Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

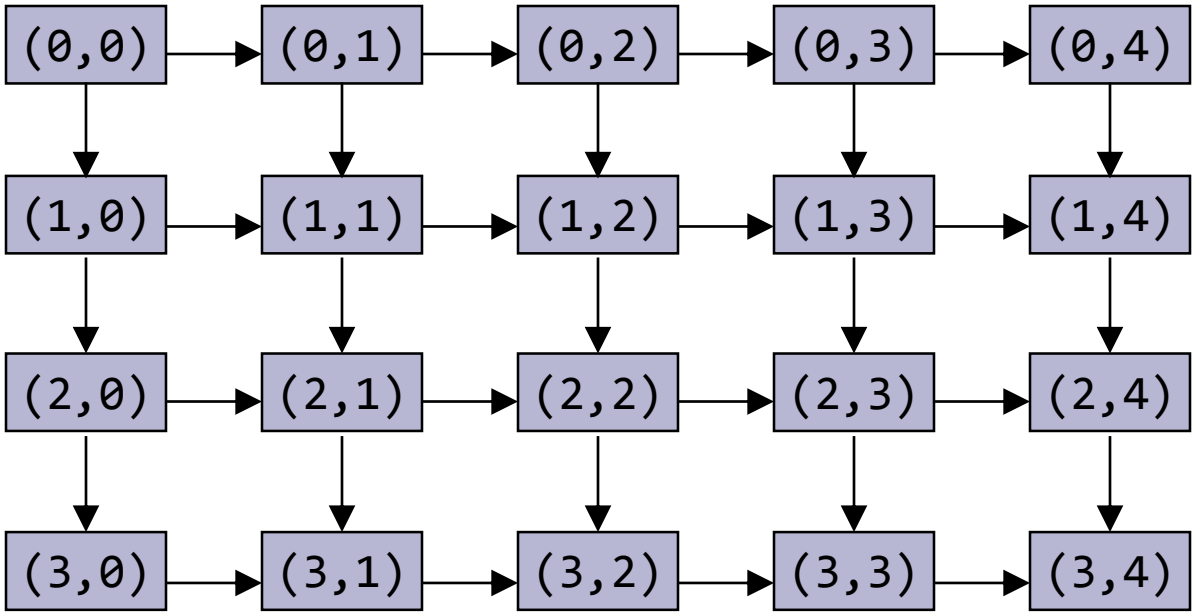
LAB-10. MPI (Μέρος F) – Τοπολογίες διεργασιών, Communicators

Τοπολογίες διεργασιών

- Ως τοπολογία διεργασιών ορίζεται η οργάνωση των διεργασιών MPI σύμφωνα με κάποιο κριτήριο.
- Δοθείσας μιας διεργασίας, καθίσταται εύκολος ο προσδιορισμός των γειτονικών διεργασιών.
- Χρήσιμο για επίλυση συγκεκριμένων αλγορίθμων (π.χ. επεξεργασία εικόνας ή σήματος).
- Παραδείγματα τοπολογίων:
  - 2-Δ Πλέγμα: Καρτεσιανή τοπολογία
  - 1-Δ: Δακτύλιος

Πλέγμα δύο διαστάσεων

- Παράδειγμα πλέγματος διεργασιών 4 x 5.



