendbuf : starting address of send buffer (IN)
INTEGER sendcount : number of elements in send buffer (IN)
INTEGER sendtype : mpi data type of send buffer elements (IN)
(CHOICE) recvbuf : starting address of recv buffer (OUT)
INTEGER recvcount : number of elements for any single receive(IN)
INTEGER recvtype : mpi data type of recv buffer elements(IN)
INTEGER root : rank of receiving process (IN)
INTEGER comm : communicator (IN)

- **Gather together values from a group of processes**

MPI_GATHER : Notice

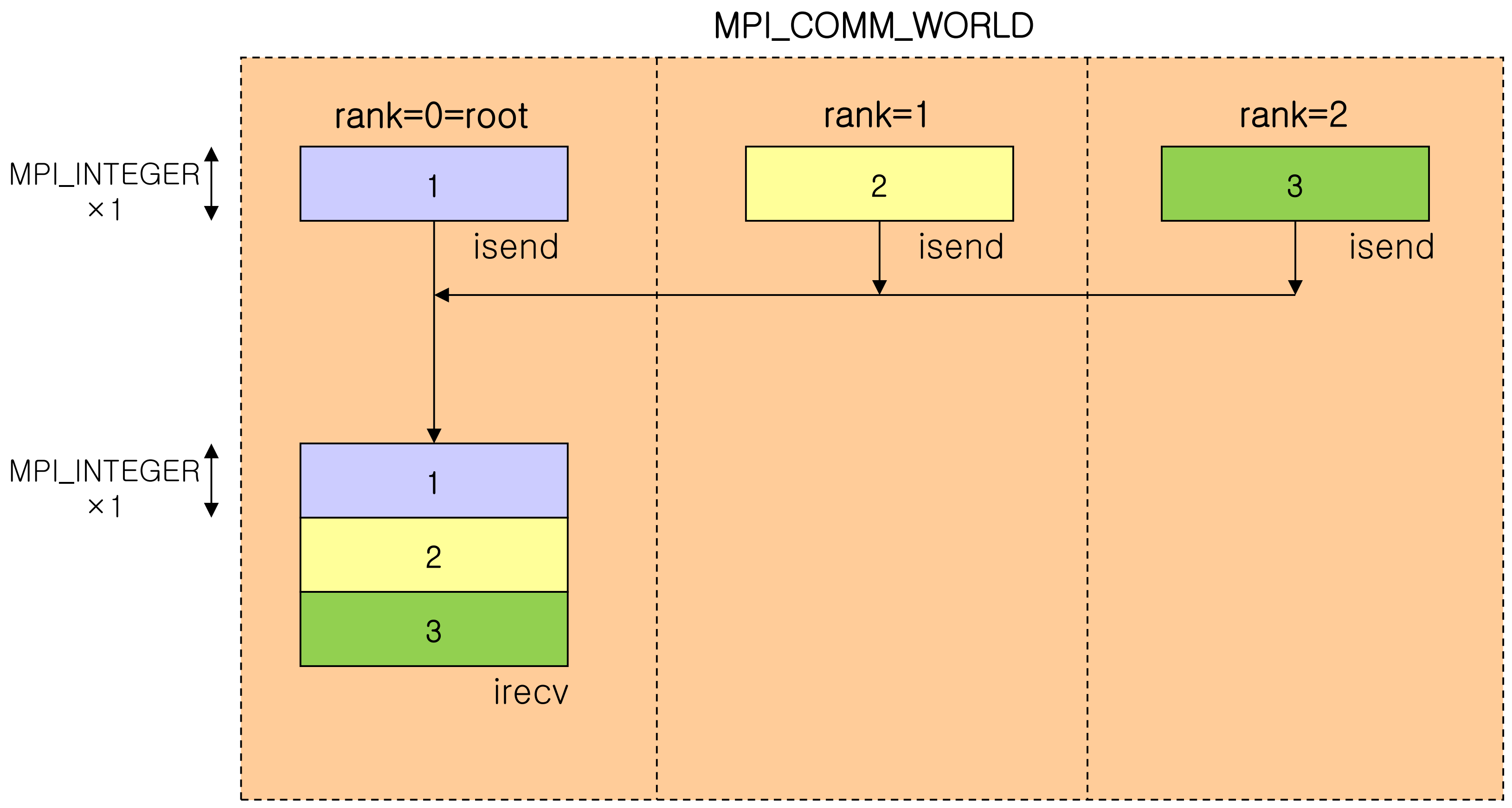
- **Sendbuf and recvbuf don't use same name**

  ➔ Apply equally to the all collective communication which use sendbuf and recvbuf

- **Same data size**

- **In case of not being same data size ➔ MPI_GATHERV**

MPI_GATHER


MPI_COMM_WORLD
rank=0=root
rank=1
rank=2
MPI_INTEGER ×1
1
2
3
isend
isend
isend
MPI_INTEGER ×1
1
2
3
irecv

Lab #5

MPI_GATHER Example: C

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
   int i, nprocs, nrank;
   int isend, irecv[4];

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

   isend = nrank + 1;
   printf("rank (%d) : isend = %d ", nrank, isend);

   MPI_Gather(&isend, 1, MPI_INT, irecv, 1, MPI_INT, 0,
  MPI_COMM_WORLD);

   if (nrank == 0) {
      printf("\n");
         for (i=0; i<3; i++)
            printf("rank (%d) : irecv[%d] = %d\n",
                                    nrank, i, irecv[i]);
   }
   printf("\n");
   MPI_Finalize();

   return 0;
}
```

MPI_GATHER Example: Fortran

```
PROGRAM gather
INCLUDE "mpif.h"
INTEGER isend, irecv(4), nprocs, nrank, ierr
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

isend = nrank + 1
print *, 'rank :', nrank, 'isend :', isend

CALL MPI_GATHER(isend, 1, MPI_INTEGER, irecv, 1, MPI_INTEGER, 0,
  MPI_COMM_WORLD, ierr)

if (nrank == 0) then
  print *, 'rank :', nrank, 'irecv =', irecv
endif

CALL MPI_FINALIZE(ierr)
END
```

MPI_GATHERV

C	`int MPI_Gatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int displs, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_GATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, ierr)`

…

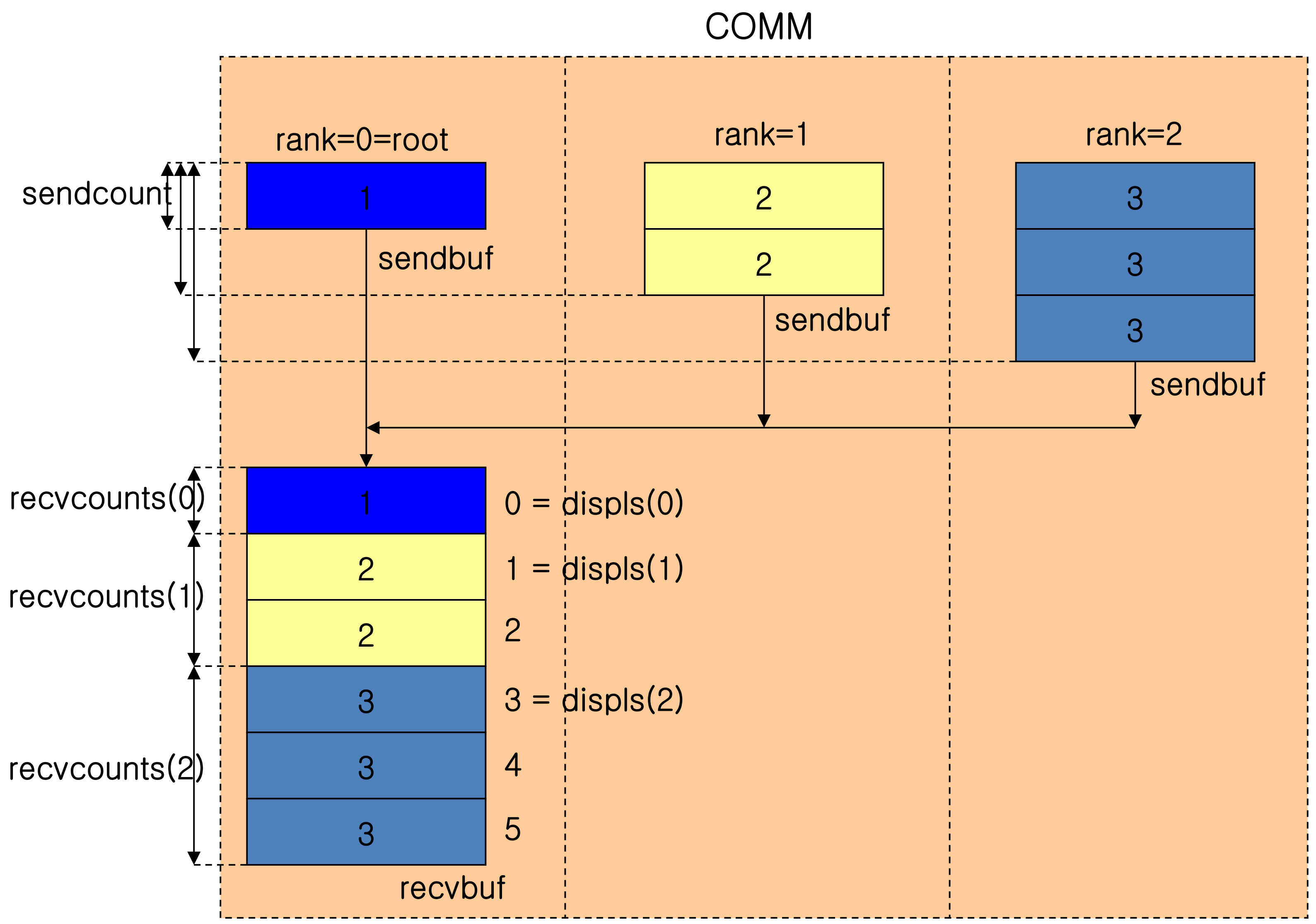
(CHOICE) recvbuf : address of receive buffer (OUT)

INTEGER recvcounts(*) : non-negative integer array (of length group size) containing the number of elements that are received from each process (IN)

INTEGER displs(*) : integer array (of length group size). Entry I specifies the displacement relative to recvbuf at which to place the incoming data from process i (IN)

…

MPI_GATHERV



COMM
rank=0=root
rank=1
rank=2
sendcount
1
sendbuf
2
2
sendbuf
3
3
3
sendbuf
recvcounts(0)
recvcounts(1)
recvcounts(2)
1
2
2
3
3
3
0 = displs(0)
1 = displs(1)
2
3 = displs(2)
4
5
recvbuf

Lab #6

MPI_GATHERV Example: C

