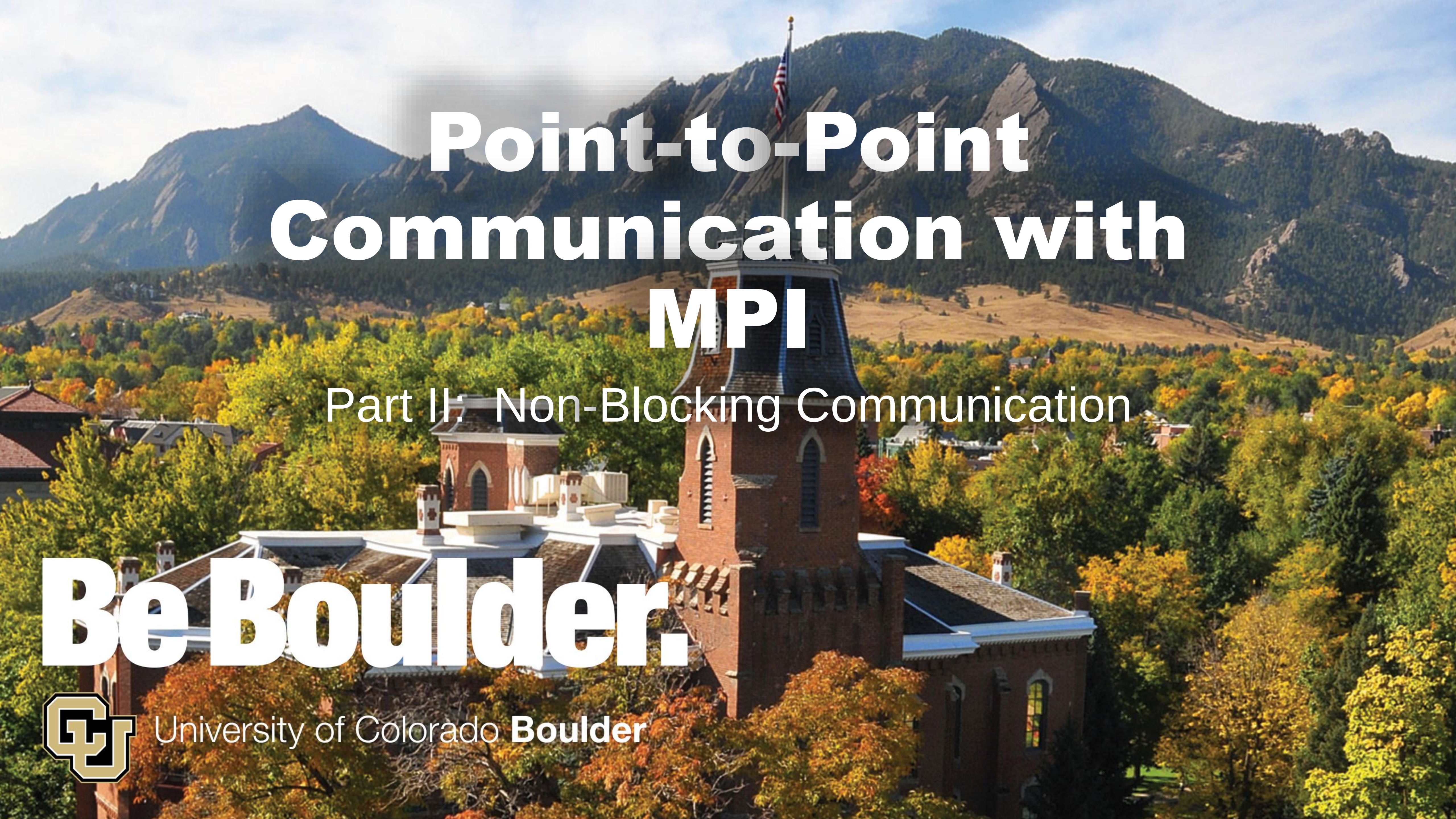


Point-to-Point Communication with MPI

Part II: Non-Blocking Communication

Be Boulder.