MPI_Cart_create

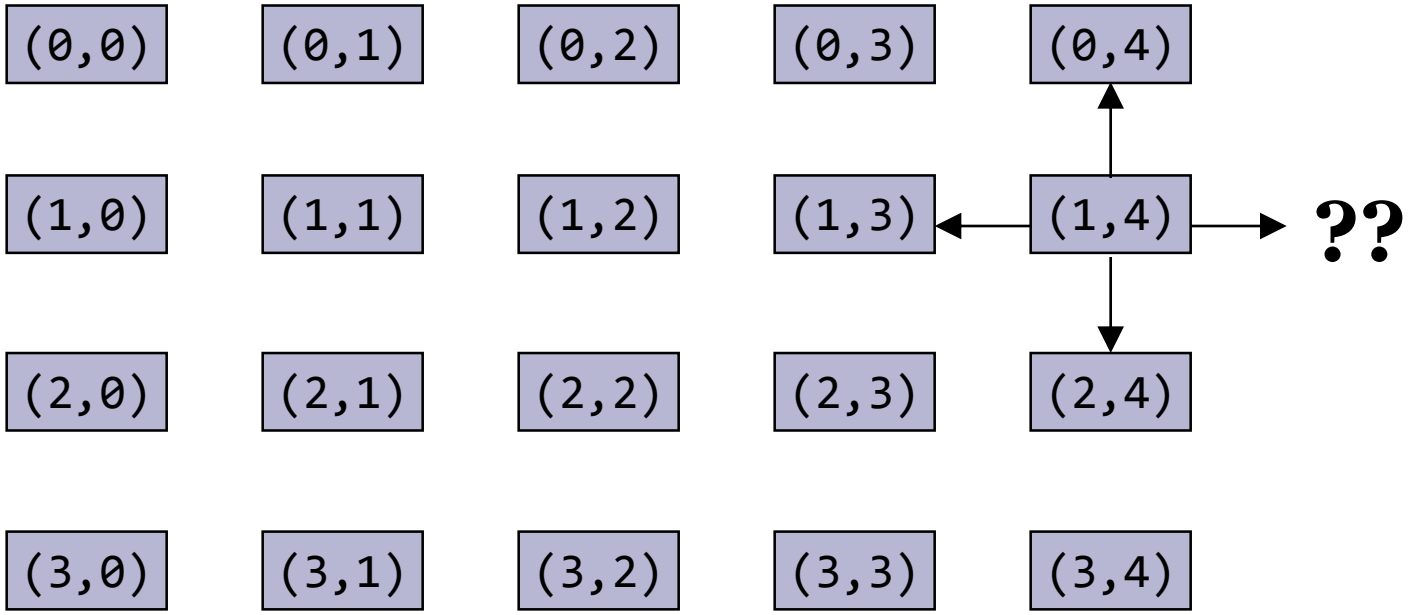
- Η συνάρτηση `MPI_Cart_create` κάνει n-διάστατη καρτεσιανή τακτοποίηση των διεργασιών.
- Δημιουργεί ένα νέο communicator στον οποίο έχει προστεθεί πληροφορία καρτεσιανής τοπολογίας.
- **`int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[], const int pers[], int reorder, MPI_Comm * comm_cart)`**
  - `comm_old`: Communicator εισόδου (ο δημιουργός).
  - `ndims`: Το πλήθος των διαστάσεων (π.χ. `ndims = 2` για το 2D πλέγμα του προηγούμενου παραδείγματος).

MPI_Cart_create

- **`int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[], const int pers[], int reorder, MPI_Comm * comm_cart)`**
  - `dims[]`: πίνακας ακεραίων μεγέθους `ndims`. Δηλώνει τον αριθμό διεργασιών ανά διάσταση πλέγματος.
  - `pers[]`: λογικός πίνακας μεγέθους `ndims`. Δηλώνει αν το πλέγμα είναι περιοδικό (true) σε κάθε διάσταση.
  - `reorder`: Λογική μεταβλητή. Δηλώνει αν τα ranks των διεργασιών μπορούν να αναδιαταχθούν.
  - `comm_cart`: Ο νέος communicator με καρτεσιανή τοπολογία.

