# Računarski sistemi visokih performansi

Nikola Vukić, Petar Trifunović, Veljko Petrović

Računarske vežbe
Zimski semestar 2024/25.

# *MPI* 4

# Sadržaj

- Promena veličine izvedenog tipa podatka.
- *MPI* U/I (*MPI I/O*):
  - model rada sa fajlovima u *MPI* aplikacijama
  - individualno čitanje i pisanje u fajl
  - pogled na fajl
  - kolektivne operacije za upis i čitanje
- Zadaci

# Promena veličine izvedenog tipa podatka

- Obrađene funkcije za kreiranje izvedenih tipova podatka definišu razmak između blokova koji čine jednu instancu tog tipa, ali ne i razmak između dve uzastopne instance.
- Podrazumevano, naredna instanca podatka počinje na prvom bajtu nakon prethodnog koji poštuje *padding* i poravnanje.
- Funkcija *MPI_Type_create_resized* se može iskoristiti za određivanje rastojanja između uzastopnih instanci *MPI* tipova.

# *MPI_Type_create_struct*

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

- Parametar *oldtype* predstavlja stari tip kom se menja veličina.
- Parametar *lb* predstavlja novu donju granicu tipa (često 0, jer se donja granica ne menja), dok *extent* predstavlja razmak između početaka dve uzastopne instance ovog tipa, izražen u bajtovima.

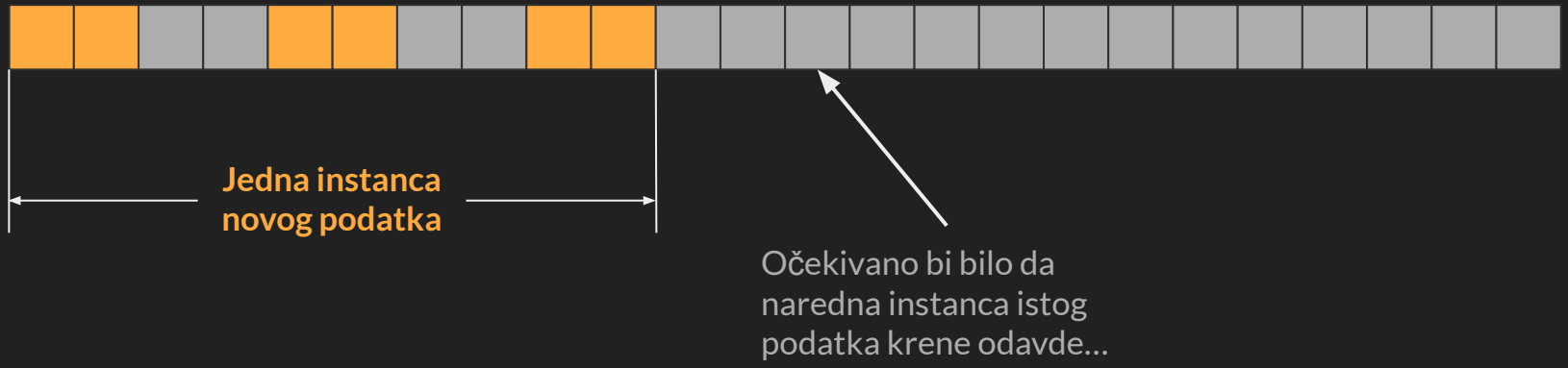
# *MPI_Type_create_struct*

- *MPI_Type_vector(3, 2, 4, MPI_INT, &vector_type)*


- Razmak između početaka uzastopnih blokva je 4 *MPI_INT* podatka.

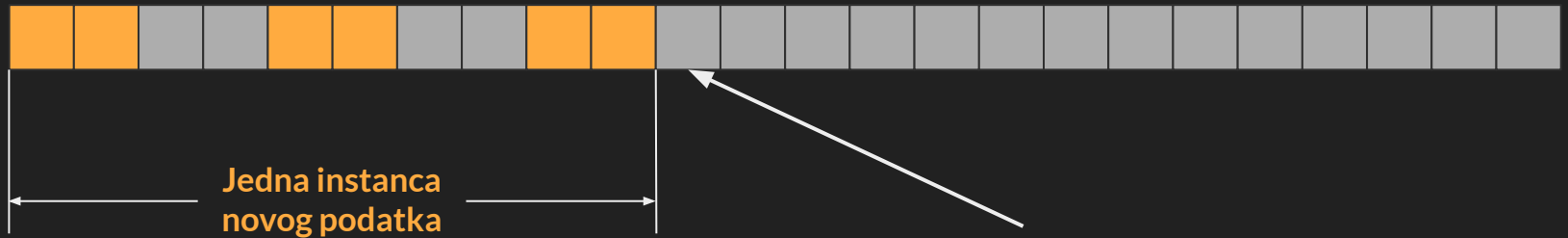
# *MPI_Type_create_struct*

- *MPI_Type_vector(3, 2, 4, MPI_INT, &vector_type)*



# *MPI_Type_create_struct*

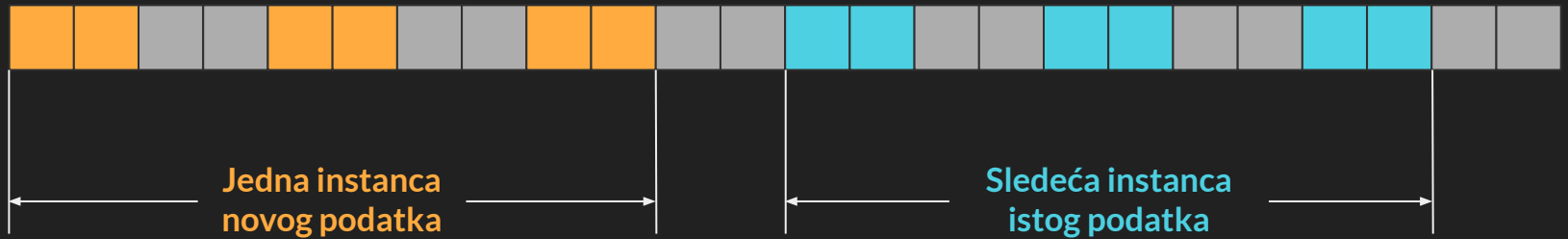
- *MPI_Type_vector(3, 2, 4, MPI_INT, &vector_type)*




…ali će zapravo krenuti odavde, jer se pri kreiranju tipa navodi razmak između internih blokova, a ne između blokova uzastopnih instanci

