Paralel ve Dağıtık Programlama

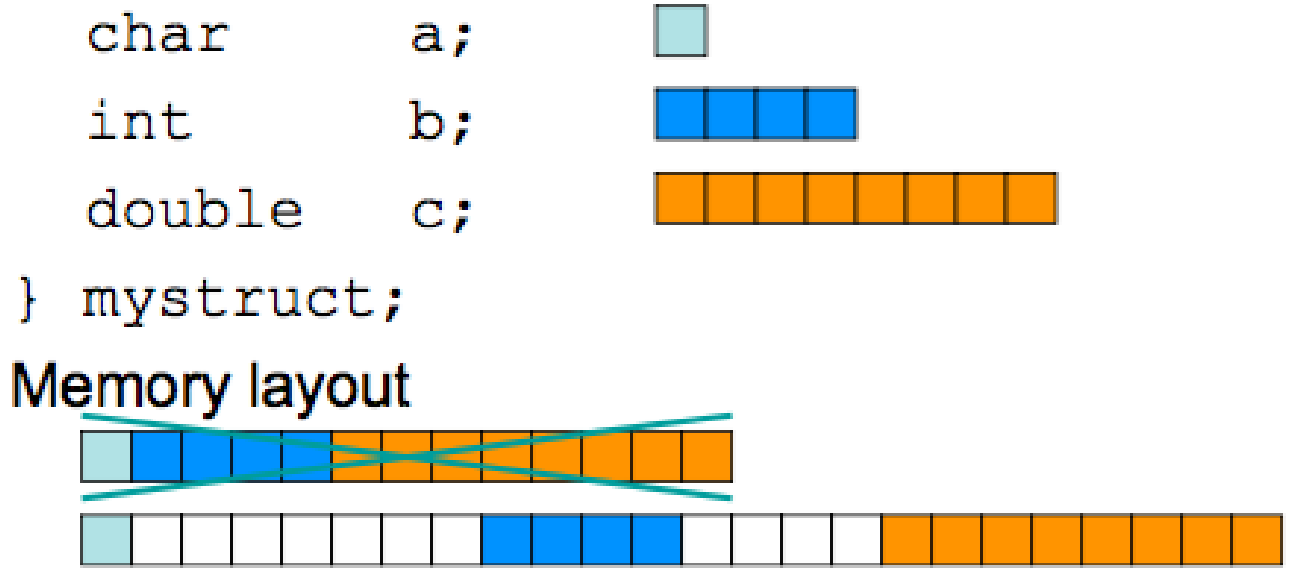
MPI'da Türetilmiş Veri Tipleri

Dr. Öğr. Üyesi Süha Tuna

Türetilmiş veri tipleri

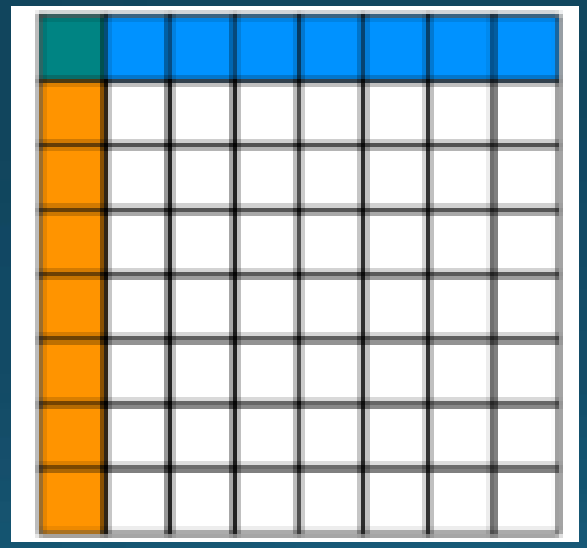
- Amaç: Kullanıcının sıkça kullandığı ve işine gelen bellek düzeni için **arayüz** oluşturmak

```
typedef struct {
   char      a;
   int       b;
   double    c;
} mystruct;
```

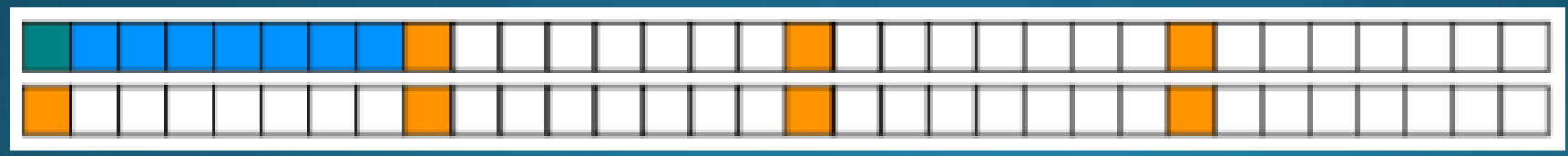


Türetilmiş veri tipi örnekleri

- Bir matrisin satır ya da sütunu

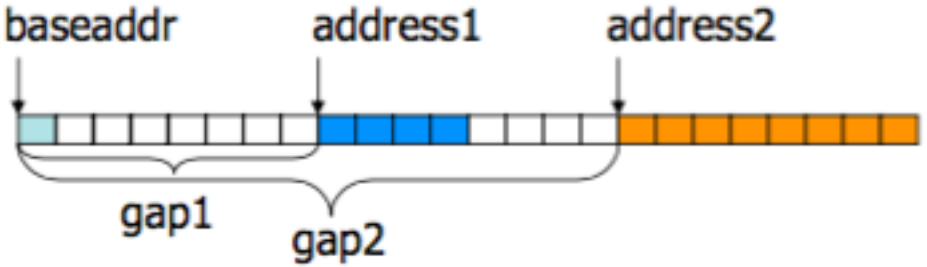


- C'deki ilgili bellek düzenleri



Süreksiz veri yapılarının anlatımı

```
typedef struct {
   char        a;
   int         b;
   double      c;
} mystruct;
```



Liste yardımıyla:

```
<baseaddr, sizeof(char)>
<address1, sizeof(int)>
<address2, sizeof(double)>
```

ya da

```
<baseaddr,       sizeof(char)>
<baseaddr+gap1, sizeof(int)>
<baseaddr+gap2, sizeof(double)>
```

MPI terminolojisi

```
<baseaddr,          1,          MPI_CHAR>
<baseaddr+gap1,     1,          MPI_INT>
<baseaddr+gap2,     1,          MPI_DOUBLE>
```

- MPI ile struct oluşturma:

```
int MPI_Type_create_struct(int count,
     const int array_of_blocklengths[],
     const MPI_Aint array_of_displacements[],
     const MPI_Datatype array_of_types[],
     MPI_Datatype *newtype)
```

MPI_Type_create_struct

- `MPI_Aint`
  - Bir MPI Adres tamsayı değişkeni
  - Bir bellek adresi tutabilen tamsayı değişken

- Bir elemanın adresini alma:
  - `MPI_Get_address(void *element, MPI_Aint *address);`

