Introduction to the Message Passing Interface (MPI)

Part II

True/False

1. Each MPI process will have their own private memory
2. MPI programs can be executed on SMP machines
3. Execution of MPI programs happen based on the order of the rank (ID of MPI process)

MPI Implementation

Goals and Scope of MPI

- ★ to provide a message-passing interface
- ★ to provide source-code portability
- ★ allow efficient implementations
- ★ great deal of functionality
- ★ support for heterogeneous parallel architectures

MPI Implementation

Header File

- #include <mpi.h>

MPI Function Format

- error = MPI_Xxxxxx(parameter, ...);
- MPI_Xxxxxx(parameter, ...)
- MPI implementation is language independent
- available in several programming languages (C, Fortran, C++ [in MPI-2])
- MPI_ …… namespace is reserved for MPI constants and routines, i.e. application routines and variable names must not begin with MPI_

MPI Implementation

```
#include <mpi.h>
int main(int argc, char **argv)
{
   MPI_Init(&argc, &argv);
   ....
```

Initializing MPI

- ★ int MPI_Init(int *argc, char ***argv)
- ★ must be first MPI routine that should be called

Starting the MPI Program

- ★ Start mechanism is implementation specific
- ★ mpirun –np number_of_processes ./executable (most implementations)
- ★ mpiexec –n number_of_processes ./executable (with MPI-2 standard)

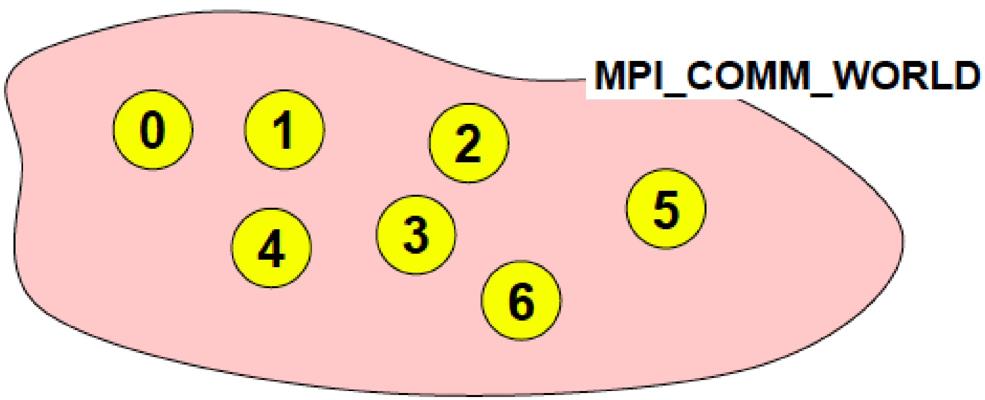
Finalizing MPI

- ★ parallel MPI processes exist after MPI_Finalize() call
- ★ MPI_Finalize() must be called last by all processes

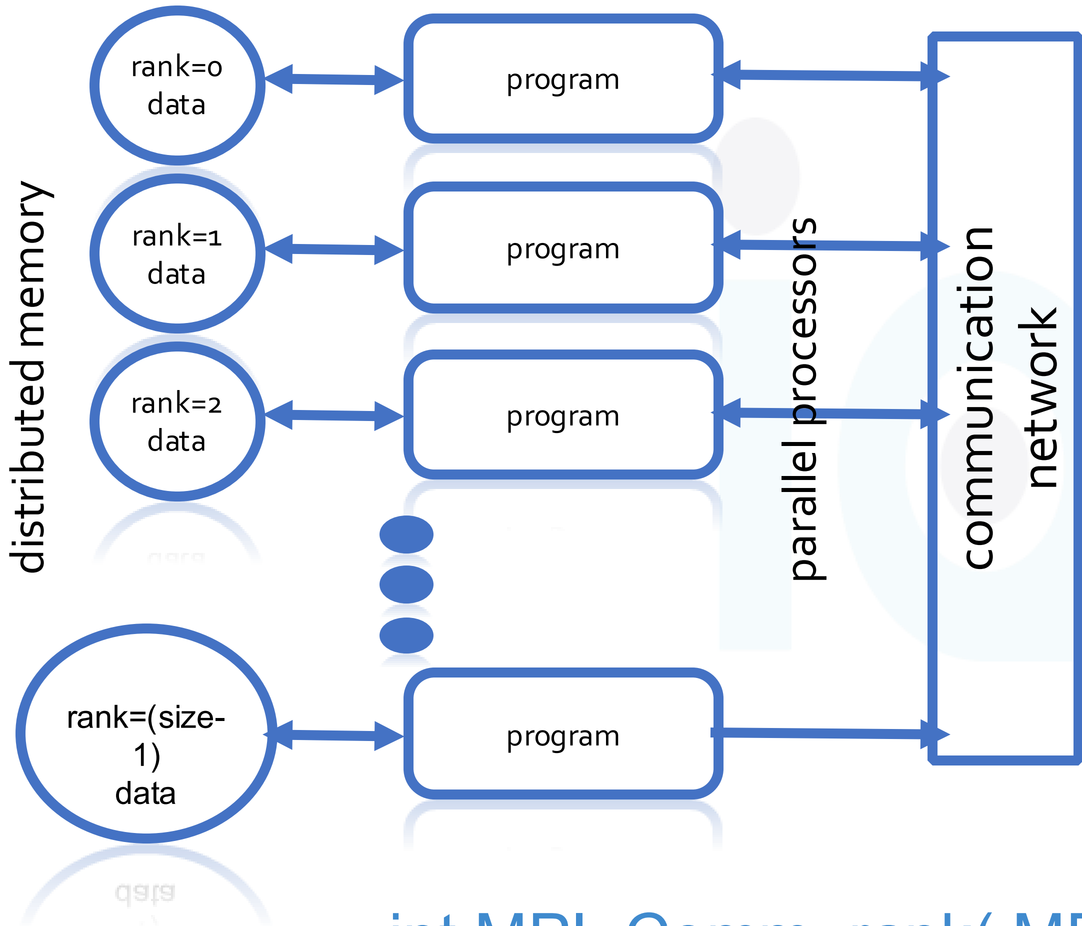
MPI Implementation

Communicator MPI_COMM_WORLD

- ★ all processes (= sub-programs) of one MPI program are combined
- ★ each process has its own rank in a communicator
  - starting with 0
  - ending with (size-1)



