# Parallel Programming

pauldj@aices.rwth-aachen.de



High Performance and Automatic Computing





### Anatomy of MPI\_Send and MPI\_Recv

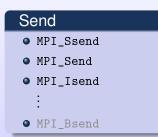
```
int MPI Send(
                                                                   ← "data"
    *buffer, count, datatype,
                                                               \leftarrow "envelope"
    destination, tag, communicator
);
int MPI Recv(
                                                                   ← "data"
    *buffer, count, datatype,
                                                               \leftarrow "envelope"
    source, tag, communicator,
    *status
);
                    message = data + envelope (+ info)
```

matching envelopes  $\rightarrow$  data transfer

Note: Meanining of count: send  $\neq$  recv

count in send = size of message vs. count in receive = size of buffer.

## Point-to-point communication



#### Receive

MPI\_Recv

MPI\_Irecv

### Send+Receive

- MPI\_Sendrecv
- MPI\_Sendrecv\_replace

### Send/Recv Modes

#### [Send] The stress is on the buffer sent: "When I can I safely overwrite it?"

- MPI\_Ssend: The program execution is blocked until a matching receive is posted. The buffer is usable as soon as the call completes.
- MPI\_Send: MPI attempts to copy the outgoing message onto a local (hidden) buffer. If possible, the execution continues and the send buffer is immediately usable, otherwise same as Ssend.
- MPI\_Isend: The execution continues Immediately. The send buffer should not be accessed until the MPI\_request allows it. To be used in conjunction with MPI\_Wait or MPI\_Test\*.
- [Recv] The stress is on the incoming buffer: "When I can I safely access it?"
  - MPI\_Recv: The program execution is blocked until a matching send is posted. The incoming buffer is usable as soon as the call completes.
  - MPI\_Irecv: The execution continues Immediately. The incoming buffer should not be accessed until the MPI\_request allows it. To be used in conjunction with MPI\_Wait or MPI\_Test\*.
- \*: See also MPI\_Waitany, MPI\_Waitall, MPI\_Waitsome, MPI\_Testany, MPI\_Testall, MPI\_Testsome.

## Recap: deadlock

- 2+ processes want to exchange data
- All processes start with a blocking send or a blocking receive Ssend, Send (in the worst case), Recv
   ⇒ BUG: deadlock
- Solution: BREAK SYMMETRY!

At the same time, careful not to serialize the code!

Approach: code, test and debug with Ssend; then replace with Send

- Other solutions?
  - Non-blocking send (Isend)
  - Non-blocking receive (Irecv)
  - Simultaneous send-receive (Sendrecv)