special communicator might be created for the collective operation so as to avoid interference with any on-going point-to-point communication at the time of the collective call. This is discussed further in Section 5.12. (*End of advice to implementors.*)

Many of the descriptions of the collective routines provide illustrations in terms of blocking MPI point-to-point routines. These are intended solely to indicate what data is sent or received by what process. Many of these examples are *not* correct MPI programs; for purposes of simplicity, they often assume infinite buffering.

## 5.2 Communicator Argument

The key concept of the collective functions is to have a group or groups of participating processes. The routines do not have group identifiers as explicit arguments. Instead, there is a communicator argument. Groups and communicators are discussed in full detail in Chapter 6. For the purposes of this chapter, it is sufficient to know that there are two types of communicators: *intra-communicators* and *inter-communicators*. An intracommunicator can be thought of as an indentifier for a single group of processes linked with a context. An intercommunicator identifies two distinct groups of processes linked with a context.

### 5.2.1 Specifics for Intracommunicator Collective Operations

All processes in the group identified by the intracommunicator must call the collective
ticket105. routine[with matching arguments].

In many cases, collective communication can occur "in place" for intracommunicators, with the output buffer being identical to the input buffer. This is specified by providing a special argument value, MPI_IN_PLACE, instead of the send buffer or the receive buffer argument, depending on the operation performed.

*Rationale.* The "in place" operations are provided to reduce unnecessary memory motion by both the MPI implementation and by the user. Note that while the simple check of testing whether the send and receive buffers have the same address will work for some cases (e.g., MPI_ALLREDUCE), they are inadequate in others (e.g., MPI_GATHER, with root not equal to zero). Further, Fortran explicitly prohibits aliasing of arguments; the approach of using a special value to denote "in place" operation eliminates that difficulty. (*End of rationale.*)

*Advice to users.* By allowing the "in place" option, the receive buffer in many of the collective calls becomes a send-and-receive buffer. For this reason, a Fortran binding that includes `INTENT` must mark these as `INOUT`, not `OUT`.

Note that MPI_IN_PLACE is a special kind of value; it has the same restrictions on its use that MPI_BOTTOM has.

Some intracommunicator collective operations do not support the "in place" option (e.g., MPI_ALLTOALLV). (*End of advice to users.*)

### 5.2.2 Applying Collective Operations to Intercommunicators

To understand how collective operations apply to intercommunicators, we can view most MPI intracommunicator collective operations as fitting one of the following categories (see, for instance, [43]):

**All-To-All** All processes contribute to the result. All processes receive the result.

- MPI_ALLGATHER, MPI_ALLGATHERV
- MPI_ALLTOALL, MPI_ALLTOALLV, MPI_ALLTOALLW
- MPI_ALLREDUCE, MPI_REDUCE_SCATTER
- MPI_BARRIER

ticket89.

**All-To-One** All processes contribute to the result. One process receives the result.

- MPI_GATHER, MPI_GATHERV
- MPI_REDUCE

**One-To-All** One process contributes to the result. All processes receive the result.

- MPI_BCAST
- MPI_SCATTER, MPI_SCATTERV

**Other** Collective operations that do not fit into one of the above categories.

- MPI_SCAN, MPI_EXSCAN [MPI-2.2 - ticket #89, passed Jun 8-10, 2009
- MPI_BARRIER ]

ticket89.

ticket89.

[MPI-2.2 - ticket #89, passed Jun 8-10, 2009 The MPI_BARRIER operation does not fit into this classification since no data is being moved (other than the implicit fact that a barrier has been called). ] The data movement patterns of MPI_SCAN and MPI_EXSCAN do not fit this taxonomy.

The application of collective communication to intercommunicators is best described in terms of two groups. For example, an all-to-all MPI_ALLGATHER operation can be described as collecting data from all members of one group with the result appearing in all members of the other group (see Figure 5.2). As another example, a one-to-all MPI_BCAST operation sends data from one member of one group to all members of the other group. Collective computation operations such as MPI_REDUCE_SCATTER have a similar interpretation (see Figure 5.3). For intracommunicators, these two groups are the same. For intercommunicators, these two groups are distinct. For the all-to-all operations, each such operation is described in two phases, so that it has a symmetric, full-duplex behavior.

