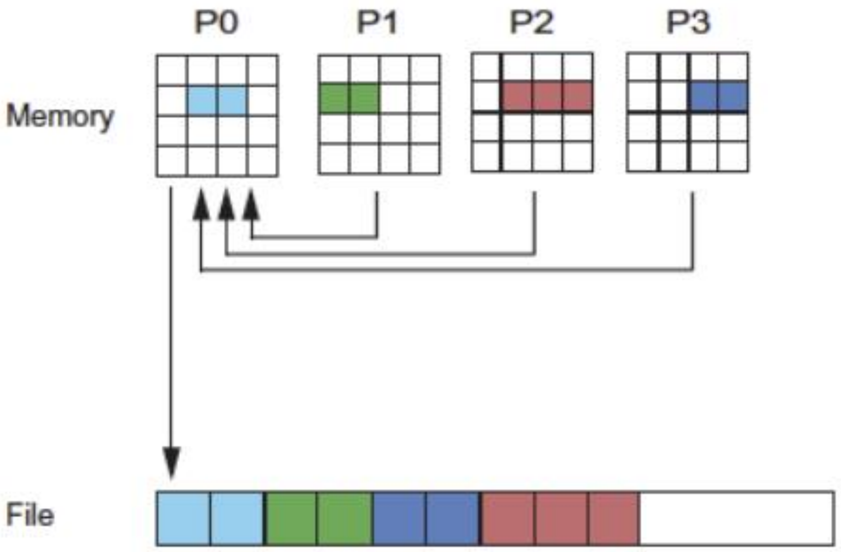
Parallel I/O

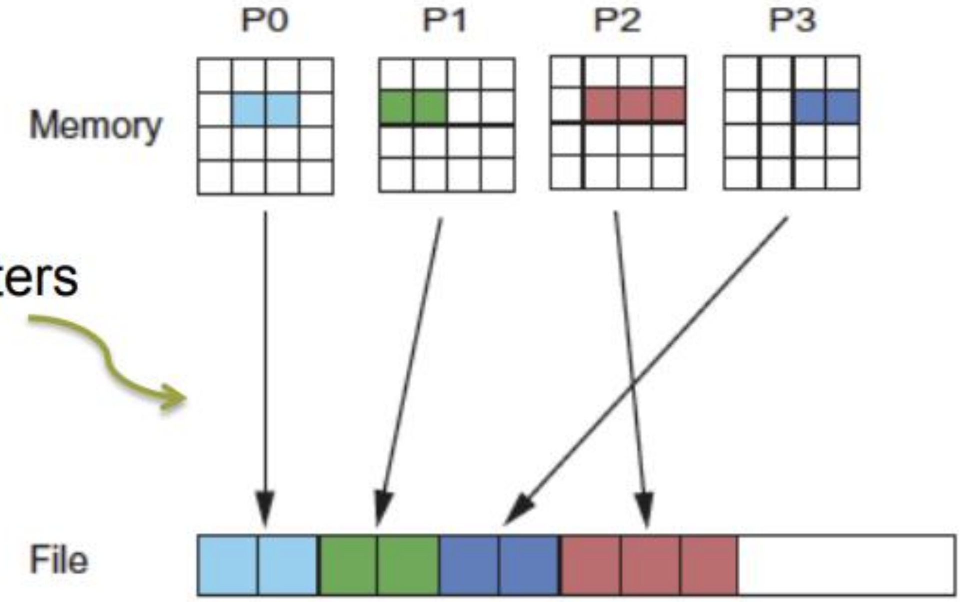
Lecture 18

April 7, 2025



From: Single file, single writer

To: Single file, many writers



Taken from "Getting Started on MPI-IO" S-2490-40

Credit: CRAY presentation@IISc

User-controlled Independent MPI-IO Parameters

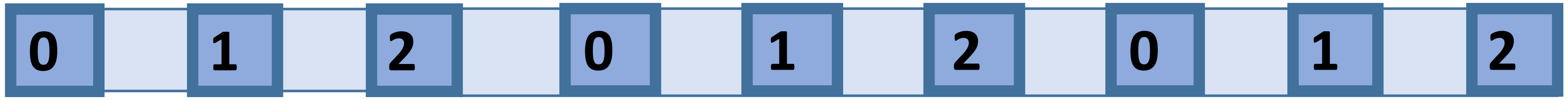
- ind_rd_buffer_size - Buffer size for data sieving for read
- ind_wr_buffer_size - Buffer size for data sieving for write
- romio_ds_read - Enable or not data sieving for read
- romio_ds_write - Enable or not data sieving for write

