Distributed computing and Big Data with Dask

devoteam

What is Dask?

- ➢ Flexible open-source Python library for parallel computing (2015);
- ➢ Built by the NumPy, Pandas, Jupyter, Scikit-Learn developer community;
- ➢ Lightweight and easily installed library;
- ➢ Ability to scale by deploying clusters (from single to thousand nodes);
- ➢ Handles out-of-memory data for big data analysis.

Main source : https://docs.dask.org/

Roadmap

Dask Concepts

Dask Arrays

Dask DataFrames

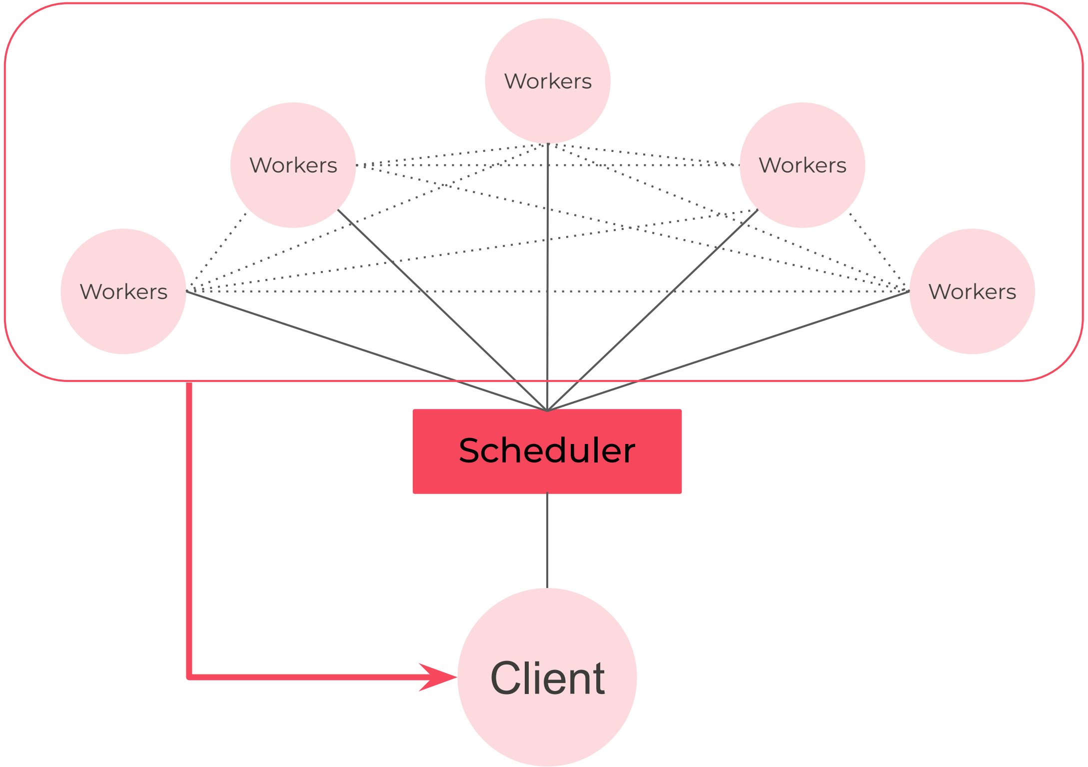
Dask Delayed

Dask Futures

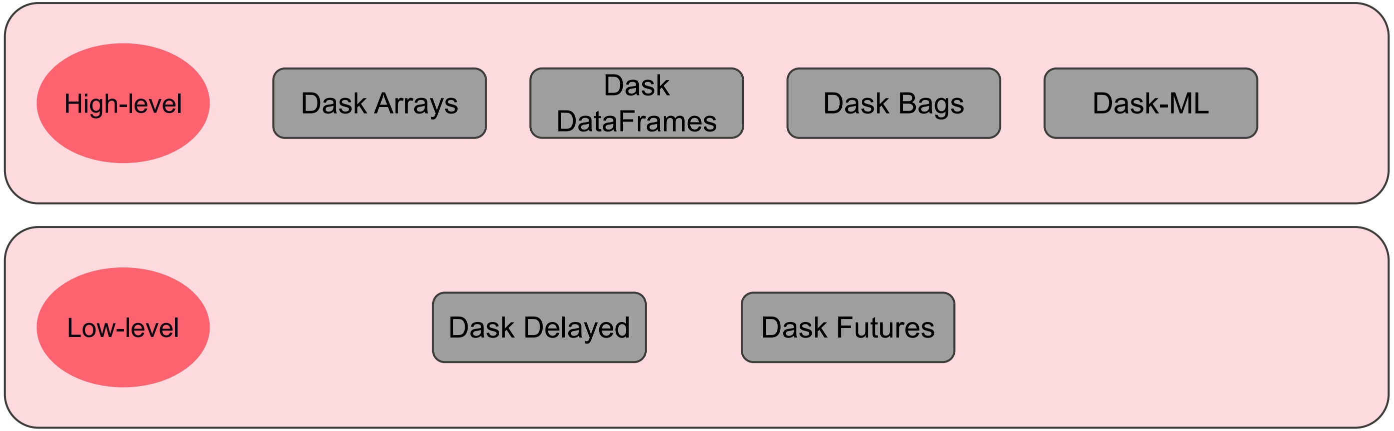
Dask ML

Deploy Clusters

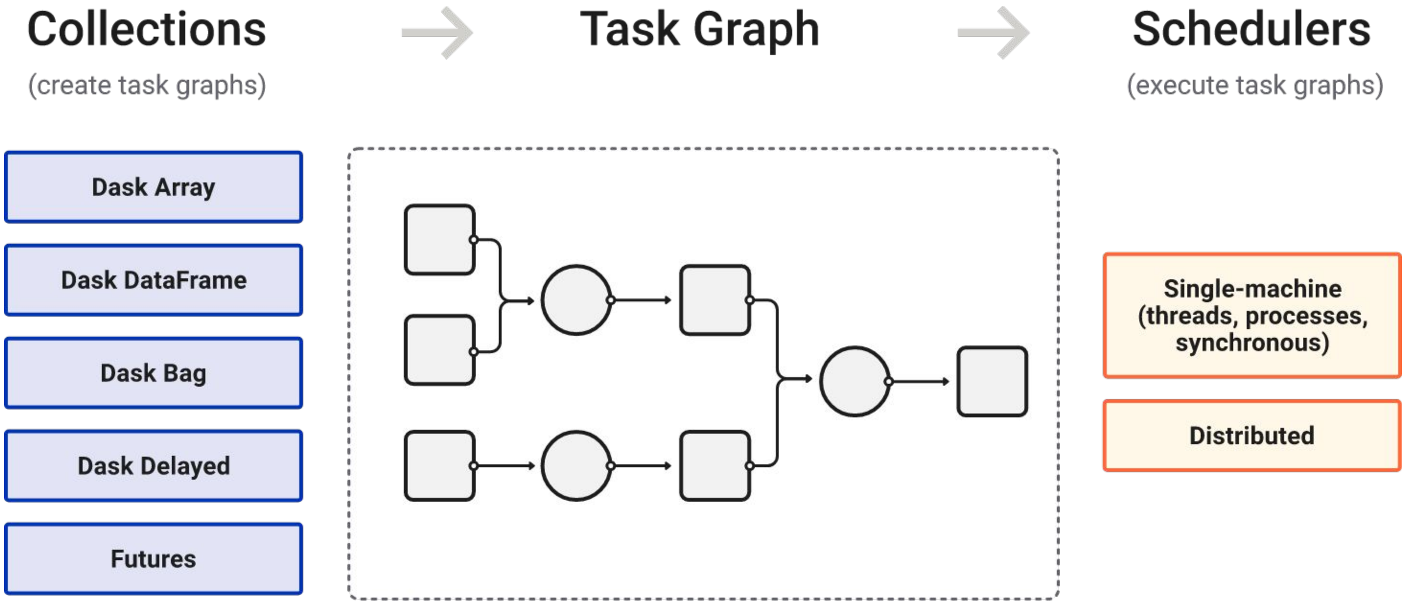
Dask concept: Architecture



Dask concept: Collections



Dask concept: Scheduler



Notebooks

Dask Arrays

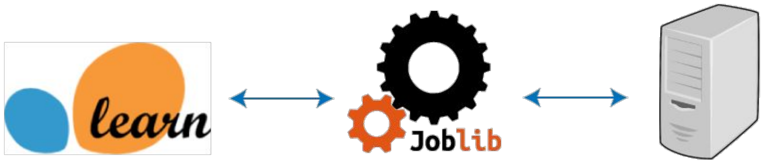
Dask DataFrames

Dask Delayed

Dask Futures

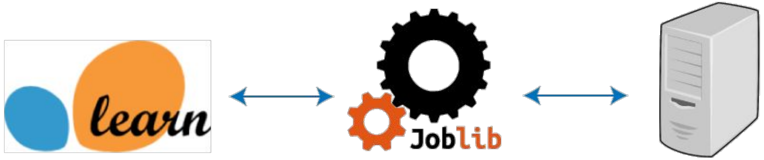
Dask ML

Provides scalable machine learning algorithms compatible with scikit-learn



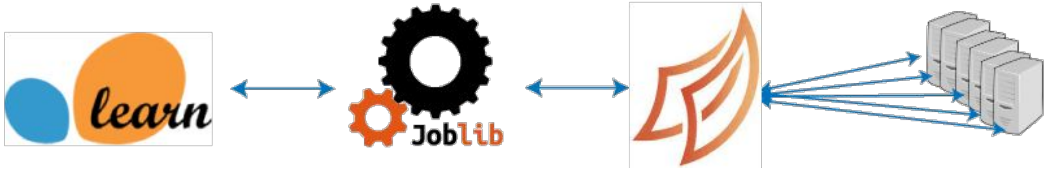
Scikit-learn

Provides scalable machine learning algorithms compatible with scikit-learn



Scikit-learn

```python
from dask.distributed import Client
import joblib

client = Client()                     # create local cluster
# client = Client("scheduler-address") # or remote cluster

with joblib.parallel_backend('dask'):
    # Your scikit-learn code
```



Scikit-learn + Dask

Use cases

- Pre-processing (dask_ml.preprocessing):
  - *MinMaxscaler, Labelencoder, OneHotEncoder, PolynomialFeatures* ...