MPI Implementation



★"rank" value is returned by a special library routine

★MPI system of "size" processes is started by special MPI initialization (eg., mpirun, mpiexe)

★all distribution decisions and control of execution are made based on "rank"

int MPI_Comm_rank(MPI_Comm comm, int *rank)

MPI Implementation: Python

```
# install mpi4py
!pip install mpi4py
```

```
%%writefile helloworld.py
from mpi4py import MPI # Import mpi4py package
# Define a function
def main():
    '''A function to print the size and rank'''
    # creating the communicator
    comm = MPI.COMM_WORLD
    # Index of the process in the communicator
    rank = comm.Get_rank()
    # total number of processes in the communicator
    size = comm.Get_size()
    # Displaying the rank and size of a communicator
    print("Hello World: My rank is {} in the communicator of size {}".format(rank,size))

#call the function
main()
```

MPI Implementation

Messages

- ★ a message contains a number of elements of some particular data
- ★ E.g.

2345	654	96574	-12	7676

MPI datatypes

- ★ basic datatype
- ★ derived datatypes
- ★ derived datatypes can be created from basic or derived datatypes
- ★ C/C++ types are different from Fortran types
- ★No need to declare in Python
- ★ Datatype handles are used to describe the type of the data in the memory

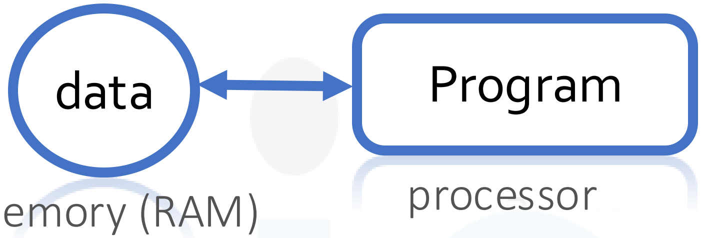
MPI Implementation

MPI Basic Datatypes - C

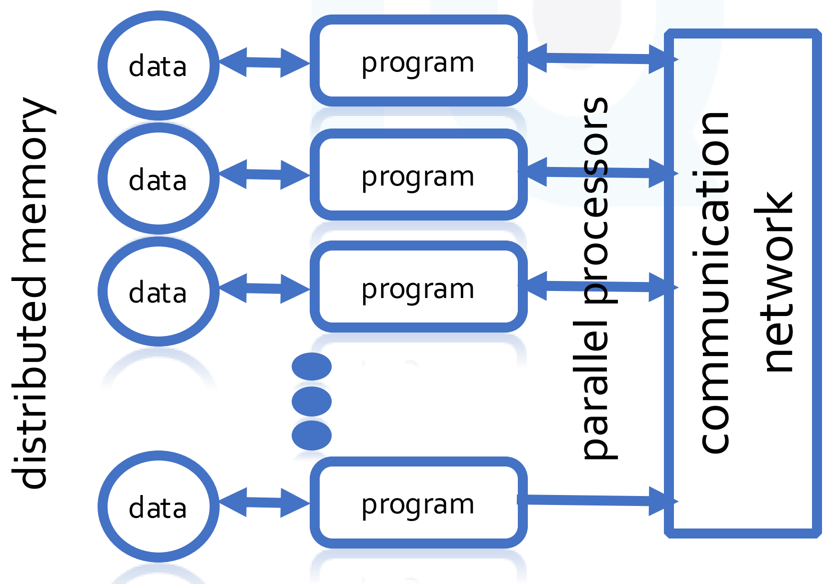
MPI Datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI Implementation

Sequential Programming



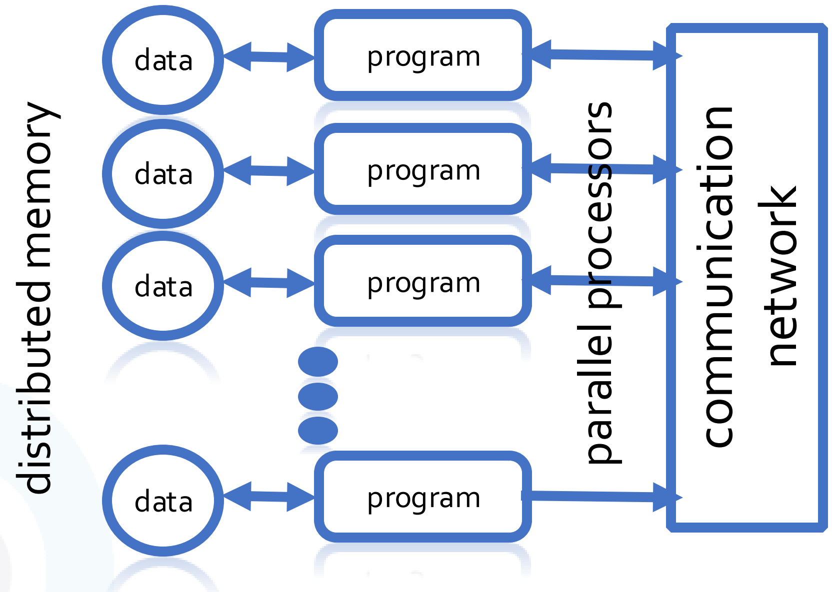
Message-Passing Programming



MPI Implementation

Each process in MPI program

- runs a (sub-)program
- standard Python, C or C++ or Fortran code
- in general, same on each processor

