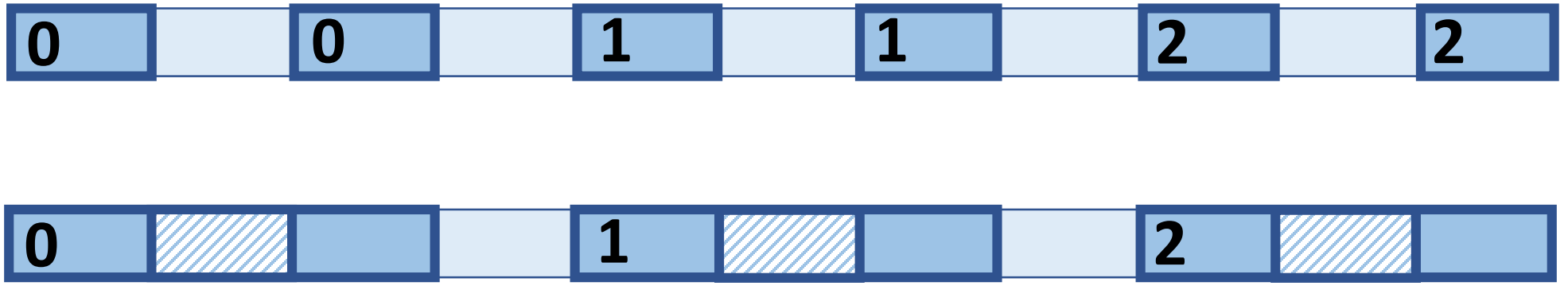
# Parallel I/O - II

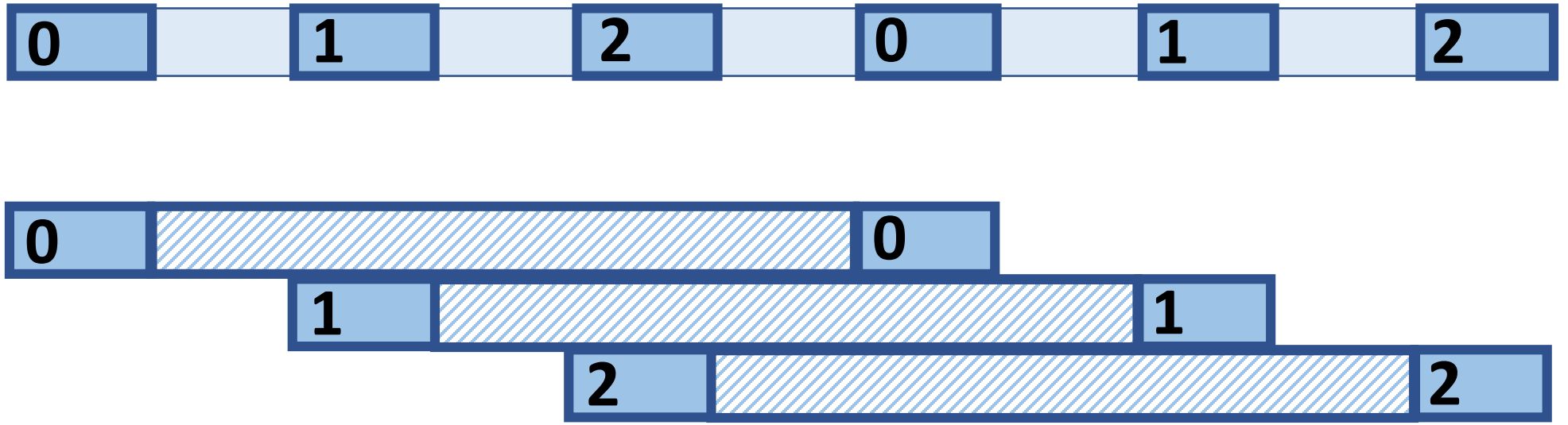
Apr 9, 2021

# Data Sieving




- Make large I/O requests and extract the data that is really needed
- Huge benefit of reading large, contiguous chunks

# Data Sieving – Interleaved data




Q: What is the problem here?

