Message Passing Interface (MPI)

Ομάδες Διεργασιών (Groups)

Ορισμός Ομάδων Διεργασιών (Groups)

Στόχος του ορισμού ομάδων διεργασιών είναι να διευκολύνει τον προγραμματιστή σε περιπτώσεις που επιθυμεί

(α) να διαφοροποιήσει τις εργασίες ή τμήματα αλγορίθμων που εκτελούνται σε διάφορα υποσύνολα των διεργασιών / επεξεργαστών του συνολικού παράλληλου περιβάλλοντος

(β) να εφαρμόσει σε συγκεκριμένα υποσύνολα διεργασιών/επεξεργαστών συναρτήσεις συλλογικής επικοινωνίας με τον βέλτιστο τρόπο

Συναρτήσεις Ορισμού Ομάδων Διεργασιών

- **MPI_Comm_group**
  Εξάγεται το handle του καθολικού group διεργασιών που αντιστοιχεί στον δοθέντα communicator (συνήθως MPI_COMM_WORLD)

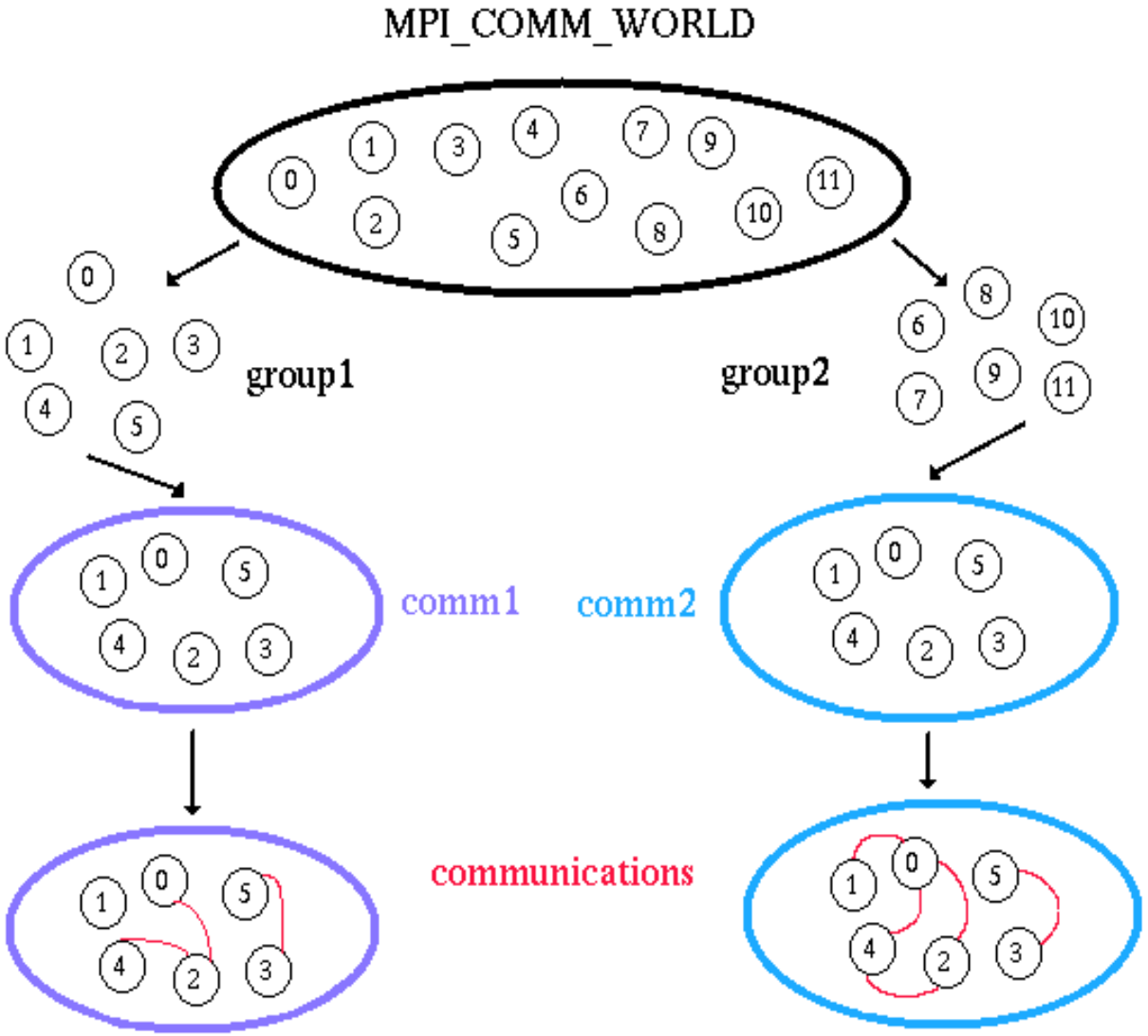
- **MPI_Group_incl ή MPI_Group_excl**
  Δημιουργείται νέο group (ως υποσύνολο ενός καθολικού group)

