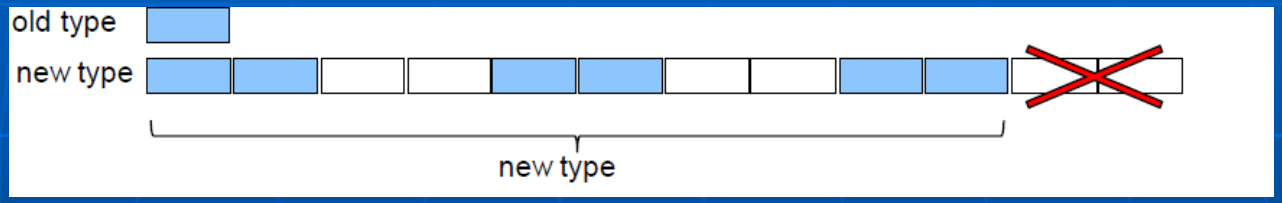
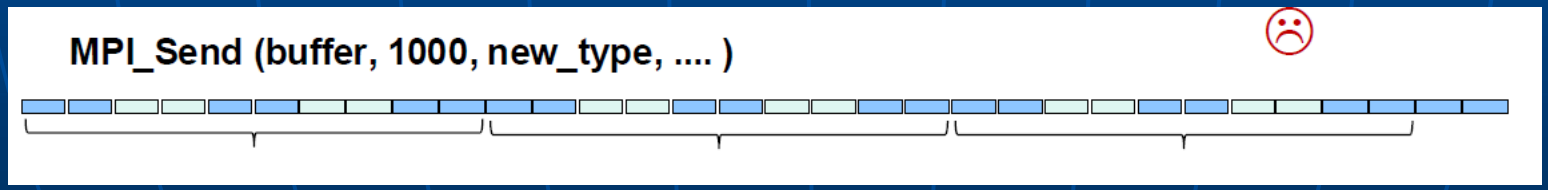
Izvedeni tipovi podataka

Ako želimo da u nekoj od funkcija za komunikaciju koristimo izvedeni tip podatka a da je pritom count>1, kao na primer MPI_Send(buff,10,izvedeni_tip,...) moramo voditi računa o tome šta će u tom slučaju biti poslato i da li je to ono što želimo da pošaljemo. Npr. funkcijom **MPI_Type_vector(3,2,4,oldtype,&newtype)** kreiramo tip:



extent = 10 x extent of “old type”

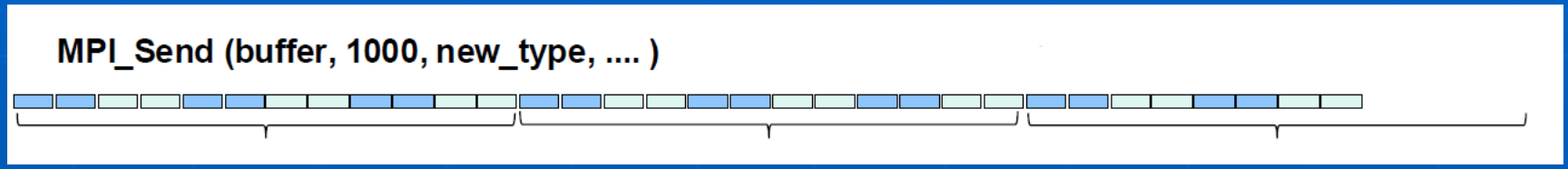
Ako želimo da sa MPI_Send pošaljemo cnt>1 podataka newtype, ono što će biti poslato je:



tj. slanje sledećeg podatka tipa newtype kreće sa adrese koja je odmah iza poslednjeg poslatog bajta podatka. Najčešće, ovo nije ono što mi želimo da bude poslato.

