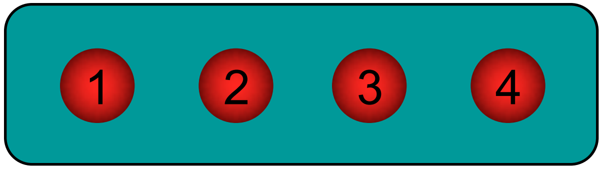
Intra and Inter Communicators

Innovative Computing Laboratory
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE

Groups

- A group is a set of processes
  - The group have a size
  - And each process have a rank
- Creating a group is a local operation
- Why we need groups
  - To make a clear distinction between processes
  - To allow communications in-between subsets of processes
  - To create intra and inter communicators …



Groups

- MPI_GROUP_*(group1, group2, newgroup)
  - Where * ∈ {UNION, INTERSECTION, DIFFERENCE}
  - Newgroup contain the processes satisfying the * operation ordered first depending on the order in group1 and then depending on the order in group2.
  - In the newgroup each process could be present only one time.
- There is a special group without any processes MPI_GROUP_EMPTY.

Groups

- group1 = {a, b, c, d, e}
- group2 = {e, f, g, b, a}
- Union
  - newgroup = {a, b, c, d, e, f, g}
- Difference
  - newgroup = {c, d}
- Intersection
  - newgroup = {a, b, e}

Groups

- MPI_GROUP_*(group, n, ranks, newgroup)
  - Where * $\in$ {INCL, EXCL}
  - N is the number of valid indexes in the ranks array.

- For INCL the order in the result group depend on the ranks order
- For EXCL the order in the result group depend on the original order

Groups

- Group = {a, b, c, d, e, f, g, h, i, j}
- N = 4, ranks = {3, 4, 1, 5}
- INCL
    - Newgroup = {c, d, a, e}
- EXCL
    - Newgroup = {b, c, f, g, h, i, j}

Groups

- MPI_GROUP_RANGE_*(group, n, ranges, newgroup)
  - Where * ∈ {INCL, EXCL}
  - N is the number of valid entries in the ranges array
  - Ranges is a tuple (start, end, stride)
- For INCL the order in the new group depend on the order in ranges
- For EXCL the order in the new group depend on the original order

Groups

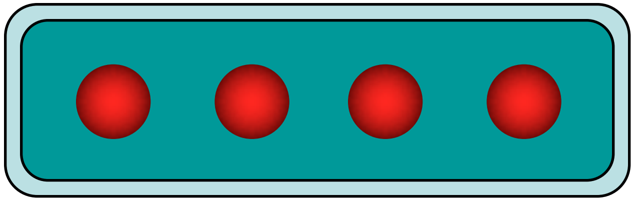
- Group = {a, b, c, d, e, f, g, h, i, j}
- N=3; ranges = ((6, 7, 1), (1, 6, 2), (0, 9, 4))
- Then the range
  - (6, 7, 1) => {g, h} (ranks (6, 7))
  - (1, 6, 2) => {b, d, f} (ranks (1, 3, 5))
  - (0, 9, 4) => {a, e, i} (ranks (0, 4, 8))
- INCL
  - Newgroup = {g, h, b, d, f, a, e, i}
- EXCL
  - Newgroup = {c, j}

Communicators

- A special channel between some processes used to exchange messages.
- Operations creating the communicators are collectives, but accessing the communicator information is a local operation.
- Special communicators: MPI_COMM_WORLD, MPI_COMM_NULL, MPI_COMM_SELF
- MPI_COMM_DUP(comm, newcomm) create an identical copy of the comm in newcomm.
  - Allow exchanging messages between the same set of nodes using identical tags (useful for developing libraries).

Intracommunicators

- What exactly is a intracommunicator ?



- some processes

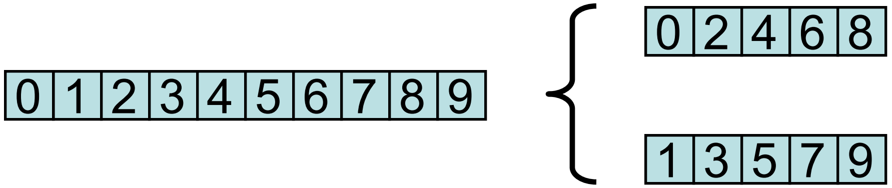
- **ONE** group

- one communicator

- MPI_COMM_SIZE, MPI_COMM_RANK
- MPI_COMM_COMPARE(comm1, comm2, result)
  - MPI_IDENT: comm1 and comm2 represent the same communicator
  - MPI_CONGRUENT: same processes, same ranks
  - MPI_SIMILAR: same processes, different ranks
  - MPI_UNEQUAL: otherwise

Intracommunicators

- MPI_COMM_CREATE(comm, group, newcomm)
  - Create a new communicator on all processes from the communicator comm who are defined on the group.
  - All others processes get MPI_COMM_NULL



0 1 2 3 4 5 6 7 8 9

0 2 4 6 8

1 3 5 7 9

```
MPI_Group_range_excl( group, 1, (0, 9, 2), odd_group );
MPI_Group_range_excl( group, 1, (1, 9, 2), even_group );
MPI_Comm_create( comm, odd_comm, odd_comm );
MPI_Comm_create( comm, even_group, even_comm );
```

Intracommunicators

- MPI_COMM_SPLIT(comm, color, key, newcomm)
  - Color : control of subset assignment
  - Key : control of rank assignement

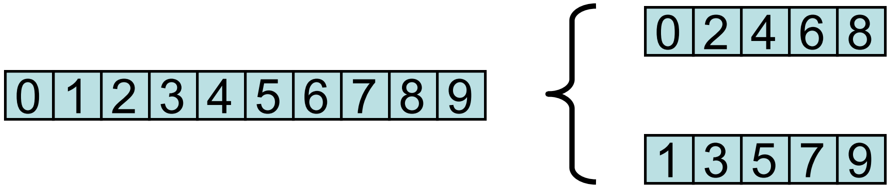
rank	0	1	2	3	4	5	6	7	8	9
process	A	B	C	D	E	F	G	H	I	J
color	0	⊥	3	0	3	0	0	5	3	⊥
key	3	1	2	5	1	1	1	2	1	0

3 different colors => 3 communicators

1. {A, D, F, G} with ranks {3, 5, 1, 1} => {F, G, A, D}
2. {C, E, I} with ranks {2, 1, 3} => {E, I, C}
3. {H} with ranks {1} => {H}

B and J get MPI_COMM_NULL as they provide an undefined color (MPI_UNDEFINED)

