COL380

Introduction to
Parallel & Distributed Programming

Derived Datatypes

- MPI does not understand language’s layout (struct, e.g.)
  - ➡ Too system architecture dependent

MPI_INT, MPI_FLOAT ..

- Typemap:
  - ➡ (type_0, disp_0), ..., (type_n, disp_n)
  - ➡ i^{th} entry is of type_i and starts at byte base + disp_i

Derived Datatypes

- MPI does not understand language’s layout (struct, e.g.)
  - ➡ Too system architecture dependent

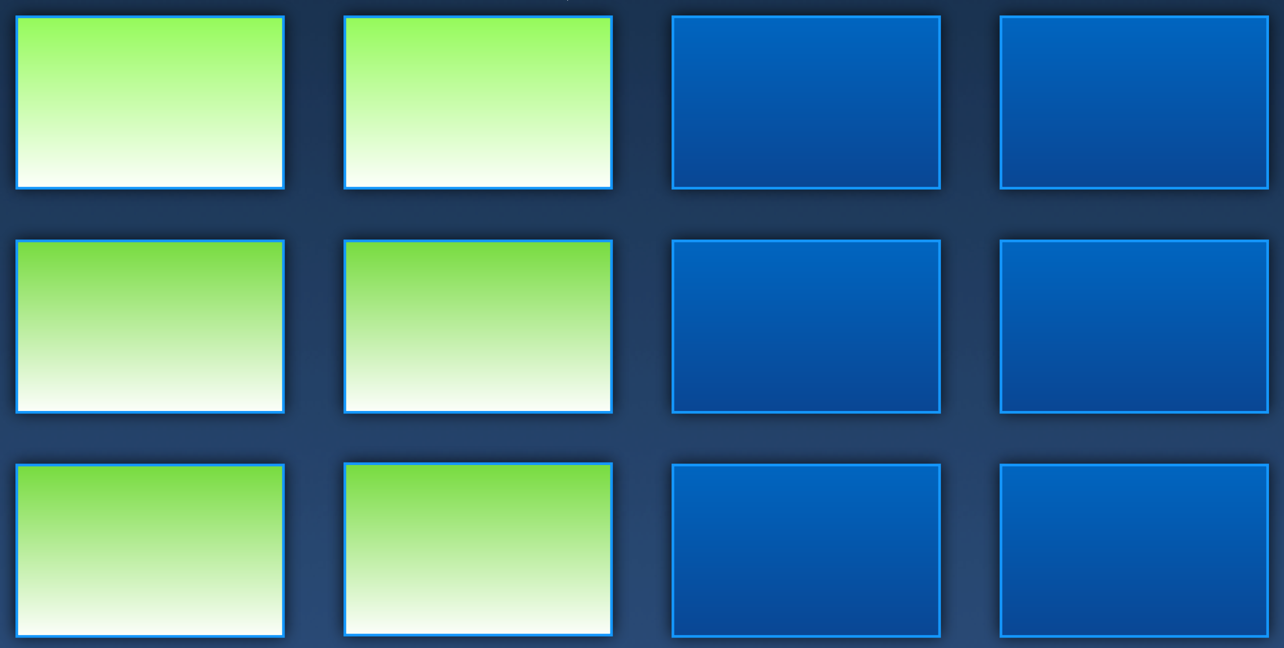
MPI_INT, MPI_FLOAT ..

- Typemap:
  - ➡ (type_0, disp_0), ..., (type_n, disp_n)
  - ➡ i^{th} entry is of type_i and starts at byte base + disp_i

```
MPI_Datatype newtype;
MPI_Type_contiguous(count, MPI_INT, &newtype);
```

Blocks

- Equally-spaced blocks of the known datatype
  - ➡ MPI_Type_vector(blockcount, blocklength, blockstride, knowntype, &newtype);
    - Assume contiguous copies of 'knowntype'
    - Stride between blocks specified in units of knowntype
    - All picked blocks are of the same length



  - ➡ MPI_Type_create_hvector(blk_count, blk_length, bytestride, knowntype, &newtype);

Gap between blocks is in bytes

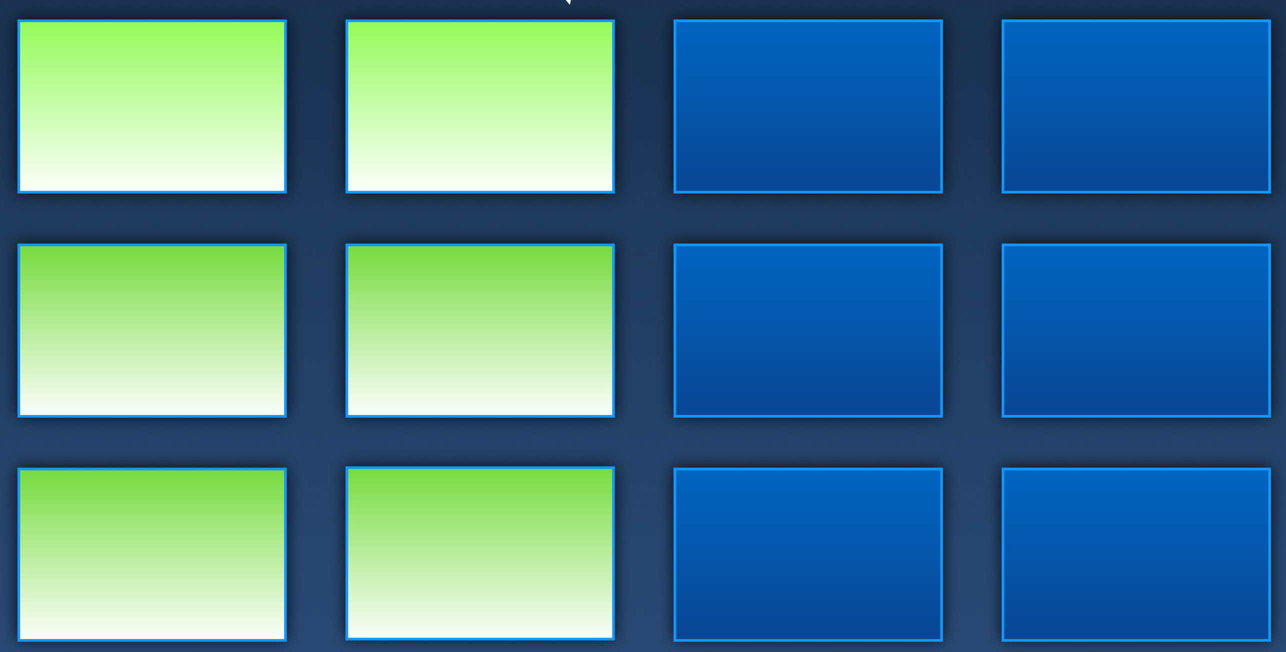
Subodh Kumar

Blocks

- Equally-spaced blocks of the known datatype

3 2 4

➡ MPI_Type_vector(blockcount, blocklength, blockstride, knowntype, &newtype);



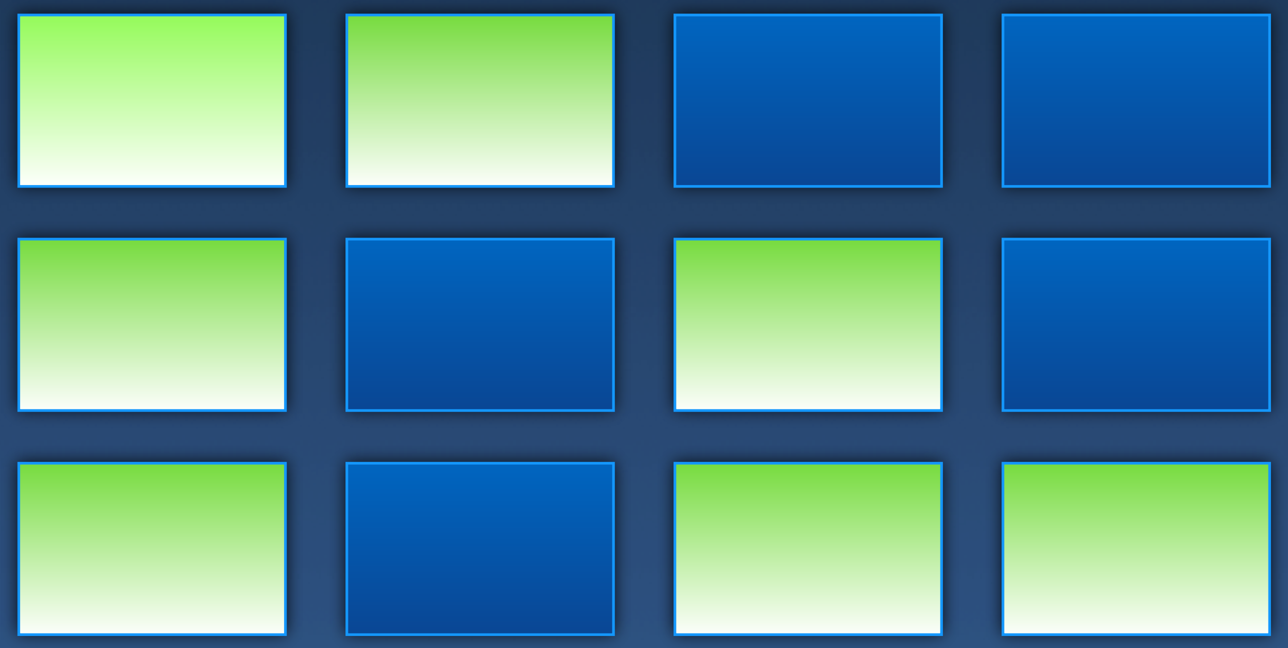
- ▸ Assume contiguous copies of ‘knowntype'
- ▸ Stride between blocks specified in units of knowntype
- ▸ All picked blocks are of the same length

➡ MPI_Type_create_hvector(blk_count, blk_length, bytestride, knowntype, &newtype);

Gap between blocks is in bytes

Generalized Blocks

- MPI_Type_indexed(count, array_of_blocklengths, array_of_strides, knowntype, &newtype);
  - ➡ Blocks can contain different number of copies
  - ➡ And may have different strides
  - ➡ But the same data type



Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_bytedisplacements, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)},
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Subodh Kumar

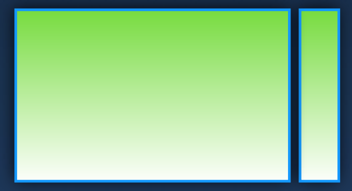
Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_bytedisplacements, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)},

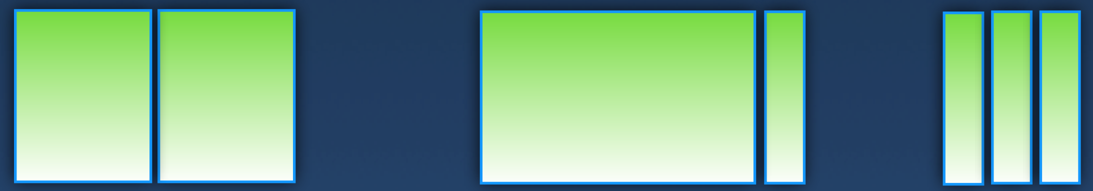
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_bytedisplacements, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)},



    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};

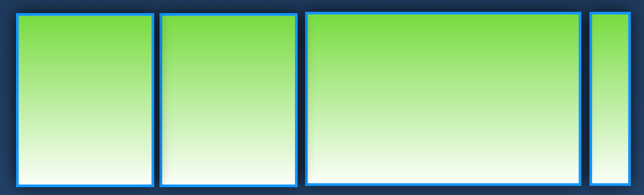


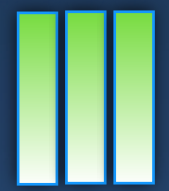
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_bytedisplacements, array_of_knowntypes, &newtype)
    - ➡ Example:
        - Suppose Type0 = {(double, 0), (char, 8)},

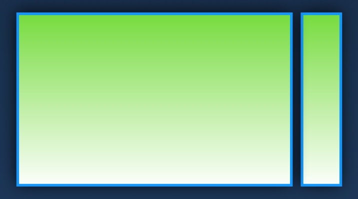
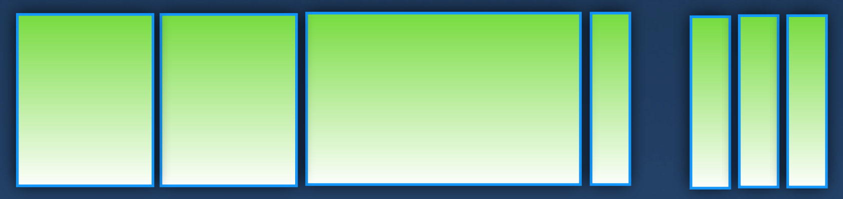
        - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};





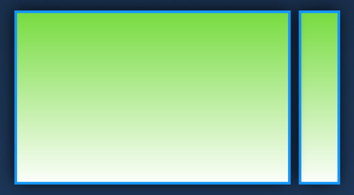
        - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
    - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
        - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

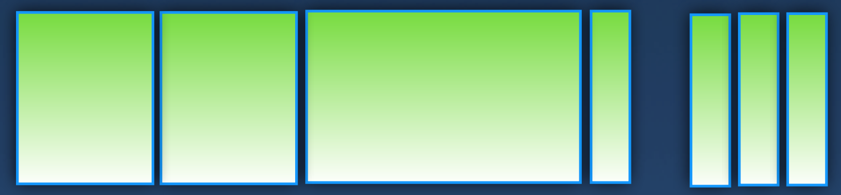
- MPI_Type_create_struct(count, array_of_blocklengths, array_of_bytedisplacements, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)}, 
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26}; 
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_bytedisplacements, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)},



    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};



    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

MPI_Type_get_contents(..)

Data Type Functions

- MPI_Type_commit(&datatype)
  - ➡ A datatype object must be committed before communication
- MPI_Type_size(datatype, &size)
  - ➡ Total size in bytes
- MPI_Type_get_extent(datatype, &beg, &extent);
- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

Data Type Functions

- MPI_Type_commit(&datatype)
  - ➡ A datatype object must be committed before communication
- MPI_Type_size(datatype, &size)
  - ➡ Total size in bytes
- MPI_Type_get_extent(datatype, &b

```
MPI_Datatype atype;
MPI_Type_contiguous(4, MPI_CHAR, &atype);
int asize;
MPI_Type_size(atype, &asize);
MPI_Type_commit(&atype);
MPI_Send(buf, nItems, atype, dest, ..);
MPI_Recv(...);
```

- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

Data Type Functions

- MPI_Type_commit(&datatype)
  - ➡ A datatype object must be committed before communication
- MPI_Type_size(datatype, &size)
  - ➡ Total size in bytes
- MPI_Type_get_extent(datatype, &beg, &extent);
- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

Derived Datatype

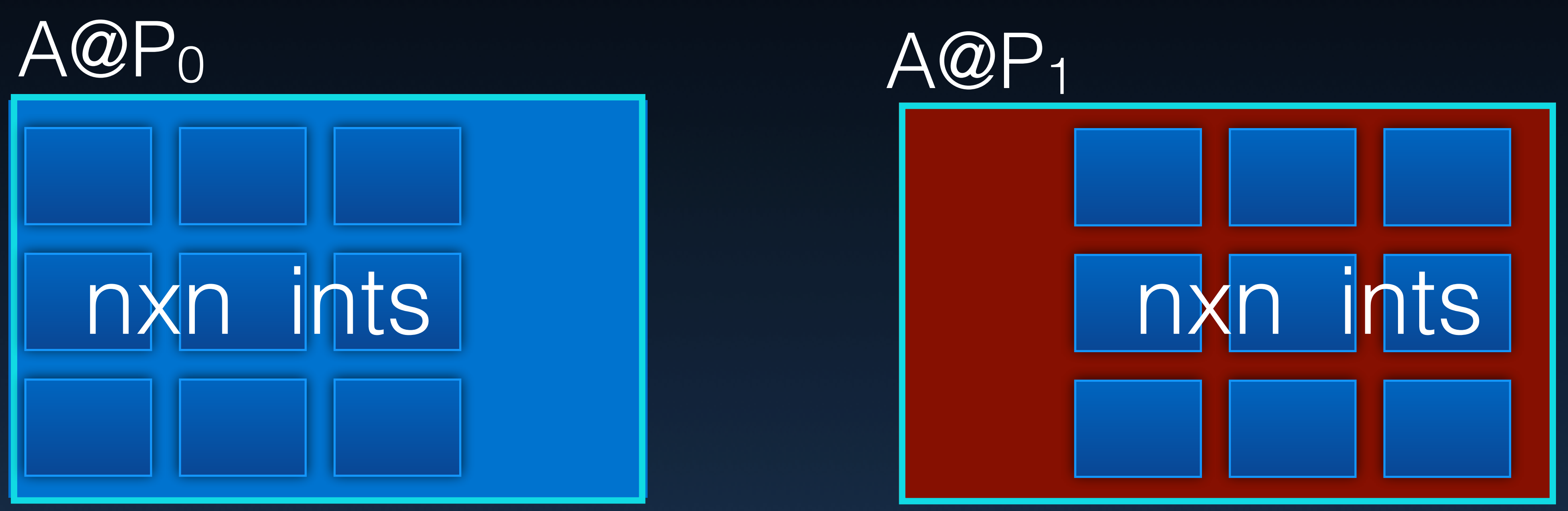
```
struct Particle
{
   int class;       // particle class
   double d[6]; // particle coordinates
   char b[7];      // some additional info
};
```