Izvedeni tipovi podataka

Ako je ono što je potrebno da bude poslato sa MPI_Send:



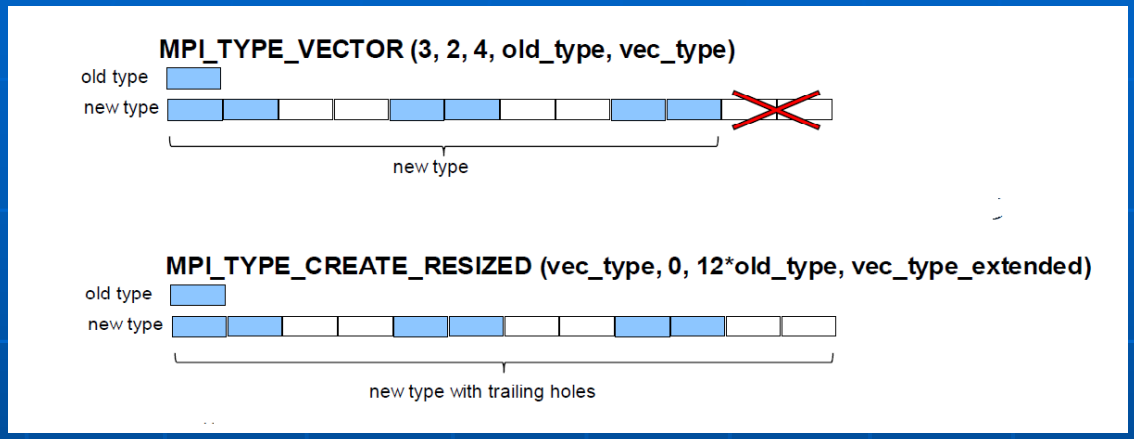
Problem možemo rešiti korišćenjem funkcije

int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint extent, MPI_Datatype *newtype)

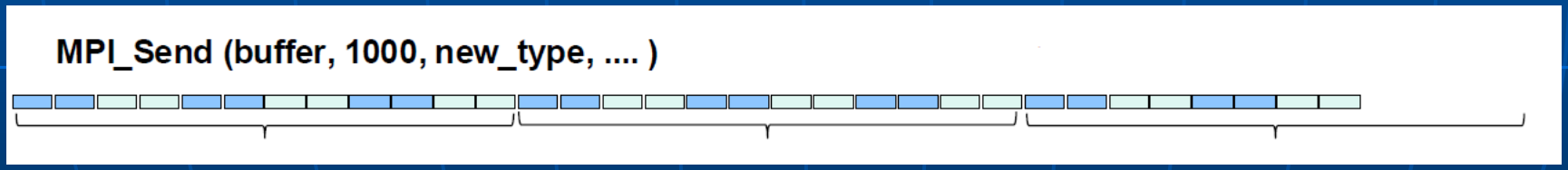
lb -nova donja granica tipa (namanji pomeraj u novom tipu, uglavnom je jednak onom u starom tipu, tj 0)
extent –nova veličina tipa koja utiče na to odakle će krenuti slanje sledeće jedinice novog tipa
oldtype-stari tip
newtype-novi tip

Izvedeni tipovi podataka

Dakle, uradićemo sledeće



Ono što će biti poslato sa MPI_Send je

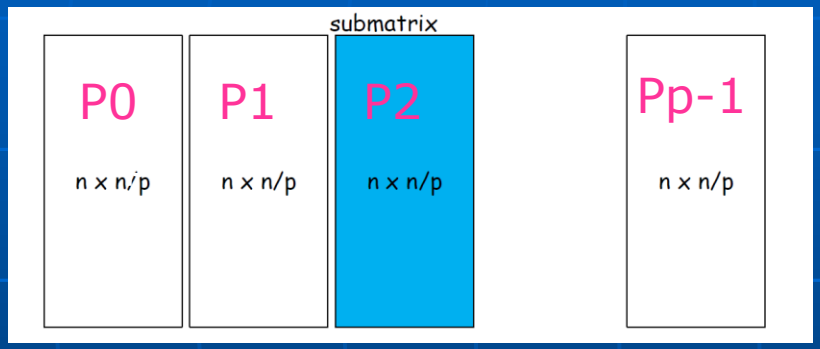


Možemo da zaključimo da MPI_Send(buff, cnt, ndt,...) praktično implementira for (i=0;i<cnt;i++) MPI_Send(buff[i*extent] , 1, ndt,...) .

