

# Računarski sistemi visokih performansi

Nikola Vukić, Petar Trifunović, Veljko Petrović

Računarske vežbe  
Zimski semestar 2024/25.

*MPI 2*

# Sadržaj

- Kolektivna komunikacija.
- Kolektivna komunikacija nad podskupom procesa.
- Koncept grupe.

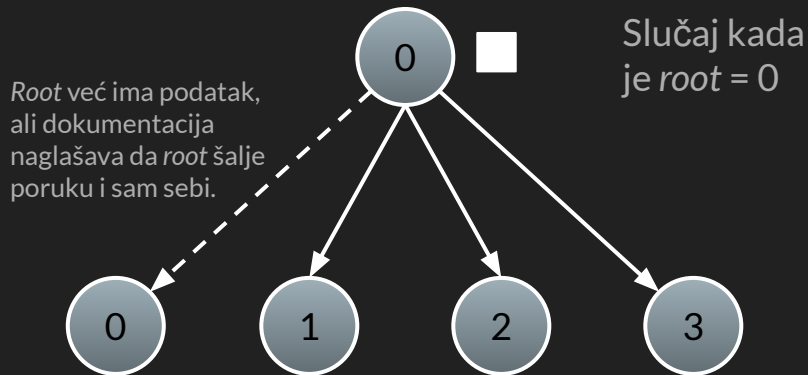
# Kolektivna komunikacija

# Kolektivna (engl. *collective*) komunikacija

- Komunikacija **svih** procesa unutar jednog komunikatora.
- **Svi** procesi iz komunikatora **moraju** izvršiti poziv funkcije da bi se ona izvršila; u suprotnom, doći će do blokiranja programa.

# Kolektivna komunikacija — *Broadcast*

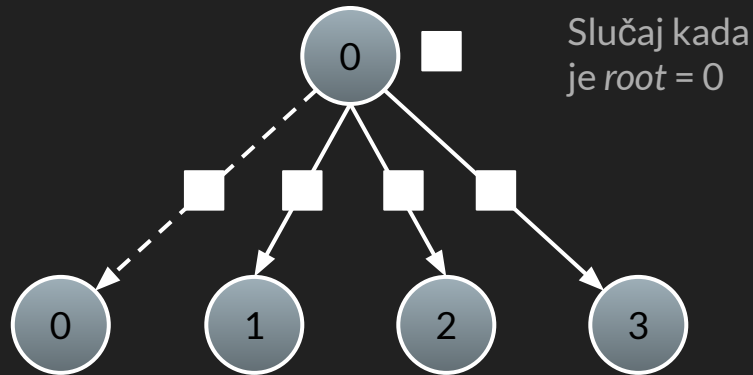
```
int MPI_Bcast(  
    void *buffer,  
    int count,  
    MPI_Datatype datatype,  
    int root,  
    MPI_Comm comm);
```



- Proces sa rankom *root* šalje *count* podataka tipa *datatype* sa memorijske lokacije *buffer* svim procesima iz komunikatora *comm*.

# Kolektivna komunikacija — *Broadcast*

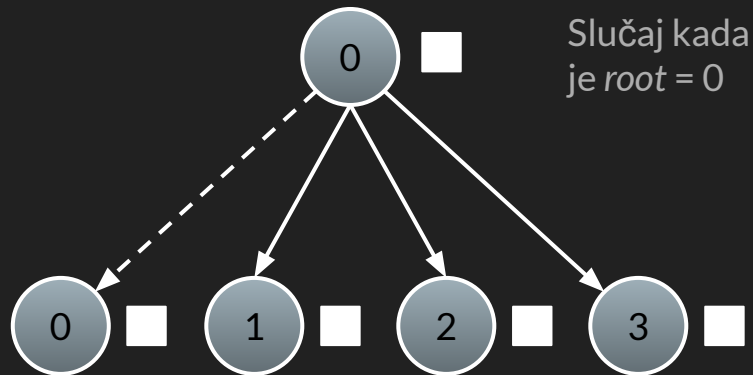
```
int MPI_Bcast(
    void *buffer,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm comm);
```



- Proces sa rankom *root* šalje *count* podataka tipa *datatype* sa memorijske lokacije *buffer* svim procesima iz komunikatora *comm*.

# Kolektivna komunikacija — *Broadcast*

```
int MPI_Bcast(  
    void *buffer,  
    int count,  
    MPI_Datatype datatype,  
    int root,  
    MPI_Comm comm);
```



- Proces sa rankom *root* šalje *count* podataka tipa *datatype* sa memorijske lokacije *buffer* svim procesima iz komunikatora *comm*.



# Kolektivna komunikacija — *Broadcast* primer

```
...  
int rank, root = 0;  
MPI_Init(&argc, &argv);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
int token;  
if (rank == root) token = 123;  
  
MPI_Bcast(&token, 1, MPI_INT, root, MPI_COMM_WORLD);  
printf("Proces %d primio token %d.\n", rank, token);  
MPI_Finalize();  
...
```

- `primeri/p05_mpi_bcast.c`

# Kolektivna komunikacija — *Broadcast* primer

```
...  
int rank, root = 0;  
MPI_Init(&argc, &argv);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
int token;  
if (rank == root) token = 123;  
  
MPI_Bcast(&token, 1, MPI_INT, root, MPI_COMM_WORLD);  
printf("Proces %d primio token %d.\n", rank, token);  
MPI_Finalize();  
...
```

svi procesi moraju izvršiti  
ovaj poziv da bi došlo do  
slanja podataka

- `primeri/p05_mpi_bcast.c`

## Zadatak 7 — *Broadcast*

- Napisati *OpenMPI* C implementaciju *MPI\_Bcast* funkcije korišćenjem *MPI\_Send* i *MPI\_Recv* funkcija.
- Ispisati poruku nakon što *root* proces pošalje podatke, kao i nakon što svaki od procesa primi podatke.
- Primer očekivanog ispisa:

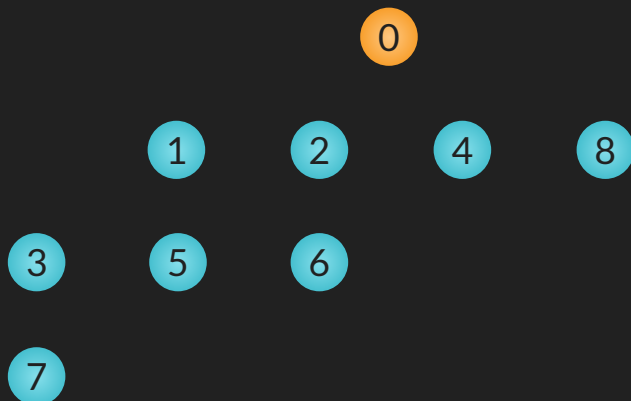
```
Process 0 šalje svima podatak 100
Process 1 prima poruku 100 od root procesa
Process 3 prima poruku 100 od root procesa
Process 2 prima poruku 100 od root procesa
```
- Primer rešenja: datoteka *resenja/07\_mpi\_bcast.c*.

## Zadatak 7 — Analiza rešenja

- Logički gledano, *Broadcast* radi tako što *root* šalje poruku od jednog procesa ka svim ostalim.
- Praktično, suština je da nakon poziva ove funkcije svi procesi sadrže istu kopiju podatka; nije neophodno da podatak teče tačno od *root* procesa ka svim ostalim.
- U zavisnosti od mrežne infrastrukture i konkretne implementacije standarda, *Broadcast* uvodi optimizacije u odnosu na jednostavan, ali neefkasan algoritam implementiran u ovom zadatku.

## Zadatak 7 — Analiza rešenja

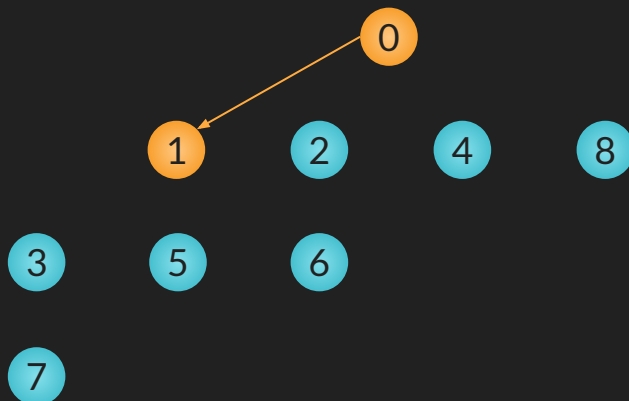
- Moguće je *Broadcast* implementirati u vidu strukture stabla, gde čvorovi koji su već primili poruku mogu da je proslede ostalim procesima.



Primer preuzet sa <https://www.codingame.com/playgrounds/349/introduction-to-mpi/broadcasting>

## Zadatak 7 — Analiza rešenja

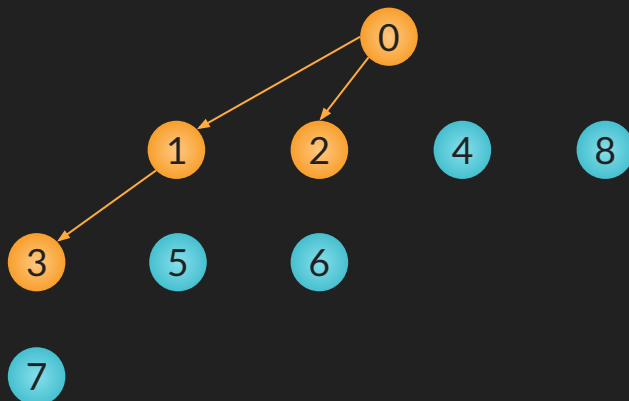
- Moguće je *Broadcast* implementirati u vidu strukture stabla, gde čvorovi koji su već primili poruku mogu da je proslede ostalim procesima.



Primer preuzet sa <https://www.codingame.com/playgrounds/349/introduction-to-mpi/broadcasting>

## Zadatak 7 — Analiza rešenja

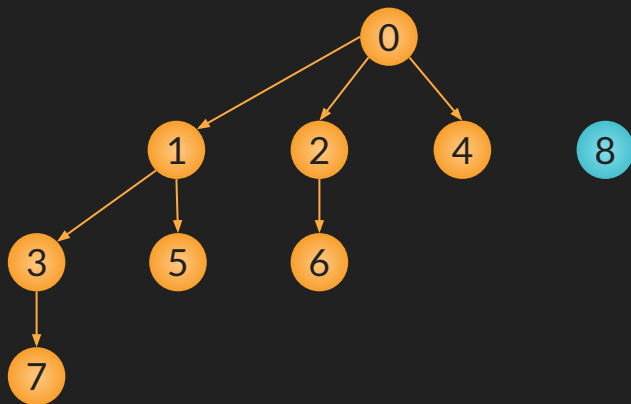
- Moguće je *Broadcast* implementirati u vidu strukture stabla, gde čvorovi koji su već primili poruku mogu da je proslede ostalim procesima.



Primer preuzet sa <https://www.codingame.com/playgrounds/349/introduction-to-mpi/broadcasting>

## Zadatak 7 — Analiza rešenja

- Moguće je *Broadcast* implementirati u vidu strukture stabla, gde čvorovi koji su već primili poruku mogu da je proslede ostalim procesima.

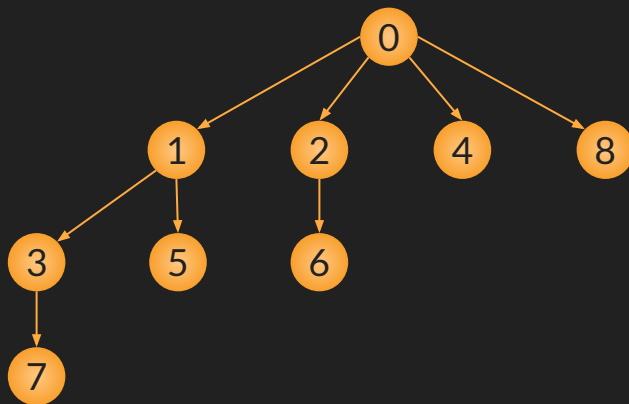


Primer preuzet sa <https://www.codingame.com/playgrounds/349/introduction-to-mpi/broadcasting>



## Zadatak 7 — Analiza rešenja

- Moguće je *Broadcast* implementirati u vidu strukture stabla, gde čvorovi koji su već primili poruku mogu da je proslede ostalim procesima.



Primer preuzet sa <https://www.codingame.com/playgrounds/349/introduction-to-mpi/broadcasting>

## Zadatak 7 — Analiza rešenja

- Kakva je razlika u vremenskoj kompleksnosti za  $N$  procesa kada jedan proces šalje poruku svima i kada se koristi struktura stabla?

## Zadatak 7 — Analiza rešenja

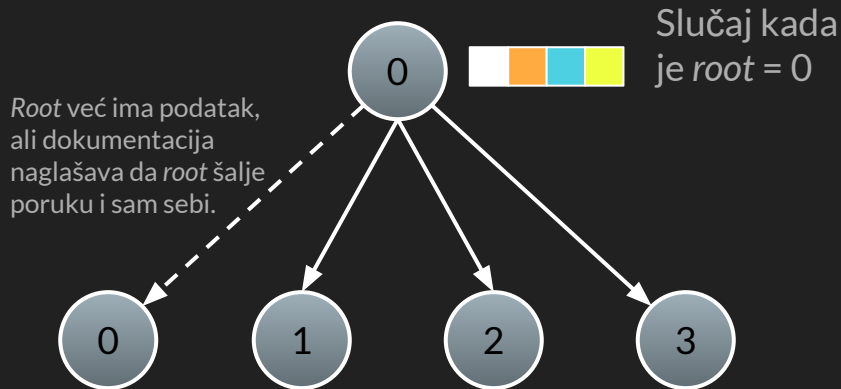
- Kakva je razlika u vremenskoj kompleksnosti za  $N$  procesa kada jedan proces šalje poruku svima i kada se koristi struktura stabla?
- U prvom slučaju, kompleksnost linearno zavisi od broja procesa, jer je potrebno  $N-1$  rundi razmene podataka, odnosno kompleksnost je  $O(N)$ .

## Zadatak 7 — Analiza rešenja

- Kakva je razlika u vremenskoj kompleksnosti za  $N$  procesa kada jedan proces šalje poruku svima i kada se koristi struktura stabla?
- U prvom slučaju, kompleksnost linearno zavisi od broja procesa, jer je potrebno  $N-1$  rundi razmene podataka, odnosno kompleksnost je  $O(N)$ .
- Kada je stablo u pitanju, broj potrebnih rundi je sledeći:
  - $N = 2$  — 1 runda
  - $N = 4$  — 2 runde
  - $N = 8$  — 3 runde
  - $N = 16$  — 4 runde
- Kada je u pitanju stablo, potrebno je  $\lceil \log_2 N \rceil$  rundi, odnosno kompleksnost je  $O(\log N)$ .

# Kolektivna komunikacija — *Scatter*

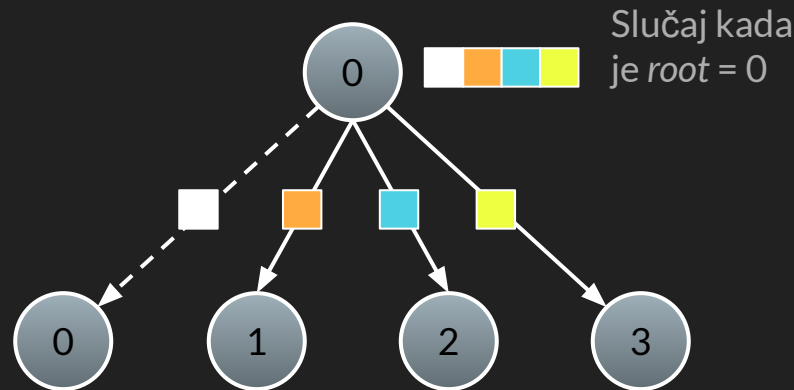
```
int MPI_Scatter(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int root,  
    MPI_Comm comm);
```



- Proces sa rankom *root* šalje svakom procesu iz komunikatora *comm* po *sendcount* podataka tipa *sendtype* sa memorijske lokacije *sendbuf*, i to počev od procesa sa najmanjim rankom.
- Svi procesi prihvataju *recvcount* podataka tipa *recvtype* na memorijsku lokaciju *recvbuf*.

# Kolektivna komunikacija — *Scatter*

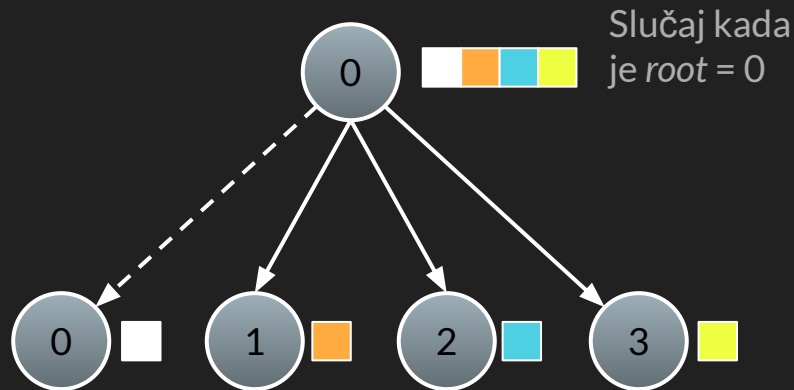
```
int MPI_Scatter(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int root,  
    MPI_Comm comm);
```



- Proces sa rankom *root* šalje svakom procesu iz komunikatora *comm* po *sendcount* podataka tipa *sendtype* sa memorijske lokacije *sendbuf*, i to počev od procesa sa najmanjim rankom.
- Svi procesi prihvataju *recvcount* podataka tipa *recvtype* na memorijsku lokaciju *recvbuf*.

# Kolektivna komunikacija — *Scatter*

```
int MPI_Scatter(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int root,  
    MPI_Comm comm);
```



- Proces sa rankom *root* šalje svakom procesu iz komunikatora *comm* po *sendcount* podataka tipa *sendtype* sa memorijske lokacije *sendbuf*, i to počev od procesa sa najmanjim rankom.
- Svi procesi prihvataju *recvcount* podataka tipa *recvtype* na memorijsku lokaciju *recvbuf*.

# Kolektivna komunikacija — *Scatter* primer

```
...
int rank, size, root = 0, datalen = 8;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

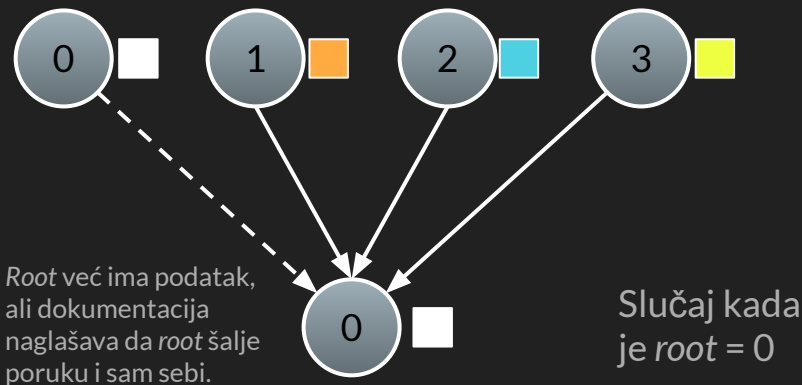
int *data = NULL, *partial_data = NULL;
int piecelen = datalen / size;
if (rank == root) {
    // inicijalizacija data niza
}
partial_data = (int *) malloc(sizeof(int) * piecelen);
MPI_Scatter(data, piecelen, MPI_INT, partial_data, piecelen, MPI_INT, root, MPI_COMM_WORLD);
...
```

- *primeri/p06\_mpi\_scatter.c*



# Kolektivna komunikacija — *Gather*

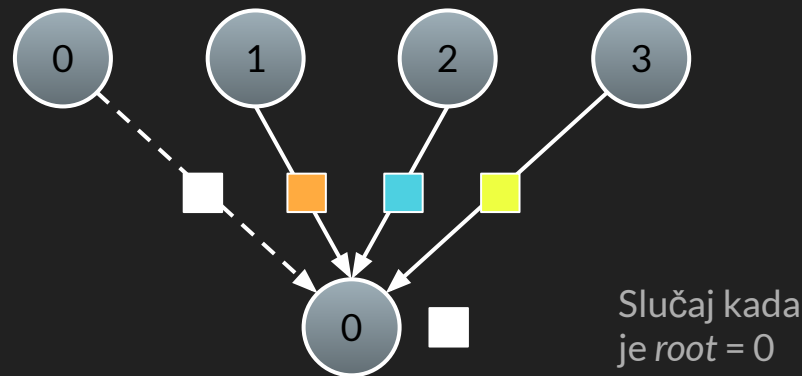
```
int MPI_Gather(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int root,  
    MPI_Comm comm);
```



- Svi procesi iz komunikatora *comm* šalju procesu sa rankom *root* *sendcount* podataka tipa *sendtype* sa memorijske lokacije *sendbuf*, a *root* ukupno prihvata *recvcount* podataka tipa *recvtype* na memorijsku lokaciju *recvbuf*, i to redom, počev od procesa sa najmanjim rankom.