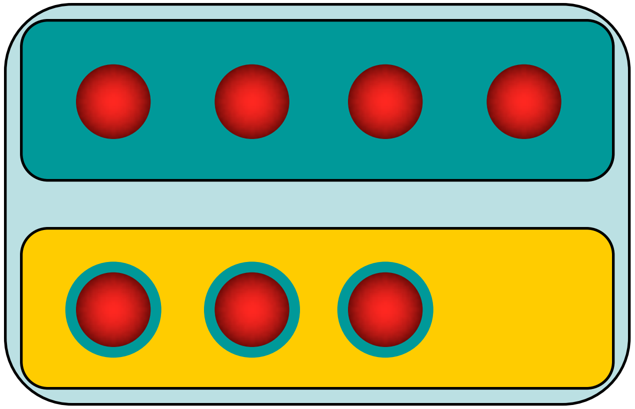
Intracommunicators



Rank	0	1	2	3	4	5	6	7	8	9
process	A	B	C	D	E	F	G	H	I	J
Color	0	1	0	1	0	1	0	1	0	1
Key	1	1	1	1	1	1	1	1	1	1

Intercommunicators

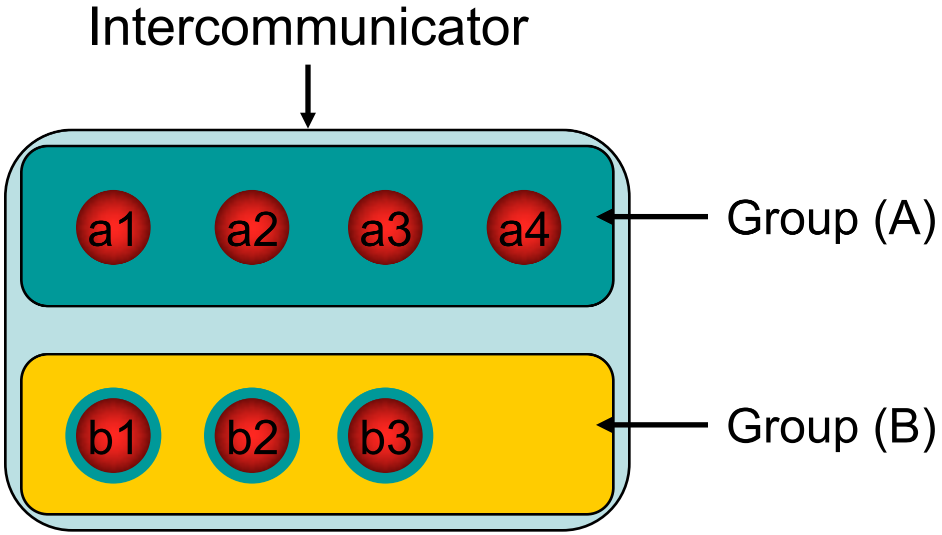
- And what’ s a intercommunicator ?



- some more processes
- **TWO** groups
- one communicator

- MPI_COMM_REMOTE_SIZE(comm, size)
  MPI_COMM_REMOTE_GROUP(comm, group)
- MPI_COMM_TEST_INTER(comm, flag)
- MPI_COMM_SIZE, MPI_COMM_RANK return the local size respectively rank

Anatomy of a Intercommunicator

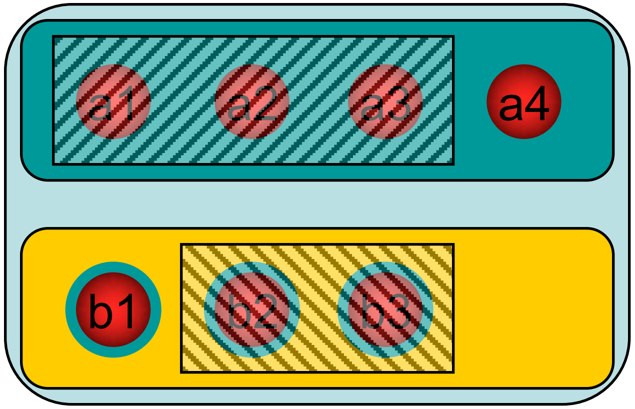


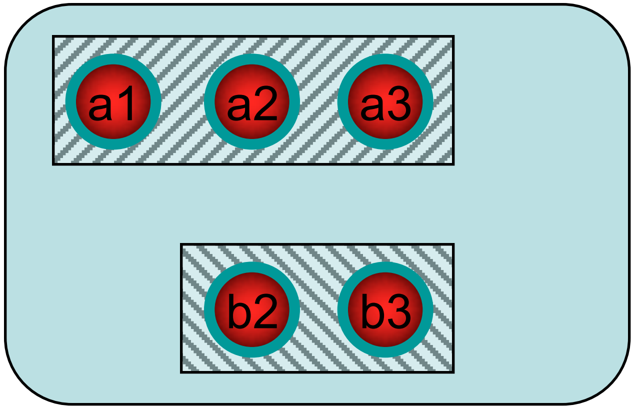
It' s not possible to send a message to a process in the same group using this communicator

For any processes from group (A)	For any processes from group (B)
• (A) is the local group • (B) is the remote group	• (A) is the remote group • (B) is the local group

Intercommunicators

- MPI_COMM_CREATE(comm, group, newcomm)
  - All processes on the left group should execute the call with the same subgroup of processes, when all processes from the right side should execute the call with the same subgroup of processes. Each of the subgroup is related to a different side.





Intercommunicators

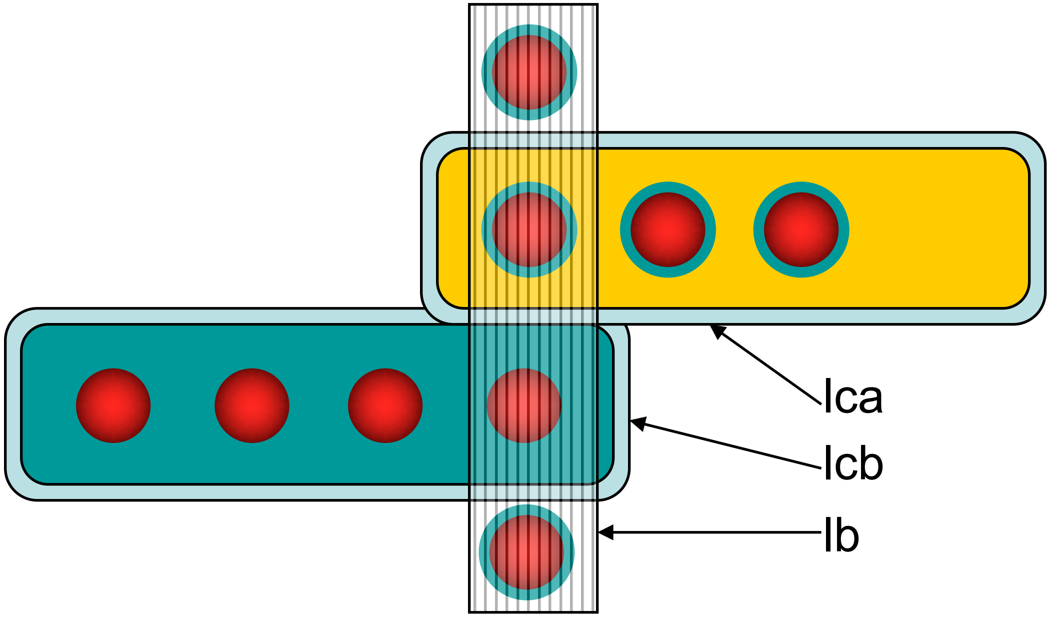
- MPI_INTERCOMM_CREATE(local_comm, local_leader, bridge_comm, remote_leader, tag, newintercomm)

