

# Lecture IN-2147 Parallel Programming

SoSe 2018

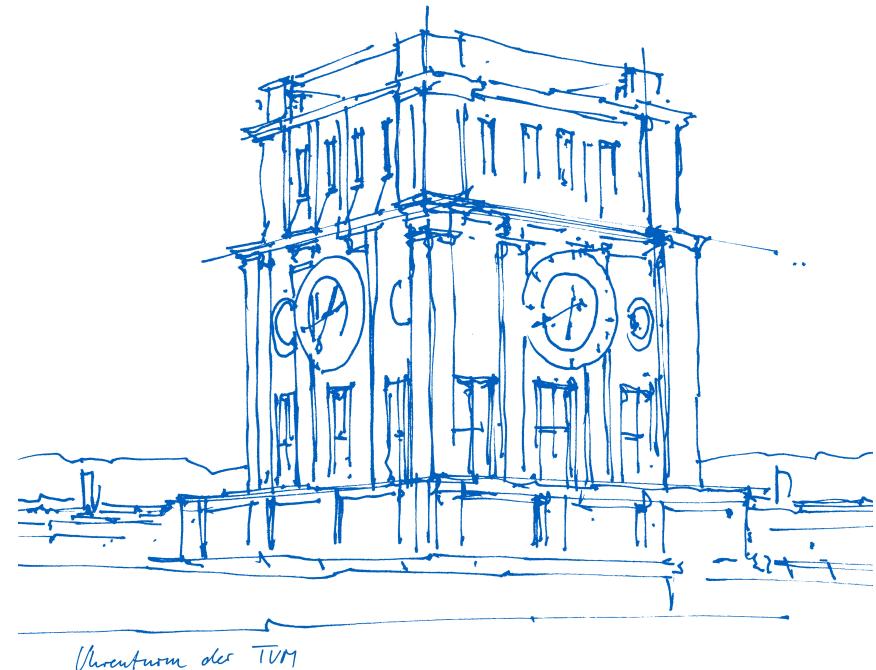
Martin Schulz

Exercises: Amir Raoofy

Technische Universität München

Fakultät für Informatik

Lecture 7:  
MPI Part 2  
Network Issues



# Administrative Topics

Exam:

July 24<sup>th</sup>, 2018, 16:00 to 17:30 in MW 2001

please double check in TUM-Online

Please register in TUM-Online by June 6<sup>th</sup>

Repetition Exam:

October 5<sup>th</sup>, 2018, 11:00 to 12:30 in MW 1801

please double check in TUM-Online

Please register in TUM-Online by September 24<sup>th</sup>

Lecture Evaluation (June 6<sup>th</sup> to June 20<sup>th</sup>)

You will get an online link, please fill it out

We will provide an extra 15min time on June 18<sup>th</sup>

# Summary From Last Time

## Distributed Memory Architectures

- Highly scalable to thousands (or more) nodes
- Example: SuperMUC
- Facilities are becoming more important as we scale

## HPC Applications are diverse and complex

- Wide range of spectrum
- Different reasons for HPC: faster solution, larger problem, ensembles, ...

## Programming distributed memory systems

- Individual processes communicating through the network
- Messaging libraries are necessary

## MPI as the most widely used HPC standard for message passing

- Currently in version 3.1 with wide range of functionality
- Continues to evolve through the open standardization body MPI Forum
- So far covered: boilerplate and basic point to point communication
- Central concept: communicators as communication contexts

# A First Simple Example: Value Forwarding



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

# A First Simple Example: Value Forwarding Sequential Program

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

# A First Simple Example: Value Forwarding MPI Boilerplate

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

# A First Simple Example: Value Forwarding Communication Structure

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

# A First Simple Example: Value Forwarding Complete Program



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

# Going Beyond the 6 Basic MPI Functions

## Init, Finalize, Get-Rank, Get-Size, Send, Recv

Point-to-Point operations

- Different variants of MPI\_Send
- Non-blocking versions

Collective operations

- Group operations across all processes of a communicator
- Examples: barriers, reductions, scatter/gather
- Neighborhood and non-blocking collectives

Group and communicator management

- Creates of groups and communicators
- Mapping of communicators to topologies

Datatypes

One-sided communication

File I/O

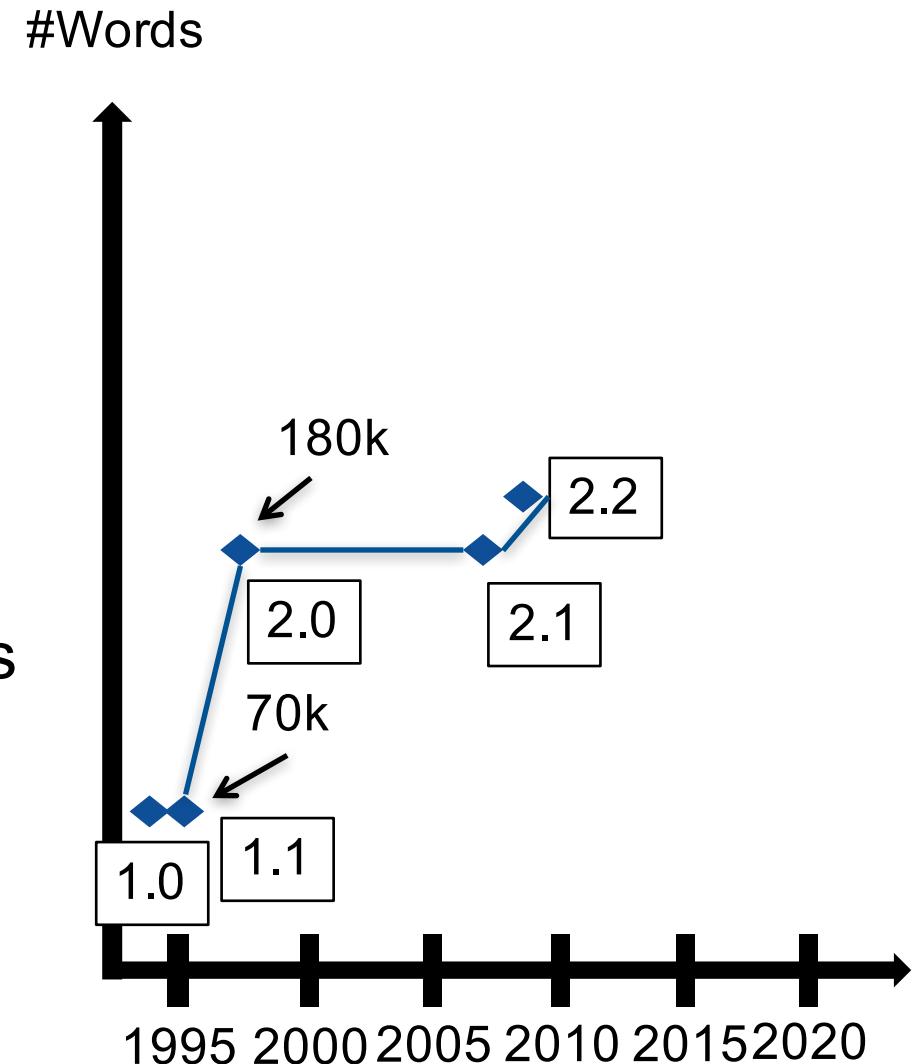
Persistent operations

# History of the Message Passing Interface

- MPI 1.0 May 1994 – 228 pages
- MPI 1.1 Nov 1995 – 238 pages (128 functions)
- MPI 2.0 Nov 1997 – 608 pages

## 10 year break

- MPI 2.1 June 2008 – 608 pages
  - Merged document
- MPI 2.2 Sep 2009 – 647 pages
  - Small, conforming additions
- In parallel, start on MPI 3.0
  - Major new concepts
  - Completed in 2012



# Notable Additions to MPI 3.0

Nonblocking collectives

Neighborhood collectives

MPI Tool Information Interface

One sided communication enhancements

Large data counts (messages more than 32bit count)

Topology aware communicator creation

Noncollective communicator creation

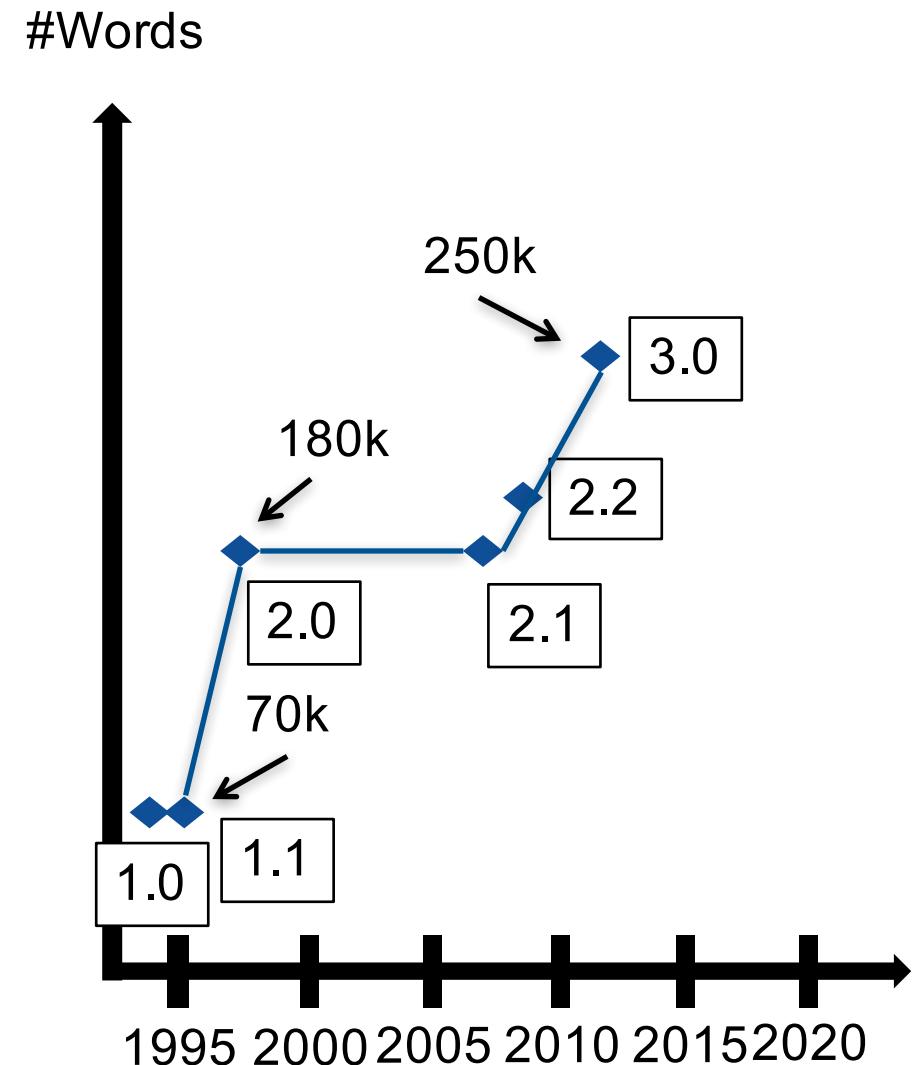
New language bindings

# MPI Keeps Growing

- MPI 1.0 May 1994 – 228 pages
- MPI 1.1 Nov 1995 – 238 pages (128 fns)
- MPI 2.0 Nov 1997 – 608 pages

## 10 year break

- MPI 2.1 June 2008 – 608 pages
- MPI 2.2 Sep 2009 – 647 pages
- MPI 3.0 Sep 2012 – 852 pages (430 fns)



