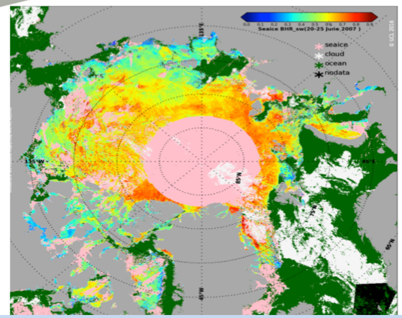
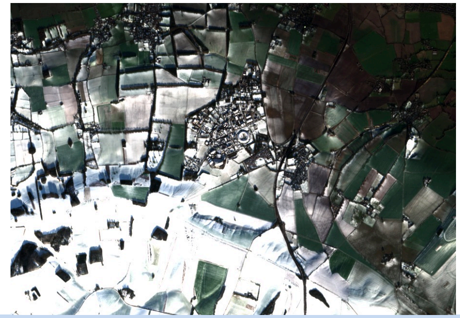
Science & Technology Facilities Council
Rutherford Appleton Laboratory





Semantic storage of climate data on object stores

SIG-IO 2018, 6th June 2018, Reading

***Neil Massey*[1]**, *David Hassell*[2], *Matt Jones*[1,2] *and Bryan Lawrence*[2,1]

[1]NCAS/NCEO, Centre for Environmental Data Analysis,
RAL Space, Rutherford Appleton Laboratory, STFC, UK

[2]NCAS-Climate/CMS, Department of Meteorology,
University of Reading

neil.massey@stfc.ac.uk

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL
National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL
esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE
Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Abstract

s3-netCDF-python is a Python (and Cython) library that enables writing and reading netCDF files to and from any storage system that has an Amazon S3 API.

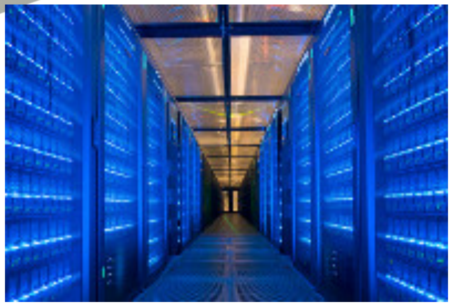
- S3 (simple storage solution) HTTP API used by
  - Object stores
  - Cloud-based storage (e.g. AWS)
- netCDF3 or netCDF4 (HDF5 based)
- netCDF file can be split into smaller “fragments”
  - Read only part(s) of the file that are required
  - Allow the parallel read and write of file fragments
  - Just-in-time reading and writing

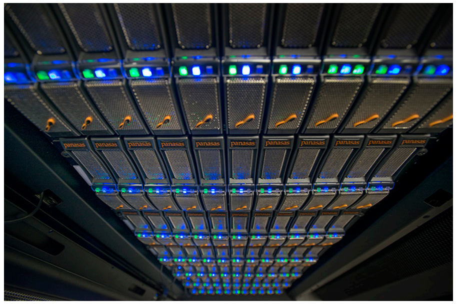
Context

- Centre for Environmental Data Analysis
  - A division of RAL Space
  - RAL Space is part of STFC
- CEDA is also part of NCAS and NCEO
- Government funded and research grants
  - EU H2020

- CEDA maintain a large archive of environmental data
  - Satellite data (esp. Sentinel and LandSat)
  - Climate and forecast data (CMIP5, ECMWF ERA-I, ERA-40)
  - Air measurement campaigns (EUFAR)
  - Ground based measurements
  - Approximately **7 PB**, including some data that is only on tape
- Provide access to the data to the scientific community and some commercial companies.

JASMIN





- A "super data-cluster”
  - a data-intensive computer
  - … and a datacentre
- Brings the compute to the data
  - and the data to the compute

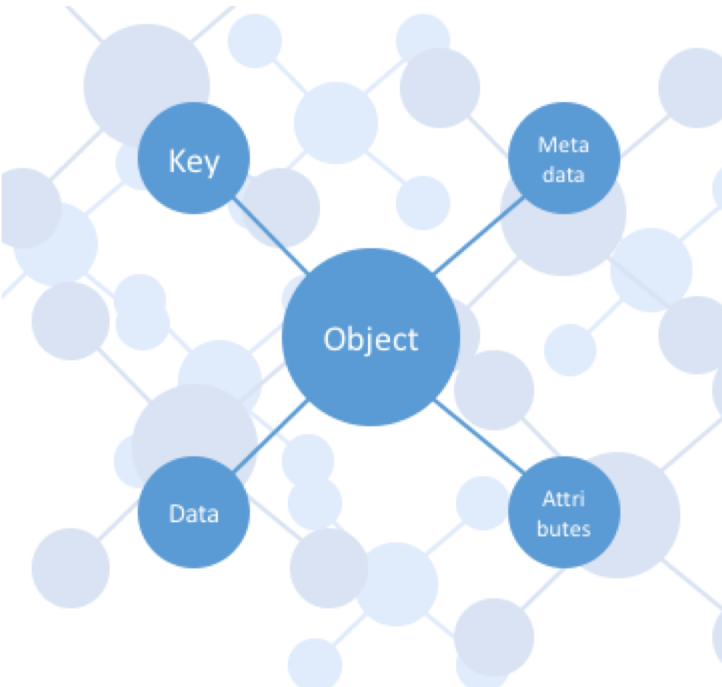
- For users it provides high-performance computing with:
  - Access to the archive
  - Storage on disk (user workspace, currently **10 PB**) and tape
  - Batch computing
  - Virtualised computing
  - Cloud computing
- A joint collaboration between CEDA and **STFC SCD** (Scientific Computing Division)

