NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Programming Techniques for Supercomputers:

More MPI – collective communication



Prof. Dr. G. Wellein[a,b] , Dr. G. Hager[a], A. Afzal[b], C.L. Alappat[b], D. Ernst[b]
[a] Erlangen National High Performance Computing Center (NHR@FAU)
[b] Department für Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg, Sommersemester 2022

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

More MPI – collective communication

Introduction to collectives in MPI



Collectives in MPI

Collectives: operations including all ranks of a communicator

All ranks must call the function!

- Blocking variants: buffer can be reused after return
- Nonblocking variants (since MPI 3.0):
  buffer can be used after completion (`MPI_Wait*`/`MPI_Test*`)

- May or may not synchronize the processes
- Cannot interfere with point-to-point communication
  - Completely separate modes of operation!

Collectives in MPI

- Rules for all collectives
  - Data type matching
  - No tags
  - Count must be exact, i.e., there is only one message length, buffer must be large enough
- Types:
  - Synchronization (barrier)
  - Data movement (broadcast, scatter, gather, all to all)
  - Collective computation (reduction, scan)
  - Combinations of data movement and computation (reduction + broadcast)
- General assumption: MPI does a better job at collectives than you trying to emulate them with point-to-point calls

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

More MPI – collective communication

Global communication



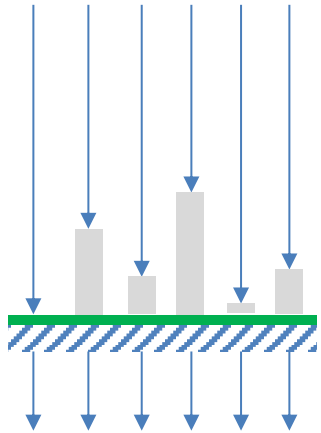
Barrier

- Explicit synchronization of all ranks from specified communicator

```
MPI_Barrier(comm);
```

- Ranks only return from call after every rank has called the function

- `MPI_Barrier()` rarely needed
  - Debugging

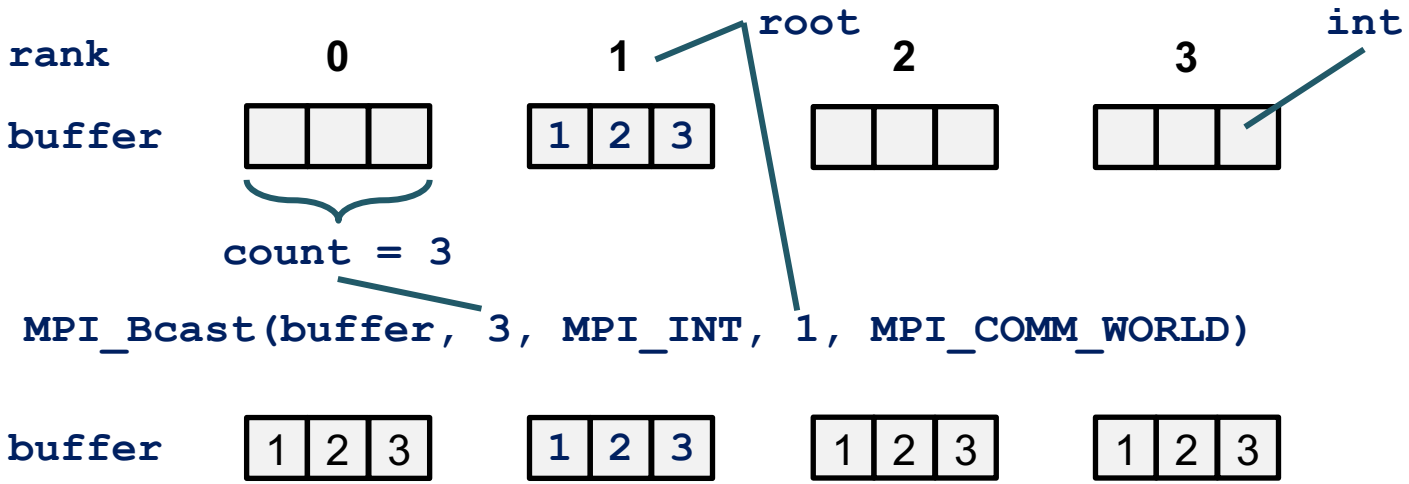


Broadcast

- Send buffer contents from one rank (“root”) to all ranks

```
MPI_Bcast(buf, count, datatype, int root, comm);
```