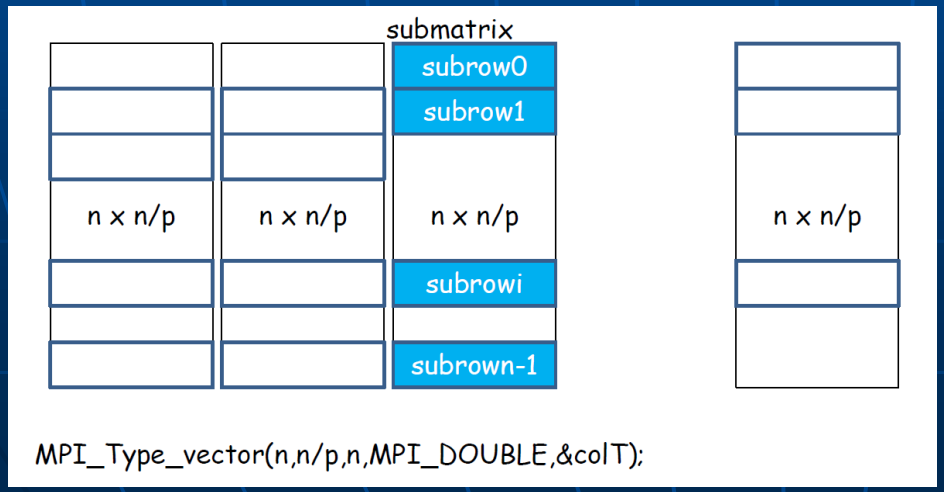
Sva ova zapažanja važe za korišćenje i drugih funkcija za komunikaciju (Scatter, Gather,...)

Izvedeni tipovi podataka

Primer. Želimo da pošaljemo iz master procesa (npr.P0) po n/p (n-dimenizija matrice, p-broj procesa, n deljivo sa p) kolona matrice A svakom procesu i da se u svakom procesu (pa i u P0) te vrednosti nakon slanja nadju u polju niz. U ovu svrhu možemo koristiti funkciju MPI_Scatter.

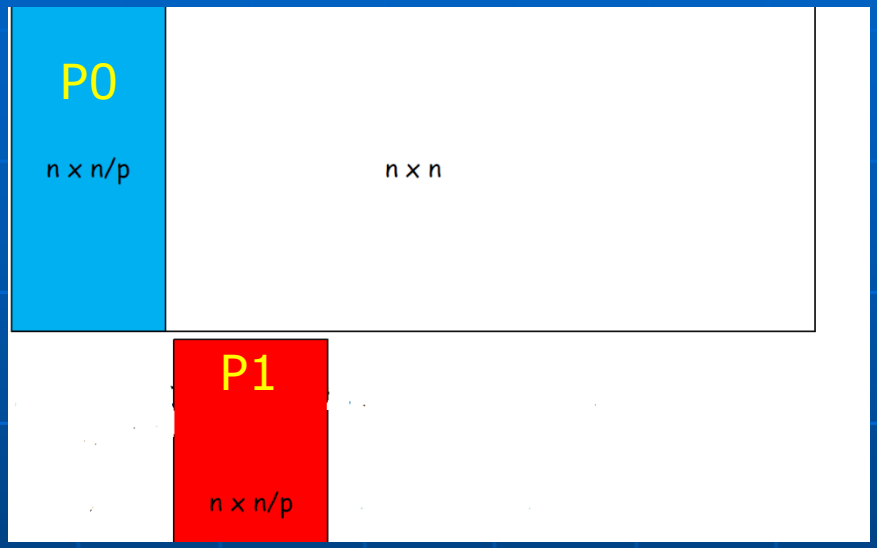


Da bi poslali n/p kolona odjednom kreiraćemo izvedeni tip colT:



Izvedeni tipovi podataka

Ono što bi se redom slalo iz svakog procesa sa MPI_Scatter(a,1,colT,niz,n*n/p,MPI_DOUBLE,0, MPI_COMM_WORLD) je:



Pošto to ne odgovara onom što smo želeli da bude poslato, koristimo:

```
MPI_Type_vector(n,n/p,n,MPI_DOUBLE,&colT);
MPI_Type_create_resized(colT,0,n/p*sizeof(double),&subT);
MPI_Type_commit(&subT);
```

koji nam omogućava da sa MPI_Scatter(a,1,subT,niz,n*n/p,MPI_DOUBLE,0, MPI_COMM_WORLD) pošaljemo ono što smo hteli.

