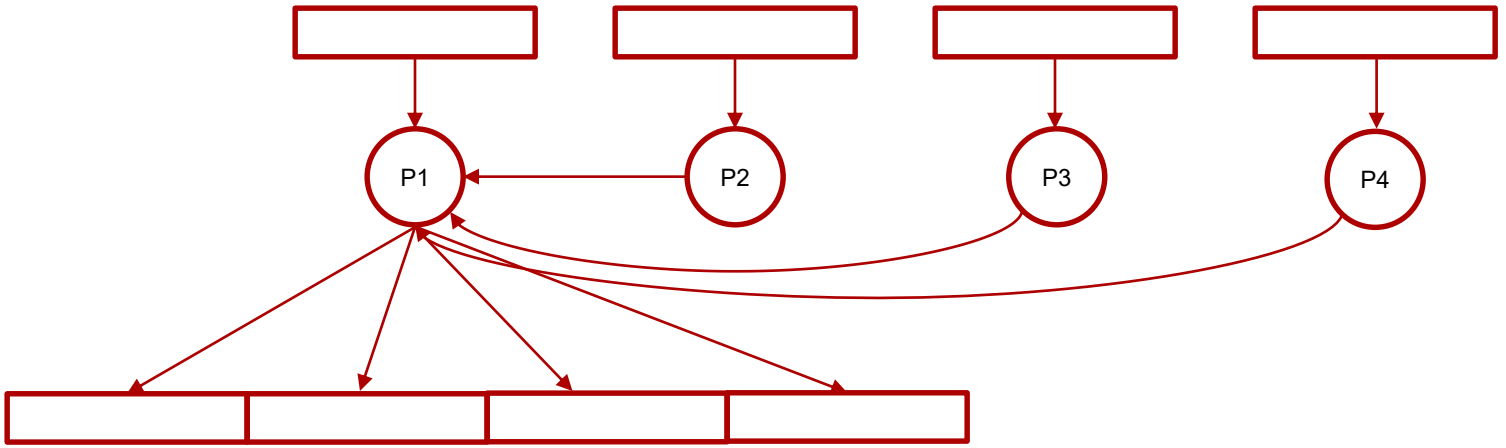
# MPI PARALLEL I/O

Mast. inž. Nađa Gavrilović
Prof. dr Natalija Stojanović

# Nekad i sad

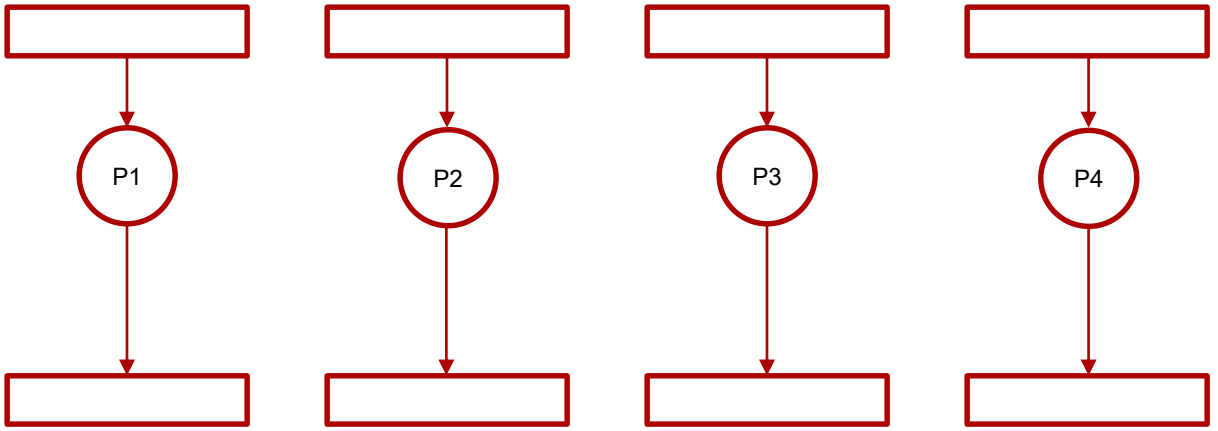
- Tradicionalni prilaz:
  1. Svi procesi šalju svoje podatke jednom procesu, koji sakuplja sve podatke (*gather*) i vrši upis u **jedan** fajl