```
/*gatherv*/
#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[]){
  int i, myrank ;
  int isend[3], irecv[6];
  int iscnt, ircnt[3]={1,2,3}, idisp[3]={0,1,3};
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  for(i=0; i<myrank+1; i++) isend[i] = myrank + 1;
  iscnt = myrank +1;
  MPI_Gatherv(isend, iscnt, MPI_INT, irecv, ircnt, idisp,
              MPI_INT, 0, MPI_COMM_WORLD);
  if(myrank == 0) {
      printf(" irecv = "); for(i=0; i<6; i++) printf(" %d", irecv[i]);
      printf("\n");
  }
  MPI_Finalize();
}
```

MPI_GATHERV Example: Fortran

```
PROGRAM gatherv
INCLUDE 'mpif.h'
INTEGER isend(3), irecv(6)
INTEGER ircnt(0:2), idisp(0:2)
DATA ircnt/1,2,3/ idisp/0,1,3/
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
DO i=1,myrank+1
   isend(i) = myrank + 1
ENDDO
iscnt = myrank + 1
CALL MPI_GATHERV(isend,iscnt,MPI_INTEGER,irecv,ircnt,idisp,&
   MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
IF (myrank==0) THEN
   PRINT *,'irecv =',irecv
ENDIF
CALL MPI_FINALIZE(ierr)
END
```

MPI_ALLGATHER

C	`int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)`

(CHOICE)sendbuf : starting address of send buffer (IN)

INTEGER sendcount : number of elements in send buffer (IN)

INTEGER sendtype : mpi data type of send buffer elements (IN)

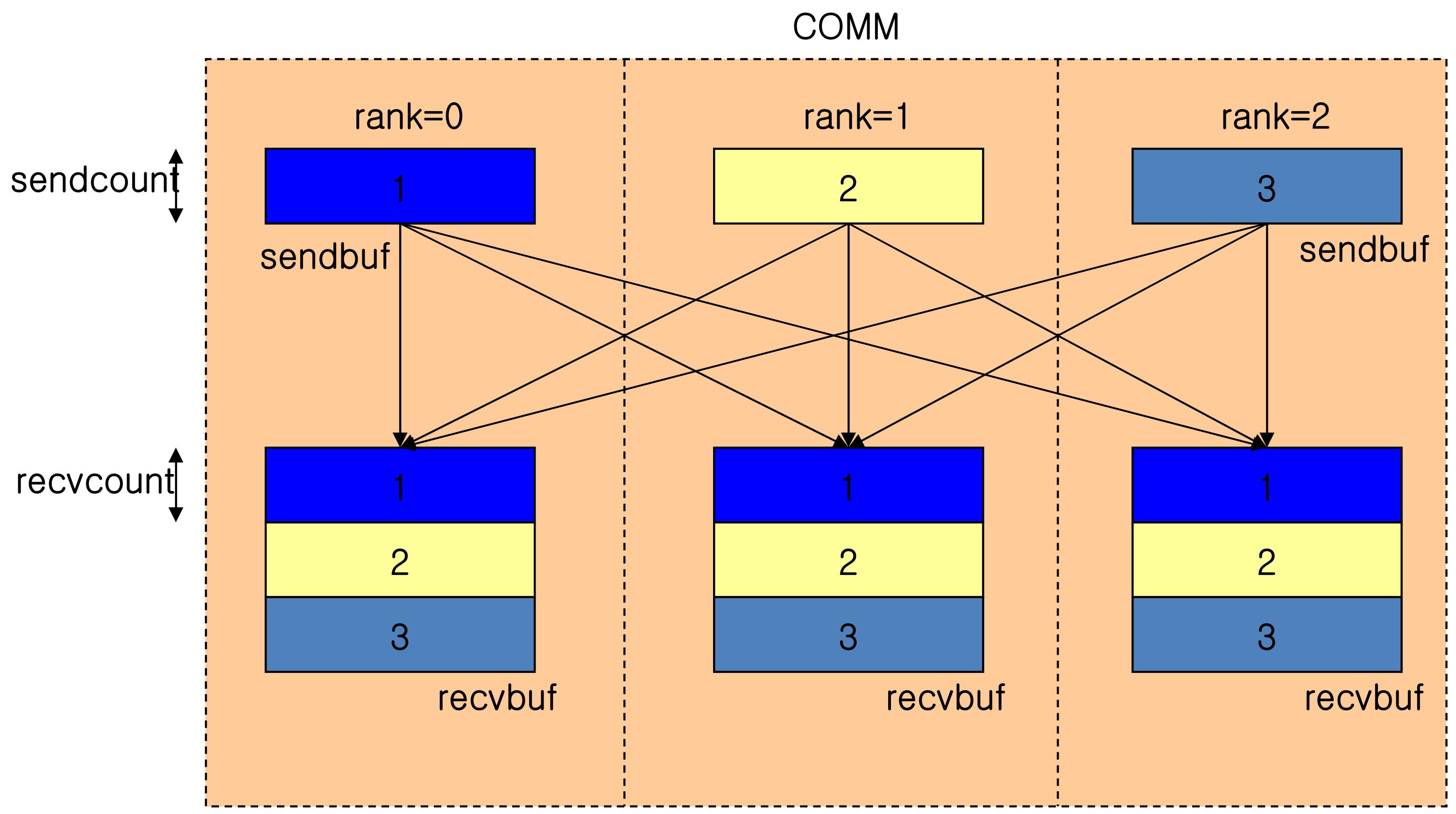
(CHOICE) recvbuf : starting address of recv buffer (OUT)

INTEGER recvcount : number of elements for any single receive(IN)

INTEGER recvtype : mpi data type of recv buffer elements(IN)

INTEGER comm : communicator (IN)

MPI_ALLGATHER Example



MPI_ALLGATHERV

C	`int MPI_Allgatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int displs, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLGATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm, ierr)`

…

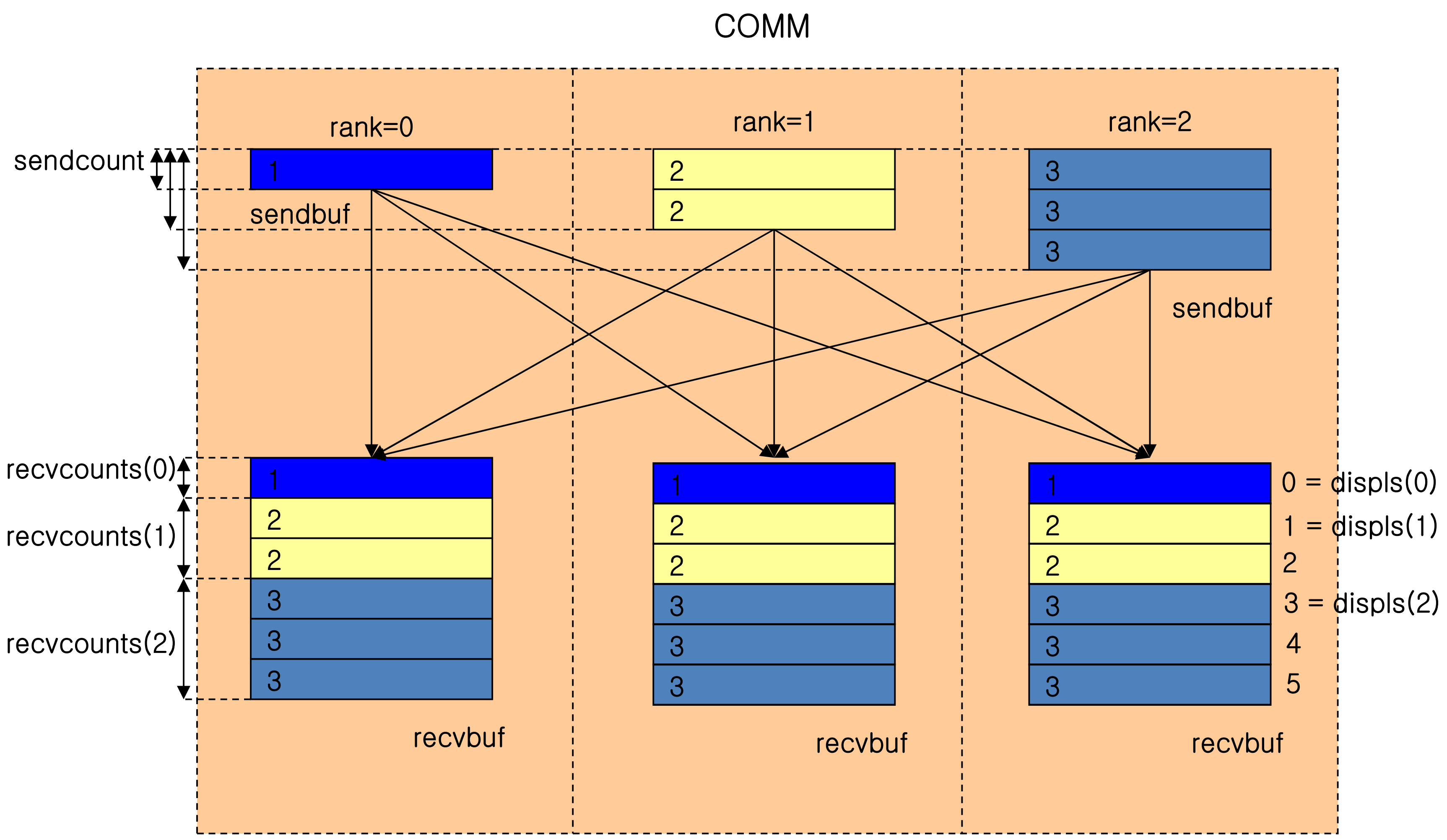
(CHOICE) recvbuf : address of receive buffer (OUT)

INTEGER recvcounts(*) : non-negative integer array (of length group size) containing the number of elements that are received from each process (IN)

INTEGER displs(*) : integer array (of length group size). Entry I specifies the displacement relative to recvbuf at which to place the incoming data from process i (IN)

…

MPI_ALLGATHERV Example



