Distributed Memory

→ Each processing element (P) has its separate main memory block (M)

→ Data exchange is achieved through message passing over the network

→ Message passing could be either explicit (MPI) or implicit (PGAS)

→ Programs typically implemented as a set of OS entities with own (virtual) address spaces – *processes*

→ No shared variables

- →No data races
- →Explicit synchronisation mostly unneeded
  - →Results as side effect of the send-receive semantics

Processes

- **A process is a running in-memory instance of an executable file**
  - → Executable code, e.g., binary machine instructions
  - → One or more threads of execution sharing memory address space
  - → Memory: data, heap, stack, processor state (CPU registers and flags)
  - → Operating system context (e.g. signals, I/O handles, etc.)
  - → PID
- **Isolation and protection**
  - → A process cannot interoperate with other processes or access their context (even on the same node) without the help of the operating system
  - → No direct inter-process data exchange (isolated/virtual address spaces)
  - → No direct inter-process synchronisation

General Structure of an MPI Program

- **Start-up, initialisation, finalisation, and shutdown – C**

C

```
① #include <mpi.h>

② int main(int argc, char **argv)
  {
    … some code …
③   MPI_Init(&argc, &argv);


④   … computation & communication …


⑤   MPI_Finalize();
    … wrap-up …
⑥   return 0;
  }
```

① Inclusion of the MPI header file

② Pre-initialisation mode: uncoordinated
- **No MPI function calls allowed with few exceptions**
- **All program instances run exactly the same code**

③ Initialisation of the MPI environment
Implicit synchronisation

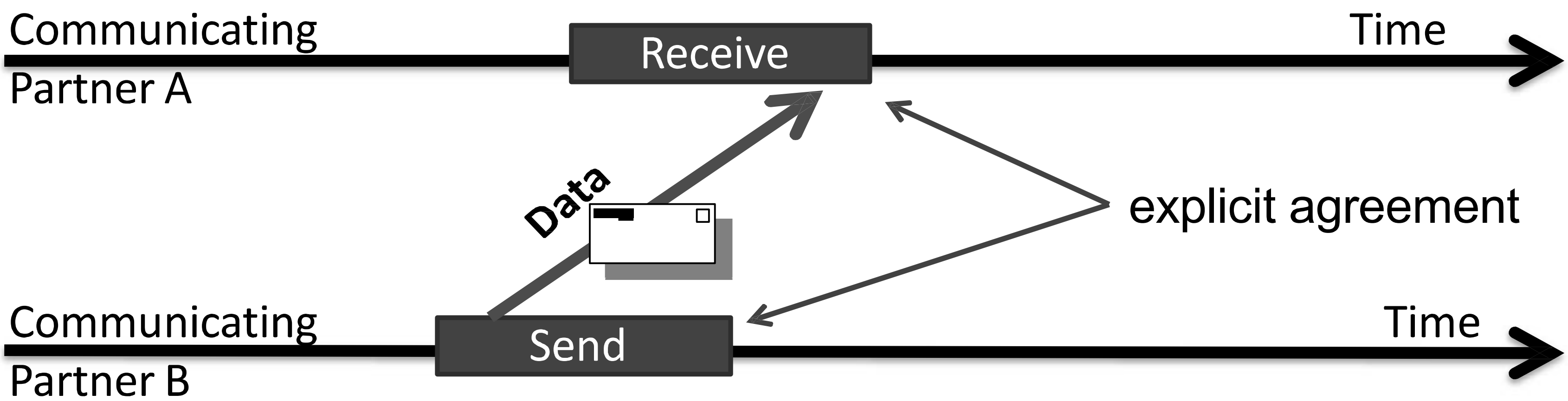
④ Parallel MPI code Typically computation and communication

⑤ Finalisation of the MPI environment
Internal buffers are flushed

⑥ Post-finalisation mode: uncoordinated
- **No MPI function calls allowed with few exceptions**

Message Passing

- **The goal is to enable communication between processes that share no memory space**



- **Explicit message passing requires:**
  - → Send and receive primitives (operations)
  - → Known addresses of both the sender and the receiver
  - → Specification of what has to be sent/received