- Nema paralelizma!
- Loše performanse
- Rešenje nije skalabilno
- Može se implementirati i bez MPI operacija

# Nekad i sad 2

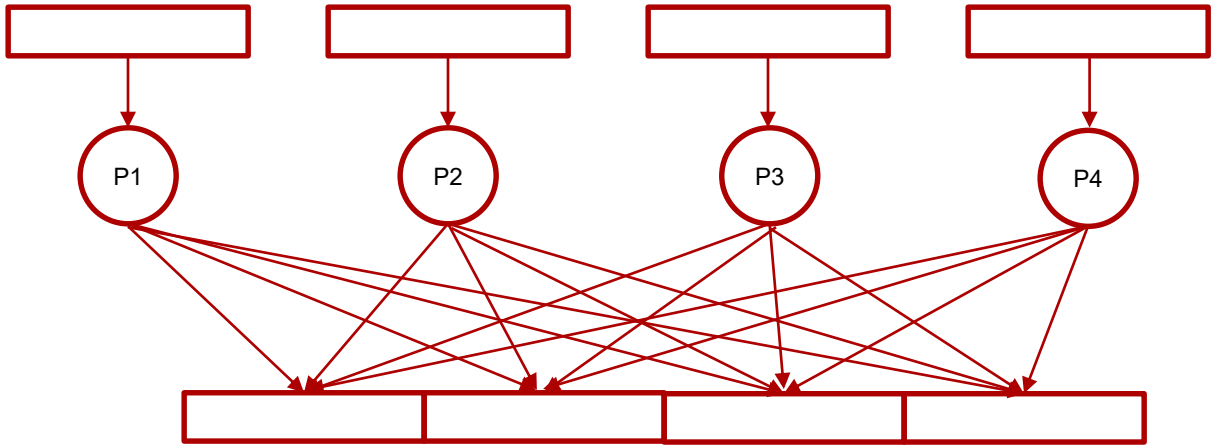
- Tradicionalni prilaz:
  2. Svi procesi vrše upis u poseban fajl




- Prednost - Paralelan upis
- Mana – Više manjih fajlova
- Može se implementirati i bez MPI operacija

# Nekad i sad 3

- Moderan (kooperativni) pristup:




  - Više procesa može vršiti upis u isti **fajl,** i to uz zadovoljavajuće performanse, uz odgovarajuću konfiguraciju
  - MPI omogućava takav način pristupa!

# Podsetnik

- File I/O u programskom jeziku C:
  - Otvaranje/zatvaranje fajla

    FILE fp = **fopen** ("file_name", "mode");

    **fclose** (file_pointer);
  - **Tekstualni** fajlovi
    - Formatirani upis/čitanje

      **fprintf**(FILE *stream, const char *format, ...)

      **fscanf**(FILE *stream, const char *format, ...)
    - Neformatirani upis/čitanje

      **fgets**(char *str, int n, FILE *stream)

      **fputs**(const char *str, FILE *stream)... ltd.
  - **Binarni** fajlovi

    **fread**(void *ptr, size_t size, size_t nmemb, FILE *stream)

    **fwrite**(const void *ptr, size_t size, size_t nmemb, FILE *stream)

    **fseek**(FILE *stream, long int offset, int whence)
- MPI I/O funkcije odnose se na rad sa **neformatiranim binarnim fajlovima**

# Osnovne MPI I/O funkcije

- Osnovne MPI I/O funkcije liče na standardne funkcije jezika C:
  - int **MPI_File_open**(MPI_Comm comm, const char *filename, int amode, MPI_Info info, MPI_File *fh)
  - int **MPI_File_seek**(MPI_File fh, MPI_Offset offset, int whence)
  - int **MPI_File_read**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
  - int **MPI_File_write**(MPI_File fh, const void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
  - int **MPI_File_close**(MPI_File *fh)

