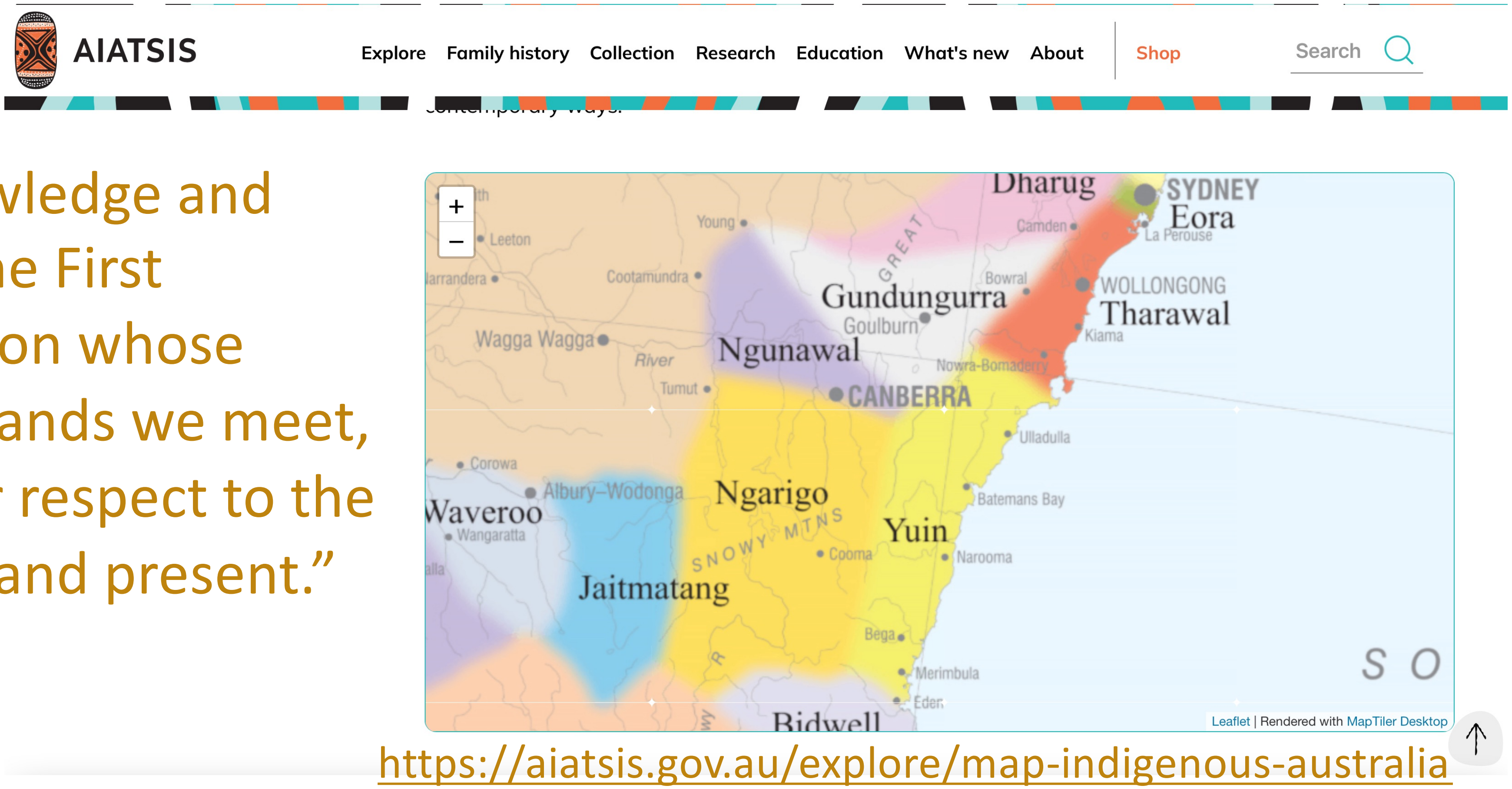
Week:
5 of 12

COMP 2120 / COMP 6120

MICROSERVICES

A/Prof Alex Potanin and Dr Melina Vidoni

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

Software architecture

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- To create a reliable, secure and efficient product, you need to pay attention to architectural design which includes:
  - its overall organization,
  - how the software is decomposed into components,
  - the server organization
  - the technologies that you use to build the software.The architecture of a software product affects its performance, usability, security, reliability and maintainability.
- There are many different interpretations of the term 'software architecture'.
  - Some focus on 'architecture' as a noun - the structure of a system and others consider 'architecture' to be a verb - the process of defining these structures.

The IEEE definition of software architecture

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Architecture is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

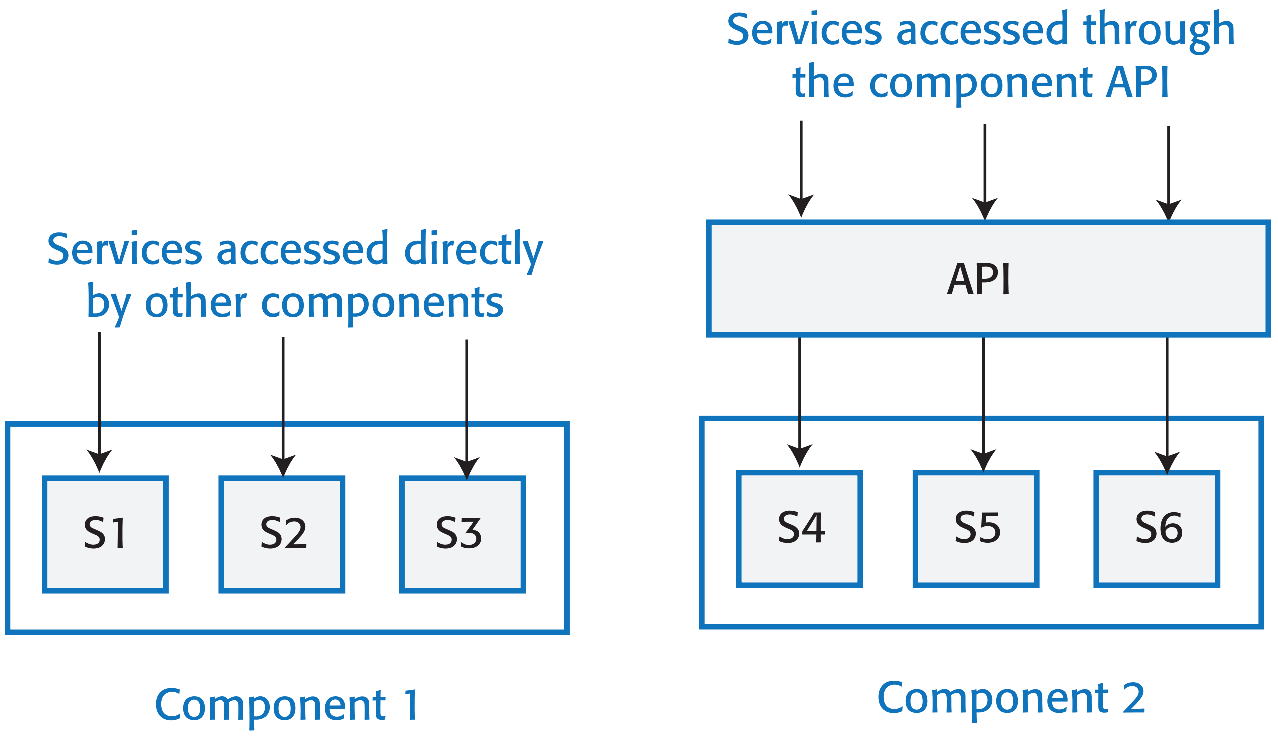
Software architecture and components

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- A component is an element that implements a coherent set of functionality or features.
- Software component can be considered as a collection of one or more services that may be used by other components.
- When designing software architecture, you don’t have to decide how an architectural element or component is to be implemented.
- Rather, you design the component interface and leave the implementation of that interface to a later stage of the development process.

