CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Message Passing Interface (MPI)

Summer School 2016 – Effective High Performance Computing

Maxime Martinasso, CSCS

July 20 – 21, 2016

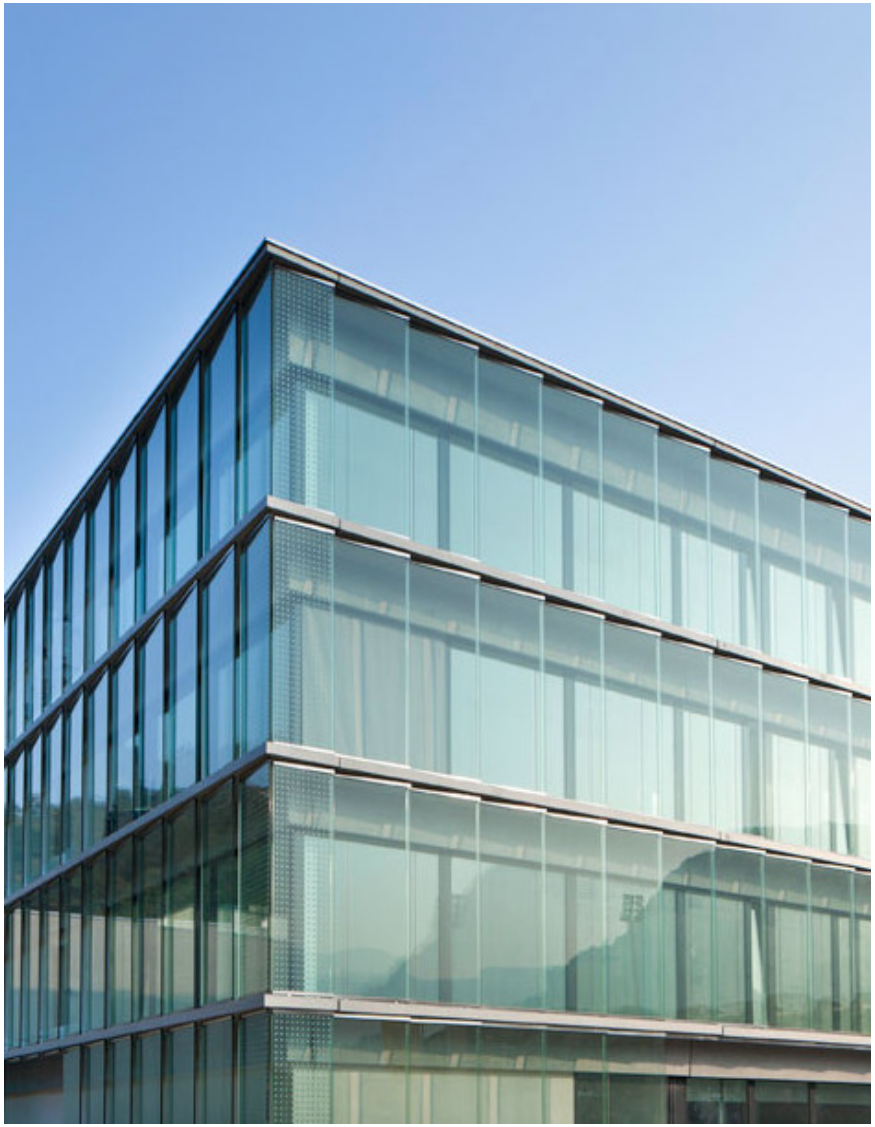
Previous course summary

- Point-to-point communication
- Blocking and non-blocking communication
- Transfer modes

Course Objectives

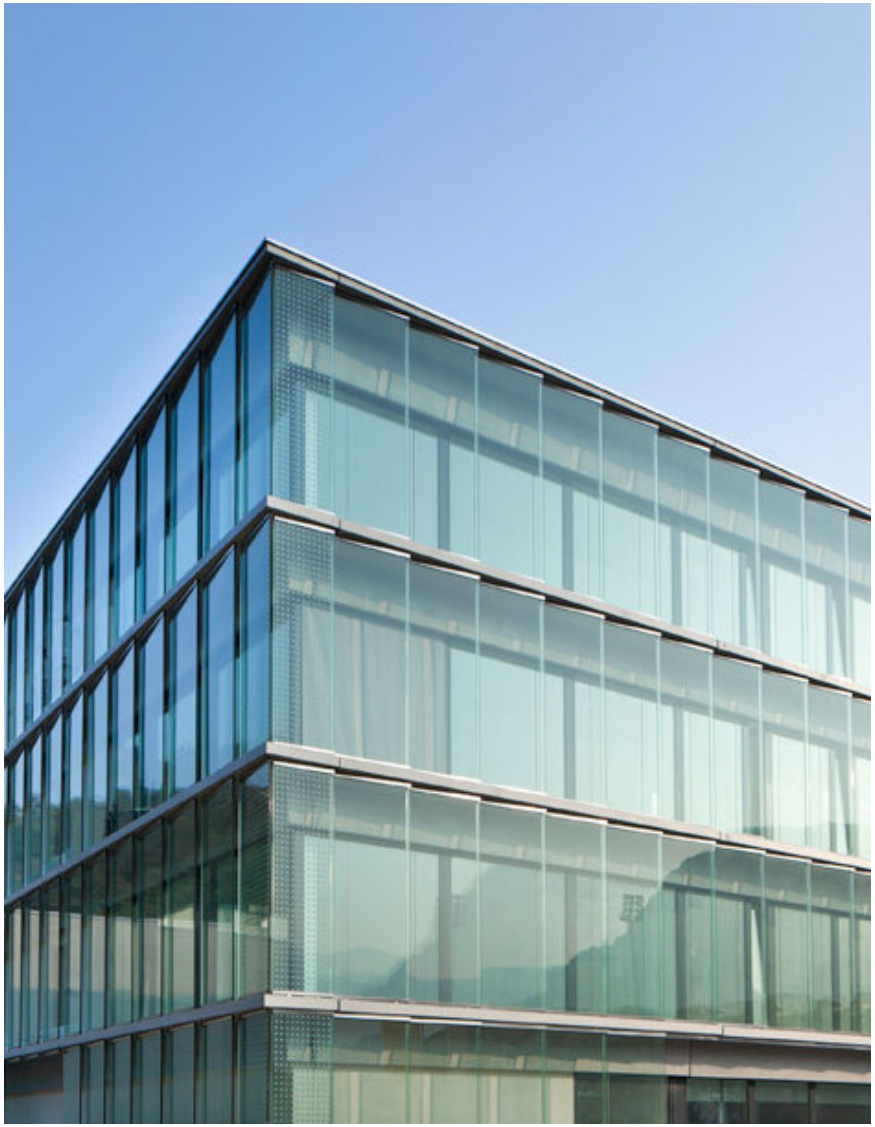
- The understanding of a collective operations
- Knowledge of the different collective operations

