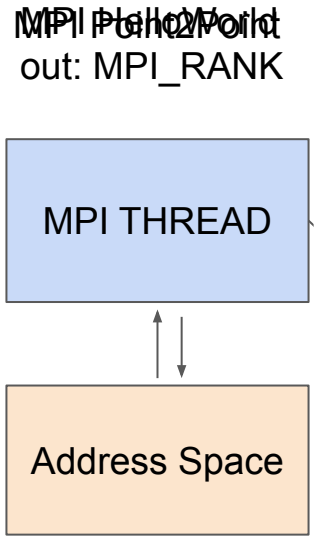
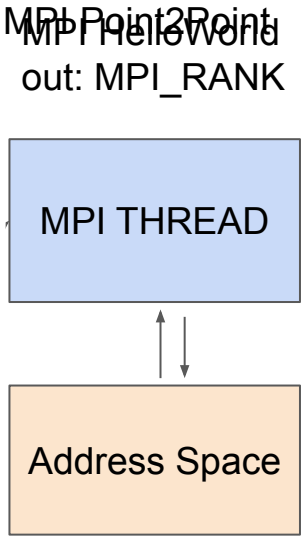
Parallelism w2

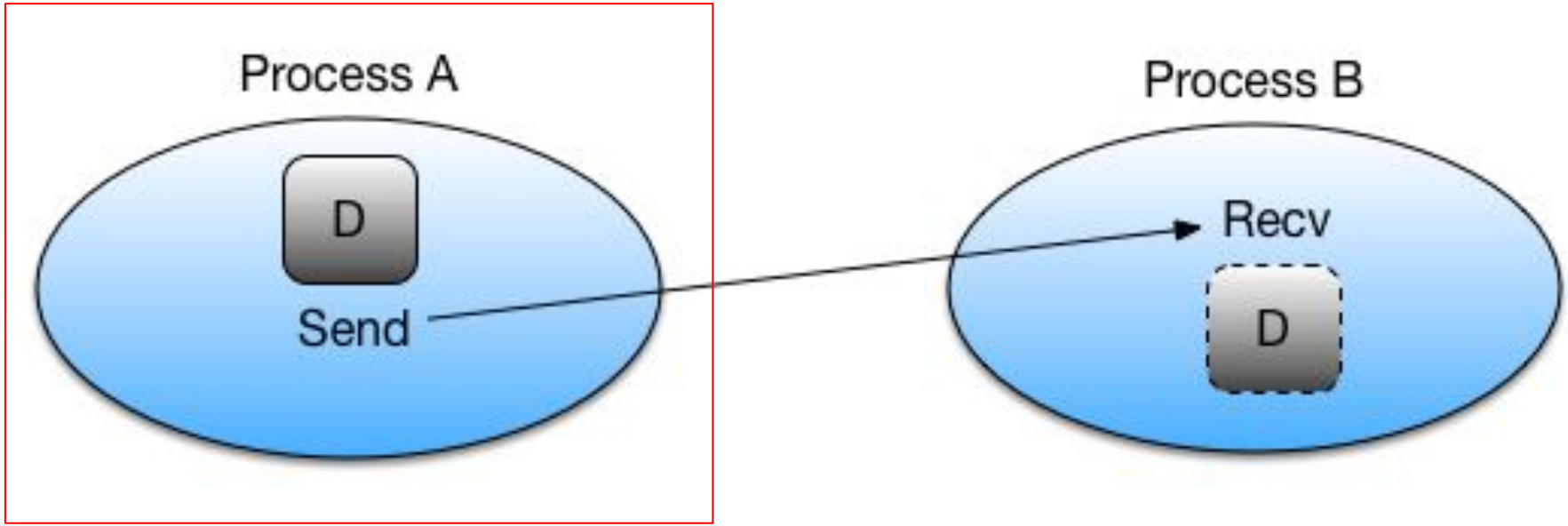
Learning to communicate

MPI - What we have learned





MPI_Send()



MPI_Send()

MPI_Send

Performs a blocking send

Synopsis

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

Input Parameters

buf
: initial address of send buffer (choice)

count
: number of elements in send buffer (nonnegative integer)

datatype
: datatype of each send buffer element (handle)

dest
: rank of destination (integer)

