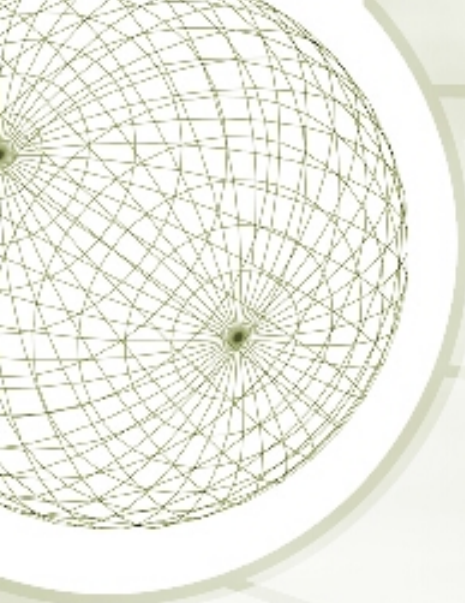


MPI API

Hugues Leroy, Dani Mezher

MPI

- Header file

  #include <mpi.h>

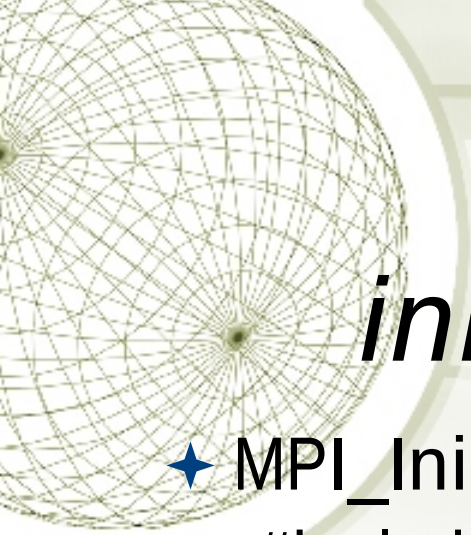
- Function declaration

  Error=MPI_Xxxxx(params)

  MPI_Xxxxx(params)

- Predefined constants

  MPI_YYYYY



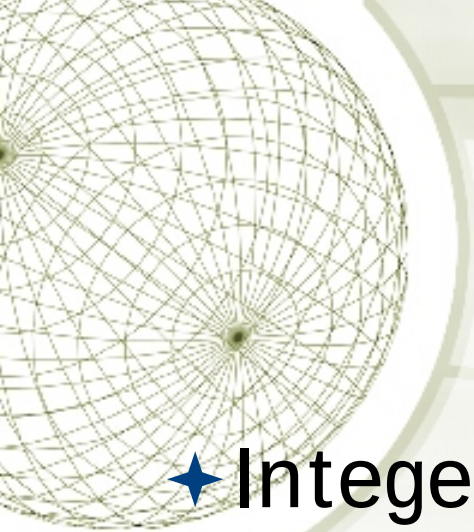
Application initialization/Termination

- MPI_Init(int *,char ***argv), MPI_Finalize()

```
#include <mpi.h>
int main(int argc,char **argv)
{  int err;
   ...
   if ((err=MPI_Init(&argc,&argv))!=MPI_SUCCESS)
   ...
   MPI_Finalize();
   ...
}
```

Communicators

- All MPI operations are relative to a communicator
- MPI_COMM_WORLD is the default communicator on application startup
- All MPI processes are members of MPI_COMM_WORLD
- Communicator data type: MPI_Comm

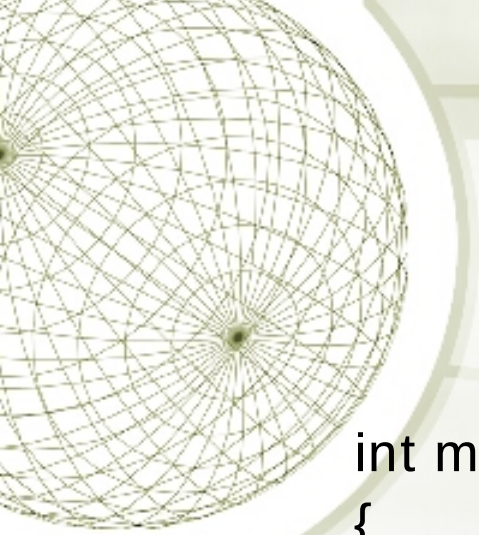


Rank and Size

- Integers identify processes
  - MPI_Comm_rank(MPI_Comm c,int *r)

    Returns the rank of the current process
  - MPI_Comm_size(MPI_Comm c,int *s)

    Returns number of processes within a communicator



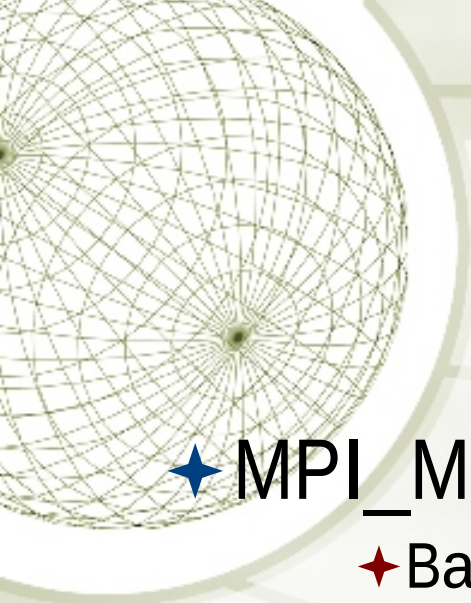
SPMD

```
int main(int argc,char **argv)
{  ….
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&r);
   if (r==0)
    {….
    }
  else
   {…
   }
  …
  MPI_Finalize();
}
```

Aborting an application

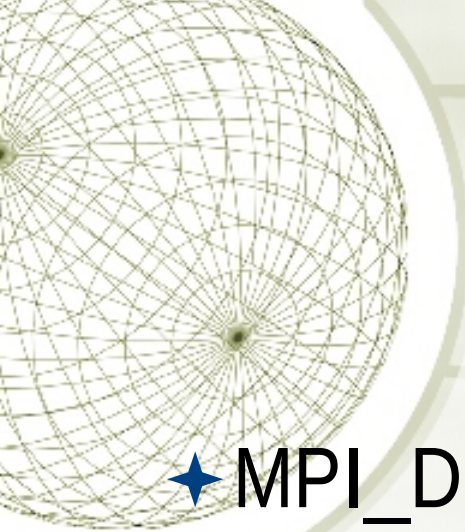
- MPI_Abort(MPI_Comm com,int e)

  Aborts all processes in communicator com with an exit error code e



MPI_Messages

- MPI_Messages hold typed data
  - Basic types(MPI_INTEGER,MPI_CHAR,...)
  - Complex types(user defined data types)
- MPI data types are hardware independent
  - Byte count
  - Byte organization
- Similar to hton?() ntoh?()

