

# Motivation and Scope of Changes to MPI Semantic Terms Section

## Semantic Terms Working Group

(latest changes: telcon 2020-05-26: now on slides 23, 23+25, 26;  
2020-05-31 modified changes of MPI wording on slides 23, (25, 26), 32, (36);  
organizational changes on slides: 19, 22-29, 32-36, 40, 50-51)

# Overview

- What is missing?
- Semantic Terms in MPI-3.1 – are they still correct?
  - Blocking / nonblocking
  - Collective
  - Local / non-local
- Current status of Sect. 2.4 Semantic Terms
- Solution → Issue #96, PR #116 (Changes v01-v10 → Part I.)
- RMA
  - Small corrections (v11-v13) → Part II.
  - Table of most RMA procedures in Annex A.2 (v14) → Part III.
  - Rational to formalize “user’s perspective” (v15) → Part IV.

# What is missing?

- Persistent:
  - Since MPI-4.0, now **two** Sections on persistent operations:
    - pt-to-pt and
    - collective
  - **Common definitions** should be in Semantic Terms
- Operations:
  - 1049 x the word “*operation*” in MPI-3.1 (and 1202x in MPI-4.x currently)
  - But **definition** of MPI operations **is missing**

# Motivation - original terms of reference

- Instruction: define “persistent” in context of “persistent collectives”
- There is/was no definition of the term “persistent” at all in MPI
  - The word is used without explanation in the body of the document.
- Persistent “fits” in the sequence {blocking, nonblocking, persistent}
- However, this is talking about MPI operations, not MPI procedures
  - Insight: there is no such thing as a “persistent procedure” in MPI.
- There is/was no definition of “MPI operation” at all in MPI!
- We need at least a definition of operation that is good enough to define “persistent operation”

# Defining “MPI operation”

- Need to differentiate **persistent operations** from **blocking** and **nonblocking** ones.
- Persistent MPI operations are expressed using 4 MPI procedures:  
`MPI_<thing>_init, MPI_Start[all], MPI_{Test|Wait}[all|some], MPI_Request_free`
- These can be seen as 4 state transitions between 2 operation states:
  - Initialisation (\*->inactive), starting (inactive->active),
  - Completion (active->inactive), freeing (inactive->\*)
- Nonblocking operation: 2 state transitions
  - Initialisation+starting & Completion+freeing
- Blocking operation: all together in one routine

# Semantic Terms in MPI-3.1 – are they still correct?

## 2.4 Semantic Terms

When discussing MPI procedures the following semantic terms are used.

**nonblocking** A procedure is nonblocking if it may return before the associated operation completes, and before the user is allowed to reuse resources (such as buffers) specified in the call. The word complete is used with respect to operations and any associated requests and/or communications. An operation completes when the user is allowed to reuse resources, and any output buffers have been updated.

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

**local** A procedure is local if completion of the procedure depends only on the local executing process.

**non-local** A procedure is non-local if completion of the operation may require the execution of some MPI procedure on another process. Such an operation may require communication occurring with another user process.

**collective** A procedure is collective if all processes in a process group need to invoke the procedure. A collective call may or may not be synchronizing. Collective calls over the same communicator must be executed in the same order by all members of the process group.

# Semantic Terms in MPI-3.1 – are they still correct?

**nonblocking** A procedure is nonblocking if it may return before the associated **operation completes**, and **before the user is allowed to reuse resources (such as buffers) specified in the call**. The word complete is used with respect to operations and any associated requests and/or communications. An operation **completes** when the user is allowed to reuse resources, and any output buffers have been updated.

Additionally, the term “**operation**” is undefined → coming later

Examples:

- MPI\_File\_read\_all\_begin:
  - This procedure is **nonblocking** because you must not reuse the buffer upon return.
  - This procedure is non-local. It may **block** (*in the normal English sense of the word*) until all processes of the process group have called this procedure.
  - To name a procedure “nonblocking” although it may block – is this a good idea? In other words, MPI-3.1 allows that a routine may both **block** (English meaning) and is **nonblocking** (based on the MPI definition) .
- MPI\_Bcast\_init (coming with MPI-4):
  - This procedure is **nonblocking** because you must not free the buffer address upon return.
  - This procedure is non-local because it may **block** until all processes of the process group have called this procedure.

# Semantic Terms in MPI-3.1 – are they still correct?

**nonblocking** A procedure is nonblocking if it may return before the associated operation completes, and before the user is allowed to reuse resources (such as buffers) specified in the call. The word complete is used with respect to operations and any associated requests and/or communications. An operation **completes** when the user is allowed to reuse resources, and any output buffers have been updated.

- Result:
- The MPI definition of nonblocking is **broken**, because in conflict with the normal use of English blocking **in a dangerous way**:  
A user may program deadlocks because he/she may overs that the procedure is non-local.  
There are 23 MPI nonblocking (MPI sense) procedures that are allowed to block (English sense)

- How to resolve:
- Introduce the new semantic term “**incomplete**” based on the definition above.
  - Define nonblocking as incomplete AND local
  - and only for operation-related procedures



# Semantic Terms in MPI-3.1 – are they still correct?

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

Problem: • When we fix “nonblocking”, then we have also to fix “blocking”

Important: • **Non**..... should be identical to logically **not** .....

How to resolve: • Define “blocking” also only for operation-related procedures  
• And just: An MPI procedure is **blocking** if it is not nonblocking 😊

# Semantic Terms in MPI-3.1 – are they still correct?

**collective** A procedure is collective if all processes in a process group need to invoke the procedure. A collective call may or may not be synchronizing. Collective calls over the same communicator must be executed in the same order by all members of the process group.

Question: These are two different concepts:

- A procedure is collective if all processes in a process group need to invoke the procedure.
- A collective call may or may not be synchronizing.

Some routines are

- Collective and they are allowed to synchronize but need not to synchronize (MPI\_Bcast, MPI\_Bcast\_init)
- Collective and they are **not** allowed to synchronize (MPI\_Ibcast)
- Collective and must synchronize (MPI\_Barrier)

Overseen when introducing the nonblocking collectives in MPI-3.0

This exception is clearly defined in the routine definition of MPI\_Barrier

→ Unclear / weakly defined definition and therefore needs to be clarified.

# Semantic Terms in MPI-3.1 – are they still correct?

**collective** A procedure is collective if all processes in a process group need to invoke the procedure. **A collective call may or may not be synchronizing.** Collective calls over the same communicator must be executed in the same order by all members of the process group.

- Result:
- The definition fits well to blocking collective operations, but since MPI-1.1, we now have many chapters with collective operations and different types of collective procedures:
    - Always: **must be called by all processes** of the group
    - Initialization procedures of collective operations must be called in the **same sequence**.
    - Initiation procedures for nonblocking collective operations and the starting of persistent collectives are **local**,
    - whereas all others, especially the blocking collectives and the persistent collective initialization procedures are allowed to **synchronize**.

- How to resolve:
- Rewriting this definition that it fits to all collective procedures.