# *MPI_Type_create_struct*

- *MPI_Type_vector(3, 2, 4, MPI_INT, &vector_type)*




Umesto ovakvog slučaja…

# *MPI_Type_create_struct*

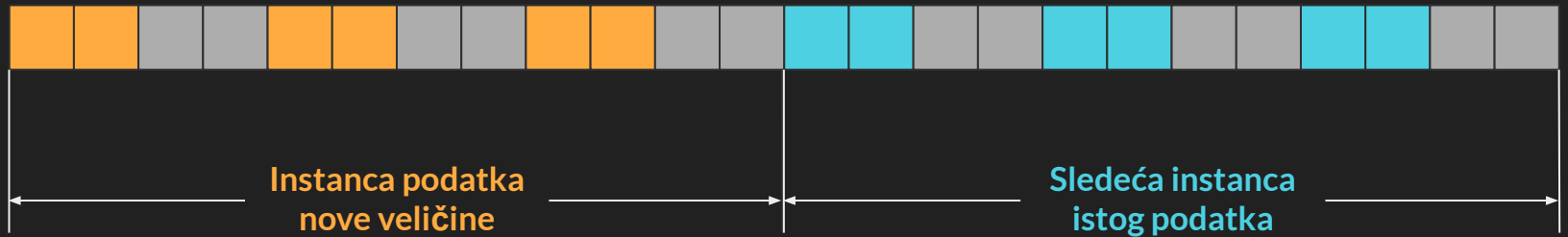
- *MPI_Type_vector(3, 2, 4, MPI_INT, &vector_type)*


…podaci će biti raspoređeni ovako

# *MPI_Type_create_struct*

- Željeni razmak se može definisati na sledeći način:

```
MPI_Type_vector(3, 2, 4, MPI_INT, &vector_type);
MPI_Type_create_resized(vector_type, 0, 4*3*sizeof(int), &vector_type_resized);
```



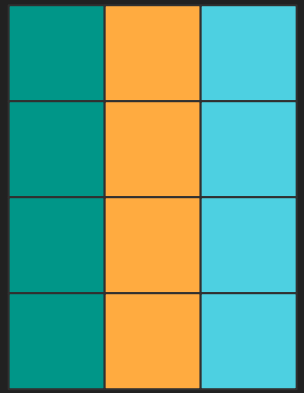

- Parametar *extent* podešen je na 12*sizeof(int), pa će sledeći podatak biti toliko bajtova udaljen od prethodnog.

# *MPI_Type_create_struct*

- Parametar *extent* se ne mora postaviti uvek na vrednost koja je veća od ukupnog memorijskog prostora koji obuhvata podatak kome se menja veličina.
- Razmotrimo situaciju sličnu primeru *primeri/p18_mpi_resize.c*…

# *MPI_Type_create_struct*

- Potrebno je po procesima rasporediti kolone linearizovane matrice.

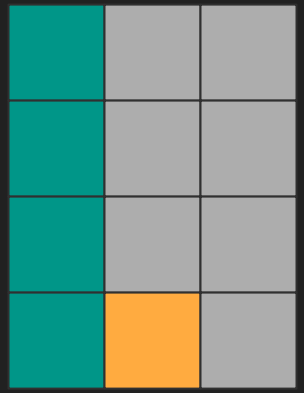


**kolona prvog procesa**
**kolona drugog procesa**
**kolona trećeg procesa**

Svakoj koloni matrice *NxM* odgovara tip podataka definisan kao:
`MPI_Type_vector(N, 1, M, &vector_type)`

# *MPI_Type_create_struct*

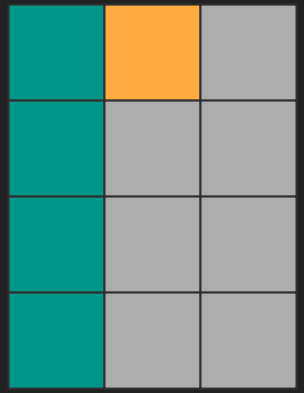
- Potrebno je po procesima rasporediti kolone linearizovane matrice.



- Već smo videli da se pri kreiranju vektora ne definiše razmak između dve njegove instance.
- Zato će sledeća instanca krenuti **sa pozicije** koja se nadovezuje na vektor prve kolone.

# *MPI_Type_create_struct*

- Potrebno je po procesima rasporediti kolone linearizovane matrice.

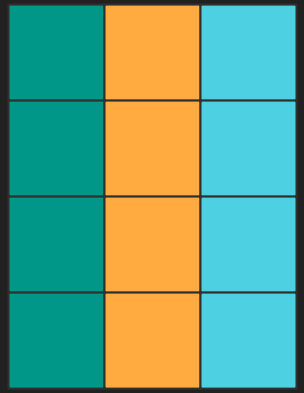


- Problem se može rešiti korišćenjem *MPI_Type_resized* funkcije, gde bi se kao *extent* prosledila vrednost manja od memorije koju obuhvata vektor.
- Ovde je potrebno da naredni podatak krene **sa pozicije** koja je udaljena za jedan podatak od početka vektora prve kolone:

```
MPI_Type_resized(vector_type, 0, sizeof(int), &vector_type_resized)
```

# *MPI_Type_create_struct*

- Potrebno je po procesima rasporediti kolone linearizovane matrice.

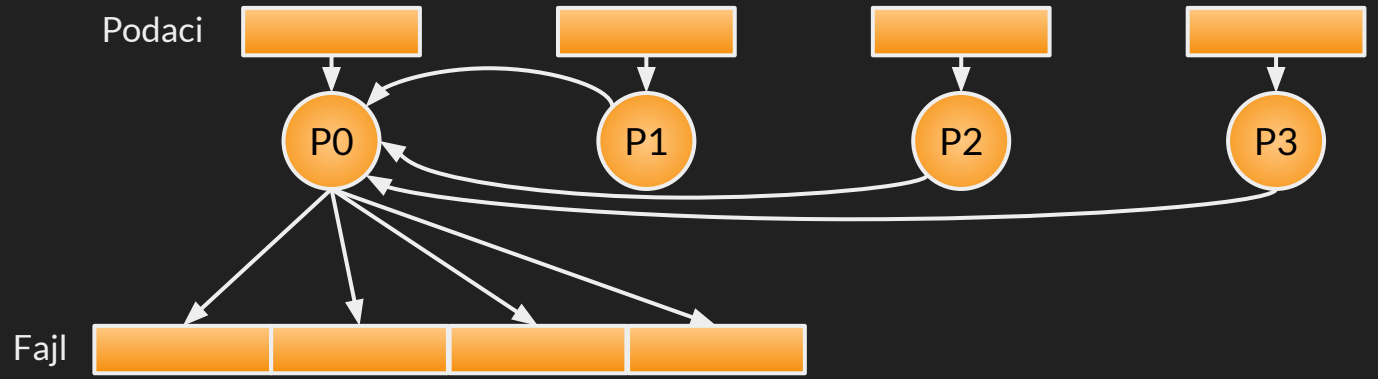


- Nakon promene veličine, svaki naredni vektor biće udaljen za jedan podatak od prethodnog, pa će kolone moći da se šalju jedna za drugom.

