MPI 3.0 Neighbourhood Collectives

Advanced Message-Passing Programming

Overview

- Review of topologies in MPI

- MPI 3.0 includes new neighbourhood collective operations:
  - MPI_Neighbor_allgather[v]
  - MPI_Neighbor_alltoall[v|w]
- Example usage:
  - Halo-exchange can be done with a single MPI communication call

- Practical:
  - Replace all point-to-point halo-exchange communication with a single neighbourhood collective in your MPP coursework code

Topology communicators

- Regular n-dimensional grid or torus topology
  - MPI_CART_CREATE
- General graph topology
  - MPI_GRAPH_CREATE
    - All processes specify all edges in the graph (not scalable)
- General graph topology (distributed version)
  - MPI_DIST_GRAPH_CREATE_ADJACENT
    - All processes specify their incoming and outgoing neighbours
  - MPI_DIST_GRAPH_CREATE
    - Any process can specify any edge in the graph (too general?)

Topology communicators

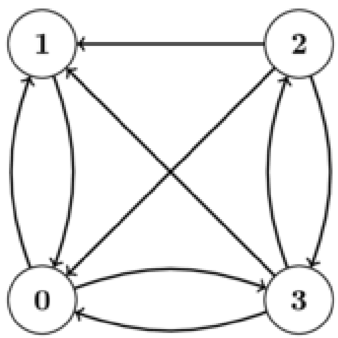
- Testing the topology type associated with a communicator
  - MPI_TOPO_TEST
- Finding the neighbours for a process
  - MPI_CART_SHIFT

  - Find out how many neighbours there are:
    - MPI_GRAPH_NEIGHBORS_COUNT
  - Get the ranks of all neighbours:
    - MPI_GRAPH_NEIGHBORS
  - Find out how many neighbours there are:
    - MPI_DIST_GRAPH_NEIGHBORS_COUNT
  - Get the ranks of all neighbours:
    - MPI_DIST_GRAPH_NEIGHBORS

Example

- Useful example program at: https://riptutorial.com/mpi/example/29195/graph-topology-creation-and-communication

Creates a graph topology in a distributed manner so that each node defines its neighbors. Each node communicates its rank among neighbors with `MPI_Neighbor_allgather`.



