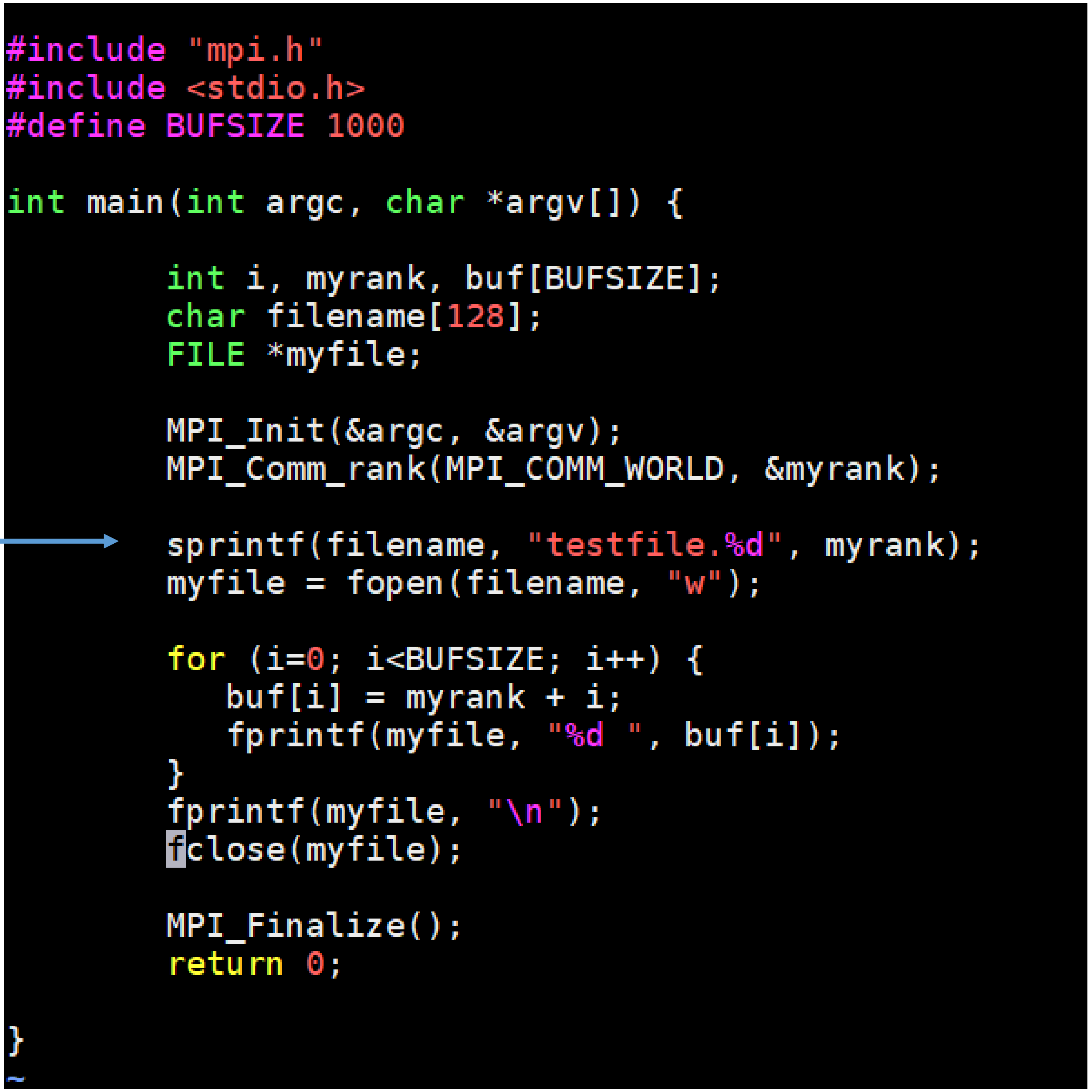
Parallel I/O – II and SLURM

Lecture 17

March 26, 2025

Revision of MPI Independent I/O

Write to Different Files



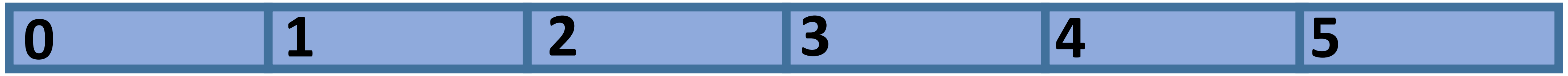
Independent writes

```
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 1000

int main(int argc, char *argv[]) {

        int i, myrank, buf[BUFSIZE];
        char filename[128];
        FILE *myfile;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

        sprintf(filename, "testfile.%d", myrank);
        myfile = fopen(filename, "w");

        for (i=0; i<BUFSIZE; i++) {
           buf[i] = myrank + i;
           fprintf(myfile, "%d ", buf[i]);
        }
        fprintf(myfile, "\n");
        fclose(myfile);

        MPI_Finalize();
        return 0;

}
~
```

Simple Parallel I/O Code – Shared File

```
MPI_File fh
file_size_per_proc = FILESIZE / nprocs
MPI_File_open (MPI_COMM_WORLD, “/scratch/largefile”,
MPI_MODE_RDONLY, MPI_INFO_NULL, &fh)
MPI_File_seek (fh, rank*file_size_per_proc, MPI_SEEK_SET)
MPI_File_read (fh, buffer, count, MPI_INT, status)
MPI_File_close (&fh)
```