General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Topology
- Datatypes

General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
  - Collective communications
  - Barrier
  - Broadcast
  - Scatter/Gather
  - All to all
  - Reduction
  - Global collective operations
  - Non-blocking coll-op
- Topology
- Datatypes

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Collective communications

Collective operations

Communications involving a group of processes part of a communicator.
Different algorithms: $1 \rightarrow N$, $N \rightarrow 1$ or $N \rightarrow N$ ($1 \rightarrow 1$ = pt2pt).
Example:

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

Features:

- All processes must call the collective routine, one is the root
- No tags

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

Barrier

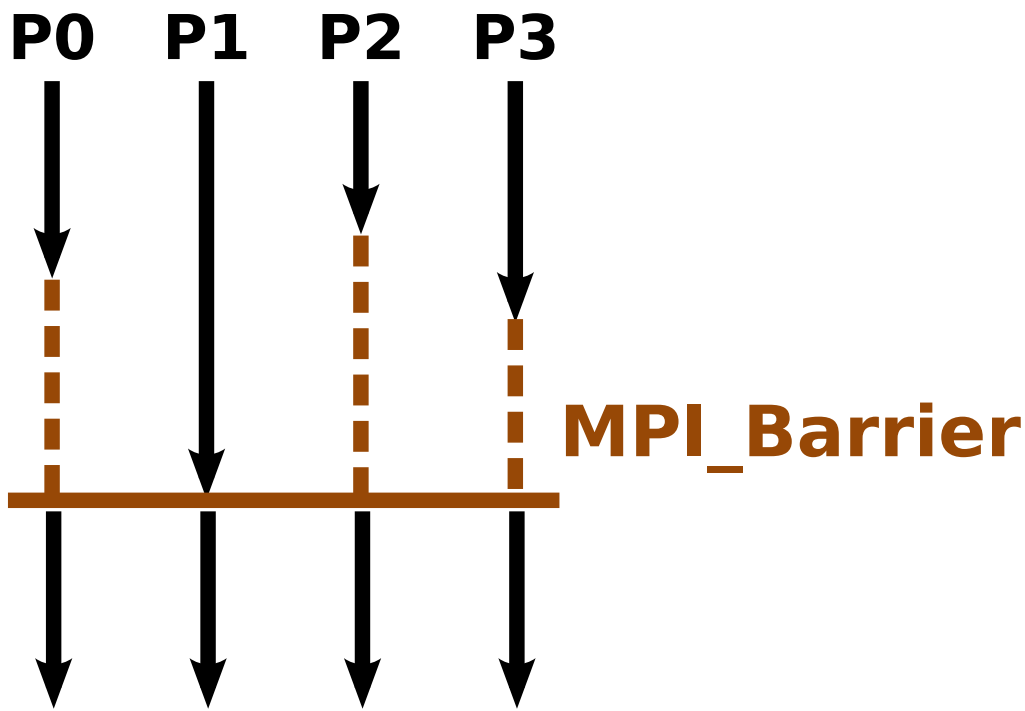
Stop processes until all processes within a communicator reach the barrier.

