

Week: COMP 2120 / COMP 6120  
8 of 12  
MICROSERVICES

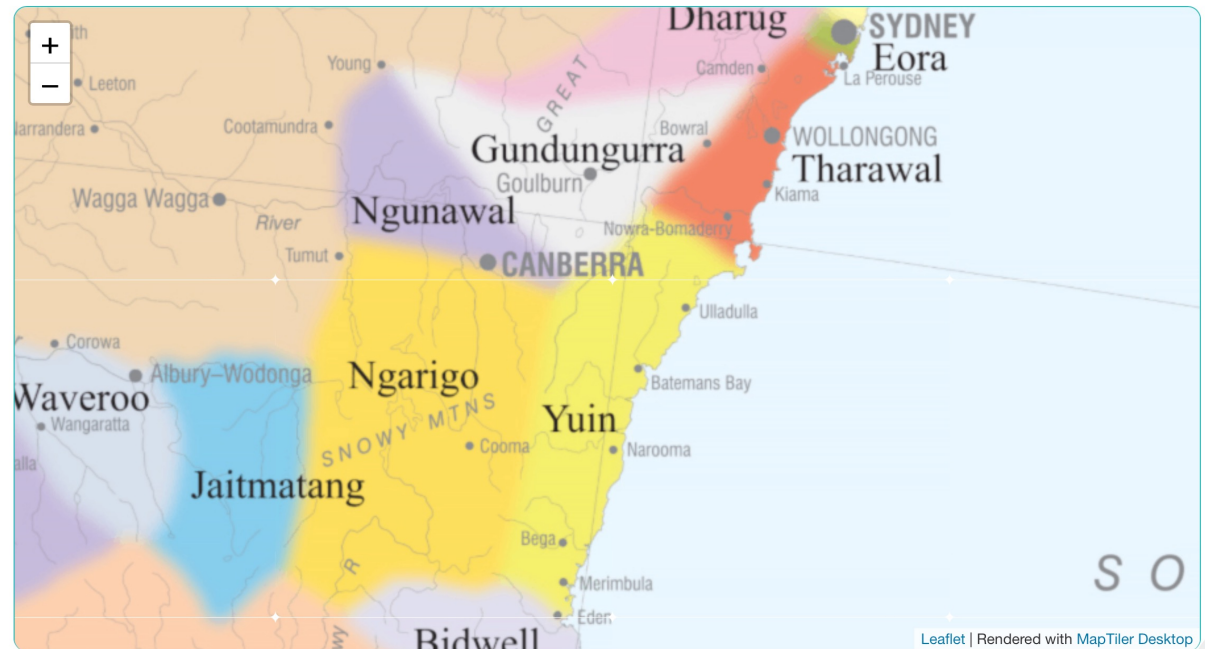
A/Prof Alex Potanin



# ANU Acknowledgment of Country



“We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.”



<https://aiatsis.gov.au/explore/map-indigenous-australia>



# Today

- Monolithic vs Service-Oriented
- Microservices
- Microservice Design Example
- RESTful Services
- Machine Learning Microservices





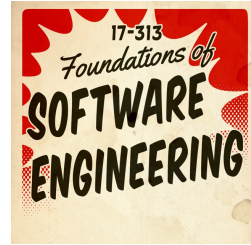
## Monolithic vs Service-Oriented



Facebook Network Engineering Team  
after doing `git push` of BGP changes:



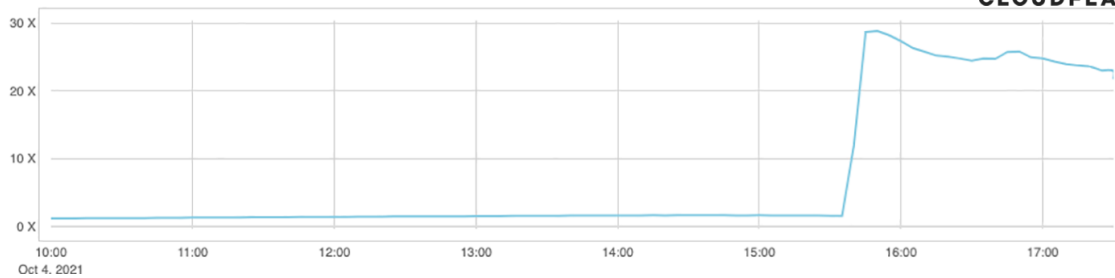
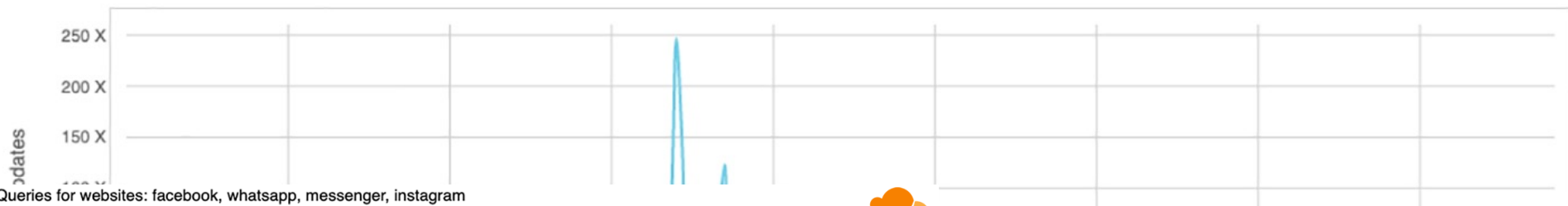
# Facebook on Oct 4, 2021



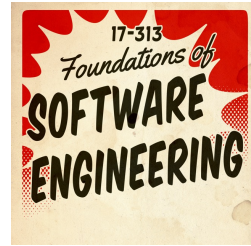
Source: <https://blog.cloudflare.com/october-2021-facebook-outage/>



BGP updates Facebook

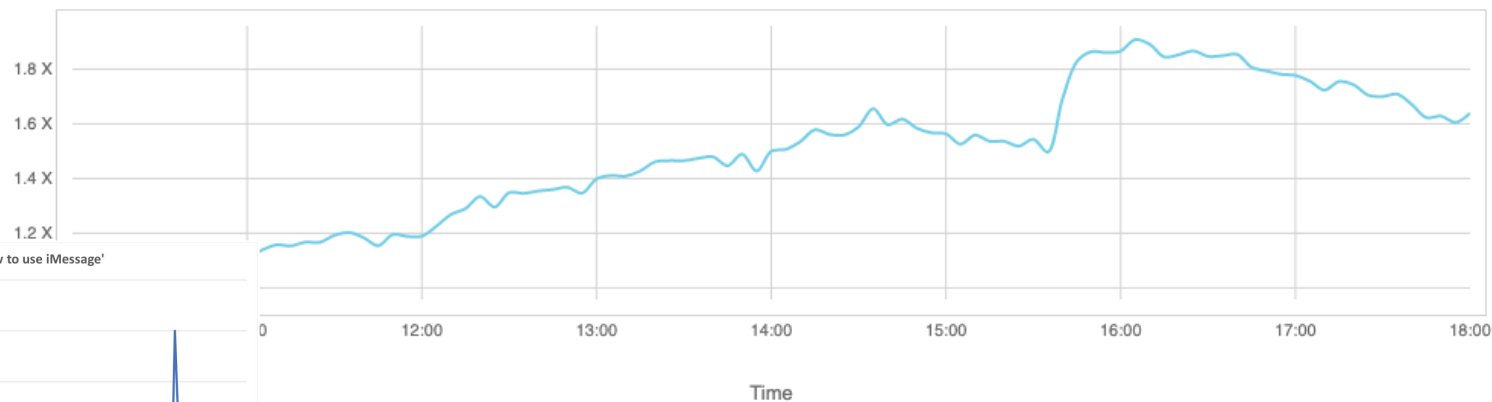


# Facebook on Oct 4, 2021



Source: <https://blog.cloudflare.com/october-2021-facebook-outage/>

Queries for websites: twitter, signal, telegram, tiktok



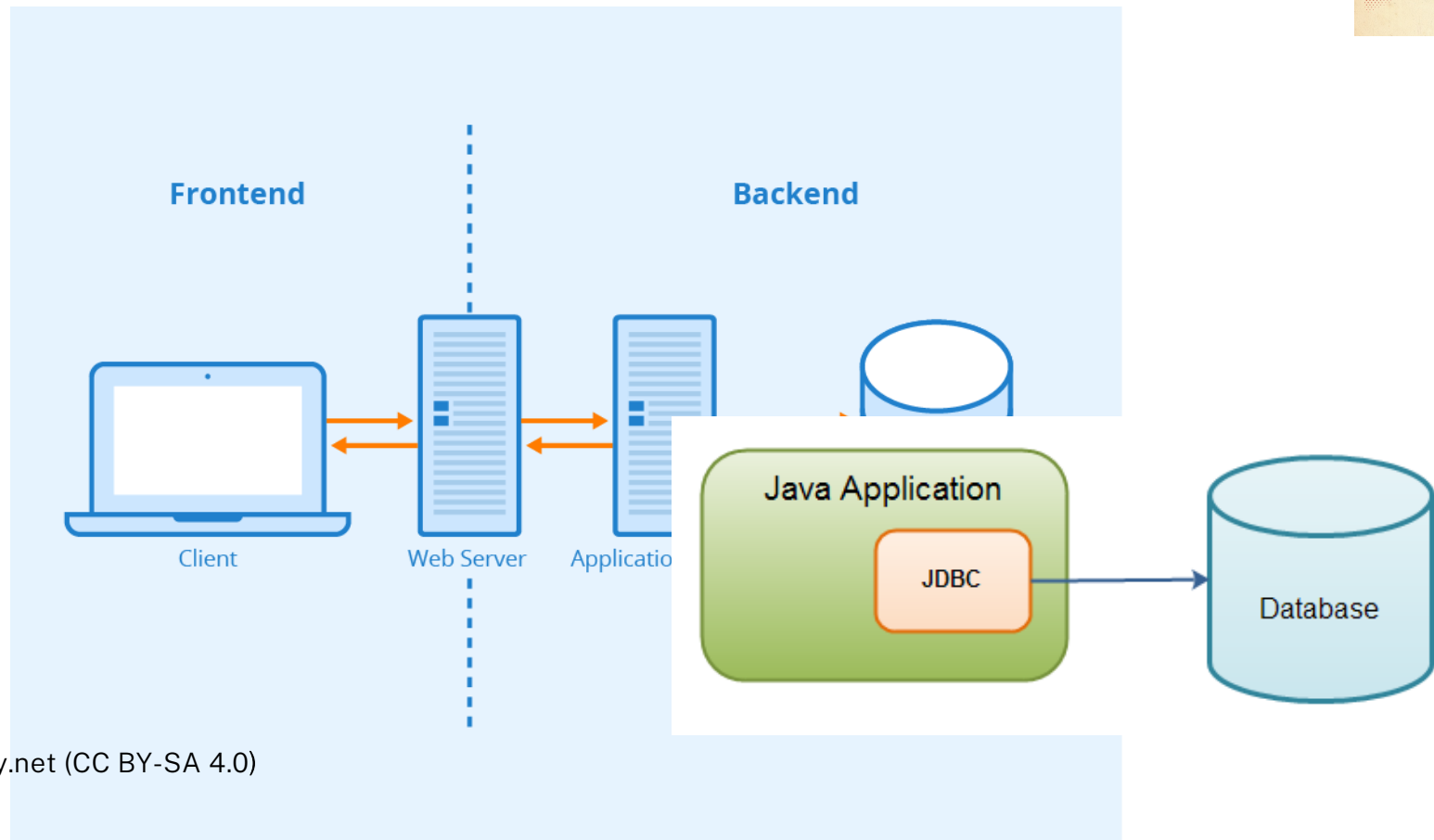
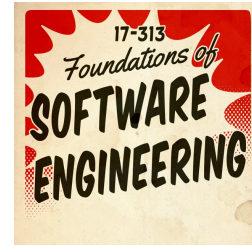
Google Search Interest: 'How to use iMessage'



Some interesting insights about the dependency web of the Web:  
[https://www.synergylabs.org/yuvraj/docs/Kashaf\\_IMC2020\\_WebDependency.pdf](https://www.synergylabs.org/yuvraj/docs/Kashaf_IMC2020_WebDependency.pdf)



# Monolithic styles

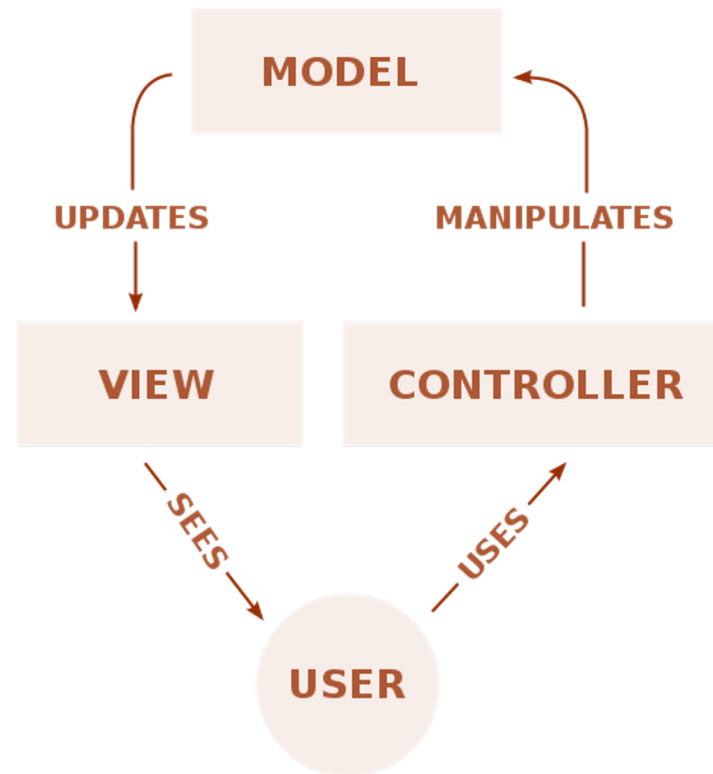
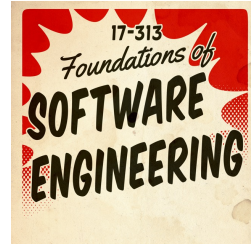


Source: <https://www.seobility.net> (CC BY-SA 4.0)





# Monolithic styles: MVC Pattern



# Monoliths

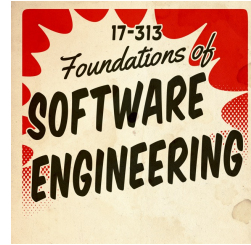


What are the consequences of this architecture? On:

- Scalability
- Reliability
- Performance
- Development
- Maintainability
- Evolution
- Testability
- Ownership
- Data Consistency



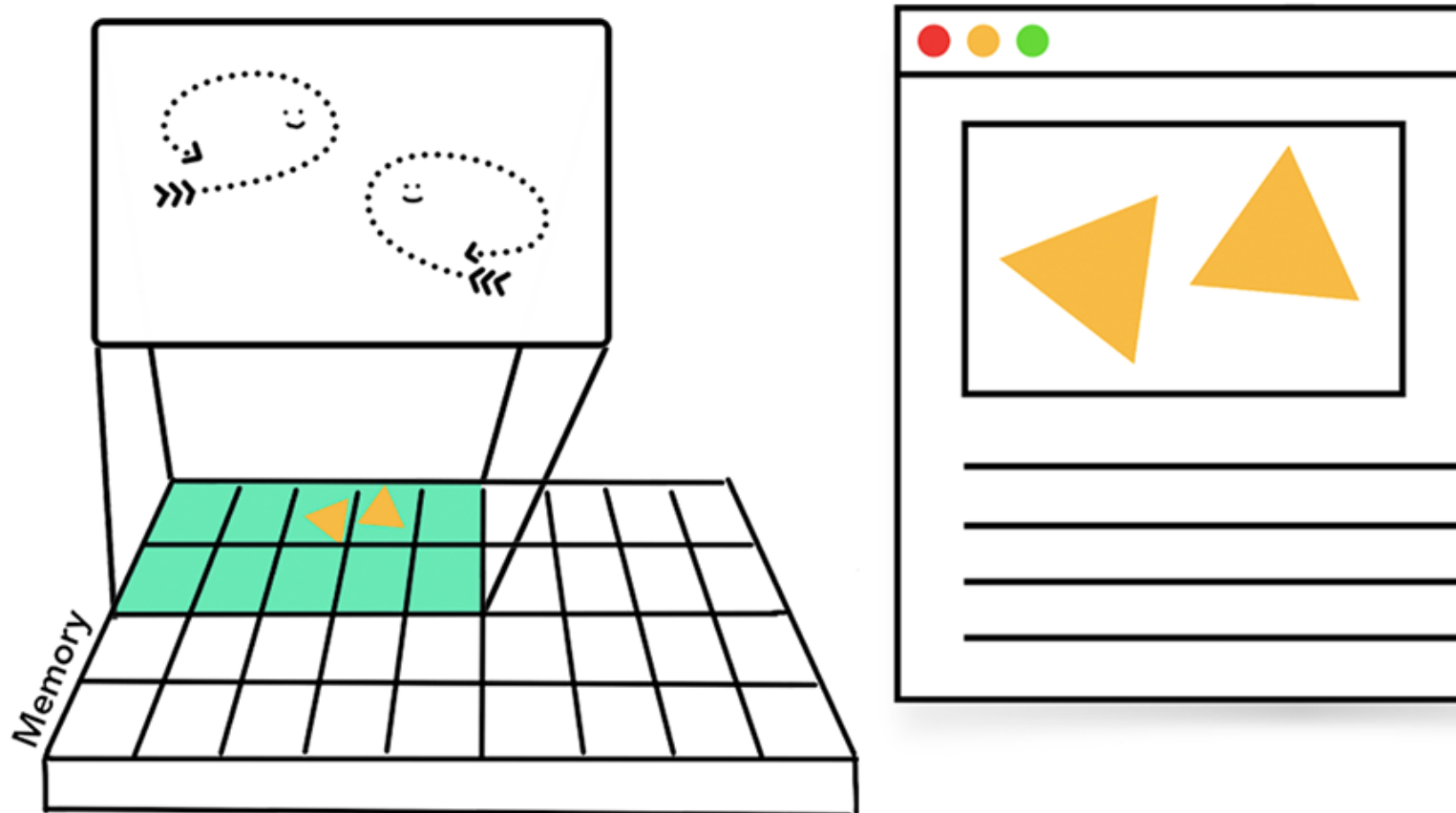
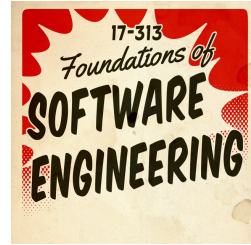
# Web Browsers



Source: <https://developers.google.com/web/updates/2018/09/inside-browser-part1> (CC BY 4.0)



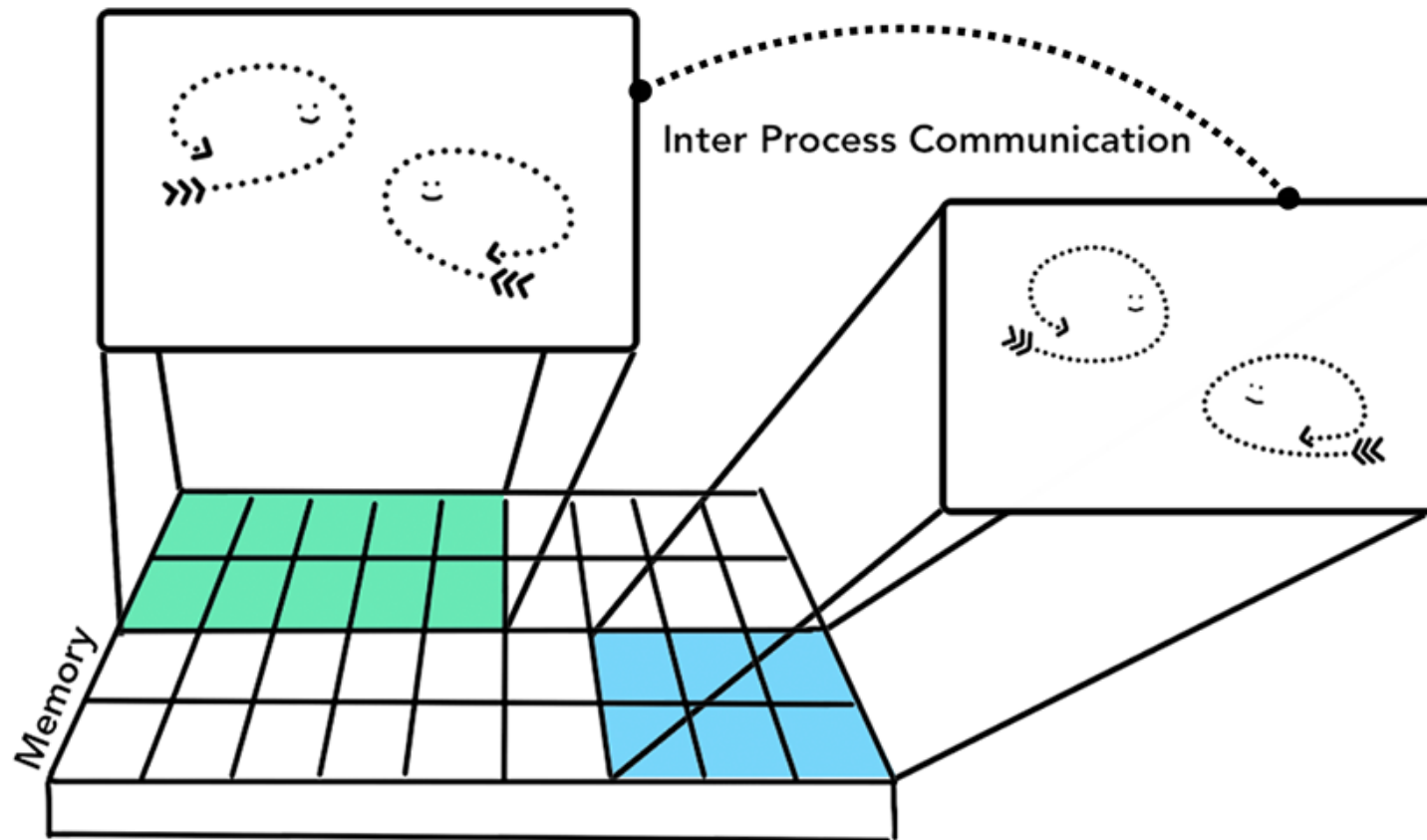
# Browser: A multi-threaded process



Source: <https://developers.google.com/web/updates/2018/09/inside-browser-part1> (CC BY 4.0)



