Large-Scale Parallel Computing

Aamer Shah

shah@cs.tu-darmstadt.de

EXERCISE 6

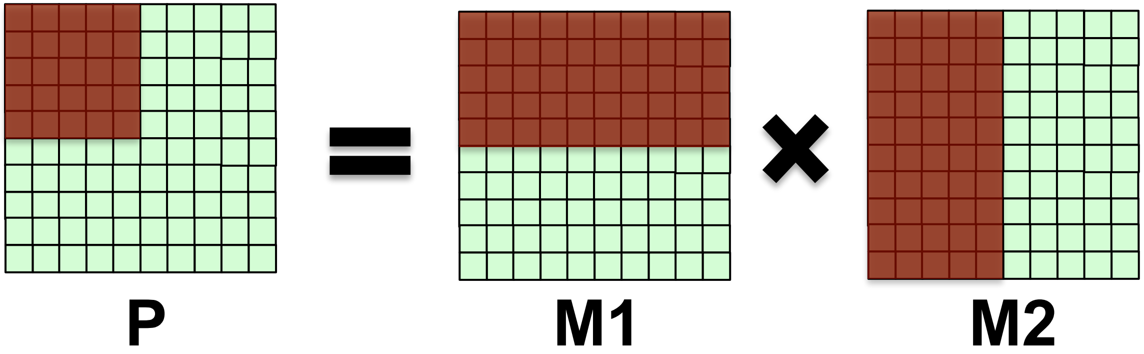
Hands-on session

- Hands-on session
- Students will develop the solution during the exercise session

1. Login to Lichtenberg cluster (with –Y option)
2. Copy **`ex06.tgz`** from **`/home/as65huly/public`**
3. Implement Task 1 in **`matmul_dist.c`**
   - Derived data types
4. Implement Task 2 in **`matmul_nbc.c`**
   - Non-blocking communication

Derived data types

- What is the problem? Matrix multiplication as an example
  - How to distribute matrix columns among processes?
    - Column is in non-contiguous memory location
    - Send element-by-element?
      - Tedious and takes time
  - Is there a way to tell MPI what a column is and then send it?

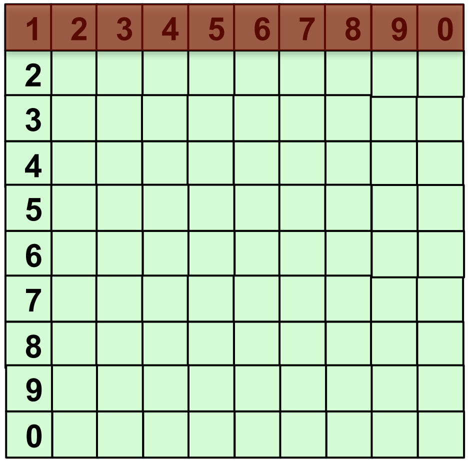


Type contiguous

- Different ways to define different types of derived data types
  - Create data types for arrays / rows of a matrix

```
int MPI_Type_contiguous(int count, MPI Datatype
oldtype, MPI Datatype *newtype)
```

```
   MPI_Type_contiguous(10, MPI_INT, &mat_row)
```



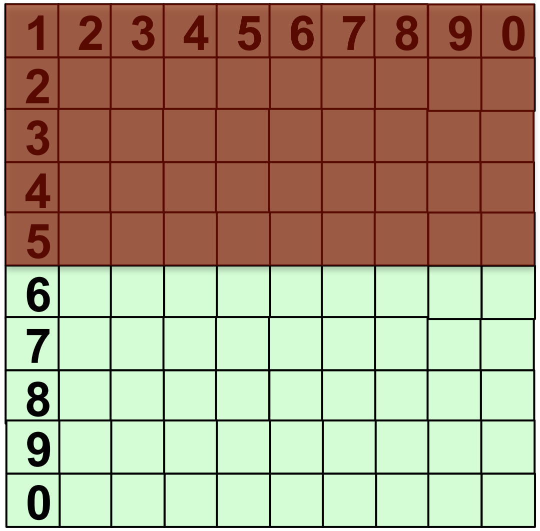
1	2	3	4	5	6	7	8	9	0
2									
3									
4									
5									
6									
7									
8									
9									
0									