* Povratna vrednost svih funkcija je *error code.*

# Otvaranje i zatvaranje fajla

- Otvaranje fajla:
  - int **MPI_File_open**(MPI_Comm comm, const char *filename, int amode, MPI_Info info, MPI_File *fh)
    - Comm – komunikator koji ukazuje na grupu procesa koja pristupa fajlu
    - Filename – ime fajla
    - Amode – mod za otvaranje fajla

| MPI_File_open mode | Opis |
|---|---|
| MPI_MODE_RDONLY | Read only |
| MPI_MODE_RDWR | Read and Write |
| MPI_MODE_WRONLY | Write only |
| MPI _MODE_CREATE | Create file if it doesn't exist |

***Kombinovanje više modova vrši se pomoću logičkog ILI u C-u („|”)

    - Info – omogućava prosleđivanje dodatnih parametara - *hints* – koristimo MPI_INFO_NULL
    - Fh – povratna vrednost – pokazivač na otvoreni fajl
- Zatvaranje fajla
  - int **MPI_File_close**(MPI_File *fh)
    - Fh – pokazivač na fajl koji se zatvara

# Upis i čitanje

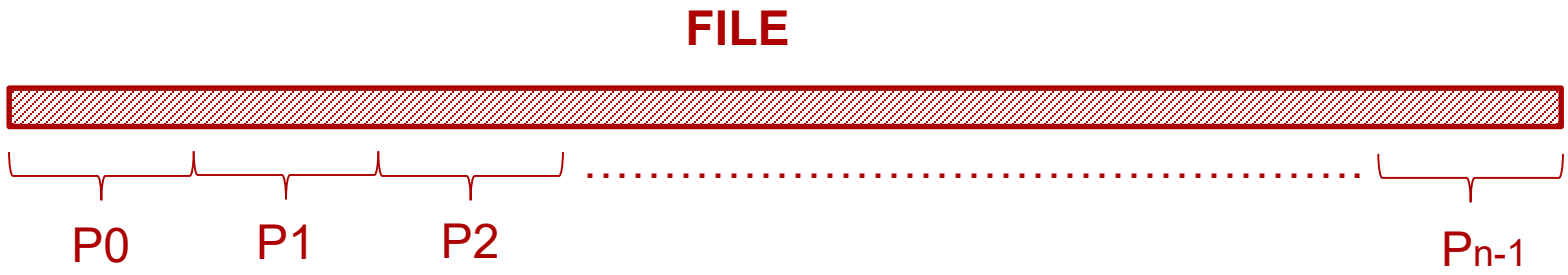
- Čitanje iz fajla
  - int **MPI_File_read**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
    - Fh - pokazivač na fajl
    - Bafer u memoriji u koji se upisuje/iz kog se čita
    - Count – broj podataka tipa datatype
    - Status – funkcija ista kao u slučaju MPI_Recv funkcije
- Upis u fajl
  - int **MPI_File_write**(MPI_File fh, const void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

# Pozicioniranje

- Pozicioniranje unutar fajla
- int **MPI_File_seek**(MPI_File fh, MPI_Offset offset, int whence)
  - Fh - pokazivač na fajl
  - Offset - Pomeraj unutar fajla, podrazumevano u bajtovima
  - Whence
    - MPI_SEEK_SET – Pointer se postavlja na vrednost pomeraja (od početka fajla)
    - MPI_SEEK_CUR – Pointer se postavlja na trenutnu poziciju pokazivača + pomeraj
    - MPI_SEEK_END - Pointer se postavlja na kraj fajla + pomeraj
- Primer:
  - **MPI_File_seek**(fh, 0, MPI_SEEK_SET); - Pozicioniranje na početak fajla
  - **MPI_File_seek**(fh, rank * bufsize , MPI_SEEK_SET); - Pozicioniranje u zavisnosti od ranga procesa

# Primer 1

- Ukupno N procesa
- Svaki proces čita po 1/n deo zajedničkog fajla, i to po rasporedu kao na slici:




- Rešenje upotrebom prethodno navedenih osnovnih MPI funkcija za rad sa fajlovima

# Primer 1 - rešenje

```
#include "mpi.h"
#define FILESIZE (1024*1024)
int main(int argc, char** argv)
{
    int* buf, rank, nprocs, nints, bufsize;
    MPI_File fh;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    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_RDONLY, MPI_INFO_NULL, &fh);
    MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
    MPI_File_read(fh, buf, nints, MPI_INT, &status);
    MPI_File_close(&fh);

    MPI_Finalize();
    return 0;
}
```

# Primer 1 + Upis u fajl

- U slučaju upisa u fajl, umesto čitanja u 1. primeru:
  - Koristi se MPI_File_write umesto MPI_File_read
  - Prilikom otvaranja fajla, Mod za otvaranje fajla postavlja se na **MPI_MODE_WRONLY ili MPI_MODE_RDWR i, po potrebi MPI_MODE_CREATE**
- Primer:
  - **MPI_File_open**(MPI_COMM_WORLD, "/pfs/datafile", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh)
  - **MPI_File_write**(fh, buf, nints, MPI_INT, &status)

# Primer 1 + upis u fajl - rešenje

```
#include "mpi.h"
#define FILESIZE (1024*1024)
int main(int argc, char** argv)
{
    int* buf, rank, nprocs, nints, bufsize;
    MPI_File fh;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    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);

    MPI_Finalize();
    return 0;
}
```

# MPI I/O funkcije - Nastavak

- Navedenih 5 funkcija može obezbediti rešavanje različitih I/O zahteva
- Ipak, u cilju poboljšanja efikasnosti, ali i pojednostavljenog pristupa nekontinualnim podacima, uvode se i druge „napredne“ MPI I/O funkcije
- Pravi benefiti I/O paralelizacije omogućavaju se upotrebom grupnih (collective) operacija, operacija za nekontinualni pristup itd.

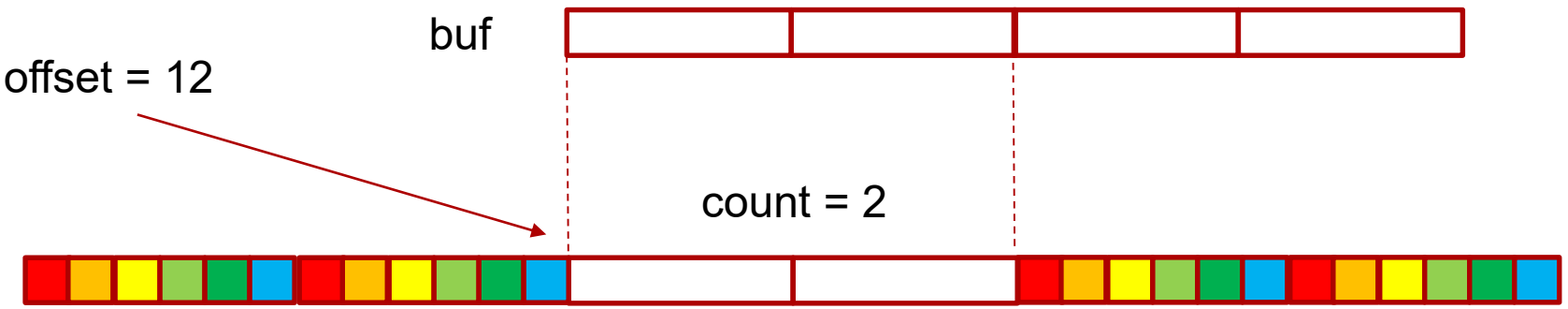
# Upotreba eksplicitnog pomeraja

- U prethodnom primeru korišćeni su individualni pointeri, koji koriste trenutnu lokaciju pokazivača kao mesto odakle se vrši čitanje/upis
- Drugi tip funkcija – funkcije sa eksplicitnim pomerajem (*explicit-offset functions*)
  - int **MPI_File_read_at**(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
  - int **MPI_File_write_at**(MPI_File fh, MPI_Offset offset, const void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
- Funkcijama se **direktno prosleđuje pomeraj (offset),** ostali parametri su isti kao i u MPI_File_read i MPI_File_write funkcijama

