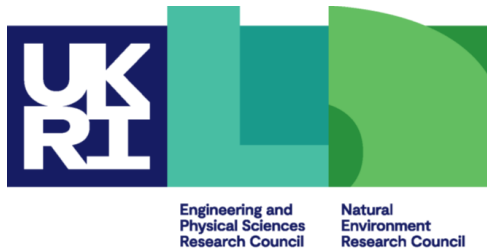# Advanced Collectives

Advanced Message-Passing Programming

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/

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.

Overview

- Motivation
- 2D gather pattern
- MPI_Gather
- Resized datatypes
- MPI_Gatherv
- Other collectives
- Summary

Motivation

- Collectives are a key feature of MPI
  - much simpler to use than implementing your own operations
  - much faster than a DIY approach
- Flexibility in what processes take part
  - e.g. pass a sub-communicator instead of MPI_COMM_WORLD
- However ...
  - what if your data layout does not match the collective's pattern?
  - what if your data type is not supported?
- Solutions
  - derived datatypes
  - derived datatypes + user-defined reduction operations (see later)

Canonical example

- Have a 2D array distributed across a 2D process grid

- Want to use MPI_Gather to collect data on single process
  - e.g. before performing serial master-IO to disk

- Study this particular example in some detail
  - straightforward to generalise to other collectives
  - e.g. MPI_Scatter, MPI_Reduce,, MPI_Allreduce, MPI_Alltoall, ...

- Difficulty is understanding how derived datatypes work with collectives
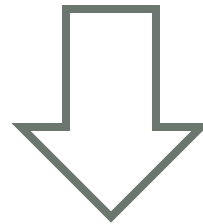 - after that, easy to apply to other cases

Canonical example (global indices)

j

↑

→ i
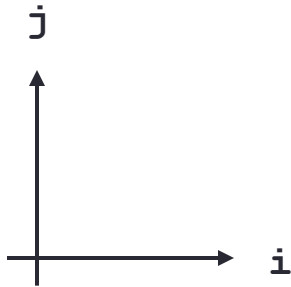
(assume integer arrays
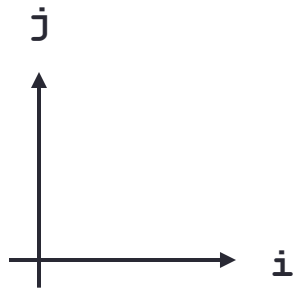and C-like array storage)

