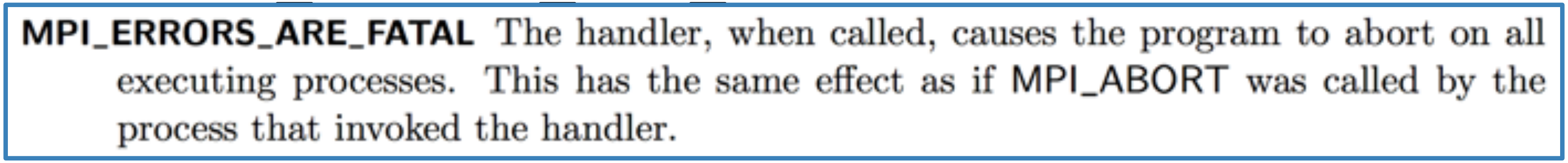
Ticket #324 Reading

Clarify `MPI_ERRORS_ARE_FATAL` scope of abort
https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/324

Fault Tolerance Working Group
June 3, 2015

Background

- Section 8.3 is imprecise about where MPI_ERRORS_ARE_FATAL is applied.



MPI_ERRORS_ARE_FATAL The handler, when called, causes the program to abort on all executing processes. This has the same effect as if MPI_ABORT was called by the process that invoked the handler.

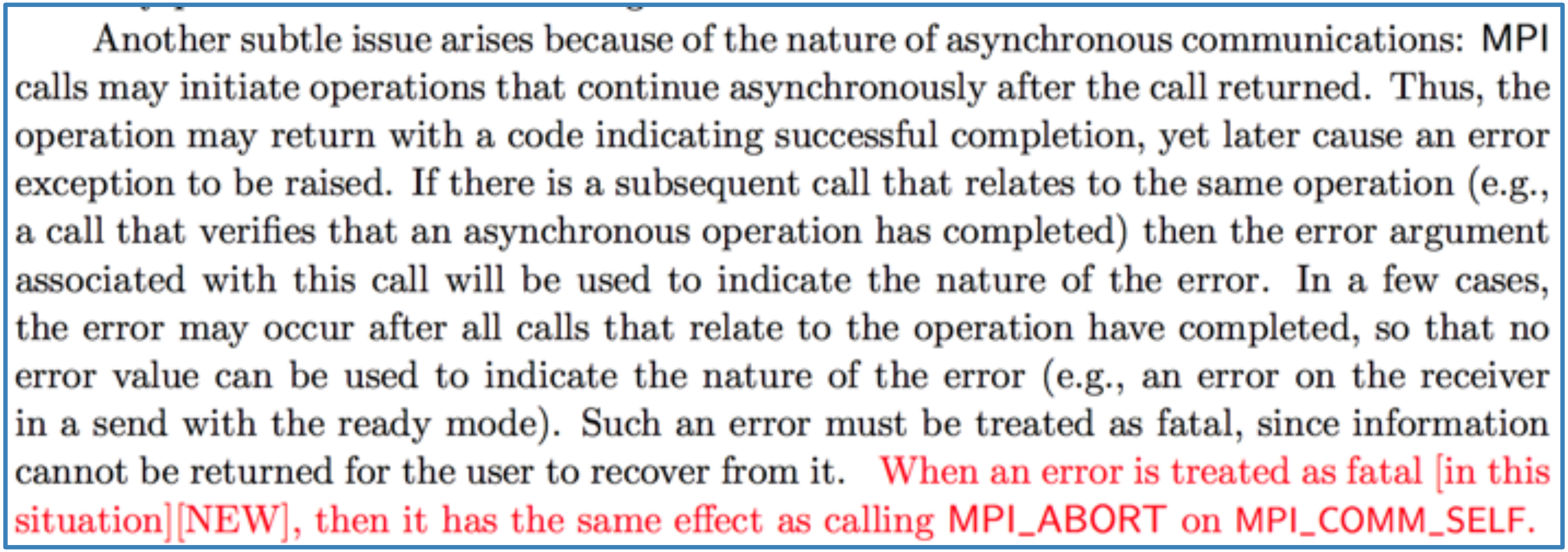
- The first and second sentences are contradictory (MPI_ABORT accepts a communicator argument)
- The second sentence is more permissive for FT implementations and good software engineering.