MPI_SCATTER

C	`int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_SCATTER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)`

(CHOICE)sendbuf : starting address of send buffer (IN)

INTEGER sendcount : number of elements in send buffer (IN)

INTEGER sendtype : mpi data type of send buffer elements (IN)

(CHOICE) recvbuf : starting address of recv buffer (OUT)

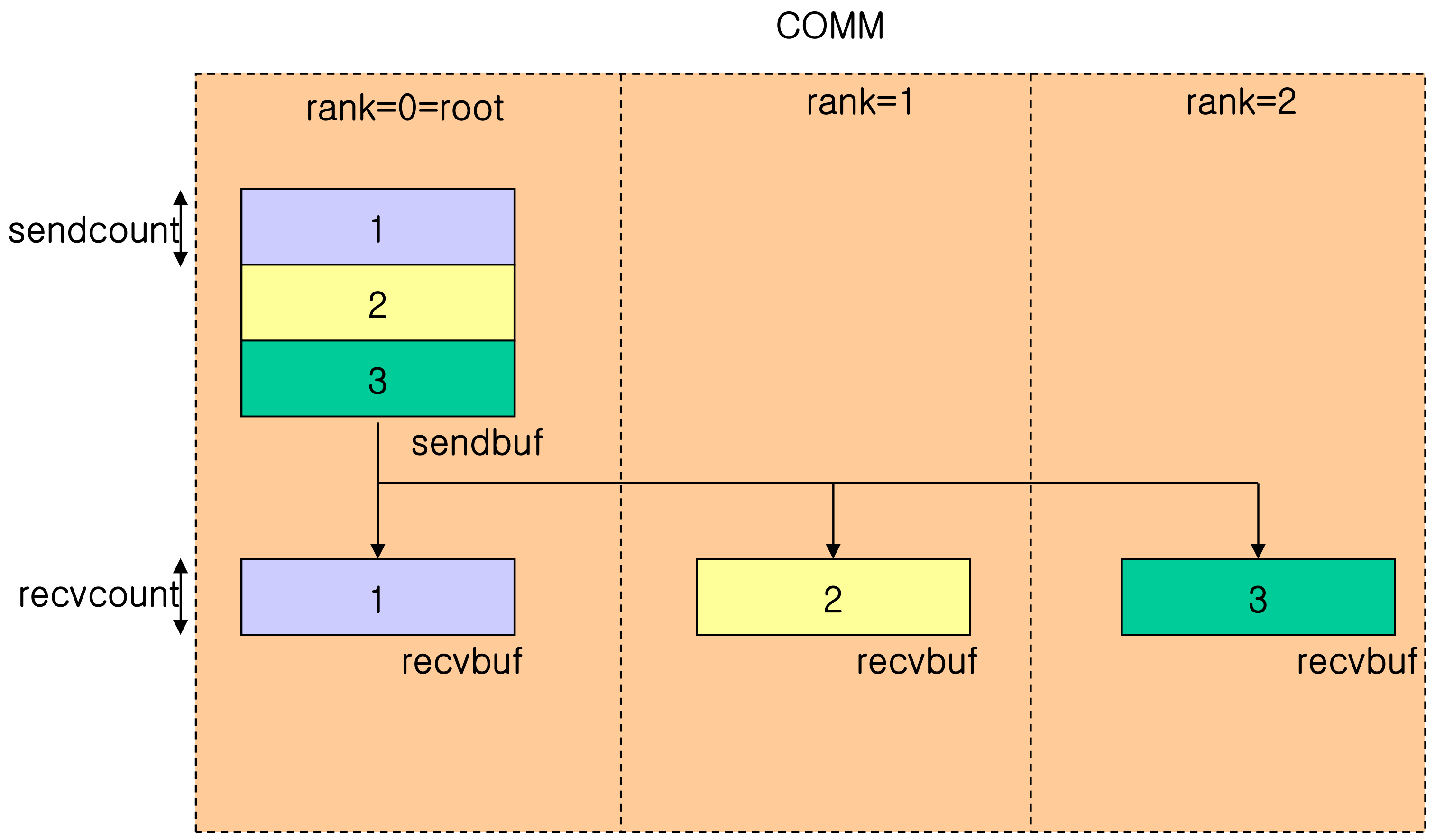
INTEGER recvcount : number of elements for any single receive(IN)

INTEGER recvtype : mpi data type of recv buffer elements(IN)

INTEGER root : rank of receiving process (IN)

INTEGER comm : communicator (IN)

MPI_SCATTER


COMM
rank=0=root
rank=1
rank=2
sendcount
1
2
3
sendbuf
recvcount
1
recvbuf
2
recvbuf
3
recvbuf

MPI_SCATTERV

C	`int MPI_Scatterv(void *sendbuf, int sendcounts, int displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
Fortran	`MPI_SCATTERV(sendbuf, sendcounts, displs, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)`

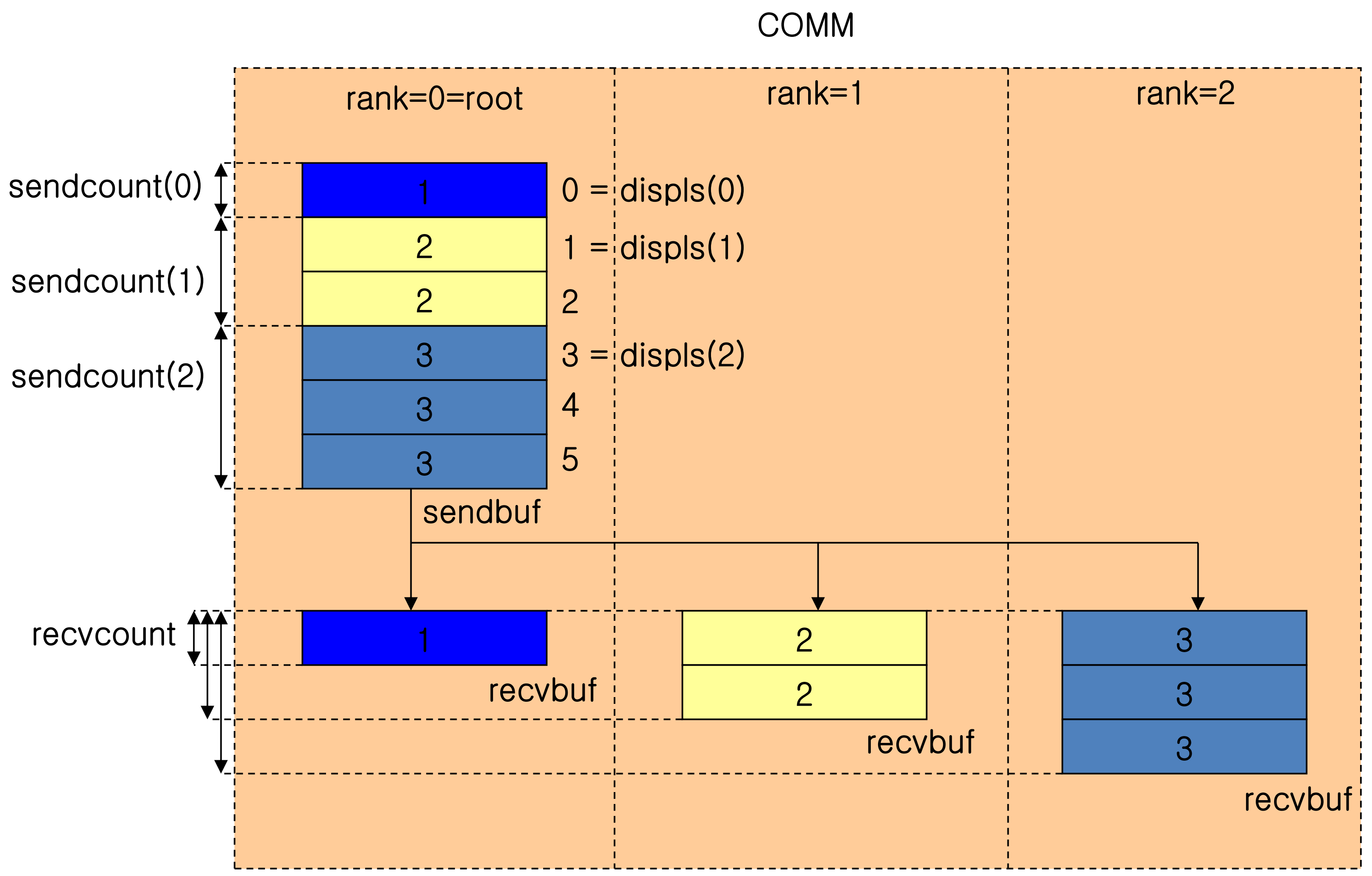
…

INTEGER sendcounts : non−negative integer array (of length group size) specifying the number of elements to send to each rank (IN)

INTEGER displs: integer array (of length group size). Entry i specifies the displacement (relative to sendbuf) from which to take the outgoing data to process i (IN)

…

MPI_SCATTERV



MPI_REDUCE

C	`int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)`
Fortran	`MPI_REDUCE(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)`

(CHOICE) sendbuf : start address of send buffer (IN)
(CHOICE) recvbuf : start address of recv buffer (OUT)
INTEGER count : number of elements in send buffer (IN)
INTEGER datatype : mpi data type of elements of send buffer(IN)
INTEGER op : reduce operation (IN)
INTEGER root : rank of root process(IN)
INTEGER comm : communicator(IN)

➢ **Reduces values on all processes to a single value**

MPI_REDUCE : Operation & Data type(1/3)

Operation	Data Type (Fortran)
MPI_SUM(sum), MPI_PROD(product)	MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_COMPLEX
MPI_MAX(maximum), MPI_MIN(minimum)	MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION
MPI_MAXLOC(max value and location), MPI_MINLOC(min value and location)	MPI_2INTEGER, MPI_2REAL, MPI_2DOUBLE_PRECISION