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

| rank 1 (0,1) | rank 3 (1,1) |
| rank 0 (0,0) | rank 2 (1,0) |

Gather to rank 0

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

Canonical example (local indices)
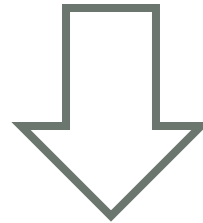
Canonical example (linear buffers)
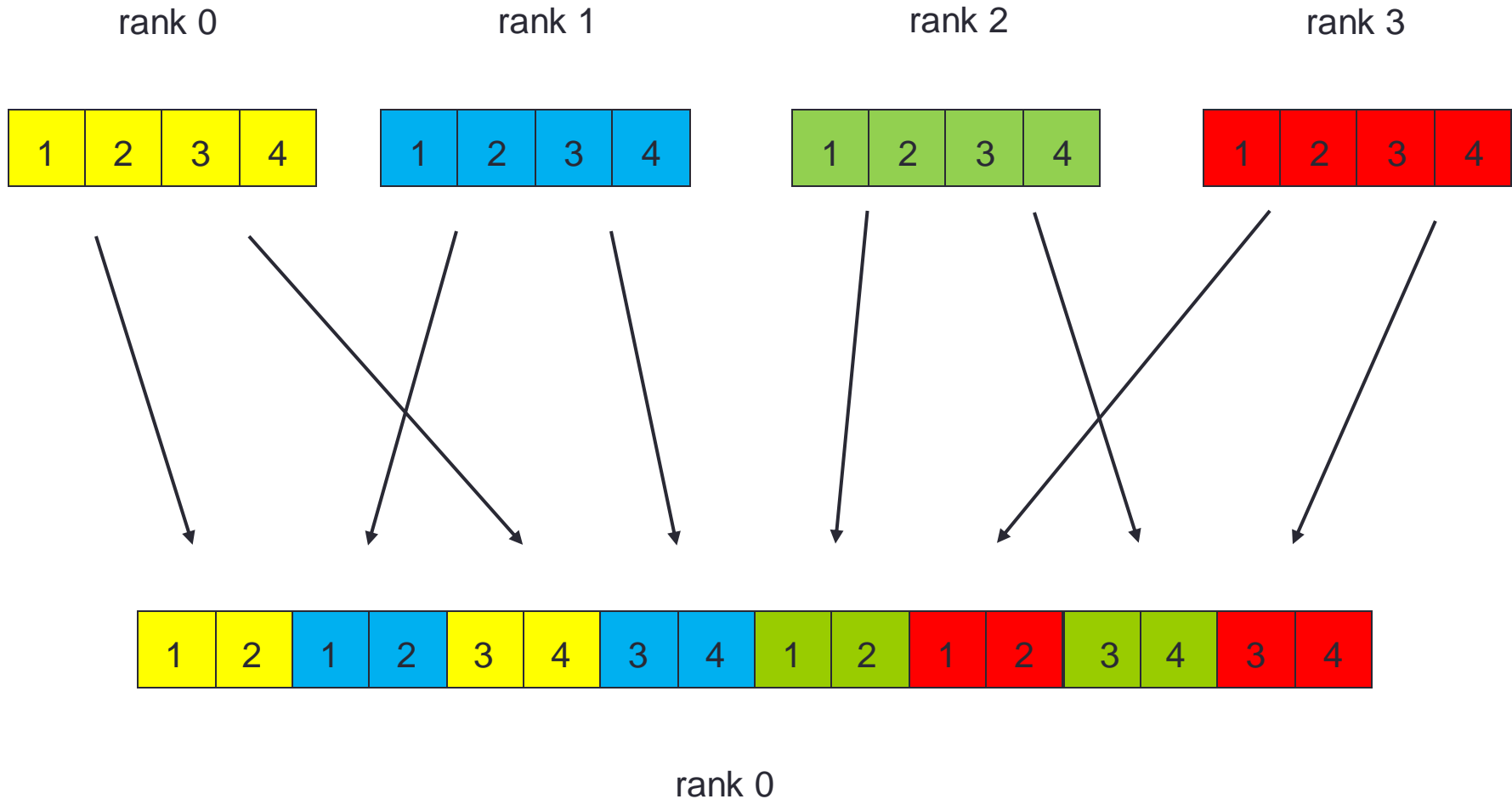
MPI_Gather (i)