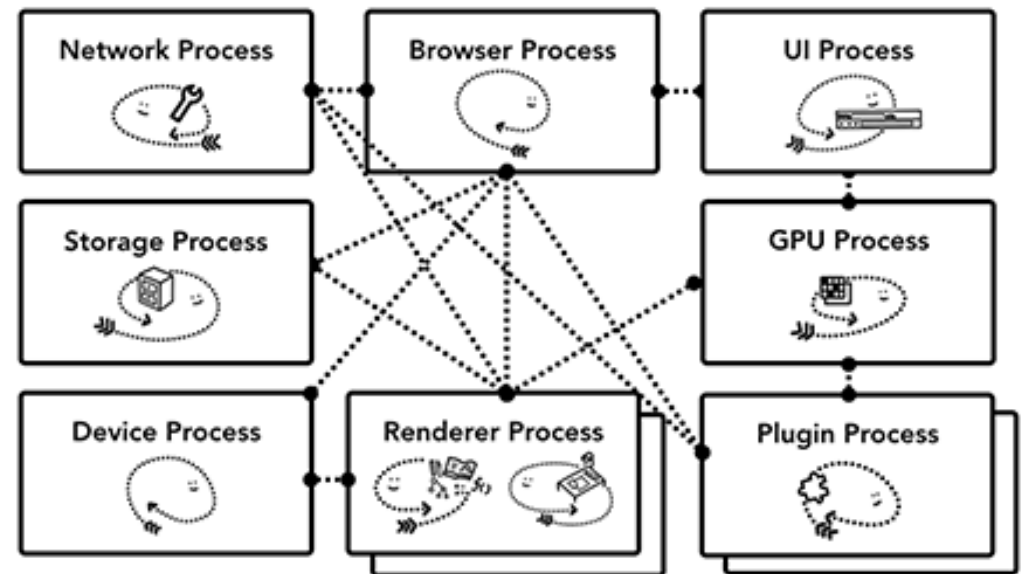
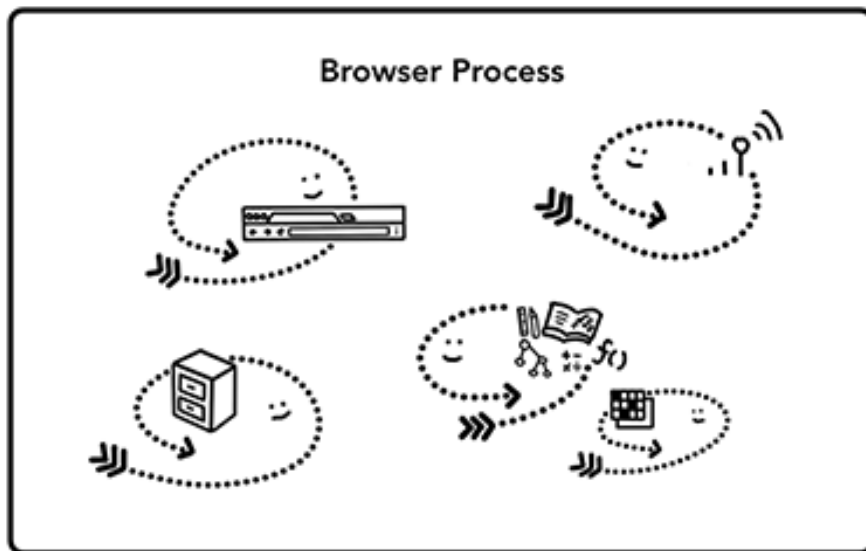
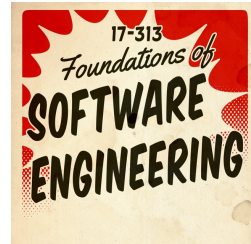
# Multi-process browser with IPC



Source: <https://developers.google.com/web/updates/2018/09/inside-browser-part1> (CC BY 4.0)



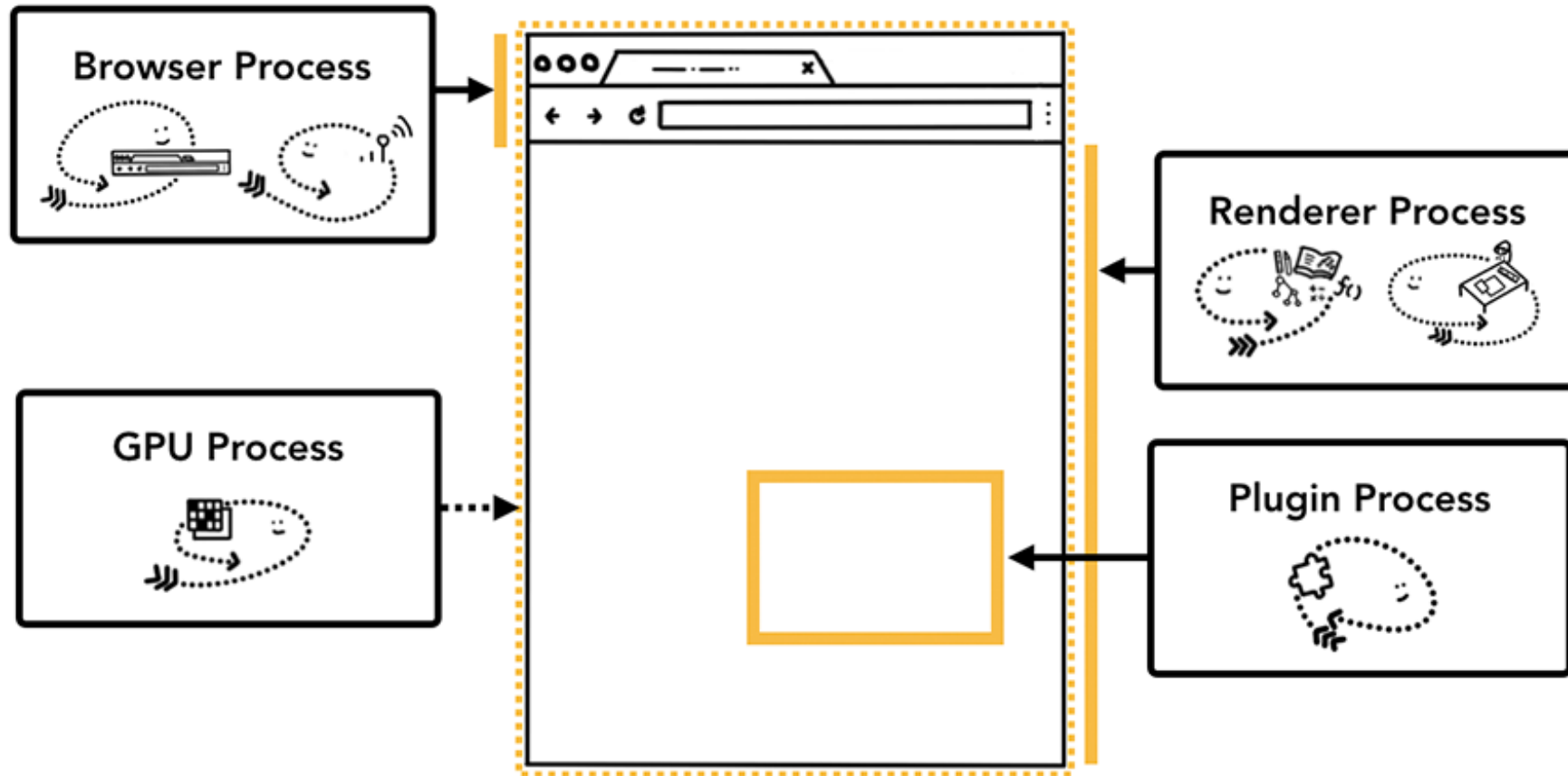
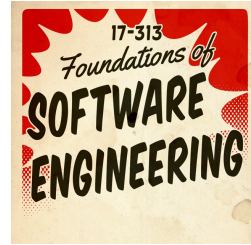
# Browser Architectures



Source: <https://developers.google.com/web/updates/2018/09/inside-browser-part1> (CC BY 4.0)



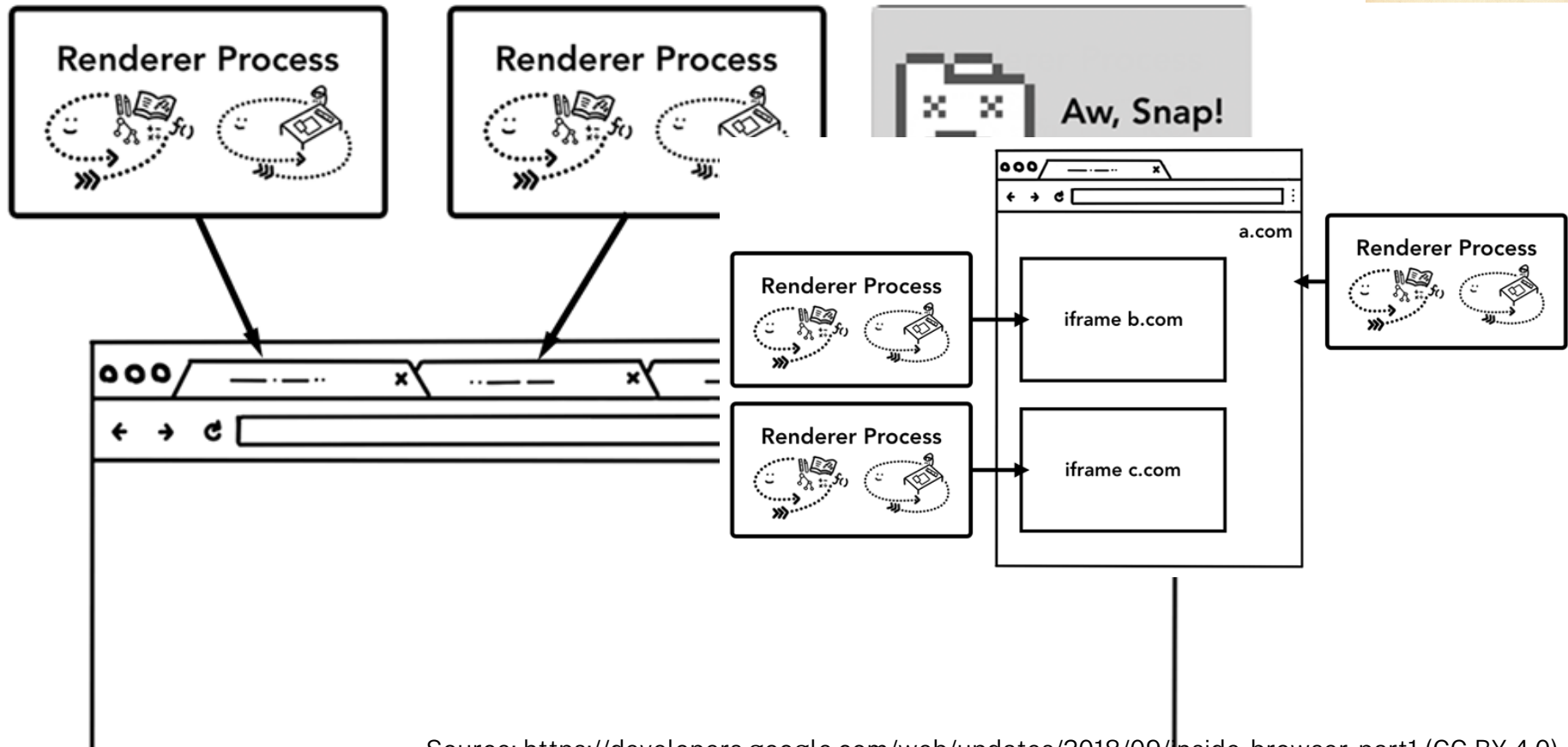
# Service-based browser architecture



Source: <https://developers.google.com/web/updates/2018/09/inside-browser-part1> (CC BY 4.0)



# Service-based browser architecture





Source: <https://developers.google.com/web/updates/2018/09/inside-browser-part1> (CC BY 4.0)





# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send [potantin](https://poll-ev.com/potantin) to 22333 

Have you written a program that uses a service before? 

Yes **(A)**

No **(B)**

I don't know **(C)**

Can you repeat the question? **(D)**



Edit the detailed description

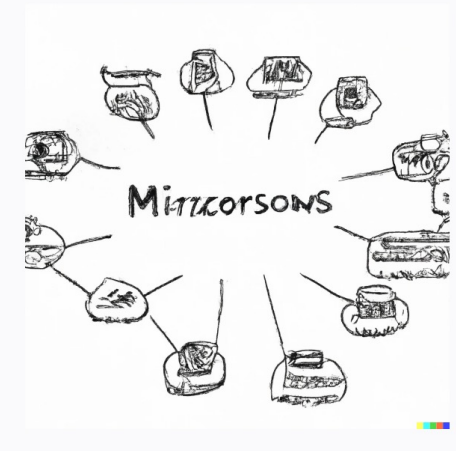
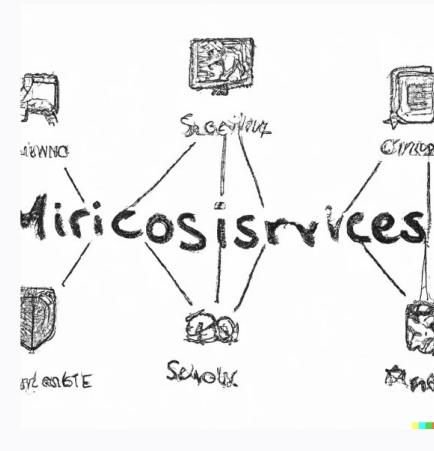
Surprise me

Upload



pencil drawing of microservices

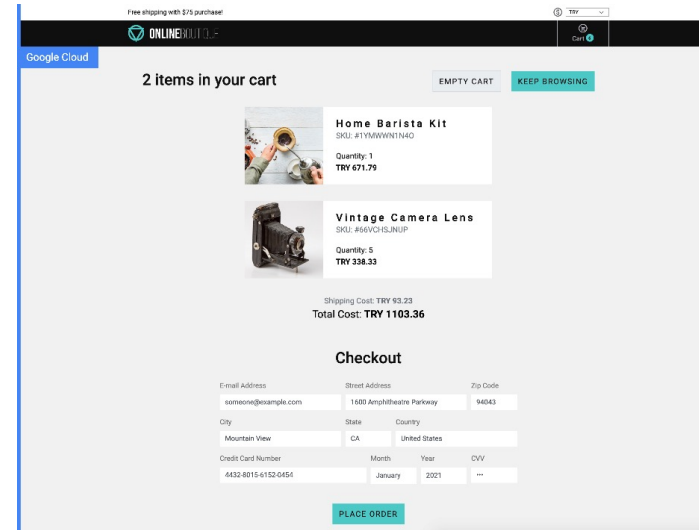
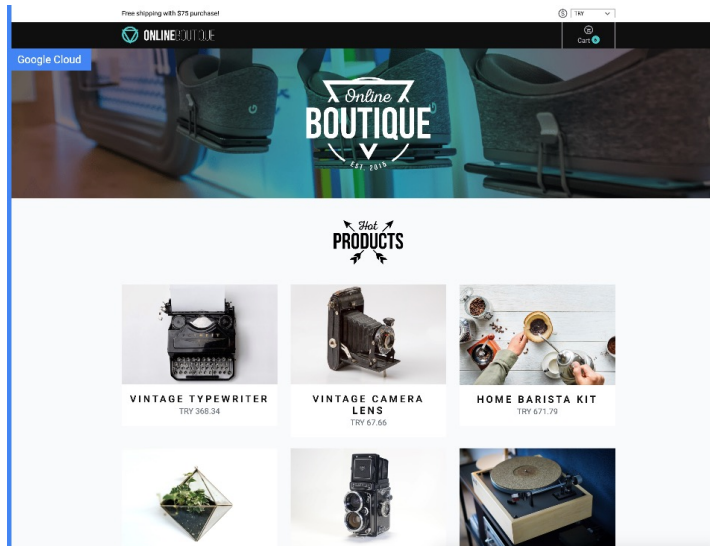
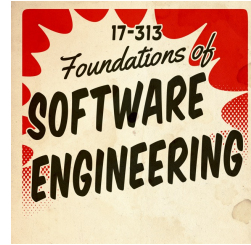
Generate



# Microservices



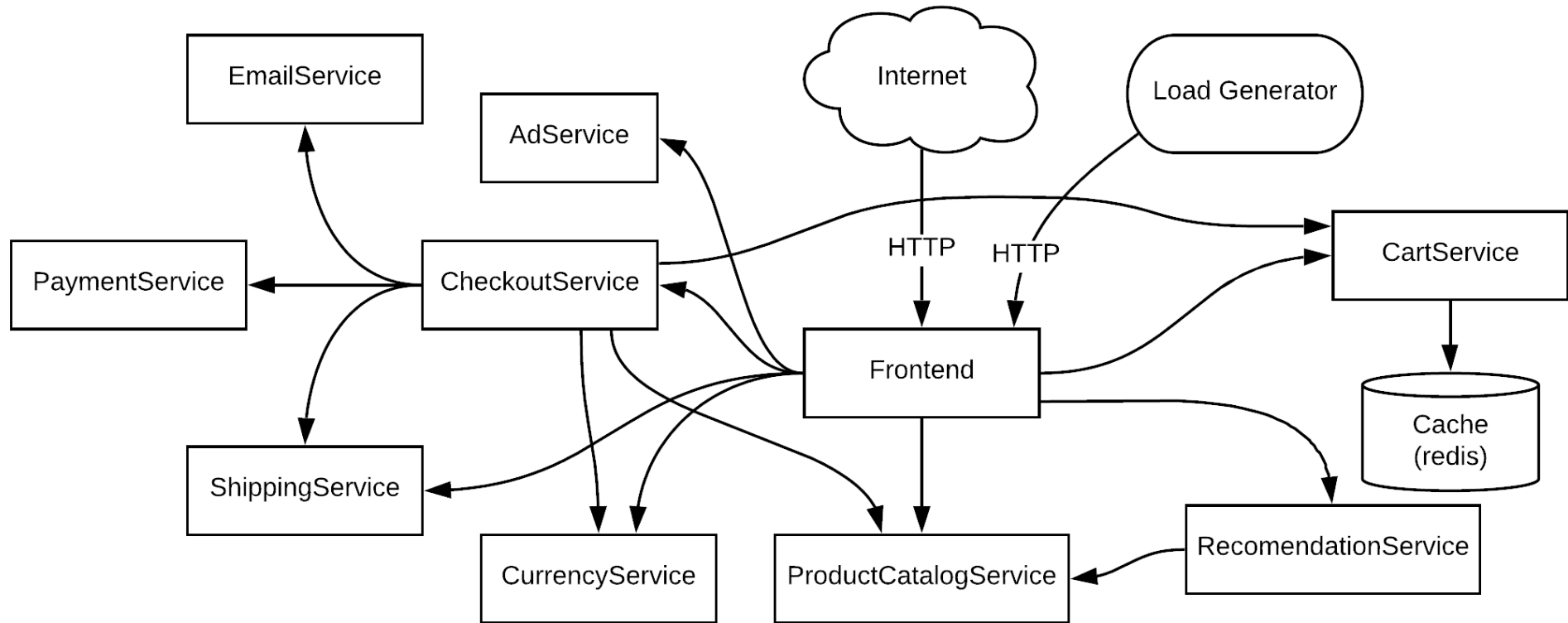
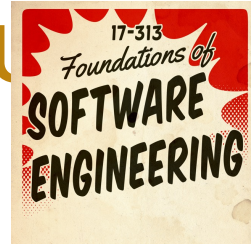
# Hipster Shop User Interface



<https://github.com/GoogleCloudPlatform/microservices-demo>



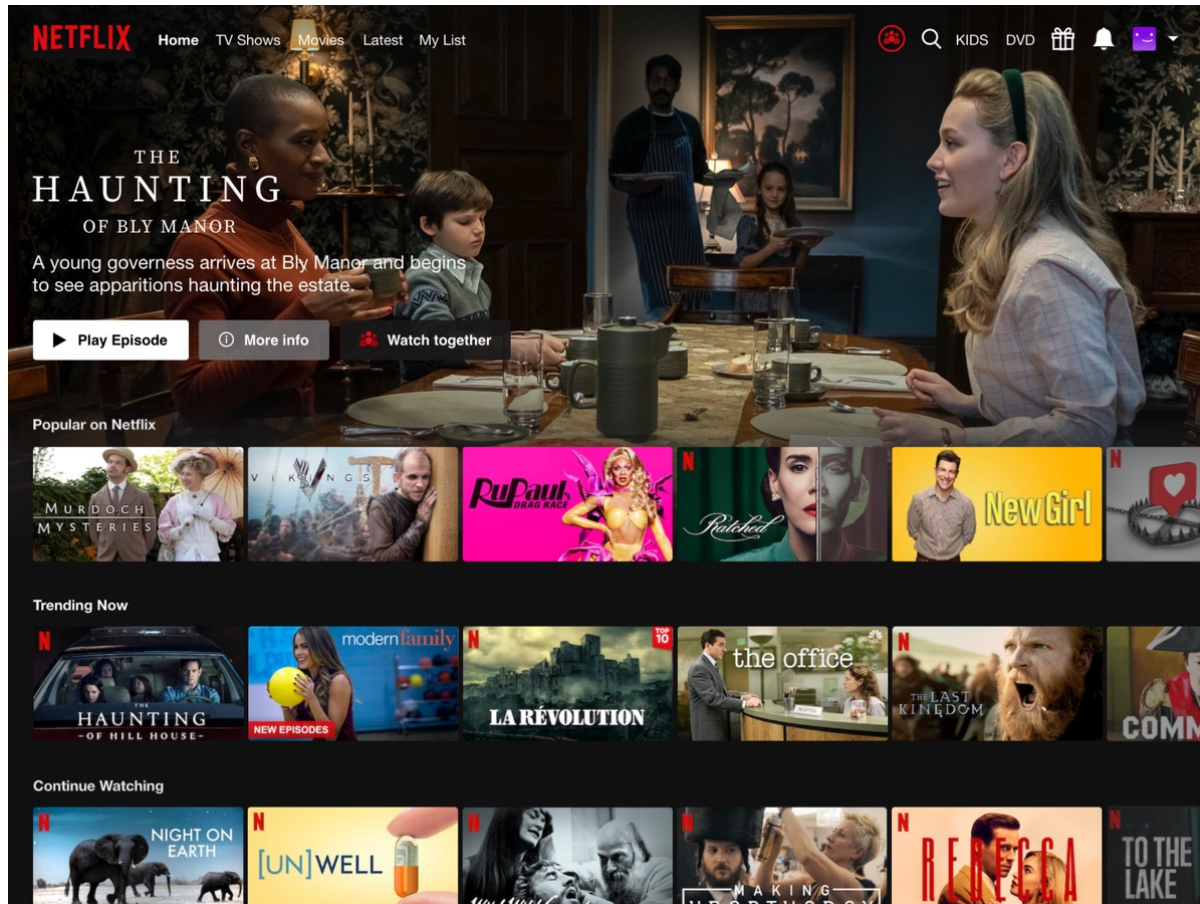
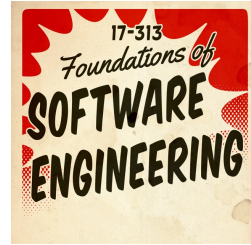
# Hipster Shop Microservice Architecture



<https://github.com/GoogleCloudPlatform/microservices-demo>

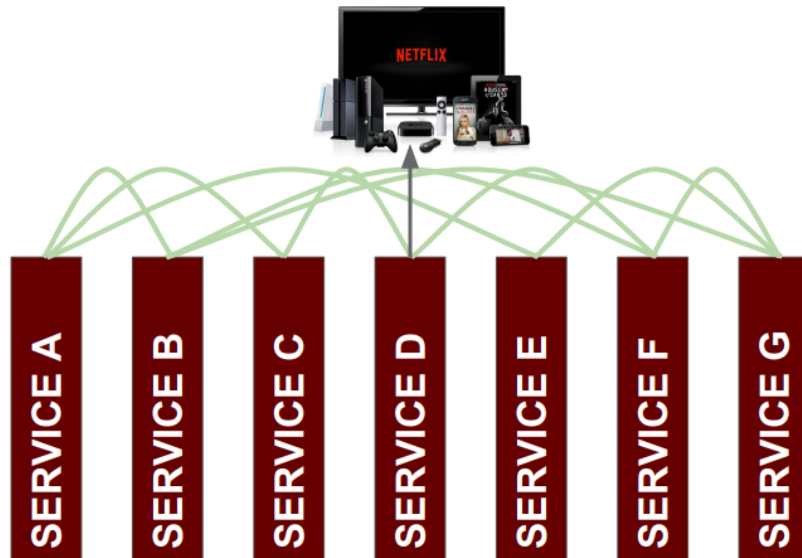


# Netflix





## AppBoot



Bookmarks

Recommendations

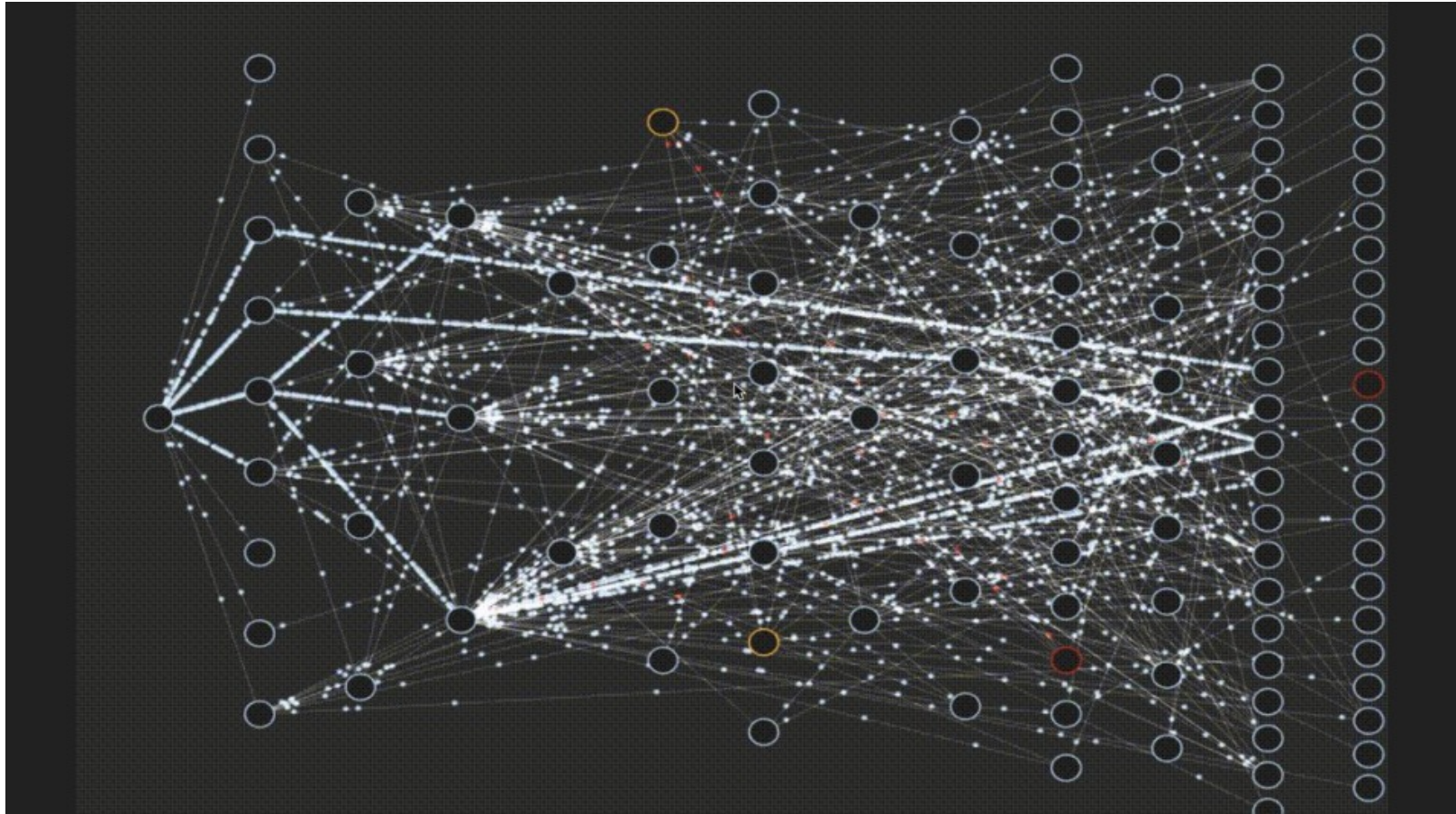
My List

Metrics

(as of 2016)



