# MPI functions

MPI is a complex system comprising of numerous functions with various parameters and variants

Six of them are indispensable, but can write a large number of useful programs already





Other functions add **flexibility** (**datatype**), robustness (non-blocking send/receive), efficiency (communication mode), modularity (communicators, groups) or convenience (collective operations, topology).



In the lectures, we are going to cover most commonly encountered functions

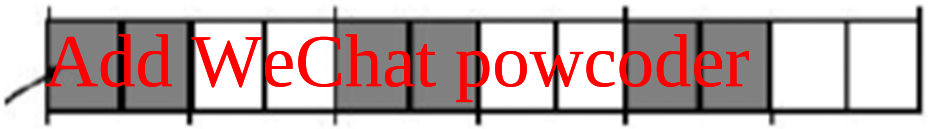
# Derived datatype

- In MPI_Send, the data to be send, expressed as a triple(addr, count, datatype), have to occupy contiguous memory address
- Now, there are two problems:

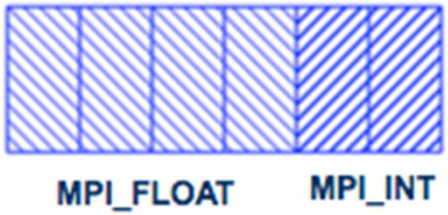


1. What if the data do not occupy contiguous address






2. What if the data to be sent do not have the same data type.



# Derived datatype

- Two possible solutions to problem 1:

1. Call a MPI_Send for each data block occupying contiguous space.

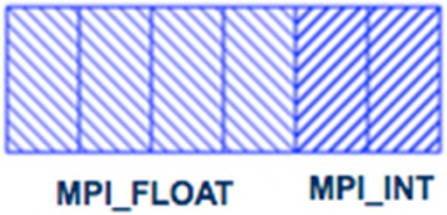


2. Copy the non-contiguous data to a contiguous buffer space and then call a MPI_Send to send the data in the buffer





- Solution to problem 2:




Call a MPI_Send for each data block which has the same datatype and occupies the contiguous space.

These solutions are tedious and don't have good performance

# Derived datatype

- Derived datatype allows the users to construct (derive) their own data types
- Derived datatypes can be used in
  - Grouping non-contiguous data.
  - Grouping data of different datatypes
- Provides the opportunity to send this kind of data conveniently and potentially improving performance

# Memory Layout of a Datatype

- **The memory layout (mapping) of a datatype in MPI is expressed as**

**{(type_0, offset_0), (type_1, offset_1), ..., (type_n, offset_n)}**

## Three Attributes to characterize non-contiguous data

| | Same type in each block | Same number of elements in each block | Same distance between consecutive block |
|---|---|---|---|
| MPI_Type_vector | Yes | Yes | Yes |
| MPI_Type_create_indexed_block | Yes | Yes | No |
| MPI_Type_indexed | Yes | No | No |
| MPI_Type_struct | No | No | No |



**A block is a contiguous data items**



| | Same type in each block | Same number of elements in each block | Same distance between consecutive block |
|---|---|---|---|
| MPI_Type_vector | Yes | Yes | Yes |
| MPI_Type_create_indexed_block | Yes | Yes | No |
| MPI_Type_indexed | Yes | No | No |
| MPI_Type_struct | No | No | No |

# MPI_Type_vector

**MPI_Type_Vector(count, blocklen, stride, oldtype, newtype)**

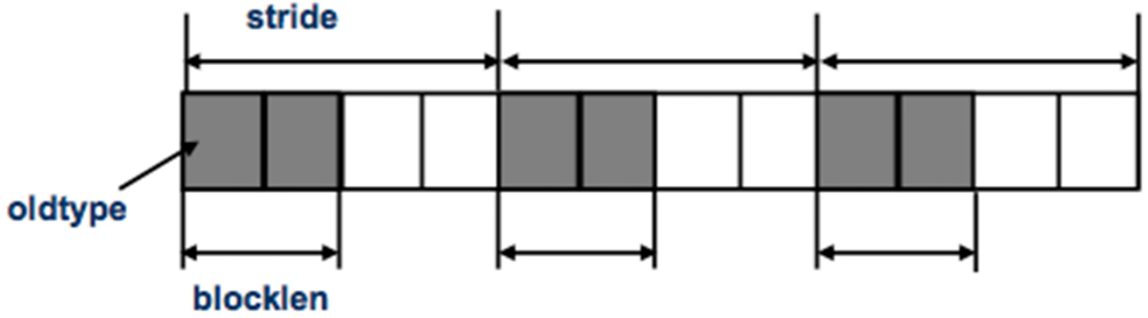
**Defines a derived type newtype comprising count consecutive blocks of data elements with datatype oldtype, with each block containing blocklen data elements, and the start of successive blocks separated by stride data elements. E.g.**



```
float data [1024];

MPI_Datatype floattype;

MPI_TYPE_vector (3, 2, 4, MPI_FLOAT, &floattype);
MPI_Type_commit (&floattype);

MPI_Send (data, 1, floattype, dest, tag, MPI_COMM_WORLD);
MPI_Type_free (&floattype)
```



