Reactive Machine Learning

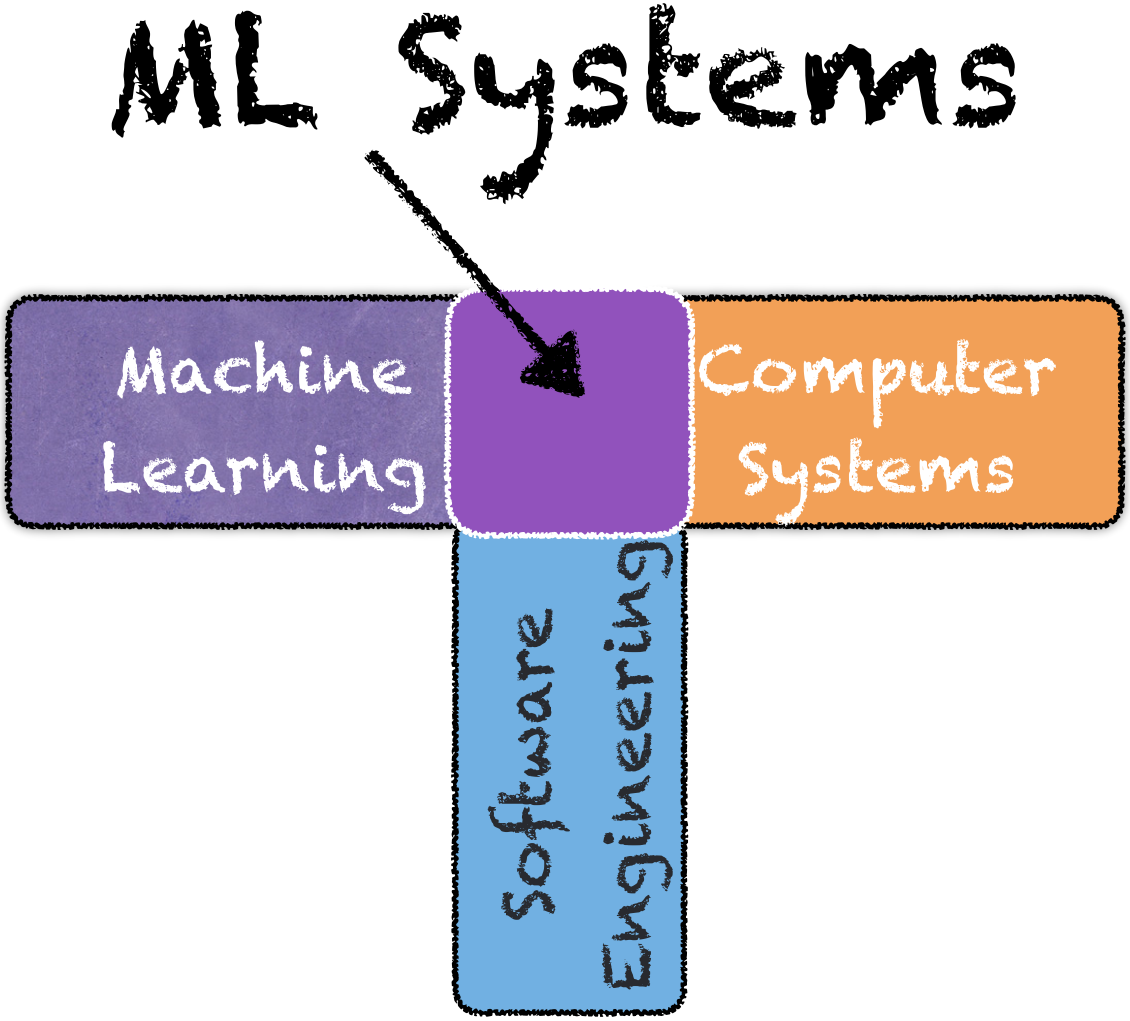
Pooyan Jamshidi
USC

Learning Goals

- Understand how to build a system that can put the power of machine learning to use.
- Understand how to incorporate ML-based components into a larger system.
- Understand the principles that govern these systems, both as software and as predictive systems.

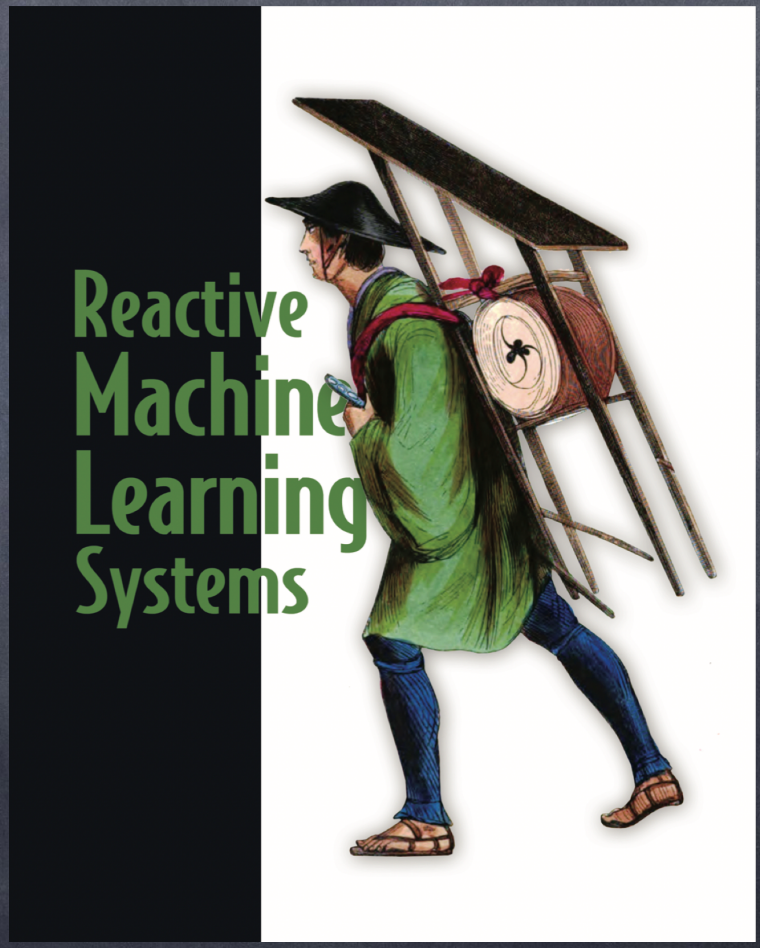
Learning Goals

- Understand challenges of building ML systems.
- Understand strategies of making ML systems reactive.


ML Systems
Machine Learning
Computer Systems
Software Engineering

Acknowledgement

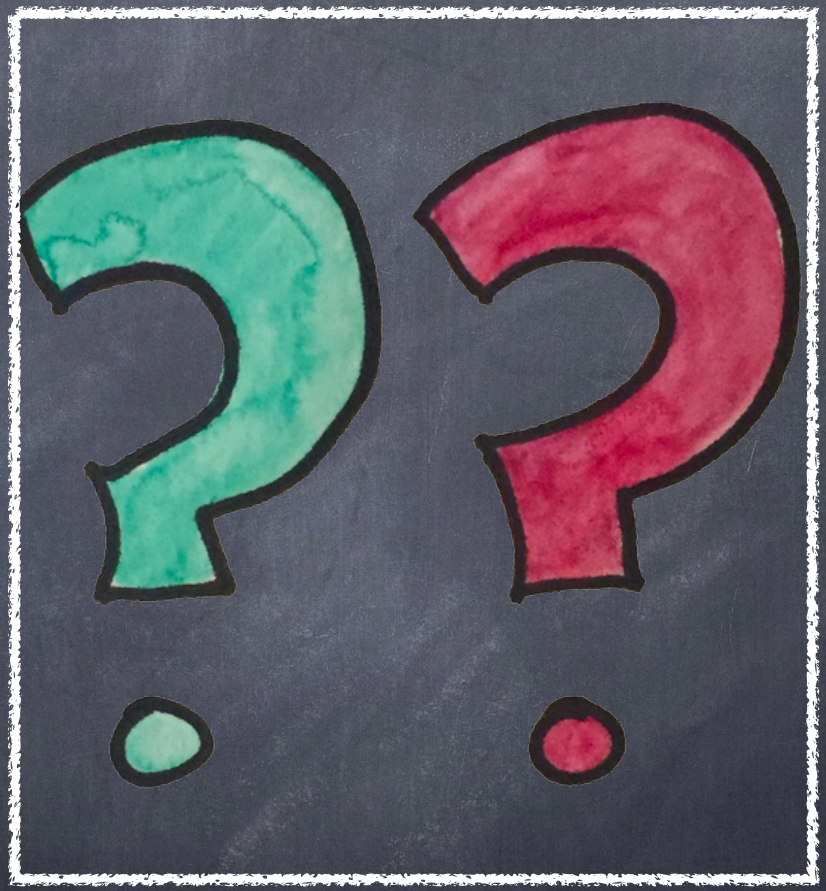
- I borrowed materials from:
  - Jeff Smith's book ->
  - Spark Documents
  - MIT course Intro to Deep Learning
  - TF document



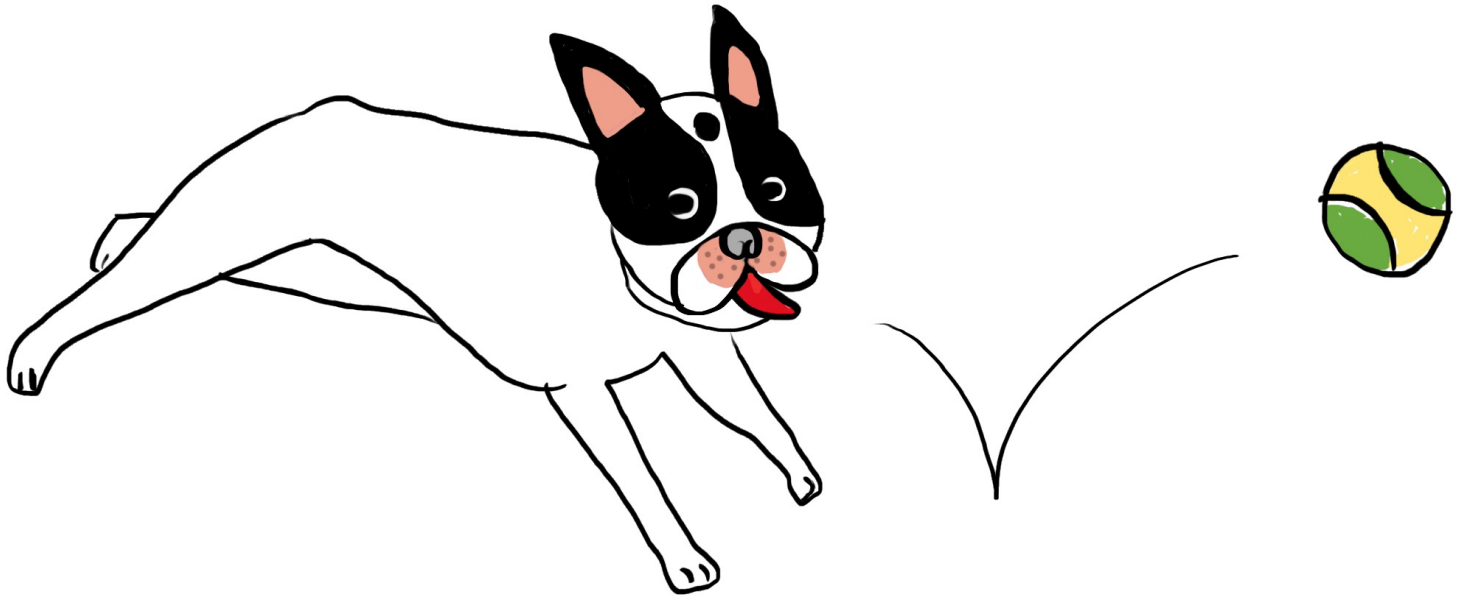
Reactive ML

- Reactive ML = using strategies for building Reactive Systems + to solve unique challenges of ML systems

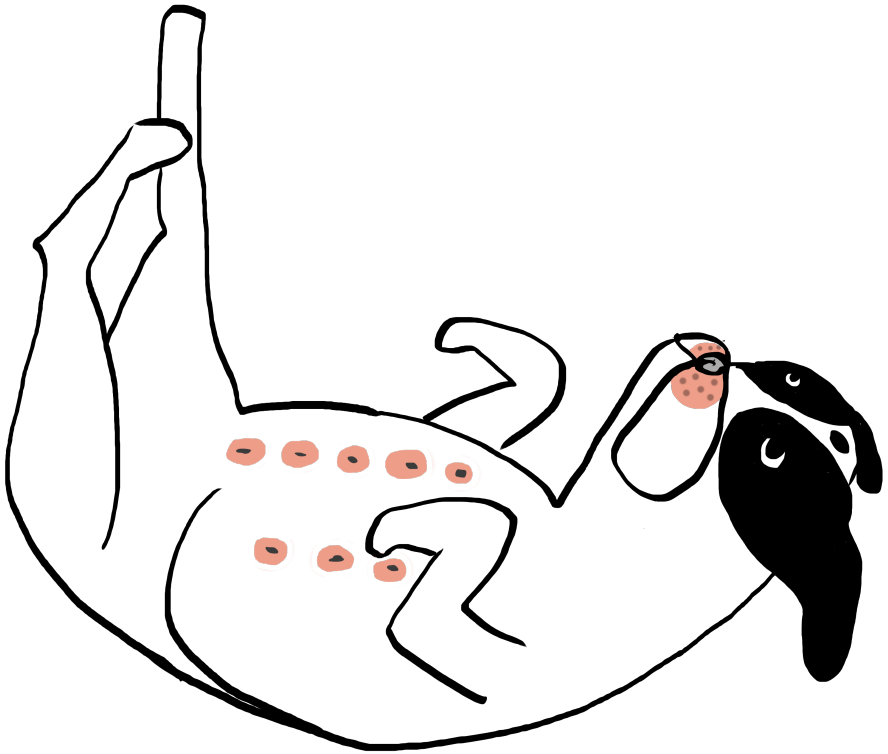
But wait, how a reactive systems looks like?



