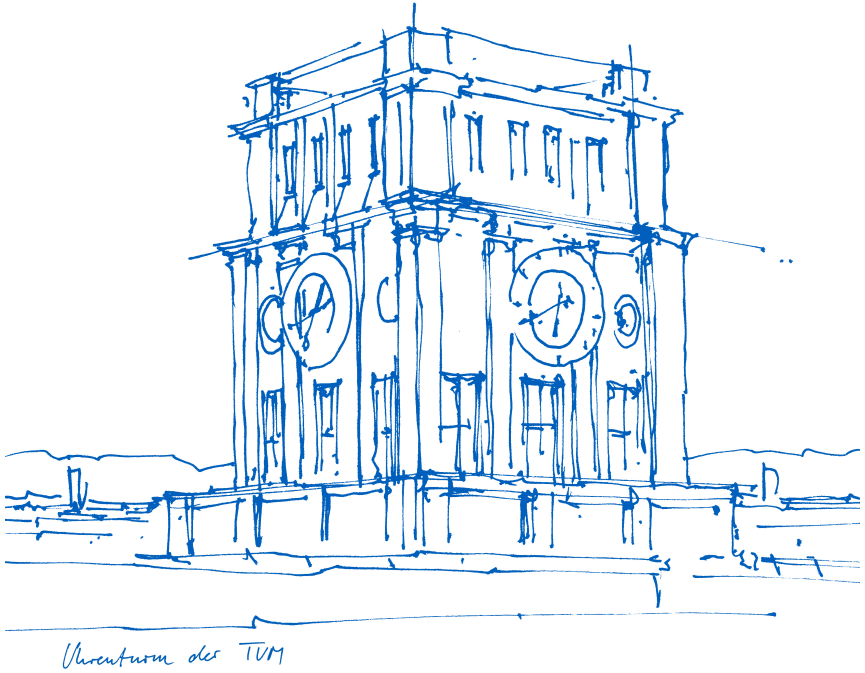
Lecture IN-2147
Parallel Programming

SoSe 2018

Martin Schulz
Exercises: Amir Raoofy
Technische Universität München
Fakultät für Informatik

Lecture 8:
MPI Part 2
Network Issues



Administrative Topics

Exam:

July 24th, 2018, 16:00 to 17:30 in MW 2001

please double check in TUM-Online

Please register in TUM-Online by June 6th

Repetition Exam:

October 5th, 2018, 11:00 to 12:30 in MW 1801

please double check in TUM-Online

Please register in TUM-Online by September 24th

Lecture Evaluation (June 6th to June 20th)

You will get an online link, please fill it out

We will provide an extra 15min time on June 18th

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

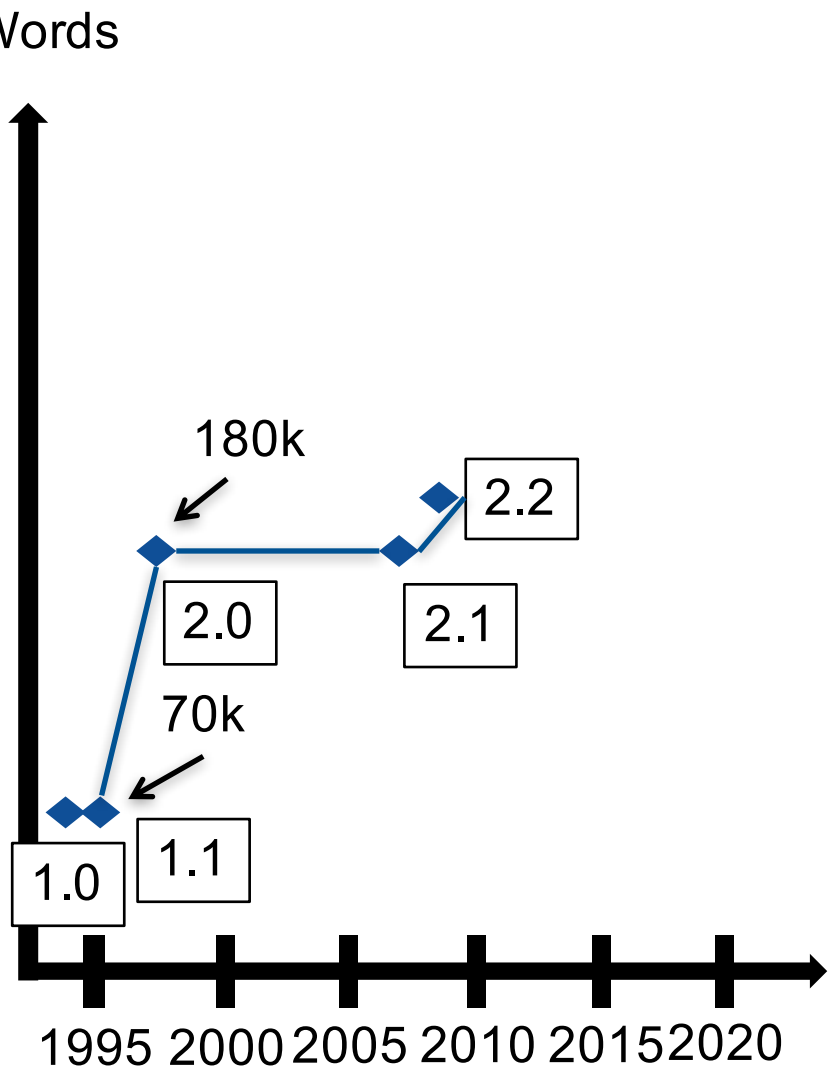
Persistent operations

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

Neigborhood collectives

MPI Tool Information Interface

One sided communication enhancements

Large data counts (messages more with more than 32bit count)

Topology aware communicator creation

Noncollective communicator creation

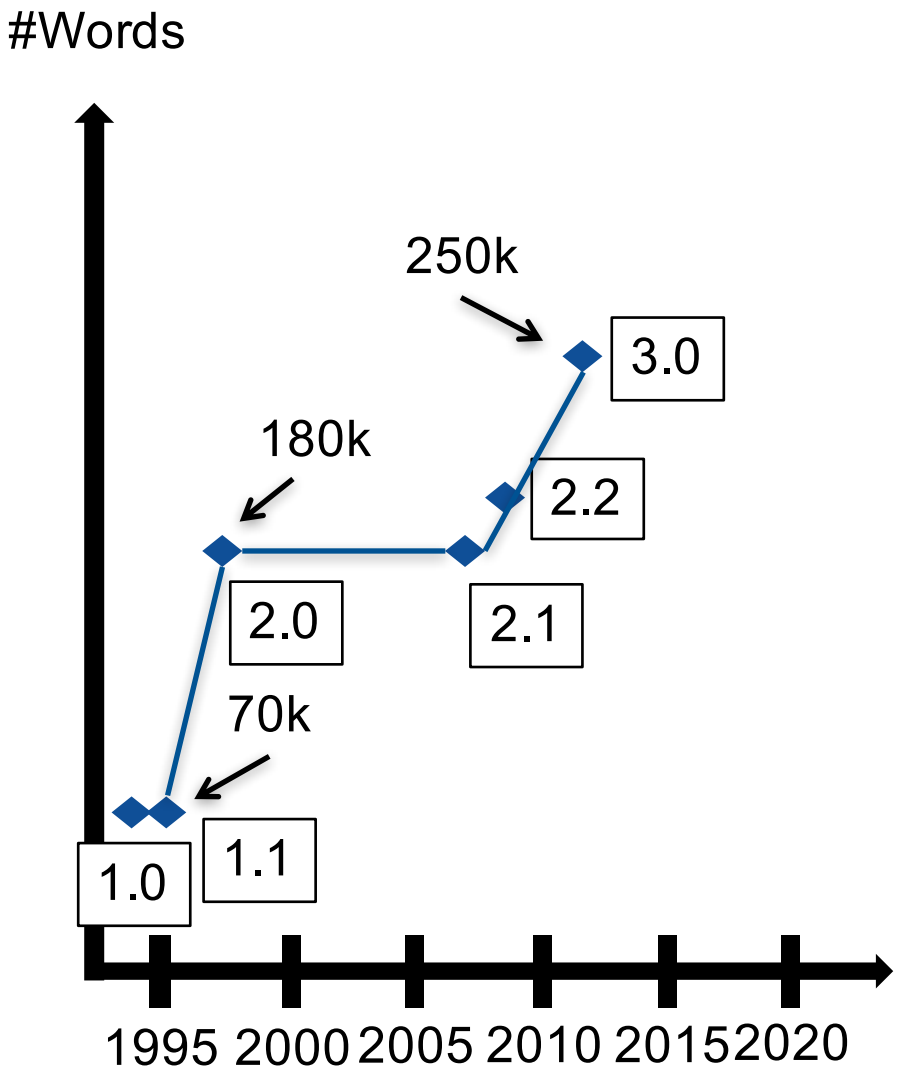
New language bindings

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)



