Programming with MPI

Topologies

Nick Maclaren

nmm1@cam.ac.uk

September 2012

Topologies

Topologies are how the processes are connected
MPI's virtual topologies map the program structure

- Independent of the actual hardware network

Topologies are almost essential if:

- You are writing structure–generic libraries
- Your program has a variable graph structure

Both are seriously advanced use – be warned

In theory, can match the network for performance

- In practice, that almost never helps at all

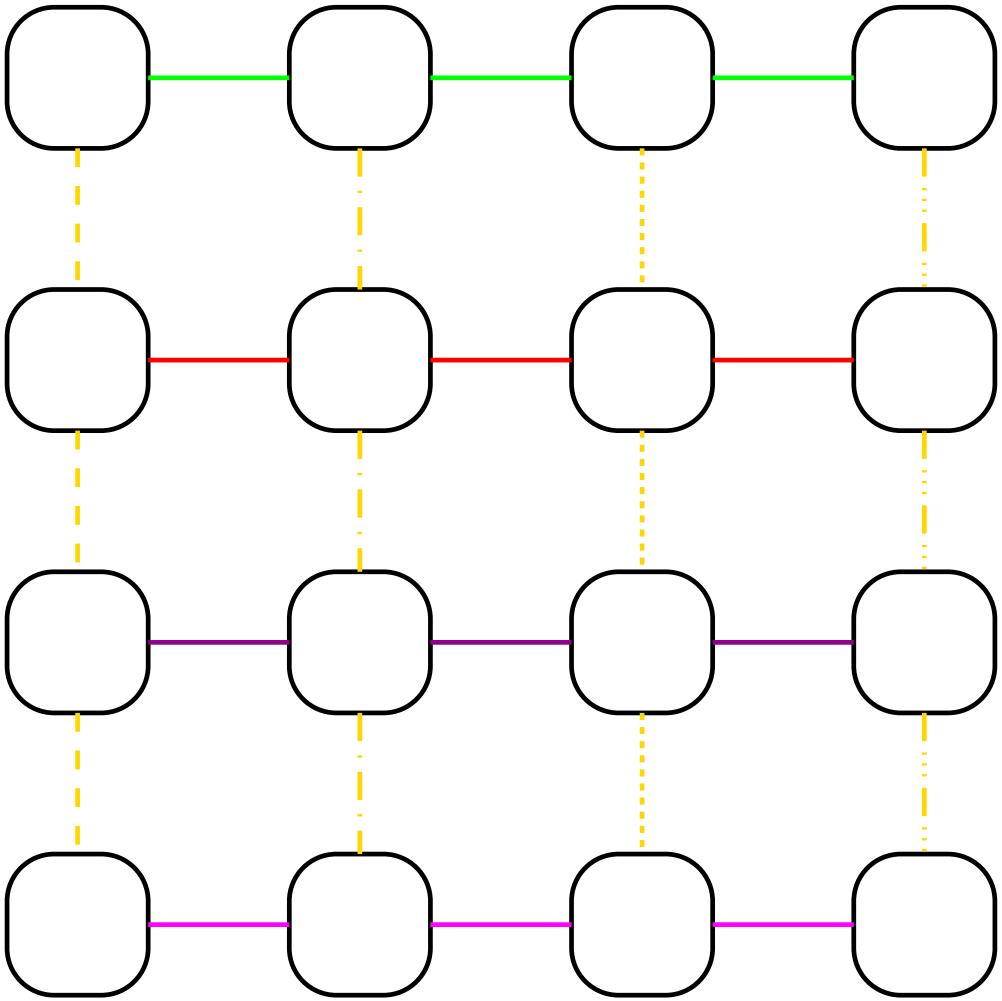
Cartesian Topologies (1)

Consider the simplest case of Cartesian topologies
These are a N–dimensional grid of MPI processes
A very common distribution in scientific code