Zad. Napisati MPI program koji realizuje množenje matrice A_{mxn} i matrice B_{nxk}, čime se dobija i prikazuje rezultujuća matrica C_{mxk}. Izračunavanje se obavlja tako što master proces šalje svakom procesu po jednu kolonu matrice A i po jednu vrstu matrice B. Svi elementi kolone matrice A se šalju odjednom. Svi procesi učestvuju u izračunavanjima potrebnim za generisanje rezultata programa. Zadatak rešiti isključivo

primenom grupnih operacija

m=2 n=3 k=4

$$\begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \end{bmatrix} * \begin{bmatrix} b00 & b01 & b02 & b03 \\ b10 & b11 & b12 & b13 \\ b20 & b21 & b22 & b23 \end{bmatrix} =$$

$$\begin{bmatrix} a00b00 + a01b10 + a02b20 & a00b01 + a01b11 + a02b21 & \dots & \dots \\ a10b00 + a11b10 + a12b20 & a10b01 + a11b11 + a12b21 & \dots & \dots \end{bmatrix}$$

Zad. Napisati MPI program koji realizuje množenje matrice A_{mxn} i matrice B_{nxk}, čime se dobija i prikazuje rezultujuća matrica C_{mxk}. Izračunavanje se obavlja tako što master proces šalje svakom procesu po jednu kolonu matrice A i po jednu vrstu matrice B. Svi elementi kolone matrice A se šalju odjednom. Svi procesi učestvuju u izračunavanjima potrebnim za generisanje rezultata programa. Zadatak rešiti isključivo primenom grupnih operacija

```
#include <stdio.h>
#include <mpi.h>
#define m 2
#define n 3
#define k 4
void main(int argc, char* argv[])
{int a[m][n],b[n][k],c[m][k],local_c[m][k],root=0,niza[m],nizb[k],rez[m],rank,i,j,p;
MPI_Datatype vector,column,dt;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&p);
```

```
MPI_Type_vector(m,1,n,MPI_INT,&vector);
MPI_Type_commit(&vector);
MPI_Type_create_resized( vector, 0, 1*sizeof(int),&column );
MPI_Type_commit(&column);
if (rank == root)
{
    for(i = 0; i < m; i++)

            for(j = 0; j < n; j++)
                        a[i][j] = i+j;

    for(i = 0; i < n; i++)
            for(j = 0; j < k; j++)
                    b[i][j] = 1+j-i;

}
MPI_Scatter(a,1,column,niza,m,MPI_INT,0,MPI_COMM_WORLD);
MPI_Scatter(b,k,MPI_INT,nizb,k,MPI_INT,0,MPI_COMM_WORLD);
for (i=0;i<m;i++)
            for (j=0;j<k;j++)
                        local_c[i][j]=niza[i]*nizb[j];
for (i=0;i<m;i++)
            for (j=0;j<k;j++)
                        c[i][j]=0;
```

```
MPI_Reduce(&local_c[0][0],&c[0][0],m*k,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD;
if (rank==0)
{
   for (i = 0; i< m; i++)
  {
            for (j = 0; j< k; j++)
                  printf("%d ",c[i][j]);

           printf("\n");
   }

     MPI_Finalize();

}
```

Grupe i komunikatori

- U nekim slučajevima potrebno je da se komunikacija obavi samo između određene grupe procesa osnovne grupe. To se može izvesti kreiranjem nove grupe procesa korišćenjem odgovarajućih MPI funkcija.
- Grupa predstavlja uređen skup procesa. Grupa je uvek vezana za neki komunikator. Komunikator sadrži grupu procesa koja može da komunicira.
- Pozivi MPI funkcija za komunikaciju između procesa koriste komunikator kao argument. MPI funkcije vezane za grupe specificiraju koji će se procesi koristiti prilikom pravljenja komunikatora.
- Procesi mogu biti članovi više od jedne grupe tj. komunikatora. Unutar svake od njih imaju jedinstveni identifikator.