# *MPI paralelni U/I*
# *(MPI parallel I/O)*

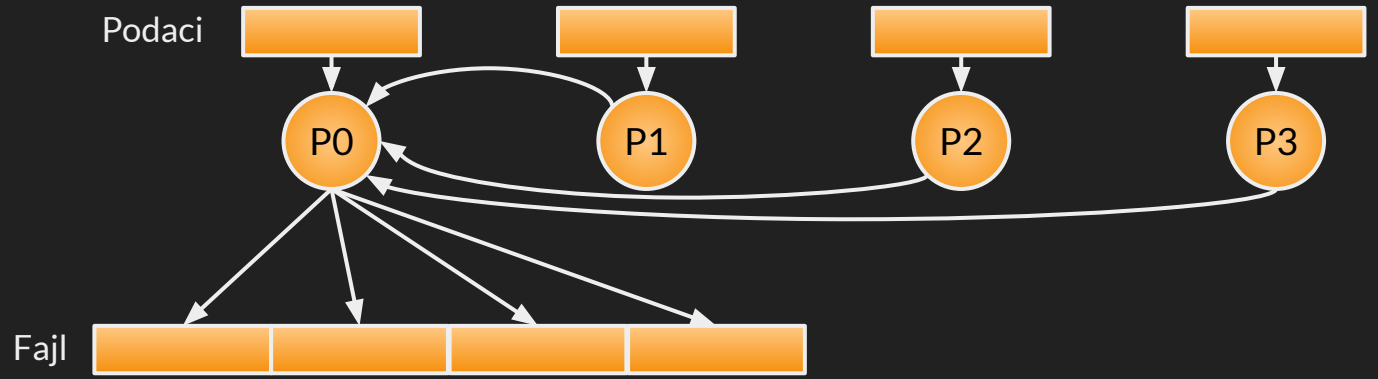
# Modeli rada sa fajlovima u paralelnim aplikacijama

- Sekvencijalan pristup fajlu od strane samo jednog procesa:
  - podaci nekoliko procesa se grupišu u jednom od njih, odakle se upisuju u fajl



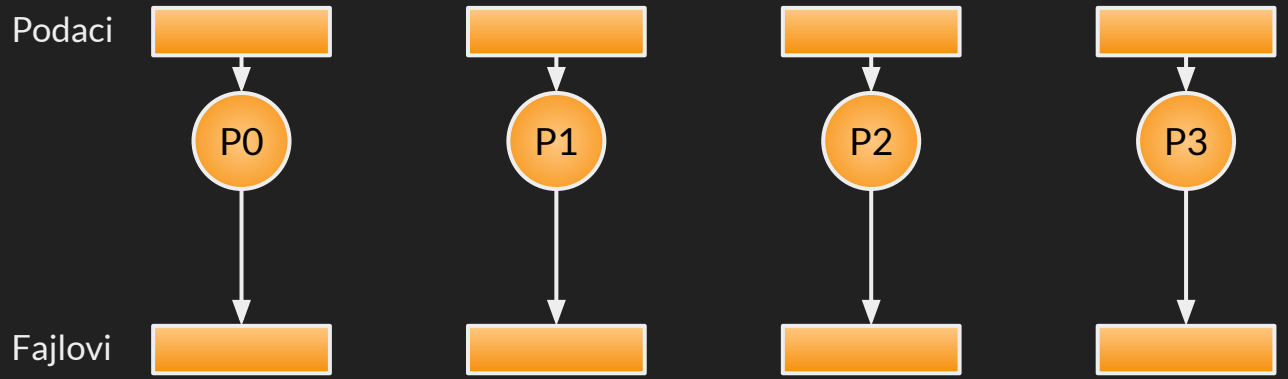
# Modeli rada sa fajlovima u paralelnim aplikacijama

- Sekvencijalan pristup fajlu od strane samo jednog procesa:
  - obezbeđuje jaku konzistentnost, ali proces upisivač je usko grlo



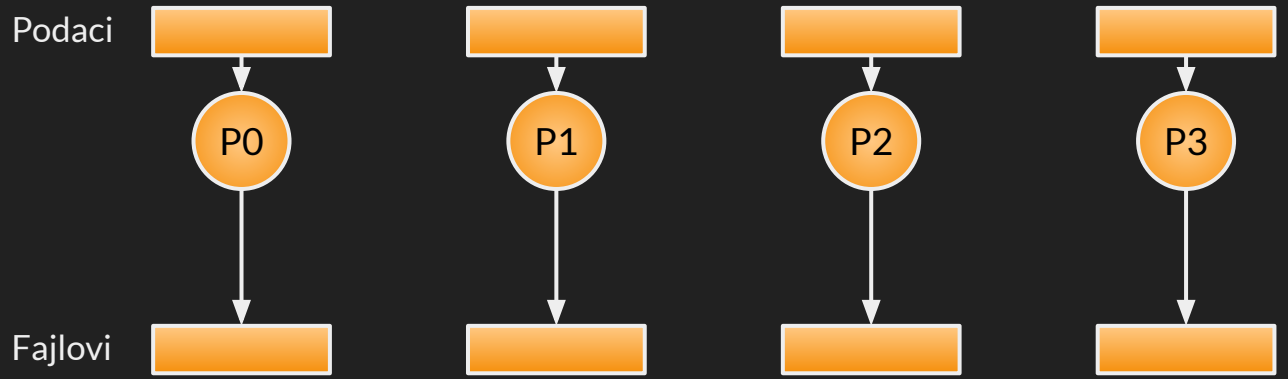
# Modeli rada sa fajlovima u paralelnim aplikacijama

- Paralelan pristup zasebnim fajlovima:
  - svaki proces potpuno nezavisno upisuje podatke u svoj fajl



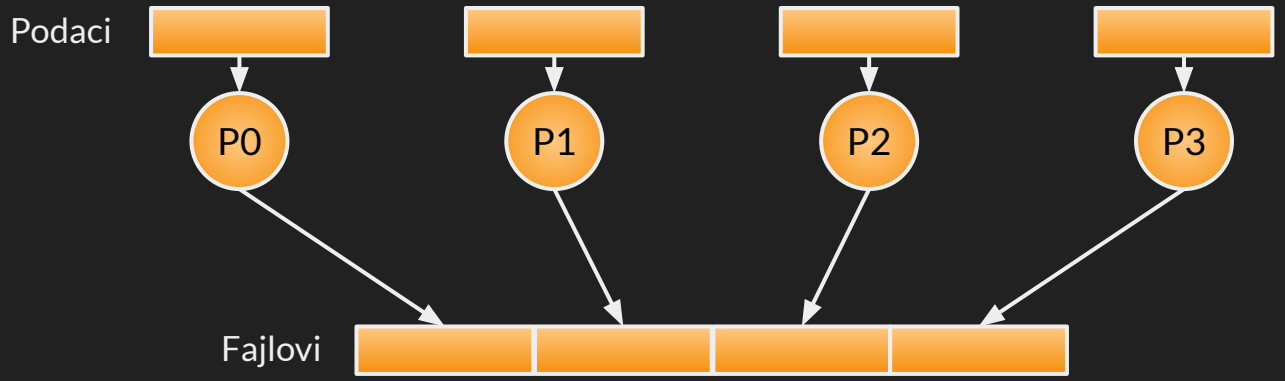
# Modeli rada sa fajlovima u paralelnim aplikacijama