- no restrictions on which rank is root – often rank 0



Scatter

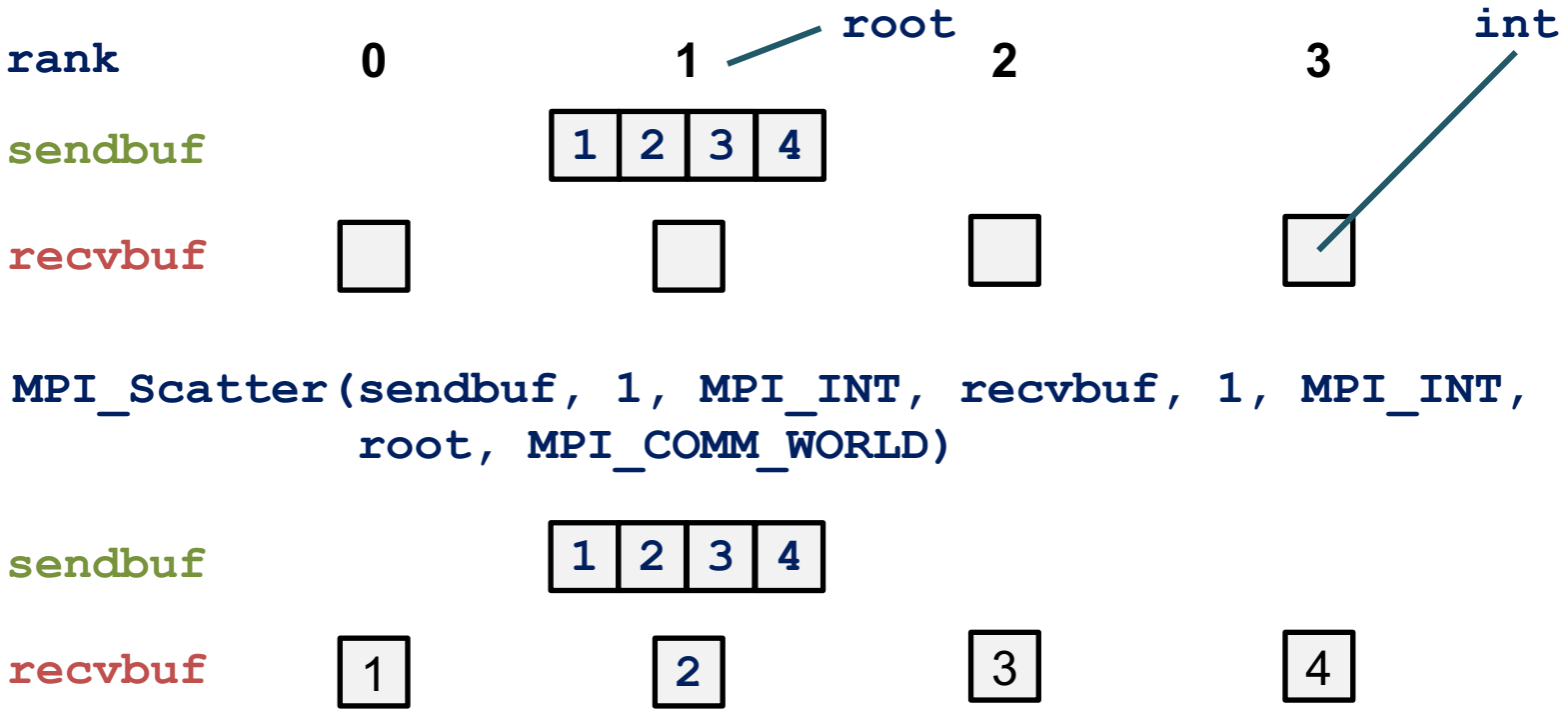
- Send the i-th chunk of an array to the i-th rank

```
MPI_Scatter(sendbuf, sendcount, sendtype,
            recvbuf, recvcount, recvtype,
            root, comm);
```