# Semantic Terms in MPI-3.1 – are they still correct?

**local** A procedure is local if completion of the procedure depends only on the local executing process.

**non-local** A procedure is non-local if completion of the operation may require the execution of some MPI procedure on another process. Such an operation may require communication occurring with another user process.

Issue: • Local / non-local is related to progress

# Semantic Terms in MPI-3.1 – are they still correct?

**local** A procedure is local if completion of the procedure depends only on the local executing process.

**non-local** A procedure is non-local if completion of the operation may require the execution of some MPI procedure on another process. Such an operation may require communication occurring with another user process.

- Important:
- Local / non-local is related to progress
  - → any change in the definition must not change the understanding of progress

- Clarification:
- Let's name this definition of "local" as "*strong local*".
  - Let's check whether this meaning of "local" is really used in MPI?

# Semantic Terms in MPI-3.1 – are they still correct?

**local** A procedure is local if completion of the procedure depends only on the local executing process.

Problems:

- Let's look at MPI\_Cancel and MPI\_Test\_cancelled of a nonblocking MPI\_Issend?

- MPI-3.1, section 8.7, Example 8.9 on pages 358+359: "The program is correct.":

```
Process 0                Process 1
-----                -----
MPI_Issend(dest=1);      MPI_Finalize();
MPI_Cancel();
MPI_Wait(..., &status);
MPI_Test_cancelled(&status, &flag);
MPI_Finalize();
```

I added this local inquiry call to show the problem

- Both, MPI\_Cancel and MPI\_Wait must be local by definition of MPI\_Cancel.
- MPI\_Cancel may require communication to check, whether the message is already received on the other process, i.e., whether flag == true or false must be returned:
  - As stated in MPI-1.1 to MPI-3.1, in the advice to implementors for MPI\_Test\_cancelled (MPI-3.1 page73 lines 21-23):  
"Note that, while communication may be needed to implement MPI\_CANCEL, this is still a local operation, since its completion does not depend on the code executed by other processes."
  - That this MPI\_Cancel is able to communicate, MPI-1.2 already mentioned for a similar example (MPI-2 page 24, lines 12-14): "An implementation may need to delay the return from MPI\_FINALIZE until all potential future message cancellations have been processed."

Result:

- Therefore, "local" was seen as weak local since MPI-1.1 → The definition of local is **broken**.

# Semantic Terms in MPI-3.1 – are they still correct?

**local** A procedure is local if completion of the procedure depends only on the local executing process.

**non-local** A procedure is non-local if completion of the operation may require the execution of some MPI procedure on another process. Such an operation may require communication occurring with another user process.

A weak-local definition would be needed for MPI\_Cancel

Problems:

- Is non-local the contrary of local?
- The contrary of local would be something like
  - **non-local** A procedure is non-local if completion of the operation may require the execution of some (MPI or non MPI) procedure on another process. Such an operation may require communication occurring with another user process.
- Gap between non-local and local → broken
- What means “of some MPI procedure on another process”?
  - Is it “may require the execution of some specific semantically-related MPI procedure”, then non-local is identical to **contrary of weak local**.
  - Or is it “may require the execution of some (specific or unspecific) MPI procedure”, then non-local is nearby to the **contrary of strong local**.
- This means, this definition is not clear enough → and therefore **double broken**.

This would be needed to be consistent with MPI\_Cancel

# Semantic Terms in MPI-3.1 – are they still correct?

**local** A procedure is local if completion of the procedure depends only on the local executing process.

**non-local** A procedure is non-local if completion of the operation may require the execution of some MPI procedure on another process. Such an operation may require communication occurring with another user process.

- Result:
- Local should be defined as weak local
  - Non-local should be defined as the contrary of weak local
  - MPI-1.1 to MPI-3.1 ever wanted to have local defined as “weak local” that implementations can implement all MPI procedures as flexible and efficient as possible.

- How to resolve:
- We take the non-local definition and add “specific semantically-related”
  - We define local as not non-local

- Remark:
- The terms complete/incomplete and blocking/nonblocking are only useful for operation-related MPI procedures,
  - Whereas local/non-local can be used for all MPI procedures.



# Current status of Sect. 2.4 Semantic Terms

**operation**

*- Missing -*

**persistent**

*- Missing -*

**nonblocking** A procedure is nonblocking if it may return before the associated operation completes, and before the user is allowed to reuse resources (such as buffers) specified in the call. The word complete is used with respect to operations and any associated requests and/or communications. An operation completes when the user is allowed to reuse resources, and any output buffers have been updated.

*- Broken -*

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

*- Broken -*

**local** A procedure is local if completion of the procedure depends only on the local executing process.

*- Broken -*

**non-local** A procedure is non-local if completion of the operation may require the execution of some MPI procedure on another process. Such an operation may require communication occurring with another user process.

*- Broken -*

**collective** A procedure is collective if all processes in a process group need to invoke the procedure. A collective call may or may not be synchronizing. Collective calls over the same communicator must be executed in the same order by all members of the process group.

*- Unclear -*

# Solution → Issue #96, PR #116

**Semantic Terms** has now three subsections:

- **MPI operations**
  - Operations consist of 4 stages: initialization, starting, completion, freeing stages
  - 3 forms: blocking, nonblocking and **persistent operations**
  - Collective / noncollective operations
- **MPI procedures**
  - **Non-local / local** (defined for all MPI procedures)
  - An MPI operation is implemented as a set of one or more MPI procedures.
  - An MPI operation-related procedure implements at least part of a stage of an MPI operation.
  - Properties of operation-related MPI procedures:
    - Initialization / Starting / initiation / **completing / incomplete** / freeing procedure
    - **Nonblocking / blocking procedure**
    - **Collective procedure**
- **MPI Datatypes**
  - (existing text is unchanged)

Some additional comments on exceptions within the chapters

New Annex A.2, showing

- the set of procedures that form an operation
- The properties of the involved procedures

## **Important:**

- We do not change the definition of any MPI procedure.
- The semantic terms should be consistent to the rest of the MPI standard.

# Status (May 20, 2020)

- Had reading: Version from Feb. 2, 2020 (2 weeks before Portland meeting (Feb.18-21, 2020))

- A set of additional changes:

- No-no-votes requested for June 29-July 1, 2020 meeting (Munich → online)
- Based on the discussions during the Portland meeting and afterwards

In chap-terms/ (v00= version 2020-02-02 of terms-2.tex)

- v00-->v01: Formatting (3 small changes with \ldots)
- v01-->v02: Better or more precise English wording (12 independent small changes)
- v02-->v03: removal of unnecessary notes, e.g., implications (3 such small removals)
- v03-->v04: moving non-local and local definitions before all operation-related procedure terms (1 move)
- v04-->v05: Stating more clearly that the other terms are only defined for operation related procedures (1 small change)
- v05-->v06: Define 'blocking' as 'not nonblocking' (1 small change for logical clarity)
- v06-->v07: More precise definition of non-local that the rationale is no more needed (adding in the definition of non-local the words “specific semantically-related” and removing the therefore no longer needed rationale and advice to users)
- v07-->v08: Define 'Incomplete' as 'not completing' (1 small change for logical clarity)

In chap-applang/ (v00= version 2020-02-02 of appLang-Const.tex)

- v00-->v09: Minor changes in the Appendix A.2 legend and footnotes (part 1)
- v09-->v10: Minor changes in the Appendix A.2 legend and footnotes (part 2)

PART I.