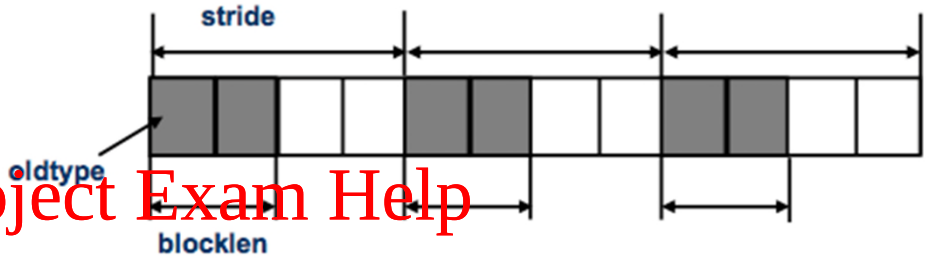




# MPI_Type_vector

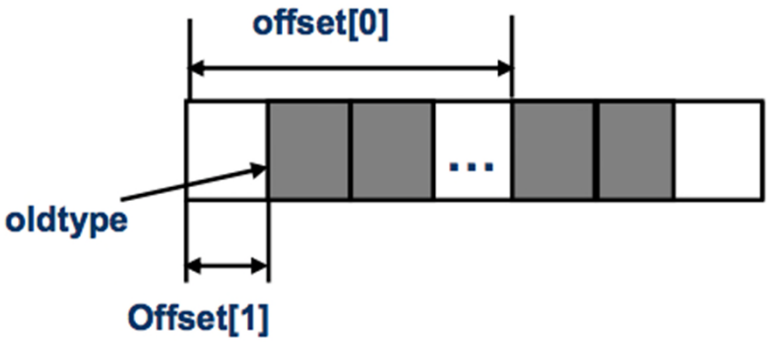
**The code in last slide is equivalent to the following code**

```
MPI_FLOAT *buff, *data;
j=0;
for (i=0; i<3; i++) {
        buff[j] = data [i*4];
        buff[j+1]=data[i*4+1];
        j+=2;}
MPI_Send (buff, 6, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
```






- **The memory layout in the example (assume MPI_FLOAT is 16 bits)**
- Old datatype = {(MPI_FLOAT, 0)}
- New datatype = {
  (MPI_FLOAT, 0), (MPI_FLOAT, 16),
  (MPI_FLOAT, 64), (MPI_FLOAT, 80),
  (MPI_FLOAT, 128), (MPI_FLOAT, 144) }






| | Same type in each block | Same number of elements in each block | Same distance between consecutive block |
|---|---|---|---|
| MPI_Type_vector | Yes | Yes | Yes |
| MPI_Type_create_indexed_block | Yes | Yes | No |
| MPI_Type_indexed | Yes | No | No |
| MPI_Type_struct | No | No | No |

# MPI_Type_create_indexed_block

***int MPI_Type_create_indexed_block( int count, int blocklength, int array_of_offsets[], MPI_Datatype oldtype, MPI_Datatype *newtype );***

***Parameters:***

***count***

- [in] length of array of displacements (integer)



***blocklength***

- [in] size of block (integer)





