ICHEC
Irish Centre for High-End Computing

MPI Collective Communication

Irish Centre for High-End Computing (ICHEC)

www.ichec.ie

ICHEC
Irish Centre for High-End Computing

Collective Communication

- Communications involving a group of processes.
- Must be called by all processes in a communicator.
- Examples:
  - Barrier synchronization.
  - Broadcast, scatter, gather.
  - Global sum, global maximum, etc.

Characteristics of Collective Communication

- Optimised Communication routines involving a group of processes
- Collective action over a communicator, i.e. all processes must call the collective routine.
- Synchronization may or may not occur.
- All collective operations are blocking.
- No tags.
- Receive buffers must have exactly the same size as send buffers.

Barrier Synchronization

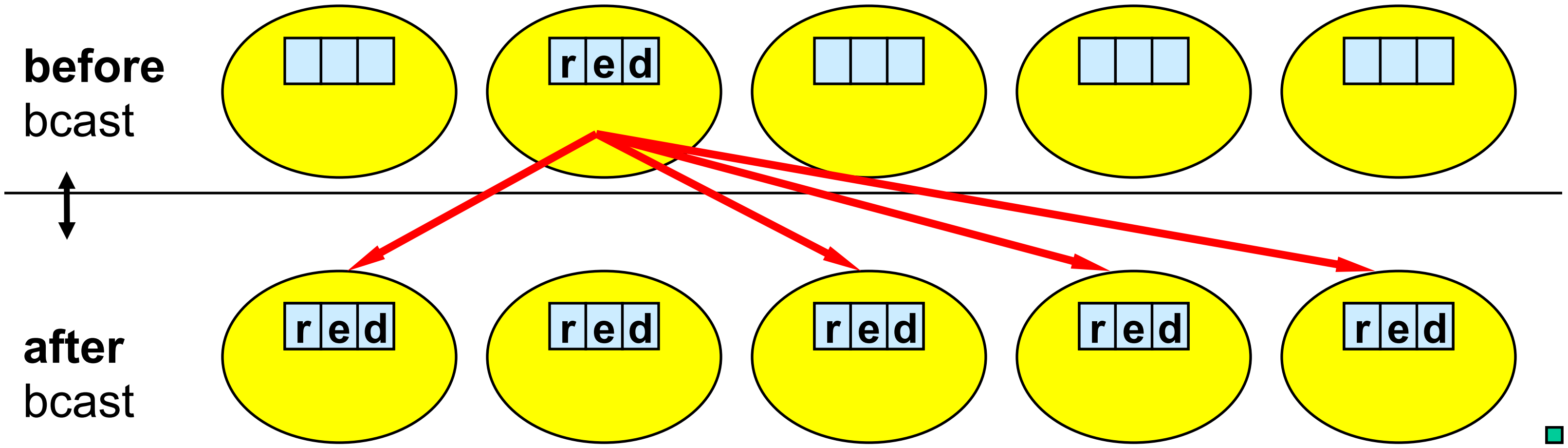
- C: int MPI_Barrier(MPI_Comm comm)
- Fortran: MPI_BARRIER(COMM, *IERROR*)
  INTEGER COMM, IERROR
- MPI_Barrier is normally never needed:
  - all synchronization is done automatically by the data communication:
    - a process cannot continue before it has the data that it needs.
  - if used for debugging:
    - please guarantee, that it is removed in production.

Broadcast

- C: int **MPI_Bcast**(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

- Fortran: **MPI_Bcast**(BUF, COUNT, DATATYPE, ROOT, COMM, IERROR)

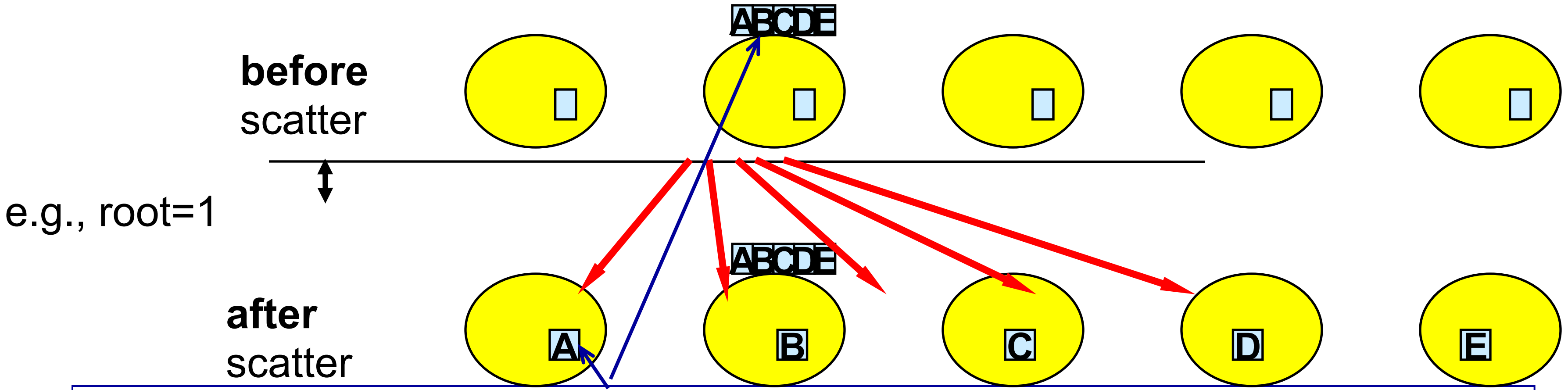
```
<type> BUF(*)
INTEGER COUNT, DATATYPE, ROOT
INTEGER COMM, IERROR
```