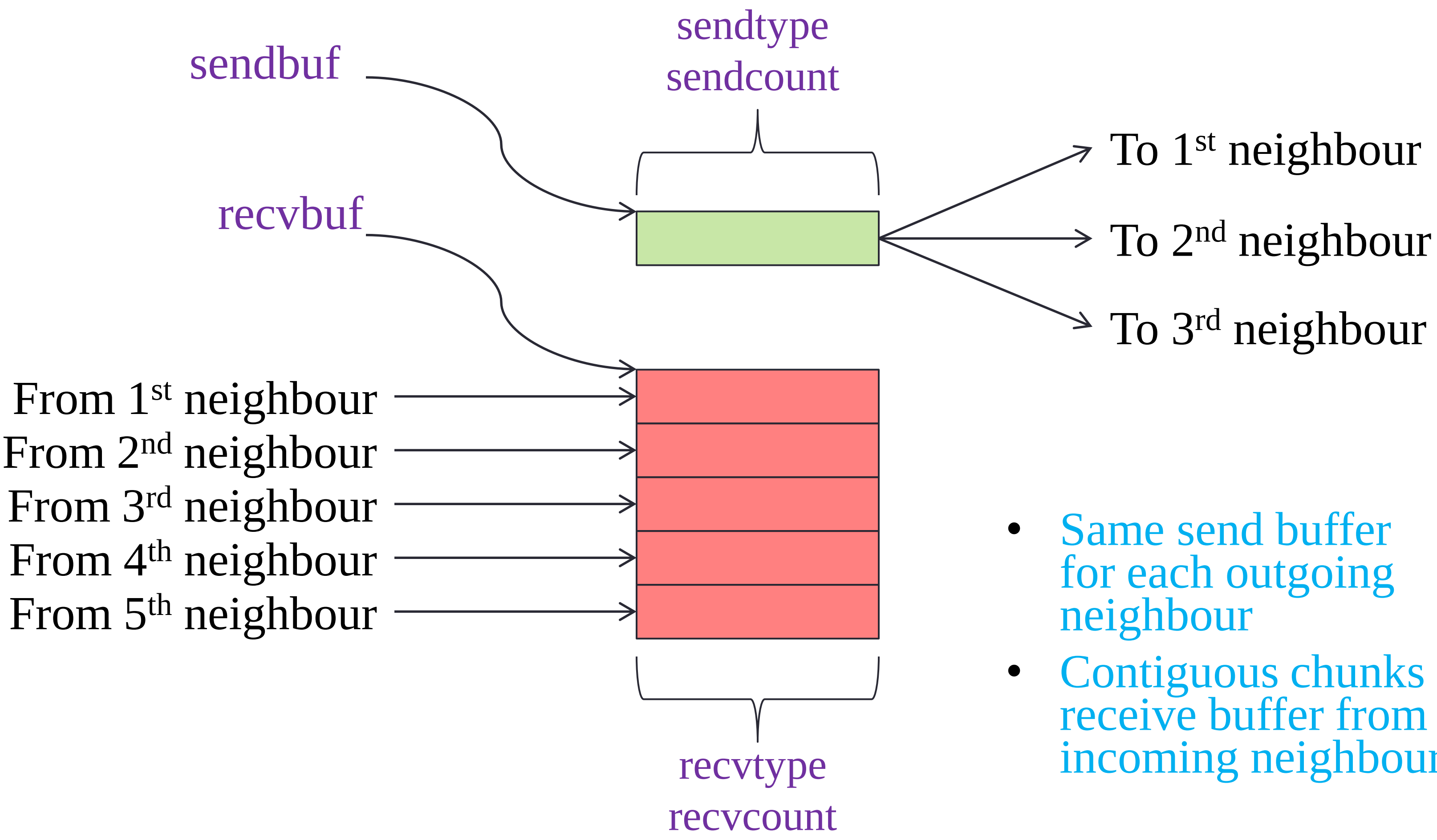
Example (cont)

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

#define nnode 4

int main()
{
    MPI_Init(NULL, NULL);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int source = rank;
    int degree;
    int dest[nnode];
    int weight[nnode] = {1, 1, 1, 1};
    int recv[nnode] = {-1, -1, -1, -1};
    int send = rank;

    // set dest and degree.
    if (rank == 0)
    {
        dest[0] = 1;
        dest[1] = 3;
        degree = 2;
    }
    else if(rank == 1)
    {
        dest[0] = 0;
        degree = 1;
    }
    else if(rank == 2)
    {
        dest[0] = 3;
        dest[1] = 0;
        dest[2] = 1;
        degree = 3;
    }
    else if(rank == 3)
    {
        dest[0] = 0;
        dest[1] = 2;
        dest[2] = 1;
        degree = 3;
    }

    // create graph.
    MPI_Comm graph;
    MPI_Dist_graph_create(MPI_COMM_WORLD, 1, &source, &degree, dest, weight,
                          MPI_INFO_NULL, 1, &graph);

    // send and gather rank to/from neighbors.
    MPI_Neighbor_allgather(&send, 1, MPI_INT, recv, 1, MPI_INT, graph);

    printf("Rank: %i, recv[0] = %i, recv[1] = %i, recv[2] = %i, recv[3] = %i\n",
            rank, recv[0], recv[1], recv[2], recv[3]);