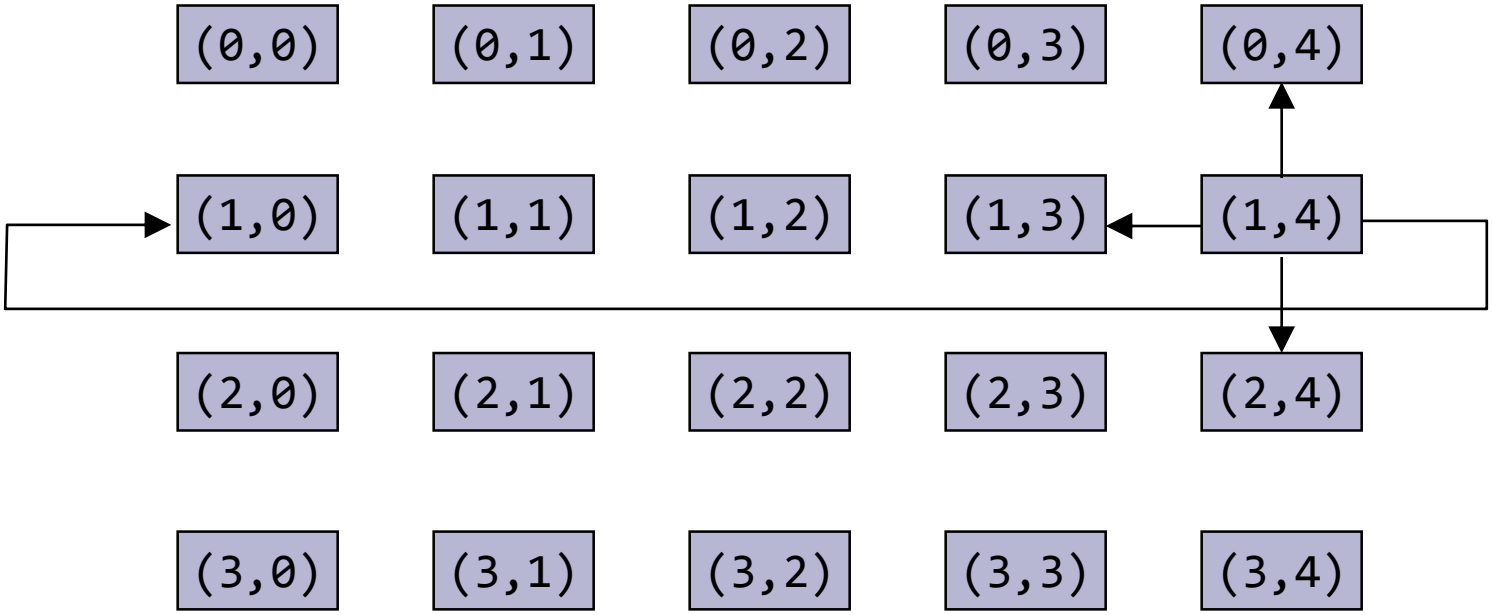
Περιοδικά/Μη-περιοδικά πλέγματα

- Ποιος είναι ο δεξιός γείτονας της διεργασίας `(1,4)`;



Το όρισμα periods (1)

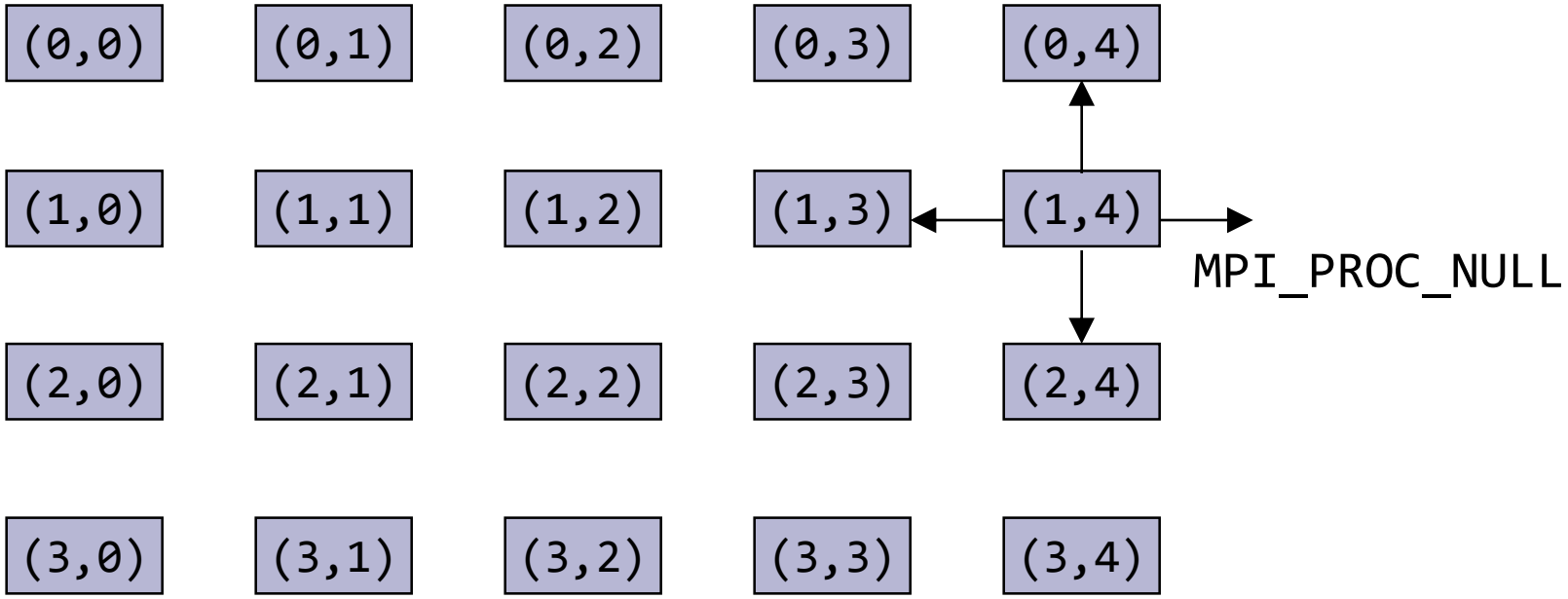
- Λογικός πίνακας μεγέθους ndims. Δηλώνει αν το πλέγμα είναι περιοδικό (true) σε κάθε διάσταση.



