Lecture 6: MPI Collectives

CMSE 822: Parallel Computing

Prof. Sean M. Couch

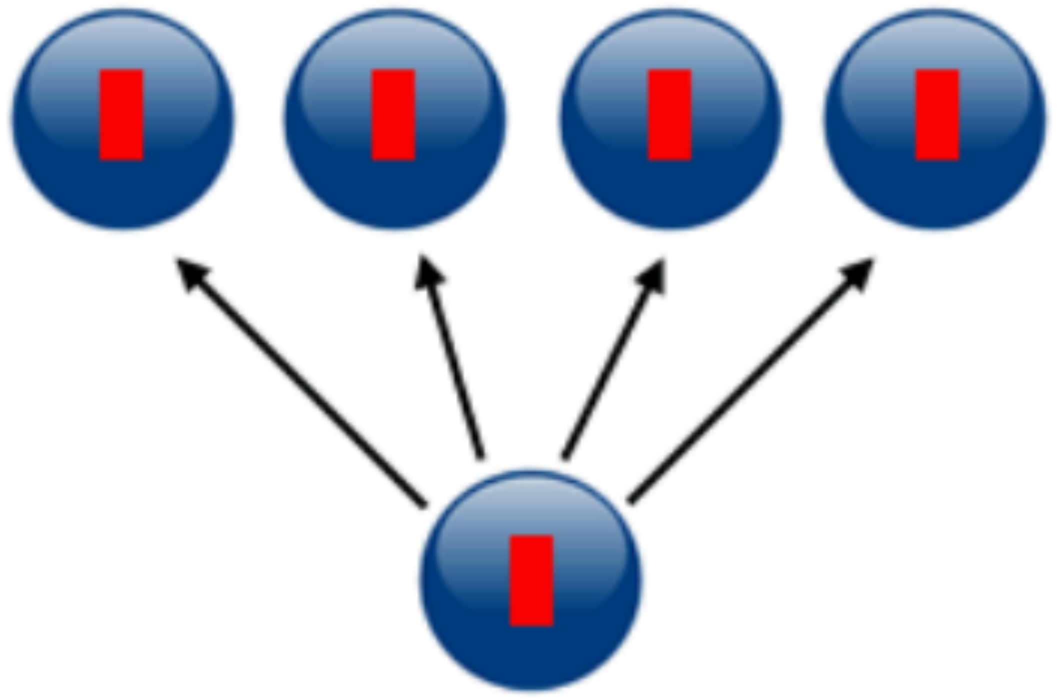
Brief MPI Tutorial

See https://computing.llnl.gov/tutorials/mpi/

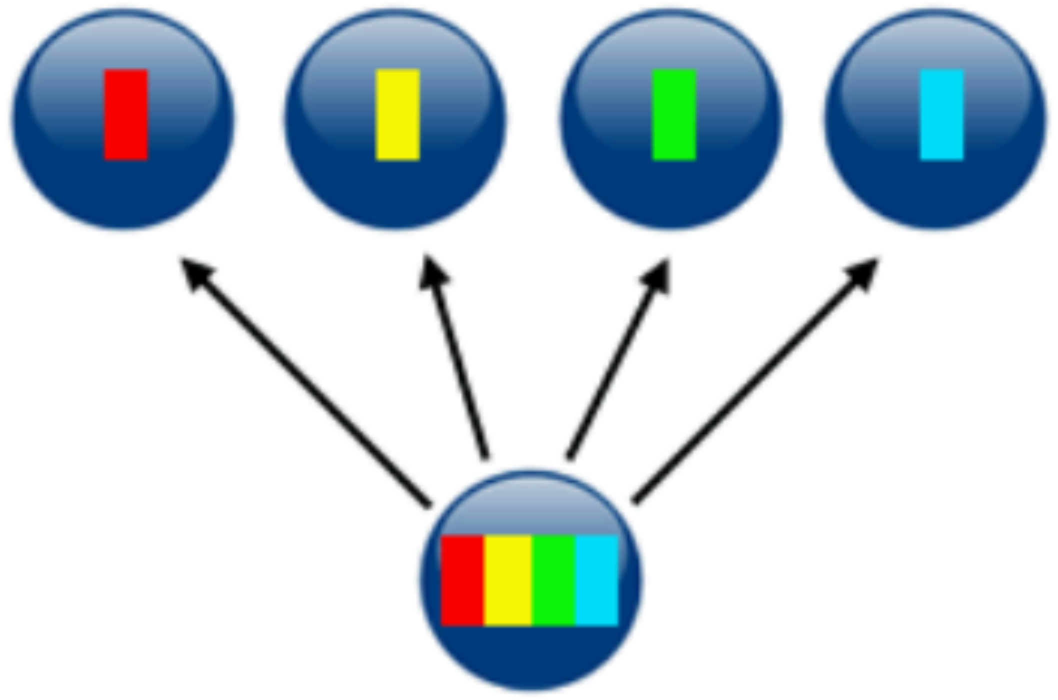
also: http://www.mpi-forum.org/docs/

MPI Collectives

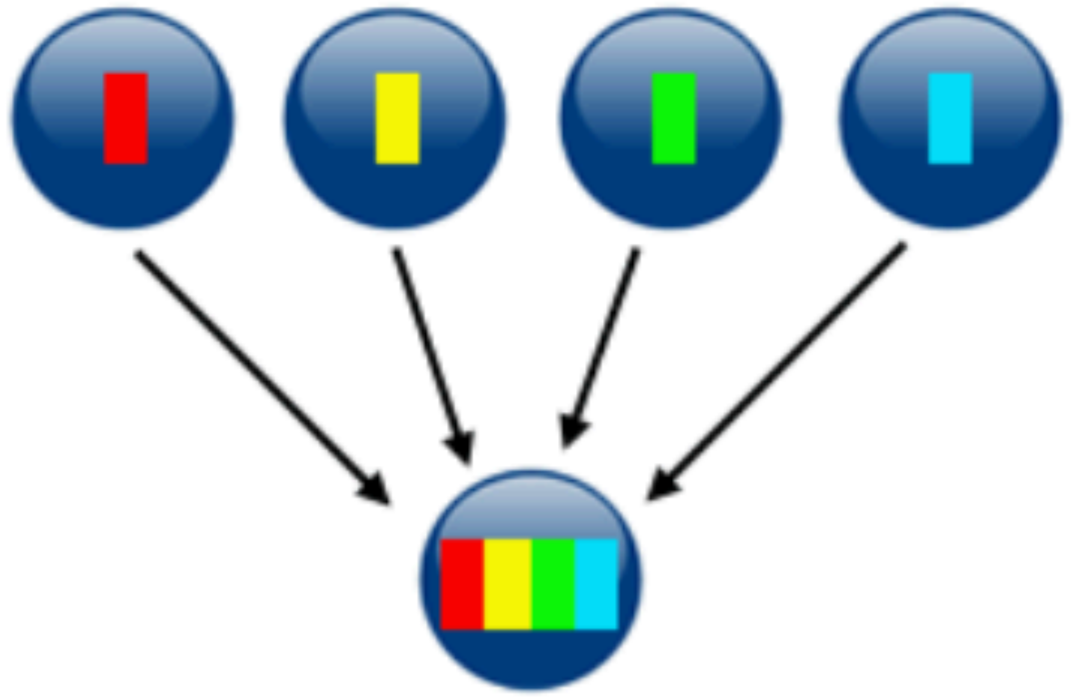
- **Types of Collective Operations:**
  - **Synchronization** - processes wait until all members of the group have reached the synchronization point.
  - **Data Movement** - broadcast, scatter/gather, all to all.
  - **Collective Computation** (reductions) - one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.



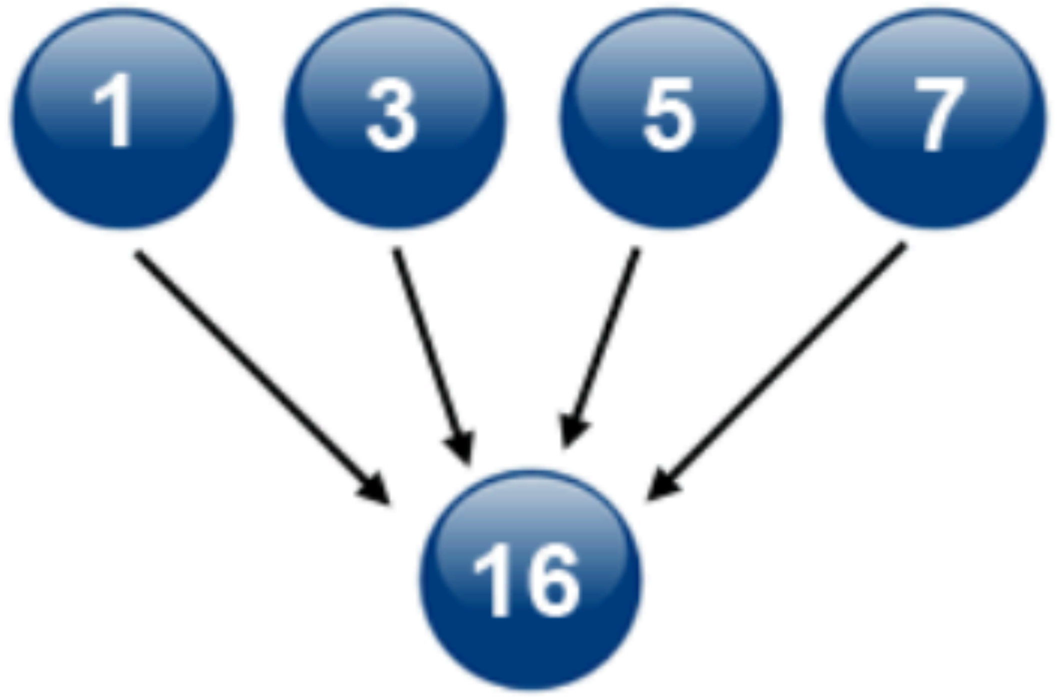
broadcast



scatter

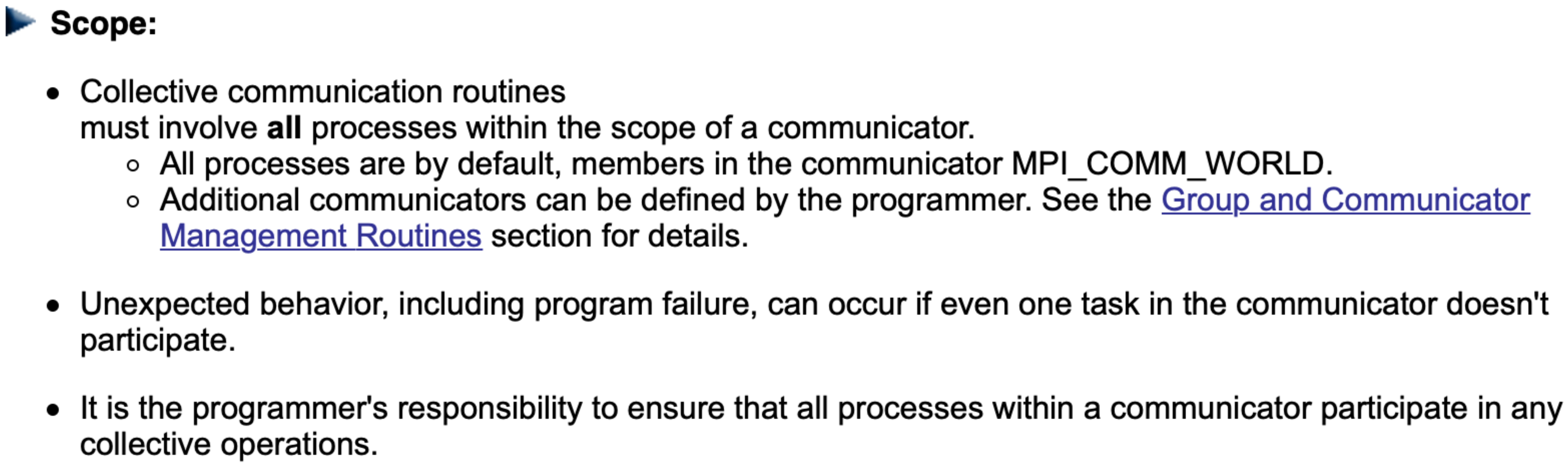


gather



reduction

MPI Collectives



Scope:

- Collective communication routines
  must involve **all** processes within the scope of a communicator.
  - All processes are by default, members in the communicator MPI_COMM_WORLD.
  - Additional communicators can be defined by the programmer. See the Group and Communicator Management Routines section for details.