    MPI_Finalize();
    return 0;
}

// Taken from https://riptutorial.com/mpi/example/29195/graph-topology-creation-
and-communication
```

Neighbourhood collective operations

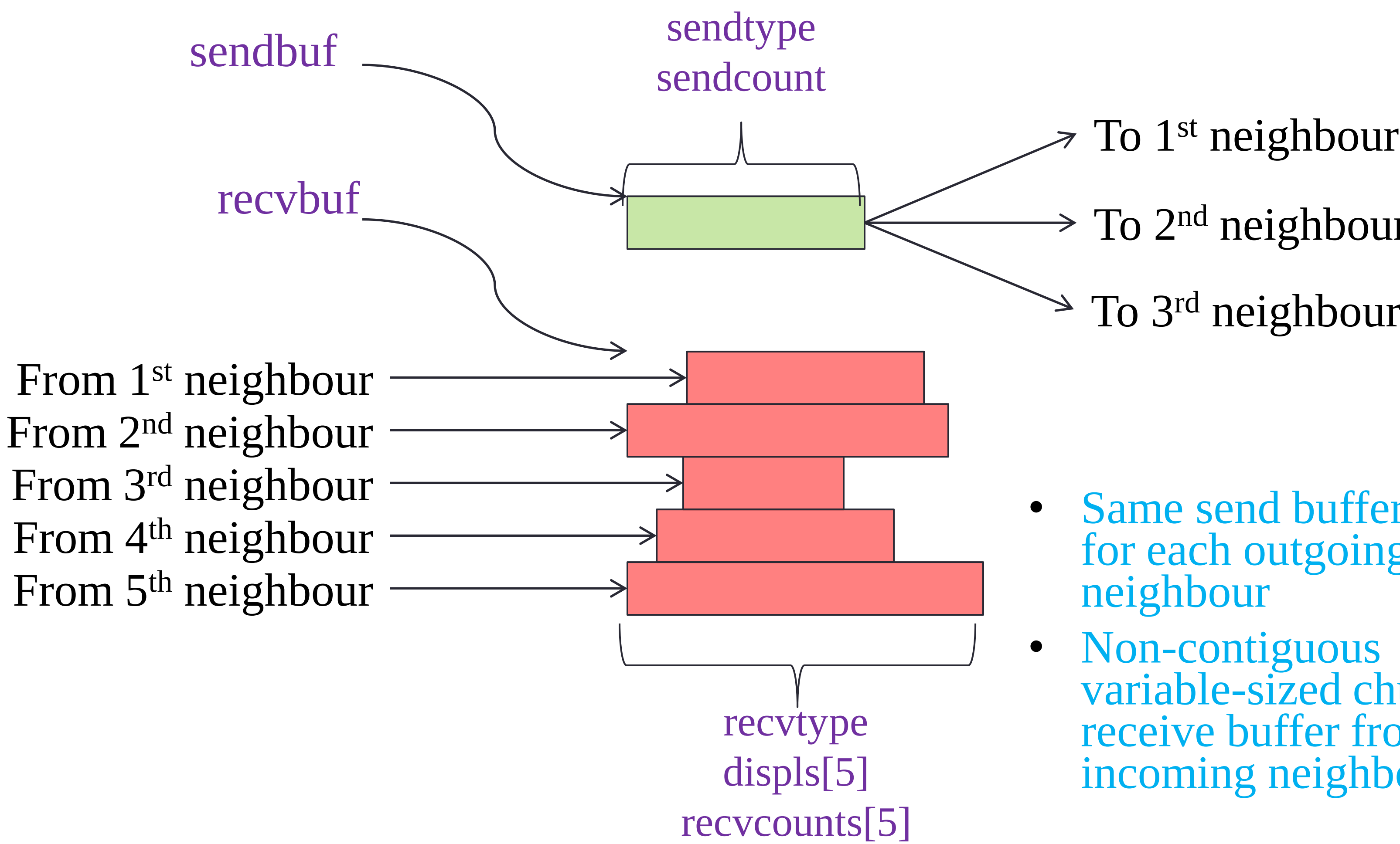
- See section 7.6 in MPI 3.0 for blocking functions
  - See section 7.7 in MPI 3.0 for non-blocking functions
  - See section 7.8 in MPI 3.0 for an example application
    - But beware of the mistake(s) in the example code!
- MPI_[N|In]eighbor_allgather[v]
  - Send one piece of data to all neighbours
  - Gather one piece of data from each neighbour
- MPI_[N|In]eighbor_alltoall[v|w]