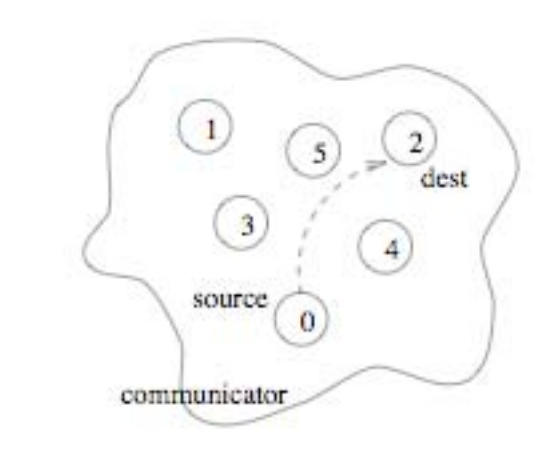


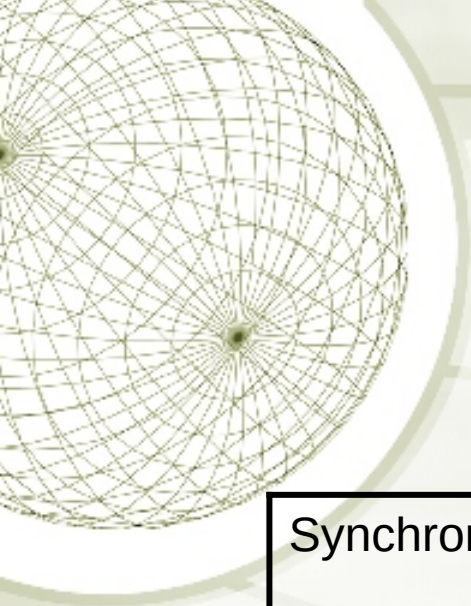
MPI data types

- MPI_Datatype constants
  - MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED

Point to Point communication

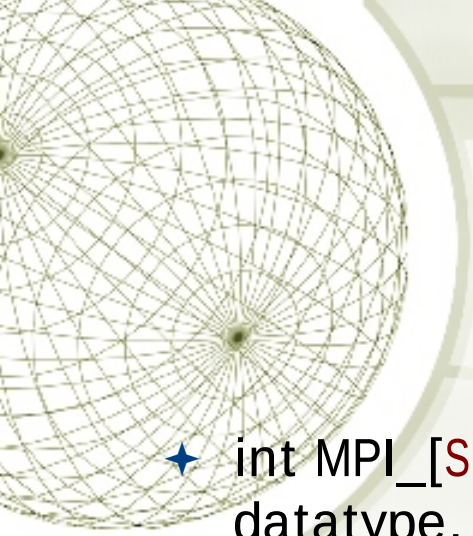
- 1 sender and 1 receiver within a single communicator
- Sender and receiver are referenced using integer ranks





Transfer modes

Synchronized	returns upon Recv request	MPI_Ssend
Buffered	returns upon buffer store	MPI_Bsend
Standard	Synchronized/Buffered Implementation dependent	MPI_Send
Ready send	Requires a receive request otherwise undetermined	MPI_Rsend
Receive	Returns upon message arrival	MPI_Recv



Functions prototypes

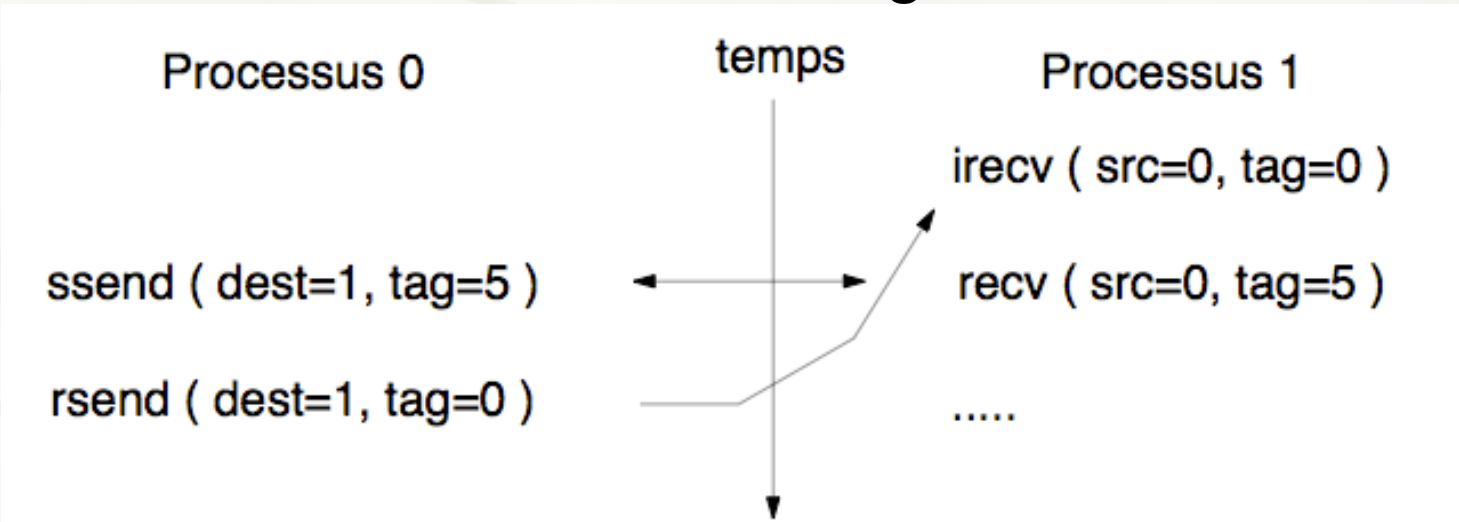
- int MPI_[S|Ss|Rs]end(void *buf,int count, MPI_Datatype datatype, int dest, int tag,MPI_Comm comm)
- int MPI_Recv(void *buf, int count, MPI_Datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
  - count= number of elements in buf.

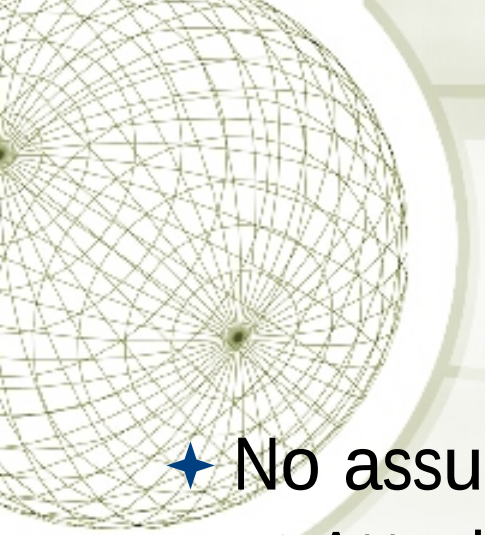
Blocking synchronized

- Sender selects synchronized mode (Ssend)
- Receiver uses blocking recv
- Sender and Receiver get synchronized

Blocking Ready Send

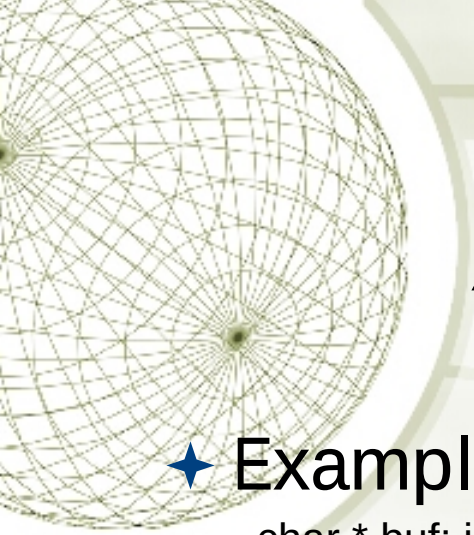
- Requires a preceding receive request, otherwise unpredictable
- Avoid data move through buffers