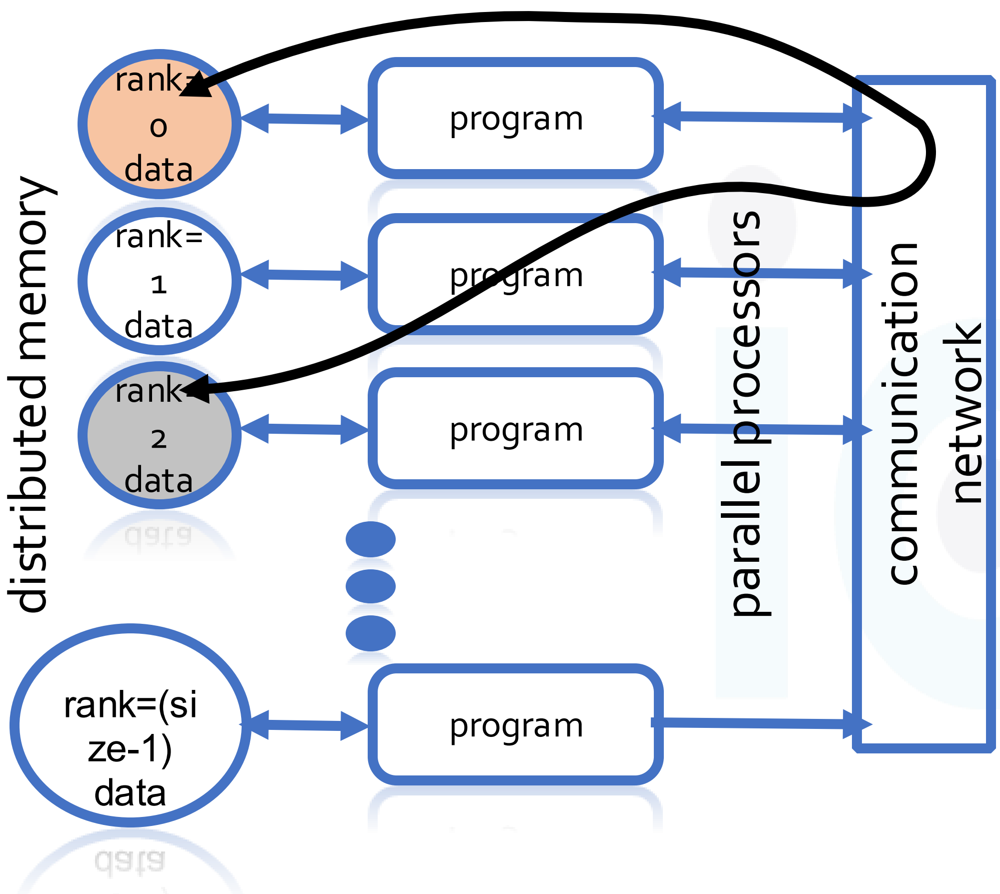


Variables in each process in MPI program

- same name across all processors
- different values/locations (distributed memory)
- all variables are private
- communication via MPI send and receive routines

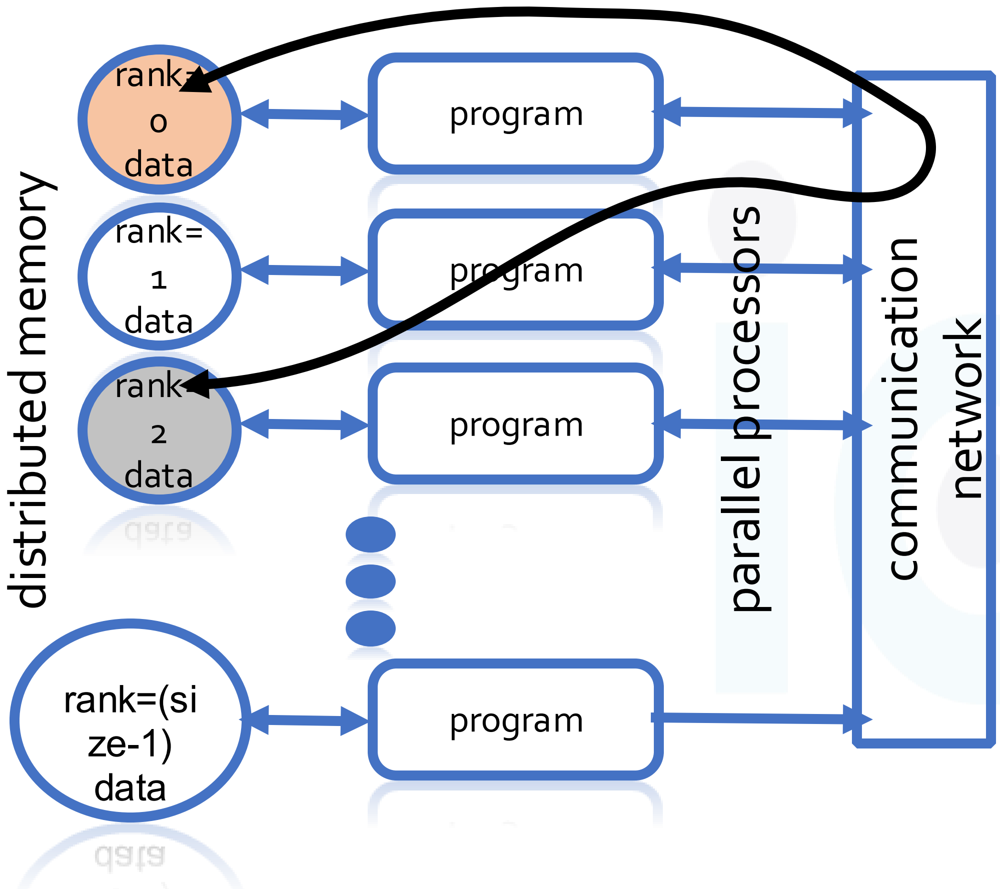
MPI Communications

MPI Implementation: MPI Communication



- ★"messages"are packets of data moving between MPI-nodes
- ★Necessary information for the message passing
  sending – receiving processes
  source – destination locations
  source – destination data type
  source – destination size