- For first vote: Version from May 20, 2020

# Lets look at pdf on Issue #96 and the small changes Part I. (v01-v10) on PR 116

- pdf: <https://github.com/mpi-forum/mpi-issues/files/4708009/mpi32-report-semantic-terms-2020-05-20-annotated.pdf>
- Issue: <https://github.com/mpi-forum/mpi-issues/issues/96>
  - Here, you can find also these slides:
  - EuroMPI2019-SemanticTerms-fromPuri-2020-04-28+RMA-rab-2020-05-31.pdf
- Pull request: <https://github.com/mpi-forum/mpi-standard/pull/116>
- Any open questions?

# What is still missing?

- RMA (Current status Feb 2 / May 20, 2020):

- Text in Semantic Terms, advice to users:

Nonblocking procedures:

- incomplete and local: MPI\_ISEND, MPI\_IRecv, MPI\_IBCAST, MPI\_PUT, MPI\_GET, MPI\_ACCUMULATE, MPI\_IMPROBE, MPI\_SEND\_INIT, MPI\_RECV\_INIT, ...

- Text in Annex A.2

Procedure	Stages	Cpl	Loc	Blk	Op	Collective		Blocked resources and remarks
						C	S/W	
MPI_PUT, MPI_GET, MPI_ACCUMULATE	i-s----	ic	l	nb	nb-op	-		buffer 14) 21)
Other one-sided procedures								21)

14) Nonblocking procedure without an l prefix.

21) In some cases, more than one MPI procedure may be needed to implement one stage of an MPI one-sided operation. For details on the semantics of one-sided operations, see Chapter 11.

- Partitioned Communication

- Currently no investigation / the proposer of the partitioned communication may check whether their text still fits to the modified Semantic Terms, especially the use of “nonblocking”

# Problems with RMA

- Do we have any problem with the change
  - from MPI-1 – 3: nonblocking := incomplete
  - to MPI-4: nonblocking := incomplete AND local

We first  
address  
these  
questions

- Is there any wrong usage of nonblocking as
  - operation vs. procedure is nonblocking
  - procedure does not block until ...
  - procedure is local

These questions should  
be also checked by the  
chapter committee of  
“Partitioned  
communication”

- Which table entries would be correct for the RMA synchronization procedures?
- Are the RMA communication procedures really nonblocking?

# RMA small wording corrections Part II.a

MPI-3.1 Section 11.3 Communication Calls, page 417, lines 10-23 should read

MPI supports the following RMA communication calls: MPI\_PUT and MPI\_RPUT transfer data from the caller memory (origin) to the target memory; MPI\_GET and MPI\_RGET transfer data from the target memory to the caller memory; MPI\_ACCUMULATE and MPI\_RACCUMULATE update locations in the target memory, e.g., by adding to these Locations values sent from the caller memory; MPI\_GET\_ACCUMULATE, MPI\_RGET\_ACCUMULATE, and MPI\_FETCH\_AND\_OP perform atomic read-modify-write and return the data before the accumulate operation; and MPI\_COMPARE\_AND\_SWAP performs a remote atomic compare and swap operation. These operations are **nonblocking**.

**Additionally, these particular procedures are nonblocking:** the call initiates the transfer, but the transfer may continue after the call returns. The transfer is completed, at the origin or both the origin and the target, when a subsequent synchronization call is issued by the caller on the involved window object. These synchronization calls are described in Section 11.5.

MPI-3.1 Section 11.5.2, page 442, lines 38-42 about Example 11.4 should read

One can also have implementations where the call to MPI\_WIN\_START ~~is nonblocking~~ **may not block**, but the call to MPI\_PUT blocks until the matching call to MPI\_WIN\_POST occurs; or implementations where the first two calls ~~are nonblocking~~ **may not block**, but the call to MPI\_WIN\_COMPLETE blocks until the call to MPI\_WIN\_POST occurred;

(similar to the unchanged text in MPI-3.1 Section 11.5.3, page 448, lines 22-25 about Example 11.5:

The call to MPI\_WIN\_LOCK may block until an exclusive lock on the window is acquired; or, the first two calls **may not block**, while MPI\_WIN\_UNLOCK blocks until a lock is acquired – the update of the target window is then postponed until the call to MPI\_WIN\_UNLOCK occurs. )

# RMA small wording corrections Part II.a, continued

MPI-3.1 Section 11.5.2, page 443, lines 32-33 on MPI\_WIN\_TEST should read

This is the ~~nonblocking~~ local version of MPI\_WIN\_WAIT. It returns flag = true if all accesses to the local window by the group to which it was exposed by the corresponding.

MPI-3.1 Section 11.7.3 Progress, page 462, lines 37-38 about Example 11.6 should read

The post calls ~~are nonblocking~~ do not block, and should complete.

MPI-4 Issue #96 Annex A.2, amendment to Footnote 21 on RMA procedures read

21) In some cases, more than one MPI procedure may be needed to implement one stage of an MPI one-sided operation.

The locality of the RMA procedures is analyzed from the user's perspective, i.e., independent of the choices to implementors about weak synchronization as described in Sections 11.5, 11.5.2 (on Example 11.4), and 11.5.3 (on Example 11.5).

For details on the semantics of one-sided operations, see Chapter 11.

- These changes are under the commits

In chap-applang/

- v10-->v11: Semantic terms / RMA small wording corrections in A.2 Footnote 21 (Part II.a)

In chap-one-sided/ (v00= version 2020-02-02 of one-sided-2.tex)

- v00-->v12: Semantic terms / RMA small wording corrections (Part II.a)



# RMA small wording corrections Part II.b

MPI-3.1 Section 11.5.2, page 442, lines 27-46 about Example 11.4 should read

Consider the sequence of calls in the example below.

Example 11.4

```
MPI_Win_start(group, flag, win);  
MPI_Put(..., win);  
MPI_Win_complete(win);
```

The call to MPI\_WIN\_COMPLETE does not return until the put call has completed at the origin; and the target window will be accessed by the put operation only after the call to MPI\_WIN\_START has matched a call to MPI\_WIN\_POST by the target process.

*Advice to implementors.* This still leaves much choice to implementors. The call to MPI\_WIN\_START can block until the matching call to MPI\_WIN\_POST occurs at all target processes. One can also have implementations where the call to MPI\_WIN\_START may not block, but the call to MPI\_PUT blocks until the matching call to MPI\_WIN\_POST occurs; or implementations where the first two calls may not block, but the call to MPI\_WIN\_COMPLETE blocks until the call to MPI\_WIN\_POST occurred; or even implementations where all three calls can complete before any target process has called MPI\_WIN\_POST – the data put must be buffered, in this last case, so as to allow the put to complete at the origin ahead of its completion at the target. However, once the call to MPI\_WIN\_POST is issued, the sequence above must complete, without further dependencies. (*End of advice to implementors.*)

*Advice to users.* In order to ensure a portable deadlock free program, a user must assume that MPI\_WIN\_START may block until the corresponding MPI\_WIN\_POST has been called. (*End of advice to users.*)

This existing part from MPI-3.1 now defined as advice to implementors.