- Paralelan pristup zasebnim fajlovima:
  - procesi mogu paralelno upisivati podatke, ali se mora voditi računa o više malih fajlova



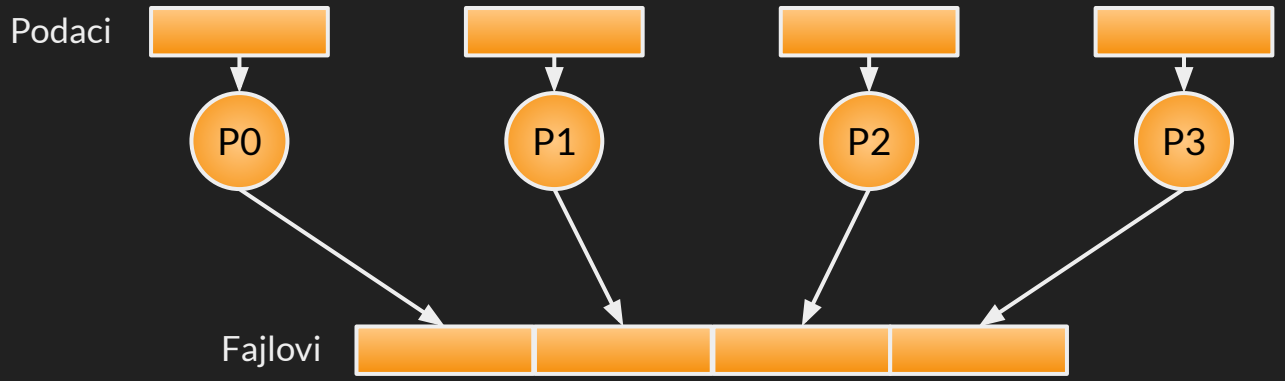
# Modeli rada sa fajlovima u paralelnim aplikacijama

- Paralelan pristup zajedničkom fajlu korišćenjem *MPI*-a:
  - logički gledano, procesi paralelno pristupaju jednom istom fajlu
  - *MPI* implementacija je zadužena za realan pristup delovima fajla na disku



# Modeli rada sa fajlovima u paralelnim aplikacijama

- Paralelan pristup zajedničkom fajlu korišćenjem *MPI*-a:
  - poboljšava performanse na račun konzistentnosti
  - omogućava grupisanje upisa, pristup proizvoljno udaljenim delovima fajla…



# Otvaranje i zatvaranje fajlova

```
int MPI_File_open(
    MPI_Comm comm,
    const char *filename,
    int amode,
    MPI_Info info,
    MPI_File *fh)
```

- **Kolektivna** funkcija koja otvara fajl po imenu *filename* u komunikatoru *comm* i vraća *handle* za otvoreni fajl preko parametra *fh*.
- Parametar *amode* označava režim otvaranja, a *info* prosleđuje dodatne opcije (*MPI_INFO_NULL* ukoliko nije od interesa).

```
int MPI_File_close(MPI_File *fh)
```

- **Kolektivna** funkcija koja sinhronizuje fajl *fh* u svim procesima (prenosi baferovane upise na disk; učini da svi procesi imaju konzistentan pogled na fajl), zatvara fajl i dealocira *handle fh*.

# Otvaranje i zatvaranje fajlova

- Neke od mogućih vrednosti parametra *amode*:
  - *MPI_MODE_CREATE*
  - *MPI_MODE_RDONLY*
  - *MPI_MODE_WRONLY*
  - *MPI_MODE_RDWR*
  - *MPI_MODE_APPEND*
  - ...
- Mogu da se kombinuju operatorom binarno *ili* (npr. *MPI_MODE_CREATE | MPI_MODE_WRONLY)*.
- Lista svih mogućih vrednosti dostupna je na linku: https://docs.open-mpi.org/en/v5.0.x/man-openmpi/man3/MPI_File_open.3.html

# Individualno čitanje i pisanje u fajl

```
int MPI_File_read(
    MPI_File fh,
    void *buf,
    int count,
    MPI_Datatype datatype,
    MPI_Status *status)
```

- Čita se *count* podataka tipa *datatype* iz fajla *fh* i podaci se smeštaju počev od memorijske lokacije *buf*.
- Broj podataka koji su zapravo pročitani smešta se u *_ucount* polje *status*-a, ali implementacija navodi da to nije deo standarda i da ovo polje treba izbegavati.
- Funkcija **nije kolektivna**.

```
int MPI_File_write(
    MPI_File fh,
    const void *buf,
    int count,
    MPI_Datatype datatype,
    MPI_Status *status)
```

- *Count* podataka tipa *datatype* koji počinju na memorijskoj lokaciji *buf* upisuje se u file *fh*.
- Broj podataka koji su se zapravo upisali se smešta u *_ucont* polje *status-a* (važi sve kao i za *MPI_File_read*).
- Funkcija **nije kolektivna**.

# Individualno čitanje i pisanje u fajl

```
int MPI_File_seek(
    MPI_File fh,
    MPI_Offset offset,
    int whence)
```

- Funkcija **nije kolektivna**.

- Postavlja individualni pokazivač pozicije u fajlu *fh*.
- Pokazivač se automatski pomera pri svakoj operaciji čitanja i pisanja za broj pročitanih ili upisanih podataka.
- Ovom funkcijom se pokazivač ručno postavlja na određenu poziciju u zavisnosti od vrednosti *whence* koja može biti:
  - *MPI_SEEK_SET* – pokazivač se postavlja na *offset*,
  - *MPI_SEEK_CUR* – na trenutnu poziciju pokazivača dodaje se *offset*, i
  - *MPI_SEEK_END* – pokazivač se postavlja na poziciju udaljenu za *offset* od kraja fajla.
- Vrednost *offset* može biti negativna (i mora biti, za *MPI_SEEK_END*).

# Individualno čitanje i pisanje u fajl

- Ukoliko se delovi fajla kojima procesi pristupaju pri upisu preklapaju, rezultat je nedefinisan.
- Zato se pre upisa može iskoristiti *MPI_File_seek* kako bi pokazivači svih procesa bili postavljeni na poziciju koja zavisi od ranka.
- Pogledati primer *primeri/p19_mpi_file_write.c* koji implementira individualni upis u fajl *filename*, i primer *primeri/p20_mpi_file_read.c* koji čita iz istog fajla.