Access to services provided by software components

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION


Services accessed through the component API
API
S4
S5
S6
Component 2
Services accessed directly by other components
S1
S2
S3
Component 1

Why is architecture important?

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- Architecture is important because the architecture of a system has a fundamental influence on the non-functional system properties, shown on the next slide.
- Architectural design involves understanding the issues that affect the architecture of your product and creating an architectural description that shows the critical components and their relationships.
- Minimizing complexity should be an important goal for architectural designers.
  - The more complex a system, the more difficult and expensive it is to understand and change.
  - Programmers are more likely to make mistakes and introduce bugs and security vulnerabilities when they are modifying or extending a complex system…

Non-functional system quality attributes

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *Responsiveness*
  Does the system return results to users in a reasonable time?
- *Reliability*
  Do the system features behave as expected by both developers and users?
- *Availability*
  Can the system deliver its services when requested by users?
- *Security*
  Does the system protect itself and users' data from unauthorized attacks and intrusions?
- *Usability*
  Can system users access the features that they need and use them quickly and without errors?
- *Maintainability*
  Can the system be readily updated and new features added without undue costs?
- *Resilience*
  Can the system continue to deliver user services in the event of partial failure or external attack?

The influence on architecture of system security

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- *A centralized security architecture*
  In the Star Wars prequel Rogue One (https://en.wikipedia.org/wiki/Rogue_One), the evil Empire have stored the plans for all of their equipment in a single, highly secure, well-guarded, remote location. This is called a centralized security architecture. It is based on the principle that if you maintain all of your information in one place, then you can apply lots of resources to protect that information and ensure that intruders can't get hold of it.
- Unfortunately (for the Empire), the rebels managed to breach their security. They stole the plans for the Death Star, an event which underpins the whole Star Wars saga. In trying to stop them, the Empire destroyed their entire archive of system documentation with who knows what resultant costs. Had the Empire chosen a distributed security architecture, with different parts of the Death Star plans stored in different locations, then stealing the plans would have been more difficult. The rebels would have had to breach security in all locations to steal the complete Death Star blueprints.

Centralized security architectures

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

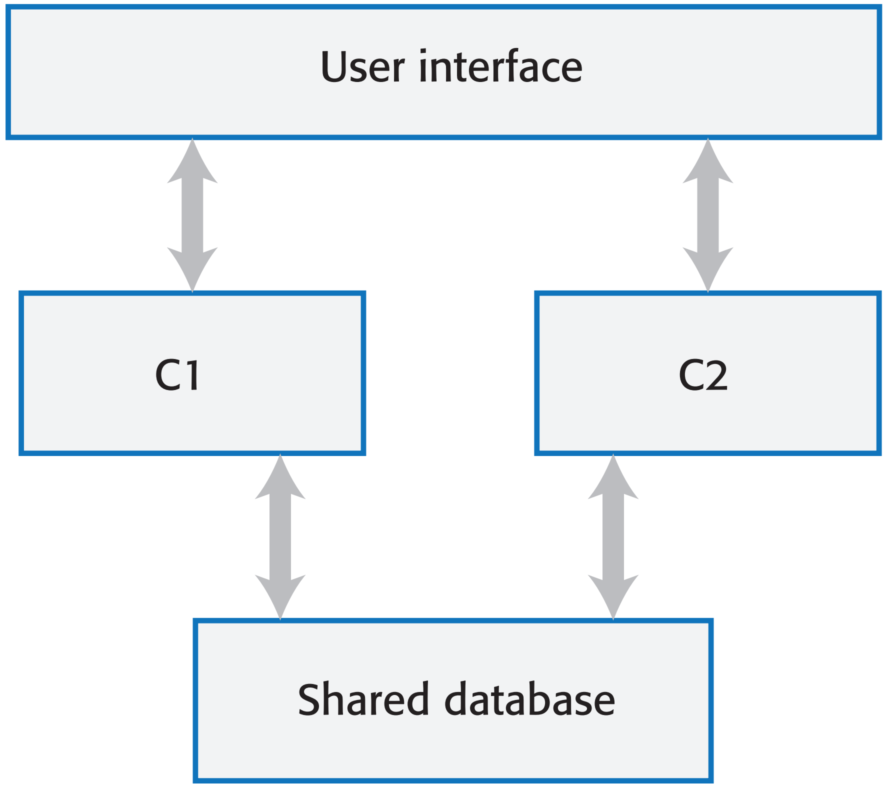
- The benefits of a centralized security architecture are that it is easier to design and build protection and that the protected information can be accessed more efficiently.
- However, if your security is breached, you lose everything.
- If you distribute information, it takes longer to access all of the information and costs more to protect it.
- If security is breached in one location, you only lose the information that you have stored there.

Shared database architecture

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

This slide shows a system with two components (C1 and C2) that share a common database.

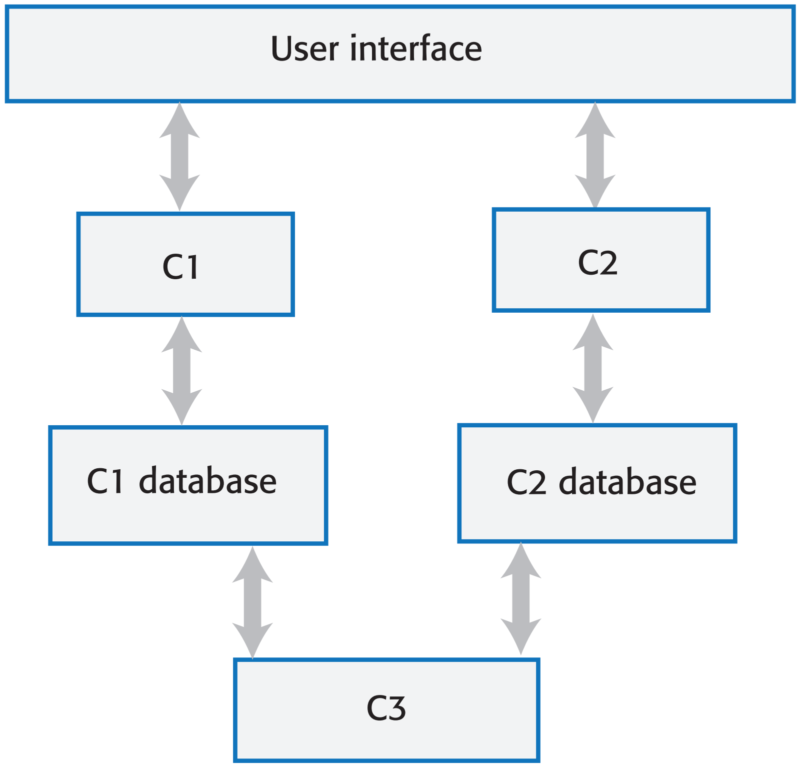
- Assume C1 runs slowly because it has to reorganize the information in the database before using it.
- The only way to make C1 faster might be to change the database. This means that C2 also has to be changed, which may, potentially, affect its response time.