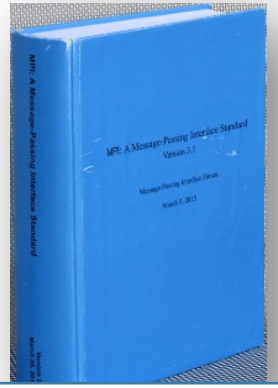
MPI 3.0 has 430 Functions

MPI_ABORT
MPI_ACCUMULATE
MPI_ADD_ERROR_CLASS
MPI_ADD_ERROR_CODE
MPI_ADD_ERROR_STRING
MPI_ADDRESS
MPI_ALLGATHER
MPI_ALLGATHERV
MPI_ALLOC_MEM
MPI_ALLOC_MEM_CPTR
MPI_ALLREDUCE
MPI_ALLTOALL
MPI_ALLTOALLV
MPI_ALLTOALLW
MPI_ATTR_DELETE
MPI_ATTR_GET
MPI_ATTR_PUT
MPI_BARRIER
MPI_BCAST
MPI_BSEND
MPI_BSEND_INIT
MPI_BUFFER_ATTACH
MPI_BUFFER_DETACH
MPI_CANCEL
MPI_CART_COORDS
MPI_CART_CREATE
MPI_CART_GET
MPI_CART_MAP
MPI_CART_RANK
MPI_CART_SHIFT
MPI_CART_SUB
MPI_CARTDIM_GET
MPI_CLOSE_PORT
MPI_COMM_ACCEPT
MPI_COMM_C2F
MPI_COMM_CALL_ERRHANDLER
MPI_COMM_COMPARE
MPI_COMM_CONNECT
MPI_COMM_CREATE
MPI_COMM_CREATE_ERRHANDLER
MPI_COMM_CREATE_GROUP
MPI_COMM_CREATE_KEYVAL
MPI_COMM_DELETE_ATTR
MPI_COMM_DISCONNECT
MPI_COMM_DUP
MPI_COMM_DUP_FN
MPI_COMM_DUP_WITH_INFO
MPI_COMM_F2C
MPI_COMM_FREE
MPI_COMM_FREE_KEYVAL
MPI_COMM_GET_ATTR
MPI_COMM_GET_ERRHANDLER
MPI_COMM_GET_INFO
MPI_COMM_GET_NAME
MPI_COMM_GET_PARENT
MPI_COMM_GROUP
MPI_COMM_IDUP
MPI_COMM_JOIN
MPI_COMM_KEYVAL_CREATE
MPI_COMM_NULL_COPY_FN
MPI_COMM_NULL_DELETE_FN
MPI_COMM_RANK
MPI_COMM_REMOTE_GROUP
MPI_COMM_REMOTE_SIZE
MPI_COMM_SET_ATTR
MPI_COMM_SET_ERRHANDLER
MPI_COMM_SET_INFO
MPI_COMM_SET_NAME
MPI_COMM_SIZE
MPI_COMM_SPAWN
MPI_COMM_SPAWN_MULTIPLE
MPI_COMM_SPLIT
MPI_COMM_SPLIT_TYPE
MPI_COMM_TEST_INTER
MPI_COMM_WORLD
MPI_COMPARE_AND_SWAP
MPI_CONVERSION_FN_NULL
MPI_DIMS_CREATE
MPI_DIST_GRAPH_CREATE
MPI_DIST_GRAPH_CREATE_ADJACENT
MPI_DIST_GRAPH_NEIGHBOR_COUNT
MPI_DIST_GRAPH_NEIGHBORS
MPI_DIST_GRAPH_NEIGHBORS_COUNT
MPI_DUP_FN
MPI_ERRHANDLER_C2F
MPI_ERRHANDLER_CREATE
MPI_ERRHANDLER_F2C
MPI_ERRHANDLER_FREE
MPI_ERRHANDLER_GET
MPI_ERRHANDLER_SET
MPI_ERROR_CLASS
MPI_ERROR_STRING
MPI_EXSCAN
MPI_F_SYNC_REG
MPI_FETCH_AND_OP
MPI_FILE_C2F
MPI_FILE_CALL_ERRHANDLER
MPI_FILE_CLOSE
MPI_FILE_CREATE_ERRHANDLER
MPI_FILE_DELETE
MPI_FILE_F2C
MPI_FILE_GET_AMODE
MPI_FILE_GET_ATOMICITY
MPI_FILE_GET_BYTE_OFFSET
MPI_FILE_GET_ERRHANDLER
MPI_FILE_GET_GROUP
MPI_FILE_GET_INFO
MPI_FILE_GET_POSITION
MPI_FILE_GET_POSITION_SHARED
MPI_FILE_GET_SIZE
MPI_FILE_GET_TYPE_EXTENT
MPI_FILE_GET_VIEW
MPI_FILE_IREAD
MPI_FILE_IREAD_AT
MPI_FILE_IREAD_SHARED
MPI_FILE_IWRITE
MPI_FILE_IWRITE_AT
MPI_FILE_IWRITE_SHARED
MPI_FILE_OPEN
MPI_FILE_PREALLOCATE
MPI_FILE_READ
MPI_FILE_READ_ALL
MPI_FILE_READ_ALL_BEGIN
MPI_FILE_READ_ALL_END
MPI_FILE_READ_AT
MPI_FILE_READ_AT_ALL
MPI_FILE_READ_AT_ALL_BEGIN
MPI_FILE_READ_AT_ALL_END
MPI_FILE_READ_ORDERED
MPI_FILE_READ_ORDERED_BEGIN
MPI_FILE_READ_ORDERED_END
MPI_FILE_READ_SHARED
MPI_FILE_SEEK
MPI_FILE_SEEK_SHARED
MPI_FILE_SET_ATOMICITY
MPI_FILE_SET_ERRHANDLER
MPI_FILE_SET_INFO
MPI_FILE_SET_SIZE
MPI_FILE_SET_VIEW
MPI_FILE_SYNC
MPI_FILE_WRITE
MPI_FILE_WRITE_ALL
MPI_FILE_WRITE_ALL_BEGIN
MPI_FILE_WRITE_ALL_END
MPI_FILE_WRITE_AT
MPI_FILE_WRITE_AT_ALL
MPI_FILE_WRITE_AT_ALL_BEGIN
MPI_FILE_WRITE_AT_ALL_END
MPI_FILE_WRITE_ORDERED
MPI_FILE_WRITE_ORDERED_BEGIN
MPI_FILE_WRITE_ORDERED_END
MPI_FILE_WRITE_SHARED
MPI_FINALIZE
MPI_FINALIZED
MPI_FREE_MEM
MPI_GATHER
MPI_GATHERV
MPI_GET
MPI_GET_ACCUMULATE
MPI_GET_ADDRESS
MPI_GET_COUNT
MPI_GET_ELEMENTS
MPI_GET_ELEMENTS_X
MPI_GET_LIBRARY_VERSION
MPI_GET_PROCESSOR_NAME
MPI_GET_VERSION
MPI_GRAPH_CREATE
MPI_GRAPH_GET
MPI_GRAPH_MAP
MPI_GRAPH_NEIGHBORS
MPI_GRAPH_NEIGHBORS_COUNT
MPI_GRAPHDIMS_GET
MPI_GREQUEST_COMPLETE
MPI_GREQUEST_START
MPI_GROUP_C2F
MPI_GROUP_COMPARE
MPI_GROUP_DIFFERENCE
MPI_GROUP_EXCL
MPI_GROUP_F2C
MPI_GROUP_FREE
MPI_GROUP_INCL
MPI_GROUP_INTERSECTION
MPI_GROUP_RANGE_EXCL
MPI_GROUP_RANGE_INCL
MPI_GROUP_RANK
MPI_GROUP_SIZE
MPI_GROUP_TRANSLATE_RANKS
MPI_GROUP_UNION
MPI_IALLGATHER
MPI_IALLGATHERV
MPI_IALLREDUCE
MPI_IALLTOALL
MPI_IALLTOALLV
MPI_IALLTOALLW
MPI_IBARRIER
MPI_IBCAST
MPI_IBSEND
MPI_IEXSCAN
MPI_IGATHER
MPI_IGATHERV
MPI_IMPROBE
MPI_IMRECV
MPI_INEIGHBOR_ALLGATHER
MPI_INEIGHBOR_ALLGATHERV
MPI_INEIGHBOR_ALLTOALL
MPI_INEIGHBOR_ALLTOALLV
MPI_INEIGHBOR_ALLTOALLW
MPI_INFO_C2F
MPI_INFO_CREATE
MPI_INFO_DELETE
MPI_INFO_DUP
MPI_INFO_F2C
MPI_INFO_FREE
MPI_INFO_GET
MPI_INFO_GET_NKEYS
MPI_INFO_GET_NTHKEY
MPI_INFO_GET_VALUELEN
MPI_INFO_SET
MPI_INIT
MPI_INIT_THREAD
MPI_INITIALIZED
MPI_INTERCOMM_CREATE
MPI_INTERCOMM_MERGE
MPI_IPROBE
MPI_IRECV
MPI_IREDUCE
MPI_IREDUCE_SCATTER
MPI_IREDUCE_SCATTER_BLOCK
MPI_IRSEND
MPI_IS_THREAD_MAIN
MPI_ISCAN
MPI_ISCATTER
MPI_ISCATTERV
MPI_ISEND
MPI_ISSEND
MPI_KEYVAL_CREATE
MPI_KEYVAL_FREE
MPI_LOCK_ALL
MPI_LOOKUP_NAME
MPI_MESSAGE_C2F
MPI_MESSAGE_F2C
MPI_MPROBE
MPI_MRECV
MPI_NEIGHBOR_ALLGATHER
MPI_NEIGHBOR_ALLGATHERV
MPI_NEIGHBOR_ALLTOALL
MPI_NEIGHBOR_ALLTOALLV
MPI_NEIGHBOR_ALLTOALLW
MPI_NULL_COPY_FN
MPI_NULL_DELETE_FN
MPI_OP_C2F
MPI_OP_COMMUTATIVE
MPI_OP_CREATE
MPI_OP_F2C
MPI_OP_FREE
MPI_OPEN_PORT
MPI_PACK
MPI_PACK_EXTERNAL
MPI_PACK_EXTERNAL_SIZE
MPI_PACK_SIZE
MPI_PCONTROL
MPI_PROBE
MPI_PUBLISH_NAME
MPI_PUT
MPI_QUERY_THREAD
MPI_RACCUMULATE
MPI_RECV
MPI_RECV_INIT
MPI_REDUCE
MPI_REDUCE_LOCAL
MPI_REDUCE_SCATTER
MPI_REDUCE_SCATTER_BLOCK
MPI_REGISTER_DATAREP
MPI_REQUEST_C2F
MPI_REQUEST_F2C
MPI_REQUEST_FREE
MPI_REQUEST_GET_STATUS
MPI_RGET
MPI_RGET_ACCUMULATE
MPI_RPUT
MPI_RSEND
MPI_RSEND_INIT
MPI_SCAN
MPI_SCATTER
MPI_SCATTERV
MPI_SEND
MPI_SEND_INIT
MPI_SENDRECV
MPI_SENDRECV_REPLACE
MPI_SIZEOF
MPI_SSEND
MPI_SSEND_INIT
MPI_START
MPI_STARTALL
MPI_STATUS_C2F
MPI_STATUS_C2F08
MPI_STATUS_F082C
MPI_STATUS_F082F
MPI_STATUS_F2C
MPI_STATUS_F2F08
MPI_STATUS_SET_CANCELLED
MPI_STATUS_SET_ELEMENTS
MPI_STATUS_SET_ELEMENTS_X
MPI_T_CATEGORY_CHANGED
MPI_T_CATEGORY_GET_CATEGORIES
MPI_T_CATEGORY_GET_CVARS
MPI_T_CATEGORY_GET_INFO
MPI_T_CATEGORY_GET_NUM
MPI_T_CATEGORY_GET_PVARS
MPI_T_CVAR_GET_INFO
MPI_T_CVAR_GET_NUM
MPI_T_CVAR_HANDLE_ALLOC
MPI_T_CVAR_HANDLE_FREE
MPI_T_CVAR_READ
MPI_T_CVAR_WRITE
MPI_T_ENUM_GET_INFO
MPI_T_ENUM_GET_ITEM
MPI_T_FINALIZE
MPI_T_INIT_THREAD
MPI_T_PVAR_GET_INFO
MPI_T_PVAR_GET_NUM
MPI_T_PVAR_HANDLE_ALLOC
MPI_T_PVAR_HANDLE_FREE
MPI_T_PVAR_READ
MPI_T_PVAR_READRESET
MPI_T_PVAR_RESET
MPI_T_PVAR_SESSION_CREATE
MPI_T_PVAR_SESSION_FREE
MPI_T_PVAR_START
MPI_T_PVAR_STOP
MPI_T_PVAR_WRITE
MPI_TEST
MPI_TEST_CANCELLED
MPI_TESTALL
MPI_TESTANY
MPI_TESTSOME
MPI_TOPO_TEST
MPI_TYPE_C2F
MPI_TYPE_COMMIT
MPI_TYPE_CONTIGUOUS
MPI_TYPE_CREATE_DARRAY
MPI_TYPE_CREATE_F90_COMPLEX
MPI_TYPE_CREATE_F90_INTEGER
MPI_TYPE_CREATE_F90_REAL
MPI_TYPE_CREATE_HINDEXED
MPI_TYPE_CREATE_HINDEXED_BLOCK
MPI_TYPE_CREATE_HVECTOR
MPI_TYPE_CREATE_INDEXED_BLOCK
MPI_TYPE_CREATE_KEYVAL
MPI_TYPE_CREATE_RESIZED
MPI_TYPE_CREATE_STRUCT
MPI_TYPE_CREATE_SUBARRAY
MPI_TYPE_DELETE_ATTR
MPI_TYPE_DUP
MPI_TYPE_DUP_FN
MPI_TYPE_EXTENT
MPI_TYPE_F2C
MPI_TYPE_FREE
MPI_TYPE_FREE_KEYVAL
MPI_TYPE_GET_ATTR
MPI_TYPE_GET_CONTENTS
MPI_TYPE_GET_ENVELOPE
MPI_TYPE_GET_EXTENT
MPI_TYPE_GET_EXTENT_X
MPI_TYPE_GET_NAME
MPI_TYPE_GET_TRUE_EXTENT
MPI_TYPE_GET_TRUE_EXTENT_X
MPI_TYPE_HINDEXED
MPI_TYPE_HVECTOR
MPI_TYPE_INDEXED
MPI_TYPE_LB
MPI_TYPE_MATCH_SIZE
MPI_TYPE_NULL_COPY_FN
MPI_TYPE_NULL_DELETE_FN
MPI_TYPE_SET_ATTR
MPI_TYPE_SET_NAME
MPI_TYPE_SIZE
MPI_TYPE_SIZE_X
MPI_TYPE_STRUCT
MPI_TYPE_UB
MPI_TYPE_VECTOR
MPI_UNPACK
MPI_UNPACK_EXTERNAL
MPI_UNPUBLISH_NAME
MPI_WAIT
MPI_WAITALL
MPI_WAITANY
MPI_WAITSOME
MPI_WIN_ALLOC
MPI_WIN_ALLOCATE
MPI_WIN_ALLOCATE_CPTR
MPI_WIN_ALLOCATE_SHARED
MPI_WIN_ALLOCATE_SHARED_CPTR
MPI_WIN_ATTACH
MPI_WIN_C2F
MPI_WIN_CALL_ERRHANDLER
MPI_WIN_COMPLETE
MPI_WIN_CREATE
MPI_WIN_CREATE_DYNAMIC
MPI_WIN_CREATE_ERRHANDLER
MPI_WIN_CREATE_KEYVAL
MPI_WIN_DELETE_ATTR
MPI_WIN_DETACH
MPI_WIN_DUP_FN
MPI_WIN_F2C
MPI_WIN_FENCE
MPI_WIN_FLUSH
MPI_WIN_FLUSH_ALL
MPI_WIN_FLUSH_LOCAL
MPI_WIN_FLUSH_LOCAL_ALL
MPI_WIN_FREE
MPI_WIN_FREE_KEYVAL
MPI_WIN_GET_ATTR
MPI_WIN_GET_ERRHANDLER
MPI_WIN_GET_GROUP
MPI_WIN_GET_INFO
MPI_WIN_GET_NAME
MPI_WIN_LOCK
MPI_WIN_LOCK_ALL
MPI_WIN_NULL_COPY_FN
MPI_WIN_NULL_DELETE_FN
MPI_WIN_POST
MPI_WIN_SET_ATTR
MPI_WIN_SET_ERRHANDLER
MPI_WIN_SET_INFO
MPI_WIN_SET_NAME
MPI_WIN_SHARED_ALLOCATE
MPI_WIN_SHARED_QUERY
MPI_WIN_SHARED_QUERY_CPTR
MPI_WIN_START
MPI_WIN_SYNC
MPI_WIN_TEST
MPI_WIN_UNLOCK
MPI_WIN_UNLOCK_ALL
MPI_WIN_WAIT
MPI_WTICK
MPI_WTIME

