High Performance Computing
Lecture 40

Matthew Jacob

Indian Institute of Science

MPI References

1. Using MPI

   Gropp, Lusk, Skjellum

   www.mcs.anl.gov/mpi/usingmpi

2. MPI: The Complete Reference

   Snir, Otto, Huss-Lederman, Walker, Dongarra

   www.netlib.org/utk/papers/mpi-book/mpi-book.html

Message Passing Interface (MPI)

Standard API

- Hides software/hardware details
- Portable, flexible

Implemented as a library

Your program	
MPI Library	
Custom software	**Standard TCP/IP**
Custom hardware	**Standard network HW**

Key MPI Functions and Constants

- MPI_Init (int *argc, char ***argv)
- MPI_Finalize (void)
- MPI_Comm_rank (MPI_COMM comm, int *rank)
- MPI_Comm_size (MPI_COMM comm, int *size)
- MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
- MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
- MPI_CHAR, MPI_INT, MPI_LONG, MPI_BYTE
- MPI_ANY_SOURCE, MPI_ANY_TAG

Making MPI Programs

- Executable must be built by compiling program and linking with MPI library
  - Header files (mpi.h) provide definitions and declarations
- MPI commonly used in SPMD mode
  - One executable file
  - Multiple instances of it executed in parallel
- Implementations provide a command to initiate execution of MPI processes (mpirun)
  - Options: number of processes, which processors they are to run on

MPI Communicators

- Defines communication domain of a communication operation: set of processes that are allowed to communicate among themselves
- Initially all processes are in the communicator MPI_COMM_WORLD
- Processes have unique ranks associated with communicator, numbered from 0 to n-1
- Other communicators can be established for groups of processes

Example

```
main (int argc, char *argv[])
{
   MPI_Init(&argc, &argv);
   .
   .
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   if (myrank == 0)
        master();
   else
        slave();
   .
   .
   MPI_Finalize();
}
```

Example

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if (myrank == 0) {
   int x;
   MPI_Send(&x, 1, MPI_INT, 1, msgtag,
        MPI_COMM_WORLD);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x, 1, MPI_INT,
        0,msgtag,MPI_COMM_WORLD,status);
}
```

MPI Message Tag

- Cooperating processes may need to send several messages between each other
- Message tag: Used to differentiate between different types of messages being sent
- The message tag is carried within the message and used in both send and receive calls

Example

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if (myrank == 0) {
    …
    MPI_Send(&x, 1, MPI_INT, 1, msgtag,MPI_COMM_WORLD);
    …
    MPI_Send(&x, 1, MPI_INT, 1, msgtag,MPI_COMM_WORLD);
} else if (myrank == 1) {
    …
    MPI_Recv(&x,1,MPI_INT,0,msgtag,MPI_COMM_WORLD,status);
    …
    MPI_Recv(&x,1,MPI_INT,0,msgtag,MPI_COMM_WORLD,status);
}
```

MPI Message Tag

- Cooperating processes may need to send several messages between each other
- Message tag: Used to differentiate between different types of messages being sent
- Message tag is carried within the message and used in both send and receive calls
- If special matching is not required, a wild card message tag is used so that the receive will match with any send
  - MPI_ANY_TAG

MPI: Matching Sends and Recvs

- Sender always specifies destination and tag
- Receiver can specify for exact match or using wild cards
  - MPI_ANY_SOURCE
  - MPI_ANY_TAG

Flavours of Sends/Receives

- Synchronous
- Asynchronous

Synchronous Message Passing

- Send/Receive routines that return when message transfer completed
- Synchronous send
  - Waits until complete message can be accepted by receiving process before sending the message
- Synchronous receive
  - Waits until the message it is expecting arrives
- Synchronous routines perform two actions
  - transfer data
  - synchronize processes

Asynchronous Message Passing

- Send/receive do not wait for actions to complete before returning
- Usually require local storage for messages
- In general, they do not synchronize processes but allow processes to move forward sooner

Parameters of Send

MPI_Send (buf, count, datatype, dest, tag, comm)

- buf: Address of send buffer
- count: Number of items to send
- datatype: Datatype of each item
- dest: Rank of destination process
- tag: Message tag
- comm: Communicator

MPI Blocking and Non-blocking

- Blocking - return after local actions complete, though the message transfer may not have been completed
- Non-blocking - return immediately
  - Assumes that data storage to be used for transfer is not modified by subsequent statements prior to being used for transfer
  - Implementation dependent local buffer space is used for keeping message temporarily

