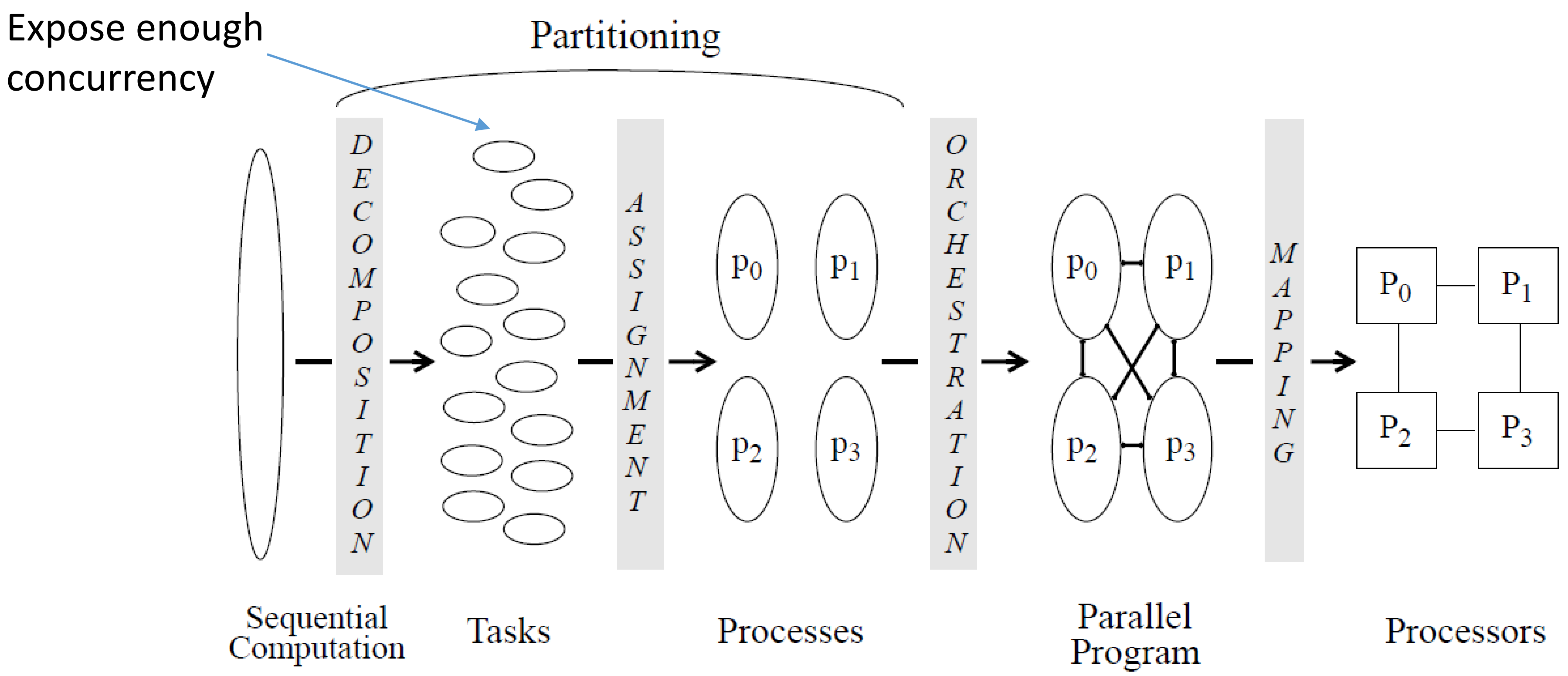
Parallelization (recap)
Derived Datatypes
Vector Variants

