DEPLOYING A MODEL

(Introduction to 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.

Recommended reading: Rick Kazman, Paul Clements, and Len Bass. Software architecture in practice. Addison-Wesley Professional, 2012, Chapter 1

LEARNING GOALS

- Understand important quality considerations when deploying ML components
- Follow a design process to explicitly reason about alternative designs and their quality tradeoffs
- Gather data to make informed decisions about what ML technique to use and where and how to deploy it
- Understand the power of design patterns for codifying design knowledge

- 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
- Deploy models locally and to the cloud
- Document model inference services

DEPLOYING A MODEL IS EASY

DEPLOYING A MODEL IS EASY

Model inference component as function/library

```
from sklearn.linear_model import LogisticRegression
model = … # learn model or load serialized model ...
def infer(feature1, feature2):
    return model.predict(np.array([[feature1, feature2]])
```

DEPLOYING A MODEL IS EASY

Model inference component as a service

```
from flask import Flask, escape, request
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = '/tmp/uploads'
detector_model = … # load model…

# inference API that returns JSON with classes
# found in an image
@app.route('/get_objects', methods=['POST'])
def pred():
    uploaded_img = request.files["images"]
    coverted_img = … # feature encoding of uploaded img
    result = detector_model(converted_img)
    return jsonify({"response":
                result['detection_class_entities']})
```

DEPLOYING A MODEL IS EASY

Packaging a model inference service in a container

```
FROM python:3.8-buster
RUN pip install uwsgi==2.0.20
RUN pip install numpy==1.22.0
RUN pip install tensorflow==2.7.0
RUN pip install flask==2.0.2
RUN pip install gunicorn==20.1.0
COPY models/model.pf /model/
COPY ./serve.py /app/main.py
WORKDIR ./app
EXPOSE 4040
CMD ["gunicorn", "-b 0.0.0.0:4040", "main:app"]
```

DEPLOYING A MODEL IS EASY

Model inference component as a service in the cloud

- Package in container or other infrastructure
- Deploy in cloud infrastructure
- Auto-scaling with demand
- MLOps infrastructure to automate all of this
- (more on this later)

- Model inference is stateless and embarrassingly parallel
- Almost always deterministic
- "*Stateless Serving Functions Pattern*"
- Lots of tooling available, including
    - BentoML (low code service creation, deployment, model registry),
    - Cortex (automated deployment and scaling of models on AWS),
    - TFX model serving (tensorflow GRPC services)
    - Seldon Core (no-code model service and many many additional services for monitoring and operations on Kubernetes)

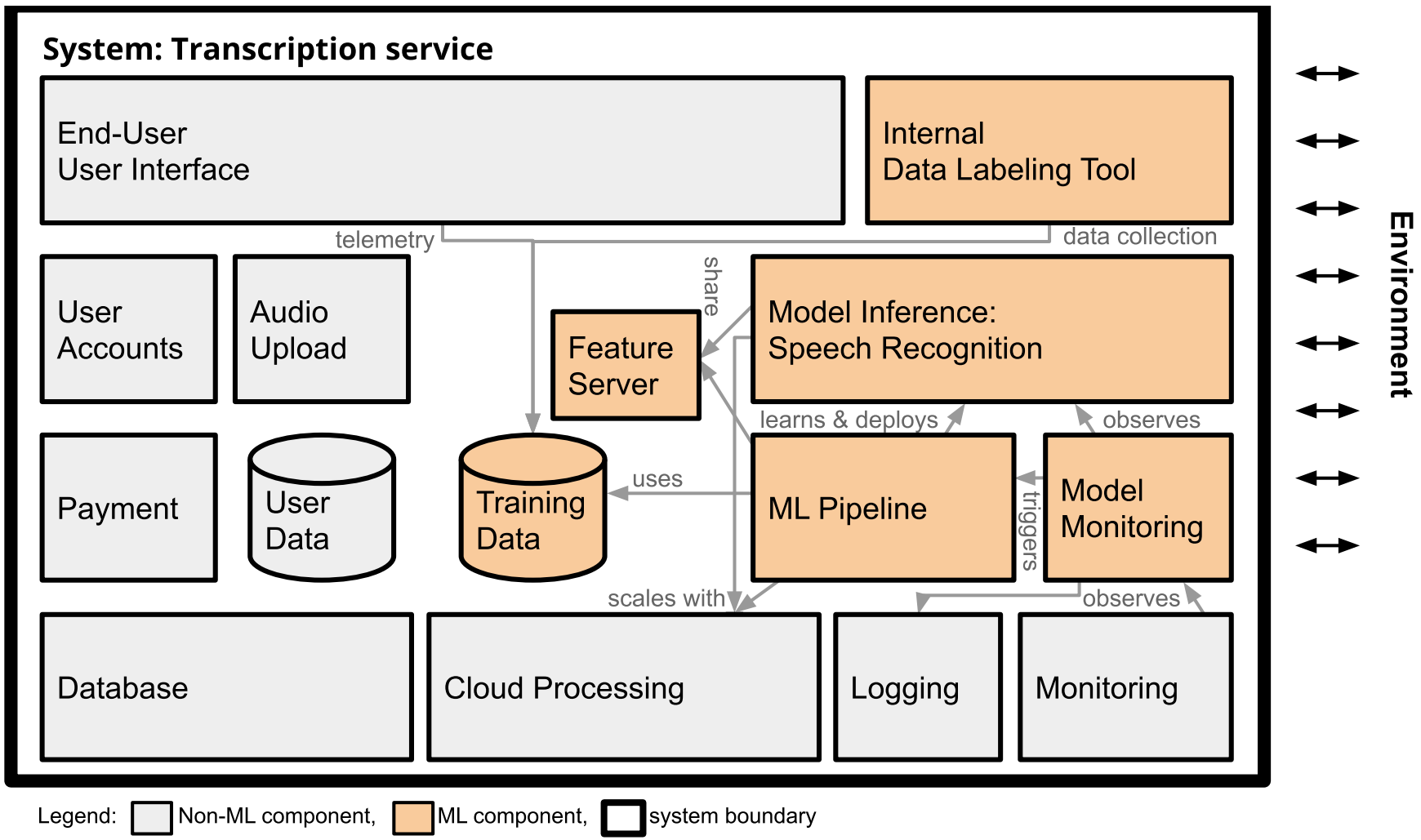
BUT IS IT REALLY EASY?

- Offline use?
- Deployment at scale?
- Hardware needs and operating cost?
- Frequent updates?
- Integration of the model into a system?
- Meeting system requirements?
- Every system is different!

EVERY SYSTEM IS DIFFERENT

- Personalized music recommendations for Spotify
- Transcription service startup
- Self-driving car
- Smart keyboard for mobile device

INFERENCE IS A COMPONENT WITHIN A SYSTEM



SOFTWARE ARCHITECTURE

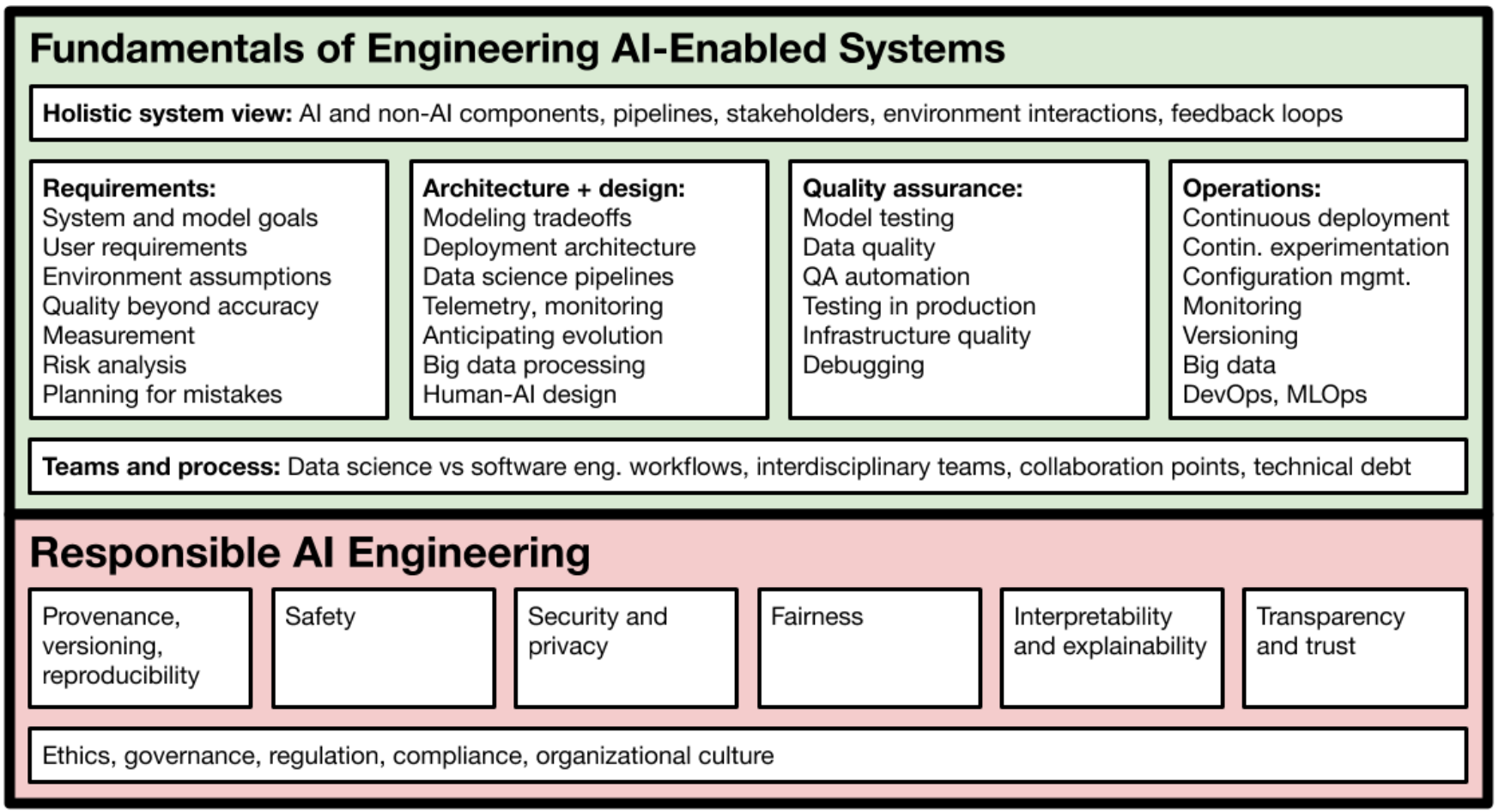
SO FAR: REQUIREMENTS

- Identify goals for the system, define success metrics
- Understand requirements, specifications, and assumptions
- Consider risks, plan for mitigations to mistakes
- Approaching component requirements: Understand quality requirements and constraints for models and learning algorithms

SOFTWARE ARCHITECTURE

Focused on reasoning about tradeoffs and desired qualities

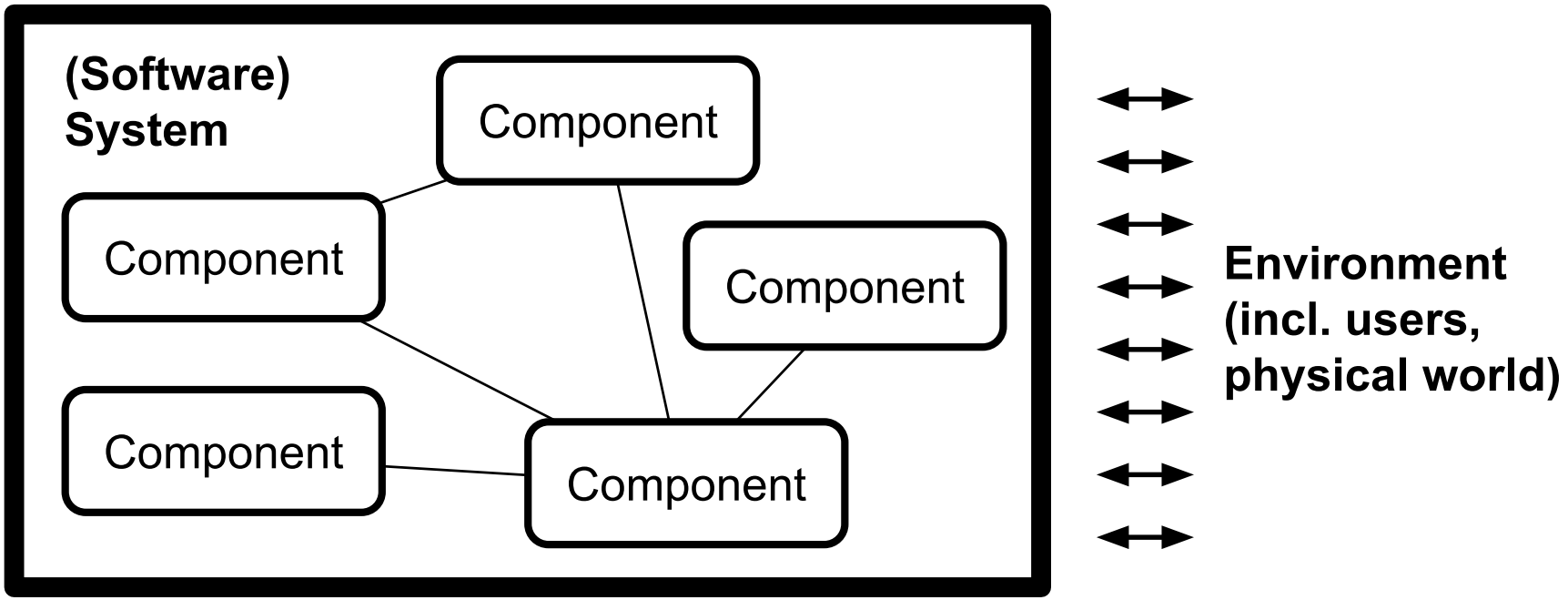
FROM REQUIREMENTS TO DESIGN/ARCHITECTURE