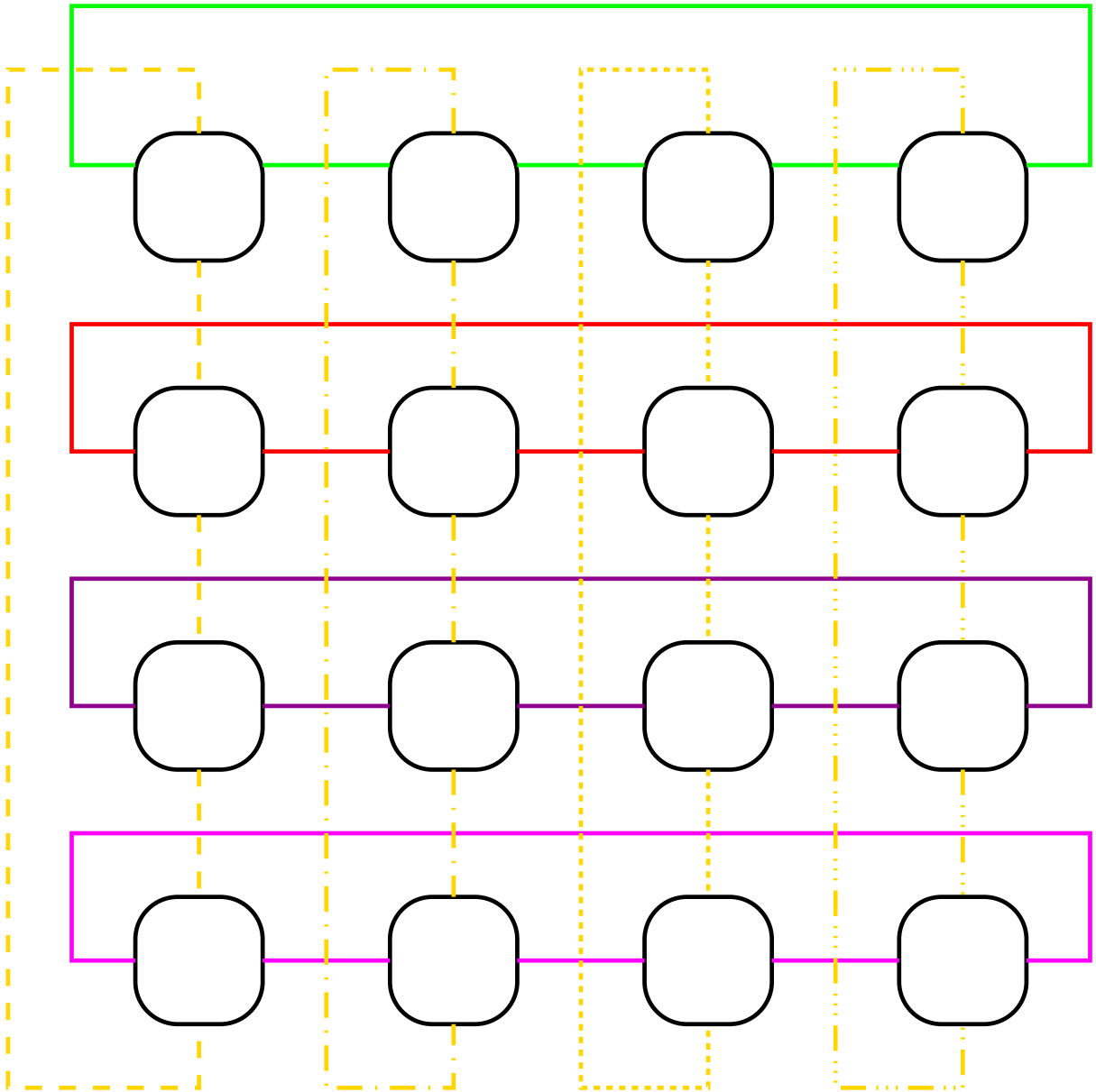
You can specify whether each dimension is periodic

No dimensions periodic is a N–dimensional grid
All dimensions periodic is a N–dimensional torus
Some dimensions periodic is a (hyper–)cylinder

