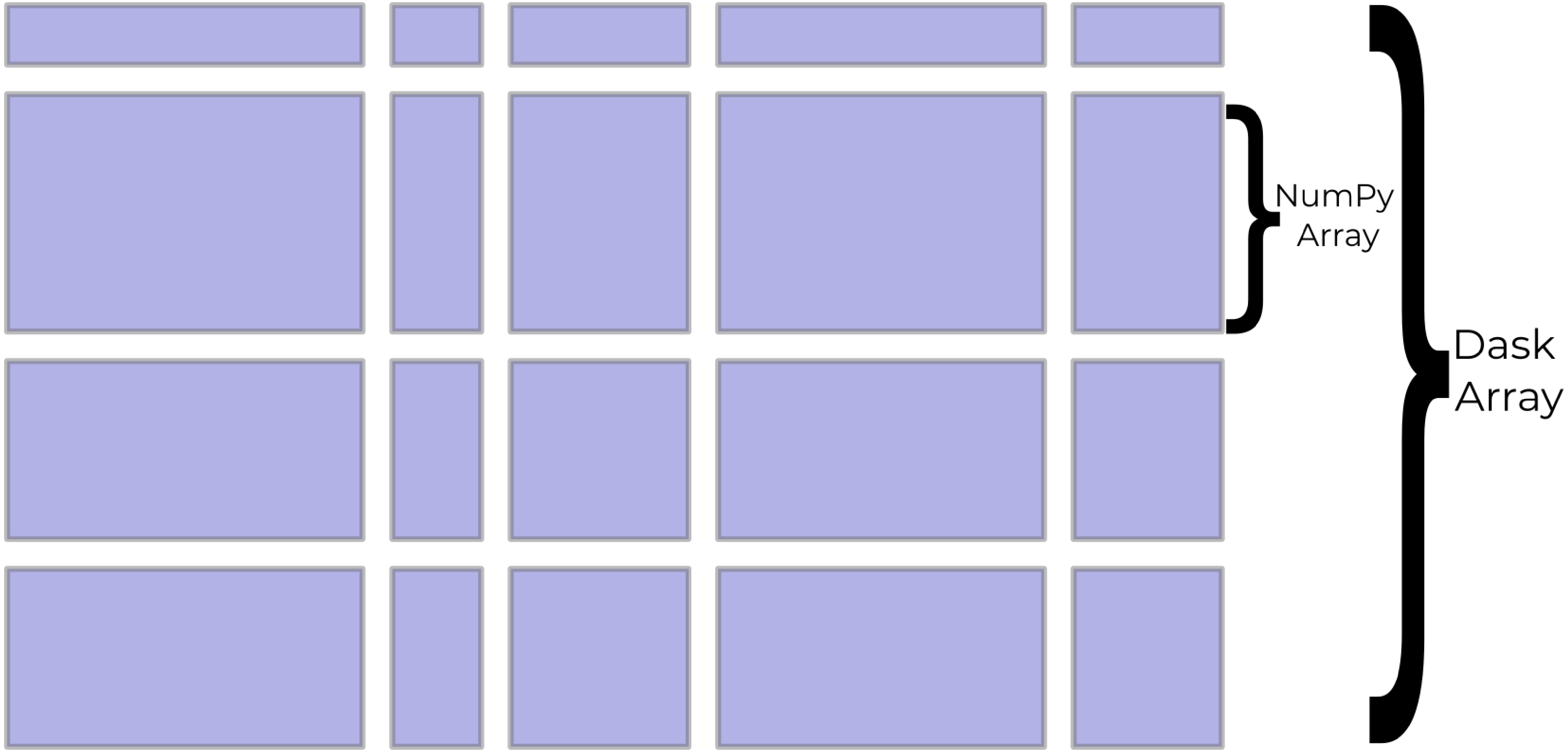
Dask arrays

PARALLEL PROGRAMMING WITH DASK IN PYTHON

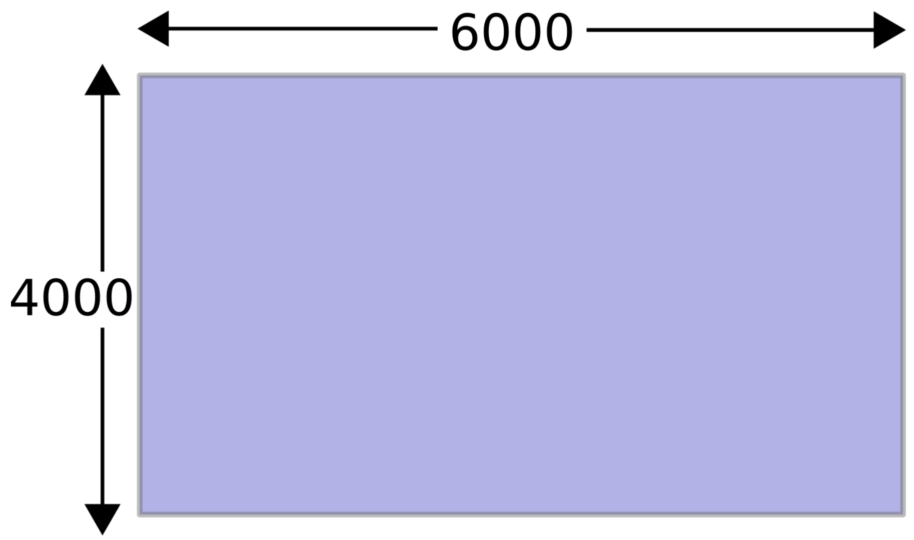
James Fulton
Climate informatics researcher