Identifikator procesa

```
MPI_Comm communicator;
int myrank;
MPI_Comm_rank(communicator, &myrank);
```

Grupe i komunikatori-funkcije

Navešćemo neke od funkcija koje se koriste u tu svrhu:

int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)

-kao argument uzima komunikator i vraća odgovarajuću grupu procesa. Ova funkcija je važna za kasnije formiranje novih grupa od osnovne grupe

int MPI_Group_rank(MPI_Group group, int *rank)

–vraća identifikator procesa unutar grupe

int MPI_Group_size(MPI_Group group, int *size)

–vraća broj procesa unutar grupe

Grupe i komunikatori-funkcije

int MPI_Group_excl(MPI_Group group, int count, int *nonmembers, MPI_Group *new_group)

-vraća novu grupu tako što se iz stare grupe isključe procesi sa identifikatorima koji su definisani sa **nonmembers**, i kojih ima **count**. Redosled procesa u novoj grupi prati redosled procesa u staroj grupi.

int MPI_Group_incl(MPI_Group old_group, int count, int *members, MPI_Group *new_group)

-vraća novu grupu tako što procesi sa identifikatorima iz stare grupe koji su definisani sa **members** čine novu grupu. Procesa u novoj grupi ima **count**. Proces members[i] u novoj grupi ima rang i.

Grupe i komunikatori-funkcije

int MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

-vraća novu grupu koja se sastoji od procesa preseka grupa group1 i group2, s tim što je redosled procesa u novoj grupi kao u prvoj grupi

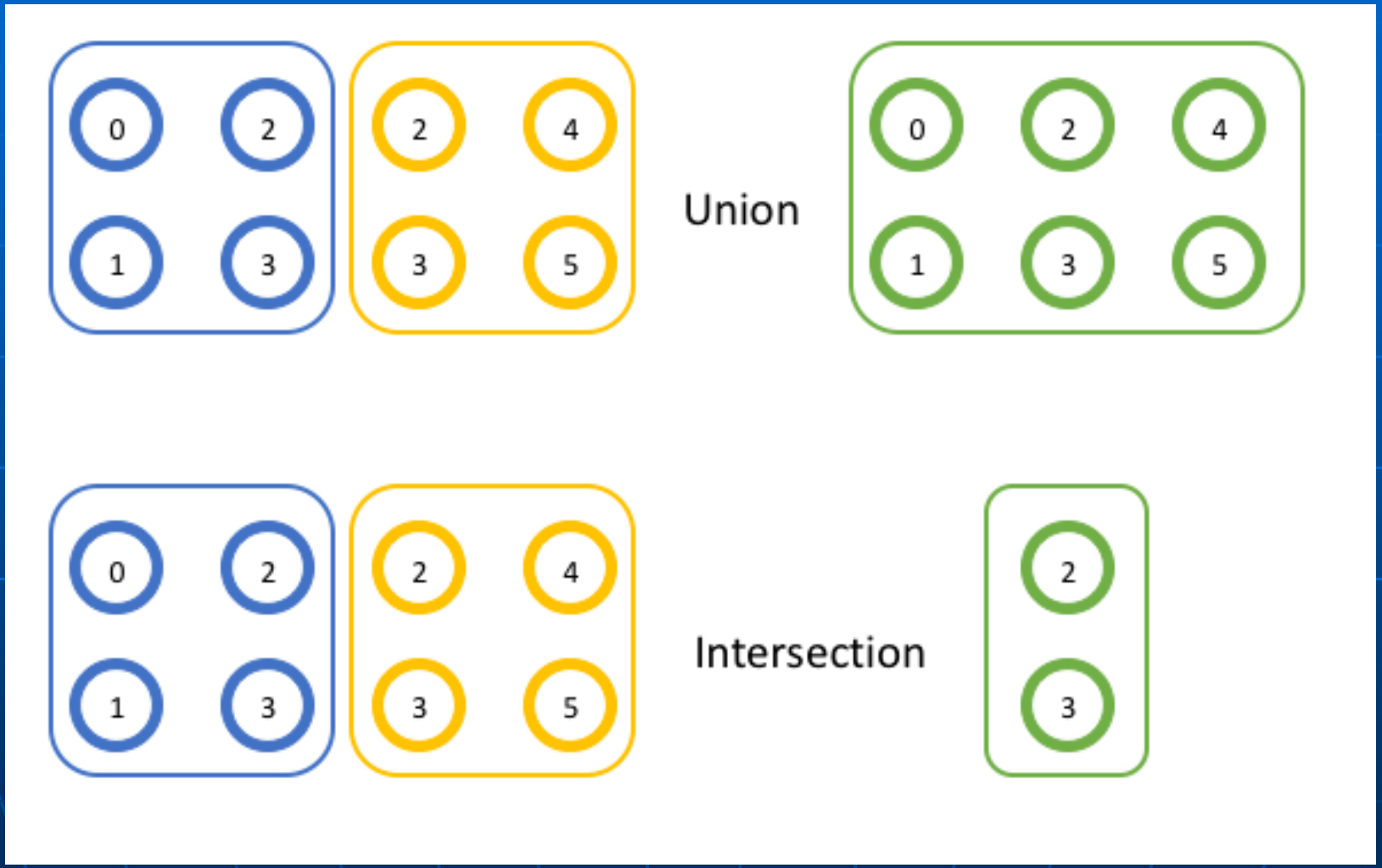
int MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

