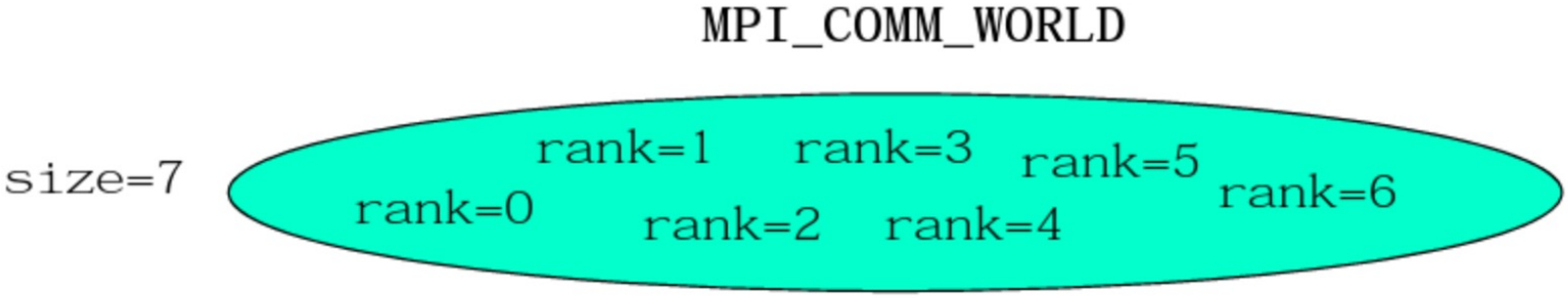
PRACE Course: Intermediate MPI

9-11 November 2022

MPI Groups and Communicators

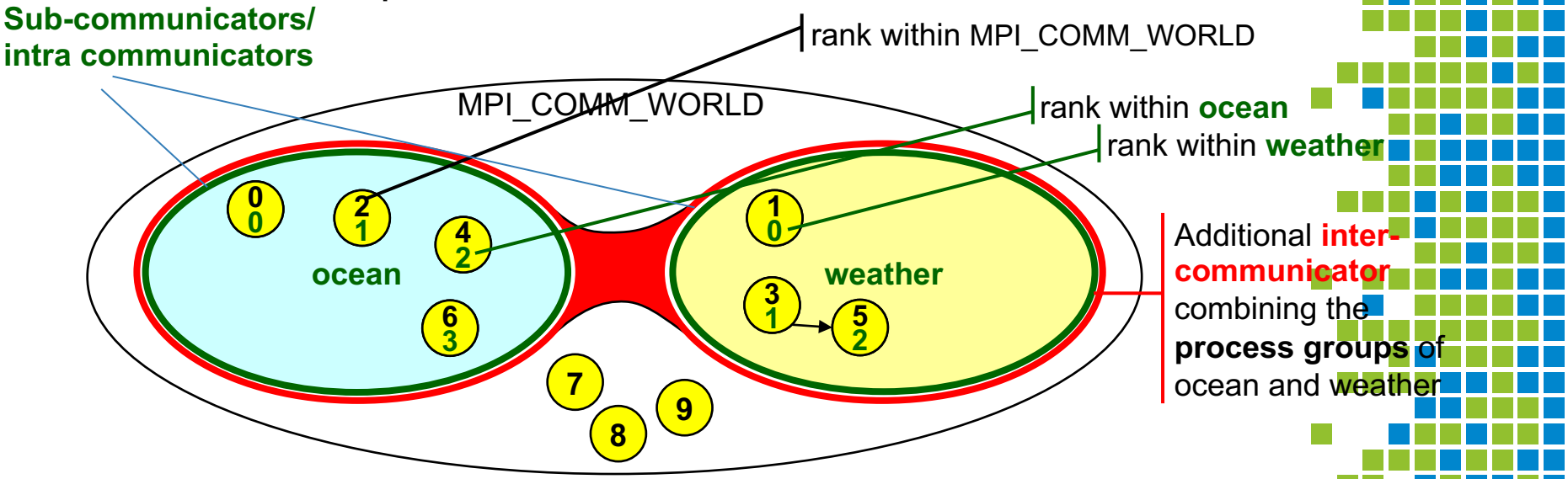
Communicators

- A communicator is a group of processors that can engage in communication
  - Source and destination of a message is identified by the process rank within the communicator
- It contains a context and a group.
- Predefined communicators: `MPI_COMM_WORLD`, `MPI_COMM_NULL`, `MPI_COMM_SELF`.