Type contiguous

- MPI_Type_contiguous() can be used for a matrix row
- A single instance can cover only one row or multiple rows

```
   MPI_Type_contiguous(50, MPI_INT, &mat_rows)
```

```
MPI_Send(mat, 1, mat_rows, dest, tag, MPI_COMM_WORLD)
```

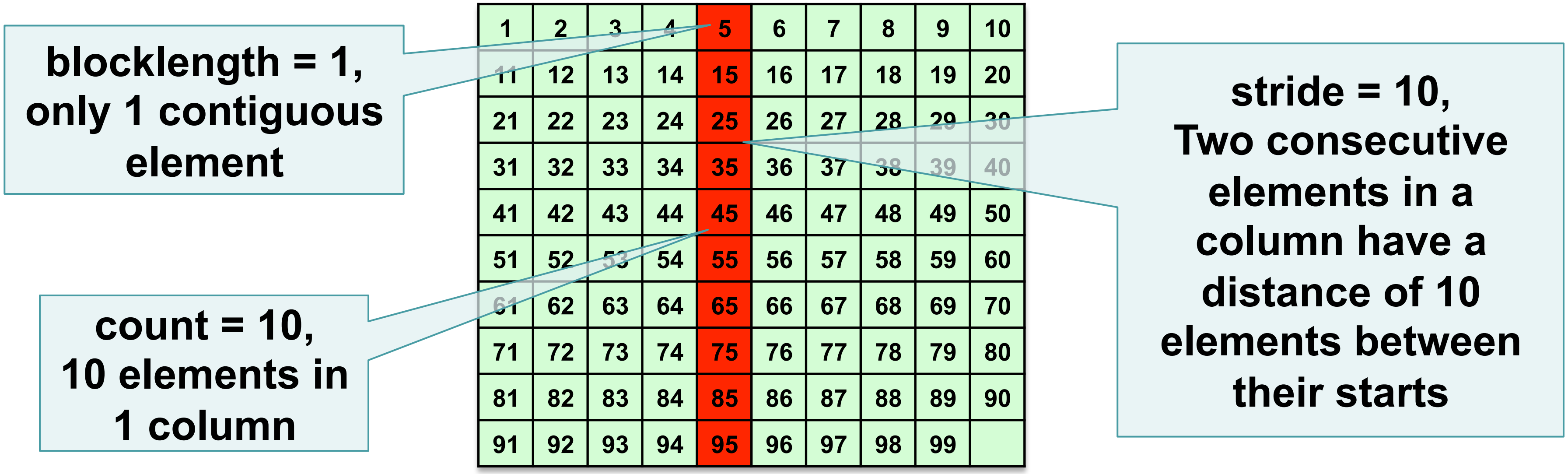


Type vector

- How to transfer columns?
  - Non contiguous in memory, BUT equal distance between each element

```
int MPI_Type_vector(int count, int blocklength, int stride,
MPI_Datatype oldtype, MPI_Datatype *newtype)
```

```
MPI_Type_vector(10, 1, 10, MPI_INT, &mat_col)
```