  Local_comm : local intracommunicator

  Local_leader : rank of root in the local_comm

  Bridge_comm : “bridge” communicator …

  Remote_leader : rank of remote leader in bridge_comm



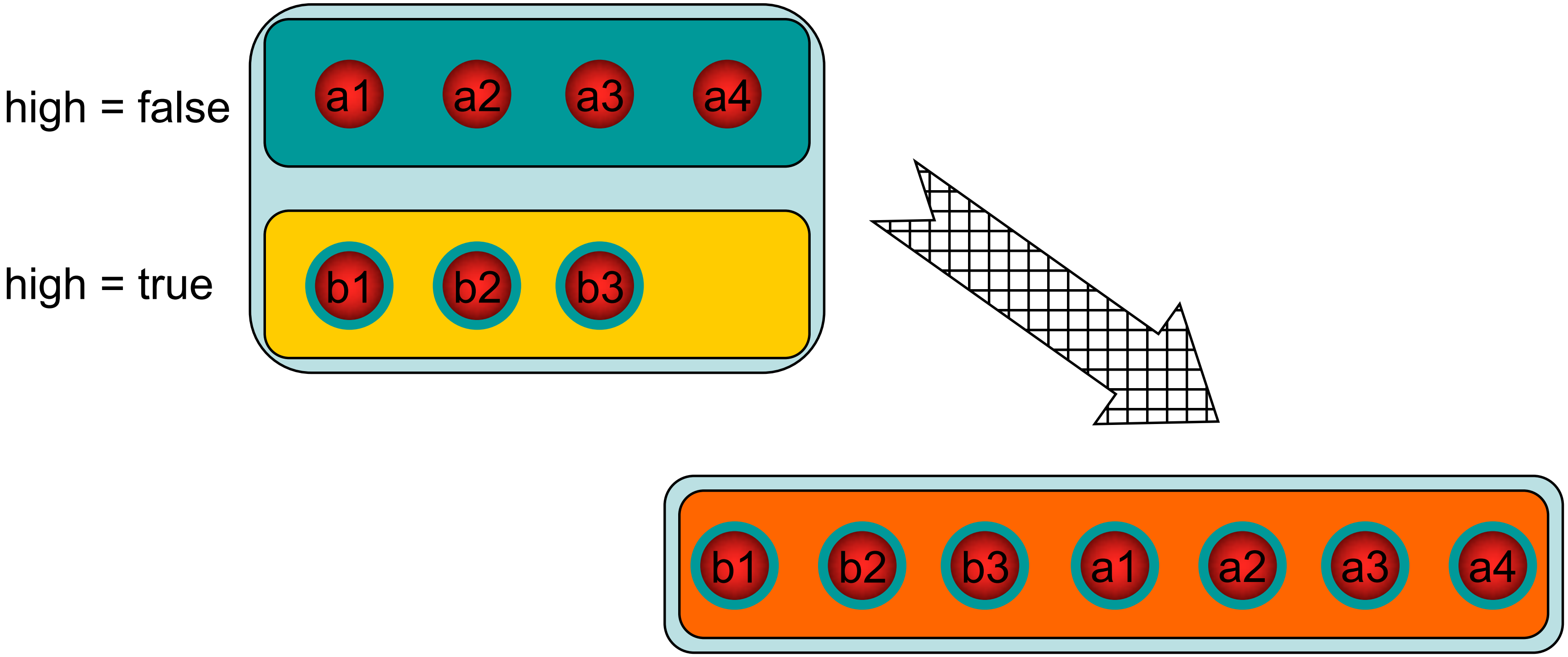
MPI_INTERCOMM_CREATE

lca, 0, lb, 2, tag, new

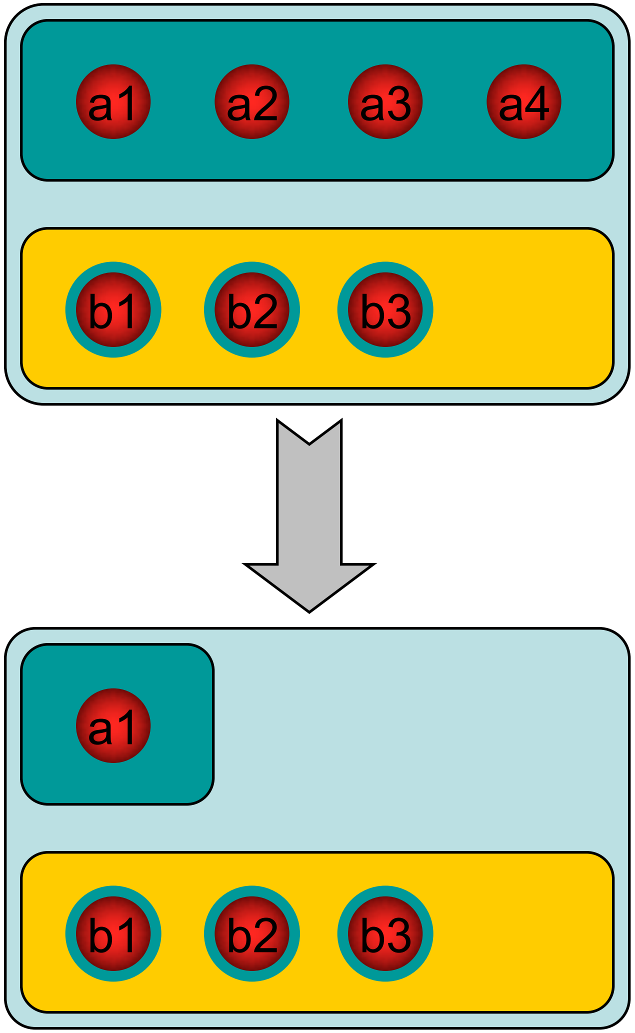
lcb, 4, lb, 1, tag, new

Intercommunicators

- MPI_INTERCOMM_MERGE(intercomm, high, intracomm)
  - Create an intracomm from the union of the two groups
  - The order of processes in the union respect the original one
  - The high argument is used to decide which group will be first (rank 0)