MPI Datatypes

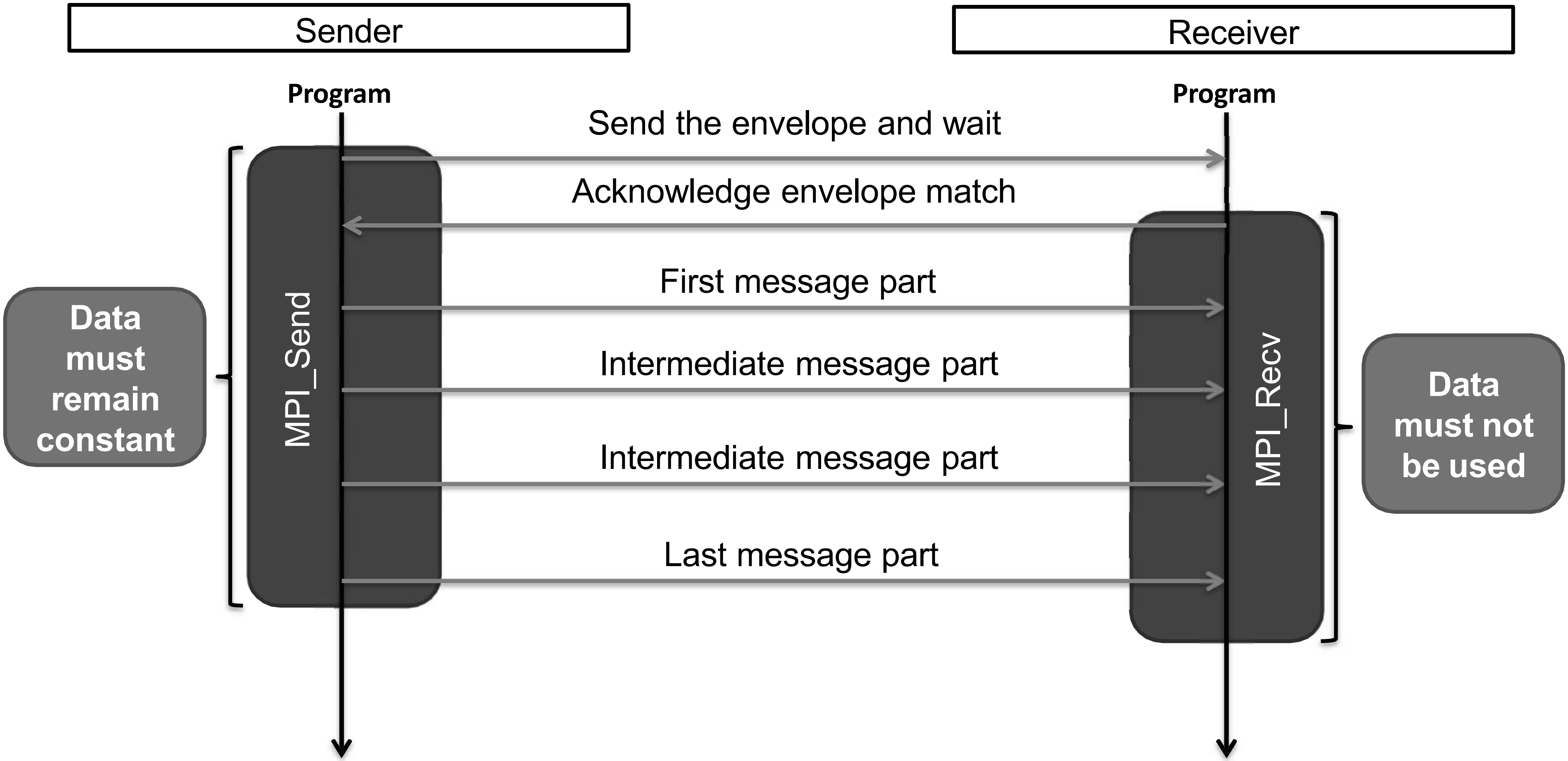
- **MPI is a library – it cannot infer the type of elements in the supplied buffer at run time and that's why it has to be told what it is**

- **MPI datatypes tell MPI how to:**
  - → read binary values from the send buffer
  - → write binary values into the receive buffer
  - → correctly apply value alignments
  - → convert between machine representations in heterogeneous environments

- **MPI datatype must match the language type(s) in the data buffer**
- **MPI datatypes are handles and cannot be used to declare variables**