  - Send different data to each neighbour
  - Receive different data from each neighbour
- Use-case: regular or irregular domain decompositions
  - Where the decomposition is static or changes infrequently
  - Because creating a topology communicator takes time

MPI_Neighbor_allgather



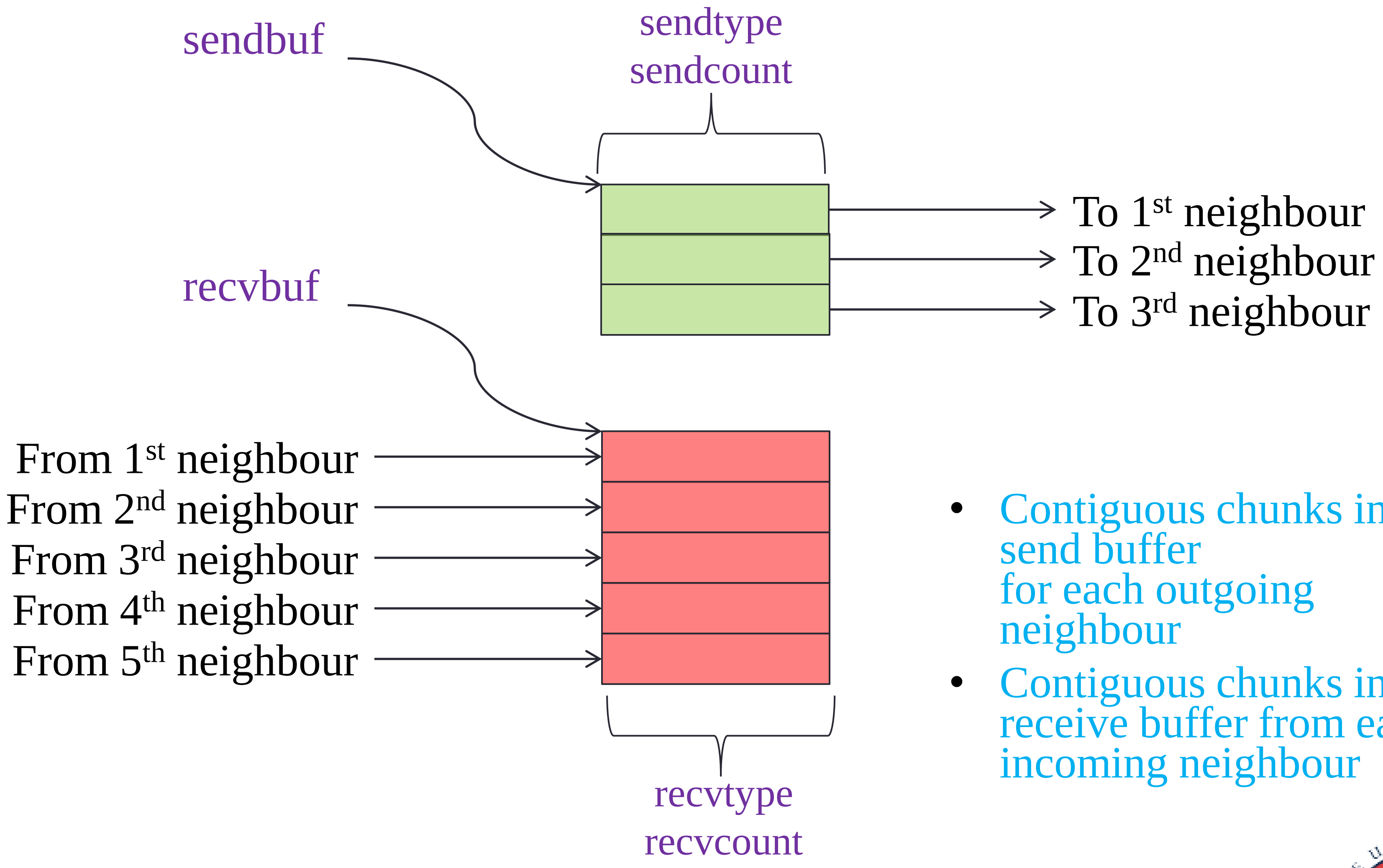
- Same send buffer for each outgoing neighbour
- Contiguous chunks in receive buffer from each incoming neighbour

MPI_Neighbor_allgatherv



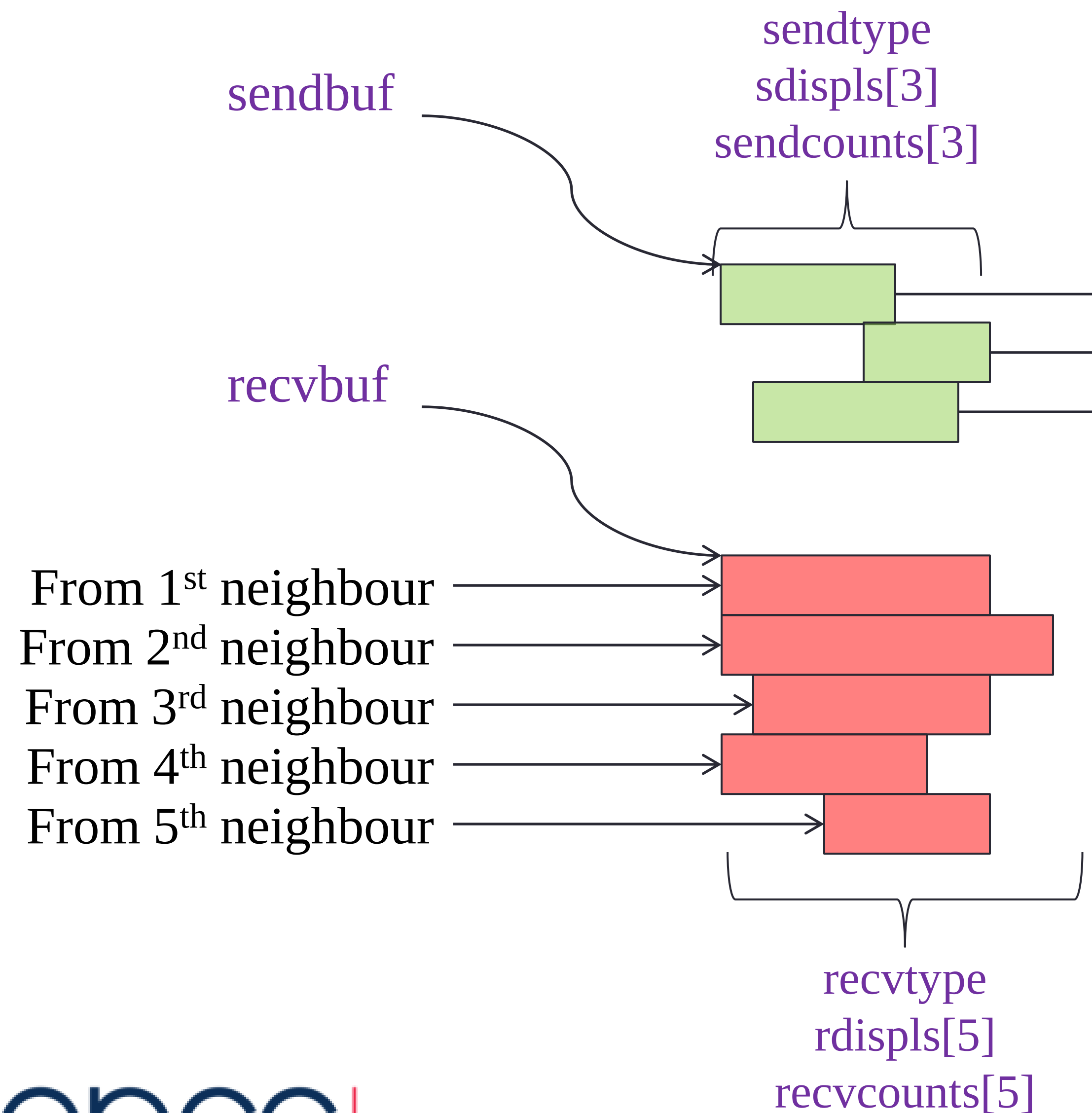
- Same send buffer for each outgoing neighbour
