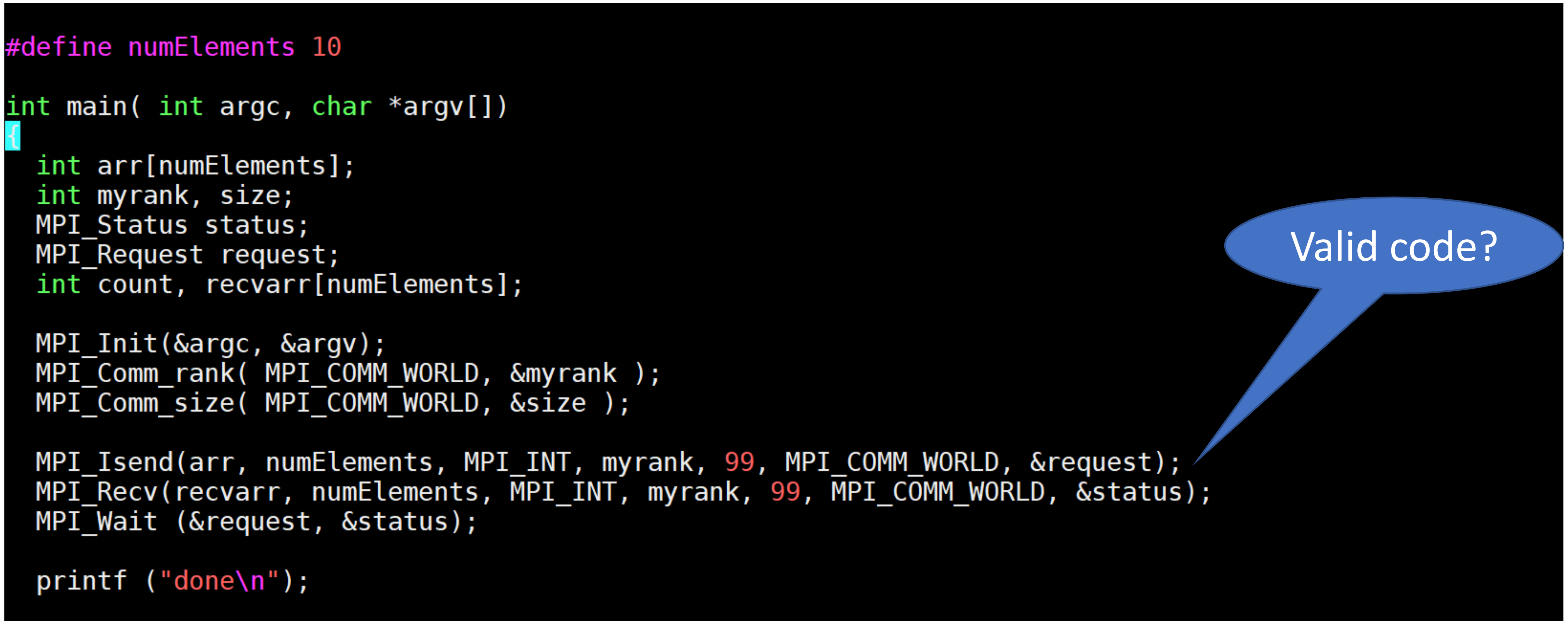
Datatypes

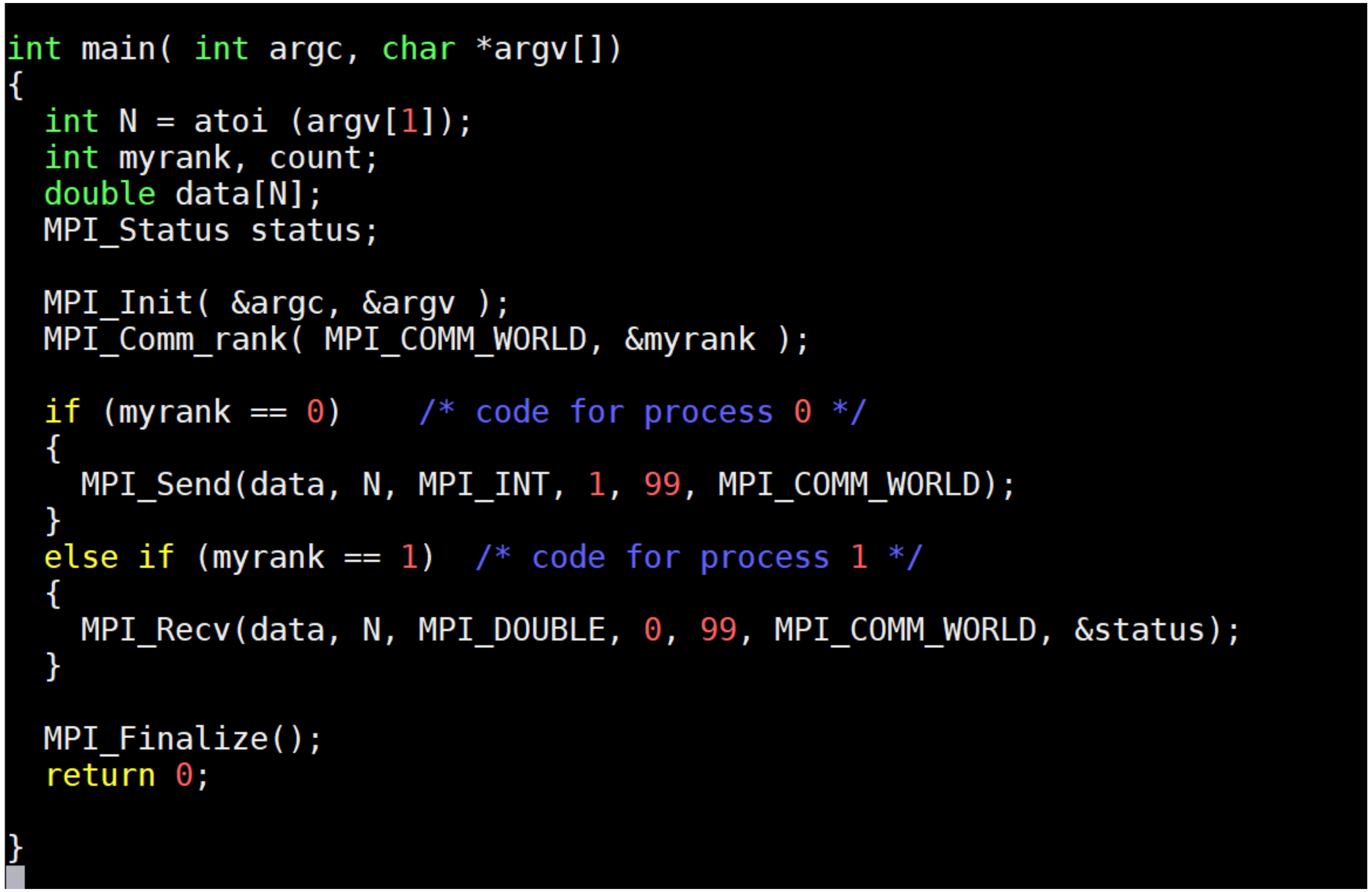
Feb 2, 2021

Revision: Example 1



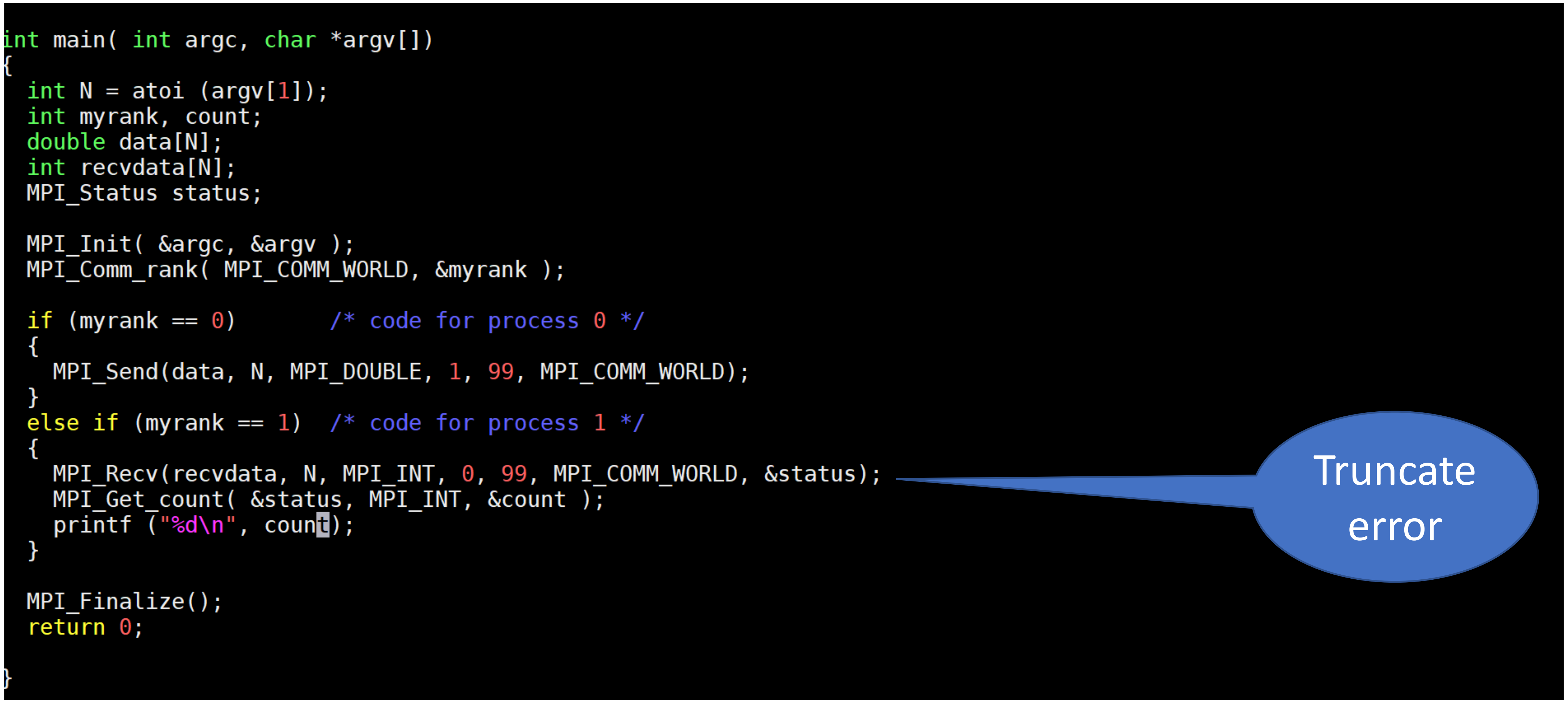
```
#define numElements 10

int main( int argc, char *argv[])
{
  int arr[numElements];
  int myrank, size;
  MPI_Status status;
  MPI_Request request;
  int count, recvarr[numElements];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  MPI_Comm_size( MPI_COMM_WORLD, &size );

  MPI_Isend(arr, numElements, MPI_INT, myrank, 99, MPI_COMM_WORLD, &request);
  MPI_Recv(recvarr, numElements, MPI_INT, myrank, 99, MPI_COMM_WORLD, &status);
  MPI_Wait (&request, &status);

  printf ("done\n");
```

