SE4ML/AI, part 2: from principles to production

Claire Le Goues Michael Hilton

institute for SOFTWARE RESEARCH

Carnegie Mellon University
School of Computer Science

Administrivia

- Homework 2 reflections due today.
- Homework 3 out this evening.
- We won't track explicit lecture activities on November 3rd, and helpfully remind you that we are posting all lecture videos to Canvas.
- Gentle/motherly reminder to get your flu shots!

Learning goals

- Understand the lower-level process of ML in SE, from model building/experimentation to deployment.
- Characterize the different challenges and design goals at the different phases (experimentation vs. production).
- Describe strategies for sanity checking data, features, and models.
- Provide key tests and properties to reduce technical debt and improve reliability for both data and ML infrastructure.

Internship leads

- We have a few leads, once we have something solid, we will post them on slack

Resumes

My Simple rules about Resumes

Top Half, focus on what makes **you special**

Bottom Half, what makes you **good enough**

An aside on (remote) group work

Group work serves our learning goals

- Software engineering involves collaborating to construct beyond the complexity manageable by a single person.
- Collaboration (along with software engineering process) is a skill and set of practices that can be taught, learned, and practiced.
- Software engineering teams are typically composed of coworkers brought together by organizational considerations (rather than, say, friendship).
- Also, group learning is reciprocal, and allows students to tackle broader problems than they can alone.
- …ergo, mandatory group work in 17-313 .

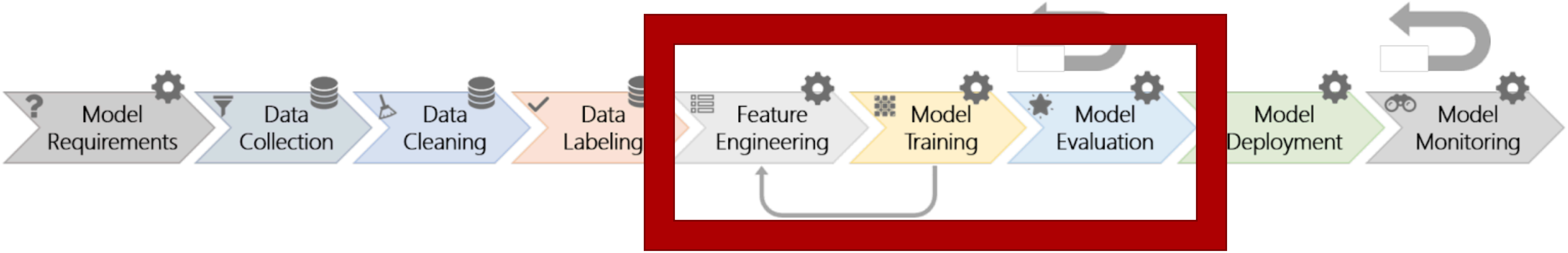
…SO THIS PANDEMIC THING SUCKS, EH?

Some tips for remote group work.

