Parallel I/O with NetCDF and HDF5

Ramses van Zon

SciNet HPC Consortium

September 21, 2015

What are NetCDF and HDF5?

Both are standardized file formats for scientific data, which are:

- Self-describing;
- Binary format;
- Many tools use these formats;
- Parallel access (NetCDF4 and HDF5).
- Format the same whether working in serial or in parallel.

SciNet compute • calcul CANADA

How do NetCDF and HDF5 differ?

- NetCDF is aimed at storing large multi-dimensional arrays, but simpler to use.
- HDF5 hold more general data, but is more complex to use;

SciNet compute • calcul CANADA

NetCDF

- Let's consider NetCDF first.
- A format as well as an Applications Program interface (API).
- MPI-IO was too low-level.
- Even for the vary common scientific use-cases.
- NetCDF gives you a higher level approach to writing and reading multi-dimensional arrays.

SciNet compute • calcul CANADA

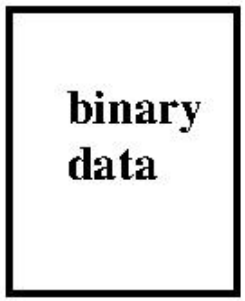
Intro to the NetCDF Format

```
$ cd pario/netcdf
$ source setup
$ make 2darray-simple # or f2darray-simple
$ ./2darray-simple # or ./f2darray-simple
$ ncdump -h file.nc?
```

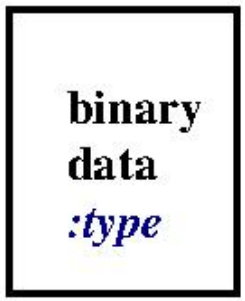
SciNet compute • calcul CANADA

NetCDF Data Model

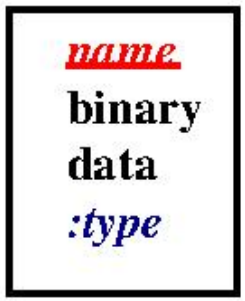
NetCDF Data Model



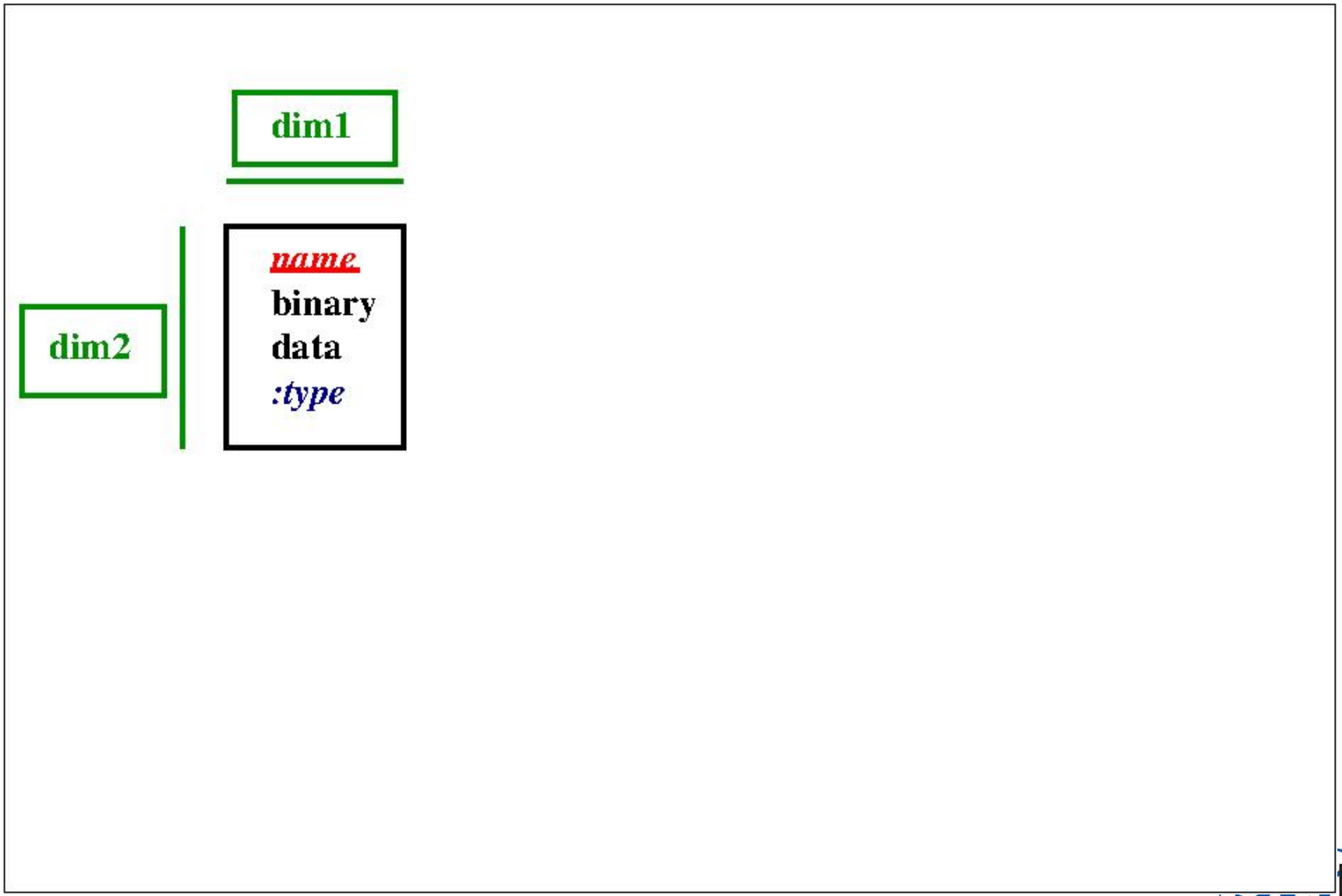
NetCDF Data Model



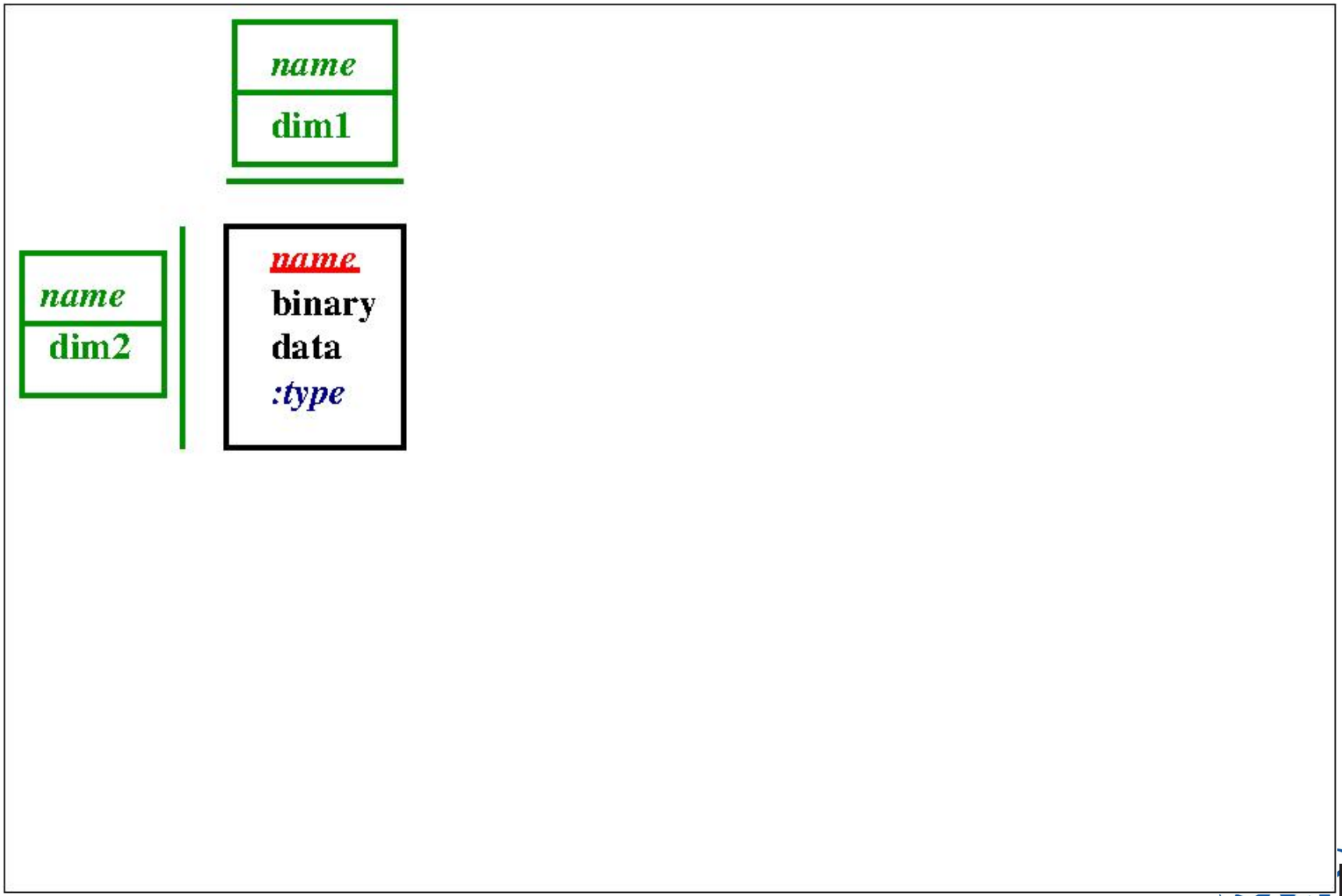
NetCDF Data Model



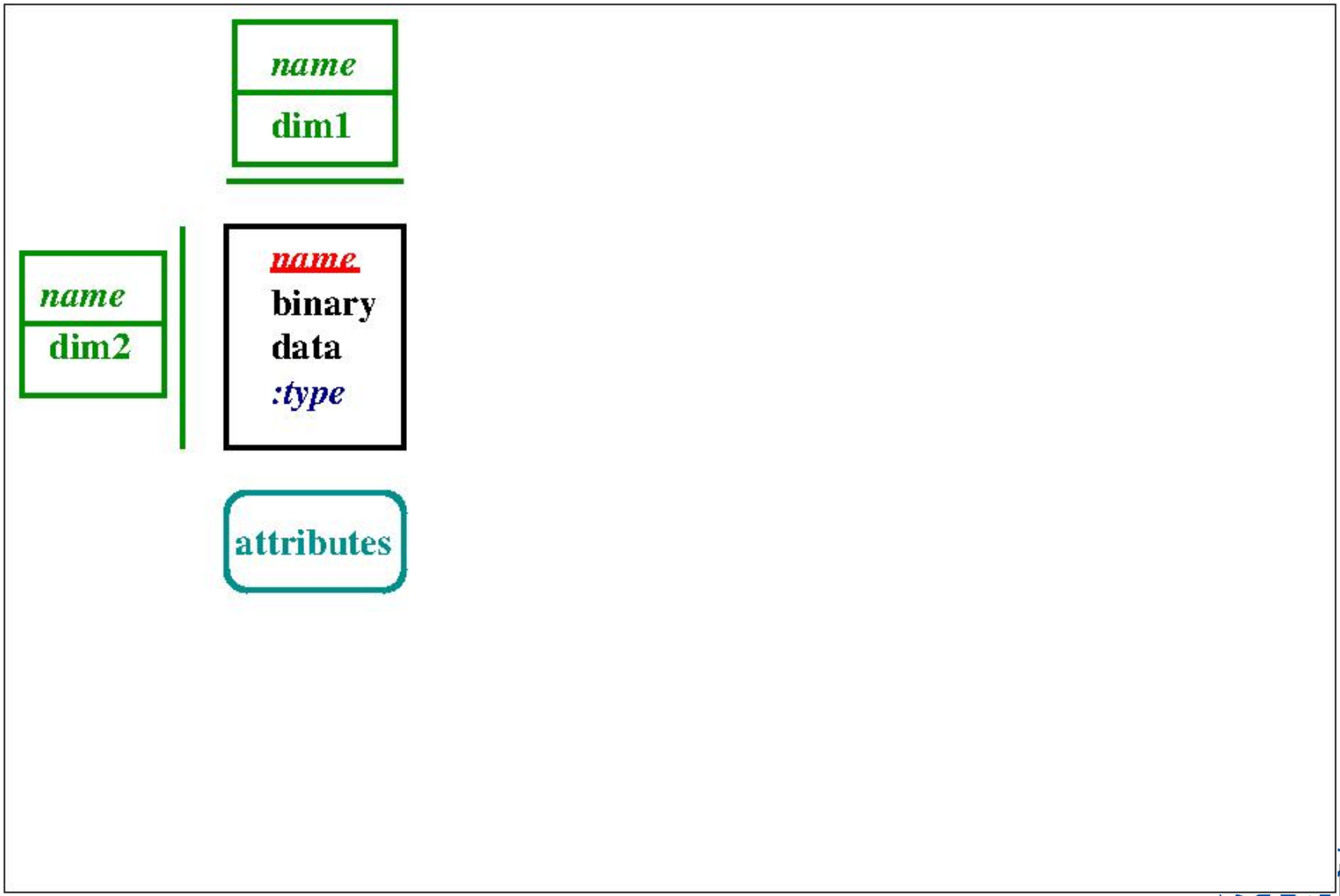
NetCDF Data Model



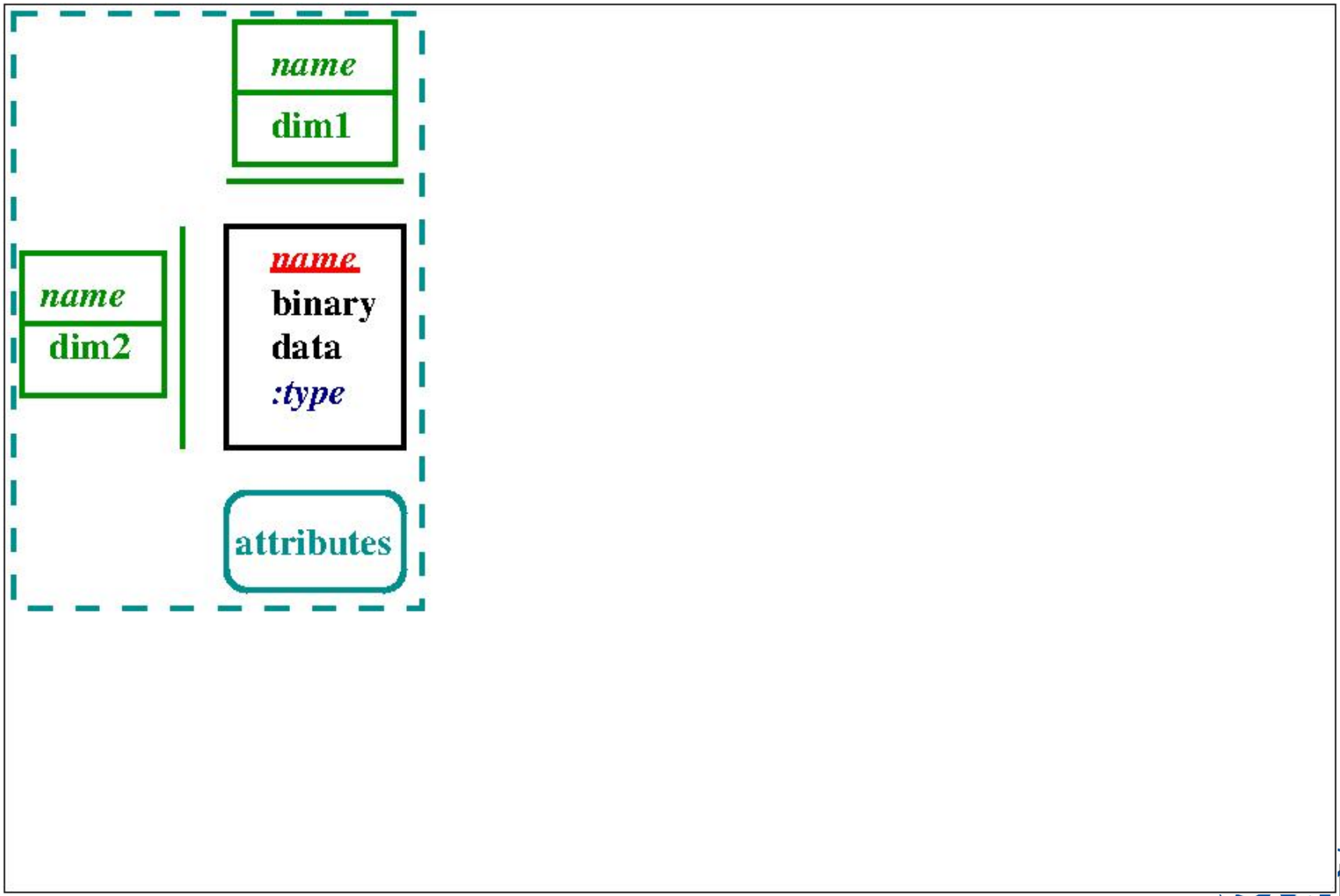
NetCDF Data Model



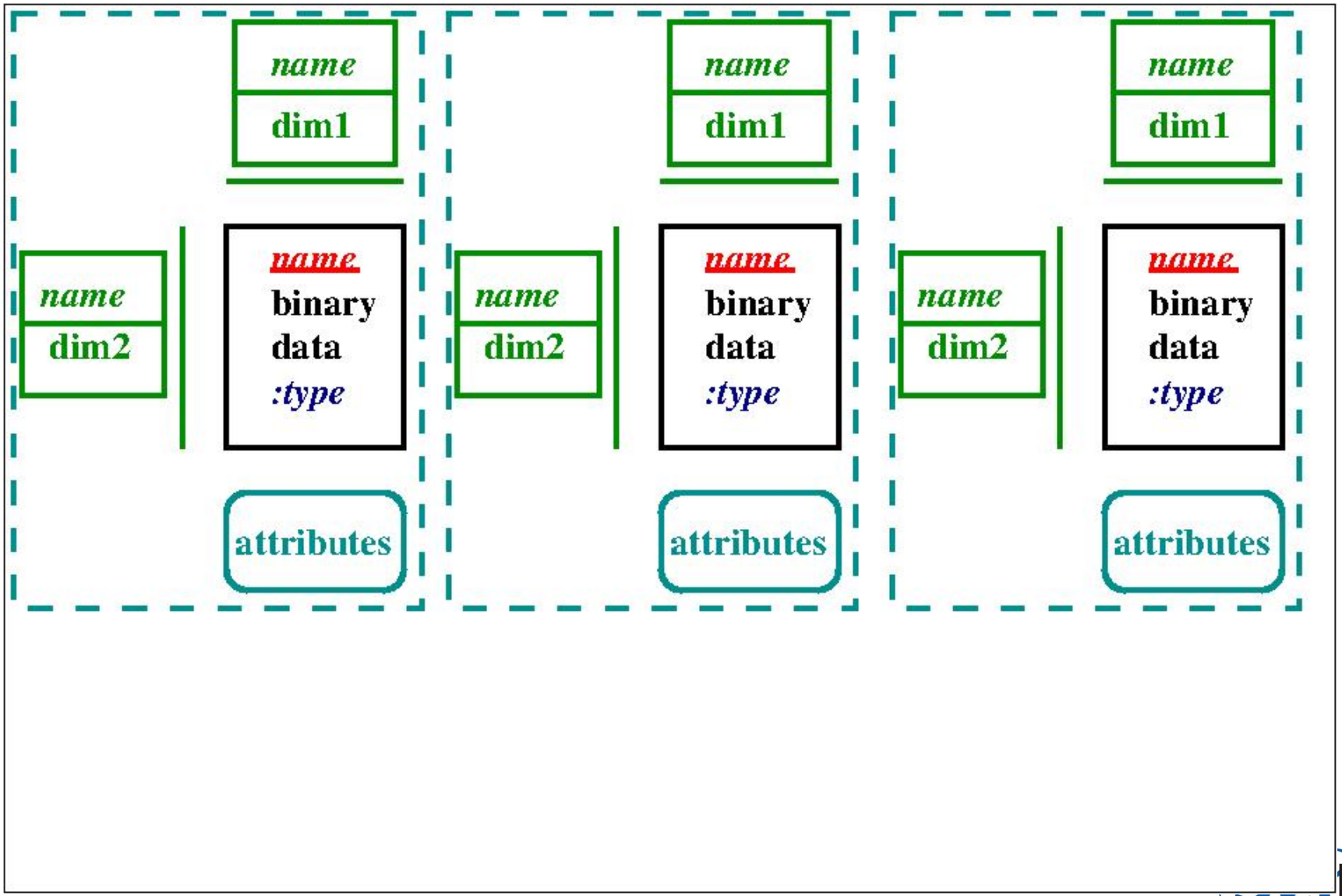
NetCDF Data Model



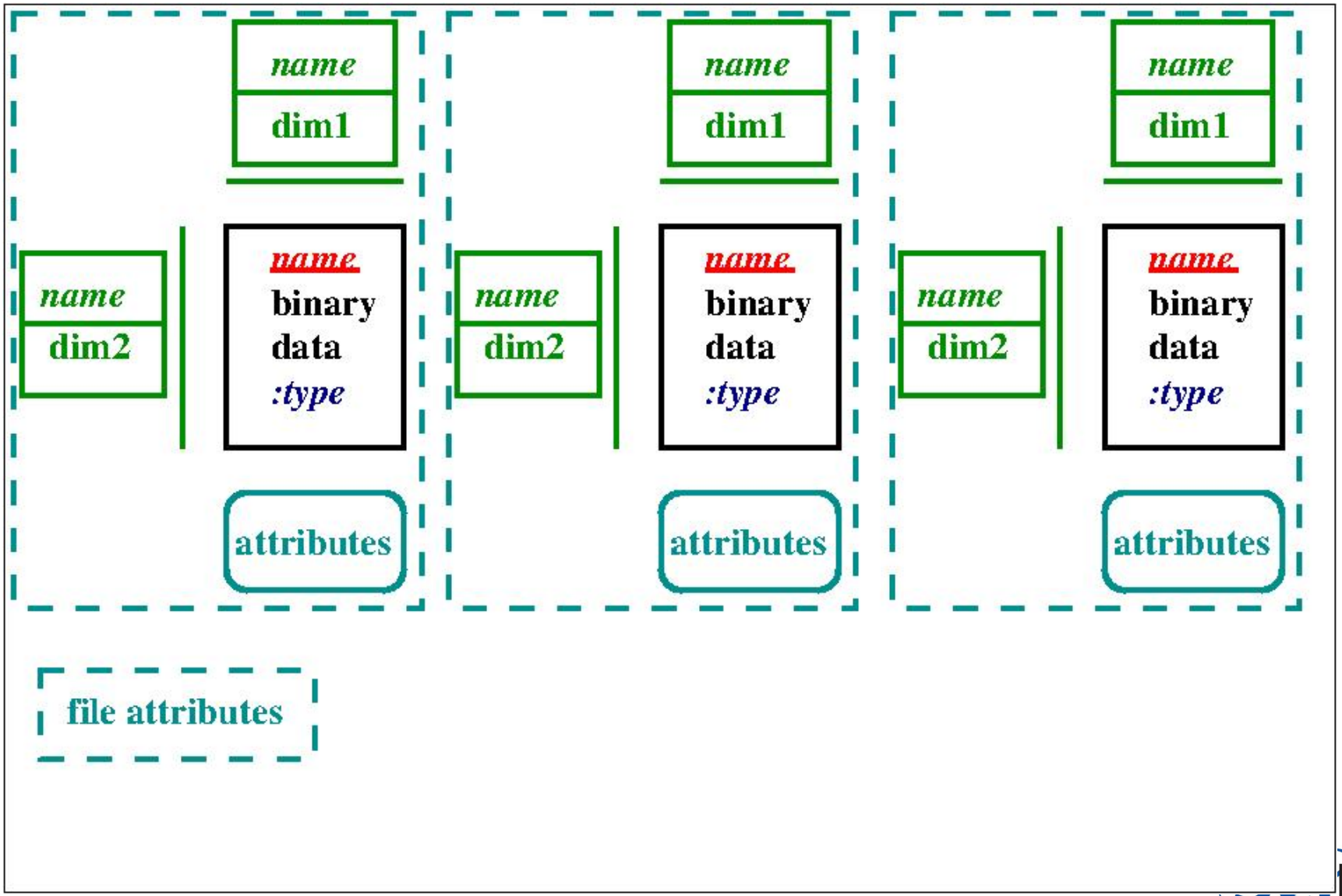
NetCDF Data Model



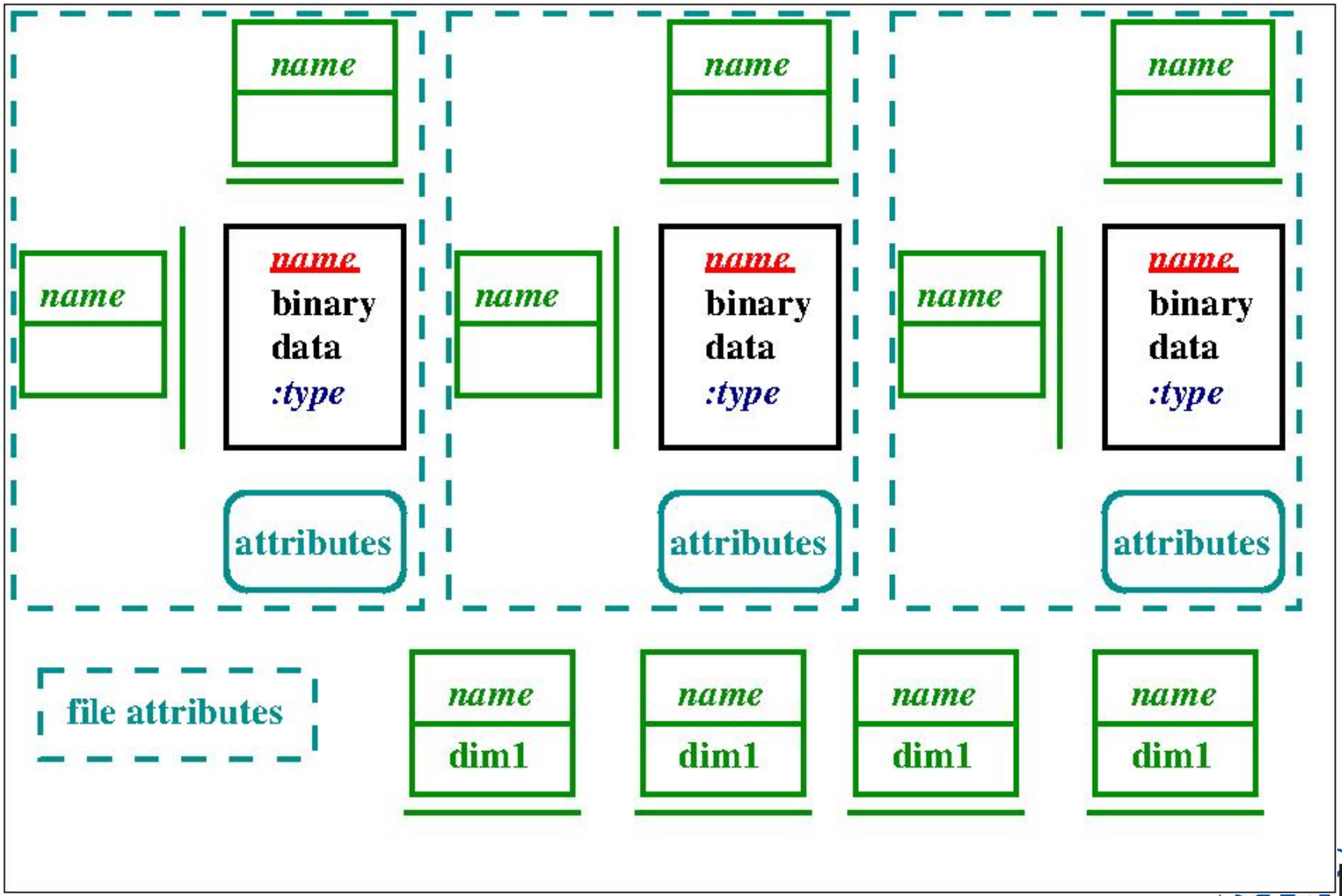
NetCDF Data Model



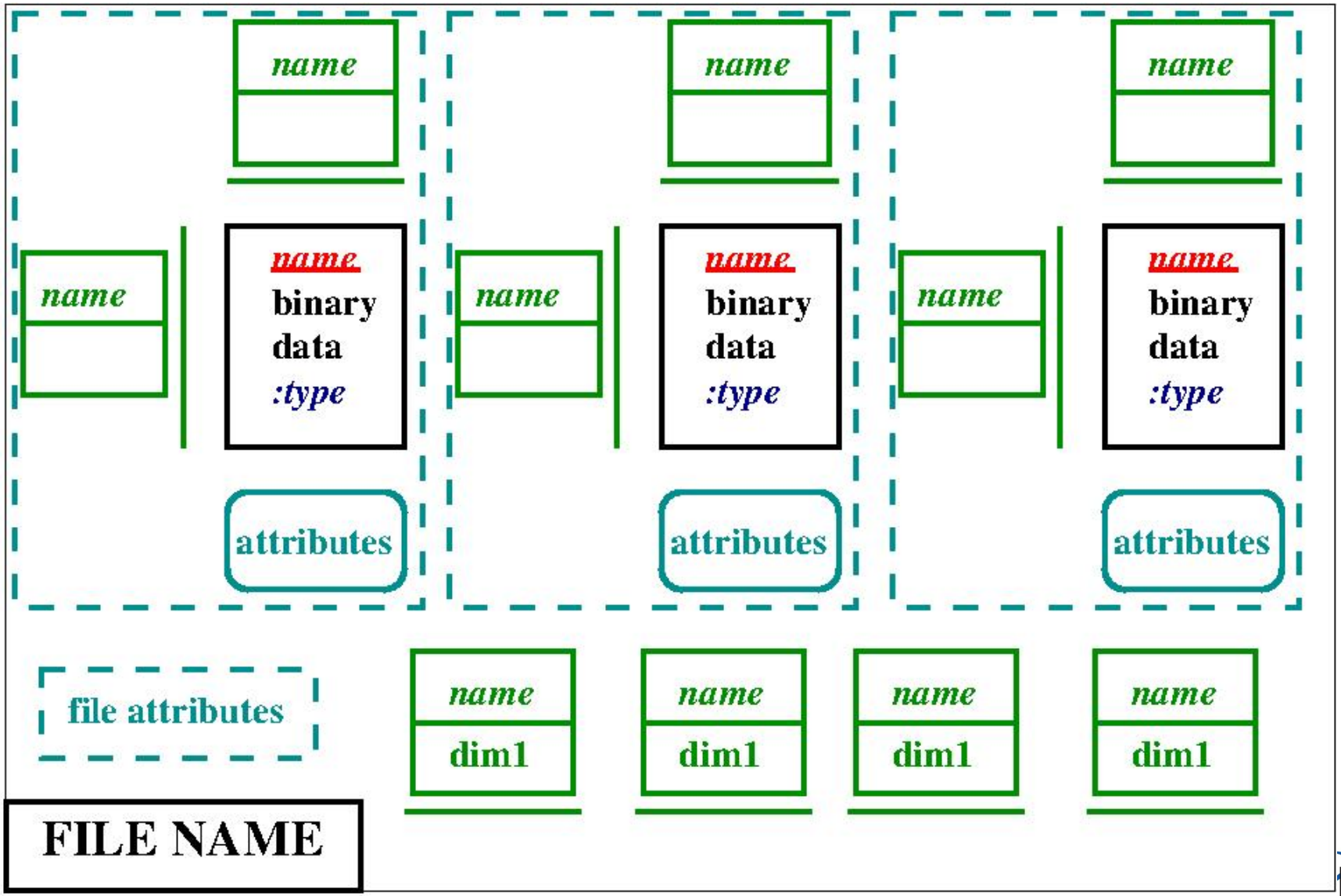
NetCDF Data Model



NetCDF Data Model



NetCDF Data Model



NetCDF API

What's an API, anyways?

