Understanding Parallel IO through profiling

Luca Parisi, EPCC, The University of Edinburgh

l.parisi@epcc.ed.ac.uk

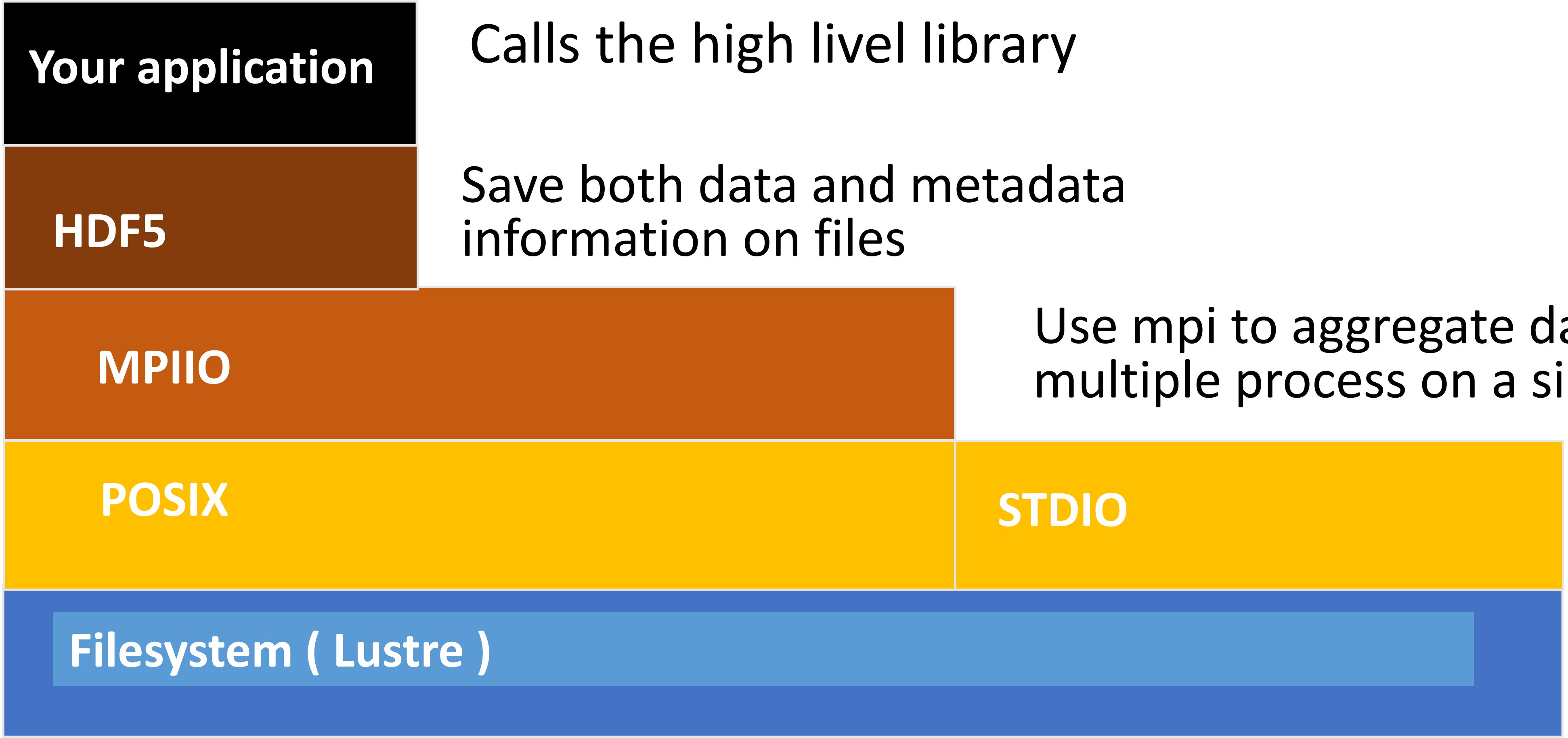
21 February 2020

www.archer2.ac.uk

Outline

- Introduction to the IO software stack
- The Darshan profiler
- Write a 2D array to disk: file per process, single shared file, mpiio
- Netkar++ example

The software stack



Your application

HDF5

MPIIO

POSIX

STDIO

Filesystem (Lustre)

Calls the high livel library

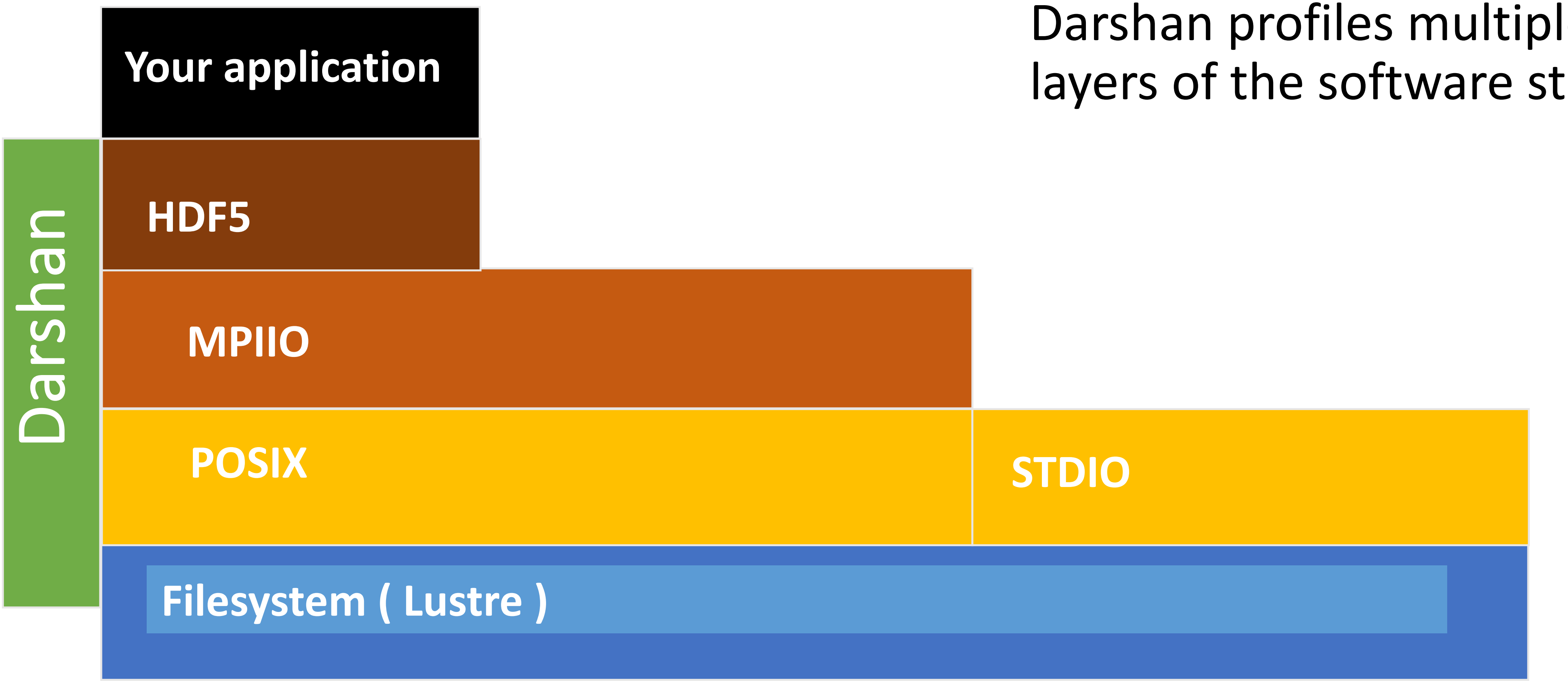
Save both data and metadata information on files

Use mpi to aggregate data from multiple process on a single file

libc libraries, single process calls

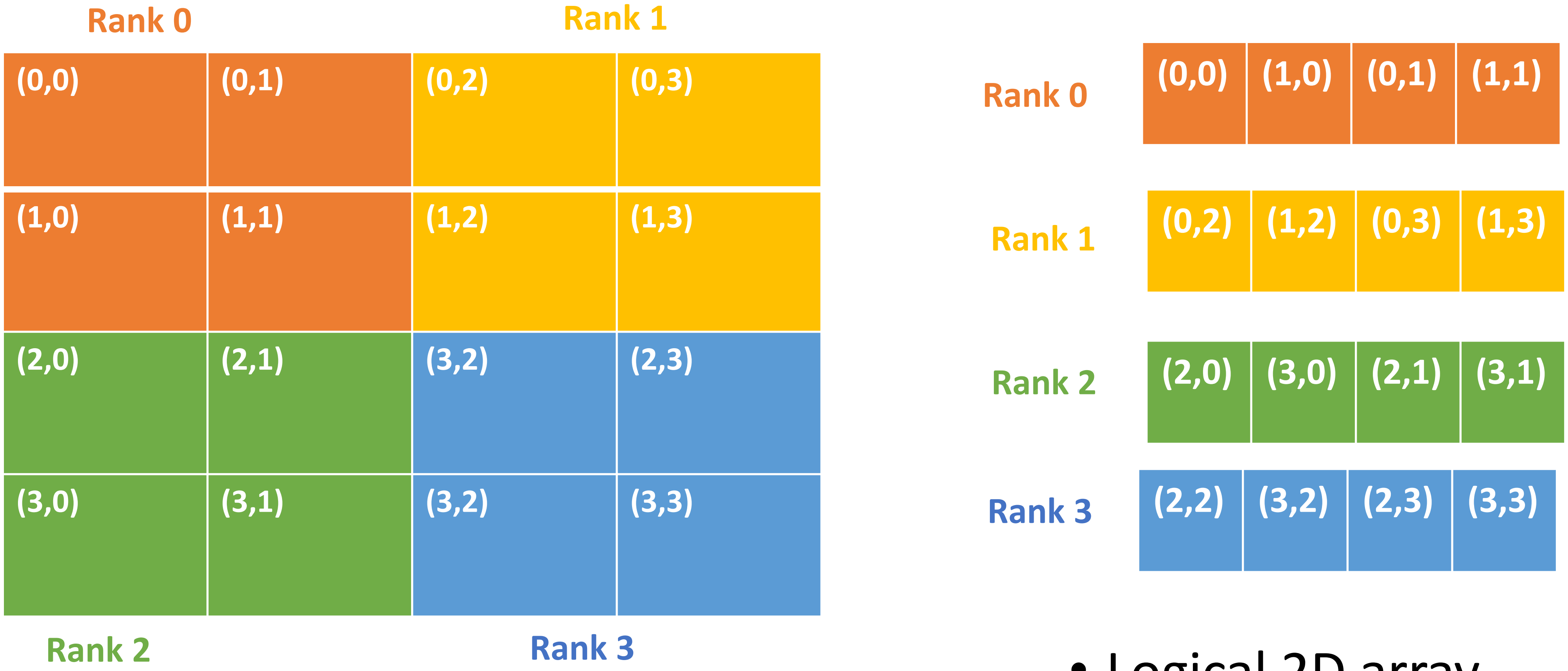
Network file system

The darshan tool



Darshan profiles multiple layers of the software stack

2D Array



Rank 0 | Rank 1

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(3,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Rank 2 | Rank 3

