CS395T: Introduction to Scientific and Technical Computing

Instructors:

Dr. Karl W. Schulz, TACC

Dr. Victor Eijkhout, TACC

Outline

- Administrative Details
  - Grades for Assignment #1 posted
  - Assignment #2 due today
- Scientific Data Representations
  - Motivation
  - Big and Little “Endian”s
  - Common Scientific Formats
    - NetCDF
    - HDF

Scientific Data Files

- Most of us need to read and write data at some point in order to do some useful analysis:
  - Read geometry configurations for simulation driven research
  - Data mining algorithms
  - Post-process generated data (simple statistics or complicated iso-surface animations)
  - Continuing from a restart file (*does everyone know what a restart file is?)*
- Data is often written to disk as a byte-stream (C coded I/O) or as sequential I/O (Fortran code):
  - In a byte stream the data is written contiguously to disk
  - In (Fortran standard) sequential files the data is also written sequentially to disk, but a 4-byte record-size value is written at the beginning and end of each record
- Note: Fortran Direct I/O doesn't have record markers, but all records (writes/reads) can only have one size, making it unpopular

Scientific Data Files

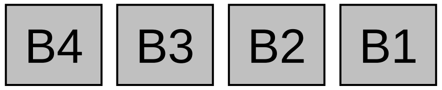
- There are three main aspects of data files that effect portability:
  - floating point representation
    - some machines, such as the Cray SV1, employ their own representation for floating point numbers (17-bit exponent, 47 bit mantissa), while most machines now use the **IEEE 754** representation (11-bit exponent, 52-bit mantissa)
    - Conversion between Cray FP an IEEE can be performed while reading or writing files, with Cray's FFIO library layer
  - byte-ordering
  - I/O formatting (eg. Fortran record markers)

Scientific Data Elements

- Byte Ordering
  - a **“*byte*”** is the lowest addressable storage unit
  - A **“*word*”** refers to a group of bytes, and is often used to represent a number:
    - in C, floats are usually 4 bytes and double are 8 bytes
    - in Fortran, reals are often 4 bytes, and double precision variables are 8 bytes
  - There are two different ways that the individual bytes of a word are stored in disk and memory. Note, this ordering has nothing to do with the individual bits in each byte-- only the order of how the bytes are stored.
  - These two orderings are called ***little*** and ***big Endian*** storage order

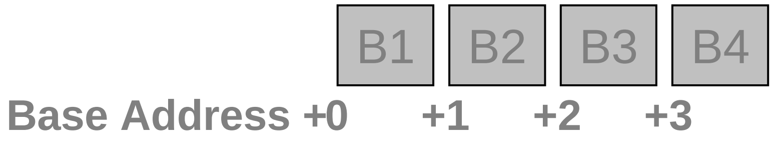
Little vs Big Endian Storage

Word (least significant digits in B1)



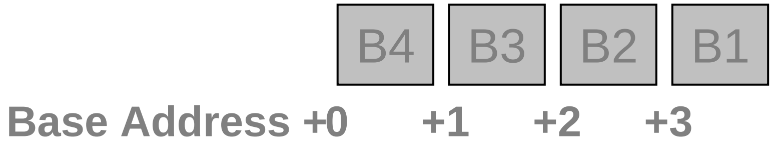
Little Ending

Little Endian means the bytes are stored from the least significant byte to the highest, beginning at the lowest address. The word is stored “little end first”



Big Ending

Big Endian means the bytes are stored from the most significant byte to the lowest, beginning at the lowest address. The word is stored “big end first”



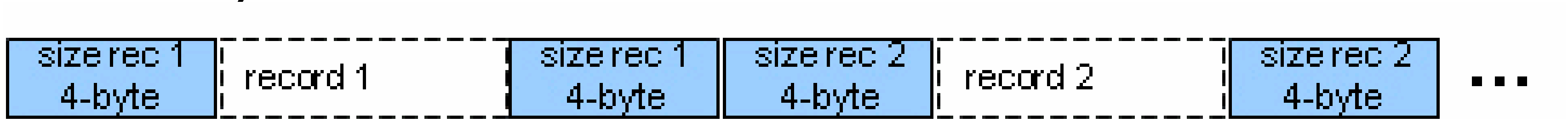
Little vs Big Endian Storage

- Example ordering of common architectures:

