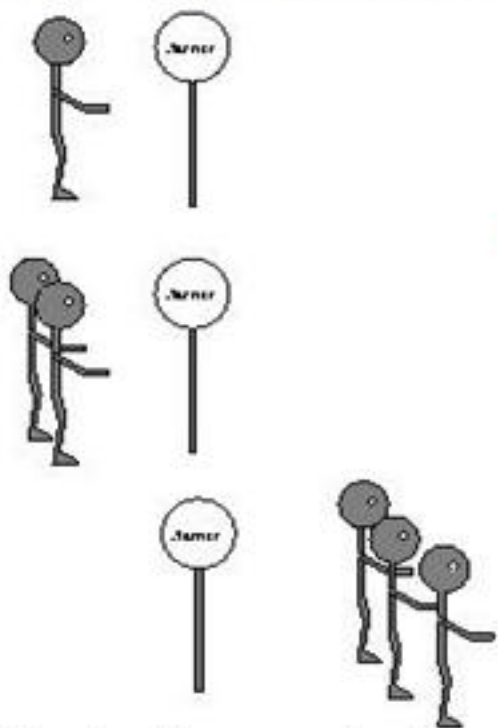


MPI Collective Communication Calls

- **Point-to-point communication operations discussed until now involve a pair of communicating processes**
- **Many message-passing systems also provide operations which allow larger numbers of processes to communicate**

- A barrier operation synchronizes a number of processes
- No data is exchanged but the barrier blocks until all participating processes have called the barrier routine

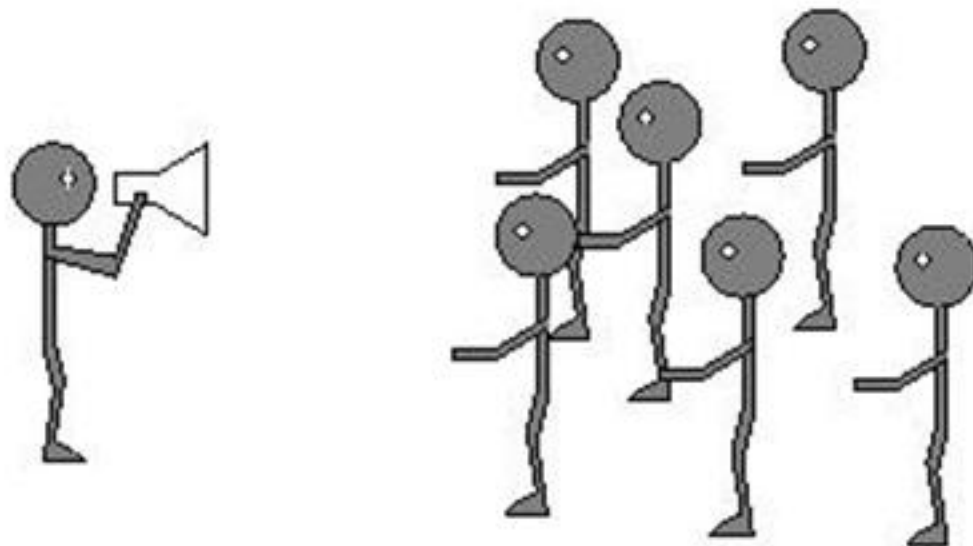


```
Call MPI_Barrier(comm, ierr)
```

```
int MPI_Barrier(MPI_Comm comm)
```

Call to MPI_Barrier returns only after all the processes in the group have called this function

- A broadcast is a one-to-many communication operation
- One process sends the same message to several destination processes with a single operation

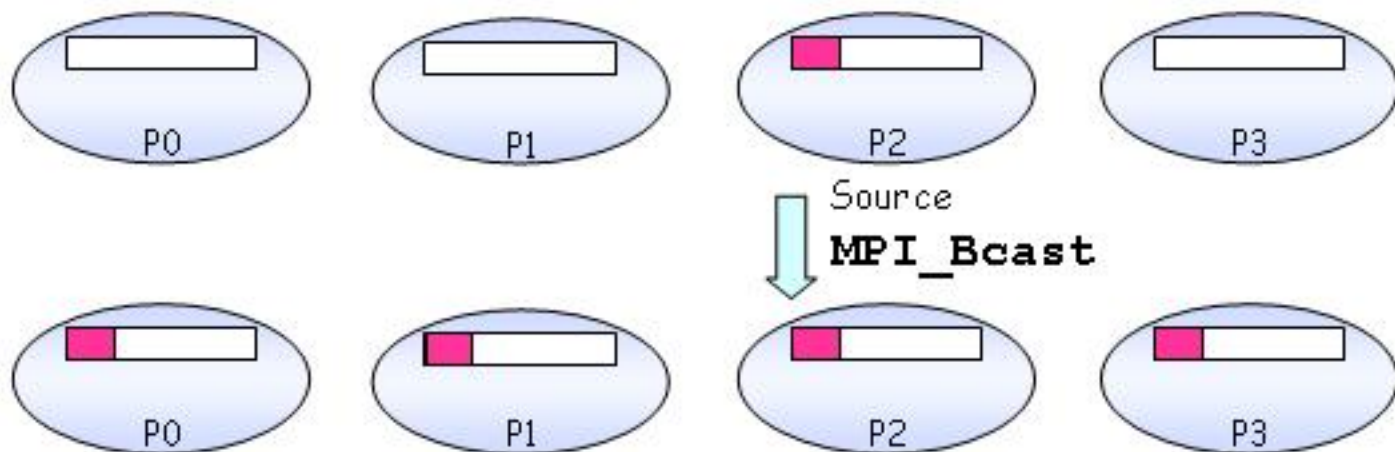



Trace from http://www.cse.cmu.edu/computing/training/document_archive/mpi-course/mpi-course.book_13.html