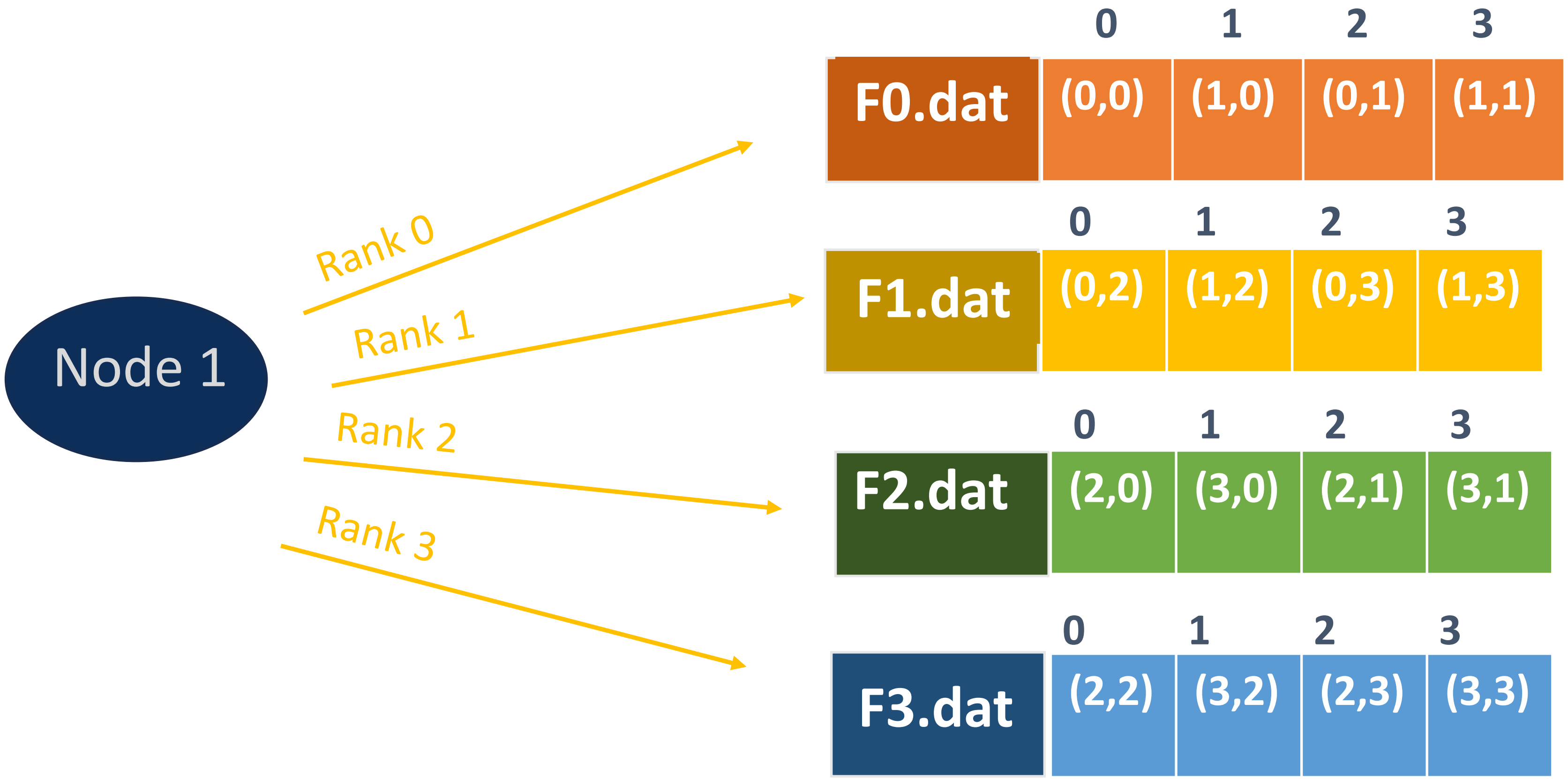
Rank 0	(0,0)	(1,0)	(0,1)	(1,1)
Rank 1	(0,2)	(1,2)	(0,3)	(1,3)
Rank 2	(2,0)	(3,0)	(2,1)	(3,1)
Rank 3	(2,2)	(3,2)	(2,3)	(3,3)

- Logical 2D array
- Column storage (Fortran like)

File per process

File per process

- Each process writes their own data to a different file
- Efficient
- Data management is difficult



Node 1

Rank 0

Rank 1

Rank 2

Rank 3

	0	1	2	3
F0.dat	(0,0)	(1,0)	(0,1)	(1,1)
F1.dat	(0,2)	(1,2)	(0,3)	(1,3)
F2.dat	(2,0)	(3,0)	(2,1)	(3,1)
F3.dat	(2,2)	(3,2)	(2,3)	(3,3)

Write Bandwidth

Using 4 nodes, 10 ranks per node

File		Performance
File Per Process		16 GB/s

Using Darshan

Using Darshan

- Load Darshan

```
module load darshan
```

- Run you executable as usual

```
srun app.exe
```

Any application launched trough srun will be linked to Darshan and profiled.

Finding the darshan profile

Darshan saves all files in a common directory specified during installation.

```
$ darshan-config --log-path
/work/z19/z19/lparisi/courses/io/io_webinar/sw/darshan/darshan-logs
```

The log directory contains subfolders named as year/month/day .

Ex: For a job run on the 23th of January,

the `.darshan` profile can be found in

`${LOG_DIR}/2024/1/23`

Generating a PDF report

- Generate a summary pdf report

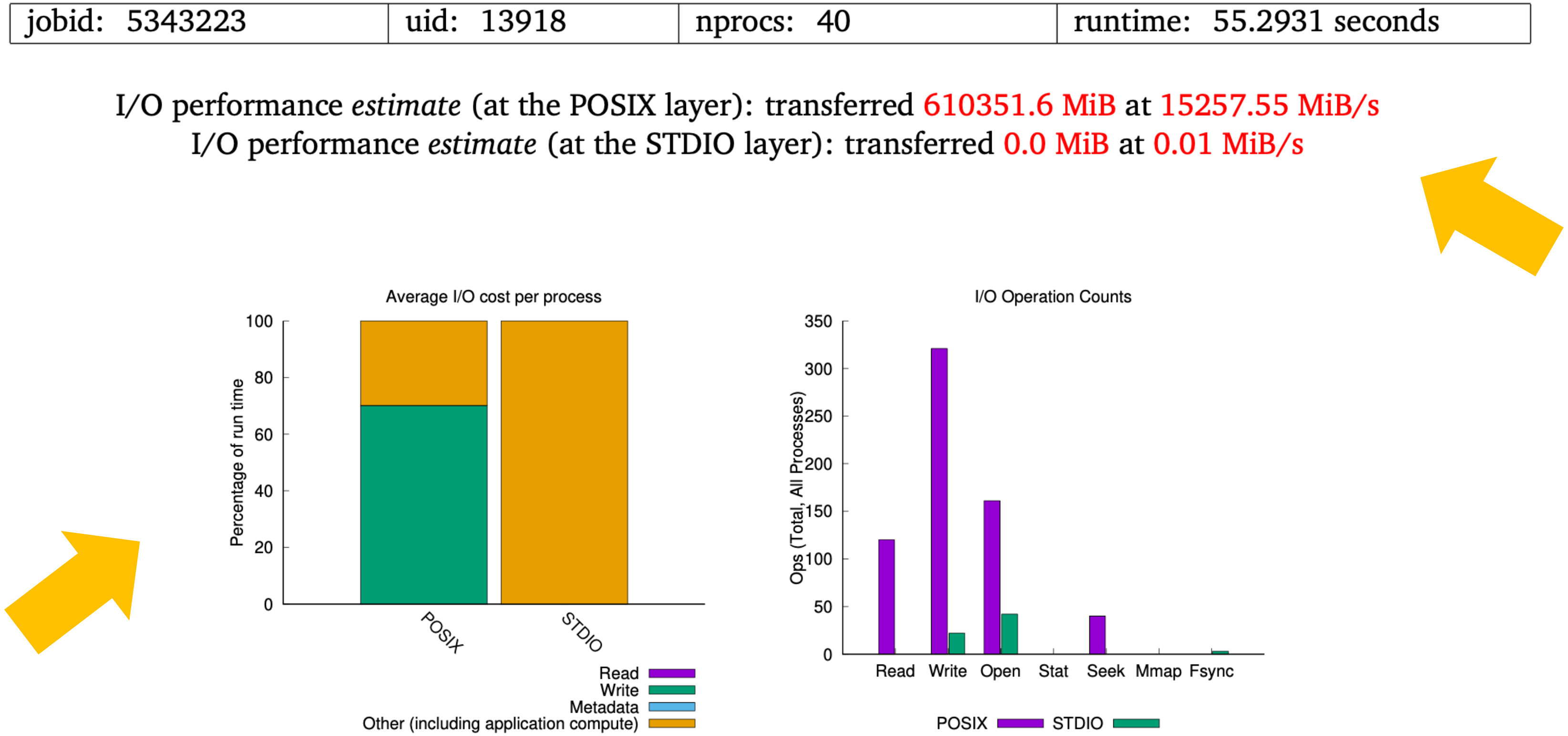
```
    darshan-job-summary.pl fp40.darshan
```

- Generate a summary pdf report. If your application writes to a lot file, you might want to proceed with caution.

```
    darshan-summary-per-file.sh fp40.darshan
reports_per_file_dir
```

Darshan Report

What job did we run ?



jobid: 5343223	uid: 13918	nprocs: 40	runtime: 55.2931 seconds

I/O performance *estimate* (at the POSIX layer): transferred 610351.6 MiB at 15257.55 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 0.01 MiB/s

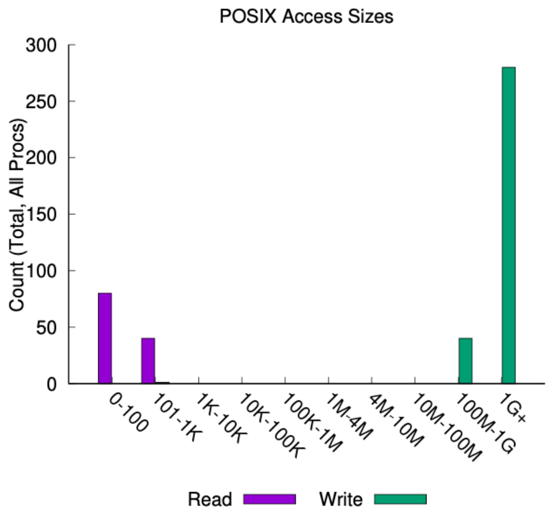
What is the overall bandwidth ?

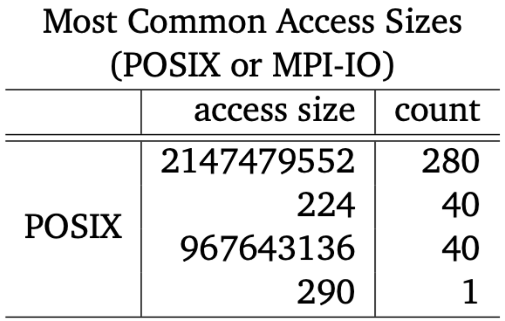
What is the fraction of runtime doing IO ?

