CME 194 Introduction to MPI

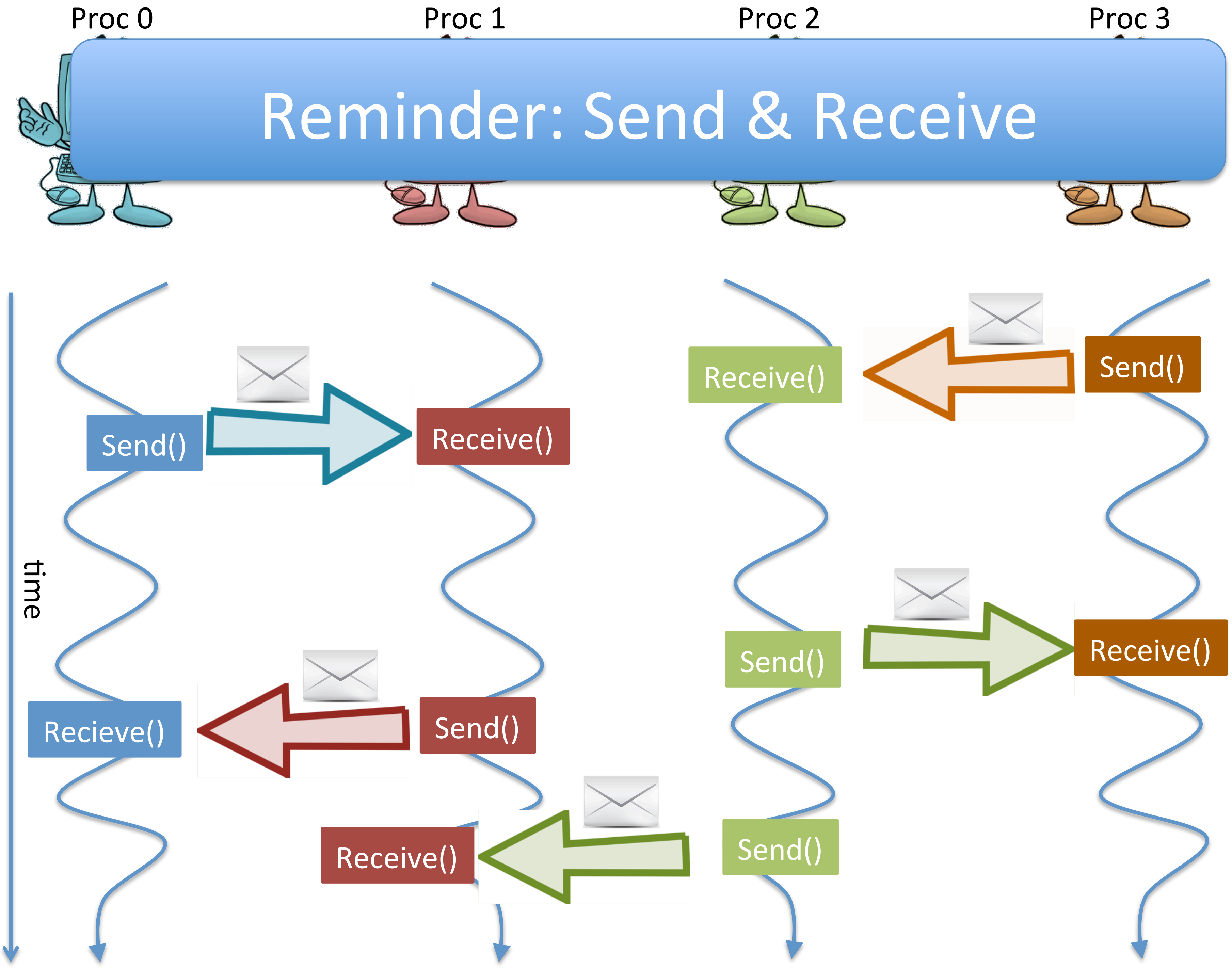
Essentia Callidus

http://cme194.stanford.edu

Recap

- **Last class:** Communicators & Derived Datatypes
  - Communication between arbitrary subsets of processes
  - Grid style communication
  - Communicate arbitrary messages
- **This class:** Point to Point Communication improved
  - Improving Send & Receive