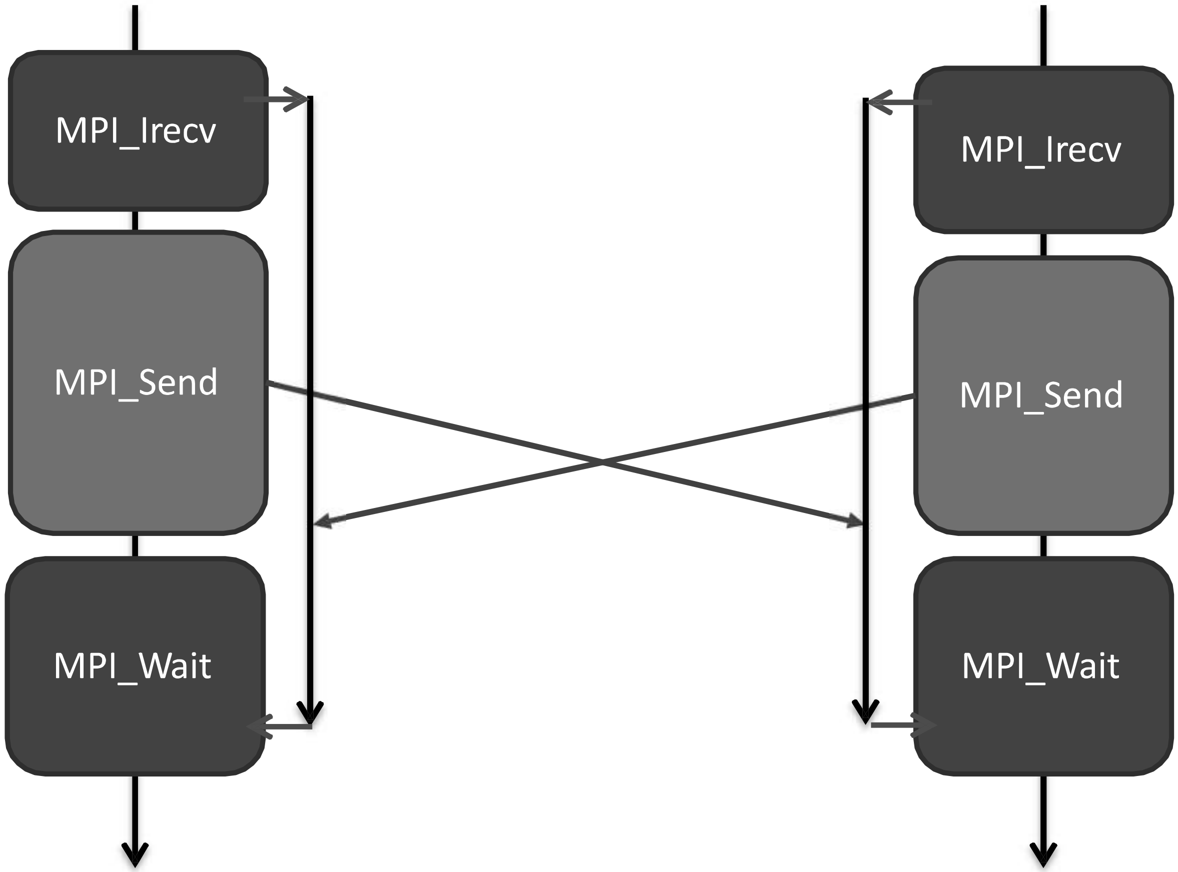
Blocking Calls

- **Blocking send (w/o buffering) and receive calls:**



Deadlock Prevention

- **Non-blocking operations can be used to prevent deadlocks in symmetric code:**



- **That is how MPI_Sendrecv is usually implemented**

Send Modes

- **Standard mode**
  → The call blocks until the message has either been transferred or copied to an internal buffer for later delivery
- **Synchronous mode**
  → The call blocks until a matching receive has been posted and the message reception has started
- **Buffered mode**
  → The call blocks until the message has been copied to a user-supplied buffer. Actual transmission may happen at a later point
- **Ready mode (don’t use!)**
  → The operation succeeds only if a matching receive has already been posted. Behaves as standard send in every other aspect

Send Modes

- **Call names:**
  - → **MPI_Send** blocking standard send
  - → **MPI_Isend** non-blocking standard send
  - → **MPI_Ssend** blocking synchronous send
  - → **MPI_Issend** non-blocking synchronous send
  - → **MPI_Bsend** blocking buffered send
  - → **MPI_Ibsend** non-blocking buffered send
  - → **MPI_Rsend** blocking ready-mode send
  - → **MPI_Irsend** non-blocking ready-mode send
- **Buffered operations require an explicitly provided user buffer**
  - → **MPI_Buffer_attach (void *buf, int size)**
  - → **MPI_Buffer_detach (void *buf, int *size)**
  - → Buffer size must account for the envelope size (**MPI_BSEND_OVERHEAD**)

Common Pitfalls – C/C++

- **Do not pass pointers to pointers in MPI calls**

```
void func (int scalar)
{
  MPI_Send(&scalar, MPI_INT, 1, …

void func (int& scalar)
{
  MPI_Send(&scalar, MPI_INT, 1, …

void func (int *scalar)
{
  MPI_Send(scalar, MPI_INT, 1, …

void func (int *array)
{
  MPI_Send(array, MPI_INT, 5, …
  … or …
  MPI_Send(&array[0], MPI_INT, 5, …
```