Tip imzası ve tip haritası

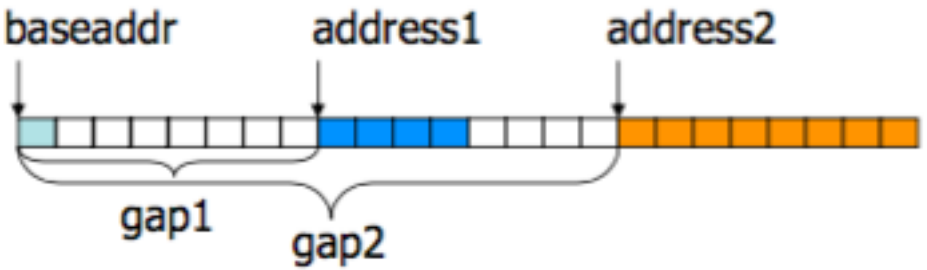
- Tip imzası (ing: type signature)
  - `typesig(mystruct) = {char, int, double}`
- Tip haritası (ing: type map)
  - `typemap(mystruct) = {(char,0),(int,8),(double,16)}`
- MPI'ın tip eşleme kuralı: Gönderici ve alıcının tip imzası eşleşmeli

Commit ve Free işlemleri

- Bir veri türü iletişim ya da I/O için kullanılmadan önce commit edilmelidir.
  - `MPI_Type_commit (MPI_Datatype *datatype);`
- Kullanıldıktan sonra işi biten veri türleri free edilmelidir.
  - `MPI_Type_free (MPI_Datatype *datatype);`
- Ön tanımlaması yapılmayan değişkenler free edilemez.

struct örneği

```
typedef struct {
   char        a;
   int         b;
   double      c;
} mystruct;
```




```
mystruct mydata;
MPI_Address ( &mydata, &baseaddr);
MPI_Address ( &mydata.b, &addr1);
MPI_Address ( &mydata.c, &addr2);
displ[0] = 0;
displ[1] = addr1 - baseaddr;
displ[2] = addr2 - baseaddr;

MPI_Type_struct ( 3, blength, displ, dtype, &newtype );
MPI_Type_commit ( &newtype );
```

```
dtype[0] = MPI_CHAR; d
type[1] = MPI_INT;
dtype[2] = MPI_DOUBLE;
blength[0] = 1;
blength[1] = 1;
blength[2] = 1;
```

Bu durumda,

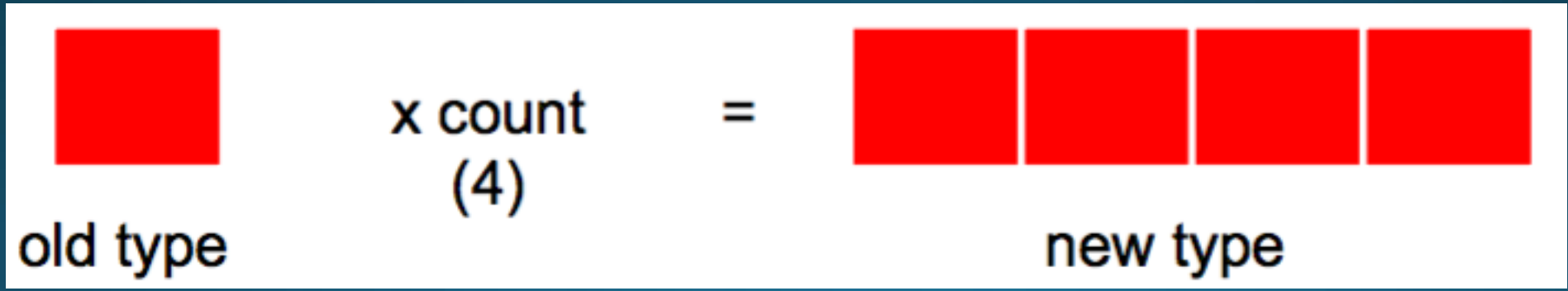
- `MPI_Type_struct` ile bellekte istenilen her düzen oluşturulabilir.

- O zaman diğer MPI veri tiplerine neden ihtiyaç var?
  - Çünkü bazı veri türlerinin anlatımı oldukça karmaşık olabilir
  - Uygunluk ve kolaylık

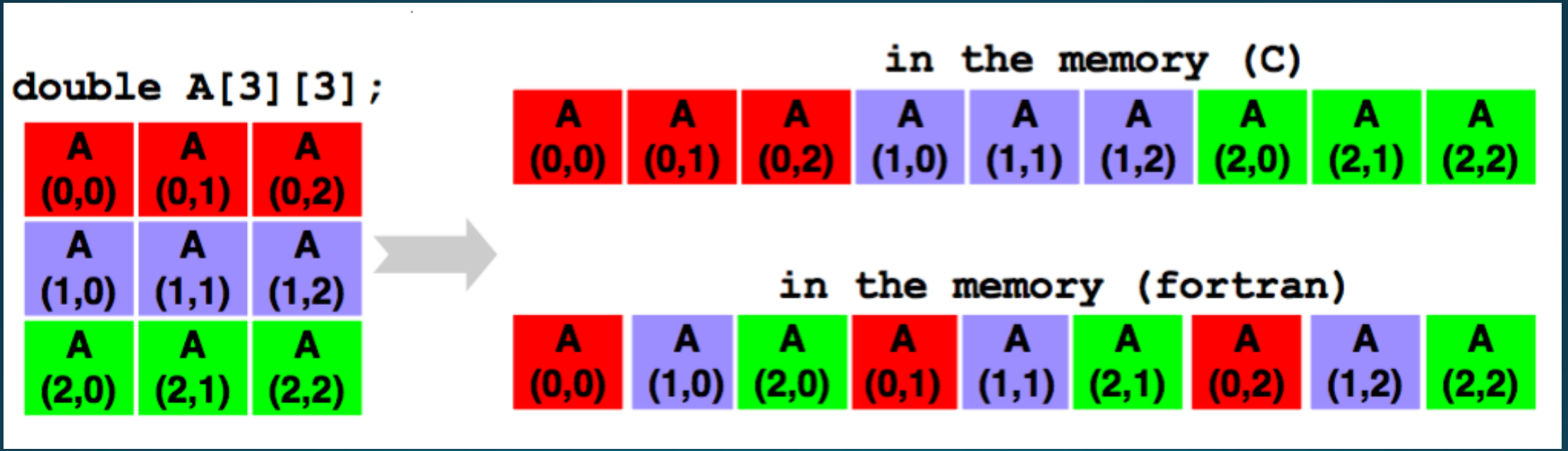
MPI_Type_contiguous

- Aynı tipten ve bellekte ardışıl olarak duran veriden yeni bir veri türü oluşturmak için kullanılır

```
MPI_Type_contiguous( int count,
      MPI_Datatype old_type, MPI_Datatype *newtype)
```

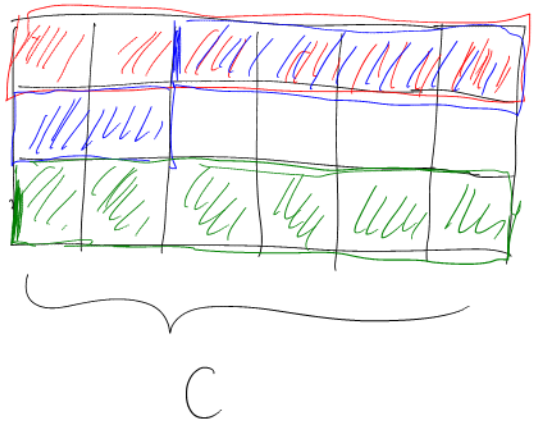


MPI_Type_contiguous (devam)