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);
```

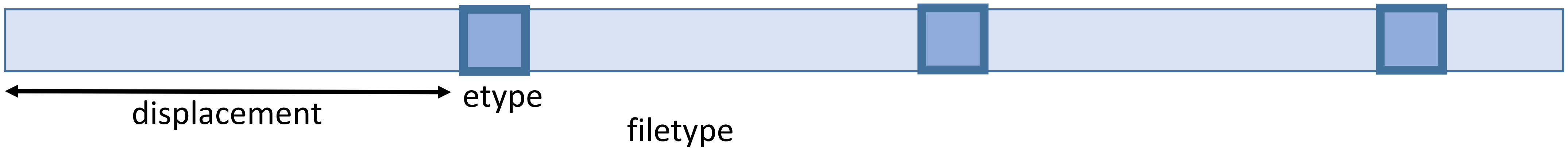
Non-contiguous I/O (3D Domain)

3D Domain Extent: 3x3x3
1D decomposition along Y-axis



File View

Non-contiguous access pattern can be specified using a view



Each process can specify a view

displacement
etype
filetype

File view

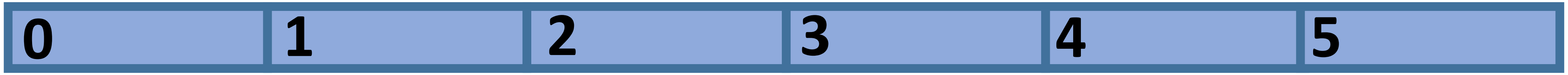
```
int MPI_File_set_view(MPI_File
fh, MPI_Offset disp,
MPI_Datatype etype, MPI_Datatype
filetype, ROMIO_CONST char
*datarep, MPI_Info info)
```

MPI_File_set_view (fh, disp, etype, filetype, “native”, MPI_INFO_NULL)

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

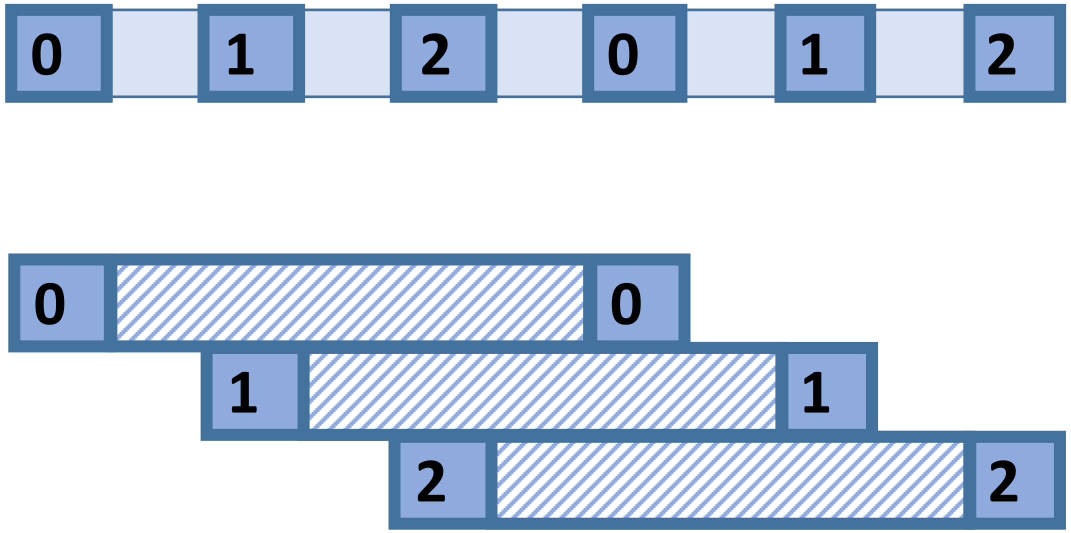
Independent I/O - Set File View

```
MPI_File_open (MPI_COMM_WORLD, filename, MPI_MODE_RDWR | MPI_MODE_CREATE, MPI_INFO_NULL, &myfile);

for (i=0; i<BUFSIZE; i++) {
   buf[i] = myrank + i;
}

// File write - set process view
MPI_File_set_view(myfile, myrank * BUFSIZE * sizeof(int), MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
MPI_File_write (myfile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);

// File read - set process view
MPI_File_set_view(myfile, myrank * BUFSIZE * sizeof(int), MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
MPI_File_read (myfile, rbuf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);