Solution – Lock the relevant portions in the file

# User-controlled MPI-IO Parameters

- ind_rd_buffer_size - Buffer size for data sieving for read
- ind_wr_buffer_size - Buffer size for data sieving for write
- romio_ds_read - Enable or not data sieving for read
- romio_ds_write - Enable or not data sieving for write

# MPI_Info – Example

```
MPI_Info_create (&info);

MPI_Info_set (info, "ind_rd_buffer_size", "2097152");

MPI_Info_set (info, "ind_wr_buffer_size", "1048576");

MPI_File_open (MPI_COMM_WORLD, filename, amode, info, &fh);
```

# Independent I/O – Summary

- Individual file pointers
  - Explicit offsets or not

MPI_File_read_at/MPI_File_read

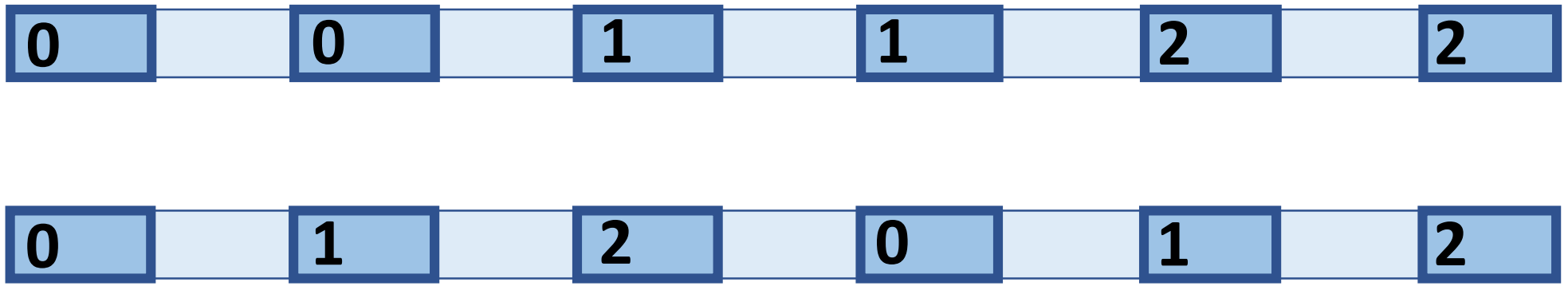
- Shared file pointers
  - Read/write starting from the current location of file pointer
  - All processes share the same file view

MPI_File_read_shared/MPI_File_write_shared

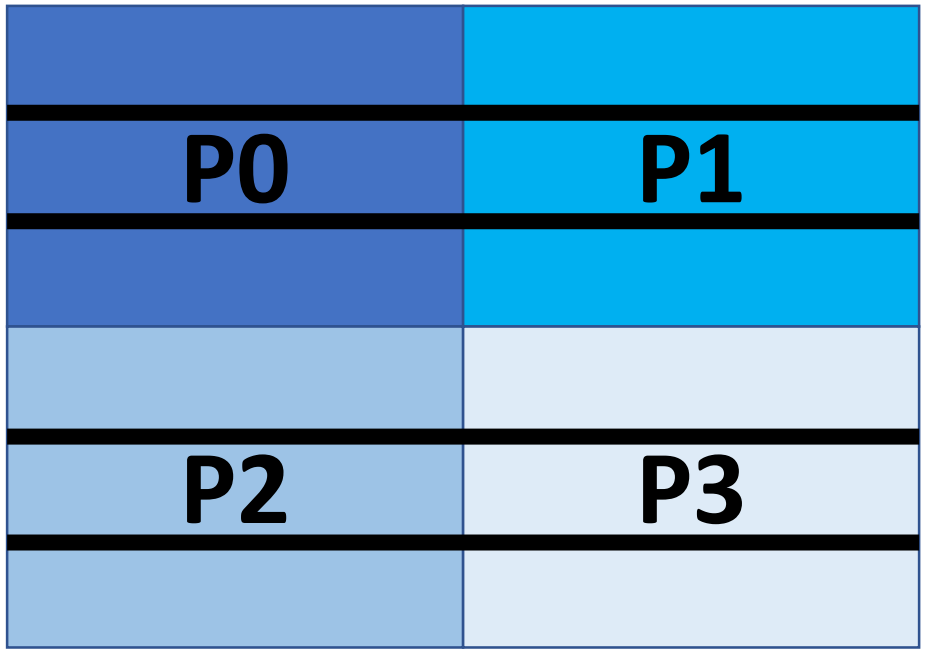
# Parallel I/O Classification

- Independent I/O (we saw till now)
- Collective I/O (we will see next)

# Non-contiguous Accesses



# Multiple Non-contiguous Accesses




- Every process' local array is non-contiguous in file
- Every process needs to make small I/O requests
- Can these requests be merged?