Reminder: Send & Receive



Proc 0

Proc 1

Proc 2

Proc 3

time

Receive()

Send()

Send()

Receive()

Send()

Receive()

Recieve()

Send()

Receive()

Send()

Reminder: Send & Receive

```
int MPI_Send(void* buf, int count,
              MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm)
```

```
int MPI_Recv( void* buf, int count,
              MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm,
              MPI_Status* status)
```

Details

- Each **Send** must be **matched** with a **Recv.**
- Messages are **delivered in** the **order sent**.
- Unmatched sends/receives **may** result in **deadlock**

Deadlocks

Process 0	Process 1	Deadlock(?)

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()	

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

Send()
Recv()
	Send()
	Recv()	

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

Send()
Recv()
	Send()
	Recv()
		Depends on library

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

Send()
Recv()
	Send()
	Recv()
		Depends on library

Send()
Recv()
	Recv()
	Send()	

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

Send()
Recv()
	Send()
	Recv()
		Depends on library

Send()
Recv()
	Recv()
	Send()
		Safe

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

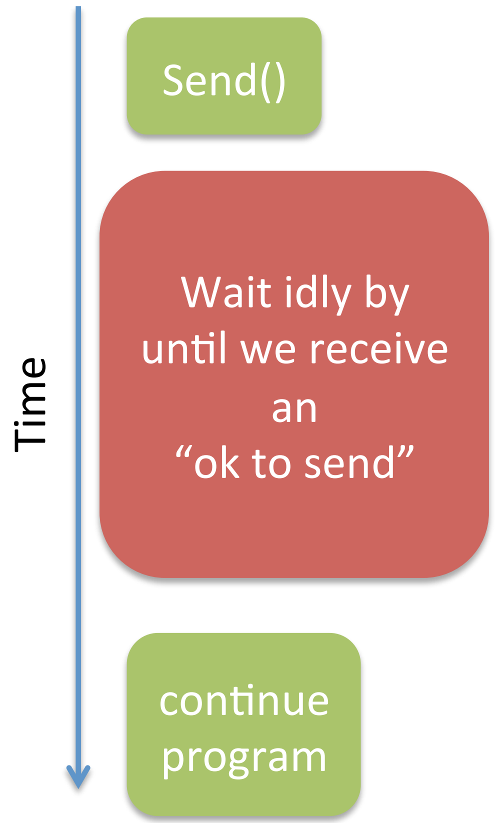
Send()
Recv()
	Send()
	Recv()
		Depends on library

Send()
Recv()
	Recv()
	Send()
		Safe

The MPI **Implementation decides if** a **standard Send()** is **buffered or not**