Revision: Example 2



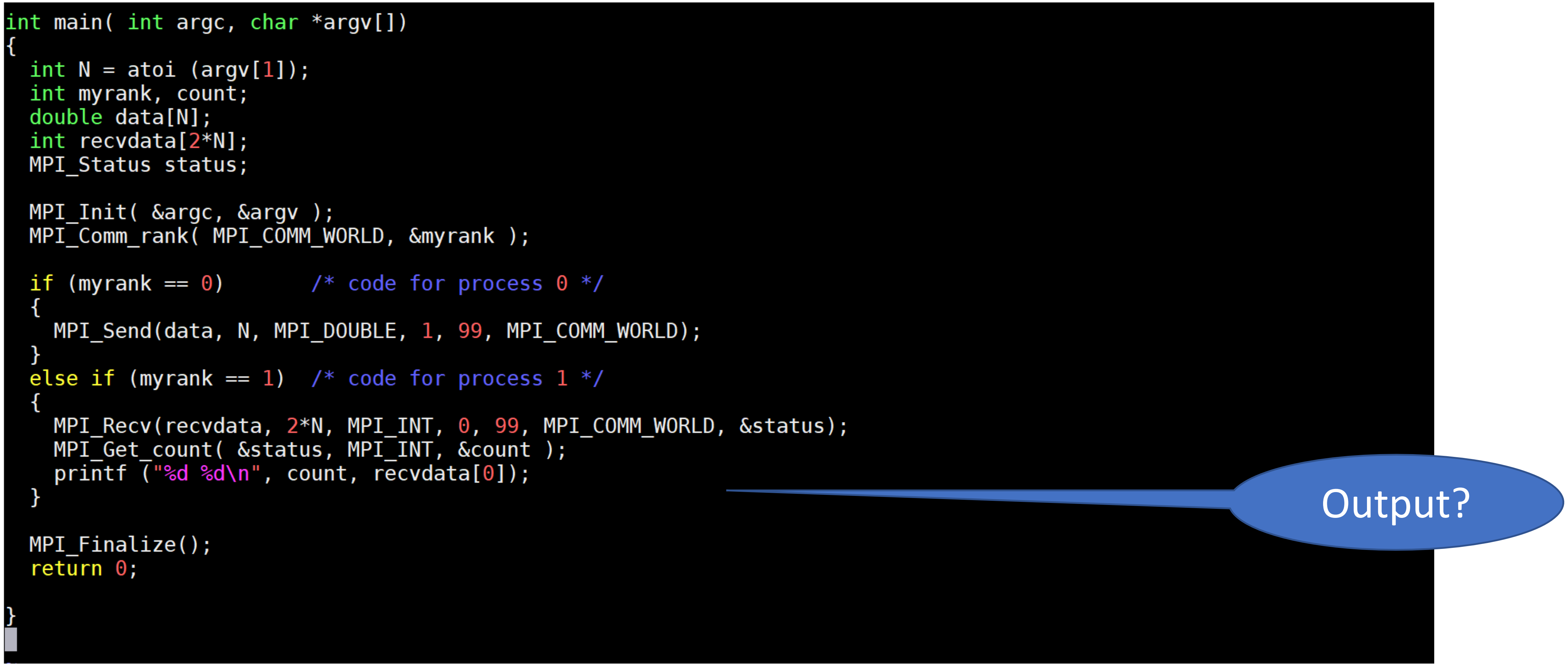
```
int main( int argc, char *argv[])
{
  int N = atoi (argv[1]);
  int myrank, count;
  double data[N];
  MPI_Status status;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

  if (myrank == 0)    /* code for process 0 */
  {
    MPI_Send(data, N, MPI_INT, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(data, N, MPI_DOUBLE, 0, 99, MPI_COMM_WORLD, &status);
  }

  MPI_Finalize();
  return 0;

}
```

Revision: Example 3



```
int main( int argc, char *argv[])
{
  int N = atoi (argv[1]);
  int myrank, count;
  double data[N];
  int recvdata[N];
  MPI_Status status;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

  if (myrank == 0)       /* code for process 0 */
  {
    MPI_Send(data, N, MPI_DOUBLE, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(recvdata, N, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
    MPI_Get_count( &status, MPI_INT, &count );
    printf ("%d\n", count);
  }

  MPI_Finalize();
  return 0;

}
```

Revision: Example 4



```
int main( int argc, char *argv[])
{
  int N = atoi (argv[1]);
  int myrank, count;
  double data[N];
  int recvdata[2*N];
  MPI_Status status;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

  if (myrank == 0)       /* code for process 0 */
  {
    MPI_Send(data, N, MPI_DOUBLE, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(recvdata, 2*N, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
    MPI_Get_count( &status, MPI_INT, &count );
    printf ("%d %d\n", count, recvdata[0]);
  }

  MPI_Finalize();
  return 0;

}
```

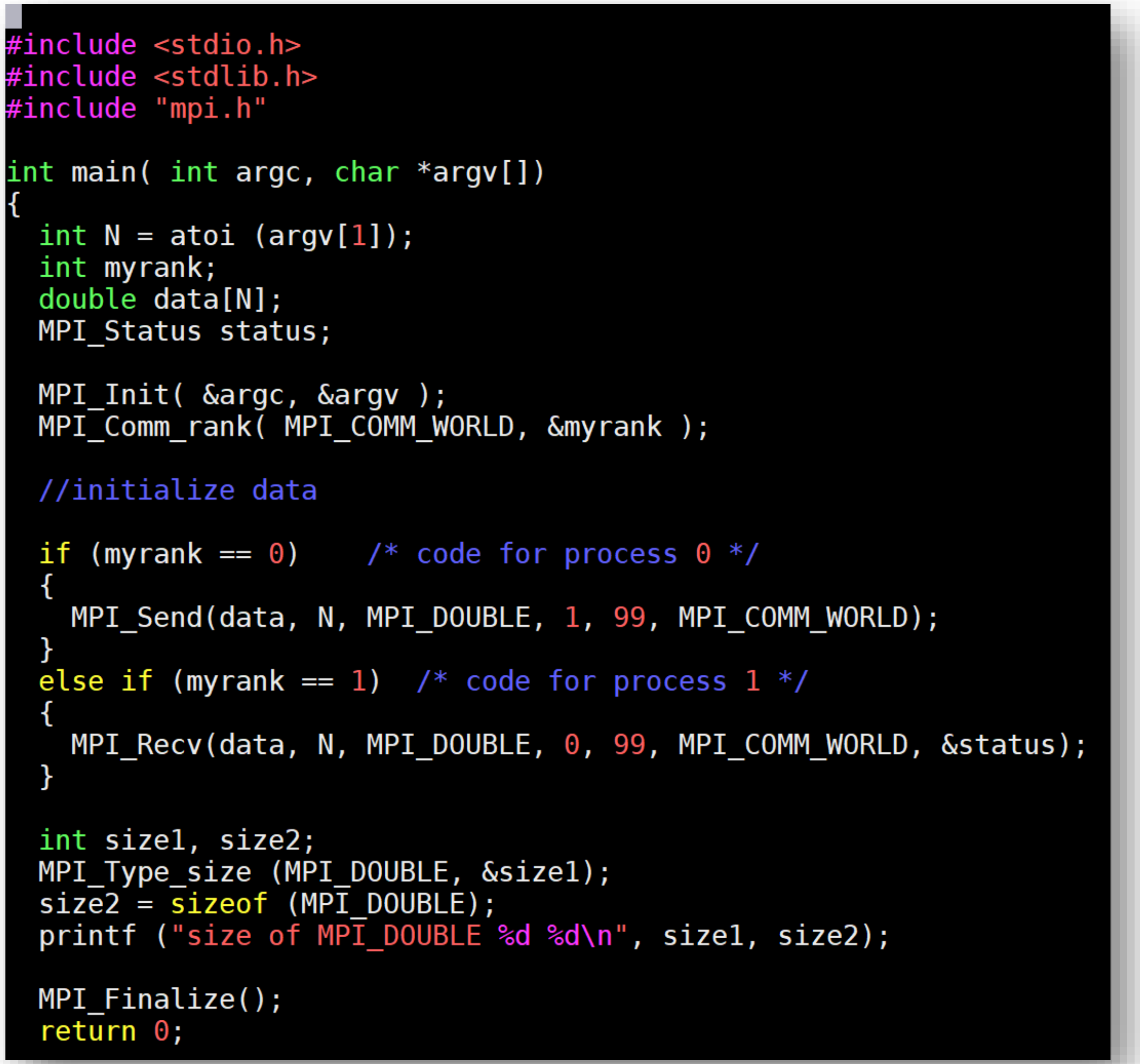
Revision: Safety

```
IF (rank.EQ.0) THEN
   MPI_SSEND (buf1, count, MPI_INT, 1, 0, comm, req1)
   MPI_SEND (buf2, count, MPI_INT, 1, 1, comm, req2)
ELSE IF (rank.EQ.1) THEN
   MPI_IRECV (buf1, count, MPI_INT, 0, 0, comm, req1)
   MPI_RECV (buf2, count, MPI_INT, 0, 1, comm, req2)
   MPI_WAIT (req1, status1)
END IF
```