The following collective operations also apply to intercommunicators:

- MPI_BARRIER,
- MPI_BCAST,
- MPI_GATHER, MPI_GATHERV,
- MPI_SCATTER, MPI_SCATTERV,
- MPI_ALLGATHER, MPI_ALLGATHERV,
- MPI_ALLTOALL, MPI_ALLTOALLV, MPI_ALLTOALLW,
- MPI_ALLREDUCE, MPI_REDUCE,

- MPI_REDUCE_SCATTER.

In C++, the bindings for these functions are in the `MPI::Comm` class. However, since the collective operations do not make sense on a C++ `MPI::Comm` (as it is neither an intercommunicator nor an intracommunicator), the functions are all pure virtual.

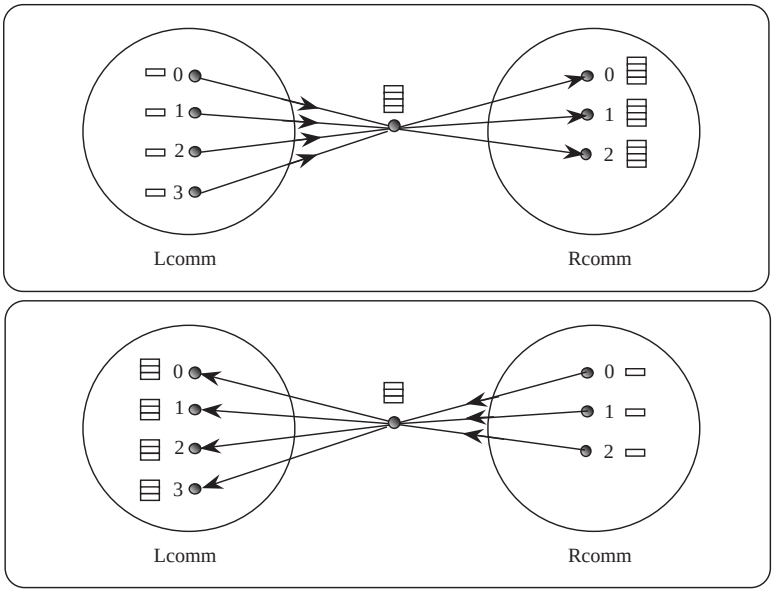


Figure 5.2: Intercommunicator allgather. The focus of data to one process is represented, not mandated by the semantics. The two phases do allgathers in both directions.

### 5.2.3 Specifics for Intercommunicator Collective Operations

All processes in both groups identified by the intercommunicator must call the collective routine. [In addition, processes in the same group must call the routine with matching arguments.] ticket105.

Note that the "in place" option for intracommunicators does not apply to intercommunicators since in the intercommunicator case there is no communication from a process to itself.

For intercommunicator collective communication, if the operation is [rooted (e.g., broadcast, gather, scatter)]in the All-To-One or One-To-All categories, then the transfer is unidirectional. The direction of the transfer is indicated by a special value of the root argument. In this case, for the group containing the root process, all processes in the group must call the routine using a special argument for the root. For this, the root process uses the special root value MPI_ROOT; all other processes in the same group as the root use MPI_PROC_NULL. All processes in the other group (the group that is the remote group relative to the root process) must call the collective routine and provide the rank of the root. If the operation is [unrooted (e.g., alltoall)]in the All-To-All category, then the transfer is bidirectional. ticket87. ticket87.

> *Rationale.* [Rooted operations]Operations in the All-To-One and One-To-All categories are unidirectional by nature, and there is a clear way of specifying direction. [Non-rooted operations, such as all-to-all,]Operations in the All-To-All category will often occur as part of an exchange, where it makes sense to communicate in both directions at once. (*End of rationale.*) ticket87. ticket87.