***array_of_offsets***

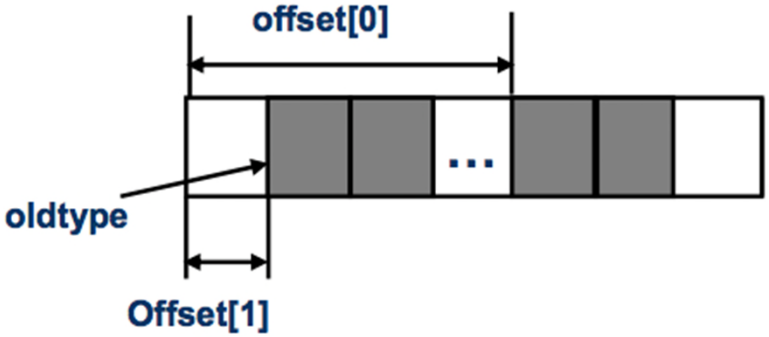
- [in] array of displacements (array of integer)

***oldtype***

- [in] old datatype (handle)

***newtype***

- [out] new datatype (handle)



# MPI_Type_create_indexed_block

*int MPI_Type_create_indexed_block( int count, int blocklength, int array_of_displacements[], MPI_Datatype oldtype, MPI_Datatype *newtype );*



*Let oldtype = {(double, 0), (char, 32)} with extent 40 bits.*

*Let D = (4, 1).*



*After calling MPI_Type_indexed(2, 2, D, oldtype, newtype)*

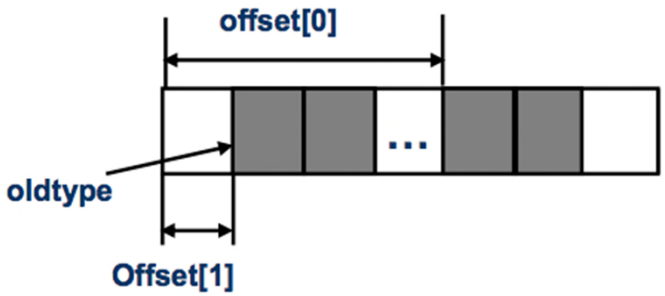


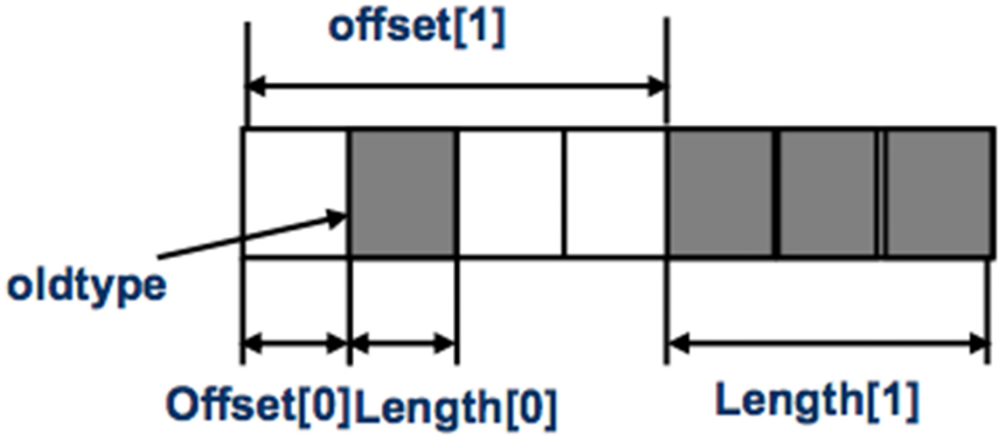
*newtype= {*

*(double, 160), (char, 192), (double, 200), (char, 232)*

*(double, 40), (char, 72), (double, 80), (char, 112)*

*}*

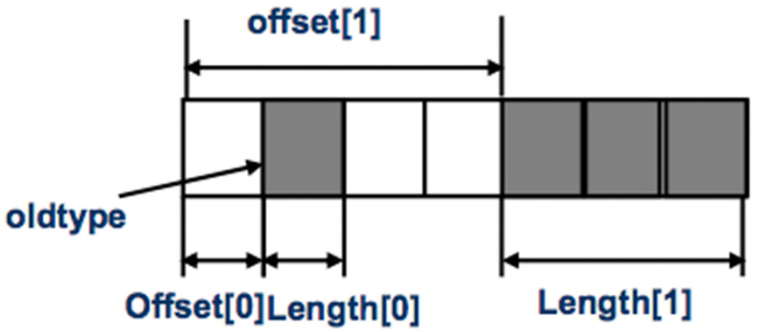







| | Same type in each block | Same number of elements in each block | Same distance between consecutive block |
|---|---|---|---|
| MPI_Type_vector | Yes | Yes | Yes |
| MPI_Type_create_indexed_block | Yes | Yes | No |
| MPI_Type_indexed | Yes | No | No |
| MPI_Type_struct | No | No | No |