- Non-contiguous variable-sized chunks in receive buffer from each incoming neighbour

MPI_Neighbor_alltoall



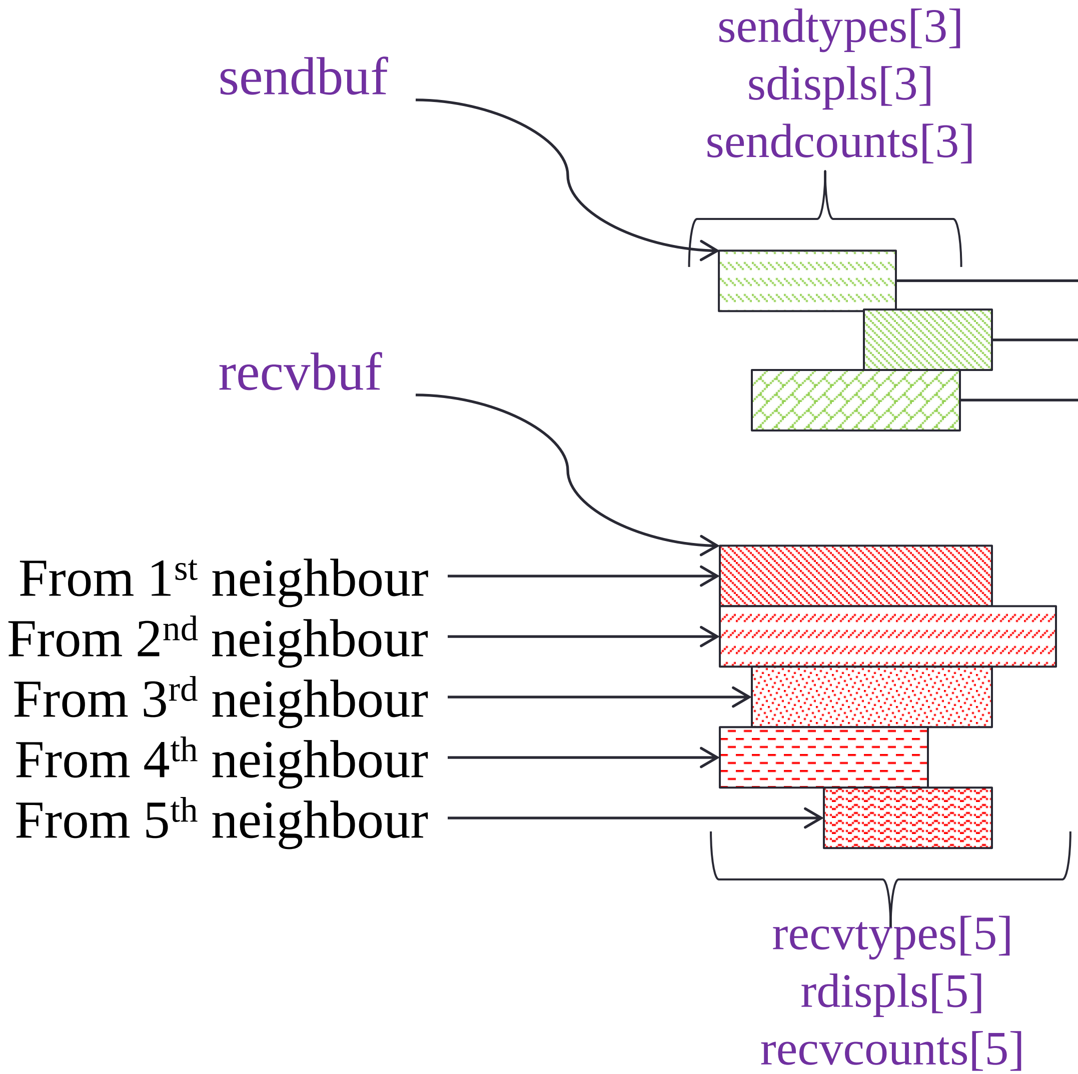
- Contiguous chunks in send buffer for each outgoing neighbour
- Contiguous chunks in receive buffer from each incoming neighbour

MPI_Neighbor_alltoallv



- Non-contiguous variable-sized chunks in send buffer
for each outgoing neighbour
- Non-contiguous variable-sized chunks in receive buffer from each incoming neighbour

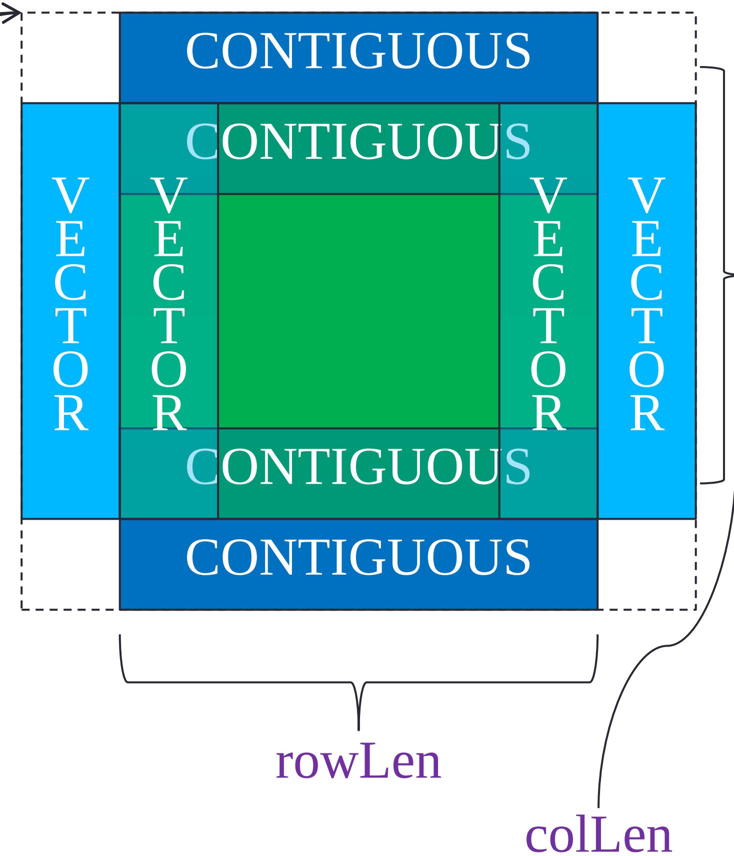
MPI_Neighbor_alltoallw



- Non-contiguous variable-sized chunks in send buffer
  for each outgoing neighbour
