UBER

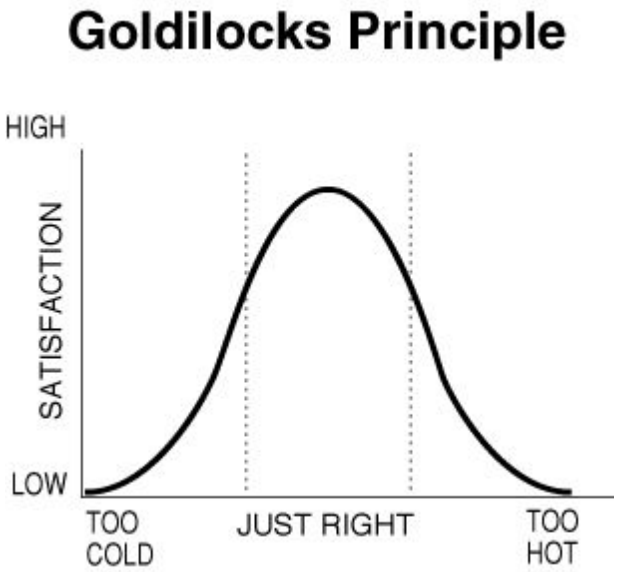
"Capacity Prediction" instead of "Capacity Planning": How Uber uses machine learning to accurately forecast resource utilization

Rick Boone, Senior Software Engineer II, Uber Capacity Engineering

Capacity Planning?

- Reliability: A service or platform will not degrade or fail due to having **too few resources**
- Efficiency: A service or platform will not be “wasteful” due to claiming **too many resources** (and not using them)

These are critical for the success of any software-based company, in both the short and long-term



Capacity Planning is hard

Capacity Planning is important...but it's hard to do at scale

- **Software is unpredictable**
- **Distributed environments are volatile and complex**
- **Engineers aren't psychics**
- **"Jedi" knowledge doesn't scale**

How can we get it right?

Predicting, not Planning

Planning

- Fuzzy
- Localized
- No measure of expected result
- “Plans are made to be broken”

Predicting

- Empirical
- Repeatable
- Scalable
- Grounded in data
- Expectation of success (with measurable confidence)

#Goals

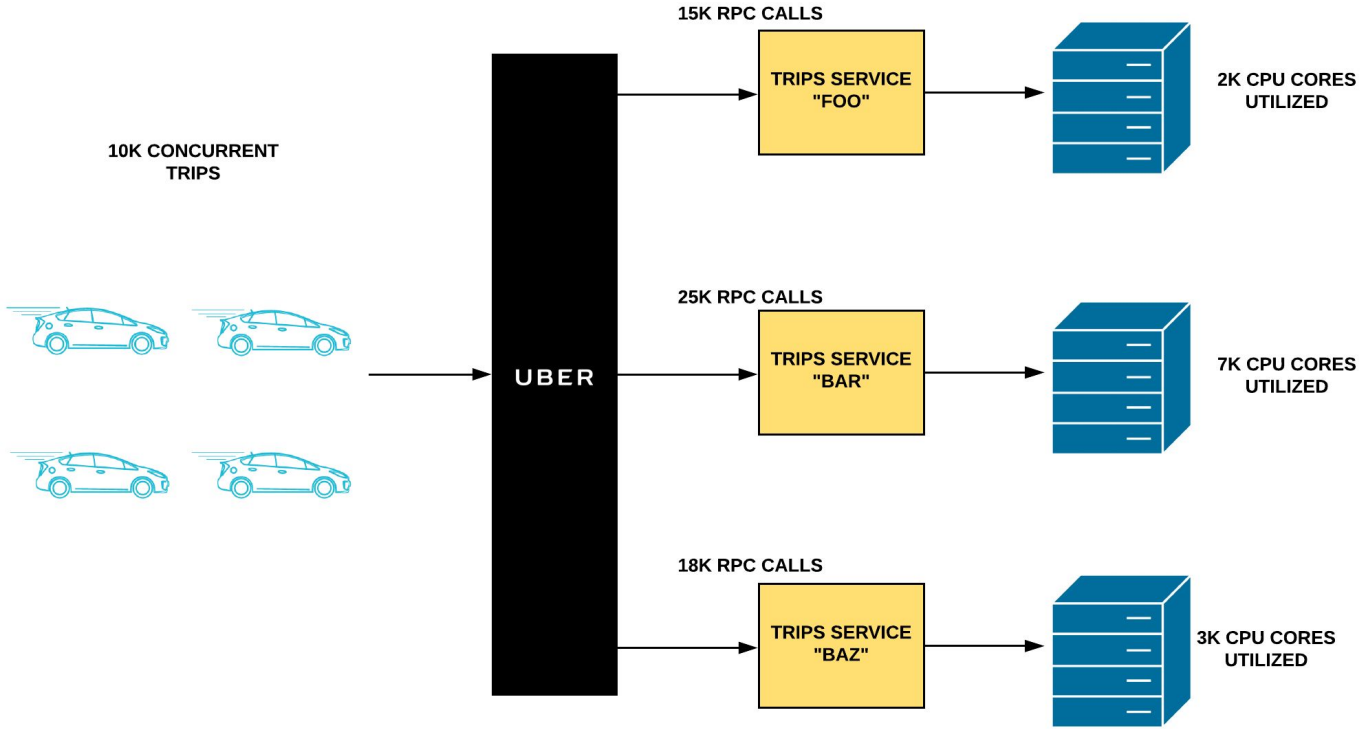
“Knowledge about how a service or platform behaves under all conditions and demands”