MPI_File_close (&myfile);

for (i=0; i<BUFSIZE; i++) {
  if (buf[i] != rbuf[i]) printf ("%d %d %d\n", i, buf[i], rbuf[i]);
}
```

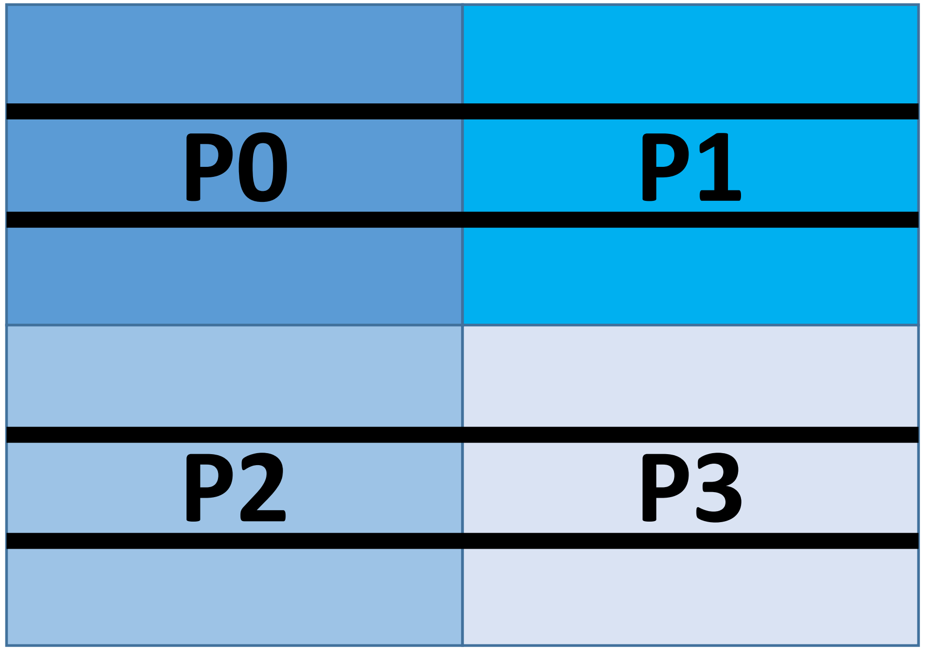


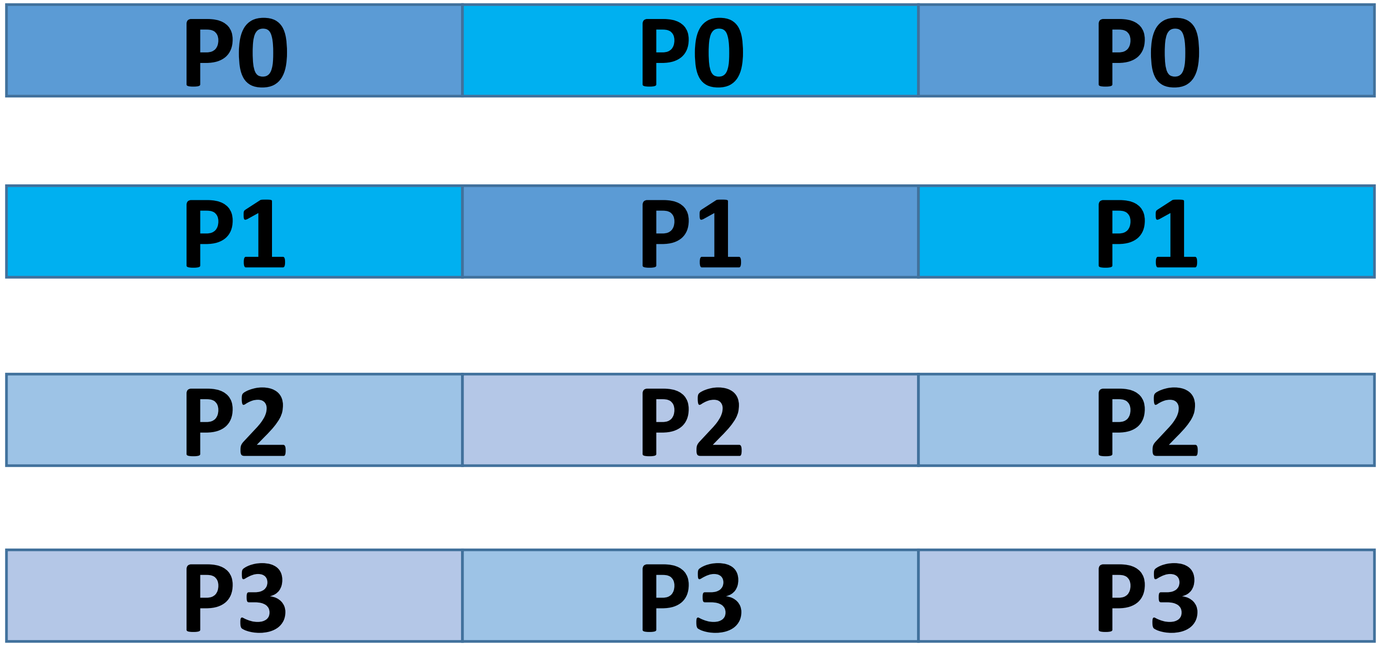
Non-contiguous Access – Set File View

MPI_Init
MPI_File_open
MPI_Type_vector
MPI_Type_commit
MPI_File_set_view
MPI_File_read
MPI_File_close
MPI_Type_free
MPI_Finalize



File domain – Example





All processes read equal amounts of data

P0 and P1 exchange, P2 and P3 exchange

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)

or

MPI_File_set_view ….

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

MPI_File_close (&fh)

MPI Collective I/O