MPI 3.0 has 430 Functions

MPI_ABORT
MPI_ACCUMULATE
MPI_ADD_ERROR_CLASS
MPI_ADD_ERROR_CODE
MPI_ADD_ERROR_STRING
MPI_ADDRESS
MPI_ALLGATHER
MPI_ALLGATHERV
MPI_ALLOC_MEM
MPI_ALLOC_MEM_CPTR
MPI_ALLREDUCE
MPI_ALLTOALL
MPI_ALLTOALLV
MPI_ALLTOALLW
MPI_ATTR_DELETE
MPI_ATTR_GET
MPI_ATTR_PUT
MPI_BARRIER
MPI_BCAST
MPI_BSEND
MPI_BSEND_INIT
MPI_BUFFER_ATTACH
MPI_BUFFER_DETACH
MPI_CANCEL
MPI_CART_COORDS
MPI_CART_CREATE
MPI_CART_GET
MPI_CART_MAP
MPI_CART_RANK
MPI_CART_SHIFT
MPI_CART_SUB
MPI_CARTDIM_GET
MPI_CLOSE_PORT
MPI_COMM_ACCEPT
MPI_COMM_C2F
MPI_COMM_CALL_ERRHANDLER
MPI_COMM_COMPARE
MPI_COMM_CONNECT
MPI_COMM_CREATE
MPI_COMM_CREATE_ERRHANDLER
MPI_COMM_CREATE_GROUP
MPI_COMM_CREATE_KEYVAL
MPI_COMM_DELETE_ATTR
MPI_COMM_DISCONNECT
MPI_COMM_DUP
MPI_COMM_DUP_FN
MPI_COMM_DUP_WITH_INFO
MPI_COMM_F2C
MPI_COMM_FREE
MPI_COMM_FREE_KEYVAL
MPI_COMM_GET_ATTR
MPI_COMM_GET_ERRHANDLER
MPI_COMM_GET_INFO
MPI_COMM_GET_NAME
MPI_COMM_GET_PARENT
MPI_COMM_GROUP
MPI_COMM_IDUP
MPI_COMM_JOIN
MPI_COMM_KEYVAL_CREATE
MPI_COMM_NULL_COPY_FN
MPI_COMM_NULL_DELETE_FN
MPI_COMM_RANK
MPI_COMM_REMOTE_GROUP
MPI_COMM_REMOTE_SIZE
MPI_COMM_SET_ATTR
MPI_COMM_SET_ERRHANDLER
MPI_COMM_SET_INFO
MPI_COMM_SET_NAME
MPI_COMM_SIZE
MPI_COMM_SPAWN
MPI_COMM_SPAWN_MULTIPLE
MPI_COMM_SPLIT
MPI_COMM_SPLIT_TYPE
MPI_COMM_TEST_INTER
MPI_COMM_WORLD
MPI_COMPARE_AND_SWAP
MPI_CONVERSION_FN_NULL
MPI_DIMS_CREATE
MPI_DIST_GRAPH_CREATE
MPI_DIST_GRAPH_CREATE_ADJACENT
MPI_DIST_GRAPH_NEIGHBOR_COUNT
MPI_DIST_GRAPH_NEIGHBORS
MPI_DIST_GRAPH_NEIGHBORS_COUNT
MPI_DUP_FN
MPI_ERRHANDLER_C2F
MPI_ERRHANDLER_CREATE
MPI_ERRHANDLER_F2C
MPI_ERRHANDLER_FREE
MPI_ERRHANDLER_GET
MPI_ERRHANDLER_SET
MPI_ERROR_CLASS
MPI_ERROR_STRING
MPI_EXSCAN
MPI_F_SYNC_REG
MPI_FETCH_AND_OP
MPI_FILE_C2F
MPI_FILE_CALL_ERRHANDLER
MPI_FILE_CLOSE
MPI_FILE_CREATE_ERRHANDLER
MPI_FILE_DELETE
MPI_FILE_F2C
MPI_FILE_GET_AMODE
MPI_FILE_GET_ATOMICITY
MPI_FILE_GET_BYTE_OFFSET
MPI_FILE_GET_ERRHANDLER
MPI_FILE_GET_GROUP
MPI_FILE_GET_INFO
MPI_FILE_GET_POSITION
MPI_FILE_GET_POSITION_SHARED
MPI_FILE_GET_SIZE
MPI_FILE_GET_TYPE_EXTENT
MPI_FILE_GET_VIEW
MPI_FILE_IREAD
MPI_FILE_IREAD_AT
MPI_FILE_IREAD_SHARED
MPI_FILE_IWRITE
MPI_FILE_IWRITE_AT
MPI_FILE_IWRITE_SHARED
MPI_FILE_OPEN
MPI_FILE_PREALLOCATE
MPI_FILE_READ
MPI_FILE_READ_ALL
MPI_FILE_READ_ALL_BEGIN
MPI_FILE_READ_ALL_END
MPI_FILE_READ_AT
MPI_FILE_READ_AT_ALL
MPI_FILE_READ_AT_ALL_BEGIN
MPI_FILE_READ_AT_ALL_END
MPI_FILE_READ_ORDERED
MPI_FILE_READ_ORDERED_BEGIN
MPI_FILE_READ_ORDERED_END
MPI_FILE_READ_SHARED
MPI_FILE_SEEK
MPI_FILE_SEEK_SHARED
MPI_FILE_SET_ATOMICITY
MPI_FILE_SET_ERRHANDLER
MPI_FILE_SET_INFO
MPI_FILE_SET_SIZE
MPI_FILE_SET_VIEW
MPI_FILE_SYNC
MPI_FILE_WRITE
MPI_FILE_WRITE_ALL
MPI_FILE_WRITE_ALL_BEGIN
MPI_FILE_WRITE_ALL_END
MPI_FILE_WRITE_AT
MPI_FILE_WRITE_AT_ALL
MPI_FILE_WRITE_AT_ALL_BEGIN
MPI_FILE_WRITE_AT_ALL_END
MPI_FILE_WRITE_ORDERED
MPI_FILE_WRITE_ORDERED_BEGIN
MPI_FILE_WRITE_ORDERED_END
MPI_FILE_WRITE_SHARED
MPI_FINALIZE
MPI_FINALIZED
MPI_FREE_MEM
MPI_GATHER
MPI_GATHERV
MPI_GET
MPI_GET_ACCUMULATE
MPI_GET_ADDRESS
MPI_GET_COUNT
MPI_GET_ELEMENTS
MPI_GET_ELEMENTS_X
MPI_GET_LIBRARY_VERSION
MPI_GET_PROCESSOR_NAME
MPI_GET_VERSION
MPI_GRAPH_CREATE
MPI_GRAPH_GET
MPI_GRAPH_MAP
MPI_GRAPH_NEIGHBORS
MPI_GRAPH_NEIGHBORS_COUNT
MPI_GRAPHDIMS_GET
MPI_GREQUEST_COMPLETE
MPI_GREQUEST_START
MPI_GROUP_C2F
MPI_GROUP_COMPARE
MPI_GROUP_DIFFERENCE
MPI_GROUP_EXCL
MPI_GROUP_F2C
MPI_GROUP_FREE
MPI_GROUP_INCL
MPI_GROUP_INTERSECTION
MPI_GROUP_RANGE_EXCL
MPI_GROUP_RANGE_INCL
MPI_GROUP_RANK
MPI_GROUP_SIZE
MPI_GROUP_TRANSLATE_RANKS
MPI_GROUP_UNION
MPI_IALLGATHER
MPI_IALLGATHERV
MPI_IALLREDUCE
MPI_IALLTOALL
MPI_IALLTOALLV
MPI_IALLTOALLW
MPI_IBARRIER
MPI_IBCAST
MPI_IBSEND
MPI_IEXSCAN
MPI_IGATHER
MPI_IGATHERV
MPI_IMPROBE
MPI_IMRECV
MPI_INEIGHBOR_ALLGATHER
MPI_INEIGHBOR_ALLGATHERV
MPI_INEIGHBOR_ALLTOALL
MPI_INEIGHBOR_ALLTOALLV
MPI_INEIGHBOR_ALLTOALLW
MPI_INFO_C2F
MPI_INFO_CREATE
MPI_INFO_DELETE
MPI_INFO_DUP
MPI_INFO_F2C
MPI_INFO_FREE
MPI_INFO_GET
MPI_INFO_GET_NKEYS
MPI_INFO_GET_NTHKEY
MPI_INFO_GET_VALUELEN
MPI_INFO_SET
MPI_INIT
MPI_INIT_THREAD
MPI_INITIALIZED
MPI_INTERCOMM_CREATE
MPI_INTERCOMM_MERGE
MPI_IPROBE
MPI_IRECV
MPI_IREDUCE
MPI_IREDUCE_SCATTER
MPI_IREDUCE_SCATTER_BLOCK
MPI_IRSEND
MPI_IS_THREAD_MAIN
MPI_ISCAN
MPI_ISCATTER
MPI_ISCATTERV
MPI_ISEND
MPI_ISSEND
MPI_KEYVAL_CREATE
MPI_KEYVAL_FREE
MPI_LOCK_ALL
MPI_LOOKUP_NAME
MPI_MESSAGE_C2F
MPI_MESSAGE_F2C
MPI_MPROBE
MPI_MRECV
MPI_NEIGHBOR_ALLGATHER
MPI_NEIGHBOR_ALLGATHERV
MPI_NEIGHBOR_ALLTOALL
MPI_NEIGHBOR_ALLTOALLV
MPI_NEIGHBOR_ALLTOALLW
MPI_NULL_COPY_FN
MPI_NULL_DELETE_FN
MPI_OP_C2F
MPI_OP_COMMUTATIVE
MPI_OP_CREATE
MPI_OP_F2C
MPI_OP_FREE
MPI_OPEN_PORT
MPI_PACK
MPI_PACK_EXTERNAL
MPI_PACK_EXTERNAL_SIZE
MPI_PACK_SIZE
MPI_PCONTROL
MPI_PROBE
MPI_PUBLISH_NAME
MPI_PUT
MPI_QUERY_THREAD
MPI_RACCUMULATE
MPI_RECV
MPI_RECV_INIT
MPI_REDUCE
MPI_REDUCE_LOCAL
MPI_REDUCE_SCATTER
MPI_REDUCE_SCATTER_BLOCK
MPI_REGISTER_DATAREP
MPI_REQUEST_C2F
MPI_REQUEST_F2C
MPI_REQUEST_FREE
MPI_REQUEST_GET_STATUS
MPI_RGET
MPI_RGET_ACCUMULATE
MPI_RPUT
MPI_RSEND
MPI_RSEND_INIT
MPI_SCAN
MPI_SCATTER
MPI_SCATTERV
MPI_SEND
MPI_SEND_INIT
MPI_SENDRECV
MPI_SENDRECV_REPLACE
MPI_SIZEOF
MPI_SSEND
MPI_SSEND_INIT
MPI_START
MPI_STARTALL
MPI_STATUS_C2F
MPI_STATUS_C2F08
MPI_STATUS_F082C
MPI_STATUS_F082F
MPI_STATUS_F2C
MPI_STATUS_F2F08
MPI_STATUS_SET_CANCELLED
MPI_STATUS_SET_ELEMENTS
MPI_STATUS_SET_ELEMENTS_X
MPI_T_CATEGORY_CHANGED
MPI_T_CATEGORY_GET_CATEGORIES
MPI_T_CATEGORY_GET_CVARS
MPI_T_CATEGORY_GET_INFO
MPI_T_CATEGORY_GET_NUM
MPI_T_CATEGORY_GET_PVARS
MPI_T_CVAR_GET_INFO
MPI_T_CVAR_GET_NUM
MPI_T_CVAR_HANDLE_ALLOC
MPI_T_CVAR_HANDLE_FREE
MPI_T_CVAR_READ
MPI_T_CVAR_WRITE
MPI_T_ENUM_GET_INFO
MPI_T_ENUM_GET_ITEM
MPI_T_FINALIZE
MPI_T_INIT_THREAD
MPI_T_PVAR_GET_INFO
MPI_T_PVAR_GET_NUM
MPI_T_PVAR_HANDLE_ALLOC
MPI_T_PVAR_HANDLE_FREE
MPI_T_PVAR_READ
MPI_T_PVAR_READRESET
MPI_T_PVAR_RESET
MPI_T_PVAR_SESSION_CREATE
MPI_T_PVAR_SESSION_FREE
MPI_T_PVAR_START
MPI_T_PVAR_STOP
MPI_T_PVAR_WRITE
MPI_TEST
MPI_TEST_CANCELLED
MPI_TESTALL
MPI_TESTANY
MPI_TESTSOME
MPI_TOPO_TEST
MPI_TYPE_C2F
MPI_TYPE_COMMIT
MPI_TYPE_CONTIGUOUS
MPI_TYPE_CREATE_DARRAY
MPI_TYPE_CREATE_F90_COMPLEX
MPI_TYPE_CREATE_F90_INTEGER
MPI_TYPE_CREATE_F90_REAL
MPI_TYPE_CREATE_HINDEXED
MPI_TYPE_CREATE_HINDEXED_BLOCK
MPI_TYPE_CREATE_HVECTOR
MPI_TYPE_CREATE_INDEXED_BLOCK
MPI_TYPE_CREATE_KEYVAL
MPI_TYPE_CREATE_RESIZED
MPI_TYPE_CREATE_STRUCT
MPI_TYPE_CREATE_SUBARRAY
MPI_TYPE_DELETE_ATTR
MPI_TYPE_DUP
MPI_TYPE_DUP_FN
MPI_TYPE_EXTENT
MPI_TYPE_F2C
MPI_TYPE_FREE
MPI_TYPE_FREE_KEYVAL
MPI_TYPE_GET_ATTR
MPI_TYPE_GET_CONTENTS
MPI_TYPE_GET_ENVELOPE
MPI_TYPE_GET_EXTENT
MPI_TYPE_GET_EXTENT_X
MPI_TYPE_GET_NAME
MPI_TYPE_GET_TRUE_EXTENT
MPI_TYPE_GET_TRUE_EXTENT_X
MPI_TYPE_HINDEXED
MPI_TYPE_HVECTOR
MPI_TYPE_INDEXED
MPI_TYPE_LB
MPI_TYPE_MATCH_SIZE
MPI_TYPE_NULL_COPY_FN
MPI_TYPE_NULL_DELETE_FN
MPI_TYPE_SET_ATTR
MPI_TYPE_SET_NAME
MPI_TYPE_SIZE
MPI_TYPE_SIZE_X
MPI_TYPE_STRUCT
MPI_TYPE_UB
MPI_TYPE_VECTOR
MPI_UNPACK
MPI_UNPACK_EXTERNAL
MPI_UNPUBLISH_NAME
MPI_WAIT
MPI_WAITALL
MPI_WAITANY
MPI_WAITSOME
MPI_WIN_ALLOC
MPI_WIN_ALLOCATE
MPI_WIN_ALLOCATE_CPTR
MPI_WIN_ALLOCATE_SHARED
MPI_WIN_ALLOCATE_SHARED_CPTR
MPI_WIN_ATTACH
MPI_WIN_C2F
MPI_WIN_CALL_ERRHANDLER
MPI_WIN_COMPLETE
MPI_WIN_CREATE
MPI_WIN_CREATE_DYNAMIC
MPI_WIN_CREATE_ERRHANDLER
MPI_WIN_CREATE_KEYVAL
MPI_WIN_DELETE_ATTR
MPI_WIN_DETACH
MPI_WIN_DUP_FN
MPI_WIN_F2C
MPI_WIN_FENCE
MPI_WIN_FLUSH
MPI_WIN_FLUSH_ALL
MPI_WIN_FLUSH_LOCAL
MPI_WIN_FLUSH_LOCAL_ALL
MPI_WIN_FREE
MPI_WIN_FREE_KEYVAL
MPI_WIN_GET_ATTR
MPI_WIN_GET_ERRHANDLER
MPI_WIN_GET_GROUP
MPI_WIN_GET_INFO
MPI_WIN_GET_NAME
MPI_WIN_LOCK
MPI_WIN_LOCK_ALL
MPI_WIN_NULL_COPY_FN
MPI_WIN_NULL_DELETE_FN
MPI_WIN_POST
MPI_WIN_SET_ATTR
MPI_WIN_SET_ERRHANDLER
MPI_WIN_SET_INFO
MPI_WIN_SET_NAME
MPI_WIN_SHARED_ALLOCATE
MPI_WIN_SHARED_QUERY
MPI_WIN_SHARED_QUERY_CPTR
MPI_WIN_START
MPI_WIN_SYNC
MPI_WIN_TEST
MPI_WIN_UNLOCK
MPI_WIN_UNLOCK_ALL
MPI_WIN_WAIT
MPI_WTICK
MPI_WTIME