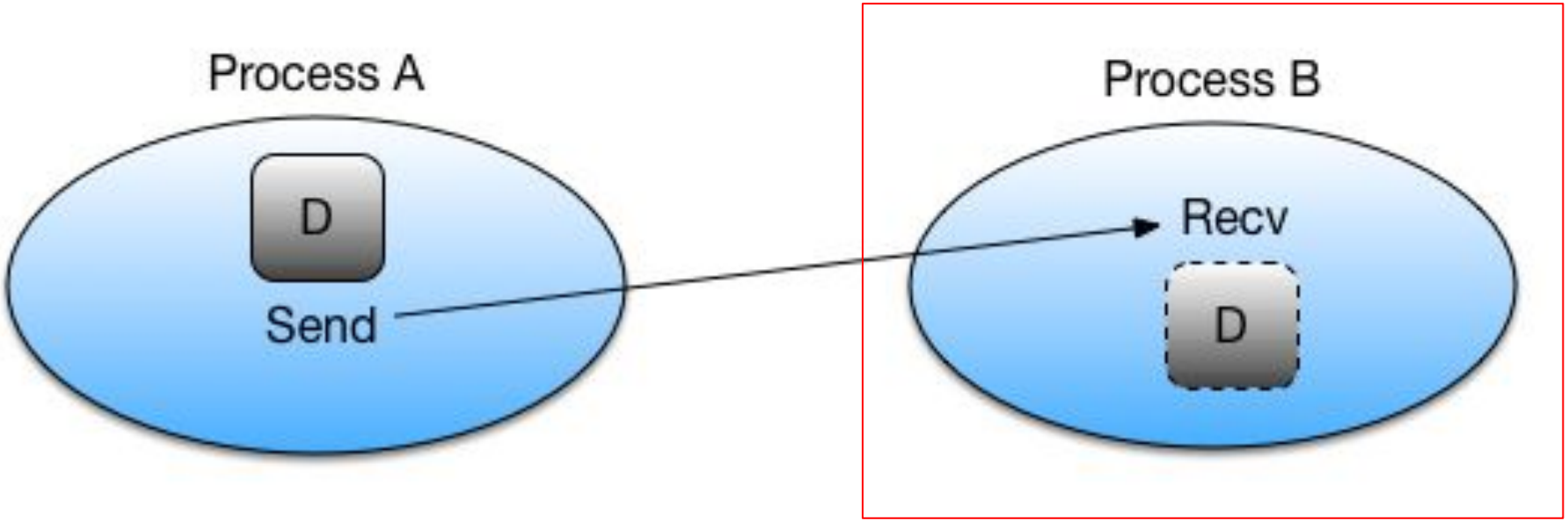
tag
: message tag (integer)

comm
: communicator (handle)

Notes

This routine may block until the message is received by the destination process.

MPI_Recv()



MPI_Recv()

MPI_Recv

Blocking receive for a message

Synopsis

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

Output Parameters

buf
: initial address of receive buffer (choice)

status
: status object (Status)

Input Parameters

count
: maximum number of elements in receive buffer (integer)

datatype
: datatype of each receive buffer element (handle)

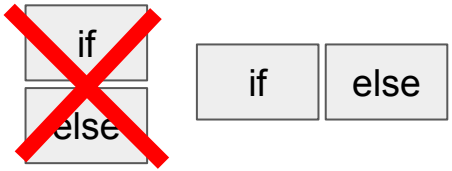
source
: rank of source (integer)

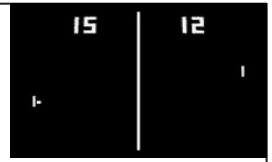
tag
: message tag (integer)

comm
: communicator (handle)

Blocking communication

- This aliasing of send() and recv() is to done to avoid deadlocks.
- Both MPI_Send() and MPI_Recv() are blocking. (**They wait for completion**)
- Use “tags” to pair them
- These scopes are parallel threads






```
. . .
MPI_Status stat;
// Allocate memory for A,B on CPU
vector<double> A_vect(N),B_vect(N);
double* A = A_vect.data();
double* B = B_vect.data();
int tag1 = 10;// can be any unique integer
int tag2 = 20;// can be any unique integer

for(int i=1; i<=5; i++){
    if(rank == 0){
        MPI_Send(A, N, MPI_DOUBLE, 1, tag1, MPI_COMM_WORLD);
        MPI_Recv(B, N, MPI_DOUBLE, 1, tag2, MPI_COMM_WORLD, &stat);
    }
    else if(rank == 1){
        MPI_Recv(B, N, MPI_DOUBLE, 0, tag1, MPI_COMM_WORLD, &stat);
        MPI_Send(A, N, MPI_DOUBLE, 0, tag2, MPI_COMM_WORLD);
    }
}
. . .
```

