MPI Communicators

Dr. Süha Tuna

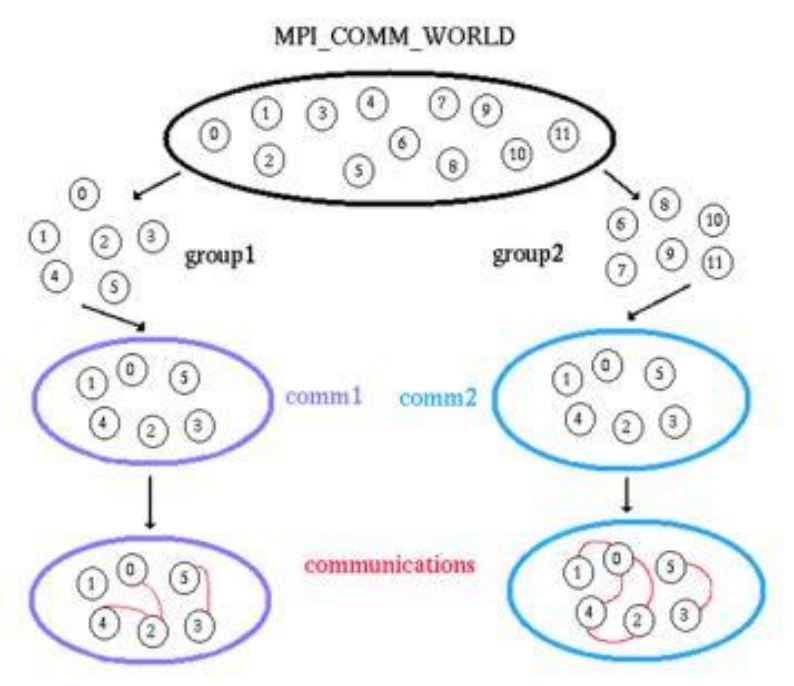
FSMVU – Department of Computer Engineering

Content

1. Introduction
2. Communicator types
3. Working with groups, contexts and communicators
4. Examples
5. Further MPI routines

Introduction

- Communicators provides a separate communication space. It is possible to treat a subset of processes as a communication universe.
- Can create sub-groups of processes, or sub-communicators



FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ 2010

Types of communicators

- **Intra-communicator:** a collection of processes that can send messages to each other and engage in collective communication operations. (`MPI_COMM_WORLD`)

- **Inter-communicator:** are used for sending messages between processes belonging to disjoint intra-communicators. (`comm1`, `comm2`)

Intra-communicator

- An intra-communicator is composed of:
  - A **group**: is an ordered collection of processes. If a group consists of p processes, each process in the group is assigned a unique rank, which is a nonnegative integer in the range $0, 1, \ldots, p-1$.
  - A **context**: a system-defined object that uniquely identifies a communicator. Two distinct communicators have different contexts, even if they have identical underlying groups.
  - **Attributes**: topology
- Remark: A minimal intra-communicator has at least a group and a context.

Working with Groups, Contexts and Communicators

Assume that there are p processes under MPI_COMM_WORLD, where $q^2 = p$.

```
MPI_Group group_world;
MPI_Group first_row_group;
MPI_Comm first_row_comm;
int *process_ranks;

// make a list of processes in the new communicator
process_ranks = (int*) malloc(q*sizeof(int));
for(int I = 0; I < q; I++)
   process_ranks[I] = I;

//get the group under MPI_COMM_WORLD
MPI_Comm_group(MPI_COMM_WORLD, &group_world);

// create the new group
MPI_Group_incl(group_world, q, process_ranks, &first_row_group);

// create the new communicator
MPI_Comm_create(MPI_COMM_WORLD, first_row_group, &first_row_comm);
```

MPI_Comm_group routine

- `int MPI_Comm_group( MPI_Comm comm, MPI_Group *group );`

- Obtain the `group` associated with the `comm`.
  - `comm`: communicator
  - `group`: group in communicator (handle)

MPI_Group_incl routine

- `int MPI_Group_incl( MPI_Group group, int n, int *ranks, MPI_Group *newgroup );`

- Produces a group by reordering an existing group and taking only listed members
  - `n`: number of elements in array `ranks`
  - `ranks`: ranks of processes in group to appear in `newgroup`
  - `newgroup`: new group constructed

MPI_Comm_create routine

- `int MPI_Comm_create( MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm );`

- Creates a new communicator, which implicitly associate context and group.

Local and collective operations

- `MPI_Comm_create()`
  - This is a collective operation. All the processes in `comm` must call this function, regardless whether, processes join new communicator or not.

- `MPI_Group_incl()  &  MPI_Comm_group()`
  - These are local operations. No communication among processes are involved.

MPI_Comm_split() to form communicators

- `int MPI_Comm_split( MPI_Comm comm, int color, int key, MPI_Comm *newcomm );`
  - Creates new communicators based on colors and keys. This function partitions the group associated with `comm` into disjoint subgroups, one for each value of color. Each subgroup contains all processes of the same color. Within each subgroup, the processes are ranked in the order defined by the value of the argument `key`, with ties broken according to their rank in the old group.
  - `comm`: old communicator
  - `color`: control of subset assignment (nonnegative integer). Processes with the same color are in the same new communicator
  - `key`: control of rank assignment
  - `newcomm`: new communicator

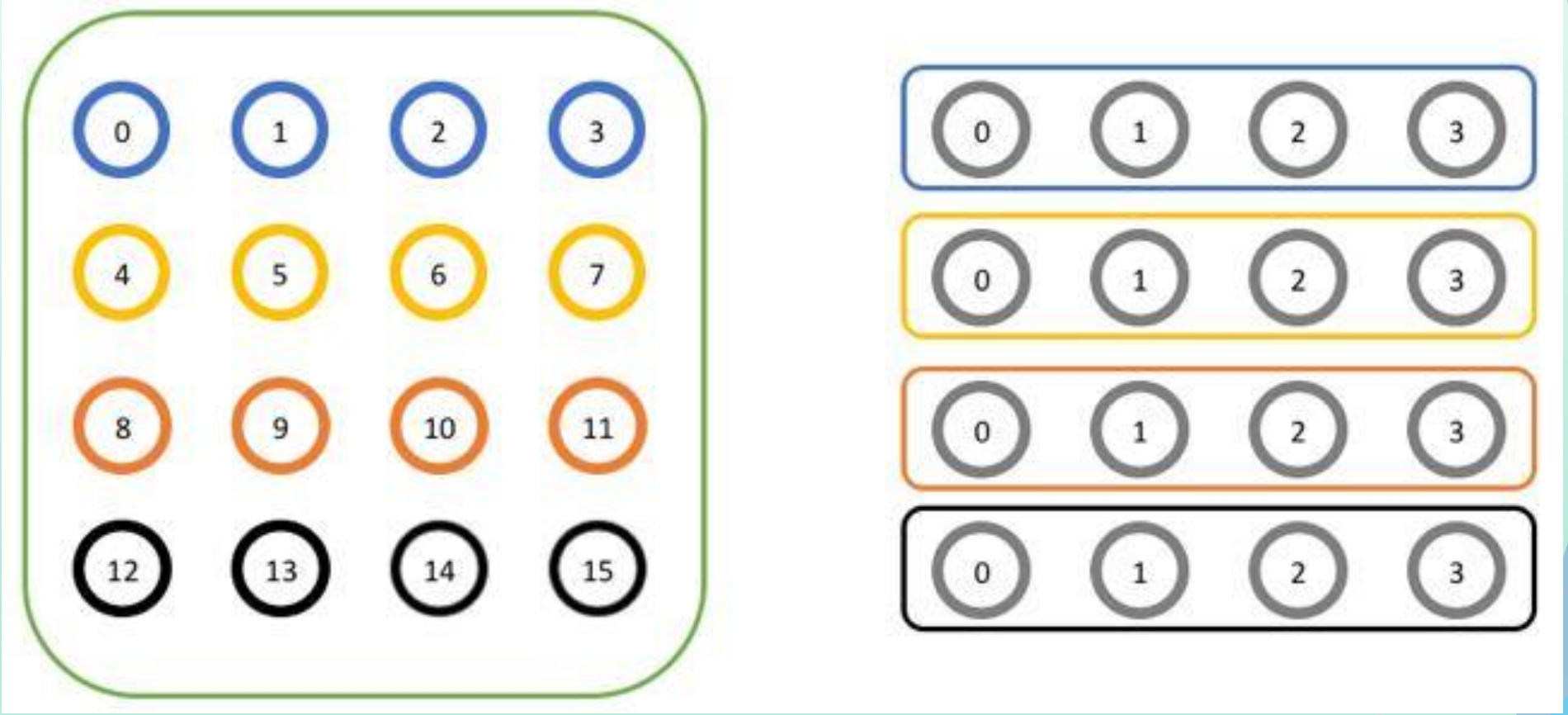
- This is a collective call. Each process can provide its own color and key.

Example: Split processes with odd and even ranks into 2 communicators

```
int main(int argc, char *argv[]) {
 int myid, numprocs;
 int color, broad_val, new_id, new_nodes;
 MPI_Comm New_Comm;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &myid);
 color = myid % 2;
 MPI_Comm_split(MPI_COMM_WORLD, color, myid, &New_Comm);
 MPI_Comm_rank(New_Comm, &new_id);
 MPI_Comm_size( New_Comm, &new_nodes);

 if(new_id == 0) broad_val = color;
 MPI_Bcast(&broad_val, 1, MPI_INT, 0, New_Comm);
 printf("Old proc[%d] has new rank %d received value %d\n", myid, new_id, broad_val);
 MPI_Finalize();
}
```

Example1: Split a large communicator into smaller ones



Example1: Code

```
// Get the rank and size in the original communicator
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int color = world_rank / 4; // Determine color based on row

// Split the communicator based on the color and use the original rank for ordering
MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm); // 4 new communicators constructed

int row_rank, row_size;
MPI_Comm_rank(row_comm, &row_rank);
MPI_Comm_size(row_comm, &row_size);

printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n", world_rank, world_size, row_rank,
row_size);

MPI_Comm_free(&row_comm);
```

Example1: Output

WORLD RANK/SIZE: 0/16	ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 1/16	ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 2/16	ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 3/16	ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 4/16	ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 5/16	ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 6/16	ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 7/16	ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 8/16	ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 9/16	ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 10/16	ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 11/16	ROW RANK/SIZE: 3/4
WORLD RANK/SIZE: 12/16	ROW RANK/SIZE: 0/4
WORLD RANK/SIZE: 13/16	ROW RANK/SIZE: 1/4
WORLD RANK/SIZE: 14/16	ROW RANK/SIZE: 2/4
WORLD RANK/SIZE: 15/16	ROW RANK/SIZE: 3/4

MPI_Comm_create_group routine

- `int MPI_Comm_create_group(MPI_Comm comm, MPI_Group group, int tag, MPI_Comm *newcomm)`

- Creates a new communicator
  - `comm`: communicator
  - `group`: group, which is a subset of the group of `comm`
  - `tag`: safe tag unused by other communication
  - `newcomm`: new communicator

Example2 - Create a new group whose ranks are prime: Code

```
// Get the rank and size in the original communicator
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

// Get the group of processes in MPI_COMM_WORLD
MPI_Group world_group;   MPI_Comm_group(MPI_COMM_WORLD, &world_group);

int n = 7;   const int ranks[7] = {1, 2, 3, 5, 7, 11, 13};

// Construct a group containing all of the prime ranks in world_group
MPI_Group prime_group;   MPI_Group_incl(world_group, 7, ranks, &prime_group);

// Create a new communicator based on the group
MPI_Comm prime_comm;
MPI_Comm_create_group(MPI_COMM_WORLD, prime_group, 0, &prime_comm);
```