- Unexpected behavior, including program failure, can occur if even one task in the communicator doesn't participate.

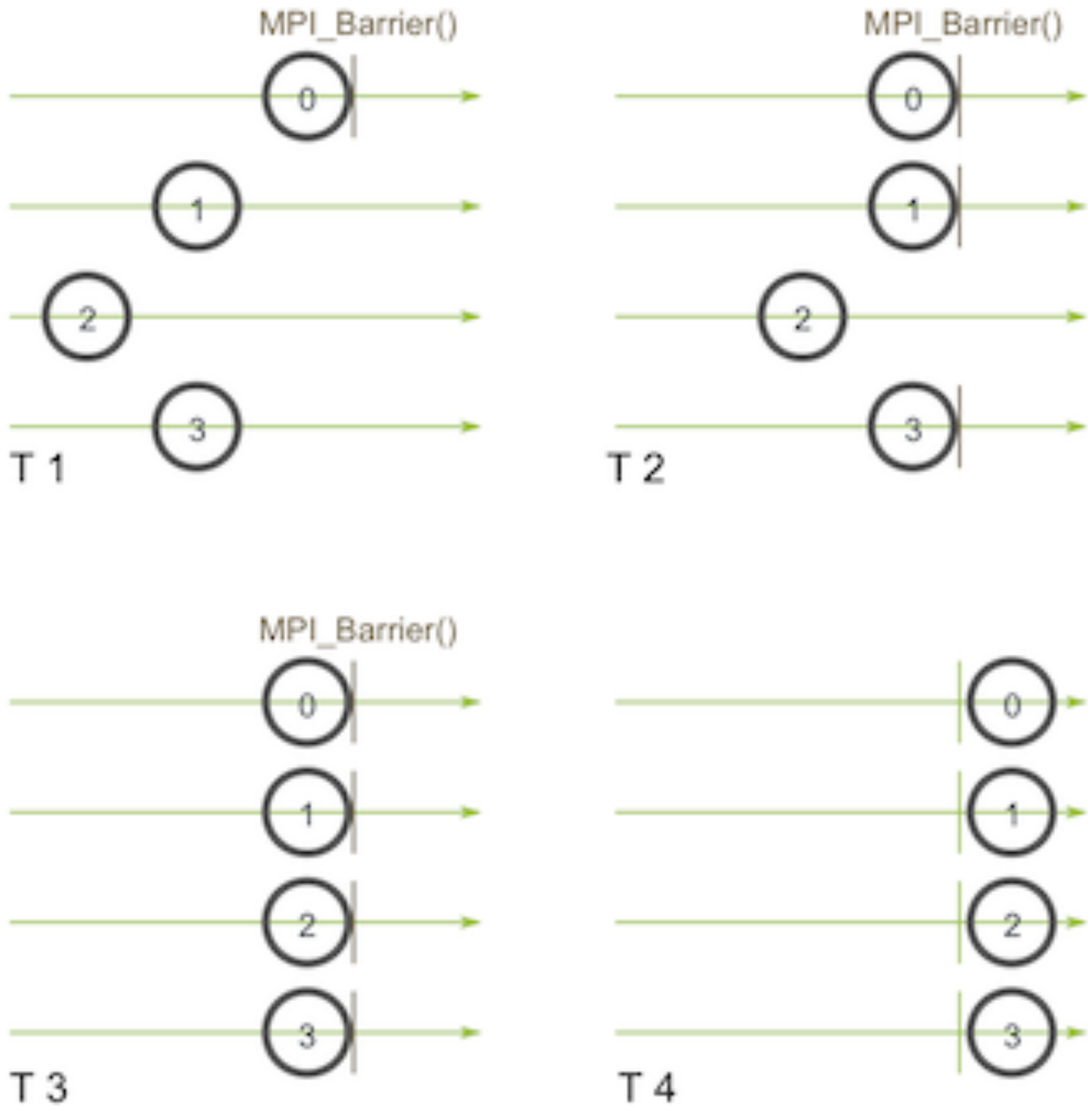
- It is the programmer's responsibility to ensure that all processes within a communicator participate in any collective operations.

MPI Collectives

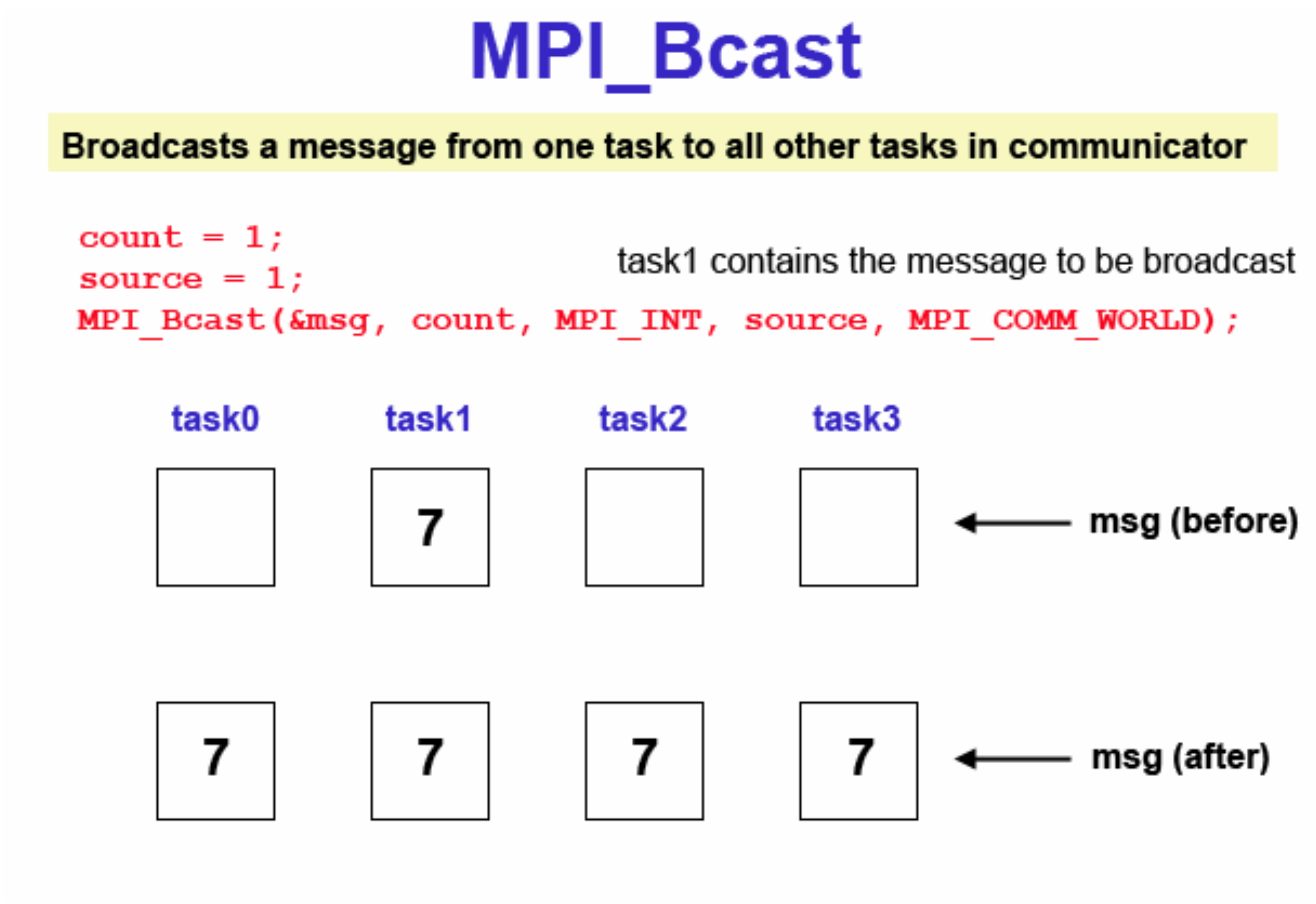
MPI_Barrier

Synchronization operation. Creates a barrier synchronization in a group. Each task, when reaching the MPI_Barrier call, blocks until all tasks in the group reach the same MPI_Barrier call. Then all tasks are free to proceed.

```
MPI_Barrier (comm)
MPI_BARRIER (comm,ierr)
```



MPI Collectives



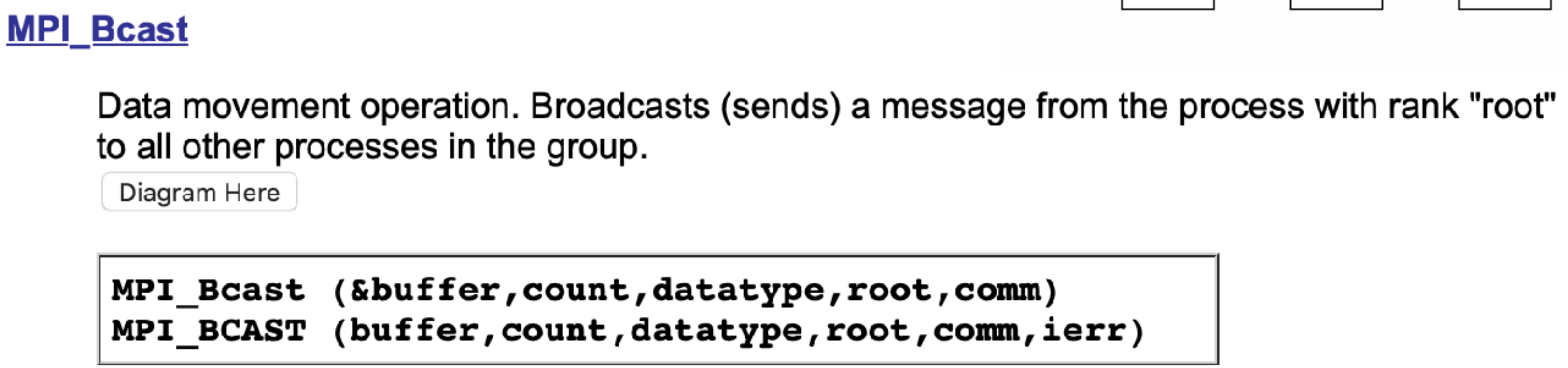
MPI_Bcast

Broadcasts a message from one task to all other tasks in communicator

```
count = 1;
source = 1;
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

task1 contains the message to be broadcast

task0	task1	task2	task3	
	7			← msg (before)
7	7	7	7	← msg (after)



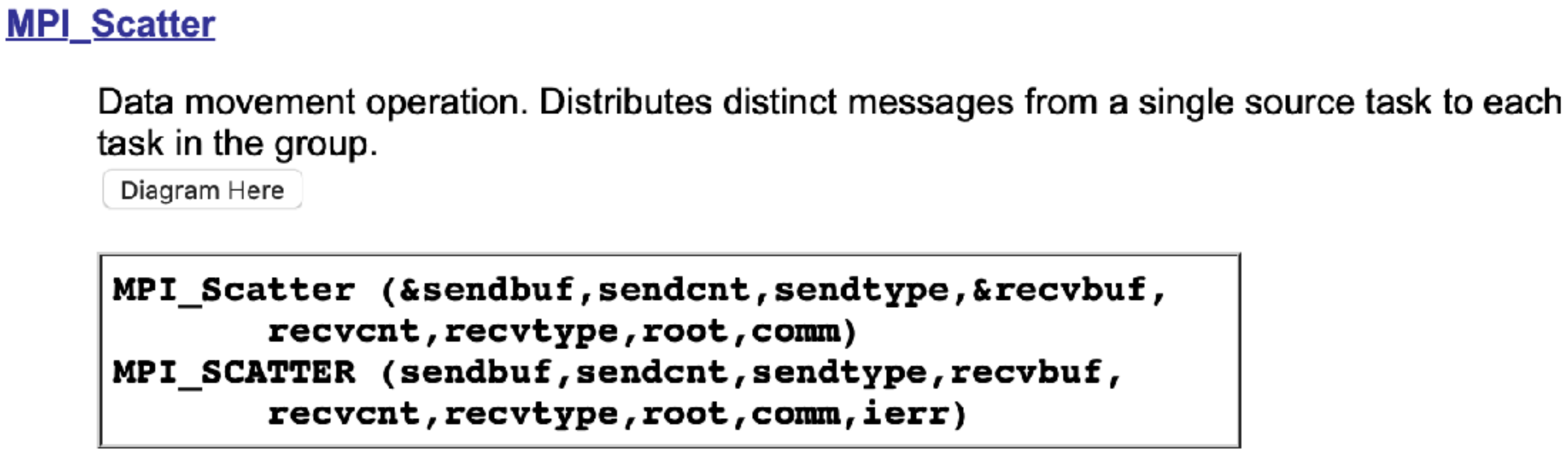
MPI_Bcast

Data movement operation. Broadcasts (sends) a message from the process with rank "root" to all other processes in the group.

Diagram Here