Buffered Blocking Send

- No assumptions regarding system buffers
  - Attach new buffer MPI_Buffer_attach(buffer,size)
  - Detach unused buffer MPI_Buffer_detach(buffer,size)
- Buffer_detach waits for all pending communications
- On attached buffer per process
- Detach is not free, attach is not malloc
- MPI_Finalize forces MPI_Buffer_detach

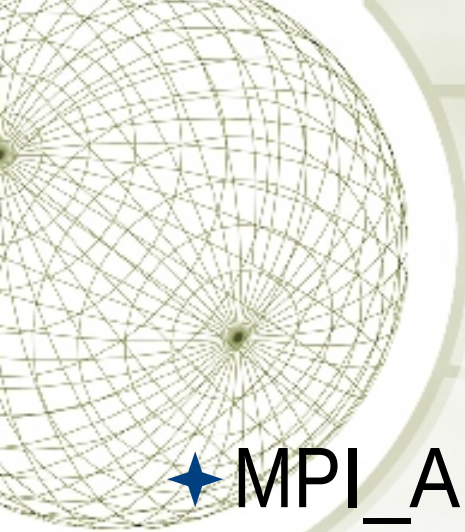


Attach, Detach example

- Example

```
char * buf; int size;
buf=(char *) malloc ( BUFSIZE );
MPI_Buffer_attach ( buf, BUFSIZE );
MPI_Bsend(); ...;
MPI_Bsend();...;
 MPI_Buffer_detach ( &buf, &size );
.....
MPI_Buffer_attach ( buf, size );
.....
MPI_Buffer_detach ( &buf, &size );
free ( buf );
```

- Buffer size=total size for all Bsend+overhead
  - Overhead=2*MPI_BSEND_OVERHEAD



Wildcards

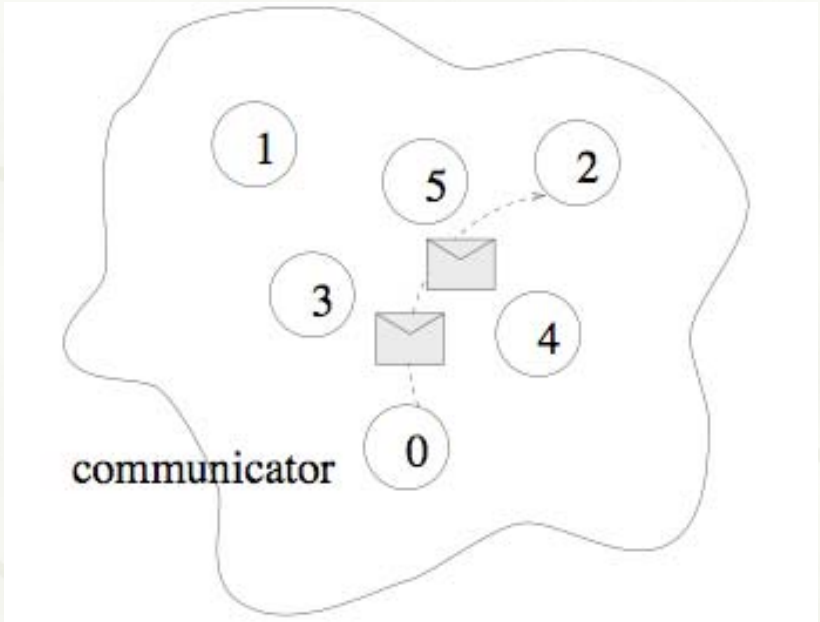
- MPI_ANY_SOURCE
  - Receive message from any sender
- MPI_ANY_TAG
  - Receive message having any tag
- Source and Tag details are included in the Recv status

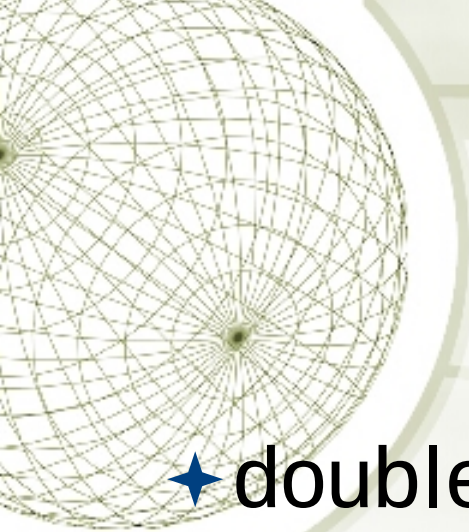
MPI_Status

- If status is an MPI_Status,
  - status.MPI_SOURCE is the sender
  - status.MPI_TAG is the sender's tag
  - MPI_Get_count(MPI_Status status, MPI_Datatype datatype,int *count) returns the number of elements in the message

Messages order

- Message order is preserved for each (sender,receiver) pair

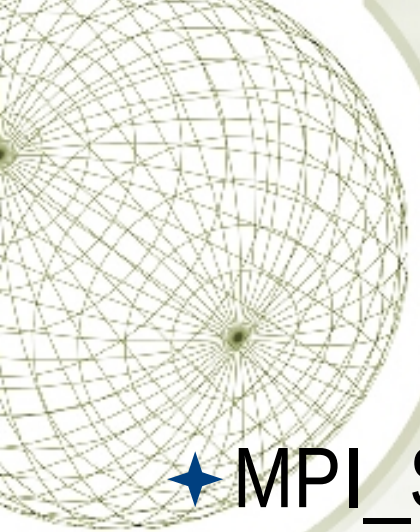




Time measurement

- double MPI_Wtime(void)
  - Return time expressed in seconds
  - ...

```
starttime=MPI_Wtime();
...
worktime=MPI_Wtime()-starttime;
```

