Research Computing
UNIVERSITY OF COLORADO

Topologies and Communicators

Outline

Motivation

- Fox’s algorithm
- Cannon’s algorithm

Topologies

- Structure imposed on the processes in a communicator that allows the processes to be addressed in different ways

Communicators

- Collection of processes that can send messages to each other

Topologies

Topologies

Simplifies code

Can allow MPI to optimize communication

Creating a new topology provides:

- A new communicator
- Mapping functions: rank -> topology -> rank

Types

- Cartesian: processes can be identified by coordinates
- Graph

Creating a Cartesian Toplogy

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

```
comm_old: existing communicator

ndims: number of dimensions

*dims: the actual dimension

*periods: logical array indicating whether a dimension is cyclic

reorder: logical, false = preserve rank order

comm_cart: new communicator
```

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

```
Creates a division of processors in a cartesian grid
```

Example

```c
const int dim = 2;
int grid[dim] = {0,0};

// Assign the grid dimensions
MPI_Dims_create(size, dim, grid);

// The new communicator
MPI_Comm comm_grid;

// Allow cyclic behavior
int preiodic[dim] = {TRUE,TRUE};

// Create the communicator
MPI_Cart_create(comm, dim, grid, preiodic, TRUE, &comm_grid);
```

Mapping functions

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

```
Determines process coords in cartesian topology given rank in group
```

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

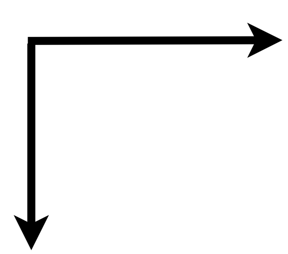
```
Determines process rank in communicator given Cartesian location
```

```
int MPI_Cart_shift(MPI_Comm comm, int direction, int displ,
                   int *source, int *dest);
```

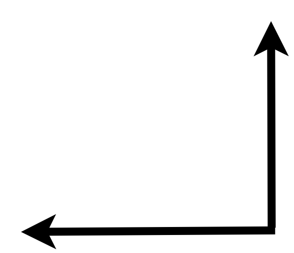
Returns source and destination ranks, given a shift direction and amount

direction = 0, direction=1, ... direction=dim-1

displ > 0



displ < 0



Example

```
// What is the rank grid coordinate?
int grid_coord[dim] = {0,0};
MPI_Cart_coords(comm_grid, rank, dim, grid_coord);

// Given a coordinate, what is the rank?
int rank_id;
MPI_Cart_rank(comm_grid, grid_coord, &rank_id);

// Who is my neighbor to the right?
int neighbor;
MPI_Cart_shift(comm_grid, 1, 1, &rank, &neighbor);

int neighbor_coord[dim] = {0,0};
MPI_Cart_coords(comm_grid, neighbor, dim, neighbor_coord);

// print rank_id grid_coord
// neighbor, neighbor_coord
```

test

```
bash-mac> mpirun -np 9 ./example communication > output
bash-mac> sort output
        0        (0,0)        1        (0,1)
        1        (0,1)        2        (0,2)
        2        (0,2)        0        (0,0)
        3        (1,0)        4        (1,1)
        4        (1,1)        5        (1,2)
        5        (1,2)        3        (1,0)
        6        (2,0)        7        (2,1)
        7        (2,1)        8        (2,2)
        8        (2,2)        6        (2,0)
bash-mac>
```

Neighbor above?

```c
// Who is my neighbor to the right?
int neighbor;
MPI_Cart_shift(comm_grid, 1, 1, &rank, &neighbor);

int neighbor_coord[dim] = {0,0};
MPI_Cart_coords(comm_grid, neighbor, dim, neighbor_coord);
```

```c
// Who is my neighbor above?
int neighbor;
MPI_Cart_shift(comm_grid, 0, -1, &rank, &neighbor);

int neighbor_coord[dim] = {0,0};
MPI_Cart_coords(comm_grid, neighbor, dim, neighbor_coord);
```

test

```
bash-mac> mpirun -np 9 ./example communication > output
bash-mac> sort output
        0          (0,0)        6           (2,0)
        1          (0,1)        7           (2,1)
        2          (0,2)        8           (2,2)
        3          (1,0)        0           (0,0)
        4          (1,1)        1           (0,1)
        5          (1,2)        2           (0,2)
        6          (2,0)        3           (1,0)
        7          (2,1)        4           (1,1)
        8          (2,2)        5           (1,2)
bash-mac>
```

Communicators

Communicators

Group of processes that can send and receive messages

Perform collective communications

Creation:

- Create groups of processors -> then create the MPI_Comm
- MPI_Comm_split: split an existing communicator
- MPI_Cart_sub: create from a topology

When you are done using them...

- MPI_Comm_free

MPI_Comm_split

```
int MPI_Comm_split(MPI_Comm comm, int split_key, int key,
                   MPI_Comm *newcomm);
```

```
split_key: processes with the same split_key will be in the same
communicator

key: control of rank assignment
```

rank_id	grid_coord[0]	grid_coord[1]
0	0	0
1	0	1
2	1	0
3	1	1

0	1
2	3

```
MPI_Comm comm_row;
MPI_Comm_split(comm_grid,                                  &comm_row);
MPI_Comm_split(comm_grid, grid_coord[0], rank_id, &comm_row);
```

MPI_Cart_sub

```c
int MPI_Cart_sub(MPI_Comm comm, int *remain_dims,
                 MPI_Comm *comm_new);
```

Partitions a communicator into subgroups which form lower-dimensional cartesian subgrids

remain_dims: ith dimension is kept in the subgrid (true) or is dropped (false)

```c
// Col communicator
MPI_Comm comm_col;

// What goes here? (true, false)
int free_coords[dim] =

// Creat the communicator
MPI_Cart_sub(comm_grid, free_coords, &comm_col);
```

Homework 6

Tasks

File I/0

- No parallel IO, you can use existing code
- Partial read and distribute
- For best performance, you will need to “chunk”