# MPI_Type_Indexed




**MPI_Type_Indexed (count, lengths[], offsets[], oldtype, newtype)**

→**Used in the case where the elements in the datatype to be constructed have the same type, but each block has different number of elements and the distance between two consecutive blocks are different**



→**Used to define a type comprising one or more blocks of a primitive or previously defined datatype, where block lengths and the displacement between blocks are specified in arrays**



→**The above call defines a type *newtype* comprising count consecutive blocks of data elements with type *oldtype*, with block *i* having a displacement of offsets data elements and containing *lengths* data elements**

# MPI_Type_Indexed

***MPI_Type_Indexed(count, lengths[], offsets[], oldtype, newtype)***

***Let oldtype = {(double, 0), (char, 32)} with extent 40 bits.***

***Let B = (1, 3) and Let D = (1, 4).***

***After calling MPI_Type_indexed(2, B, D, oldtype, newtype)***

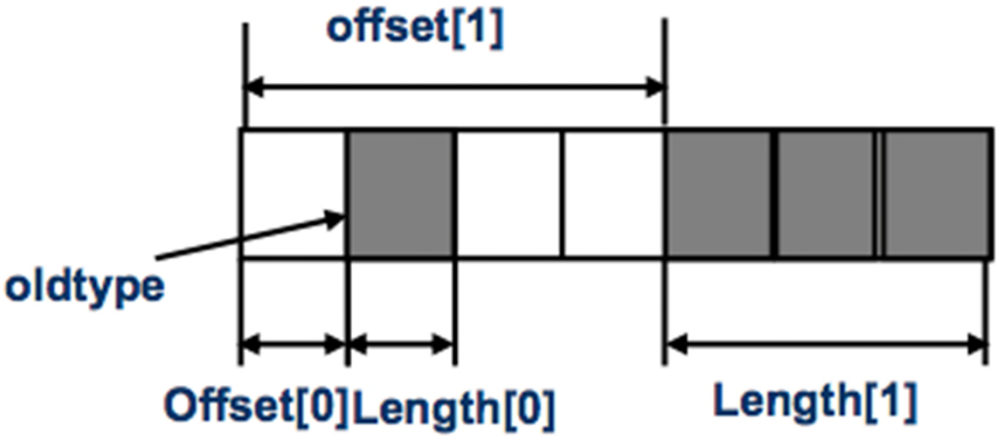


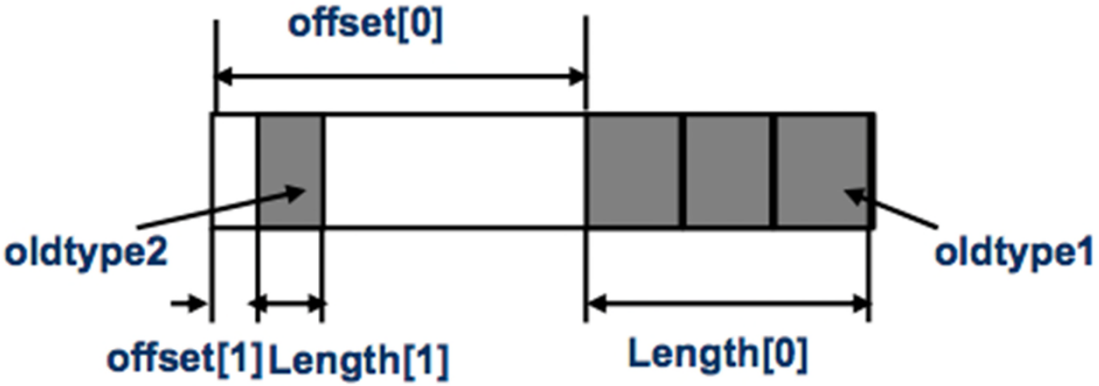
***newtype= {***

***(double, 40), (char, 72)***

***(double, 160), (char, 192), (double, 200), (char, 232), , (double, 240), (char, 272)***

***}***








| | Same type in each block | Same number of elements in each block | Same distance between consecutive block |
|---|---|---|---|
| MPI_Type_vector | Yes | Yes | Yes |
| MPI_Type_create_indexed_block | Yes | Yes | No |
| MPI_Type_indexed | Yes | No | No |
| MPI_Type_struct | No | No | No |