Example

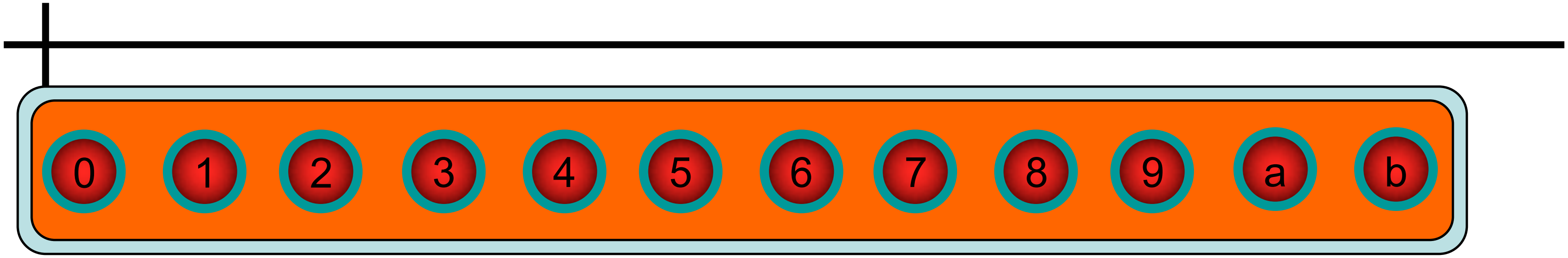


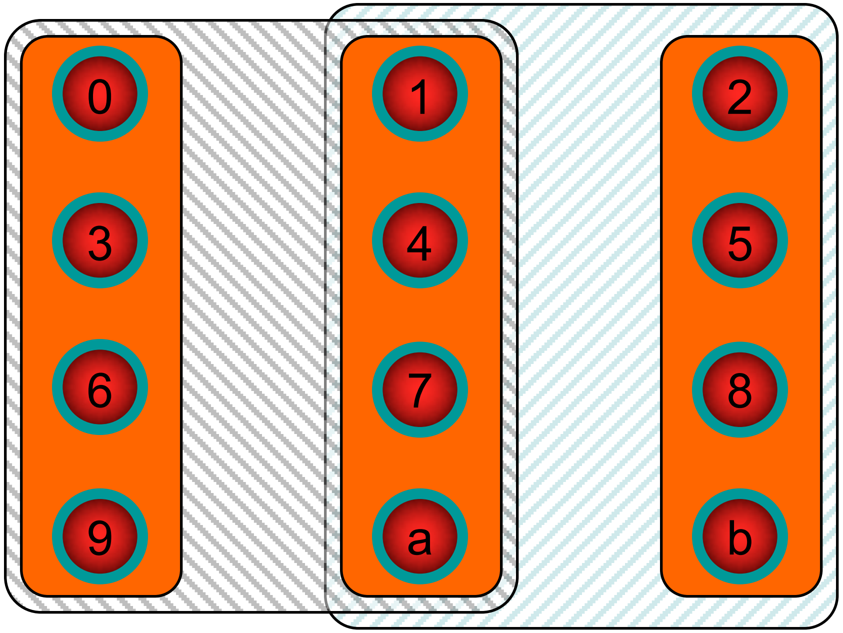
```
MPI_Comm inter_comm, new_inter_comm;
MPI_Group local_group, group;
int rank = 0;

if( /* left side (ie. a*) */ ) {
  MPI_Comm_group( inter_comm, &local_group);
  MPI_Group_incl( local_group, 1, &rank, &group);
  MPI_Group_free( &local_group );
} else
  MPI_Comm_group( inter_comm, &group );

MPI_Comm_create( inter_comm, group,
                                 &new_inter_comm );
MPI_Group_free( &group );
```

Exercice



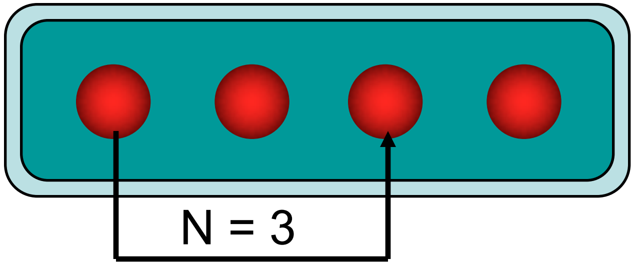


Intercommunicators – P2P

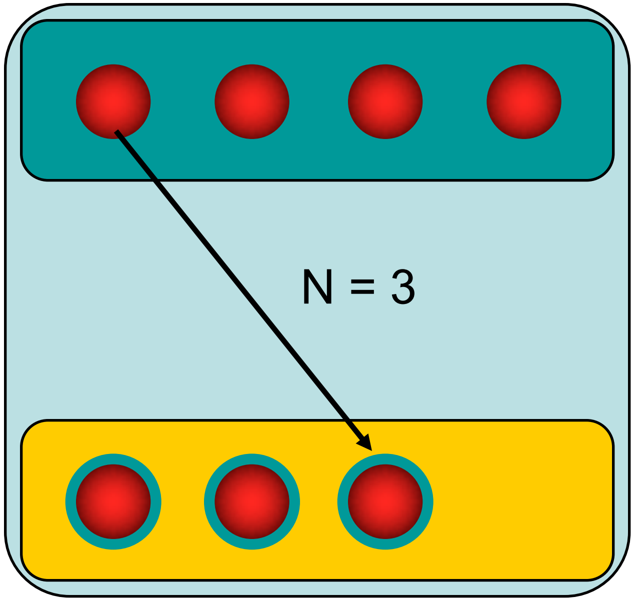
On process 0:

```
MPI_Send( buf, MPI_INT, 1, n, tag, intercomm )
```

- Intracommunicator



- Intercommunicator

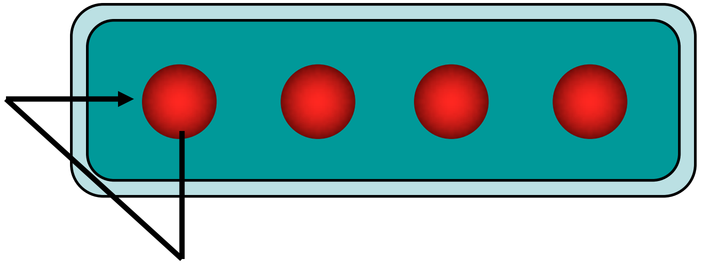


Intercommunicators– P2P

On process 0:

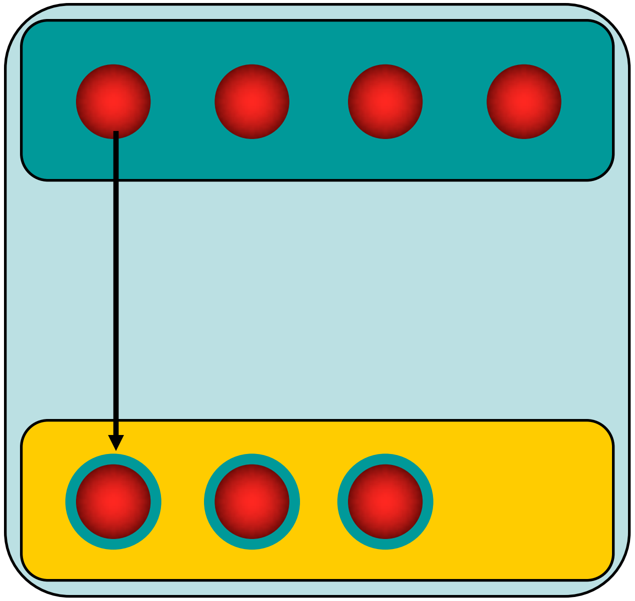
MPI_Send(buf, MPI_INT, 1, 0, tag, intercomm)

- Intracommunicator



Not MPI safe if the receive was not posted before.

- Intercommunicator



Communicators - Collectives

- Simple classification by operation class
- **One-To-All** (simplex mode)
  - One process contributes to the result. All processes receive the result.
    - MPI_Bcast
    - MPI_Scatter, MPI_Scatterv
- **All-To-One** (simplex mode)
  - All processes contribute to the result. One process receives the result.
    - MPI_Gather, MPI_Gatherv
    - MPI_Reduce
- **All-To-All** (duplex mode)
  - All processes contribute to the result. All processes receive the result.
    - MPI_Allgather, MPI_Allgatherv
    - MPI_Alltoall, MPI_Alltoallv
    - MPI_Allreduce, MPI_Reduce_scatter
- **Other**
  - Collective operations that do not fit into one of the above categories.
    - MPI_Scan
    - MPI_Barrier

Collectives

	Who generate the result	Who receive the result
One-to-all	One in the local group	All in the local group
All-to-one	All in the local group	One in the local group
All-to-all	All in the local group	All in the local group
Others	?	?

Extended Collectives

From each process point of view

	Who generate the result	Who receive the result
One-to-all	One in the local group	All in the remote group
All-to-one	All in the local group	One in the remote group
All-to-all	All in the local group	All in the remote group
Others	?	?

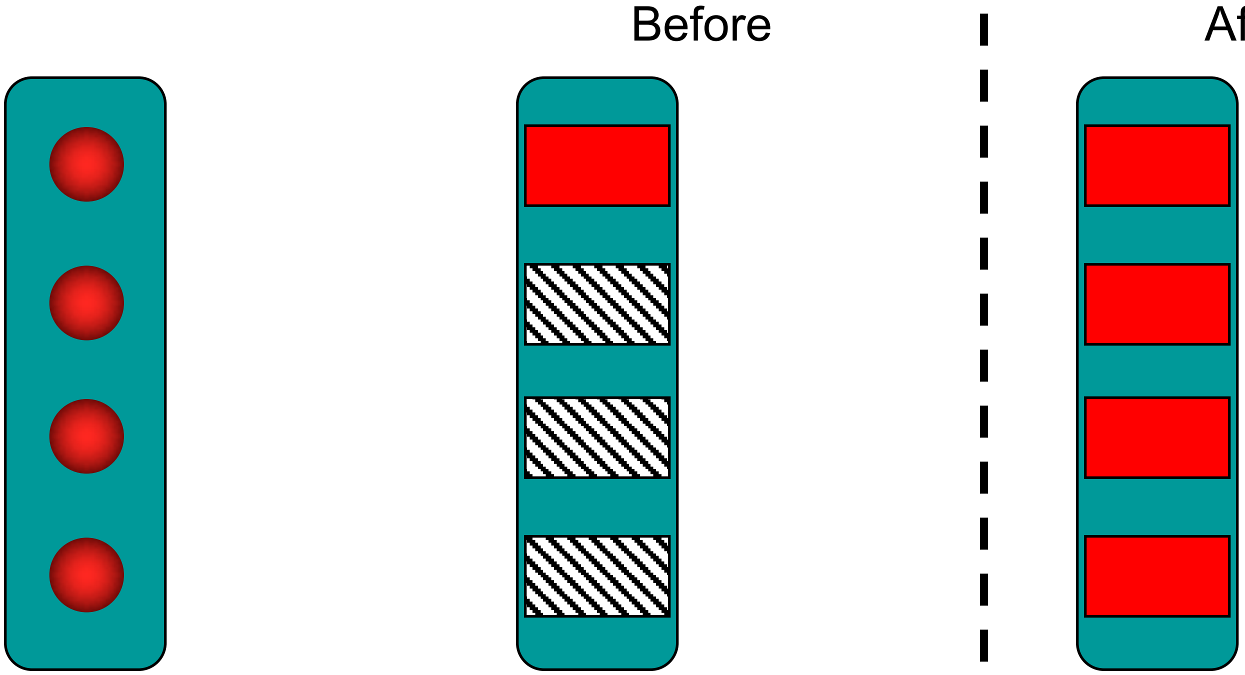
Extended Collectives

- Simplex mode (ie. rooted operations)
  - A root group
    - The root use MPI_ROOT as root process
    - All others use MPI_PROC_NULL
  - A second group
    - All use the real rank of the root in the remote group
- Duplex mode (ie. non rooted operations)
  - Data send by the process in one group is received by the process in the other group and vice-versa.

Broadcast

One-to-all	One in the local group	All in the local group

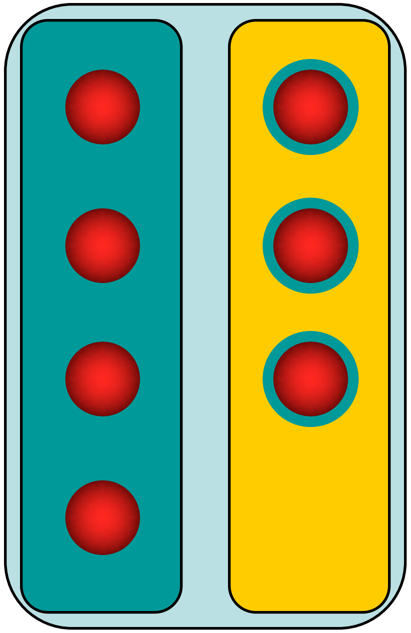
MPI_Bcast(buf, 1, MPI_INT, 0, intracomm)