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*

RECALL: SYSTEMS THINKING

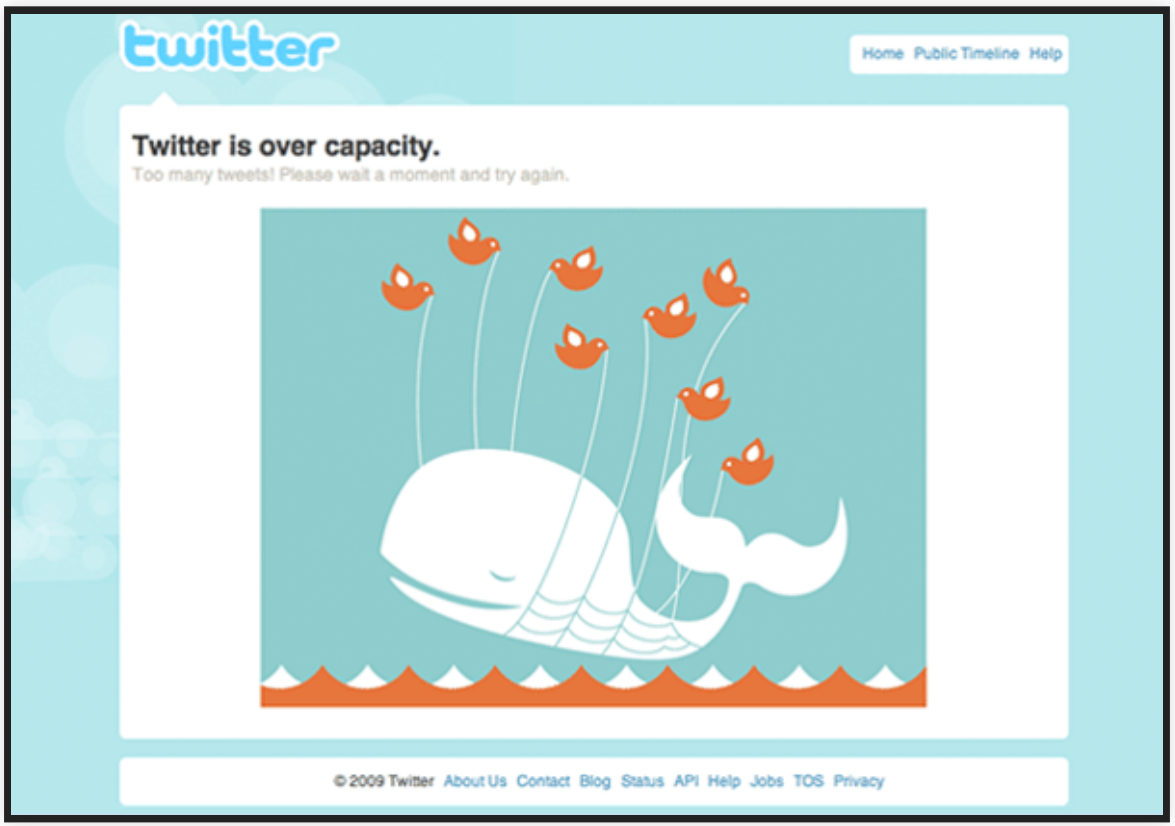


A system is a set of inter-related components that work together in a particular environment to perform whatever functions are required to achieve the system's objective -- Donella Meadows

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

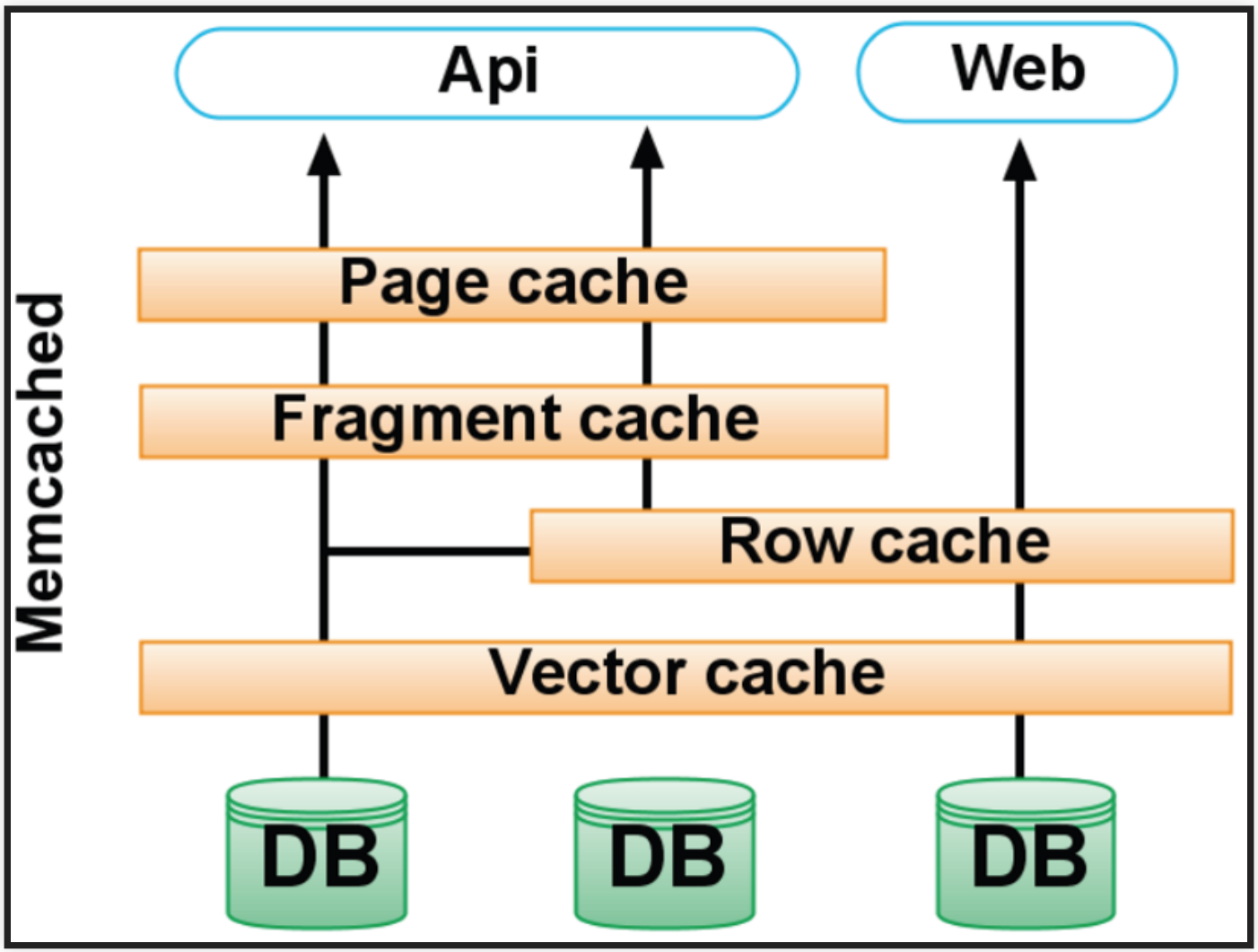
CASE STUDY: TWITTER



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

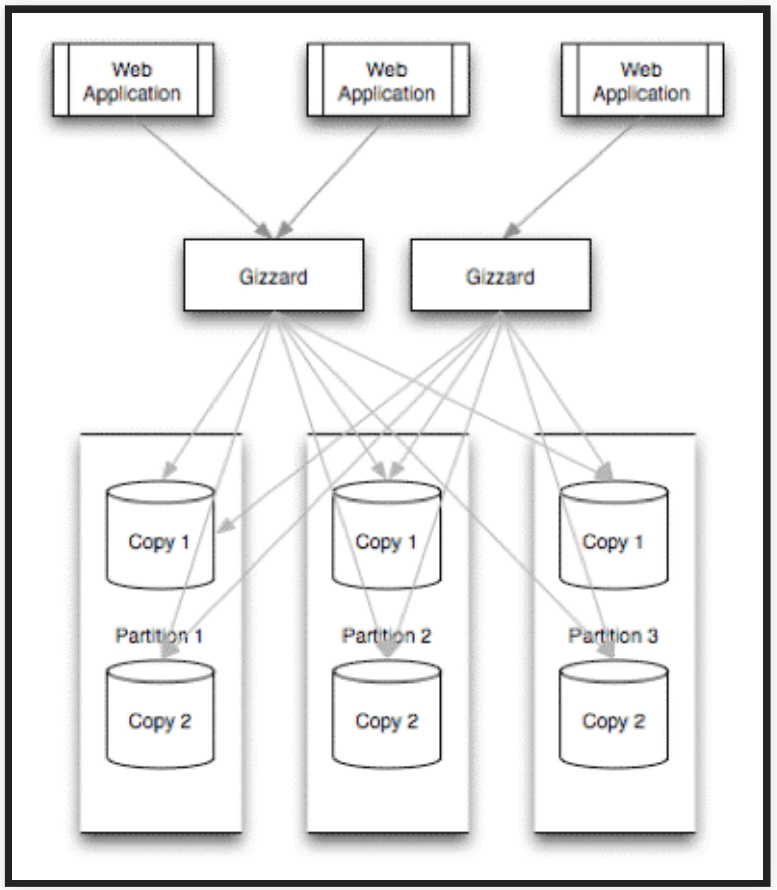
TWITTER'S REDESIGN GOALS

- 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

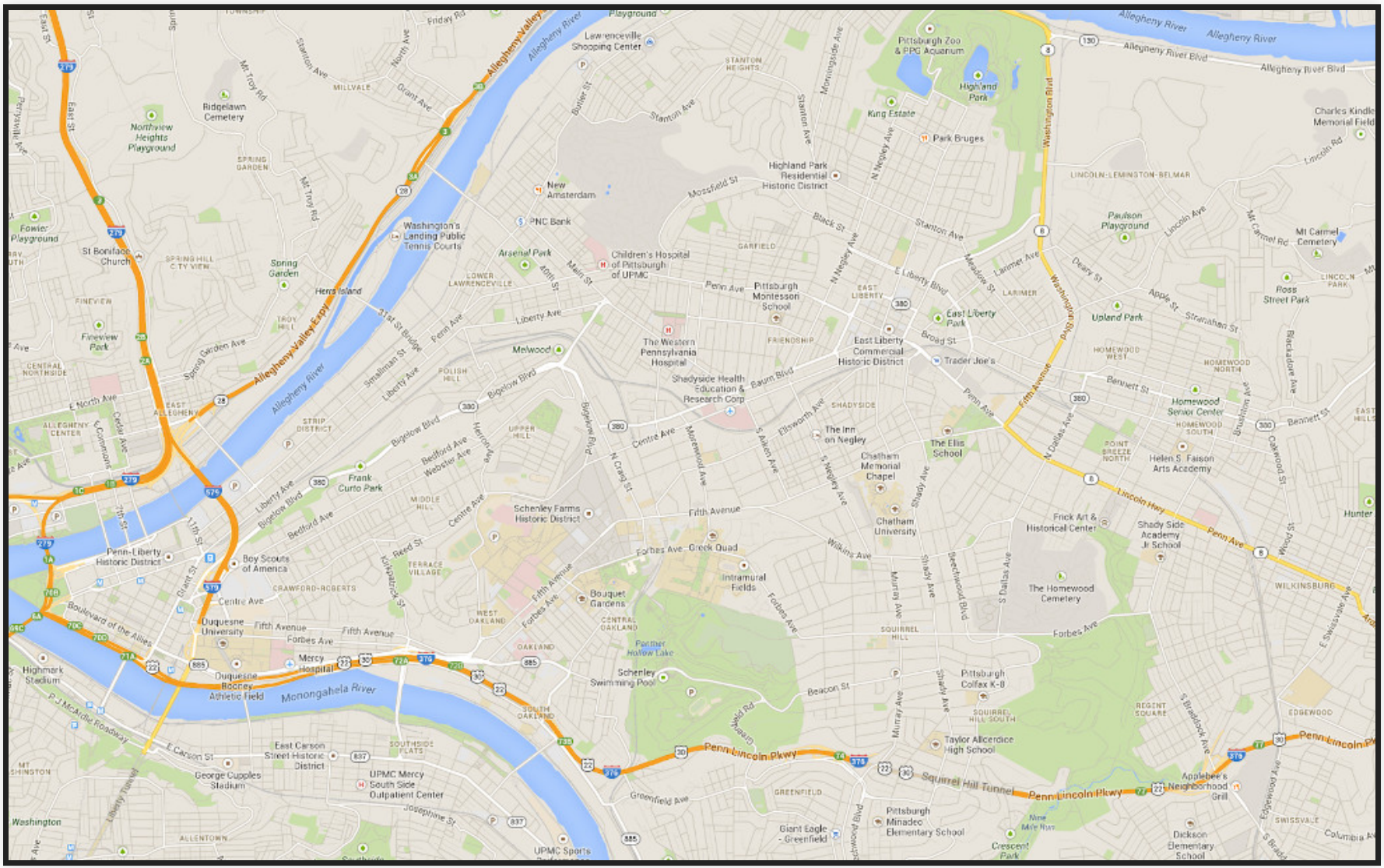
- Ruby on Rails -> JVM/Scala
- Monolith -> Microservices
- RPC framework with monitoring, connection pooling, failover strategies, loadbalancing, ... built in
- New storage solution, temporal clustering, "roughly sortable ids"
- Data driven decision making