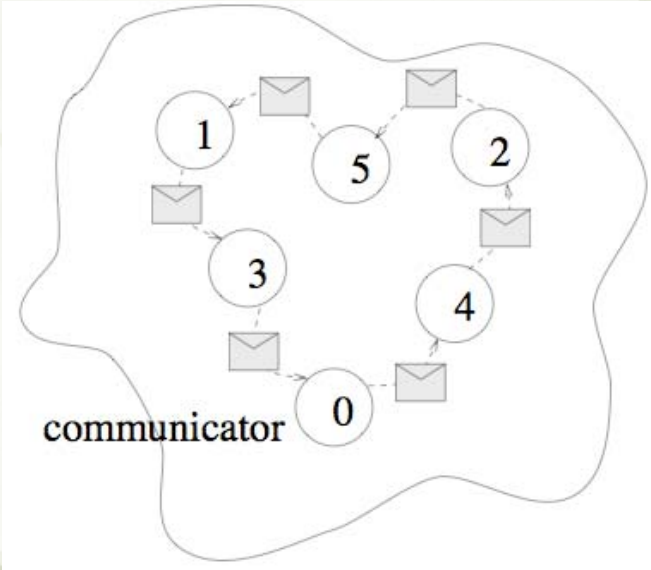


Advanced point to point communications

- MPI_Sendrecv
- MPI_Sendrecv_replace

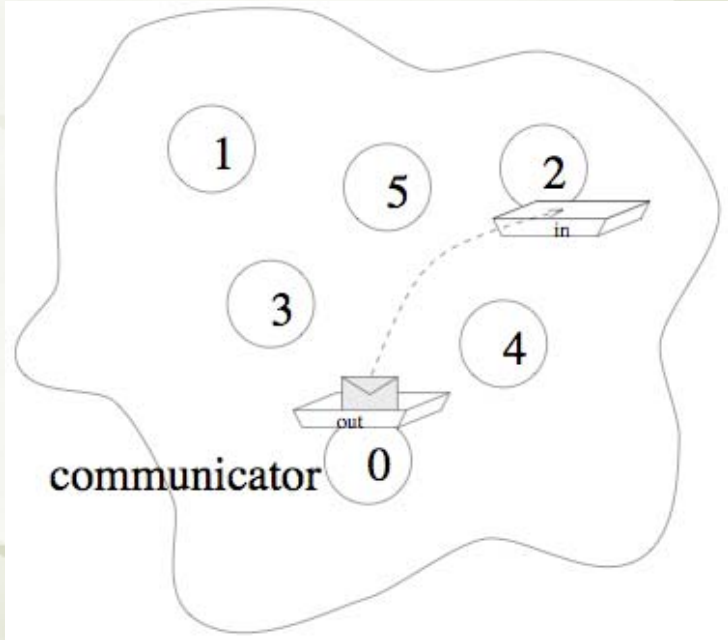
Non blocking point to point communication

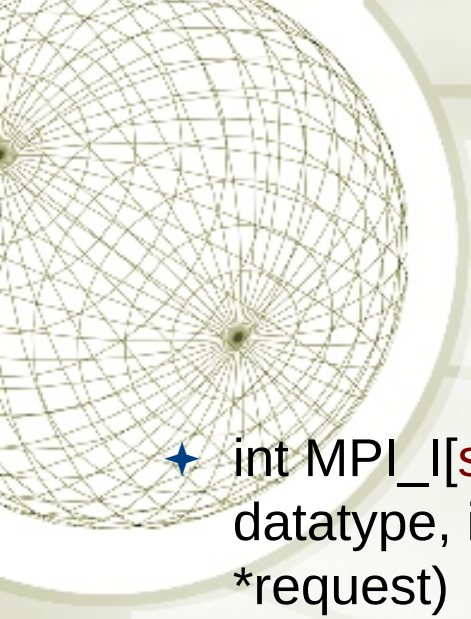
- Avoid deadlocks
- Reduce communication overhead
- Computation communication overlapping



Operation mode

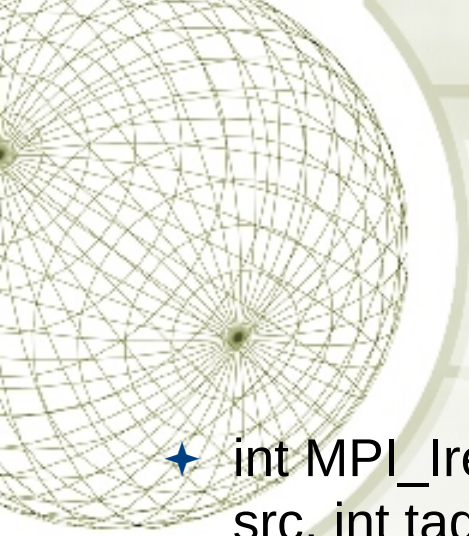
- Instantiate communication
- Do some useful work
- Wait? for communication





Non blocking send

- int MPI_I[ss|s|bs|rs]end(void*buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
- ...
- int MPI_Wait (MPI_Request *request, MPI_Status *status)

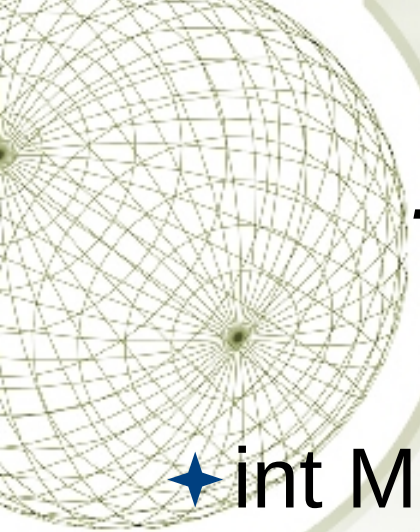


Non blocking receive

- int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int src, int tag,MPI_Comm comm, MPI_Request *handle)
- ...
- int MPI_Wait (MPI_Request *handle, MPI_Status *status)

Waiting for pending communications

- Waiting/Testing is a must
  - int MPI_Wait (MPI_Request *handle, MPI_Status *status)
  - int MPI_Waitall(...)
  - int MPI_Waitany(...)
  - int MPI_Waitsome(...)



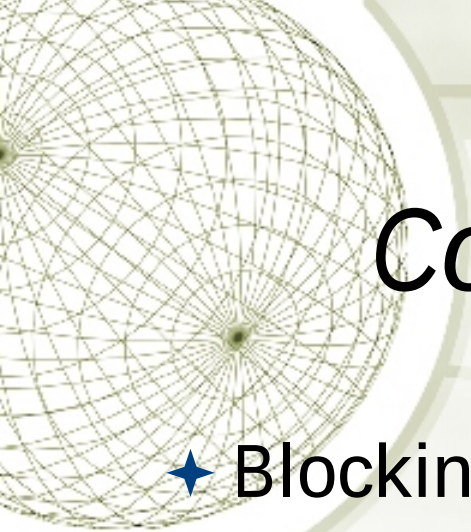
Testing for pending status

- int MPI_Test(MPI_Request request, int *flag, MPI_Status *st)
- int MPI_Testall(...)
- int MPI_Testsome (...)
- int MPI_Testany (...)

Persistent communications

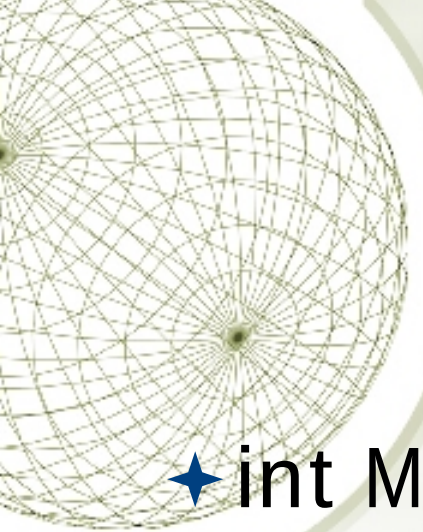
- Exchange same variables

```
Loop { ... ; a= ... ;
  MPI_Send ( &a, 1, MPI_INT, ...);
 }
MPI_Send_init (&a, 1, MPI_INT,...,&req);
Loop { ....; a=...; MPI_Start( &req); ....}
MPI_Request_free(&req);
```



Collective communcations

- Blocking, no tags
- Types
  - Synchronization
  - Data exchange (Broadcast, scatter, gather)
  - Reduction (global sum, global max,...)
- All processes in the communicator must participate
- Same function used by senders and receivers