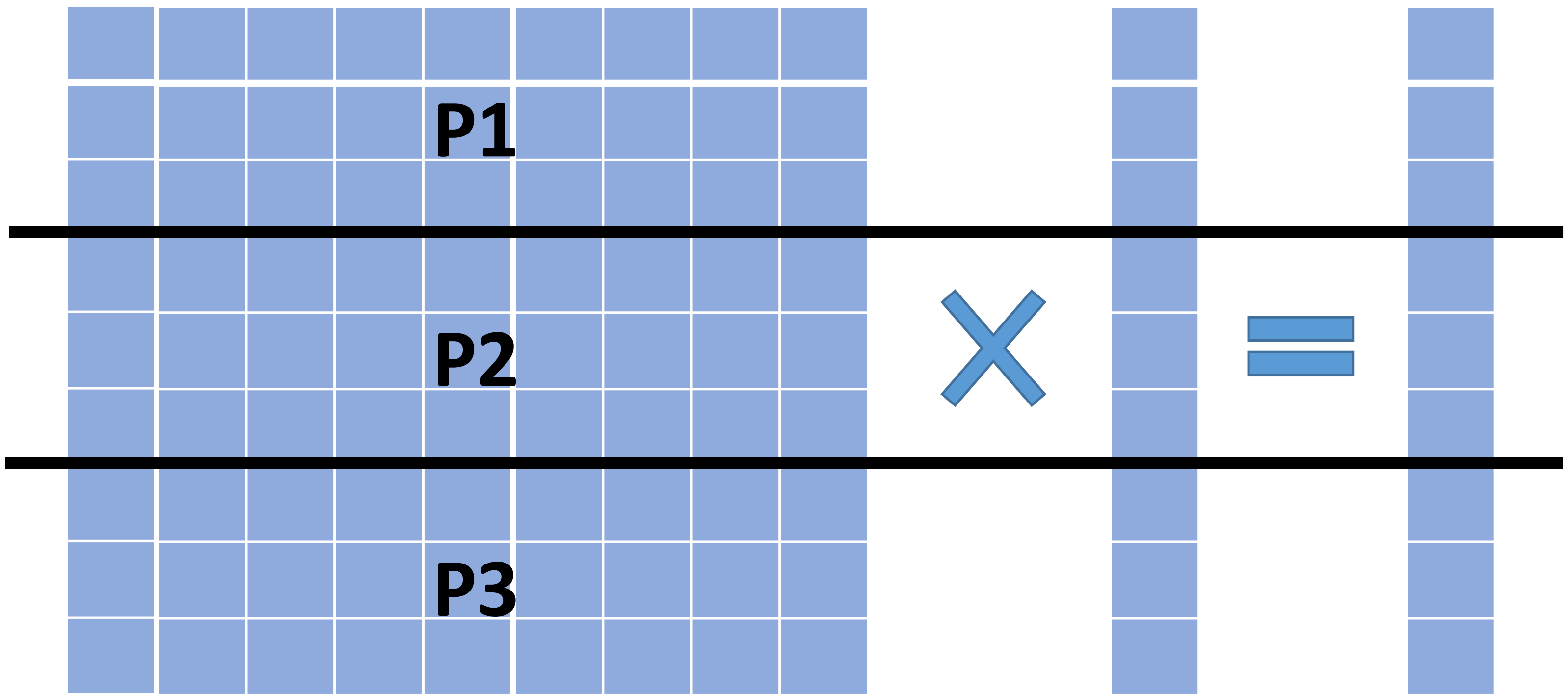
Lecture 13
March 4, 2024

Parallelization Steps

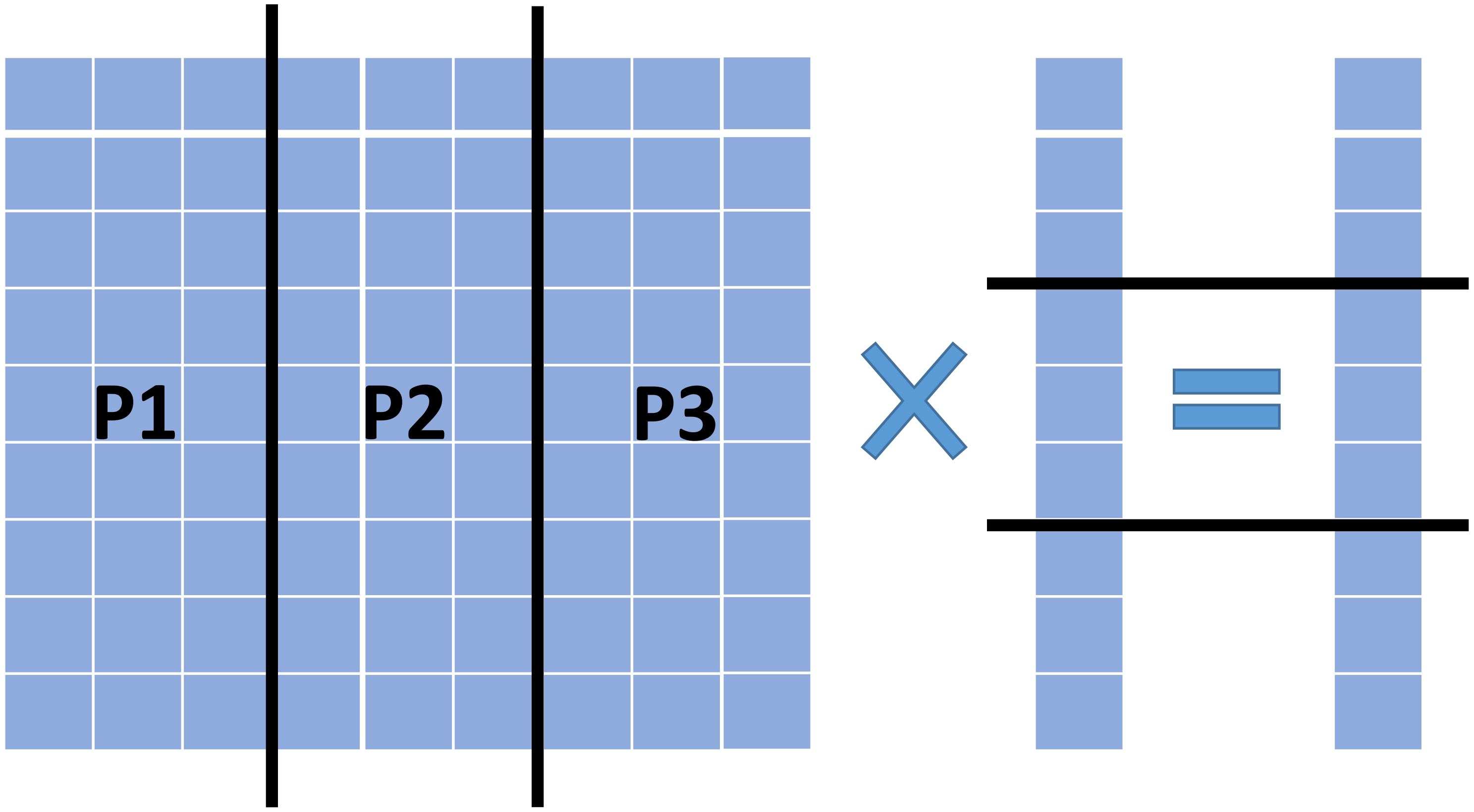


Source: Culler et al.

Matrix Vector Multiplication – Row-wise Decomposition

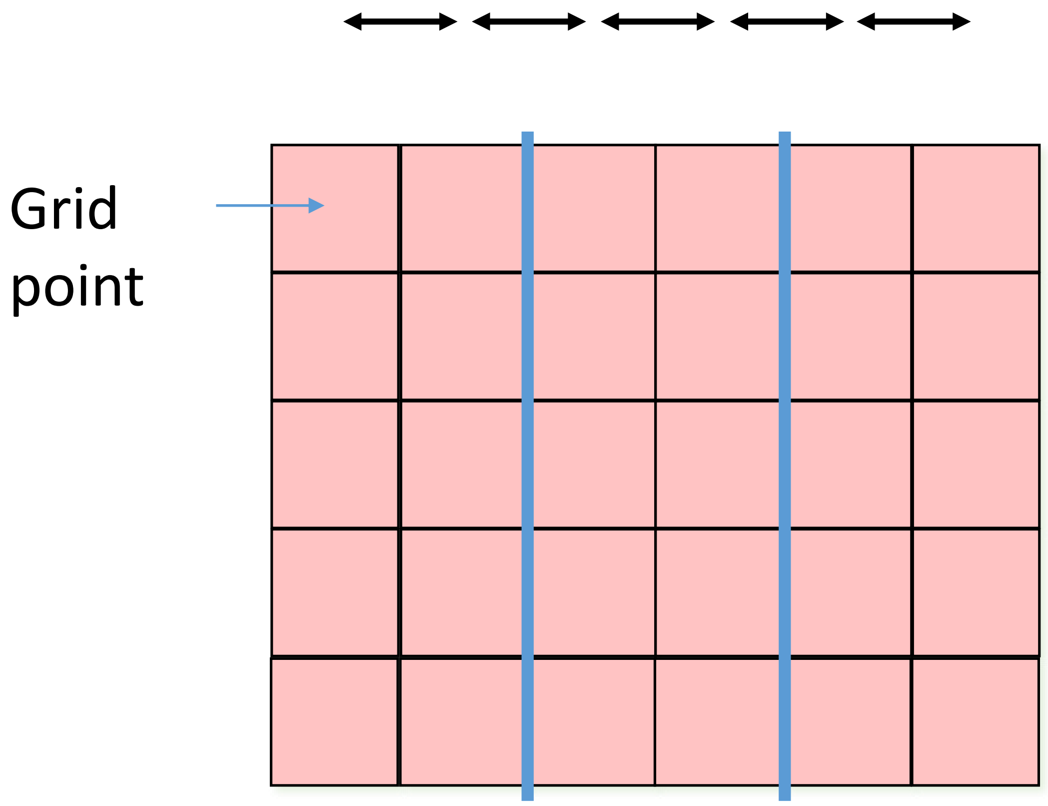


Matrix Vector Multiplication – Column-wise Decomposition



Row-wise vs. column-wise partitioning

1D Domain Decomposition



2D domain

N grid points
P processes
N/P points per process

#Communications?

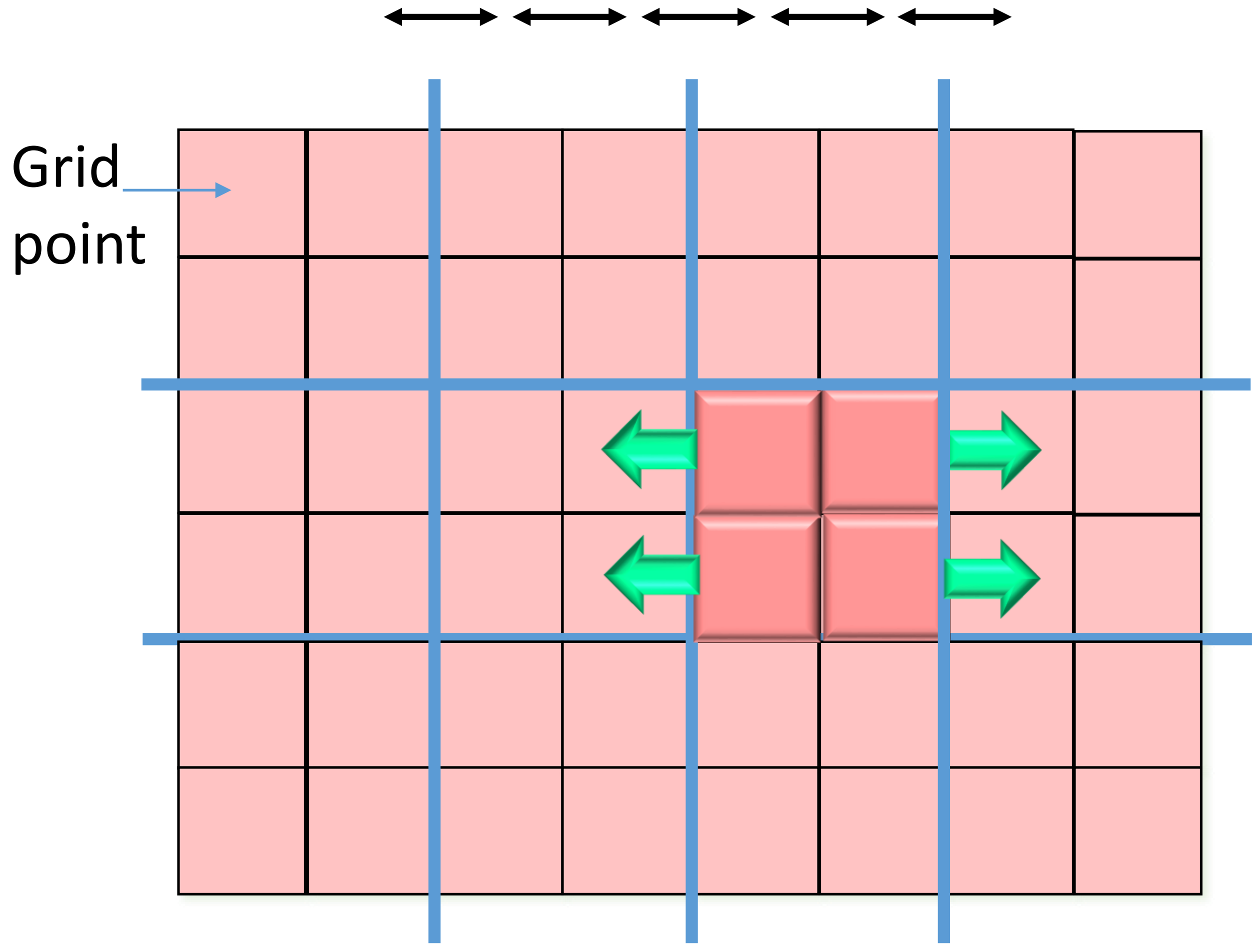
2√N (assuming square grid)