blocklength = 1, only 1 contiguous element

count = 10, 10 elements in 1 column

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

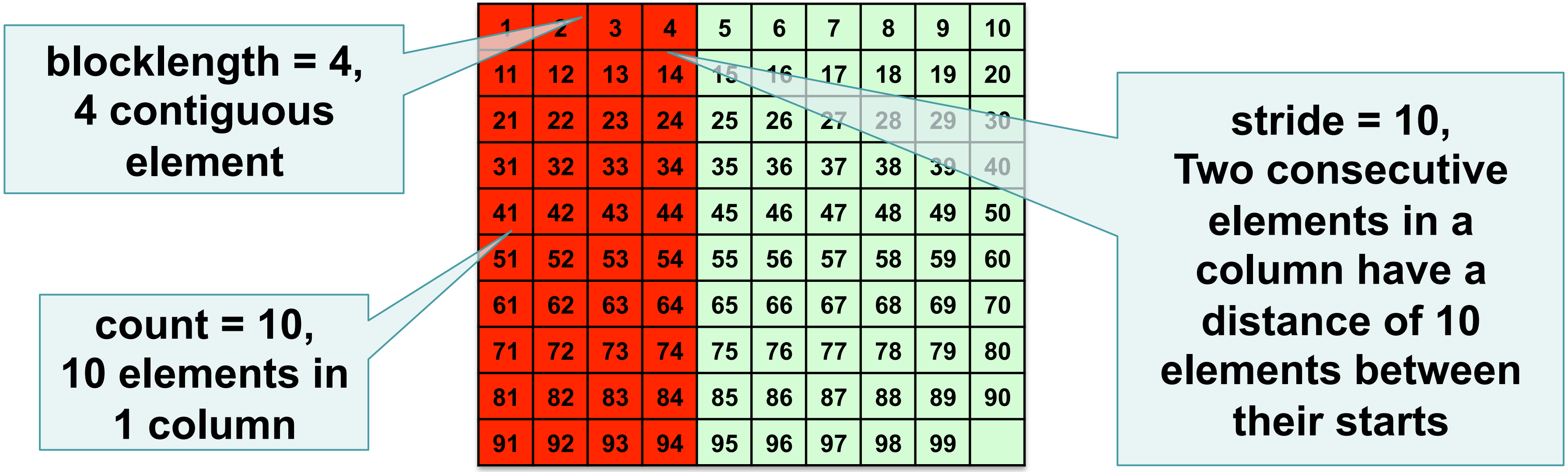
stride = 10, Two consecutive elements in a column have a distance of 10 elements between their starts

Type vector

- A single instance can cover either one column or multiple columns

```
MPI_Type_vector(10, 4, 10, MPI_INT, &mat_cols)
```

```
MPI_Send(mat, 1, mat_cols, dest, tag, MPI_COMM_WORLD)
```



1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

blocklength = 4, 4 contiguous element

count = 10, 10 elements in 1 column

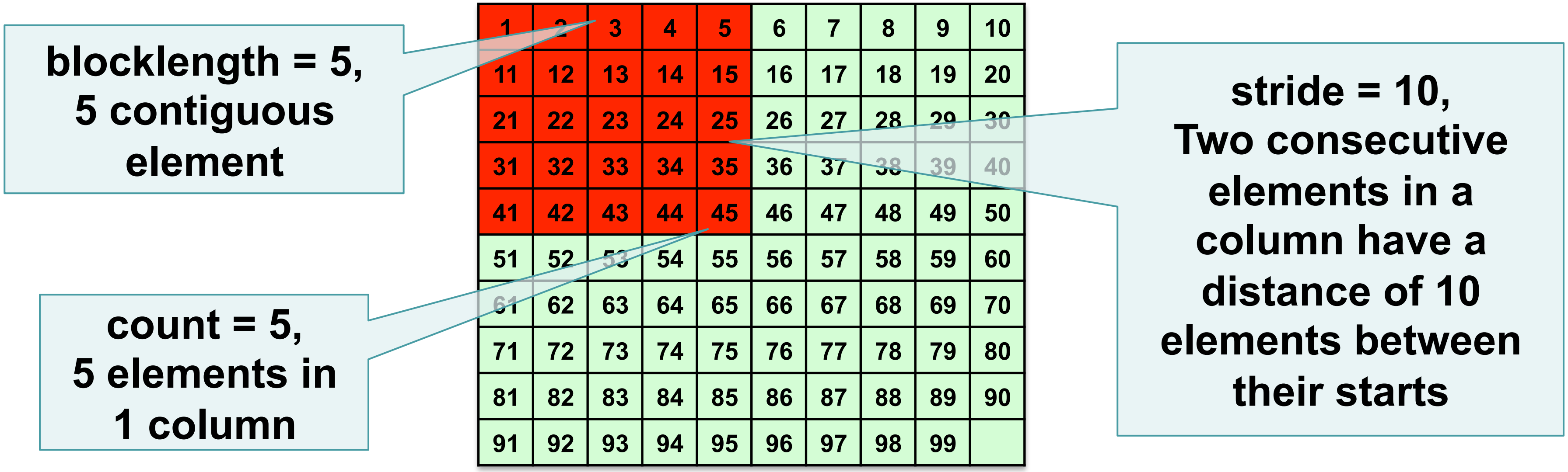
stride = 10, Two consecutive elements in a column have a distance of 10 elements between their starts