# Netflix Microservices

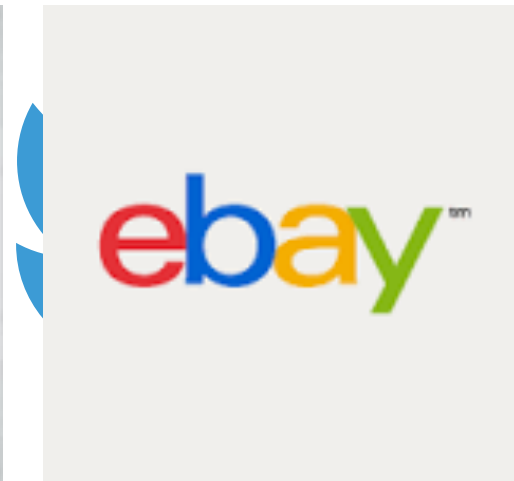
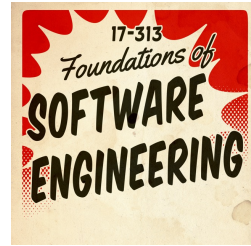
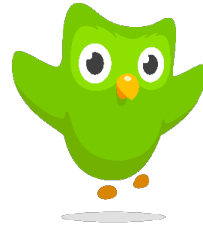


(as of 2016)

<https://www.youtube.com/watch?v=CZ3wluvmHeM>

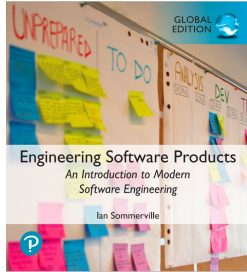


# Who uses Microservices?





# Software services

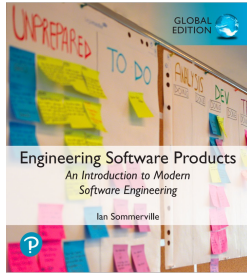


- A software service is a software component that can be accessed from remote computers over the Internet. Given an input, a service produces a corresponding output, without side effects.
  - The service is accessed through its published interface and all details of the service implementation are hidden.
  - Services do not maintain any internal state. State information is either stored in a database or is maintained by the service requestor.
- When a service request is made, the state information may be included as part of the request and the updated state information is returned as part of the service result.
- As there is no local state, services can be dynamically reallocated from one virtual server to another and replicated across several servers.



# Modern web services

- After various experiments in the 1990s with service-oriented computing, the idea of ‘big’ Web Services emerged in the early 2000s.
- These were based on XML-based protocols and standards such as SOAP for service interaction and WSDL for interface description.
- Most software services don’t need the generality that’s inherent in the design of web service protocols.
- Consequently, modern service-oriented systems, use simpler, ‘lighter weight’ service-interaction protocols that have lower overheads and, consequently, faster execution.

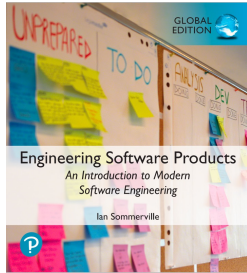


In short, the microservice architectural style [1] is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

<https://martinfowler.com/articles/microservices.html>



# Microservices



- Microservices are small-scale, stateless, services that have a single responsibility. They are combined to create applications.
- They are completely independent with their own database and UI management code.
- Software products that use microservices have a *microservices architecture*.
- If you need to create cloud-based software products that are adaptable, scalable and resilient then it is recommended that you design them around a



# Microservices

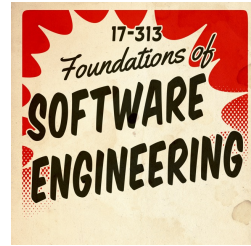


What are the consequences of this architecture? On:

- Scalability
- Reliability
- Performance
- Development
- Maintainability
- Evolution
- Testability
- Ownership
- Data Consistency



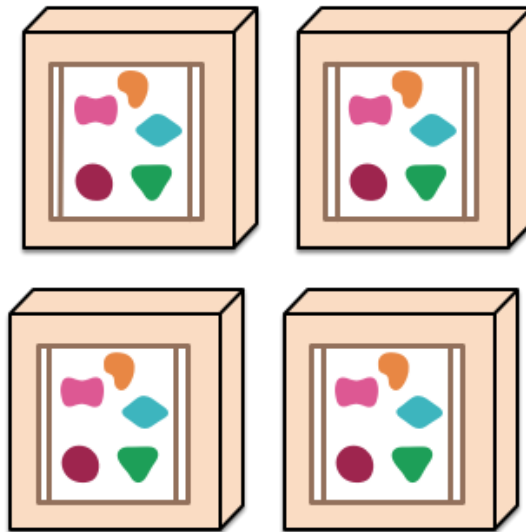
# Scalability



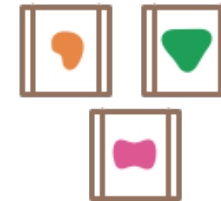
*A monolithic application puts all its functionality into a single process...*



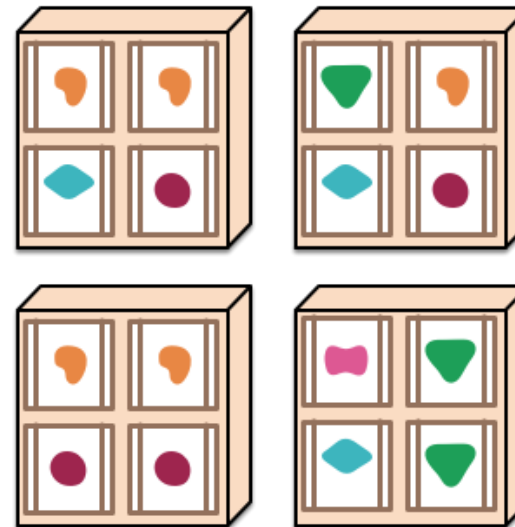
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



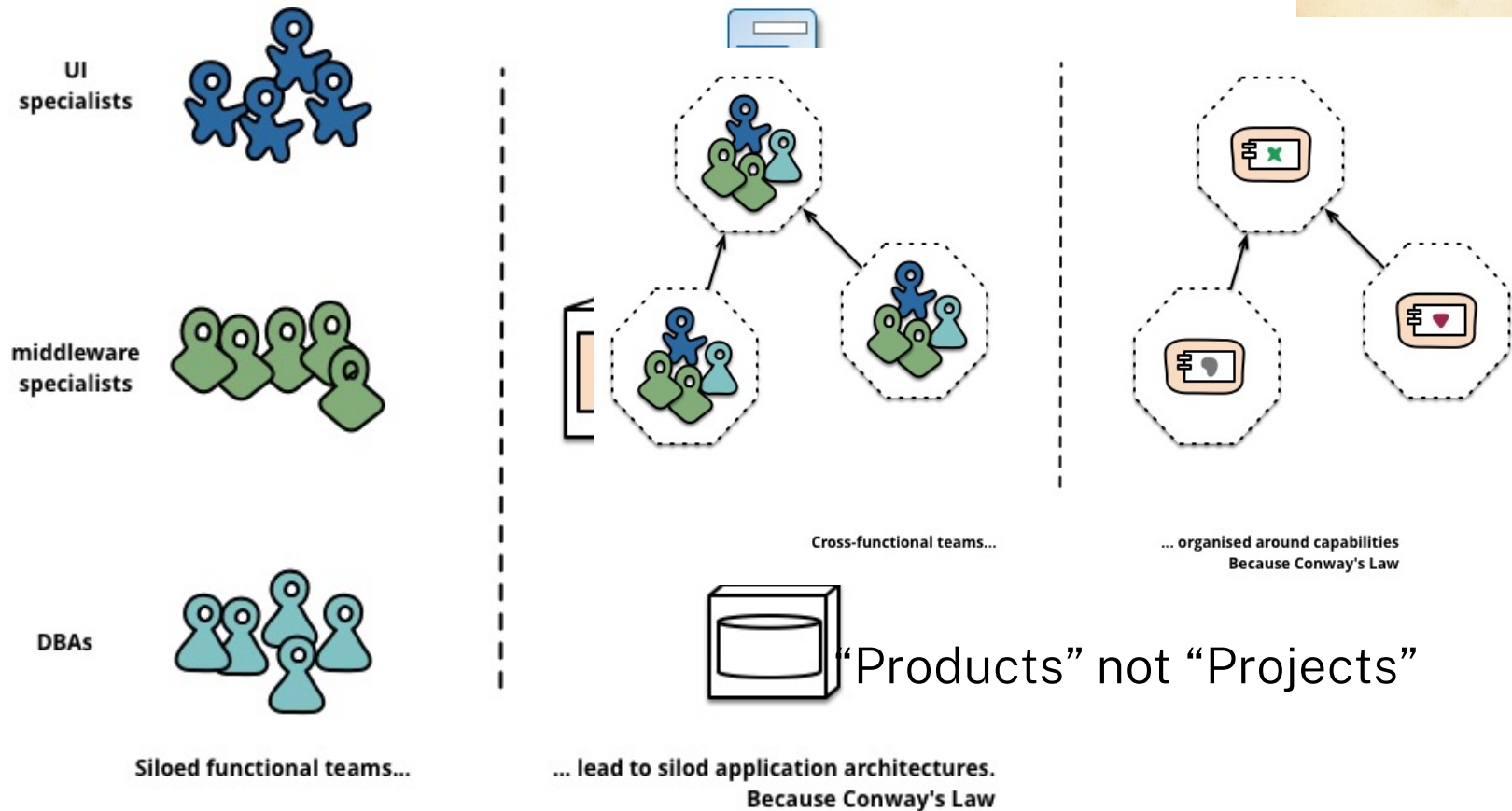
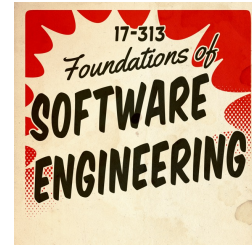
*... and scales by distributing these services across servers, replicating as needed.*



Source: <http://martinfowler.com/articles/microservices.html>



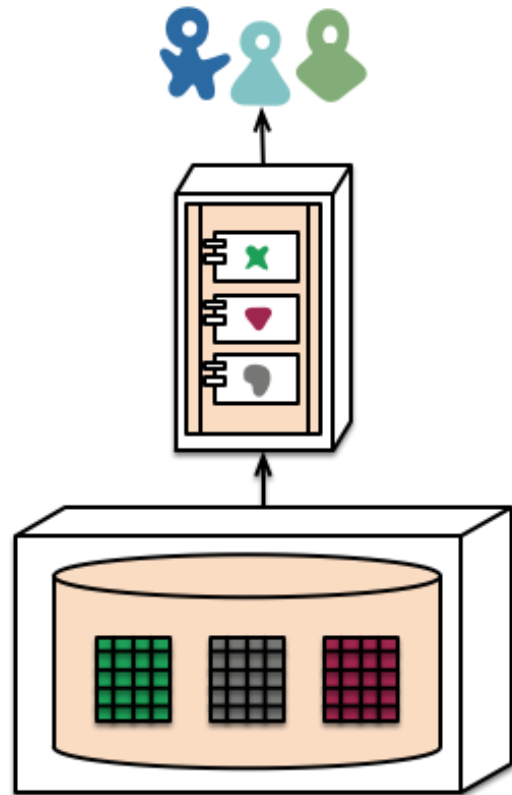
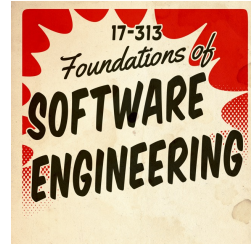
# Team Organization (Conway's Law)



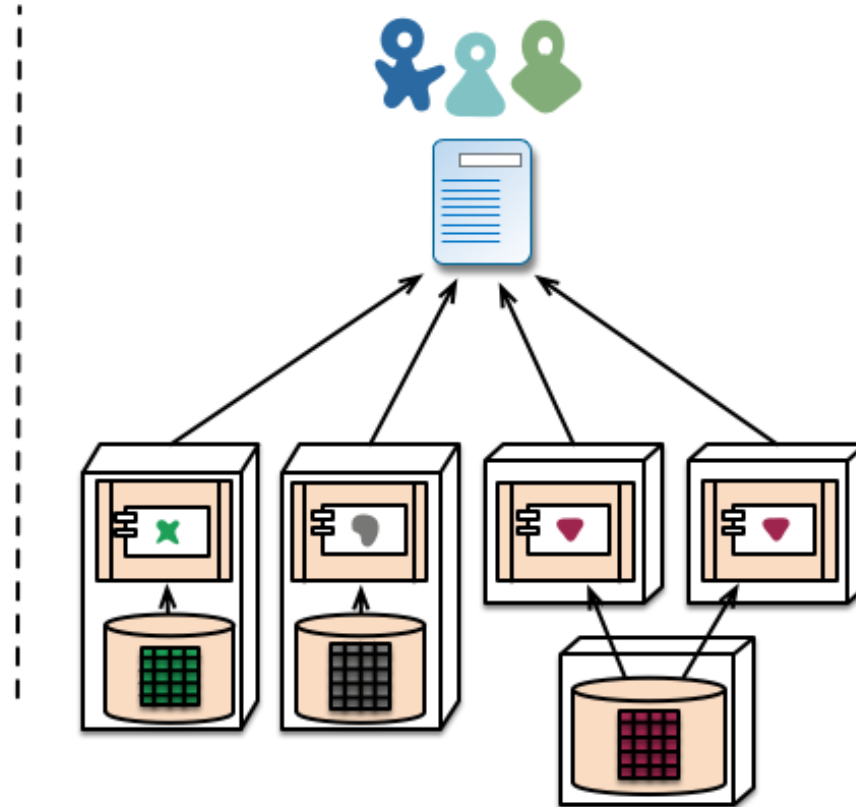
Source: <http://martinfowler.com/articles/microservices.html>



# Data Management and Consistency



monolith - single database



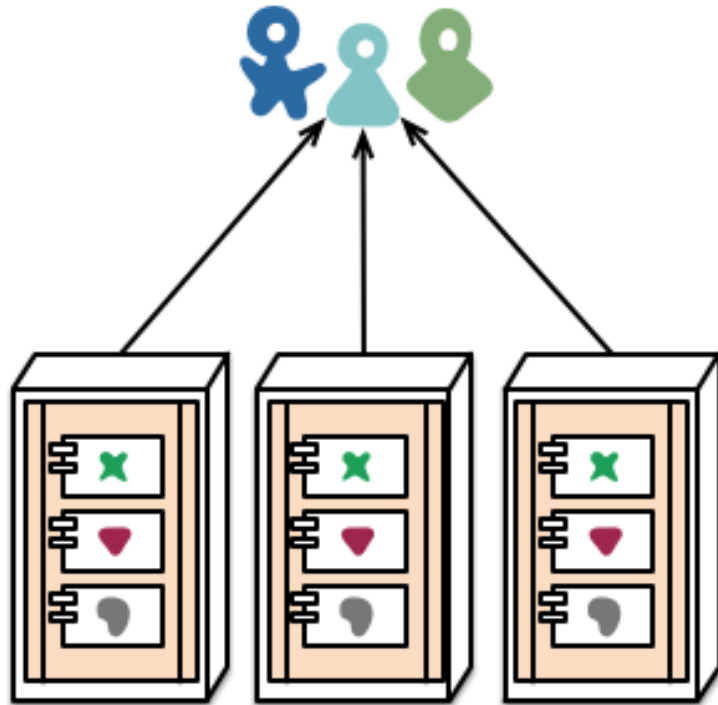
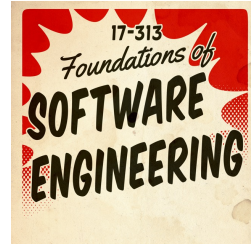
microservices - application databases

Source: <http://martinfowler.com/articles/microservices.html>

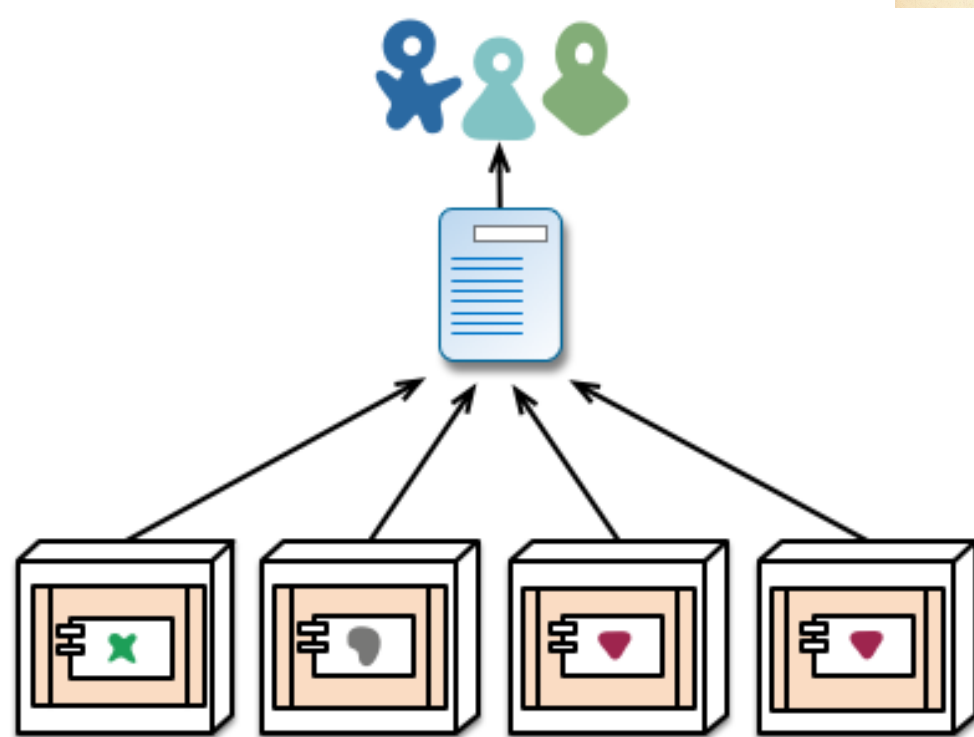




# Deployment and Evolution



monolith - multiple modules in the same process

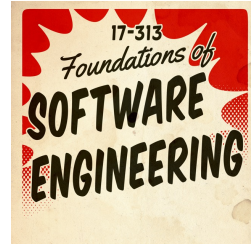


microservices - modules running in different processes

Source: <http://martinfowler.com/articles/microservices.html>



# Microservices



- Building applications as suite of small and easy to replace services
  - fine grained, one functionality per service (sometimes 3-5 classes)
  - composable
  - easy to develop, test, and understand
  - fast (re)start, fault isolation
  - modelled around business domain
- Interplay of different systems and languages
- Easily deployable and replicable
- Embrace automation, embrace faults
- Highly observable



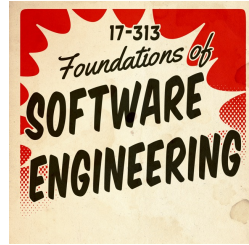
# Technical Considerations



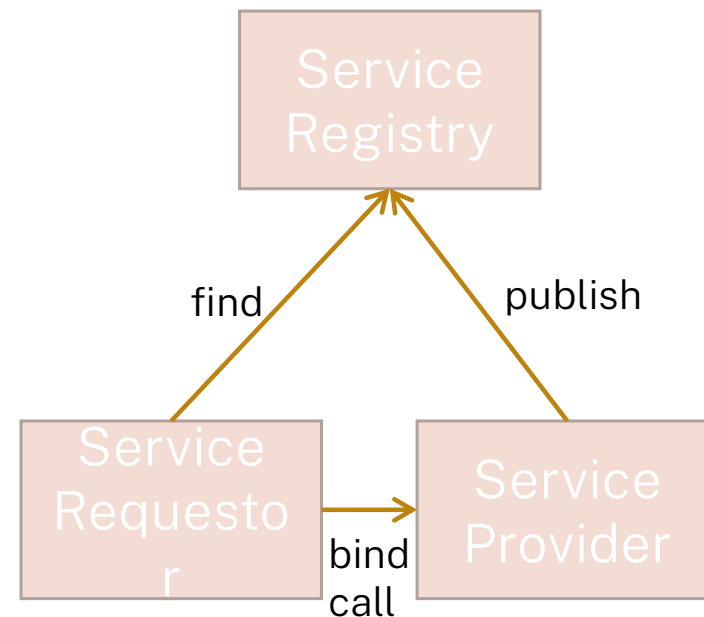
- HTTP/REST/JSON/GRPC/etc. communication
- Independent development and deployment
- Self-contained services (e.g., each with own database)
  - multiple instances behind load-balancer
- Streamline deployment



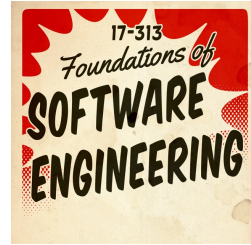
# Service Oriented Architectures (SOA)



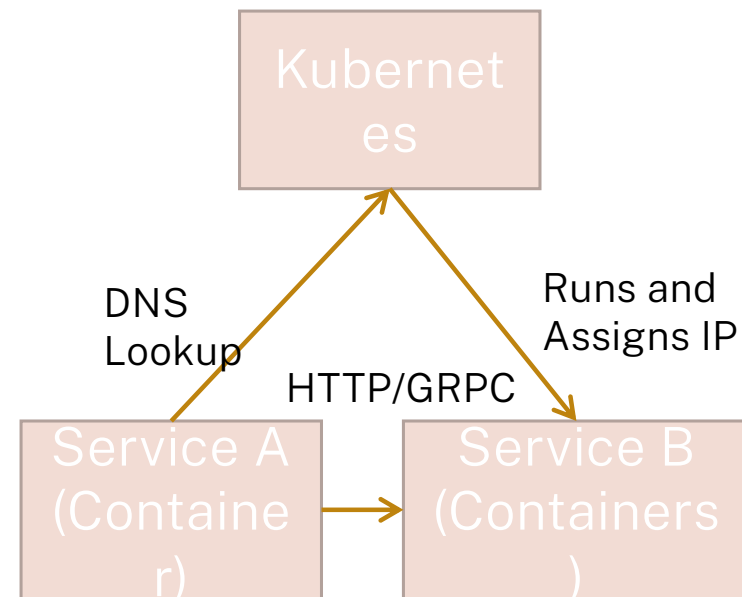
- Service: self-contained functionality
- Remote invocation, language-independent interface
- Dynamic lookup possible
- Often used to wrap legacy systems



