Here all proposed changes:

Adding at the end of MPI-3.1 Section 2.3, page 11, after line 22:

The words function, procedure, procedure call, and call are used as synonyms within this standard.

Substituting MPI-3.1 Section 2.4, page 11, lines 24-48 by

2.4 Semantic Terms

When discussing MPI procedures the following semantic terms are used.

An **MPI operation** is a set of one or more MPI procedures leading from a well-defined input state to a well-defined output state. An operation consists of four stages: initialization, starting, completion, and freeing:

Initialization hands over the argument list to the operation but not the content of the message data buffers. For an operation it may be specified that array arguments must not be changed until the operation is freed.

Starting hands over the content of the message data buffer to the associated operation.

Note that **initiation** refers to the combination of the initialization and starting stages.

Completion returns control of the content of the message data buffer and indicates that any output buffers have been updated.

Freeing returns control of the rest of the argument list.

MPI procedures can be blocking or nonblocking:

An **MPI procedure is blocking** if return from the procedure indicates the user is allowed to reuse resources specified in the call.

An **MPI procedure is nonblocking** if it returns before the user is allowed to reuse resources (such as buffers) specified in the call.

MPI operations can be blocking, nonblocking, or persistent:

For a **blocking operation**, all four stages are combined in a single blocking procedure call.

For a **nonblocking operation**, the initialization and starting stages are combined into a single nonblocking procedure call and the completion and freeing stages are done with a separate single procedure call, which can be blocking or nonblocking.

For a **persistent operation**, all four stages are done with separate procedure calls, each of which can be blocking or nonblocking.

In addition to the concept of blocking and nonblocking there is the orthogonal concept of locality:

An **MPI procedure is local** if it returns control to the calling MPI process based only on the state of the local MPI process that invoked it.

An **MPI procedure is non-local** if its return may require the execution of some MPI procedure on another MPI process. Such a procedure may require communication occurring with another MPI process.

Advice to users. Note that for communication-related procedures, in most cases nonblocking procedures are local and blocking procedures are non-local. Exceptions are noted where such procedures are defined.
In many cases, in the procedure name, the additional letter "I" as an abbreviation of the word "incomplete" marks nonblocking procedures and/or as an abbreviation of the word "immediately", it marks local procedures. *(End of advice to users.)*

Additionally as a third orthogonal aspect, a procedure can be either collective or not.

An **MPI procedure is collective** if all processes in a process group need to invoke the procedure.

Collective MPI operations are also available as blocking, nonblocking and persistent operations as defined above.

Collective initialization calls over the same process group must be executed in the same order by all members of the process group.
Blocking collective procedures and persistent collective initialization procedures may or may not be **synchronizing**, that is, may or may not return before all processes in the group have called the procedure.
Nonblocking collective initiation procedures and the start procedure of persistent collective operations are local and shall not be synchronizing.
In case of nonblocking or persistent collective operations, the completion stage may or may not finish before all processes in the group have started the operation.

Advice to users.
Calling any synchronising function when there is no possibility of concurrent calls at all other processes in the associated group is erroneous because it can cause deadlock.
Waiting for completion of any operation when there is no possibility that all other processes in the associated group will be able to start the operation is erroneous because it can cause deadlock.
(End of advice to users.)

When the words operation and procedure are used, usually MPI operation and MPI procedure are meant.
Annex A.2 summarizes the semantics of all communicating MPI routines.

For datatypes, the following terms are defined:
.....

MPI-3.1 Section 3.4, MPI_BSEND, page 37, after lines 36-43
A **buffered** mode send operation can be started whether or not a matching receive has been posted. It may complete before a matching receive is posted. However, unlike the standard send, this operation is local, and its completion does not depend on the occurrence of a matching receive. Thus, if a send is executed and no matching receive is posted, then MPI must buffer the outgoing message, so as to allow the send call to complete. An error will occur if there is insufficient buffer space. The amount of available buffer space is controlled by the user — see Section 3.6. Buffer allocation by the user may be required for the buffered mode to be effective.
the following sentence and advice are added
According to the definitions in Section 2.4, MPI_BSEND is a blocking procedure because the user can re-use all resources given as arguments, including the message data buffer. It is also a local procedure because it returns immediately without depending on the execution of any MPI procedure in any other MPI process.

Advice to users. This is one of the exceptions in which a blocking procedure is local. *(End of advice to users.)*

MPI-3.1 Section 3.7.2, Communication Initiation, page 48, lines 37-38 read
In addition a prefix of I (for immediate) indicates that the call is nonblocking.
but should read
In addition a prefix of I (for immediate and incomplete) indicates that the call is local and nonblocking.