C/C++

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

Fortran

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

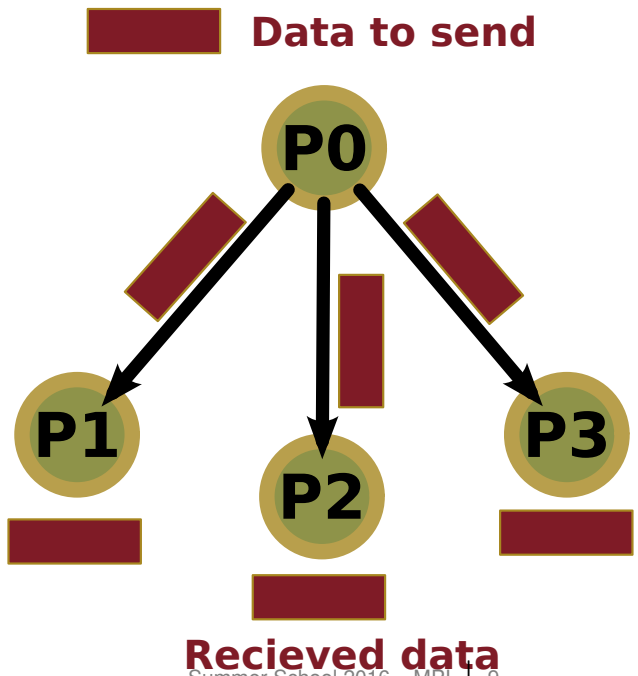


Broadcast

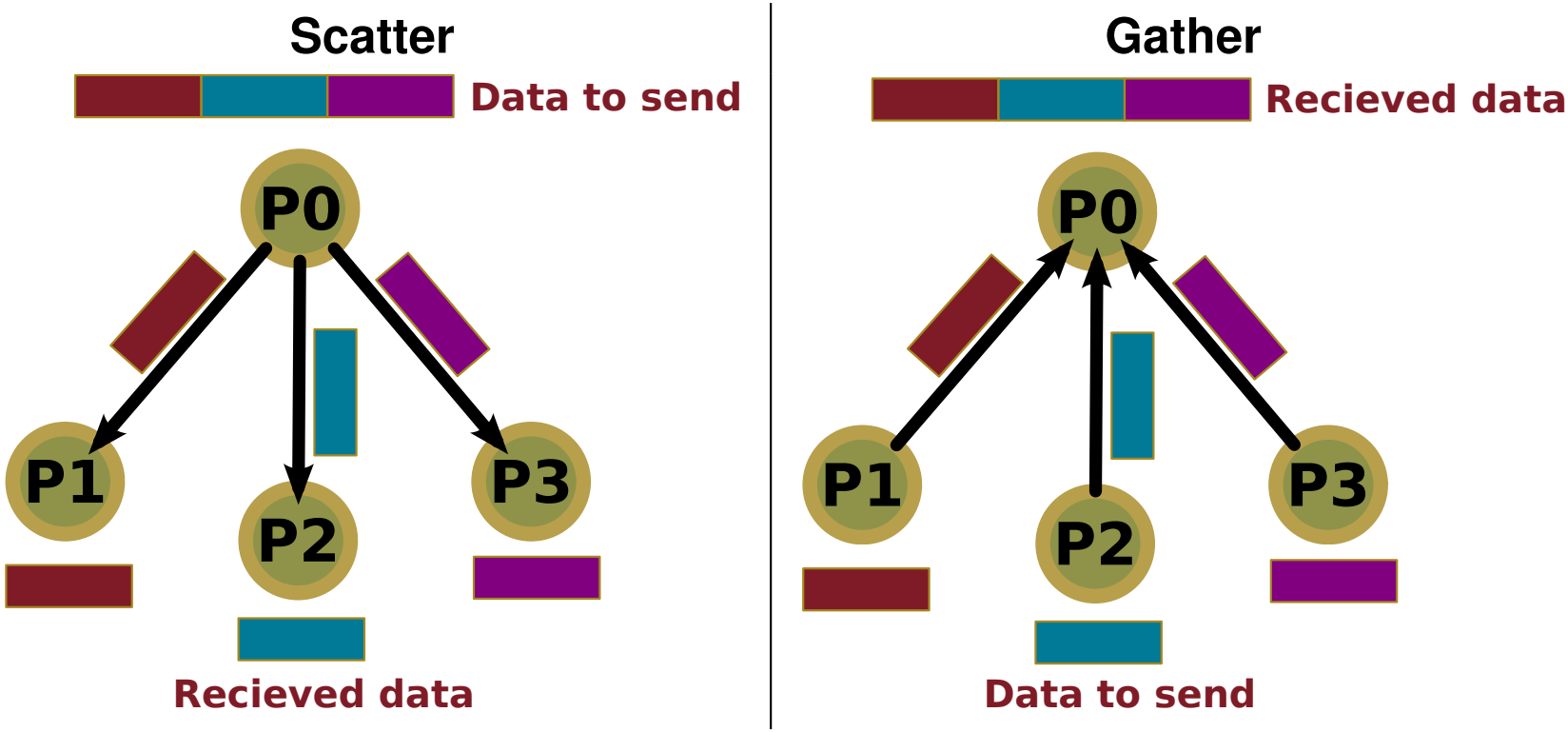
One-to-all communication: same data sent from root process to all other processes in the communicator.

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

root	(INTEGER) rank being the initiator of the collective operation



Scatter and Gather



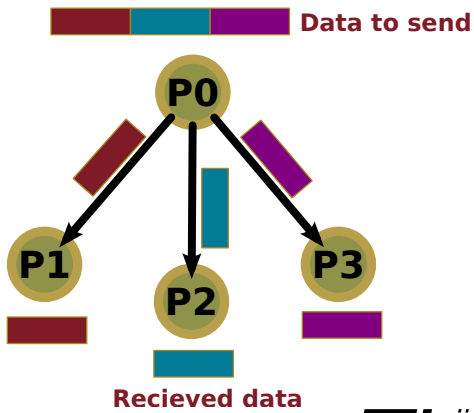
Scatter

One-to-all communication: different data sent from the root process to all other processes in the communicator.