TWITTER CASE STUDY: KEY INSIGHTS

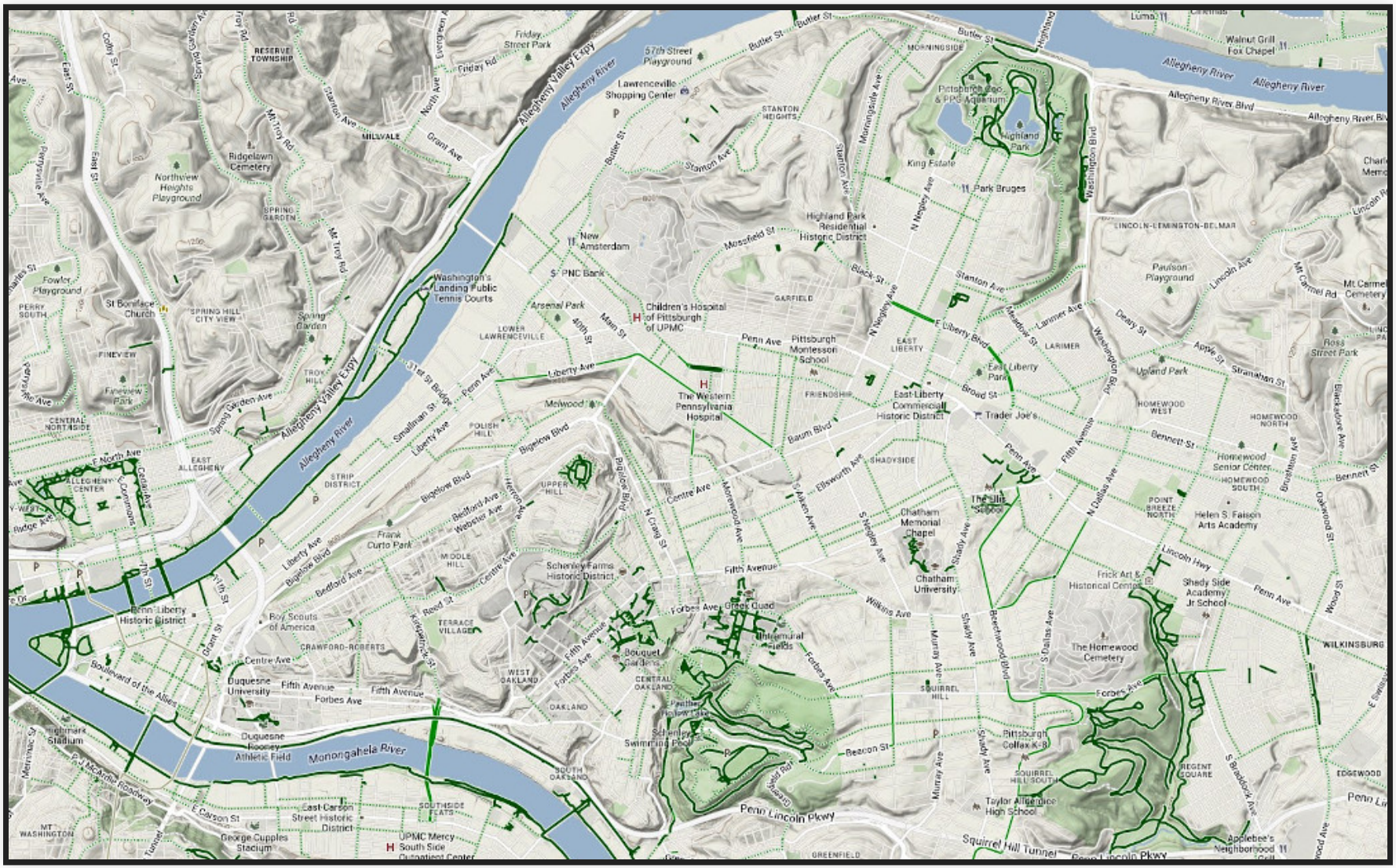
- Architectural decisions affect entire systems, not only individual modules
- Abstract, different abstractions for different scenarios
- Reason about quality attributes early
- Make architectural decisions explicit
- Question: **Did the original architect make poor decisions?**

ARCHITECTURAL MODELING AND REASONING


Allegheny River
Allegheny River Blvd
Pittsburgh Zoo & PPG Aquarium
Highland Park
King Estate
Park Bruges
Highland Park Residential Historic District
Charles Kindle Memorial Field
Lawrenceville Shopping Center
Northview Heights Playground
Ridgelawn Cemetery
Washington's Landing Public Tennis Courts
New Amsterdam
PNC Bank
Arsenal Park
Children's Hospital of Pittsburgh of UPMC
Pittsburgh Montessori School
Paulson Playground
Mt Carmel Cemetery
Ross Street Park
Upland Park
East Liberty Park
East Liberty Commercial Historic District
Trader Joe's
Homewood Senior Center
Helen S. Faison Arts Academy
Fowler Playground
St Boniface Church
Spring Garden
Herrs Island
Fineview Park
Allegheny Valley Expy
Melwood
The Western Pennsylvania Hospital
Shadyside Health Education & Research Corp
The Inn on Negley
Chatham Memorial Chapel
The Ellis School
Chatham University
Frick Art & Historical Center
Shady Side Academy Jr School
Schenley Farms Historic District
Frank Curto Park
Penn-Liberty Historic District
Boy Scouts of America
Bouquet Gardens
Intramural Fields
Forbes Ave - Greek Quad
The Homewood Cemetery
Duquesne University
Mercy Hospital
Duquesne Bayer Athletic Field
Monongahela River
Schenley Swimming Pool
Pittsburgh Colfax K-8
Taylor Allderdice High School
Penn-Lincoln Pkwy
Squirrel Hill Tunnel
Applebee's Neighborhood Grill
George Cupples Stadium
East Carson Street Historic District
UPMC Mercy South Side Outpatient Center
Giant Eagle - Greenfield
Pittsburgh Minadeo Elementary School
Dickson Elementary School
Crescent Park
Highmark Stadium
Washington
UPMC Sports
Lincoln Ave
Penn Ave
Fifth Avenue
Forbes Ave
Centre Ave
Liberty Ave
Bigelow Blvd
Beacon St
Greenfield Ave

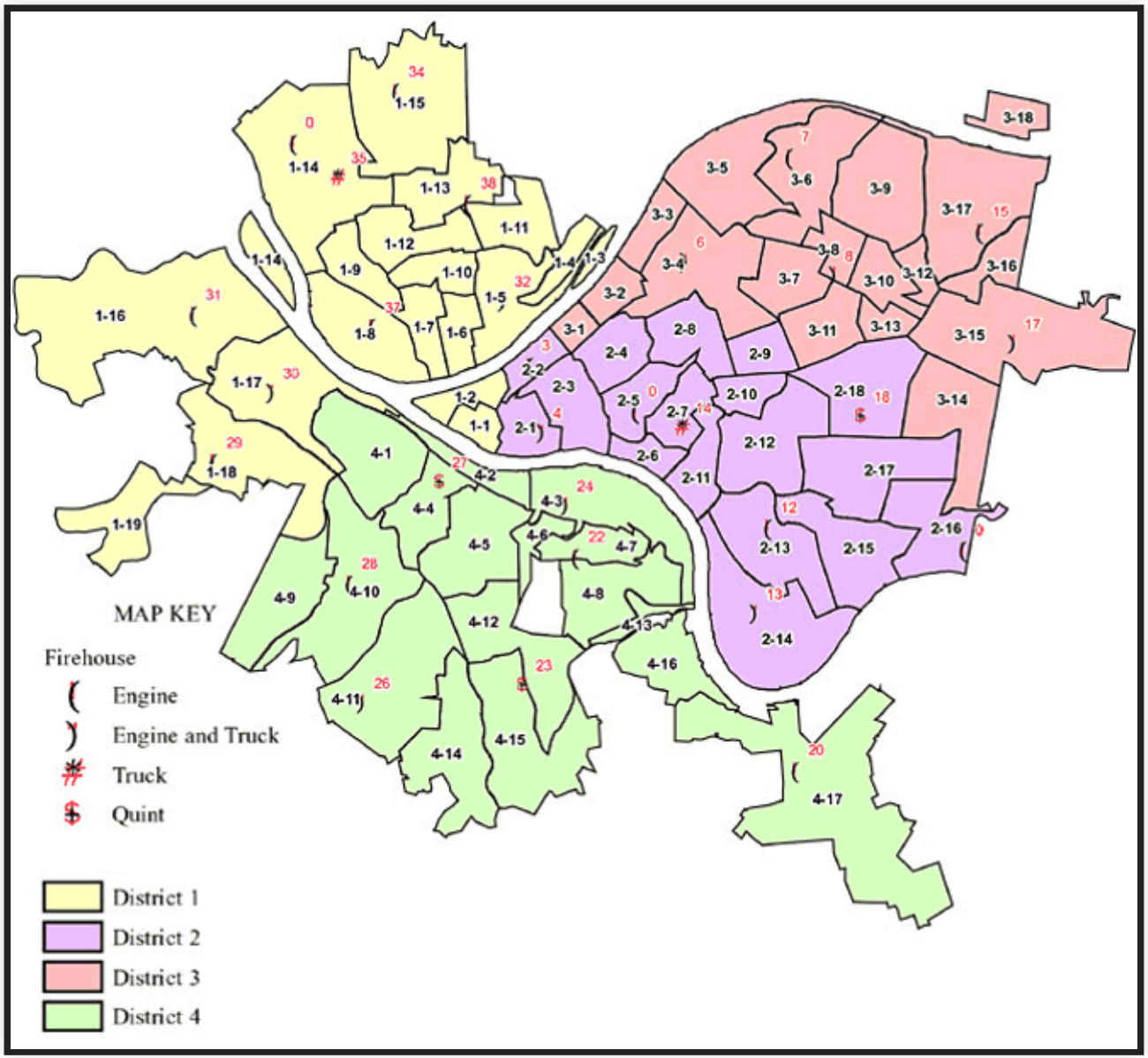
Speaker notes

Map of Pittsburgh. Abstraction for navigation with cars.


Allegheny River
Monongahela River
Allegheny Valley Expy
Penn Lincoln Pkwy
Fifth Avenue
Forbes Ave
Liberty Ave
Penn Ave
Bigelow Blvd
Centre Ave
Butler St
Stanton Ave
Baum Blvd
Walnut Grill Fox Chapel
Lawrenceville Shopping Center
Pittsburgh Zoo & PPG Aquarium
Highland Park
King Estate
Park Bruges
Highland Park Residential Historic District
New Amsterdam
PNC Bank
Washington's Landing Public Tennis Courts
Children's Hospital of Pittsburgh of UPMC
Arsenal Park
Pittsburgh Montessori School
East Liberty Park
East Liberty Commercial Historic District
Trader Joe's
The Western Pennsylvania Hospital
Paulson Playground
Upland Park
Ross Street Park
Mt Carmel Cemetery
Homewood Senior Center
Helen S. Faison Arts Academy
The Ellis School
Chatham Memorial Chapel
Chatham University
Frick Art & Historical Center
Shady Side Academy Jr School
The Homewood Cemetery
Pittsburgh Colfax K-8
Taylor Allderdice High School
Applebee's Neighborhood Grill
Squirrel Hill Tunnel
Schenley Farms Historic District
Frank Curto Park
Bouquet Gardens
Schenley Plaza
Panther Hollow Lake
Schenley Swimming Pool
Duquesne University
Duquesne Rooney Athletic Field
Heinz Field
George Cupples Stadium
East Carson Street Historic District
UPMC Mercy South Side Outpatient Center
Boy Scouts of America
Penn Liberty Historic District
Allegheny Center
Fineview Park
Fowler Playground
St Boniface Church
Northview Heights Playground
Ridgelawn Cemetery
Reserve Township
Millvale
Spring Garden
Spring Hill City View
Troy Hill
Central Northside
East Allegheny
Strip District
Polish Hill
Upper Hill
Middle Hill
Terrace Village
Crawford-Roberts
West Oakland
Central Oakland
Oakland
South Oakland
Southside Flats
Mt Washington
Greenfield
Squirrel Hill
Squirrel Hill South
Regent Square
Edgewood
Wilkinsburg
Point Breeze North
Homewood West
Homewood North
Homewood South
Lincoln-Lemington-Belmar
Garfield
Lower Lawrenceville
Friendship
Shadyside
Larimer
East Liberty
Morningside
Stanton Heights
Milwaukee
Friday Street Park
57th Street Playground