Multiple database architecture

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

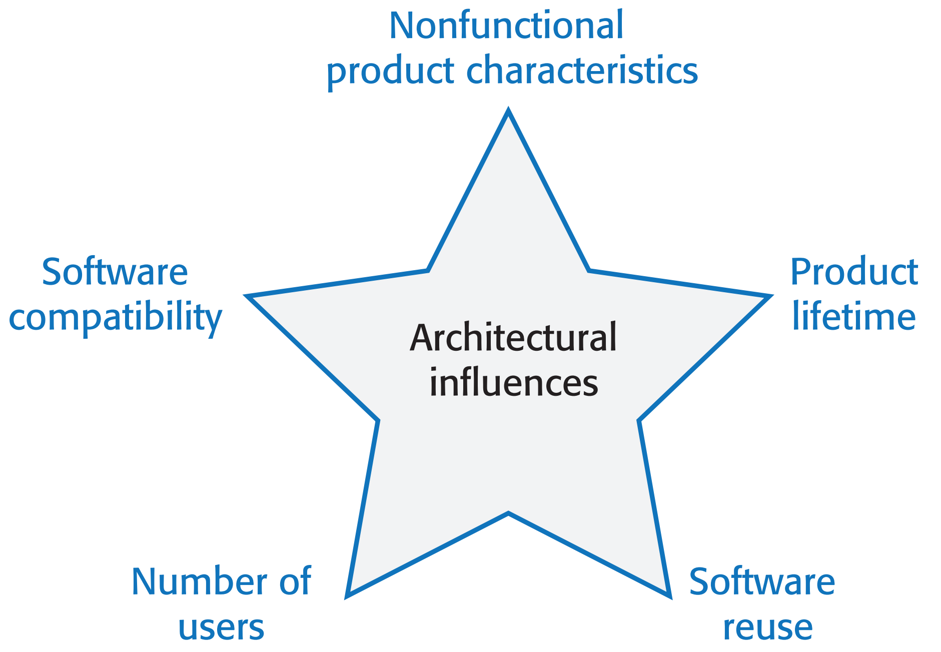
- This shows a different architecture used where each component has its own copy of the parts of the database that it needs.
  - If one component needs to change the database organization, this does not affect the other component.
- However, a multi-database architecture may run more slowly and may cost more to implement and change.
  - A multi-database architecture needs a mechanism (component C3) to ensure that the data shared by C1 and C2 is kept consistent when it is changed.



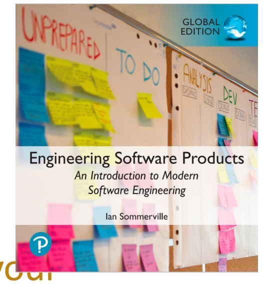
Database reconciliation

Issues that influence architectural decisions

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



The importance of architectural design issues


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *Nonfunctional product characteristics*
  Nonfunctional product characteristics such as security and performance affect all users. If you get these wrong, your product will is unlikely to be a commercial success. Unfortunately, some characteristics are opposing, so you can only optimize the most important.

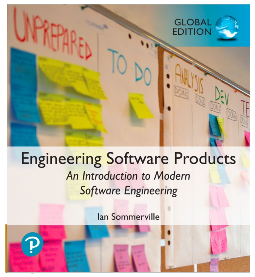
- *Product lifetime*
  If you anticipate a long product lifetime, you will need to create regular product revisions. You therefore need an architecture that is evolvable, so that it can be adapted to accommodate new features and technology.

- *Software reuse*
  You can save a lot of time and effort, if you can reuse large components from other products or open-source software. However, this constrains your architectural choices because you must fit your design around the software that is being reused.

- *Number of users*
  If you are developing consumer software delivered over the Internet, the number of users can change very quickly. This can lead to serious performance degradation unless you design your architecture so that your system can be quickly scaled up and down.

- *Software compatibility*
  For some products, it is important to maintain compatibility with other software so that users can adopt your product and use data prepared using a different system. This may limit architectural choices, such as the database software that you can use.

Trade off: Maintainability vs performance


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

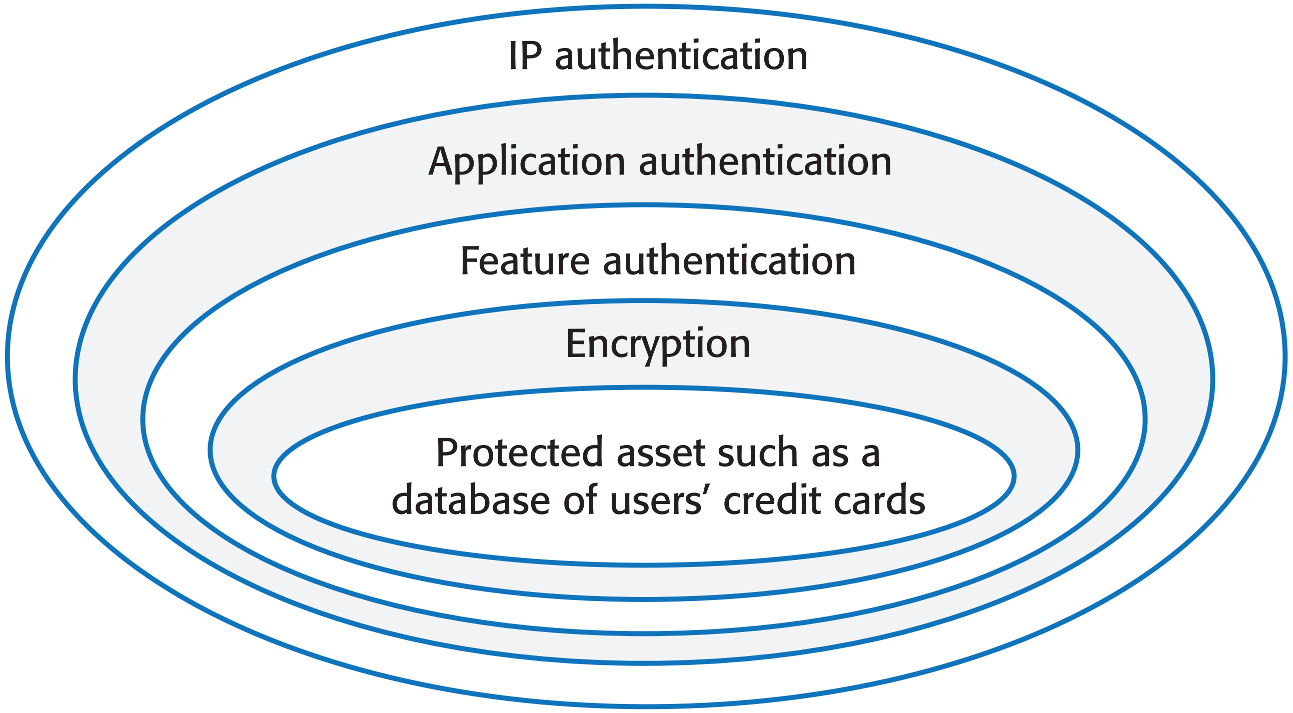
- System maintainability is an attribute that reflects how difficult and expensive it is to make changes to a system after it has been released to customers.
  - You improve maintainability by building a system from small self-contained parts, each of which can be replaced or enhanced if changes are required.
- In architectural terms, this means that the system should be decomposed into fine-grain components, each of which does one thing and one thing only.
  - However, it takes time for components to communicate with each other. Consequently, if many components are involved in implementing a product feature, the software will be slower.

Trade off: Security vs usability

- You can achieve security by designing the system protection as a series of layers (next slide).
  - An attacker has to penetrate all of those layers before the system is compromised.
- Layers might include system authentication layers, a separate critical feature authentication layer, an encryption layer and so on.
- Architecturally, you can implement each of these layers as separate components so that if one of these components is compromised by an attacker, then the other layers remain intact.

Authentication layers

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION


IP authentication
Application authentication
Feature authentication
Encryption
Protected asset such as a database of users' credit cards

Usability issues

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A layered approach to security affects the usability of the software.
  - Users have to remember information, like passwords, that is needed to penetrate a security layer. Their interaction with the system is inevitably slowed down by its security features.
  - Many users find this irritating and often look for work-arounds so that they do not have to re-authenticate to access system features or data.
- To avoid this, you need an architecture:
  - that doesn’t have too many security layers,
  - that doesn’t enforce unnecessary security,
  - that provides helper components that reduce the load on users.