Processor	Endianess
Pentium 4	little-endian
Itanium 2	little-endian
Athlon	little-endian
Opteron	little-endian
PC970 (G5 - MAC)	big-endian
Power3/4/5 (AIX)	big-endian
MIPS (IRIX)	big-endian
Cray (Unicos)	big-endian

I/O Formatting - Fortran Unformatted

- Fortran sequential unformatted files are formatted to contain additional information to aid in file seeks
- For each Fortran write, a 4-byte integer is inserted at the beginning and end of the data sequence being written to disk (that is, for each write statement in the code)
- The two 4-byte additions are identical integers containing the number of data bytes (note that this number does not include the 8-bytes for the size values)



Q: what happens if you read big endian files on a little endian platform?

Little vs Big Endian Files

- In order to convert Fortran unformatted files, we have to swap the record header and footer, and the individual elements in each record

- Steps for Translation of Fortran file:
  - Determine whether file uses 4-byte or 8-byte words.
  - Read first 4 bytes. SWAP (determines length of next record=L)
  - SWAP next L bytes (either as 4-byte or 8-byte words)
  - Read next 4 bytes. SWAP (should match L read at beginning)
  - Repeat above 3 steps until an EOF (end of file) is reached.

Little vs Big Endian Conversion

- C code uses shift and "and" macro.
  - For 4-byte words: Shift all bytes to correct position and zero out everything else, then "or" components:

```
#define SWAP32(x) \
 x = ((((x) & 0xff000000) >> 24) | (((x) & 0x00ff0000) >>  8) |
\
      (((x) & 0x0000ff00) <<  8 ) | (((x) & 0x000000ff) << 24))
```

  - For 8-byte words:
    - use a 4-byte pointer (a) to the data,
    - SWAP on both 4 byte groups and then exchange the first 4 bytes and last 8 bytes:

```
SWAP32(a[j]); SWAP32(a[j+1]);
atmp   = a[j];
a[j]      = a[j+1];
a[j+1]  = atmp;
```

Little vs Big Endian Conversion Utilities

- *fendian_conv.c*: converts Fortran unformatted file.
  - Use conversion utility on machine that generated data. (Can be used on remote machine to regenerate original data. Could be modified to run anywhere.)
  - Syntax (4 and 8 byte data):
    - `fendian_conv 4 -i <input_file> -o <output_file>`
    - `fendian_conv 8 -i <input_file> -o <output_file>`

- *uswap* utility: also converts between big/little endian formats and can accommodate mixed word lengths
  - Can use uswap on machine that generated data, or on alternate endianness machine (with "-x" option).
  - Syntax:
    - `uswap -i infile -o outfile` (fixed 4 byte word length)
    - `uswap -cdf -i infile -o outfile` (char, double, and floats)

- More information for both utilities (and source code) at: www.tacc.utexas.edu/resources/user_guides/porting

Little vs Big Endian Conversion Utilities

- Other options exist for converting unformatted files using compiler options on the supporting platform

- Be sure to read the Fortran compiler man page for info on endian conversion - some compilers provide *runtime* options.

- For example, the Intel Fortran compiler includes a feature to convert I/O to big-endian output for predefined I/O units:

Intel Environment Variable (units=10,11,12):

```
C SHELL: setenv F_UFMTENDIAN 10,11,12
BASH:    export F_UFMTENDIAN=10,11,12
```

Platform Independent Binary Files

- XDR
  - Developed by SUN as part of NSF
  - Using XDR API gives platform independent binary files
  - `man xdr_vector`
  - `man xdr_string`
- NETCDF – Unidata Network Common Data Form
  - Common format used in Climate/Weather/Ocean applications
  - http://my.unidata.ucar.edu/content/software/netcdf/docs.html
  - `module load netcdf; man netcdf`
- HDF - Hierarchical Data Format developed by NCSA
  - http://hdf.ncsa.uiuc.edu/

NetCDF Overview

netCDF Overview

- netCDF stands for Network Common Data Format
  - a binary data format standard for exchanging scientific data
  - developed and supported by Unidata/UCAR (University Corporation for Atmospheric Research - a nonprofit consortium of 66 universities)
- NetCDF data is:
  - **Self-Describing**: file includes information about the data it contains.
  - **Portable:** file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers
  - **Appendable:** data may be appended to a properly structured netCDF file without copying the dataset or redefining its structure.
  - **Sharable:** one writer and multiple readers may simultaneously access the same netCDF file.
  - **Archivable:** access to all earlier forms of netCDF data will be supported by current and future versions of the software
- Data access in netCDF is direct, not sequential
  - This design means that a small subset of a large dataset may be accessed efficiently, without first reading through all the preceding data (and when the order in which data is read is different from the way it is written)

netCDF Overview

- netCDF provides an effective way to store and retrieve scientific data through and provides binaries as well as common language bindings
- API's exist for reading and writing data:
  - C/C++
  - Fortran
  - Java
  - Perl, MATLAB, Python, and Ruby
- netCDF is available as a precompiled library on TACC systems (*which provides the C/C++ and Fortran interfaces*)

Why Use netCDF?

- Facilitate the use of common datasets by distinct applications
- Permit datasets to be transported between or shared by dissimilar computers transparently (without translation)
- Reduce programming effort usually spent interpreting formats
- Reduce errors from misinterpreting data and ancillary data
- Output data from one application into another
- Establish an interface standard which simplifies the inclusion of new software into existing system

History of NetCDF

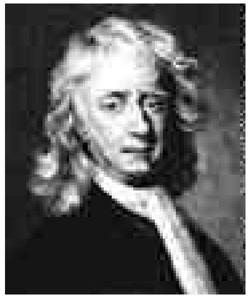
1988 — netCDF developed at Unidata

1991 — netCDF 2.0 released

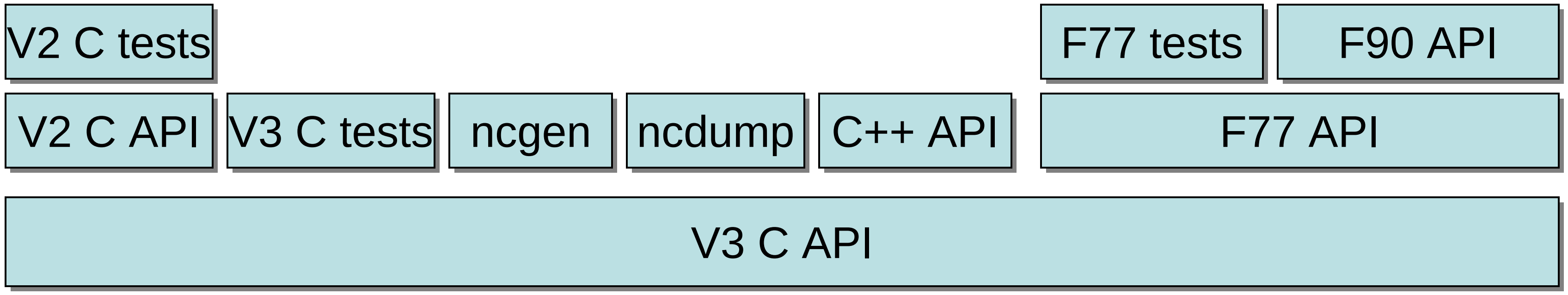
1996 — netCDF 3.0 released

2004 — netCDF 3.6.0 released

2005 — netCDF 4.0 beta released



Software Architecture of NetCDF-3



- Fortran, C++ and V2 APIs are all built on top of the C API.
- Other language APIs (perl, python, MatLab, etc.) also use the C API

netCDF Dataset Objects

- Variables: named arrays
  - name, type, shape, attributes, values
  - Fixed sized variables: array of fixed dimensions
  - Record variables: array with its most-significant dimension UNLIMITED
  - Coordinate variables: 1-D array with the same name as its dimension
- Dimensions
  - name, length
- Attributes - “metadata about the data”
  - name, type, values, length
  - Variable attributes
  - Global attributes

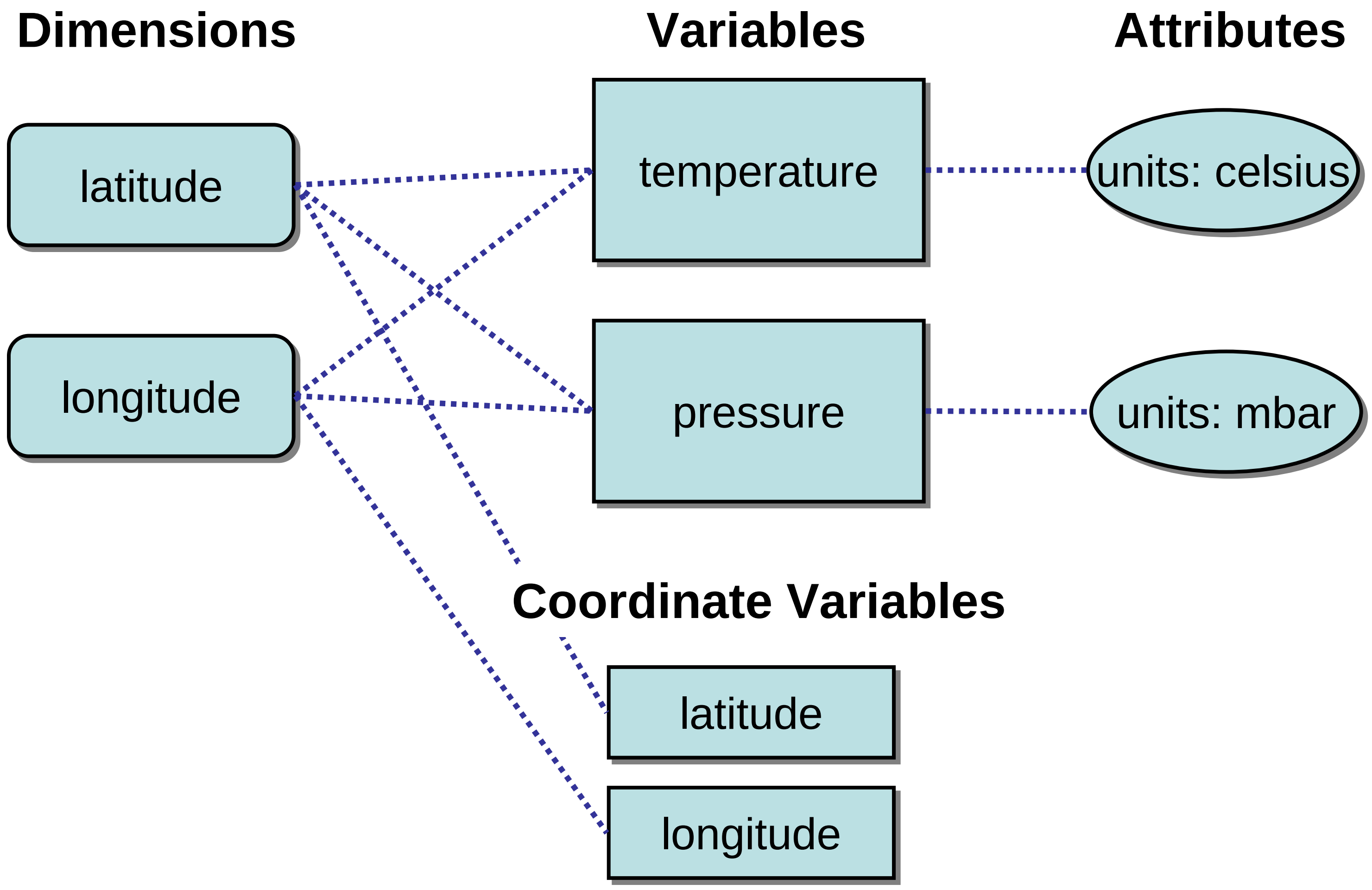
netCDF Dataset Objects

- Coordinate Variable – a 1D variable with the same name as a dimension, which stores values for each dimension value (for example, “latitude”)
- Unlimited Dimension – a dimension which has no maximum size
  - Data can always be extended along the unlimited dimension
  - Recommended when creating a new netCDF dataset that may be extended later

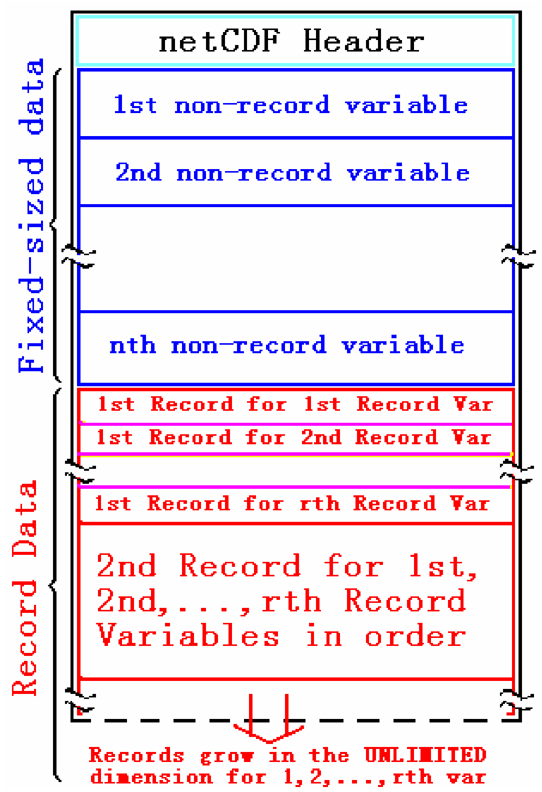
netCDF Example

- Suppose you to store temperature and pressure values on a 2D latitude/longitude grid
- In addition to the data *(ie. Temperature and Pressure),* you want to store information about the exact values of the lat/lon grid
- You may also have additional data to store, for example, the units of the data values

netCDF Example



netCDF File Format - binary


netCDF Header
1st non-record variable
2nd non-record variable
nth non-record variable
1st Record for 1st Record Var
1st Record for 2nd Record Var
1st Record for rth Record Var
2nd Record for 1st, 2nd,...,rth Record Variables in order
Records grow in the UNLIMITED dimension for 1,2,...,rth var
Fixed-sized data
Record Data

- Header
  - –Number of record variables
  - –Dimension list
  - –Global attribute list
  - –Variable list
- Data (row-major)
  - –Fixed-sized data

    data for each variable is stored contiguously
  - –Record data

    a variable number of fixed-size records, each of which contains one record for each of the record variables in order.
- Both use extended XDR

Accessing netCDF Data

- Access is direct - efficient
- Can access elements of arrays
- Can stride through arrays to pick up portions of data

Important NetCDF Functions

- nc_create and nc_open: used to create and open files.
- nc_enddef, nc_close: used to end definition of and close files
- nc_def_dim, nc_def_var, nc_put_att_*: used to define dimensions, variables, and attributes
- nc_inq, nc_inq_var, nc_inq_dim, nc_get_att_* :used to learn about dims, vars, and atts.
- nc_put_vara_* , nc_get_vara_*: used to write and read data.

C Functions to Define Metadata

```
/* Create the file. */
 if ((retval = nc_create(FILE_NAME, NC_CLOBBER, &ncid)))
    return retval;

  /* Define the dimensions. */
  if ((retval = nc_def_dim(ncid, LAT_NAME, LAT_LEN, &lat_dimid)))
    return retval;
  if ((retval = nc_def_dim(ncid, LON_NAME, LON_LEN, &lon_dimid)))
    return retval;

  /* Define the variables. */
  dimids[0] = lat_dimid;
  dimids[1] = lon_dimid;
  if ((retval = nc_def_var(ncid, PRES_NAME, NC_FLOAT, NDIMS, dimids, &pres_varid)))
    return retval
  if ((retval = nc_def_var(ncid, TEMP_NAME, NC_FLOAT, NDIMS, dimids, &temp_varid)))
    return retval;

  /* End define mode. */
  if ((retval = nc_enddef(ncid)))
    return retval;
```

NC_CLOBBER option indicates that it is ok to overwrite existing file

C Functions to Write Data

```
/* Write the data. */
 if ((retval = nc_put_var_float(ncid, pres_varid, pres_out)))
    return retval;
 if ((retval = nc_put_var_float(ncid, temp_varid, temp_out)))
    return retval;

 /* Close the file. */
 if ((retval = nc_close(ncid)))
    return retval;
```

the nc_put_var_float() function writes all the values for the variable; to write a single value use nc_put_var1_float()

C Example – Getting Data

```
 /* Open the file. */

if ((retval = nc_open(FILE_NAME, 0, &ncid)))
      return retval;

/* Query the variable of interest */

status = nc_inq_varid (ncid, “Pressure", &pres_in);
if (status != NC_NOERR)
       handle_error(status);

/* Read the data. */
if ((retval = nc_get_var_float(ncid, 0, pres_in)))
    return retval;

/* Do something useful with the data… */

/* Close the file. */
   if ((retval = nc_close(ncid)))
      return retval;
```

Data Reading and Writing Functions

- There are 5 ways to read/write data of each type.
  - var – reads/writes entire variable at once
  - var1 – reads/writes a single value
  - vara – reads/writes an array subset
  - vars – reads/writes an array by slices
  - varm – reads/writes a mapped array
- Ex.: nc_put_vars_short writes shorts by slices.

Attributes

- Attributes are 1-D arrays of any of the 6 netCDF types
- Read/write them with functions like: nc_get_att_float and nc_put_att_int
- Attributes may be attached to a variable, or may be global to the file

NetCDF File Formats

- Starting with 3.6.0, netCDF supports two binary data formats:
  - NetCDF Classic Format is the format that has been in use for netCDF files from the beginning
  - NetCDF 64-bit Offset Format was introduced in 3.6.0 and allows much larger files (*ie. files > 2 GB)*
- Best to build netCDF on Linux with 64-bit offset support:

```
export CPPFLAGS="-D_FILE_OFFSET_BITS=64
-D_LARGEFILE_SOURCE"
```

Useful commands - ncdump

- Ncdump useful for quick glimpse of what kind of data file contains - queries and displays the file's metadata

```
> ncdump -h wrfinput_d01 | less
 netcdf wrfinput_d01 {
 dimensions:
      Time = UNLIMITED ; // (1 currently)
      DateStrLen = 19 ;
      west_east = 424 ;
      south_north = 299 ;
      west_east_stag = 425 ;
 ............................................
 float V(Time, bottom_top, south_north_stag, west_east) ;
              V:FieldType = 104 ;
              V:MemoryOrder = "XYZ" ;
              V:description = "y-wind component" ;
              V:units = "m s-1" ;
              V:stagger = "Y" ;
```

NetCDF Compilations

- Once you start using the netCDF API, you will need to start linking your application against the library

- Also need to include the appropriate header file in your application (or you can access a F90 module if you have built a Fortran interface):

```
#include "netcdf.h"
```

- Example compile:

```
> icc -I /opt/apps/netcdf/netcdf-3.6.1/include/
    -L /opt/apps/netcdf/netcdf-3.6.1/lib/
    example.c -lnetcdf
```