```
MPI_Bcast (&buffer,count,datatype,root,comm)
MPI_BCAST (buffer,count,datatype,root,comm,ierr)
```

MPI Collectives

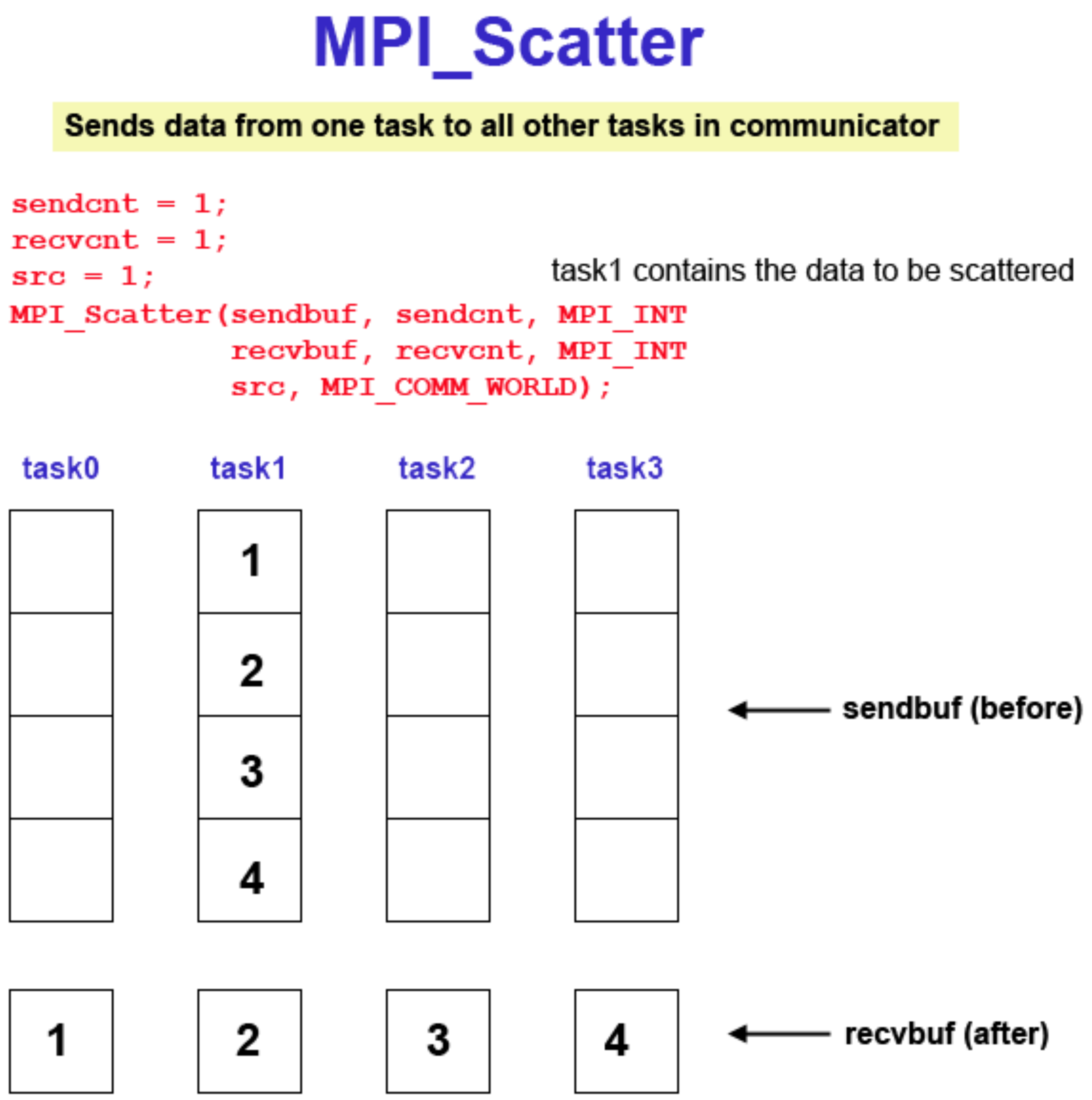


MPI_Scatter

Data movement operation. Distributes distinct messages from a single source task to each task in the group.

Diagram Here

```
MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf,
       recvcnt,recvtype,root,comm)
MPI_SCATTER (sendbuf,sendcnt,sendtype,recvbuf,
       recvcnt,recvtype,root,comm,ierr)
```



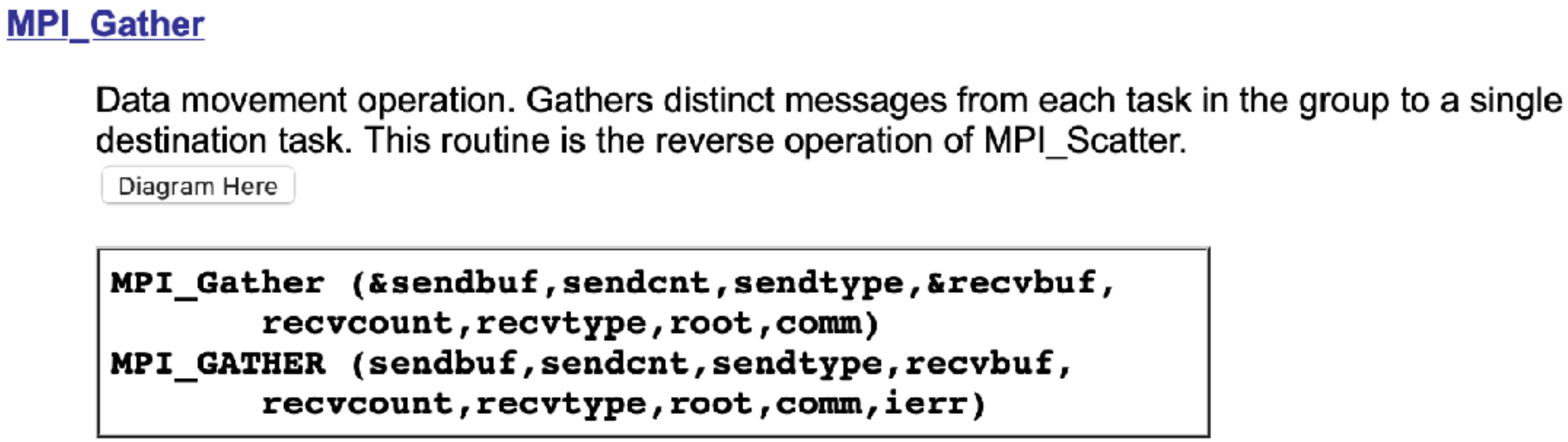
MPI_Scatter

Sends data from one task to all other tasks in communicator

```
sendcnt = 1;
recvcnt = 1;
src = 1;
MPI_Scatter(sendbuf, sendcnt, MPI_INT
            recvbuf, recvcnt, MPI_INT
            src, MPI_COMM_WORLD);
```

task1 contains the data to be scattered

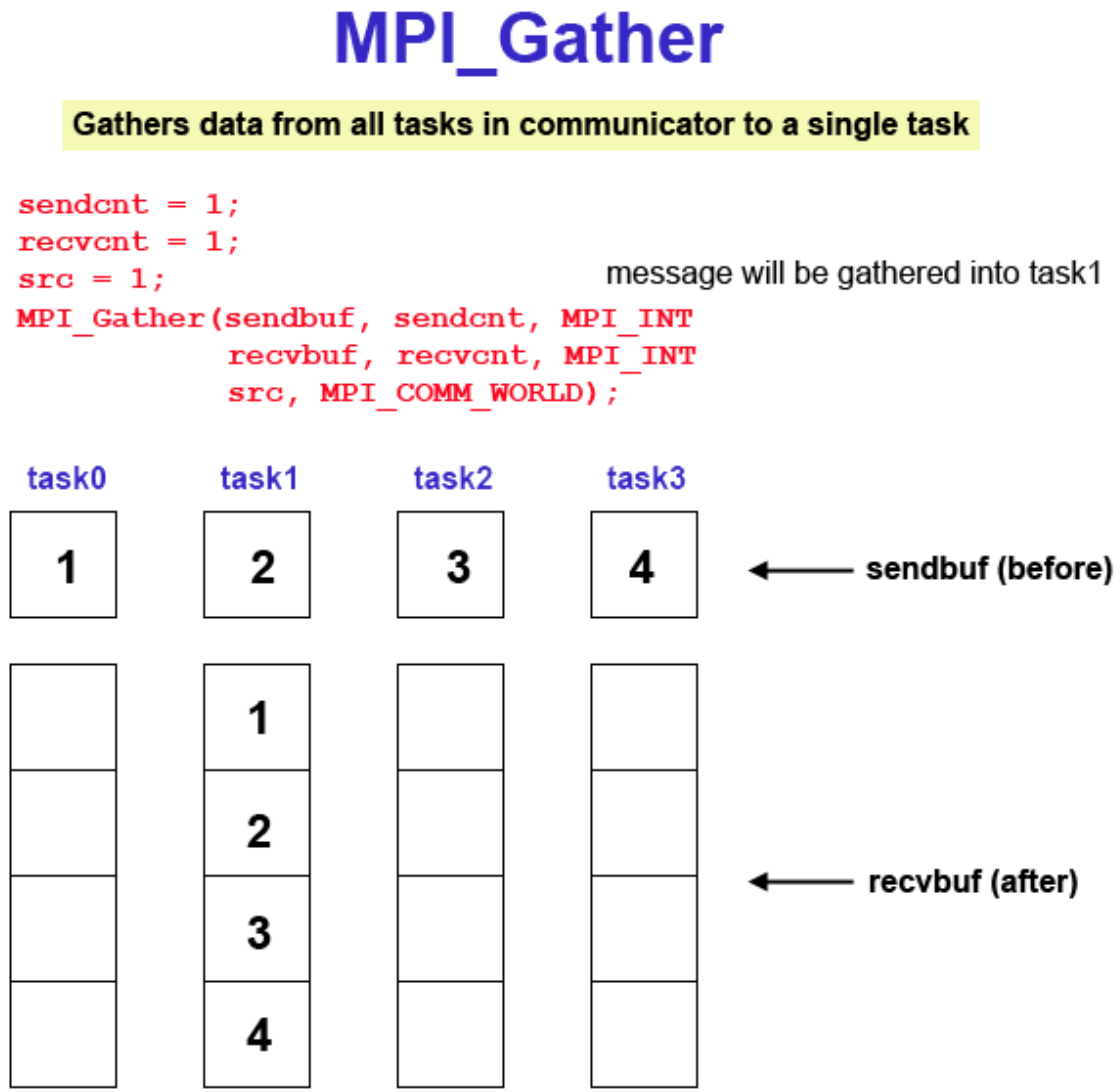
MPI Collectives



MPI_Gather

Data movement operation. Gathers distinct messages from each task in the group to a single destination task. This routine is the reverse operation of MPI_Scatter.

