MPI Advance - Multi-program with port connection

강지훈 (jhkang@kisti.re.kr)

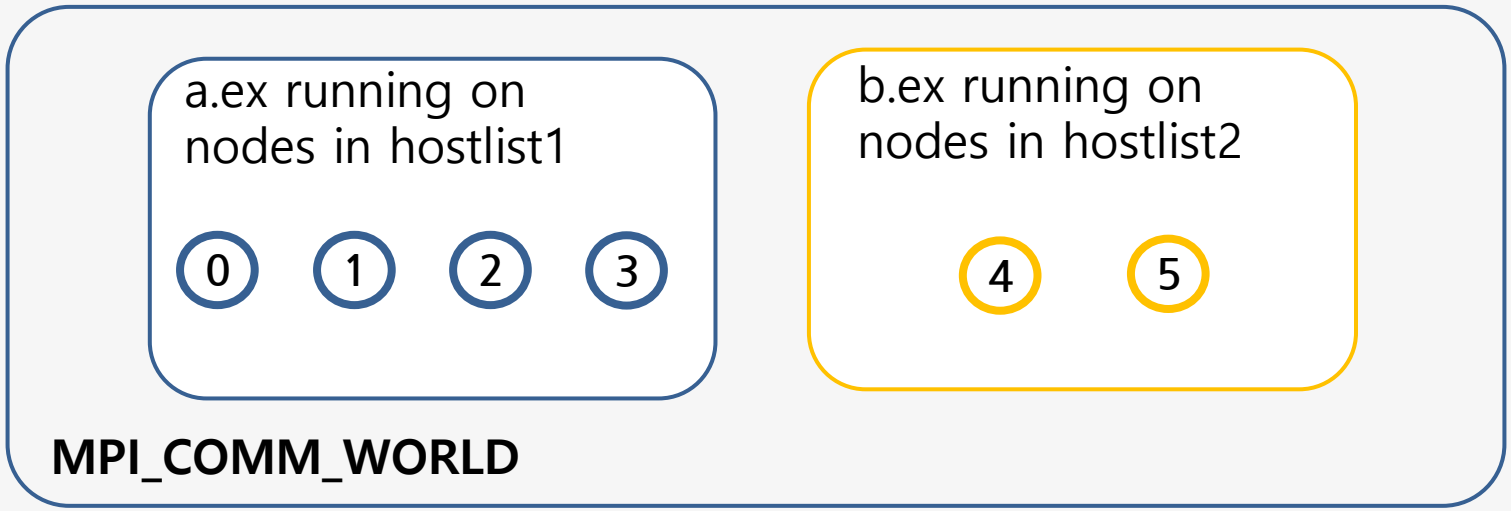
Supercomputing Application Center

Division of National Supercomputing

MPI multiple program multiple data

- MPI supports multiple program execution with the mpirun command
- All program shares the world communicator, MPI_COMM_WORLD
- MPMD is quite similar to SPMD in terms of communication, except that world communicator of all programs and local communicators of each program are simultaneously used.

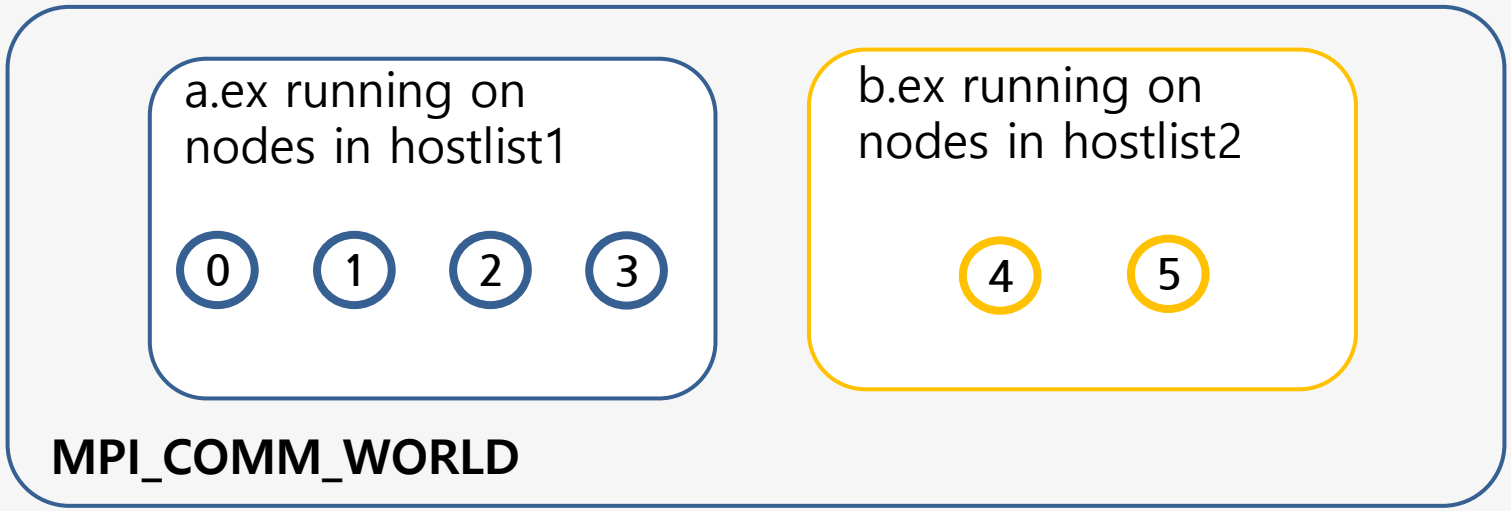
mpirun –hostfile hostlist1 –np 4 a.ex : -hostfile hostlist2 –np 2 b.ex