How many IO operations ?

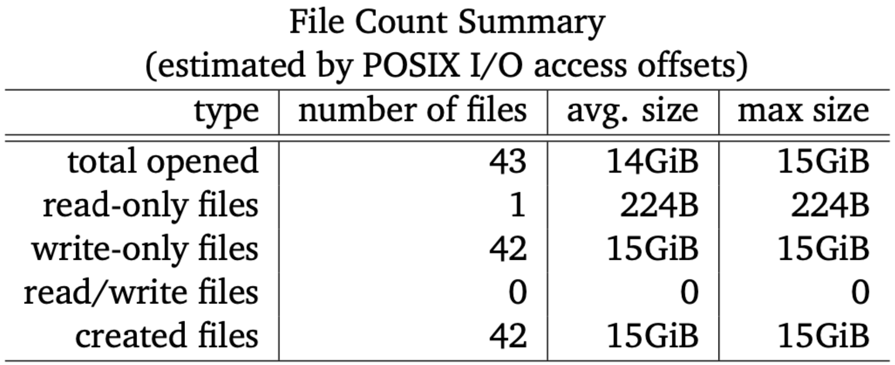
Darshan Report

- 4 nodes x 10 ranks per node = 40 processors
- 8 writes per processor
- Each write is about 2GiB write , except one
- Data written is about 15GiB per file




Most Common Access Sizes
(POSIX or MPI-IO)

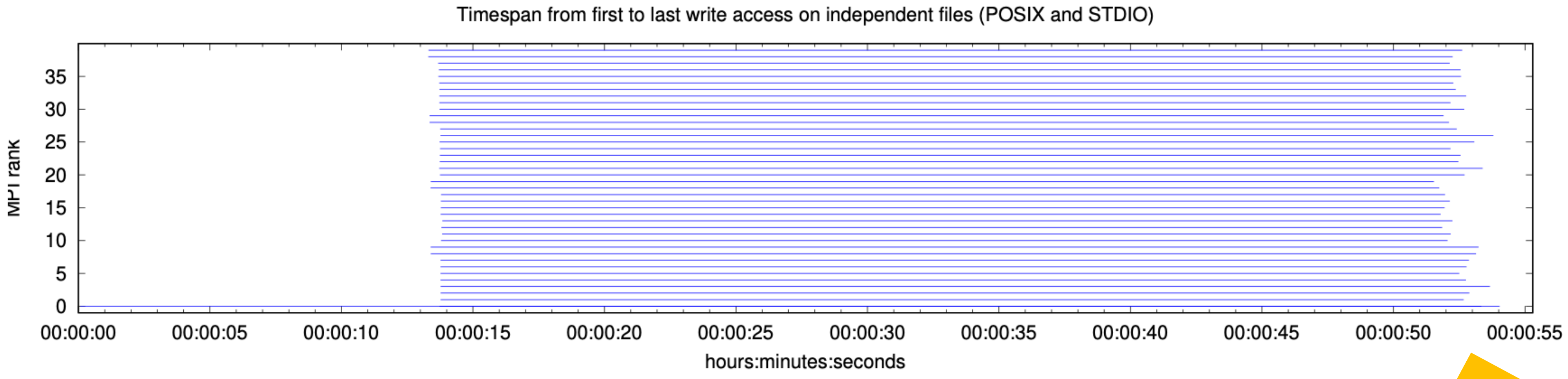
	access size	count
POSIX	2147479552	280
	224	40
	967643136	40
	290	1


File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	43	14GiB	15GiB
read-only files	1	224B	224B
write-only files	42	15GiB	15GiB
read/write files	0	0	0
created files	42	15GiB	15GiB

Darshan Report

Shows a timeline of all POSIX and STDIO output



Roughly the same time spent writing per process

Darshan Summary

- Generate the textual summary report

```
darshan-parser posix_file_per_process.darshan >
summary_posix_file_per_process.txt
```

- To collect statistics per rank, without aggregating aver all process , disable shared reduction before launching the job

```
export DARSHAN_DISABLE_SHARED_REDUCTION=1
```

Darshan Summary

```
#<module> <rank> <record id> <counter> <value> <file name> <mount pt> <fs type>

POSIX 0 10315675029492807030 POSIX_OPENS 1 /mnt/lustre/a2fs-
work4/work/z19/z19/lparisi/io_data/posix/data0.out /mnt/lustre/a2fs-work4 lustre

POSIX 0 10315675029492807030 POSIX_FILENOS 0 /mnt/lustre/a2fs-
work4/work/z19/z19/lparisi/io_data/posix/data0.out /mnt/lustre/a2fs-work4 lustre

POSIX 0 10315675029492807030 POSIX_DUPS 0 /mnt/lustre/a2fs-
work4/work/z19/z19/lparisi/io_data/posix/data0.out /mnt/lustre/a2fs-work4 lustre

POSIX 0 10315675029492807030 POSIX_READS 0 /mnt/lustre/a2fs-
work4/work/z19/z19/lparisi/io_data/posix/data0.out /mnt/lustre/a2fs-work4 lustre

POSIX 0 10315675029492807030 POSIX_WRITES 8 /mnt/lustre/a2fs-
work4/work/z19/z19/lparisi/io_data/posix/data0.out /mnt/lustre/a2fs-work4 lustre
```

Darshan Tracing

- Enable Tracing before launching the job

```
export DXT_ENABLE_IO_TRACE=1
```

- Generate the trace textual report

```
darshan-dxt-parser posix_file_per_process.darshan >
trace_posix_file_per_process.txt
```

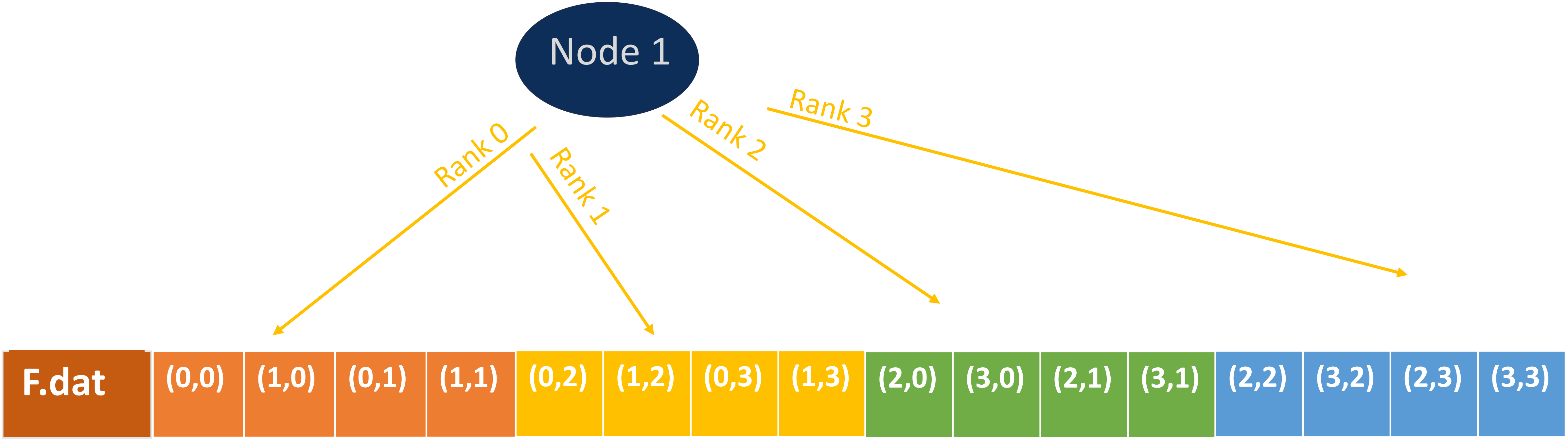
Darshan Tracing – File Per Process

- # DXT, file_id: 1514362557366186124, file_name: /mnt/lustre/a2fs-work4/work/z19/z19/lparisi/io_data/posix/data2.out
- # DXT, rank: 2, hostname: nid001848
- # DXT, write_count: 8, read_count: 0
- # DXT, mnt_pt: /mnt/lustre/a2fs-work4, fs_type: lustre
- # DXT, Lustre stripe_size: 1048576, Lustre stripe_count: 1
- # DXT, Lustre OST obdidx: 5
- # Module Rank Wt/Rd Segment Offset Length Start(s) End(s) [OST]
- X_POSIX 2 write 0 0 2147479552 13.7719 18.6430 [5]
- X_POSIX 2 write 1 2147479552 2147479552 18.6430 23.4592 [5]
- X_POSIX 2 write 2 4294959104 2147479552 23.4679 28.6368 [5]

Single shared file

Shared file

- Write in parallel to a single shared file
- Each process write its own data to a separate section of the file
- Works for POSIX on LUSTRE, but not STDIO



Node 1

Rank 0

Rank 1

Rank 2

Rank 3

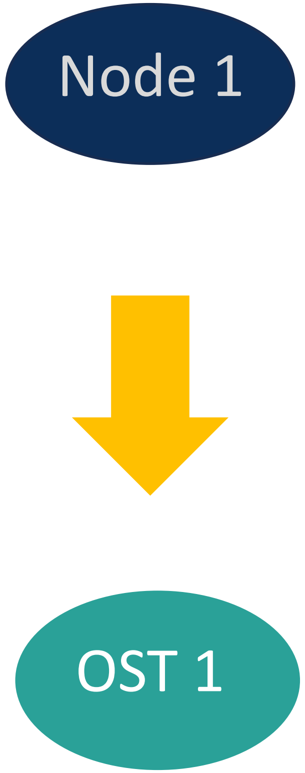
F.dat	(0,0)	(1,0)	(0,1)	(1,1)	(0,2)	(1,2)	(0,3)	(1,3)	(2,0)	(3,0)	(2,1)	(3,1)	(2,2)	(3,2)	(2,3)	(3,3)