# Individualno čitanje i pisanje u fajl

```
…
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
bufsize = FILESIZE / nprocs;
buf = (int *)malloc(bufsize);
nints = bufsize / sizeof(int);
…
MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
MPI_File_write(fh, buf, nints, MPI_INT, &status);
MPI_File_close(&fh);
…
```

svaki proces upisuje *nints* celobrojnih podataka, koji ukupno zauzimaju *bufsize* bajtova

# Individualno čitanje i pisanje u fajl

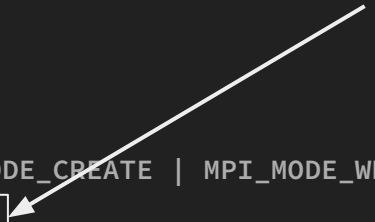
```
…
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
bufsize = FILESIZE / nprocs;
buf = (int *)malloc(bufsize);
nints = bufsize / sizeof(int);
…
MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
MPI_File_write(fh, buf, nints, MPI_INT, &status);
MPI_File_close(&fh);
…
```

otvara se fajl po imenu *filename*, koji se kreira ako ne postoji (*MPI_MODE_CREATE*), i koji će služiti samo za upis (*MPI_MODE_WRONLY*)

# Individualno čitanje i pisanje u fajl

```
…
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
bufsize = FILESIZE / nprocs;
buf = (int *)malloc(bufsize);
nints = bufsize / sizeof(int);
…
MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
MPI_File_write(fh, buf, nints, MPI_INT, &status);
MPI_File_close(&fh);
…
```

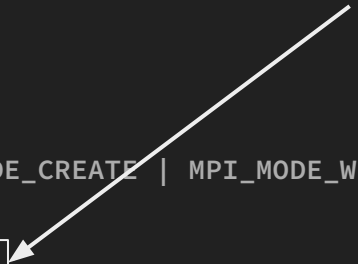
svaki proces postavlja svoj pokazivač na mesto određeno rankom, kako ne bi bilo preklapanja pri upisu



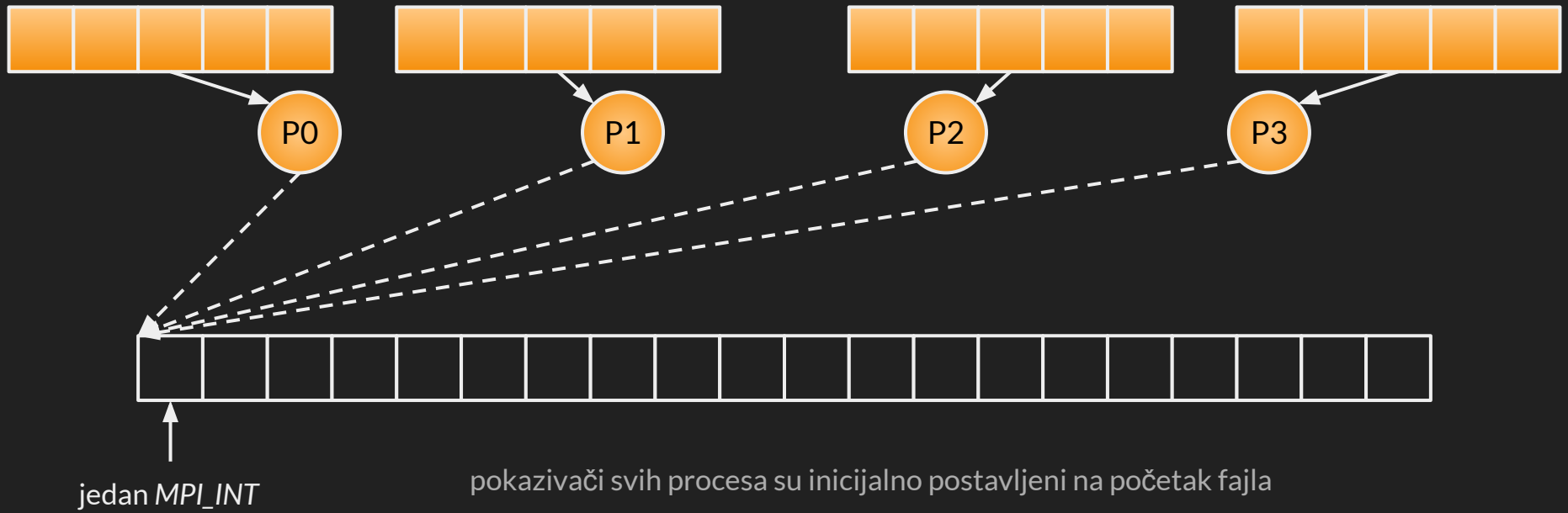
# Individualno čitanje i pisanje u fajl

```
…
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
bufsize = FILESIZE / nprocs;
buf = (int *)malloc(bufsize);
nints = bufsize / sizeof(int);
…
MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
MPI_File_write(fh, buf, nints, MPI_INT, &status);
MPI_File_close(&fh);
…
```

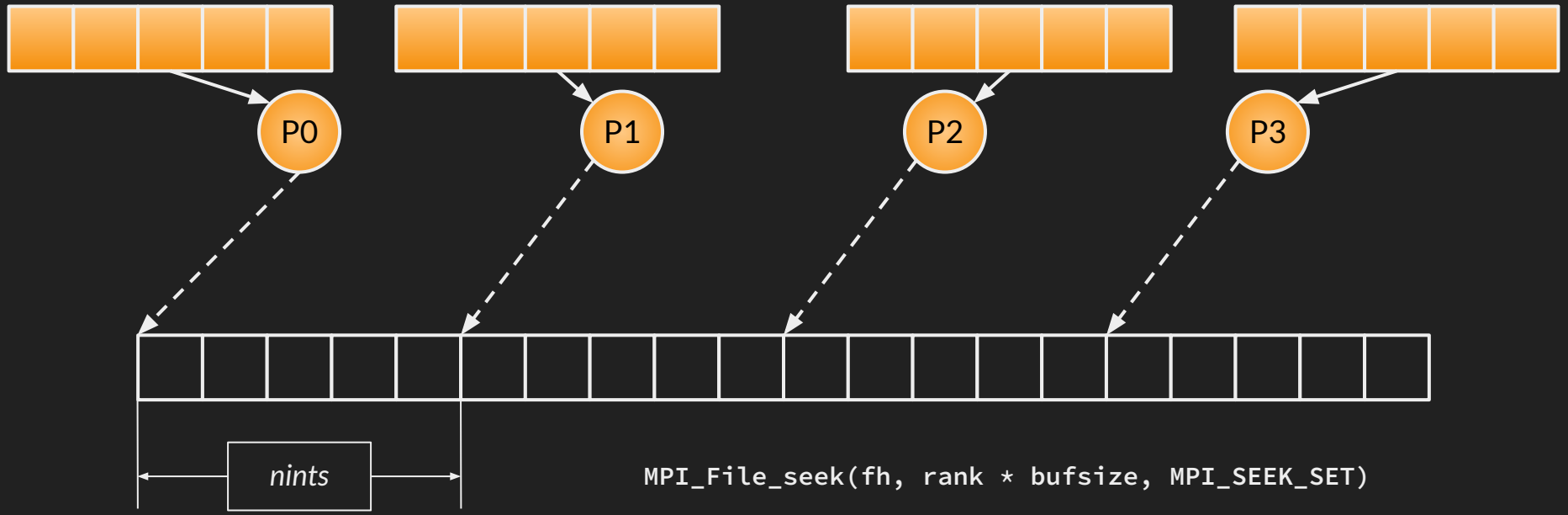
svaki proces počinje sa upisom od mesta postavljenog *MPI_File_seek* funkcijom i upisuje prethodno izračunat broj podataka