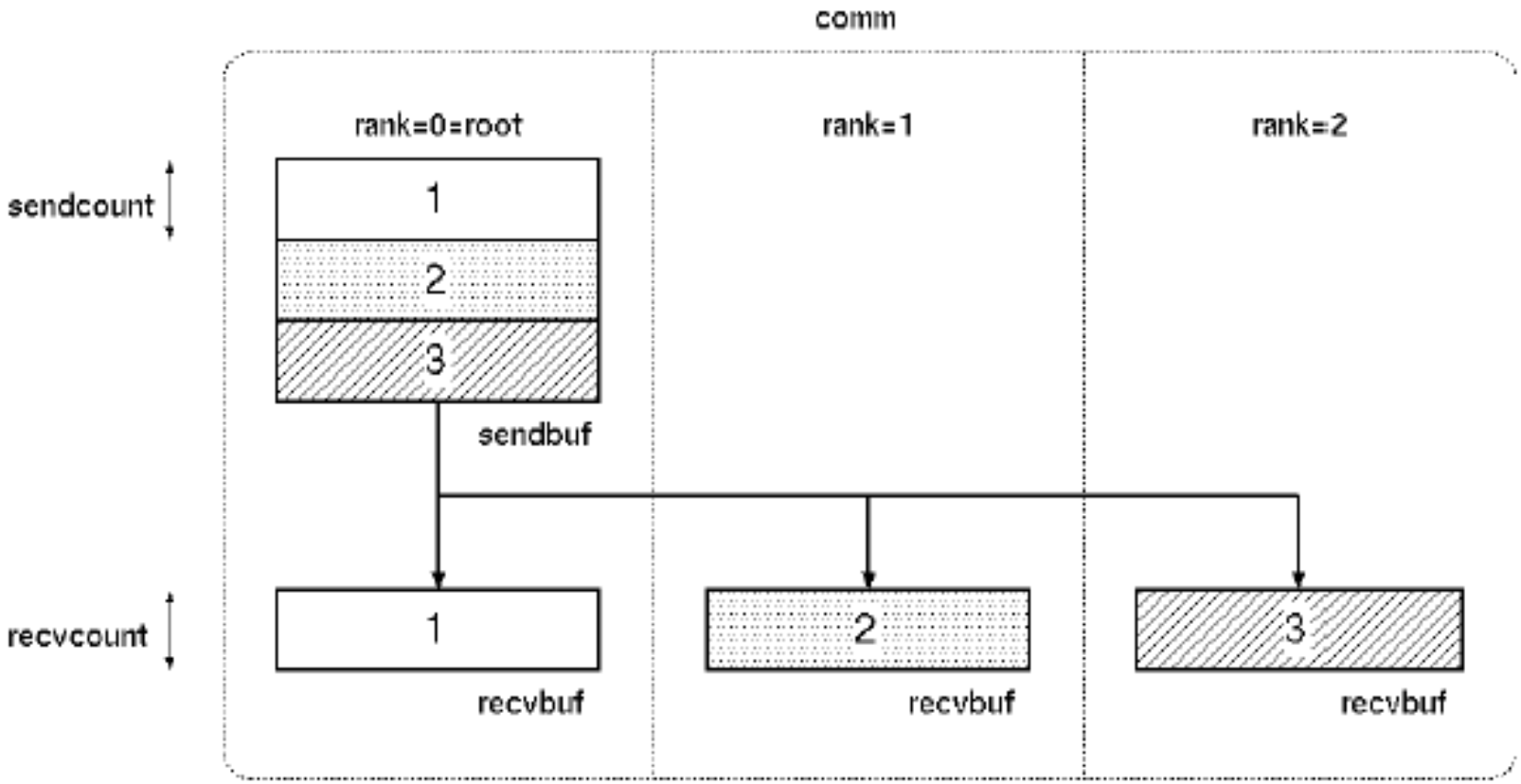
```
Fortran
CALL MPI_SCATTER(sndbuf, sndcount, sndtype,
                 rcvbuf, rcvcount, rcvtype,
                 root, comm, ierr)
```

sndcount	(INTEGER) number of elements sent to each process, not the size of sndbuf, that should be sndcount times the number of process in the communicator
rcvcount	(INTEGER) number of element in the receive buffer

The sender arguments are meaningful only for root.