- **MPI_Comm_create**
  Δημιουργείται νέος communicator για ένα νέο group διεργασιών

- **MPI_Group_rank**
  Εξάγεται η τάξη της καλούσας διεργασίας στο δημιουργηθέν group

- **MPI_Comm_free and MPI_Group_free**

Ομάδες & communicators



Η συνάρτηση MPI_Comm_group

Εξάγεται το λεκτικό του group που αντιστοιχεί στον δοθέντα communicator:

int MPI_Comm_group (MPI_Comm comm, MPI_Group *group)

comm: Ο αρχικός communicator για τον οποίον θέλουμε το αντίστοιχο group.

group: Το λεκτικό του group που αντιστοιχεί στον communicator

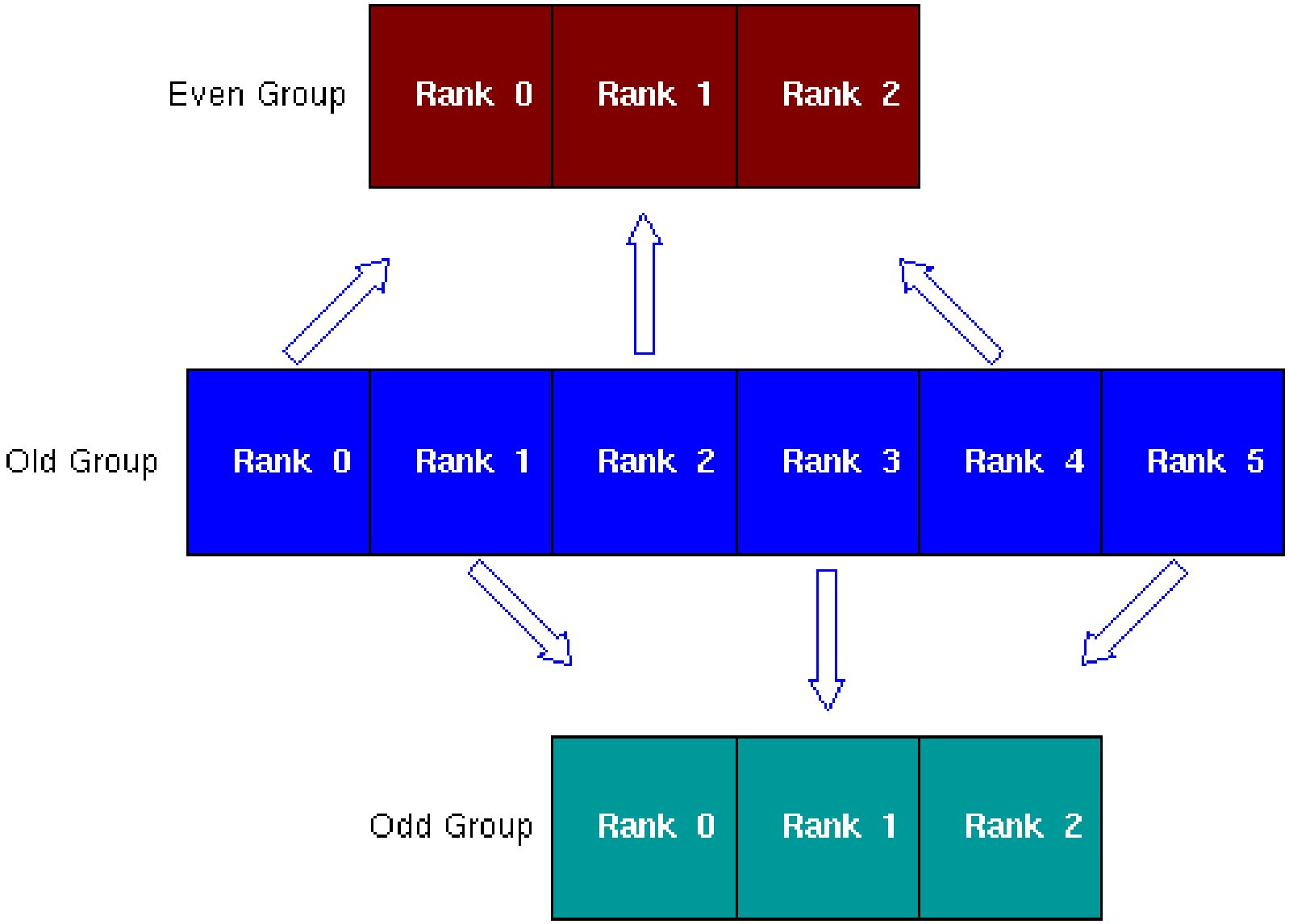
Η συνάρτηση MPI_Group_incl

Δημιουργείται ένα νέο group από ένα αρχικό group, ορίζοντας ποιες διεργασίες του αρχικού group θα συμμετάσχουν στο νέο.

```
int MPI_Group_incl (MPI_Group group, int n, int *ranks,
    MPI_Group *gr_out)
```

- **group:** Το αρχικό/γονικό group από το οποίο ξεκινάμε.
- **n:** Ο αριθμός των διεργασιών που θα συμμετάσχουν στο νέο group.
- **ranks:** Οι ταυτότητες των διεργασιών που θα συμμετάσχουν στο νέο group.
- **gr_out:** Το λεκτικό που αντιπροσωπεύει το νέο group.

MPI_Group_incl - Παράδειγμα



MPI_Group_incl - Παράδειγμα

```
#include "mpi.h"
MPI_Group group_world, odd_group, even_group;
int i, p, Neven, Nodd, members[8], ierr;

MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_group(MPI_COMM_WORLD, &group_world);