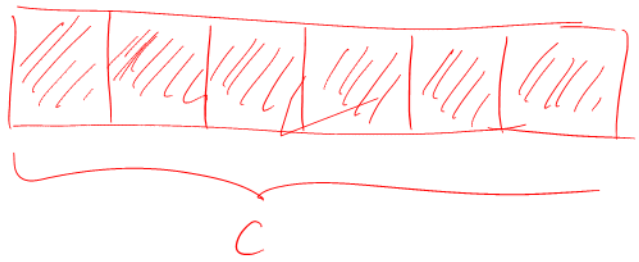

```
MPI_Type_contiguous(3, MPI_DOUBLE, &rowType);
```

A =



Each row consists of C number of MPI_FLOATs.

rowType



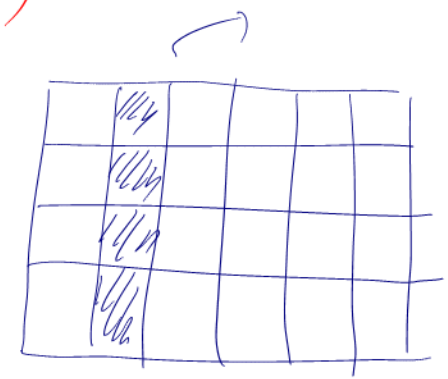
MPI_Type_contiguous(C, MPI_FLOAT, &rowType)

MPI_Type_contiguous'la yapamayız.

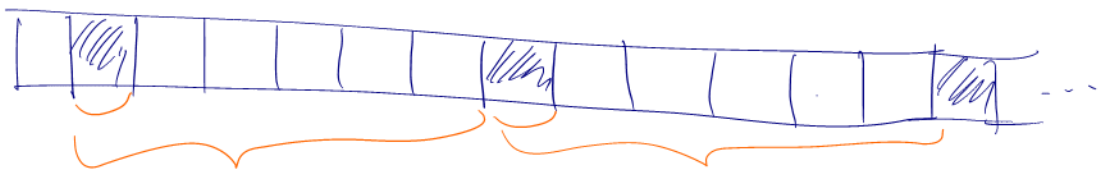
MPI_Send(&A[0], 1, rowType, ...

MPI_Send(&A[2], 1, rowType, ...

MPI_Send(&A[2*C], 1, rowType, ...



MPI_Type_vector ←

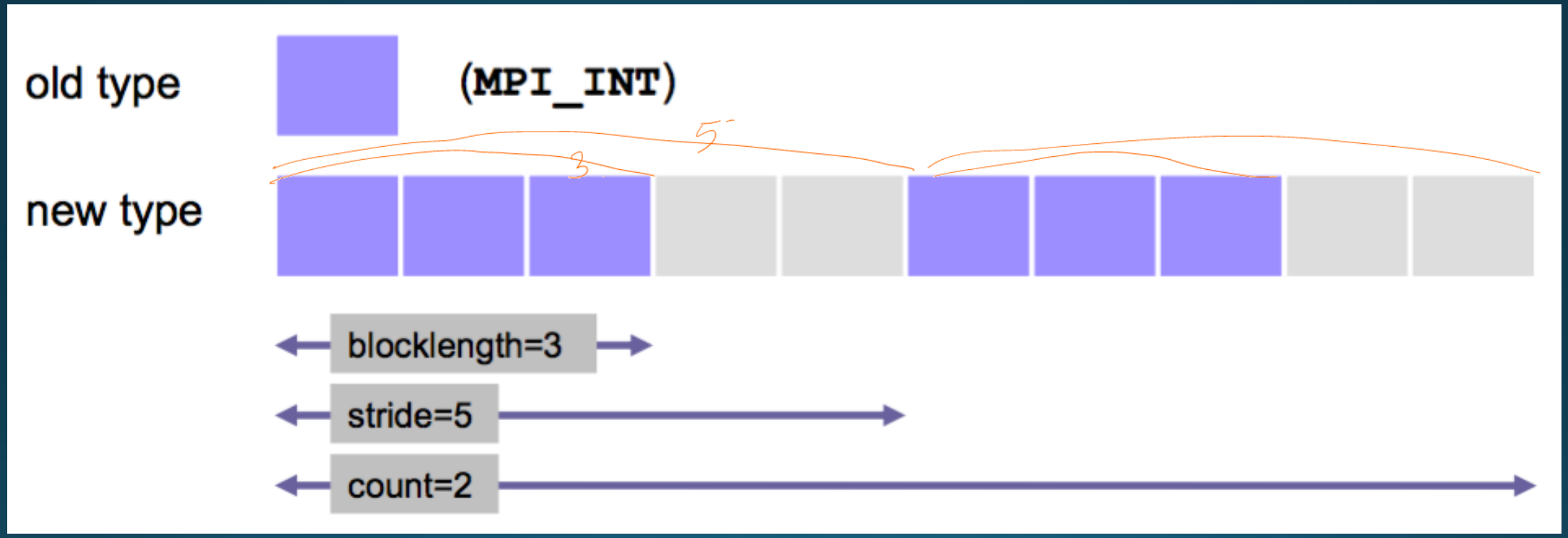


MPI_Type_vector

- Eğer bir arada bulunması istenen verilerin aralarında düzenli boşluklar varsa bu MPI fonksiyonu kullanılır.

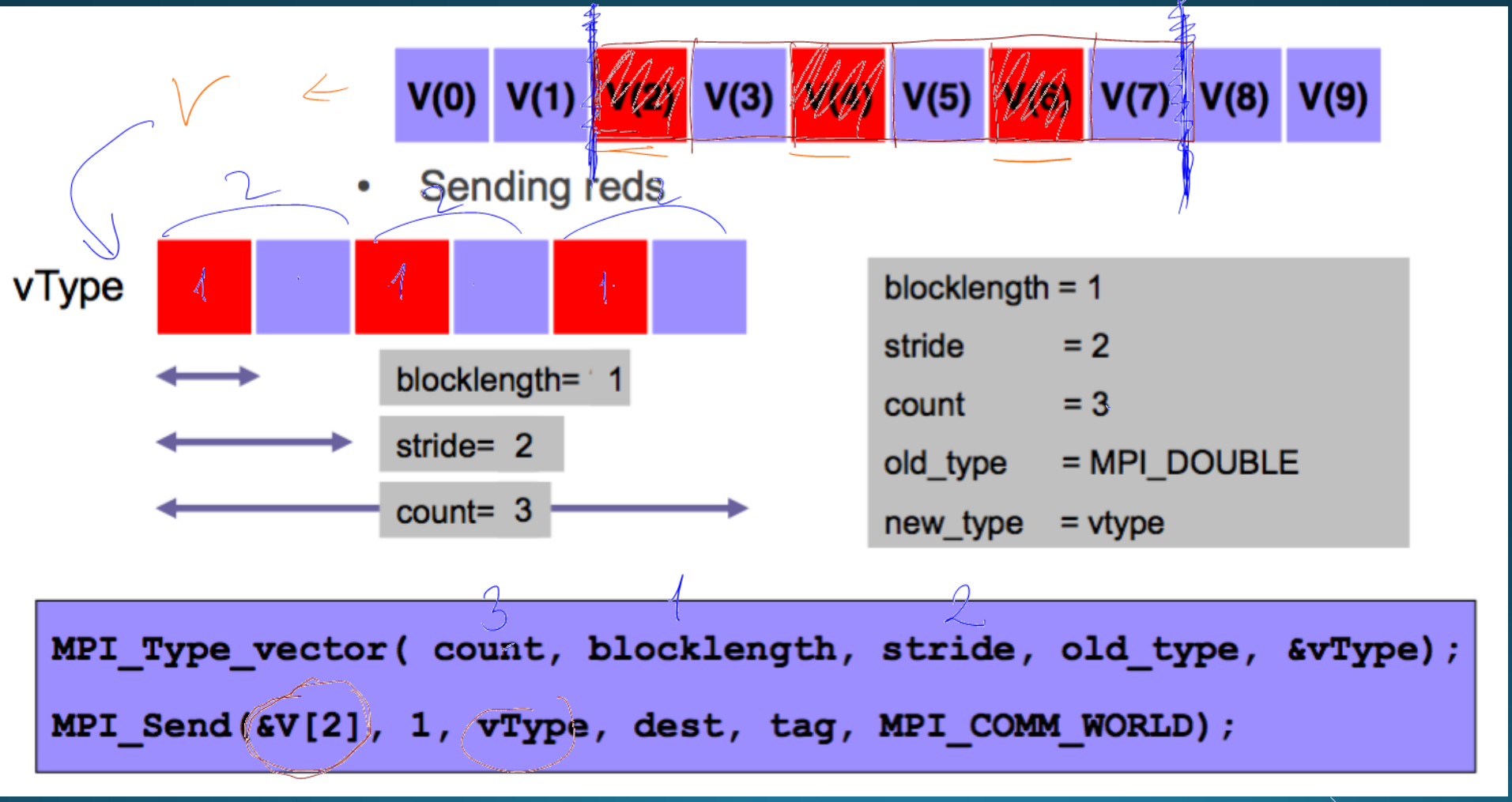
```
MPI_Type_vector(int count, int blocklength,
     int stride, MPI_Datatype old_type,
     MPI_Datatype *new_type);
```

