Parallel Programming using MPI
Collective Communications

Andreas Jocksch, Claudio Gheller
CSCS

jocksch@cscs.ch, cgheller@cscs.ch

Collective Communications

Communications involving a group of processes

Called by all processes in a communicator:

- Barrier Synchronization
- Broadcast
- Gather/Scatter
- Reduction (sum, max, prod, …)

All processes must call the collective routine

No non-blocking collective communication (MPI-2)

No tags

The MPI library should use the most efficient communication algorithm for the particular platform

MPI_Barrier

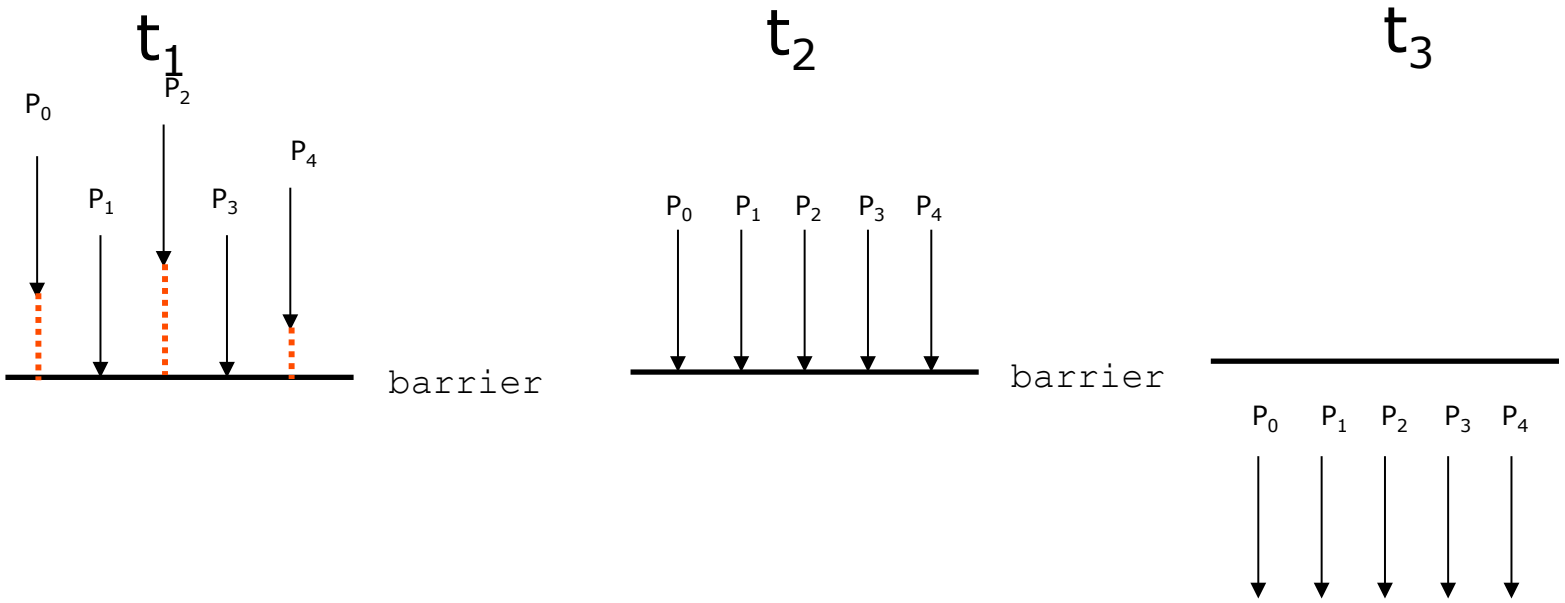
Stop processes until all processes within a communicator reach the barrier

Fortran:

```
CALL MPI_BARRIER(comm, ierr)
```

C:

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



Broadcast (MPI_BCAST)

One-to-all communication: same data sent from root process to all others in the communicator

