Parallel I/O - II

Feb 8, 2019

Independent I/O – Recap

- Individual file pointers
  - Explicit offsets

MPI_File_read/MPI_File_read_at

- 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

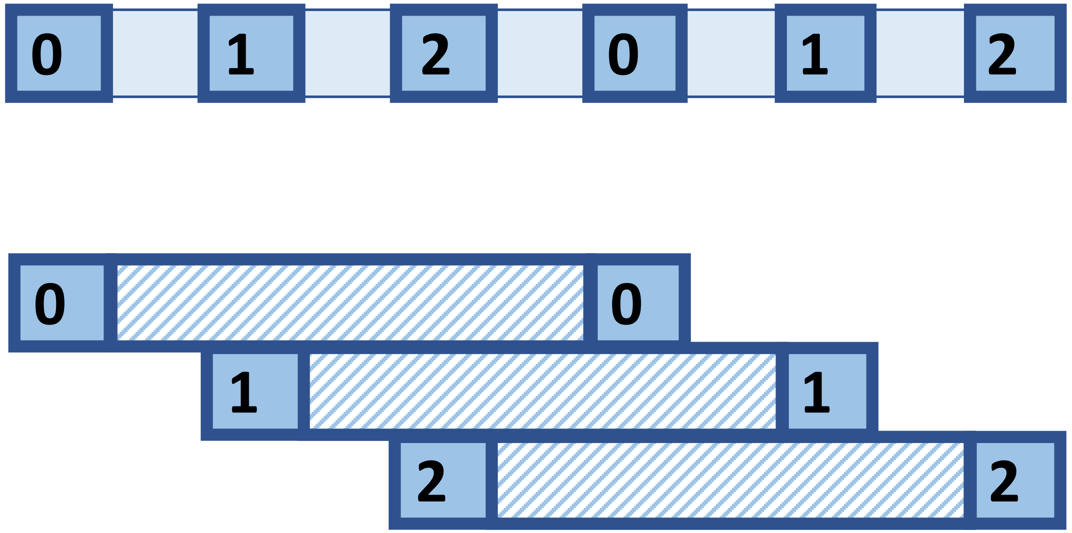
Collective I/O – Recap

- Individual file pointers — MPI_File_read_all
- Shared file pointers — MPI_File_read_ordered/MPI_File_write_ordered

Cons?
Requires synchronization

Parallel I/O – Recap

MPI_Init
MPI_File_open
MPI_Type_vector
MPI_Type_commit
MPI_File_set_view
MPI_File_read_all
MPI_File_close
MPI_Type_free
MPI_Finalize



2D array I/O

MPI_File_open
MPI_Type_create_vector
MPI_Type_commit
MPI_File_set_view
MPI_File_write_all
MPI_File_close

0	1	2
3	4	5
6	7	8

Darray

MPI_File_open

MPI_Type_create_darray

MPI_Type_commit

MPI_File_set_view

MPI_File_write_all

MPI_File_close

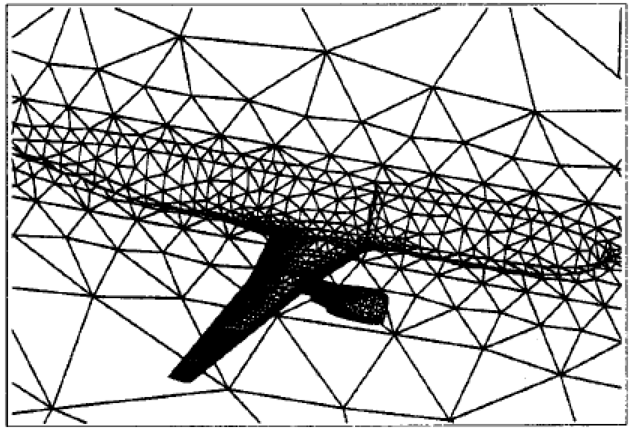
0	**1**	**2**
3	**4**	**5**
6	**7**	**8**