#Computations?

N/P (assuming square grid)

Communication to computation ratio=?

2D Domain decomposition



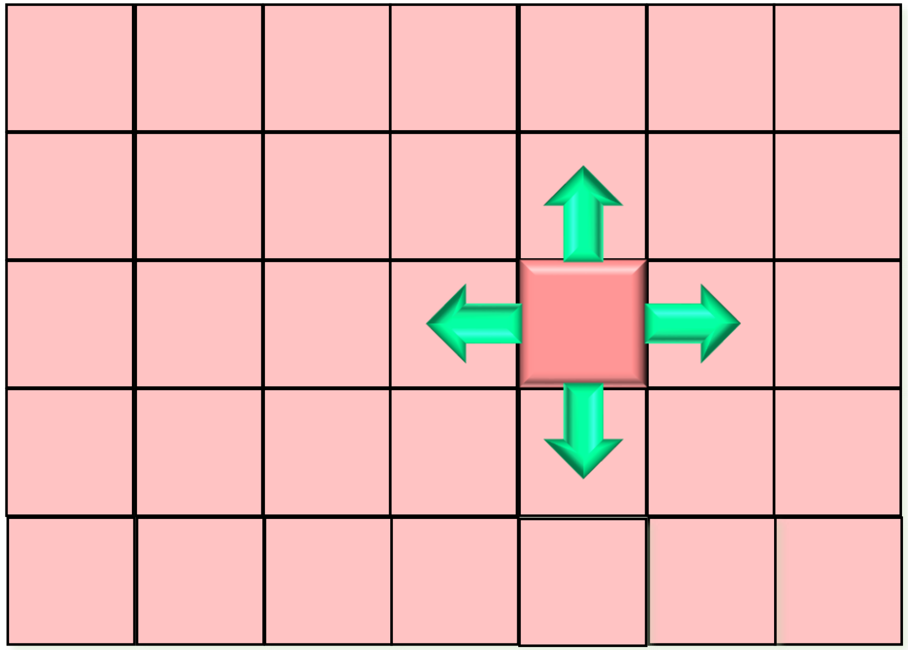
2 Sends()
2 Recvs()

#Communications?
$2\sqrt{N}/\sqrt{P}$ (assuming square grid)

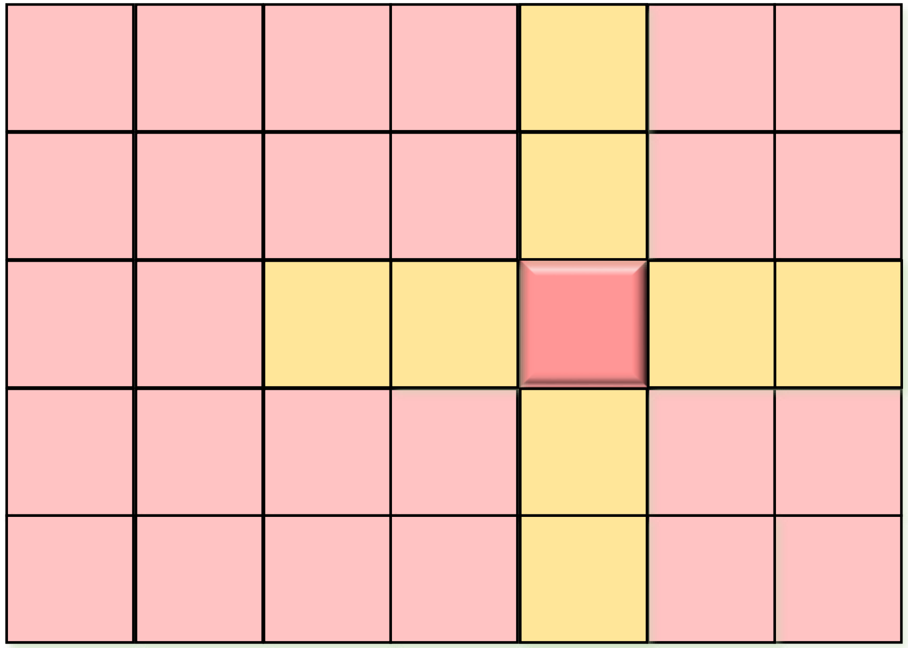
#Computations?
N/P (assuming square grid)

Communication to computation ratio=?