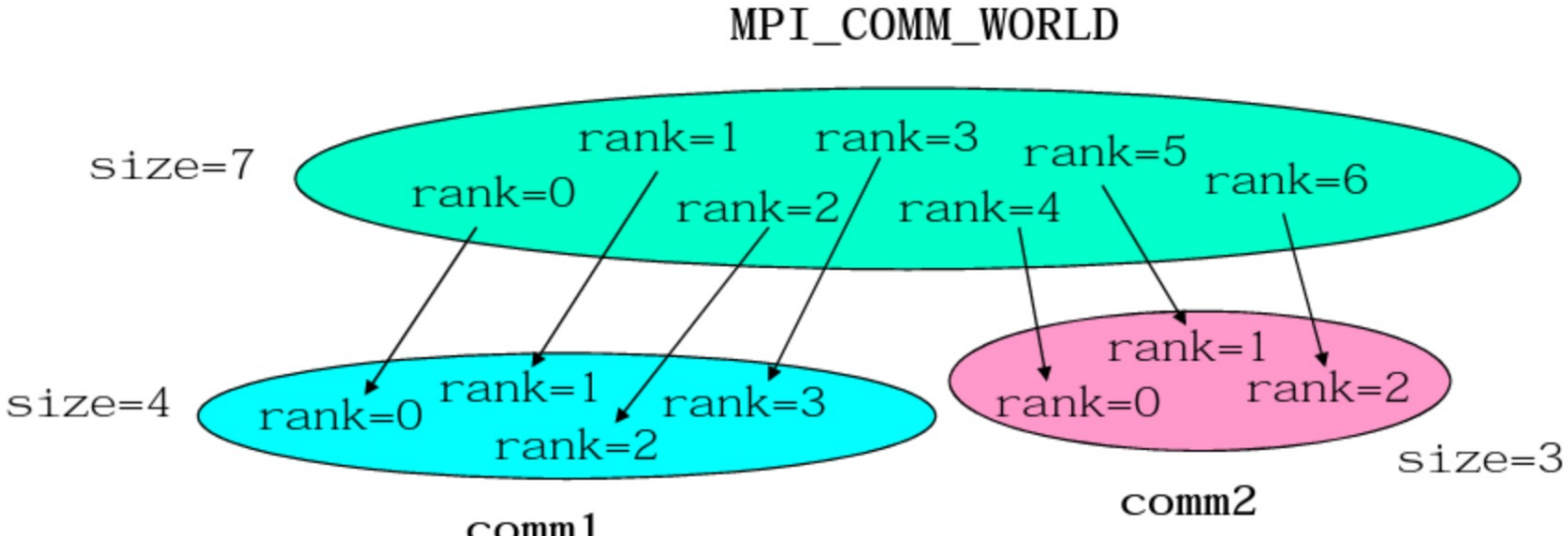
Types of Communicators

- Inter communicators: contains two groups
- Intra communicators: contains a single group
    1. Cartesian communicators
    2. Graph communicators



Sub-Communicators

- Processes can be divided into sub-groups of processes, or sub-communicators
  - Task level parallelism with process groups performing separate duties together
  - Scalability (avoids unnecessary)



Motivations

- Need to create sets of processes
  - For programming convenience
  - Make use of collectives routines

- Need to map the abstract topology onto the natural topology of the problem domain
  - For programming convenience
  - For performance