Derived Datatype

```
sendParticles(struct Particle particle[], int N):
    MPI_Datatype Particletype;
    MPI_Datatype types[3] = {MPI_INT, MPI_DOUBLE, MPI_CHAR};
    int blockcount[3] = {1, 6, 7};

    /* compute displacements of structure components */
    MPI_Aint disp[3];
    MPI_Address(particle, disp);
    MPI_Address(particle[0].d, disp+1);
    MPI_Address(particle[0].b, disp+2);
    for (int i=2; i >= 0; i--) disp[i] -= disp[0];

    MPI_Type_struct(3, blockcount, disp, types, &Particletype);
    MPI_Type_commit( &Particletype);
    MPI_Send(particle, N, Particletype, dest, tag, comm);
```

```
struct Particle
{
   int class;        // particle class
   double d[6]; // particle coordinates
   char b[7];       // some additional info
};
```

Data Transfer



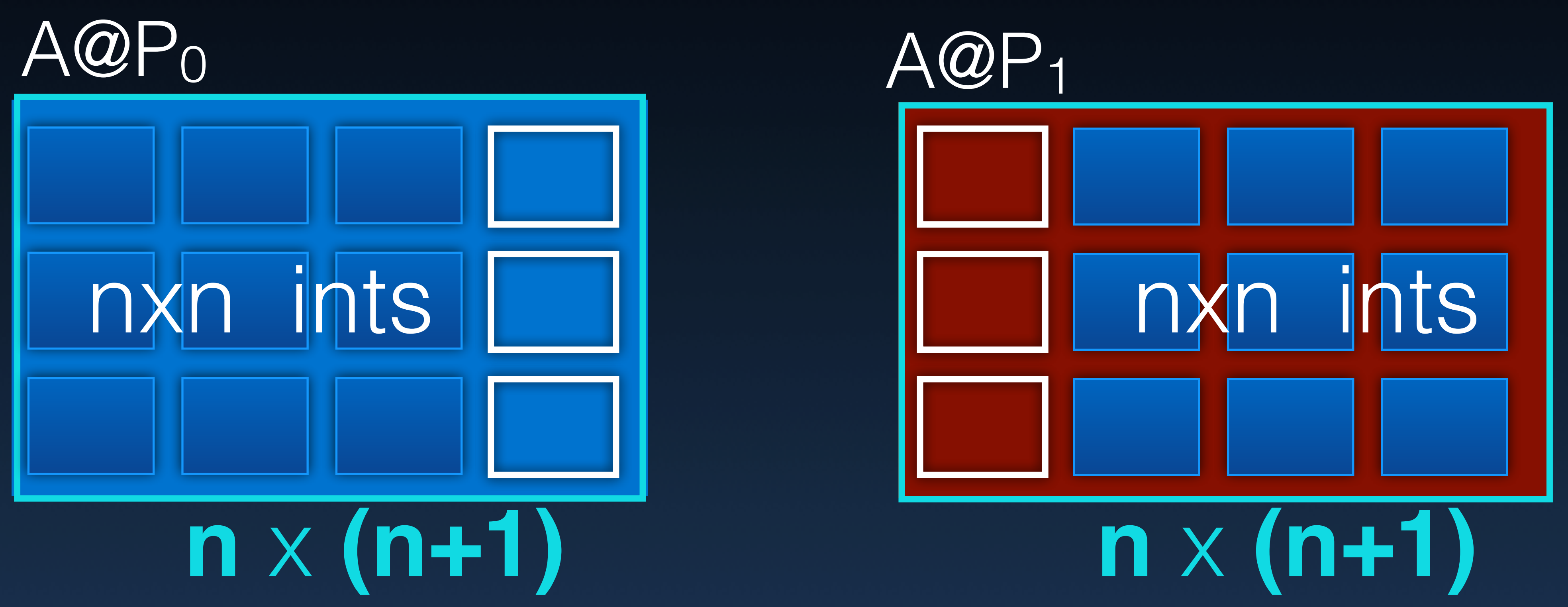
$A@P_0$

nxn ints

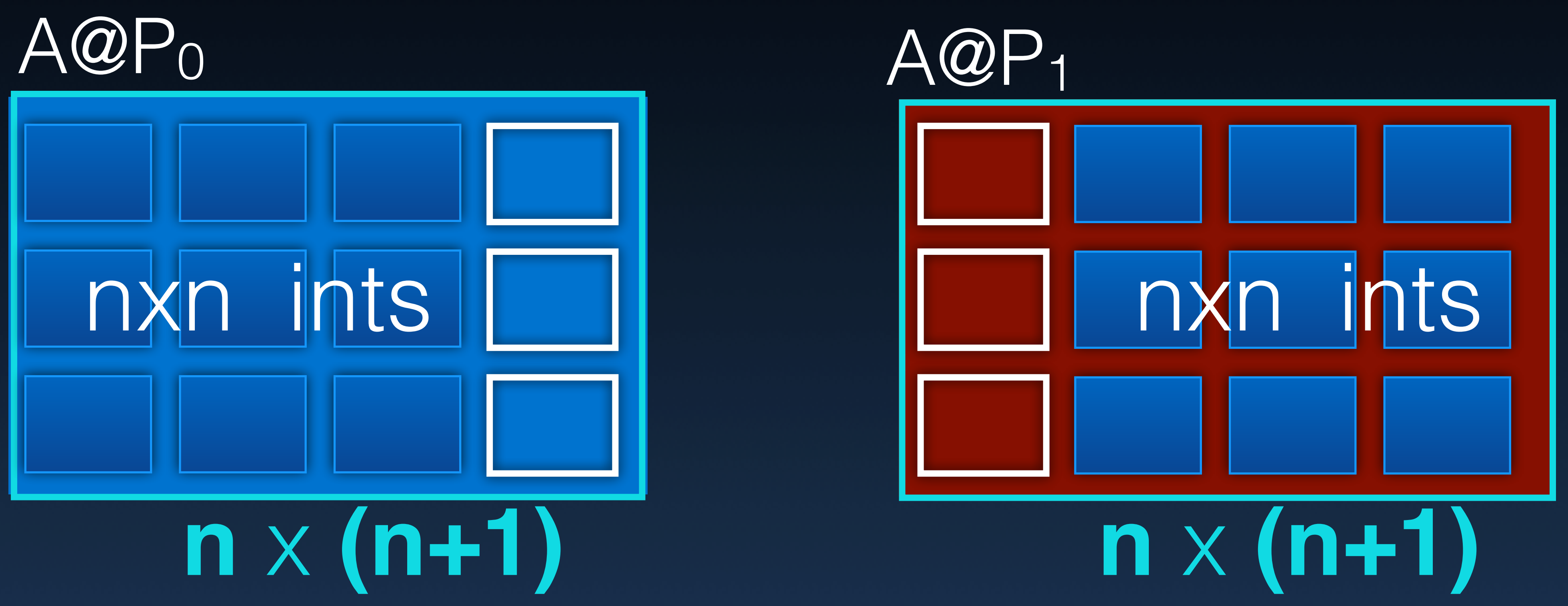
$A@P_1$

nxn ints

Data Transfer



Data Transfer



```
MPI_Status status;
MPI_Datatype column;
MPI_Type_vector(n, 1, n+1, MPI_INT, &column);
MPI_Type_commit(&column);
if(rank == 0) {
   MPI_Send(A+n-1, 1, column, 1, tag, MPI_COMM_WORLD);
   MPI_Recv(A+n, 1, column, 1, tag, MPI_COMM_WORLD, &status);
}
if(rank == 1) {
   MPI_Recv(A, 1, column, 0, tag, MPI_COMM_WORLD, &status);
   MPI_Send(A+1, 1, column, 0, tag, MPI_COMM_WORLD);
}
```

Collective Communication

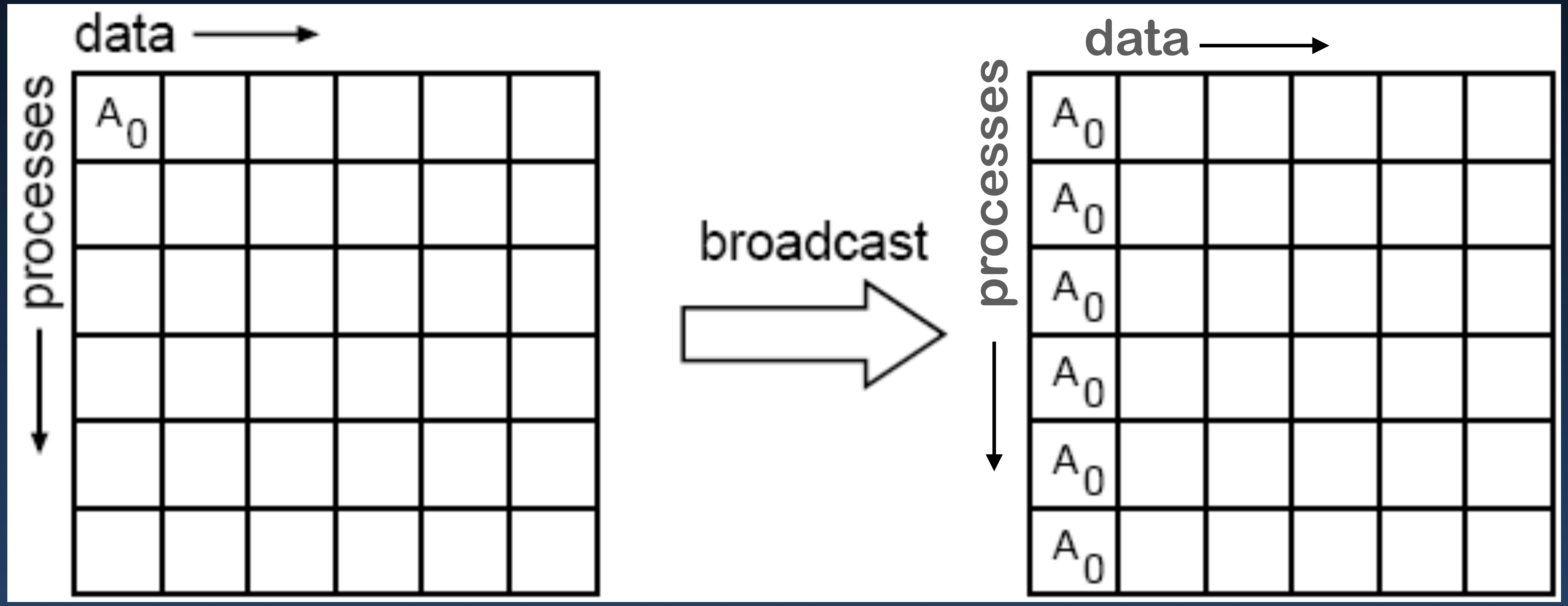
- MPI_Barrier
  - Barrier synchronization across all members of a group
- MPI_Bcast
  - Broadcast from one member to all members of a group
- MPI_Scatter, MPI_Gather, MPI_Allgather
  - Gather data from all members of a group to one
- MPI_Alltoall
  - complete exchange or all-to-all
- MPI_Allreduce, MPI_Reduce
  - Reduction operations
- MPI_Reduce_Scatter
  - Combined reduction and scatter operation
- MPI_Scan, MPI_Exscan
  - Prefix

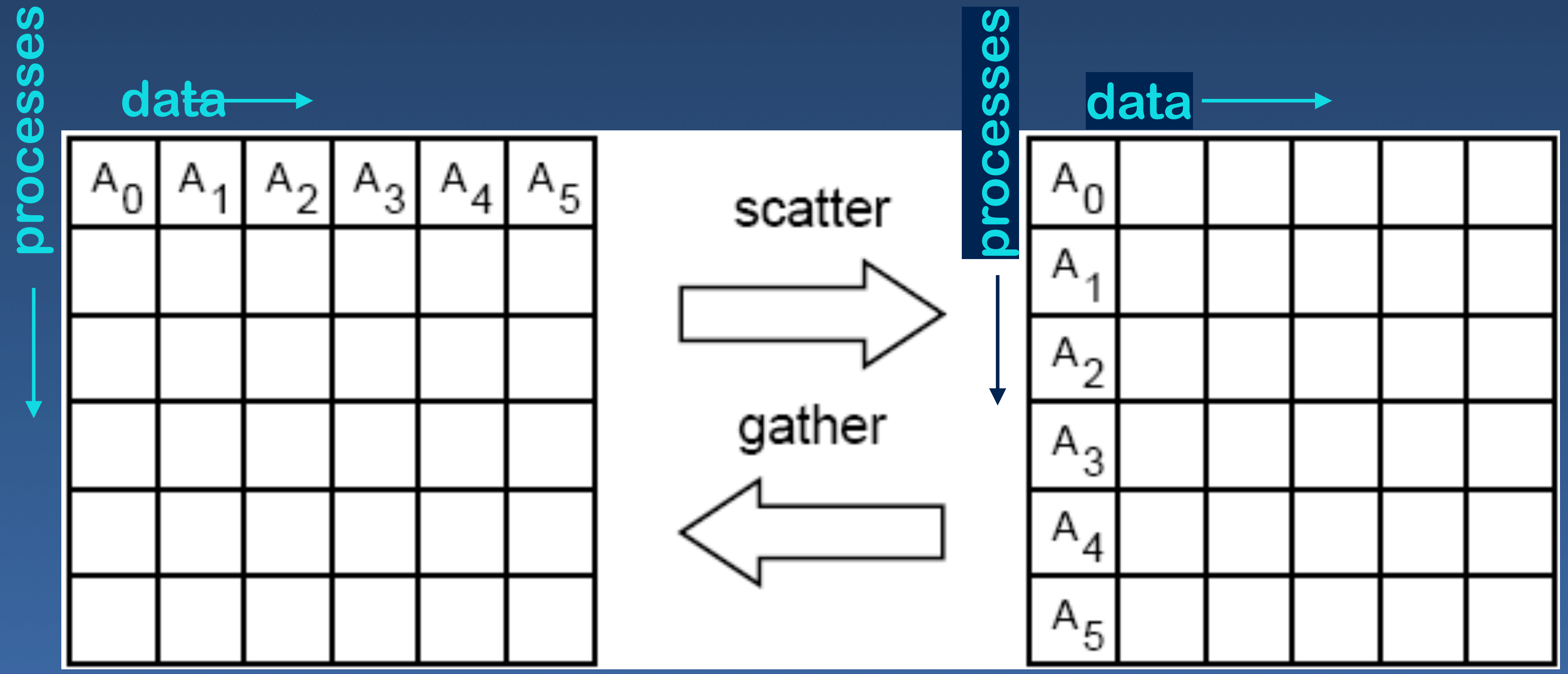
Barrier

- Synchronization of the calling processes
  - the call blocks until all of the processes have placed the call

```
MPI_Barrier(comm);
```