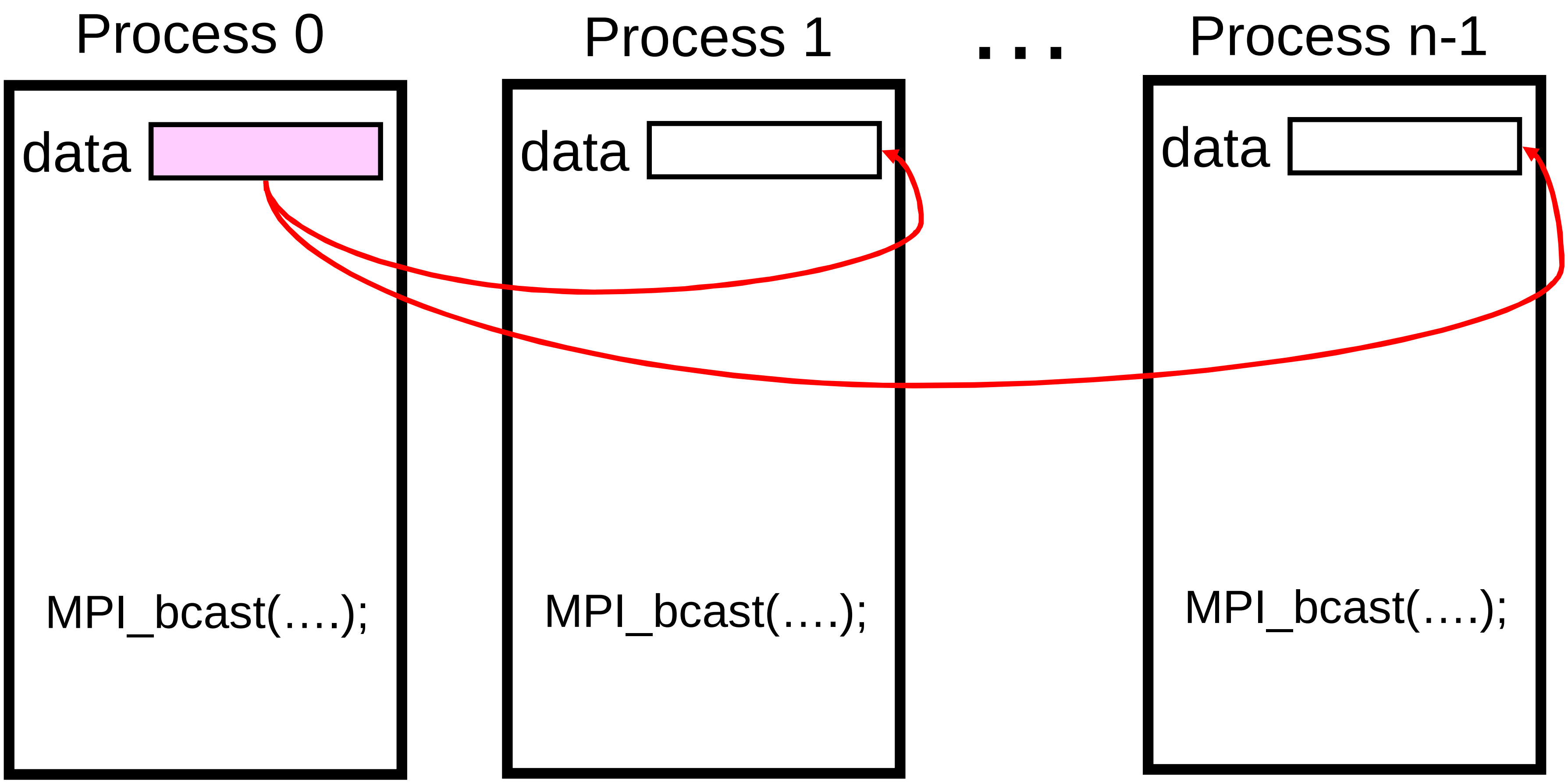
Non-blocking Routines

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)
- Completion detected by MPI_Wait() and MPI_Test()
  - MPI_Wait() waits until operation completed and then returns
  - MPI_Test() returns with flag set indicating whether or not operation has completed

MPI Group Communication

- Until now we have looked at what are called point-to-point messages
- MPI also provides routines that sends messages to a group of processes or receive messages from a group of processes
  - Not absolutely necessary for programming
  - More efficient than separate point-to-point routines
- Examples: broadcast, gather, scatter, reduce, barrier
  - MPI_Bcast, MPI_Reduce, MPI_Allreduce, MPI_Alltoall, MPI_Scatter, MPI_Gather, MPI_Barrier