# MPI 3.0 has 430 Functions



|                                |                              |                                  |                            |
|--------------------------------|------------------------------|----------------------------------|----------------------------|
| MPI_ABORT                      | MPI_ERRHANDLER_GET           | MPI_GROUP_DIFFERENCE             | MPI_TYPE_DELETE_ATTR       |
| MPI_ACCUMULATE                 | MPI_ERRHANDLER_SET           | MPI_GROUP_EXCL                   | MPI_TYPE_DUP               |
| MPI_ADD_ERROR_CLASS            | MPI_ERROR_CLASS              | MPI_GROUP_F2C                    | MPI_TYPE_DUP_FN            |
| MPI_ADD_ERROR_CODE             | MPI_ERROR_STRING             | MPI_GROUP_FREE                   | MPI_TYPE_EXTENT            |
| MPI_ADD_ERROR_STRING           | MPI_EXSCAN                   | MPI_GROUP_INCL                   | MPI_TYPE_F2C               |
| MPI_ADDRESS                    | MPI_F_SYNC_REG               | MPI_GROUP_INTERSECTION           | MPI_TYPE_FREE              |
| MPI_ALLGATHER                  | MPI_FETCH_AND_OP             | MPI_GROUP_RANGE_EXCL             | MPI_TYPE_FREE_KEYVAL       |
| MPI_ALLGATHERV                 | MPI_FILE_C2F                 | MPI_GROUP_RANGE_INCL             | MPI_TYPE_GET_ATTR          |
| MPI_ALLOC_MEM                  | MPI_FILE_CALL_ERRHANDLER     | MPI_GROUP_RANK                   | MPI_TYPE_GET_CONTENTS      |
| MPI_ALLOC_MEM_CPTR             | MPI_FILE_CLOSE               | MPI_GROUP_SIZE                   | MPI_TYPE_GET_ENVELOPE      |
| MPI_ALLREDUCE                  | MPI_FILE_CREATE_ERRHANDLER   | MPI_GROUP_TRANSLATE_RANKS        | MPI_TYPE_GET_EXTENT        |
| MPI_ALLTOALL                   | MPI_FILE_DELETE              | MPI_GROUP_UNION                  | MPI_TYPE_GET_EXTENT_X      |
| MPI_ALLTOALLV                  | MPI_FILE_F2C                 | MPI_ALLGATHER                    | MPI_TYPE_GET_NAME          |
| MPI_ALLTOALLW                  | MPI_FILE_GET_AMODE           | MPI_ALLGATHERV                   | MPI_TYPE_GET_TRUE_EXTENT   |
| MPI_ATTR_DELETE                | MPI_FILE_GET_ATOMICTY        | MPI_ALLREDUCE                    | MPI_TYPE_GET_TRUE_EXTENT_X |
| MPI_ATTR_GET                   | MPI_FILE_GET_BYTE_OFFSET     | MPI_ALLTOALL                     | MPI_TYPE_HINDEXED          |
| MPI_AVOID_PUT                  | MPI_FILE_GET_ERRHANDLER      | MPI_ALLTOALLV                    | MPI_TYPE_HVECTOR           |
| MPI_BARRIER                    | MPI_FILE_GET_GROUP           | MPI_BARRIER                      | MPI_TYPE_INDEXED           |
| MPI_CART_C2F                   | MPI_FILE_GET_INFO            | MPI_BCAST                        | MPI_TYPE_I8                |
| MPI_BSEND                      | MPI_FILE_GET_POSITION        | MPI_BSEND                        | MPI_TYPE_MATCH_SIZE        |
| MPI_BSEND_INIT                 | MPI_FILE_GET_POSITION_SHARED | MPI_EXSCAN                       | MPI_TYPE_NULL_COPY_FN      |
| MPI_BUFFER_ATTACH              | MPI_FILE_GET_SIZE            | MPI_GATHER                       | MPI_TYPE_NULL_DELETE_FN    |
| MPI_BUFFER_DETACH              | MPI_FILE_GET_TYPE_EXTENT     | MPI_GATHERV                      | MPI_TYPE_SET_ATTR          |
| MPI_CANCEL                     | MPI_FILE_GET_VIEW            | MPI_IMPROBE                      | MPI_TYPE_SET_NAME          |
| MPI_CART_COORDS                | MPI_FILE_IREAD               | MPI_IMRECV                       | MPI_TYPE_SIZE              |
| MPI_CART_CREATE                | MPI_FILE_IREAD_AT            | MPI_NEIGHBOR_ALLGATHER           | MPI_TYPE_SIZE_X            |
| MPI_CART_GET                   | MPI_FILE_IREAD_SHARED        | MPI_NEIGHBOR_ALLGATHERV          | MPI_TYPE_STRUCT            |
| MPI_CART_MAP                   | MPI_FILE_IWRITE              | MPI_NEIGHBOR_ALLTOALL            | MPI_TYPE_UB                |
| MPI_CART_RANK                  | MPI_FILE_IWRITE_AT           | MPI_NEIGHBOR_ALLTOALLV           | MPI_TYPE_VECTOR            |
| MPI_CART_SHIFT                 | MPI_FILE_IWRITE_SHARED       | MPI_NEIGHBOR_ALLTOALLV           | MPI_UNPACK                 |
| MPI_CART_SUB                   | MPI_FILE_OPEN                | MPI_INFO_C2F                     | MPI_UNPACK_EXTERNAL        |
| MPI_CARTDIM_GET                | MPI_FILE_PALLOCATE           | MPI_INFO_C2F08                   | MPI_UNPUBLISH_NAME         |
| MPI_CLOSE_PORT                 | MPI_FILE_READ                | MPI_INFO_F082C                   | MPI_WAIT                   |
| MPI_COMM_ACCEPT                | MPI_FILE_READ_ALL            | MPI_INFO_F082F                   | MPI_WAITALL                |
| MPI_COMM_C2F                   | MPI_FILE_READ_ALL_BEGIN      | MPI_INFO_F2C                     | MPI_WAITANY                |
| MPI_COMM_CALL_ERRHANDLER       | MPI_FILE_READ_ALL_END        | MPI_INFO_F2C08                   | MPI_WAITSOME               |
| MPI_COMM_COMPARE               | MPI_FILE_READ_AT             | MPI_STATUS_SET_CANCELLED         | MPI_WIN_ALLOC              |
| MPI_COMM_CONNECT               | MPI_FILE_READ_AT_ALL         | MPI_STATUS_SET_ELEMENTS          | MPI_WIN_ALLOCATE           |
| MPI_COMM_CREATE                | MPI_FILE_READ_AT_ALL_BEGIN   | MPI_STATUS_SET_ELEMENTS_X        | MPI_WIN_ALLOCATE_CPTR      |
| MPI_COMM_CREATE_ERRHANDLER     | MPI_FILE_READ_AT_ALL_END     | MPI_T_CATEGORY_CHANGED           | MPI_WIN_ALLOCATE_SHARED    |
| MPI_COMM_CREATE_GROUP          | MPI_FILE_READ_ORDERED        | MPI_T_CATEGORY_GET_CATEGORIES    | MPI_WIN_ATTACH             |
| MPI_COMM_CREATE_KEYVAL         | MPI_FILE_READ_ORDERED_BEGIN  | MPI_T_CATEGORY_GET_CVARS         | MPI_WIN_C2F                |
| MPI_COMM_DELETE_ATTR           | MPI_FILE_READ_ORDERED_END    | MPI_T_CATEGORY_GET_INFO          | MPI_WIN_CALL_ERRHANDLER    |
| MPI_COMM_DISCONNECT            | MPI_FILE_READ_SHARED         | MPI_T_CATEGORY_GET_NUM           | MPI_WIN_COMPLETE           |
| MPI_COMM_DUP                   | MPI_FILE_SEEK                | MPI_T_INTERCOMM_CREATE           | MPI_WIN_CREATE             |
| MPI_COMM_DUP_FN                | MPI_FILE_SEER_SHAR_ED        | MPI_T_INTERCOMM_MERGE            | MPI_WIN_CREATE_DYNAMIC     |
| MPI_COMM_DUP_WITH_INFO         | MPI_FILE_SET_ATOMICTY        | MPI_PROBE                        | MPI_WIN_CREATE_ERRHANDLER  |
| MPI_COMM_F2C                   | MPI_FILE_SET_ERRHANDLER      | MPI_RECV                         | MPI_WIN_CREATE_KEYVAL      |
| MPI_COMM_FREE                  | MPI_FILE_SET_INFO            | MPI_REDUCE                       | MPI_WIN_DELETE_ATTR        |
| MPI_COMM_FREE_KEYVAL           | MPI_FILE_SET_SIZE            | MPI_REDUCE_SCATTER               | MPI_WIN_DELEACH            |
| MPI_COMM_GET_ALL               | MPI_FILE_SET_NEW             | MPI_REDUCE_SCATTER_BLOCK         | MPI_WIN_DUP_FN             |
| MPI_COMM_GET_ERRHANDLER        | MPI_FILE_SYNC                | MPI_RSND                         | MPI_WIN_F2C                |
| MPI_COMM_GET_INFO              | MPI_FILE_WRITE               | MPI_IS_THREAD_MAIN               | MPI_WIN_FENCE              |
| MPI_COMM_GET_NAME              | MPI_FILE_WRITE_ALL           | MPI_SCAN                         | MPI_WIN_FLUSH              |
| MPI_COMM_GET_PARENT            | MPI_FILE_WRITE_ALL_BEGIN     | MPI_SCATTER                      | MPI_WIN_FLUSH_ALL          |
| MPI_COMM_GROUP                 | MPI_FILE_WRITE_ALL_END       | MPI_SCATTERV                     | MPI_WIN_FLUSH_LOCAL        |
| MPI_COMM_IDUP                  | MPI_FILE_WRITE_AT            | MPI_SEND                         | MPI_WIN_FLUSH_LOCAL_ALL    |
| MPI_COMM_JOIN                  | MPI_FILE_WRITE_AT_ALL        | MPI_ISSEND                       | MPI_WIN_FREE               |
| MPI_COMM_KEYVAL_CREATE         | MPI_FILE_WRITE_AT_ALL_BEGIN  | MPI_KEYVAL_CREATE                | MPI_WIN_FREE_KEYVAL        |
| MPI_COMM_NULL_COPY_FN          | MPI_FILE_WRITE_AT_ALL_END    | MPI_KEYVAL_FREE                  | MPI_WIN_GET_ATTR           |
| MPI_COMM_NULL_DELETE_FN        | MPI_FILE_WRITE_ORDERED       | MPI_LOCK_ALL                     | MPI_WIN_GET_ERRHANDLER     |
| MPI_COMM_RANK                  | MPI_FILE_WRITE_ORDERED_BEGIN | MPI_LOOKUP_NAME                  | MPI_WIN_GET_GROUP          |
| MPI_COMM_REMOTE_GROUP          | MPI_FILE_WRITE_ORDERED_END   | MPI_MESSAGE_C2F                  | MPI_WIN_GET_INFO           |
| MPI_COMM_REMOTE_SIZE           | MPI_FILE_WRITE_SHARED        | MPI_MESSAGE_F2C                  | MPI_WIN_GET_NAME           |
| MPI_COMM_SET_ATTR              | MPI_FINALIZE                 | MPI_probe                        | MPI_WIN_LOCK               |
| MPI_COMM_SET_ERRHANDLER        | MPI_FINALIZED                | MPI_RECV                         | MPI_WIN_LOCK_ALL           |
| MPI_COMM_SET_INFO              | MPI_FREE_MEM                 | MPI_TEST                         | MPI_WIN_NULL_COPY_FN       |
| MPI_COMM_SET_NAME              | MPI_GATHER                   | MPI_TEST_CANCELLED               | MPI_WIN_NULL_DELETE_FN     |
| MPI_COMM_SIZE                  | MPI_GATHERV                  | MPI_TESTALL                      | MPI_WIN_POST               |
| MPI_COMM_SPAWN                 | MPI_GET                      | MPI_TESTANY                      | MPI_WIN_SET_ATTR           |
| MPI_COMM_SPAWN_MULTIPLE        | MPI_GET_ACCUMULATE           | MPI_TESTSOME                     | MPI_WIN_SET_ERRHANDLER     |
| MPI_COMM_SPLIT                 | MPI_GET_ADDRESS              | MPI_TOPO_TEST                    | MPI_WIN_SET_INFO           |
| MPI_COMM_SPLIT_TYPE            | MPI_GET_COUNT                | MPI_TYPE_C2F                     | MPI_WIN_SET_NAME           |
| MPI_COMM_TEST_INTER            | MPI_GET_ELEMENTS             | MPI_TYPE_COMMIT                  | MPI_WIN_SHARED_ALLOCATE    |
| MPI_COMM_WORLD                 | MPI_GET_ELEMENTS_X           | MPI_TYPE_CONTIGUOUS              | MPI_WIN_SHARED_QUERY       |
| MPI_COMPARE_AND_SWAP           | MPI_GET_LIBRARY_VERSION      | MPI_TYPE_CREATE_DARRAY           | MPI_WIN_SHARED_QUERY_CPTR  |
| MPI_CONVERSION_FN_NULL         | MPI_GET_PROCESSOR_NAME       | MPI_TYPE_CREATE_F90_COMPLEX      | MPI_WIN_START              |
| MPI_DIMS_CREATE                | MPI_GET_VERSION              | MPI_TYPE_CREATE_F90_INTEGER      | MPI_WIN_SYNC               |
| MPI_DIST_GRAPH_CREATE          | MPI_GRAPH_CREATE             | MPI_TYPE_CREATE_F90_REAL         | MPI_WIN_TEST               |
| MPI_DIST_GRAPH_CREATE_ADJACENT | MPI_GRAPH_GET                | MPI_TYPE_CREATE_HINDEXED         | MPI_WIN_UNLOCK             |
| MPI_DIST_GRAPH_NEIGHBOR_COUNT  | MPI_GRAPH_MAP                | MPI_TYPE_CREATE_HINDEXED_BLOCK_K | MPI_WIN_UNLOCK_ALL         |
| MPI_DIST_GRAPH_NEIGHBORS       | MPI_GRAPH_NEIGHBORS          | MPI_TYPE_CREATE_EXTERNAL         | MPI_WIN_WAIT               |
| MPI_DIST_GRAPH_NEIGHBORS_COUNT | MPI_GRAPH_NEIGHBORS_COUNT    | MPI_TYPE_CREATE_INDEXED_BLOCK    | MPI_WTICK                  |
| MPI_DUP_FN                     | MPI_GRAPHDIMS_GET            | MPI_TYPE_CREATE_KEYVAL           | MPI_WTIME                  |
| MPI_ERRHANDLER_C2F             | MPI_GREQUEST_COMPLETE        | MPI_PROBE                        |                            |
| MPI_ERRHANDLER_CREATE          | MPI_GREQUEST_START           | MPI_PUBLISH_NAME                 |                            |
| MPI_ERRHANDLER_F2C             | MPI_GROUP_C2F                | MPI_PUT                          |                            |
| MPI_ERRHANDLER_FREE            | MPI_GROUP_COMPARE            |                                  |                            |