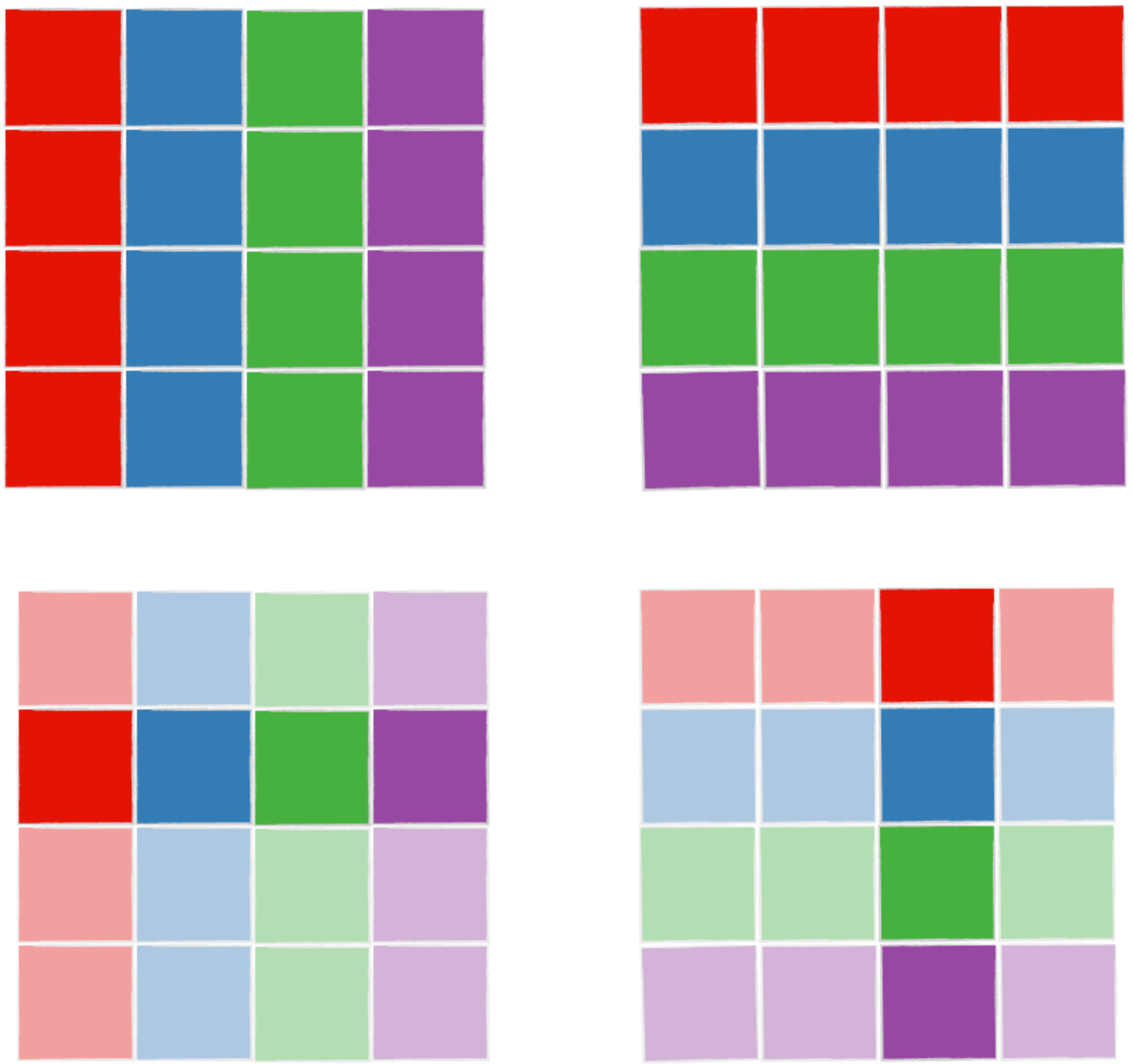
Matrix Multiply

- General Comments
- Cannon’s Algorithm
- Fox’s Algorithm

Advice