Trade off: Availability vs time-to-market

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Availability is particularly important in enterprise products, such as produc the finance industry, where 24/7 operation is expected.
- The availability of a system is a measure of the amount of 'uptime' of that system.
  - Availability is normally expressed as a percentage of the time that a system is available to deliver user services.
- Architecturally, you achieve availability by having redundant components in a system.
  - To make use of redundancy, you include sensor components that detect failure, and switching components that switch operation to a redundant component when a failure is detected.
- Implementing extra components takes time and increases the cost of system development. It adds complexity to the system and therefore increases the chances of introducing bugs and vulnerabilities.

Architectural design questions

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

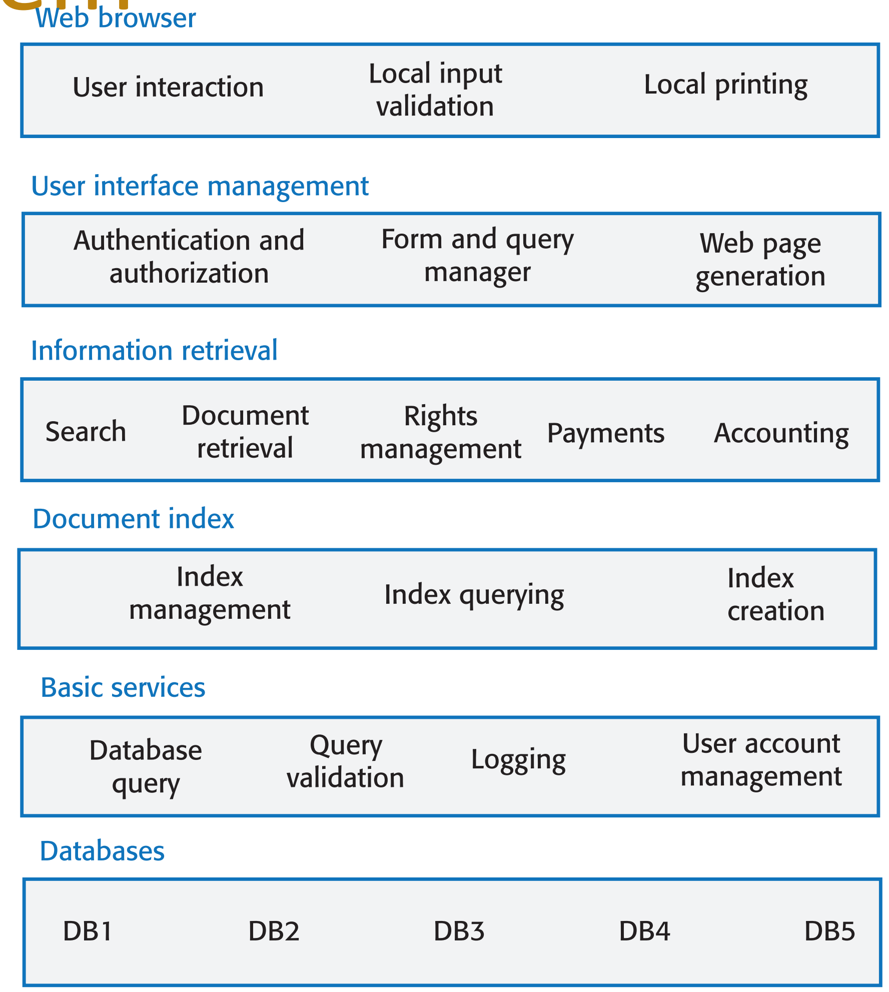
- How should the system be organized as a set of architectural components, where each of these components provides a subset of the overall system functionality?
  - The organization should deliver the system security, reliability and performance that you need.
- How should these architectural components be distributed and communicate with each other?
- What technologies should you use in building the system and what components should be reused?

Component organization

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Abstraction in software design means that you focus on the essential elements of a system or software component without concern for its details.
- At the architectural level, your concern should be on large-scale architectural components.
- Decomposition involves analysing these large-scale components and representing them as a set of finer-grain components.
- Layered models are often used to illustrate how a system is composed of components.

An architectural model of a document retrieval system



Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

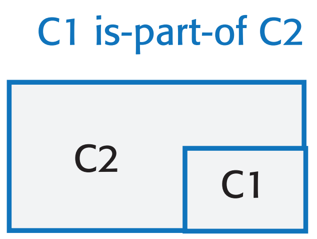
Architectural complexity

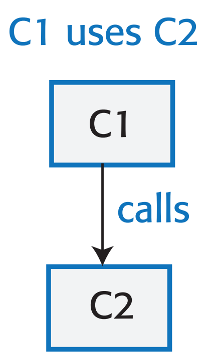
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

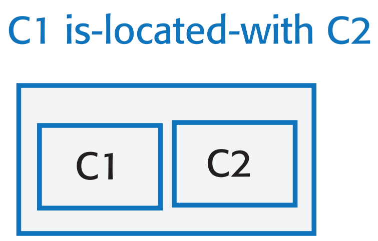
- Complexity in a system architecture arises because of the number a the nature of the relationships between components in that system.
- When decomposing a system into components, you should try to avoid unnecessary software complexity.
  - *Localize relationships*
    If there are relationships between components A and B, these are easier to understand if A and B are defined in the same module.
  - *Reduce shared dependencies*
    Where components A and B depend on some other component or data, complexity increases because changes to the shared component mean you have to understand how these changes affect both A and B.
- It is always preferable to use local data wherever possible and to avoid sharing data if you can.

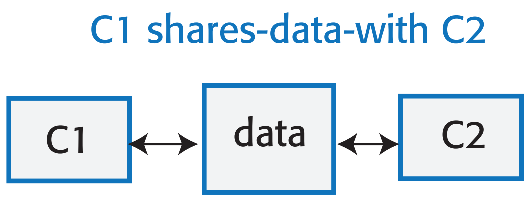
Examples of component relationships

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION





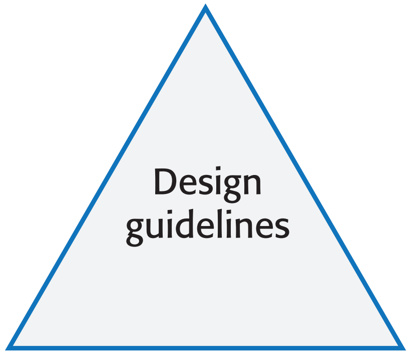




Architectural design guidelines

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Separation of concerns
Organize your architecture into components that focus on a single concern



Stable interfaces
Design component interfaces that are coherent and that change slowly

Implement once
Avoid duplicating functionality at different places in your architecture

Design guidelines and layered architectures

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- Each layer is an area of concern and is considered separately from o layers.
  - The top layer is concerned with user interaction, the next layer down with user interface management, the third layer with information retrieval and so on.
- Within each layer, the components are independent and do not overlap in functionality.
  - The lower layers include components that provide general functionality so there is no need to replicate this in the components in a higher level.
- The architectural model is a high-level model that does not include implementation information.
  - Ideally, components at level X (say) should only interact with the APIs of the components in level X-1. That is, interactions should be between layers and not across layers.