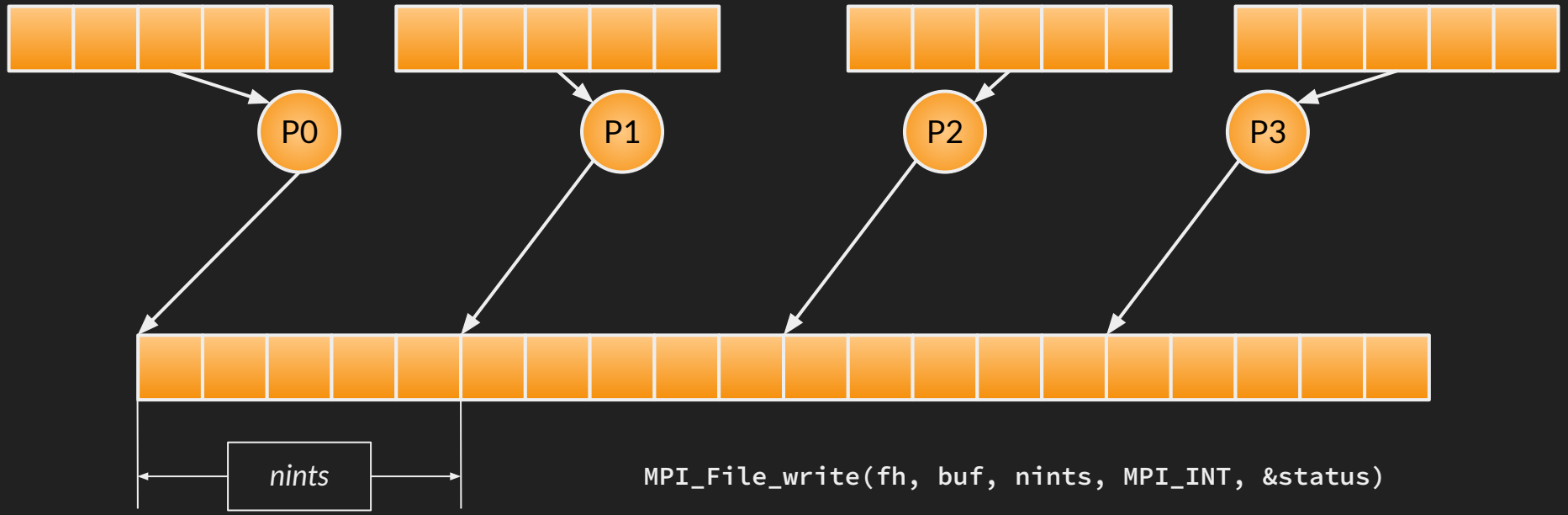
# Individualno čitanje i pisanje u fajl



# Individualno čitanje i pisanje u fajl



# Individualno čitanje i pisanje u fajl



# Individualno čitanje i pisanje u fajl uz pomeraj

```
int MPI_File_read_at(
     MPI_File fh,
     MPI_Offset offset,
     void *buf,
     int count,
     MPI_Datatype datatype,
     MPI_Status *status)
```

- Čita se *count* podataka tipa *datatype* iz fajla *fh*, počev od pozicije udaljene *offset* bajtova od početka fajla i podaci se smeštaju počev od memorijske lokacije *buf*.
- *status* – ista uloga kao i u *MPI_File_read*.
- Funkcija **nije kolektivna.**

```
int MPI_File_write_at(
     MPI_File fh,
     MPI_Offset offset,
     const void *buf,
     int count,
     MPI_Datatype datatype,
     MPI_Status *status)
```

- *Count* podataka tipa *datatype* koji počinju na memorijskoj lokaciji *buf* upisuje se u file *fh* počev od pozicije udaljene *offset* bajtova od početka fajla.
- *status* – ista uloga kao i u *MPI_File_write*.
- Funkcija **nije kolektivna**.

# Individualno čitanje i pisanje u fajl uz pomeraj

- Pogledati primer *primeri/p21_mpi_file_write_at.c* koji implementira individualni upis u fajl *filename*, i primer *primeri/p22_mpi_file_read_at.c* koji čita iz istog fajla.

# Pogled na fajl (engl. *file view*)

- *MPI* pruža mogućnost podešavanja različitih pogleda na fajl za različite procese.
- Pogled na fajl definiše poziciju od koje proces vidi fajl (podrazumevano je to početak fajla) kao i tip podatka u kojima proces posmatra fajl.
- Tip podatka može biti neki od osnovnih, ali može se definisati i izvedeni tip, što sa programerske strane omogućava da se uz mali broj *MPI* specifičnih instrukcija pristupa nekontinualnim blokovima unutar fajla.

# Pogled na fajl

```
int MPI_File_set_view(
    MPI_File fh,
    MPI_Offset disp,
    MPI_Datatype etype,
    MPI_Datatype filetype,
    const char *datarep,
    MPI_Info info)
```

- Funkcija je **kolektivna po parametru *fh*** (svi procesi koji su učestvovali u kolektivnom pozivu *MPI_File_open* koji je kreirao *handle fh*, moraju pozvati ovu funkciju).

- Kreira se pogled na fajl *fh*.
- Parametar *disp* označava udaljenost pogleda od početka fajla, u bajtovima.
- Parametar *etype* postaje novi osnovni tip pogleda na ovaj fajl (podrazumevano je to tip veličine jednog bajta). Pristup samom fajlu vrši se u jedinicama *etype*-a.
- Parametar *filetype* je tip podatka koji je ili isti kao *etype*, ili je izveden od njega.
- Parametar *datarep* naznačava da li se koristi ista reprezentacija podataka u memoriji i u fajlu. Važno kada se radi u heterogenim okruženjima. Vrednost “native” se prosleđuje za korišćenje identične reprezentacije u fajlu i u memoriji i koristi se za homogena okruženja.
- Parametar *info* prosleđuje dodatne pomoćne opcije.

