HANDLING DATA AND PROVENANCE IN AN AI WORLD

TOOLS AND TECHNIQUES FOR MANY FILES AND REMOTE ACCESS

BRYAN LAWRENCE, DAVID HASSELL, SADIE BARTHOLOMEW AND VALERIU PREDOI

18TH NOVEMBER 2025

A little tour of some recent work carried out in CMS, aiming at delivering and exploiting FAIR data, accessible and usable locally and remotely

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

University of Reading

Outline

Abstract: NCAS CMS spend a lot of effort on tools and information systems to handle the data and complexity of both observational and simulation data. In this seminar we cover a few of those tools and explain how we're positioning these tools to support the FAIR production and use of observations, simple (e.g Healpix) and complex(e.g. LFRIC mesh) model data , in the context of rapidly exploding volumes and needs. We're working on data compression, on model documentation and many aspects of the publication workflow for CMIP7, and starting a project on methods to evaluate the data produced by AI models (both local and remote). The seminar will introduce a way of thinking about the relationship between metadata and data use, and showcase a a few of these the tools we are working on. The underlying message will be one of exploiting domain specific tools rather than re-inventing things for every application.

1. Intro: Motivation and Agreeing some language
2. From Metadata to Tools (A).
3. Documentation (B)
4. Aggregating Data (mixing A and B)
5. Why you should care about chunking
6. Applications
7. Summary

The CMS team

CMS 2025
(*Lawrence*)
Head: *Lister*

(Luke Abraham.
Cambridge)

- Data Lead *Hassell*
  - CF Tools *Bartholomew*
  - CF Tools & Standards Hassell
  - ESMValTool (UKESM) *Predoi*
- Workflow Lead *Hatcher*
  - PUMA & HPC Workflow *Hatcher*
  - HPC Workflow & Support *A.N.Other*
- HPC Lead *Wilson*
  - HPC Specialist *Bilbao*
  - HPC Specialist *Case*
  - HPC Specialist *Osprey*
  - HPC Specialist *Wilson*
- Applications Lead *Lawrence*
  - AFESP (ECMWF et al) *Ahmadi*
  - Land Surface (JULES) *Livings*
  - Coupled Systems (UKESM) *Stringer*

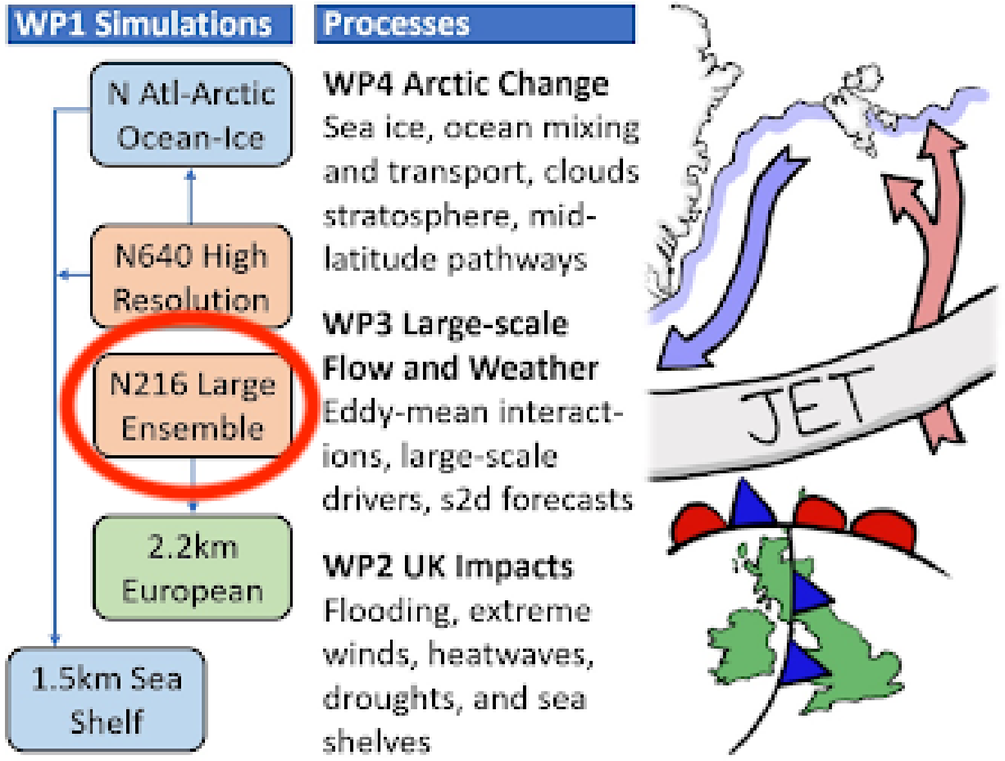
National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Handling data and provenance in an AI world
18th November 2025

University of Reading

Motivation: CANARI

Len Shaffrey, Jon Robson, Reinhard Schiemann



Model

- HadGEM3-GC3.1 (the UK Met Office CMIP6 model)
- N216-ORCA0.25 (~60 km atmosphere, ¼° ocean)

Experiments

- CMIP6 historical (1950 – 2014)
- CMIP6 SSP3-7.0 scenario (2015 – 2100)

Ensemble

- 40 members
- Hybrid initialisation
- 8 macro ICs x 5 micro perturbations

Output

- 6000 years - “CMIP-like”.
- UM boundary conditions (“Frames”) for identified European storms, NZ and East Asia
- RCM boundary conditions for all CORDEX regions

Managing Data - CANARI Example

Simulations

- 40 Historical
- 40 Future
- = 40 x 150 = 6000 years of simulation

Output

- 822 ”atomic” datasets per atmospheric simulation.
- UM boundary conditions (“frames”) for several regions.
- Restart dumps monthly plus extra for “specially identified storms”.
- Additional variables to support all CORDEX regions.

Data Volumes (compressed): ~5.5 PB

- 2 PB atmos fields
- 0.5 PB atmos restarts
- 0.4 PB ocean fields
- 0.1 PB ocean restarts
- 0.1PB ice fields and restarts
- 2.1 PB of frames etc

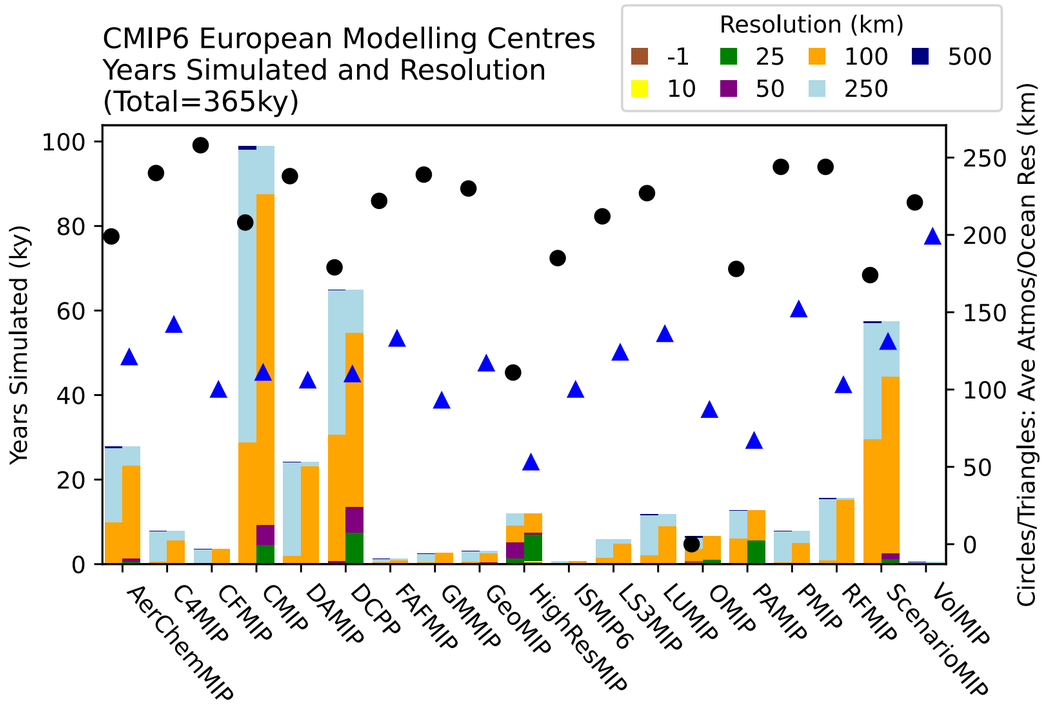
Files: 4.5 Million Files!

Managed mostly on tape, with ~ 1 PB of JASMIN group work space (cache disk).

Doing this sort of thing requires sustained human effort and lots of software!

Motivation - CMIP6, CMIP7

CMIP6



CMIP7

Working Groups:

- WCRP ESMO Infrastructure Panel (WIP)
  - (CMIP6 Lawrence)
  - CMIP7 Hassell
- Obs4MIPS Working Group:
  - Bartholomew
- Citation
  - Lawrence
- Model Documentation
  - Hassell, Lawrence

CMIP data are defined by the Climate Forecast conventions!

AI models make making data too easy!

I had an email discussing thousand member ensembles using an AI model.

My response:

- For sure that's computationally possible, but could we handle the data?
- What if we wanted to inter-compare thousand member ensembles?
- We won't be moving that data, so do we have methods to find, and remotely analyse such data?

No real answers to the this last question in this seminar, but some baby steps ...

Actually, never mind AI: For anyone working with high-resolution models now, data wrangling is a first order constraint on what is possible, both for storage and the people involved.

LANGUAGE AND TOOLS

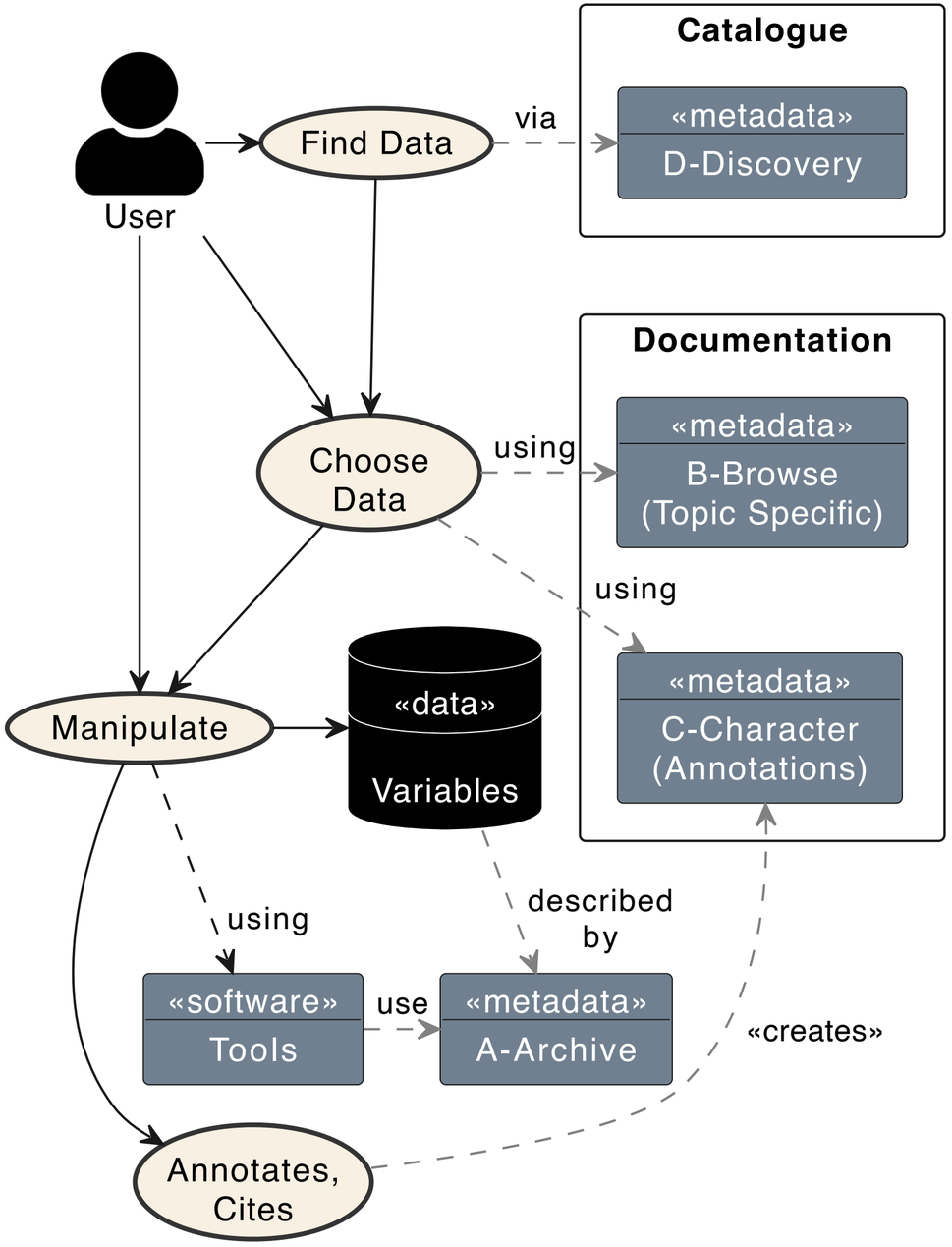
Why we need metadata

FAIR

Findable, Accessible, Interoperable, Reusable

(In our world that means MORE than CF compliant!)