Motivation

- Allows the application to clean itself up after an error.
  - Flush state to disk, close files, etc.
- Step 0 for any FT solution
  - Could be part of ULFM, but this is more generic than that solution.
- Allows very basic FT implementations and applications
  - Without these changes, if **any** communicator uses `MPI_ERRORS_ARE_FATAL`, **all** use it.

Text Changes
Section 2.8

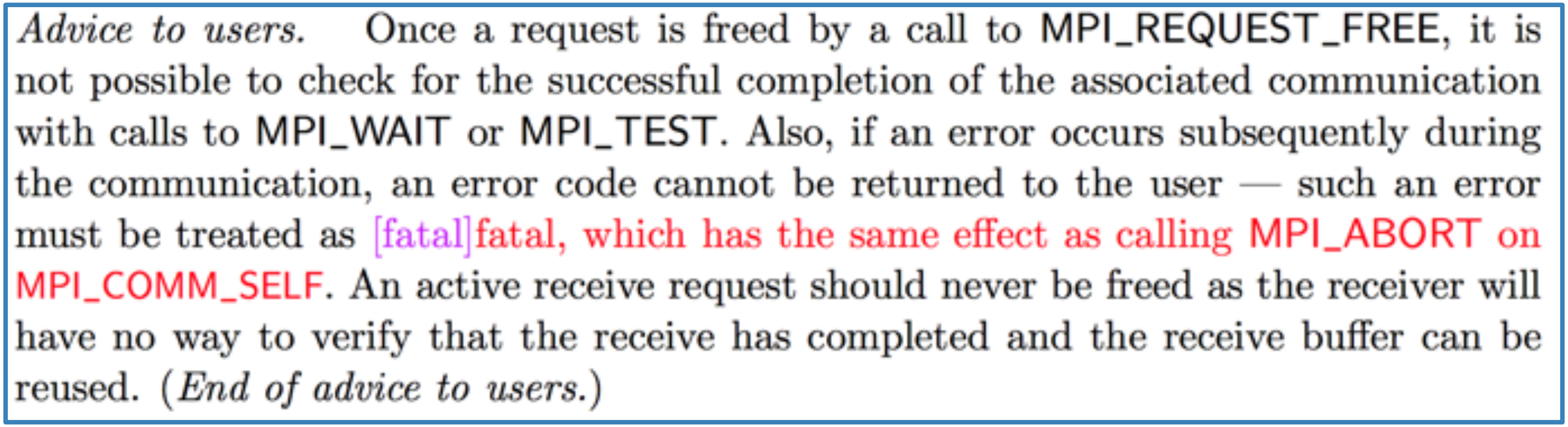
- Clarify that a generic fatal error is the same as calling `MPI_ABORT(MPI_COMM_SELF)`



Another subtle issue arises because of the nature of asynchronous communications: MPI calls may initiate operations that continue asynchronously after the call returned. Thus, the operation may return with a code indicating successful completion, yet later cause an error exception to be raised. If there is a subsequent call that relates to the same operation (e.g., a call that verifies that an asynchronous operation has completed) then the error argument associated with this call will be used to indicate the nature of the error. In a few cases, the error may occur after all calls that relate to the operation have completed, so that no error value can be used to indicate the nature of the error (e.g., an error on the receiver in a send with the ready mode). Such an error must be treated as fatal, since information cannot be returned for the user to recover from it. When an error is treated as fatal [in this situation][NEW], then it has the same effect as calling MPI_ABORT on MPI_COMM_SELF.