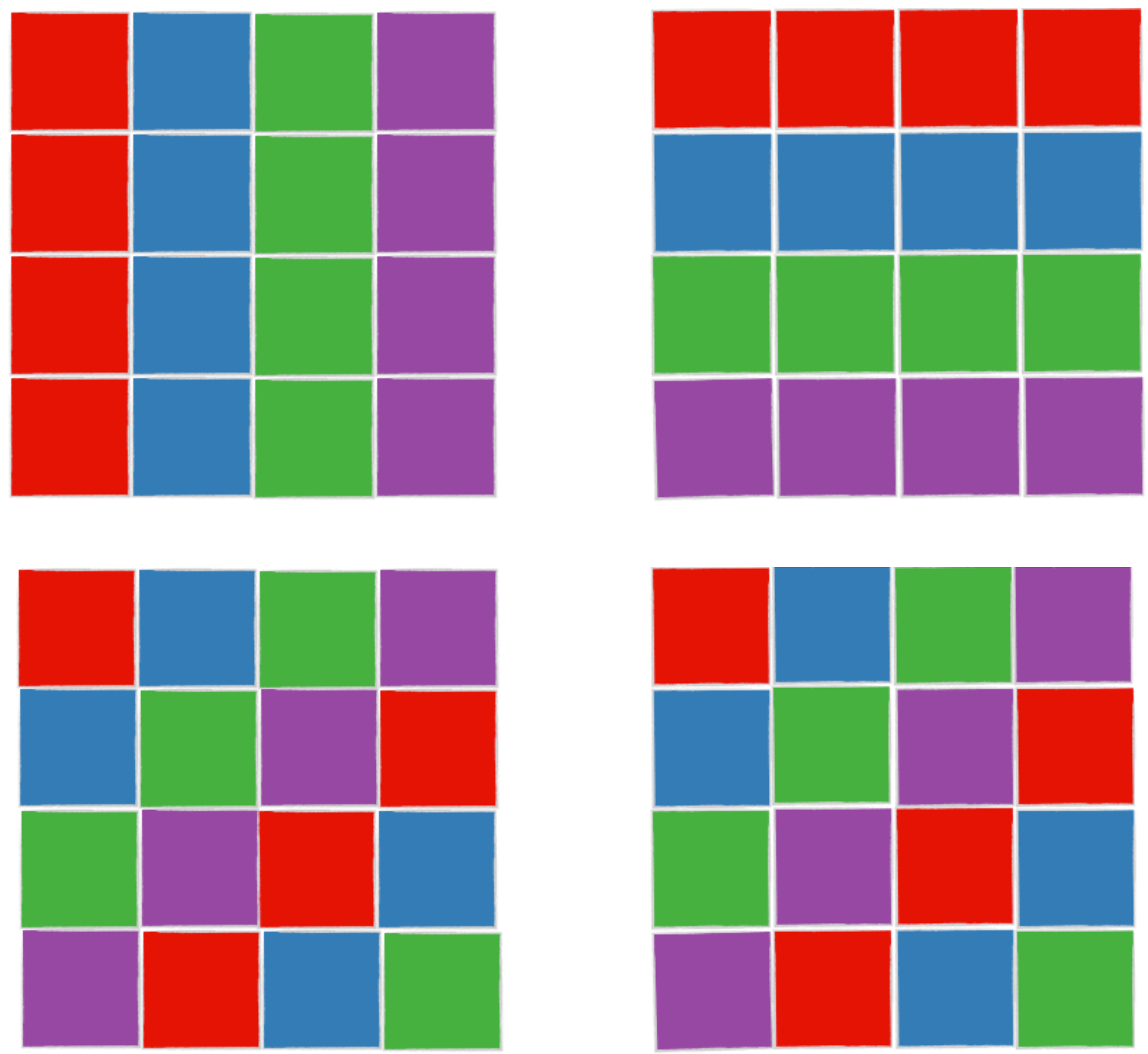
Example

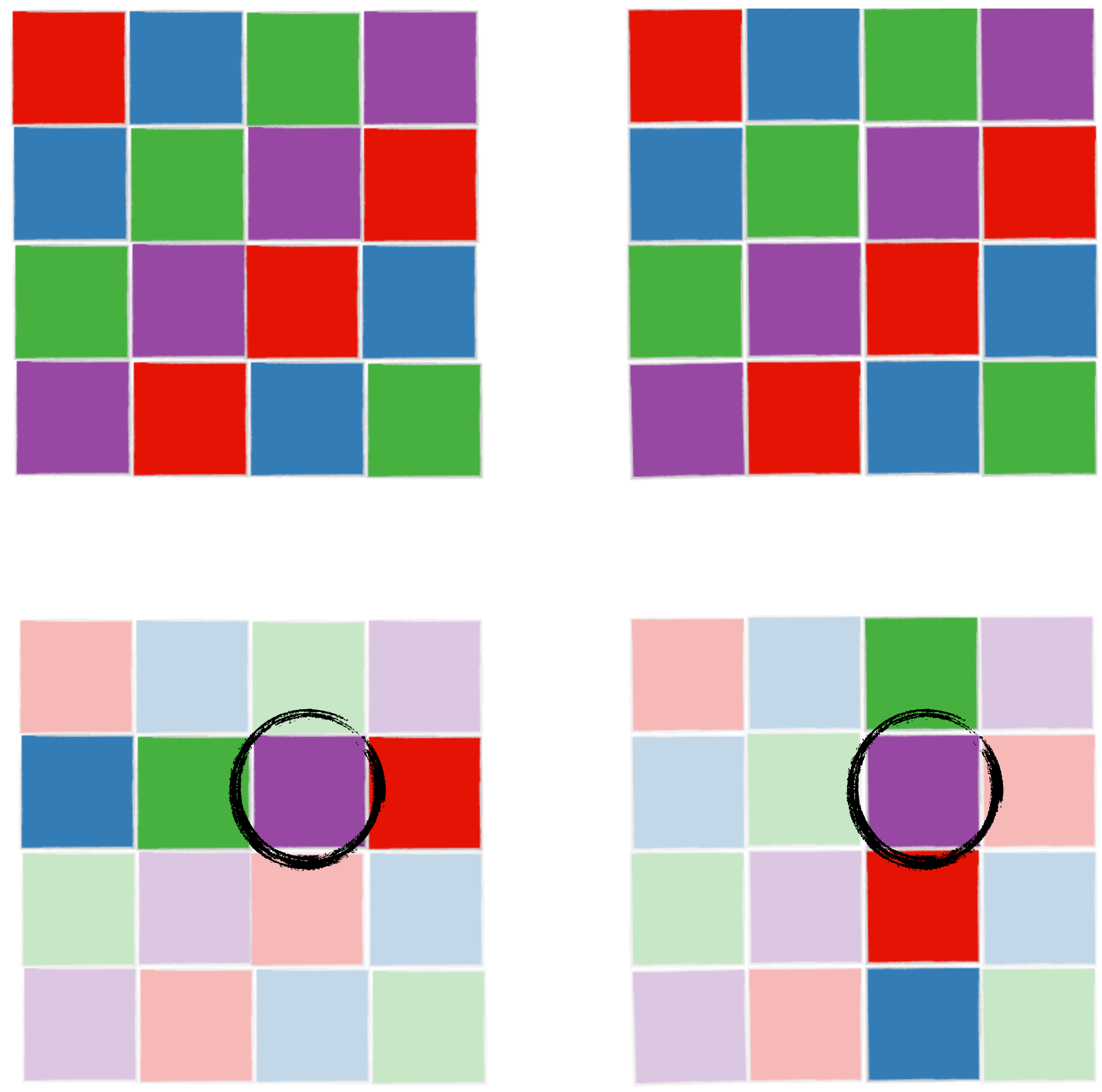
Only blocks of the same color can be multiplied