Call `MPI_Bcast(buf, count, datatype, source, comm, ierr)`

```
int MPI_Bcast(void *buf, int count, MPI_Datatype
datatype, int source, MPI_Comm comm)
```

- Sends data stored in buffer `buf` of process `source` to all the other processes in the group `comm`**

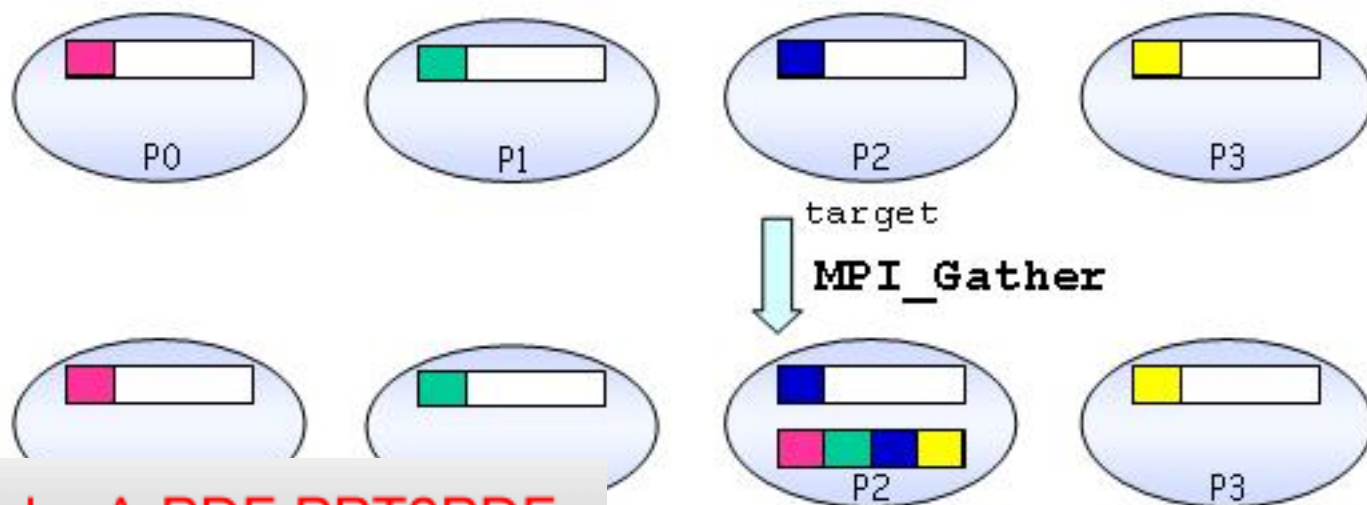


Note:  contains `count * datatype` elements

Call `MPI_Gather(sendbuf, sendcount, senddatatype, recvbuf, recvcount, recvdatatype, target, comm, ierr)`

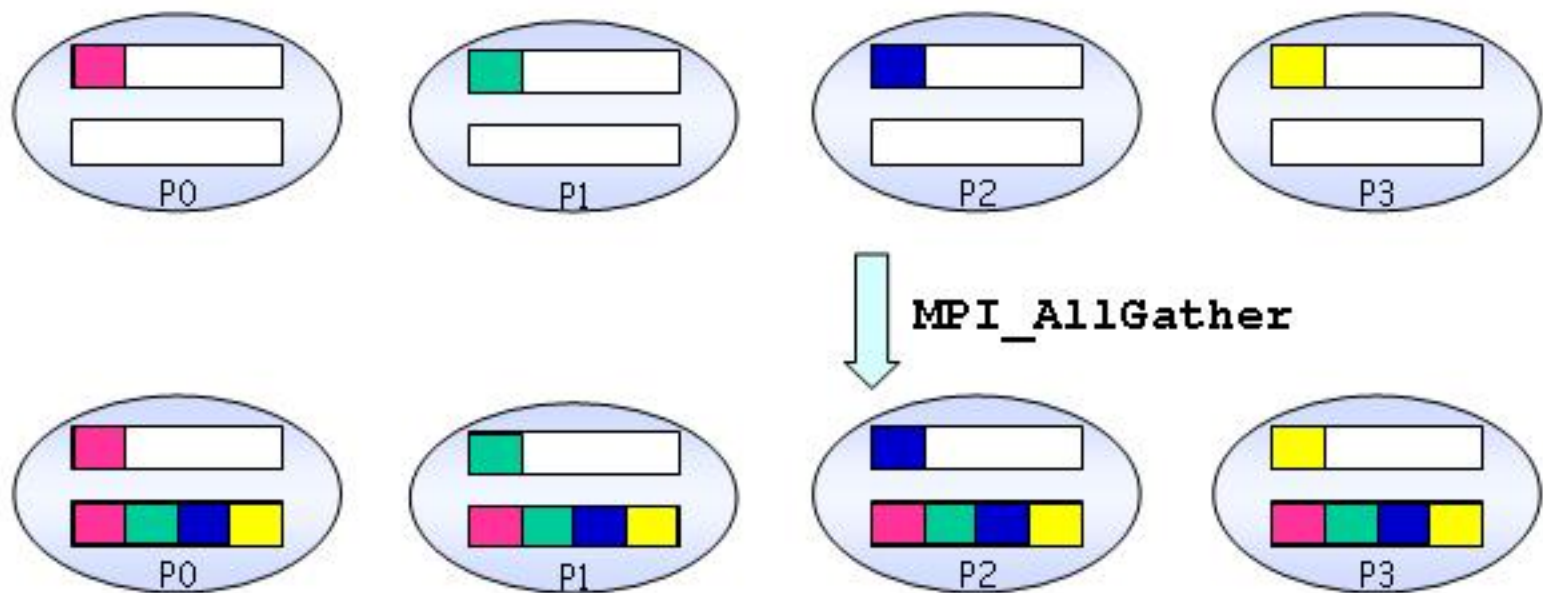
```
int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype
senddatatype, void *recvbuf, int recvcount, MPI_Datatype
recvdatatype, int target, MPI_Comm comm)
```

