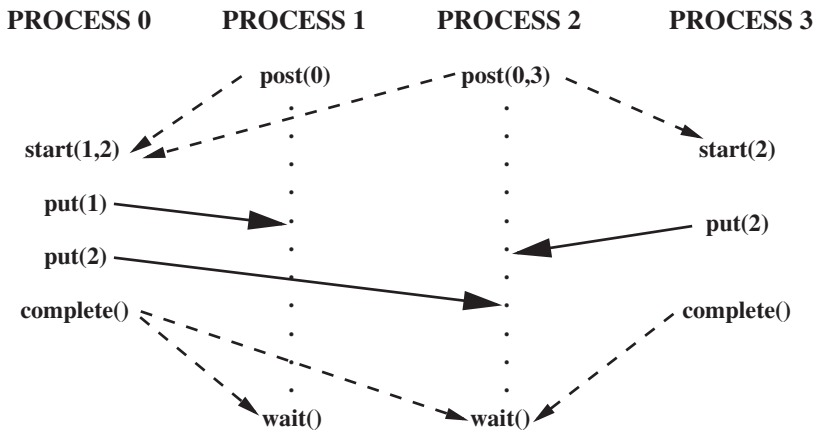

Figure 11.4: Active target communication. Dashed arrows represent synchronizations and solid arrows represent data transfer.

Figure 11.4 illustrates the use of these four functions. Process 0 puts data in the windows of processes 1 and 2 and process 3 puts data in the window of process 2. Each start call lists the ranks of the processes whose windows will be accessed; each post call lists the ranks of the processes that access the local window. The figure illustrates a possible timing for the events, assuming strong synchronization; in a weak synchronization, the start, put or complete calls may occur ahead of the matching post calls.

MPI_WIN_TEST(win, flag)

IN	win	window object (handle)
OUT	flag	success flag (logical)

```
int MPI_Win_test(MPI_Win win, int *flag)

MPI_WIN_TEST(WIN, FLAG, IERROR)
    INTEGER WIN, IERROR
    LOGICAL FLAG
```

ticket150.
ticket150.
ticket70.

{`bool MPI::Win::Test() const` *(binding deprecated, see Section 15.2)* }

[This is the nonblocking version of MPI_WIN_WAIT. It returns flag = true if MPI_WIN_WAIT would return, flag = false, otherwise. The effect of return of MPI_WIN_TEST with flag = true is the same as the effect of a return of MPI_WIN_WAIT. If flag = false is returned, then the call has no visible effect.] This is the nonblocking 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_WIN_POST call have been completed as signalled by matching MPI_WIN_COMPLETE calls, and flag = false otherwise. In the former case MPI_WIN_WAIT would have returned immediately.

ticket70.

MPI_WIN_TEST should be invoked only where MPI_WIN_WAIT can be invoked. Once the call has returned flag = true, it must not be invoked anew, until the window is posted anew.