Communicator Management

- At the start of, all its processes belong to the communicator `MPI_COMM_WORLD`. Each has its unique rank.
- Partition processes into subgroups forming separate communicators.
  - Each including the processes belonging a particular task.
- Creating/freeing a communicator is a collective operation, i.e., all processes of the original communicator have to call the function with the same arguments.
- A communicator is of type `MPI_Comm`.
- Routines: `MPI_Comm_size`, `MPI_Comm_rank`, `MPI_Comm_compare`, `MPI_Comm_dup`, `MPI_Comm_create`, `MPI_Comm_free`, `MPI_Comm_split`

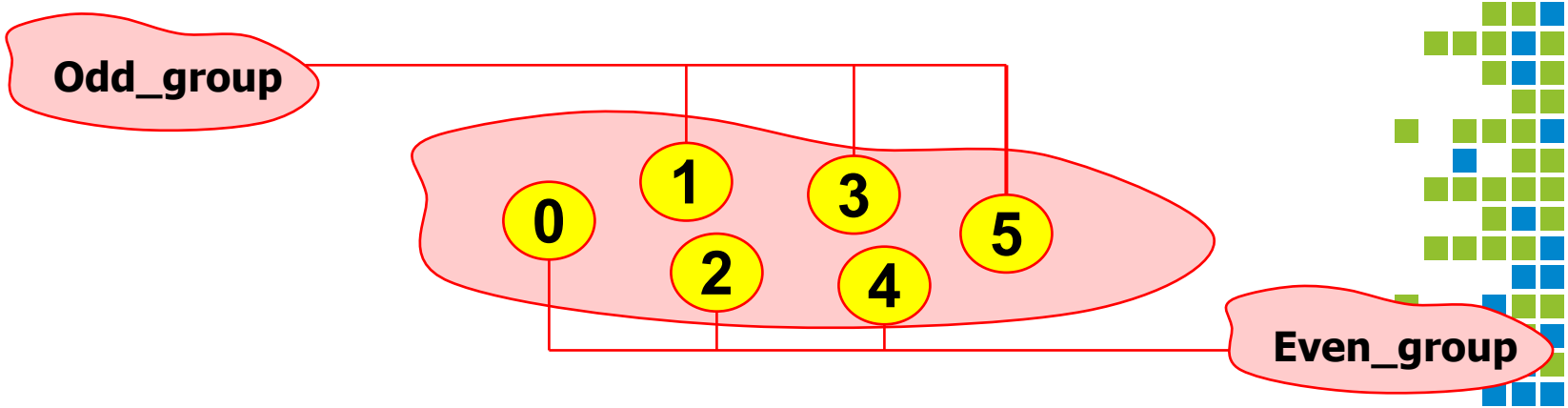
Creating new Communicators

- Methods to create new communicators:

1. duplicate an existing communicator
2. creating subgroups of the original communicator
3. splitting the original communicator into n-parts
4. re-ordering of processes based on topology information
5. spawn new processes
6. connect two applications and merge their communicators