Stencils

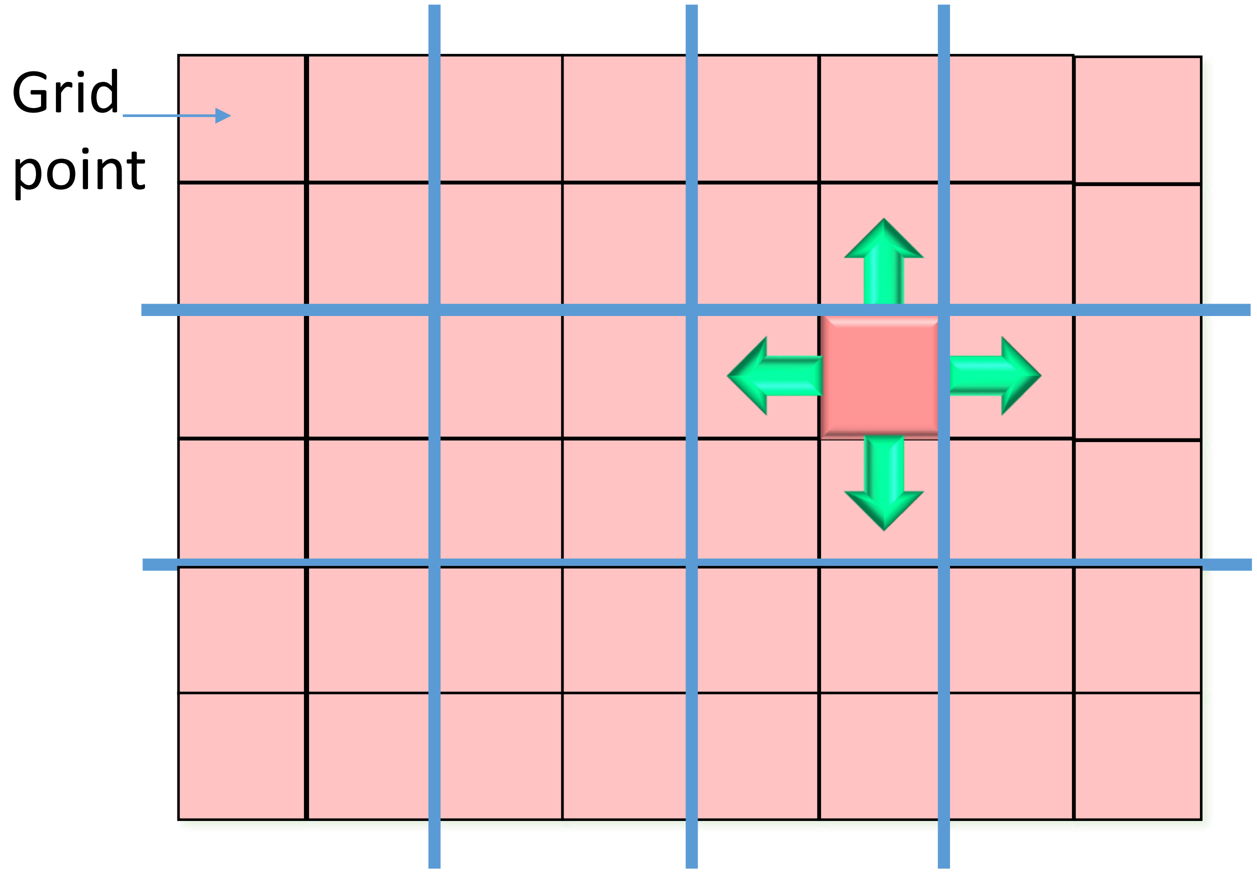


Five-point stencil



Nine-point stencil

2D Domain decomposition



4 Sends()
4 Recvs()

N grid points ($\sqrt{N}$ x $\sqrt{N}$ grid)
P processes ($\sqrt{P}$ x $\sqrt{P}$ grid)
N/P points per process

#Communications?

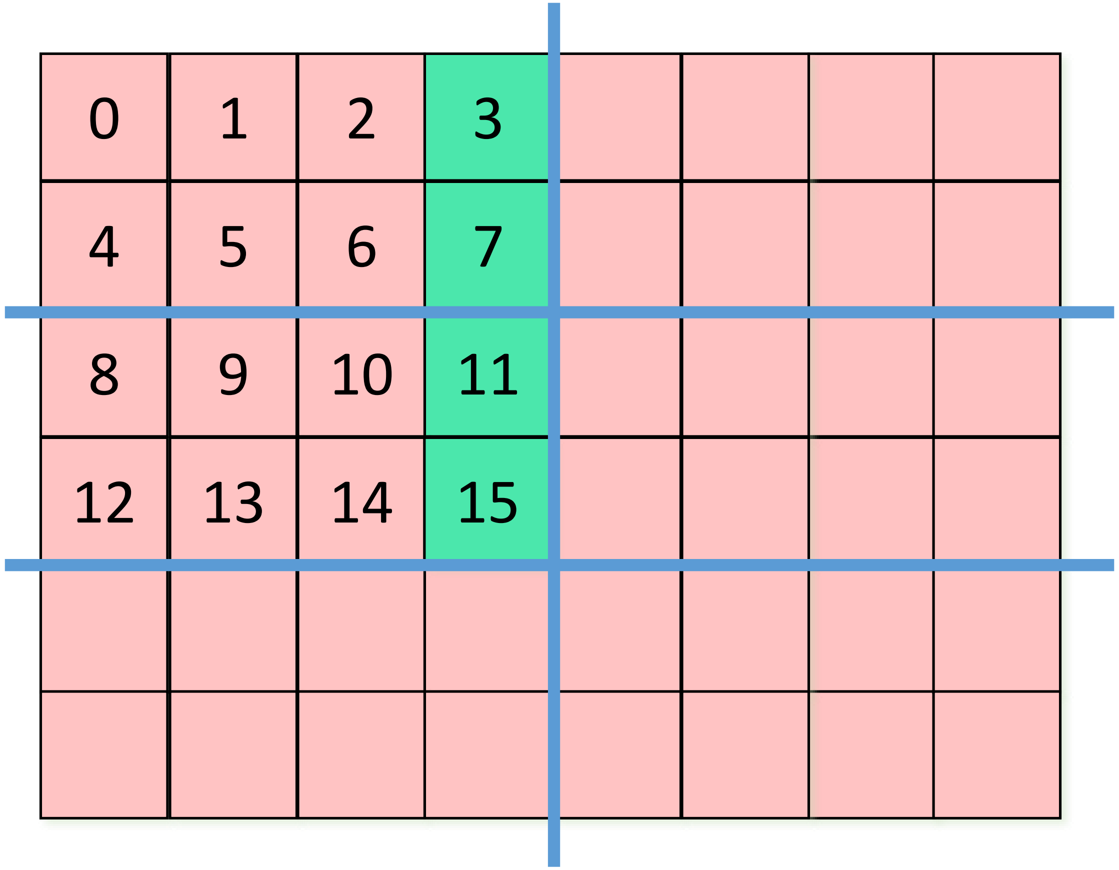
$4\sqrt{N}/\sqrt{P}$ (assuming square grid)

#Computations?

N/P (assuming square grid)

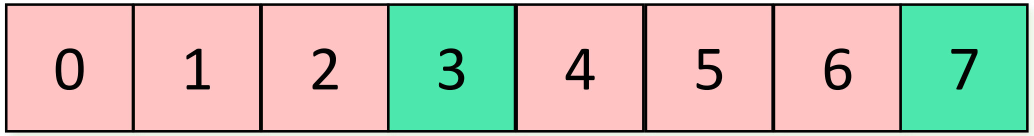
Communication to computation ratio=?

Send / Recv

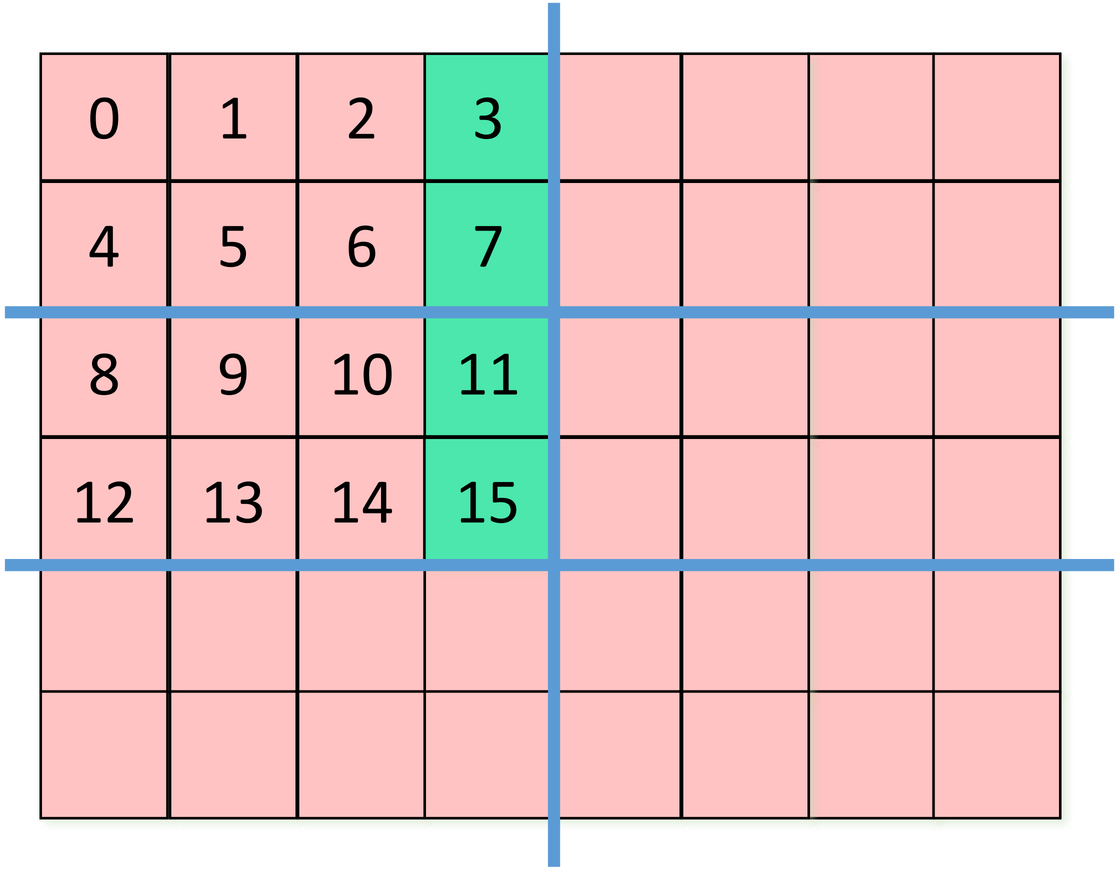


MPI_Send

MPI_Recv

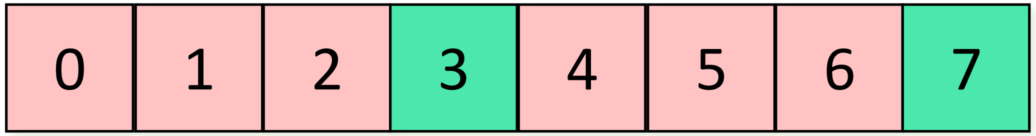


Send / Recv



MPI_Pack (buf)
MPI_Send (buf)

MPI_Recv (buf)
MPI_Unpack (buf)



MPI_Pack

int MPI_Pack (const void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outsize, int *position, MPI_Comm comm)

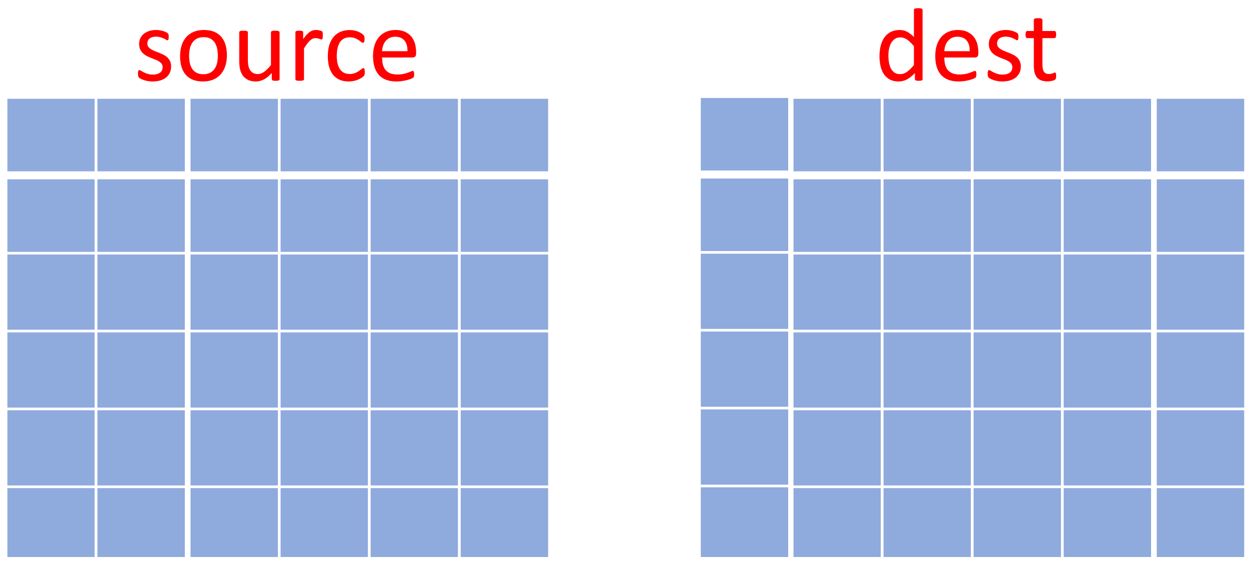
```
MPI_Pack (&num1, 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Pack (&num2, 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Send (buffer, position, MPI_PACKED, dest, 0, MPI_COMM_WORLD);


MPI_Recv (recvbuf, 2, MPI_INT, source, 0, MPI_COMM_WORLD)
```

MPI_Unpack

int MPI_Unpack (const void *inbuf, int insize, int *position, void *outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)

```
for (int r=0; r<6; r++)
   MPI_Pack (&array[r][5], 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Send (buffer, position, MPI_PACKED, dest, 0, MPI_COMM_WORLD);

MPI_Recv (recvBuf, count, MPI_PACKED, source, 0, MPI_COMM_WORLD, &status);
for (int r=0; r<6; r++)
   MPI_Unpack (recvBuf, 1000, &position, &recvArr[r][0], 1, MPI_INT, MPI_COMM_WORLD);
```

MPI_PACK Example




```
for (int r=0; r<6; r++)
  MPI_Pack (&array[r][5], 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Send (buffer, position, MPI_PACKED, dest, 0, MPI_COMM_WORLD);

MPI_Recv (recvColumn, count, MPI_INT, source, 0, MPI_COMM_WORLD, &status);
```

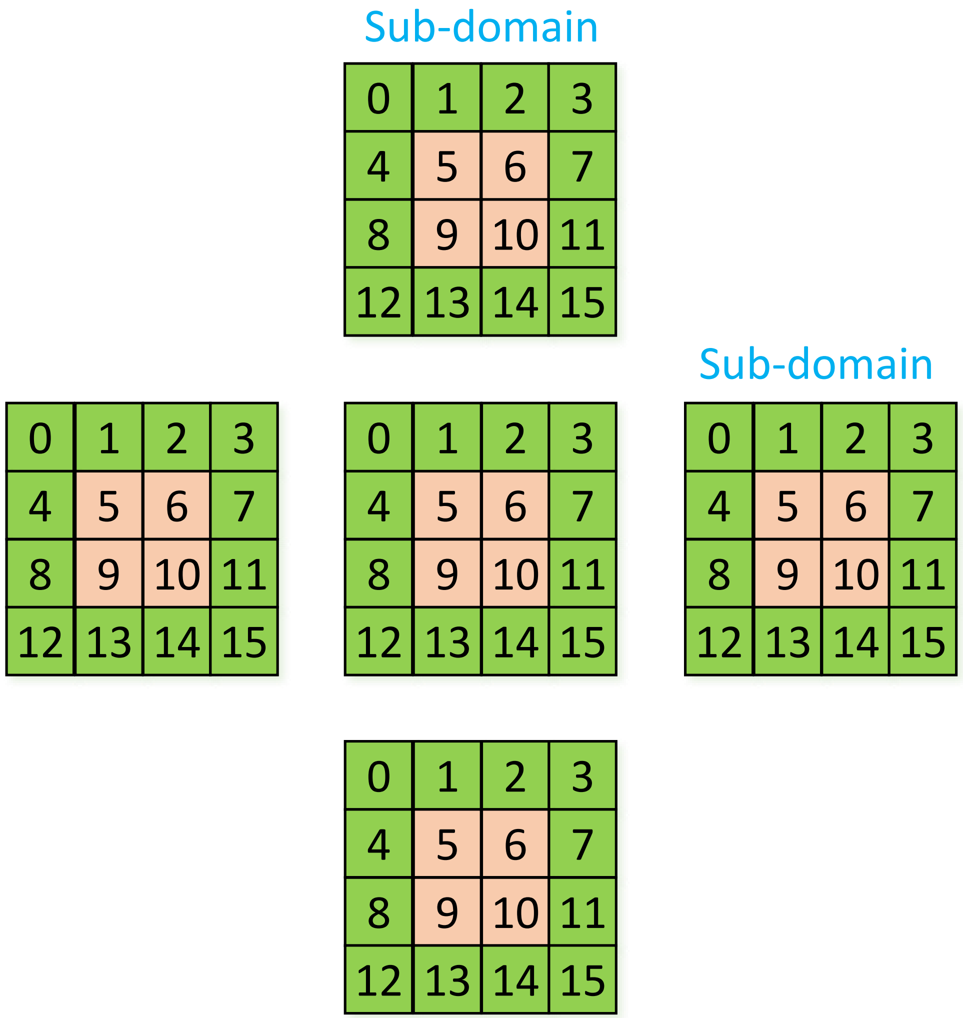
MPI_Pack

pack.c

```c
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);
```

int MPI_Pack (const void *inbuf, int incount, MPI_Datatype datatype, void **outbuf*, int outsize, int **position*, MPI_Comm comm)

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

Derived Datatypes

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

```c
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);
```

vector.c

mpirun -np 2 ./a.out 10 5 1 2 1

1 0 3 0 5 0 7 0 9 0

mpirun -np 2 ./a.out 20 5 1 2 2

1 0 3 0 5 0 7 0 9

10 0 12 0 14 0 16 0 18 0

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

?

?

vector2D.c

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Examples

```
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];
```

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

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

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

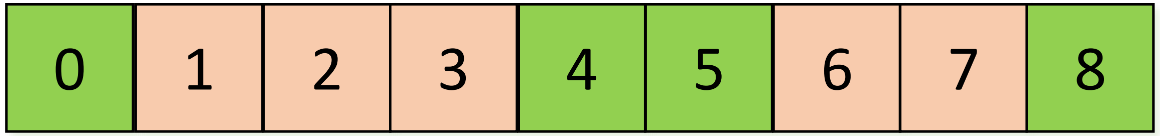
0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

MPI_Type_indexed

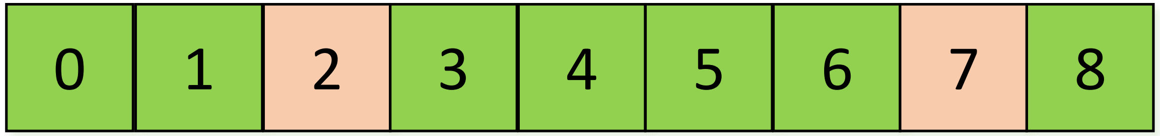
count = #blocks

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

```
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);
```

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

```
ss 1 */

99, MPI_COMM_WORLD, &status);
```

Lower Triangular Matrix

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

Summary

MPI_Datatype 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_commit (newtype)

……

MPI_Type_free (newtype)

MPI Collectives Variants

Collectives – Variable Data

- Communicate unequal amount of data to/from each process involved in the collective function call

2B		
2B		
2B		

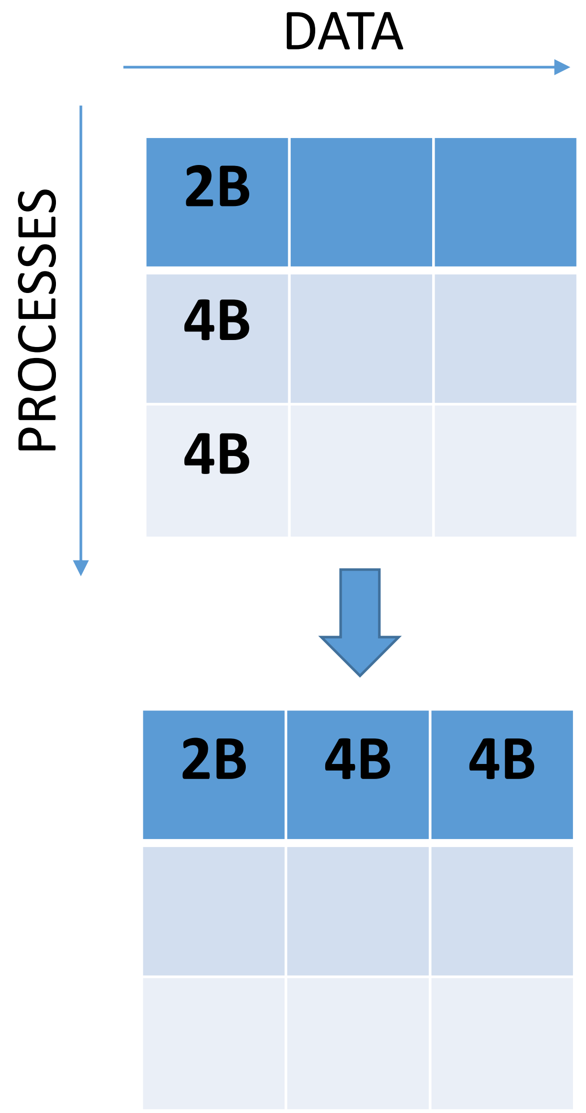
2B		
3B		
4B		

Gatherv

- Root gathers different amounts of data from the other processes
- int MPI_Gatherv (sendbuf, sendcount, sendtype, *recvbuf*, recvcounts, displs, recvtype, root, comm)
- recvcounts – Number of elements to be received from each process
- displs – Displacement at which to place received data

MPI_Recv (recvbuf+displs[i], recvcounts[i], recvtype, i, i, comm, &status) at root for i^{th} process

MPI_Send at non-root



```c
int message[arrSize];
int countArray[numtasks], displArray[numtasks];

int displ = 0;                          // note that root process is 0 here

// this information is needed by the root
if (!rank)
 for (i = 0; i < numtasks; i++) {
  countArray[i] = arrSize*(i+1);         // depends on the counts, root may need to get it from the processes
  displArray[i] = displ;
  displ += countArray[i];
  printf ("%d %d %d\n", i, countArray[i], displArray[i]);
 }

if (!rank)
  printf ("\n");

// every process initializes their local array
srand(time(NULL));
for (i = 0; i < arrSize; i++) {
  message[i] = i; // (double)rand() / (double)RAND_MAX;
}

int recvMessage[displ];    // significant at the root process

// receive different counts of elements from different processes
MPI_Gatherv (message, arrSize, MPI_INT, recvMessage, countArray, displArray, MPI_INT, 0, MPI_COMM_WORLD);

if (!rank)
 for (i = 0; i < displ; i++) {
   printf ("%d %d\n", i, recvMessage[i]);
 }
```

Demo

```
class $ mpirun -np 2 ./gatherv 2
0 2 0
1 4 2

0 0
1 1
2 0
3 1
4 2
5 3
class $ mpirun -np 3 ./gatherv 2
0 2 0
1 4 2
2 6 6

0 0
1 1
2 0
3 1
4 2
5 3
6 0
7 1
8 2
9 3
10 4
11 5
```

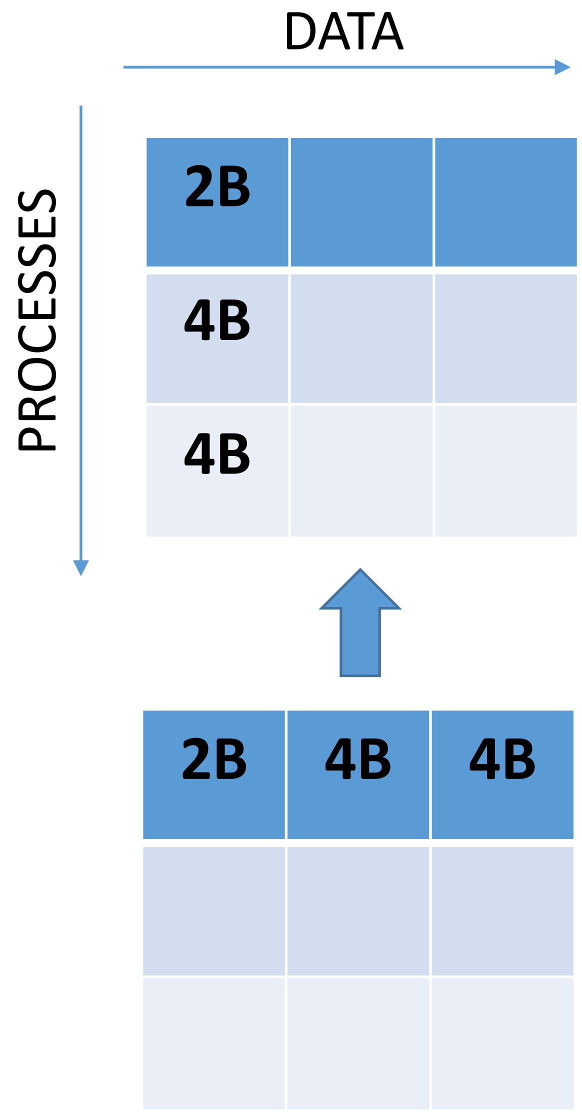
Bug

```
Fatal error in PMPI_Gatherv: Message truncated, error stack:
PMPI_Gatherv(435)......................: MPI_Gatherv failed(sbuf=0x7ffd379c4250, scount=2, MPI_INT, rbuf=0x7ffd379c4210, rcnts=0x7ffd379c4240,
 displs=0x7ffd379c4230, MPI_INT, root=0, MPI_COMM_WORLD) failed
MPIR_Gatherv_impl(235).................:
MPIR_Gatherv(151)......................:
MPIDI_CH3_PktHandler_EagerShortSend(363): Message from rank 1 and tag 4 truncated; 16 bytes received but buffer size is 8
class $ vi gatherv.c
class $ !mpicc
mpicc -o gatherv gatherv.c
class $ !mpirun
mpirun -np 2 ./gatherv 2
0 2 0
1 4 2
```

```
class $ time mpirun -np 3 ./gatherv 2

real    0m0.007s
user    0m0.004s
sys     0m0.010s
class $ time mpirun -np 30 -hosts csews4:10,csews2:10,csews3:10 ./gatherv 200

real    0m0.695s
user    0m0.026s
sys     0m0.006s
class $ time mpirun -np 30 -hosts csews4:10,csews2:10,csews3:10 ./gatherv 20000

real    0m1.857s
user    0m0.033s
sys     0m0.002s
class $ time mpirun -np 60 -hosts csews4:10,csews2:10,csews3:10,csews5:10,csews6:10,csews7:10 ./gatherv 20000

real    0m2.965s
user    0m12.571s
sys     0m1.559s
class $ time mpirun -np 60 -hosts csews4:10,csews2:10,csews3:10,csews5:10,csews6:10,csews7:10 ./gatherv 800000

real    1m5.621s
user    7m2.519s
sys     1m2.687s
class $
```

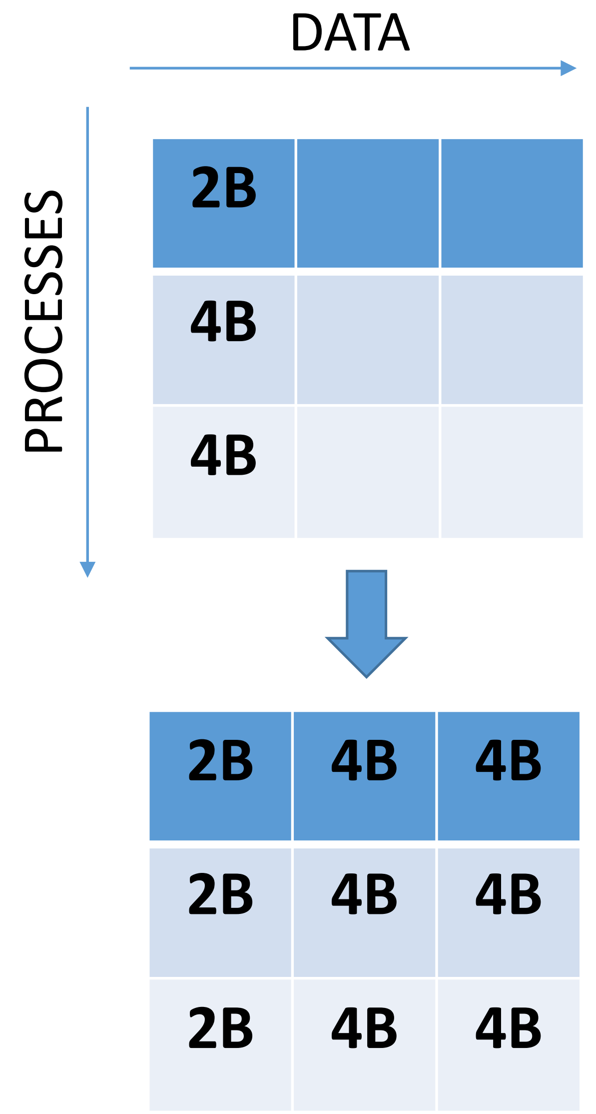
MPI_Scatterv

- Root scatters different amounts of data to the other processes
- int MPI_Scatterv (const void *sendbuf, const int *sendcounts, const int *displs, MPI_Datatype sendtype, void *_recvbuf_, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- sendcounts – Number of elements to be sent to each process
- displs – Displacement (relative to sendbuf) at which the data to be sent resides



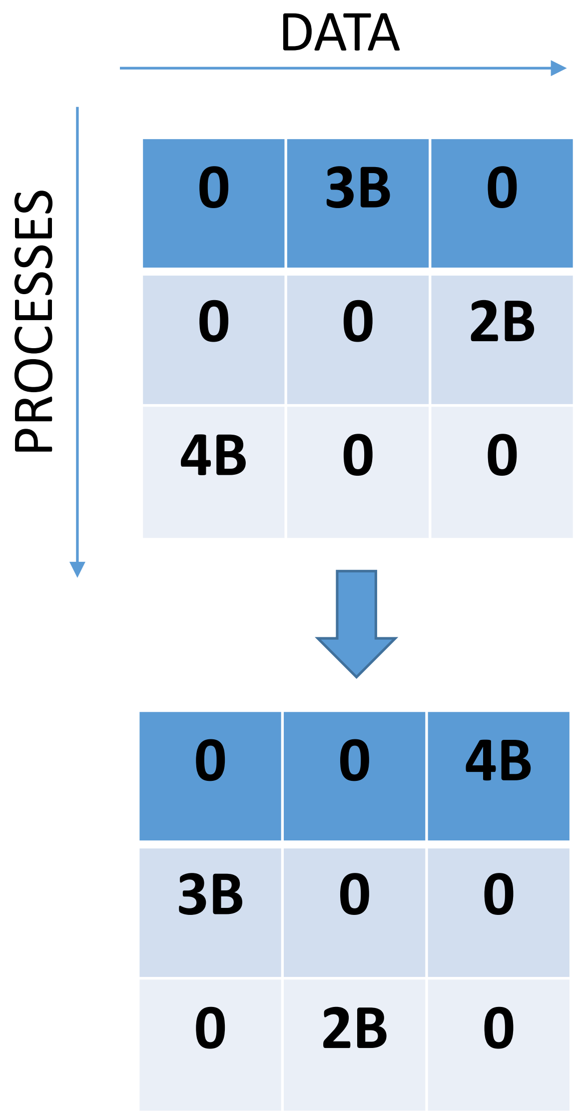
Allgatherv

- All processes gather values of different lengths from all processes
- int MPI_Allgatherv (sendbuf, sendcount, sendtype, *recvbuf*, recvcounts, displs, recvtype, comm)
- recvcounts – Number of elements to be received from each process
- displs – Displacement at which to place received data



Alltoallv

- Every process sends data of different lengths to other processes
- int MPI_Alltoallv (sendbuf, sendcount, sdispls, sendtype, *recvbuf*, recvcount, rdispls, recvtype, comm)
- Output parameter – recvbuf
- It’s not necessary to receive some data from all processes, i.e. some entries of count and displs may be 0



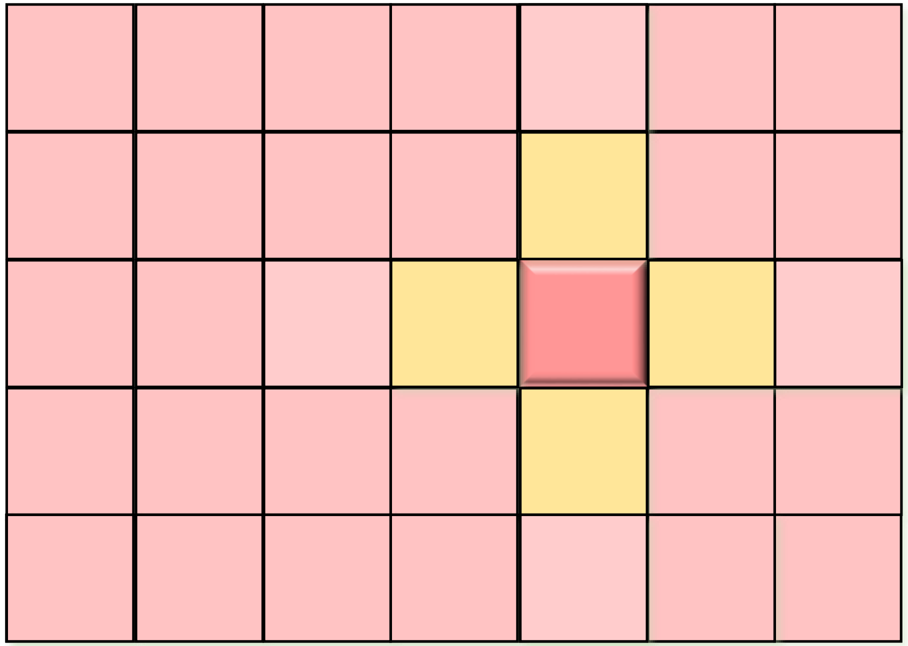
Non-blocking Collectives

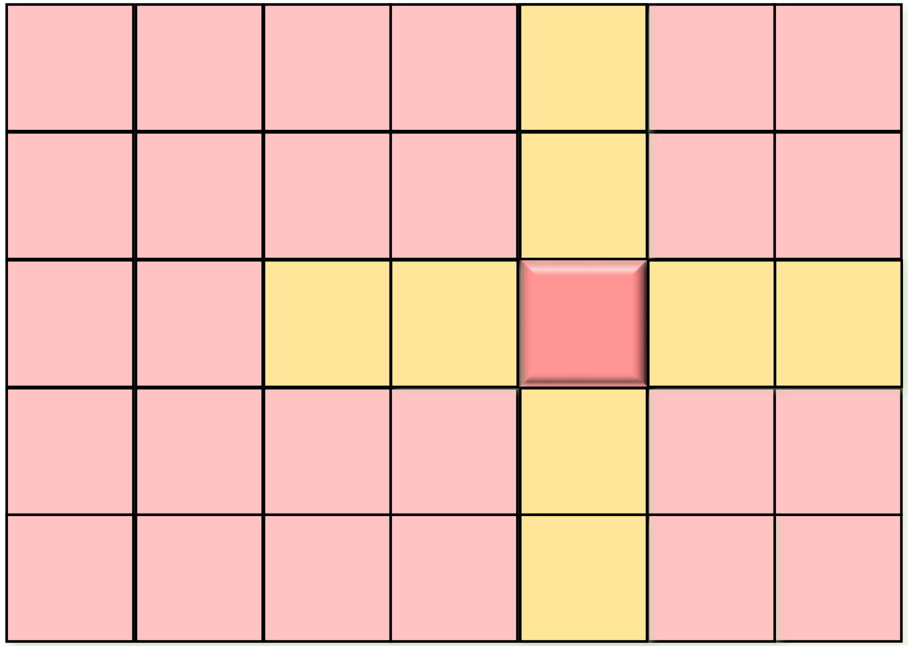
- Introduced in MPI-3
- Benefit of non-blocking point-to-point
- Overlap communication and computation
- Reduce synchronization
- Improve performance for overlapping communicators
- How do we ensure completion?
    - MPI_Wait (request, status)

Non-blocking Collectives

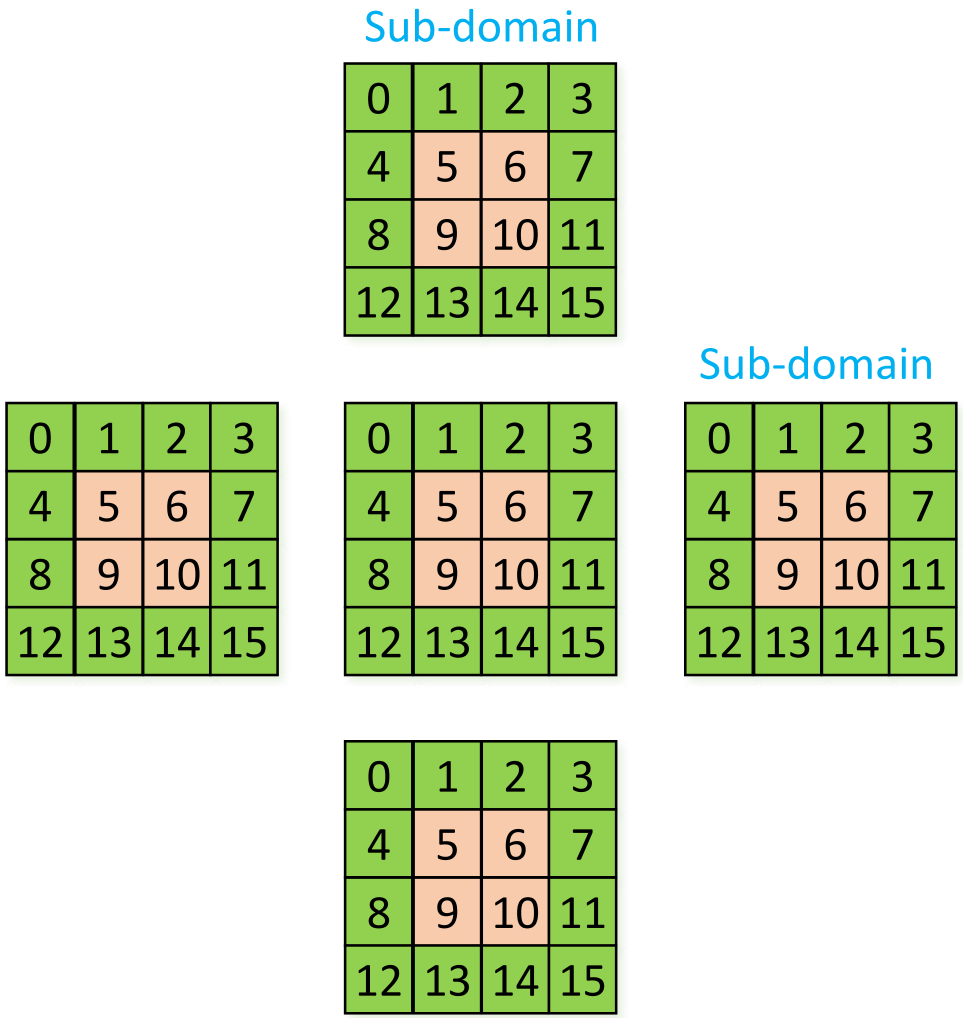
- MPI_Ibcast (buffer, count, datatype, root, comm, request)
- MPI_Igather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- MPI_Igatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, request)
- MPI_Ialltoall (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, request)
- …

Assignment 1 (Compare 5- and 9-point Stencil)





Halo Exchange (5- and 9-point Stencil)



- Every time step t
  - Communicate halo regions
    - MPI_Pack + send
    - recv + MPI_Unpack
  - Stencil computation
    - Val_{t+1} = Average of Val_t (x neighboring points) and itself

    [x= 4 or 8]