# Pogled na fajl

definisanje novog
tipa za pogled na fajl

```
…
int mat_col[N];
MPI_Datatype vector_type;
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_Type_commit(&vector_type);

MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
MPI_File_write(fh, mat_col, N, MPI_INT, &status);
MPI_File_close(&fh);
…
```

- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
…
int mat_col[N];
MPI_Datatype vector_type;
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_Type_commit(&vector_type);

MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
MPI_File_write(fh, mat_col, N, MPI_INT, &status);
MPI_File_close(&fh);
…
```


- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
…
int mat_col[N];
MPI_Datatype vector_type;
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_Type_commit(&vector_type);

MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
MPI_File_write(fh, mat_col, N, MPI_INT, &status);
MPI_File_close(&fh);
…
```

kreiranje pogleda na fajl koji zavisi od ranka procesa i koristi izvedeni tip

- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
…
int mat_col[N];
MPI_Datatype vector_type;
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_Type_commit(&vector_type);

MPI_File_open(MPI_COMM_WORLD, "filename", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
MPI_File_write(fh, mat_col, N, MPI_INT, &status);
MPI_File_close(&fh);
…
```

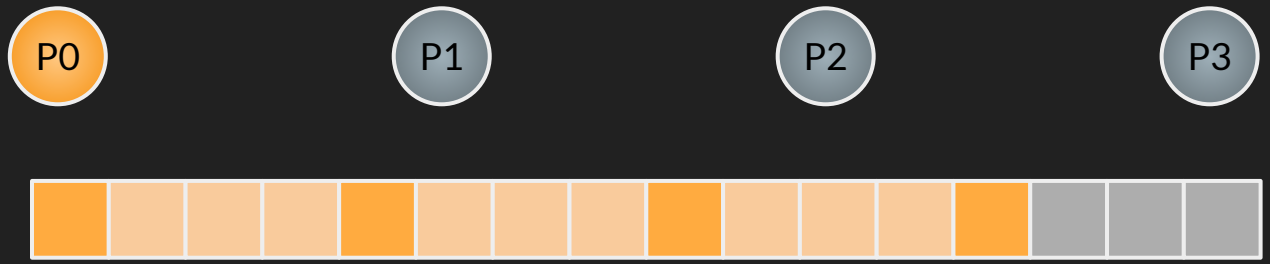
pri upisu u fajl može se koristiti tip podatka različit od onog korišćenog za stvaranje pogleda (slično kao što se pri slanju poruka može poslati jedan, a primiti potpuno drugačiji tip)

- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
```

- Ilustracija pogleda različitih procesa za *N* = 4 (svako polje je veličine *int*-a):

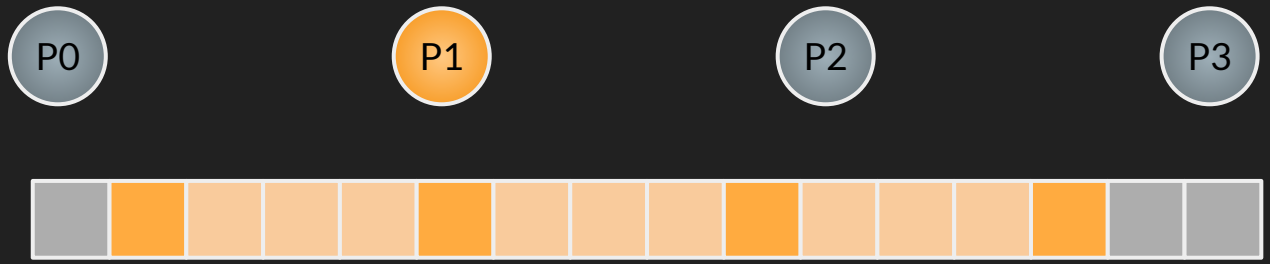



- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
```

- Ilustracija pogleda različitih procesa za *N* = 4 (svako polje je veličine *int*-a):

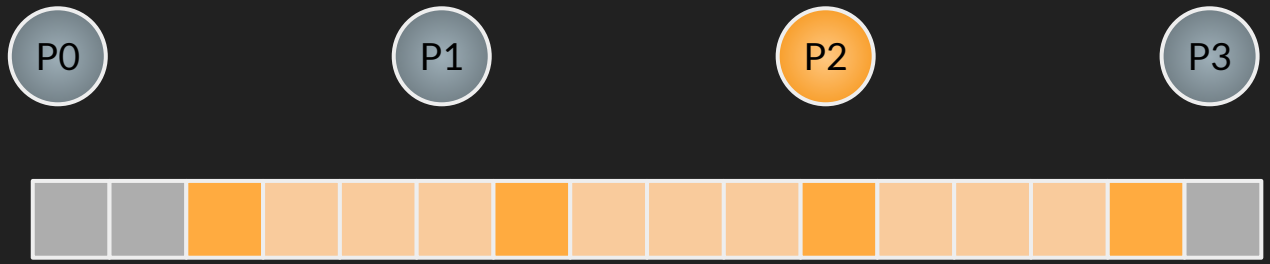



- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
```

- Ilustracija pogleda različitih procesa za *N* = 4 (svako polje je veličine *int*-a):

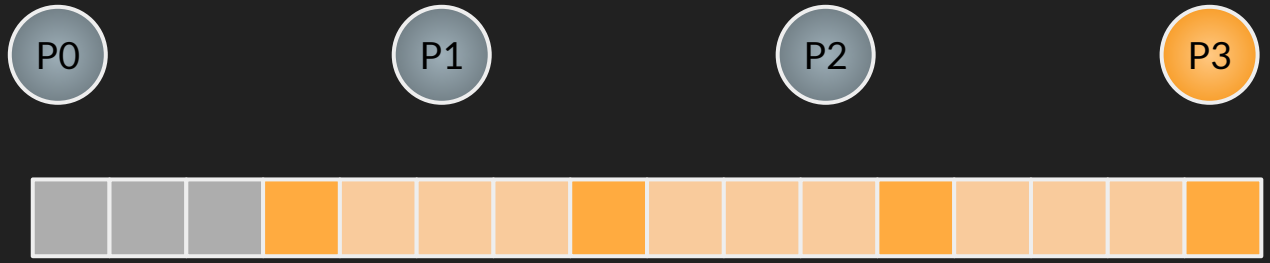



- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

```
MPI_Type_vector(N, 1, N, MPI_INT, &vector_type);
MPI_File_set_view(fh, rank * sizeof(int), MPI_INT, vector_type, "native", MPI_INFO_NULL);
```

- Ilustracija pogleda različitih procesa za *N* = 4 (svako polje je veličine *int*-a):