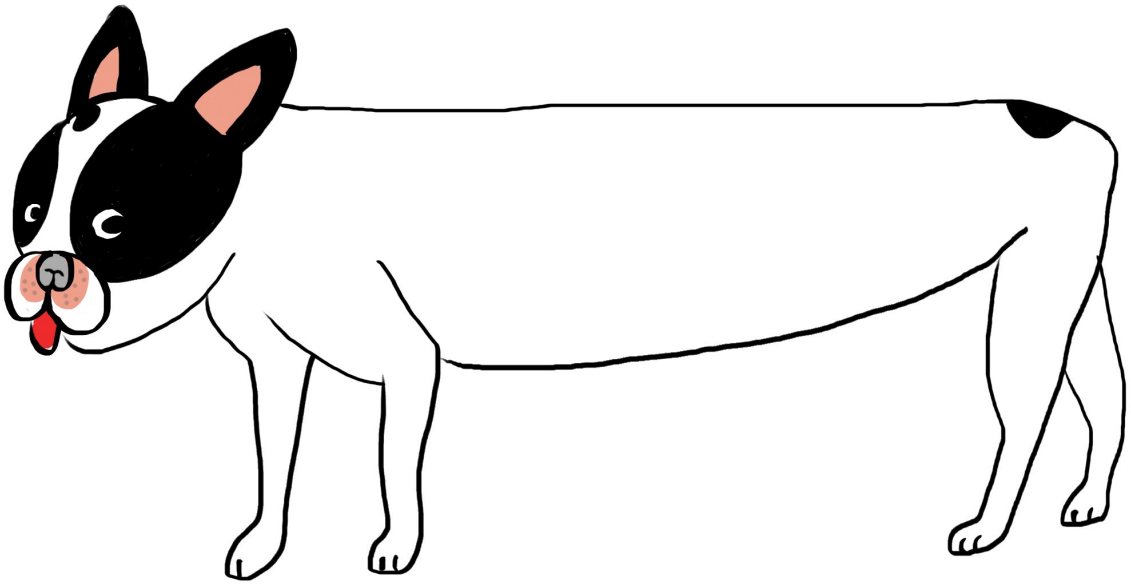
Responsive



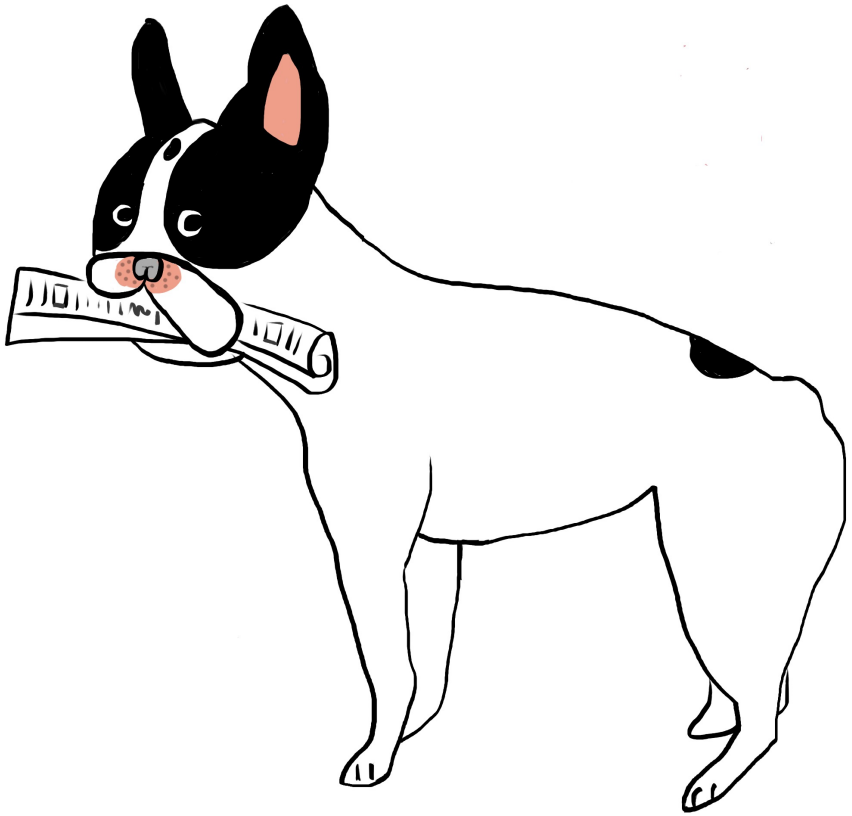
Resilient



Elastic



Message-driven



Case study

- A startup that tries to build a machine learning system from the ground up and finds it very, very hard!

Sniffable

- Sniffable is the Instagram for dogs.
- Dog owners post pictures of their dogs,
- and other dog owners (sniffers!) like, share, and comment on those pictures.

A New Feature for Sniffable

- Sniffers want to promote their specific dog
- Sniffers want their dogs to become a celebrity!
- So, they want a new tool to make their pupdates, more viral.

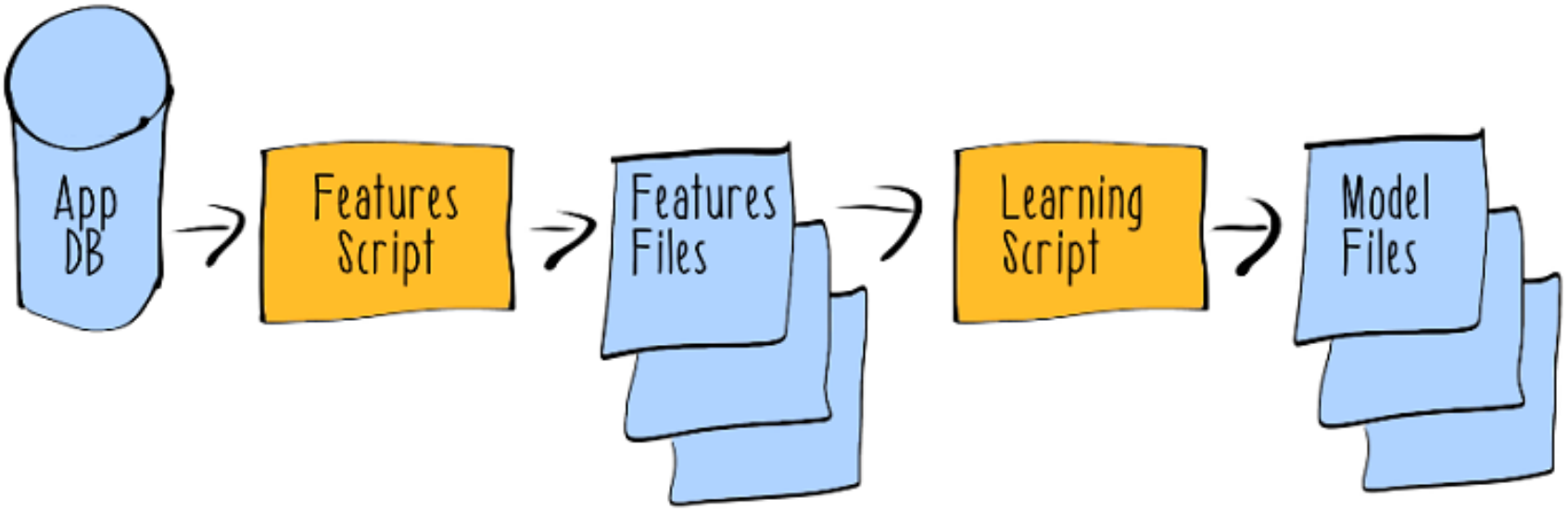
Pooch Predictor

- A new feature to figure out which picture would get the biggest response on Sniffable
- A new feature to predict the number of likes a given pupdate might get, based on the hashtags used.

Why did the startup come up with Pooch Predictor?

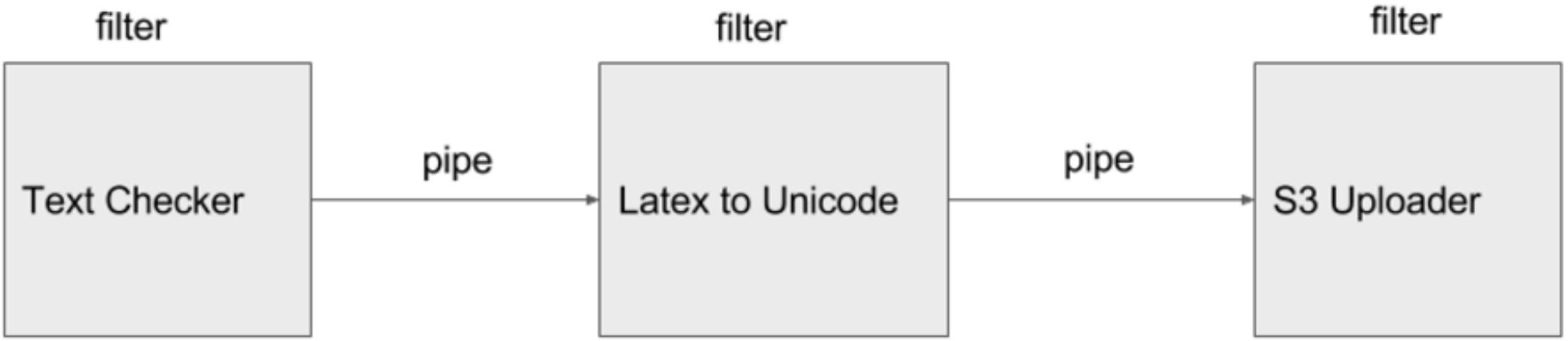
- It would engage the dog owners,
- help them create viral content,
- and grow the Sniffable network as a whole

Pooch Predictor 1.0 Architecture

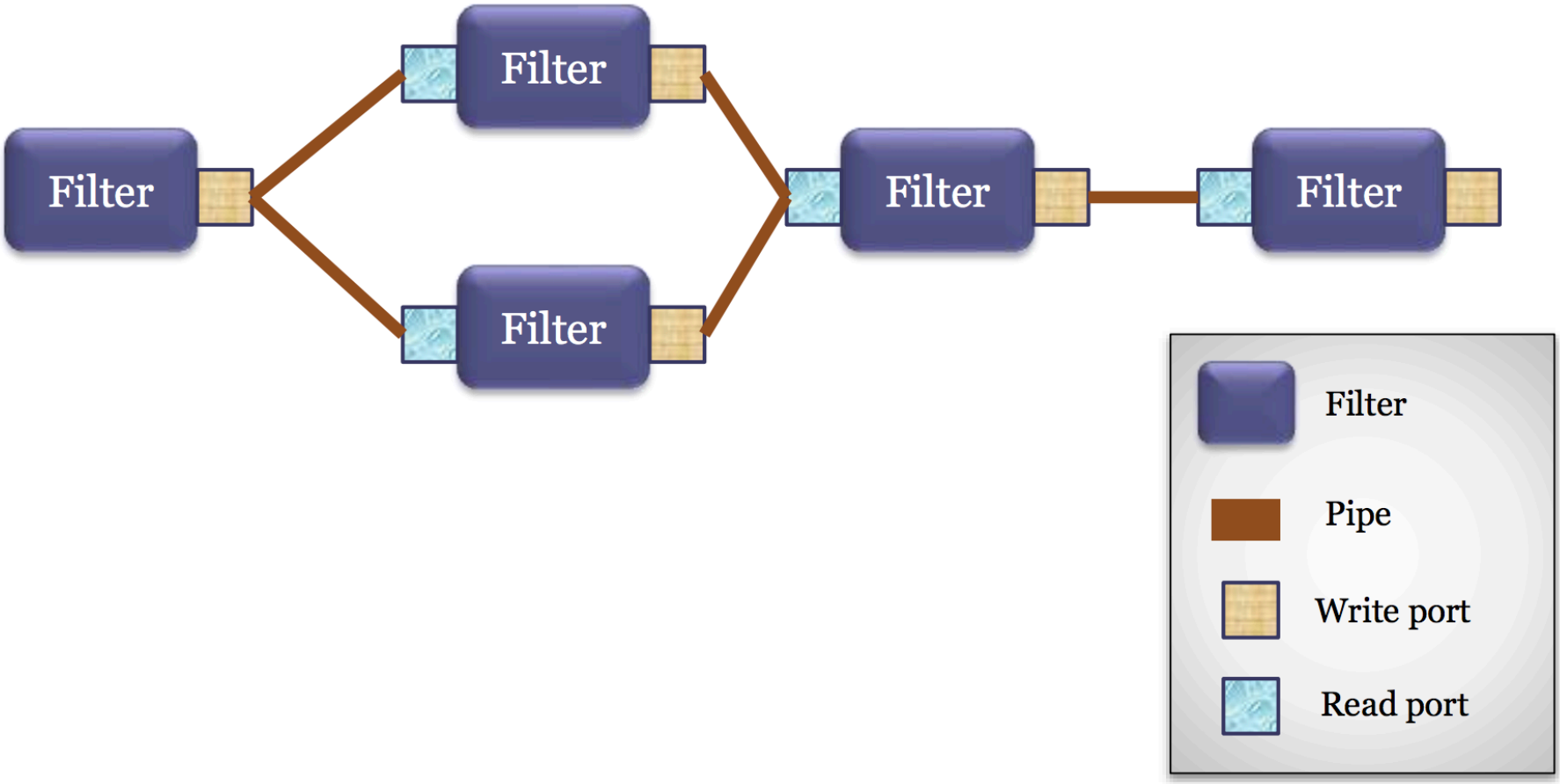


What kind of architectural style is this?

How does pipes and filters style look like?