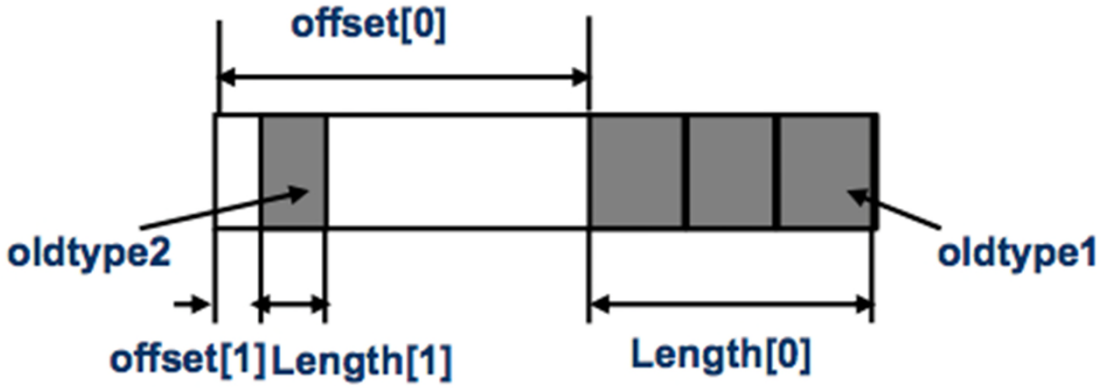
# MPI_Type_struct

**int MPI_Type_struct(int count, int *array_of_blocklengths, MPI_int *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)**



**The derived datatype includes different datatypes, each block with different displacements and different data elements**





# MPI_Type_struct

*Let oldtype1 = {(double, 0), (char, 32bits)} with extent 5 bytes, and oldtype2 = {(float, 0), (char, 16bits)} with extend 3 bytes.*