- **Each process (incl. target), sends data stored in array sendbuf, containing sendcount elements to the target process**
 - `recvbuf` of target process stores data in rank order
 - `recvcount` specifies no. of elements received from each process



```
Call MPI_Allgather(sendbuf, sendcount, senddatatype,
recvbuf, recvcount, recvdatatype, comm, ierr)
```

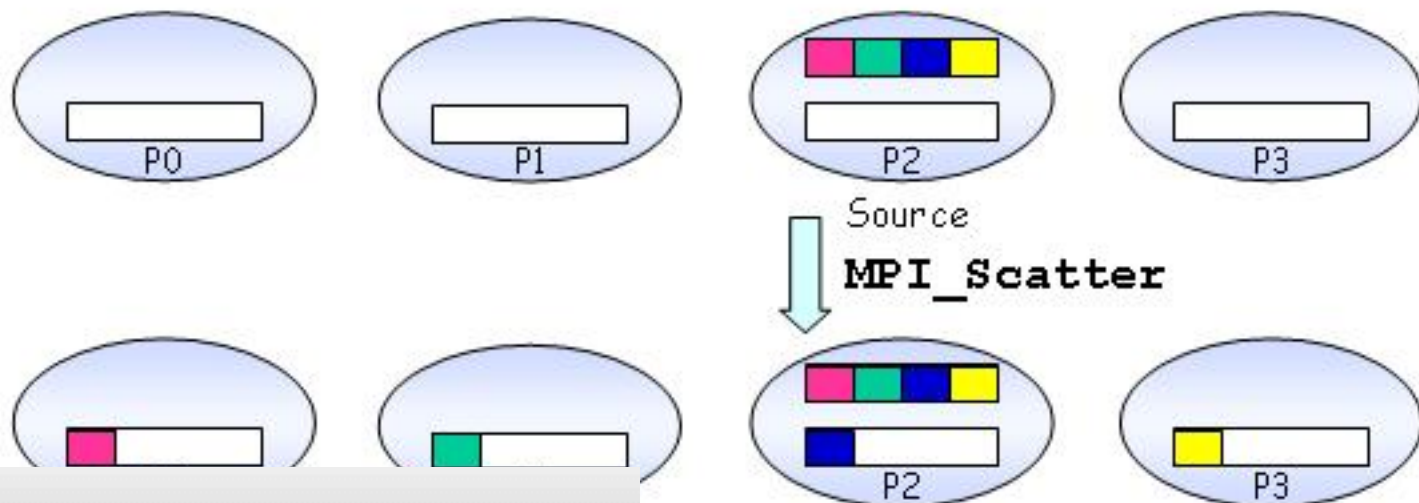
```
int MPI_Allgather(void *sendbuf, int sendcount,
MPI_Datatype senddatatype, void *recvbuf, int recvcount,
MPI_Datatype recvdatatype, MPI_Comm comm)
```