New advice to users to make clear that MPI\_WIN\_START has to be assumed as non-local for portable application programming.

# RMA small wording corrections Part II.b, continued

MPI-3.1 Section 11.5.3 *Lock*, page 448, lines 14-28 about Example 11.5 should read

Consider the sequence of calls in the example below.

Example 11.5

```
MPI_Win_lock(MPI_LOCK_EXCLUSIVE, rank, assert, win);  
MPI_Put(..., rank, ..., win);  
MPI_Win_unlock(rank, win);
```

The call to MPI\_WIN\_UNLOCK will not return until the put transfer has completed at the origin and at the target.

*Advice to implementors.* This still leaves much freedom to implementors. The call to MPI\_WIN\_LOCK may block until an exclusive lock on the window is acquired; or, the first two calls may not block, while MPI\_WIN\_UNLOCK blocks until a lock is acquired – the update of the target window is then postponed until the call to MPI\_WIN\_UNLOCK occurs. However, if the call to MPI\_WIN\_LOCK is used to lock a local window, then the call must block until the lock is acquired, since the lock may protect local load/store accesses to the window issued after the lock call returns. (*End of advice to implementors.*)

*Advice to users.* In order to ensure a portable deadlock free program, a user must assume that MPI\_Win\_lock may block until an exclusive lock on the window is acquired. (*End of advice to users.*)

- These changes are under the commits

In chap-one-sided/

– v12-->v13: Semantic terms / RMA small wording corrections (Part II.b)

# Status (May 31, 2020)

- A set of additional changes:
  - No-no-votes requested for June 29-July 1, 2020 meeting (Munich → online)
  - Based on the discussions after the Portland meeting:
    - In chap-applang/
      - v10-->v11: Semantic terms / RMA small wording corrections in A.2 Footnote 21 (Part II.a)  
(1 small change)
    - In chap-one-sided/ (v00= version 2020-02-02 of one-sided-2.tex)
      - v00-->v12: Semantic terms / RMA small wording corrections (Part II.a)  
(5 small changes)
    - In chap-one-sided/
      - v12-->v13: Semantic terms / RMA small wording corrections (Part II.b)  
(4 small changes)
    - In chap-changes/ (v00= version 2020-02-02 of terms-2.tex)
      - v00-->v14: Semantic terms / RMA small wording corrections (Part II.a+b)  
(4 small changes)
- For first vote: Version from May 31, 2020

PART II.

# Lets look at pdf on Issue #96 and the small changes on PR 116

- pdf: <https://github.com/mpi-forum/mpi-issues/files/4708015/mpi32-report-semantic-terms-2020-05-31-annotated.pdf>
- Issue: <https://github.com/mpi-forum/mpi-issues/issues/96>
  - Here, you can find also these slides:
  - EuroMPI2019-SemanticTerms-fromPuri-2020-04-28+RMA-rab-2020-05-31.pdf
- Pull request: <https://github.com/mpi-forum/mpi-standard/pull/116>
- Any open questions?

# Problems with RMA

- Do we have any problem with the change
  - from MPI-1 – 3: nonblocking := incomplete
  - to MPI-4: nonblocking := incomplete AND local
- Is there any wrong usage of nonblocking as
  - operation vs. procedure is nonblocking
  - procedure does not block until ...
  - procedure is local

Second  
question

- Which table entries would be correct for the RMA synchronization procedures?
- Are the RMA communication procedures really nonblocking?

# Additional changes to Issue 96 to more completely handle RMA in Annex A.2 (part 3)

MPI-4 Issue #96 Annex A.2 2<sup>nd</sup> table on RMA procedures reads

Procedure	Stages	Cpl	Loc	Blk	Op	Collective			Blocked resources and remarks
						C	sq	S/W	
MPI_PUT, MPI_GET, MPI_ACCUMULATE	i-s----	ic	l	nb	nb-op	-			buffer 14) 21)
Other one-sided procedures									21)

but should read

Procedure	Stages	Cpl	Loc	Blk	Op	Collective			Blocked resources and remarks
						C	sq	S/W	
On an origin process with Fence Synchronization									
MPI_WIN_FENCE	i-----		nl			C	sq	W1	
MPI_PUT, MPI_GET, MPI_ACCUMULATE, ...	i-s----	ic	l	nb	nb-op	-			buffer 14) 21)
MPI_WIN_FENCE	----c-f	c+f	l			C	sq	W1	
On a target process									
MPI_WIN_CREATE/_ALLOCATE/..._SHARED	i-----		nl			C	sq	W1	buffer address
MPI_WIN_FENCE	--s----	ic	l			C	sq	W1	buffer address+content
MPI_WIN_FENCE	----c--	c	nl			C	sq	W1	buffer address
MPI_WIN_FREE	-----f	f	nl			C	sq	W1	

Continuation on next slide

# Additional changes to Issue 96 to more completely handle RMA in Annex A.2 (part 3, continued)

and

Procedure	Stages	Cpl	Loc	Blk	Op	Collective			Blocked resources and remarks
						C	sq	S/W	
On an origin process with General Active Target Synchronization									
MPI_WIN_START	i-----		nl			-			21)
MPI_PUT, MPI_GET, MPI_ACCUMULATE, ...	i-s----	ic	l	nb	nb-op	-			buffer 14) 21)
MPI_WIN_COMPLETE	----c-f	c+f	l			-			21)
On a target process									
MPI_WIN_CREATE/_ALLOCATE/..._SHARED	i-----		nl			C	sq	W1	buffer address
MPI_WIN_POST	--s----	ic	l			-			buffer address+content
MPI_WIN_WAIT	----c--	c	nl			-			buffer address
MPI_WIN_FREE	-----f	f	nl			C	sq	W1	
On an origin process with Lock/Unlock Synchronization									
MPI_WIN_LOCK and others	i-----		nl			-			21)
MPI_PUT, MPI_GET, MPI_ACCUMULATE, ...	i-s----	ic	l	nb	nb-op	-			buffer 14) 21)
MPI_WIN_UNLOCK and others	----c-f	c+f	l			-			21)
On a target process									
MPI_WIN_CREATE/_ALLOCATE/..._SHARED	i-----		nl			C	sq	W1	buffer address
MPI_WIN_FREE	-----f	f	nl			C	sq	W1	

Continuation on next slide

# Additional changes to Issue 96 to more completely handle RMA in Annex A.2 (part 3, continued)

and

Procedure	Stages	Cpl	Loc	Blk	Op	Collective		Blocked resources and remarks
						C sq	S/W	
On an origin process with Lock/Unlock Synchronization								
MPI_WIN_LOCK and others	i-----		nl			-		21)
MPI_RPUT, MPI_RGET, MPI_RACCUMULATE, ...	i-s----	ic	l	nb	nb-op	-		buffer 14) 21)
MPI_WAIT	----c-f	c+f	l			-		21)
MPI_WIN_UNLOCK and others	-----		l			-		21)
On a target process								
MPI_WIN_CREATE/_ALLOCATE/..._SHARED	i-----		nl			C sq	W1	buffer address
MPI_WIN_FREE	-----f	f	nl			C sq	W1	



# Status (May 31, 2020)

- A set of additional changes:
  - No-no-votes requested for June 29-July 1, 2020 meeting (Munich → online)
  - Based on the discussions after the Portland meeting:
    - In chap-applang/
      - v11-->v15: Semantic terms / RMA tables in A.2 (Part III.)  
(RMA removed from 2<sup>nd</sup> table in A.2; 3<sup>rd</sup> table added for RMA)
    - In figures/
      - Modified MPI-semantics-appendix\_for-pptx.xlsx (containing all tables)
      - Modified MPI-semantics-appendix.pptx (now including also img4)
      - Modified MPI-semantics-appendix\_img3.png / .eps / .pdf (without the 2-rows entry for MPI\_PUT, ...)
      - New MPI-semantics-appendix\_img4.png / .eps / .pdf (new table for RMA)
- For first vote: Version from May 31, 2020

PART III.

# Lets look at pdf on Issue #96 and the small changes on PR 116

- pdf: <https://github.com/mpi-forum/mpi-issues/files/4708015/mpi32-report-semantic-terms-2020-05-31-annotated.pdf>
- Issue: <https://github.com/mpi-forum/mpi-issues/issues/96>
  - Here, you can find also these slides:
  - EuroMPI2019-SemanticTerms-fromPuri-2020-04-28+RMA-rab-2020-05-31.pdf
- Pull request: <https://github.com/mpi-forum/mpi-standard/pull/116>
- Any open questions?

# Problems with RMA

- Do we have any problem with the change
  - from MPI-1 – 3: nonblocking := incomplete
  - to MPI-4: nonblocking := incomplete AND local
- Is there any wrong usage of nonblocking as
  - operation vs. procedure is nonblocking
  - procedure does not block until ...
  - procedure is local
- Which table entries would be correct for the RMA synchronization procedures?

Third  
question

- Are the RMA communication procedures really nonblocking?

# Recap of RMA in MPI-3.1

## 11.3 Communication Calls (page 417, lines 10-23)

MPI supports the following RMA communication calls: **MPI\_PUT** and **MPI\_RPUT** transfer data from the caller memory (origin) to the target memory; **MPI\_GET** and **MPI\_RGET** transfer data from the target memory to the caller memory; **MPI\_ACCUMULATE** and **MPI\_RACCUMULATE** update locations in the target memory, e.g., by adding to these Locations values sent from the caller memory; **MPI\_GET\_ACCUMULATE**, **MPI\_RGET\_ACCUMULATE**, and **MPI\_FETCH\_AND\_OP** perform atomic read-modify-write and return the data before the accumulate operation; and **MPI\_COMPARE\_AND\_SWAP** performs a remote atomic compare and swap operation. **These operations are nonblocking.**

**Additionally, these particular procedures are *nonblocking*: the call initiates the transfer, but the transfer may continue after the call returns.** The transfer is completed, at the origin or both the origin and the target, when a subsequent synchronization call is issued by the caller on the involved window object. These synchronization calls are described in Section 11.5.

That is what we expected:  
MPI\_PUT, MPI\_GET, MPI\_ACCUMULATE, ..., they are *nonblocking* procedures according to MPI-3.1 definition of nonblocking.  
Are they also *local*?

# Recap of RMA in MPI-3.1

## 11.7.3 Progress (page 462, lines 16-20)

One-sided communication has the same progress requirements as point-to-point communication: once a communication is enabled it is guaranteed to complete. **RMA calls must have local semantics, except when required for synchronization with other RMA calls.**

Yes, they are also *local*!

What does this exception mean?  
Can only RMA synchronization calls be  
*non-local*?

# Recap of RMA in MPI-3.1

## 11.5.2 General Active Target Synchronization (page 441, line 28-32, 47 - page 442, line 4)

`MPI_WIN_START(group, assert, win)`

Starts an RMA access epoch for win. RMA calls issued on win during this epoch must access only windows at processes in group. Each process in group must issue a matching call to `MPI_WIN_POST`. RMA accesses to each target window will be delayed, if necessary, until the target process executed the matching call to `MPI_WIN_POST`. **`MPI_WIN_START` is allowed to block until the corresponding `MPI_WIN_POST` calls are executed, but is not required to.**



Okay, `MPI_WIN_START` is *non-local*!


# Recap of RMA in MPI-3.1

## 11.5.2 General Active Target Synchronization (page 442, lines 10, 21-26)

`MPI_WIN_COMPLETE(win)`

Completes an RMA access epoch on `win` started by a call to `MPI_WIN_START`. All RMA communication calls issued on `win` during this epoch will have completed at the origin when the call returns.

`MPI_WIN_COMPLETE` enforces completion of preceding RMA calls at the origin, but not at the target. A put or accumulate call may not have completed at the target when it has completed at the origin.



`MPI_WIN_COMPLETE` has no defined  
right to block.  
Expectation: It is *local* !?!?

# Recap of RMA in MPI-3.1

## 11.5.2 General Active Target Synchronization (page 442, lines 27-46)

Consider the sequence of calls in the example below.

Example 11.4

```
MPI_Win_start(group, flag, win);  
MPI_Put(..., win);  
MPI_Win_complete(win);
```

The call to `MPI_WIN_COMPLETE` does not return until the put call has completed at the origin; and the target window will be accessed by the put operation only after the call to `MPI_WIN_START` has matched a call to `MPI_WIN_POST` by the target process.

*Advice to implementors.*

This still leaves much choice to implementors. The call to `MPI_WIN_START` can block until the matching call to `MPI_WIN_POST` occurs at all target processes. One can also have implementations where the call to `MPI_WIN_START` may not block, but the call to `MPI_PUT` blocks until the matching call to `MPI_WIN_POST` occurs; or implementations where the first two calls may not block, but the call to `MPI_WIN_COMPLETE` blocks until the call to `MPI_WIN_POST` occurred; or even implementations where all three calls can complete before any target process has called `MPI_WIN_POST` – the data put must be buffered, in this last case, so as to allow the put to complete at the origin ahead of its completion at the target. However, once the call to `MPI_WIN_POST` is issued, the sequence above must complete, without further dependencies. (*End of advice to implementors.*)

*Advice to users.* In order to ensure a portable deadlock free program, a user must assume that `MPI_WIN_START` may block until the corresponding `MPI_WIN_POST` has been called. (*End of advice to users.*)

**Wow!!!!**  
“MPI\_PUT is allowed to block until ...”  
**Is MPI\_PUT still a local procedure???**

**From the users view, these sentences are not relevant:**  
**The user has already to expect that MPI\_WIN\_START has blocked until ...**

**This means:**  
**Only the BLUE sentence is relevant for the user’s perspective for writing deadlock-free MPI applications ...**

**... as also mentioned in the new advice to users.**



# Recap of RMA in MPI-3.1

Often overseen in many discussions!