Speaker notes

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


MAP KEY
Firehouse
Engine
Engine and Truck
Truck
Quint
District 1
District 2
District 3
District 4
1-1
1-2
1-3
1-4
1-5
1-6
1-7
1-8
1-9
1-10
1-11
1-12
1-13
1-14
1-15
1-16
1-17
1-18
1-19
2-1
2-2
2-3
2-4
2-5
2-6
2-7
2-8
2-9
2-10
2-11
2-12
2-13
2-14
2-15
2-16
2-17
2-18
3-1
3-2
3-3
3-4
3-5
3-6
3-7
3-8
3-9
3-10
3-11
3-12
3-13
3-14
3-15
3-16
3-17
3-18
4-1
4-2
4-3
4-4
4-5
4-6
4-7
4-8
4-9
4-10
4-11
4-12
4-13
4-14
4-15
4-16
4-17

Speaker notes

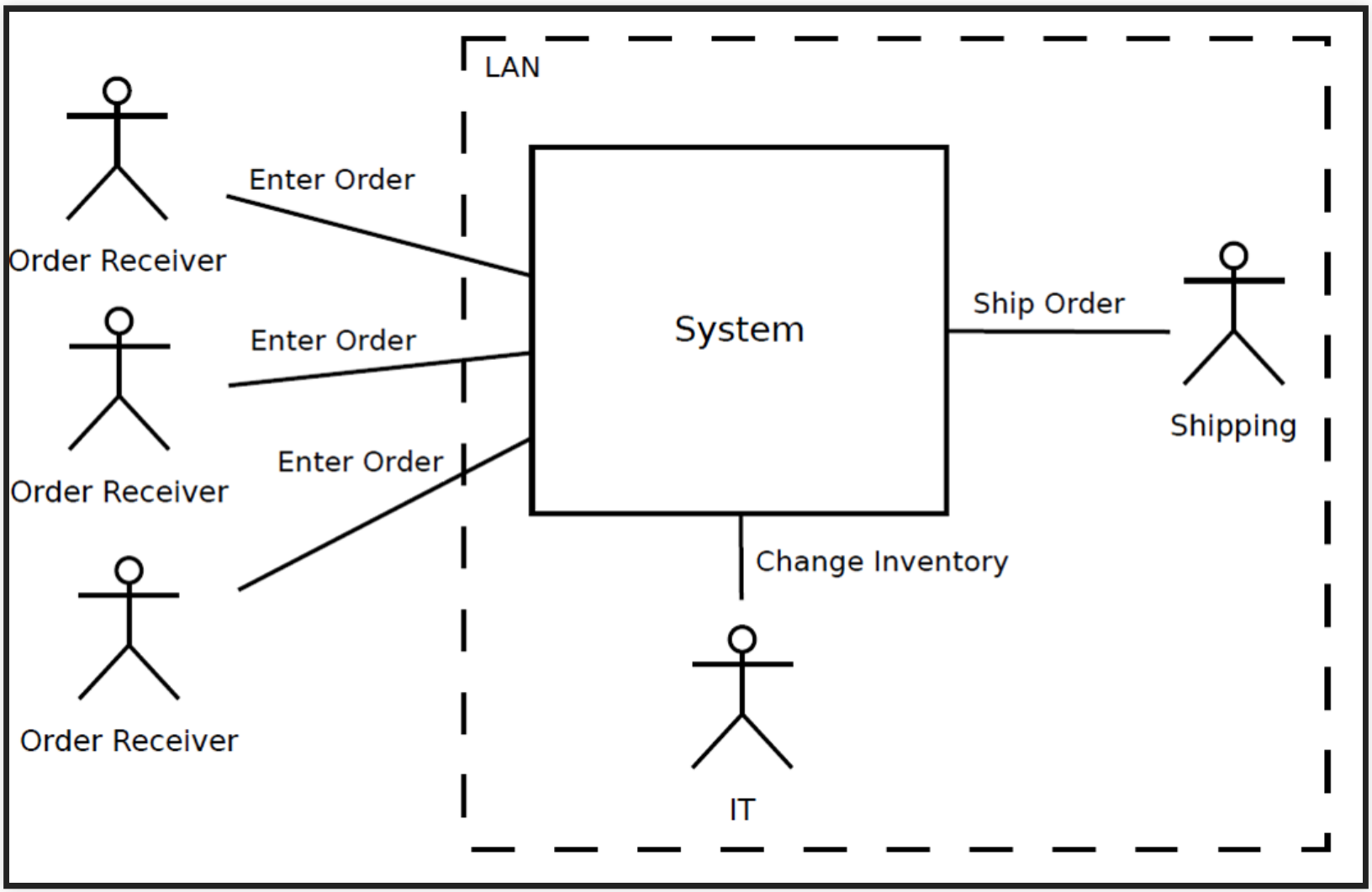
Fire zones of Pittsburgh. Various use cases, e.g., for city planners.

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?

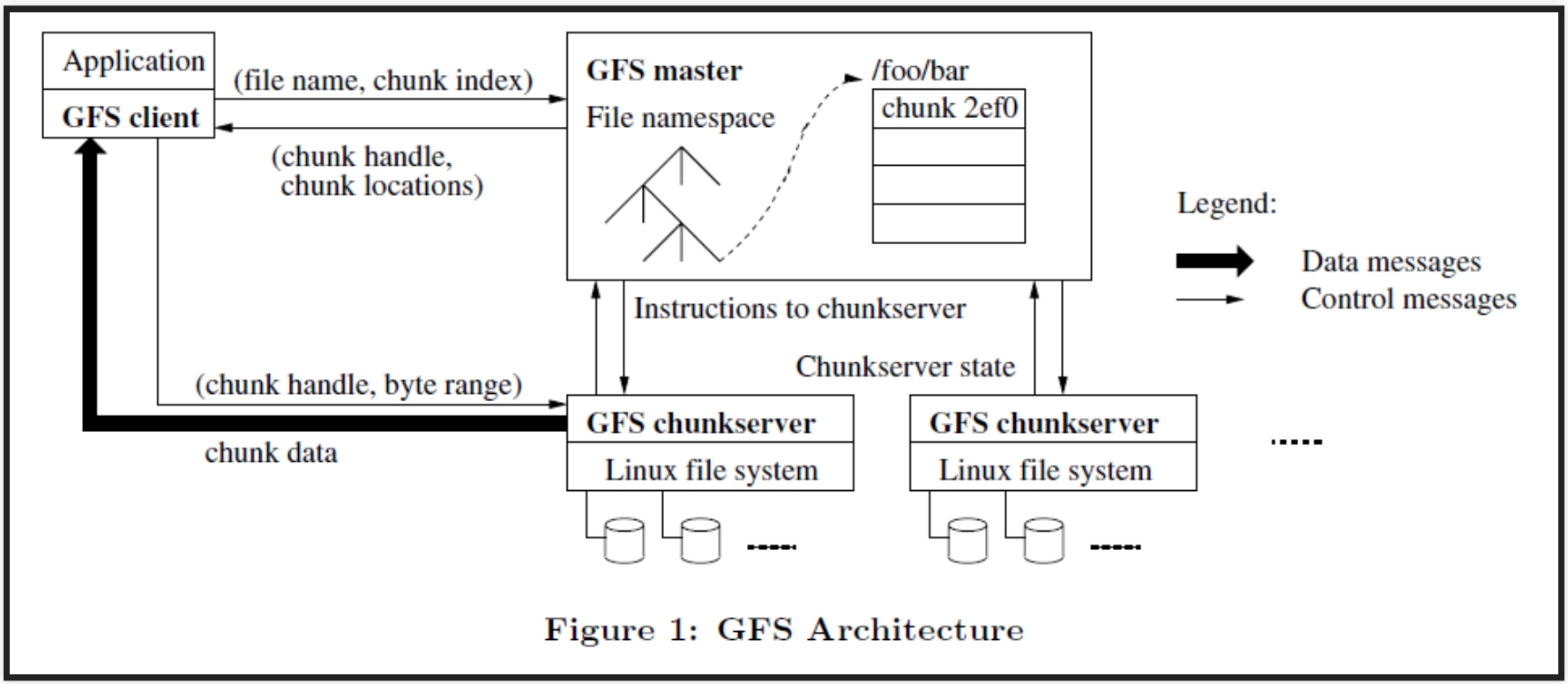


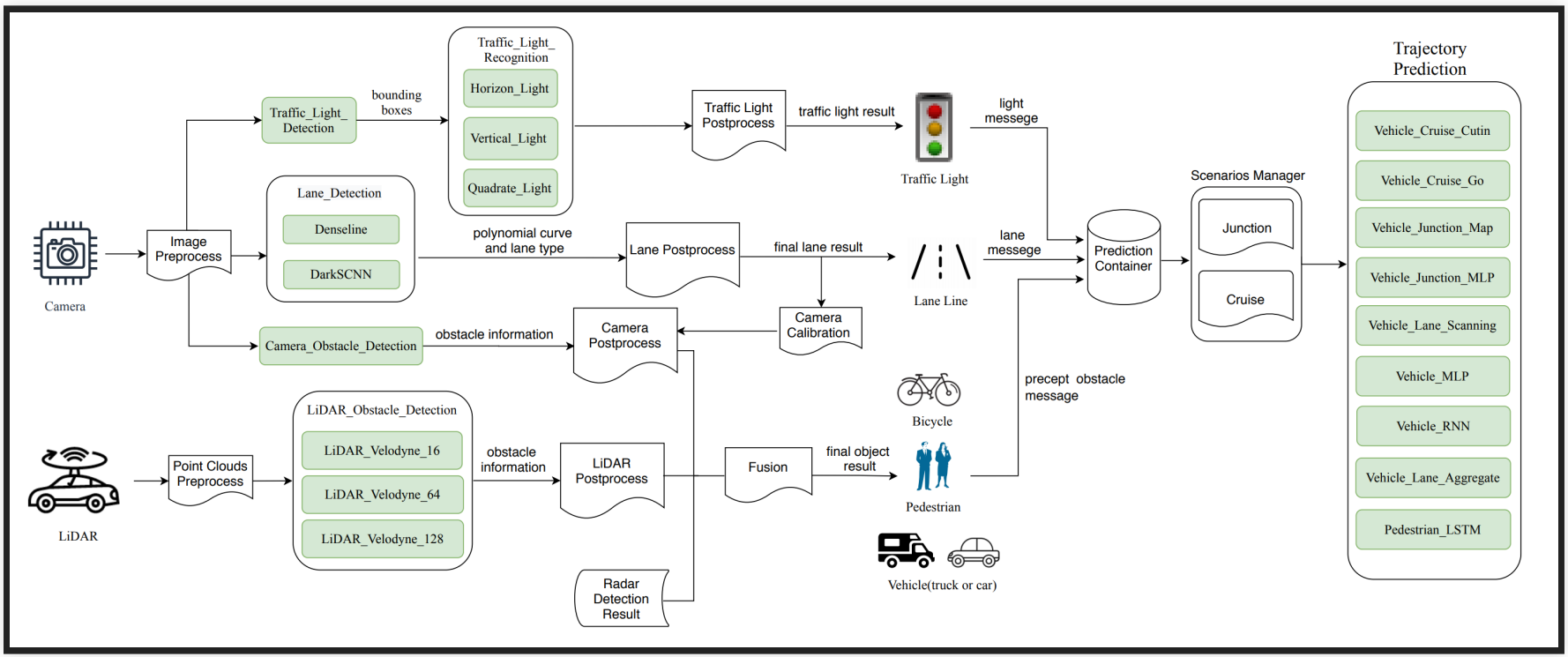
Figure 1: GFS Architecture

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

WHAT CAN WE REASON ABOUT?



Peng, Zi, Jinqiu Yang, Tse-Hsun Chen, and Lei Ma. "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on Apollo." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1240-1250. 2020.

SUGGESTIONS FOR GRAPHICAL NOTATIONS

- 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 toGoogle translate, but run it on embedded hardware in glasses as an augmented reality service.

SYSTEM QUALITIES OF INTEREST?

DESIGN DECISION: SELECTING ML ALGORITHMS

What ML algorithms to use and why? Tradeoffs?


現金のみ
Japanese
English
cash only

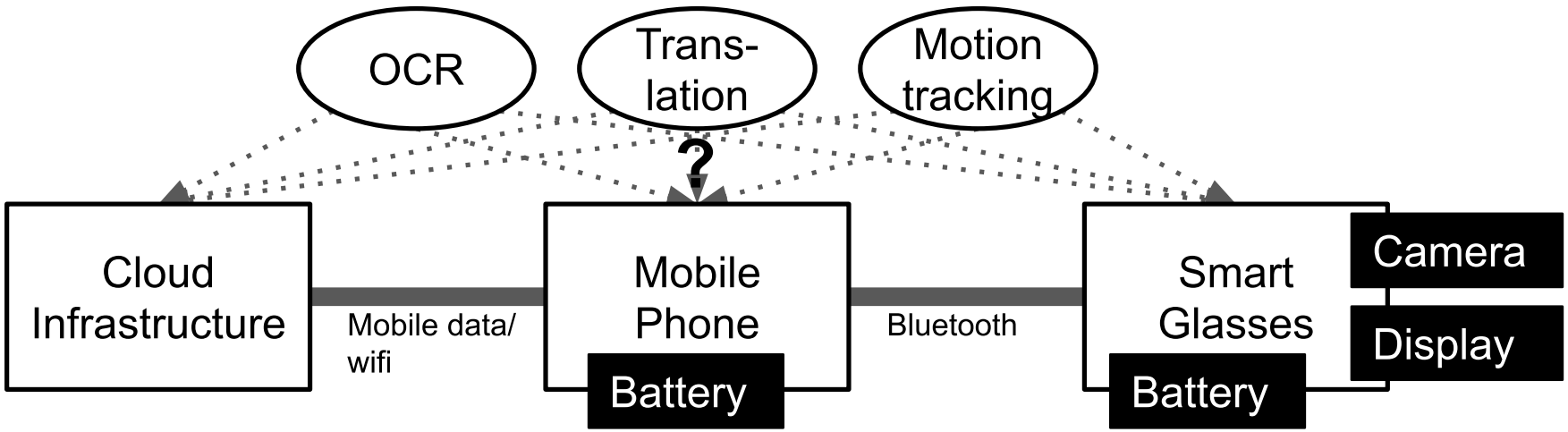
Speaker notes

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?