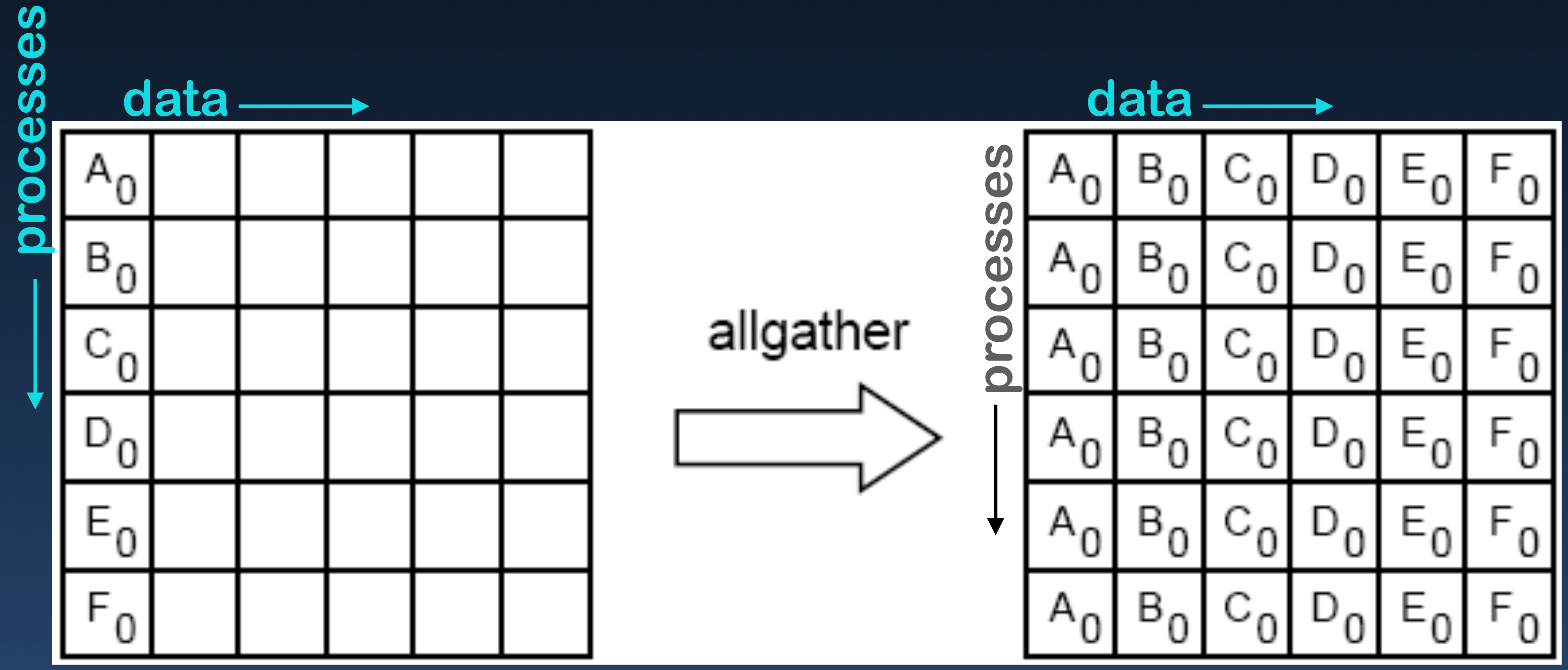
Collective Communication





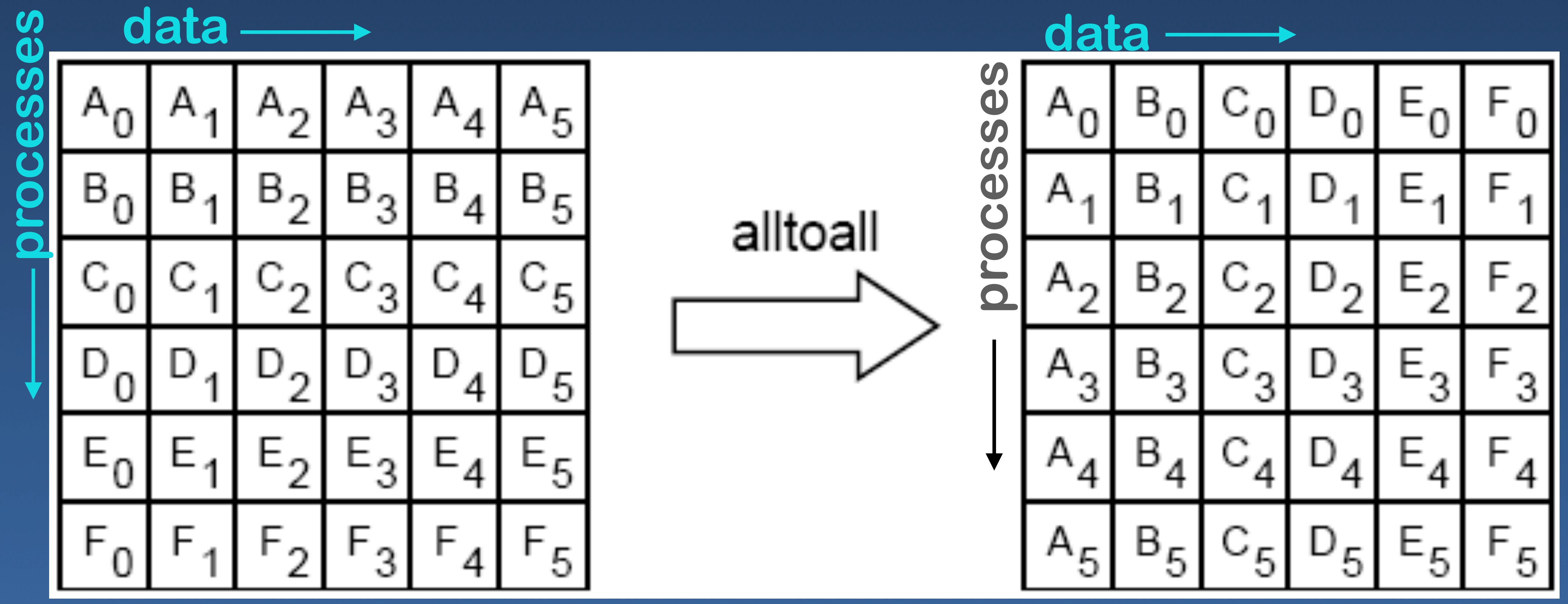
Example from MPI document

Collective Communication



A_0					
B_0					
C_0					
D_0					
E_0					
F_0					

A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0



A_0	A_1	A_2	A_3	A_4	A_5
B_0	B_1	B_2	B_3	B_4	B_5
C_0	C_1	C_2	C_3	C_4	C_5
D_0	D_1	D_2	D_3	D_4	D_5
E_0	E_1	E_2	E_3	E_4	E_5
F_0	F_1	F_2	F_3	F_4	F_5

A_0	B_0	C_0	D_0	E_0	F_0
A_1	B_1	C_1	D_1	E_1	F_1
A_2	B_2	C_2	D_2	E_2	F_2
A_3	B_3	C_3	D_3	E_3	F_3
A_4	B_4	C_4	D_4	E_4	F_4
A_5	B_5	C_5	D_5	E_5	F_5

Example from MPI document

Broadcast

- Broadcast: one sender, many receivers
- Includes all processes in communicator
    - ➡ all processes must make a call to MPI_Bcast
    - ➡ Must agree on sender
- Broadcast does not mandate global synchronization
    - ➡ Some implementations may incur synchronization
    - ➡ Call may return before other have received, e.g.
    - ➡ Different from MPI_Barrier(communicator)

Broadcast

```
MPI_Bcast(mesg, count, MPI_INT, root, comm);
```

`mesg`	pointer to message buffer
`count`	number of items sent
`MPI_INT`	type of item sent
`root`	sending processor

- **Again**: All participants must call
- count and type should be the same on all members
- Can broadcast on inter-communicators also

Deadlock?

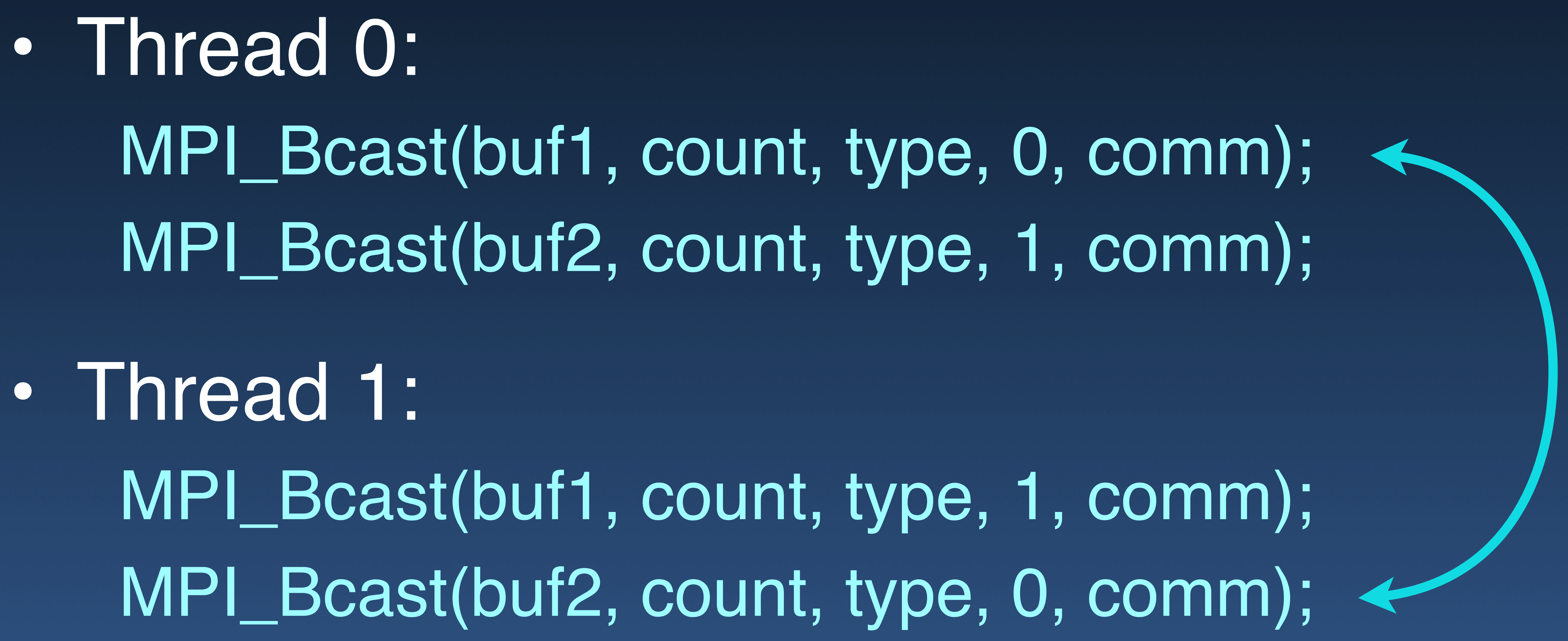
- Thread 0:

```
MPI_Bcast(buf1, count, type, 0, comm);
MPI_Bcast(buf2, count, type, 1, comm);
```

- Thread 1:

```
MPI_Bcast(buf1, count, type, 1, comm);
MPI_Bcast(buf2, count, type, 0, comm);
```

Deadlock?



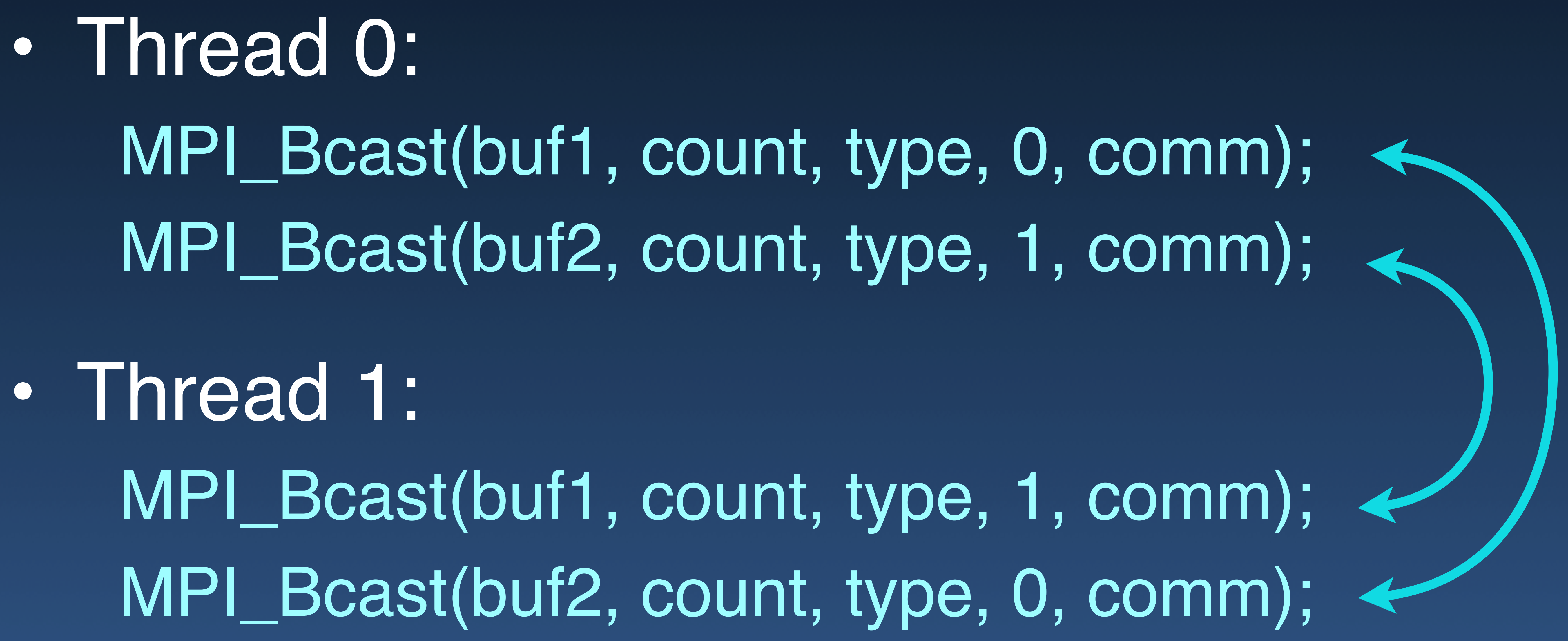
- Thread 0:

```
MPI_Bcast(buf1, count, type, 0, comm);
MPI_Bcast(buf2, count, type, 1, comm);
```

- Thread 1:

```
MPI_Bcast(buf1, count, type, 1, comm);
MPI_Bcast(buf2, count, type, 0, comm);
```

Deadlock?



- Thread 0:

```
MPI_Bcast(buf1, count, type, 0, comm);
MPI_Bcast(buf2, count, type, 1, comm);
```

- Thread 1:

```
MPI_Bcast(buf1, count, type, 1, comm);
MPI_Bcast(buf2, count, type, 0, comm);
```

MPI_Gather

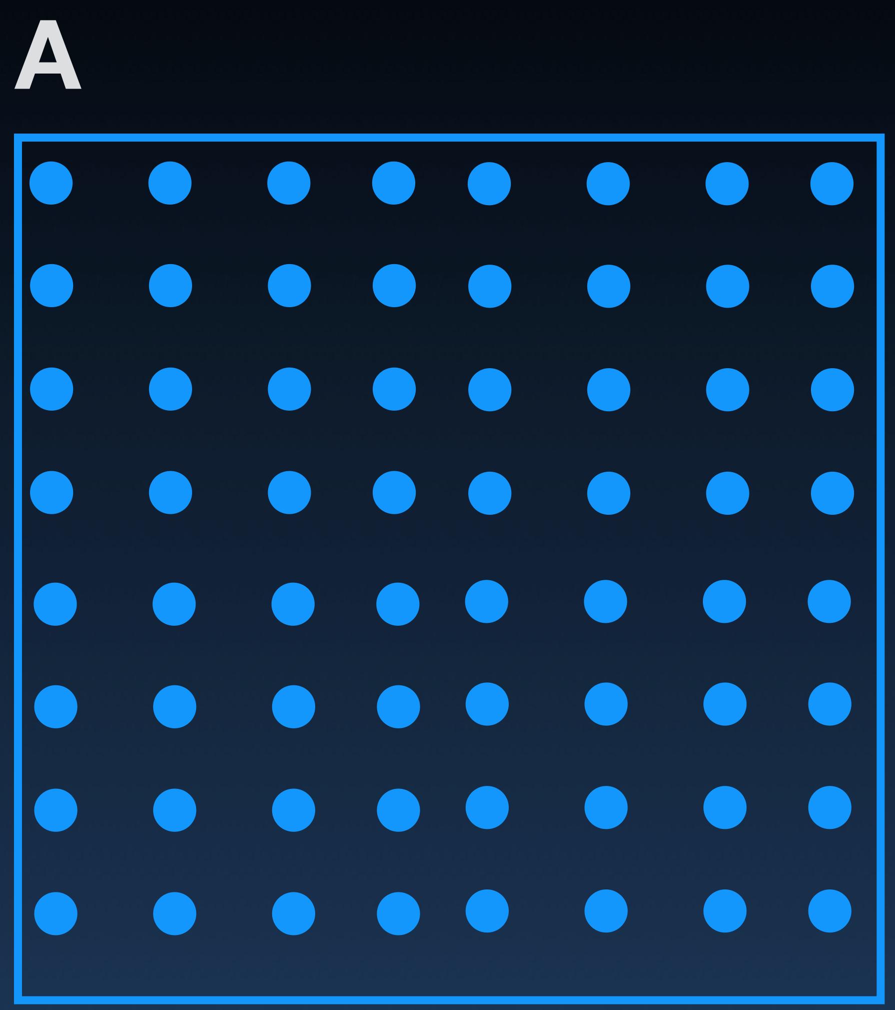
```
MPI_Gather(sendbuf, sendcount, sendtype,
  recvbuf, recvcount, recvtype, root, comm);
```

- Similar to non-roots sending:
  - MPI_Send(sendbuf, sendcount, sendtype, root, ...),
- and the root receiving n times:
  - MPI_Recv(recvbuf + i * recvcount *extent(recvtype), recvcount, recvtype, i, ...),
- **`MPI_Gatherv`** allows different size data to be gathered
- **`MPI_Allgather`** has No root, all nodes get result

Gather Example

```
MPI_Comm com;
int gsize, sendarray[100];
int root, *recvbuf;
MPI_Datatype rtype;
 ...
MPI_Comm_size( comm, &gsize);
MPI_Type_contiguous( 100, MPI_INT, &rtype );
MPI_Type_commit( &rtype );
recvbuf = (int *) malloc(gsize * 100 * sizeof(int));
MPI_Gather(sendarray, 100, MPI_INT, recvbuf, 1, rtype, root, comm)
```

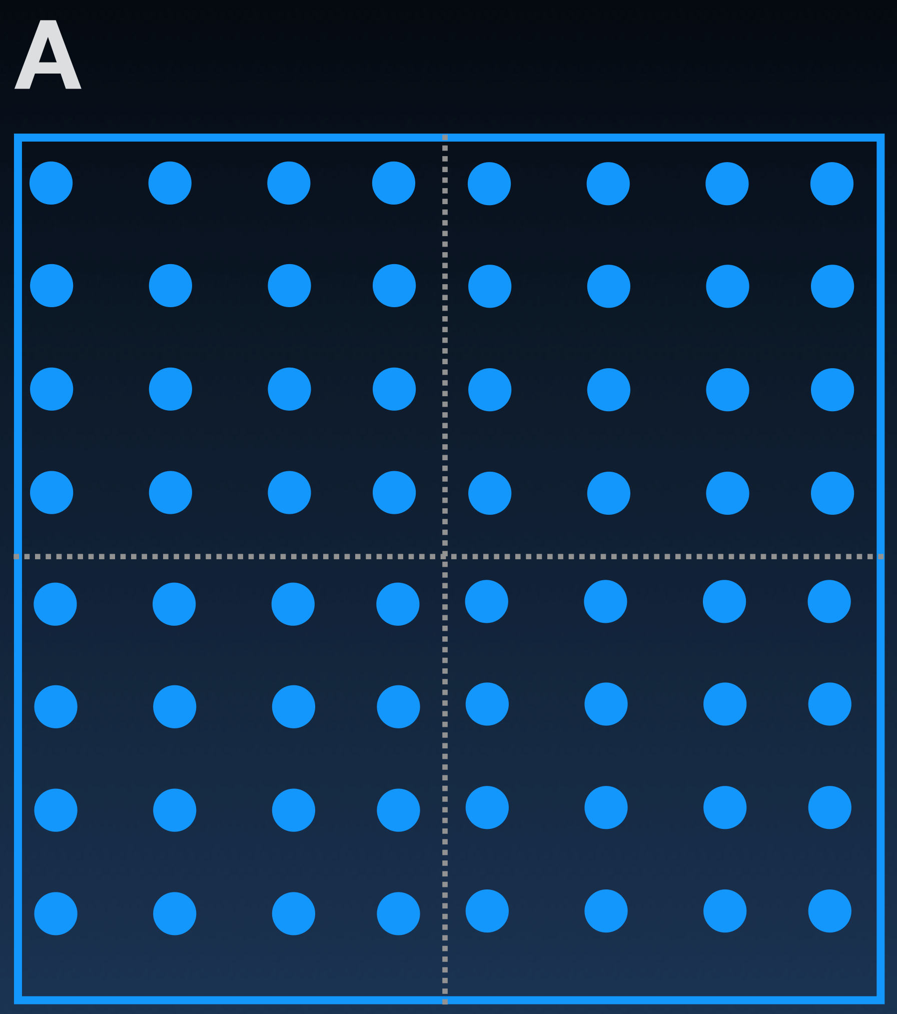
Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                    alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Subodh Kumar

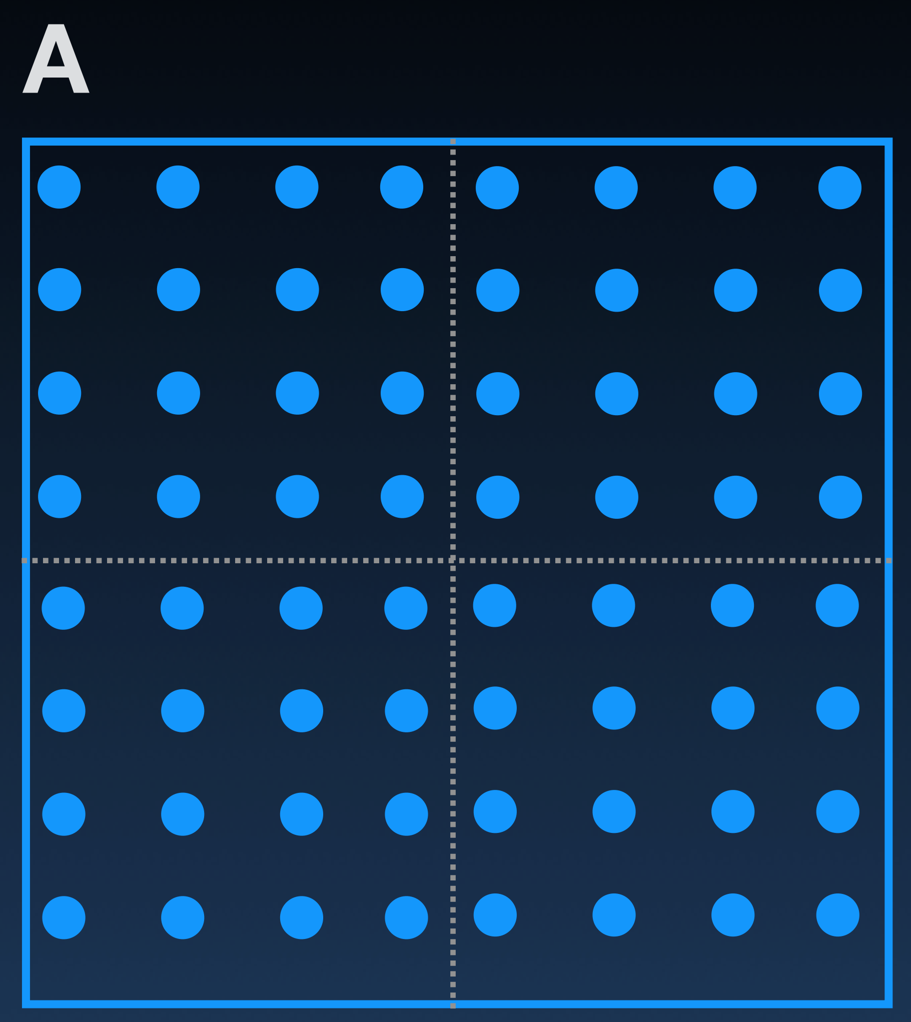
Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                   alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

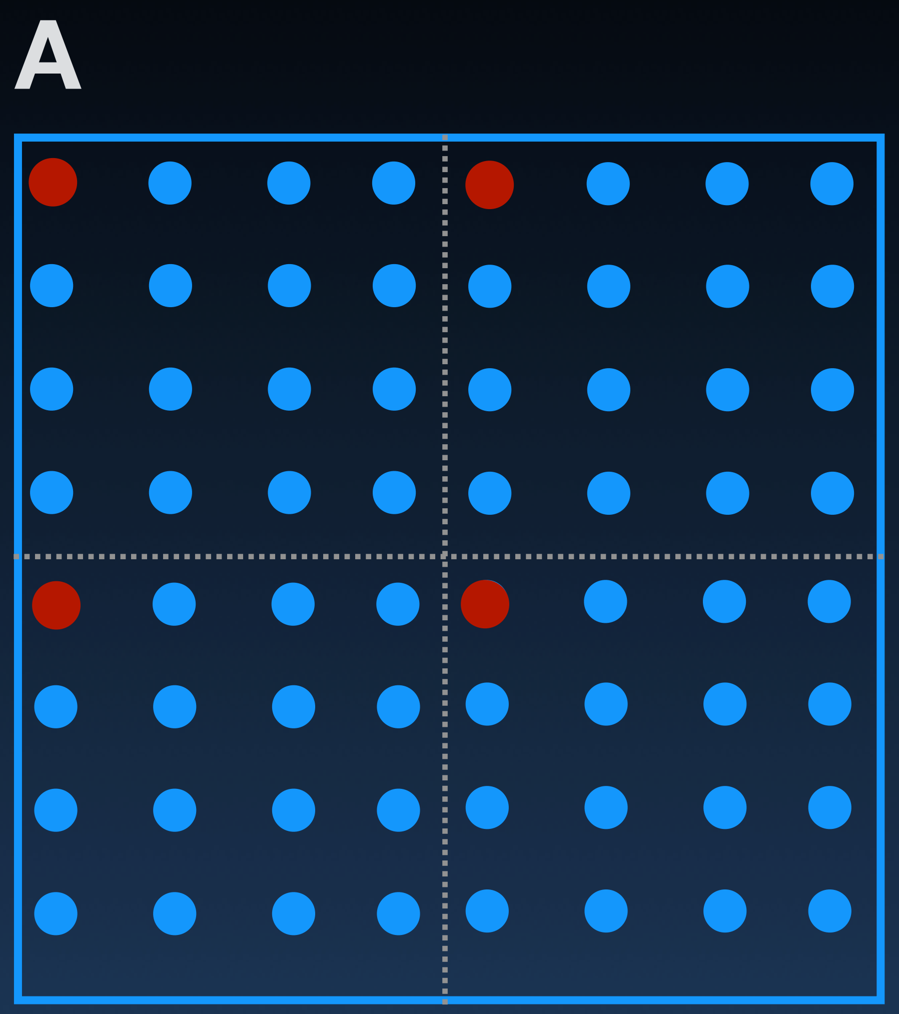
Subodh Kumar

Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                        alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

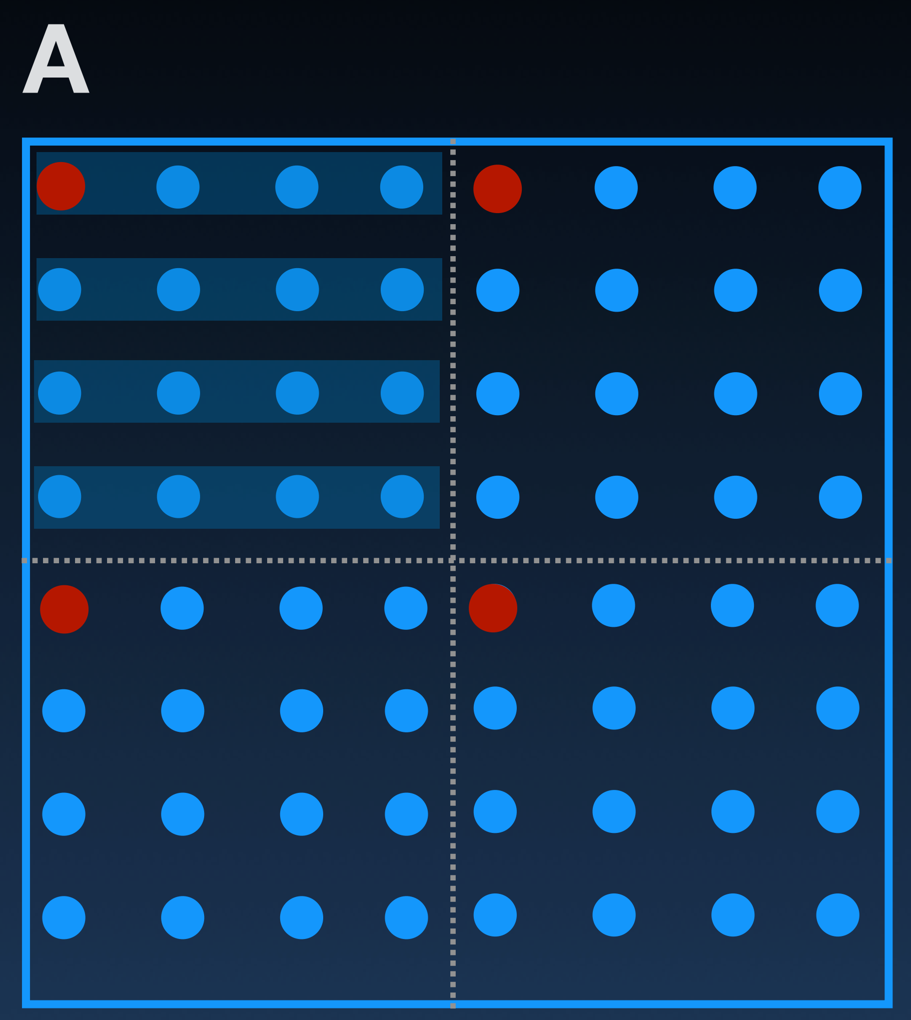
Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                       alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

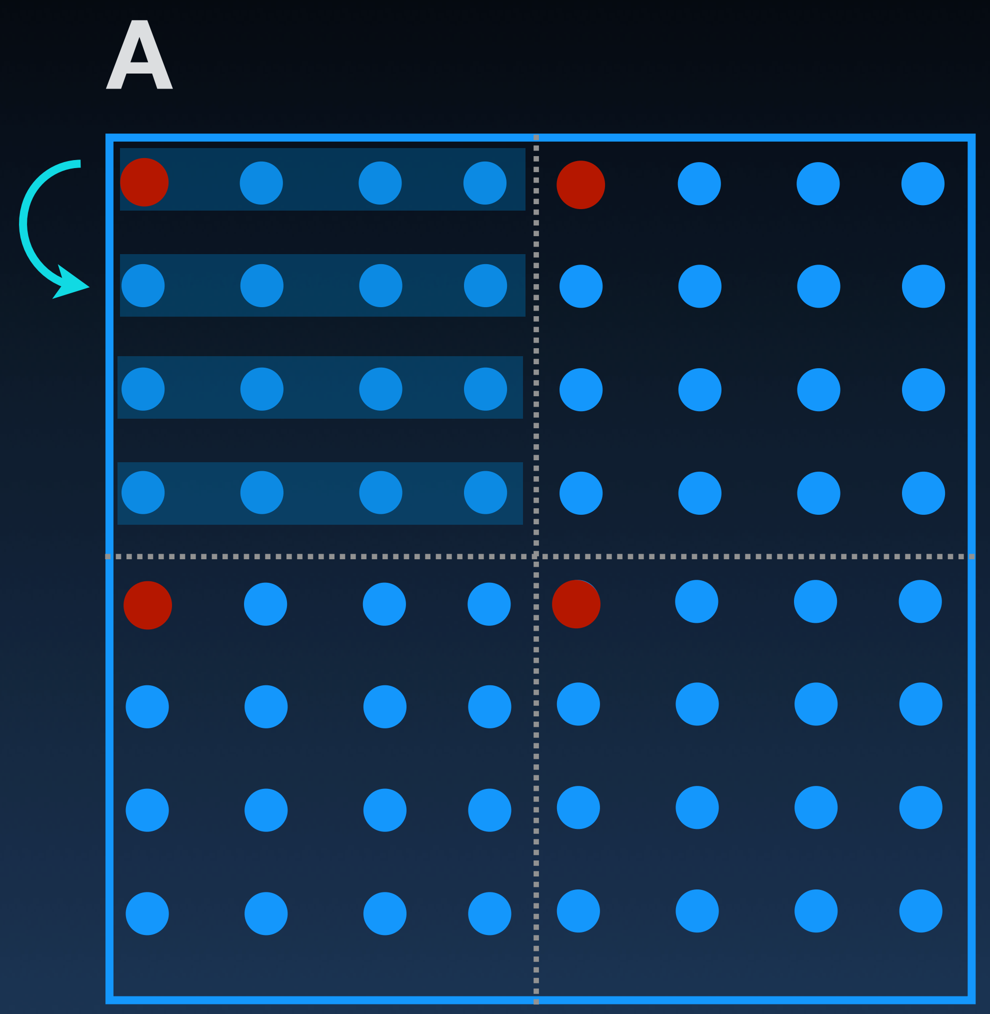
Subodh Kumar

Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                    alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

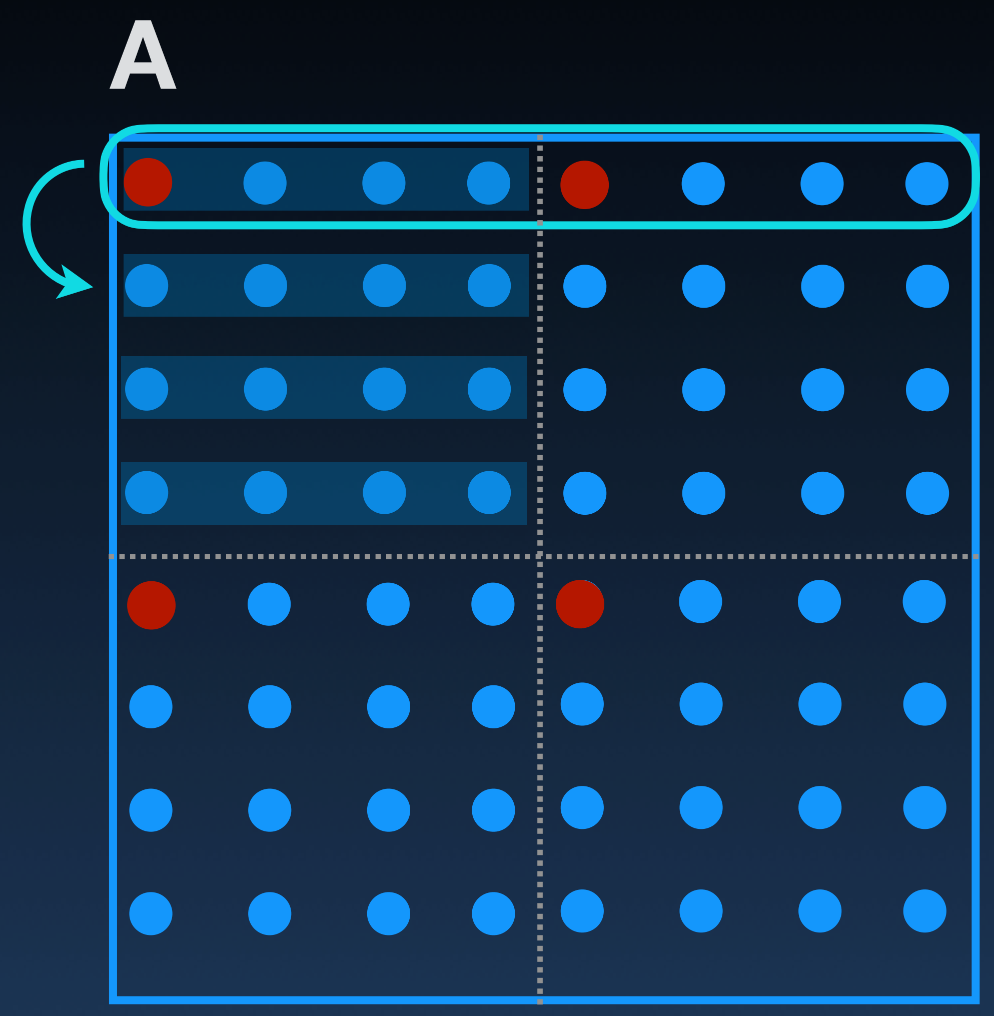
Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                         alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

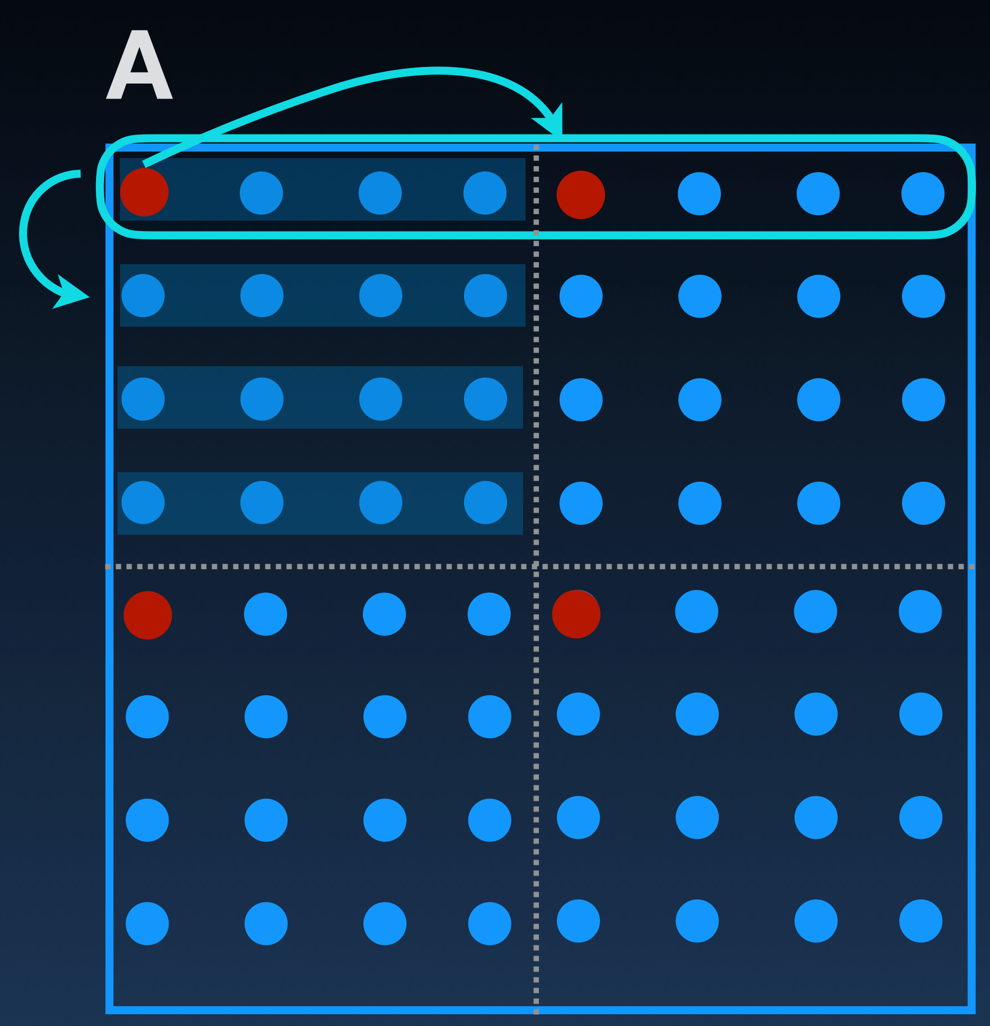
Subodh Kumar

Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                              alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

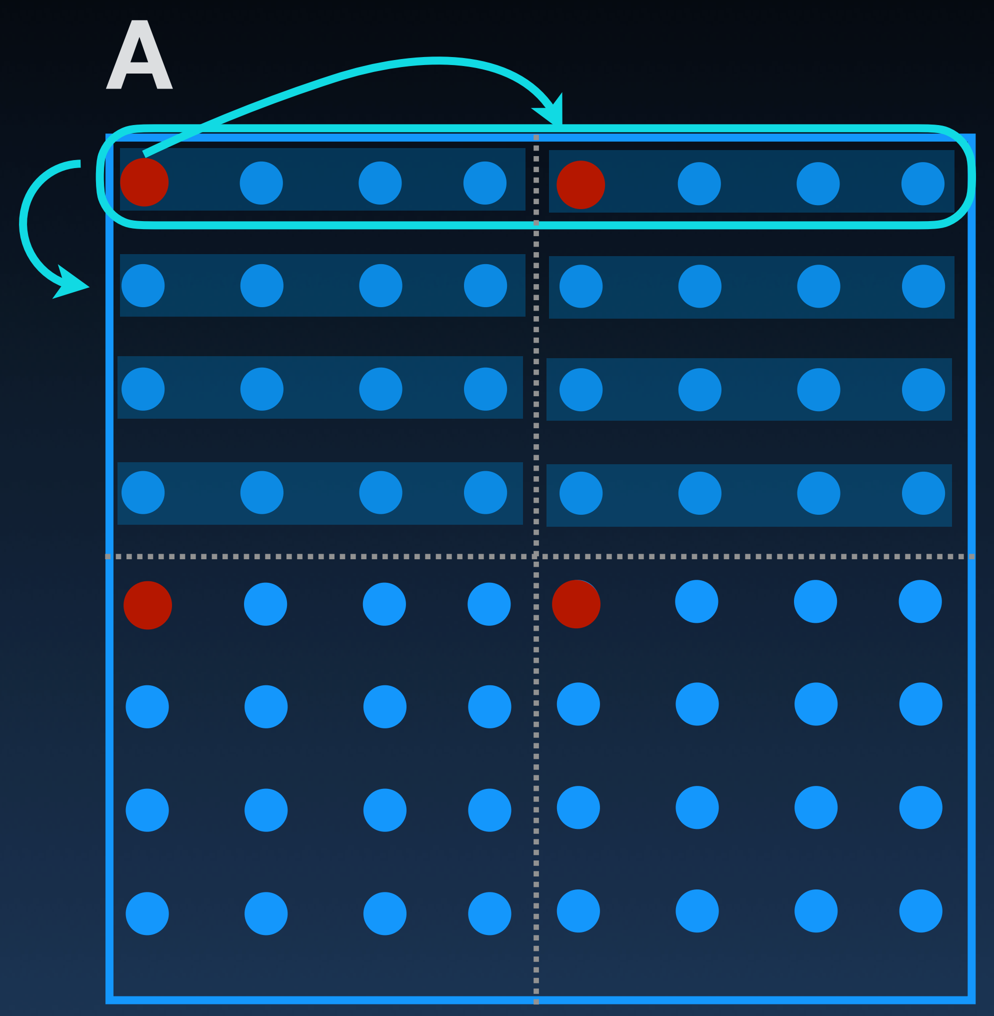
Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
    MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                           alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Subodh Kumar

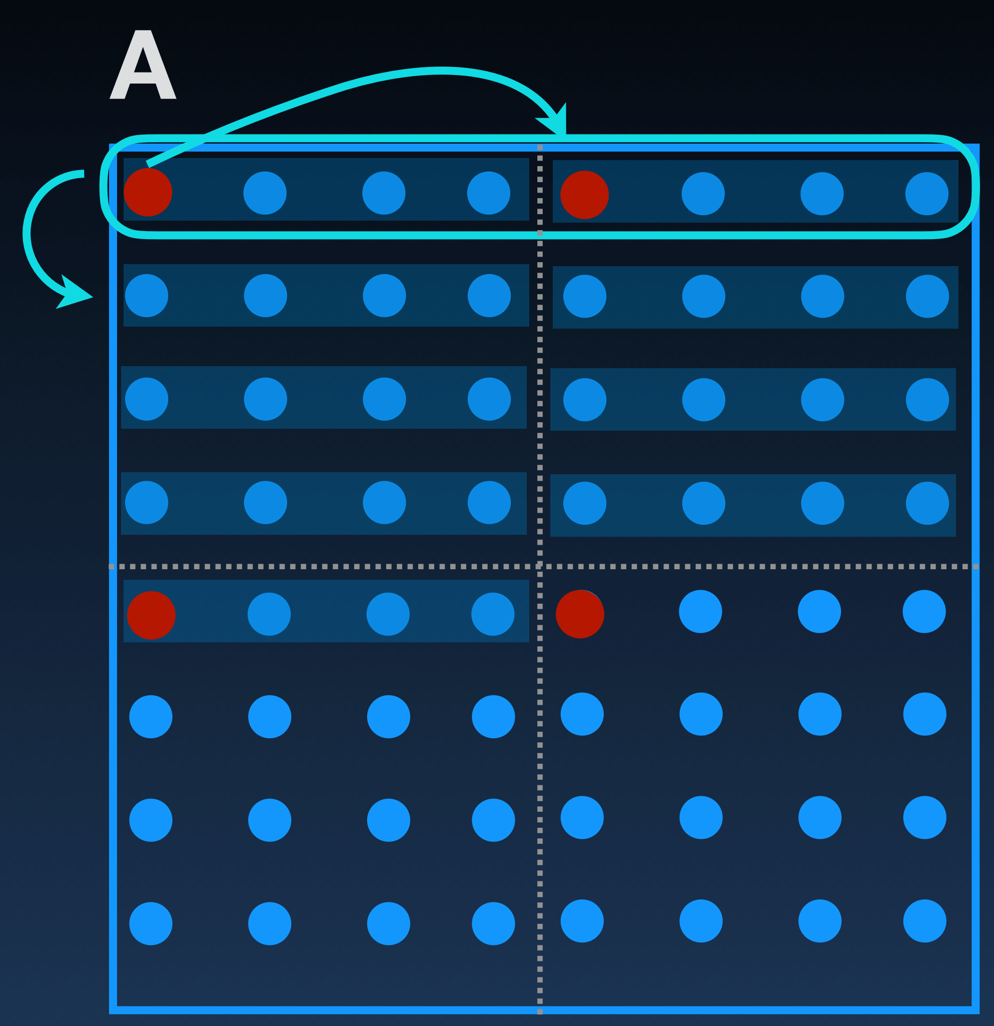
Scatter Matrix



```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

      initialize(A);
      MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
      MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
      MPI_Type_commit(&stype );

      // Setup the Scatter values for the send buffer
      sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
      // Starting locations in A of the four sub matrices in terms of stype
      sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
      MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
      MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                                   alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```

Scatter Matrix

```
double A[8][8], alocal[4][4];
int i, j, r, rank, size, sendcount[4], sdispls[4];
MPI_Datatype stype, vtype;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (size != 4)  MPI_Abort( MPI_COMM_WORLD, 1 );
if (rank == 0) {

    initialize(A);
    MPI_Type_vector(4, 4, 8, MPI_DOUBLE, &vtype );  // 4 sets of 4 doubles, separated by 8
    MPI_Type_create_resized(vtype, 0, 4*sizeof(double), &stype); // Artificial type for scatter
    MPI_Type_commit(&stype );

    // Setup the Scatter values for the send buffer
    sendcount[0] =  sendcount[1] = sendcount[2] = sendcount[3] = 1; // Send one to each
    // Starting locations in A of the four sub matrices in terms of stype
    sdispls[0] = 0; sdispls[1] = 1; sdispls[2] = 8; sdispls[3] = 9;
    MPI_Scatterv(A, sendcount, sdispls, stype,  alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
} else {
     MPI_Scatterv( (void *)0, (void *)0, (void *)0, MPI_DATATYPE_NULL,
                                   alocal, 4*4, MPI_DOUBLE, 0, MPI_COMM_WORLD );
}
```



Subodh Kumar

MPI_Reduce

```
MPI_Reduce(dataArray, resultArray, count,
           type, MPI_SUM, root, com);
```

dataArray	data sent from each processor
Result	stores result of combining operation
count	number of items in each of dataArray, result
MPI_SUM	combining operation, one of a predefined set
root	rank of processor receiving data

- Multiple elements can be reduced in one shot
- Illegal to alias input and output arrays

MPI_Reduce variants

- MPI_Reduce: result is at the root
  - ➡ operation repeated for each element of the input arrays on each processor
- MPI_Allreduce: result is sent out to everyone
- MPI_Reduce_scatter: equivalent to a reduce followed by a scatter
- User defined operations

User-defined reduce operation

```
void rfunction(void *invec, void *inoutvec, int *len,
               MPI_Datatype *datatype){
    // accumulate *len type items of invec into inoutvec
}

MPI_Op op;
MPI_Op_create(rfunction, commute, &op);
MPI_Reduce(inArray, outArray, count, type, op, root, com);
```

Later:

```
MPI_op_free(&op);
```

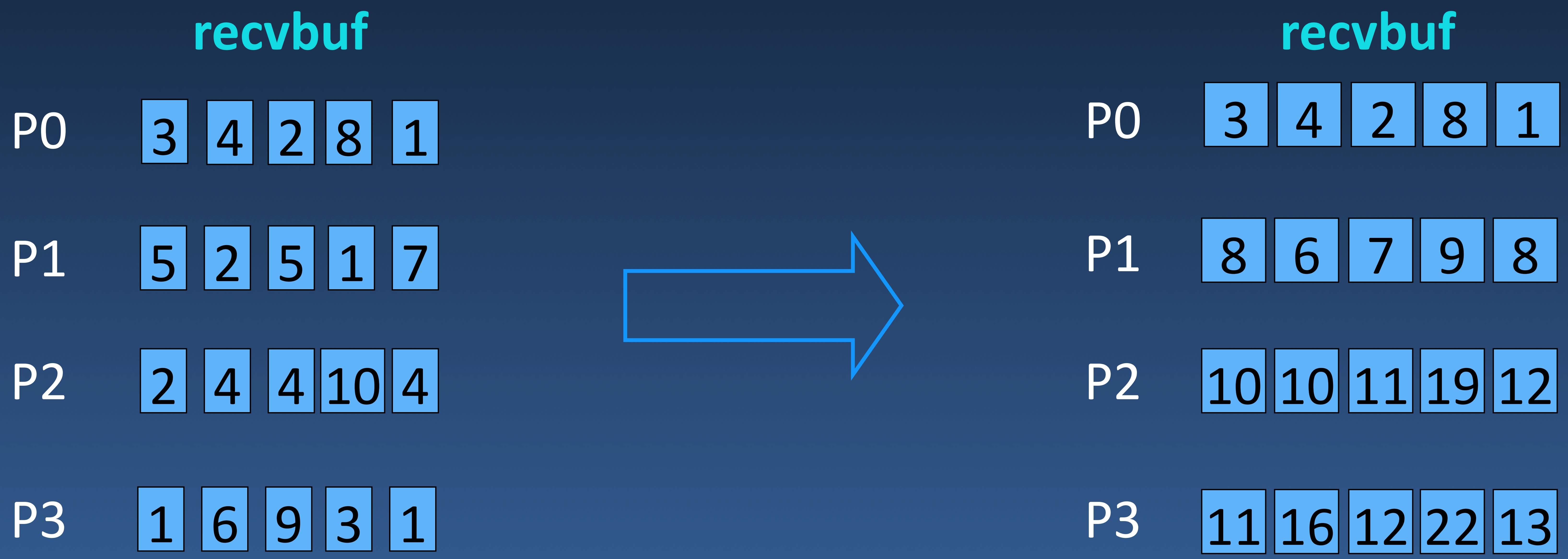
Prefix Scan

```
MPI_Scan(sendbuf, recvbuf, count, datatype, op, comm);
```

- Prefix reduction on data in sendbuf
    - ➡ Multiple prefix ops in one shot
- Returns in the receive buffer of the process i:
    - ➡ reduction of the values in the send buffers of processes 0,...,i (inclusive)
- All ranks must agree on op, datatype, count
- **`MPI_EScan`** for exclusive scan

In-place MPI_Scan