Common Pitfalls – C/C++

- **Use flat multidimensional arrays;** arrays of pointers do not work

```
// Static arrays are OK
int mat2d[10][10];
MPI_Send(&mat2d, MPI_INT, 10*10, …

// Flat dynamic arrays are OK
int *flat2d = new int[10*10];
Fill array flat2d …
MPI_Send(flat2d, MPI_INT, 10*10, …

// DOES NOT WORK
int **p2d[10] = new int*[10];
for (int i = 0; i < 10; i++)
   p2d[i] = new int[10];
MPI_Send(p2d, MPI_INT, 10*10, …
… or …
MPI_Send(&p2d[0][0], MPI_INT, 10*10, …
```

MPI has no way to know that there is a hierarchy of pointers

Message Passing: Summary

- No notion of global data
- Data communication is done by explicit message passing
  - expensive performance-wise
- Trade-off between:
  - one-copy data
    - *more communication is needed, less consistency issues*
  - local data replication
    - *less communication, consistency is problematic*
- Techniques to improve performance:
  - replicate read-only data
  - computation and communication overlapping
  - message aggregation

MPI Quick Reference in C

#include <mpi.h>

Environmental Management:

int **MPI_Init**(int *argc, char ***argv)

int **MPI_Finalize**(void)

int **MPI_Initialized**(int *flag)

int **MPI_Finalized**(int *flag)

int **MPI_Comm_size**(MPI_Comm comm, int *size)

int **MPI_Comm_rank**(MPI_Comm comm, int *rank)

int **MPI_Abort**(MPI_Comm comm, int errorcode)

double **MPI_Wtime**(void)

double **MPI_Wtick**(void)

Blocking Point-to-Point-Communication:

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

Related: MPI_Bsend, MPI_Ssend, MPI_Rsend

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

int **MPI_Probe** (int source, int tag, MPI_Comm comm, MPI_Status *status)

int **MPI_Get_count** (const MPI_Status *status, MPI_Datatype datatype, int *count)

Related: MPI_Get_elements

int **MPI_Sendrecv** (const void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

int **MPI_Sendrecv_replace** (void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

int **MPI_Buffer_attach** (void *buffer, int size)

int **MPI_Buffer_detach** (void *buffer_addr, int *size)

Non-Blocking Point-to-Point-Communication:

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

Related: MPI_Ibsend, MPI_Issend, MPI_Irsend

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

int **MPI_Iprobe** (int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)

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

int **MPI_Test** (MPI_Request *request, int *flag, MPI_Status *status)

int **MPI_Waitall** (int count, MPI_Request request_array[], MPI_Status status_array[])

Related: MPI_Testall

int **MPI_Waitany** (int count, MPI_Request request_array[], int *index, MPI_Status *status)

Related: MPI_Testany

int **MPI_Waitsome** (int incount, MPI_Request request_array[], int *outcount, int index_array[], MPI_Status status_array[])

Related: MPI_Testsome,

int **MPI_Request_free** (MPI_Request *request)

Related: MPI_Cancel

int **MPI_Test_cancelled** (const MPI_Status *status, int *flag)

Collective Communication:

int **MPI_Barrier** (MPI_Comm comm)

int **MPI_Bcast** (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

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

int **MPI_Gatherv** (const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, const int recvcount_array[], const int displ_array[], MPI_Datatype recvtype, int root, MPI_Comm comm)

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

int **MPI_Scatterv** (const void *sendbuf, const int sendcount_array[], const int displ_array[], MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

int **MPI_Allgather** (const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

Related: MPI_Alltoall

int **MPI_Allgatherv** (const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, const int recvcount_array[], const int displ_array[], MPI_Datatype recvtype, MPI_Comm comm)

Related: MPI_Alltoallv

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

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

Related: MPI_Scan, MPI_Exscan

int **MPI_Reduce_scatter** (const void *sendbuf, void *recvbuf, const int recvcount_array[], MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

int **MPI_Op_create** (MPI_User_function *func, int commute, MPI_Op *op)

int **MPI_Op_free** (MPI_Op *op)

