Parallel I/O - II

Lecture 16

March 13, 2024

Recap

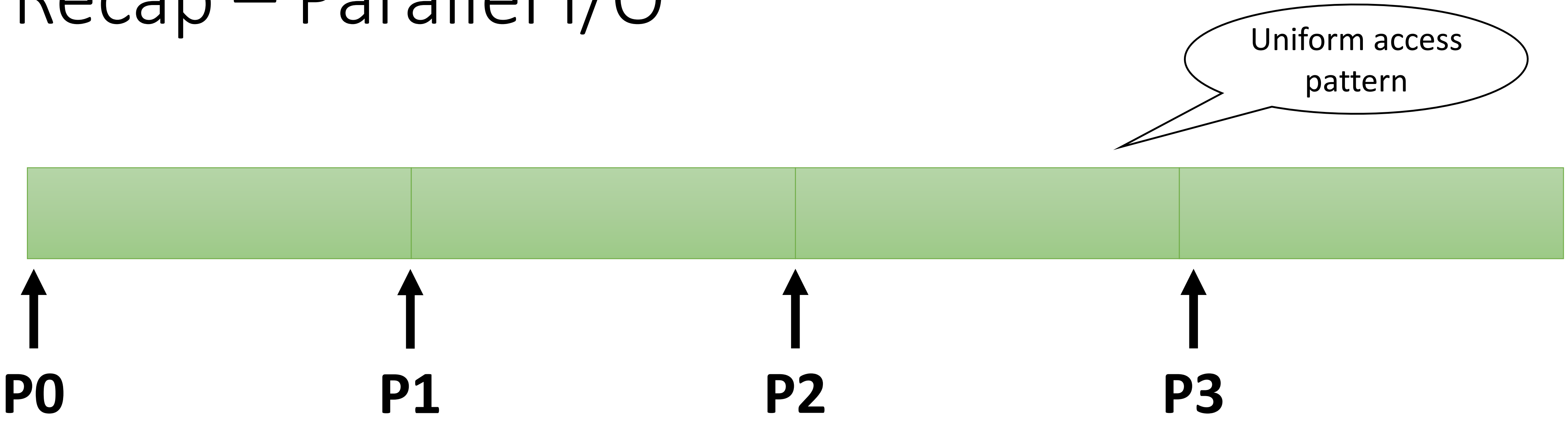
Independent I/O

- Individual file pointers   MPI_File_read
- Explicit offsets   MPI_File_read_at

Collective I/O

- Individual file pointers   MPI_File_read_all
- Explicit offsets   MPI_File_read_at_all

Recap – Parallel I/O



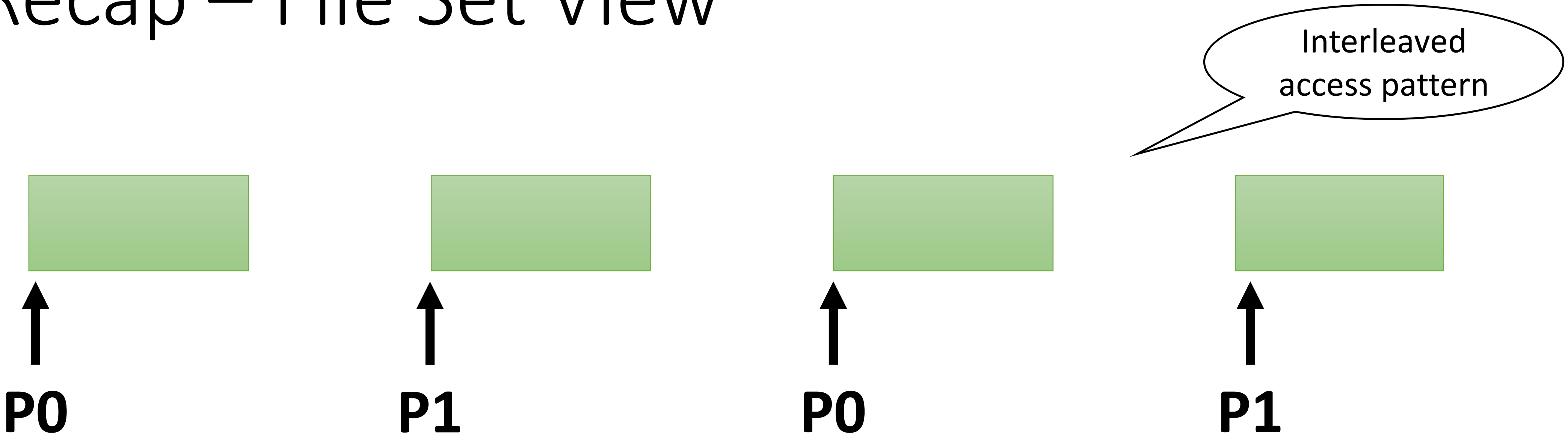
Each process reads a big chunk of data (one-fourth of the file) from a common file