Write Bandwidth

Using 4 nodes, 10 ranks per node

File	Striping	Performance
File Per Process		16 GB/s
Shared File POSIX		3 GB/s

Lustre

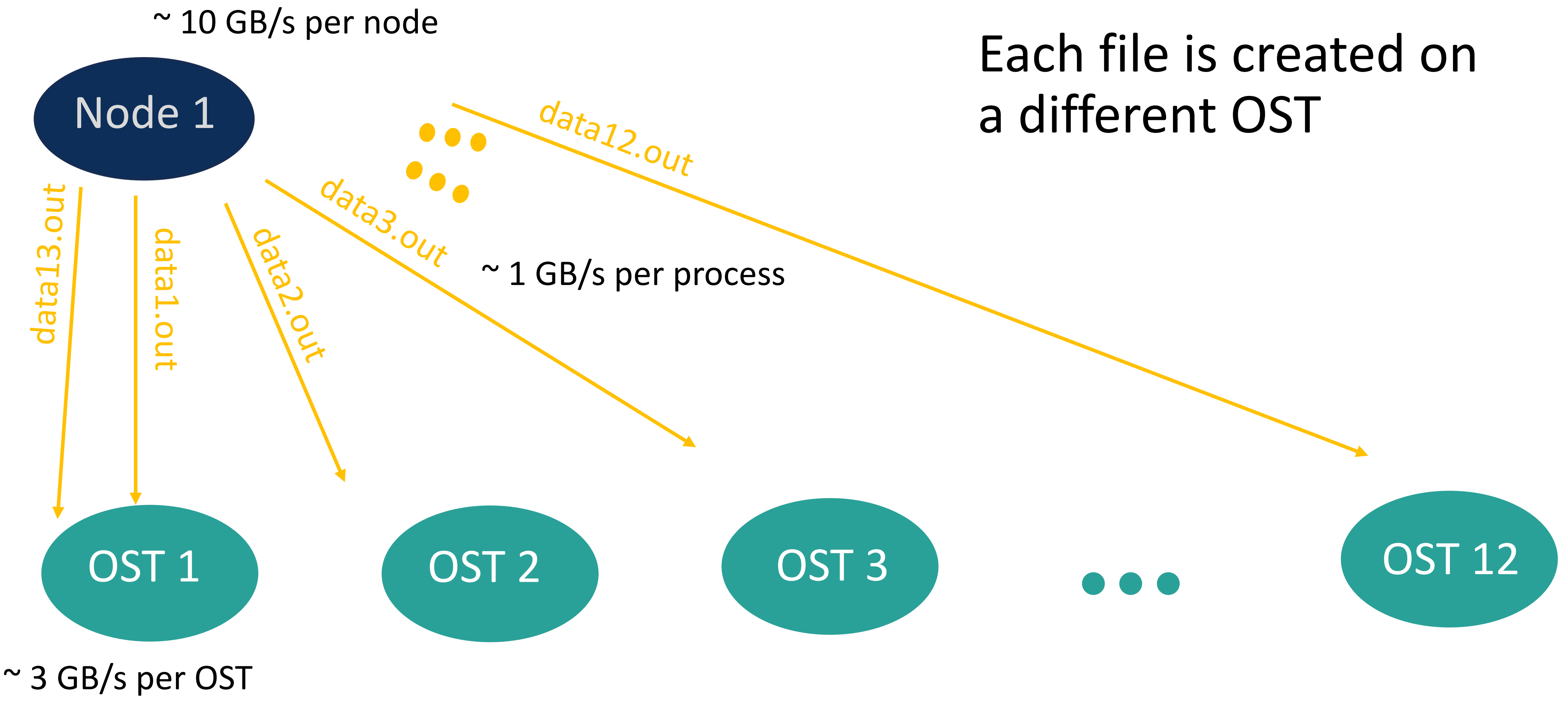


Job runs on a compute node.

The data is saved on a different device,

The Object Storage Target (OST)

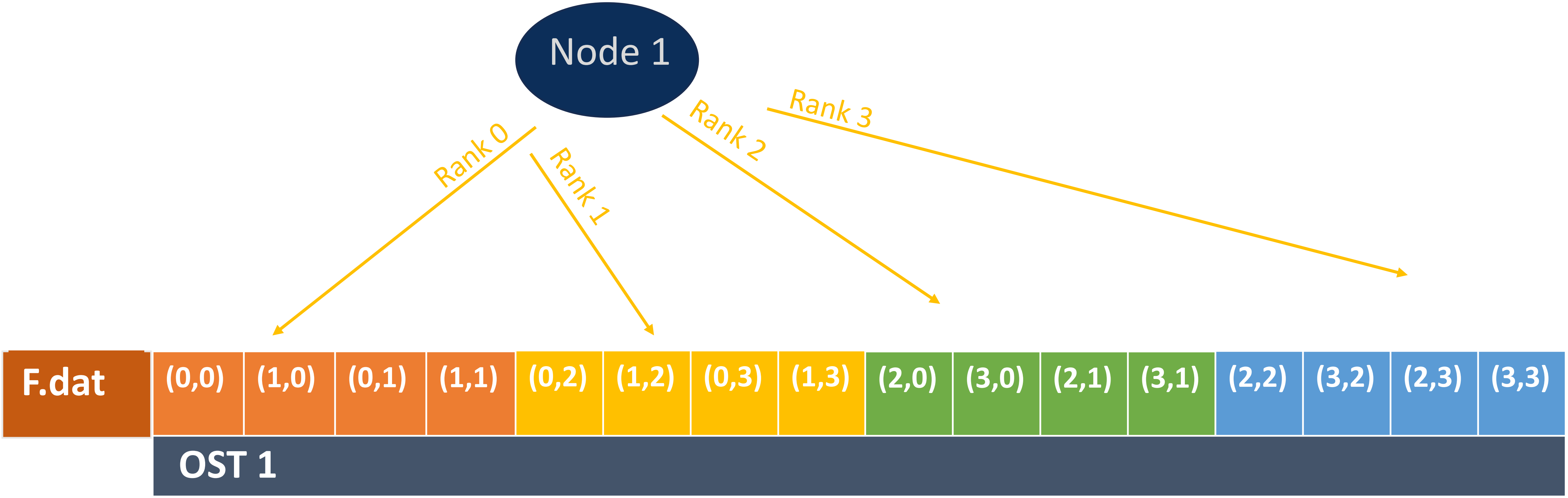
Lustre



Each file is created on a different OST

Shared file

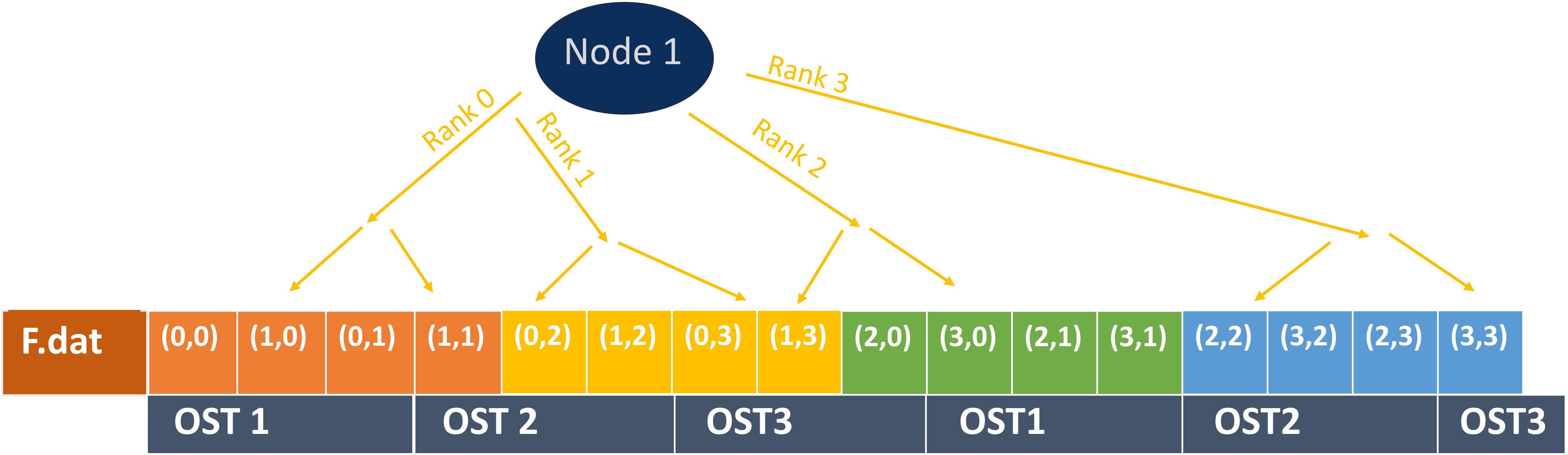
- By default, the whole file is on a single OST



F.dat	(0,0)	(1,0)	(0,1)	(1,1)	(0,2)	(1,2)	(0,3)	(1,3)	(2,0)	(3,0)	(2,1)	(3,1)	(2,2)	(3,2)	(2,3)	(3,3)
	OST 1															

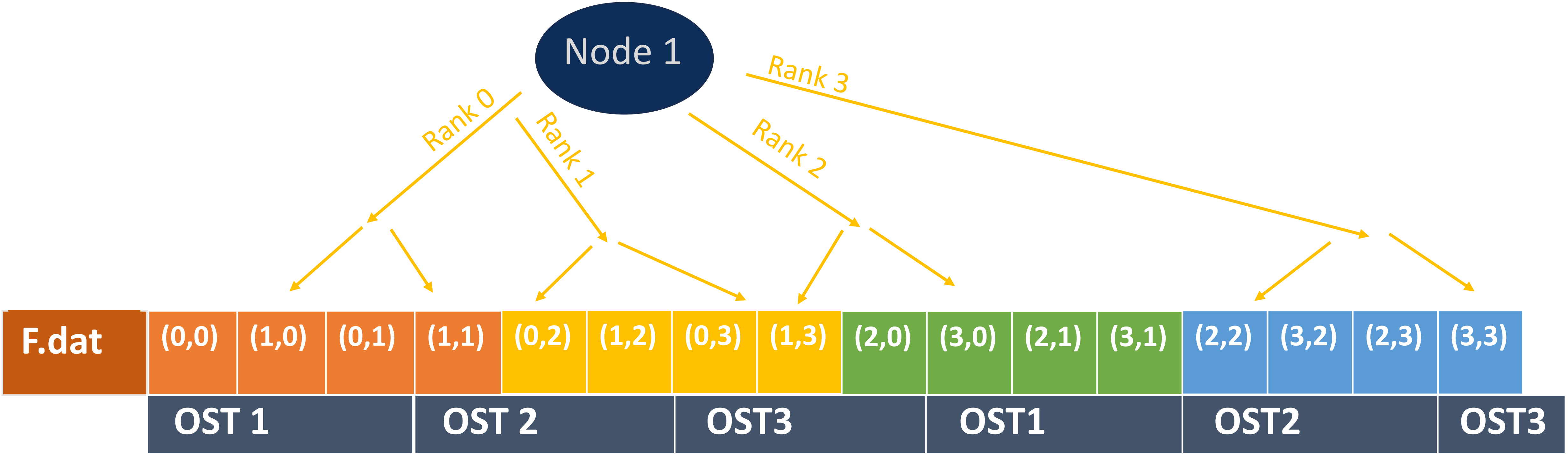
Striping

- File is divided in chunks called stripes.
- Stripes are assigned to OSTs in a round robin fashion



Striping

- Number of Stripes: number of OSTs to wrap over (3 in the example)
- Stripe size: equal for all stripes, 1MiB by default



Striping

- Needs to be set at file or directory creation
- The number of stripes can be set on newly created directory

```
lfs setstripe -c ${NUMBER_OF_STRIPES} write_dir
```

- To set the number of stripes equal to the number of OSTS, set

```
${NUMBER_OF_STRIPES}=-1
```

