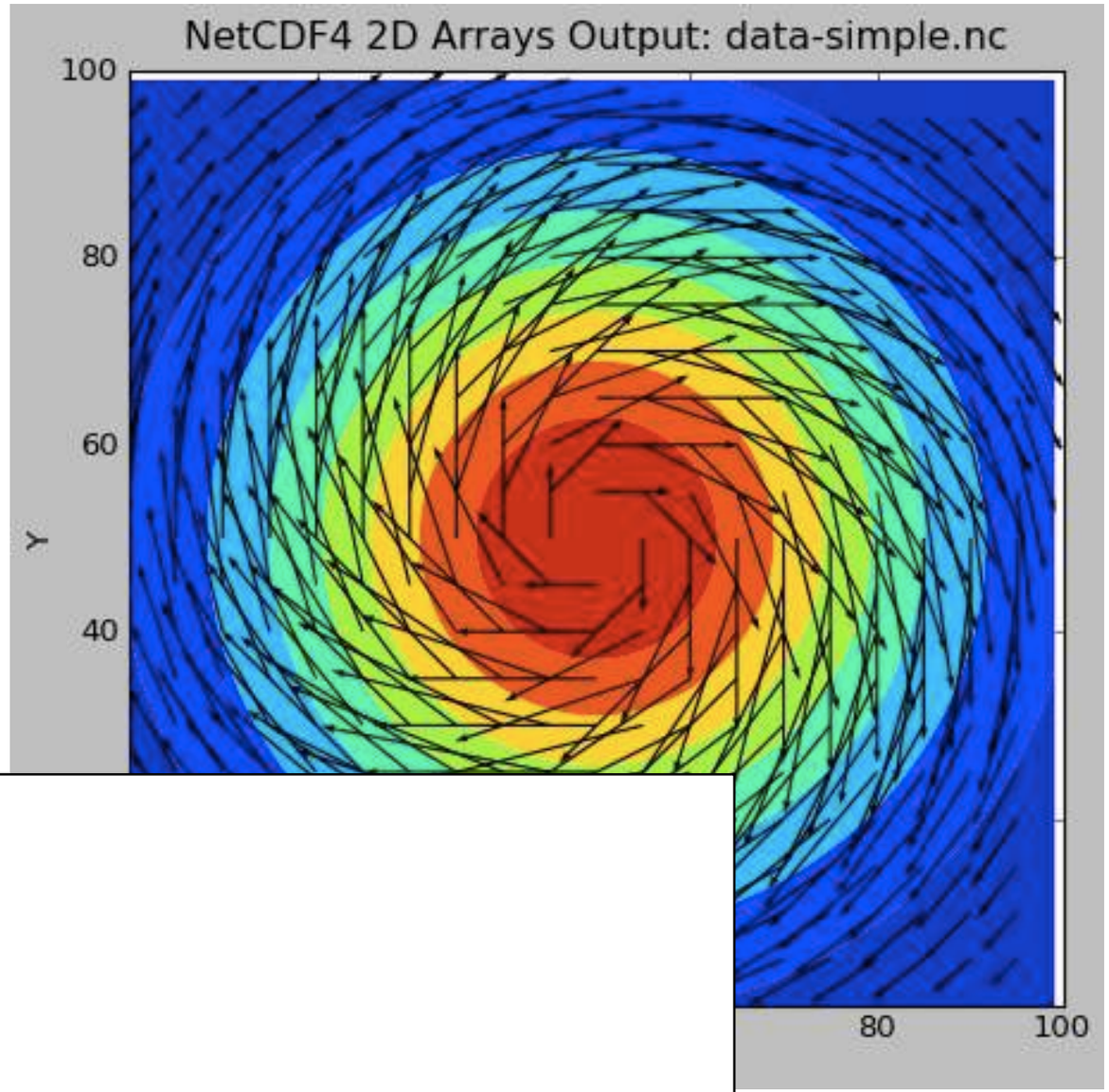
Sample Code

http://www.hdfgroup.org/HDF5/

```
$ cd parIO/hdf5

$ source ../parallellibs
$ make serial or
$ make 2darray (C), or
$ make f2darray (F90)

$ ./{f,}2darray
$ ls *.h5

$ ../plots.py *.h5
```

What is this .h5 file?