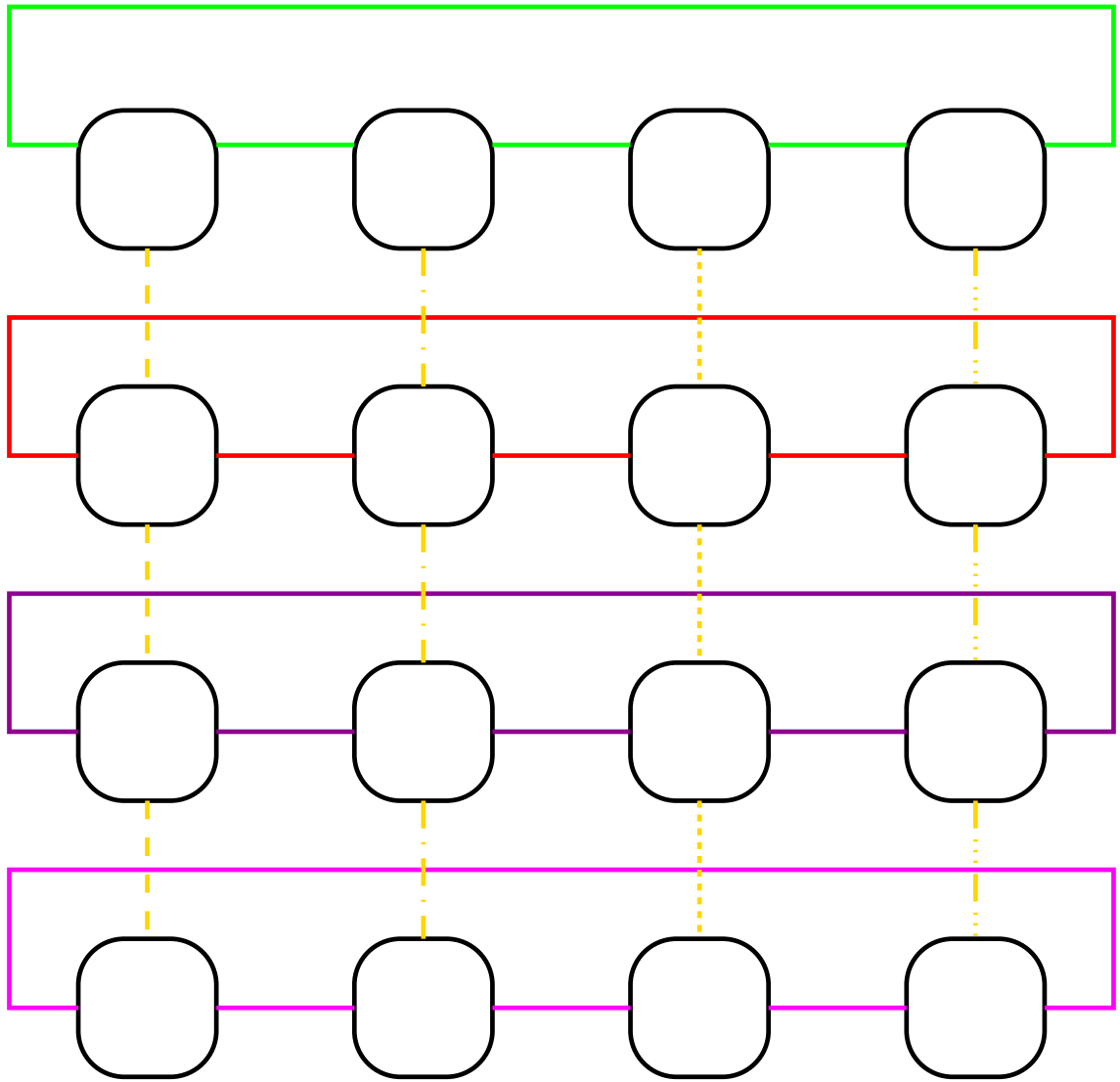
2–D Grid/Mesh



2–D Torus



Cylinder



Cartesian Topologies (2)