Text Changes
Section 3.7.3

- Definition of `MPI_REQUEST_FREE`
- Continue clarification from previous change

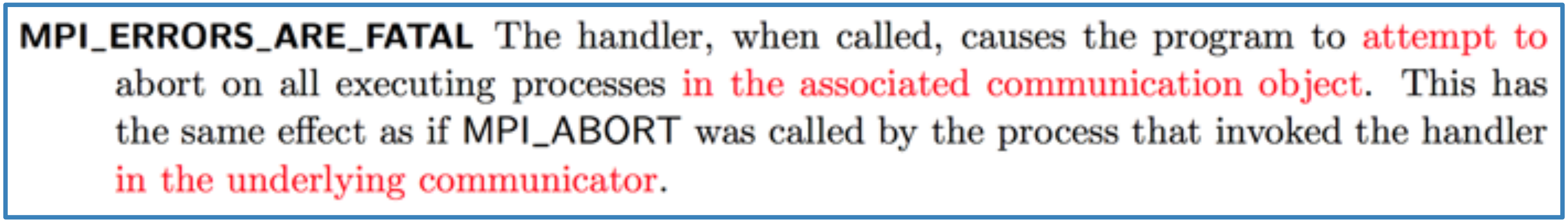


Advice to users. Once a request is freed by a call to MPI_REQUEST_FREE, it is not possible to check for the successful completion of the associated communication with calls to MPI_WAIT or MPI_TEST. Also, if an error occurs subsequently during the communication, an error code cannot be returned to the user — such an error must be treated as [fatal]fatal, which has the same effect as calling MPI_ABORT on MPI_COMM_SELF. An active receive request should never be freed as the receiver will have no way to verify that the receive has completed and the receive buffer can be reused. (*End of advice to users.*)

Text Changes
Section 8.3

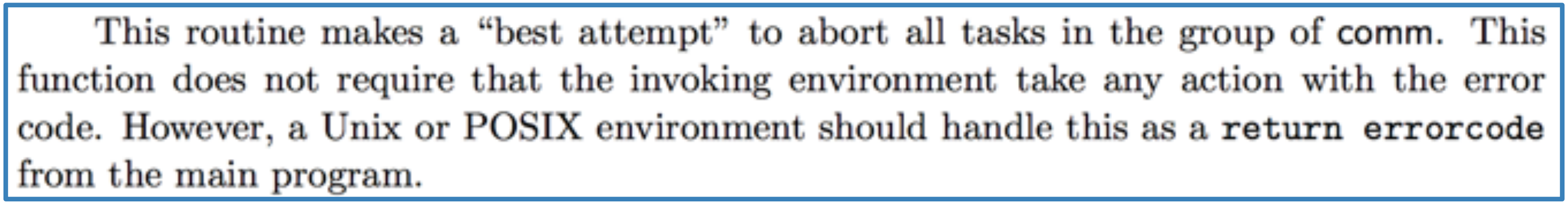
- Definition of MPI_ERRORS_ARE_FATAL

- Define the scope of abort for error handler to just the current communicator



MPI_ERRORS_ARE_FATAL The handler, when called, causes the program to attempt to abort on all executing processes in the associated communication object. This has the same effect as if MPI_ABORT was called by the process that invoked the handler in the underlying communicator.

