

Neighbourhood Collectives

EPSRC

NERC SCIENCE OF THE ENVIRONMENT

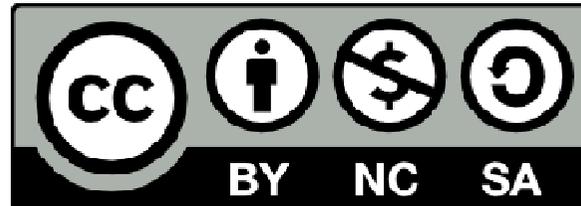


CRAY
THE SUPERCOMPUTER COMPANY

epcc



Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.



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

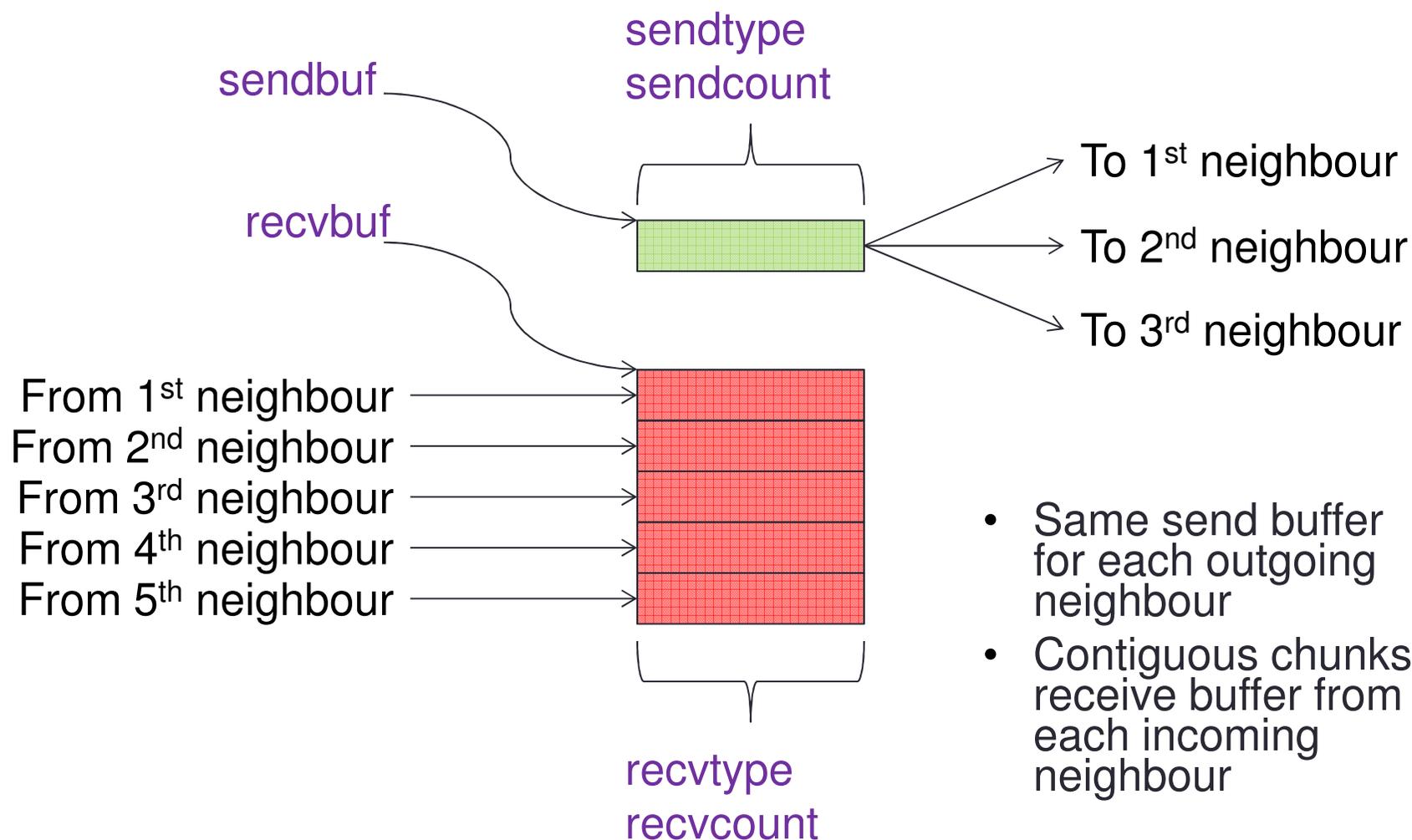


Neighbourhood collective operations

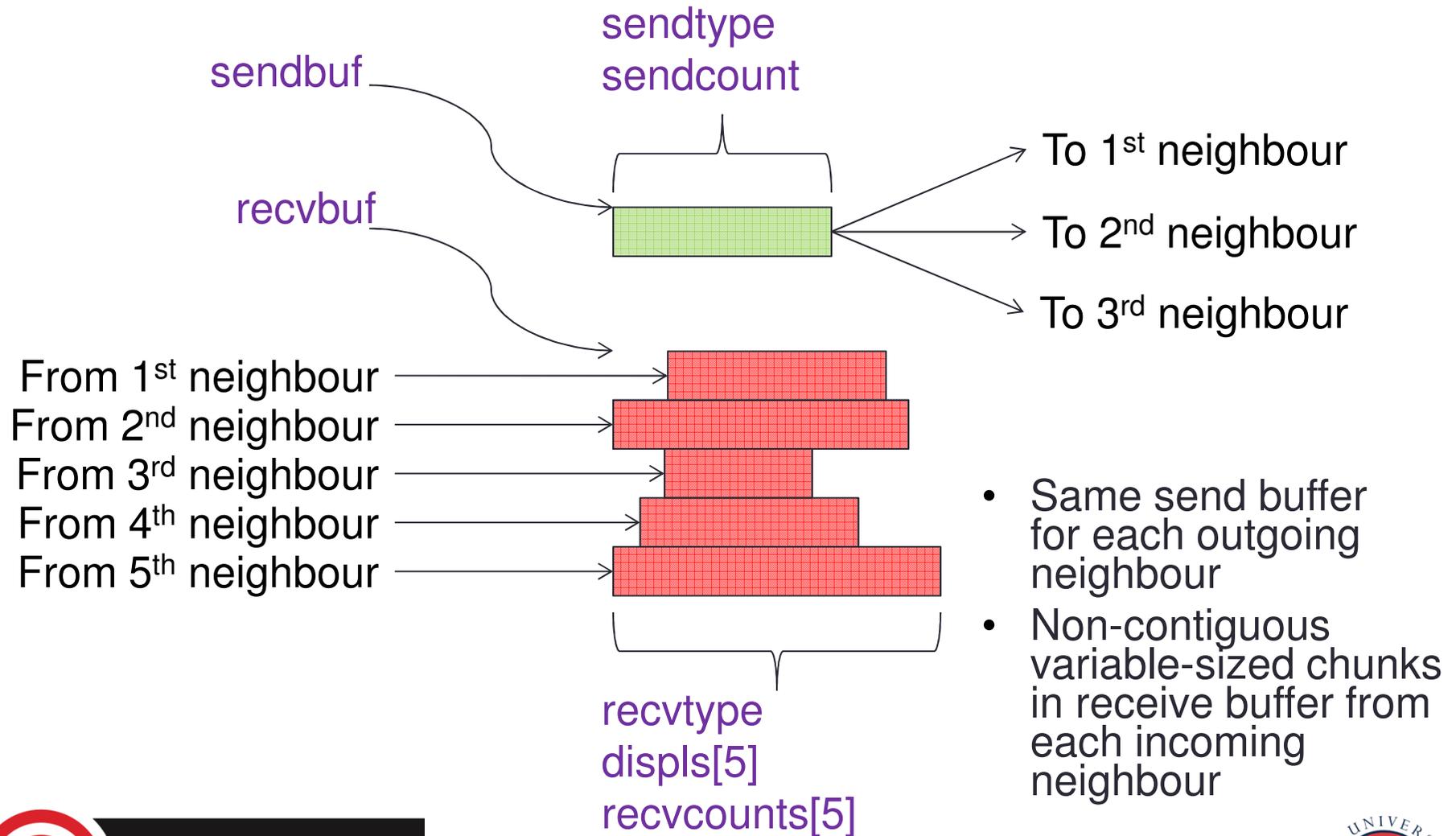
- See section 7.6 in MPI 3.0 for blocking functions
 - See section 7.7 in MPI 3.0 for non-blocking functions
 - See section 7.8 in MPI 3.0 for an example application
 - But beware of the mistake(s) in the example code!
- `MPI_[N|In]ighbor_allgather[v]`
 - Send one piece of data to all neighbours
 - Gather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