Independent I/O

```
MPI_File_set_view (fh, rank*file_size_per_proc, …)
MPI_File_read (fh, data, datacount, MPI_INT, status)
```

fh: individual file pointer

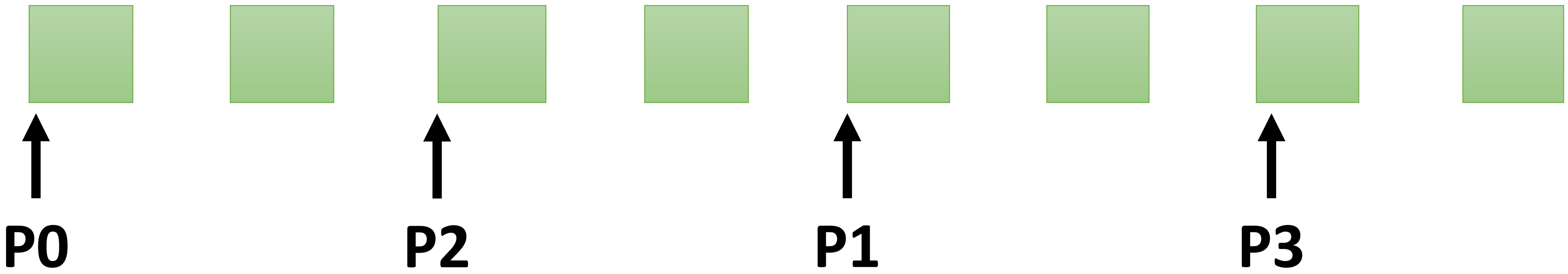
Recap – File Set View



Each process reads a small chunk of data from a common file

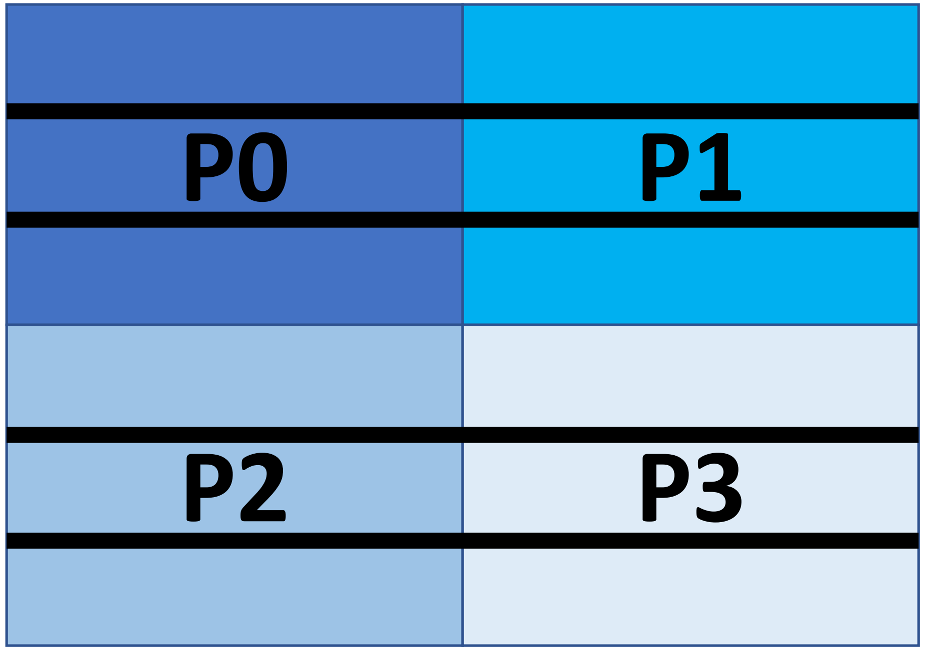
```
MPI_File_set_view (fh, displacement, etype, filetype, “native”, info)
MPI_File_read_all (fh, data, datacount, MPI_INT, status)
```

Non-contiguous Access Pattern



Each process reads several non-contiguous chunks of data from a common file

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?

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

- Post-processing management overhead

MPI Collective I/O - Recap

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 ….

