MPI
Local view of communication ops

• Blocking: Call doesn’t return until all resources (buffers etc) are available for reuse. All state transitions related to the call are done before it returns.

• Non-blocking: Call only starts the operation; it may return before all effects are completed and before resources can be reused
Global view of communication ops

• Synchronous: Operation does not complete until both sides have started the operation

• Asynchronous: Sender can do its side without coordination with receiver
Which of the following is true about MPI_Send and MPI_Recv?

A. Both are synchronous
B. Both are asynchronous
C. One is synchronous and the other is asynchronous
D. Whether they are synchronous or asynchronous is implementation dependent
E. I hate MPI already
Which of the following is true about MPI_Send and MPI_Recv?

A. Both are synchronous
B. Both are asynchronous
C. One is synchronous and the other is asynchronous
D. Whether they are synchronous or asynchronous is implementation dependent
E. I hate MPI already
What are the arguments to MPI_Send?

(Try to come up with them without looking; what arguments are needed?)
int MPI_Send(void* mesg, int count,
    MPI_datatype datatype, int dest, int tag,
    MPI_Comm comm);

MPI_Recv adds MPI_Status* status to the end
MPI delivers messages between a pair of processes in the order they are sent, but can appear otherwise if more than 2 processes are involved

• Illustrate this in code

• Why is this allowed?
How can MPI_Send and MPI_Recv cause deadlocks?
Which of the following is NOT true about MPI’s non-blocking calls?

A. These calls make it easier to overlap communication and computation

B. After making a non-blocking send, the programmer must not change the sent buffer until confirming that the send completed

C. It is possible to test whether a previous call has finished

D. Non-blocking collective communication operations are available

E. Not exactly one of the above
Which of the following is NOT true about MPI’s non-blocking calls?

A. These calls make it easier to overlap communication and computation
B. After making a non-blocking send, the programmer must not change the sent buffer until confirming that the send completed
C. It is possible to test whether a previous call has finished
D. Non-blocking collective communication operations are available
E. Not exactly one of the above
MPI+X

• X is used to manage intra-process parallelism

• X can be OpenMP, CUDA, OpenCL, ...

• May also assignment multiple MPI ranks to a given physical node (e.g. one per core)
MPI and task mapping