Darshan Tracing

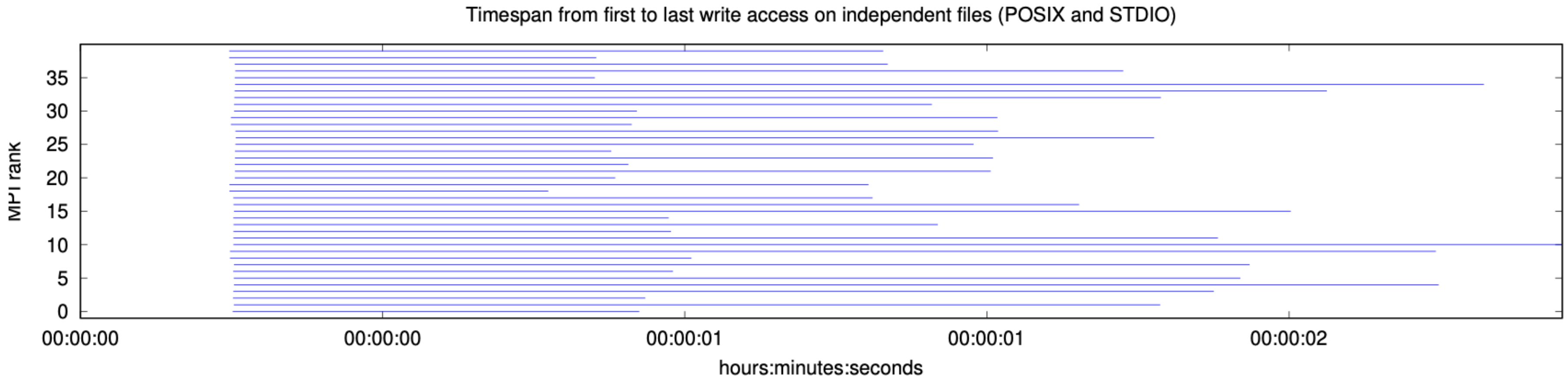
- # DXT, file_id: 2595840042677522317, file_name: /mnt/lustre/a2fs-work4/work/z19/z19/lparisi/io_data/posix/posix_shared/striped/data.out
- # DXT, rank: 27, hostname: nid001893
- # DXT, write_count: 8, read_count: 0
- # DXT, mnt_pt: /mnt/lustre/a2fs-work4, fs_type: lustre
- # DXT, Lustre stripe_size: 1048576, Lustre stripe_count: 12
- # DXT, Lustre OST obdidx: 8 9 10 11 0 1 2 3 4 5 6 7
- # Module Rank Wt/Rd Segment Offset Length Start(s) End(s) [OST]
- X_POSIX 27 write 0 432000000000 2147479552 14.5021 25.9009 [11] [0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
- X_POSIX 27 write 1 434147479552 2147479552 25.9009 38.4769 [7] [8] [9] [10] [11] [0] [1] [2] [3] [4] [5] [6]
- X_POSIX 27 write 2 436294959104 2147479552 38.4808 52.6008 [3] [4] [5] [6] [7] [8] [9] [10] [11] [0] [1] [2]

Write Bandwidth

Using 4 nodes, 10 ranks per node

File	Striping	Performance
File Per Process	unstriped	16 GB/s
Shared File POSIX	unstriped	3 GB/s
Shared File POSIX	striped	4 GB/s

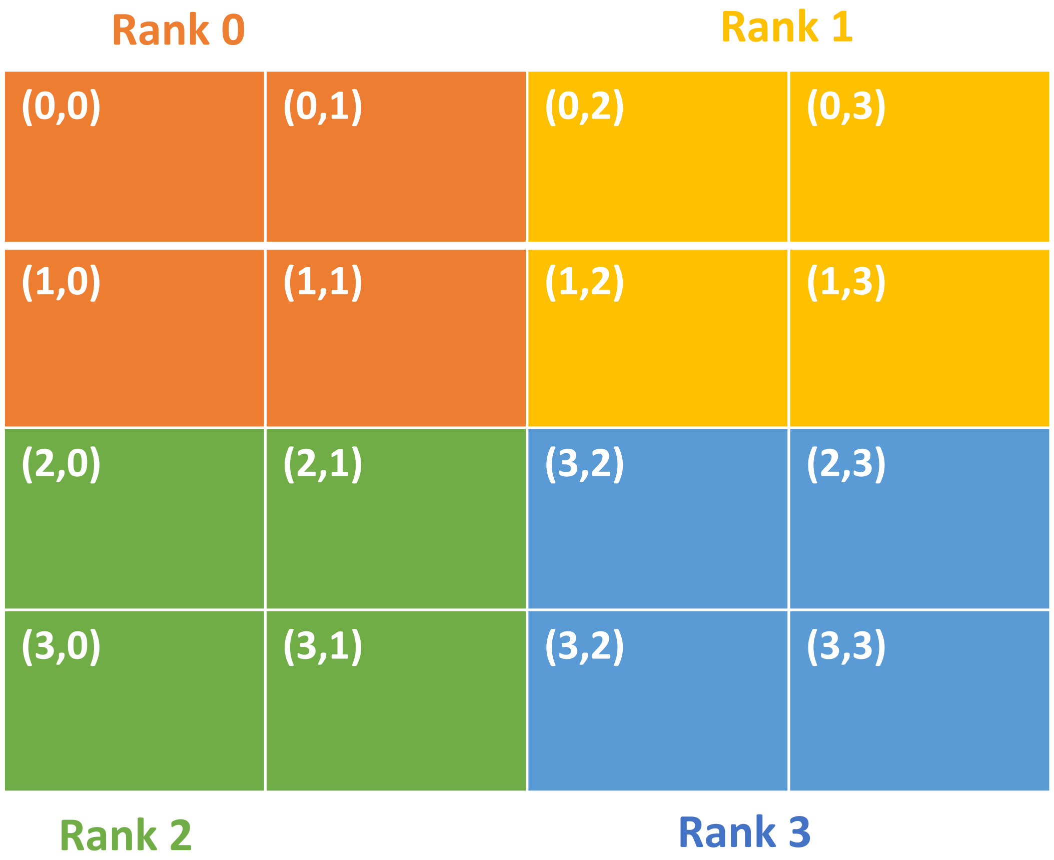
Cache coherence & Locking



- Cache coherence: once a POSIX call completes, any other call must see the result of the previous operation
- Locking : To guarantee cache coherence, lustre locks sections of the files.

  Neighbouring processes have to wait for other processes to finish

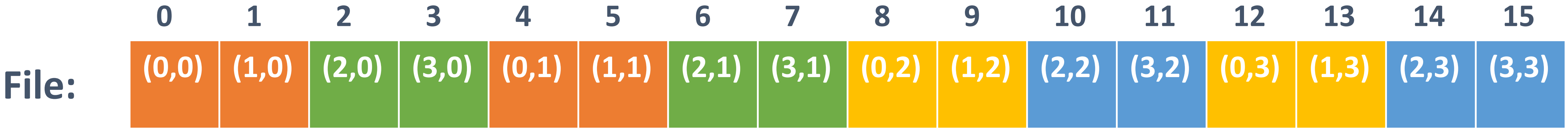
2D Array



(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(3,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

2D array is distributed across several process

Each rank needs to write to many different sections on the file



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
File:	(0,0)	(1,0)	(2,0)	(3,0)	(0,1)	(1,1)	(2,1)	(3,1)	(0,2)	(1,2)	(2,2)	(3,2)	(0,3)	(1,3)	(2,3)	(3,3)

MPIIO

MPI-IO

- Write distributed data to a single shared file

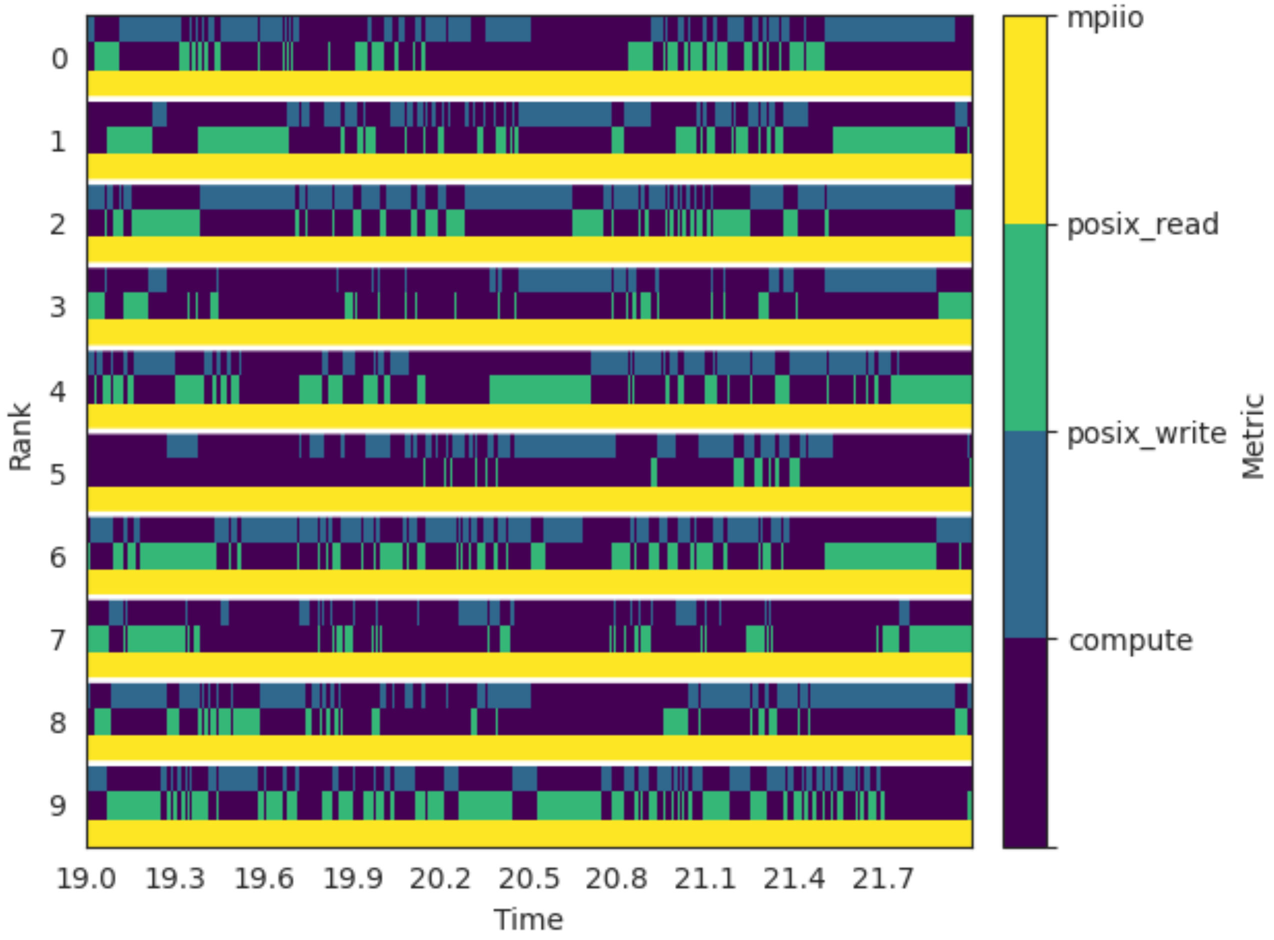
- Handles writing to non contiguous sections of the file
- Supports independent and collective operations