```
for i in `seq 1 5` ; do mpirun -np 4 ./indep 16384 ; done
time diff 0.020404 3.063181 MB/s
time diff 0.021844 2.861241 MB/s
time diff 0.019550 3.196956 MB/s
time diff 0.023124 2.702850 MB/s
time diff 0.029048 2.151631 MB/s
```

```
for i in `seq 1 5` ; do mpirun -np 4 ./coll 16384 ; done
time diff 0.004425 14.124899 MB/s
time diff 0.002833 22.062279 MB/s
time diff 0.002439 25.627530 MB/s
time diff 0.002959 21.123610 MB/s
time diff 0.002194 28.481530 MB/s
```

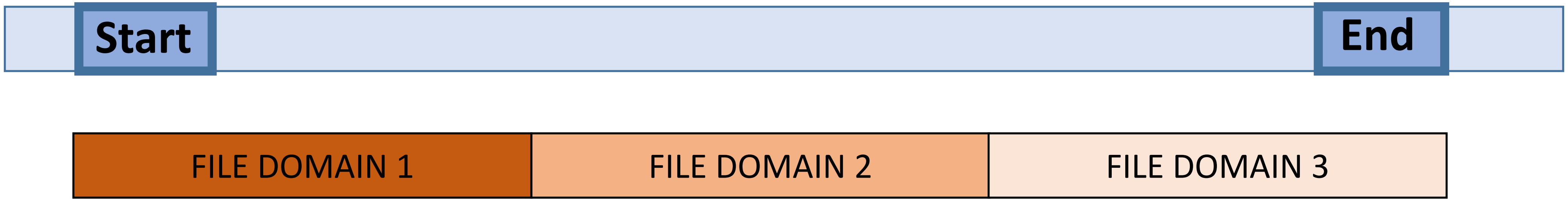
Two-phase I/O

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*

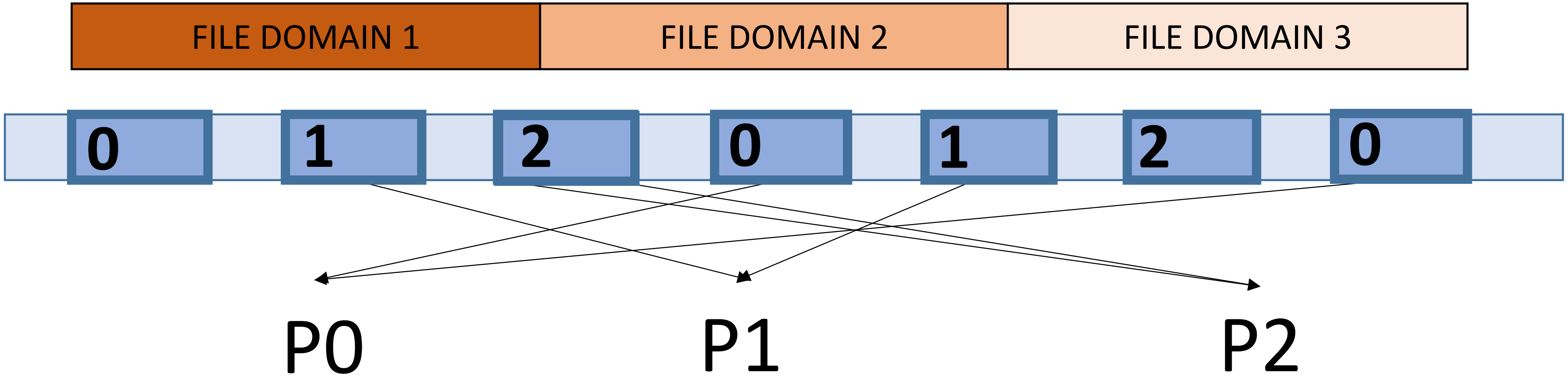


mpich/src/mpi/romio/adio/common/ad_read(write)_coll.c

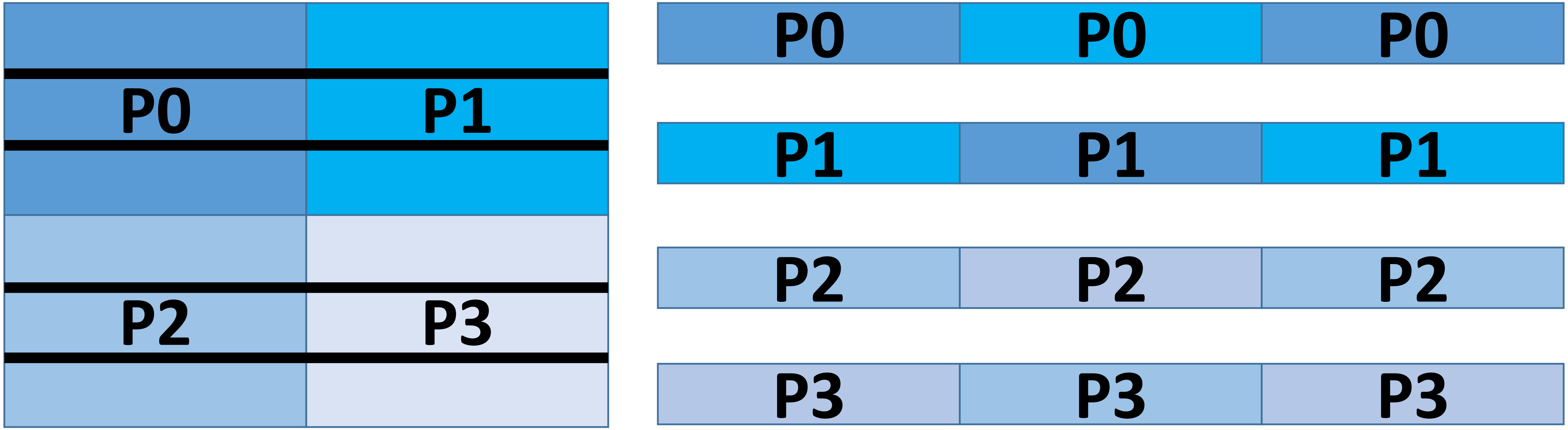
```
MPI_Allgather(&start_offset, 1, ADIO_OFFSET, st_offsets, 1, ADIO_OFFSET, fd->comm);
MPI_Allgather(&end_offset, 1, ADIO_OFFSET, end_offsets, 1, ADIO_OFFSET, fd->comm);