- Εδώ: `periods[0] = true;`
- Ομοίως για τις άλλες διαστάσεις (π.χ. `periods[1]`).

Το όρισμα periods (2)

- Εδώ: `periods[0] = false;`



- Ομοίως για τις άλλες διαστάσεις (π.χ. `periods[1]`).

Rank και συντεταγμένες διεργασίας

- Το rank μιας διεργασίας στον νέο communicator επιστρέφεται κατά τα γνωστά με την `MPI_Comm_rank`.
- Οι συντεταγμένες μιας διεργασίας πάνω στο καρτεσιανό πλέγμα επιστρέφονται με χρήση της `MPI_Cart_coords`.
- Περνιέται ως όρισμα ο νέος communicator και το rank της διεργασίας στον νέο communicator.

MPI_Cart_coords

- Επιστρέφονται οι συντεταγμένες μιας διεργασίας μέσα σε μία καρτεσιανή τοπολογία.
- **`int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int coords[])`**
  - `comm`: ο communicator με καρτεσιανή τοπολογία.
  - `rank`: η τάξη τα διεργασίας **μέσα στον `comm`**.
  - `maxdims`: το μέγεθος του πίνακα `coords`.
  - `coords[]`: πίνακας μεγέθους `maxdims` στον οποίο αποθηκεύονται οι συντεταγμένες της διεργασίας σε κάθε διάσταση του πλέγματος.

```
int main(int argc, char** argv) {
    int NumProcs, ProcRank, ProcCoordinates[2];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &NumProcs);

    int CommDims[2] = {4, 5};              /// The 2d topology: 4 rows, 5 cols.
    int CommPers[2] = {true, true};        /// Both dimensions will be periodic.
    int reorder = true;                            /// Assign arbitrary ranks.

    /// Create a communicator given the specified 2D topology.
    MPI_Comm CNEW;
    MPI_Cart_create(MPI_COMM_WORLD, 2, CommDims, CommPers, reorder, &CNEW);

    /// Get the rank and the coordinates of the current process in CNEW
    MPI_Comm_rank(CNEW, &ProcRank);
    MPI_Cart_coords(CNEW, ProcRank, 2, ProcCoordinates);

    printf("[MPI process %d] I am located at (%d, %d).\n",
           ProcRank, ProcCoordinates[0], ProcCoordinates[1]);

    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

Γειτονικές διεργασίες

- Η `MPI_Cart_create` δημιουργεί ένα νέο communicator με τις ίδιες διεργασίες όπως ο αρχικός communicator εισόδου, αλλά με τη συγκεκριμένη τοπολογία.
- Η εύρεση των γειτονικών διεργασιών πραγματοποιείται με την κλήση της συνάρτησης `MPI_Cart_shift`.
- Η συνάρτηση δέχεται μία μετατόπιση προς μία συγκεκριμένη κατεύθυνση και επιστρέφει τα ranks των αντίστοιχων διεργασιών πηγής και προορισμού.

MPI_Cart_shift

- Επιστρέφει τα ranks των διεργασιών πηγής και προορισμού με δεδομένη την κατεύθυνση και το βήμα της μετατόπισης.
- **`int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int *src, int *dest)`**
  - `comm`: communicator με καρτεσιανή τοπολογία.
  - `direction`: η διάσταση της μετατόπισης.
  - `disp`: το βήμα της μετατόπισης
  - `src`: rank της διεργασίας-πηγής.
  - `dest`: rank της διεργασίας-προορισμού.

MPI_Cart_shift