Pipes and filters



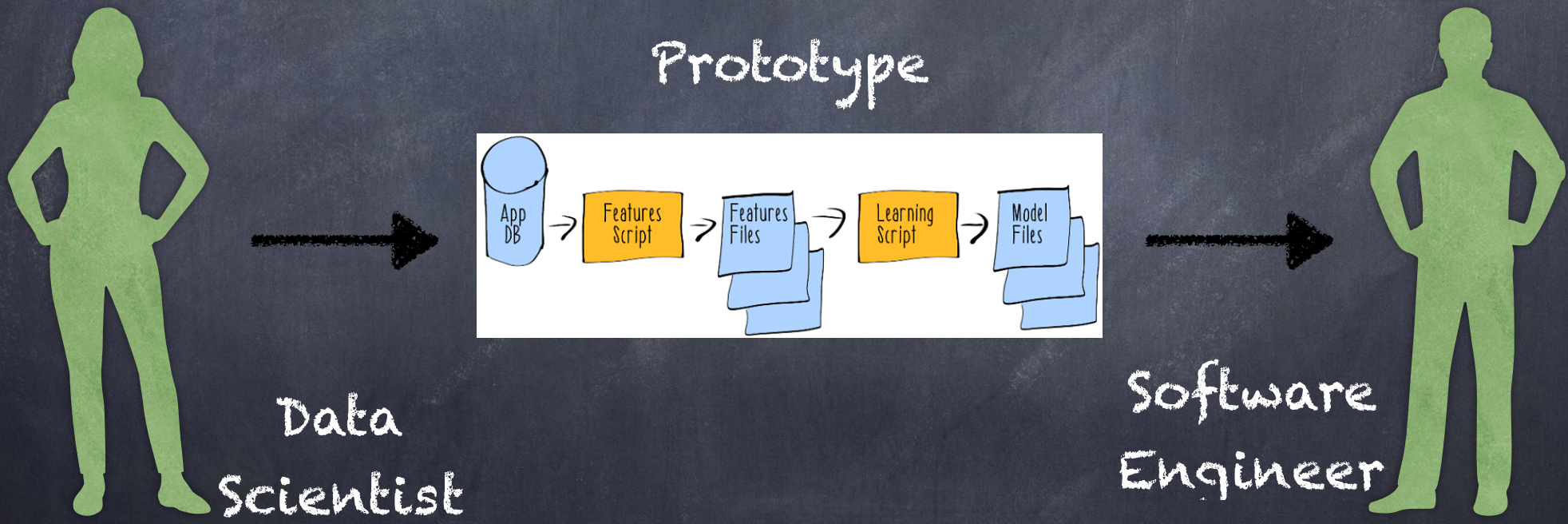
Pipes and filters

- Filter: component that transforms data
  - Filters can be executed concurrently
- Pipe: Takes output data from one filter to the input of another filter
  - Properties: buffer size, data format, interaction protocol

Benefits of pipes and filters?

- Each component is decoupled from one another and can be maintained independently.
- Each component can also be tested in isolation.
- You can re-use components to create different chains.

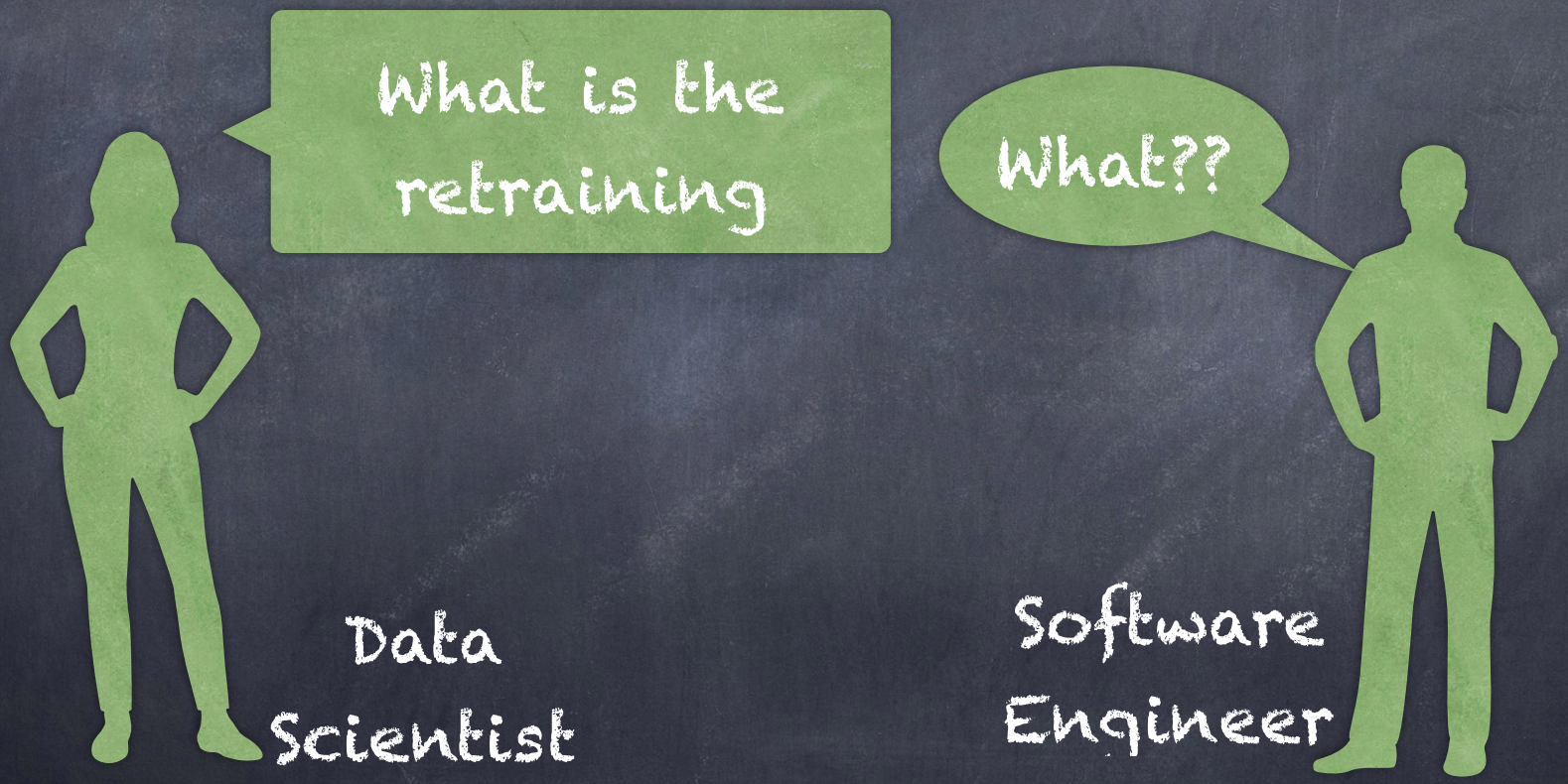
What happened at the organizational level



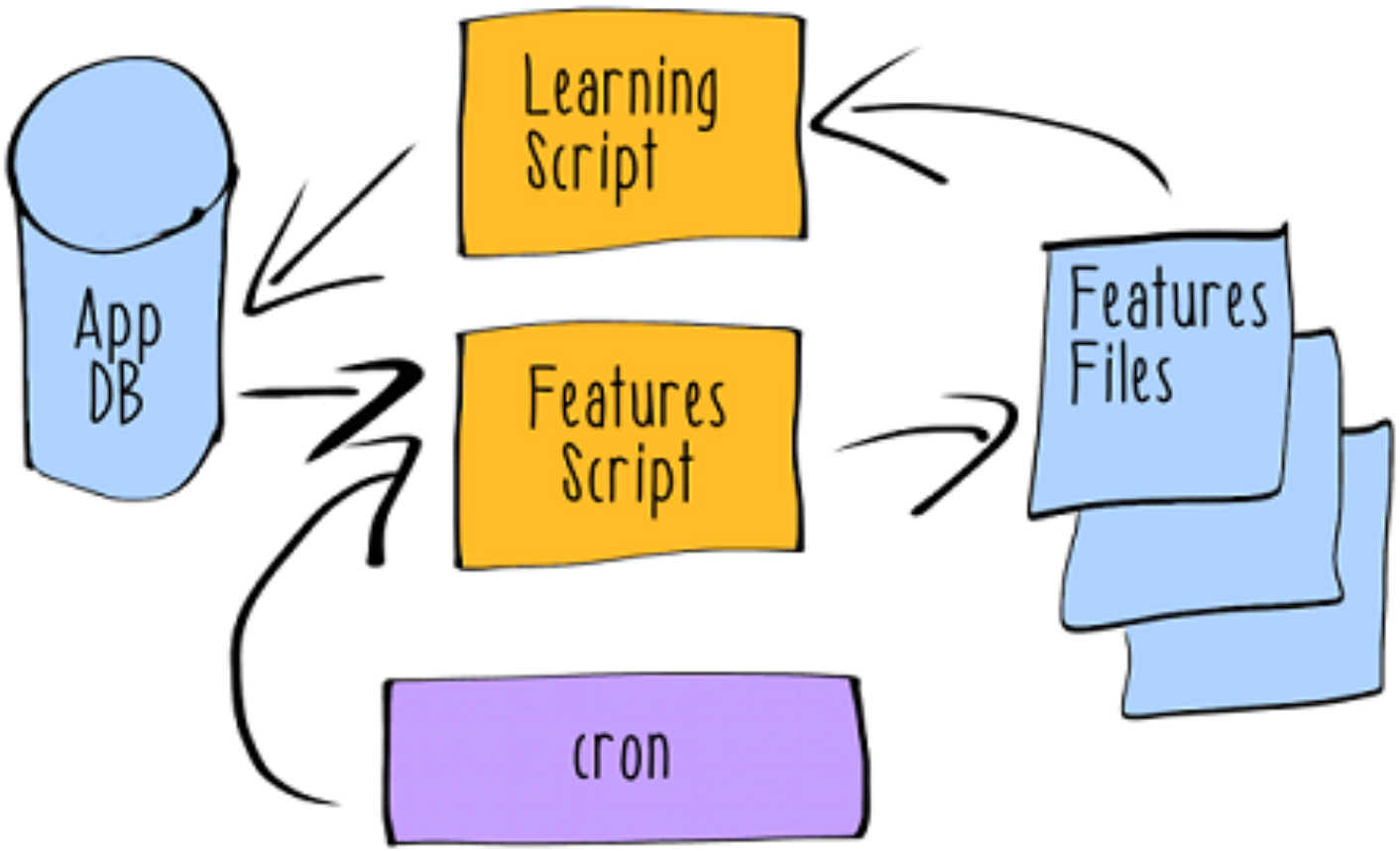
Something seems wrong!

- Issue: Predictions weren't changing much over a period of time despite adding more data to the system

Misunderstanding between stakeholders



Pooch Predictor 1.1 Architecture



System issues appeared

- Pooch Predictor wasn't very reliable
- Change in the database -> the queries would fail
- High load on the server -> the modeling job would fail
- More and more as the size of the social network increased

Failures were hard to detect

- Failures were hard to detect without building up more sophisticated monitoring infrastructure.
- There wasn't much that could be done other than restarting the job.

Modeling issues

- Data scientist realized that some of the features weren't being correctly extracted from the raw data.
- It was also really hard to understand how a change to the features would impact modeling performance.

A major issue happened

- For a couple of weeks, their interaction rates steadily trended down.
- For users outside of the US, Pooch Predictor would always predict a negative number of likes!!
- It was a problem with the modeling system's location-based features.
- The data scientist and engineer paired to fix the issue

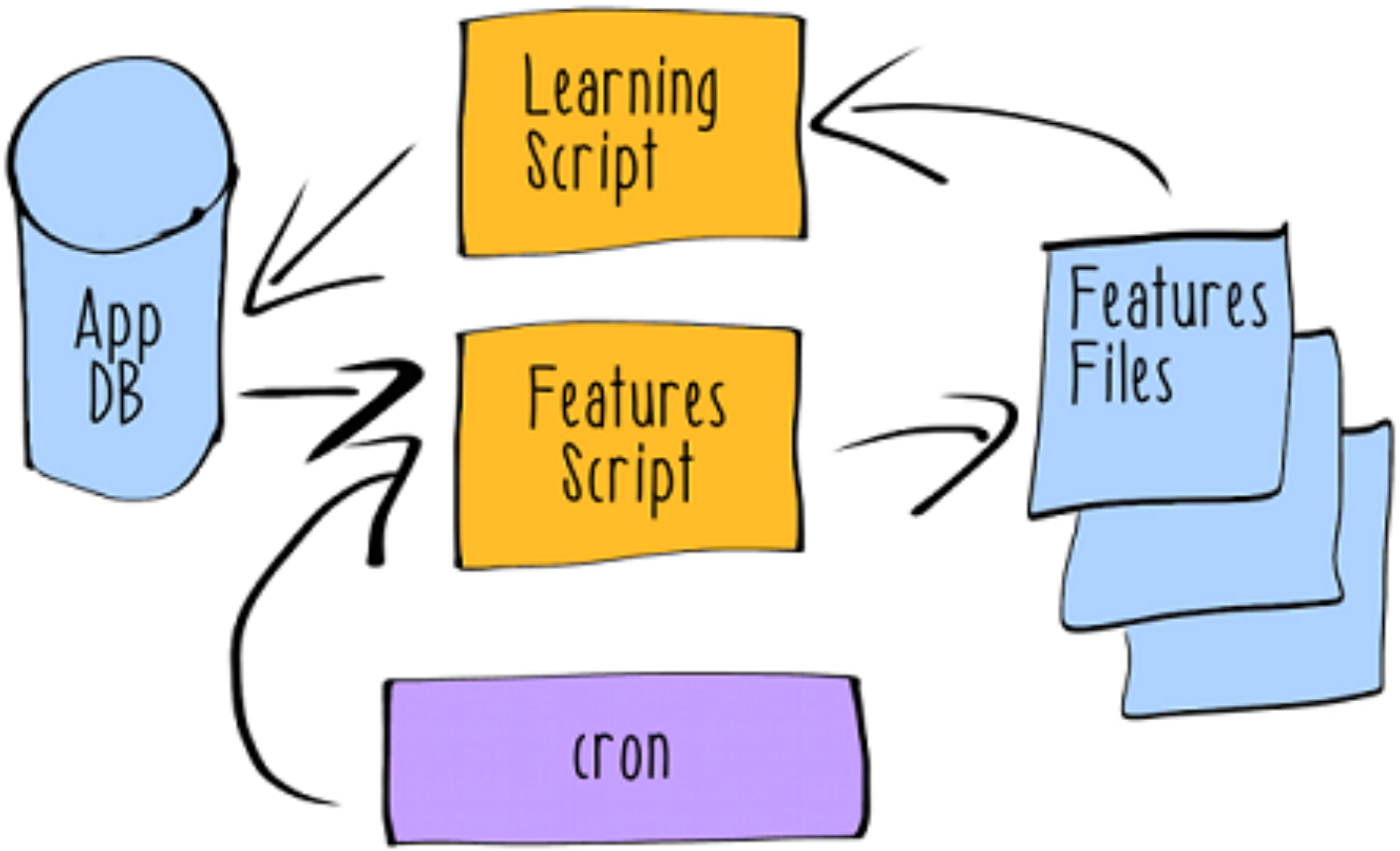
Things went from bad to worse!

- Appeared when data scientist implementing more feature-extraction functionality to hopefully improve modeling quality.
- The app slowed down dramatically.

What was the reason?

- Sending the data from the app back to server took way too long to maintain reasonable responsiveness.
- The server would throw an exception any time the prediction functionality saw any of the new features.
- The prediction functionality within the server that supported the app didn't handle the new features properly.

Pooch Predictor 1.1 Architecture



So what they did??

- After understanding where things had gone wrong, the team quickly rolled back all of the new functionality and restored the app to a normal operational state.
- They held a retrospective to build a better system...

Building a better system (operation)

- Sniffle must remain responsive, regardless of any problems with the predictive system.
- The predictive component needs to be less tightly coupled to the rest of the systems.
- The predictive system needs to behave predictably regardless of high load or errors in the system itself.

Building a better system (development)

- It should be easier for different developers to make changes to the predictive system.
- The code needs to use different programming idioms that ensure better performance.

Building a better system (ML)

- The predictive system must measure its modeling performance better.
- The predictive system should support evolution and change.
- The predictive system should support online experimentation.
- Easy for developers to supervise the predictive system and rapidly correct any rogue behavior.

Sniffable team missed something big, right?

- They built what initially looked like a useful ML system that added value to their core product.
- Production issues with their ML system frequently pulled the team away from work on improvements to the capability of the system.
- Even though they had a bunch of smart people to develop model to predict the dynamics of dog-based social networking, their system repeatedly failed at its mission.

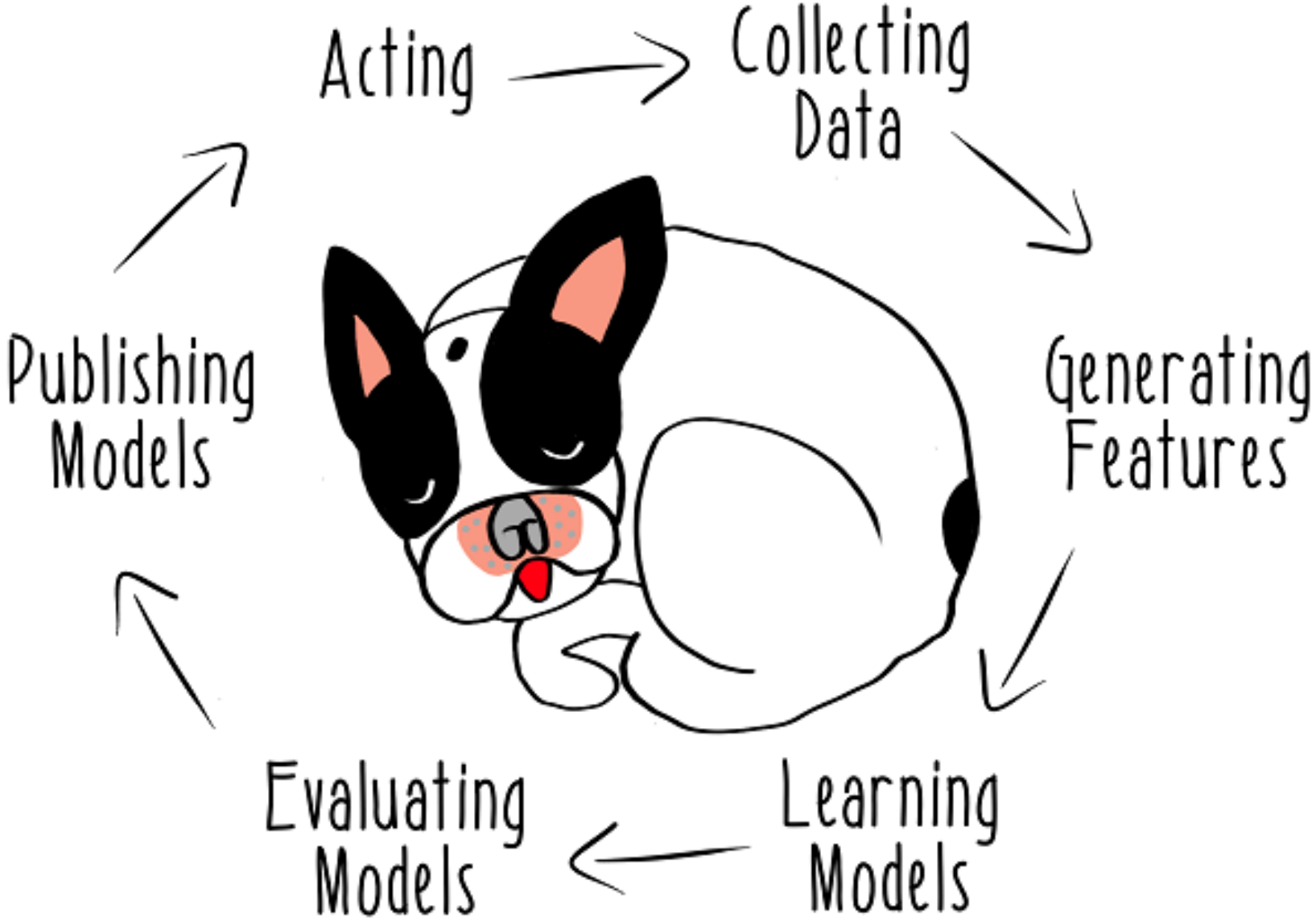
Building ML systems is hard!

- The data scientist knew how to do ML.
- Pooch Predictor totally worked on his laptop.
- But the data scientist was not thinking of ML as a system - but as a technique!
- Pooch Predictor was a failure both as a predictive system and as a software.

To build it right: Reactive ML

- Reactive ML = ML as a system

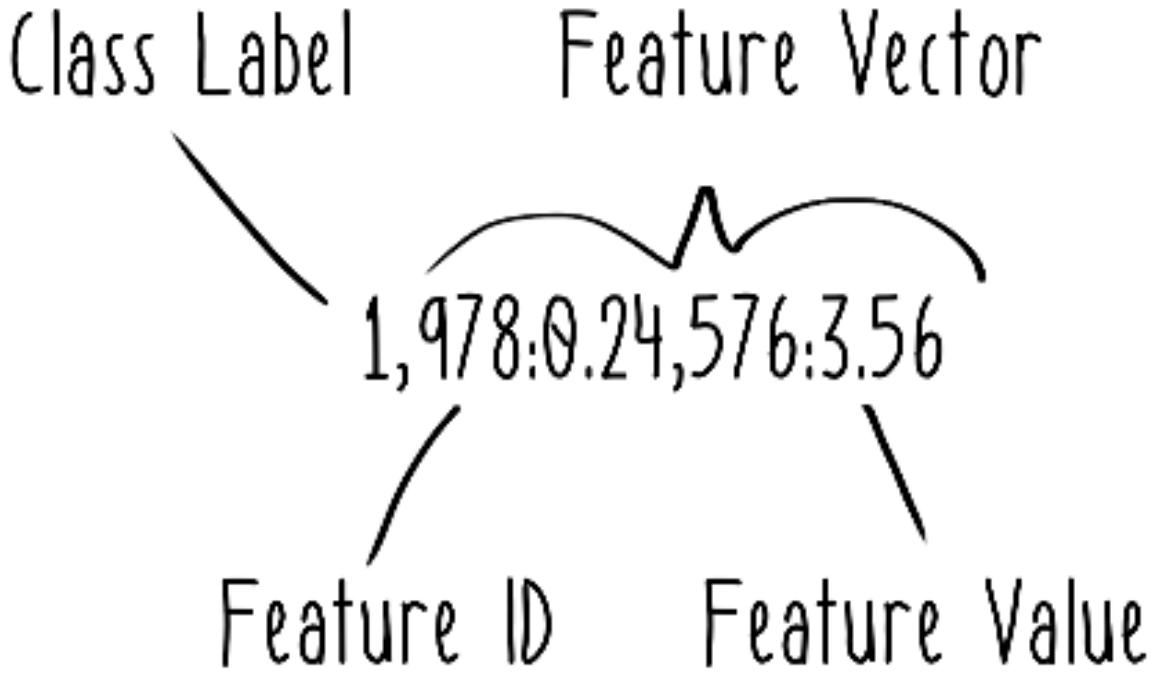
ML Pipeline


Acting
Collecting Data
Generating Features
Learning Models
Evaluating Models
Publishing Models

Starting from the beginning

- A ML system must collect data from the outside world.
- In the Pooch Predictor example, the team was trying to skip this concern by using the data that their application already had.
- Obviously, this approach was quick, but it tightly couples the Sniffable application data model to the Pooch Predictor data model.

Derived representations of the raw data = instances



What is "feature"?

- Features are meaningful data points derived from raw data related to the entity being predicted on.
- A Sniffable example of a "feature" would be ??
  - the number of friends a given dog has.

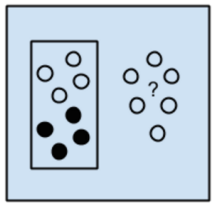
What is the feature vector?

- The feature values for a given instance are collectively called a feature vector.

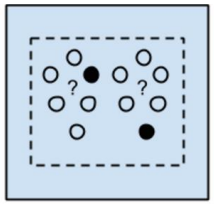
What is a concept?

- A concept is the thing that the system is trying to predict.
- In the context of Pooch Predictor, a concept would be the number of likes a given post receives.
- When a concept is discrete (i.e. not continuous), it can be called a class label.

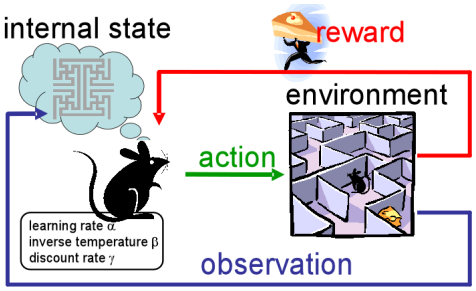
Types of Learning



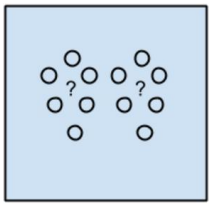
Supervised Learning



Semi-Supervised Learning



Reinforcement Learning



Unsupervised Learning

- Supervised: given data, predict label
- Unsupervised: given data, learn about that data
- RL: given data, choose action to maximize expected long-term reward

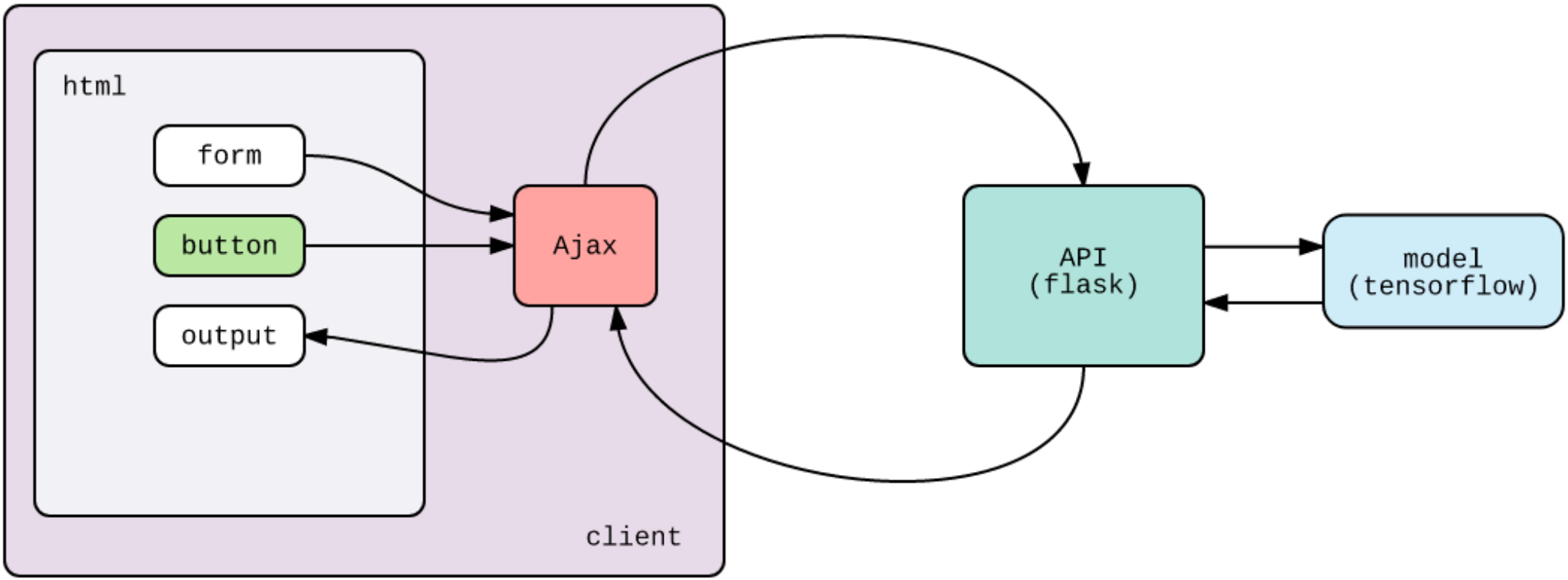
What is a "model"?

- A program that maps from features to predicted concepts.
- For the Pooch Predictor, a program that takes as input the feature representation of the hashtag data and returns the predicted number of likes that a given pupdate might receive.

What is model publishing?

- Model publishing means making the model program available outside of the context it was learned in,
- so that it can make predictions on data it hasn't seen before.

making the model program available outside of the context it was learned in



Remember any issue regarding model publishing in sniffable?

- Sniffable team largely skipped it in their original implementation.
- They didn't even set up their system to retrain the model on a regular basis.
- Their next approach at implementing model retraining also ran into difficulty, causing their models to be out of sync with their feature extractors.

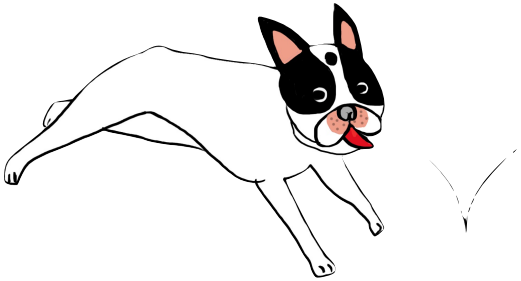
ML systems are different from transactional systems

- Some of the pooch predictor system problems stemmed from treating their predictive system like a transaction business application.
- An approach that relies upon strong consistency guarantees just doesn't work for modern distributed systems,
- Out of synch with the pervasive and intrinsic uncertainty in a machine learning system.

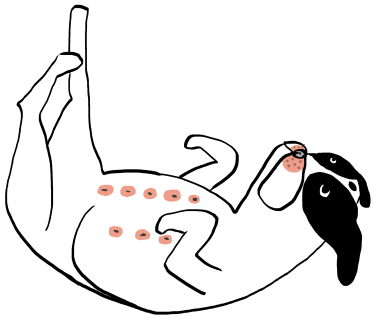
ML apps are dynamic systems

- Other problems the Sniffable team experienced had to do with not thinking about their system in dynamic terms.
- ML systems must evolve, and they must support parallel tracks for that evolution through experimentation.

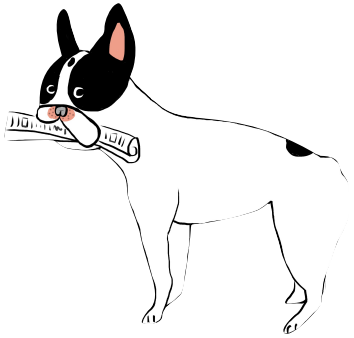
ML Quality Attributes for Reactive ML Systems



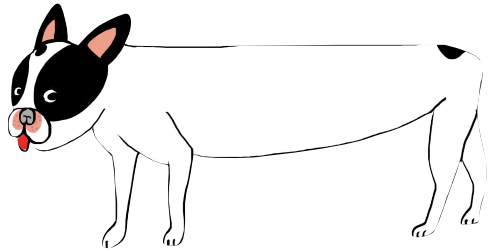
Responsive



Resilient



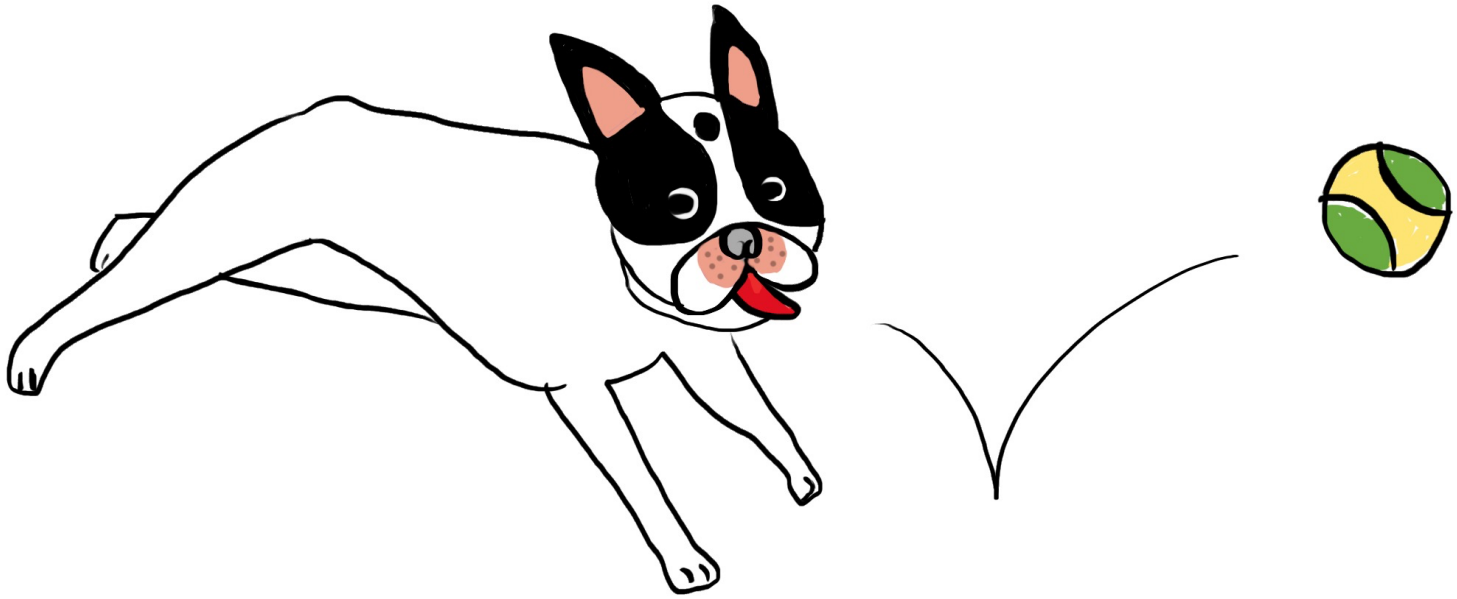
Message-Driven



Elastic

[Most of these figures are taken from Jeff Smith's slides and book]

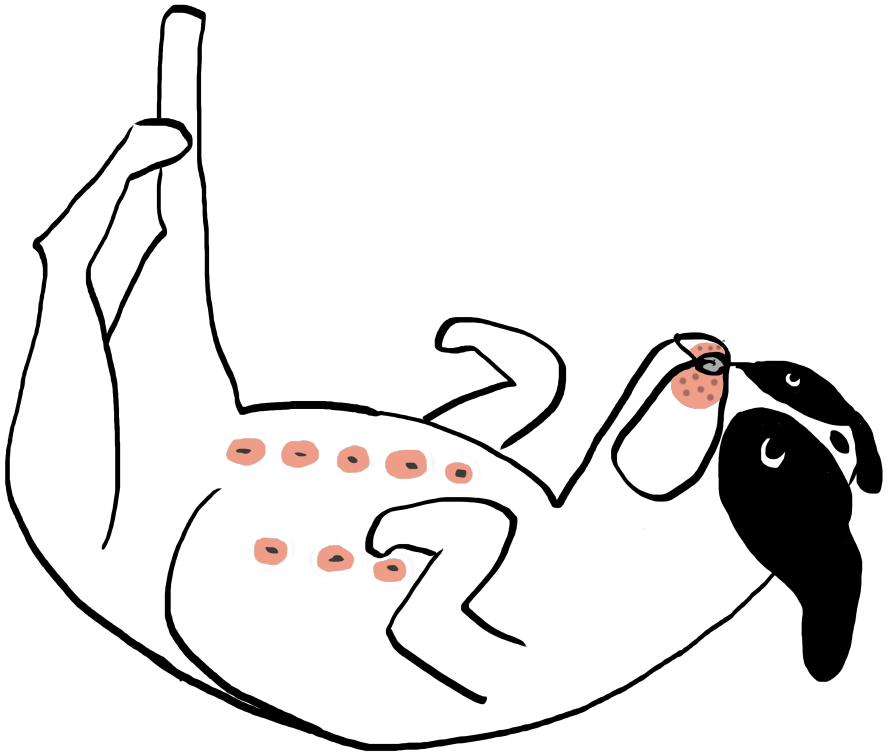
Responsive



Responsive

- If a system doesn't respond to its users, then it's useless.
- Think of the Sniffable team causing a massive slowdown in the Sniffable app due to the responsiveness of their ML system.

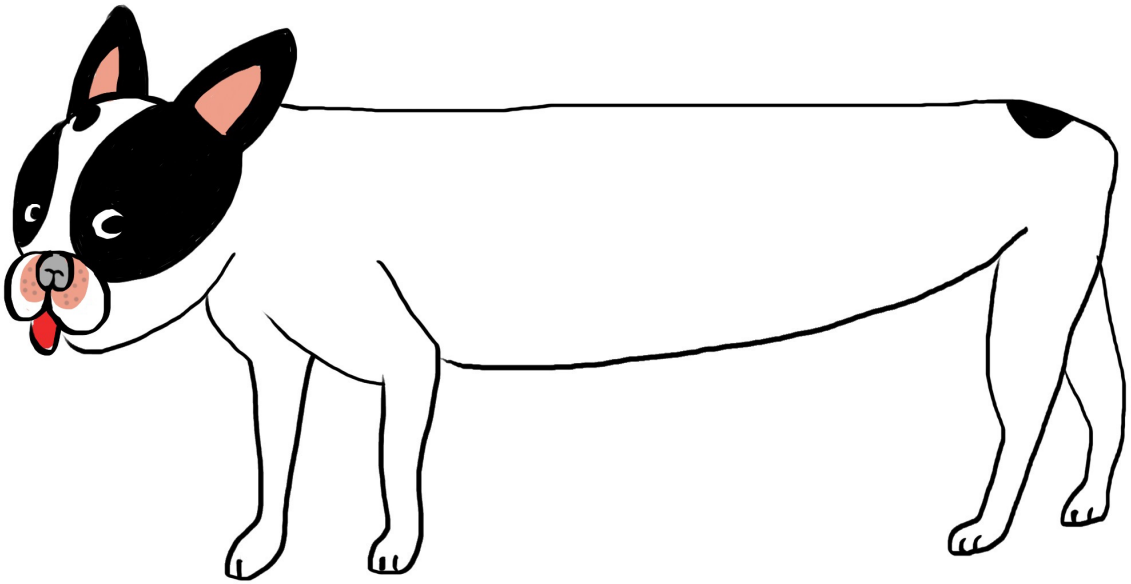
Resilient



Resilient

- Whether the cause is hardware, human error, or design flaws, software always breaks.
- Providing some sort of acceptable response even when things dont go as planned is a key part of ensuring that users view a system as being responsive.

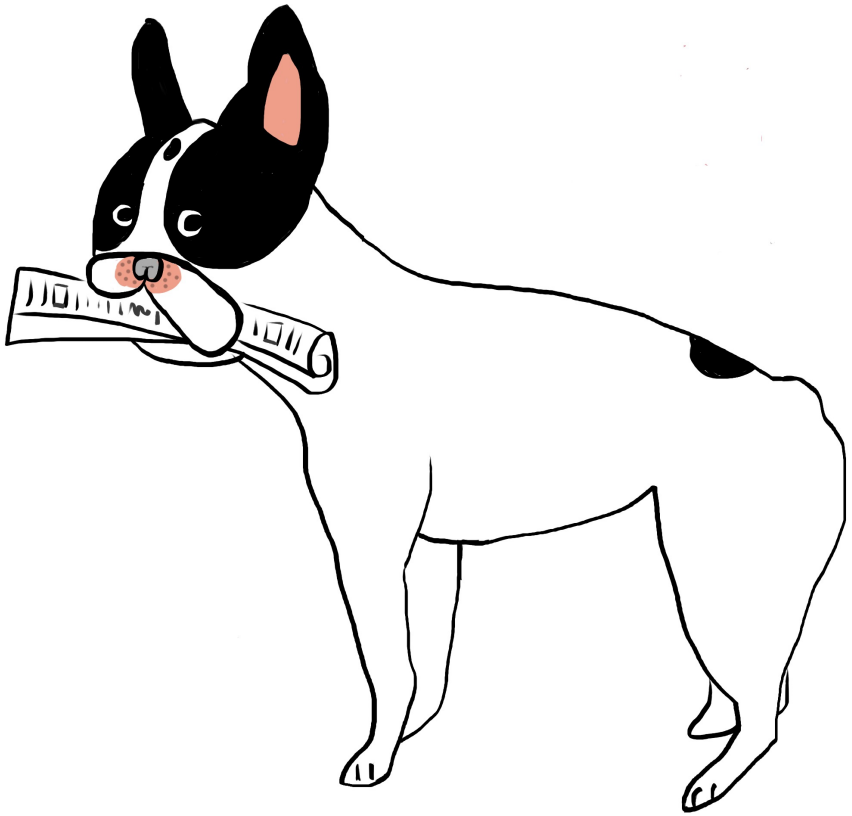
Elastic



Elastic

- Elastic systems should respond to increases or decreases in load.
- The Sniffable team saw this when their traffic ramped up and the Pooch Predictor system couldnt keep up with the load.

Message-driven



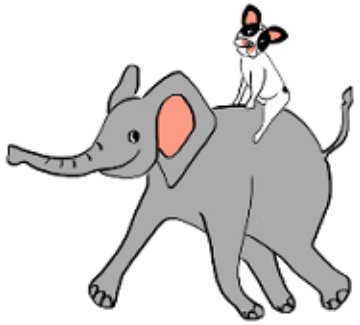
Message-driven

- A loosely coupled system organized around message passing can make it easier to detect failure or issues with load.
- Moreover, a design with this trait helps contain any of the effects of errors isolated, rather than flaming production issues that needed to be immediately addressed, as they were in Pooch Predictor.

Wait, how do you build a system that actually has these qualities?

Replication

Containment



Supervision

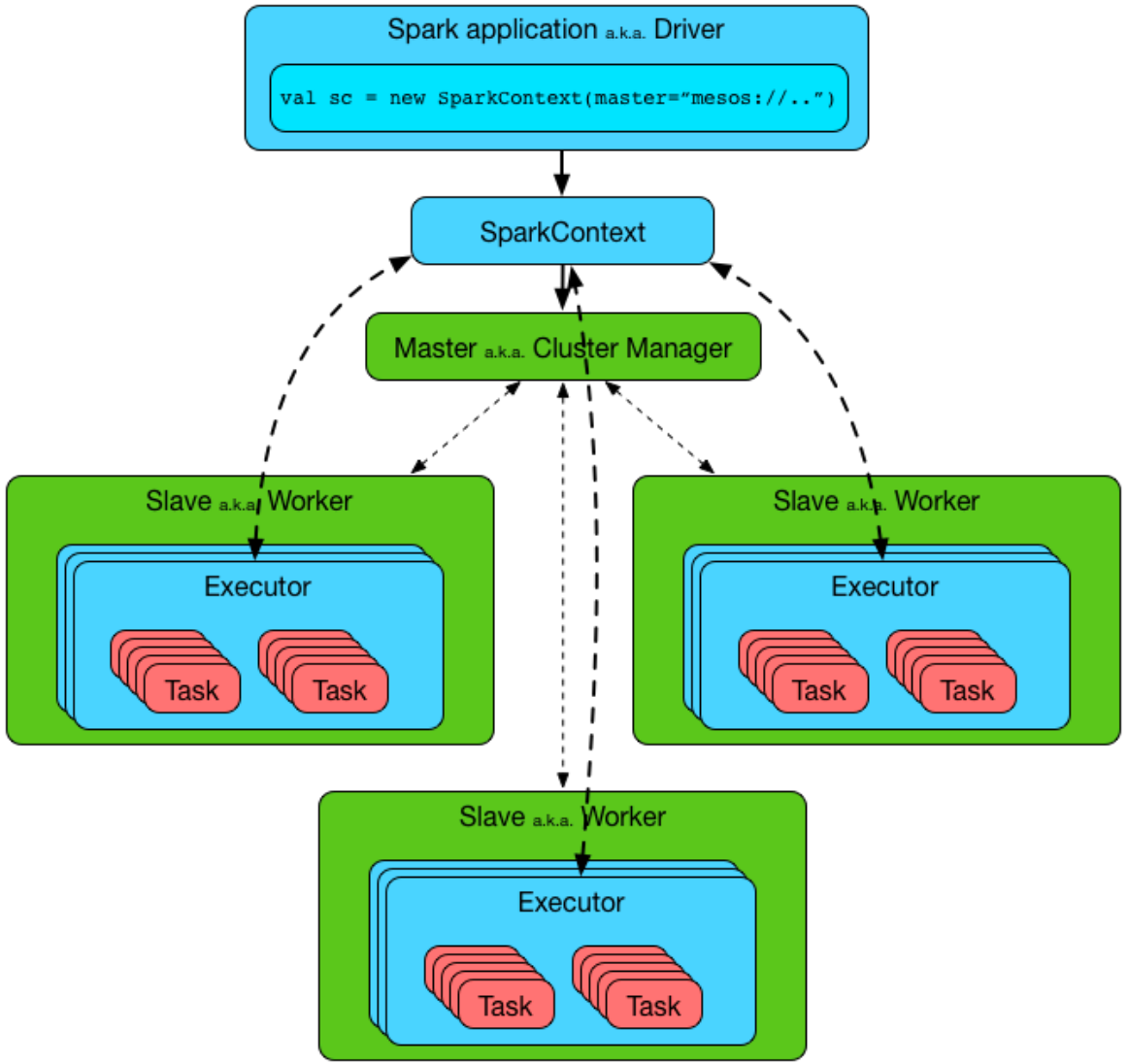
Tactics

Replication

- Data, whether at rest or in motion, should be redundantly stored or processed.
- In the Sniffable example, there was a time when the server that ran the model-learning job failed, and no model was learned.
  - two or more model-learning jobs

An Example: Spark Architecture

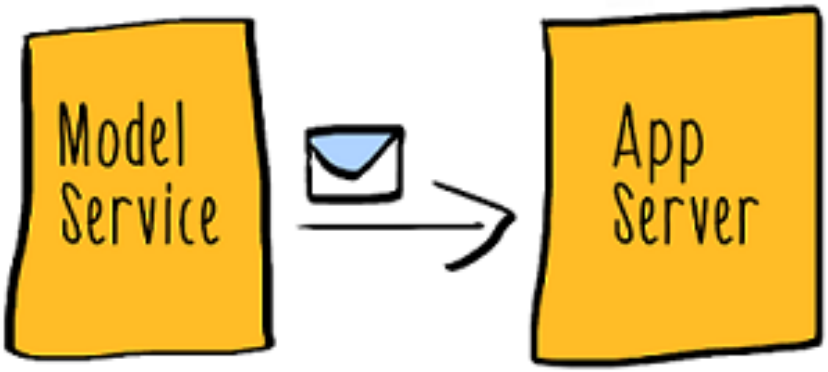
Rather than requiring you to always have two pipelines executing, Spark gives you automatic, fine-grained replication so that the system can recover from failure.



Containment

- Prevent the failure of any single component of the system from affecting any other component.
- Docker or rkt? no, this strategy isn't about any one implementation. Containment can be implemented using many different systems.
- The point is to prevent the sort of cascading failure we saw in Pooch Predictor, and to do so at a structural level.

A Contained Model-Serving Architecture

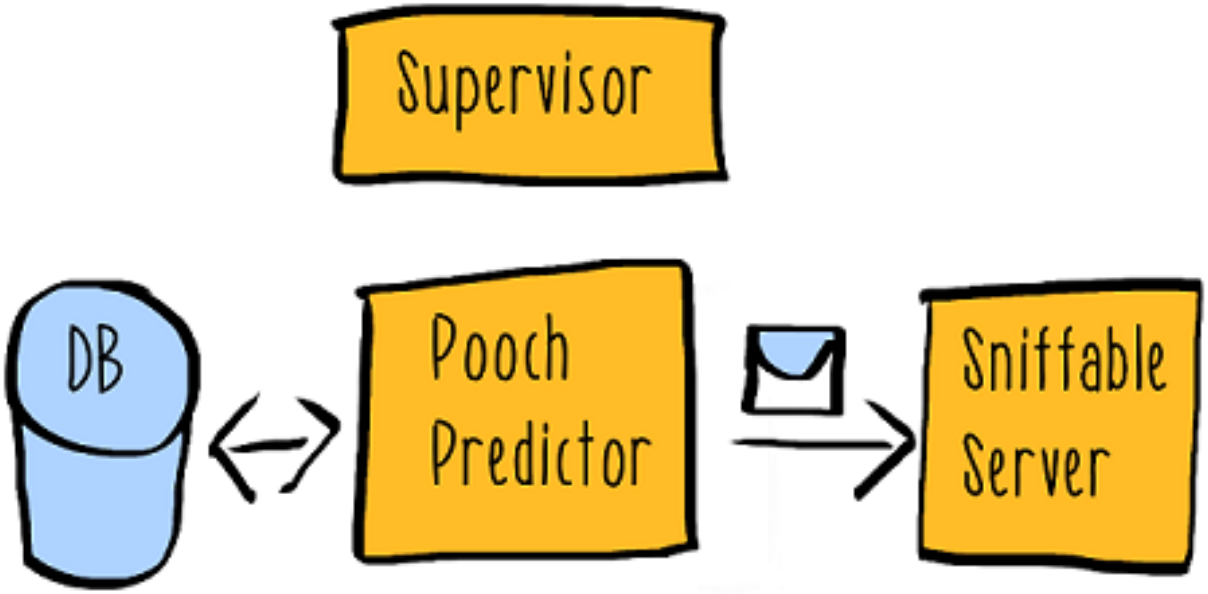


- Remember the model and the features were out of sync, resulting in exceptions during model serving??

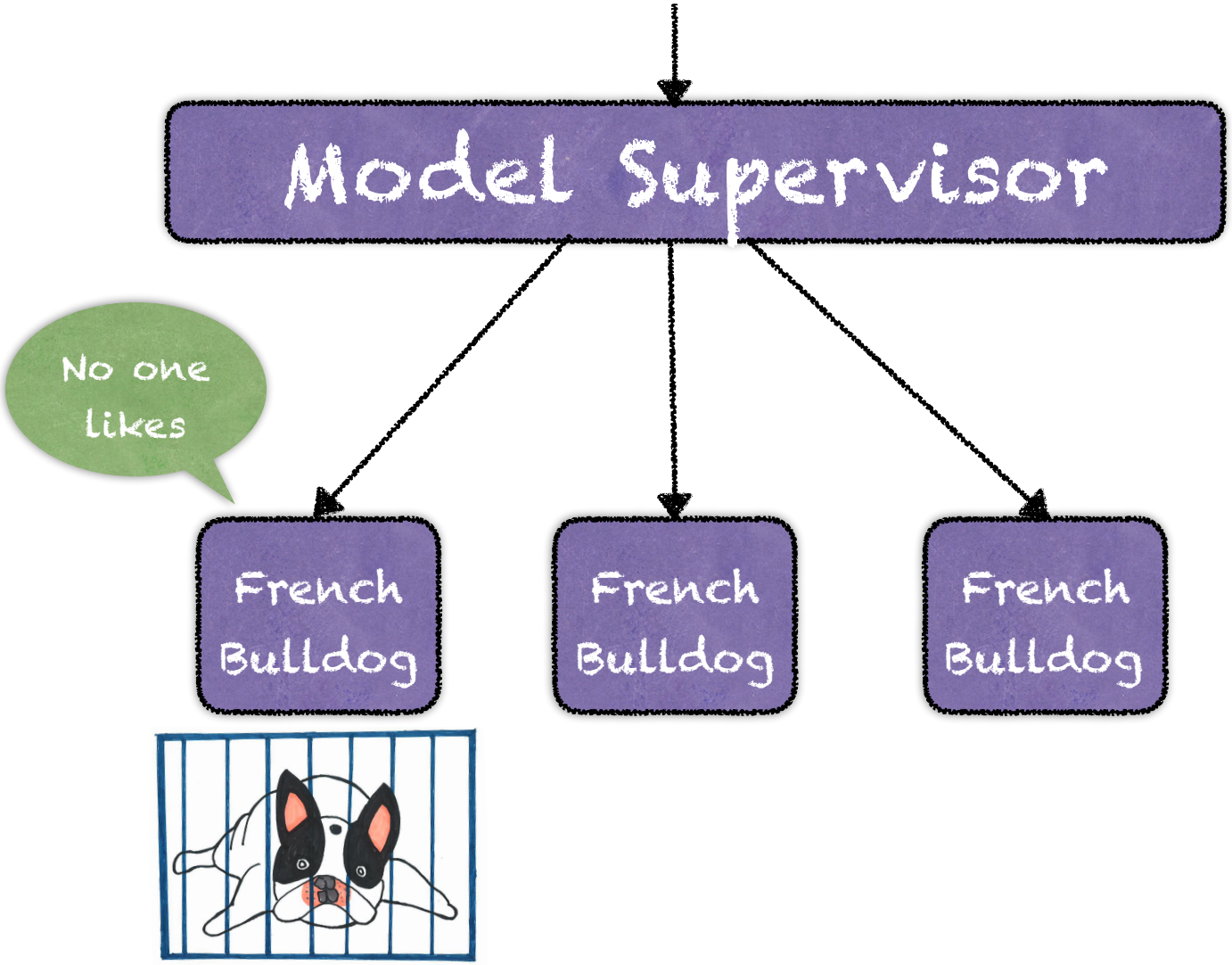
Supervision

- Identify the components that could fail and some other component is responsible for their lifecycles.

A Supervisory Architecture



- The published models are observed by the model supervisor. Should their behavior ever deviate from acceptable bounds, the supervisor would stop sending them messages requesting predictions.


Model Supervisor
No one likes
French Bulldog
French Bulldog
French Bulldog

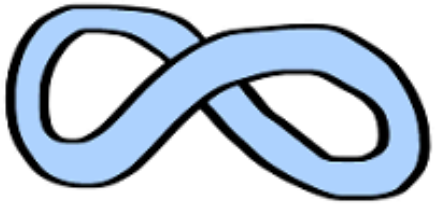
Supervisory architecture allows self-healing

- The model supervisor could even completely destroy a model it knows to be bad, allowing the system to be self-healing.

Why Turning ML systems to reactive

- Ultimately, a reactive ML system gives you the ability to deliver value through ever better predictions.

Strategies to make ML reactive



Infinite Data

Uncertain Data

Laziness

Pure Functions

Immutable Facts

Possible Worlds

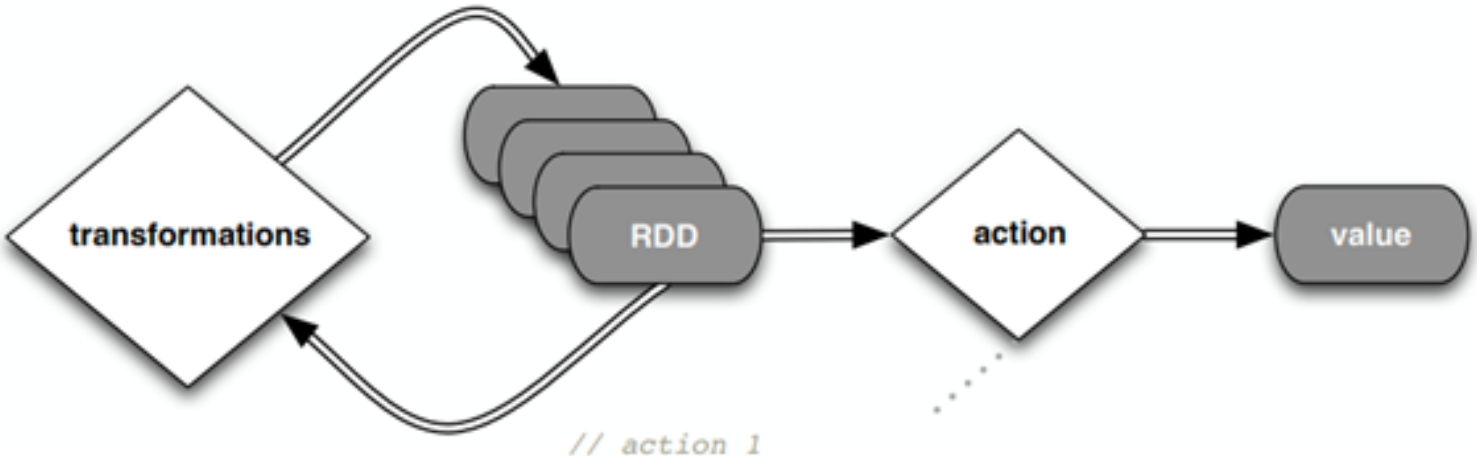
Infinite Data

- Ideally, with its new ML capabilities, Sniffable is going to take off and see tons of traffic.
  - Some posts would have big dogs, others small ones.
  - Some posts would use filters, and others would be more natural.
  - Some would be rich in hashtags, and some wouldn't have any annotations.

Strategy 1: Laziness

- Delay of execution, to separate the composition of functions to execute from their actual execution.
- Conceive of the data flow in terms of infinite streams instead of finite batches.

Lazy Evaluation in Spark



- The execution will not start until an action is triggered

Strategy 2: Pure functions

- Evaluating the function must not result in some sort of side effect, such as changing the state of some variable or performing some I/O.
- Functions can be passed to other functions as arguments -> Higher Order Functions

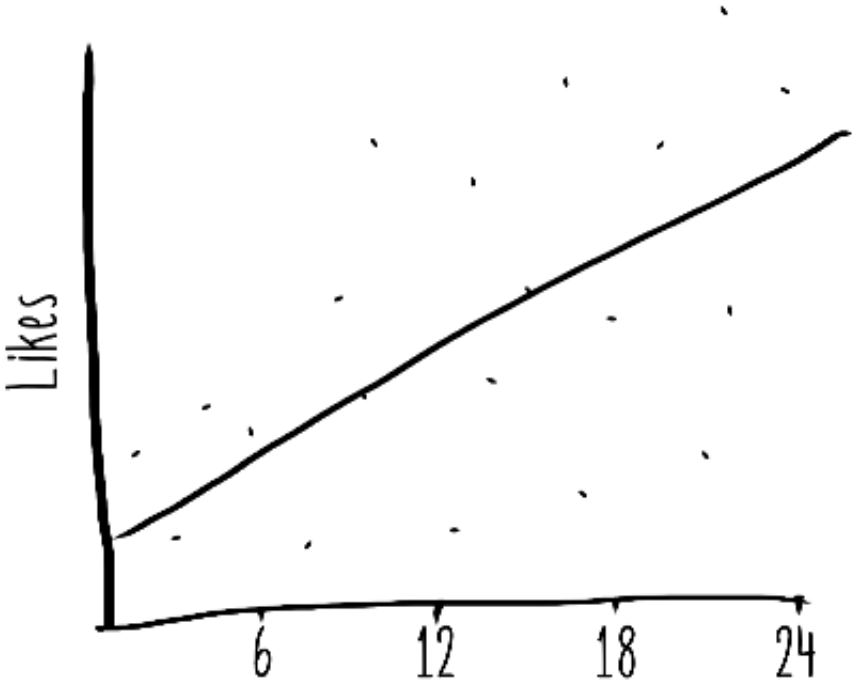
Uncertainty

- Even before making a prediction, a ML system must deal with the uncertainty of the real world outside of the ML system.
- For example, do sniffers using the hashtag #adorabull mean the same thing as sniffers using the hashtag #adorable, or should those be viewed as different features?

Strategy 3: Immutable facts

- Data that can simply be written once and never modified
- The use of immutable facts will allow us to reason about uncertain views of the world at specific points in time.
- Having a complete record of all facts that occurred over the lifetime of the system will also enable important machine learning, like model experimentation and automatic model validation.

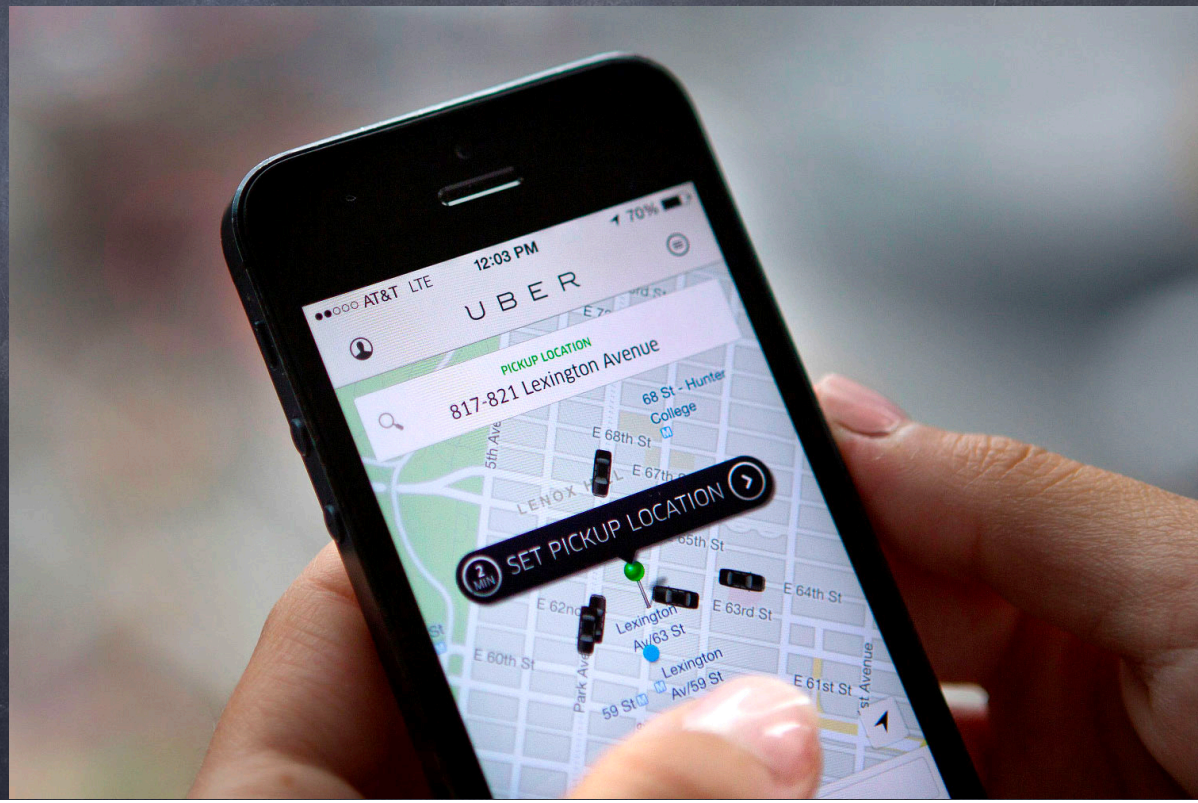
Strategy 4: Possible Worlds



Model Type	Pupdates	Likes/Pupdate
Historical	6,500	23
Machine-Learned	7,250	28

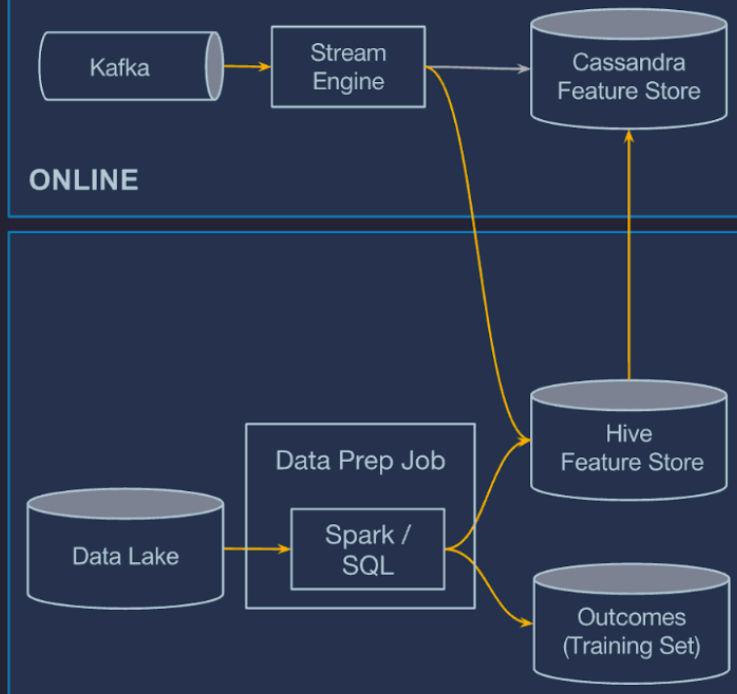
Until now things were fictional, lets see how things are done in real-world

At Uber

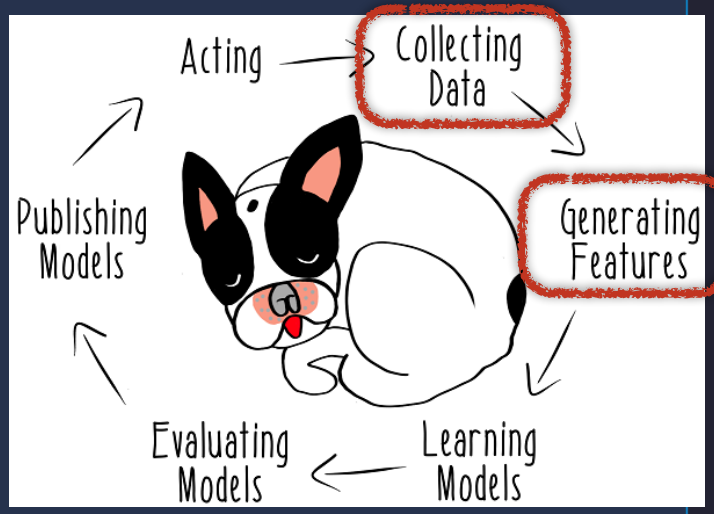


Production-ready ML at Scale (Uber)

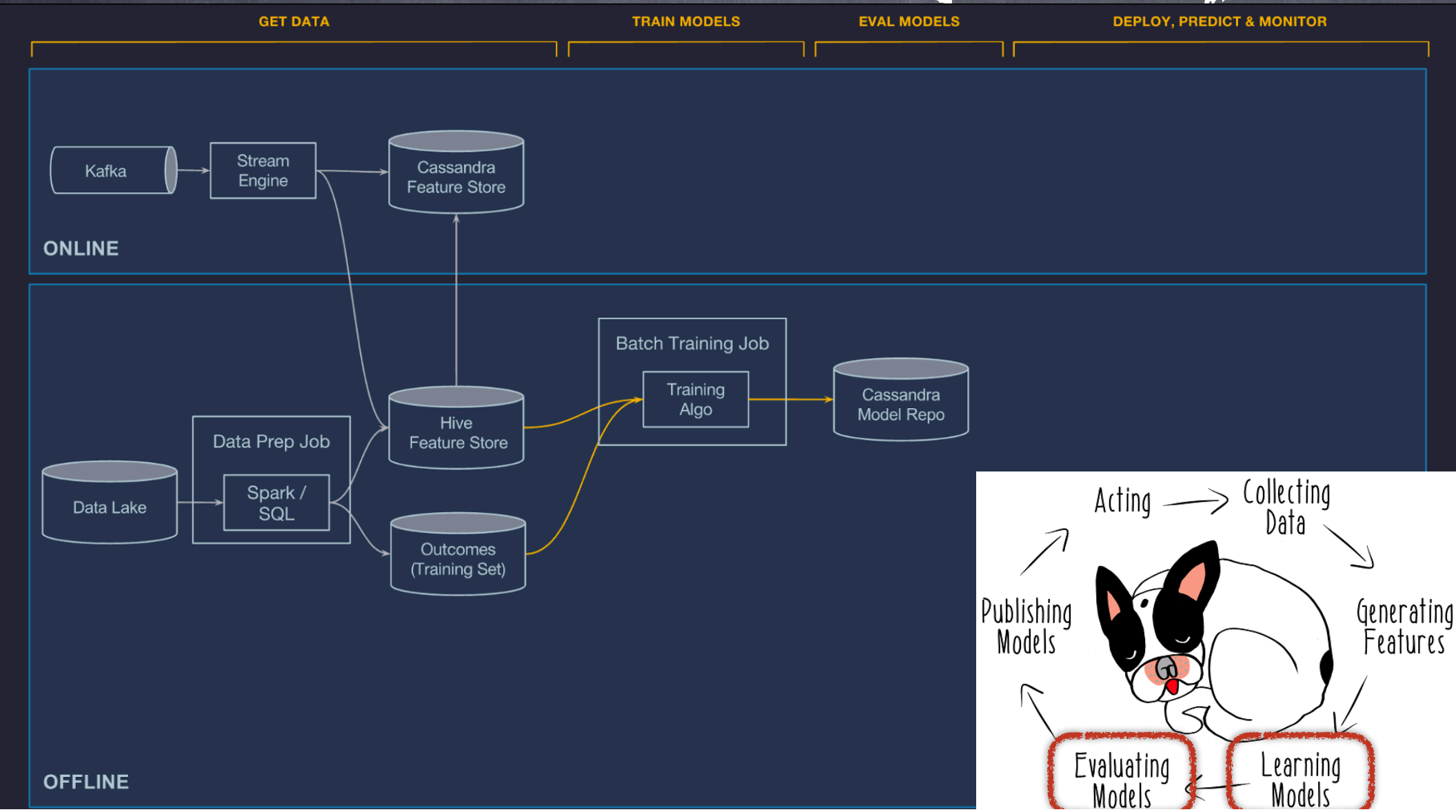
GET DATA | TRAIN MODELS | EVAL MODELS | DEPLOY, PREDICT & MONITOR



https://eng.uber.com/michelangelo/

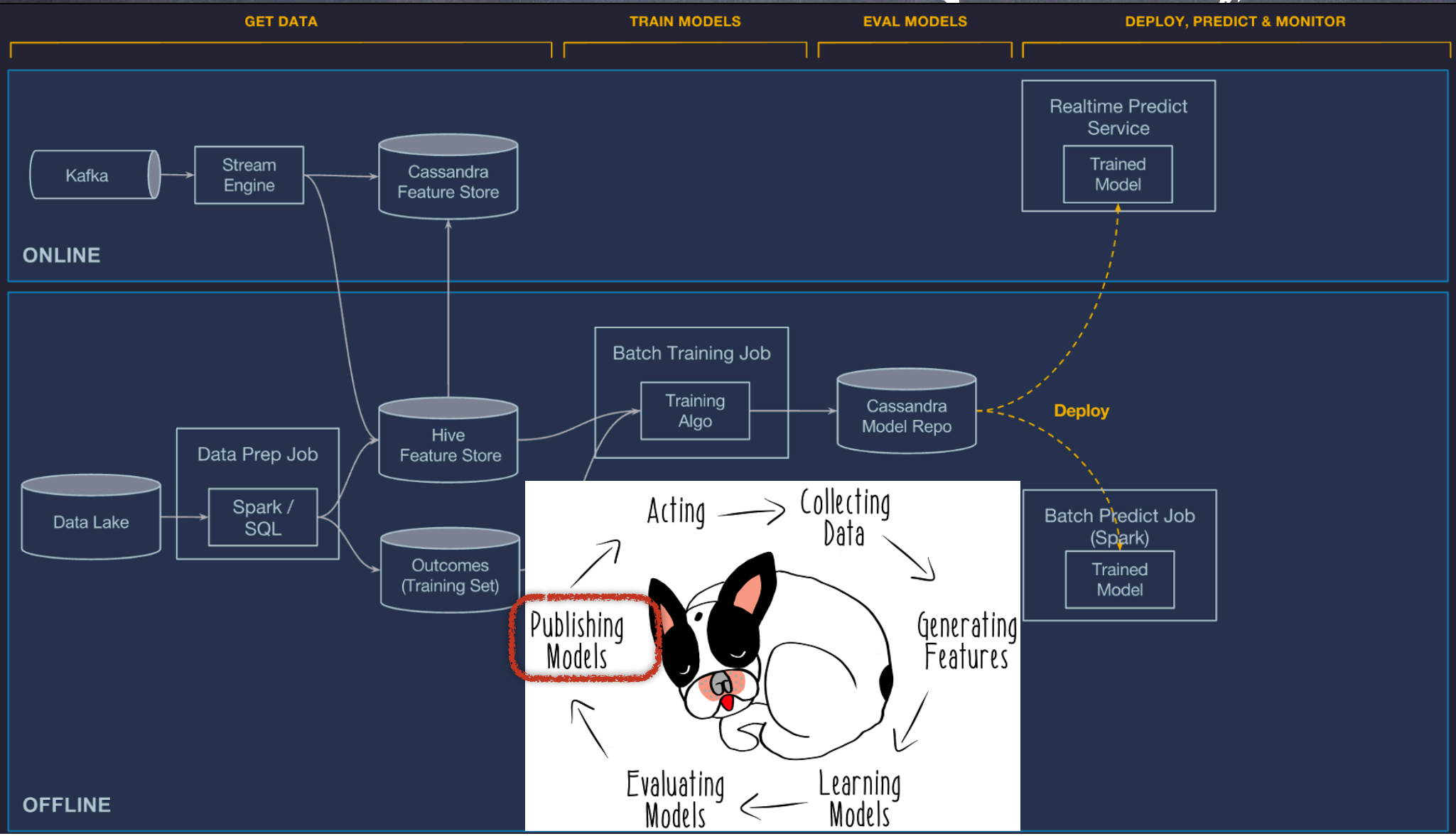


Production-ready ML at Scale (Uber)

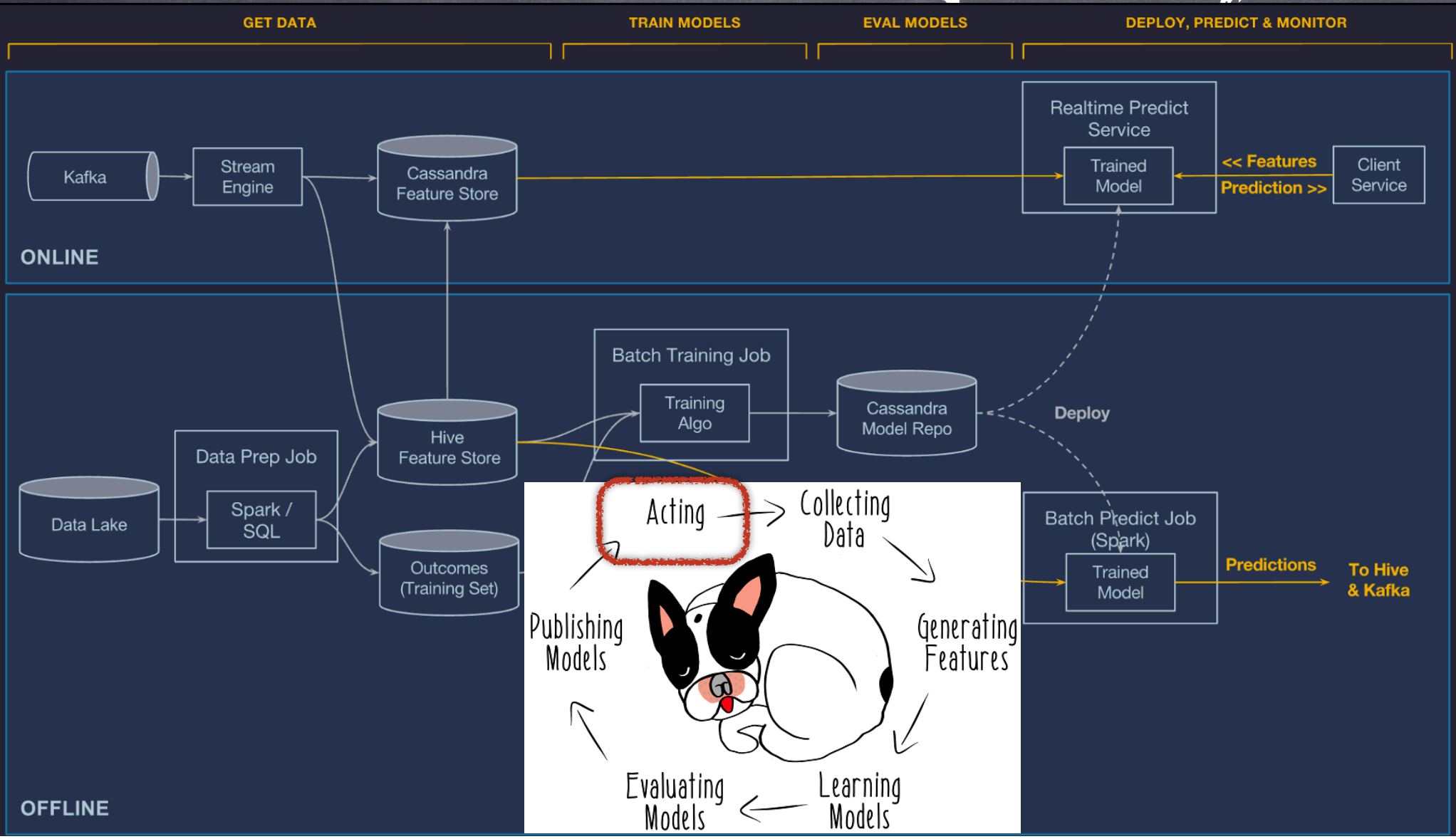


An example of model evaluation in practice (feature importance)

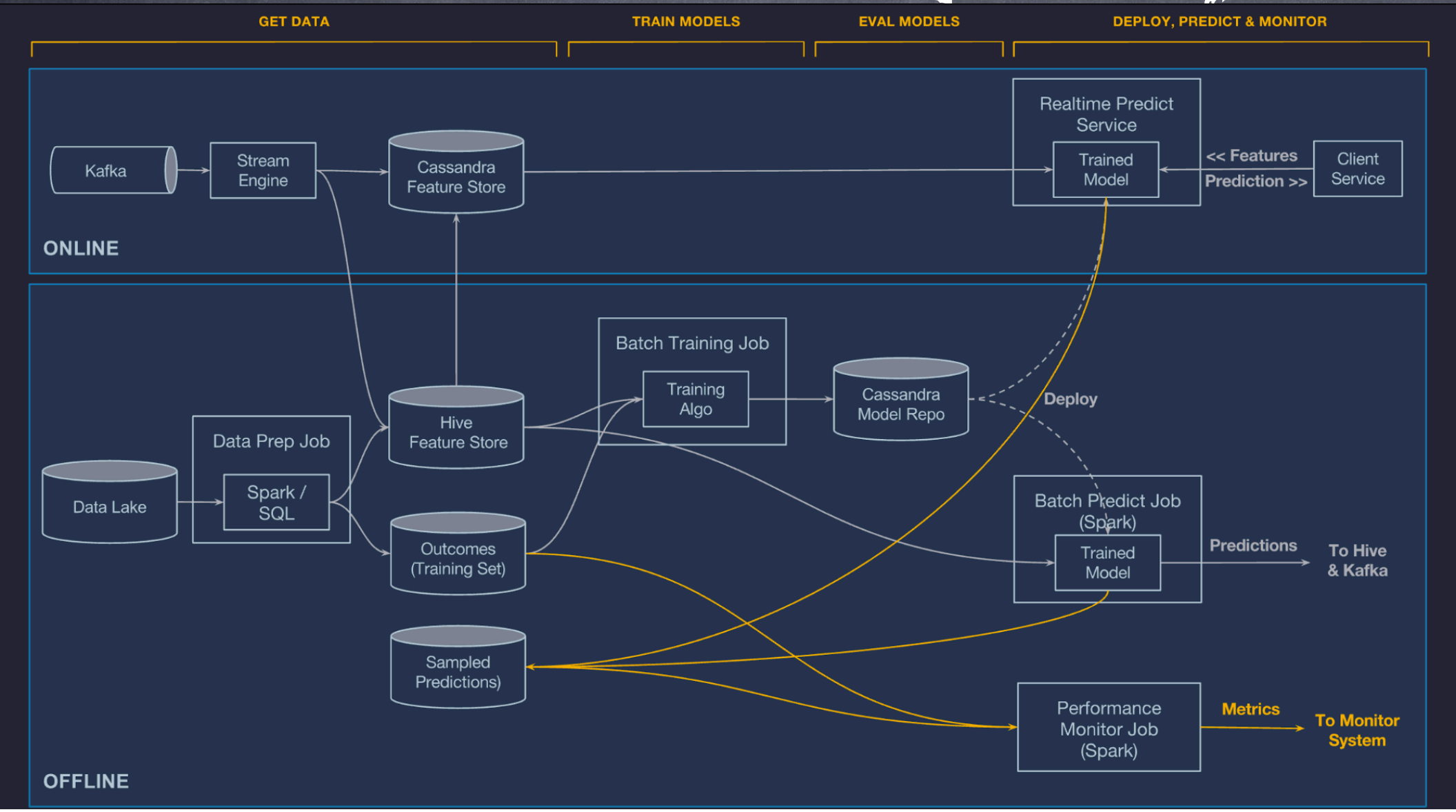
Production-ready ML at Scale (Uber)



Production-ready ML at Scale (Uber)



Production-ready ML at Scale (Uber)



Summary

- ML offers a powerful toolkit for building useful prediction systems.
- Building an ML model is only part of a larger system development process.
- We went through practices in distributed systems and software engineering in a pragmatic approach to building real-world ML systems.

Summary

- ML can be viewed as an application, and not just as a technique.
- We learned how to incorporate ML-based components into a larger complex system.
- Reactive is a way to build sophisticated systems, and some ML systems don't need to be sophisticated.

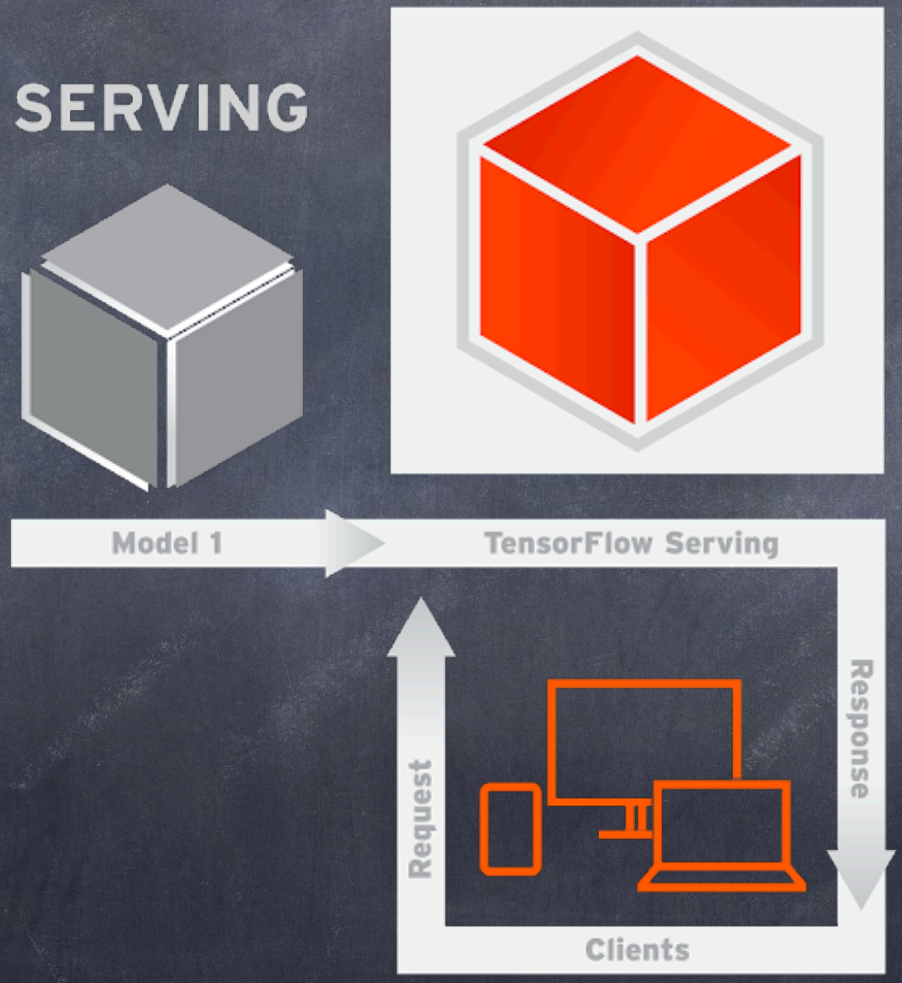
Resources

- Reading assignment: Reactive ML book, Chapter 1

Project

- Project3: How you can decrease latency of model serving (TF serving) by changing some of parameters like caching, remote procedure call protocol, etc.

Model serving



Model serving