Scatter with identical buffer size

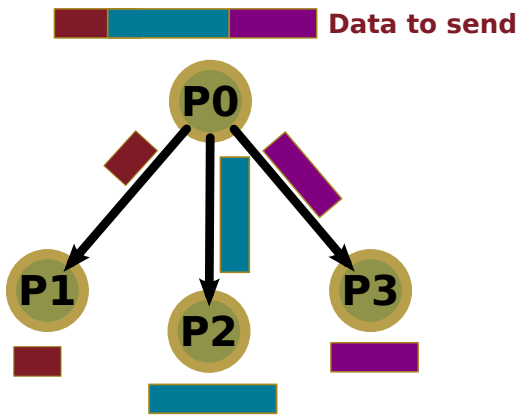


Scatter with different buffers size

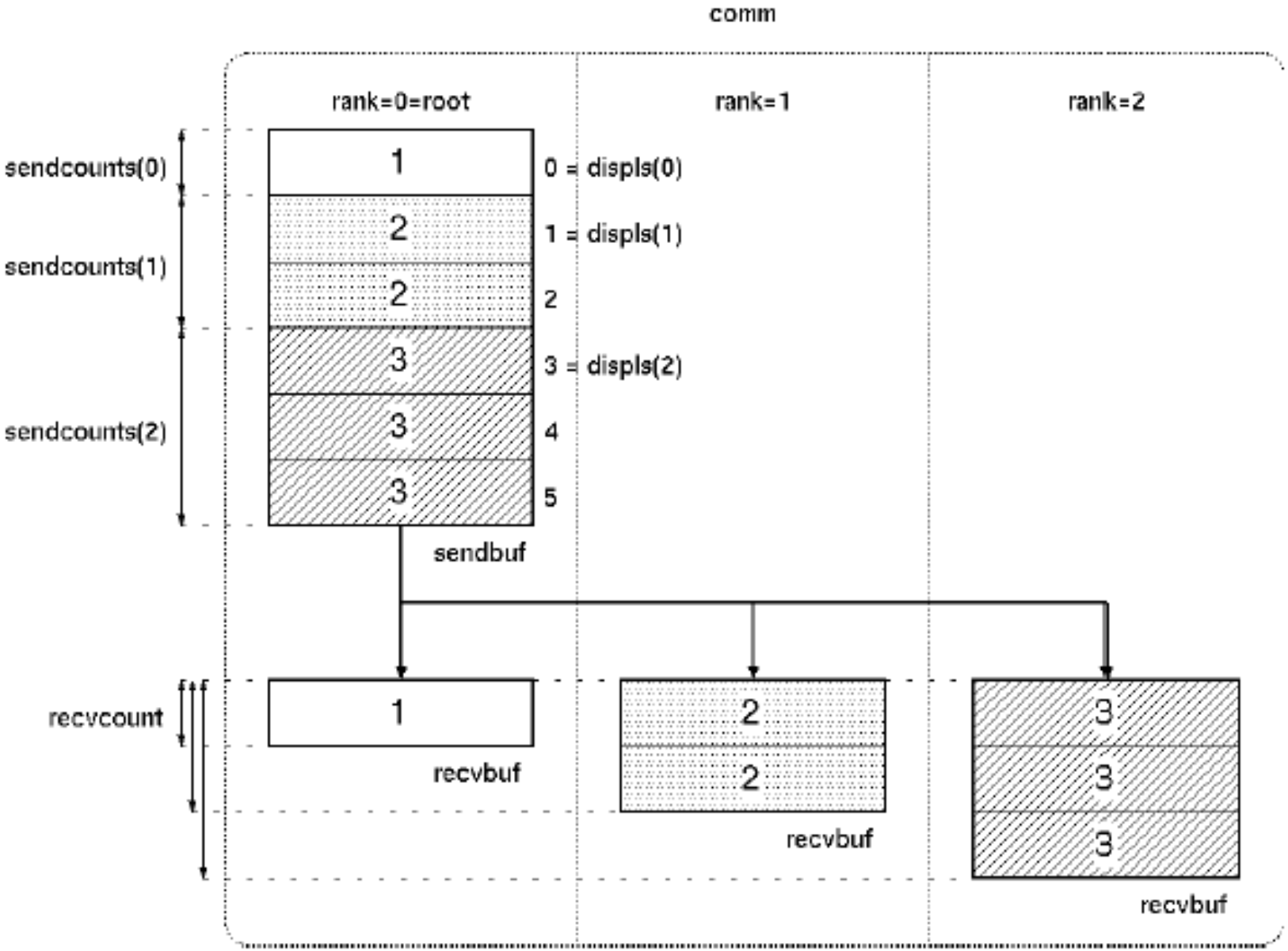
One-to-all communication: Scatter + distributes individual messages from root to each process in communicator. Messages can have different sizes and displacements.

```
Fortran
CALL MPI_SCATTERV(sndbuf, sndcount, displs, sndtype,
                  rcvbuf, rcvcount, rcvtype,
                  root, comm, ierr)
```

displs (INTEGER) entry i specifies the displacement (relative to sendbuf) from which to take the outgoing data to process i.



Scatter with different buffers size



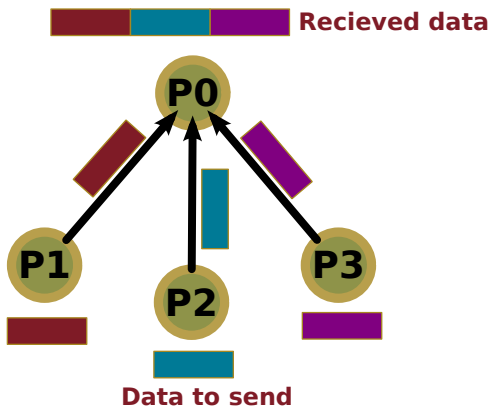
Gather

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

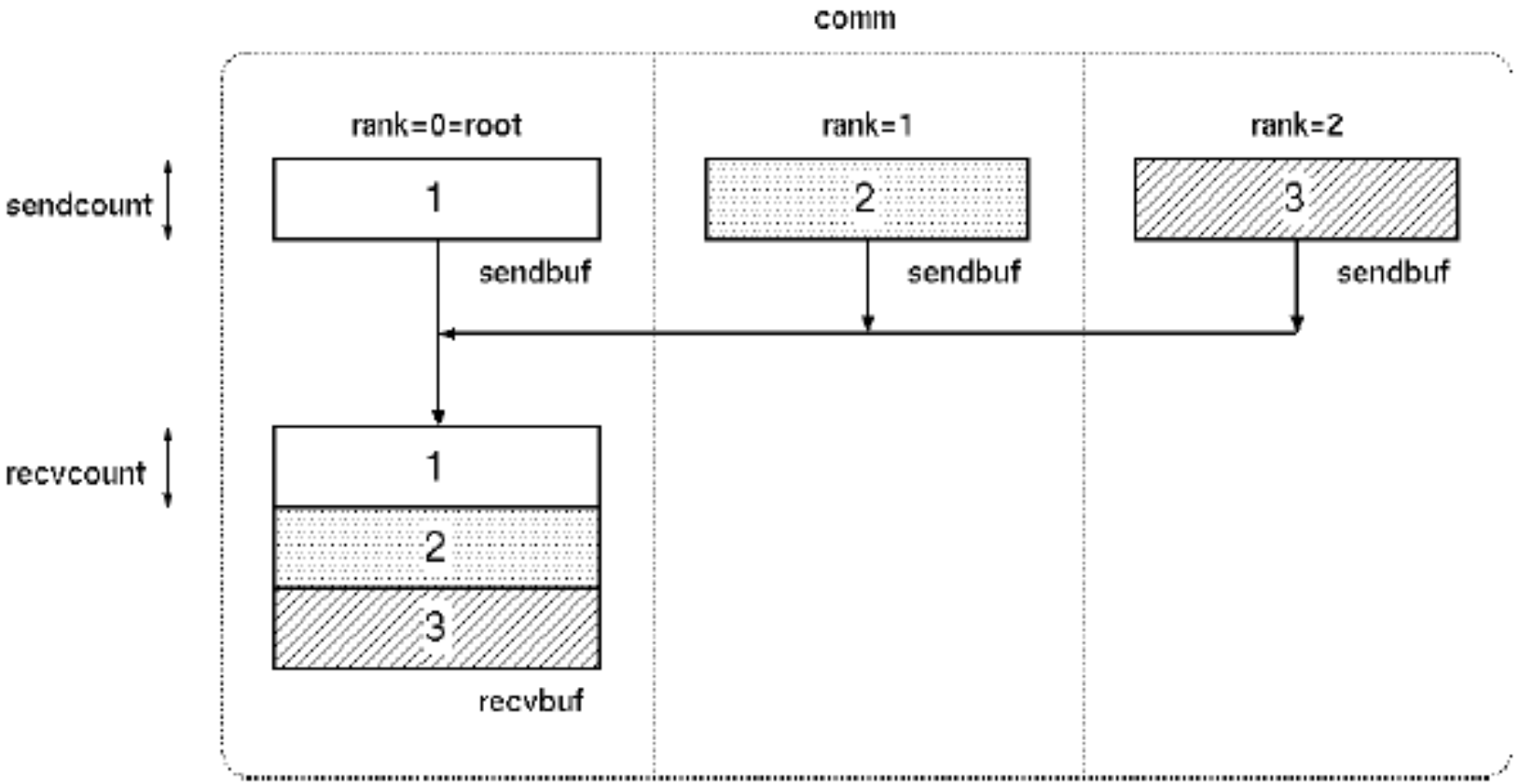
```
Fortran
CALL MPI_GATHER(sndbuf, sndcount, sndtype,
                rcvbuf, rcvcount, rcvtype,
                root, comm, ierr)
```