You use MPI_Cart_create exactly like MPI_Split
It creates a new communicator with a topology

You pass the dimensions of the grid, not the colour
You say whether the dimension is periodic or not

You say if the implementation may reorder processes
This might result in improved performance

```
MPI_Cart_create ( oldcomm , ndims ,
      dims , periodic , reorder , newcomm )
```

Cartesian Topologies (3)

ndims may be zero, but that's advanced use

Excess processes return MPI_COMM_NULL
Too few processes is an error – don't do it

Allow reordering, unless you have reason not to
Without reordering, the ranks stay the same
Grid points map to ranks in C array order

Fortran Example (1)

Let us assume that we have 8 processes

```
INTEGER :: newcomm , error
INTEGER , PARAMETER :: dims ( 2 ) = (/ 2 , 3 /)
LOGICAL, PARAMETER :: reorder = .TRUE. ,    &
      periodic ( 2 ) = .FALSE.

CALL MPI_Cart_create ( MPI_COMM_WORLD ,       &
      SIZE ( dims ) , dims , periodic , reorder ,       &
      newcomm , error )
```

In 2 processes, newcomm is MPI_COMM_NULL

The rest are in the new communicator, in some order

Fortran Example (2)

```
INTEGER :: newcomm , error
INTEGER , PARAMETER :: dims ( 2 ) = (/ 2 , 3 /)
LOGICAL, PARAMETER :: periodic ( 2 ) = .FALSE.

CALL MPI_Cart_create ( MPI_COMM_WORLD ,      &
     SIZE ( dims ) , dims , periodic , .FALSE. ,      &
     newcomm , error )
```

Rank	Grid	Rank	Grid
0	(0,0)	4	(1,1)
1	(0,1)	5	(1,2)
2	(0,2)	6	None
3	(1,0)	7	None

C Example

```
MPI_Comm newcomm ;
static const int dims [ ] = { 2 , 3 } ,
    periodic [ ] = { 0 , 0 } , reorder = 1 ;
int error ;
error = MPI_Cart_create ( MPI_COMM_WORLD ,
        2 , dims , periodic , reorder ,
        newcomm ) ;
```

In 2 processes, newcomm is MPI_COMM_NULL
The rest are in the new communicator, in some order

reorder = 0 does the same as in Fortran

Creating a Grid (1)

Store any fixed dimensions in an integer vector
And set all of the other elements to zero

Pass the number of nodes; it fills in the vector
Tries to make the dimensions about the same sizes

It is an error if there is no exact decomposition
But results like (257,1,1,1,1,1,1,1,1,1) are not

Use this if it helps – ignore it if it doesn't

Creating a Grid (2)

Fortran example:

```
INTEGER :: comm , nprocs , dims ( 3 ) , error
CALL MPI_Comm_size (        &
      MPI_COMM_WORLD , nprocs , error )
CALL MPI_Dims_create ( nprocs , 3 , dims , error )
```

C example:

```
MPI_Comm comm ;
int nprocs , dims [ 3 ] , error ;
error = MPI_Comm_size ( MPI_COMM_WORLD ,
      nprocs ) ;
error = MPI_Dims_create ( nprocs , 3 , dims ) ;
```

Finding Coordinates

MPI_Cart_coords converts a rank to coordinates

Fortran example:

```
INTEGER :: comm , rank , coords ( 3 ) , error
CALL MPI_Cart_coords ( comm , rank , 3 ,       &
     coords , error )
```

C example:

```
MPI_Comm comm ;
int nprocs , coords [ 3 ] , rank , error ;
error = MPI_Cart_rank ( comm , rank ,
     3 , coords , error ) ;
```

Finding Ranks

MPI_Cart_rank converts coordinates to a rank

You do not specify the number of dimensions

Fortran example:

```
INTEGER :: comm , coords ( 3 ) , rank , error
CALL MPI_Cart_rank ( comm , coords , rank , error )
```