for (i = 1; i < nprocs; i++)
    if ((st_offsets[i] < end_offsets[i - 1]) && (st_offsets[i] <= end_offsets[i]))
        interleave_count++;
```

Two-phase I/O

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

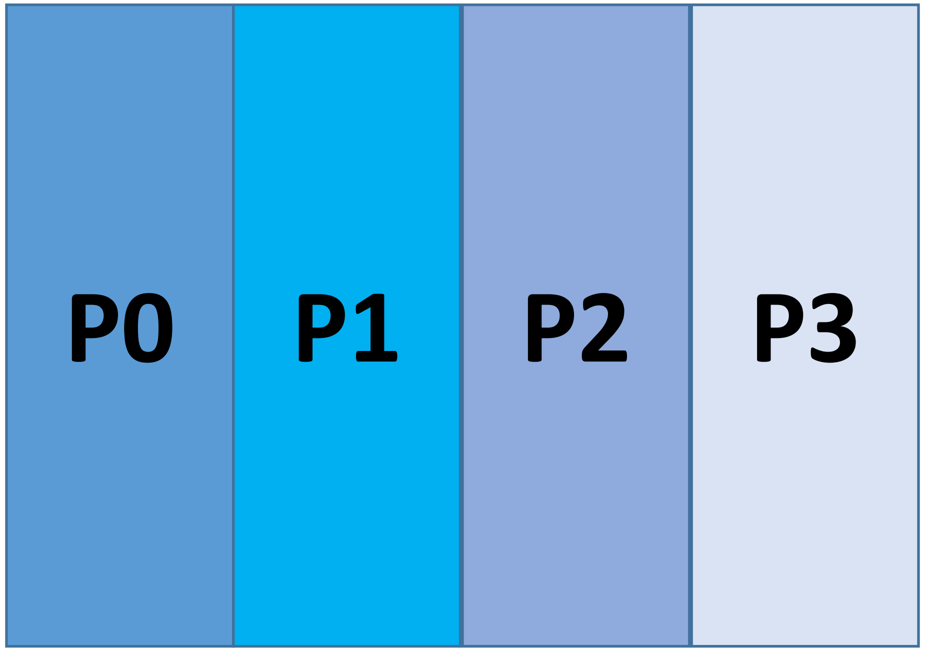


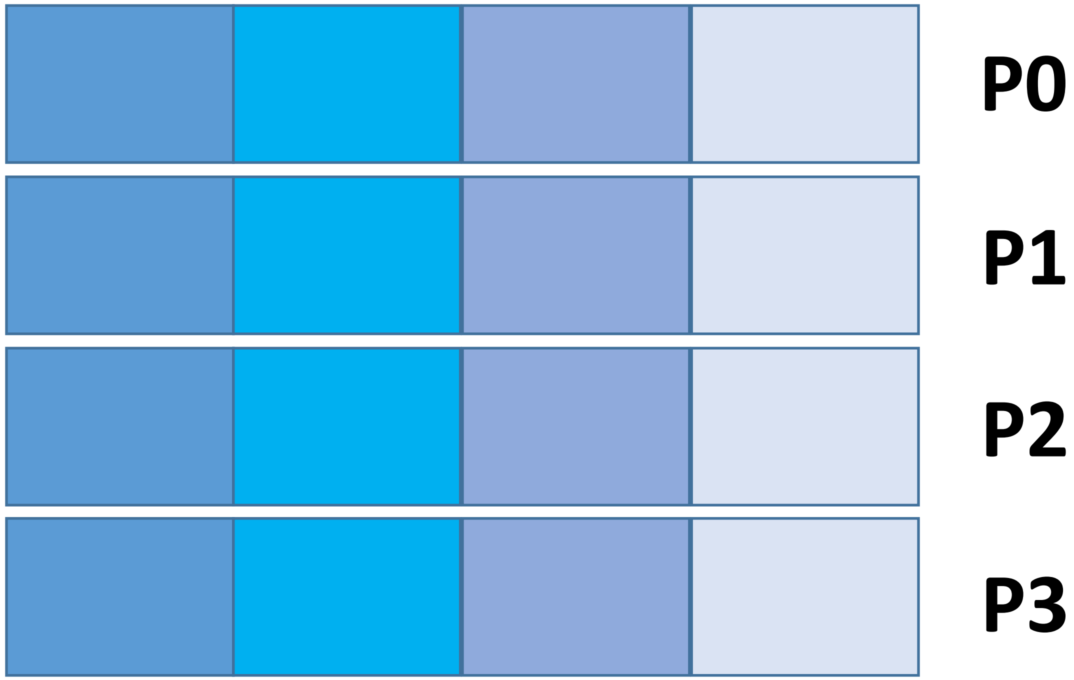
File domain – Example



P0 and P1 exchange, P2 and P3 exchange

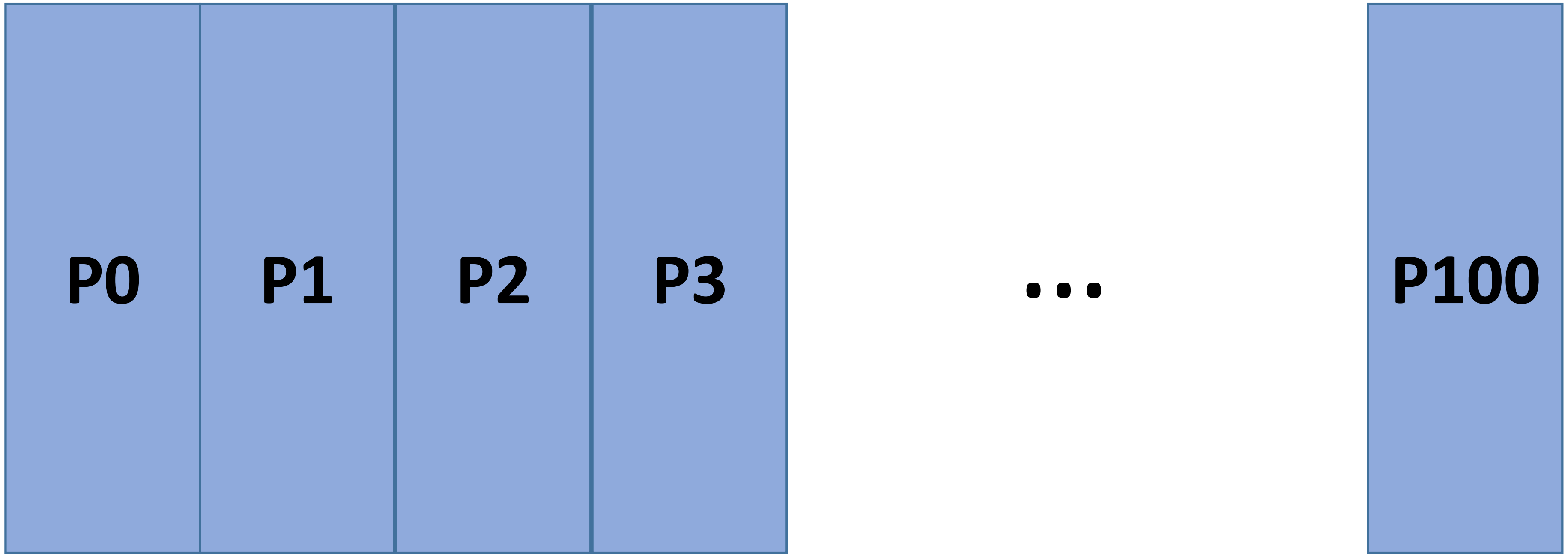
File domain – Example 2





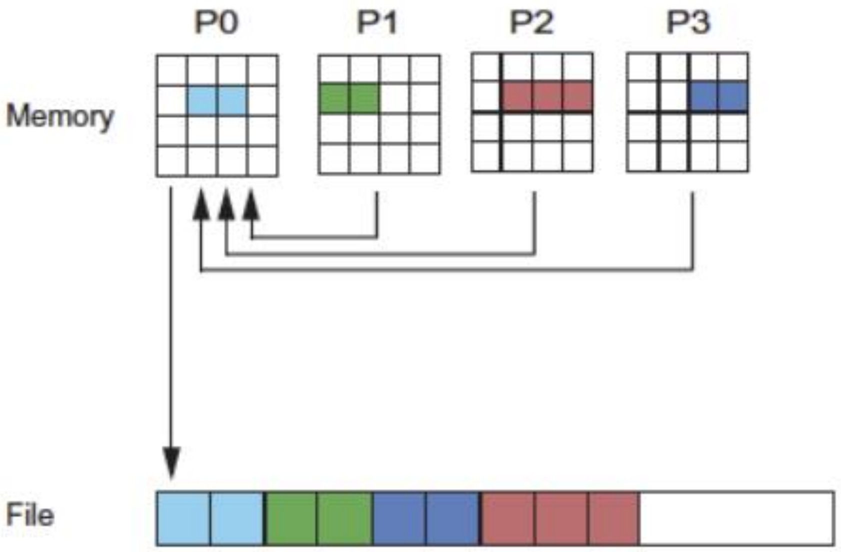
Everyone needs data from every other process

Collective I/O

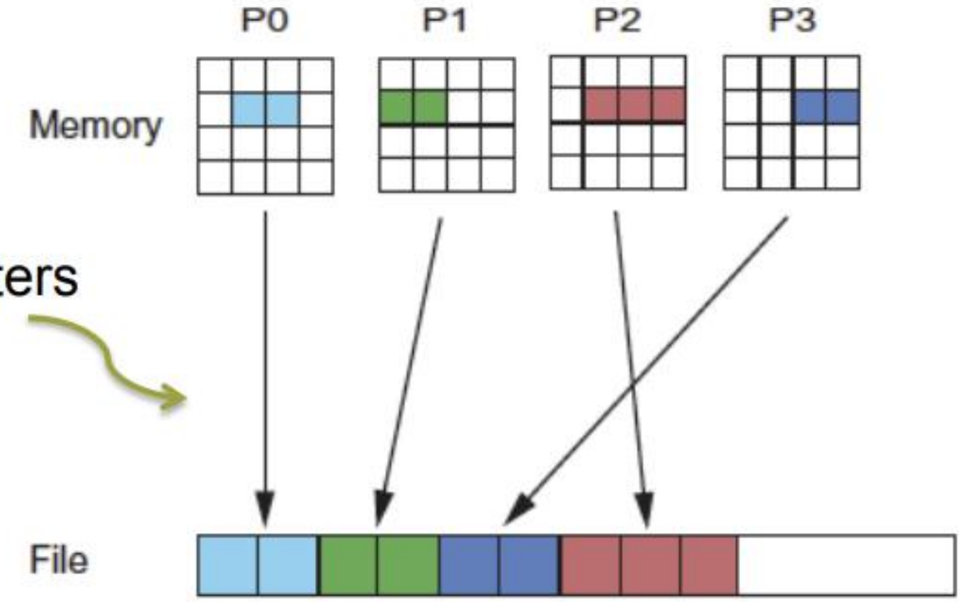


Communication may become bottleneck?

Requires 100 processes participating in Alltoall communications