Useful commands - ncdump

- In addition to displaying metadata, ncdump can show all data values, or values for a requested variable:

```
> ncdump -v V wrfinput_d01 | less
```

```
data:

 V =
  -5.943535, -5.954164, -5.959685, -5.960224, -5.957259, -5.952528,
   -5.947741, -5.944627, -5.944635, -5.947225, -5.949976, -5.950274,
   -5.945539, -5.933652, -5.914801, -5.890519, -5.861694, -5.829077,
   -5.793406, -5.755308, -5.714824, -5.671347, -5.624143, -5.57258,
   -5.516121, -5.452713, -5.380764, -5.30263, -5.222619, -5.145173,
   -5.074714, -5.012296, -4.95566, -4.903169, -4.853917, -4.806993,
   -4.761831, -4.719834, -4.681822, -4.64602, -4.610251, -4.57234,
   -4.530297, -4.483756, -4.434185, -4.383417, -4.333199, -4.285205,
```

Why Add to NetCDF-3?

- Increasingly complex data sets call for greater organization
- Size limits, unthinkably huge in 1988, are routinely reached in 2006
- Parallel I/O is required for advanced scientific applications
- Desire to have interoperability with HDF5

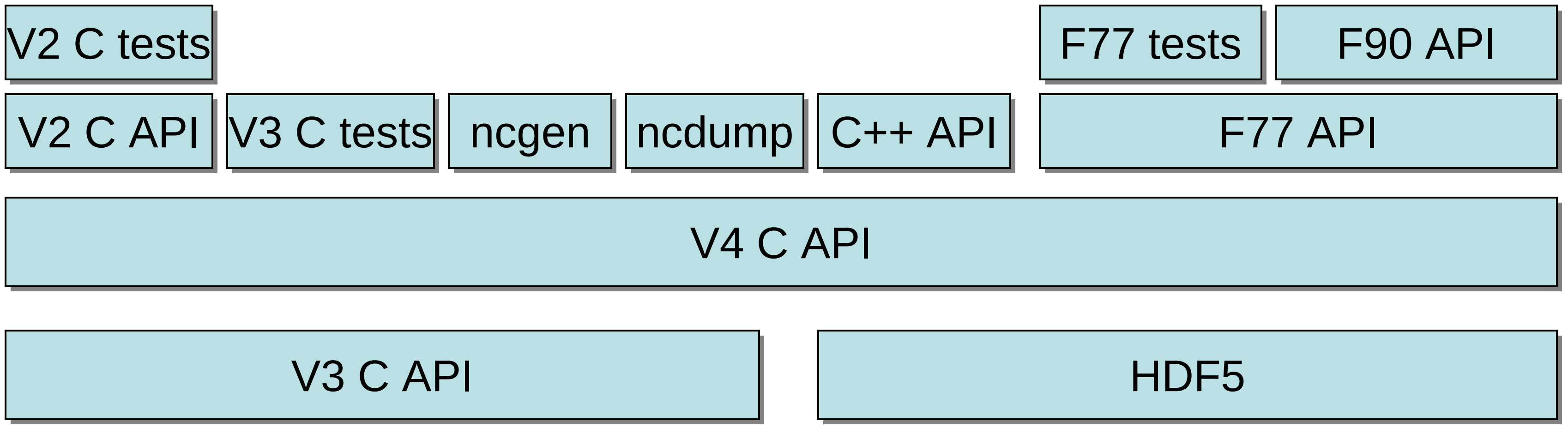
NetCDF-4

- NetCDF-4 aims to provide the netCDF API as a front end for HDF5
- Funded by NASA, executed at Unidata and NCSA
- Includes reliable netCDF-3 code, and is fully backward compatible

NetCDF-4 Feature Review

- Multiple unlimited dimensions
- How to use groups
- Using compound types
- Other new types
- Variable length arrays
- Parallel I/O
- HDF5 Interoperability

Software Architecture of NetCDF-4



References/Acknowledgements

- NetCDF Software: http://www.unidata.ucar.edu/software/netcdf/
- NetCDF Documentation: http://www.unidata.ucar.edu/software/netcdf/docs/
- NetCDF Version 4: http://www.unidata.ucar.edu/software/netcdf/netcdf-4/