Write Bandwidth

Using 4 nodes, 10 ranks per node

File	Striping	Performance
File Per Process	unstriped	16 GB/s
Shared File POSIX	unstriped	3 GB/s
Shared File POSIX	striped	4 GB/s
Shared File MPIIO - independent	striped	0.04GB/s

MPIIO - Independent



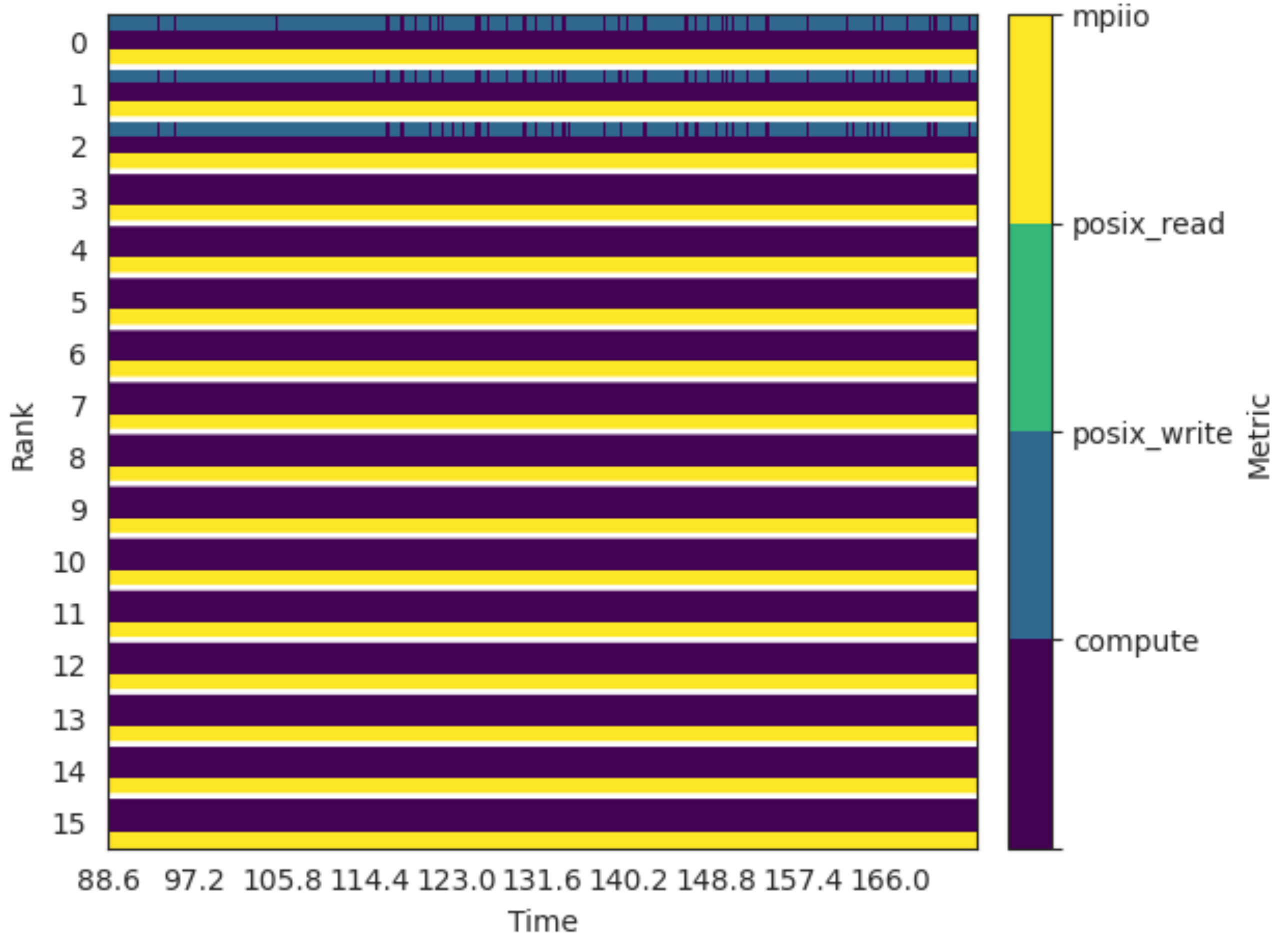
- Only write calls in MPIIO
- MPIIO issues POSIX calls in the background
- Both read and write POSIX calls are issued
- Due to an optimization called `data sieving`

Write Bandwidth

Using 4 nodes, 10 ranks per node

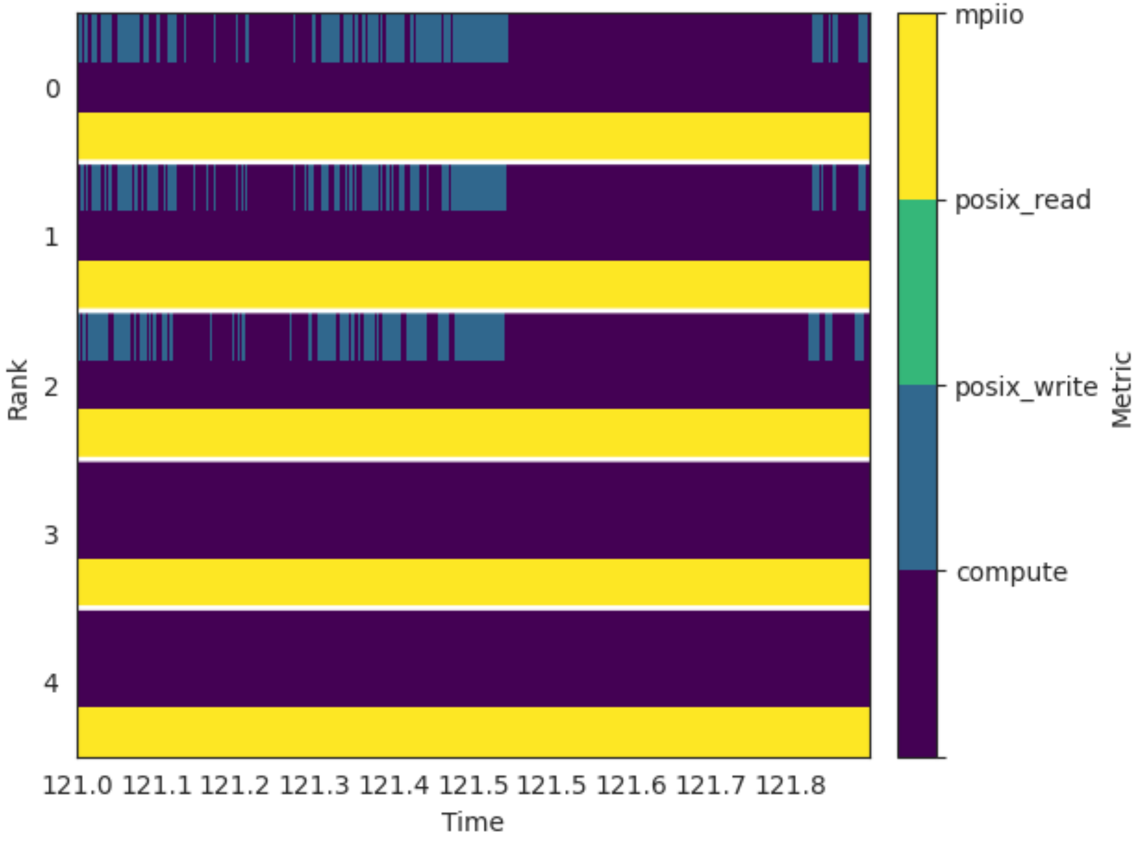
File	Striping	Performance
File Per Process	unstriped	16 GB/s
Shared File POSIX	unstriped	3 GB/s
Shared File POSIX	striped	4 GB/s
Shared File MPIIO - independent	striped	0.04GB/s
Shared File MPIIO - collective	striped	2 GB/s

MPIIO - Collective



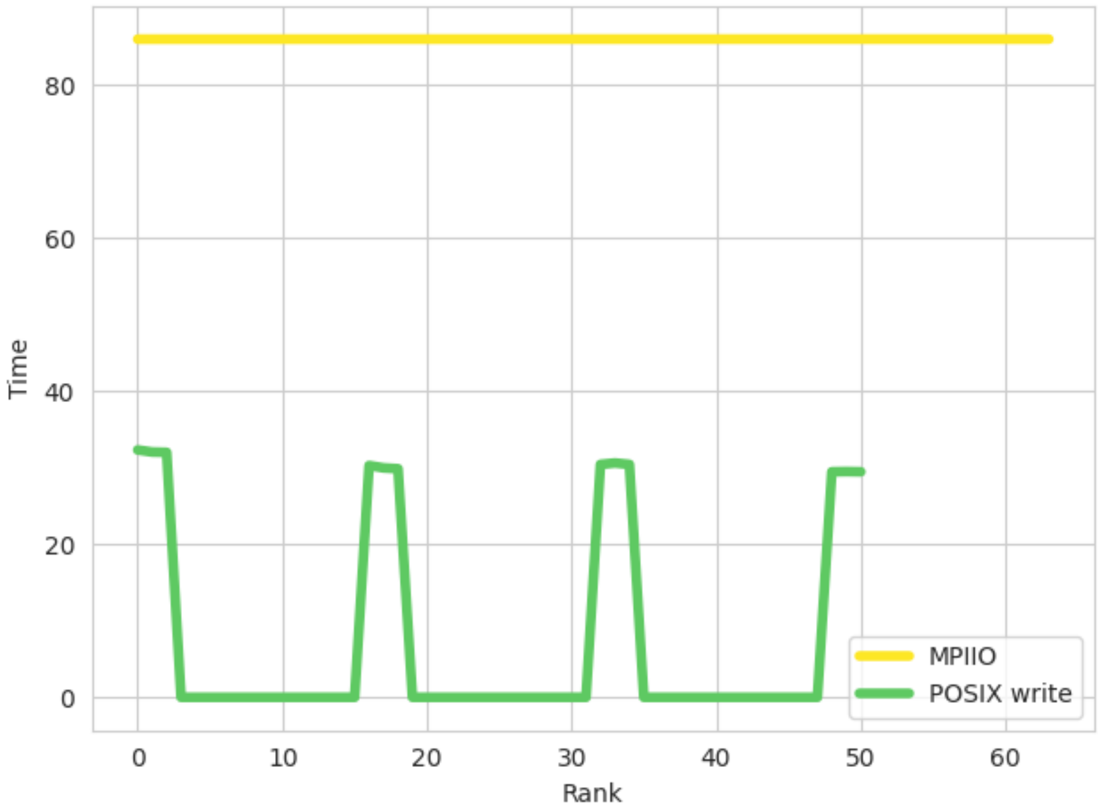
- No data sieving
- Only 3 processes per node (aggregators) are executing POSIX writes

MPIIO - Collective



- Many small successive writes, interleaved with larger areas with no I/O
- As data is spread on all ranks, but only a few are writing to disk there must be communication

MPIIO - Collective



- About 30% of time is spent writing to disk
- The remaining 70% is spent in the MPI library
- Likely large overhead from communication