C example:

```
MPI_Comm comm ;
int nprocs , coords [ 3 ] , rank , error ;
error = MPI_Cart_rank ( comm , coords ,
        & rank , error ) ;
```

Nearby Processes

The rank that is ±N along a dimension
The function is MPI_Cart_shift if you want it

When they go over the limit, periodic ones wrap
and others return MPI_PROC_NULL

Does not enable you to go up a diagonal
You have to use MPI_Cart_rank for that

Not covered further for that reason, but it works
Most people will probably use only MPI_Cart_rank

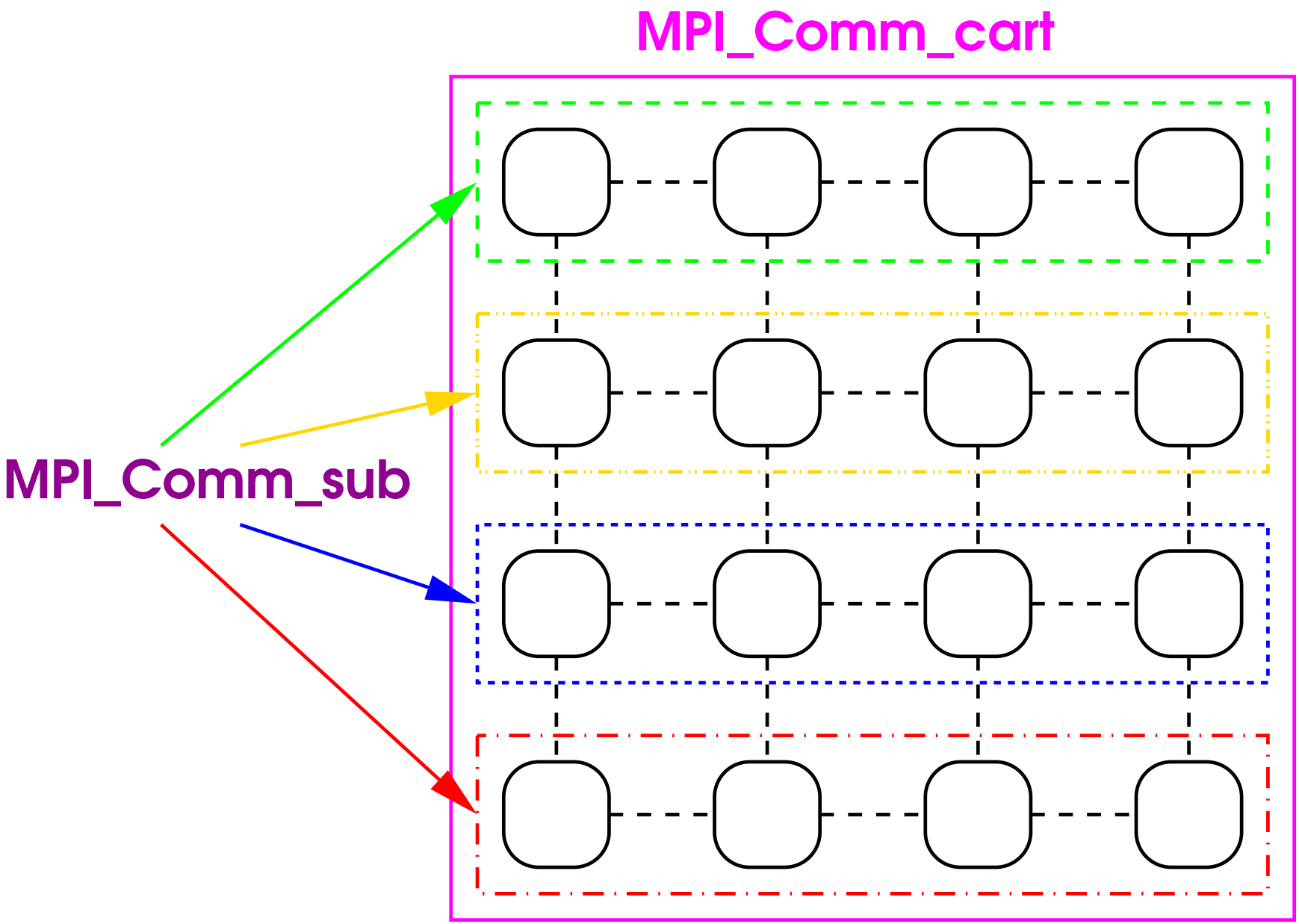
Subspaces

You very often want to work with subspaces of grids
E.g. using rows/columns of a matrix
You can create derived communicators to do just that

Like MPI_Comm_split, it returns multiple ones
E.g. each row will be in its own communicator

MPI_Cart_sub takes a Boolean array argument
You specify the dimensions you want included
The others create separate communicators

Subsetting by Rows



Fortran Example

If MPI_Cart_create returned a 3–D grid in comm

```
INTEGER :: comm , newcomm , error
LOGICAL, PARAMETER :: keep ( 3 ) =       &
      (/ .TRUE. , .FALSE. , .TRUE. /)

CALL MPI_Cart_sub ( comm , keep , newcomm , error )
```

Each plane in dims 1 and 3 will be a communicator
Each index in dimension 2 will return a separate one

C Example

If MPI_Cart_create returned a 3–D grid in comm

```
MPI_Comm comm , newcomm ;
int error ;
static const int keep [ 3 ] = { 1 , 0 , 1 } ;

error = MPI_Cart_sub ( comm , keep , newcomm ) ;
```

Each plane in dims 1 and 3 will be a communicator
Each index in dimension 2 will return a separate one

Query Functions

- It is a bad idea to fix assumptions in library code

Libraries should query environment and use that data