DESIGN DECISION: WHERE SHOULD THE MODEL LIVE?

(Deployment Architecture)

WHERE SHOULD THE MODELS LIVE?



Cloud? Phone? Glasses?

What qualities are relevant for the decision?

Speaker notes

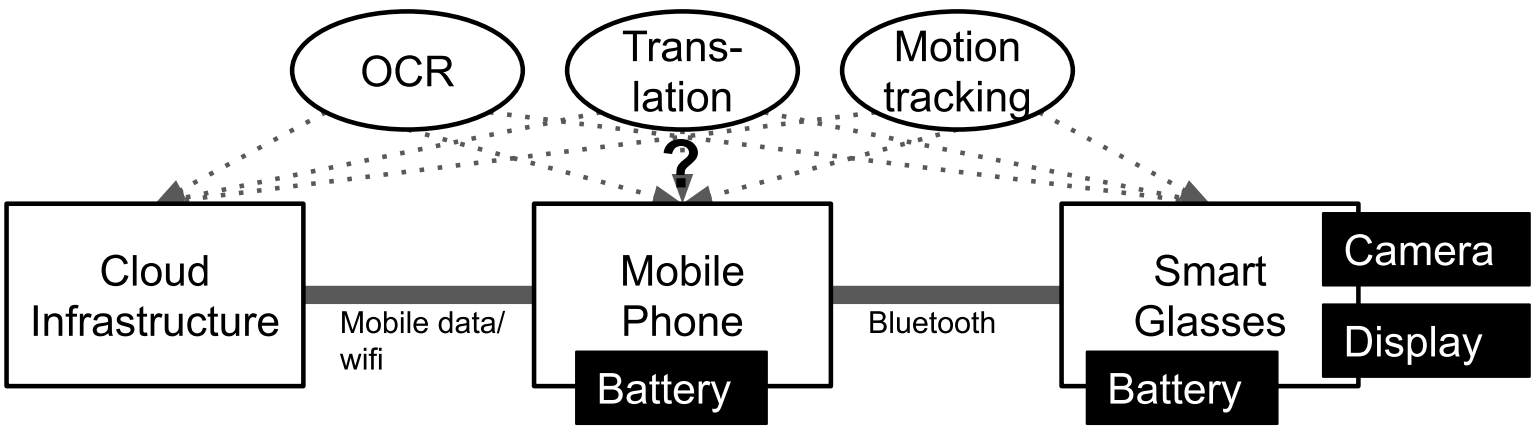
Trigger initial discussion

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?

BREAKOUT: LATENCY AND BANDWIDTH ANALYSIS OF AR TRANSLATION

1. Estimate latency and bandwidth requirements between components
2. Discuss tradeoffs among different deployment models



Post on Slack in `#lecture`:

- Recommended deployment for OCR (with justification):
- Recommended deployment for Translation (with justification):

Speaker notes

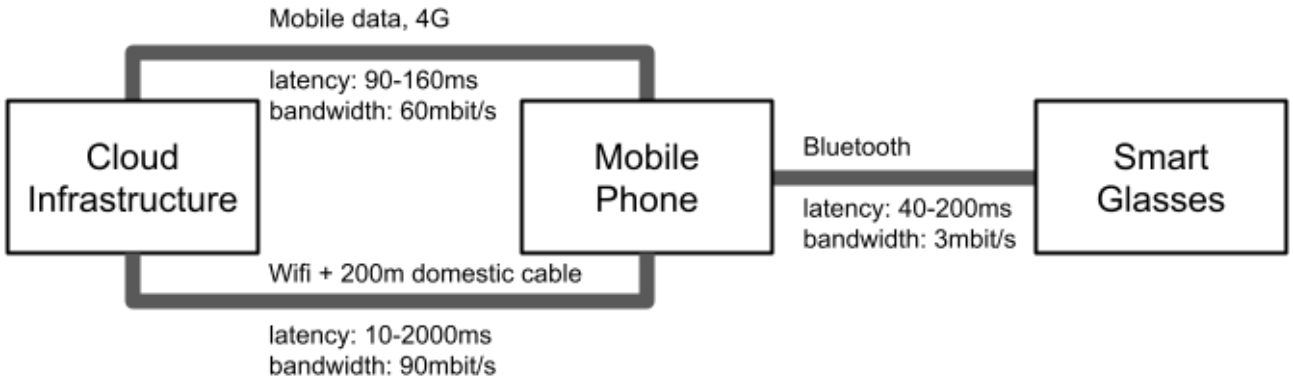
Identify at least OCR and Translation service as two AI components in a larger system. Discuss which system components are worth modeling (e.g., rendering, database, support forum). Discuss how to get good estimates for latency and bandwidth.

Some data: 200ms latency is noticable as speech pause; 20ms is perceivable as video delay, 10ms as haptic delay; 5ms referenced as cybersickness threshold for virtual reality 20ms latency might be acceptable

bluetooth latency around 40ms to 200ms

bluetooth bandwidth up to 3mbit, wifi 54mbit, video stream depending on quality 4 to 10mbit for low to medium quality

google glasses had 5 megapixel camera, 640x360 pixel screen, 1 or 2gb ram, 16gb storage


Mobile data, 4G
latency: 90-160ms
bandwidth: 60mbit/s
Cloud Infrastructure
Mobile Phone
Bluetooth
latency: 40-200ms
bandwidth: 3mbit/s
Smart Glasses
Wifi + 200m domestic cable
latency: 10-2000ms
bandwidth: 90mbit/s

FROM THE READING: WHEN WOULD ONE USE THE FOLLOWING DESIGNS?

- Static intelligence in the product
- Client-side intelligence (user-facing devices)
- Server-centric intelligence
- Back-end cached intelligence
- Hybrid models

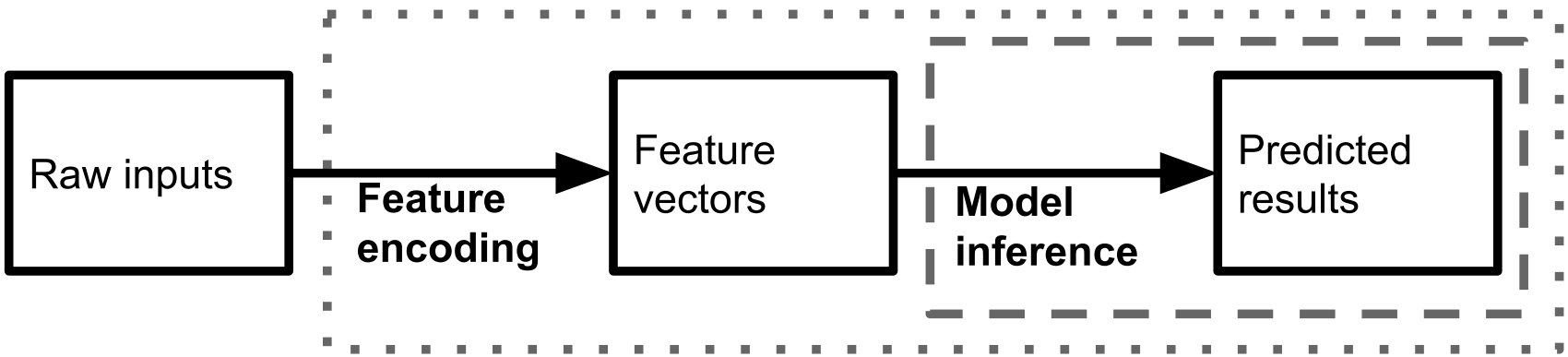
- Consider: Offline use, inference latency, model updates, application updates, operating cost, scalability, protecting intellectual property

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

WHERE SHOULD FEATURE ENCODING HAPPEN?

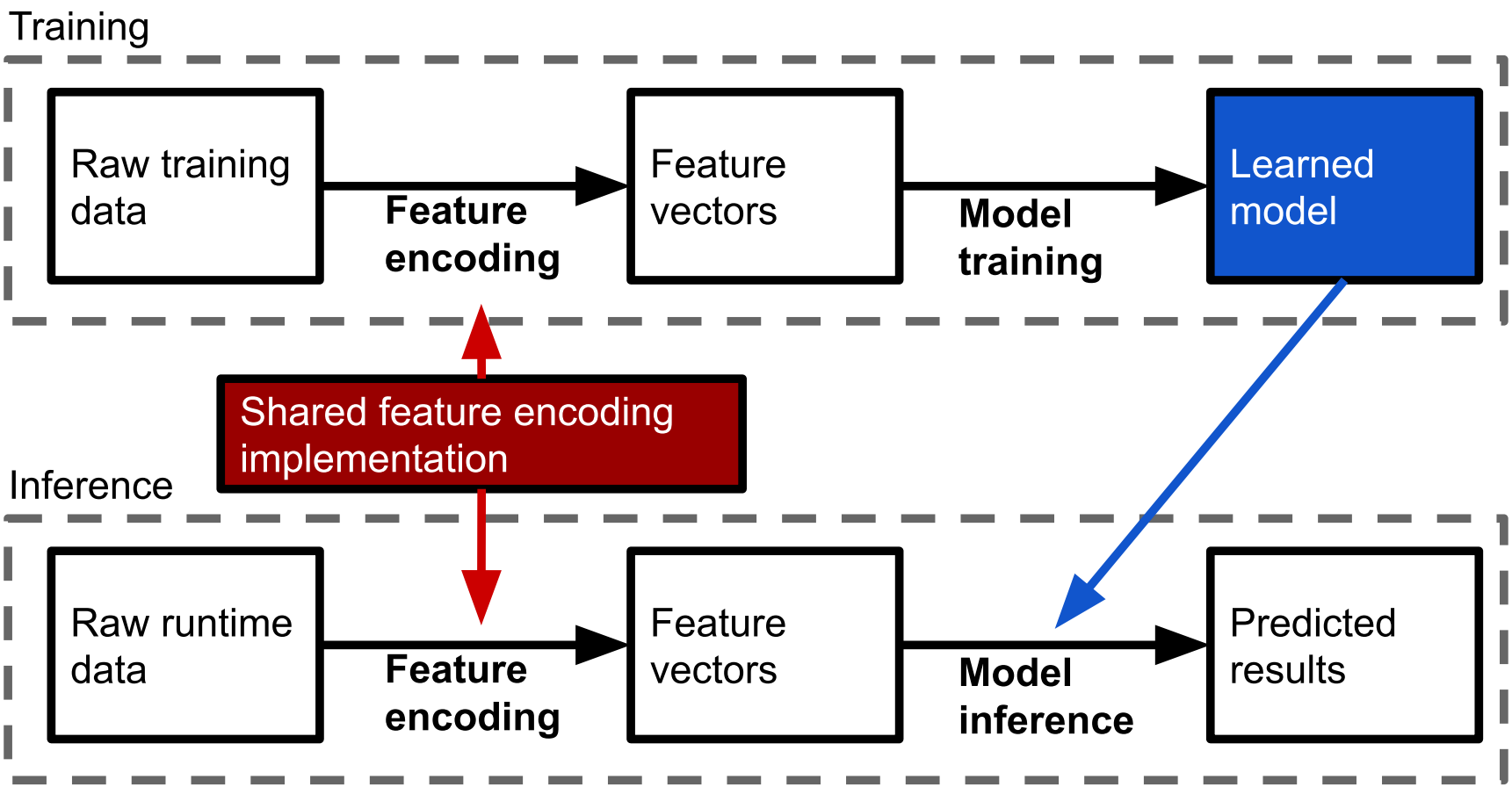


Should feature encoding happen server-side or client-side? Tradeoffs?

Speaker notes

When thinking of model inference as a component within a system, feature encoding can happen with the model-inference component or can be the responsibility of the client. That is, the client either provides the raw inputs (e.g., image files; dotted box in the figure above) to the inference service or the client is responsible for computing features and provides the feature vector to the inference service (dashed box). Feature encoding and model inference could even be two separate services that are called by the client in sequence. Which alternative is preferable is a design decision that may depend on a number of factors, for example, whether and how the feature vectors are stored in the system, how expensive computing the feature encoding is, how often feature encoding changes, how many models use the same feature encoding, and so forth. For instance, in our stock photo example, having feature encoding being part of the inference service is convenient for clients and makes it easy to update the model without changing clients, but we would have to send the entire image over the network instead of just the much smaller feature vector for the reduced 300 x 300 pixels.