Datatypes

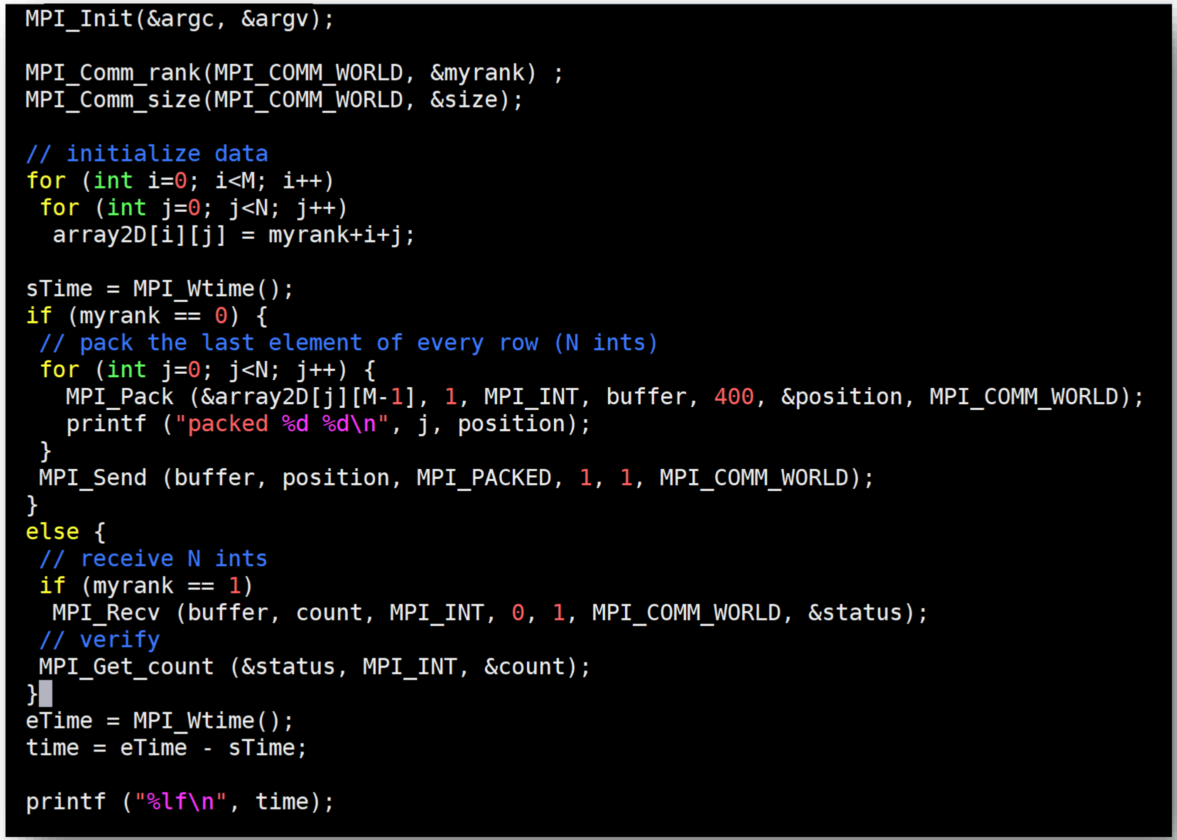
sizeof (MPI_DOUBLE);

MPI_Type_size

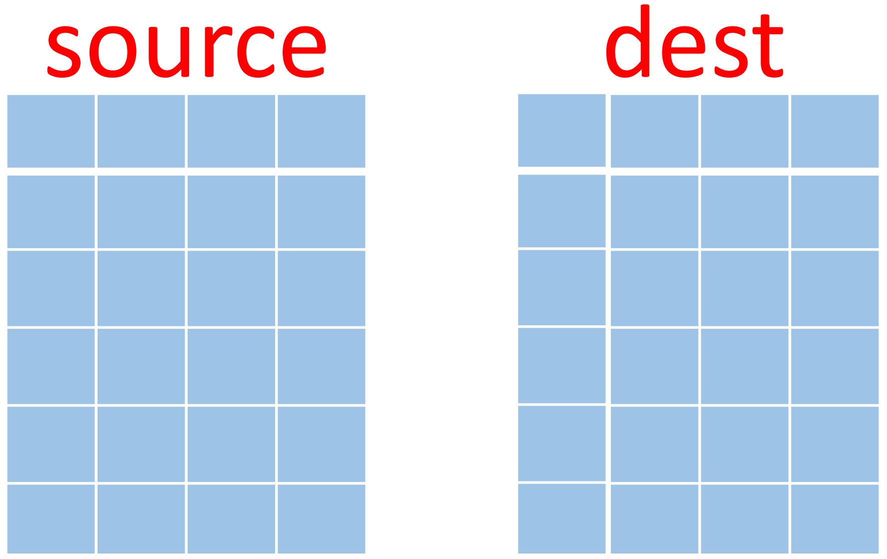


```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
  int N = atoi (argv[1]);
  int myrank;
  double data[N];
  MPI_Status status;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

  //initialize data

  if (myrank == 0)    /* code for process 0 */
  {
    MPI_Send(data, N, MPI_DOUBLE, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(data, N, MPI_DOUBLE, 0, 99, MPI_COMM_WORLD, &status);
  }

  int size1, size2;
  MPI_Type_size (MPI_DOUBLE, &size1);
  size2 = sizeof (MPI_DOUBLE);
  printf ("size of MPI_DOUBLE %d %d\n", size1, size2);

  MPI_Finalize();
  return 0;
```

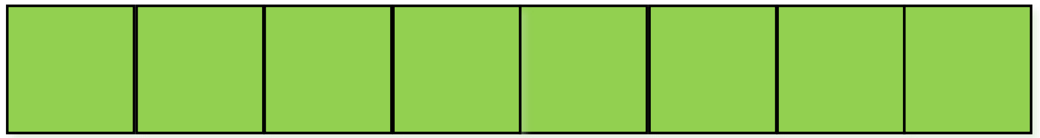
MPI_PACK



```
MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
MPI_Comm_size(MPI_COMM_WORLD, &size);

// initialize data
for (int i=0; i<M; i++)
 for (int j=0; j<N; j++)
  array2D[i][j] = myrank+i+j;

sTime = MPI_Wtime();
if (myrank == 0) {
 // pack the last element of every row (N ints)
 for (int j=0; j<N; j++) {
   MPI_Pack (&array2D[j][M-1], 1, MPI_INT, buffer, 400, &position, MPI_COMM_WORLD);
   printf ("packed %d %d\n", j, position);
 }
 MPI_Send (buffer, position, MPI_PACKED, 1, 1, MPI_COMM_WORLD);
}
else {
 // receive N ints
 if (myrank == 1)
  MPI_Recv (buffer, count, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
 // verify
 MPI_Get_count (&status, MPI_INT, &count);
}
eTime = MPI_Wtime();
time = eTime - sTime;

printf ("%lf\n", time);
```

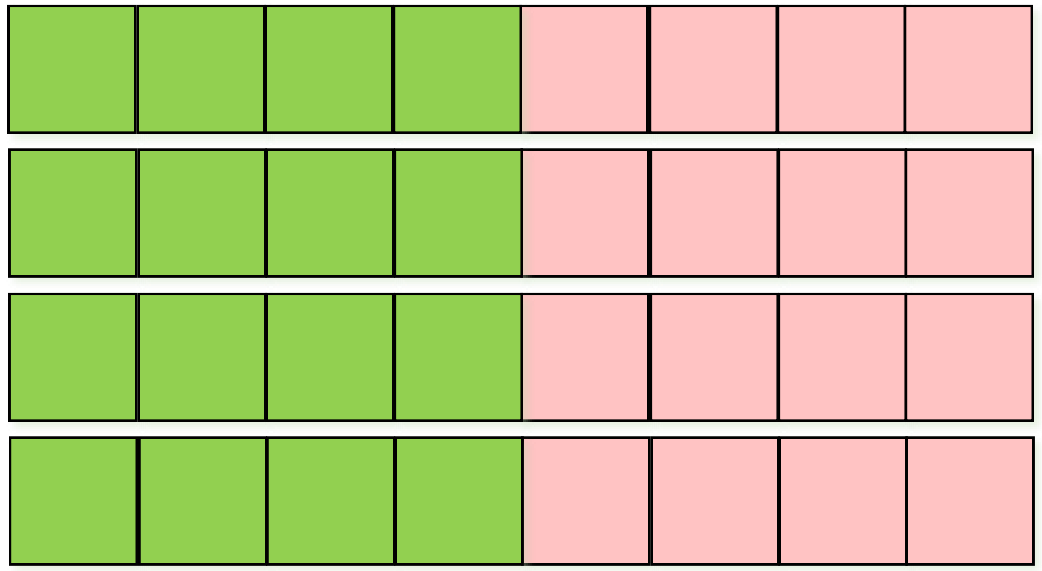


MPI Derived Datatypes



Q: How would you send contiguous data ?

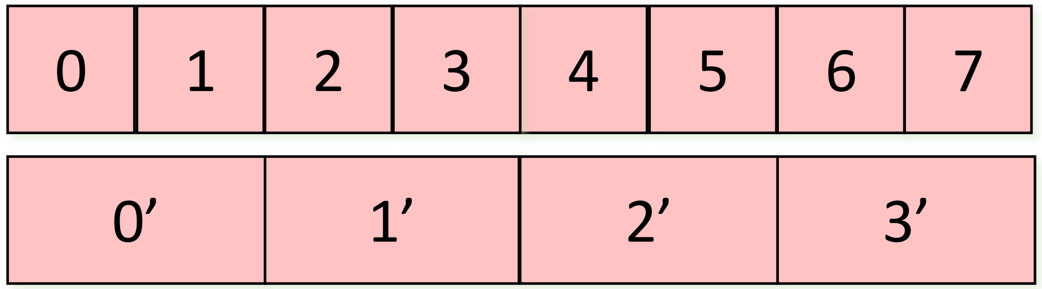
A: MPI_Send (specify starting address, #bytes)



Q: How would you send block of data (non-contiguous data transfer) ?

A: MPI derived datatypes

MPI Derived Datatypes

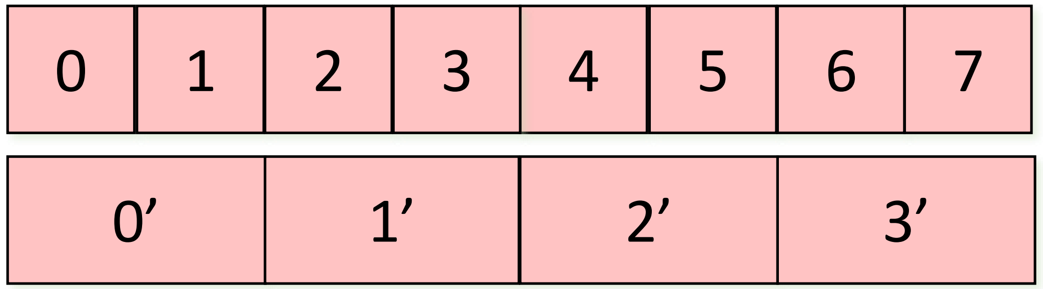


count = ?