Library code should be as generic as is reasonable

- Obviously, return an error if they can't handle it

- These are also very useful for debugging

Answer "Is the code in the state it should be?"
Most people will use them only for that, but that's fine

Checking the Topology (1)

You can check if a communicator has a topology

The returned value is an integer, which is one of:

MPI_UNDEFINED (none),
MPI_CART (Cartesian),
MPI_GRAPH or MPI_DIST_GRAPH

Fortran example:

```
INTEGER :: comm , result , error
CALL MPI_Topo_test ( comm , result , error )
```

Checking the Topology (2)

C example:

```
MPI_Comm comm ;
int result , error ;
error = MPI_Topo_test ( comm , & result ) ;
```

Number of Dimensions

You can get the number of number of dimensions
The communicator must be Cartesian

Fortran example:

```
INTEGER :: comm , ndims , error
CALL MPI_Cartdim_get ( comm , ndims , error )
```

C example:

```
MPI_Comm comm ;
int ndims , error ;
error = MPI_Cartdim_get  ( comm , & ndims ) ;
```

Cartesian Information (1)

You can get all of the other information, too
One call gets the dimensions, periodicities and
grid coordinates of calling process
The communicator must be Cartesian

Fortran example:

```
INTEGER :: comm , dims (3 ) , coords ( 3 ) , error
LOGICAL :: periodic ( 3 )
CALL MPI_Cart_get ( comm , 3 , dims ,       &
     periodic , coords , error )
```

Cartesian Information (2)

C example:

```
MPI_Comm comm ;
int dims [ 3 ] , periodic [ 3 ] ,  coords [ 3 ] , error ;
error = MPI_Cart_get  ( comm , 3 , dims ,
        periodic , coords ) ;
```

Graph Topologies

Some programs have a natural graph topology
This is always messy to code, which causes errors