REUSING FEATURE ENGINEERING CODE



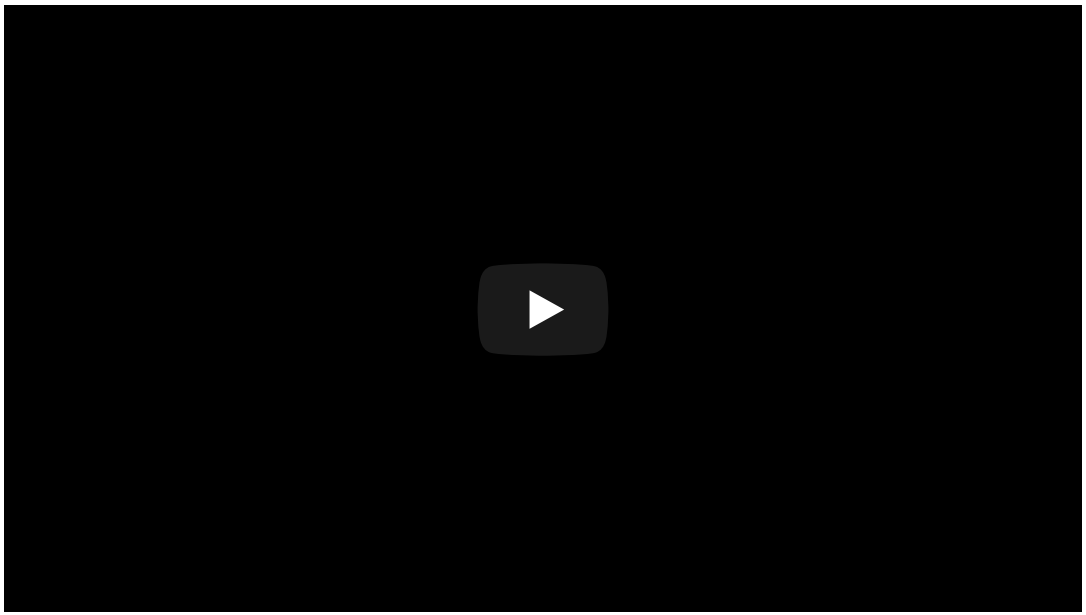
Avoid *training–serving skew*

THE FEATURE STORE PATTERN

- Central place to store, version, and describe feature engineering code
- Can be reused across projects
- Possible caching of expensive features

Many open source and commercial offerings, e.g., Feast, Tecton, AWS SageMaker Feature Store

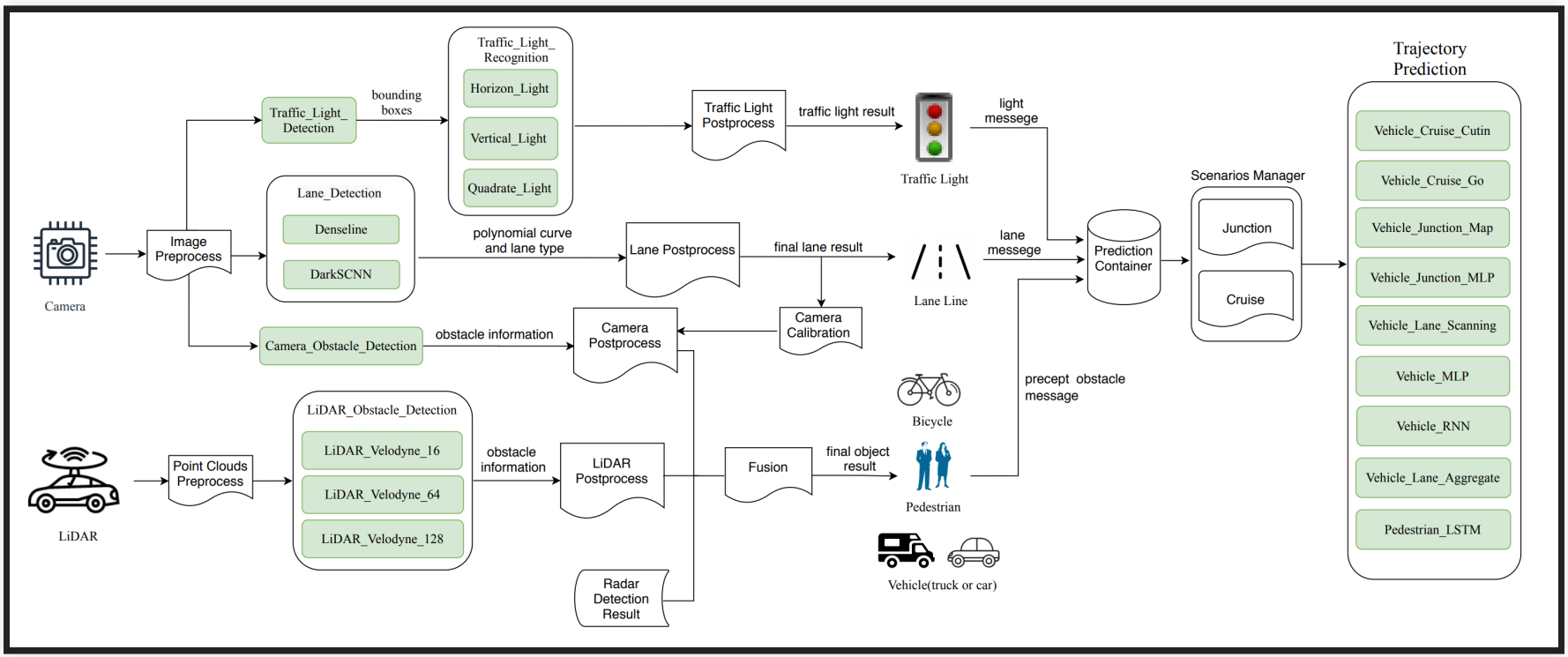
TECTON FEATURE STORE



MORE CONSIDERATIONS FOR DEPLOYMENT DECISIONS

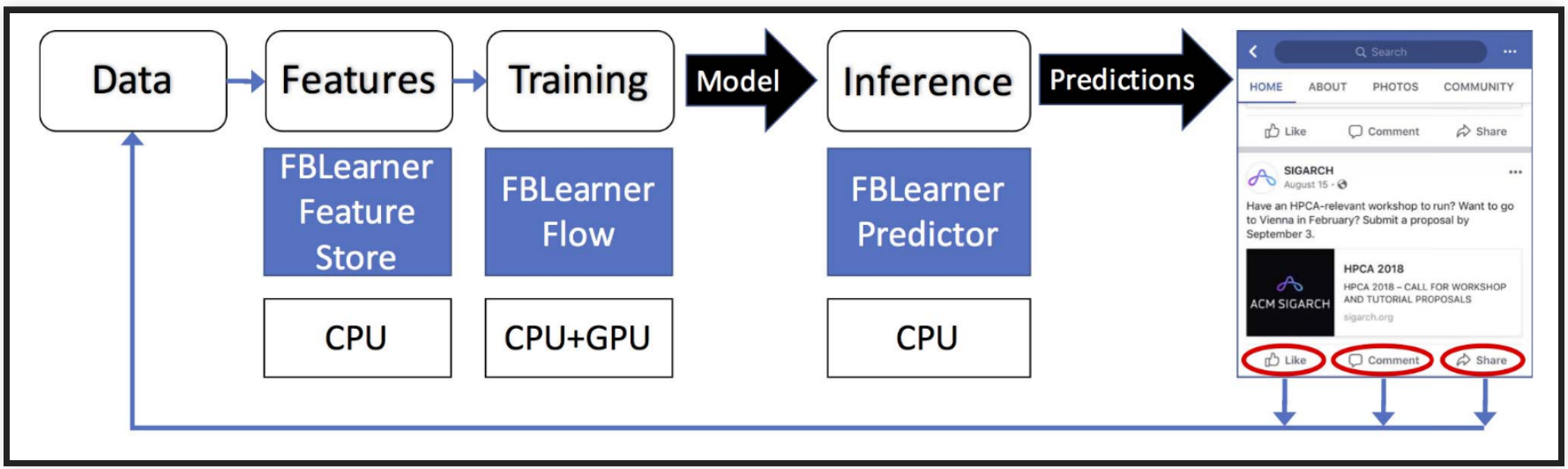
- Coupling of ML pipeline parts
- Coupling with other parts of the system
- Ability for different developers and analysts to collaborate
- Support online experiments
- Ability to monitor

REAL-TIME SERVING; MANY MODELS



Peng, Zi, Jinqiu Yang, Tse-Hsun Chen, and Lei Ma. "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on Apollo." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1240-1250. 2020.

INFRASTRUCTURE PLANNING (FACEBOOK EXAMPLE)



Hazelwood, Kim, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy et al. "Applied machine learning at facebook: A datacenter infrastructure perspective." In 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 620-629. IEEE, 2018.

CAPACITY PLANNING (FACEBOOK EXAMPLE)

Services	Relative Capacity	Compute	Memory
News Feed	100x	Dual-Socket CPU	High
Facer (face recognition)	10x	Single-Socket CPU	Low
Lumos (image understanding)	10x	Single-Socket CPU	Low
Search	10x	Dual-Socket CPU	High
Lang. Translation	1x	Dual-Socket CPU	High
Sigma (anomaly and spam detection)	1x	Dual-Socket CPU	High
Speech Recognition	1x	Dual-Socket CPU	High

Trillions of inferences per day, in real time

Preference for cheap single-CPU machines whether possible

Different latency requirements, some "nice to have" predictions

Some models run on mobile device to improve latency and reduce communication cost

Hazelwood, Kim, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy et al. "Applied machine learning at facebook: A datacenter infrastructure perspective." In 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 620-629. IEEE, 2018.

OPERATIONAL 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?

PREVIEW: TELEMETRY DESIGN

TELEMETRY DESIGN

How to evaluate system performance and 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

- **Interacts with deployment decisions**

TELEMETRY TRADEOFFS

What data to collect? How much? When?

Estimate data volume and possible bottlenecks in system.


現金のみ
Japanese
English
cash only

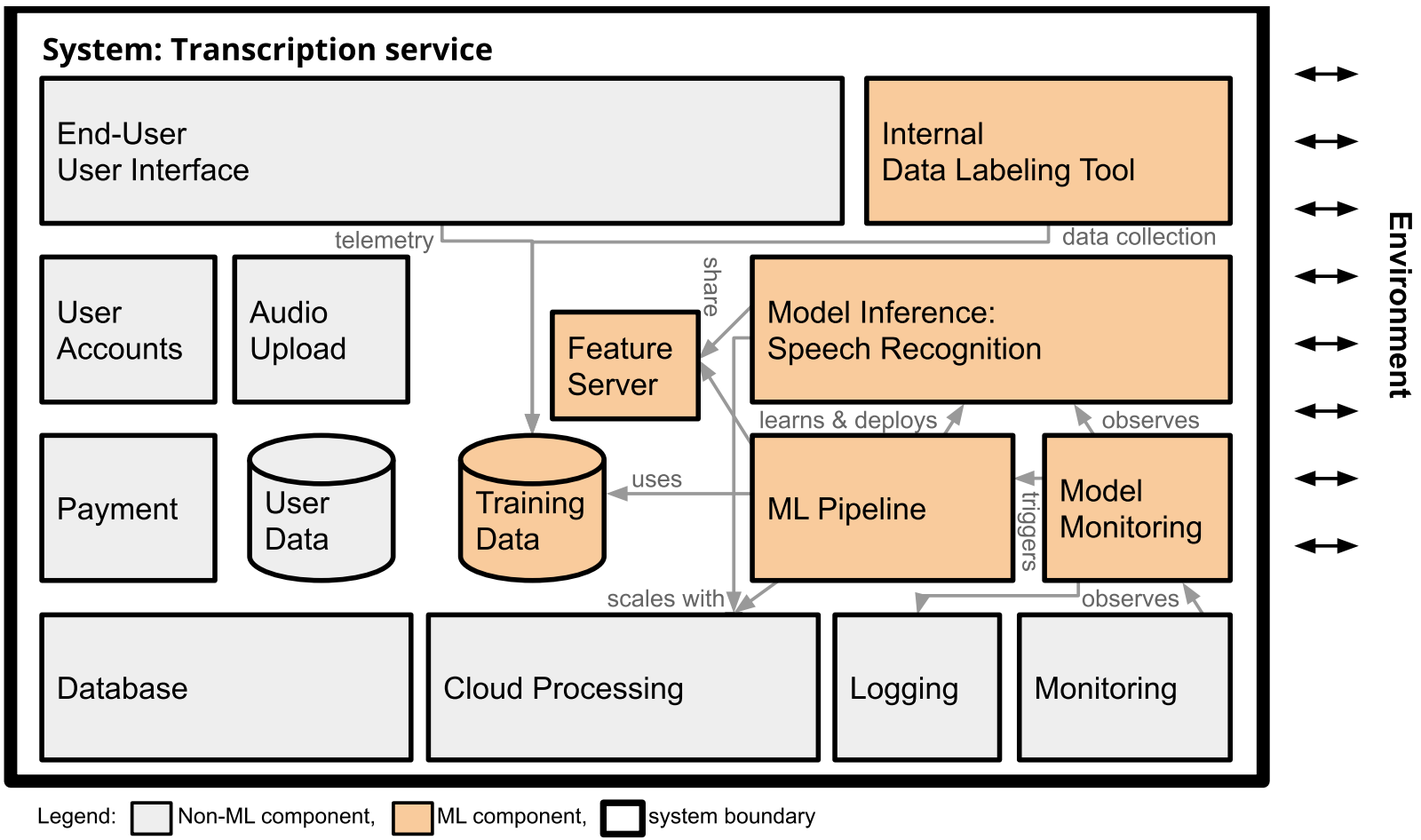
Speaker notes

Discuss alternatives and their tradeoffs. Draw models as suitable.

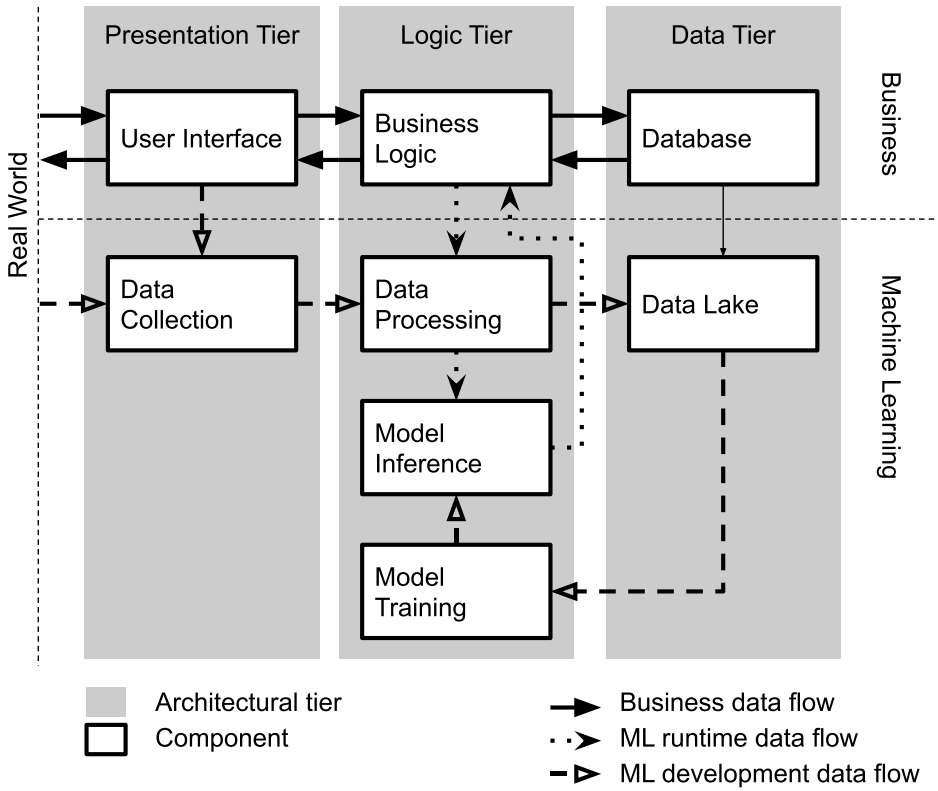
Some data for context: Full-screen png screenshot on Pixel 2 phone (1080x1920) is about 2mb (2 megapixel); Google glasses had a 5 megapixel camera and a 640x360 pixel screen, 16gb of storage, 2gb of RAM. Cellar cost are about $10/GB.

INTEGRATING MODELS INTO A SYSTEM

RECALL: INFERENCE IS A COMPONENT WITHIN A SYSTEM



SEPARATING MODELS AND BUSINESS LOGIC

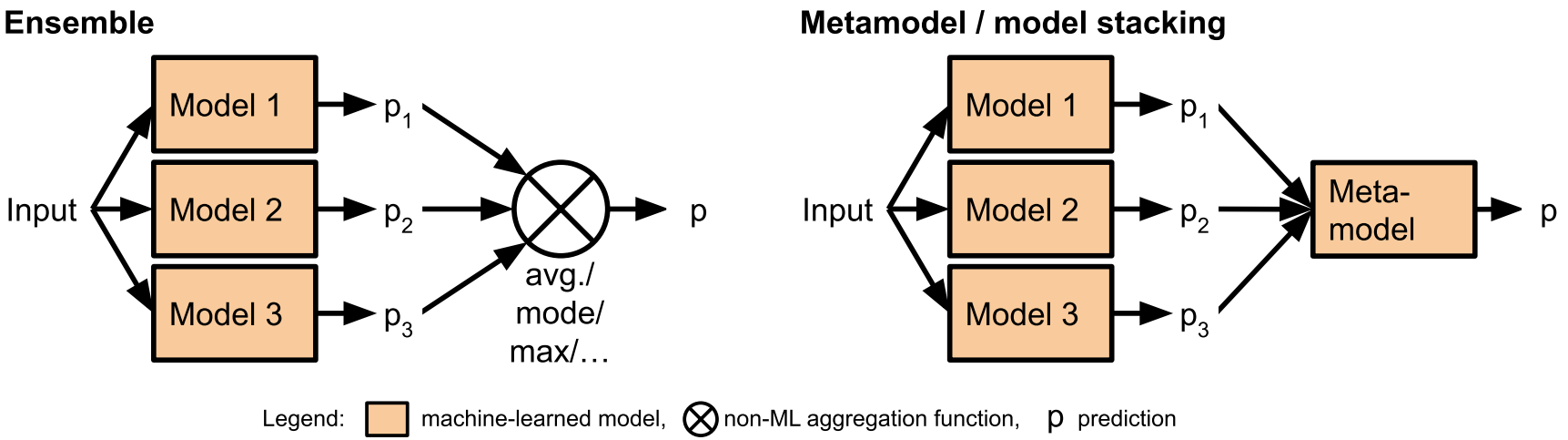


Based on: Yokoyama, Haruki. "Machine learning system architectural pattern for improving operational stability." In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 267-274. IEEE, 2019.

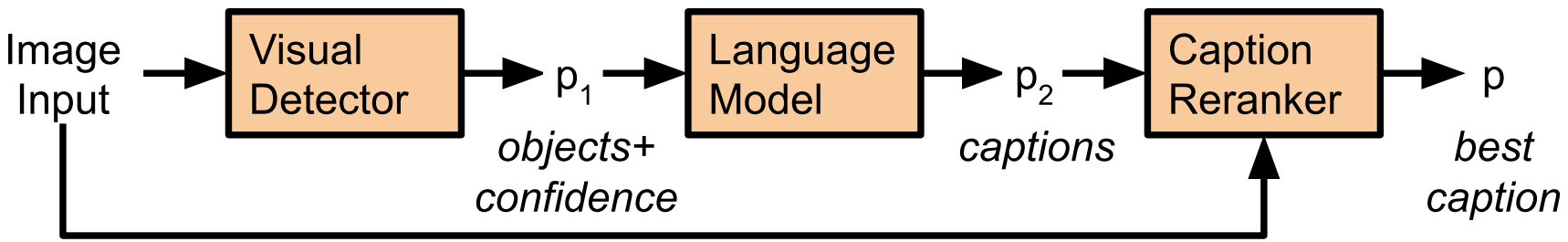
SEPARATING MODELS AND BUSINESS LOGIC

- Clearly divide responsibilities
- Allows largely independent and parallel work, assuming stable interfaces
- Plan location of non-ML safeguards and other processing logic

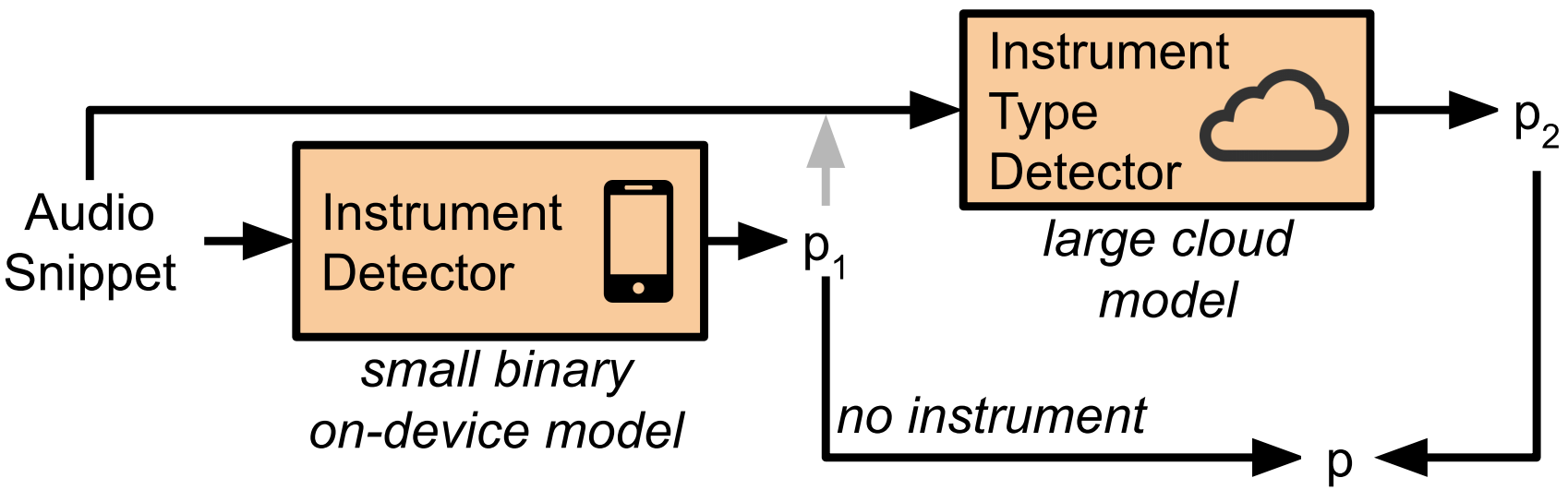
COMPOSING MODELS: ENSEMBLE AND METAMODELS



COMPOSING MODELS: DECOMPOSING THE PROBLEM, SEQUENTIAL



COMPOSING MODELS: CASCADE/TWO-PHASE PREDICTION



DOCUMENTING MODEL INFERENCE INTERFACES

WHY DOCUMENTATION

- Model inference between teams:
  - Data scientists developing the model
  - Other data scientists using the model, evolving the model
  - Software engineers integrating the model as a component
  - Operators managing model deployment
- Will this model work for my problem?
- What problems to anticipate?

CLASSIC API DOCUMENTATION

```
/**
 * compute deductions based on provided adjusted
 * gross income and expenses in customer data.
 *
 * see tax code 26 U.S. Code A.1.B, PART VI
 */
float computeDeductions(float agi, Expenses expenses);
```

WHAT TO DOCUMENT FOR MODELS?

DOCUMENTING INPUT/OUTPUT TYPES FOR INFERENCE COMPONENTS

```
{
  "mid": string,
  "languageCode": string,
  "name": string,
  "score": number,
  "boundingPoly": {
    object (BoundingPoly)
  }
}
```

From Google's public object detection API.

DOCUMENTATION BEYOND INPUT/OUTPUT TYPES

- Intended use cases, model capabilities and limitations
- Supported target distribution (vs preconditions)
- Accuracy (various measures), incl. slices, fairness
- Latency, throughput, availability (service level agreements)
- Model qualities such as explainability, robustness, calibration
- Ethical considerations (fairness, safety, security, privacy)

Example for OCR model? How would you describe these?

MODEL CARDS

- Proposal and template for documentation from Google
- 1-2 page summary
- Focused on fairness
- Includes
    - Intended use, out-of-scope use
    - Training and evaluation data
    - Considered demographic factors
    - Accuracy evaluations
    - Ethical considerations
- Widely discussed, but not frequently adopted

Mitchell, Margaret, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. "Model cards for model reporting." In *Proceedings of the conference on fairness, accountability, and transparency*, pp. 220-229. 2019.

Model Card - Toxicity in Text

Model Details

- The TOXICITY classifier provided by Perspective API [32], trained to predict the likelihood that a comment will be perceived as toxic.
- Convolutional Neural Network.
- Developed by Jigsaw in 2017.

Intended Use

- Intended to be used for a wide range of use cases such as supporting human moderation and providing feedback to comment authors.
- Not intended for fully automated moderation.
- Not intended to make judgments about specific individuals.

Factors

- Identity terms referencing frequently attacked groups, focusing on sexual orientation, gender identity, and race.

Metrics

- Pinned AUC, as presented in [11], which measures threshold-agnostic separability of toxic and non-toxic comments for each group, within the context of a background distribution of other groups.

Ethical Considerations

- Following [31], the Perspective API uses a set of values to guide their work. These values are Community, Transparency, Inclusivity, Privacy, and Topic-neutrality. Because of privacy considerations, the model does not take into account user history when making judgments about toxicity.

Training Data

- Proprietary from Perspective API. Following details in [11] and [32], this includes comments from a online forums such as Wikipedia and New York Times, with crowdsourced labels of whether the comment is "toxic".
- "Toxic" is defined as "a rude, disrespectful, or unreasonable comment that is likely to make you leave a discussion."

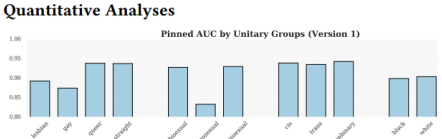
Evaluation Data

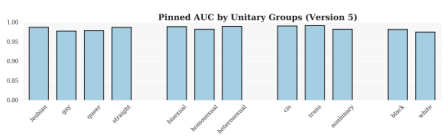
- A synthetic test set generated using a template-based approach, as suggested in [11], where identity terms are swapped into a variety of template sentences.
- Synthetic data is valuable here because [11] shows that real data often has disproportionate amounts of toxicity directed at specific groups. Synthetic data ensures that we evaluate on data that represents both toxic and non-toxic statements referencing a variety of groups.

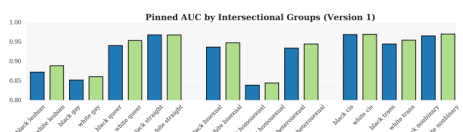
Caveats and Recommendations

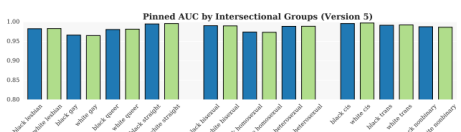
- Synthetic test data covers only a small set of very specific comments. While these are designed to be representative of common use cases and concerns, it is not comprehensive.

Quantitative Analyses





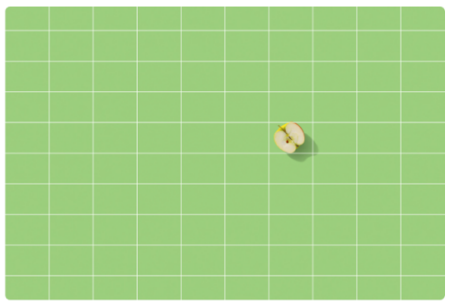




Example from Model Cards paper

Limitations

The following factors may degrade the model's performance.



Object size: Object size must be at least 1% of the image area to be detected.

"Things" vs "stuff": Model was designed to detect discrete objects with clearly discernible shapes ("things"), not a group of overlapping objects or background clutter ("stuff").

Lighting: Poor or harsh, high-contrast illumination (e.g. nighttime, back-lit, side-lit) may degrade model performance.

Occlusion or clutter: Partially obstructed or truncated objects may not be detected. For example, a shirt underneath a jacket, or where less than 25% of an object is visible in the image.

Camera positioning and lens type: Camera angle and positioning (e.g. oblique angles, long-distance), and lens type (e.g. fisheye) may impact model performance.

Blur or noise: Blurry objects, rapid movement between frames, or encoding/decoding noise may degrade model performance.

From: https://modelcards.withgoogle.com/object-detection

FACTSHEETS

- Proposal and template for documentation from IBM
- Intended to communicate intended qualities and assurances
- Longer list of criteria, including
    - Service intention
    - Technical description
    - Intended use
    - Target distribution
    - Own and third-party evaluation results
    - Safety and fairness considerations
    - Explainability
    - Preparation for drift and evolution
    - Security
    - Lineage and versioning

Arnold, Matthew, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, Karthikeyan Natesan Ramamurthy, Darrell Reimer, Alexandra Olteanu, David Piorkowski, Jason Tsay, and Kush R. Varshney. "FactSheets: Increasing trust in AI services through supplier's declarations of conformity." *IBM Journal of Research and Development* 63, no. 4/5 (2019): 6-1.

RECALL: CORRECTNESS VS FIT

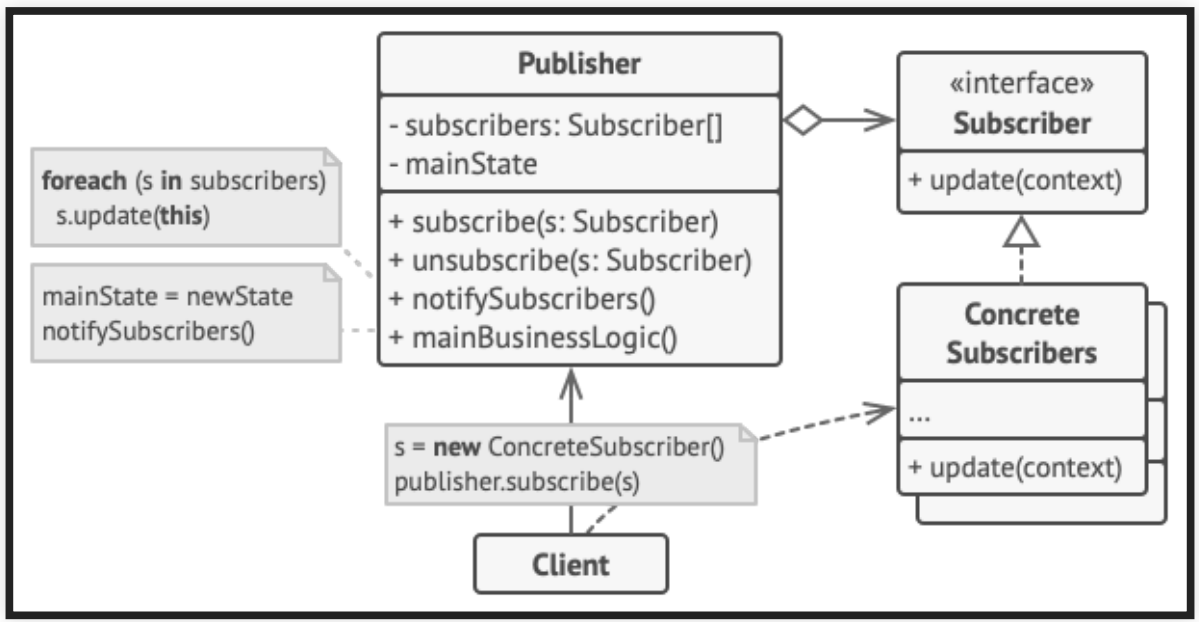
- Without a clear specification a model is difficult to document
- Need documentation to allow evaluation for *fit*
- Description of *target distribution* is a key challenge

DESIGN PATTERNS FOR AI ENABLED SYSTEMS

(no standardization, *yet*)

DESIGN PATTERNS ARE CODIFIED DESIGN KNOWLEDGE

Vocabulary of design problems and solutions

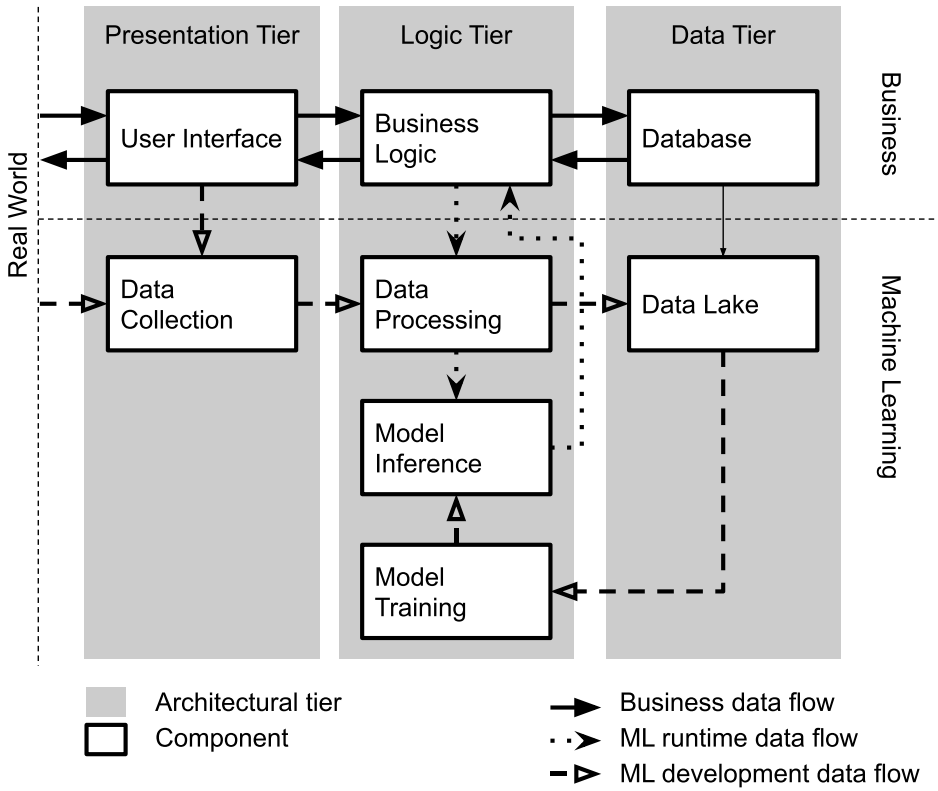


Example: *Observer pattern* object-oriented design pattern describes a solution how objects can be notified when another object changes without strongly coupling these objects to each other

COMMON SYSTEM STRUCTURES

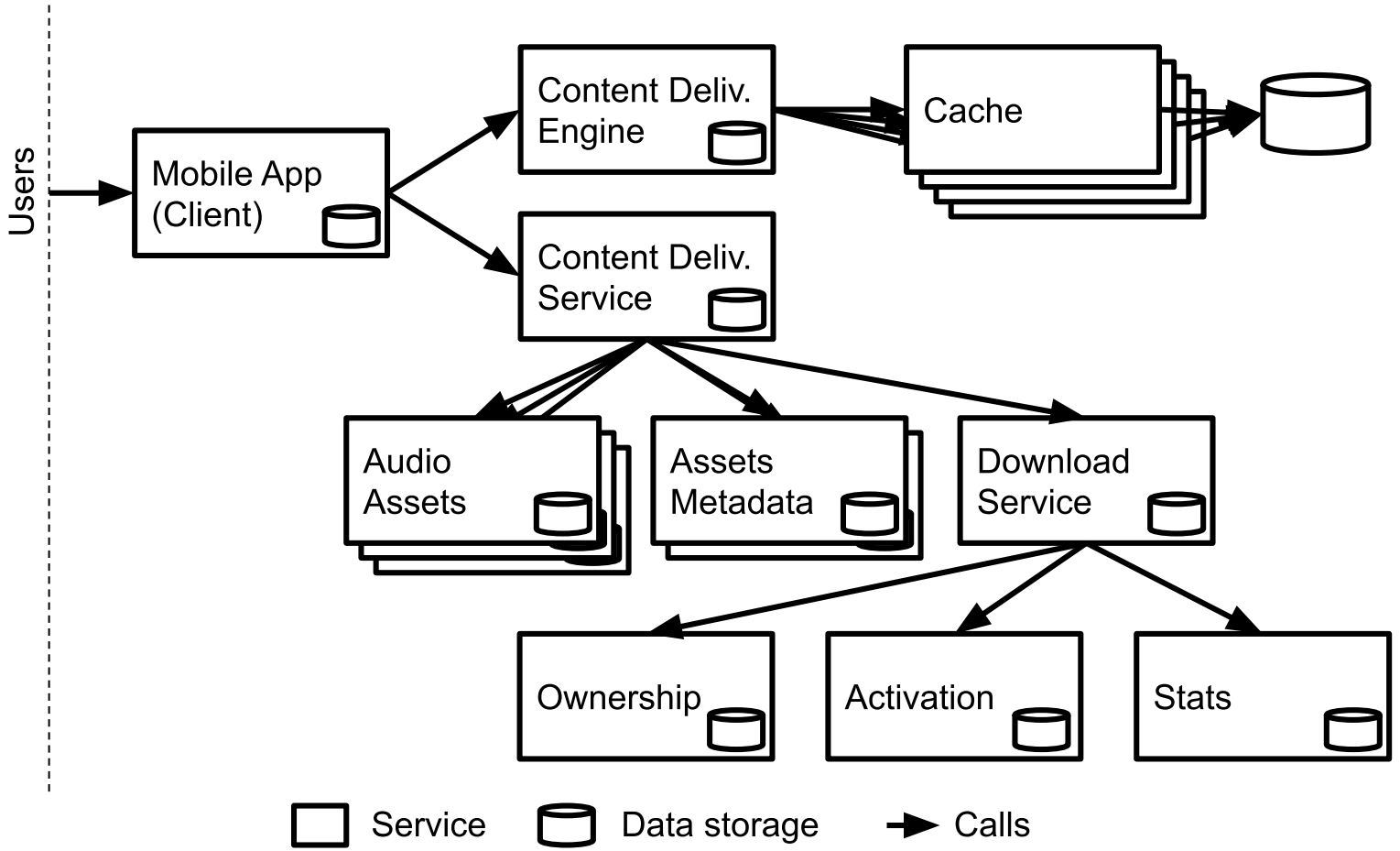
- Client-server architecture
- Multi-tier architecture
- Service-oriented architecture and microservices
- Event-based architecture
- Data-flow architecture

MULTI-TIER ARCHITECTURE



Based on: Yokoyama, Haruki. "Machine learning system architectural pattern for improving operational stability." In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 267-274. IEEE, 2019.

MICROSERVICES



(more later)

PATTERNS FOR ML-ENABLED SYSTEMS

- Stateless/serverless Serving Function Pattern
- Feature-Store Pattern
- Batched/precomuted serving pattern
- Two-phase prediction pattern
- Batch Serving Pattern
- Decouple-training-from-serving pattern

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.

SUMMARY

- Model deployment seems easy, but involves many design decisions
  - What models to use?
  - Where to deploy?
  - How to design feature encoding and feature engineering?
  - How to compose with other components?
  - How to document?
  - How to collect telemetry?
- 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
- Codifying design knowledge as patterns