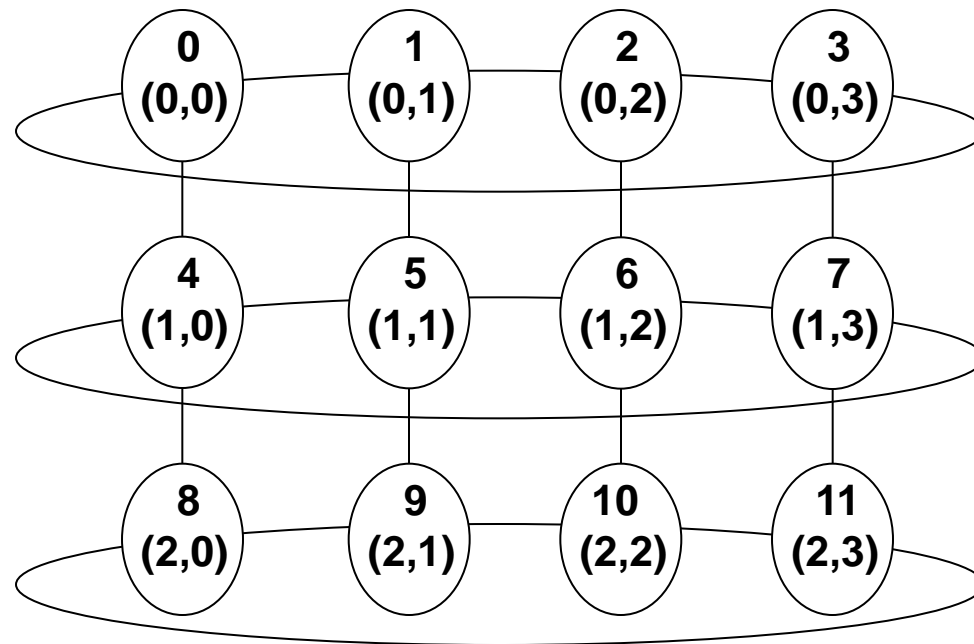


Virtual Topologies

- ▶ Convenient process naming.
- ▶ Naming scheme to fit the communication pattern.
- ▶ Simplifies writing of code.
- ▶ Can allow MPI to optimise communications.

- ▶ Creating a topology produces a new communicator.
- ▶ MPI provides ``mapping functions".
- ▶ Mapping functions compute processor ranks, based on the topology naming scheme.

A 2-dimensional Cylinder



▶ Cartesian topologies

- each process is “connected” to its neighbours in a virtual grid.
 - boundaries can be cyclic, or not.
 - optionally re-order ranks to allow MPI implementation to optimise for underlying network interconnectivity.
- processes are identified by cartesian coordinates.

▶ Graph topologies

- general graphs
- not covered here

▶ C:

```
int MPI_Cart_create(MPI_Comm comm_old,  
                   int ndims, int *dims, int *periods,  
                   int reorder, MPI_Comm *comm_cart)
```

▶ Fortran:

```
MPI_CART_CREATE(COMM_OLD, NDIMS, DIMS,  
               PERIODS, REORDER, COMM_CART, IERROR)
```

```
INTEGER COMM_OLD, NDIMS, DIMS(*), COMM_CART, IERROR  
LOGICAL PERIODS(*), REORDER
```

▶ C:

```
int MPI_Dims_create(int nnodes, int ndims,  
                   int *dims)
```

▶ Fortran:

```
MPI_DIMS_CREATE(NNODES, NDIMS, DIMS, IERROR)
```

```
INTEGER NNODES, NDIMS, DIMS(*), IERROR
```

- ▶ Call tries to set dimensions as close to each other as possible

dims before the call	function call	dims on return
(0, 0)	MPI_DIMS_CREATE(6, 2, dims)	(3, 2)
(0, 0)	MPI_DIMS_CREATE(7, 2, dims)	(7, 1)
(0, 3, 0)	MPI_DIMS_CREATE(6, 3, dims)	(2, 3, 1)
(0, 3, 0)	MPI_DIMS_CREATE(7, 3, dims)	erroneous call

- ▶ Non zero values in dims sets the number of processors required in that direction.
 - **WARNING:- make sure dims is set to 0 before the call!**

Mapping process grid coordinates to ranks

▶ C:

```
int MPI_Cart_rank(MPI_Comm comm,  
                  int *coords, int *rank)
```

▶ Fortran:

```
MPI_CART_RANK (COMM, COORDS, RANK, IERROR)
```

```
INTEGER COMM, COORDS (*), RANK, IERROR
```

Mapping ranks to process grid coordinates

▶ C:

```
int MPI_Cart_coords(MPI_Comm comm, int rank,  
                  int maxdims, int *coords)
```

▶ Fortran:

```
MPI_CART_COORDS(COMM, RANK, MAXDIMS, COORDS,  
                IERROR)
```

```
INTEGER COMM, RANK, MAXDIMS, COORDS(*), IERROR
```

Computing ranks of my neighbouring processes
Following conventions of MPI_SendRecv

▶ C:

```
int MPI_Cart_shift(MPI_Comm comm,  
                  int direction, int disp,  
                  int *rank_source, int *rank_dest)
```

▶ Fortran:

```
MPI_CART_SHIFT(COMM, DIRECTION, DISP,  
              RANK_SOURCE, RANK_DEST, IERROR)
```

```
INTEGER COMM, DIRECTION, DISP,  
        RANK_SOURCE, RANK_DEST, IERROR
```

- ▶ What if you ask for the rank of a non-existent process?
 - or look off the edge of a non-periodic grid?
- ▶ MPI returns a NULL processor
 - rank is `MPI_PROC_NULL`
- ▶ `MPI_PROC_NULL` is a black hole
 - sends and receives complete immediately
 - send buffer disappears, receive buffer isn't touched
 - like UNIX `/dev/null`

- ▶ Cut a grid up into “slices”.
- ▶ A new communicator is produced for each slice.
- ▶ Each slice can then perform its own collective communications.
- ▶ `MPI_Cart_sub` and `MPI_CART_SUB` generate new communicators for the slices.
 - Use array to specify which dimensions should be retained in the new communicator.

▶ C:

```
int MPI_Cart_sub (MPI_Comm comm,  
                 int *remain_dims,  
                 MPI_Comm *newcomm)
```

▶ Fortran:

```
MPI_CART_SUB (COMM, REMAIN_DIMS,  
              NEWCOMM, IERROR)
```

```
INTEGER COMM, NEWCOMM, IERROR  
LOGICAL REMAIN_DIMS (*)
```

- ▶ See Exercise 6 on the sheet
- ▶ Rewrite the exercise passing numbers round the ring using a one-dimensional ring topology.