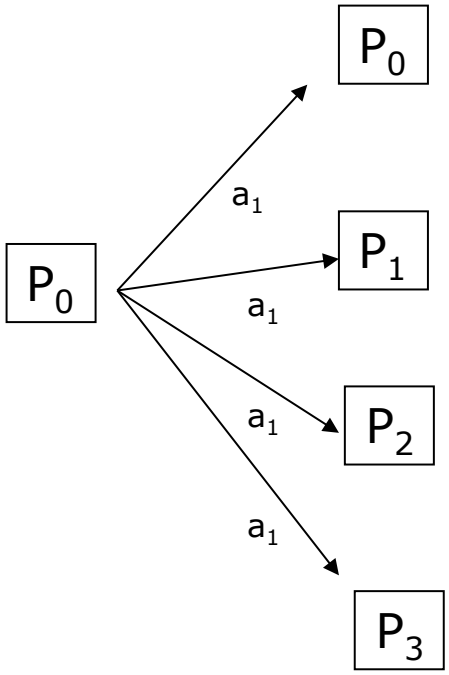
Fortran:

```
INTEGER count, type, root, comm, ierr
CALL MPI_BCAST(buf, count, type, root,
   comm, ierr)
```

Buf array of type **type**

C:

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

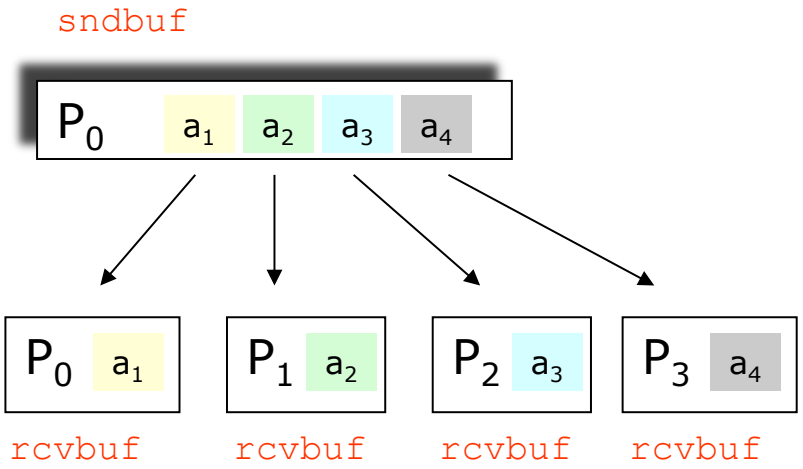


Exercise broadcast_mpi

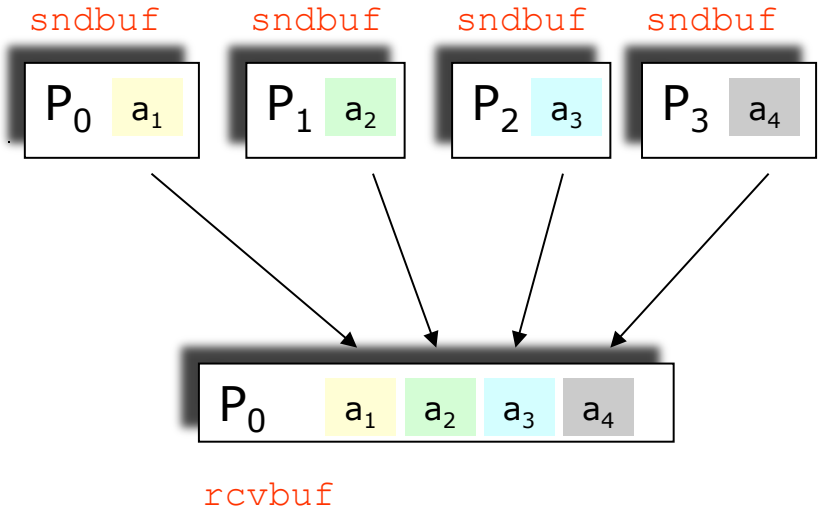
Read data from terminal and broadcast it

Scatter / Gather

Scatter



Gather



MPI_Scatter

One-to-all communication: different data sent from root process to all others in the communicator