# Upotreba eksplicitnog pomeraja - primer

- Primer:

```
offset=12;
MPI_File_write_at(fh, offset, buf, 2, MPI_INT, &status);
```



# Primer 1 – rešenje + *explicit-offset*

```
#include "mpi.h"
#define FILESIZE (1024*1024)
int main(int argc, char** argv)
{
    int* buf, rank, nprocs, nints, bufsize;
    MPI_File fh;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    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_RDONLY, MPI_INFO_NULL, &fh);
    MPI_File_read_at(fh, rank * bufsize, buf, nints, MPI_INT, &status);
    MPI_File_close(&fh);

    MPI_Finalize();
    return 0;
}
```

# Nekontinualni pristup i grupne operacije

# Nekontinualni pristup i grupne operacije

- Prethodni primer zahtevao je da svaki proces pristupa kontinualnom delu podataka
- Mnoge paralelne aplikacije zahtevaju da svaki proces pristupi većem broju malih delova podataka, koji se nalaze u različitim delovima fajla
- Jedan način rešenja – veliki broj *MPI_File_read / MPI_File_write* poziva, za svaki poseban deo podataka u fajlu – **Ni malo efikasan pristup**!
- Rešenje – **pristup nekontinualnim podacima jednim pozivom funkcije**
- Dodatno, **grupne I/O operacije** obezbeđuju simultani pristup zajedničkom fajlu

# Nekontinualni pristup

- **File view -** Pogled na fajl
  - Definiše koji deo fajla je „vidljiv“ od strane procesa
  - Funkcije za čitanje/upis podataka mogu pristupati samo tom delu fajla (svi ostali podaci se preskaču)
  - Podrazumevano, pri prvom otvaranju fajla, proces može pristupiti celom fajlu. Inicijalno, pomeraj (offset) je postavljen na 0!

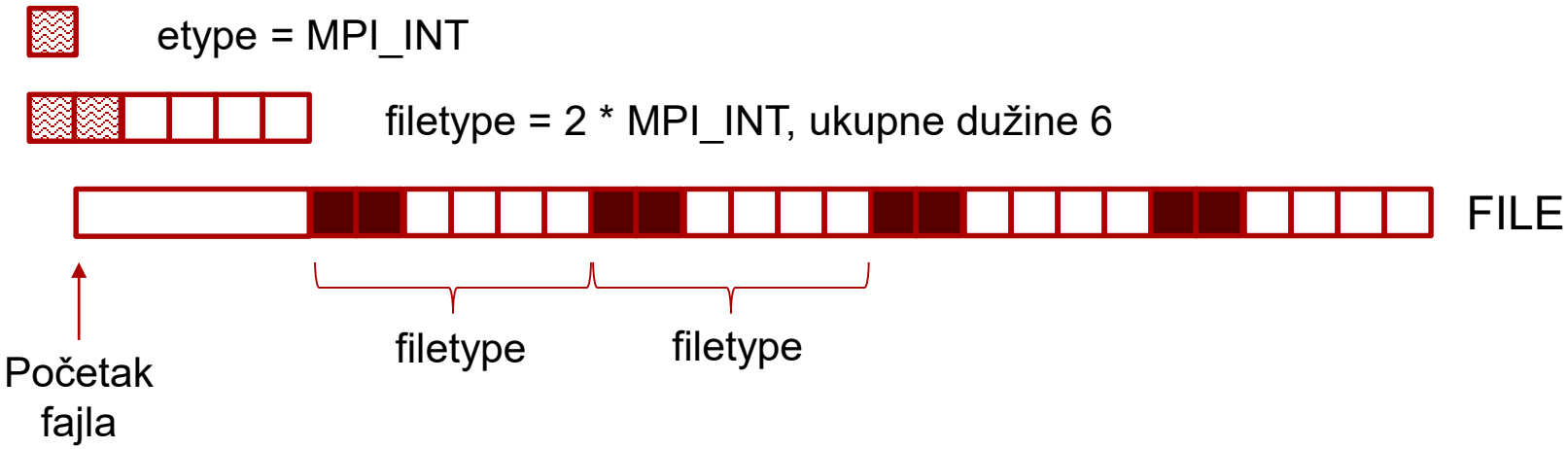
- Moguće je promeniti podrazumevani pogled na fajl! Razlozi:
  - Postavljanje konkretnog tipa podatka kome proces pristupa (integer, floating-point, umesto podrazumevanog pristupa bajtovima)
  - Podešavanje delova fajla koje treba preskočiti – tj. Specificiranje nekontinualnog pristupa