MPI-3.1 Section 3.7.3, MPI_REQUEST_FREE, page 55, lines 16-18 read
Mark the request object for deallocation and set request to MPI_REQUEST_NULL. An ongoing communication that is associated with the request will be allowed to complete. The request will be deallocated only after its completion.
but should read
Mark the request object for deallocation and set request to MPI_REQUEST_NULL. [An] Ongoing communication that is associated with the request will be allowed to [complete] continue until it is finished. The request will be deallocated only after its [completion] associated communication has finished.

MPI-3.1 Section 3.8.1, MPI_IPROBE, page 65, after lines 20-22
If MPI_IPROBE returns flag = true, then the content of the status object can be sub- sequently accessed as described in Section 3.2.5 to find the source, tag and length of the probed message.
the following paragraph and advice are added
MPI_IPROBE is a local procedure since it does not depend on MPI calls in other MPI processes. According to the definitions in Section 2.4 with respect to the status output argument as resource, it is a blocking procedure although it returns immediately.

Advice to users. This is one of the exceptions in which a blocking procedure is local. *(End of advice to users.)*

MPI-3.1 Section 3.8.2, MPI_IMPROBE, page 68, after lines 30-31

In addition, it returns in message a handle to the matched message. Otherwise, the call returns flag = false, and leaves status and message undefined.

the following paragraph is added

MPI_IMPROBE is a local procedure. According to the definitions in Section 2.4 and in contrast to MPI_IPROBE, it is a nonblocking procedure because it is the initialization of a matched receive operation.

mpi32-report-ticket25-barcelona-vote-sep2018.pdf

MPI-3.NEXT #25 Section 5.13 Persistent Collective Operations, page 216, after lines 3-8

Initialization calls for MPI persistent collective operations are non-local and follow all the existing rules for collective operations, in particular ordering; programs that do not conform to these restrictions are erroneous. After initialization, all arrays associated with input arguments (such as arrays of counts, displacements, and datatypes in the vector versions of the collectives) must not be modifed until the corresponding persistent request is freed with MPI_REQUEST_FREE.

the following sentence and advice are added

According to the definitions in Section 2.4, the persistent collective initialization procedures are nonblocking. They are also non-local procedures because they may or may not return before they are called in all MPI processes of the process group.

Advice to users. This is one of the exceptions in which nonblocking procedures are non-local. *(End of advice to users.)*

MPI-3.1 Section 13.4.5 Split Collective Data Access Routines, page 528, after lines 20-24

- An implementation is free to implement any split collective data access routine using the corresponding blocking collective routine when either the begin call (e.g., MPI_FILE_READ_ALL_BEGIN) or the end call (e.g., MPI_FILE_READ_ALL_END) is issued. The begin and end calls are provided to allow the user and MPI implementation to optimize the collective Operation.

the following sentence and advice are added

According to the definitions in Section 2.4, the begin procedures are nonblocking. They are also non-local procedures because they may or may not return before they are called in all MPI processes of the process group.

Advice to users. This is one of the exceptions in which nonblocking procedures are non-local. *(End of advice to users.)*

Annex A.2 Summary of the Semantics of all Communicating MPI Routines

Stages: i=initialization, s=starting, c=completion, f=freeing

Blk: b=blocking, nb=nonblocking. Note that from a user's view point, this column is only a hint. Relevant is,

whether a routine is local or not and which resources are blocked until when. See next and last column.

Loc: l=local, nl=non-local

Bold: exceptions, e.g., b+l and nb+nl

Ixx: Using names with "I", Ix2 = "I" means immediate and incomplete, Im = "I" means only immediate

Op: part of operation type: b-op = blocking operation,
nb-op = nonblocking operation, p-op = persistent operation

Collective procedures:

- C = all processes of the group must call the procedure
- sq = in the same sequence
- S1 = blocking synchronization, S2 = start-complete-synchronization

Blocked resources: They are blocked after the call until the end of subsequent stage where this resource is not further mentioned.