MPI Implementation: MPI Communication

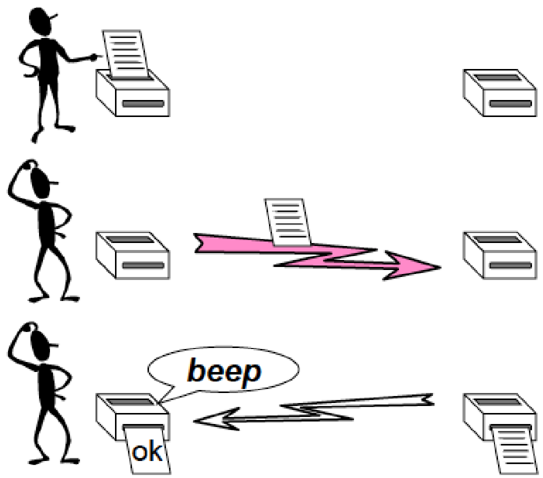


Point-to-Point Communication

- simplest form of message passing
- process “0” sends/receives a message to/from process “2”
- different types of send
  – synchronous send
  — buffered (asynchronous)

MPI Implementation: MPI Communication

Blocking Operations

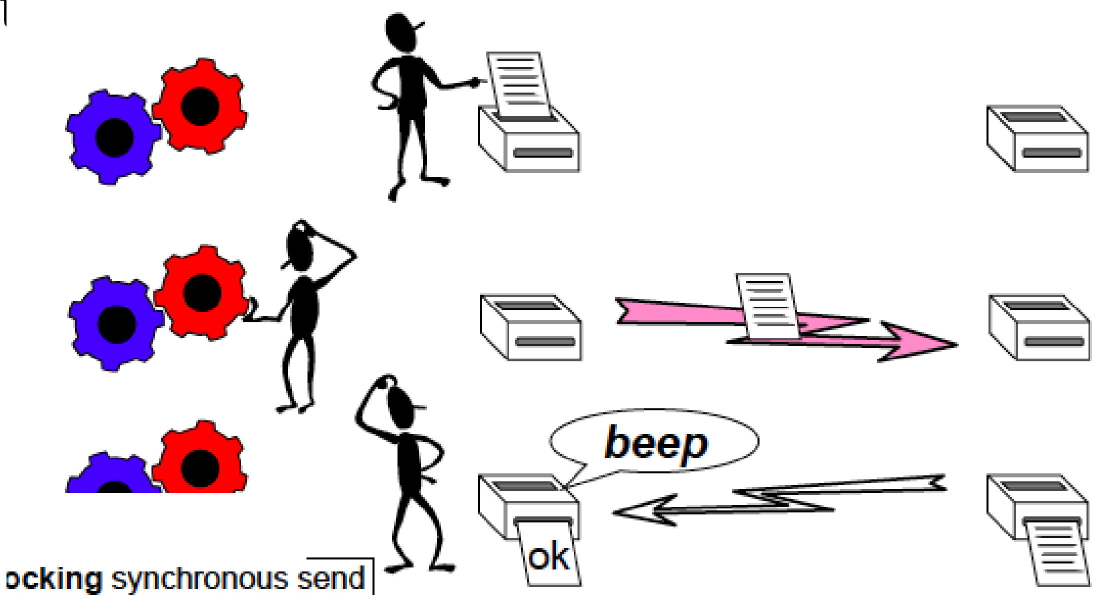


- ★ Operations are local activities, e.g.,
  - sending (a message)
  - receiving (a message)
- ★ Some operations may block until another process acts
  - synchronous send operation blocks until receive is posted
  - receive operation blocks until message is sent
- ★ Relates to the completion of an operation
- ★ Blocking subroutine returns only when the operation has completed

MPI Implementation: MPI Communication

Non-Blocking Operations

- ★ returns immediately and allow the process to perform other work
- ★ at some later time the process must test or wait for the complet...

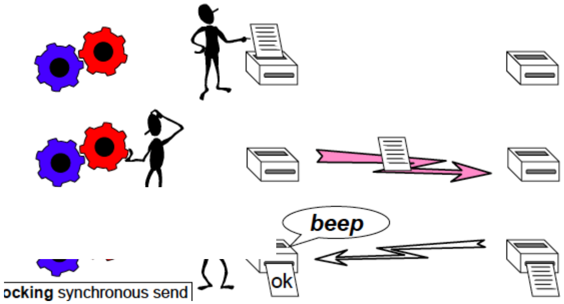


MPI Implementation: MPI Communication

Non-Blocking Operations (cont'd)

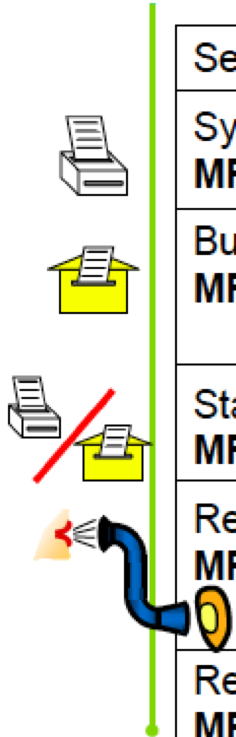
- ★ all non-blocking operations must have matching wait (or test) operations.(Some system or application resources can be freed only when the nonblocking operation is completed.)
- ★ a non-blocking operation immediately followed by a matching wait is equivalent to a blocking operation.
- ★ Non-blocking operations are not the same as sequential subroutine calls

  - Operations may continue while the process executes the next statements!



