Tasks 1 and 2

1 Pairs of opposites

Blocking/non-blocking: different in *function returning*
blocking: returns only when success criterion has proven valid
non-blocking: returns immediately

Asynchronous/synchronous: different in *success criterion*
asynchronous: criterion = "buffer is emptied"
synchronous: criterion = "data are received" ($\rightarrow$ handshake)

Buffered/unbuffered: different in *system buffer use*
buffered: copies data immediately to system buffer for later transmission
implies asynchronous

One-sided/two-sided: different in *number of parties* directly involved
one-sided: only one party (getter or putter) = remote memory access (RMA)
(but 2nd party needs to prepare access)
two-sided: two parties - corresponding sender and receiver

Remarks:

Standard send: may be buffered or synchronous

Ready send: completes immediately
Succeeds normally if a matching receive is already posted. Otherwise, the outcome is undefined.

2 Overview of send/recv routines

two-sided

type	blocking	synchronous	buffered	remarks
MPI_Bsend	yes	no	yes	
MPI_Ibsend	no	no	yes	
MPI_Ssend	yes	yes	no	
MPI_Issend	no	yes	no	
MPI_Send	yes	depends	depends	"standard"
MPI_Isend	no	depends	depends	— "—
MPI_Rsend	yes	depends	depends	"ready"
MPI_Irsend	no	depends	depends	= receiver must be ready
MPI_Sendrecv	yes	depends	depends	no deadlocks
MPI_Recv	yes	not applicable	not applicable	
MPI_Irecv	no	not applicable	not applicable	

one-sided

type	blocking
MPI_Put	yes
MPI_Rput	no
MPI_Get	yes
MPI_Rget	no

Significance of "depends" = routines employ an (implementation-dependent) default system buffer. If too small, behavior switches to "synchronous".
$\longrightarrow$ occurence of deadlocks may depend on message size

3 Rankings

Memory usage: one-sided, synchronous $\longrightarrow$ buffered

Time: ready, one-sided $\longrightarrow$ buffered $\longrightarrow$ synchronous

Potential for concurrency: blocking synchronous $\longrightarrow$ blocking buffered
$\longrightarrow$ non-blocking, one sided

Result reproducibility: ready $\longrightarrow$ synchronous

Pros and cons:

type	pros	cons
non-blocking	communication continues in background, process can continue with other work, returning later to check successful communication completion → communication in two stages: initiation — completion test; no deadlocks	placement of completion check requires great care
buffered	predictability – sender and receiver guaranteedly not synchronised; defined behavior at network overload: error occurs.	no pre-allocated buffer space, must explicitly be attached
synchronous	safe as network can never become overloaded with undeliverable messages; more predictable than standard mode as sender and receiver always synchronised; simpler debugging as no undelivered and "invisible" messages	can be slower than buffered mode
one-sided	arbitrary access to remote memory → communication partner can be dynamical function of data	requires great care to avoid interfering accesses

Guidelines:

- develop code and debug initially with safest (most predictable) mode: blocking synchronous
- for performance optimization, enable concurrency at increasing levels of error proneness and debugging intricacy:
  ⟶ nonblocking, asynchronous ⟶ one-sided