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

MPI_GATHER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE,
           ROOT, COMM, IERROR)
```

- All processes in comm:

  - send **sendcount** items of type **sendtype** from **sendbuf** to rank **root**

- Root process only:

  - receive **recvcount** items of type **recvtype** separately from every process

  - these are received into **recvbuf** in rank order

  - ... but where exactly are they placed?

MPI_Gather (ii)

- Message from **rank** is received at (byte) displacement:
  - disp = **rank***recvcount*extent(recvtype)
  - straightforward for basic datatypes where recvtype = sendtype
    - in this case: sendtype = recvtype= MPI_INT, sendcount = recvcount = 4

1	2	3	4

rank **0**

1	2	3	4

rank **1**

1	2	3	4

rank **2**

1	2	3	4

rank **3**

1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

0*4*sizeof(int)   **1***4*sizeof(int)   **2***4*sizeof(int)   **3***4*sizeof(int)
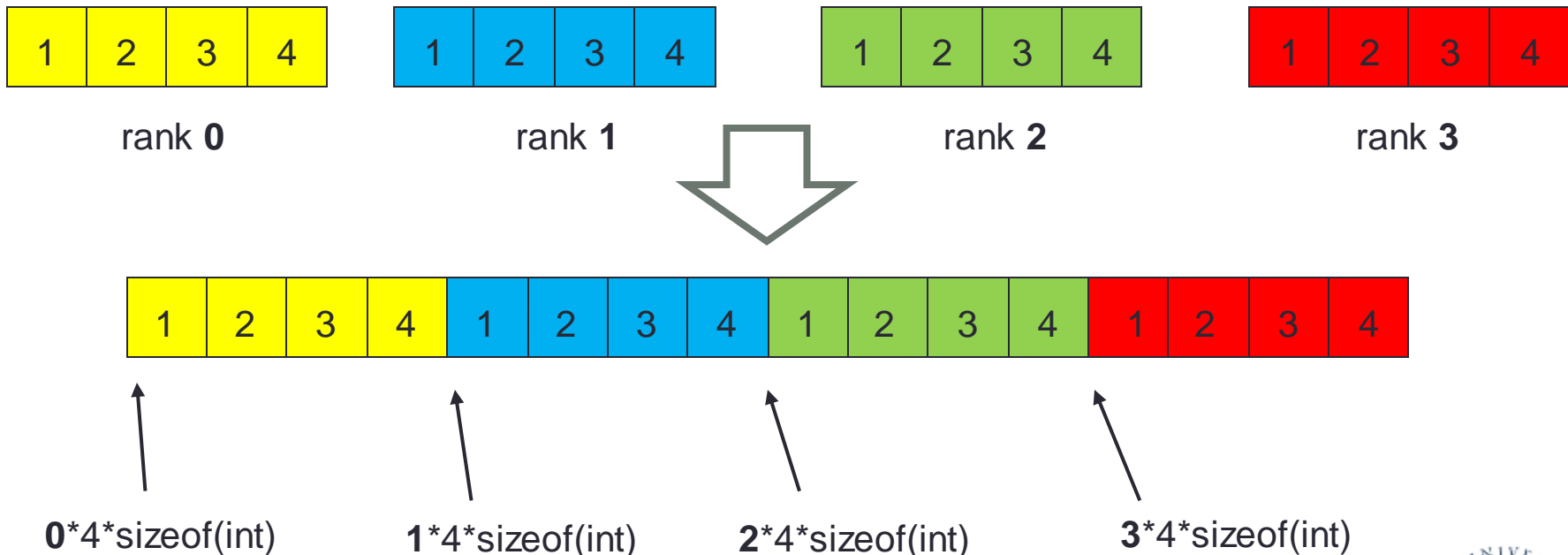
First problem

- Data pattern at receive side is incorrect
 - incoming messages needs to be scattered into receive buffer

- Solution
  - specify a vector (or subarray) for recvtype