MPI_LAND(logical AND), MPI_LOR(logical OR), MPI_LXOR(logical XOR)	MPI_LOGICAL
MPI_BAND(bitwise AND), MPI_BOR(bitwise OR), MPI_BXOR(bitwise XOR)	MPI_INTEGER, MPI_BYTE

MPI_REDUCE : Operation & Data type(2/3)

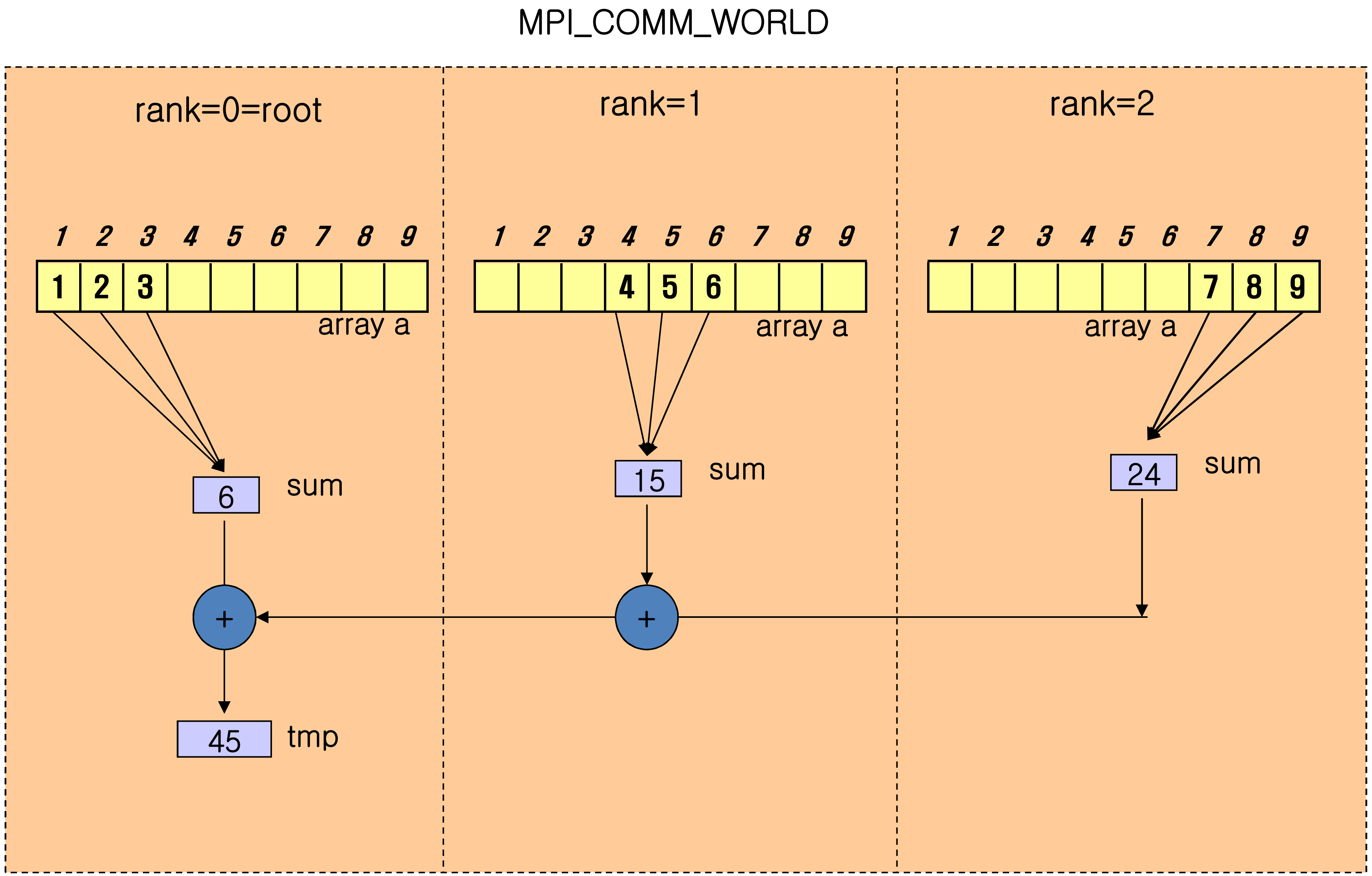
Operation	Data Type (C)
MPI_SUM(sum), MPI_PROD(product) MPI_MAX(maximum), MPI_MIN(minimum)	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED MPI_UNSIGNED_LONG, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE
MPI_MAXLOC(max value and location), MPI_MINLOC(min value and location)	MPI_FLOAT_INT, MPI_DOUBLE_INT, MPI_LONG_INT, MPI_2INT, MPI_SHORT_INT, MPI_LONG_DOUBLE_INT
MPI_LAND(logical AND), MPI_LOR(logical OR), MPI_LXOR(logical XOR)	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED MPI_UNSIGNED_LONG
MPI_BAND(bitwise AND), MPI_BOR(bitwise OR), MPI_BXOR(bitwise XOR)	MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED MPI_UNSIGNED_LONG, MPI_BYTE

MPI_REDUCE : Operation & Data type(3/3)

- Data type MPI_MAXLOC, MPI_MINLOC in C

Data Type	Description (C)
MPI_FLOAT_INT	{ MPI_FLOAT, MPI_INT}
MPI_DOUBLE_INT	{ MPI_DOUBLE, MPI_INT}
MPI_LONG_INT	{ MPI_LONG, MPI_INT}
MPI_2INT	{ MPI_INT, MPI_INT}
MPI_SHORT_INT	{ MPI_SHORT, MPI_INT}
MPI_LONG_DOUBLE_INT	{ MPI_LONG_DOUBLE, MPI_INT}

MPI_REDUCE


MPI_COMM_WORLD
rank=0=root
rank=1
rank=2
1 2 3 4 5 6 7 8 9
1 2 3
4 5 6
7 8 9
array a
sum
6
15
24
+
45
tmp

Lab #7

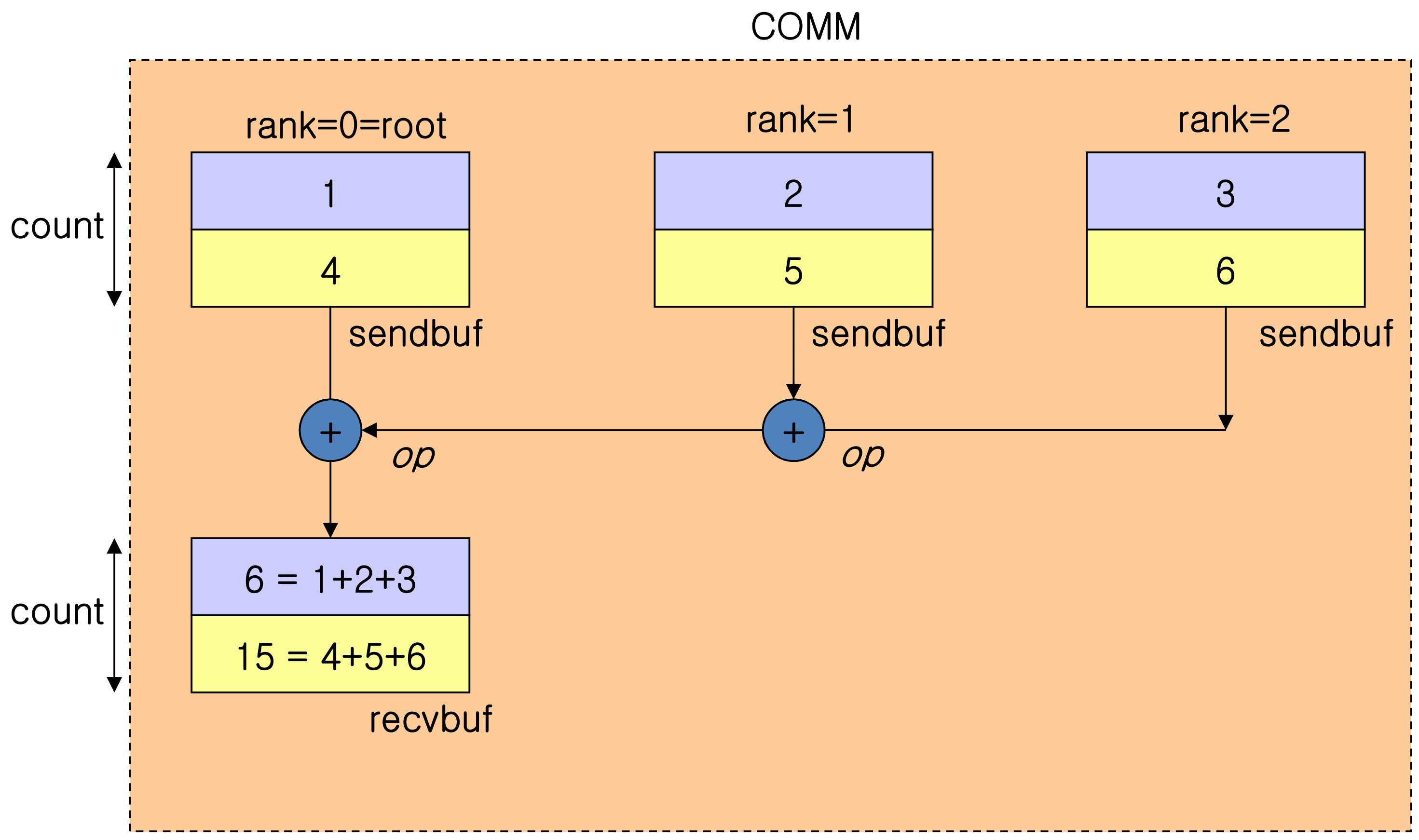
MPI_REDUCE Example: C

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

int main(int argc, char *argv[])
{
     int i, nrank, start, end, ROOT = 0;
     double a[9], sum, tsum;

     MPI_Init(&argc, &argv);
     MPI_Comm_rank(MPI_COMM_WORLD, &nrank);

     start = nrank * 3;
     end = start + 2;

     for (i=start; i<end+1; i++) {
           a[i] = i + 1;
           if (i == start) printf("rank (%d)  ", nrank);
           printf("a[%d] = %.2f   ", i, a[i]);
     }

     sum = 0.0;
     for (i=start; i<end+1; i++) sum = sum + a[i];

     MPI_Reduce(&sum, &tsum, 1, MPI_DOUBLE, MPI_SUM,ROOT, MPI_COMM_WORLD);

     if (nrank == ROOT) printf("\nrank(%d):sum= %.2f.\n", nrank, tsum);
     printf("\n");

     MPI_Finalize();
     return 0;
}
```