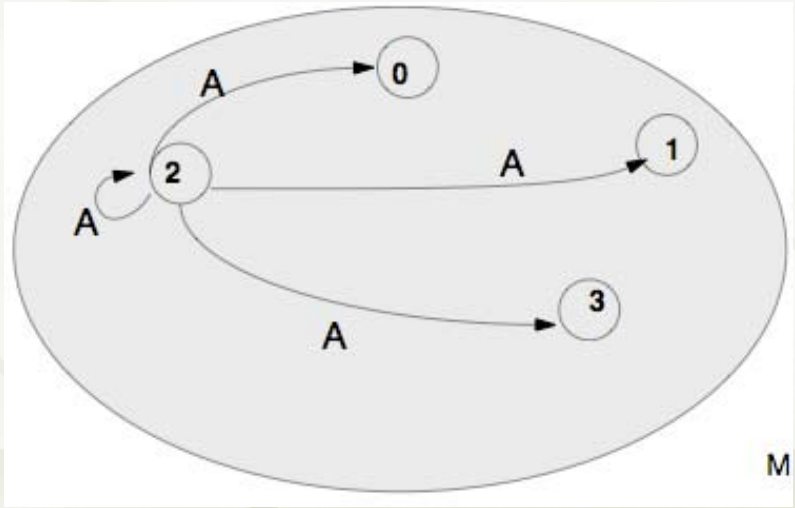


Synchronization Barrier

- int MPI_Barrier(MPI_Comm c)

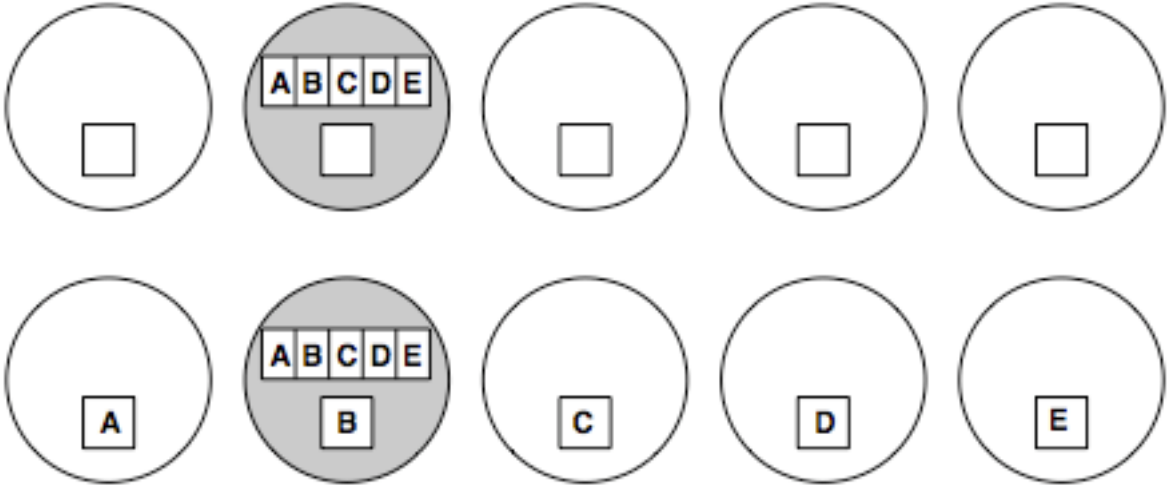
Broadcast

- int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm communicator)
- root is the sender, others are receivers



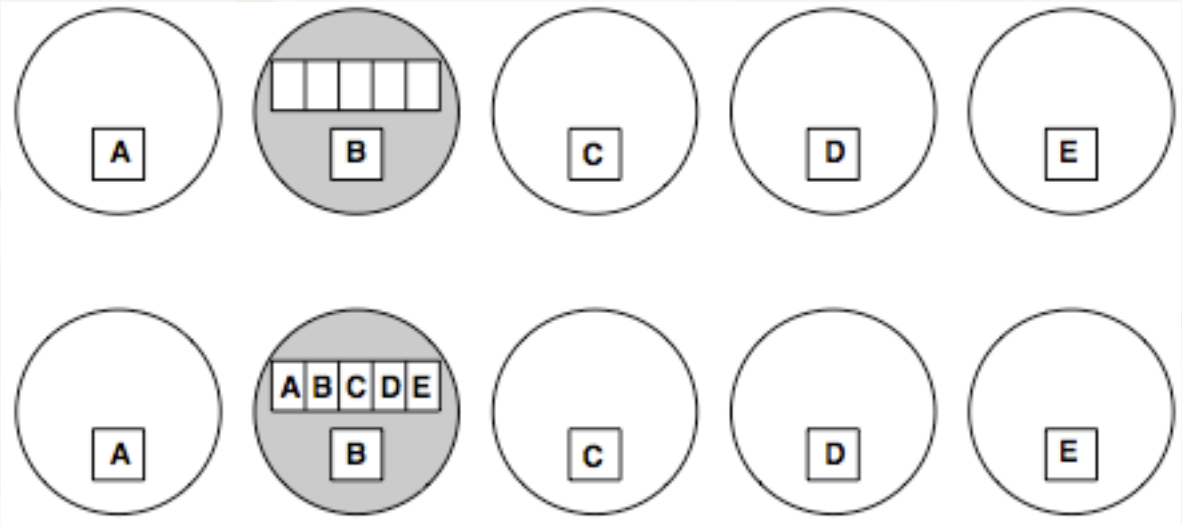
Scatter

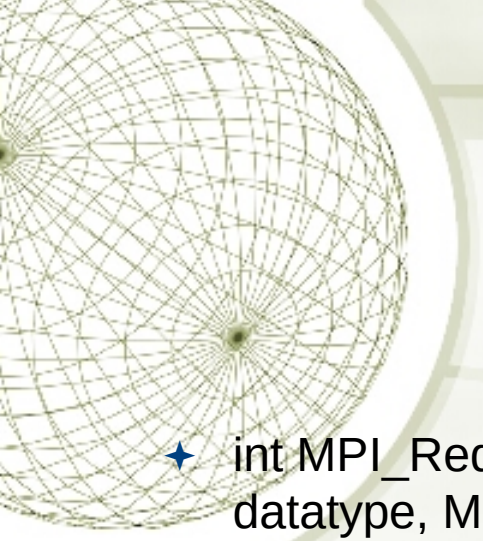
- MPI_Scatter (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)



Gather

- MPI_Gather (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)
- recvcnt=sendcnt=slice size