Remarks:

1) Must not return before the corresponding MPI_receive operation is started.
2) In a correct MPI program, a call to MPI_(I)RSEND requires that the receiver has already started the corresponding receive. Under this asumption, the call is local.
3) Usually, MPI_Wait is non-local, but in this case it is local.
4) In case of a MPI_(I)BARRIER, the S1/S2 synchronization is required (instead of "may or may not").
5) In this case, MPI_REQUEST_FREE is nonlocal, see the Advice to implementors in Section 6.4.3
6) It also may not return until MPI_INIT has been called in the children.
7) Cached on the request handle.
8) One of the rare cases that a nonblocking call is non-local.
9) One shall not free or deallocate the buffer before the operation is freed, that is MPI_REQUEST_FREE returned.
10) For MPI_WAIT and MPI_TEST, see corresponding lines for a) MPI_BSEND, or b) MPI_IBCAST.

Procedure	**Stages**	**Blk**	**Loc**	**Ixx**	**Op**	**Collective**			**Blocked resources and remarks**
						C	**sq**	**S1/2**	
MPI_SEND	i-s-c-f	b	nl		b-op	-			
MPI_SSEND	i-s-c-f	b	nl		b-op	-			1)
MPI_RSEND	i-s-c-f	**b**	**l**		b-op	-			2)
MPI_BSEND	i-s-c-f	**b**	**l**		b-op	-			
MPI_RECV	i-s-c-f	b	nl		b-op	-			
MPI_ISEND, MPI_ISSEND	i-s----	nb	l	Ix2	nb-op	-			buffer
MPI_IRECV	i-s----	nb	l	Ix2	nb-op	-			buffer
corresponding MPI_Wait	----c-f		nl		nb-op	-			
corr. MPI_TEST returning flag=TRUE	----c-f		l		nb-op	-			
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op	-			buffer cached on req
MPI_IBSEND	i-s----	nb	l	Ix2	nb-op	-			buffer
MPI_IRSEND	i-s----	nb	l	Ix2	nb-op	-			buffer 2)
corresponding MPI_Wait	----c-f		**l**		nb-op	-			3)
corr. MPI_TEST returning flag=TRUE	----c-f		l		nb-op	-			
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op	-			buffer 7)
MPI_PROBE	i-s-c-f	b	nl		b-op	-			
MPI_IPROBE	i-s-c-f	**b**	**l**	**Im**	b-op	-			
MPI_RECV of a probed message	i-s-c-f	**b**	**l**		b-op	-			

MPI_IRECV of a probed message	i-s----	nb	l	Ix2	nb-op	-	buffer
corresponding MPI_Wait	----c-f		l		nb-op	-	3)
corr. MPI_TEST returning flag=TRUE	----c-f		l		nb-op	-	
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op	-	buffer 7)
MPI_MPROBE	i-s-c-f	**nb**	**nl**		b-op	-	the message itself **8)**
MPI_IMPROBE	i-s-c-f	nb	l	Ix2	b-op	-	the message itself
MPI_MRECV of a probed message	i-s-c-f	**b**	**l**		b-op	-	
MPI_IMRECV of a probed message	i-s----	nb	l	Ix2	nb-op	-	buffer
corresponding MPI_Wait	----c-f		l		nb-op	-	3)
corr. MPI_TEST returning flag=TRUE	----c-f		l		nb-op	-	
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op	-	buffer 7)
MPI_(-|S|R)SEND_INIT, MPI_RECV_INIT	i------	nb	l		p-op	-	buffer address
corresponding MPI_START, MPI_STARTALL	--s----	nb	l		p-op	-	buffer address+content 7)
corresponding MPI_Wait	----c--		nl		p-op	-	buffer address 7)
corr. MPI_TEST returning flag=TRUE	----c--		l		p-op	-	buffer address 7)
corr. MPI_TEST returning flag=FALSE	-------		l		p-op	-	buffer content+address 7)
corr. MPI_REQUEST_FREE (for inactive req-handle)	------f		l		p-op	-	