# MPI 3.0 has 430 Functions



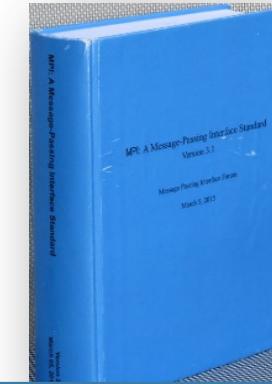
|                                       |                                |                               |                                      |                            |
|---------------------------------------|--------------------------------|-------------------------------|--------------------------------------|----------------------------|
| MPI_ABORT                             | MPI_ERRHANDLER_GET             | MPI_GROUP_DIFFERENCE          | MPI_QUERY_THREAD                     | MPI_TYPE_DELETE_ATTR       |
| MPI_ACCUMULATE                        | MPI_ERRHANDLER_SET             | MPI_GROUP_EXCL                | MPI_RACCUMULATE                      | MPI_TYPE_DUP               |
| MPI_ADD_ERROR_CLASS                   | MPI_ERROR_CLASS                | MPI_GROUP_F2C                 | MPI_RECV                             | MPI_TYPE_DUP_FN            |
| MPI_ADD_ERROR_CODE                    | MPI_ERROR_STRING               | MPI_GROUP_FREE                | MPI_RECV_INIT                        | MPI_TYPE_EXTENT            |
| MPI_ADD_ERROR_STRING                  | MPI_EXSCAN                     | MPI_GROUP_INCL                | MPI_REDUCE                           | MPI_TYPE_F2C               |
| MPI_ADDRESS                           | MPI_F_SYNC_REG                 | MPI_GROUP_INTERSECTION        | MPI_REDUCE_LOCAL                     | MPI_TYPE_FREE              |
| MPI_ALLGATHER                         | MPI_FETCH_AND_OP               | MPI_GROUP_RANGE_EXCL          | MPI_REDUCE_SCATTER                   | MPI_TYPE_FREE_KEYVAL       |
| MPI_ALLGATHERV                        | MPI_FILE_C2F                   | MPI_GROUP_RANGE_INCL          | MPI_REDUCE_SCATTER_BLOCK             | MPI_TYPE_GET_ATTR          |
| MPI_ALLOC_MEM                         | MPI_FILE_CALL_ERRHANDLER       | MPI_GROUP_RANK                | MPI_REGISTER_DATAREP                 | MPI_TYPE_GET_CONTENTS      |
| MPI_ALLOC_MEM_CPTR                    | MPI_FILE_CLOSE                 | MPI_GROUP_SIZE                | MPI_REQUEST_C2F                      | MPI_TYPE_GET_EXTENT        |
| MPI_ALLREDUCE                         | MPI_FILE_CREATE_ERRHANDLER     | MPI_GROUP_UNION               | MPI_REQUEST_F2C                      | MPI_TYPE_GET_EXTENT_X      |
| MPI_ALLTOALL                          | MPI_FILE_DELETE                | MPI_ALLGATHER                 | MPI_REQUEST_FREE                     | MPI_TYPE_GET_NAME          |
| MPI_ALLTOALLV                         | MPI_FILE_F2C                   | MPI_ALLGATHERV                | MPI_REQUEST_GET_STATUS               | MPI_TYPE_GET_TRUE_EXTENT   |
| MPI_ALLTOALLW                         | MPI_FILE_GET_AMODE             | MPI_ALLREDUCE                 | MPI_RGET                             | MPI_TYPE_GET_TRUE_EXTENT_X |
| MPI_ATTR_DELETE                       | MPI_FILE_GET_ATOMICTY          | MPI_ALLTOALL                  | MPI_RGET_ACCUMULATE                  | MPI_TYPE_HINDEXED          |
| MPI_ATTR_GET                          | MPI_FILE_GET_BYTE_OFFSET       | MPI_ALLTOALLV                 | MPI_RPUT                             | MPI_TYPE_HVECTOR           |
| MPI_AVOID_PUT                         | MPI_FILE_GET_ERRHANDLER        | MPI_BARRIER                   | MPI_RSEND                            | MPI_TYPE_INDEXED           |
| MPI_BARRIER                           | MPI_FILE_GET_GROUP             | MPI_BCAST                     | MPI_RSEND_INIT                       | MPI_TYPE_I_B               |
| MPI_CART_C2F                          | MPI_FILE_GET_INFO              | MPI_BSEND                     | MPI_SCATTER                          | MPI_TYPE_MATCH_SIZE        |
| MPI_BSEND                             | MPI_FILE_GET_POSITION          | MPI_EXSCAN                    | MPI_SCATTERV                         | MPI_TYPE_NULL_COPY_FN      |
| MPI_BSEND_INIT                        | MPI_FILE_GET_POSITION_SHARED   | MPI_GATHER                    | MPI_SEND                             | MPI_TYPE_NULL_DELETE_FN    |
| MPI_BUFFER_ATTACH                     | MPI_FILE_GET_SIZE              | MPI_GATHERV                   | MPI_SEND_INIT                        | MPI_TYPE_SET_ATTR          |
| MPI_BUFFER_DETACH                     | MPI_FILE_GET_TYPE_EXTENT       | MPI_IMPROBE                   | MPI_SENDRECV                         | MPI_TYPE_SET_NAME          |
| MPI_CANCEL                            | MPI_FILE_GET_VIEW              | MPI_IRECVR                    | MPI_SENDRECV_REPLACE                 | MPI_TYPE_SIZE              |
| MPI_CART_COORDS                       | MPI_FILE_IREAD                 | MPI_NEIGHBOR_ALLGATHER        | MPI_SIZEOF                           | MPI_TYPE_SIZE_X            |
| MPI_CART_CREATE                       | MPI_FILE_IREAD_AT              | MPI_NEIGHBOR_ALLGATHERV       | MPI_SSSEND                           | MPI_TYPE_STRUCT            |
| MPI_CART_GET                          | MPI_FILE_IREAD_SHARED          | MPI_NEIGHBOR_ALLTOALL         | MPI_SSSEND_INIT                      | MPI_TYPE_UB                |
| MPI_CART_MAP                          | MPI_FILE_IWRITE                | MPI_NEIGHBOR_ALLTOALLV        | MPI_START                            | MPI_TYPE_VECTOR            |
| MPI_CART_RANK                         | MPI_FILE_IWRITE_AT             | MPI_NEIGHBOR_ALLTOALLW        | MPI_STARTALL                         | MPI_UNPACK                 |
| MPI_CART_SHIFT                        | MPI_FILE_IWRITE_SHARED         | MPI_OPEN                      | MPI_STATUS_C2F                       | MPI_UNPACK_EXTERNAL        |
| MPI_CART_SUB                          | MPI_FILE_OPEN                  | MPI_PINFO_C2F8                | MPI_UNPUBLISH_NAME                   | MPI_WAIT                   |
| MPI_CARTDIM_GET                       | MPI_FILE_PALLOCATE             | MPI_STATUS_F082C              | MPI_WAITALL                          | MPI_WAITALL                |
| MPI CLOSE_PORT                        | MPI_FILE_READ                  | MPI_STATUS_F082F              | MPI_WAITANY                          | MPI_WAITANY                |
| MPI_COMM_ACCEPT                       | MPI_FILE_READ_ALL              | MPI_STATUS_F2C                | MPI_WAITSOME                         | MPI_WAITSOME               |
| MPI_COMM_C2F                          | MPI_FILE_READ_ALL_BEGIN        | MPI_STATUS_F2F08              | MPI_WIN_ALLOC                        | MPI_WIN_ALLOCATE           |
| MPI_COMM_CALL_ERRHANDLER              | MPI_FILE_READ_ALL_END          | MPI_STATUS_SET_CANCELLED      | MPI_WIN_ALLOCATE_CPTR                | MPI_WIN_ALLOCATE_SHARED    |
| MPI_COMM_COMPARE                      | MPI_FILE_READ_AT               | MPI_STATUS_SET_ELEMENTS       | MPI_WIN_ALLOCATE_SHARED_CPTR         | MPI_WIN_ATTACH             |
| MPI_COMM_CONNECT                      | MPI_FILE_READ_AT_ALL           | MPI_STATUS_SET_ELEMENTS_X     | MPI_WIN_CREATE                       | MPI_WIN_C2F                |
| MPI_COMM_CREATE                       | MPI_FILE_READ_AT_ALL_BEGIN     | MPI_T_CATEGORY_CHANGED        | MPI_WIN_CREATE_DYNAMIC               | MPI_WIN_CALL_ERRHANDLER    |
| MPI_COMM_CREATE_ERRHANDLER            | MPI_FILE_READ_AT_ALL_END       | MPI_T_CATEGORY_GET_CATEGORIES | MPI_WIN_CREATE_ERRHANDLER            | MPI_WIN_COMPLETE           |
| <b>MPI_COMM_CREATE_GROUP</b>          | MPI_FILE_READ_ORDERED          | MPI_T_CATEGORY_GET_CVARS      | MPI_WIN_CREATE_KEYVAL                | MPI_WIN_DELETE_ATTR        |
| MPI_COMM_CREATE_KEYVAL                | MPI_FILE_READ_ORDERED_BEGIN    | MPI_T_CATEGORY_GET_INFO       | MPI_WIN_DELETE_FN                    | MPI_WIN_DESTROY            |
| MPI_COMM_DELETE_ATTR                  | MPI_FILE_READ_ORDERED_END      | MPI_T_CATEGORY_GET_NUM        | MPI_WIN_DUP_FN                       | MPI_WIN_F2C                |
| MPI_COMM_DISCONNECT                   | MPI_FILE_READ_SHARED           | MPI_T_CATEGORY_GET_PVARS      | MPI_WIN_FENCE                        | MPI_WIN_FLUSH              |
| MPI_COMM_DUP                          | MPI_FILE_SEEK                  | MPI_T_CVAR_GET_INFO           | MPI_WIN_FLUSH_ALL                    | MPI_WIN_FLUSH_LOCAL        |
| MPI_COMM_DUP_FN                       | MPI_FILE_SEEK_SHARED           | MPI_T_CVAR_HANDLE_ALLOC       | MPI_WIN_FLUSH_LOCAL_ALL              | MPI_WIN_FLUSH_ALL          |
| <b>MPI_COMM_DUP_WITH_INFO</b>         | MPI_FILE_SET_ATOMICITY         | MPI_T_CVAR_HANDLE_FREE        | MPI_WIN_FREE                         | MPI_WIN_FREE_KEYVAL        |
| MPI_COMM_F2C                          | MPI_FILE_SET_ERRHANDLER        | MPI_T_CVAR_READ               | MPI_WIN_GET_ATTR                     | MPI_WIN_GET_ERRHANDLER     |
| MPI_COMM_FREE                         | MPI_FILE_SET_INFO              | MPI_T_CVAR_WRITE              | MPI_WIN_GET_GROUP                    | MPI_WIN_GET_INFO           |
| MPI_COMM_FREE_KEYVAL                  | MPI_FILE_SET_NEW               | MPI_T_ENUM_GET_INFO           | MPI_WIN_GET_NAME                     | MPI_WIN_GET_NAME           |
| MPI_COMM_GET_ATTR                     | MPI_FILE_SET_NEW               | MPI_T_ENUM_GET_ITEM           | MPI_WIN_LOCK                         | MPI_WIN_LOCK_ALL           |
| MPI_COMM_GET_ERRHANDLER               | MPI_FILE_SYNC                  | MPI_T_FINALIZE                | MPI_TEST                             | MPI_WIN_NULL_COPY_FN       |
| <b>MPI_COMM_GET_INFO</b>              | MPI_FILE_WRITE                 | MPI_T_INIT_THREAD             | MPI_TEST_CANCELLED                   | MPI_WIN_NULL_DELETE_FN     |
| MPI_COMM_GET_NAME                     | MPI_FILE_WRITE_ALL             | MPI_IS_THREAD_MAIN            | MPI_TESTALL                          | MPI_WIN_POST               |
| MPI_COMM_GET_PARENT                   | MPI_FILE_WRITE_ALL_BEGIN       | MPI_ISCAN                     | MPI_TESTANY                          | MPI_WIN_SET_ATTR           |
| <b>MPI_COMM_IDUP</b>                  | MPI_FILE_WRITE_ALL_END         | MPI_ISCATTER                  | MPI_TESTSOME                         | MPI_WIN_SET_ERRHANDLER     |
| MPI_COMM_GROUP                        | MPI_FILE_WRITE_AT              | MPI_ISCATTERV                 | MPI_TOPO_TEST                        | MPI_WIN_SET_INFO           |
| MPI_COMM_JOIN                         | MPI_FILE_WRITE_AT_ALL          | MPI_ISEND                     | MPI_TYPE_C2F                         | MPI_WIN_SET_NAME           |
| MPI_COMM_KEYVAL_CREATE                | MPI_FILE_WRITE_AT_ALL_BEGIN    | MPI_ISSEND                    | MPI_TYPE_COMMIT                      | MPI_WIN_SHARED_ALLOCATE    |
| MPI_COMM_NULL_COPY_FN                 | MPI_FILE_WRITE_AT_ALL_END      | MPI_KEYVAL_CREATE             | MPI_TYPE_CONTIGUOUS                  | MPI_WIN_SHARED_QUERY       |
| MPI_COMM_NULL_DELETE_FN               | MPI_FILE_WRITE_ORDERED         | MPI_KEYVAL_FREE               | MPI_TYPE_CREATE_DARRAY               | MPI_WIN_SHARED_QUERY_CPTR  |
| MPI_COMM_RANK                         | MPI_FILE_WRITE_ORDERED_BEGIN   | <b>MPI_LOCK_ALL</b>           | MPI_TYPE_CREATE_F90_COMPLEX          | MPI_WIN_START              |
| MPI_COMM_REMOTE_GROUP                 | MPI_FILE_WRITE_ORDERED_END     | MPI_LOOKUP_NAME               | MPI_TYPE_CREATE_F90_INTEGER          | <b>MPI_WIN_SYNC</b>        |
| MPI_COMM_REMOTE_SIZE                  | MPI_FILE_WRITE_SHARED          | MPI_MESSAGE_C2F               | MPI_TYPE_CREATE_F90_REAL             | MPI_WIN_TEST               |
| MPI_COMM_SET_ATTR                     | MPI_FINALIZE                   | MPI_MESSAGE_F2C               | MPI_TYPE_CREATE_INDEXED              | MPI_WIN_UNLOCK             |
| MPI_COMM_SET_ERRHANDLER               | MPI_FINALIZED                  | <b>MPI_MPROBE</b>             | <b>MPI_TYPE_CREATE_INDEXED_BLOCK</b> | MPI_WIN_BLOCK_ALL          |
| <b>MPI_COMM_SET_INFO</b>              | MPI_FREE_MEM                   | MPI_MRECV                     | MPI_TYPE_CREATE_INDEXED_BLOCK_FN     | MPI_WIN_WAIT               |
| MPI_COMM_SET_NAME                     | MPI_GATHER                     | MPI_NEIGHBOR_ALLGATHER        | MPI_TYPE_CREATE_INDEXED_BLOCK_FN     | MPI_WTICK                  |
| MPI_COMM_SIZE                         | MPI_GATHERV                    | MPI_NEIGHBOR_ALLGATHERV       | MPI_TYPE_CREATE_KEYVAL               | MPI_WTIME                  |
| MPI_COMM_SPAWN                        | MPI_GET                        | MPI_NEIGHBOR_ALLTOALL         | MPI_TYPE_CREATE_RESIZED              |                            |
| MPI_COMM_SPAWN_MULTIPLE               | <b>MPI_GET_ACCUMULATE</b>      | MPI_NEIGHBOR_ALLTOALLV        | MPI_TYPE_CREATE_STRUCT               |                            |
| MPI_COMM_SPLIT                        | MPI_GET_ADDRESS                | MPI_NULL_COPY_FN              | MPI_TYPE_CREATE_SUBARRAY             |                            |
| <b>MPI_COMM_SPLIT_TYPE</b>            | MPI_GET_COUNT                  | MPI_NULL_DELETE_FN            |                                      |                            |
| MPI_COMM_TEST_INTER                   | MPI_GET_ELEMENTS               | MPI_OP_C2F                    |                                      |                            |
| MPI_COMM_WORLD                        | <b>MPI_GET_ELEMENTS_X</b>      | MPI_OP_COMMUTATIVE            |                                      |                            |
| <b>MPI_COMPARE_AND_SWAP</b>           | <b>MPI_GET_LIBRARY_VERSION</b> | MPI_OP_CREATE                 |                                      |                            |
| MPI_CONVERSION_FN_NULL                | MPI_GET_PROCESSOR_NAME         | MPI_OP_F2C                    |                                      |                            |
| MPI_DIMS_CREATE                       | MPI_GET_VERSION                | MPI_OP_FREE                   |                                      |                            |
| <b>MPI_DIST_GRAPH_CREATE</b>          | MPI_GRAPH_CREATE               | MPI_OPEN_PORT                 |                                      |                            |
| <b>MPI_DIST_GRAPH_CREATE_ADJACENT</b> | MPI_GRAPH_CREATE               | MPI_PACK                      |                                      |                            |
| <b>MPI_DIST_GRAPH_NEIGHBOR_COUN</b>   | MPI_GRAPH_GET                  | MPI_PACK_EXTERNAL             |                                      |                            |
| <b>MPI_DIST_GRAPH_NEIGHBORS</b>       | MPI_GRAPH_MAP                  | MPI_PACK_SIZE                 |                                      |                            |
| <b>MPI_DIST_GRAPH_NEIGHBORS_COUN</b>  | MPI_GRAPH_NEIGHBORS            | MPI_PCONTROL                  |                                      |                            |
| MPI_DIST_GRAPH_NEIGHBORS_COUN         | MPI_GRAPH_NEIGHBORS_COUNT      | MPI_PROBE                     |                                      |                            |
| MPI_DUP_C2F                           | MPI_GRAPHDIMS_GET              | MPI_PUBLISH_NAME              |                                      |                            |
| MPI_ERRHANDLER_C2F                    | MPI_GREQUEST_COMPLETE          | MPI_PUT                       |                                      |                            |
| MPI_ERRHANDLER_CREATE                 | MPI_GREQUEST_START             |                               |                                      |                            |
| MPI_ERRHANDLER_F2C                    | MPI_GROUP_C2F                  |                               |                                      |                            |
| MPI_ERRHANDLER_FREE                   | MPI_GROUP_COMPARE              |                               |                                      |                            |



# MPI Continuous to Evolve

MPI 3.0 ratified in September 2012

- Available at <http://www mpi-forum.org/>
- Several major additions compared to MPI 2.2



Available through HLRS  
-> MPI Forum Website

MPI 3.1 ratified in June 2015

- Inclusion for errata (mainly RMA, Fortran, MPI\_T)
- Minor updates and additions (address arithmetic and non-block. I/O)
- Adaption in most MPIS progressing fast



# On the Road to MPI 4.0

Some of the topics currently under discussion

- Better support for persistent communication
- Improved tool interfaces
- Support for fault tolerance
- Improved support for Hybrid programming
  - How to better interact with OpenMP or CUDA?
  - MPI+X model
- Streaming communication
- Improved runtime support for scaling

# The MPI Forum Drives MPI

<https://www mpi-forum.org/>

Standardization body for MPI

- Discusses additions and new directions
- Oversees the correctness and quality of the standard
- Represents MPI to the community

Open membership

- Any organization is welcome to participate
- Consists of working groups and the actual MPI forum
- Physical meetings 4 times each year (3 in the US, one with EuroMPI conference)
  - Working groups meet between forum meetings (via phone)
  - Plenary/full forum work is done mostly at the physical meetings
- Voting rights depend on attendance
  - An organization has to be present two out of the last three meetings (incl. the current one) to be eligible to vote

Technical work driven by the working groups

- <https://www mpi-forum.org/mpi-40/>

# Typical Way New Features Get Added to MPI

1. New items brought to a matching working group for discussion
2. Creation of preliminary proposal
3. Socializing of idea driven by the WG  
Through community discussions, user feedback, publications, ...  
  
Development of full proposal  
In many cases accompanied with prototype development work
4. MPI forum reading/voting process  
One reading  
Two votes  
Slow and consensus driven process
5. Once enough topics are completed:  
Publication of a new standard



# Programming in (Pure) MPI

Target: Distributed Memory / Shared Nothing Architectures<sup>independent processes, memory</sup>

- Need to split work into independent pieces
- Each piece must be its own separate program

divide  
Carve up “shared” data structures

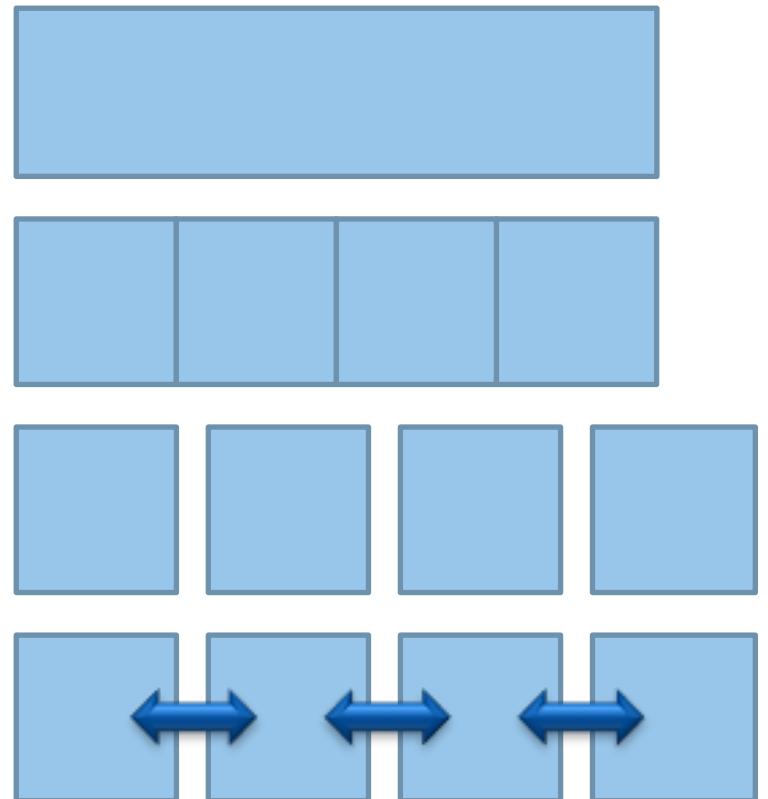
- Each MPI process must have its local data
- Avoid or duplicate data needed by one or more

Allocate data elements locally

- Independent allocations with own addresses
- Avoid central initialization
  - initialization only in first process will be costly

Add communication

- Exchange of data
- Synchronization



# Create a “Logical” Topology

Arrange data items as needed

- Define base elements
- Create mapping of element to MPI process

Create “logical topology”

- Establish overlap, e.g., <sup>halos</sup> have to be communicated with neighbors
- Define neighborhood information
- Establish communication direction

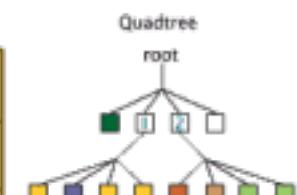
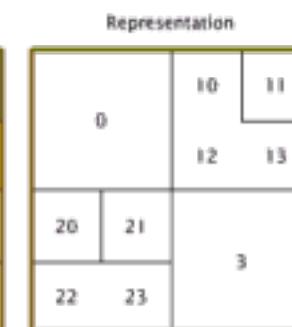
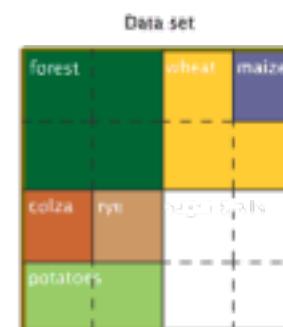
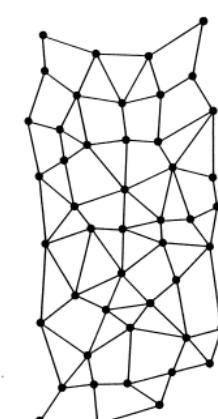
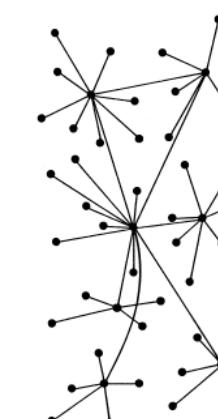
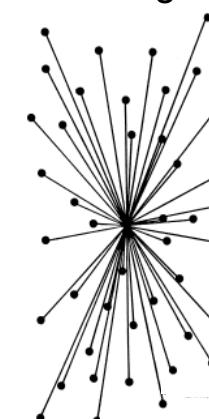
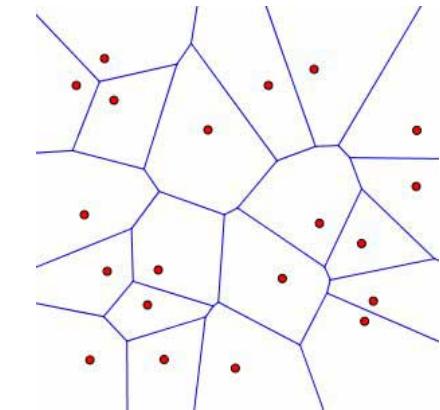
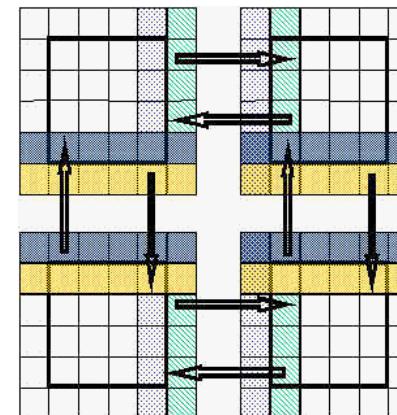
Understand frequency of needed communication