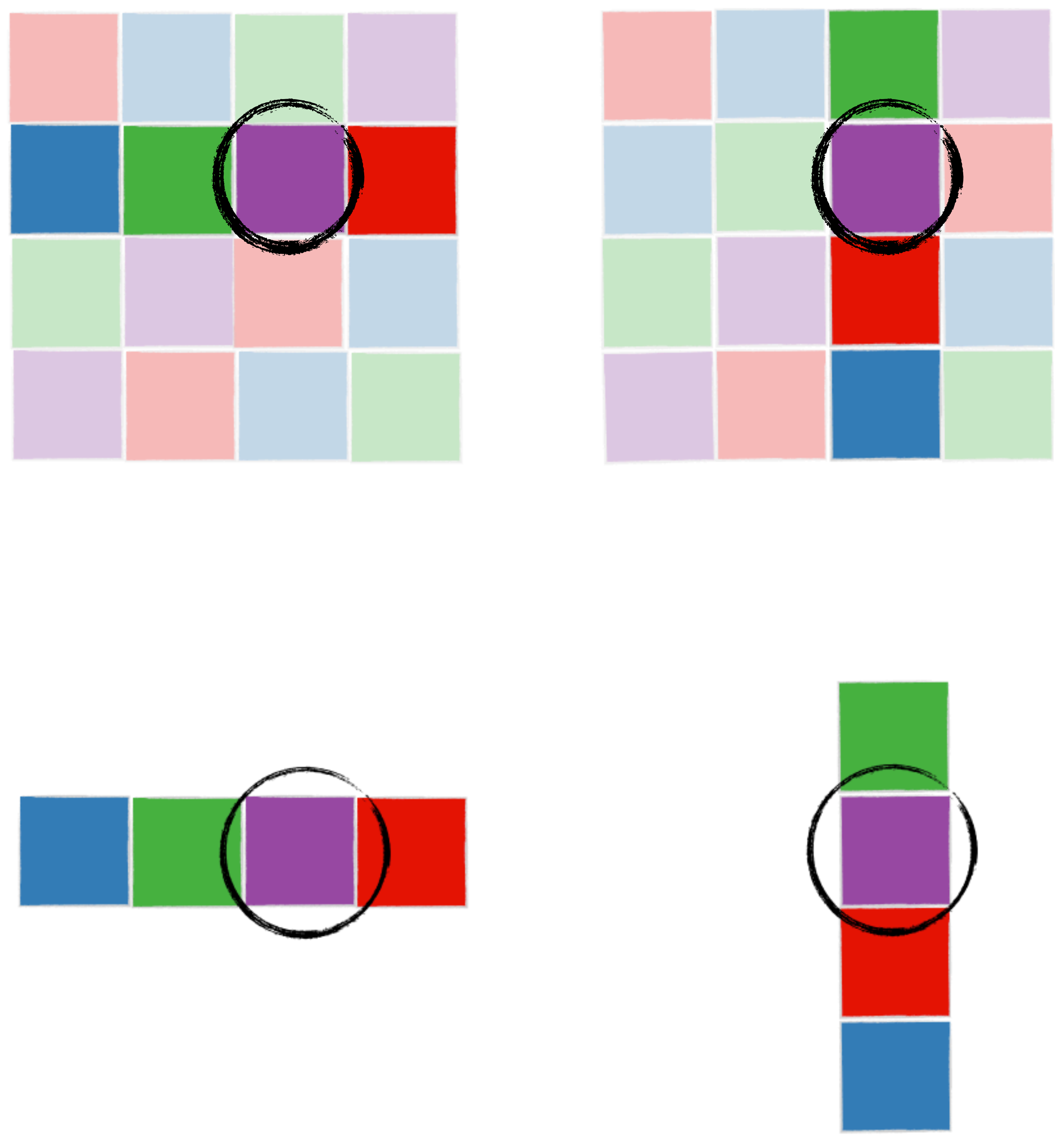


Cannon's Algorithm

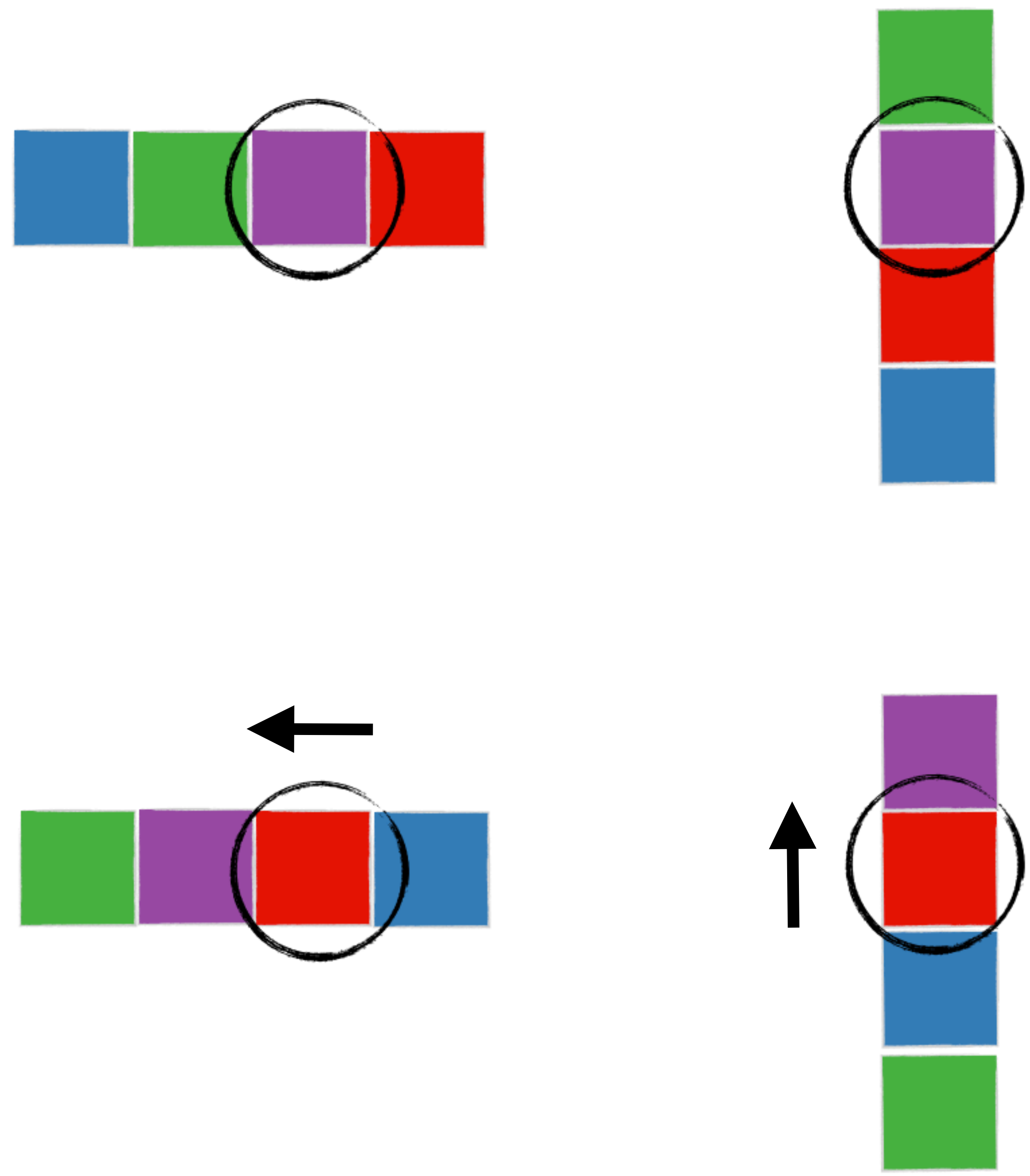
Rearrange the read and distribute $$p(i, j) \rightarrow A(i, k)B(k, j)$$