```
int NeighboursRanks[4];
char * directions[4] = {"Up", "Right", "Down", "Left" };

MPI_Cart_shift(COMM_NEW, 0, 1, &NeighboursRanks[0],
        &NeighboursRanks[2]);

MPI_Cart_shift(COMM_NEW, 1, 1, &NeighboursRanks[3],
        &NeighboursRanks[1]);

 printf("\tNeighbours:\n");
 for (int i = 0; i < 4; i++) {
    if (NeighboursRanks[i] == MPI_PROC_NULL) {
        printf("\t\t%s: NO NEIGHBOUR\n", directions[i]);
    } else {
        printf("\t\t%s: %d\n", directions[i], NeighboursRanks[i]);
    }
 }
```

MPI_Graph_create

- Η συνάρτηση MPI_Graph_create επιτρέπει γενικά τη δημιουργία μιας τοπολογίας που περιγράφεται από οποιοδήποτε γράφο.
- **`int MPI_Graph_create(MPI_Comm comm_old, int nnodes, const int indx[], const int edges[], int reorder, MPI_Comm * comm_grp)`**
  - `comm_old`: Communicator εισόδου (ο δημιουργός).
  - nnodes: το πλήθος κόμβων στο γράφο.
  - index: πίνακας που περιγράφει τους βαθμούς των κόμβων.
  - `edges`: πίνακας που περιγράφει τις ακμές του γράφου.
  - `reorder`: το αγνοούμε.
  - `comm_grp`: communicator με την τοπολογία γράφου.

Χρησιμότητα τοπολογιών

- Σε πολλές παράλληλες διασυνδέσεις υπολογιστών, κάποιοι υπολογιστές είναι πιο κοντά από κάποιους άλλους. Οι συναρτήσεις αυτές προσφέρουν μια διευθέτηση των διεργασιών σε μια τοπολογία η οποία κάνει τους «λογικούς» γείτονες να είναι κοντά στους φυσικούς «γείτονες».
- Κάποια συστήματα έχουν διάφορες τοπολογίες με πολλαπλά μονοπάτια που τροποποιούν τους γείτονες σε διαφορετικούς χρόνους. Οι συναρτήσεις αυτές απαλλάσσουν τον προγραμματιστή από την επίλυση θεμάτων τοπολογίας.

Groups διεργασιών

- Το MPI δίνει τη δυνατότητα ομαδοποίησης διεργασιών.
- Σκοπός της ομαδοποίησης είναι κυρίως η μετέπειτα διαχείριση των διεργασιών από κάποιο communicator.
- Είναι όμως δυνατή και η διαχείριση μέσω συναρτήσεων ένωσης (`MPI_Group_union`) και τομής (`MPI_Group_intersection`).

MPI_Comm_group

- Λήψη του group διεργασιών που διαχειρίζεται ένας συγκεκριμένος communicator.
- **`int MPI_Comm_group(MPI_Comm comm, MPI_Group * group)`**
  - `comm`: ο communicator του οποίου τις διεργασίες θα λάβουμε.
  - `group`: επιστρέφεται το group των διεργασιών σε ένα αντικείμενο `MPI_Group`.

MPI_Group_size

- Επιστρέφει το πλήθος των διεργασιών σε ένα group.
- **`int MPI_Group_size(MPI_Group group, int *size)`**
  - `group`: Το group των διεργασιών.
  - `size`: επιστρέφεται το πλήθος των διεργασιών στο group.

MPI_Group_free

- Καταργεί (απελευθερώνει) μια ομαδοποίηση.
- **`int MPI_Group_free(MPI_Group *group)`**
  - group: Το group των διεργασιών που θα καταργηθεί.

Παράδειγμα