MPI_REDUCE Example: Fortran

```
PROGRAM reduce
IMPLICIT NONE
INCLUDE "mpif.h"

     INTEGER nrank, ierr, ista, iend, i
     REAL a(9), sum, tsum

     CALL MPI_INIT(ierr)
     CALL MPI_COMM_RANK(MPI_COMM_WORLD, nrank, ierr)

     ista = nrank * 3 + 1
     iend = ista + 2

     DO i=ista, iend
          a(i) = i
     ENDDO

     sum = 0.0
     DO i=ista, iend
          sum = sum + a(i)
     ENDDO

     CALL MPI_REDUCE(sum, tsum, 1, MPI_REAL, MPI_SUM, 0, MPI_COMM_WORLD, ierr)

     IF (nrank == 0) THEN
          PRINT *, 'sum =',tsum
     ENDIF
     CALL MPI_FINALIZE(ierr)
END
```

MPI_REDUCE : Array

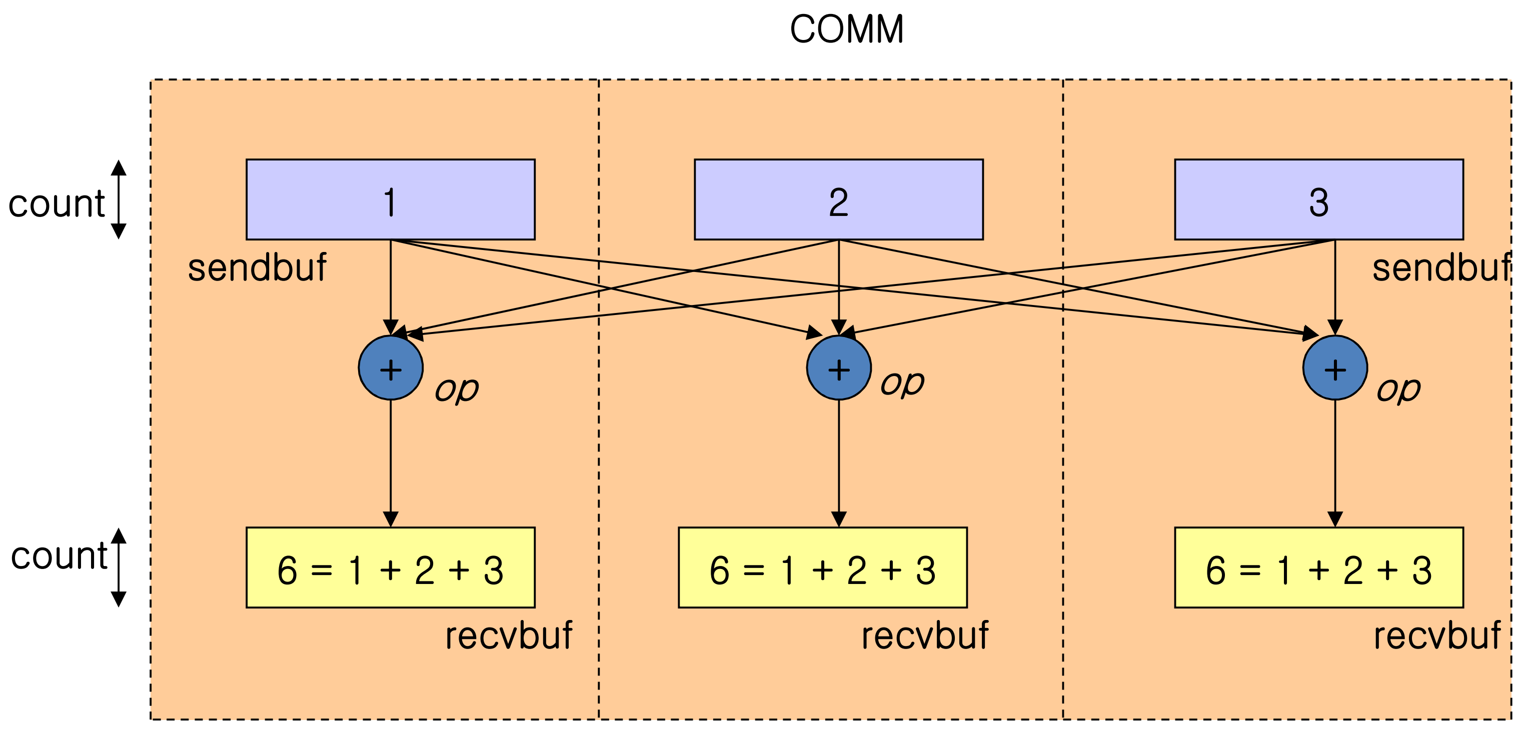


MPI_ALLREDUCE

C	`int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)`
Fortran	`MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, op, comm, ierr)`

- **Combines values from all processes and distributes the result back to all processes**

MPI_ALLREDUCE

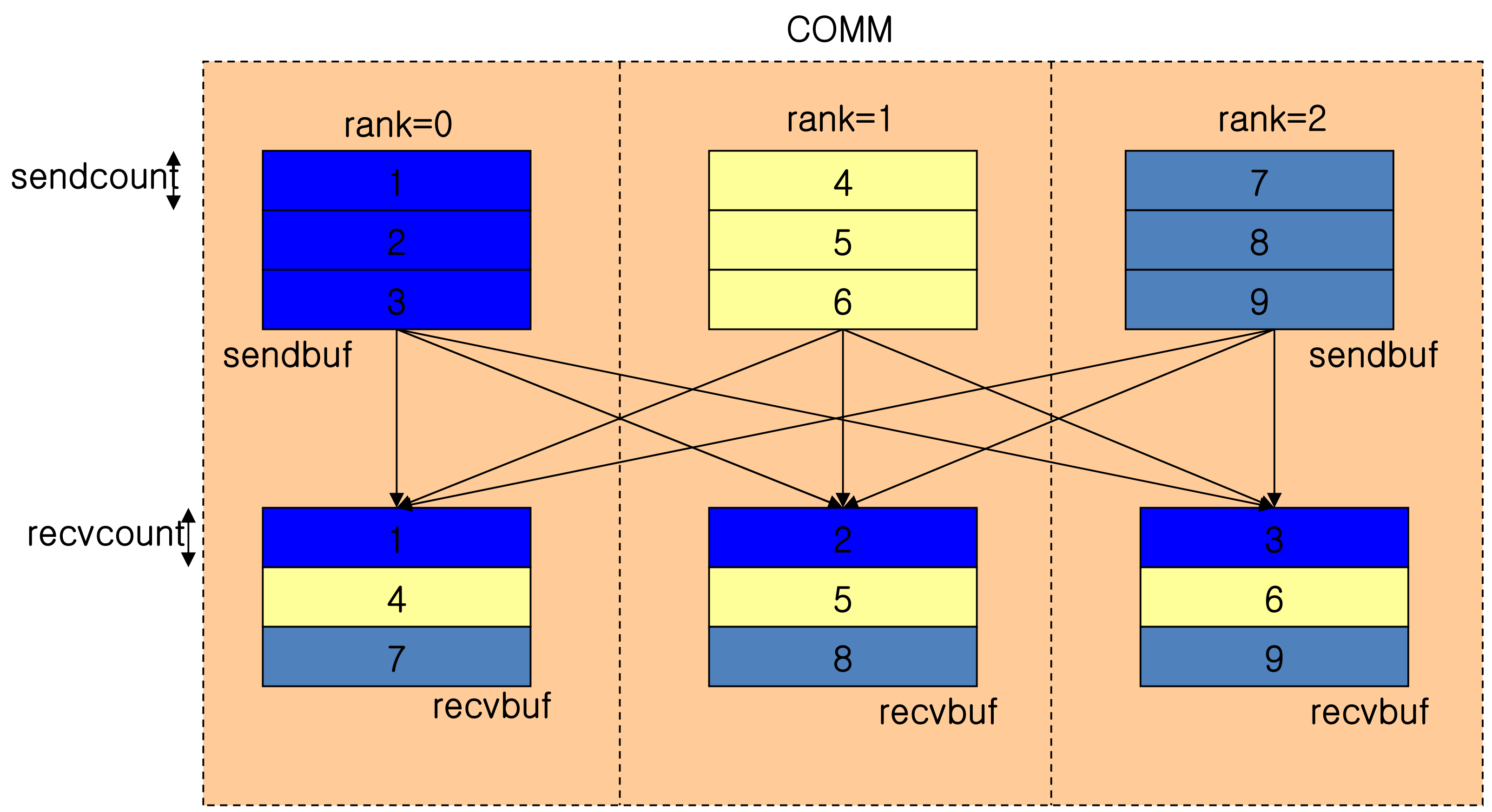


MPI_ALLTOALL

C	`int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLTOALL(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)`

- **MPI_ALLTOALL is an extension of MPI_ALLGATHER to the case where each process sends distinct data to each of the receivers. The j-th block sent from process i is received by process j and is placed in the i-th block of recvbuf.**