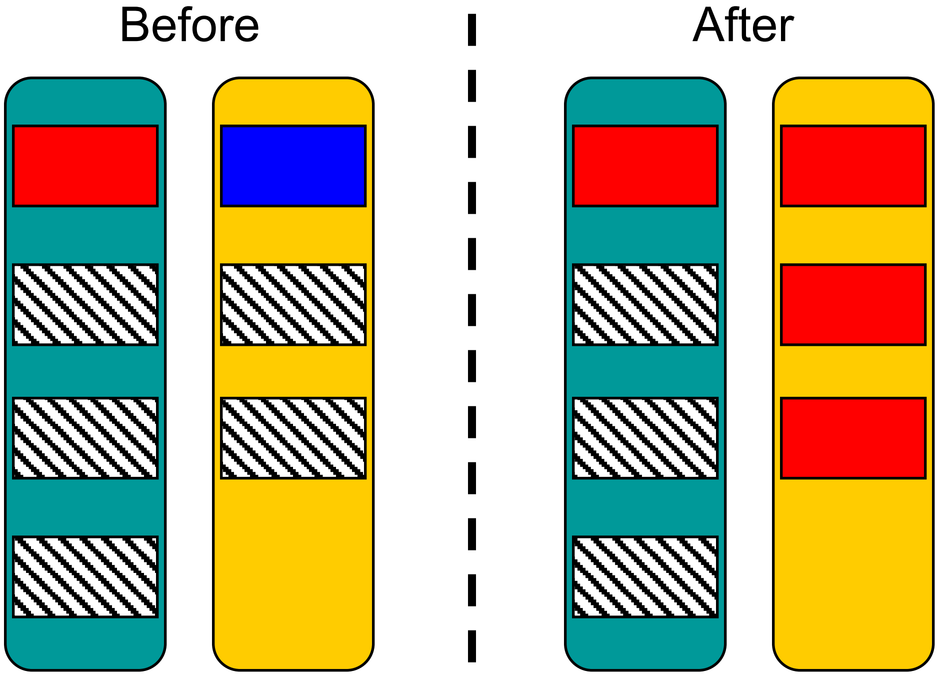


Extended Broadcast

One-to-all	One in the local group	All in the remote group

```
Root group root process:        MPI_Bcast( buf, 1, MPI_INT, MPI_ROOT, intercomm )
Root group other processes: MPI_Bcast( buf, 1, MPI_INT, MPI_PROC_NULL, intercomm )
Other group                        MPI_Bcast( buf, 1, MPI_INT, root_rank, intercomm )
```