“Knowledge of Uber’s (and therefore, the service’s) future conditions and demands”

Business actions drive hardware usage

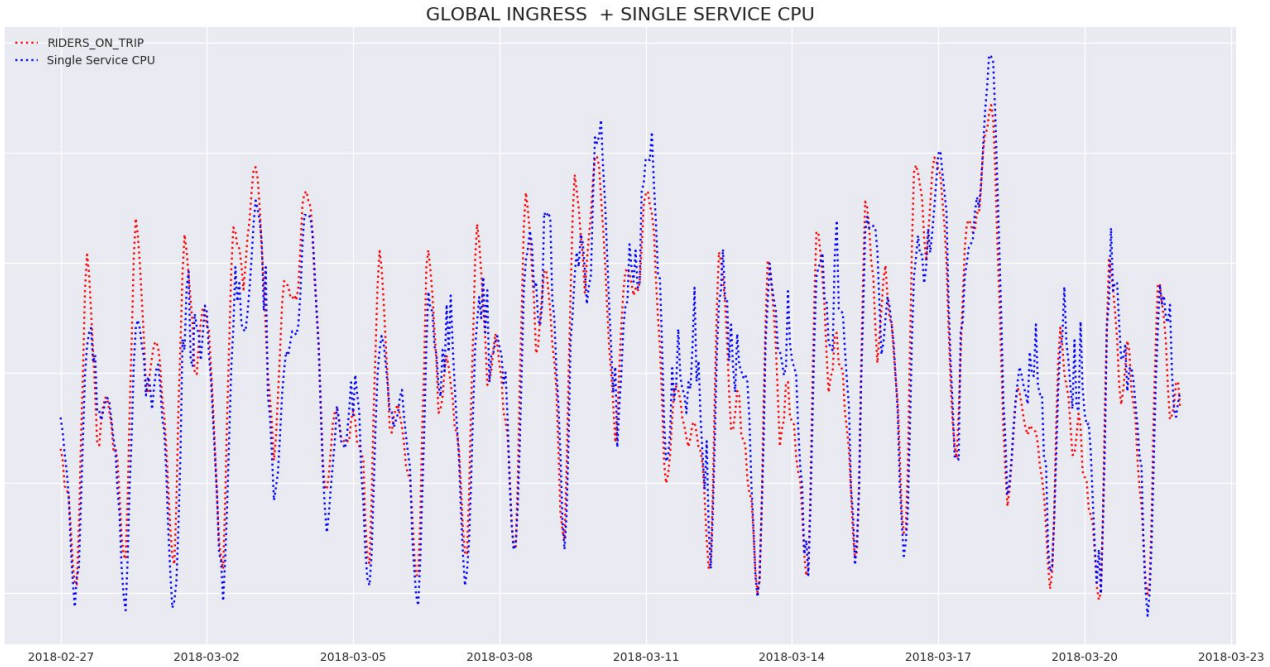
Our solution is built upon a simple, fundamental concept:

The behavior and performance of our services is directly driven by and tied to our key business metrics (“ingresses”), such as active trips, riders and drivers.



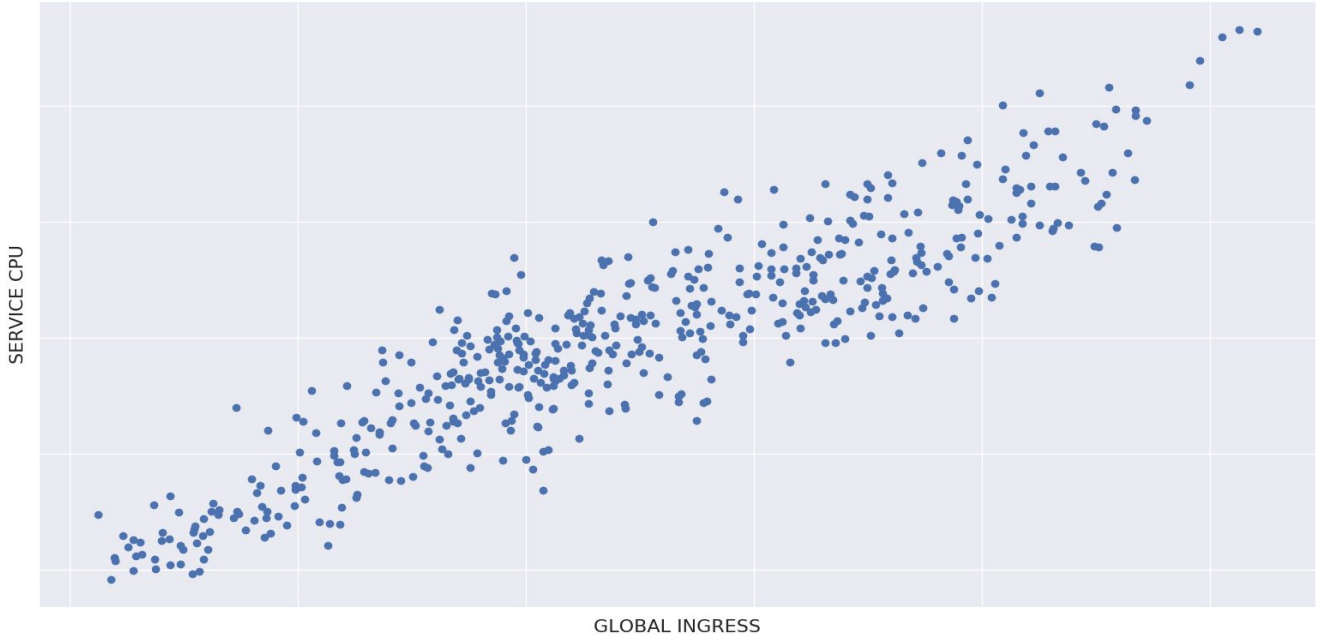
Demand -> Consumption

Timeseries plot of Ingress (Riders) vs CPU



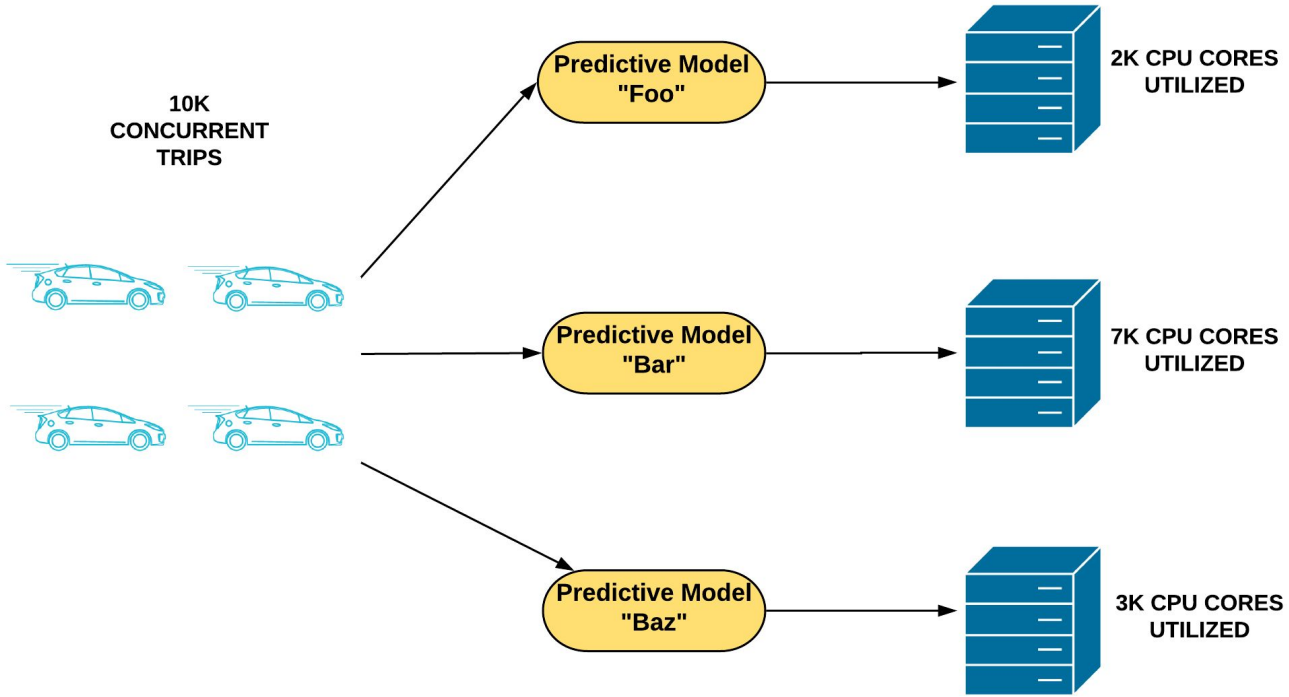
Demand -> Consumption

Scatter plot of Ingress vs CPU



Translational Model

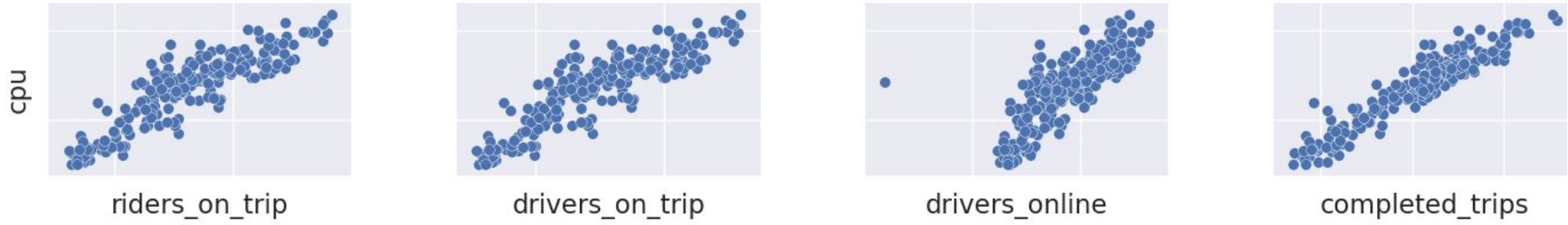
A predictive model to represent behavior and utilization



Which business metric?

Every service is driven by a different ingress/work flow.

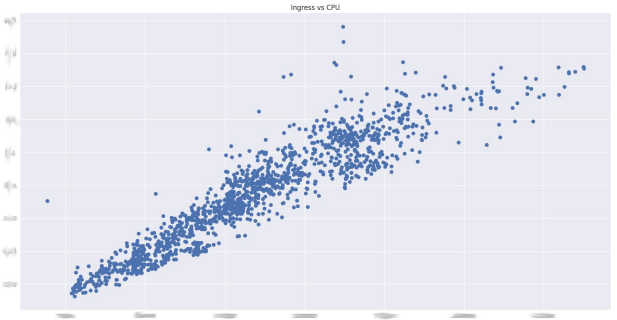
How can we choose the right one?



What's the relationship?

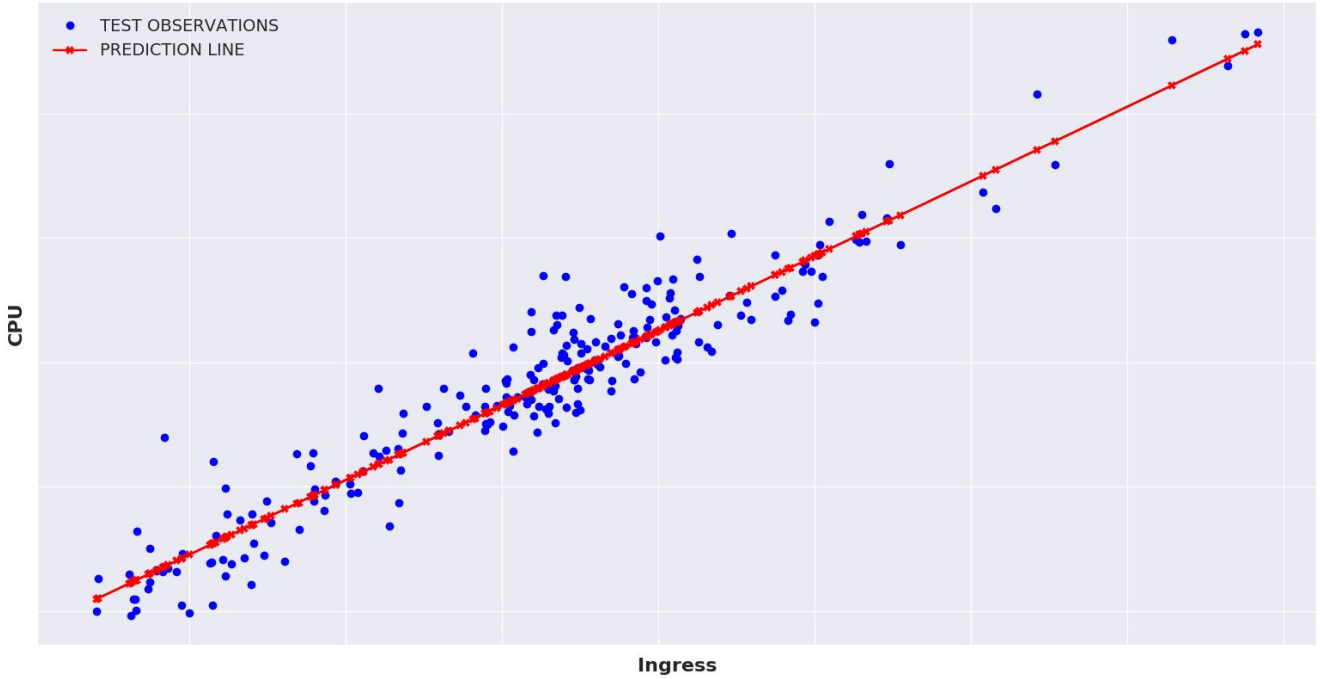
How can we mathematically represent this relationship between Ingress and CPU?

Machine Learning and statistical modeling are perfect for this



What's the relationship?

A **linear regression** for ingress onto CPU.
(Minimization performed with a genetic algorithm, though we started with gradient descent.)



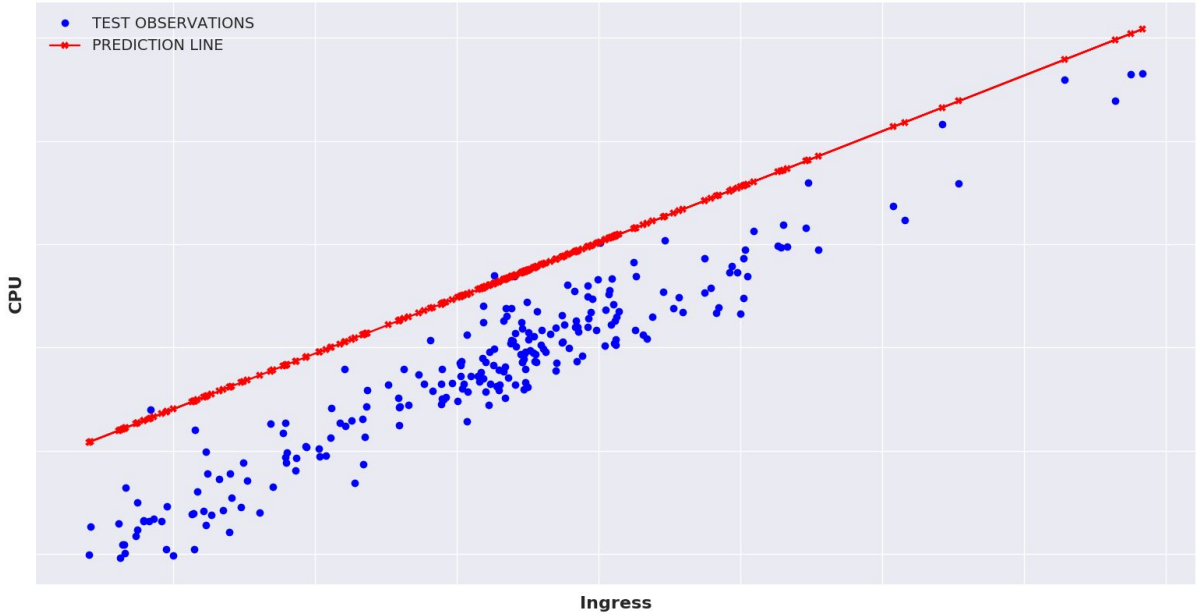
Wait. Something's wrong...

Capacity Safe

A ~~**linear**~~ **-> Quantile regression (@99%)** for ingress onto CPU

Y = mX + b
(CPU = m * Ingress + b)

We can now store (m, b) in a database to represent a service

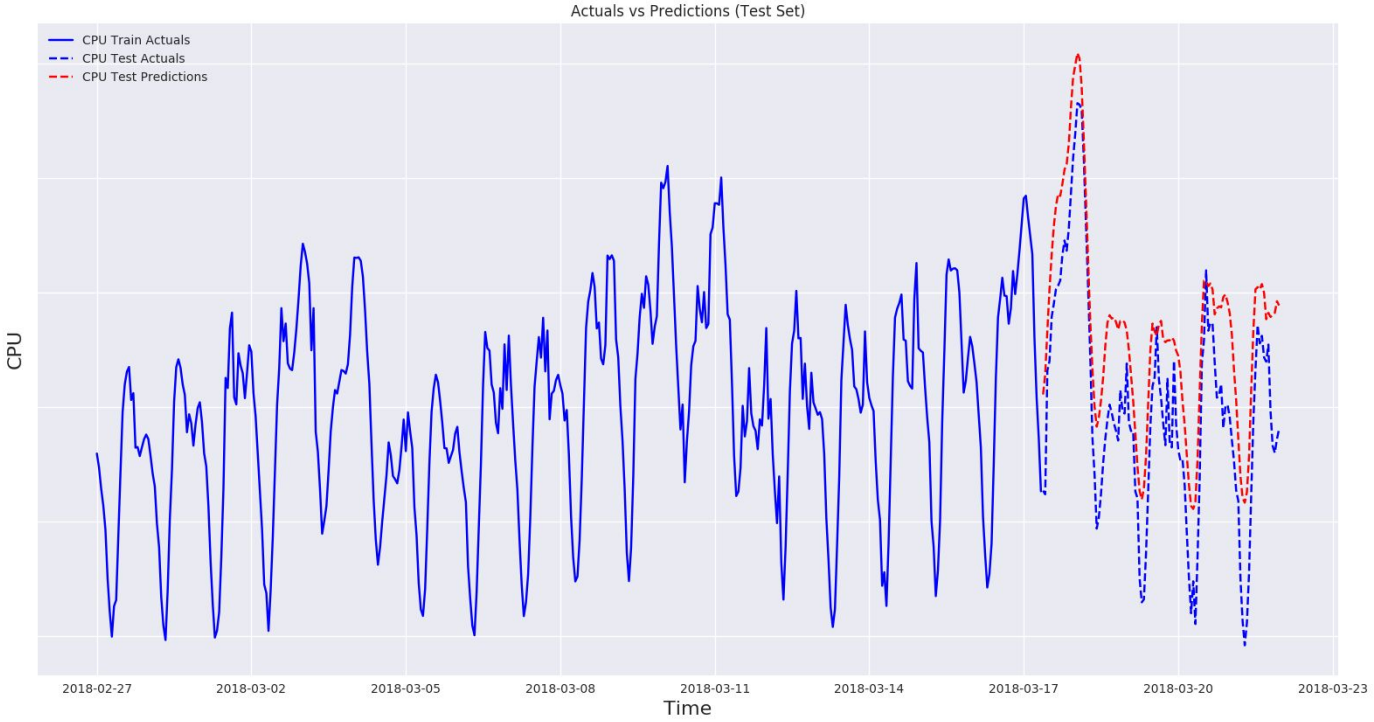


We also now have empirical accuracy scores (sMAPE, MAE) for our predictions, which can also be stored.

“Built Capacity Safe™”

Test, test, test

Performance against a test set



#Goals

“Knowledge about how a service or platform behaves under all conditions and demands”

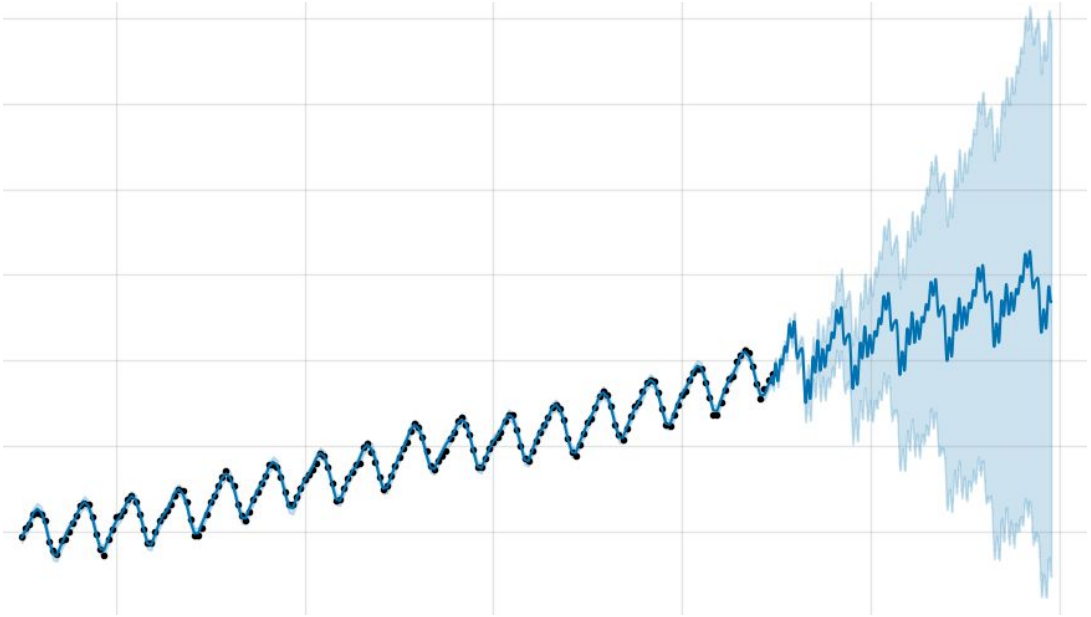
Done!
(Ingress -> Predicted CPU model)

“Knowledge of Uber’s (and therefore, the service’s) future conditions and demands”

How can we represent this?

One more piece...

What will Uber demand/ingresses look like in the future?



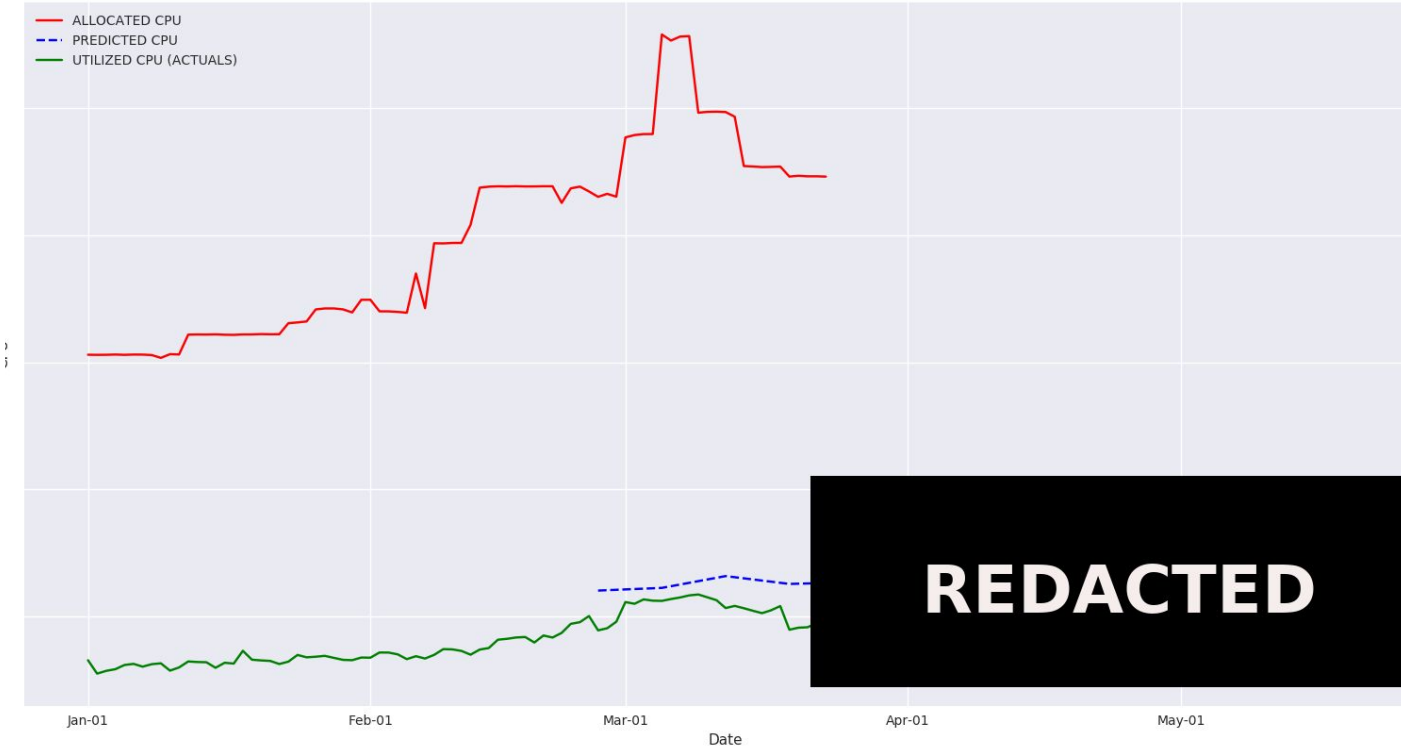
Observe, Relate, Predict

We have a data science team which provides accurate, long-term demand forecasts of our business metrics

(These are not Uber's actual business metrics. Of course. :))

Predictions!

Computationally-built CPU predictions for a service over the next few months!



Please, try this at home!

1. Consider what drives your services resource consumption

- Forecasted metrics are best, but not required
- Use empirical and repeatable processes to confirm relationship
  - Correlation analysis, Regularization w/ Feature Selection, Feature elimination
- Stay away from service RPS alone

2. Gather data and build aligned datasets

- If it's not available now, see if you can begin to ingest and store it
- Infrastructure data is VERY valuable

3. Build a predictive model via machine learning methods

- Many types of models are viable
  - Regressions, Decision Trees, Neural Nets
- Many off-the-shelf options are available
  - Scikit-Learn, R Libraries, TensorFlow….
- Models are "Build periodically, consume often" -- use your laptop to bootstrap!

4. Store the weights, accuracy scores and metadata

- Using Cassandra to store our model data

5. Apply the inputs

- Adhoc reports and analysis during bootstrapping

UBER

THANKS!!

Email: kineticrick@gmail.com
LinkedIn: https://www.linkedin.com/in/kineticrick/

Many thanks to the following engineers + data scientists:

Niels Lindgren
Scott Phung
Chen Lin
&
Calvin Worsnup