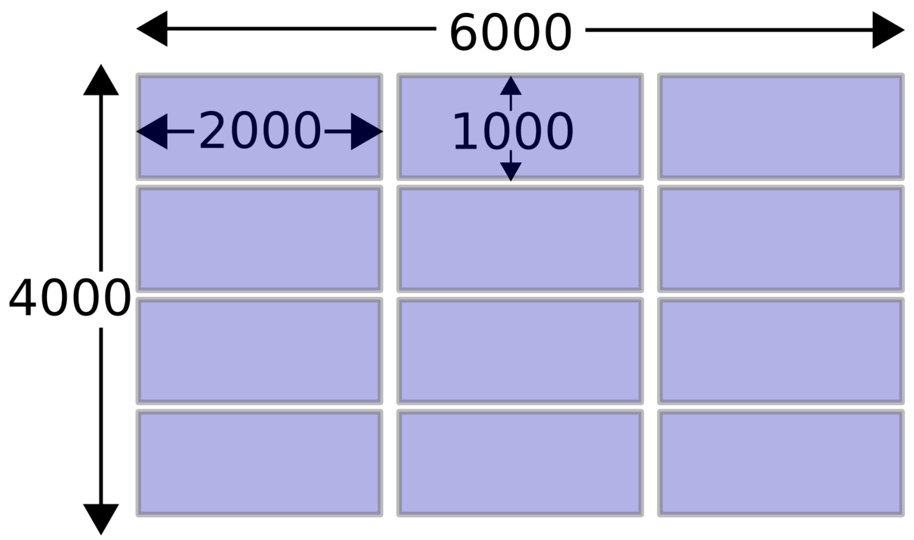
Chunking arrays

Chunking arrays



NumPy vs. Dask arrays






```python
import numpy as np
x = np.ones((4000, 6000))

print(x.sum())
```

```
24000000.0
```

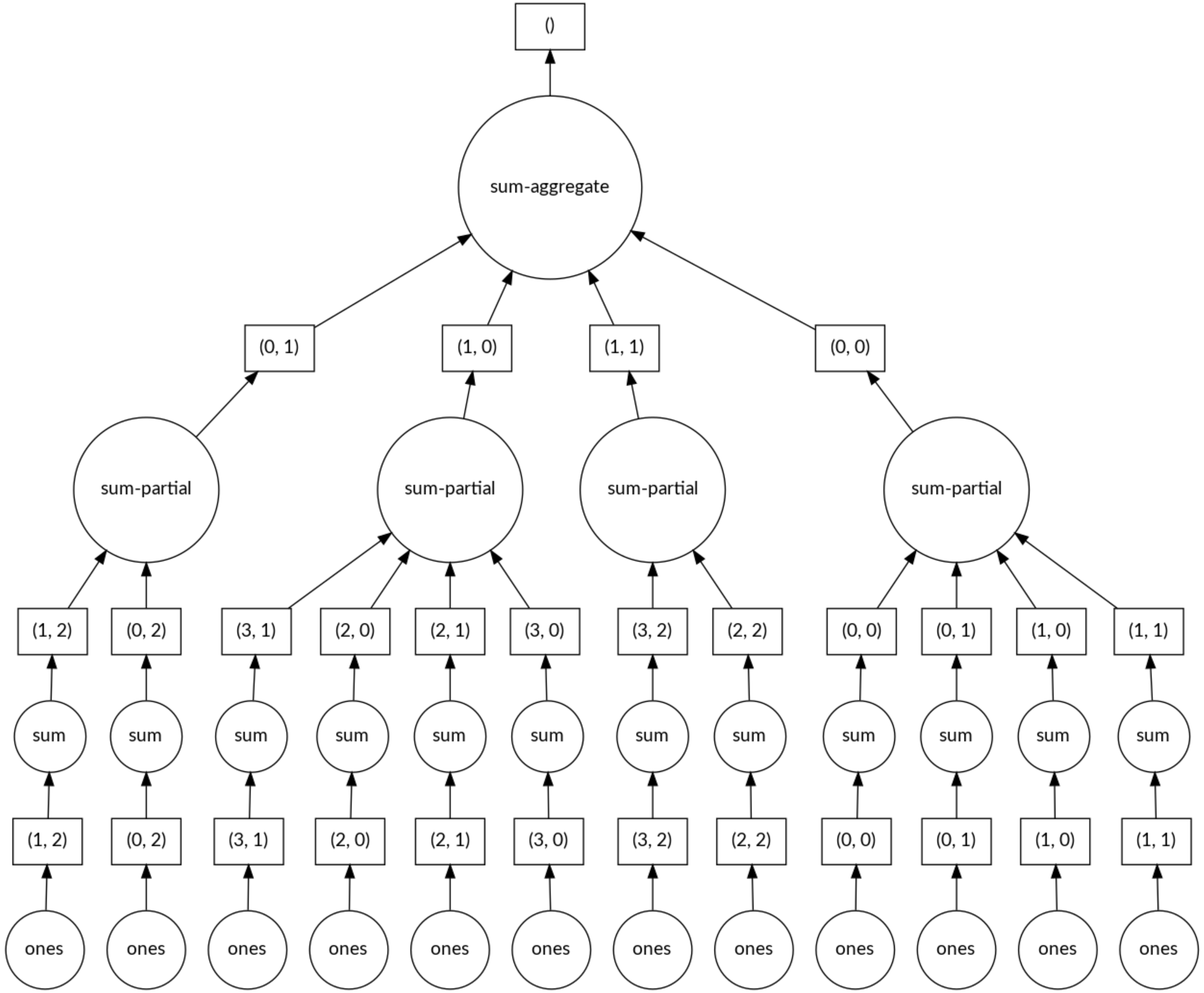
- Takes 740 milliseconds to run

```python
import dask.array as da
x = da.ones((4000, 6000),
            chunks=(1000,2000))
print(x.sum().compute())
```

```
24000000.0
```

- Takes 60 milliseconds to run

Dask array task graph



Dask array methods

Dask arrays have almost all the methods that NumPy arrays have.