MPI_ALLTOALL

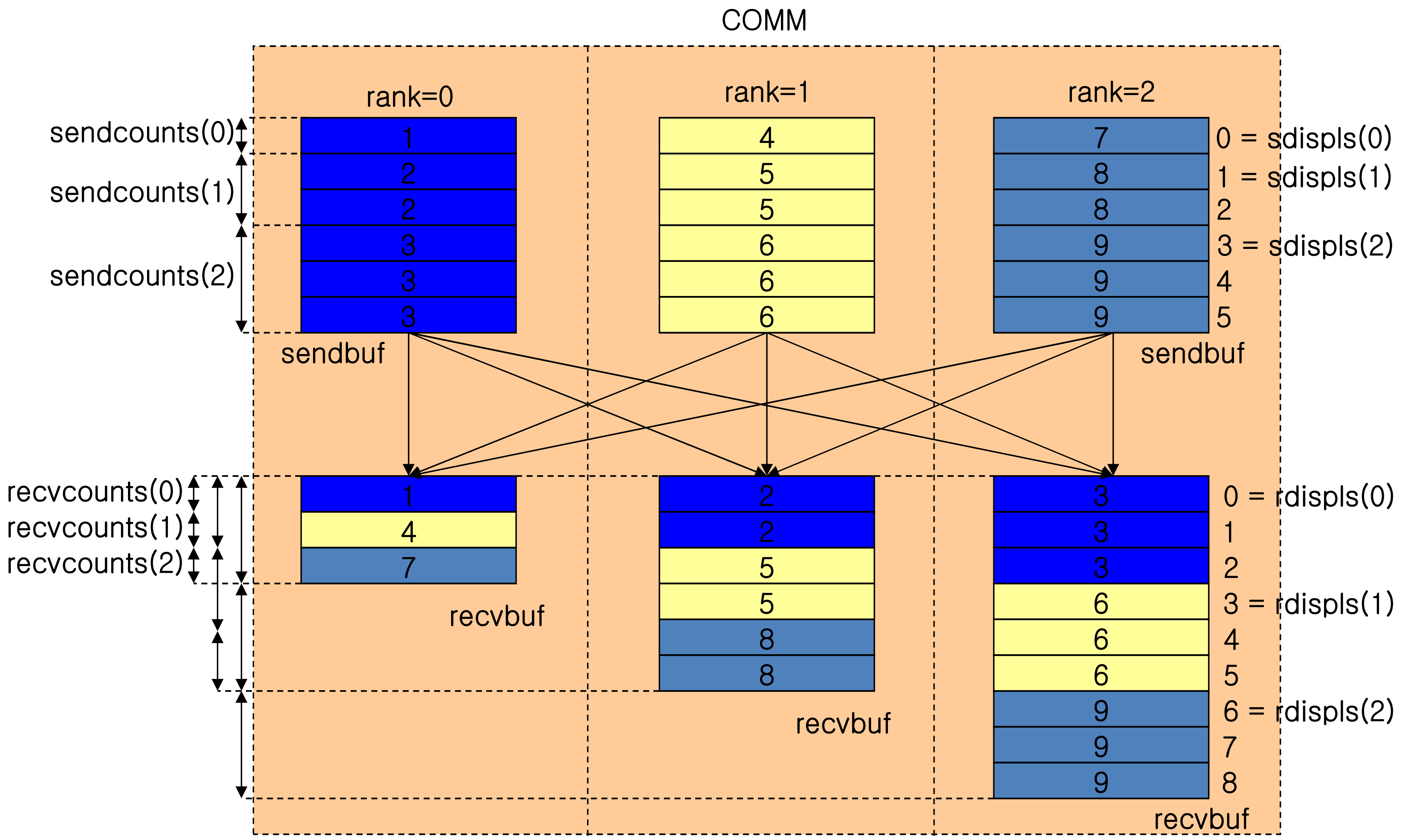


MPI_ALLTOALLV

C	`int MPI_Alltoallv(void *sendbuf, int sendcounts, int sdispls, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int rdispls, MPI_Datatype recvtype, MPI_Comm comm)`
Fortran	`MPI_ALLTOALLV(sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, recvtype, comm, ierr)`

- **MPI_ALLTOALLV adds flexibility to MPI_ALLTOALL in that the location of data for the send is specified by sdispls and the location of the placement of the data on the receive side is specified by rdispls.**

MPI_ALLTOALLV



MPI_BARRIER

C	`int MPI_Barrier(MPI_Comm comm)`
Fortran	`MPI_BARRIER(comm, ierr)`

➢ **Blocks until all processes in the communicator have reached here**

Derived Data Type

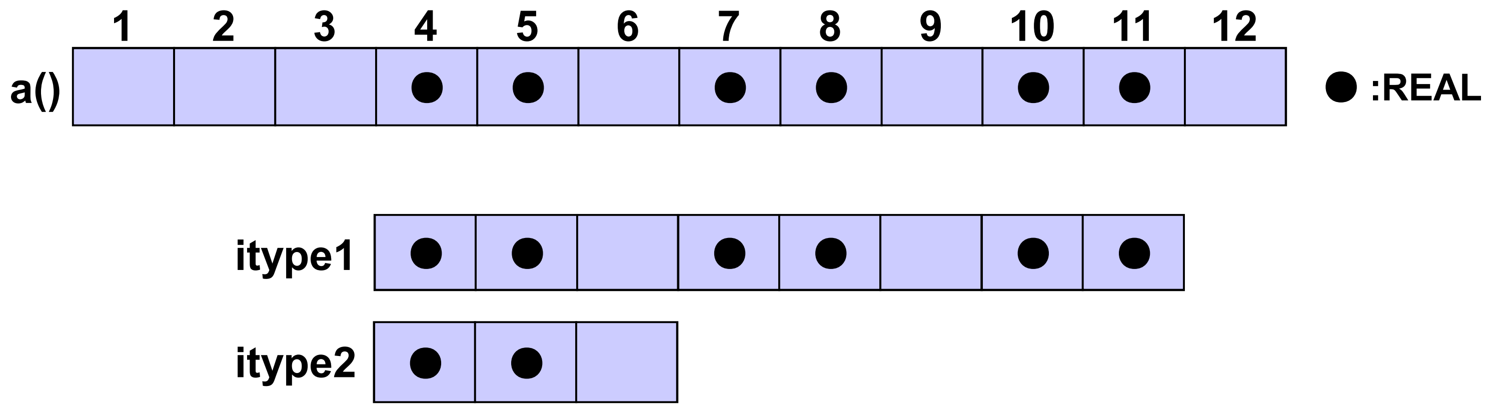
Derived Data Type (1/2)

- A user can make new data types

- A different or noncontiguous data type transfer
  - Noncontiguous data which has same data type
  - Contiguous data which has different data type
  - Noncontiguous data which has different data type

Derived Data Type (2/2)

➢ a(4), a(5), a(7), a(8), a(10), a(11) transfer



– Derived data type – itype1, one element transfer

```
 CALL MPI_SEND(a(4), 1, itype1, idst, itag, MPI_COMM_WORLD,
ierr)
```

– Derived data type – itype2, three element transfer

```
 CALL MPI_SEND(a(4), 3, itype2, idst, itag, MPI_COMM_WORLD,
ierr)
```

Derived Data Type in MPI

- **Construct**
  - MPI_Type_contiguous
  - MPI_Type_(h)vector
  - MPI_Type_struct (MPI_Type_create_struct in 3.0)
- **Commit**
  - Register data type
    - MPI_Type_commit
- **Use**

MPI_TYPE_COMMIT

C	`int MPI_Type_commit (MPI_Datatype *datatype)`
Fortran	`MPI_TYPE_COMMIT (datatype, ierr)`

INTEGER datatype : data type handles(INOUT)

- **Commits the data type**
- **MPI_TYPE_FREE**

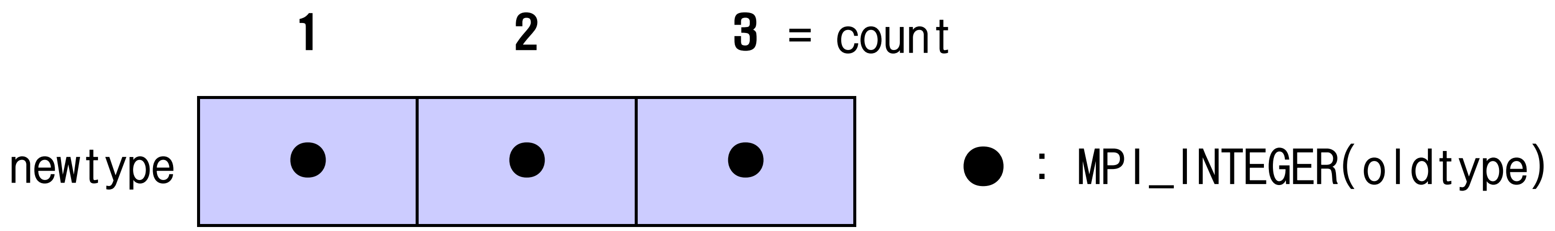
MPI_TYPE_CONTIGUOUS

C	`int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype *newtype)`
Fortran	`MPI_TYPE_CONTIGUOUS (count, oldtype, newtype, ierr)`

INTEGER count : replication count (IN)
INTEGER oldtype : old data type (IN)
INTEGER newtype : new data type (OUT)

➢ **Creates a contiguous data type**

MPI_TYPE_CONTIGUOUS



Lab #8

MPI_TYPE_CONTIGUOUS : C

```
/*type_contiguous*/
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[]){
  int i, myrank, ibuf[20];
  MPI_Datatype inewtype ;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if(myrank==0)  for(i=0; i<20; i++) ibuf[i]=i+1;
  else for(i=0; i<20; i++) ibuf[i]=0;
  MPI_Type_contiguous(3, MPI_INT, &inewtype);
  MPI_Type_commit(&inewtype);
  MPI_Bcast(ibuf, 3, inewtype, 0, MPI_COMM_WORLD);
  printf("%d : ibuf =", myrank);
  for(i=0; i<20; i++) printf(" %d", ibuf[i]);
  printf("\n");
  MPI_Finalize();
}
```

MPI_TYPE_CONTIGUOUS : C

```
PROGRAM type_contiguous
INCLUDE ‘mpif.h’
INTEGER ibuf(20)
INTEGER inewtype
CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
IF (myrank==0) THEN
   DO i=1,20
      ibuf(i) = i
   ENDDO
ENDIF
CALL MPI_TYPE_CONTIGUOUS(3, MPI_INTEGER, inewtype, ierr)
CALL MPI_TYPE_COMMIT(inewtype, ierr)
CALL MPI_BCAST(ibuf, 3, inewtype, 0, MPI_COMM_WORLD, ierr)
PRINT *,’ibuf =’,ibuf
CALL MPI_FINALIZE(ierr)
END
```

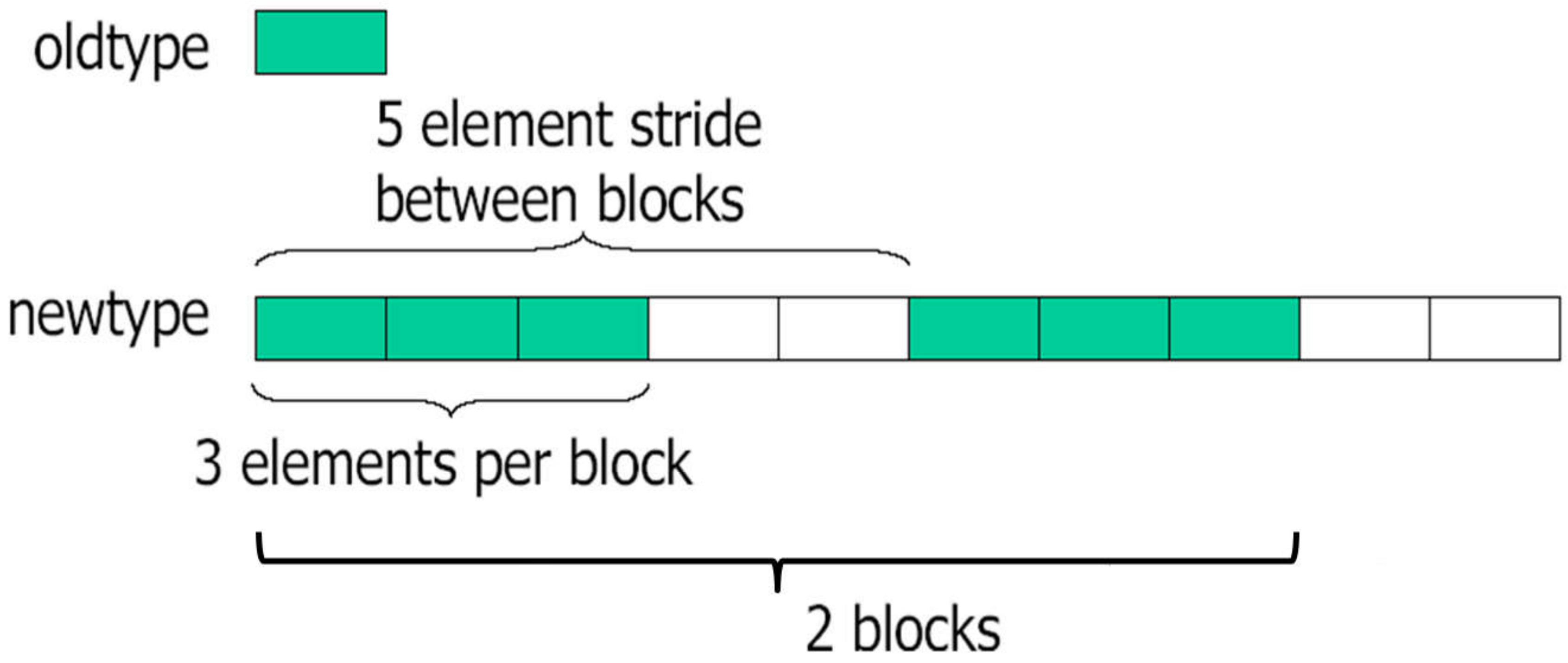
MPI_TYPE_VECTOR (1/2)

C	`int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)`
Fortran	`MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype, ierr)`

INTEGER count : number of blocks
INTEGER blocklength : number of elements in each block
INTEGER stride : number of elements start of each block
INTEGER oldtype : old data type
INTEGER newtype : new data type

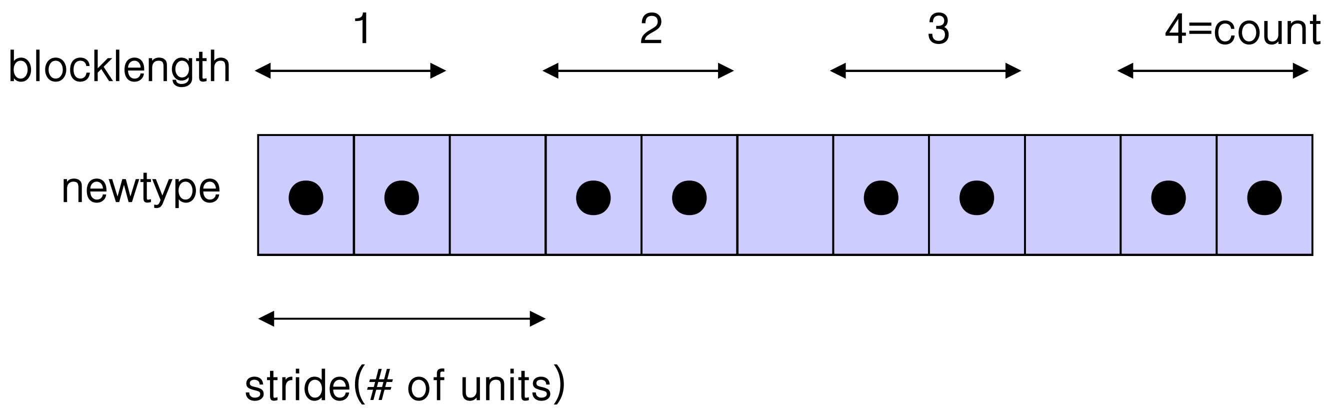
➢ **Creates a new data type which has same interval**

MPI_TYPE_VECTOR (2/2)



- **count = 2**
- **stride = 5**
- **blocklength = 3**

MPI_TYPE_VECTOR



● :MPI_INTEGER(oldtype)

Lab #9

MPI_TYPE_VECTOR : C

```
/*type_vector*/
#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[]){
  int i, myrank, ibuf[20];
  MPI_Datatype inewtype ;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if(myrank==0) for(i=0; i<20; i++) ibuf[i]=i+1;
  else for(i=0; i<20; i++) ibuf[i]=0;
  MPI_Type_vector(4, 2, 3, MPI_INT, &inewtype);
  MPI_Type_commit(&inewtype);
  MPI_Bcast(ibuf, 1, inewtype, 0, MPI_COMM_WORLD);
  printf("%d : ibuf =", myrank);
  for(i=0; i<20; i++) printf(" %d", ibuf[i]);
  printf("\n");
  MPI_Finalize();
}
```

MPI_TYPE_VECTOR : C

```
PROGRAM type_vector
INCLUDE 'mpif.h'
INTEGER ibuf(20), inewtype
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
IF (myrank==0) THEN
DO i=1,20
  ibuf(i) = i
ENDDO
ENDIF
CALL MPI_TYPE_VECTOR(4, 2, 3, MPI_INTEGER, inewtype, ierr)
CALL MPI_TYPE_COMMIT(inewtype, ierr)
CALL MPI_BCAST(ibuf, 1, inewtype, 0, MPI_COMM_WORLD, ierr)
PRINT *, 'ibuf =', ibuf
CALL MPI_FINALIZE(ierr)
END
```

MPI_TYPE_STRUCT (1/2)

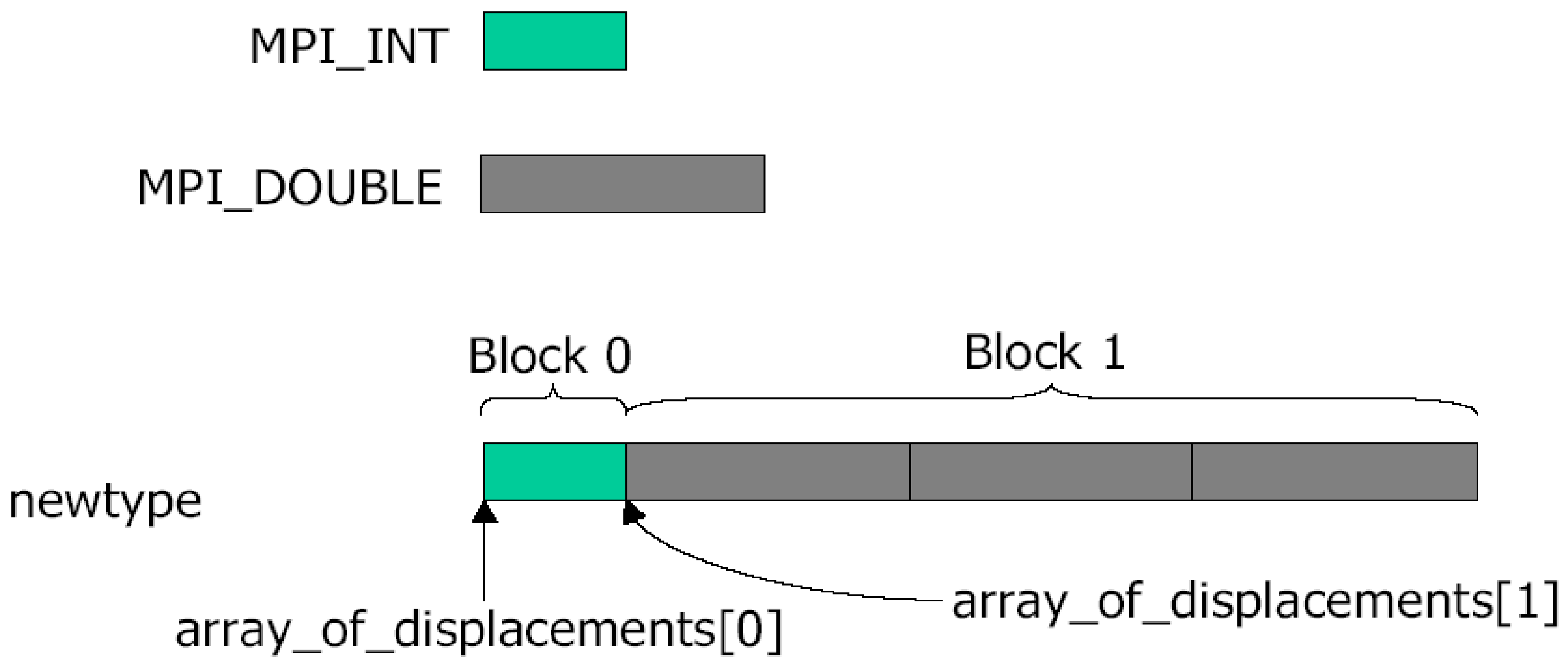
C	`int MPI_Type_struct (int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)`
Fortran	`MPI_TYPE_STRUCT (count, array_of_blocklengths, array_of_displacements, array_of_types, newtype, ierr)`

INTEGER array_of_blocklengths(*) : number of elements in each block (array of non-negative integer) (IN)

INTEGER array_of_displacements(*) : byte displacement of each block (array of integer) (IN)

INTEGER array_of_types(*) : type of elements in each block (array of handles to datatype objects) (IN)

MPI_TYPE_STRUCT (2/2)



- **count = 2**
- **array_of_blocklengths = { 1, 3}**
- **array_of_types = {MPI_INT, MPI_DOUBLE}**
- **array_of_displacements = {0, extent(MPI_INT)}**

How to Parallelize Your Program: Loop

Block Distribution

iteration	1	2	3	4	5	6	7	8	9	10	11	12
rank	0	0	0	1	1	1	2	2	2	3	3	3

Block Distribution

- Suppose when you divide n by p, the quotient is q and the remainder is r.
  - $n = p \times q + r$

- Processes 0..r-1 are assigned *q* + 1 iterations each. The other processes are assigned *q* iterations.
  - $n = r(q+1) + (p-r)q$

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3

Block Distribution: C

```
void  para_range(int  n1,int  n2,  int  nprocs,  int
   myrank, int *ista, int *iend){
    int iwork1, iwork2;
    iwork1 = (n2-n1+1)/nprocs;
    iwork2 = (n2-n1+1) % nprocs;
    *ista= myrank*iwork1 + n1 + min(myrank, iwork2);
    *iend = *ista + iwork1 - 1;
    if(iwork2 > myrank) *iend = *iend + 1;
}

int min(int x, int y){
 int v;
 if (x>=y) v = y;
 else v = x;
 return v;
}
```