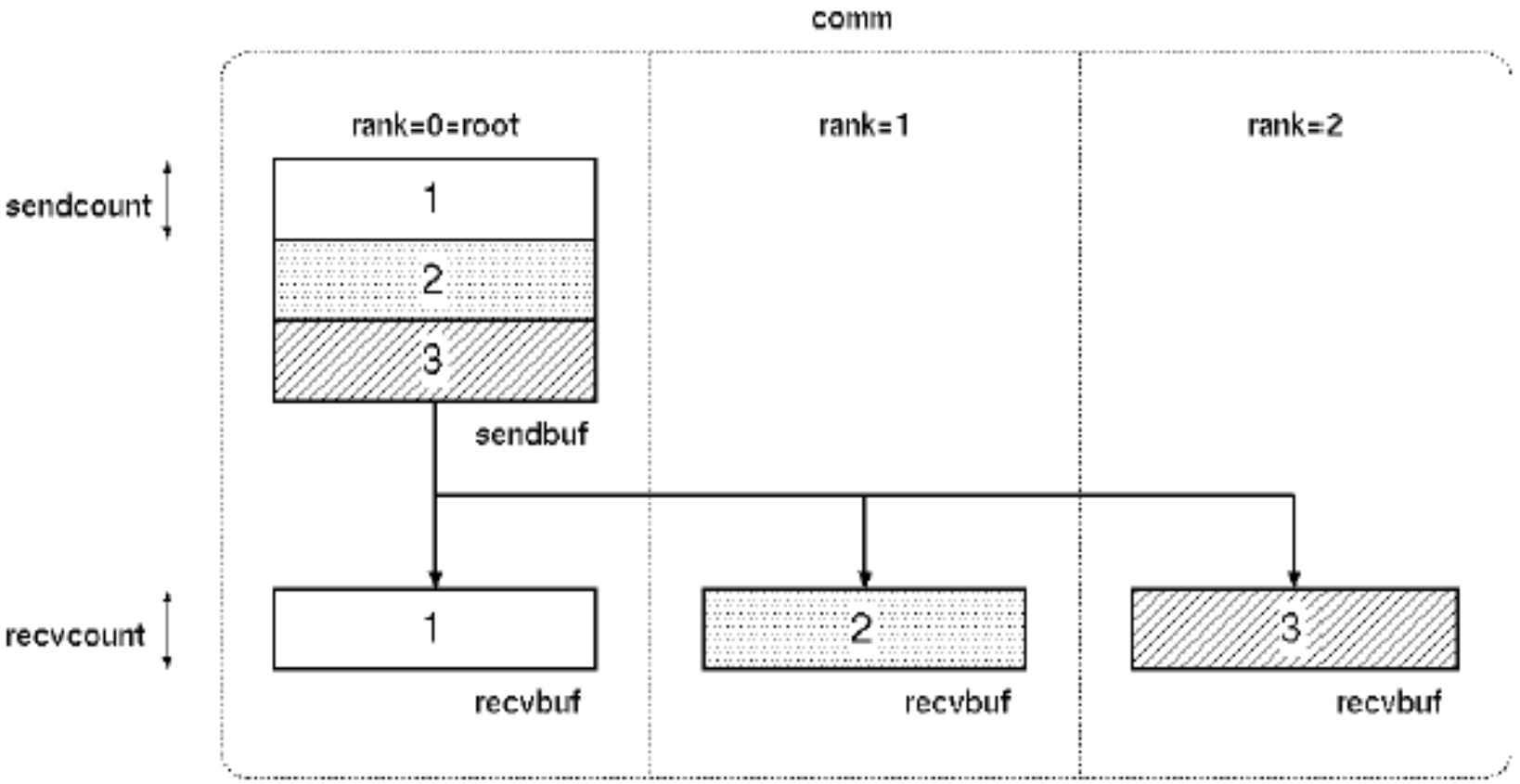
Fortran:

```
CALL MPI_SCATTER(sndbuf, sndcount, sndtype, rcvbuf,
  rcvcount, rcvtype, root, comm, ierr)
```