Neven = (p+1)/2;  /* processes of MPI_COMM_WORLD are divided */
Nodd = p - Neven;   /* into odd- and even-numbered groups */
for (i=0; i<Neven; i++) {   /* "members" determines members of
                                                    even_group */
   members[i] = 2*i;
};
MPI_Group_incl(group_world, Neven, members, &even_group);
```

Η συνάρτηση MPI_Group_excl

Δημιουργείται ένα νέο group από ένα αρχικό group, ορίζοντας ποιες διεργασίες του αρχικού group δεν θα συμμετάσχουν στο νέο.

```
int MPI_Group_excl (MPI_Group group, int n, int *ranks,
    MPI_Group *gr_out)
```

- **group:** Το αρχικό/γονικό group από το οποίο ξεκινάμε.
- **n:** Ο αριθμός των διεργασιών που δεν θα συμμετάσχουν στο νέο group.
- **ranks:** Οι ταυτότητες των διεργασιών που δεν θα συμμετάσχουν στο νέο group.
- **gr_out:** Το λεκτικό που αντιπροσωπεύει το νέο group.

Η συνάρτηση MPI_Comm_create

Μέσω της συνάρτησης αυτής δημιουργείται ο νέος communicator ο οποίος αντιστοιχεί σε ένα ήδη δημιουργηθέν group:

```
int MPI_Comm_create (MPI_Comm comm, MPI_Group group,
    MPI_Comm *comm_out )
```

- **comm:** Ο γονικός communicator από τον οποίον θα προκύψει ο νέος.
- **group:** Το ήδη δημιουργηθέν group για το οποίο θα φτιαχτεί ο νέος comm.
- **comm_out:** Το λεκτικό που θα αντιστοιχεί στον νέο communicator.

Η συνάρτηση MPI_Group_rank

Εξάγεται η νέα ταυτότητα της κάθε διεργασίας στο νέο δημιουργηθέν group (που δίνεται εδώ ως παράμετρος):

int MPI_Group_rank (MPI_Group group, int *rank)

group: **Το λεκτικό του group μέσα στο οποίο ψάχνουμε την ταυτότητα της κάθε διεργασίας.**

rank: **Η ταυτότητα της κάθε διεργασίας.**

MPI_Group_rank

```
#include "mpi.h"
MPI_Group group_world, new_group;
int i, p, ierr, group_rank;

MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_group(MPI_COMM_WORLD,
   &group_world);
MPI_Group_excl(group_world, 1, 0, &new_group);

MPI_Group_rank(new_group, &group_rank);
```

MPI_Group_rank

- In the above example, first a new group is created whose members are all but process 0 of the group MPI_COMM_WORLD. Then a query is made for the group rank.
- The ranks of MPI_COMM_WORLD have the range (0, 1, 2, ..., p-1). For this simple example, the rank range of the new group is (0, 1, 2, ..., p-2) as it has one less member (i.e., process 0) than MPI_COMM_WORLD.
- Consequently, the calling process' corresponding rank number in the new group would be 1 smaller. For instance, if the calling process is "i" (which is the rank number in the MPI_COMM_WORLD group), the corresponding rank number in the new group would be "i-1".
- Note, however, that if the calling process is process 0 which does not belong to new_group, MPI_Group_rank would return the value of MPI_UNDEFINED for group_rank, indicating that it is not a member of the new_group.

MPI_Group_rank

- It returns MPI_UNDEFINED (a negative number) if current process does not belong to group.
- MPI_UNDEFINED is implementation-dependent. For instance
  - MPI_UNDEFINED = -3 for SGI's MPI
  - MPI_UNDEFINED = -32766 for MPICH
- The foolowing code checks if calling process belongs to group:

```
if (group_rank == MPI_UNDEFINED) then
  /* group_rank does not belong to group */
        ...
else
  /* group_rank belongs to group */
        ...
endif
```

Παράδειγμα (group.c)

```
#include “mpi.h”
#include <stdio.h>
#define NPROCS 8
int main(argc,argv)
int argc;
char *argv[];  {
int       rank, new_rank, sendbuf, recvbuf, numtasks, ranks1[4]={0,1,2,3},
     ranks2[4]={4,5,6,7};
MPI_Group  orig_group, new_group;
MPI_Comm   new_comm;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
sendbuf = rank;

/* Extract the original group handle */
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
```

```
/* Divide tasks into two distinct groups based upon rank */
if (rank < NPROCS/2) {
  MPI_Group_incl(orig_group, NPROCS/2, ranks1, &new_group);
  }
else {
  MPI_Group_incl(orig_group, NPROCS/2, ranks2, &new_group);
  }

/* Create  new communicator */
MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM, new_comm);

MPI_Group_rank (new_group, &new_rank);
printf("rank= %d newrank= %d recvbuf= %d\n",rank,new_rank,recvbuf);

MPI_Finalize();
}
```

Πιθανή Έξοδος:

```
rank= 7 newrank= 3 recvbuf= 22
rank= 0 newrank= 0 recvbuf= 6
rank= 1 newrank= 1 recvbuf= 6
rank= 2 newrank= 2 recvbuf= 6
rank= 6 newrank= 2 recvbuf= 22
rank= 3 newrank= 3 recvbuf= 6
rank= 4 newrank= 0 recvbuf= 22
rank= 5 newrank= 1 recvbuf= 22
```

int MPI_Allreduce(void *sendbuf, void *recvbuf, int count,
MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Μέσω της συνάρτησης αυτής επιτελείται ότι και με τη συνάρτηση MPI_Reduce, με τη διαφορά ότι το αποτέλεσμα το γνωρίζουν όλες οι διεργασίες. Ισοδυναμεί με την εκτέλεση του ζεύγους συναρτήσεων MPI_Reduce και MPI_Bcast.

Η συνάρτηση MPI_Group_free

- **MPI_Group_free is used to free the group or deallocate a group object. Current operations using the group will complete normally.**

  int MPI_Group_free(MPI_Group *group)

- **The function returns an int error flag.**

Η συνάρτηση MPI_Comm_free

- Freeing a group does not free the communicator to which it belongs.

MPI_Comm_free

- Used to free the communicator or deallocate it. Current and pending operations will complete normally. Deallocation occurs only if no other active references to the communicator are present.

```
main(int argc, char **argv) {
    int me, count, count2;
    void *send_buf, *recv_buf, *send_buf2, *recv_buf2;
   MPI_Group MPI_GROUP_WORLD, grprem;
   MPI_Comm commslave;
   static int ranks[] = {0};
   MPI_Init(&argc, &argv);
   MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
   MPI_Comm_rank(MPI_COMM_WORLD, &me);
   MPI_Group_excl(MPI_GROUP_WORLD, 1, ranks, &grprem);
   MPI_Comm_create(MPI_COMM_WORLD, grprem, &commslave);
   if(me != 0){ /* compute on slave */
      MPI_Reduce(send_buf,recv_buff,count, MPI_INT, MPI_SUM, 1,  commslave);
                }

    MPI_Reduce(send_buf2, recv_buff2, count2, MPI_INT, MPI_SUM, 0,
           MPI_COMM_WORLD);
    MPI_Comm_free(&commslave);
    MPI_Group_free(&MPI_GROUP_WORLD);
    MPI_Group_free(&grprem);
    MPI_Finalize();  }
```

Άσκηση (group2.c)

- Να γραφεί κώδικας ο οποίος θα διαιρεί 8 διεργασίες του MPI_COMM_WORLD σε δύο νέα groups. Το ένα group αποτελείται από τις διεργασίες του MPI_COMM_WORLD με άρτιο rank και το άλλο group από τις διεργασίες του MPI_COMM_WORLD με περιττό rank .
- Στη συνέχεια θα εκτελείται μία ενδεικτική πράξη συλλογικής επικοινωνίας (συγκέντρωση των τάξεων των διεργασιών στη διεργασία-ρίζα ξεχωριστά για κάθε μία από τις ομάδες – και τους αντίστοιχους communicators – που δημιουργήθηκαν)

```
#include <stdio.h>
#include "mpi.h"
#define N 8
 int main(argc,argv)
int argc;
char *argv[];
{
int i, rank, new_rank, sendrank, recvarray[N/2], numtasks;
int list1[4]={0,2,4,6}, list2[4]={1,3,5,7};
 MPI_Group  orig_group, new_group;
MPI_Comm   new_comm;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
 sendrank = rank;
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
```

```
if ((rank%2)==0) {
  MPI_Group_incl(orig_group, N/2, list1, &new_group);
  }
else {
  MPI_Group_incl(orig_group, N/2, list2, &new_group);
  }
  MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);

MPI_Gather(&sendrank, 1, MPI_INT, recvarray, 1, MPI_INT, 0,
     new_comm);

MPI_Group_rank (new_group, &new_rank);
printf("rank= %d newrank= %d \n", rank, new_rank);
if (new_rank==0) {
  printf("gathered ranks for process %d = ", rank);
  for (i=0; i<N/2; i++)
    printf("%d ", recvarray[i]);
  printf("\n");
}
MPI_Finalize();
}
```

Η συνάρτηση MPI_Comm_split

Διαχωρίζει σε ένα βήμα, τον αρχικό/γονικό communicator σε δύο ή περισσότερα επιμέρους groups-communicators χρησιμοποιώντας αντίστοιχες ξεχωριστές τιμές-κλειδιά για κάθε έναν από αυτούς.

- Δημιουργεί απευθείας τους νέους communicators (και όχι πρώτα τα αντίστοιχα groups που πρέπει μετά να μετατραπούν σε communicators με άλλη συνάρτηση).

- Φτιάχνει όλους τους νέους communicators μαζί με μία κλήση και όχι με πολλές ξεχωριστές κλήσεις.

Η συνάρτηση MPI_Comm_split

int MPI_Comm_split (MPI_Comm comm, int color, int key,
MPI_Comm *comm_out)

- **comm: Ο αρχικός/γονικός communicator από τον οποίο ξεκινάμε.**
- **color: Οι διεργασίες με ίδια τιμή 'color' τοποθετούνται στον ίδιο communicator.**
- **key: Η ξεχωριστή τιμή-κλειδί προσδιορίζει την τάξη κάθε διεργασίας σε κάθε διαφορετικό νέο communicator.**
- **comm_out: Το λεκτικό κλήσης του νέου communicator (είναι κοινό για όλους τους νέους communicators στους οποίους διαχωρίστηκε ο αρχικός)**

MPI_Comm_split -Παράδειγμα

Δημιουργούνται τρεις νέοι communicators, μία με τις (5 συνολικά) διεργασίες με ταυτότητα από 0 έως 4, μία με τις (10 συνολικά) διεργασίες από 5 έως 14, και μία με τις (5 συνολικά) διεργασίες με ταυτότητα από 15 έως 19.