# File view

- Funkcija za postavljanje pogleda:
  - int **MPI_File_set_view**(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, const char * datarep, MPI_Info info)
    - Disp – broj **bajtova** koji se preskače od **početka** fajla
    - Etype – Tip podataka (može biti osnovni ili izvedeni tip). Predstavlja osnovnu jedinicu pristupa podacima. Svi pomeraji se specificiraju u odnosu na broj *etype* jedinica
    - Filetype – Tip podatka (može biti osnovni ili izvedeni tip). Mora biti ili istog tipa kao etype, ili **izvedeni tip koji se sastoji od više jedinica tipa *etype***
    - Datarep *(data representation)* – podrazumevana vrednost *native* – reprezentacija podataka u fajlu je ista kao u memoriji
    - Info – MPI_INFO_NULL (hints)
- Pogled na fajl počinje od zadatog pomeraja i sadrži više kontinualnih „kopija“ filetype-a
- Podrazumevani file view: disp = 0, filetype = MPI_BYTE
- Funkcija se može pozivati i pogled na fajl se može menjati više puta u toku programa

# File view - Primer

- Kreirati pogled koji definiše početni pomeraj od 5 integer-a, etype MPI_INT, i filetype koji sadrži dva integer-a, koje prati „praznina“ od 4 integer-a




- Pristupa se samo osenčenim delovima fajla sa slike, prazni/beli delovi fajla se preskaču

# File view – Primer - Rešenje

- Pre definisanja pogleda na fajl, neophodno je kreirati izvedeni tip podatka!

```
MPI_Type_contiguous(2, MPI_INT, &contig);
MPI_Type_create_resized(contig, 0, 6 * sizeof(int), &filetype);

MPI_Type_commit(&filetype);

MPI_File_set_view(fh, 5 * sizeof(int), etype, filetype, "native",
MPI_INFO_NULL);
```

# File view – Primer – Rešenje 2

- Pre definisanja pogleda na fajl, neophodno je kreirati izvedeni tip podatka!

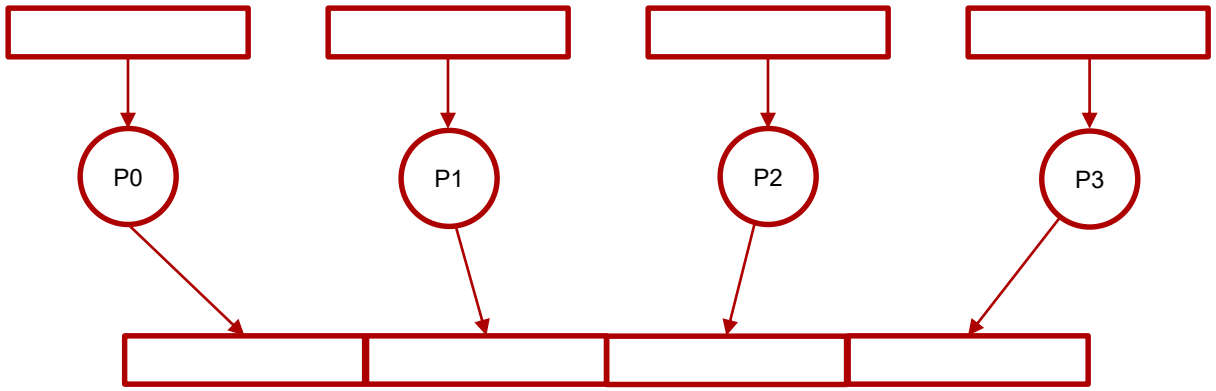
```
MPI_Type_vector(1, 2, 6, MPI_INT, &filetype);

MPI_Type_commit(&filetype);

MPI_File_set_view(fh, 5 * sizeof(int), etype, filetype, "native",
MPI_INFO_NULL);
```