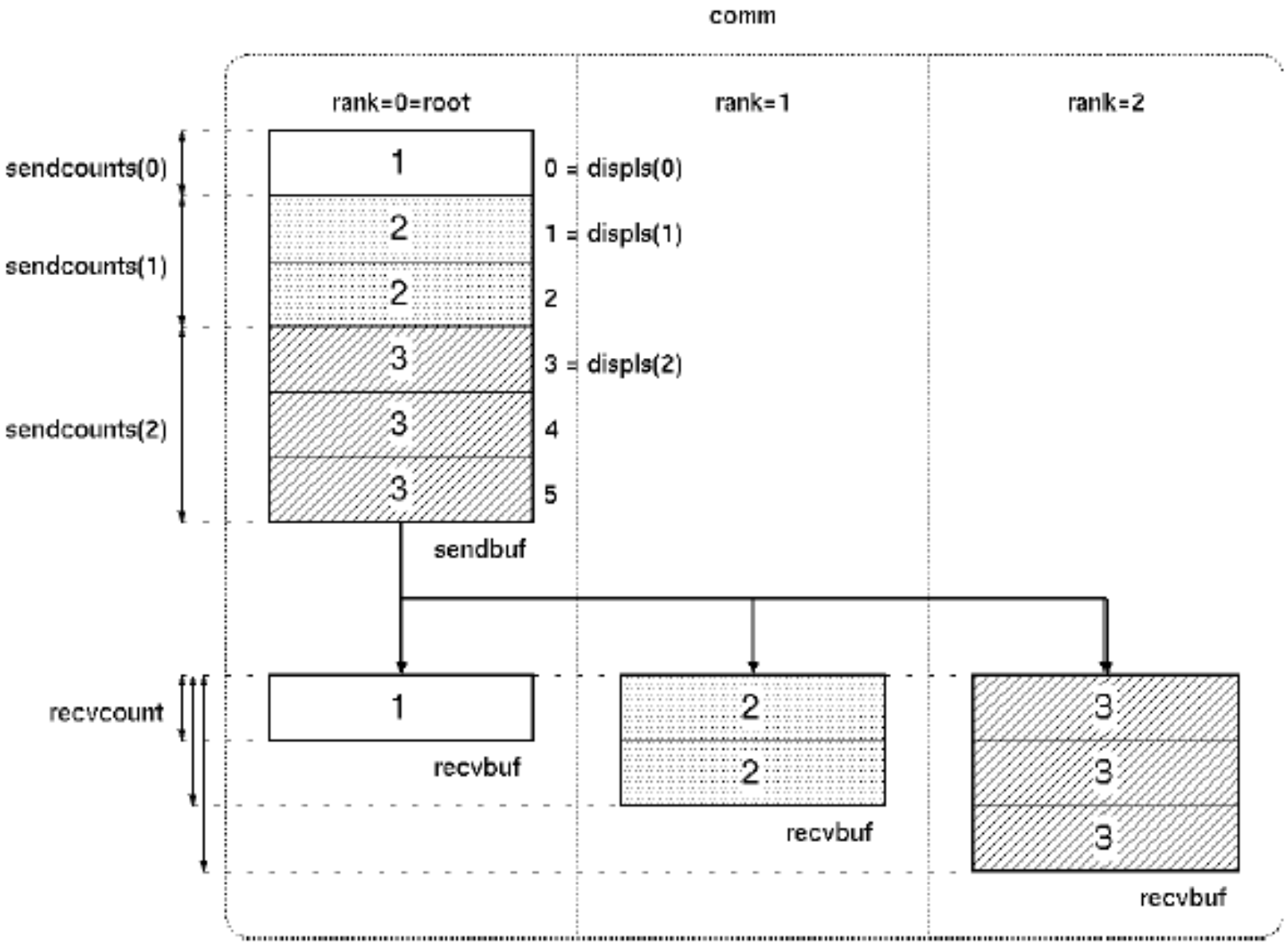
sndcount is the number of elements sent to each process, not the size of sndbuf, that should be sndcount times the number of process in the communicator

The sender arguments are meaningful only for root

Scatter


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

Scatter: changing the buffers size



MPI_SCATTERV

Usage

```
int MPI_Scatterv( void* sendbuf,              /* in */
                  int* sendcounts,            /* in */
                  int* displs,                /* in */
                  MPI_Datatype sendtype,      /* in */
                  void* recvbuf,              /* in */
                  int recvcount,              /* in */
                  MPI_Datatype recvtype,      /* in */
                  int root,                   /* in */
                  MPI_Comm comm);             /* in */
```

Description

- Distributes individual messages from root to each process in communicator
- **Messages can have different sizes and displacements**

MPI_Gather

One-to-all communication: different data collected by the root process, from all others processes in the communicator.