Often helpful to establish local coordinates

- Extract from rank and size information

Often helpful to explicitly define neighbors

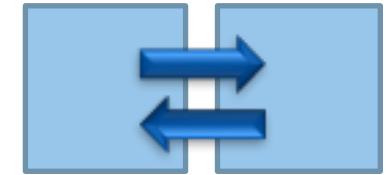
- Option 1: calculate or deduce neighbors
- Option 2: active data exchange



# Common Operation: Bidirectional Exchange

Transfer data elements between two “adjacent”

- Same or similar data exchanged between two MPI processes
- Example: halo exchange



`MPI_Send( to: 1 )`

`MPI_Recv( from: 1 )`

`MPI_Send( to: 0 )`

`MPI_Recv( from: 0 )`



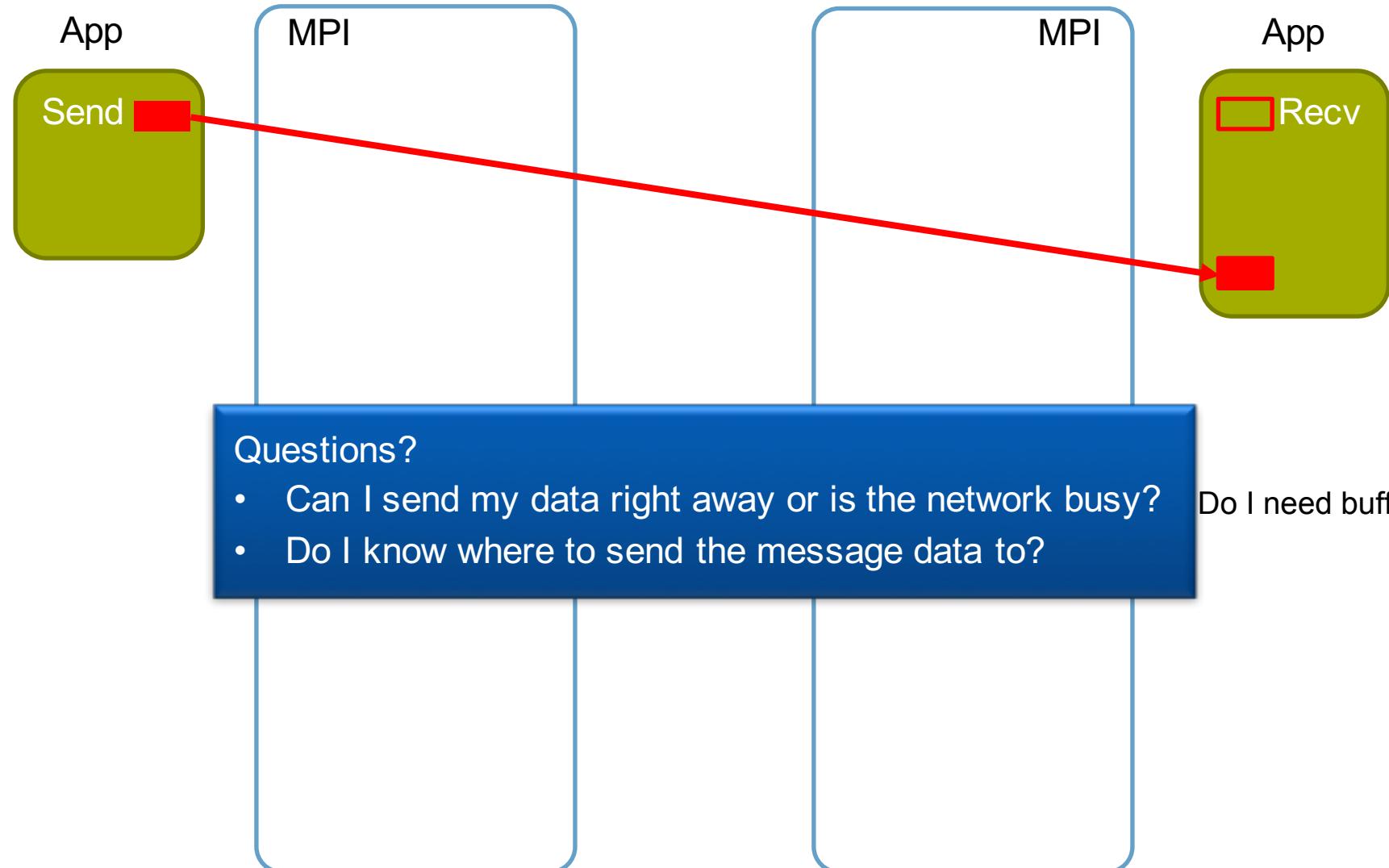
Problem: deadlock possible

Not a good idea!

- `MPI_Send` may need resources on receiver side

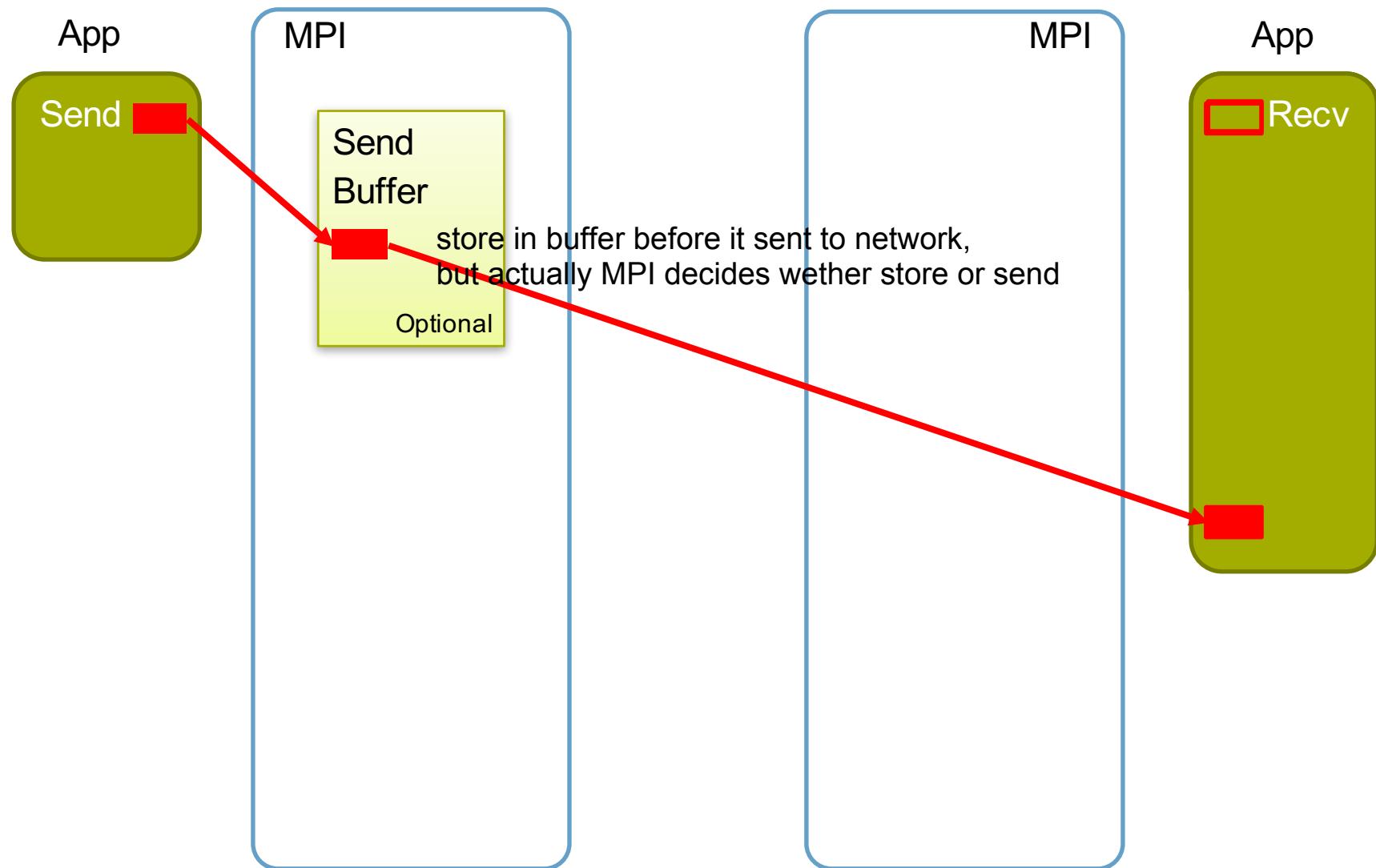
# How Messages are Transferred

Sender



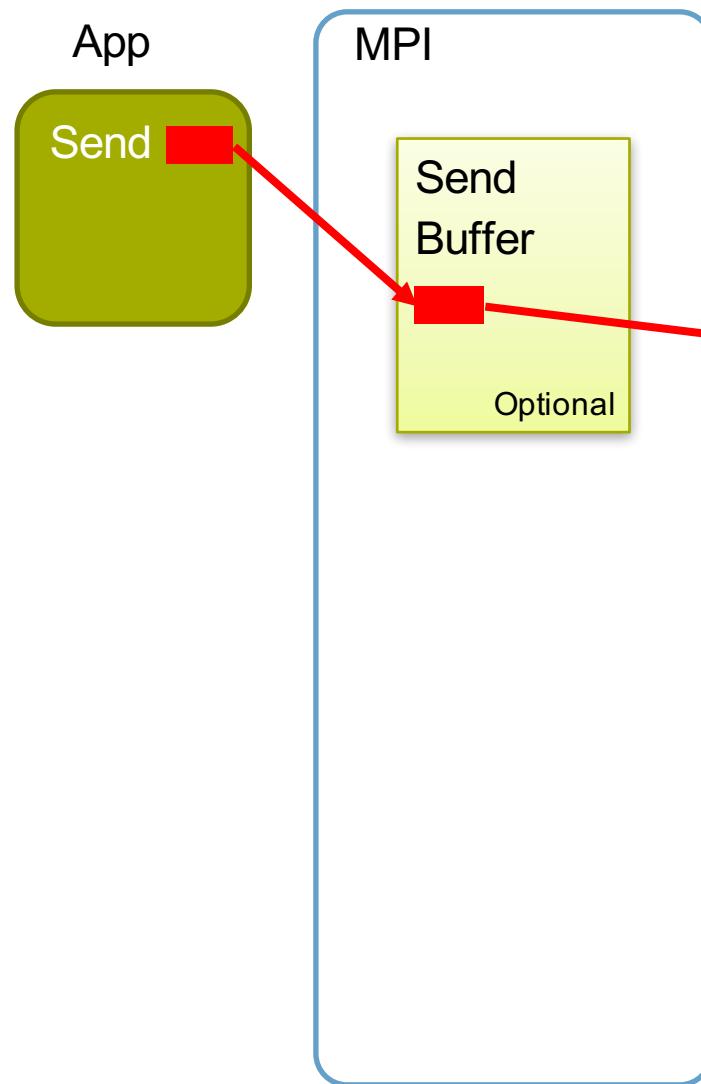
# How Messages are Transferred

Sender

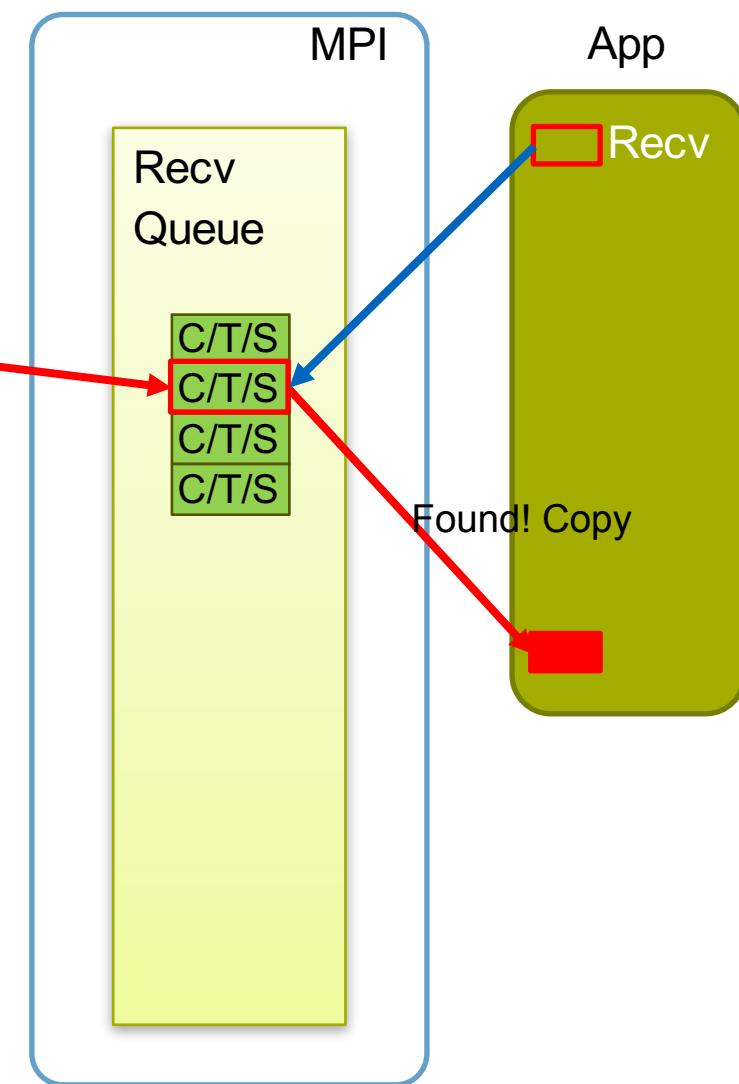


# How Messages are Transferred

Sender

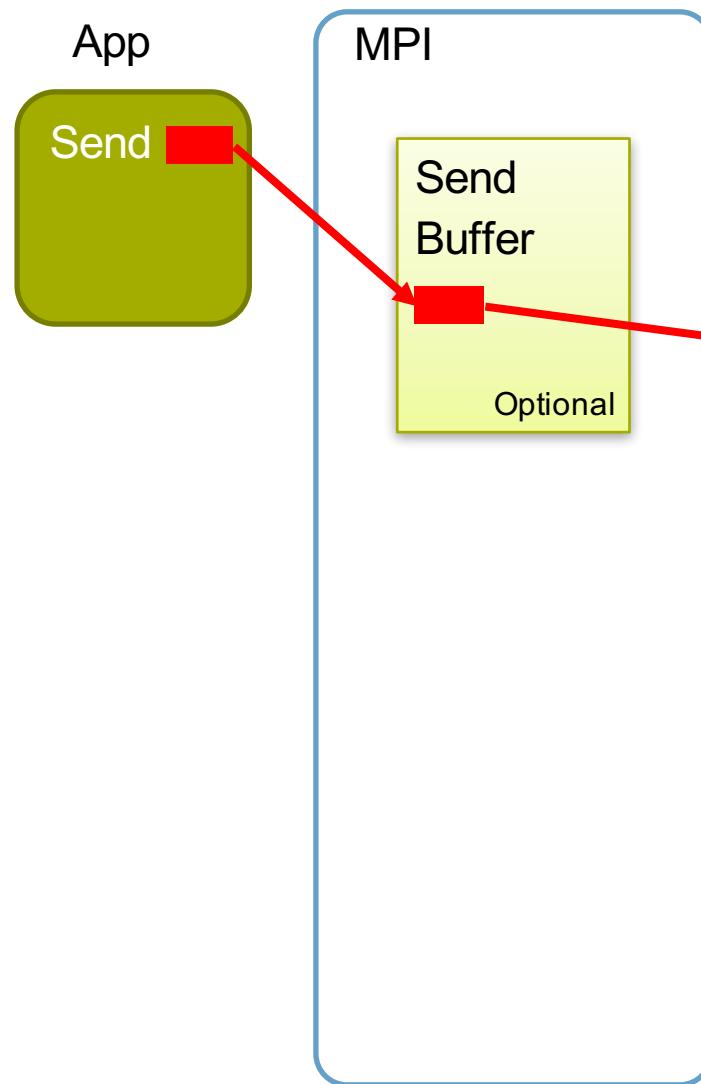


Receiver

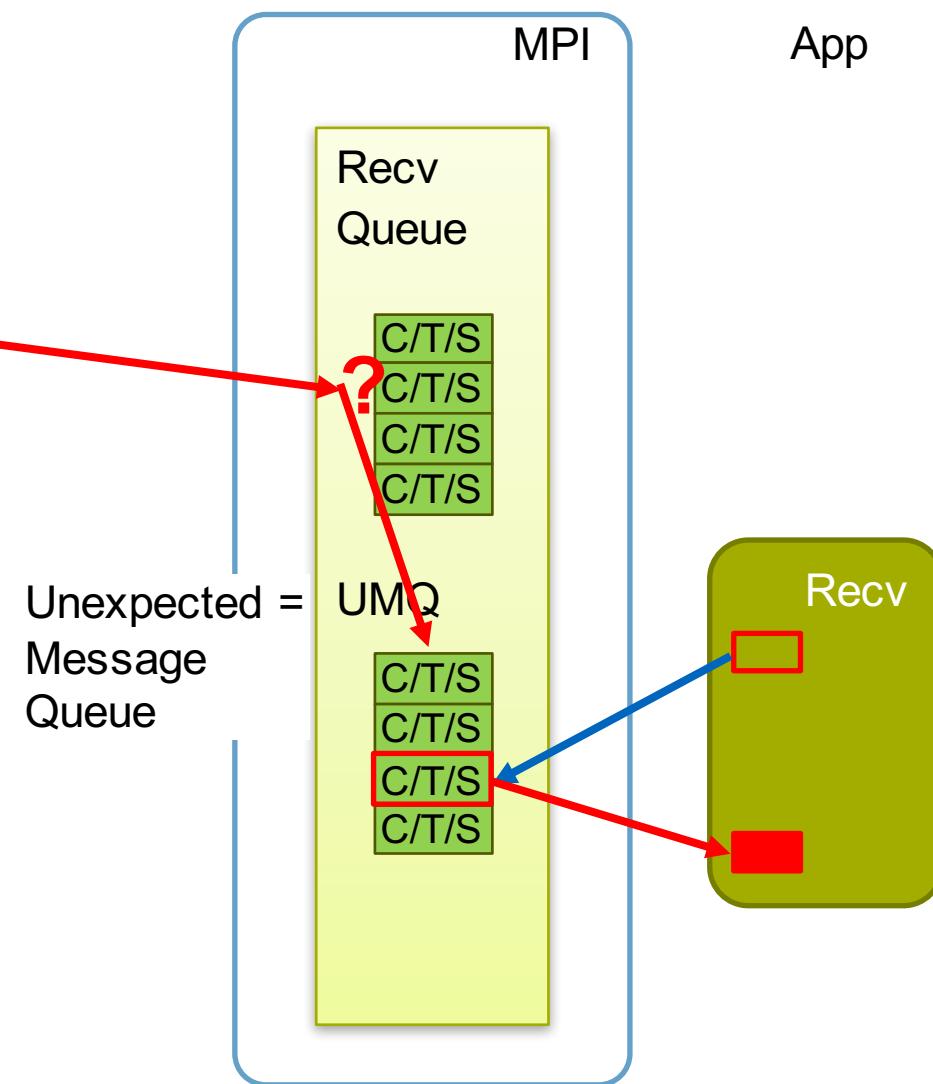


# How Messages are Transferred

Sender



Receiver



There are so many different ways to communicate between sender and receiver.



# Send Variants

## **MPI\_Send**

- Standard send operation, no extra synchronization
- Sender side handling implementation defined

## **MPI\_Bsend**

- Buffered Send
- Force the use of a send buffer
- Returns immediately, but costs resources      Could avoid deadlock

**MPI\_Ssend** Keep continuous blocking, waiting for receiver and for acknowledgment is costly

- Synchronous send
- Only returns once receive has started
- Adds extra synchronization, but can be costly

## **MPI\_Rsend**

- Ready Send receiver side is guaranteed posted, therefore MPI will do less and faster
- User must ensure that receive has been posted
- Enables faster communication, but needs implicit synchronization

# Checking the UMQ with MPI\_Probe/Iprobe

They are expensive operations

Allow polling of incoming messages without actually receiving them.

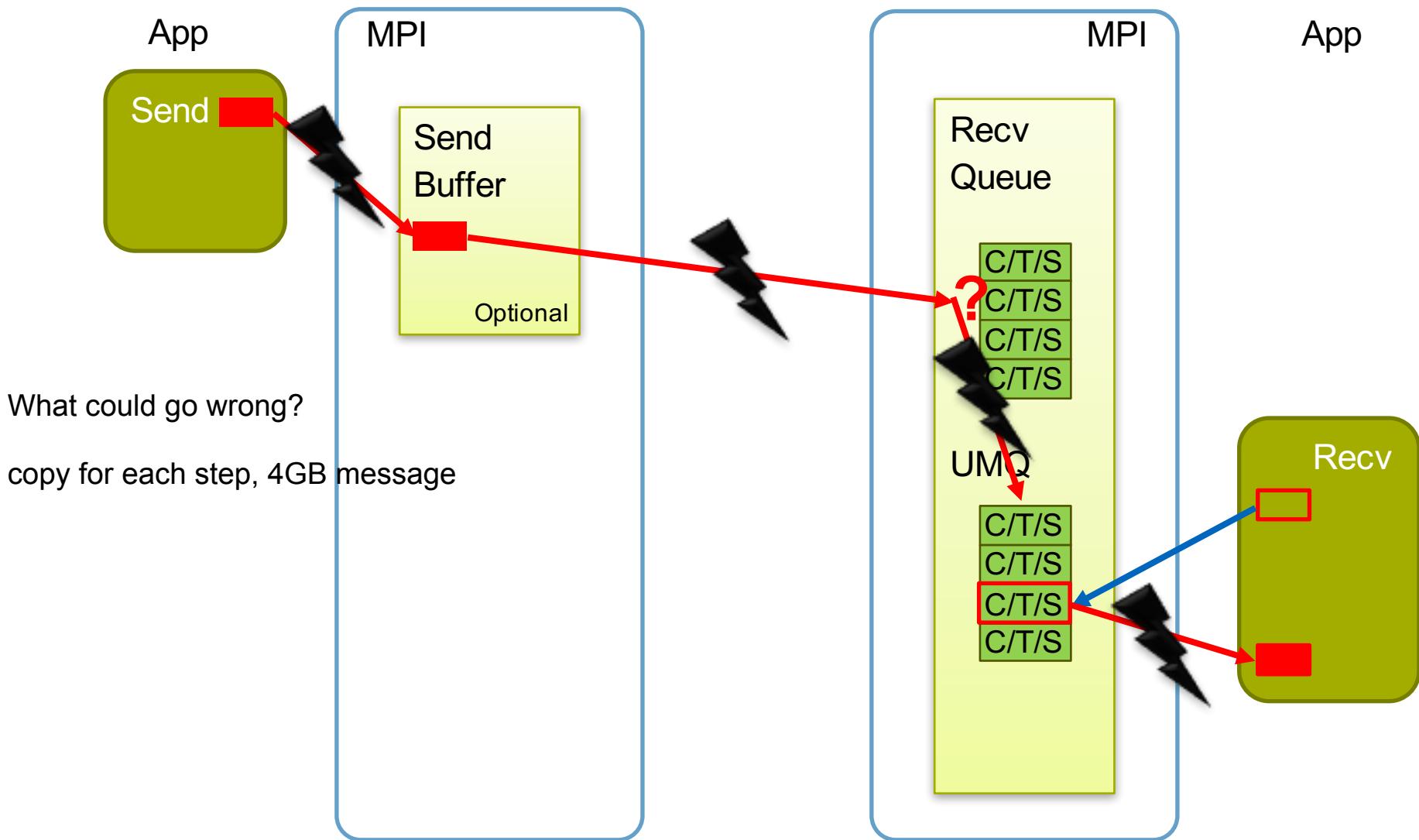
- **MPI\_Probe**
  - Blocks until a matching message was found
  - Input: source, tag, communicator
  - Output: status object
- **MPI\_Iprobe** I: immediatly, meaning no blocking
  - Terminates after checking whether a matching messages is available
  - Input: source, tag, communicator
  - Output: flag (message found yes/no) and status object

## Usage

- Both operations inspect the Unexpected message queue
- Allows to wait for a message before issuing the receive
- Useful for wildcard receives
  - Issue probe call with wildcard
  - Call returns concrete source/tag pair
- **MPI\_Iprobe** can be used to overlap wait time with useful work.

# Issue: Copy Overhead

Sender

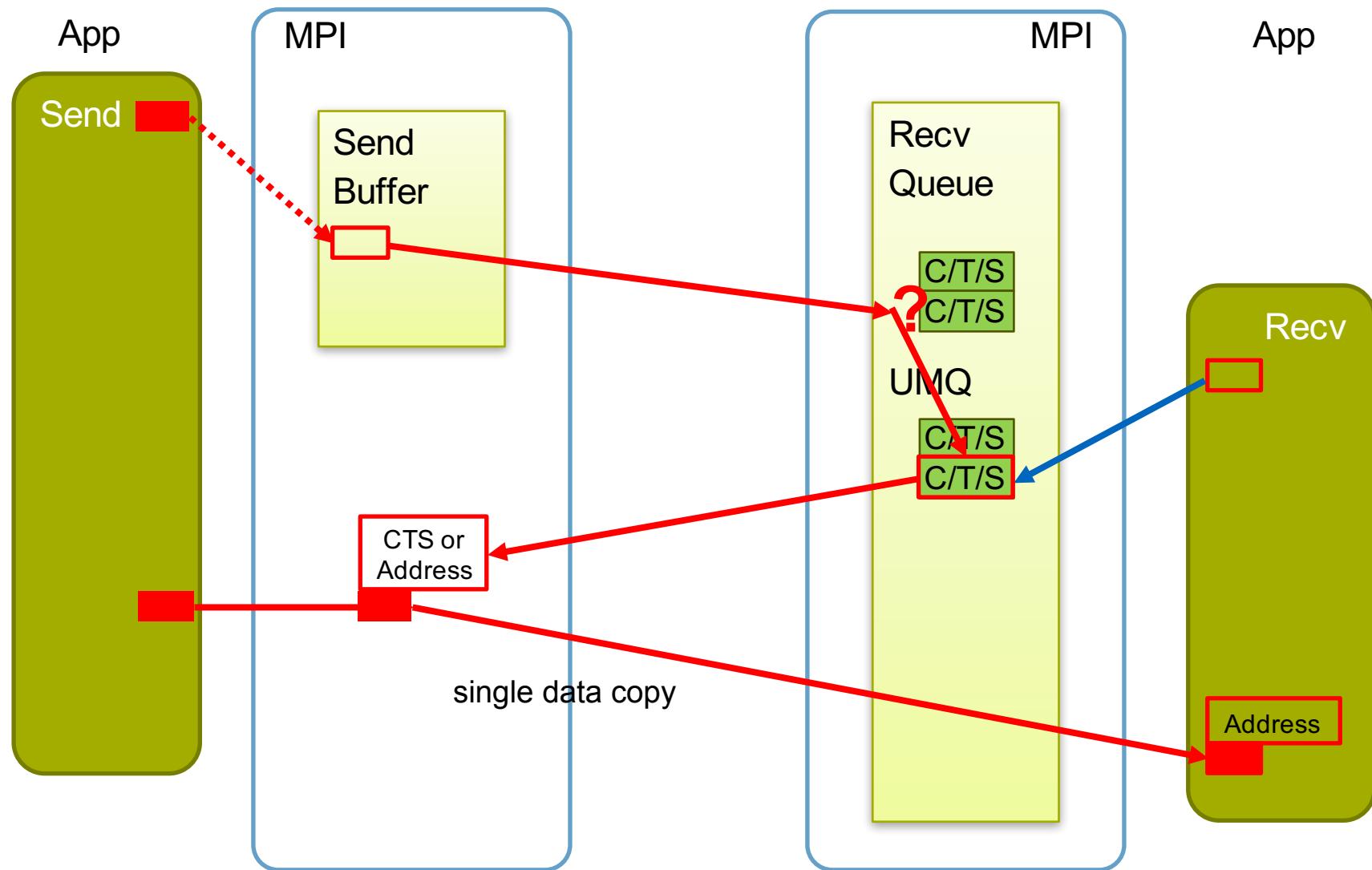


# Rendevouz Protocol

Sender

Send the envelope, rather the message

Receiver



# Eager vs. Rendezvous Protocol

Eager protocol No handshake

- Send messages directly
  - Advantage: avoids extra handshake
  - Disadvantage: adds extra copies
- Suitable for short messsages with low copy overhead

Rendezvous protocol

- Send header to retrieve address, then send message
  - Advantage: Can deposit message directly
  - Disadvantage: extra handshake
- Suitable for long messsages with high copy overhead

How does MPI decide which protocol to use? The size!

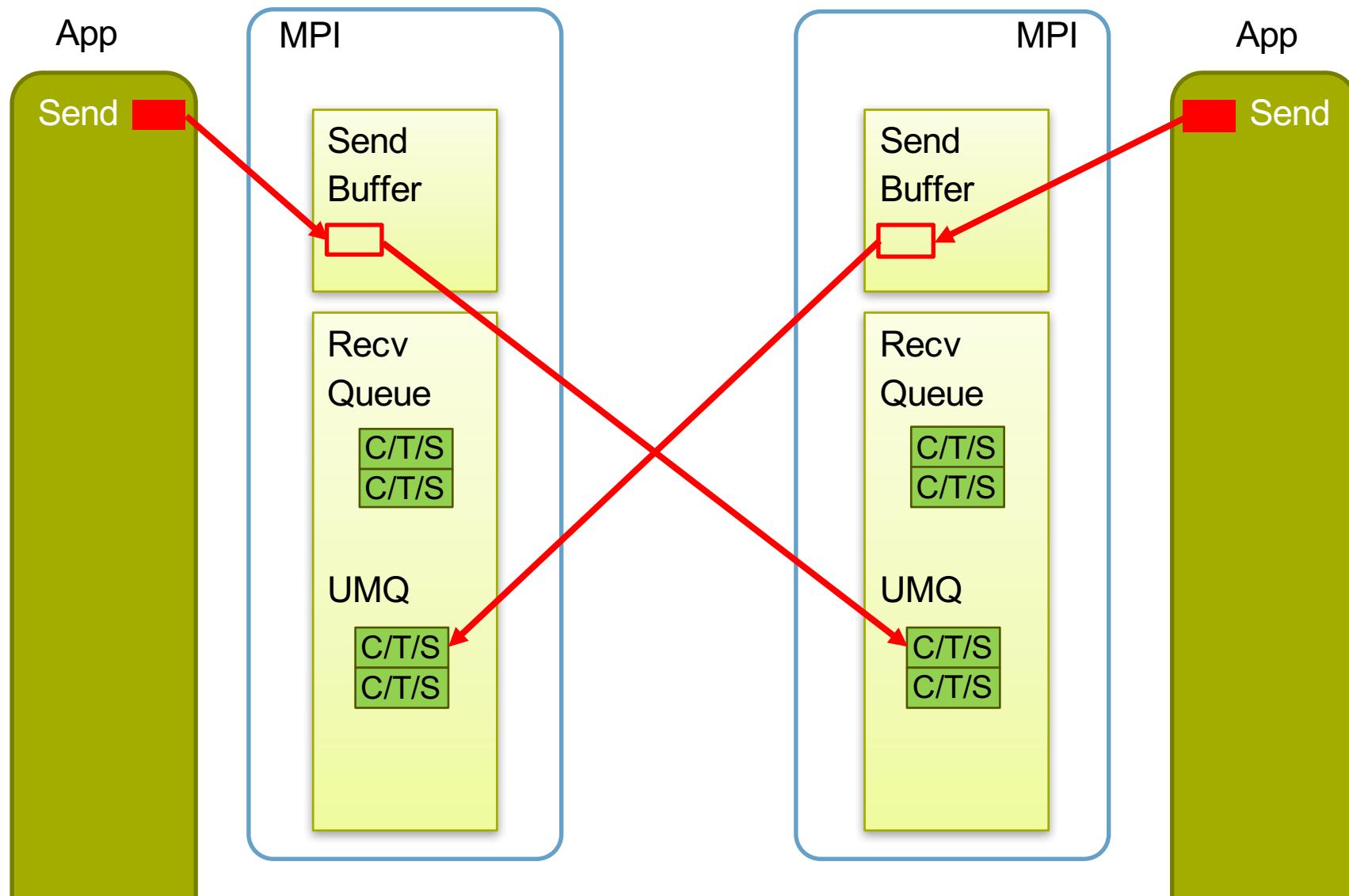
Cross-over point

- MPI implementations typically switch protocol based on message size
- Typically defined as eager limit
- Depends on system setup
- Can often be modified

# Deadlock with Rendevouz Protocol

Sender

Receiver



# Common Operation: Bidirectional Exchange

Transfer data elements between two “adjacent”

- Same or similar data exchanged between two MPI processes
- Example: halo exchange



MPI\_Send( to: 1 )                            MPI\_Send( to: 0 )  
MPI\_Recv( from: 1 )                            MPI\_Recv( from: 0 )



Problem: deadlock possible

- MPI\_Send may need resources on receiver side

Changing the order of send and receive

MPI\_Send( to: 1 )                            MPI\_Recv( from: 0 )  
MPI\_Recv( from: 1 )                            MPI\_Send( to: 0 )

# Common Operation: Bidirectional Exchange

## Transfer data elements between two “adjacent”

- Same or similar data exchanged between two MPI processes
  - Example: halo exchange



## MPI Send( to: 1 )

MPI Recv( from: 1 )

**MPI\_Send( to: 0 )**

**MPI\_Recv( from: 0 )**

## Problem: deadlock possible

- MPI\_Send may need resources on receiver side

# Changing the order of send and receive

## MPI\_Send( to: 1 )

## MPI\_Recv( from: 1 )

sequential performance.