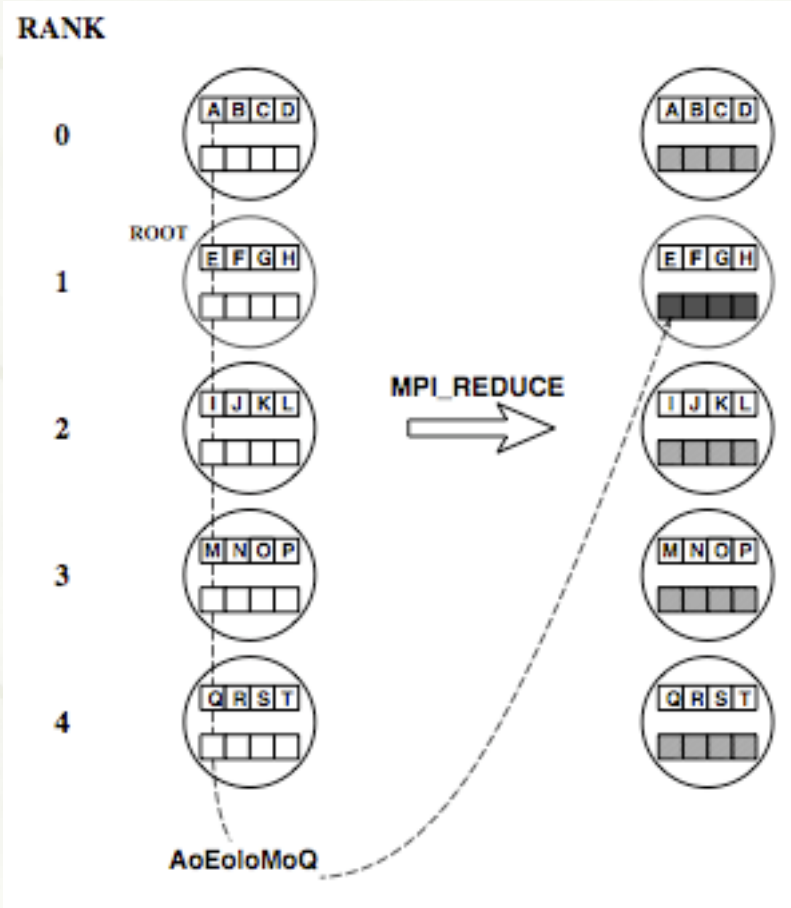


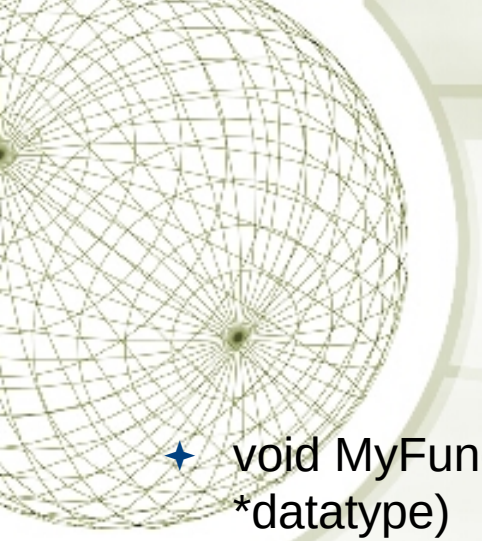


Reductions

- int MPI_Reduce (void *sendbuf,void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
  - MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, MPI_LOR, MPI_BOR, MPI_LXOR, MPI_BXOR, MPI_MAXLOC, MPI_MINLOC

Reductions


RANK
0
1
2
3
4
ROOT
A B C D
E F G H
I J K L
M N O P
Q R S T
MPI_REDUCE
AoEoIoMoQ



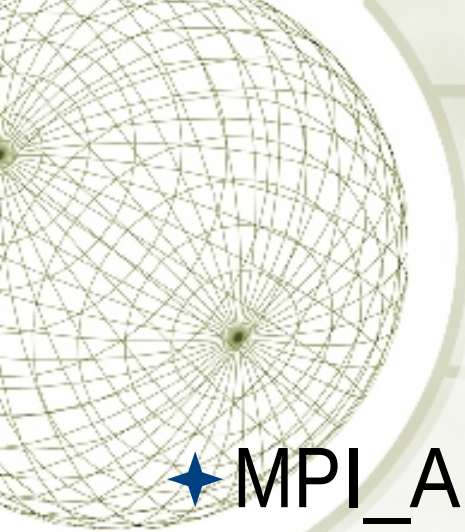
User defined reductions

- void MyFunction (void *invec, void *inoutvec, int *len, MPI_Datatype *datatype)

  for (i = 0; i < len ; i++)

  inoutvec(i) = inoutvec(i) o invec(i)

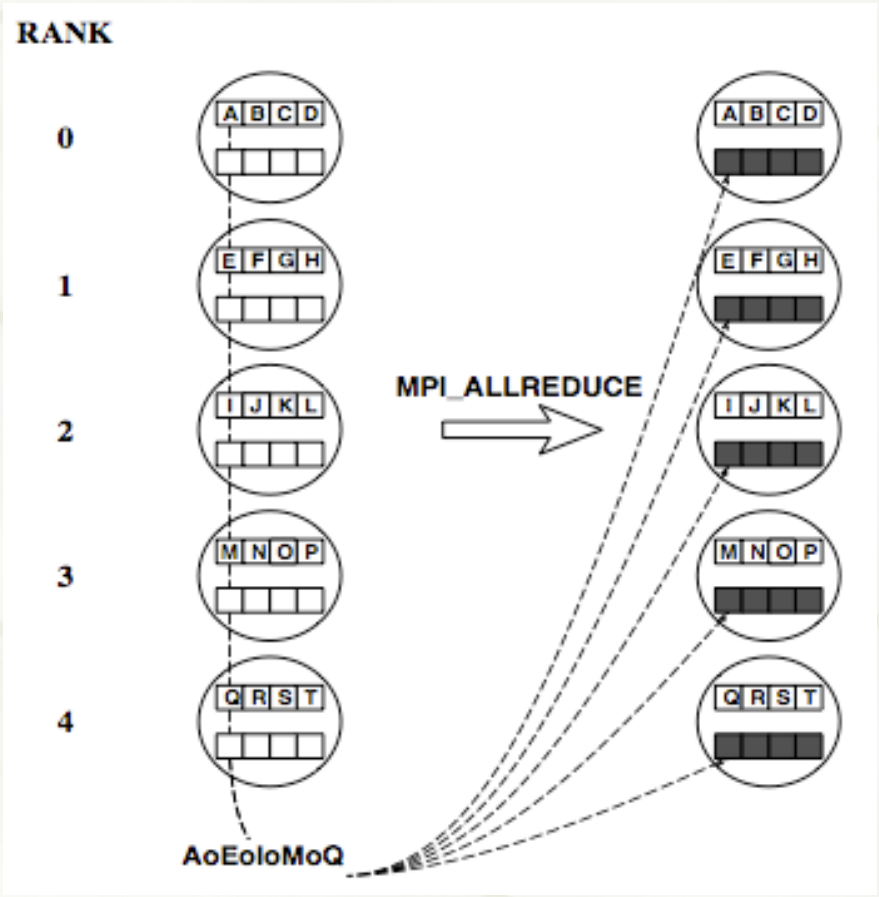
- int MPI_Op_create (MPI_User_function *function, int commute, MPI_Op *op)