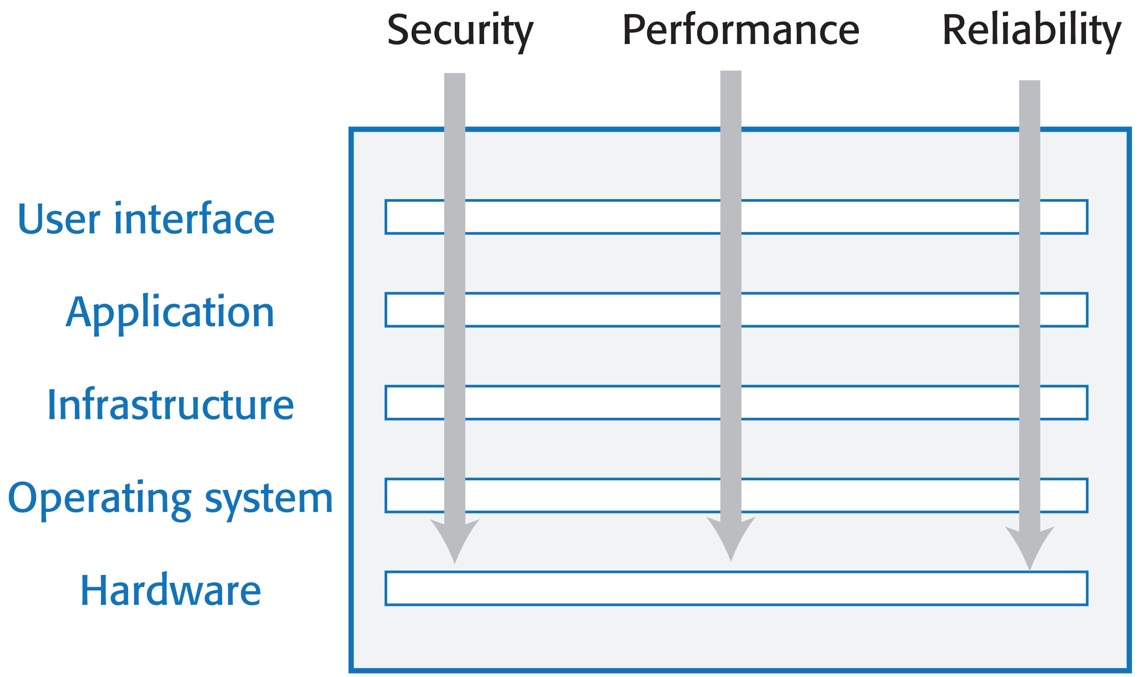
Cross-cutting concerns

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Cross-cutting concerns are concerns that are systemic, that is, they the whole system.
- In a layered architecture, cross-cutting concerns affect all layers in the system as well as the way in which people use the system.
- Cross-cutting concerns are completely different from the functional concerns represented by layers in a software architecture.
- Every layer has to take them into account and there are inevitably interactions between the layers because of these concerns.
- The existence of cross-cutting concerns is the reason why modifying a system after it has been designed to improve its security is often difficult.

Cross-cutting concerns

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



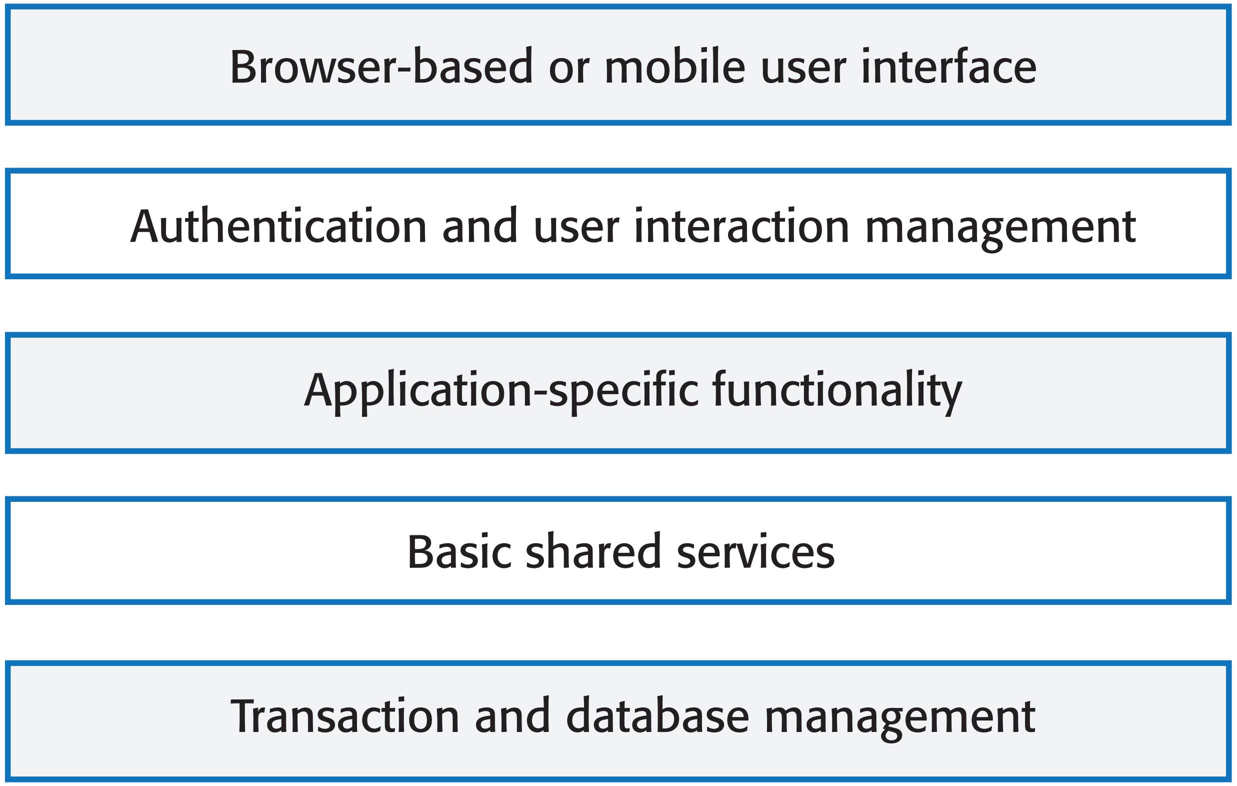
Security as a cross-cutting concern

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *Security architecture*
  Different technologies are used in different layers, such as an SQL database or a Firefox browser. Attackers can try to use of vulnerabilities in these technologies to gain access.
- Consequently, you need protection from attacks at each layer as well as protection, at lower layers in the system, from successful attacks that have occurred at higher-level layers.
- If there is only a single security component in a system, this represents a critical system vulnerability. If all security checking goes through that component and it stops working properly or is compromised in an attack, then you have no reliable security in your system.
- By distributing security across the layers, your system is more resilient to attacks and software failure (remember the Rogue One example earlier).

A generic layered architecture for a web-based application

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville


Browser-based or mobile user interface
Authentication and user interaction management
Application-specific functionality
Basic shared services
Transaction and database management

Layer functionality in a web-based application

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- *Browser-based or mobile user interface*
  A web browser system interface in which HTML forms are often used to collect user input. Javascript components for local actions, such as input validation, should also be included at this level. Alternatively, a mobile interface may be implemented as an app.

- *Authentication and UI management*
  A user interface management layer that may include components for user authentication and web page generation.

- *Application-specific functionality*
  An 'application' layer that provides functionality of the application. Sometimes, this may be expanded into more than one layer.

- *Basic shared services*
  A shared services layer, which includes components that provide services used by the application layer components.

- *Database and transaction management*
  A database layer that provides services such as transaction management and recovery. If your application does not use a database then this may not be required.