# Kolektivna komunikacija — *Gather*

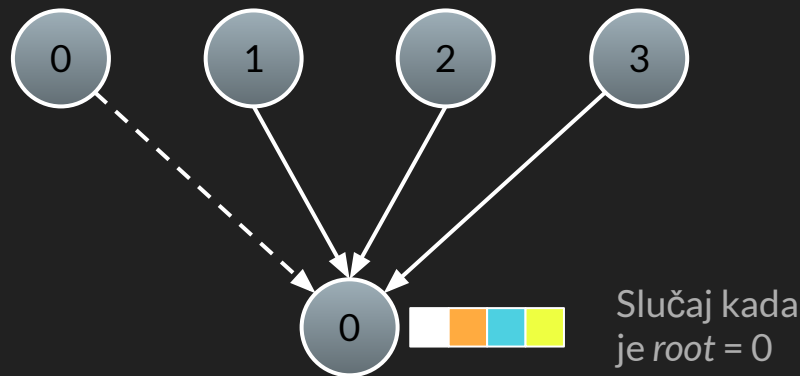
```
int MPI_Gather(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int root,  
    MPI_Comm comm);
```



- Svi procesi iz komunikatora *comm* šalju procesu sa rankom *root* *sendcount* podataka tipa *sendtype* sa memorijske lokacije *sendbuf*, a *root* ukupno prihvata *recvcount* podataka tipa *recvtype* na memorijsku lokaciju *recvbuf*, i to redom, počev od procesa sa najmanjim rankom.

# Kolektivna komunikacija — *Gather*

```
int MPI_Gather(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int root,  
    MPI_Comm comm);
```



- Svi procesi iz komunikatora *comm* šalju procesu sa rankom *root* *sendcount* podataka tipa *sendtype* sa memorijske lokacije *sendbuf*, a *root* ukupno prihvata *recvcount* podataka tipa *recvtype* na memorijsku lokaciju *recvbuf*, i to redom, počev od procesa sa najmanjim rankom.

# Kolektivna komunikacija — *Gather* primer

```
...
int rank, size, root = 0, datalen = 8;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int *data = NULL, *partial_data = NULL;
int piecelen = datalen / size;
partial_data = (int *) malloc(sizeof(int) * piecelen);

for (int i = 0; i < piecelen; i++) {
    // inicijalizacija parcijalnog niza
}

if (rank == root) {
    data = (int *) malloc(sizeof(int) * datalen);
}
MPI_Gather(partial_data, piecelen, MPI_INT, data, piecelen, MPI_INT, root, MPI_COMM_WORLD);
...
```

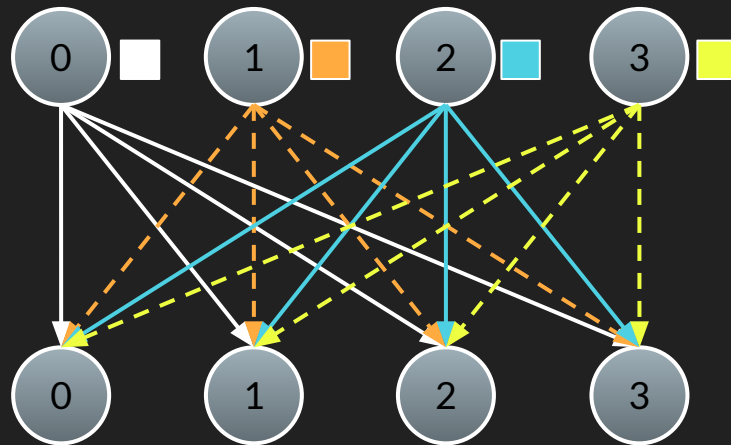
- *primeri/p07\_mpi\_gather.c*

## Zadatak 8 — Računanje srednje vrednosti

- Napisati *OpenMPI C* program koji računa srednju vrednost elemenata niza u više procesa korišćenjem funkcija *MPI\_Scatter* i *MPI\_Gather*.
- Program napisati tako da:
  - Korenski proces inicijalizuje niz dužine  $n$  nasumično izgenerisanim celim brojevima.
  - Razdeliti izgenerisani niz na jednake delove između svih procesa.
  - Svaki proces treba da izračuna svoju parcijalnu srednju vrednost.
  - Nakon što su sve parcijalne vrednosti sračunate, prebacuju se nazad korenskom procesu koji od parcijalnih računa ukupnu srednju vrednost.
  - Kao argument komandne linije proslediti broj željenih elemenata po procesu.
- Primer rešenja: datoteka *resenja/08\_mpi\_avg.c*.

# Kolektivna komunikacija — *Allgather*

```
int MPI_Allgather(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    MPI_Comm comm);
```

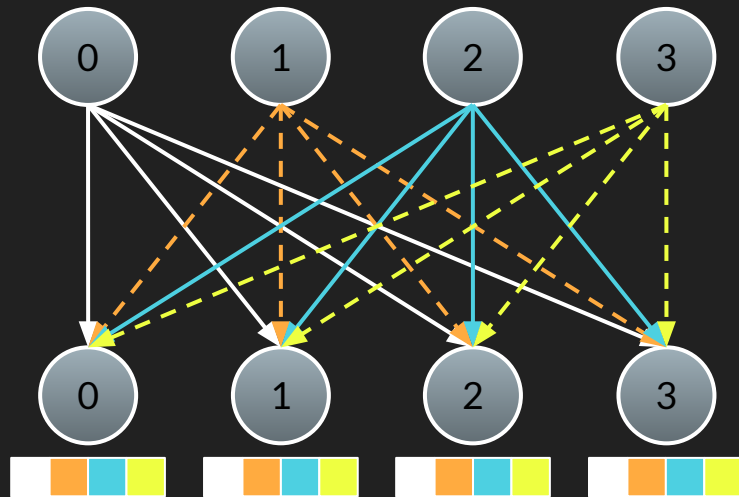


(Linije su isprekidane samo radi preglednosti)

- Efekat je kao da je svaki svaki od  $n$  procesa iz komunikatora *comm* pozvao po  $n$  puta funkciju *MPI\_Gather*, za vrednosti *root* = 0, ...,  $n-1$ .
- Može se posmatrati i kao poziv *MPI\_Gather* praćen pozivom *MPI\_Bcast*.

# Kolektivna komunikacija — *Allgather*

```
int MPI_Allgather(  
    const void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    MPI_Comm comm);
```



- Efekat je kao da je svaki svaki od  $n$  procesa iz komunikatora *comm* pozvao po  $n$  puta funkciju *MPI\_Gather*, za vrednosti *root* = 0, ...,  $n-1$ .
- Može se posmatrati i kao poziv *MPI\_Gather* praćen pozivom *MPI\_Bcast*.

# Kolektivna komunikacija — *Allgather* primer

```
...  
int rank, size;  
MPI_Init(&argc, &argv);  
MPI_Comm_size(MPI_COMM_WORLD, &size);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
int *data = (int *) malloc(sizeof(int) * size);  
int token = rank;  
MPI_Allgather(&token, 1, MPI_INT, data, 1, MPI_INT, MPI_COMM_WORLD);  
...
```

- *primeri/p08\_mpi\_allgather.c*



# Kolektivna komunikacija — *Allgather* primer

...

```
int rank, size;
```

```
MPI_Init(&argc, &argv);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

nakon ovog poziva, svi procesi će  
imati identične elemente u nizu  
*data*, čija je veličina jednaka *size*

```
int *data = (int *) malloc(sizeof(int) * size);
```

```
int token = rank;
```

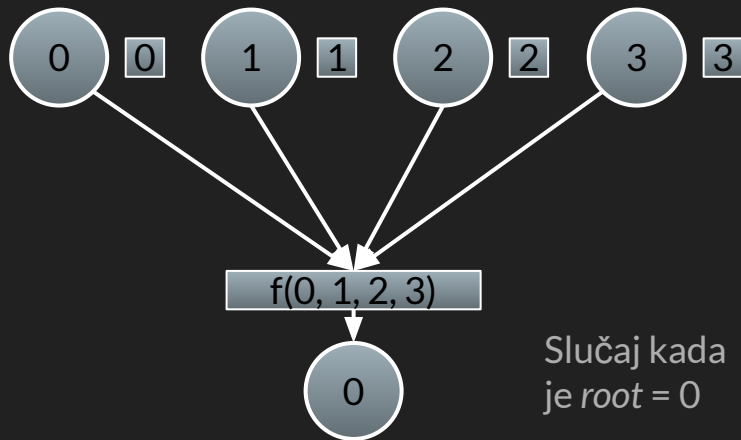
```
MPI_Allgather(&token, 1, MPI_INT, data, 1, MPI_INT, MPI_COMM_WORLD);
```

...

- *primeri/p08\_mpi\_allgather.c*

# Kolektivna komunikacija — *Reduce*

```
int MPI_Reduce(  
    const void *sendbuf,  
    void *recvbuf,  
    int count,  
    MPI_Datatype datatype,  
    MPI_Op op,  
    int root,  
    MPI_Comm comm);
```



- Svi procesi iz komunikatora *comm* šalju po *count* podataka tipa *datatype* procesu *root*, počev od memorijske lokacije *sendbuf*, a *root* prihvata podatke i pretvara ih u jedan podatak tipa *datatype* operacijom *op*, smeštajući rezultat na memorijsku lokaciju *recvbuf*.

# Kolektivna komunikacija — *Reduce* primer

```
int rank, size, root = 0;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

int token = rank, result;
MPI_Reduce(&token, &result, 1, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD);
printf("Proces %d: result = %d.\n", rank, result);
```

- *primeri/p09\_mpi\_reduce.c*
- Primer ispisa za 4 procesa:

Proces 2: result = 4096.

Proces 3: result = 4096.

Proces 0: result = 6.

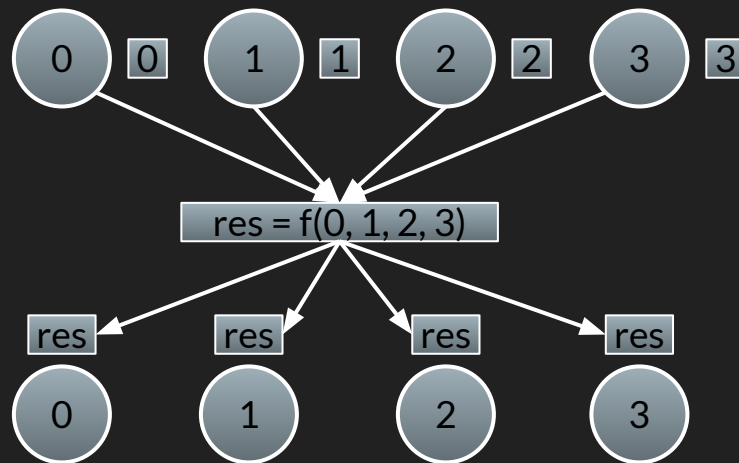
Proces 1: result = 4096.

## Zadatak 9 — Računanje srednje vrednosti 2

- Modifikovati zadatak 8 tako da se srednja vrednost izračuna korišćenjem *MPI\_Reduce* funkcije u odgovarajućem koraku.

# Kolektivna komunikacija — *Allreduce*

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



- Kao *MPI\_Reduce*, samo se rezultat na kraju nađe u *recvbuf* svih procesa iz komunikatora *comm*.

# Kolektivna komunikacija — *Allreduce* primer

```
int rank, size;  
MPI_Init(&argc, &argv);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
int token = rank, result;  
MPI_Allreduce(&token, &result, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);  
printf("Proces %d: result = %d.\n", rank, result);
```

- *primeri/p10\_mpi\_allreduce.c*
- Primer ispisa za 4 procesa:

Proces 2: result = 6.

Proces 3: result = 6.

Proces 0: result = 6.

Proces 1: result = 6.

Kolektivna komunikacija nad podskupom  
procesa

# Kolektivna komunikacija nad podskupom procesa

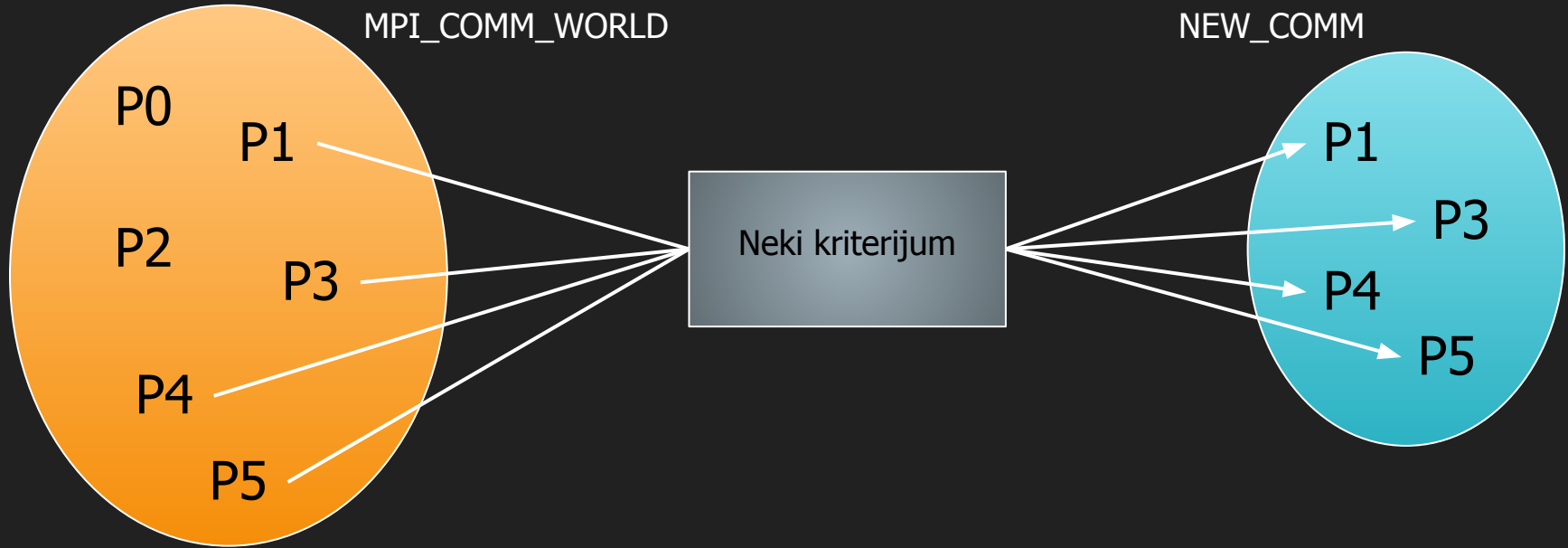
- U kolektivnoj komunikaciji učestvuju svi procesi unutar komunikatora.
- Nekad je potrebno poslati podatke samo delu procesa iz komunikatora.
- To je svakako moguće uraditi dobro uređenim pozivima *MPI\_Send* i *MPI\_Recv* funkcija.
- Kako poslati podatak samo delu procesa korišćenjem kolektivne komunikacije?



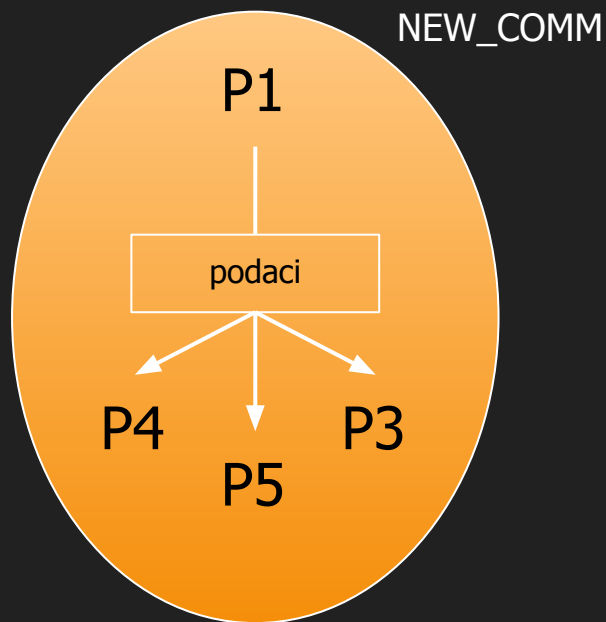
# Kolektivna komunikacija nad podskupom procesa

- U kolektivnoj komunikaciji učestvuju svi procesi unutar komunikatora.
- Nekad je potrebno poslati podatke samo delu procesa iz komunikatora.
- To je svakako moguće uraditi dobro uređenim pozivima *MPI\_Send* i *MPI\_Recv* funkcija.
- Kako poslati podatak samo delu procesa korišćenjem kolektivne komunikacije?
- Rešenje:
  - Napraviti novi komunikator za procese kojima treba poslati podatke i koristiti kolektivnu komunikaciju na nivou novonapravljenog komunikatora.

# Kolektivna komunikacija nad podskupom procesa



# Kolektivna komunikacija nad podskupom procesa



# Kreiranje proizvoljnog komunikatora

- Za kreiranje proizvoljnog komunikatora, potrebno je prvo izdvojiti odgovarajuće procese u **grupu**, pa na osnovu nje kreirati novi komunikator.
- Čest skup koraka za kreiranje novog komunikatora je sledeći:
  1. napraviti grupu od procesa iz nekog već postojećeg komunikatora (neretko *MPI\_COMM\_WORLD*),
  2. iz grupe dobijene u koraku 1, izvući podskup procesa u novu grupu na osnovu nekog kriterijuma, u jednom, ili u više koraka, i
  3. kreirati nov komunikator od postojećeg, koristeći grupu dobijenu u koraku 2.

# Kreiranje proizvoljnog komunikatora