- `x.max()`
- `x.min()`
- `x.sum()`
- `x.mean()`
- etc.

```
print(sum_down_columns.compute())
```

```
array([1000., 1000., 1000., 1000.,
    1000., 1000., 1000., 1000., 1000.,
    1000.])
```

Treating Dask arrays like NumPy arrays

```python
# Lazy mathematics with Dask array
y1 = x**2 + 2*x + 1

# Lazy slicing
y2 = x[:10]

# Applying NumPy functions is lazy too
y3 = np.sin(x)
```

```python
print(y1)
```

```
dask.array<add, shape=(1000, 10), ...
```

```python
print(y2)
```

```
dask.array<getitem, shape=(10, 10), ...
```

```python
print(y3)
```

```
dask.array<sin, shape=(1000, 10), ...
```

Loading arrays of images

```
import dask.array as da
import da.image

image_array = da.image.imread('images/*.png')

print(image_array)
```

```
dask.array<imread, shape=(40000, 256, 256, 3), dtype=uint8,
    chunksize=(1, 256, 256, 3), chunktype=numpy.ndarray>
```

Applying custom functions over chunks

```
def instagram_filter(image):
    ...
    return pretty_image

# Apply function to each image independently
pretty_image_array = image_array.map_blocks(instagram_filter)

print(pretty_image_array)
```

```
dask.array<instagram_filter, shape=(40000, 256, 256, 3), dtype=uint8,
    chunksize=(1, 256, 256, 3), chunktype=numpy.ndarray>
```

Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

Dask DataFrames

PARALLEL PROGRAMMING WITH DASK IN PYTHON

James Fulton
Climate Informatics Researcher

pandas DataFrames vs. Dask DataFrames

```
import pandas as pd
# Read a single csv file
pandas_df = pd.read_csv(
    "dataset/chunk1.csv"
)
```

This reads a single CSV file immediately.

```
import dask.dataframe as dd
# Lazily read all csv files
dask_df = dd.read_csv(
    "dataset/*.csv"
)
```

This reads all the CSV files in the `dataset` folder lazily.

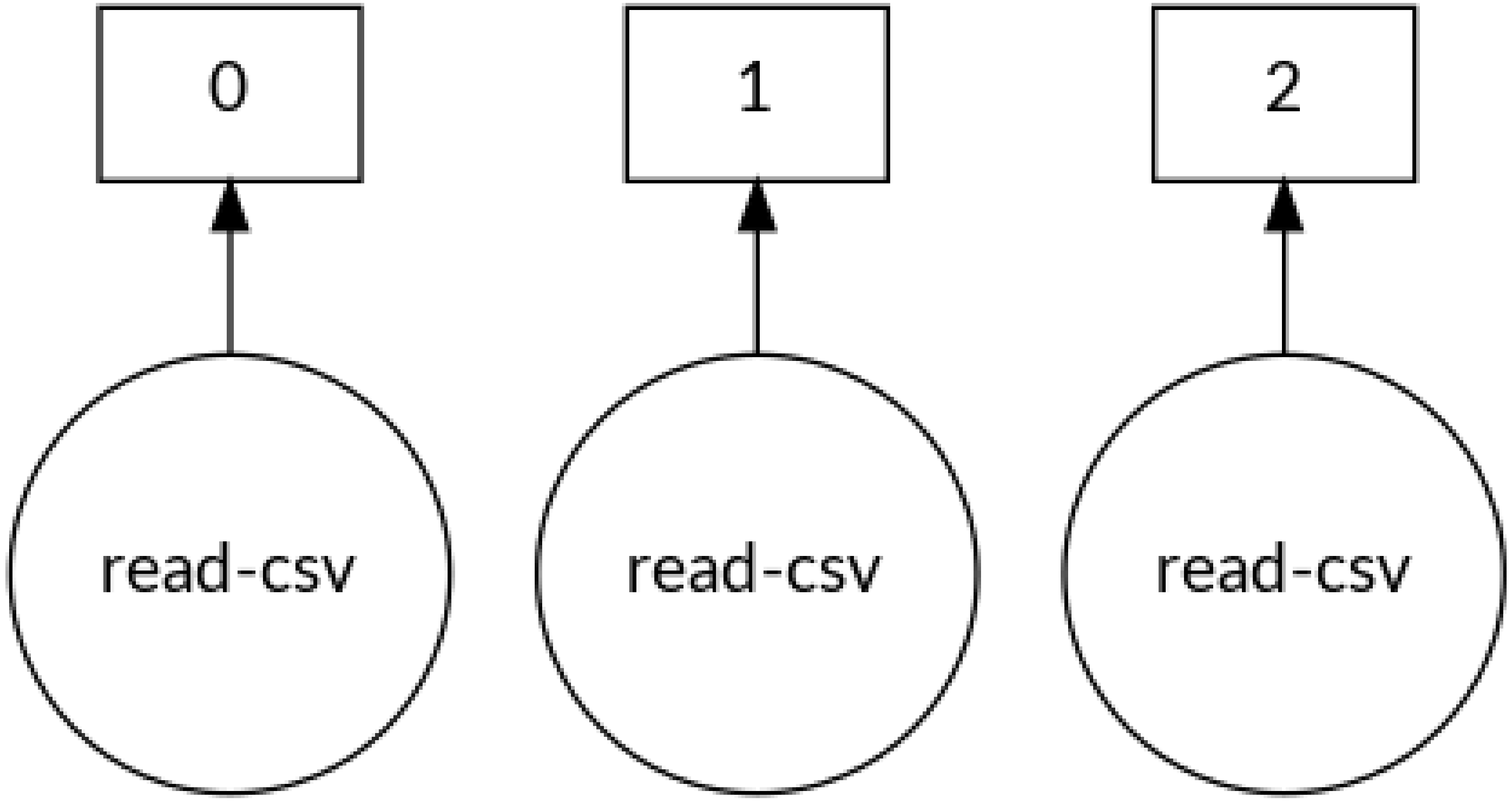
Dask DataFrames

```
print(dask_df)
```

```
Dask DataFrame Structure:
                   ID      col1      col2     col3      col4      ...
npartitions=3
                int64    object    object    int64   float64      ...
                  ...       ...       ...      ...       ...      ...
                  ...       ...       ...      ...       ...      ...
                  ...       ...       ...      ...       ...      ...
Dask Name: getitem, 3 tasks
```

Dask DataFrame task graph

```
dask.visualize(dask_df)
```



Controlling the size of blocks

```
# Set the maximum memory of a chunk
dask_df = dd.read_csv("dataset/*.csv", blocksize="10MB")
print(dask_df)
```

```
Dask DataFrame Structure:
                  ID      col1      col2     col3      col4      ...
npartitions=7
               int64    object    object    int64   float64      ...
                 ...       ...       ...      ...       ...      ...
                 ...       ...       ...      ...       ...      ...
                 ...       ...       ...      ...       ...      ...
Dask Name: getitem, 7 tasks
```

Explaining partitions

```
# Set the maximum memory of a chunk
dask_df = dd.read_csv("dataset/*.csv", blocksize="10MB")
```

Why 7 partitions?

```
size      file
  9M      dataset/chunk1.csv
 18M      dataset/chunk2.csv
 32M      dataset/chunk3.csv
```

Explaining partitions

```
# Set the maximum memory of a chunk
dask_df = dd.read_csv("dataset/*.csv", blocksize="10MB")
```

Why 7 partitions?

```
size      file
  9M      dataset/chunk1.csv    # becomes 1 partition
 18M      dataset/chunk2.csv    # becomes 2 partitions
 32M      dataset/chunk3.csv    # becomes 4 partitions
```

Analysis with Dask DataFames

- Select column

```
col1 = dask_df['col1']
```

- Assigning columns

```
dask_df['double_col1'] = 2 * col1
```

- Mathematical operations, e.g.

```
dask_df.std()
dask_df.min()
```

- Groupby

```
dask_df.groupby(col1).mean()
```

- Even fuctions you used earlier

```
dask_df.nlargest(n=3, columns='col1')
```

Datetimes and other pandas functionality

```python
import pandas as pd

# Converting string to datetime format
pd.to_datetime(pandas_df['start_date'])

# Accessing datetime attributes
pandas_df['start_date'].dt.year
pandas_df['start_date'].dt.day
pandas_df['start_date'].dt.hour
pandas_df['start_date'].dt.minute
```

```python
import dask.dataframe as dd

# Converting string to datetime format
dd.to_datetime(dask_df['start_date'])

# Accessing datetime attributes
dask_df['start_date'].dt.year
dask_df['start_date'].dt.day
dask_df['start_date'].dt.hour
dask_df['start_date'].dt.minute
```

Making results non-lazy

```
# Show 5 rows
print(dask_df.head())
```

```
         ID      double_col1      col1      col2      col3      ...
0    543795               20        10       436         0      ...
1    874535               24        12       268         0      ...
2    781326               62        31       211         0      ...
3    112457               18         9       898         1      ...
4    103256              142        71       663         0      ...
```

```
# Convert lazy Dask DataFrame to in-memory pandas DataFrame
results_df = df.compute()
```

Sending answer straight to file

```python
# 7 partitions (chunks) so 7 output files
dask_df.to_csv('answer/part-*.csv')
```

```
part-0.csv
part-1.csv
part-2.csv
part-3.csv
part-4.csv
part-5.csv
part-6.csv
```

Faster file formats - Parquet

```
# Read from parquet
dask_df = dd.read_parquet('dataset_parquet')


# Save to parquet
dask_df.to_parquet('answer_parquet')
```

- Parquet format is multiple times faster to read data from than CSV
- Can be faster to write to

Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

Multidimensional arrays

PARALLEL PROGRAMMING WITH DASK IN PYTHON

James Fulton
Climate Informatics Researcher

Types of multi-dimensional data

- Weather forecasts/observations
- 3D biomedical scans
- Satellite images
- Data from other scientific instruments

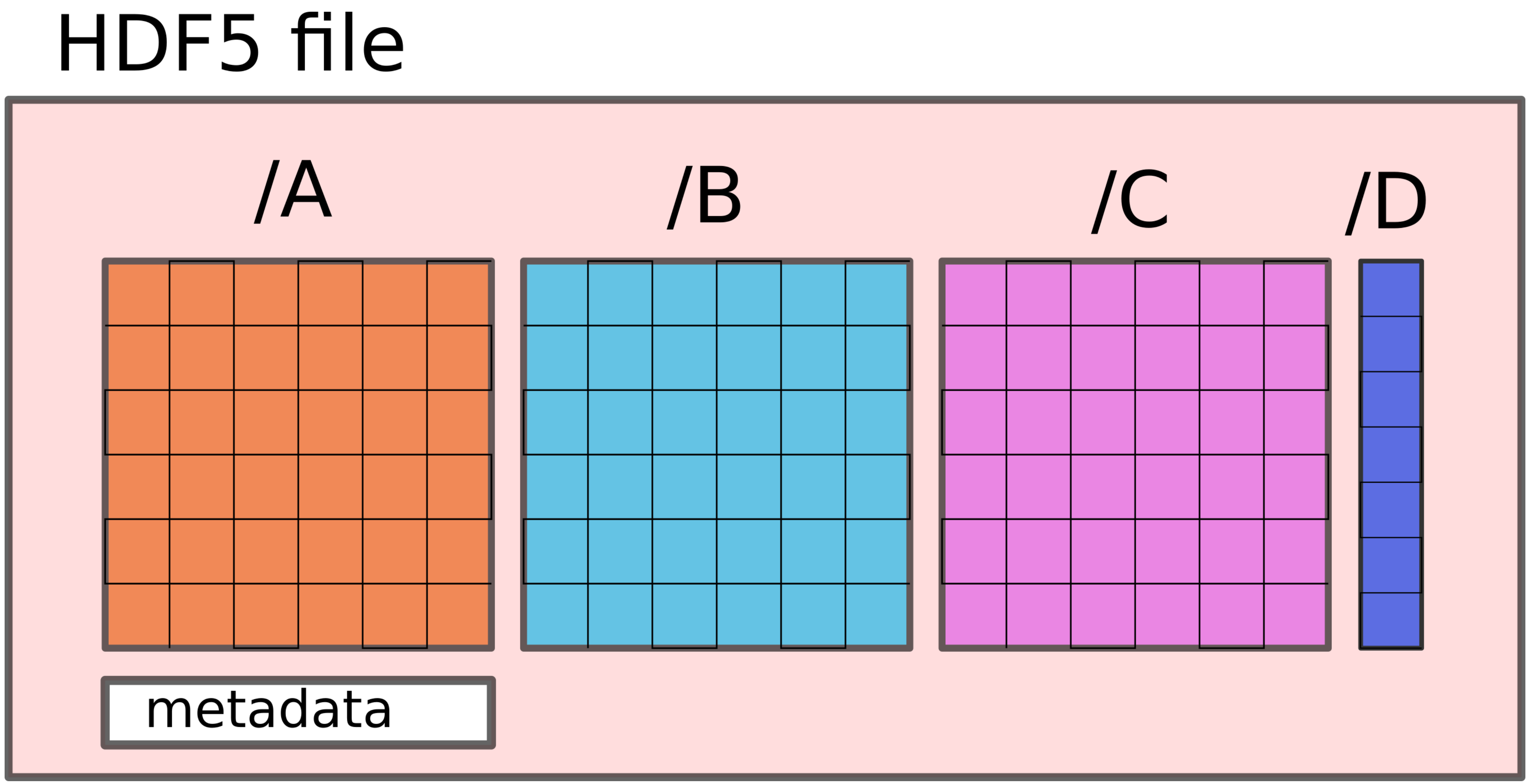
HDF5

- Hierarchical Data Format
- Stored in hierarchical format - like (sub)directories

What does an HDF5 file look like?

HDF5 file

What does an HDF5 file look like?



Navigating HDF5 files with h5py

```python
import h5py

# Open the HDF5 file
file = h5py.File('data.hdf5')

# Print the available datasets inside the file
print(file.keys())
```

```
<KeysViewHDF5 ['A', 'B', 'C', 'D']>
```

Navigating HDF5 files with h5py

```python
import h5py

# Open the HDF5 file
file = h5py.File('data.hdf5')

# Select dataset A
dataset_a = file['/A']

print(dataset_a)
```

```
<HDF5 dataset "A": shape (10000, 100, 100), type "<f4">
```

Loading from HDF5

```
import dask.array as da

# Load dataset into a Dask array
a = da.from_array(dataset_a, chunks=(100, 20, 20))

print(a)
```

```
dask.array<array, shape=(10000, 100, 100), dtype=float32, chunksize=(100, 20, 20),
    chunktype=numpy.ndarray>
```

Zarr

- Hierarchical dataset like HDF5
- Designed to be chunked
- Good for streaming over cloud computing services like AWS, Google Cloud, etc.
- Navigable file structure

Loading from Zarr

```python
import dask.array as da

a = da.from_zarr("dataset.zarr", component="A")

print(a)
```

```
dask.array<from-zarr, shape=(10000, 100, 100), dtype=float32,
    chunksize=(100, 20, 20), chunktype=numpy.ndarray>
```

Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

Xarray

PARALLEL PROGRAMMING WITH DASK IN PYTHON

James Fulton
Climate Informatics Researcher

Xarray - like pandas in more dimensions

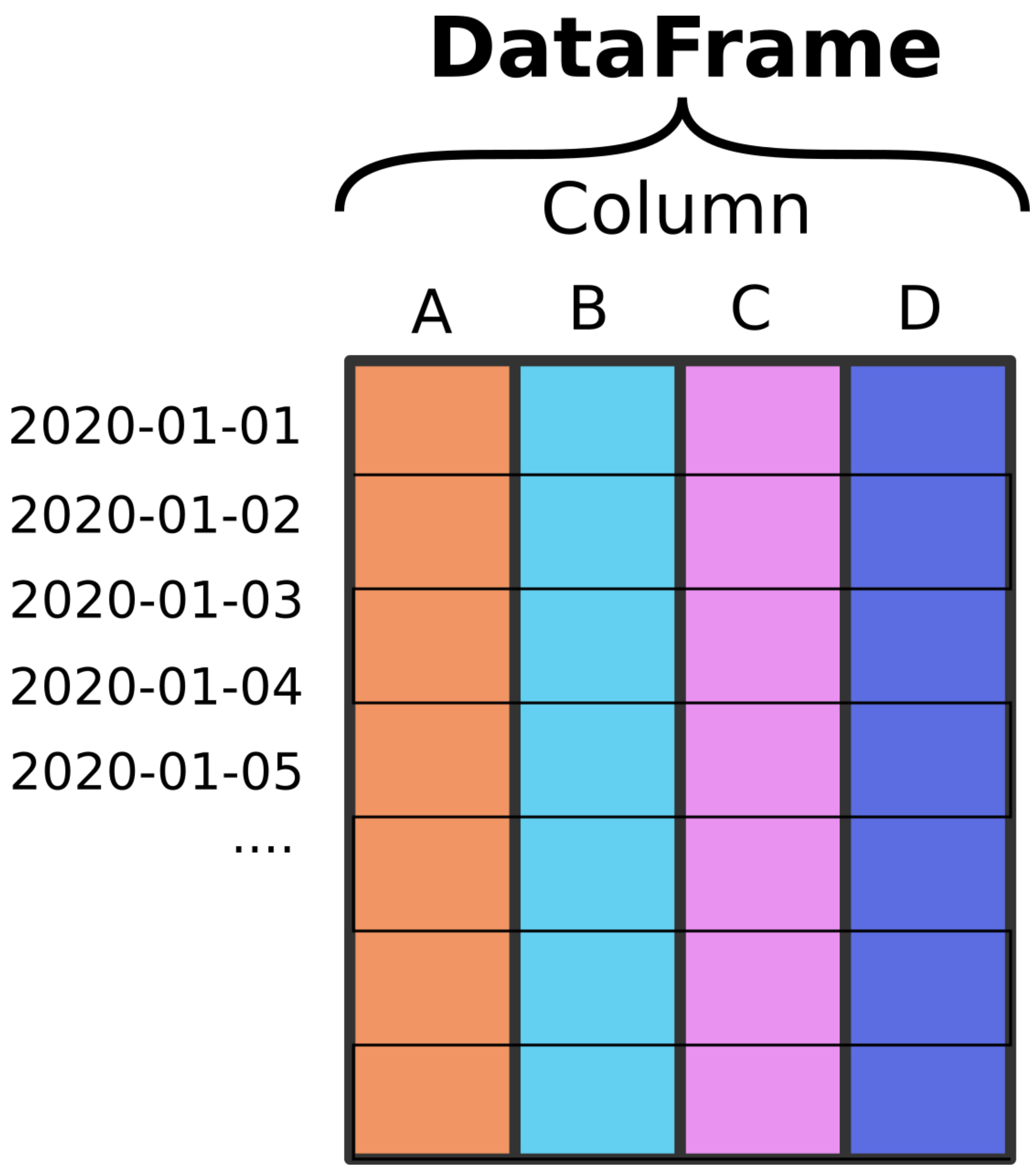
pandas

- Applies index labels to tabular data

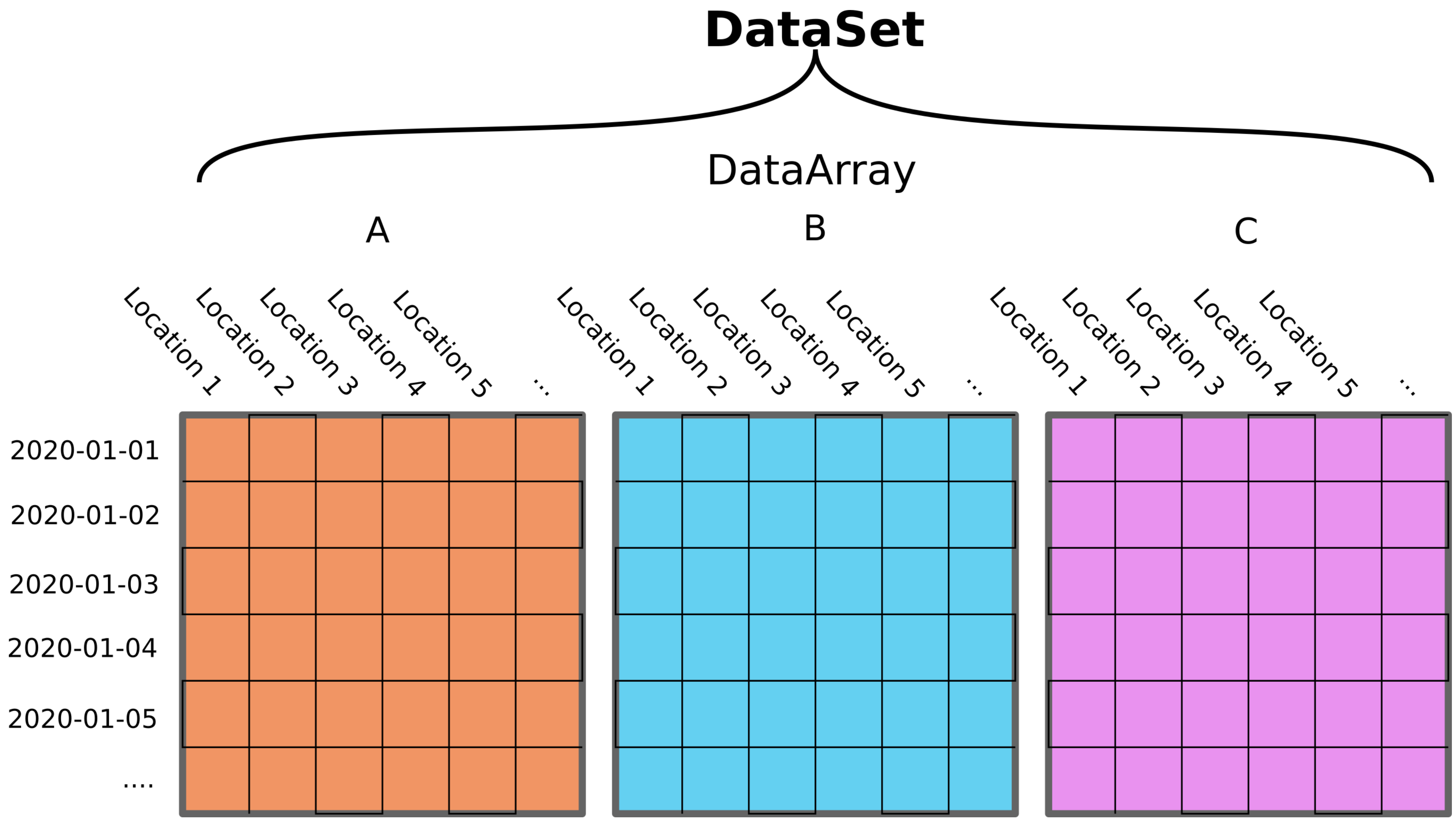
Xarray

- Applies index labels to high dimensional arrays

DataFrame


DataFrame
Column
A
B
C
D
2020-01-01
2020-01-02
2020-01-03
2020-01-04
2020-01-05
....

DataSet


DataSet
DataArray
A
B
C
Location 1
Location 2
Location 3
Location 4
Location 5
...
Location 1
Location 2
Location 3
Location 4
Location 5
...
Location 1
Location 2
Location 3
Location 4
Location 5
...
2020-01-01
2020-01-02
2020-01-03
2020-01-04
2020-01-05
....

Loading a DataSet from Zarr

```
import xarray as xr
ds = xr.open_zarr("data/era_eu.zarr")


print(ds)
```

```
<xarray.Dataset>
Dimensions:  (lat: 30, lon: 45, time: 504)
Coordinates:
  * lat      (lat) float64 35.5 36.5 37.5 38.5 39.5 ... 60.5 61.5 62.5 63.5 64.5
  * lon      (lon) float64 -14.5 -13.5 -12.5 -11.5 -10.5 ... 26.5 27.5 28.5 29.5
  * time     (time) datetime64[ns] 1979-05-31 1979-06-30 ... 2021-04-30
Data variables:
    precip   (time, lat, lon) float32 dask.array<chunksize=(12, 15, 15), ... >
    temp     (time, lat, lon) float32 dask.array<chunksize=(12, 15, 15), ... >
```

DataFrame vs. DataSet

pandas DataFrame

```
# Select a particular date
df.loc['2020-01-01']

# Select by index number
df.iloc[0]

# Select column
df['column1']
```

Dask DataSet

```
# Select a particular date
ds.sel(time='2020-01-01')

# Select by index number
ds.isel(time=0)

# Select variable
ds['variable1']
```

DataFrame vs. DataSet

pandas DataFrame

```
# Perform mathematical operations
df.mean()



# Groupby and mean
df.groupby(df['time'].dt.year).mean()

# Rolling mean
rolling_mean = df.rolling(5).mean()
```

Dask DataSet

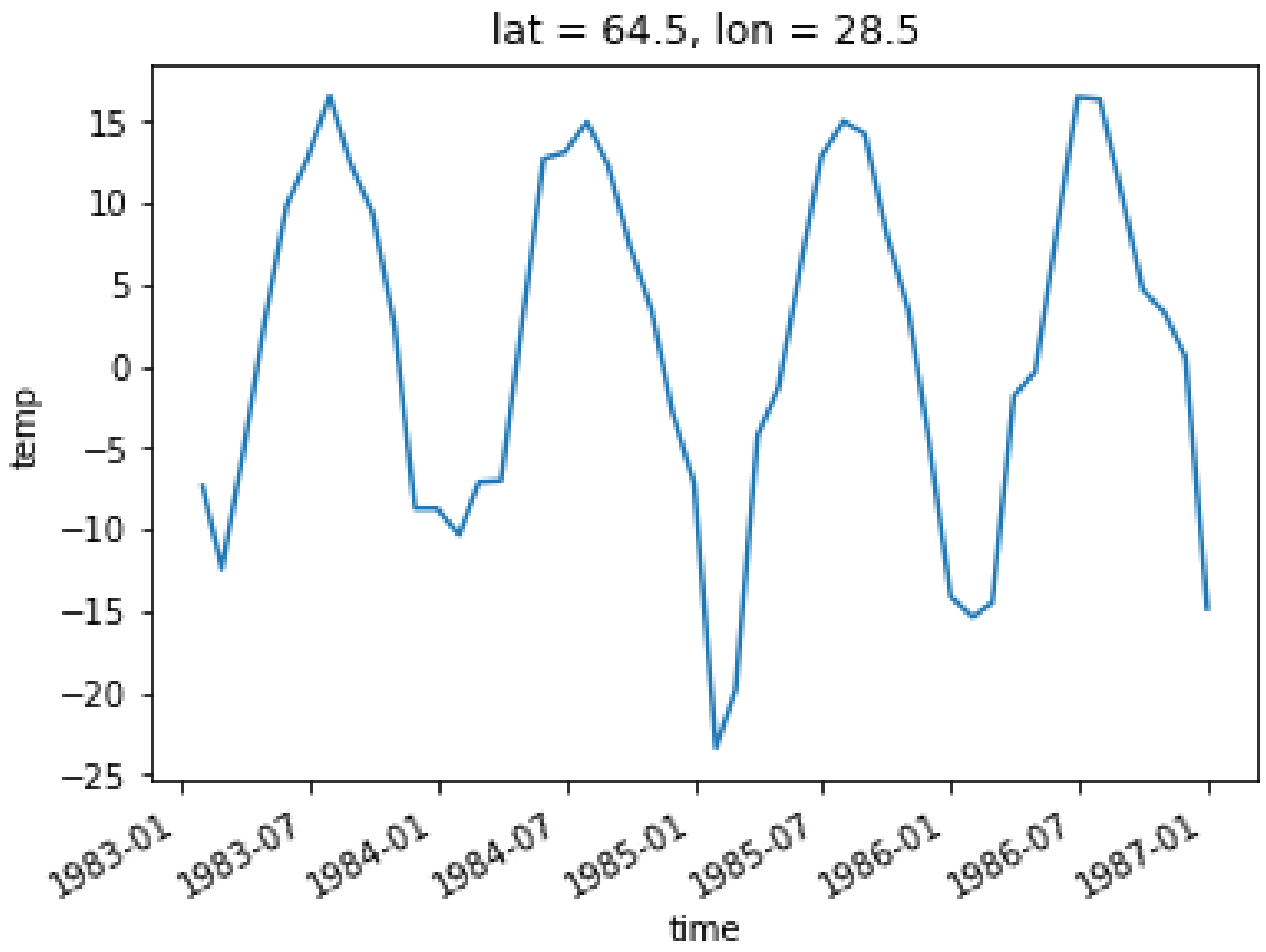
```
# Perform mathematical operations
ds.mean()
ds.mean(dim='dim1')
ds.mean(dim=('dim1', 'dim2'))

# Groupby and mean
ds.groupby(ds['time'].dt.year).mean()

# Rolling mean
rolling_mean = ds.rolling(dim1=5).mean()
rolling_mean.compute()
```