MPI Continuous to Evolve



Available through HLRS
-> MPI Forum Website

MPI 3.0 ratified in September 2012
- Available at http://www.mpi-forum.org/
- Several major additions compared to MPI 2.2

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

- Need to split work into independent pieces
- Each piece must be it's own separate program

Carve up "shared" data structures

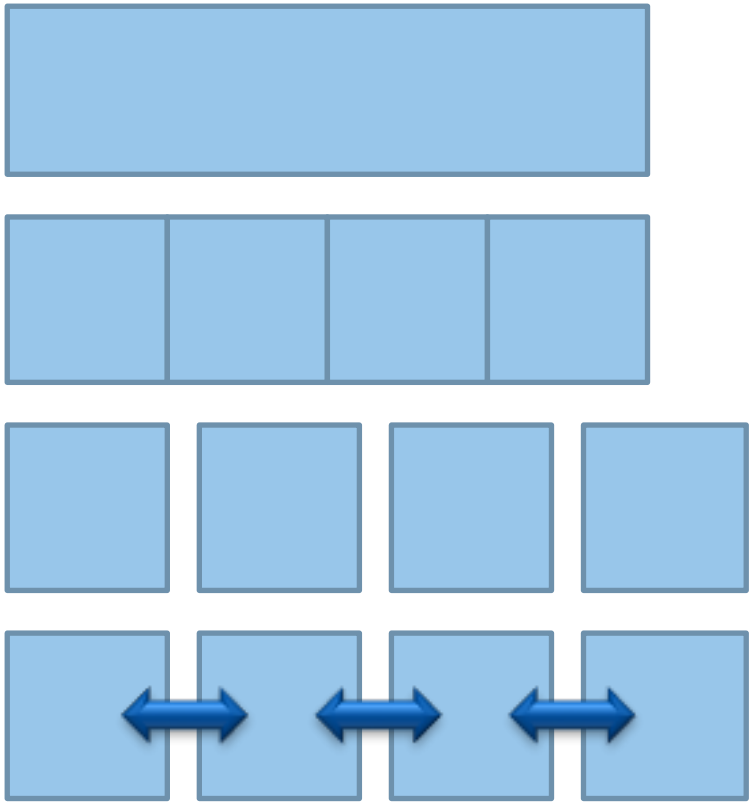
- Each MPI process must have its local data
- Avoid or duplicate data needed by one or more

Allocate data elements locally

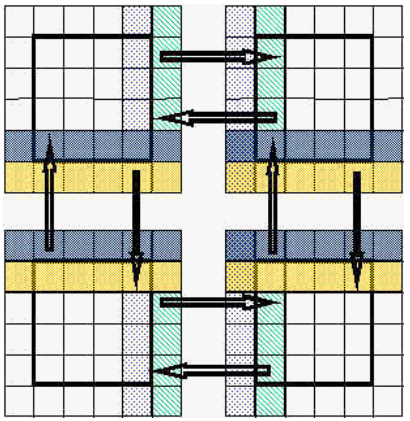
- Independent allocations with own addresses
- Avoid central initialization

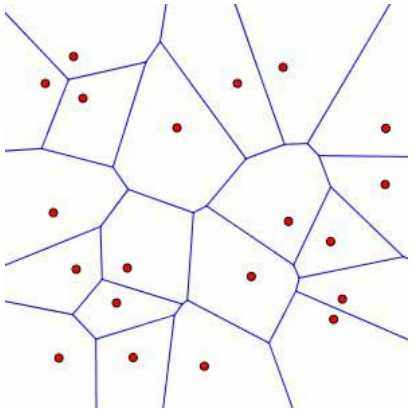
Add communication

- Exchange of data
- Synchronization



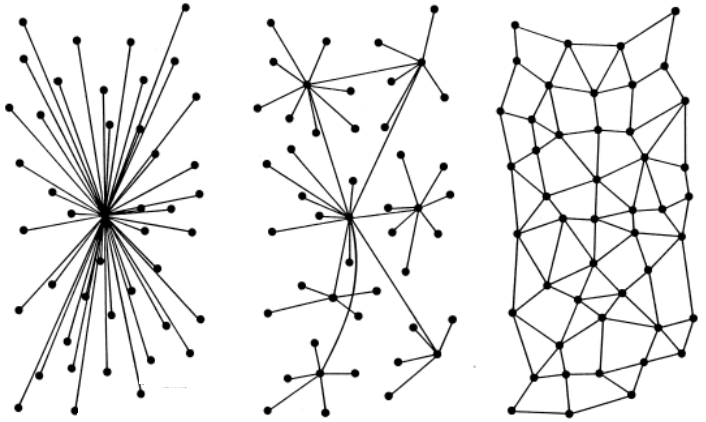
Create a “Logical” Topology





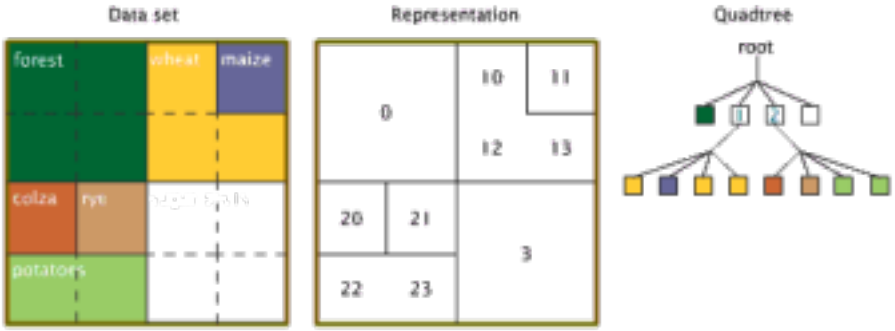
Arrange data items as needed
- Define base elements
- Create mapping of element to MPI process

Create “logical topology”
- Establish overlap, e.g., halos
- 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

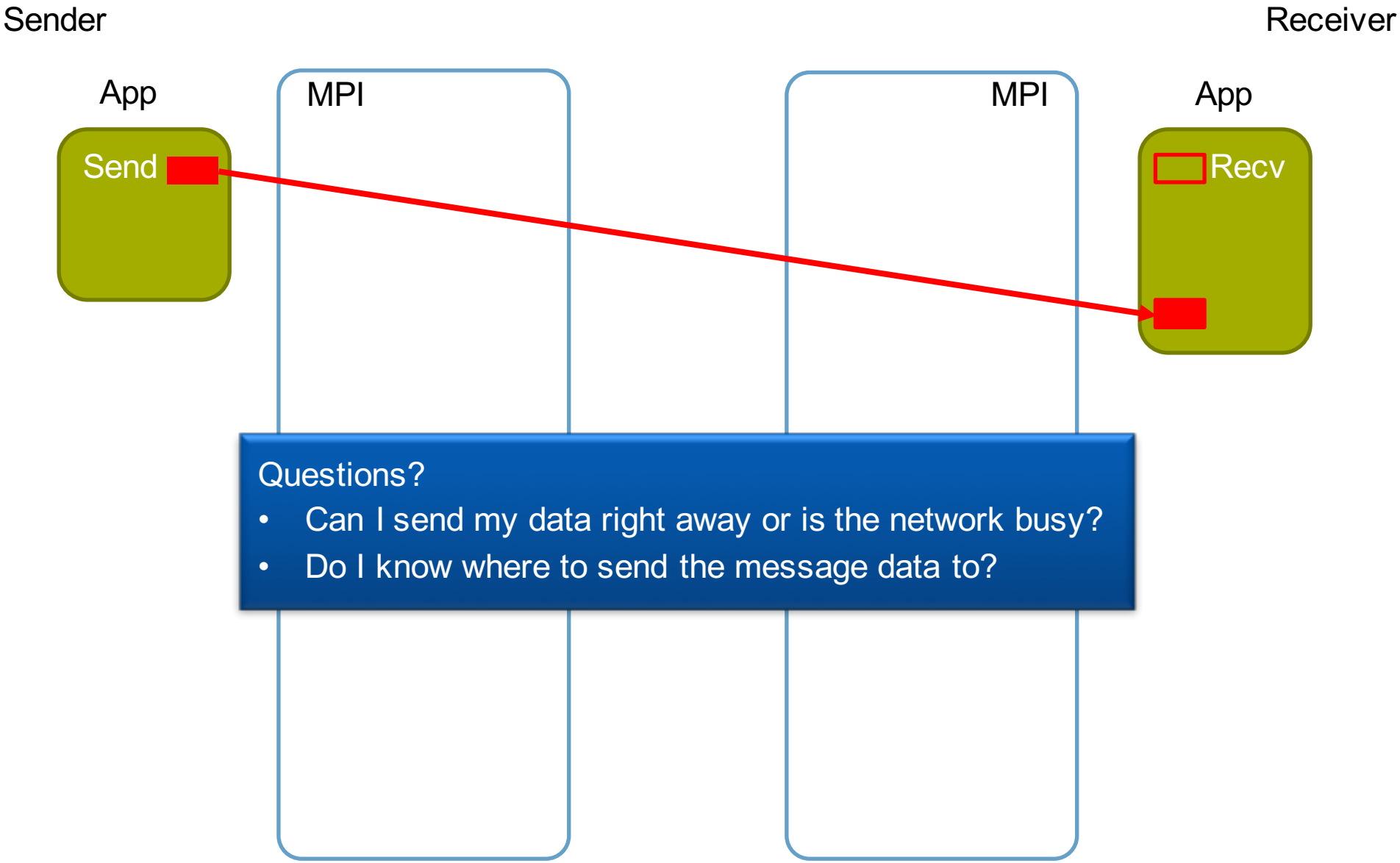
Transfer data elements between two “adjacent”