# File view – Primer – Celo rešenje

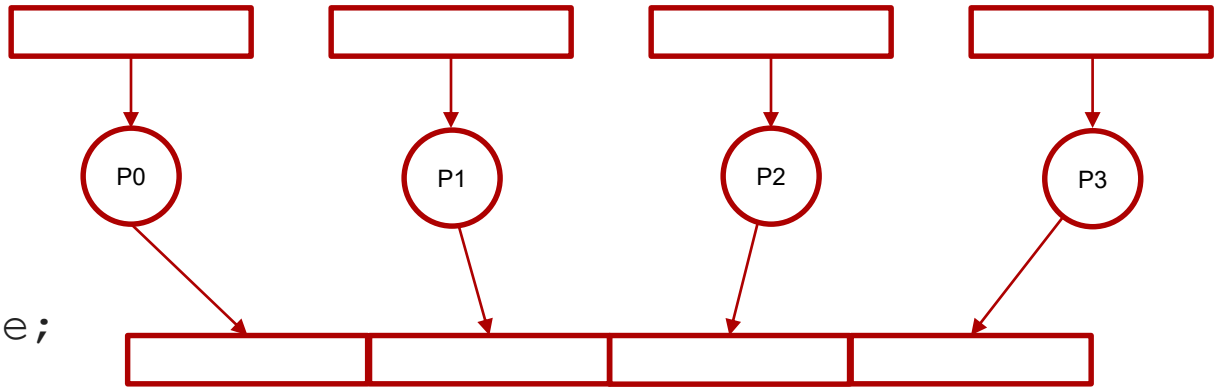
```
MPI_Aint lb, extent;
MPI_Datatype etype, filetype, contig;
MPI_Offset disp;
MPI_File fh;
int buf[1000];

MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
          MPI_MODE_CREATE | MPI_MODE_RDWR, MPI_INFO_NULL, &fh);

MPI_Type_contiguous(2, MPI_INT, &contig);
lb = 0;
extent = 6 * sizeof(int);
MPI_Type_create_resized(contig, lb, extent, &filetype);
MPI_Type_commit(&filetype);

disp = 5 * sizeof(int); /* assume displacement in this file
                           view is of size equal to 5 integers */
etype = MPI_INT;

MPI_File_set_view(fh, disp, etype, filetype, "native",
                  MPI_INFO_NULL);
MPI_File_write(fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);
```

# File view primer 2

- Svaki proces upisuje deo fajla, kao na slici (po jedan blok)
- Početak od kog svaki proces upisuje podatke se razlikuje!
- Svaki proces ima drugačiji **pogled na fajl**