- In general, **`sendcount`** = **`recvcount`**
  - This is the length of the chunk

- **`sendbuf`** is ignored on non-root ranks because there is nothing to send

Scatter


rank
0
1
root
2
3
int
sendbuf
1 2 3 4
recvbuf
MPI_Scatter(sendbuf, 1, MPI_INT, recvbuf, 1, MPI_INT,
root, MPI_COMM_WORLD)
sendbuf
1 2 3 4
recvbuf
1
2
3
4

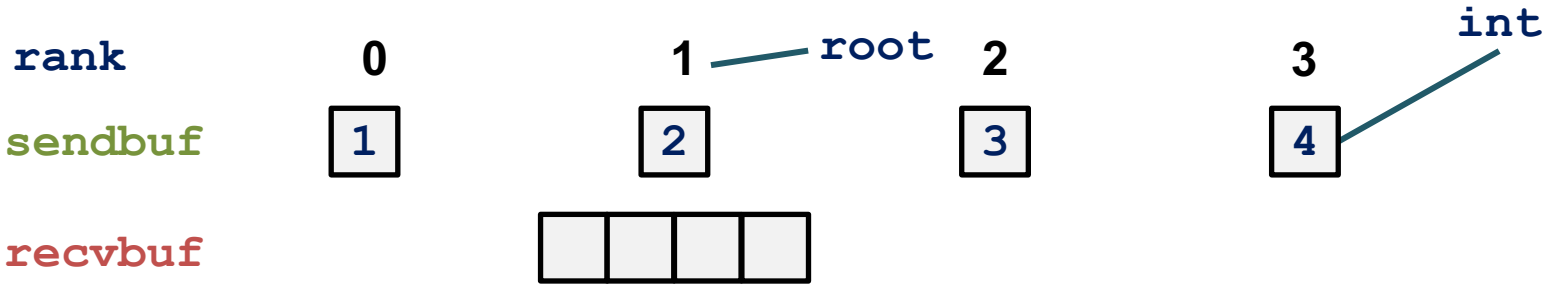
Gather

- Receive a message from each rank and place i-th rank's message at i-th position in receive buffer

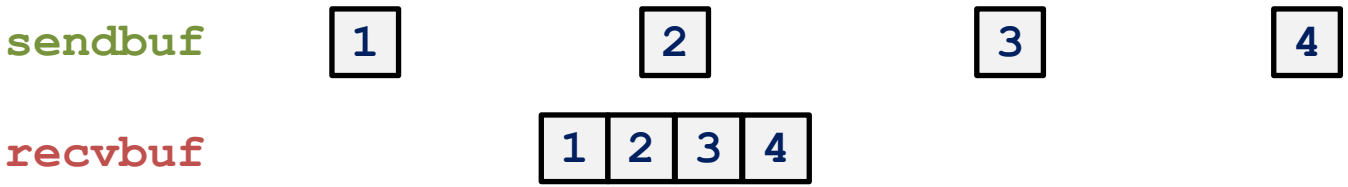
```
MPI_Gather(sendbuf, sendcount, sendtype,
           recvbuf, recvcount, recvtype,
           root, comm)
```

- In general, **`sendcount`** = **`recvcount`**
- **`recvbuf`** is ignored on non-root ranks because there is nothing to receive

Gather




```
MPI_Gather(sendbuf, 1, MPI_INT, recvbuf, 1, MPI_INT,
           root, MPI_COMM_WORLD)
```



