Parallel I/O doesn't have to be so hard: The ADIOS Library

Jonathan Dursi, SciNet
ljdursi@scinet.utoronto.ca

SciNet

compute • calcul CANADA

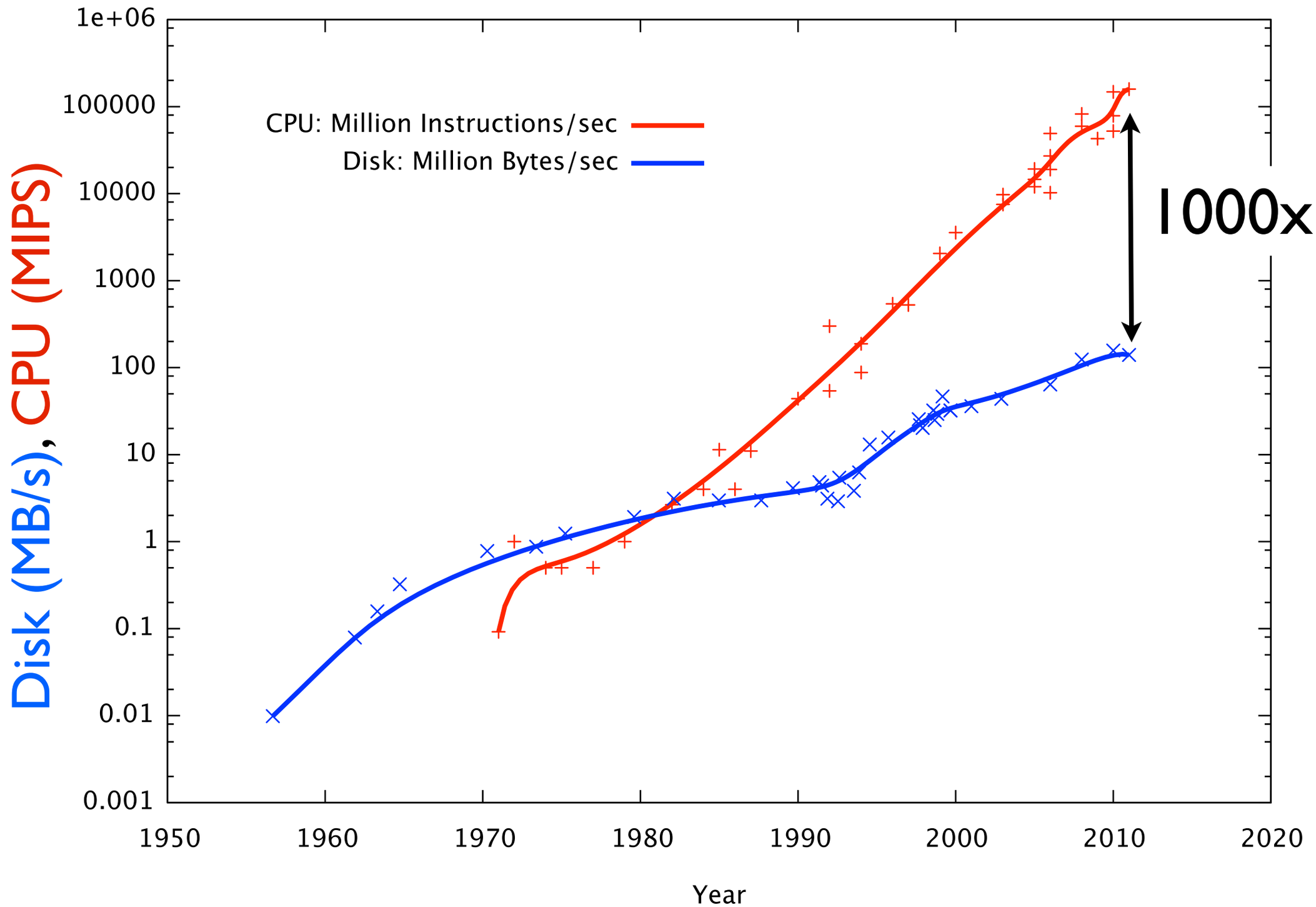
Agenda

- Intro to I/O
- MPI-IO
- HDF5, NetCDF4
- Parallel HDF5/NetCDF4
- ADIOS

SciNet

compute • calcul CANADA

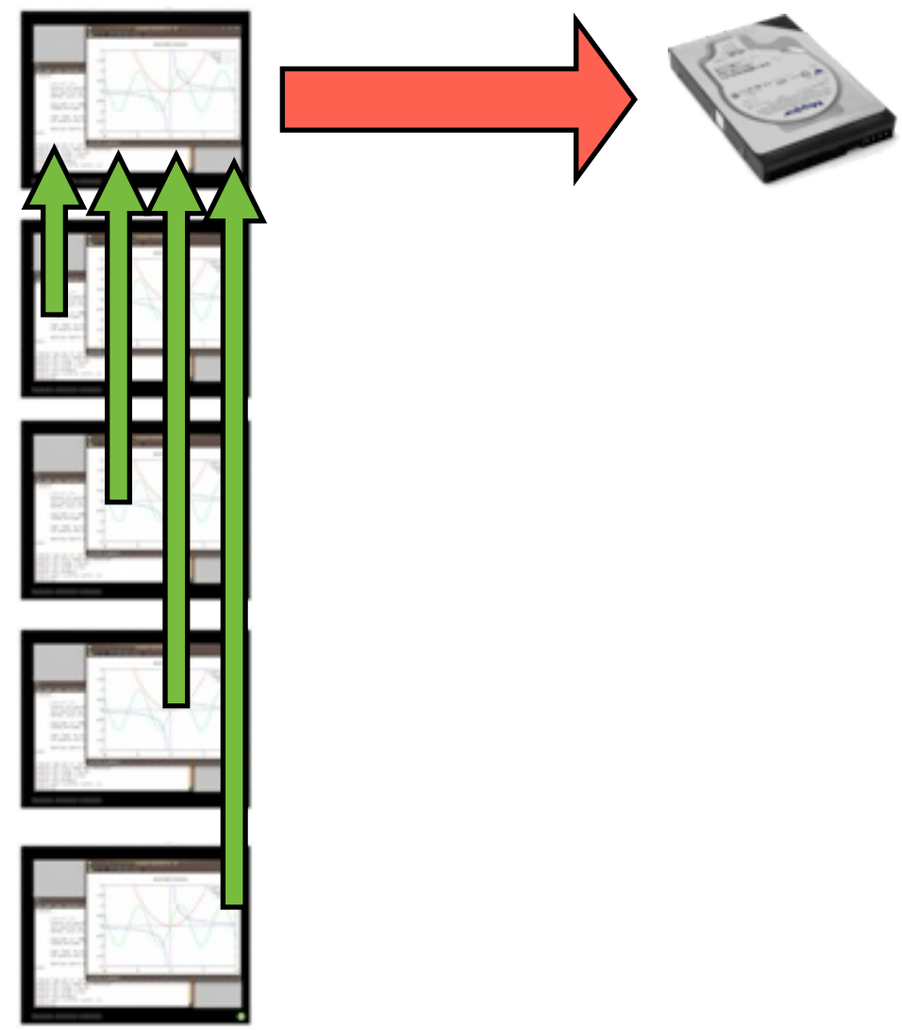
Disks are slower than CPUs (and getting slower)



Disk data from "The future of magnetic storage technology", Thompson & Best, and Tom's Hardware;

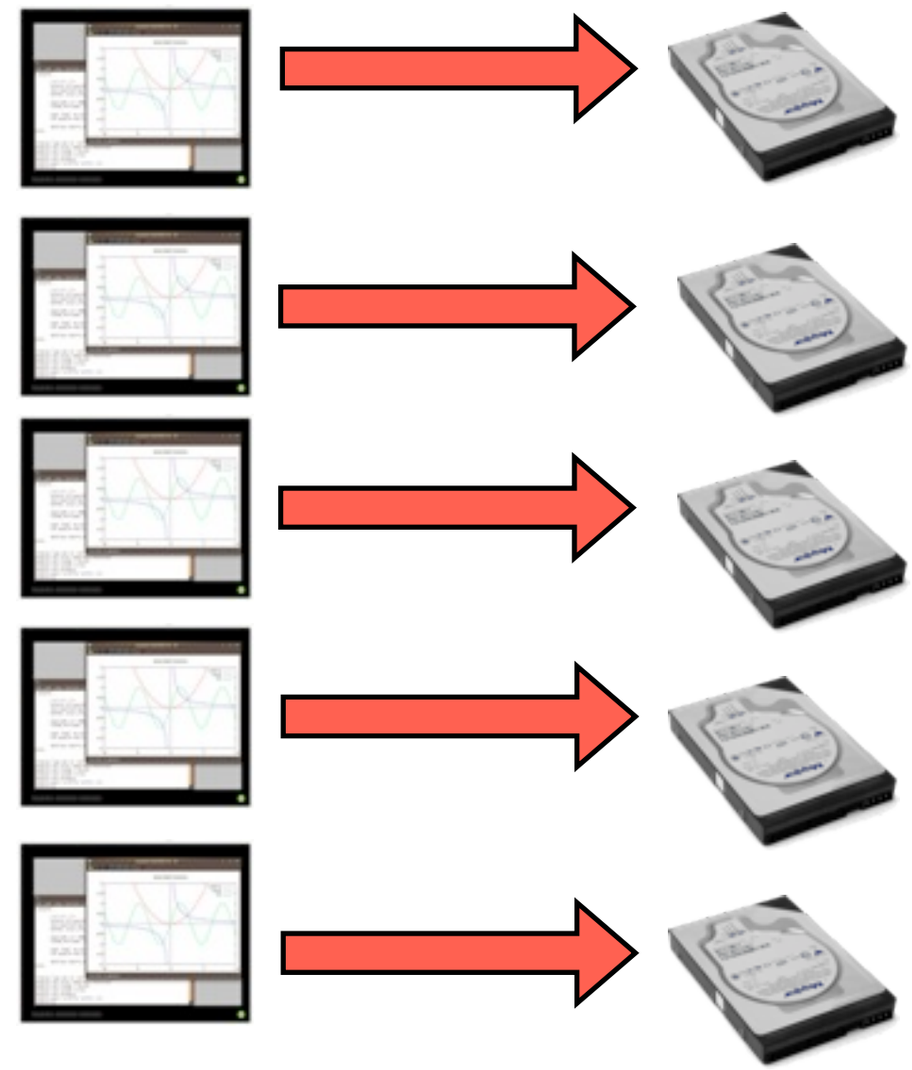
MIPS data from Wikipedia, "Instructions per Second".

Planning your I/O



- Parallel computation, several options.
- Everyone sends data to process 0
- Process 0 writes.
- Serialize I/O - huge bottleneck.

Planning your I/O



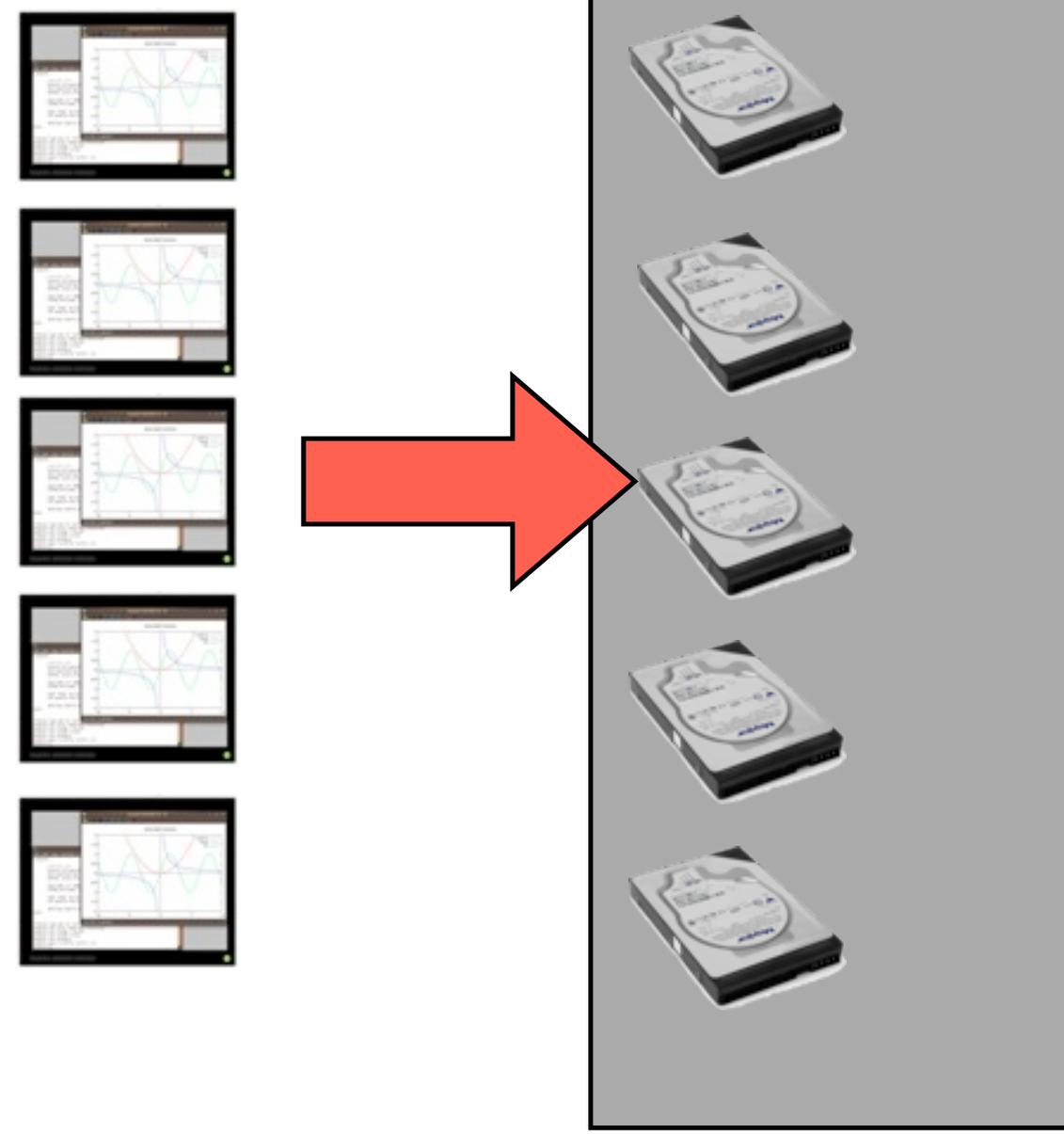
- Parallel computation, several options.
- Each process writes a file, possibly to local disk.
- Postpones the problem - how do you analyze, or restart with different # of procs?

SciNet

compute • calcul CANADA

Planning your I/O

- Parallel computation, several options.
- We're going to learn to avoid doing this by using Parallel I/O
- **Coordinated** single output of multiple processes.



Parallel I/O and large file systems

- Large disk systems featuring many servers, disks
- Can serve files to many clients concurrently
- Parallel File Systems -
- Lustre, Panasas, GlusterFS, Ceph, GPFS...

SciNet ~2k drives

SciNet

compute • calcul CANADA

SciNet's File Systems

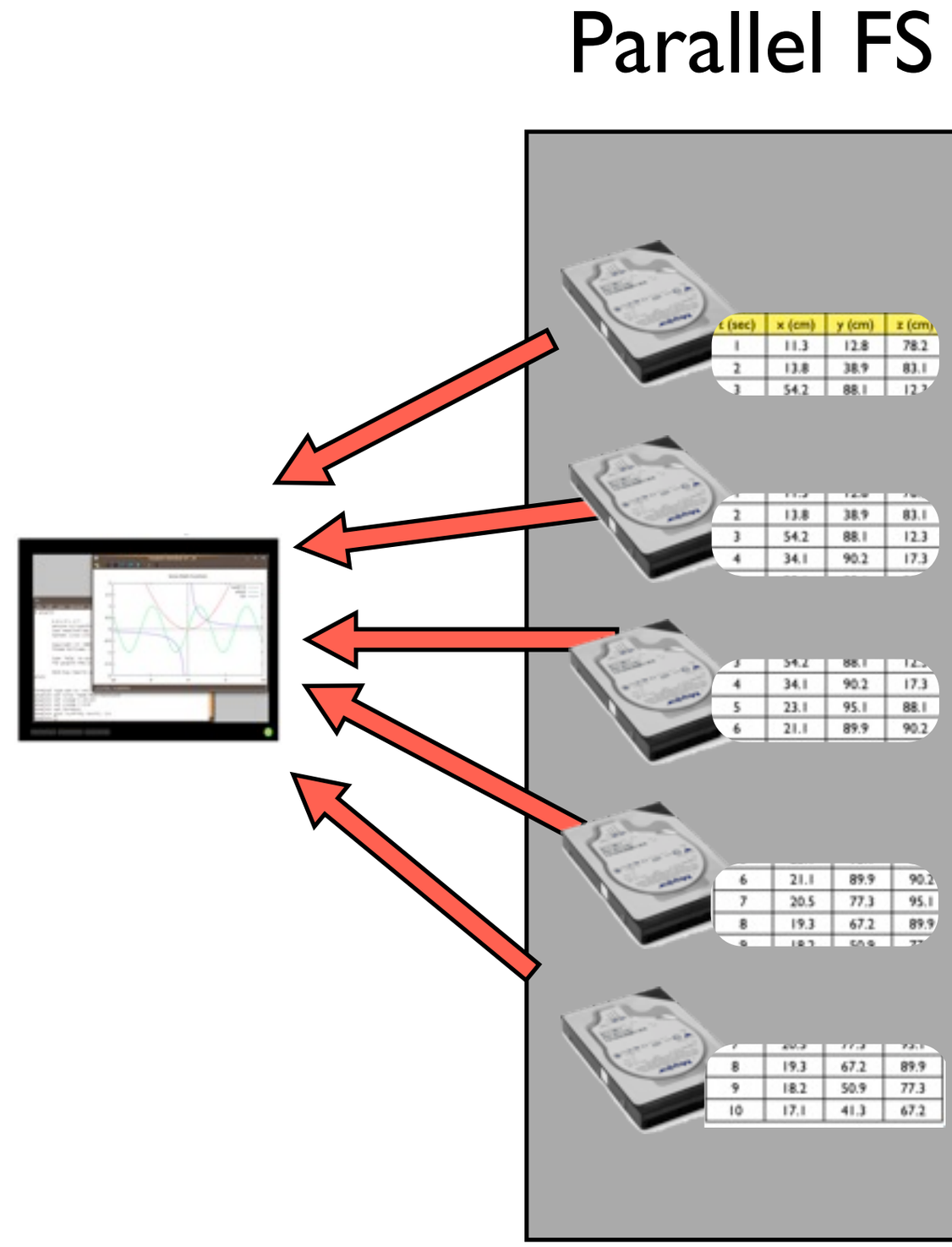
- 2x DCS9900 couplets
- 1,790 1TB SATA disk drives
- 1.4 PB of storage
- Single GPFS domain, accessed by all machines (TCS and GPC).
- Data to compute nodes via IB

