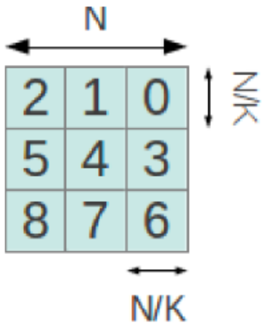
Problem 5:

Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - K % N = 0

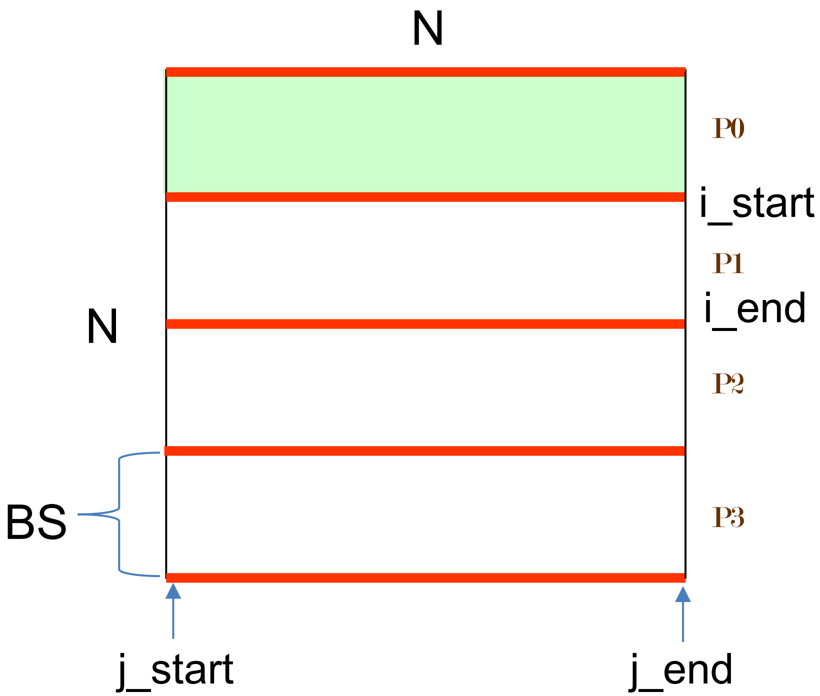



```
typedef struct {
    int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[num_threads];

void InitDecomposition(limits * decomposition, int N, int nt) { ...  }

void main (int argc, char *argv[]) {
  #pragma omp parallel
  #pragma omp single
     InitDecomposition(decomposition,N,omp_get_num_thread());
  #pragma omp parallel
  {
     ...
     int i_start = ...
     int i_end   = ...
     int j_start = ...
     int j_end   = ...
     foo(i_start,i_end,j_start,j_end);
  }
}
```

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```




```
void main (int argc, char *argv[]) {

    #pragma omp parallel
    #pragma omp single
     InitDecomposition (decomposition, N, omp_get_num_thread());

    #pragma omp parallel
    {
     …
     int i_start = …
     int i_end = …
     int j_start = …
     int j_end = …
     foo (i_start, i_end, j_start, j_end);
    }
}
```

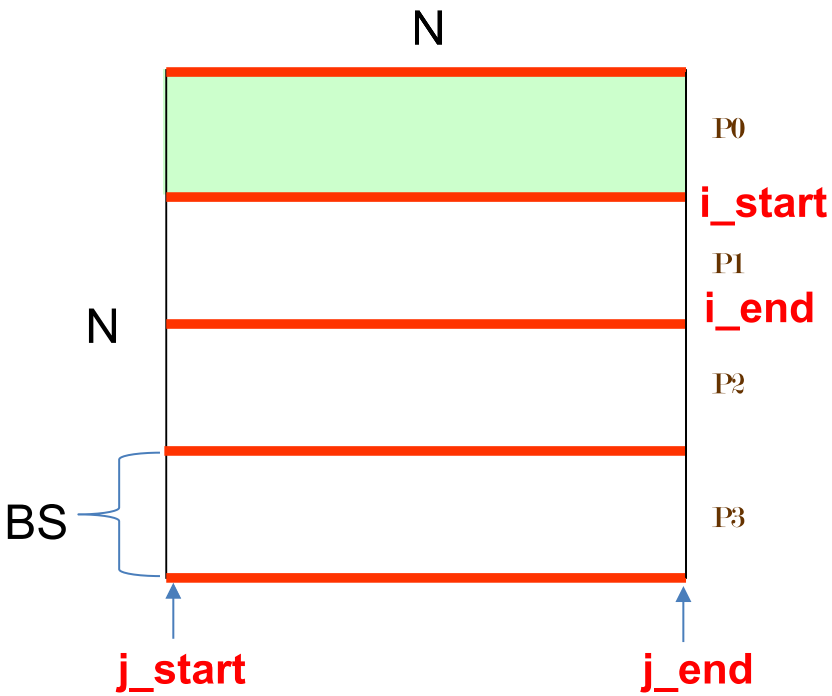
nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```




```
void main (int argc, char *argv[]) {

    #pragma omp parallel
    #pragma omp single
     InitDecomposition (decomposition, N, omp_get_num_thread());

    #pragma omp parallel
    {
     …
     int i_start = …
     int i_end = …
     int j_start = …
     int j_end = …
     foo (i_start, i_end, j_start, j_end);
    }
}
```

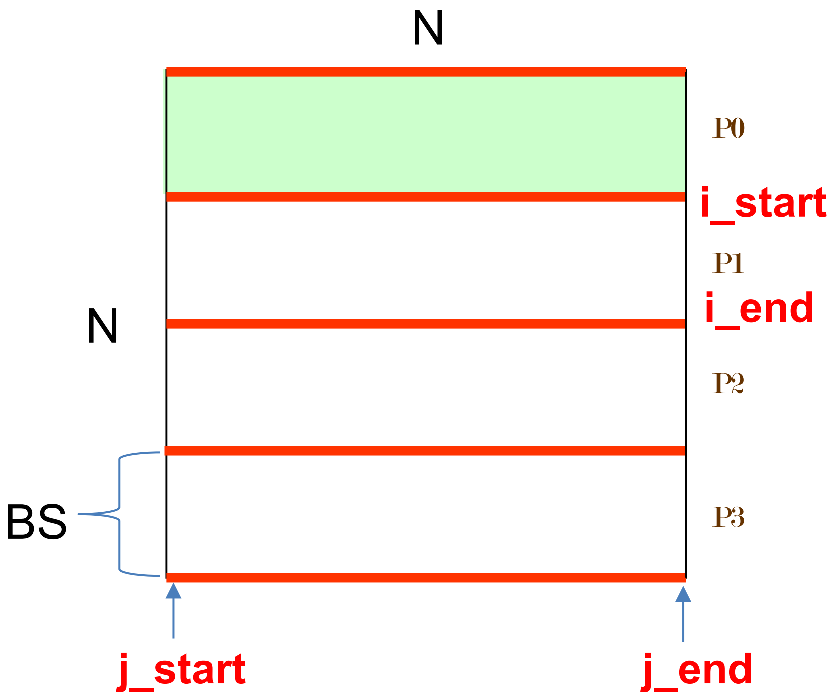
nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```




```
void main (int argc, char *argv[]) {

    #pragma omp parallel
    #pragma omp single
     InitDecomposition (decomposition, N, omp_get_num_thread());

    #pragma omp parallel
    {
     int myid = omp_get_thread_num();
     int i_start = ...
     int i_end = …
     int j_start = …
     int j_end = …
     foo (i_start, i_end, j_start, j_end);
    }
}
```

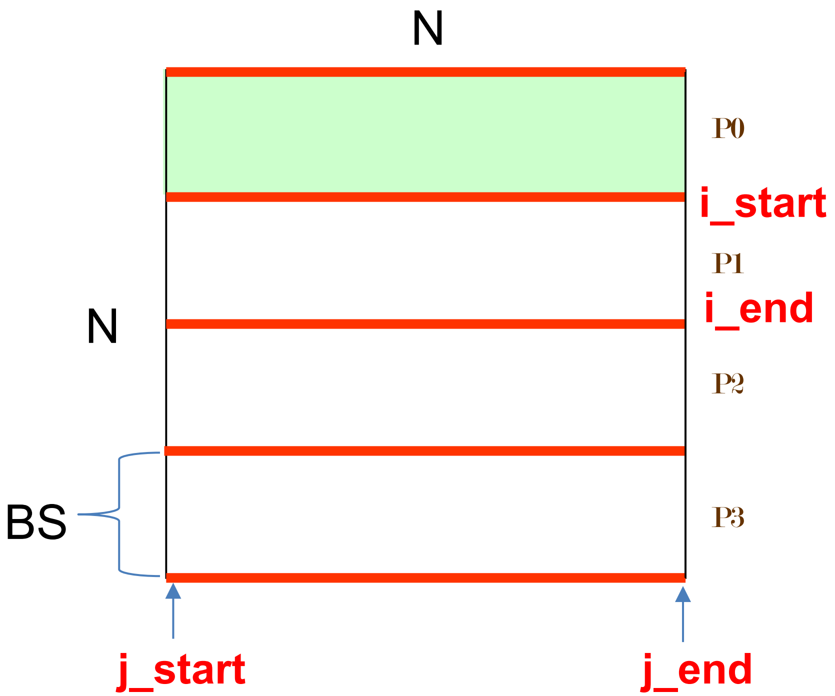
nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
    - N is not a multiple of the number of threads
    - Unbalance is at most equal to 1 row

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```



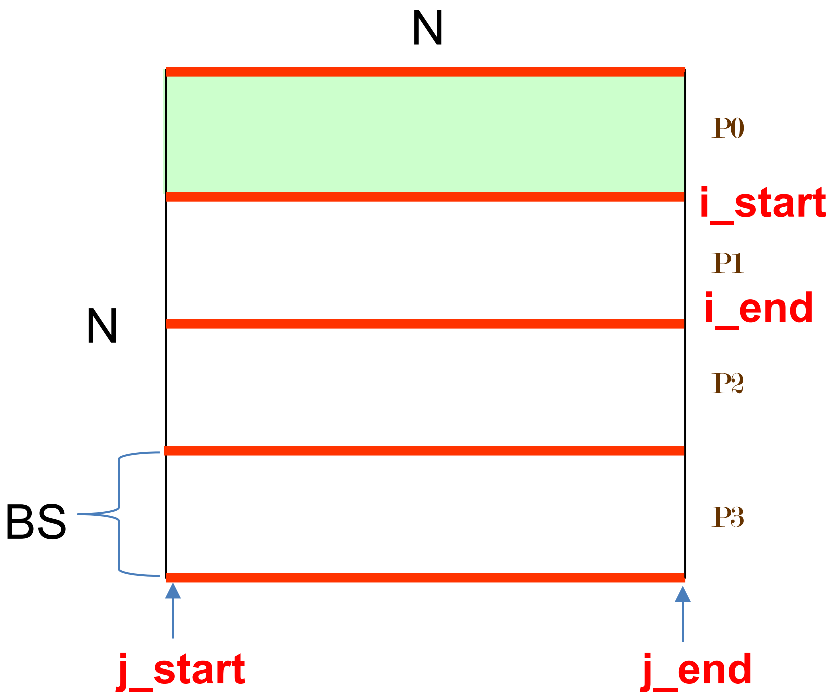

```
void main (int argc, char *argv[]) {

    #pragma omp parallel
    #pragma omp single
     InitDecomposition (decomposition, N, omp_get_num_thread());

    #pragma omp parallel
    {
     int myid = omp_get_thread_num();
     int i_start = decomposition [myid].i_start;
     int i_end = decomposition [myid].i_end;
     int j_start = decomposition [myid].j_start;
     int j_end = decomposition [myid].j_end;
     foo (i_start, i_end, j_start, j_end);
    }
}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row



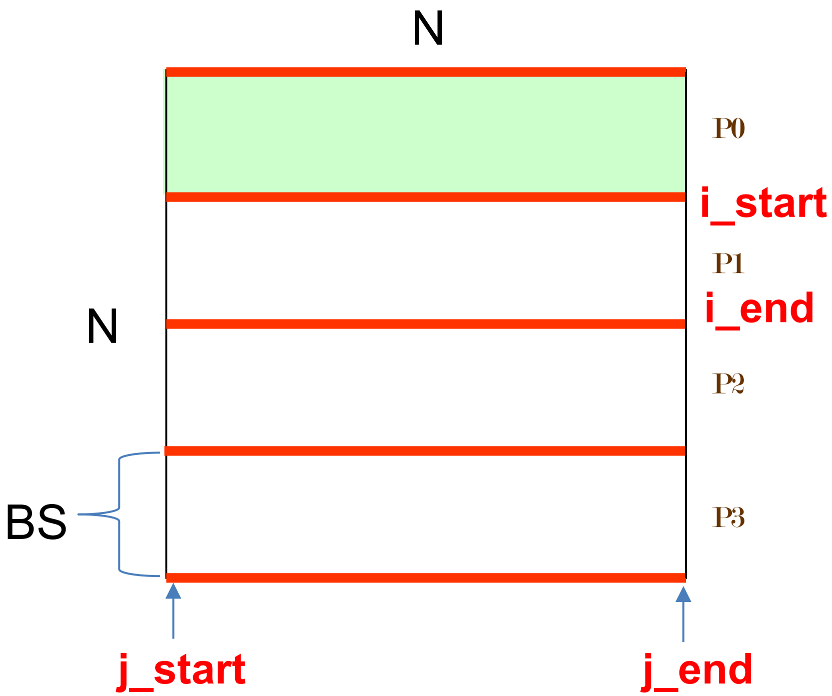

```
void InitDecomposition (limits *decomposition, int N, int nt) {
  int i;

  for (i = 0; i < nt; i++) {

        // initialize decomposition  for each thread



  }
}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row



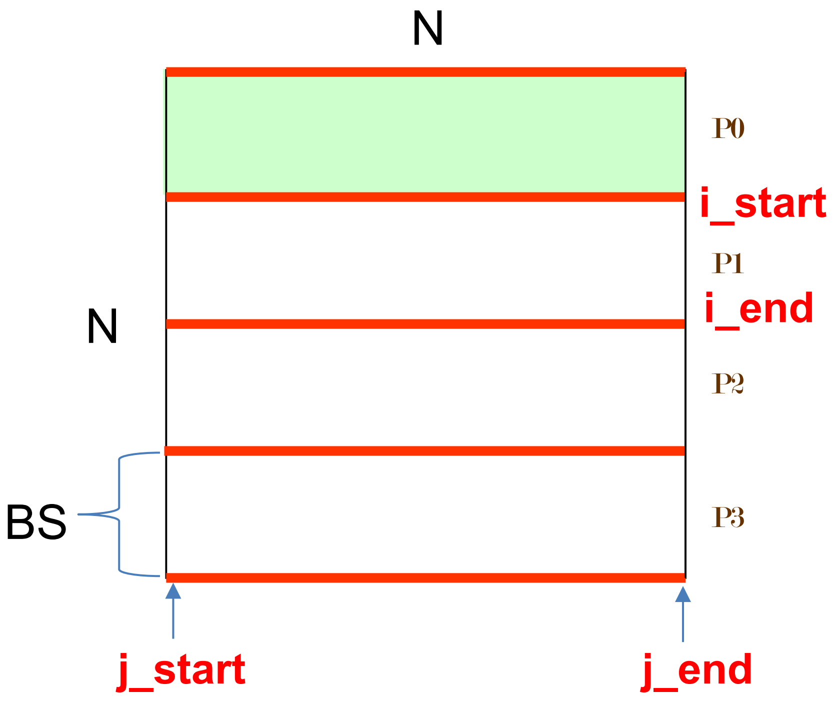

```
void InitDecomposition (limits *decomposition, int N, int nt) {
  int i;

  for (i = 0; i < nt; i++) {
        decomposition[i].i_start =
        decomposition[i].i_end =

        decomposition[i].j_start =
        decomposition[i].j_end =
  }
}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row




```
void InitDecomposition (limits *decomposition, int N, int nt) {
  int i, BS = N / nt;

  for (i = 0; i < nt; i++) {
        decomposition[i].i_start = BS * i;
        decomposition[i].i_end = decomposition[i].i_start + BS;

        decomposition[i].j_start = 0;
        decomposition[i].j_end = N-1;
  }
}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;

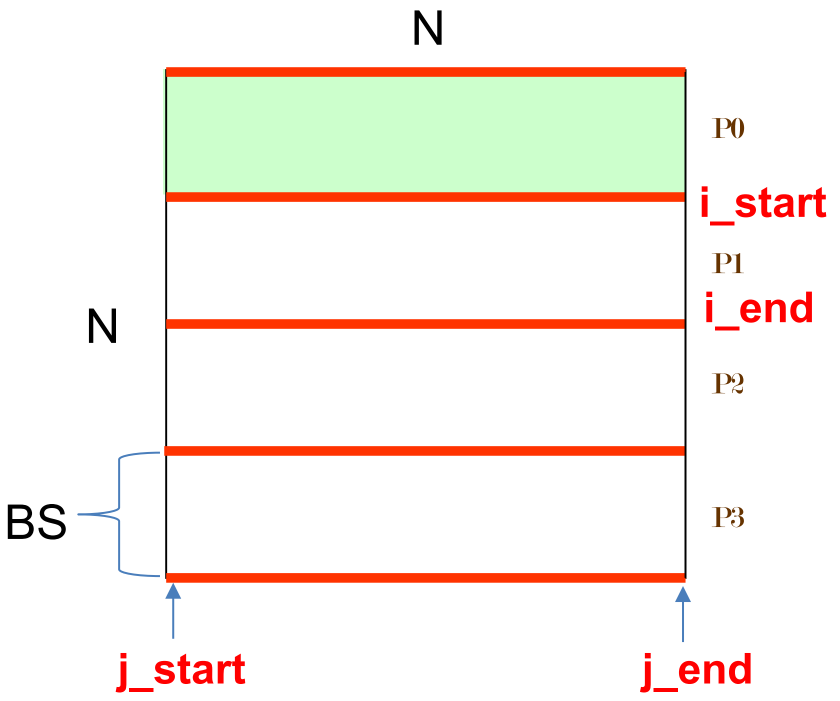
mod = N % nt ≠ 0

➔ we need to balance number of rows per block

➔ Let's distribute mod rows, adding one row to the first mod threads

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

1) Blocks by rows where:
   - N is not a multiple of the number of threads
   - Unbalance is at most equal to 1 row




```
void InitDecomposition (limits *decomposition, int N, int nt) {
  int i, mod = N % nt, BS = N / nt;

  for (i = 0; i < nt; i++) {
        decomposition[i].i_start = BS * i;
        decomposition[i].i_end = decomposition[i].i_start + BS;
        if (mod > 0) {
                if (i < mod) {
                        decomposition[i].i_start += i;
                        decomposition[i].i_end += i+1;
                }
                else {
                        decomposition[i].i_start += mod;
                        decomposition[i].i_end += mod;
                }
        }
        decomposition[i].j_start = 0;
        decomposition[i].j_end = N-1;
  }
}
```

nt = number of threads

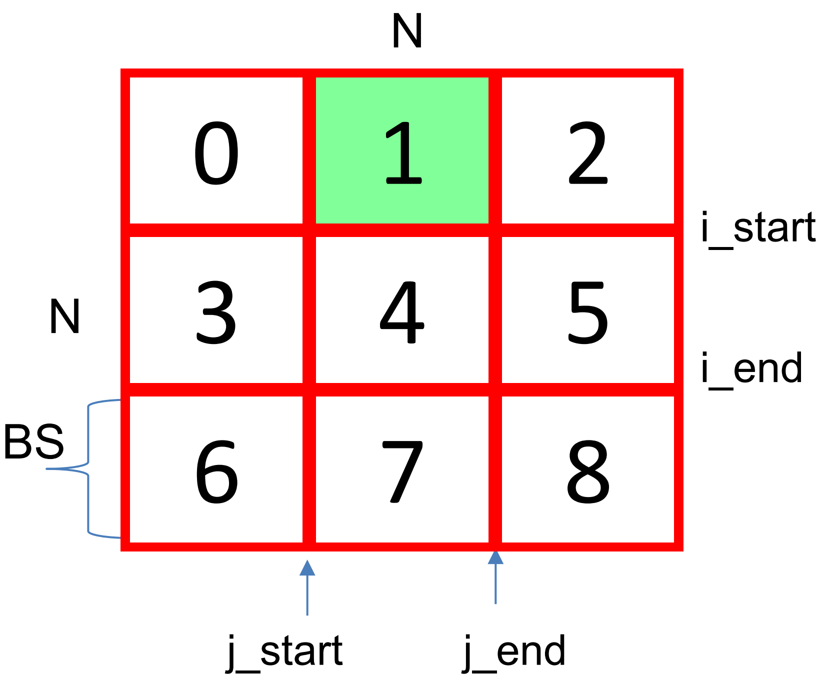
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```




```
void main (int argc, char *argv[]) {

    #pragma omp parallel
    #pragma omp single
     InitDecomposition (decomposition, N, omp_get_num_thread());

    #pragma omp parallel
    {
     …
     int i_start = …
     int i_end = …
     int j_start = …
     int j_end = …
     foo (i_start, i_end, j_start, j_end);
    }
}
```

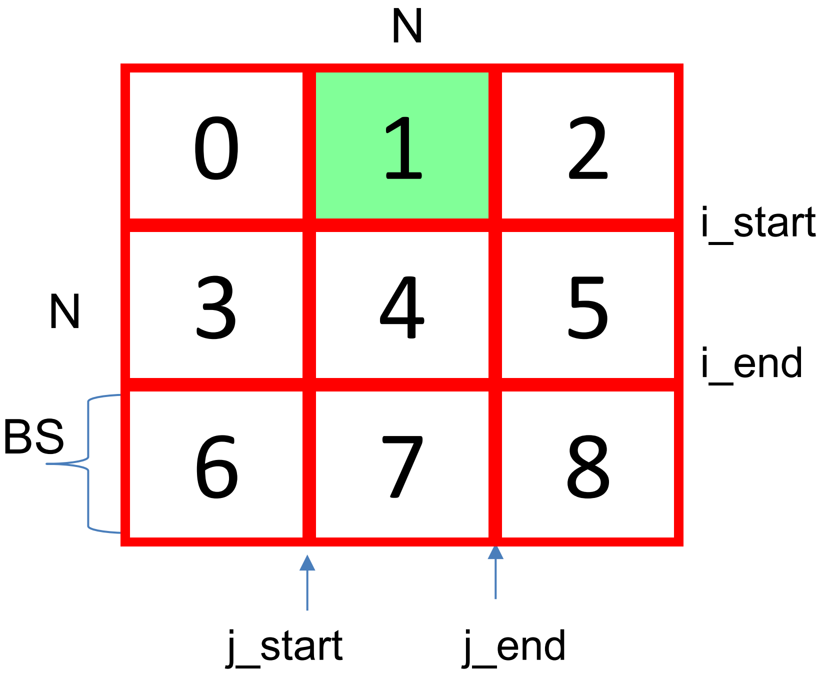
nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```




```
void main (int argc, char *argv[]) {

    #pragma omp parallel
    ~~#pragma omp single~~
     InitDecomposition (decomposition, N, omp_get_num_threads());

    #pragma omp parallel
    {
     …
     int i_start = …
     int i_end = …
     int j_start = …
     int j_end = …
     foo (i_start, i_end, j_start, j_end);
    }
}
```

nt = number of threads

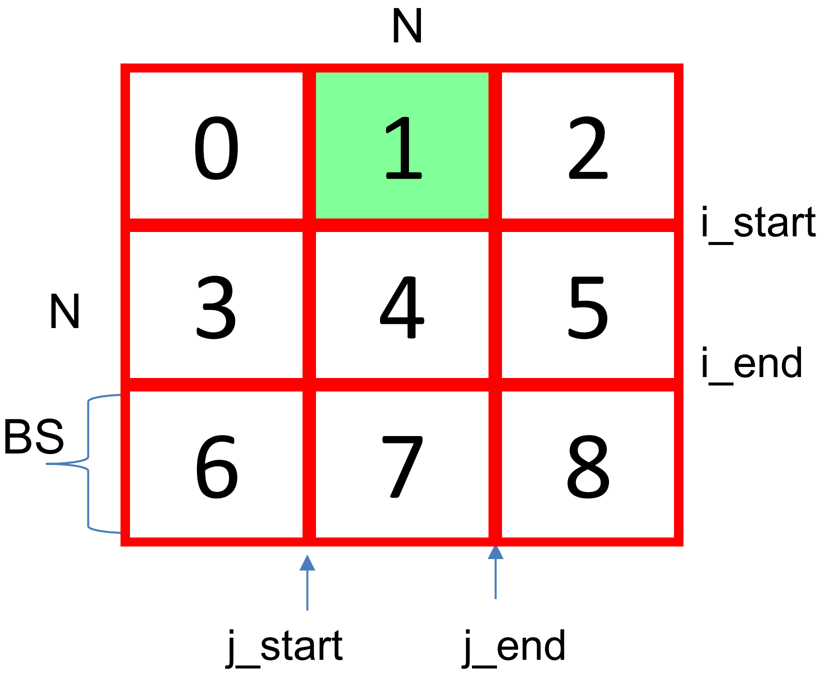
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0

```
typedef struct {
        int i_start, i_end, j_start, j_end;
} limits;
limits decomposition[nthreads];
```



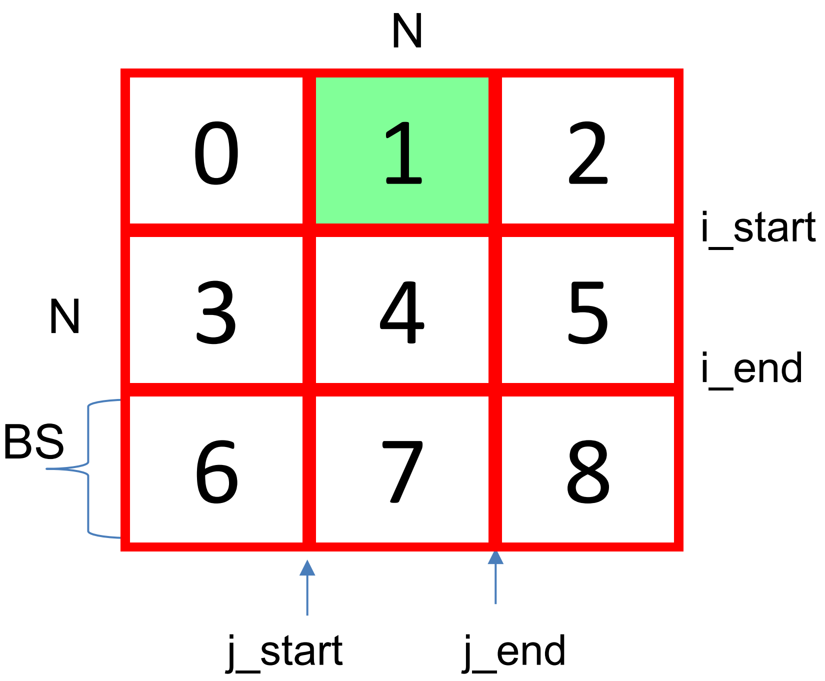

```
void main (int argc, char *argv[]) {

    #pragma omp parallel
     InitDecomposition (decomposition, N, omp_get_num_threads());

    #pragma omp parallel
    {
     int myid = omp_get_thread_num();
     int i_start = decomposition [myid].i_start;
     int i_end = decomposition [myid].i_end;
     int j_start = decomposition [myid].j_start;
     int j_end = decomposition [myid].j_end;
     foo (i_start, i_end, j_start, j_end);
    }
}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0



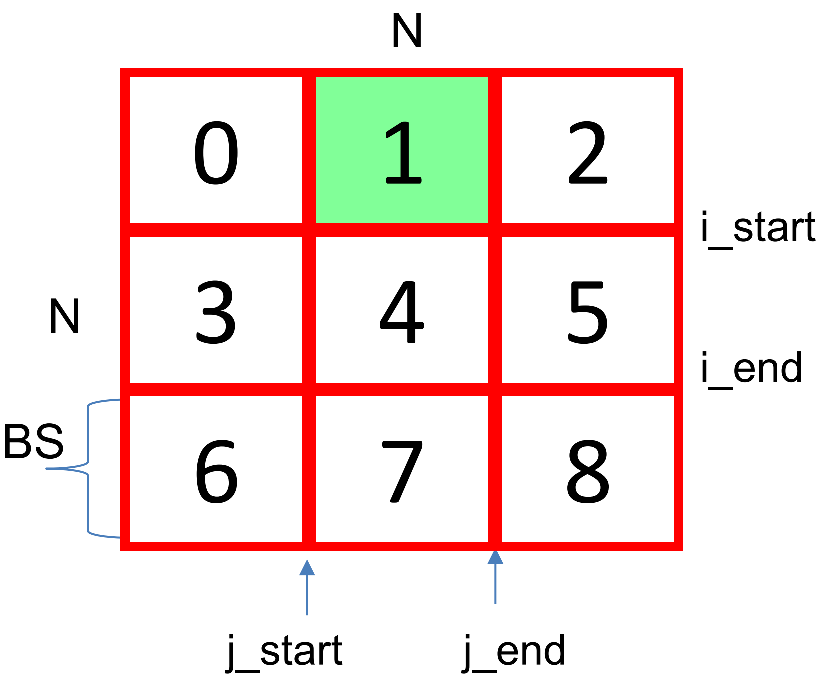

```
void InitDecomposition (limits *decomposition, int N, int nt) {

        // initialize decomposition  for each thread
        // The function is already within a parallel region

}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0 (in the example below K = 3)




```
void InitDecomposition (limits *decomposition, int N, int nt) {

        // initialize decomposition  for each thread
        // the function is already within a parallel region

}
```

blocks = number of threads = K * K

blocks per row = K, # blocks per column = K

myid = omp_get_thread_num()

i_block = myid / K

j_block = myid % K

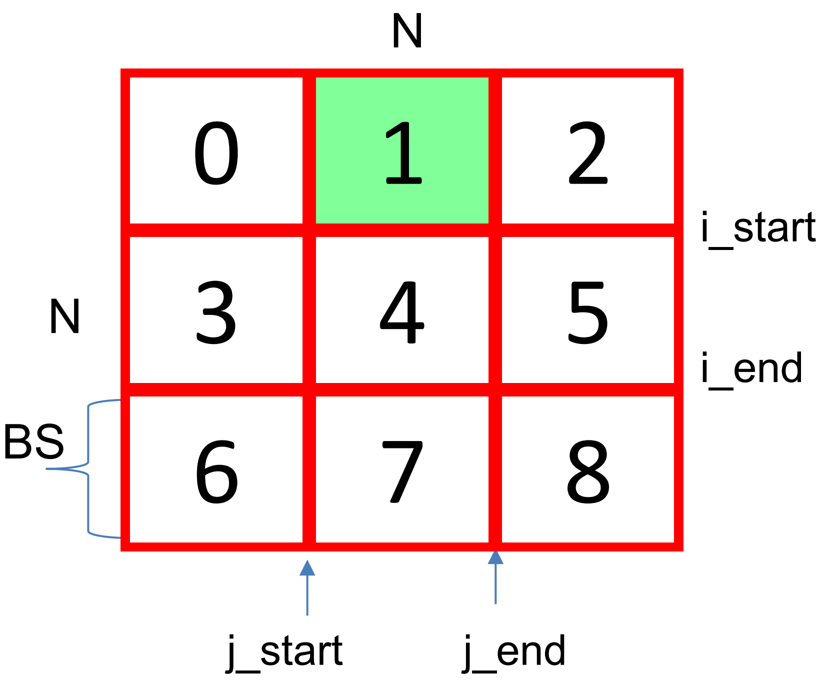
nt = number of threads

BS = number of rows in a block

BS = N / nt;

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0




```
void InitDecomposition (limits *decomposition, int N, int nt) {
    int myid, K, BS;

    myid = omp_get_thread_num();
    K = sqrt(nt);
    BS = N / K;

    int i_block = myid / K;
    decomposition[myid].i_start =
    decomposition[myid].i_end =

    int j_block = myid % K;
    decomposition[myid].j_start =
    decomposition[myid].j_end =

}
```

nt = number of threads
BS = number of rows in a block

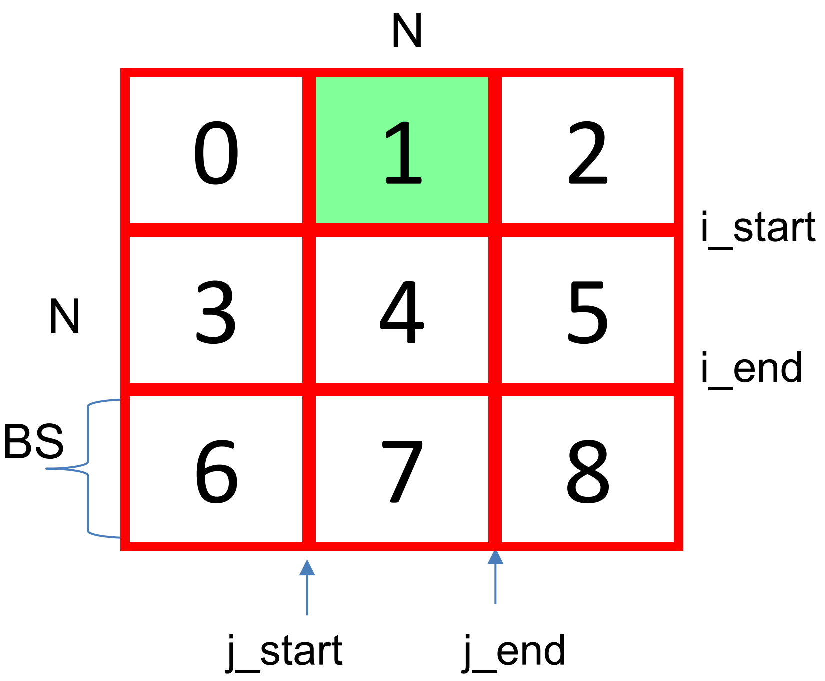
BS = N / nt;

blocks = number of threads = K * K
blocks per row = K, # blocks per column = K

myid = omp_get_thread_num()
i_block = myid / K
j_block = myid % K

Problem 5: Complete the code of *InitDecomposition* and *main* program following a **Block geometric data decomposition** such that:

2) Blocks by row/columns partition where:
   - K^2 = number of threads
   - N % K = 0




```
void InitDecomposition (limits *decomposition, int N, int nt) {
    int myid, K, BS;

    myid = omp_get_thread_num();
    K = sqrt(nt);
    BS = N / K;

    int i_block = myid / K;
    decomposition[myid].i_start = i_block * BS;
    decomposition[myid].i_end = decomposition[id].i_start + BS;

    int j_block = myid % K;
    decomposition[myid].j_start = j_block * BS;
    decomposition[myid].j_end = decomposition[id].j_start + BS;

}
```

nt = number of threads
BS = number of rows in a block

BS = N / nt;