# MPI Collective I/O

MPI_File_open (MPI_COMM_WORLD, “/scratch/largefile”, MPI_MODE_RDONLY, MPI_INFO_NULL, &fh)

MPI_File_read_at_all (fh, offset, buffer, count, MPI_INT, status)

MPI_File_read_all (fh, buffer, count, MPI_INT, status)

MPI_File_close (&fh)

```
class mpirun -np 10 ./2.indepIOnoncontig 100000
time diff 0.089348 1.067375 MB/s
class mpirun -np 10 ./2.indepIOnoncontig 100000
time diff 0.147186 0.647937 MB/s
class mpirun -np 10 ./2.indepIOnoncontig 100000
time diff 0.535093 0.178226 MB/s
class
class
class mpirun -np 10 ./3.collIOnoncontig 100000
time diff 0.005033 18.947468 MB/s
class mpirun -np 10 ./3.collIOnoncontig 100000
time diff 0.002475 38.531933 MB/s
class mpirun -np 10 ./3.collIOnoncontig 100000
time diff 0.002582 36.931031 MB/s
```

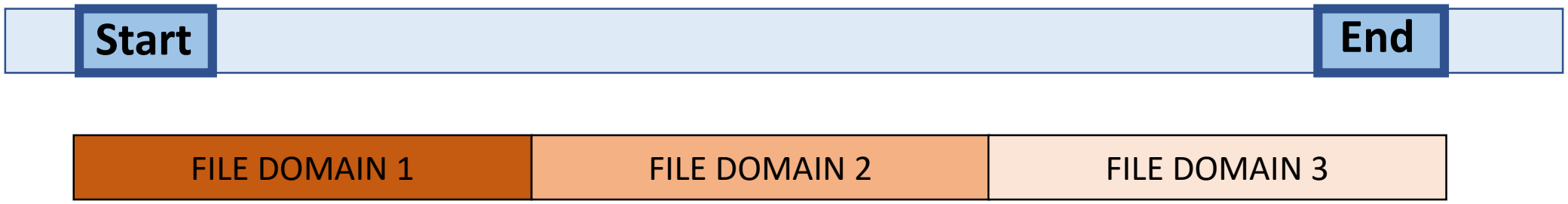
# Two-phase I/O

Entire access pattern must be known before making file accesses

- Phase 1
  - Processes request for a single large contiguous chunk
  - Reduced file I/O cost due to large accesses
- Phase 2
  - Processes redistribute data among themselves
  - Additional inter-process communications

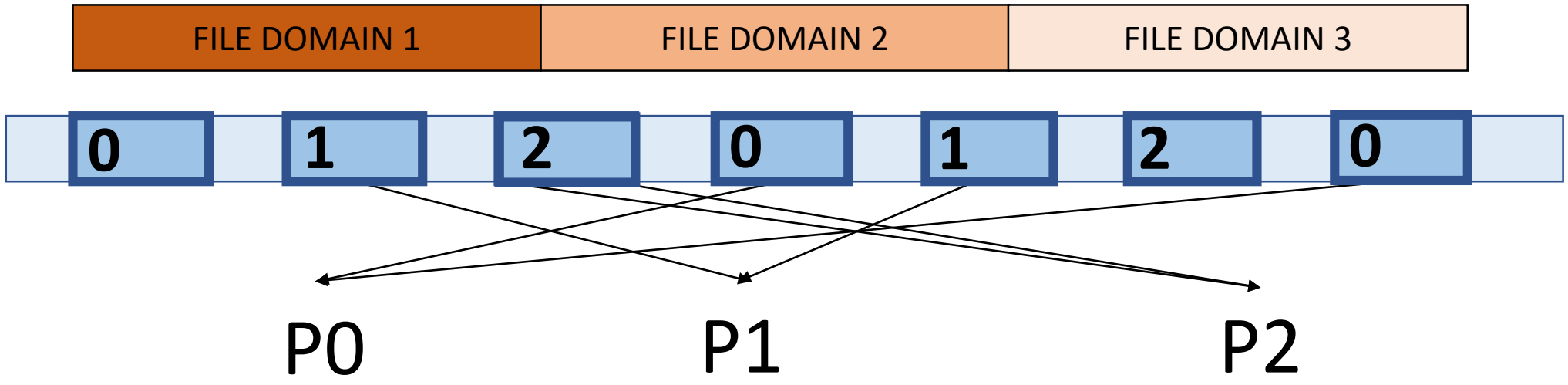
# Two-phase I/O

- Phase 1
  - Processes analyze their own I/O requests
  - Create list of offsets and list of lengths
  - Everyone broadcasts start offset and end offset to others
  - Each process reads its own *file domain*