## 1.8 Who Should Use This Standard? (page 5, lines 1-9)

This standard is intended for use by all those who want to write portable message-passing programs in Fortran and C (and access the C bindings from C++). This includes individual application programmers, developers of software designed to run on parallel machines, and creators of environments and tools. In order to be attractive to this wide audience, the standard must provide a simple, easy-to-use interface for the basic user while not semantically precluding the high-performance message-passing operations available on advanced machines.

Let's formalize our definition of non-local  
From the "application programmers"  
viewpoint

# How to resolve the problem that MPI\_Put should be local and is allowed to block

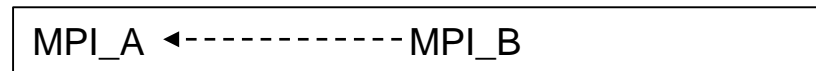
Semantic Terms, **definition of non-local**:

**Non-local procedure** An MPI procedure is non-local if returning may require the execution of some specific semantically-related MPI procedure on another MPI process during the execution of the MPI procedure.

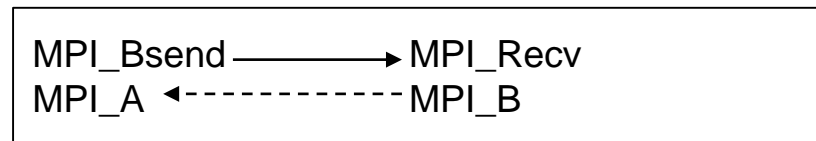
How to decide whether an MPI procedure is non-local, especially, if an MPI implementation is allowed to postpone a “may block until ...” to a later MPI procedure call?

**In this case, we may formalize our definition of non-local:**

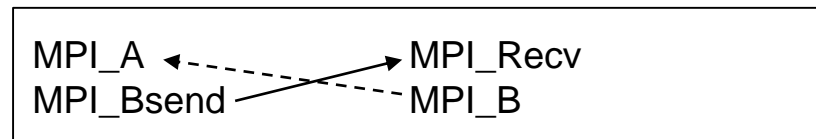
An MPI procedure MPI\_A is non-local, if there exists a correct (i.e. also deadlock-free) MPI program with a call to MPI\_A on one process and if there exists a call to an MPI procedure MPI\_B on another process



so that if we add a test message as in the following figure



then the application is still correct (i.e. also deadlock-free), whereas the following modification may cause a deadlock:

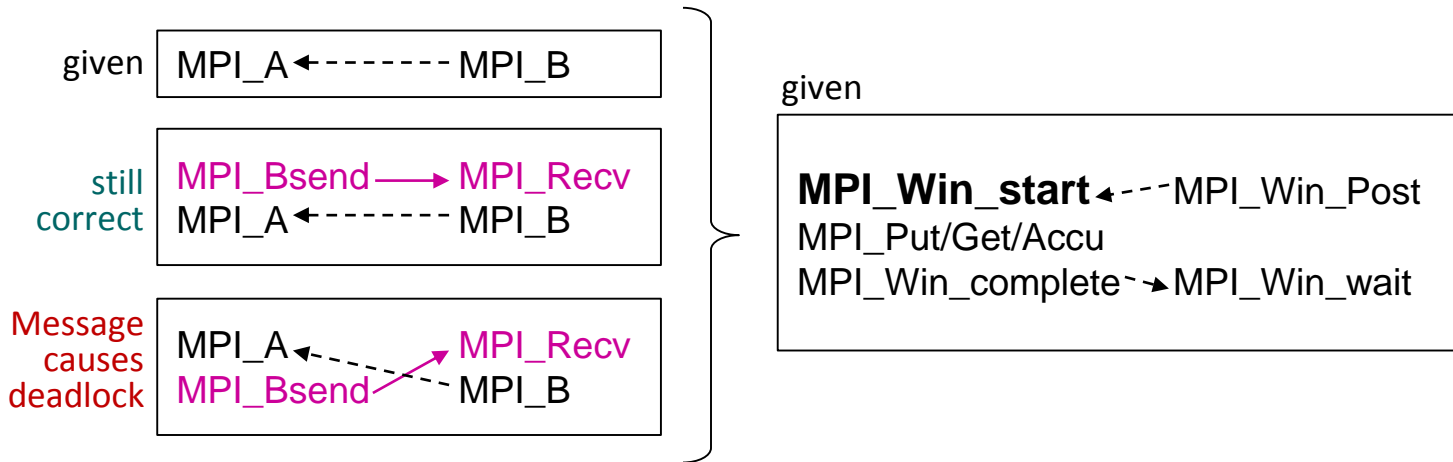


If such a correct MPI program and such MPI\_B exist, then MPI\_B is such a “specific semantically-related MPI procedure on another MPI process” in the definition of *non-local*.

If no such correct MPI program with any such MPI\_B exist, then MPI\_A is *local*.

# Let's apply the test scheme to MPI\_Win\_start:

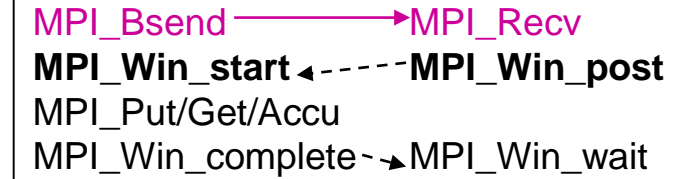
Is A=MPI\_Win\_start non-local?



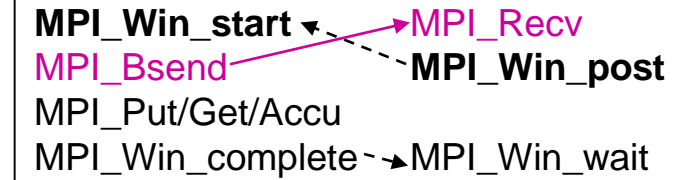
**Final result:**  
MPI\_Win\_start is non-local because its return may requires the call of MPI\_Win\_post in another process.