Contiguous

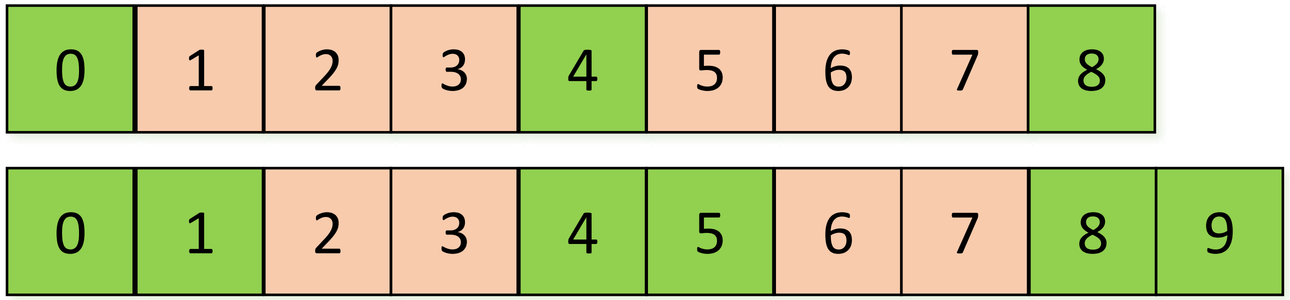
- Replication of datatype into contiguous locations
- MPI_Type_contiguous (count, oldtype, newtype)
    - count – replication count

MPI_Type_contiguous



```
int buffer[200];
MPI_Type_contiguous (2, MPI_INT, &twoints)
MPI_Type_commit (&twoints)
MPI_Send (buffer, 10, twoints, …..)
```

MPI_Type_vector



MPI_Type_vector

count = #blocks

blocklength = #elements in each block

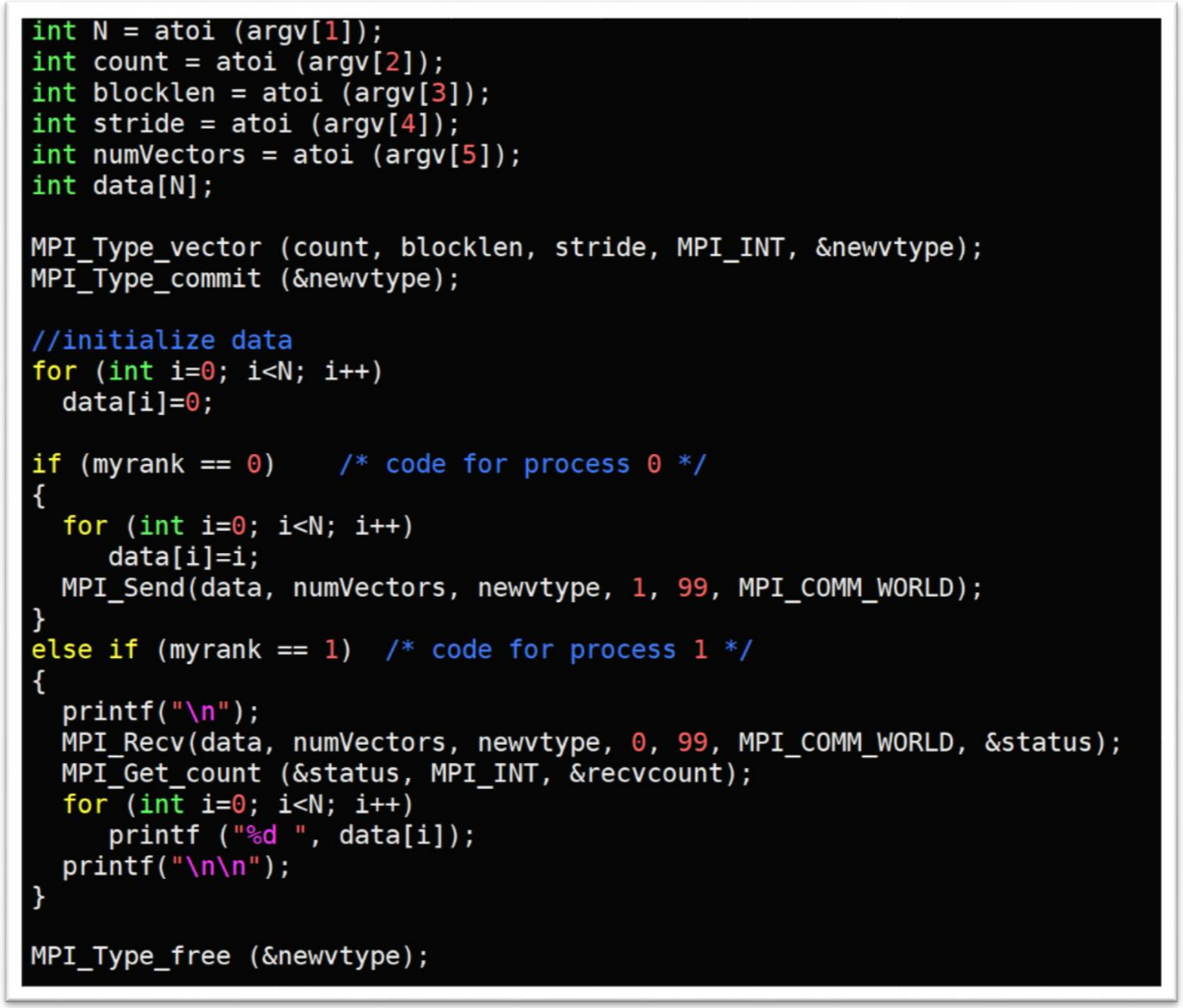
stride = #elements between start of each block

count = 3, blocklength = 1, stride = 4

count = 3, blocklength = 2, stride = 4

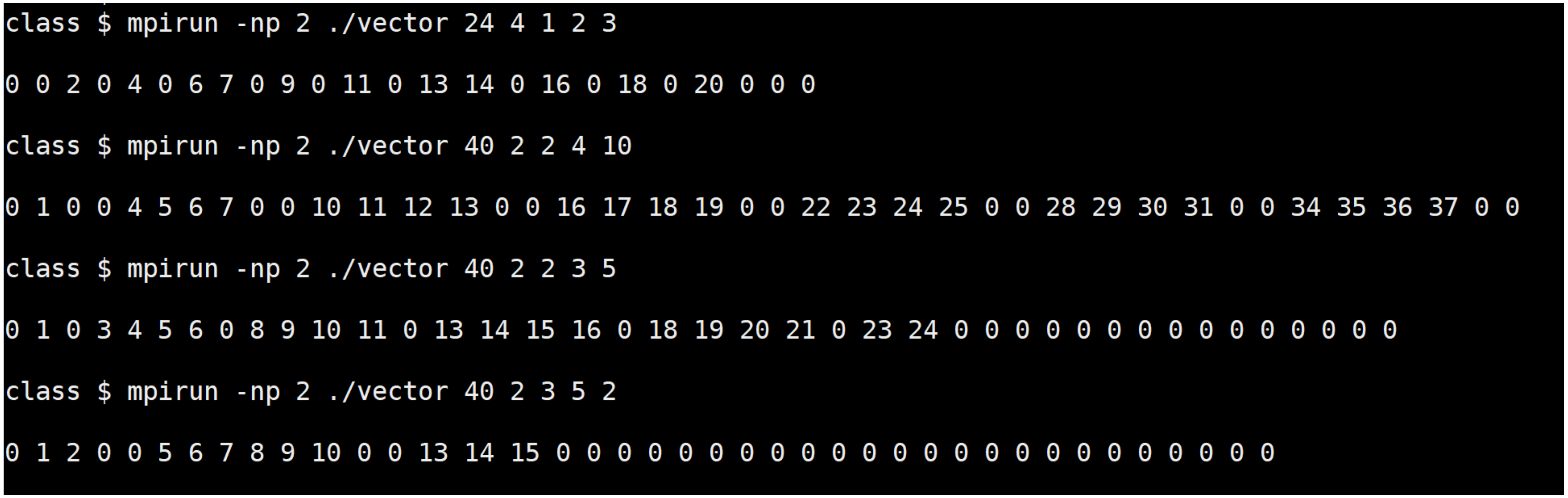
MPI_Type_vector (count, blocklength, stride, oldtype, newtype)

Code



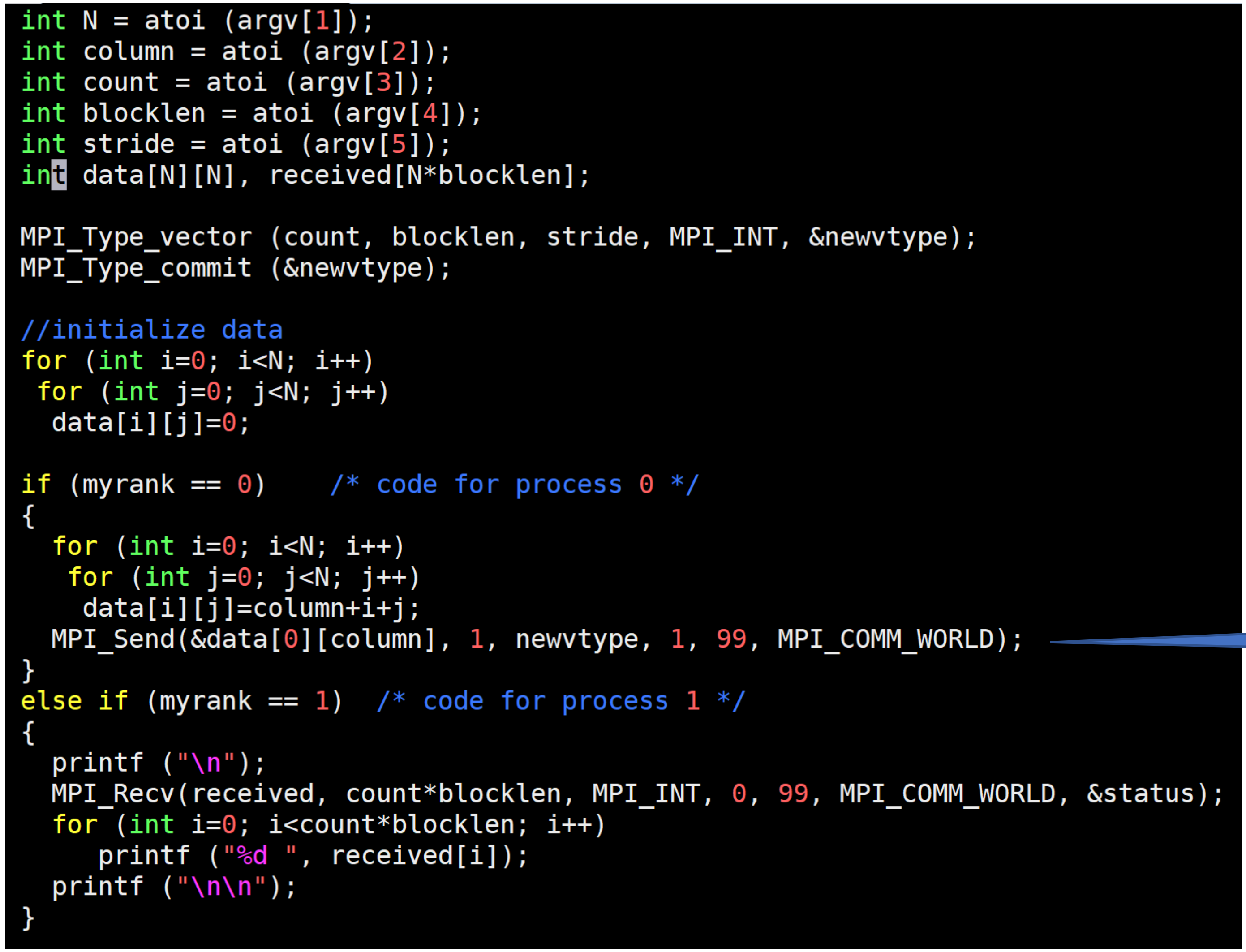
```
int N = atoi (argv[1]);
int count = atoi (argv[2]);
int blocklen = atoi (argv[3]);
int stride = atoi (argv[4]);
int numVectors = atoi (argv[5]);
int data[N];

MPI_Type_vector (count, blocklen, stride, MPI_INT, &newvtype);
MPI_Type_commit (&newvtype);

//initialize data
for (int i=0; i<N; i++)
  data[i]=0;

if (myrank == 0)    /* code for process 0 */
{
  for (int i=0; i<N; i++)
     data[i]=i;
  MPI_Send(data, numVectors, newvtype, 1, 99, MPI_COMM_WORLD);
}
else if (myrank == 1)  /* code for process 1 */
{
  printf("\n");
  MPI_Recv(data, numVectors, newvtype, 0, 99, MPI_COMM_WORLD, &status);
  MPI_Get_count (&status, MPI_INT, &recvcount);
  for (int i=0; i<N; i++)
     printf ("%d ", data[i]);
  printf("\n\n");
}

MPI_Type_free (&newvtype);
```

Output



```
class $ mpirun -np 2 ./vector 24 4 1 2 3

0 0 2 0 4 0 6 7 0 9 0 11 0 13 14 0 16 0 18 0 20 0 0 0

class $ mpirun -np 2 ./vector 40 2 2 4 10

0 1 0 0 4 5 6 7 0 0 10 11 12 13 0 0 16 17 18 19 0 0 22 23 24 25 0 0 28 29 30 31 0 0 34 35 36 37 0 0

class $ mpirun -np 2 ./vector 40 2 2 3 5

0 1 0 3 4 5 6 0 8 9 10 11 0 13 14 15 16 0 18 19 20 21 0 23 24 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

class $ mpirun -np 2 ./vector 40 2 3 5 2

0 1 2 0 0 5 6 7 8 9 10 0 0 13 14 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

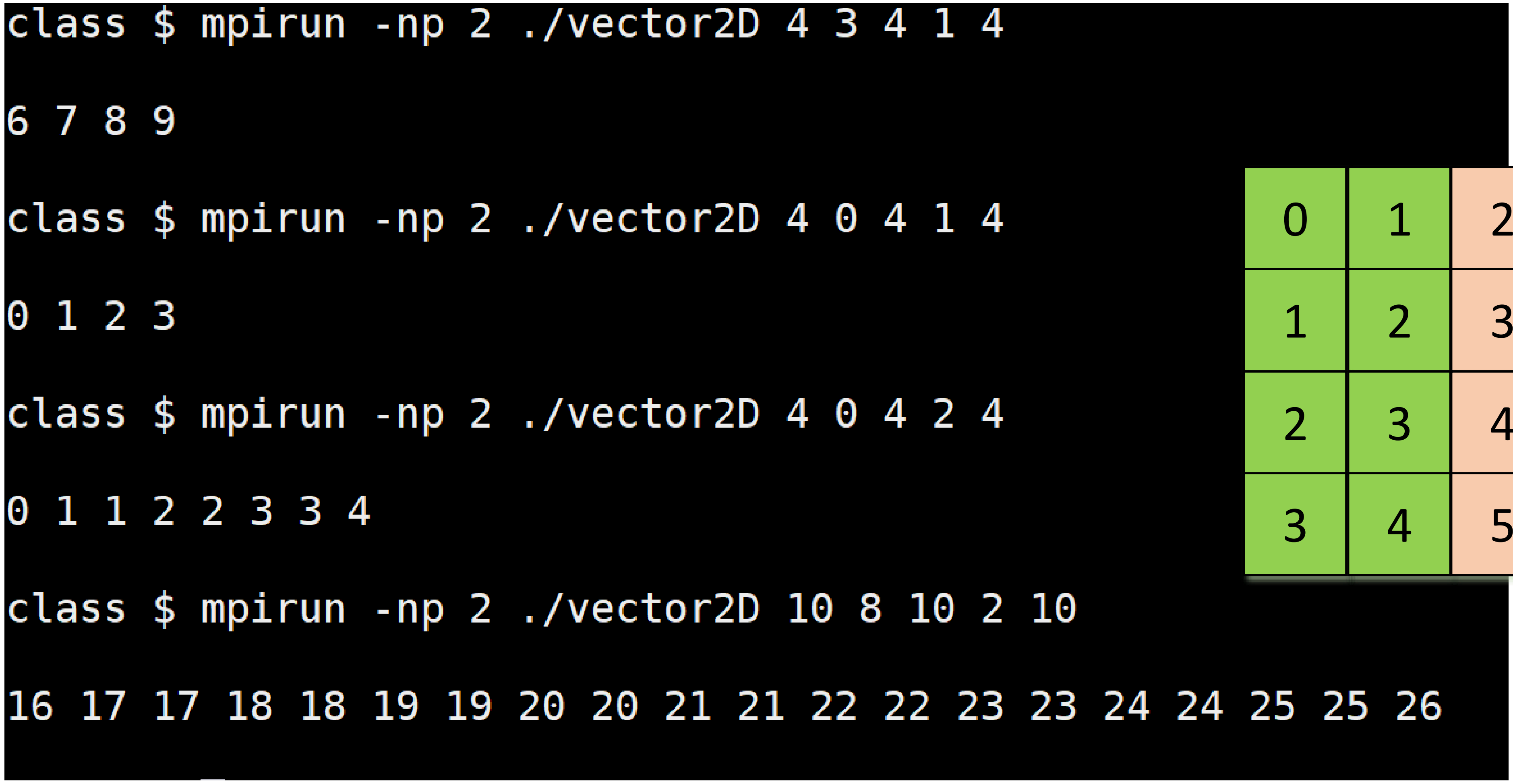
Code – Sending selected columns



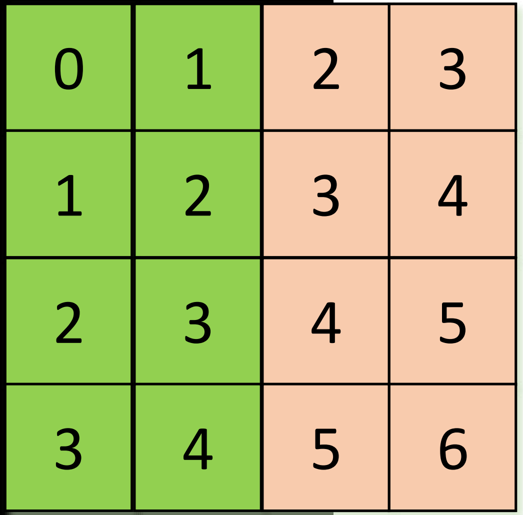
```
int N = atoi (argv[1]);
int column = atoi (argv[2]);
int count = atoi (argv[3]);
int blocklen = atoi (argv[4]);
int stride = atoi (argv[5]);
int data[N][N], received[N*blocklen];

MPI_Type_vector (count, blocklen, stride, MPI_INT, &newvtype);
MPI_Type_commit (&newvtype);

//initialize data
for (int i=0; i<N; i++)
 for (int j=0; j<N; j++)
  data[i][j]=0;

if (myrank == 0)    /* code for process 0 */
{
  for (int i=0; i<N; i++)
   for (int j=0; j<N; j++)
    data[i][j]=column+i+j;
  MPI_Send(&data[0][column], 1, newvtype, 1, 99, MPI_COMM_WORLD);
}
else if (myrank == 1)  /* code for process 1 */
{
  printf ("\n");
  MPI_Recv(received, count*blocklen, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
  for (int i=0; i<count*blocklen; i++)
     printf ("%d ", received[i]);
  printf ("\n\n");
}
```

Number of elements = 1

Output

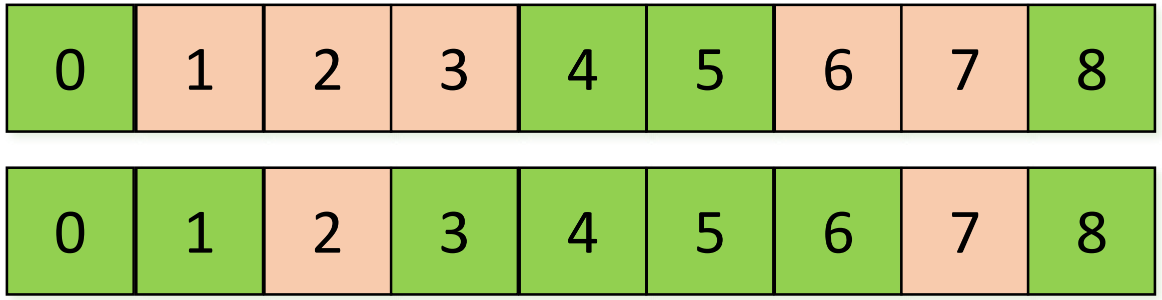