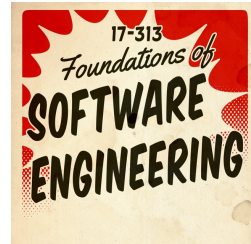
# ~~Service Oriented Architectures (SOA)~~ Microservice Architecture



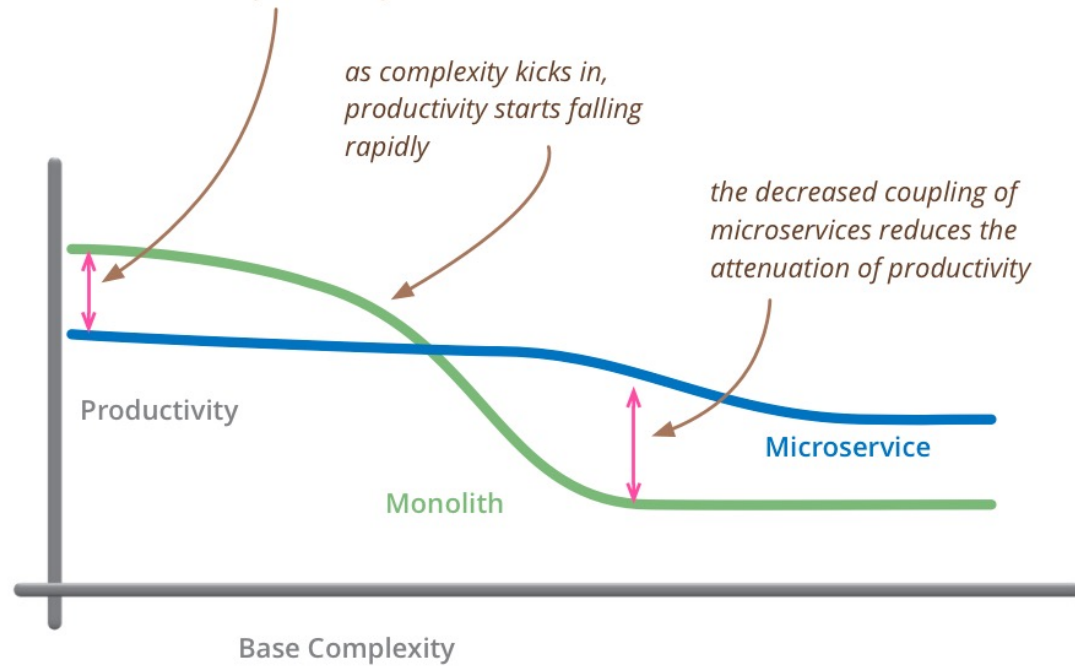
- Service: self-contained functionality
- Language-independent interface
- Dynamic lookup



# Microservices overhead



*for less-complex systems, the extra baggage required to manage microservices reduces productivity*



*but remember the skill of the team will outweigh any monolith/microservice choice*



# Microservice challenges



- Complexities of distributed systems
  - network latency, faults, inconsistencies
  - testing challenges
- Resource overhead, RPCs
  - Requires more thoughtful design (avoid "chatty" APIs, be more coarse-grained)\_
- Shifting complexities to the network
- Operational complexity
- Frequently adopted by breaking down monolithic application
- HTTP/REST/JSON communication



# Discussion of Microservices

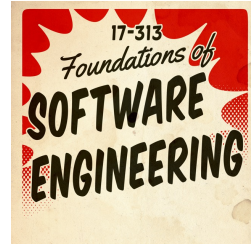


- Are they really “new”?
- Do microservices solve problems, or push them down the line?
- What are the impacts of the added flexibility?
- Beware of the cult (HackerNews-driven development?)
- “If you can’t build a well-structured monolith, what makes you think microservices is the answer?” – Simon Brown
- Leads to more API design decisions





# Serverless (Functions-as-a-Service)




- Instead of writing minimal services, write just functions
- No state, rely completely on cloud storage or other cloud services
- Pay-per-invocation billing with elastic scalability
- Drawback: more ways things can fail, state is expensive
- Examples:  
AWS lambda, CloudFlare workers, Azure Functions
- What might this be good for?
  
- (New in 2019/20) Stateful Functions:  
Azure Durable Entities, CloudFlare Durable Objects



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send [potantin](https://poll-ev.com/potantin) to 22333



Have you ever implemented a service or microservice before? 0

Yes **(A)**

No **(B)**

I don't know **(C)**

Can you repeat the question? **(D)**





DALL-E History Collections

Edit the detailed description

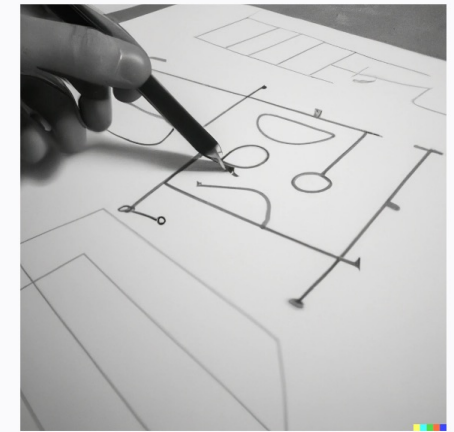
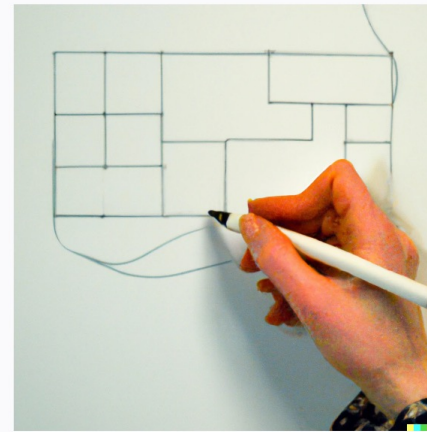
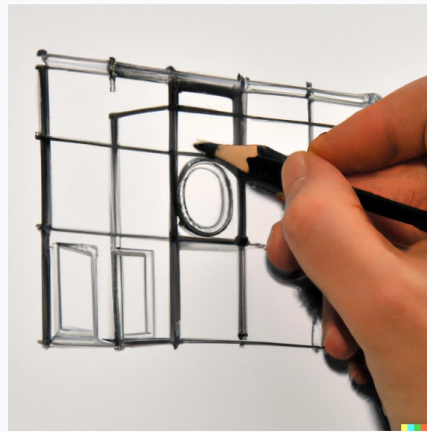
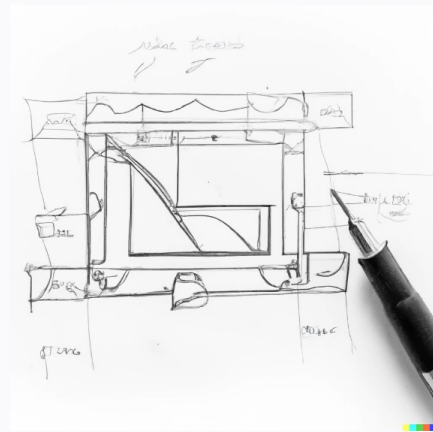
Surprise me

Upload



pencil drawing of a design example

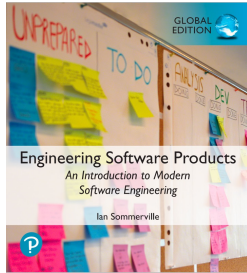
Generate



## Microservice Design Example



# A microservice example



- System authentication

- User registration, where users provide information about their identity, security information, mobile (cell) phone number and email address.
  - Authentication using UID/password.
  - Two-factor authentication using code sent to mobile phone.
  - User information management e.g. change password or mobile phone number.
  - Reset forgotten password.
- Each of these features could be implemented as a separate service that uses a central shared database to hold authentication information.
  - However, these features are too large to be microservices. To identify the microservices that might be used in the authentication system, you need to break down the coarse-grain features into more detailed functions.



# Functional breakdown of authentication features



## User registration

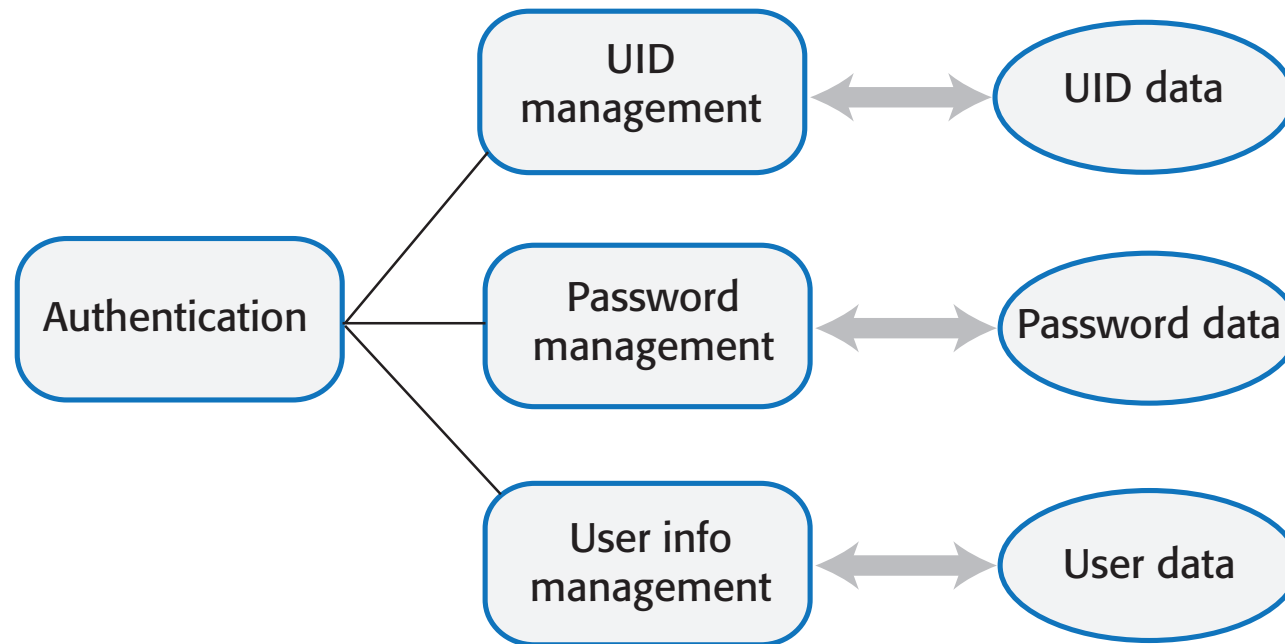
Setup new login id
Setup new password
Setup password recovery information
Setup two-factor authentication
Confirm registration

## Authenticate using UID/password

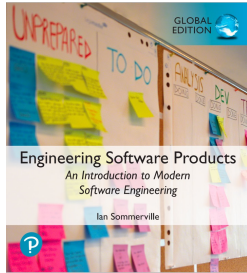
Get login id
Get password
Check credentials
Confirm authentication



# Authentication microservices



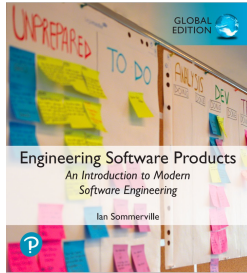
# Characteristics of microservices



- **Self-contained**  
Microservices do not have external dependencies. They manage their own data and implement their own user interface.
- **Lightweight**  
Microservices communicate using lightweight protocols, so that service communication overheads are low.
- **Implementation-independent**  
Microservices may be implemented using different programming languages and may use different technologies (e.g. different types of database) in their implementation.
- **Independently deployable**  
Each microservice runs in its own process and is independently deployable, using automated systems.
- **Business-oriented**  
Microservices should implement business capabilities and needs, rather than simply provide a technical service.



# Microservice communication

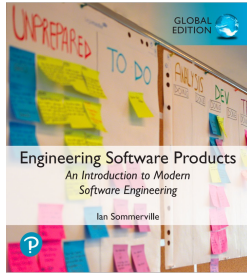


- Microservices communicate by exchanging messages.
- A message that is sent between services includes some administrative information, a service request and the data required to deliver the requested service.
- Services return a response to service request messages.
  - An authentication service may send a message to a login service that includes the name input by the user.
  - The response may be a token associated with a valid user name or might be an error saying that there is no registered user.





# Microservice characteristics



- A well-designed microservice should have high cohesion and low coupling.
  - Cohesion is a measure of the number of relationships that parts of a component have with each other. High cohesion means that all of the parts that are needed to deliver the component's functionality are included in the component.
  - Coupling is a measure of the number of relationships that one component has with other components in the system. Low coupling means that components do not have many relationships with other components.
- Each microservice should have a single responsibility i.e. it should do one thing only and it should do it well.
  - However, 'one thing only' is difficult to define in a way that's applicable to all services.
- Responsibility does not always mean a single, functional activity.

# Password management functionality



## User functions

Create password
Change password
Check password
Recover password

## Supporting functions

Check password validity
Delete password
Backup password database
Recover password database
Check database integrity
Repair password DB



# Microservice support code



## Microservice X

Service functionality	
Message management	Failure management
UI implementation	Data consistency management



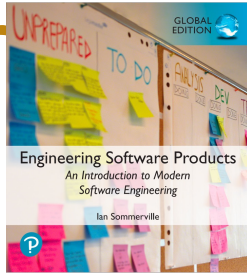
# Microservices architecture



- A microservices architecture is an *architectural style* – a tried and tested way of implementing a logical software architecture.
- This architectural style addresses two problems with **monolithic applications**
  - The whole system has to be rebuilt, re-tested and re-deployed when any change is made. This can be a slow process as changes to one part of the system can adversely affect other components.
  - As the demand on the system increases, the whole system has to be scaled, even if the demand is localized to a small number of system components that implement the most popular system functions.



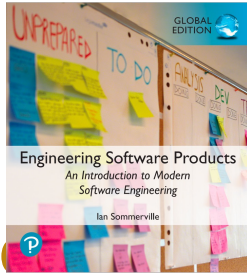
# Benefits of microservices architecture



- Microservices are self-contained and run in separate processes.
- In cloud-based systems, each microservice may be deployed in its own container. This means a microservice can be stopped and restarted without affecting other parts of the system.
- If the demand on a service increases, service replicas can be quickly created and deployed. These do not require a more powerful server so 'scaling-out' is, typically, much cheaper than 'scaling up'.



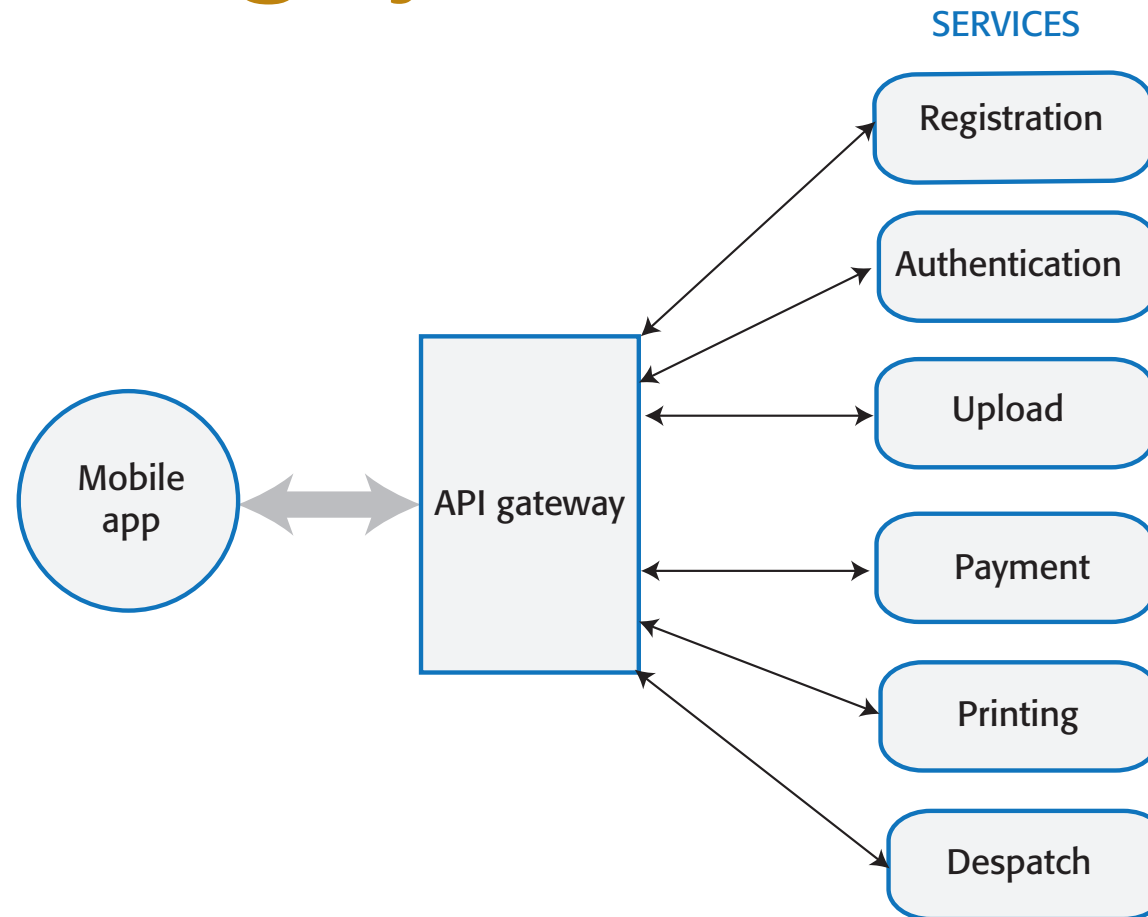
# A photo printing system for mobile



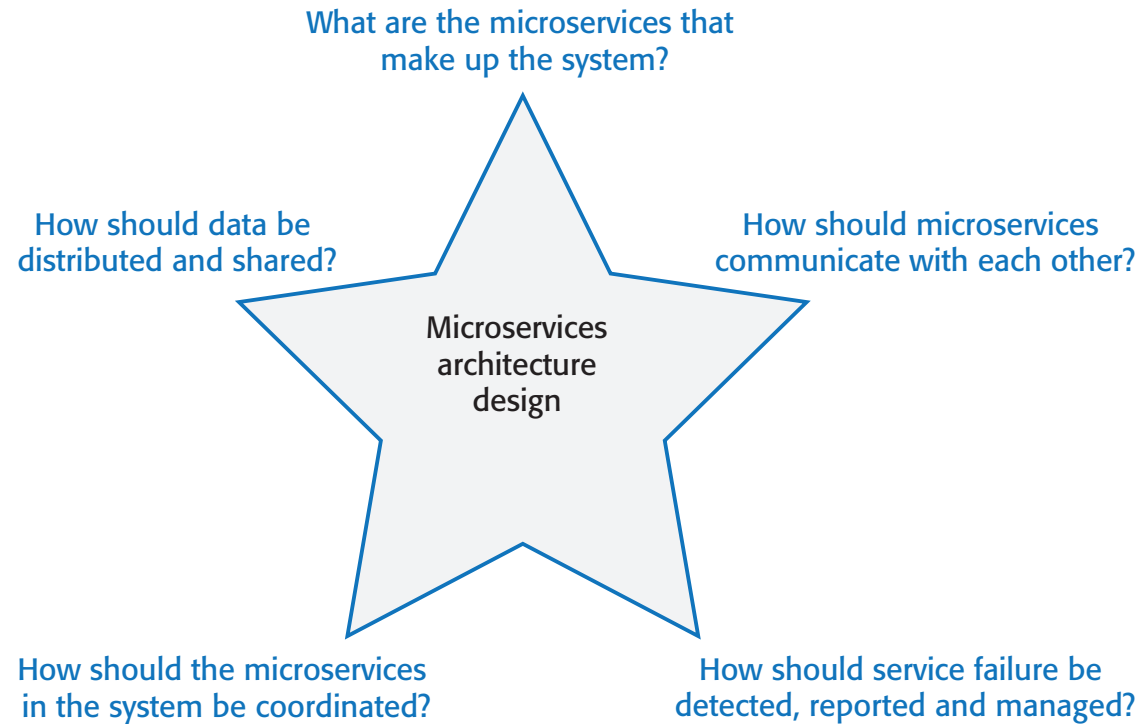
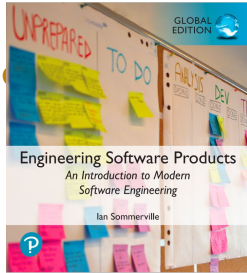
- Imagine that you are developing a photo printing service for mobile devices. Users can upload photos to your server from their phone or specify photos from their Instagram account that they would like to be printed. Prints can be made at different sizes and on different media.
- Users can choose print size and print medium. For example, they may decide to print a picture onto a mug or a T-shirt. The prints or other media are prepared and then posted to their home. They pay for prints either using a payment service such as Android or Apple Pay or by registering a credit card with the printing service provider.



# A microservices architecture for a photo printing system



# Microservices architecture - key design questions





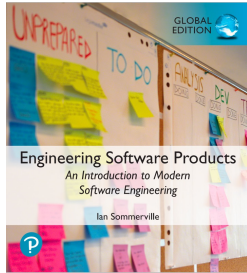
# Decomposition guidelines



- **Balance fine-grain functionality and system performance**
  - Single-function services mean that changes are limited to fewer services but require service communications to implement user functionality. This slows down a system because of the need for each service to bundle and unbundle messages sent from other services.
- **Follow the ‘common closure principle’**
  - Elements of a system that are likely to be changed at the same time should be located within the same service. Most new and changed requirements should therefore only affect a single service.
- **Associate services with business capabilities**
  - A business capability is a discrete area of business functionality that is the responsibility of an individual or a group. You should identify the services that are required to support each business capability.
- **Design services so that they only have access to the data that they need**
  - If there is an overlap between the data used by different services, you need a mechanism to propagate data changes to all services using the same data.



# Service communications

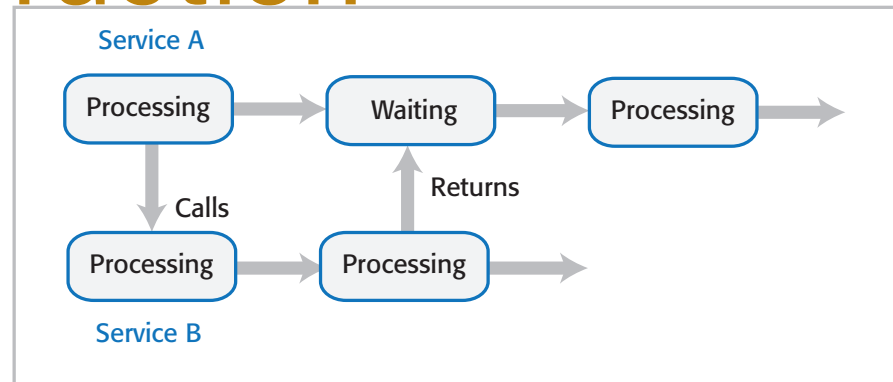


- Services communicate by exchanging messages that include information about the originator of the message, as well as the data that is the input to or output from the request.
- When you are designing a microservices architecture, you have to establish a standard for communications that all microservices should follow. Some of the key decisions that you have to make are
  - should service interaction be synchronous or asynchronous?
  - should services communicate directly or via message broker middleware?
  - what protocol should be used for messages exchanged between services?