```
MPI_Gather (&sendbuf,sendcnt,sendtype,&recvbuf,
       recvcount,recvtype,root,comm)
MPI_GATHER (sendbuf,sendcnt,sendtype,recvbuf,
       recvcount,recvtype,root,comm,ierr)
```



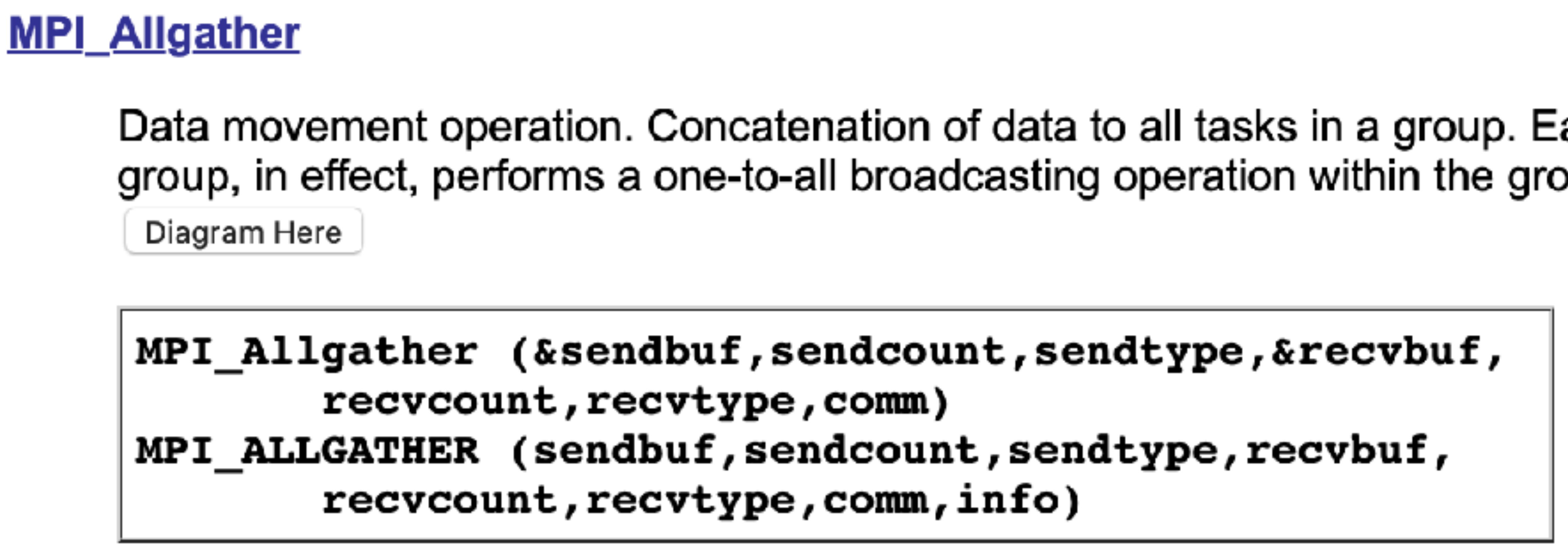
MPI_Gather

Gathers data from all tasks in communicator to a single task

```
sendcnt = 1;
recvcnt = 1;
src = 1;
MPI_Gather(sendbuf, sendcnt, MPI_INT
           recvbuf, recvcnt, MPI_INT
           src, MPI_COMM_WORLD);
```

message will be gathered into task1

MPI Collectives

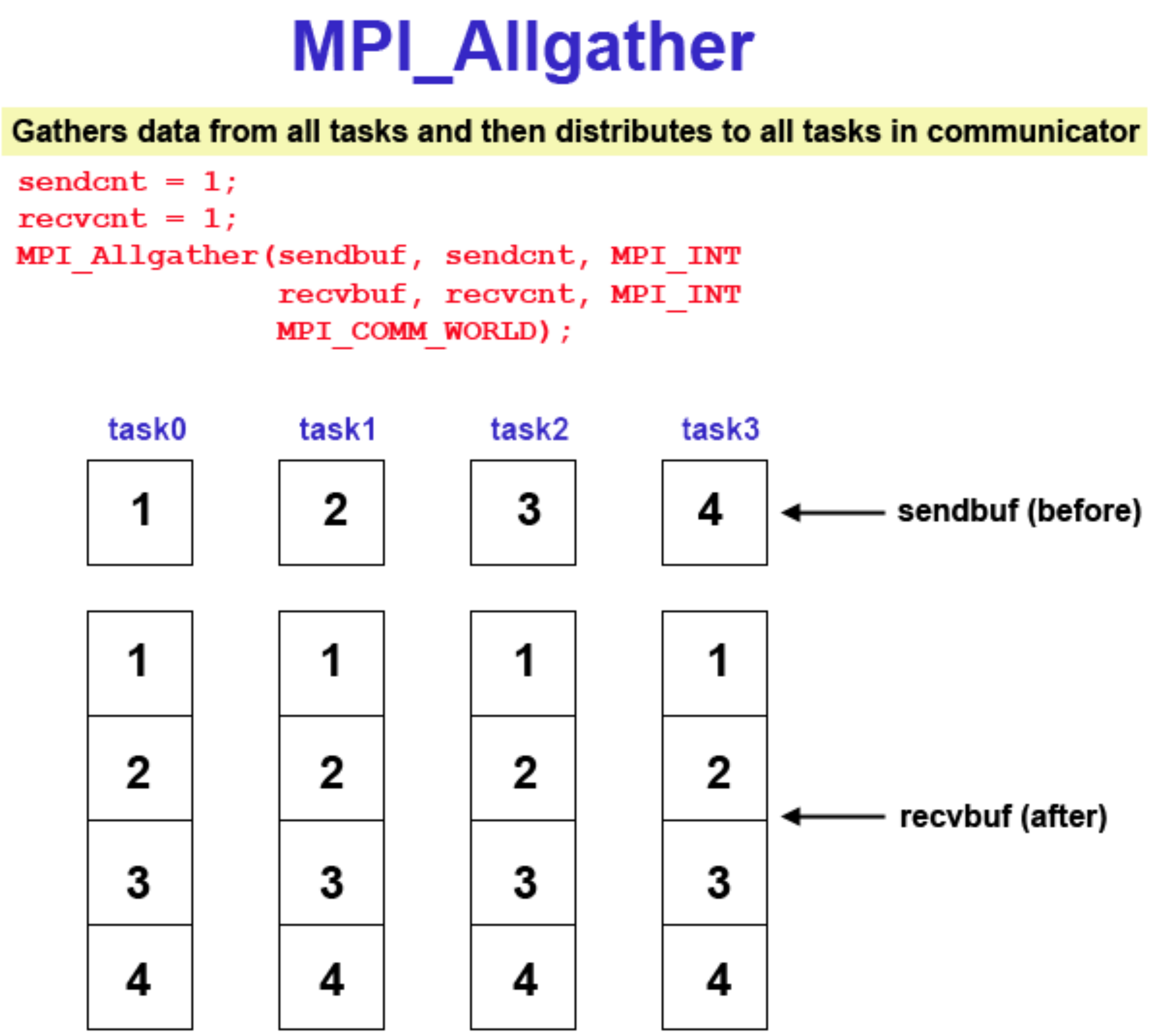


MPI_Allgather

Data movement operation. Concatenation of data to all tasks in a group. Each task in the group, in effect, performs a one-to-all broadcasting operation within the group.

Diagram Here

```
MPI_Allgather (&sendbuf,sendcount,sendtype,&recvbuf,
       recvcount,recvtype,comm)
MPI_ALLGATHER (sendbuf,sendcount,sendtype,recvbuf,
       recvcount,recvtype,comm,info)
```

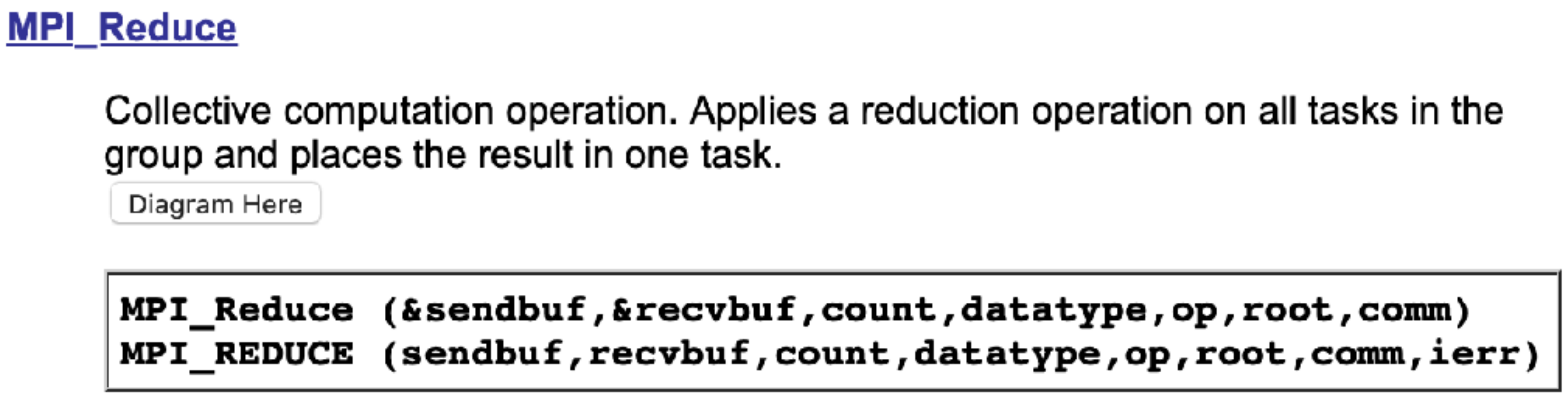


MPI_Allgather

Gathers data from all tasks and then distributes to all tasks in communicator

