0.1 `mpi_recv`

- Can use `mpi_any_source` and `mpi_any_tag` for the source and tag arguments (respectively); should use them when possible, but only when needed
- `status(mpi_source) =` source of the message just received
- `status(mpi_tag) =` tag of the message just received
- `status(mpi_error) =` not useful
- `status(mpi_status_ignore)`; reduces overhead related to status tracking

1 Sending routines

1.1 Blocking sends

- `mpi_ssend`: synchronous send; hand-shake between sending and receiving processes, must complete before the processes proceed with execution;
- `mpi_rsend`: send results immediately, requires a receive to be waiting; if receive is not waiting, then behavior is undefined; shouldn't use
- `mpi_bsend`: buffered send; copies message to a buffer and execution continues; can help with load balancing; caveat is that the user must manage the buffer space; (see `mpi_buffer_attach` and `mpi_buffer_detach`)
- `mpi_send`: IBM made their implementation public a few years ago (if message was small enough and the number of MPI processes was not too large, message would be sent to a buffer on the receiving process; otherwise, used `mpi_ssend`); notice that issues may not appear until large problems are run; debugging with `mpi_ssend` is critical to avoiding such mistakes

1.2 Non-blocking routines

The intent of non-blocking routines is to enable communication and computation to overlap and to avoid deadlocking

1.2.1 Deadlocks (deadly embrace)

Basically, you're waiting for something to happen that is never going to happen. You can avoid them by

- be careful how sends and receives are ordered
- use `mpi_sendrecv` or `mpi_sendrecv_replace`
- use nonblocking routines (if you can)
- use buffered sends