  - pattern is a 2x2 subsection of a 4x4 array

- Now: sendcount, sendtype **not equal** to recvcount, recvtype
  - sendcount=4, sendtype=MPI_INT; recvcount=1, recvtype=vector2x2

- But they are **compatible** as they both contain 4 integers

Required pattern

rank 0

rank 1

rank 2

rank 3

0*sizeof(int)

2*sizeof(int)

8*sizeof(int)

10*sizeof(int)
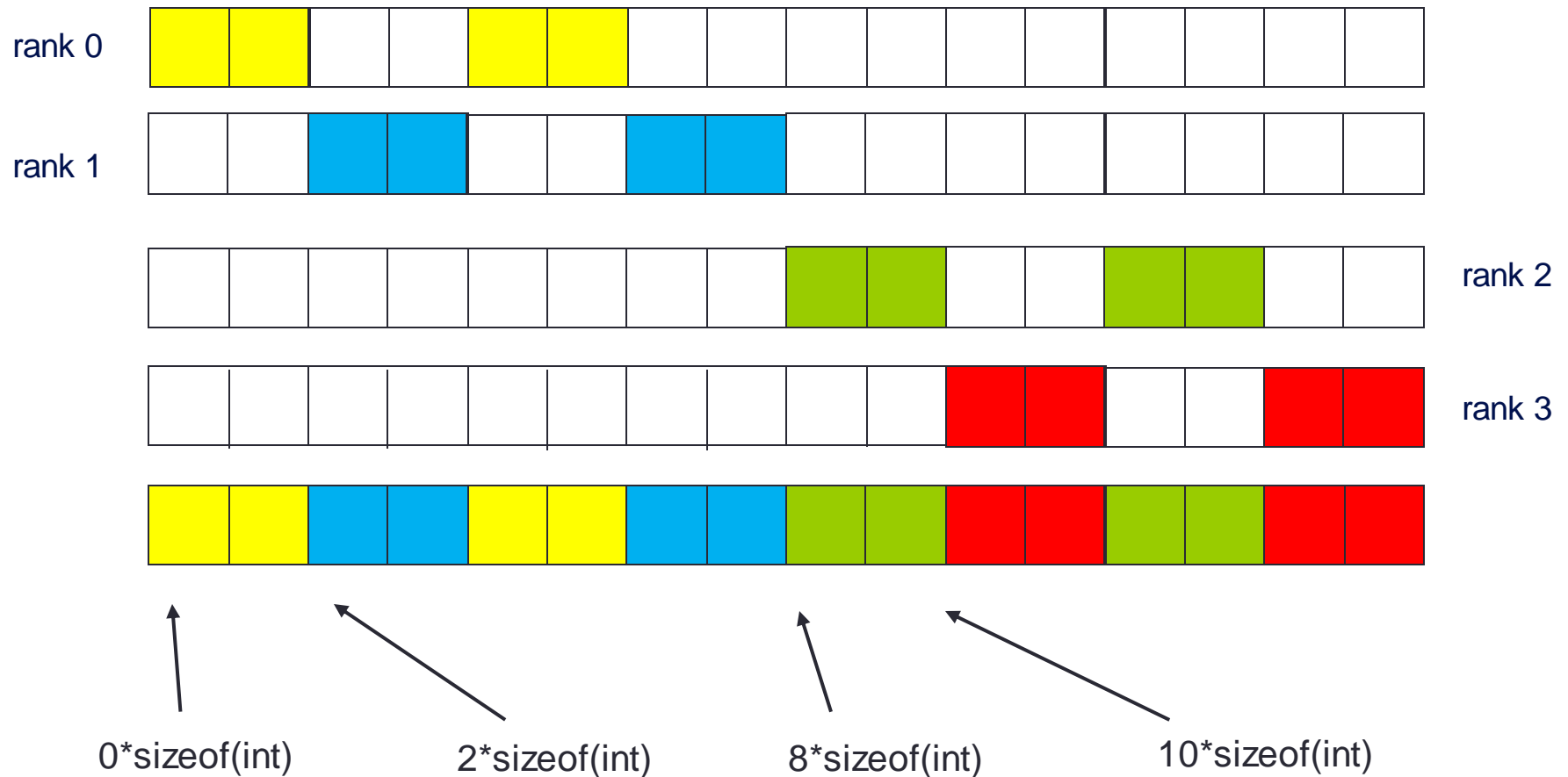
Second problem

- Displacements in receive buffer are not regular
  - counting in integers: 0, 2, 8 and 10

- Solution
  - MPI_Gatherv takes vectors of recvcounts and displacements
  - all are counted in terms of number of recvtypes
  - MPI_Gather assumes: recvcounts = 1, 1, 1, ...; displs = 0, 1, 2, 3, ...

- So what is the extent of the recvtype?
  - extent is distance from start of first to end of last element
  - MPI_Type_get_extent(vector2x2, ...) = 6 integers

Third problem

- Displacements in receive buffer are not multiples of extent
  - counting in integers, required displacements are: 0, 2, 8 and 10
  - extent of vector2x2= 6, so can only place at 0, 6, 12, 18, ...

- Solution
  - resize new datatype so it has a more useful extent, e.g. 1 integer