*Let B = (3, 1) and D = (20, 2) and C=(oldtype1, oldtype2).*

*After calling MPI_Type_struct (2, B, D, C, newtype)*



*newtype= {*

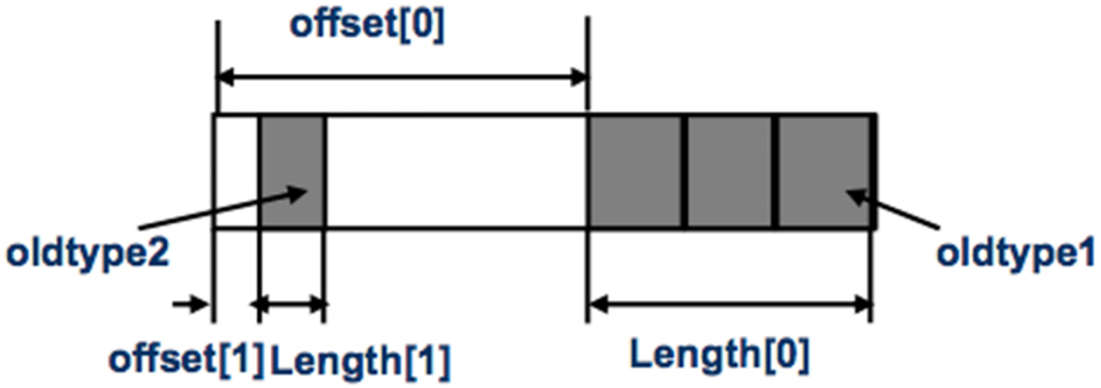


*(double, 20), (char, 24), (double, 25), (char, 29) (double, 30), (char, 34),*

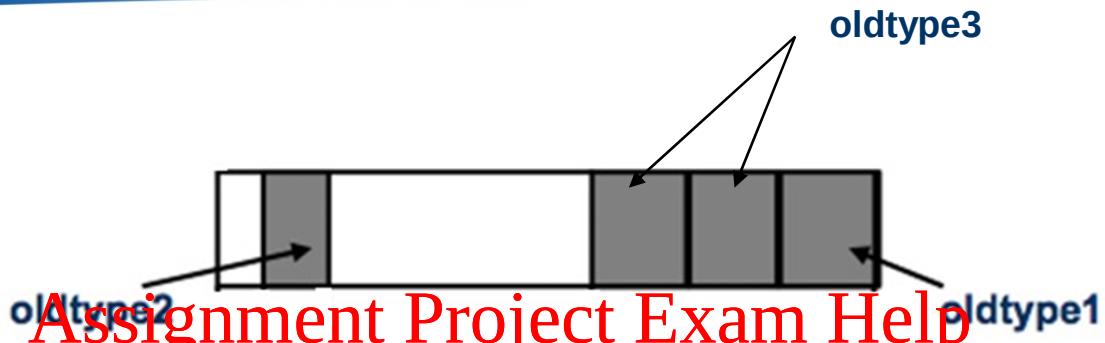


*(float, 2), (char, 4)*

*}*



# Derived datatype summary



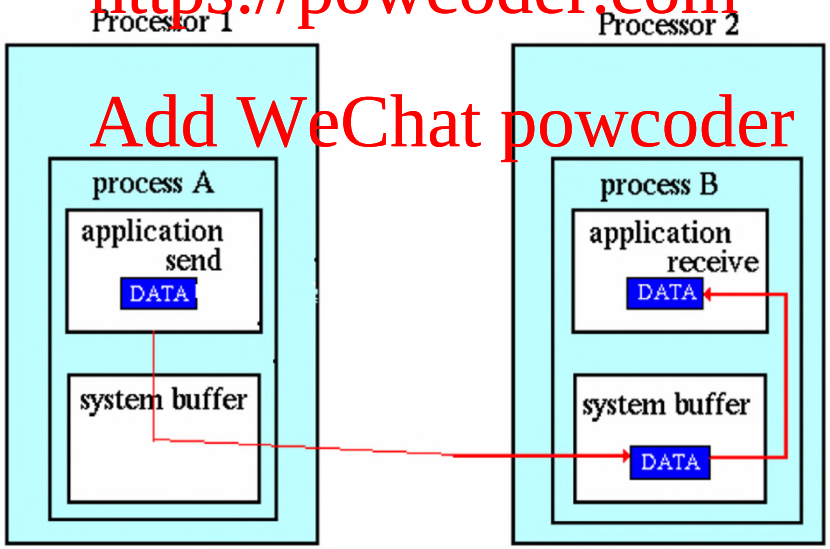



| | Same type in each block | Same number of elements in each block | Same distance between consecutive block |
|---|---|---|---|
| MPI_Type_vector | Yes | Yes | Yes |
| MPI_Type_create_indexed_block | Yes | Yes | No |
| MPI_Type_indexed | Yes | No | No |
| MPI_Type_struct | No | No | No |

# How are the data with derived data type are sent?

- Pack the data elements specified by the memory layout into the contiguous space in system buffer
- MPI library sends the data in system buffer






Path of a message buffered at the receiving process

# Why using derived data type can improve performance?

- Two possible solutions to non-contiguous data:



1. Call a MPI_Send for each data block that occupies contiguous space



Calling MPI_Send multiple times, which occurs higher overhead (e.g., handshaking between sender and receiver)



2. Copy the non-contiguous data to a contiguous buffer space and then call a MPI_Send to send the data in the buffer

Needs to copy the data twice: 1) copy the non-contiguous data to a contiguous buffer space; 2) copy the data in the buffer to system buffer

# MPI_Type_contiguous

```
int MPI_Type_contiguous( int count, MPI_Datatype old_type,
MPI_Datatype *new_type_p );
```

```
MPI_Type_contiguous(3, MPI_REAL, newtype)
// returns a new datatype that represents the concatenation of 3
instances of // MPI_REAL.
MPI_Type_commit(newtype)
// commits the datatype, must be done before communication
MPI_Send(buff, 1, newtype, dest, tag, MPI_COMM_WORLD)
// sends the data at location data to dest
MPI_Type_free(newtype)
// frees the datatype
```

This is equivalent to the following single call

```
MPI_SEND(buff, 3, MPI_REAL, dest, tag, MPI_COMM_WORLD)
```

Where the elements to be sent are already contiguous

# MPI_Type_contiguous

- **MPI_Type_contiguous(3, MPI_REAL, newtype)**



- Old data type = {(MPI_REAL, 0)}
- **Memory layout of the newtype (assume MPI_REAL occupies 32 bits)**
  - New datatype = {(MPI_REAL, 0), (MPI_REAL,32), (MPI_REAL, 64)}

# Summary of MPI

**Point-to-point communication**

**Collective communication**



**Communication modes**



**Virtual topology**



**Derived datatype**