



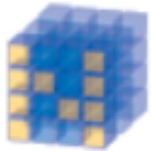
Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften



Computing and Plotting Libraries



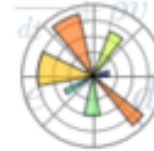
scipy



NumPy
Base N-dimensional
array package



SciPy library
Fundamental library
for scientific
computing



Matplotlib
Comprehensive 2D
Plotting



IPython
Enhanced Interactive
Console



Sympy
Symbolic mathematics



pandas
Data structures &
analysis



numpy



-
- a powerful N-dimensional array object
 - sophisticated (broadcasting) functions
 - tools for integrating C/C++ and Fortran code
 - useful linear algebra, Fourier transform, and random number capabilities



```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
>>> y = np.arange(10) # like Python's range, but returns an array
>>> y
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a = np.array([1, 2, 3, 6])
>>> b = np.linspace(0, 2, 4)
# create an array with four equally spaced points starting with 0 and ending
with 2.
>>> c = a - b
>>> c
array([ 1.          ,  1.33333333,  1.66666667,  4.          ])
>>> a**2
array([ 1,  4,  9, 36])
```




pandas



- DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.



pandas

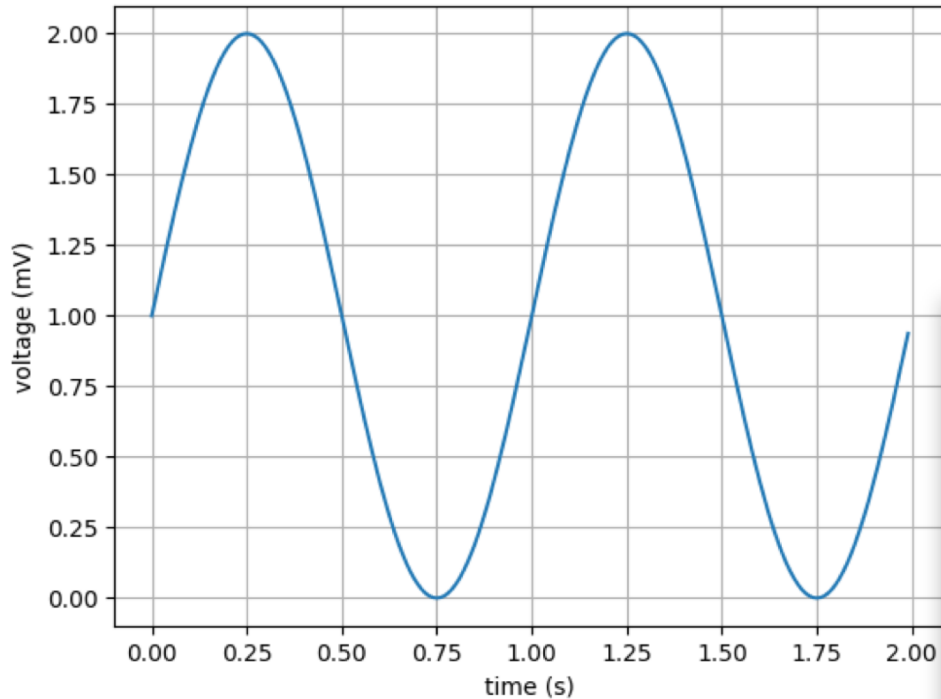
```
>>> import pandas as pd
>>> df = pd.read_csv("ign.csv")
>>> df.head()
>>> df.tail()
>>> df.shape
>>> df.loc[0:5,:]
>>> df.iloc[0:5,:]
>>> df.index
>>> df.loc[:5,["score","release_year"]]
>>> df["score"].mean()
>>> df.corr()
>>> r1=df["scores"] > 7
>>> df[r1]
>>> df["score"].plot(kind="hist")
```



matplotlib

matplotlib

About as simple as it gets, folks



```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Data for plotting
```

```
t = np.arange(0.0, 2.0, 0.01)
```

```
s = 1 + np.sin(2 * np.pi * t)
```

```
# Note that using plt.subplots below is equivalent to using
```

```
# fig = plt.figure() and then ax = fig.add_subplot(111)
```

```
fig, ax = plt.subplots()
```

```
ax.plot(t, s)
```

```
ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
```

```
ax.grid()
```

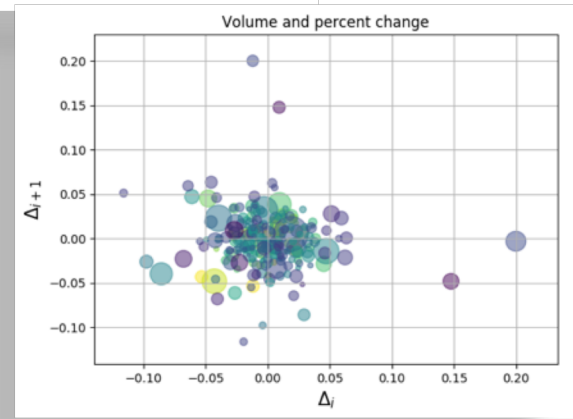
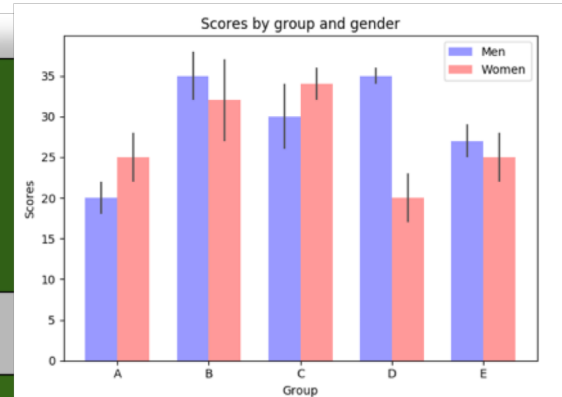
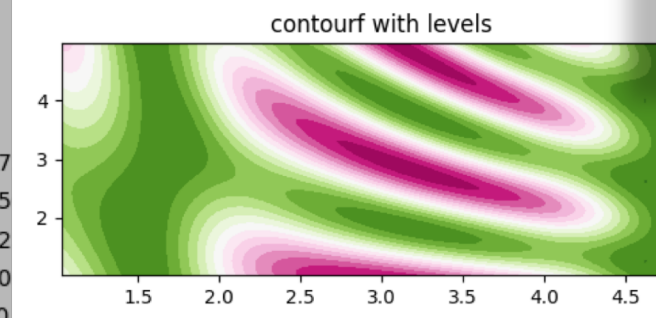
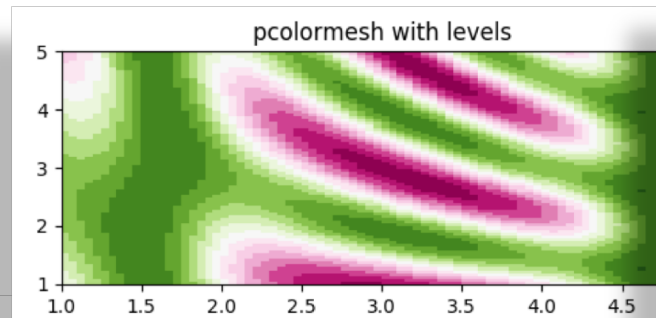
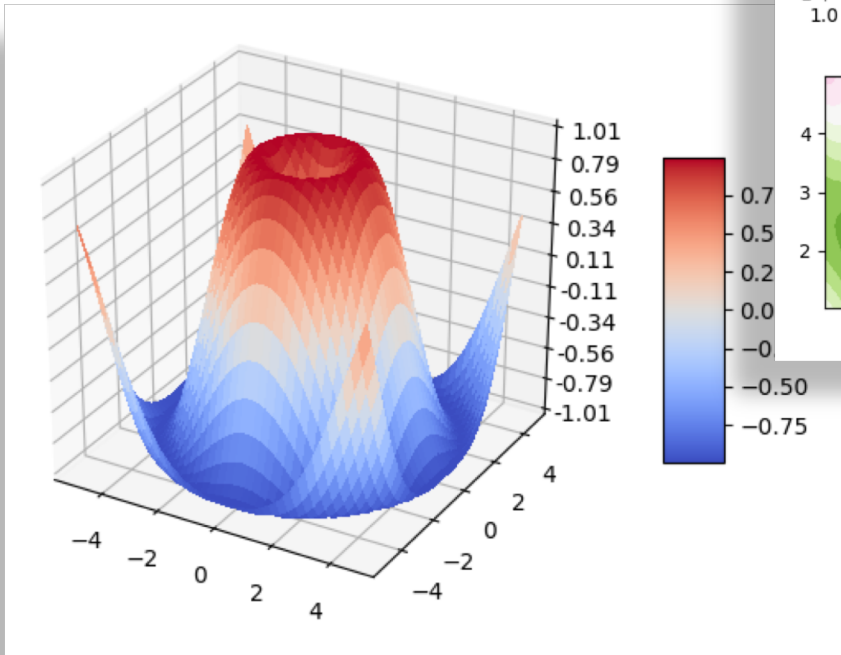
```
fig.savefig("test.png")
```

```
plt.show()
```