1) Testing with B=MPI\_Win\_post:

Still correct? → YES



Message causes deadlock? → YES

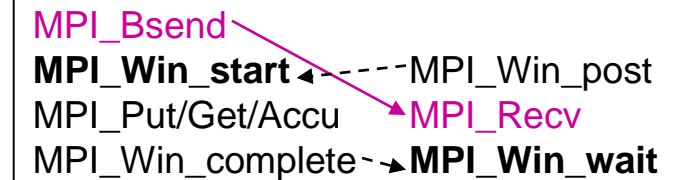


Result: MPI\_Win\_start is non-local

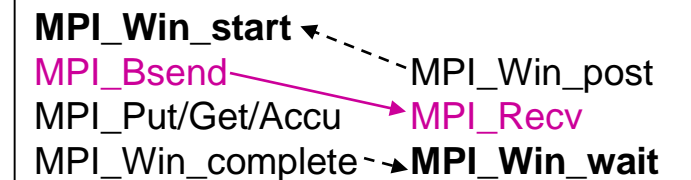
2) Testing with B=MPI\_Win\_wait:

(not needed, if one B is already found)

Still correct? → YES



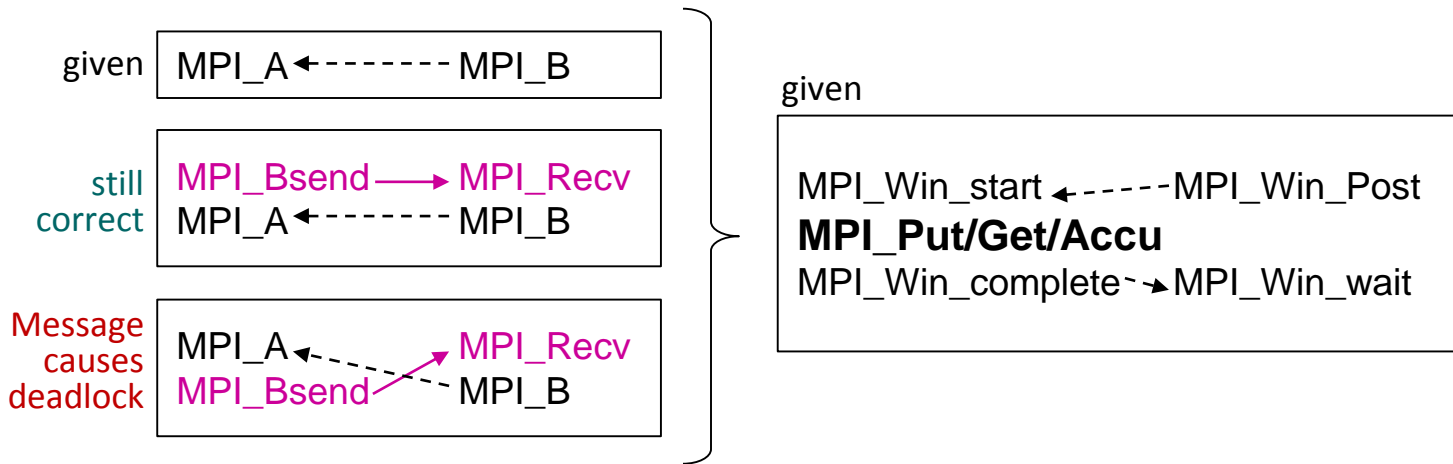
Message causes deadlock? → NO (failed)



Result: No answer from this test

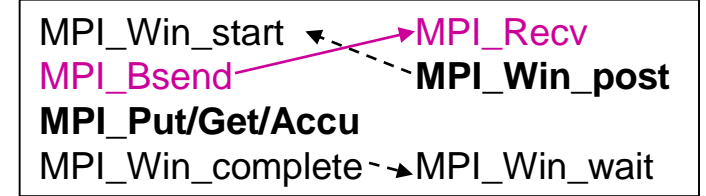
# Let's apply the test scheme to MPI\_Put/Get/Accumulate:

Is A=MPI\_Put/Get/Accu non-local?



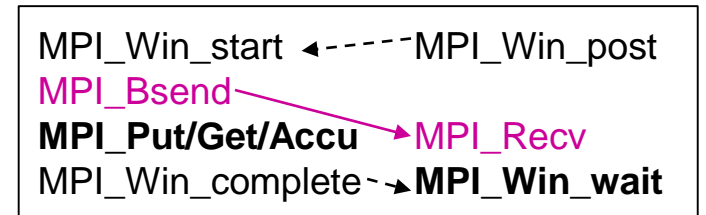
**Final result:** MPI\_Put, Get, Accumulate are local because there isn't such MPI\_B to detect non-locality.

1) Testing with B=MPI\_Win\_post:  
Still correct? → NO (because it may deadlock)

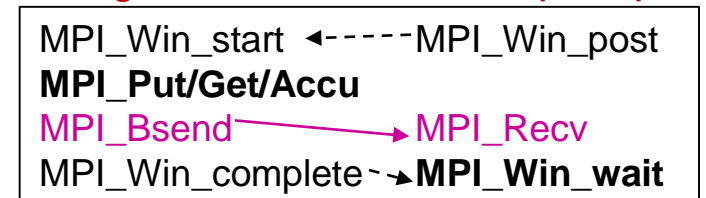


Result: No answer from this test

2) Testing with B=MPI\_Win\_wait:  
(not needed, if one B is already found)  
Still correct? → YES



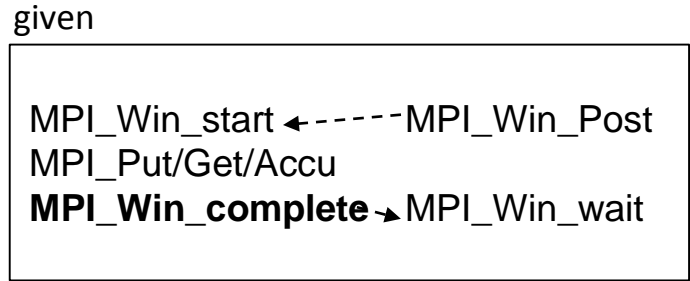
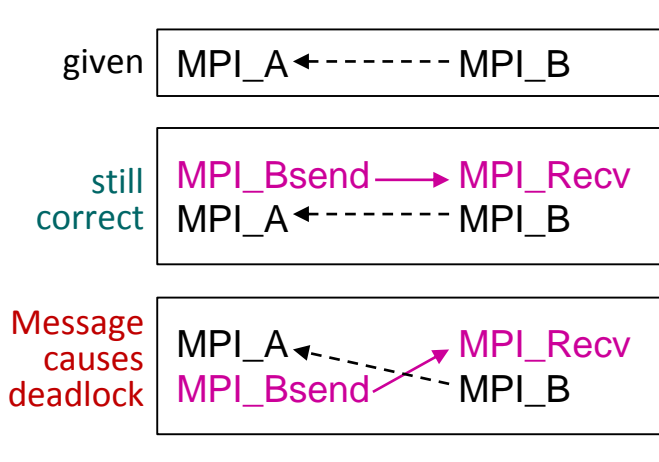
Message causes deadlock? → NO (failed)



Result: No answer from this test

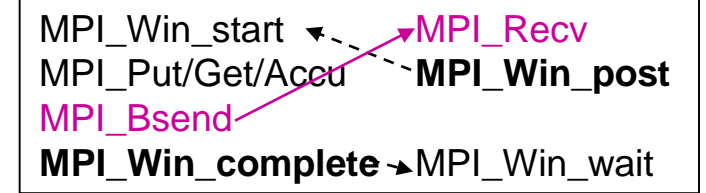
# Let's apply the test scheme to **MPI\_Win\_complete**:

Is **A=MPI\_Win\_complete** non-local?