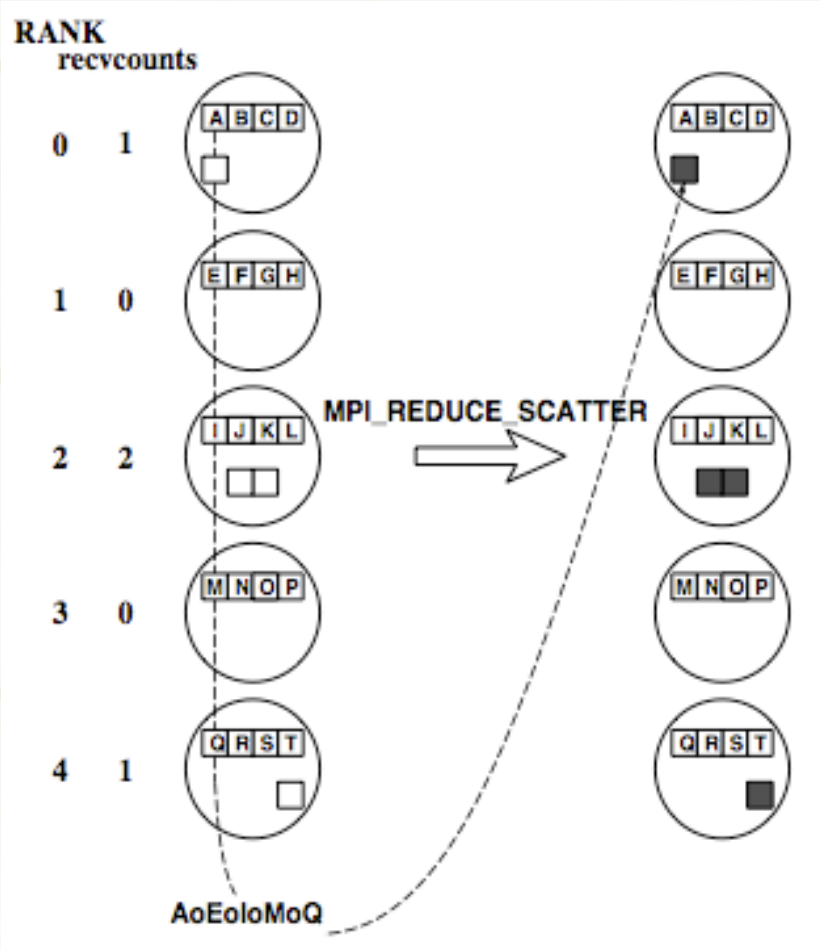
Reduction variants

- MPI_Allreduce
- MPI_Reduce_scatter
- MPI_Scan

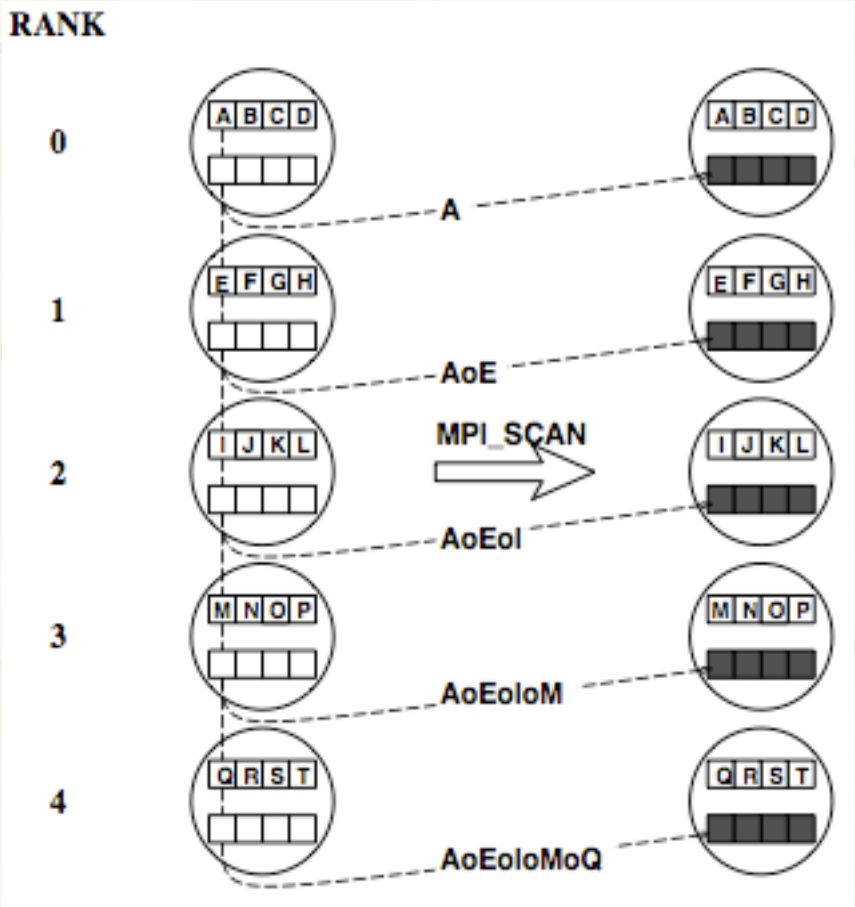
MPI_Allreduce



MPI_Reduce_scatter

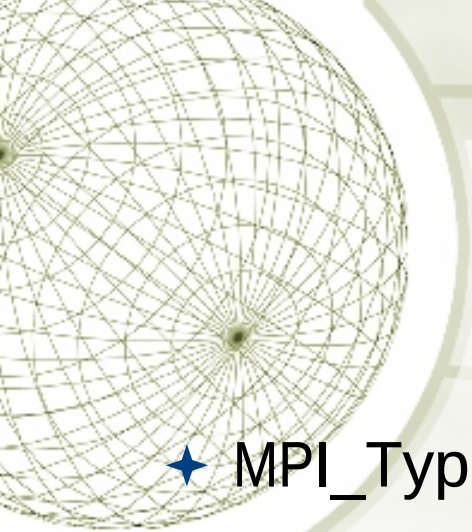


MPI_Scan


RANK
0
1
2
3
4
A B C D
E F G H
I J K L
M N O P
Q R S T
MPI_SCAN
A
AoE
AoEoI
AoEoIoM
AoEoIoMoQ

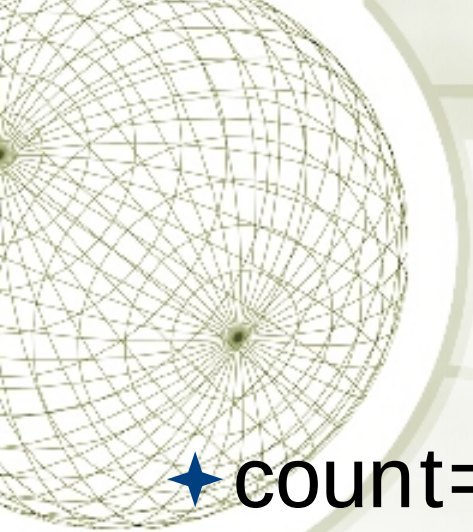
Complex data types

- Users can define complex data types
  - Vectors,
  - Structures
  - Other



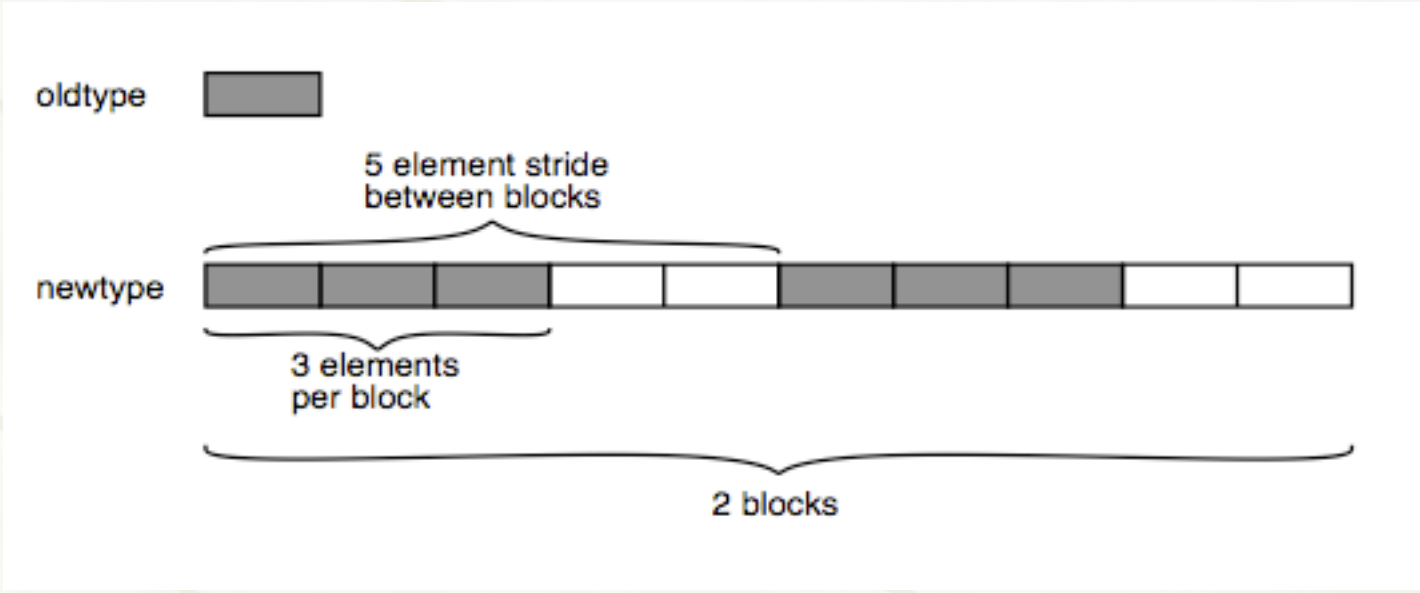
Complex data types

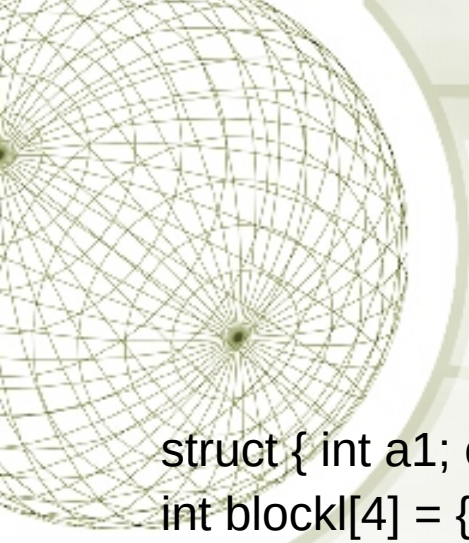
- MPI_Type_commit(MPI_Datatype *datatype)
- int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype *newtype)
- int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)
- int MPI_Type_struct (int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)



MPI_Type_vector

- count=2, stride=5, blocklength=3



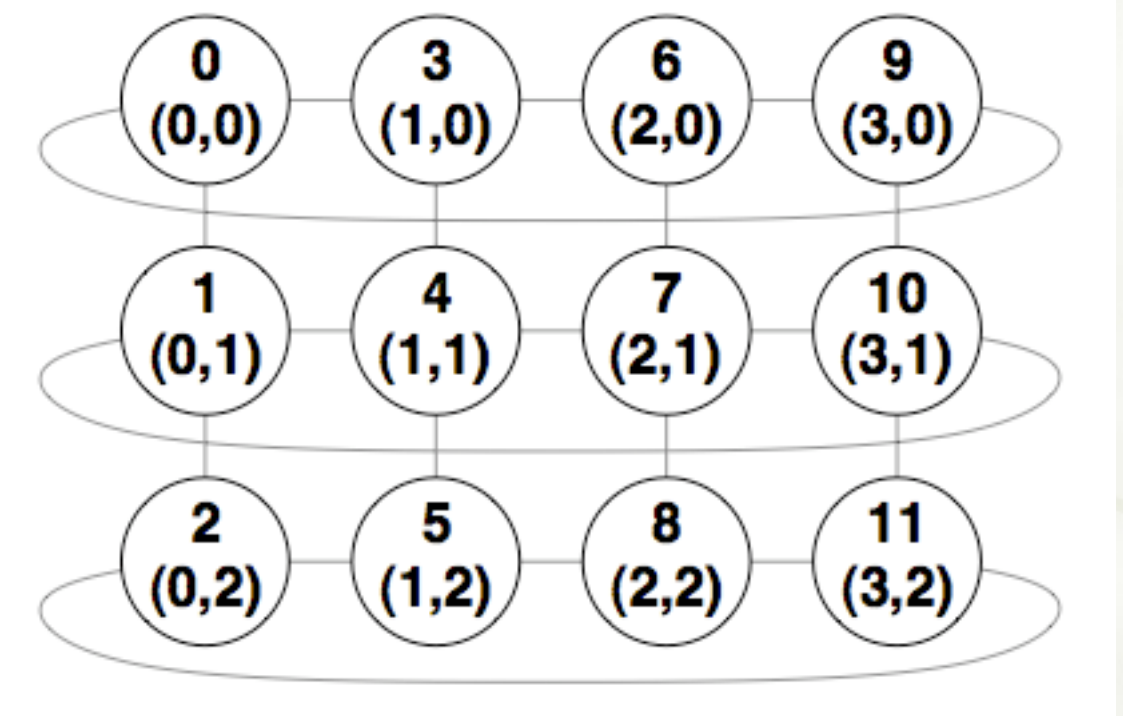