Procedure	**Stages**	**Blk**	**Loc**	**Ixx**	**Op**	**Collective** **C sq S1/2**	**Blocked resources and remarks**
MPI_BSEND_INIT	i------	nb	l		p-op	-	buffer address 9)
corresponding MPI_START, MPI_STARTALL	--s----	nb	l		p-op	-	buffer address+content 7)
corresponding MPI_Wait	----c--		l		p-op	-	buffer address 7,9)
corr. MPI_TEST returning flag=TRUE	----c--		l		p-op	-	buffer address 7,9)
corr. MPI_TEST returning flag=FALSE	-------		l		p-op	-	buffer address+content 7)
corr. MPI_REQUEST_FREE (for inactive req-handle)	------f		l		p-op	-	
MPI_CANCEL of nonblock./persistent pt-to-pt			l		p-op	-	
MPI_SENDRECV(_REPLACE)	i-s-c-f	b	nl		b-op	-	
MPI_BCAST and others	i-s-c-f	b	nl		b-op	C sq S1	4)
MPI_IBCAST and others	i-s----	nb	l	Ix2	nb-op	C sq	buffer 4)
MPI_IGATHER**V** and other **…V / …W**	i-s----	nb	l	Ix2	nb-op	C sq	buffer, array arguments
corresponding MPI_Wait	----c--		nl		nb-op	C S2	4)
corr. MPI_TEST returning flag=TRUE	----c--		l		nb-op	C S2	4)
corr. MPI_TEST returning flag=FALSE	-------		l		nb-op		buffer, array arguments 7)
MPI_BCAST_INIT and others	i------	**nb**	**nl**		p-op	C sq S1	buffer address **8)** 9)
MPI_GATHER**V**_INIT and other **…V/…W**_INIT	i-s----	**nb**	**nl**		p-op	C sq S1	buffer address, array arguments **8)** 9)
corresponding MPI_START, MPI_STARTALL	--s----	nb	l		p-op	C	buffer addr.+content, 4,7)
corresponding MPI_Wait	----c--		nl		p-op	C S2	buffer address and array arguments cached on the reqest handle, 4,7,9)
corr. MPI_TEST returning flag=TRUE	----c--		l		p-op	C S2	buf-addr&arr-args 4,7,9)
corr. MPI_TEST returning flag=FALSE	-------		l		p-op		buf addr+content&arr-args 7)
corr. MPI_REQUEST_FREE	------f		**nl**		p-op	C sq S1	5)

MPI_COMM_CREATE	i-s-c--	b nl	b-op	C sq S1	coll. over comm arg.
MPI_COMM_CREATE_GROUP	i-s-c--	b nl	b-op	C sq S1	coll. over group arg.
MPI_COMM_DUP, MPI_COMM_DUP_WITH_INFO, MPI_COMM_SPLIT, MPI_COMM_SPLIT_TYPE, MPI_CART_CREATE, MPI_GRAPH_CREATE, MPI_DIST_GRAPH_CREATE_ADJACENT, MPI_DIST_GRAPH_CREATE, MPI_CART_SUB: see MPI_COMM_CREATE					
MPI_INTERCOMM_CREATE, MPI_INTERCOMM_MERGE	i-s-c--	b nl	b-op	C sq S1	coll. over union of local & remote group
MPI_COMM_IDUP	i-s----	nb l Ix2	nb-op	C sq	communicator handle
corresponding MPI_Wait	----c--	nl	nb-op	C S2	
corr. MPI_TEST returning flag=TRUE	----c--	l	nb-op	C S2	
corr. MPI_TEST returning flag=FALSE	-------	l	nb-op		
MPI_COMM_FREE	------f	b nl	b-op	C sq S1	see, 6.4.3, Adv. to impl.
MPI_INIT, MPI_INIT_THREAD	i-s-c-f	b nl	b-op	C sq S1	collective over MPI_COMM_WORLD
MPI_FINALIZE	i-s-c-f	b nl	b-op	C sq S1	collective over all connected processes
MPI_COMM_SPAWN, …._MULTIPLE	i-s-c-f	b nl	b-op	C sq S1	collective over comm, 6)
MPI_COMM_ACCEPT, MPI_COMM_CONNECT	i-s-c-f	b nl	b-op	C sq S1	collective over comm
One-sided procedures					See corresponding chapter
MPI_FILE_READ/WRITE[_AT|SHARED], MPI_FILE_DELETE/SEEK/GET_VIEW	i-s-c-f	**b l**	b-op	-	
MPI_FILE_READ/WRITE[_AT]**_[ALL|ORDERED]**, MPI_FILE_OPEN/CLOSE/SEEK_SHARED, MPI_FILE_PREALLOCATE/SYNC, MPI_FILE_SET_VIEW/SIZE/INFO/ATOMICITY	i-s-c-f	b nl	b-op	C sq S1	
MPI_FILE_**I**READ/**I**WRITE[_AT|SHARED]	i-s----	nb l Ix2	nb-op	-	buffer 10a)
MPI_FILE_**I**READ/**I**WRITE[_AT]**_ALL**	i-s----	nb l Ix2	nb-op	C sq	buffer 10b)
MPI_FILE_READ/WRITE[_AT]_ALL|ORDERED**_BEGIN**	i-s----	**nb nl**	b-op	C sq S1	buffer **8)**
MPI_FILE_READ/WRITE[_AT]_ALL|ORDERED**_END**	----c-f	b nl	b-op	C sq S1	