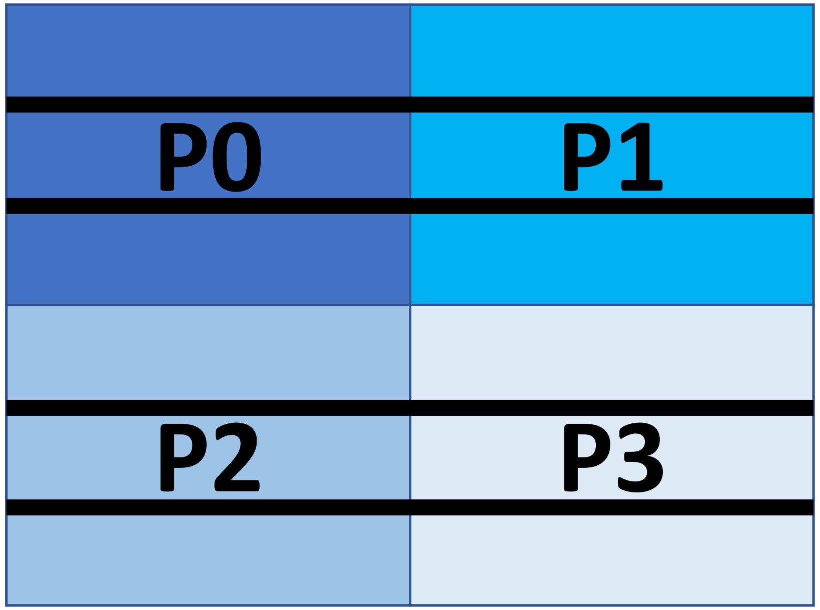
MPI_File_read_all (fh, buffer, count, MPI_INT, status)

MPI_File_close (&fh)

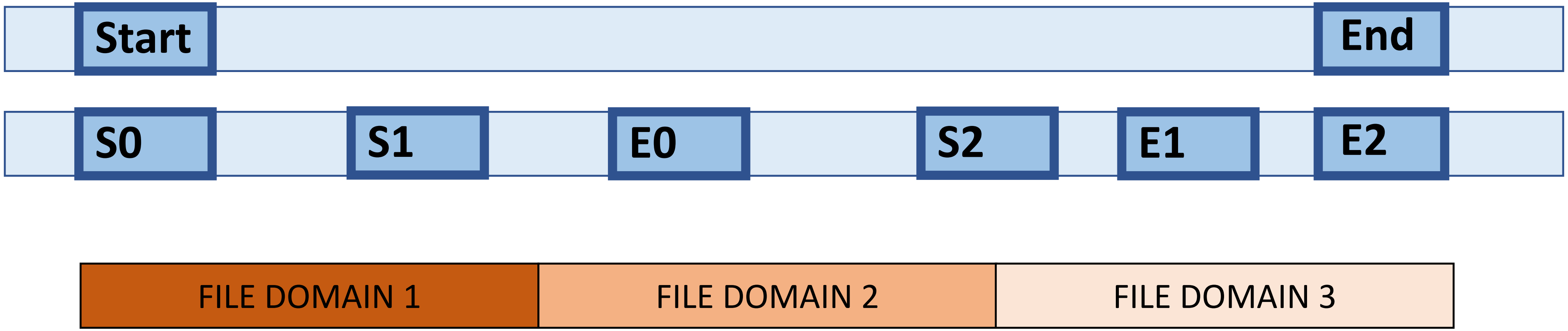
Timing

```
class mpirun -np 6 -hosts csews10,csews12,csews13 ./indepIO 200000
Time to write 0.140769 Bandwidth = 32.518728
Time to read 0.041609 Bandwidth = 110.016674
class mpirun -np 6 -hosts csews10,csews12,csews13 ./indepIO 200000
Time to write 0.052870 Bandwidth = 86.583601
Time to read 0.038442 Bandwidth = 119.080106
class
class
class mpirun -np 6 -hosts csews10,csews12,csews13 ./collIO 200000
Time to write    4.578 MB = 0.236185 seconds WB =    19.382 MB/s
Time to read    4.578 MB = 0.024631 seconds RB =   185.847 MB/s
class mpirun -np 6 -hosts csews10,csews12,csews13 ./collIO 200000
Time to write    4.578 MB = 0.055783 seconds WB =    82.062 MB/s
Time to read    4.578 MB = 0.028324 seconds RB =   161.615 MB/s
```