Category	Definition
D - Discovery	Information to tell you the dataset exists.
C - Character/ Citation	What you and others say about the data.
B - Browse	Information to help find what you want in the dataset
A - Access/ Archive	Metadata needed by the tools to make sense of the data in the files

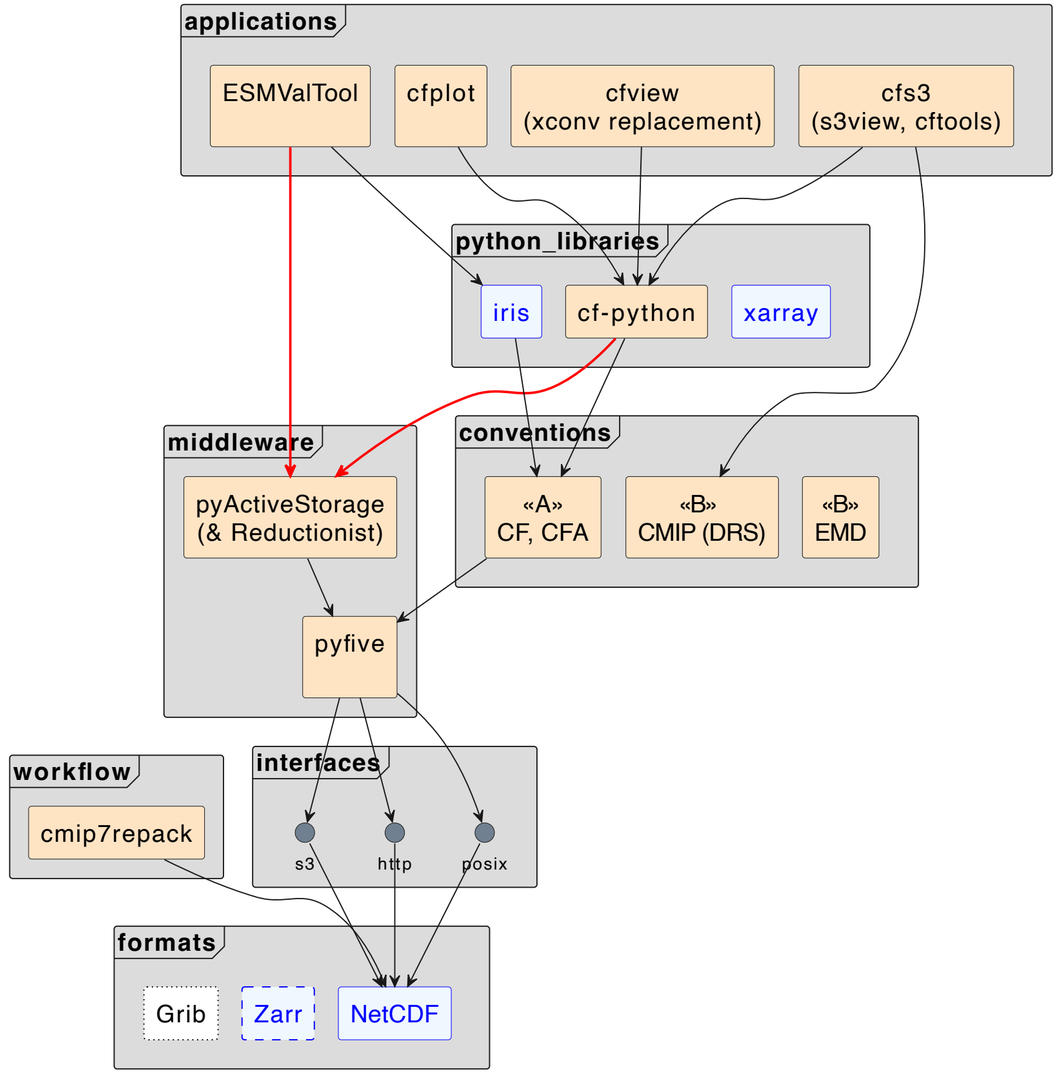


The (CMS) Application Stack

This talk will be a brief introduction to what is going on in most of the things[1] shaded in brown:

- Applications and Libraries: Getting science done (including creating FAIR data with sensible chunking).
- Middleware: Efficient computation and access
- Conventions: Enabling FAIRness and library functionality.
- Interfaces: We need to think about the different ways data is exposed and the implications for how we store it.
- Workflow: (Chunking again)

[1]: We don't have time to talk about cfview this time, or indeed say much about cfplot (sorry Sadie!).

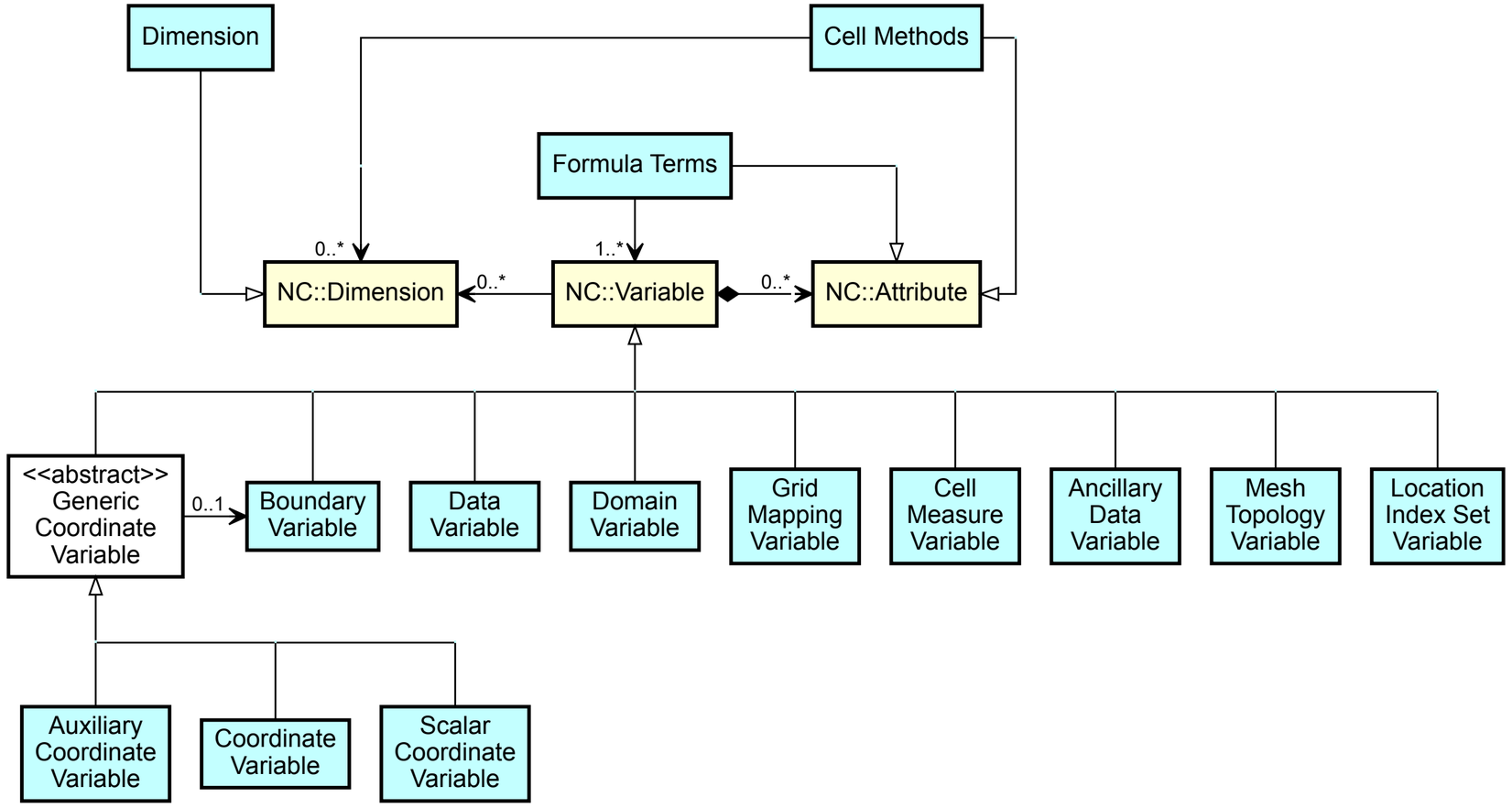


FROM METADATA TO TOOLS (A)

A: The CF conventions

The relationships between CF-netCDF elements and their corresponding netCDF variables, dimensions and attributes (identified here with the "NC" prefix).

(From the CF conventions themselves).



A - The CF Data Model

The CF data model is independent of the storage format and outlines the things we really care about from a domain specific point of view.

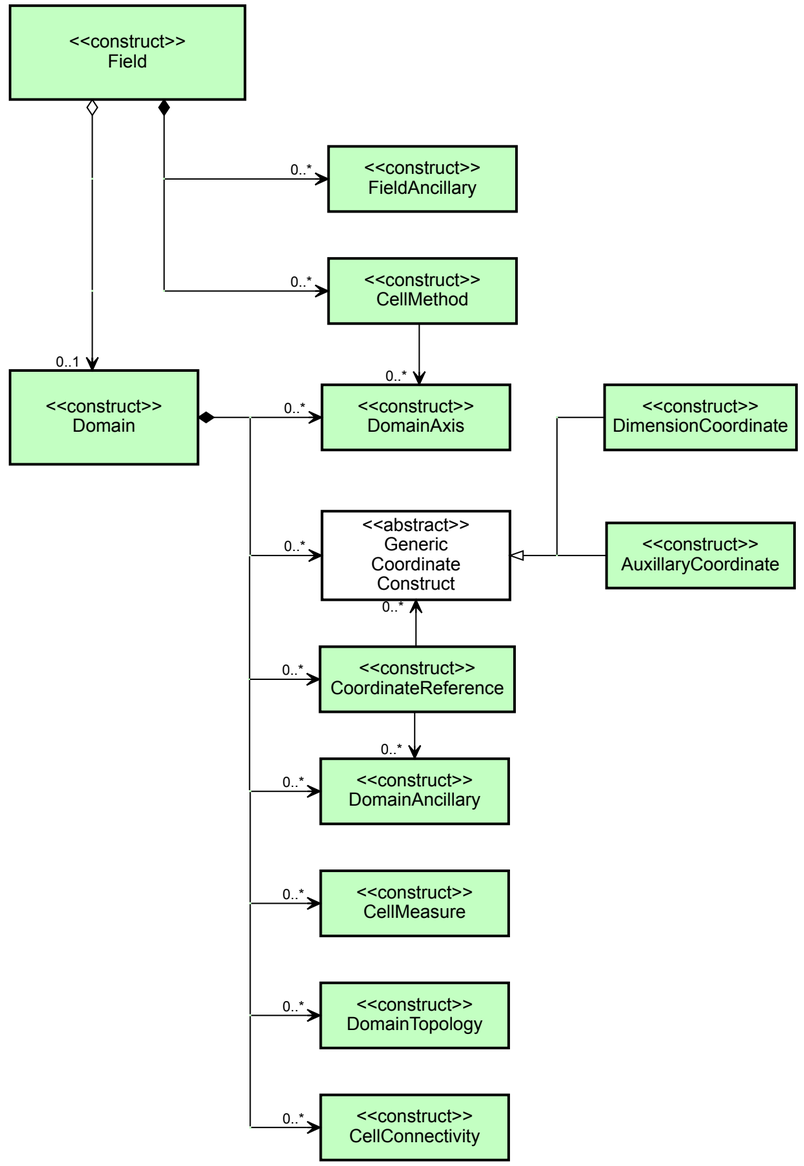
The data model has several benefits:

- Format agnostic.
- Clarity about what the words in the convention mean.
- Software needs to understand and exploit the data model.

The big disadvantage:

- Software needs to understand and exploit the data model.

But someone else can do that for you! Libraries!



A: Using CF to make it easy to create and use data

cf 3.18.2

A CF-compliant earth science data analysis library

Star 144

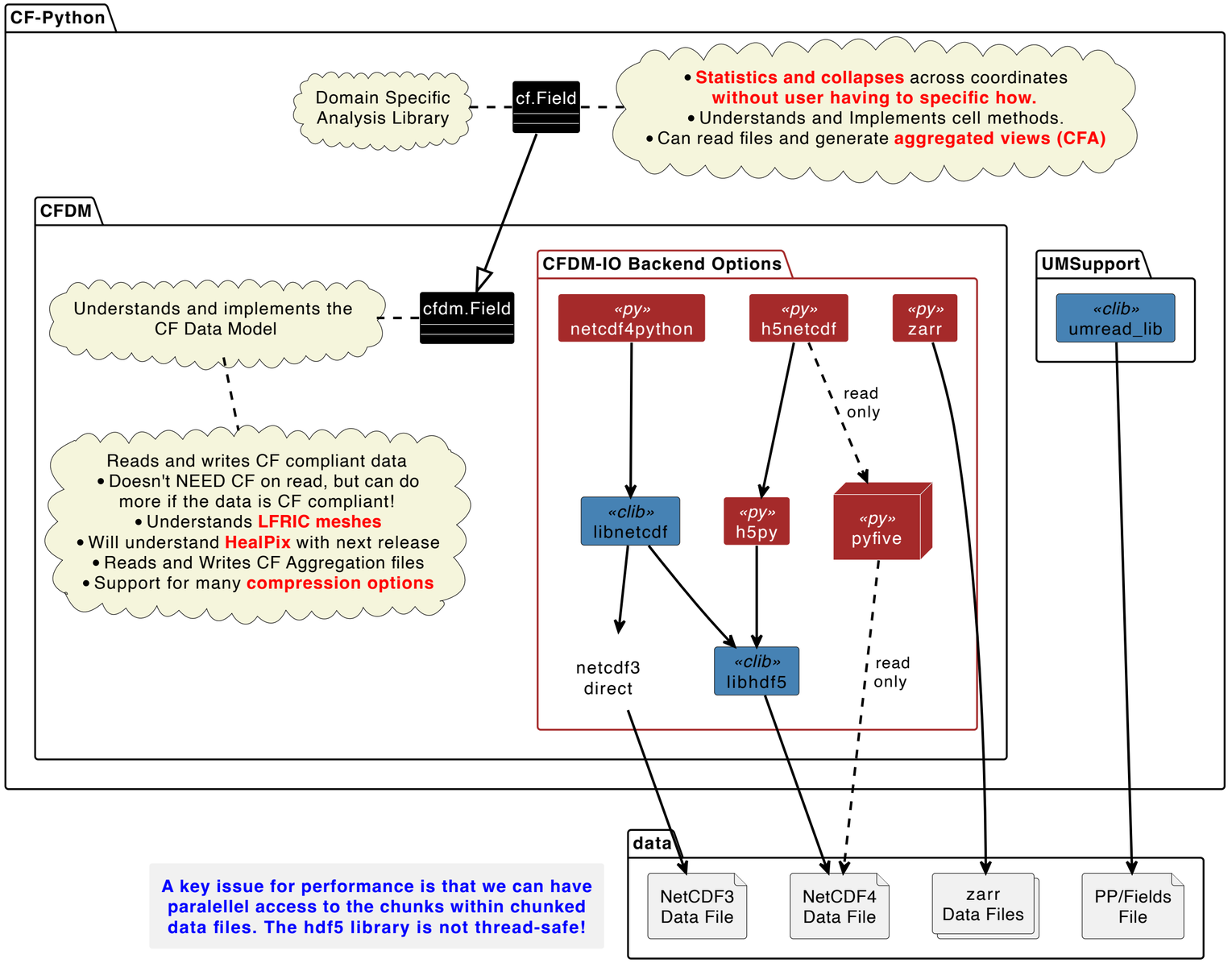
Quick search

Go

- Introduction
- Installation
- Cheat Sheet
- Recipes using cf
- Tutorial
- API reference
- Aggregation rules
- Performance
- Releases
- Change log
- Contributing

cf development has been supported by the ERC through Seachange and Couplet; by the EC Horizon 2020 programme through IS-ENES3; by NERC through UKFAFMIP; and by NCAS.