MPI Implementation: MPI Communication

Communication Modes - Definitions



Sender mode	Definition	Notes
Synchronous send **MPI_SSEND**	Only completes when the receive has started	
Buffered send **MPI_BSEND**	Always completes (unless an error occurs), irrespective of receiver	needs application-defined buffer to be declared with MPI_BUFFER_ATTACH
Standard send **MPI_SEND**	Either synchronous or buffered	uses an internal buffer
Ready send **MPI_RSEND**	May be started **only** if the matching receive is already posted!	highly dangerous!
Receive **MPI_RECV**	Completes when a message has arrived	same routine for all communication modes

https://iamsorush.com/posts/mpi-send-types/

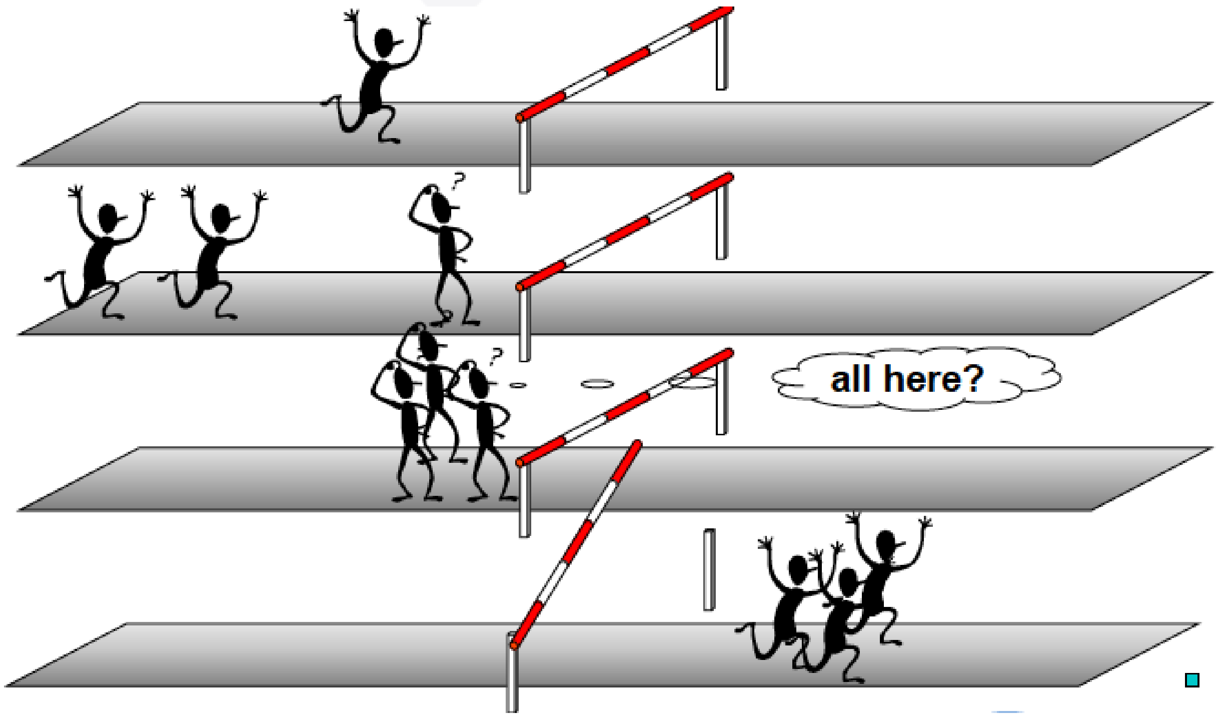
MPI Implementation: MPI Communication

Barrier

MPI Implementation: MPI Communication

Barriers

- ★ synchronize processes



MPI Implementation: MPI Communication

Collective communication

- ★ communications involving a group of processes
- ★ called by all processes in a communicator
- ★ Examples:
  - Barrier synchronization
  - Broadcast, scatter, gather
  - Global sum, global maximum, etc.

Characteristics of collective communication

- ★ collective action over a communicator
- ★ all process in the communicator must communicate, i.e. all process must call the collective routine
- ★ synchronization may or may not occur, therefore all processes must be able to start the collective routine:
- ★ Receive buffers must have exactly the same size as send buffers

MPI Implementation: MPI Communication

Barrier Synchronization

- ★ int MPI_Barrier(MPI_Comm comm)
- ★ MPI_Barrier is normally never needed
  - all synchronization is done automatically by the data communication:
    - a process cannot continue before it has the data that it needs
  - if used for debugging:
    - guarantee, that it is removed in production
  - for profiling: to separate time measurement of:
    - load imbalance of computation: MPI_Wtime(); MPI_Barrier(); MPI_Wtime()
    - communication epochs: MPI_Wtime(); MPI_Allreduce(); …; MPI_Wtime()

Collective Communications

MPI Implementation: MPI Communication

Collective Communications

★ collective communication routines are higher level routines

★ several processes are involved at a time

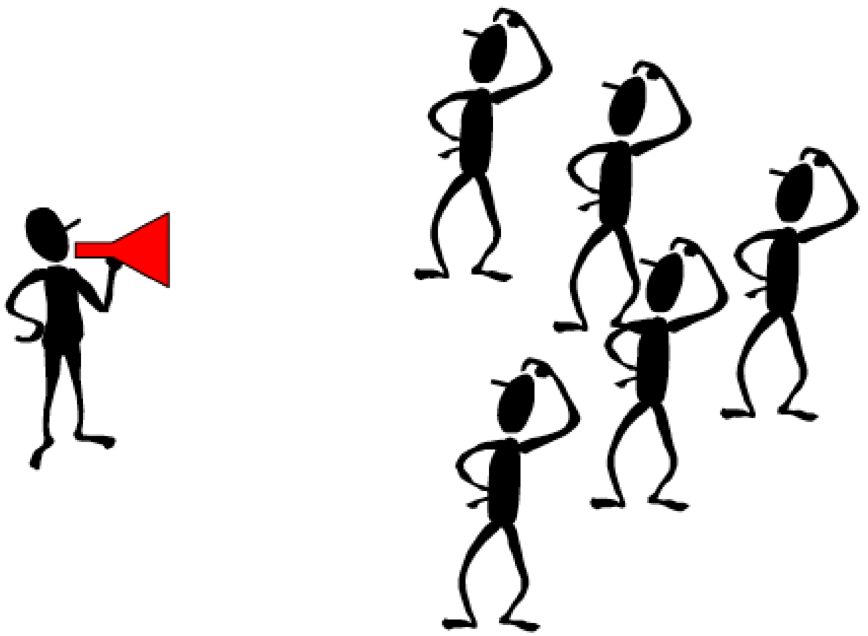
★ may allow optimized internal implementations, e.g., tree based algorithms