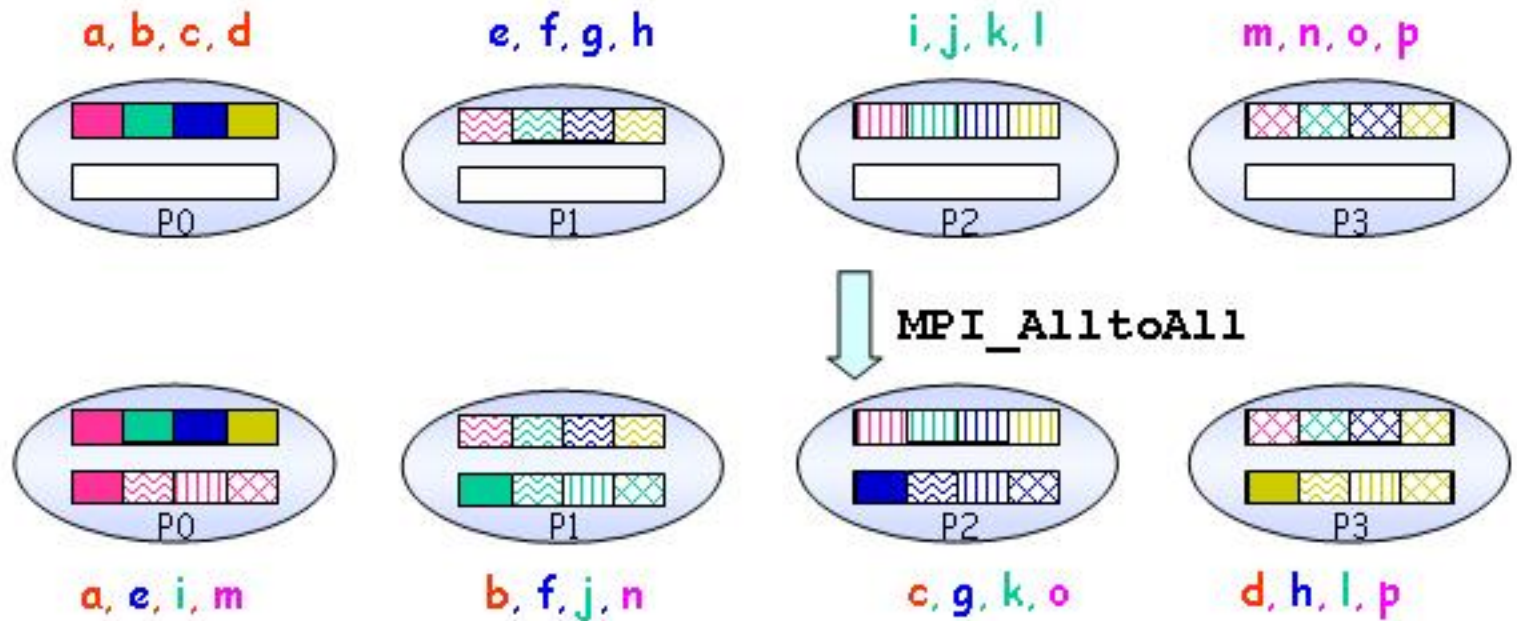


Call `MPI_Scatter(sendbuf, sendcount, senddatatype, recvbuf, recvcount, recvdatatype, source, comm, ierr)`

```
int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype
senddatatype, void *recvbuf, int recvcount, MPI_Datatype
recvdatatype, int source, MPI_Comm comm)
```

- **The source process sends different part of sendbuf to each process (incl. itself)**
 - `recvbuf` of target process stores data in rank order
 - `sendcount` specifies no. of elements sent to each process





```
Call MPI_AlltoAll(sendbuf, sendcount, senddatatype,
recvbuf, recvcount, recvdatatype, comm, ierr)
```

```
int MPI_AlltoAll(void *sendbuf, int sendcount, MPI_Datatype
senddatatype, void *recvbuf, int recvcount, MPI_Datatype
recvdatatype, MPI_Comm comm)
```

- **Each process sends a different portion of sendbuf to each other process (incl. itself)**
 - **recvbuf of target process stores data in rank order**
 - **sendcount specifies no. of elements sent to each process**

```

#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
    int rank, size, i, j, sndcnt,rcvnt=1;
    double param[4],mine;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    if(rank==3)
    {
        for(i=0;i<4;i++) param[i]=23.0+i;
        sndcnt=1;
    }
    MPI_Scatter(param,sndcnt,MPI_DOUBLE,&mine,rcvnt,MPI_DOUBLE,3,MPI_COMM_WORLD);
    printf("Process %d: Mine is %f\n",rank,mine);
    MPI_Finalize();
}

```

```
$mpiexec -np 4 scatter
```

Program Output:

```

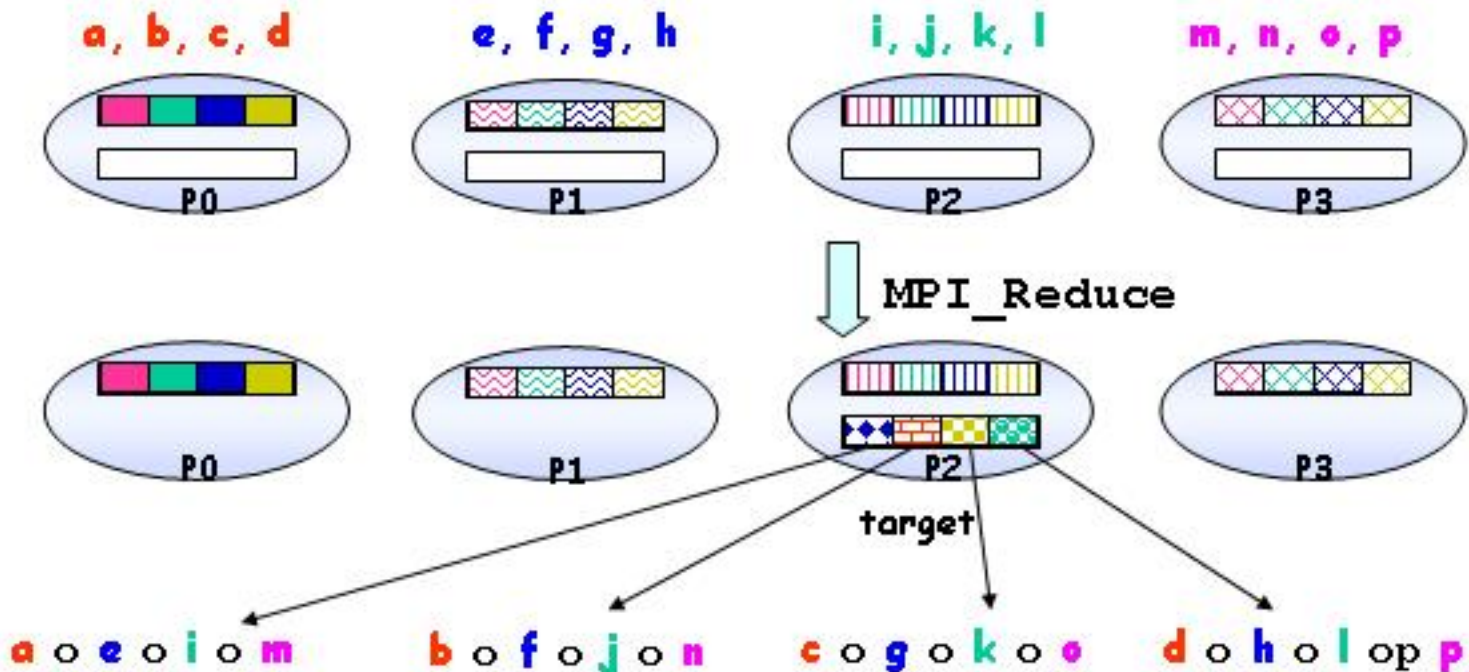
Process 0: Mine is 23.000000
Process 1: Mine is 24.000000
Process 2: Mine is 25.000000
Process 3: Mine is 26.000000