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

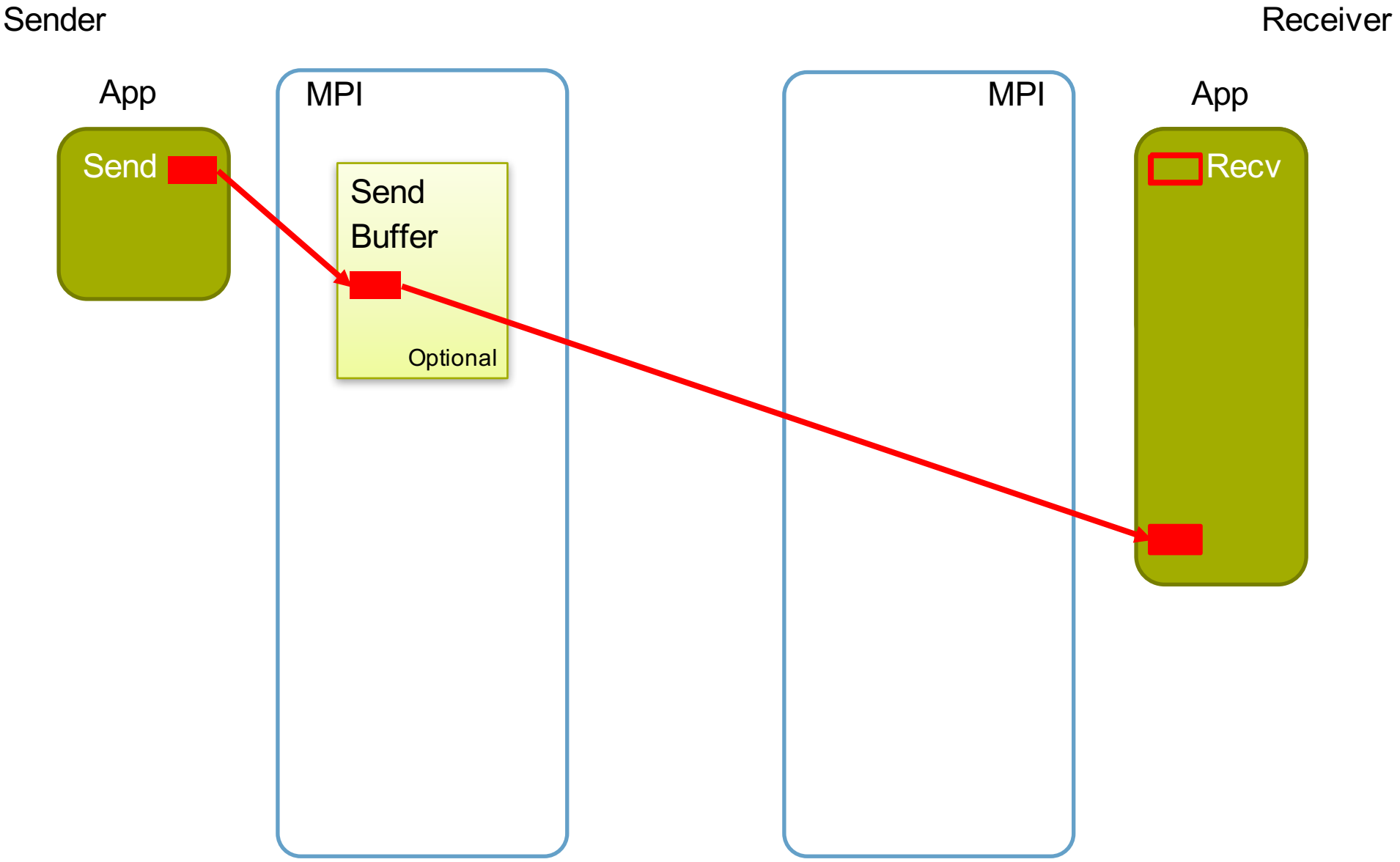
Problem: deadlock possible

- MPI_Send may need resources on receiver side

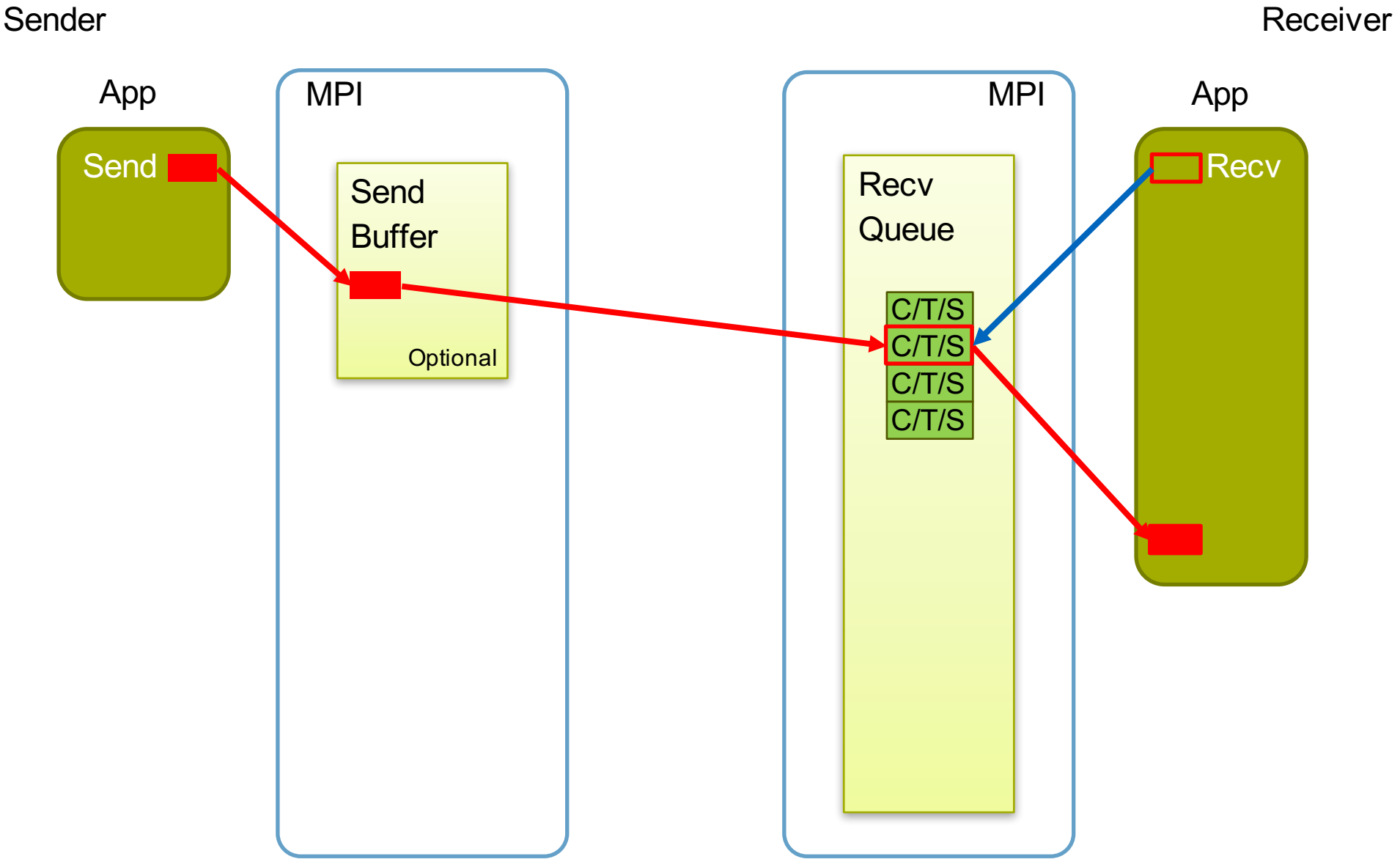
How Messages are Transferred



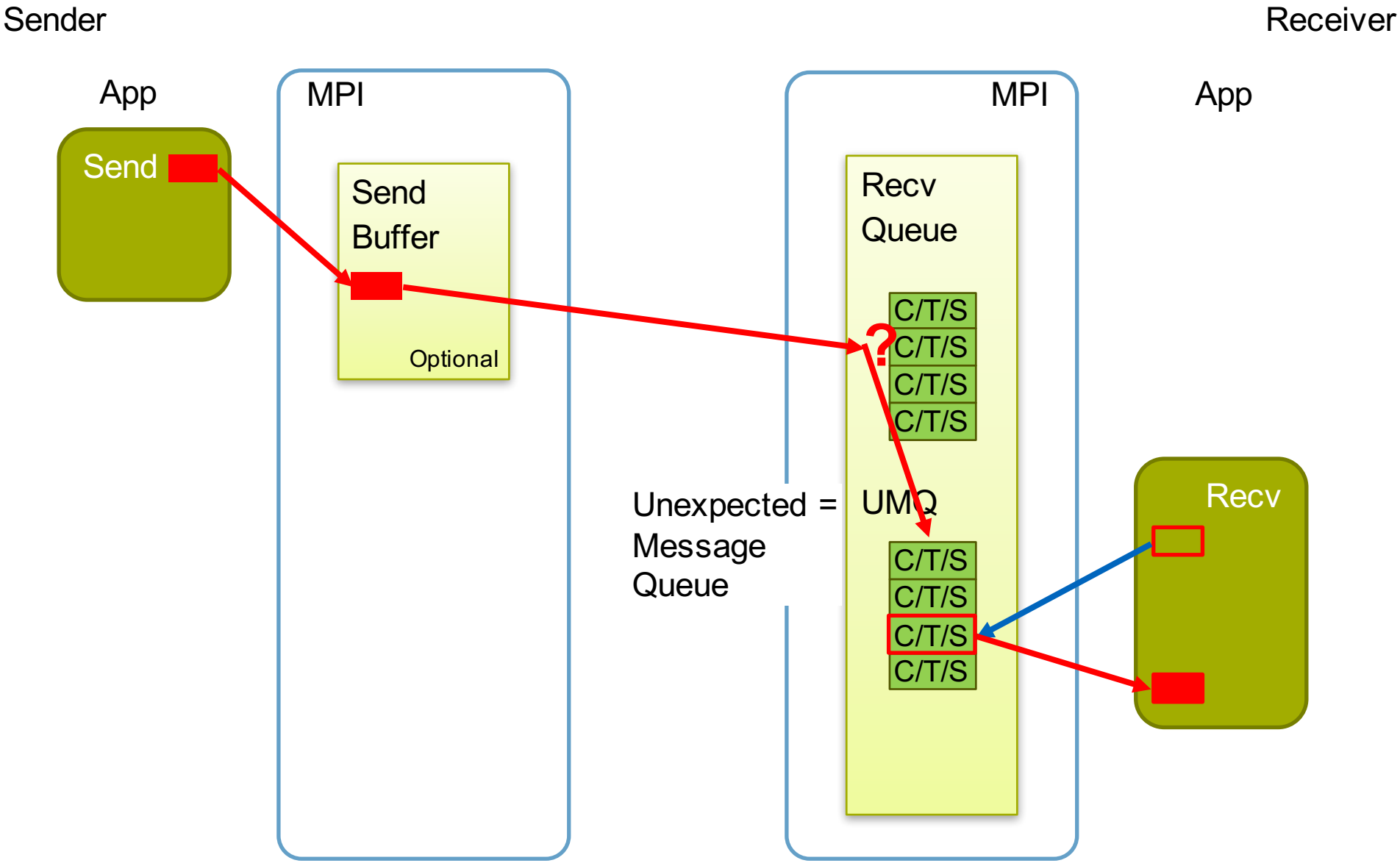
How Messages are Transferred



How Messages are Transferred



How Messages are Transferred



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

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

`MPI_Rsend`
- Ready Send
- User must ensure that receive has been posted
- Enables faster communication, but needs implicit synchronization

Checking the UMQ with MPI_Probe/Iprobe

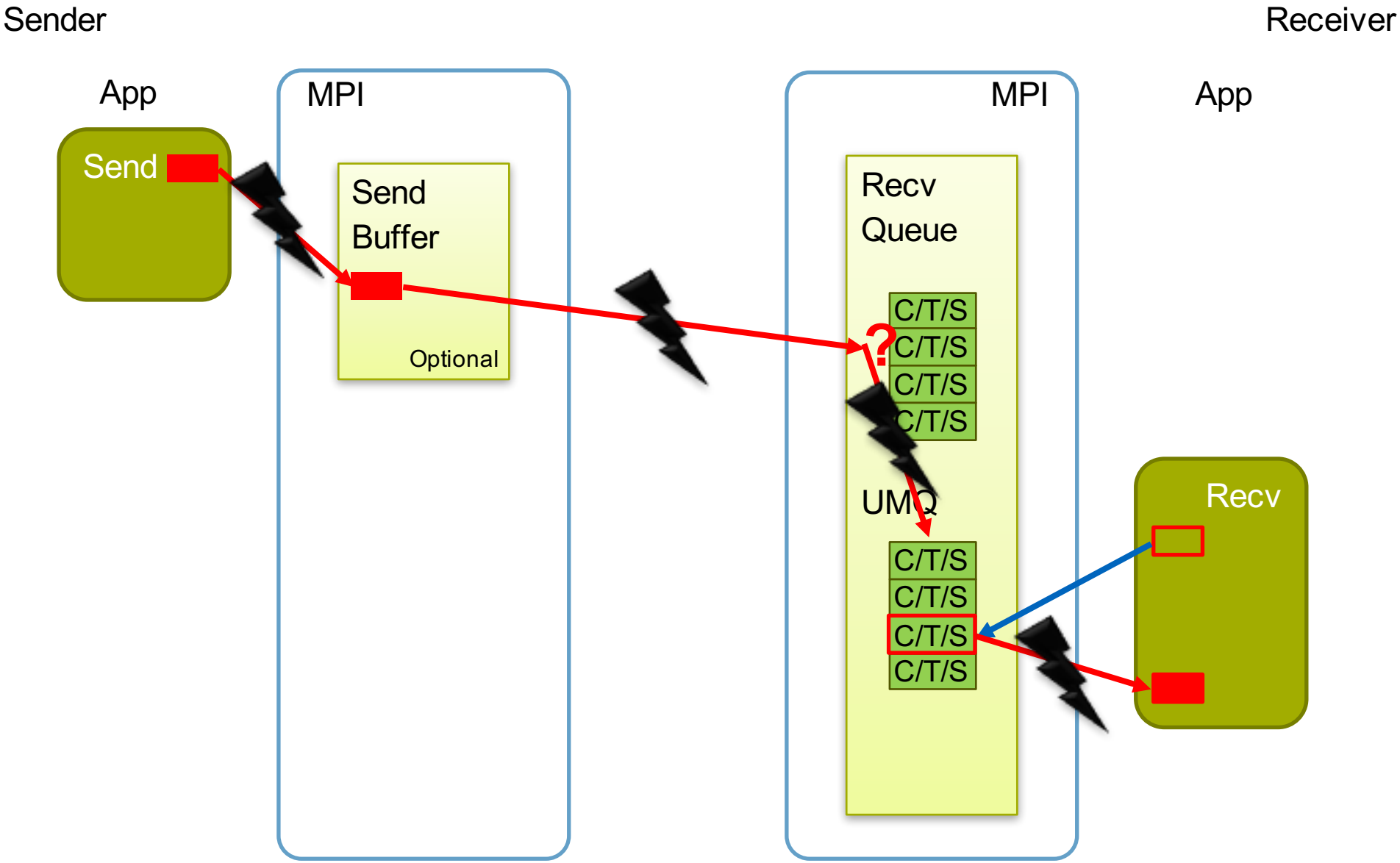
Allow polling of incoming messages without actually receiving them.