★ can be built out of point-to-point communications

MPI Implementation: MPI Communication

Broadcast

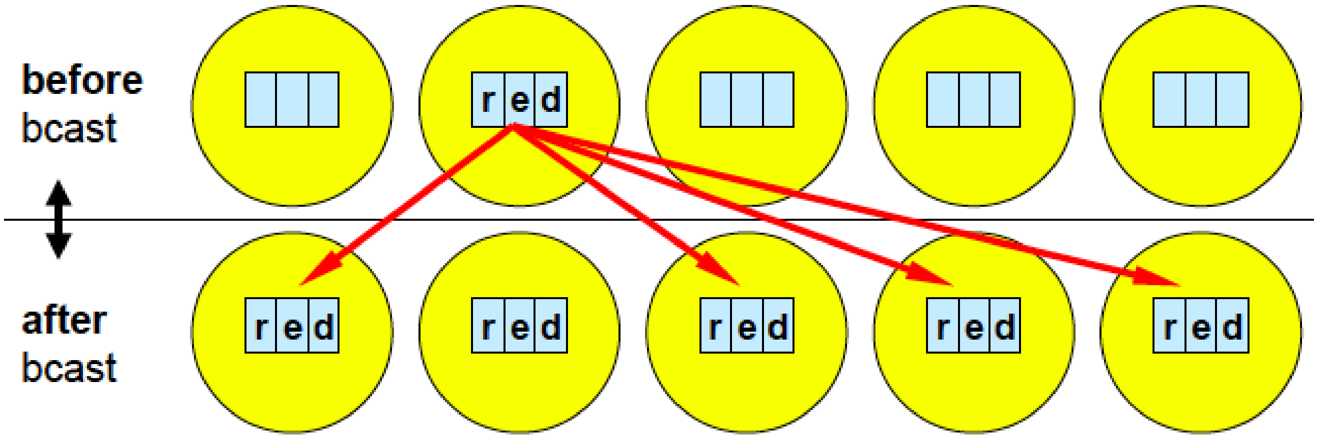
- one-to-many communication



MPI Implementation: MPI Communication

Broadcast

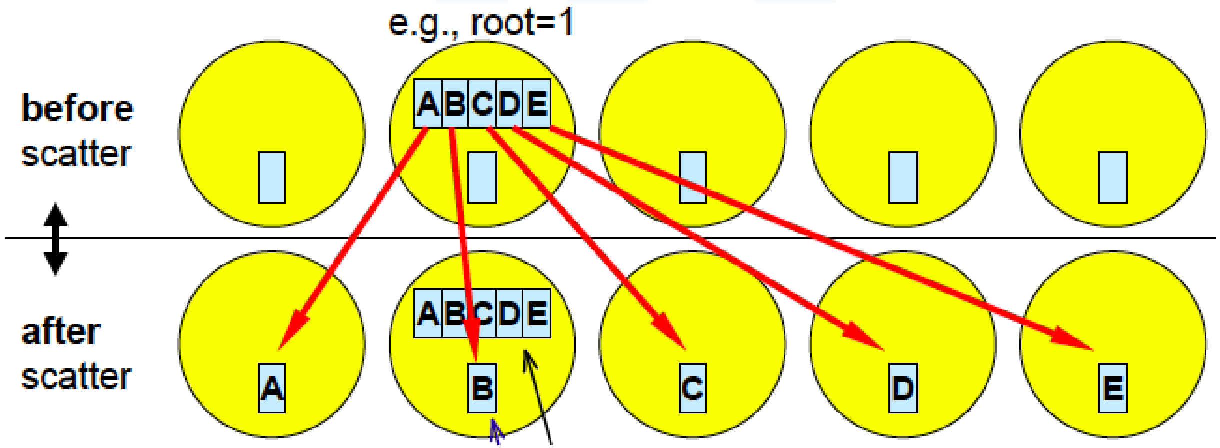
- ★ int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)



- ★ MPI-3.0 :> int MPI_Ibcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm, MPI_Request *request)

Scatter

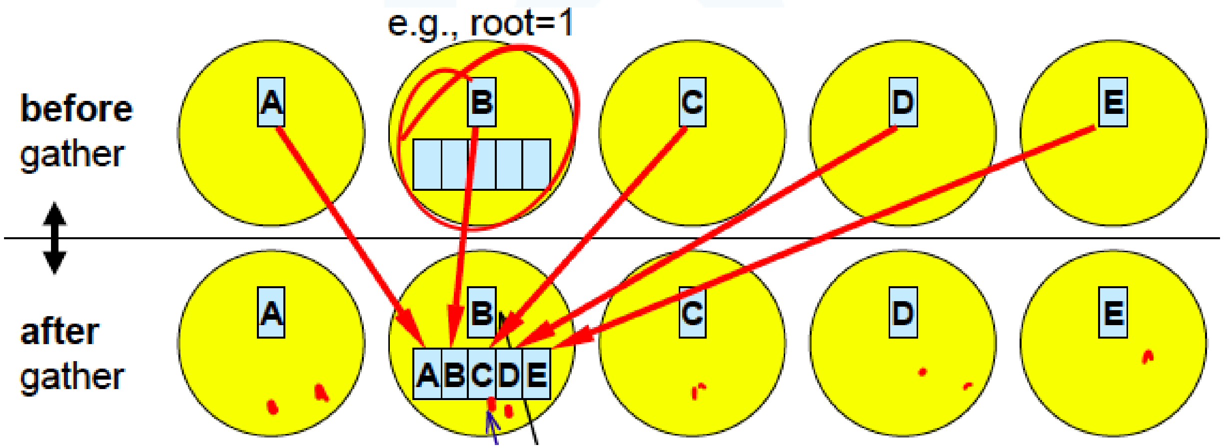
★ int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)