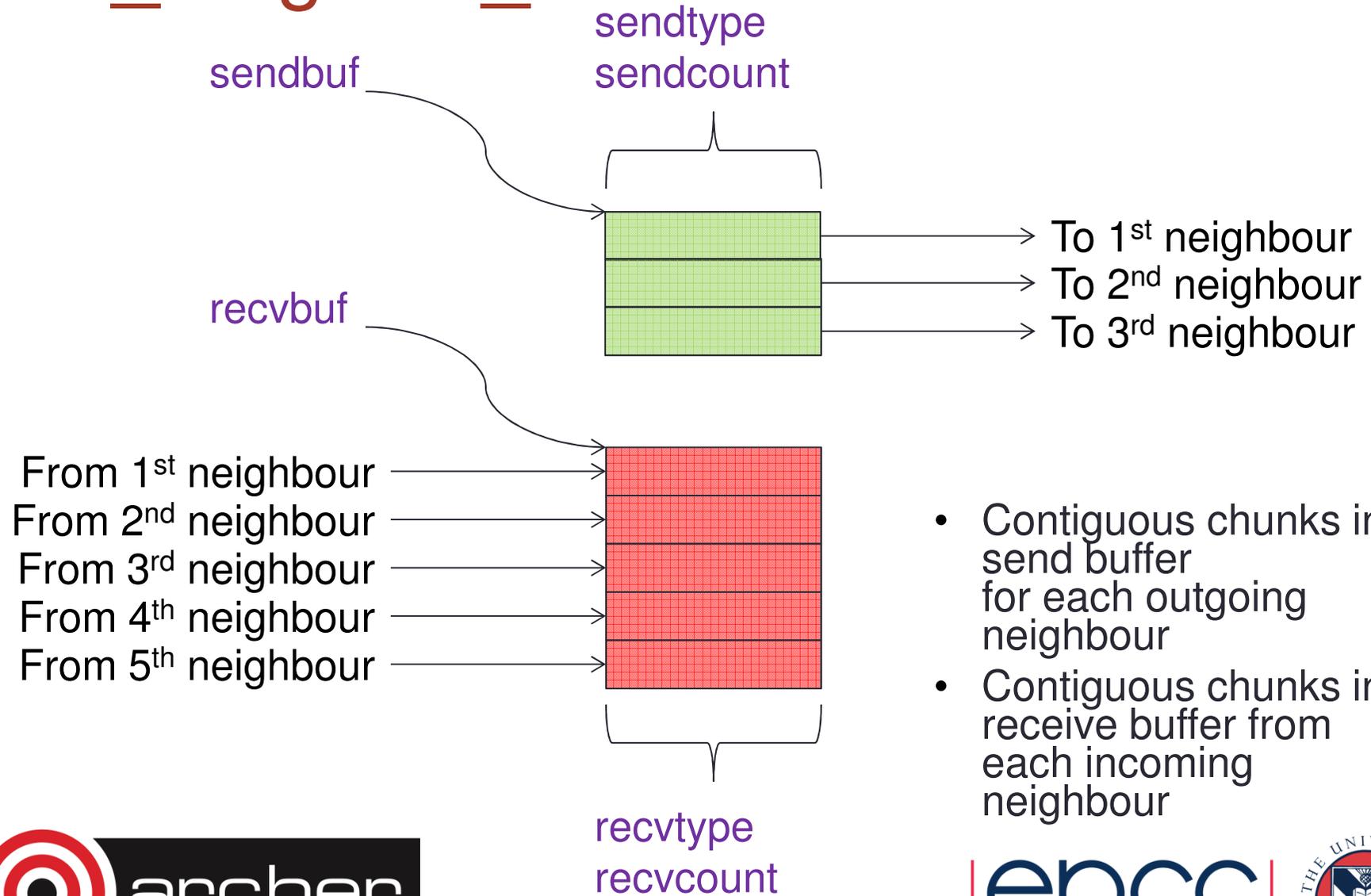
MPI_Neighbor_allgather



MPI_Neighbor_allgatherv

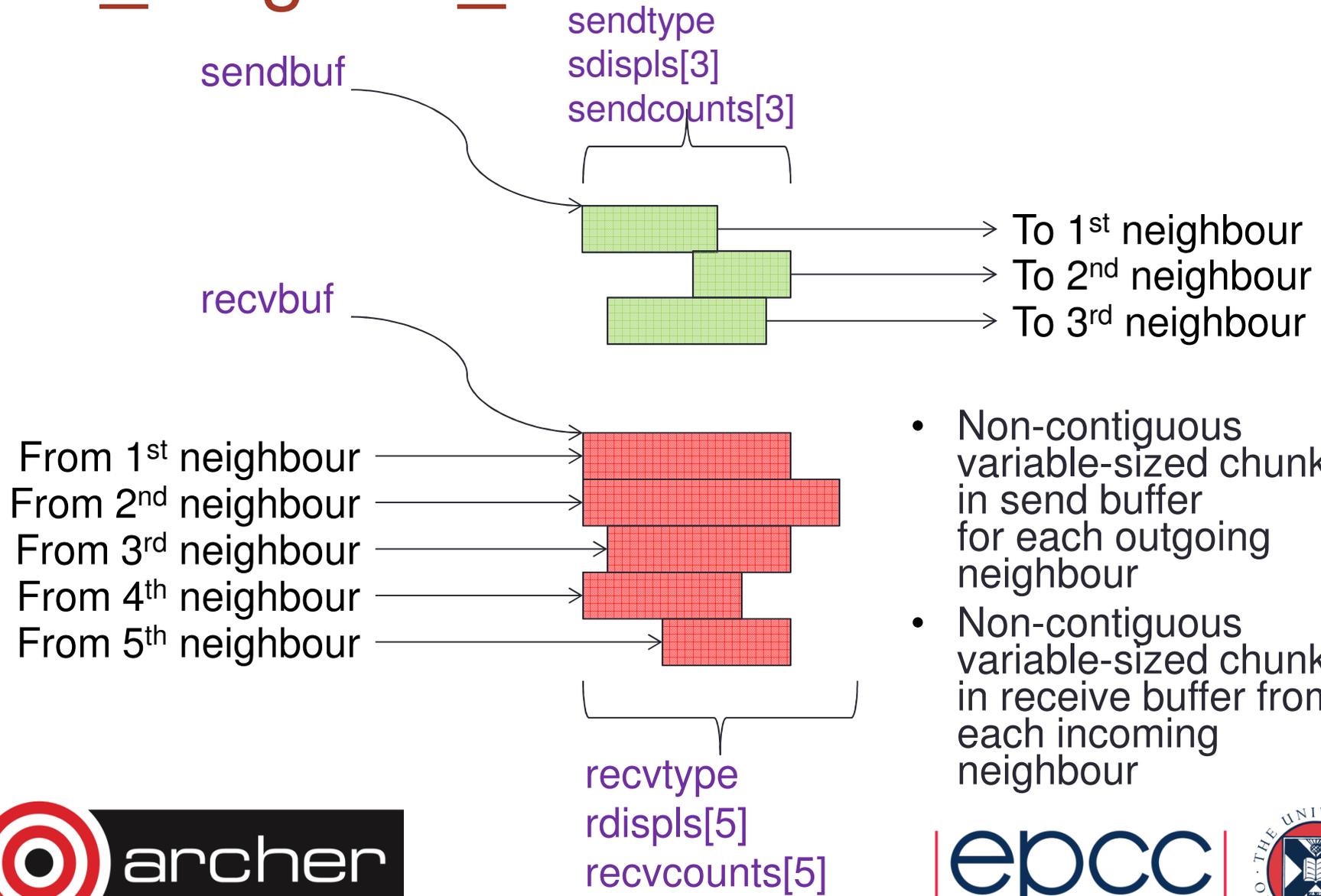


MPI_Neighbor_alltoall



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

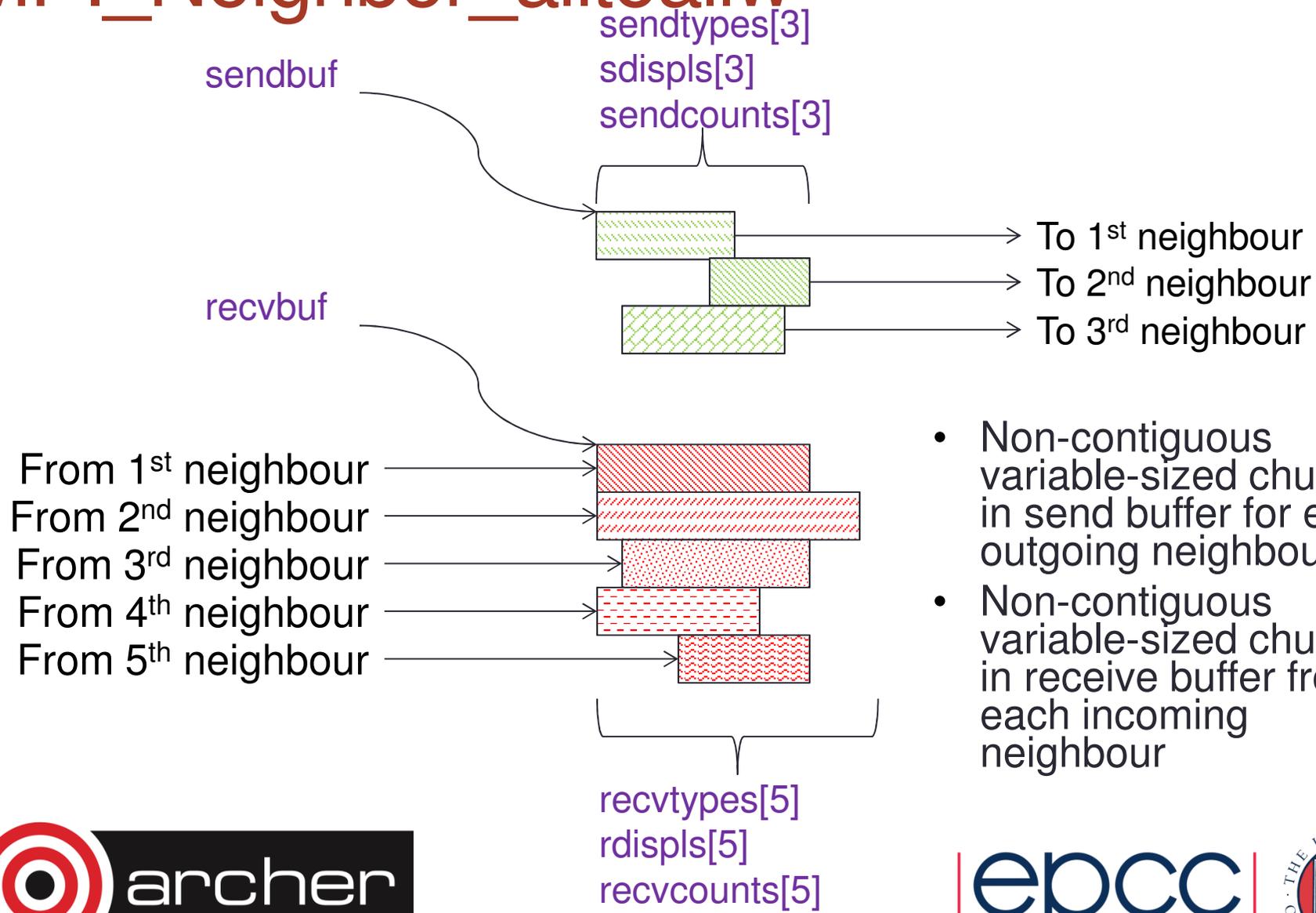
MPI_Neighbor_alltoallv



- 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