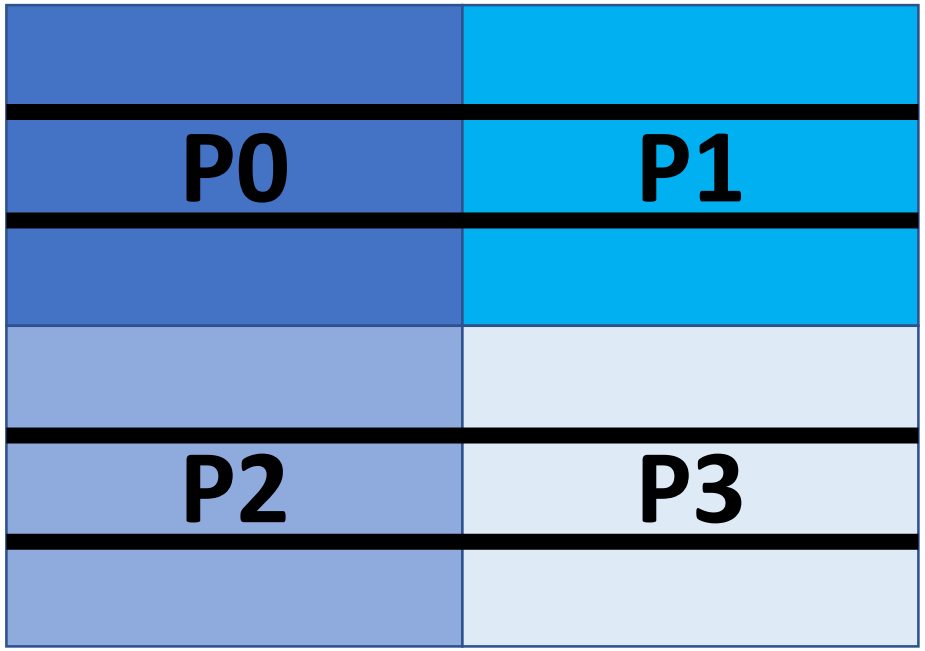


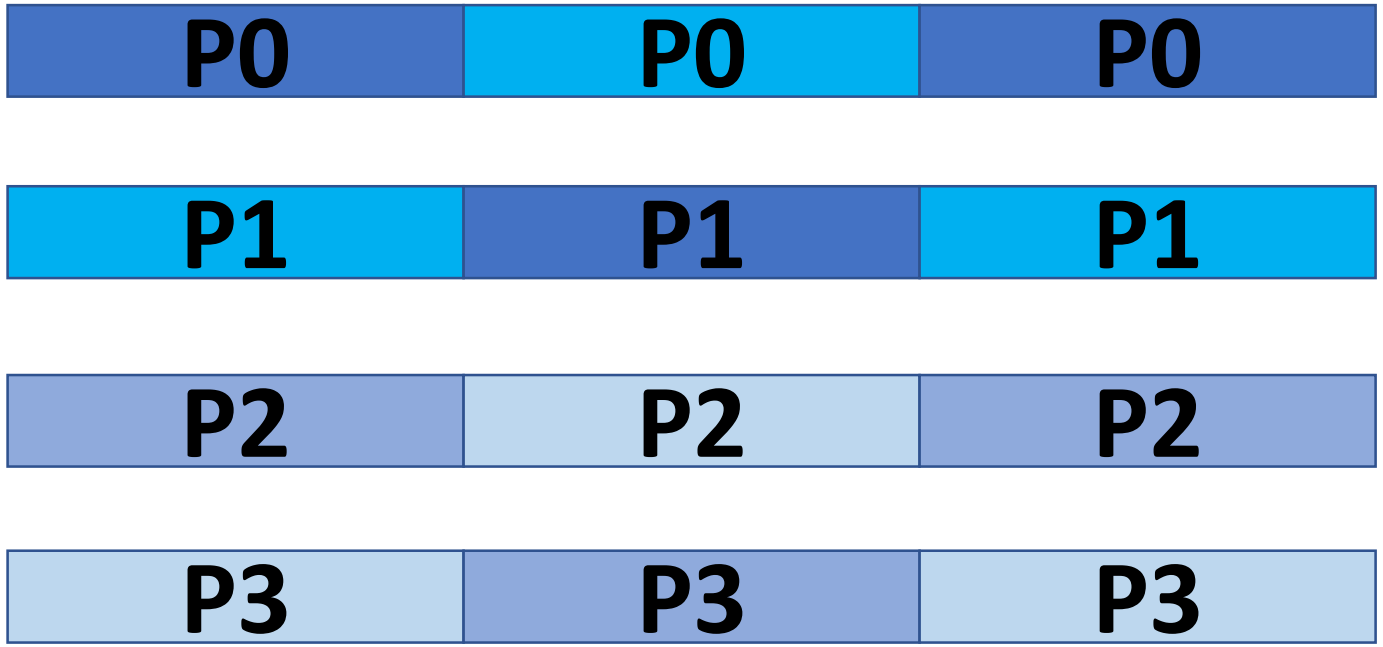
# Two-phase I/O

- Phase 2
  - Processes analyze the file domains
  - Processes exchange data with the corresponding process