Send and buffering

Unbuffered send



Time

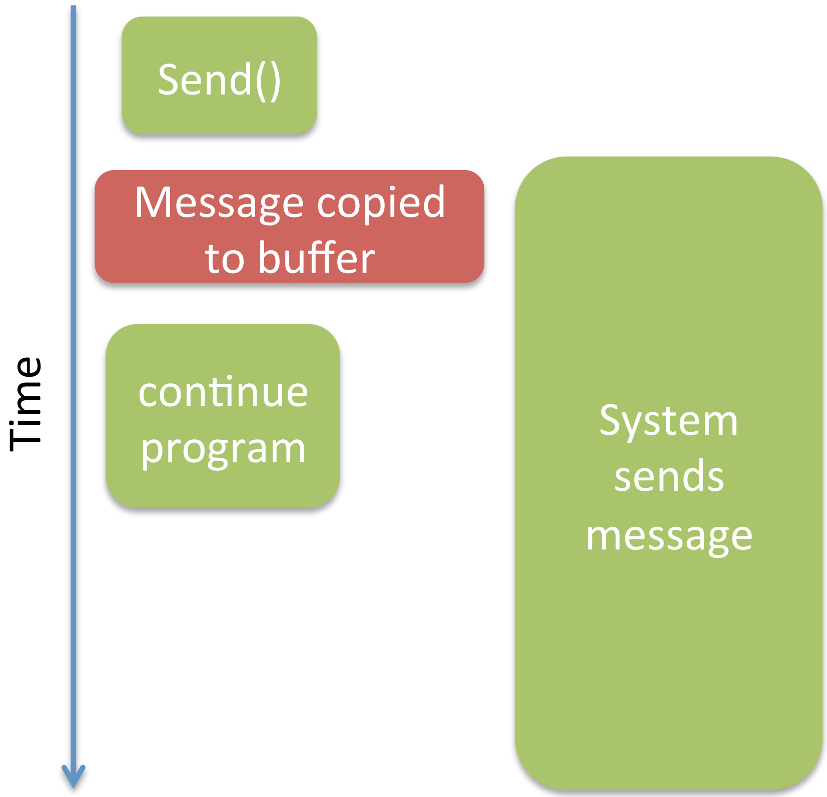
Send()

Wait idly by until we receive an “ok to send”

continue program

Note:
No guarantee other process has received

Buffered send



Time

Send()

Message copied to buffer

continue program

System sends message

The MPI **Implementation decides if** a **standard Send()** is **buffered or not**

Communication Modes

Various communication modes for point to point

- Blocking (vs. non-blocking e.g immediate)
- Buffered (vs. unbuffered)
- Synchronous (vs. asynchrous)
- Ready mode

Some imply others:

- Unbuffered implies blocking

Explicit Buffering

- We may **force** the system to buffer messages:

```
int MPI_Bsend(void* msg, int count,
                 MPI_Datatype datatype, int dest,
                 int tag, MPI_Comm comm)
```

requires explicitly **providing** a block of memory for the **buffer**

```
  int MPI_Buffer_attach(void* buf, int size)
  int MPI_Buffer_detach(void* buf, int size)
```

This is usually a bad idea

- Error prone (what if no space?)
- Fly's in face of goal (doubling memory for messages)
- May be unneeded (what if we know receive is posted?)
- Better techniques exist.
- **Note: MPI_Brecv() makes no sense**

Deadlocks

Process 0	Process 1	Deadlock(?)

Recv()
Send()
	Recv()
	Send()
		Always

Send()
Recv()
	Send()
	Recv()
		Depends on library

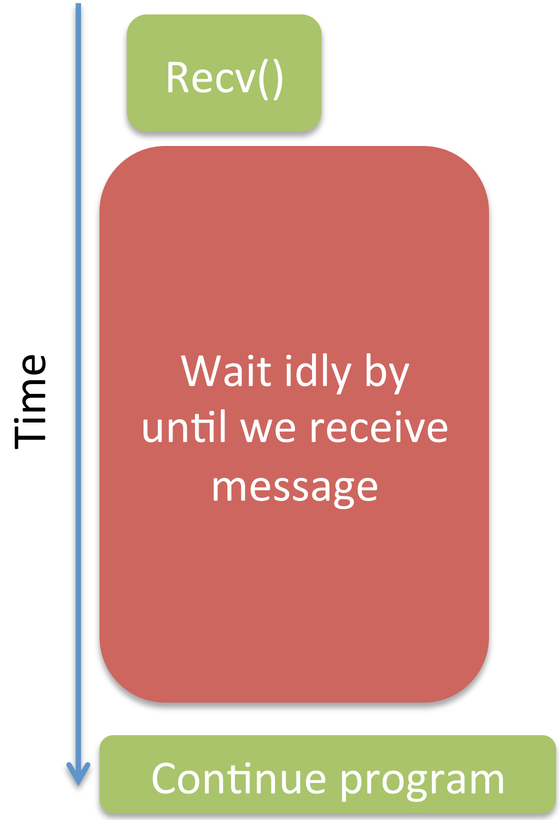
Send()
Recv()
	Recv()
	Send()
		Safe

- Deadlocks may be result of inexpressibility of the language
- Example: Replace Example 1 above with:
  - Receives which return **immediately**

```
MPI_Irecv(void* msg, int count,
        MPI_Datatype datatype, int source,
        int tag, MPI_Comm comm, MPI_Request* req)
```