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL



A key issue for performance is that we can have paralellel access to the chunks within chunked data files. The hdf5 library is not thread-safe!

UKCP Example

Consider a netcdf file with this (abbreviated) global metadata:

```
tas:Conventions = "CF-1.6" ;
                tas:label_units = "C" ;
                tas:domain = "uk" ;
                tas:version = "v20180607" ;
                tas:scenario = "rcp85" ;
                tas:title = "UKCP18 land projections - 12km regional climate ...";
                tas:project = "UKCP18" ;
                tas:level = "1.5m" ;
                tas:collection = "land-rcm" ;
                tas:creation_date = "2018-06-07T13:33:09" ;
                tas:institution_id = "MOHC" ;
                tas:source = "UKCP18 regional realisation from a set of ..." ;
                tas:frequency = "mon" ;
                tas:references = "Murphy JM, ..." ;
```

(Without the bounds variables and with metadata lines removed):

```
p5dump ukcp_rcm_test.nc
File: ukcp_rcm_test.nc {
dimensions:
        strlen27 = 27;
        bounds2 = 2;
        projection_y_coordinate = 110;
        strlen64 = 64;
        time = 1;
        projection_x_coordinate = 83;
        ensemble_member = 1;
variables:
        int32 ensemble_member(ensemble_member) ;
        float64 time(ensemble_member) ;
        float64 projection_y_coordinate(projection_y_coordinate) ;
        float64 projection_x_coordinate(projection_x_coordinate) ;
        float32 strlen27(strlen27) ;
        char ensemble_member_id(ensemble_member, strlen27) ;
        float64 latitude(projection_y_coordinate, projection_x_coordinate) ;
        float64 longitude(projection_y_coordinate, projection_x_coordinate) ;
        int32 month_number(time) ;
        int32 year(time) ;
        float32 strlen64(strlen64) ;
        char yyyymm(time, strlen64) ;
        char transverse_mercator ;
                transverse_mercator:latitude_of_projection_origin = [49.] ;
```

The xarray version:

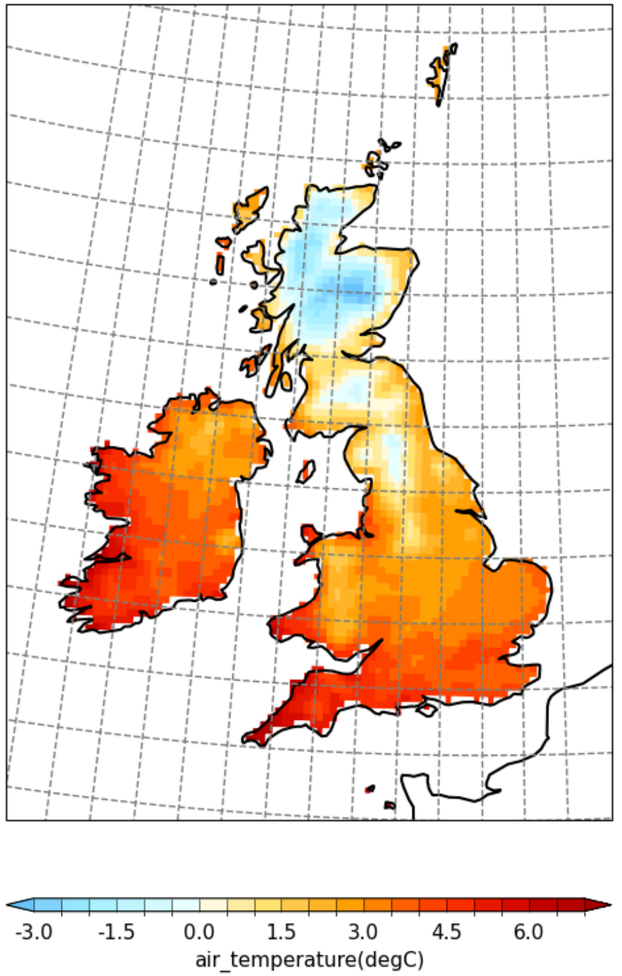
```
print(ds)
<xarray.Dataset>> Size: 187kB
Dimensions:                          (time: 1, bounds2: 2,
                                      projection_y_coordinate: 110,
                                      projection_x_coordinate: 83,
                                      ensemble_member: 1)
Coordinates:
  * time                             (time) object 8B 1980-12-16 00:00:00
  * projection_y_coordinate          (projection_y_coordinate) float64 880B -8.7...
  * projection_x_coordinate          (projection_x_coordinate) float64 664B -1.8...
  * ensemble_member                  (ensemble_member) int32 4B 1
    ensemble_member_id               (ensemble_member) |S27 27B ...
    latitude                         (projection_y_coordinate, projection_x_coordinate) float64 73kB ...
    longitude                        (projection_y_coordinate, projection_x_coordinate) float64 73kB ...
    month_number                     (time) int32 4B ...
    year                             (time) int32 4B ...
    yyyymm                           (time) |S64 64B ...
Dimensions without coordinates: bounds2
Data variables:
    time_bnds                        (time, bounds2) object 16B ...
    projection_y_coordinate_bnds     (projection_y_coordinate, bounds2) float64 2kB ...
    projection_x_coordinate_bnds     (projection_x_coordinate, bounds2) float64 1kB ...
    transverse_mercator              |S1 1B ...
    tas                              (ensemble_member, time, projection_y_coordinate, projection_x_coordinat
```

The cf-python version:

```
cfdump ukcp_rcm_test.nc
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(long_name=ensemble_member(1),
                    time(1), projection_y_coordinate(110), projection_x_coordinate(83)) degC
Cell methods    : time(1): mean
Dimension coords: long_name=ensemble_member(1) = [1] 1
                : time(1) = [1980-12-16 00:00:00] 360_day
                : projection_y_coordinate(110) =
                    [-87500.0, ..., 1220500.0] m
                : projection_x_coordinate(83) =
                    [-187500.0, ..., 796500.0] m
Auxiliary coords: long_name=ensemble_member_id(long_name=ensemble_member(1)) = [HadGEM3-GC3.05-r001i1p000
                : latitude(projection_y_coordinate(110), projection_x_coordinate(83)) =
                    48.8334819352858, ..., 60.670672077100846 degrees_north
                : longitude(projection_y_coordinate(110), projection_x_coordinate(83)) =
                    -10.009903208999626, ..., 5.266814195155682 degrees_east
                : long_name=month_number(time(1)) = [12] 1
                : long_name=year(time(1)) = [1980] 1
                : long_name=yyyymm(time(1)) = [198012] 1
Coord references: grid_mapping_name:transverse_mercator
```

Using domain specific (CF) knowledge

```python
import cf
import cfplot as cfp
import numpy as np

f = cf.read('ukcp_rcm_test.nc')
first = f[0]

# projection is independent of coordinates!
cfp.mapset(proj="UKCP", resolution="50m")

cfp.levs(-3, 7, 0.5)
cfp.setvars(grid_colour="grey")

cfp.con(
    first,
    lines=False,
    blockfill=True,
    # Centered over UK region with spacing of 1 each
    xticks=np.arange(14) - 11,
    yticks=np.arange(13) + 49,
)

```



cf.Field aware statistics

cf.Field.digitize

`Field.digitize(bins, upper=False, open_ends=False, closed_ends=None, return_bins=False, inplace=False)` [source]

Return the indices of the bins to which each value belongs.

Values (including masked values) that do not belong to any bin result in masked values in the output field construct of indices.

Bins defined by percentiles are easily created with the `percentile` method

Example:

Find the indices for bins defined by the 10th, 50th and 90th percentiles:

```
>>> bins = f.percentile([0, 10, 50, 90, 100], squeeze=True)
>>> i = f.digitize(bins, closed_ends=True)
```

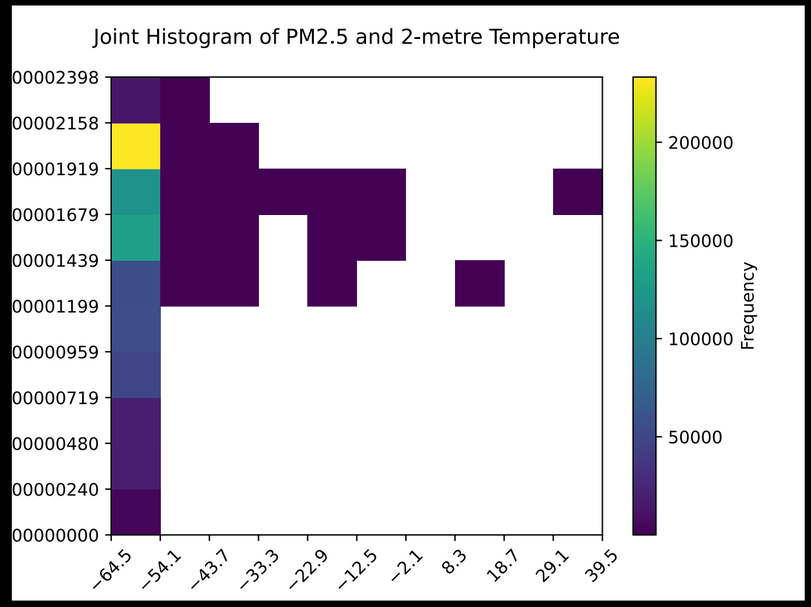
cf.histogram

`cf.histogram(*digitized, density=False)` [source]

Return the distribution of a set of variables in the form of an N-dimensional histogram.

The number of dimensions of the histogram is equal to the number of field constructs provided by the *digitized* argument. Each such field construct defines a sequence of bins and provides indices to the bins that each value of one of the variables belongs. There is no upper limit to the number of dimensions of the histogram.

```
# Digitize the PM2.5 mass concentration and 2-metre temperature fields

pm25_indices, pm25_bins = pm25_field.digitize(10, return_bins=True)
temp_indices, temp_bins = temp_field.digitize(10, return_bins=True)

# Create a joint histogram of the digitized fields:
joint_histogram = cf.histogram(pm25_indices, temp_indices)

# Get histogram data from the ``joint_histogram``:
histogram_data = joint_histogram.array
```



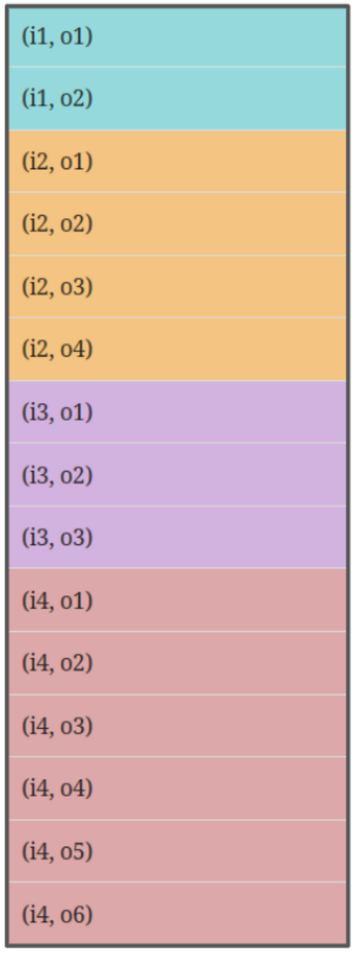
(See tutorial example 09 in the cf-python documentation)

Handling compression by convention

The CF conventions describe several methods of compression, but ideally handling compressed data should be trivial from a user perspective (though should care when lossy compression has been used; metadata should tell users exactly what lossy compression might have been applied).

Consider one example: A ragged array of station data:

Santander	Vienna	Paris	Reading
(i1, o1)	(i2, o1)	(i3, o1)	(i4, o1)
(i1, o2)	(i2, o2)	(i3, o2)	(i4, o2)
	(i2, o3)	(i3, o3)	(i4, o3)
	(i2, o4)		(i4, o4)
			(i4, o5)
			(i4, o6)



Comparison of how two libraries see these data

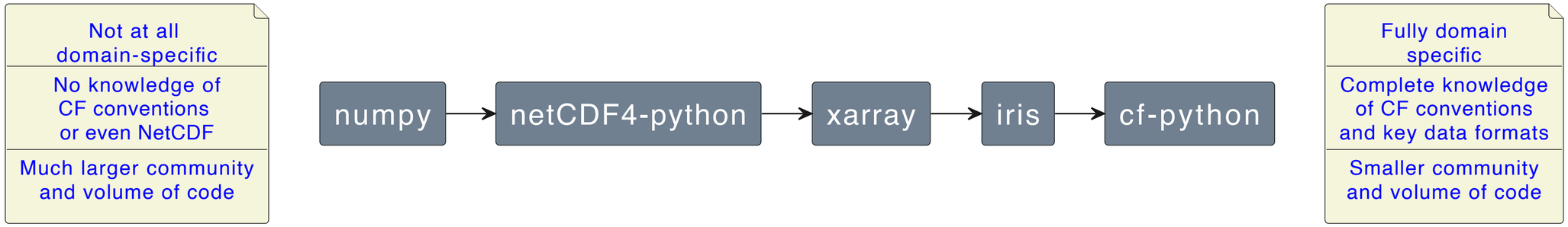
```
gg = cf.read('mydata.nc')
print(gg[0])
Field: specific_humidity (ncvar%humidity)
-----------------------------------------
Data            : specific_humidity(ncdim%station(4), ncdim%timeseries(9)) 1
Auxiliary coords: time(ncdim%station(4), ncdim%timeseries(9)) = 1969-12-30 00:0
                : latitude(ncdim%station(4)) = [43.46, ..., 51.46] degrees_nort
                : longitude(ncdim%station(4)) = [-3.82, ..., -0.98] degrees_eas
                : height(ncdim%station(4)) = [15.0, ..., 61.0] m
                : platform_name(ncdim%station(4)) = [Santander, ..., Reading]

Reading = q.subspace(platform_name='Reading')
print(Reading)
Field: specific_humidity (ncvar%humidity)
-----------------------------------------
Data            : specific_humidity(ncdim%station(1), ncdim%data(6)) 1
Auxiliary coords: time(ncdim%station(1), ncdim%data(9)) = 1969-12-30, ..., 1970-
                : latitude(ncdim%station(1)) = [51.46] degrees_north
                : longitude(ncdim%station(1)) = [-0.98] degrees_east
                : height(ncdim%station(1)) = [61.0] m
                : platform_name(ncdim%station(1)) = [Reading]
```

```
ds = xarray.open_dataset('mydata.nc')
ds
<xarray.Dataset> Size: 464B
Dimensions:        (station: 4, element: 12)
Coordinates:
    time           (element) datetime64[ns] 96B ...
    lat            (station) float64 32B ...
    lon            (station) float64 32B ...
    alt            (station) float64 32B ...
    platform_name  (station) <U9 144B ...
Dimensions without coordinates: station, element
Data variables:
    count          (station) int64 32B ...
    humidity       (element) float64 96B ...
Attributes:
    Conventions:  CF-1.12
    featureType:  timeSeries
```

Only cf-python is guaranteed to understand and implement the CF conventions (generally within weeks of new functionality being added to the conventions).

Domain Specific Languages for Analysis

There is a range of CF-netCDF domain-specificness across Python libraries (neglecting, for the moment Pandas et al):



Being domain-specific is not the be-all and end-all of analysis tools, and there will be times when you don't want it, but it does enable your software to do things for you which are otherwise (sometimes very) difficult for to do. For instance, what if you wanted to create a subspace of your data that was the Niño 3 region?

In netCDF4-python you'd need to manually:

- Identify which data axes correspond to latitude and longitude; find which elements of the corresponding 1-d latitude and longitude coordinates lie in the Niño 3 region; convert those elements to indices understood by numpy; apply those to the data; subspace the coordinates and other associated variables (such as cell measures), etc.

It's (relatively) easy to create CF compliant data with cf-python

Very straightforward.

See https://ncas-cms.github.io/cf-python/method/cf.Field.creation_commands.html#cf.Field.creation_commands

Essentially, you build up the bits using cf cf commands like

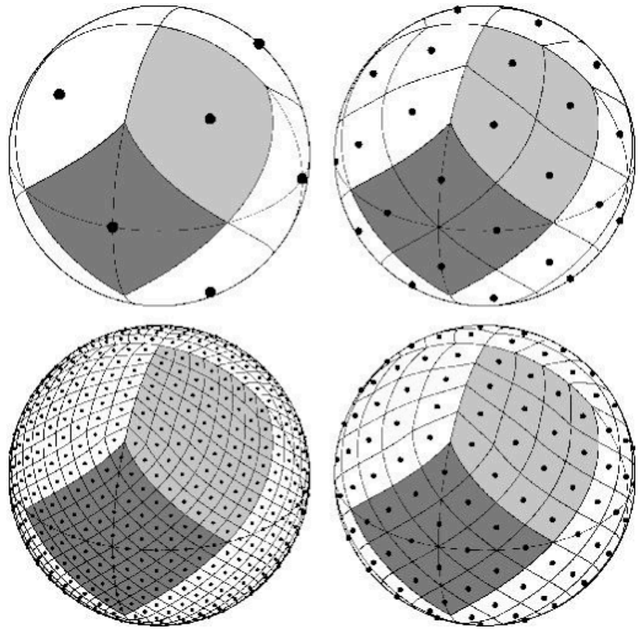
```
field = cf.Field()
field.set_properties({'Conventions': 'CF-1.12', 'project': 'research', 'standard_name': 'specific_humidity', 'units': '1'})
field.nc_set_variable('q')
data = cf.Data(0.007, 0.034, ..., 0.013, units='1', dtype='f8')
field.set_data(data)
```

and

```
# dimension_coordinate: time
c = cf.DimensionCoordinate()
c.set_properties({'units': 'days since 2018-12-01', 'standard_name': 'time'})
c.nc_set_variable('time')
data = cf.Data([31.0], units='days since 2018-12-01', dtype='f8')
c.set_data(data)
field.set_construct(c, axes=('domainaxis2',), key='dimensioncoordinate2', copy=False)
#
```

etc ... and you are guaranteed to have cf-compliant data. If you have created your field by manipulating other fields, you don't have to do anything, cf-python writes cf-compliant data by default.

Coming soon: HealPix (going into CF soon)



HEALPix:Hierarchical Equal Area isoLatitude Pixelation of a sphere.

- This pixelation produces a subdivision of a spherical surface in which each pixel exactly covers the same surface area of the globe.
- HEALPix resolutions are restricted to resolutions that nest hierarchically, i.e. one cell at a given resolution always contains exactly four whole cells at next higher resolution. This means you can easily move between resolutions by simple unweighted averaging/summing/broadcasting, rather than expensive and (to some extent) inaccurate regridding techniques.
- The cells are completely defined by mathematical formulae (amount, shape, and location), so you don't need to to store coordinates and bounds.
- Why isolatitude? To optimise spherical harmonic and zonal calculations.

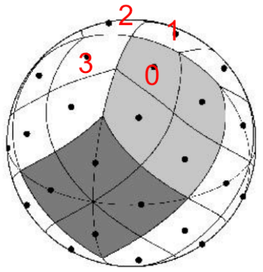
HEALPix: Why "soon" why not now?

- Issue raised on the CF conventions website in March 2023.
- We (CMS) committed to shepherd this through as part of the EXPECT proejct.
- As of 17th November 2025, 199 detailed technical comments on the issue (addressing things like the indexing options).
- A pull request was opened in September this year (more than two years later). There have so far been 34 comments on that as part of the review process.
- Getting this sort of thing right so it doesn't change is hard!

Indexing schemes

ring

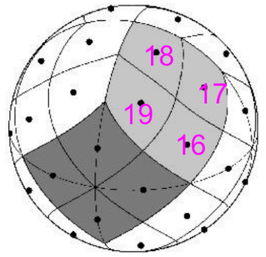
sorts pixels along isolatitude rings from north to south, resulting in *consecutive pixels share a latitude*

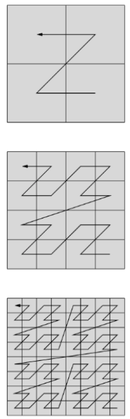


nuniq

effectively concatenates nested indices from lower to the higher refinement levels, resulting in *geographic proximity for consecutive pixels within a refinement level but not across different refinement levels:*

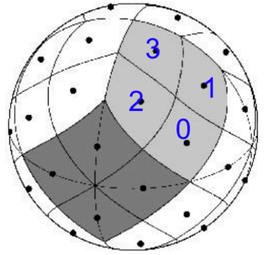
$i = 4^{R+1} + i_{nested}$

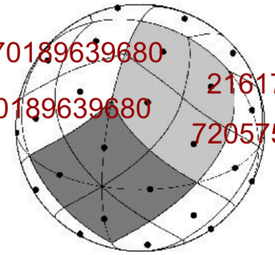




nested

sorts pixels along the Z-order curve within each of the 12 base level, resulting in *geographic proximity for consecutive pixels*





zuniq

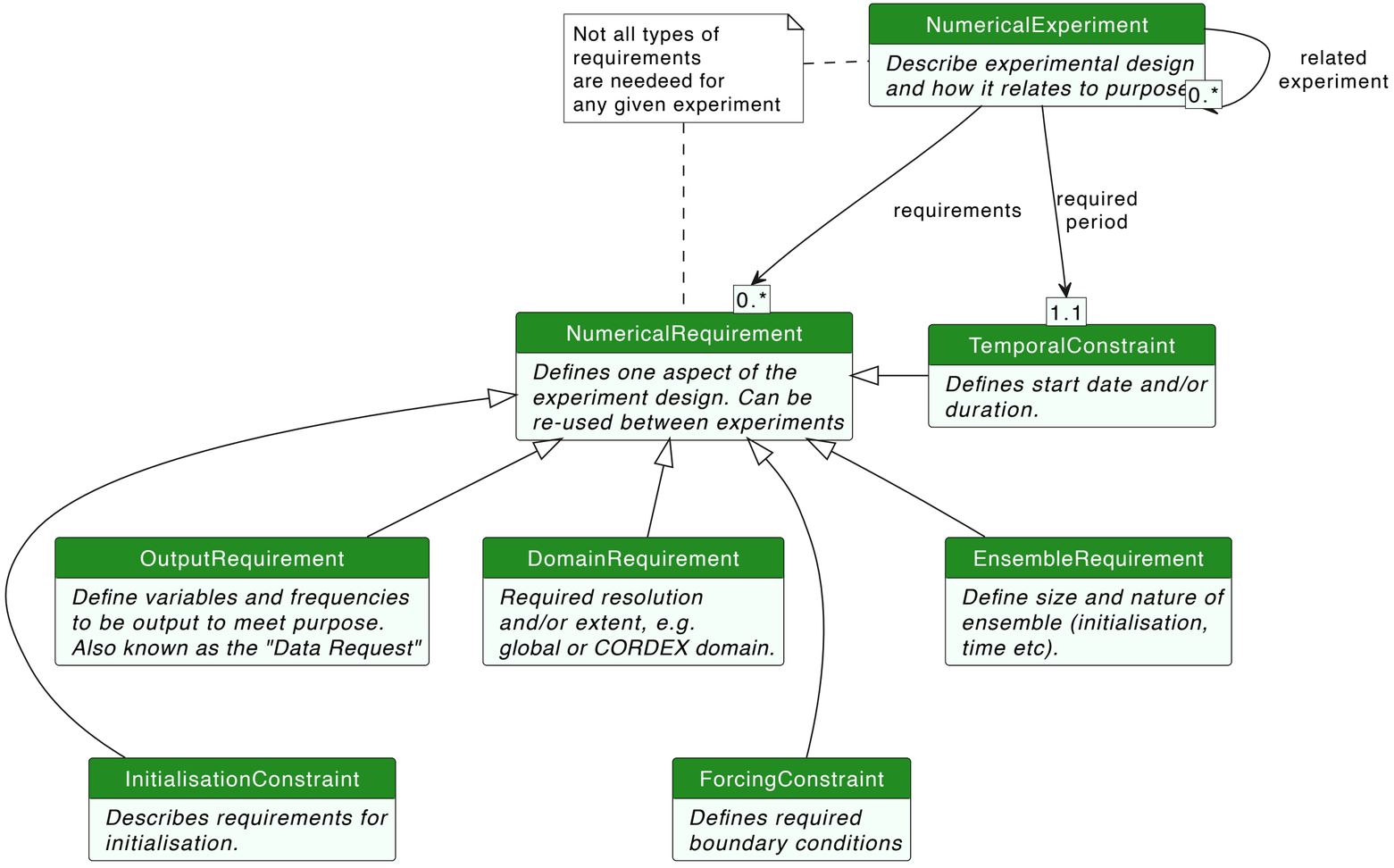
sorts pixels of all refinement levels along the Z-order curve resulting in *geographic proximity for consecutive pixels, regardless of their refinement level:*

$i = 4^{29-R}(2i_{nested} + 1)$

"consecutive pixels" means consecutive in the data array

DOCUMENTATION: B-METADATA

Numerical Experiments



ESDOC V2; BNL April 2023

Model Workflow

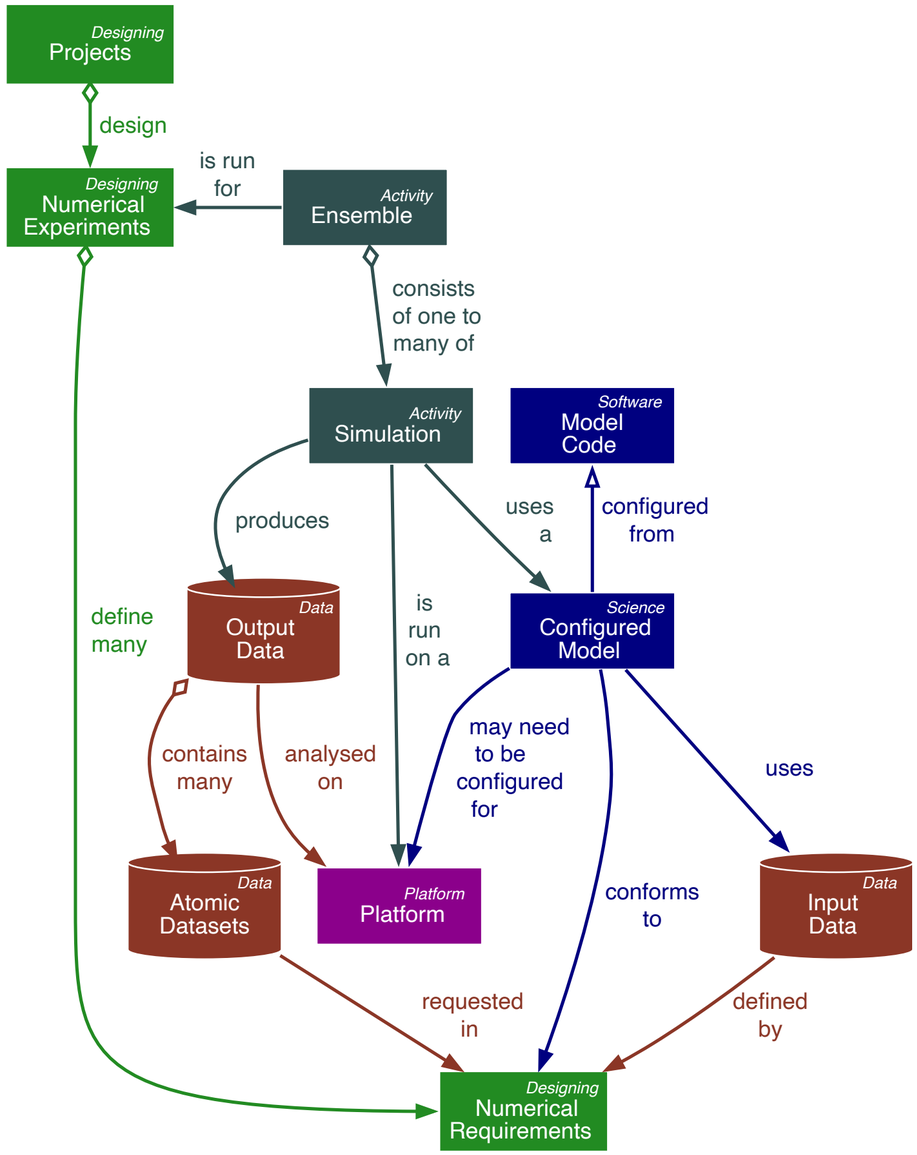
A model is configured to meet the numerical requirements of one or more experiments and produces simulations which need to be analysed and shared (and sometimes curated). What data is produced needs to be agreed a priori often as part of a data request.

> *"the CMIP6 models saying X ..."*

They do no such thing!

The same model running a different experiment might get a different result, and folks are often confused by the similarities and difference between configured models.

The advent of models calibrated for different purposes will cause even more problem in this space, never mind AI models, where the training dataset is as important as the model!



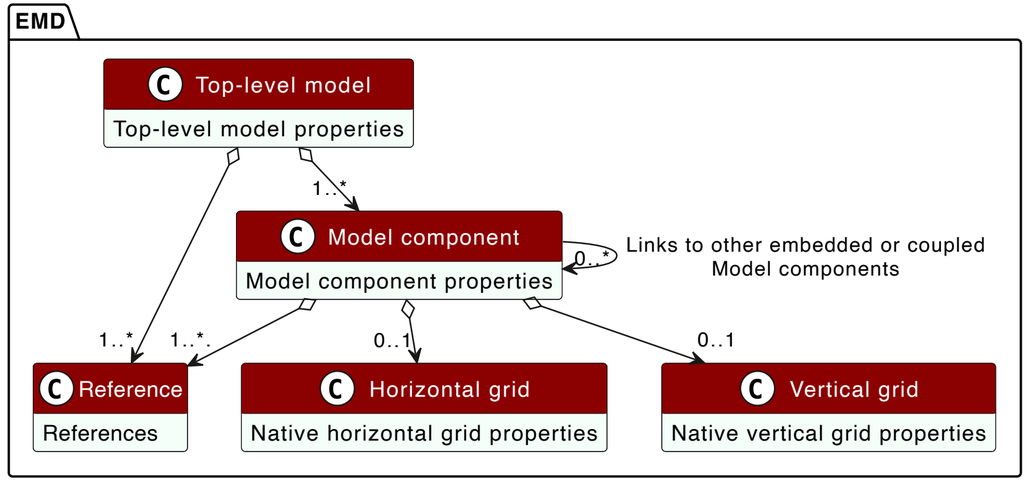
Essential Model Documentation

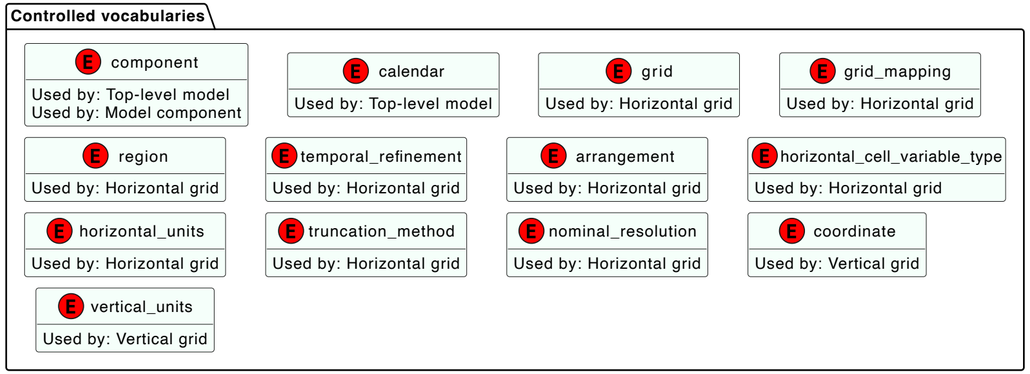
For CMIP7 we are working on defining and collecting "Essential Model Documentation" - all contributors will be required to provide model definitions before uploading to the CMIP7 archive.

Models are described by building out descriptions of components using a combination of free text (e.g. description) and controlled vocabularies (CVs).

CVs allow one to search and build comparison tables.

There will be a one-to-one correspondence between an EMD model description and the model "source" name.





EMD Example - Land Surface Description

- name: CLM4
- family: CLM
- description: The model represents several aspects of the land surface including surface heterogeneity ... The surface variables and fluxes required by the atmosphere are obtained by averaging the subgrid quantities weighted by their fractional areas.
- references
  - citation: Ke, Y., Leung,et. al.: Development of high resolution land surface parameters for the Community Land Model, Geosci. Model Dev., 5, 1341–1362, https://doi.org/10.5194/gmd-5-1341-2012, 2012.
  - doi: https://doi.org/10.5194/gmd-5-1341-2012, 2012
- embedded_in: atmosphere
- native_horizontal_grid:
  - grid: regular_latitude_longitude
  - grid_mapping: latitude_longitude
  - region: global
  - arrangement: arakawa_c
  - temporal_refinement: static
  - resolution_x: 1.25o resolution_y: 0.9
  - horizontal_units: degree
  - n_cells: 55296
  - resolution_range_km: 100.0, 170.1
  - mean_resolution_km: 139.5
  - nominal_resolution: 100 km
- native_vertical_grid
  - description: Vegetated, wetland, and glacier landunits have 15 vertical layers.
  - Lakes have 10 layers. Snow can have up to 5 layers.
  - coordinate: depth

Data Reference Syntax - Essential B-Glue

THINGS WE USE FOR PROVENANCE AND ORGANISATION

A DRS (some parts can be used in a filename):

- `<institute>` (e.g., MOHC)
- `<source_id>` (e.g., HadCM3)
- `<experiment>` (e.g., rcp45)
- `<frequency>` (e.g., mon, day)
- `<realm>` (e.g., atmos, ocean)
- `<variable_id>` (e.g., tas)
- `<variant_label>` (e.g., r1i1p1)
- `<version>` (e.g., v20140318)

BEYOND THE DRS

Information to go *in* the file files:

- `<source>` (e.g. and EMD description)
- `<standard_name>` (eg air_temperature)
- `<nominal_resolution>` (e.g. 100 km)
- `<further_info_url>` (should be a link to explain the ensemble identifier)
- `<variant_info>` some exaplanation of the ensemble member differences
- `<tracking_id>` should be a UUID we can use to match files to metadata
- `<branch_time_in_parent>` (if necessary)

A Data Reference Syntax allows us to use "faceted browse", it's not just (and sometimes, not even) about a directory struture!

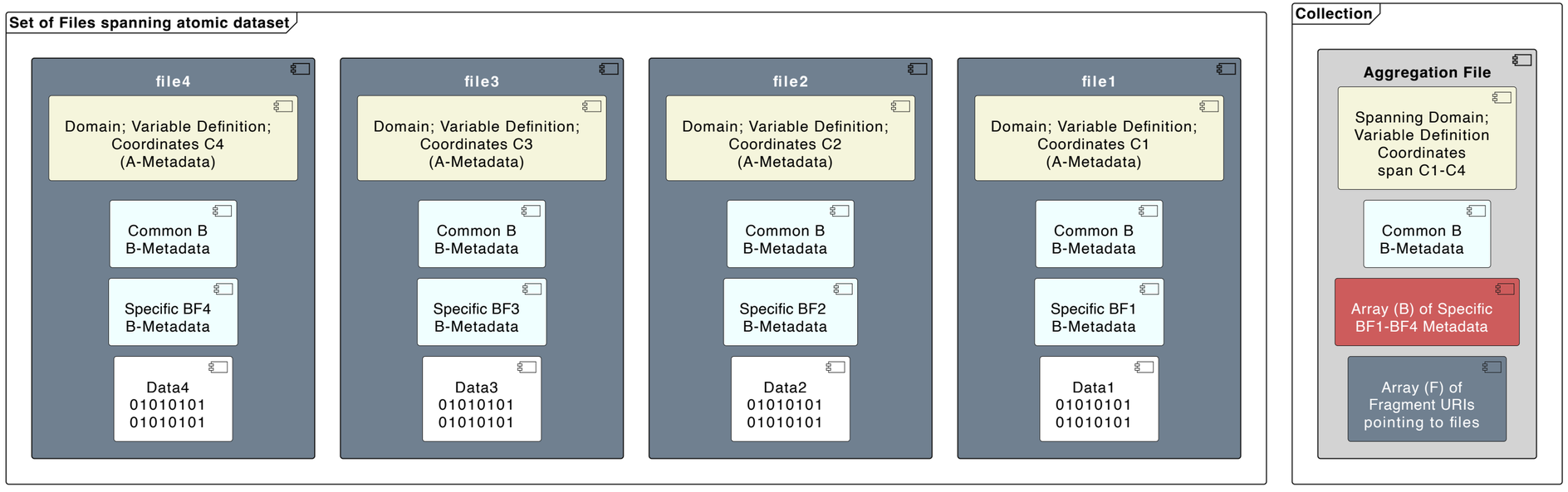
ATOMIC DATASETS AND AGGREGATION (MIXING A AND B)

Climate Forecast Aggregation

One file per variable, and not too big (as will see) is all very well, but too many files is problematic for the analyst (and the data manager).

Most tools implement the notion of an "aggregated dataset" but experience has told us that such aggregations need to be binary.

Some have invented new tools, but we have an aggregation convention (now part of CF) that exploits NetCDF itself.



An aggregation file indexes the content of files by constructing a spanning coordinate domain, and extracting b-metadata

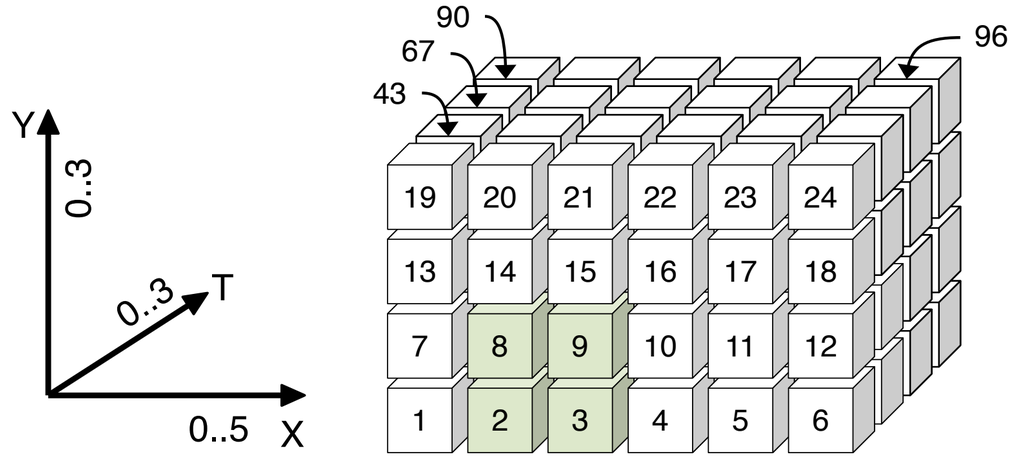
CFA - Some key capabilities

- Pointers to files can be local or remote (e.g. to files on tape or on a remote object store)
- CFA files behave just like other CF files, we can subset etc on coordinates, without touching the data.
  - This means we can use the coordinates to identify files of interest, and for example, only grab those files back from tape.
- Standards compliant (as of this month) - changes to CFA would have to go through a process.
- Full implementation in `cf-python`, you can use and create CFA files as if they were standard files. From a user perspective you ought not be able to tell if `cf-python` is accessing chunks from a file itself, or a remote file (presuming it is actually accessible).
  - Extra support to ascertain which files are involved in a particular subsetting operation.
- Partial implementation in `xarray` (courtesy of Dan Westwood) - complete support for reading CFA, so that in `xarray` you can manipulate CFA files as if they were one file.

WHY YOU SHOULD CARE ABOUT CHUNKING

Chunking 101 - Writing Data

Consider a simple model writing a simple 6 x 4 lon-lat grid for four timesteps to a NetCDF4 file.

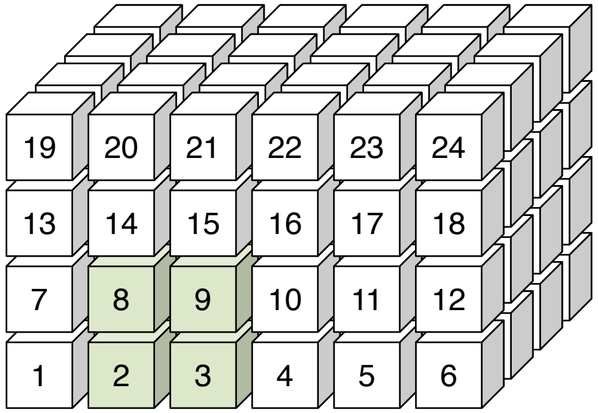


In practice what gets written to the file in this case would likely be four chunks, one per timestep, and these chunks would be distributed through the file if there were multiple variables in teh file: $(T^{C=1}\_t = 0, u^1\_0, v^1\_0), (T^2\_1, u^2\_1, v^2\_1), \ldots$ and there will be a chunk index written to the file as well (so we know how to find the chunks for a given variable).

Normally, each chunk will be compressed as well. In reality with a bigger grid, we might have multiple chunks per XY slice per timestep, and the index would be distributed through the file too!

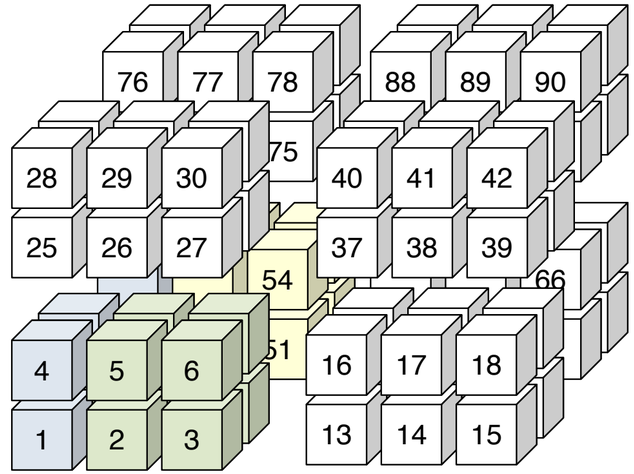
Chunking 101 - Reading Data

Consider accessing a time series of the colour region for the duration:



Access involves reading the entire file.

But we can rechunk the file after it has been written:



Now accessing that region involves half the number of reads!

With the rechunked file, lots of different read patterns are more efficient. This REALLY matters...

Storage 101

POSIX FILE SYSTEMS

("Normal" file systems for most of us!)

- Hierarchical directories & paths (`/dir/file`)
- Mutable files — random read/write & partial updates
- Access via OS/system calls (`open`, `read`, `write`)
- Strong filesystem semantics (rename, permissions, locks)
- Low-latency when local; used for DBs, HPC, transactional apps

OBJECT STORES

- Flat/key-based namespace (`bucket/key`) — prefixes emulate folders
- Objects are immutable; updates = rewrite whole object
- Access via HTTP APIs (PUT/GET/DELETE) (often remote)
- Designed for massive scale, durability, and distribution
- Ideal for backups, ML data lakes, media, cloud-native storage.

For ESGF2 (CMIP7) we will be able to access POSIX files via https servers in the same way as we access object stores & that will have high latency like object stores!

Remote access to HDF5 (NetCDF4) and Zarr

For remote access we want "lazy loading":

- open file, and inspect data variables, should be cheap, no data loaded.
- accessing data with `x[5:10,5:10]` should only load the relevant chunks, which means we need the index first.
- We want to get chunks in parallel!

	HDF5	Zarr
Chunks	Chunks of variables can be interleaved through a file.	Each chunk is a a file to V2. Multiple chunks in a file for V3.
Chunk Index	Chunk indexes can be spread throughout a file	Chunk index is a separate file.
Remote Inspection prior to lazy loading	Very slow if chunk indexes distributed through the file (which is often the case). **But it can be repacked**.	Very quick, download chunk index.
Lazy loading	Very slow if poorly chunked (which is often the case). **But it can be rechunked**.	Very slow if poorly chunked (but since nearly nothing is born Zarr, **it has nearly always been rechunked**).
Parallel Access in Python	Only possible with `pyfive` : **a new(ish) tool** we are responsible for.	Python native.

Putting all this together with real data

Consider this CMIP6 file (ridiculously big ~19GB):

```
Field: eastward_wind (ncvar%uas)
Data           : eastward_wind(time(292192), latitude(143), longitude(144)) m s-1
Cell methods   : area: mean time(292192): point
Dimension coords: time(292192) = [1870-01-01 03:00:00, ..., 1970-01-01 00:00:00] gregorian
               : latitude(143) = [-90.0, ..., 90.0] degrees_north
               : longitude(144) = [0.0, ..., 357.5] degrees_east
               : height(1) = [10.0] m
Cell measures  : measure:area (external variable: ncvar%areacella)
```

And if we examine a little bit of the file layout as retrieved by `p5dump -s` (a utility in our new package):

```
float64 time(time) ;
               time:axis = "T" ;
               time:standard_name = "time" ;
               time:_n_chunks = 292192 ;
               time:_chunk_shape = (1,) ;
               time:_btree_range = (31808, 19854095942) ;
               time:_first_chunk = 9094 ;
float32 uas(time, lat, lon) ;
               uas:standard_name = "eastward_wind" ;
               uas:_Storage = "Chunked" ;
               uas:_n_chunks = 292192 ;
               uas:_chunk_shape = (1, 143, 144) ;
               uas:_btree_range = (28672, 19854809382) ;
               uas:_first_chunk = 36520 ;
               uas:_compression = "gzip(4)" ;
```

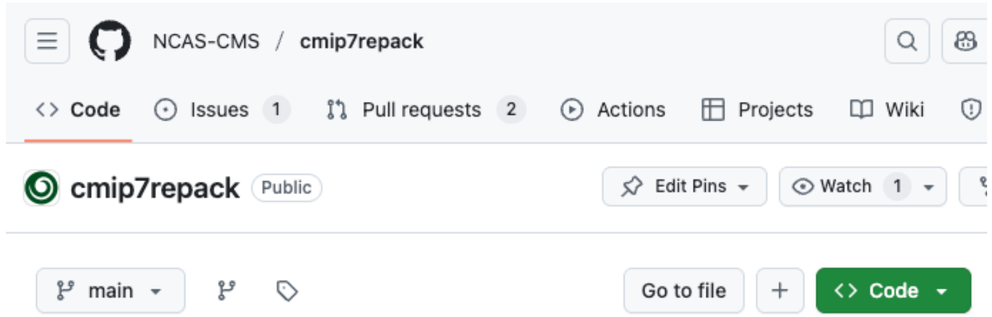
It's dreadfully slow manipulating this file on POSIX, but you can wait for the heat death of the universe before you can remotely access even one timestep from this file!

Which brings us to cmip7-repack

cmip7repack is a bespoke CMIP7 command-line tool which can be used by the modelling groups, prior to dataset publication, to "repack" their files in such as way as to improve their read-performance in the CMIP7 archive (note that CMIP7 datasets are written only once, but read many times).

Full details are available with `cmip7repack –h`.

ON GITHUB:



The relevant metadata for the same file:

```
float32 uas(time, lat, lon) ;
                uas:standard_name = "eastward_wind" ;
            uas:_n_chunks = 5844 ;
                uas:_chunk_shape = (50, 143, 144) ;
                uas:_btree_range = (18591, 347871) ;
                uas:_first_chunk = 41742441 ;
                uas:_compression = "gzip(4)" ;
```

Remote Access Patterns using filesystem middleware

REMOTE S3

```
from s3fs import S3FileSystem

fs_options = {
    'default_cache_type': 'readahead',
    'default_block_size': blocks_MB * 2**20
    'endpoint_url': server_url}
}

fs = S3FileSystem(anon=True,**fs_options)

with fs.open(filename,'rb') as as s3file:
    with pyfive.File(s3file, 'r') as f:
        dset = f['varname']
```

REMOTE HTTPS

```
import fsspec

fs_options = {
    'cache_type': 'readahead',
    'block_size': blocks_MB * 2**20
}

fs = fsspec.filesystem("https")

with fs.open(server_url + filename,'rb',**fs_options) as remote_file:
    with pyfive.File(remote_file,'r') as f:
        dset = f['varname']
```

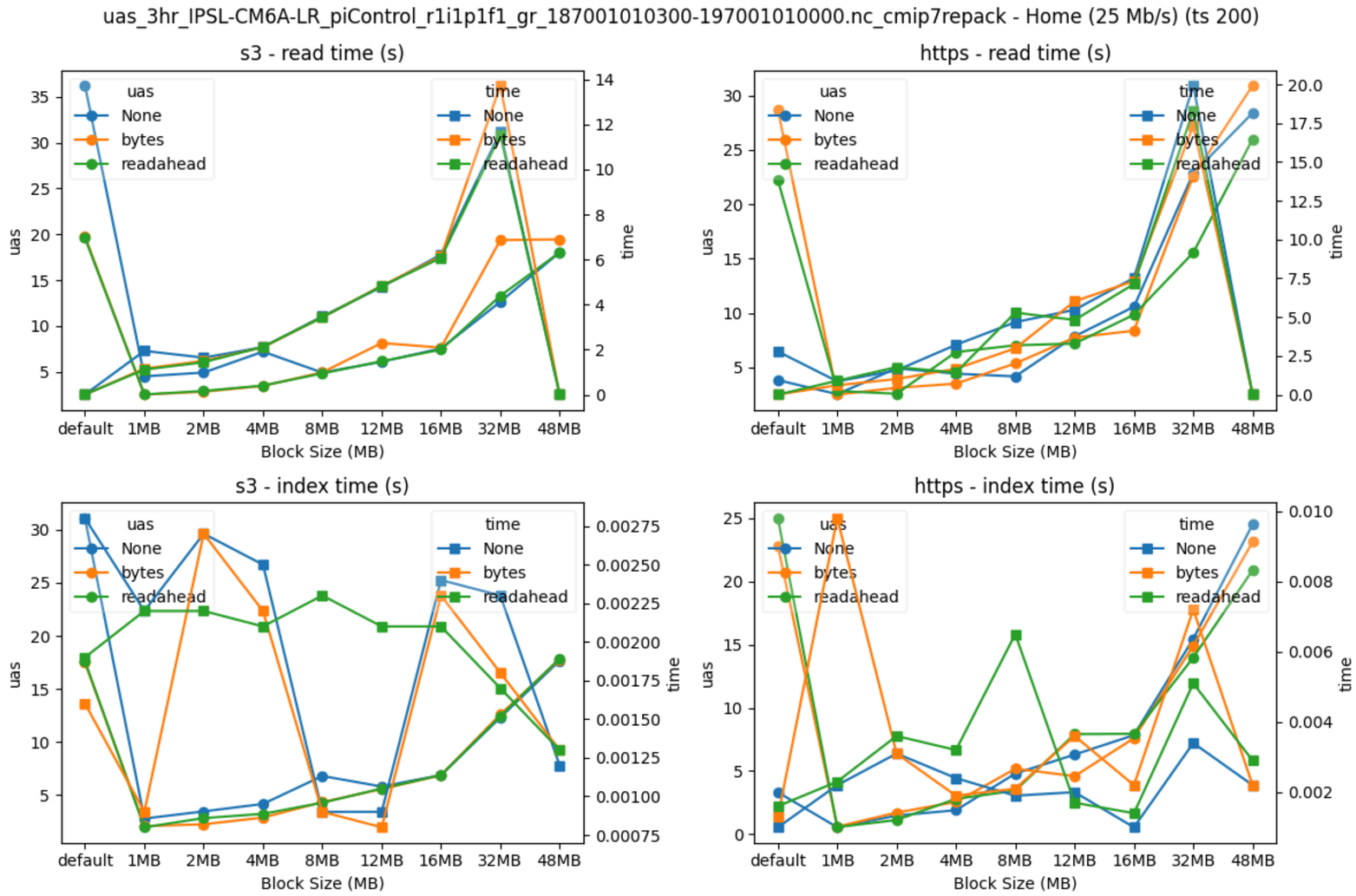
You do need to care about the cache type and the block size, particularly if you are not "close" to the server!

Accessing real data from home at 25 Mbit/s

Slow internet is useful to show what is going on!

- Three different types of middleware caching.
- left hand axes data
- right hand axes the time index itself
- top panels: data; bottom panels: building index

Zarr assumptions and defaults are bad!



Py[*five*]

- Invented by Jonathan Helmus prior to 2018. Core implementation.
- In 2024 we discovered it as we realised (as par of PyActiveStorage development) we needed pure-python, thread-safe, HDF5 reader
- As it was it didn't support what we needed, so we added lazy loading and lots of other goodies, and with the agreement of the original author, took over package management.
- In doing so, we also realised that having our own HDF5/NetCDF4 reader solved a lot of other problems!
- Now the maintainers of h5netcdf and xarray are implementing pyfive support!

`pyfive` provides a pure Python HDF reader which has been designed to be a thread-safe drop in replacement for h5py with no dependencies on the HDF C library. It aims to support the same API as for reading files. Cases where access to a file uses a feature that is supported by the high-level `h5py` interface but not `pyfive` are considered bugs and should be reported in our Issues. Writing HDF5 is not a goal of pyfive and portions of the `h5py` API which apply only to writing will not be implemented.

Note

While `pyfive` is designed to be a drop-in replacement for `h5py`, the reverse may not be possible. It is possible to do things with `pyfive` that will not work with `h5py`, and `pyfive` definitely includes *extensions* to the `h5py` API. This documentation makes clear which parts of the API are extensions and where behaviour differs *by design* from `h5py`.

The motivation for `pyfive` development were many, but recent developments prioritised thread-safety, lazy loading, and performance at scale in a cloud environment both standalone, and as a backend for other software such as cf-python, xarray, and h5netcdf.

https://pyfive.readthedocs.io

Before we leave chunking: cfs3 cftools

https://github.com/ncas-cms/cfs3

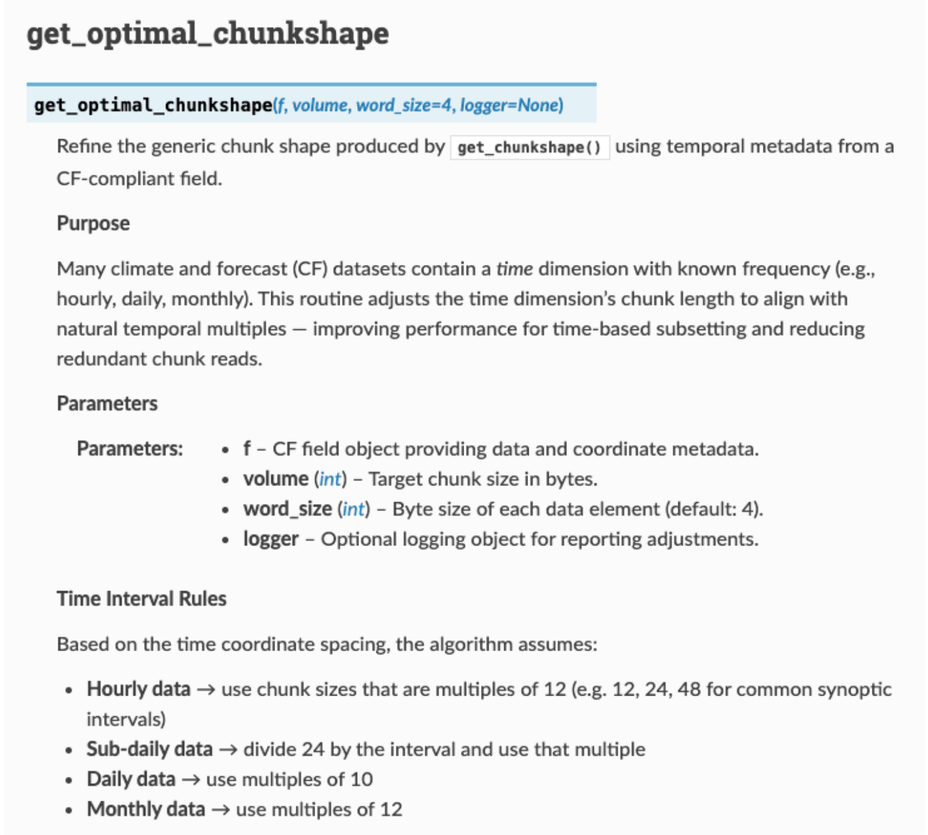
A library of tools to carry out a "lightweight" "cmor-isation" of existing data and in doing so:

- add user-defined DRS information ('B-metadata')
- rechunk appropriately
- split multi-variable files into single-variable files

Also includes tools to:

- Upload files *and their metadata* to object stores.

The intention is that users build their own workflow with these tools, rather than being a drop-in solution.



get_optimal_chunkshape

get_optimal_chunkshape(*f, volume, word_size=4, logger=None*)

Refine the generic chunk shape produced by `get_chunkshape()` using temporal metadata from a CF-compliant field.

Purpose

Many climate and forecast (CF) datasets contain a *time* dimension with known frequency (e.g., hourly, daily, monthly). This routine adjusts the time dimension's chunk length to align with natural temporal multiples — improving performance for time-based subsetting and reducing redundant chunk reads.

Parameters

Parameters:
- **f** – CF field object providing data and coordinate metadata.
- **volume** (*int*) – Target chunk size in bytes.
- **word_size** (*int*) – Byte size of each data element (default: 4).
- **logger** – Optional logging object for reporting adjustments.

Time Interval Rules

Based on the time coordinate spacing, the algorithm assumes:

- **Hourly data** → use chunk sizes that are multiples of 12 (e.g. 12, 24, 48 for common synoptic intervals)
- **Sub-daily data** → divide 24 by the interval and use that multiple
- **Daily data** → use multiples of 10
- **Monthly data** → use multiples of 12

cfs3 has two components - library tools (cftools), and an s3 command line interface (s3view). cfs3 is very much a prototype right now!

cfs3 - Entering s3view

```
(cfs3) bnl28@MX6H7D9YGP cfs3 % s3view hrs3
You have entered a lightweight management tool for organising "files" inside an S3 object store
WARNING: Found unexpected S3 API S3v2 for gcs in configuration file /Users/bnl28/.mc/config.json
Buckets:  hrcm
hrs3> cb hrcm
Bucket: hrcm contains 73.2TiB in 11889 files/objects.
hrs3> help

Documented commands (use 'help -v' for verbose/'help <topic>' for details):
================================================================================
cb  cflist   help  lb   loglevel  match  mv      pwd   rm
cd  drsview  ipy   loc  ls        mb     p5dump  quit  tag

hrs3> help ls
Usage: ls [-h] [-l] [-s] [-w [WIDTH]] [-m] [-t] [-d] [-o [ORDER]] [-n MAX_NUMBER] [path]

List the files and directories in a bucket, potentially with a wild card.

positional arguments:
  path                  Path should be a valid path in your current bucket and location, possibly with a wildcard.

optional arguments:
  -h, --help            show this help message and exit
  -l, --long            Same as -s -d -m
  -s, --size            Tell us about size
  -w, --width [WIDTH]   width of display for standard output
  -m, --metadata        Show user metadata
  -t, --tags            Show tags
  -d, --date            Show dates
  -o, --order [ORDER]   Order by size|date
  -n, --max_number MAX_NUMBER
                        Limit the number of files returned

hrs3> ls -m -n 1 *epv*
Listing 1 files/objects (2.0GiB)
epv_HadGA7EA-N1280_highresSST-present_r1i1p1f1_6hrPt_1979-03-01T0600_N120.nc
   {standard-name: ertel_potential_vorticity, long-name: ERTEL POTENTIAL VORTICITY THETA SURF, domain: air_potential_temperature(1), latitude(1921), longitu
de(2560), time(120), chunk-shape: [12, 1, 113, 2560]/(120, 1, 1921, 2560), domain-name: global, ensemble-type: Perturbed stochastic physics, experiment: hig
hresSST-present, further-info-url: https://github.com/ncas-cms/further_info/HRCM/highresSST-present, grid-label: gn, grid-name: UM/N1280, institution: NCAS,
 mip-era: CMIP6, mip: HighResMIP, model-configuration: GA7.2.1-Easy Aerosol L85 / GL8 , nominal-resolution: 10 km, notes: Not formally part of HighResMIP bu
t conforming to the protocol, parentage: Follows one year of u-cd936 then 1 year of u-cf432, processing: pp_to_nice_netcdf:2024-08-29., project: HRCM, realm
: atmos, runid: u-ch330, source-id: HadGA7EA-N1280, source-type: AGCM, tracking-id: b586432c-27db-4a88-b4ac-e160ed7645a5, variant-label: r1i1p1f1, version:
UM11.6}
```

cfs3 - Using s3view and the metadata

```
hrs3> help drsview
Usage: drsview [-h] [-c COLLAPSE_LIST] [-u] [-s SELECT] [-o OUTPUT] [path] [drs]

Extract DRS components at location

positional arguments:
  path                  Path should be a valid path in your current bucket and location, possibly with a wildcard.
  drs                   Comma seperated string of DRS components to be used for contents (ignored for metadata option)
                        default = Variable,Source,Experiment,Variant,Frequency,Period,nFields

optional arguments:
  -h, --help            show this help message and exit
  -c, --collapse_list COLLAPSE_LIST
                        Comma seperated string of DRS terms where short lists are wanted
  -u, --use_metadata    build drs-like view from metadata
  -s, --select SELECT   Specfiy DRS component selections as key=value (multipe -s allowed) and return listing
  -o, --output OUTPUT   Default output is a DRS view, alternative is "list" view.

hrs3> drsview -c Period
output options are "drs" or "list" (you said "drs")
Variable : ['epv', 'hfnpuv', 'hus', 'pr', 'prlsprof', 'prrc', 'ps', 'psl', 'rlut', 'sfcWind', 'ta', 'tmfurg', 'tmfvrg', 'ua', 'uas', 'va', 'vas', 'wa', 'zg'
, 'zg500']
Source : ['HadGA7EA-N1280']
Experiment : ['highresSST-present']
Variant : ['r1i1p1f1', 'r2i1p1f1', 'r3i1p1f1']
Frequency : ['1hr', '1hrPt', '6hr', '6hrPt']
Period : [1979-03-01T0030 ... 2005-12-01T0100] (len=1115)
nFields : ['N120', 'N719', 'N720']
hrs3> drsview -s Variable=epv -c Period
output options are "drs" or "list" (you said "drs")
Variable : ['epv']
Source : ['HadGA7EA-N1280']
Experiment : ['highresSST-present']
Variant : ['r1i1p1f1', 'r2i1p1f1', 'r3i1p1f1']
Frequency : ['6hrPt']
Period : [1979-03-01T0600 ... 2000-12-01T0600] (len=262)
nFields : ['N120']
```

(Still very much a prototype)

APPLICATIONS

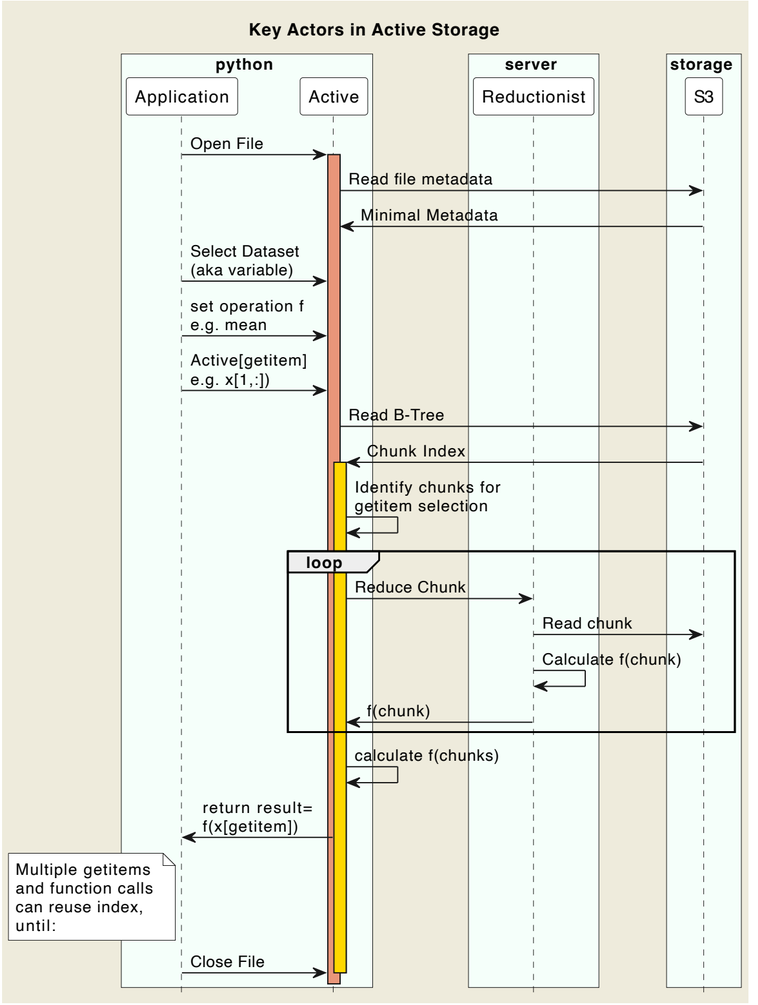
PyActiveStorage

Moving data is expensive. Storing extra copies of data is expensive. Modern storage systems have a lot of compute in them.

We have borrowed an idea from HPC and implemented a server system for object storage (`Reductionist`, with StackHPC, a UK SME) and for POSIX (with DDN, a global HPC storage company) which can do compute on NetCDF and HDF5 data held in that storage.

Essentially we can do *reductions* on any subset of the data by sending reduction requests and getting results, instead of sending data requests and doing reductions on our client.

Series of projects funding this, with exciting results!



Testing PyActiveStorage

EXPERIMENT

Loop over 40 timesteps of N1280 (10km) global 3D fields and perform hemispheric mean on one level. Do this on the JASMIN LAN (JAS), from the University of Reading LAN (UREAD), from a 40 Mb/s home broadband (HOM), and from 100 MB/s home broadband in New Zealand (NZ).

HEADLINE RESULTS

- No value for using reductionist on JASMIN LAN
- Remote uni access using active storage roughly as fast as JASMIN LAN (and 4X compare to normal method).
- Using multiple threads and 1 MB caching gives great results: useful even from New Zealand.
- Remote access impractical from some locations without active storage.

SELECTED RESULTS

(time in seconds, averaged)

Where	Method	ST1-MB	ST-5MB	MT-1MB	MT-5MB
JAS	active	67	69	23	26
UREAD	active	80	107	29	46
HOM	active	108	161	48	110
NZ	active	327	411	103	192
JAS	normal	86	103	25	40
UREAD	normal	264	564	139	520
HOM	normal	909	2783	666	2287
NZ	normal	1211	2389	668	DNF

- Different cache sizes (all readahead) 1MB, 5MB
- Single versus multiple threading ST, MT
- Compare Active versus Normal

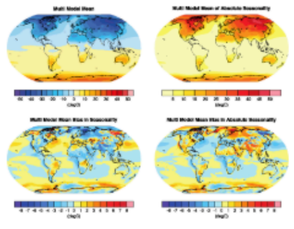
Intro to ESMValTool

ESMValTool is **community driven** open-source framework that automates and standardizes the **analysis, diagnostics, and evaluation** of Earth System Models (ESMs).