Definition of MPI_ABORT

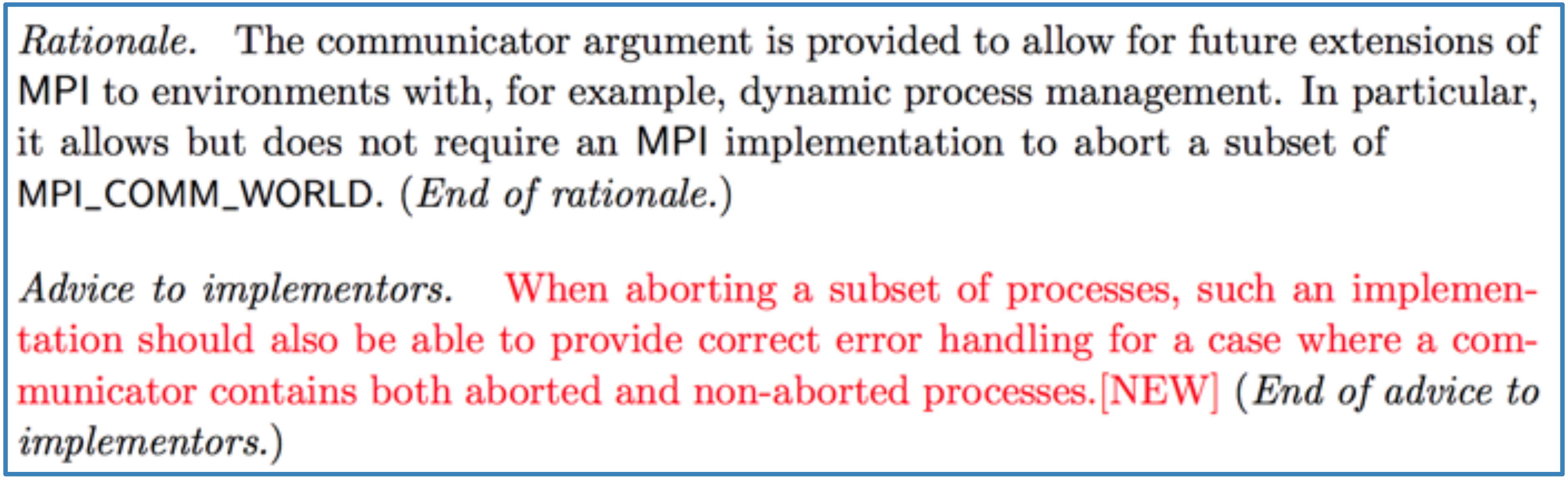


This routine makes a "best attempt" to abort all tasks in the group of comm. This function does not require that the invoking environment take any action with the error code. However, a Unix or POSIX environment should handle this as a return errorcode from the main program.

Text Changes
Section 8.7

- Definition of MPI_ABORT

- Add advice to say that implementations should provide a graceful way of dealing with aborted processes.

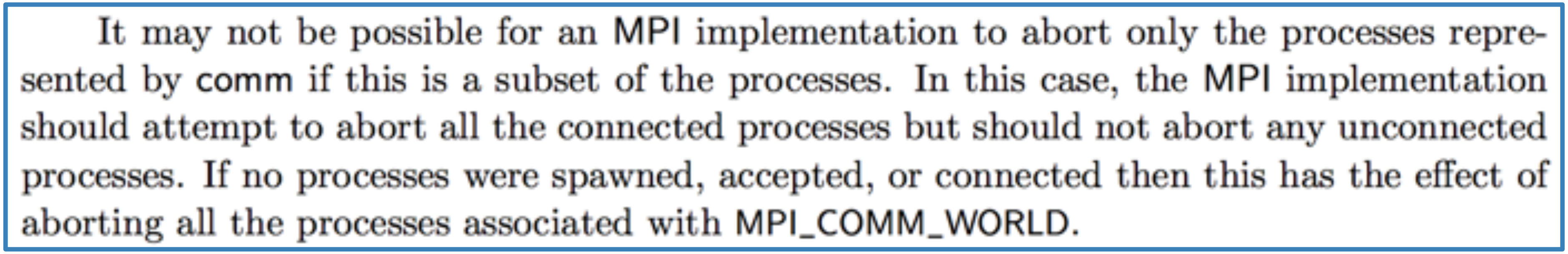


Rationale. The communicator argument is provided to allow for future extensions of MPI to environments with, for example, dynamic process management. In particular, it allows but does not require an MPI implementation to abort a subset of MPI_COMM_WORLD. (*End of rationale.*)

Advice to implementors. When aborting a subset of processes, such an implementation should also be able to provide correct error handling for a case where a communicator contains both aborted and non-aborted processes.[NEW] (*End of advice to implementors.*)

Previous Objections

- This prevents the application from handling an exception safely! (`MPI_SEND` kills apps)
  - If you want to be safe, set your error handler to `MPI_ERRORS_RETURN` on communicators that you want to keep alive.
  - Completing `MPI_SEND` does not imply that the message has been received (or that the receiver is alive).
- This has a large burden on implementations!
  - This is just as optional as FT has always been. Feel free to abort.



It may not be possible for an MPI implementation to abort only the processes represented by comm if this is a subset of the processes. In this case, the MPI implementation should attempt to abort all the connected processes but should not abort any unconnected processes. If no processes were spawned, accepted, or connected then this has the effect of aborting all the processes associated with MPI_COMM_WORLD.