Non-blocking Receive

Blocking Receive



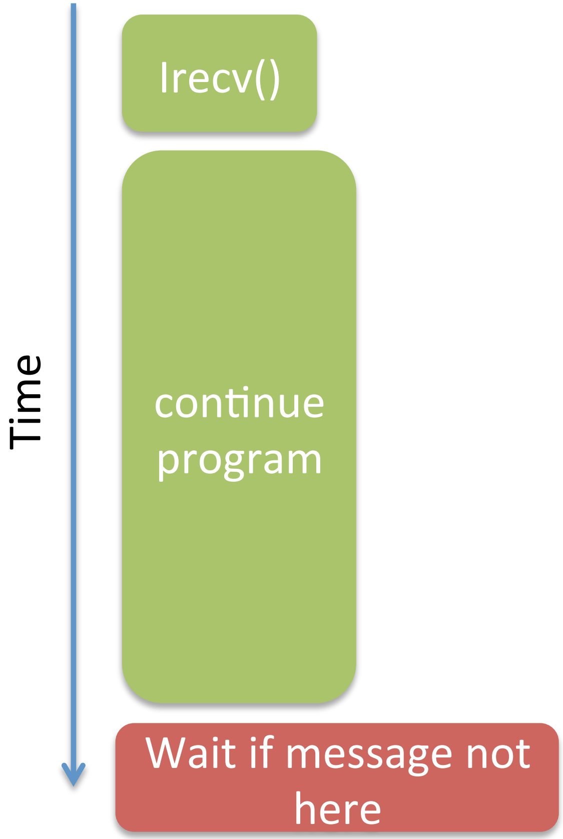
Time

Recv()

Wait idly by until we receive message

Continue program

Non-blocking Receive



Time

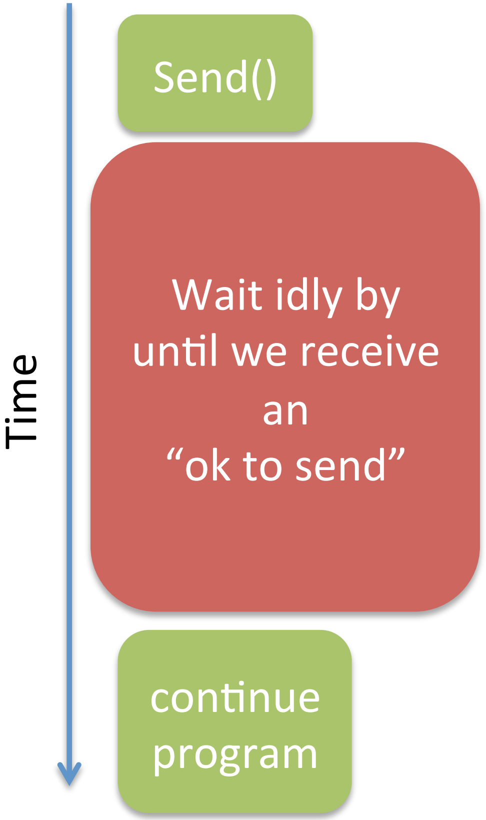
Irecv()