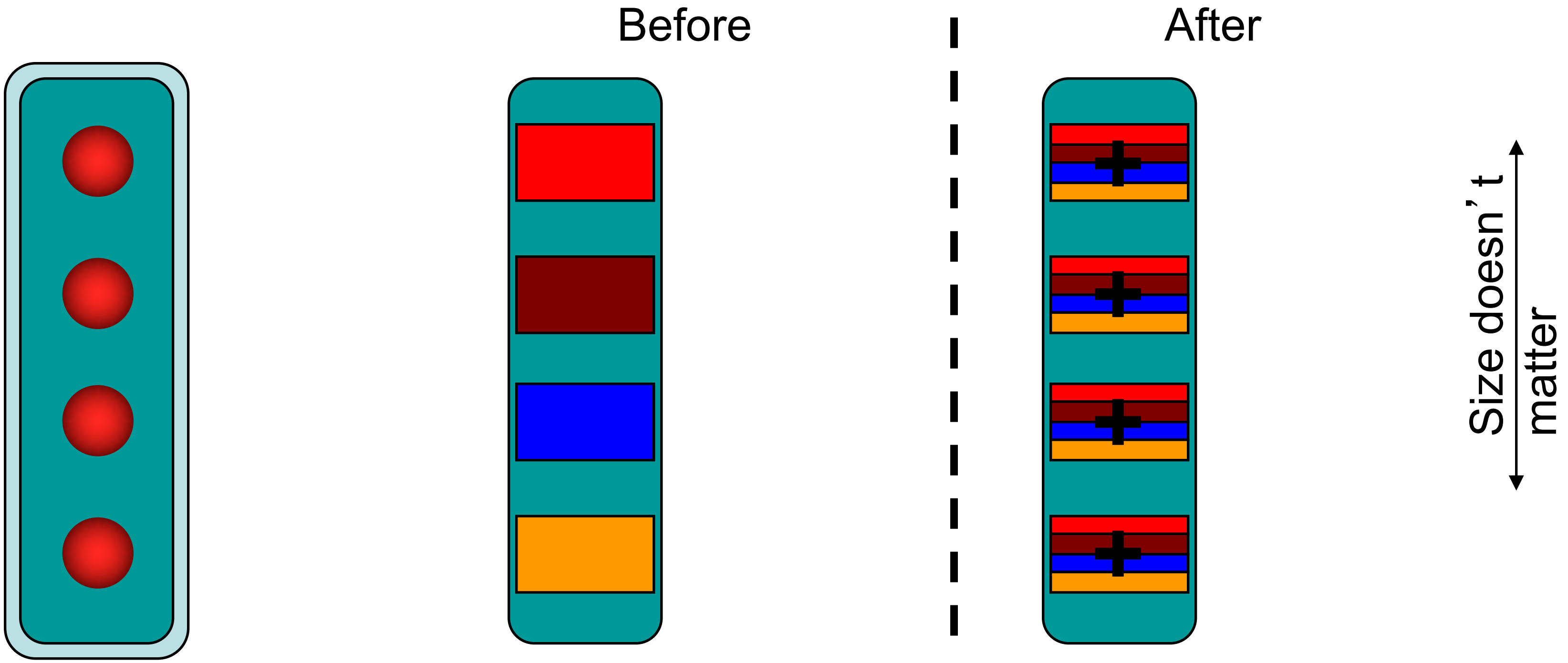




Allreduce

All-to-all	All in the local group	All in the local group

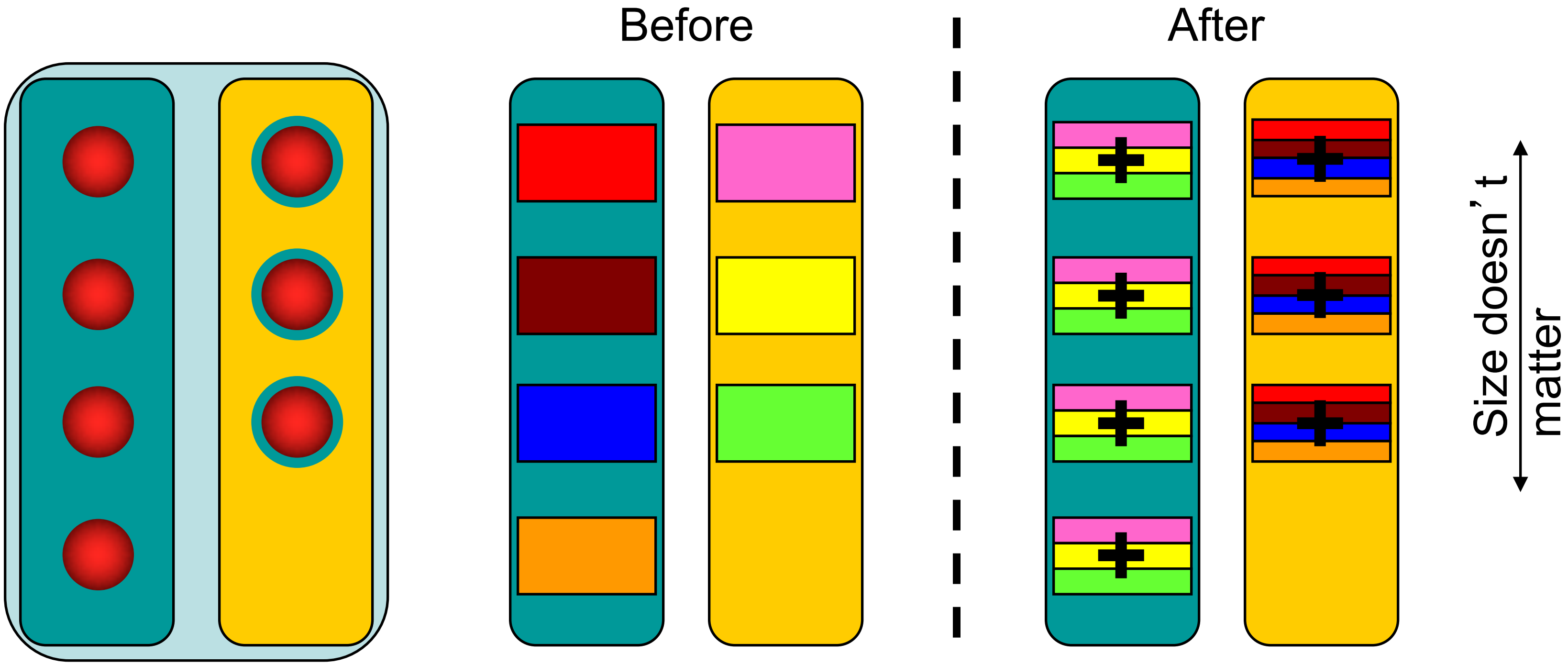
MPI_Allreduce(sbuf, rbuf, 1, MPI_INT, +, intracomm)



Extended Allreduce

All-to-all	All in the local group	All in the remote group

```
MPI_Allreduce( sbuf, rbuf, 1, MPI_INT, +, intercomm )
```