Type vector

- How to make a data type for a tile?

```
MPI_Type_vector(5, 5, 10, MPI_INT, &mat_tile)
```

```
MPI_Send(mat, 1, mat_tile, dest, tag, MPI_COMM_WORLD);
```



blocklength = 5,
5 contiguous
element

count = 5,
5 elements in
1 column

stride = 10,
Two consecutive
elements in a
column have a
distance of 10
elements between
their starts

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

Size and extent?

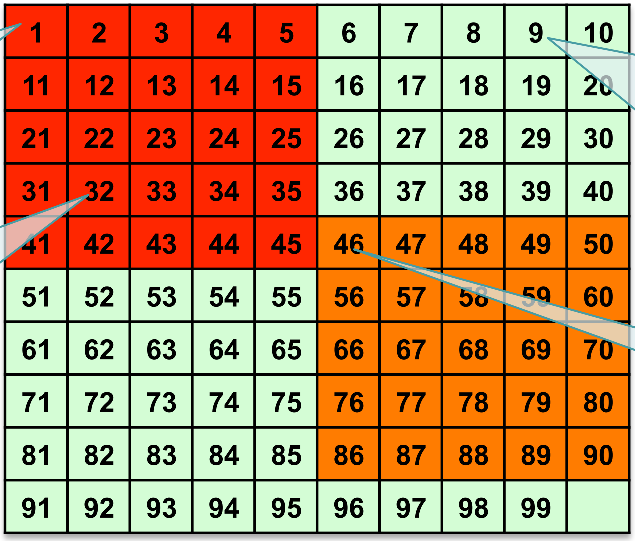
- Whats happens with this?
  - What is the share of the second process?

```
MPI_Type_vector(5, 5, 10, MPI_INT, &mat_tile)
```

```
MPI_Scatter(mat, 1, mat_tile, proc_mat, 1, mat_tile, 0, MPI_COMM_WORLD);
```

Rank 0 share starts here:

- Size is the amount of space needed to store the data



1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

- Extent includes the holes in between data types
- MPI communication functions use extent of data types

Rank 1 share starts here:

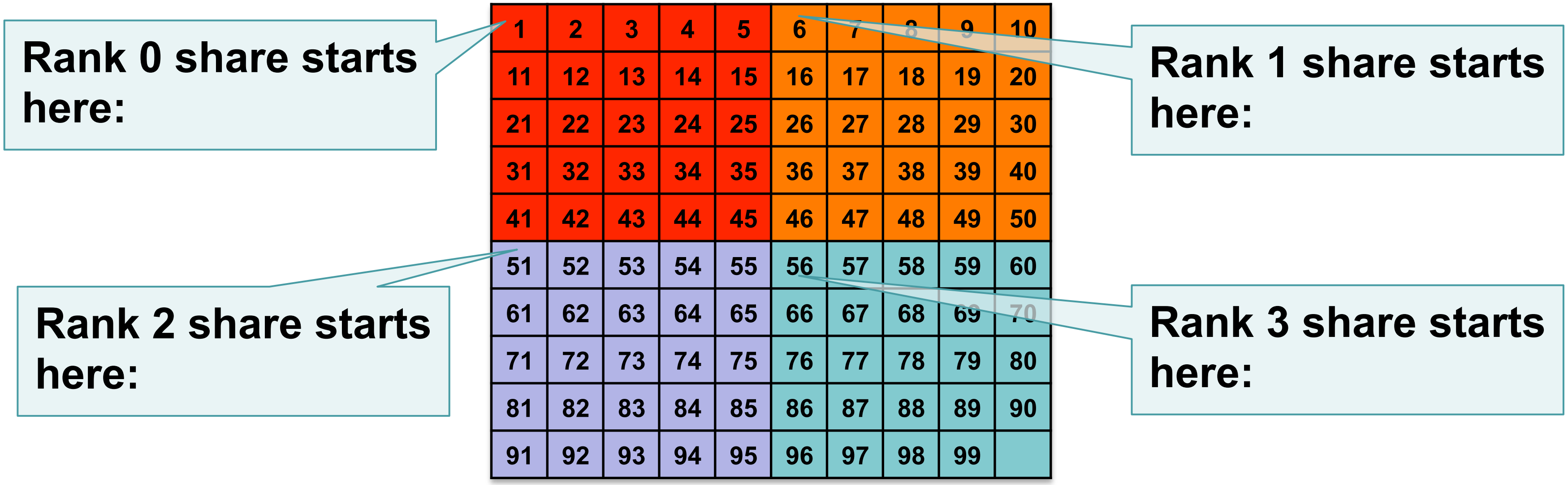
How to distribute columns/tiles?

```
MPI_Send(mat, 1, mat_tile, 0, tag, MPI_COMM_WORLD); //Rank 0
```

```
MPI_Send(&mat[tile_row_size], 1, mat_tile, 1, tag, MPI_COMM_WORLD); //Rank 1
```

```
MPI_Send(&mat[mat_row_size * tile_column_size], 1, mat_tile, 2, tag,
MPI_COMM_WORLD); //Rank 2
```

```
MPI_Send(&mat[mat_row_size * tile_column_size + tile_row_size], 1,
mat_tile, 2, tag, MPI_COMM_WORLD); //Rank 3
```