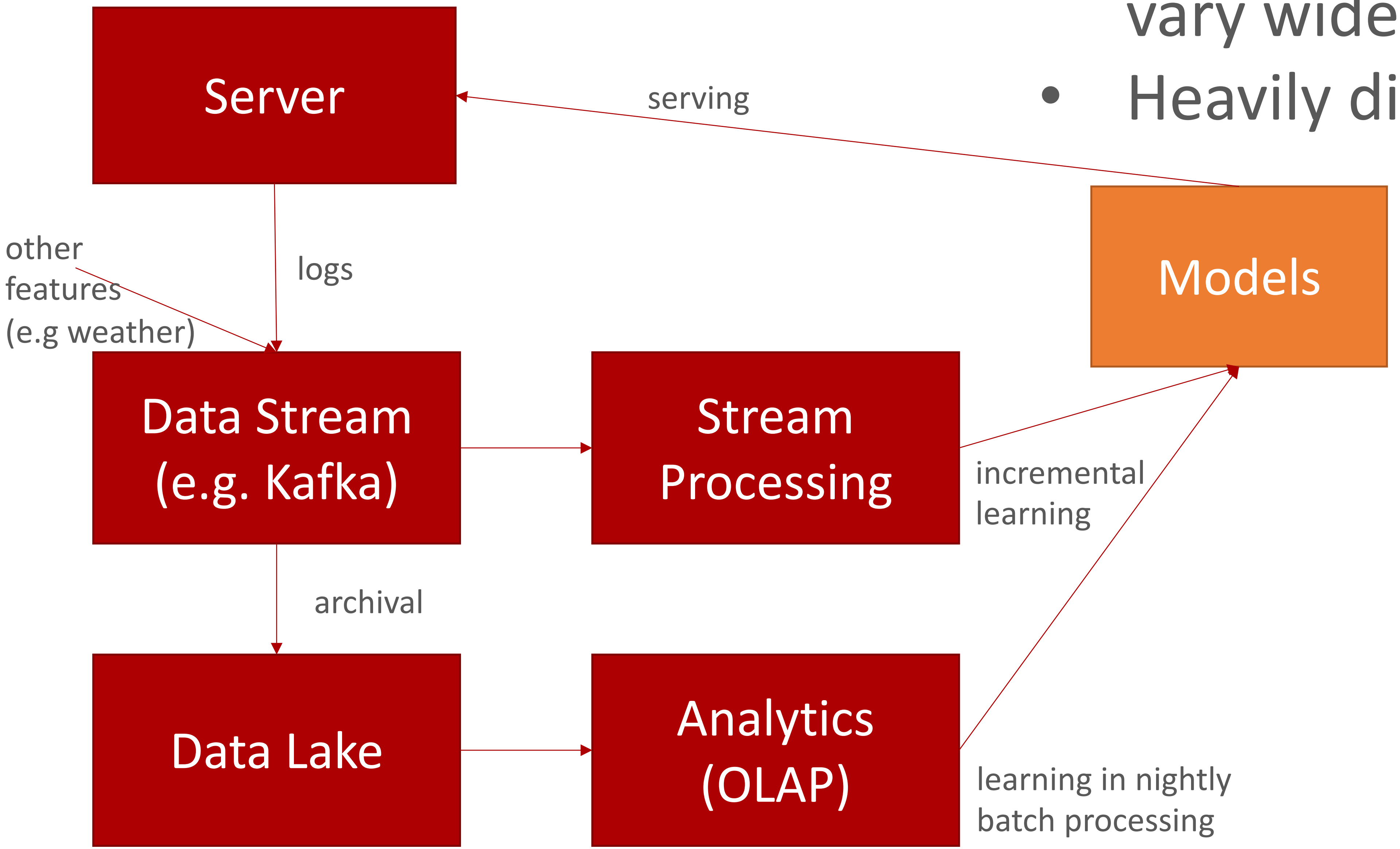
- Set synchronous work time.
  - ...we've noticed you're dividing and conquering a little *too* much.
  - Try working in a shared group meeting for an hour or so, like you would if you were in person.
  - Consider the vscode pair programming mode.
- Consider partnering up instead of fully dividing and conquering.