Performance Comparison

Read performance in unstructured grid applications

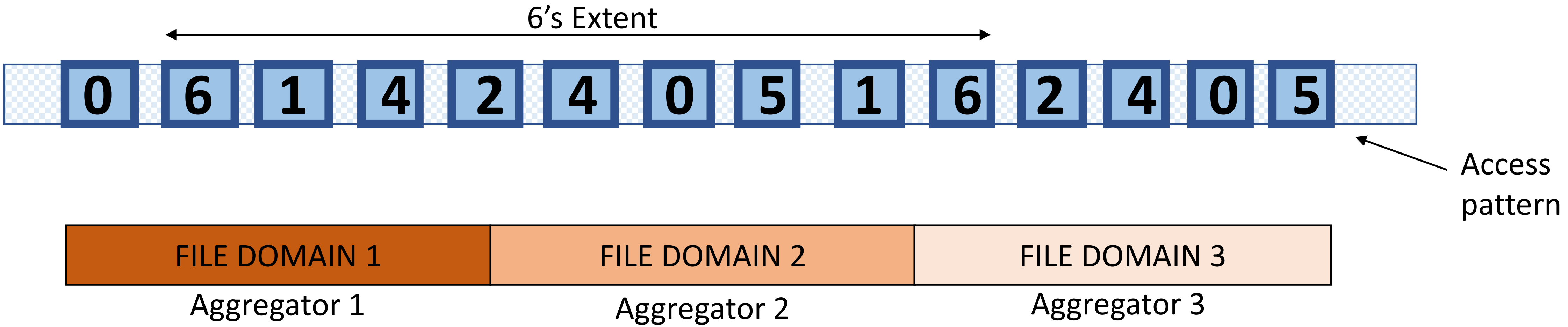
			Bandwidth (Mbytes/s)	
Machine	**Processors**	**Grid Points**	**Level 2**	**Level 3**
HP Exemplar	64	8 million	3.15	35.0
IBM SP	64	8 million	1.63	73.3
Intel Paragon	256	8 million	1.18	78.4
NEC SX-4	8	8 million	152	101
SGI Origin2000	32	4 million	30.0	80.8

Source: Thakur et al., A Case for Using MPI's Derived Datatypes to Improve I/O Performance, SC98



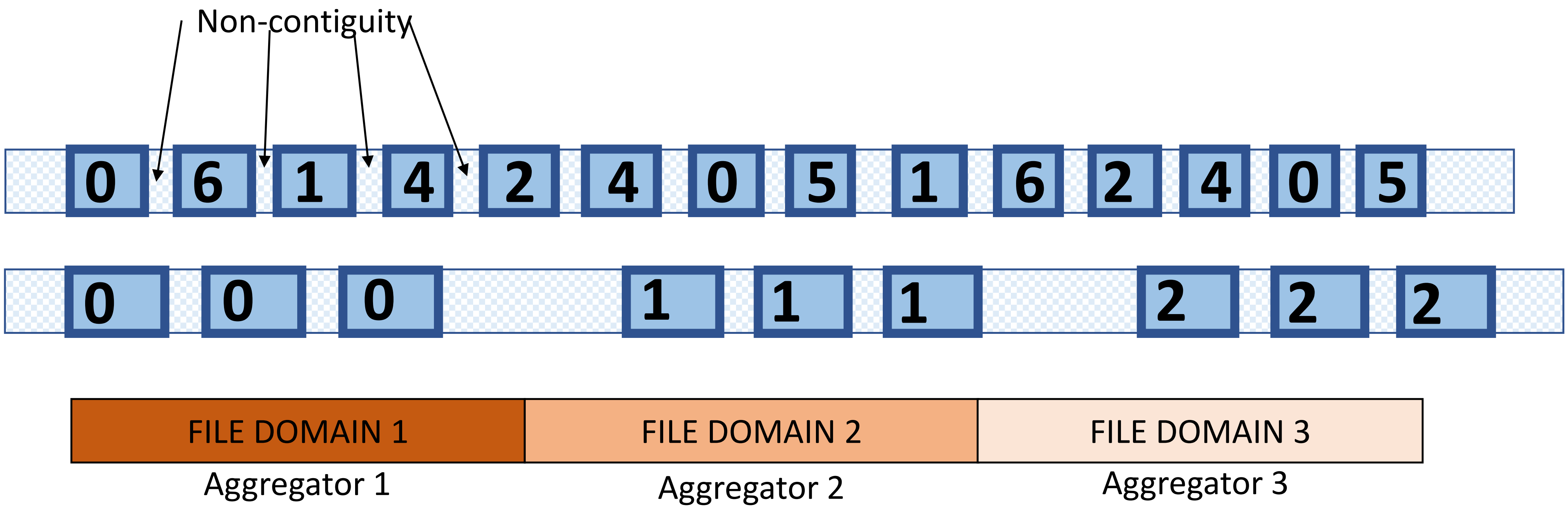
Example of unstructured mesh [Mavriplis et al.]

