PYSPARK IN PRACTICE

PYDATA LONDON 2016

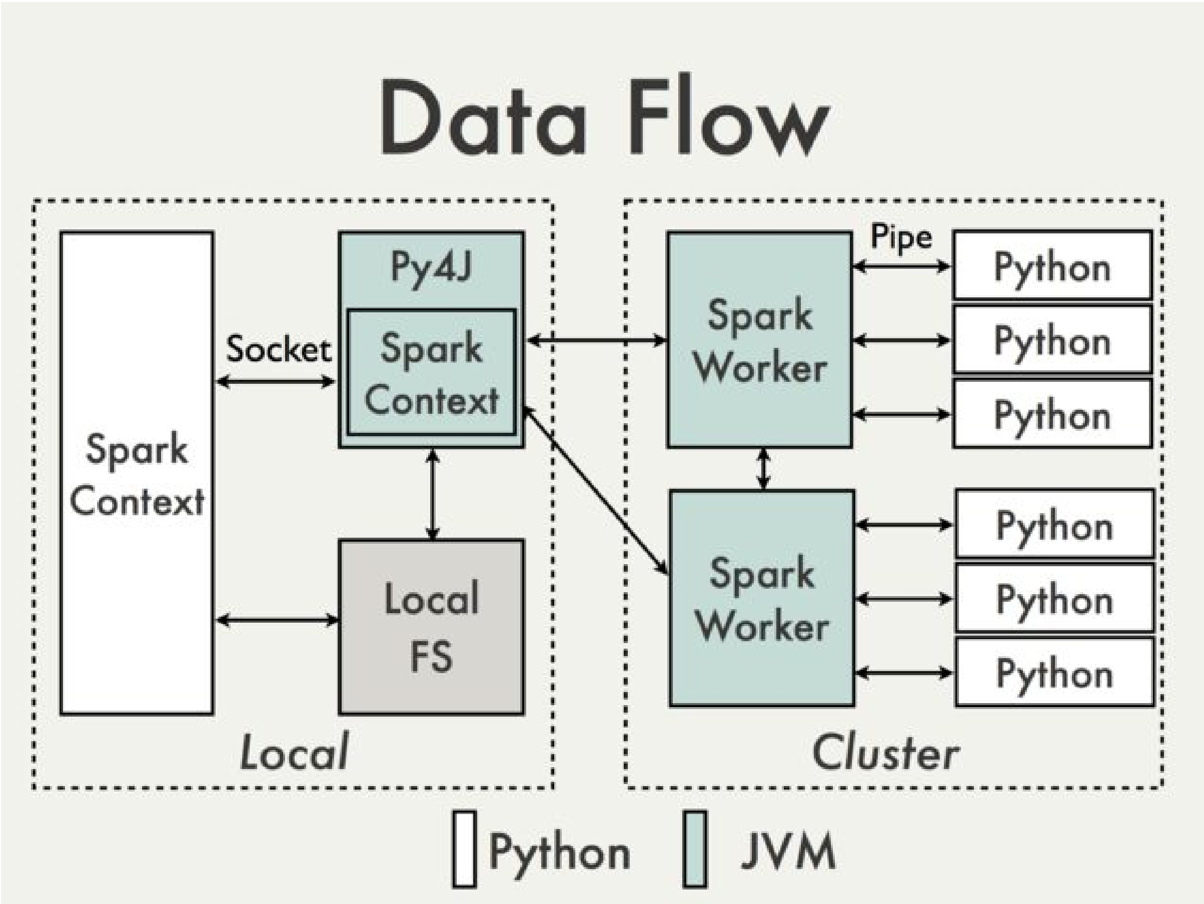
Ronert Obst Senior Data Scientist

Dat Tran Data Scientist

AGENDA

- Short introduction
- Data structures
- Configuration and performance
- Unit testing with PySpark
- Data pipeline management and workflows
- Online learning with PySpark streaming
- Operationalisation

WHAT IS PYSPARK?



DATA STRUCTURES

DATA STRUCTURES

- RDDs
- DataFrames
- DataSets

DATAFRAMES

- Built on top of RDD
- Include metadata
- Turns PySpark API calls into query plan
- Less flexibel than RDD
- Python UDFs impact performance, use builtin functions whenever possible
- HiveContext ftw

PYTHON DATAFRAME PERFORMANCE

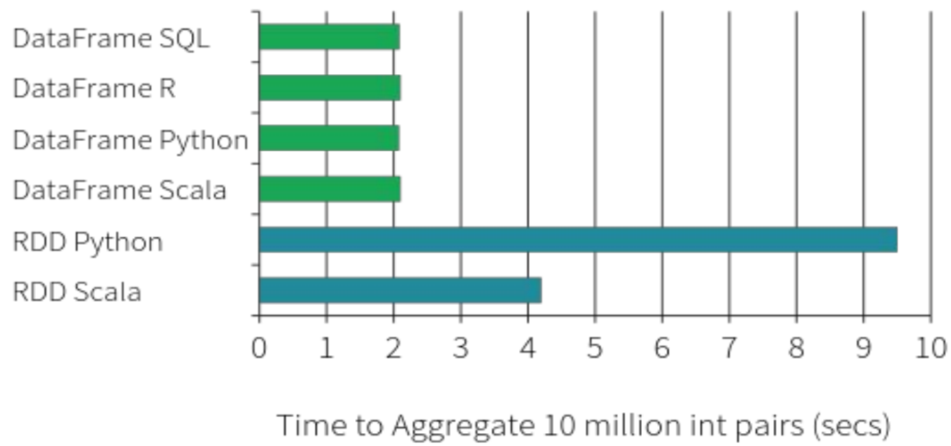
Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Using DataFrames

```
sqlCtx.table("people") \
   .groupBy("name") \
   .agg("name", avg("age")) \
   .collect()
```

(At least as much as possible!)



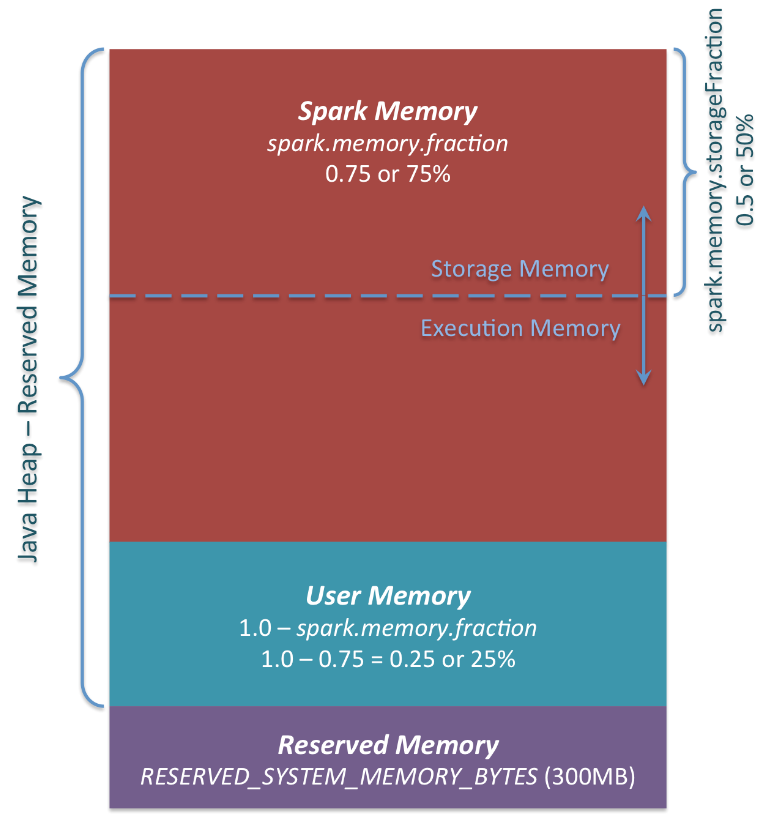
Source: Databricks - Spark in Production (Aaron Davidson)

CONFIGURATION AND PERFORMANCE

CONFIGURATION PRECEDENCE

1. Programatically (`SparkConf()`)
2. Command line (`spark-submit --master yarn --executor-memory 20G ...`)
3. Configuration files (`spark-defaults.conf, spark-env.sh`)

SPARK 1.6 MEMORY MANAGEMENT



Source: Spark Memory Management (Alexey Grishchenko) (https://0x0fff.com/spark-memory-management/)

WORKED EXAMPLE

CLUSTER HARDWARE:

- 6 nodes with 16 cores and 64 GB RAM each

YARN CONFIG:

- yarn.nodemanager.resource.memory-mb: 61 GB
- yarn.nodemanager.resource.cpu-vcores: 15

SPARK-DEFAULTS.CONF

- num-executors: 4 executors per node * 6 nodes = 24
- executor-cores: 6
- executor-memory: 11600 MB

PYTHON OOM

More Python memory, e.g. scikit-learn

- Reduce concurrency (4 executors per node with 4 executor cores)
- Set spark.python.worker.memory = 1024 MB (default is 512 MB)
- Leaves spark.executor.memory = 10600 MB

OTHER USEFUL SETTINGS

- Save substantial memory: `spark.rdd.compress = true`
- Relaunch stragglers: `spark.speculation = true`
- Fix Python 3 hashseed issue:
  `spark.executorEnv.PYTHONHASHSEED = 0`

TUNING PERFORMANCE

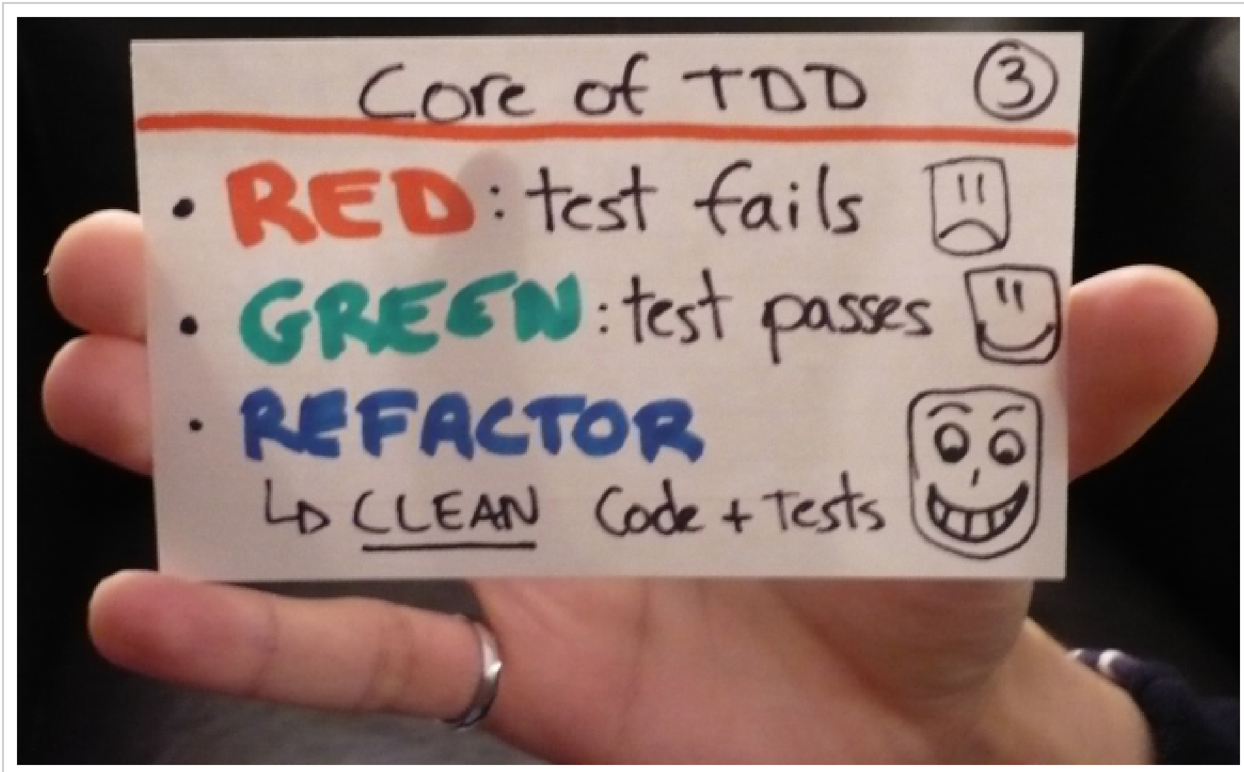
- Re-use RDDs with `cache()` (set storage level to `MEMORY_AND_DISK`)
- Avoid groupByKey() on RDDs
- Distribute by key `data = sql_context.read.parquet(hdfs://...).repartition(N_PARTITIONS, "key")`
- Use `treeReduce` and `treeAggregate` instead of `reduce` and `aggregate`
- `sc.broadcast(broadcast_var)` small tables containing frequently accessed data

UNIT TESTING WITH PYSPARK

MOST DATA SCIENTIST WRITE CODE THAT JUST WORKS

BUGS ARE EVERYWHERE...

TDD IS THE SOLUTION!



THE REALITY CAN BE DIFFICULT…

EXPLORATION PHASE

PRODUCTION PHASE

```
# Import script modules here
import clustering

class ClusteringTest(unittest.TestCase):

    def setUp(self):
        """Create a single node Spark application."""
        conf = SparkConf()
        conf.set("spark.executor.memory", "1g")
        conf.set("spark.cores.max", "1")
        conf.set("spark.app.name", "nosetest")
        self.sc = SparkContext(conf=conf)
        self.mock_df = self.mock_data()

    def tearDown(self):
        """Stop the SparkContext."""
        self.sc.stop()
```

TEST:

```
def test_assign_cluster(self):
    """Check if rows are labeled are as expected."""
    input_df = clustering.convert_df(self.sc, self.mock_df)
    scaled_df = clustering.rescale_df(input_df)
    label_df = clustering.assign_cluster(scaled_df)
    self.assertEqual(label_df.map(lambda x: x.label).collect(), [0, 0, 0,
1, 1])
```

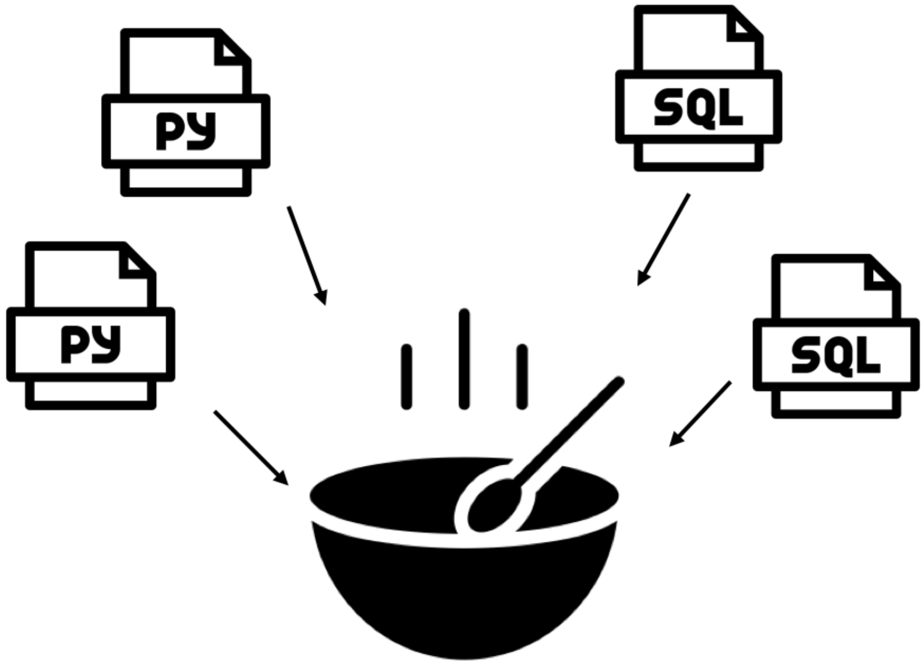
IMPLEMENTATION:

```
def assign_cluster(data):
    """Train kmeans on rescaled data and then label the rescaled data."""
    kmeans = KMeans(k=2, seed=1, featuresCol="features_scaled", predictio
nCol="label")
    model = kmeans.fit(data)
    label_df = model.transform(data)
    return label_df
```

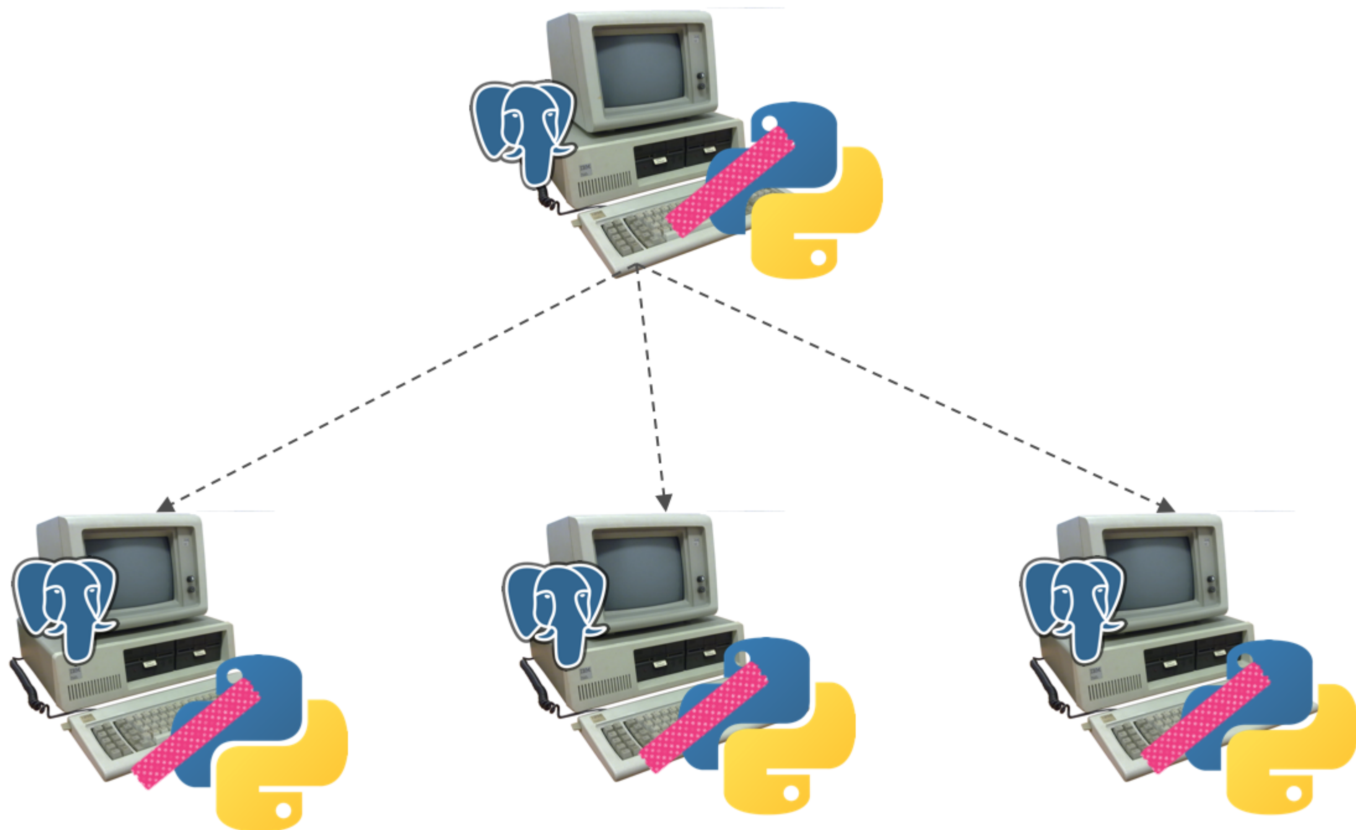
Full example: https://github.com/datitran/spark-tdd-example (https://github.com/datitran/spark-tdd-example)

DATA PIPELINE MANAGEMENT AND WORKFLOWS

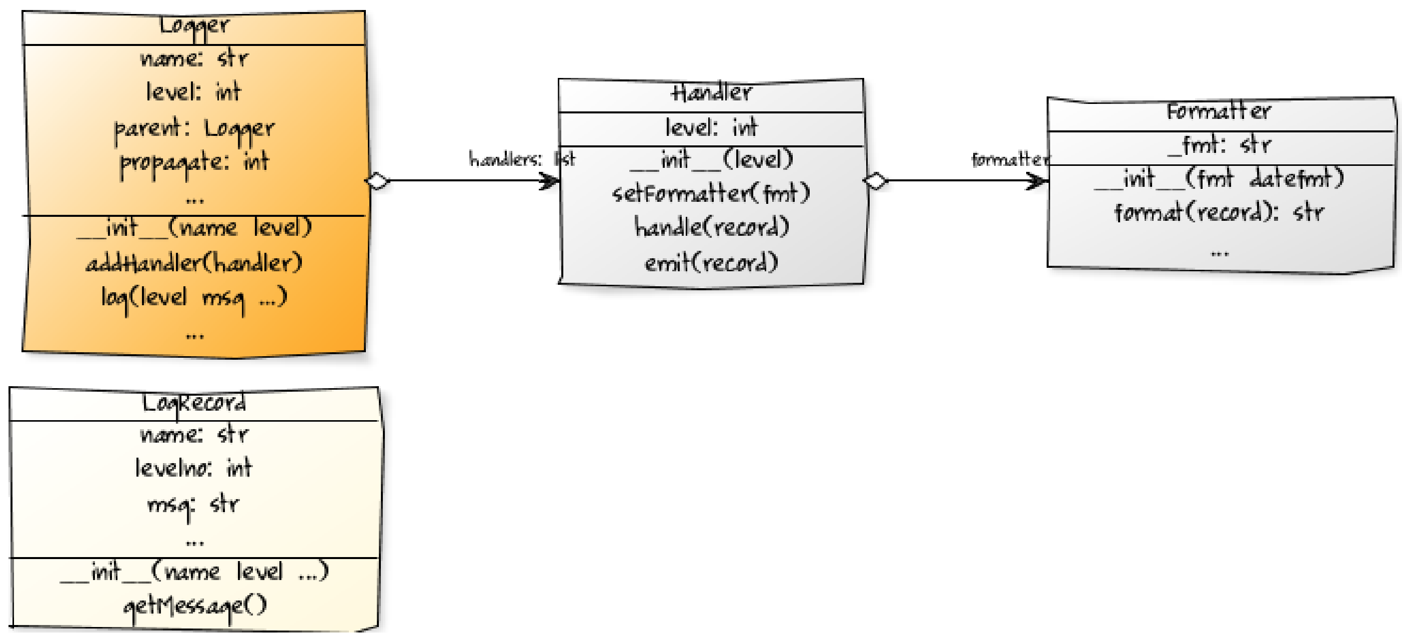
PIPELINE MANAGEMENT CAN BE DIFFICULT…



IN THE PAST...



CUSTOM LOGGERS



- 100% Python
- Web dashboard
- Parallel workers
- Central scheduler
- Many templates for various tasks e.g. Spark, SQL etc.
- Well configured logger
- Parameters

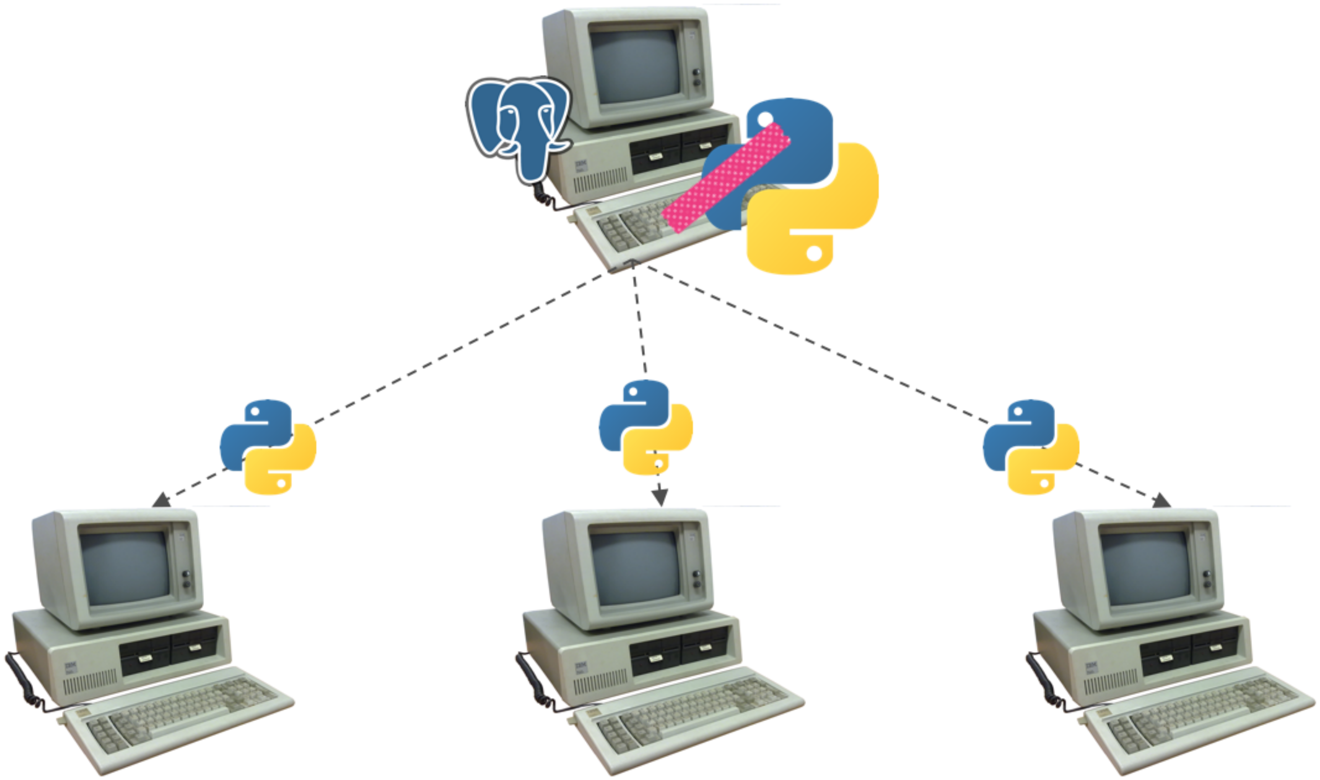
EXAMPLE SPARK TASK

```python
class ClusteringTask(SparkSubmitTask):
    """Assign cluster label to some data."""
    date = luigi.DateParameter()
    num_k = luigi.IntParameter()

    name = "Clustering with PySpark"
    app = "../clustering.py"

    def app_options(self):
        return [self.num_k]

    def requires(self):
        return [FeatureEngineeringTask(date=self.date)]

    def output(self):
        return HdfsTarget("hdfs://...")
```

EXAMPLE CONFIG FILE

```
client.cfg

[resources]
spark: 2

[spark]
master: yarn
executor-cores: 3
num-executors: 11
executor-memory: 20G
```

WITH LUIGI



CAVEATS

- No built-in job trigger
- Documentation could be better
- Logging can be too much when having multiple workers
- Duplicated parameters in all downstream tasks

ONLINE LEARNING WITH PYSPARK STREAMING

ONLINE LEARNING WITH PYSPARK

```
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.regression import StreamingLinearRegressionWithSGD

def parse(lp):
    label = float(lp[lp.find('(') + 1: lp.find(',')])
    vec = Vectors.dense(lp[lp.find('[') + 1: lp.find(']')].split(','))
    return LabeledPoint(label, vec)

stream = scc.socketTextStream("localhost", 9999).map(parse)
numFeatures = 3
model = StreamingLinearRegressionWithSGD()
model.setInitialWeights([0.0, 0.0, 0.0])
model.trainOn(stream)
print(model.predictOn(stream.map(lambda lp: (lp.label, lp.features))))

ssc.start()
ssc.awaitTermination()
```

OPERATIONALISATION

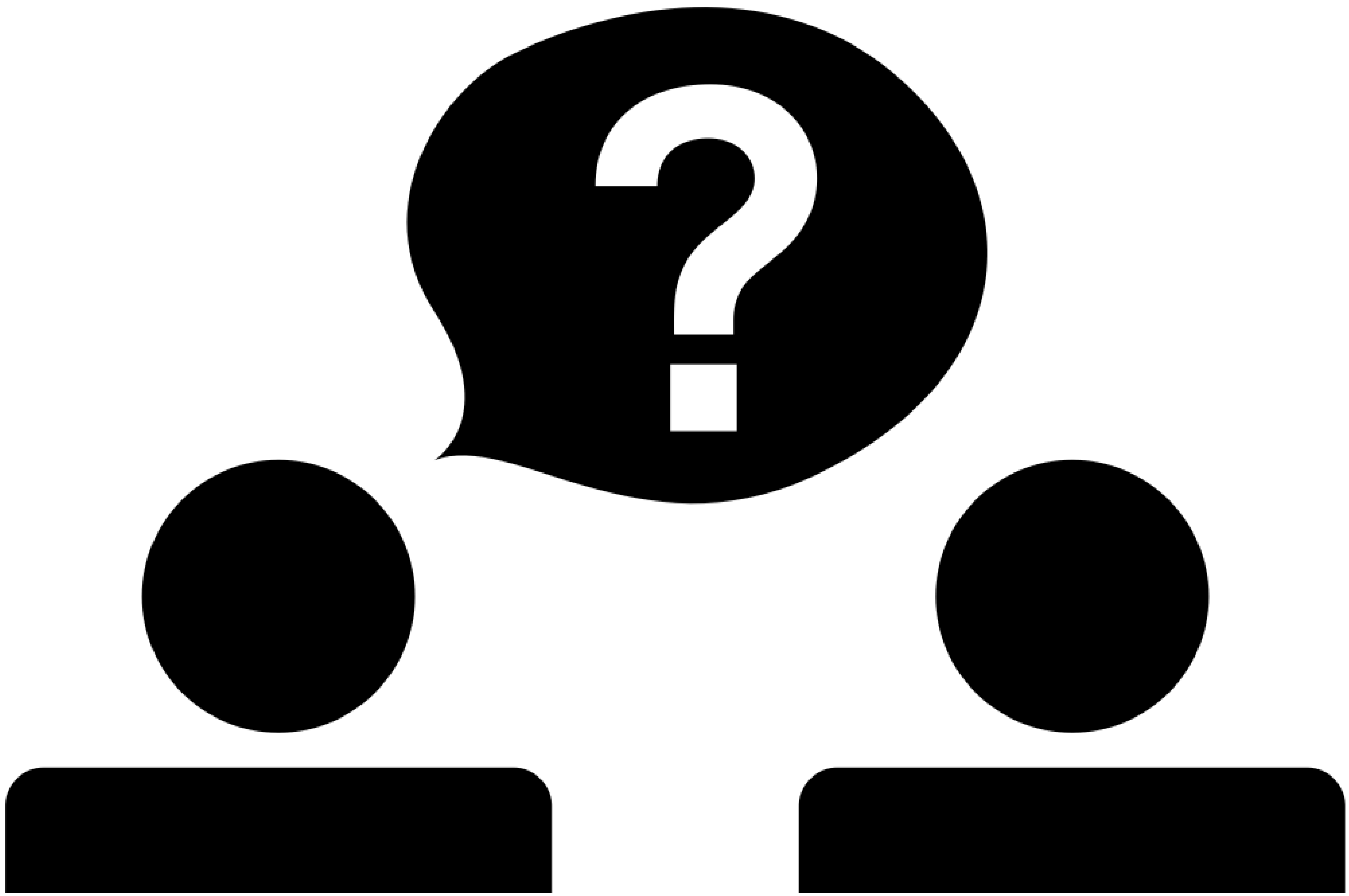
RESTFUL API EXAMPLE WITH FLASK

```
import pandas as pd
from flask import Flask, request

app = Flask(__name__)

# Load model stored on redis or hdfs
app.model = load_model()

def predict(model, features):
    """Use the trained cluster centers to assign cluster to new feature
s."""
    differences = pd.DataFrame(
        model.values - features.values, columns=model.columns)
    differences_square = differences.apply(lambda x: x**2)
    col_sums = differences_square.sum(axis=1)
    label = col_sums.idxmin()
    return int(label)
```

```python
@app.route("/clustering", methods=["POST"])
def clustering():
    """
    curl -i -H "Content-Type: application/json" -X POST -d @example.json
"x.x.x.x:5000/clustering"
    """
    features_json = request.json
    features_df = pd.DataFrame(features_json)
    return str(predict(load_model(), feature_df))
```

DEPLOY YOUR APPS WITH `CF PUSH`

- Platform as a Service (PaaS)
- "Here is my source code. Run it on the cloud for me. I do not care how." (Onsi Fakhouri)
- Trial version: https://run.pivotal.io/ (https://run.pivotal.io/)
- Python Cloud Foundry examples: https://github.com/ihuston/python-cf-examples (https://github.com/ihuston/python-cf-examples)



Pivotal

Transforming How The World Builds Software