Scatterv: more flexible scatter

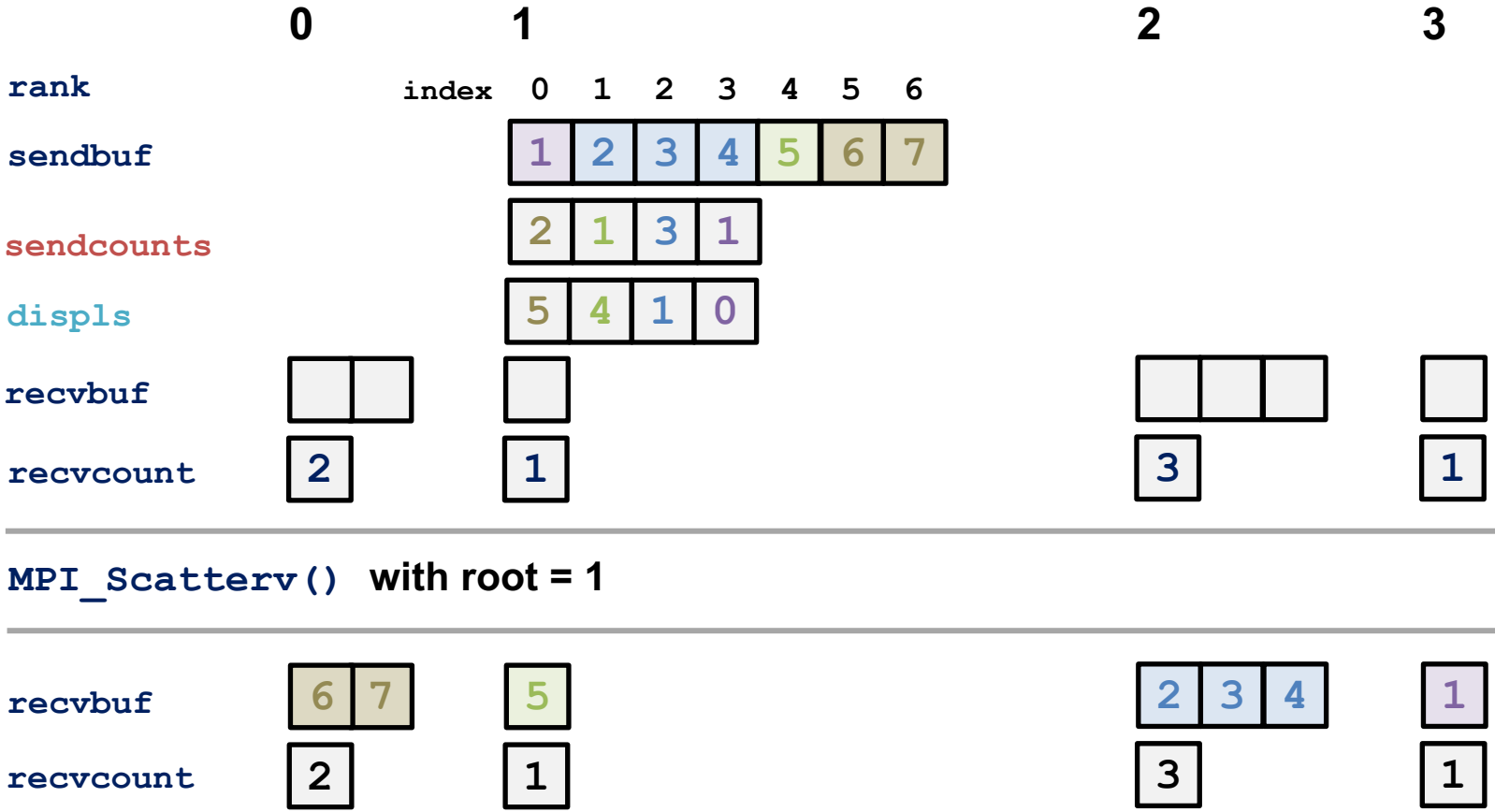
- Send chunks of different sizes to different ranks

```
MPI_Scatterv(
    sendbuf, int sendcounts[], int displs[], sendtype,
    recvbuf, recvcount, recvtype,
    root, comm);
```