-vraća novu grupu koja se sastoji od procesa grupe group1 na koju se nadovezuju elementi group2 koji nisu group1

int MPI_Group_difference(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

-vraća novu grupu koja se sastoji od procesa grupe group1 koji nisu group2 uređene kao u group1.

Grupe i komunikatori-funkcije



Grupe i komunikatori-funkcije

int MPI_Comm_create(MPI_Comm old_comm, MPI_Group, MPI_Comm * new_comm)

-kreira novi komunikator new_comm od procesa iz grupe group koja kreirana iz old_comm

int MPI_Comm_split(MPI_Comm old_comm, int color, int key, MPI_Comm *new_comm)

-razbija komunikator old_comm na više delova tako sto svi procesi koji imaju istu vrednost za color pripadaju istom podkomunikatoru. Key određuje redosled procesa u podkomunikatorima. Ako je key isti za sve procese onda se redosled procesa preuzima iz starog komunikatora

Grupe i komunikatori-funkcije

int MPI_Comm_split(MPI_Comm, int color, int key, MPI_Comm *new_comm)

-razbija komunikator old_comm na više delova tako sto svi procesi koji imaju istu vrednost za color pripadaju istom podkomunikatoru. Key određuje redosled procesa u podkomunikatorima. Ako je key isti za sve procese onda se redosled procesa preuzima iz starog komunikatora

Rank	0	1	2	3	4	5	6	7	8	9	10
Process	a	b	c	d	e	f	g	h	i	j	k
Color	U	3	1	1	3	7	3	3	1	U	3
Key	0	1	2	3	1	9	3	8	1	0	0

{i, c, d}
{k, b, e, g, h}
{f}

Grupe i komunikatori-zadaci

Zad. Napisati MPI program kojim se vrši podela procesa članova komunikatora MPI_COMM_WORLD u dve grupe: grupu procesa sa neparnim identifikatorima i grupu procesa sa parnim identifikatorima

```
#include <mpi.h>
#include <stdio.h>
void main(int argc, char* argv[]) {
MPI_Group group_world, odd_group, even_group;
int i, p, Neven, Nodd, members[8], group_rank1,group_rank2,rank;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_group(MPI_COMM_WORLD, &group_world);
Neven = (p+1)/2;
Nodd = p - Neven;
for (i=0; i<Neven; i++) {
        members[i] = 2*i;
};
MPI_Group_incl(group_world, Neven, members, &even_group);
MPI_Group_rank(even_group, &group_rank1);
MPI_Group_excl(group_world, Neven, members, &odd_group);
```