MPMD local communicator

- If we want to do communication only in each program, we should create local communicator from MPI_COMM_WORLD, using communicator split or groups.
- When we use MPI_COMM_SPLIT, we can keep a same communicator name for each local communicator, or distinguish them using MPI_COMM_DUP.
- We can use MPI_group instead of MPI_COMM_SPLIT

mpirun –hostfile hostlist1 –np 4 a.ex : -hostfile hostlist2 –np 2 b.ex

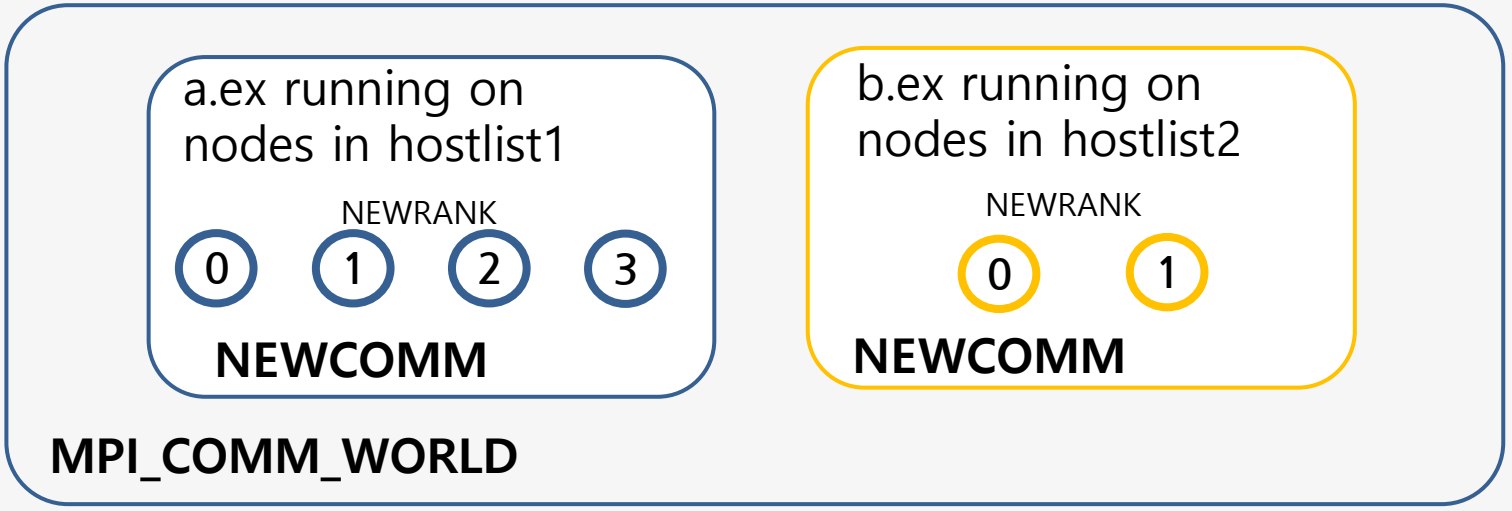


Creating local communicators

- Using MPI_COMM_SPLIT

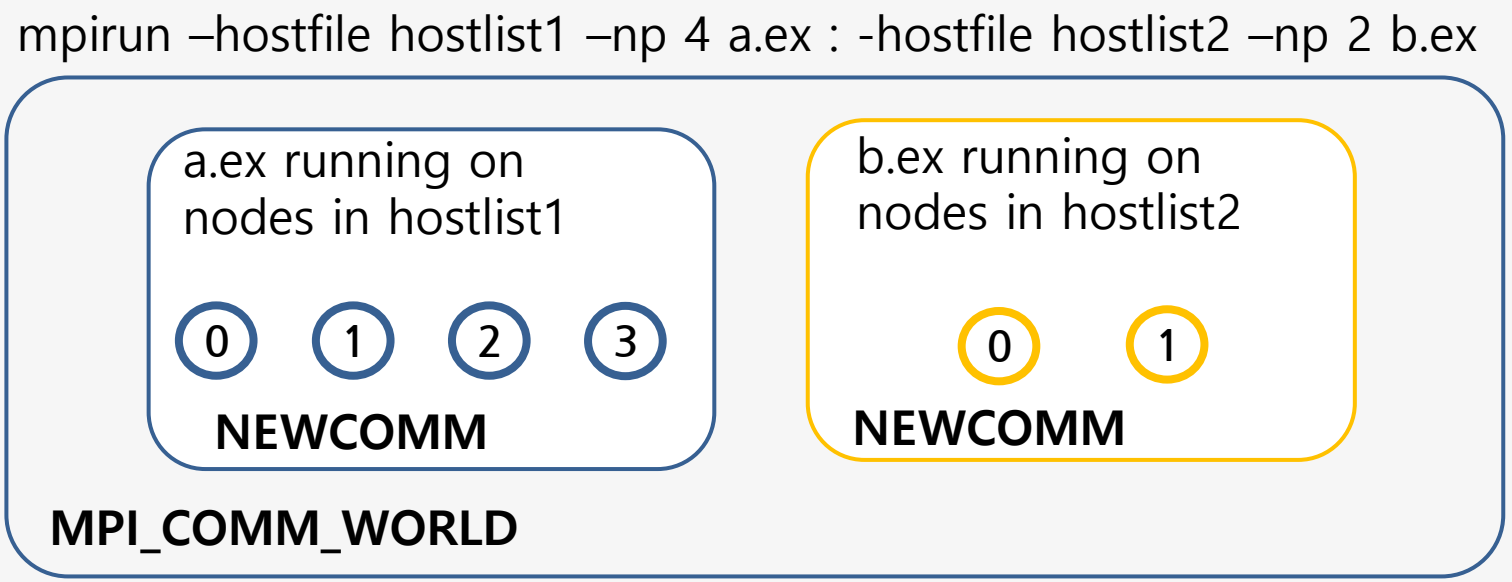
```
Call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
…
If(myrank.LT.4) then
   color = 0
Else
   color = 1
Endif
Call MPI_COMM_SPLIT(MPI_COMM_WORLD, color, myrank, newcomm, ierr)
call MPI_COMM_RANK(newcomm, newrank,ierr)
```