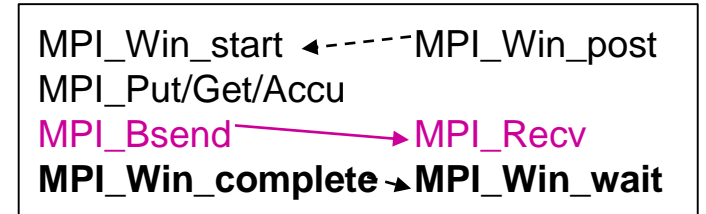
**Final result:** **MPI\_Win\_complete** is **local** because there isn't such MPI\_B to detect non-locality.

1) Testing with **B=MPI\_Win\_post**:  
Still correct? → NO (because it may deadlock)

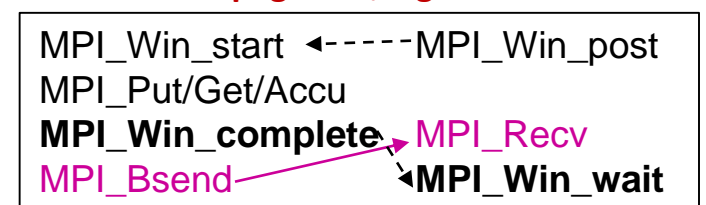


Result: No answer from this test

2) Testing with **B=MPI\_Win\_wait**:  
(not needed, if one B is already found)  
Still correct? → YES



Message causes deadlock? → NO (failed)  
See MPI-3.1 page 463, Fig. 11.8



Result: No answer from this test

# Recap of RMA in MPI-3.1

## 11.5.2 General Active Target Synchronization (page 443, lines 1, 17-19, 22, 33-38)

`MPI_WIN_POST(group, assert, win)`

Starts an RMA exposure epoch for the local window associated with `win`. Only processes in `group` should access the window with RMA calls on `win` during this epoch. Each process in `group` must issue a matching call to `MPI_WIN_START`. **`MPI_WIN_POST` does not block.**

It is *local*

`MPI_WIN_WAIT(win)`

Completes an RMA exposure epoch started by a call to `MPI_WIN_POST` on `win`. This call matches calls to `MPI_WIN_COMPLETE(win)` issued by each of the origin processes that were granted access to the window during this epoch. **The call to `MPI_WIN_WAIT` will block until all matching calls to `MPI_WIN_COMPLETE` have occurred.** This guarantees that all these origin processes have completed their RMA accesses to the local window. When the call returns, all these RMA accesses will have completed at the target window.

It is *non-local*

# Results, independent from implementation choices:

MPI_Win_Start	non-local
MPI_Put	local (AND incomplete → nonblocking)
MPI_Get	local (AND incomplete → nonblocking)
MPI_Accumulate	local (AND incomplete → nonblocking)
MPI_Win_complete	local

From the MPI text and of course also with same testing one can see:

MPI_Win_post	local
MPI_Win_wait	non-local
MPI_Win_test	local
MPI_Win_fence	non-local
MPI_Win_lock	non-local
MPI_Win_unlock	local

As already shown  
in the proposal for  
the RMA tables in  
Annex A.2

# Additional rationale to Issue 96 to more completely describe the “user’s perspective” in Annex A.2, Footnote 21 (part 4)

MPI-4 Issue #96 Annex A.2 Footnote 21 on RMA procedures read (including the change from Part 1)

- 21) In some cases, more than one MPI procedure may be needed to implement one stage of an MPI one-sided operation.  
The locality of the RMA procedures is analyzed from the user’s perspective, i.e., independent of the choices to implementors about weak synchronization as described in Sections 11.5, 11.5.2 (on Example 11.4), and 11.5.3 (on Example 11.5).  
For details on the semantics of one-sided operations, see Chapter 11.

but should read

- 21) In some cases, more than one MPI procedure may be needed to implement one stage of an MPI one-sided operation.  
The locality of the RMA procedures is analyzed from the user’s perspective, i.e., independent of the choices to implementors about weak synchronization as described in Sections 11.5, 11.5.2 (on Example 11.4), and 11.5.3 (on Example 11.5).

Rationale.

For this user’s perspective, the definition of non-local can be formalized as follows:

(see next slide)

(End of rationale.)

For details on the semantics of one-sided operations, see Chapter 11.



# Additional rationale to Issue 96 to more completely describe the “user’s perspective” in Annex A.2, Footnote 21 (part 4, continued)

21) In some cases, more than one MPI procedure may be needed to implement one stage of an MPI one-sided operation.

The locality of the RMA procedures is analyzed from the user’s perspective, i.e., independent of the choices to implementors about weak synchronization as described in Sections 11.5, 11.5.2 (on Example 11.4), and 11.5.3 (on Example 11.5).

from  
PART II.

Rationale.

For this user’s perspective, the definition of a *non-local procedure* (see Section 2.4.2) can be formalized as follows:

An MPI procedure MPI\_A is non-local, if there exists a correct (i.e. also deadlock-free) MPI program with a call to MPI\_A on one process and if there exists a call to an MPI procedure MPI\_B on another process

MPI\_A ←----- MPI\_B

so that if we add a test message as in the following figure

MPI\_Bsend → MPI\_Recv

MPI\_A ←----- MPI\_B

then the application is still correct (i.e. also deadlock-free), whereas the following modification may cause a deadlock:

MPI\_A ←----- MPI\_Recv

MPI\_Bsend → MPI\_B

Note that MPI\_B is then such a specific semantically-related MPI procedure on another MPI process as mentioned in the definition of the semantic term *non-local procedure* in Section 2.4.2.

(End of rationale.)

PART IV.

For details on the semantics of one-sided operations, see Chapter 11.

# Status (May 31, 2020)

- A set of additional changes:
  - No-no-votes requested for June 29-July 1, 2020 meeting (Munich → online)
  - Additional changes for the RMA chapter
    - In chap-applang/
      - v15-->v16: Semantic terms / RMA: Rationale for A.2 Footnote 21 (1 change)
    - In figures/
      - Modified MPI-semantic-appendix.pptx (now including also img5a-c)
      - New MPI-semantic-appendix\_img5a.png / .eps / .pdf (for the rationale)
      - New MPI-semantic-appendix\_img5b.png / .eps / .pdf
      - New MPI-semantic-appendix\_img5c.png / .eps / .pdf
- For first vote: Version from May 31, 2020

PART IV.

# Lets look at pdf on Issue #96 and the small changes on PR 116

- Final version from **May 31, 2020**
- pdf: <https://github.com/mpi-forum/mpi-issues/files/4708015/mpi32-report-semantic-terms-2020-05-31-annotated.pdf>
- Issue: <https://github.com/mpi-forum/mpi-issues/issues/96>
  - Here, you can find also these slides:
  - EuroMPI2019-SemanticTerms-fromPuri-2020-04-28+RMA-rab-2020-05-31.pdf
- Pull request: <https://github.com/mpi-forum/mpi-standard/pull/116>
- Any open questions?

# Thanks for listening

Questions?