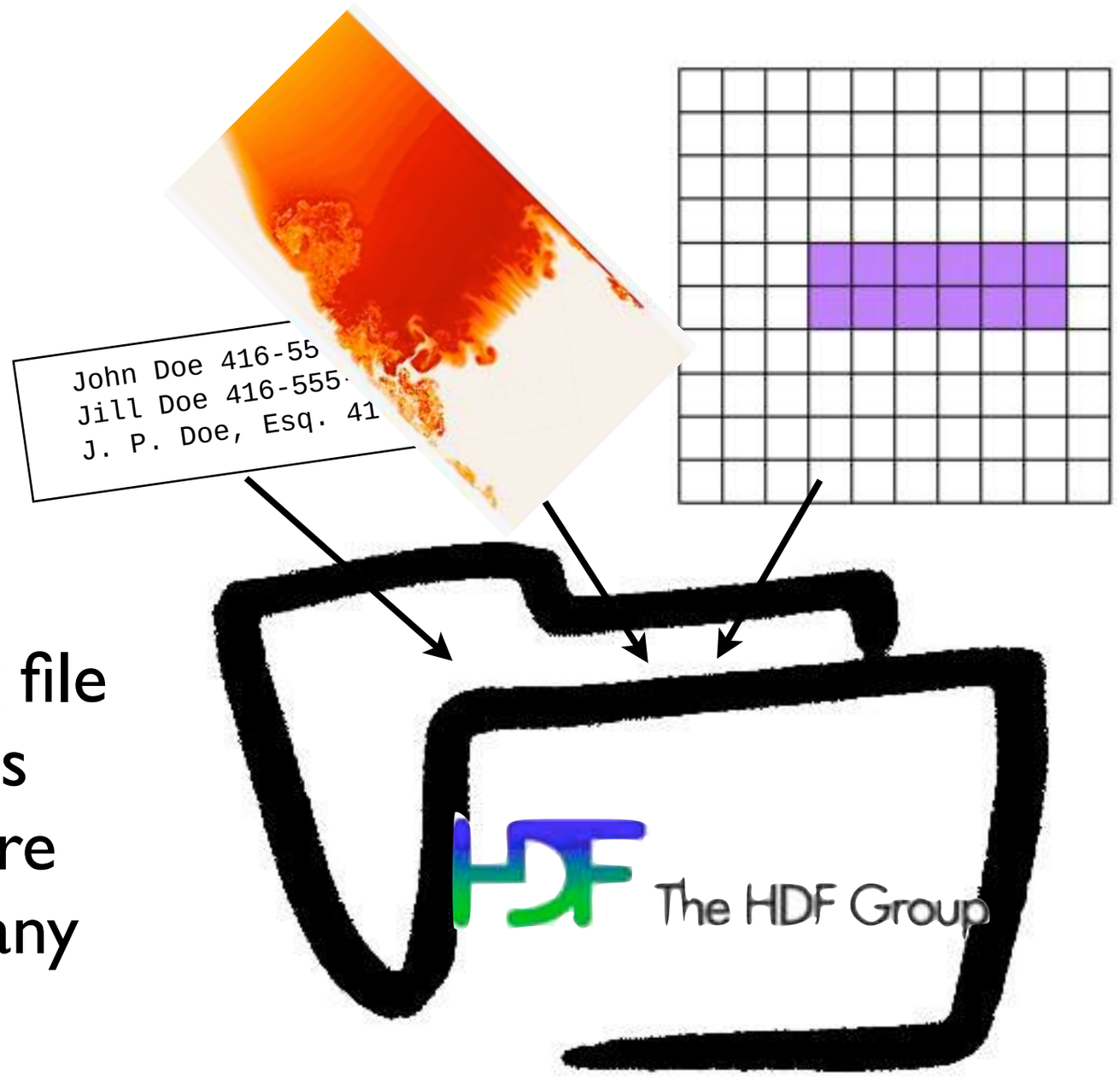



```
$ h5ls data-fort.h5
ArrayData                Group
OtherStuff               Group

$ h5ls data-fort.h5/ArrayData
dens                     Dataset {100, 100}
vel                      Dataset {100, 100, 2}
```

SciNet compute • calcul CANADA

HDF5

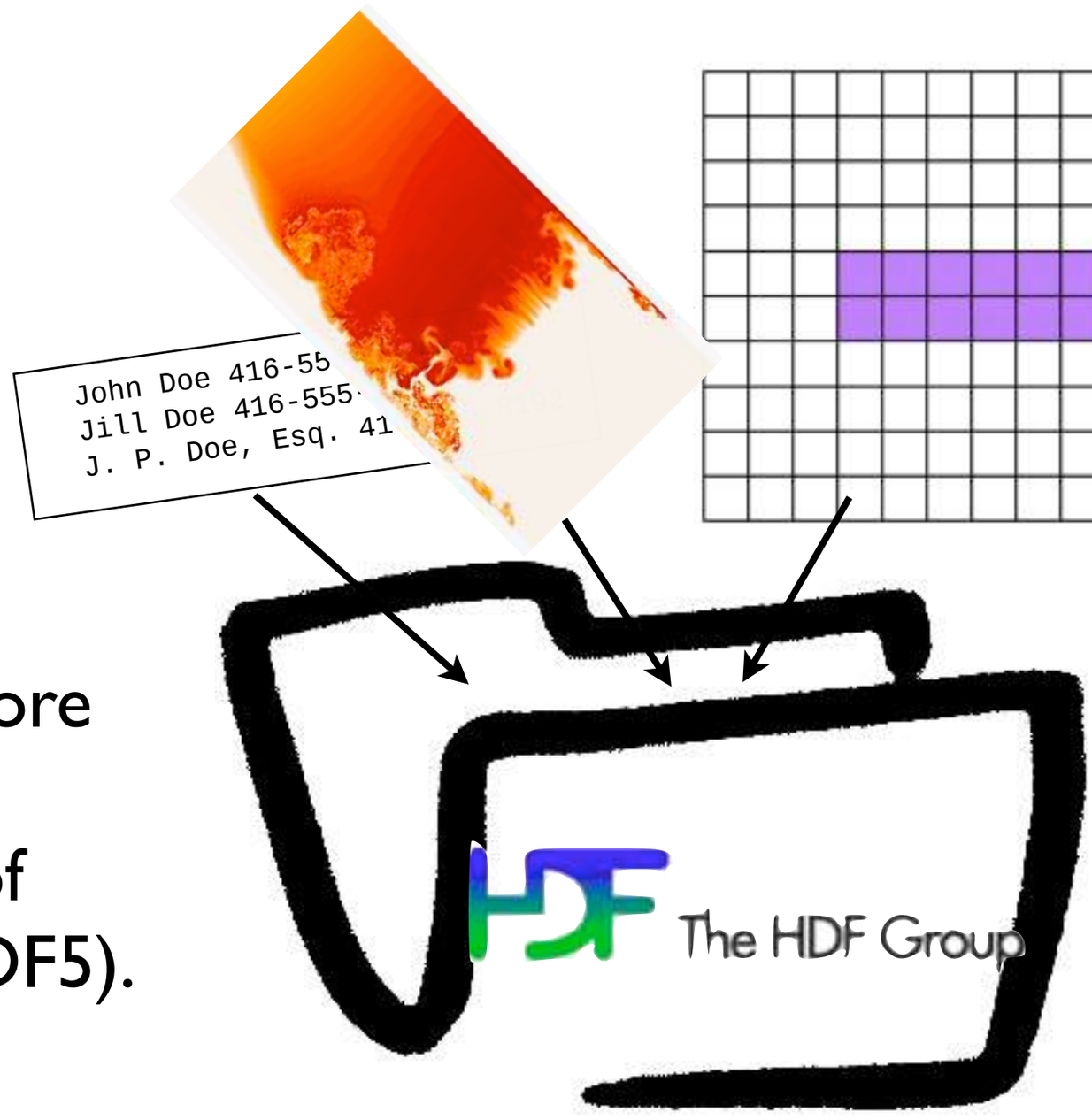
HDF5 is also self-describing file format and set of libraries
Unlike NetCDF, much more general; can shove almost any type of data in there
(We'll just be looking at large arrays, since that's our usual use case)



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

2darray-simple.c

```
/* identifiers */
hid_t file_id, dens_dataset_id, vel_dataset_id;
hid_t dens_dataspace_id, vel_dataspace_id;

/* sizes */
hsize_t densdims[2], veldims[3];

/* status */
herr_t status;

/* Create a new file - truncate anything existing, use default properties
*/
file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT,
H5P_DEFAULT);

/* HDF5 routines generally return a negative number on failure.
 * Should check return values! */
if (file_id < 0) {
    fprintf(stderr,"Could not open file %s\n", rundata.filename);
    return;}
```

2darray-simple.c

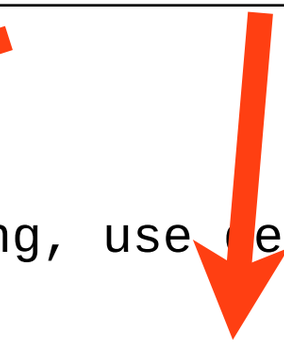
```
/* identifiers */
hid_t file_id, dens_dataset_id, vel_dataset_id;
hid_t dens_dataspace_id, vel_dataspace_id;


/* sizes */
hsize_t densdims[2], veldims[3];


/* status */
herr_t status;


/* Create a new file - truncate anything
*/
file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT,
H5P_DEFAULT);


/* HDF5 routines generally return a negative number on failure.
 * Should check return values! */
if (file_id < 0) {
    fprintf(stderr,"Could not open file %s\n", rundata.filename);
    return;}
```

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

2darray-simple.c

```
/* identifiers */
hid_t file_id, dens_dataset_id, vel_dataset_id;
hid_t dens_dataspace_id, vel_dataspace_id;


/* sizes */
hsize_t densdims[2], veldims[3];


/* status */
herr_t status;


/* Create a new file - truncate anything existing, use default properties
*/
file_id = H5Fcreate(rundata.filename, H5F_ACC_TRUNC, H5P_DEFAULT,
H5P_DEFAULT);


/* HDF5 routines generally return a negative number on failure.
 * Should check return values! */
if (file_id < 0) {
    fprintf(stderr,"Could not open file %s\n", rundata.filename);
    return;}
```

H5F, H5P… ?



et
calcul

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

2darray-simple.c

```
/* Create the data space for the two datasets. */
densdims[0] = rundata.nx; densdims[1] = rundata.ny;
veldims[0] = 2; veldims[1] = rundata.nx; veldims[2] = rundata.ny;

dens_dataspace_id = H5Screate_simple(2, densdims, NULL);
vel_dataspace_id  = H5Screate_simple(3, veldims,  NULL);

/* Create the datasets within the file.
 * H5T_IEEE_F64LE is a standard (IEEE) double precision (64 bit)
 * floating (F) data type and will work on any machine.
 * H5T_NATIVE_DOUBLE would work too */

dens_dataset_id = H5Dcreate(file_id, "dens", H5T_IEEE_F64LE,
                            dens_dataspace_id, H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);

vel_dataset_id  = H5Dcreate(file_id, "vel",  H5T_IEEE_F64LE,
                            vel_dataspace_id,  H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);
```

2darray-simple.c

```
/* Create the data space for the two datasets. */
densdims[0] = rundata.nx; densdims[1] = rundata.ny;
veldims[0] = 2; veldims[1] = rundata.nx; veldims[2] = rundata.ny;

dens_dataspace_id = H5Screate_simple(2, densdims, NULL);
vel_dataspace_id  = H5Screate_simple(3, veldims,  NULL);

/* Create the datasets within the file
 * H5T_IEEE_F64LE is a standard (IEEE)
 * floating (F) data type and will work
 * H5T_NATIVE_DOUBLE would work too */

dens_dataset_id = H5Dcreate(file_id, "d
                            dens_da
                            H5P_DEF

vel_dataset_id  = H5Dcreate(file_id, "v
                            vel_dat
                            H5P_DEF
```

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

2darray-simple.c

```
/* Create the data space for the two datasets. */
densdims[0] = rundata.nx; densdims[1] =
veldims[0] = 2; veldims[1] = rundata.nx


dens_dataspace_id = H5Screate_simple(2,
vel_dataspace_id  = H5Screate_simple(3,


/* Create the datasets within the file.
 * H5T_IEEE_F64LE is a standard (IEEE)
 * floating (F) data type and will work
 * H5T_NATIVE_DOUBLE would work too */


dens_dataset_id = H5Dcreate(file_id, "dens", H5T_IEEE_F64LE,
                            dens_dataspace_id, H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);


vel_dataset_id  = H5Dcreate(file_id, "vel",  H5T_IEEE_F64LE,
                            vel_dataspace_id,  H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);
```

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

et
calcul

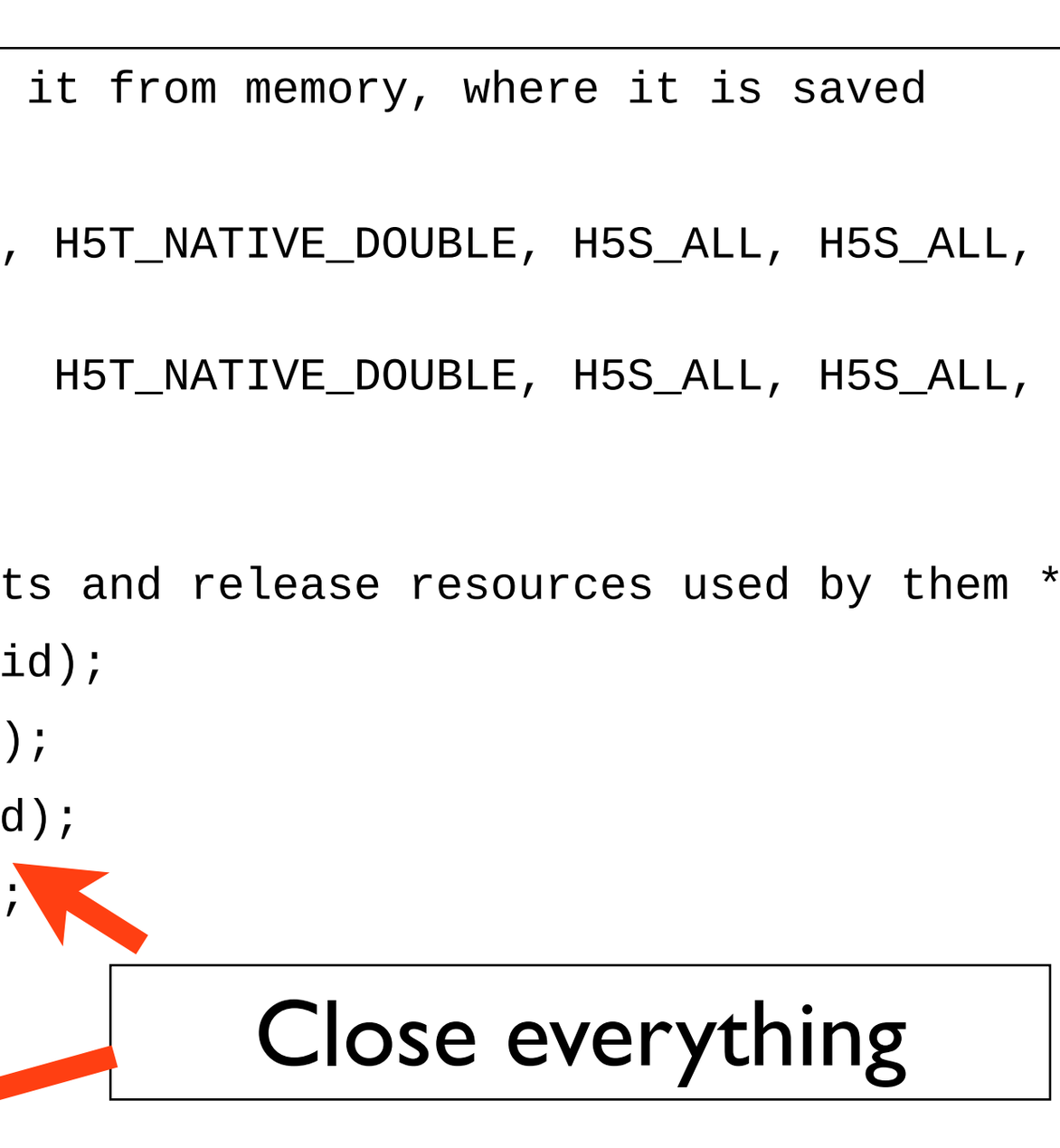
2darray-simple.c

```
/* Write the data.  We're writing it from memory, where it is saved
 * in NATIVE_DOUBLE format */
status = H5Dwrite(dens_dataset_id, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
H5P_DEFAULT, &(dens[0][0]));
status = H5Dwrite(vel_dataset_id,  H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
H5P_DEFAULT, &(vel[0][0][0]));


/* End access to groups & data sets and release resources used by them */
status = H5Sclose(dens_dataspace_id);
status = H5Dclose(dens_dataset_id);
status = H5Sclose(vel_dataspace_id);
status = H5Dclose(vel_dataset_id);

/* Close the file */
status = H5Fclose(file_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.

2darray-simple.c

```
/* Write the data.  We're writing it from memory, where it is saved
 * in NATIVE_DOUBLE format */
status = H5Dwrite(dens_dataset_id, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
H5P_DEFAULT, &(dens[0][0]));
status = H5Dwrite(vel_dataset_id,  H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
H5P_DEFAULT, &(vel[0][0][0]));


/* End access to groups & data sets and release resources used by them */
status = H5Sclose(dens_dataspace_id);
status = H5Dclose(dens_dataset_id);
status = H5Sclose(vel_dataspace_id);
status = H5Dclose(vel_dataset_id);

/* Close the file */
status = H5Fclose(file_id);
```



f2darray-simple.f90

```
integer(hid_t) :: file_id
integer(hid_t) :: dens_space_id, vel_space_id
integer(hid_t) :: dens_id, vel_id
integer(hsize_t), dimension(2) :: densdims
integer(hsize_t), dimension(3) :: veldims

integer :: status

! first we have to open the FORTRAN inter
call h5open_f(status)

! create the file, check return code
call h5fcreate_f(rundata%filename, H5F_ACC_TRUNC_F, file_id, status)
if (status /= 0) then
    print *,'Could not open file ', rundata%filename
    return
endif
```

Fortran: values are integer(hid_t) or integer(hsize_t)

f2darray-simple.f90

```
integer(hid_t) :: file_id
integer(hid_t) :: dens_space_id, vel_space_id
integer(hid_t) :: dens_id, vel_id
integer(hsize_t), dimension(2) :: densdims
integer(hsize_t), dimension(3) :: veldims

integer :: status

! first we have to open the FORTRAN interface.
call h5open_f(status)

! create the file, check return code
call h5fcreate_f(rundata%filename, H5F_ACC_TRUNC_F, file_id, status)
if (status /= 0) then
    print *,'Could not open file ', rundata%filename
    return
endif
```

Have to start the FORTRAN interface

f2darray-simple.f90

```
integer(hid_t) :: file_id
integer(hid_t) :: dens_space_id, vel_space_id
integer(hid_t) :: dens_id, vel_id
integer(hsize_t), dimension(2) :: densdims
integer(hsize_t), dimension(3) :: veldims

integer :: status

! first we have to open the FORTRAN interface.
call h5open_f(status)

! create the file, check return code
call h5fcreate_f(rundata%filename, H5F_ACC_TRUNC_F, file_id, status)
if (status /= 0) then
    print *,'Could not open file ', rundata%filename
    return
endif
```

See what I mean about _F?

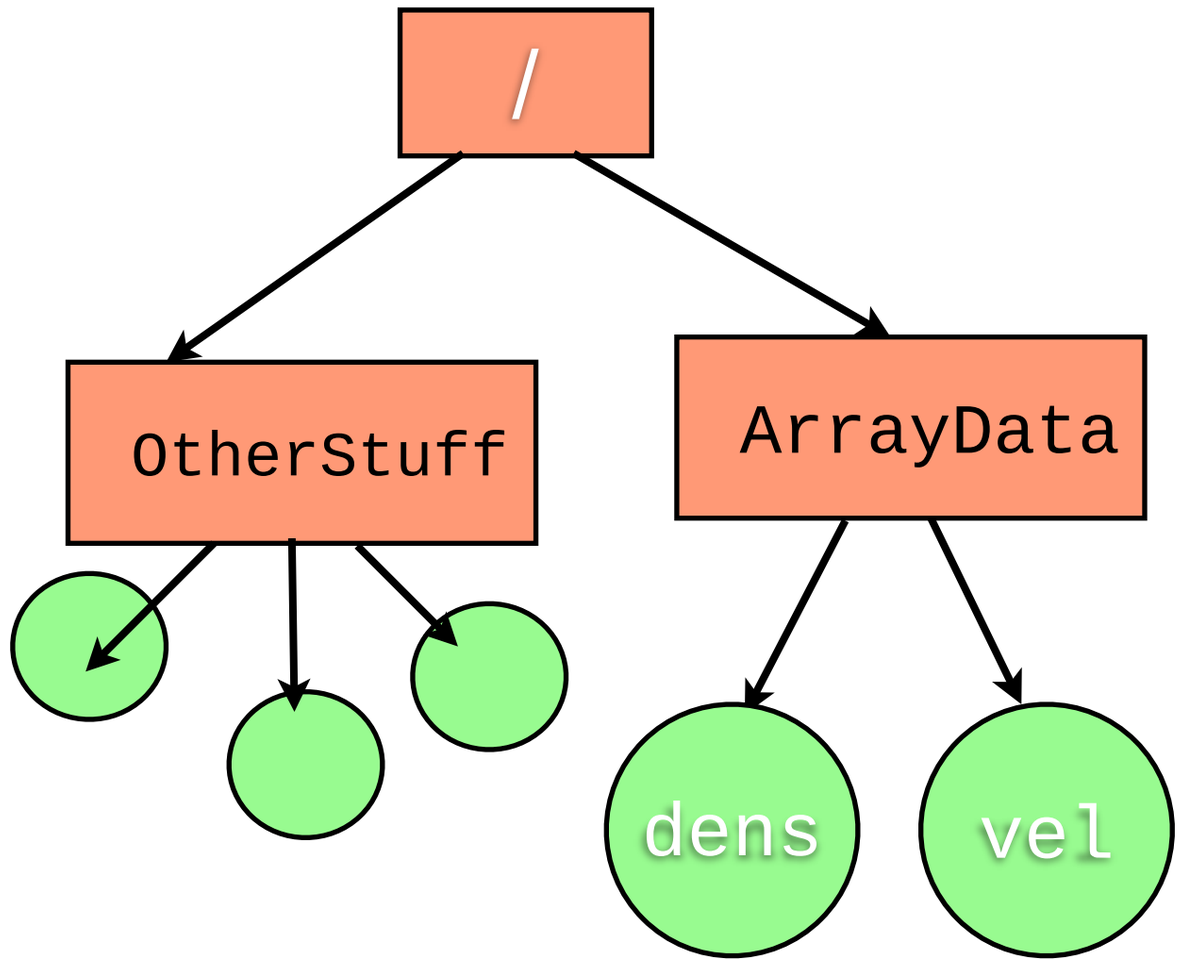
f2darray-simple.f90

```
! create the dataspaces corresponding to our variables
densdims = (/ rundata % nx, rundata % ny /)
call h5screate_simple_f(2, densdims, dens_space_id, status)

veldims = (/ 2, rundata % nx, rundata % ny /)
call h5screate_simple_f(3, veldims, vel_space_id, status)

! now that the dataspaces are defined, we can define variables on them

call h5dcreate_f(file_id, "dens", H5T_IEEE_F64LE, dens_space_id, dens_id,
status)
call h5dcreate_f(file_id, "vel" , H5T_IEEE_F64LE, vel_space_id,  vel_id,
status)
```

In F90 interface, a lot of less-common arguments are optional; fewer H5P_DEFAULTs kicking around

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

2darray.c

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

dens_dataset_id = H5Dcreate(file_id, "/ArrayData/dens", H5T_IEEE_F64LE,
                            dens_dataspace_id, H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);
vel_dataset_id  = H5Dcreate(file_id, "/ArrayData/vel",  H5T_IEEE_F64LE,
                            vel_dataspace_id,  H5P_DEFAULT,
                            H5P_DEFAULT  H5P_DEFAULT);
```

Can specify that a dataset goes in a group by giving it an “absolute path”...

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

dens_dataset_id = H5Dcreate(arr_group_id, "dens", H5T_IEEE_F64LE,
                            dens_dataspace_id, H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);
vel_dataset_id  = H5Dcreate(arr_group_id, "vel",  H5T_IEEE_F64LE,
                            vel_dataspace_id,  H5P_DEFAULT,
                            H5P_DEFAULT, H5P_DEFAULT);
```

...or just by creating it *in* the group, rather than the file.

What NetCDF, HDF *aren't*

Databases
Seem like - lots of information,
in key value pairs.
Relational databases -
interrelated tables of **small**
pieces of data
Very easy/fast to query
But can't do subarrays, etc..

Books

bid	title	isbn	author	date	volume
1	Big Cats	24589673-0	Cat, Simon	2003	2
2	Plants	24316759-1	Smith, Rose	1967	1
3	Sailing	34817645-0	Jones, Tom	1868	1

Transactions

tid	date	bid	pid	duedate	
1	02/11/08	3	2	16/11/08	
2	04/11/08	1	3	18/11/08	

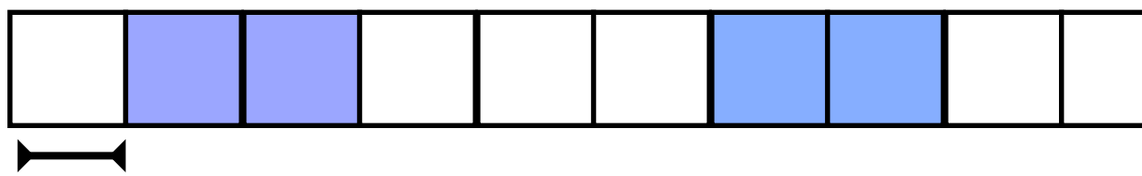
Borrowers

pid	firstname	lastname	address	phone	fines
1	Fred	Thompson	2 Reach Rd.	827-9867	2.25
2	Sam	Trunker	23 stone St.	243-0955	0
3	Tony	Sanchas	4 two Rd.	123-6453	0

SciNet
compute • calcul
CANADA

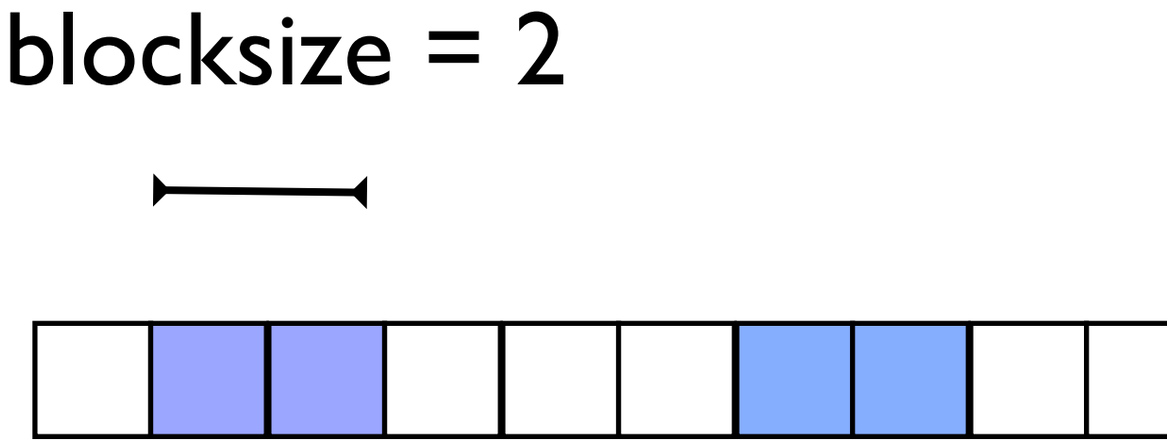
HDF5 Hyperslabs

- 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

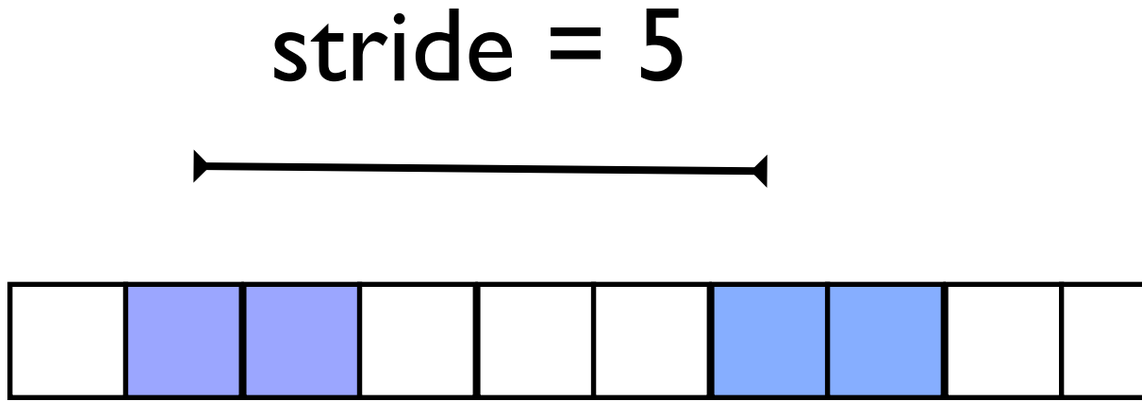
HDF5 Hyperslabs



- 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

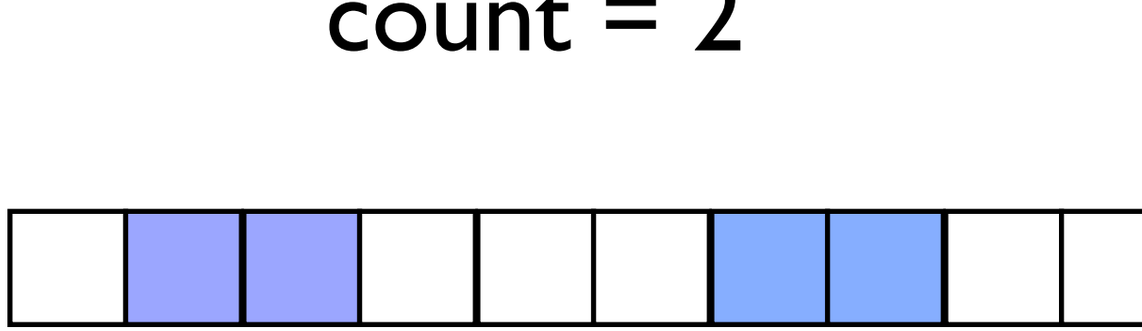
HDF5 Hyperslabs

- 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



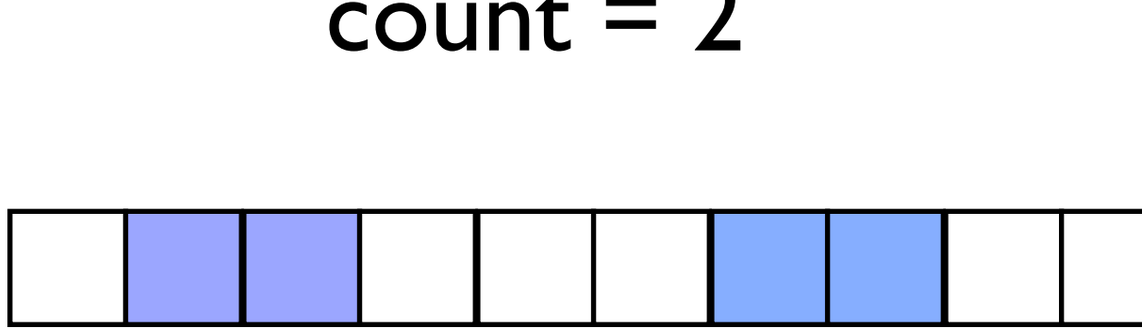
HDF5 Hyperslabs

- 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)



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.



parallel2darray.c

```
/* set the MPI-IO hints for better performance on GPFS */
MPI_Info_create(&info);
MPI_Info_set(info,"IBM_largeblock_io","true");

/* Set up the parallel environment for file access*/
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 collectively */
H5Pset_dxpl_mpio(dist_id, H5FD_MPIO_COLLECTIVE);
```

parallel2darray.c

```
/* set the MPI-IO hints for better performance on GPFS */
MPI_Info_create(&info);
MPI_Info_set(info,"IBM_largeblock_io","true");

/* Set up the parallel environment for file access*/
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 collectively */
H5Pset_dxpl_mpio(dist_id, H5FD_MPIO_COLLECTIVE);
```

Same as NetCDF; this is a property of the *file*

parallel2darray.c

```
/* set the MPI-IO hints for better performance on GPFS */
MPI_Info_create(&info);
MPI_Info_set(info,"IBM_largeblock_io","true");

/* Set up the parallel environment for file access*/
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 collectively */
H5Pset_dxpl_mpio(dist_id, H5FD_MPIO_COLLECTIVE);
```

Collective/independant: this is a property of accessing a *variable*

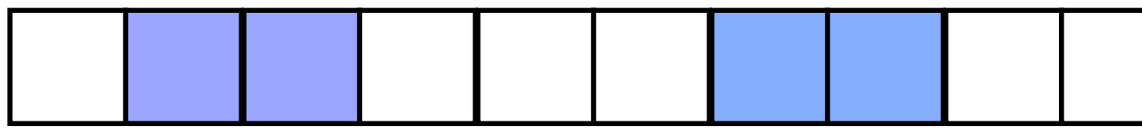
parallel2darray.c

```
offsets[0] = (rundata.globalnx/rundata.npx)*rundata.myx;
offsets[1] = (rundata.globalny/rundata.npy)*rundata.myy;
blocks[0]  = rundata.localnx;
strides[0] = strides[1] = 1;
counts[0] = counts[1] = 1;


globaldensspace = H5Dget_space(dens_dataset_id);
H5Sselect_hyperslab(globaldensspace,H5S_SELECT_SET, offsets,
strides, counts, blocks);


status = H5Dwrite(dens_dataset_id, H5T_NATIVE_DOUBLE,
loc_dens_dataspace_id, globaldensspace, dist_id, &(dens[0]
[0]));
```

Select hyperslab, and write; parallelism is in distribution_id

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