mpirun –hostfile hostlist1 –np 4 a.ex : -hostfile hostlist2 –np 2 b.ex



Communication details

- If we want do communication in each local communicator, we use newrank and newcomm (newsize if required).
- If we want do communication in world communicator, we use world rank (myrank) and MPI_COMM_WORLD.
- We should know **the relation between local rank and world rank.**



World rank and local rank relation

- 1. Use hard coding – easiest way
  - The rank of roota/rootb is hard-coded or assigned though input argument or input file.

```
roota = 0
rootb = 4
…
If(myrank.eq.roota) then
   call MPI_RECV(data,1, MPI_INTEGER, rootb, tag1, MPI_COMM_WORLD, status, ierr)
Else if(myrank.eq.rootb) then
   call MPI_SEND(data,1 MPI_INTEGER, roota, tag1, MPI_COMM_WORLD, status, ierr)
```

World rank and local rank relation

- 2. Use rank translator

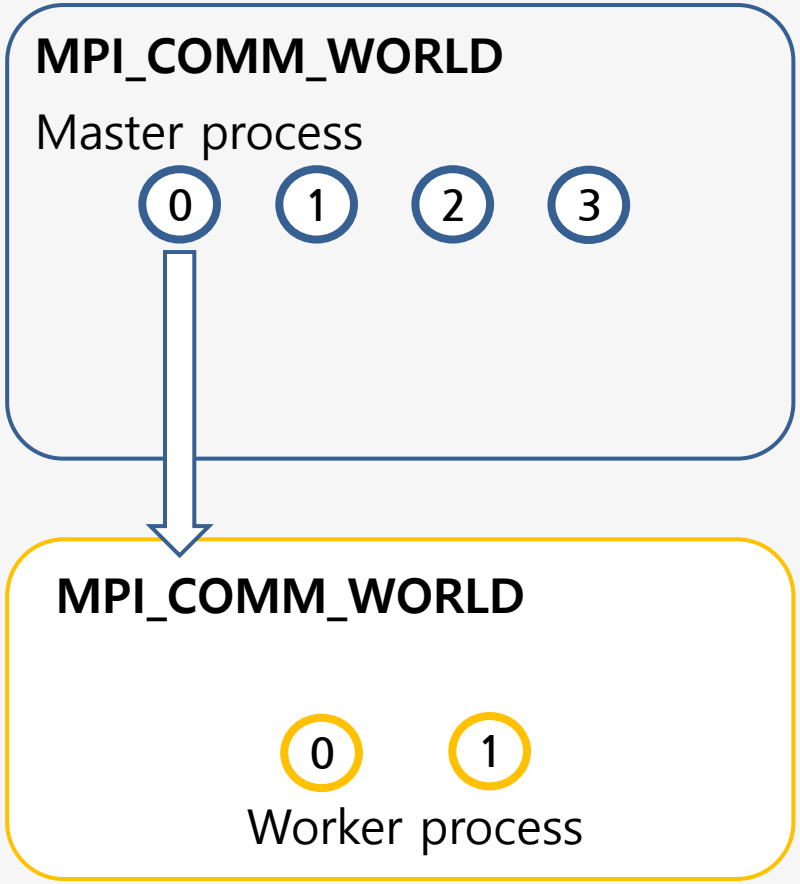
```
Call MPI_Comm_group(MPI_COMM_WORLD, world_group, ierror)
Call MPI_Comm_group(newcomm, newgroup, ierror)

If(color.eq.0) then
   Call MPI_Group_translate_ranks(newcomm, 1, 0, MPI_COMM_WORLD, roota, ierr)
Endif
Call MPI_Bcast(roota, 1, MPI_INTEGER, roota, MPI_COMM_WORLD, ierr

If(color.eq.1) then
   Call MPI_Group_translate_ranks(newcomm, 1, 0, MPI_COMM_WORLD, roota, ierr)
Endif
Call MPI_Bcast(rootb, 1, MPI_INTEGER, rootb, MPI_COMM_WORLD, ierr)

…
If(myrank.eq.roota) then
   call MPI_RECV(data,1, MPI_INTEGER, rootb, tag1, MPI_COMM_WORLD, status, ierr)
Else if(myrank.eq.rootb) then
   call MPI_SEND(data,1 MPI_INTEGER, roota, tag1, MPI_COMM_WORLD, status, ierr)
```