```

```

int MPI_Scatter(void *sendbuf, int sendcount,
MPI_Datatype senddatatype, void *recvbuf, int recvcount,
MPI_Datatype recvdatatype, int source, MPI_Comm comm)

```



Call `MPI_Reduce(sendbuf, recvbuf, count, datatype, op, target, comm, ierr)`

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
MPI_Datatype datatype, MPI_Op op, int target, MPI_Comm comm)
```

- **Combines elements in sendbuf of each process in group using op operation, and returns combined values in recvbuf of target process**
- **All processes must call MPI_Reduce with same value for count**

```
Call MPI_AllReduce(sendbuf, recvbuf, count, datatype,
op, comm, ierr)
```

```
int MPI_AllReduce(void *sendbuf, void *recvbuf, int
count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

- All processes receive the result of the operation, hence, there is no target argument

Operation	Meaning	Datatypes
MPI_MAX	Maximum	Integers and floating point
MPI_MIN	Minimum	Integers and floating point
MPI_SUM	Sum	Integers and floating point
MPI_PROD	Product	Integers and floating point
MPI_LAND	Logical AND	Integers
MPI_BAND	Bit-wise AND	Integers and byte
...
MPI_MAXLOC	max-min value-location	Data-pairs
MPI_MINLOC	min-min value-location	Data-pairs

Value	15	17	11	12	17	11
Process	0	1	2	3	4	5

$\text{MinLoc}(\text{Value}, \text{Process}) = (11, 2)$ *(Min value, Min processor ID)*

$\text{MaxLoc}(\text{Value}, \text{Process}) = (17, 1)$ *(Max value, Min processor ID)*

```

#include <mpi.h>
/* Run with 16 processes */
void main (int argc, char *argv[])
{
    int rank, root=7;
    struct
    {
        double value;
        int rank;
    } in, out;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    in.value=rank+1;
    in.rank=rank;
    MPI_Reduce(&in, &out, 1, MPI_DOUBLE_INT, MPI_MAXLOC, root, MPI_COMM_WORLD);
    if(rank==root) printf("PE:%d max=%lf at rank %d\n",
        rank, out.value, out.rank);
    MPI_Reduce(&in, &out, 1, MPI_DOUBLE_INT, MPI_MINLOC, root, MPI_COMM_WORLD);
    if(rank==root) printf("PE:%d min=%lf at rank %d\n",
        rank, out.value, out.rank);
    MPI_Finalize();
}

```

Program Output:

```

PE:7 max=16.000000 at rank 15
PE:7 min=1.000000 at rank 0

```

- **MPI also provides a mechanism for user-defined operations used in `MPI_ALLREDUCE` and `MPI_REDUCE`**
 - **User can define his/her own reduce operation using the `MPI_OP_CREATE` function**

- **Syntax:**

Call `MPI_Op_create` (function, commute, op, ierr)

```
int MPI_Op_create (MPI_User_function *function, int commute, MPI_Op *op)
```

If commute is TRUE, reduction may be faster

```

#include <mpi.h>
typedef struct
{
    double real,imag;
} complex;

void cprod(complex *in, complex *inout, int *len, MPI_Datatype *dptr)
{
    int i;
    complex c;
    for (i=0; i<*len; ++i)
    {
        c.real=(*in).real * (*inout).real - (*in).imag * (*inout).imag;
        c.imag=(*in).real * (*inout).imag + (*in).imag * (*inout).real;
        *inout=c;
        in++;
        inout++;
    }
}

```

```

void main (int argc, char *argv[])
{
    int rank, root;
    complex source,result;

    MPI_Op myop;
    MPI_Datatype ctype;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    MPI_Type_contiguous(2,MPI_DOUBLE,&ctype);
    MPI_Type_commit(&ctype);
    MPI_Op_create(cprod,TRUE,&myop);
    root=2;
    source.real=rank+1;
    source.imag=rank+2;
    MPI_Reduce(&source, &result, 1, ctype, myop, root, MPI_COMM_WORLD);
    if(rank==root) printf ("PE:%d result is %lf + %lfi\n",rank,
        result.real, result.imag);
    MPI_Finalize();
}