MPI_Type_vector (devam)



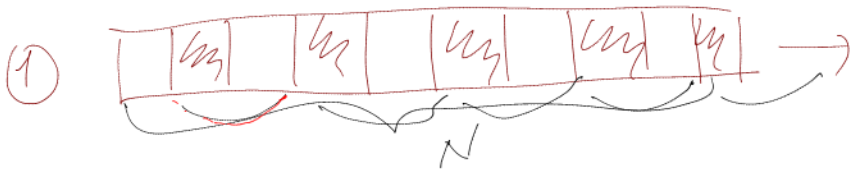

```
MPI_Type_vector(count, blocklength, stride, old_type,
    &new_type);
```

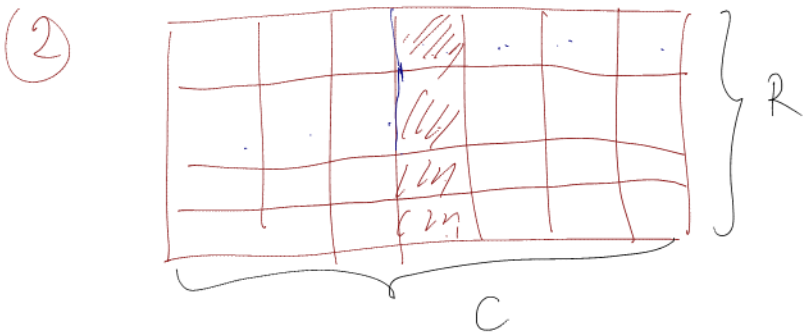
MPI_Type_vector (devam)

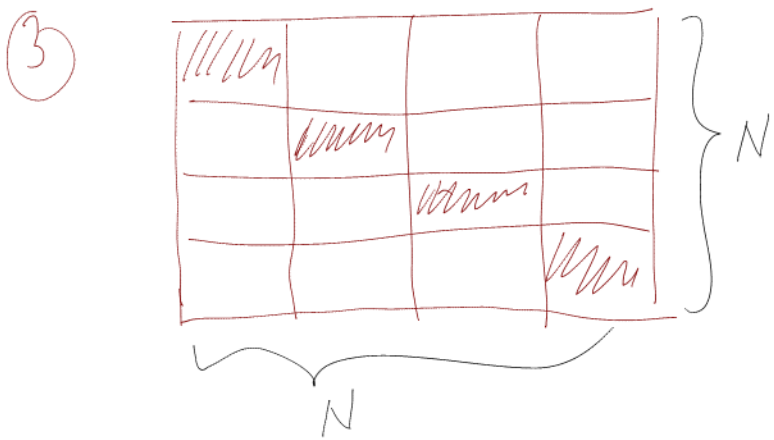



```
MPI_Type_vector( count, blocklength, stride, old_type, &vType);
MPI_Send(&V[2], 1, vType, dest, tag, MPI_COMM_WORLD);
```

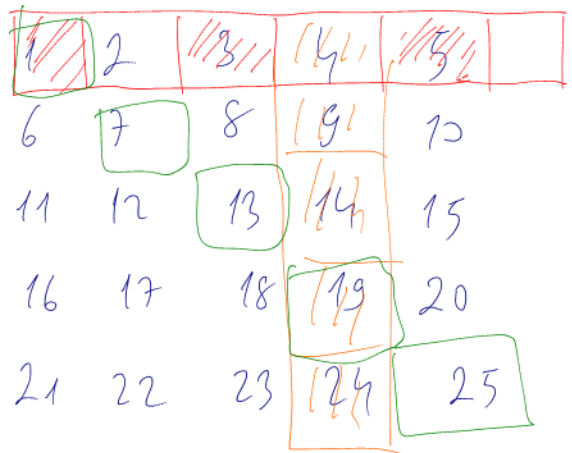
MPI_Recv(u, 3, MPI_DOUBLE, source, tag, comm, &stat);



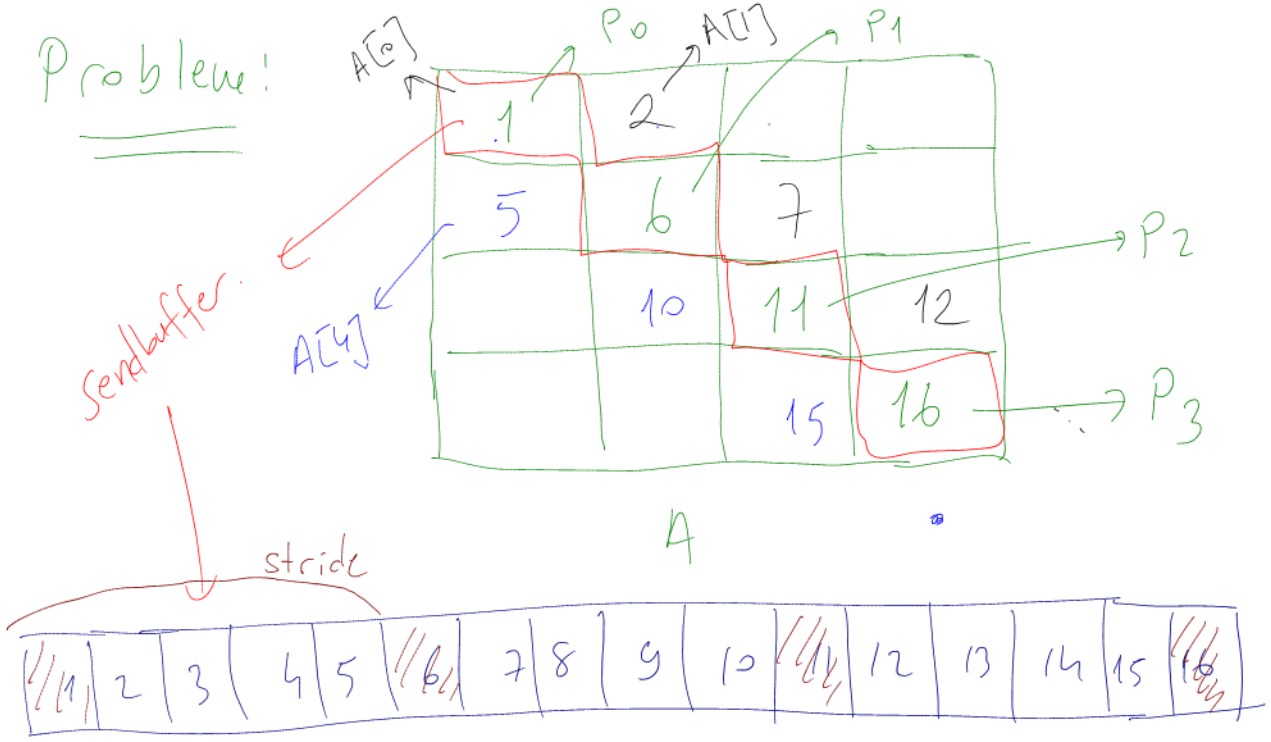




	Count	blocklength	stride
	floor(N/2)	1	2
	R	1	C
	N	1	N+1

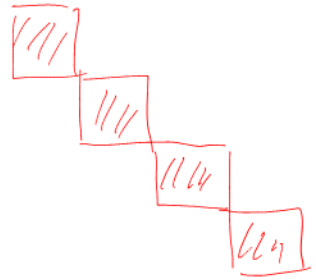


Problem:



old
- 4 adet MPI_Send ve MPI_Recv

New
- MPI_Type_vector

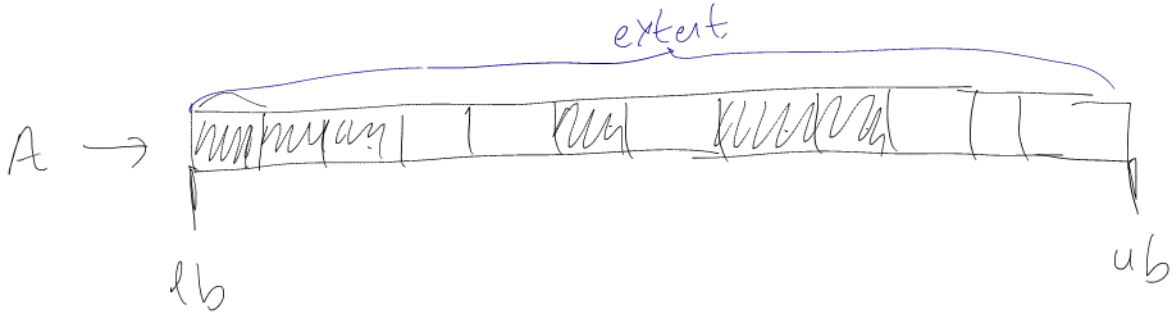


MPI_Type_vector (count, blen, stride, MPI_FLOAT, &drag);
(N, 1, N+1)

MPI_Send ✓

MPI_Scatter ?

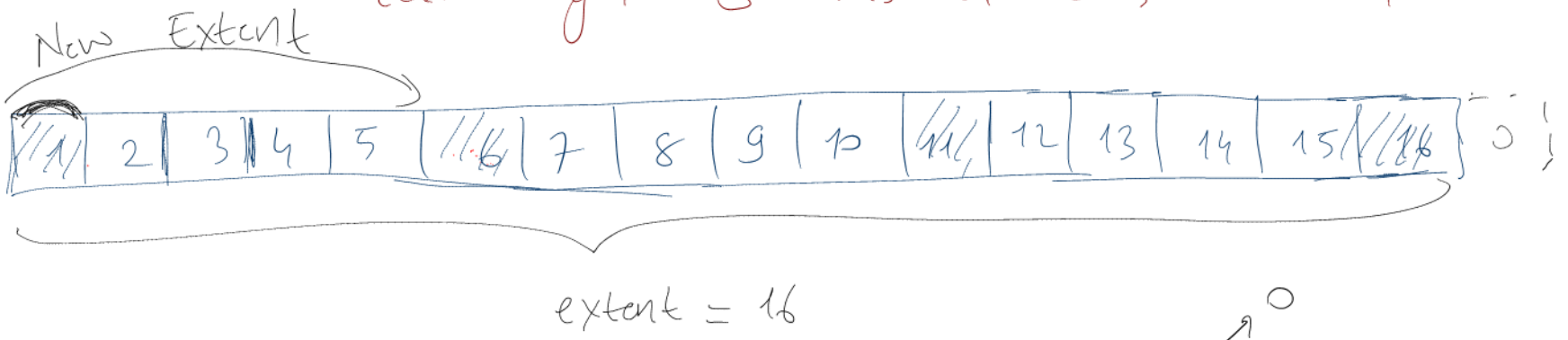
extent: (Kaplam)



size(A) = 6

extent(A) = ub − lb = 12

MPI_Scatter; Her chunk kadar veryi ilgili proseste gönderdikten sonra tekrar gönderecek kısma erişmek için extent kadar yol gider.



1
0 -
0 -
0 -

MPI_Type_create_resized(oldType, lb, newExtent, *newType);

lb → 0

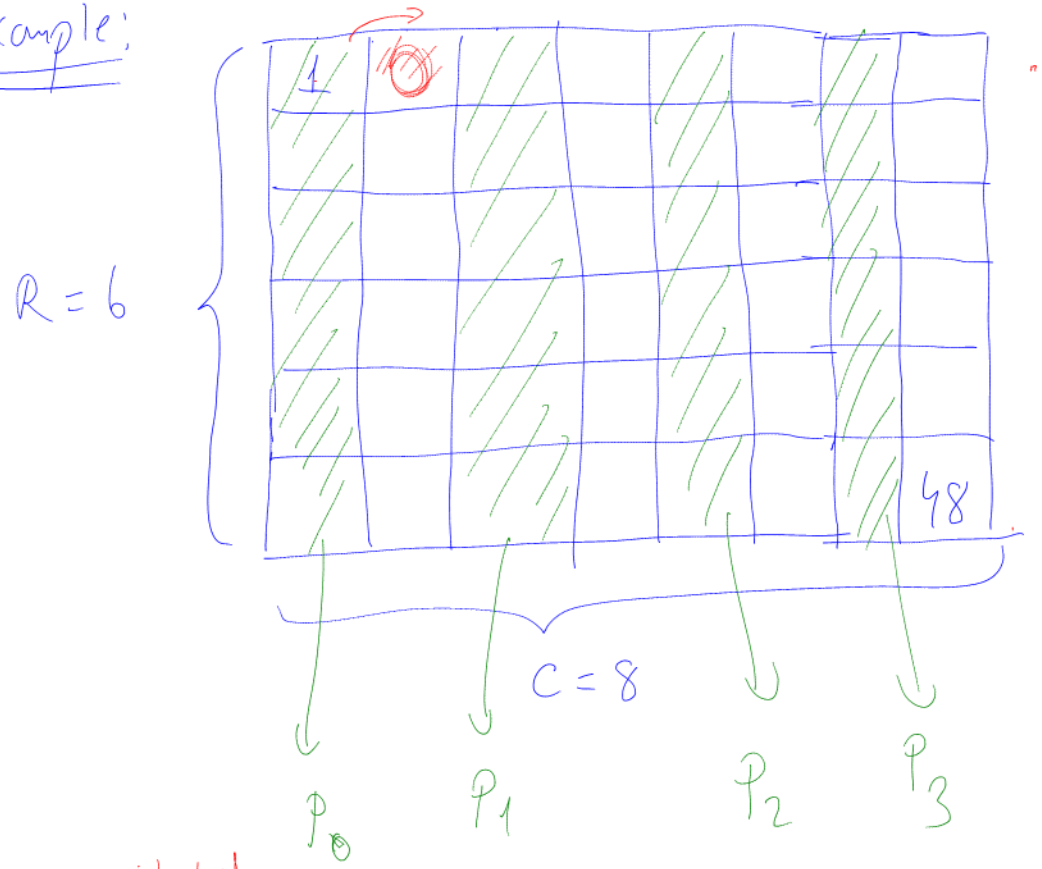
newType → newDiagType.

oldType → diagType

newExtent → (N+1) * sizeof(float)

commit

Example:



① # of procs : 6

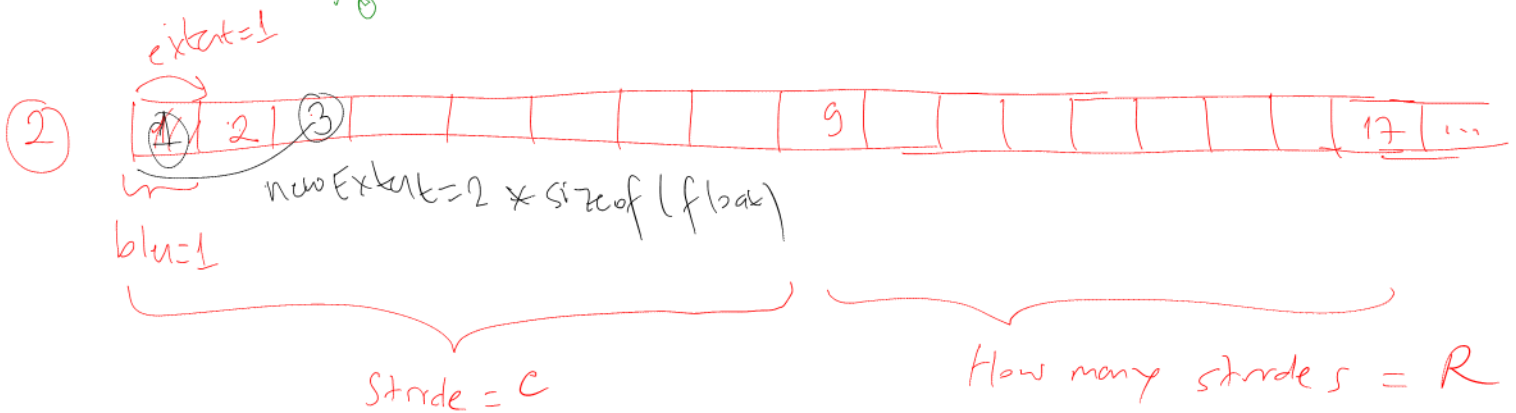
Send each row to a process consecutively.

MPI_Type_contiguous

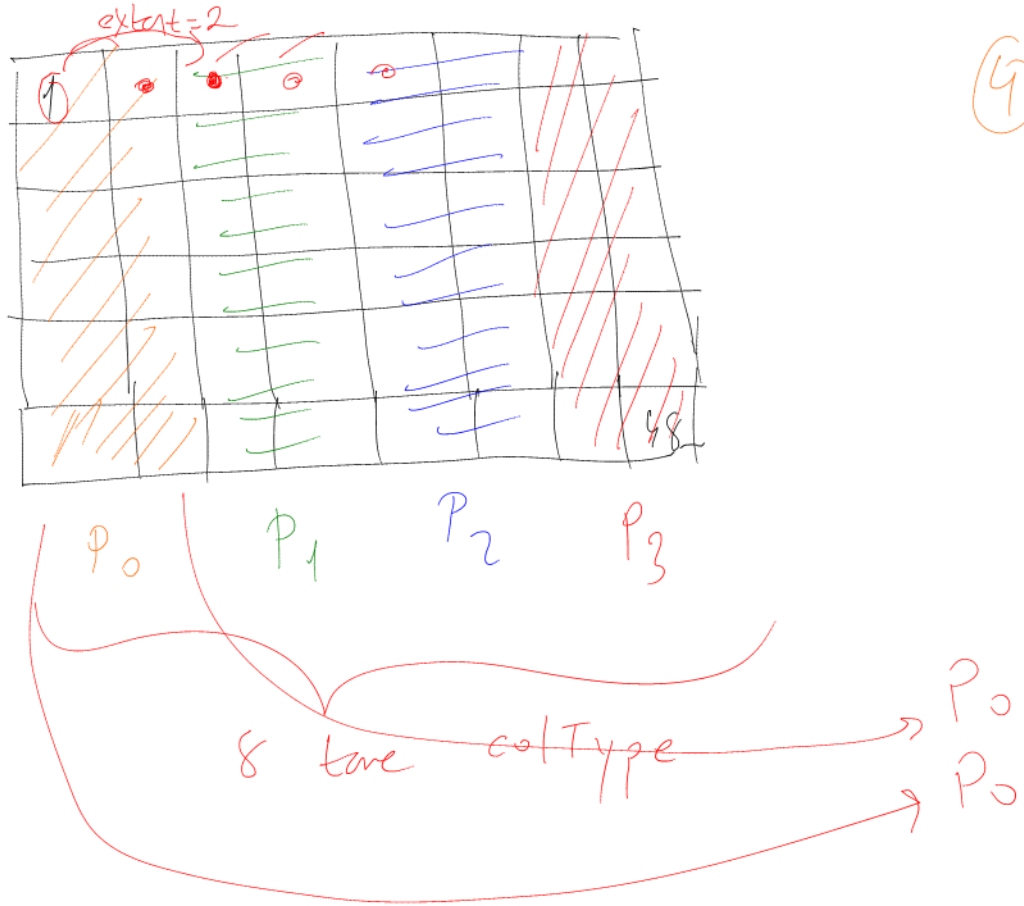
② # of procs : 8

Send each column to a process consecu.

③ # of procs : 4 → MPI_Type_vector.



MPI_Type_create_resized(colType, 0, 1*sizeof(float), newType)



(4) # of procs = 4

MPI_Type_indexed

- Eğer bir arada bulunması istenen veriler arasındaki boşluklar düzenli değilse bu MPI fonksiyonu kullanılır.