- *primeri/p23_mpi_view_write.c*

# Pogled na fajl

- Pokrenuti primer *primeri/p23_mpi_view_write.c* za upis u fajl *filename* korišćenjem pogleda na fajl, a zatim primer *primeri/p24_mpi_view_read.c* za čitanje iz istog fajla.
- Primer *p23* kreira poglede tako da svaki proces vidi po *N* blokova veličine jednog *MPI_INT* podatka, između kojih je rastojanje takođe *N MPI_INT*-ova. Ovime se praktično postiže da svaki proces pristupa nekontinualnim delovima fajla koji odgovaraju podacima koji čine jednu kolonu linearizovane matrice (pogledati prethodne ilustracije).
- U primeru *p24*, samo jedan proces pristupa fajlu (i zbog toga koristi *MPI_COMM_SELF* komunikator) kako bi pročitao uzastopne podatke iz fajla, i zatim ih ispisao, potvrđujući da je u fajl na ispravan način smeštena cela matrica, čiju je svaku kolonu upisao po jedan proces.

# Pogled na fajl

- Tip podatka (*filetype*) koji se koristi za kreiranje pogleda ne mora obuhvatiti čitav prostor koji će proces iskoristiti za upis.
- Parametar *filetype* definiše samo jedinicu pogleda, a ona se po potrebi replicira kada se pristupa fajlu.
- Ono što je važno jeste dobro definisati koliko je rastojanje između uzastopnih instanci korišćenog tipa, upotrebom funkcije *MPI_Type_create_resized*.

# Pogled na fajl

- Pristup fajlu iz primera *primeri/p23_mpi_view_write.c* implementiran je u primeru *primeri/p25_mpi_view_write_resized.c*.
- Tip je prvo kreiran kao:
  ```
  MPI_Type_contiguous(1, MPI_INT, &contig_type);
  ```
- Zatim mu je promenjena veličina:
  ```
  MPI_Type_create_resized(contig_type, 0, N * sizeof(int), &contig_type_resized);
  ```
- Ovime je kreiran kontinualan tip od jednog *MPI_INT*-a (nema previše smisla, ali je tu radi primera), a zatim je izmenjen tako da između dve instance ovog tipa bude *N* celih brojeva.
- Taj tip će se replicirati onoliko puta koliko je potrebno da bi se podržao upis svih podatak u fajl.

# Kolektivne operacije za upis i čitanje

```
int MPI_File_read_all(
    MPI_File fh,
    void *buf,
    int count,
    MPI_Datatype datatype,
    MPI_Status *status)
```

- Isto kao operacija za individualno čitanje, samo je **kolektivna po parametru *fh*** (svi procesi koji su učestvovali u kolektivnom pozivu *MPI_File_open* koji je kreirao *handle fh* moraju pozvati ovu funkciju).

```
int MPI_File_write_all(
    MPI_File fh,
    const void *buf,
    int count,
    MPI_Datatype datatype,
    MPI_Status *status)
```

- Isto kao operacija za individualan upis, samo je **kolektivna po parametru *fh*** (svi procesi koji su učestvovali u kolektivnom pozivu *MPI_File_open* koji je kreirao *handle fh* moraju pozvati ovu funkciju).

# Kolektivne operacije za upis i čitanje

- Pre korišćenja kolektivnog upisa i čitanja, treba postaviti poglede na fajlove tako da postoji planiran pristup delovima fajla od strane svakog procesa.
- Kolektivne operacije su performantnije od više individualnih, s obzirom na to da one samo izdaju zahtev za upisom i čitanjem, a *MPI* interni mehanizmi, zavisni od implementacije, vrše optimizovan pristup disku.
- Optimizacija se pretežno ogleda u mogućnosti grupisanja zahteva koji dolaze od procesa, tako da se obavlja manje pojedinačnih transakcija sa sekundarnom memorijom, i tako da u samoj komunikaciji sa diskom učestvuje samo podskup procesa kod kojih se nalazi grupisan zahtev.

# Dalje istraživanje

- Interesantan nastavak ove teme uključuje:
  - Eksplicitnu sinhronizaciju (*MPI_File_sync*):
    - *MPI* ne pruža visoku garanciju konzistentnosti, ali korisnik može iskoristiti funkciju *MPI_File_sync* za uvođenje tačaka konzistentnosti.
  - Neblokirajuć rad sa fajlovima:
    - većina viđenih funkcija imaju svoju neblokirajuću varijantu, npr. *MPI_File_iwrite*, *MPI_File_iread*…
  - Deljeni pokazivač na fajl:
    - pokazivač koji je zajednički svim procesima; funkcije sa nastavcima *shared* i *ordered*.
  - Split-collective operacije, korišćenje višedimenzionalnih tipova (*subarray*, *darray*) za rad sa fajlovima…

# Korišćena literatura

- *Online* materijale:
  - https://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture32.pdf
  - https://www.open-mpi.org/video/internals/Parallel_EdgarGabriel-2up.pdf
  - https://www.mpi-forum.org/docs/
- Materijali sa drugih kurseva:
  - prezentacije za predmet *Paralelni sistemi* na smeru *Računarstvo i informatika*, *Elektronski fakultet*, *Univerzitet u Nišu*, predavača Nađe Gavrilović i Natalije Stojanović

# Zadaci

# Zadatak 11

- Prepraviti primere *p23*, *p24* i *p25* tako da koriste kolektivne operacije za čitanje i pisanje u fajl.

# Zadatak 12

- Napisati *MPI C* program za upis u fajl:
  - Svaki proces ima sopstveni bafer čija je veličina deljiva sa dva (podesiti veličinu na proizvoljnu parnu vrednost).
  - Procesi jedan za drugim upisuju prve polovine svojih bafera u prvu polovinu fajla, tako da prvo proces 0 upiše svoju polovinu, pa nju sledi polovina procesa 1, pa procesa 2 i tako dalje.
  - Procesi pune drugu polovinu fajla drugim polovinama svojih bafera, poređanih po istom principu kao i prve polovine.
  - Kreirati pogled na fajl za svaki proces, tako da kreće od odgovarajućeg mesta u fajlu i koristi odgovarajuću izvedeni tip za upis.
  - Iskoristiti bilo koji tip operacije za upis u fajl.

# Zadatak 13

- Napisati *MPI C* program za čitanje iz fajla u koji je upisano u zadatku 12:
  - Prvi proces čitaće iz fajla prvu polovinu koju je upisao prvi proces, i drugu polovinu koju je upisao poslednji; drugi proces čitaće prvu polovinu koju je upisao drugi proces i drugu polovinu koju je upisao pretposlednji, i tako dalje.
  - Definisati odgovarajuće poglede i tipove podataka za ispravno čitanje iz fajla.
  - Iskoristiti bilo koji tip operacije za čitanje iz fajla.