- Non-contiguous variable-sized chunks in receive buffer from each incoming neighbour
- Displacements in bytes

MPI_Neighbor_alltoallw

```
for (int i=0;i<4;++i) {
    sendcounts[i] = 1;
    recvcounts[i]=1; }

sendtypes[0] = contigType;
senddispls[0] = (colLen*(rowLen+2)+1)*dblesize;
sendtypes[1] = contigType;
senddispls[1] = (1*(rowLen+2)+1) )*dblesize;
sendtypes[2] = vectorType;
senddispls[2] = (1*(rowLen+2)+1)*dblesize;
sendtypes[3] = vectorType;
senddispls[3] = (2*(rowLen+2)-2) )*dblesize;

// similarly for recvtypes and recvdispls

MPI_Neighbor_alltoallw(sendbuf, sendcounts, senddispls, sendtypes,
                       recvbuf, recvcounts, recvdsipls, recvtypes,
                       comm);
```



Summary

- Useful for regular or irregular domain decomposition
  - Where the decomposition is static or changes infrequently
- Investigate replacing point-to-point communication
  - E.g. halo-exchange communication
- With neighbourhood collective communication
  - Probably MPI_Neighbor_alltoallw / MPI_Ineighbor_alltoallw
- So that MPI can optimise the whole pattern of messages
  - Rather than trying to optimise each message individually
- And so your application code is simpler and easier to read