Two-phase I/O

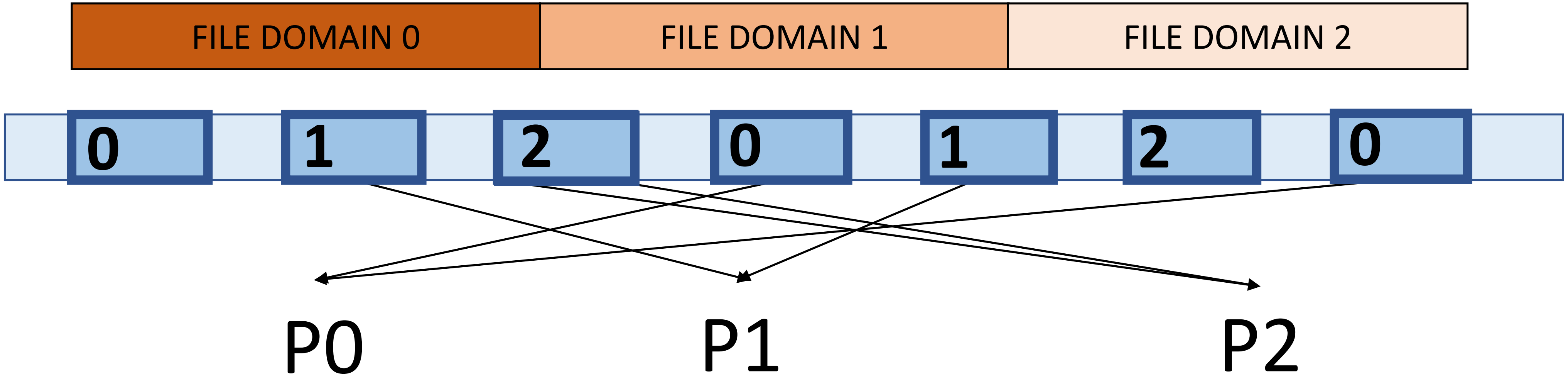


- Phase 1
  - Processes analyze their own I/O requests
  - Create list of offsets and list of lengths
  - Everyone broadcasts start offset and end offset to others
  - Each process reads its own *file domain*

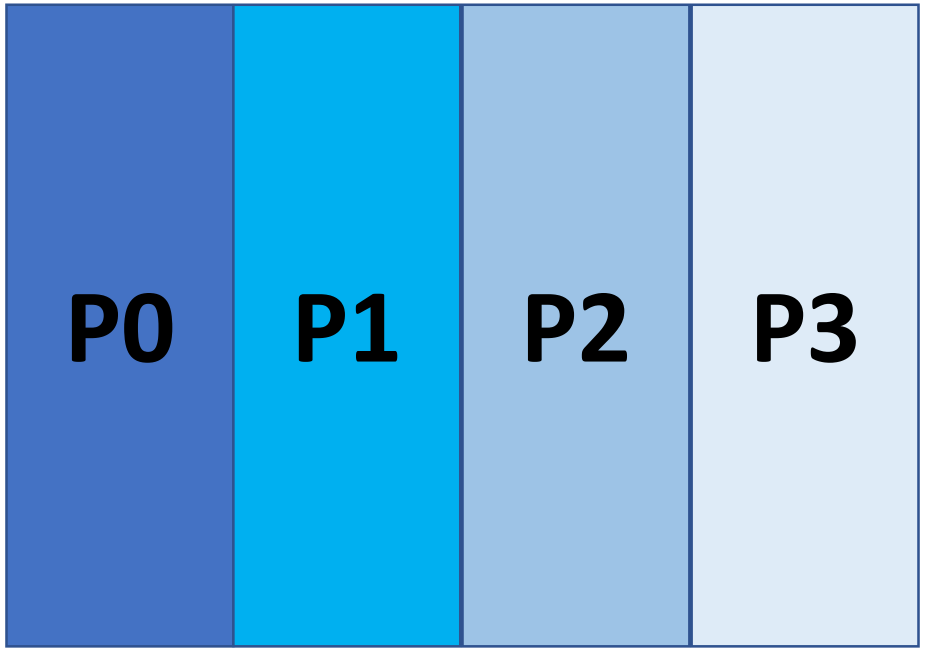


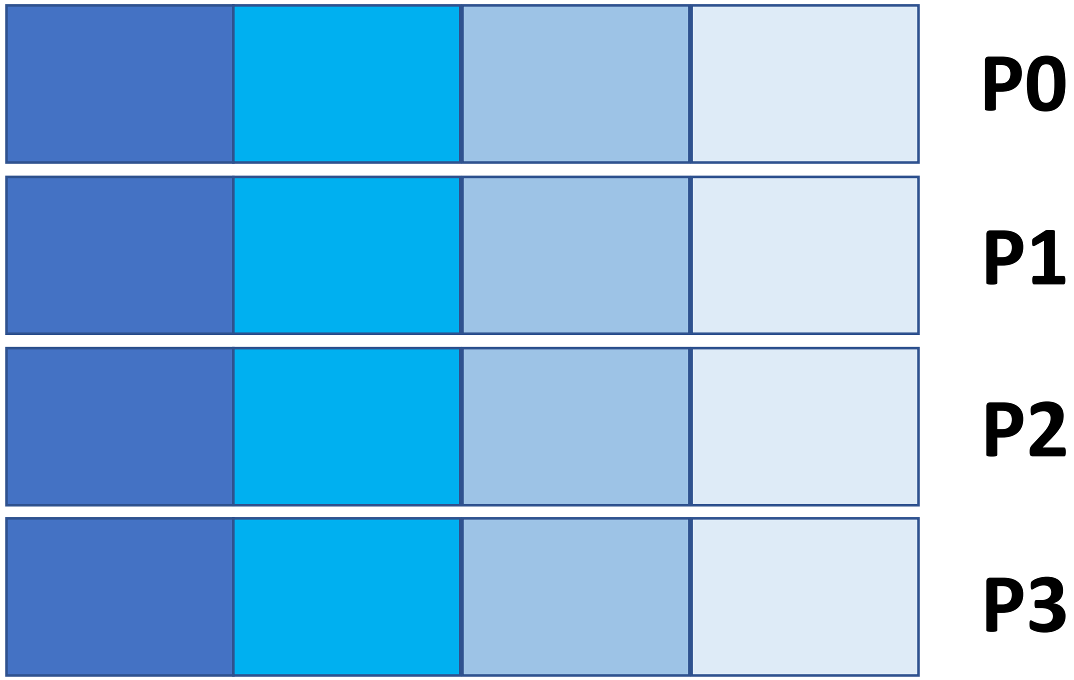
Two-phase I/O

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



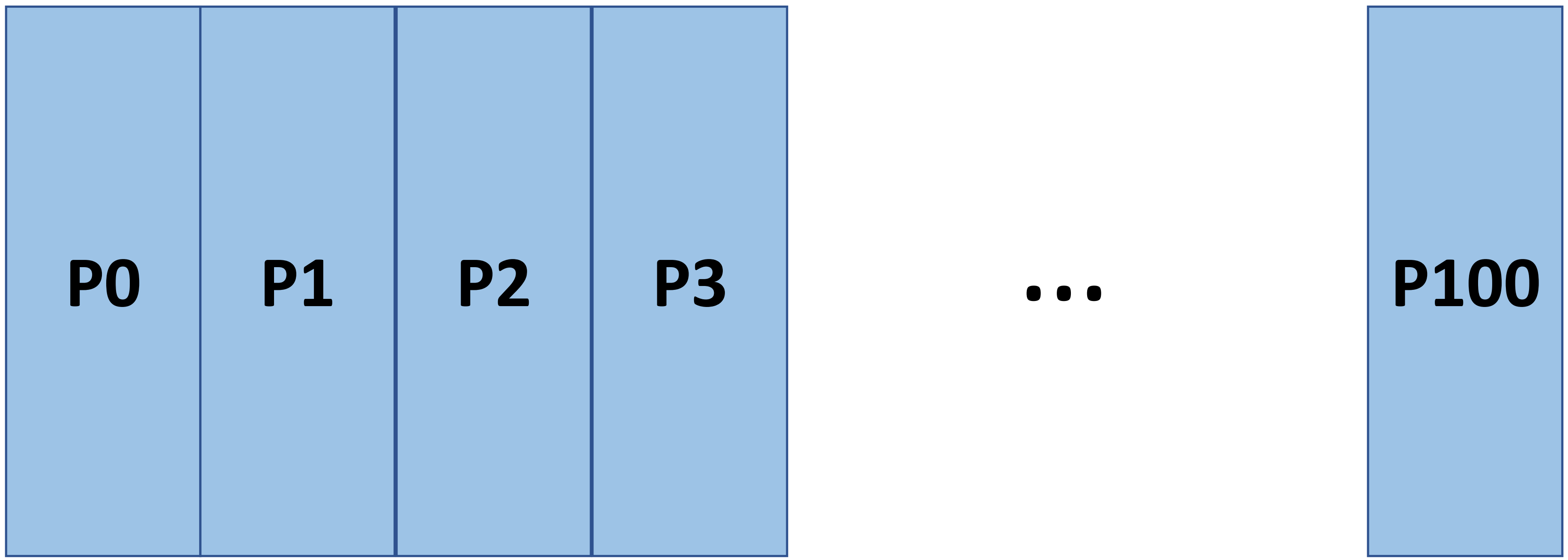
File domain – Example





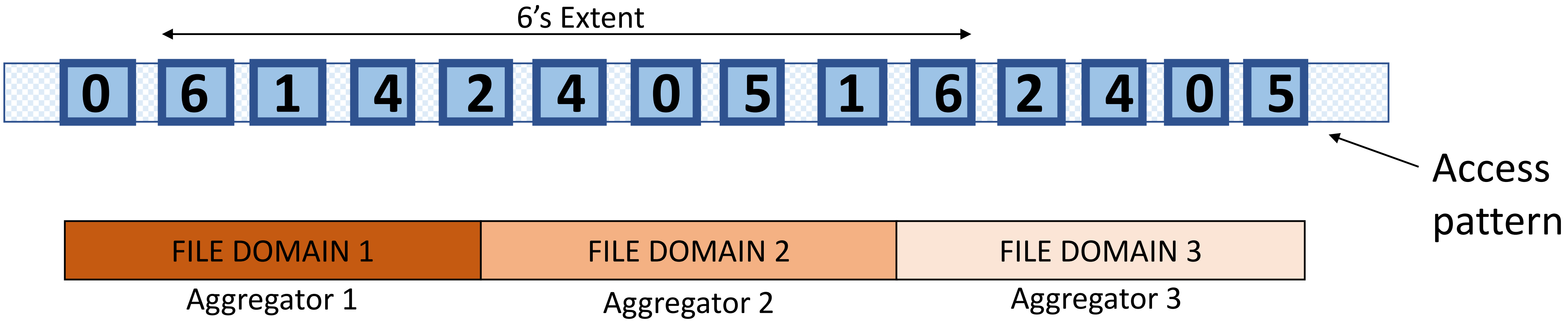
Everyone needs data from every other process

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?

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
…
```

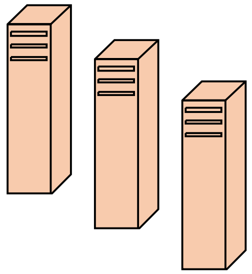
MPIIO Hints

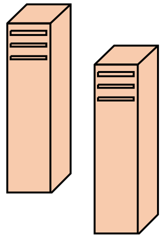
```
MPI_File_open (MPI_COMM_WORLD, "/pfs/datafile", MPI_MODE_CREATE |
MPI_MODE_RDWR, MPI_INFO_NULL, &fh);
MPI_File_get_info (fh, &info_used);
MPI_Info_get_nkeys (info_used, &nkeys);
for (i=0; i<nkeys; i++) {
    MPI_Info_get_nthkey (info_used, i, key);
    MPI_Info_get (info_used, key, MPI_MAX_INFO_VAL,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

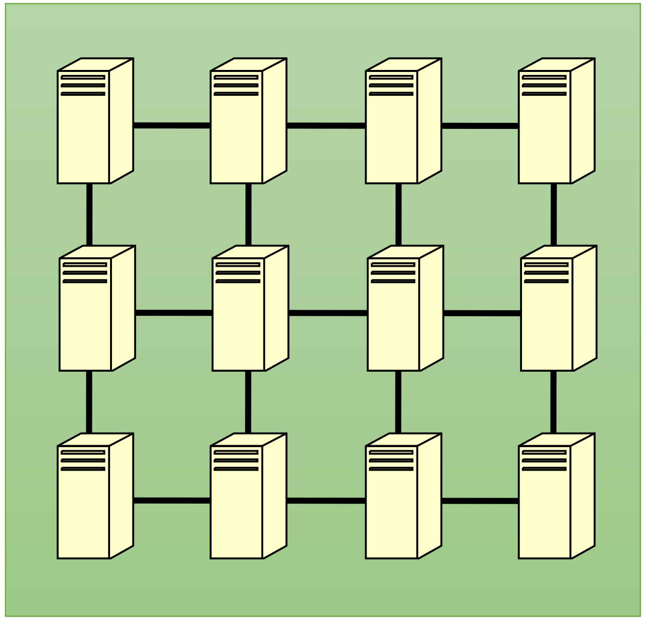
Non-blocking I/O

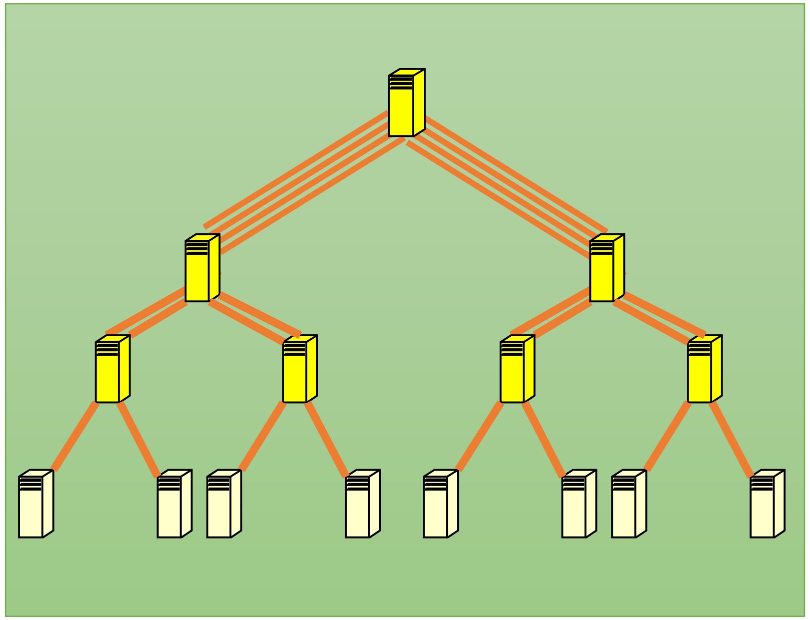
```
MPI_Request request;

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

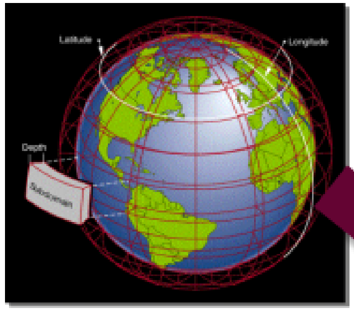
- Depends on the architecture, file system, data size, access pattern
- Depends on the network topology, number of nodes and node placements





Practical I/O

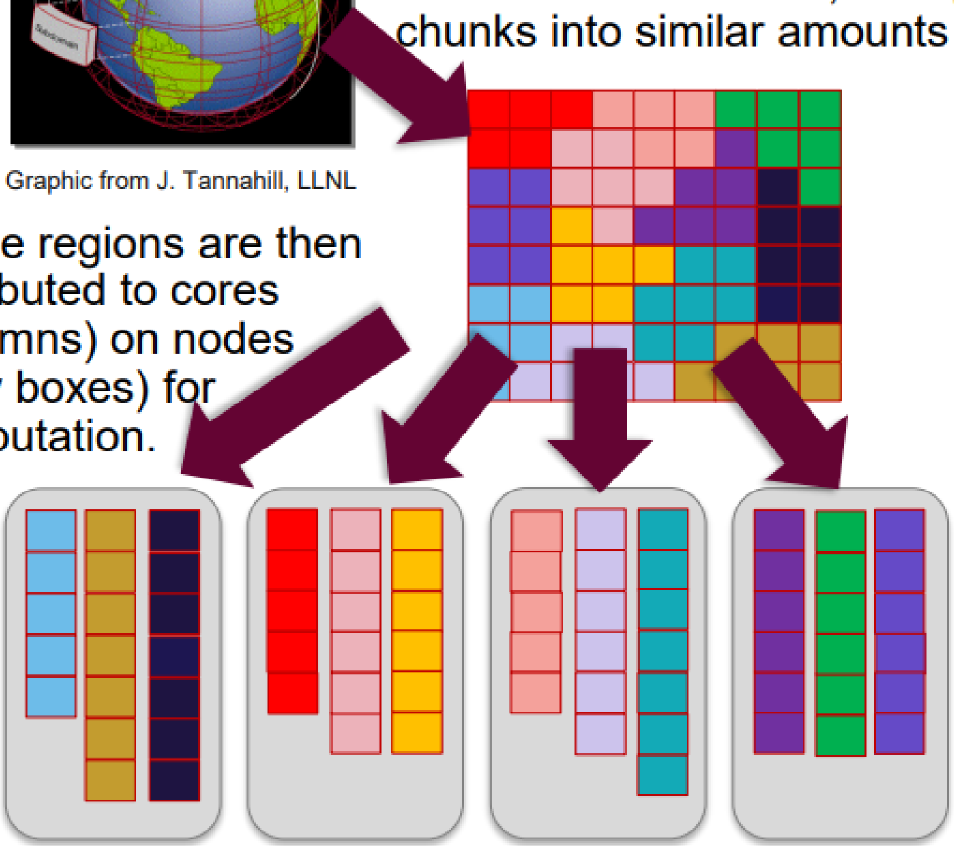
https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-7-Talk-7-latham-mpiio.pdf

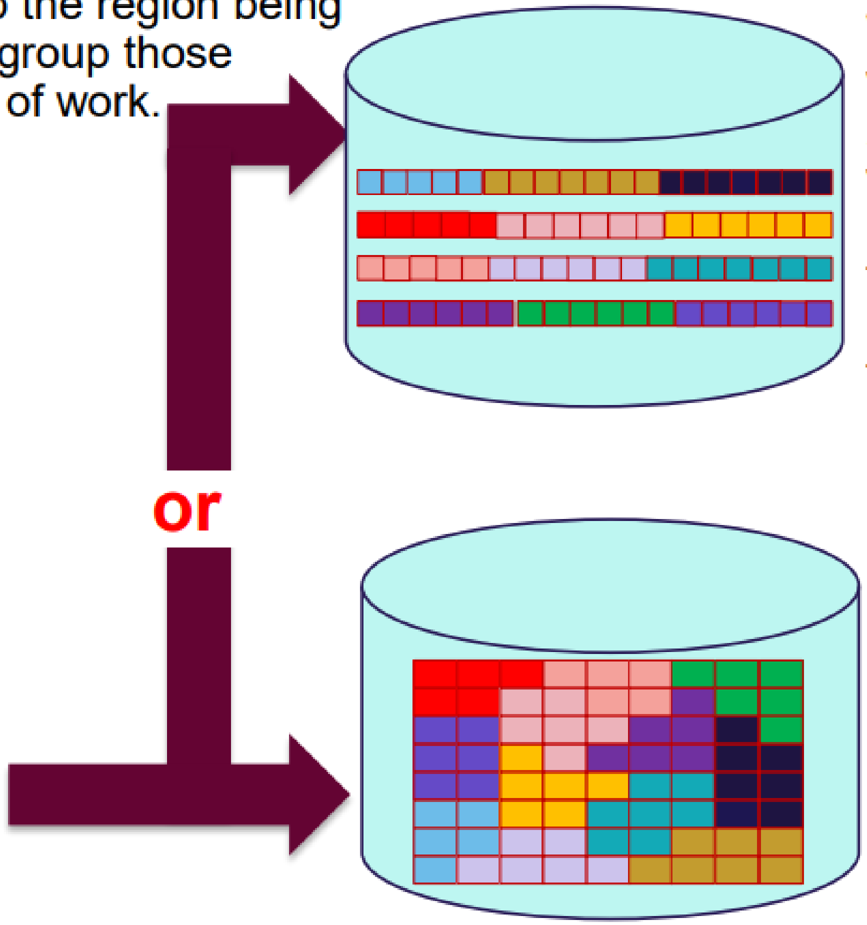


Graphic from J. Tannahill, LLNL

Typical simulations divide up the region being simulated into chunks, then group those chunks into similar amounts of work.

These regions are then distributed to cores (columns) on nodes (grey boxes) for computation.





When speed of writing is the priority, ***blobs*** of data are written from each node into individual files that must then be post-processed for analysis.

To prepare data for analysis, a code can write in a ***canonical*** view by processing the data while it is in memory, resulting in a better organized dataset.

High Data Throughput

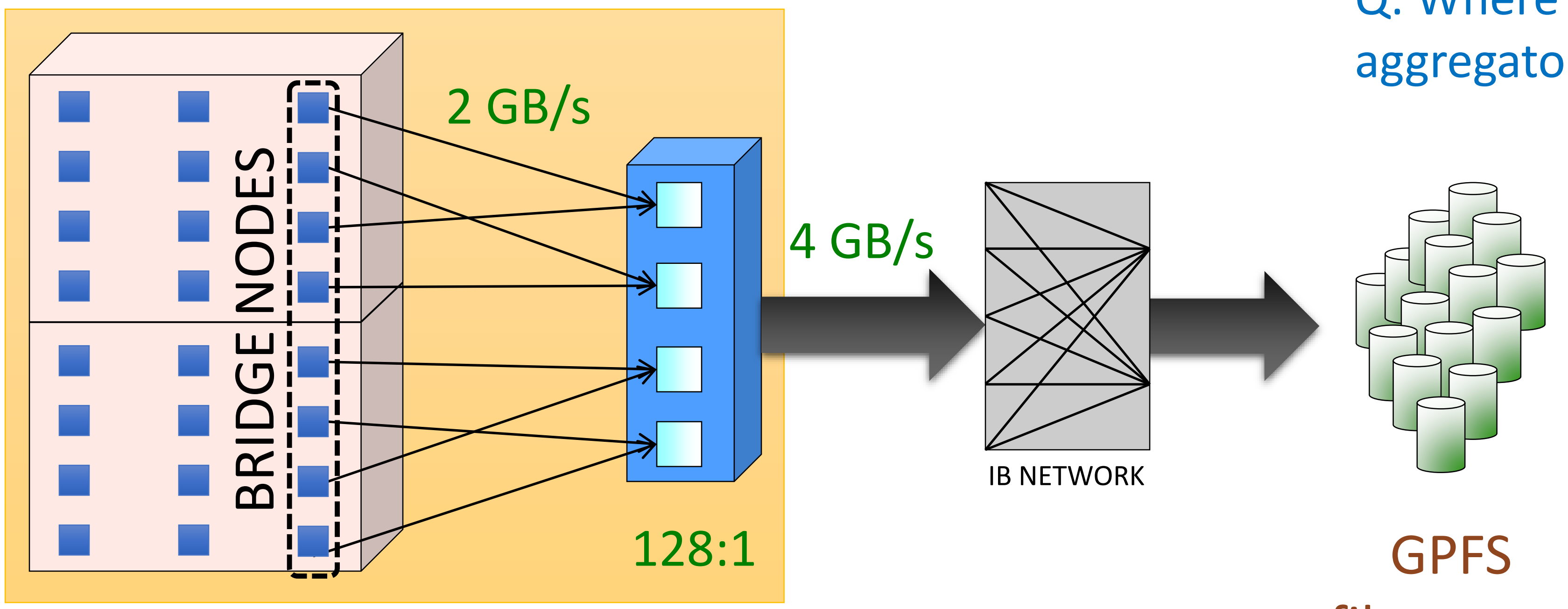
How?

- I/O forwarding from compute to I/O nodes
- Multiple I/O servers, each manage a set of disks
- A large file may be striped across several disks

BG/Q – I/O Node Architecture

Q: Where should the aggregators be placed?

512 compute nodes



GPFS filesystem

Compute node rack

1024 compute nodes
16 bridge nodes

I/O nodes

2 bridge nodes
connect to 1 I/O node