```
class $ mpirun -np 2 ./vector2D 4 3 4 1 4

6 7 8 9

class $ mpirun -np 2 ./vector2D 4 0 4 1 4

0 1 2 3

class $ mpirun -np 2 ./vector2D 4 0 4 2 4

0 1 1 2 2 3 3 4

class $ mpirun -np 2 ./vector2D 10 8 10 2 10

16 17 17 18 18 19 19 20 20 21 21 22 22 23 23 24 24 25 25 26
```



0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

MPI_Type_indexed



MPI_Type_indexed

count = #blocks

blocklengths = #elements in each block

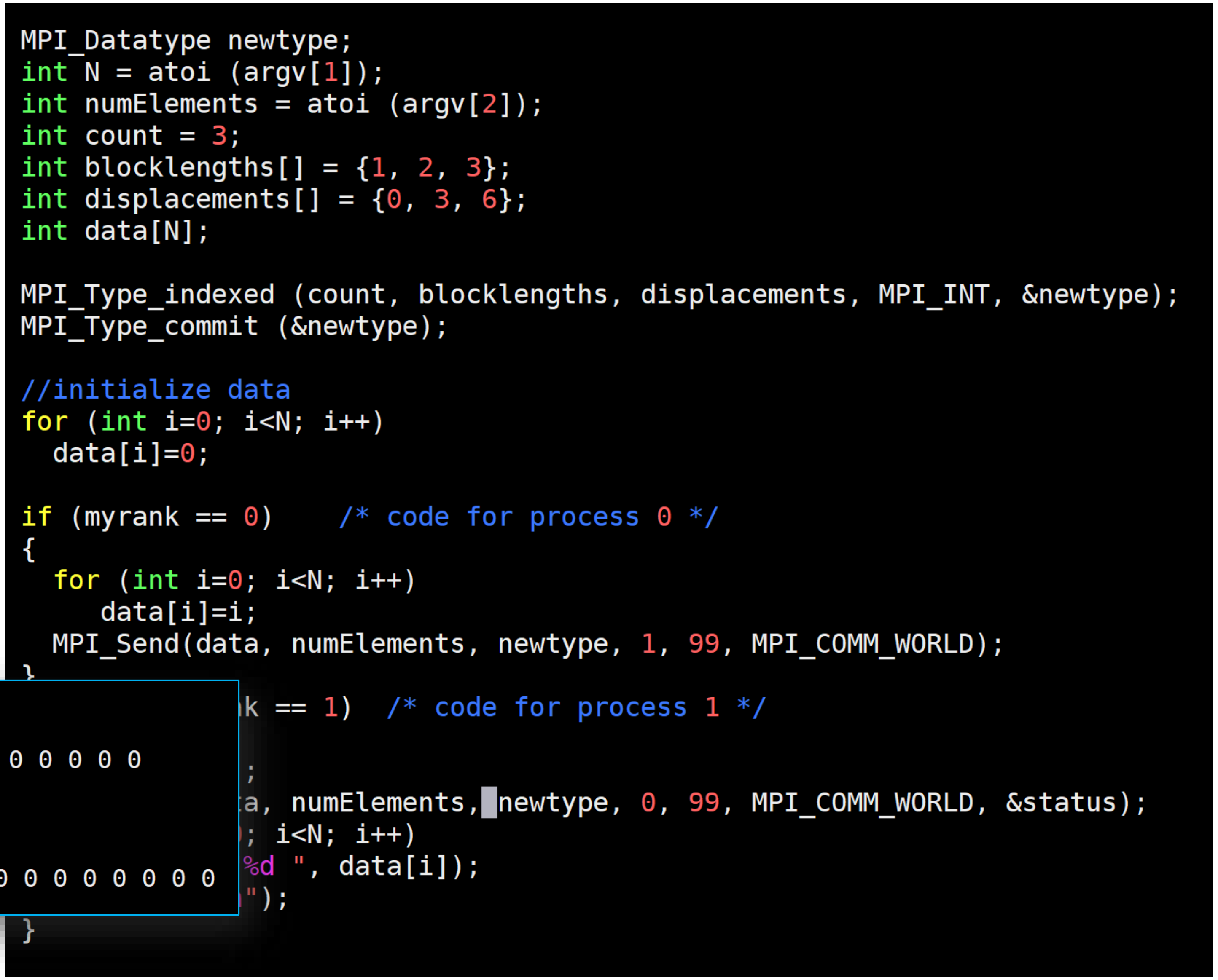
displacements = displacement of start of each block

count = 3, blocklengths = 1,2,1
displacements = 0,4,8

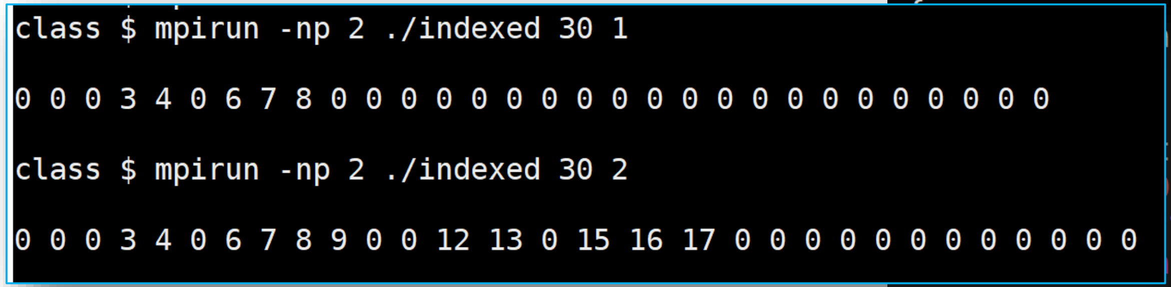
count = 3, blocklengths = 2,4,1
displacements = 0,3,8

MPI_Type_indexed (count, blocklengths, displacements, oldtype, newtype)

Code



```
MPI_Datatype newtype;
int N = atoi (argv[1]);
int numElements = atoi (argv[2]);
int count = 3;
int blocklengths[] = {1, 2, 3};
int displacements[] = {0, 3, 6};
int data[N];

MPI_Type_indexed (count, blocklengths, displacements, MPI_INT, &newtype);
MPI_Type_commit (&newtype);

//initialize data
for (int i=0; i<N; i++)
  data[i]=0;

if (myrank == 0)    /* code for process 0 */
{
  for (int i=0; i<N; i++)
     data[i]=i;
  MPI_Send(data, numElements, newtype, 1, 99, MPI_COMM_WORLD);
}
                 k == 1)  /* code for process 1 */

                 ;
                 a, numElements, newtype, 0, 99, MPI_COMM_WORLD, &status);
                 ; i<N; i++)
                 %d ", data[i]);
                 ");
}
```

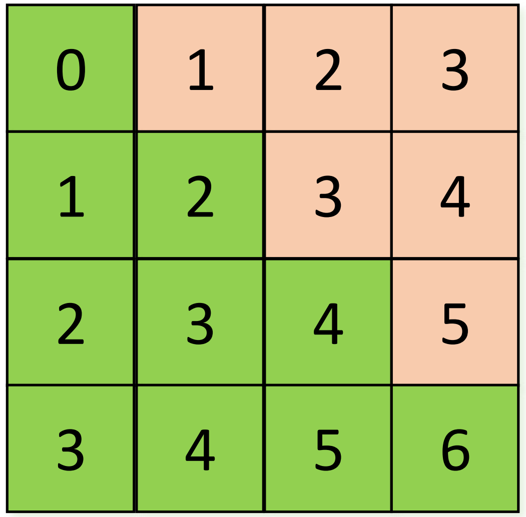


```
class $ mpirun -np 2 ./indexed 30 1

0 0 0 3 4 0 6 7 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

class $ mpirun -np 2 ./indexed 30 2

0 0 0 3 4 0 6 7 8 9 0 0 12 13 0 15 16 17 0 0 0 0 0 0 0 0 0 0 0 0
```

Lower Triangular matrix

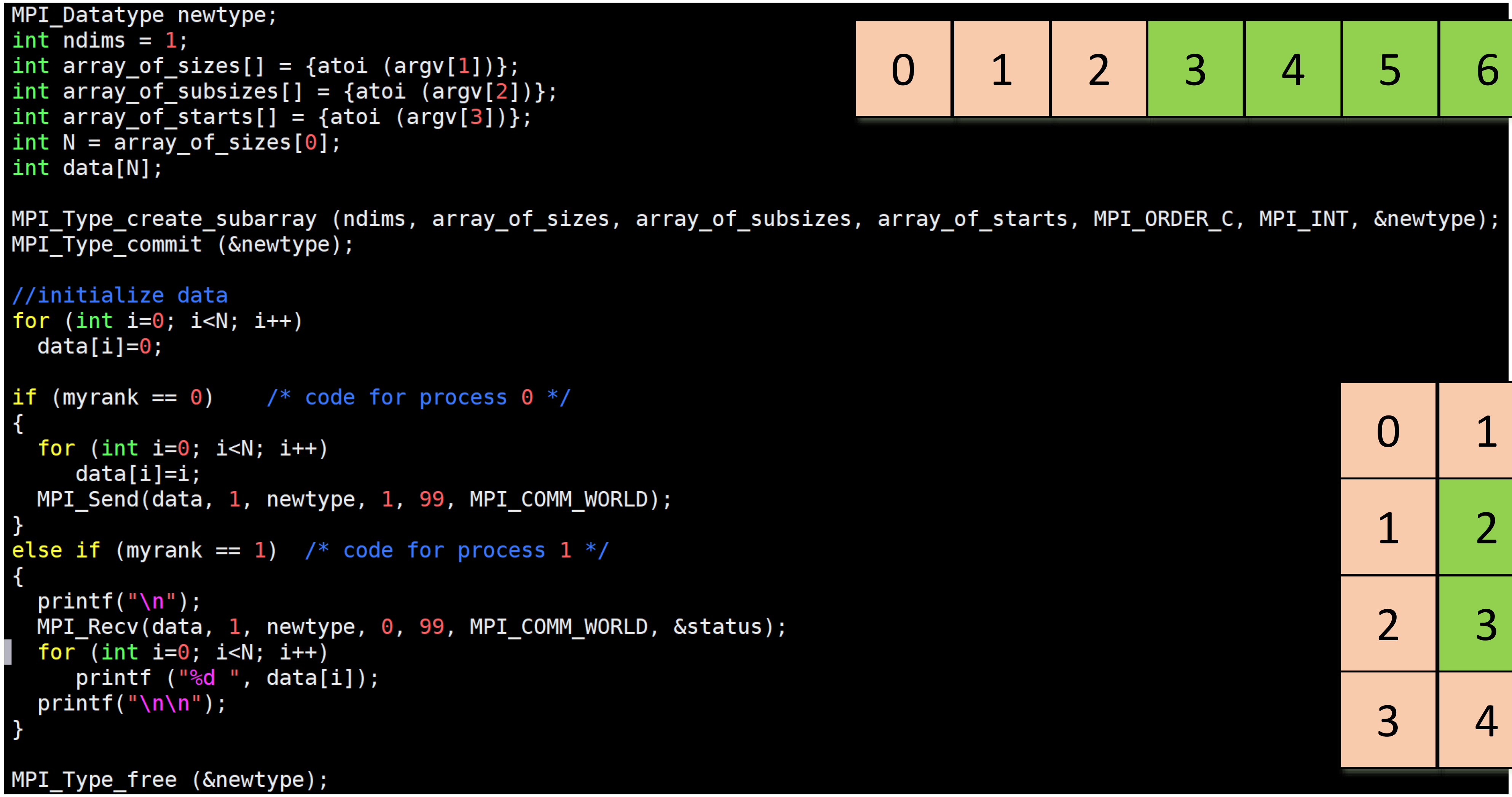
```c
int N = atoi (argv[1]);
int data[N][N];
int count = N;
int blocklengths[N], displacements[N];

for (int i=0; i<N; i++)
  blocklengths[i] = i+1, displacements[i] = i*N;
MPI_Type_indexed (count, blocklengths, displacements, MPI_INT, &newtype);
MPI_Type_commit (&newtype);

//initialize data
for (int i=0; i<N; i++)
 for (int j=0; j<N; j++)
  data[i][j]=0;

if (myrank == 0)    /* code for process 0 */
{
  for (int i=0; i<N; i++)
   for (int j=0; j<N; j++)
    data[i][j]=i+j;
  MPI_Send(data, 1, newtype, 1, 99, MPI_COMM_WORLD);
}
```

```
mpirun -np 2 ./indexed2D 4
```

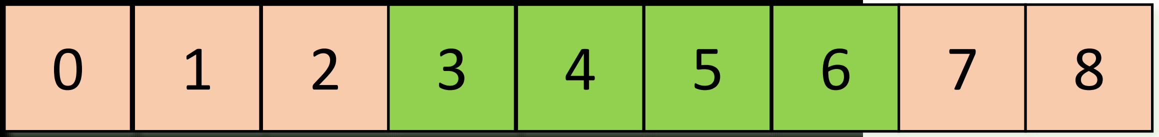


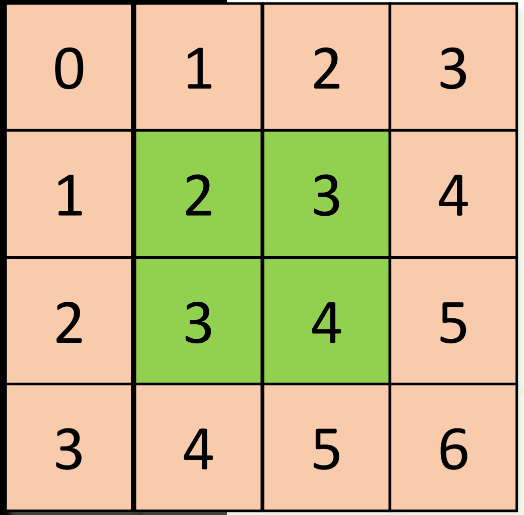
0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

MPI_Type_create_subarray

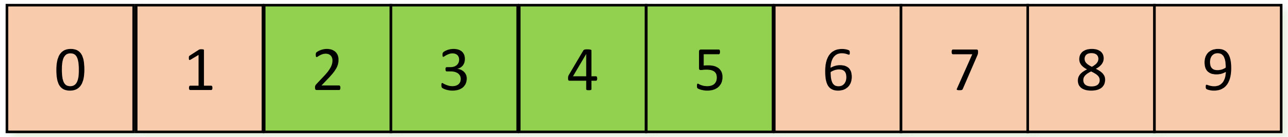


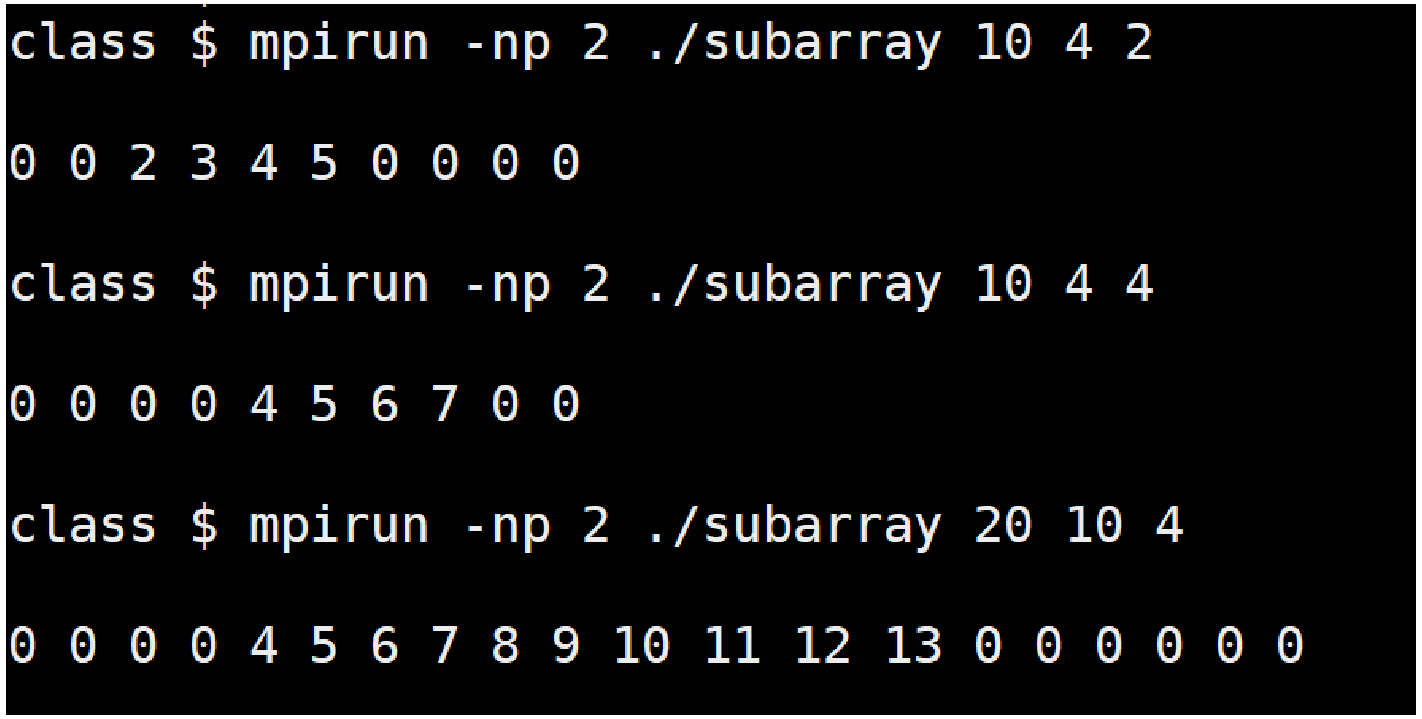
```
MPI_Datatype newtype;
int ndims = 1;
int array_of_sizes[] = {atoi (argv[1])};
int array_of_subsizes[] = {atoi (argv[2])};
int array_of_starts[] = {atoi (argv[3])};
int N = array_of_sizes[0];
int data[N];

MPI_Type_create_subarray (ndims, array_of_sizes, array_of_subsizes, array_of_starts, MPI_ORDER_C, MPI_INT, &newtype);
MPI_Type_commit (&newtype);

//initialize data
for (int i=0; i<N; i++)
  data[i]=0;

if (myrank == 0)    /* code for process 0 */
{
  for (int i=0; i<N; i++)
     data[i]=i;
  MPI_Send(data, 1, newtype, 1, 99, MPI_COMM_WORLD);
}
else if (myrank == 1)  /* code for process 1 */
{
  printf("\n");
  MPI_Recv(data, 1, newtype, 0, 99, MPI_COMM_WORLD, &status);
  for (int i=0; i<N; i++)
     printf ("%d ", data[i]);
  printf("\n\n");
}

MPI_Type_free (&newtype);
```





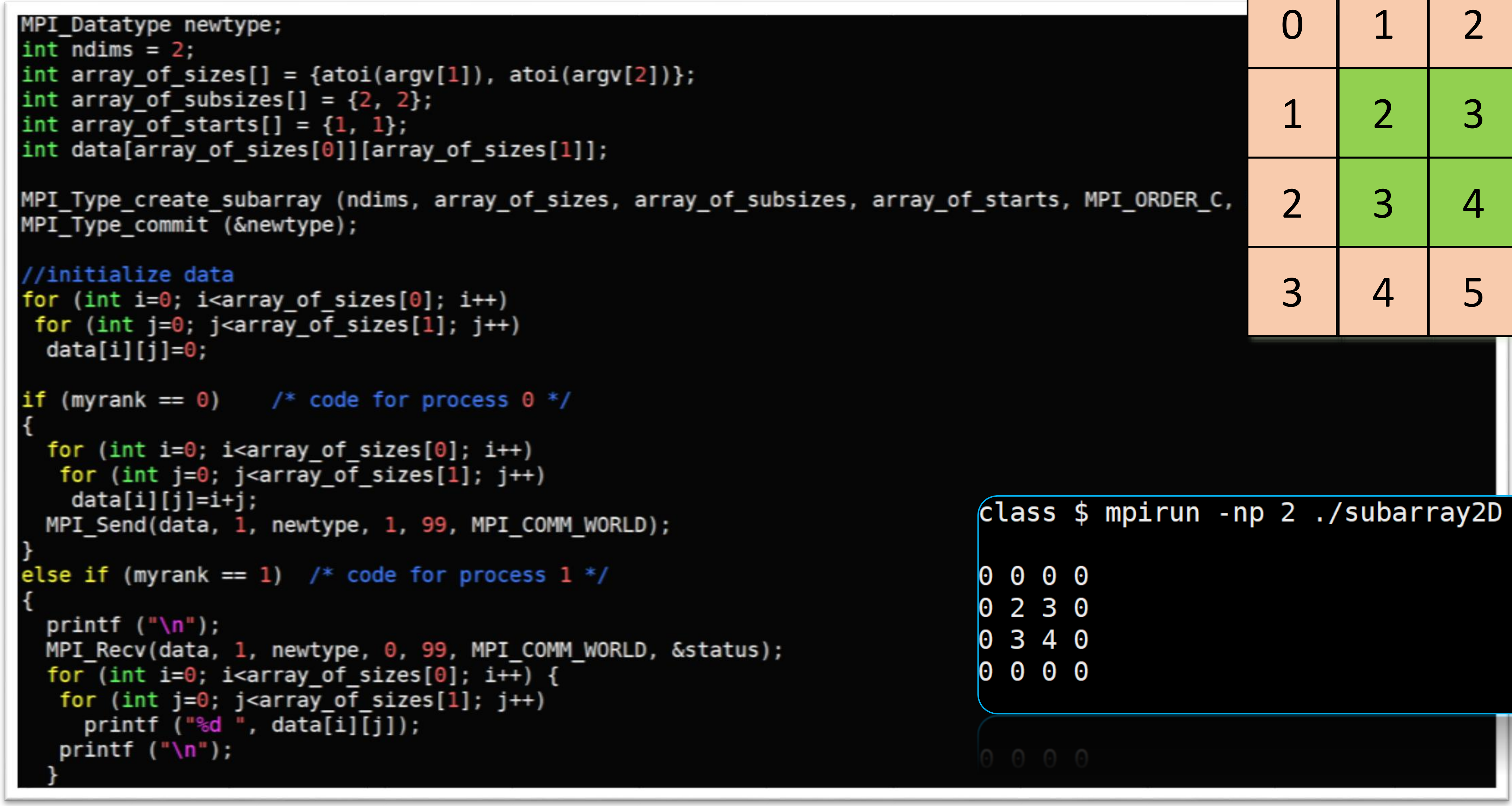
Output (Array dimension=1)



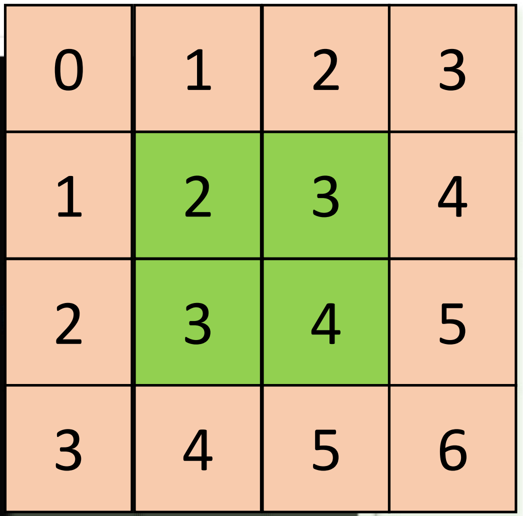