```
sendcnt = 1;
recvcnt = 1;
MPI_Allgather(sendbuf, sendcnt, MPI_INT
              recvbuf, recvcnt, MPI_INT
              MPI_COMM_WORLD);
```

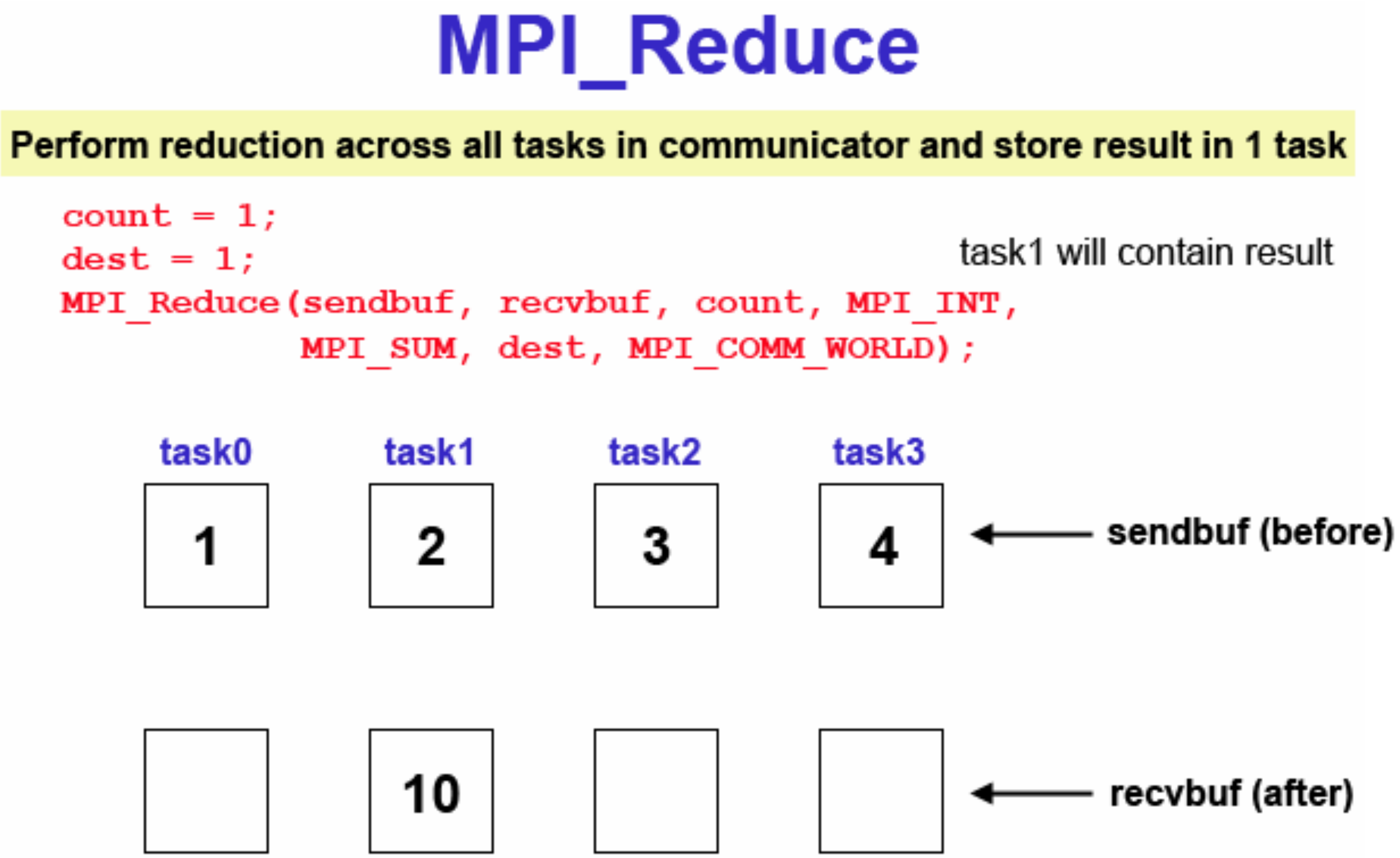
MPI Collectives



MPI_Reduce

Collective computation operation. Applies a reduction operation on all tasks in the group and places the result in one task.

```
MPI_Reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm)
MPI_REDUCE (sendbuf,recvbuf,count,datatype,op,root,comm,ierr)
```



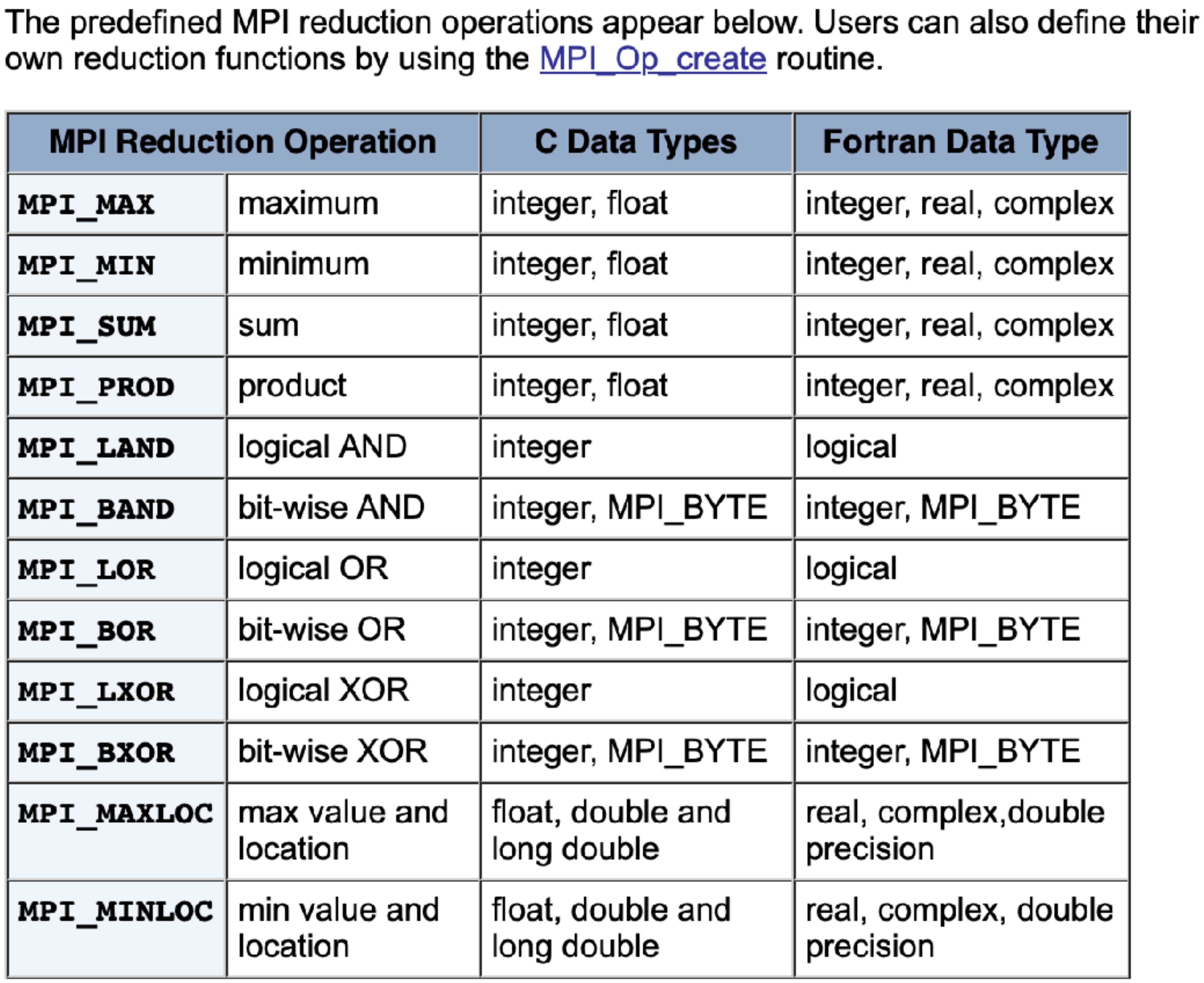
MPI_Reduce

Perform reduction across all tasks in communicator and store result in 1 task

```
count = 1;
dest = 1;
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT,
           MPI_SUM, dest, MPI_COMM_WORLD);
```

task1 will contain result

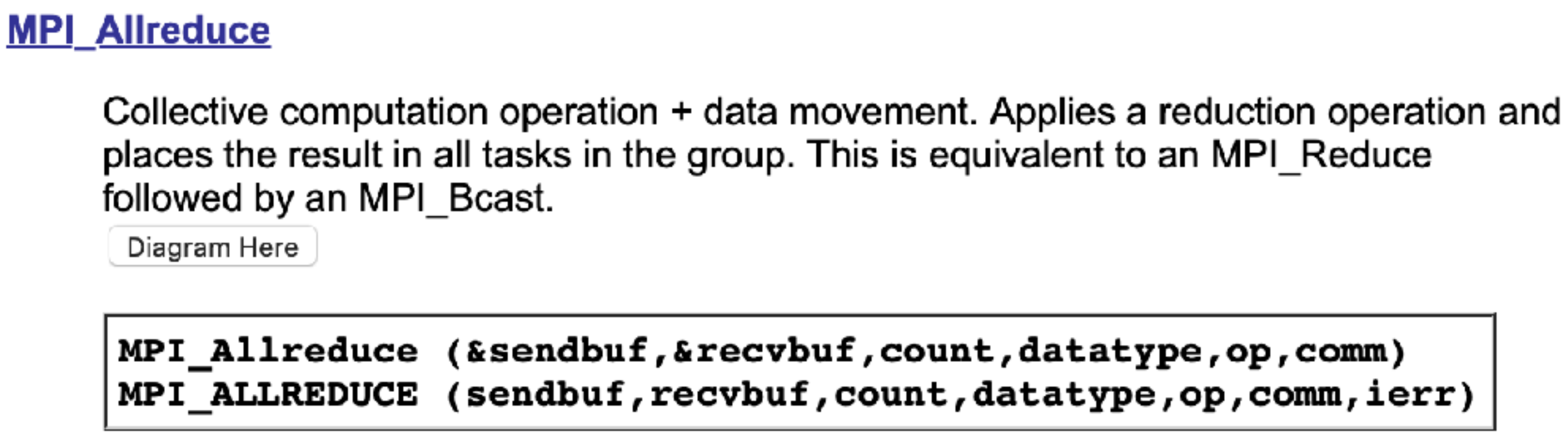
MPI Collectives



The predefined MPI reduction operations appear below. Users can also define their own reduction functions by using the MPI_Op_create routine.

MPI Reduction Operation		C Data Types	Fortran Data Type
MPI_MAX	maximum	integer, float	integer, real, complex
MPI_MIN	minimum	integer, float	integer, real, complex
MPI_SUM	sum	integer, float	integer, real, complex
MPI_PROD	product	integer, float	integer, real, complex
MPI_LAND	logical AND	integer	logical
MPI_BAND	bit-wise AND	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LOR	logical OR	integer	logical
MPI_BOR	bit-wise OR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer	logical
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double	real, complex,double precision
MPI_MINLOC	min value and location	float, double and long double	real, complex, double precision

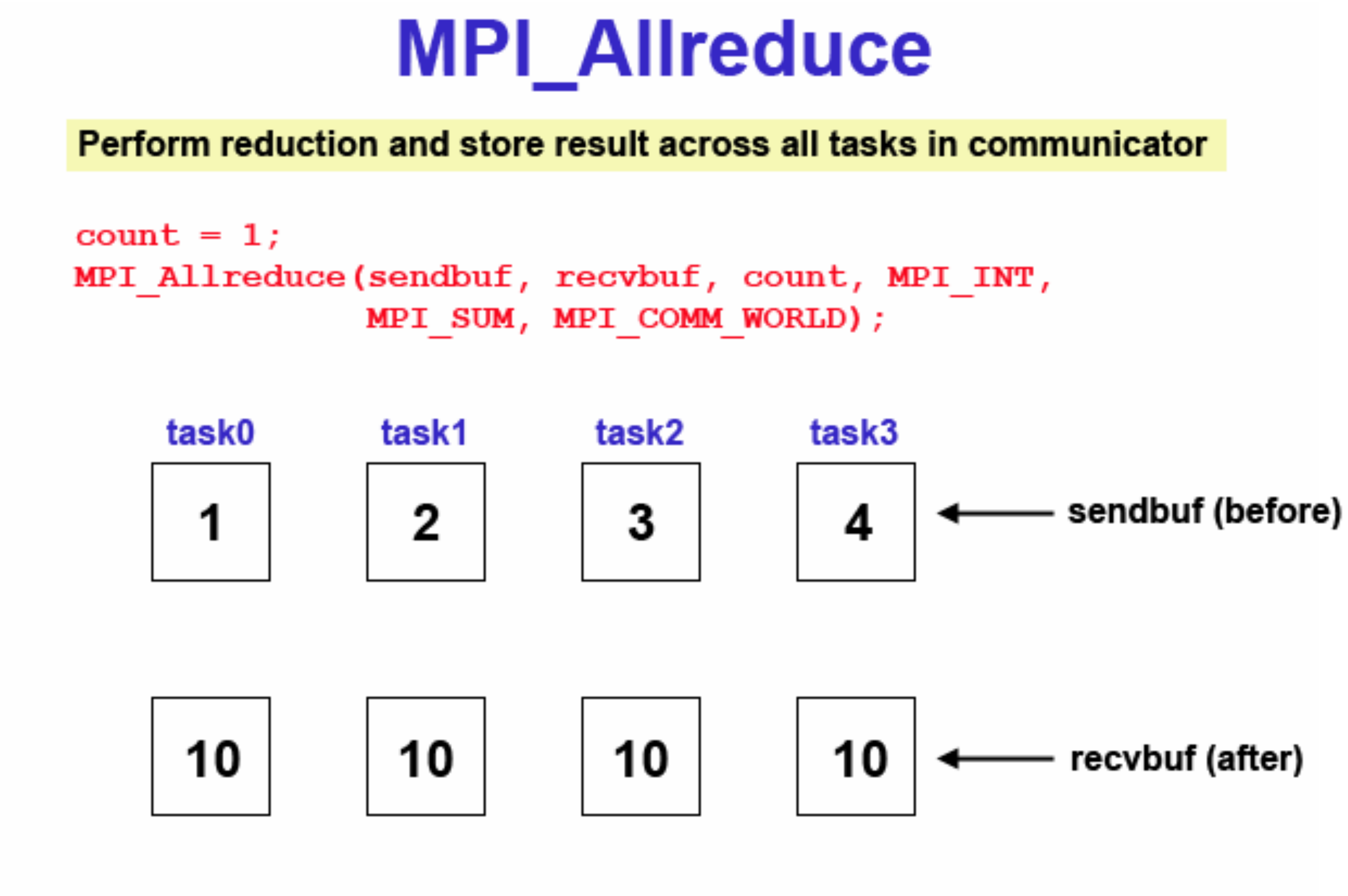
MPI Collectives



MPI_Allreduce

Collective computation operation + data movement. Applies a reduction operation and places the result in all tasks in the group. This is equivalent to an MPI_Reduce followed by an MPI_Bcast.