SciNet

compute • calcul CANADA

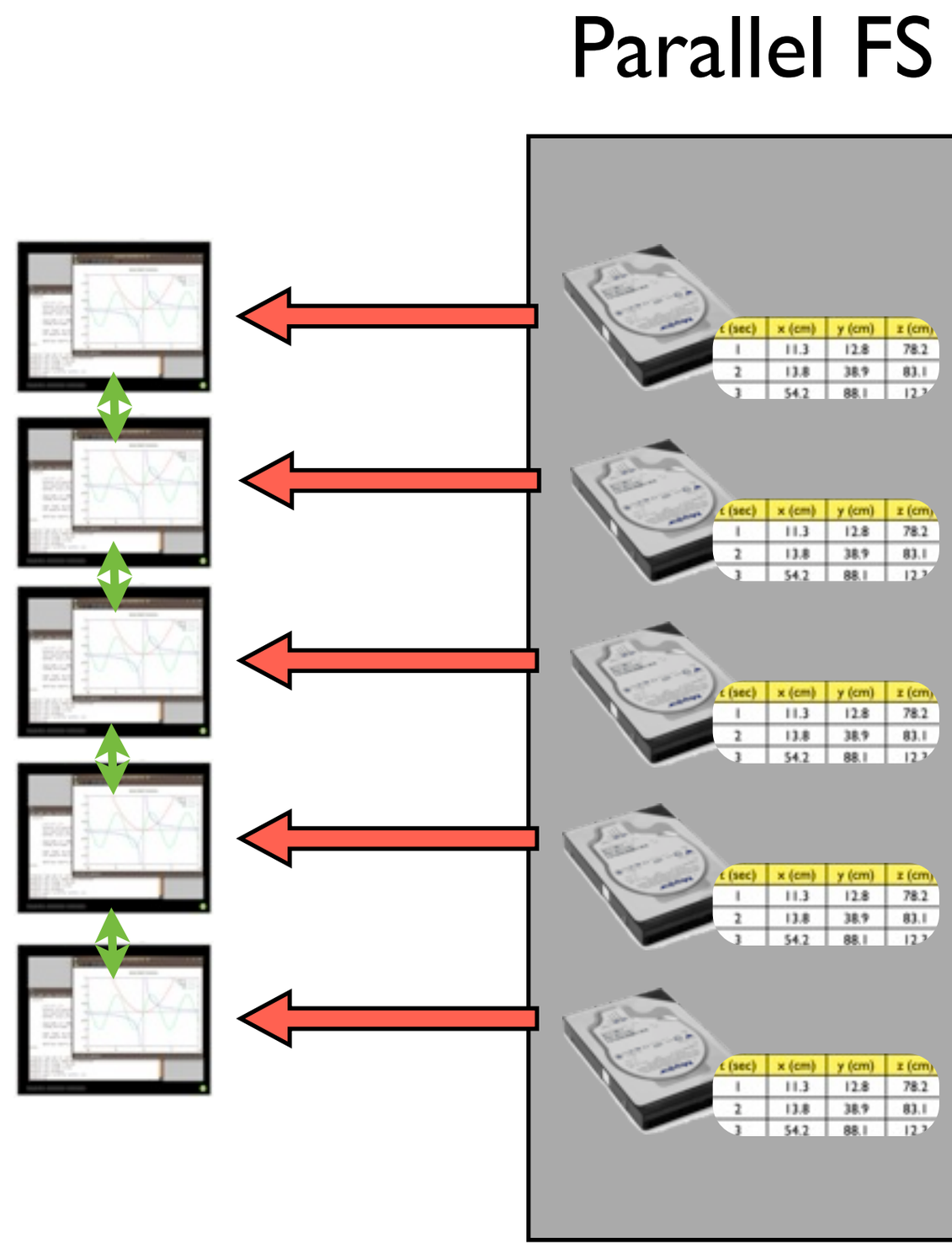
Striping data across disks

- Single client can make use of multiple disk systems simultaneously
- “Stripe” file across many drives
- One drive can be finding next block while another is sending current block



Parallel operations on separate data

- Or can do truly parallel operations
  - multiple clients doing independent work
  - Easy parallelism (good for lots of small data) - process many small files separately
  - Harder parallelism - each does part of a larger analysis job on a big file.

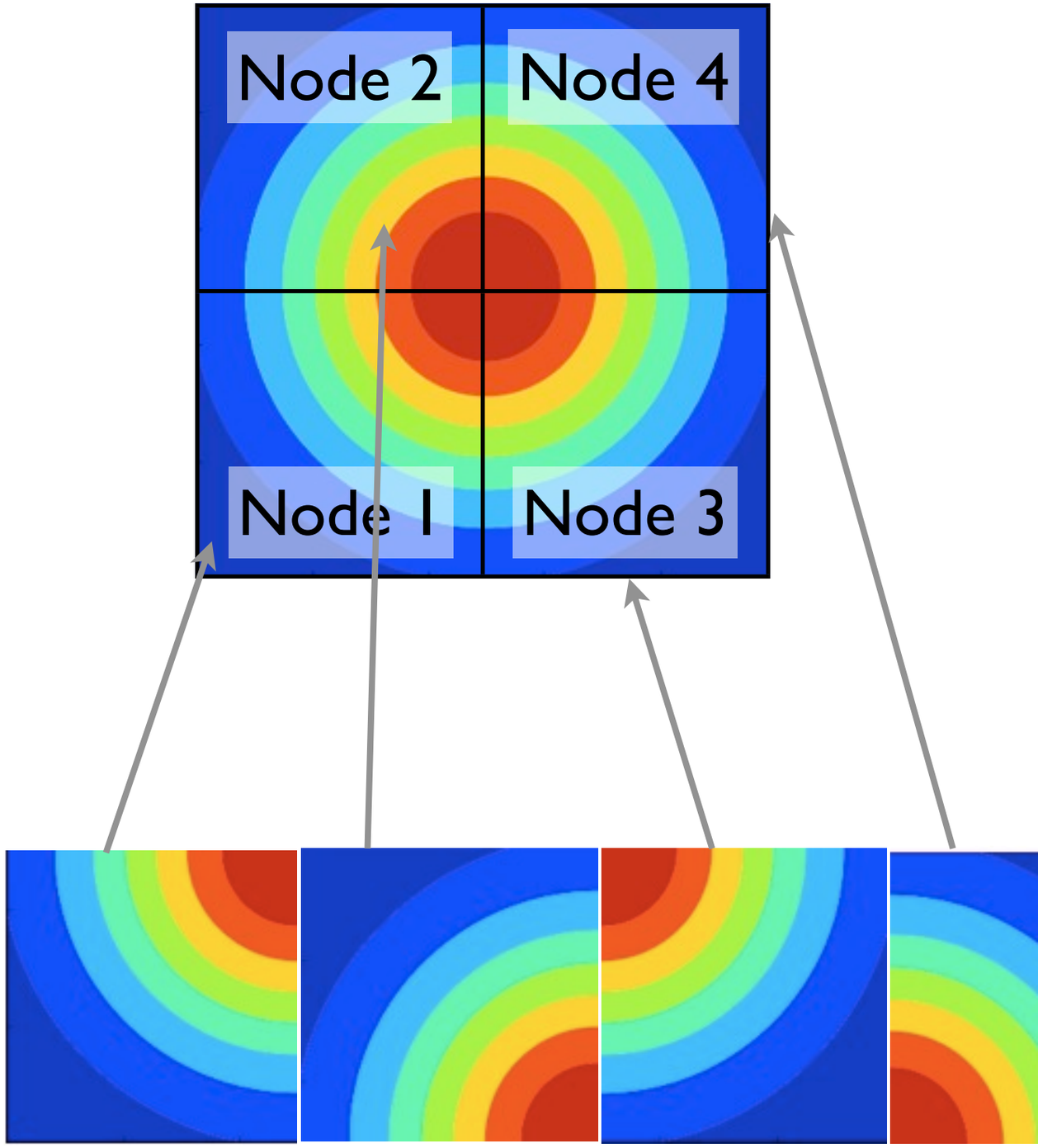


SciNet

compute • calcul CANADA

Data files must take advantage of parallel I/O

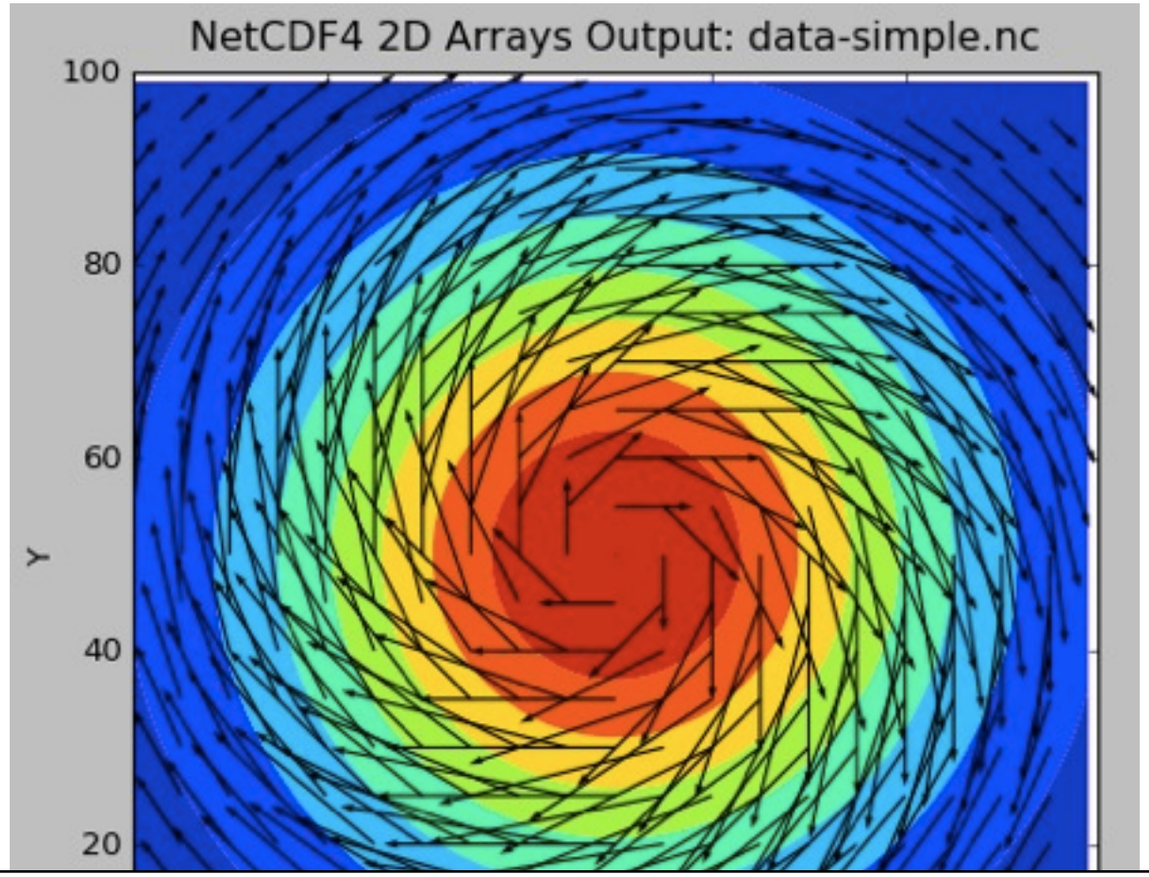
- For parallel operations on single big files, parallel filesystem isn't enough
- Data must be written in such a way that nodes can efficiently access relevant subregions
- HDF5, NetCDF formats typical examples for scientific data



Disk: (HDF5, NetCDF, pVTK..)

These formats are *self-describing*

- HDF5, NetCDF have other advantages anyway
- Binary
- Self describing - contains not only data but names, descriptions of arrays, etc
- Many tools can read these formats
- Big data - formats matter



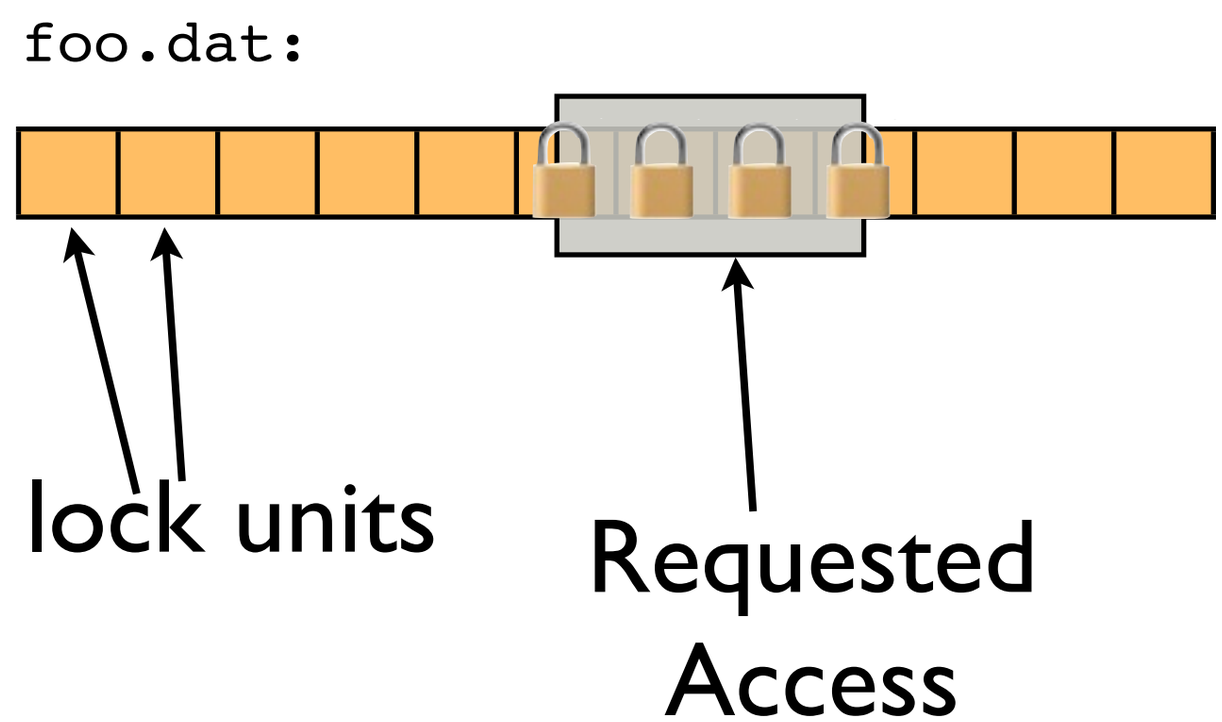

```
$ ncdump -h data-simple-fort.nc
netcdf data-simple-fort {
dimensions:
   X = 100 ;
   Y = 100 ;
   velocity components = 2 ;
variables:
   double Density(Y, X) ;
   double Velocity(Y, X, velocity components) ;
}
```

SciNet

compute • calcul CANADA

Coordinating I/O

- Multiple nodes all accessing same filesystem.
- To prevent anarchy, locks for some requested accesses.
- File broken up into lock units, locks handed out upon request.
- "False sharing", etc, possible.
- Files **and** directories.
- Makes (some) IOPS even more expensive