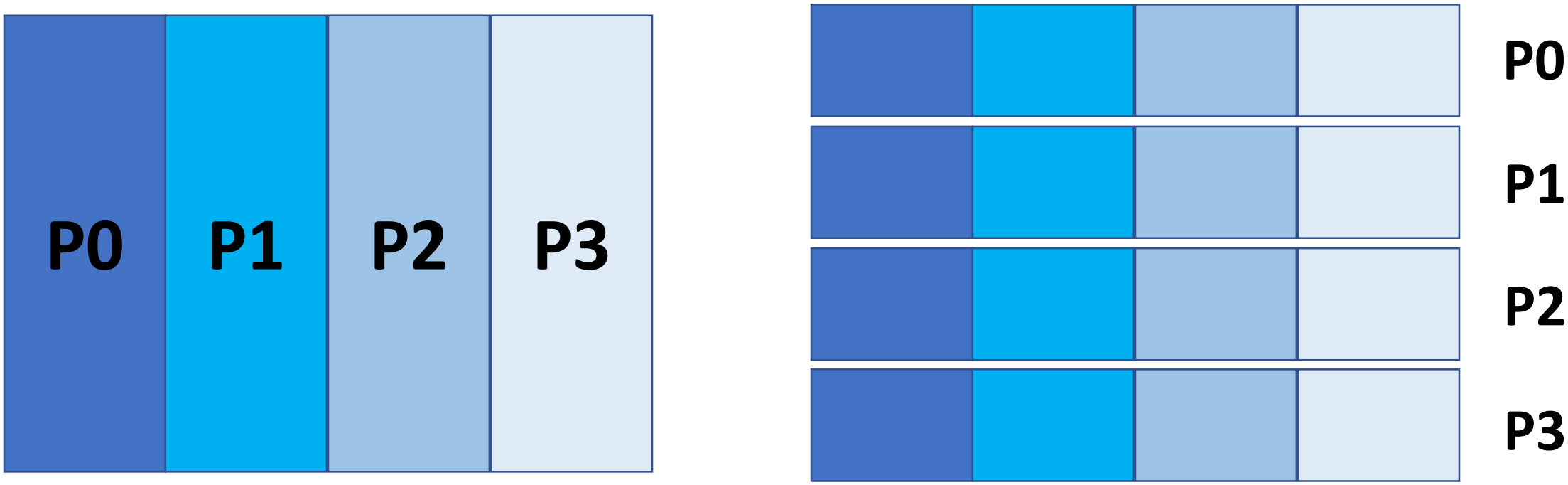
# File domain – Example



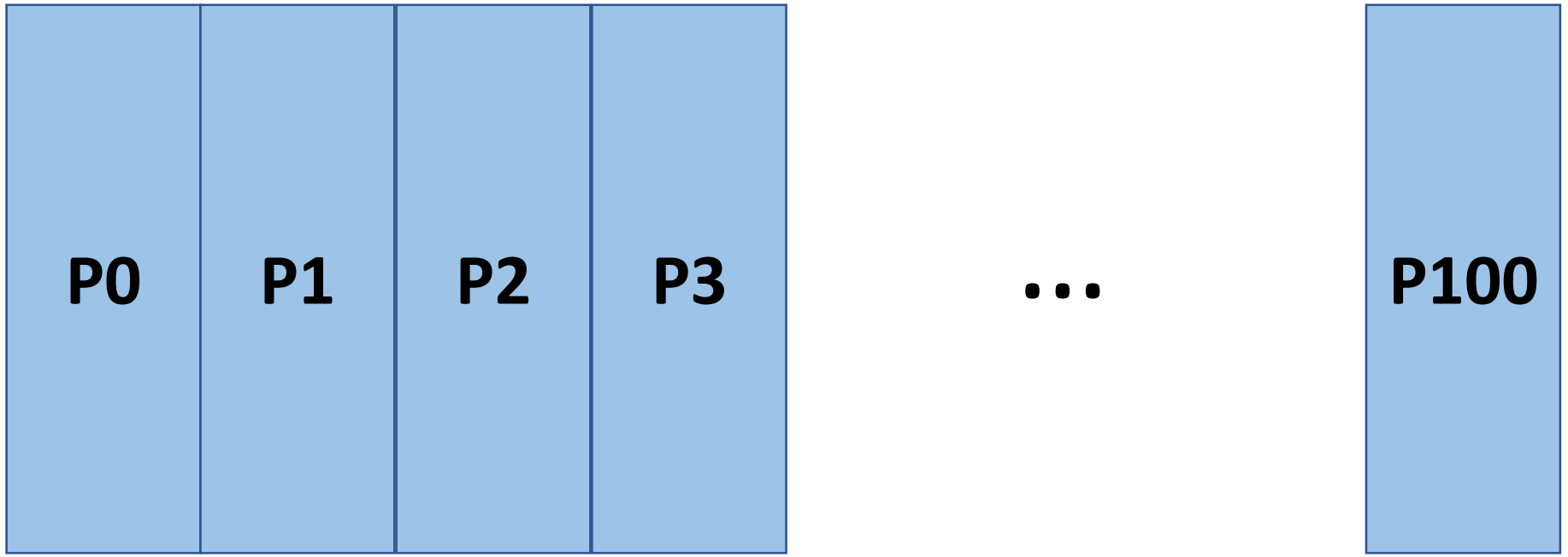

P0 and P1 exchange, P2 and P3 exchange