Collective I/O Summary



- 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

I/O Aggregators



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

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

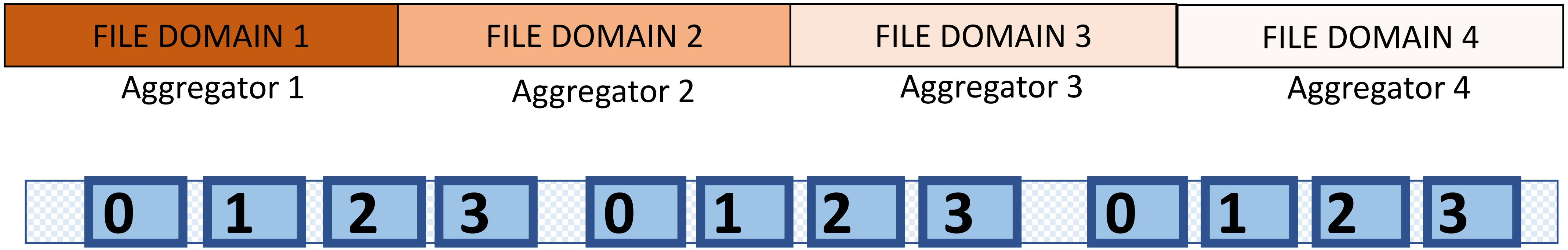
- Collective I/O may be done in several iterations
- Double buffering may help

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

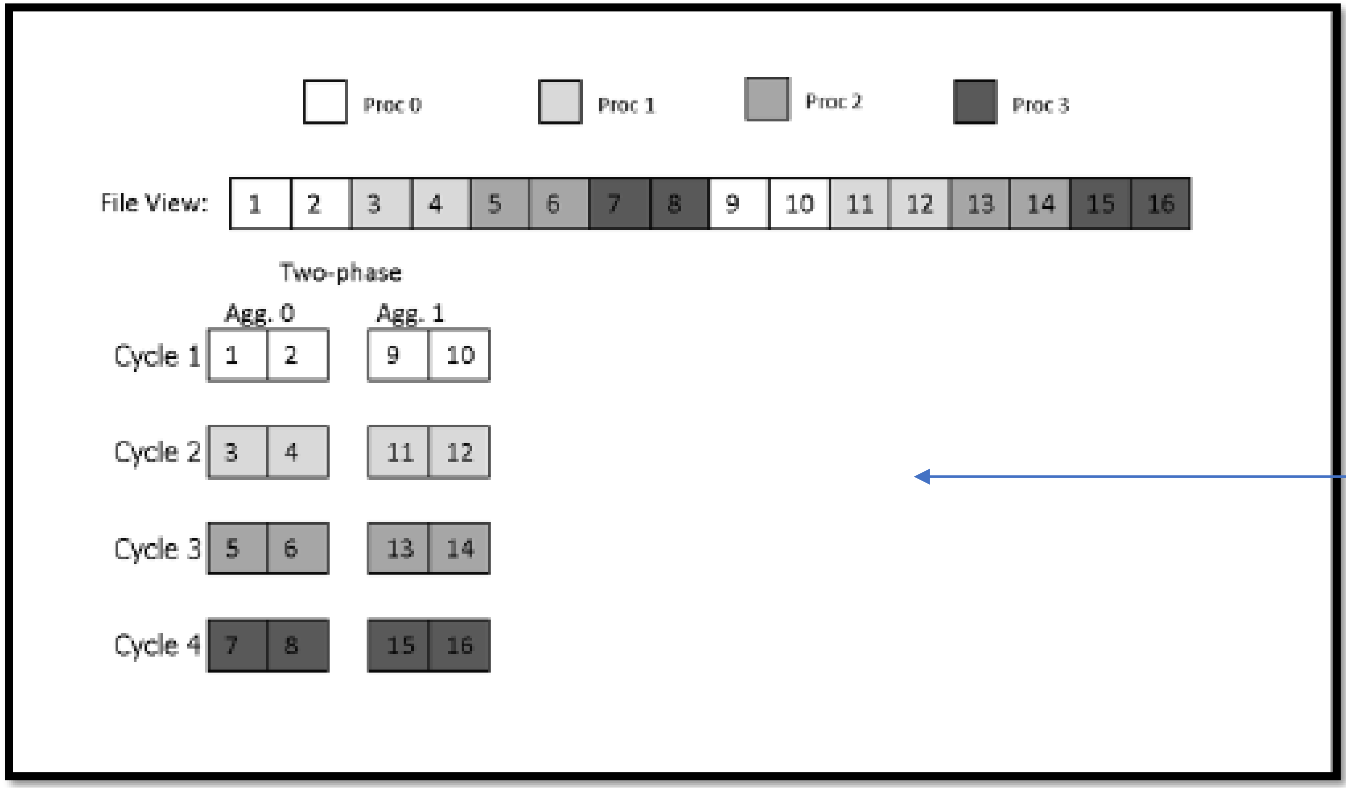
I/O Aggregators



It is possible that during one of the iterations

- One or more aggregators do not perform the I/O operation
- Processes may or may not receive data

Collective I/O – extensions



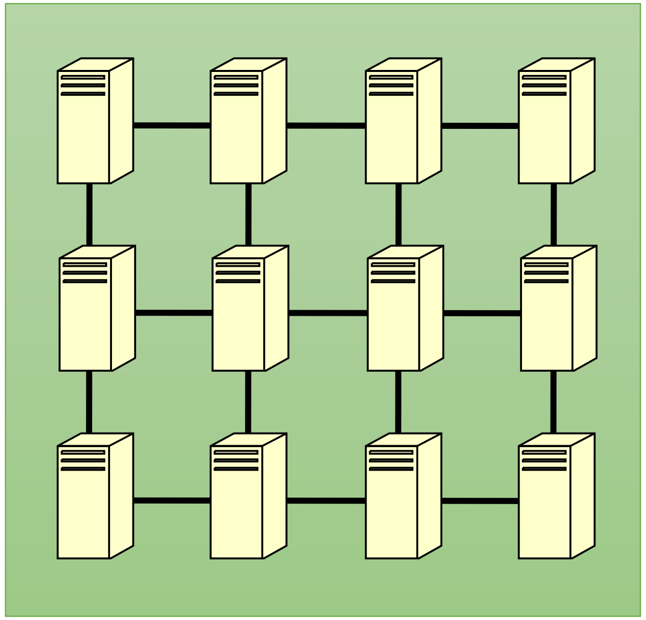
In case of dynamic segmentation, with what can you can replace Alltoallv?

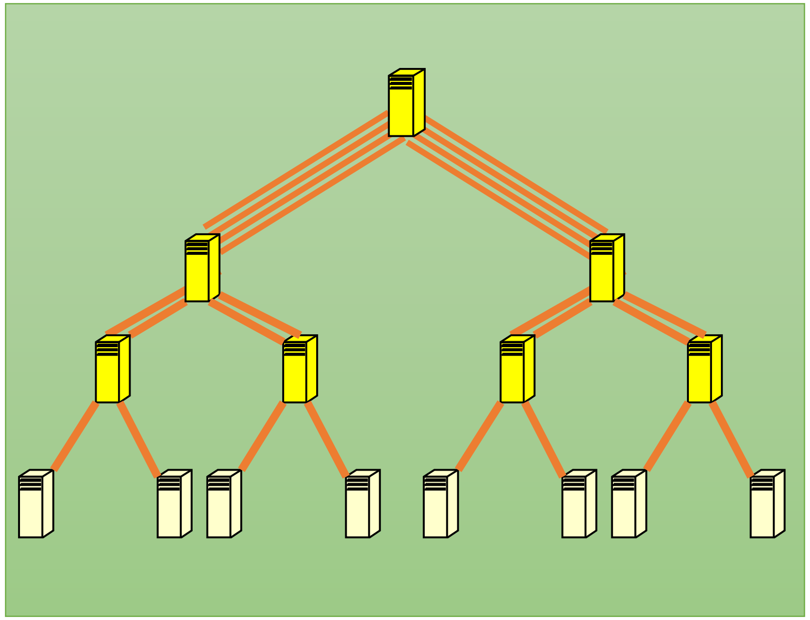
Source: Chaarawi et al., Automatically Selecting the Number of Aggregators for Collective I/O Operations, CC11

Aggregators

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.





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)

User-controlled Parameters

- Number of aggregators (cb_nodes)
- Placement of aggregators (cb_config_list)
- Buffer size in aggregators (cb_buffer_size)
- …

- Can be set via hints
- MPI_Info object is used to pass hints

MPI_Info – Example

```
MPI_Info_create(&info);
MPI_Info_set(info, "cb_nodes", "8");

…

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

MPI_Info

```
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);
}
```

Non-blocking Independent I/O

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

Split Collective I/O

MPI_File_write_all_begin (MPI_File fh, void *buf, int count, MPI_Datatype datatype)

/* computation */

MPI_File_write_all_end (MPI_File fh, void *buf, MPI_Status *status)

Note: Overlapping split collective I/O operations are not allowed

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

High Data Throughput

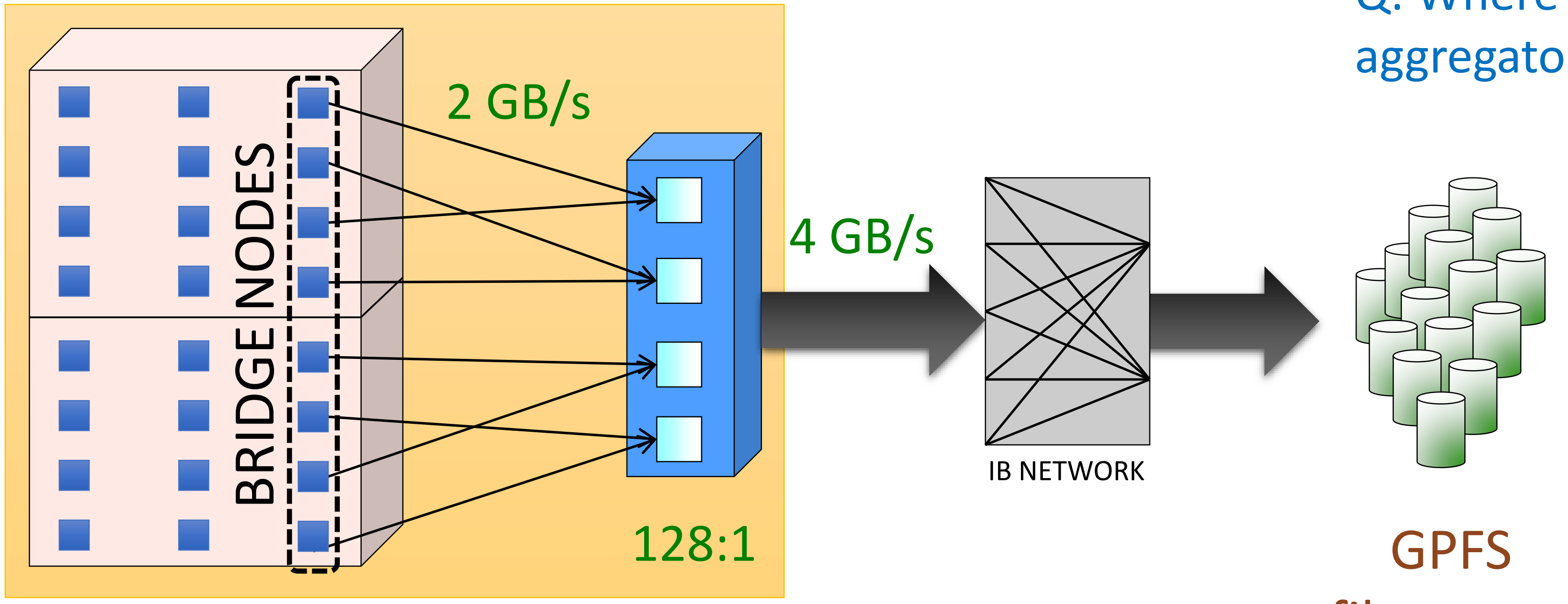
How?

- 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