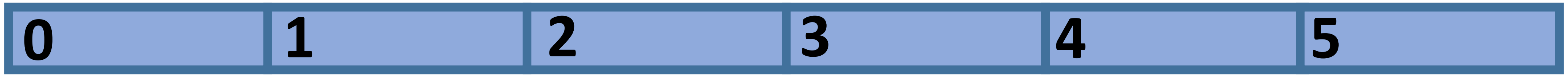
Returns file handle



Parallel Read using Explicit Offset

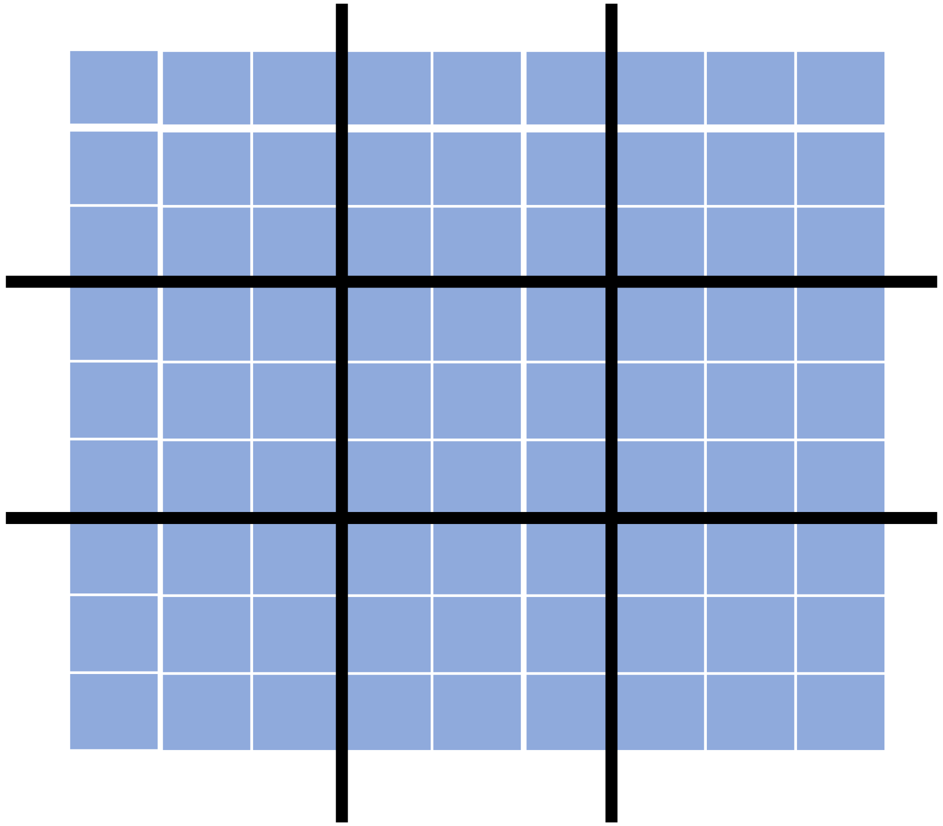
```
MPI_Offset offset = (MPI_Offset) rank*file_size_per_proc*sizeof(int)
MPI_File_open (MPI_COMM_WORLD, “/scratch/largefile”,
MPI_MODE_RDONLY, MPI_INFO_NULL, &fh)
MPI_File_read_at (fh, offset, buffer, count, MPI_INT, status)
MPI_File_close (&fh)
```