matplotlib

matplotlib



IP[y]: Notebook spectrogram Last Checkpoint: a few seconds ago (autosaved) Python (Python 3)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

Simple spectral analysis

An illustration of the [Discrete Fourier Transform](#) using windowing, to reveal the frequency content of a sound signal.

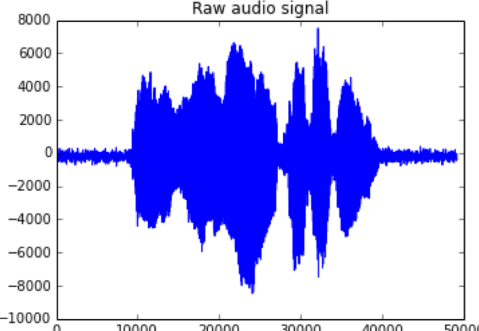
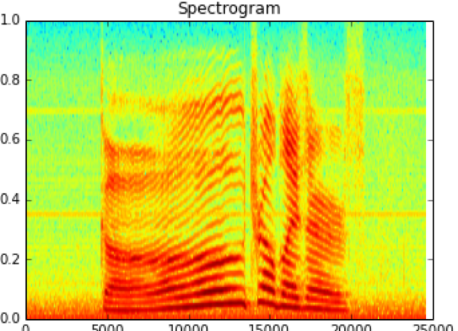
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin specgram routine:

```
In [2]: %matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```

A plot titled "Raw audio signal" showing a blue waveform. The y-axis ranges from -10000 to 8000, and the x-axis ranges from 0 to 50000. The signal is centered around zero and shows a complex, oscillatory pattern.A spectrogram plot titled "Spectrogram" showing a heatmap of frequency content. The y-axis ranges from 0.0 to 1.0, and the x-axis ranges from 0 to 25000. The plot shows a series of vertical lines and horizontal bands, indicating the frequency components of the audio signal over time.



Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften



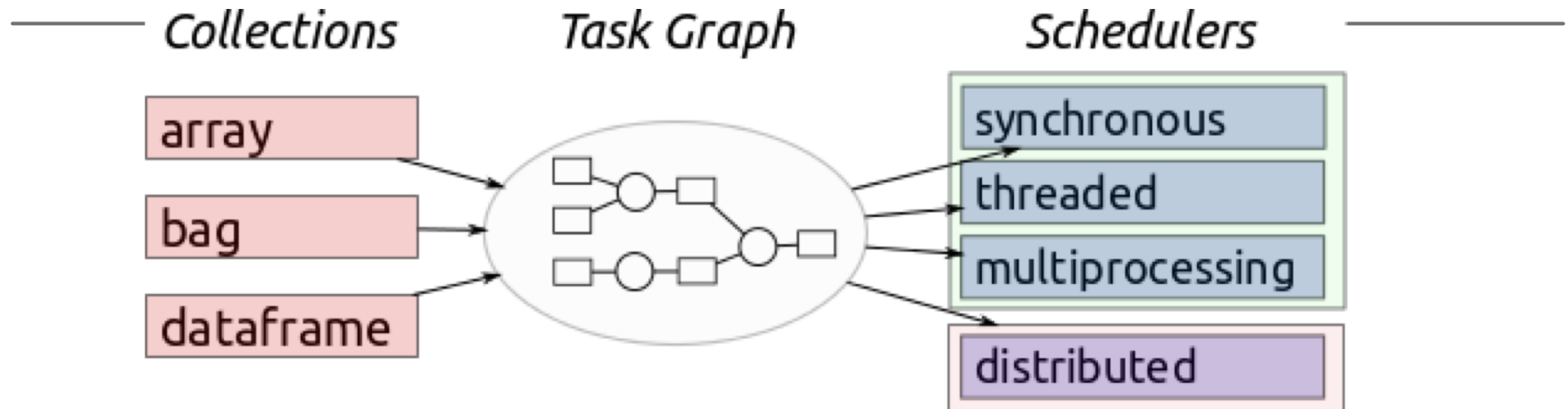
Data Analysis



dask



DASK



Familiar: Provides parallelized NumPy array and Pandas DataFrame objects

Flexible: Provides a task scheduling interface for more custom workloads and integration with other projects.

Native: Enables distributed computing in Pure Python with access to the PyData stack.

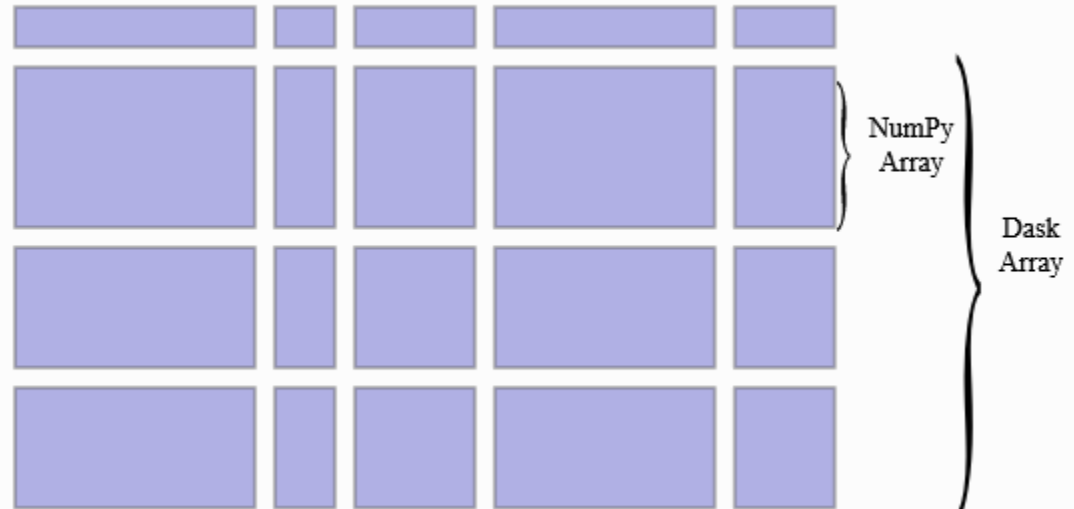
Fast: Operates with low overhead, low latency, and minimal serialization necessary for fast numerical algorithms

Scales up: Runs resiliently on clusters with 1000s of cores

Scales down: Trivial to set up and run on a laptop in a single process, even on a smartphone running android

Responsive: Designed with interactive computing in mind it provides rapid feedback and diagnostics to aid humans

- dask arrays are composed of numpy arrays.
- the subarrays can live in the same process or in another process on a different node
- dask has a scheduler which distributes the work on a whole cluster if needed

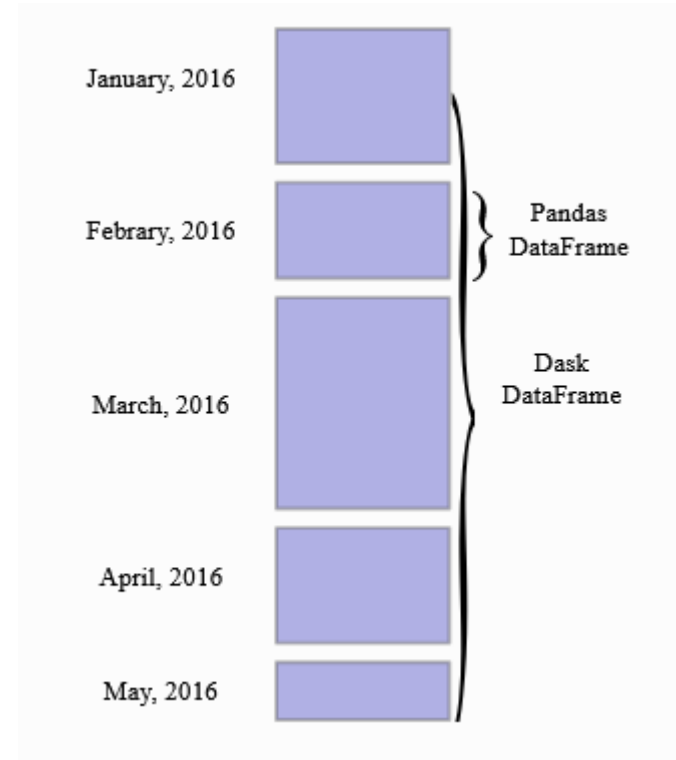


```
>>> import dask.array as da
>>> a=da.random.uniform(size=1000, chunks=100)
```



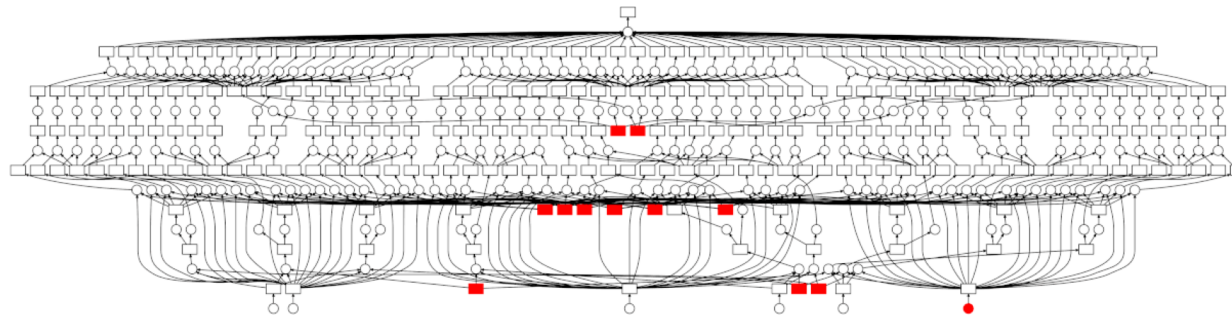
- like `dask.arrays` uses numpy arrays, `dask.dataframe` uses pandas
- `dask.dataframes` can be distributed over a cluster of nodes and operations on them are scheduled by the dask scheduler

```
>>> import dask.dataframe as dd
>>> df=dd.read_csv('2014-*.csv')
```



```
>>> a=da.random.uniform(size=1000, chunks=100)
>>> b=a.sum()
>>> c=a.mean()*a.size
>>> d=b-c
>>> d.compute()
```

the computation starts at the last command. If you have a dask cluster then all computations can be distributed to the cluster.





Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften



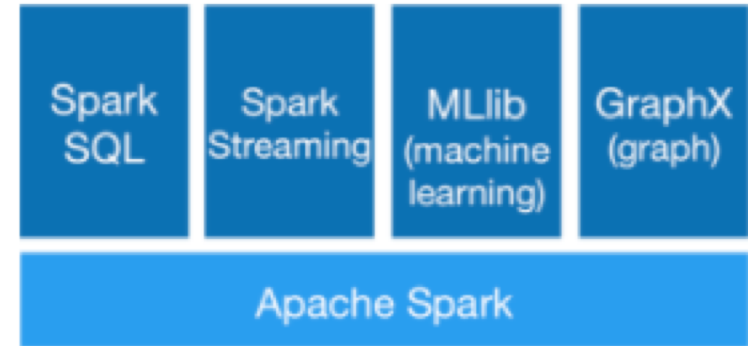
Data Streams



spark



Apache Spark™ is a fast and general engine for large-scale data processing.



```
text_file = spark.textFile("hdfs://...")
```

```
text_file.flatMap(lambda line: line.split())  
    .map(lambda word: (word, 1))  
    .reduceByKey(lambda a, b: a+b)
```



kubernetes

Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing

- written in java
- built on top of Hadoop cluster technology
- language bindings for python, R and scala
- plugs seamlessly into the python ecosystem (scipy, matplotlib, jupyter)



spark



The screenshot shows a Jupyter Notebook interface in a browser window. The browser address bar shows the URL `localhost:8888/notebooks/dev/sicara/titanic/jupyter/main.py.ipynb`. The notebook title is `main.py` and it indicates the last checkpoint was on 11/17/2016. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar contains various icons for file operations and execution. The notebook content shows two input cells:

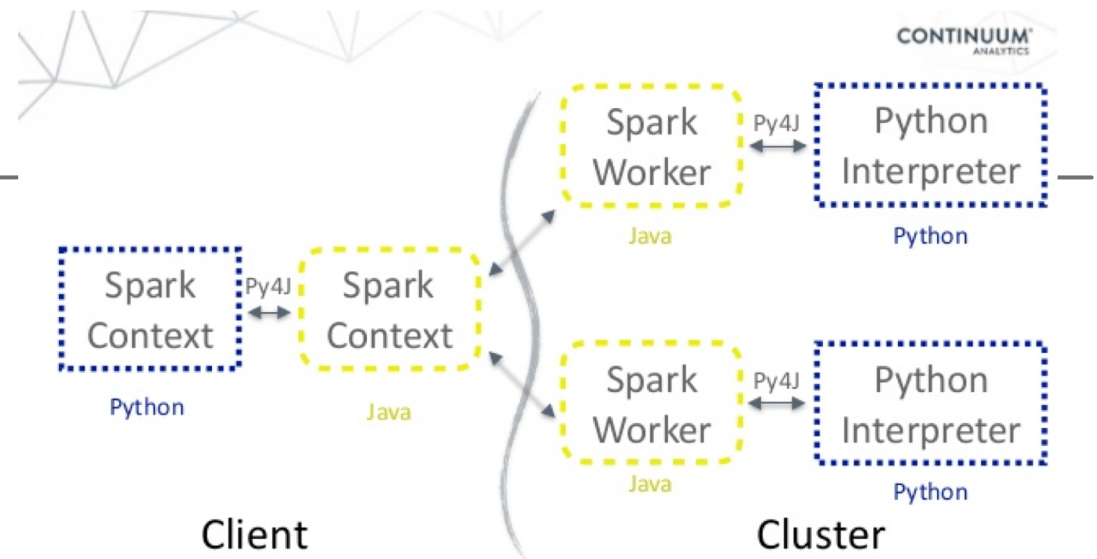
```
In [1]: import pandas as pd
import numpy as np

# create data frame containing your data, each column can be accessed # by df['column name']
data = pd.read_csv('./data/train.csv')
```

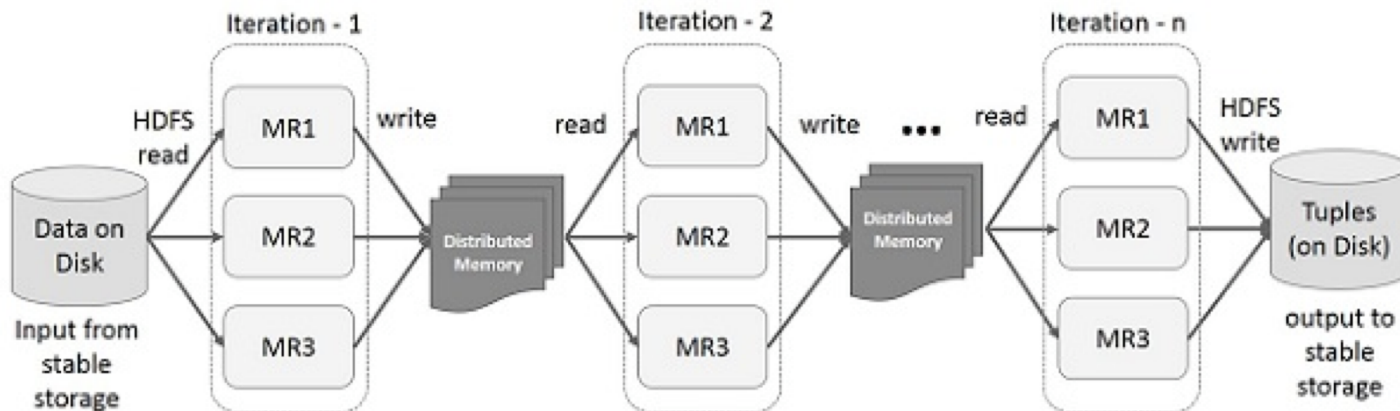
In [2]: data

2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S

Client/server Architecture



RDD (Resilient Distributed Dataset)
read-only, partitioned collection of records



- map
- filter
- flatMap
- mapPartitions
- union / intersection
- distinct
- groupByKey, reduceByKey, aggregateByKey
- sortByKey
- join



Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften



Machine Learning Packages



theano

theano

Theano:

- numerical computation library for Python
- computations are expressed using a Numpy-esque syntax
- compiled to run efficiently
- CPU or GPU architectures
- Dead since 2017, but still in use



tensorflow

- TensorFlow
- open-source software library
- dataflow programming across a range of tasks
- symbolic math library
- used for machine learning applications
- neural networks
- research and production at Google
- very active
- steep learning curve





tensorflow

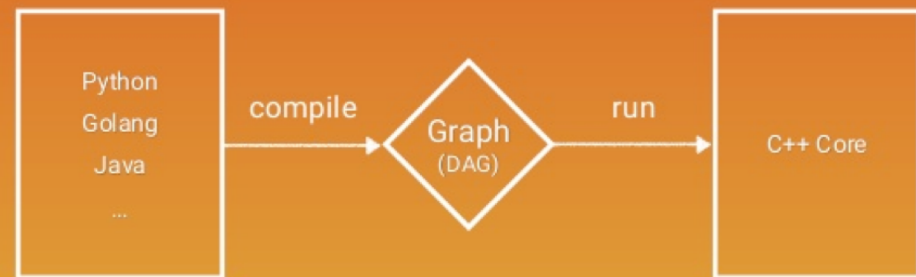
```
# load TensorFlow
>>> import tensorflow as tf
# Initialize two vectors
>>> x = tf.constant([1,2,3,4])
>>> y = tf.constant([5,6,7,8])
# Multiply
z= tf.multiply(x, y)
# Initialize Session and run
>>> with tf.Session() as sess:
. . . out = sess.run(z)
. . . print(out)
6
```





```
# load TensorFlow
>>> import tensorflow as tf
# Initialize two vectors
>>> x = tf.constant([1,2,3,4])
>>> y = tf.constant([1,2,3,4])
# Multiply
z = tf.multiply(x, y)
# Initialize Session
>>> with tf.Session() as sess:
    . . . out = sess.run(z)
    . . . print(out)
6
```

How does TensorFlow work





Keras



Keras

- Keras is a high-level neural networks API
- Running on top of TensorFlow, CNTK, or Theano
- Developed with a focus on enabling fast experimentation
- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility)
- Supports both convolutional networks and recurrent networks, as well as combinations of the two
- Runs seamlessly on CPU and GPU



Keras in 30 seconds

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=32)

classes = model.predict(x_test, batch_size=128)
```



Keras in 30 seconds

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> from keras.layers import Dense
>>> model_add = fn(model.add)

>>> Dense(units=64, activation='relu', input_dim=100) >> model_add
>>> Dense(units=10, activation='softmax') >> model_add
>>> model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['accuracy'])
>>> model.fit(x_train, y_train, epochs=5, batch_size=32)
>>> classes = model.predict(x_test, batch_size=128)
```



```
# resnet50 pretrained application in keras

from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')
img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
# Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265',
u'tusker', 0.1122357), (u'n02504458', u'African_elephant', 0.061040461)]
```