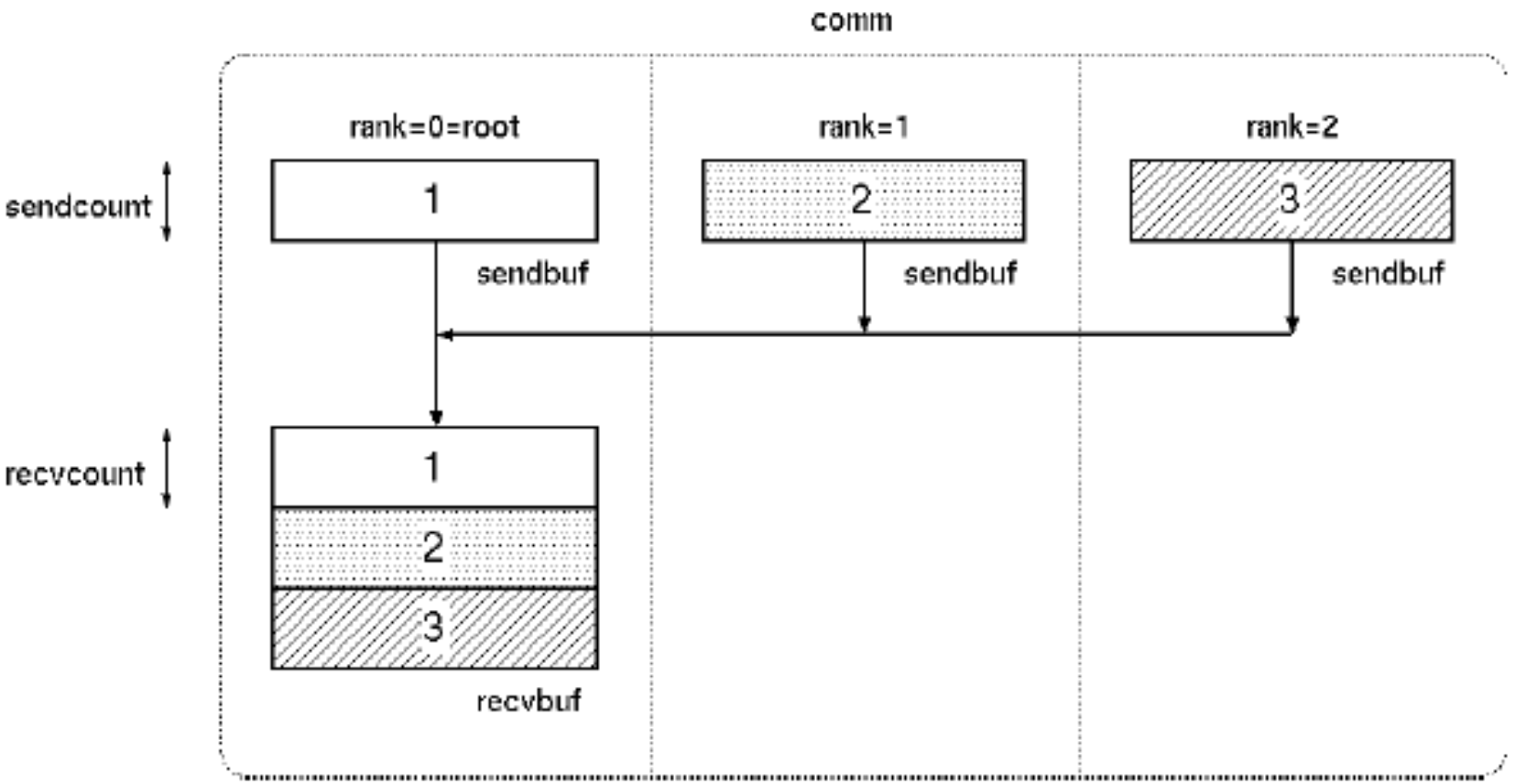
Fortran:

```
CALL MPI_GATHER(sndbuf, sndcount, sndtype, rcvbuf,
   rcvcount, rcvtype, root, comm, ierr)
```