Before

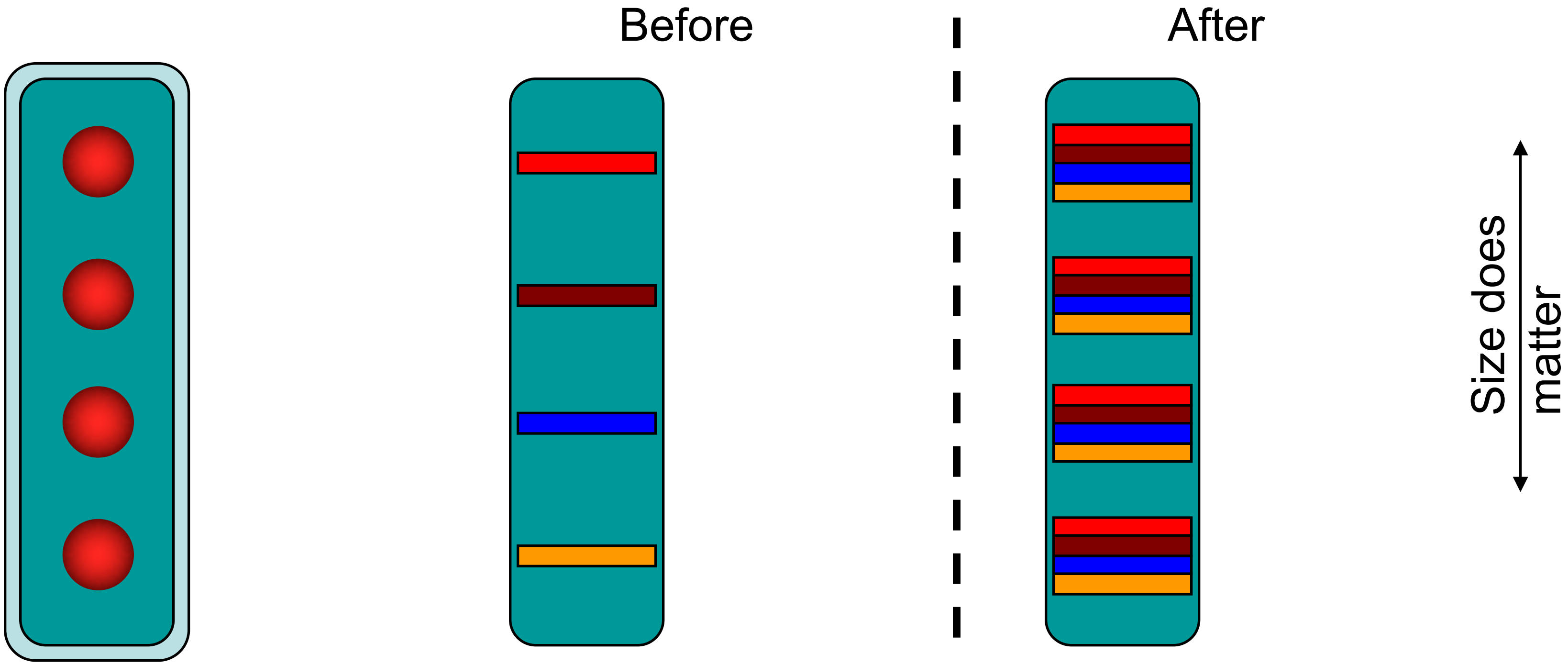
After

Size doesn’t matter

AllGather

All-to-all	All in the local group	All in the local group

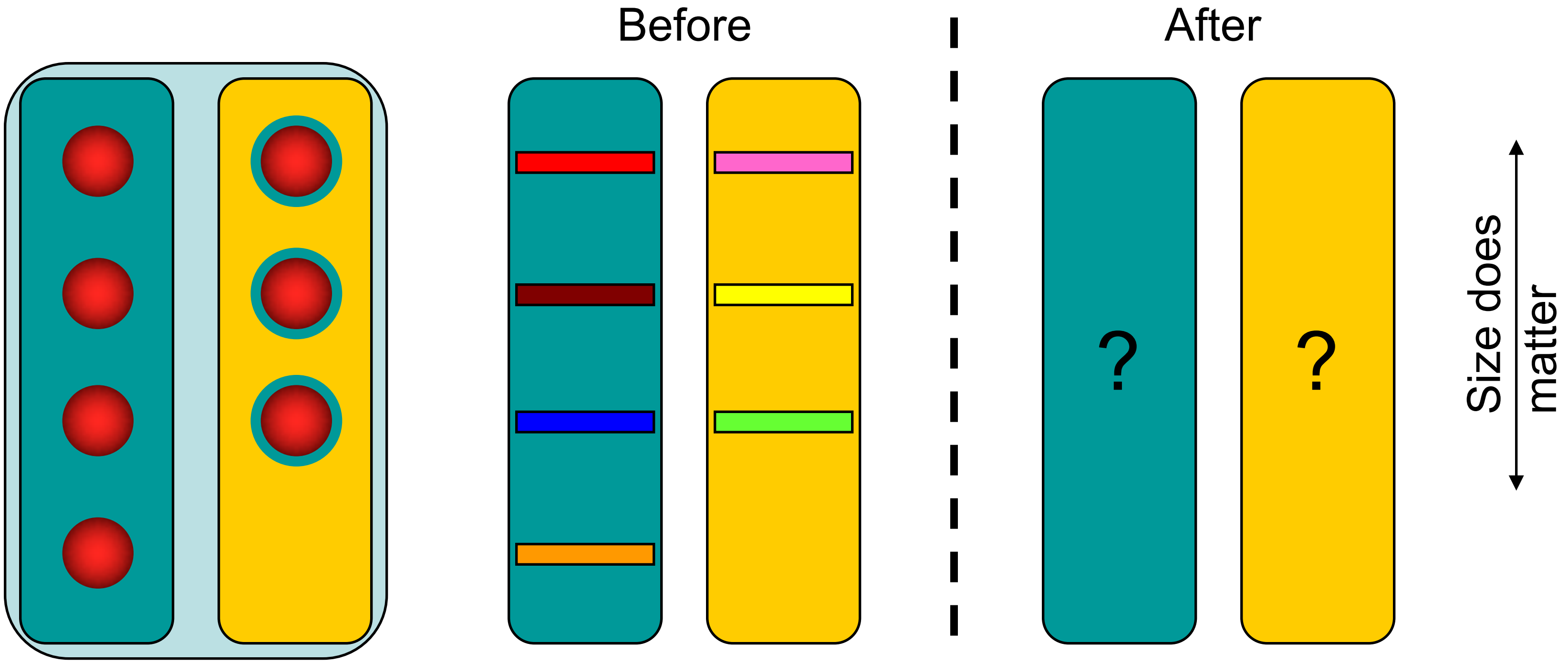
MPI_Allgather(sbuf, 1, MPI_INT, rbuf, 1, MPI_INT, intracomm)



Extended AllGather

All-to-all	All in the local group	All in the remote group

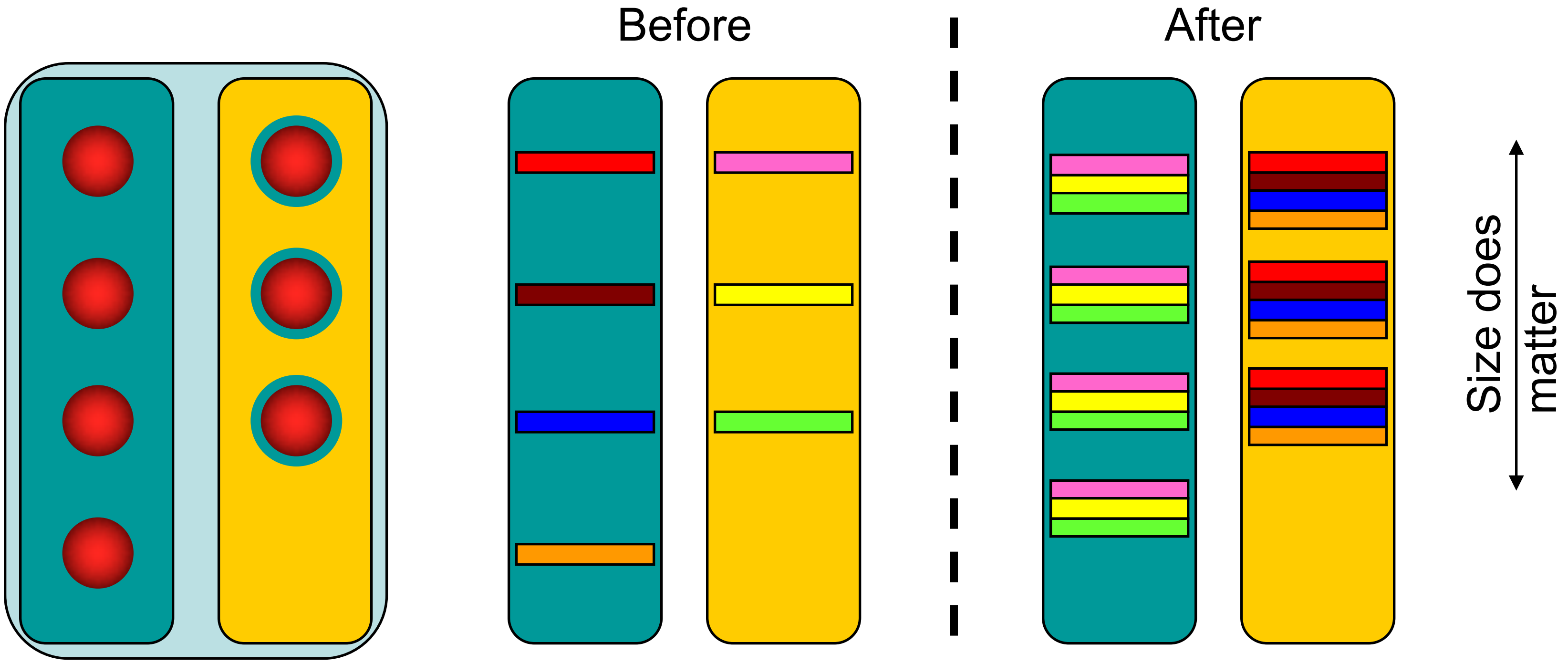
MPI_Allgather(sbuf, 1, MPI_INT, rbuf, 1, MPI_INT, intercomm)



Extended AllGather

All-to-all	All in the local group	All in the remote group

```
MPI_Allgather( sbuf, 1, MPI_INT, rbuf, 1, MPI_INT, intercomm )
```



Scan/Exscan and Barrier

- Scan and Exscan are illegal on intercommunicators
- For MPI_Barrier all processes in a group may exit the barrier when all processes on the other group have entered in the barrier.