Each call writes 1MiB of data
= stripe size

```
# Module Rank Wt/Rd Segment Offset Length Start(s) End(s) [OST]
X_POSIX 0 write 0 0 1048576 22.5985 22.6008 [ 7]
X_POSIX 0 write 1 12582912 1048576 22.6177 22.6189 [ 7]
X_POSIX 0 write 2 25165824 1048576 22.6204 22.6216 [ 7]
X_POSIX 0 write 3 37748736 1048576 22.6229 22.6241 [ 7]
X_POSIX 0 write 4 50331648 1048576 22.6258 22.6269 [ 7]
```

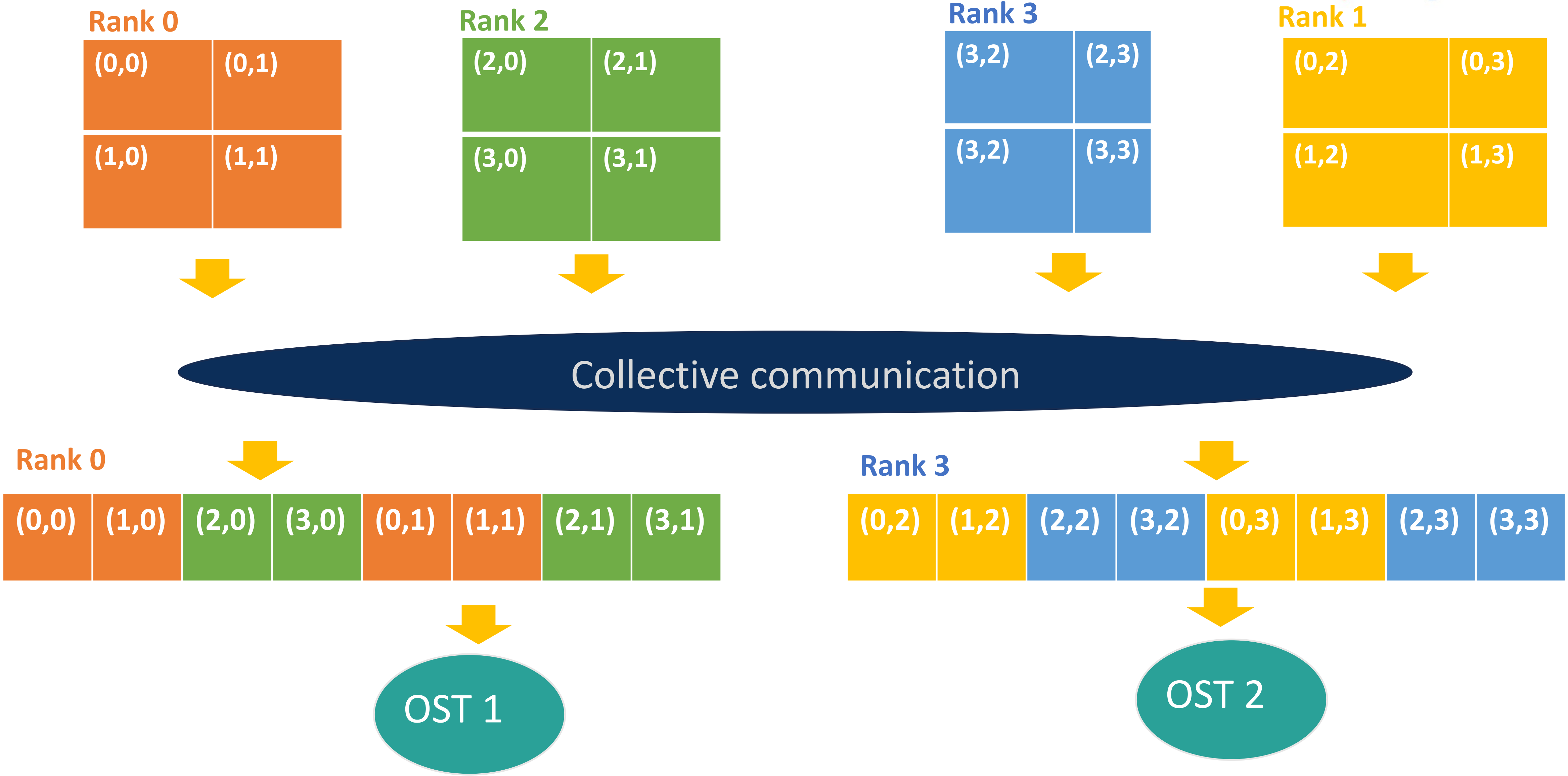
Each aggregator writes to a different OST

```
# Module Rank Wt/Rd Segment Offset Length Start(s) End(s) [OST]
• X_POSIX 256 write 0 2097152 1048576 22.5972 22.5995 [ 9]
• X_POSIX 256 write 1 14680064 1048576 22.6008 22.6019 [ 9]
• X_POSIX 256 write 2 27262976 1048576 22.6162 22.6172 [ 9]
```

By default, one aggregator per stripe . The number of aggregators can be changed using environment variables

For 12 stripes , that means 3 aggregators per node

MPIIO –Two phase



Rank 0

(0,0) (0,1)

(1,0) (1,1)

Rank 2

(2,0) (2,1)

(3,0) (3,1)

Rank 3

(3,2) (2,3)

(3,2) (3,3)

Rank 1

(0,2) (0,3)

(1,2) (1,3)

Collective communication

Rank 0

(0,0) (1,0) (2,0) (3,0) (0,1) (1,1) (2,1) (3,1)

Rank 3

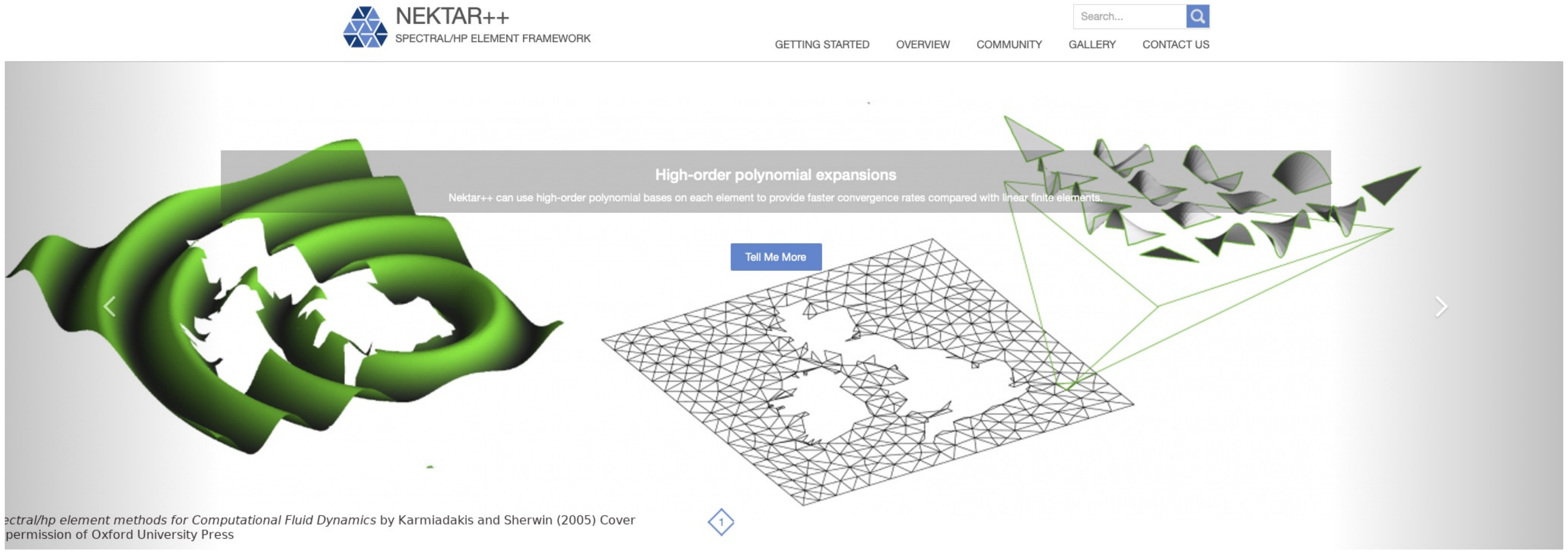
(0,2) (1,2) (2,2) (3,2) (0,3) (1,3) (2,3) (3,3)

OST 1

OST 2

Nektar++

Nektar++



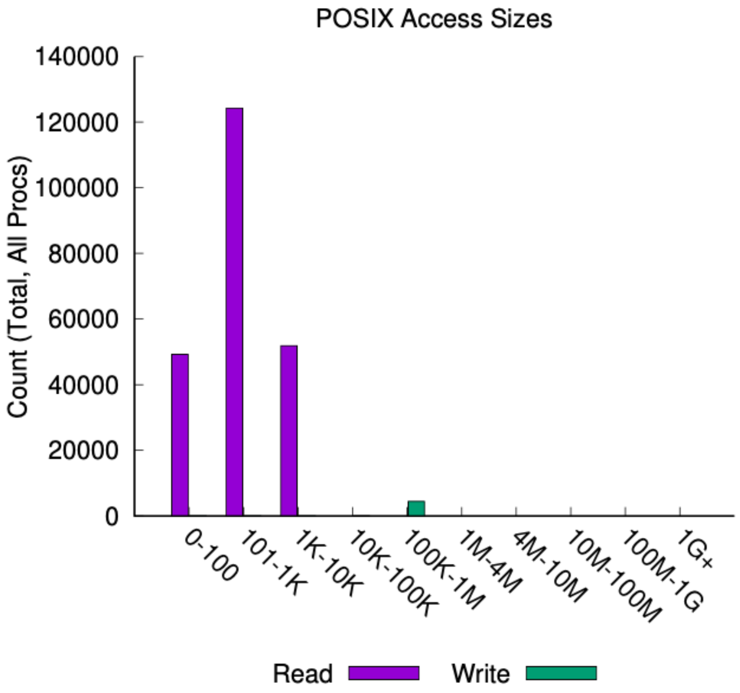
High-order polynomial expansions

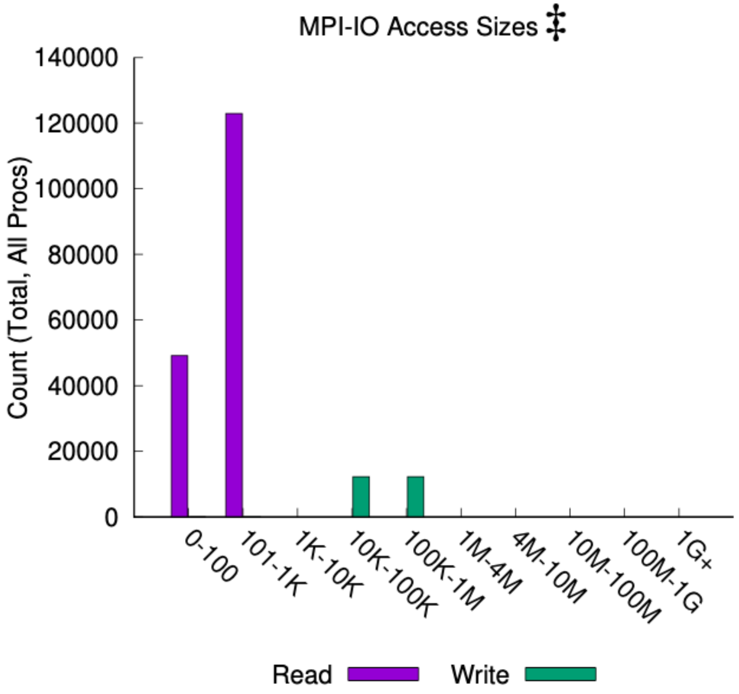
Nektar++ can use high-order polynomial bases on each element to provide faster convergence rates compared with linear finite elements.