Code example - MPMD

- Example in mpmd folder
- Make and run with "mpirun –np 4 ./pi.ex : -np 2 ./mm.ex 20"

MPI Spawn

- Spawning worker process with different MPI_COMM_WORLD.



MPI_COMM_SPAWN

- Called by master process

```
MPI_COMM_SPAWN(command, argv, maxprocs, info, root, comm, intercomm,
                                array_of_errcodes, ierror)
    CHARACTER(LEN=*), INTENT(IN) :: command, argv(*)
    INTEGER, INTENT(IN) :: maxprocs, root
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Comm), INTENT(OUT) :: intercomm
    INTEGER :: array_of_errcodes(*)
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

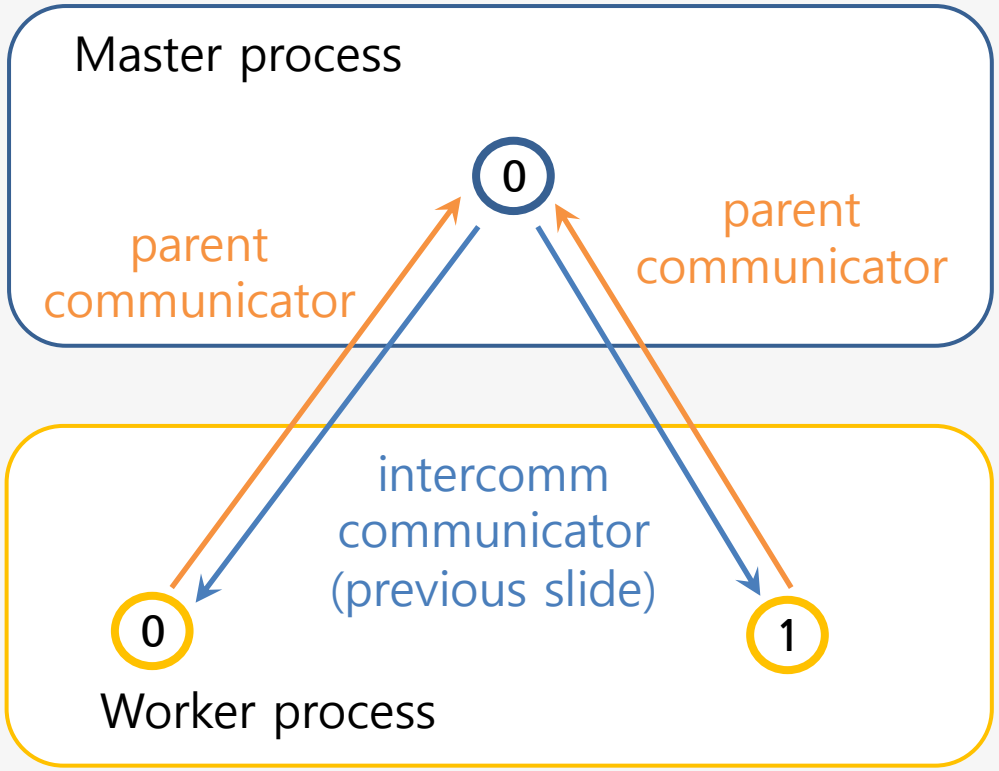
- Spawning **command** program
  - With **ARGV** command argument
  - With **MAXPROCS** processes
  - With MPI_Info **INFO**
  - By **ROOT** process
  - From **COMM** communicator (Usually MPI_COMM_WORLD)
  - With **INTERCOMM** communicator to connect COMM communicator and worker communicator

MPI_COMM_GET_PARENT

- Called by worker process

```
MPI_Comm_get_parent(parent, ierror)
    TYPE(MPI_Comm), INTENT(OUT) :: parent
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