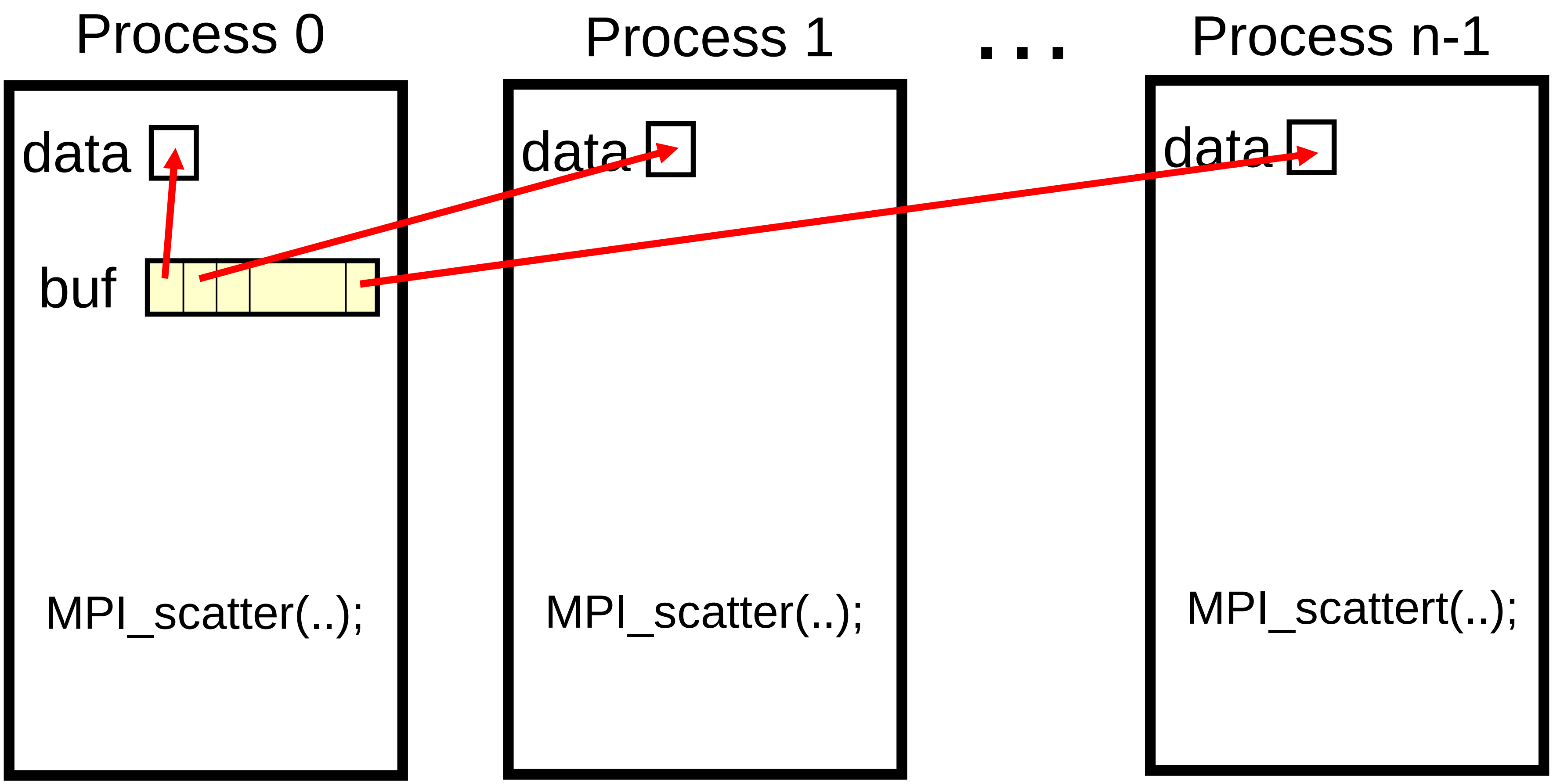
Broadcast


Process 0
Process 1
. . .
Process n-1
data
data
data
MPI_bcast(….);
MPI_bcast(….);
MPI_bcast(….);