# File domain – Example



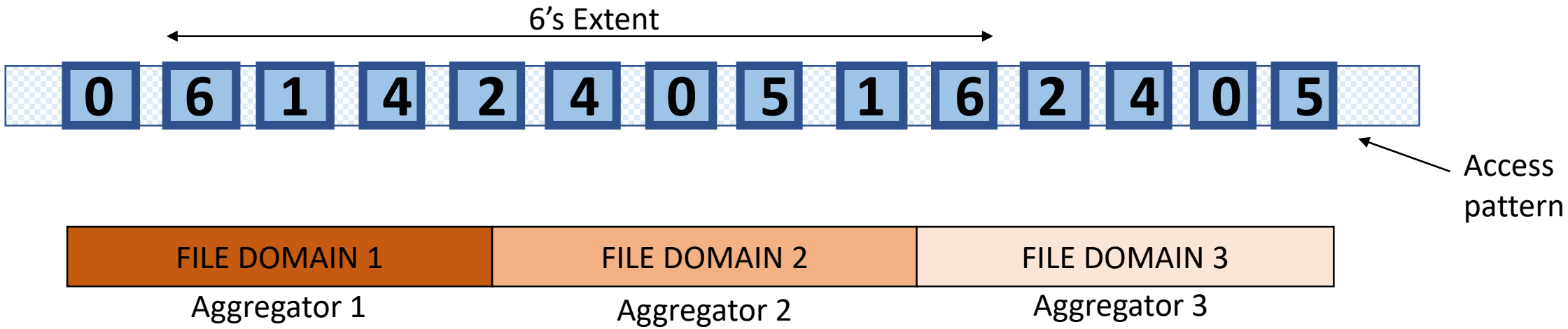

Everyone needs data from every other process

# Collective I/O




Communication may become bottleneck?