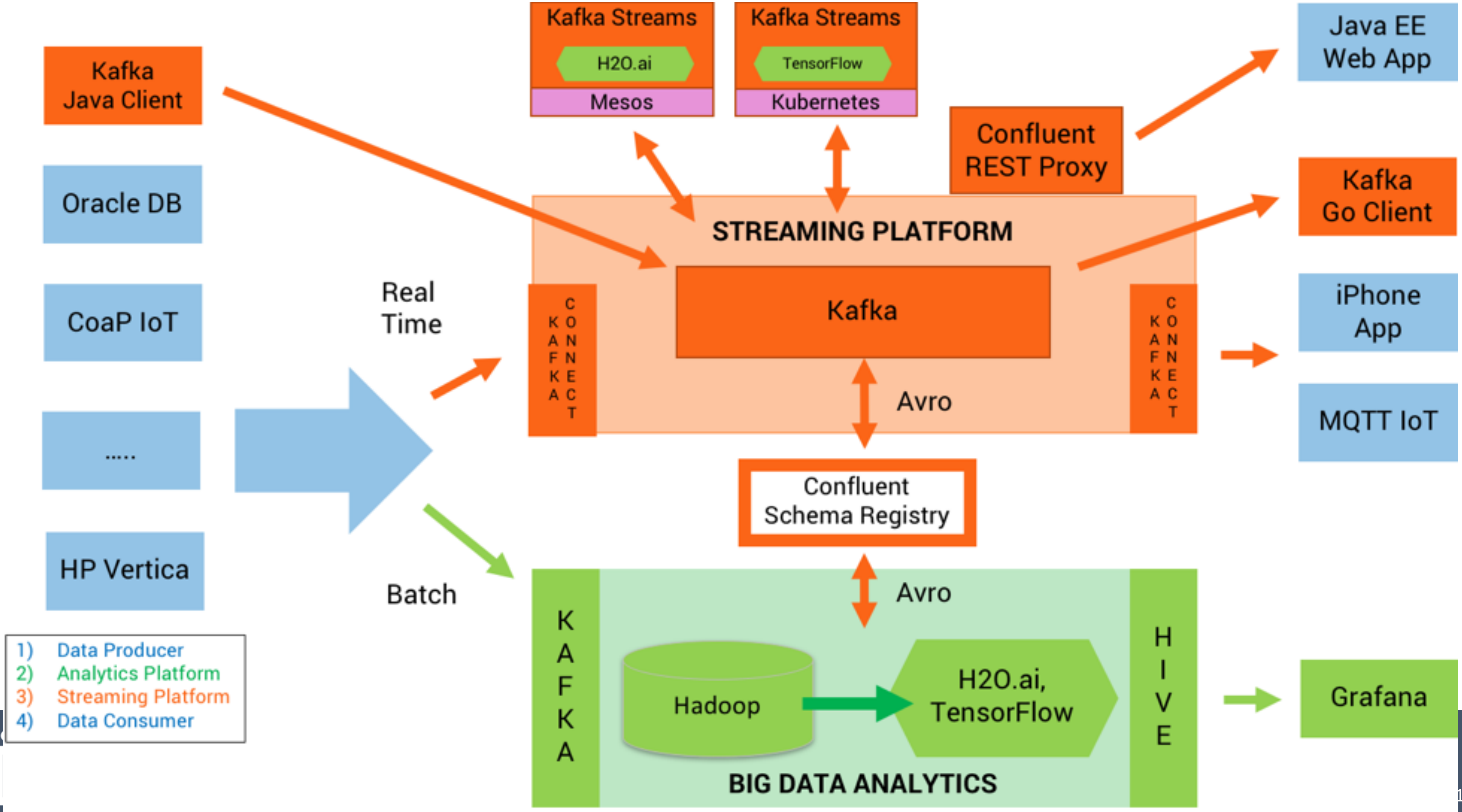
That said: you’re all doing great.
Seriously.

Microsoft's view of Software Engineering for ML



- Latency and automation vary widely
- Heavily distributed





Kafka Java Client
Oracle DB
CoaP IoT
.....
HP Vertica
1) Data Producer
2) Analytics Platform
3) Streaming Platform
4) Data Consumer
Real Time
Batch
Kafka Streams
H2O.ai
Mesos
Kafka Streams
TensorFlow
Kubernetes
Confluent REST Proxy
STREAMING PLATFORM
KAFKA CONNECT
Kafka
Avro
KAFKA CONNECT
Confluent Schema Registry
Avro
KAFKA
Hadoop
H2O.ai, TensorFlow
HIVE
BIG DATA ANALYTICS
Java EE Web App
Kafka Go Client
iPhone App
MQTT IoT
Grafana

How do we get from playing around with some data on our machine to deployment?

- Goal today: get a little more concrete about activities involved.
- Address specific concerns and activities that guide design choices as you move to deployment.
  - This is secretly an architecture case study that ultimately motivates next week’s material, but there are ML-specific concerns of note!

Demo part 1: Titanic

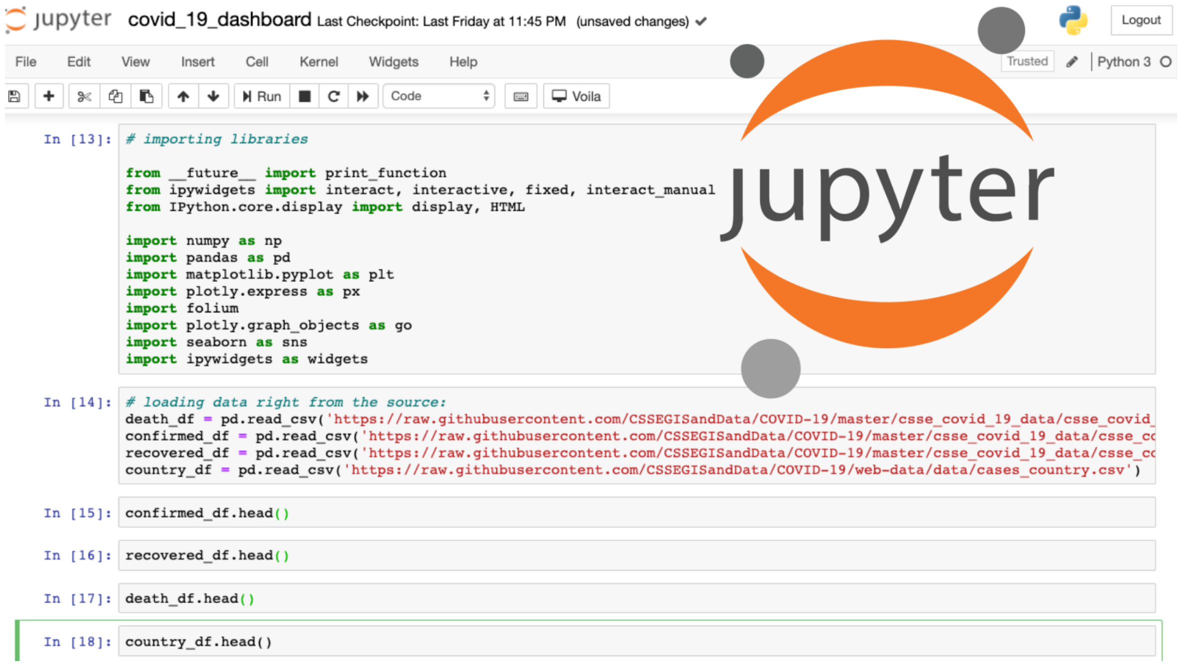
Quick activity to wake you up!

- Go to: https://www.kaggle.com/c/titanic
- Click on "Data"
  - train.csv contains a training dataset.
  - Data Description gives a bit more info.
  - Ignore gender_submission.csv and test.csv
- Read over "Feature Engineering", linked from top

- Post to Slack:
  - Your Andrew IDs, and answers to the following:
    - An observation about the data that you think is interesting.
    - If you were going to predict who survived based on the data, which feature(s) do you guess would be most predictive?
    - Do you think any of the computed features in Feature Engineering would help? Why or why not?

Demo part 1: Scikit-learn + Titanic

So what do we have?



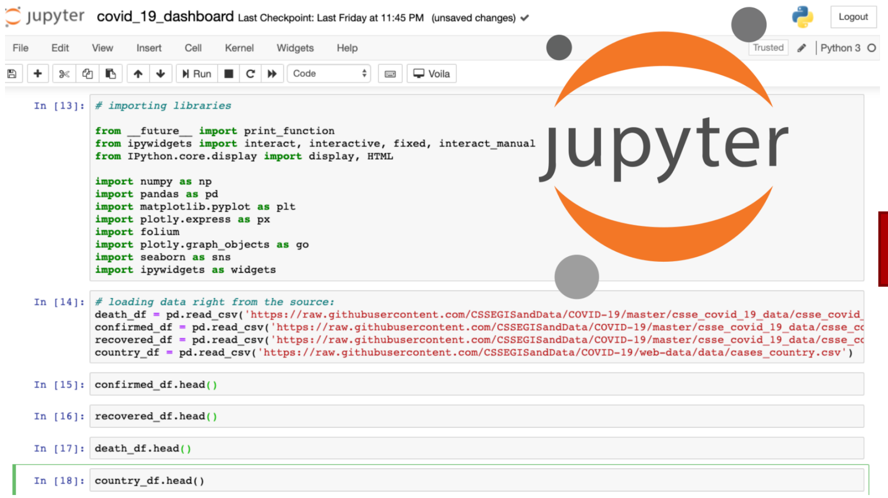
…A model that can predict whether someone was likely to have survived on the Titanic…

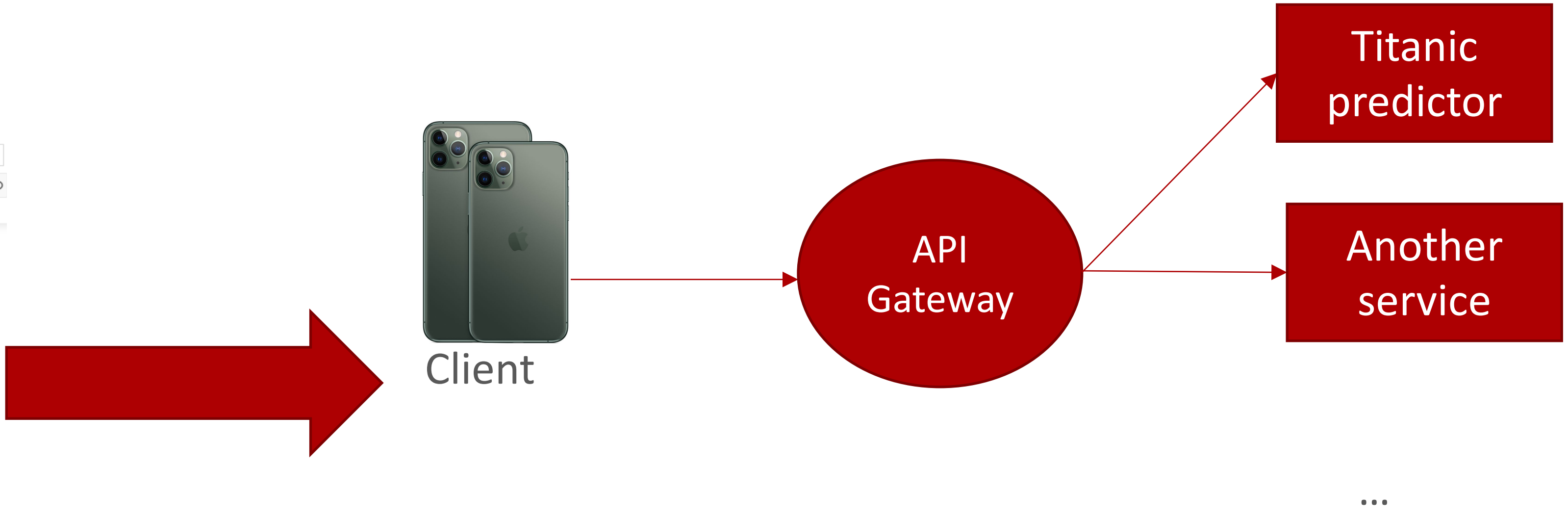
Gratuitous videogame flavortext!

As a Titanic enthusiast I want my avatar’s likelihood of surviving the Titanic-in-space disaster to be influence by historical events…to…have…fun?

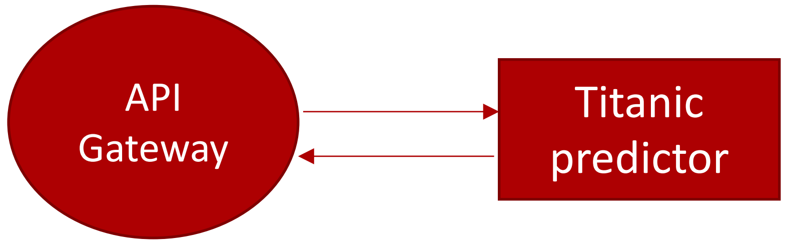
In actuality…

- …we'll restrict attention to a simple deployment to start:





Goal: abstract the model!



- ML, in the form of trained models, is typically packaged up into containers with specific associated API endpoints.
  - Wide variety of ways to interact, including retraining, load balancing, etc.
- Consider a simple predictor based on a static model that simply produces prediction based on features provided in a request.
  - Frameworks exist for this! In HW4, we'll use Flask, which supports loading saved models and passing requests using json, HTTP POST requests, or calls to a RESTFul API.

Quick Flask example that we'll see again.

```
from flask import Flask, jsonify
from sklearn.externals import joblib
import pandas as pd
```

```
app.route('/predict', methods=['POST'])
ef predict():
     json_ = request.json
     query_df = pd.DataFrame(json_)
     query = pd.get_dummies(query_df)
     prediction = clf.predict(query)
     return jsonify({'prediction': list(prediction)})
```

```
if __name__ == '__main__':
     clf = joblib.load('model.pkl')
     app.run(port=8080)
```

Going from a notebook to production?

- Notebook/research: iterative experiments of machine learning workloads usually work off a small data set (e.g., spreadsheets, notebooks).
- Production environment has different characteristics:
  - Data sources vary.
  - Must prepare and denormalize the data to create features.
  - Models must be updated periodically.
  - Trained model needs to be served against real-time requests.
  - Once deployed, must be monitored.
- We are going to ignore:
  - The automation of the overall pipeline.
  - Issues of scaling and incumbent quality attributes.

A roadmap for production readiness

- “ML reliability involves a host of issues not found in small toy examples or even large offline experiments, which lead to surprisingly large amounts of technical debt.” – Breck et al.
- Today’s focus: *processes/activities and API design considerations for moving from experimentation to deployment.*

First, consider some sanity checks

- "Machine learning systems differ from traditional software-based systems in that the behavior of ML systems is not specified directly in code but is learned from data."
- This isn't a data science class, but it's still good to be able to interrogate the black boxes you're building.
- Sanity check 1: how does your model compare to (1) a previous model, or (2) a naïve baseline?
    - Quick Slack question: what's a good naïve baseline, here?
- Sanity check 2: let's see another cool tool.

Demo part 2: LIME

"The ML Test Score: A rubric for ML production readiness and technical debt"

- **Tests for Features and Data**
- Tests for Model Development
  - Most of the principles in the literature are tied directly to QA, so we'll return to this in later lectures.
- **Tests for ML Infrastructure**
- Monitoring for ML
  - Check for stability, consistency between testing and serving, dependency changes are flagged/considered, staleness, slow-leak regressions in speed, latency, etc, or prediction quality.

Tests for features and data (ML-specific!)

- Data 1: capture feature expectations in a schema. Sometimes can be used for automatic checking, later (e.g., an adult human is between 1-10 feet tall!)
  - How? Calculate statistics from training data, adjust as appropriate based on domain knowledge, write down expectations and compare to data to avoid bias. Tools can help!
- Note that every feature has a cost! So, ensure:
  - Data 2: all features are beneficial. Data exploration, statistics! Correlations, leave-one-out comparisons
  - Data 3: No feature costs too much. Consider inference and computation latency, RAM usage, upstream data dependencies, instability.

Tests for features and data (ML-specific!)

- Data 4: model should adhere to meta-level requirements, like "cannot rely on forbidden features like age or race."
  - Sometimes it's valuable to experiment with potential features in development and experimentation!
  - Enforce these requirements programmatically!
- Data 5: implement appropriate privacy controls.
  - Do this by including enough time in your budget during new feature development depending on sensitive data to allow for proper handling, and test user-requested data deletion.
- Maintainability:
  - Data 6: new features can be added quickly.
  - Data 7: all input feature code is tested.