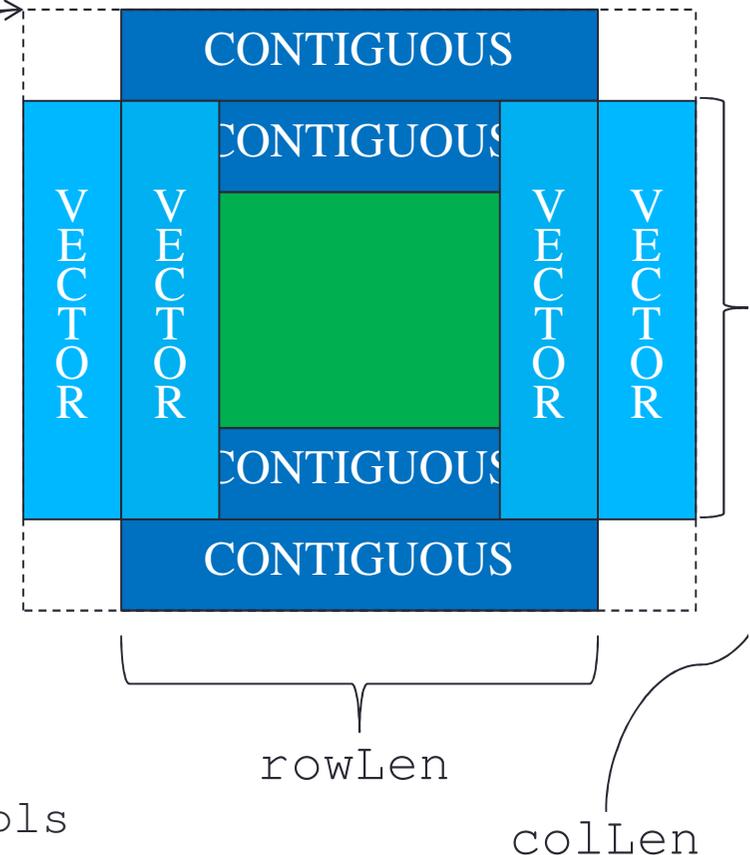


- Non-contiguous variable-sized chunks in send buffer for each outgoing neighbour
- Non-contiguous variable-sized chunks in receive buffer from each incoming neighbour



MPI_Neighbor_alltoallw

```
for (int i=0;i<4;++i) {sendbuf
    sendcounts[i] = 1;
    recvcounts[i] = 1; recvbuf
}
sendtypes[0] = contigType;
senddispls[0] = colLen*(rowLen+2)+1;
sendtypes[1] = contigType;
senddispls[1] = 1*(rowLen+2)+1;
sendtypes[2] = vectorType;
senddispls[2] = 1*(rowLen+2)+1;
sendtypes[3] = vectorType;
senddispls[3] = 2*(rowLen+2)-2;
// similarly for recvtypes and recvdispls
```



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



Summary

- Regular or irregular domain decomposition codes
 - Where the decomposition is static or changes infrequently
- Should investigate replacing point-to-point communication
 - E.g. halo-exchange communication
- With neighbourhood collective communication
 - Probably 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

