ENABLING TRANSFORMATIONAL SCIENCE

NCI's Data Analysis Environments

5th Feb 2021

AMOS Australian Meteorological & Oceanographic Society

NCI AUSTRALIA

Special thanks to all supporters:

Agenda

Time	topic
11:00-11:20	Overview of Pangeo Deployment at NCI
11:20-11:40	Documentation and Live demo of setting up Pangeo
11:40-12:00	Q&A and 10min break
12:00-12:15	Introduction to VDI and NCI-data-analysis environments *(this presentation)*
12:15-12:30	Overview of Xarray and Dask *(this presentation)*
12:30-12:45	Live demo
12:45-1:00	Q&A and wrap up the workshop

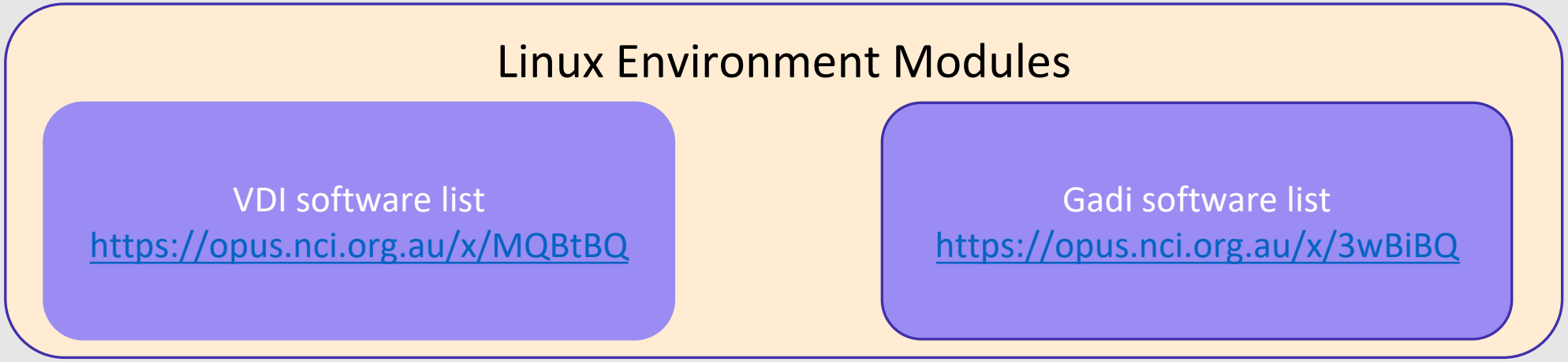
Introduction to data analysis environments

Learning goals

- Knowing where to query and load software
- Understand how to use NCI's **V**irtual **D**esktop **I**nfrastructure (VDI)
- How to use the NCI-data-analysis virtual environment

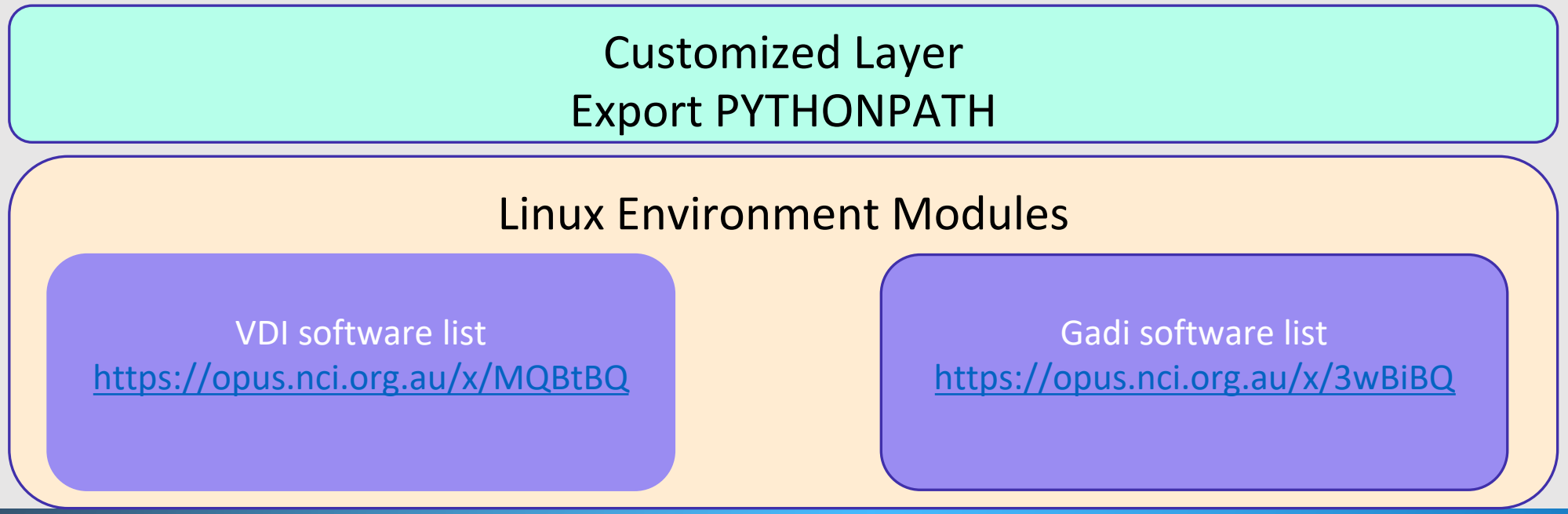
NCI's data analysis environments – system level