```

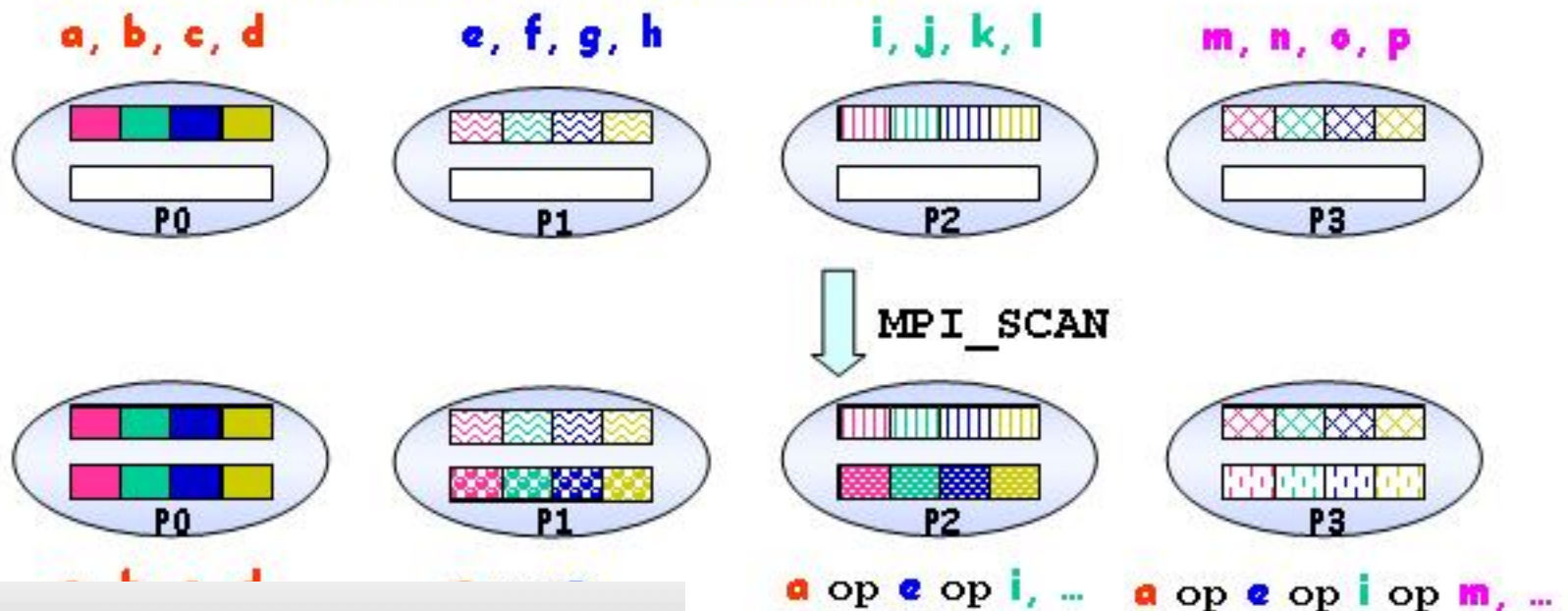
Program Output:

PE: 2 result is -185.000000 + -180.000000i

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

```
int MPI_Scan(void *sendbuf, void *recvbuf, int count,
MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

- Performs prefix reduction of data in `sendbuf` buffer at each process and returns the result in buffer `recvbuf`



- `MPI_Gatherv(...)`, `MPI_Allgatherv(...)`
 - **Allow different number of data elements to be sent by each process**
 - Parameter `recvcount` replaced with array `recvcounts`
 - Additionally, another array parameter determines where in `recvbuf` the data sent by each process will be stored

- `MPI_Scatterv(...)`
 - **Allows different amounts of data to be sent to different processes**
 - Parameter `sendcount` replaced with array `sendcounts`
 - Additionally, another array parameter determines where in `sendbuf` these elements will be sent from

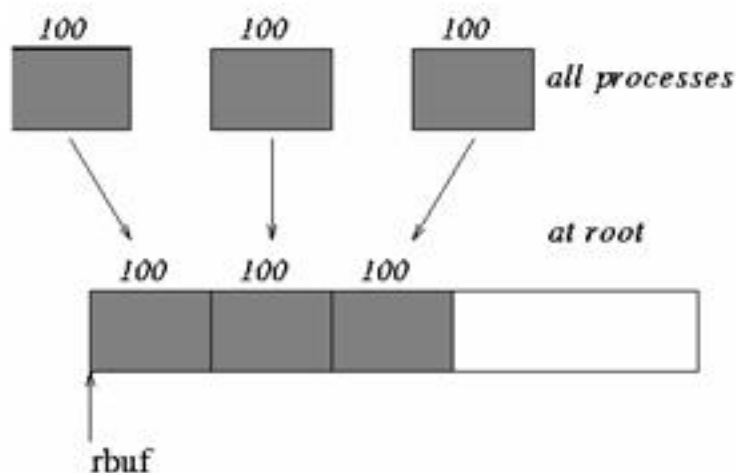
- `MPI_AlltoAllv(...)`
 - **Allows different amounts of data to be sent to and received from each process**