- Cross Validation (for instance extension to dask arrays):
  - *dask_ml.model_selection.train_test_split()*
- Hyper Parameter Search optimization.
- Generalized Linear Models:
  - *Linear Regression, Logistic Regression, Poisson Regression* ...
- Clustering:
  - *KMeans, Spectral Clustering* ...

Dask Arrays — Dask DataFrames — Dask Delayed — Dask Futures — Dask ML — Deploy Clusters

Deploy Clusters anywhere!

- Locally (*dask.distributed.Client()*) ;

Deploy Clusters anywhere!

- ➢ Locally (*dask.distributed.Client()*) ;
- ➢ High Performance Computers (HPC) with job schedulers (SGE, SLURM, PBS);

```
# SGE cluster
from dask.distributed import Client
from dask_jobqueue import SGECluster

cluster = SGECluster(name="dask-worker", walltime="12:00:00",
                     memory="4GB", death_timeout=240, project="P_ztf",
                     resource_spec="sps=1", local_directory="$TMPDIR",
                     cores=1, processes=1)


cluster.scale(10) # How many workers?
client = Client(cluster)
```

Deploy Clusters anywhere!

➢ Locally (*dask.distributed.Client()*) ;

➢ High Performance Computers (HPC) with job schedulers (SGE, SLURM, PBS);

➢ Kubernetes (Native, Helm);

```
from dask_kubernetes.operator import KubeCluster, make_cluster_spec

config = {
    "name": "foo",
    "n_workers": 2,
    "resources":{"requests": {"memory": "2Gi"}, "limits": {"memory": "64Gi"}}
}

cluster = KubeCluster(**config)
# is equivalent to
cluster = KubeCluster(custom_cluster_spec=make_cluster_spec(**config))
```

Deploy Clusters anywhere!