Derived Datatypes:

int **MPI_Type_commit** (MPI_Datatype *datatype)

int **MPI_Type_free** (MPI_Datatype *datatype)

int **MPI_Type_contiguous** (int count, MPI_Datatype oldtype, MPI_Datatype *newtype) int **MPI_Type_vector** (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Type_indexed** (int count, const int blocklength_array[], const int displ_array[], MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Type_create_struct** (int count, const int blocklength_array[], const MPI_Aint displ_array[], const MPI_Datatype oldtype_array[], MPI_Datatype *newtype)

int **MPI_Type_create_subarray** (int ndims, const int size_array[], const int subsize_array[], const int start_array[], int order, MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Get_address** (const void *location, MPI_Aint *address)

int **MPI_Type_size** (MPI_Datatype *datatype, int *size)

int **MPI_Type_get_extent** (MPI_Datatype datatype, MPI_Aint *lb, MPI_Aint *extent)

int **MPI_Pack** (const void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outcount, int *position, MPI_Comm comm)

int **MPI_Unpack** (const void *inbuf, int insize, int *position, void *outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)

int **MPI_Pack_size** (int incount, MPI_Datatype datatype, MPI_Comm comm, int *size)

Related: MPI_Type_create_hvector, MPI_Type_create_hindexed, MPI_Type_create_indexed_block, MPI_Type_create_darray, MPI_Type_create_resized, MPI_Type_get_true_extent, MPI_Type_dup, MPI_Pack_external, MPI_Unpack_external, MPI_Pack_external_size

Groups and Communicators:

int **MPI_Group_size** (MPI_Group group, int *size)

int **MPI_Group_rank** (MPI_Group group, int *rank)

int **MPI_Comm_group** (MPI_Comm comm, MPI_Group *group)

int **MPI_Group_translate_ranks** (MPI_Group group1, int n, const int ranks1[], MPI_Group group2, const int ranks2[])

int **MPI_Group_compare** (MPI_Group group1, MPI_Group group2, int *result)

MPI_IDENT, MPI_COMGRUENT, MPI_SIMILAR, MPI_UNEQUAL

int **MPI_Group_union** (MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

Related: MPI_Group_intersection, MPI_Group_difference

int **MPI_Group_incl** (MPI_Group group, int n, const int ranks[], MPI_Group *newgroup)

Related: MPI_Group_excl

int **MPI_Comm_create** (MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)

int **MPI_Comm_compare** (MPI_Comm comm1, MPI_Comm comm2, int *result)

MPI_IDENT, MPI_COMGRUENT, MPI_SIMILAR, MPI_UNEQUAL

int **MPI_Comm_dup** (MPI_Comm comm, MPI_Comm *newcomm)

int **MPI_Comm_split** (MPI_Comm comm, int color, int key, MPI_Comm *newcomm)

int **MPI_Comm_free** (MPI_Comm *comm)

Topologies:

int **MPI_Dims_create** (int nnodes, int ndims, int dims[])

int **MPI_Cart_create** (MPI_Comm comm_old, int ndims, const int dims[], const int periods[], int reorder, MPI_Comm *comm_cart)

int **MPI_Cart_shift** (MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)

int **MPI_Cartdim_get** (MPI_Comm comm, int *ndim)

int **MPI_Cart_get** (MPI_Comm comm, int maxdims, int dims[], int periods[], int coords[])

int **MPI_Cart_rank** (MPI_Comm comm, const int coords[], int *rank)

int **MPI_Cart_coords** (MPI_Comm comm, int rank, int maxdims, int coords[])

int **MPI_Cart_sub** (MPI_Comm comm_old, const int remain_dims[], MPI_Comm *comm_new)

int **MPI_Cart_map** (MPI_Comm comm_old, int ndims, const int dims[], const int periods[], int *new_rank)

int **MPI_Graph_create** (MPI_Comm comm_old, int nnodes, const int index[], const int edges[], int reorder, MPI_Comm *comm_graph)

int **MPI_Graph_neighbors_count** (MPI_Comm comm, int rank, int *nneighbors)

int **MPI_Graph_neighbors** (MPI_Comm comm, int rank, int maxneighbors, int neighbors[])

int **MPI_Graphdims_get** (MPI_Comm comm, int *nnodes, int *nedges)

int **MPI_Graph_get** (MPI_Comm comm, int maxindex, int maxedges, int index[], int edges[])

int **MPI_Graph_map** (MPI_Comm comm_old, int nnodes, const int index[], const int edges[], int *new_rank)

int **MPI_Topo_test** (MPI_Comm comm, int *status)

Wildcards:

MPI_ANY_TAG, MPI_ANY_SOURCE

Basic Datatypes:

MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED

Predefined Groups and Communicators:

MPI_GROUP_EMPTY, MPI_GROUP_NULL, MPI_COMM_WORLD, MPI_COMM_SELF, MPI_COMM_NULL

Reduction Operations:

MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND, MPI_LOR, MPI_LXOR

Status Object:

status.MPI_SOURCE, status.MPI_TAG, status.MPI_ERROR