e.g., root=1

- **rank of the sending process (i.e., root process)**
- **must be given identically by all processes**

Scatter

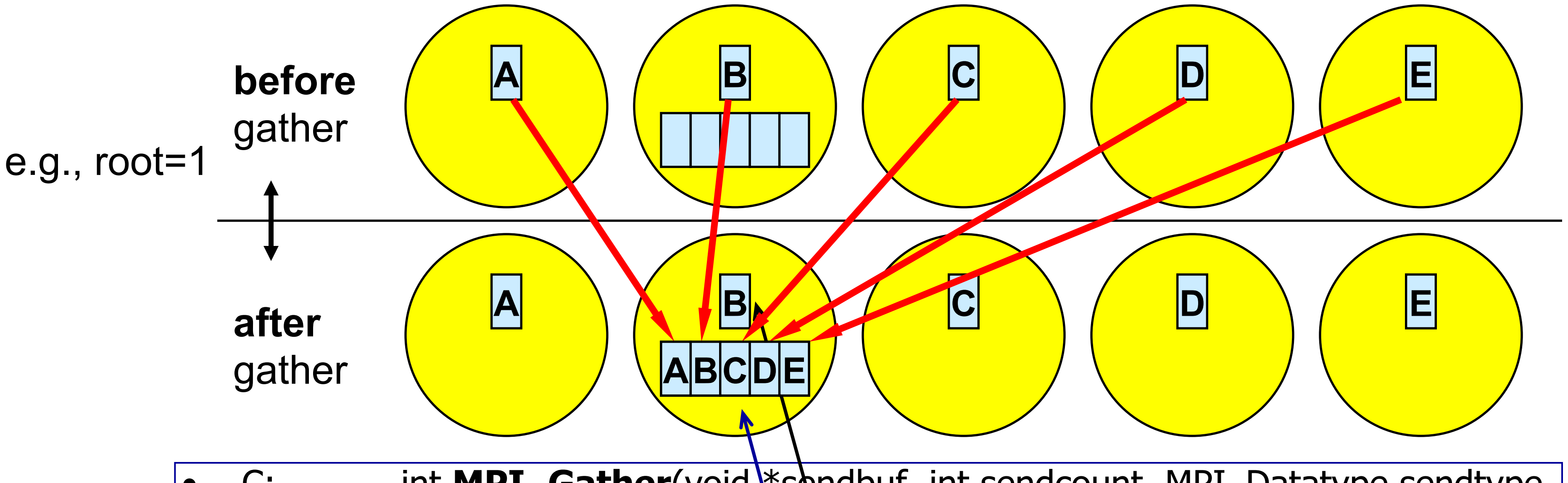


- C: int **MPI_Scatter**(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- Fortran: **MPI_SCATTER**(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)

```
<type>    SENDBUF(*), RECVBUF(*)
INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE
INTEGER ROOT, COMM, IERROR
```

Gather

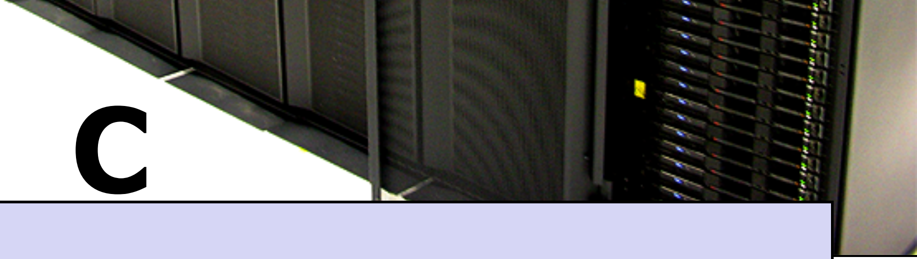


- C: int **MPI_Gather**(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- Fortran: **MPI_GATHER**(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)