```
MPI_Allreduce (&sendbuf,&recvbuf,count,datatype,op,comm)
MPI_ALLREDUCE (sendbuf,recvbuf,count,datatype,op,comm,ierr)
```

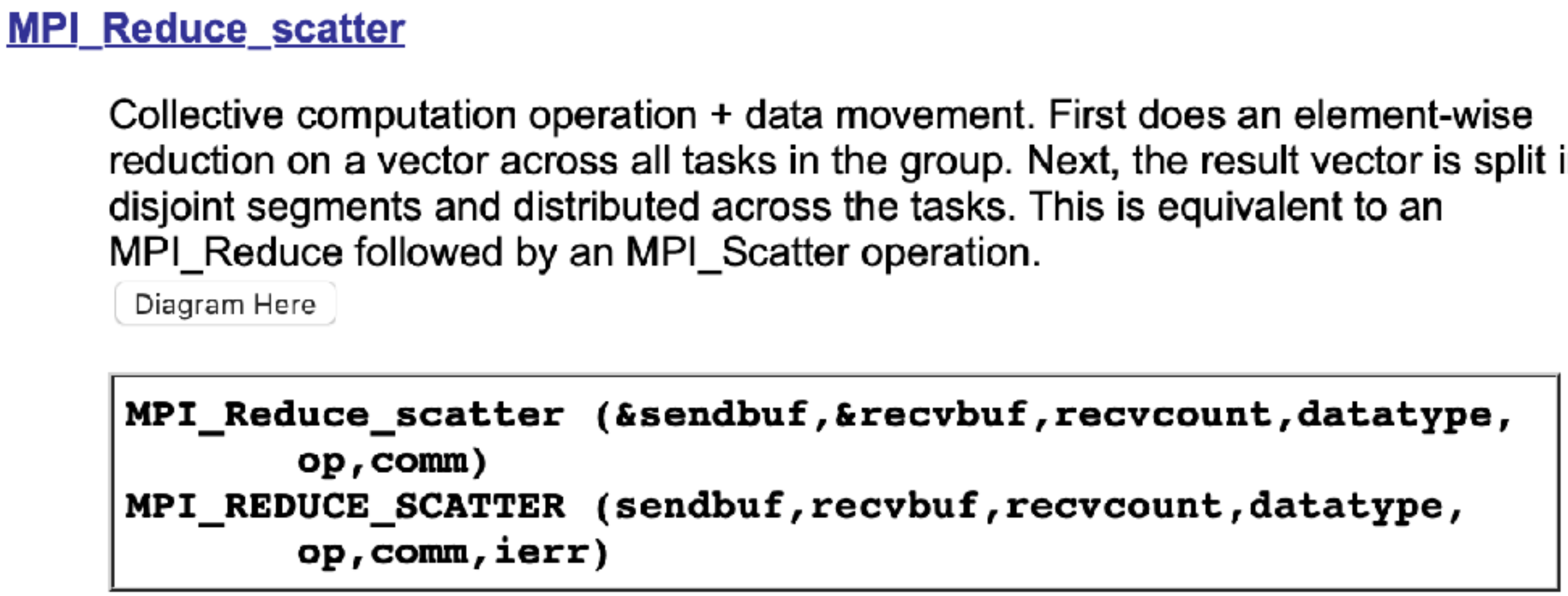


MPI_Allreduce

Perform reduction and store result across all tasks in communicator

```
count = 1;
MPI_Allreduce(sendbuf, recvbuf, count, MPI_INT,
                 MPI_SUM, MPI_COMM_WORLD);
```

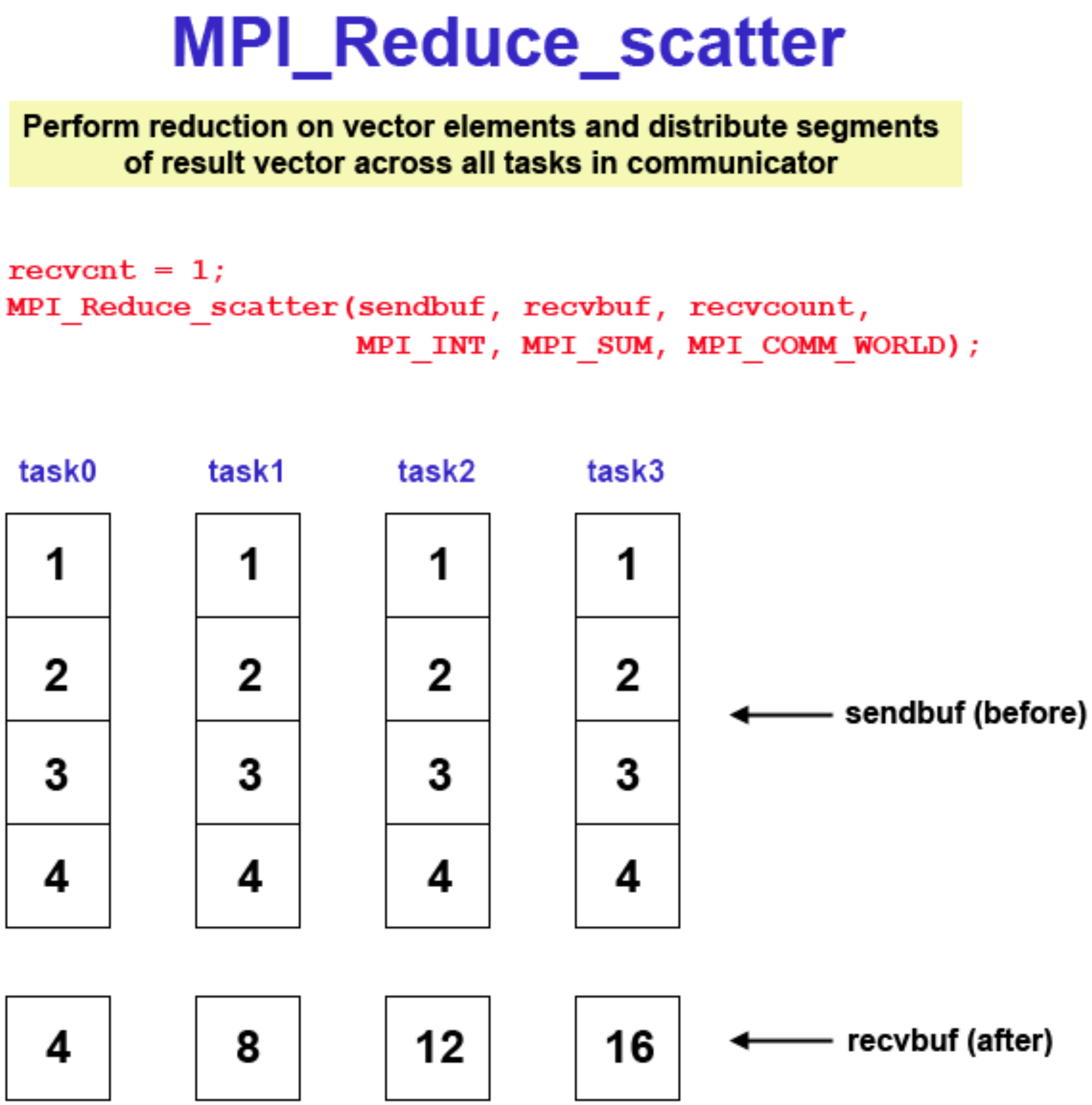
MPI Collectives



MPI_Reduce_scatter

Collective computation operation + data movement. First does an element-wise reduction on a vector across all tasks in the group. Next, the result vector is split into disjoint segments and distributed across the tasks. This is equivalent to an MPI_Reduce followed by an MPI_Scatter operation.

```
MPI_Reduce_scatter (&sendbuf,&recvbuf,recvcount,datatype,
       op,comm)
MPI_REDUCE_SCATTER (sendbuf,recvbuf,recvcount,datatype,
       op,comm,ierr)
```

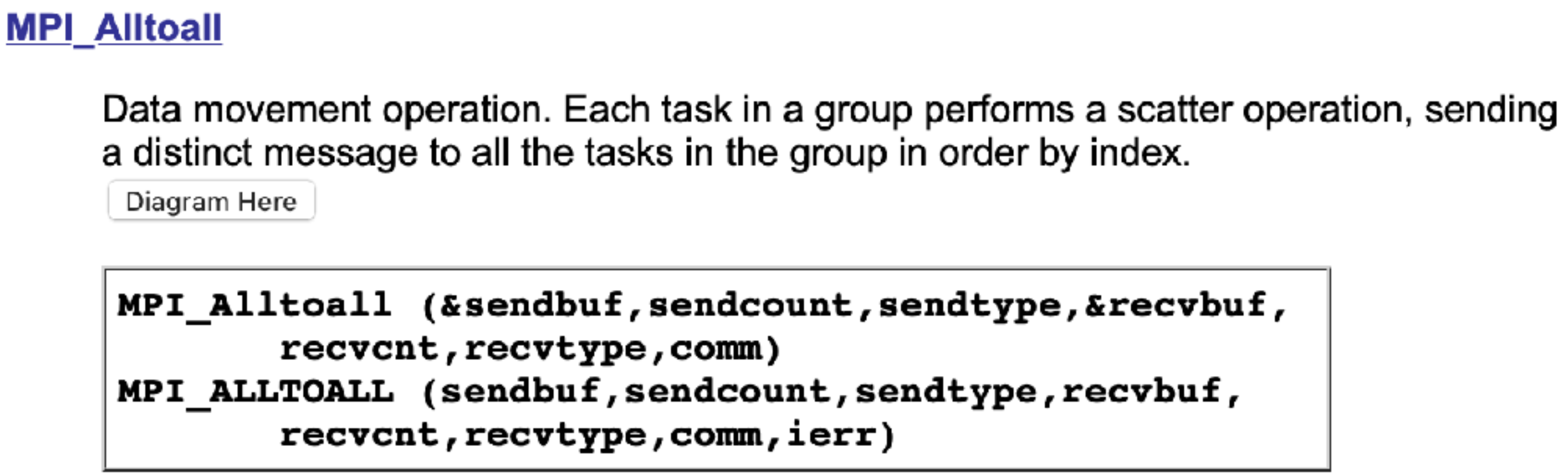


MPI_Reduce_scatter

Perform reduction on vector elements and distribute segments of result vector across all tasks in communicator