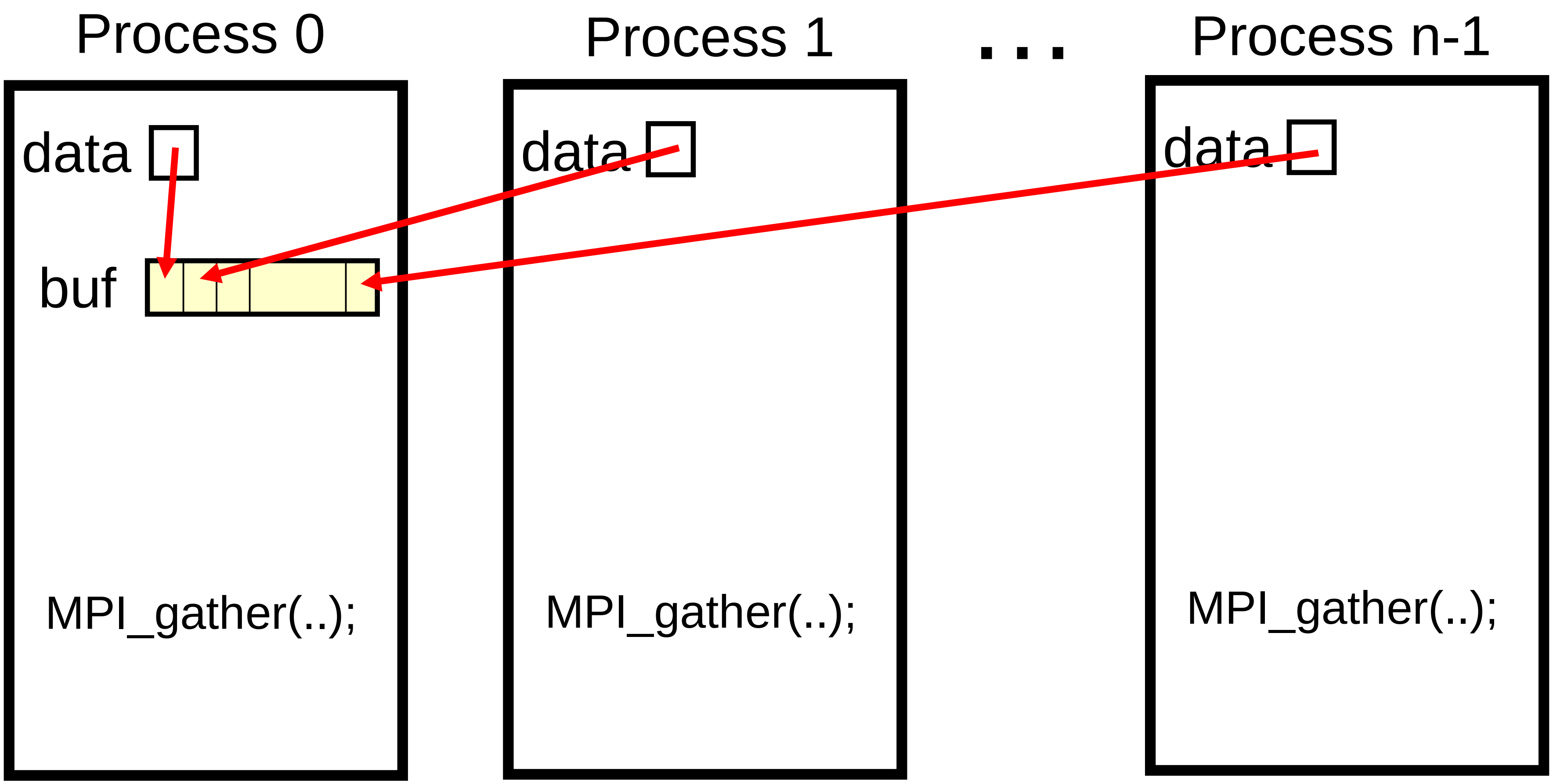
MPI Broadcast

```
MPI_Bcast (void *buf,
           int count,
           MPI_Datatype datatype,
           int root,
           MPI_Comm Comm )
```