- ➢ Locally (*dask.distributed.Client()*) ;
- ➢ High Performance Computers (HPC) with job schedulers (SGE, SLURM, PBS);
- ➢ Kubernetes (Native, Helm);
- ➢ Cloud providers (AWS, GCP, Azure, …)

```
from dask_cloudprovider.azure import AzureVMCluster
cluster = AzureVMCluster(resource_group="<resource group>",
                         vnet="<vnet>",
                         security_group="<security group>",
                         n_workers=1)

from dask.distributed import Client
client = Client(cluster)
```

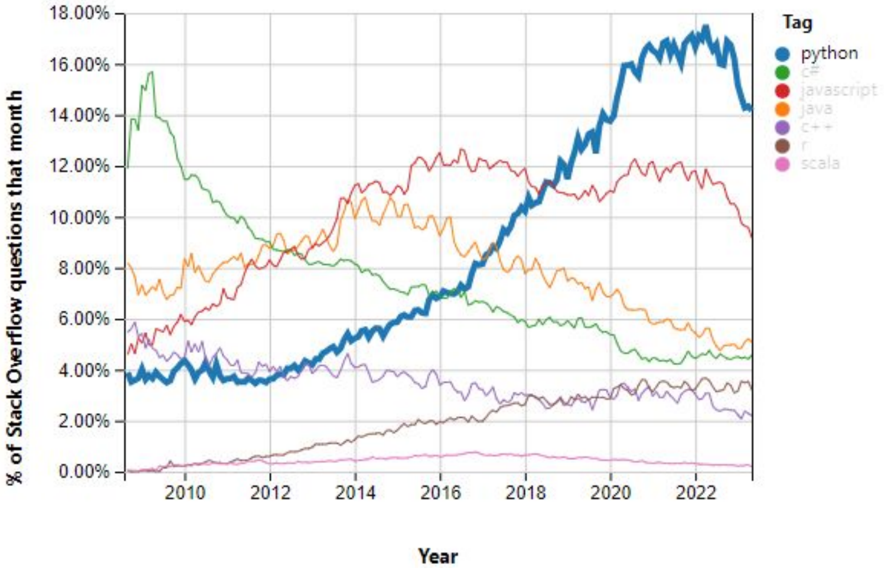
Conclusion

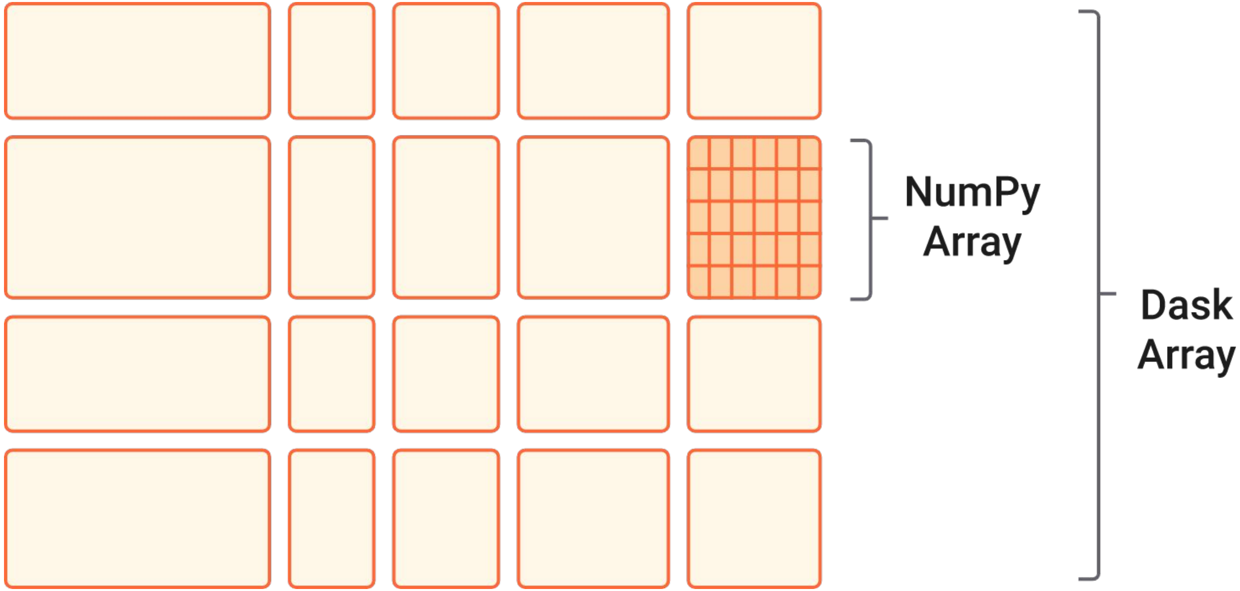
- Pure Python library;
- Numpy and Pandas APIs, familiar for Python users;
- Distributed computing (locally and on clusters);
- Allowing Big Data analysis;
- Highly flexible use thanks to Delayed and Futures;

No Notebooks Version

What is Dask?

➢ Flexible open-source Python library for parallel computing (2015);





Parallel Numpy:

- Single array chunked into smaller ones;
- Load of array larger than RAM;
- Computation optimization.

Highly use in:

- Science (astronomy, oceanography...);
- Large scale imaging;
- Numerical algorithms
- ...

```python
import numpy as np
```

```python
size=2000
```

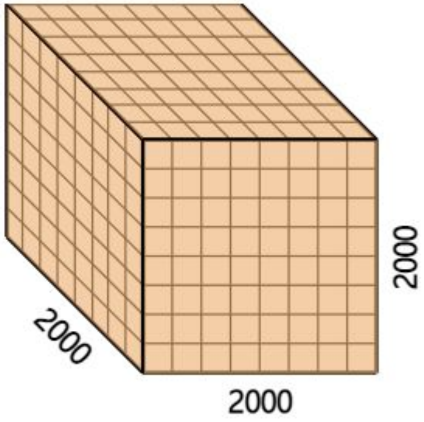
```python
arr = np.random.random((size,size,size))
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6680\925286073.py in <module>
----> 1 arr = np.random.random((size,size,size))

mtrand.pyx in numpy.random.mtrand.RandomState.random()

mtrand.pyx in numpy.random.mtrand.RandomState.random_sample()

_common.pyx in numpy.random._common.double_fill()

MemoryError: Unable to allocate 59.6 GiB for an array with shape (2000, 2000, 2000) and data type float64
```

```python
import dask.array as da
```

```python
chunk = 'auto'
x = da.random.random((size,size,size), chunks=(chunk,chunk,chunk))
x
```

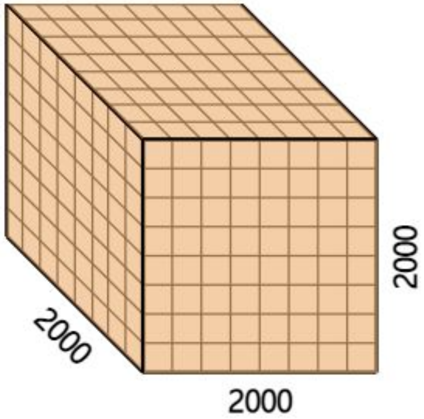
	Array	Chunk
Bytes	59.60 GiB	119.21 MiB
Shape	(2000, 2000, 2000)	(250, 250, 250)
Count	512 Tasks	512 Chunks
Type	float64	numpy.ndarray



```python
import dask.array as da
```

```python
chunk = 'auto'
x = da.random.random((size,size,size), chunks=(chunk,chunk,chunk))
x
```

	Array	Chunk
Bytes	59.60 GiB	119.21 MiB
Shape	(2000, 2000, 2000)	(250, 250, 250)
Count	512 Tasks	512 Chunks
Type	float64	numpy.ndarray



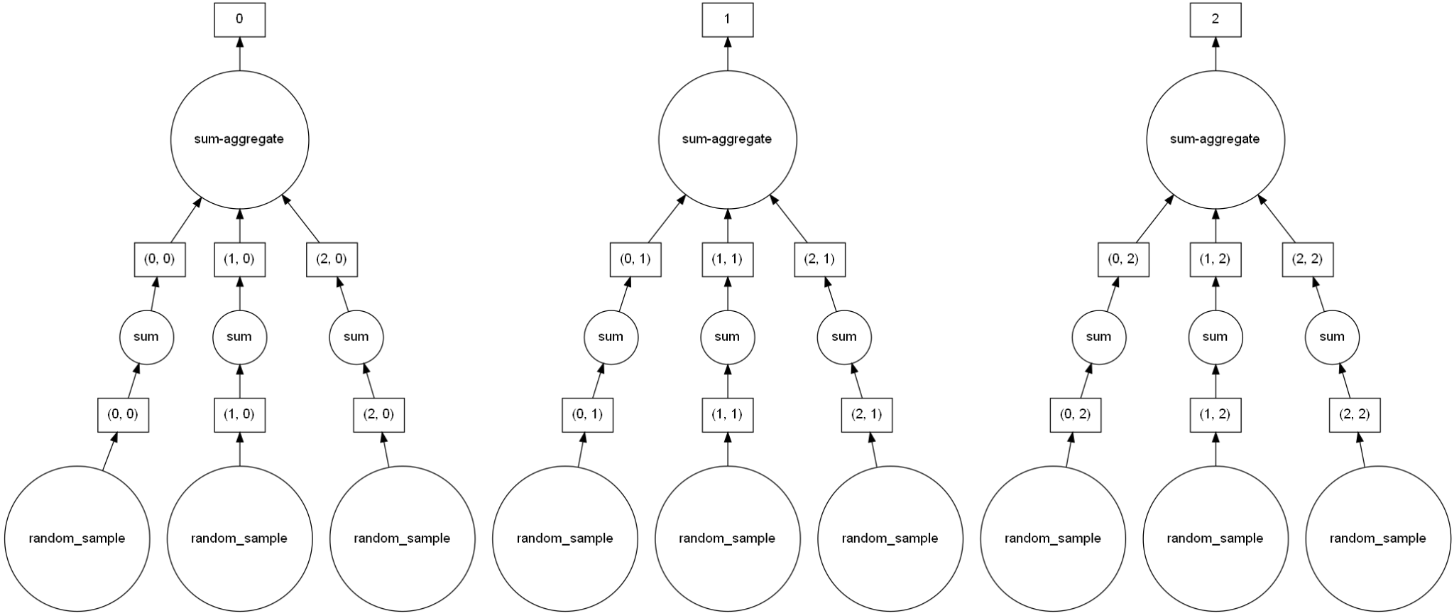

```python
m = x.mean()
```

```python
%%time
m.compute()
```

```
Wall time: 21.3 s
0.4999996563332088
```

```python
x = da.random.random((15, 15), chunks=(5,5))
comp = x.sum(axis=0)
```

```python
comp.visualize(bgcolor='transparent')
```

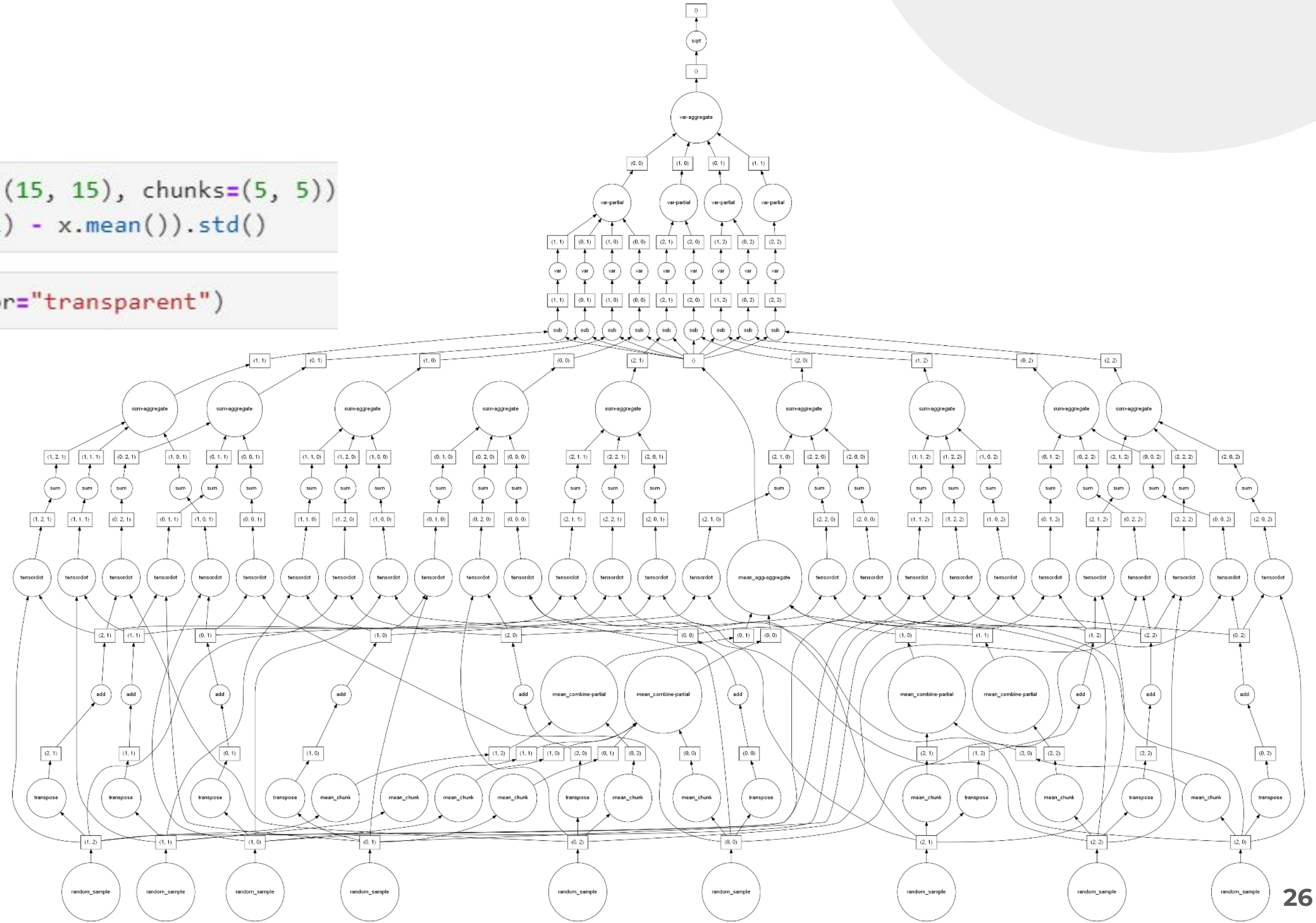


*Note: needs **graphviz** engine*

Dask Arrays

```python
x = da.random.random((15, 15), chunks=(5, 5))
comp = (x.dot(x.T + 1) - x.mean()).std()
```

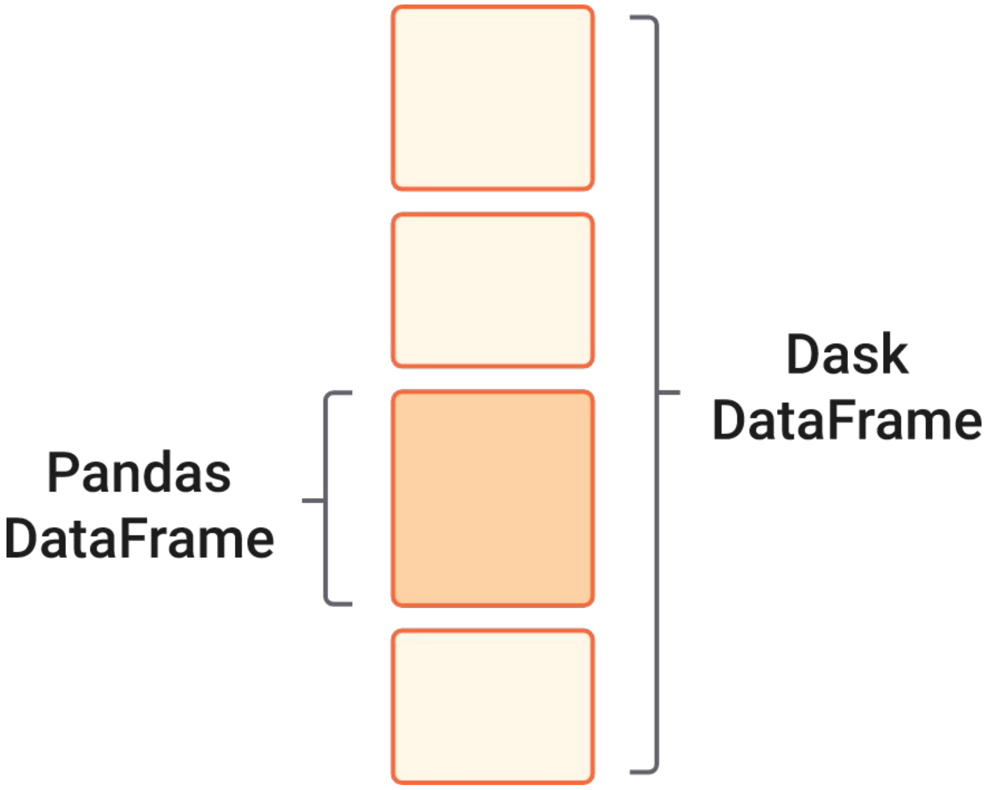
```python
comp.visualize(bgcolor="transparent")
```



<u>Note</u>: *needs **graphviz** engine*

Dask Arrays

Dask DataFrames



Parallel Pandas:

- Large dataframe chunked into smaller ones;
- As for dask arrays, allows load of DF larger than RAM;
- Speed up the computation;
- Allow big data visualization (hvplot, datashader...).

Taxi data on Kaggle

```
!ls -l --block-size=G "Data/"
```

```
total 7G
-rw-r--r-- 1 jlezmy Domain Users 2G Feb  2 14:56 yellow_tripdata_2015-01.csv
-rw-r--r-- 1 jlezmy Domain Users 2G Feb  2 14:57 yellow_tripdata_2016-01.csv
-rw-r--r-- 1 jlezmy Domain Users 2G Feb  2 14:57 yellow_tripdata_2016-02.csv
-rw-r--r-- 1 jlezmy Domain Users 2G Feb  2 14:57 yellow_tripdata_2016-03.csv
```

```
import dask.dataframe as dd
```

```
%%time
ddf = dd.read_csv( os.path.join("Data/*.csv"), blocksize="64MB" )
```

```
Wall time: 43.3 ms
```

```
len(ddf)
```

```
47248845
```

```
list(ddf.columns)
```

```
['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
 'trip_distance',
 'pickup_longitude',
 'pickup_latitude',
 'RateCodeID',
 'store_and_fwd_flag',
 'dropoff_longitude',
 'dropoff_latitude',
 'payment_type',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount']
```

```python
from dask.distributed import Client, LocalCluster
```

```python
# Let's create a local cluster:

cluster = LocalCluster()
client = Client(cluster)

### exactly the same than client = Client(), but more explicit.
```

```python
client
```

Client

Client-d04b90bd-a6c8-11ed-bcd0-fcb3bce22673

Connection method: Cluster object	**Cluster type:** distributed.LocalCluster
Dashboard: http://127.0.0.1:8787/status	

▾ Cluster Info

LocalCluster

4cbadea0

Dashboard: http://127.0.0.1:8787/status	**Workers:** 4
Total threads: 8	**Total memory:** 7.71 GiB
Status: running	**Using processes:** True

▸ Scheduler Info

Let’s create a local cluster:

- Visibility on the task graph;
- “ “ task stream;
- “ “ workers status;
- … through an awesome dashboard!

Note: a dask extension is available on jupyter-lab

```python
subdf = ddf[['payment_type','total_amount',
             'trip_distance']]
groupdf = subdf.groupby('payment_type').agg({'total_amount':"mean",
                                              'trip_distance':"mean"})
```

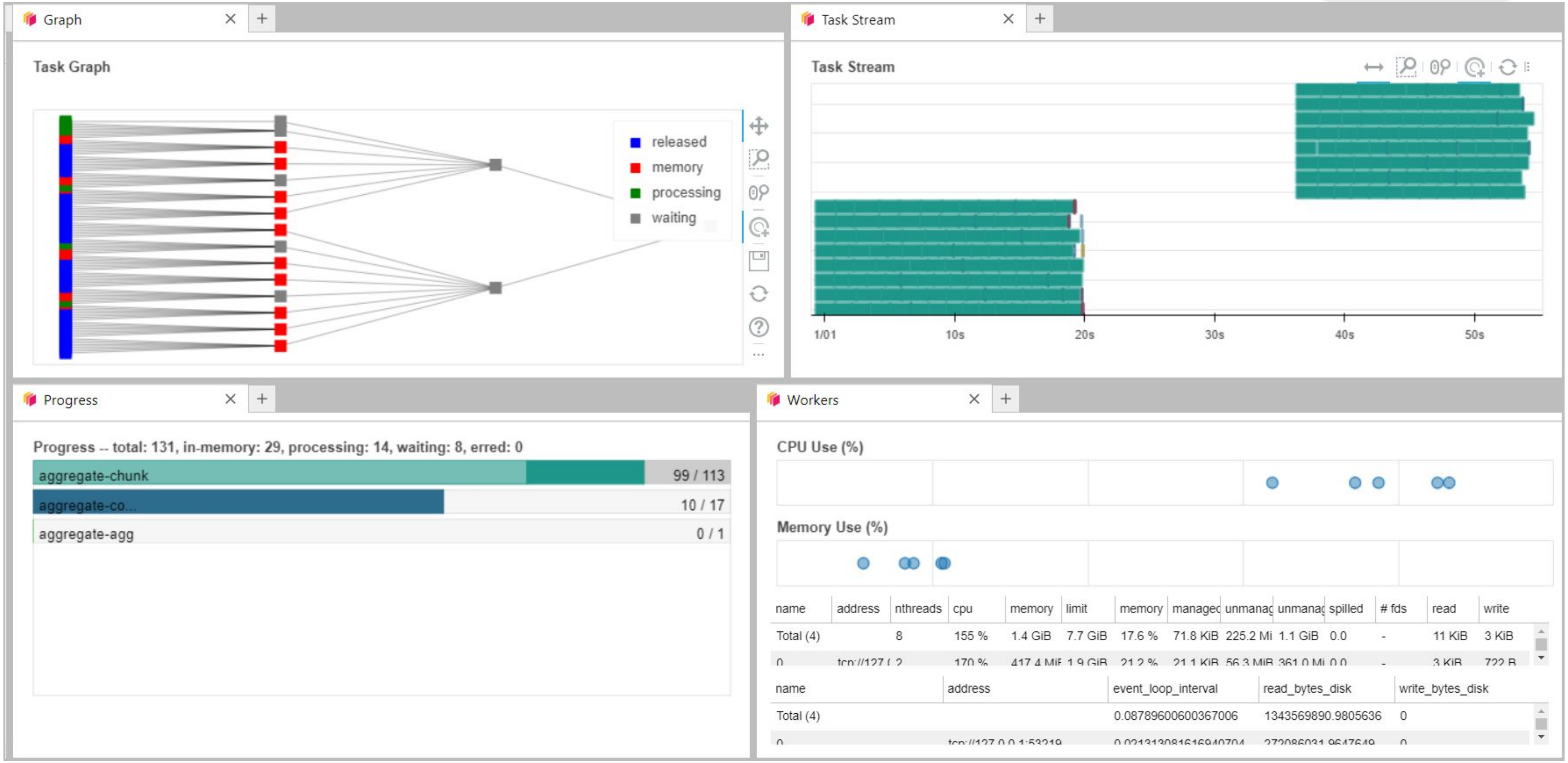
```python
%%time
groupdf.compute()
```

```
Wall time: 19.6 s
```

payment_type	total_amount	trip_distance
1	17.109936	4.171999
2	12.711215	13.210922
3	15.030975	65.266945
4	12.290095	32.067464
5	6.200000	1.166667

Dask Arrays

Dask DataFrames



Big Data visualization: Datashader

```
subdf = ddf[['pickup_longitude',
             'pickup_latitude',
             'passenger_count']]
```

```
import datashader
from datashader import transfer_functions as tf
from datashader.colors import Hot

def render(df, x_range=(-74.1, -73.7), y_range=(40.6, 40.9)):
    canvas = datashader.Canvas(
        x_range=x_range,
        y_range=y_range,
    )
    agg = canvas.points(
        source=df,
        x="pickup_longitude",
        y="pickup_latitude",
        agg=datashader.count("passenger_count"),
    )
    return tf.shade(agg, cmap=Hot, how="eq_hist")
```

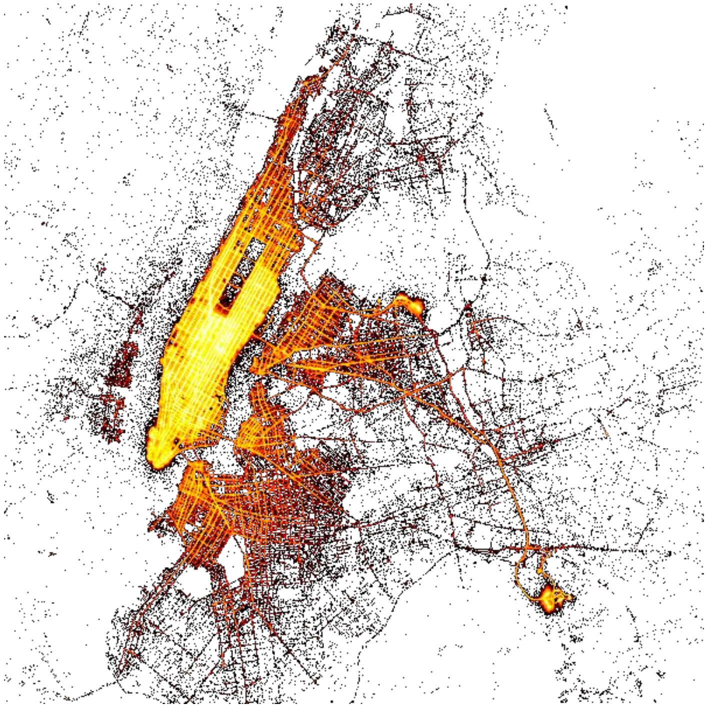
Big Data visualization: Datashader

```
subdf = ddf[['pickup_longitude',
             'pickup_latitude',
             'passenger_count']]
```

```
import datashader
from datashader import transfer_functions as tf
from datashader.colors import Hot

def render(df, x_range=(-74.1, -73.7), y_range=(40.6, 40.9)):
    canvas = datashader.Canvas(
        x_range=x_range,
        y_range=y_range,
    )
    agg = canvas.points(
        source=df,
        x="pickup_longitude",
        y="pickup_latitude",
        agg=datashader.count("passenger_count"),
    )
    return tf.shade(agg, cmap=Hot, how="eq_hist")
```

```
%time render(subdf) ## 48M points !!
```

```
Wall time: 49.9 s
```



Dask Arrays

Dask DataFrames

Dask Delayed

Low-level API for **parallelizable problems** which don't fit into high-level abstractions (Dask Array / DataFrame)

- ➢ Lazy objects: computed only if explicitly asked to;
- ➢ Equivalent to DAG nodes;
- ➢ Dask creates an optimised graph, determining the dependencies between each delayed object;
- ➢ Doesn't need a *Client*.

```python
from dask import delayed
def inc(x):
    return x + 1

def custom_op(x):
    return 2*x**2

def add(x, y):
    return x + y

data = np.arange(0,6,1)

a = [delayed(inc)(x) for x in data]
b = [delayed(custom_op)(x) for x in data]
output = [delayed(add)(i, j) for i,j in zip(a,b)]

total = delayed(sum)(output)
total
```

```
Delayed('sum-09ce14df-d4a1-4b0a-981e-e3a15f42bd9f')
```

```python
total.compute()
```

```
131
```

Dask Arrays Dask DataFrames Dask Delayed

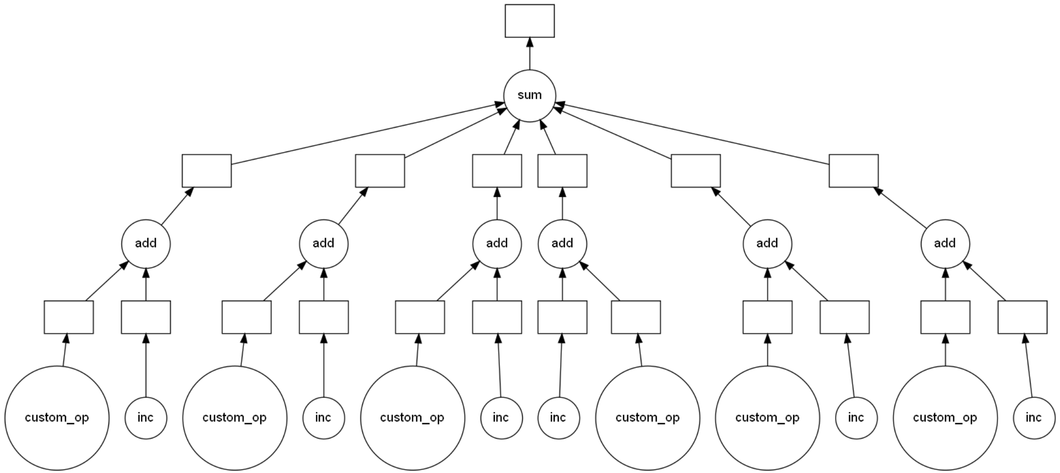
```python
from dask import delayed
def inc(x):
    return x + 1

def custom_op(x):
    return 2*x**2

def add(x, y):
    return x + y

data = np.arange(0,6,1)

a = [delayed(inc)(x) for x in data]
b = [delayed(custom_op)(x) for x in data]
output = [delayed(add)(i, j) for i,j in zip(a,b)]

total = delayed(sum)(output)
total
```

```
Delayed('sum-09ce14df-d4a1-4b0a-981e-e3a15f42bd9f')
```

```python
total.compute()
```

```
131
```

```python
total.visualize(bgcolor='transparent')
```



Dask Delayed as a *decorator*

```
@delayed
def inc(x):
    return x + 1
@delayed
def custom_op(x):
    return 2*x**2
@delayed
def add(x, y):
    return x + y

data = np.arange(0,6,1)

a = [inc(x) for x in data]
b = [custom_op(x) for x in data]
output = [add(i, j) for i,j in zip(a,b)]

total = delayed(sum)(output)
print(inc(0))
```

```
Delayed('inc-789d1c17-df7b-4d14-954f-aab785f5fab5')
```

Dask Delayed as a *decorator*

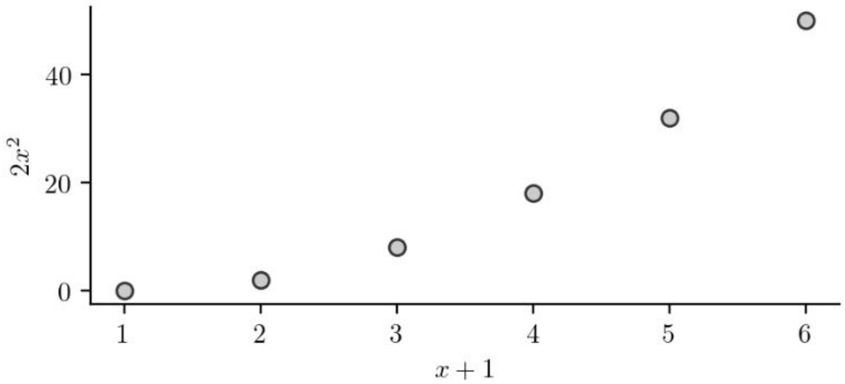
```python
@delayed
def inc(x):
    return x + 1
@delayed
def custom_op(x):
    return 2*x**2
@delayed
def add(x, y):
    return x + y

data = np.arange(0,6,1)

a = [inc(x) for x in data]
b = [custom_op(x) for x in data]
output = [add(i, j) for i,j in zip(a,b)]

total = delayed(sum)(output)
print(inc(0))
```

```
Delayed('inc-789d1c17-df7b-4d14-954f-aab785f5fab5')
```

```python
@delayed
def plot(a,b):
    fig,ax = plt.subplots(figsize=(5,2), dpi=200)
    ax.scatter(a,b, marker='o', facecolor='0.8', edgecolor='0.2')
    ax.set_xlabel('$x+1$')
    ax.set_ylabel(r'$2x^{2}$')
    ax.spines[['right','top']].set_visible(False)
    return fig
plot(a,b)
```

```
Delayed('plot-65238b37-98a6-45b2-8cf5-f6b6a70ec463')
```

```python
fig = plot(a,b).compute()
```



Dask Arrays

Dask DataFrames

Dask Delayed

Dask Futures

Low-level API : real-time task framework that extends Python's concurrent.futures interface

- ➢ ~~Lazy~~ Eager objects: computed **immediately**;
- ➢ Scale your Python futures workflow across a Dask cluster with minimal code changes;
- ➢ Add **flexibility**;
- ➢ *Dask client* is **needed** to use future interface.