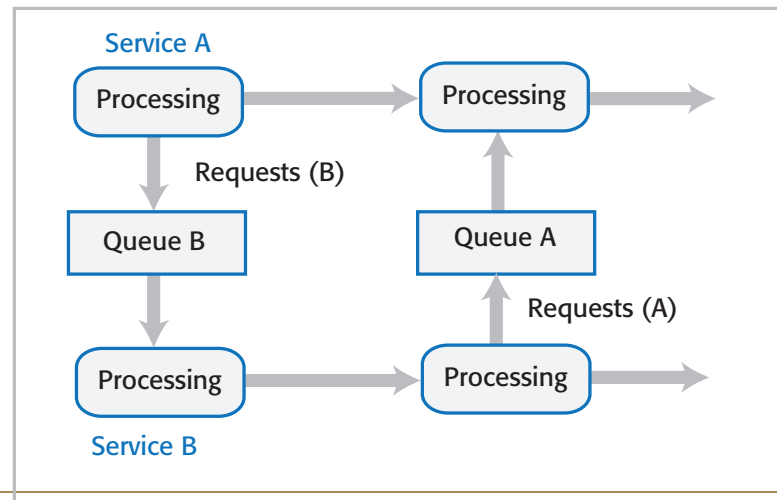


# Synchronous and asynchronous microservice interaction

Synchronous - A waits for B



Asynchronous - A and B execute concurrently



# Synchronous and asynchronous interaction

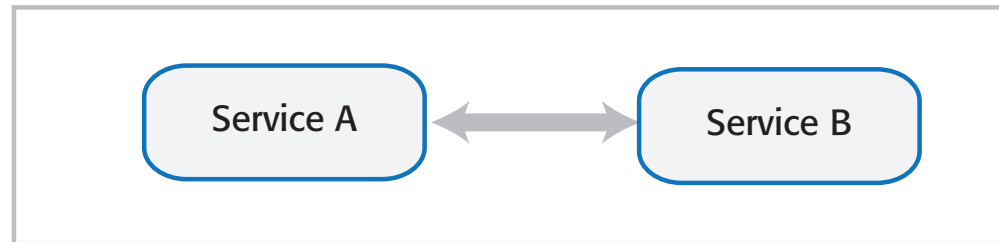
- In a synchronous interaction, service A issues a request to service B. Service A then suspends processing while B is processing the request.
- It waits until service B has returned the required information before continuing execution.
- In an asynchronous interaction, service A issues the request that is queued for processing by service B. A then continues processing without waiting for B to finish its computations.
- Sometime later, service B completes the earlier request from service A and queues the result to be retrieved by A.
- Service A, therefore, has to check its queue periodically to see if a result is available.



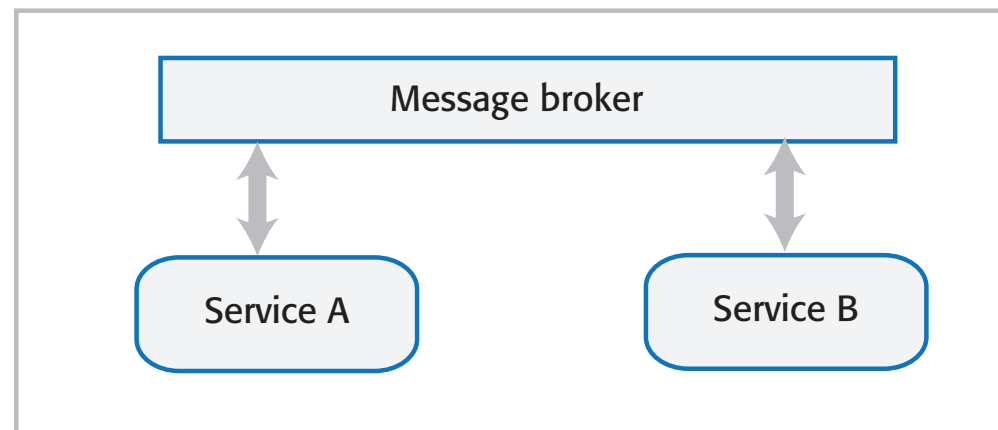
# Direct and indirect service communication



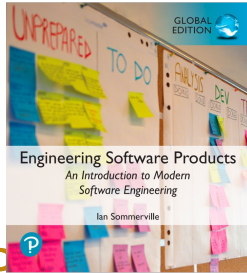
Direct communication - A and B send messages to each other



Indirect communication - A and B communicate through a message broker



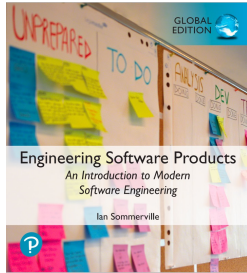
# Direct and indirect service communication



- Direct service communication requires that interacting services know each other's address.
- The services interact by sending requests directly to these addresses.
- Indirect communication involves naming the service that is required and sending that request to a message broker (sometimes called a message bus).
- The message broker is then responsible for finding the service that can fulfil the service request.



# Microservice data design



- You should isolate data within each system service with as little data sharing as possible.
- If data sharing is unavoidable, you should design microservices so that most sharing is ‘read-only’, with a minimal number of services responsible for data updates.
- If services are replicated in your system, you must include a mechanism that can keep the database copies used by replica services consistent.



# Inconsistency management



- An ACID (Atomicity, Consistency, Isolation, Durability) transaction bundles a set of data updates into a single unit so that either all updates are completed or none of them are. ACID transactions are impractical in a microservices architecture.
- The databases used by different microservices or microservice replicas need not be completely consistent all of the time.
- **Dependent data inconsistency**
  - The actions or failures of one service can cause the data managed by another service to become inconsistent.
- **Replica inconsistency**
  - There are several replicas of the same service that are executing concurrently. These all have their own database copy and each updates its own copy of the service data. You need a way of making these databases 'eventually consistent' so that all replicas are working on the same data.





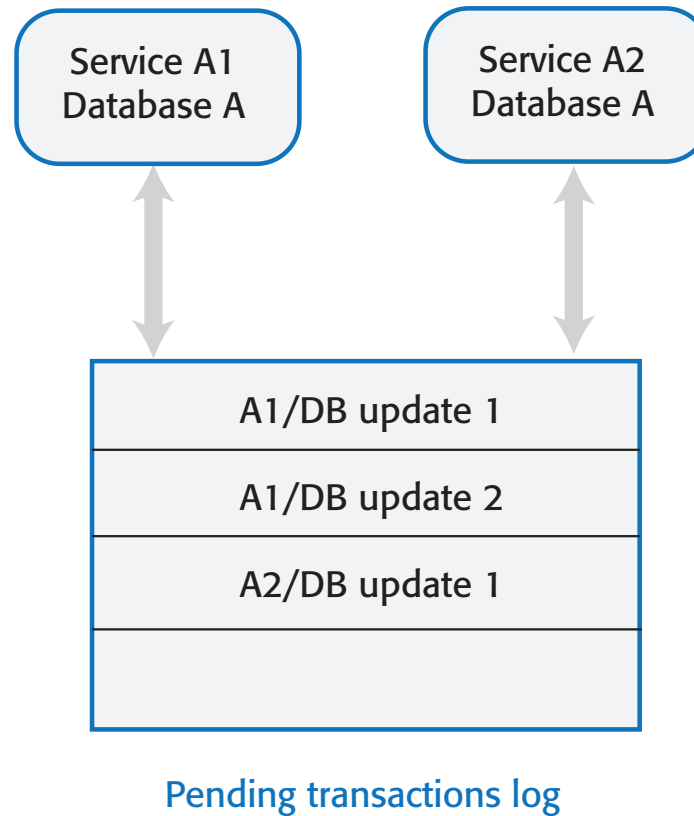
# Eventual consistency



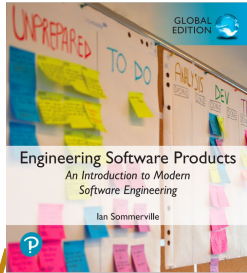
- Eventual consistency is a situation where the system guarantees that the databases will eventually become consistent.
- You can implement eventual consistency by maintaining a transaction log.
- When a database change is made, this is recorded on a 'pending updates' log.
- Other service instances look at this log, update their own database and indicate that they have made the change.



# Using a pending transaction log



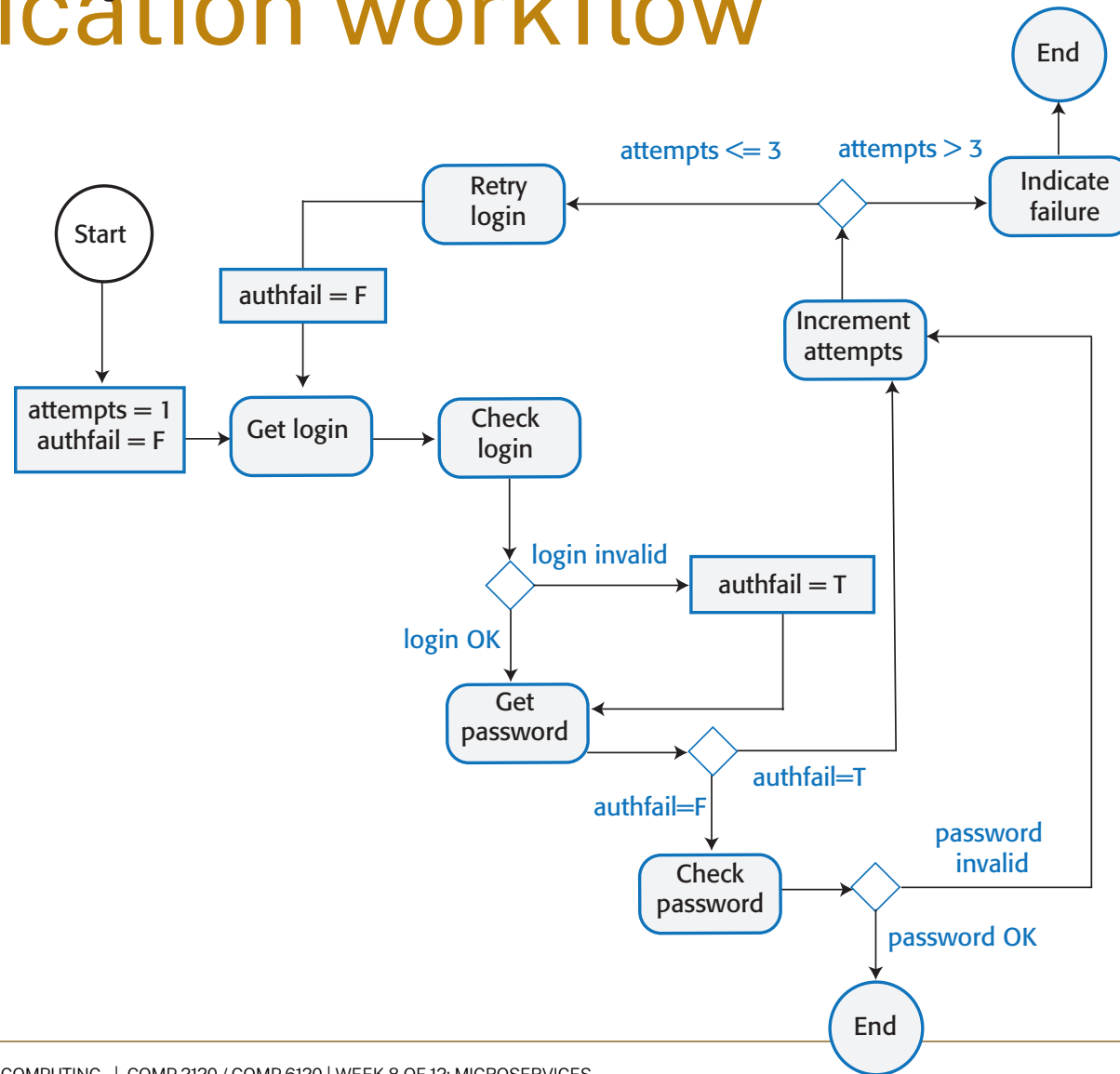
# Service coordination



- Most user sessions involve a series of interactions in which operations have to be carried out in a specific order.
- This is called a workflow.
  - An authentication workflow for UID/password authentication shows the steps involved in authenticating a user.
  - In this example, the user is allowed 3 login attempts before the system indicates that the login has failed.

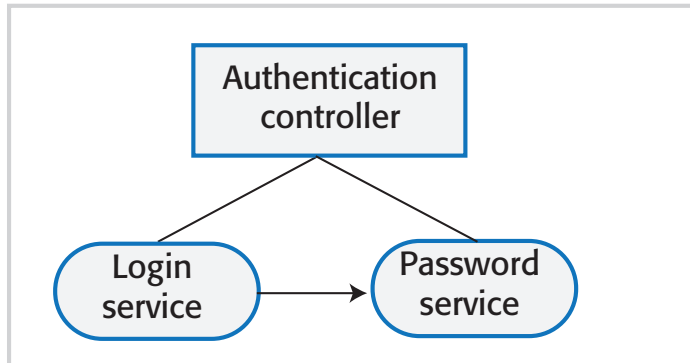


# Authentication workflow

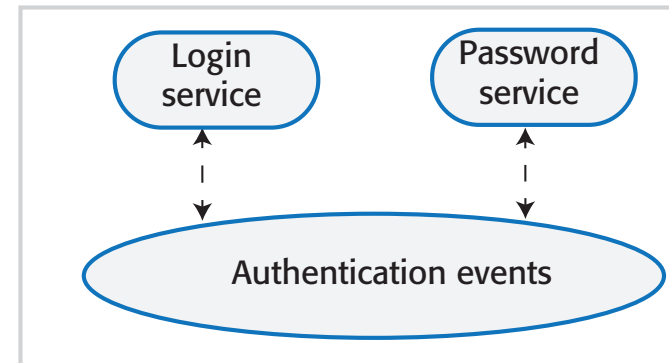


# Orchestration and choreography

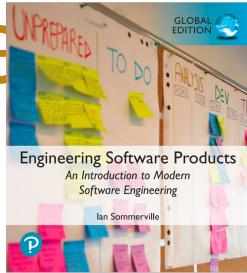
Service orchestration



Service choreography



# Failure types in a microservices system



- **Internal service failure**

These are conditions that are detected by the service and can be reported to the service client in an error message. An example of this type of failure is a service that takes a URL as an input and discovers that this is an invalid link.

- **External service failure**

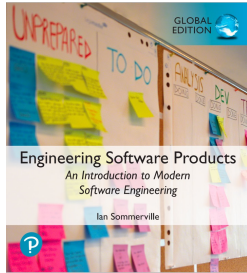
These failures have an external cause, which affects the availability of a service. Failure may cause the service to become unresponsive and actions have to be taken to restart the service.

- **Service performance failure**

The performance of the service degrades to an unacceptable level. This may be due to a heavy load or an internal problem with the service. External service monitoring can be used to detect performance failures and unresponsive services.



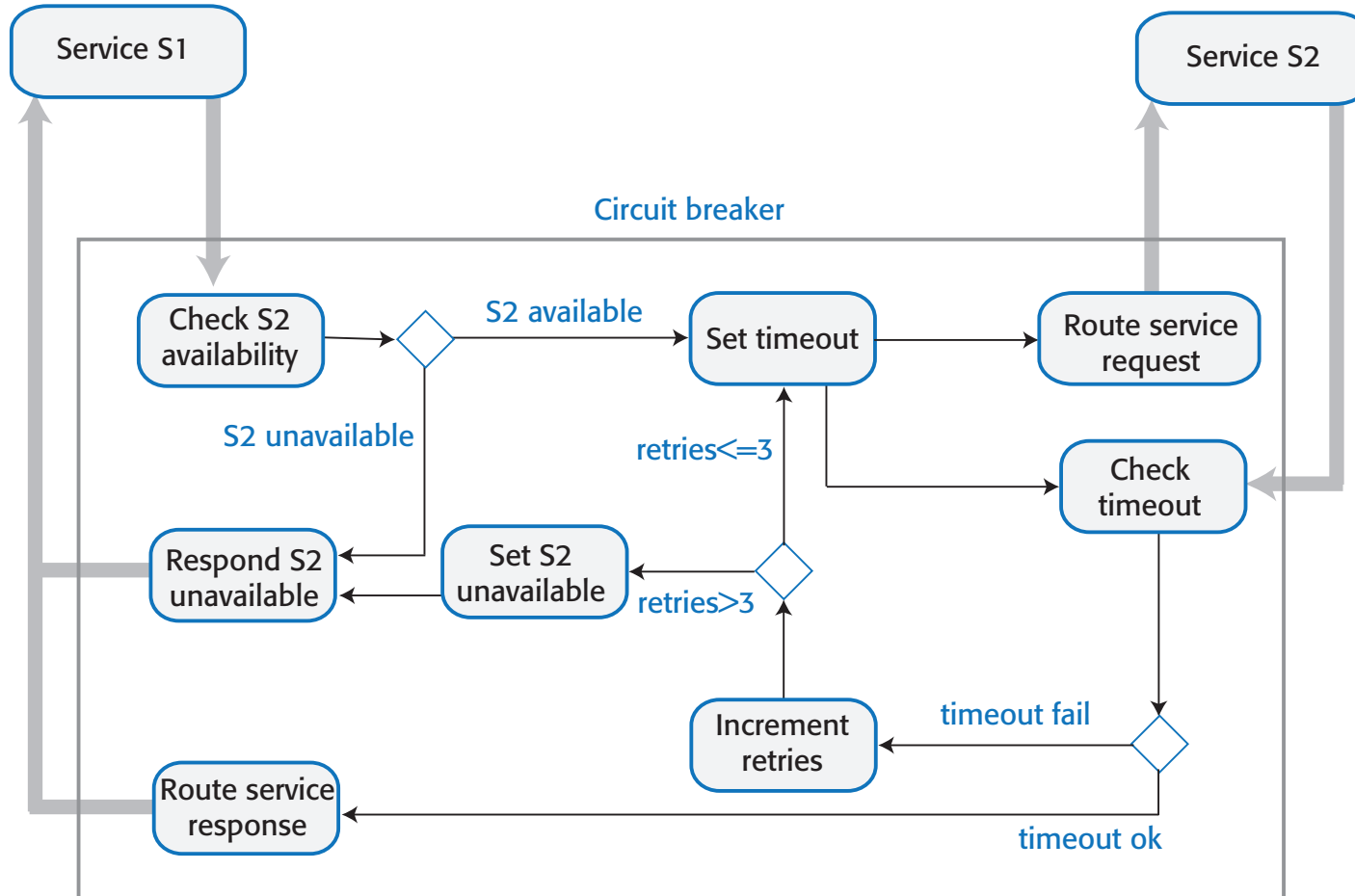
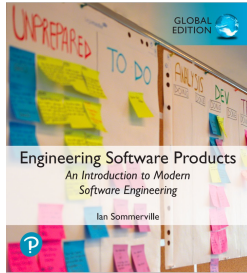
# Timeouts and circuit breakers



- A timeout is a counter that is associated with the service requests and starts running when the request is made.
- Once the counter reaches some predefined value, such as 10 seconds, the calling service assumes that the service request has failed and acts accordingly.
- The problem with the timeout approach is that every service call to a 'failed service' is delayed by the timeout value so the whole system slows down.
- Instead of using timeouts explicitly when a service call is made, he suggests using a circuit breaker. Like an electrical circuit breaker, this immediately denies access to a failed service without the delays associated with timeouts.




# Using a circuit breaker to cope with service failure






# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333**



Can ACID transactions be used to guarantee eventual consistency? 

Yes **(A)**

No **(B)**

I don't know **(C)**

Can you repeat the question? **(D)**



Edit the detailed description

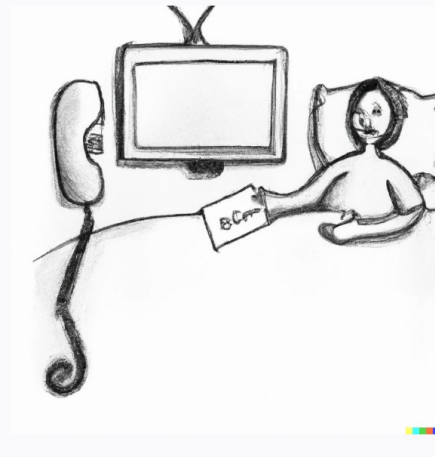
Surprise me

Upload



pencil drawing of RESTful services

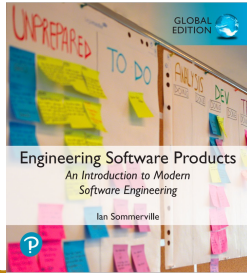
Generate



# RESTful Services



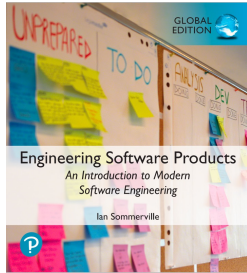
# RESTful services



- The REST (REpresentational State Transfer) architectural style is based on the idea of transferring representations of digital resources from a server to a client.
  - You can think of a resource as any chunk of data such as credit card details, an individual's medical record, a magazine or newspaper, a library catalogue, and so on.
  - Resources are accessed via their unique URI and RESTful services operate on these resources.
- This is the fundamental approach used in the web where the resource is a page to be displayed in the user's browser.
  - An HTML representation is generated by the server in response to an HTTP GET request and is transferred to the client for display by a browser or a special-purpose app.



# RESTful service principles



- **Use HTTP verbs**

The basic methods defined in the HTTP protocol (GET, PUT, POST, DELETE) must be used to access the operations made available by the service.

- **Stateless services**

Services must never maintain internal state. As I have already explained, microservices are stateless so fit with this principle.

- **URI addressable**

All resources must have a URI, with a hierarchical structure, that is used to access sub-resources.

- **Use XML or JSON**

Resources should normally be represented in JSON or XML or both. Other representations, such as audio and video representations, may be used if appropriate.



# RESTful service operations



- **Create**

Implemented using HTTP POST, which creates the resource with the given URI. If the resource has already been created, an error is returned.

- **Read**

Implemented using HTTP GET, which reads the resource and returns its value. GET operations should never update a resource so that successive GET operations with no intervening PUT operations always return the same value.

- **Update**

Implemented using HTTP PUT, which modifies an existing resource. PUT should not be used for resource creation.

- **Delete**

Implemented using HTTP DELETE, which makes the resource inaccessible using the specified URI. The resource may or may not be physically deleted.



# Road information system

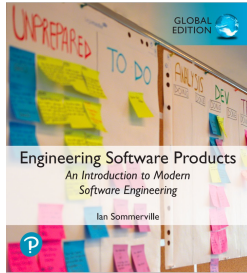


- Imagine a system that maintains information about incidents, such as traffic collisions, roadworks and accidents on a national road network. This system can be accessed via a browser using the URL:
  - <https://trafficinfo.net/incidents/> (not a real link!)
- Users can query the system to discover incidents on the roads on which they are planning to travel.
- When implemented as a RESTful web service, you need to design the resource structure so that incidents are organized hierarchically.
  - For example, incidents may be recorded according to the road identifier (e.g. A90), the location (e.g. stonehaven), the carriageway direction (e.g. north) and an incident number (e.g. 1). Therefore, each incident can be accessed using its URI:
    - <https://trafficinfo.net/incidents/A90/stonehaven/north/1> (not a real link!)

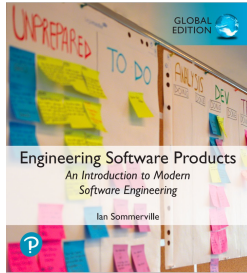


# Incident description

- Incident ID: A90N17061714391
- Date: 17 June 2017
- Time reported: 1439
- Severity: Significant
- Description: Broken-down bus on north carriageway. One lane closed. Expect delays of up to 30 minutes



# Service operations



- **Retrieve**

- Returns information about a reported incident or incidents. Accessed using the GET verb.

- **Add**

- Adds information about a new incident. Accessed using the POST verb.

- **Update**

- Updates the information about a reported incident. Accessed using the PUT verb.