```
MPI_Gatherv( void *sendbuf, int sendcnt, MPI_Datatype
             sendtype, void *recvbuf, int *recvcnts, int *displs,
             MPI_Datatype recvtype, int root, MPI_Comm comm)
```

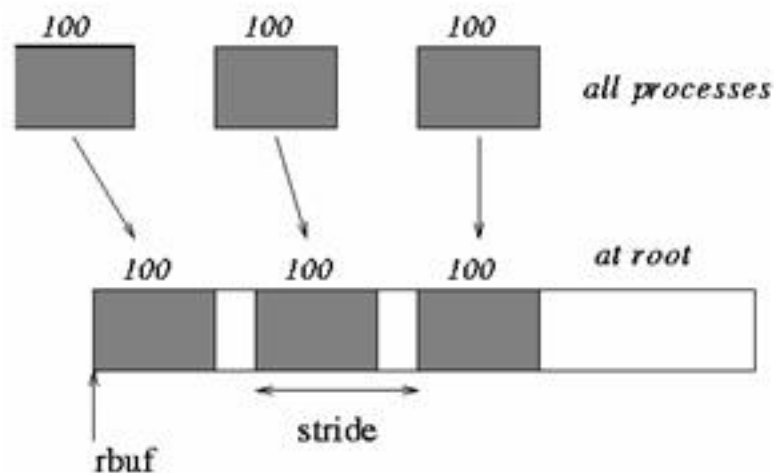
- **Input Parameters**

- **sendbuf** starting address of send buffer (choice)
- **sendcount** number of elements in send buffer (integer)
- **sendtype** data type of send buffer elements (handle)
- **recvcnts** integer array (of length group size) containing no. of elements received from each process (significant only at root)
- **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* (significant only at root)
- **recvtype** data type of recv buffer elements (significant only at root) (handle)
- **root** rank of receiving process (integer)
- **comm** communicator (handle)

```
MPI_Gather(sendarray, 100, MPI_INT,
          rbuf, 100, MPI_INT, root, comm);
```



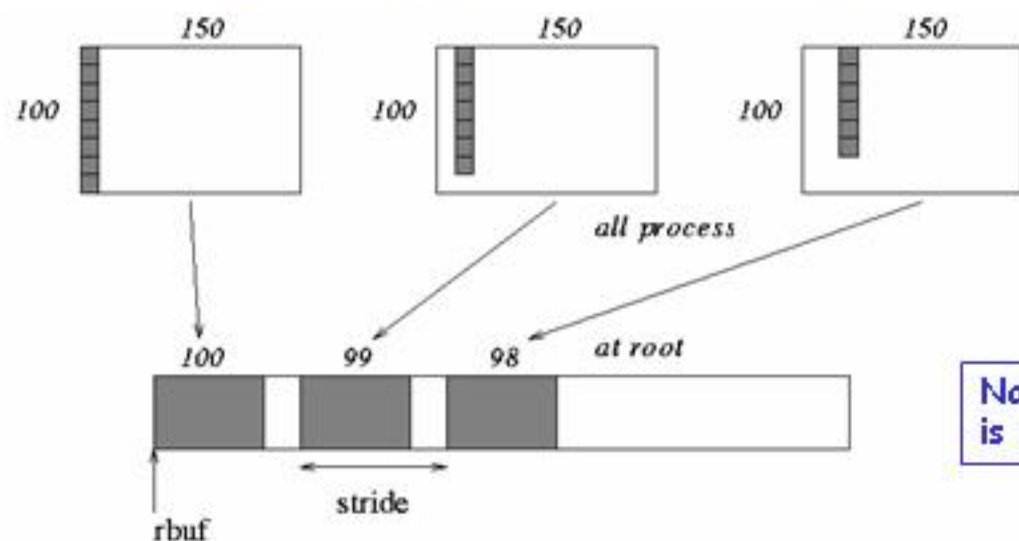
```
for (i=0; i<gsize; ++i) { displs[i] = i*stride;
                          rcounts[i] = 100;}
MPI_Gatherv(sendarray, 100, MPI_INT,
            rbuf, rcounts, displs, MPI_INT, root,
            comm);
```