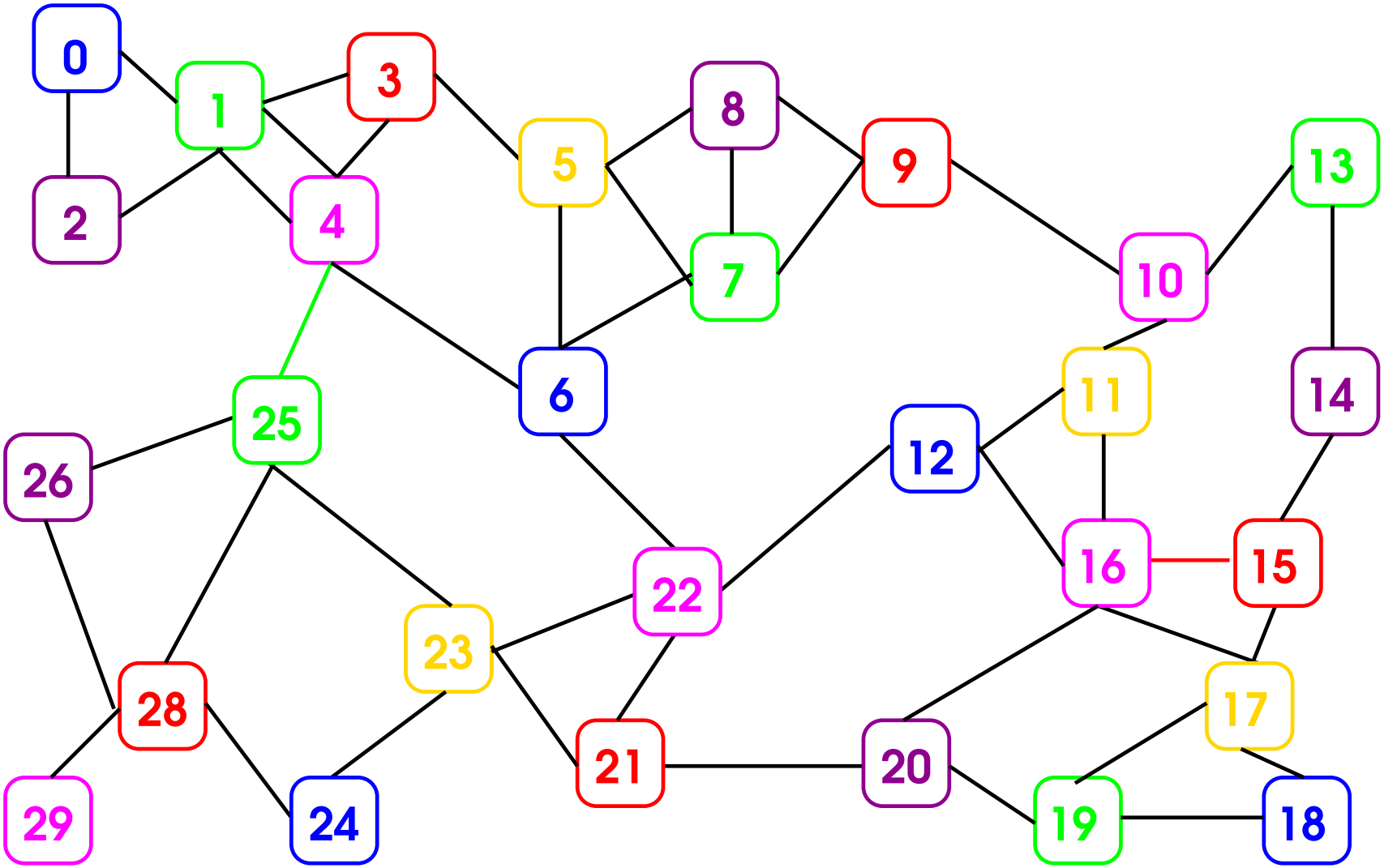
- Design your program very carefully

Fixed graph topologies are relatively straightforward
But are inflexible and very often poor design
Modifying them is very messy and error–prone

- But variable ones are seriously advanced use

MPI topologies may help to simplify your code
MPI 3.0 will add a lot of useful collectives
They are not covered further in this course

Graph Structure



Epilogue

That is essentially all about Cartesian topologies

A few simple exercises to play with them