Advice to users. A user can write correct programs by following the following rules:

fence: During each period between fence calls, each window is either updated by put or accumulate calls, or updated by local stores, but not both. Locations updated by put or accumulate calls should not be accessed during the same period (with the exception of concurrent updates to the same location by accumulate calls). Locations accessed by get calls should not be updated during the same period.

post-start-complete-wait: A window should not be updated locally while being posted, if it is being updated by put or accumulate calls. Locations updated by put or accumulate calls should not be accessed while the window is posted (with the exception of concurrent updates to the same location by accumulate calls). Locations accessed by get calls should not be updated while the window is posted.

With the post-start synchronization, the target process can tell the origin process that its window is now ready for RMA access; with the complete-wait synchronization, the origin process can tell the target process that it has finished its RMA accesses to the window.

lock: Updates to the window are protected by exclusive locks if they may conflict. Nonconflicting accesses (such as read-only accesses or accumulate accesses) are protected by shared locks, both for local accesses and for RMA accesses.

changing window or synchronization mode: One can change synchronization mode, or change the window used to access a location that belongs to two overlapping windows, when the process memory and the window copy are guaranteed to have the same values. This is true after a local call to MPI_WIN_FENCE, if RMA accesses to the window are synchronized with fences; after a local call to MPI_WIN_WAIT, if the accesses are synchronized with post-start-complete-wait; after the call at the origin (local or remote) to MPI_WIN_UNLOCK if the accesses are synchronized with locks.

In addition, a process should not access the local buffer of a get operation until the operation is complete, and should not update the local buffer of a put or accumulate operation until that operation is complete.

ticket37.

The RMA synchronization operations define when updates are guaranteed to become visible in public and private windows. Updates may become visible earlier, but such behavior is implementation dependent. (*End of advice to users.*)

ticket37.

The semantics are illustrated by the following examples:

Example 11.11 Rule 5:

```
Process A:                Process B:
                          window location X

                          MPI_Win_lock(EXCLUSIVE,B)
                          store X /* local update to private copy of B */
                          MPI_Win_unlock(B)
                          /* now visible in public window copy */
```

```
MPI_Barrier                MPI_Barrier

MPI_Win_lock(EXCLUSIVE,B)
MPI_Get(X) /* ok, read from public window */
MPI_Win_unlock(B)
```

Example 11.12 Rule 6:

```
Process A:                 Process B:
                           window location X

MPI_Win_lock(EXCLUSIVE,B)
MPI_Put(X) /* update to public window */
MPI_Win_unlock(B)

MPI_Barrier                MPI_Barrier

                           MPI_Win_lock(EXCLUSIVE,B)
                           /* now visible in private copy of B */
                           load X
                           MPI_Win_unlock(B)
```

Note that the private copy of X has not necessarily been updated after the barrier, so omitting the lock-unlock at process B may lead to the load returning an *obsolete* value.

Example 11.13 The rules do *not* guarantee that process A in the following sequence will see the value of X as updated by the local store by B before the lock.

```
Process A:                 Process B:
                           window location X

                           store X /* update to private copy of B */
                           MPI_Win_lock(SHARED,B)
MPI_Barrier                MPI_Barrier

MPI_Win_lock(SHARED,B)
MPI_Get(X) /* X may not be in public window copy */
MPI_Win_unlock(B)
                           MPI_Win_unlock(B)
                           /* update on X now visible in public window */
```

Example 11.14 In the following sequence

```
Process A:                 Process B:
window location X
window location Y

store Y
MPI_Win_post(A,B) /* Y visible in public window */
MPI_Win_start(A)           MPI_Win_start(A)
```

```
store X /* update to private window */

MPI_Win_complete            MPI_Win_complete
MPI_Win_wait
/* update on X may not yet visible in public window */

MPI_Barrier                 MPI_Barrier

                            MPI_Win_lock(EXCLUSIVE,A)
                            MPI_Get(X) /* may return an obsolete value */
                            MPI_Get(Y)
                            MPI_Win_unlock(A)
```

it is *not* guaranteed that process B reads the value of X as per the local update by process A, because neither MPI_Win_wait nor MPI_Win_complete calls by process A ensure visibility in the public window copy. To allow B to read the value of X stored by A the local store must be replaced by a local MPI_Put that updates the public window copy. Note that by this replacement X may become visible in the private copy in process memory of A only after the MPI_Win_wait call in process A. The update on Y made before the MPI_Win_post call is visible in the public window after the MPI_Win_post call and therefore correctly gotten by process B. The MPI_Get(Y) call could be moved to the epoch started by the MPI_Win_start operation, and process B would still get the value stored by A.

Example 11.15 Finally, in the following sequence

```
Process A:                  Process B:
                            window location X

MPI_Win_lock(EXCLUSIVE,B)
MPI_Put(X) /* update to public window */
MPI_Win_unlock(B)

MPI_Barrier                 MPI_Barrier

                            MPI_Win_post(B)
                            MPI_Win_start(B)

                            load X /* access to private window */
                                   /* may return an obsolete value */

                            MPI_Win_complete
                            MPI_Win_wait
```

rules (5,6) do *not* guarantee that the private copy of X at B has been updated before the load takes place. To ensure that the value put by process A is read, the local load must be replaced with a local MPI_Get operation, or must be placed after the call to MPI_Win_wait.