- `MPI_Probe`
  - Blocks until a matching message was found
  - Input: source, tag, communicator
  - Output: status object
- `MPI_Iprobe`
  - Terminates after checking whether a matching messages is available
  - Input: source, tag, communicator
  - Output: flag (message found yes/no) and status object

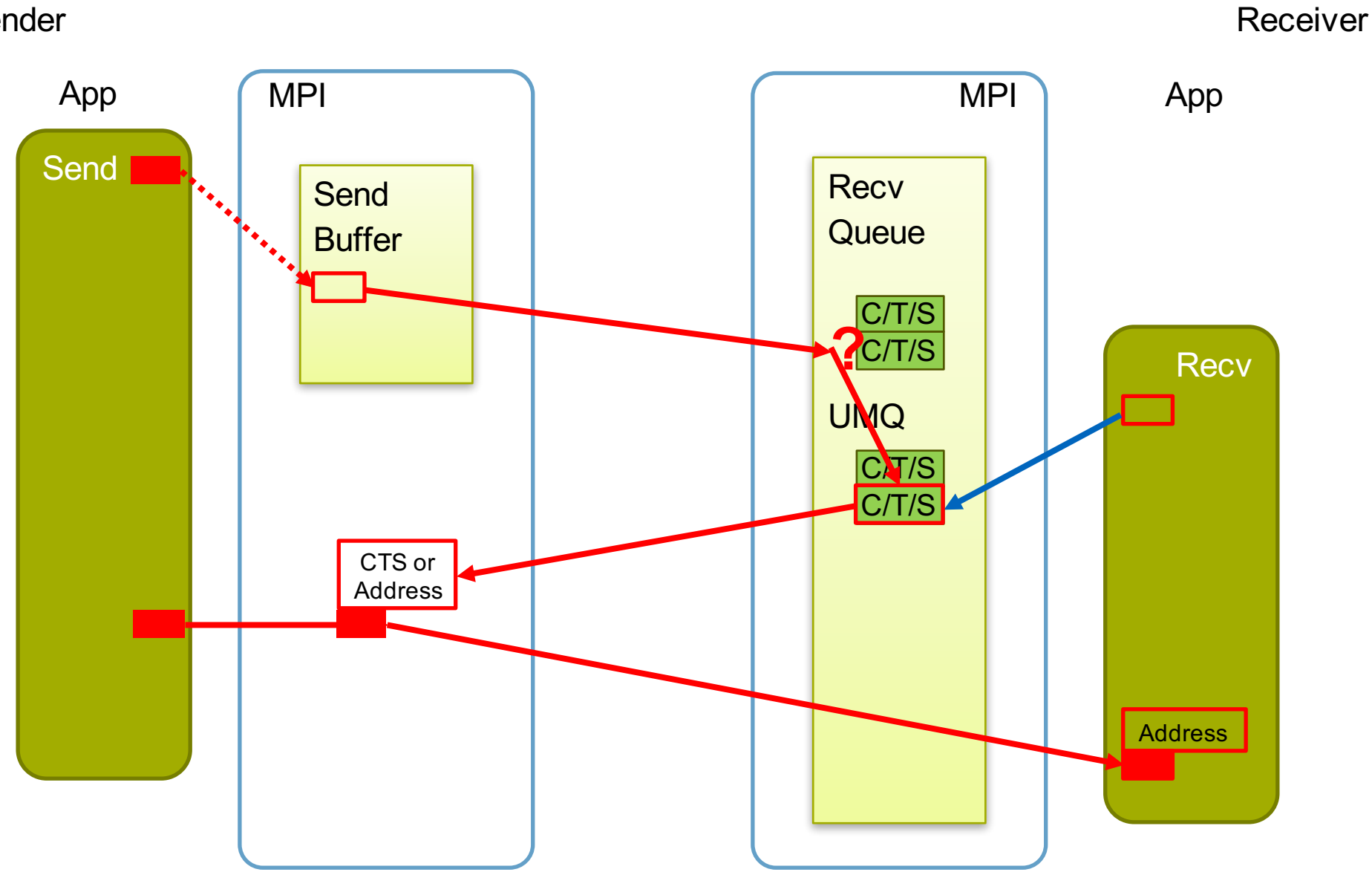
Usage

- 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



Rendevouz Protocol



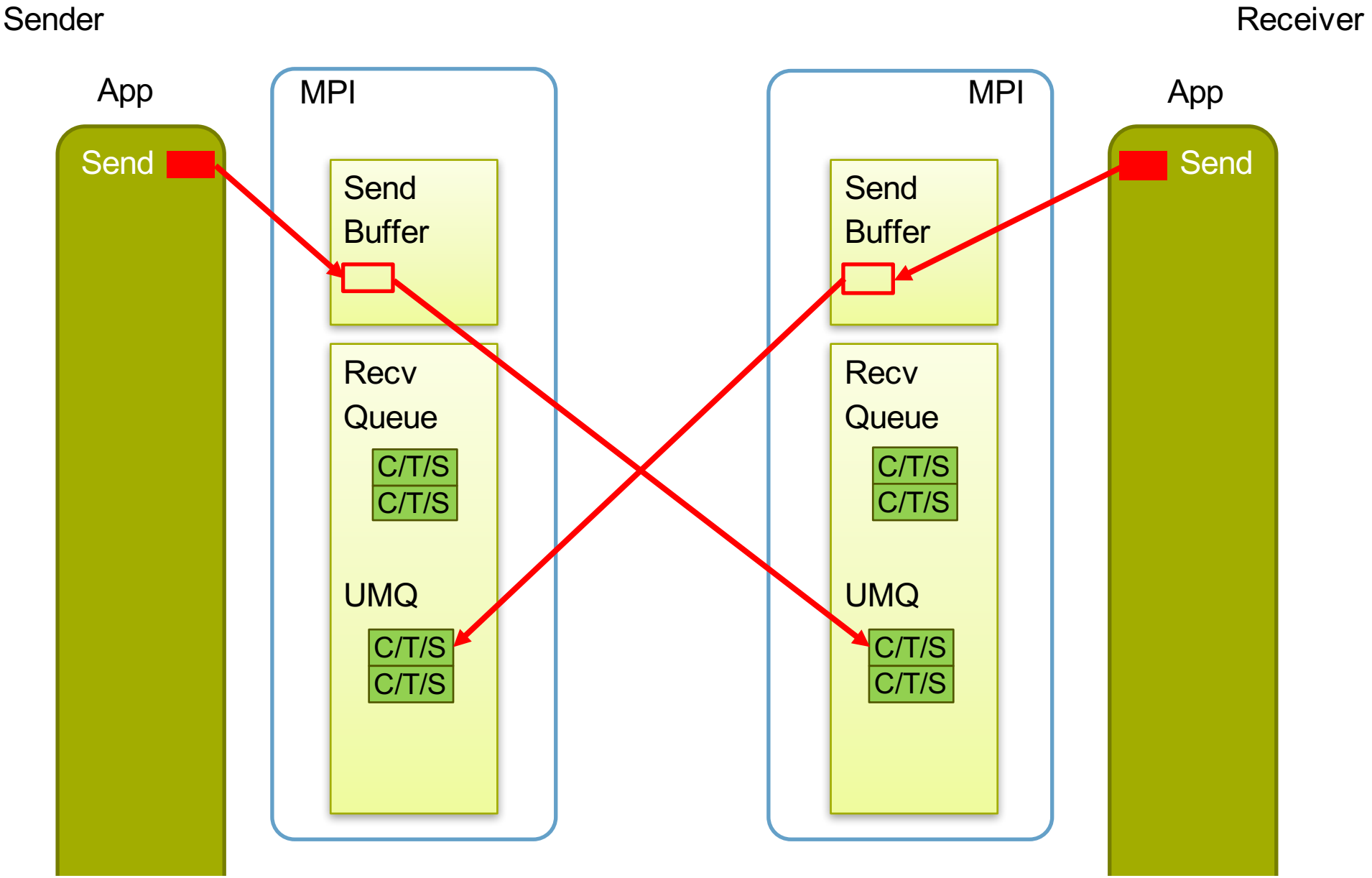
Eager vs. Rendezvous Protocol

Eager protocol
- 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

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



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

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

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,
                         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?

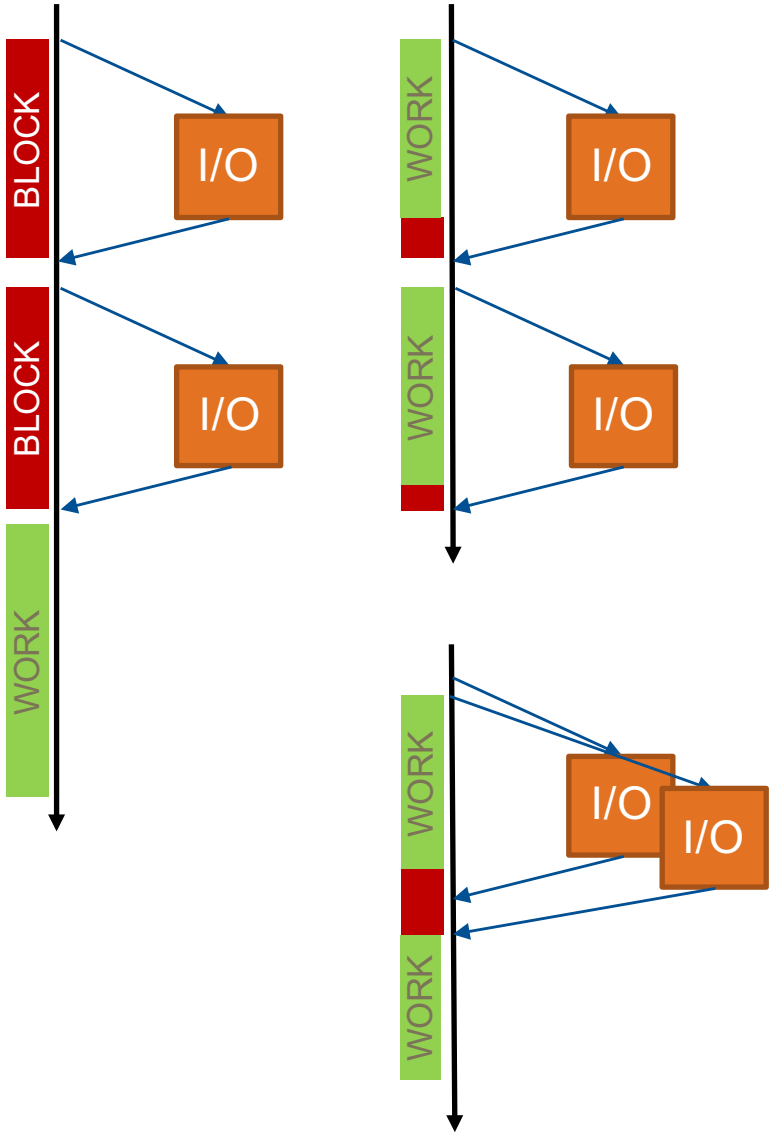
Non-blocking Operations

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



Non-blocking P2P Operations in MPI

Initiation routines as counterparts to blocking P2P operations

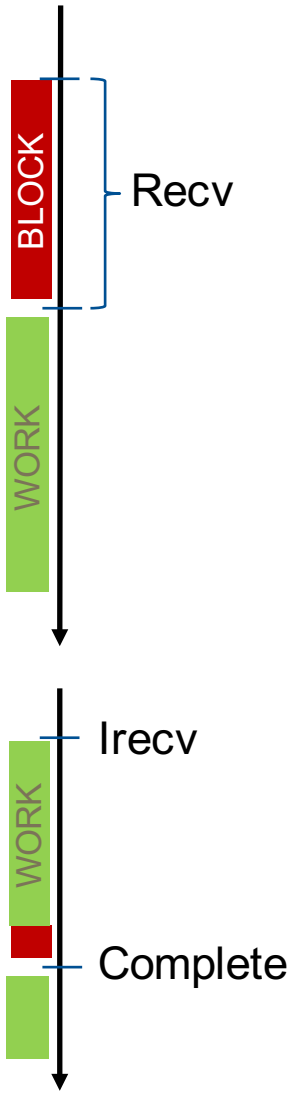
- Similar arguments, same semantics
- Can be matched with blocking operations
- Returns request object to track progress

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



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`

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_Test`
- Checks completion of one request
- Output: flag, True if complete
- 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_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_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_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_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
- Again tied to a communicator
- Enables implicit communication through nodes

Collective Operations

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

MPI provides three classes of collective operations
- Synchronization
  - Barrier
- Communication, e.g.,
  - Broadcast
  - Gather
  - Scatter
- Reduction, e.g.,
  - Global value returned to one or all processes.
  - Combination with subsequent scatter.
  - Parallel prefix operations