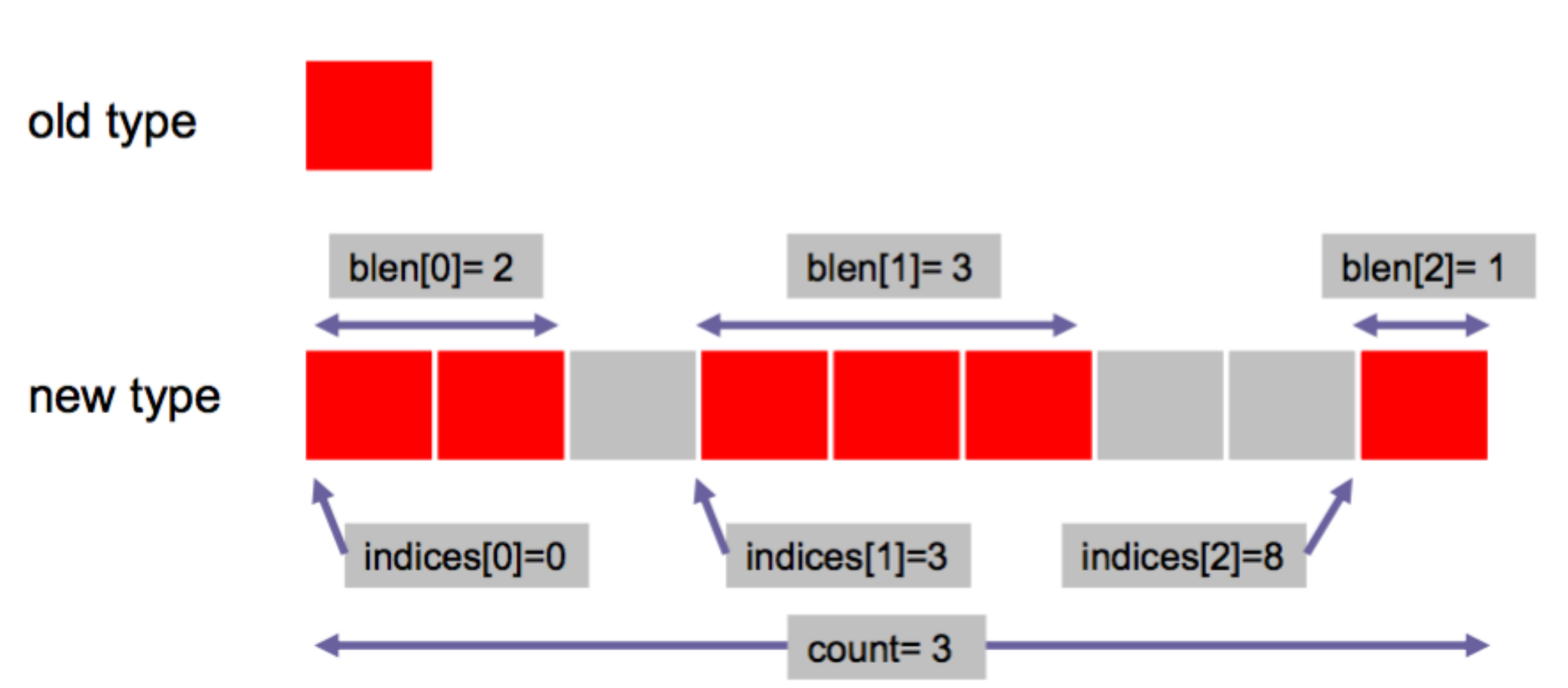
```
MPI_Type_indexed(int count,int blocklength[],
      int indices[], MPI_Datatype old_type,
      MPI_Datatype *newtype);
```

`count:` number of blocks

`blocklength:` number of elements in each block

`indices:` displacement for each block, measured as number of elements

MPI_Type_indexed (devam)



MPI_Type_indexed (devam)

double A[4][4];

A (0,0)	A (0,1)	A (0,2)	A (0,3)
A (1,0)	A (1,1)	A (1,2)	A (1,3)
A (2,0)	A (2,1)	A (2,2)	A (2,3)
A (3,0)	A (3,1)	A (3,2)	A (3,3)

old type = MPI_DOUBLE

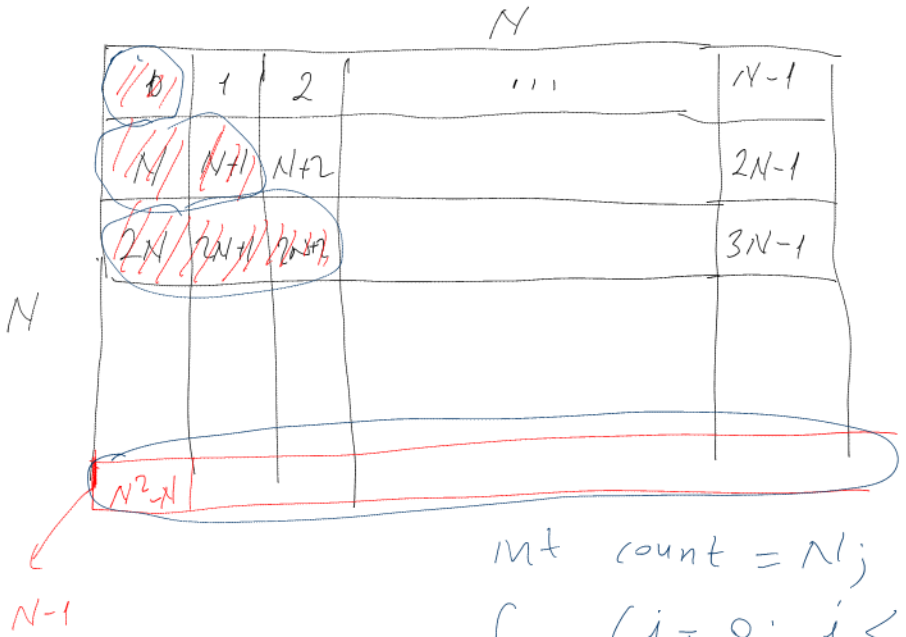
new type = upper

count = 4

blocklen[ ] = (4, 3, 2, 1)

indices[ ] = (0, 5, 10, 15)

```
MPI_Type_indexed(count, blocklen, indices, MPI_DOUBLE, upper);
```



lowerType

blen = {1, 2, 3, ... , N}

count = N

indices = {0, N, 2N, 3N, ... , (N-1)xN}

```
int count = N;
for (i = 0; i < N; i++) {
    blen[i] = i+1;
    indices[i] = i * N;
}
```

MPI_Type_indexed(count, blen, indices, MPI_FLOAT, &lowerType);

→ Contiguous, vector, indexed → Send, recv

create_resized → Scatter

Bir veri tipini kopyalamak (duplication)

