Using processes and threads

PARALLEL PROGRAMMING WITH DASK IN PYTHON

James Fulton
Climate Informatics Researcher

Dask default scheduler

Threads

- Dask arrays
- Dask DataFrames
- Delayed pipelines created with `dask.delayed()`

Processes

- Dask bags

Choosing the scheduler

```
# Use default
result = x.compute()
result = dask.compute(x)

# Use threads
result = x.compute(scheduler='threads')
result = dask.compute(x, scheduler='threads')

# Use processes
result = x.compute(scheduler='processes')
result = dask.compute(x, scheduler='processes')
```

Recap - threads vs. processes

Threads

- Are very fast to initiate
- No need to transfer data to them
- Are limited by the GIL, which allows one thread to read the code at once

Processes

- Take time to set up
- Slow to transfer data to
- Each have their own GIL and so don't need to take turns reading the code

Creating a local cluster

```
from dask.distributed import LocalCluster

cluster = LocalCluster(
    processes=True,
    n_workers=2,
    threads_per_worker=2
)

print(cluster)
```

```
LocalCluster(..., workers=2, threads=4, memory=31.38 GiB)
```

Creating a local cluster

```
from dask.distributed import LocalCluster

cluster = LocalCluster(
    processes=False,
    n_workers=2,
    threads_per_worker=2
)

print(cluster)
```

```
LocalCluster(..., workers=2, threads=4, memory=31.38 GiB)
```

Simple local cluster

```
cluster = LocalCluster(processes=True)

print(cluster)
```

```
LocalCluster(..., workers=4 threads=8, memory=31.38 GiB)
```

```
cluster = LocalCluster(processes=False)

print(cluster)
```

```
LocalCluster(..., workers=1 threads=8, memory=31.38 GiB)
```

Creating a client

```
from dask.distributed import Client, LocalCluster
cluster = LocalCluster(
    processes=True,
    n_workers=4,
    threads_per_worker=2
)

client = Client(cluster)
print(client)
```

```
<Client: 'tcp://127.0.0.1:61391' processes=4 threads=8, memory=31.38 GiB>
```

Creating a client easily

Create cluster then pass it into client

```
cluster = LocalCluster(
    processes=True,
    n_workers=4,
    threads_per_worker=2
)

client = Client(cluster)

print(client)
```

```
<Client: ... processes=4 threads=8, ...>
```

Create client which will create its own cluster

```
client = Client(
    processes=True,
    n_workers=4,
    threads_per_worker=2
)



print(client)
```

```
<Client: ... processes=4 threads=8, ...>
```

Using the cluster

```
client = Client(processes=True)

# Default uses the client
result = x.compute()

# Can still change to other schedulers
result = x.compute(scheduler='threads')

# Can explicitly use client
result = client.compute(x)
```

Other kinds of cluster

- `LocalCluster()` - A cluster on your computer.
- Other cluster types split computation across different computers

Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

Training machine learning models on big datasets

PARALLEL PROGRAMMING WITH DASK IN PYTHON

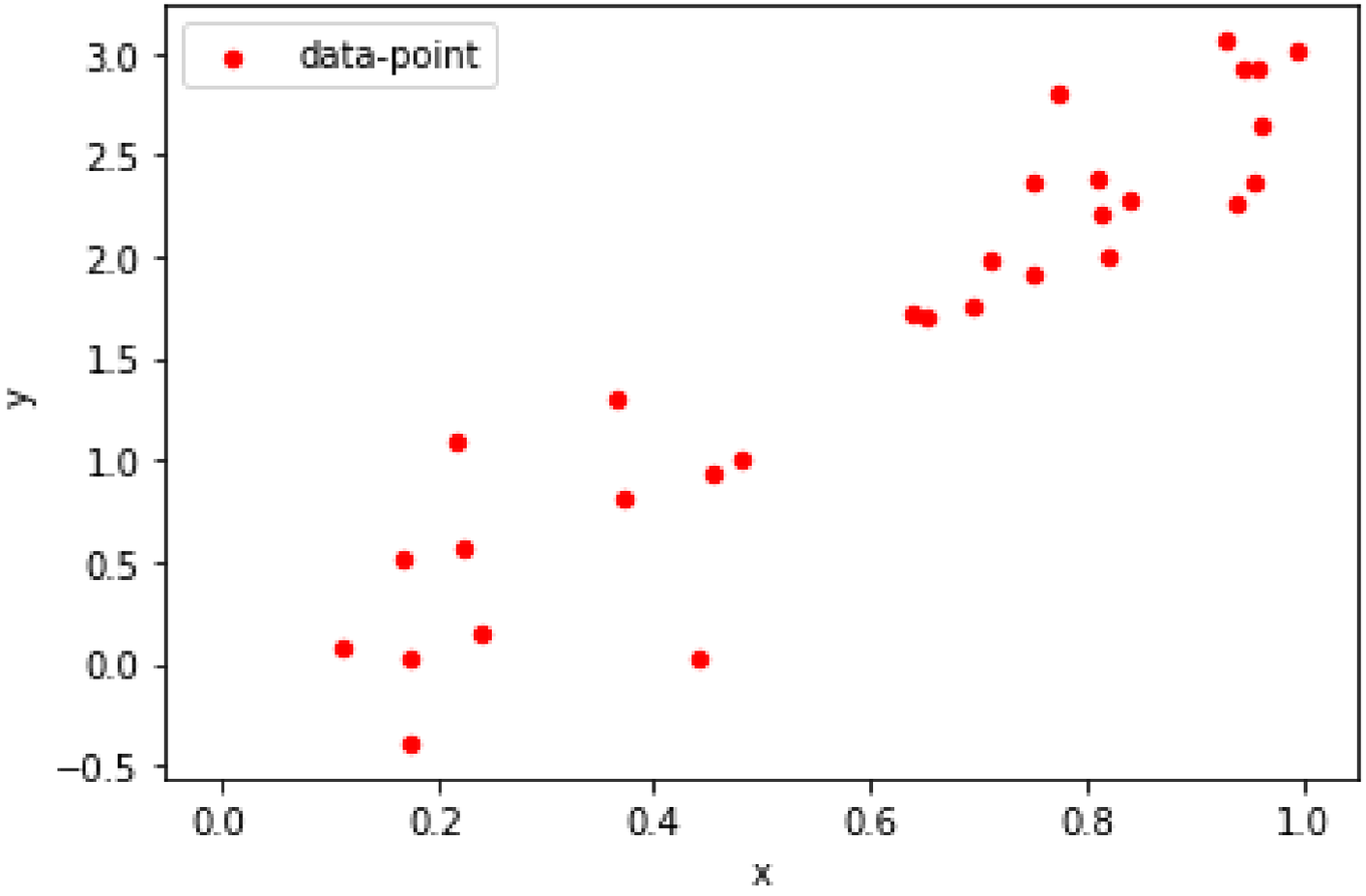
James Fulton
Climate Informatics Researcher