Επίσης, οι υπόλοιπες τυχόν υπάρχουσες διεργασίες (με ταυτότητα δηλαδή μεγαλύτερη ή ίση του 20) να μην ανήκουν σε κανέναν από τους δύο communicators (ομάδες).

```
MPI_Comm MPI_COMM_WORLD, newcomm; int myid, color;

/* βρες την ταυτότητα της κάθε διεργασίας προκειμένου να μπορέσεις στη συνέχεια να
     τους διαχωρίσεις και να δημιουργήσεις τις ζητούμενες νέες ομάδες */
 MPI_Comm_rank(comm, &myid);

/* καθόρισε ένα διαφορετικό 'color' για τις διεργασίες κάθε μίας από τις διαφορετικές
     ομάδες που θα δημιουργηθούν */
 if (myid < 5)              /* Επέλεξε τους 5 πρώτους */
    color = 1;
else if (myid < 15)      /* Επέλεξε στη συνέχεια τους */
    color = 2;              /* επόμενους 10 */
else if (myid < 20)      /* Επέλεξε στη συνέχεια τους */
     color = 3;             /* επόμενους 5 */
else                  /* Οι υπόλοιποι δε συμμετέχουν */
  color = MPI_UNDEFINED; /* σε κανένα group */

/* δημιουργία των νέων communicators */
 MPI_Comm_split(MPI_COMM_WORLD, color, myid, &newcomm);
```

MPI_Comm_split - Παράδειγμα

- For a 2D logical grid, create subgrids of rows and columns

```
/* logical 2D topology with nrow rows and mcol columns */

irow = my_rank/mcol;       /* logical row number */
jcol = my_rank%mcol;  /* logical column number */
comm2D = MPI_COMM_WORLD;

MPI_Comm_split(comm2D, irow, jcol, row_comm);
MPI_Comm_split(comm2D, jcol, irow, col_comm);
```