```
...
MPI_Group world_group;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);

int n = 5;
const int ranks[5] = {0, 1, 4, 6, 7};

MPI_Group new_group;
MPI_Group_incl(world_group, 5, ranks, &new_group);

MPI_Comm new_comm;
MPI_Comm_create_group(MPI_COMM_WORLD, new_group, 0, &new_comm);
...
```

- *primeri/p11\_mpi\_user\_comm\_1.c*

# Kreiranje proizvoljnog komunikatora

```
...
MPI_Group world_group;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);

int n = 5;
const int ranks[5] = {0, 1, 4, 6, 7};

MPI_Group new_group;
MPI_Group_incl(world_group, 5, ranks, &new_group);

MPI_Comm new_comm;
MPI_Comm_create_group(MPI_COMM_WORLD, new_group, 0, &new_comm);
...
```


1. korak – napraviti grupu (*world\_group*) od postojećeg komunikatora (*MPI\_COMM\_WORLD*)

- *primeri/p11\_mpi\_user\_comm\_1.c*

# Kreiranje proizvoljnog komunikatora

```
...  
MPI_Group world_group;  
MPI_Comm_group(MPI_COMM_WORLD, &world_group);  
  
int n = 5;  
const int ranks[5] = {0, 1, 4, 6, 7};  
  
MPI_Group new_group;  
MPI_Group_incl(world_group, 5, ranks, &new_group);  
  
MPI_Comm new_comm;  
MPI_Comm_create_group(MPI_COMM_WORLD, new_group, 0, &new_comm);  
...
```

2. korak – na osnovu nekog kriterijuma, izvući određene procese iz postojeće grupe (*world\_group*) i prebaciti ih u novu (*new\_group*)



- *primeri/p11\_mpi\_user\_comm\_1.c*

# Kreiranje proizvoljnog komunikatora


```
...
MPI_Group world_group;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);

int n = 5;
const int ranks[5] = {0, 1, 4, 6, 7};

MPI_Group new_group;
MPI_Group_incl(world_group, 5, ranks, &new_group);

MPI_Comm new_comm;
MPI_Comm_create_group(MPI_COMM_WORLD, new_group, 0, &new_comm);
...
```

2. korak – funkcija *MPI\_Group\_incl*  
objašnjena je na narednim slajdovima



- *primeri/p11\_mpi\_user\_comm\_1.c*



# Kreiranje proizvoljnog komunikatora

```
...
MPI_Group world_group;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);

int n = 5;
const int ranks[5] = {0, 1, 4, 6, 7};

MPI_Group new_group;
MPI_Group_incl(world_group, 5, ranks, &new_group);
```

```
MPI_Comm new_comm;
MPI_Comm_create_group(MPI_COMM_WORLD, new_group, 0, &new_comm);
...
```

3. korak – kreirati novi komunikator (*new\_comm*) od postojećeg (*MPI\_COMM\_WORLD*), koristeći prethodno kreiranu grupu (*new\_group*). Grupa se koristi da naznači koji procesi će se naći u novom komunikatoru.

- *primeri/p11\_mpi\_user\_comm\_1.c*

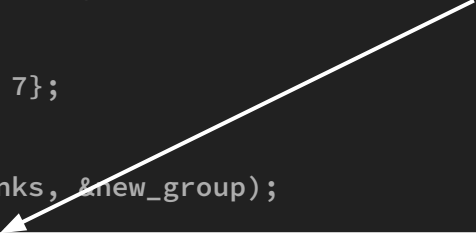
# Kreiranje proizvoljnog komunikatora

```
...
MPI_Group world_group;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);

int n = 5;
const int ranks[5] = {0, 1, 4, 6, 7};

MPI_Group new_group;
MPI_Group_incl(world_group, 5, ranks, &new_group);

MPI_Comm new_comm;
MPI_Comm_create_group(MPI_COMM_WORLD, new_group, 0, &new_comm);
...
```



3. korak – svi procesi iz grupe (*new\_group*) moraju pozvati ovu grupnu funkciju

- `primeri/p11_mpi_user_comm_1.c`

# Kreiranje proizvoljnog komunikatora

```
int new_comm_rank = -1, new_comm_size = -1;

if (MPI_COMM_NULL != new_comm) {
    MPI_Comm_rank(new_comm, &new_comm_rank);
    MPI_Comm_size(new_comm, &new_comm_size);
}

if (MPI_COMM_NULL != new_comm) {
    MPI_Bcast(&data, 1, MPI_INT, 0, new_comm);
}
```

- *primeri/p11\_mpi\_user\_comm\_1.c*

# Kreiranje proizvoljnog komunikatora