Rank 0 share starts here:

Rank 1 share starts here:

Rank 2 share starts here:

Rank 3 share starts here:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

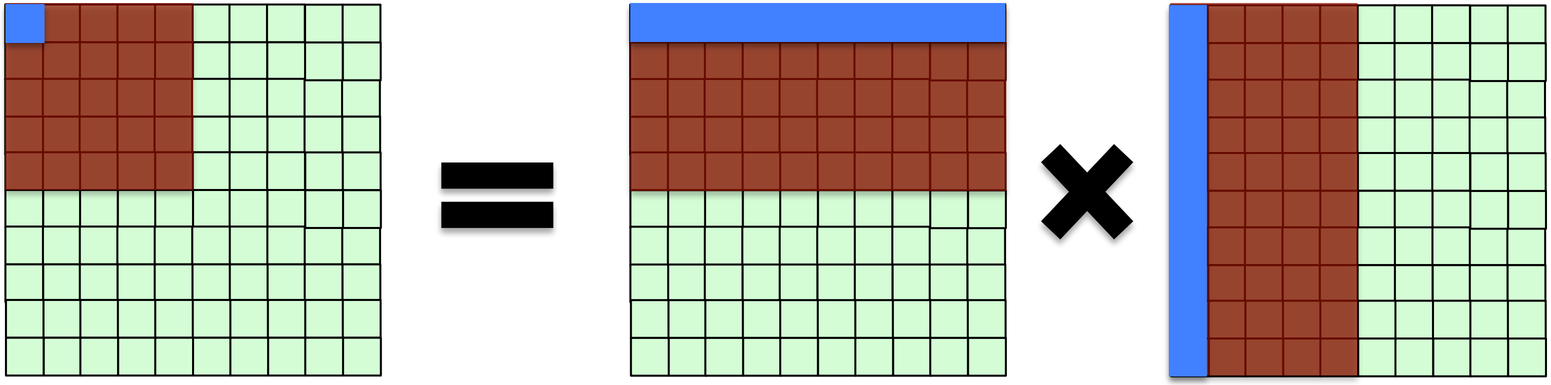
Task 1

- Matrix multiplication where each process calculates a product tile
- Use derived data types to distribute rows, columns and tiles
  - Can rows/columns be distributed by MPI_Scatter?
  - Can the tiles be collected by MPI_Gather?
- Steps