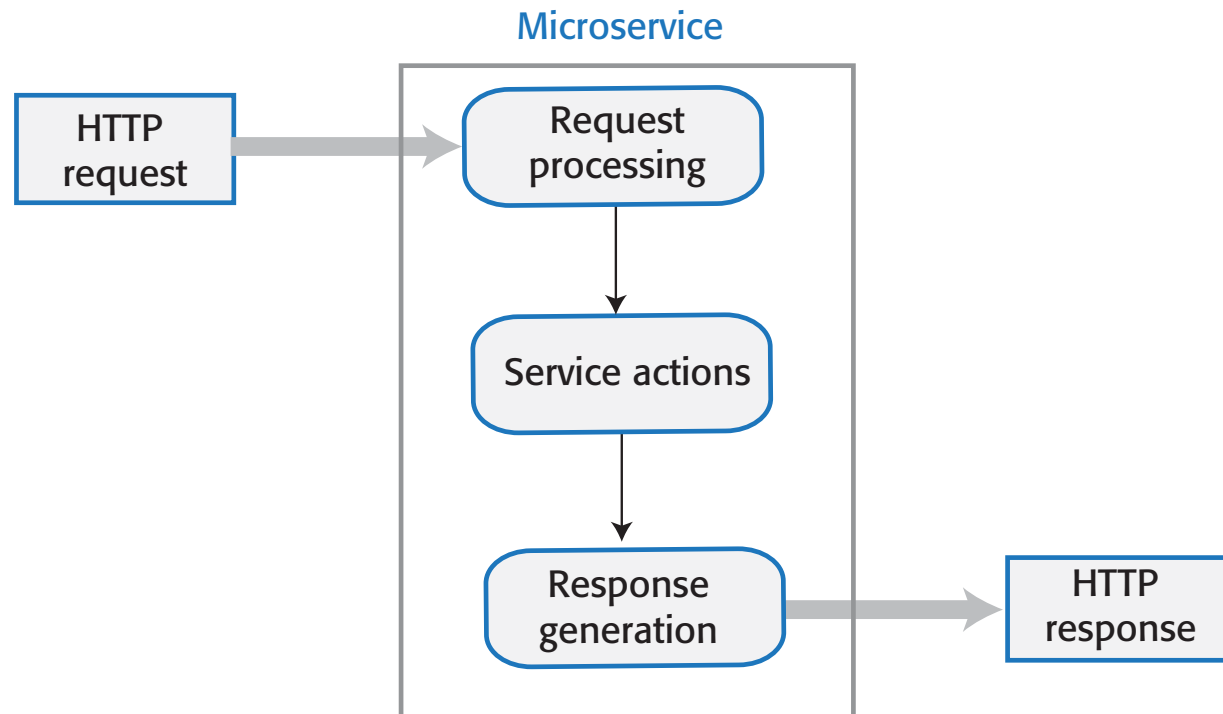
- **Delete**

- Deletes an incident. The DELETE verb is used when an incident has been cleared.





# HTTP request and response process



# HTTP request and response message organization



## REQUEST

[HTTP verb]	[URI]	[HTTP version]
[Request header]		
[Request body]		

## RESPONSE

[HTTP version]	[Response code]
[Response header]	
[Response body]	



# XML and JSON descriptions

## JSON

```
{  
  id: "A90N17061714391",  
  "date": "20170617",  
  "time": "1437",  
  "road_id": "A90",  
  "place": "Stonehaven",  
  "direction": "north",  
  "severity": "significant",  
  "description": "Broken-down bus on north carriageway.  
One lane closed. Expect delays of up to 30 minutes."  
}
```



# XML and JSON descriptions

## XML

```
<id>
A90N17061714391
</id>
<date>
20170617
</date>
<time>
1437
</time>
```

...

```
<description>Broken-down bus on north carriageway. One
lane closed. Expect delays of up to 30 minutes.
</description>
```

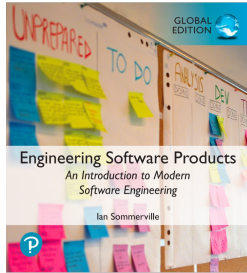


# A GET request and the associated response

REQUEST			RESPONSE	
GET	incidents/A90/stonehaven/	HTTP/1.1	HTTP/1.1	200
Host: trafficinfo.net ... Accept: text/json, text/xml, text/plain Content-Length: 0			... Content-Length: 461 Content-Type: text/json	
			<pre>{   "number": "A90N17061714391",   "date": "20170617",   "time": "1437",   "road_id": "A90",   "place": "Stonehaven",   "direction": "north",   "severity": "significant",   "description": "Broken-down bus on north carriageway. One lane closed. Expect delays of up to 30 minutes." } {   "number": "A90S17061713001",   "date": "20170617",   "time": "1300",   "road_id": "A90",   "place": "Stonehaven",   "direction": "south",   "severity": "minor",   "description": "Grass cutting on verge. Minor delays" }</pre>	



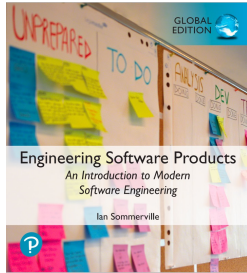
# Service deployment



- After a system has been developed and delivered, it has to be deployed on servers, monitored for problems and updated as new versions become available.
- When a system is composed of tens or even hundreds of microservices, deployment of the system is more complex than for monolithic systems.
- The service development teams decide which programming language, database, libraries and other support software should be used to implement their service. Consequently, there is no 'standard' deployment configuration for all services.
- It is now normal practice for microservice development teams to be responsible for deployment and service management as well as software development and to use continuous deployment.
- Continuous deployment means that as soon as a change to a service has been made and validated, the modified service is redeployed.



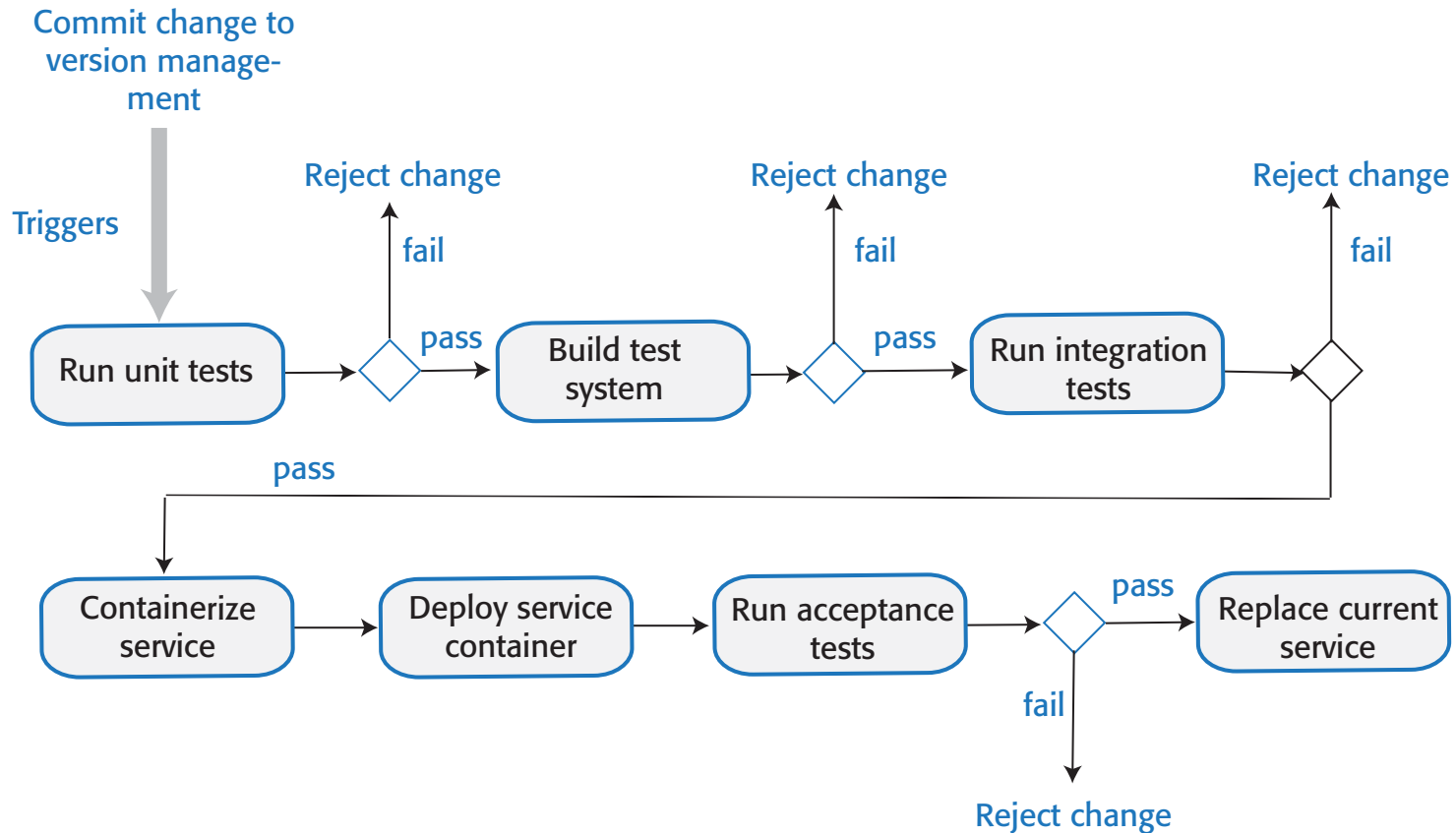
# Deployment automation



- Continuous deployment depends on automation so that as soon as a change is committed, a series of automated activities is triggered to test the software.
- If the software ‘passes’ these tests, it then enters another automation pipeline that packages and deploys the software.
- The deployment of a new service version starts with the programmer committing the code changes to a code management system such as Git.
- This triggers a set of automated tests that run using the modified service. If all service tests run successfully, a new version of the system that incorporates the changed service is created.
- Another set of automated system tests are then executed. If these run successfully, the service is ready for deployment.

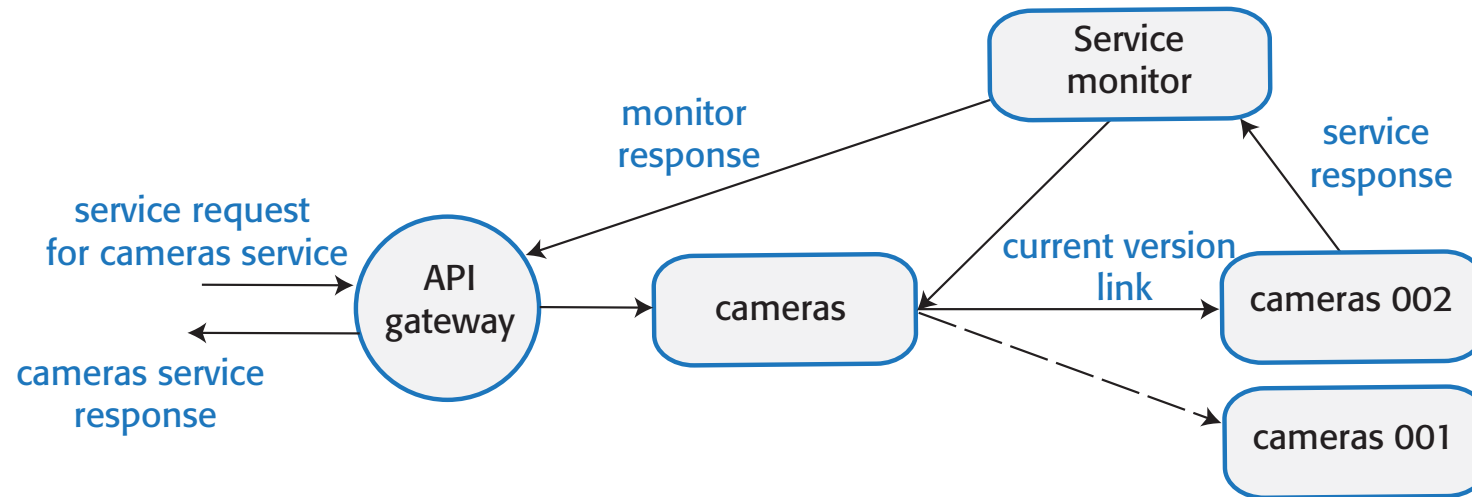


# A continuous deployment pipeline







# Versioned services



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333**



Have you ever written code that used POST/GET/PUT/DELETE (e.g. HTML Form) before?  0

Yes **(A)**

No **(B)**

I still do not know **(C)**

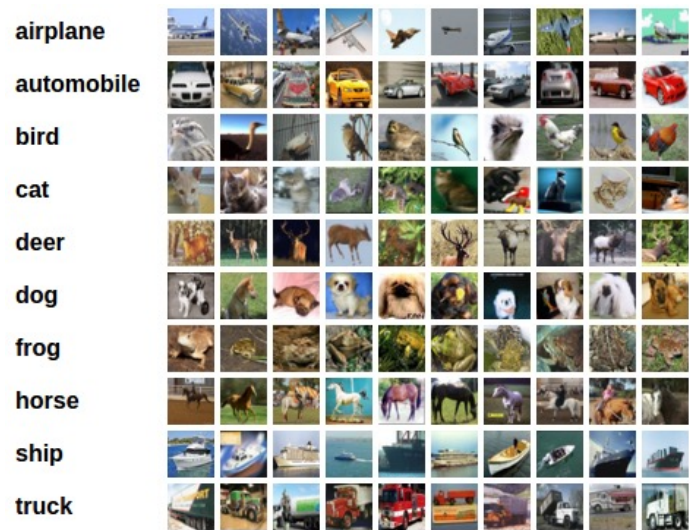


# Machine Learning Microservices



# Machine Learning in One Slide

(Supervised)



Lots of labelled data  
(Inputs, outputs)



Training



Model



Input



Output

“Bird”



Input



Output

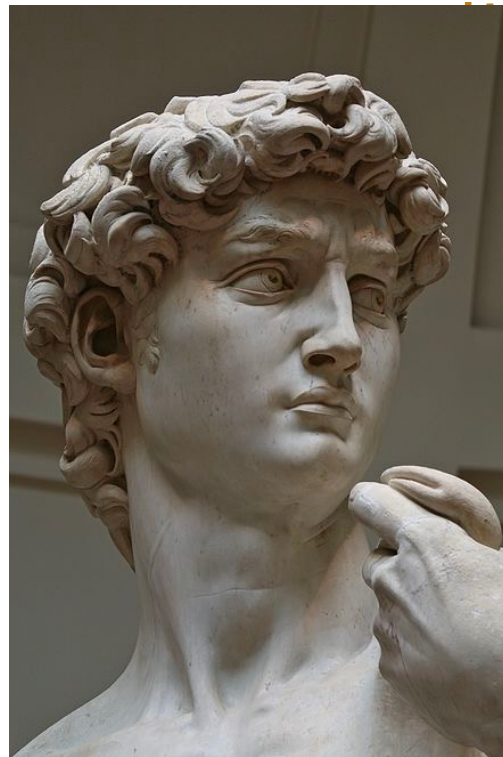
“Bird”



# Traditional Software Development



“It is easy. You just chip away the stone that doesn’t look like David.” –(probably not) Michelangelo

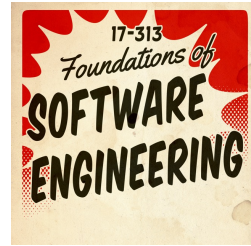


# ML Development

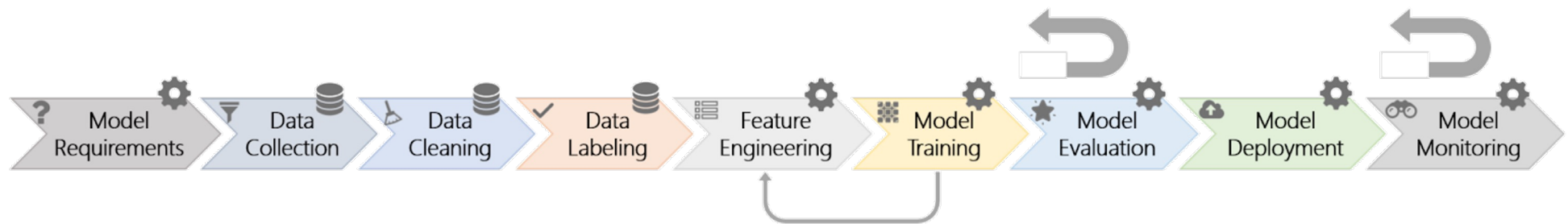
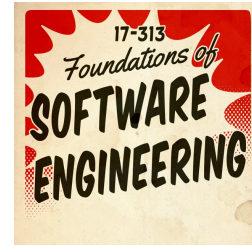
- Observation
- Hypothesis
- Predict
- Test
- Reject or Refine Hypothesis



# Black-box View of Machine Learning



# Microsoft's view of Software Engineering for ML



Source: "Software Engineering for Machine Learning: A Case Study" by Amershi et al. [1]





# Three Fundamental Differences:

- Data discovery and management
- Customization and Reuse
- No modular development of model itself



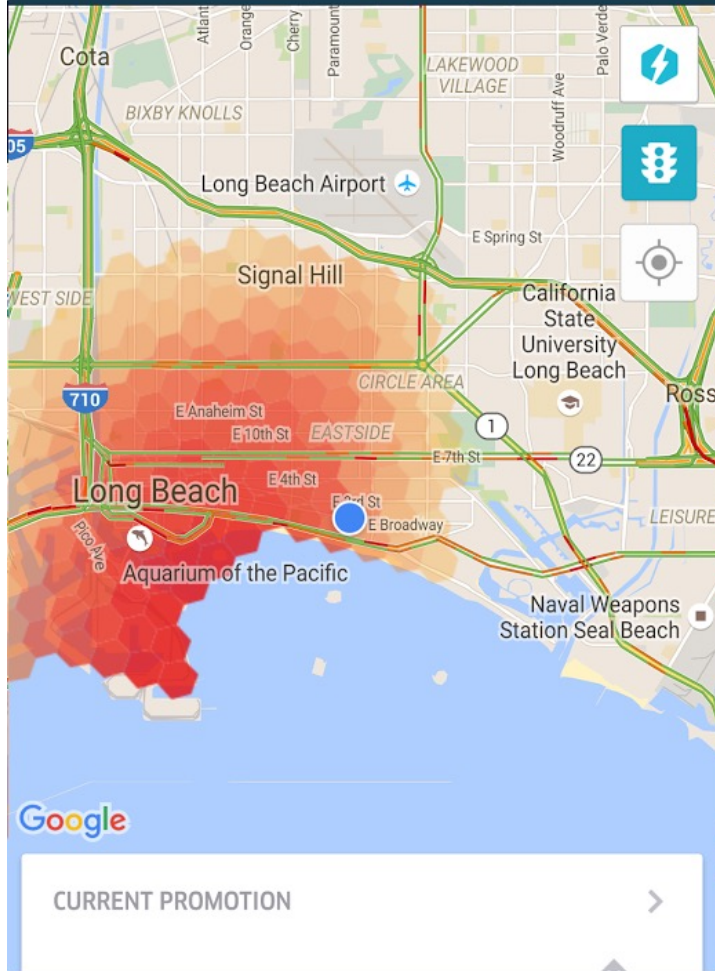
現金のみ

← Japanese ↔ English  
**cash only**

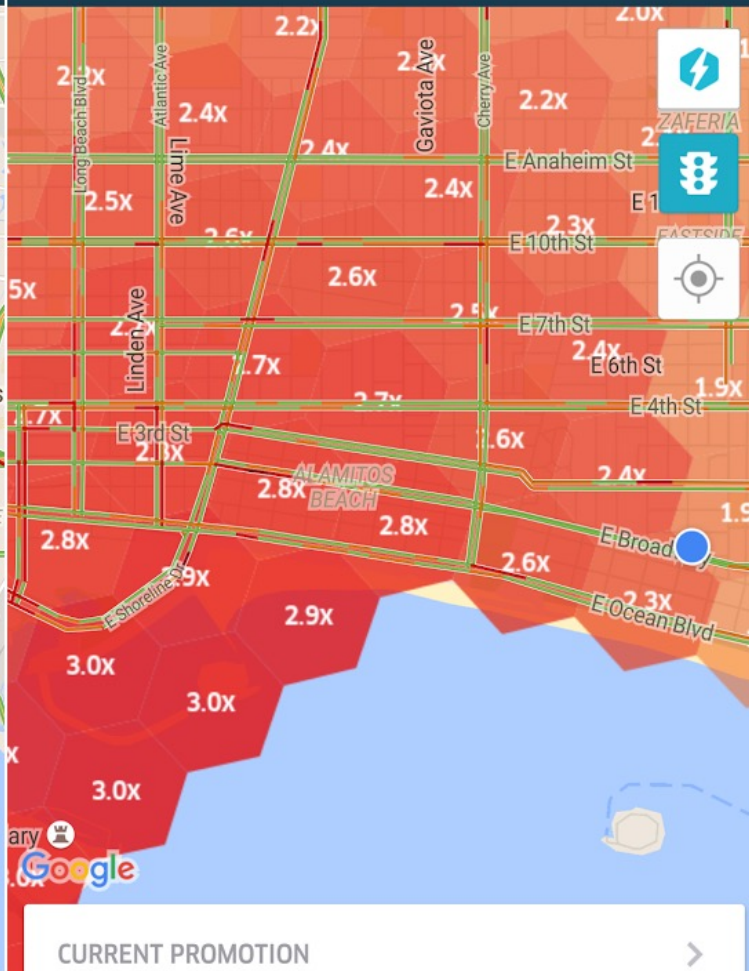




GO OFFLINE



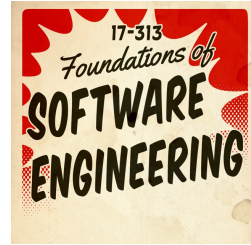
GO OFFLINE



A circular inset map showing a specific location in Long Beach, CA. The map is centered on Temple Ave and E Colorado St. A green pin is placed at the intersection. Below the map, the text reads '4 MINUTES' and 'Ave, Long Beach, CA 90814, USA'. At the bottom, there is a rating of '5.0 ★', a 'POOL' icon, and a multiplier of '1.9X'.

- HOME
- EARNINGS
- RATINGS
- ACCOUNT

# Typical ML Pipeline



- **Static**

- Get labeled data (data collection, cleaning and, labeling)
- Identify and extract features (feature engineering)
- Split data into training and evaluation set
- Learn model from training data (model training)
- Evaluate model on evaluation data (model evaluation)
- Repeat, revising features

- **with production data**

- Evaluate model on production data; monitor (model monitoring)
- Select production data for retraining (model training + evaluation)
- Update model regularly (model deployment)



# Example Data



OCR Helper Tool

Input Image: C:\tmp\MyHandWriting.jpg (Re)Process

Model Params: Load Model

0 Blobs selected

Hover controls for tooltips

- Show Binarized Image
- Show Rows
- Binarization Threshold: 200
- Height Merge Sensitivity: 15
- Width Merge Sensitivity: 10
- Pre Merge Filter Size: 10
- Post Merge Filter Size: 100
- Extracted Back Color: 0

Move Selected Blobs

Interval: 2

Export

Export Size (W/H): 20

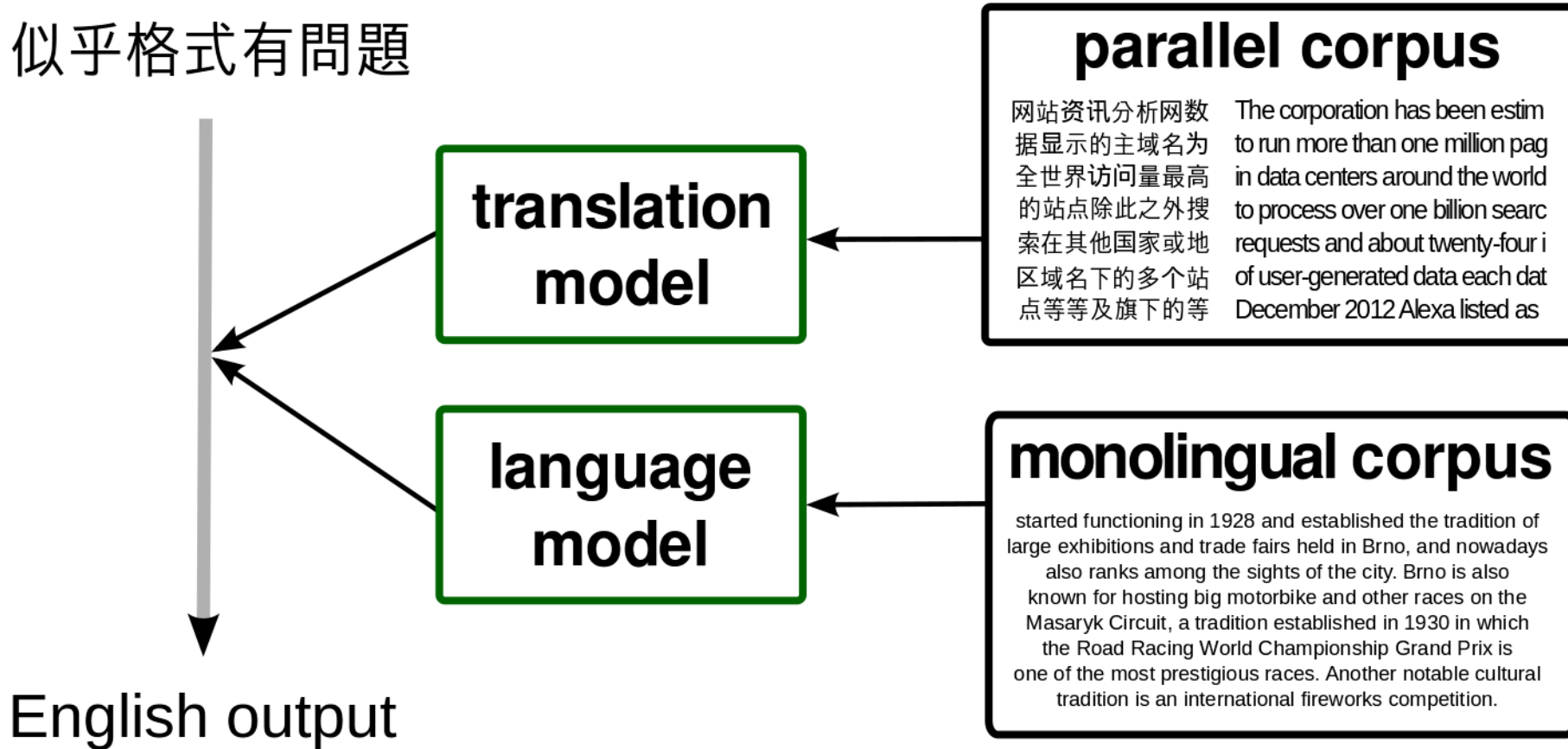
Output:

Export Blobs

# Learning Data



似乎格式有問題





# Feature Engineering



- Identify parameters of interest that a model may learn on
- Convert data into a useful form
- Normalize data
- Include context
- Remove misleading things



Features?