continue program

Wait if message not here

Non-blocking Send

Un-buffered blocking send



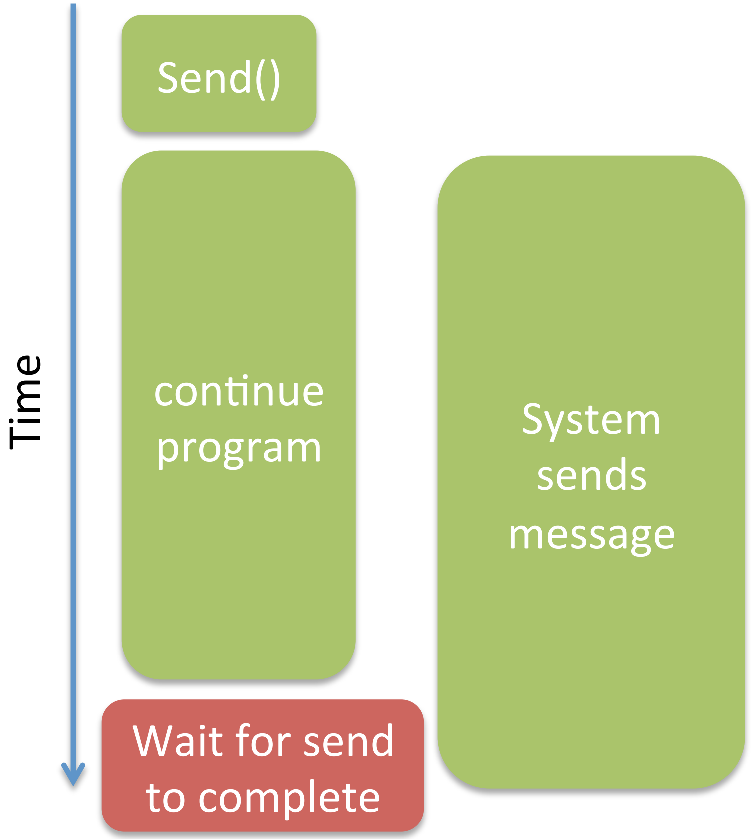
Time

Send()

Wait idly by until we receive an
“ok to send”

continue program

Non-blocking send



Time

Send()

continue program

System sends message

Wait for send to complete

Non-blocking communication

```
MPI_Isend(void* msg, int count,
       MPI_Datatype datatype, int dest,
       int tag, MPI_Comm comm, MPI_Request* req)
```

```
MPI_Irecv(void* msg, int count,
       MPI_Datatype datatype, int source,
       int tag, MPI_Comm comm, MPI_Request* req)
```

- Sends are receive calls return immediately
- Operation was not necessarily successful
- Send **may be** asynchronous
- Send **may be** synchronous
- Data was not necessarily buffered

May explicitly block via:

```
MPI_Wait(MPI_Request* req, MPI_Status* status)
```

Non-blocking communication

```
MPI_Isend(void* msg, int count,
       MPI_Datatype datatype, int dest,
      int tag, MPI_Comm comm, MPI_Request* req)
```

```
MPI_Irecv(void* msg, int count,
       MPI_Datatype datatype, int source,
      int tag, MPI_Comm comm, MPI_Request* req)
```

- Sends are receive calls return immediately
- Operation was not necessarily successful
- Send **may be** asynchronous
- Send **may be** synchronous
- Data was not necessarily buffered

May explicitly block via:

```
MPI_Wait(MPI_Request* req, MPI_Status* status)
```

Non-blocking communication

```
MPI_Ibsend(void* msg, int count,
       MPI_Datatype datatype, int dest,
      int tag, MPI_Comm comm, MPI_Request* req)
```

```
MPI_Ibrecv() makes no sense.
```

- Sends are receive calls return immediately
- Operation was not necessarily successful
- Send **may be** asynchronous
- Send **may be** synchronous
- Data ~~was not~~ necessarily buffered

Explicit Buffering

```
MPI_Wait(MPI_Request* req, MPI_Status* status)
```

Synchronous communication

- Want a guarantee:
  - after send returns **receiving has began**
- MPI_Srecv() makes no sense.

```
MPI_Ssend(void* msg, int count,
       MPI_Datatype datatype, int dest,
       int tag, MPI_Comm comm)
```