- Create parent communicator which connect current process and parent process



Communication between master and worker

- Master

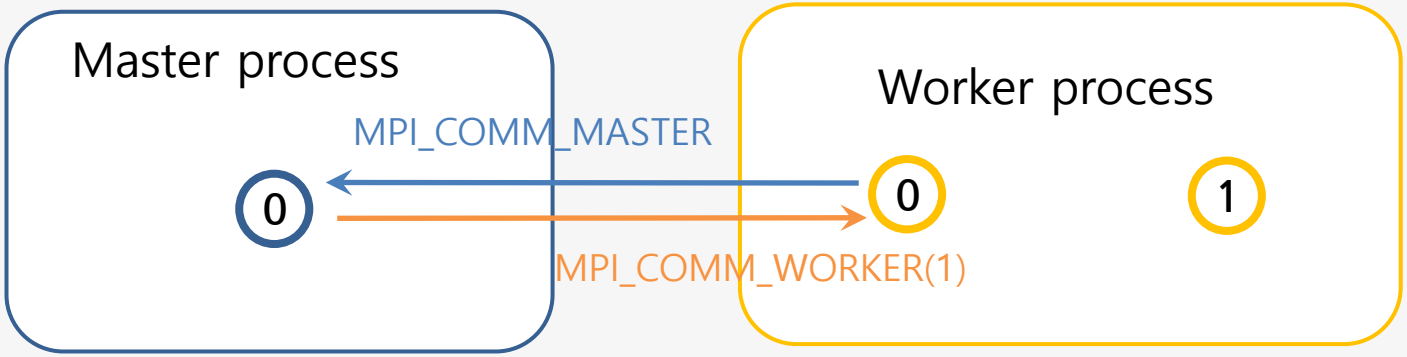
```
call MPI_Comm_spawn(worker(1), argv, worker_np(1), info(1), 0, MPI_COMM_WORLD,
                    MPI_COMM_WORKER(1), MPI_ERRCODES_IGNORE, ierr)
call MPI_RECV(pi, 1, MPI_REAL8, 0, 1, MPI_COMM_WORKER(1), MPI_STATUS_IGNORE, ierr)
```

  – Receive data from process 0 of worker communicator

- Worker

```
call MPI_Comm_get_parent(MPI_COMM_MASTER,ierr)
if(myrank == 0) then
    call MPI_SEND(pi, 1, MPI_REAL8, 0, 1, MPI_COMM_MASTER, ierr)
endif
```

  – Send data to process 0 of master communicator



MPI_Info

- MPI_Info
  - It stores an unordered set of (key, value) pairs (both key and value are strings).
  - A key can have only one value.
  - MPI reserves several keys and requires that if an implementation uses a reserved key, it must provide the specified functionality.
  - An implementation is not required to support these keys and may support any others not reserved by MPI.
  - **Keys depends on MPI implementation (OpenMPI, MPICH, etc) : keys should be modified according to MPI implementation**
  - Created by **MPI_Info_create(MPI_Info info, integer ierror)**
  - Modified by **MPI_Info_set(MPI_Info info, character(len=*) key, character(len=*) key)**

- Example

```
call MPI_Info_create(info(1),ierr)
call MPI_Info_set(info(1),"add-host","jihoon",ierr) ! Add host information to info object
```

Code example - Spawn

- Example in spawn folder
- Make and run with "mpirun –np 1 ./master.ex"