rcvcount	(INTEGER) 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
sndcount	(INTEGER) number of element in the send buffer

The receive arguments are meaningful only for root.



Gather with identical buffer size

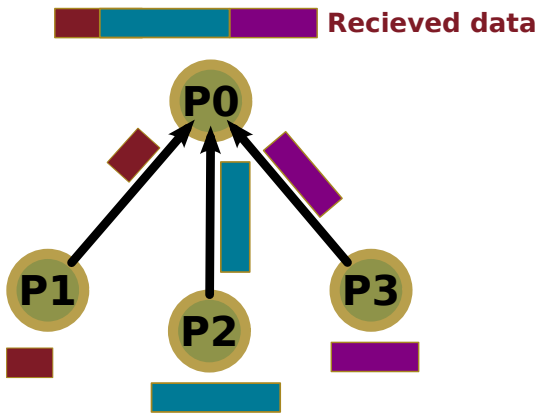


Gather with different buffers size

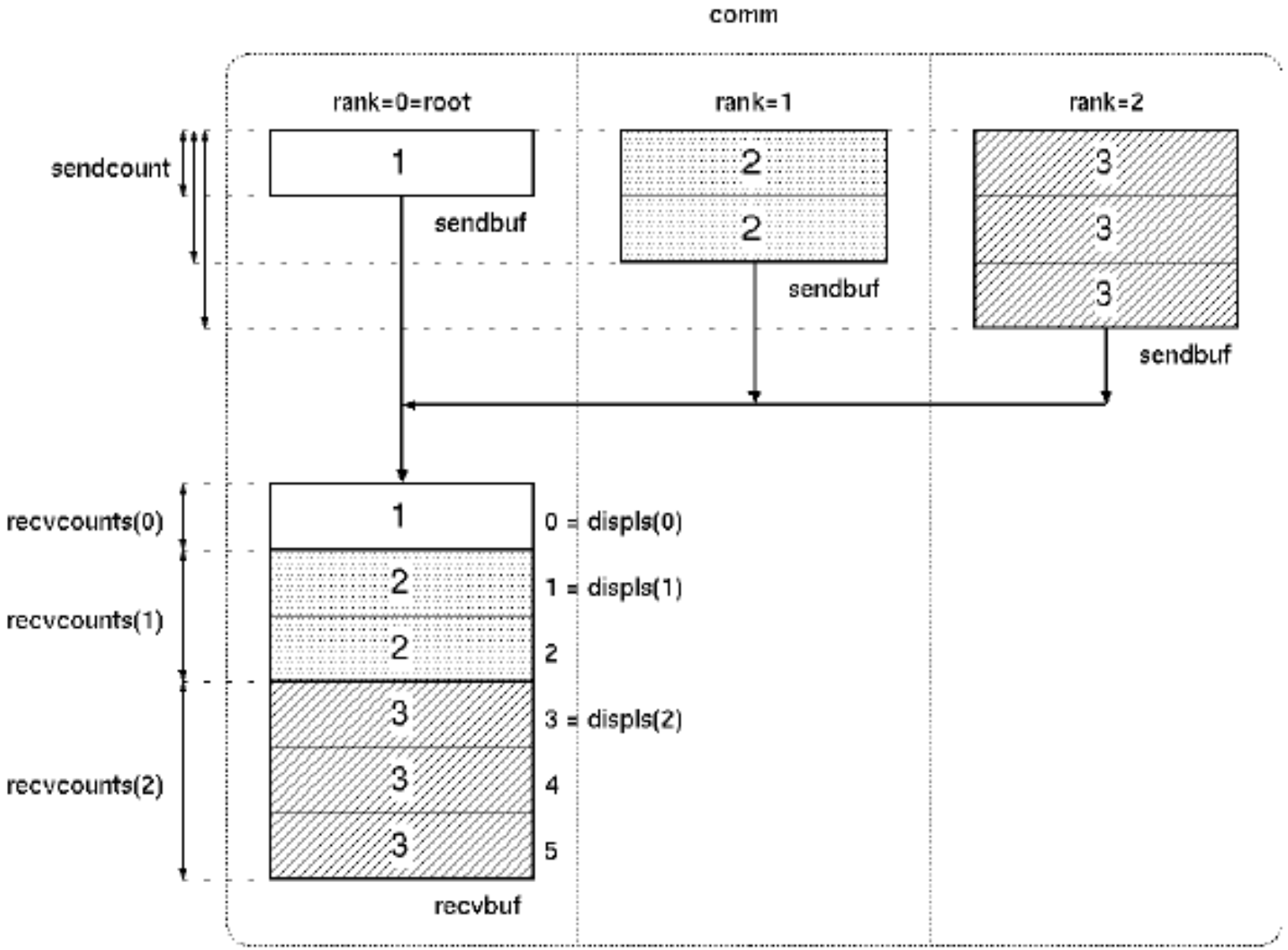
One-to-all communication: Gather + collects individual messages from each process in communicator to the root process and store them in rank order. Messages can have different sizes and displacements.

```
Fortran
CALL MPI_GATHERV(sndbuf, sndcount, sndtype,
                 rcvbuf, rcvcount, displs, rcvtype,
                 root, comm, ierr)
```