MPI_Comm_split - Παράδειγμα

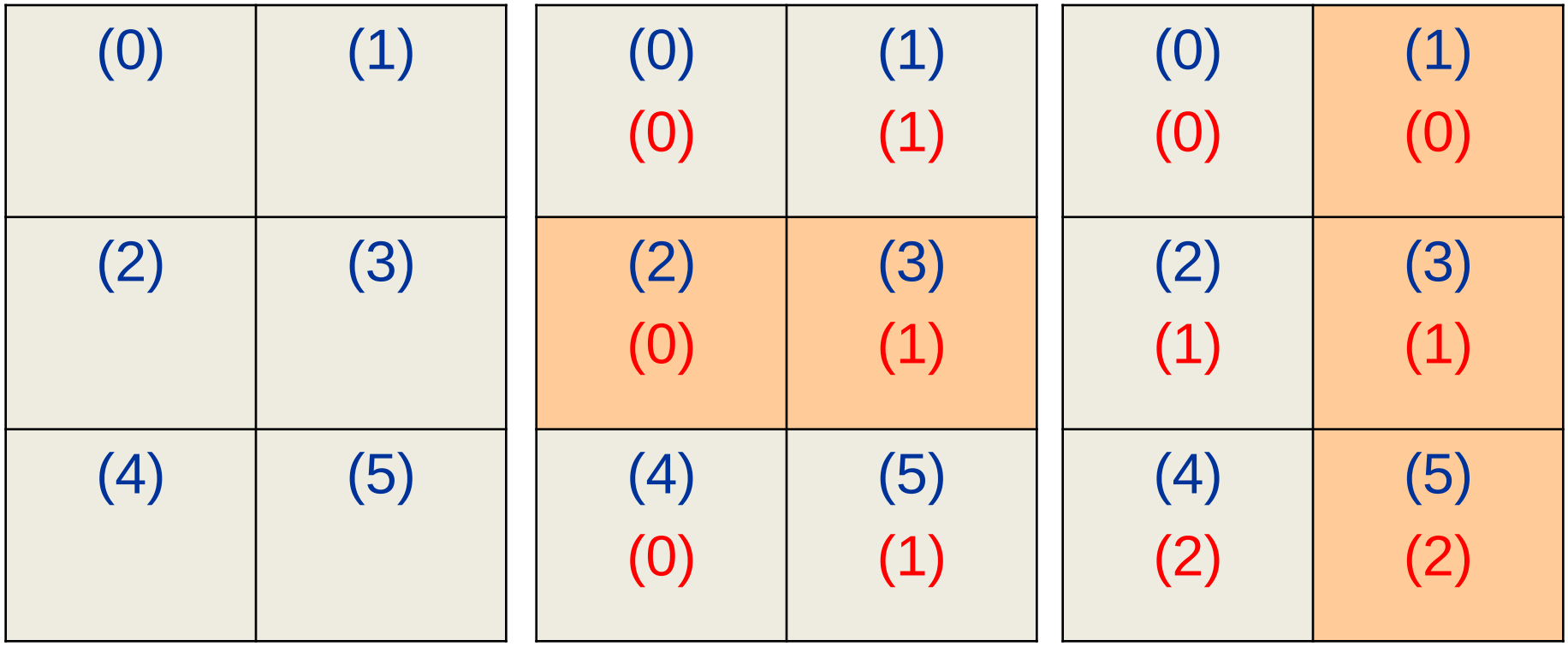
- Έστω ότι έχουμε 6 διεργασίες (0, 1, ..., 5)
- Ας θεωρήσουμε τις διεργασίες αυτές οργανωμένες σε ένα 3x2 πλέγμα.
- Προσδιορίζουμε τις τιμές irow and jcol ως συνάρτηση της τάξης my_rank της κάθε διεργασίας στον αρχικό communicator, όπως φαίνονται στον επόμενο πίνακα.

MPI_Comm_split - Παράδειγμα

- **Η πρώτη κλήση MPI_COMM_SPLIT προσδιορίζει το *irow* ως το "color" με *jcol* ως το "key" (ή τη διακριτή ταυτότητα κάθε διεργασίας μέσα στο group). Έτσι κάθε σειρά αποτελεί ένα νέο group (row subgrid).**
- **Η δεύτερη κλήση MPI_COMM_SPLIT προσδιορίζει το *jcol* ως το "color" με *irow* ως το "key" (ή τη διακριτή ταυτότητα κάθε διεργασίας μέσα στο group). Έτσι κάθε στήλη αποτελεί ένα νέο group (column subgrid).**

my_rank	0	1	2	3	4	5
irow	0	0	1	1	2	2
jcol	0	1	0	1	0	1

MPI_Comm_split - Παράδειγμα



2D λογικό πλέγμα (grid)

3 row subgrids

2 column subgrids

**Οι κόκκινοι αριθμοί είναι τα ranks στους νέους communicators*

MPI_Comm_split

- Processes in old_comm not part of any new group must have color defined as MPI_UNDEFINED. The corresponding returned NEW_COMM for these processes have value MPI_COMM_NULL.
- If two or more processes have the same key, the resulting rank numbers of these processes in the new communicator are ordered relative to their respective ranks in the old communicator. Recall that ranks in a communicator are by definition unique and ordered as (0, 1, 2, ..., p-1).

Παράδειγμα (split_ex.c)

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char **argv)
{
  MPI_Comm row_comm, col_comm;
  int myrank, size, P=4, Q=3, p, q;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  /* Determine row and column position */
  p = my_rank / Q;
  q = my_rank % Q; /* pick a row-major mapping */
```

```
/* Split comm into row and column comms */
  MPI_Comm_split(MPI_COMM_WORLD, p, q, &row_comm);
  /* color by row, rank by column */
  MPI_Comm_split(MPI_COMM_WORLD, q, p, &col_comm);
  /* color by column, rank by row */

  printf("[%d]:My coordinates are (%d,%d)\n",myrank,p,q);
  MPI_Finalize();
}
```