MPI Implementation: MPI Communication

Gather

- int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

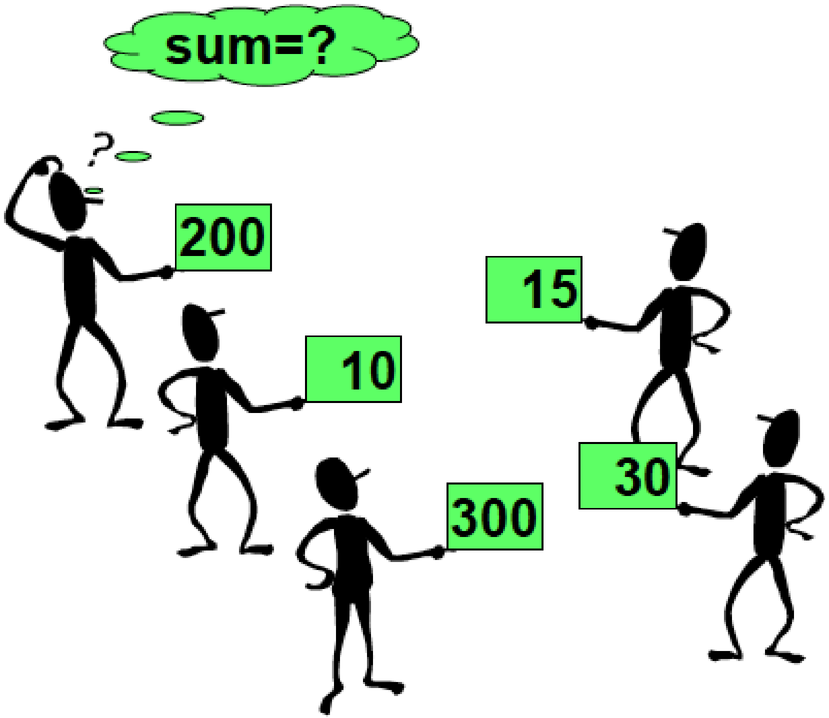


MPI Reduction

MPI Implementation: MPI Communication

Reduction Operations

★ combine data from several processes to produce a single result



MPI Implementation: MPI Communication

Global Reduction Operations

- To perform a global reduce operation across all members of a group
- $d_0 \circ d_1 \circ d_2 \circ d_3 \circ \ldots \circ d_{s-2} \circ d_{s-1}$
  - d_i = data in process rank i
    - single variable, or
    - vector
  - $\circ$ = associative operation
  - Example:
    - global sum or product
    - global maximum or minimum
    - global user-defined operation
- floating point rounding may depend on usage of associative law:
  - $[(d_0 \circ d_1) \circ (d_2 \circ d_3)] \circ [\ldots \circ (d_{s-2} \circ d_{s-1})]$
  - $((((((d_0 \circ d_1) \circ d_2) \circ d_3) \circ \ldots) \circ d_{s-2}) \circ d_{s-1})$

MPI Implementation: MPI Communication

Example of Global Reduction

- ★ global integer sum
- ★ sum of all inbuf values should be returned in resultbuf
- ★ MPI_Reduce(&inbuf, &resultbuf, 1, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD)
- ★ result is placed only in resultbuf at the root process

MPI Implementation: MPI Communication

Predefined Reduction Operation Handles

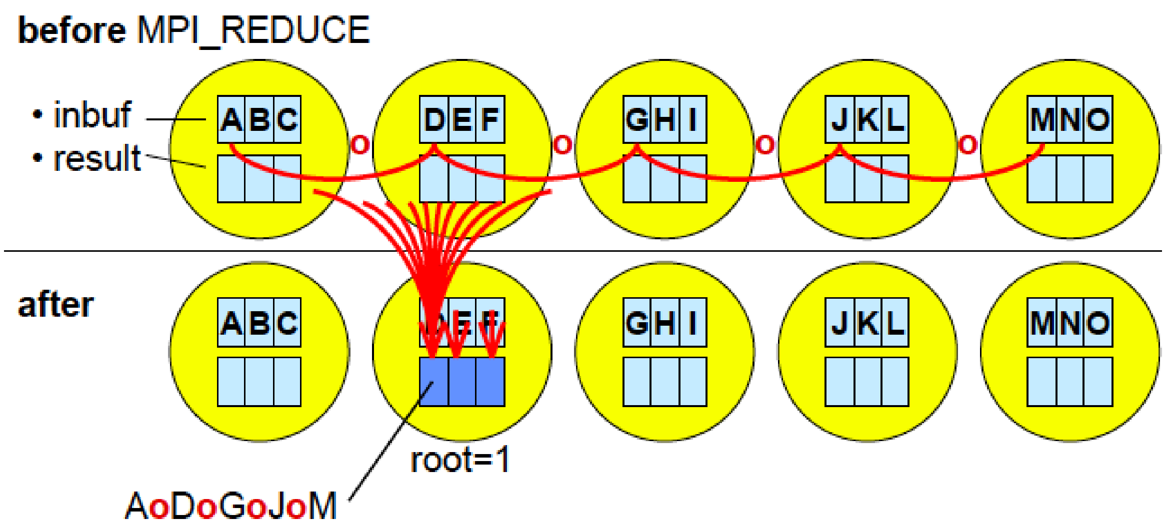
Predefined operation handle	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location of the maximum
MPI_MINLOC	Minimum and location of the minimum