```
recvcnt = 1;
MPI_Reduce_scatter(sendbuf, recvbuf, recvcount,
                   MPI_INT, MPI_SUM, MPI_COMM_WORLD);
```

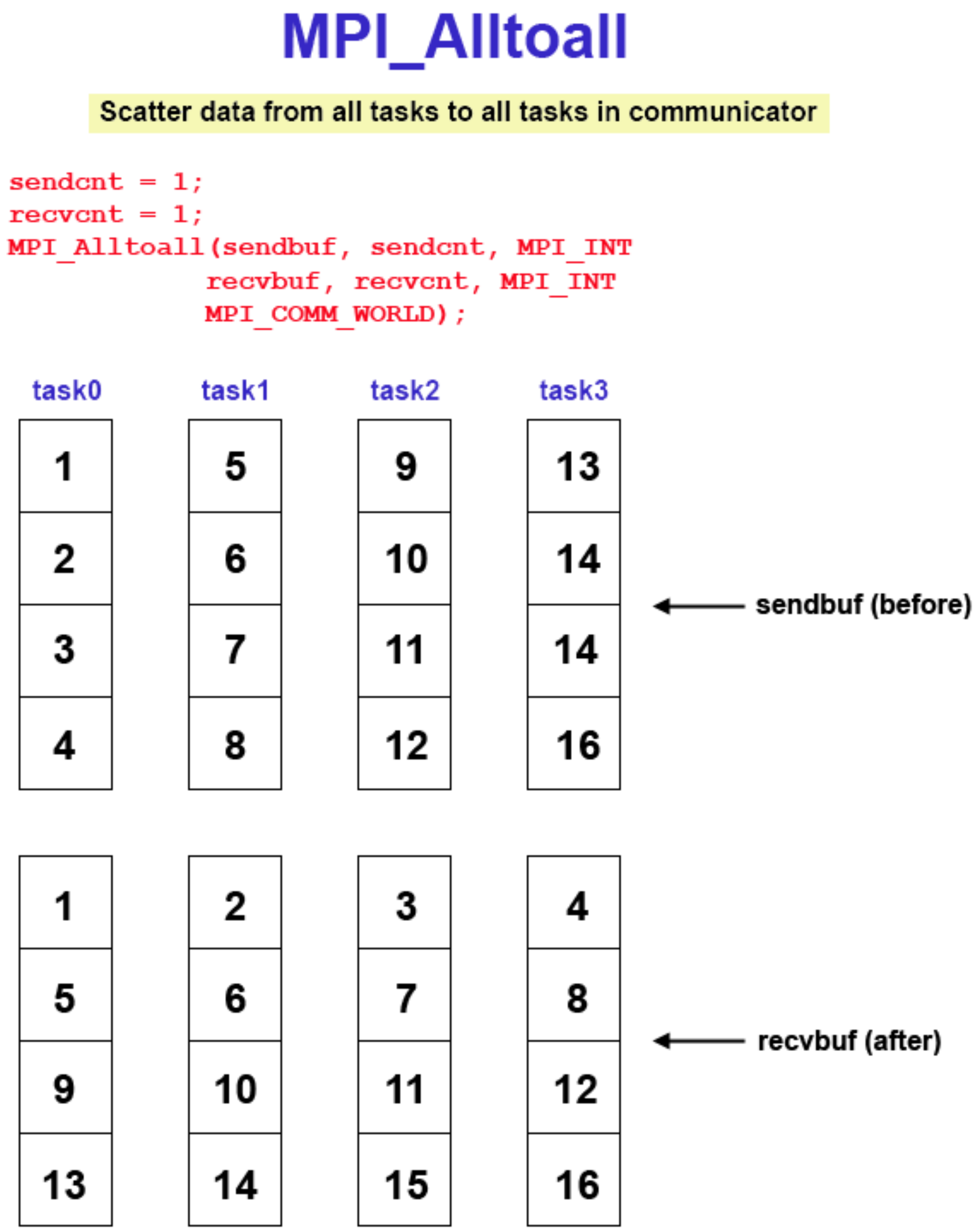
MPI Collectives



MPI_Alltoall

Data movement operation. Each task in a group performs a scatter operation, sending a distinct message to all the tasks in the group in order by index.

```
MPI_Alltoall (&sendbuf,sendcount,sendtype,&recvbuf,
      recvcnt,recvtype,comm)
MPI_ALLTOALL (sendbuf,sendcount,sendtype,recvbuf,
      recvcnt,recvtype,comm,ierr)
```



MPI_Alltoall

Scatter data from all tasks to all tasks in communicator

```
sendcnt = 1;
recvcnt = 1;
MPI_Alltoall(sendbuf, sendcnt, MPI_INT
             recvbuf, recvcnt, MPI_INT
             MPI_COMM_WORLD);
```

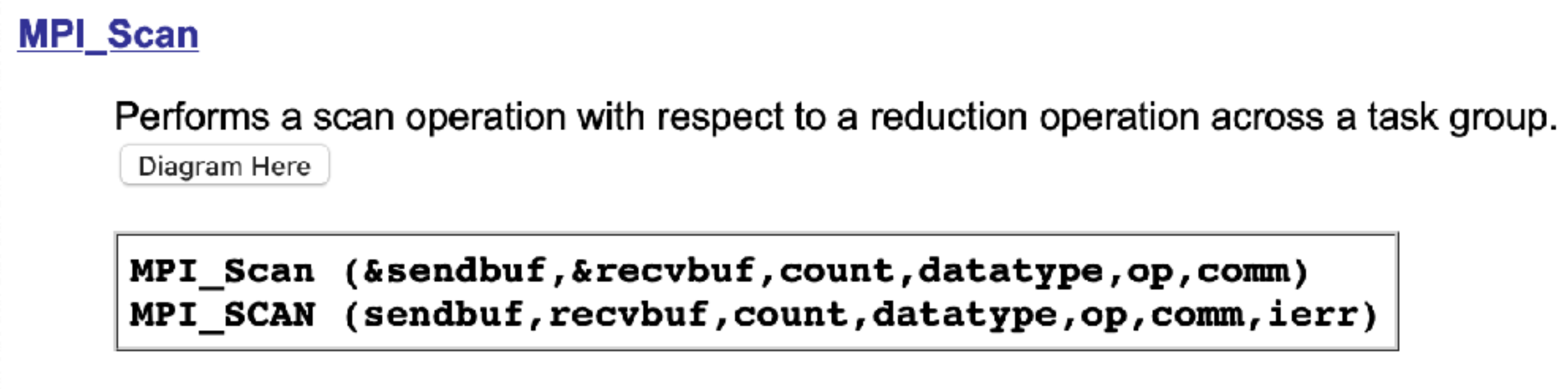
sendbuf (before)

task0	task1	task2	task3
1	5	9	13
2	6	10	14
3	7	11	14
4	8	12	16

recvbuf (after)

task0	task1	task2	task3
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

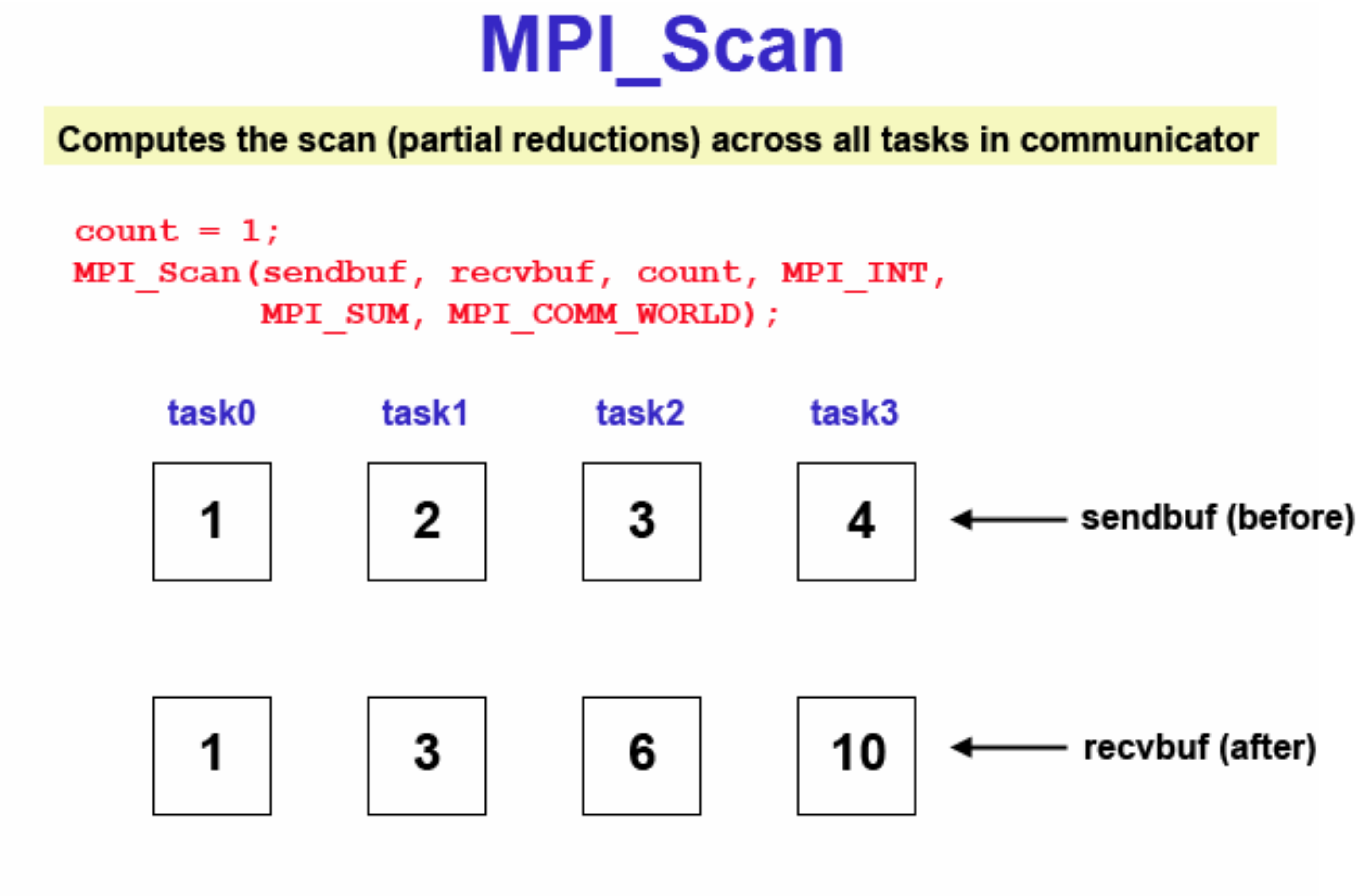
MPI Collectives



MPI_Scan

Performs a scan operation with respect to a reduction operation across a task group.