```
class $ mpirun -np 2 ./subarray 10 4 2

0 0 2 3 4 5 0 0 0 0

class $ mpirun -np 2 ./subarray 10 4 4

0 0 0 0 4 5 6 7 0 0

class $ mpirun -np 2 ./subarray 20 10 4

0 0 0 0 4 5 6 7 8 9 10 11 12 13 0 0 0 0 0 0
```

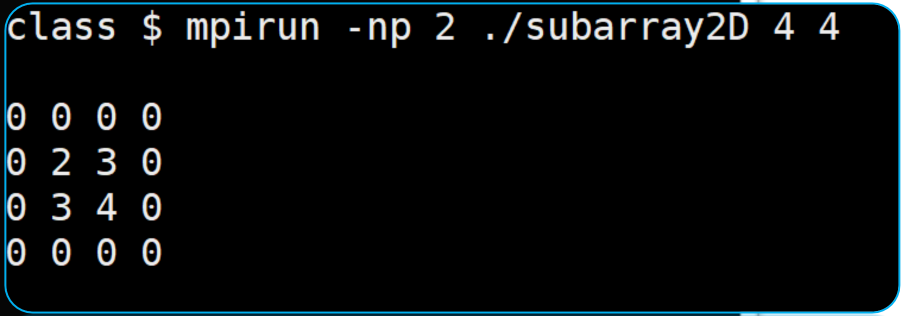
Code (2D subarray)



```c
MPI_Datatype newtype;
int ndims = 2;
int array_of_sizes[] = {atoi(argv[1]), atoi(argv[2])};
int array_of_subsizes[] = {2, 2};
int array_of_starts[] = {1, 1};
int data[array_of_sizes[0]][array_of_sizes[1]];

MPI_Type_create_subarray (ndims, array_of_sizes, array_of_subsizes, array_of_starts, MPI_ORDER_C,
MPI_Type_commit (&newtype);

//initialize data
for (int i=0; i<array_of_sizes[0]; i++)
 for (int j=0; j<array_of_sizes[1]; j++)
  data[i][j]=0;

if (myrank == 0)    /* code for process 0 */
{
  for (int i=0; i<array_of_sizes[0]; i++)
   for (int j=0; j<array_of_sizes[1]; j++)
    data[i][j]=i+j;
  MPI_Send(data, 1, newtype, 1, 99, MPI_COMM_WORLD);
}
else if (myrank == 1)  /* code for process 1 */
{
  printf ("\n");
  MPI_Recv(data, 1, newtype, 0, 99, MPI_COMM_WORLD, &status);
  for (int i=0; i<array_of_sizes[0]; i++) {
   for (int j=0; j<array_of_sizes[1]; j++)
     printf ("%d ", data[i][j]);
   printf ("\n");
  }
```



0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6



```
class $ mpirun -np 2 ./subarray2D 4 4

0 0 0 0
0 2 3 0
0 3 4 0
0 0 0 0
```

MPI Derived Datatypes Summary

MPI_Datatype newtype

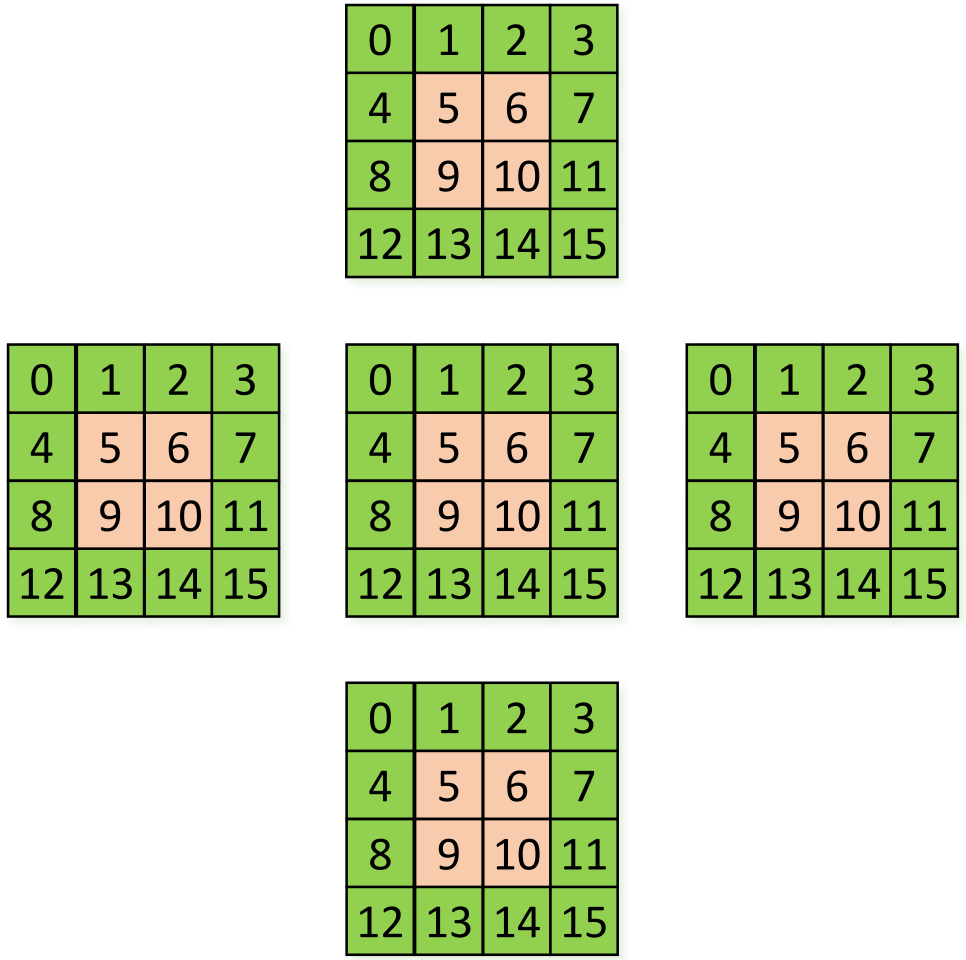
- MPI_Type_contiguous (count, oldtype, newtype)
- MPI_Type_vector (count, blocklength, stride, oldtype, newtype)
- MPI_Type_indexed (count, blocklengths, displacements, oldtype, newtype)
- MPI_Type_create_subarray (ndims, array_of_sizes, array_of_subsizes, array_of_starts, order, oldtype, newtype)
- MPI_Type_create_struct (count, array_of_blocklengths, array_of_displacements, array_of_types, newtype)

MPI_Type_commit (newtype)

……

MPI_Type_free (newtype)

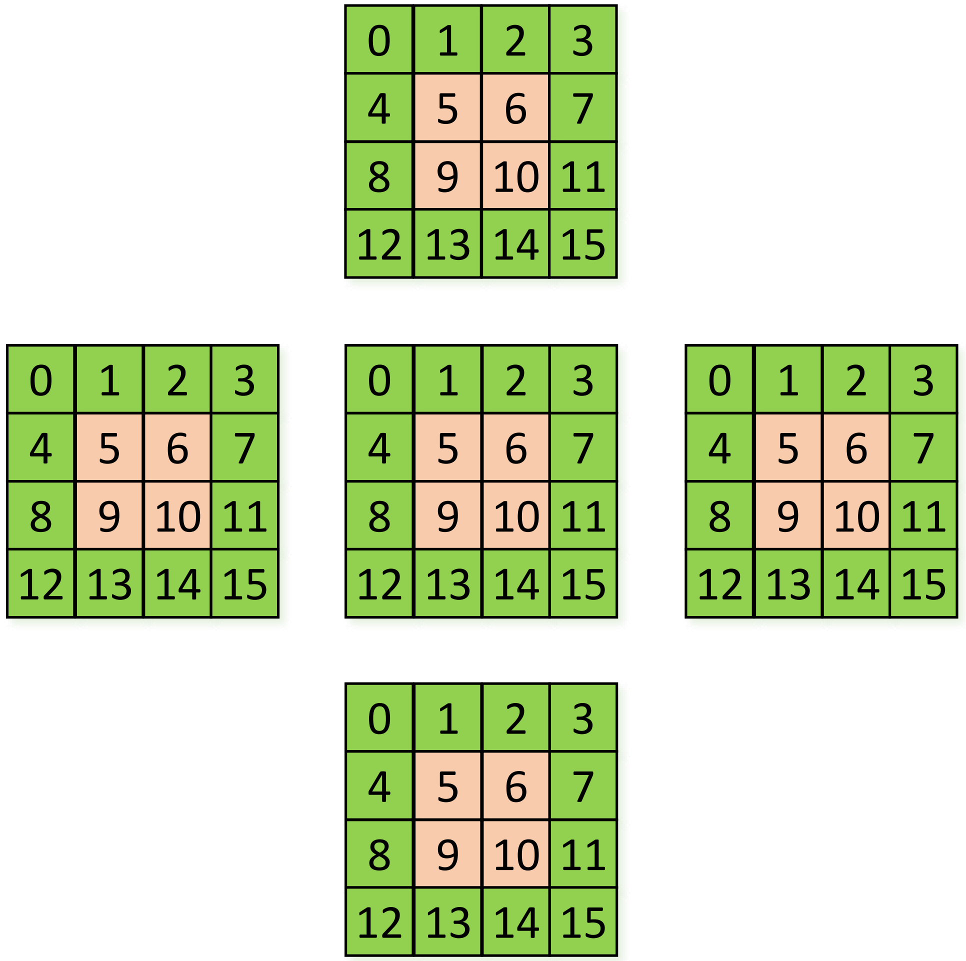
Halo Exchange



- Every time step
  - Stencil computation
    - Val_{t+1} = Average of Val_t 4 neighboring points
  - Communicate halo regions

- Multiple MPI_Sends
- MPI_Pack + MPI_Send
- MPI Derived datatype + MPI_Send

Assignment 1



Compare the three methods

- Multiple MPI_Sends
- MPI_Pack + MPI_Send
- MPI Derived datatype + MPI_Send

- Send and Recv (Blocking/non-blocking, any mode), Sendrecv, …

```
for P in 9, 16, 25, 36, ….
  for N_per_ps in 64, 256, 1024, ….
    mpirun –np P ./halo N_per_ps
```

General Instructions

- https://git.cse.iitk.ac.in
- Create private project named ‘CS633-2020-21-2’ and add pmalakar and TAs (cpathare, fahad, lavleshm, piprotar, mabir, samvid, sharhp, tusharag) and your assignment group member
- Give at least reporter access to the instructor and the TAs
- Create subdirectories for each assignment
  - AssignmentN for assignment #N
- Run your code on CSE lab cluster
  - Notify us if you find most of these are unreachable
  - Do not hard code host names (dynamically generate)
  - Optionally use NodeAllocator

Assignment1 Logistics

- Directory named Assignment1 (not assignment, Assignment- etc.) on git
- Reporter access at least (not guest)
- Send a submission email to me and the TAs as soon as you submit
  - [CS633] Assignment1
- Follow the instructions regarding file names VERY carefully
- The necessary files should not reside inside subsubdirectories
- Your 'job script' should be executable and exactly in the format/location as specified
- Plots should always be boxplots

Assignment Logistics

- Grading based on your effort as well
- Credit for early submissions (+5 / day)
  - Max credit: +15 / assignment
- A total of 2 extra days may be taken
- Score reduction for late submissions (-5 / day)
  - Max 3 late days / assignment
- Indicate in your submission email how many early/extra/late days you have used
- None of the assignments can be completed in a day!
- Start coding early
  - Parallel programs are harder to debug!