`sendcounts[]`: array specifying the number of elements to send to each rank: send **`sendcounts[i]`** elements to rank **`i`**

`displs[]`: integer array specifying the displacements in **`sendbuf`** from which to take the outgoing data to each rank, specified in number of elements

Scatterv


0
1
2
3
rank
index 0 1 2 3 4 5 6
sendbuf
1 2 3 4 5 6 7
sendcounts
2 1 3 1
displs
5 4 1 0
recvbuf
recvcount
2
1
3
1
MPI_Scatterv() with root = 1
recvbuf
6 7
5
2 3 4
1
recvcount
2
1
3
1

Gatherv: more flexible gather

- Receive segments of different sizes from different ranks

```
MPI_Gatherv(
    sendbuf, sendcount, sendtype,
    recvbuf, int recvcounts[], int displs[], recvtype,
    root, comm)
```

`recvcounts[]`: array specifying the number of elements to receive from each rank: receive **`recvcounts[i]`** elements from rank **`i`**

`displs[]`: integer array specifying the displacements where received data from specific rank is put in **`recvbuf`**, in units of elements:

Allgather

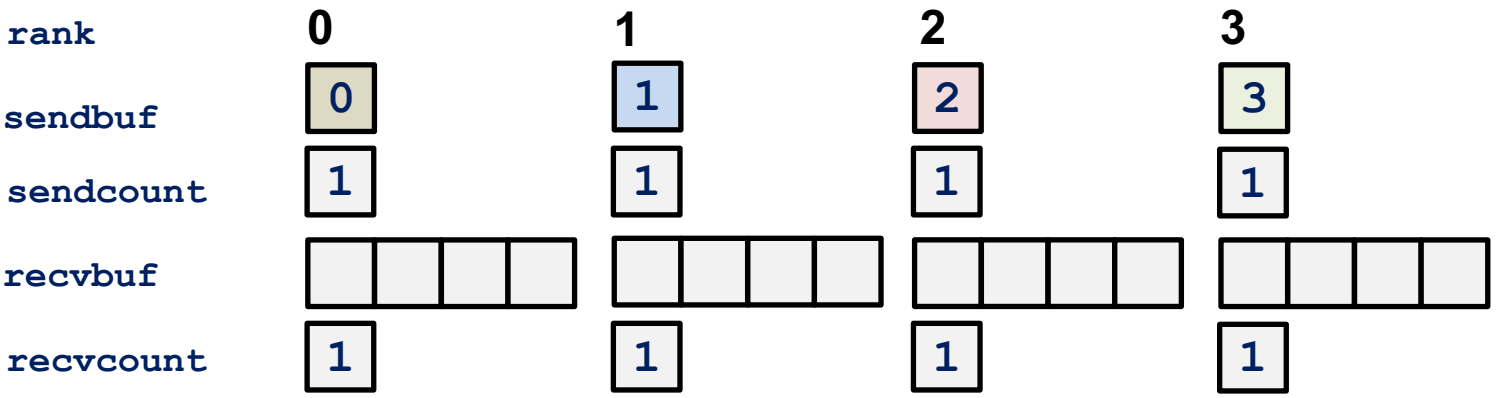
- Combination of gather and broadcast

```
MPI_Allgather(sendbuf, sendcount, sendtype,
              recvbuf, recvcount, recvtype,
              comm);
```

- Also available: `MPI_Allgatherv()` (cf. `MPI_Gatherv()`)

- Why not just use gather followed by a broadcast instead?
  - MPI library has more options for optimization
  - General assumption: Combined collectives are faster than using separate ones

Allgather



`MPI_Allgather()` (no root required)



