4Developers

Interdyscyplinarne wydarzenie dla programistów

# Pyt(h)on vs słoń: aktualny stan przetwarzania dużych danych w Python

Jakub Nowacki

Yosh.AI, SigDelta, Sages

# whoami

CTO @ Yosh.AI (yosh.ai)

Lead Data Scientist @ SigDelta (sigdelta.com)

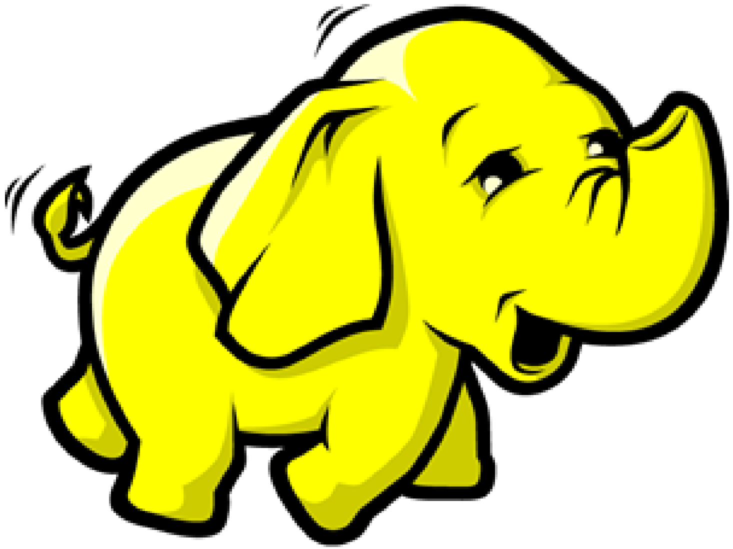
Trainer @ Sages (sages.com.pl)

I can code, I do maths

@jsnowacki

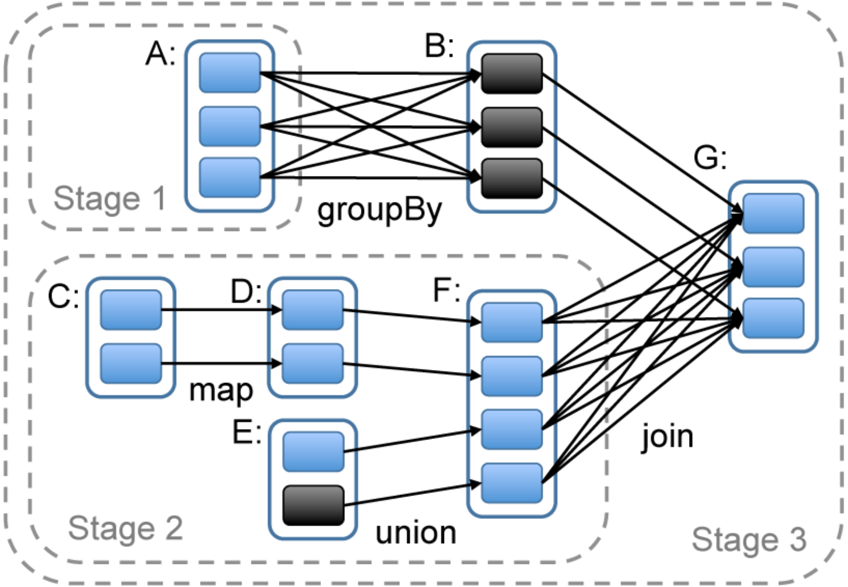
SAY BIG DATA
ONE MORE TIME
memegenerator.net

# Jak to było kiedyś?



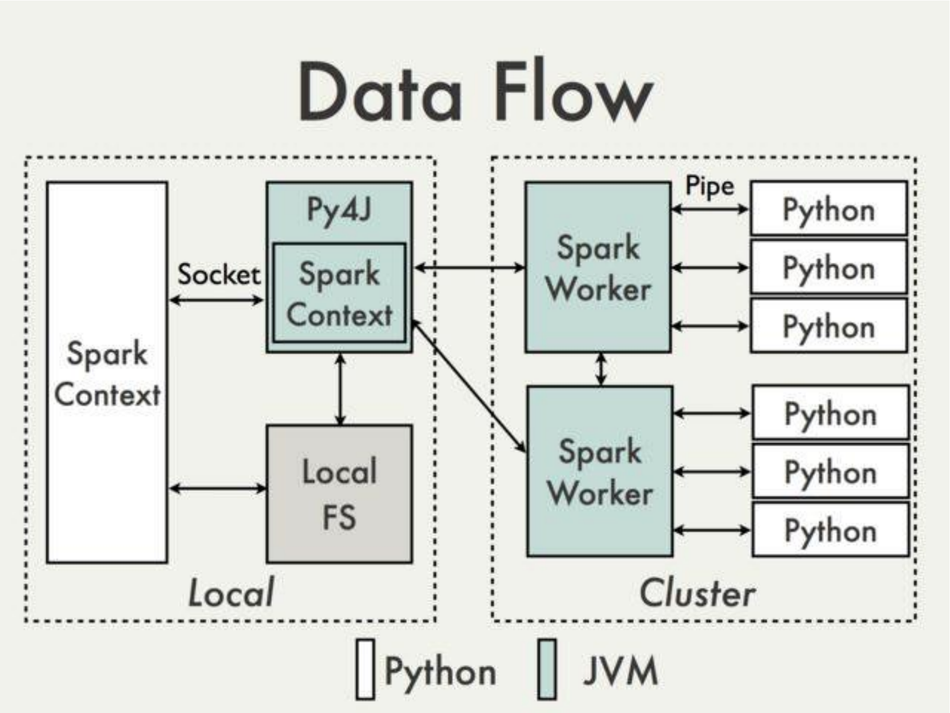
# Apache Spark!

# Spark RDD



```
sc.textFile("hdfs://...") \
        .flatMap(lambda line: line.split()) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda a, b: a + b) \
        .saveAsTextFile("hdfs://...")
```

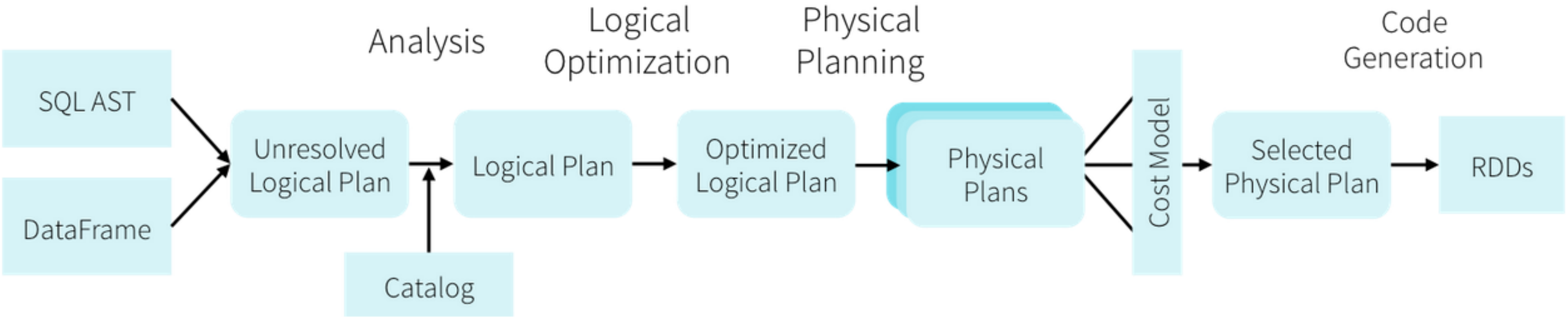
# Spark RDD – co gdzie?



Źródło: https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals

# Spark SQL - DataFrame

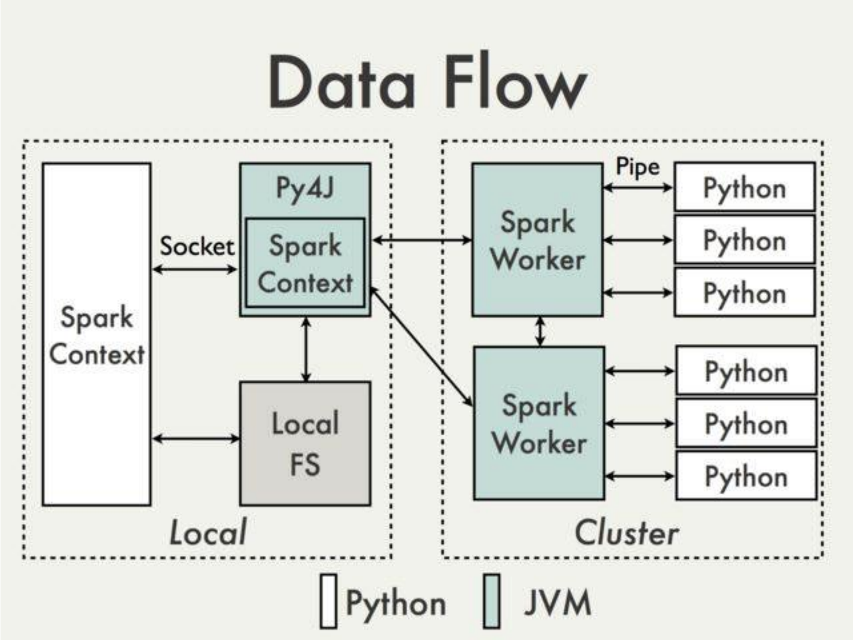
```
from pyspark.sql.functions import *

spark.read.text('hdfs://...’) \
        .select(explode(split('value', '\W+')).alias('word')) \
        .groupBy('word') \
        .count() \
        .orderBy(desc('count')) \
        .write.parquet('hdfs://...')
```



Źródło: https://databricks.com/blog/2015/03/24/spark-sql-graduates-from-alpha-in-spark-1-3.html

# UDF?!

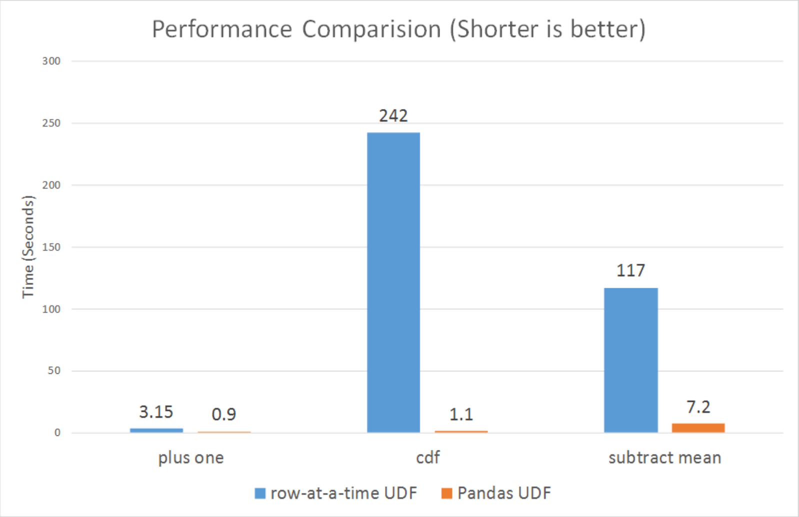
```
from pyspark.sql.types import IntegerType

@udf(returnType=IntegerType())
def add_one(x):
        if x is not None:
                return x + 1
```



Rozwiązanie - Java/Scala UDF to Python: http://sigdelta.com/blog/scala-spark-udfs-in-python/

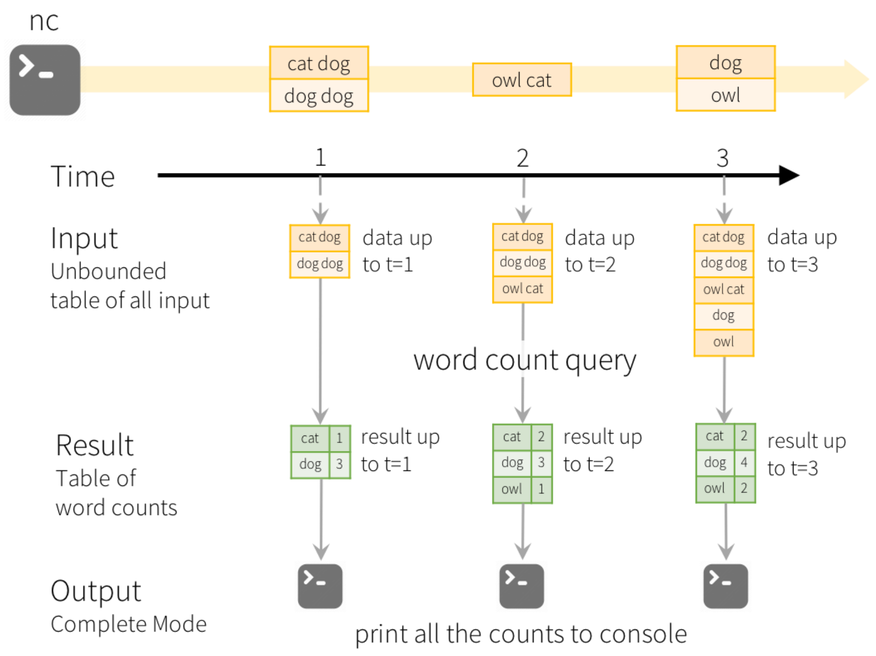
# Vectorized UDFs

```
import pandas as pd
from pyspark.sql.types import LongType
def multiply_func(a, b):
      return a * b
multiply = pandas_udf(multiply_func,
                      returnType=LongType())
pdf = pd.DataFrame([1, 2, 3], columns=["x"])
print(multiply_func(pdf.x, pdf.x))
# 0 1
# 1 4
# 2 9
# dtype: int64
df = spark.createDataFrame(pdf)
df.select(multiply(col("x"), col("x"))).show()
# +-------------------+
# |multiply_func(x, x)|
# +-------------------+
# | 1| # | 4| # | 9|
# +-------------------+
```



Źródło:
[https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html](https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html)

# Spark Structured Streaming



Model of the Quick Example

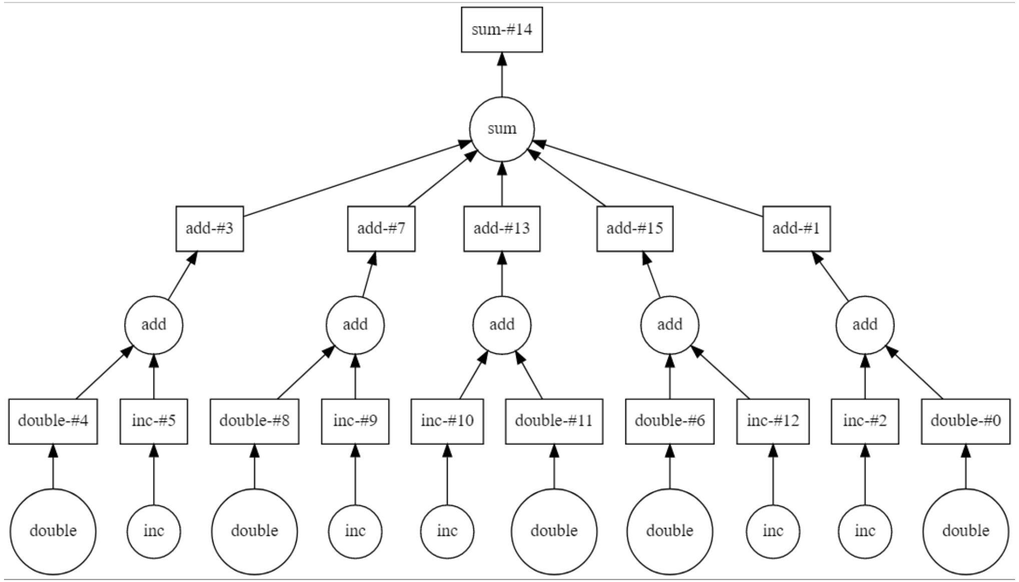
Źródło: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

# PySpark w PyPI

```
pip install pyspark
conda install pyspark
...
```

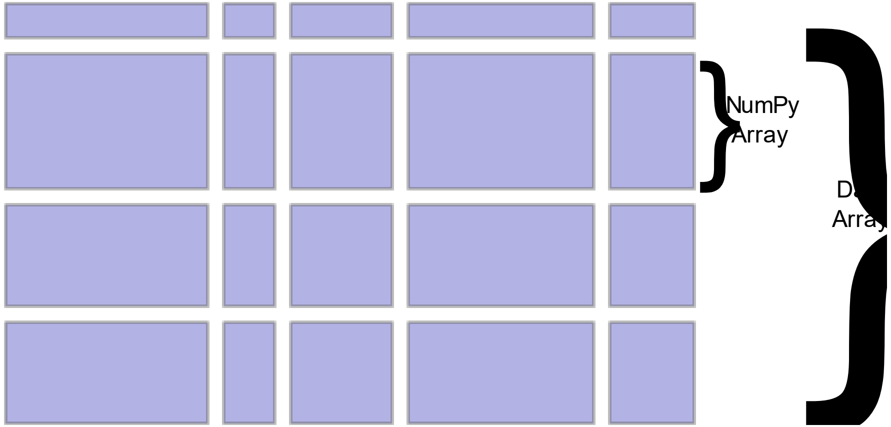
Opis: [http://sigdelta.com/blog/how-to-install-pyspark-locally/](http://sigdelta.com/blog/how-to-install-pyspark-locally/)

# Dask!



Źródło: https://dask.pydata.org/

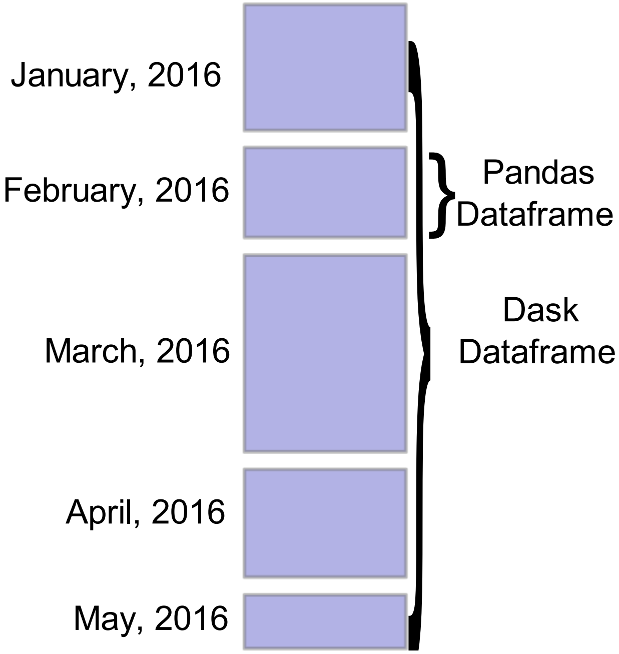
# Dask Array



```
import dask.array as da
import numpy as np

x = da.ones(10, chunks=(5,))
y = np.ones(10)
z = x + y
print(z)
# dask.array<add, shape=(10,),
#  … dtype=float64, chunksize=(5,)>
```

Źródło: https://dask.pydata.org/

# Dask DataFrame



```
import dask.dataframe as dd

posts = dd.read_parquet('data/posts_tags.parq’)\
        .set_index('id’)
posts_count = posts.creation_date.dt.date\
        .value_counts()

posts_count_df = posts_count.compute()
posts_count_df.head()

# 2017-08-23 9531
# 2017-07-27 9450
# 2017-08-24 9366
# 2017-08-03 9345
# 2017-03-22 9342
# Name: creation_date, dtype: int64
```

Źródło: https://dask.pydata.org/

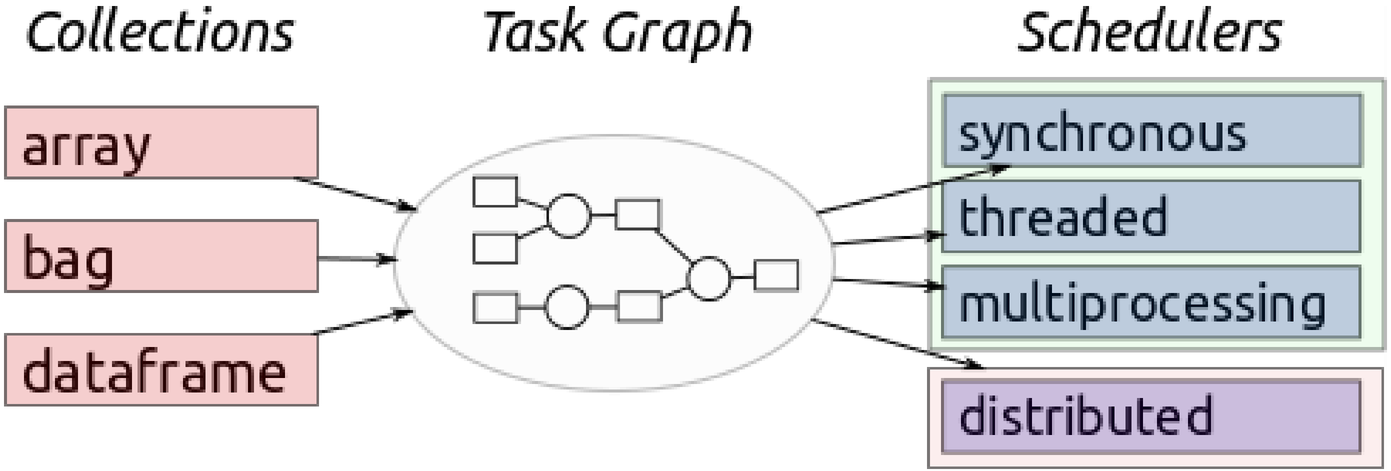
Przykład: http://sigdelta.com/blog/stackpverflow-tags-with-dask/

# Dask Bag

```
import dask.bag as db

tags_xml = db.read_text('data/Tags.xml', encoding='utf-8’)
tags_xml.take(5)
# ('\ufeff<?xml version="1.0" encoding="utf-8"?>\n’,
#  '<tags>\n’,
#  ' <row Id="1" TagName=".net" Count="257092" … />\n’,
#  ' <row Id="2" TagName="html" Count="683981" … />\n’,
#  ' <row Id="3" TagName="javascript" Count="1457944" … />\n’)
tags_rows = tags_xml.filter(lambda line: line.find('<row') >= 0)
tags_rows.take(5)
# (' <row Id="1" TagName=".net" Count="257092" … />\n’,
#  ' <row Id="2" TagName="html" Count="683981" … />\n’,
#  ' <row Id="3" TagName="javascript" Count="1457944" … />\n’,
#  ' <row Id="4" TagName="css" Count="490198" … />\n’,
#  ' <row Id="5" TagName="php" Count="1114030" … />\n’)
tags = tags_rows.map(extract_tags_columns).to_dataframe()
```

Przykład: http://sigdelta.com/blog/dask-introduction/

# Na jednej maszynie lub wielu



Źródło: https://dask.pydata.org/

# Dask?!

```
...

t.reset_index().head()

# ------------------------------------------------------------------
# ValueError Traceback (most recent call last)
# <ipython-input-100-e6186d78fb03> in <module>()
# ----> 1 t.reset_index().head()
#
# ...
#
# ValueError: Length mismatch: Expected axis has 3 elements, new
#  values have 2 elements
```

Źródło: [https://github.com/dask/dask/issues/3038](https://github.com/dask/dask/issues/3038) (naprawione)

# Ray

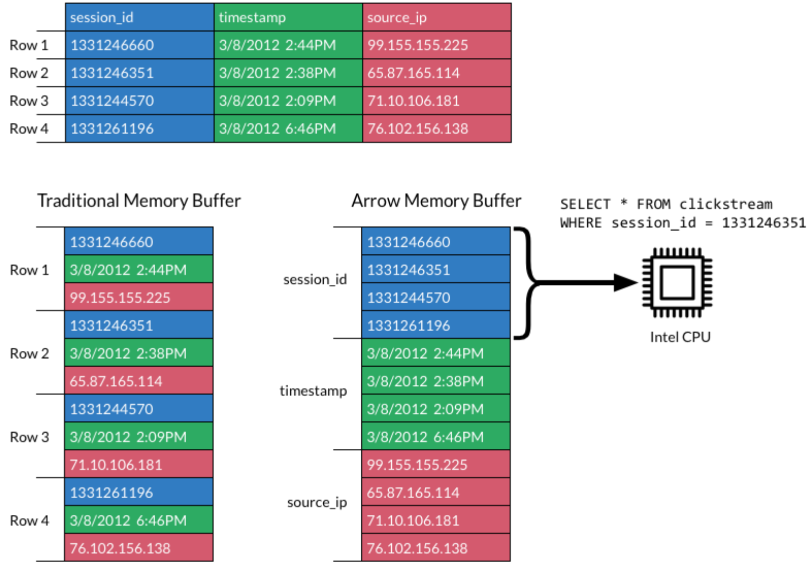
```
# import pandas as pd
import ray.dataframe as pd
stocks_df = pd.read_csv("all_stocks_5yr.csv")
print(type(stocks_df))
# <class 'ray.dataframe.dataframe.DataFrame'>
positive_stocks_df = stocks_df.query("close > open")
print(positive_stocks_df['date'].head(n=5))
# 0 2013-02-13
# 1 2013-02-15
# 2 2013-02-26
# 3 2013-02-27
# 4 2013-03-01
```

```
@ray.remote def f():
      time.sleep(1)
      return 1

ray.init()
results = ray.get([
      f.remote()
      for i in range(4)
])
```

Źródło: [https://rise.cs.berkeley.edu/blog/pandas-on-ray/](https://rise.cs.berkeley.edu/blog/pandas-on-ray/)

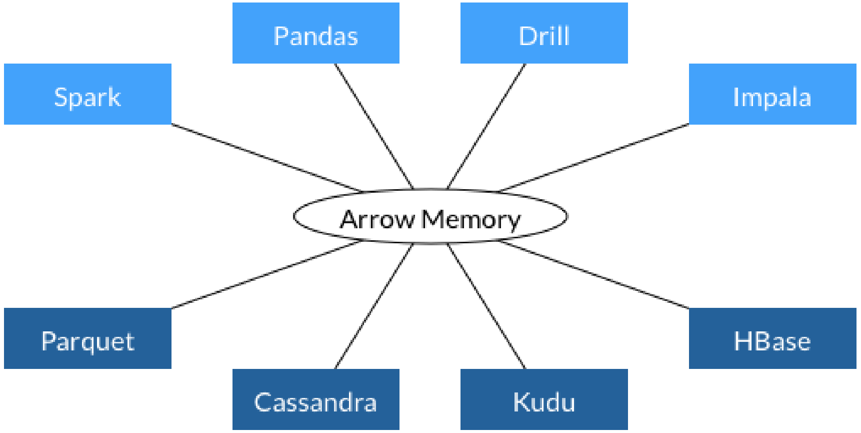
Źródło: [http://ray.readthedocs.io/en/latest/index.html](http://ray.readthedocs.io/en/latest/index.html)

# Apache Arrow



| | session_id | timestamp | source_ip |
|---|---|---|---|
| Row 1 | 1331246660 | 3/8/2012 2:44PM | 99.155.155.225 |
| Row 2 | 1331246351 | 3/8/2012 2:38PM | 65.87.165.114 |
| Row 3 | 1331244570 | 3/8/2012 2:09PM | 71.10.106.181 |
| Row 4 | 1331261196 | 3/8/2012 6:46PM | 76.102.156.138 |

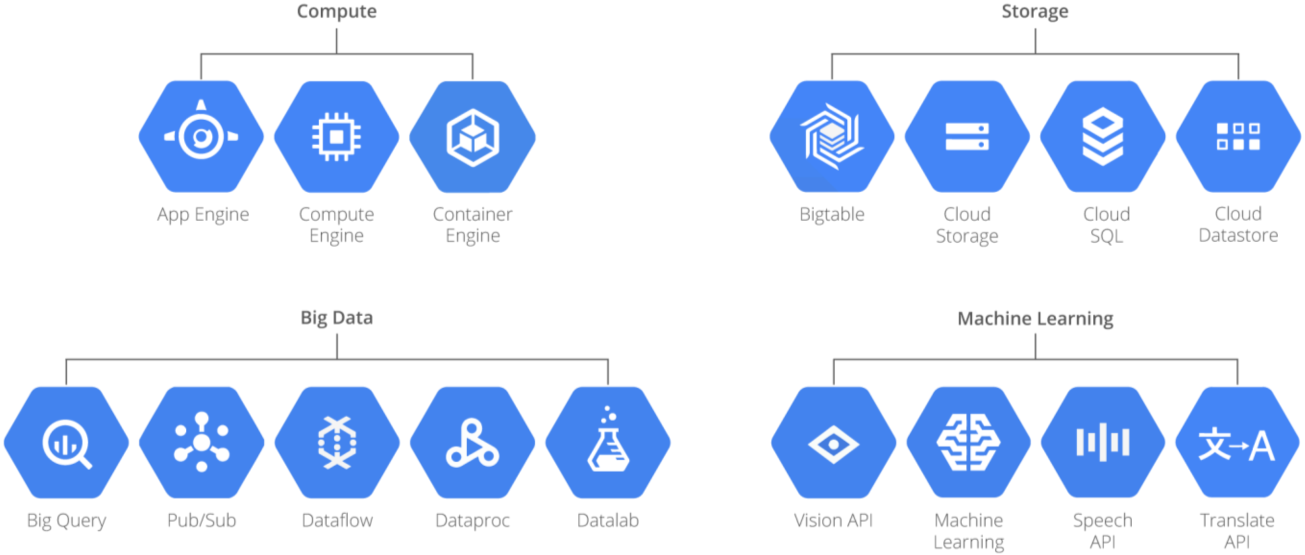
```
SELECT * FROM clickstream
WHERE session_id = 1331246351
```



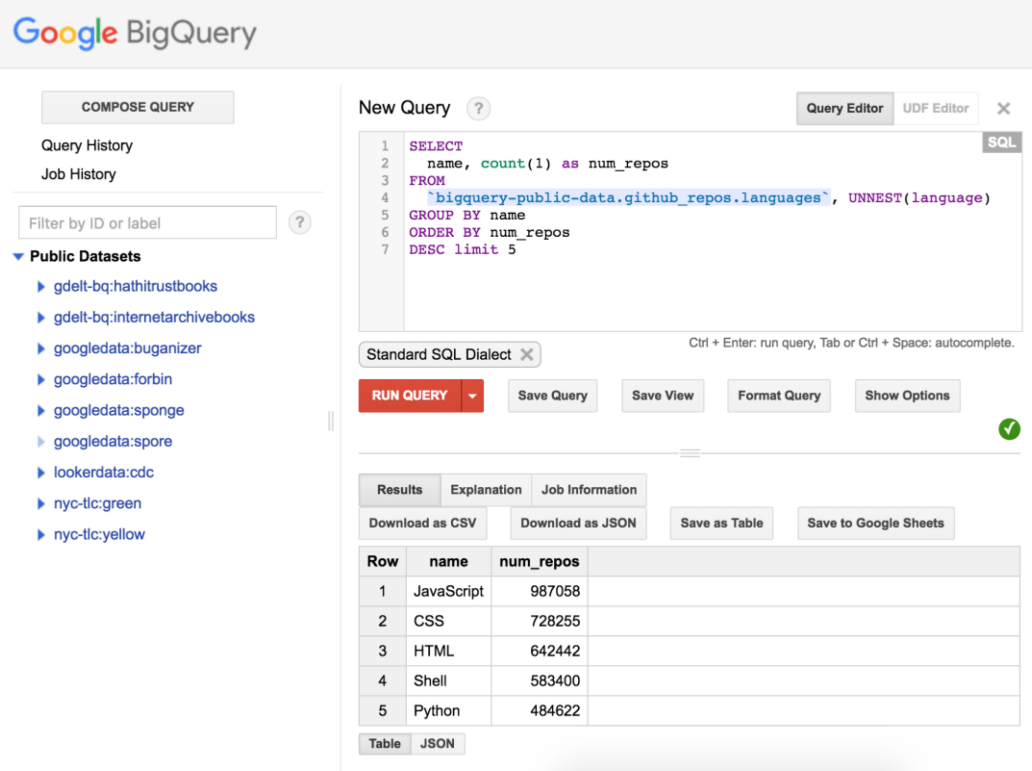
Źródło: https://arrow.apache.org/

# Platformy chmurowe

# Google Cloud Platform



# Google BigQuery



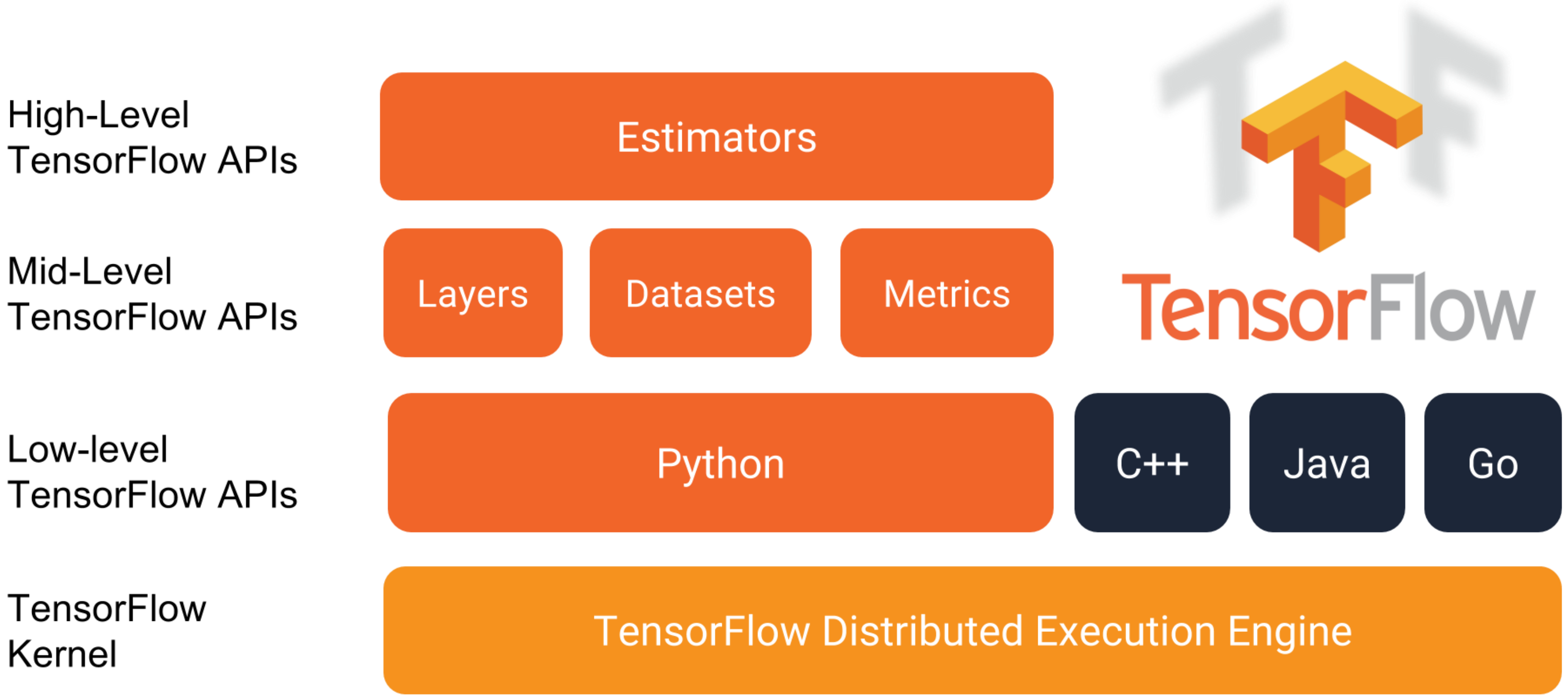
```sql
SELECT
  name, count(1) as num_repos
FROM
  `bigquery-public-data.github_repos.languages`, UNNEST(language)
GROUP BY name
ORDER BY num_repos
DESC limit 5
```

| Row | name | num_repos |
|---|---|---|
| 1 | JavaScript | 987058 |
| 2 | CSS | 728255 |
| 3 | HTML | 642442 |
| 4 | Shell | 583400 |
| 5 | Python | 484622 |

# Google BigQuery vs Pandas

```
# pip install pandas-gbq

projectid = "xxxxxxxx"

df = pd.read_gbq('SELECT * FROM test_dataset.test_table’,
                 index_col='index_column_name’,
                 col_order=['col1', 'col2', 'col3’],
                 projectid)

df.to_gbq(df, 'my_dataset.my_table', projectid, if_exists='fail')
```

# TensorFlow



High-Level TensorFlow APIs

Estimators

Mid-Level TensorFlow APIs

Layers

Datasets

Metrics

Low-level TensorFlow APIs

Python

C++

Java

Go

TensorFlow Kernel

TensorFlow Distributed Execution Engine

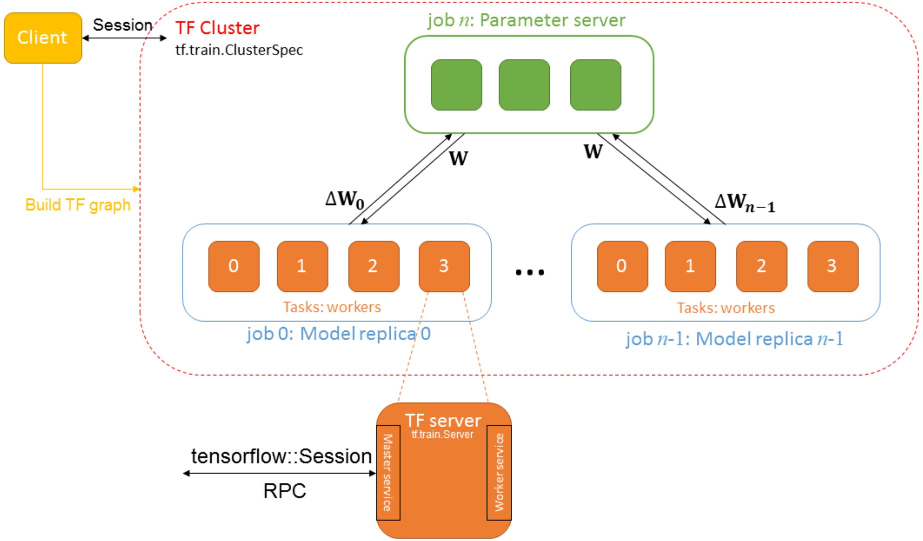
Źródło: [https://www.tensorflow.org/get_started/premade_estimators](https://www.tensorflow.org/get_started/premade_estimators)

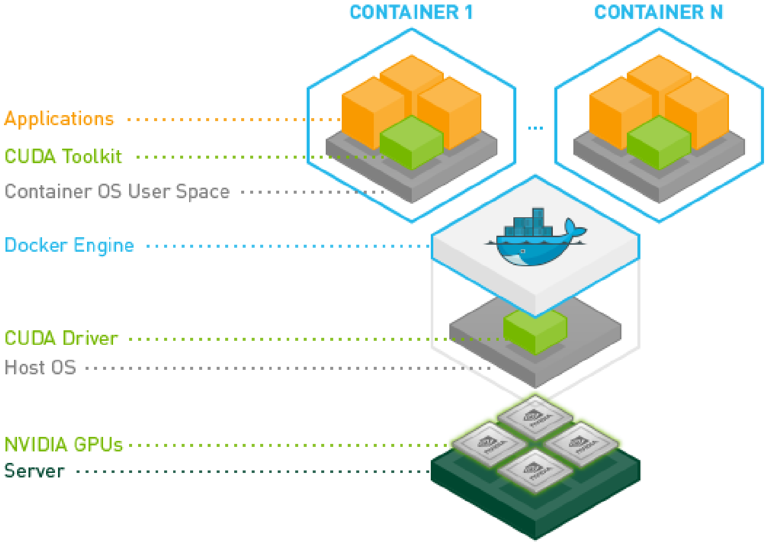
# TensorFlow Data

```
dataset2 = tf.data.Dataset.from_tensor_slices(
       (tf.random_uniform([4]),
        tf.random_uniform([4, 100], maxval=100, dtype=tf.int32)))
print(dataset2.output_types) # ==> "(tf.float32, tf.int32)"
print(dataset2.output_shapes) # ==> "((), (100,))"

dataset3 = tf.data.Dataset.zip((dataset1, dataset2))
print(dataset3.output_types) # ==> (tf.float32, (tf.float32, tf.int32))
print(dataset3.output_shapes) # ==> "(10, ((), (100,)))"

dataset1 = dataset1.map(lambda x: ...)
dataset2 = dataset2.flat_map(lambda x, y: ...)
dataset3 = dataset3.filter(lambda x, (y, z): ...)
```

Źródło: https://www.tensorflow.org/programmers_guide/datasets

# TensorFlow GPU & Distributed

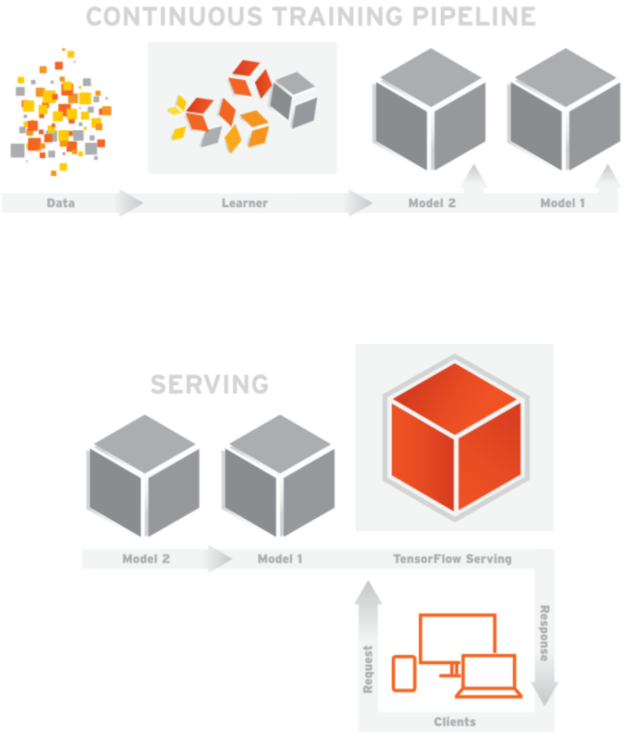




Źródło: http://www.pittnuts.com/2016/08/glossary-in-distributed-tensorflow/

Źródło: https://towardsdatascience.com/using-docker-to-set-up-a-deep-learning-environment-on-aws-6af37a78c551

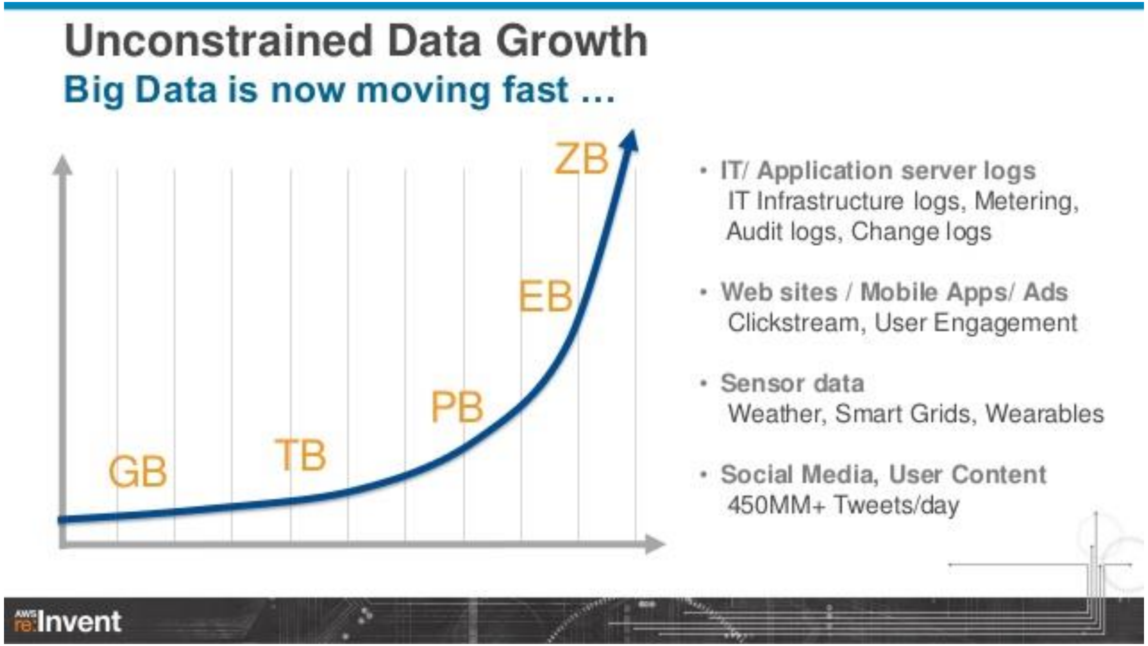
# TensorFlow Serving



Źródło: https://www.tensorflow.org/serving/

Źródło: https://cloud.google.com/products/machine-learning/

# Co przyniesie przyszłość? ¯_(ツ)_/¯



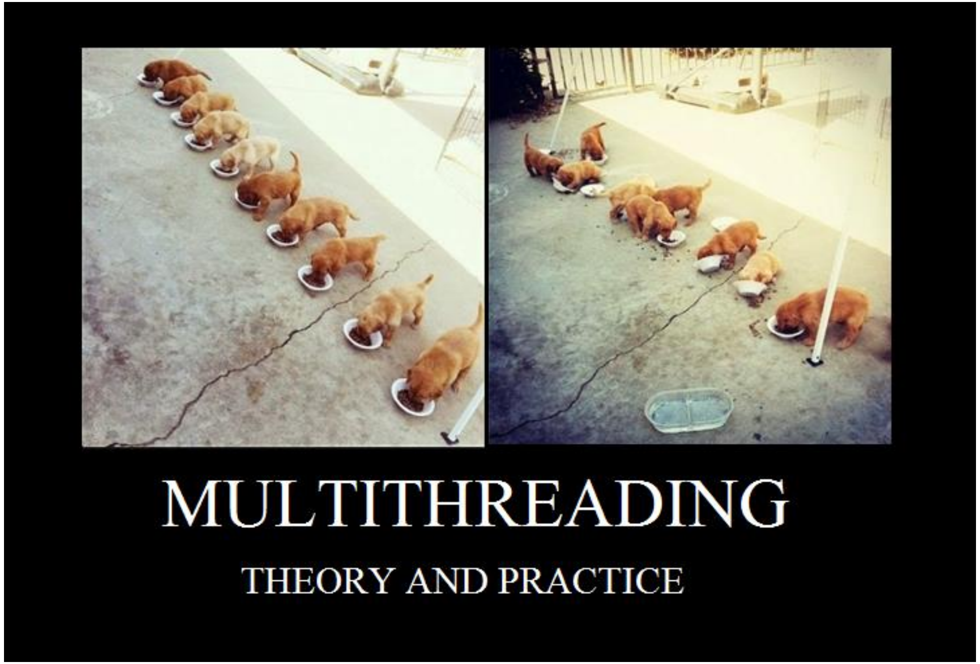
## Unconstrained Data Growth

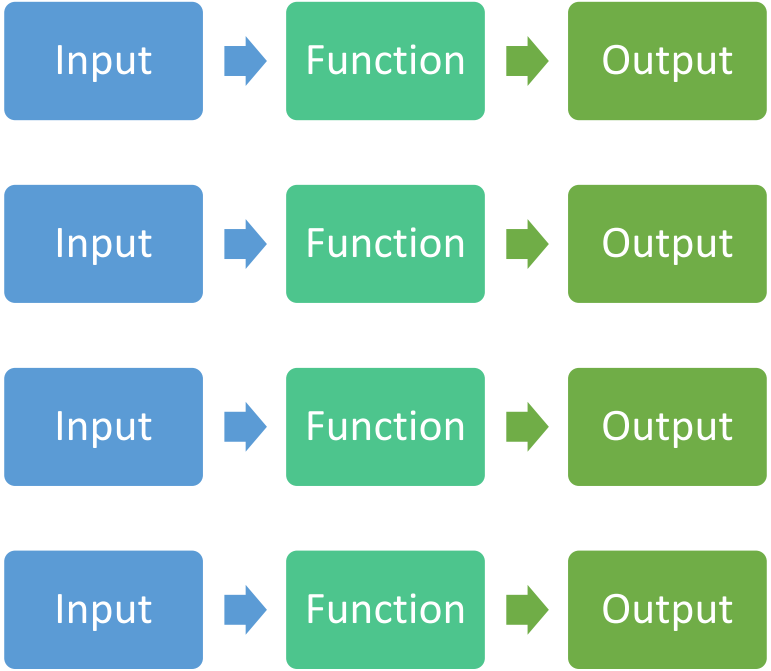
**Big Data is now moving fast …**

- **IT/ Application server logs**
  IT Infrastructure logs, Metering, Audit logs, Change logs
- **Web sites / Mobile Apps/ Ads**
  Clickstream, User Engagement
- **Sensor data**
  Weather, Smart Grids, Wearables
- **Social Media, User Content**
  450MM+ Tweets/day

Źródło: [https://www.slideshare.net/AmazonWebServices/introducing-amazon-kinesis-realtime-processing-of-streaming-big-data-bdt103-aws-reinvent-2013](https://www.slideshare.net/AmazonWebServices/introducing-amazon-kinesis-realtime-processing-of-streaming-big-data-bdt103-aws-reinvent-2013)

# Programowanie funkcyjne





Input → Function → Output

Input → Function → Output

Input → Function → Output

Input → Function → Output

# SQL

APACHE
Spark
TM

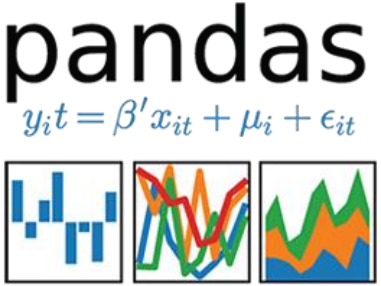
HIVE

presto

aws

Azure

Google Cloud


pandas
$y_it = \beta' x_{it} + \mu_i + \epsilon_{it}$

DASK

Twoja opinia na temat mojej prelekcji jest dla mnie bardzo ważna.

1. Wejdź w mój wykład znajdujący się w agendzie w aplikacji Eventory.
2. Oceń moją prelekcję i dodaj swój komentarz.

Dzięki temu będę wiedział/a, co Ci się podobało a co powinienem/am ulepszyć!

eventory

for better events

# Q & A

4Developers