```
MPI_Scan (&sendbuf,&recvbuf,count,datatype,op,comm)
MPI_SCAN (sendbuf,recvbuf,count,datatype,op,comm,ierr)
```

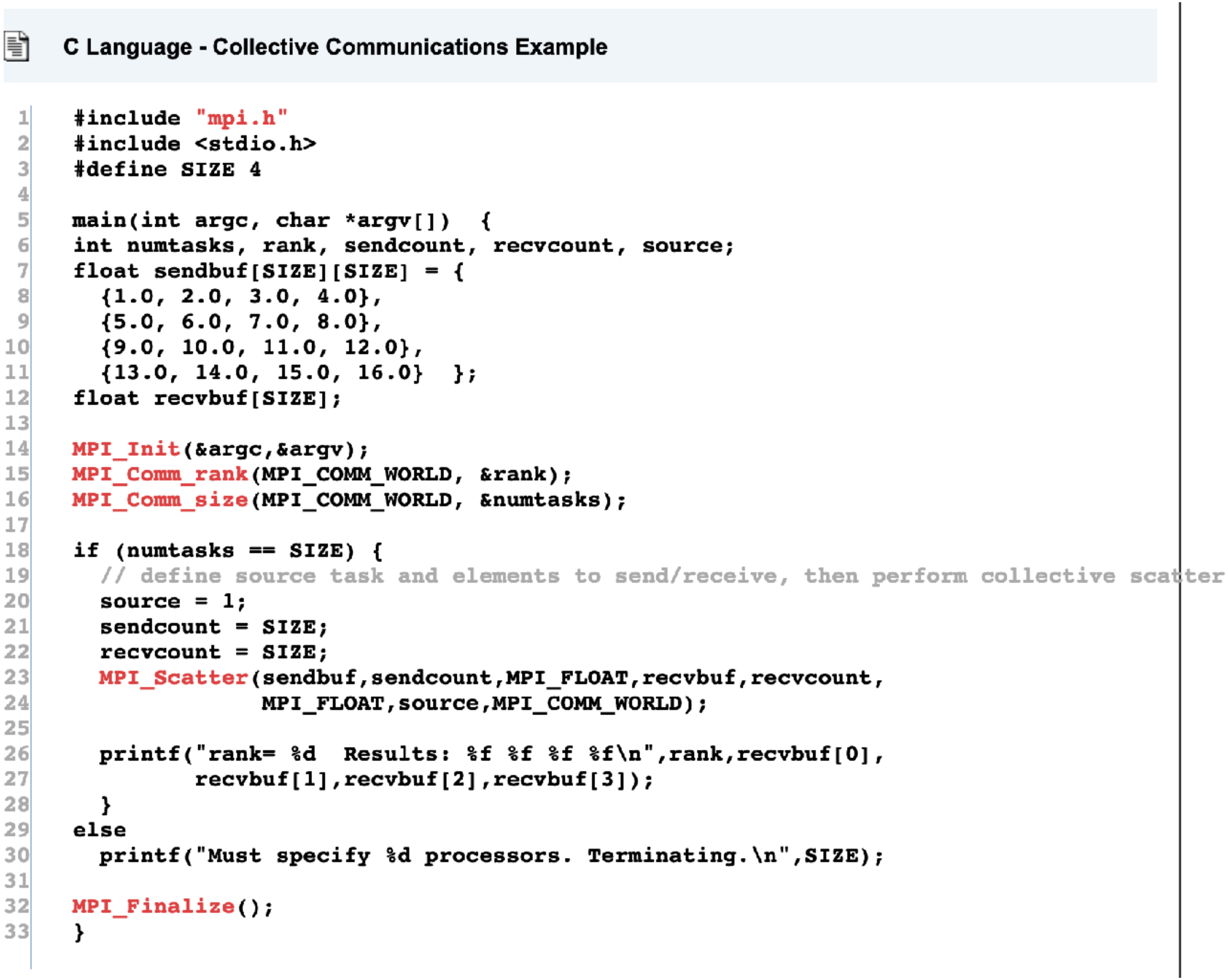


MPI_Scan

Computes the scan (partial reductions) across all tasks in communicator

```
count = 1;
MPI_Scan(sendbuf, recvbuf, count, MPI_INT,
         MPI_SUM, MPI_COMM_WORLD);
```

Example

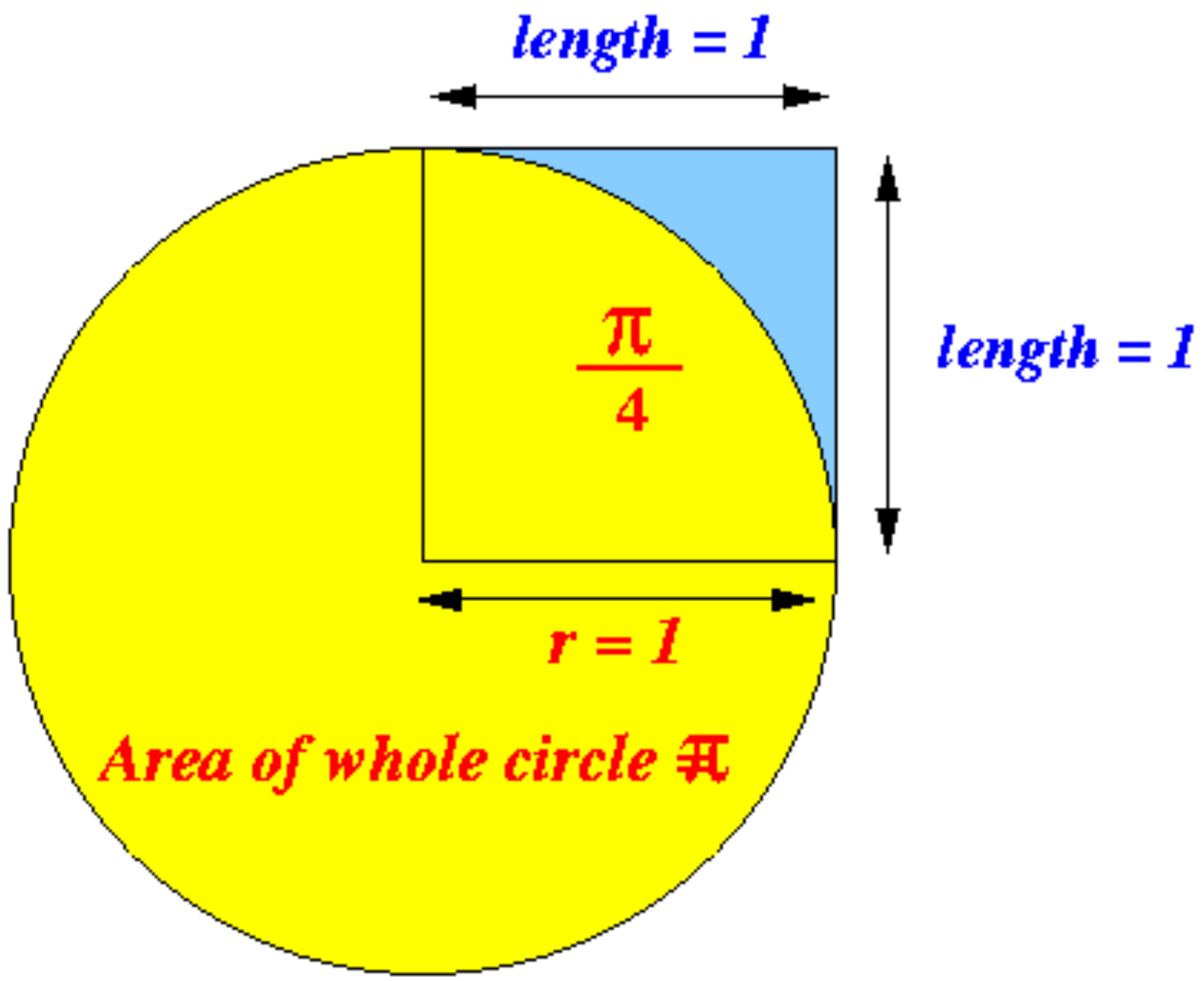


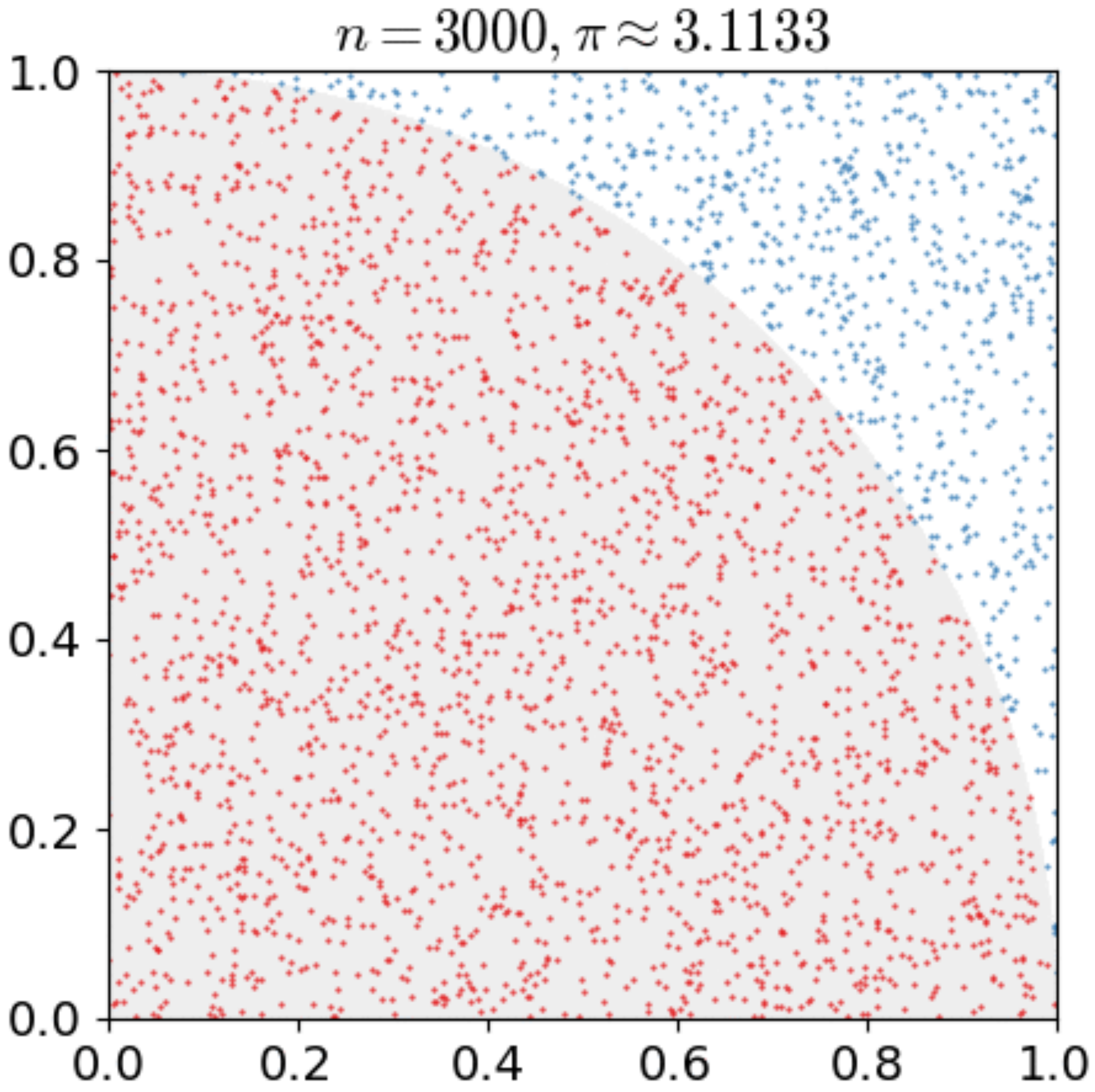
C Language - Collective Communications Example

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

main(int argc, char *argv[])  {
int numtasks, rank, sendcount, recvcount, 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];

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

if (numtasks == SIZE) {
  // define source task and elements to send/receive, then perform collective scatter
  source = 1;
  sendcount = SIZE;
  recvcount = SIZE;
  MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
              MPI_FLOAT,source,MPI_COMM_WORLD);

  printf("rank= %d  Results: %f %f %f %f\n",rank,recvbuf[0],
         recvbuf[1],recvbuf[2],recvbuf[3]);
  }
else
  printf("Must specify %d processors. Terminating.\n",SIZE);

MPI_Finalize();
}
```

Project 2

Pi by MPI


length = 1
length = 1
$\frac{\pi}{4}$
r = 1
Area of whole circle π


$n = 3000, \pi \approx 3.1133$