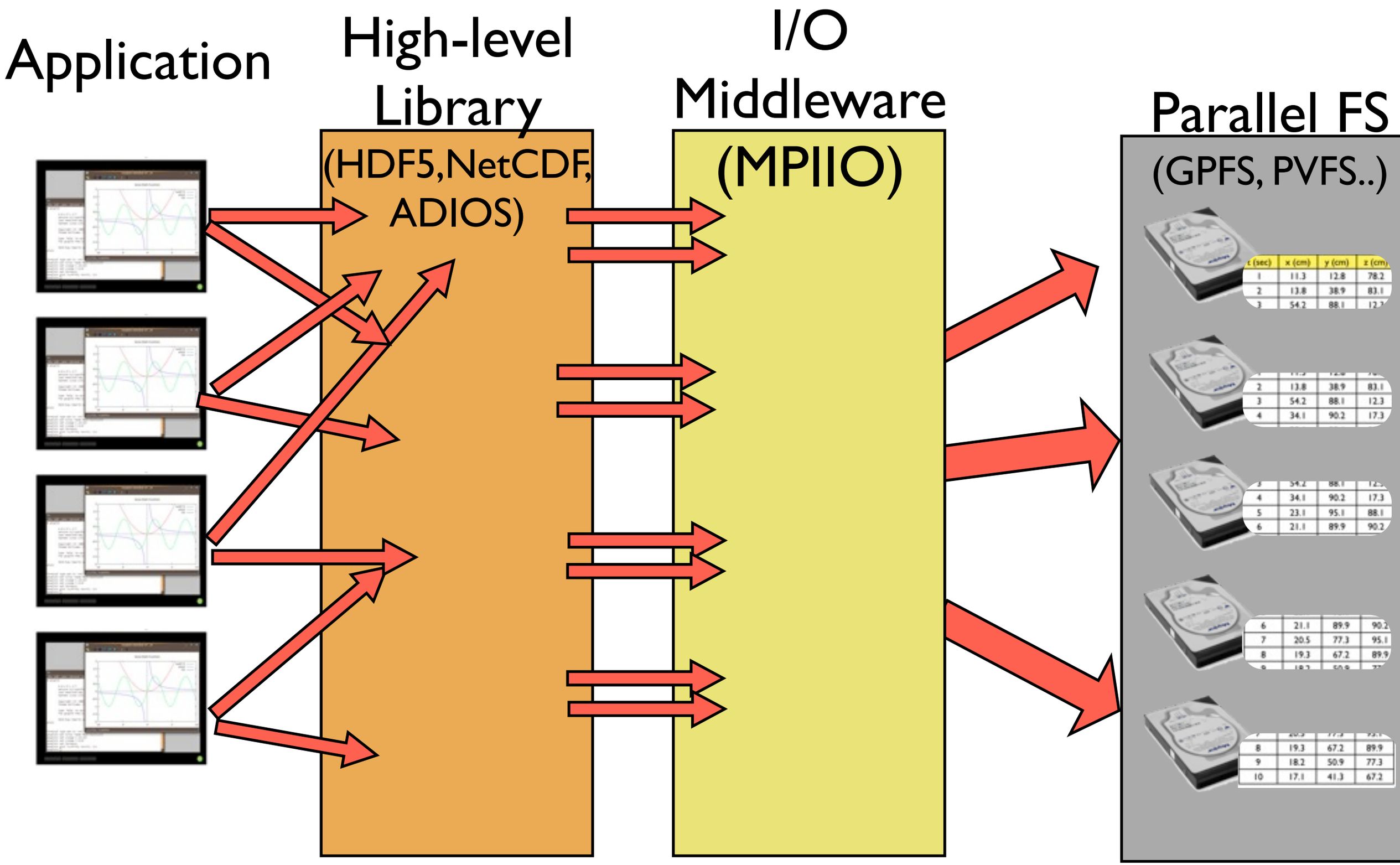


SciNet's File Systems

- Designed for HPC workloads
- High bandwidth to large files - big data.
- Does not do well with millions of little files:
  - lots of small scattered access is *terrible* for performance, even on desktop; multiply by hundreds of processors, can be disastrous

SciNet

compute • calcul CANADA

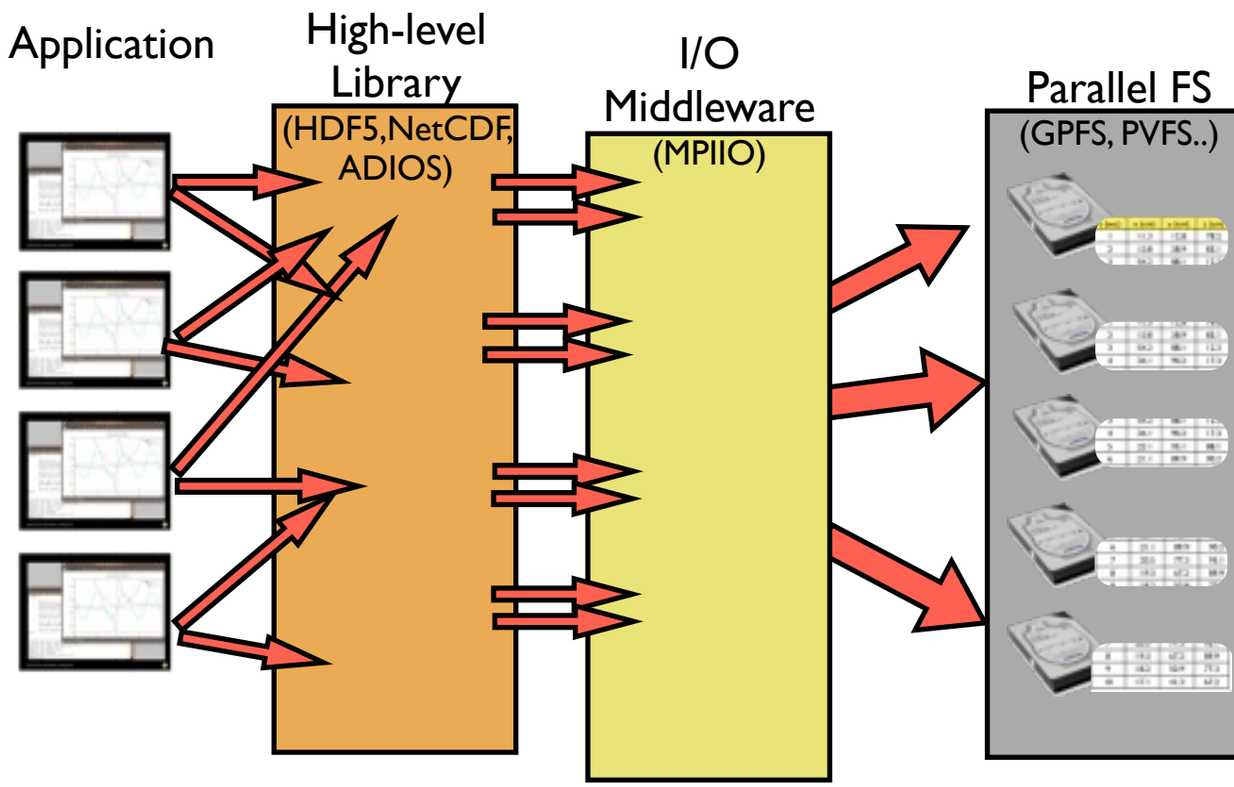


SciNet

compute • calcul CANADA

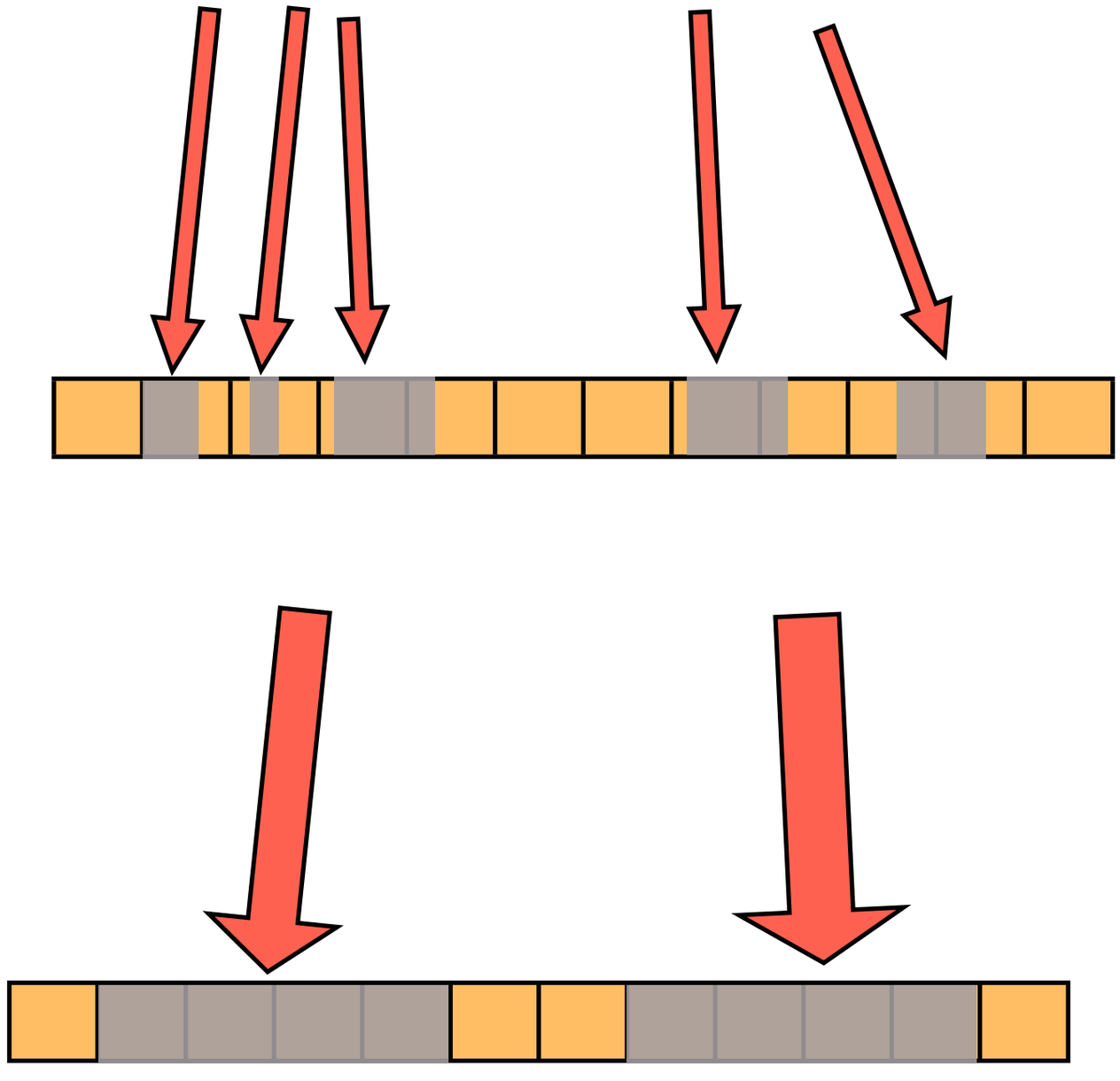
Abstraction Layers

- High Level libraries can simplify programmers tasks
  - Express IO in terms of the data structures of the code, not bytes and blocks
- I/O middleware can coordinate, improve performance
  - Data Sieving
  - 2-phase I/O



Data Sieving

- Combine many non-contiguous IO requests into fewer, bigger IO requests
- "Sieve" unwanted data out
- Reduces IOPS, makes use of high bandwidth for sequential IO

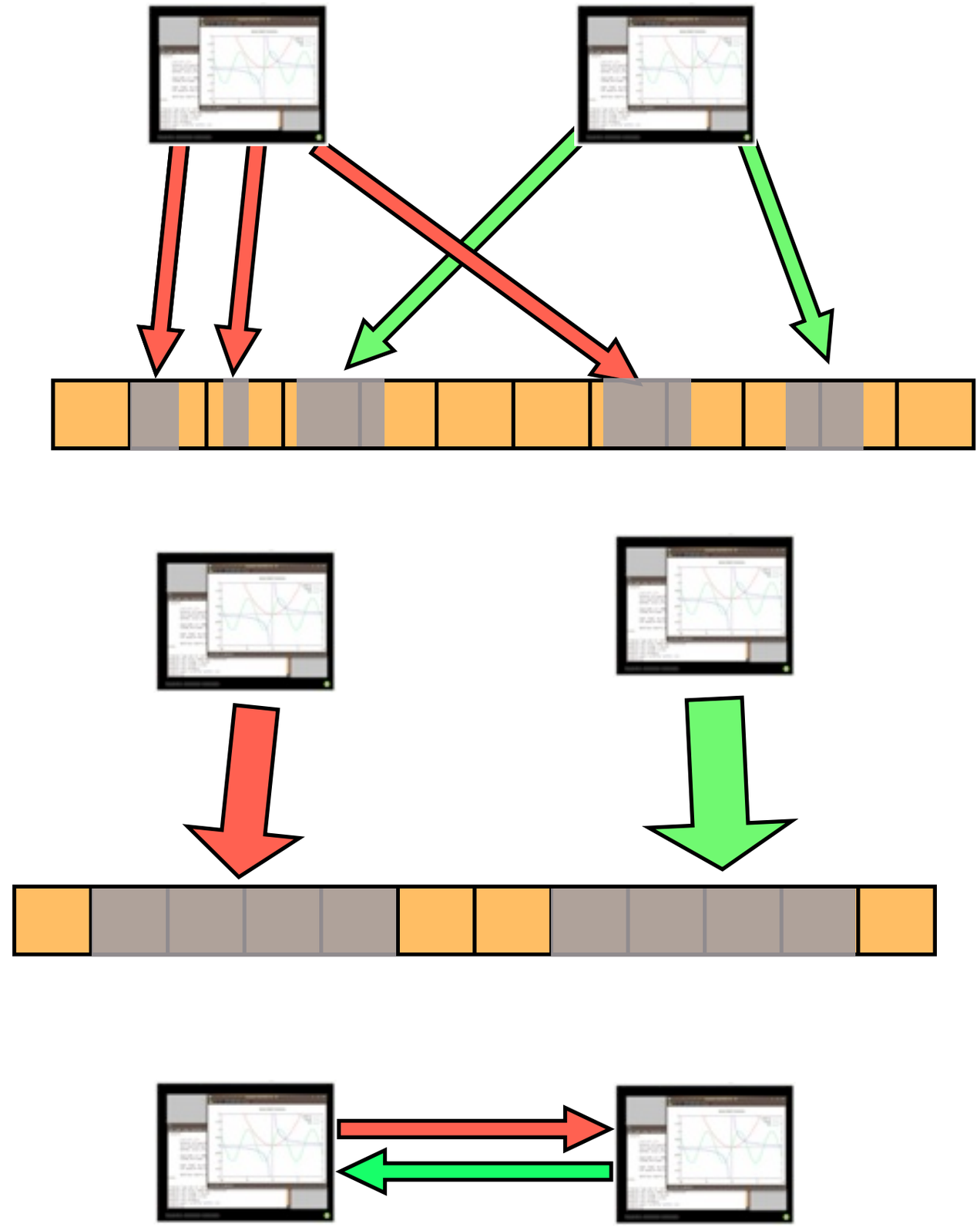


SciNet

compute • calcul CANADA

Two-Phase IO

- Collect requests into larger chunks
- Have individual nodes read big blocks
- Then use network communications to exchange pieces
- Fewer IOPS, faster IO
- Network communication usually faster

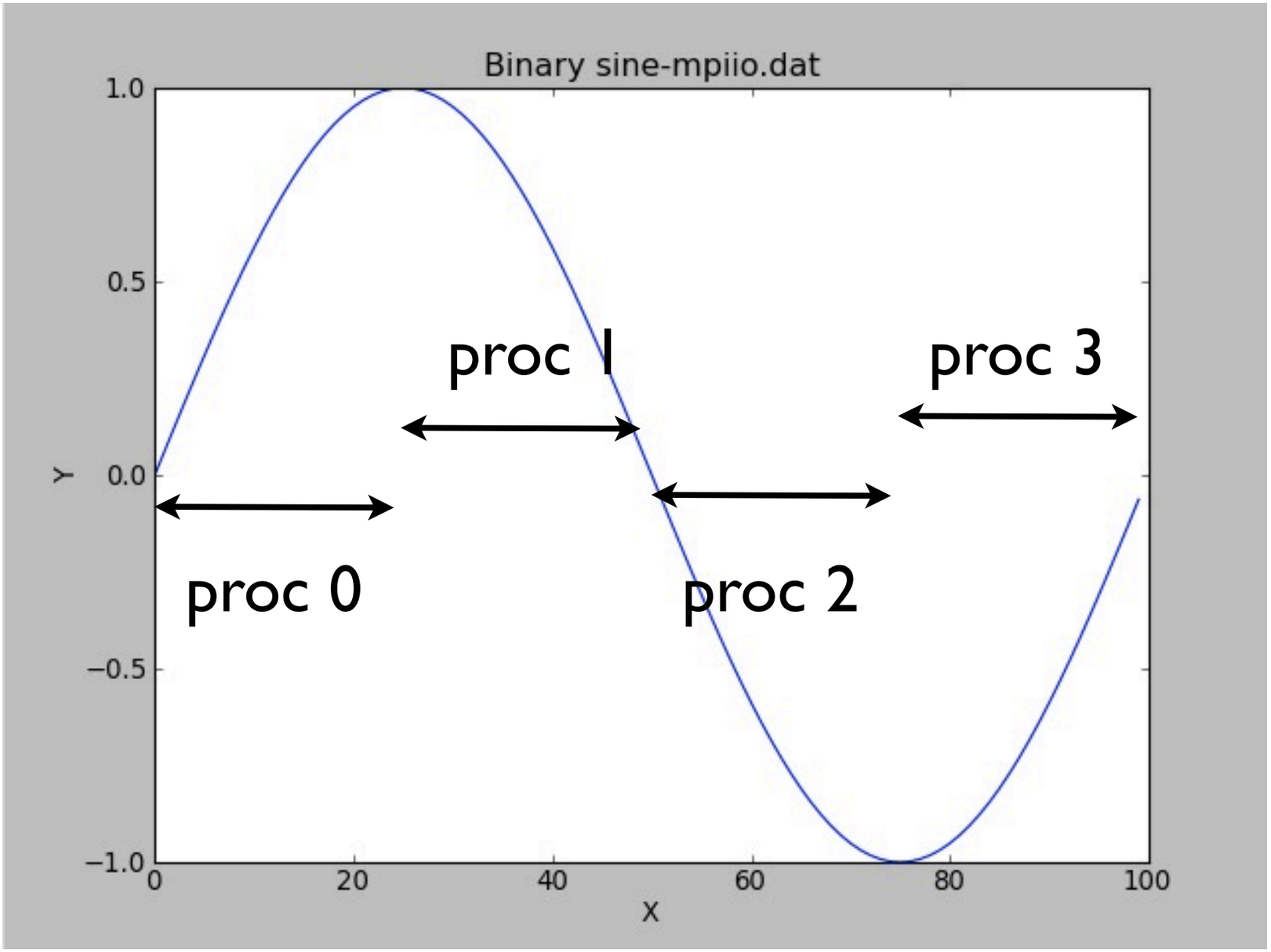


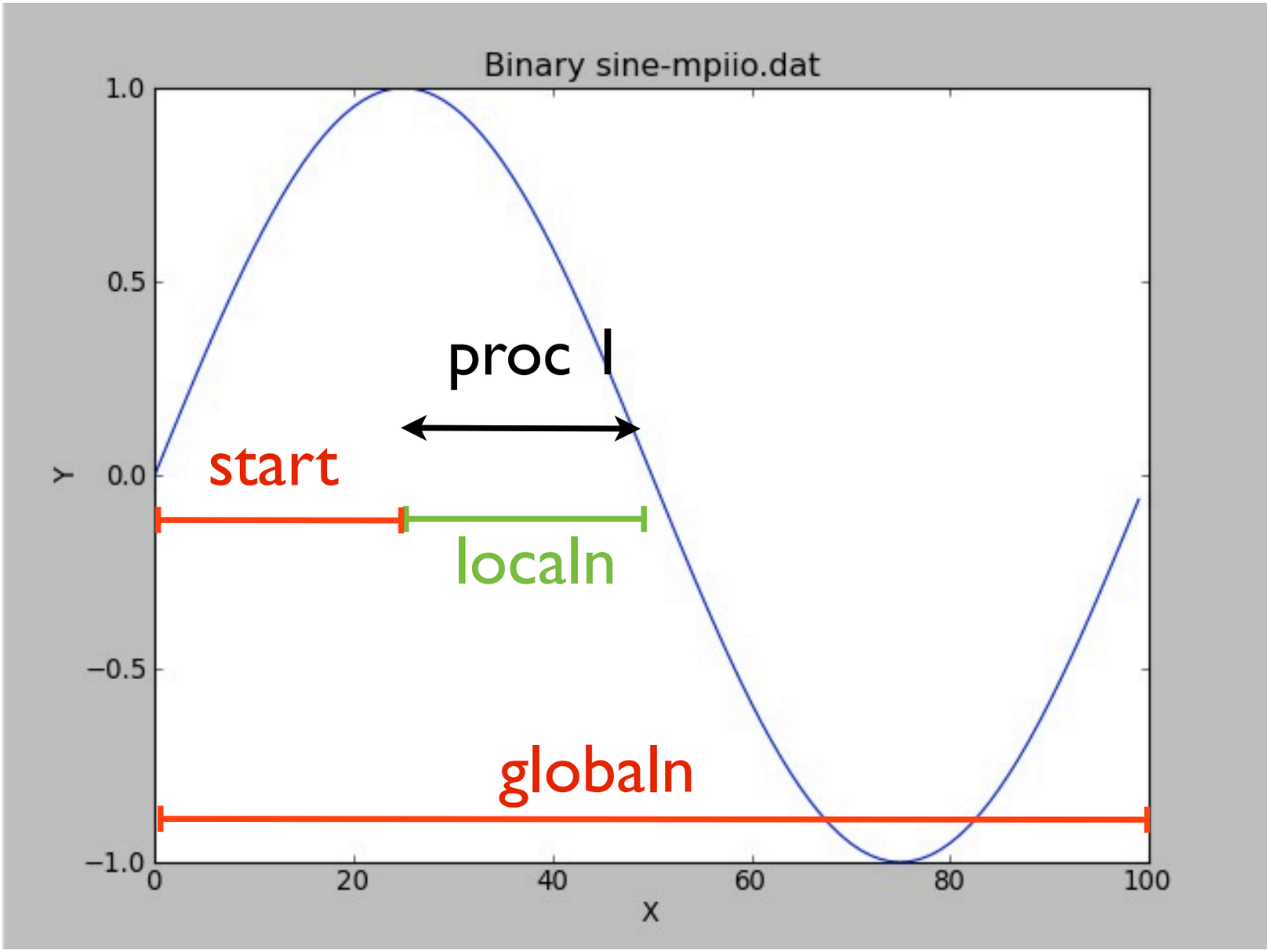
SciNet

compute • calcul CANADA

MPI-IO

- Part of MPI-2 standard
- Started at IBM Watson
- Maps I/O reads and writes to message passing
- ROMIO is the implementation found in MPICH2, OpenMPI
- Really only widely-available scientific computing parallel I/O middleware


Binary sine-mpiio.dat
1.0
0.5
0.0
−0.5
−1.0
Y
0
20
40
60
80
100
X
proc 0
proc 1
proc 2
proc 3



SciNet

compute • calcul
CANADA

sine-mpiio.c

```
    data = malloc(rundata.localn * sizeof(double));

    for (int i=0; i<rundata.localn; i++)
        data[i] = sin((rundata.start+i)*rundata.dx);

    writempiiofile(rundata, data);
```

sine-mpiio.c

```
MPI_File_open(MPI_COMM_WORLD, rundata.filename,
              MPI_MODE_CREATE|MPI_MODE_WRONLY,
              MPI_INFO_NULL, &file);

offset = 0;

int globalsizes[1] = {rundata.globaln};
int localsizes [1] = {rundata.localn};
int starts     [1] = {rundata.start};

MPI_Type_create_subarray( 1, globalsizes, localsizes, starts,
                          MPI_ORDER_C, MPI_DOUBLE, &viewtype );
MPI_Type_commit(&viewtype);

MPI_File_set_view(file, offset,  MPI_DOUBLE, viewtype,
                       "native", MPI_INFO_NULL);

MPI_File_write_at_all(file, offset*sizeof(double), data, rundata.localn,
                      MPI_DOUBLE, &status);

MPI_File_close(&file);
```

MPI_File_Open

Communicator; **collective** operation.

```
call MPI_File_Open( integer        communicator,
                    character(*)  *filename,
                    integer        mode,
                    integer        info,
                    integer        handle,
                    integer        ierr);
```

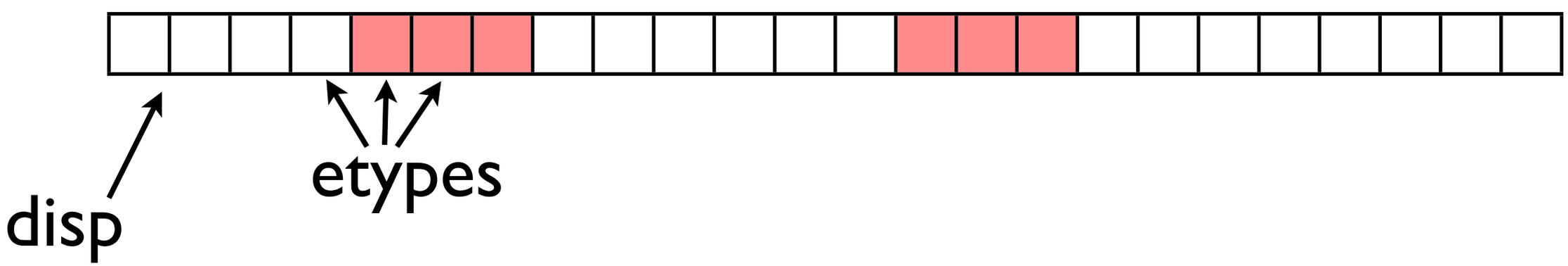
```
int MPI_File_Open( MPI_Comm  communicator,
                   char     *filename,
                   int       mode,
                   MPI_Info  info,
                   MPI_File *handle);
```

SciNet

compute • calcul CANADA

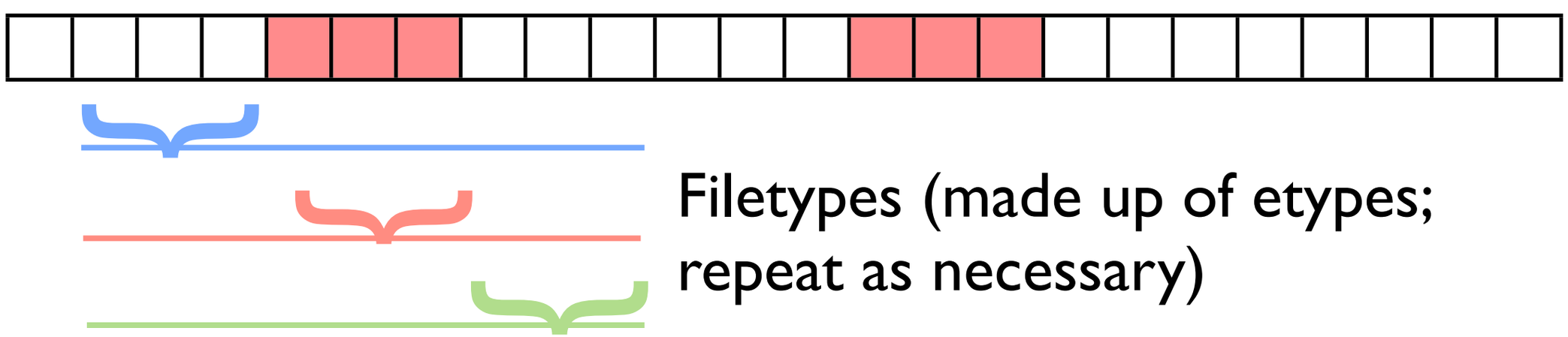
MPI-IO File View

- int MPI_File_set_view(
  MPI_File fh,
  MPI_Offset disp, /* displacement in *bytes* from start */
  MPI_Datatype etype, /* elementary type */
  MPI_Datatype filetype, /* file type; prob different for each proc */
  char *datarep, /* 'native' or 'internal' */
  MPI_Info info) /* MPI_INFO_NULL */



MPI-IO File View

- int MPI_File_set_view(
  MPI_File fh,
  MPI_Offset disp, /* displacement in *bytes* from start */
  MPI_Datatype etype, /* elementary type */
  MPI_Datatype filetype, /* file type; prob different for each proc */
  char *datarep, /* 'native' or 'internal' */
  MPI_Info info) /* MPI_INFO_NULL */



Filetypes (made up of etypes; repeat as necessary)

MPI-IO File Write

- int MPI_File_write_all(
  MPI_File fh,
  void *buf,
  int count,
  MPI_Datatype datatype,
  MPI_Status *status)

Writes (_all: collectively) to part of file **within view.**

SciNet

compute • calcul
CANADA

Workflow with MPI-IO

- Create a file view which describes our "view" of file (or part thereof)
- read, write that file, collective or individual

```
MPI_File_open(MPI_COMM_WORLD, rundata.filename,
              MPI_MODE_CREATE|MPI_MODE_WRONLY,
              MPI_INFO_NULL, &file);

offset = 0;

int globalsizes[1] = {rundata.globaln};
int localsizes [1] = {rundata.localn};
int starts     [1] = {rundata.start};

MPI_Type_create_subarray( 1, globalsizes, localsizes, st
                          MPI_ORDER_C, MPI_DOUBLE, &view
MPI_Type_commit(&viewtype);

MPI_File_set_view(file, offset,  MPI_DOUBLE, viewtype,
                       "native", MPI_INFO_NULL);

MPI_File_write_at_all(file, offset*sizeof(double), data,
                      MPI_DOUBLE, &status);

MPI_File_close(&file);
```

SciNet

compute • calcul CANADA

Upsides of MPI-IO

- Ubiquitous
- Requires no additional libraries

```
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv) {
    int ierr, rank, size;
    MPI_Offset offset;
    MPI_File   file;
    MPI_Status status;
    const int msgsize=6;
    char message[msgsize+1];

    ierr = MPI_Init(&argc, &argv);
    ierr|= MPI_Comm_size(MPI_COMM_WORLD, &size);
    ierr|= MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if ((rank % 2) == 0) strcpy (message, "Hello "); else strcpy (message

    offset = (msgsize*rank);

    MPI_File_open(MPI_COMM_WORLD, "helloworld.txt", MPI_MODE_CREATE|MPI_M
                  MPI_INFO_NULL, &file);
    MPI_File_seek(file, offset, MPI_SEEK_SET);
    MPI_File_write(file, message, msgsize, MPI_CHAR, &status);
    MPI_File_close(&file);

    MPI_Finalize();
    return 0;
}
```

SciNet

compute • calcul CANADA

Downside of MPI-IO

- Very low level
- Writing raw data to file
- Raw binary very "brittle"; adding new variables can break file unless at end
- Different architectures w/ different floating point formats (eg, GPC vs TCS) can be problematic

```
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv) {
    int ierr, rank, size;
    MPI_Offset offset;
    MPI_File   file;
    MPI_Status status;
    const int msgsize=6;
    char message[msgsize+1];

    ierr = MPI_Init(&argc, &argv);
    ierr|= MPI_Comm_size(MPI_COMM_WORLD, &size);
    ierr|= MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if ((rank % 2) == 0) strcpy (message, "Hello "); else strcpy (message

    offset = (msgsize*rank);

    MPI_File_open(MPI_COMM_WORLD, "helloworld.txt", MPI_MODE_CREATE|MPI_M
                  MPI_INFO_NULL, &file);
    MPI_File_seek(file, offset, MPI_SEEK_SET);
    MPI_File_write(file, message, msgsize, MPI_CHAR, &status);
    MPI_File_close(&file);

    MPI_Finalize();
    return 0;
}
```

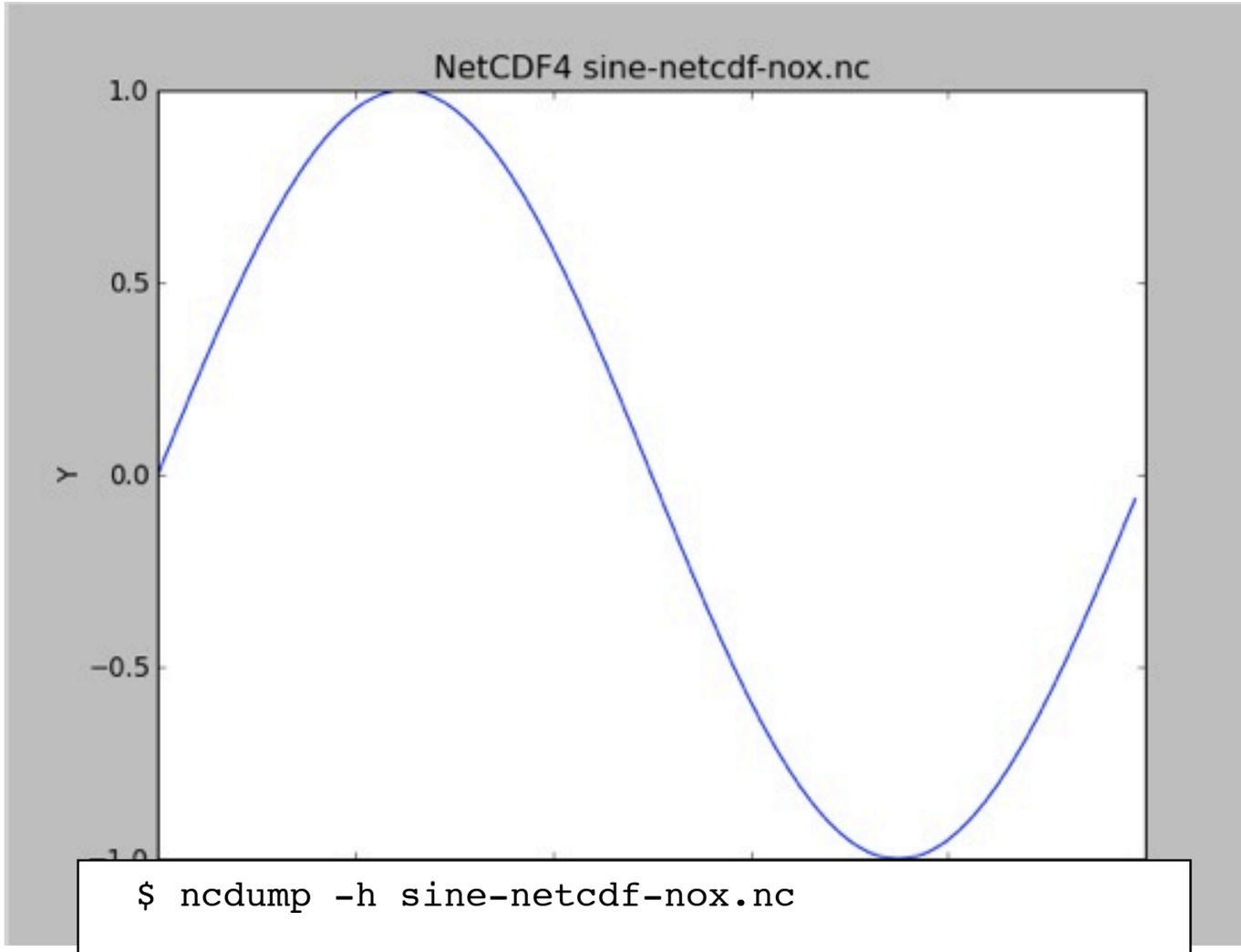
SciNet

compute • calcul CANADA

NetCDF

- NetCDF is a set of libraries and formats for:
  - portable,
  - efficient
  - "self-describing"
- way of storing and accessing large arrays (eg, for scientific data)