MPI Implementation: MPI Communication

MPI_REDUCE

★ reduces values on all processes to a single value

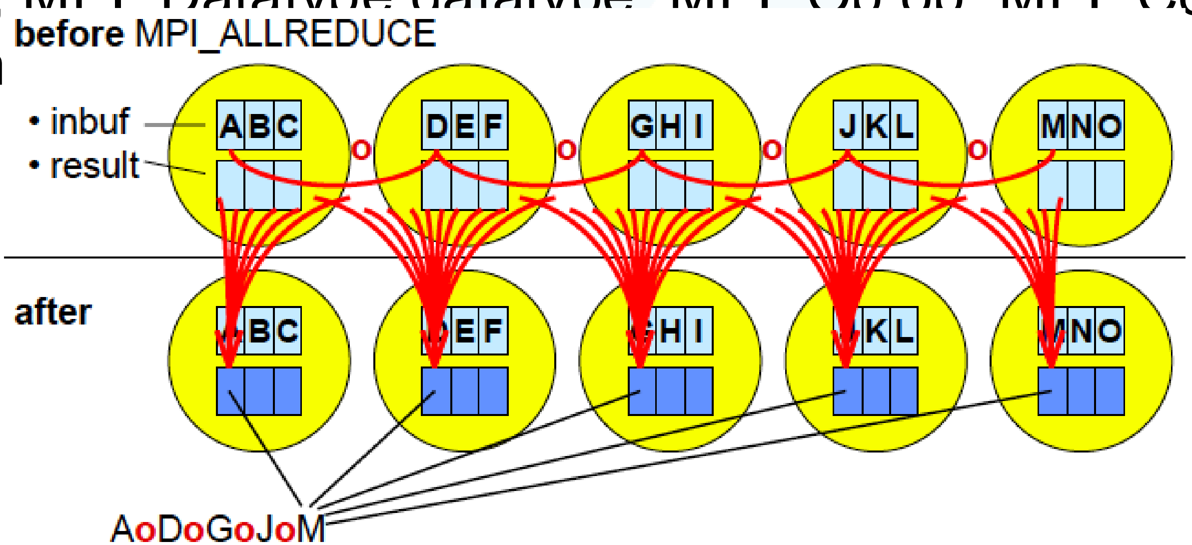
★ int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)



MPI Implementation: MPI Communication

MPI_ALLREDUCE

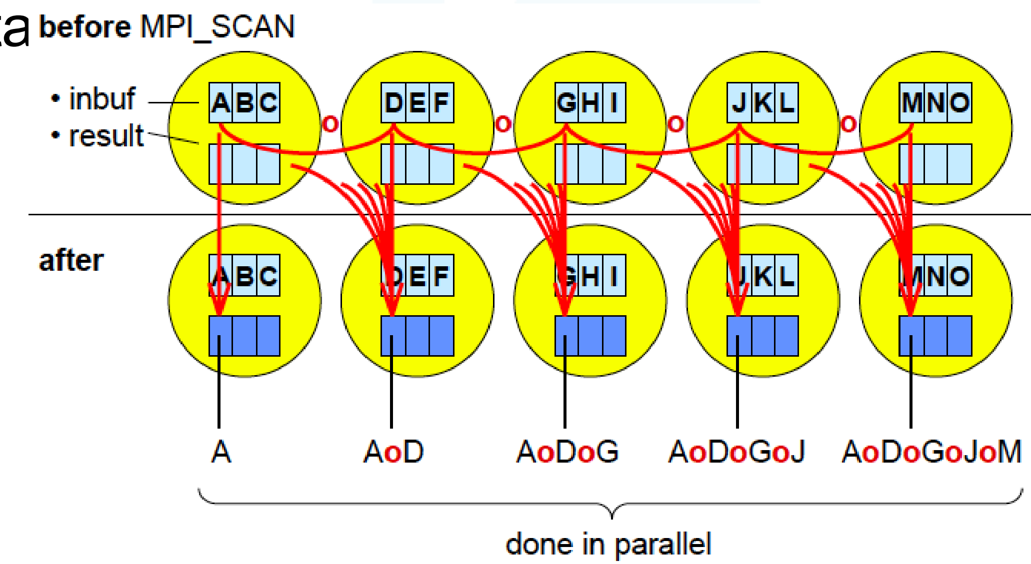
- ★ reduces values on all processes to a single value in all processes
- ★ int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm



MPI Implementation: MPI Communication

MPI_SCAN

- ★ computes the scan (partial reductions) of data on a collection of processes
- ★ int MPI_Scan(const void *sendbuf, void *recvbuf, int count, MPI_Data n comm)



MPI Implementation: MPI Communication

User-Defined Reduction Operations

- ★ operator handles
  - predefined – see table
  - user-defined
- ★ int MPI_Op_create(MPI_User_function *user_fn, int commute, MPI_Op *op)

MPI Implementation: MPI Communication

Other MPI features:

- ★ Point-to-point
  - MPI_Sendrecv & MPI_Sendrecv_replace
  - Null processes, MPI_PROC_NULL
  - MPI_Pack & MPI_Unpack
  - MPI_Probe: check length (tag, source rank) before calling MPI_Recv
  - MPI_Iprobe: check whether a message is available
  - MPI_Request_free, MPI_Cancel
  - MPI_BOTTOM (in point-to-point and collective communication)
- ★ Collective Operations
  - MPI_Allgather

A	⇨	A	B	C
B		A	B	C
C		A	B	C

  - MPI_Alltoall

A1	B1	C1	⇨	A1	A2	A3
A2	B2	C2		B1	B2	B3
A3	B3	C3		C1	C2	C3

  - MPI_Reduce_scatter

A1	B1	C1	⇨	Σ A
A2	B2	C2		Σ B
A3	B3	C3		Σ C

  - MPI_..........v (Gatherv, Scatterv, Allgatherv, Alltoallv)

MPI Implementation

MPI provider

- ★ vendor of your supercomputers
- ★ network provider (e.g. with MYRINET)
- ★ MPICH – the public domain MPI library from Argonne
- ★ Recent standard MPI-3.0 and MPI-4.1(www.mpi-forum.org)
- ★ Propose your ideas to next release?