```
int new_comm_rank = -1, new_comm_size = -1;

if (MPI_COMM_NULL != new_comm) {
    MPI_Comm_rank(new_comm, &new_comm_rank);
    MPI_Comm_size(new_comm, &new_comm_size);
}

if (MPI_COMM_NULL != new_comm) {
    MPI_Bcast(&data, 1, MPI_INT, 0, new_comm);
}
```

procesi koji se ne nalaze u novom komunikatoru imaju `MPI_COMM_NULL` kao vrednost promenljive novog komunikatora

- `primeri/p11_mpi_user_comm_1.c`

# Kreiranje proizvoljnog komunikatora

```
int new_comm_rank = -1, new_comm_size = -1;

if (MPI_COMM_NULL != new_comm) {
    MPI_Comm_rank(new_comm, &new_comm_rank);
    MPI_Comm_size(new_comm, &new_comm_size);
}

if (MPI_COMM_NULL != new_comm) {
    MPI_Bcast(&data, 1, MPI_INT, 0, new_comm);
}
```

novi komunikator se može koristiti za grupnu komunikaciju; potrebno je da svi procesi iz novog komunikatora pozovu ovu funkciju

- *primeri/p11\_mpi\_user\_comm\_1.c*

# Kreiranje proizvoljnog komunikatora

```
MPI_Group_free(&world_group);
```

```
if (MPI_COMM_NULL != new_comm) {  
    MPI_Group_free(&new_group);  
    MPI_Comm_free(&new_comm);  
}
```

- *primeri/p11\_mpi\_user\_comm\_1.c*
- Na kraju je potrebno osloboditi iskorišćene resurse.
- Funkcije za oslobađanje grupa i komunikatora su grupne i moraju ih pozvati svi procesi koji pripadaju datoj grupi/komunikatoru.

# Funkcije za kreiranje nove grupe od postojeće

```
int MPI_Group_incl(  
    MPI_Group group,  
    int n,  
    const int ranks[],  
    MPI_Group *newgroup);
```

- `primeri/p11_mpi_user_comm_1.c`

- Nova grupa *newgroup* sadržaće *n* procesa iz grupe *group*, i to one čiji su rankovi u grupi *group* nalaze u nizu *ranks*.
- Rankovi procesa u grupi *newgroup* određeni su redosledom u nizu *ranks* – proces *ranks[i]* imaće rank *i* u *newgroup*.

# Funkcije za kreiranje nove grupe od postojeće

```
int MPI_Group_excl(  
    MPI_Group group,  
    int n,  
    const int ranks[],  
    MPI_Group *newgroup);
```

- `primeri/p11_mpi_user_comm_1.c`

- Nova grupa *newgroup* biće kreirana izbacivanjem *n* procesa iz grupe *group*, i to onih čiji su rankovi navedeni u nizu *ranks*.
- Rankovi procesa u grupi *newgroup* određeni su redosledom u grupi *group* – ako se, na primer, od 3 procesa izbaci srednji, onda će proces 2 iz *group* imati rank 1 u *newgroup*.



# Funkcije za kreiranje nove grupe od postojeće

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

- Nova grupa *newgroup* sadržaće procese koji se nalaze i u grupi *group1* i u grupi *group2*, odnosno njihov presek.
- Rankovi procesa u grupi *newgroup* određeni su redosledom procesa u *group1*.

- *primeri/p12\_mpi\_user\_comm\_2.c*

# Funkcije za kreiranje nove grupe od postojeće

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

- `primeri/p12_mpi_user_comm_2.c`

- Neka su *group1* i *group2* nastale od *world\_group* na sledeći način:

```
MPI_Group group_1, group_2;
int group_1_processes[3] = {0, 3, 2};
MPI_Group_incl(world_group, 3,
    group_1_processes, &group_1);

int group_2_processes[3] = {2, 1, 3};
MPI_Group_incl(world_group, 3,
    group_2_processes, &group_2);
```

- Presek su procesi 2 i 3, ali će proces 3 u *newgroup* imati rank 0, a proces 2 rank 1, jer se ređaju po redosledu iz *group1*.

# Funkcije za kreiranje nove grupe od postojeće

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

- Nova grupa *newgroup* sadržaće sve procese iz grupa *group1* i *group2*, bez duplikata.
- Rankovi procesa u grupi *newgroup* se redom dodeljuju prvo procesima iz grupe *group1*, pa zatim procesima iz *group2* kojih nema u *group1*.

- *primeri/p12\_mpi\_user\_comm\_2.c*

# Funkcije za kreiranje nove grupe od postojeće

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

- `primeri/p12_mpi_user_comm_2.c`

- Neka su *group1* i *group2* nastale od *world\_group* na sledeći način:

```
MPI_Group group_1, group_2;  
int group_1_processes[2] = {1, 2};  
MPI_Group_incl(world_group, 2,  
    group_1_processes, &group_1);
```