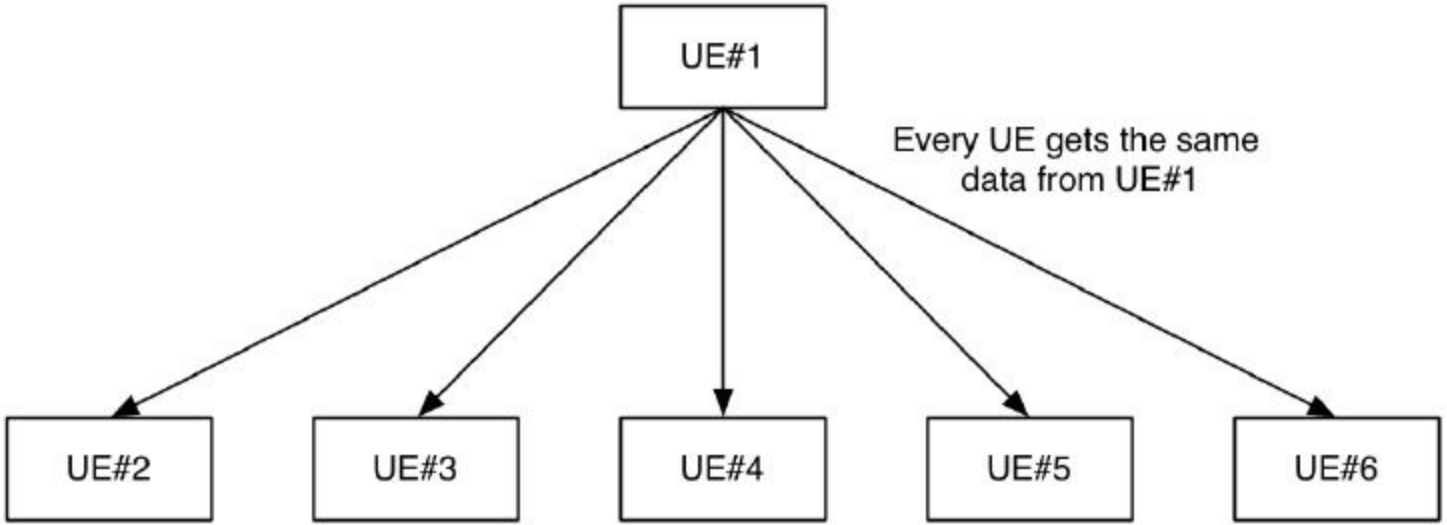
1

2

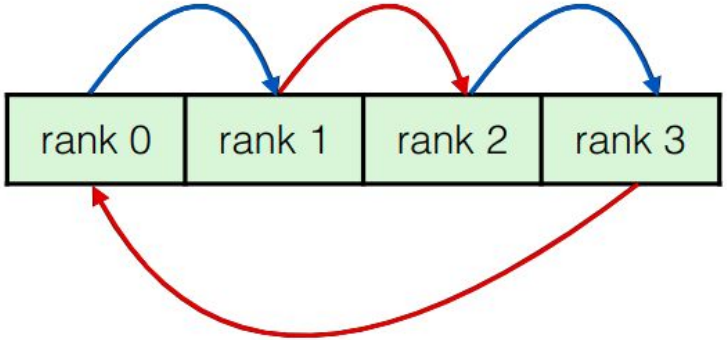
1

2

Application: Sequential Broadcast



Application: Ring Sequential



Must be aware of deadlocks:

1. send from even to odd
2. send from odd to even

HyperCube Broadcast

- Hypercube broadcasting is a communication pattern often used in parallel computing, where data needs to be distributed from one process (typically the root) to all other processes in a structured manner.
- In the context of MPI (Message Passing Interface), you can perform hypercube broadcasting using MPI_Send and MPI_Recv calls.