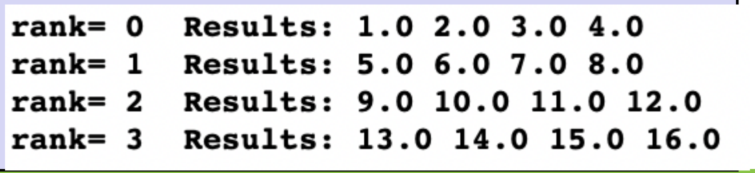
```
<type>    SENDBUF(*), RECVBUF(*)
INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE
INTEGER   ROOT, COMM, IERROR
```

Irish Centre for High-End Computing



C

```
#include <stdio.h>
#include <mpi.h>
#define SIZE 4

int main(int argc, char **argv){
   int myRank, uniSize, ierror, source;
   float sendbuf[SIZE][SIZE] = {
   {1.0, 2.0, 3.0, 4.0},
   {5.0, 6.0, 7.0, 8.0},
   {9.0, 10.0, 11.0, 12.0},
   {13.0, 14.0, 15.0, 16.0}  };
   float recvbuf[SIZE];

   ierror=MPI_Init(&argc,&argv);
   ierror=MPI_Comm_rank(MPI_COMM_WORLD,&myRank);
   ierror=MPI_Comm_Size(MPI_COMM_WORLD,&uniSize);
   if (uniSize == SIZE) {
      source = 0;
      ierror=MPI_Scatter(sendbuf,SIZE,MPI_FLOAT,recvbuf,SIZE, MPI_FLOAT,source,MPI_COMM_WORLD);
      printf("rank= %d  Results: %f %f %f %f\n", myRank,recvbuf[0],recvbuf[1],recvbuf[2],recvbuf[3]);
   }
    ierror=MPI_Finalize();
    return 0;
}
```



```
rank= 0  Results: 1.0 2.0 3.0 4.0
rank= 1  Results: 5.0 6.0 7.0 8.0
rank= 2  Results: 9.0 10.0 11.0 12.0
rank= 3  Results: 13.0 14.0 15.0 16.0
```

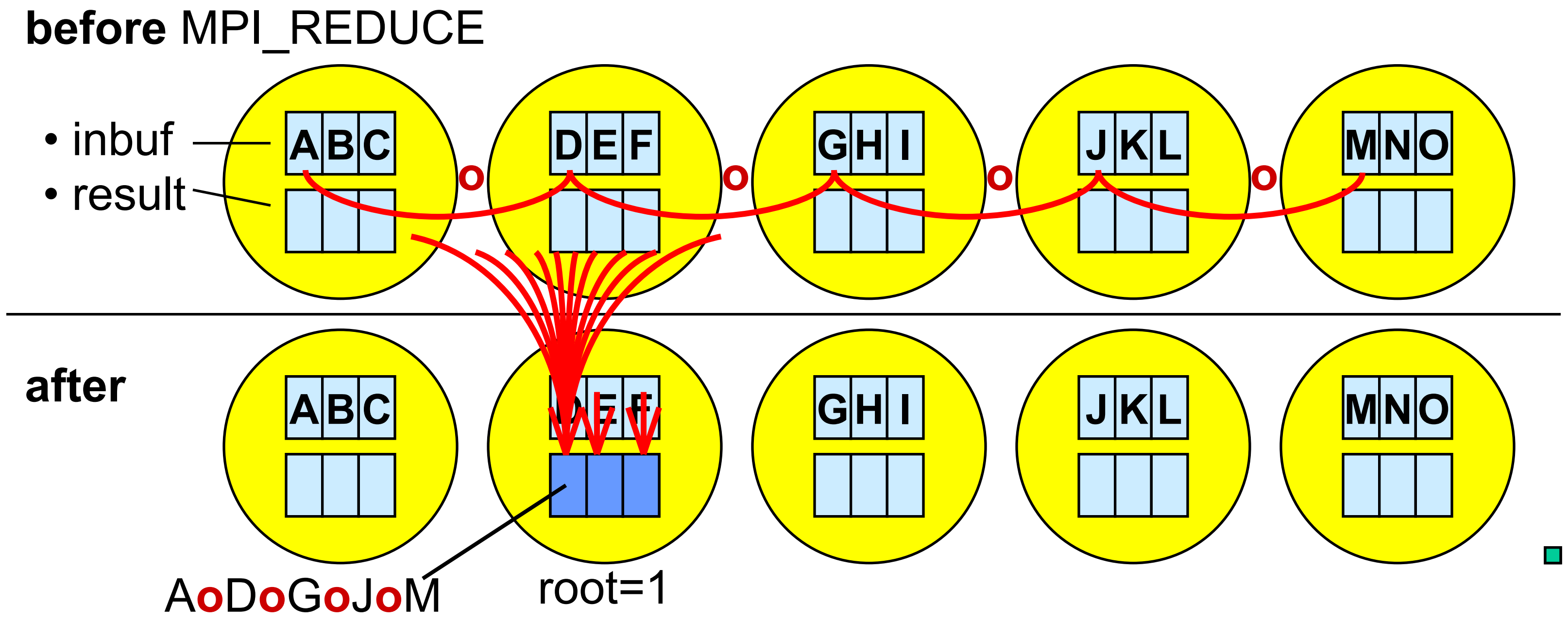
Fortran

```
program testMPI
use mpi
implicit none

integer :: myRank,uniSize,ierror, source, SIZE
parameter(SIZE=4)
real*4 sendbuf(SIZE,SIZE), recvbuf(SIZE)
data sendbuf /1.0, 2.0, 3.0, 4.0, &
           5.0, 6.0, 7.0, 8.0, &
           9.0, 10.0, 11.0, 12.0, &
           13.0, 14.0, 15.0, 16.0 /

call MPI_Init(ierror)
call MPI_Comm_rank(MPI_COMM_WORLD,myRank,ierror)
call MPI_Comm_Size(MPI_COMM_WORLD,uniSize,ierror)
if (uniSize .eq. SIZE) then
   source = 0
   call MPI_SCATTER(sendbuf,SIZE,MPI_REAL,recvbuf,SIZE,MPI_REAL,source,MPI_COMM_WORLD,ierr)
   print *, 'rank= ‘,myRank,' Results: ',recvbuf
endif
call MPI_Finalize(ierror)
end program testMPI
```

Global Reduction Operations

- To perform a global reduce operation across all members of a group.
- $d_0 \ \mathbf{o}\ d_1 \ \mathbf{o}\ d_2 \ \mathbf{o}\ d_3 \ \mathbf{o}\ \ldots \ \mathbf{o}\ d_{s-2} \ \mathbf{o}\ d_{s-1}$
  - d_i = data in process rank i
    - single variable, or
    - vector
  - **o** = associative operation
  - Example:
    - global sum or product
    - global maximum or minimum
    - global user-defined operation
- floating point rounding may depend on usage of associative law:
  - $[(d_0 \ \mathbf{o}\ d_1) \ \mathbf{o}\ (d_2 \ \mathbf{o}\ d_3)] \ \mathbf{o}\ [\ldots \ \mathbf{o}\ (d_{s-2} \ \mathbf{o}\ d_{s-1})]$
  - $((((((d_0 \ \mathbf{o}\ d_1) \ \mathbf{o}\ d_2) \ \mathbf{o}\ d_3) \ \mathbf{o}\ \ldots\ ) \ \mathbf{o}\ d_{s-2}) \ \mathbf{o}\ d_{s-1})$

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, IERROR)

```
<type> SENDBUF(*), RECVBUF(*)

INTEGER COUNT, DATATYPE, OP, ROOT, COMM, IERROR)
```

ICHEC Irish Centre for High-End Computing

Predefined Reduction Operation Handles

Predefined operation handle	**Function**
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location of the maximum
MPI_MINLOC	Minimum and location of the minimum

User-Defined Reduction Operations

- Operator handles
  - predefined – see table above
  - user-defined
- User-defined function:
  - C: typedef void **MPI_User_function** (void *invec, void *inoutvec, int *len, MPI_Datatype *datatype)
  - Fortran: FUNCTION **USER_FUNCTION** (INVEC(*), INOUTVEC(*), LEN, TYPE)

    <type> INVEC(LEN), INOUTVEC(LEN)
    INTEGER LEN, TYPE
- Registering a user-defined reduction function:
  - C: **MPI_Op_create**(MPI_User_function *func, int commute, MPI_Op **op*)
  - Fortran: **MPI_OP_CREATE**(FUNC, COMMUTE, OP, IERROR)
- COMMUTE tells the MPI library whether FUNC is commutative.

Variants of Reduction Operations

- MPI_ALLREDUCE
  - no root,
  - returns the result in all processes
- MPI_REDUCE_SCATTER
  - result vector of the reduction operation
    is scattered to the processes into the real result buffers
- MPI_SCAN
  - prefix reduction
  - result at process with rank i :=
    reduction of inbuf-values from rank 0 to rank i