1. Create data types for rows/columns/tiles
2. Distribute rows
3. Distribute columns
4. Multiply
5. Collect tiles

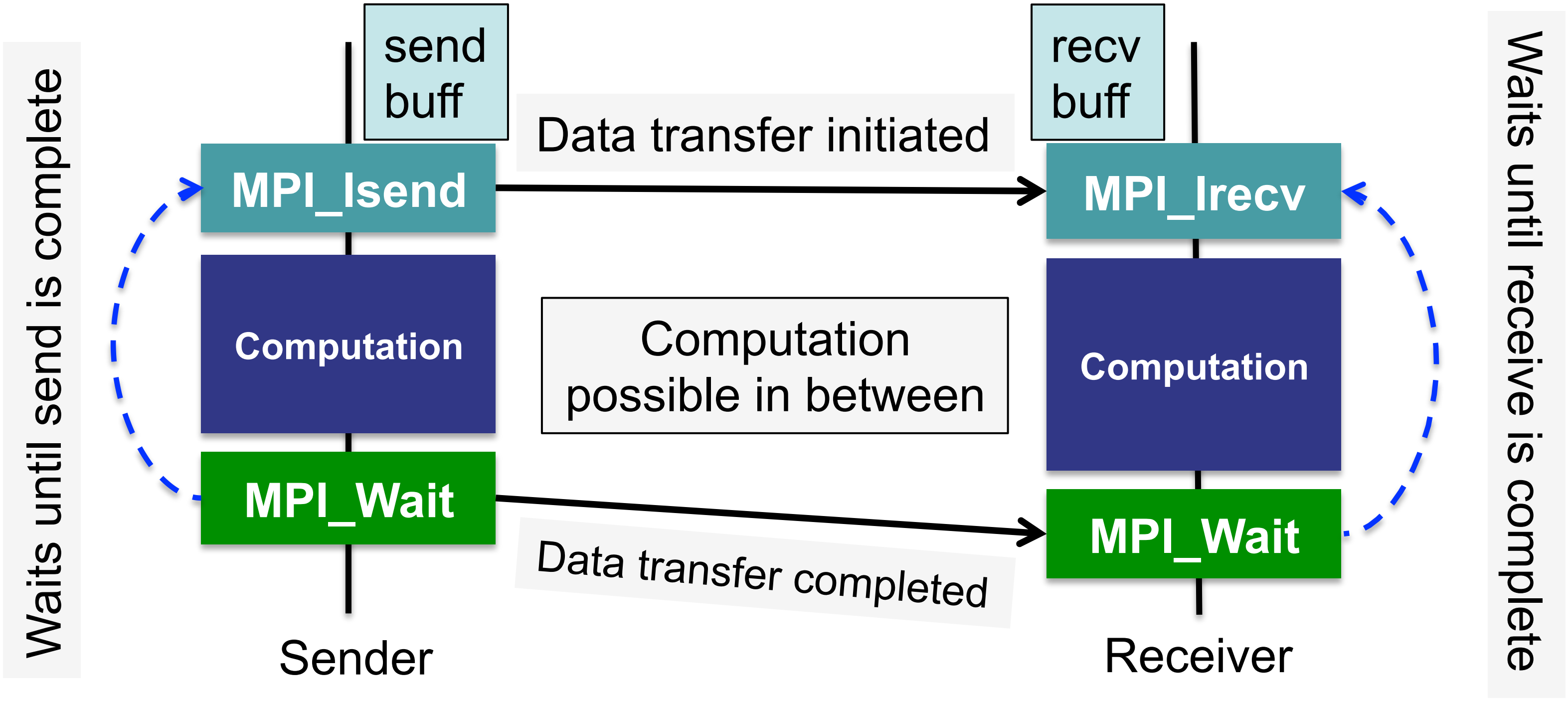
Non-blocking communication

- Distributing data among processes takes time
- Not all data is needed for each process to **start** multiplication
- If computation can be started when enough data is transferred to a process, the data distribution time can be reduced
- Non-blocking communication : overlap computation and communication



Non-blocking communication

sendbuff and recvbuff should not be modified between MPI_Isend/MPI_recv and MPI_Wait



send buff

recv buff

Data transfer initiated

MPI_Isend

MPI_Irecv

Computation

Computation possible in between

Computation

MPI_Wait

MPI_Wait

Data transfer completed

Waits until send is complete

Waits until receive is complete

Sender

Receiver

Non-blocking communication

```
int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm, MPI_Request *request)
```

```
int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int source,
int tag, MPI_Comm comm, MPI_Request *request)
```