Grupe i komunikatori-zadaci

```
MPI_Group_rank(odd_group, &group_rank2);
printf("moj rank je: %d, moj even rank je %d, moj odd rank je
%d\n",rank,group_rank1,group_rank2);
MPI_Finalize();
}
```

Izlaz:

```
moj rank je: 4, moj even rank je 2, moj odd rank je -32766
moj rank je: 6, moj even rank je 3, moj odd rank je -32766
moj rank je: 5, moj even rank je -32766, moj odd rank je 2
moj rank je: 2, moj even rank je 1, moj odd rank je -32766
moj rank je: 0, moj even rank je 0, moj odd rank je -32766
moj rank je: 3, moj even rank je -32766, moj odd rank je 1
moj rank je: 1, moj even rank je -32766, moj odd rank je 0
```

Grupe i komunikatori-zadaci

Rešenje zadatka na drugi način:

```
#include <mpi.h>
#include <stdio.h>
void main(int argc, char* argv[]) {
int i, p,color,rank,rankc,size,key;
MPI_Comm newcomm;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
color=rank%2;key=7;
MPI_Comm_split(MPI_COMM_WORLD,color,key,&newcomm);
MPI_Comm_size(newcomm, &size);
MPI_Comm_rank(newcomm, &rankc);
printf("moj rank je: %d, moj key  je %d, moj rankc je %d moj color je
%d moj size je %d\n",rank,key,rankc,color,size);
MPI_Finalize();
}
```

Grupe i komunikatori-zadaci

Izlaz:
moj rank je: 2, moj key je 7, moj rankc je 1 moj color je 0 moj size je 4
moj rank je: 3, moj key je 7, moj rankc je 1 moj color je 1 moj size je 3
moj rank je: 0, moj key je 7, moj rankc je 0 moj color je 0 moj size je 4
moj rank je: 5, moj key je 7, moj rankc je 2 moj color je 1 moj size je 3
moj rank je: 6, moj key je 7, moj rankc je 3 moj color je 0 moj size je 4
moj rank je: 4, moj key je 7, moj rankc je 2 moj color je 0 moj size je 4
moj rank je: 1, moj key je 7, moj rankc je 0 moj color je 1 moj size je 3

Grupe i komunikatori-zadaci

```
#include "stdio.h"
#include "mpi.h"
void main(int argc, char *argv[])
{
        int mcol, irow, jcol, p;
        MPI_Comm row_comm, col_comm, comm2D;
        int Iam, row_id, col_id;
        mcol=2;
        /* Starts MPI processes ... */
        MPI_Init(&argc, &argv);                    /* starts MPI */
        MPI_Comm_rank(MPI_COMM_WORLD, &Iam);  /* get current
process id */
        MPI_Comm_size(MPI_COMM_WORLD, &p);
        irow = Iam/mcol;       /* row number:          0 0 1 1 2 2 */
        jcol = Iam%mcol;       /* column number:       0 1 0 1 0 1 */


         comm2D = MPI_COMM_WORLD;
         MPI_Comm_split(comm2D, irow, jcol, &row_comm);
         MPI_Comm_split(comm2D, jcol, irow, &col_comm);
```

Grupe i komunikatori-zadaci

```
        MPI_Comm_rank(row_comm, &row_id);
        MPI_Comm_rank(col_comm, &col_id);

        printf("%8d %8d %8d %8d %8d\n",Iam,irow,jcol,row_id,col_id);
 MPI_Finalize();
}
```

Figure a.
2D logical Grid

(0)	*(1)*
(2)	*(3)*
(4)	*(5)*

Figure b.
3 Row Subgrids

(0) *(0)*	*(1)* *(1)*
(2) *(0)*	*(3)* *(1)*
(4) *(0)*	*(5)* *(1)*

Figure c.
2 Column Subgrids

(0) *(0)*	*(1)* *(0)*
(2) *(1)*	*(3)* *(1)*
(4) *(2)*	*(5)* *(2)*

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