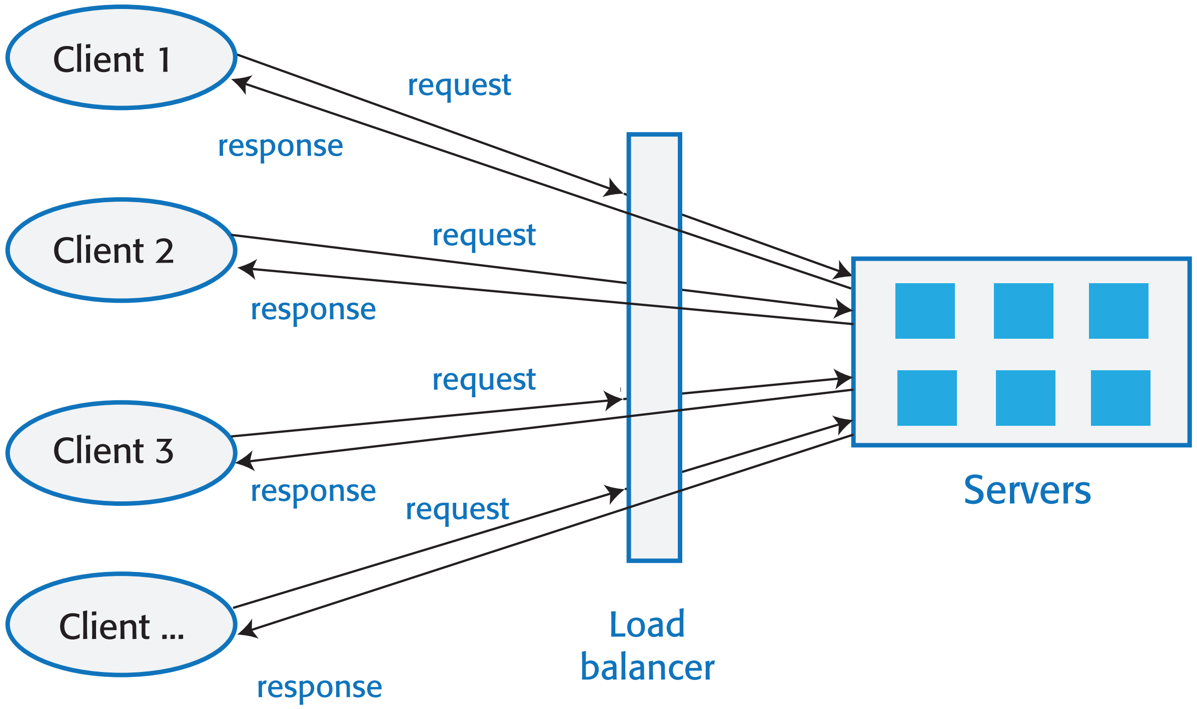
Distribution architecture

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- The distribution architecture of a software system defines the servers in the system and the allocation of components to these servers.
- Client-server architectures are a type of distribution architecture that is suited to applications where clients access a shared database and business logic operations on that data.
- In this architecture, the user interface is implemented on the user's own computer or mobile device.
  - Functionality is distributed between the client and one or more server computers.

Client-server architecture

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



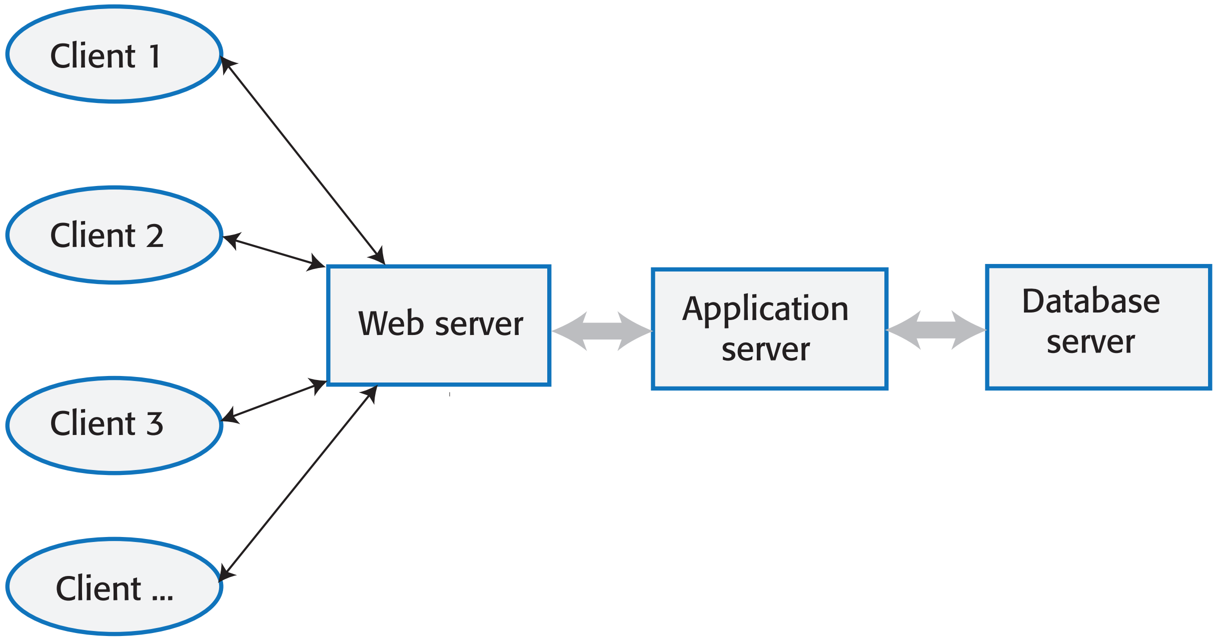
Client-server communication

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- Client-server communication normally uses the HTTP protocol.
  - The client sends a message to the server that includes an instruction such as GET or POST along with the identifier of a resource (usually a URL) on which that instruction should operate. The message may also include additional information, such as information collected from a form.
- HTTP is a text-only protocol so structured data has to be represented as text. There are two ways of representing this data that are widely used, namely XML and JSON.
  - XML is a markup language with tags used to identify each data item.
  - JSON is a simpler representation based on the representation of objects in the Javascript language.

Multi-tier client-server architecture

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



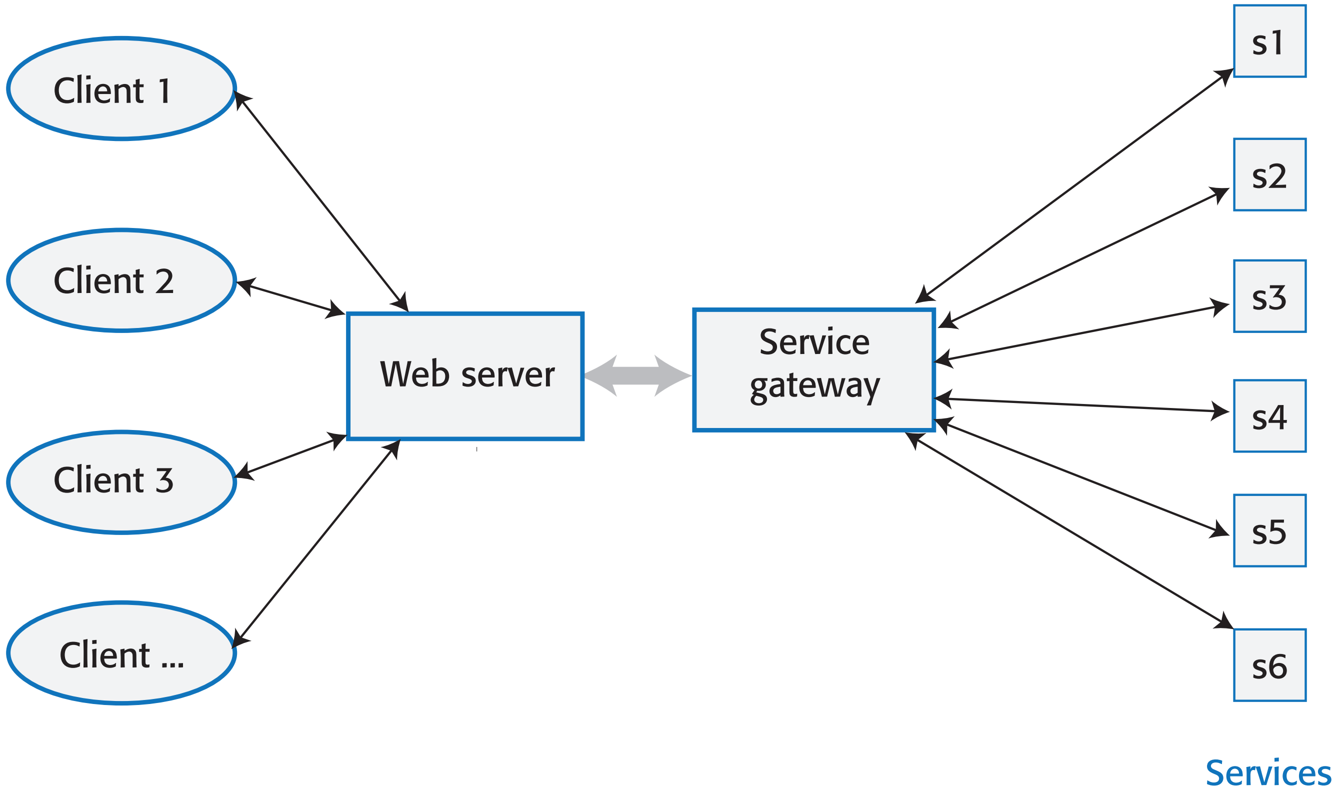
Service-oriented architecture

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- Services in a service-oriented architecture are stateless components, which means that they can be replicated and can migrate from one computer to another.
- Many servers may be involved in providing services
- A service-oriented architecture is usually easier to scale as demand increases and is resilient to failure.

Service-oriented architecture

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



Issues in architectural choice

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Data type and data updates
  - If you are mostly using structured data that may be updated by different system features, it is usually best to have a single shared database that provides locking and transaction management. If data is distributed across services, you need a way to keep it consistent and this adds overhead to your system.
- Change frequency
  - If you anticipate that system components will be regularly changed or replaced, then isolating these components as separate services simplifies those changes.
- The system execution platform
  - If you plan to run your system on the cloud with users accessing it over the Internet, it is usually best to implement it as a service-oriented architecture because scaling the system is simpler.
  - If your product is a business system that runs on local servers, a multi-tier architecture may be more appropriate.

Technology choices

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *Database*
  Should you use a relational SQL database or an unstructured NOSQL database?
- *Platform*
  Should you deliver your product on a mobile app and/or a web platform?
- *Server*
  Should you use dedicated in-house servers or design your system to run on a public cloud? If a public cloud, should you use Amazon, Google, Microsoft, or some other option?
- *Open source*
  Are there suitable open-source components that you could incorporate into your products?
- *Development tools*
  Do your development tools embed architectural assumptions about the software being developed that limit your architectural choices?

Database

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- There are two kinds of database that are now commonly used:
  - Relational databases, where the data is organised into structured tables
  - NoSQL databases, in which the data has a more flexible, user-defined organization.
- Relational databases, such as MySQL, are particularly suitable for situations where you need transaction management and the data structures are predictable and fairly simple.
- NoSQL databases, such as MongoDB, are more flexible and potentially more efficient than relational databases for data analysis.
  - NoSQL databases allow data to be organized hierarchically rather than as flat tables and this allows for more efficient concurrent processing of ‘big data’.

Delivery platform

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

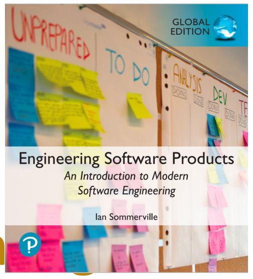
- Delivery can be as a web-based or a mobile product or both
- Mobile issues:
  - *Intermittent connectivity* You must be able to provide a limited service without network connectivity.
  - *Processor power* Mobile devices have less powerful processors, so you need to minimize computationally-intensive operations.
  - *Power management* Mobile battery life is limited so you should try to minimize the power used by your application.
  - *On-screen keyboard* On-screen keyboards are slow and error-prone. You should minimize input using the screen keyboard to reduce user frustration.
- To deal with these differences, you usually need separate browser-based and mobile versions of your product front-end.
  - You may need a completely different decomposition architecture in these different versions to ensure that performance and other characteristics are maintained.

Server

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A key decision that you have to make is whether to design your system to run on customer servers or to run on the cloud.
- For consumer products that are not simply mobile apps I think it almost always makes sense to develop for the cloud.
- For business products, it is a more difficult decision.
  - Some businesses are concerned about cloud security and prefer to run their systems on in-house servers. They may have a predictable pattern of system usage so there is less need to design your system to cope with large changes in demand.
- An important choice you have to make if you are running your software on the cloud is which cloud provider to use.

Open source



GLOBAL EDITION

Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

- Open source software is software that is available freely, which you can ch and modify as you wish.
  - The advantage is that you can reuse rather than implement new software, which reduces development costs and time to market.
  - The disadvantages of using open-source software is that you are constrained by that software and have no control over its evolution.
- The decision on the use of open-source software also depends on the availability, maturity and continuing support of open source components.
- Open source license issues may impose constraints on how you use the software.
- Your choice of open source software should depend on the type of product that you are developing, your target market and the expertise of your development team.

Development tools

- Development technologies, such as a mobile development toolkit or a web application framework, influence the architecture of your software.
  - These technologies have built-in assumptions about system architectures and you have to conform to these assumptions to use the development system.
- The development technology that you use may also have an indirect influence on the system architecture.
  - Developers usually favour architectural choices that use familiar technologies that they understand. For example, if your team have a lot of experience of relational databases, they may argue for this instead of a NoSQL database.

Software services

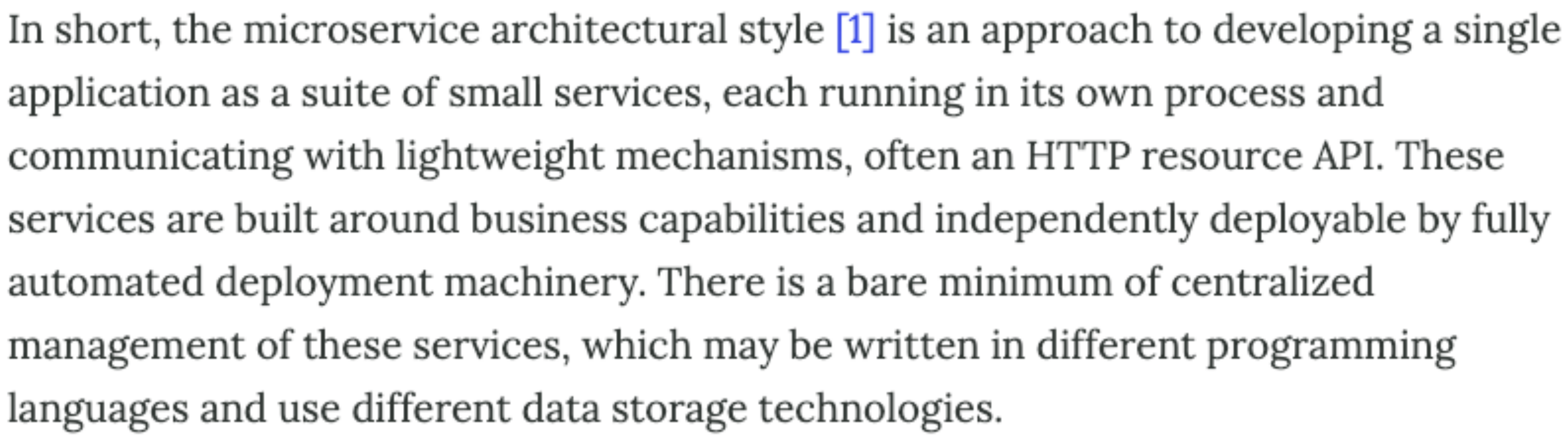
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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.