Block Distribution: Fortran

```
SUBROUTINE para_range(n1, n2, nprocs, irank,
  ista, iend)
  iwork1 = (n2 - n1 + 1) / nprocs
  iwork2 = MOD(n2 - n1 + 1, nprocs)
  ista = irank * iwork1 + n1 + MIN(irank,
  iwork2)
  iend = ista + iwork1 - 1
  IF (iwork2 > irank) iend = iend + 1
END
```

Cyclic Distribution

```
DO i = n1, n2
   computation
ENDDO
```

```
DO i = n1+myrank, n2, nprocs
   computation
ENDDO
```

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	1	2	3	0	1	2	3	0	1	2	3	0	1

- **More balanced workload for processes than the block distribution**
- **More cache misses than the block distribution**

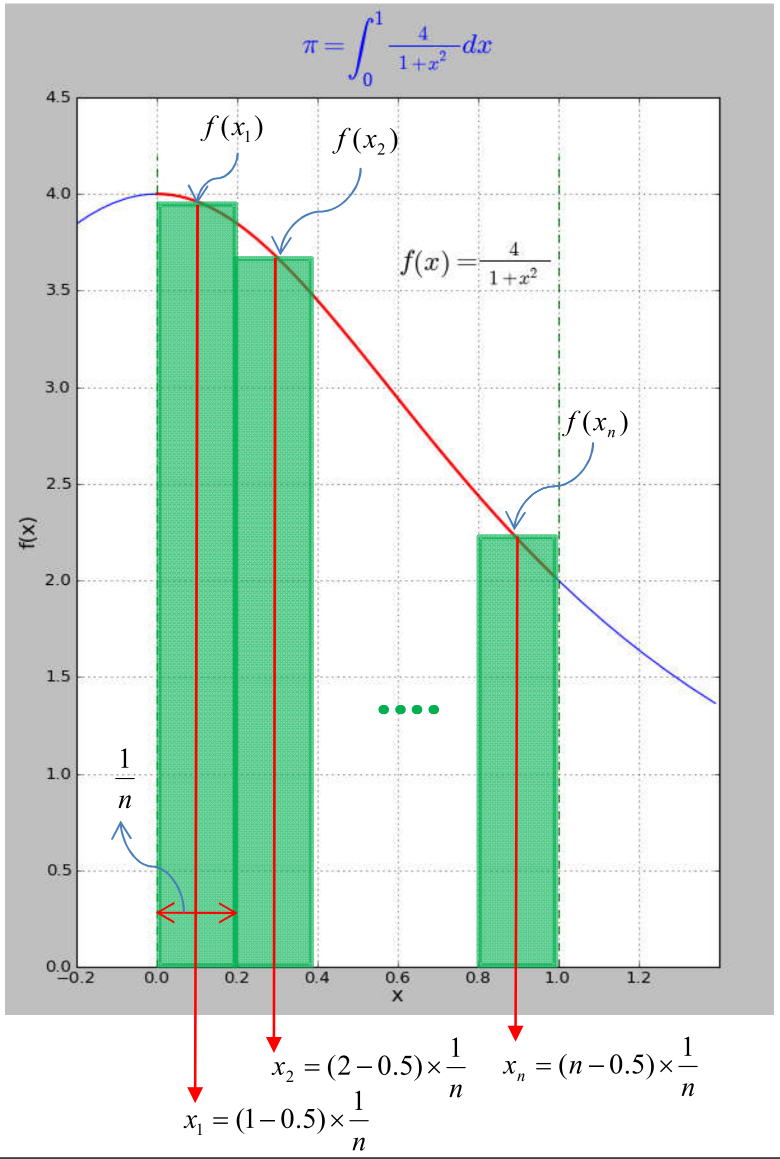
Advanced Lab: PI (Numerical Integration)

- **<Problem>**
  - Get PI using Numerical integration

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

- **<Requirement>**
  - Point to point communication

$$\pi \approx \sum_{i=1}^{n} \frac{4}{1+((i-0.5)\times\frac{1}{n})^2} \times \frac{1}{n}$$



$\pi = \int_0^1 \frac{4}{1+x^2}dx$
$f(x_1)$
$f(x_2)$
$f(x) = \frac{4}{1+x^2}$
$f(x_n)$
$\frac{1}{n}$
f(x)
x
$x_1 = (1-0.5)\times\frac{1}{n}$
$x_2 = (2-0.5)\times\frac{1}{n}$
$x_n = (n-0.5)\times\frac{1}{n}$

Advanced Lab: PI Numerical Integration :serial - C

```
#include <stdio.h>
#include <math.h>
#define num_steps 1000000000

int main(int argc, char *argv[]) {
  double sum, step, x, pi;
  double t1, t2;
  int i;

  sum=0.0;
  step=1./(double)num_steps;

  for(i=1; i<num_steps; i++){
     x = (i-0.5)*step;
     sum = sum + 4.0/(1.0+x*x);
  }

  pi = step*sum;
  printf(" numerical  pi = %.15f \n", pi);
  printf(" analytical pi = %.15f \n", acos(-1.0));
  printf(" Error = %E \n", fabs(acos(-1.0)-pi));
  return 0;
}
```

Advanced Lab: PI Numerical Integration – C(1/3)

```
/*
    Example Name    : pi_integral.c
    Compile         : $ mpicc -g -o pi_integral -Wall pi_integral.c
    Run             : $ mpirun -np 4 -hostfile hosts pi_integral
*/

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

#define   SCOPE     100000000

int main(int argc, char *argv[])
{
    int i, n = SCOPE;
    double sum, step, pi, x, tsum, ROOT = 0;
    int nRank, nProcs;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
    MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

    if (nRank == ROOT) {
         for (i = 1; i < nProcs; i++)
```

Advanced Lab: PI Numerical Integration – C(2/3)

```
                    MPI_Send(&n, 1, MPI_INT, i, 55, MPI_COMM_WORLD);
    }
    else
          MPI_Recv(&n, 1, MPI_INT, ROOT, 55, MPI_COMM_WORLD, &status);

    step = 1.0 / (double)n;

    sum = 0.0;
    tsum = 0.0;

    for (i = nRank; i < n; i += nProcs) {
          x = ((double)i-0.5)*step;
          sum = sum + 4 /(1.0 + x*x);
    }

    if (nRank == ROOT) {
          tsum = sum;

          for (i = 1; i < nProcs; i++) {
                    MPI_Recv(&sum, 1, MPI_DOUBLE, i, 56, MPI_COMM_WORLD, &status);
                    tsum = tsum + sum;
          }

          pi = step * tsum;
```

Advanced Lab: PI Numerical Integration – C(3/3)

```
          printf("--------------------------------------------\n");
          printf("PI = %.15f (Error = %E)\n", pi, fabs(acos(-1.0) - pi));
          printf("--------------------------------------------\n");
     }
     else
          MPI_Send(&sum, 1, MPI_DOUBLE, ROOT, 56, MPI_COMM_WORLD);

     MPI_Finalize();

     return 0;
}
```

Advanced Lab: PI Compile & Run

```
$ mpicc -o pi_integral pi_integral.c

$ mpirun -np 4 -hostfile hosts pi_integral
---------------------------------------------
PI = 3.141592673590217 (Error = 2.000042E-08)
---------------------------------------------
```

Advanced Lab: PI Numerical Integration :Serial - Fortran

```
integer, parameter:: num_steps=1000000000
real(8) sum, step, x, pi;

sum=0.0
step=1./dble(num_steps)

do i=1, num_steps
   x = (i-0.5)*step
   sum = sum + 4.0/(1.0+x*x)
enddo

pi = step*sum
print*, "numerical  pi = ", pi
print*, "analytical pi = ", dacos(-1.d0)
print*, " Error = ", dabs(dacos(-1.d0)-pi)

end
```

Advanced Lab: PI Numerical Integration – Fortran(1/2)

```
! Example Name        : pi_monte.f90
! Compile             : $ mpif90 -g -o pi_integral.x -Wall pi_integral.f90
! Run                 : $ mpirun -np 4 -hostfile hosts pi_integral.x

PROGRAM pi_integral
IMPLICIT NONE
INCLUDE "mpif.h"

    INTEGER*8 ::                i, n = 1000000, ROOT = 0
    DOUBLE PRECISION            sum, step, mypi, x, tsum
    INTEGER                     nRank, nProcs, iErr
    INTEGER                     status(MPI_STATUS_SIZE)

    CALL MPI_INIT(iErr)
    CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nProcs, iErr)
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, nRank, iErr)

    IF (nRank .EQ. ROOT) THEN
         DO i=1, nProcs-1
                 CALL MPI_SEND(n, 1, MPI_INTEGER8, i, 55, MPI_COMM_WORLD, iErr)
         END DO
    ELSE
         CALL MPI_RECV(n, 1, MPI_INTEGER8, ROOT, 55, MPI_COMM_WORLD, status, iErr)
    END IF

    step = (1.0d0 / dble(n))
    sum = 0.0d0
```

Advanced Lab: PI Numerical Integration – Fortran(2/2)

```
    tsum = 0.0d0

    DO i=nRank+1, n, nProcs
        x = (dble(i) - 0.5d0) * step
        sum = sum + 4.d0 / (1.d0 + x*x)
    END DO

    IF (nRank .EQ. ROOT) THEN
        tsum = sum

        DO i=1, nProcs-1
               CALL MPI_RECV(sum, 1, MPI_DOUBLE_PRECISION, i, 56, MPI_COMM_WORLD,
                                    status, iErr)
               tsum = tsum + sum
        END DO

        mypi = step * tsum
        WRITE (*, 400)
        WRITE (*, 100) mypi, dabs(dacos(-1.d0) - mypi)
        WRITE (*, 400)
100 FORMAT ('mypi = ', F17.15, ' (Error = ', E11.5, ')')
400 FORMAT ('----------------------------------------')
    ELSE
        CALL MPI_SEND(sum, 1, MPI_DOUBLE_PRECISION, ROOT, 56, MPI_COMM_WORLD, iErr)
    END IF

    CALL MPI_FINALIZE(iErr)
END
```

Advanced Lab #2 PI Compile & Run

```
$ mpif90 -o pi_integral.x pi_integral.f90

$ mpirun -np 4 -hostfile hosts pi_integral.x
-----------------------------------------
mypi = 3.141592653589903 (Error = 0.11013E-12)
-----------------------------------------
```

Q&A

Thank
you