From: Single file, single writer



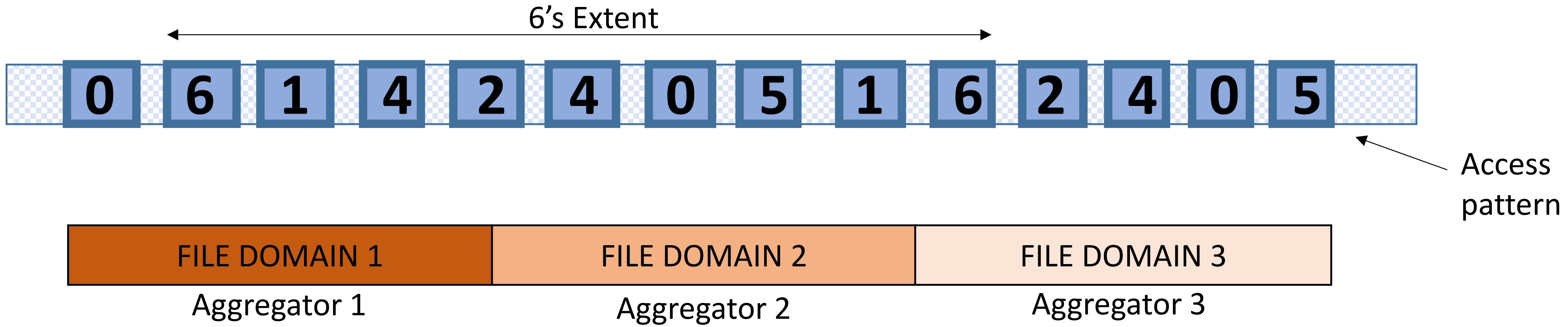
To: Single file, many writers

Taken from "Getting Started on MPI-IO"
S-2490-40

COMPUTE |

Credit: CRAY presentation@IISc

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?

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
- Placement – Specific to file system
  - src/mpi/romio/adio/ad_gpfs/ad_gpfs_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

FILE DOMAIN 1	FILE DOMAIN 2	FILE DOMAIN 3
Aggregator 1	Aggregator 2	Aggregator 3

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
```

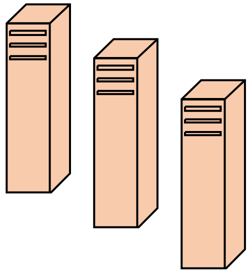
MPIIO Hints

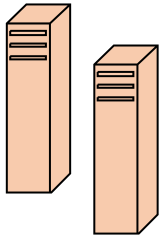
```
MPI_File_open (MPI_COMM_WORLD, "/pfs/datafile", MPI_MODE_CREATE |
MPI_MODE_RDWR, MPI_INFO_NULL, &fh);
MPI_File_get_info (fh, &info_used);
MPI_Info_get_nkeys (info_used, &nkeys);
for (i=0; i<nkeys; i++) {
    MPI_Info_get_nthkey (info_used, i, key);
    MPI_Info_get (info_used, key, MPI_MAX_INFO_VAL, 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 ./coll-IOnoncontig 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 ./coll-IOnoncontig 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
```

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);
```

Aggregator Selection

Number of aggregators and their placements

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