MPI Send( to: 0 )

## Problems:

- **Serialization**

# Bidirectional Send/Recv

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

Combined blocking send and receive

- Can be matched with regular send/recv calls
- Can be matched with other Sendrecv calls with other destinations

```
int MPI_Sendrecv_replace(void *buf, int count, MPI_Datatype datatype,  
send out the buffer and use the same buffer to receive  
int dest, int sendtag,  
int source, int recvtag,  
MPI_Comm comm, MPI_Status *status)
```

Same principle, but combined send and receive buffer

- Requires same message types

# Common Operation: Bidirectional Exchange

Transfer data elements between two “adjacent”

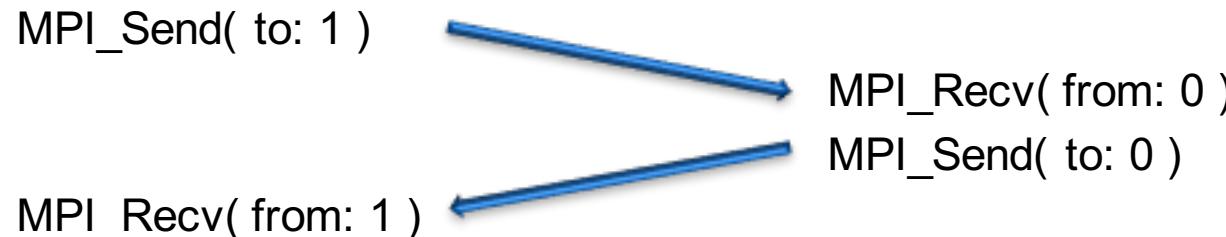
- Same or similar data exchanged between two MPI processes
- Example: halo exchange



Problem: deadlock possible

- `MPI_Send` may need resources on receiver side

Changing the order of send and receive



Problems:

- Serialization
- What to do with more than two communication partners?

# Non-blocking Operations

Split operations into start and completion

- Starting an operation finishes immediately
- Completion
  - Can be tested for
  - Can be waited for
- Can complete other work in between
- Can start multiple operations

Useful for long operations

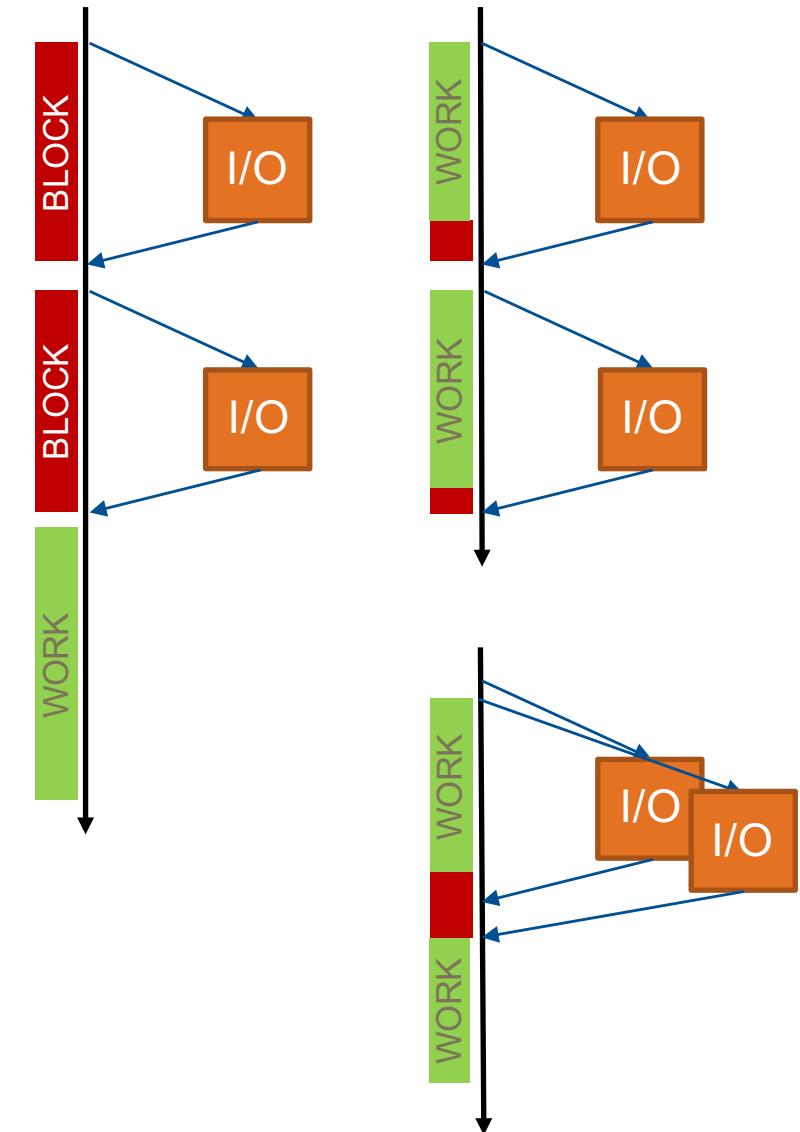
- Waiting for events
- Long I/O operations due to data size

Completion options

- Separate wait operations
- Polling
- Interrupts

General issue: progress

MPI is a library, keep in mind we need extra thread to guarantee progressing



# Non-blocking P2P Operations in MPI

Initiation routines as counterparts to blocking P2P operations

- Similar arguments, same semantics
- Can be matched with blocking operations
- Returns request object to track progress

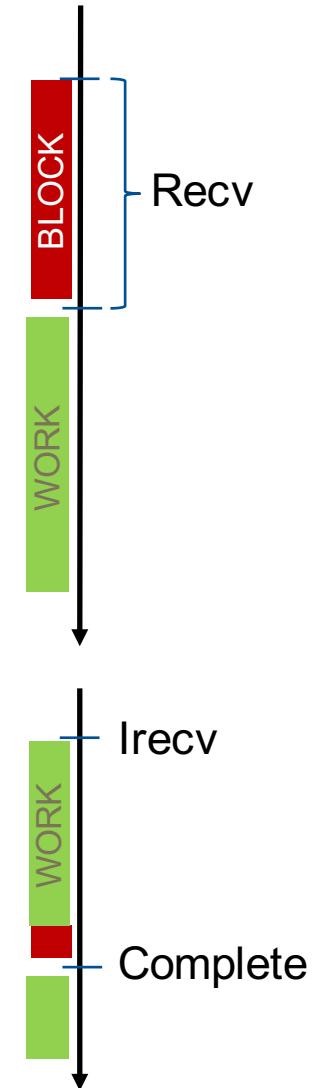
Initiation call only starts operation

- Completion is indicated later
- Send and receive buffers are off limits before completion
- Semantics of operation

## MPI\_Request

- Request object
- User allocates object, but MPI maintains state
- Initiated operations must be completed

remember to clean up the memory leak



# Non-blocking P2P Operations in MPI

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

IN buf: *Send buffer*

IN count: *Number of elements*

IN dtype: *Data type*

IN dest: *Receiver*

IN tag: *Tag*

IN comm: *Communicator*

OUT request: *Reference to request*

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

IN buf: *receive buffer*

IN count: *number of entries in receive buffer*

IN dtype: *data type*

IN dest: *sender*

IN tag: *tag*

IN comm: *communicator*

OUT request: *reference to request*

# Non-Blocking Send Variants

## **MPI\_ISEND**

- Standard send operation, no extra synchronization
- Sender side handling implementation defined

## **MPI\_IBSEND**

- Buffered Send
- Force the use of a send buffer
- Completes immediately, but costs resources

## **MPI\_ISSEND**

- Synchronous send
- Only completes once receive has started
- Adds extra synchronization, but can be costly

## **MPI\_IRSEND**

- Ready Send
- User must ensure that receive has been posted at Irsend time
- Enables faster communication, but needs implicit synchronization

# Completion Operations

Option 1: Blocking completion

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

INOUT request: *request*

OUT status: *status of completed operation*

Returns if ...

request is complete

request is **MPI\_REQUEST\_NULL** dummy request

Sets request to **MPI\_REQUEST\_NULL**

Frees all resources associated with request

# A Simple Example

```
{  
    MPI_Request req;  
    MPI_Status status;  
    int msg[10];  
    ...  
    MPI_Irecv(msg, 10, MPI_INT, MPI_ANY_SOURCE, 42,  
              MPI_COMM_WORLD, &req);  
    ...  
    <do work>  
    ...  
    MPI_Wait(&req, &status);  
    printf("Processing message from %i\n",  
          status.MPI_SOURCE);  
    ...  
}
```

# Completion Operations

Option 2: Non-Blocking/Polling completion

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

INOUT request: *request*

OUT flag: flag, whether complete or not

OUT status: *status of completed operation*

Returns if immediately

flag set to True if request is complete  
request is **MPI\_REQUEST\_NULL**

If completed,

Sets request to **MPI\_REQUEST\_NULL**

Frees all resources associated with request

# A Simple Non-Blocking Example

```
{  
    MPI_Request req;  
    MPI_Status status;  
    int msg[10], flag;  
    ...  
    MPI_Irecv(msg, 10, MPI_INT, MPI_ANY_SOURCE, 42,  
              MPI_COMM_WORLD, &req);  
    do  
    {  
        ...  
        <do work>  
        ...  
        MPI_Test(&req, &flag, &status);  
    } while (flag==0);  
  
    printf("Processing message from %i\n",  
          status.MPI_SOURCE);  
}
```

# Extended Wait and Test Operations

## `MPI_Wait`

- Blocking wait for one request
- Output: status object

## `MPI_Waitall`

- Blocking wait for several requests
  - Input: array of requests
  - Some can be `MPI_REQUEST_NULL`
- Returns once all are complete
  - Output: array of status objects

## `MPI_Waitany` (`MPI_Waitsome`)

- Blocking wait for several requests
  - Input: array of requests
  - Some can be `MPI_REQUEST_NULL`
- Returns once at least one is complete
  - Output: “completed” index (array)
  - Output: (array of) status object(s)

## `MPI_Test`

- Checks completion of one request
- Output: flag, True if complete
- Output: status object

## `MPI_Testall`

- Checks completion of several requests
  - Input: array of requests
  - Some can be `MPI_REQUEST_NULL`
- Returns immediately
  - Output: flag, True if all complete
  - Output: array of status objects

## `MPI_Testany` (`MPI_Testsome`)

- Checks completion of several requests
  - Input: array of requests
  - Some can be `MPI_REQUEST_NULL`
- Returns immediately
  - Output: flag, True if at least one complete
  - Output: “completed” index (array)
  - Output: (array of) status object(s)

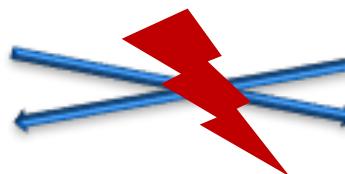
# Example: Bidirectional Exchange

Transfer data elements between two “adjacent”

- Same or similar data exchanged between two MPI processes
- Example: halo exchange



MPI\_Send( to: 1 )      MPI\_Send( to: 0 )  
MPI\_Recv( from: 1 )      MPI\_Recv( from: 0 )



```
MPI_Request req[2];
MPI_Status  status[2];
int msg_in[10], msg_out[10];

MPI_Isend(msg_out, 10, MPI_INT, 1, 42, MPI_COMM_WORLD, &(req[0]));
MPI_Irecv(msg_in, 10, MPI_INT, 1, 42, MPI_COMM_WORLD, &(req[1]));

MPI_Waitall(2, req, status);

/* status[1] is the receive status */
```

# The Need for Global Coordination

Point to point operations are sufficient to implement all communication

- Basic unit of data exchange: one message between two processes

More complex communication structure can be built

Disadvantages

- Complex to implement
- Many standard patterns that are present on many applications
- Tied to the processes calling them

MPI defines collective operations

- Basic primitives
- Affect a group of processes how do you group processes
- Again tied to a collector
- Enables implicit communication through nodes

# Collective Operations

## Properties

- Must be executed by all processes of the communicator
- Must be executed in the same sequence User responsibility
- Collective operations can be blocking or non-blocking operations
- We will discuss blocking first
- Arguments across all MPI processes must be consistent (or equal)

if you have group operation, you have to match!

MPI provides three classes of collective operations

- Synchronization
  - Barrier
- Communication, e.g., Dealing with data
  - Broadcast
  - Gather
  - Scatter
- Reduction, e.g., sum, multiplication operation
  - Global value returned to one or all processes.
  - Combination with subsequent scatter.
  - Parallel prefix operations

# MPI\_Barrier

no process leave the barrier, until all processes arrived Barrier

```
int MPI_Barrier (MPI_Comm comm)
```

IN comm: *Communicator*

This operation synchronizes all MPI processes, i.e., no MPI process can return from the call until all MPI processes have called it.

```
#include "mpi.h"

main (int argc,char *argv[ ]){
    ...
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    MPI_Barrier(MPI_COMM_WORLD); wait for all processes, can be costly
    ...
}
```

# Broadcast Communication: MPI\_Bcast

```
int MPI_Bcast (void *buf, int count, MPI_Datatype dtype,  
               int root, MPI_Comm comm)
```

IN buf: *Adress of send/receive buffer*

IN count: *Number of elements*

IN dtype: *Data type*

IN root: *Rank of send/broadcasting MPI process*

IN comm: *Communicator* root is inside communicator

The contents of the send buffer is copied to all other MPI processes

<sup>M</sup><sub>a</sub><sup>t</sup> vs P2P operations

- No tag
- Number of elements sent must be collective is not synchronized equal to number of elements received

The routines do not necessarily synchronize processes wrt. P2P

# Example

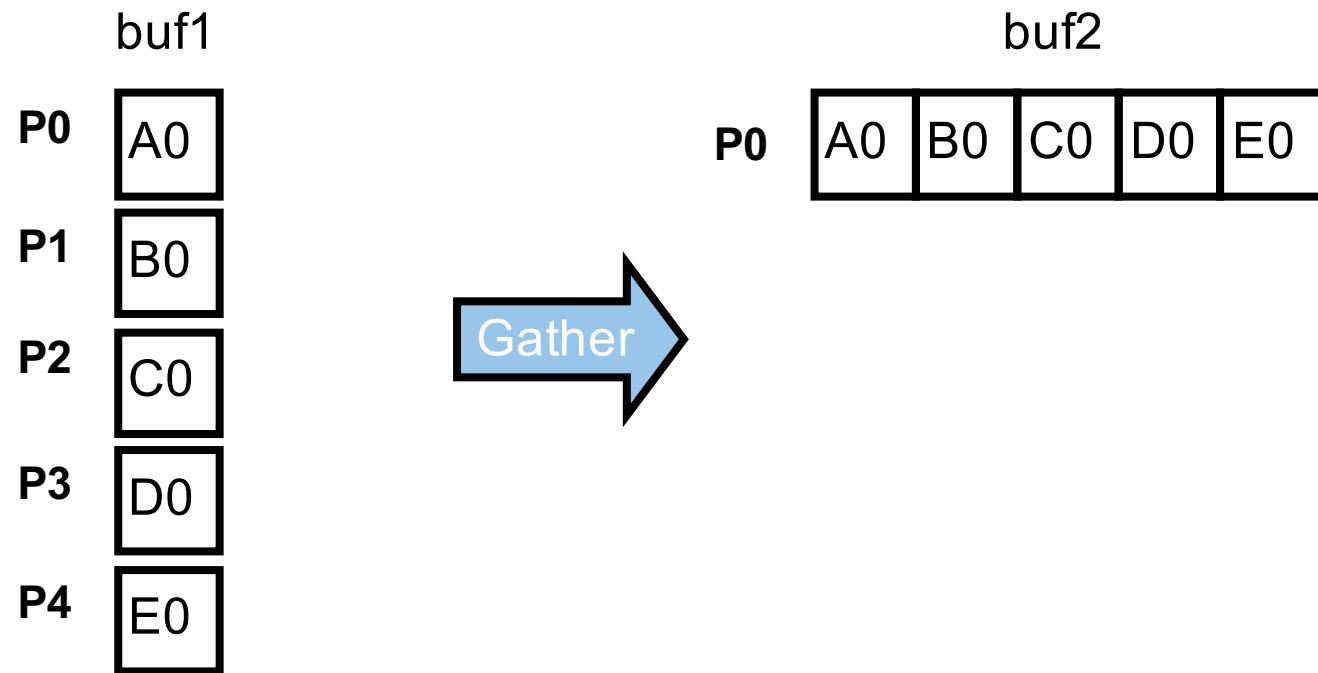
```
switch (rank) {  
case 0:  
    MPI_Bcast (buf1, cout, tp, 0, comm);  
  
    break;  
case 1:  
  
    MPI_Bcast (buf1, ct, tp, 0, comm);  
  
    break;  
case 2:  
  
    MPI_Bcast (buf1, ct, tp, 0, comm);  
    break;  
}
```

# Example

```
switch (rank) {  
case 0:  
    MPI_Bcast (buf1, ct, tp, 0, comm);  
    MPI_Send (buf2, ct, tp, 1, tag, comm);  
    break;  
case 1:  
    MPI_Recv (buf2, ct, tp, MPI_ANY_SOURCE, tag, ...);  
    MPI_Bcast (buf1, ct, tp, 0, comm);  
    MPI_Recv (buf2, ct, tp, MPI_ANY_SOURCE, tag, ...);  
    break;  
case 2:  
    MPI_Send (buf2, ct, tp, 1, tag, comm);  
    MPI_Bcast (buf1, ct, tp, 0, comm);  
    break;  
}
```

# Gather Operation

Get data from all MPI processes into one local array



# MPI\_Gather

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

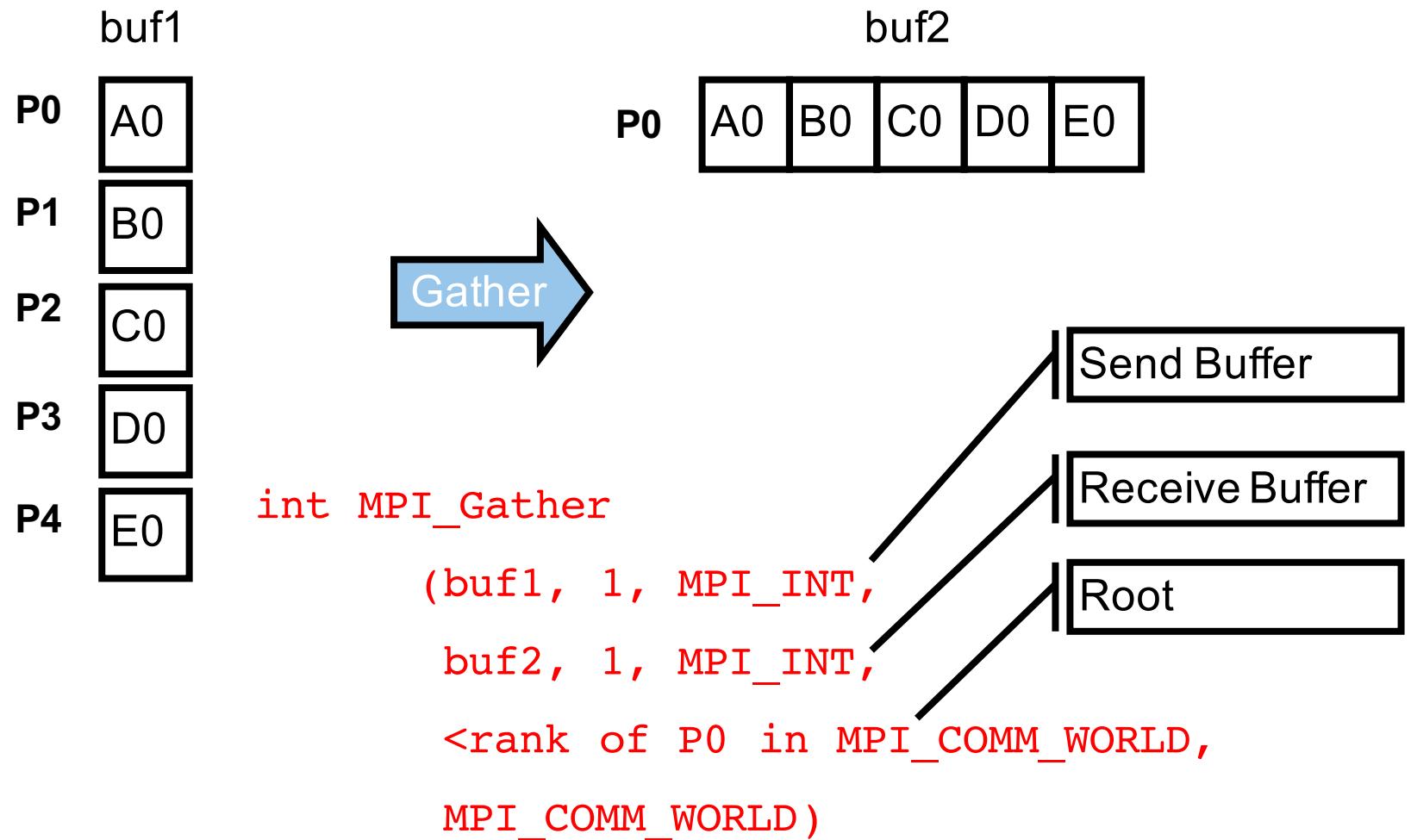
|               |   |
|---------------|---|
| IN sendbuf:   | <i>Send buffer</i>  |
| IN sendcount: | <i>Number of elements to be sent to the root</i>                |
| IN sendtype:  | <i>Data type</i>  |
| OUT recvbuf:  | <i>Receive buffer</i>   |
| IN recvcount: | <i>Number of elements to be received from each MPI process.</i> |
| IN recvtype:  | <i>Data type</i>  |
| IN root:      | <i>Rank of receiving MPI Process</i>                            |
| IN comm       | <i>Communicator</i>   |

The root receives from all processes the data in the send buffer.

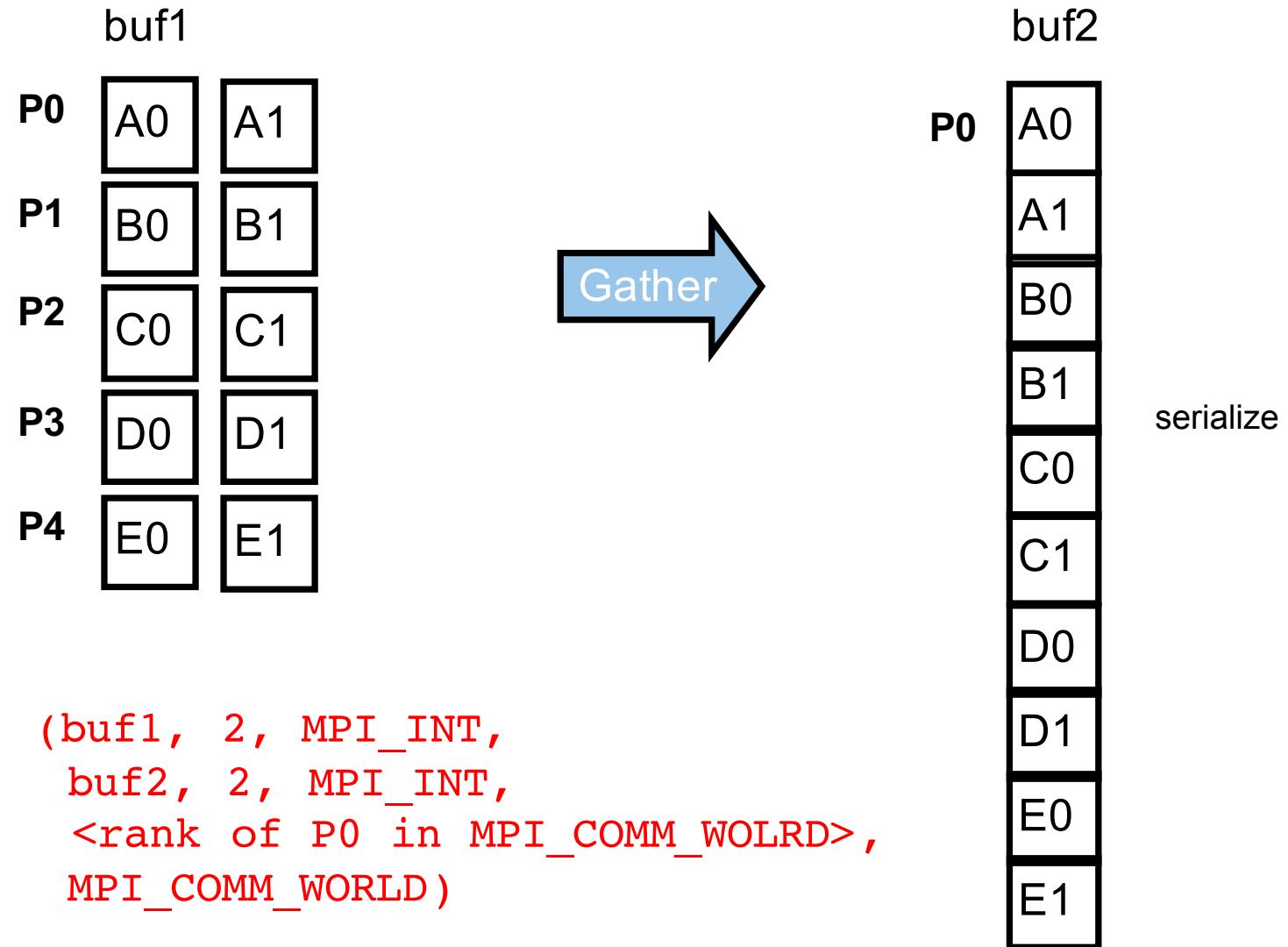
It stores the data in the receive buffer ordered by the process number of the senders.

# Gather Operation

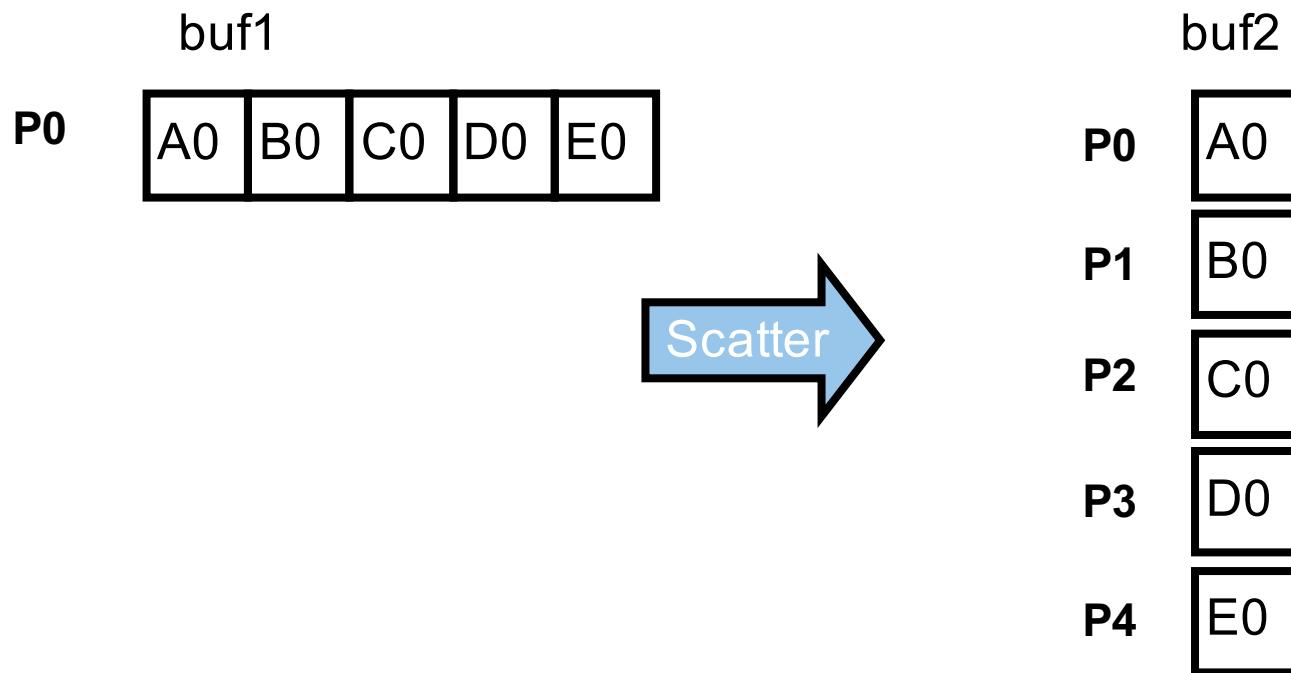
Get data from all MPI processes into one local array



# Gather Example



# Scatter



# MPI\_Scatter

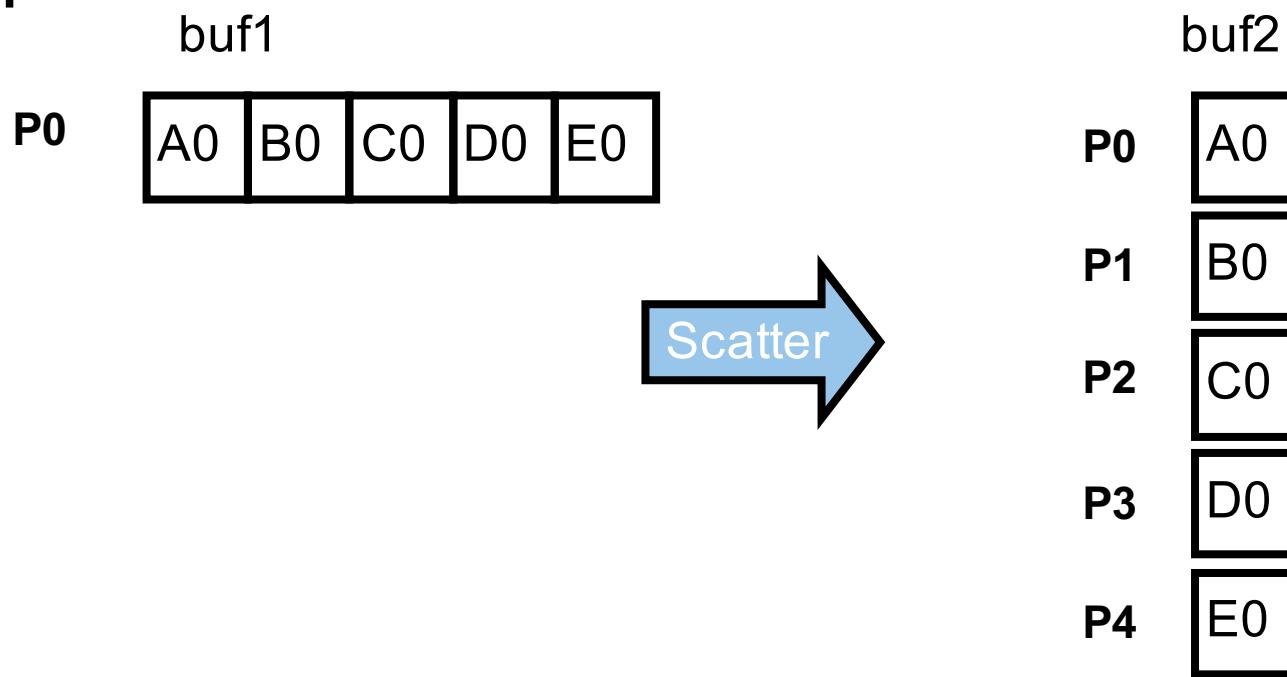
```
int MPI_Scatter (void *sendbuf, int sendcount, MPI_Datatype sendtype,  
                 void* recvbuf, int recvcount, MPI_Datatype recvtype,  
                 int root, MPI_Comm comm)
```

|               |   |
|---------------|---|
| IN sendbuf:   | <i>Send buffer</i>  |
| IN sendcount: | <i>Number of elements sent to each MPI process</i>        |
| IN sendtype:  | <i>Data type</i>  |
| OUT recvbuf:  | <i>Receive buffer</i>                                     |
| IN recvcount: | <i>Number of elements to be received by a MPI process</i> |
| IN recvtype:  | <i>Data type</i>  |
| IN root:      | <i>Rank of sending/scattering MPI process</i>             |
| IN comm:      | <i>Communicator</i>                                       |

The root sends a part of its send buffer to each process.

Process  $k$  receives  $sendcount$  elements starting with  $sendbuf+k*sendcount$ .