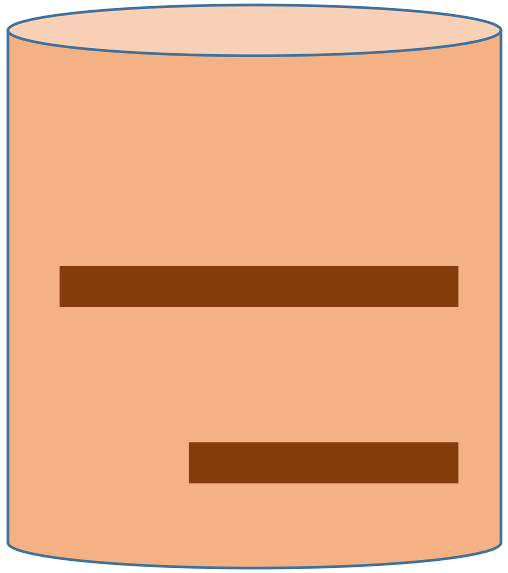
Every process reads a contiguous chunk

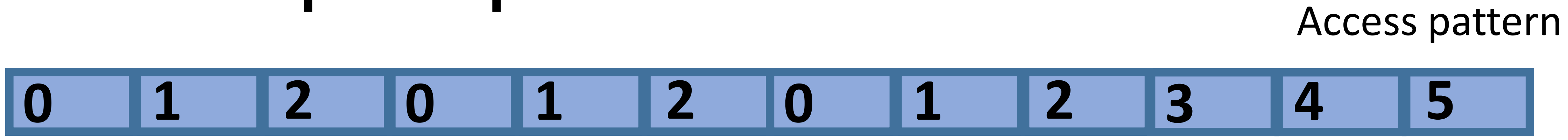


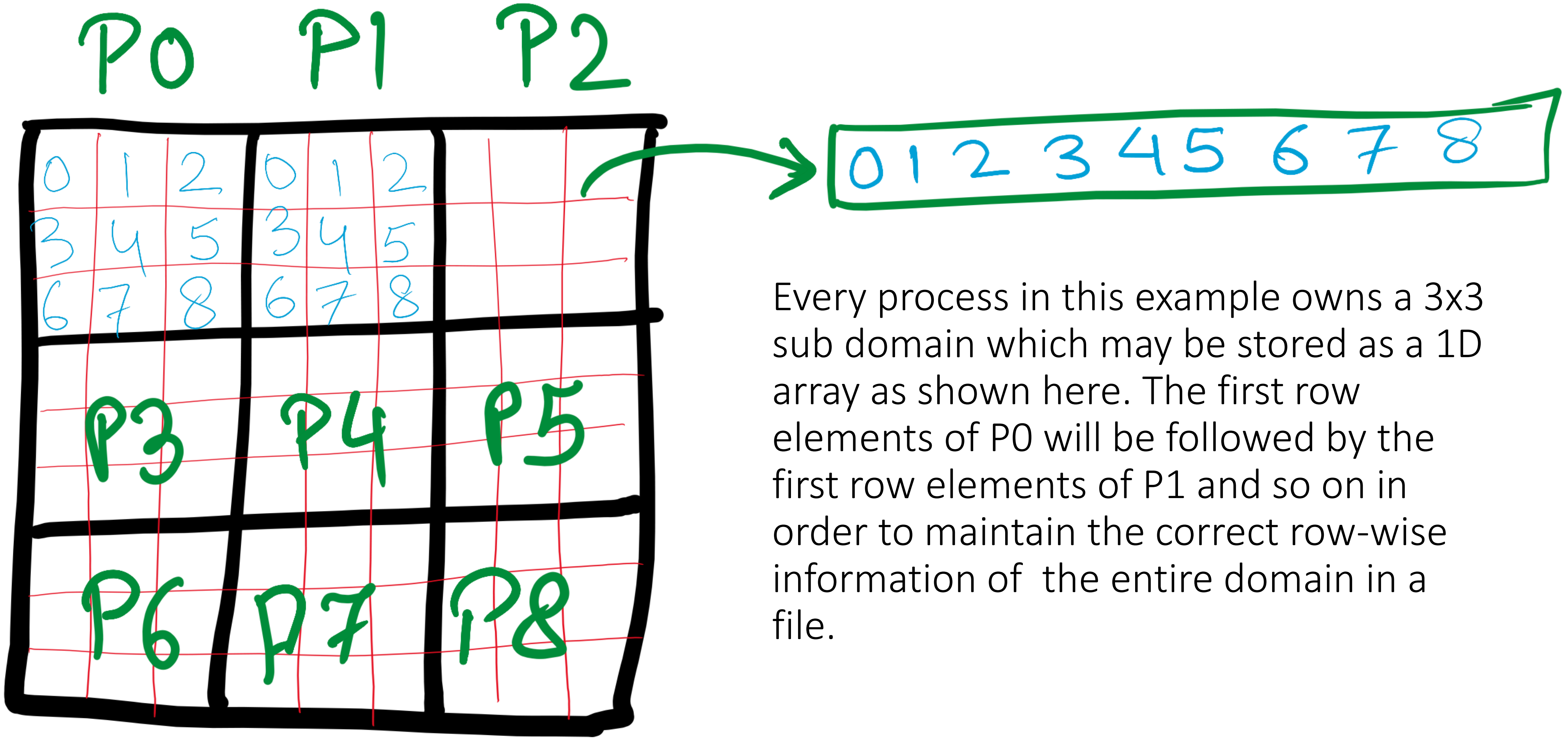
```c
MPI_File fh;  // FILE

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

for (int i=0; i<BUFSIZE ; i++)
  buf[i]=i;

strcpy(filename, "testfileIO");

// File open, fh: individual file pointer
MPI_File_open (MPI_COMM_WORLD, filename, MPI_MODE_CREATE | MPI_MODE_RDWR, MPI_INFO_NULL, &fh);

MPI_Offset fo = (MPI_Offset) myrank*BUFSIZE*sizeof(int);

// File write using explicit offset (independent I/O)
MPI_File_write_at (fh, fo, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE); //fwrite

// File read using explicit offset (independent I/O)
MPI_File_read_at (fh, fo, rbuf, BUFSIZE, MPI_INT, &status);   //fread

MPI_File_close (&fh);         //fclose

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

Large Domain



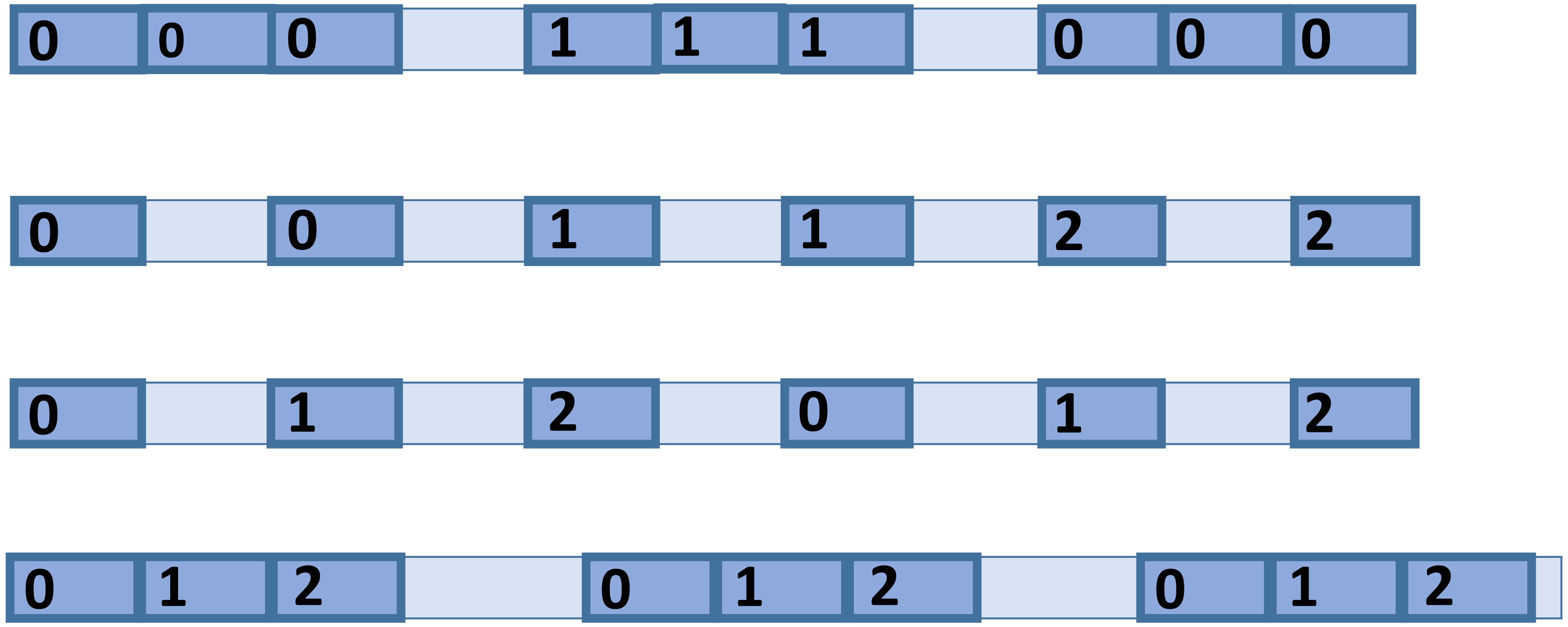






Every process in this example owns a 3x3 sub domain which may be stored as a 1D array as shown here. The first row elements of P0 will be followed by the first row elements of P1 and so on in order to maintain the correct row-wise information of the entire domain in a file.

Non-contiguous Access Patterns



Multi-variable Dataset

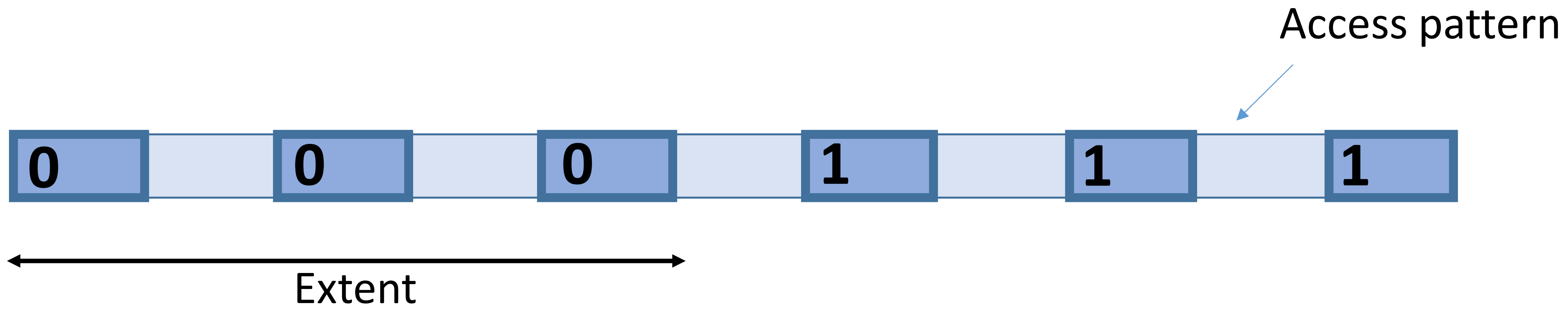
Two ways to store multiple fields/variables

- Var1 for the entire domain, followed by Var2 for the entire domain, followed by Var3 for the entire domain and so on
- Var1, Var2, Var3, … for grid point 1, Var1, Var2, Var3, … for grid point 2 and so on

Every process analytically computes the offsets in both cases

HW: Write the code to write 4 variables of a MxN domain from PxQ processes using the above two approaches.

Multiple Short Accesses

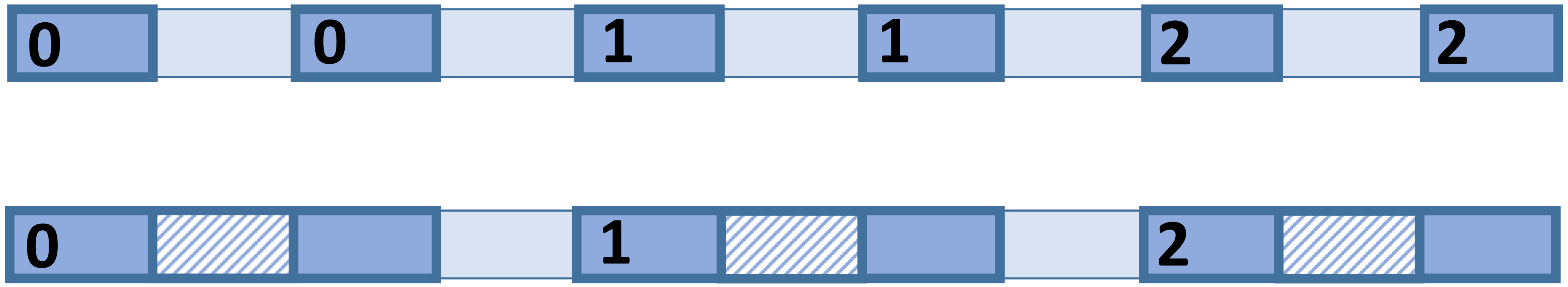


```
MPI_File_read_at (fh, offset1, buffer1, count1, MPI_INT, status)
MPI_File_read_at (fh, offset2, buffer2, count2, MPI_INT, status)
MPI_File_read_at (fh, offset3, buffer3, count3, MPI_INT, status)
```

What is the problem with non-contiguous accesses?

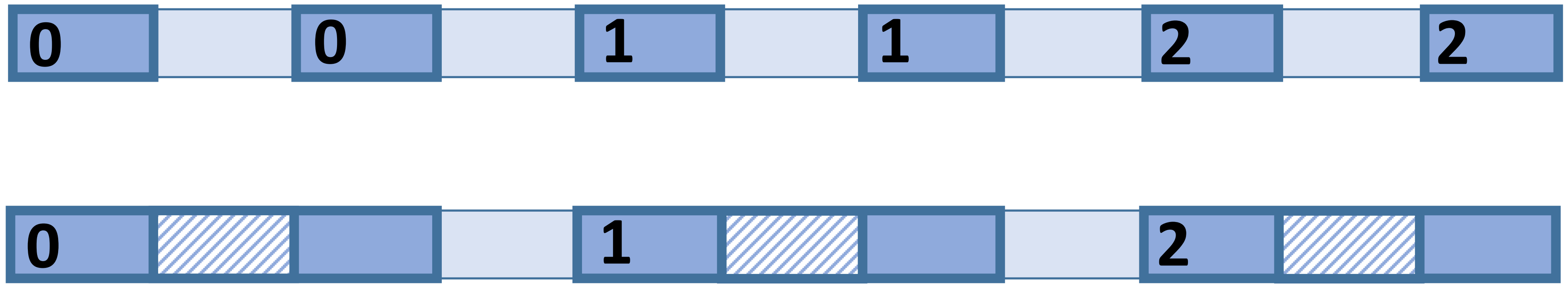
Can we instead use one read call?

Optimization – Data Sieving



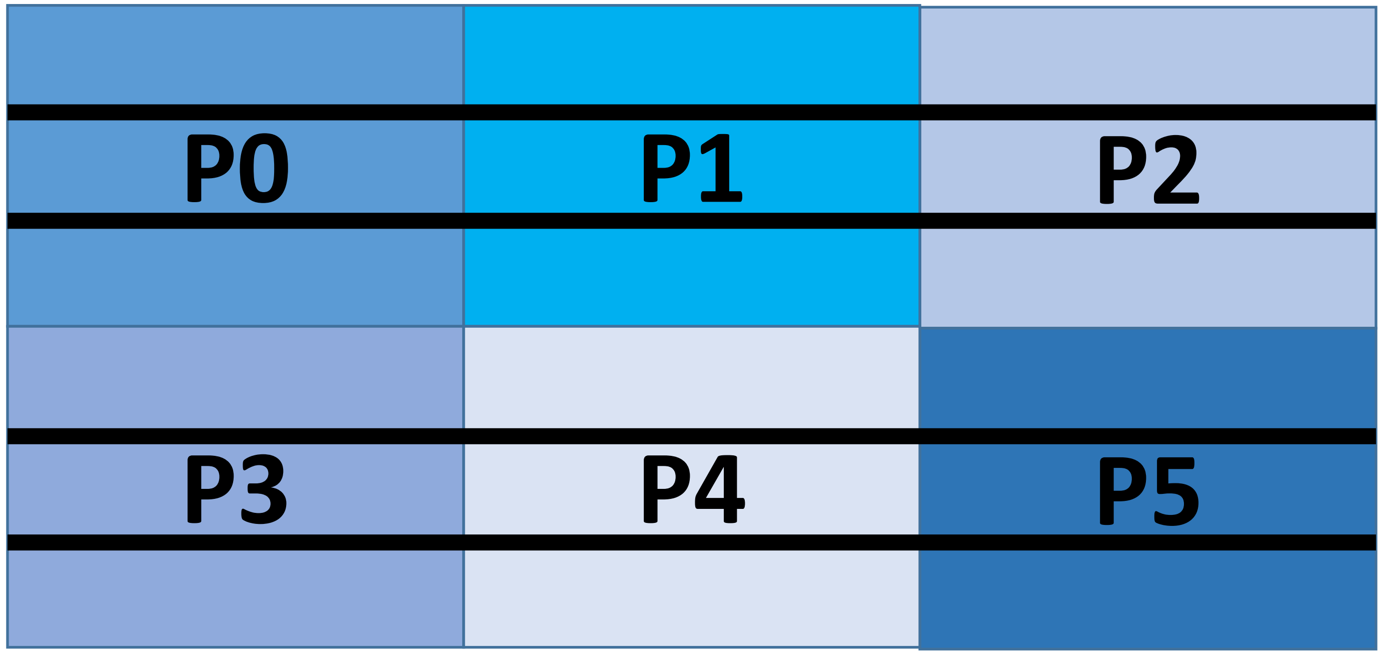
- Make large I/O requests and extract the data that is really needed
- Huge benefit of reading large, contiguous chunks

Data Sieving for Writes



- Copy only the user-modified data into the write buffer
- Write only the data that was modified – read-modify-write

Multiple Non-contiguous Accesses

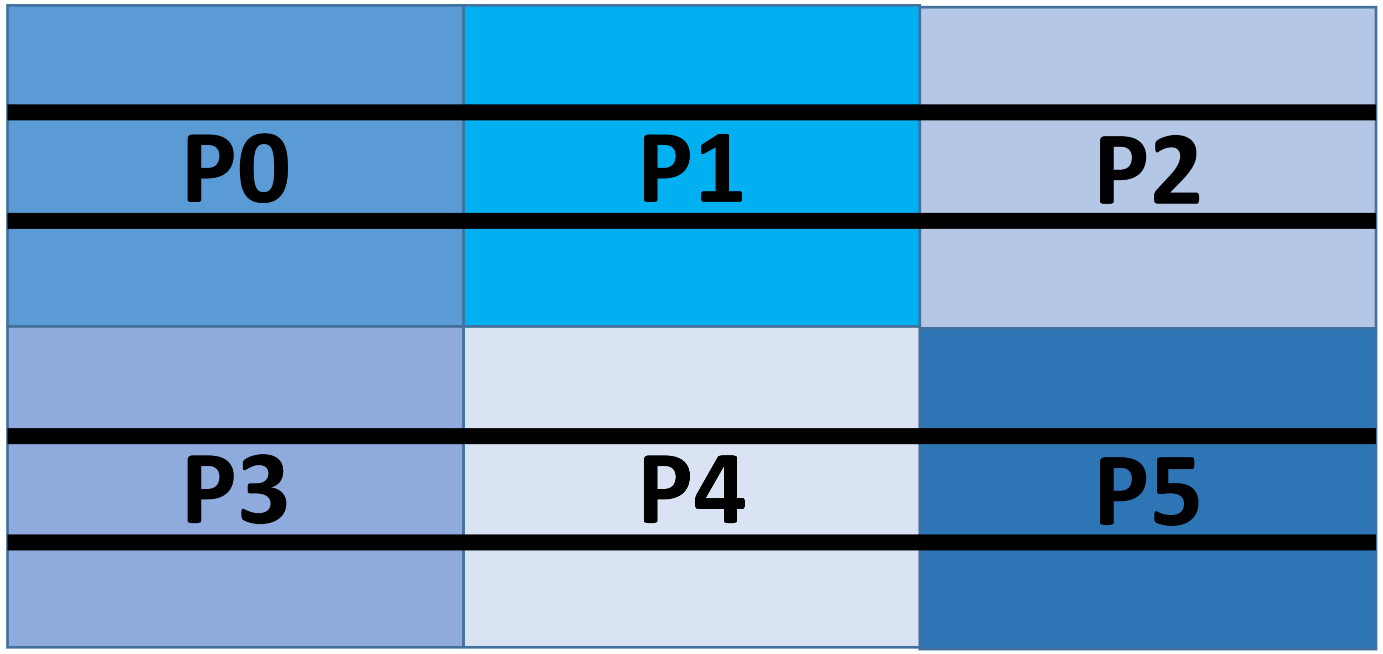


Can some of these requests be merged?

What is the access pattern?

P0 P1 P2 P0 P1 P2 P0 P1 P2 P3 P4 P5 P3 P4 P5 P3 P4 P5

Create Sub-groups for Actual I/O



Can some of these requests be merged?

P0 P1 P2 P0 P1 P2 P0 P1 P2 P3 P4 P5 P3 P4 P5 P3 P4 P5

P0 ………………………………… P3 …………………………………

Create Sub-groups for Actual I/O

P0 P1 P2 P0 P1 P2 P0 P1 P2 P3 P4 P5 P3 P4 P5 P3 P4 P5

P0 ……………………………….…… P3 ………………………………………

Two leaders created

Data per process = 1 MB

P0 P1 P2 P0 P1 P2 P0 P1 P2 P3 P4 P5 P3 P4 P5 P3 P4 P5

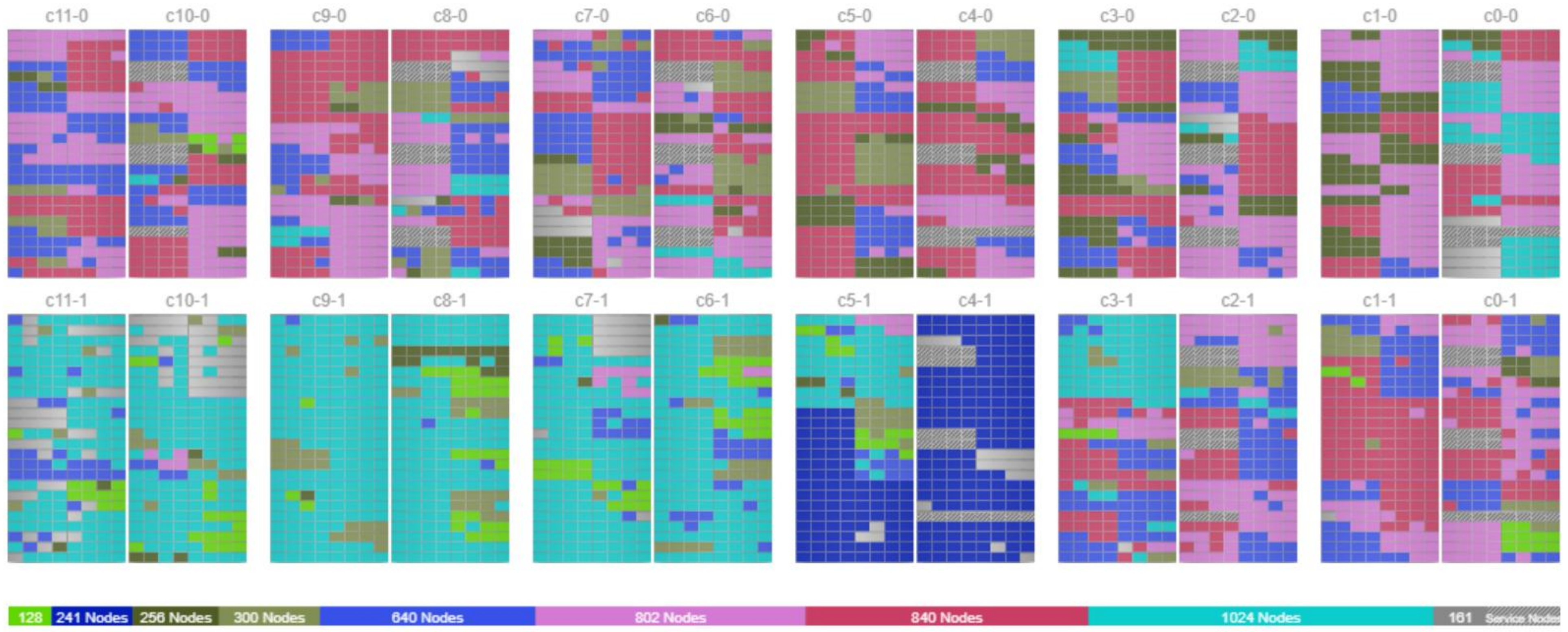
Assign more leaders

Data per process = 1 GB

P0 P1 P2 P3 P4 P5 P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P6 P7 P8 P9 P10 P12

P0 and P3 ………………………………………..P6 and P9…………………………………………….

Supercomputer Job Allocation



Running | Starting | Queued | Reservations

Total Running Jobs: 8

Job Id		Project	Nodes	Start Time	Run Time	Walltime	Queue	Mode
512304	🟦	EstopSim_2	1024	9:13:30 AM	00:12:13	16:00:00	default	script
512623	🟥	TurbShockWalls	840	7:57:42 AM	01:28:01	1d 00:00:00	default	script
498557	🟪	TurbShockWalls	802	9:43:57 PM	11:41:46	1d 00:00:00	default	script
513358	🟦	PSFMat_2	640	8:29:27 AM	00:56:16	12:00:00	default	script
511830	🟫	ReconDepth	300	7:50:53 AM	01:34:50	06:00:00	default	script
514000	🟫	HighLumin	256	8:50:22 AM	00:35:21	06:00:00	default	script
514114	🟦	CVD_CityCOVID	241	1:28:47 AM	07:56:56	1d 12:00:00	CVD_Research	script
514178	🟩	FDTD_Cancer_2a	128	8:00:51 AM	01:24:52	03:00:00	default	script

status.alcf.anl.gov -> Theta (retired)

Resources Required

- Number of nodes
- Wall-clock time
- Users are charged for node-hours

Should there be any constraints on the above requirements?

User Jobs

- Different types of applications
- Interactive vs. batch jobs
  - Debug in interactive mode
- Exclusive vs. shared access
- Charged based on total resource usage
  - Job is killed when requested wall-clock time is over
  - Need to plan resource usage apriori

Batch Queueing Systems

- Schedules jobs based on queues
- Has full knowledge of queued, running jobs
- Has full knowledge of the resource usage
- Often combination of best fit, fair share, priority-based
- Designed to be generic, can be customized
- Suited to meet demands of the scheduling goals of the centre
- Typically FIFO/FCFS with backfilling

Workload managers/Schedulers

- Portable Batch System (PBS)
- LoadLeveler
- Application Level Placement Scheduler (ALPS)
- Moab/Torque
- Simple Linux Utility for Resource Management (SLURM)

Example Batch Scheduler

- Network Queueing System developed at NASA
- Supported multiple queues of several types
- Disable/enable each queue
- Tune the #jobs running in each queue

Henderson, “Job Scheduling Under the Portable Batch System”, JSSPP 1995.

Portable Batch Scheduler

- Genesis of PBS in NASA (from NQS)
- Client commands for submission, modification, and monitoring jobs
- Daemons running on service nodes, compute nodes, and servers

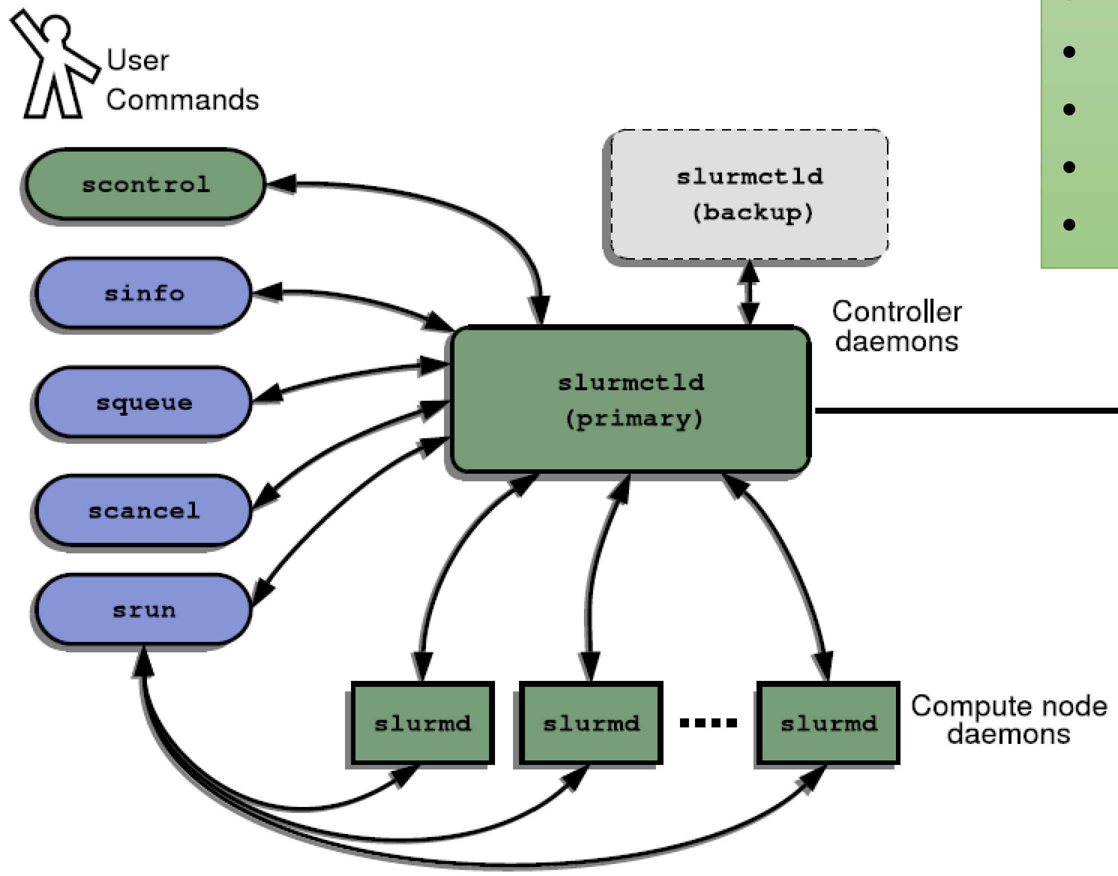
SLURM

- Monitors states of nodes
- Accepts job requests
- Maintains queue of requests
- Schedules jobs
- Initiates job execution and cleanup
- Polls slurmd periodically
- Maintains complete state information

qstat (PBS)

qdel (PBS)

qsub (PBS)



SLURM architecture [Jette et al.]

- Responds to controller requests
- Maintains job state
- Initiate, manage, cleanup processes
- I/O handling

PARAM Rudra C-DAC (IUAC Delhi)

- Thanks to C-DAC
- Based on Intel Xeon
- 100 Gbps Mellanox Infiniband interconnect (fat-tree topology)
- Use a maximum of 2 nodes and 48 cores for a maximum wall-clock time of 10 minutes (for now)
- SLURM scheduler

SLURM Commands

```
#SBATCH –N 2
#SBATCH --ntasks-per-node=2
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --time=00:10:00
#SBATCH --partition=standard
mpirun –np 4 ./exe <args>
```

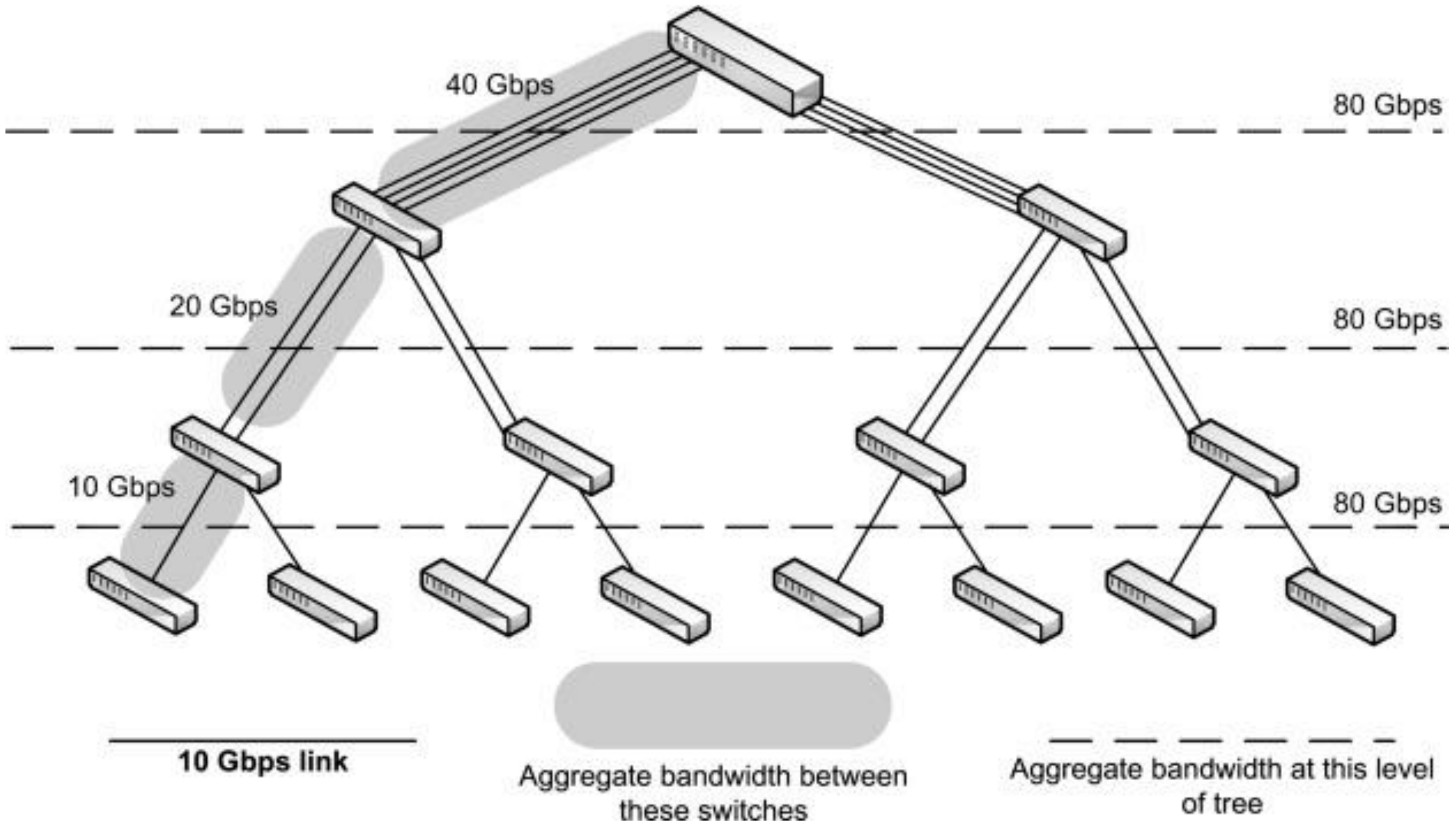
Job Script

```
sbatch <jobscript>
squeue
scancel <jobid>
```

squeue

```
squeue

JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          12181       cpu    QE_FC abc  R    1:21:59     16 rdcn[17-32]

squeue --me

JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

Fat-tree Topology


40 Gbps
80 Gbps
20 Gbps
80 Gbps
10 Gbps
80 Gbps
10 Gbps link
Aggregate bandwidth between these switches
Aggregate bandwidth at this level of tree

Job Allocation

Assume #nodes per rack = 20

Job 1: cn[17-32]
Spills from Rack 1 to Rack 2

Job 2: cn[1-2],cn[33-40]
Fragmentation

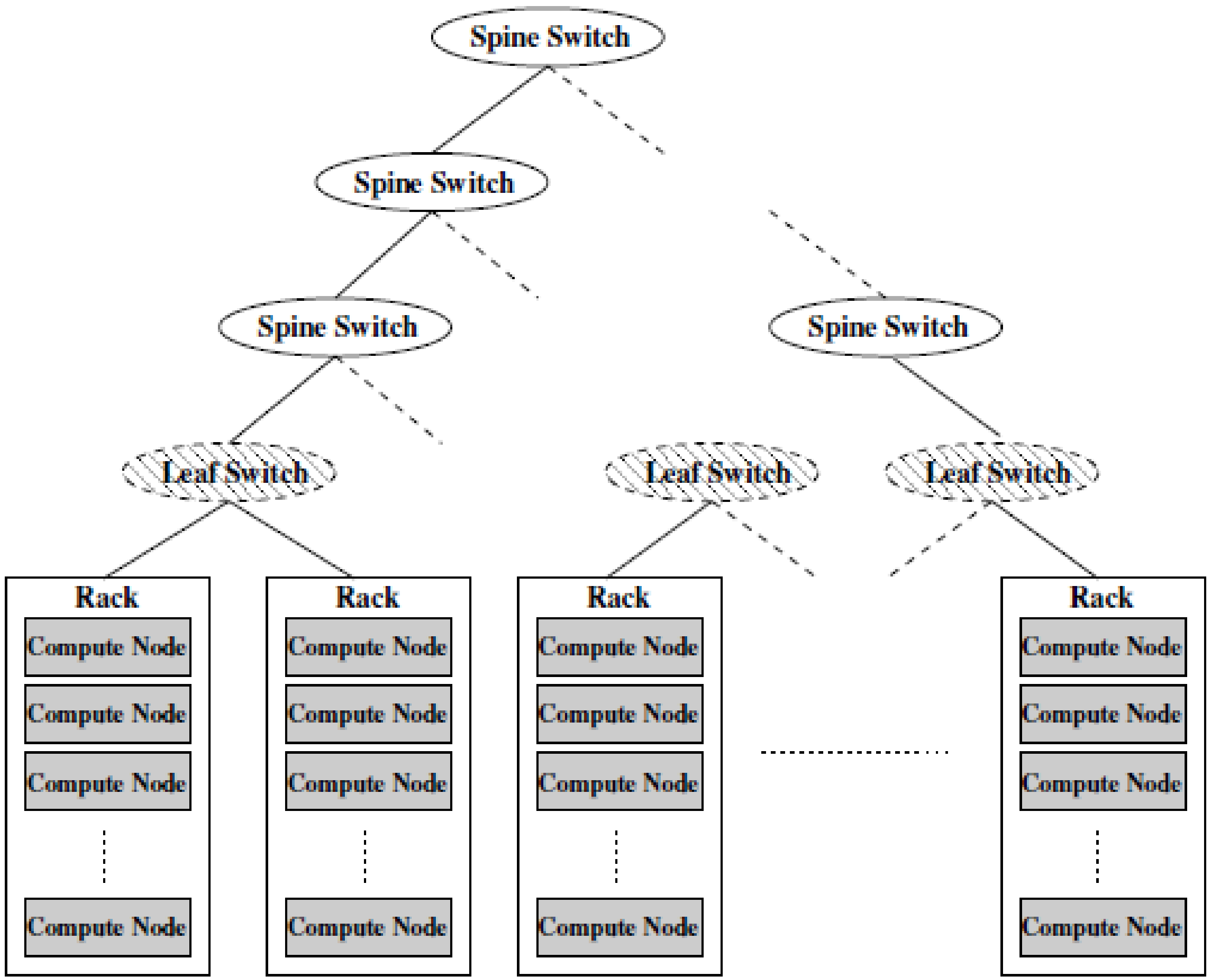
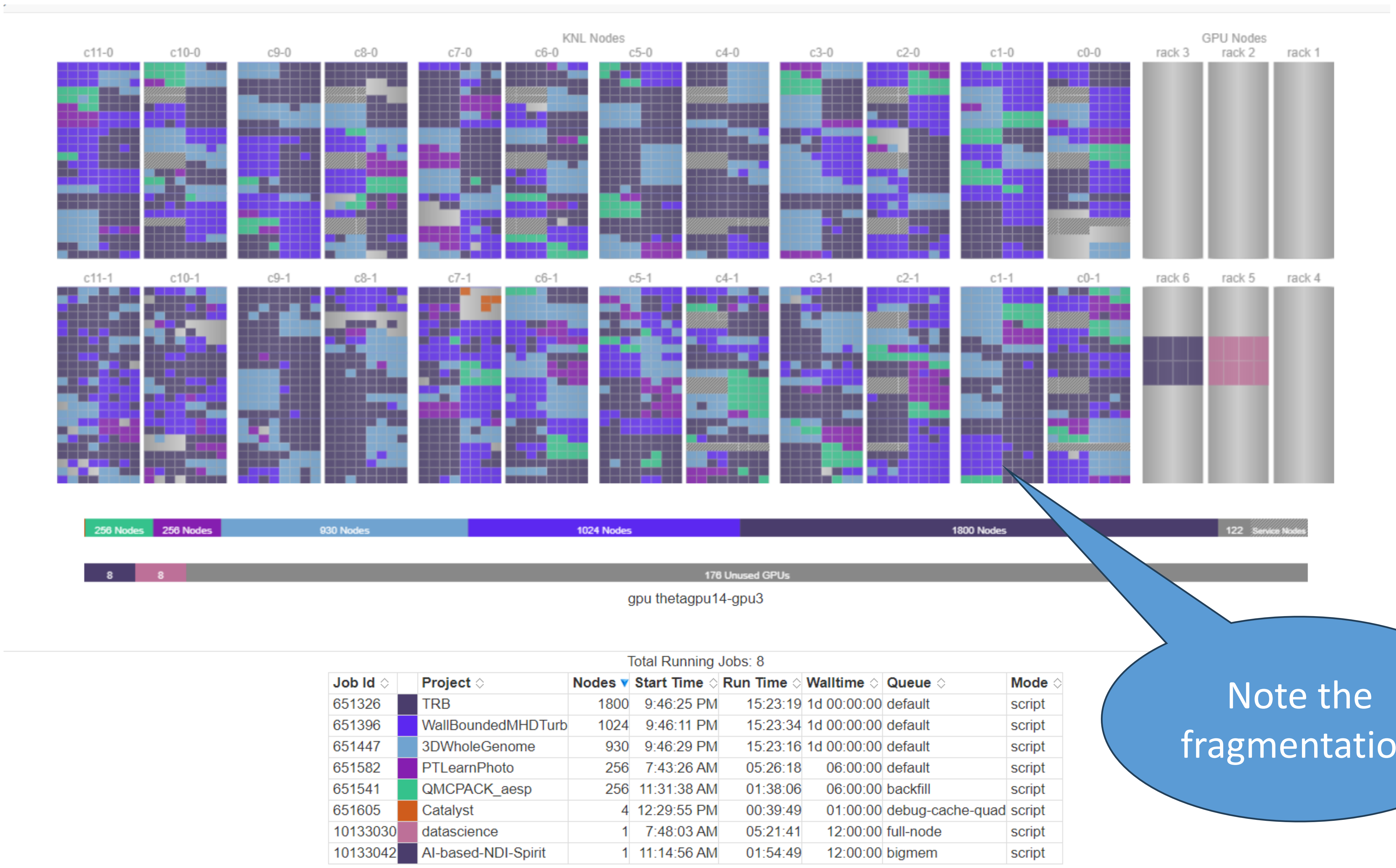


Figure 1. A Typical Topology



Total Running Jobs: 8

Job Id ◇		Project ◇	Nodes ▾	Start Time ◇	Run Time ◇	Walltime ◇	Queue ◇	Mode ◇
651326	🟪	TRB	1800	9:46:25 PM	15:23:19	1d 00:00:00	default	script
651396	🟦	WallBoundedMHDTurb	1024	9:46:11 PM	15:23:34	1d 00:00:00	default	script
651447	🟦	3DWholeGenome	930	9:46:29 PM	15:23:16	1d 00:00:00	default	script
651582	🟪	PTLearnPhoto	256	7:43:26 AM	05:26:18	06:00:00	default	script
651541	🟩	QMCPACK_aesp	256	11:31:38 AM	01:38:06	06:00:00	backfill	script
651605	🟧	Catalyst	4	12:29:55 PM	00:39:49	01:00:00	debug-cache-quad	script
10133030	🟥	datascience	1	7:48:03 AM	05:21:41	12:00:00	full-node	script
10133042	🟪	AI-based-NDI-Spirit	1	11:14:56 AM	01:54:49	12:00:00	bigmem	script

Note the fragmentation

Important Metrics

User

- Job wait time = Job start time – Job submit time
- Job turnaround time = Job end time – Job Submit time

System

- Throughput = Number of jobs completed in a time unit
- Idle nodes = Number of idle nodes in a time unit

PARAM Sanganak (IITK) Partitions (Queues)

Partition Name	Minimum #CPU cores (equivalent nodes)	Maximum #CPU cores (equivalent nodes)	Maximum Time [Days-HH:MM:SS]	Default Time [HH:MM:SS]	Nodes allocated to each partition (some nodes are shared in more than one partition)	Total Nodes
serial	1	24	02-00:00:00	02:00:00	hm[001-002]	2(hm)
small	48 (1 node)	96 (2 nodes)	02-00:00:00	02:00:00	cn[001-050], hm[003-029]	50(cn)+27(hm)
medium	96 (2 nodes)	192 (4 nodes)	02-00:00:00	02:00:00	cn[051-110], hm[030-054]	60(cn)+25(hm)
large	192 (4 nodes)	480 (10 nodes)	03-00:00:00	02:00:00	cn[111-187], hm[055-078]	76(cn)+24(hm)
fat	480 (10 nodes)	____	02-00:00:00	02:00:00	cn[171-217], hm[055-078]	47(cn)+24(hm)
hm	48 (1 node)	288 (6 nodes)	02-00:00:00	02:00:00	hm[003-078]	76(hm)
gpu	cpu=1,gres/gpu=1	cpu=160,gres/gpu=8	02-00:00:00	02:00:00	gpu[001-020]	20(gpu)

* Maximum #CPU Cores has not been set explicitly in the FAT partition, but it can be dictated by the maximum number of nodes in the partition.