Alltoall

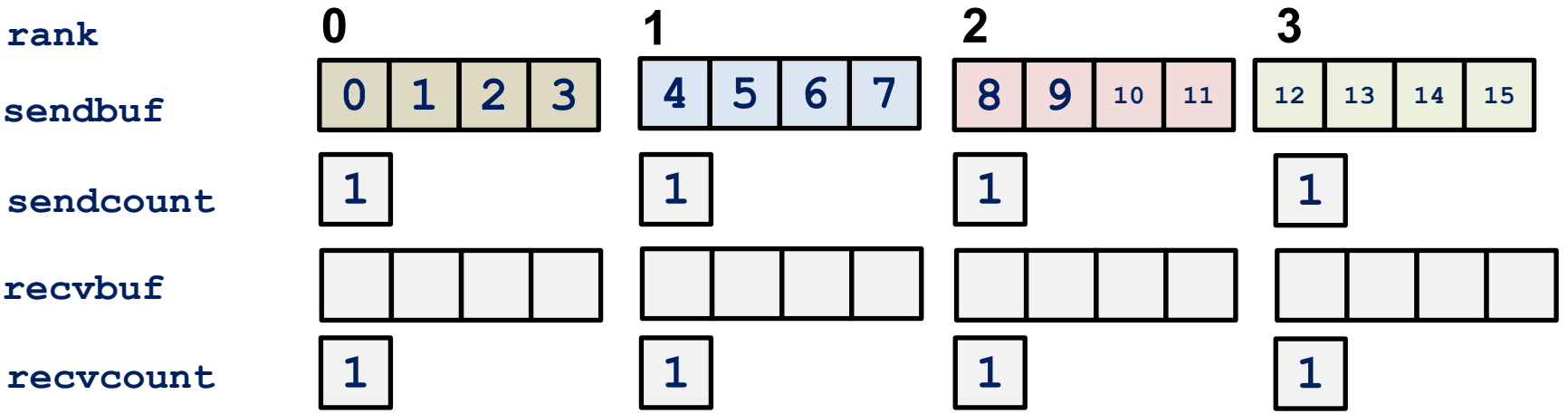
- `MPI_Alltoall`: For all ranks, send i-th chunk to i-th rank

```
MPI_Alltoall(sendbuf, sendcount, sendtype,
             recvbuf, recvcount, recvtype,
             comm);
```

- `MPI_Alltoallv`: Allows different number of elements to be send/received by each rank

- `MPI_Alltoallw`: Allows also different data types and displacements in bytes

Alltoall



`MPI_Alltoall()` (no root required)



NHR FAU

Friedrich-Alexander-Universität Erlangen-Nürnberg

More MPI – collective communication

Global operations

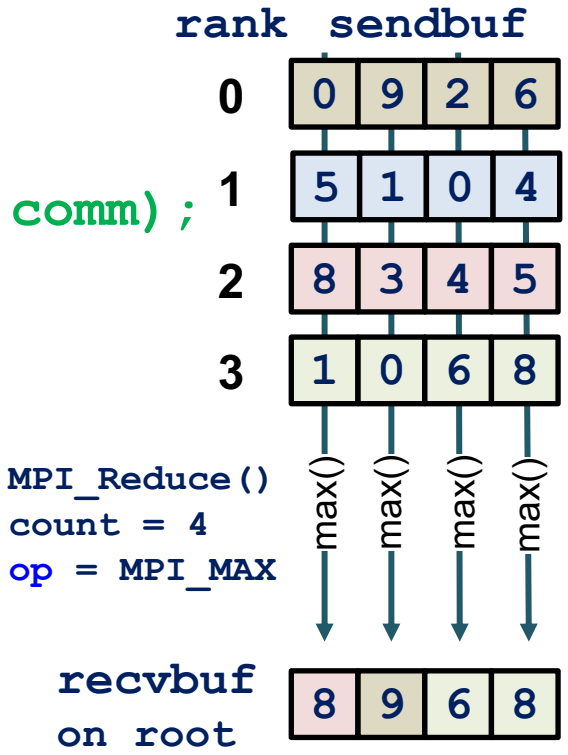


Global operations: reduction

- Compute results over distributed data

```
MPI_Reduce(sendbuf, recvbuf, count,
           datatype, MPI_Op op, root, comm);
```