- Software applications are managed by sys admins
- Centralised installation with version control in /apps
- Search by "module avail package-name"
- Can request to install for users if widely used by the community



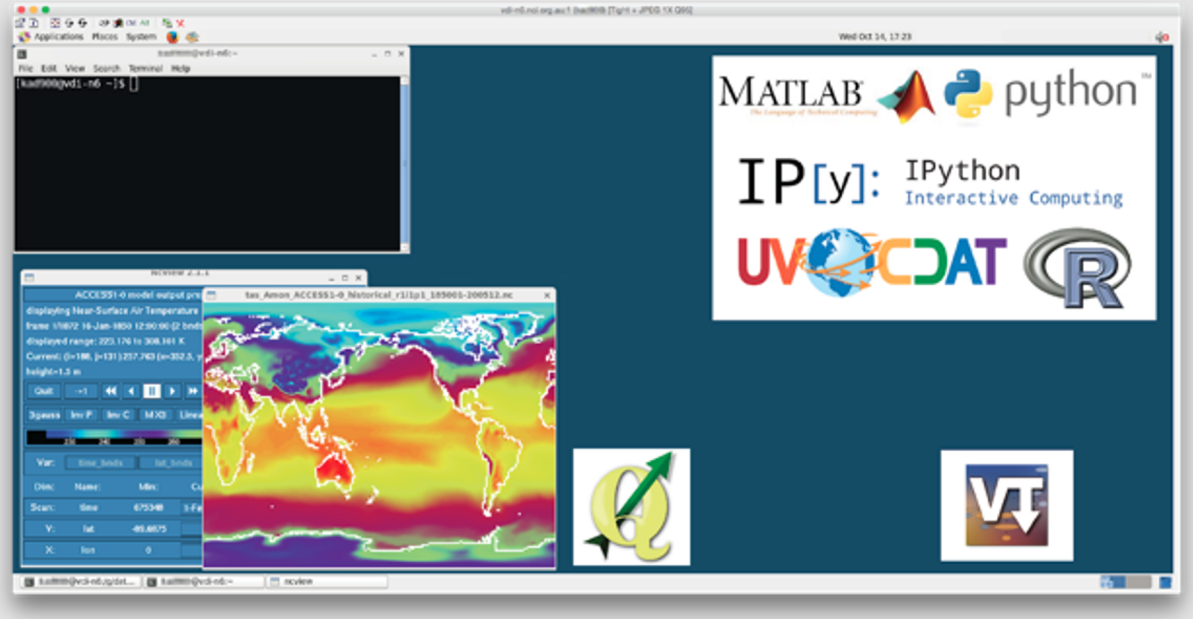
NCI's data analysis environments – user level

- Users can install additional software in their own space
  - Short term use – /scratch
  - Long term use – /g/data/compute-project
- Difference ways to customise personal data analysis environments using software package managers like pip or conda

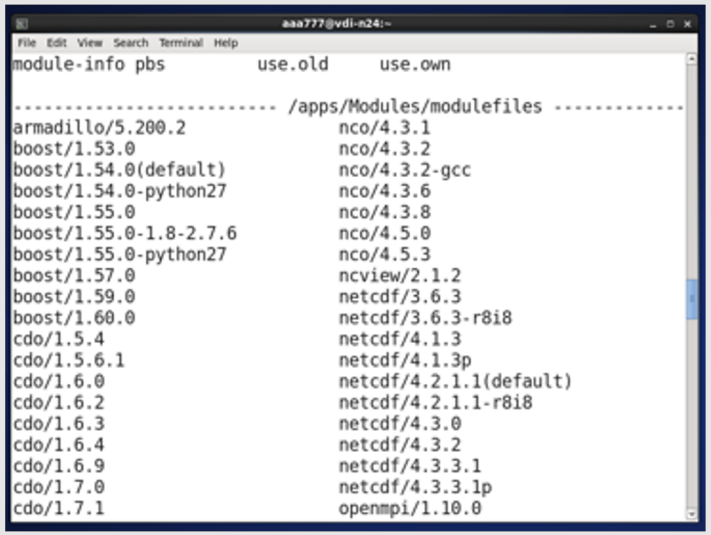


NCI's Virtual Desktop Infrastructure (VDI)

VDI is a graphical desktop-like interface giving you control over powerful cloud computing resources suited for data analysis, code development, visualisation and more. Datasets stored on NCI's /g/data filesystem are easily accessible through the VDI



For further information, see the VDI user guide:
https://opus.nci.org.au/display/Help/VDI+User+Guide



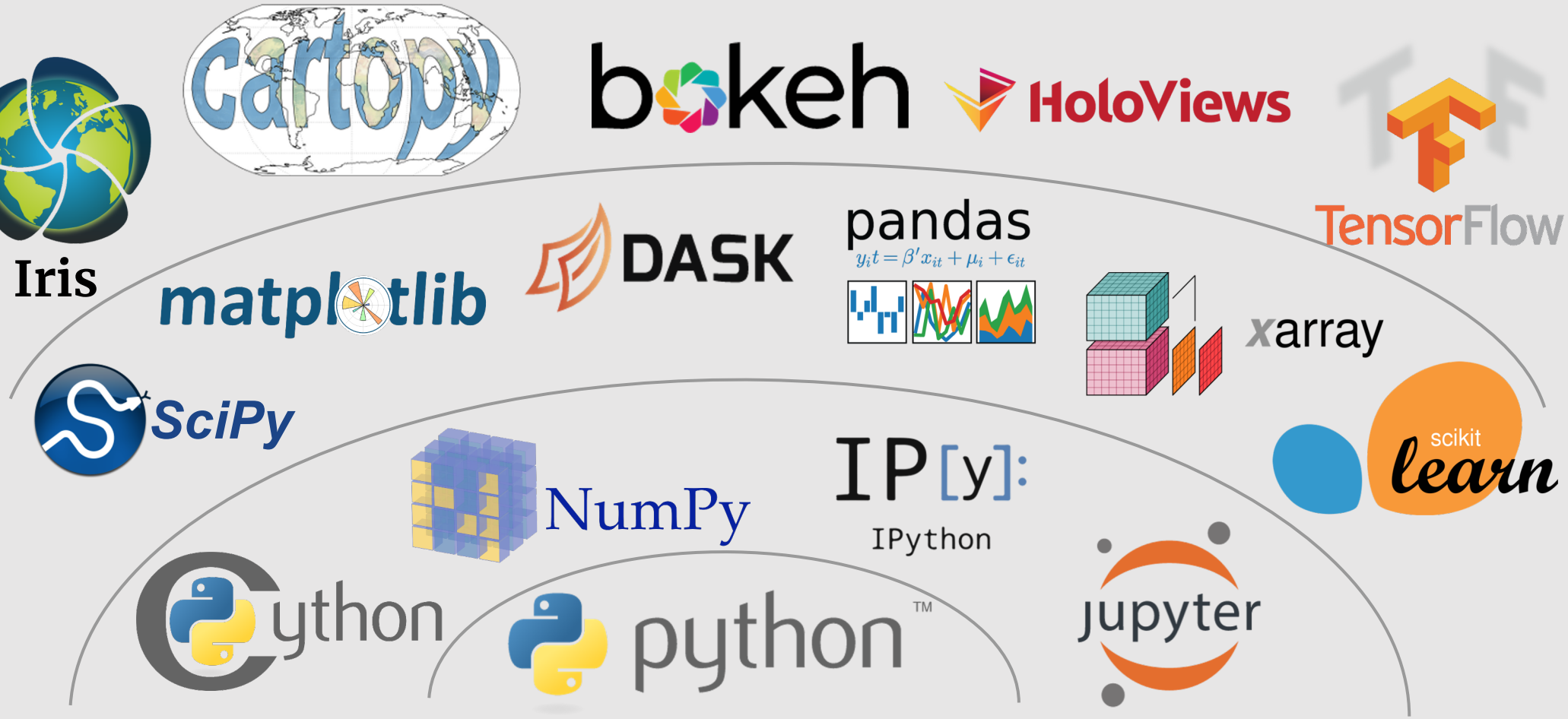
NCI-data analysis environment – dk92 Python environment

To ease the user experience of software installation, NCI provides a versioned virtual environment which includes a number of Python packages for data processing and visualisation. They are managed within project dk92. To use this virtual environment:

```
$ module use /g/data/dk92/apps/Modules/modulefiles
$ module load NCI-data-analysis/2020.12
```

A combination of Pangeo and NCI data-analysis virtual environments reduces the hassle of installing software and solving dependencies, and provides an interactive JupyterLab working environment.

NCI-data-analysis environment includes many of the essential Python packages



Credit: Jake Vanderplas, SciPy 2015

To get hands on experience…

1. You will need to join dk92 through MyNCI: https://my.nci.org.au/mancini/login
2. Use NCI-data-analysis environment for a quick start

More in dk92 – data analysis demonstration examples

[abc123 @Gadi:] tree -L 1 /g/data/dk92/notebooks

```
/g/data/dk92/notebooks
├── climate-cmip
├── demo_data
├── examples-dask
├── examples-gsky
├── examples-thredds
├── examples-xarray
├── geophysics-mt
└── geophysics-seismic
```

These examples are also available on GitHub:
https://github.com/NCI-data-analysis-platform

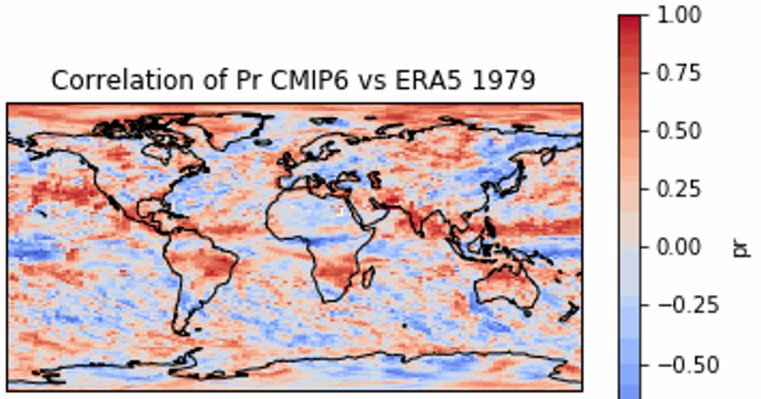
Data analysis demonstration examples - climate-cmip

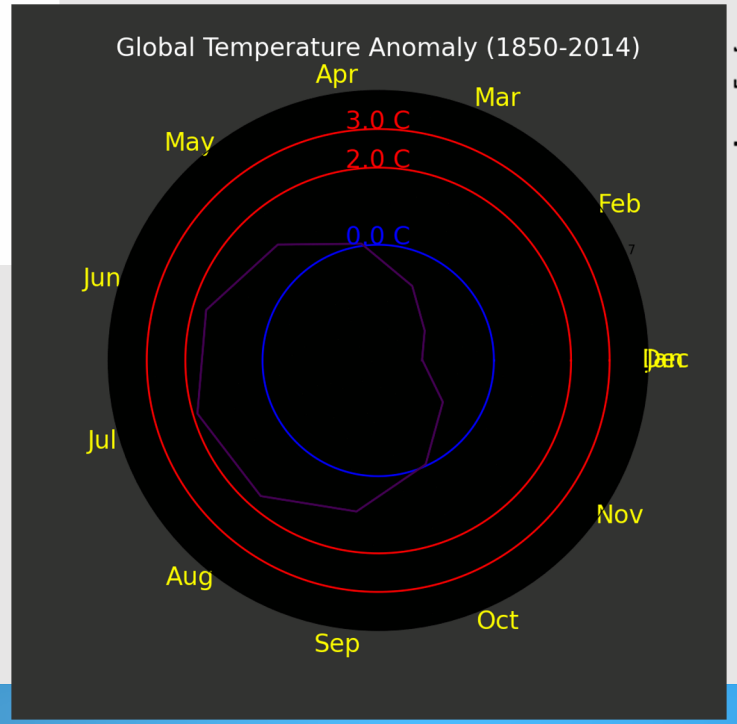
$ git clone https://github.com/NCI-data-analysis-platform/climate-cmip.git

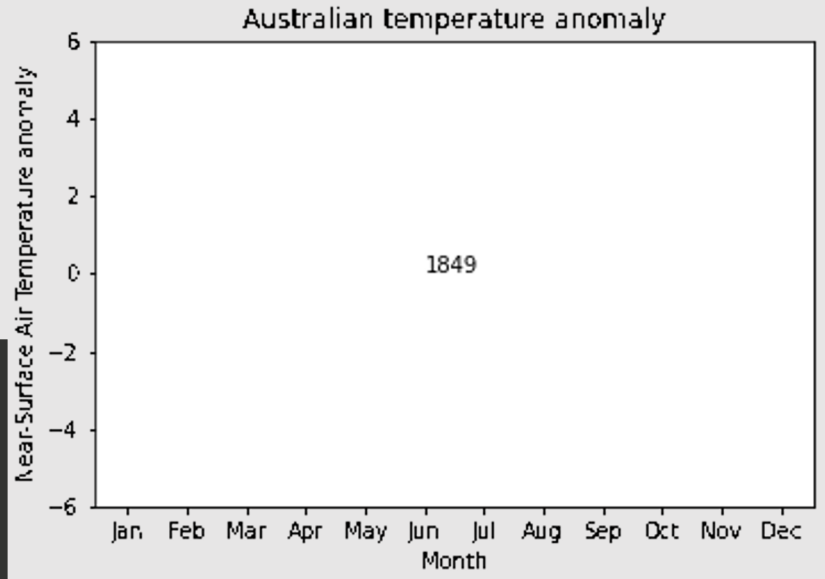
CMIP examples developed for use on NCI's Virtual Desktop Infrastructure (VDI). Examples include:

- using Climate Data Operations (CDO) for analysis
- how to access CMIP data in HDFView , Panoply and Paraview
- creating interesting animations using the CMIP dataset

Data analysis demonstration examples - climate-cmip







NCI data analysis platform - Dask and xarray examples

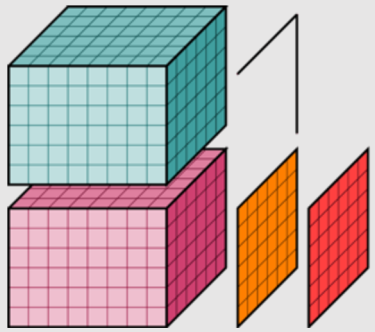
You can clone the notebook tutorials by running:

git clone https://github.com/NCI-data-analysis-platform/examples-dask.git

git clone https://github.com/NCI-data-analysis-platform/examples-xarray.git

In order for the Xarray and Dask examples to work, you will need to join the following project codes via MyNCI (https://my.nci.org.au/mancini):

- fs38 - ESGF CMIP6 Australian Data Publication
- oi10 - ESGF CMIP6 Replication Data
- rr3 - ESGF CMIP5 Australian Data Publication
- al33 - ESGF CMIP5 Replication Data
- fx3 - eReefs Hydrodynamic model data products
- yj45 - Australian Climate Observations Reference Network (ACORN)
- dk92 - Jupyter and Pangeo software



xarray

- Python package that works with **labelled** multi-dimensional arrays
- **Labels** in the form of dimensions, coordinates and attributes on top of raw NumPy-like arrays
- Allows for a more intuitive, more concise and less error-prone developer experience
- Interoperable with the core scientific Python packages (e.g. pandas, NumPy, Matplotlib)
- Large (and growing) library of domain-agnostic functions for advanced analytics and visualisation with these data structures
- Particularly tailored to work with netCDF files
- Integrates tightly with Dask for parallel computing

Xarray labels

- NumPy provides the fundamental data structure and API for working with raw ND arrays
- Real world datasets are usually more than just raw numbers - they have labels which encode information about how the array values map to locations in space, time, etc.
- Xarray uses metadata in the form of labeled dimensions (e.g., ‘latitude’ or ‘frequency’) and coordinate values (e.g., the date ‘2021-02-05’) to enable a suite of expressive, label based operations

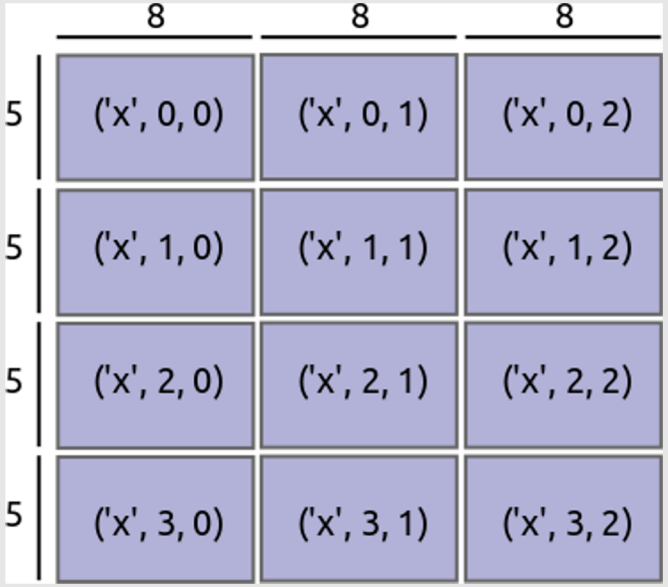
```
### Example of using labels with xarray:
data.sel(time='2020-12-01'.max(['latitude', 'longitude'])

### If using only NumPy:
data.temp.values[[10,11,12,13]].max(axis=(1,2));
```

What do labels enable?

- Can apply operations over dimensions by name: **x.sum('time')**
- Can select values by label instead of integer location: **x.loc['2021-02-05']** or **x.sel(time='2021-02-05')**
- Mathematical operations (e.g. x - y) vectorise across multiple dimensions based on dimension names, not shape
- Use the split-apply-combine paradigm with groupby: **x.groupby('time.dayofyear').mean()**
- Database-like alignment based on coordinate labels that smoothly handles missing values: **x, y = xr.align(x, y, join='outer')**
- Keep track of arbitrary metadata in the form of a Python dictionary: **x.attrs**

Xarray - Parallel computing with Dask

- Xarray integrates with Dask to support parallel computations and streaming computation on datasets that don’t fit into memory
- Dask divides arrays into many small pieces (chunks), each of which is presumed to be small enough to fit into memory.
- Dask scales up (to a cluster) and down (to a single machine)



Xarray Example 1:

Accessing CMIP data with Xarray

https://nbviewer.jupyter.org/github/NCI-data-analysis-platform/examples-xarray/blob/main/Xarray_01_data_access_CMIP5.ipynb

Xarray Example 2:

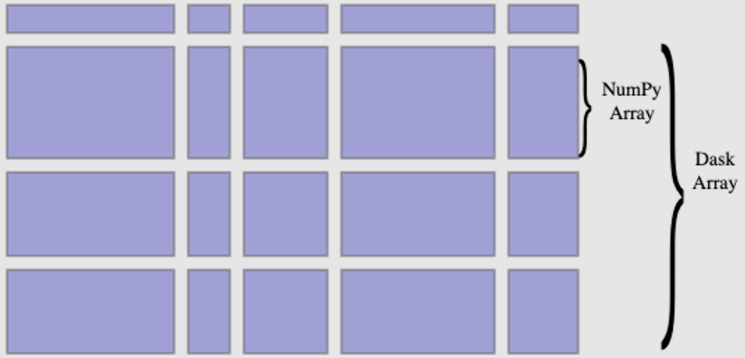
Subset and Plot CMIP Data Using Xarray

https://nbviewer.jupyter.org/github/NCI-data-analysis-platform/examples-xarray/blob/main/Xarray_02_subset_slicing_plot_CMIP6.ipynb

Dask overview:

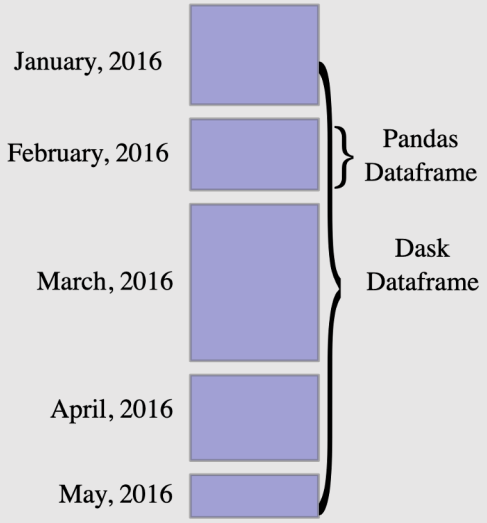
- Dask is a parallel computing library that scales the existing Python ecosystem

Numpy



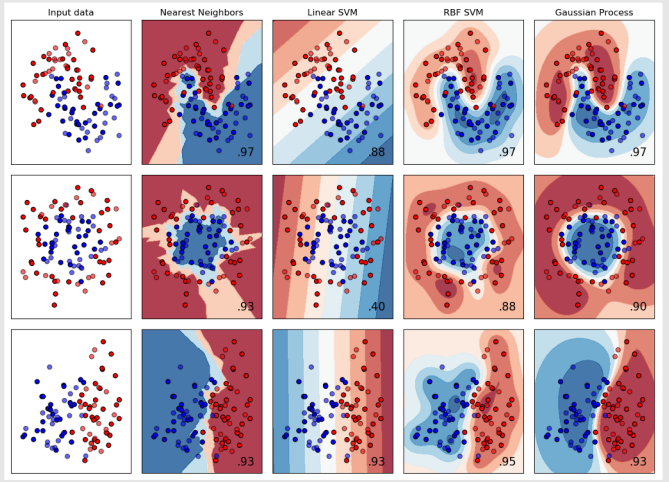
Dask arrays scale Numpy workflows, enabling multi-dimensional data analysis in earth science, satellite imagery, genomics, biomedical applications, and machine learning algorithms.

Pandas



Dask dataframes scale Pandas workflows, enabling applications in time series, business intelligence, and general data munging on big data.

Scikit-Learn

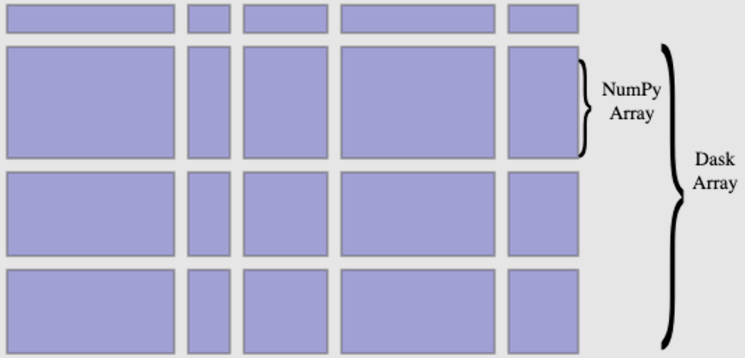


Dask-ML scales machine learning APIs like Scikit-Learn and XGBoost to enable scalable training and prediction on large models and large datasets.

Dask overview:

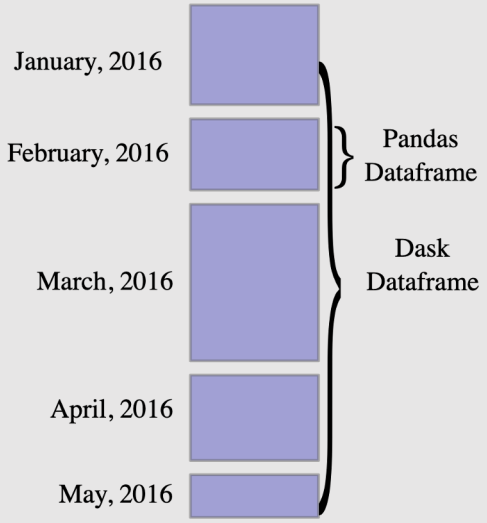
- Dask provides consistent user experience that stays true to the existing Python community of projects

Numpy



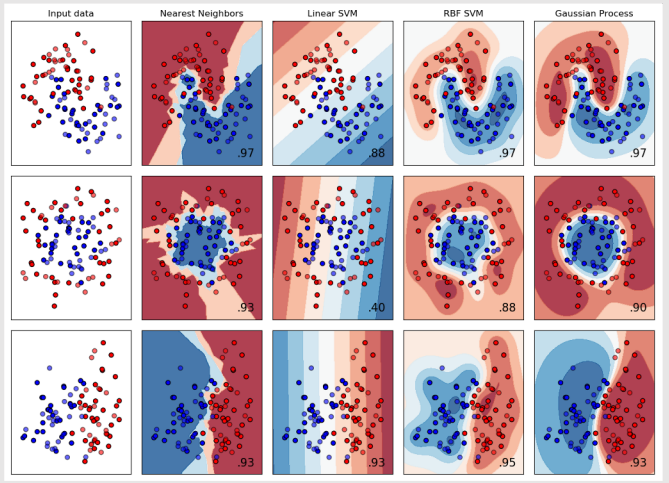

```
import numpy as np

x = np.ones((1000, 1000))
x + x.T - x.mean(axis=0)
```

Pandas




```
import pandas as pd

df = pd.read_csv("/g/data/ab12/*.csv")
df.groupby("x").y.mean()
```

Scikit-Learn

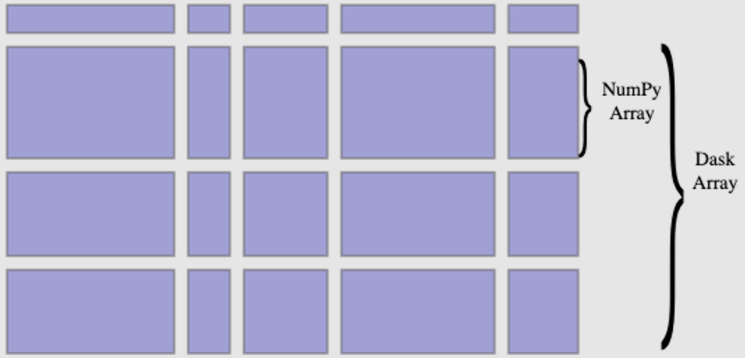



```
from scikit_learn.linear_model  import
LogisticRegression

lr = LogisticRegression()
lr.fit(data, labels)
```

Dask overview:

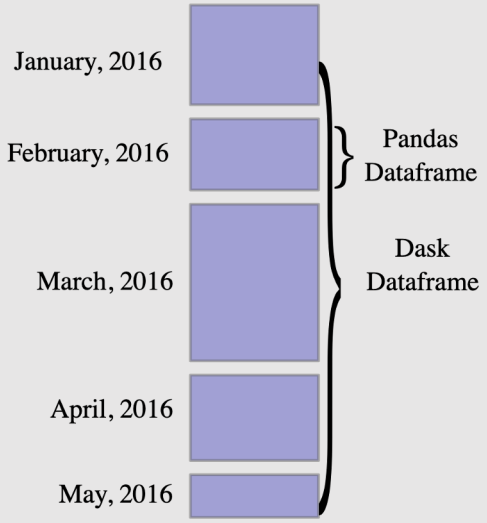
- Dask provides consistent user experience that stays true to the existing Python community of projects

Numpy



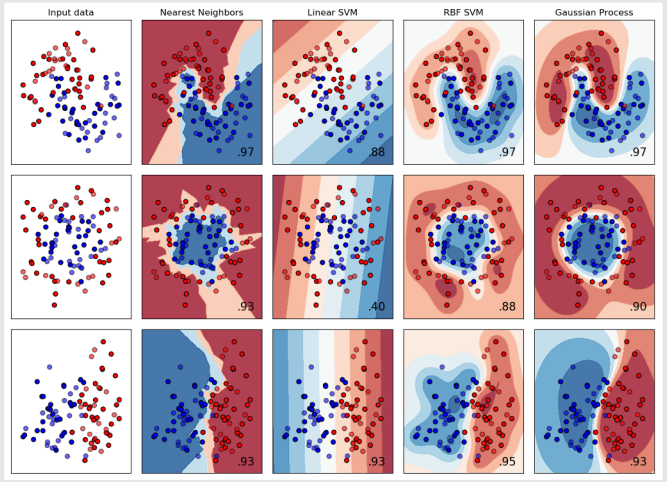

```
import dask.array as da

x = da.ones((10000, 10000))
x + x.T - x.mean(axis=0)
```

Pandas




```
import dask.dataframe as dd

df = dd.read_csv("/g/data/ab12/*.csv")
df.groupby("x").y.mean()
```

Scikit-Learn




```
from dask_ml.linear_model import
LogisticRegression

lr = LogisticRegression()
lr.fit(data, labels)
```

Dask overview:

- Dask provides multi-core and distributed parallel execution on larger-than-memory datasets
- Dask can scale down to your laptop and scale up to a cluster

High level collections: Dask provides high-level Array, Bag, and DataFrame collections that mimic NumPy, lists, and Pandas but can operate in parallel on datasets that don't fit into memory.

Low Level schedulers: Dask provides dynamic task schedulers that execute task graphs in parallel. These execution engines power the high-level collections mentioned above but can also power custom, user-defined workloads. These schedulers are low-latency (around 1ms) and work hard to run computations in a small memory footprint.

Dask: A conceptual scientific workflow

```
# import some packages
Import Xarray, dask, Shapely……
# read data
dat = xr.open_mfdataset(…)
# Pre-process data
dat_mean = dat.mean(axis=0)
# Data analysis
result = analysis_algorithm(dat_mean)
# Build model and predict the future
Prediction = ML_algorithm(Model,
input_dataset)
# Save the prediction
Output = save_to_HDF5(prediction)
```

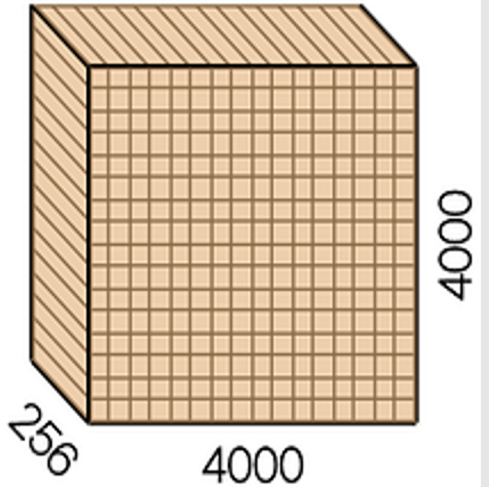
Dask works with different data types (e.g. time series, data frames, data arrays, text, list, dictionary, etc) in a similar way to how users would normally work with Python libraries such as NumPy, Pandas, Scikit-learn, etc.

Dask: A conceptual scientific workflow

```
# import some packages
Import Xarray, dask, Shapely……
# read data
dat = xr.open_mfdataset(…)
# Pre-process data
dat_mean = dat.mean(axis=0)
# Data analysis
result = analysis_algorithm(dat_mean)
# Build model and predict the future
Prediction = ML_algorithm(Model,
input_dataset)
# Save the prediction
Output = save_to_HDF5(prediction)
```

Dask can read larger-than-memory datasets through chunking to scale up calculations

	Array	Chunk
Bytes	32.77 GB	128.00 MB
Shape	(256, 4000, 4000)	(256, 250, 250)
Count	8560 Tasks	256 Chunks
Type	float64	numpy.ndarray



Dask: A conceptual scientific workflow

```
# import some packages
Import Xarray, dask, Shapely……
# read data
dat = xr.open_mfdataset(…)
# Pre-process data
dat_mean = dat.mean(axis=0)
# Data analysis
result = analysis_algorithm(dat_mean)
# Build model and predict the future
Prediction = ML_algorithm(Model,
input_dataset)
# Save the prediction
Output = save_to_HDF5(prediction)
```

Dask can process data that doesn't fit into memory by breaking it into blocks and specifying task chains.

Dask operates lazily (i.e. it does not evaluate until you explicitly ask for a result using the .compute() method).

Dask: A conceptual scientific workflow

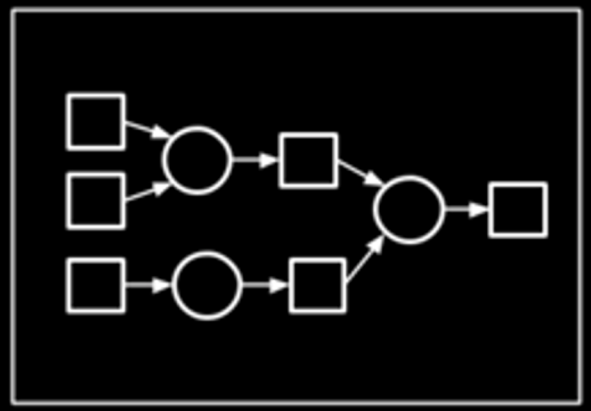
```
# import some packages
Import Xarray, dask, Shapely……
# read data
dat = xr.open_mfdataset(…)
# Pre-process data
dat_mean = dat.mean(axis=0)
# Data analysis
result = analysis_algorithm(dat_mean)
# Build model and predict the future
Prediction = ML_algorithm(Model,
input_dataset)
# Save the prediction
Output = save_to_HDF5(prediction)
```

Dask can process data in chunks and run parallel computation automatically, or with some light weight configuration on threads or processes (local or distributed)

Dask APIs build graphs, Dask schedulers execute them

Collections
(create task graphs)
→
Task Graph
→
Schedulers
(execute task graphs)

- Dask Array
- Dask DataFrame
- Dask Bag
- Dask Delayed
- Futures



- Single-machine (threads, processes, synchronous)
- Distributed

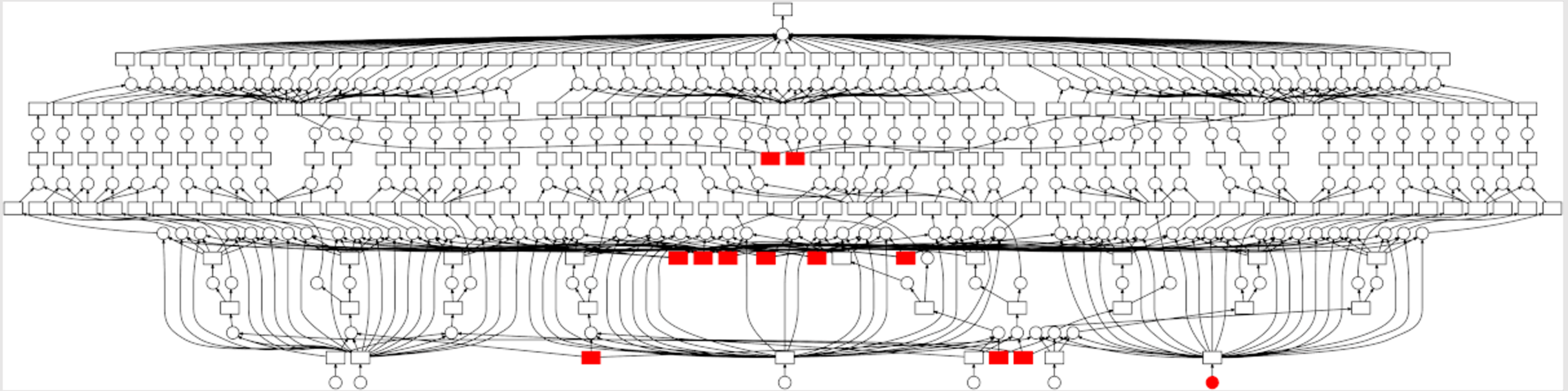
A variety of frontend APIs

A variety of backend Execution systems

Scheduling

Two families of schedulers:

1. Single machine scheduler
2. Distributed scheduler



Dask: A conceptual scientific workflow

```
# import some packages
Import Xarray, dask, Shapely……
# read data
dat = xr.open_mfdataset(…)
# Pre-process data
dat_mean = dat.mean(axis=0)
# Data analysis
result = analysis_algorithm(dat_mean)
# Build model and predict the future
Prediction = ML_algorithm(Model,
input_dataset)
# Save the prediction
Output = save_to_HDF5(prediction)
```

Dask-ML provides scalable machine learning in Python using Dask alongside popular machine learning libraries like Scikit-Learn

```
import dask.dataframe as dd
df = dd.read_parquet('...')
data = df[['age', 'income', 'married']]
labels = df['outcome']

from dask_ml.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(data, labels)
```

Dask: A conceptual scientific workflow

```
# import some packages
Import Xarray, dask, Shapely……
# read data
dat = xr.open_mfdataset(…)
# Pre-process data
dat_mean = dat.mean(axis=0)
# Data analysis
result = analysis_algorithm(dat_mean)
# Build model and predict the future
Prediction = ML_algorithm(Model,
input_dataset)
# Save the prediction
Output = save_to_HDF5(prediction)
```

You can store Dask arrays in a variety of common sources like HDF5, NetCDF, Zarr or any format that supports NumPy-style slicing

Dask Example:

Expensive CMIP6 calculation

https://nbviewer.jupyter.org/github/NCI-data-analysis-platform/examples-dask/blob/main/Dask_12_intensive_calculation_cmip6.ipynb

Further resources:

- NCI Data Analysis Environments user guide: https://opus.nci.org.au/x/cAA2Bg
- Pangeo community website: http://pangeo.io
- Xarray: http://xarray.pydata.org/
- Dask: https://dask.org/