```
MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb,
MPI_Aint extent, MPI_Datatype *newtype)


MPI_TYPE_CREATE_RESIZED(OLDTYPE, LB, EXTENT, NEWTYPE, IERR)
INTEGER OLDTYPE, NEWTYPE, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) LB, EXTENT
```

Resizing a datatype

- "lower bound" specifies where datatype starts
  - e.g. create a leading gap (not needed here so lb=0)
  - lb and extent are 64-bit types: **`MPI_Aint`** or **`MPI_ADDRESS_KIND`**

```
MPI_Aint intlb, intsize, lb = 0;
MPI_Type_get_extent(MPI_INT, &intlb, &intsize);
MPI_Type_create_resized(vector2x2, lb, intsize, &vecresize);
MPI_Type_commit(&vecresize);
```

```
INTEGER(KIND=MPI_ADDRESS_KIND) :: INTLB, INTSIZE, LB=0
CALL MPI_TYPE_GET_EXTENT(MPI_INTEGER, INTLB, INTSIZE, IERR)
CALL MPI_TYPE_CREATE_RESIZED(VECTOR2x2, LB, INTSIZE,
VECRESIZE, IERR)
CALL MPI_TYPE_COMMIT(VECRESIZE, IERR)
```

MPI_Gatherv

- MPI_Gatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm)

  - sendcount = 4, sendtype = MPI_INT
  - recvcounts = [1,1,1,1], displs = [0, 2, 8, 10], recvtype = vecresize
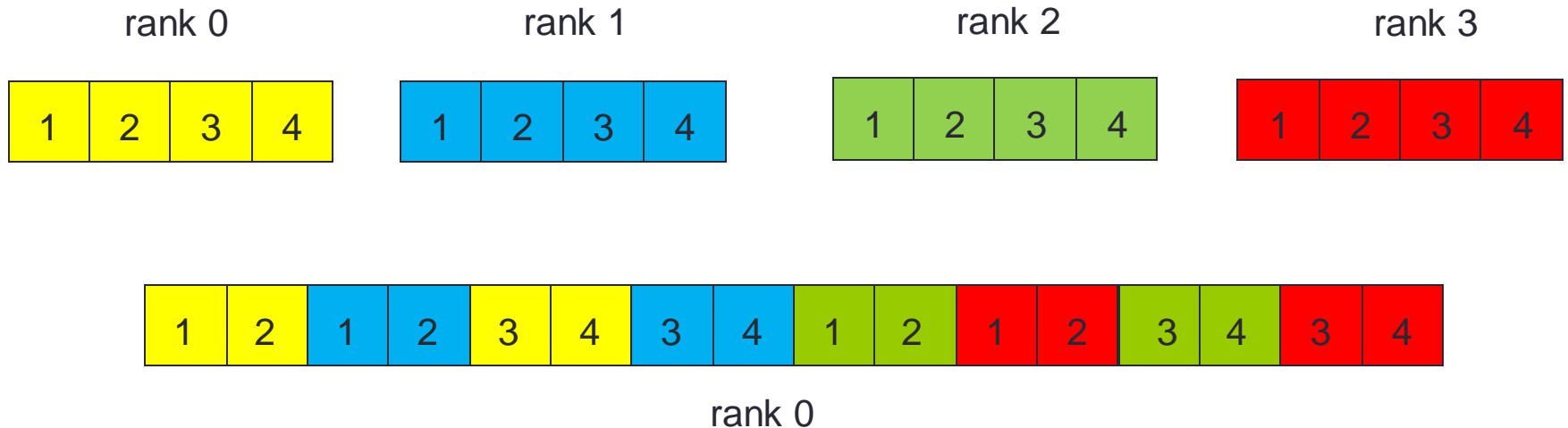
Other collectives

- Similar tricks can be used for scatter
  - MPI_Allgather / Allscatter also have "vector" versions

- Many scientific applications use Alltoall pattern
  - e.g. transposing a matrix between row and column decompositions
  - vector version, Alltoallv, plus derived types can ensure all data ends up directly in the correct place – avoids copy-in / copy-out
  - Alltoallv has single sendtype and recvtype, but vectors for sendcounts and sdispls as well as recvcounts and rdispls
    - all displacements in terms of extent(type) as for Gatherv
  - Even more general form MPI_Alltoallw exists
    - vectors for sendtypes and recvtypes as well as counts and disps
    - no obvious base unit for disps: Alltoallw uses **byte** displacements (yuk!)

Summary

- Technicalities of derived datatypes can be complicated
  - may have to play tricks with extents so collectives work as expected

- However, it is worth the effort!
  - MPI collectives are very highly optimised
  - naive DIY implementation will send P messages on P processes
  - optimised collectives should scale as $\log_2(P)$
  - 100 times faster on as few as 1000 processes!

- Derived types in collectives avoids ugly copy-in / copy out
  - rearrangement of data done automatically by MPI
  - MPI_Alltoall[v,w] used by many parallel scientific applications