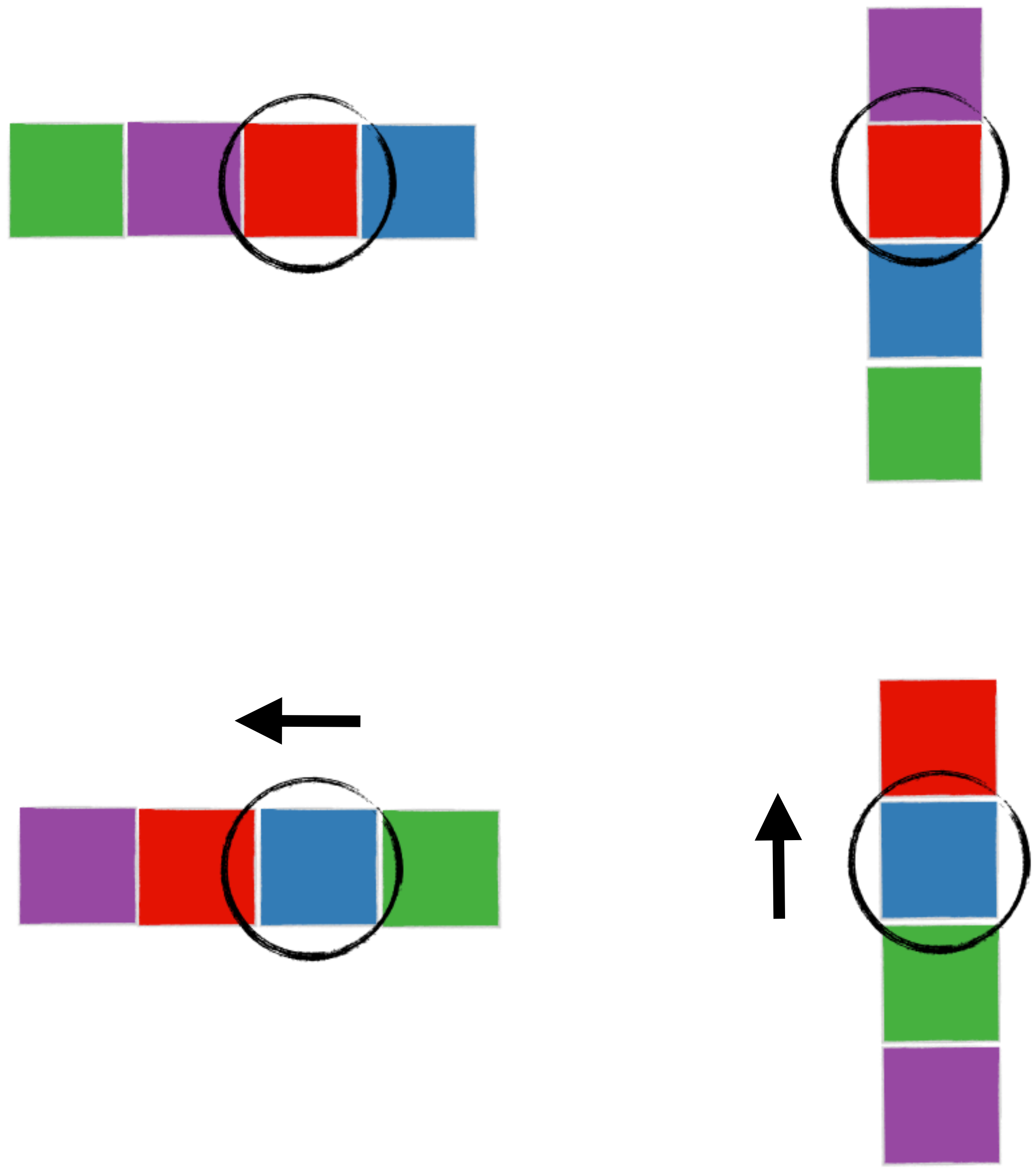




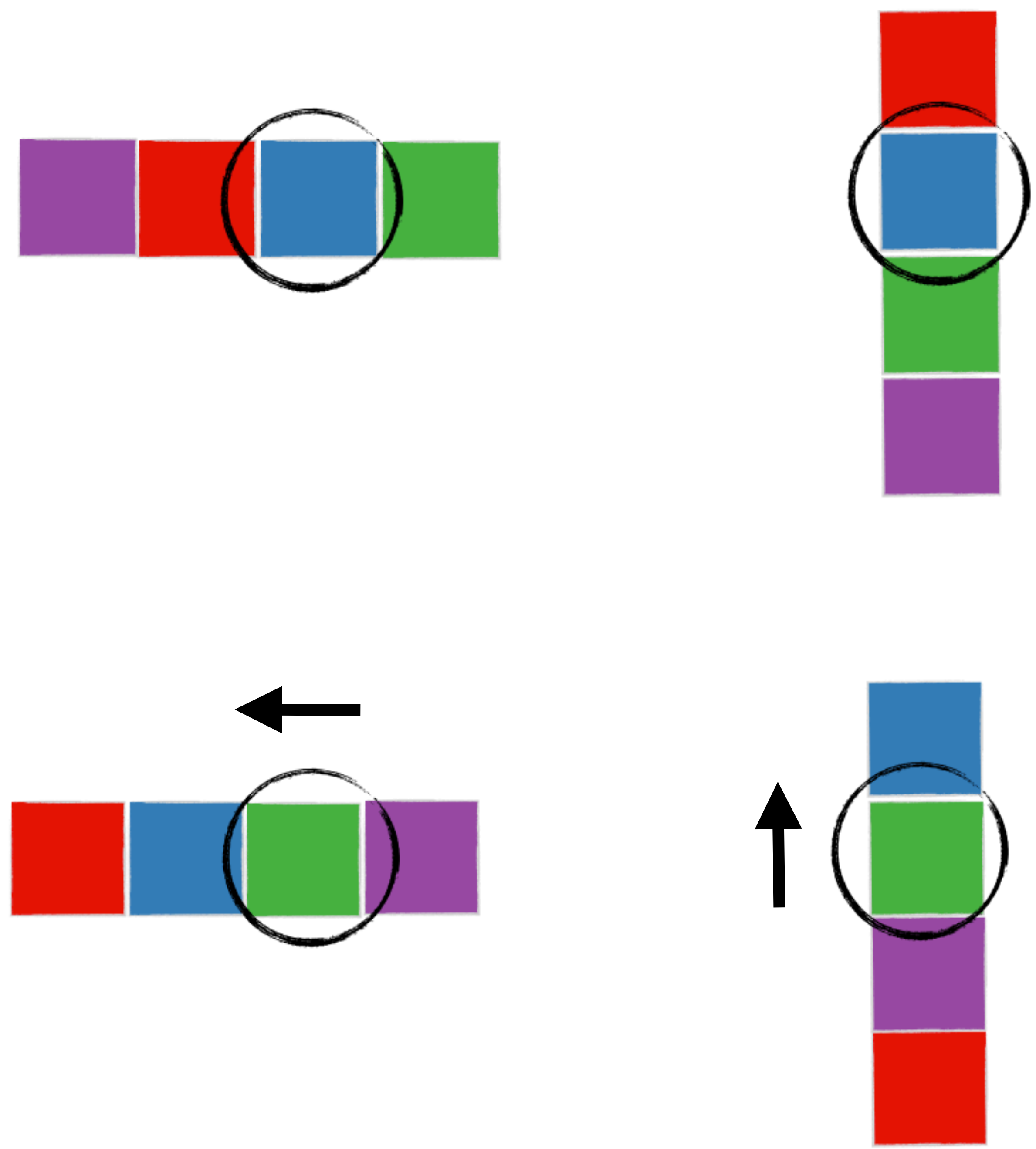
Step 1



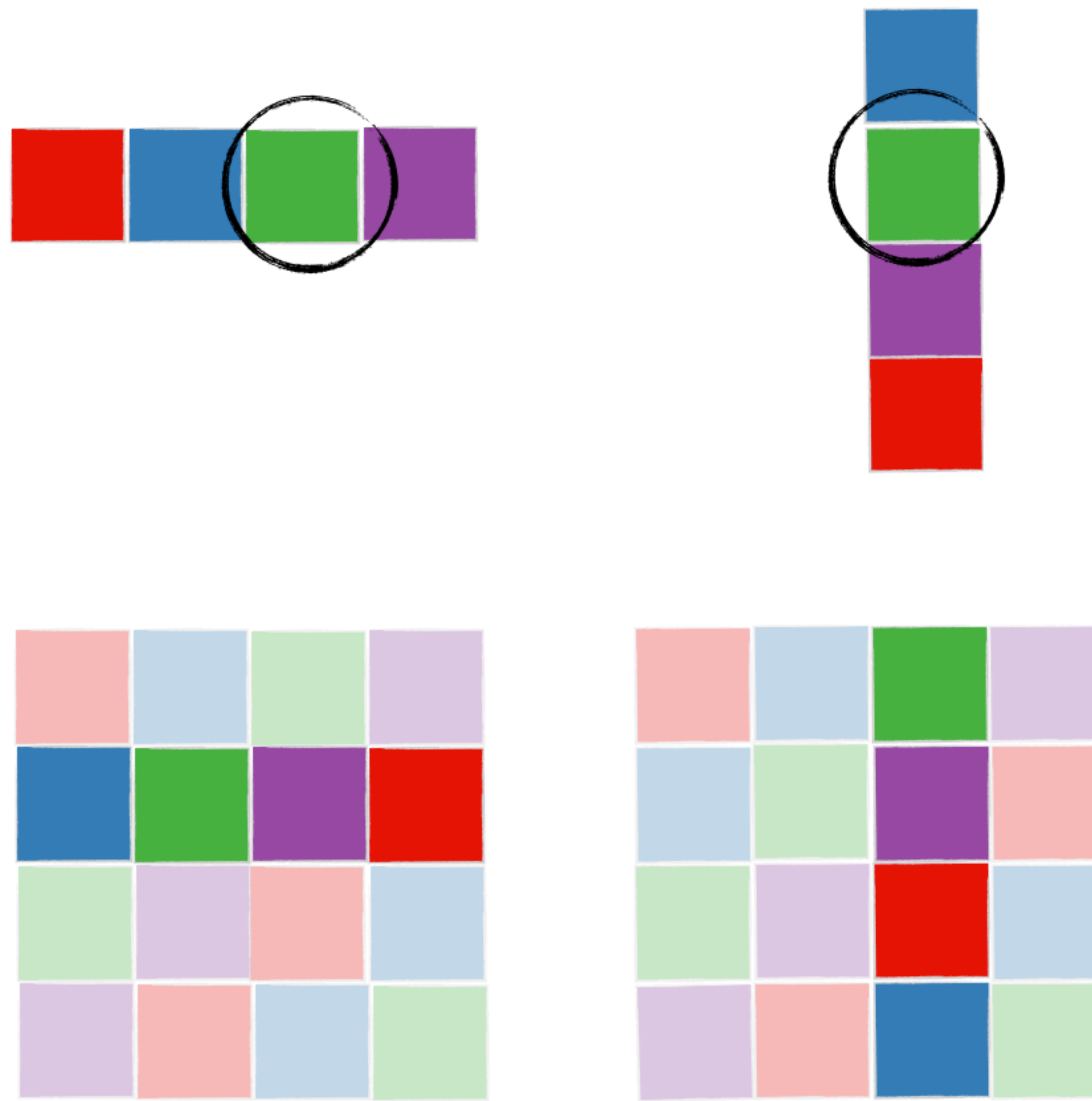
Step 2



Step 3



Step 4



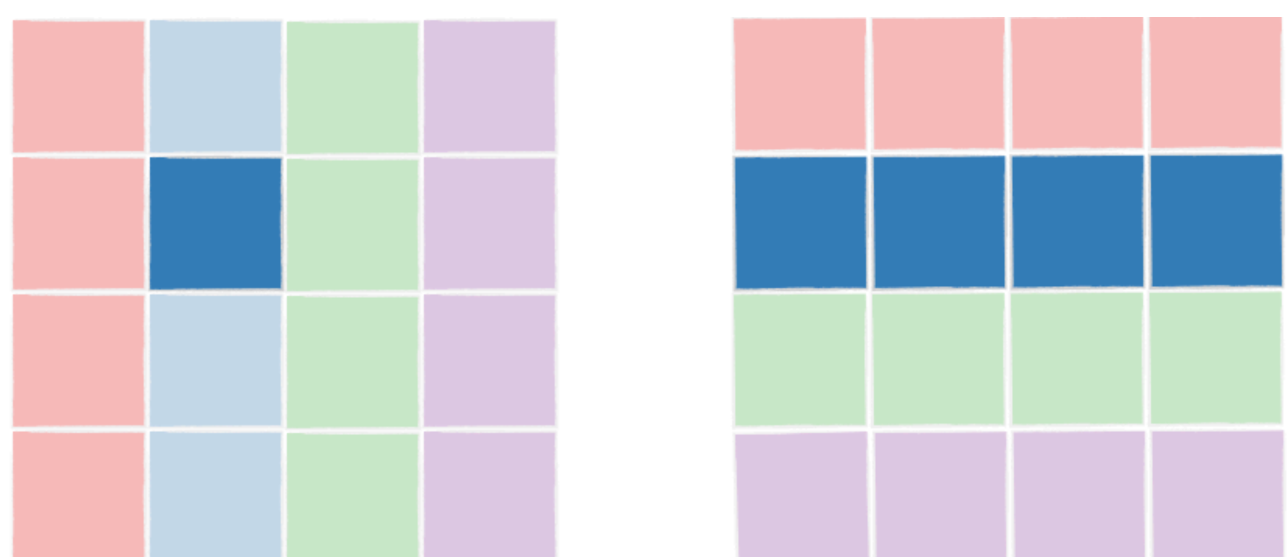