displs (INTEGER) entry i specifies the displacement (relative to sendbuf) from which to take the outgoing data to process i.



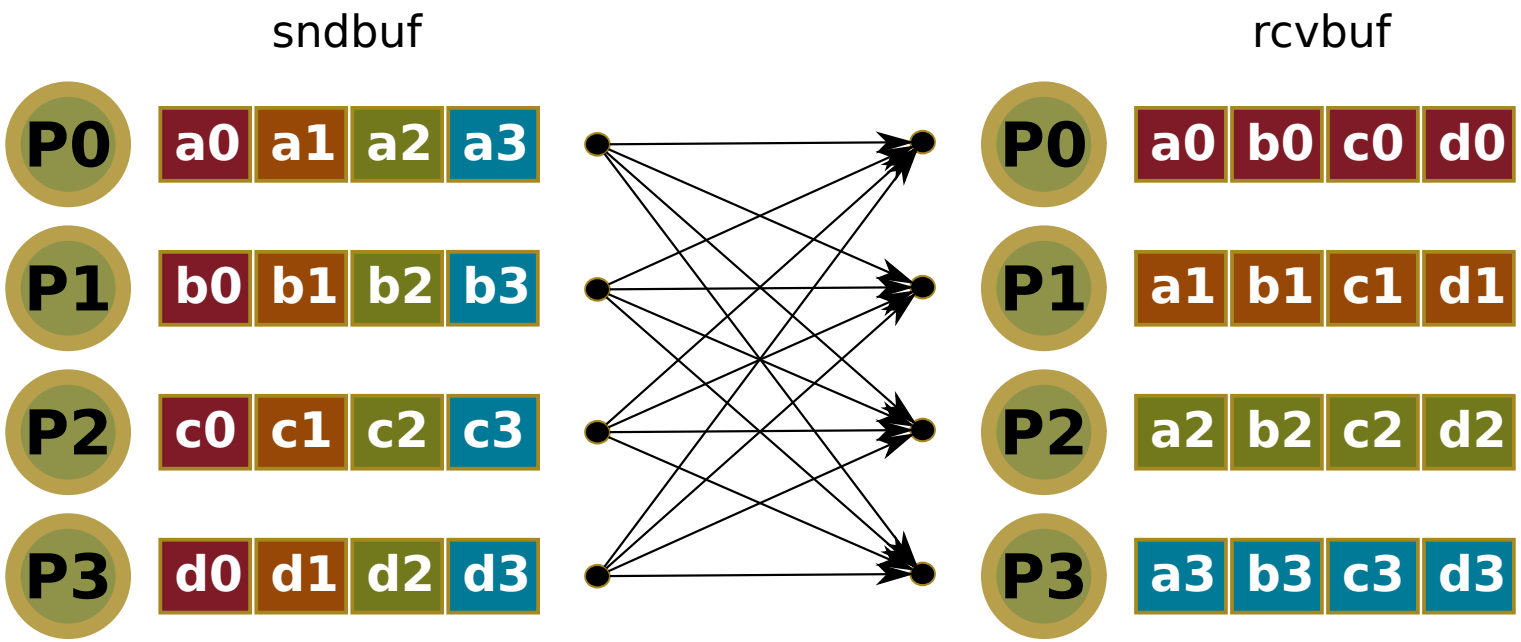
Gather with different buffers size



Global exchange: All to All

All-to-all communication: global exchange, all processes exchange their data. Useful for data transposition.

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



Reduction

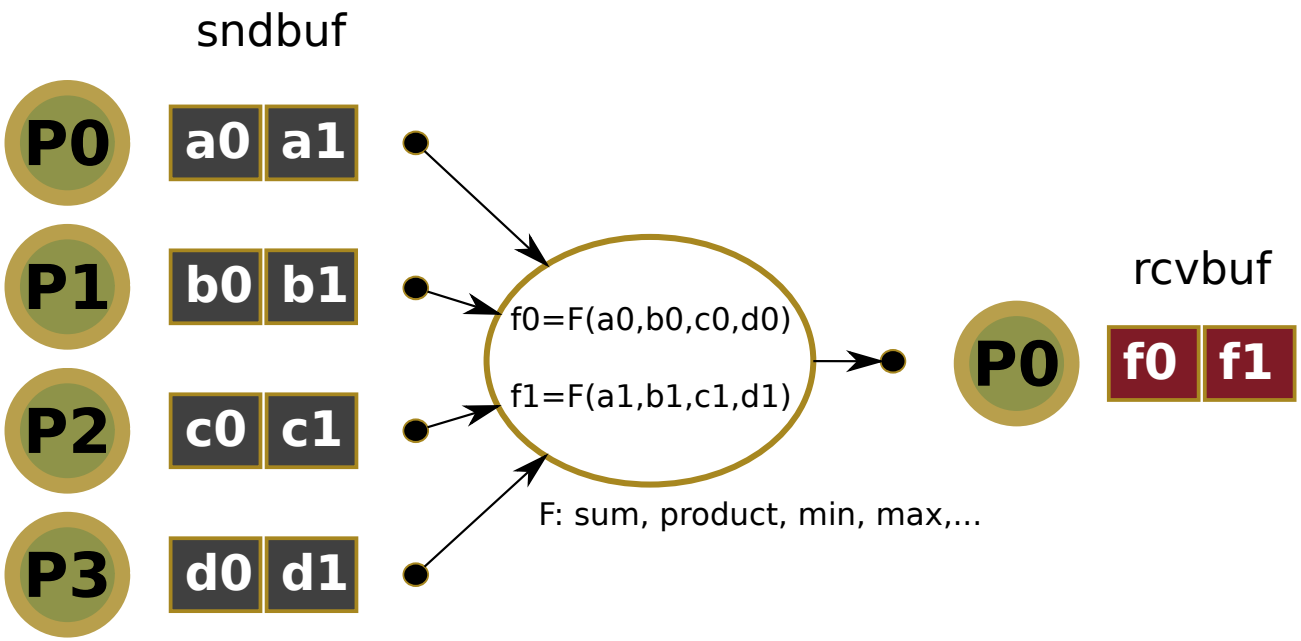
The reduction operation allows to:

- 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

Reduction

```
Fortran
CALL MPI_REDUCE(sndbuf, rcvbuf, count, type, op,
                root, comm, ierr)
```

op (INTEGER) parallel operation to perform



Reduction operators

MPI op	Operation
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

Global collective operations

The result of the one-to-all operation is known by all ranks at the end of the operation.

Fortran

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

CALL MPI_ALLGATHERV(sndbuf, sndcount, sndtype,
                    rcvbuf, rcvcount, displs, rcvtype,
                    comm, ierr)

CALL MPI_ALLREDUCE(sndbuf, rcvbuf, count, type, op,
                   comm, ierr)
```

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

Non-blocking collective operations

All collective operations have a non-blocking version.

Example:

```
Fortran
MPI_IBCAST(buf, count, type, root, comm, request, ierr)
```

Other functions:

```
Fortran
MPI_Iallgather, MPI_Iallgatherv, MPI_Iallreduce,
    MPI_Ialltoall
MPI_Ibarrier, MPI_Igather, MPI_Igatherv, MPI_Ireduce,
    MPI_Iscatter, MPI_Iscatterv
```

Other functions

- All-to-All operations, different buffer sizes and types:

```
MPI_AlltoAllv, MPI_AlltoAllw
```

- Neighbor operations, based on topology:

```
MPI_Neighbor_gather, MPI_Neighbor_alltoall
```

- Partial reduction:

```
MPI_Scan, MPI_Exscan
```

- Create your own operator:

```
MPI_Op_create, MPI_Op_free
```

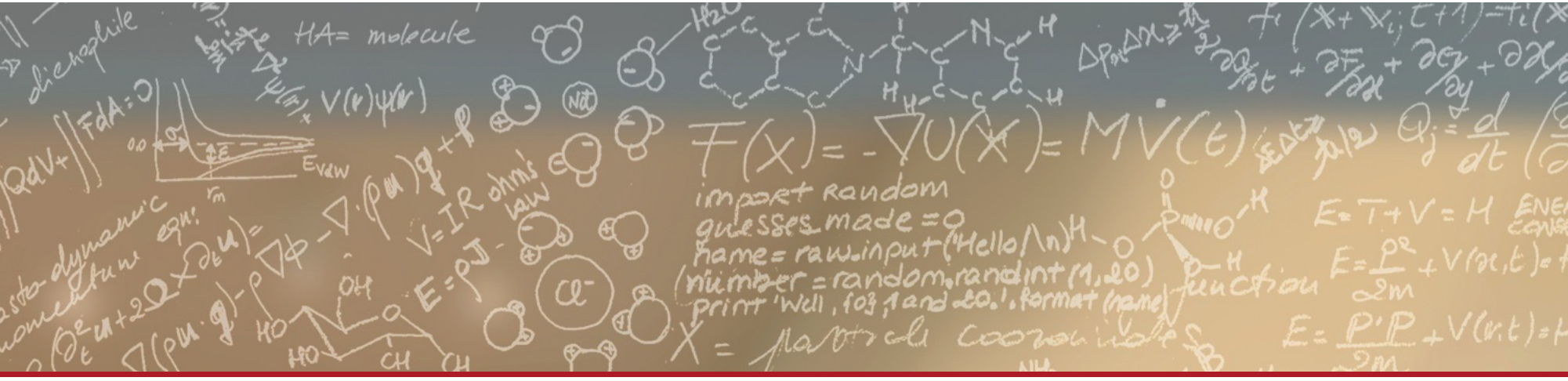
- Reduce+Scatter:

```
MPI_Reduce_scatter, MPI_Reduce_scatter_block
```

Practicals

Exercise: 03.MPI_Coll

1. Read from the terminal and broadcast the input
2. Initialise an array and scatter it
3. Read data from the terminal and reduce it
4. Read data from the terminal and reduce it if to all rank (allreduce)



Thank you for your attention.