Dask-ML

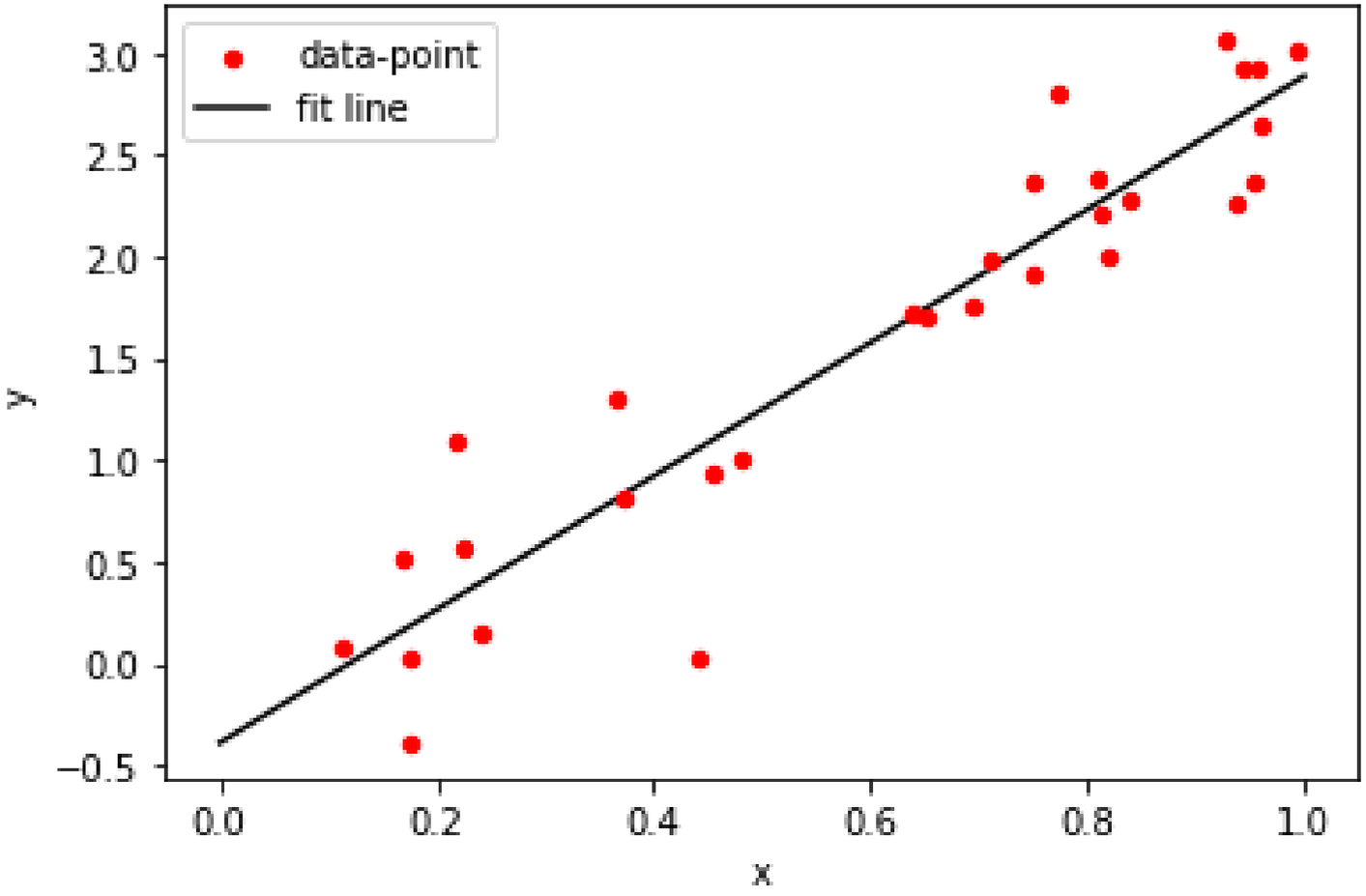
```
import dask_ml
```

- Speeds up machine learning tasks

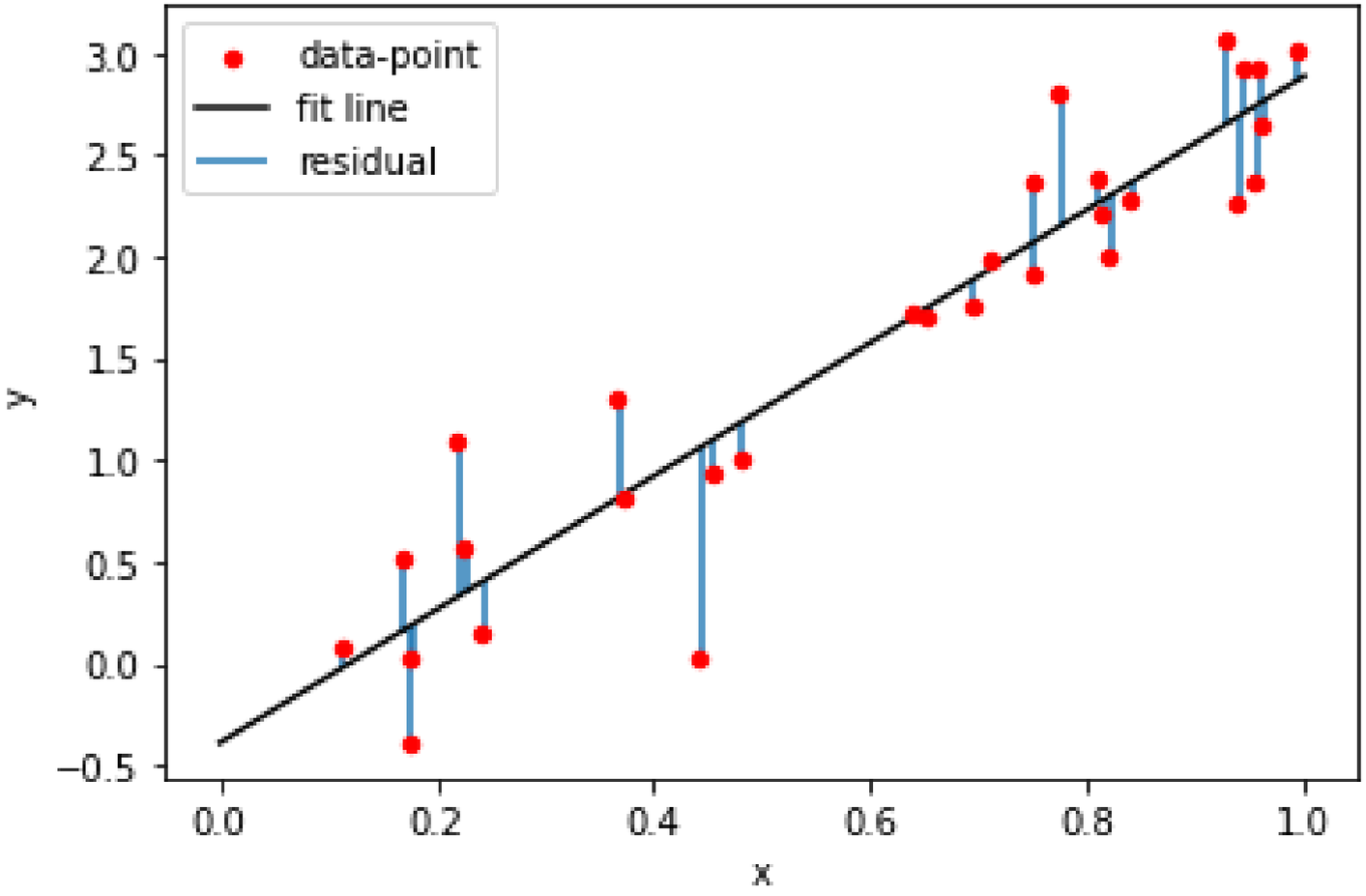
Linear regression



Linear regression



Linear regression



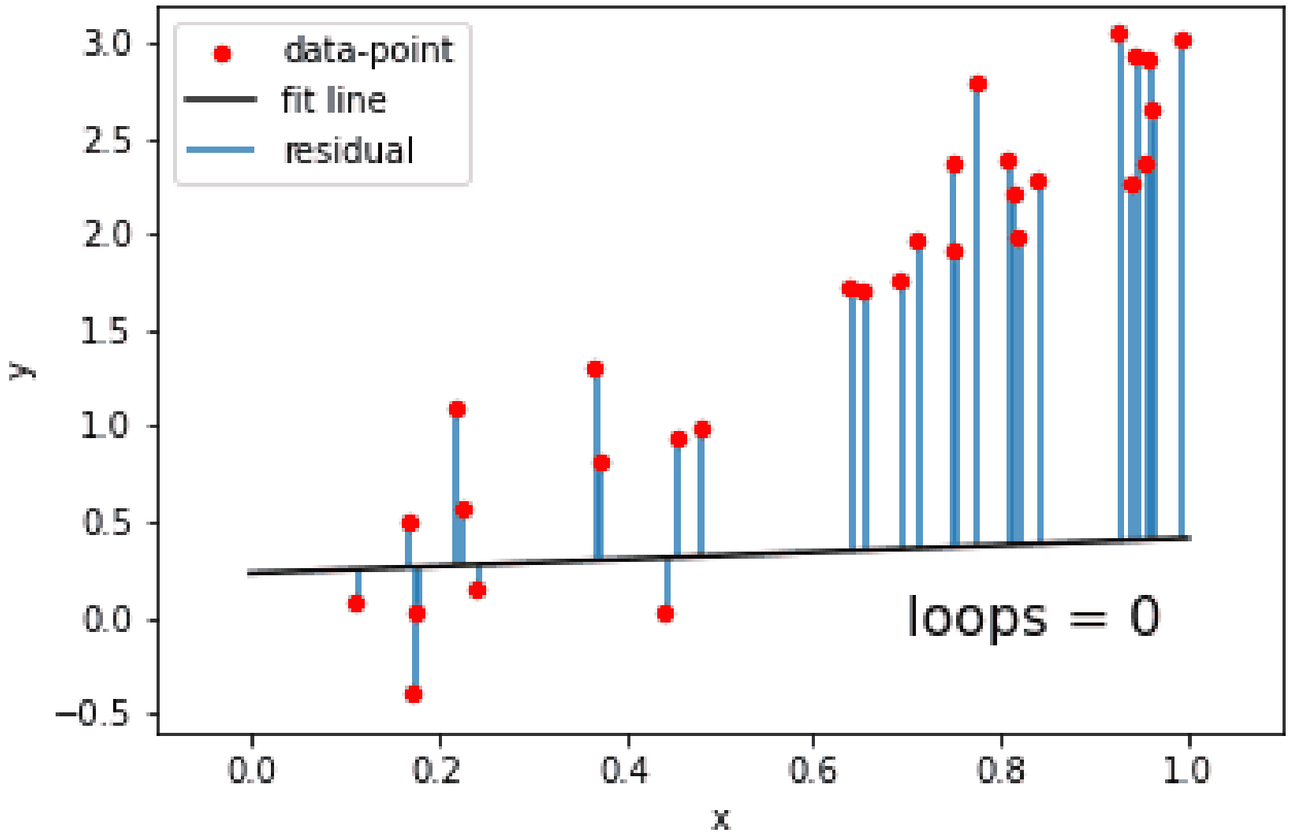
Fitting a linear regression model

```
# Import regression model
from sklearn.linear_model import SGDRegressor

# Create instance of model
model = SGDRegressor()

# Fit model to data
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)
```

Using a scikit-learn model with Dask

```
# Import regression model
from sklearn.linear_model import SGDRegressor

# Create instance of model
model = SGDRegressor()

# Import Dask-ML wrapper for model
from dask_ml.wrappers import Incremental

# Wrap model
dask_model = Incremental(model, scoring='neg_mean_squared_error')

# Fit on Dask DataFrames or arrays
dask_model.fit(dask_X, dask_y)  # not lazy
```

Fitting takes multiple iterations



Training an Incremental model

```
# Loop through data multiple times
for i in range(10):
    dask_model.partial_fit(dask_X, dask_y)  # not lazy
```

Generating predictions

```
y_pred = dask_model.predict(dask_X)


print(y_pred)
```

```
dask.array<_predict, shape=(nan,), dtype=int64, chunksize=(nan,), chunktype=...>
```

```
print(y_pred.compute())
```

```
array([0.465557, 0.905675, 0.285214, ..., 0.249454, 0.559624, 0.823475])
```

Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

Machine learning with big datasets

PARALLEL PROGRAMMING WITH DASK IN PYTHON

James Fulton
Climate Informatics Researcher

datacamp

Loading and preprocessing data

```
# Load tabular dataset
import dask.dataframe as dd
dask_df = dd.read_parquet("dataset_parquet")
```

```
X = dask_df[['feature1', 'feature2', 'feature3']]
y = dask_df['target_column']
```

```
from dask_ml.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X)  # This is not lazy

standardized_X = scaler.transform(X)  # This is lazy
```

Train-test split

```
from dask_ml.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.2)
print(X_train)
```

```
Dask DataFrame Structure:
               feature1    feature2    feature3
npartitions=7
                 int64     float64     float64
                   ...         ...         ...
```

Scoring

```
# Test the fit model on training data
train_score = dask_model.score(X_train, y_train) # Not lazy
print(train_score)
```

```
-0.12321
```

```
# Test the fit model on testing data
test_score = dask_model.score(X_test, y_test)  # Not lazy
print(test_score)
```

```
-0.23453
```

Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

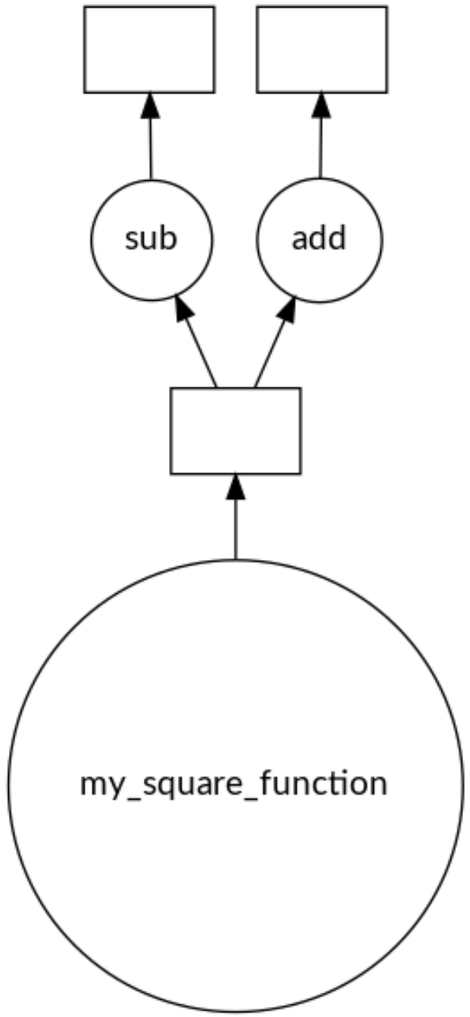
Wrap-up

PARALLEL PROGRAMMING WITH DASK IN PYTHON

James Fulton
Climate Informatics Researcher

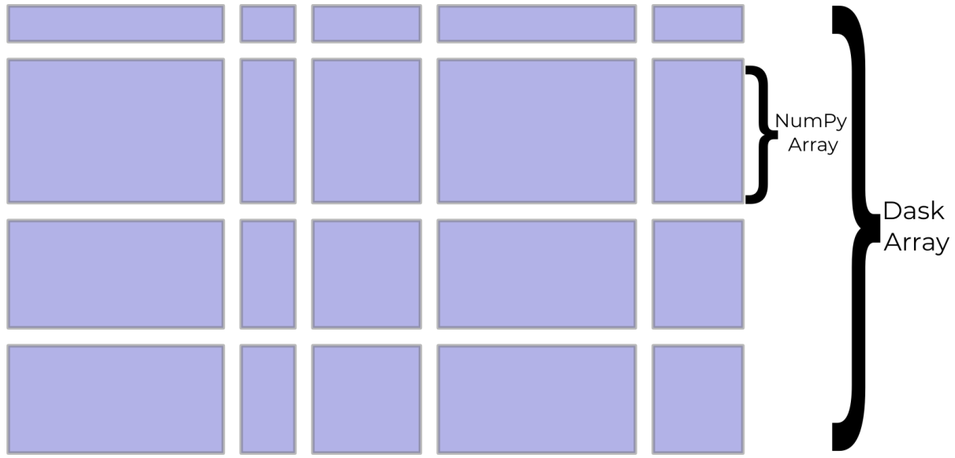
Recap - chapter 1

- Task graphs
- Lazy evaluation
- Threads vs. processes
- `dask.delayed()`



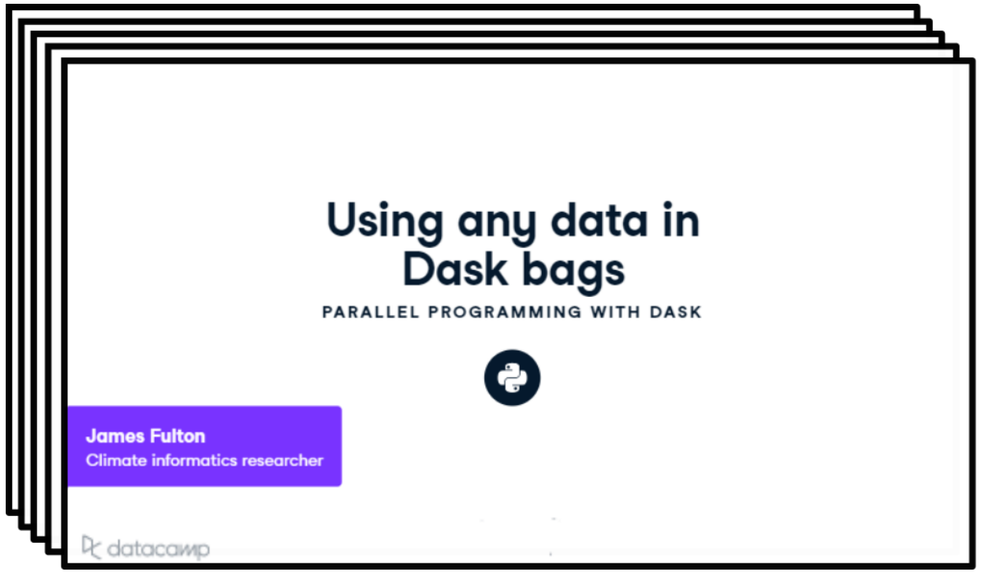
Recap - chapter 2

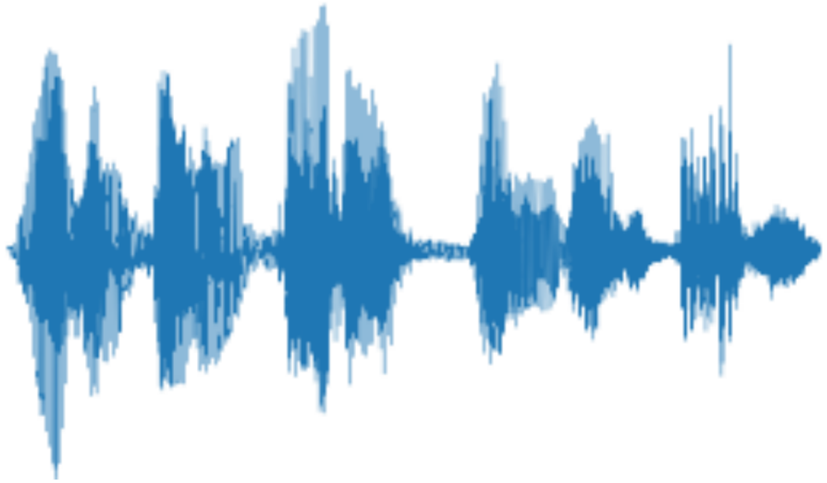
- Analyzing big structured data
- Dask arrays
- Dask DataFrames
- Advanced data formats: `h5py` , `zarr` , `parquet`
- `pandas` & `numpy` -> `dask`



Recap - chapter 3

- Dask bags for big unstructured and semi-structured data
- e.g., JSON, text, and audio





Recap - chapter 4

- Using `LocalCluster` and other clusters
- Dask-ML
- Training ML on big data
- Lazily preprocessing big data

Next steps

- A wider range of functions for
  - Dask arrays
  - Dask DataFrames
  - Dask bags
- Documentation at
  - **https://docs.dask.org**

Congratulations!

PARALLEL PROGRAMMING WITH DASK IN PYTHON