University of Colorado **Boulder**

Outline

- Blocking vs. Non-blocking Communication
- ISend/IReceive Syntax
- Sample program
- Deadlock
- Exercises

Useful MPI References

- General MPI (C++/Fortran):

  https://www.mpich.org/documentation/guides/

- Mpi4py (Python):

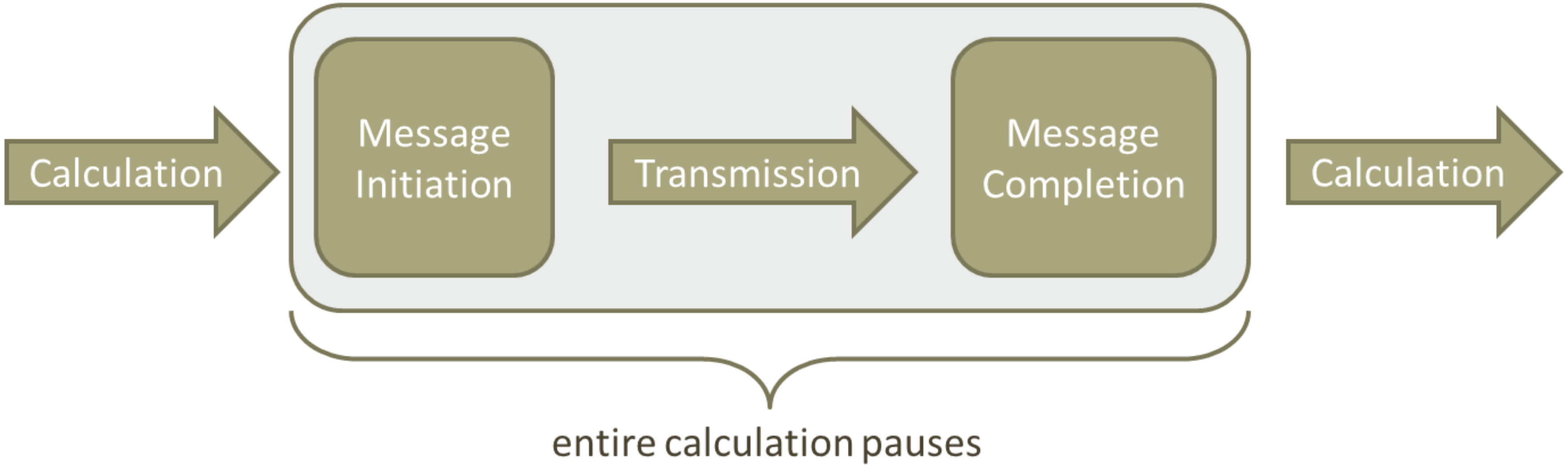
  http://mpi4py.scipy.org/docs/usrman/index.html

- pdbMPI (R):

  https://cran.r-project.org/web/packages/pbdMPI/index.html

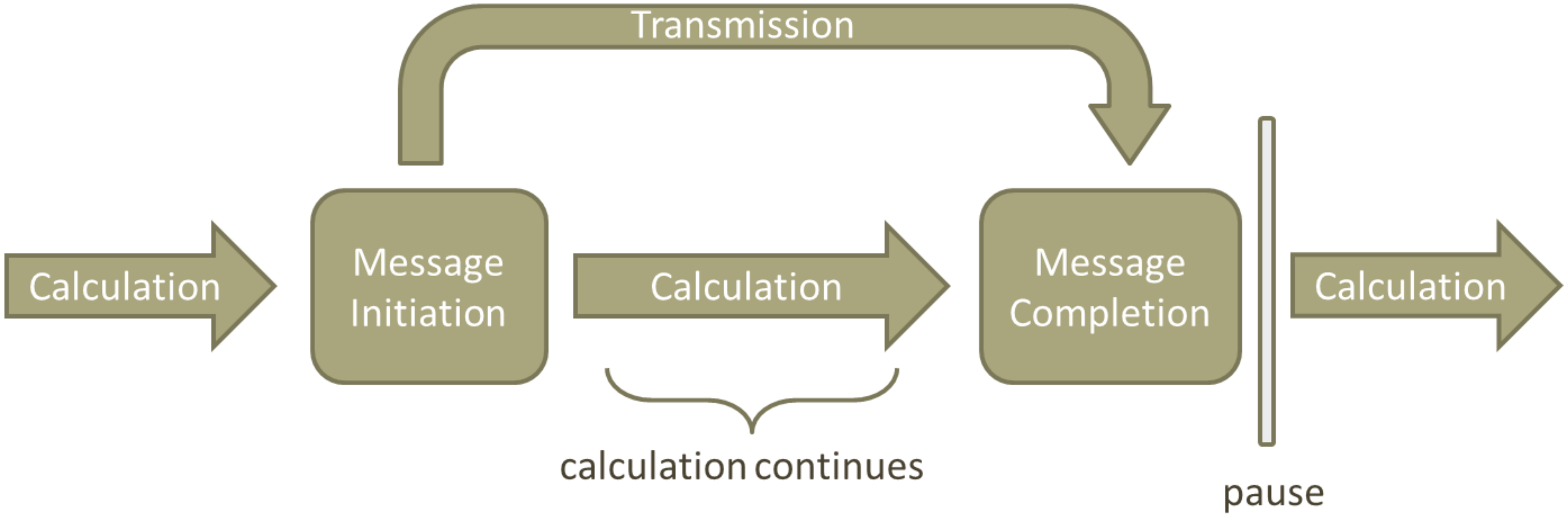
Blocking Communication: Program Flow

- Programs written using blocking sends & receives possess portions similar to schematic below:



Non-Blocking Communication: Program Flow

- Programs written using ISends & IReceives possess portions that are schematically similar to:



- Useful for maximizing CPU usage

Non-Blocking Send & Receive

- Same syntax as MPI_Send() and MPI_Recv()
    - Addition of a request handle argument.
- Calls return immediately
- Data in the buffer (send and receive) *should not* be accessed until operation is complete.
- Send and receive are completed by
    - MPI_Test
    - MPI_Wait

MPI_ISEND (General Syntax)

- Same syntax as MPI_SEND with the addition of a request handle
- Calling syntax:
    - MPI_ISend(buf, cnt, dtype, dest, tag, comm, request, ierr)
- Request is a handle (int in Fortran; MPI_Request in C) used to check for completeness of the send
- This call returns immediately
- Data in buf *should not* be accessed until the user has completed the send operation
- The send is completed by a successful call to MPI_TEST or a call to MPI_WAIT

MPI IRecv

- Same syntax as MPI_RECV except status is replaced with a request handle
- Calling syntax:
  - MPI_IRECV(buf, cnt, dtype, source, tag, comm, request, ierr)
- Request is a handle used to check on IRecv status

  (int in Fortran; MPI_Request in C; special class in Python)
- This call returns immediately
- Data in buf *should not* be accessed until the user has completed the receive operation
- The receive is completed by a call to MPI_TEST or a call to MPI_WAIT

MPI_WAIT

- Calling syntax:
  - Call MPI_Wait(request, status, ierr)
- Request is the handle returned by the non-blocking send or receive call
- Upon return, status holds source, tag, and error code information
- This call does not return until the non-blocking call referenced by *request* has completed
- Upon return, the request handle is freed
- If *request* was returned by a call to MPI_ISEND, return of this call indicates nothing about the destination process

MPI_WAITALL

- Calling syntax:
  - Call MPI_Waitall(count,requests, statuses, ierr)
- *requests* is an array of handles returned by non-blocking send or receive calls
- *count* is the number of requests
- This call does not return until all non-blocking call referenced by *requests* have completed
- Upon return, *statuses* hold source, tag, and error code information for all the calls that completed
- Upon return, the request handles stored in *requests* are all freed

Python Considerations

- Wait and Waitall are methods of the Request class
- Calling syntax:
  - My_request.Wait()
  - My_request.Waitall([My_request, My_other_request,…])
  - ... a bit non-intuitive

Example Program

- Some examples of non-blocking communication:

Advanced_P2P/mpi_imessages.{f90,cpp,py,R}

- Let’s look at the code

- Uncomment the appropriate lines in job.sh

- Submit your batch script

MPI_Waitall

Good Logic

- Only processes sending and receiving call waitall

```
If (my_rank == 0):
        isend( to rank N)
        ireceive( from rank N)
        MPI_waitall()
If (my_rank == N):
        ireceive( from rank 0)
        isend( to rank 0)
        MPI_waitall()
```

Bad Logic

- Processes not sending and receiving call waitall…

```
If (my_rank == 0):
        isend( to rank N)
        ireceive( from rank N-1)
If (my_rank == N):
        isend( to rank 0)
        ireceive( from rank 0)
MPI_waitall()
```

MPI_Waitall

Good Logic

- Only processes sending and receiving call waitall

```
If (my_rank == 0):
        isend( to rank N)
        ireceive( from rank N)
        MPI_waitall()
If (my_rank == N):
        ireceive( from rank 0)
        isend( to rank 0)
        MPI_waitall()
```

Bad Logic

Quick Exercise:
Mimic this bad logic to "break" your program

```
If (my_rank == 0):
        isend( to rank N)
        ireceive( from rank N-1)
If (my_rank == N):
        isend( to rank 0)
        ireceive( from rank 0)
MPI_waitall()
```

Exercises

- Exercise 1: Clearing a deadlock/poor Isend/Irecv logic
- Exercise 2 & 3: Convert send/recv code to Isend/Irecv’s