```
int group_2_processes[2] = {2, 0};  
MPI_Group_incl(world_group, 2,  
    group_2_processes, &group_2);
```

- Dodela rankova procesima iz *world\_group* ide redom počev od *group1*:
  - 1 – dobija rank 0
  - 2 – dobija rank 1
- Nakon toga se prelazi na *group2*, gde proces 0 dobija rank 2.

# Funkcije za kreiranje nove grupe od postojeće

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

- Nova grupa *newgroup* sadržaće sve procese iz grupe *group1* koji se ne nalaze u *group2*.
- Rankovi procesa u grupi *newgroup* se redom dodeljuju preostalim procesima po redosledu pojavljivanja u *group1*.

- *primeri/p12\_mpi\_user\_comm\_2.c*

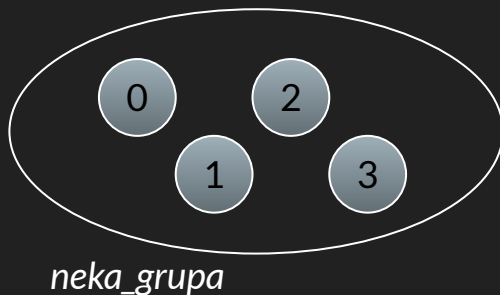
Koncept grupe

# Potreba za konceptom grupe

- U do sada viđenim primerima, čini se da je kreiranje grupe od početnog komunikatora, i potom pretvaranje nove grupe u nov komunikator, redundantno.
- Takođe, videli smo da i komunikator i grupa imaju svoju veličinu, da sadrže procese, i da vode evidenciju o njihovim rankovima.
- Čemu onda služi grupa, ukoliko je prilično nalik komunikatoru?

# Potreba za konceptom grupe

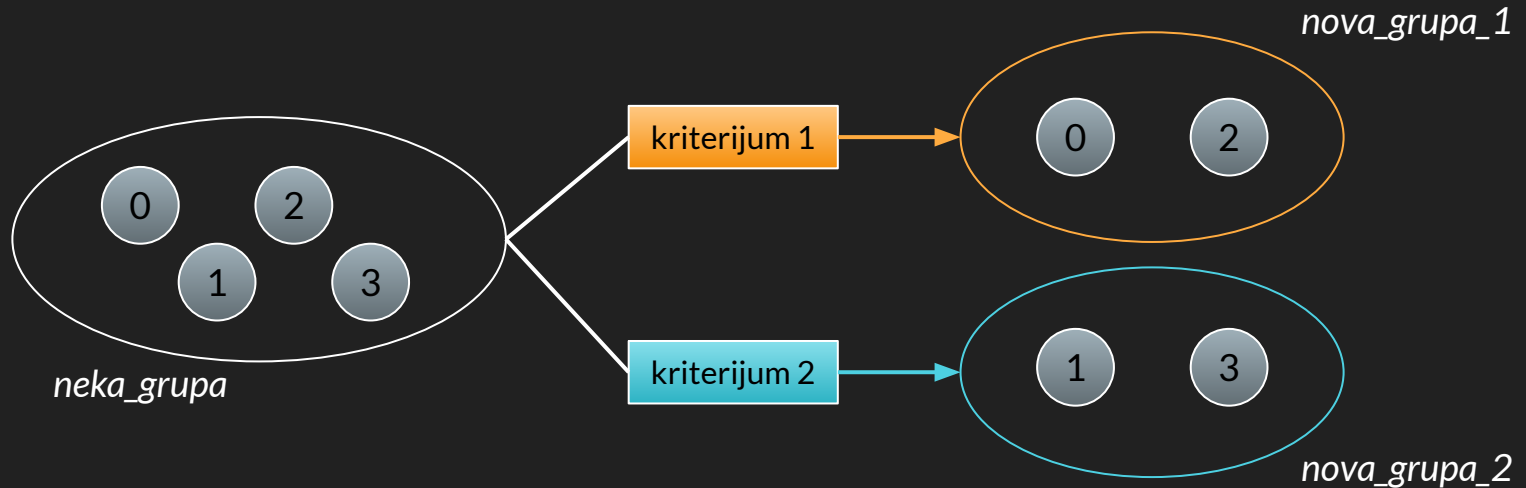
- Grupa je uređeni skup procesa, od kojih svaki sadrži jedinstveni identifikator, odnosno rank.





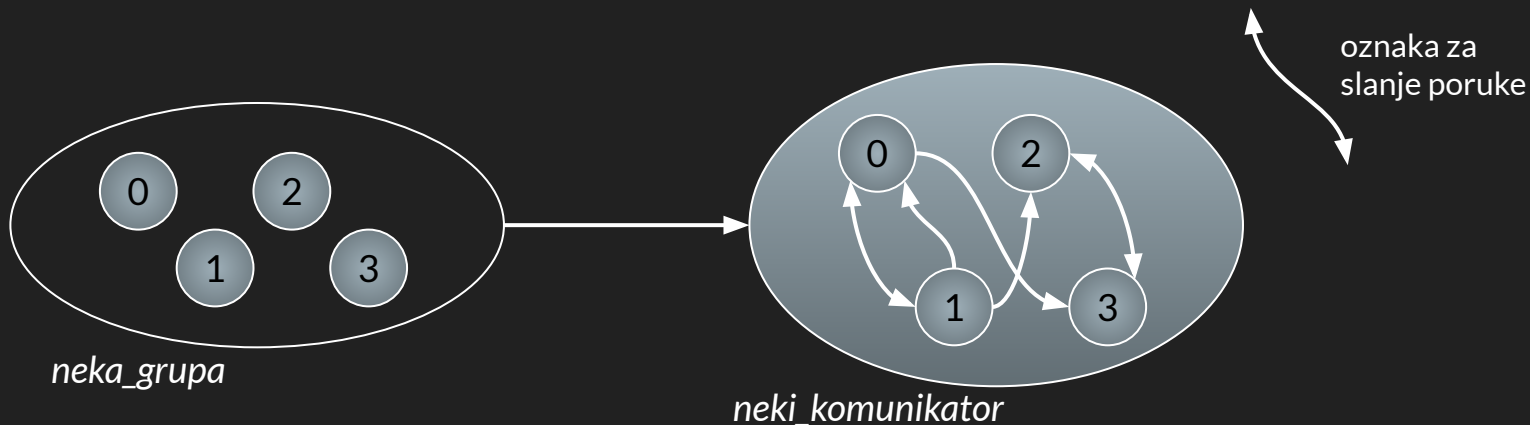
# Potreba za konceptom grupe

- Od jedne grupe, može se različitim kriterijumima dobiti više novih grupa.



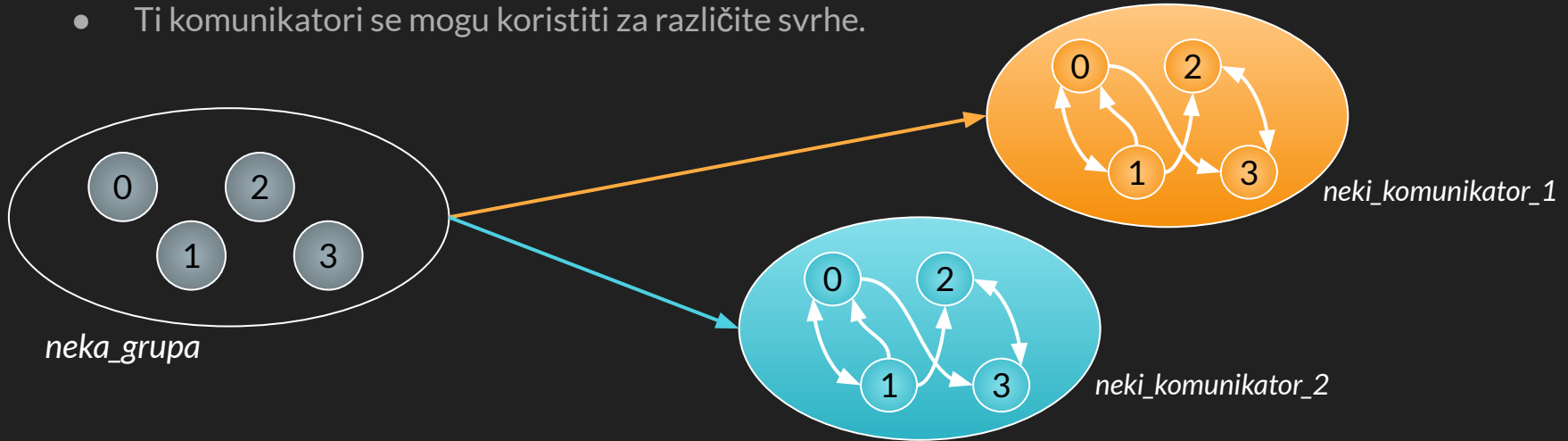
# Potreba za konceptom grupe

- Komunikator obuhvata grupu procesa koji međusobno mogu da komuniciraju.



# Potreba za konceptom grupe

- Nekada možda postoji potreba da se kreira više komunikatora od iste grupe.
- Ti komunikatori se mogu koristiti za različite svrhe.



# Potreba za konceptom grupe

- Komunikatori kakve smo do sada videli obuhvataju procese iz jedne grupe, makar ta grupa bila kreirana od neke druge dve.
- Ovakvi komunikatori se nazivaju intrakomunikatori.
- Moguće je napraviti komunikator koji obezbeđuje komunikaciju između dveju grupa.
- Ovakvi komunikatori nazivaju se interkomunikatori i mogu se kreirati funkcijom *MPI\_Intercomm\_create*.
- Interkomunikatori su izvan opsega ovih vežbi.