Fox’s Algorithm

Read in normally

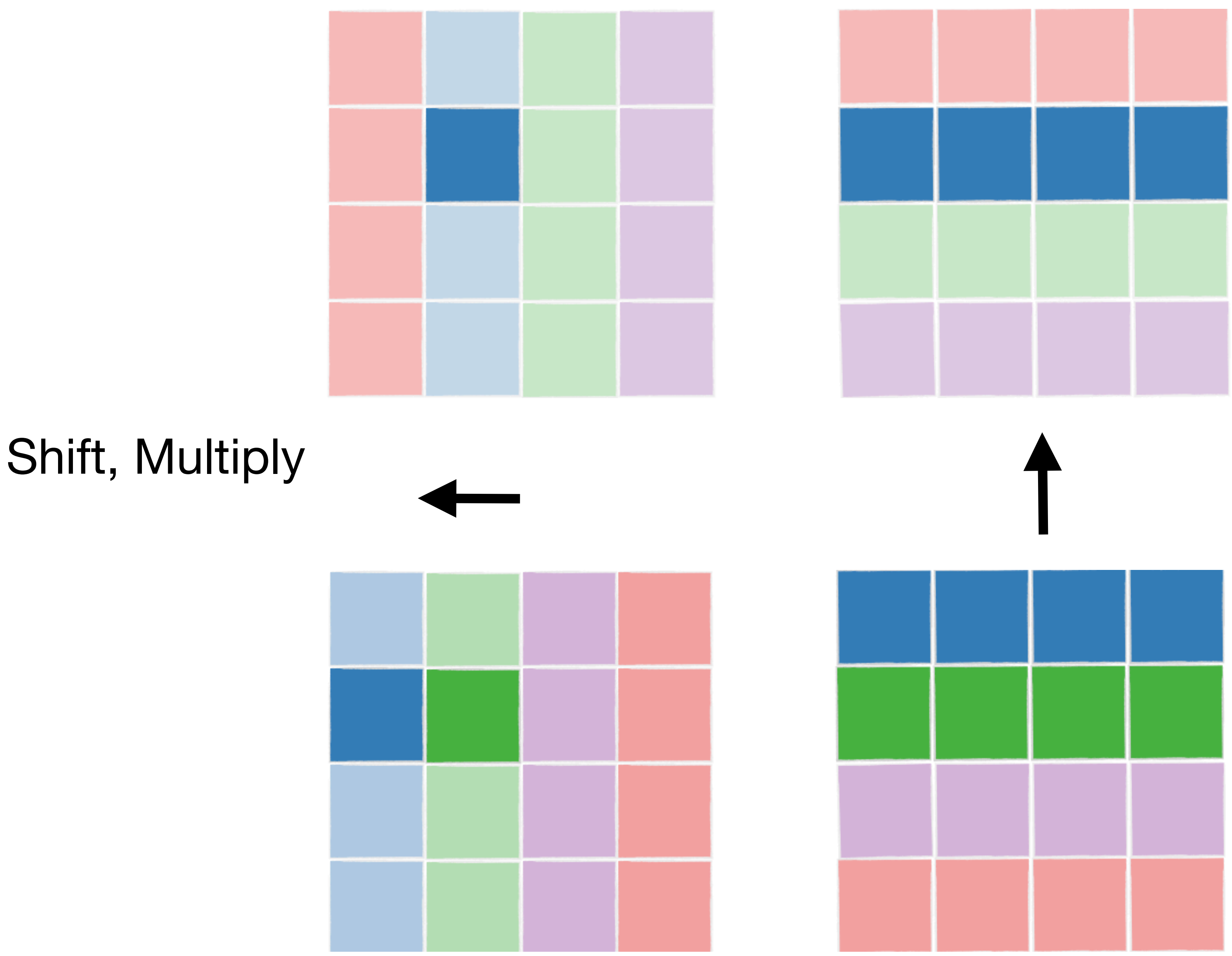
Step 1

Broadcast diagonal element to each member of row

Multiply

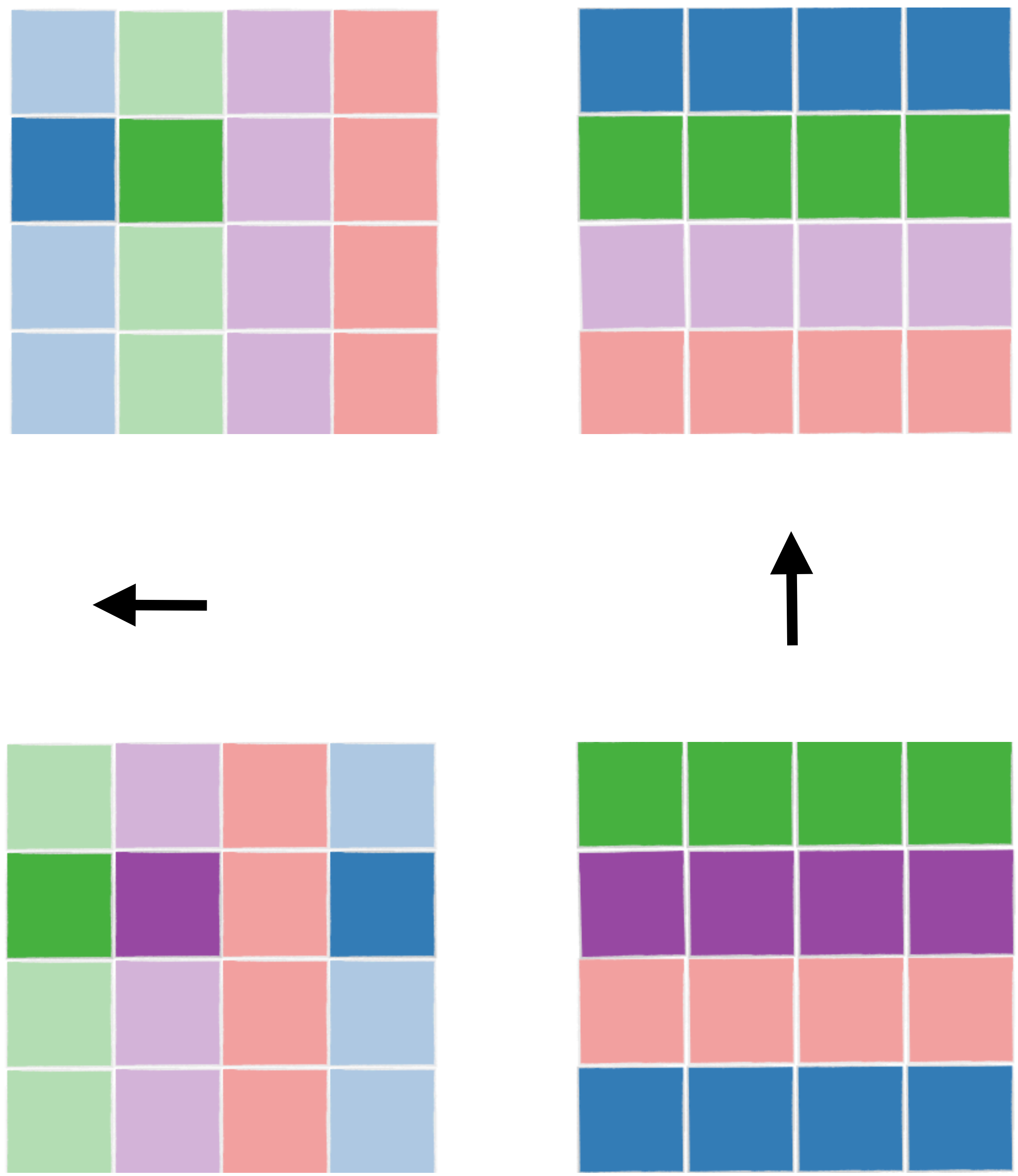


Step 2