# Collective I/O Aggregators




- Multiple small non-contiguous I/O requests from different processes are combined
- A subset of processes, I/O aggregators, access their file domains (I/O phase)
- Data redistributed among all processes (communication phase)
- Cons - Synchronization

# Aggregators

- Too few aggregators
  - Large buffer size required per aggregator and multiple I/O iterations
  - Underutilization of the full bandwidth of the storage system
- Too many aggregators
  - Request for large number of small chunks ➔ suboptimal file system performance
  - Increased cost of data exchange operations

In MPICH

- Buffer size in aggregators = 16 MB
- Default number of aggregators – #unique hosts which open the file
- Placement – Specific to file system
  - mpich/src/mpi/romio/adio/ad_gpfs/pe/ad_pe_aggrs.c (GPFS)

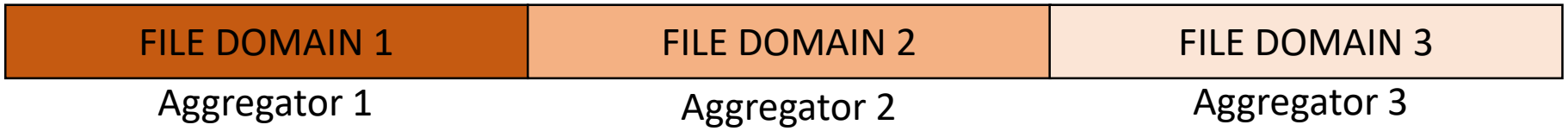
# I/O Aggregators – Limited buffer

Total number of processes = 1024
Let each process read $2^{20}$ doubles (= 1 MB)
Total number of aggregators = 16
Temporary buffer in each aggregator process = 4 MB

- Collective I/O may be done in several iterations
- Double buffering may help

I/O data size per aggregator (D) = $\frac{1024 * 1}{16}$ MB

Number of times each aggregator needs to do the I/O = $\frac{D}{4}$ = 16



# User-controlled Parameters

- Number of aggregators
- Buffer size in aggregators

# User-controlled Parameters

- Number of aggregators (cb_nodes, default=1 per node)
- Placement of aggregators (cb_config_list)
- Buffer size in aggregators (cb_buffer_size, default=16MB)
- …

- Can be set via hints (ROMIO_HINTS file)
- MPI_Info object is used to pass hints

# Set Hints via MPI_Info

```
MPI_Info_create(&info);

MPI_Info_set(info, "cb_nodes", "8");

MPI_File_open(MPI_COMM_WORLD, filename, amode, info, &fh);
```

# Set Hints via Environment Variable

- export ROMIO_HINTS=$PWD/romio_hints

```
romio_cb_read enable
romio_cb_write enable
ind_rd_buffer_size 4194304
ind_wr_buffer_size 1048576
cb_config_list=*:1
```

# MPIIO Hints

```
MPI_File_open (MPI_COMM_WORLD, "/pfs/datafile", MPI_MODE_CREATE |
MPI_MODE_RDWR, MPI_INFO_NULL, &fh);
MPI_File_get_info (fh, &info_used);
MPI_Info_get_nkeys (info_used, &nkeys);
for (i=0; i<nkeys; i++) {
    MPI_Info_get_nthkey (info_used, i, key);
    MPI_Info_get (info_used, key, MPI_MAX_INFO_VAL, value, &flag);
    printf ("Process %d, Default: key = %s, value = %s\n", rank, key, value);
}
```