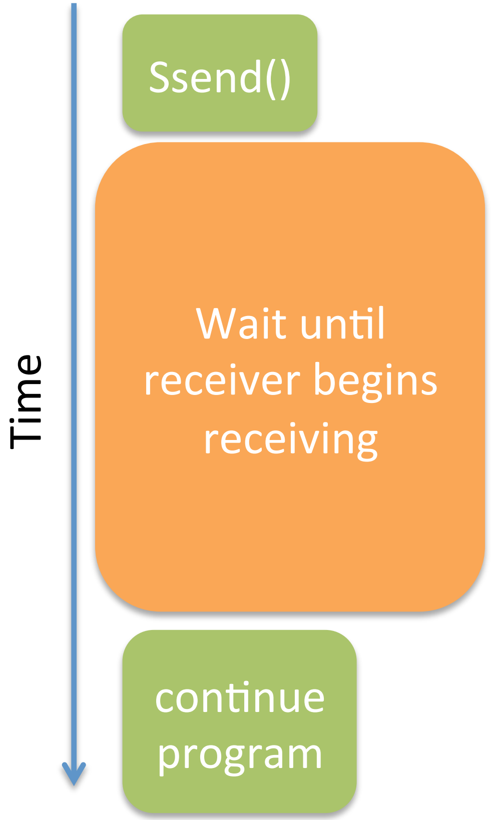
Useful for guaranteeing process has gotten to a certain stage of program.

It is perfectly reasonable to have a non-blocking equivalent:

```
MPI_Issend(void* msg, int count,
       MPI_Datatype datatype, int dest,
       int tag, MPI_Comm comm, MPI_Request* req)
```

Synchronous communication

Synchronous blocking send



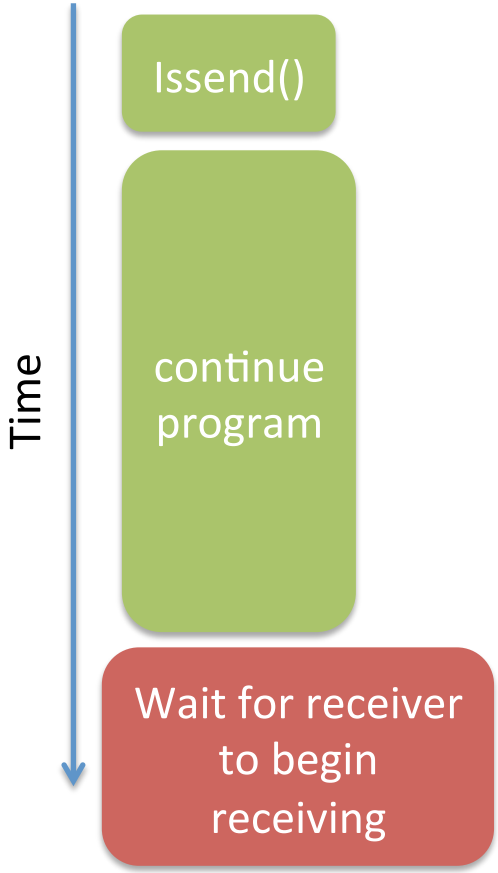
Time

Ssend()

Wait until receiver begins receiving

continue program

Non-blocking Synchronous send



Time

Issend()

continue program

Wait for receiver to begin receiving

What if we know that the receiver is ready to receive already?

Ready Send

If we know that the receiver is ready to receive already?

Example:

Irecv() followed immediately by Ssrecv()

We may optimize the internal send with:

```
MPI_Rsend(void* msg, int count,
       MPI_Datatype datatype, int dest,
       int tag, MPI_Comm comm)
```

```
MPI_Irsend(void* msg, int count,
       MPI_Datatype datatype, int dest,
       int tag, MPI_Comm comm, MPI_Request* req)
```

Rest of lecture

- Volunteers for Lecture 8
- Work on homeworks!

Next Lecture

- Volunteers for Lecture 8
- Work on homeworks!

Guest Lecture: Rob Schreiber from H.P. Labs on High Performance MPI software