Minimal example

```python
from dask.distributed import Client, LocalCluster
import numpy as np
import time

cluster = LocalCluster()
client = Client(cluster) # Same as client=Client()
```

```python
def load(x):
    time.sleep(0.2)
    return np.arange(10000) + x

def process1(x):
    return x**2

def process2(x,y):
    return 2*x + y

def save(x):
    time.sleep(0.2)
    return 'Saved'
```

```python
inputs_1, inputs_2 = np.arange(0,50), np.arange(50,100)
futures = []

for i,j in zip(inputs_1,inputs_2):
    x = client.submit(load, i) #client.submit(fucn, *args)
    y = client.submit(load, j)
    xp = client.submit(process1, x)
    xyp = client.submit(process2, xp,y)
    z = client.submit(save, xyp)
    futures.append(z)
z
```

Future: save status: pending, type: NoneType, key: save-c6c436fe0ee6c4795657b742886162df

```python
z
```

Future: save status: finished, type: str, key: save-c6c436fe0ee6c4795657b742886162df

```python
#result = [future.result() for future in futures]
results = client.gather(futures) ### faster
```

Combining Futures and Delayed ?

```python
from dask import delayed
@delayed
def inc(x):
    return x + 1
@delayed
def custom_op(x):
    return 2*x**2
@delayed
def add(x, y):
    return x + y
```

```python
@delayed
def plot(a,b,savefile=None):
    fig,ax = plt.subplots(figsize=(5,2) )
    ax.scatter(a,b, marker='o', facecolor='0.8', edgecolor='0.2')
    ax.set_xlabel(r'$x+1$')
    ax.set_ylabel(r'$2x^{2}$')
    ax.spines[['right','top']].set_visible(False)
    fig.tight_layout()
    if savefile != None:
        fig.savefig(savefile,transparent=True)
        return
    return fig
```

```python
def compute_single(data, show=False, savefile=None):

    stored =[]
    a = [inc(x) for x in data]
    b = [custom_op(x) for x in data]
    output = [add(i, j) for i,j in zip(a,b)]
    stored.append(delayed(sum)(output))
    if not show:
        return stored
    stored.append( plot(a,b,savefile) )
    return stored
```

```python
stored = compute_single(data=np.arange(0,6,1),
                        show=True, savefile=None)
stored
```

```
[Delayed('sum-b6638107-bfe6-426a-90a9-4f33b2295286'),
 Delayed('plot-c30f1ea7-224c-420f-b8de-e24dbc69efb6')]
```

```python
future = client.compute(stored)
results = client.gather(future);
results
```

```
[131, <Figure size 500x200 with 1 Axes>]
```

Use cases

- Pre-processing (dask_ml.preprocessing):
  - *MinMaxscaler, Labelencoder, OneHotEncoder, PolynomialFeatures ...*
- Cross Validation (for instance extension to dask arrays):
  - *dask_ml.model_selection.train_test_split()*
- Hyper Parameter Search optimization.

Random forest algorithms with tunable parameters:

Bootstrap = True *max_depth = [8, 10]* *n_estimator = [20, 100, 150]*

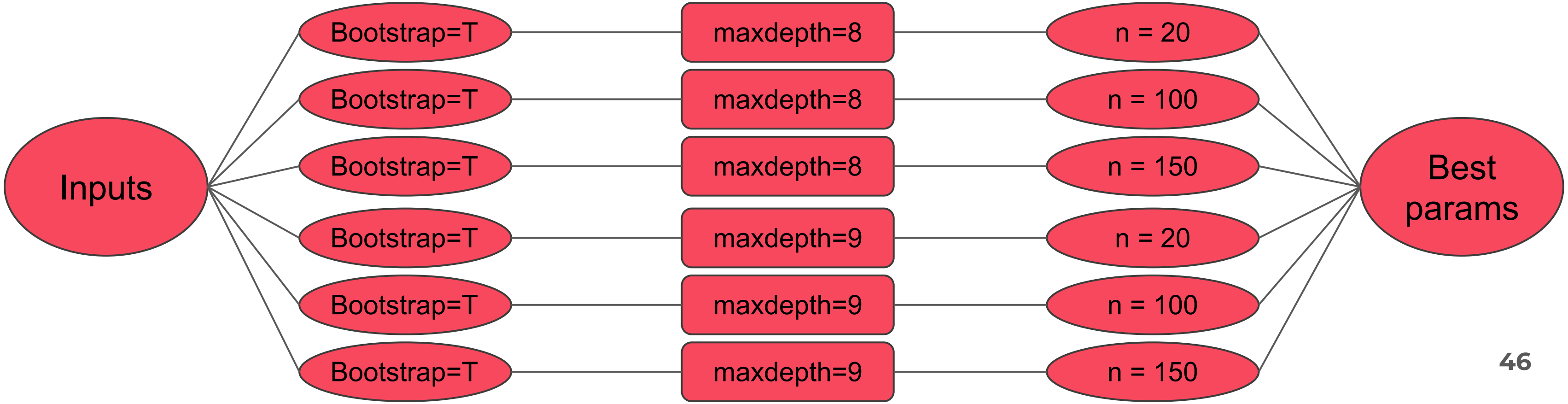
Use cases

- Pre-processing (dask_ml.preprocessing):
  - *MinMaxscaler, Labelencoder, OneHotEncoder, PolynomialFeatures ...*

- Cross Validation (for instance extension to dask arrays):
  - *dask_ml.model_selection.train_test_split()*

- Hyper Parameter Search optimization.

<u>*SKlearn*</u>



Use cases

- ➢ Pre-processing (dask_ml.preprocessing):
  - *MinMaxscaler, Labelencoder, OneHotEncoder, PolynomialFeatures* ...
- ➢ Cross Validation (for instance extension to dask arrays):
  - *dask_ml.model_selection.train_test_split()*
- ➢ Hyper Parameter Search optimization.

Dask