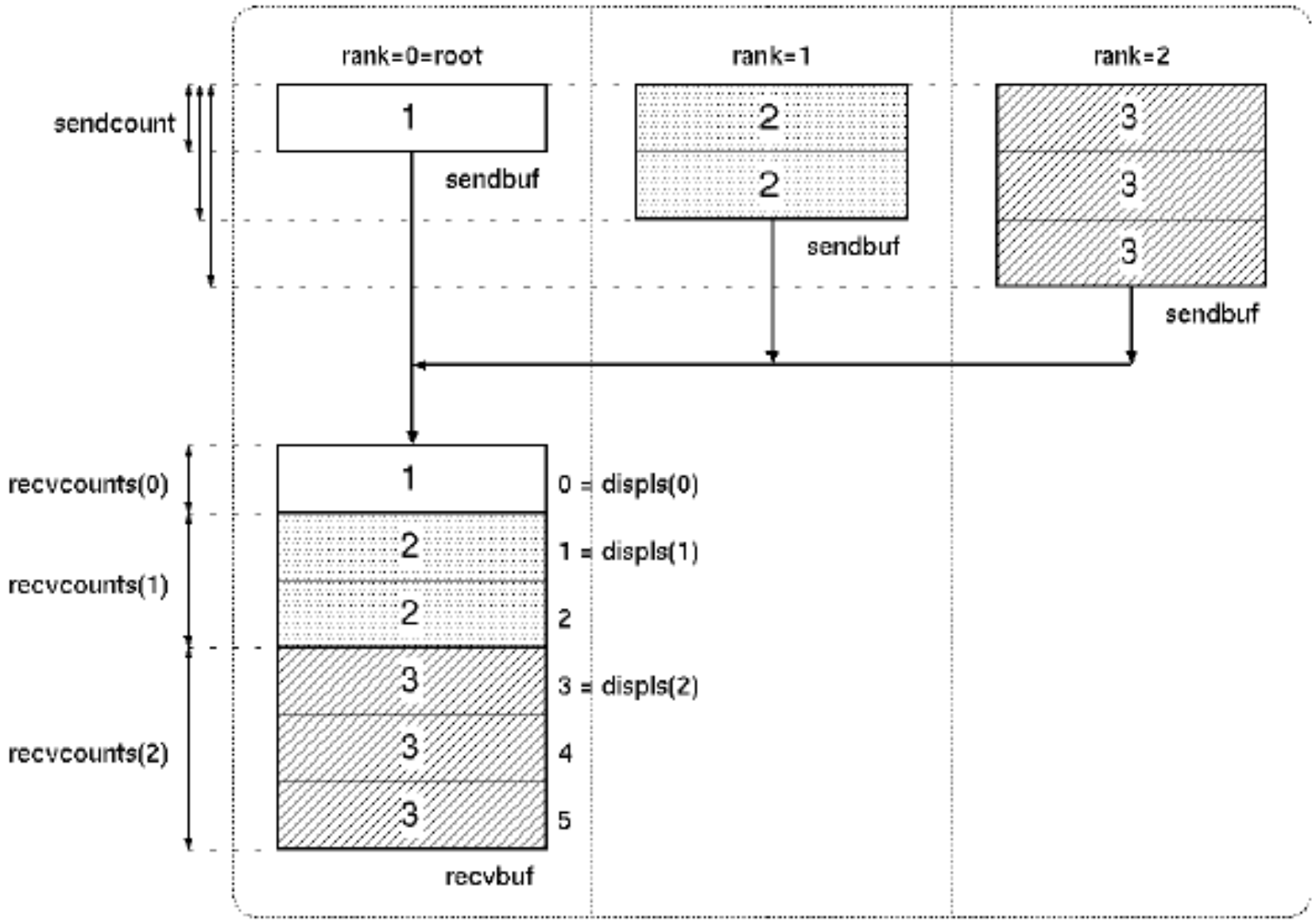
rcvcount is the number of elements collected from each process, not the size of rcvbuf, that should be rcvcount times the number of process in the communicator