JASMIN Phase 4

- JASMIN is currently undergoing a major upgrade
  - **38.5 PB** of storage is being added
    - Including **5 PB** of Object Storage
  - **11 PB** of POSIX storage will be retired by the end of 2018
- Currently the computing environments have direct POSIX access to the data on disk
- However, with the IaaS (Infrastructure-as-a-Service) cloud model this is undesirable and may not be possible
- Using storage with a (S3) HTTP API overcomes this problem and provides access to the Object Store
  - Also allows access to off-site storage (e.g. personal AWS)

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Object Storage

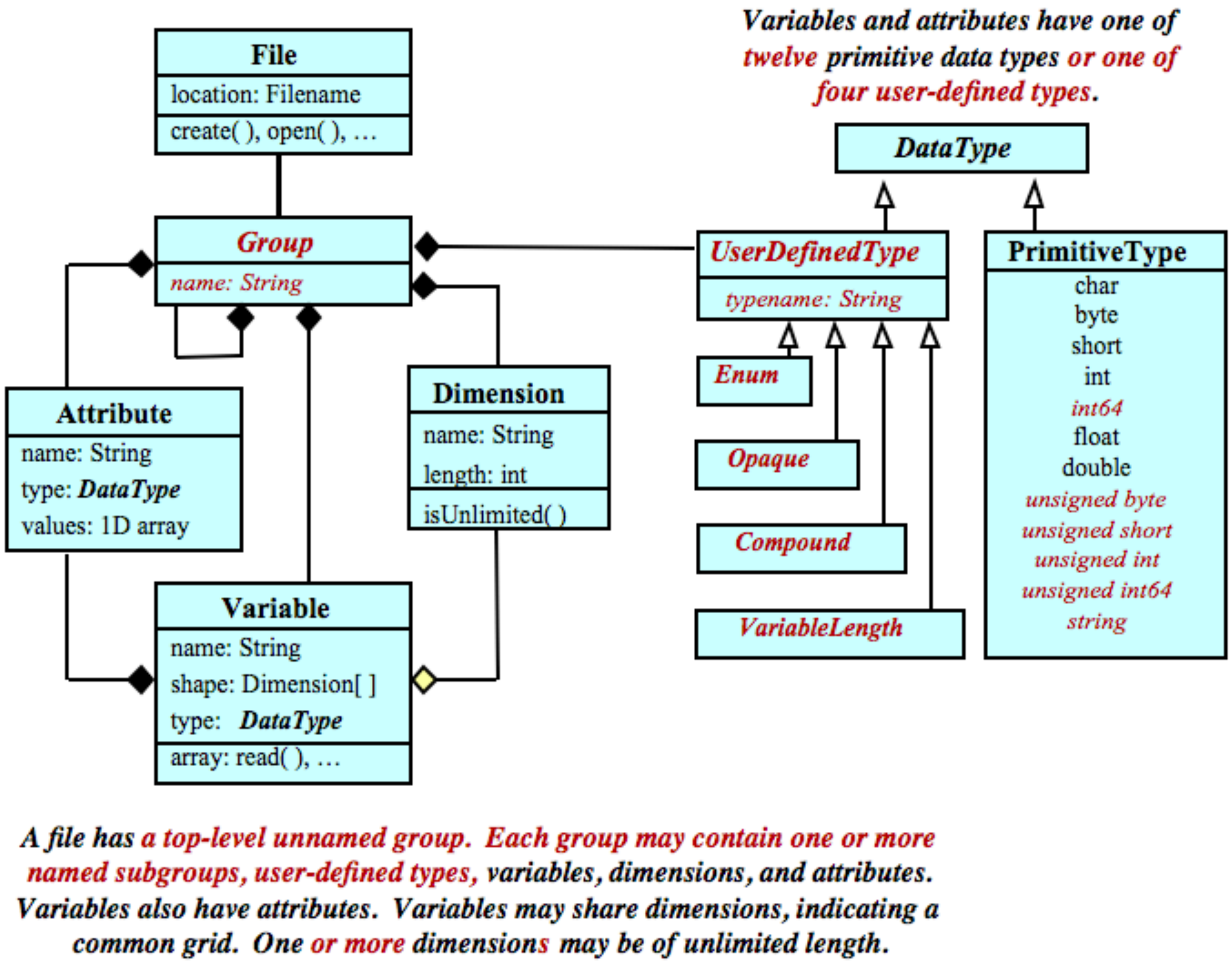


- A computer storage architecture in which *Objects* are stored in a flat structure
- Objects are identified by a unique key (a URL)
- Objects are organised into *Buckets*
- Object store can be accessed over a HTTP interface
  - Amazon's S3 HTTP REST API is the most popular
  - Data is uploaded and downloaded using PUT and GET operations respectively
- Supports two levels of metadata
  - System level metadata
  - Extendable metadata
- Allows searching for data without opening the file and custom searches for user data

What is a netCDF file?

- Network Common Data Form
- Array oriented data format
  - Multidimensional array variables
    - Variables are typed (int, float, etc.)
  - Coordinates for the dimensions (time, lat, lon, height, etc.)
  - Metadata for the variables (typed, including string)
  - Global metadata
  - Can take a “slice” (subdomain) from an array
- http://www.unidata.ucar.edu/software/netcdf/docs/user_guide.html

Structure of a netCDF file



http://www.unidata.ucar.edu/software/netcdf/docs/netcdf_data_set_components.html

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Structure of a netCDF file

The most useful command: **ncdump -h <filename>**

```
netcdf cru_ts3.24.01.1901.1910.tmp.dat {
dimensions:
     lon = 720 ;
     lat = 360 ;
     time = UNLIMITED ; // (120 currently)
variables:
     double lon(lon) ;
          lon:long_name = "longitude" ;
          lon:units = "degrees_east" ;
     double lat(lat) ;
          lat:long_name = "latitude" ;
          lat:units = "degrees_north" ;
     double time(time) ;
          time:long_name = "time" ;
          time:units = "days since 1900-1-1" ;
          time:calendar = "gregorian" ;
     double tmp(time, lat, lon) ;
          tmp:long_name = "near-surface temperature" ;
          tmp:units = "degrees Celsius" ;
          tmp:correlation_decay_distance = 1200. ;
          tmp:_FillValue = 9.96920996838687e+36 ;
          tmp:missing_value = 9.96920996838687e+36 ;
```

Dimension definitions

Dimension variables

Attributes

Field Variable

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

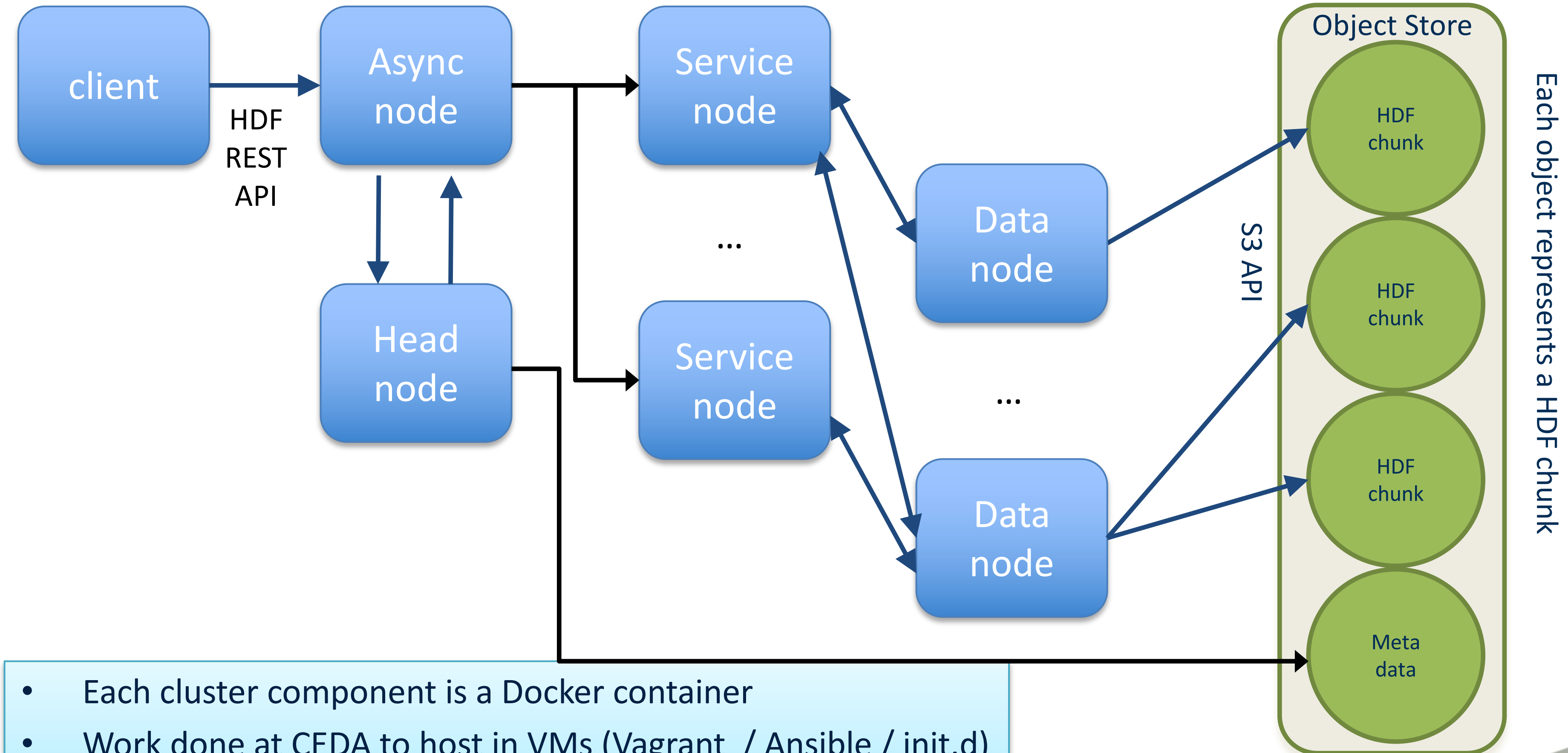
Writing netCDF files to object store

- It’s easy enough to write the whole file as an object, using minio API or boto3 API (Amazon’s API)
- Numerous disadvantages:
  1. Have to read / write the entire file at once to the object store
  2. Could use **range** function of (S3 API) but performance is unknown
  3. Does not permit parallel read / writes
  4. Have to read the entire file just to search the metadata!
- Instead we propose a method of splitting a netCDF file into fragments consisting of:
  1. A master array file, containing the variable definitions and metadata
  2. A number of subarray files containing subdomains of the variable data

HDF cloud and S3-netCDF

- **HDF cloud** from the HDF group also solves some of these problems
- HDF cloud has a cluster / server architecture:
  - The client connects to the cluster
  - The cluster serves the data from the object store to the client
  - The cluster load balances the requests
- **S3-netCDF** has a client-only model:
  - The client connects directly to the object store
  - The user is responsible for the load balancing
- We wanted to provide an easy to use library that can be used in the user workspaces for their own data

HSDS / HDF Cloud



- Each cluster component is a Docker container
- Work done at CEDA to host in VMs (Vagrant / Ansible / init.d)
- Client connects using h5pyd Python library (HDF REST API)

Key differences between HDF Cloud and S3netCDF4

HDF Cloud

- Any HDF file
- Connect using any HDF REST API client
- Cluster oriented architecture
- Parallel reads / writes limited to data nodes in the cluster
- Read / write fragments only to S3
- Fragments are HDF chunks with no semantic information

S3netCDF4

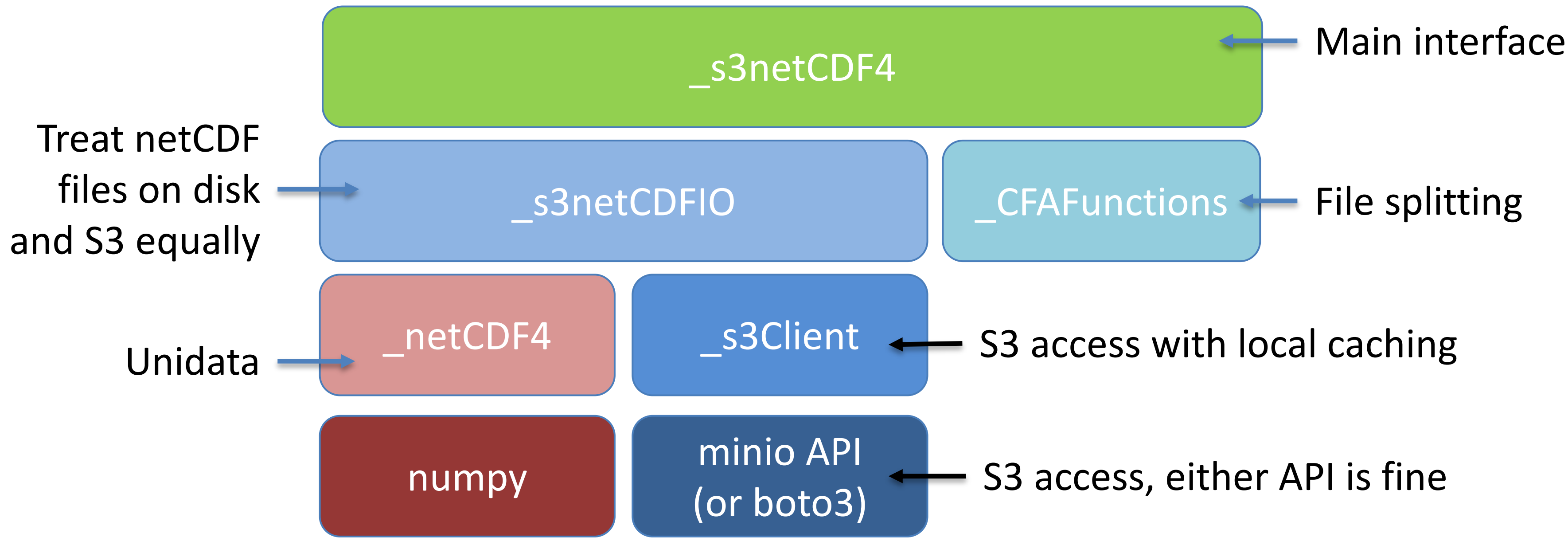
- netCDF4 & netCDF3 only
- Python only (currently)
- Client oriented architecture
- Parallel reads / writes limited to cores on the client machine
- Read / write fragments to S3, OpenDAP or local disk
- Fragments are self contained netCDF files
- Aggregation / data cube

S3-netCDF-python

- The library is implemented as a subclass of the standard Unidata python package (netCDF4.Dataset)

- Three main components:
  - **The interface** – matches netCDF4.Dataset as closely as possible
  - **A client** to read / write objects from / to a S3 object store, with the ability to stream to / from memory or to cache objects to disk, with sensible choices made based on available memory, object size and user input
  - **An array splitter** to split large netCDF4 variables into smaller ones, using the netCDF-CFA conventions

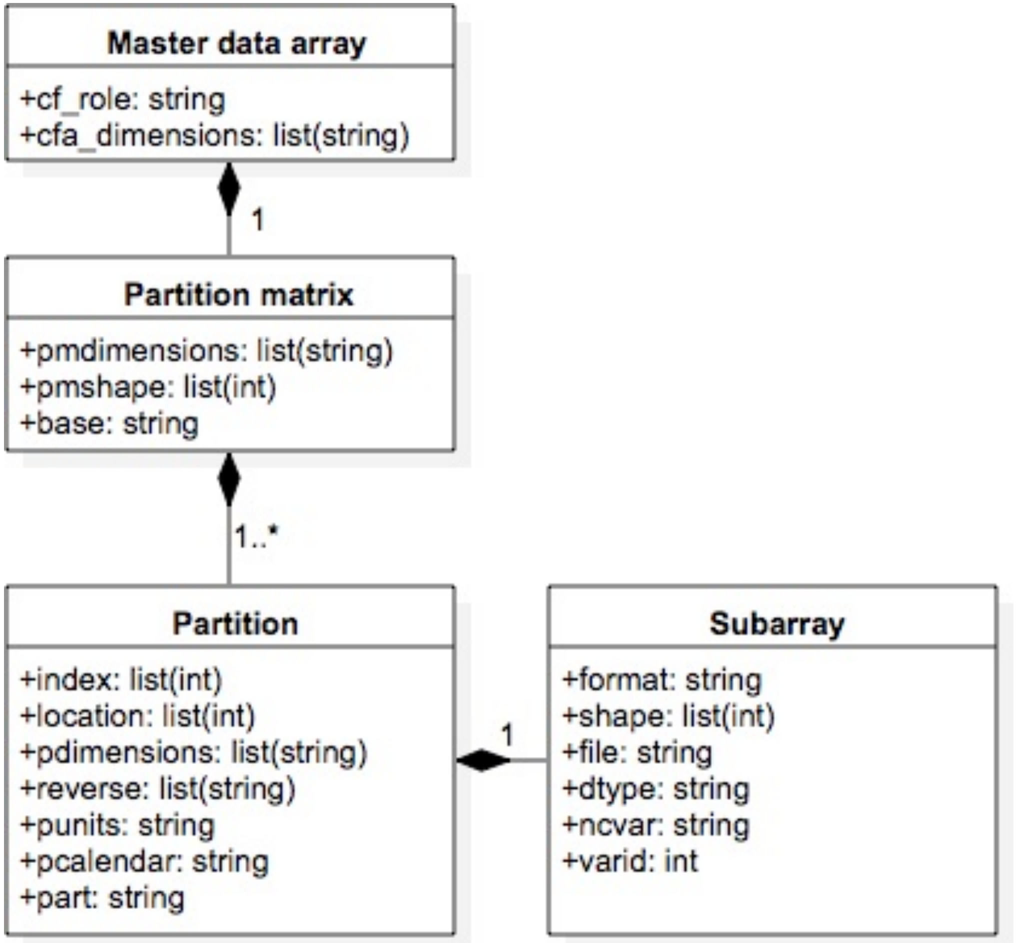
S3-netCDF software stack



CFA conventions

- Climate and forecast (CF) aggregation rules
  - Describe how multiple CF fields may be combined into one larger field
  - CFA-netCDF conventions for their efficient storage in netCDF files
  - Extension to netCDF via JSON encoded attributes
- A **master array** file (*kBs in size*)
  - Domains and metadata for a number of variables
  - Coordinates for the domains
  - Metadata for the subarrays, position in the master array
  - No field data
- A number of **subarray** files (*a single object, MBs to GBs in size*)
  - Subdomain and metadata (replicated from master array)
  - Coordinates for the subdomain
  - Field data

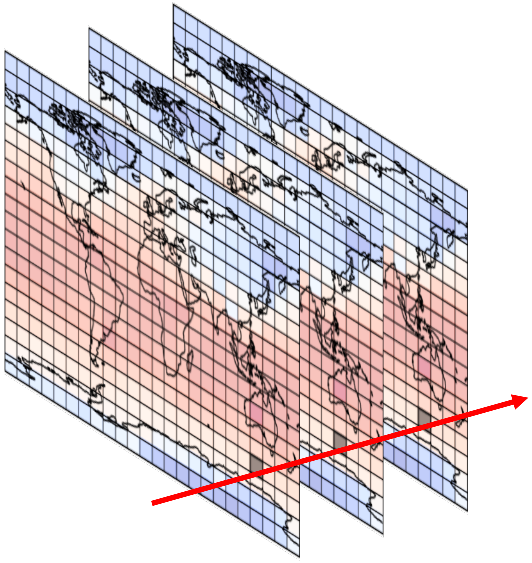
CFA Conventions

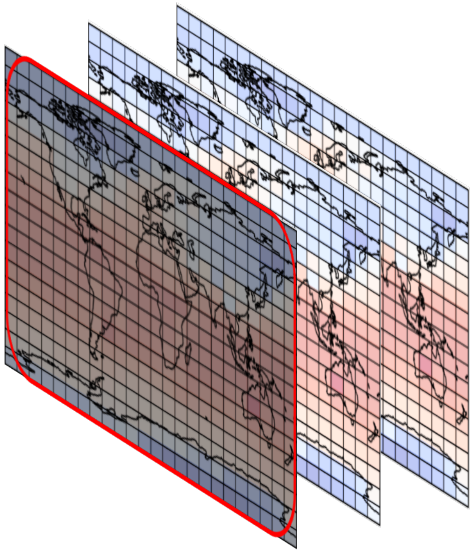


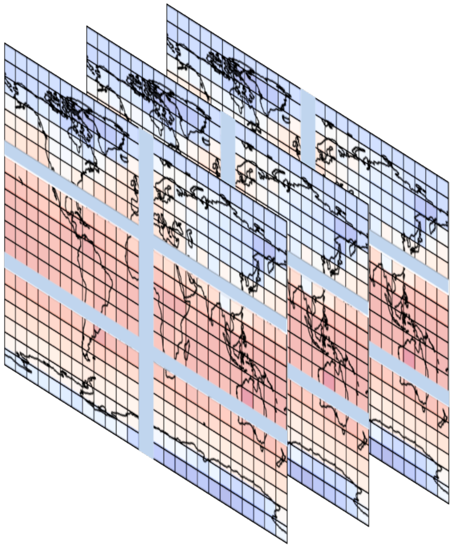
File splitting strategy

- NetCDF files are first split by **group**, then **variable**, then each variable is split into sub-domains. These sub-domains form the **sub-array** files.
- Access to the variable data involves reading and writing to the **sub-array** files.
- The size of the **sub-arrays** is optimised for two reading and writing use cases:
    1. The user reads a single spatial point (grid-box) for all the timesteps
    2. The user reads all the data (field) for a single timestep

File splitting strategy







1. $N = n_T/d_T$

2. $N = n_{lat}/d_{lat} * n_{lon}/d_{lon}$

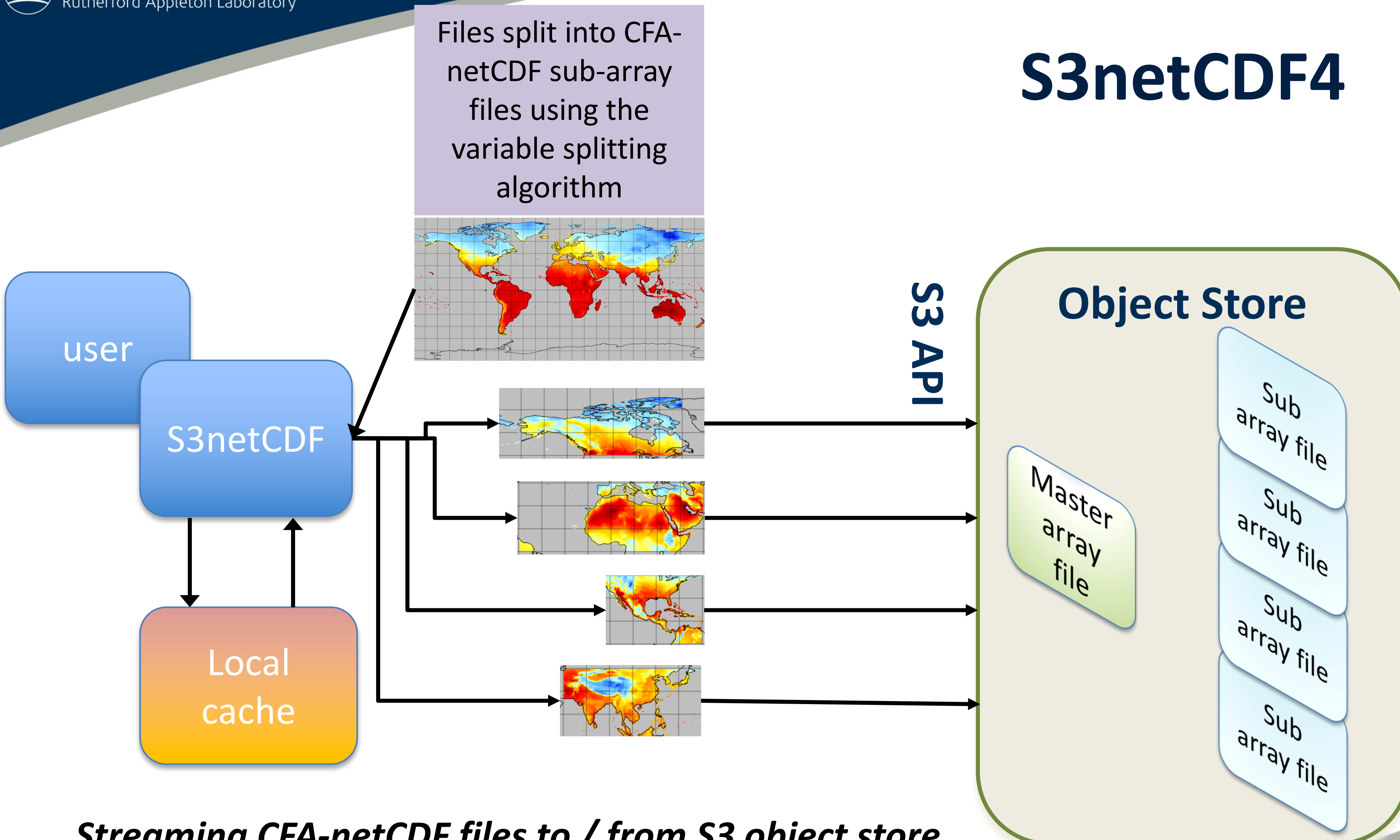
3. Approximately equal size "fragments"

N = number of operations needed to read entire timeseries / field
n = number of elements in the dimension
d = number of splits (divisions) in the dimension to form the sub-arrays

Reading / writing and data collection

- Client based architecture
  - A user library
  - Number of parallel reads limited to cores on client machine
  - Could be containerised (Docker) and multiple instances load-balanced (Kubernetes) for a server architecture
- Reading / writing consists of three stages:
  - Determine which subarrays are in the slice of the master array
  - Fetch the subarray data to either memory or a cached file
  - Copy the subarray data from memory to a memory mapped array

S3netCDF4



Streaming CFA-netCDF files to / from S3 object store

Why “semantic”?

- The master array file contains all the metadata and domains for all the subarrays
  - Only need to read the master array file to search the data
- Each subarray file contains all of its metadata and subdomain as well
  - Can reconstruct the master array file if it is lost
- The array splitter knows what each dimension represents (time, latitude, longitute, etc.) and acts accordingly
- Also good for aggregation
  - Add field data as it becomes available, e.g. each timestep of a GCM run
  - No need to rewrite the entire file – just the master array file and new subarray file

Current status and future work

- Current status:
  - Read / write to object store, disk or openDAP
  - Can take a slice, and only the subarray files in the slice are read from / written to
  - Mostly code compatible with netCDF4
  - Read and write subarray files in parallel (Python threads found to be the fastest)
- Future work:
  - SemSL – Semantic storage layers

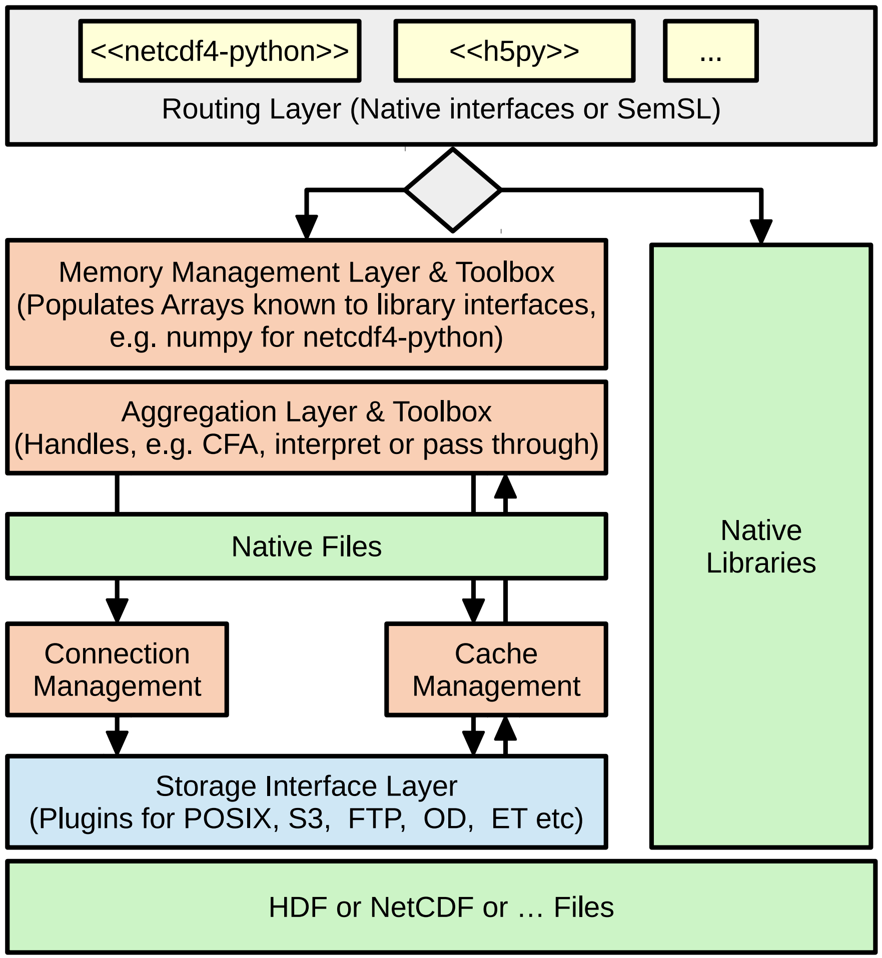
SemSL
Beyond object stores and netCDF

- ESiWACE project to expand on the capabilities of S3-netCDF
- Extend the fragmenting to other file formats:
  - netCDF *
  - HDF5 *
  - ESA SAFE
  - Tar files
  - Zip files
- Each of these “frontends” will be based upon existing libraries, e.g. netCDF4, h5py, etc.
  - Inherited and overloaded member functions where possible

SemSL
Beyond object stores and netCDF

- Extend the write capability to other storage technologies:
  - S3 / object store / AWS *
  - POSIX disk *
  - Parallel file systems
  - FTP *
  - Tape
- These "backends" will be based up existing libraries:
  - Boto3
  - FTPlib

SemSL – software architecture



SemSL

- The *Memory Managment* layer creates memory-mapped array objects, or memory-mapped file objects for non- array file-formats
- The *Aggregation Layer* handles the reading and writing of a *Master-File* and its *Fragments*
- *Cache Management:* A virtual or physical cache to temporarily stage files
- The *Connection Management* function allows connections to persist and be reused when reading / writing fragments
- The *Storage interface layer* provides a common interface to any number of external storage systems

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Acknowledgements

- This work has been supported primarily by European research funded by the H2020 research and innovation project ESIWACE, under grant agreement 675191.
- Additional support was received from UK Natural Environment Research Council national capability funding for the National Centre for Atmospheric Science (NCAS).
- None of it would have been possible without excellent support from the JASMIN systems team: Jonathan Churchill, Cristina Delcano and Athanasios Kanaris.

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

References

- S3-netCDF-python github
  https://github.com/cedadev/S3-netcdf-python
- Unidata netCDF4 package
  http://unidata.github.io/netcdf4-python/
- netCDF CFA conventions
  http://www.met.reading.ac.uk/~david/cfa/0.4/

Example code - write

```
# create a NETCDF4 file and upload to S3 storage
# this just follows the tutorial at http://unidata.github.io/netcdf4-python/
with Dataset(S3_WRITE_NETCDF_PATH, mode='w', diskless=True, format="CFA3") as s3_data:
    # create the dimensions
    leveld = s3_data.createDimension("level", len(levels_data))
    timed = s3_data.createDimension("time", None)
    latd = s3_data.createDimension("lat", 196)
    lond = s3_data.createDimension("lon", 256)
    # create the dimension variables
    times = s3_data.createVariable("time", "f8", ("time",))
    levels = s3_data.createVariable("level", "i4", ("level",))
    latitudes = s3_data.createVariable("lat", "f4", ("lat",))
    longitudes = s3_data.createVariable("lon", "f4", ("lon",))
    # create the field variable
    temp = s3_data.createVariable("tmp", "f4", ("time", "level", "lat", "lon"))

    # add some attributes
    s3_data.source = "netCDF4 python module tutorial"
    latitudes.units = "degrees north"
    longitudes.units = "degrees east"
    levels.units = "hPa"
    temp.units = "K"
    times.units = "hours since 0001-01-01 00:00:00.0"
    times.calendar = "gregorian"
```

Example code - read

```
# Test opening a CFA file on the object
with Dataset(WAH_S3_DATASET_PATH, 'r') as nc_file:
    nc_var = nc_file.getVariable("field8")
    print nc_var.shape
    print nc_var.dimensions
    print nc_var.name
    print nc_var.datatype
    print nc_var.size
    print type(nc_var)
    print np.mean(nc_var[0:10,0,40:80,40:80])

# load the original file and take the mean
with Dataset(WAH_NC4_DATASET_PATH) as src_file:
    src_var = src_file.variables["field8"]
    print type(src_var)
    print np.mean(src_var[0:10,0,40:80,40:80])
```