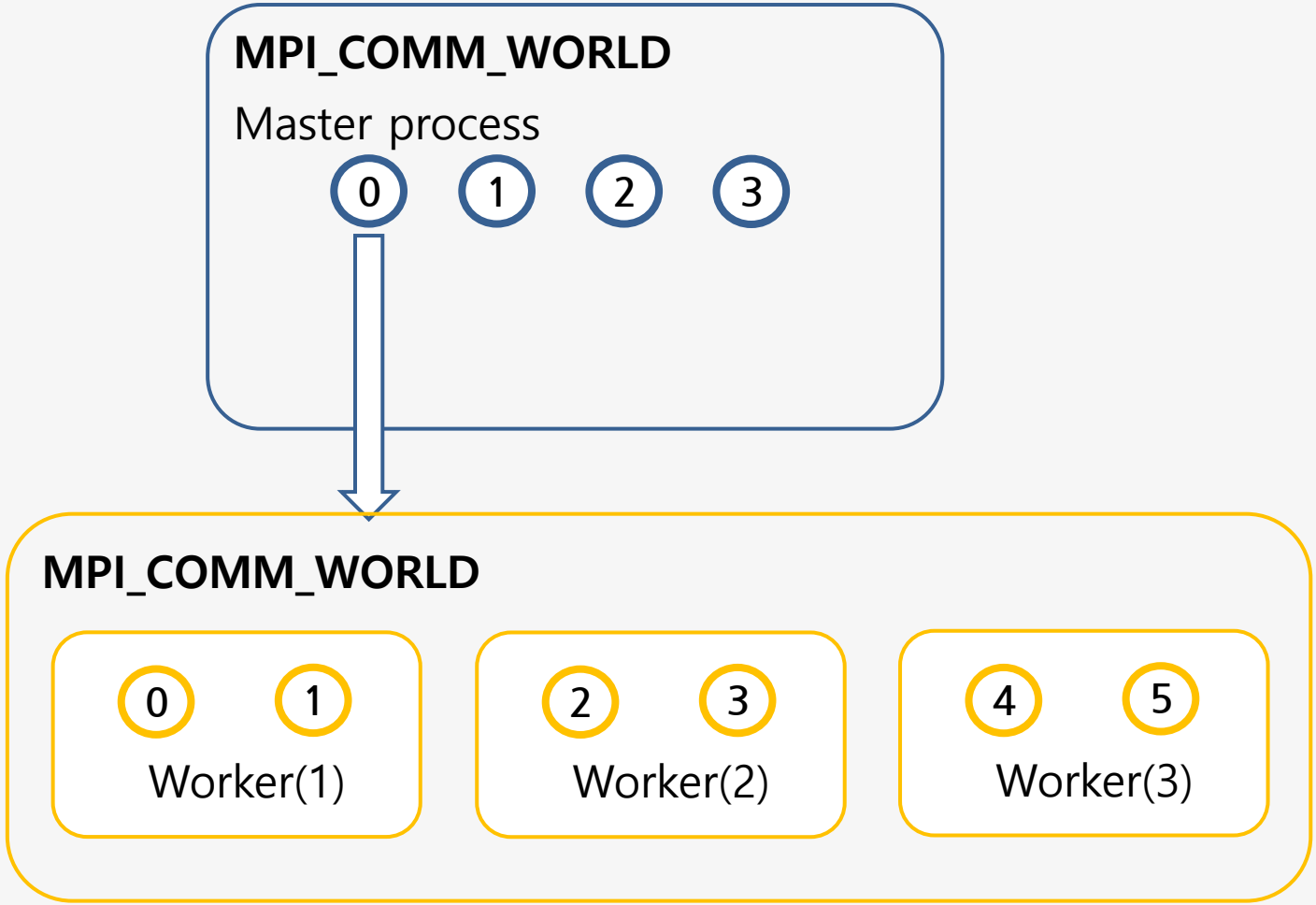
MPI_Comm_spawn_multiple

- Called by master process
- Create multiple workers which are in same MPI_WORLD_COMMUNICATOR

```
MPI_Comm_spawn_multiple(count, array_of_commands, array_of_argv,
    array_of_maxprocs, array_of_info, root, comm, intercomm,
    array_of_errcodes, ierror)
    INTEGER, INTENT(IN) :: count, array_of_maxprocs(*), root
    CHARACTER(LEN=*), INTENT(IN) :: array_of_commands(*)
    CHARACTER(LEN=*), INTENT(IN) :: array_of_argv(count, *)
    TYPE(MPI_Info), INTENT(IN) :: array_of_info(*)
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Comm), INTENT(OUT) :: intercomm
    INTEGER :: array_of_errcodes(*)
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

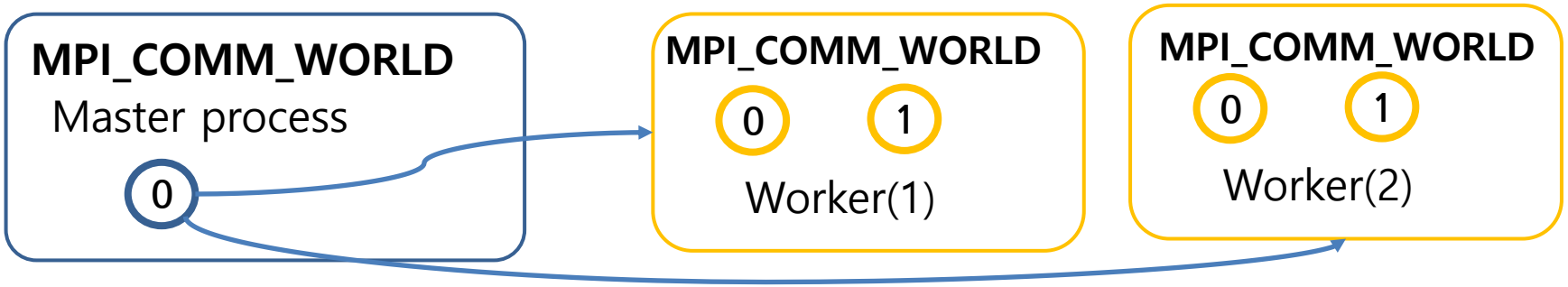
MPI_Comm_spawn_multiple

- All workers are in common MPI_COMM_WORLD with continuous ranks

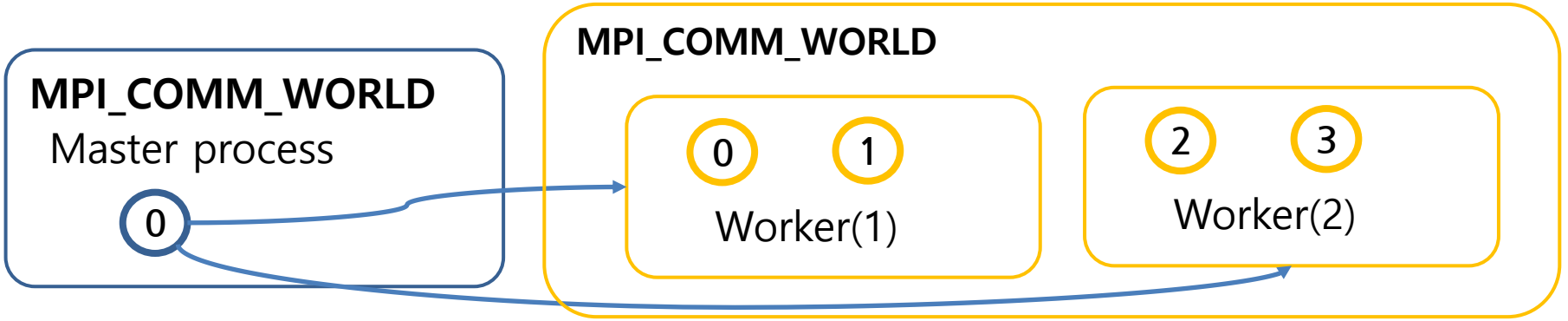


MPI_COMM_SPAWN vs. MPI_COMM_SPAWN_MULTIPLE