Scatter

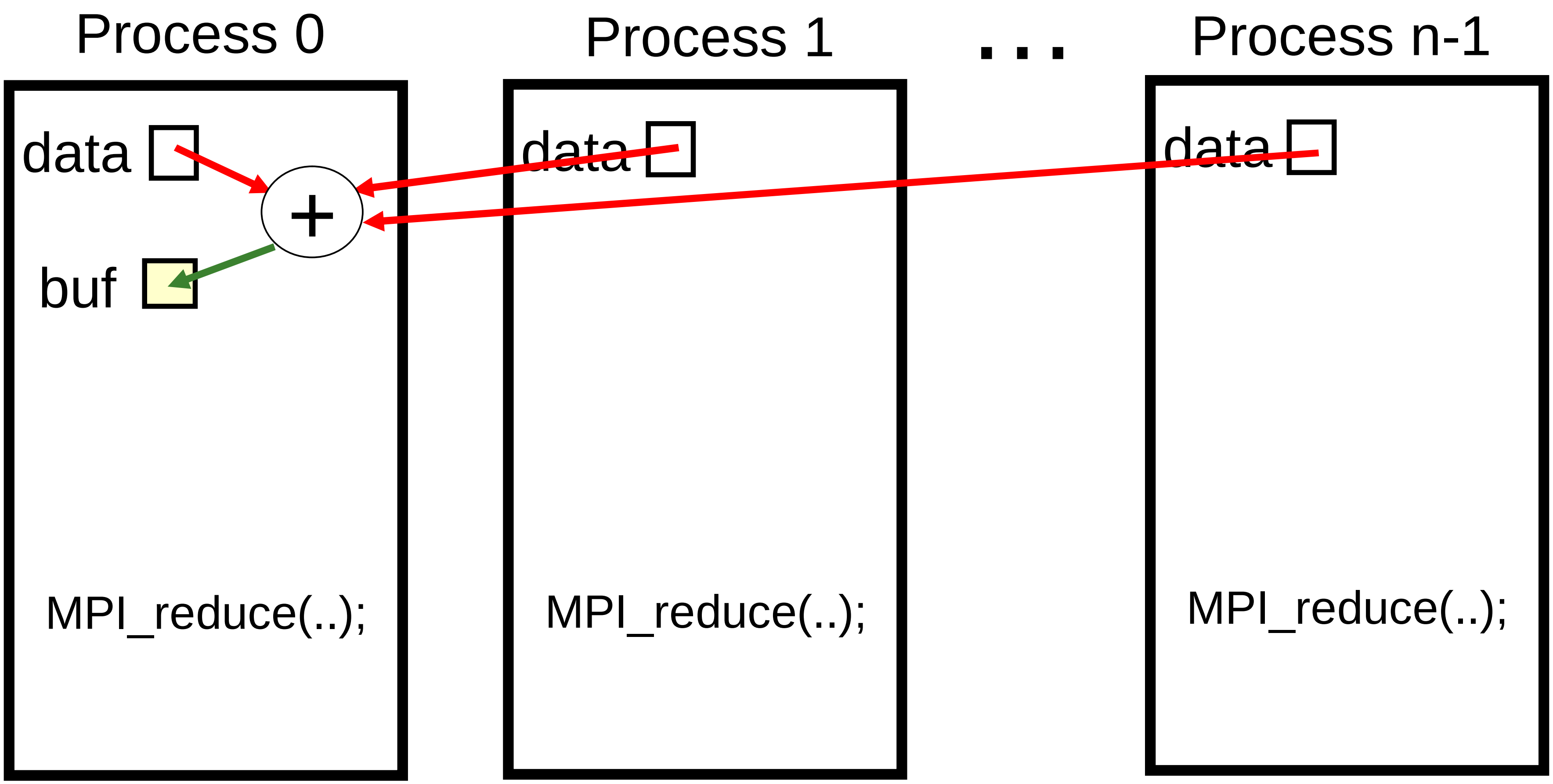

Process 0
Process 1
. . .
Process n-1
data
buf
data
data
MPI_scatter(..);
MPI_scatter(..);
MPI_scattert(..);

Gather



Process 0
Process 1
. . .
Process n-1
data
buf
MPI_gather(..);
data
MPI_gather(..);
data
MPI_gather(..);

Reduce



Process 0
Process 1
. . .
Process n-1
data
+
buf
MPI_reduce(..);
data
MPI_reduce(..);
data
MPI_reduce(..);

MPI Reduce

MPI_Reduce (void *sbuf, void *rbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

- Operations: MPI_SUM, MPI_MAX
- Reduction includes value coming from root

Gather Example

```
int data[10];  /*data to be gathered from processes*/
.
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
   MPI_Comm_size(MPI_COMM_WORLD,&grp_size);
   buf = (int *)malloc(grp_size*10*sizeof(int));
}
MPI_Gather(data,10,MPI_INT,buf,grp_size*10,MPI_IN
   T,0,MPI_COMM_WORLD);
```