Burden on Implementations

- No new implementation required
  - Implementations are still free to abort on errors
- Implementations that choose to do better now have clear instructions

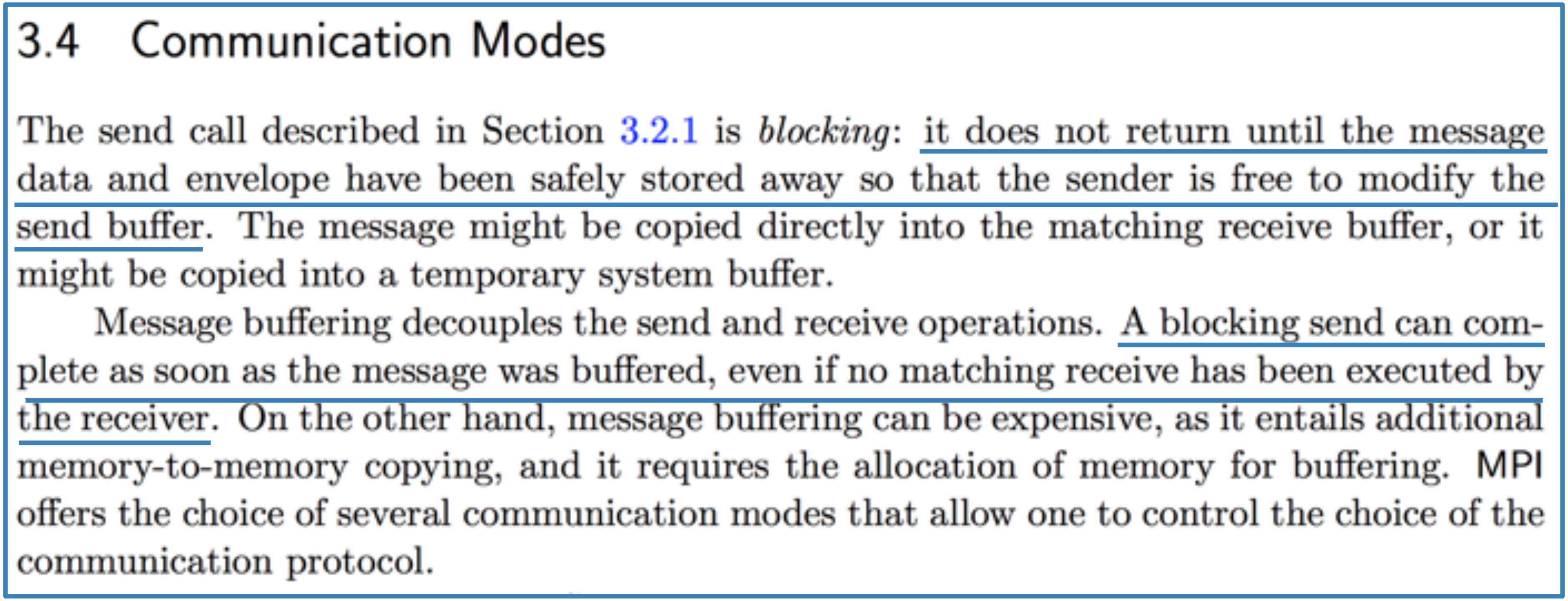
Implementation

- MPICH
  - Needs a patch to abort a subset instead of MPI_COMM_WORLD
    - In the review process
  - Other infrastructure has been present since v3.0
- Open MPI
  - Runtime doesn’t support continuing with a subset of processes

Backup Slides

Why MPI_Send can succeed when the receiver is dead

Returning from MPI_Send does not imply that the message is received, only buffered somewhere.



3.4 Communication Modes

The send call described in Section 3.2.1 is *blocking*: it does not return until the message data and envelope have been safely stored away so that the sender is free to modify the send buffer. The message might be copied directly into the matching receive buffer, or it might be copied into a temporary system buffer.

Message buffering decouples the send and receive operations. A blocking send can complete as soon as the message was buffered, even if no matching receive has been executed by the receiver. On the other hand, message buffering can be expensive, as it entails additional memory-to-memory copying, and it requires the allocation of memory for buffering. MPI offers the choice of several communication modes that allow one to control the choice of the communication protocol.