Method 1: Working with Groups

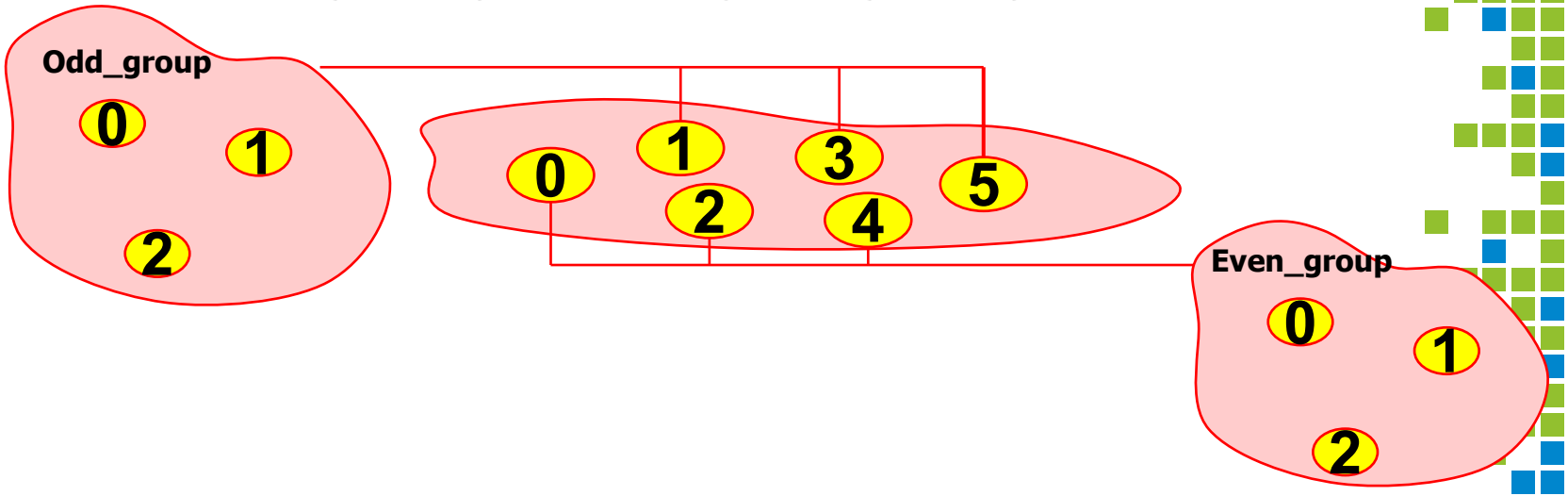
- Select processes ranks to create groups
- Associate to these groups new communicators
- Use these new communicators as usual
- `MPI_Comm_group`(comm, group) returns in group the group associated to the communicator comm
- Communicators are dynamic and can be created/destroyed during runtime
- A process can belong simultaneously to several communicators, and in each, has a unique ID



Working with Groups

- Rank numbers restart from 0 in the new communicators
  - Each group is an ordered set of process identifiers
  - Each process in a group is associated with a rank

- WORLD {1, 3, 5} → Odd_group {0, 1, 2}
- WORLD {0, 2, 4} → Even_group {0, 1, 2}



MPI_Comm_group

- Extract handle of global group from `MPI_COMM_WORLD` using `MPI_Comm_group()`
- Form new group as a subset of global group using `MPI_Group_incl()`/ `MPI_Group_range_incl()` or `MPI_Group_excl()`/ `MPI_Group_range_excl()`
- Create new communicator for new group using `MPI_Comm_create()`, `MPI_Comm_create_group()`
- Determine new rank in new communicator using `MPI_Comm_rank()`
- Conduct communications using any MPI message passing routine
- When finished, free up new communicator and group (optional) using `MPI_Comm_free()` and `MPI_Group_free()`

MPI_Comm_group

1. communicator → `MPI_Comm_group()` → group handle
2. group handle → `MPI_Group_incl()` (mapping_array) → reordered group
3. Communicator + reordered group → `MPI_Comm_create()` → reordered comm

.

- **Example: (odd/even splitting of processes)**

```
Odd_ranks={1, 3, 5}, Even_ranks={0, 2, 4};
MPI_Comm_group(MPI_COMM_WORLD, newgroup);
MPI_Group_incl(newgroup, 3, Odd_ranks, &Odd_group);
MPI_Group_incl(newgroup, 3, Even_ranks, &Even_group);
int MPI_Comm_create(MPI_COMM_WORLD, Odd_group, Odd_Comm);
int MPI_Comm_create(MPI_COMM_WORLD, Even_group, Even_Comm);
```

- an `MPI_Group` is the object describing the list of processes forming a logical entity: `MPI_Group_size()`, `MPI_Group_rank()`

Example: MPI_Comm_group

```
int uniSize, ierror, odd_ranks[3]={1, 3, 5}, newRank, oddRank;
  MPI_Comm odd_comm;
  MPI_Group new_group, odd_group;

  ierror=MPI_Init(&argc,&argv);
  ierror=MPI_Comm_size(MPI_COMM_WORLD,&uniSize);

  if(uniSize==6){
    ierror=MPI_Comm_group(MPI_COMM_WORLD, &new_group);
    ierror=MPI_Group_rank(new_group, &newRank);
    ierror=MPI_Group_incl(new_group, uniSize/2, odd_ranks, &odd_group);
    ierror=MPI_Comm_create(MPI_COMM_WORLD, odd_group, &odd_comm);
    ierror=MPI_Group_rank(odd_group, &oddRank);

    if(oddRank != MPI_UNDEFINED){
      printf("I am process %d in new group and %d in the odd group.\n", newRank, oddRank);
    }
    else{
      printf("I am process %d in new group but I am not part of the odd group.\n", newRank);
    }
  }

  ierror=MPI_Finalize();
```

Run with six processes

Method 2: MPI_Comm_split - 1

- C:

```
int MPI_Comm_split (MPI_Comm comm, int color, int
 key, MPI_Comm *newcomm)
```

- Fortran:

```
MPI_COMM_SPLIT (comm, color, key, newcomm, ierror)
   TYPE(MPI_Comm) :: comm, newcomm
   INTEGER :: color, key; INTEGER, OPTIONAL :: ierror
```

- Each subgroup contains all processes of the same color. Within each subgroup, the processes are ranked in the order defined by the value of key. A new communicator is created for each subgroup and returned in newcomm.
- If color = `MPI_UNDEFINED`, a process does not belong to any of the new communicators. newcomm returns `MPI_COMM_NULL`.

MPI_Comm_split

- Partition comm into sub-communicators
  - all processes having the same color will be in the same subcommunicator
  - order processes with the same color according to the key value
- **Example: (odd/even splitting of processes)**

```
if (myid%2 == 0) {color = 1;}
else {color = 2;}
MPI_Comm_split(MPI_COMM_WORLD, color, myid, &subcomm);
MPI_Comm_rank(subcomm, &mysubid);
```

- `MPI_COMM_CREATE(comm, group, newcomm)` is equivalent to `MPI_COMM_SPLIT(comm, color, key, newcomm)`

- a process
  - can just be part of one of the generated communicators
  - can not see the other communicators
  - can not see how many communicators have been created

Example: MPI_Comm_split

```
  int myRank, uniSize, ierror;
  int color, key, newRank;
  MPI_Comm newcomm;

  ierror=MPI_Init(&argc,&argv);
  ierror=MPI_Comm_rank(MPI_COMM_WORLD,&myRank);
  ierror=MPI_Comm_size(MPI_COMM_WORLD,&uniSize);

  if(myRank % 2 == 0){
      color=1;
      key=myRank;}
  else{
      color=2;
      key=uniSize-myRank;}

  ierror=MPI_Comm_split(MPI_COMM_WORLD, color, key, &newcomm);
  ierror=MPI_Comm_rank(newcomm, &newRank);
  ierror=MPI_Allreduce (&newRank, &sum, 1, MPI_INT, MPI_SUM, newcomm);

  printf("I am process %d in MPI_COMM_WORLD and %d in the new communicator.\n", myRank,
newRank);
  if(newRank==0){
    printf("Color and Sum of ranks in the new communicator: %d, %d.\n", color, sum);
  }
```

MPI_Info Object

- Used to pass hints to the implementation
  - Consist of (key, value) pairs, key and value being strings
  - The maximum key size is `MPI_MAX_INFO_KEY`
  - The maximum value size is `MPI_MAX_INFO_VAL` (implementation dependent)
  - handle of type MPI_Info in C and TYPE(MPI_Info) in Fortran
  - Portable programs may use `MPI_INFO_NULL` as the info argument
  - Allows applications to pass environment-specific information
  - To improve performance or resource utilization
  - Applications: Parallel I/O, RMA, Dynamic processes.

MPI_Info Functions

- `MPI_Info_create()`, `MPI_Info_free()`, `MPI_Info_set()`, `MPI_Info_delete()`, `MPI_Info_get()`, `MPI_Info_get_nkeys()`, `MPI_Info_get_valuelen()`, `MPI_Info_get()`

- For Communicators:
- `MPI_Comm_set_info()`, `MPI_Comm_get_info()`, `MPI_Comm_dup_with_info()`, `MPI_Comm_idup_with_info()`