- Result in `recvbuf` only available on root process
- Perform operation on all `count` elements of an array
- If all ranks require result, use `MPI_Allreduce()`



- If the 12 predefined ops are not enough use `MPI_Op_create`/`MPI_Op_free` to create own ones

Global operations – predefined operators

Name	Operation	Name	Operation
`MPI_SUM`	Sum	`MPI_PROD`	Product
`MPI_MAX`	Maximum	`MPI_MIN`	Minimum
`MPI_LAND`	Logical AND	`MPI_BAND`	Bit-AND
`MPI_LOR`	Logical OR	`MPI_BOR`	Bit-OR
`MPI_LXOR`	Logical XOR	`MPI_BXOR`	Bit-XOR
`MPI_MAXLOC`	Maximum+Position	`MPI_MINLOC`	Minimum+Position

- Define own operations with `MPI_Op_create`/`MPI_Op_free`
- MPI assumes that the operations are associative
  → be careful with floating-point operations

“In-place” buffer specification

Override local input buffer with a result

MPI_Reduce

```
int partial_sum = …, total_sum;
MPI_Reduce(&partial_sum, &total_sum,
           1, MPI_INT,
           MPI_SUM, root, comm);
```

```
int partial_sum = …, total_sum;
if (rank == root) {
  total_sum = partial_sum;
  MPI_Reduce(MPI_IN_PLACE, &total_sum,
             1, MPI_INT,
             MPI_SUM, root, comm);
}
else {
  MPI_Reduce(&partial_sum, &total_sum,
             1, MPI_INT,
             MPI_SUM, root, comm);
}
```

MPI_Allreduce

```
int partial_sum = …, total_sum;
MPI_AllReduce(&partial_sum, &total_sum,
              1, MPI_INT,
              MPI_SUM, comm);
```

```
int partial_sum = …, total_sum;

total_sum = partial_sum;
MPI_AllReduce(MPI_IN_PLACE, &total_sum,
              1, MPI_INT,
              MPI_SUM, comm);
```

MPI_IN_PLACE cheat sheet

Function	MPI_IN_PLACE argument	@ rank(s)	Comment [MPI 3.0]
MPI_GATHER	send buffer	root	Recv value at root already in the correct place in receive buffer.
MPI_GATHERV	send buffer	root	Recv value at root already in the correct place in receive buffer.
MPI_SCATTER	receive buffer	root	Root-th segment of send buffer is not moved.
MPI_SCATTERV	receive buffer	root	Root-th segment of send buffer is not moved.
MPI_ALLGATHER	send buffer	all	Input data at the correct place were process would receive its own contribution.
MPI_ALLGATHERV	send buffer	all	Input data at the correct place were process would receive its own contribution.
MPI_ALLTOALL	send buffer	all	Data to be sent is taken from receive buffer and replaced by received data, data sent/received must be of the same type map specified in receive count/receive type.
MPI_ALLTOALLV	send buffer	all	Data to be sent is taken from receive buffer and replaced by received data. Data sent/received must be of the same type map specified in receive count/receive type. The same amount of data and data type is exchanged between two processes.
MPI_REDUCE	send buffer	root	Data taken from receive buffer, replaced with output data.
MPI_ALLREDUCE	send buffer	all	Data taken from receive buffer, replaced with output data.

Summary of MPI collective communication

- MPI (blocking) collectives
  - All ranks in communicator must call the function
- Communication and synchronization
  - Barrier, broadcast, scatter, gather, and combinations thereof
- Global operations
  - Reduce, allreduce, some more…
- In-place buffer specification `MPI_IN_PLACE`
  - Save some space if need be