- Defines function prototypes:
  - What's their input?
  - What's their output?
  - What are they supposed to do?
- Constants:
  - Encode options
  - Encode errors
- Types
  - NetCDF just uses `int` for c types.
  - The types of the data encoded in integer options.

NetCDF's main APIs are for Fortran and C.

There are interfaces for C++, python, R, ruby, ...

Sample Code

```
$ cp -r /scinet/course/pario15 .
$ cd pario15/netcdf
$ source setup
$ make netcdftest
$ ./netcdftest
...
```

Sample Code

```
$ cp -r /scinet/course/pario15 .
$ cd pario15/netcdf
$ source setup
$ make netcdftest
$ ./netcdftest
...
```

```
$ ncdump -h test.nc
netcdf netcdftest {
dimensions:
        X = 48 ;
        Y = 48 ;
variables:
        int M(X,Y) ;
}
```

Sample Code

```
$ cp -r /scinet/course/pario15 .
$ cd pario15/netcdf
$ source setup
$ make netcdftest
$ ./netcdftest
...
```

```
$ ./ncview test.nc
```

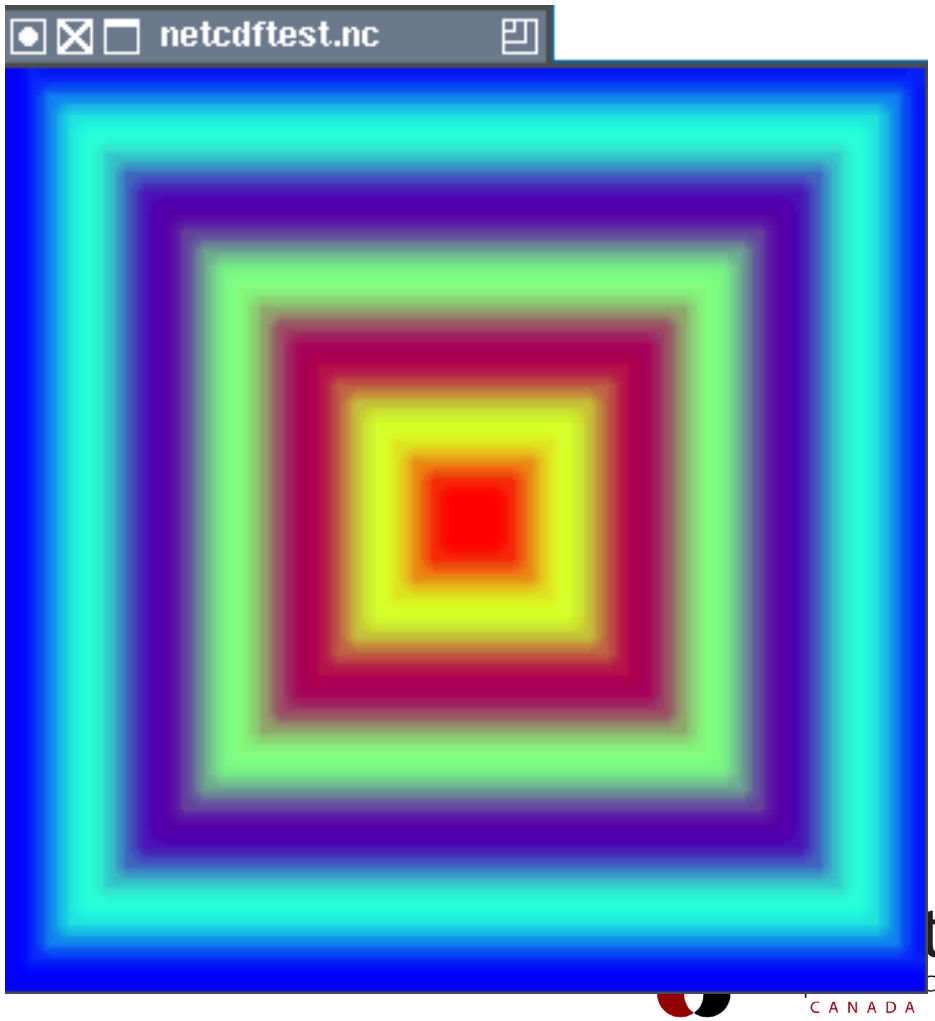
```
$ ncdump -h test.nc
netcdf netcdftest {
dimensions:
        X = 48 ;
        Y = 48 ;
variables:
        int M(X,Y) ;
}
```

Sample Code

```
$ cp -r /scinet/course/pario15 .
$ cd pario15/netcdf
$ source setup
$ make netcdftest
$ ./netcdftest
...
```

```
$ ncdump -h test.nc
netcdf netcdftest {
dimensions:
        X = 48 ;
        Y = 48 ;
variables:
        int M(X,Y) ;
}
```

```
$ ./ncview test.nc
```



netcdftest.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <netcdf.h>
#define MIN(x,y) ((x)<(y)?(x):(y))
int main(void) {
 const int N = 48;
 int ncid, varid, status, dimid[2], *data;
 printf("Testing i/o in netcdf4\n");
 data = malloc(sizeof(int)*N*N);
 for (int i = 0; i < N*N; i++)
    data[i] = MIN(N/2-abs((i%N)-N/2), N/2-abs((i/N)-N/2));
 status = nc_create("test.nc", NC_CLOBBER|NC_NETCDF4, &ncid);
 status = nc_def_dim(ncid, "X", N, &dimid[0]);
 status = nc_def_dim(ncid, "Y", N, &dimid[1]);
 status = nc_def_var(ncid, "M", NC_INT, 2, dimid, &varid);
 status = nc_enddef(ncid);
 status = nc_put_var_int(ncid, varid, data);
 status = nc_close(ncid);
 free(data);
 printf("Done.\n"); }
```

fnetcdftest.f90

```
program fnetcdftest
 use netcdf
 integer, parameter :: N=48
 integer :: i, j, ncid, varid, status, dimidx, dimidy
 integer, dimension(:,:), allocatable :: data
 print *, "Testing i/o in netcdf4"
 allocate(data(N,N));
 do i=1,N; do j=1,N
   data(i,j) = min(N/2-abs(i-N/2), N/2-abs(j-N/2))
 enddo; enddo
 status = nf90_create("test.nc",IOR(NF90_NETCDF4,NF90_CLOBBER),ncid)
 status = nf90_def_dim(ncid,"X",N,dimidx)
 status = nf90_def_dim(ncid,"Y",N,dimidy)
 status = nf90_def_var(ncid,"M",NF90_INT,(/dimidx,dimidy/),varid)
 status = nf90_enddef(ncid)
 status = nf90_put_var(ncid, varid, data)
 status = nf90_close(ncid)
 deallocate(data)
 print *, "Done."
end program fnetcdftest
```

Writing a NetCDF File

To write a NetCDF file, we go through the following steps:

- Create the file
- Define dimensions
- Define variables
- End definitions
- Write variables
- Close file

SciNet compute • calcul CANADA

Reading a NetCDF File

To read in (part of) a NetCDF file, we go through the following steps:

- Open the file
- Get dimension ids
- Get dimension lengths
- Get variable ids
- Read variables
- Close file

Example Reading NetCDF File (C)

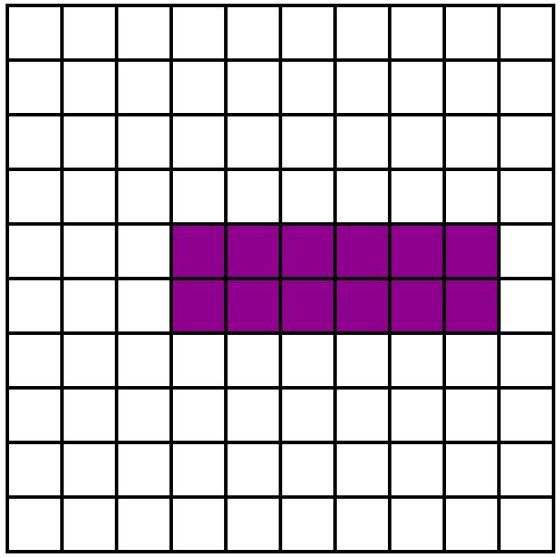
```c
#include "netcdf.h"
#define MAX(x,y) ((x)>(y)?(x):(y))
int main(void){
  int fileid, varid, status, dimid[2], maximum=0, *data;
  size_t nx, ny;
  char name[256];
  printf("Testing read in of a netcdf4 file\n");
  status = nc_open("test.nc", NC_NOWRITE, &fileid);
  status = nc_inq_dimid(fileid, "X", &dimid[0]);
  status = nc_inq_dimid(fileid, "Y", &dimid[1]);
  status = nc_inq_dim(fileid, dimid[0], name, &nx);
  status = nc_inq_dim(fileid, dimid[1], name, &ny);
  data = malloc(nx*ny*sizeof(int));
  status = nc_inq_varid(fileid, "M", &varid);
  status = nc_get_var(fileid, varid, data);
  status = nc_close(fileid);
  for (int i=0; i<nx*ny; i++)
     maximum = maximum<data[i]?data[i]:maximum;
  printf("Max. value = %d\n", maximum);
  free(data);  printf ("Done.\n");  }
```

Example Reading NetCDF File (Fortran)

```
program fnetcdfread
  use netcdf
  integer, parameter :: N=48
  integer :: maximum, fileid, varid, status, dimidx, dimidy, nx, ny
  integer, dimension(:,:), allocatable :: data
  print *, "Testing read in of netcdf4 file"
  status = nf90_open("test.nc", NF90_NOWRITE, fileid)
  status = nf90_inq_dimid(fileid, "X", dimidx)
  status = nf90_inq_dimid(fileid, "Y", dimidy)
  status = nf90_inquire_dimension(fileid, dimidx, len=nx)
  status = nf90_inquire_dimension(fileid, dimidy, len=ny)
  allocate(data(nx,ny));
  status = nf90_inq_varid(fileid, 'M', varid)
  status = nf90_get_var(fileid, varid, data)
  status = nf90_close(fileid)
  maximum = maxval(data)
  print *, "Max. value =", maximum
  deallocate(data)
  print *, "Done."
end program fnetcdfread
```

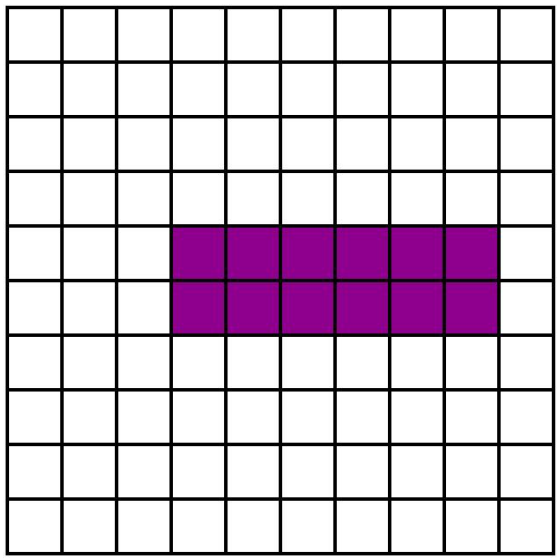
Accessing subregions in a file

- `nc_put_var_type` or `nf90_put_var` puts in the whole array.
- Subarrays can be specified with starts and counts.



Accessing subregions in a file

- `nc_put_var_type` or `nf90_put_var` puts in the whole array.
- Subarrays can be specified with starts and counts.



C

```
start[0] = 3;
start[1] = 4;
count[0] = 6;
count[1] = 2;
nc_put_vara_int(fileid,varid,
                start,count,data);
```

Fortran

```
start(1) = 4
start(2) = 5
count(1) = 6
count(2) = 2
nf90_put_var(fileid,varid,data,
       START=start,COUNT=count)
```

Further notes on (serial) NetCDF

- NetCDF adapts to the storage convention of C and Fortran as appropriate.
- Can have 'unlimited' size (e.g. can grow).
- Can add attributes: do this!

SciNet compute • calcul CANADA

Parallel I/O with NetCDF

- NetCDF4 builds on top of HFD5, which can use MPI-IO.
- NetCDF4's parallel IO uses the subregions construction.
- Must use in conjunction with MPI (no threaded parallel I/O).
- Can be as simple as changing the creation/opening of the file.
- And using the subregions in `put` and `get`.

Parallel I/O with NetCDF (C)

```
nc_create_par(filename,mode,MPI_COMM,MPI_INFO,fileid);
nc_open_par(filename,mode,MPI_COMM,MPI_INFO,fileid);
nc_var_par_access(fileid,varid,NC_COLLECTIVE);
nc_put_vara_type(...)
nc_get_vara_type(...)
```

Parallel I/O with NetCDF (Fortran)

```
nf90_create_par(filename,mode,MPI_COMM,MPI_INFO,fileid);
nf90_open_par(filename,mode,MPI_COMM,MPI_INFO,fileid);
nf90_var_par_access(fileid,varid,NC_COLLECTIVE);
nf90_put_var(...,START=,COUNT=)
nf90_get_var(...,START=,COUNT=)
```

SciNet compute • calcul CANADA

Parallel I/O Example with NetCDF (1)

```
#include <netcdf_par.h>
int main(int argc, char **argv) {
  const int N = 48;
  int size, rank, fileid, varid, dimid[2], *localdata;
  size_t start[2], count[2];
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank==0) printf("Testing parallel i/o in netcdf4\n");
  start[0] = (rank*N)/size;  count[0] = N/size;
  start[1] = 0;              count[1] = N;
  localdata = malloc(sizeof(int)*count[0]*count[1]);
  for (int i = 0; i < count[0]*count[1]; i++)
     localdata[i] = MIN(N/2-abs(start[1]+(i%count[1])-N/2),
                        N/2-abs(start[0]+(i/count[1])-N/2));
  nc_create_par("netcdfpartest.nc", NC_NETCDF4|NC_MPIIO,
     MPI_COMM_WORLD, MPI_INFO_NULL, &fileid);
  nc_def_dim(fileid, "X", N, &dimid[0]);
  nc_def_dim(fileid, "Y", N, &dimid[1]);
```

Parallel I/O Example with NetCDF (2)

```
  nc_def_var(fileid, "M", NC_INT, 2, dimid, &varid);
  nc_enddef(fileid);
  nc_var_par_access(fileid, varid, NC_COLLECTIVE);
  nc_put_vara_int(fileid, varid, start, count, localdata);
  nc_close(fileid);
  MPI_Finalize();
  free(localdata);
  if (rank==0) printf("Done.\n");
}
```

NetCDF References

- C Interface:
  http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c/
- Fortran Interface:
  http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90/

HDF5

- HDF=Hierarchical Data Format
- Can be seen as a generalization of NetCDF.
- HDF5 allows for parallel IO, using MPI-IO under the hood.

- But HDF5 allows for a lot more than NetCDF:
  - Object-oriented description of datasets, groups, attributes, types, data spaces and property lists.
  - File content can be arranges in a Unix-like file system
  - Groups can contains structures that hold data sets, other groups, etc.
  - Optional compression.
- But: have to do more work when reading and writing HDF5 files.

SciNet compute • calcul CANADA

Use other slide deck for rest of HDF5

Old slides still valid!

Conclusions

- Quite a few options for doing parallel I/O.
- MPI-IO underlies most of these.
- NetCDF and HDF5 allow you to store metadata, structure your data, and make it portable: recommended!
- Apart from performance gains, also makes your output independent of the number of preocesses that produces it.
- Pay attention to disk I/O! Bandwidth and IOPs are limited, and shared with other users.
- Still keep other common I/O best practices in mind: Fewer files, binary formats, store only what you need to keep, write in big chunks.

SciNet compute • calcul CANADA