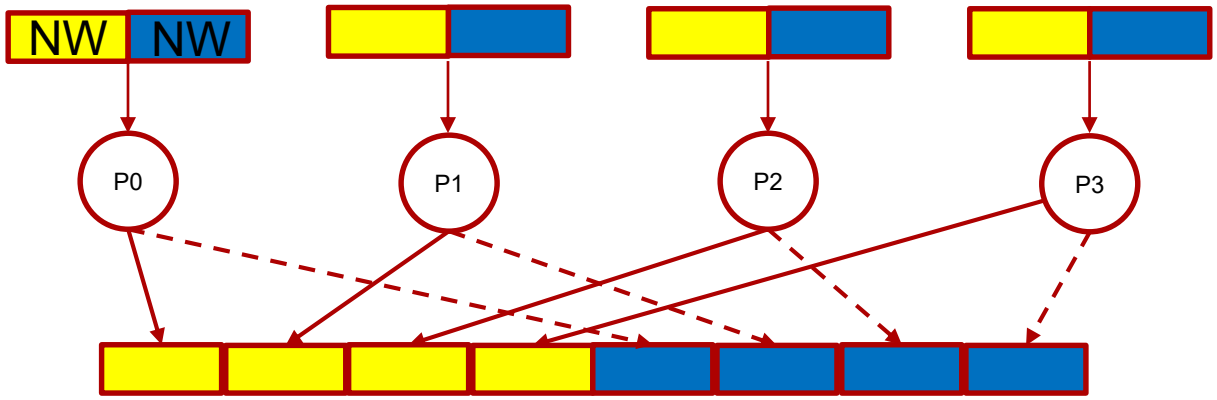
# File view primer 2 - rešenje



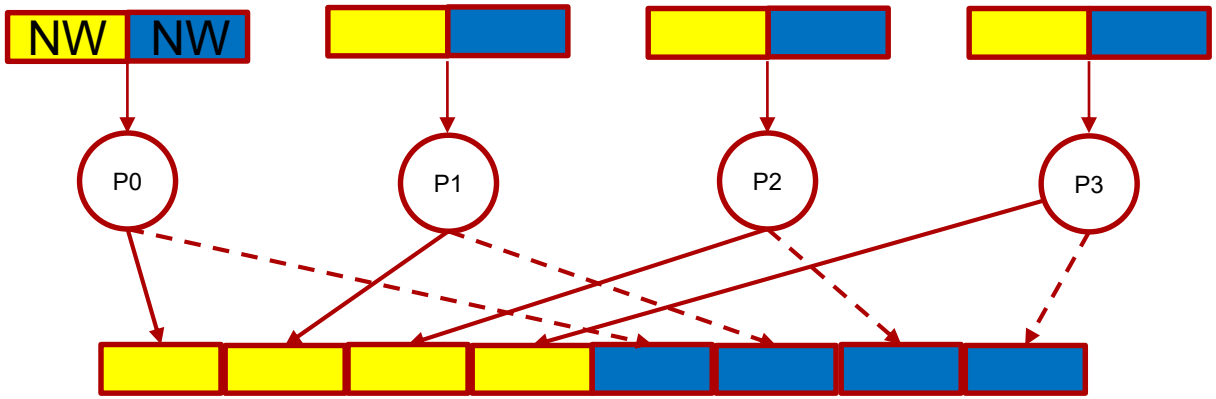

```
#define N 100
MPI_Datatype arraytype;
MPI_Offset disp;

disp = rank*sizeof(int)*N;
etype = MPI_INT;

MPI_Type_contiguous(N, MPI_INT, &arraytype);
MPI_Type_commit(&arraytype);
MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile", MPI_MODE_CREATE |
MPI_MODE_RDWR, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, disp, etype, arraytype, "native",
MPI_INFO_NULL);
MPI_File_write(fh, buf, N, etype, MPI_STATUS_IGNORE);
```

# File view primer 3

- Svaki proces upisuje svoje kontinualne podatke u dva različita bloka
- Početak od kog svaki proces upisuje podatke se razlikuje!
- Svaki proces ima drugačiji **pogled na fajl**
- Potrebno je definisati izvedeni tip podatka



# File view primer 3 - Rešenje




```
int buf[NW*2];
MPI_File_open(MPI_COMM_WORLD, "/data2", MPI_MODE_RDWR, MPI_INFO_NULL,
&fh);

/* definišemo pogled na 2 bloka od po NW integera, na NW*nprocs udaljenosti*/
MPI_Type_vector(2, NW, NW*nprocs, MPI_INT, &fileblk);
MPI_Type_commit(&fileblk);
disp = (MPI_Offset)rank*NW*sizeof(int);
MPI_File_set_view(fh, disp, MPI_INT, fileblk, "native", MPI_INFO_NULL);

/* Upis 2 'ablk' tipa, gde svaki ima po NW integera*/
MPI_Type_contiguous(NW, MPI_INT, &ablk);
MPI_Type_commit(&ablk);
MPI_File_write(fh, (void *)buf, 2, ablk, &status);
```