# Scatter



```
int MPI_Scatter (buf1, 1, MPI_INT,  
                 buf2, 1, MPI_INT,  
<rank of P0 in MPI_COMM_WORLD,  
MPI_COMM_WORLD)
```

# Other Collective Communication Routines

## **MPI\_Gatherv**

- Gather operation with different number of data elements per process

## **MPI\_Allgather**

- Gather operation with broadcast to all processes

## **MPI\_Allgatherv**

- Gather operation with different number of data elements AND broadcast

## **MPI\_Scatterv**

- Scatter operation with different number of data elements per process

# MPI\_Gatherv

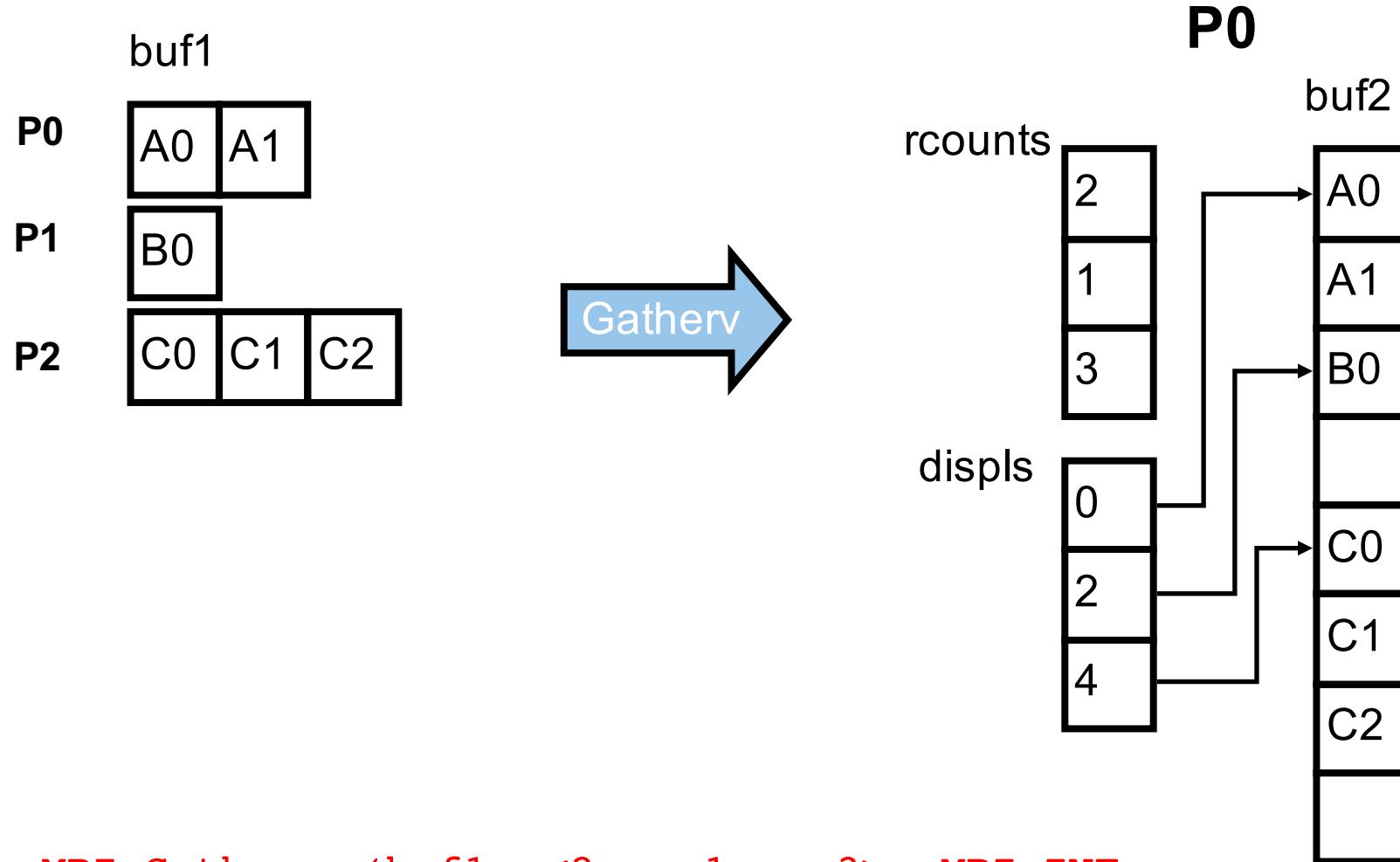
```
int MPI_Gatherv (void *sendbuf, int sendcount, MPI_Datatype sendtype,  
                 void* recvbuf, int *recvcount, int *displs,  
                 MPI_Datatype recvtype, int root, MPI_Comm comm)
```

|               |   |
|---------------|---|
| IN sendbuf    | <i>Send buffer</i>  |
| IN sendcount  | <i>Number of elements sent</i>  |
| IN sendtype   | <i>Send type</i>  |
| OUT recvbuf   | <i>Receive buffer</i>   |
| IN recvcounts | <i>Array with the number of elements to be received from each MPI process</i> |
| IN displs     | <i>Array with the first index in the receive buffer for each MPI process</i>  |
| IN recvtype   | <i>Receive data type</i>  |
| IN Root       | <i>Receiving/Gathering MPI process rank</i>                                   |
| IN comm       | <i>Communicator</i>   |

In contrast to MPI\_Gather, each process can send a different number of elements

The individual messages are stored according to *displs* in the receive buffer

# MPI\_Gatherv



```
int MPI_Gatherv (buf1, <2 or 1 or 3>, MPI_INT,  
                  buf2, rcounts, displs, MPI_INT,  
<rank of P0 in MPI_COMM_WORLD>,  
MPI_COMM_WORLD)
```

# Other Collective Communication Routines

## **MPI\_Gatherv**

- Gather operation with different number of data elements per process

## **MPI\_Allgather**

- Gather operation with broadcast to all processes

## **MPI\_Allgatherv**

- Gather operation with different number of data elements AND broadcast

## **MPI\_Scatterv**

- Scatter operation with different number of data elements per process

## **MPI\_Alltoall**

- Every process sends to every other process (different data)

## **MPI\_Alltoallv**

- All to all with different numbers of data elements per process

## **MPI\_Alltoallw**

- All to all with different numbers of data elements and types per process

# MPI\_Alltoall

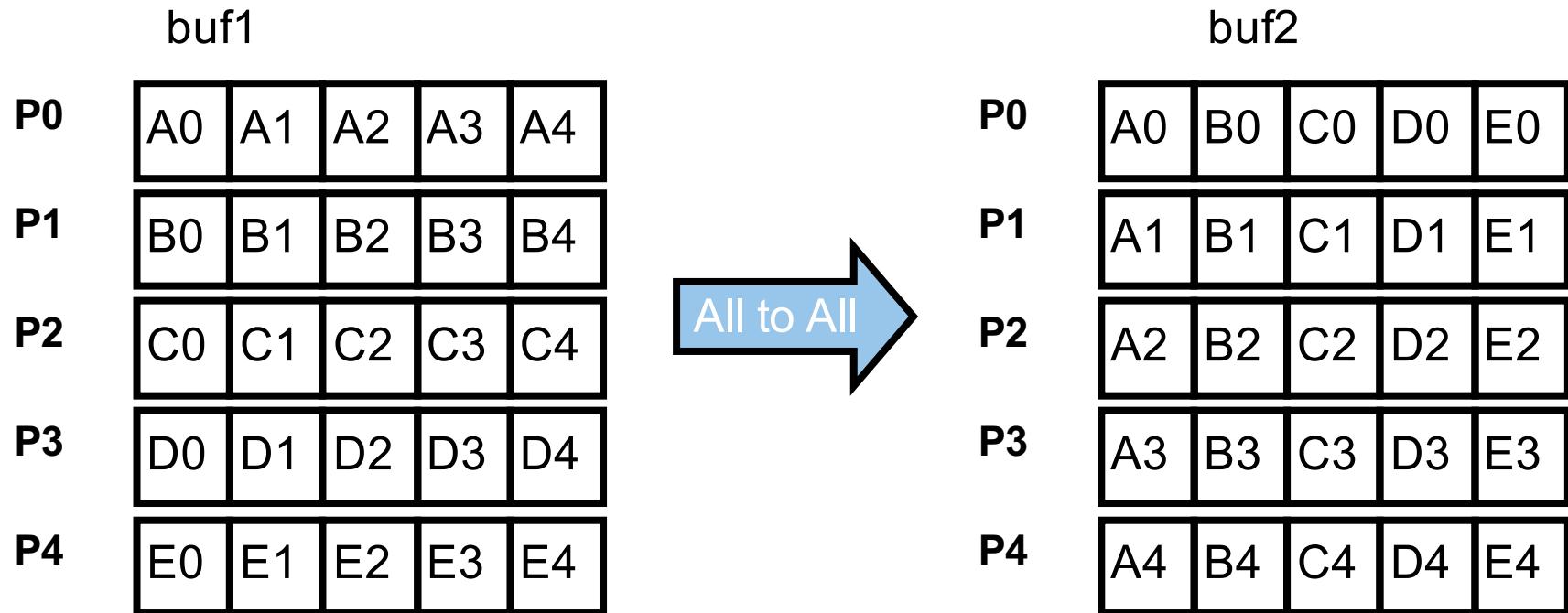
```
int MPI_Alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype,  
                 void* recvbuf, int recvcount, MPI_Datatype recvtype,  
                 MPI_Comm comm)
```

|              |   |
|--------------|---|
| IN sendbuf   | <i>Send buffer</i>  |
| IN sendcount | <i>Number of elements sent</i>                            |
| IN sendtype  | <i>Send type</i>  |
| OUT recvbuf  | <i>Receive buffer</i>                                     |
| IN recvcount | <i>Number of elements to be received by a MPI process</i> |
| IN recvtype  | <i>Data type</i>  |
| IN root      | <i>Rank of sending/scattering MPI process</i>             |
| IN comm      | <i>Communicator</i>                                       |

Sends data from each MPI process to each other MPI process

Warning: this is often a scaling bottleneck

# MPI\_Alltoall



```
int MPI_Alltoall (buf1, 1, MPI_INT,  
                  buf2, 1, MPI_INT,  
                  MPI_COMM_WORLD)
```

# Reductions

The data of the processes are combined via a specified operation

- Predefined operators, like '+'
- User defined operators (see `MPI_Op_create` and `MPI_Op_free`)

Often implemented in optimized way, e.g., using tree structures

## `MPI_Reduce`

- Reduce elements from all processes based on operation

## `MPI_Allreduce`

- Reduction followed by broadcast

## `MPI_Reducescatter_block`

- Reduction followed by a scatter operation (equal chunks)

## `MPI_Reducescatter`

- Reduction followed by a scatter operation (varying chunks)

## `MPI_Scan`

- Reduction using a prefix operation

# Example: Scalar Reduction

```
s=0  
for (i=1; i<n; i++)  
    s=s+a[i]
```



```
s=0  
for (i=0; i<local_n; i++){  
    s=s+a[i]  
}  
MPI_Reduce(s, s1, 1,  
            MPI_INT, MPI_SUM, P0,  
            MPI_COMM_WORLD)  
s=s1
```

# MPI\_Reduce

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

- IN sbuf: *Send buffer*
- OUT rbuf: *Receive buffer*
- IN count: *Number of elements*
- IN dtype: *Data type*
- IN op: *Operation*
- IN root: *Rank of MPI process receiving the reduced results*
- IN comm: *Communicator*

Combines the elements in the send buffer according to “op”  
Delivers the result to root

# Non-Blocking Collectives (since MPI 3.0)

Same concept as non-blocking P2P

- Defined for all collective operations
  - Include `MPI_Ibarrier`, the non-blocking barrier
  - Enables overlap with synchronization
- Returns request object
- Request can be used in `MPI_Wait` and `MPI_Test` operations

Cannot be mixed with blocking collectives

Example:

```
int MPI_Ibcast(void* buffer, int count, MPI_Datatype datatype,
               int root, MPI_Comm comm, MPI_Request *request)
```

INOUT buffer starting address of buffer (choice)

IN count number of entries in buffer (non-negative integer)

IN datatype data type of buffer (handle)

IN root rank of MPI process acting as broadcast root (integer)

IN comm communicator (handle)

OUT request communication request (handle)

# Communicators and MPI\_COMM\_WORLD

Central concept in MPI: Communicators

- Group of MPI processes
- Communication context

Any communication operations is done in the context of a communicator

- Argument to many functions
- Addressing is relative to a communicator
  - Each process has a rank in a communicator
  - Ranks go from 0 to N-1 are fixed
  - A process can have different ranks in different communicators
- Communication across communicators are not possible
  - Different context
  - But: one MPI process can be in multiple communicators

Default communicators:

- **MPI\_COMM\_WORLD**: all initial MPI processes
- **MPI\_COMM\_SELF**: contains only the own MPI process

Other communicators can be derived

# Creating New Communicators



Why want anything but MPI\_COMM\_WORLD

- Creation of subgroups for collective communication
- Isolation between communication operations

Isolation important for libraries

- Different libraries should use different communicators
- Avoids “cross-talk” and side effects

Typical strategy for libraries

- Libraries get a communicator passed at initialization
- This is often MPI\_COMM\_WORLD
- The library then clones the communicator and stores the handle

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)
    IN comm      communicator(handle)
    OUT newcomm  copy of comm (handle)
```

Collective operation

Also exists as `MPI_Comm_Idup`

# Creating Subcommunicators

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

|             |  |
|-------------|--|
| IN comm     | <i>Parent communicator</i>                             |
| IN color    | <i>Subset color</i>                                    |
| IN key      | <i>Key to determine rank order in new communicator</i> |
| OUT newcomm | <i>New communicator</i>                                |

Creates new communicator(s)

- Collective operation over parent communicator
- All MPI processes that pass the same **color**, will be in same new communicator
- **Key** argument determines the rank order in the new communicator

MPI processes can opt out

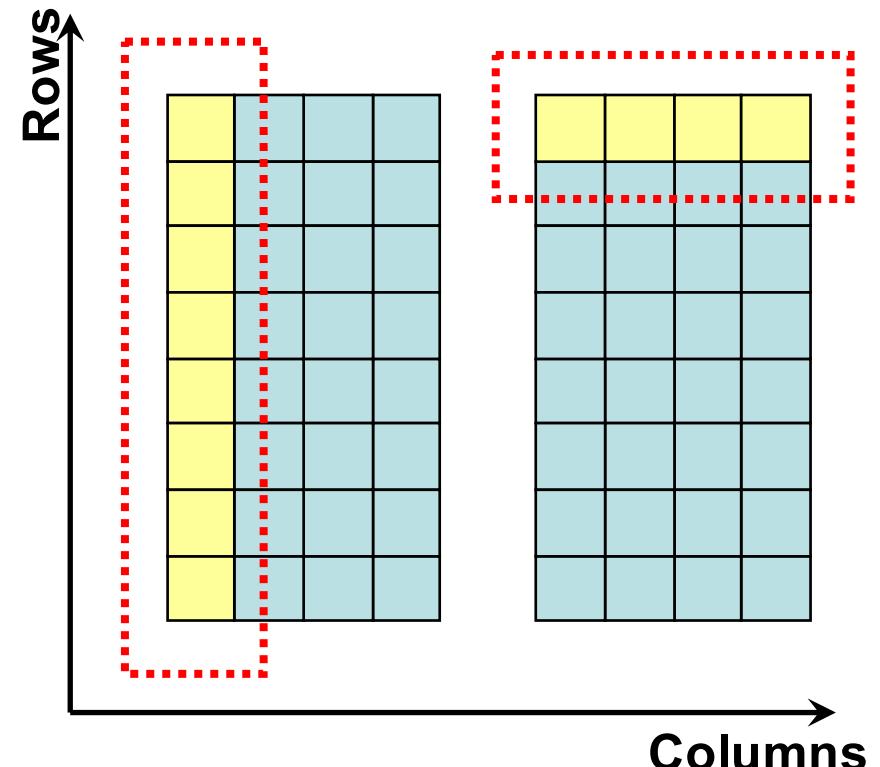
- Pass **MPI\_UNDEFINED** as color
- Will return **MPI\_COMM\_NULL** as new communicator

Communicators should be freed, when no longer in use

```
int MPI_Comm_free(MPI_Comm *comm)
```

# Example: Row and Column Communicators

```
int rank, size, col, row;  
MPI_Comm row_comm, col_comm;  
  
MPI_Comm_size (MPI_COMM_WORLD, &size);  
MPI_Comm_rank (MPI_COMM_WORLD, &rank);  
  
Row = rank % N;  
Col = rank / N;  
  
MPI_Comm_split(MPI_COMM_WORLD,  
                row, col, &row_comm);  
  
MPI_Comm_split(MPI_COMM_WORLD,  
                col, row, &col_comm);  
  
...  
  
MPI_Bcast(buf, 1, MPI_INT, 0, col_comm);
```



# Summary

MPI as the most widely used HPC standard for message passing

- Currently in version 3.1 with wide range of functionality
- Continues to evolve through the open standardization body MPI Forum

Basic message protocols

- Receive queues and Unexpected Message Queues
- Eager vs. Rendezvous communication

Non-blocking communication

- Functions return `MPI_Request` object
- Can be waited or tested for completion

Collective operations

- Operations involving all MPI processes in a communicator
- Enable faster and more efficient implementations

Communicator creation

- Duplication enables library isolation
- Communicator splitting to create subcommunicators