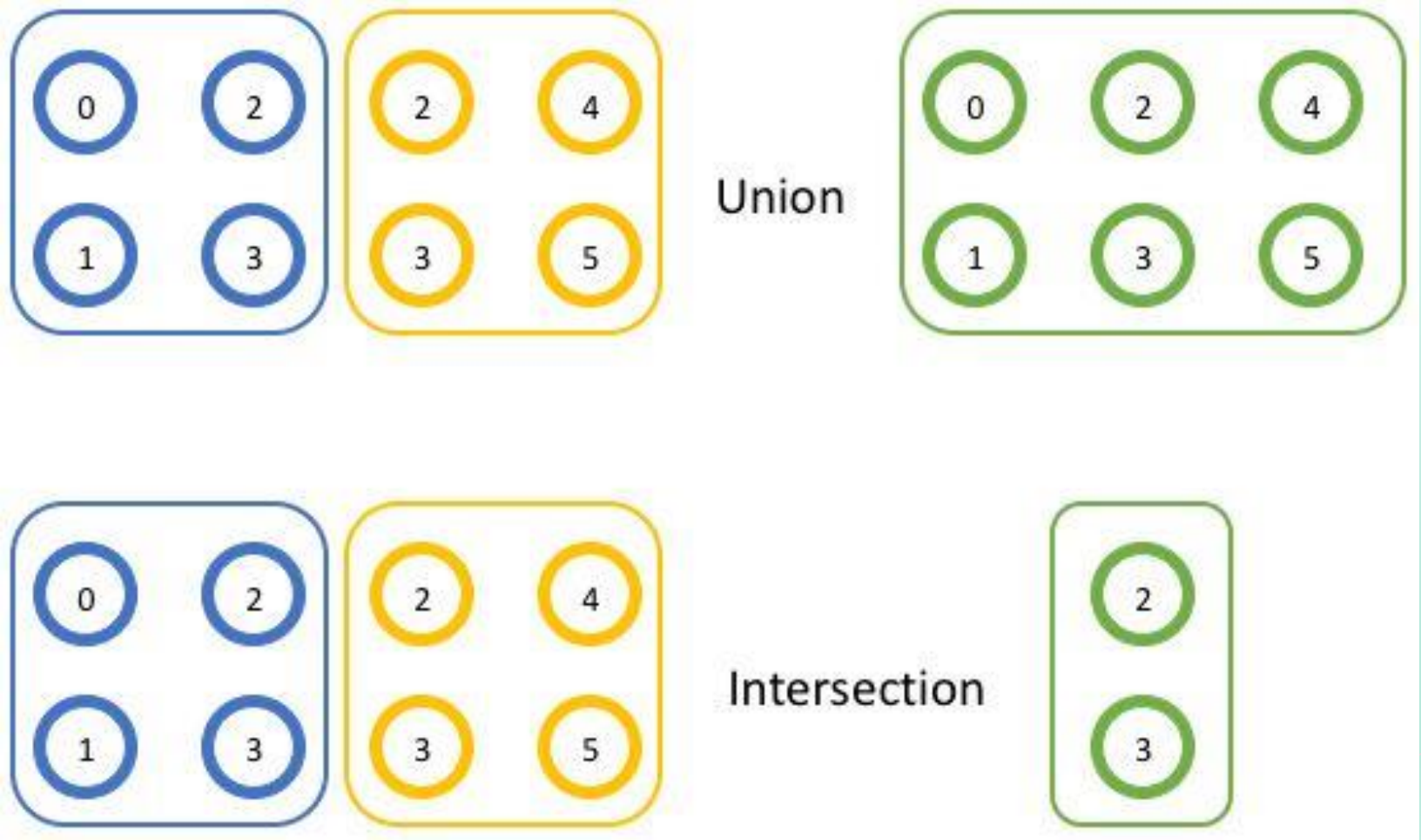
Example2: Code (cont.)

```
int prime_rank = -1, prime_size = -1;
// If this rank isn't in the new communicator, it will be
// MPI_COMM_NULL. Using MPI_COMM_NULL for MPI_Comm_rank or
// MPI_Comm_size is erroneous
if (MPI_COMM_NULL != prime_comm) {
     MPI_Comm_rank(prime_comm, &prime_rank);
     MPI_Comm_size(prime_comm, &prime_size);
}

printf("WORLD RANK/SIZE: %d/%d \t PRIME RANK/SIZE: %d/%d\n",
     world_rank, world_size, prime_rank, prime_size);

MPI_Group_free(&world_group);
MPI_Group_free(&prime_group);
MPI_Comm_free(&prime_comm);
```

Union and intersection of groups



Further MPI routines

- `int MPI_Group_union( MPI_Group group1, MPI_Group group2, MPI_Group* newgroup)`
  - Constructs `newgroup` as the union of groups `group1` and `group2`

- `int MPI_Group_intersection( MPI_Group group1, MPI_Group group2, MPI_Group* newgroup)`
  - Constructs `newgroup` as the intersection of groups `group1` and `group2`