MPI_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);
...
```

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*

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

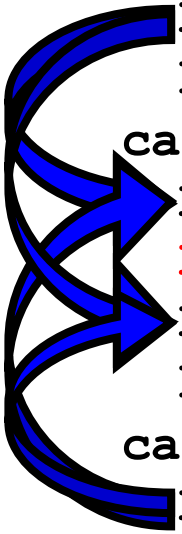
Collective vs P2P operations
- No tag
- Number of elements sent must be equal to number of elements received

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;
}
```

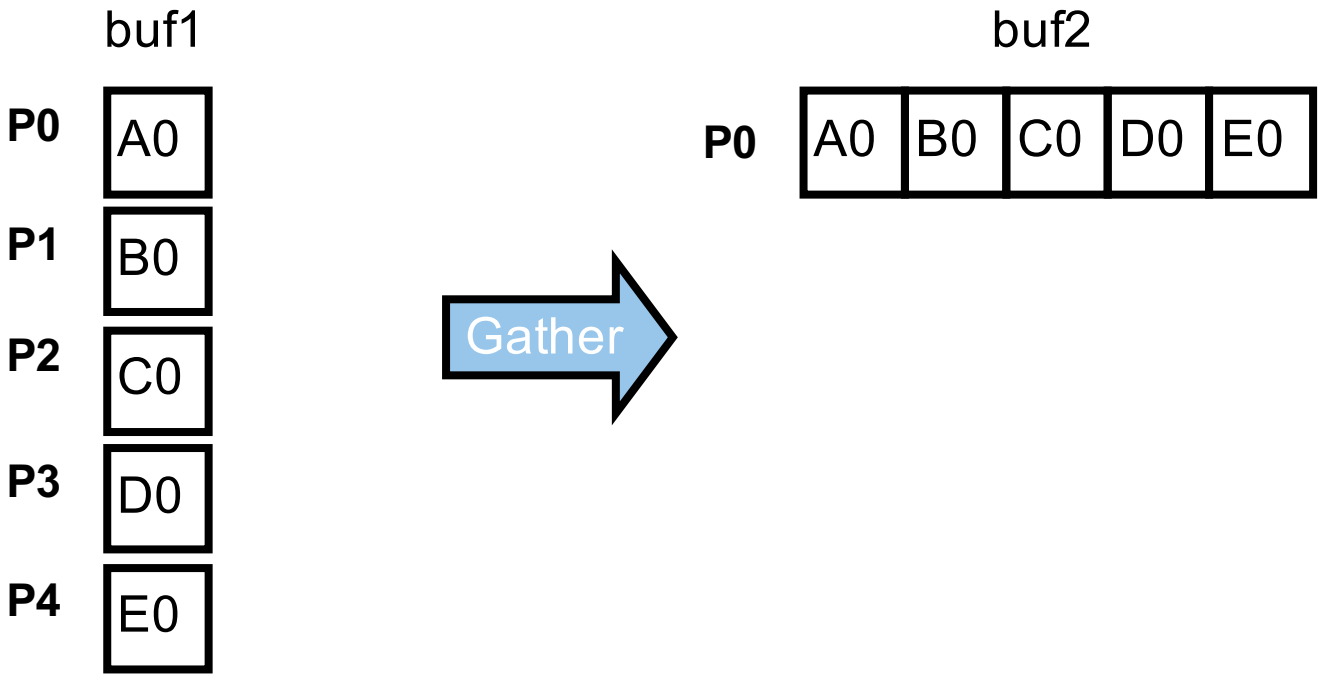
Example



```
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)
```

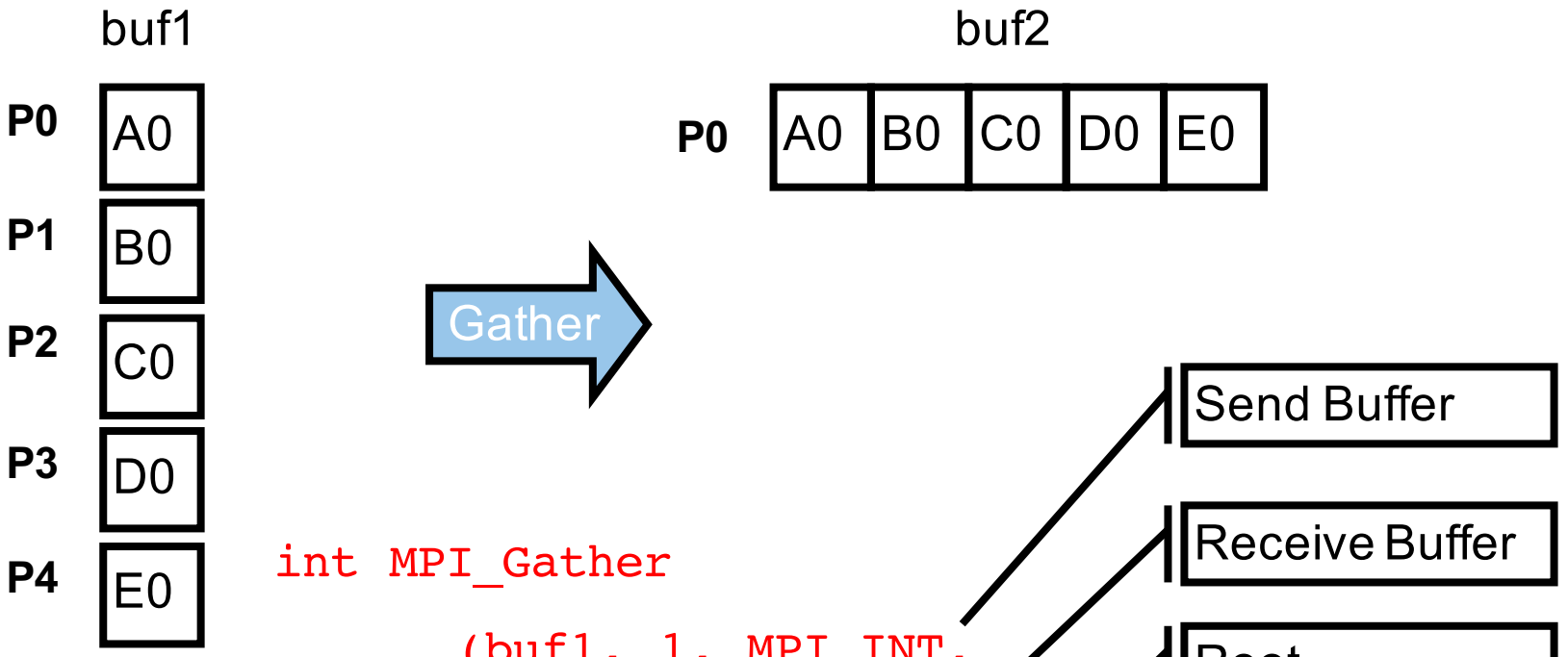
IN sendbuf:	*Send buffer*
IN sendcount:	*Number of elements to be sent to the root*
IN sendtype:	*Data type*
OUT recvbuf:	*Receive buffer*
IN recvcount:	*Number of elements to be received from each MPI process.*
IN recvtype:	*Data type*
IN root:	*Rank of receiving MPI Process*
IN comm	*Communicator*

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

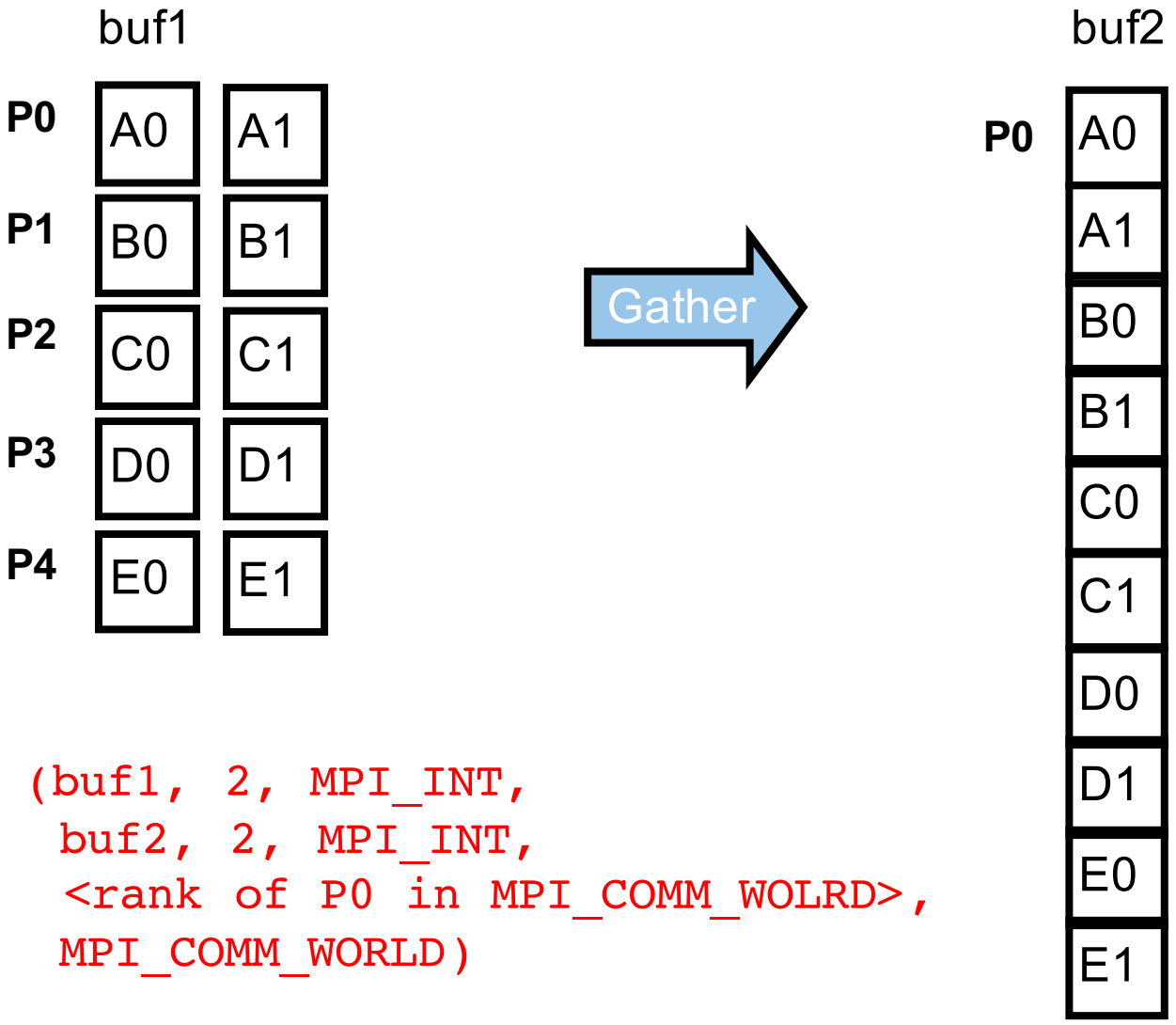
It stores the data in the receive buffer ordered by the process number of the senders.

Gather Operation

Get data from all MPI processes into one local array

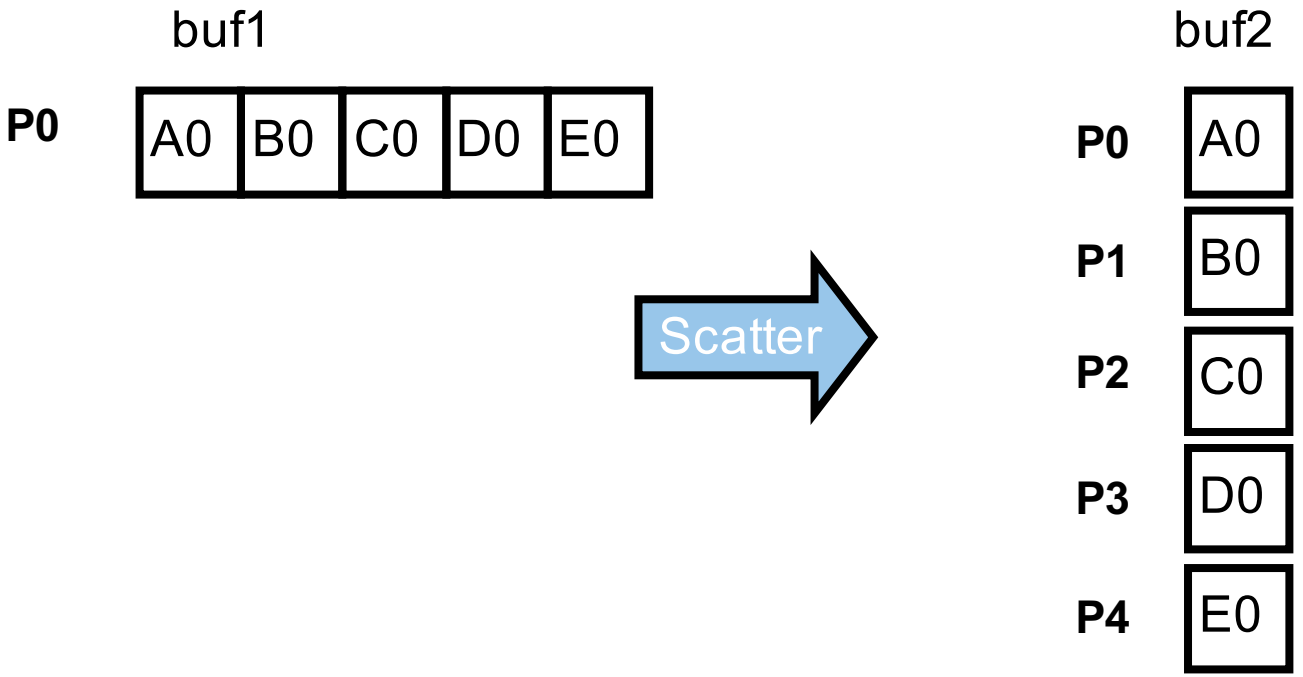


Gather Example




```
int MPI_Gather (buf1, 2, MPI_INT,
                buf2, 2, MPI_INT,
                <rank of P0 in MPI_COMM_WOLRD>,
                MPI_COMM_WORLD)
```