```

for (i=0; i<gsize; ++i) {displs[i] = i*stride;
                        rcounts[i] = 100-i; // note change from previous example }
// Create datatype for the column we are sending
MPI_Type_vector( 100-myrank, 1, 150, MPI_INT, &stype);
MPI_Type_commit( &stype );
sptr = &sendarray[0][myrank];           // sptr is address of start of "myrank" column
MPI_Gatherv(sptr, 1, stype, rbuf, rcounts, displs, MPI_INT, root, comm);

```



Note: Different amount of data is received from each process

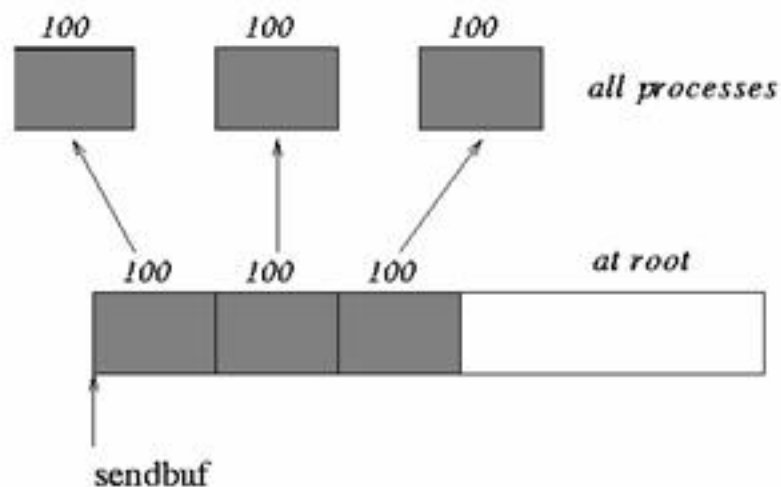
```
MPI_Scatterv(void *sendbuf, int *sendcnts, int *displs,
             MPI_Datatype sendtype, void *recvbuf, int recvcnt,
             MPI_Datatype recvtype, int root, MPI_Comm comm)
```

- **Input Parameters**

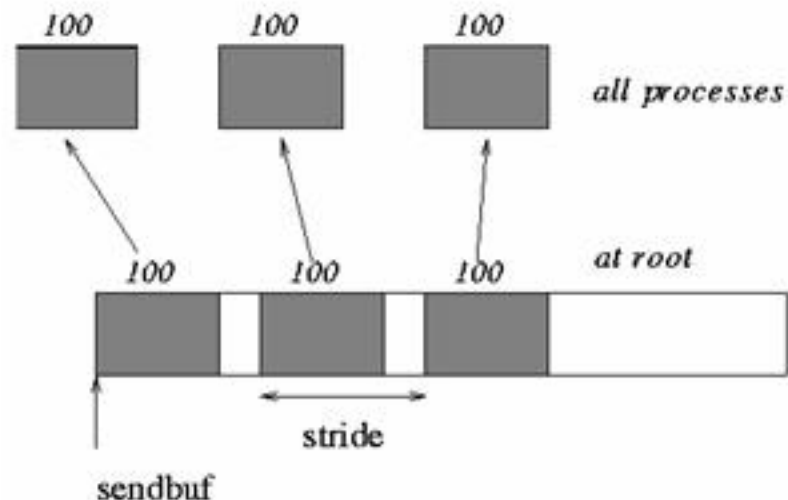
- **sendbuf** address of send buffer (choice, significant only at root)
- **sendcounts** integer array (of length group size) specifying the number of elements to send to each processor
- **displs** integer array (of length group size)
 - Entry *i* specifies the displacement relative to **sendbuf** from which to take the outgoing data to process *i*
- **sendtype** data type of send buffer elements (handle)
- **recvcount** number of elements in receive buffer (integer)
- **recvtype** data type of receive buffer elements (handle)
- **root** rank of sending process (integer)
- **comm** communicator (handle)

Scatter and Scatterv example

```
MPI_Scatter(sendbuf, 100, MPI_INT,
           rbuf, 100, MPI_INT, root, comm);
```



```
for (i=0; i<gsize; ++i) {displs[i] = i*stride;
                          counts[i] = 100;}
MPI_Scatterv(sendbuf, counts, displs,
            MPI_INT, rbuf, 100, MPI_INT, root,
            comm);
```

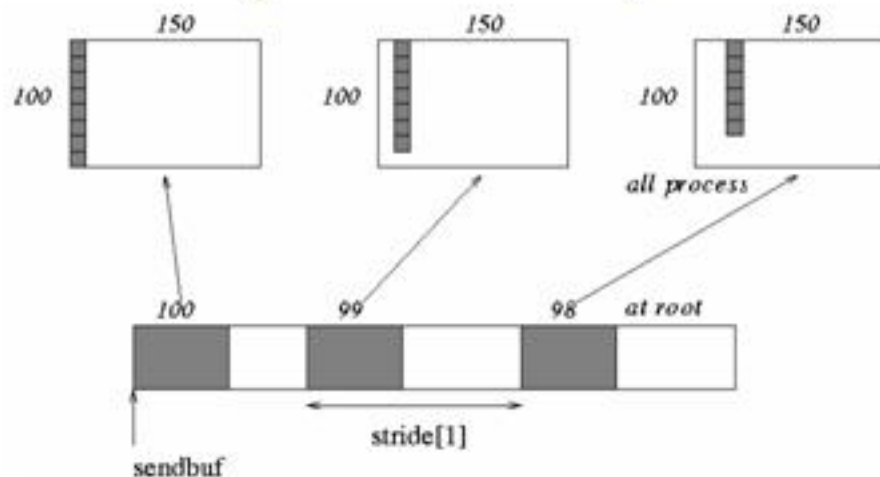


```


for (i=0; i<gsize; ++i) { displs[i] = offset;           // offset is set to zero, initially
                          offset += stride[i];
                          scounts[i] = 100 - i;}

// Create datatype for the column we are receiving
MPI_Type_vector(100-myrank, 1, 150, MPI_INT, &rtype);
MPI_Type_commit(&rtype);
rptr = &recvarray[0][myrank];           // rptr is address of start of "myrank" column
MPI_Scatterv(sendbuf, scounts, displs, MPI_INT, rptr, 1, rtype, root, comm);

```



Note: Different amount of data is scattered to each process

P0	P1	P2*	P3	Function	P0	P1	P2	P3
a	b	c	d	MPI_Gather			a, b, c, d	
a	b	c	d	MPI_Allgather	a, b, c, d	a, b, c, d	a, b, c, d	a, b, c, d
		a, b, c, d		MPI_Scatter	a	b	c	d
a, b, c, d	e, f, g, h	i, j, k, l	m, n, o, p	MPI_AlltoAll	a, e, i, m	b, f, j, n	c, g, k, o	d, h, l, p
		b		MPI_Bcast	b	b	b	b
SBuf	SBuf	SBuf	SBuf		RBuf	RBuf	RBuf	RBuf

* Sender/Root process required by MPI_Gather, MPI_Scatter, MPI_Bcast