The receiver arguments are meaningful only for root

Gather


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

Gather: changing the buffers size



MPI_GATHERV

Usage

```
int MPI_Gatherv ( void* sendbuf,          /* in */
                  int sendcount,          /* in */
                  MPI_Datatype sendtype,  /* in */
                  void* recvbuf,          /* out */
                  int* recvcount,         /* in */
                  int* displs,            /* in */
                  MPI_Datatype recvtype,  /* in */
                  int root,               /* in */
                  MPI_Comm comm );        /* in */
```

Description

- –Collects individual messages from each process in communicator to the root process and store them in rank order
- **–Messages can have different sizes and displacements**

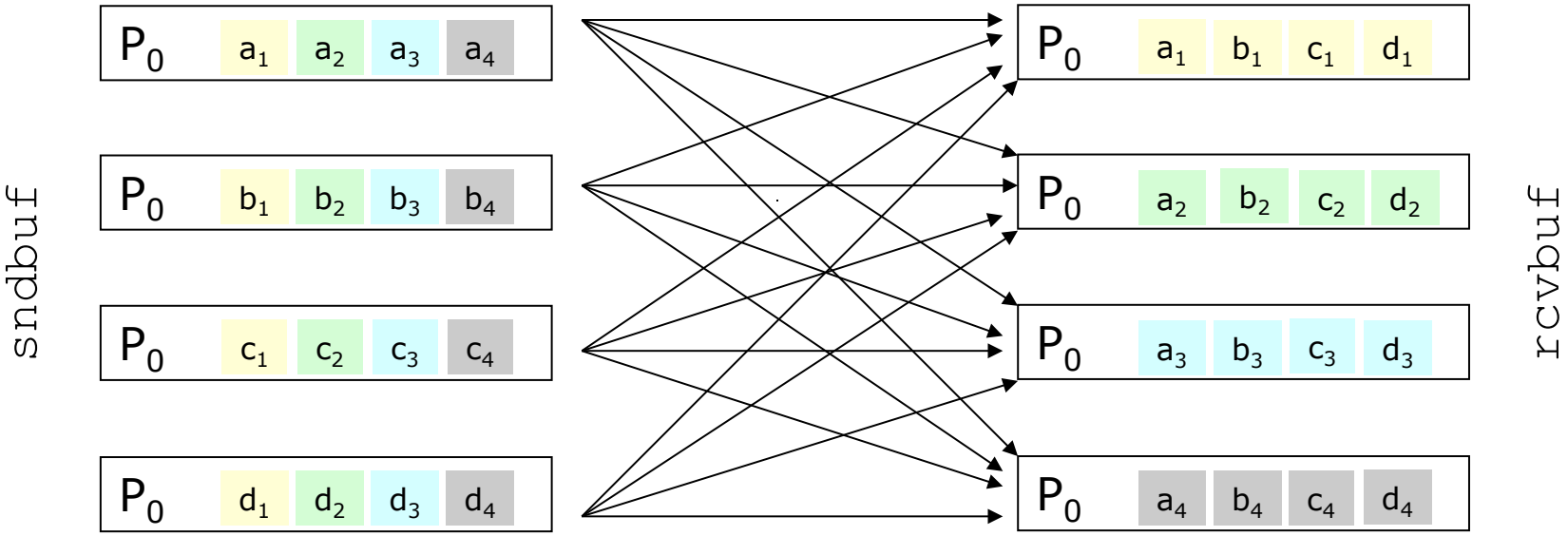
Exercise scatter_mpi

- Scatter: initialize an array and scatter it

MPI_Alltoall

Fortran:

```
CALL MPI_ALLTOALL(sndbuf, sndcount, sndtype, rcvbuf, rcvcount,
rcvtype, comm, ierr)
```



