Introduction to Parallel Computing (CMSC498X / CMSC818X)

Lecture 5: Programming in MPI

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Assignments dates are on the website
- Project milestone dates are on the website

Basic MPI routines

- MPI_Init
- MPI_Finalize
- MPI_Comm_rank
- MPI_Comm_size
- MPI_Send
- MPI_Recv

MPI communicators

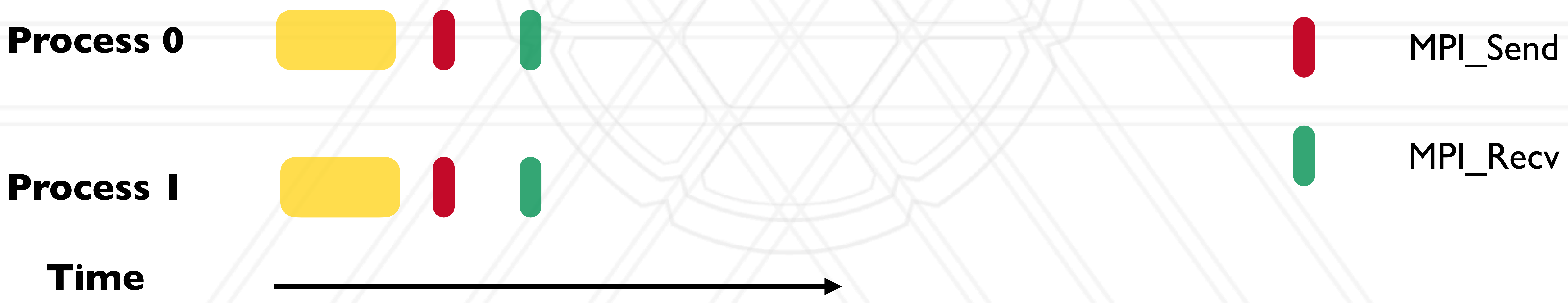
- Communicator represents a group or set of processes numbered 0, … , n-1
- Every program starts with MPI_COMM_WORLD (default communicator)
  - Defined by the MPI runtime, this group includes all processes
- Several MPI routines to create sub-communicators
  - MPI_Comm_split
  - MPI_Cart_create
  - MPI_Group_incl

MPI datatypes

- Can be a pre-defined one: MPI_INT, MPI_CHAR, MPI_DOUBLE, …
- Derived or user-defined datatypes:
  - Array of elements of another datatype
  - struct data type to accomodate sending multiple datatypes

Basic MPI_Send and MPI_Recv

- MPI_Send and MPI_Recv routines are blocking
  - Only return when the buffer specified in the call can be used
  - Send: Returns once sender can reuse the buffer
  - Recv: Returns once data from Recv is available in the buffer