```
MPI_Scan(MPI_IN_PLACE, recvbuf, 5, MPI_INT, MPI_SUM, comm);
```



recvbuf

P0	3	4	2	8	1
P1	5	2	5	1	7
P2	2	4	4	10	4
P3	1	6	9	3	1

recvbuf

P0	3	4	2	8	1
P1	8	6	7	9	8
P2	10	10	11	19	12
P3	11	16	12	22	13

Process Start

- MPI_Comm_Spawn(command, argv, maxprocs, info, root, comm, intercomm, array_of_errcodes)
- The children have their own MPI_COMM_WORLD
- May not return until MPI_INIT has been called in the children
- More efficient to start all processes at once

Remote Memory

```
MPI_Win_create(basemem, size, displ_unit, info, MPI_COMM_WORLD,
&win);
…
MPI_Win_free(&win);
```

MPI_Win

MPI_Info

- Weak synchronization
- Collective call
- Info specifies system-specific information (e.g., memory locking)
  - ➡ Designed for optimizing performance
- See MPI_Alloc_mem/MPI_Win_allocate for basemem allocation

MPI_Put, MPI_Get

- MPI_Put(my_addr, my_count, my_datatype,
  there_rank, there_disp, there_count, there_datatype, win);
  - ➡ Written in the dest window-buffer at address
    - ▸ window_base + disp×disp_unit
  - ➡ Must fit in the target buffer
  - ➡ there_datatype defined on the “putter”
    - ▸ But refers to memory “there”
    - ▸ Usually defined on both sides

MPI_Put, MPI_Get

- MPI_Put(my_addr, my_count, my_datatype,
  there_rank, there_disp, there_count, there_datatype, win);
  - ➡ Written in the dest window-buffer at address
    - ▶ window_base + disp×disp_unit
  - ➡ Must fit in the target buffer
  - ➡ there_datatype defined on the "putter"
    - ▶ But refers to memory "there"
    - ▶ Usually defined on both sides

MPI_Get does the reverse: there ➜ my

Also see:
MPI_Accumulate
performs an "op" at destination

Subodh Kumar

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

```
int winbuf[10];
MPI_Win windo;
MPI_Win_create(winbuf, 10*sizeof(int), sizeof(int),
                    MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective

if(rank == 1) {
        int lbuf[5];
        initialize(lbuf);
        MPI_Put(lbuf, 5, MPI_INT, 0, 5, 5, MPI_INT, windo);
}

MPI_Win_fence(0, windo);// Wait for MPI_Put complete

if(my_rank == 0)
        use(winbuf+5);
```

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

```
int winbuf[10];
MPI_Win windo;
MPI_Win_create(winbuf, 10*sizeof(int), sizeof(int),
                    MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective

if(rank == 1) {
      int lbuf[5];
      initialize(lbuf);
      MPI_Put(lbuf, 5, MPI_INT, 0, 5, 5, MPI_INT, windo);
}

MPI_Win_fence(0, windo);// Wait for MPI_Put complete

if(my_rank == 0)
      use(winbuf+5);
```

“Assert”

Remote Memory Synchronization

- MPI_Win_fence
- MPI_Win_flush
- MPI_Win_lock
- MPI_Win_unlock
- MPI_Win_start
- MPI_Win_complete
- MPI_Win_post
- MPI_Win_Wait
- MPI_Win_Test

Look these up

```
int winbuf[10];
MPI_Win windo;
MPI_Win_create(winbuf, 10*sizeof(int), sizeof(int),
                    MPI_INFO_NULL, MPI_COMM_WORLD, &windo);
MPI_Win_fence(0, windo); // Collective
```

“Assert”

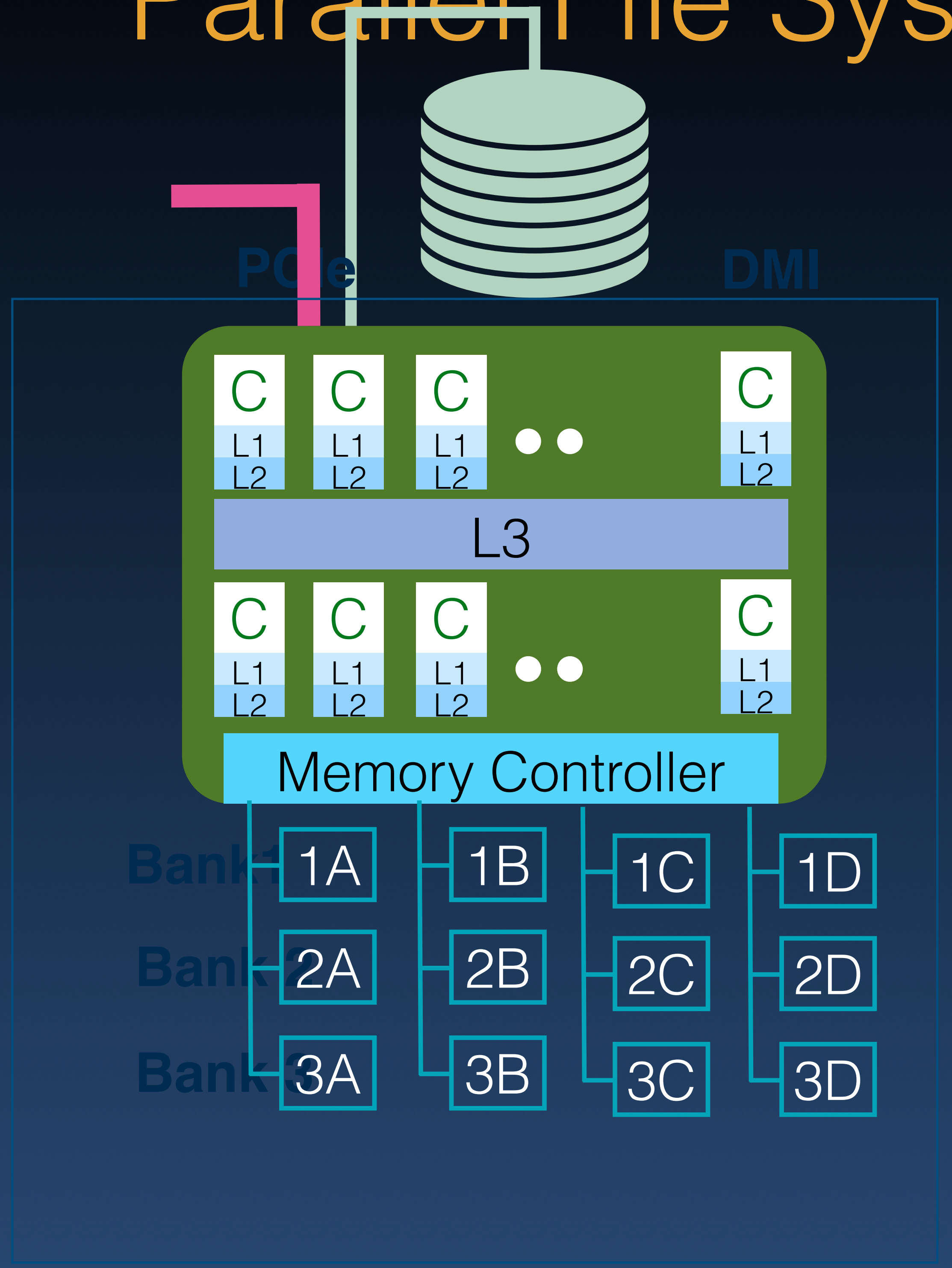
```
if(rank == 1) {
        int lbuf[5];
        initialize(lbuf);
        MPI_Put(lbuf, 5, MPI_INT, 0, 5, 5, MPI_INT, windo);
}

MPI_Win_fence(0, windo);// Wait for MPI_Put complete

if(my_rank == 0)
        use(winbuf+5);
```

Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access

Parallel File Systems

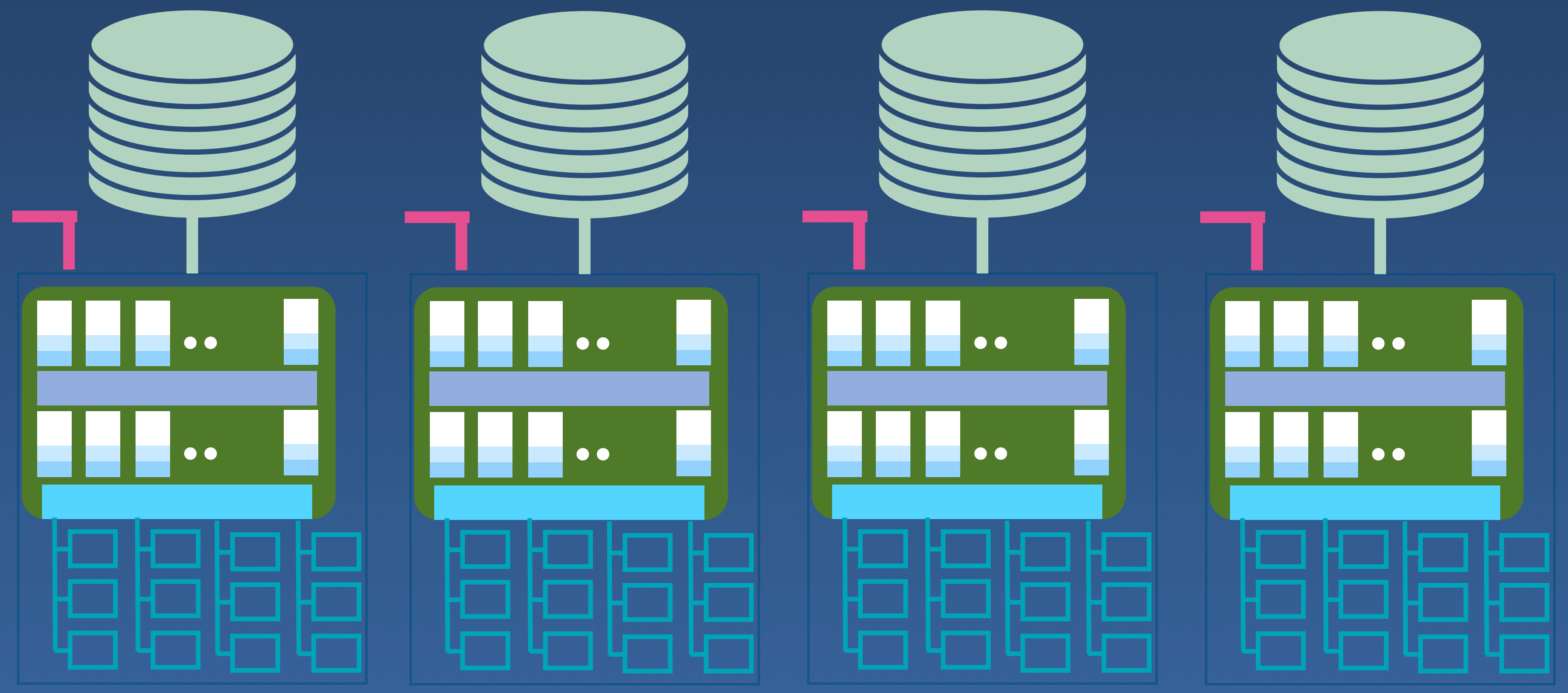
- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



Subodh Kumar

Parallel File Systems

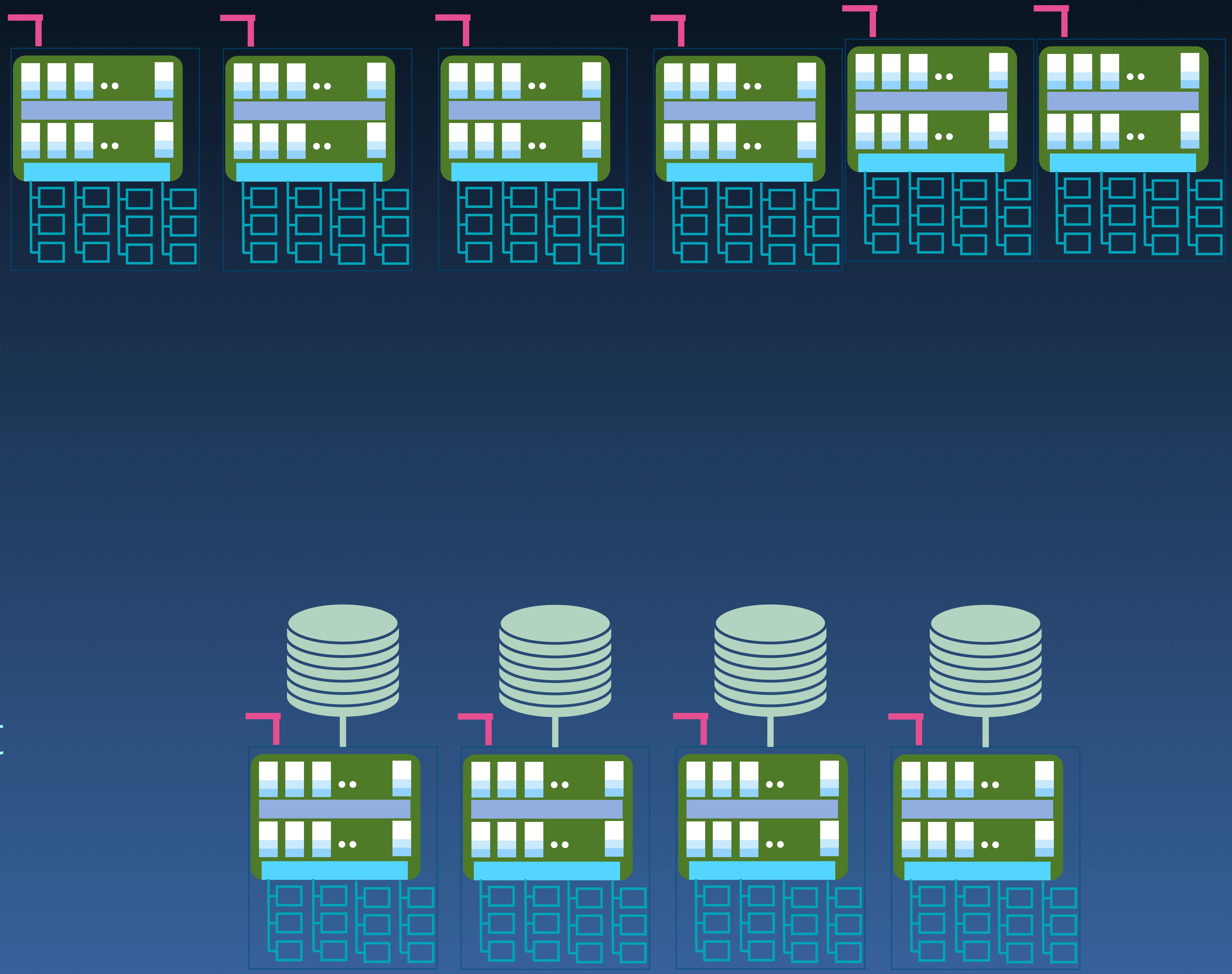
- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



Subodh Kumar

Parallel File Systems

- Multiple disk servers
  - ➡ With multiple network paths to disks
- Designed for performance
  - ➡ Large block sizes (~MB)
  - ➡ Parallel fetch
  - ➡ Concurrent I/O
  - ➡ Metadata operations less performant
- Traditional file API
  - ➡ Additional APIs for faster access



Subodh Kumar

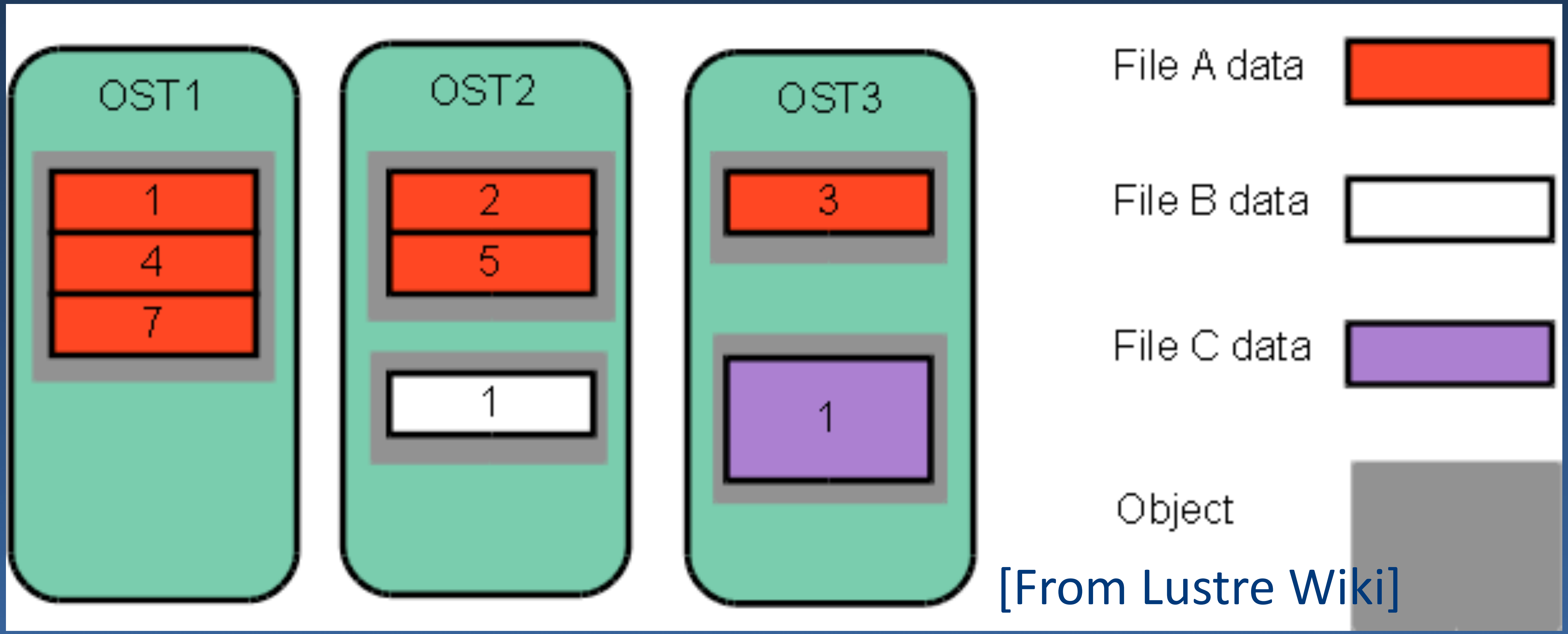
PFS Striping

- Configuration per file
  - ➡ number of stripes, stripe size, and OSTs to use

Stripe counts

File A: 3
File B: 1
File C: 1

Stripe size of File C is larger



[From Lustre Wiki]

PFS Striping

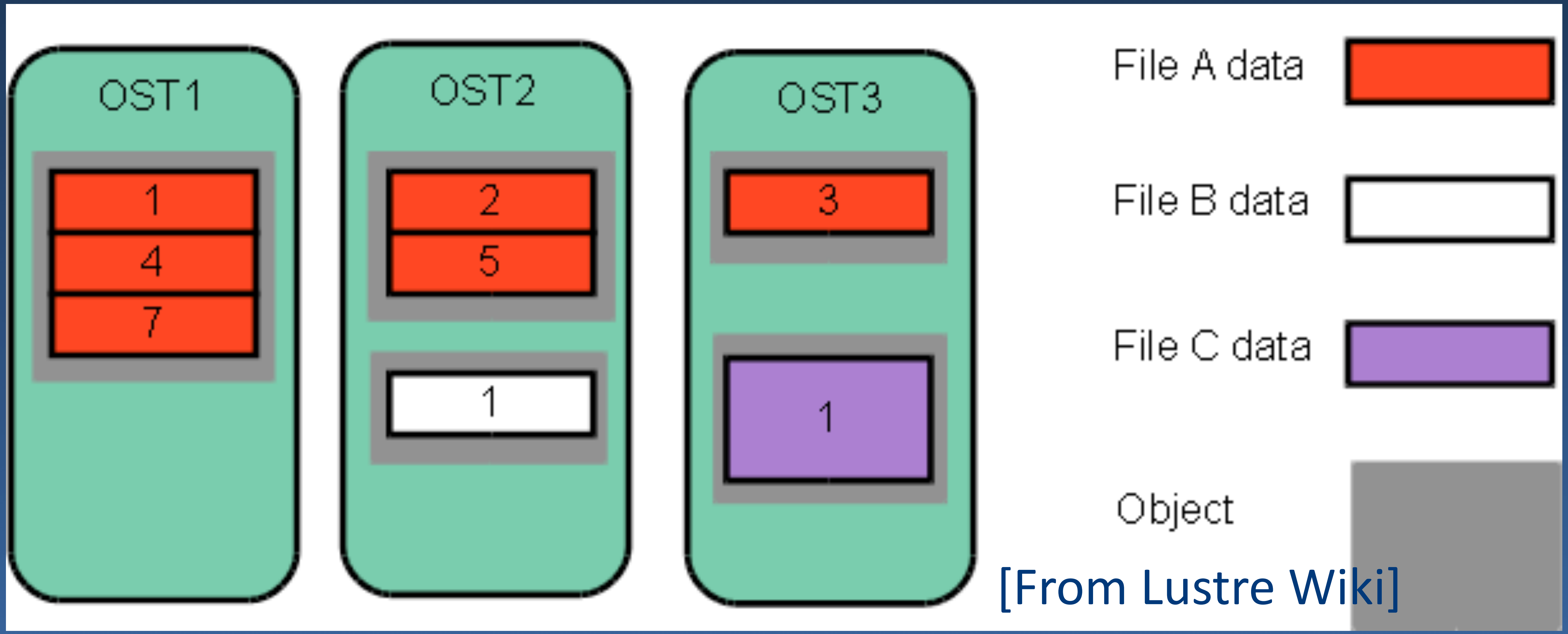
- Configuration per file
  - ➡ number of stripes, stripe size, and OSTs to use

Stripe counts

File A: 3
File B: 1
File C: 1