```
call MPI_Comm_spawn(worker(1), argv, worker_np(1), info(1), 0, MPI_COMM_WORLD,
                    MPI_COMM_WORKER(1), MPI_ERRCODES_IGNORE, ierr)
call MPI_Comm_spawn(worker(2), argv, worker_np(2), info(2), 0, MPI_COMM_WORLD,
                    MPI_COMM_WORKER(2), MPI_ERRCODES_IGNORE, ierr)
```




```
call MPI_Comm_spawn_multiple(2,worker,array_of_argv, worker_np, info, 0,
                    MPI_COMM_WORLD,MPI_COMM_WORKER,MPI_ERRCODES_IGNORE,ierr)
```



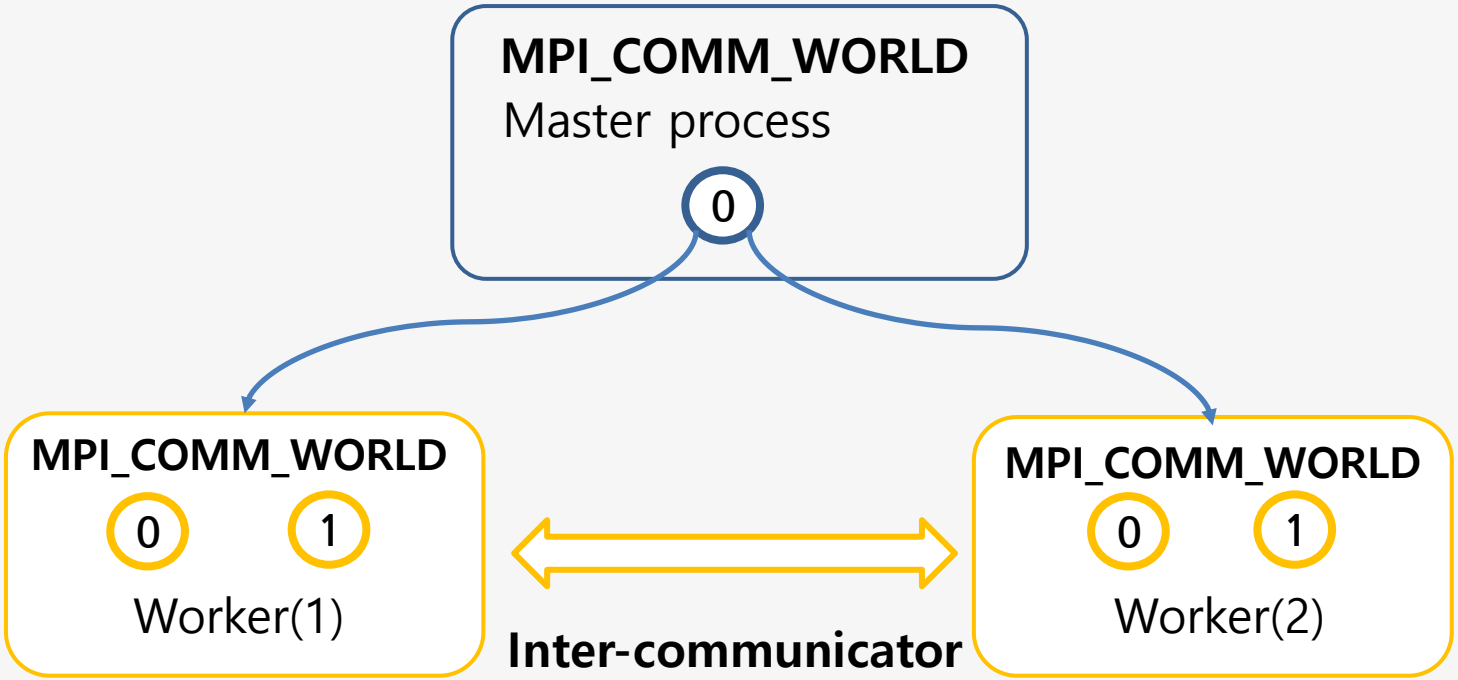
Code example - Multiple spawn

- Example in spawn_multiple folder
- Make and run with "mpirun –np 1 ./master.ex"

Create inter-communicator

- Inter-communicator between spawned processes with different MPI_COMM_WORLD