Scatter



MPI_Scatter

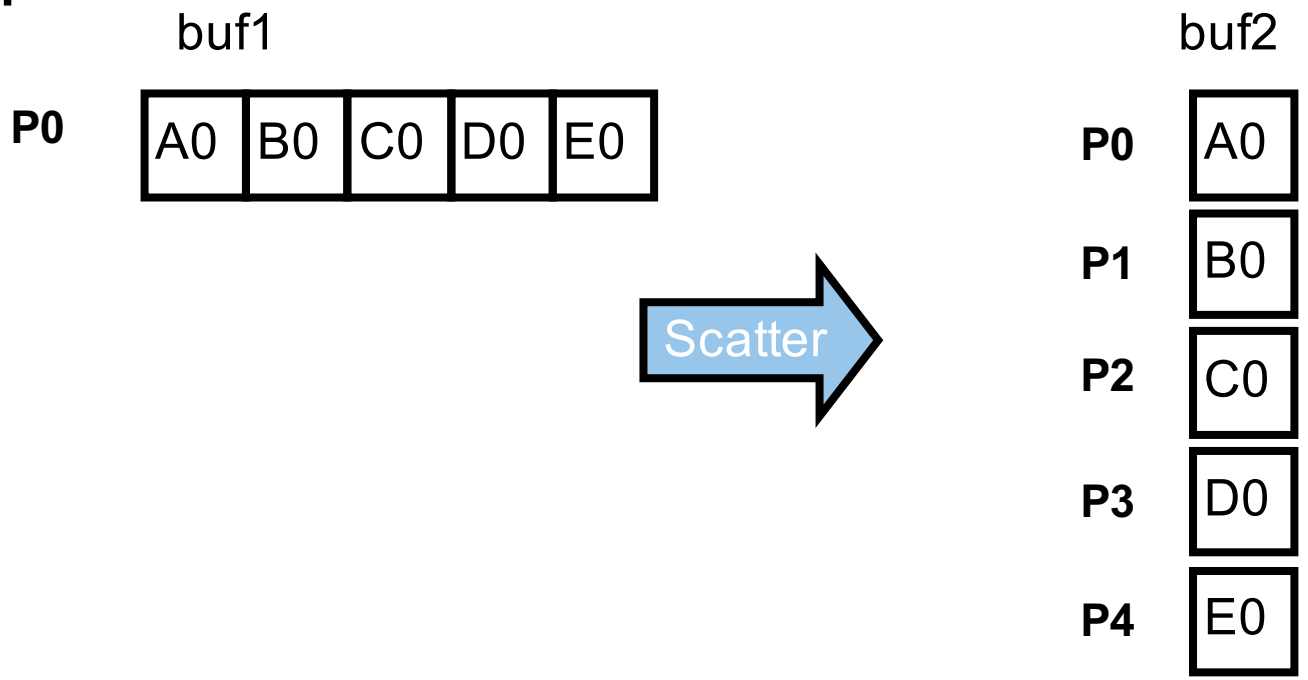
```
int MPI_Scatter (void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 void* recvbuf, int recvcount, MPI_Datatype recvtype,
                 int root, MPI_Comm comm)
```

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

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

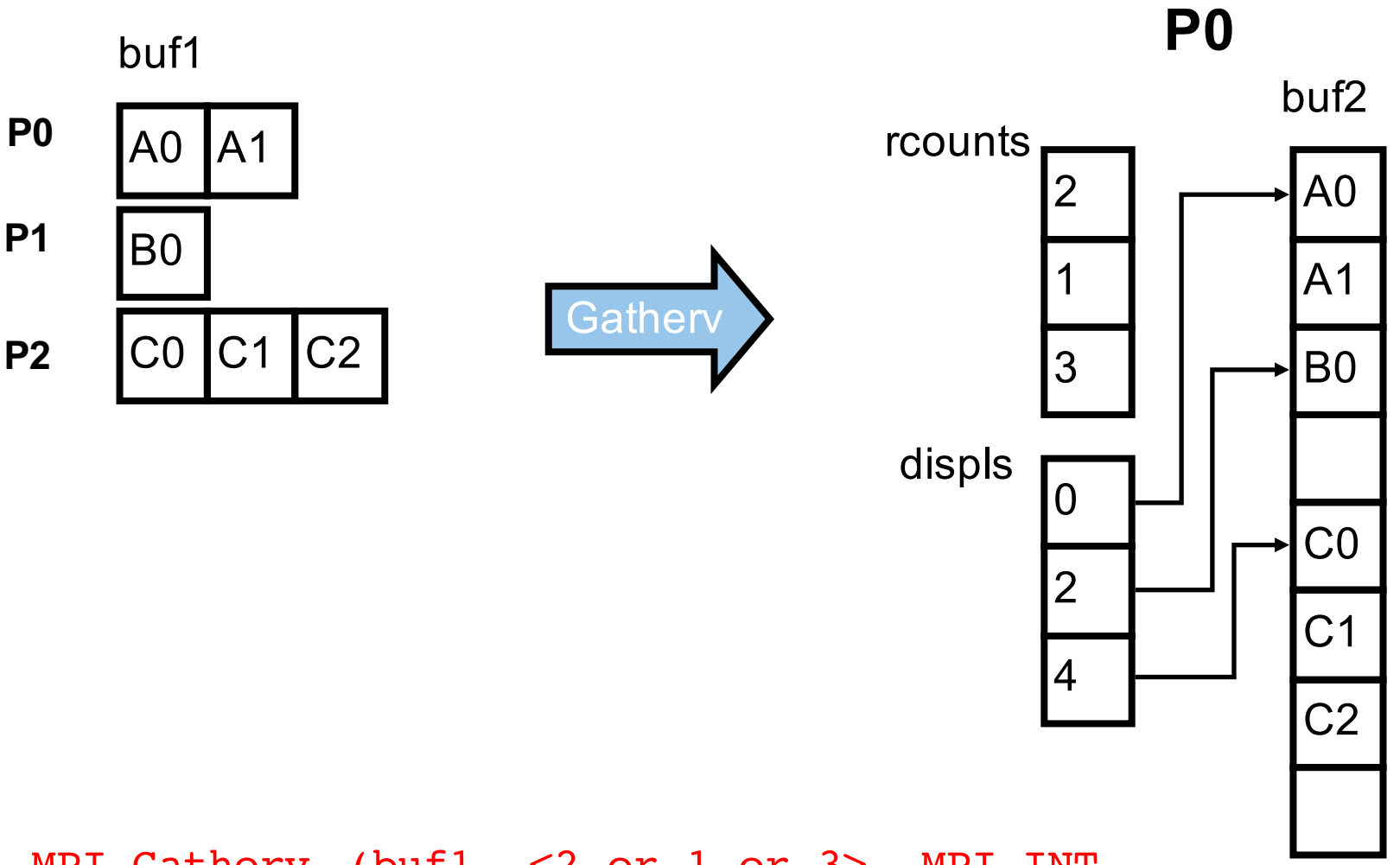
```
int MPI_Gatherv (void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 void* recvbuf, int *recvcount, int *displs,
                 MPI_Datatype recvtype, int root, MPI_Comm comm)
```

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

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

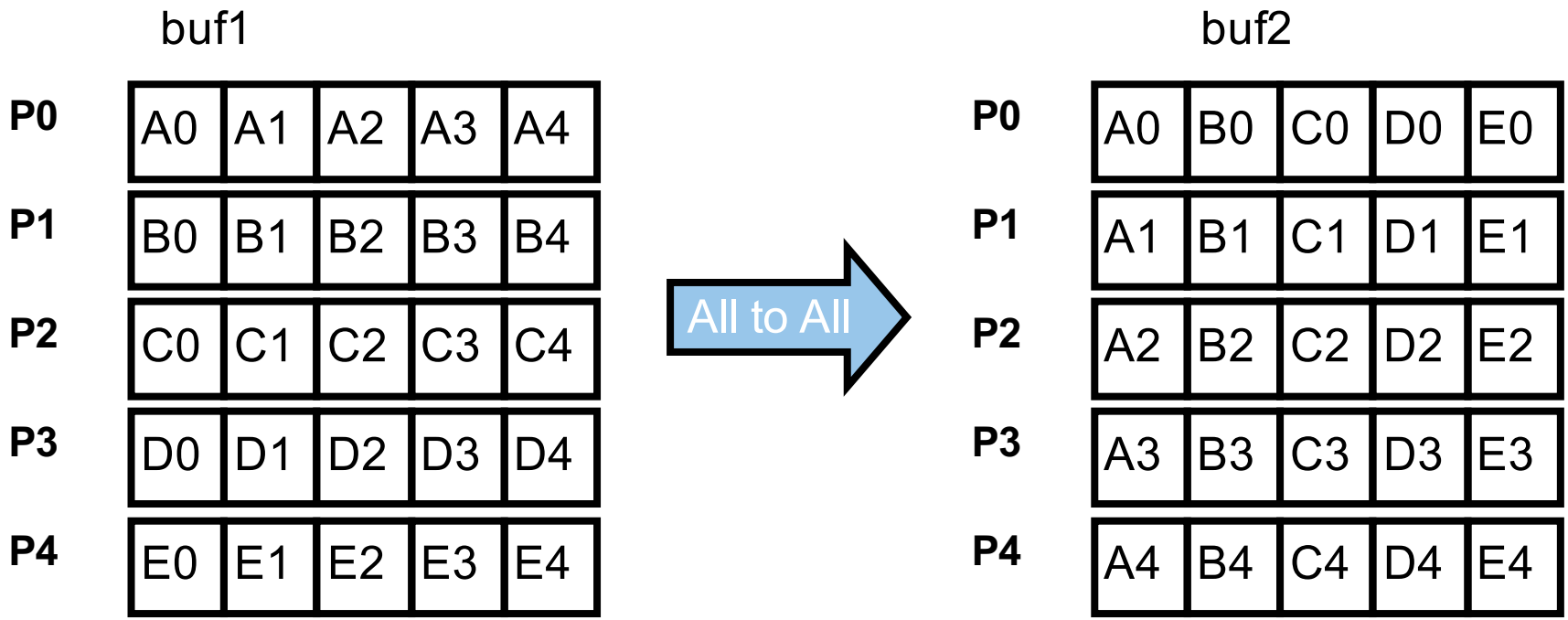
```
int MPI_Alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype,
                 void* recvbuf, int recvcount, MPI_Datatype recvtype,
                 MPI_Comm comm)
```

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

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	*Parent communicator*
IN color	*Subset color*
IN key	*Key to determine rank order in new communicator*
OUT newcomm	*New communicator*

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

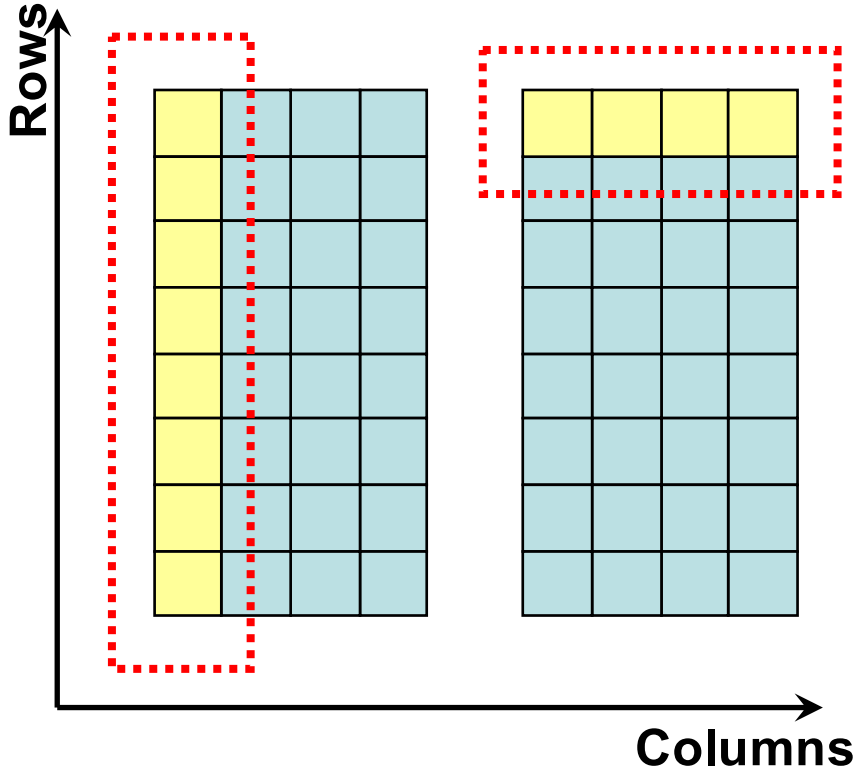
- Pass `MPI_UNDEFINED` as color
- Will return `MPI_COMM_NULL` as new communicator

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