```
> lfs getstripe <filename>
> lfs setstripe <dirname>
```

Stripe size of File C is larger



[From Lustre Wiki]

MPI-I/O

```
MPI_File fh;
MPI_Status status;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
bufsize = FILESIZE/nprocs;
nints = bufsize/sizeof(int);
MPI_File_open(MPI_COMM_WORLD, “file", MPI_MODE_RDONLY,
              MPI_INFO_NULL,  &fh);
MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
MPI_File_read(fh, buf, nints, MPI_INT, &status);
MPI_File_close(&fh);
```

File View

- 3-tuple: <displacement, etype, filetype>
  - ➡ byte displacement from the start of the file
  - ➡ etype: data unit type
  - ➡ filetype: portion of the file visible to the process
- MPI_File_set_view

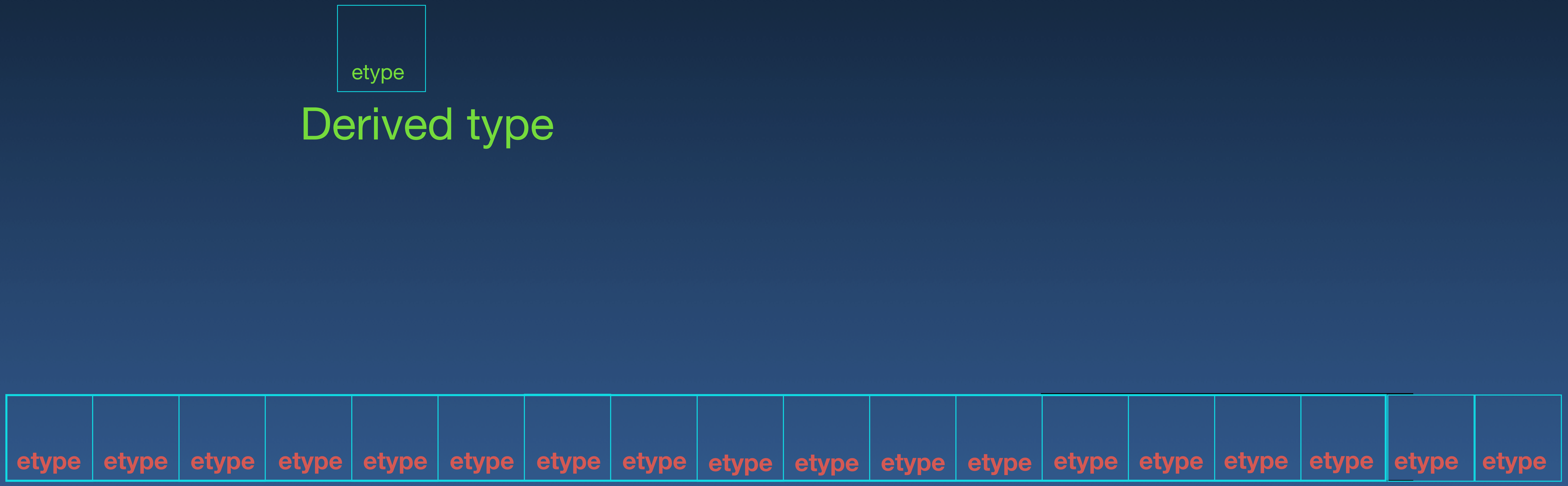
File View

```
int MPI_File_set_view(
    MPI_File fh,
    MPI_Offset disp,          // in bytes
    MPI_Datatype etype,       // file’s a sequence of etypes
    MPI_Datatype filetype,    // interpret as filetypes
    char *datarep,
    MPI_Info info)
```

File View

Derived type

File View



etype

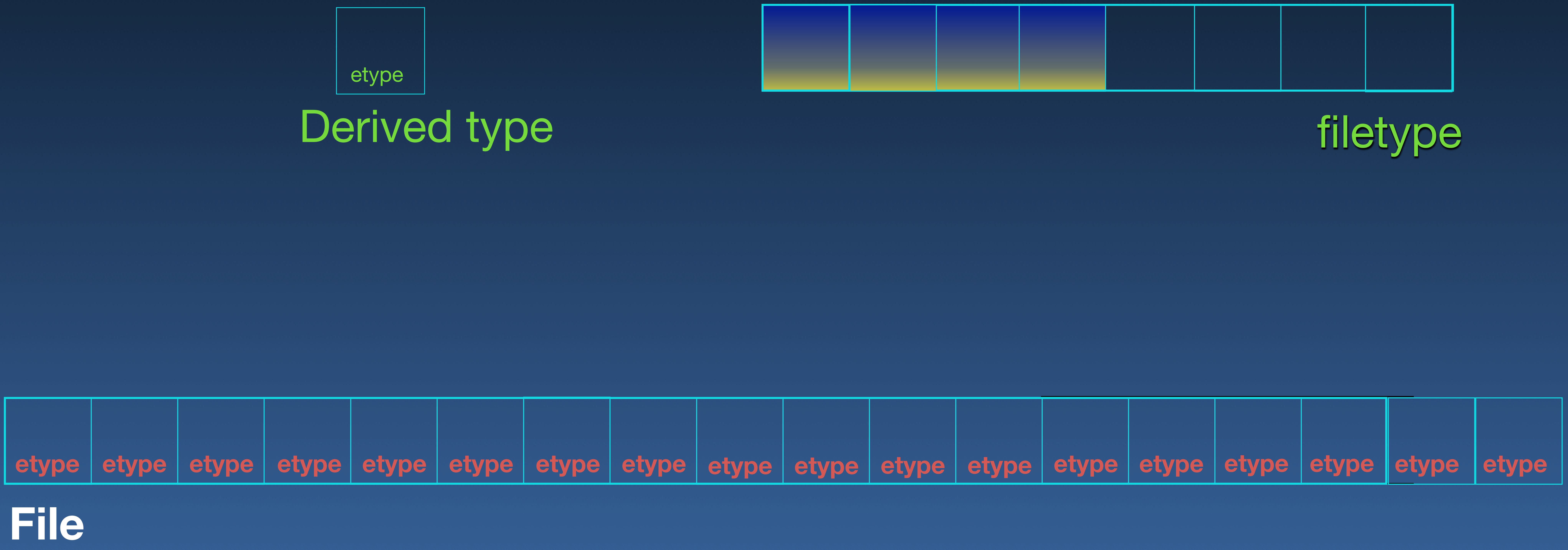
Derived type

etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype

File

Subodh Kumar

File View



etype

Derived type

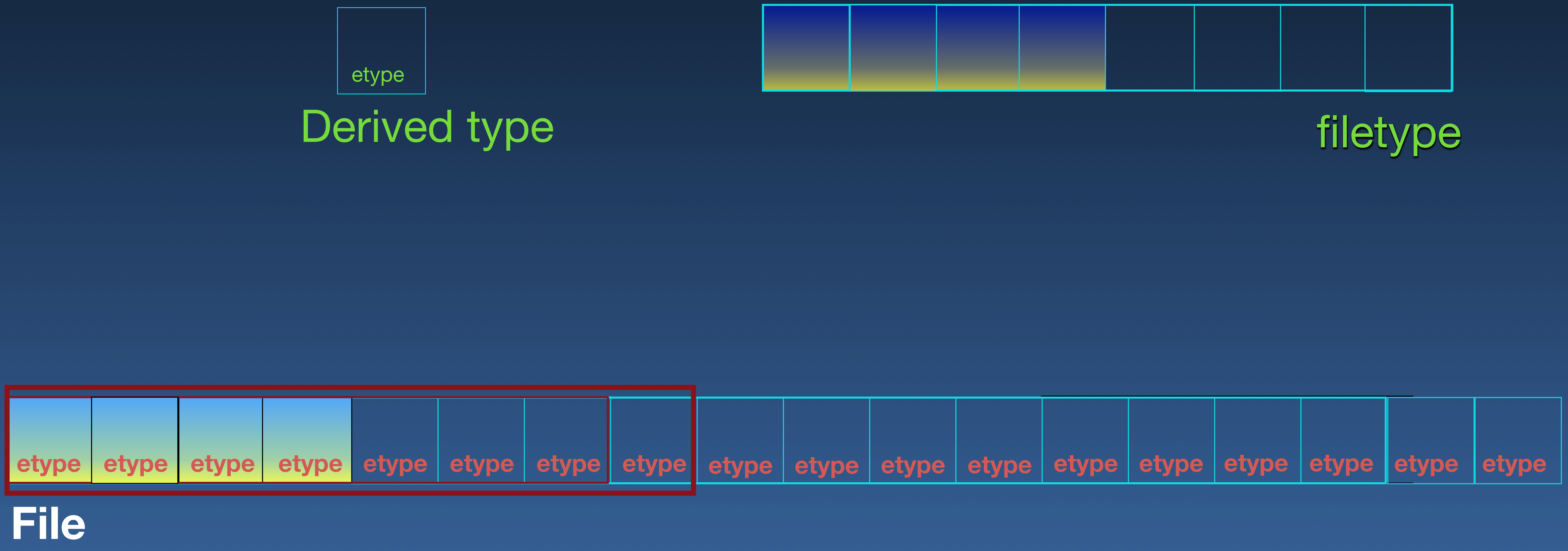
filetype

etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype

File

Subodh Kumar

File View



etype

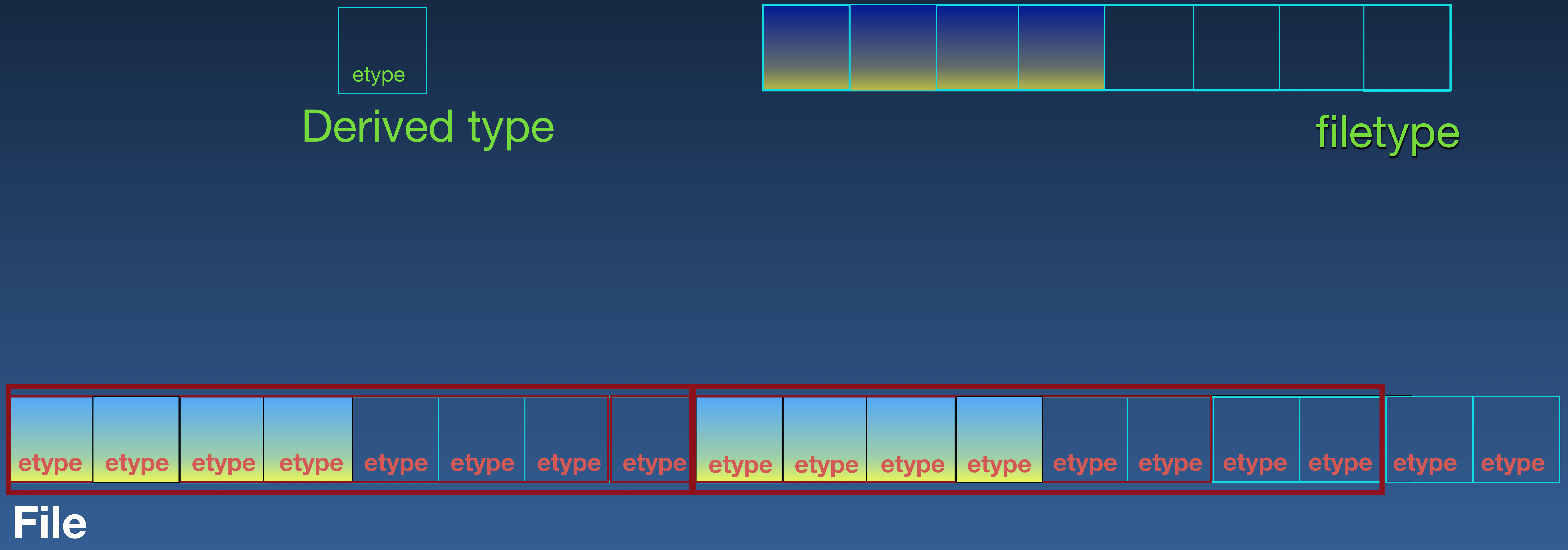
Derived type

filetype

etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype

File

File View



etype

Derived type

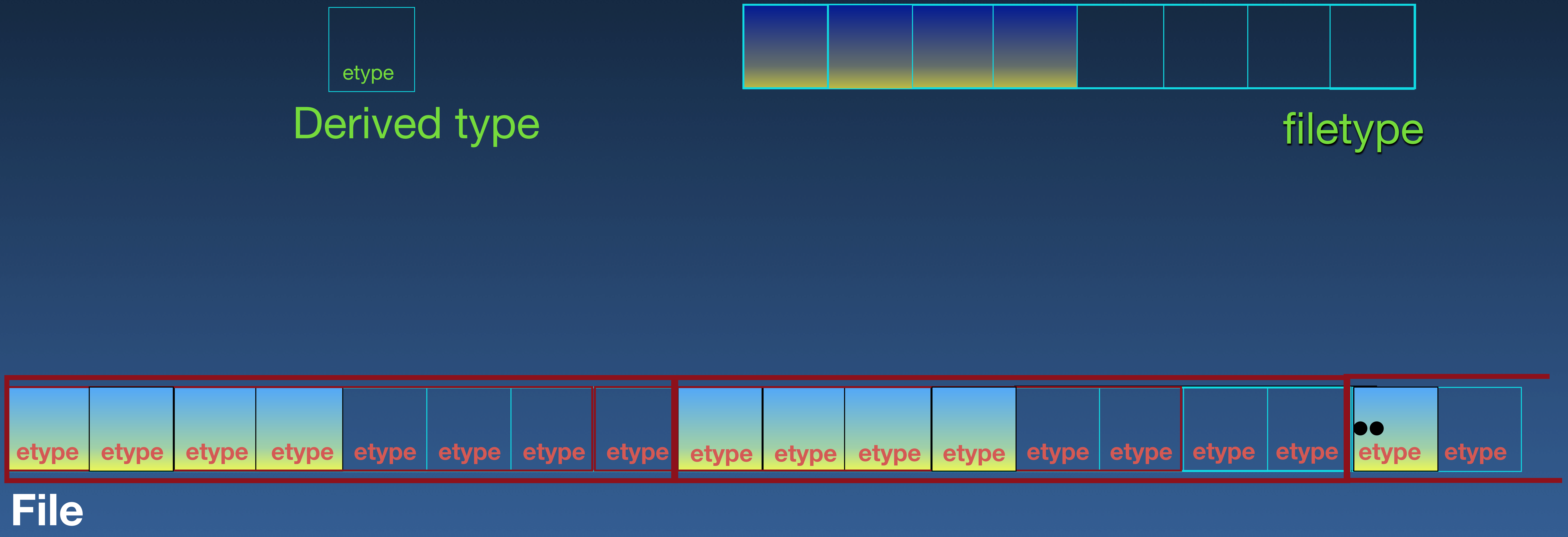
filetype

etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype etype

File

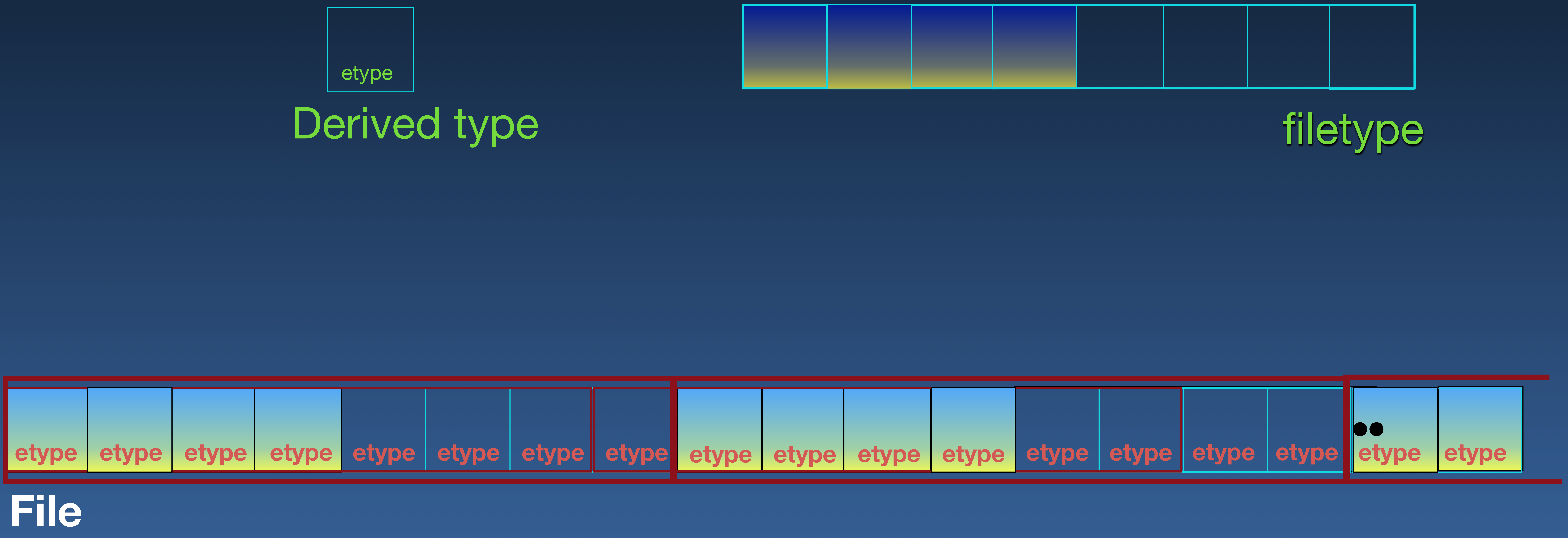
Subodh Kumar

File View



Subodh Kumar

File View



Subodh Kumar

Example: Write in Set View

```
MPI_File pfile;

for (i=0; i<BUFSIZE; i++)
    buf[i] = myrank * BUFSIZE + i;
MPI_File_open(MPI_COMM_WORLD, "file",
            MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &pfile);
MPI_File_set_view(pfile, myrank * BUFSIZE * sizeof(int), MPI_INT, MPI_INT,
                    "native", MPI_INFO_NULL);
MPI_File_write(pfile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
MPI_File_close(&pfile);
```

Blocking, Collective

Blocking, Individual

IO Variants

- Location

  MPI_File_read_**at**(fh, offset, buffer, count, datatype, &status)

- Non-blocking

  MPI_File_**i**read(fh, buffer, count, datatype, &request)

- Collective

  MPI_File_read_**all**(fh, buffer, count, datatype, &status)

- Shared File pointer (Common data IO)

  MPI_File_read_**shared**(fh, buffer, count, datatype, &status) // Not collective

  MPI_File_read_**ordered** (fh, buffer, count, datatype, &status) // Collective

IO Consistency

- Writes from one process become visible to others at arbitrary times

MPI_File_set_atomicity (MPI_File fh, int flag);

- ➡ Collective

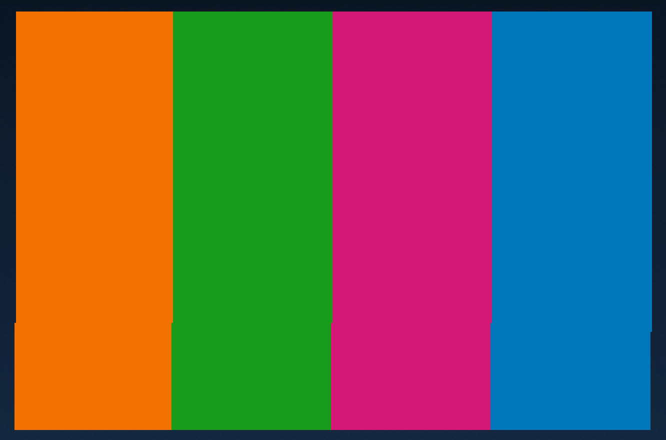
MPI_File_sync (MPI_File fh);

- ➡ Collective
- ➡ Flush all writes

Example: Collective IO

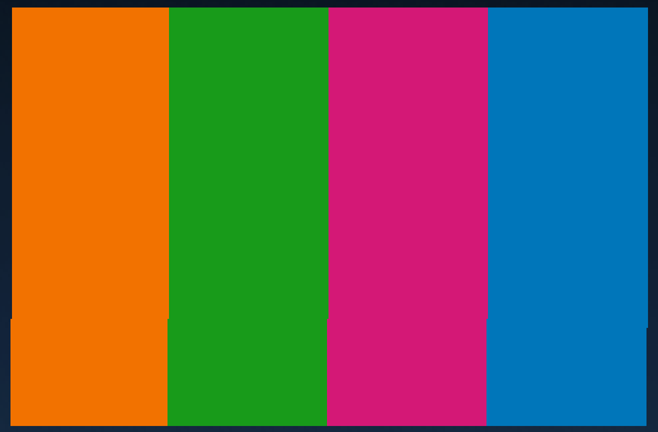
```
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_File_open(MPI_COMM_WORLD, "file", MPI_MODE_RDWR|MPI_MODE_CREATE,
                        MPI_INFO_NULL, &fh );

MPI_File_write_ordered( fh, buf, 1, MPI_INT, &status );
MPI_Barrier(MPI_COMM_WORLD);                         // Let all writes complete

MPI_File_seek( fh, 0, MPI_SEEK_SET );                // Rewind to the top
MPI_File_read_all( fh, buf, size, MPI_INT, &status ); // Everyone reads size ints

MPI_File_seek_shared(fh, 0, MPI_SEEK_SET );          // Rewind to the top again
MPI_File_read_ordered(fh, buf, 1, MPI_INT, &status ); // Read one int in round-robin order

MPI_File_close( &fh );
```

Example: Views in IO

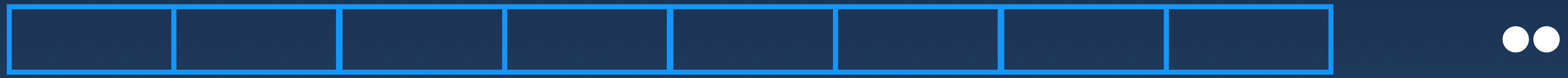


```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
        displ[i] = rank + i * size;
        blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,”file”, MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, “native”, MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close( &fh );
```

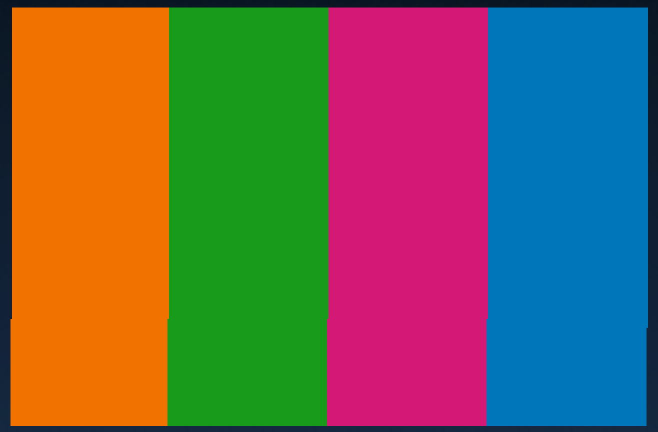
Example: Views in IO



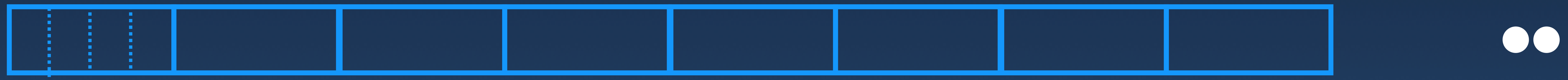
```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
        displ[i] = rank + i * size;
        blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,”file”, MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, “native”, MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close( &fh );
```



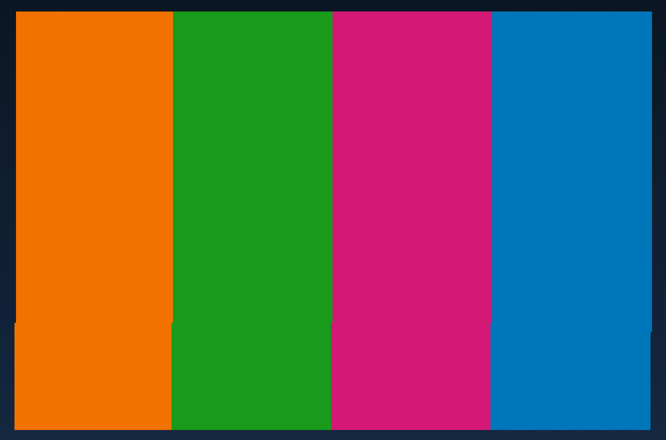
Example: Views in IO



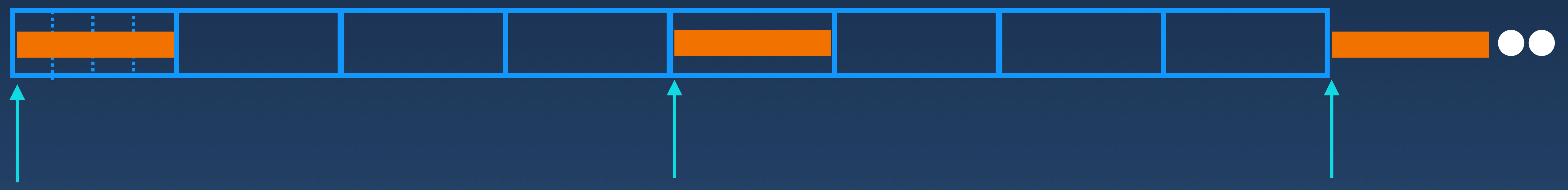
```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
        displ[i] = rank + i * size;
        blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,”file”, MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, “native”, MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close( &fh );
```



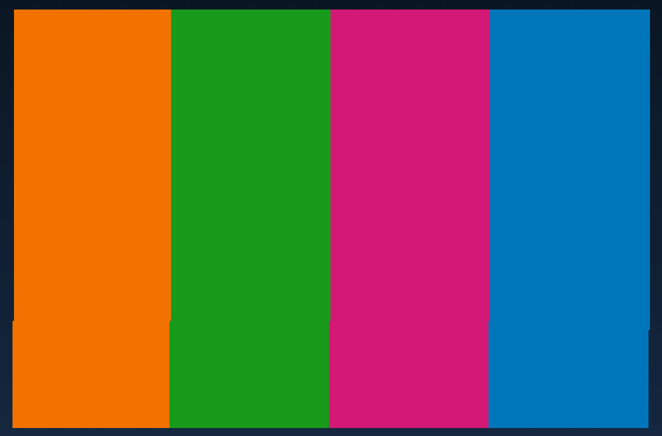
Example: Views in IO



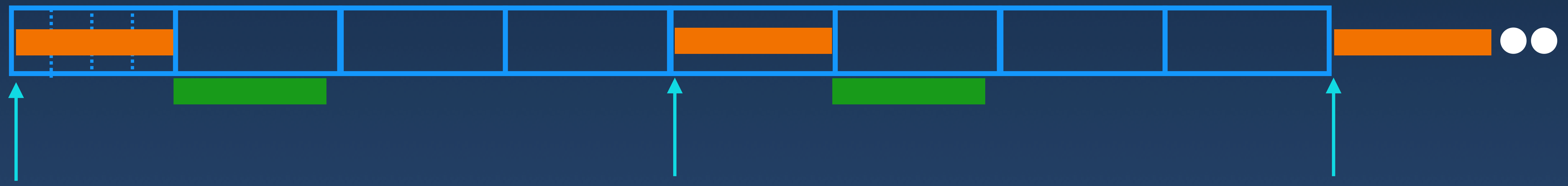
```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
        displ[i] = rank + i * size;
        blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,”file”, MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, “native”, MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close( &fh );
```



Example: Views in IO



```
MPI_Comm_rank(MPI_COMM_WORLD, &rank );
MPI_Comm_size(MPI_COMM_WORLD, &size );
MPI_Type_contiguous (4, MPI_DOUBLE, &etype);
MPI_Type_commit ( &etype );

for ( i = 0; i < 4; i++) {
      displ[i] = rank + i * size;
      blocklength[i] = 1;
}
MPI_Type_indexed (4, blocklength, displ, etype, &filetype );
MPI_Type_commit ( &filetype );

MPI_File_open ( MPI_COMM_WORLD,”file”, MPI_MODE_RDONLY, MPI_INFO_NULL , &fh);
MPI_File_set_view (fh, 0, etype, filetype, “native”, MPI_INFO_NULL);
MPI_File_read_all (fh, buf, 16, etype, &status );
MPI_File_close( &fh );
```



Subodh Kumar