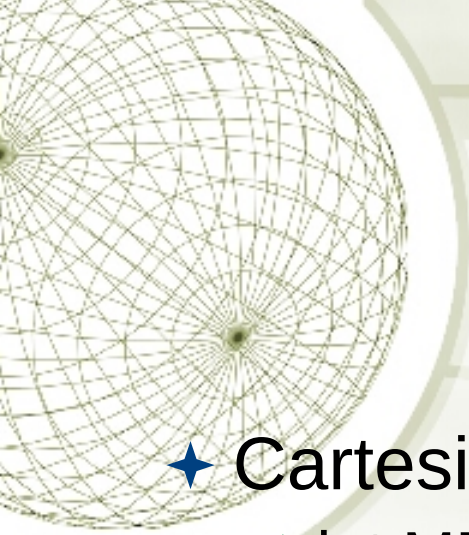
MPI_Type_struct

```
struct { int a1; char c1, c2; int a2; } essai;
int blockl[4] = {1, 1, 1, 1};
MPI_Aint disp[4];
MPI_Datatype types[4] = {MPI_INT, MPI_CHAR, MPI_CHAR, MPI_INT};
MPI_Datatype struct_t;
MPI_Address( &essai.a1, &disp[0] );
MPI_Address( &essai.c1, &disp[1] );
MPI_Address( &essai.c2, &disp[2] );
MPI_Address( &essai.a2, &disp[3] );
for (i=1; i<4; i++)
    disp[i] = disp[i] - disp[0];
disp[0] = 0;
MPI_Type_struct(4, blockl, disp, types, &struct_t);
MPI_Type_commit(&struct_t);
```

Defining new topologies

- Enhance readability





Supported Topologies

- Cartesian
  - int MPI_Cart_create (MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)
  - int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)
  - int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int *coords)
  - int MPI_Cart_shift (MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)
- Graph
  - ....