Process 0

Process 1

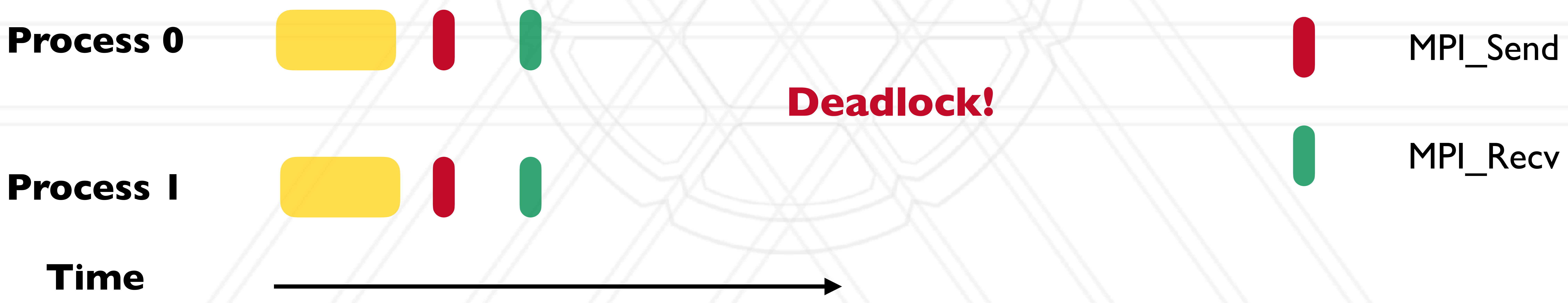
Time

🟥 MPI_Send

🟩 MPI_Recv

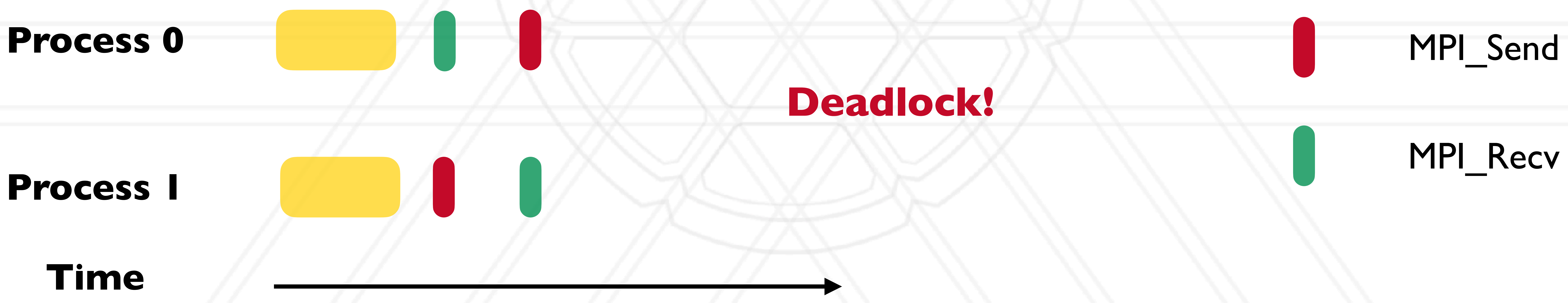
Basic MPI_Send and MPI_Recv

- MPI_Send and MPI_Recv routines are blocking
  - Only return when the buffer specified in the call can be used
  - Send: Returns once sender can reuse the buffer
  - Recv: Returns once data from Recv is available in the buffer



Basic MPI_Send and MPI_Recv

- MPI_Send and MPI_Recv routines are blocking
  - Only return when the buffer specified in the call can be used
  - Send: Returns once sender can reuse the buffer
  - Recv: Returns once data from Recv is available in the buffer



Process 0

Process 1

Time

Deadlock!

🟥 MPI_Send

🟩 MPI_Recv

Non-blocking point-to-point calls

- MPI_Isend and MPI_Irecv
- Two parts:
  - post the operation
  - Wait for results: need to call MPI_Wait or MPI_Test
- Can help with overlapping computation with communication

MPI_Isend

```
int MPI_Isend( const void *buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm, MPI_Request *request )
```

`buf`: address of send buffer

`count`: number of elements in send buffer

`datatype`: datatype of each send buffer element

`dest`: rank of destination process

`tag`: message tag

`comm`: communicator

`request`: communication request

MPI_Irecv

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

buf: address of receive buffer

count: maximum number of elements in receive buffer

datatype: datatype of each receive buffer element

source: rank of source process

tag: message tag

comm: communicator

request: communication request

MPI_Wait

```
int MPI_Wait( MPI_Request *request, MPI_Status *status )
```

`request`: communication request

status: status object

- Status object can provide information about:
  - Source process for a message: status.source
  - Message tag: status.tag
  - Number of elements: `MPI_Get_count( MPI_Status *status, MPI_Datatype datatype, int count)`

Non-blocking send/receive in MPI

```
int main(int  argc, char *argv) {
  ...

  MPI_Request req;
  MPI_Status stat;
  if (rank == 0) {
    data = 7;
    MPI_Isend(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &req);
  } else if (rank == 1) {
    MPI_Irecv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &req);

    ...
    MPI_Wait(&req, &stat);
    printf("Process 1 received data %d from process 0\n", data);
  }

  ...
}
```

Other calls

- `int MPI_Test( MPI_Request *request, int *flag, MPI_Status *status )`
- `int MPI_Waitall( int count, MPI_Request array_of_requests[], MPI_Status *array_of_statuses[] )`
- `MPI_Waitany`
- `MPI_Waitsome`

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu