Deploying Data Science Systems

Dr. Daniel Schien

Daniel.schien@bristol.ac.uk

Outline

- Recap
- Motivation
- Data Science Process
- Cloud, Docker
- Big Data

Recap

- Lec 1 Intro
- Lec 2 Data Ingress
- Lec 3 Recommender Systems
- Lec 4 Databases
- Lec 5 Data Wrangling
- Lec 6 Data Fusion
- Lec 7 Data Exploration
- Lec 8 Data Visualisation
- Lec 9 Data sharing, privacy and anonymisation
- **Lec 10 Deploying data science systems**
- Lec 11 The future of data science

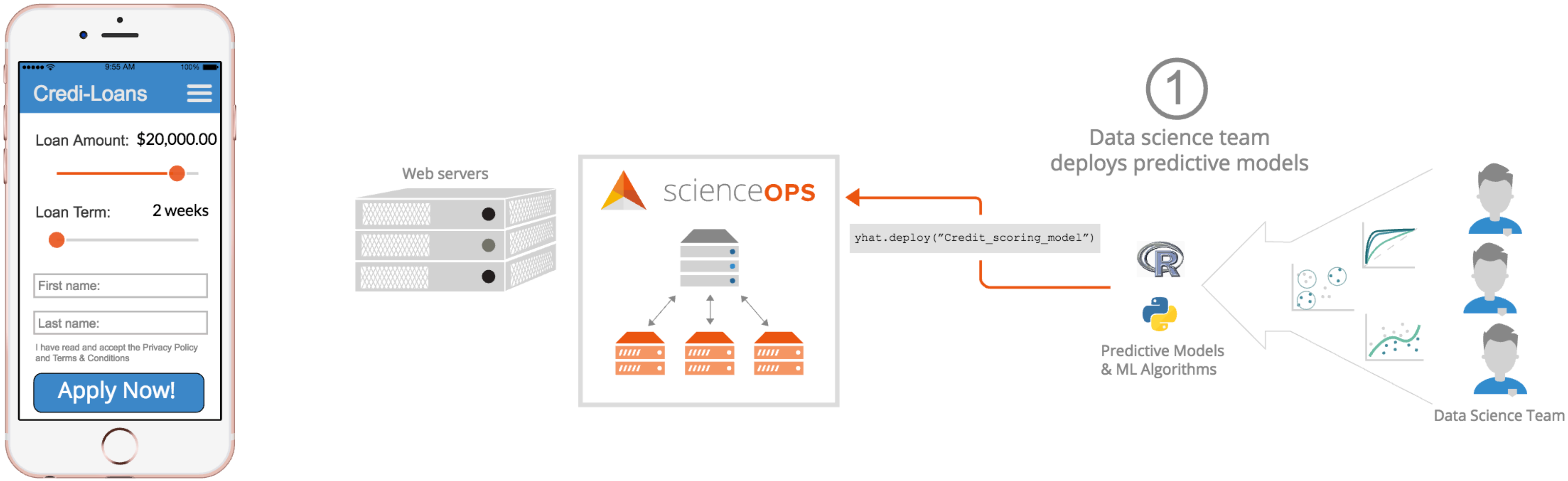
Learning Outcomes

- deployment and knowledge of **infrastructure** with the role of data scientists
- Have awareness for some key properties in the data science **software process**
- Have awareness for the key concepts in **Big Data** infrastructures
- Have an overview of key **cloud** deployment building blocks
- Be able to **package an ipython notebook** for cloud deployment

- Question: What is deployment?

Deployment Concerns

- On your machine



- Question: How does deployment overlap with the role of the data scientist?

The data science process

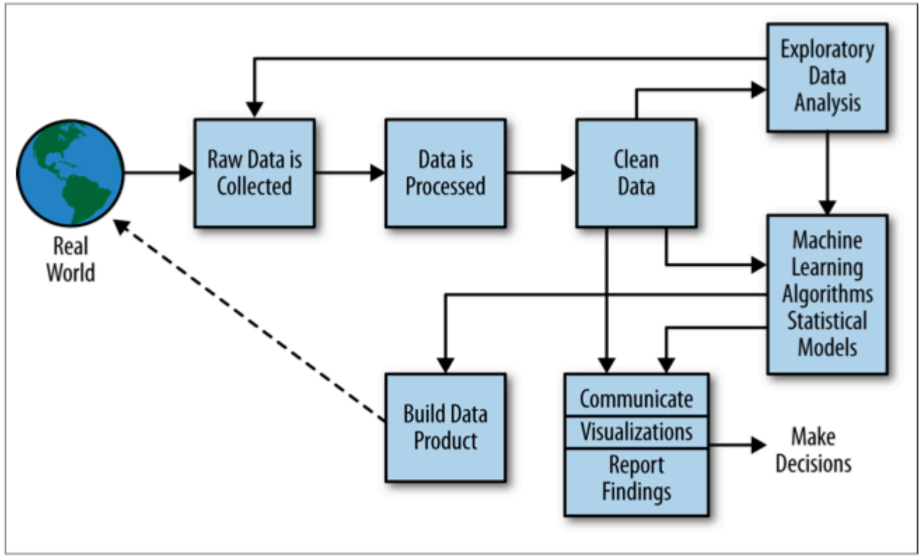
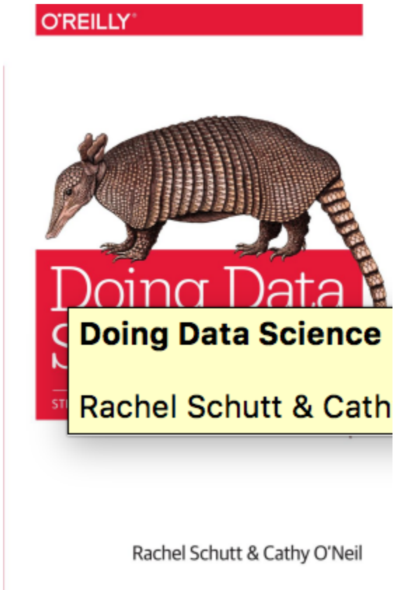
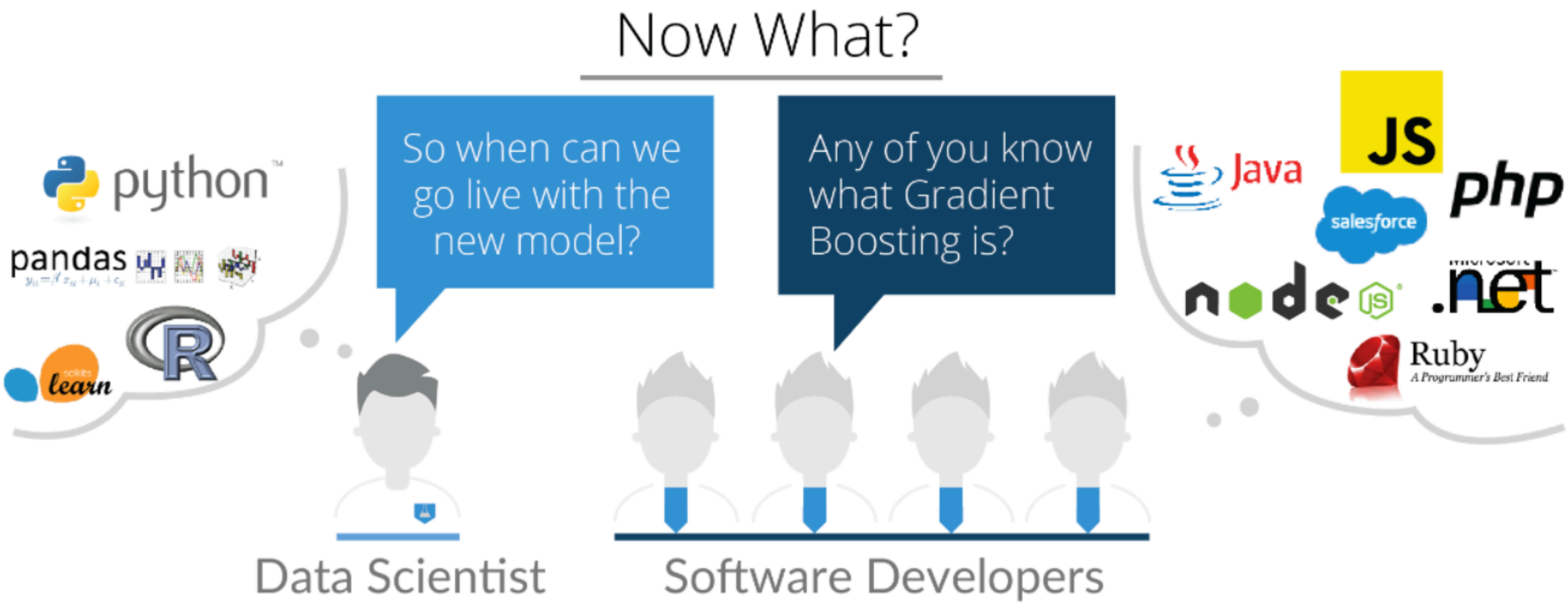


Figure 2-2. The data science process



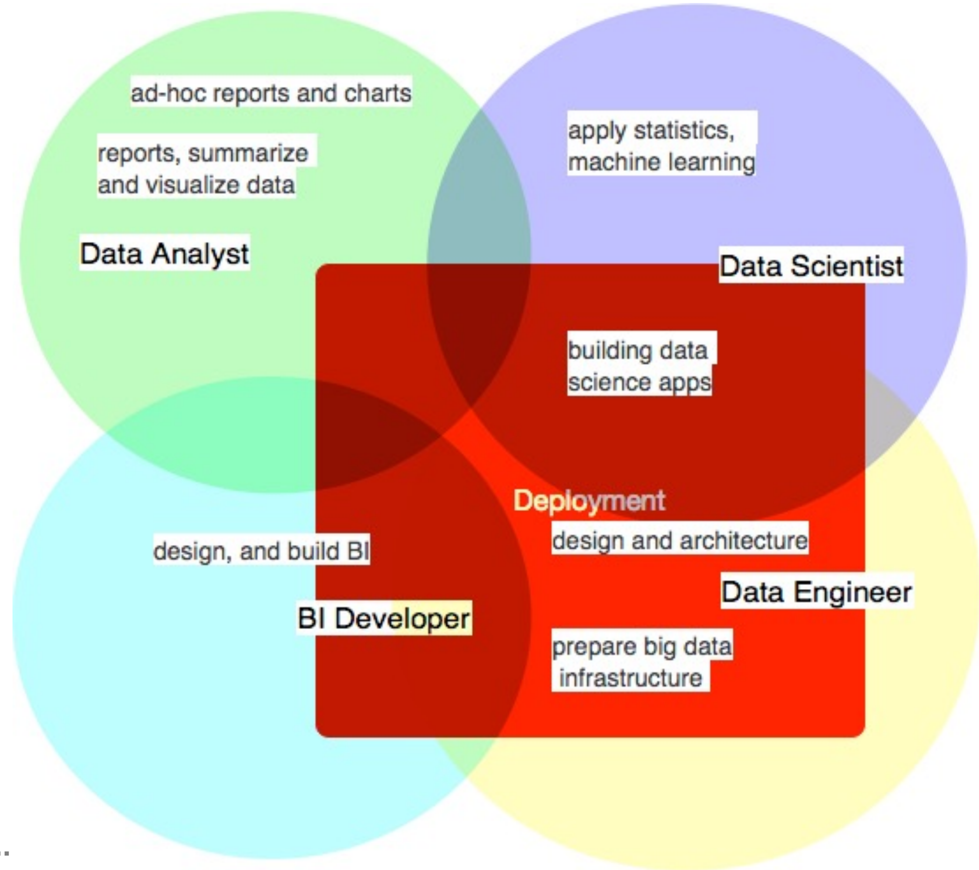
Data Products

- Business Intelligence and Reports
- Data Analytics Systems
  - Commercial systems to build data products by analysists
    - Tableau
    - https://looker.com/product
    - Knime
    - etc

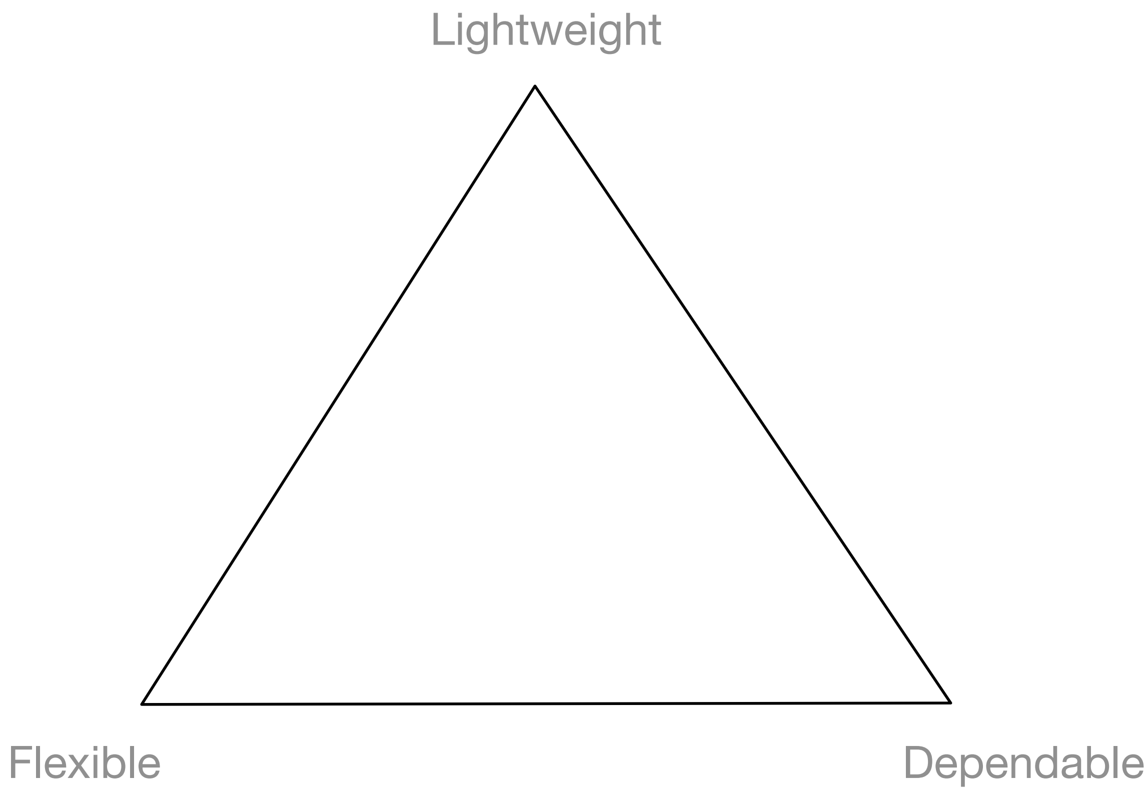


Now What?
python
pandas
learn
R
So when can we go live with the new model?
Any of you know what Gradient Boosting is?
Java
JS
php
salesforce
node
.net
Ruby
A Programmer's Best Friend
Data Scientist
Software Developers

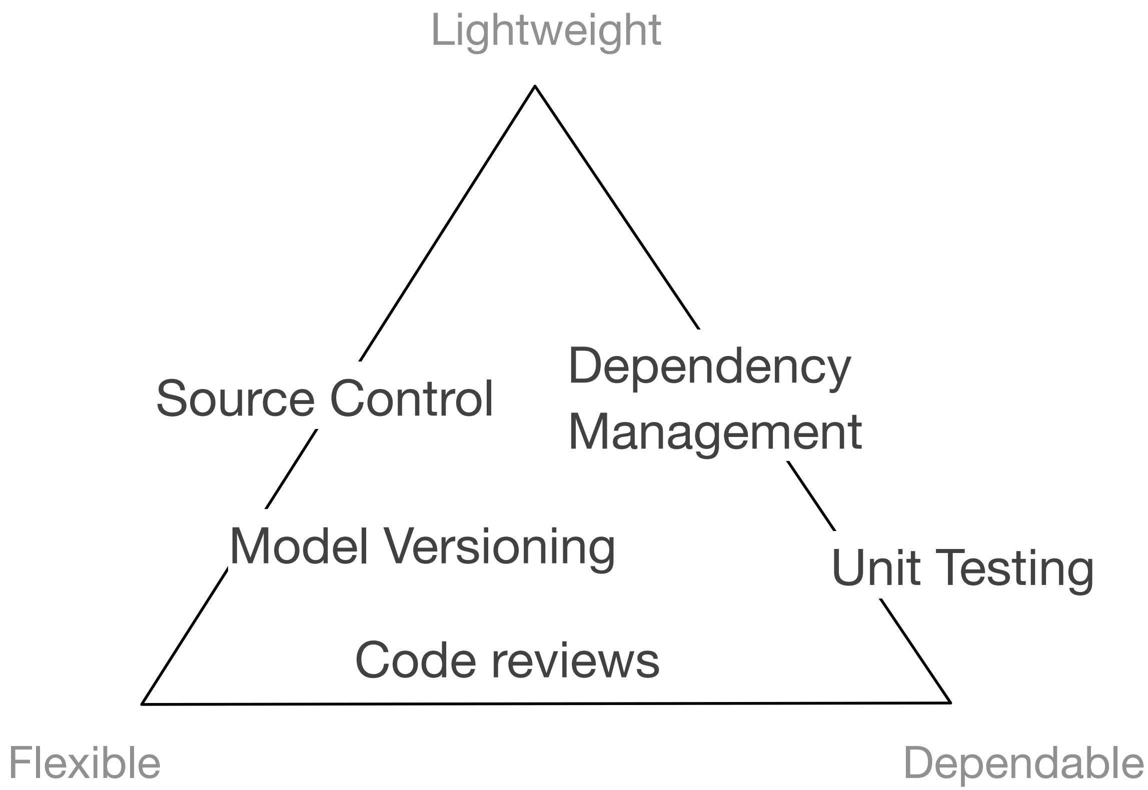
Data Roles


ad-hoc reports and charts
reports, summarize and visualize data
Data Analyst
apply statistics, machine learning
Data Scientist
building data science apps
Deployment
design and architecture
Data Engineer
prepare big data infrastructure
design, and build BI
BI Developer

Data Science Software Engineering 101



Data Science Software Engineering 101



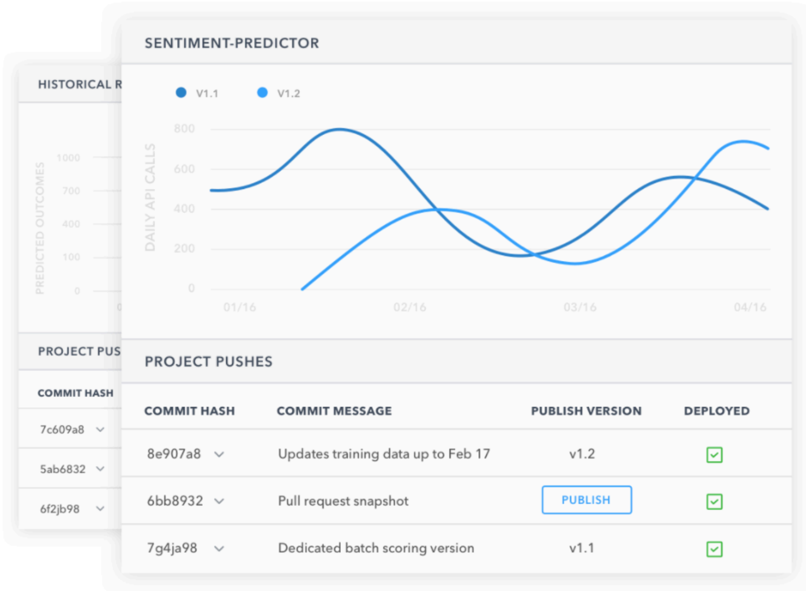
Dependency Management

- Dependency hell
- Reproducabilty
- Facilitation to on-board new people

Model Versioning

MODEL MANAGEMENT

Test multiple models at once, promote new models into production, and continuously iterate over time.



Unit Testing

- TDD
- CI
- nosetests, pytest, unittest2
- Source Control – remove output from ipynb’s before commit

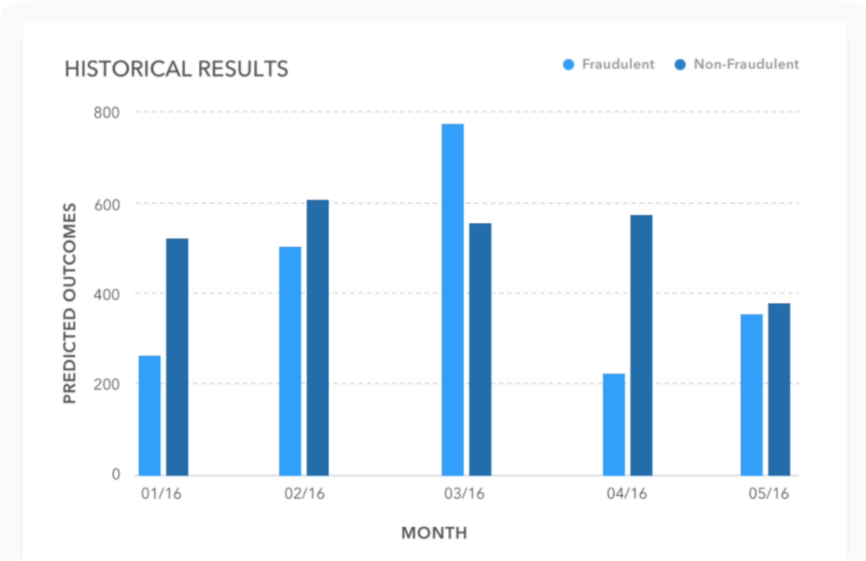
Quality

- Code reviews
- Pair programming
- Verification
- Validation

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Monitoring

- System Health Overview
- Logging



MODEL MONITORING

Measure model performance and its impact on your business with data from API calls, training, and cross validation.

Infrastructure as Code

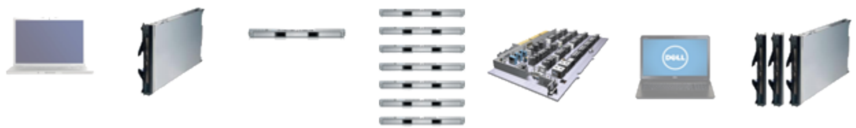
- maintainable, versionable, testable, and collaborative.
- Fabric
- CloudFormation
- https://www.terraform.io/
- Chef
- etc

Important infrastructure

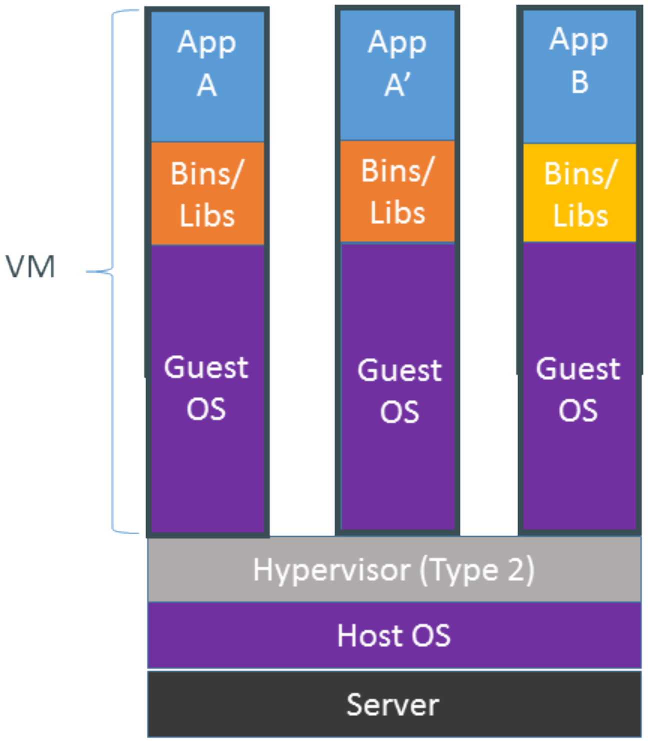
- Your machine
- Cloud
- Big Data

Docker the-matrix-from-hell

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

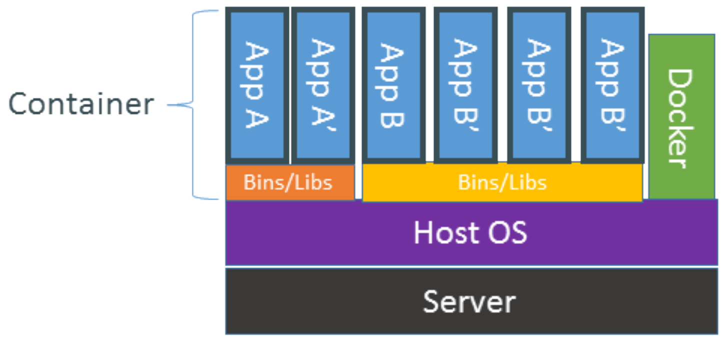


Docker vs VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries

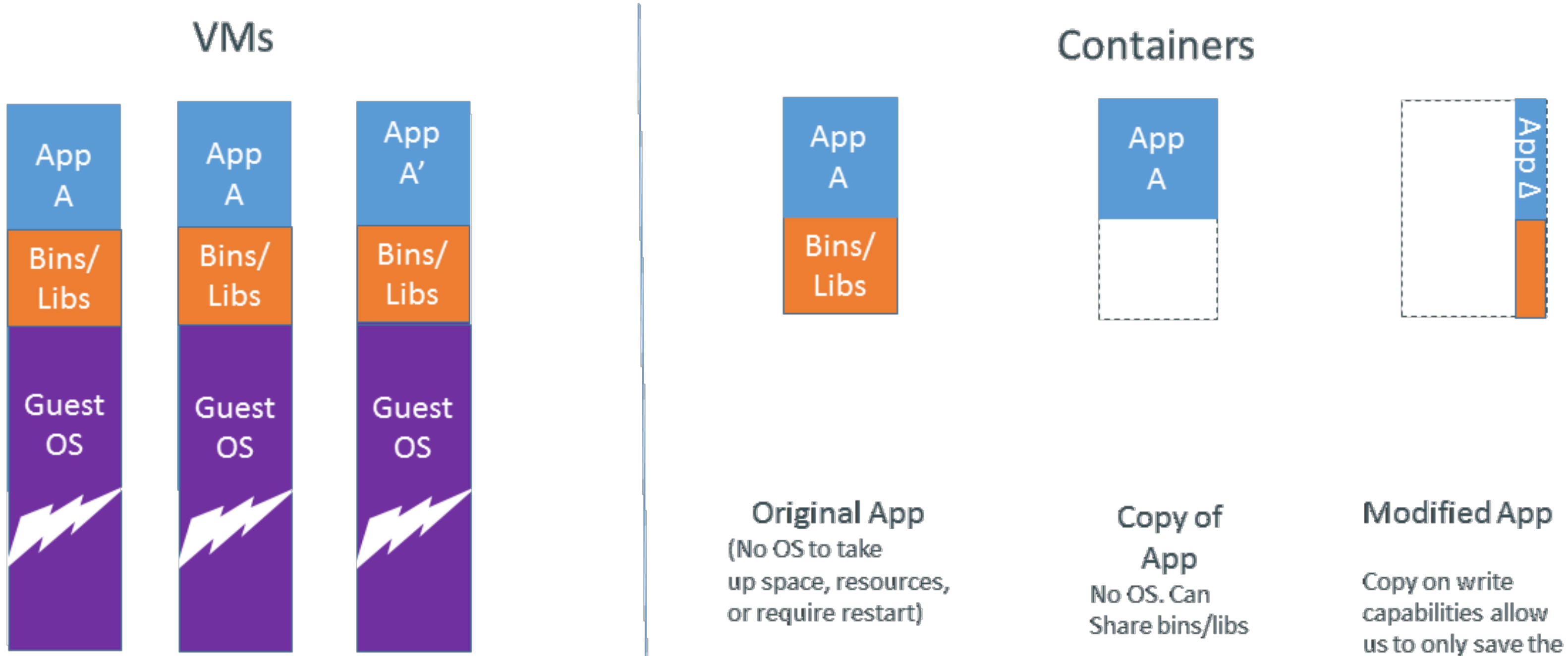
...result is significantly faster deployment, much less overhead, easier migration, faster restart



University of BRISTOL

27 April 2017

Docker - Diffs and Layers



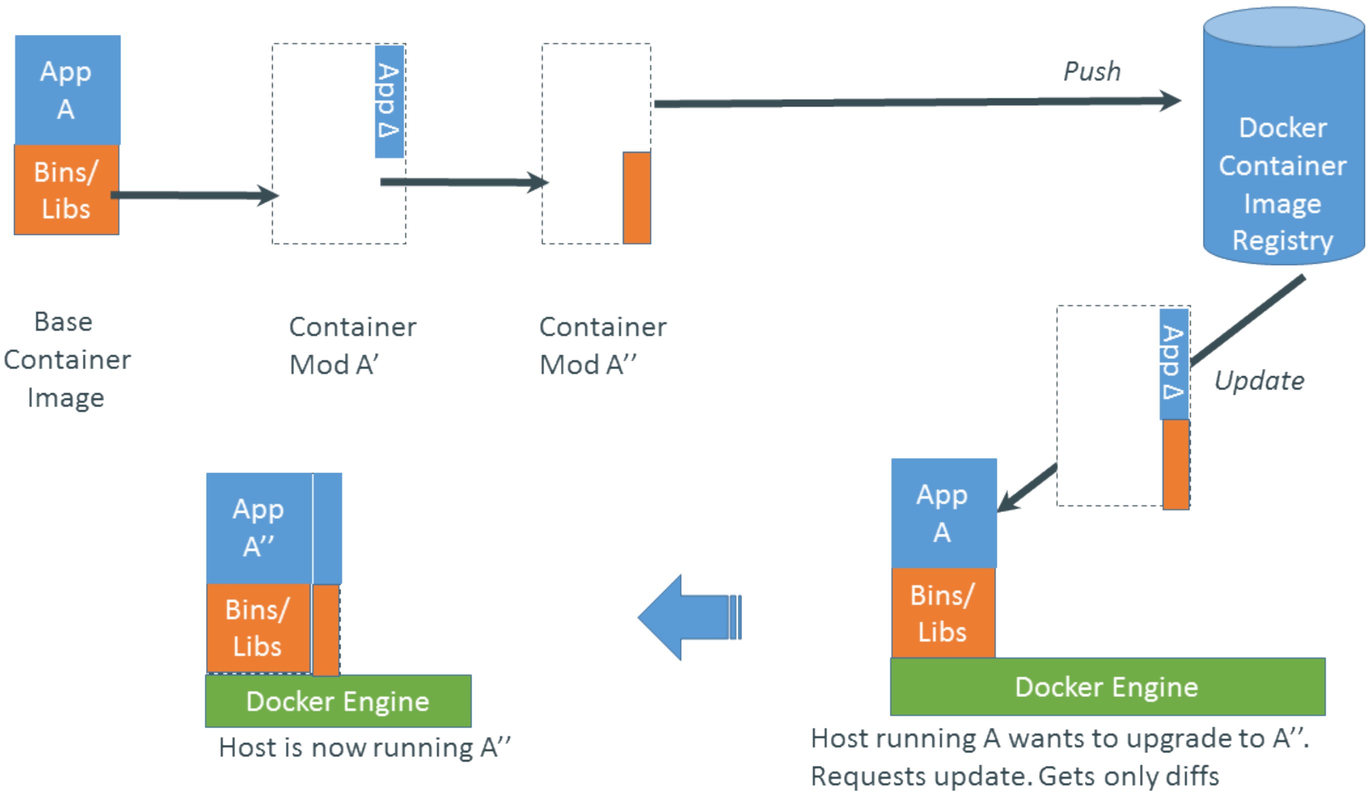
VMs
Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Original App
(No OS to take up space, resources, or require restart)

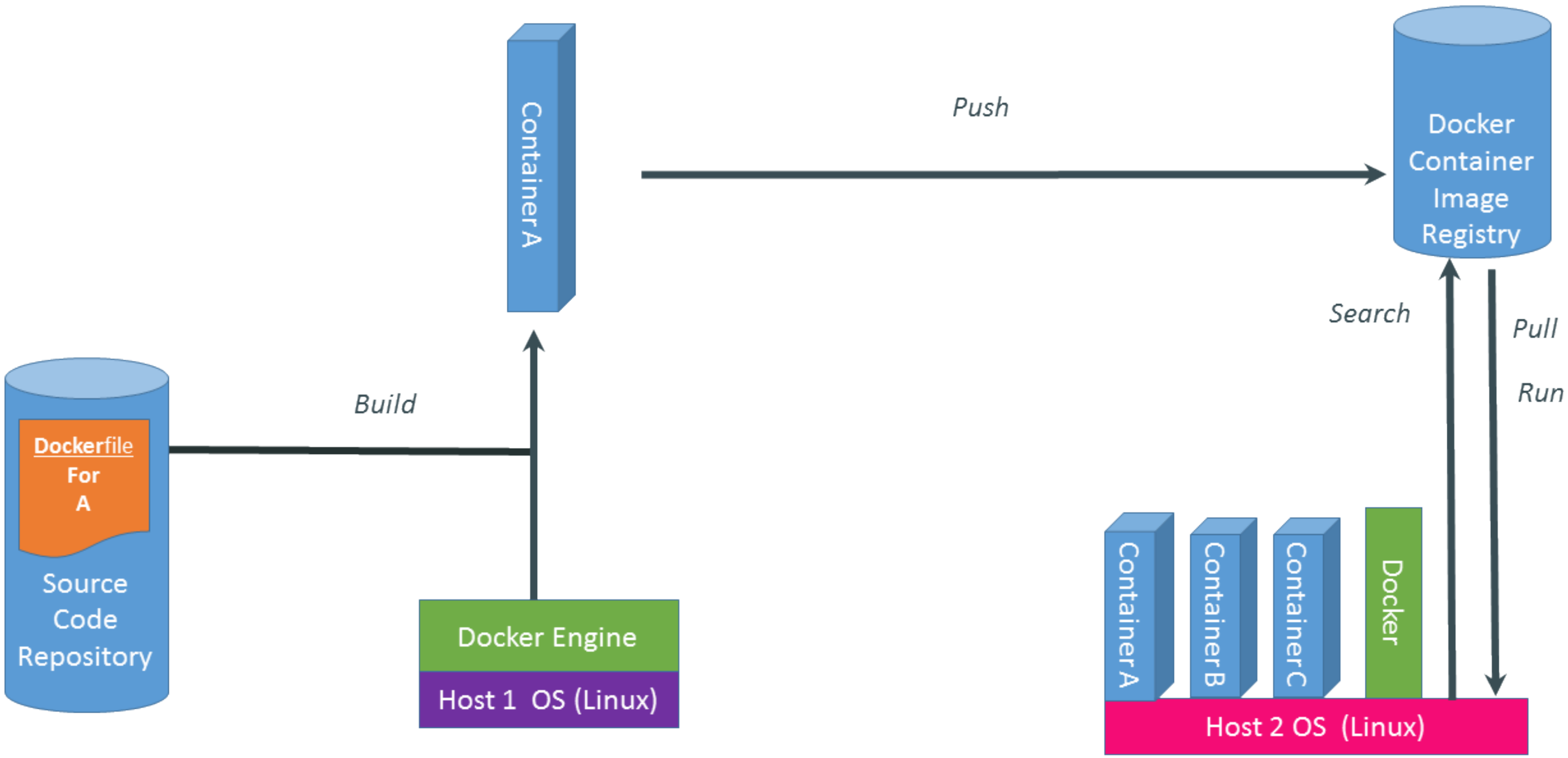
Copy of App
No OS. Can Share bins/libs

Modified App
Copy on write capabilities allow us to only save the diffs Between container A and container A’

Docker - Change Cycle



Docker - Deployment



Basic Deployment

- Ipython notebook in a container
- Deploy your jupyter notebook online
- simple notebook with widgets
- Dockerfile
- build container
- run with mount and password
- deploy remote with fabric - fabfile.org

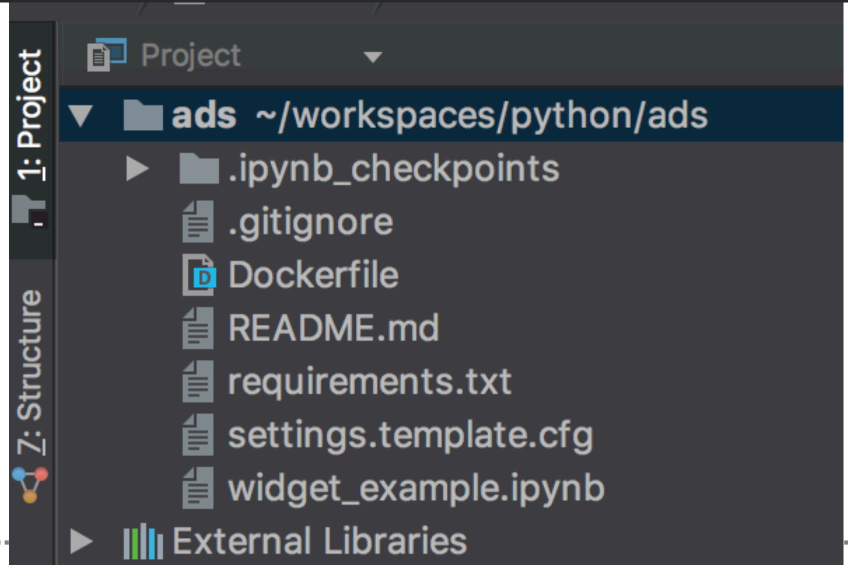
Fabric

Demo

https://github.com/dschien/ads_notebook

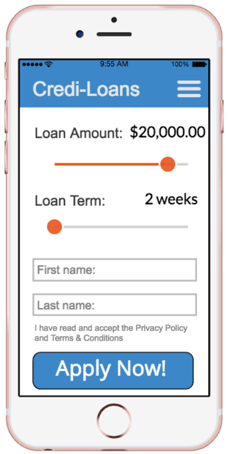
```
FROM jupyter/minimal-notebook

MAINTAINER Dan Schien <dschien@gmail.com>

COPY requirements.txt /opt/app/requirements.txt
WORKDIR /opt/app

RUN pip install -r requirements.txt


RUN pip install jupyter_dashboards
RUN jupyter dashboards quick-setup --sys-prefix

RUN jupyter nbextension enable --py --sys-prefix widgetsnbextension

USER jovyan
```

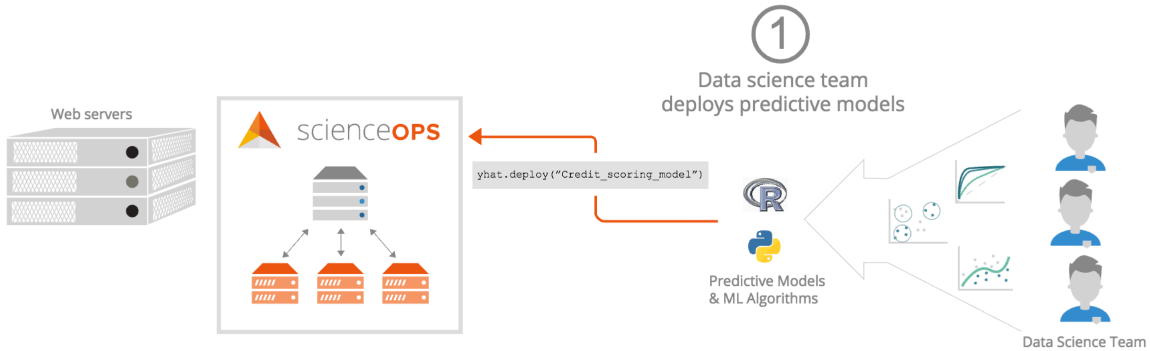
```
→ ads git:(master) ✗ docker build -t dschien/ads .
Sending build context to Docker daemon 179.2 kB
Step 1/9 : FROM jupyter/minimal-notebook
 ---> d9b21ff9ceac
Step 2/9 : MAINTAINER Dan Schien <dschien@gmail.com>
 ---> Using cache
 ---> 23087455a6fb
Step 3/9 : COPY requirements.txt /opt/app/requirements.txt
 ---> Using cache
 ---> 7db351f9a3b1
Step 4/9 : WORKDIR /opt/app
 ---> Using cache
 ---> 505a4a3adc51
Step 5/9 : RUN pip install -r requirements.txt
 ---> Using cache
 ---> 2c01e4710577
Step 6/9 : RUN pip install jupyter_dashboards
 ---> Using cache
 ---> 112ed28b3bc3
Step 7/9 : RUN jupyter dashboards quick-setup --sys-prefix
 ---> Using cache
 ---> 162d98b873ed
Step 8/9 : RUN jupyter nbextension enable --py --sys-prefix widgetsnbextension
 ---> Using cache
 ---> 79c769041a1d
Step 9/9 : USER jovyan
 ---> Using cache
 ---> 31cc2868e89e
Successfully built 31cc2868e89e
```



Further Packaging your APP

- Provide the results but not the model
- Custom App with Flask
- REST Api





Compute Developments

- CPUs don’t get faster -> since about 2005 they have stabilised around 3Gh, since then we need to parallelise
- if your datasets get very large, they no longer fit into your (or any commodity hardware) machine -> parallelise
- how can you efficiently **parallelise**
  - Many machines
  - Many cores
- at scale, machines fail -> you need to know that because your algorithms will be affected by it (or rather, you can design algorithms accordingly)

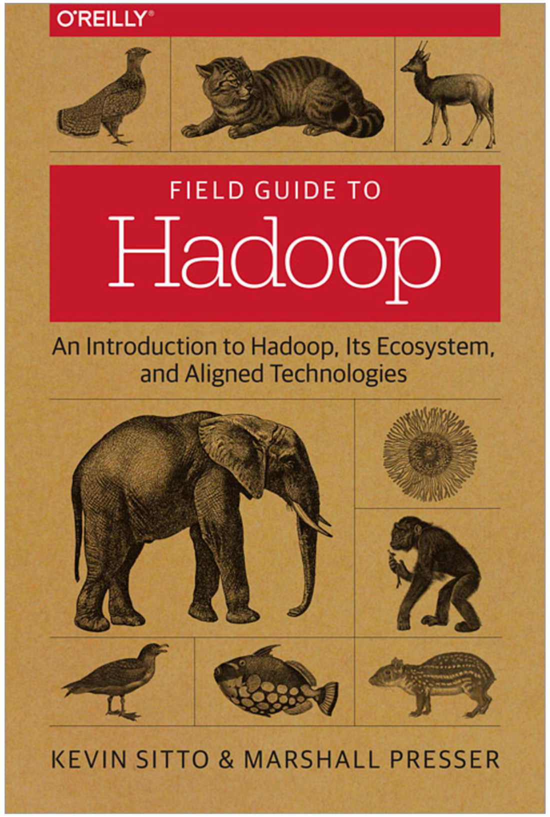
Model Optimisation

- Bottlenecks,
- stragglers
- hot spots
- Simple timing approaches, benchmarking

University of BRISTOL

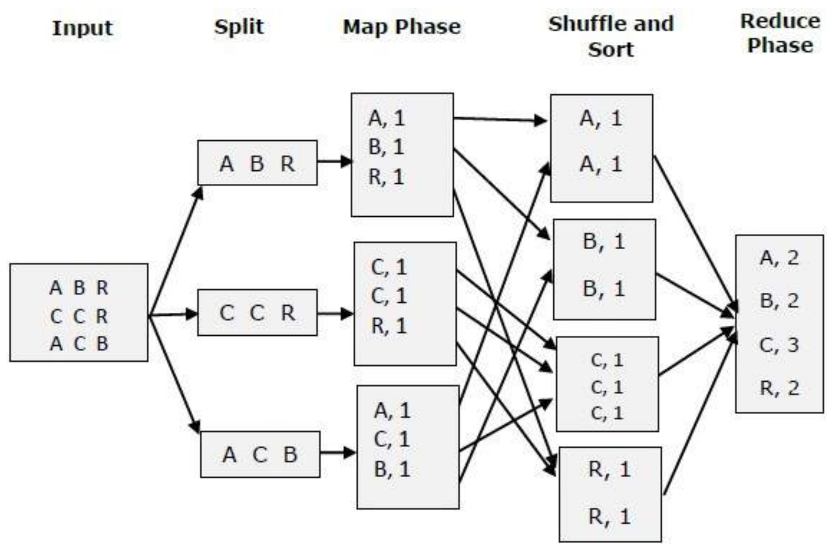
Hadoop

- **HDFS-** The default storage layer
  - splits data up
  - provides failure recovery through duplication of contents among separate nodes
- **MapReduce**
  - Googles key innovation to process HTML file indexes at web scale
- **YARN-** Responsible for cluster management and scheduling user applications
- Hosted MapReduce available
  - [Cloudera](), [Hortonworks]() and [Splice]()



Roll your own MapReduce

- 2 Python scripts, mapper, reducer
- Word count example



Command Line MapReduce Runtime ~8s (single core)

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

# --- get all lines from stdin ---
for line in sys.stdin:
    # --- remove leading and trailing whitespace---
    line = line.strip()

    # --- split the line into words ---
    words = line.split()

    # --- output tuples [word, 1] in tab-delimited format---
    for word in words:
        print '%s\t%s' % (word, "1")

```

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

# maps words to their counts
word2count = {}

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue

    try:
        word2count[word] = word2count[word] + count
    except:
        word2count[word] = count

# write the tuples to stdout
# Note: they are unsorted
for word in word2count.keys():
    print '%s\t%s' % (word, word2count[word])

```

```
→ mapreduce_wc time cat miserables.txt | ./mapper.py | sort | ./reducer.py | sort -k 2 -r -n > result.txt
cat miserables.txt  0.00s user 0.00s system 1% cpu 0.358 total
./mapper.py  0.34s user 0.02s system 96% cpu 0.370 total
sort  6.74s user 0.03s system 86% cpu 7.791 total
./reducer.py  0.77s user 0.02s system 10% cpu 7.844 total
sort -k 2 -r -n > result.txt  0.47s user 0.01s system 5% cpu 8.324 total
```

Spark

- Holds all data in memory
- creates query plans that are optimised
- Does Repeat analysis without going to disk -> speed up
- API to manipulate data frames
- Scala /Python REPL
    - makes interactive queries possible

Spark Demo

- install brew scala, brew spark
- run /usr/local/Cellar/apache-spark/2.1.0/libexec/sbin/start-master.sh
- add to env
  - if which pyspark > /dev/null; then
  - export SPARK_HOME="/usr/local/Cellar/apache-spark/2.1.0/libexec/"
  - export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/build:$PYTHONPATH
  - export PYTHONPATH=$SPARK_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH
  - fi
- start master
  - /usr/local/Cellar/apache-spark/2.1.0/libexec/sbin/start-master.sh
  - check
    - http://localhost:8080/
- start slave
  - /usr/local/Cellar/apache-spark/2.1.0/libexec/sbin/start-slave.sh spark://it033887.fen.bris.ac.uk:7077
- start ipython notebook
  - pyspark --master spark://it033887.fen.bris.ac.uk:7077

Time down to ~1s – uses all 8 cores of my MBP

3 Spark Word Count

In [1]:
```
from operator import add
```

In [3]:
```
def spark_wc(file_name):
    distFile = sc.textFile(file_name)
    counts = distFile.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(add)

    sorted_wcs = counts.sortBy(lambda word_count_pair: word_count_pair[1], ascending=False)
    sorted_wcs.saveAsTextFile("./spark_wc_result")
```

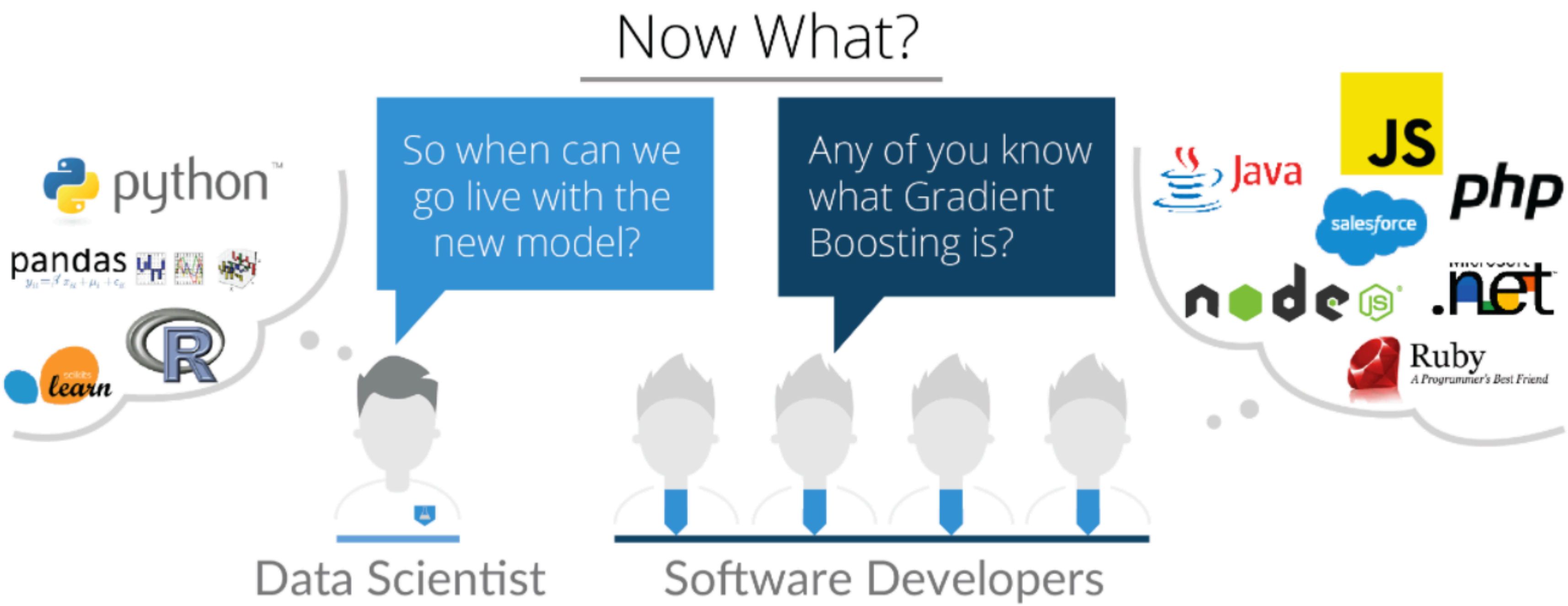
```
1.0379528559860773
```

In [ ]:
```
from timeit import default_timer as timer

start = timer()
spark_wc("mapreduce_wc/miserables.txt")
end = timer()
print(end - start)
```

Stream Processing

- Spark Streaming with micro batches - convenience of familiar paradigm, sufficient for 98% of use cases
- Storm - for real single event processing

Horizon Scanning: Predictive Model Markup Language (PMML)



Summary

- In startup environments, data scientists can contribute to **deployment** on several fronts
- knowledge of distributed computing paradigms is important for efficient **implementations**, better models
- for sharing in **trusted** environments, docker packaged notebooks work great

Thank you for your attention