```
call MPI_Comm_spawn(worker(1), argv, worker_np(1), info(1), 0, MPI_COMM_WORLD,
                    MPI_COMM_WORKER(1), MPI_ERRCODES_IGNORE, ierr)
call MPI_Comm_spawn(worker(2), argv, worker_np(2), info(2), 0, MPI_COMM_WORLD,
                    MPI_COMM_WORKER(2), MPI_ERRCODES_IGNORE, ierr)
```



Work-around solution using MPI port connection

- Master process
  - Open port
  - Send portname to worker rank
  - Accept port connection

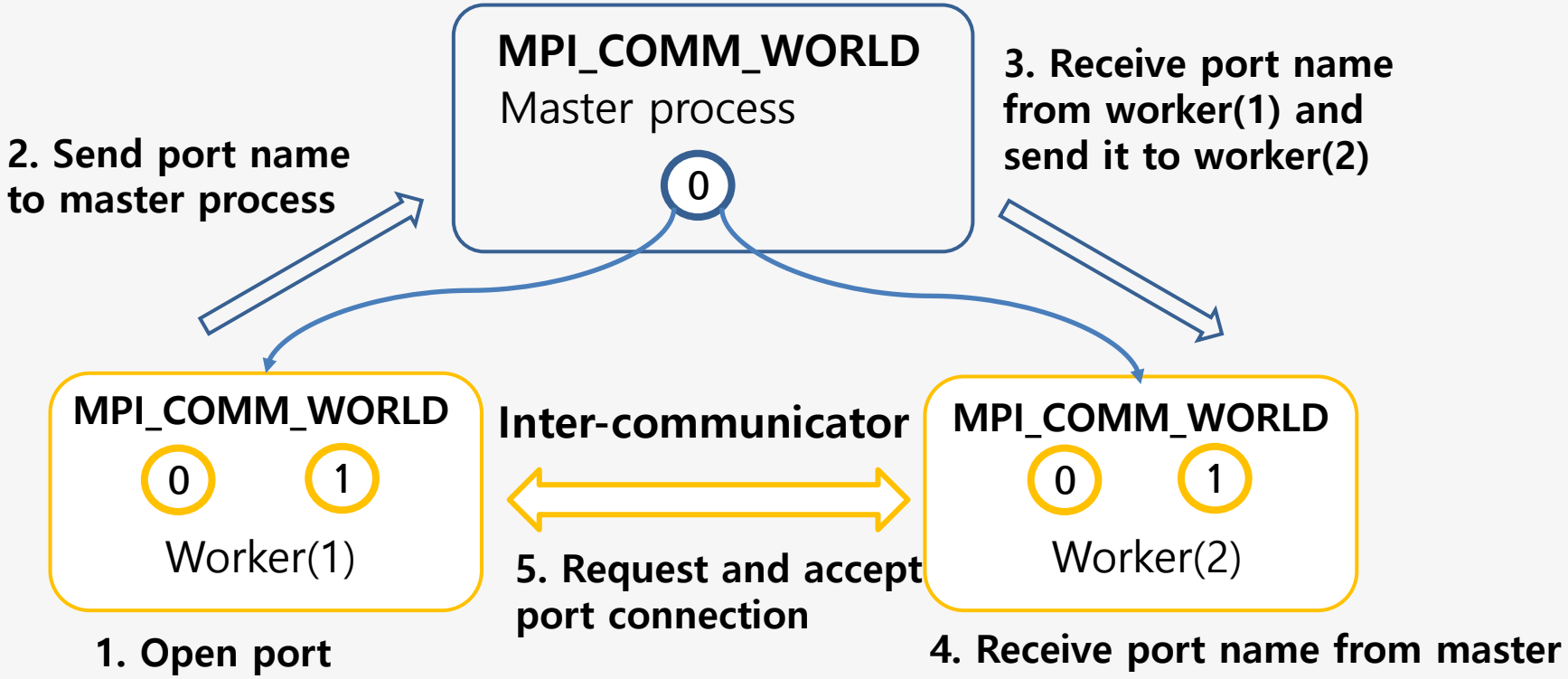
```
call MPI_OPEN_PORT(MPI_INFO_NULL, portname, ierr)
! send portname to client process
call MPI_SEND(portname, 255, MPI_CHAR, worker_rank, 2, MPI_COMM_WORLD, ierr)
! caution : use MPI_COMM_SELF, not MPI_COMM_WORLD for port connection
call MPI_COMM_ACCEPT(portname, MPI_INFO_NULL, 0, MPI_COMM_SELF, intercomm, ierr)
! do something
call MPI_CLOSE_PORT(portname, ierr)
```

- Worker process
  - Receive portname from master process
  - Request port connection

```
! receive portname from master process
call MPI_RECV(portname, 255, MPI_CHAR, master_rank, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
! caution : use MPI_COMM_SELF, not MPI_COMM_WORLD for port connection
call MPI_COMM_CONNECT(portname, MPI_INFO_NULL, 0, MPI_COMM_SELF, intercomm, ierr)
! do something
call MPI_COMM_DISCONNECT(MPI_COMM_WORKER0, ierr)
```

Port connection between two spawned processes

- Share the port name through master process which spawns two worker processes.



– A port name is created by the process which opens the port, so communication among three processes is required to share the port name

– After inter-communicator is created, the direct communication between spawned processes is possible

Code example - spawn and port connection

- Example in spawn_port folder
- Make and run with "mpirun –np 1 ./master.ex"
- Caution : officially OpenMPI depreciated MPI_Open_port from 1.8 version, so MPICH or Intel MPI is favorable.


thank you
danke
gracias
merci
obrigado
teşekkür ederim
go raibh maith agat
dziękuje
bedankt
спасибо
ngiyabonga
謝謝
sukriya
kop khun krap
terima kasih
mochchakkeram
arigatô
dakujem
grazie
tapadh leat
dank je
takk
sagolun
hvala
mauruuru
мерси
감사합니다
ευχαριστώ
ありがとう
рахмат
Баярлалаа
хвала
дякую
enkosi