```
int main (int argc, char** argv) {
    MPI_Init(&argc, &argv);

    // Get the group from the default communicator
    MPI_Group group;
    MPI_Comm_group(MPI_COMM_WORLD, &group);

    // Get the size of the group
    int size;
    MPI_Group_size(group, &size);

    // Each process prints the number of processes in the group
    printf("We are %d MPI processes in the group of the default
            communicator MPI_COMM_WORLD.\n", size);

    MPI_Finalize();

    return EXIT_SUCCESS;
}
```

Συναρτήσεις δημιουργίας group (1)

- Οι ακόλουθες συναρτήσεις δημιουργούν group διεργασιών από άλλα, ήδη υπάρχοντα group.
- Δημιουργία group από πίνακα με ranks διεργασιών:
  - `int MPI_Group_incl (MPI_Group group, int n, int *ranks, MPI_Group *group_out)`
  - `ranks`: ο πίνακας με τα ακέραια ranks των διεργασιών.
- Δημιουργία group από πίνακα με ranges διεργασιών:
  - `int MPI_Group_range_incl (MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)`
  - `ranges`: πίνακας τριάδων (first rank, last rank, stride).
  - `n`: πλήθος τριάδων.

Συναρτήσεις δημιουργίας group (2)

- Δημιουργία group από ένωση δύο άλλων groups:
  - `int MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *group_out)`
  - `group1, group2`: τα groups που θα ενωθούν.
- Δημιουργία group από την τομή δύο άλλων groups:
  - `int MPI_Group_intersection (MPI_Group group1, MPI_Group group2, MPI_Group *group_out)`
  - `group1, group2`: τα groups των οποίων η τομή θα υπολογιστεί.

Communicators από groups διεργασιών

- Πέραν των τοπολογιών, το MPI προσφέρει τη δυνατότητα απευθείας δημιουργίας νέων communicators χρησιμοποιώντας συγκεκριμένα μέλη από έναν άλλο communicator.
- Δηλαδή communicators μπορούν να δημιουργούνται απευθείας από δοθέντα groups διεργασιών.

MPI_Comm_create

- Δημιουργία communicator από δοθέν group διεργασιών.
- **`int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)`**
  - `comm`: communicator εισόδου (ο δημιουργός).
  - `group`: το group εισόδου των διεργασιών. Πρέπει να είναι υποσύνολο του `comm`.
  - `newcomm`: ο νέος communicator αποτελείται από τις διεργασίες της ομαδοποίησης `group`.

Παράδειγμα (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char ** argv) {
   int NumProcs, ProcRank;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &NumProcs);

   /// Get the group or processes of the default communicator
   MPI_Group world_group;
   MPI_Comm_group(MPI_COMM_WORLD, &world_group);

   /// Keep only the processes 0 and 1 in the new group.
   int ranks[2] = {0, 1};
   MPI_Group new_group;
   MPI_Group_incl(world_group, 2, ranks, &new_group);

   /// Create the new communicator from that group of processes.
   MPI_Comm new_communicator;
   MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_communicator);
```

Παράδειγμα (2)

```
  /// Do a broadcast between all processes
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  int value = 10;
  MPI_Bcast(&value, 1, MPI_INT, 0, MPI_COMM_WORLD);
  printf("Process %d took part to the global comm broadcast.\n", ProcRank);

  /// Wait all processes before proceeding to the second phase.
  MPI_Barrier(MPI_COMM_WORLD);

  /// Do a broadcast only between the processes of the new communicator.
  if (new_communicator == MPI_COMM_NULL) {
     printf("Process %d is OUT of the new comm broadcast.\n\n", ProcRank);
  } else {
     MPI_Bcast(&value, 1, MPI_INT, 0, new_communicator);
     printf("Process %d took part in the new comm broadcast.\n\n", ProcRank);
  }

  MPI_Finalize();

  return EXIT_SUCCESS;
}
```