Big Data Analysis in Python

Data Wrangling with Apache Spark, Pandas, and Matplotlib

Jared M. Smith
@jaredthecoder

Cyber Security Researcher
Oak Ridge National Laboratory

About Me

- Cyber Security Researcher and Project Lead at Oak Ridge National Laboratory
- CS PhD Student at the University of Tennessee, Knoxville
- Founder of HackUTK and VolHacks
- Technical Mentor at Knoxville Entrepreneur Center
- Technical Interests: Security (Network, Forensics, Vehicle), Python, Machine Learning, Malware Analysis, other language strengths (Java, C, C++, JavaScript, Rust, Go)

“Getting information off the Internet is like taking a drink from a fire hydrant.”

–Mitch Kapor
Founder of Lotus Software and inventor of the first spreadsheet, VisiCalc

Why Data Science?

Volume

Velocity

Variety

Veracity

How can I handle that much data!?

Setup

- **Goal:** Compute total of randomly generated house prices per zip code for all U.S. zip codes

- **File Format:** *zip_code*, *house_price*

- **Size:** 43583 total zip codes x 100 house prices for each zip code = **4.35 million entries**

The Code

```python
#!/usr/bin/env python

import timeit
import csv


def python():
    counts = {}
    with open('hugefile.csv') as f:
        reader = csv.DictReader(f)
        for row in reader:
            if counts.get(row['zip_code']) is None:
                counts[row['zip_code']] = float(row['house_price'])
            else:
                counts[row['zip_code']] += float(row['house_price'])


if __name__ == '__main__':
    elapsed_time = timeit.timeit(python, number=1)
    print(elapsed_time)
```

15.71s

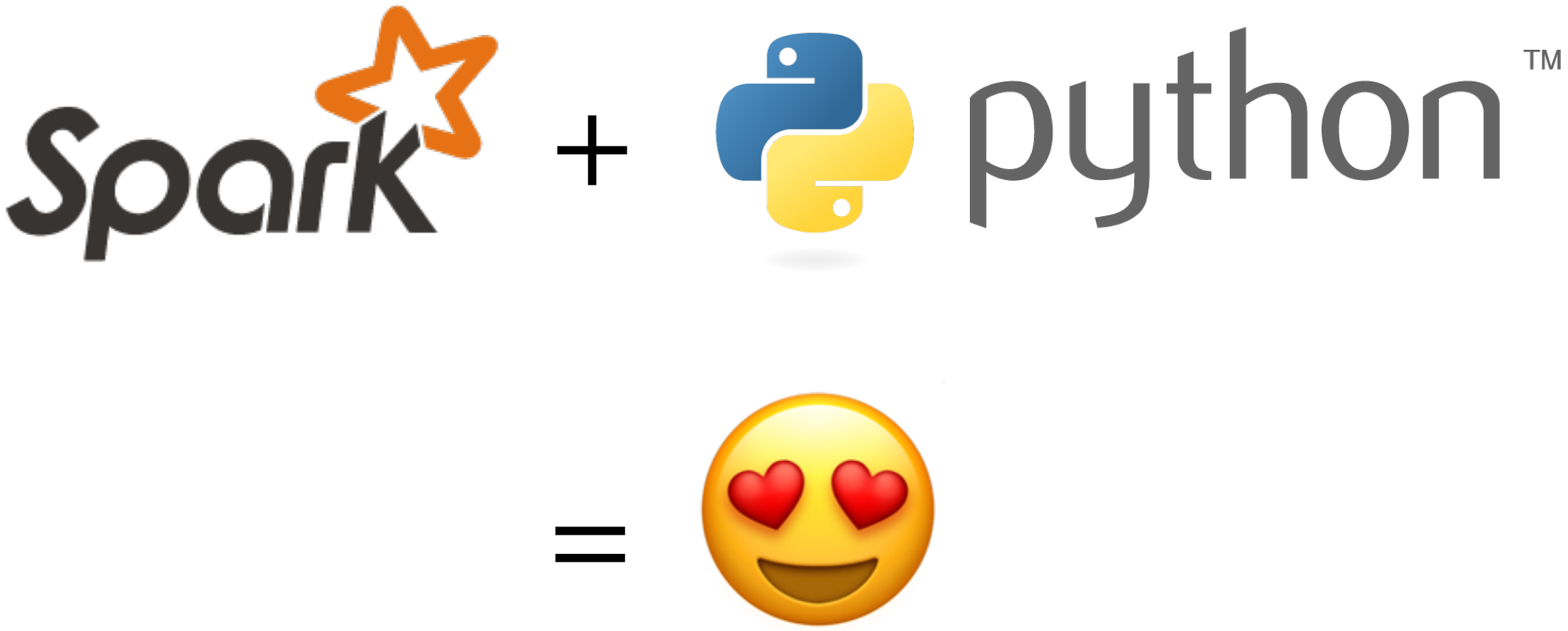
Machine Stats:

Macbook Pro (Late 2015 Model)

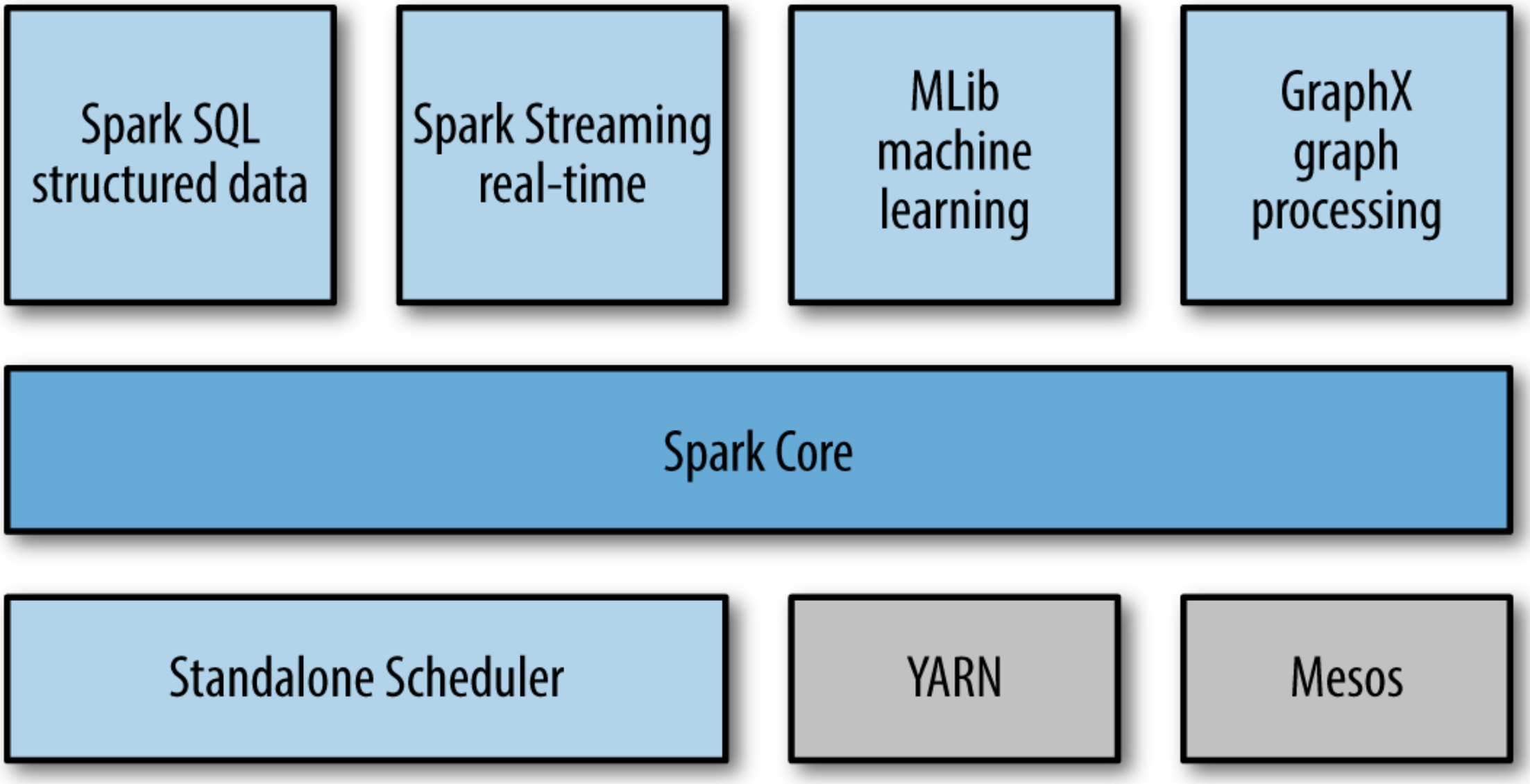
Quad-Core 2.5 GHz Intel Core i7

16 GB 1600 MHz DDR3

We can do better!


Spark
+
python™
=

What’s Apache Spark?


Spark SQL
structured data
Spark Streaming
real-time
MLib
machine
learning
GraphX
graph
processing
Spark Core
Standalone Scheduler
YARN
Mesos

Overview

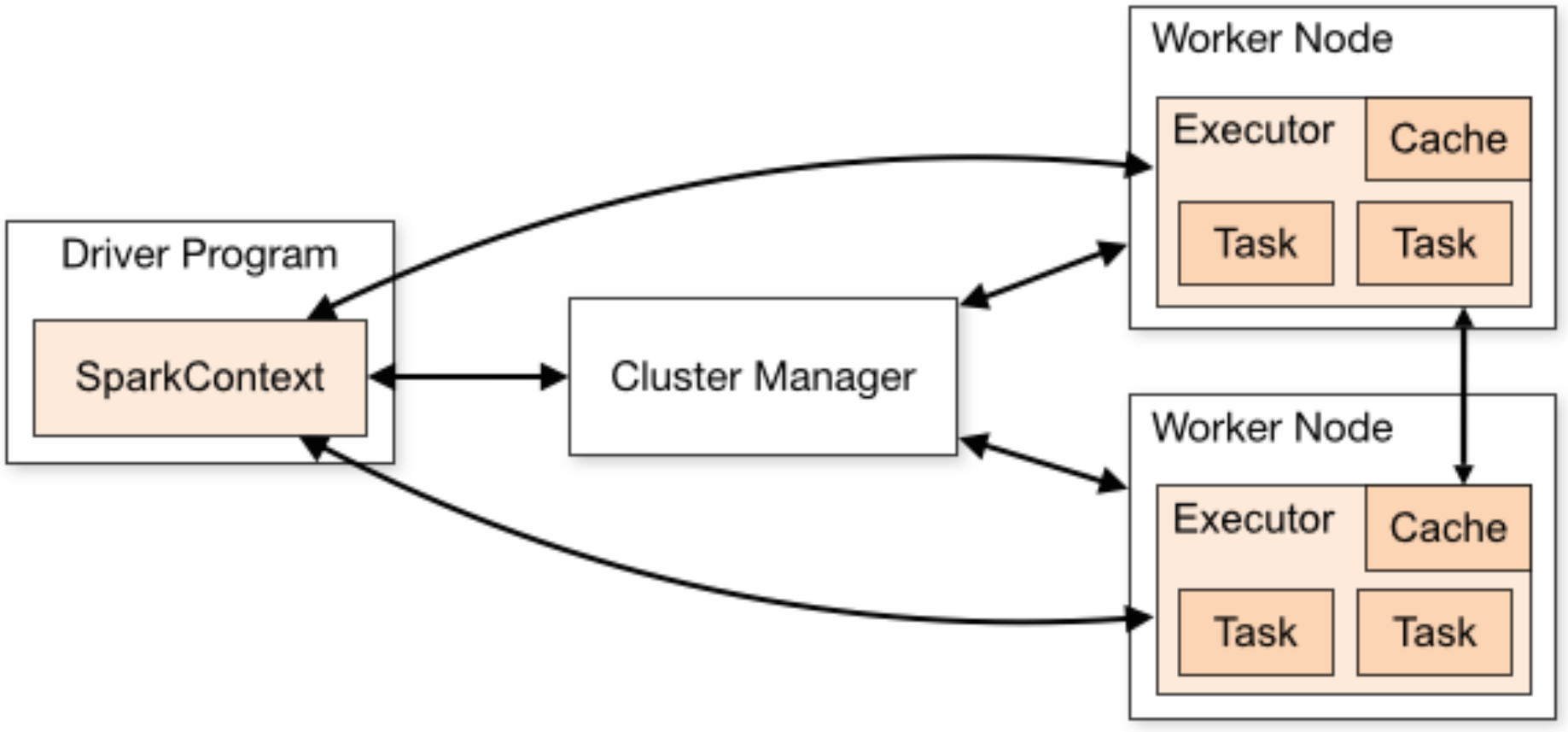
- Speed:
  - Uses a DAG (Directed Acyclic Graph) execution engine supporting cyclic data flow and in-memory computing
- Usability:
  - Over 80 high-level operators for building parallel applications and with interactive shells in Scala, Python, and R

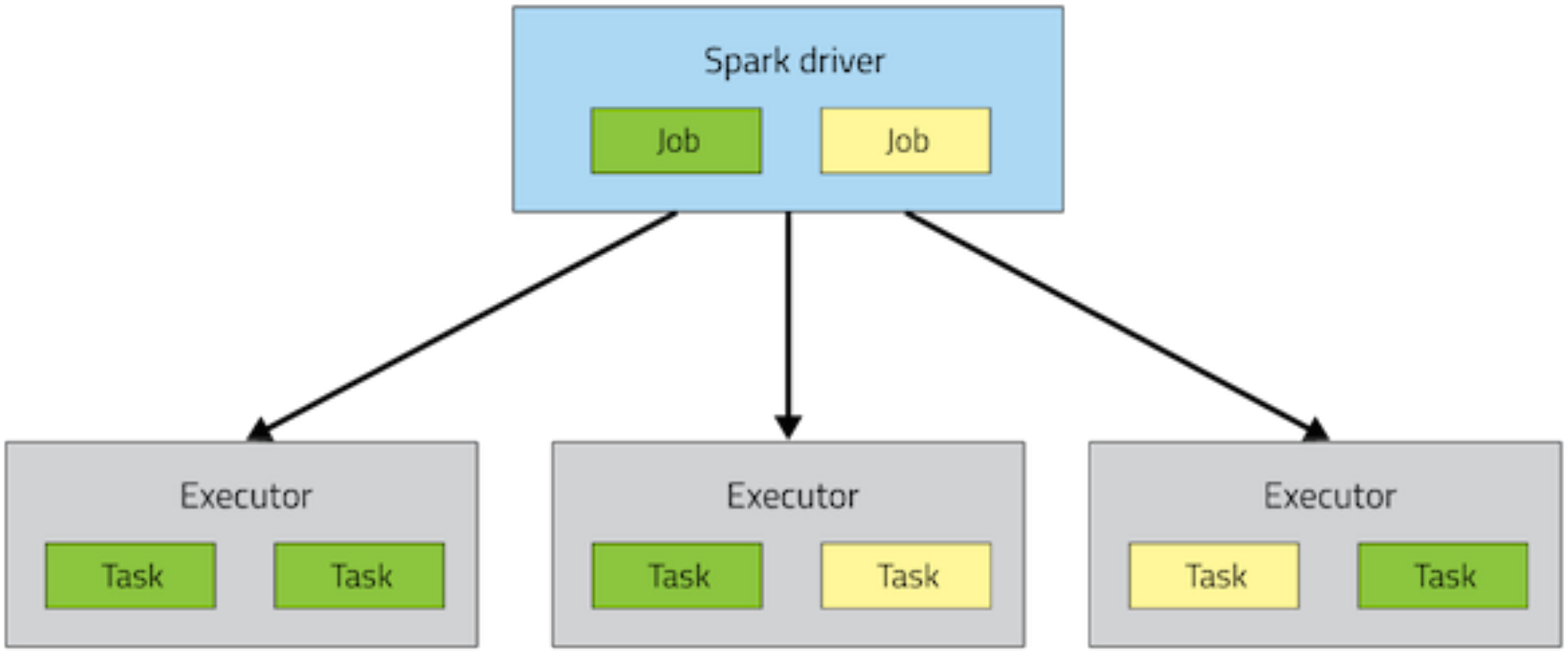
Building Blocks

- Task
  - A piece of computation
  - Example: a portion of a dataset that needs to be filtered will be done in a task

Building Blocks

- Stage
  - A set of tasks that define an operation in the DAG
  - Example: a filter of a dataset is one stage, and a shuffle to coalesce data between two previous stages could be another stage


Worker Node
Executor
Cache
Task
Task
Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task


Spark driver
Job
Job
Executor
Task
Task
Executor
Task
Task
Executor
Task
Task

Spark Semantics

- Spark's principle data structure is an RDD
  - Resilient Distributed Dataset
- RDD's have operations on them like *map*, *reduce*, *count*, *max*, *min*, statistics functions (*mean*, *variance*, etc.), etc.
- And more advanced operations: *combineByKey*, *groupByKey*, *leftOuterJoin*, *sortByKey*, etc.

Assumptions

- You've downloaded Apache Spark (with Hadoop built-in) from http://spark.apache.org/downloads.html

- You're running Spark in standalone mode

- You're using PySpark, the officially-supported built-in Python API for Apache Spark

- The "spark-submit" executable is accessible through your terminal, which we need to run PySpark jobs

The Code

```python
#!/usr/bin/env python

import timeit
import csv

from pyspark import SparkConf, SparkContext


def wrapper(func, *args, **kwargs):
    def wrapped():
        return func(*args, **kwargs)
    return wrapped


def spark(sc, pairs):
    sums = pairs.combineByKey(lambda value: value,
                              lambda x, value: x + value,
                              lambda x, y: x + y)
    sums.collect()


if __name__ == '__main__':

    # Setup
    conf = (SparkConf()
         .setMaster("local")
         .setAppName("LocalSparkExample")
         .set("spark.executor.memory", "1g"))
    sc = SparkContext(conf=conf)

    data = []
    with open('hugefile.csv') as f:
         reader = csv.DictReader(f)
         for row in reader:
            data.append((int(row['zip_code']), float(row['house_price'])))

    pairs = sc.parallelize(data)

    # Wrap up arguments for timing call
    wrapped = wrapper(spark, sc, pairs)

    # Call test function
    elapsed_time = timeit.timeit(wrapped, number=1)
    print(elapsed_time)
```

```
1   #!/usr/bin/env python
2
3   import timeit
4   import csv
5
6   from pyspark import SparkConf, SparkContext
```

```
24      # Setup
25      conf = (SparkConf()
26           .setMaster("local")
27           .setAppName("LocalSparkExample")
28           .set("spark.executor.memory", "1g"))
29      sc = SparkContext(conf=conf)
```

```
31      data = []
32      with open('hugefile.csv') as f:
33          reader = csv.DictReader(f)
34          for row in reader:
35              data.append((int(row['zip_code']), float(row['house_price'])))
36
37      pairs = sc.parallelize(data)
```

```
 9    def wrapper(func, *args, **kwargs):
10        def wrapped():
11            return func(*args, **kwargs)
12        return wrapped
```

```
39    # Wrap up arguments for timing call
40    wrapped = wrapper(spark, sc, pairs)
41
42    # Call test function
43    elapsed_time = timeit.timeit(wrapped, number=1)
44    print(elapsed_time)
```

```
15    def spark(sc, pairs):
16        sums = pairs.combineByKey(lambda value: value,
17                                  lambda x, value: x + value,
18                                  lambda x, y: x + y)
19        sums.collect()
```

combineByKey(*createCombiner*, *mergeValue*, *mergeCombiners*, *numPartitions=None*, *partitionFunc=<function portable_hash at 0x7fc35dbc8e60>*)

Generic function to combine the elements for each key using a custom set of aggregation functions.

Turns an RDD[(K, V)] into a result of type RDD[(K, C)], for a “combined type” C.

Users provide three functions:

- **createCombiner**, which turns a V into a C (e.g., creates a one-element list)
- **mergeValue**, to merge a V into a C (e.g., adds it to the end of a list)
- **mergeCombiners**, to combine two C’s into a single one.

In addition, users can control the partitioning of the output RDD.

Note: V and C can be different – for example, one might group an RDD of type (Int, Int) into an RDD of type (Int, List[Int]).

3.50s

Machine Stats:

Macbook Pro (Late 2015 Model)

Quad-Core 2.5 GHz Intel Core i7

16 GB 1600 MHz DDR3

What about *real* data?

The Power of Spark

- Runs on top of existing Hadoop clusters (data processing framework, originated based on Google's MapReduce paper)

- You can scale out your cluster, adding memory and processors

- At Spark Summit 2015:

  - 8000 node cluster at Tencent

  - 1 Petabyte single job at Alibaba

1,000,000,000,000,000,000

bytes

100 x the contents of the entire US Library of Congress

So, why Spark and not X?

1. Fast

2. Friendly

3. Community

Why not?

1. Not much data

2. You can’t use Java or run the JVM

3. You don’t have sufficient resources

What if I want to use something else?

Other Options

- Python Dask (http://dask.pydata.org/en/latest/)
- Apache Storm (http://storm.apache.org/)
- Apache Flink (https://flink.apache.org/)
- …

What if I don’t have a lot of data?

What if I just need to know about my data?

Python can help there too!

First…

ANACONDA®

Anaconda

- Python distribution for data science
- 100 of the most popular Python, R, and Scala packages for data science are built-in
- Install over 720 more or build your own with *conda*, the built-in dependency and environment manager
- Program Python (2/3), R, or Julia in the browser interactively with Jupyter

Pandas

- Python data analysis toolkit
- Provides two fundamental data structures
    - Series (1-D) and DataFrame (2-D)
- Provides operations for merging, reshaping, grouping, aligning, slicing, and more
- Makes graphing as easy as calling *.plot()*

Matplotlib

- Matplotlib is a popular data visualization library in Python
- Inspired by the Matlab interface to graphing
- Can be rough around the edges, though very robust
- Publication ready

Other Python Data Visualization Libraries

- Seaborn - http://seaborn.pydata.org/
- Altair - https://altair-viz.github.io/
- ggplot - http://ggplot.yhathq.com/
- Bokeh - http://bokeh.pydata.org/en/latest/
- Plotly - https://plot.ly/python/

Pandas + Viz Tools

=

Demo at https://goo.gl/zCthsO

on Pandas, Bokeh, Matplotlib, Seaborn

Or https://github.com/jaredthecoder/pytn2017

Takeaways

- Big data processing can’t always be done with only Python
- Consider frameworks like Apache Spark or Dask to do large-scale processing
- To manipulate and explore data on a small scale, use libraries like Pandas and Matplotlib

Q&A

Twitter:	twitter.com/jaredthecoder
GitHub:	github.com/jaredthecoder
LinkedIn	linkedin.com/in/jaredthecoder
Website:	jaredthecoder.com
Blog:	blog.jaredthecoder.com
Email:	jared@jaredsmith.io