ectral/hp element methods for Computational Fluid Dynamics by Karmiadakis and Sherwin (2005) Cover
permission of Oxford University Press

- Unstructured mesh
- Time evolution of a diffusion equation
- *Very short* simulation on 8 nodes (128 tasks per core) , unstriped

Nektar++

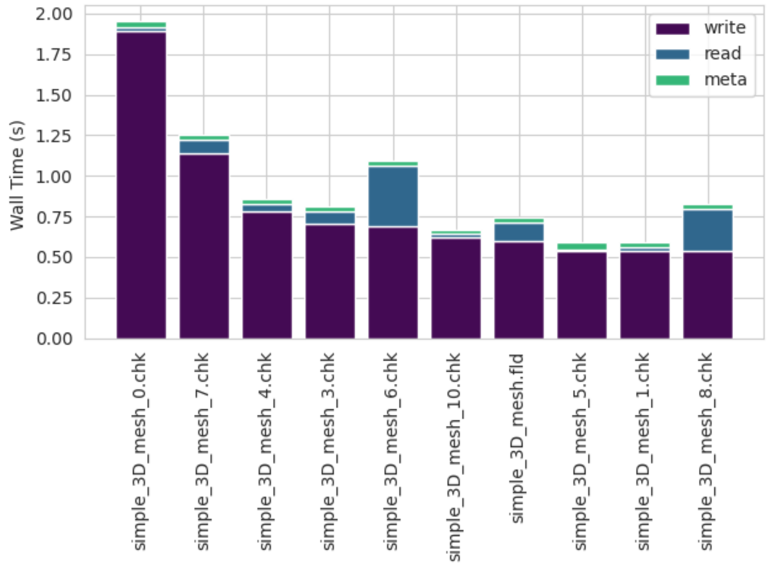




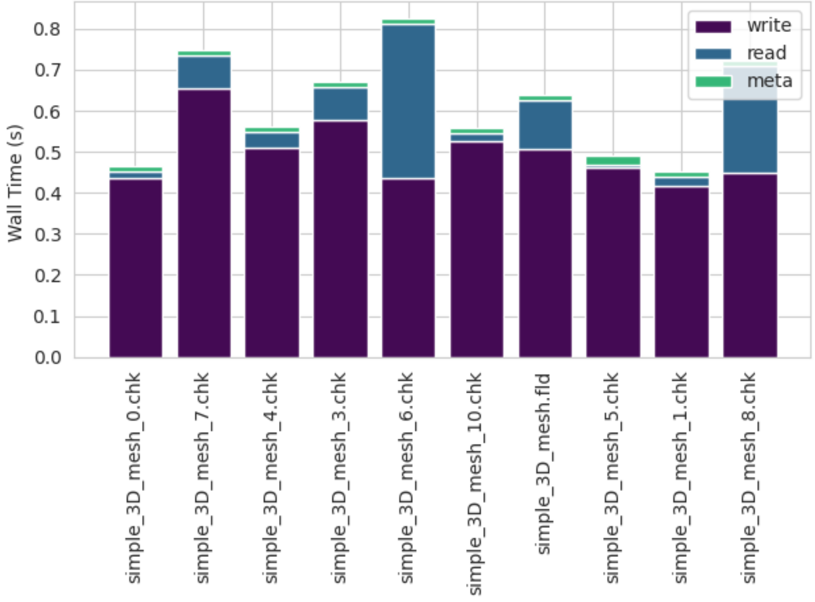
- Lots of small reads
- Fewer big and medium write accesses

Nektar++

MPIIO



POSIX



- Top 10 time consuming files
- Dominated by write operations, but significant contribution from read operations

Nektar++

```
MPI-IO 0 11539591632334063225 MPIIO_INDEP_READS 14 simple_3D_mesh_7.chk
MPI-IO 0 11539591632334063225 MPIIO_INDEP_WRITES 1 simple_3D_mesh_7.chk
MPI-IO 0 11539591632334063225 MPIIO_COLL_READS 0 simple_3D_mesh_7.chk
MPI-IO 0 11539591632334063225 MPIIO_COLL_WRITES 2 simple_3D_mesh_7.chk
```

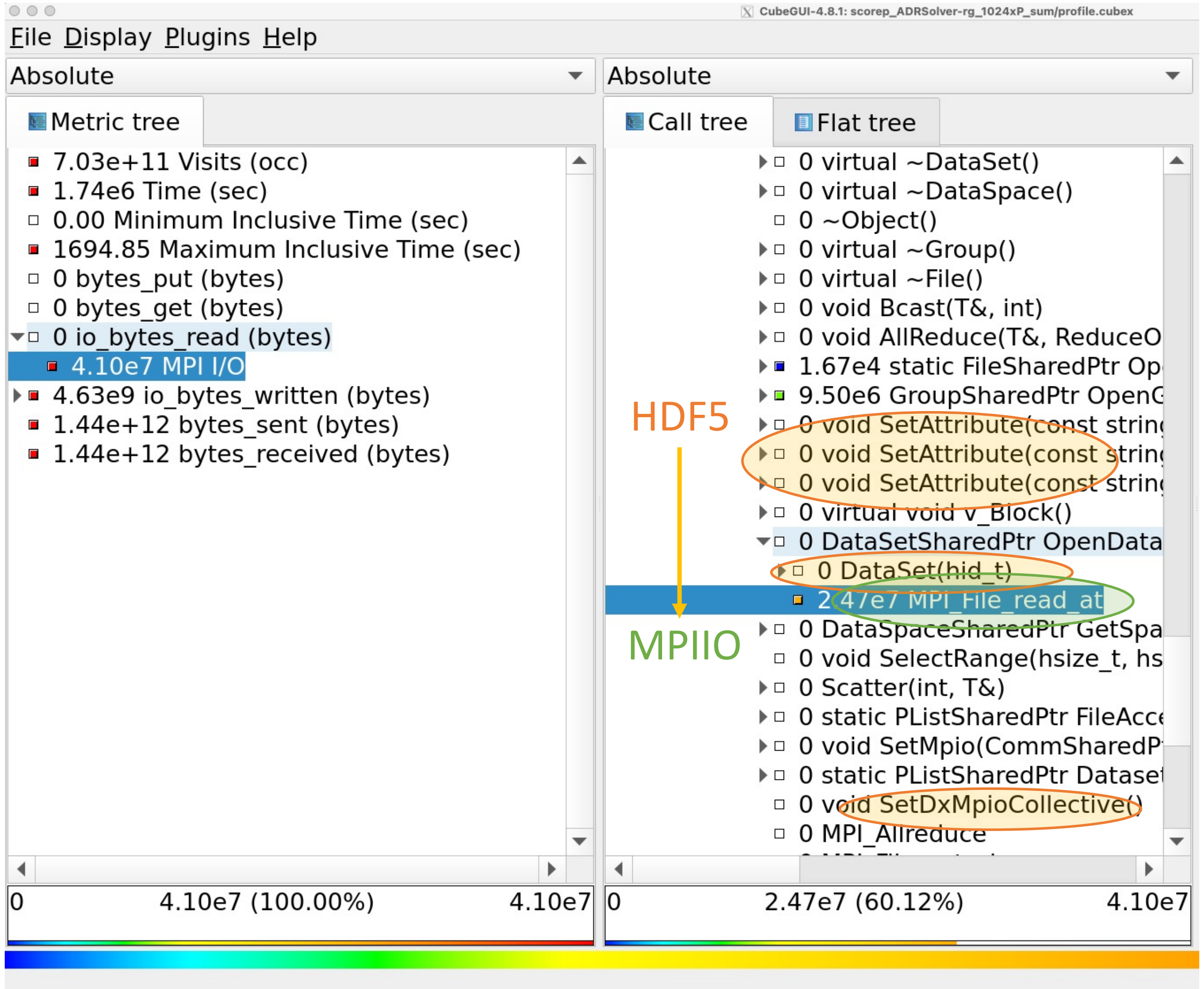
```
# Module Rank Wt/Rd Segment Offset Length Start(s) End(s)
X_MPIIO 0 write 0 73008 18088 451.1505 452.4895
X_MPIIO 0 write 1 18455856 361760 452.5259 453.0412
X_MPIIO 0 write 2 0 96 453.0505 453.0568
X_MPIIO 0 read 0 0 8 450.9990 450.9999
X_MPIIO 0 read 1 0 9 450.9999 450.9999
X_MPIIO 0 read 2 9 87 450.9999 450.9999
X_MPIIO 0 read 3 96 512 450.9999 450.9999
```

Two big collective writes

One small independent write

Several small and very quick reads

Nektar++ : SCALASCA



CubeGUI-4.8.1: scorep_ADRSolver-rg_1024xP_sum/profile.cubex

```
7.03e+11 Visits (occ)
1.74e6 Time (sec)
0.00 Minimum Inclusive Time (sec)
1694.85 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
0 io_bytes_read (bytes)
  4.10e7 MPI I/O
4.63e9 io_bytes_written (bytes)
1.44e+12 bytes_sent (bytes)
1.44e+12 bytes_received (bytes)

0          4.10e7 (100.00%)          4.10e7
```

```
0 virtual ~DataSet()
0 virtual ~DataSpace()
0 ~Object()
0 virtual ~Group()
0 virtual ~File()
0 void Bcast(T&, int)
0 void AllReduce(T&, ReduceO
1.67e4 static FileSharedPtr Op
9.50e6 GroupSharedPtr OpenG
0 void SetAttribute(const string
0 void SetAttribute(const string
0 void SetAttribute(const string
0 virtual void v_Block()
0 DataSetSharedPtr OpenData
  0 DataSet(hid_t)
  2.47e7 MPI_File_read_at
0 DataSpaceSharedPtr GetSpa
0 void SelectRange(hsize_t, hs
0 Scatter(int, T&)
0 static PListSharedPtr FileAcce
0 void SetMpio(CommSharedP
0 static PListSharedPtr Dataset
0 void SetDxMpioCollective()
0 MPI_Allreduce

0          2.47e7 (60.12%)          4.10e7
```

- Can use a regular profiler for function calls, such as Scalasca
- Small reads are issued by HDF5 metadata operations
- SCALASCA reports size of data written/read by subroutines

Summary

- Can use Darshan to profile multiple layers of the IO software stack (filesystem, MPIIO, POSIX, etc..)
- Can combine with general profilers such as SCALASCA
- Guides setting up your environment (MPIIO hints such as the number of aggregators, striping etc..) regardless of which high level library you use
- Guides development of applications

Reusing this material