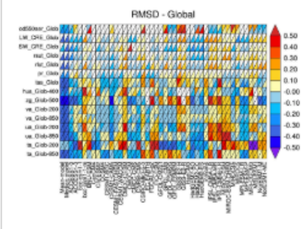
Key Features	What it is used for
Large library of diagnostics (atmosphere, ocean, land, extremes, carbon cycle)	IPCC-style Analysis: multi-model intercomparison.
Reproducible workflows via YAML “recipes”	Generic Climate Model Evaluation - bias assessment etc.
Standardized pre-processing (CMORization, regridding, masking, units)	Deriving climate indices and metrics
Python & R based diagnostic modules	Routine analysis for model development groups
Parallel execution for large datasets	

Goal: Make climate analysis *transparent, reproducible, and shareable* across institutions.

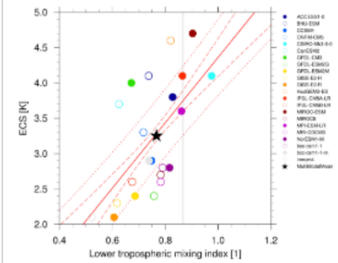
ESMToolExamples



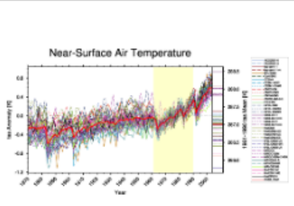
recipe_flato13ipcc.yml



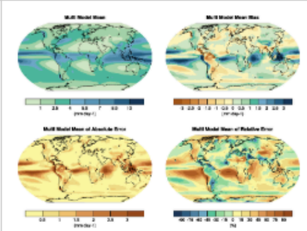
recipe_perfmetrics_CMIP5.yml



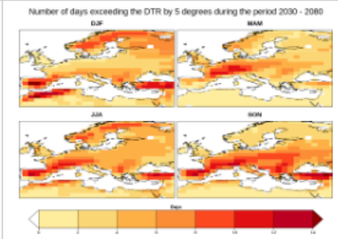
recipe_ecs_scatter.yml



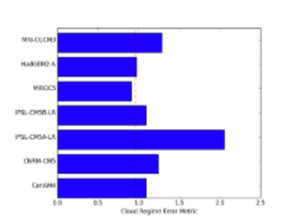
recipe_flato13ipcc.yml



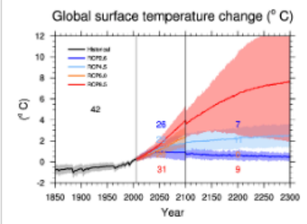
recipe_flato13ipcc.yml



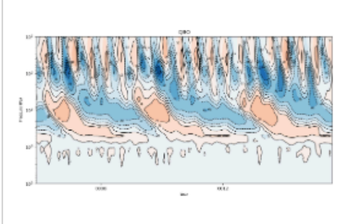
recipe_diurnal_index.yml



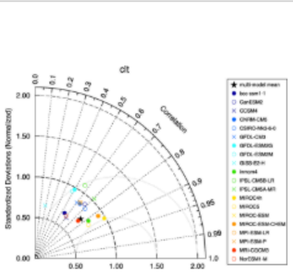
recipe_crem.yml



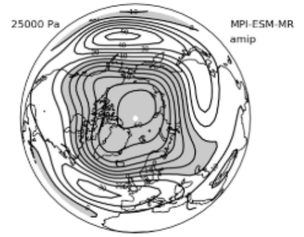
recipe_collins13ipcc.yml



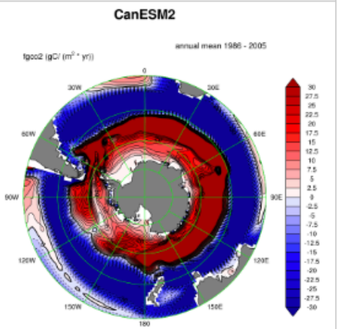
recipe_autoassess_stratosphere.yml



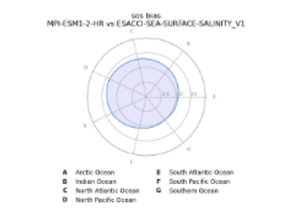
recipe_lauer13jclim.yml



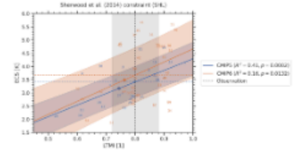
recipe_zmnam.yml



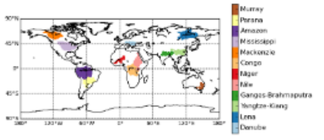
recipe_russell18jgr.yml



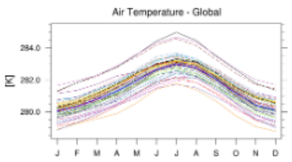
recipe_sea_surface_salinity.yml



recipe_schlund20esd.yml

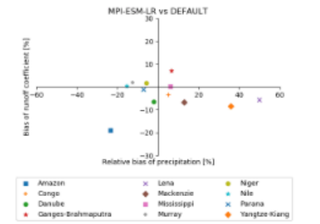


recipe_runoff_et.yml



recipe_perfmetrics_CMIP5.yml

recipe_spei.yml



recipe_runoff_et.yml

Maintaining ESMValTool

RECENT DEVELOPMENTS

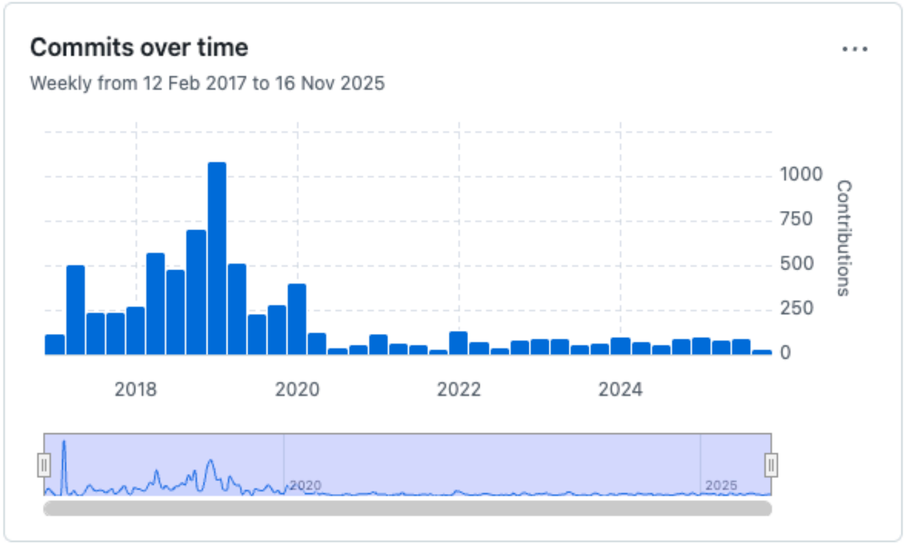
File and Metadata Support

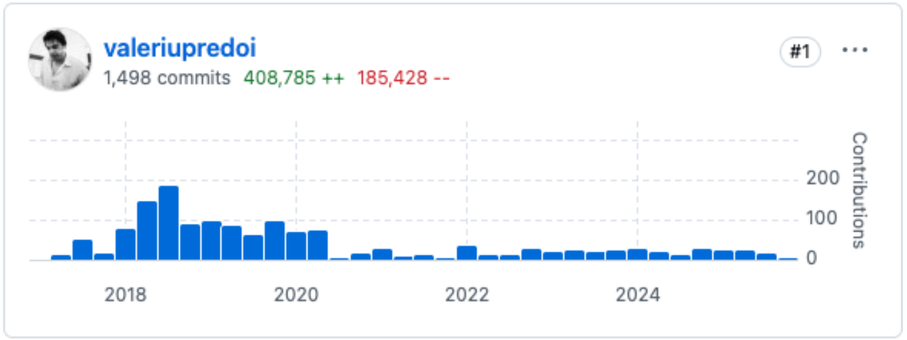
- Implementing Zarr support
- ESF2 support (Intake-esgf catalogues)

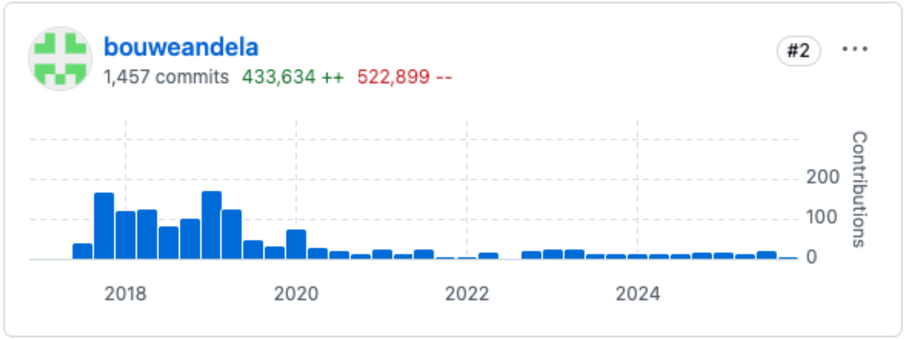
UPCOMING DEVELOPMENTS

Remote Calculations

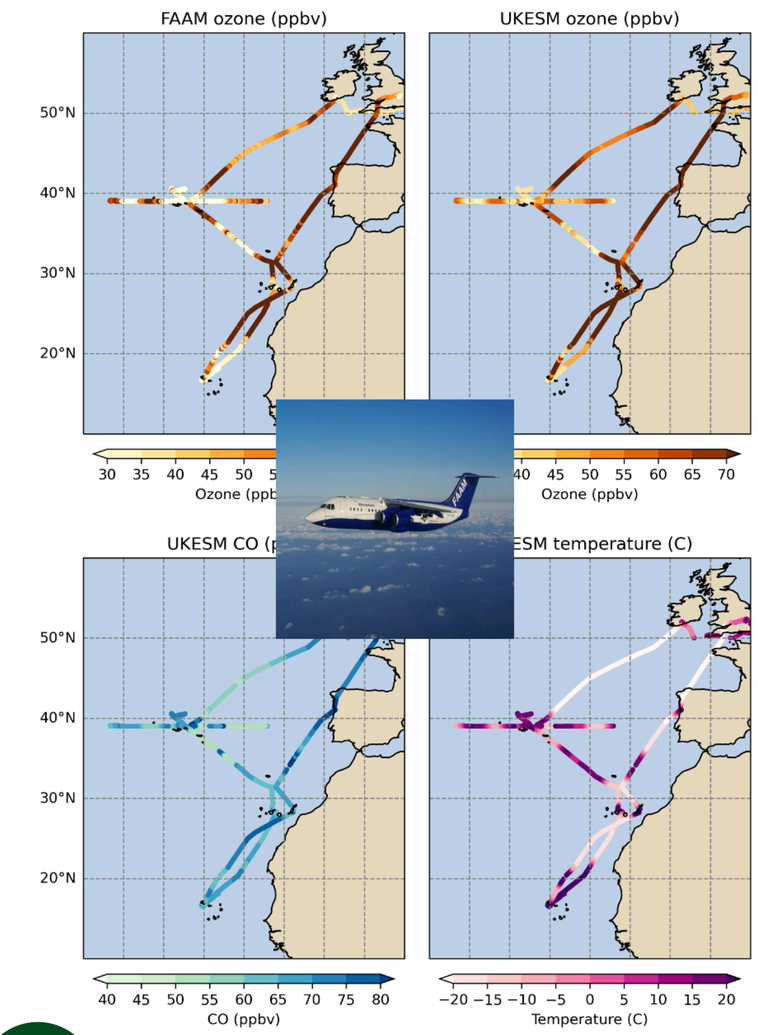
- Support for PyActiveStorage: identifying one or more recipes that can exploit remote reductions to minimise data movement. Part of AIVAL - where the motivation is to minimise data movement when data volumes are large.







TwineVision



PROJECT: VISION: TOWARDS SEAMLESS INTEGRATION OF MODEL, SATELLITE, AND IN-SITU OBSERVATION DATA

(Luke Abraham, Maria Russo)

VISION TOOLKIT

- an accessible package that is easy and fast to use, and is portable so can be used by anyone, with any CF-compliant model and observational dataset.
- Written in python with cf-python and cf-plot, data analysis and visualisation libraries developed and maintained within NCAS.
- An important aspect of portability is that the software relies on standardised CF-compliant data from/for the models. Supporting users with usage guidance to define what this entails.

See initial specification here: https://doi.org/10.5194/gmd-18-181-2025

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

University of Reading

SUMMARY

1. At scale, you can't wing it, you need to produce and depend on metadata.
2. There are a range of tools of increasing "metadata sophistication" - "if you have a hammer, not everything is a nail".
   - It's worth learning multiple tools!
3. You are probably going to start encountering remote data, if you haven't already.
   - It doesn't have to be Zarr, but whether it is or not, you do need to think about chunking and parallelism.
   - If it's HDF5 or NetCDF4, you and you need remote access and parallelism, it's pyfive or bust!
   - Time thinking about data structure will pay off big-time in the performance of your analysis tools.
4. There are a lot analysis tools that you can use out of the box, don't let ChatGPT encourage you to write more buggy software when you can use things that already do 90% of the job well!
   - If a tool is broken for your application, raise an issue on the tool maintainer's website, don't give up and write your own broken tools. If it's 90% of the way there, extend it and contribute!