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

```
int MPI_Waitall(int count, MPI_Request *array_of_requests, MPI_Status
*array_of_statuses)
```

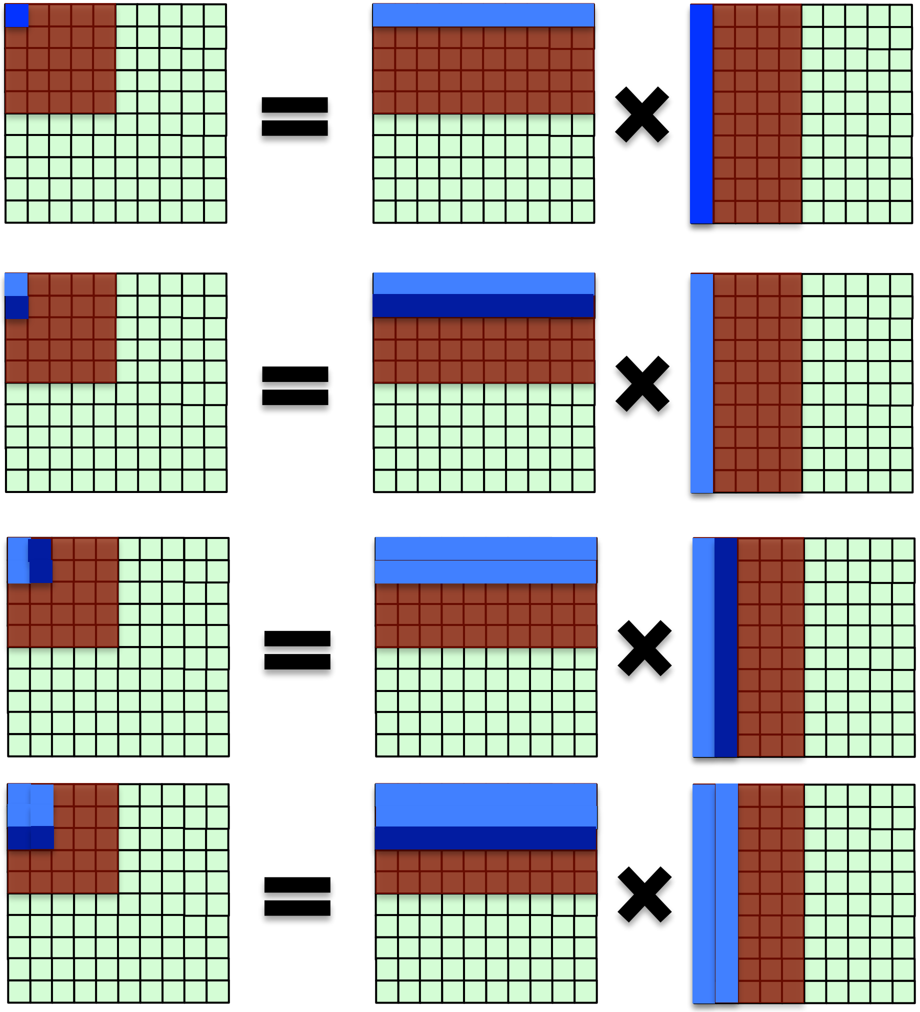
```
int MPI_Waitany(int count, MPI_Request *array_of_requests, int *index,
MPI_Status *status)
```

```
int MPI_Waitsome(int count, MPI_Request *array_of_requests, int
*numcompl, int *indices, MPI_Status *statuses)
```

Matrix multiplication

- Transfer rows and columns one-by-one

1. Receive a row and multiply it with already transferred columns
2. Receive a column and multiply it with already transferred rows



Task 2

- Use non-blocking communication to transfer rows and columns
- The data type should be defined such that
  - Row type is two rows of the actual matrix
  - Column type is two columns of actual matrix
- Use element-wise multiplication algorithm to find product
- Collect product at the end using the same method as Task 1