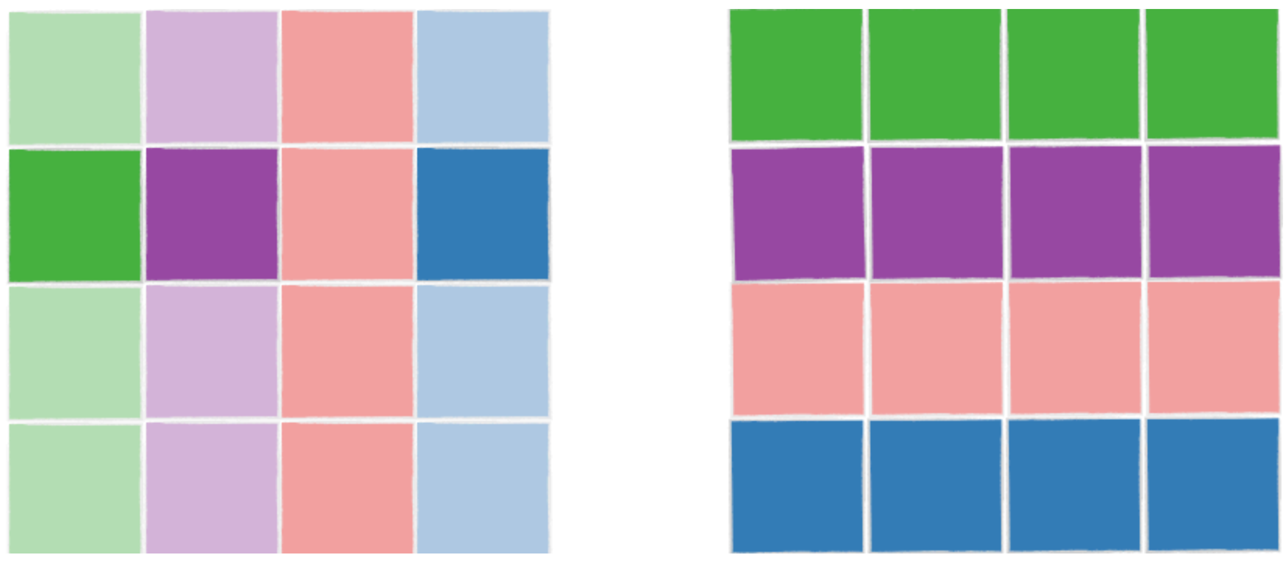


Step 3

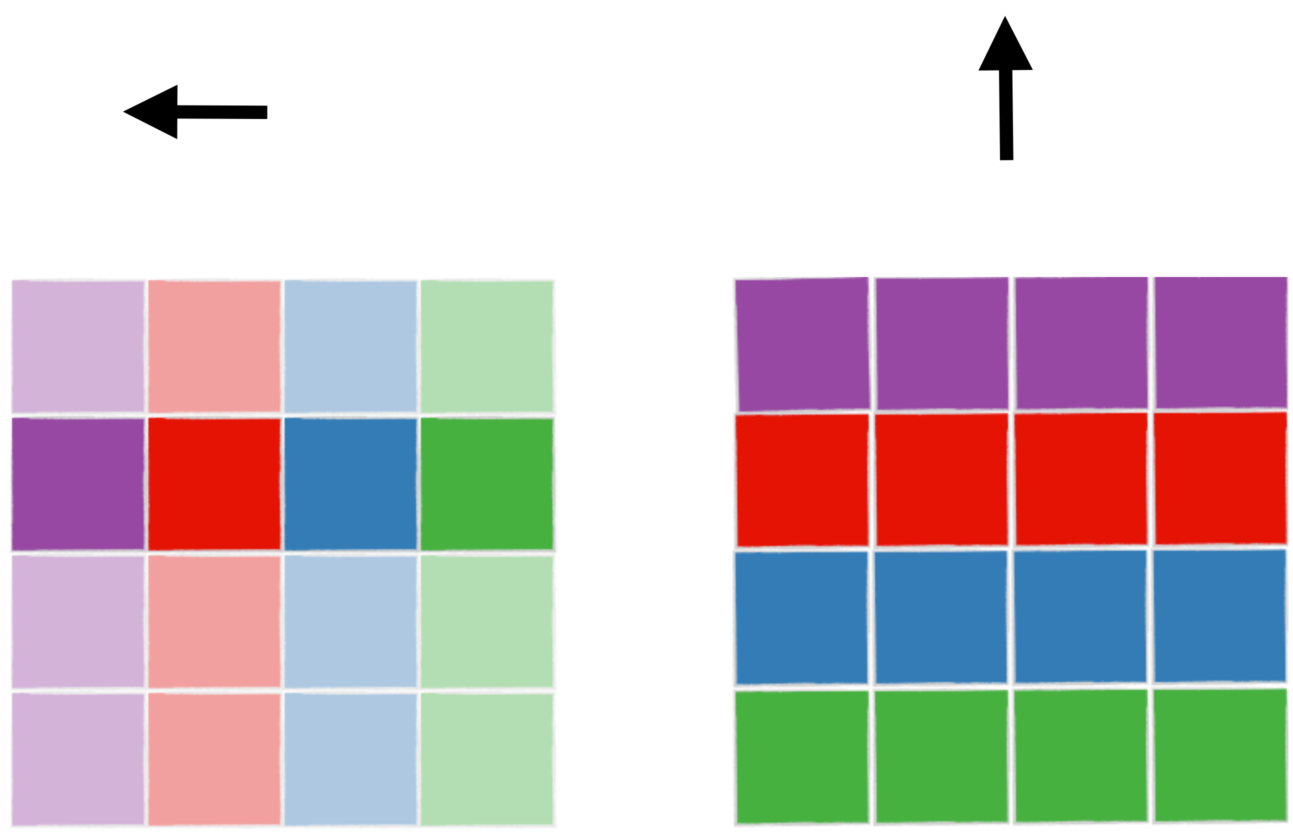
Shift, Multiply



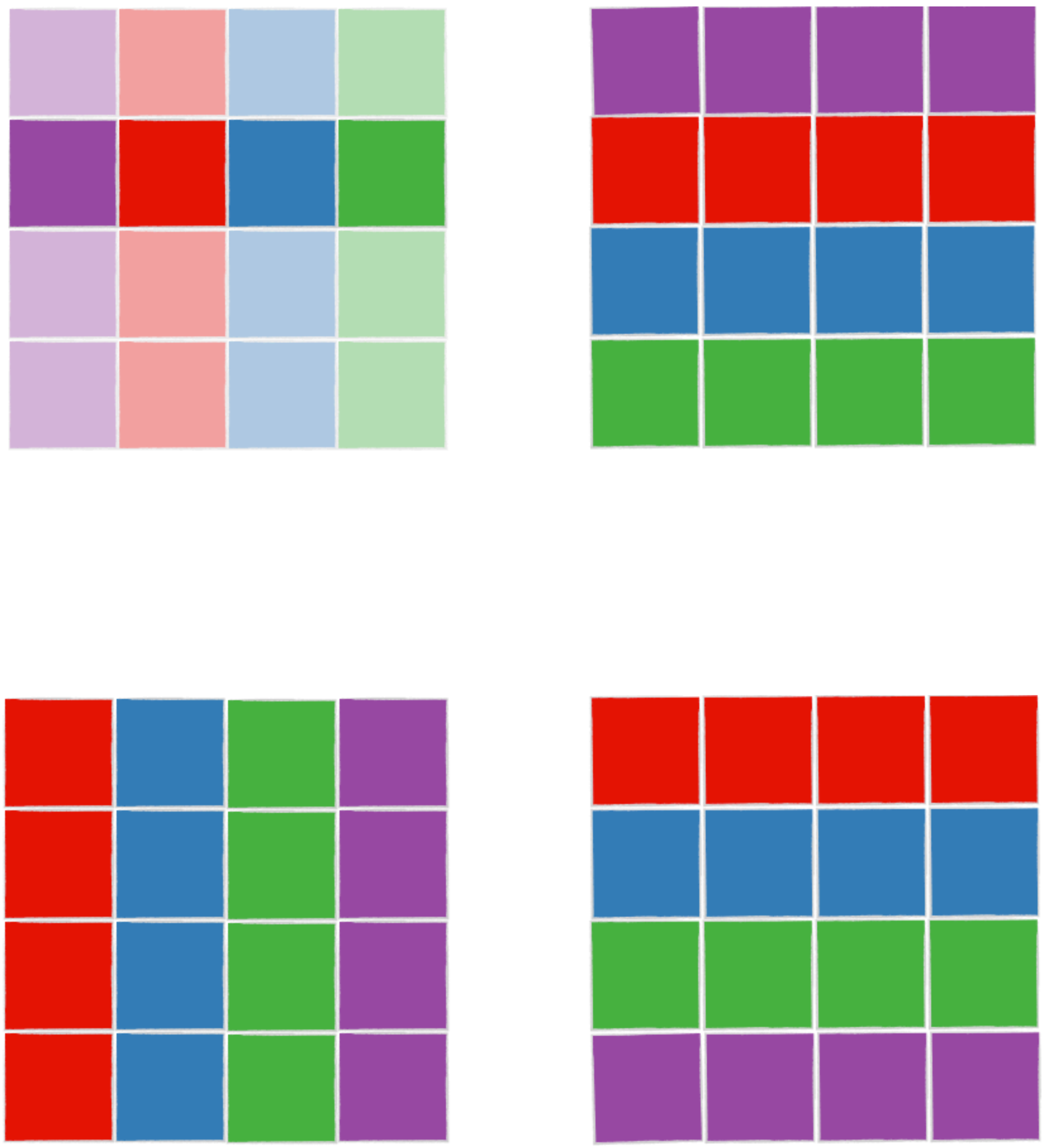
Step 4



Shift, Multiply



Done



Advice

Start early

Don’t worry about IO at first.

- Read in entire matrix
- Send the correct commponents

Consider a class or struct for keeping information

Use a simple derived MPI_Datatype

- e.g. 1D array for matrix representation