現金のみ

← Japanese ↔ English  
**cash only**

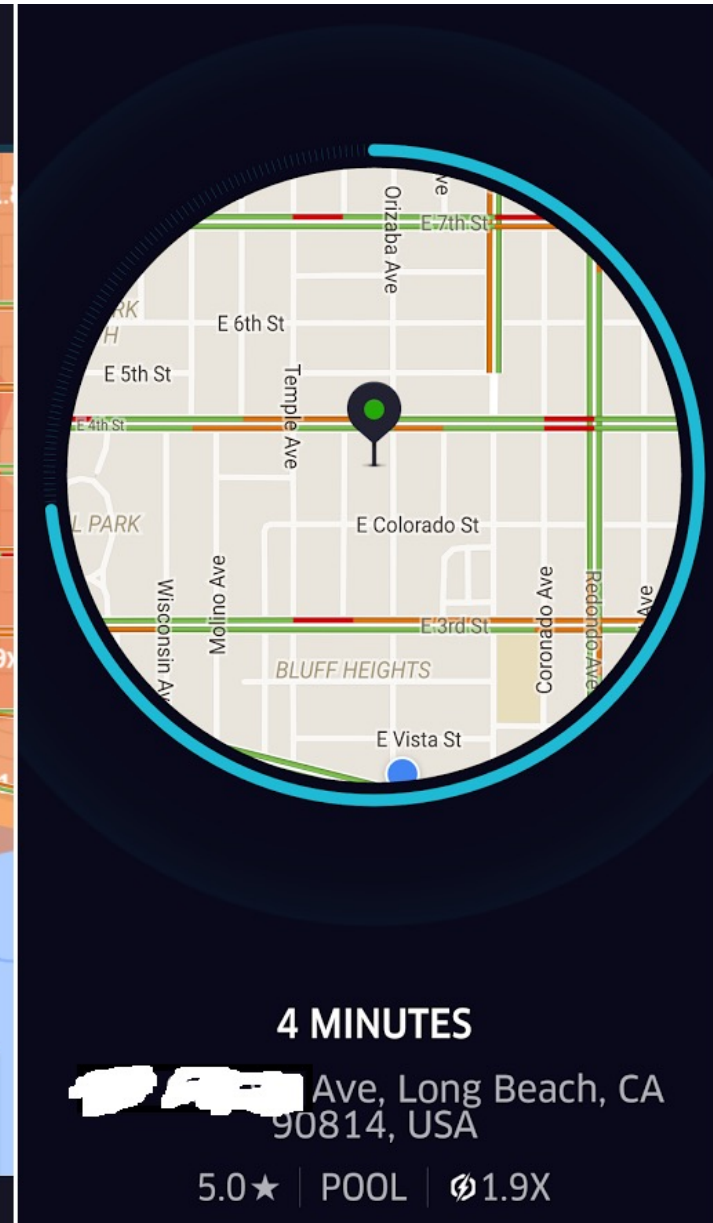
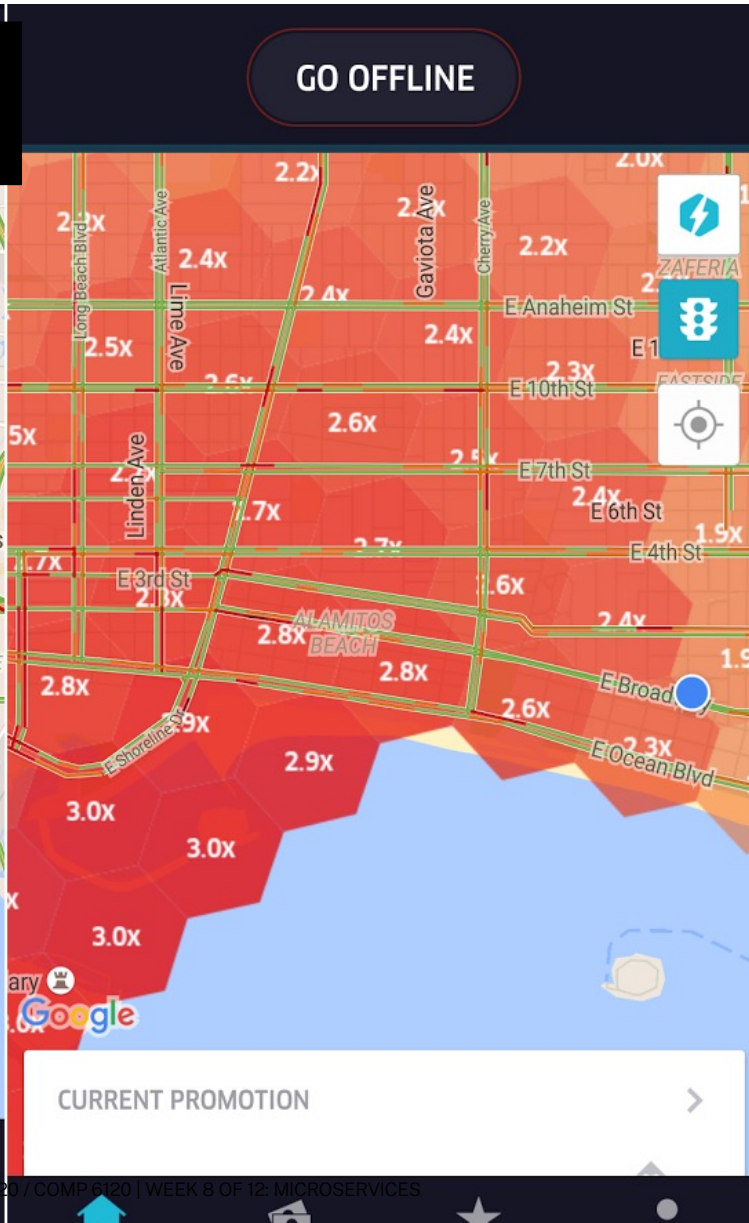
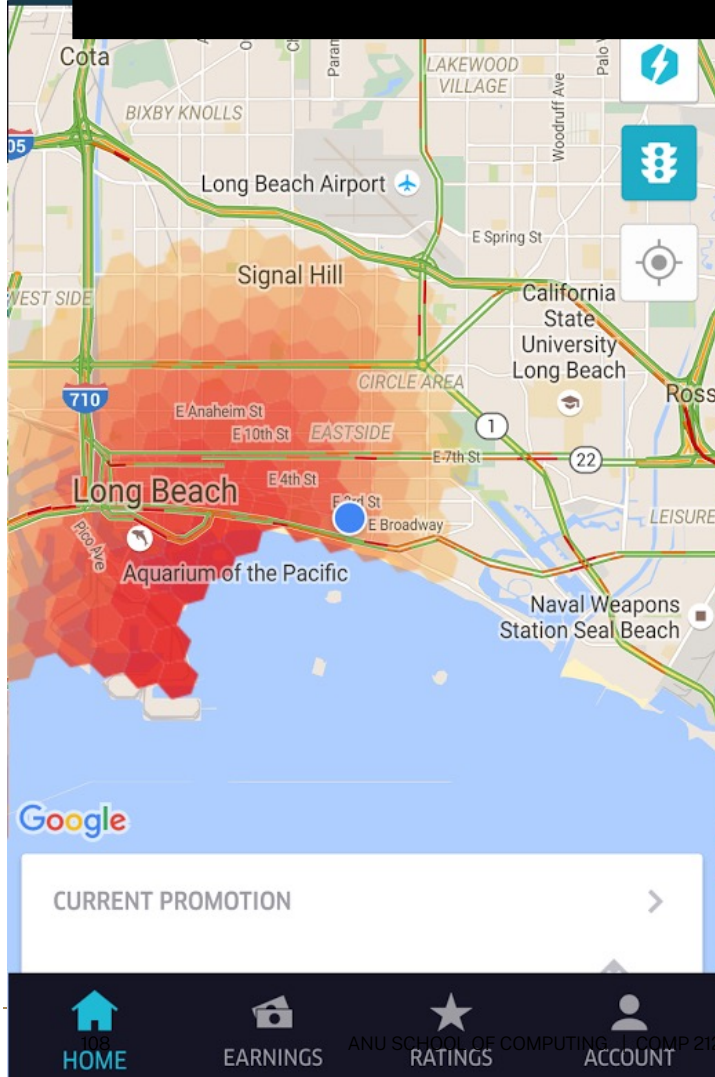
# Feature Extraction



- In OCR/translation:
  - Bounding boxes for text of interest
  - Character boundaries
  - Line segments for each character
  - GPS location of phone (to determine likely source language)



# Features?



# Feature Extraction



- In surge prediction:
  - Location and time of past surges
  - Events
  - Number of people traveling to an area
  - Typical demand curves in an area
  - Demand in other areas
  - Weather



# Data Cleaning

- Removing outliers
- Normalizing data
- Missing values
- ...



# Learning



- Build a predictor that best describes an outcome for the observed features



# Evaluation



- Prediction accuracy on learned data vs
- Prediction accuracy on unseen data
  - Separate learning set, not used for training
  
- For binary predictors: false positives vs. false negatives, precision vs. recall
- For numeric predictors: average (relative) distance between real and predicted value
- For ranking predictors: top-K, etc.



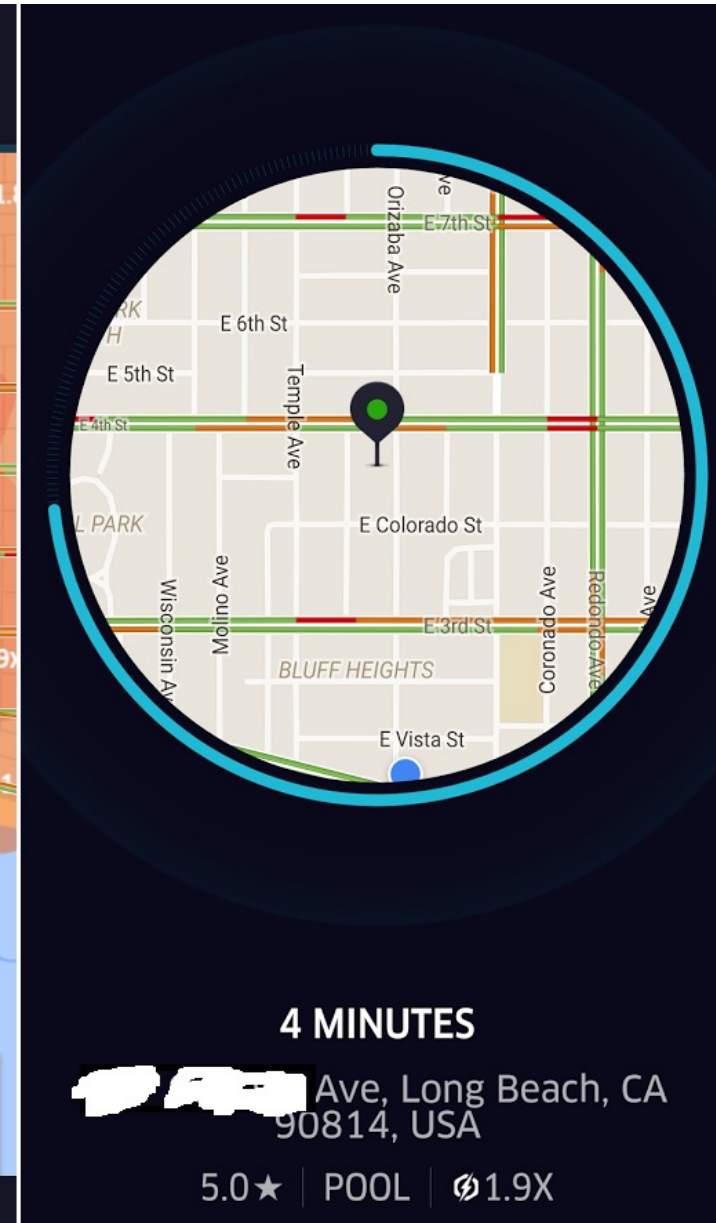
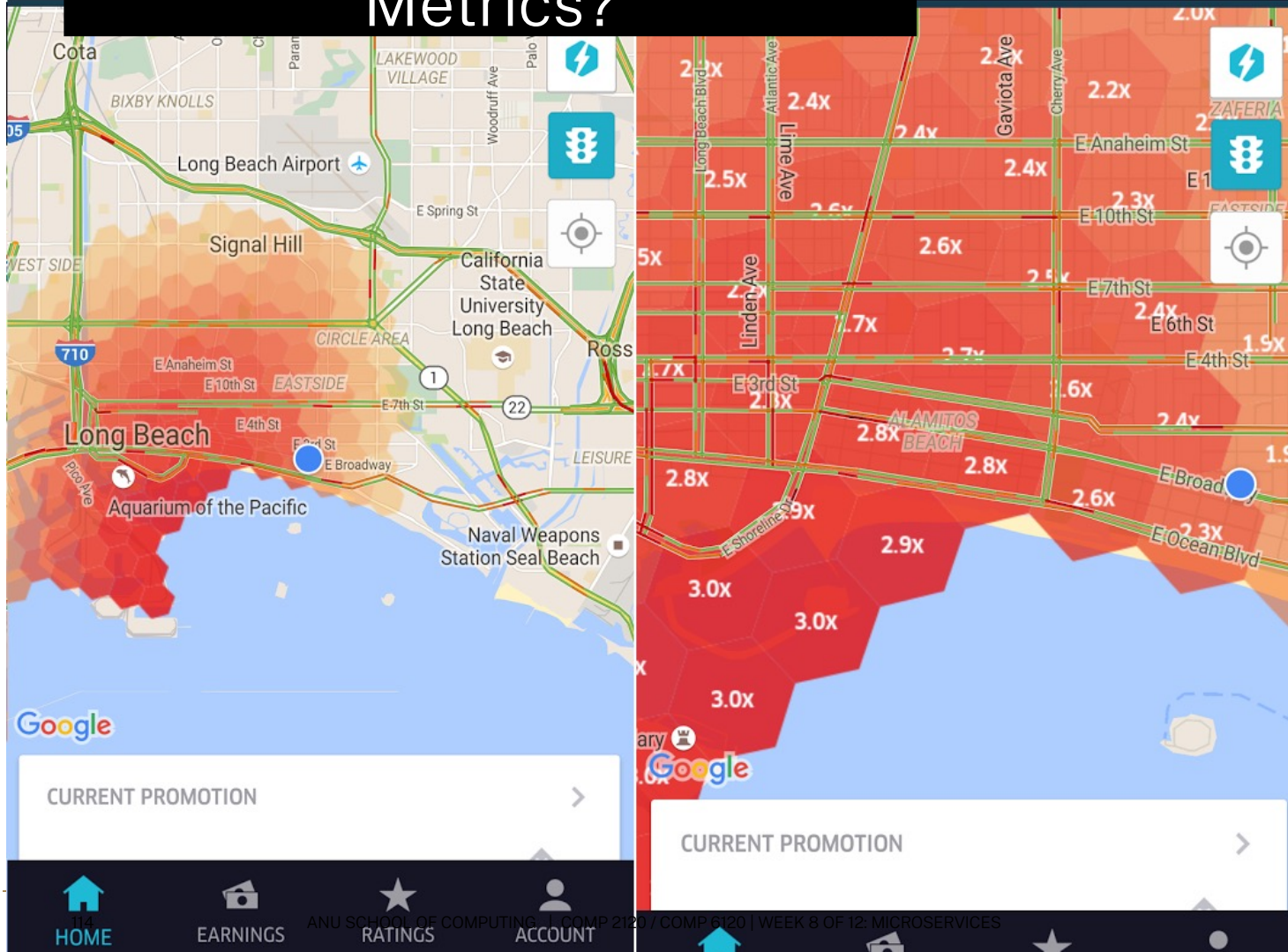


# Evaluation Data and Metrics?

現金のみ

Japanese ↔ English  
**cash only**

# Evaluation Data and Metrics?



# Learning and Evaluating in Production

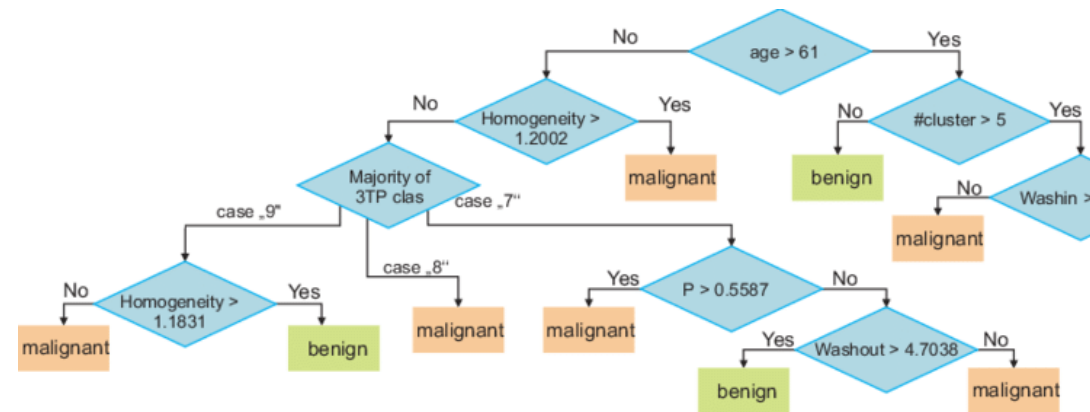
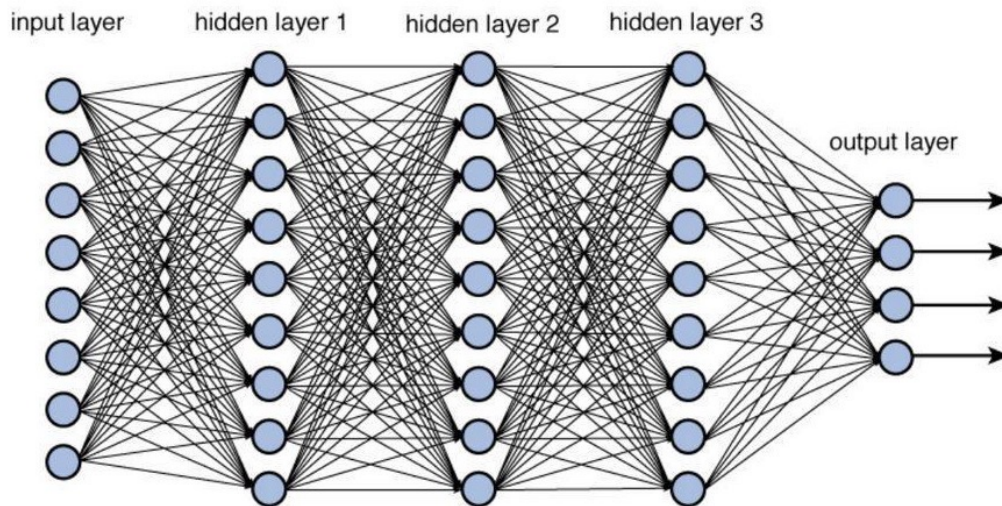
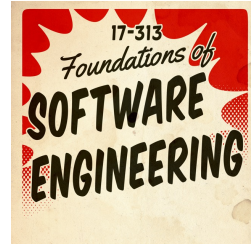


- Beyond static data sets, **build telemetry**
- Design challenge: identify mistakes in practice
  
- Use sample of live data for evaluation
- Retrain models with sampled live data regularly
- Monitor performance and intervene



# Understanding Capabilities and Tradeoffs

- Deep Neural Networks
- Decision Trees



# ML Model Tradeoffs



- Accuracy
- Capabilities (e.g. classification, recommendation, clustering...)
- Amount of training data needed
- Inference latency
- Learning latency; incremental learning?
- Model size
- Explainable? Robust?
- ...



# Where should the model live?



Glasses

Phone

Cloud

OCR  
Component

Translation  
Component



# Where should the model live?



Vehicle

Phone

Cloud

Surge  
Prediction



# 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?





# Typical Designs



- **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



# Typical Designs



- **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**



# Other 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



# Reactive Systems



- **Responsive**
  - consistent, high performance
- **Resilient**
  - maintain responsive in the face of failure, recovery, rollback
- **Elastic**
  - scale with varying loads



# Common Design Strategies



- Message-driven, lazy computation, functional programming
  - asynchronous, message passing style
- Replication, containment, supervision
  - replicate and coordinate isolated components, e.g. with containers
- Data streams, “infinite data”, immutable facts
  - streaming technologies, data lakes
- See “big data systems” and “cloud computing”



# Making Decisions

- What steps to take?
- What information to collect?



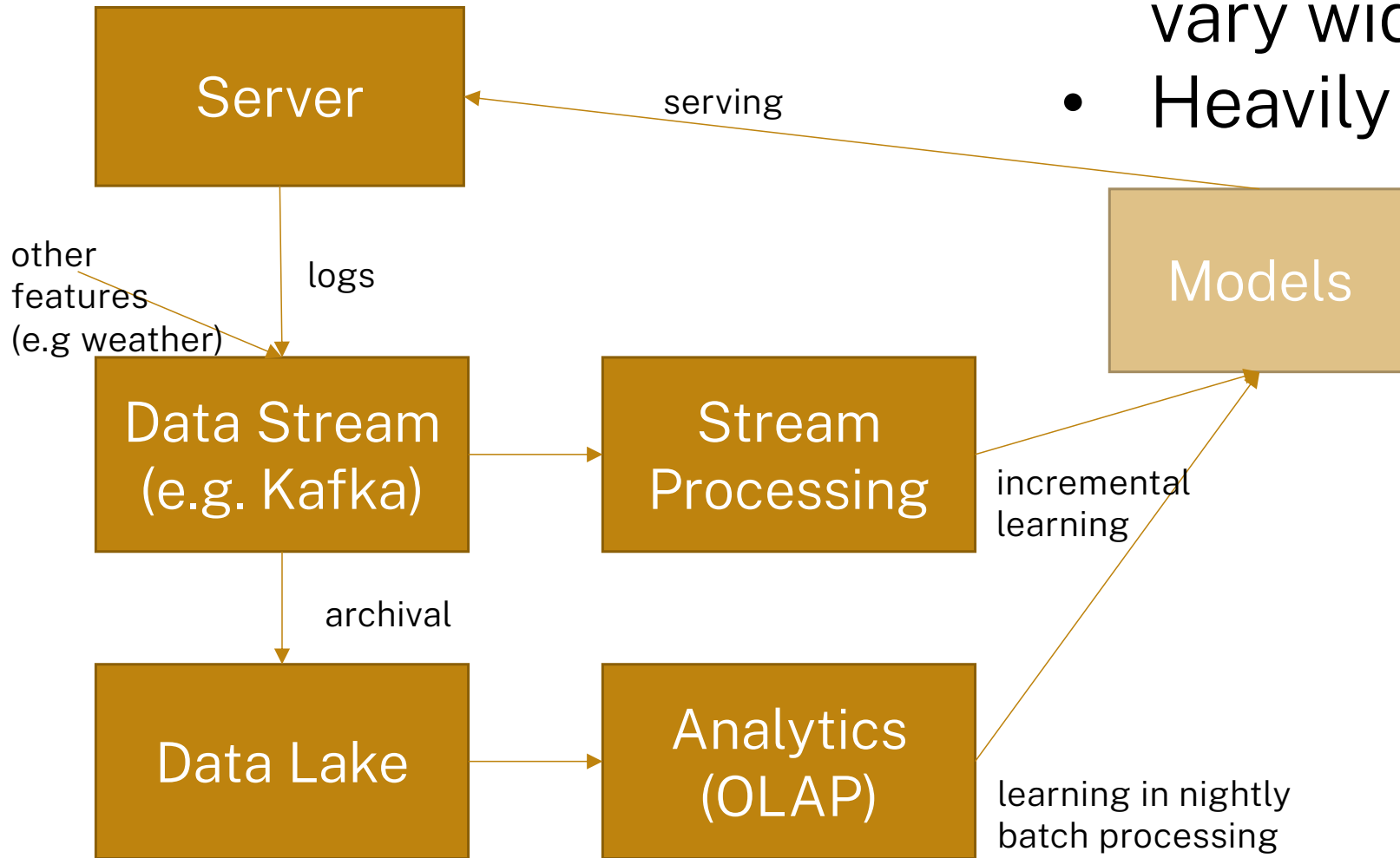
# Updating Models



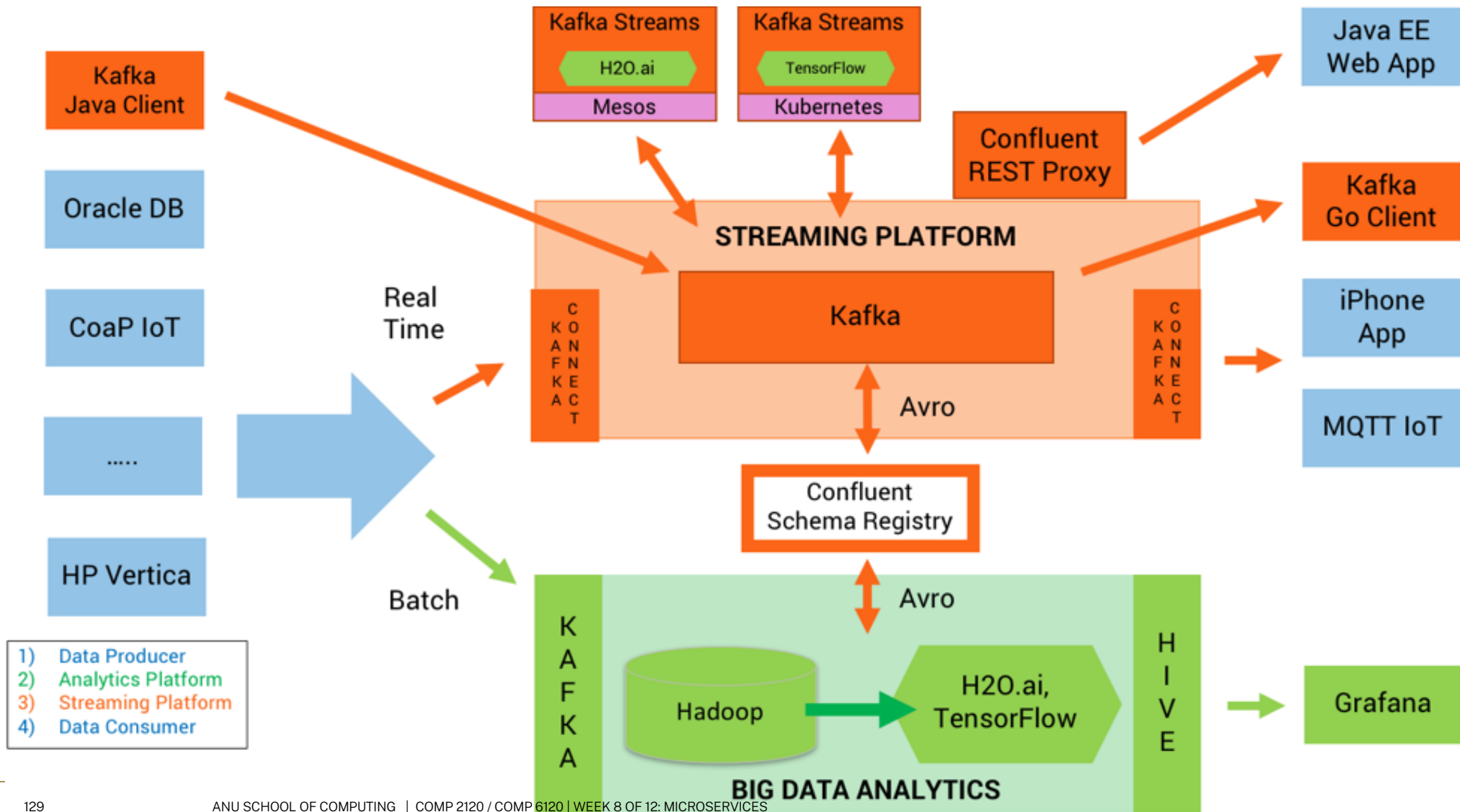
- Models are rarely static outside the lab
- Data drift, feedback loops, new features, new requirements
- When and how to update models?
- How to version? How to avoid mistakes?



- Latency and automatic vary widely
- Heavily distributed





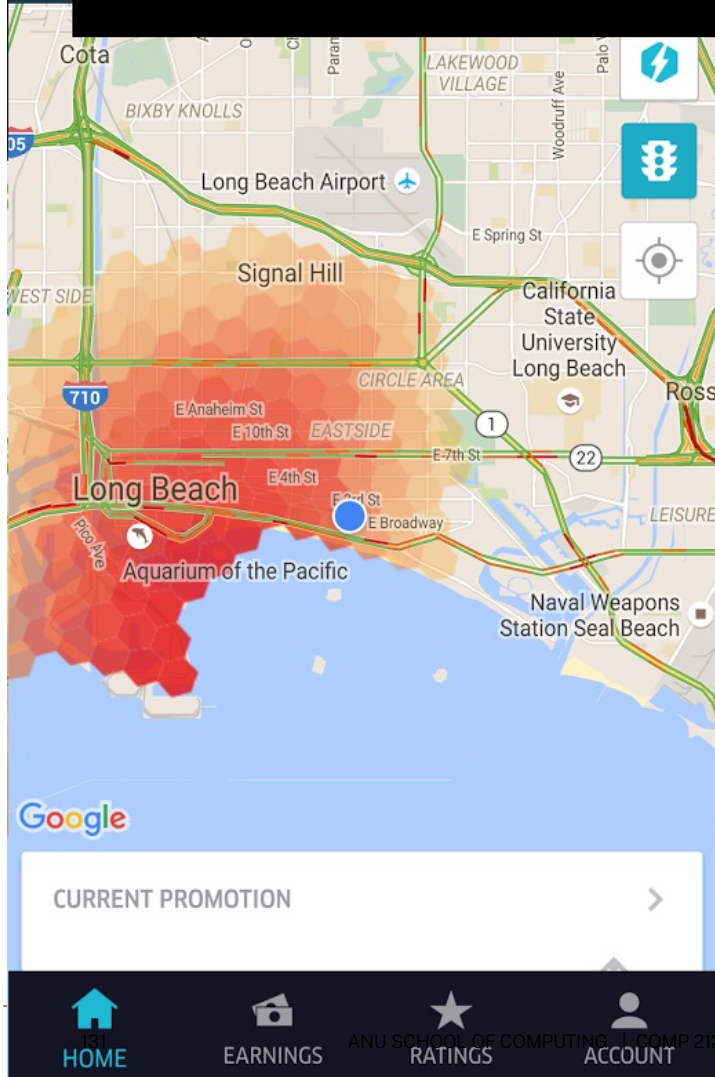


# Update Strategy?

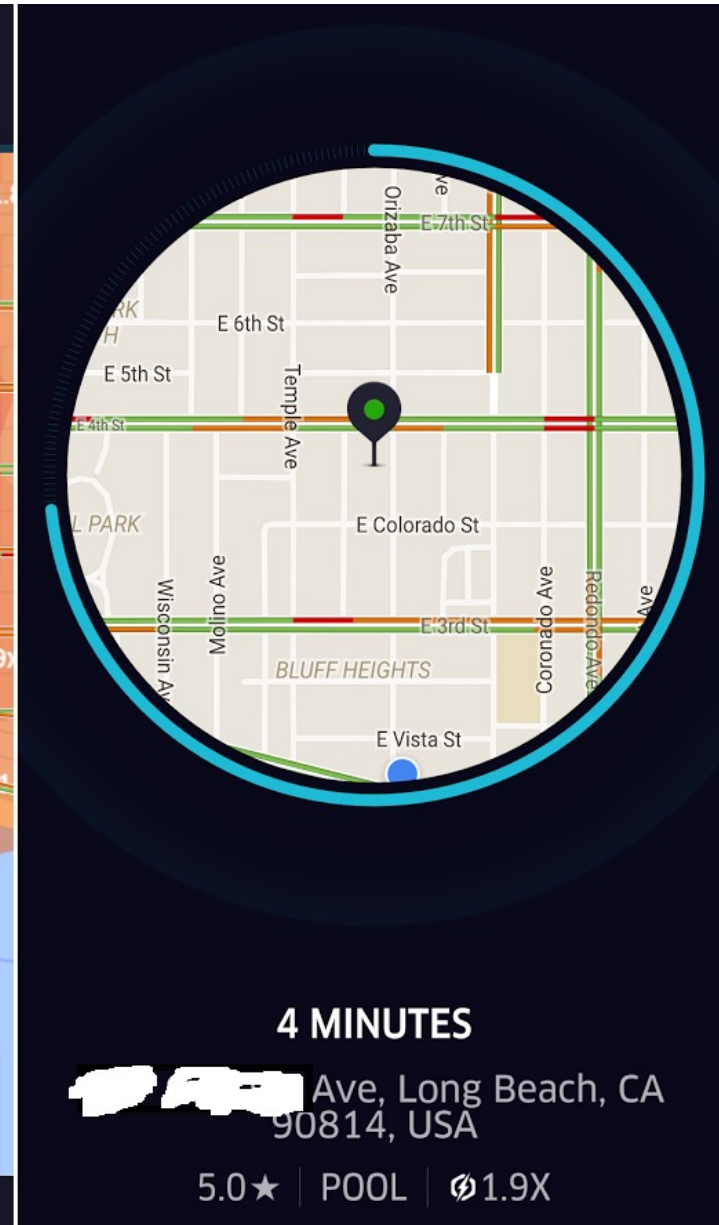
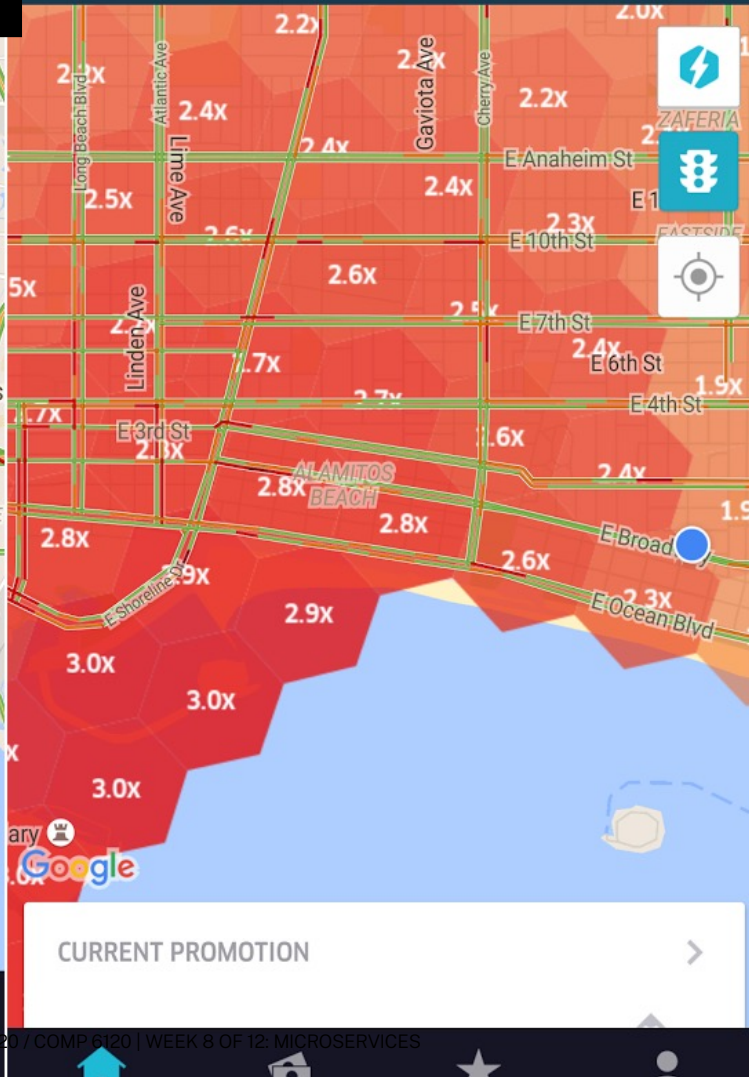
現金のみ

← Japanese ↔ English  
**cash only**

# Update Strategy?

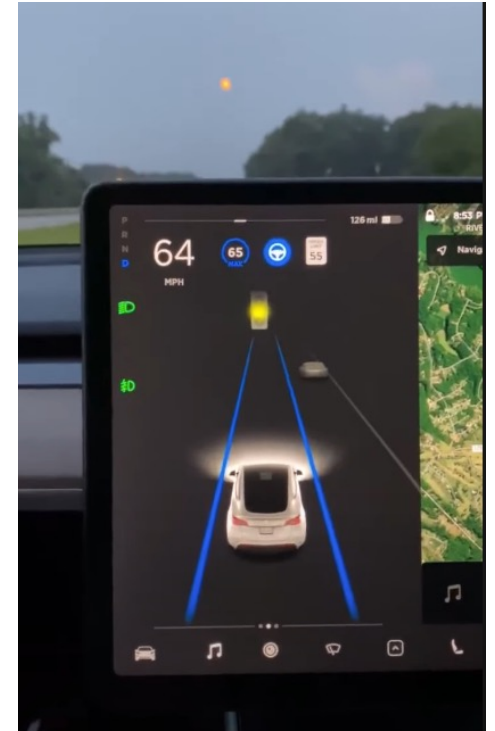
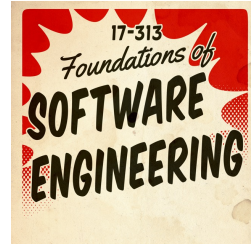


GO OFFLINE



# Mistakes will happen

- No specification
- ML components detect patterns from data (real and spurious)
- Predictions are often accurate, but mistakes always possible
- Mistakes are not predicable or explainable or similar to human mistakes
- Plan for mistakes
- Telemetry to learn about mistakes?



# How Models can Break



- System outage
- Model outage
  - model tested? deployment and updates reliable? file corrupt?
- Model errors
- Model degradation
  - data drift, feedback loops



# Hazard Analysis

- Worst thing that can happen?
- Backup strategy? Undoable? Nontechnical compensation?



# Mitigating Mistakes



- Investigating in ML
  - e.g., more training data, better data, better features, better engineers
- Less forceful experience
  - e.g., prompt rather than automate decisions, turn off
- Adjust learning parameters
  - e.g., more frequent updates, manual adjustments
- Guardrails
  - e.g., heuristics and constraints on outputs
- Override errors
  - e.g., hardcode specific results



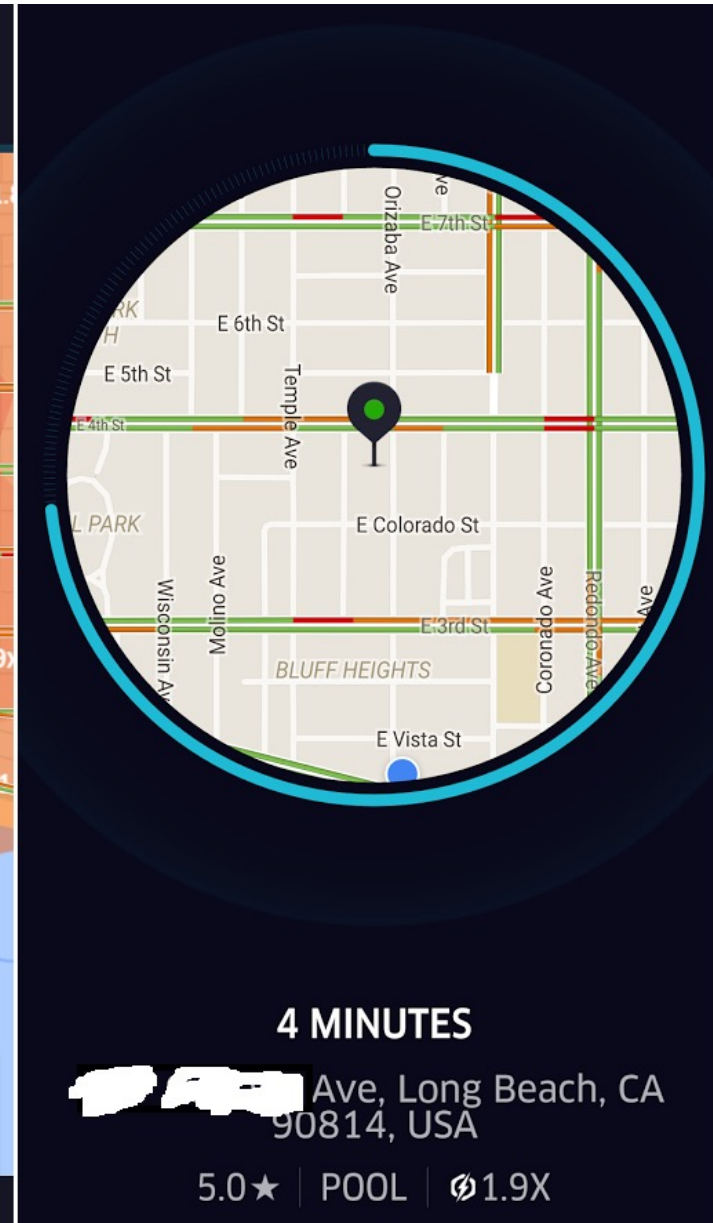
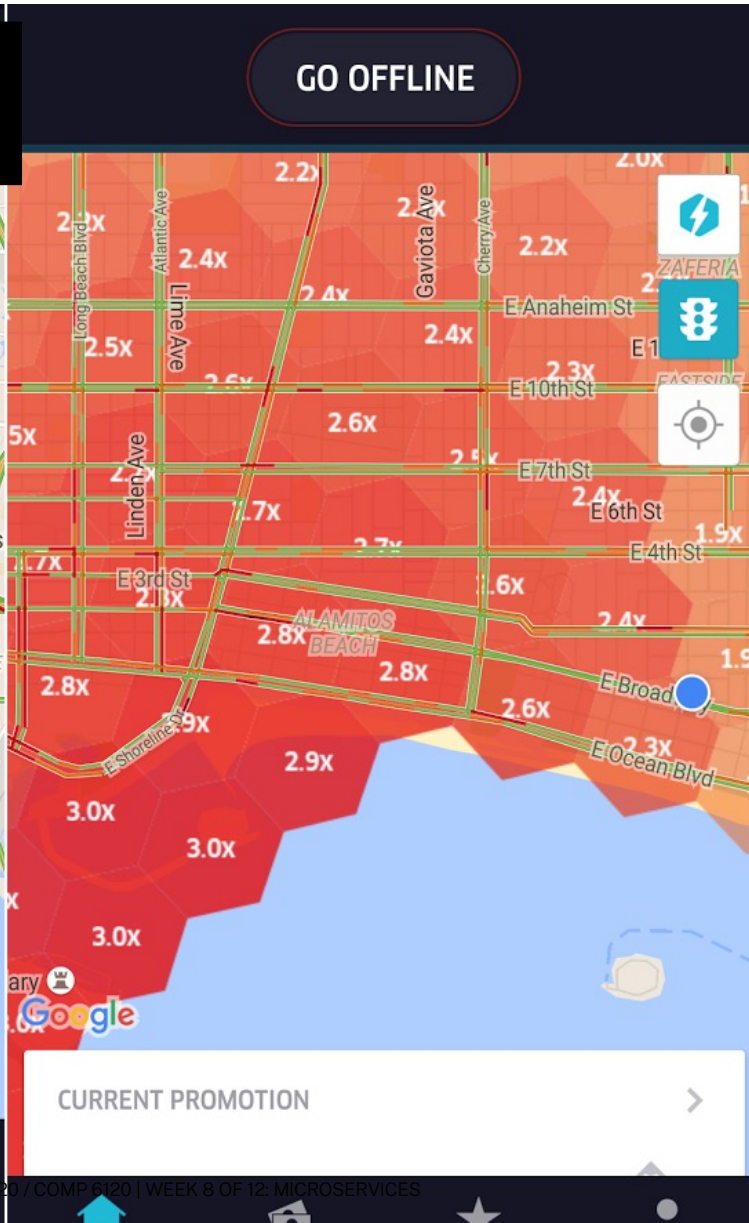
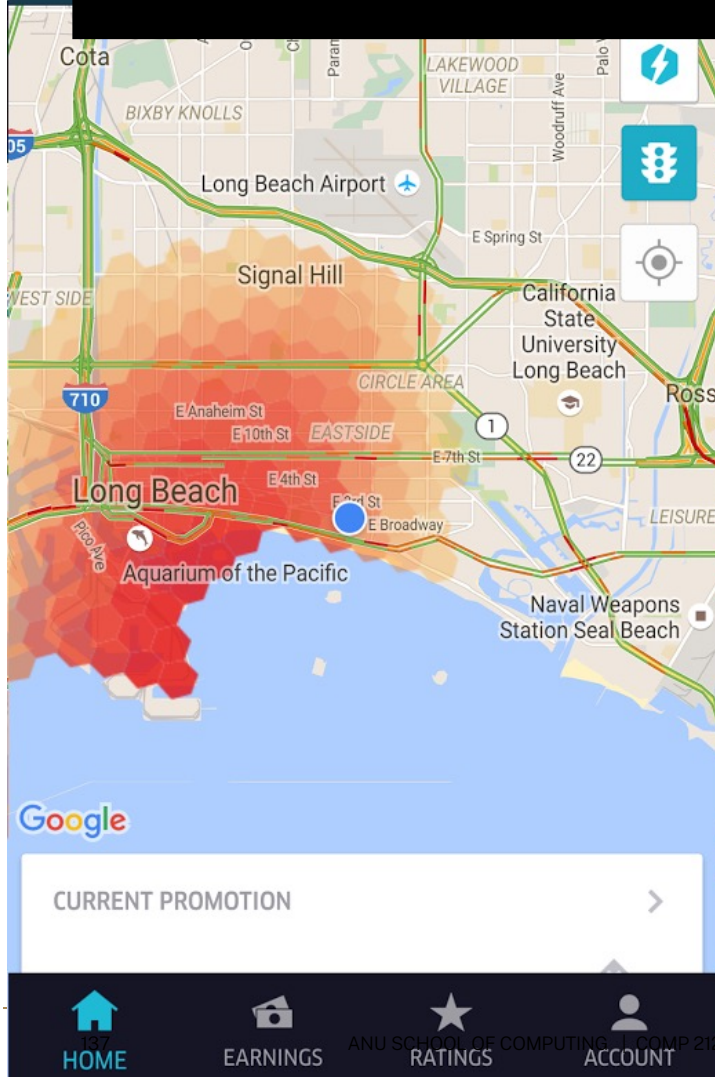
Mistakes?

現金のみ

← Japanese ↔ English  
**cash only**



# Mistakes?



# Telemetry



- **Purpose:**

- monitor operation
- monitor success (accuracy)
- improve models over time (e.g., detect new features)


- **Challenges:**


- too much data – sample, summarization, adjustable
- hard to measure – intended outcome not observable? proxies?
- rare events – important but hard to capture
- cost – significant investment must show benefit
- privacy – abstracting data



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333**



Did you spend money on OpenAI's ChatGPT or DALL-E or similar?  0

Yes **(A)**

No **(B)**

I cannot say as I used it to cheat on assignments **(C)**