- Current version is NetCDF4

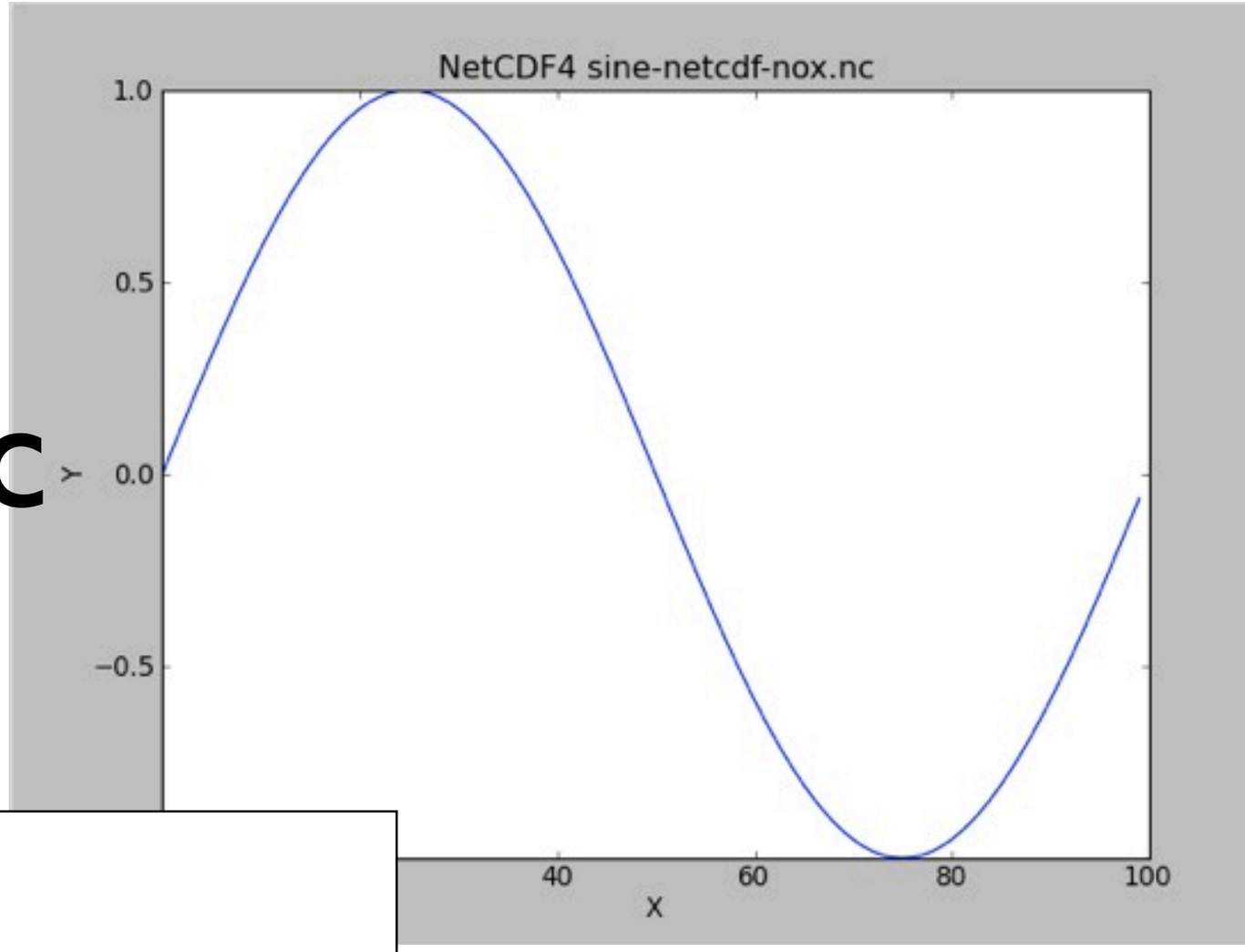



```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
   X = 100 ;
variables:
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

What is this .nc file?



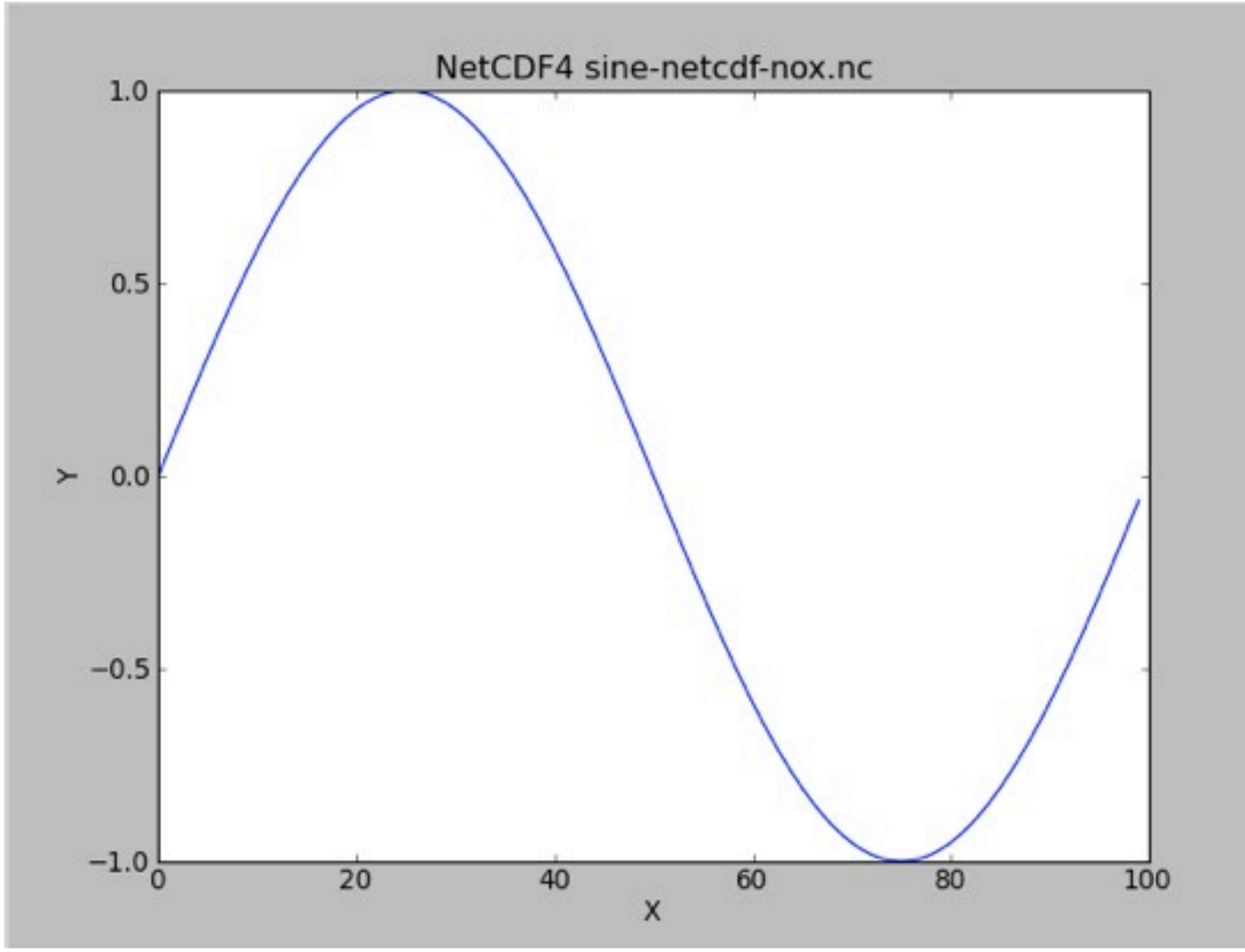

```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
  X = 100 ;
variables:
  double Data(X) ;
      Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

NetCDF: *Portable*

- Binary files, but common output format so that different sorts of machines can share files.
- Libraries accessible from C, C++, Fortran-77, Fortran 90/95/2003, python, etc.



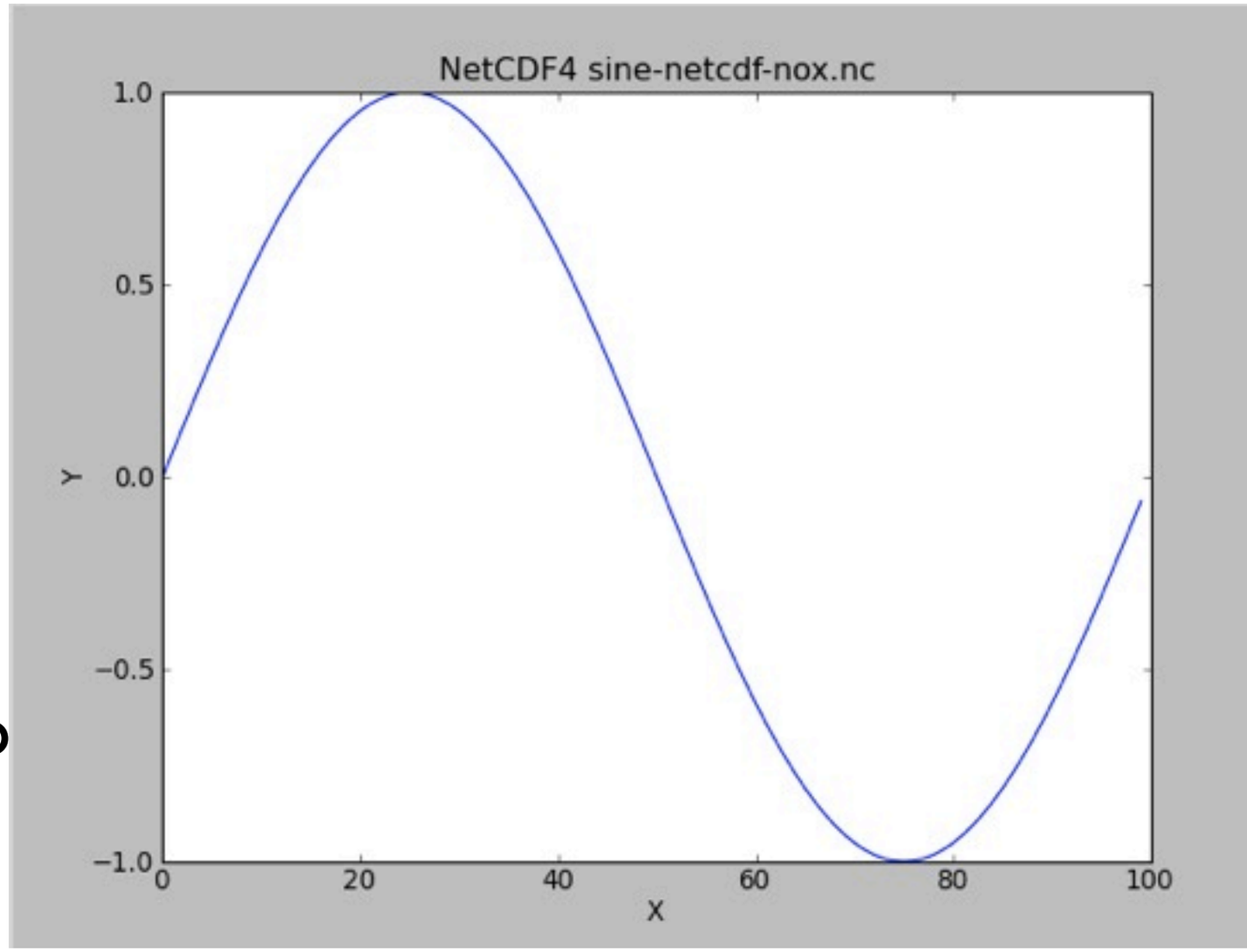

```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
   X = 100 ;
variables:
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

NetCDF: *Self-Describing*

- Header contains the metadata to describe the big data
- Lists:
  - Array names
  - Dimensions
  - *shared* dimensions - information about how the arrays relate
  - Other, related information



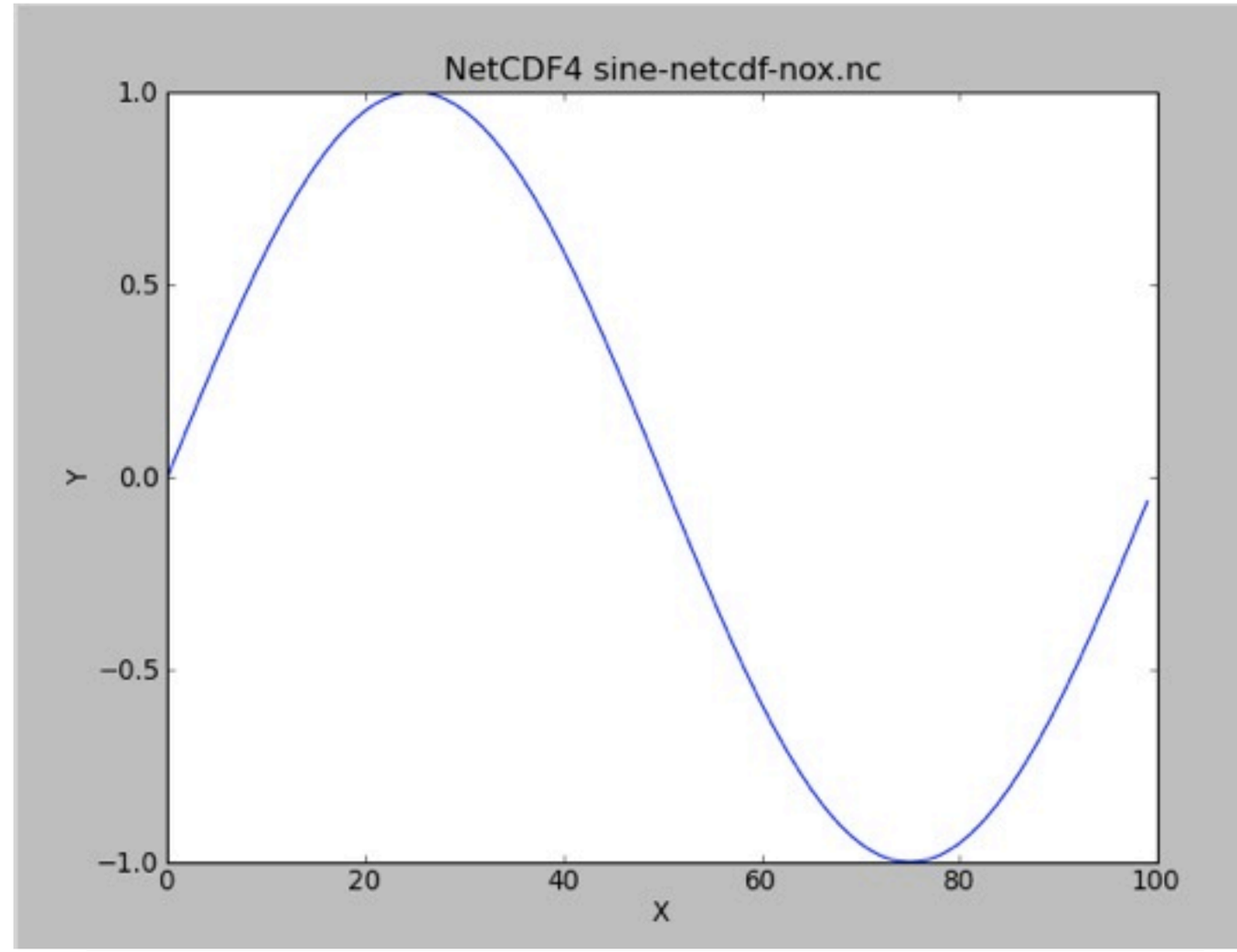

```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
   X = 100 ;
variables:
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

NetCDF: *Efficient*

- Binary, so less translation (as little is used as possible)
- IO libraries themselves are written for performance
- API, data format makes it easy to efficiently read,write subregions of arrays (slices, or 'hyperslabs')
- Still possible to make things slow - lots of metadata queries, modifications




```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
   X = 100 ;
variables:
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

sine-netcdf.c

```
#include <netcdf.h>

void writenetcdffile(rundata_t rundata, double **dens,
                     double ***vel) {
    ...
    status = nc_create_par(rundata.filename, NC_MPIIO | NC_NETCDF4 |
                           NC_CLOBBER, comm, info, &file_id);
    ...
    nc_def_dim(file_id, "X", rundata.globaln, &xdim_id);
    datadims[0] = xdim_id;

    /* define the coordinate variables,... */
    nc_def_var(file_id, "Data",  NC_DOUBLE, 1, datadims, &data_id);
    ...
    nc_enddef(file_id);

    ...
```

Create a new file, with name rundata.filename

sine-netcdf.c

```
#include <netcdf.h>

void writenetcdffile(rundata_t rundata, double **dens,
                     double ***vel) {
    ...
    status = nc_create_par(rundata.filename, NC_MPIIO | NC_NETCDF4 |
                           NC_CLOBBER, comm, info, &file_id);
    ...
    nc_def_dim(file_id, "X", rundata.globaln, &xdim_id);
    datadims[0] = xdim_id;

    /* define the coordinate variables,... */
    nc_def_var(file_id, "Data",  NC_DOUBLE, 1, datadims, &data_id);
    ...
    nc_enddef(file_id);

    ...
```

Define dimensions

sine-netcdf.c

```
#include <netcdf.h>

void writenetcdffile(rundata_t rundata, double **dens,
                     double ***vel) {
    ...
    status = nc_create_par(rundata.filename, NC_MPIIO | NC_NETCDF4 |
                           NC_CLOBBER, comm, info, &file_id);
    ...
    nc_def_dim(file_id, "X", rundata.globaln, &xdim_id);
    datadims[0] = xdim_id;

    /* define the coordinate variables,... */
    nc_def_var(file_id, "Data",  NC_DOUBLE, 1, datadims, &data_id);
    ...
    nc_enddef(file_id);
```

Define variables on the dimensions

sine-netcdf.c

```
nc_var_par_access(file_id, data_id,    NC_COLLECTIVE);

starts[0] = (rundata.start);
counts[0]  = rundata.localn;

nc_put_vara_float (file_id, xcoord_id, starts, counts, &(x[0]));

nc_close(file_id);
```

Set access method (collective, independant) on each variable

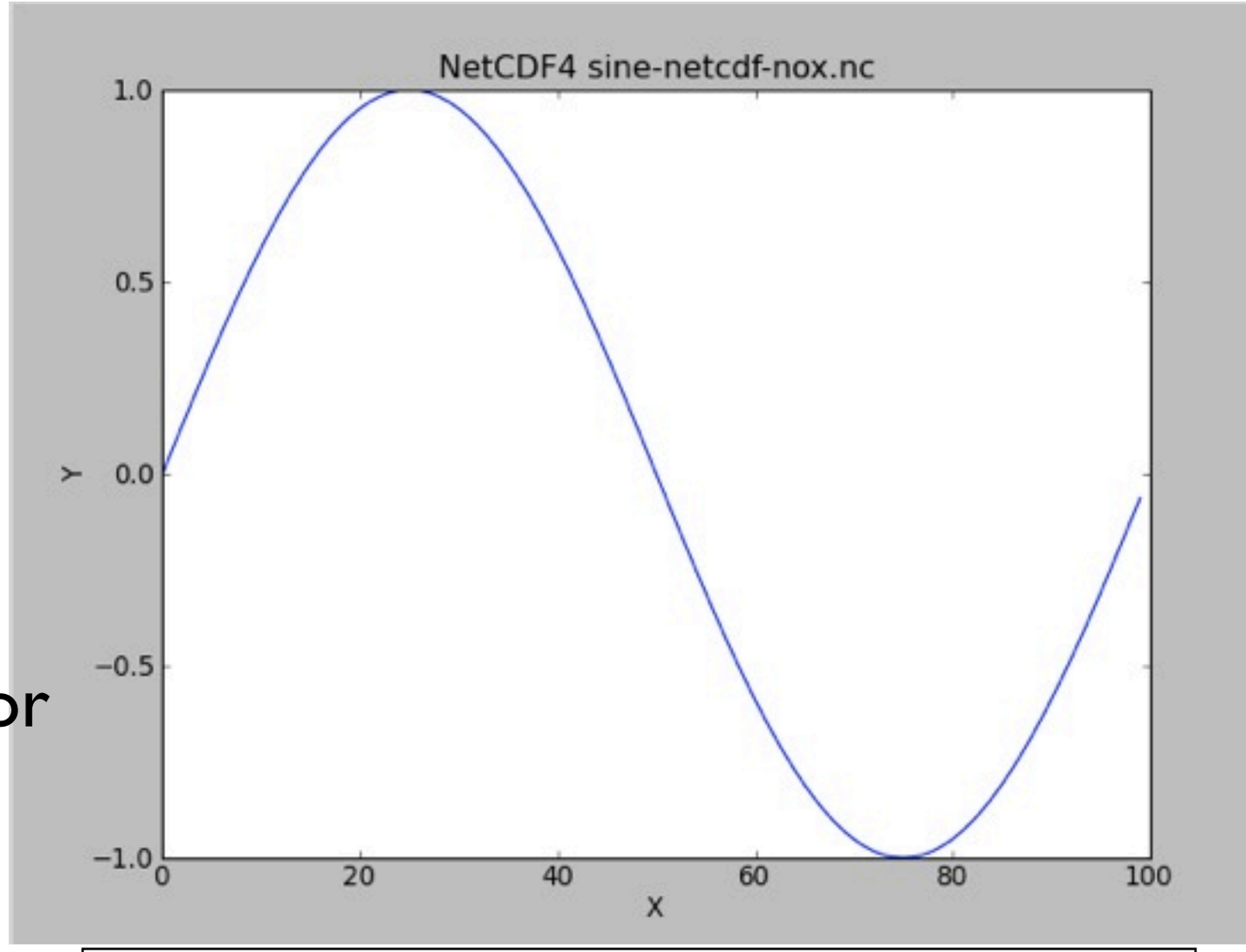
sine-netcdf.c

```
nc_var_par_access(file_id, data_id,    NC_COLLECTIVE);

starts[0] = (rundata.start);
counts[0]  = rundata.localn;

nc_put_vara_double(file_id, data_id,   starts, counts, &(data[0]));

nc_close(file_id);
```

Output our region (starts, counts) of arrays

Writing a NetCDF File

- To write a NetCDF file, we go through the following steps:
  - **Create** the file (or open it for appending)
  - **Define dimensions** of the arrays we'll be writing
  - **Define variables** on those dimensions
  - **End definition** phase
  - **Write variables**
  - **Close file**



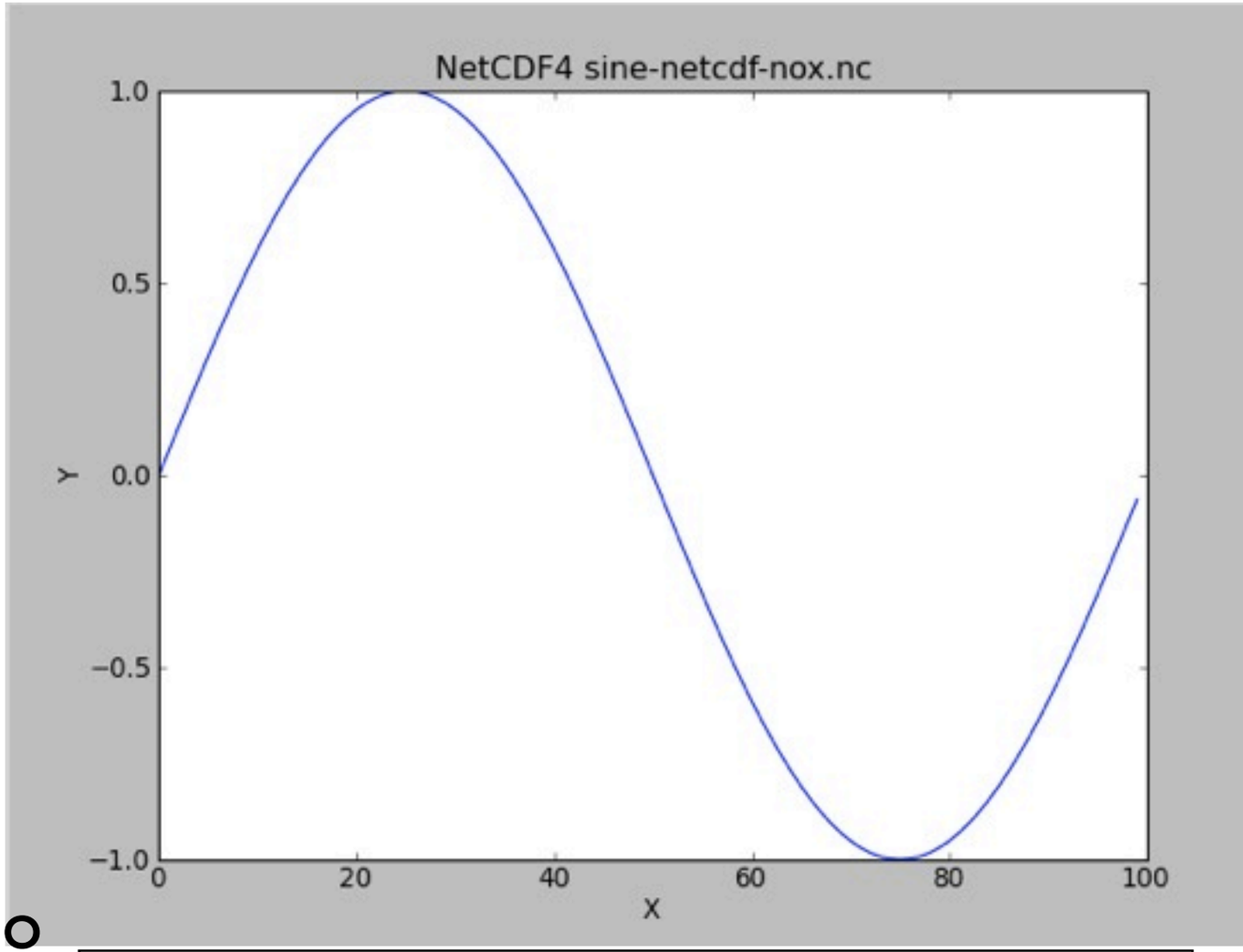

```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
   X = 100 ;
variables:
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

Reading a NetCDF File

- Flow is slightly different
  - **Open** the file for reading
  - **Get dimension ids** of the the dimensions in the files
  - **Get dimension lengths** so you can allocate the files
  - **Get variable ids** so you can access the data
  - **Read variables**
  - **Close file**



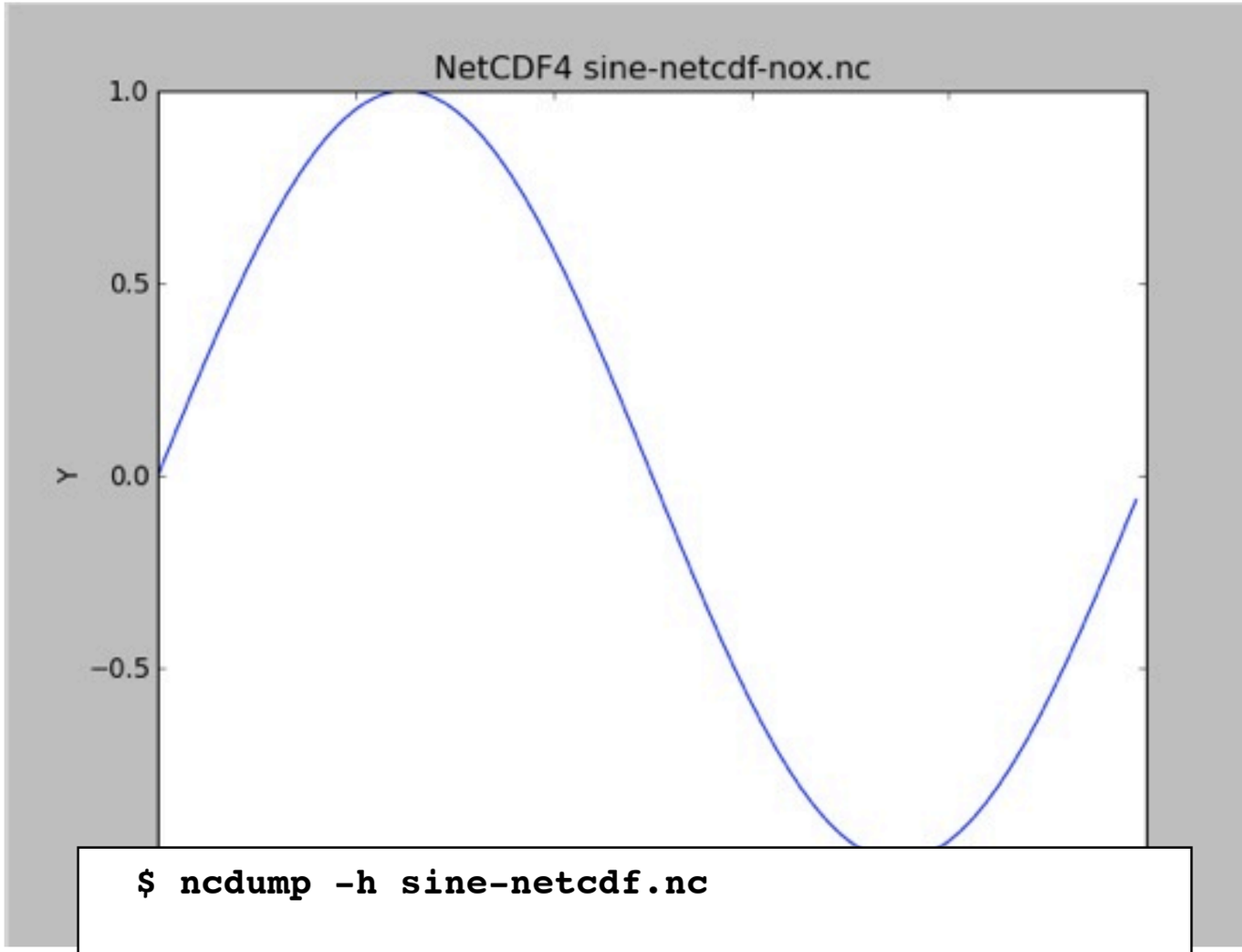

```
$ ncdump -h sine-netcdf-nox.nc

netcdf sine-netcdf-nox {
dimensions:
   X = 100 ;
variables:
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

A better file

- Can easily add variables
- Old plotting routines don't even "see" them - you have to look them up explicitly
- New analysis routines can deal gracefully with older files that don't contain the new variables
- Major advantage of "self-describing" formats.



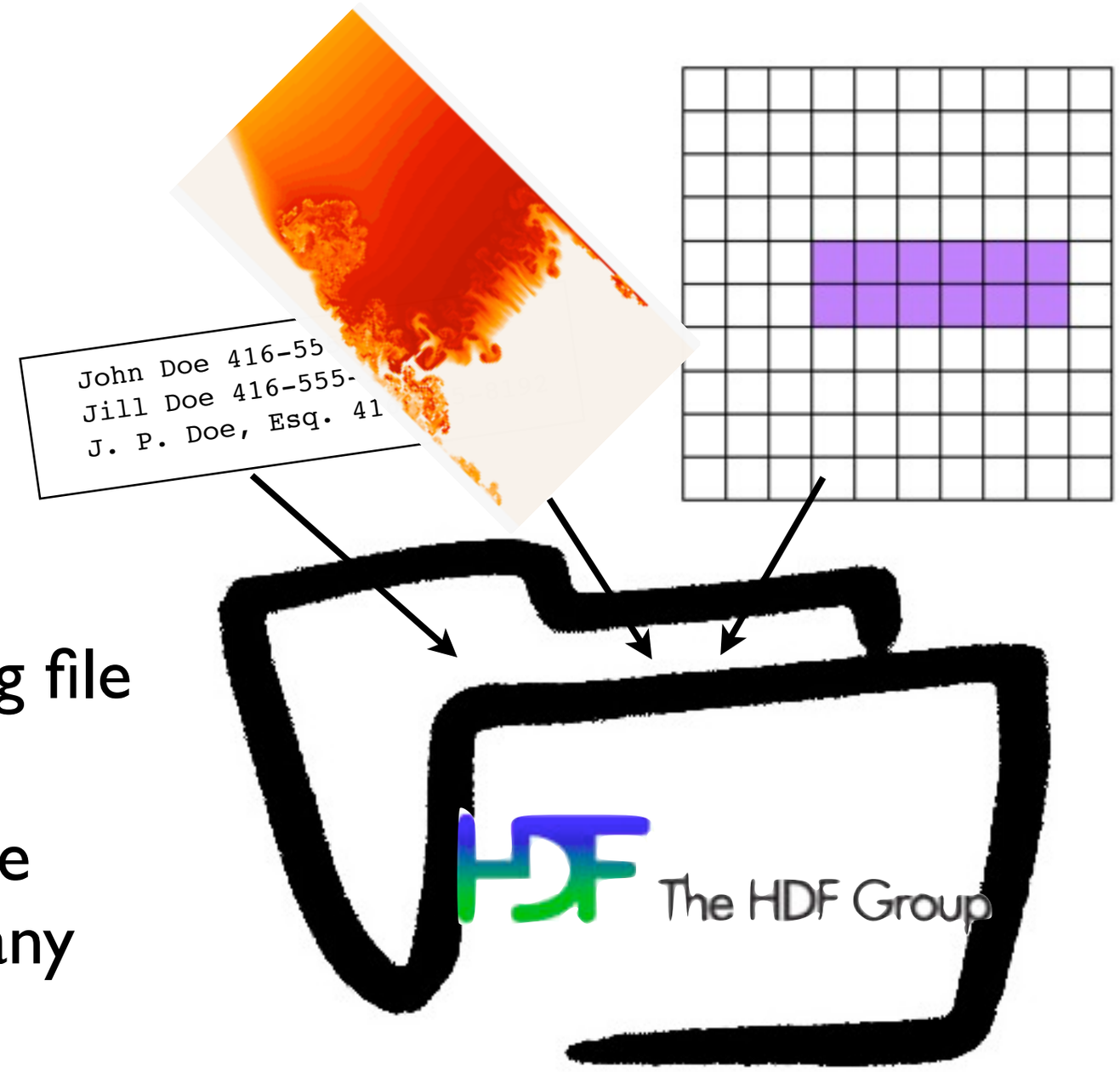

```
$ ncdump -h sine-netcdf.nc

netcdf sine-netcdf {
dimensions:
   X = 100 ;
variables:
   float x\ Coordinates(X) ;
        x\ Coordinates:units = "cm" ;
   double Data(X) ;
        Data:units = "m" ;
}
```

SciNet

compute • calcul CANADA

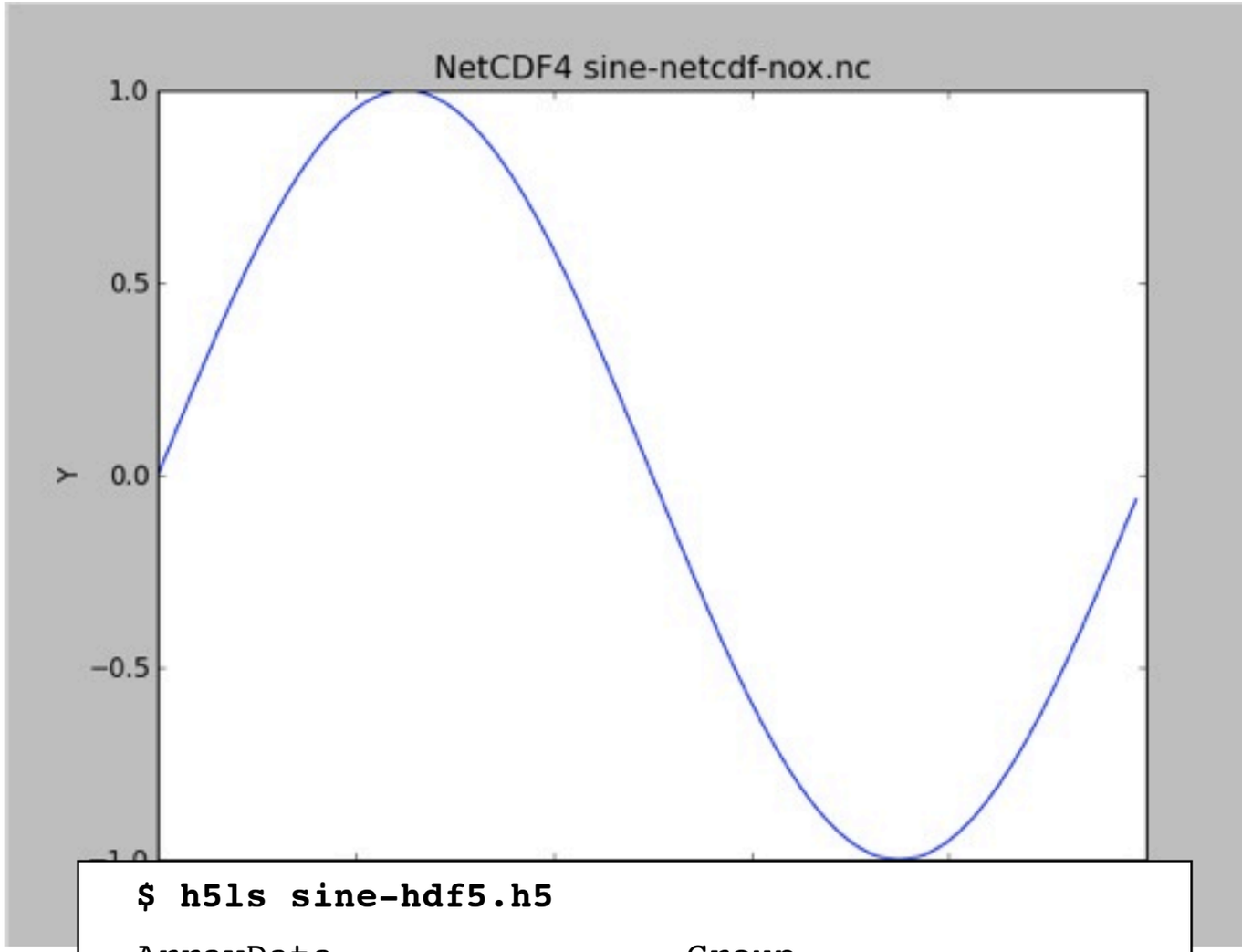
HDF5

- HDF5 is also self-describing file format and set of libraries
- Unlike NetCDF, much more general; can shove almost any type of data in there



HDF5

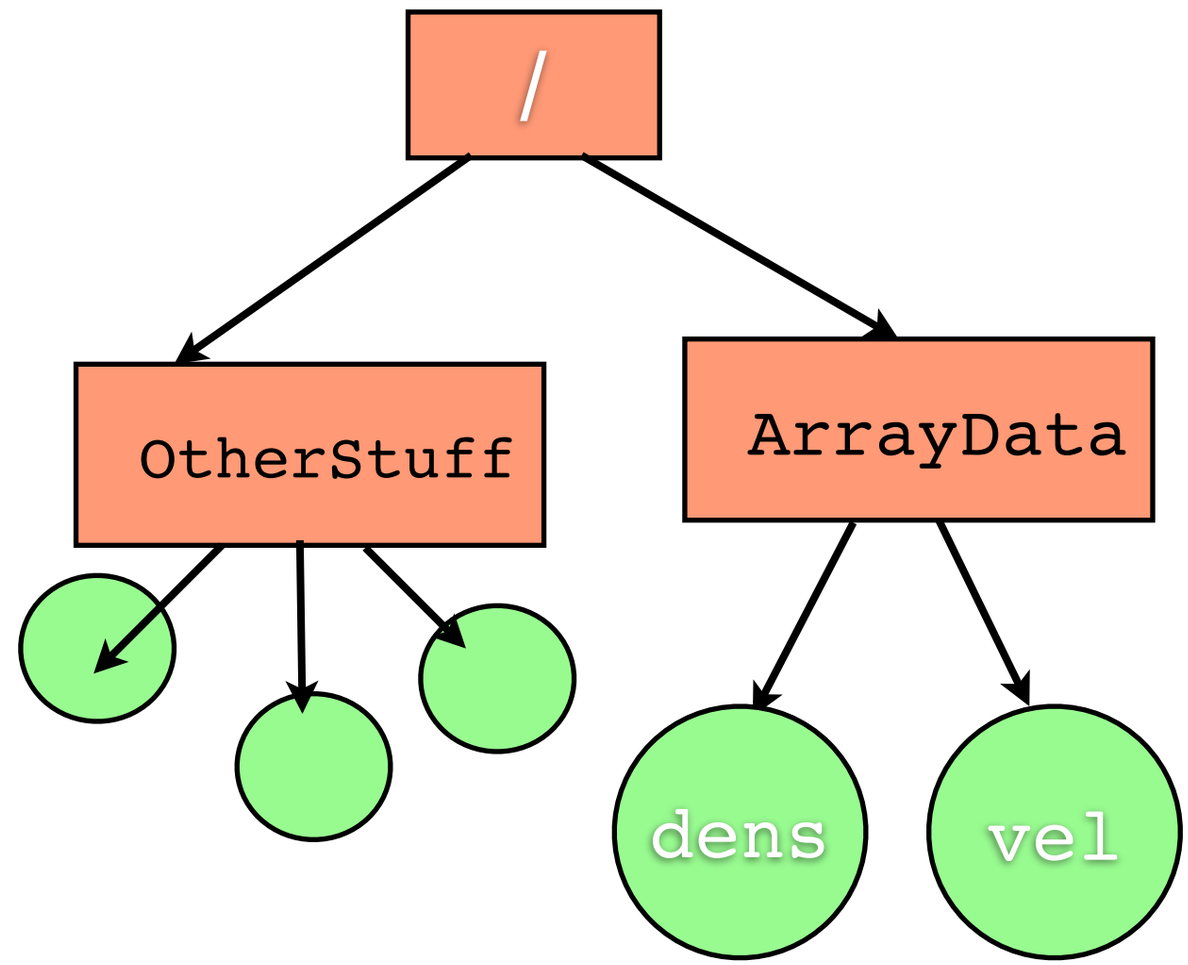
- Similar to NetCDF; header information can be queried by utilities
- File can be arranged into "directories" like file system
- NetCDF4 can now do this too (based on HDF5)




```
$ h5ls sine-hdf5.h5
ArrayData                    Group
OtherStuff                   Group

$ h5ls sine-hdf5.h5/ArrayData
Data                         Dataset {100}
x\ Coordinates               Dataset {100}
```

HDF5 Groups

- HDF5 has a structure a bit like a unix filesystem:
- “Groups” - directories
- “Datasets” - files
- NetCDF4 now has these, but breaks compatibility with NetCDF3 files

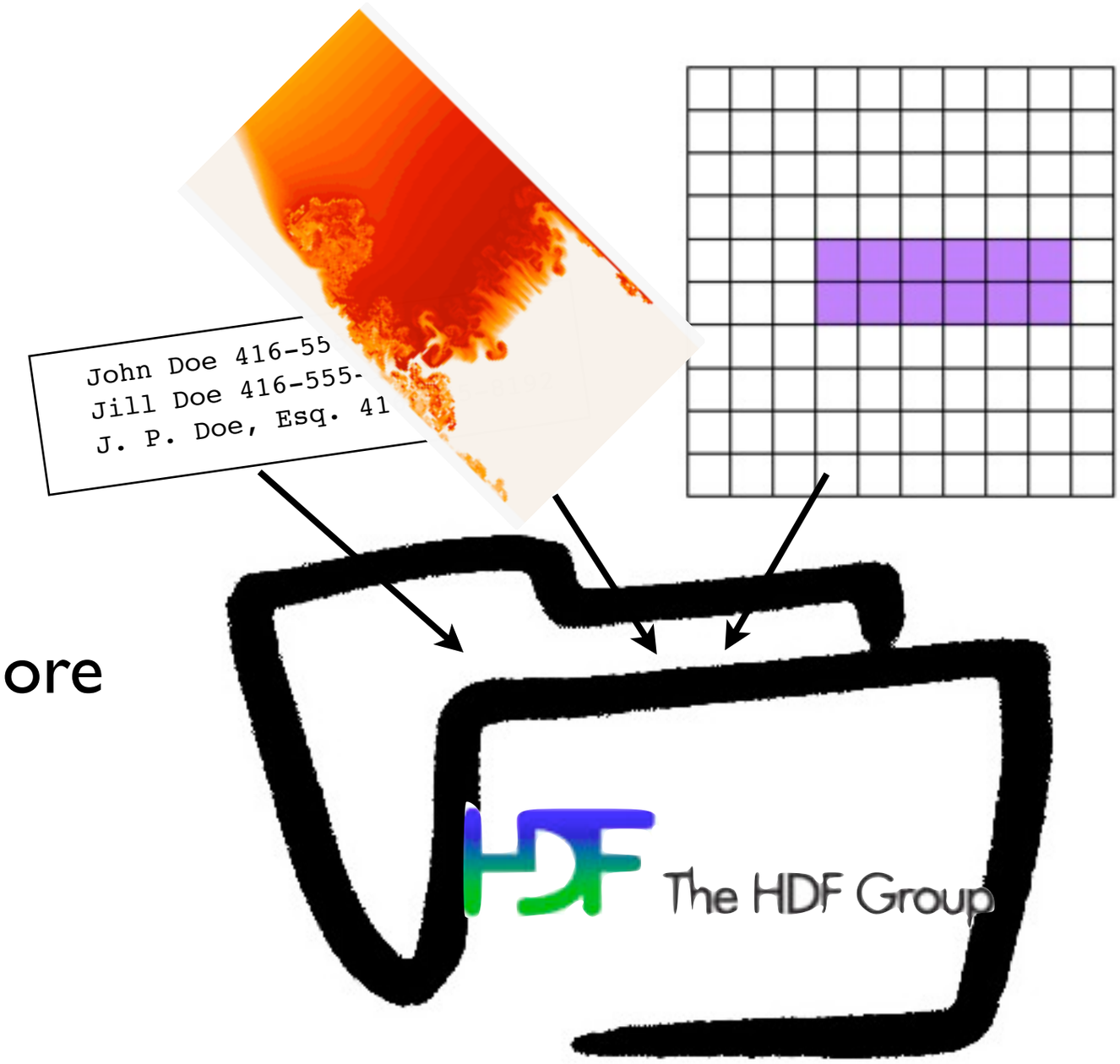


SciNet

compute • calcul CANADA

HDF5

- Much more general, and more low-level than NetCDF.
- (In fact, newest version of NetCDF implemented in HDF5).
- Pro: *can* do more!
- Con: **have** to do more.



SciNet

compute • calcul CANADA

sine-hdf5.c

```
hid_t file_id, arr_group_id, data_dataset_id, x_dataset_id;
hid_t data_dataspace_id, x_dataspace_id;
...
/* sizes */
hsize_t datadims[1], xdims[1];
hsize_t locdatadims[1], locxdims[1];
...
/* parameters of the hyperslab */
hsize_t counts[1];
hsize_t strides[1];
...
fap_id = H5Pcreate(H5P_FILE_ACCESS);
/* Include the file access property with IBM hint */
H5Pset_fapl_mpio(fap_id, MPI_COMM_WORLD, info);

/* Set up the parallel environment */
dist_id = H5Pcreate(H5P_DATASET_XFER);
/* we'll be writing collectixy */
H5Pset_dxpl_mpio(dist_id, H5FD_MPIO_COLLECTIVE);
```

NetCDF used ints for everything - HDF5 distinguishes between ids, sizes, errors, uses its own types.

calcul ADA

sine-hdf5.c

```
hid_t file_id, arr_group_id, data_dataset_id, x_dataset_id;
hid_t data_dataspace_id, x_dataspace_id;
...
/* sizes */
hsize_t datadims[1], xdims[1];
hsize_t locdatadims[1], locxdims[1];
...
/* parameters of the hyperslab */
hsize_t counts[1];
hsize_t strides[1];
...
fap_id = H5Pcreate(H5P_FILE_ACCESS);
/* Include the file access property with IBM hint */
H5Pset_fapl_mpio(fap_id, MPI_COMM_WORLD, info);

/* Set up the parallel environment */
dist_id = H5Pcreate(H5P_DATASET_XFER);
/* we'll be writing collectixy */
H5Pset_dxpl_mpio(dist_id, H5FD_MPIO_COLLECTIVE);
```

H5F, H5P... ?

calcul ADA

Decomposing the HDF5 API

- HDF5 API is large
- Constants, function calls start with H5x; x tells you what part of the library
- Table tells you (some) of those parts...
- Fortran the same, but usually end with _F

H5A	**A**ttributes
H5D	**D**atasets
H5E	**E**rrors
H5F	**F**iles
H5G	**G**roups
H5P	**P**roperties
H5S	Data **S**paces
H5T	Data **T**ypes

SciNet

compute • calcul CANADA

sine-hdf5.c

```
    file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT, fap_id);


    /* Create a new group within the new file */
    arr_group_id = H5Gcreate(file_id,"/ArrayData", H5P_DEFAULT, H5P_DEFAULT,
H5P_DEFAULT);


    data_dataspace_id = H5Screate_simple(1, datadims, NULL);
    ...


    data_dataset_id = H5Dcreat(file_id,
                               data_data
                               H5P_DEFAU


    loc_data_dataspace_id = H5Screate_si
    loc_x_dataspace_id    = H5Screate_si


    globaldataspace = H5Dget_space(data_
    H5Sselect_hyperslab(globaldataspace,
counts, blocks);


    H5Dwrite(data_dataset_id, H5T_NATIVE_DOUBLE, loc_data_dataspace_id,
```

All data (in file or in mem) in HDF5 has a dataspace it lives in.

In NetCDF, just cartesian product of dimensions; here more general

sine-hdf5.c

```
    file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT, fap_id);


    /* Create a new group within the new 
    arr_group_id = H5Gcreate(file_id,"/Ar
H5P_DEFAULT);


    data_dataspace_id = H5Screate_simpl(
    ...


    data_dataset_id = H5Dcre e(file_id,
                                data_data
                                H5P_DEFAULT);


    loc_data_dataspace_id = H5Screate_simple(1, locdatadims, NULL);
    loc_x_dataspace_id    = H5Screate_simple(1, locxdims,  NULL);


    globaldataspace = H5Dget_space(data_dataset_id);
    H5Sselect_hyperslab(globaldataspace, H5S_SELECT_SET, offsets, strides,
counts, blocks);


    H5Dwrite(data_dataset_id, H5T_NATIVE_DOUBLE, loc_data_dataspace_id,
```

Creating a data set like defining a variable in NetCDF.
Also declare the type you want it to be on disk.

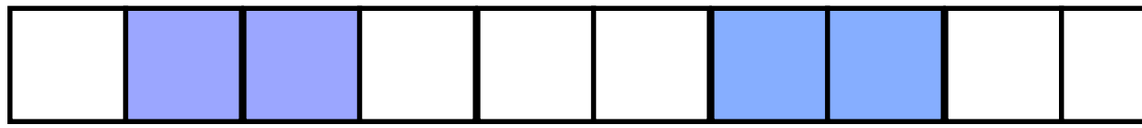
sine-hdf5.c

```
    file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT, fap_id);

    /* Create a new group within the ne
    arr_group_id = H5Gcreate(file_id,"/
H5P_DEFAULT);

    data_dataspace_id = H5Screate_simpl
    ...

    data_dataset_id = H5Dcreate(file_id
                                data_dataspace_id, H5P_DEFAULT, H5P_DEFAULT,
                                H5P_DEFAULT);

    loc_data_dataspace_id = H5Screate_simple(1, locdatadims, NULL);
    loc_x_dataspace_id    = H5Screate_simple(1, locxdims,  NULL);

    globaldataspace = H5Dget_space(data_dataset_id);
    H5Sselect_hyperslab(globaldataspace, H5S_SELECT_SET, offsets, strides,
counts, blocks);