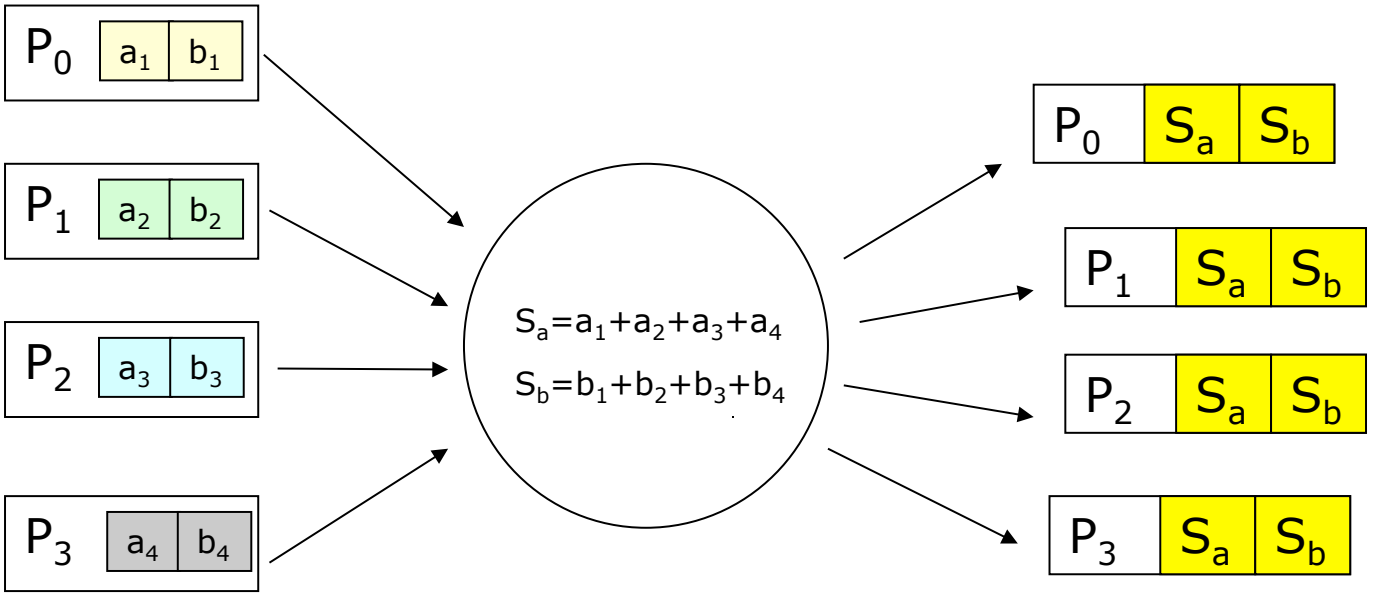
Useful, e.g., for data transposition

Reduction

The reduction operation allows to:

- Collect data from each process
- Reduce the data to a single value
- Store the result on the root processes
- Store the result on all processes
- Overlap communication and computation

Reduce, Parallel Sum



Reduction function works with arrays

other operation: product, min, max, and, ….

MPI_REDUCE and MPI_ALLREDUCE

Fortran:

MPI_REDUCE(snd_buf, rcv_buf, count, type, op, root, comm, ierr)

`snd_buf` input array of type `type` containing local values.
`rcv_buf` output array of type `type` containing global results
`Count` `(INTEGER)` number of element of `snd_buf` and `rcv_buf`
`type` `(INTEGER)` MPI type of `snd_buf` and `rcv_buf`
`op` `(INTEGER)` parallel operation to be performed
`root` `(INTEGER)` MPI id of the process storing the result
`comm` `(INTEGER)` communicator of processes involved in the operation
`ierr` `(INTEGER)` output, error code (if `ierr=0` no error occours)

MPI_ALLREDUCE(snd_buf, rcv_buf, count, type, op, comm, ierr)

The argument **`root`** is missing, the result is stored to all processes.

Predefined Reduction Operations

MPI op	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
MPI_MINLOC	Minimum and location

Exercise reduce_mpi, ring_allreduce.c

- Read data from terminal and reduce / allreduce