Plotting

```
ds['variable'].plot()
```

- **Makes a line plot if 1D**
- Makes a heatmap if 2D
- Makes a histogram if 3D+

Example

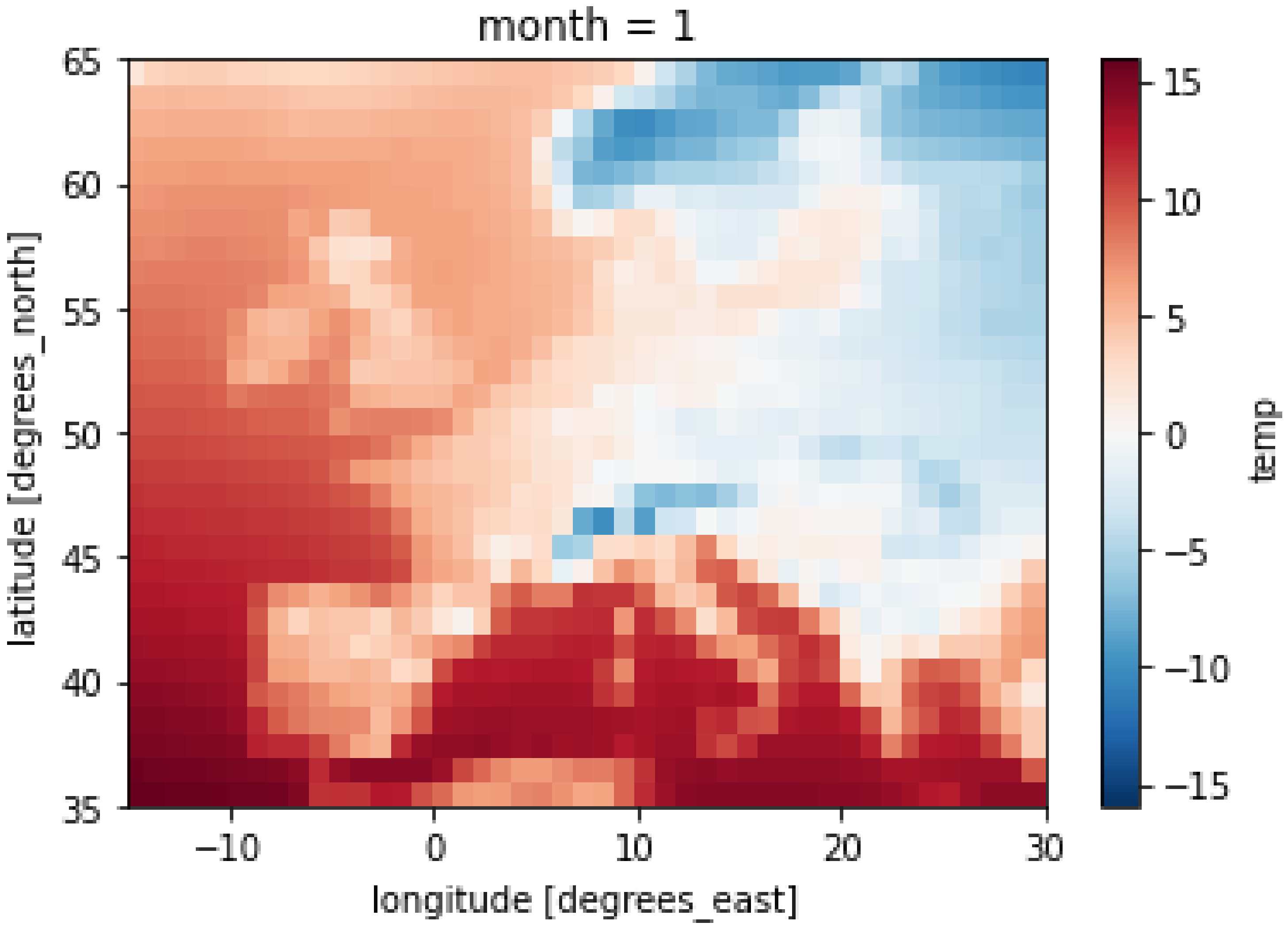


Plotting

```
ds['variable'].plot()
```

- Makes a line plot if 1D
- **Makes a heatmap if 2D**
- Makes a histogram if 3D+

Example

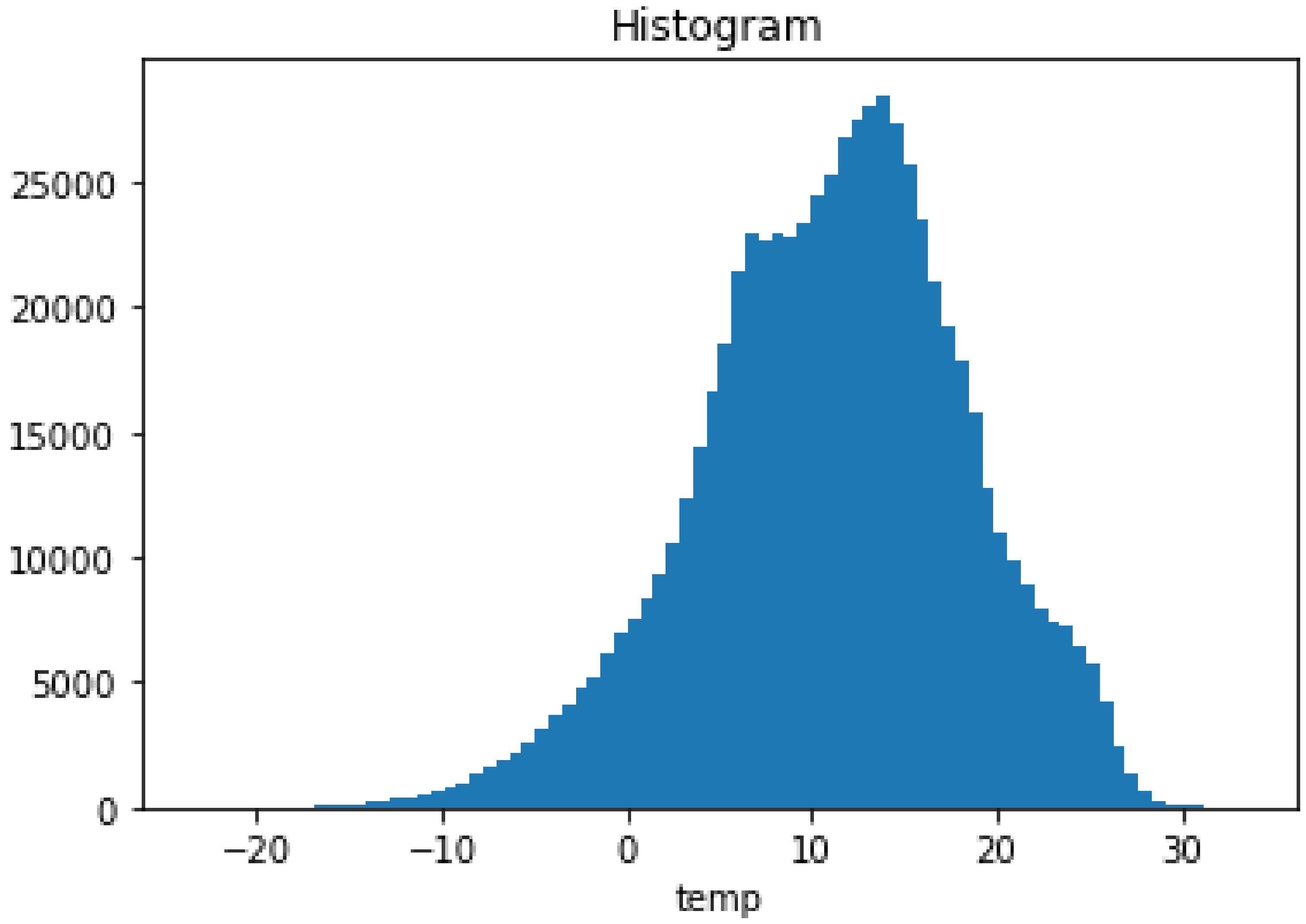


Plotting

```
ds['variable'].plot()
```

- Makes a line plot if 1D
- Makes a heatmap if 2D
- **Makes a histogram if 3D+**

Example



Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON