

# SOFTWARE ARCHITECTURE OF AI-ENABLED SYSTEMS

Christian Kaestner

Required reading:

- ◻ Hulten, Geoff. "[Building Intelligent Systems: A Guide to Machine Learning Engineering.](#)" Apress, 2018, Chapter 13 (Where Intelligence Lives).
- ◻ Daniel Smith. "[Exploring Development Patterns in Data Science.](#)" TheoryLane Blog Post. 2017.



# LEARNING GOALS

- Create architectural models to reason about relevant characteristics
- Critique the decision of where an AI model lives (e.g., cloud vs edge vs hybrid), considering the relevant tradeoffs
- Deliberate how and when to update models and how to collect telemetry

# SOFTWARE ARCHITECTURE



# SOFTWARE ARCHITECTURE



Focused on reasoning about tradeoffs and desired qualities

# SOFTWARE ARCHITECTURE

*The software architecture of a program or computing system is the **structure or structures** of the system, which comprise **software elements**, the **externally visible properties** of those elements, and the relationships among them.* -- [Kazman et al. 2012](#)

# WHY ARCHITECTURE? (KAZMAN ET AL. 2012)

- Represents earliest design decisions.
- Aids in **communication** with stakeholders
  - Shows them “how” at a level they can understand, raising questions about whether it meets their needs
- Defines **constraints** on implementation
  - Design decisions form “load-bearing walls” of application
- Dictates **organizational structure**
  - Teams work on different components
- Inhibits or enables **quality attributes**
  - Similar to design patterns
- Supports **predicting** cost, quality, and schedule
  - Typically by predicting information for each component
- Aids in software **evolution**
  - Reason about cost, design, and effect of changes
- Aids in **prototyping**
  - Can implement architectural skeleton early

# CASE STUDY: TWITTER



## Speaker notes

Source and additional reading: Raffi. [New Tweets per second record, and how!](#) Twitter Blog, 2013



# TWITTER - CACHING ARCHITECTURE



## Speaker notes

- Running one of the world's largest Ruby on Rails installations
- 200 engineers
- Monolithic: managing raw database, memcache, rendering the site, and \* presenting the public APIs in one codebase
- Increasingly difficult to understand system; organizationally challenging to manage and parallelize engineering teams
- Reached the limit of throughput on our storage systems (MySQL); read and write hot spots throughout our databases
- Throwing machines at the problem; low throughput per machine (CPU + RAM limit, network not saturated)
- Optimization corner: trading off code readability vs performance



# TWITTER'S REDESIGN GOALS

- Performance
  - Improve median latency; lower outliers
  - Reduce number of machines 10x
- Reliability
  - Isolate failures
- Maintainability
  - "We wanted cleaner boundaries with “related” logic being in one place": encapsulation and modularity at the systems level (rather than at the class, module, or package level)
- Modifiability
  - Quicker release of new features: "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams"

Raffi. [New Tweets per second record, and how!](#) Twitter Blog, 2013

# TWITTER: REDESIGN DECISIONS

- Ruby on Rails -> JVM/Scala
- Monolith -> Microservices
- RPC framework with monitoring, connection pooling, failover strategies, loadbalancing, ... built in
- New storage solution, temporal clustering, "roughly sortable ids"
- Data driven decision making



# TWITTER CASE STUDY: KEY INSIGHTS

- Architectural decisions affect entire systems, not only individual modules
- Abstract, different abstractions for different scenarios
- Reason about quality attributes early
- Make architectural decisions explicit
- Question: **Did the original architect make poor decisions?**

# ARCHITECTURAL MODELING AND REASONING





## Speaker notes

Map of Pittsburgh. Abstraction for navigation with cars.





## Speaker notes

Cycling map of Pittsburgh. Abstraction for navigation with bikes and walking.



# Fire Zones & Firehouses





## Speaker notes

Fire zones of Pittsburgh. Various use cases, e.g., for city planners.



# ANALYSIS-SPECIFIC ABSTRACTIONS

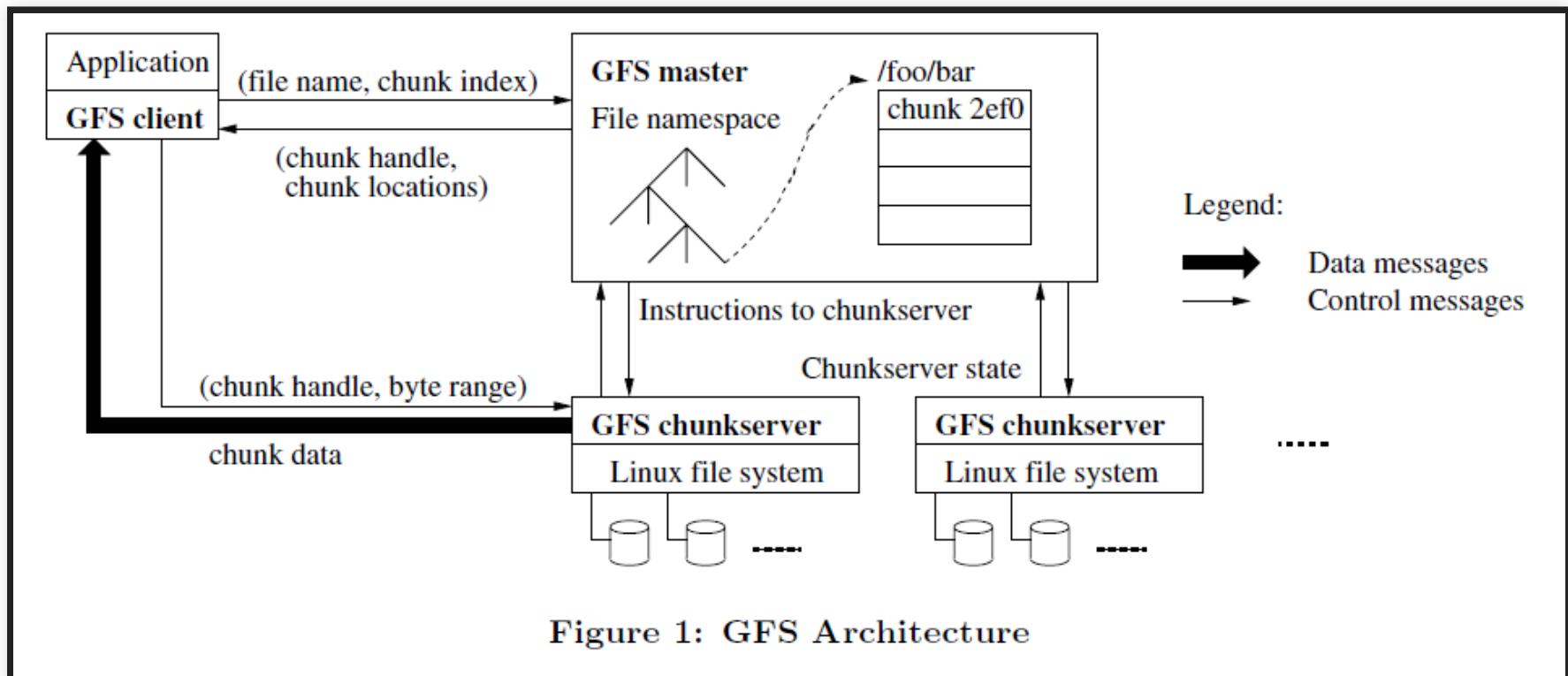
- All maps were abstractions of the same real-world construct
- All maps were created with different goals in mind
  - Different relevant abstractions
  - Different reasoning opportunities
- Architectural models are specific system abstractions, for reasoning about specific qualities
- No uniform notation



# WHAT CAN WE REASON ABOUT?



# WHAT CAN WE REASON ABOUT?



Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "[The Google file system.](#)" ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

## Speaker notes

Scalability through redundancy and replication; reliability wrt to single points of failure; performance on edges; cost



# MODELING RECOMMENDATIONS

- Use notation suitable for analysis
- Document meaning of boxes and edges in legend
- Graphical or textual both okay; whiteboard sketches often sufficient
- Formal notations available

# CASE STUDY: AUGMENTED REALITY TRANSLATION



## Speaker notes

Image: <https://pixabay.com/photos/nightlife-republic-of-korea-jongno-2162772/>



# CASE STUDY: AUGMENTED REALITY TRANSLATION



# CASE STUDY: AUGMENTED REALITY TRANSLATION



## Speaker notes

Consider you want to implement an instant translation service similar to Google translate, but run it on embedded hardware in glasses as an augmented reality service.



# QUALITIES OF INTEREST?



# ARCHITECTURAL DECISION: SELECTING AI TECHNIQUES

What AI techniques to use and why? Tradeoffs?



現金のみ



## Speaker notes

Relate back to previous lecture about AI technique tradeoffs, including for example Accuracy Capabilities (e.g. classification, recommendation, clustering...) Amount of training data needed Inference latency Learning latency; incremental learning? Model size Explainable? Robust?



# ARCHITECTURAL DECISION: WHERE SHOULD THE MODEL LIVE?

# WHERE SHOULD THE MODEL LIVE?

- Glasses
- Phone
- Cloud

What qualities are relevant for the decision?



Speaker notes

Trigger initial discussion



# CONSIDERATIONS

- How much data is needed as input for the model?
- How much output data is produced by the model?
- How fast/energy consuming is model execution?
- What latency is needed for the application?
- How big is the model? How often does it need to be updated?
- Cost of operating the model? (distribution + execution)
- Opportunities for telemetry?
- What happens if users are offline?

# **EXERCISE: LATENCY AND BANDWIDTH ANALYSIS OF AR TRANSLATION**



1. Identify key components of a solution and their interactions
2. Estimate latency and bandwidth requirements between components
3. Discuss tradeoffs among different deployment models



## Speaker notes

Identify at least OCR and Translation service as two AI components in a larger system. Discuss which system components are worth modeling (e.g., rendering, database, support forum). Discuss how to get good estimates for latency and bandwidth.



# WHEN WOULD ONE USE THE FOLLOWING DESIGNS?

- Static intelligence in the product
- Client-side intelligence
- Server-centric intelligence
- Back-end cached intelligence
- Hybrid models

## Speaker notes

From the reading:

- Static intelligence in the product
  - difficult to update
  - good execution latency
  - cheap operation
  - offline operation
  - no telemetry to evaluate and improve
- Client-side intelligence
  - updates costly/slow, out of sync problems
  - complexity in clients
  - offline operation, low execution latency
- Server-centric intelligence
  - latency in model execution (remote calls)
  - easy to update and experiment
  - operation cost
  - no offline operation
- Back-end cached intelligence
  - precomputed common results
  - fast execution, partial offline
  - saves bandwidth, complicated updates
- Hybrid models



# MORE CONSIDERATIONS

- Coupling of ML pipeline parts
- Coupling with other parts of the system
- Ability for different developers and analysts to collaborate
- Support online experiments
- Ability to monitor



# ARCHITECTURAL DECISION: TELEMETRY REQUIREMENTS



# TELEMETRY DESIGN

How to evaluate mistakes in production?





## Speaker notes

Discuss strategies to determine accuracy in production. What kind of telemetry needs to be collected?



# THE RIGHT AND RIGHT AMOUNT OF TELEMETRY

Purpose:

- Monitor operation
- Monitor mistakes (e.g., accuracy)
- Improve models over time (e.g., detect new features)

Challenges:

- too much data
- no/not enough data
- hard to measure, poor proxy measures
- rare events
- cost
- privacy



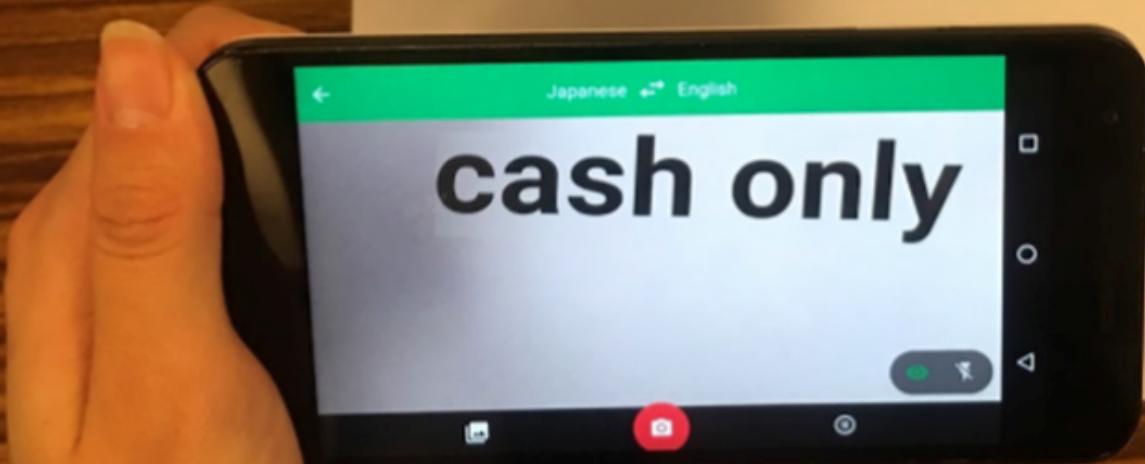
# TELEMETRY TRADEOFFS

What data to collect? How much? When?

Estimate data volume and possible bottlenecks in system.



現金のみ



## Speaker notes

Discuss alternatives and their tradeoffs. Draw models as suitable.



# RELATED: COST OF DATA AND FEATURE ENGINEERING

- How much data do we acquire for training and evaluating models?
- What data sources at what scale and latency (considering engineering cost, storage cost, processing cost, license cost, ...)
- Is it worth investing more time in feature engineering? What if additional data sources are needed?
- What is the cost for cleaning, preprocessing the data and the value of the additional accuracy?

# ARCHITECTURAL DECISION: INDEPENDENT MODEL SERVICE

Microservice architecture:

Model Inference and Model Learning as a RESTful Service?

# COUPLING AND CHANGEABILITY

What's the interface between the AI component and the rest of the system?

- Learning data and process
- Inference API
  - Where does feature extraction happen?
  - Provide raw data (images, user profile, all past purchases) to service, grant access to shared database, or provide feature vector?
  - Cost of feature extraction? Who bears the cost?
  - Versioned interface?
- Coupling to other models? Direct coupling to data sources (e.g., files, databases)? Expected formats for raw data (e.g., image resolution)?
- Coupling to telemetry?



# MODEL SERVICE API

Consider encapsulating the model as a microservice. Sketch a (REST) API.





# FUTURE-PROOFING AN API

- Anticipating and encapsulating change
  - What parts around the model service are likely to change?
  - Rigid vs flexible data formats?
- Versioning of APIs
  - Version numbers vs immutable services?
  - Expecting to run multiple versions in parallel? Implications for learning and evolution?

# ROBUSTNESS

- Redundancy for availability?
- Load balancer for scalability?
- Can mistakes be isolated?
  - Local error handling?
  - Telemetry to isolate errors to component?
- Logging and log analysis for what qualities?



# ARCHITECTURAL DECISION: UPDATING MODELS

- Design for change!
- Models are rarely static outside the lab
- Data drift, feedback loops, new features, new requirements
- When and how to update models?
- How to version? How to avoid mistakes?



# RISK OF STALE MODELS

What could happen if models become stale?



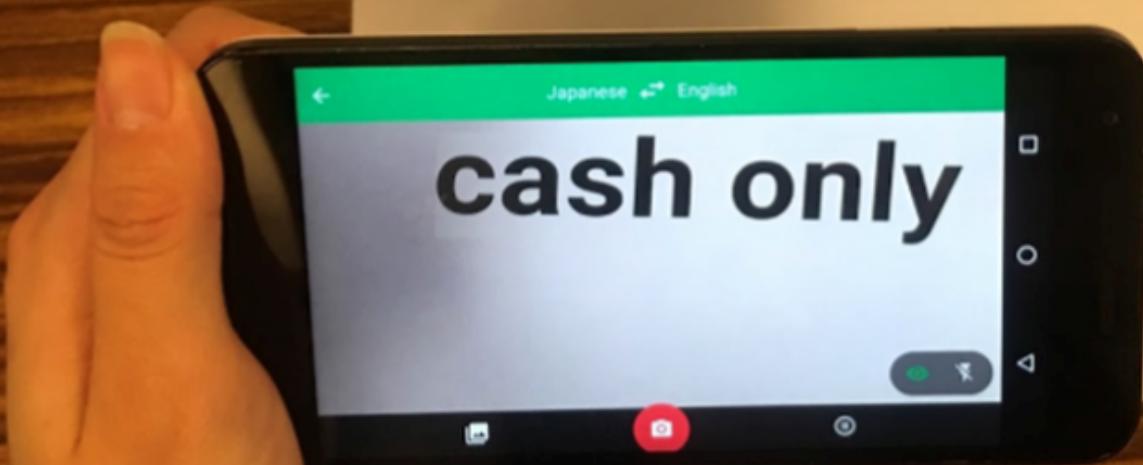
Risk: Discuss drift, adversarial interactions, feedback loops

# UPDATE REQUIREMENTS OR GOALS

Estimate the required update frequency and the related cost regarding training, data transfer, etc.



現金のみ



## Speaker notes

Discuss how frequently the involved models need to be updated. Are static models acceptable? Identify what information to collect and estimate the relevant values.



# OUTLOOK: BIG DATA DESIGNS

## Stream + Batch Processing

Carnegie Mellon University



- Latency and automation vary widely
- Heavily distributed



# **ARCHITECTURAL STYLES / TACTICS / DESIGN PATTERNS FOR AI ENABLED SYSTEMS**

(no standardization, yet)



# ARCHITECTURES AND PATTERNS

- The Big Ass Script Architecture
  - Decoupled multi-tiered architecture (data vs data analysis vs reporting; separate business logic from ML)
  - Microservice architecture (multiple learning and inference services)
  - Gateway Routing Architecture
- 
- Pipelines
  - Data lake, lambda architecture
  - Reuse between training and serving pipelines
  - Continuous deployment, ML versioning, pipeline testing
- 
- Daniel Smith. "[Exploring Development Patterns in Data Science](#)." TheoryLane Blog Post. 2017.
  - Washizaki, Hironori, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. "[Machine Learning Architecture and Design Patterns](#)." Draft, 2019



# ANTI-PATTERNS

- Big Ass Script Architecture
  - Dead Experimental Code Paths
  - Glue code
  - Multiple Language Smell
  - Pipeline Jungles
  - Plain-Old Datatype Smell
  - Undeclared Consumers
- 
- Washizaki, Hironori, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. "[Machine Learning Architecture and Design Patterns](#)." Draft, 2019
  - Sculley, David, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. "[Hidden technical debt in machine learning systems](#)." In Advances in neural information processing systems, pp. 2503-2511. 2015.

# AI AS A SERVICE

Third-Party AI Components in the Cloud

AI Components as Microservices



# READYMADE AI COMPONENTS IN THE CLOUD

- Data Infrastructure
  - Large scale data storage, databases, stream (MongoDB, Bigtable, Kafka)
- Data Processing
  - Massively parallel stream and batch processing (Sparks, Hadoop, ...)
  - Elastic containers, virtual machines (docker, AWS lambda, ...)
- AI Tools
  - Notebooks, IDEs, Visualization
  - Learning Libraries, Frameworks (tensorflow, torch, keras, ...)
- Models
  - Image, face, and speech recognition, translation
  - Chatbots, spell checking, text analytics
  - Recommendations, knowledge bases



# The Microsoft AI platform

Cloud-powered AI for every developer

## Azure AI Services

### PRE-BUILT AI

Cognitive Services

### CONVERSATIONAL AI

Bot Service

### CUSTOM AI

Azure Machine Learning

## Tools

### CODING & MANAGEMENT TOOLS

VS Tools  
for AI

Azure ML  
Studio

Azure ML  
Workbench

Others (PyCharm, Jupyter Notebooks...)

## Azure Infrastructure

### AI ON DATA

Cosmos  
DB

SQL  
DB

SQL  
DW

Data  
Lake

### AI COMPUTE

Spark

DSVM

Batch  
AI

ACS

IoT  
Edge

CPU, FPGA, GPU

3rd Party

Cognitive  
Toolkit

TensorFlow

Caffe

Others (Scikit-learn, MXNet, Keras,  
Chainer, Gluon...)

# BUILD VS BUY

Hardware, software, models?



Speaker notes

Discuss privacy implications



# REFLECTION

Qualities of interest? Important design tradeoffs? Decisions?



現金のみ



# SUMMARY

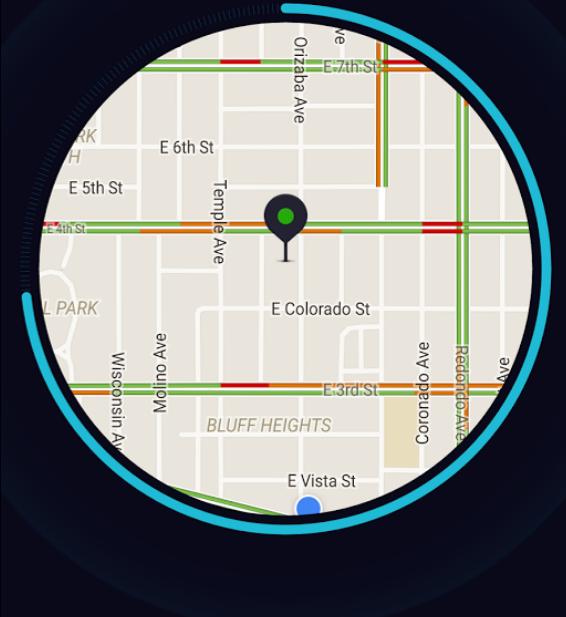
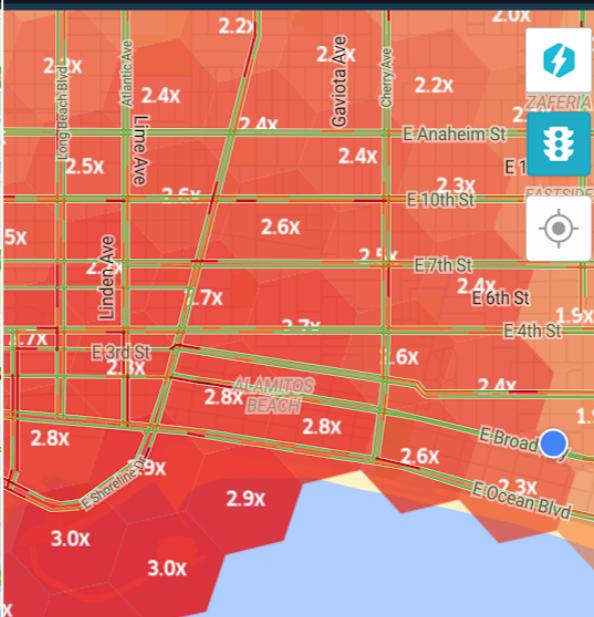
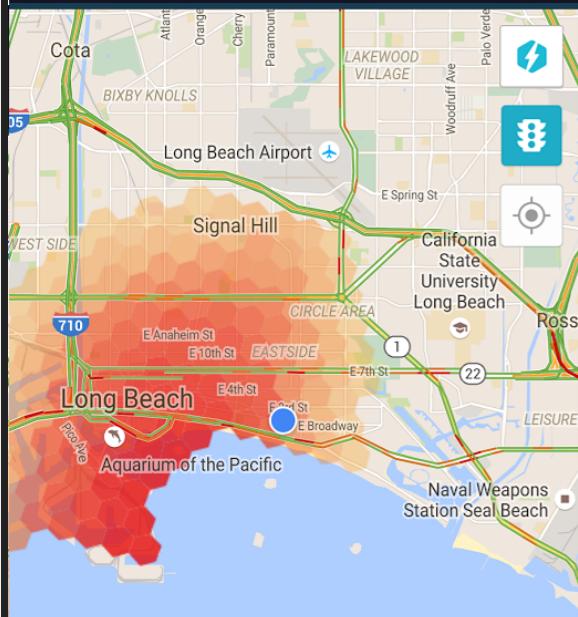
- Software architecture is an established discipline to reason about design alternatives
- Understand relevant quality goals
- Problem-specific modeling and analysis: Gather estimates, consider design alternatives, make tradeoffs explicit
- Examples of important design decision:
  - modeling technique to use
  - where to deploy the model
  - how and how much telemetry to collect
  - whether and how to modularize the model service
  - when and how to update models
  - build vs buy, cloud resources

# CASE STUDY 2: UBER SURGE PREDICTION



GO OFFLINE

GO OFFLINE



## Speaker notes

Consider you work at Uber and want to predict where rider demand is going to be high.



# QUALITIES OF INTEREST?

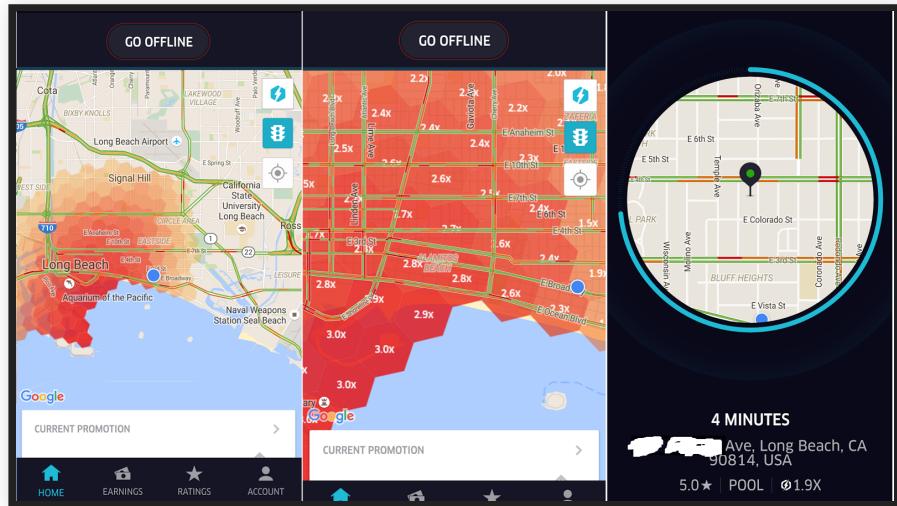




# WHERE SHOULD THE MODEL LIVE?

- Car
- Phone
- Cloud

What qualities are relevant for the decision?



Speaker notes

Trigger initial discussion



# TELEMETRY DESIGN

How to evaluate mistakes in production?