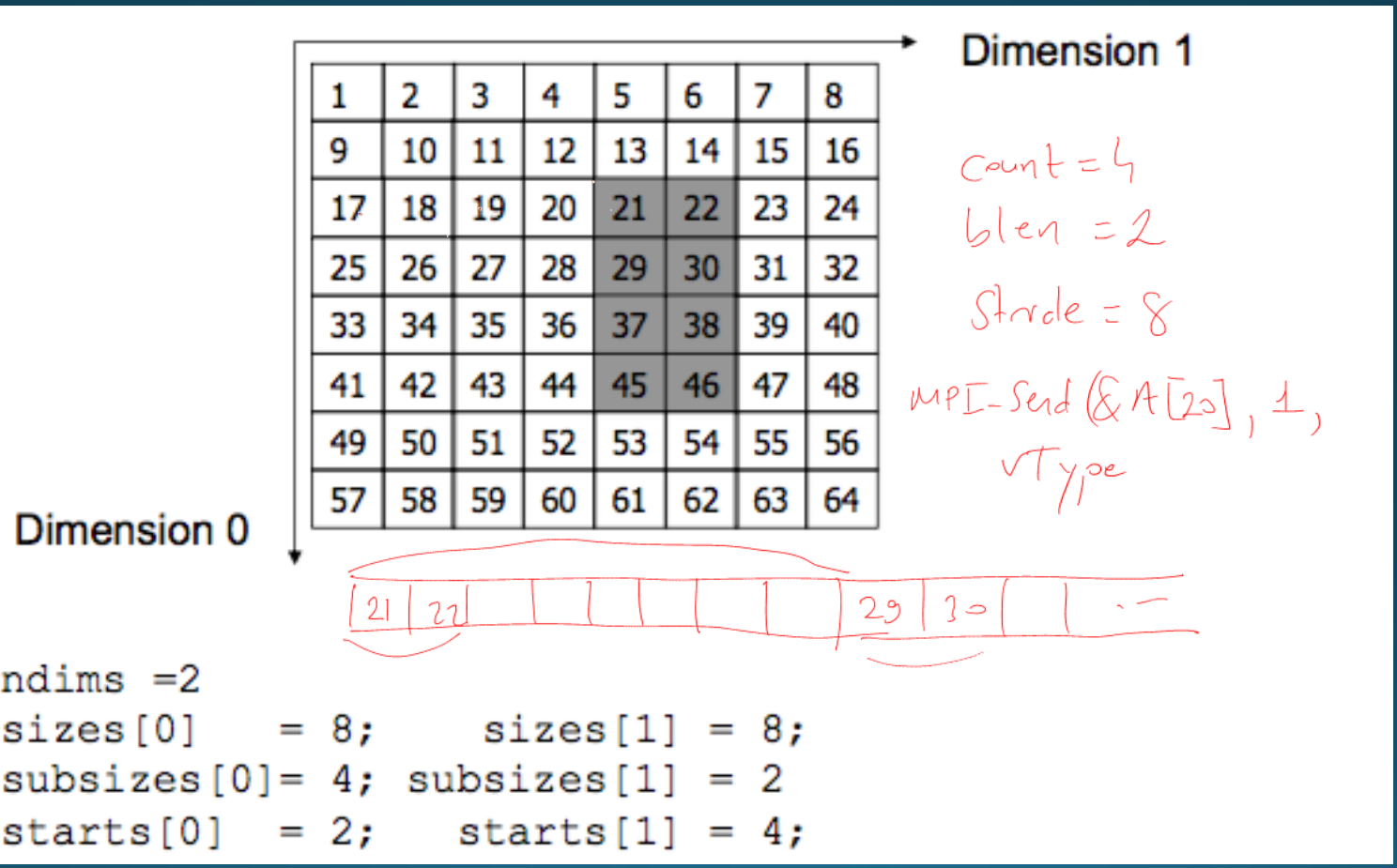
- `MPI_Type_dup(MPI_Datatype datatype, MPI_Datatype *newtype);`

- Kütüphane geliştiricileri için oldukça kullanışlıdır.
- Yeni veri tipinin commitlenme durumu bir önceki veri tipinin commitlenme durumu ile aynıdır.
  - Yani eğer `datatype` commitlenmiş ise, `newtype` da otomatik olarak commitlenmiş olur.

MPI_Type_create_subarray

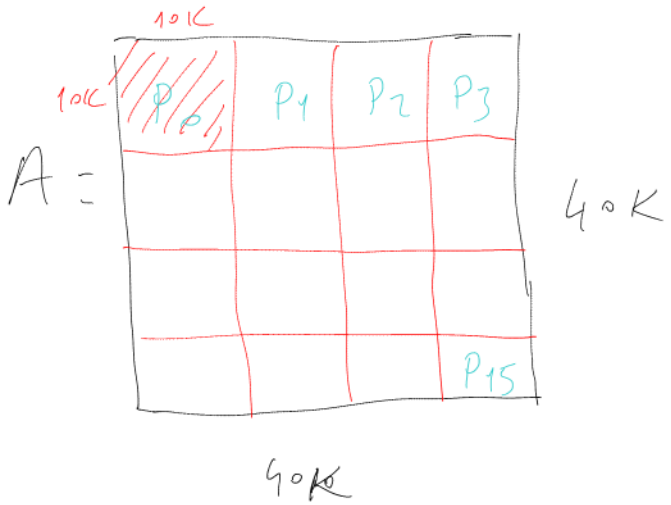
- `MPI_Type_create_subarray (int ndims, int sizes[], int subsizes[], int starts[], int order, MPI_Datatype datatype,MPI_Datatype *newtype);`
- *n* boyutlu verinin alt *n* boyutlu verilerini oluşturur.
- `sizes[]`: tüm verinin boyut bilgisini içeren dizi
- `subsizes[]`: alt dizinin boyut bilgisini içeren dizi
- `starts[]`: altdizinin başlangıç elemanının asıl dizideki indislerini içeren dizi
- `order`:
  - `MPI_ORDER_C`: Satır yönlü sıralama için
  - `MPI_ORDER_FORTRAN`: sütun yönlü sıralama için

MPI_Type_create_subarray




```
MPI_Type_create_subarray(ndims, sizes, subsizes, starts,
    MPI_ORDER_C, &newtype);
```

MPI_FLOAT



# of procs = 16

→ MPI - Type - create - subarray

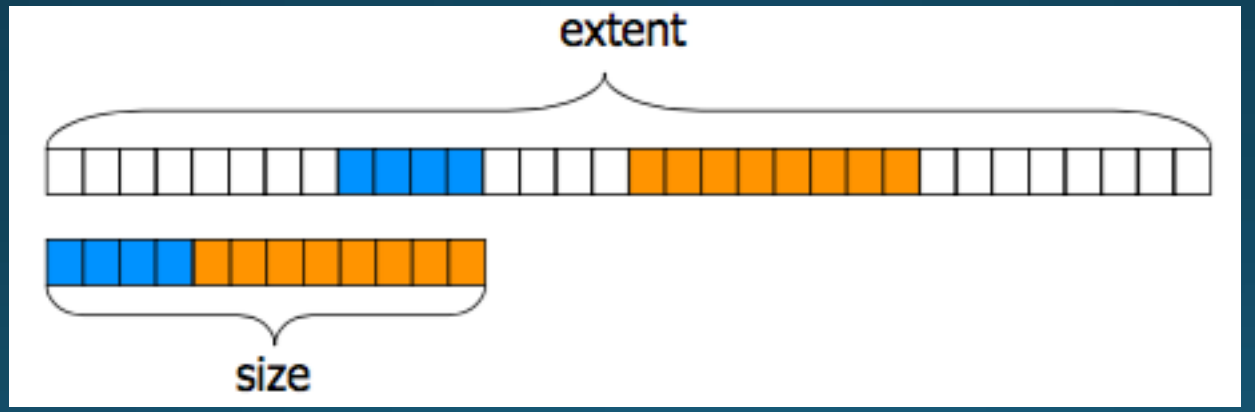
→ MPI - Type - create - resized

→ MPI - Scatter

(1) MPI - Type - create - subArray

Kaplam (extent) ve Boyut (size)

- `MPI_Type_extent(MPI_Datatype dat, MPI_Aint *ext);`
- `MPI_Type_size(MPI_Datatype dat, int *size);`



extent = alt sınır – üst sınır
size = Gerçekten transfer edilen byte miktarı

SORULAR?