export ROMIO_PRINT_HINTS=1

# Performance

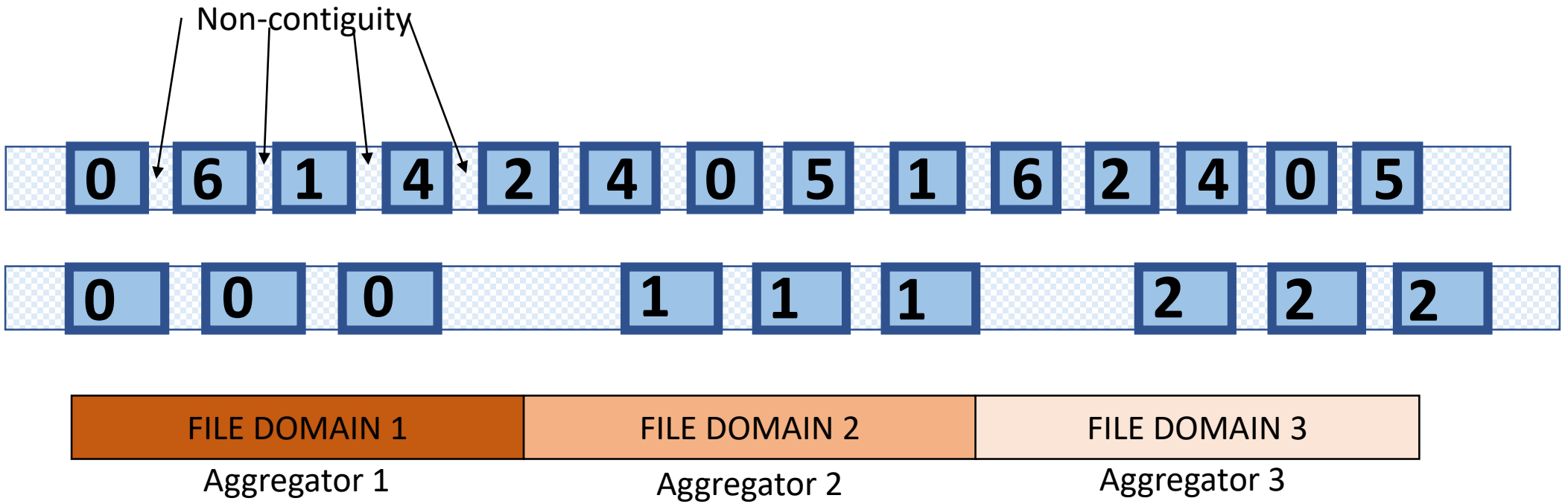
```
$ mpirun -np 6 -hosts csews2:2,csews20:2,csews30:2 ./collIOnoncontig 10485760
key = cb_buffer_size          value = 16777216
key = romio_cb_read           value = enable
key = romio_cb_write          value = enable
key = cb_nodes                value = 3
36.014944 MB/s
$ mpirun -np 6 -hosts csews2:3,csews20:3 ./collIOnoncontig 10485760
key = cb_buffer_size          value = 16777216
key = romio_cb_read           value = enable
key = romio_cb_write          value = enable
key = cb_nodes                value = 2
 27.474843 MB/s
```

# Performance

```
$ mpirun -np 6 -hosts csews2:2,csews20:2,csews30:2 ./collIOnoncontig 10485760
key = cb_buffer_size          value = 16777216
key = romio_cb_read           value = enable
key = romio_cb_write          value = disable
key = cb_nodes                value = 3
5.106368 MB/s

$ mpirun -np 6 -hosts csews2:3,csews20:3 ./collIOnoncontig 10485760
key = cb_buffer_size          value = 16777216
key = romio_cb_read           value = enable
key = romio_cb_write          value = disable
key = cb_nodes                value = 2
6.028663 MB/s
```

# I/O Aggregators




- Data sieving
- Independent I/O may be called if there is no benefit of collective I/O

# Recap

## Independent I/O

- Individual file pointers — MPI_File_read
- Explicit offsets — MPI_File_read_at
- Shared file pointers — MPI_File_read_shared

## Collective I/O

- Individual file pointers — MPI_File_read_all
- Explicit offsets — MPI_File_read_at_all
- Shared file pointers — MPI_File_read_ordered

# Non-blocking I/O

```
MPI_Request request;
MPI_File_iwrite_at (fh, offset, buf, count, datatype, &request);
MPI_File_iwrite_at_all (fh, buf, count, datatype, &request);
…
/* computation */

MPI_Wait (&request, &status);
```

# Parallel I/O approaches

- Shared file
  - Independent I/O
  - Collective I/O
- File per process
- File per group of processes

File system locking overhead is high

Numerous files, large overhead

Locking overhead nil

- Challenging for analysis and visualization codes
- Restriction on #processes while restarting

# Aggregator Selection

How many aggregators, and their placements?

- Depends on the architecture, file system, data size, access pattern
- Depends on the network topology, number of nodes and their placements, etc.