    H5Dwrite(data_dataset_id, H5T_NATIVE_DOUBLE, loc_data_dataspace_id,
```

Write memory from all of memory to all of the dataset on the file. Values in mem are in the native double precision format.

HDF5 Hyperslabs

- Parallel HDF5 similar to parallel NetCDF - fairly modest changes to structure of code
- Different (more low-level, natch) way of dealing with sub-regions
- Offset, block, count, stride

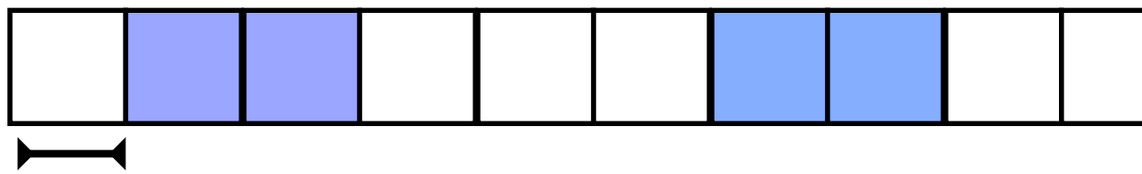


SciNet

compute • calcul CANADA

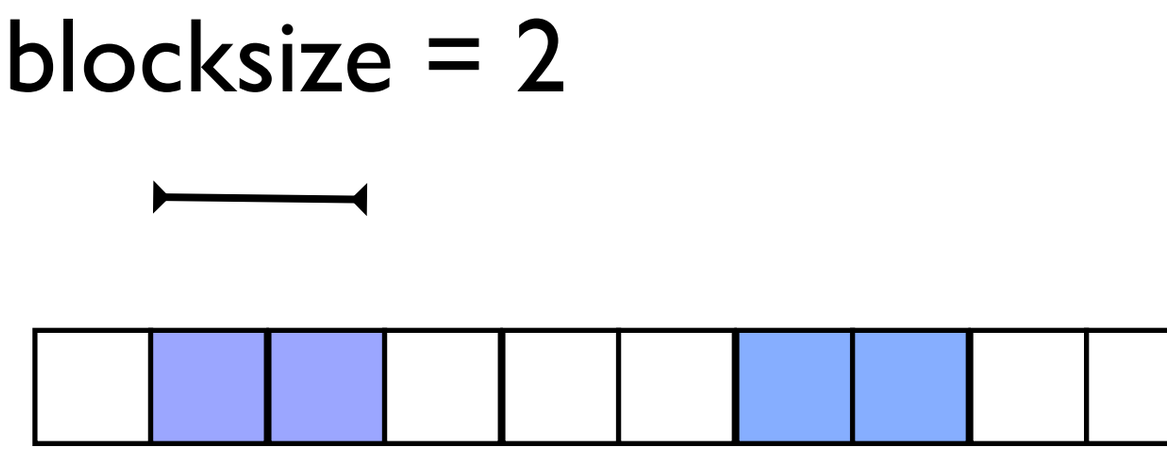
HDF5 Hyperslabs

- Parallel HDF5 similar to parallel NetCDF - fairly modest changes to structure of code
- Different (more low-level, natch) way of dealing with sub-regions
- Offset, block, count, stride



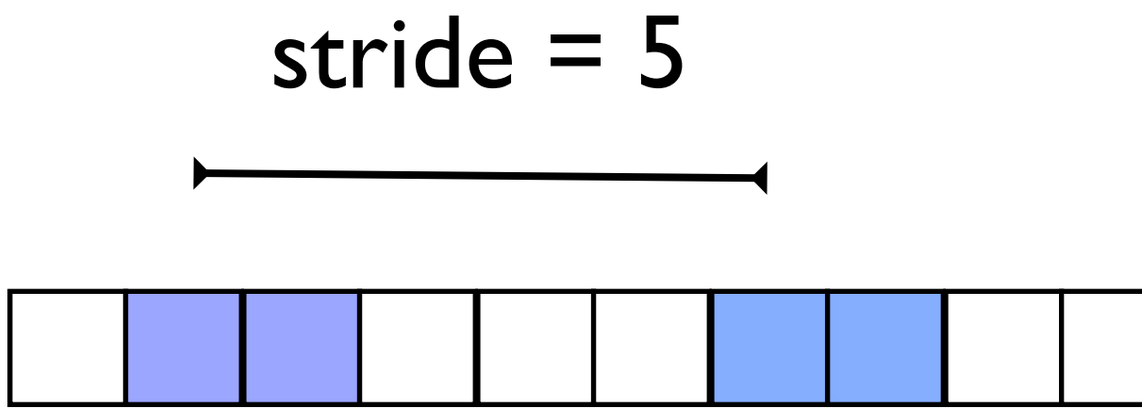
Offset = 1

HDF5 Hyperslabs



- Parallel HDF5 similar to parallel NetCDF - fairly modest changes to structure of code
- Different (more low-level, natch) way of dealing with sub-regions
- Offset, block, count, stride

HDF5 Hyperslabs

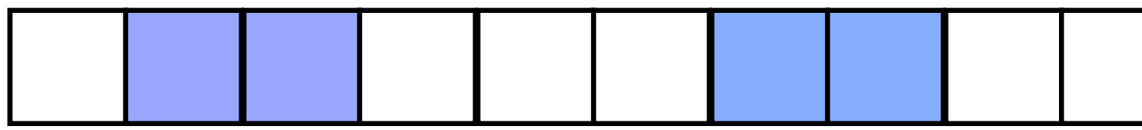


- Parallel HDF5 similar to parallel NetCDF - fairly modest changes to structure of code
- Different (more low-level, natch) way of dealing with sub-regions
- Offset, block, count, stride

HDF5 Hyperslabs

- Parallel HDF5 similar to parallel NetCDF - fairly modest changes to structure of code
- Different (more low-level, natch) way of dealing with sub-regions
- Offset, block, count, stride
- (MPI_Type_vector)

count = 2



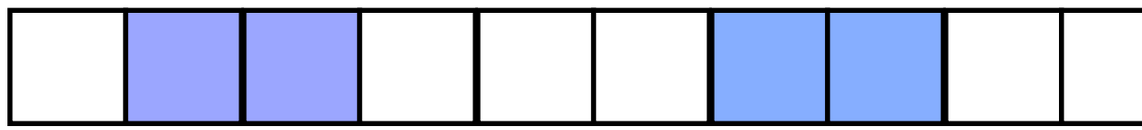
SciNet

compute • calcul CANADA

HDF5 Hyperslabs

- Parallel HDF5 similar to parallel NetCDF - fairly modest changes to structure of code
- Different (more low-level, natch) way of dealing with sub-regions
- Offset, block, count, stride
- Hyperslab - one of these per dimensions.
- (offset,block) just like (start, counts) in netcdf.

count = 2



SciNet

compute • calcul CANADA

Adaptable IO System

- ADIOS
- A library for IO for scientific code
- Uses MPIIO, HDF5, etc... under the hood
- Allows changing of IO strategy, method; no rewriting code and maybe not even a recompile.

SciNet

compute • calcul CANADA

Adaptable IO System

- Key insights:
- Grunt work of defining processor-specific regions of files can be abstracted away from the underlying libraries used
- Much of the library-call writing can be automated

SciNet

compute • calcul CANADA

sine-hdf5.c

```
void writehdf5file(rundata_t rundata, float *x, double *data) {
    /* identifiers */
    hid_t file_id, arr_group_id, data_dataset_id, x_dataset_id;
    hid_t data_dataspace_id, x_dataspace_id;
    hid_t loc_data_dataspace_id, loc_x_dataspace_id;
    hid_t globaldataspace,globalxspace;
    hid_t dist_id;
    hid_t fap_id;

    hsize_t datadims[1], xdims[1];
    hsize_t locdatadims[1], locxdims[1];

    MPI_Info info;

    hsize_t counts[1];
    hsize_t strides[1];
    hsize_t offsets[1];
    hsize_t blocks[1];

    MPI_Info_create(&info);
    MPI_Info_set(info,"IBM_largeblock_io","true");

    fap_id = H5Pcreate(H5P_FILE_ACCESS);
    H5Pset_fapl_mpio(fap_id, MPI_COMM_WORLD, info);

    dist_id = H5Pcreate(H5P_DATASET_XFER);
    H5Pset_dxpl_mpio(dist_id, H5FD_MPIO_COLLECTIVE);
    file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT, fap_id);
     if (file_id < 0) {
        fprintf(stderr,"Could not open file %s\n", rundata.filename);
        return;
    }

    arr_group_id = H5Gcreate(file_id,"/ArrayData", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

    datadims[0] = rundata.globaln;
    xdims[0]    = rundata.globaln;

    data_dataspace_id = H5Screate_simple(1, datadims, NULL);
    x_dataspace_id  = H5Screate_simple(1, xdims,  NULL);

    data_dataset_id = H5Dcreate(file_id, "/ArrayData/Data", H5T_IEEE_F64LE,
                                data_dataspace_id, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
    x_dataset_id  = H5Dcreate(file_id, "/ArrayData/x Coordinates",  H5T_IEEE_F32LE,
                                x_dataspace_id,  H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

    locdatadims[0] = rundata.localn;
    locxdims[0] =    rundata.localn;

    loc_data_dataspace_id = H5Screate_simple(1, locdatadims, NULL);
    loc_x_dataspace_id    = H5Screate_simple(1, locxdims,  NULL);

    offsets[0] = rundata.start;
    counts[0]  = rundata.localn;
    blocks[0]  = 1;
    strides[0] = 1;

    globaldataspace = H5Dget_space(data_dataset_id);
    H5Sselect_hyperslab(globaldataspace, H5S_SELECT_SET, offsets, strides, counts, blocks);

    globalxspace = H5Dget_space(x_dataset_id);
    H5Sselect_hyperslab(globalxspace, H5S_SELECT_SET, offsets, strides, counts, blocks);

    H5Dwrite(data_dataset_id, H5T_NATIVE_DOUBLE, loc_data_dataspace_id, globaldataspace, dist_id, data);
    H5Dwrite(x_dataset_id,    H5T_NATIVE_FLOAT,  loc_x_dataspace_id,    globalxspace,    dist_id, x);
```

```
{
      hid_t other_group_id;
      hid_t timestep_id, timestep_space;
      hid_t comptime_id, comptime_space;
      int timestep=13;
      float comptime=81.773;

      /* create group */
      other_group_id = H5Gcreate(file_id,"/OtherStuff", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

      /* scalar space, data for integer timestep */
      timestep_space = H5Screate(H5S_SCALAR);
      timestep_id = H5Dcreate(other_group_id, "Timestep", H5T_STD_U32LE,
                              timestep_space, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
      H5Dwrite(timestep_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT, &timestep);
      H5Dclose(timestep_id);
      H5Sclose(timestep_space);

      /* scalar space, data for floating compute time */
      comptime_space = H5Screate(H5S_SCALAR);
      comptime_id = H5Dcreate(other_group_id, "Compute Time", H5T_IEEE_F32LE,
                              comptime_space, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
      H5Dwrite(comptime_id, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, &comptime);
      H5Dclose(comptime_id);
      H5Sclose(comptime_space);

      H5Gclose(other_group_id);
    }


    /* End access to groups & data sets and release resources used by them */
    H5Sclose(data_dataspace_id);
    H5Dclose(data_dataset_id);
    H5Sclose(x_dataspace_id);
    H5Dclose(x_dataset_id);
    H5Gclose(arr_group_id);
    H5Pclose(fap_id);
    H5Pclose(dist_id);

    /* Close the file */
    H5Fclose(file_id);
    return;
}
```

SciNet

compute • calcul CANADA

sine-adios.c

```
void writeadiosfile(rundata_t rundata, float *x, double *data) {
    int         adios_err=0;
    uint64_t    adios_groupsize, adios_totalsize;
    int64_t     adios_handle;
    MPI_Comm    comm = MPI_COMM_WORLD;

    adios_init ("1d.xml");
    adios_open (&adios_handle, "ArrayData", rundata.filename, "w", &comm);
    #include "gwrite_ArrayData.ch"

    if (adios_err)
        fprintf(stderr,"Error doing adios write.\n");

    adios_close (adios_handle);
}
```

gwrite_ArrayData.ch

```
adios_groupsize = 4 \
                + 4 \
                + 4 \
                + 4 * (rundata.localn) \
                + 8 * (rundata.localn);
adios_group_size (adios_handle, adios_groupsize, &adios_totalsize);
adios_write (adios_handle, "rundata.localn", &rundata.localn);
adios_write (adios_handle, "rundata.globaln", &rundata.globaln);
adios_write (adios_handle, "rundata.start", &rundata.start);
adios_write (adios_handle, "x Coordinates", x);
adios_write (adios_handle, "Data", data);
```

1d.xml

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communicator="comm">
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offsets="rundata.start">
           <var name="x Coordinates" gwrite="x"    type="float"
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type="double"
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

SciNet

compute • calcul CANADA

ADIOS workflow

- Write XML file describing data, layout
- gpp.py [file].xml - generates C or Fortran code: adios calls, size calculation
- Build code
- Separates data layout, code.

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communica
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offs
           <var name="x Coordinates" gwrite="x"    type=
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type=
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

```
void writeadiosfile(rundata_t rundata, float *x, double
    int         adios_err=0;
    uint64_t    adios_groupsize, adios_totalsize;
    int64_t     adios_handle;
    MPI_Comm    comm = MPI_COMM_WORLD;

    adios_init ("1d.xml");
    adios_open (&adios_handle, "ArrayData", rundata.file
    #include "gwrite_ArrayData.ch"

    if (adios_err)
        fprintf(stderr,"Error doing adios write.\n");

    adios_close (adios_handle);
}
```

SciNet

compute • calcul CANADA

ADIOS workflow

- Separation isn't perfect; xml file references code variables, etc.
- But allows "componentization" of I/O.
- Changes that don't result in changes to gwrite_Array.ch don't require recompilation (eg, only changing number, size of variables in group).

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communica
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offs
           <var name="x Coordinates" gwrite="x"    type=
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type=
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

```
void writeadiosfile(rundata_t rundata, float *x, double
    int         adios_err=0;
    uint64_t    adios_groupsize, adios_totalsize;
    int64_t     adios_handle;
    MPI_Comm    comm = MPI_COMM_WORLD;

    adios_init ("1d.xml");
    adios_open (&adios_handle, "ArrayData", rundata.file
    #include "gwrite_ArrayData.ch"

    if (adios_err)
        fprintf(stderr,"Error doing adios write.\n");

    adios_close (adios_handle);
}
```

SciNet

compute • calcul CANADA

Variable Groups

- Multiple groups of variables possible: (eg) restart files vs. files for analysis
- Variables can appear in mutiple groups
- Each group can be handled with different methods

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communica
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offs
           <var name="x Coordinates" gwrite="x"    type=
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type=
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

```
void writeadiosfile(rundata_t rundata, float *x, double
    int         adios_err=0;
    uint64_t    adios_groupsize, adios_totalsize;
    int64_t     adios_handle;
    MPI_Comm    comm = MPI_COMM_WORLD;

    adios_init ("1d.xml");
    adios_open (&adios_handle, "ArrayData", rundata.file
    #include "gwrite_ArrayData.ch"

    if (adios_err)
        fprintf(stderr,"Error doing adios write.\n");

    adios_close (adios_handle);
}
```

SciNet

compute • calcul CANADA

I/O methods

- Possible methods: parallel HDF5 (PHDF5), one-per-process posix files (POSIX), raw MPI-IO
- Change between methods: edit xml file, that's it.
- P-1 (PHDF5, NC4,MPI), P-P (POSIX), or even P-M possible (PHDF5, etc with multiple communicators)

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communica
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offs
           <var name="x Coordinates" gwrite="x"    type=
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type=
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

```
void writeadiosfile(rundata_t rundata, float *x, double
    int         adios_err=0;
    uint64_t    adios_groupsize, adios_totalsize;
    int64_t     adios_handle;
    MPI_Comm    comm = MPI_COMM_WORLD;

    adios_init ("1d.xml");
    adios_open (&adios_handle, "ArrayData", rundata.file
    #include "gwrite_ArrayData.ch"

    if (adios_err)
        fprintf(stderr,"Error doing adios write.\n");

    adios_close (adios_handle);
}
```

SciNet

compute • calcul CANADA

Simplifies IO code

- Even if you aren't planning to switch between IO strategies, can greatly simplify code
- Many mechanical steps (eg, pasting together rectangular multi-dimentional arrays) done for you.
- Eliminates tedious, error-prone boilerplate code

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communica
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offs
           <var name="x Coordinates" gwrite="x"    type=
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type=
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

```
void writeadiosfile(rundata_t rundata, float *x, double
    int         adios_err=0;
    uint64_t    adios_groupsize, adios_totalsize;
    int64_t     adios_handle;
    MPI_Comm    comm = MPI_COMM_WORLD;

    adios_init ("1d.xml");
    adios_open (&adios_handle, "ArrayData", rundata.file
    #include "gwrite_ArrayData.ch"

    if (adios_err)
        fprintf(stderr,"Error doing adios write.\n");

    adios_close (adios_handle);
}
```

SciNet

compute • calcul CANADA

Data Formats

- When using a method like HDF5, NCDF4, uses underlying file formats
- When using a method like MPIIO/POSIX, with no fixed file format, uses its own "binary packed" format
- (Like HDF5, but built for HPC - stripped down, fast)

Utilities:

bpls - list file
bp2h5 - convert to HDF5
bp2ncd - convert to NetCDF
bpdump - dump output
bp2ascii - print output

Other Methods

- Other output method:
- MPICIO: collective output (default is independant)
- MPIAIO: asynchronous output (I/O while computation) - you put in "hints" about when good IO times would be
- Other non-MPI-IO based transports (DataTap, georgia tech)
- Allows you to play with different IO technologies without rewriting your code

P-1, P-P, P-M

- MPIIO, PHDF5, etc: P processes write to 1 file
- POSIX: P processes write to P files in a directory named "filename.dir", with header file "filename" describing what's where
- Parallel transports allow P-M.
- Break MPI_COMM_WORLD up into M communicators in I/O routine (1-3 lines of code)
- each *communicator* outputs its own file, like POSIX
- Completely change I/O pattern with couple lines of code!

1d.xml

```
<?xml version="1.0"?>
<adios-config host-language="C">

    <adios-group name="ArrayData" coordination-communicator="comm">
        <var name="rundata.localn"  type="integer" />
        <var name="rundata.globaln" type="integer" />
        <var name="rundata.start"   type="integer" />
        <global-bounds dimensions="rundata.globaln" offsets="rundata.start">
           <var name="x Coordinates" gwrite="x"    type="float"
                dimensions="rundata.localn" />
           <var name="Data"          gwrite="data" type="double"
                dimensions="rundata.localn" />
        </global-bounds>
    </adios-group>

<method group="ArrayData" method="MPI" />

<buffer size-MB="2" allocate-time="now"/>

</adios-config>
```

SciNet

compute • calcul CANADA

Using ADIOS

- module load gcc openmpi hdf5/(parallel using openmpi) netcdf/(parallel using openmpi) adios
- Write xml file
- adios_lint [xml file]
- write very simple IO routine
- Get started!
- /scinet/gpc/src/adios/adios-1.3.1/examples
- Source code, slides for this talk on wiki
- http://www.olcf.ornl.gov/center-projects/adios/

SciNet

compute • calcul CANADA