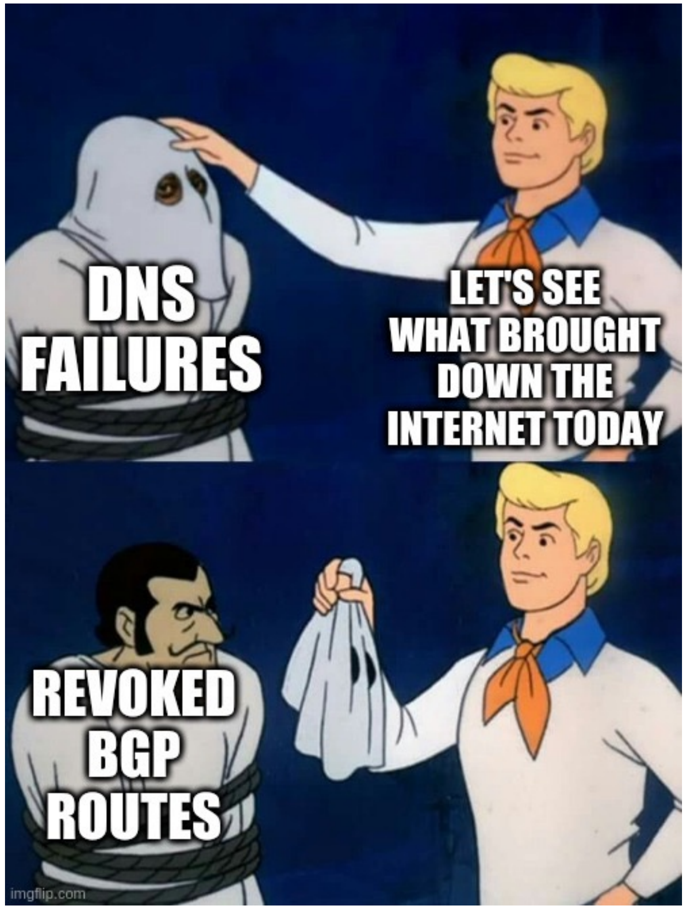


In short, the microservice architectural style [1] is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently 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

Mini Break in Monday Lecture

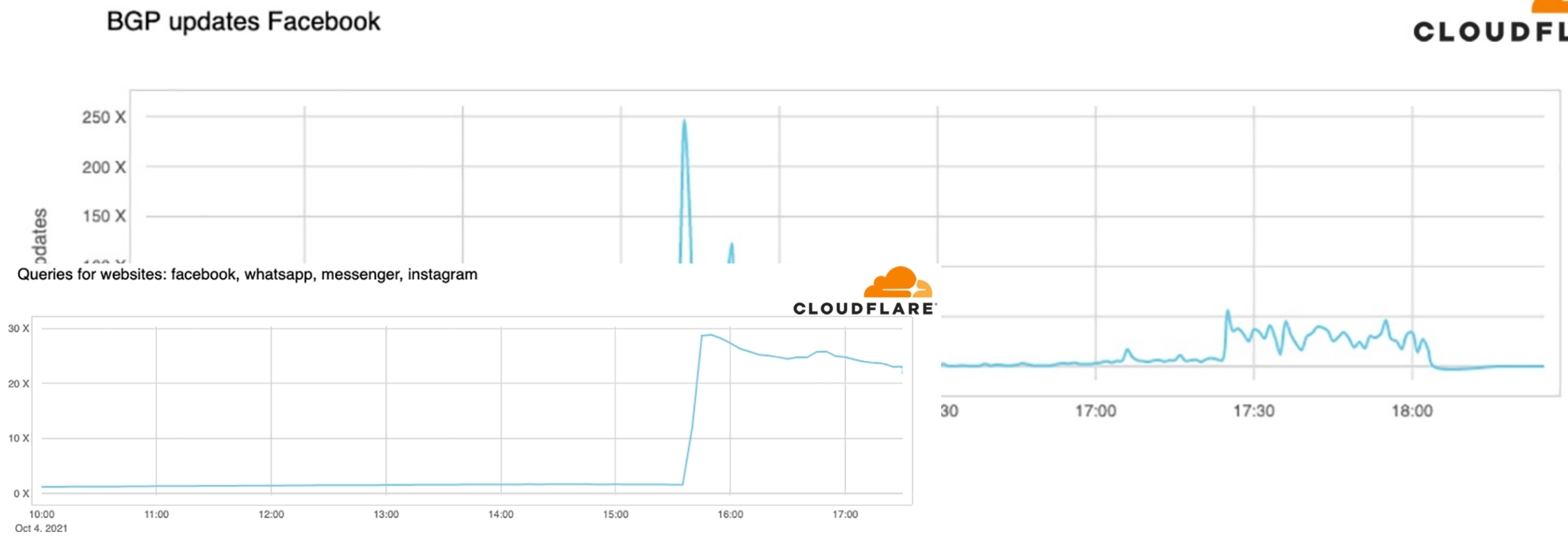
There has been a failure
on the INTERNETS



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

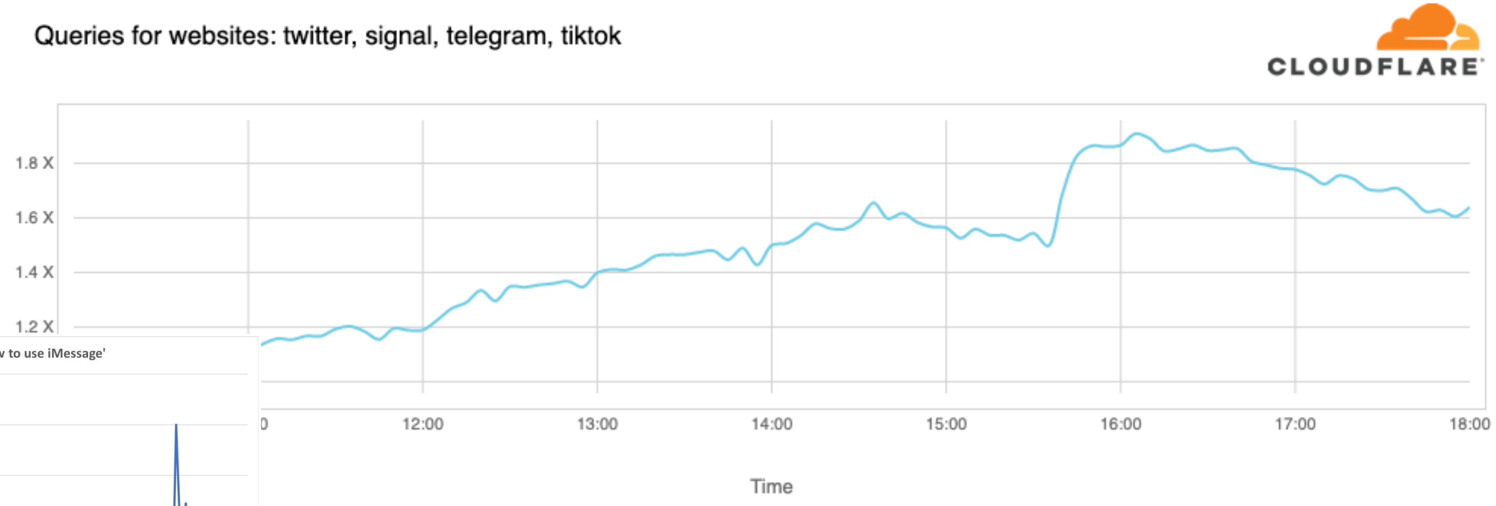
Facebook on Oct 4, 2021

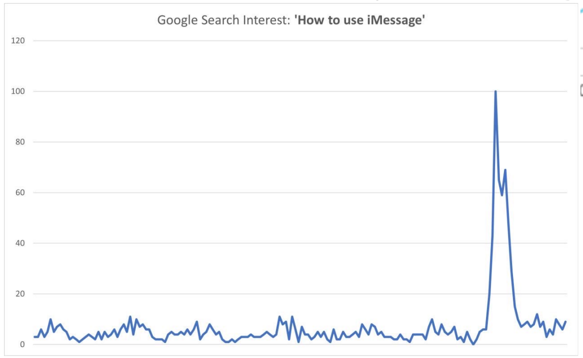
Source: https://blog.cloudflare.com/october-2021-facebook-outage/



Facebook on Oct 4, 2021

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

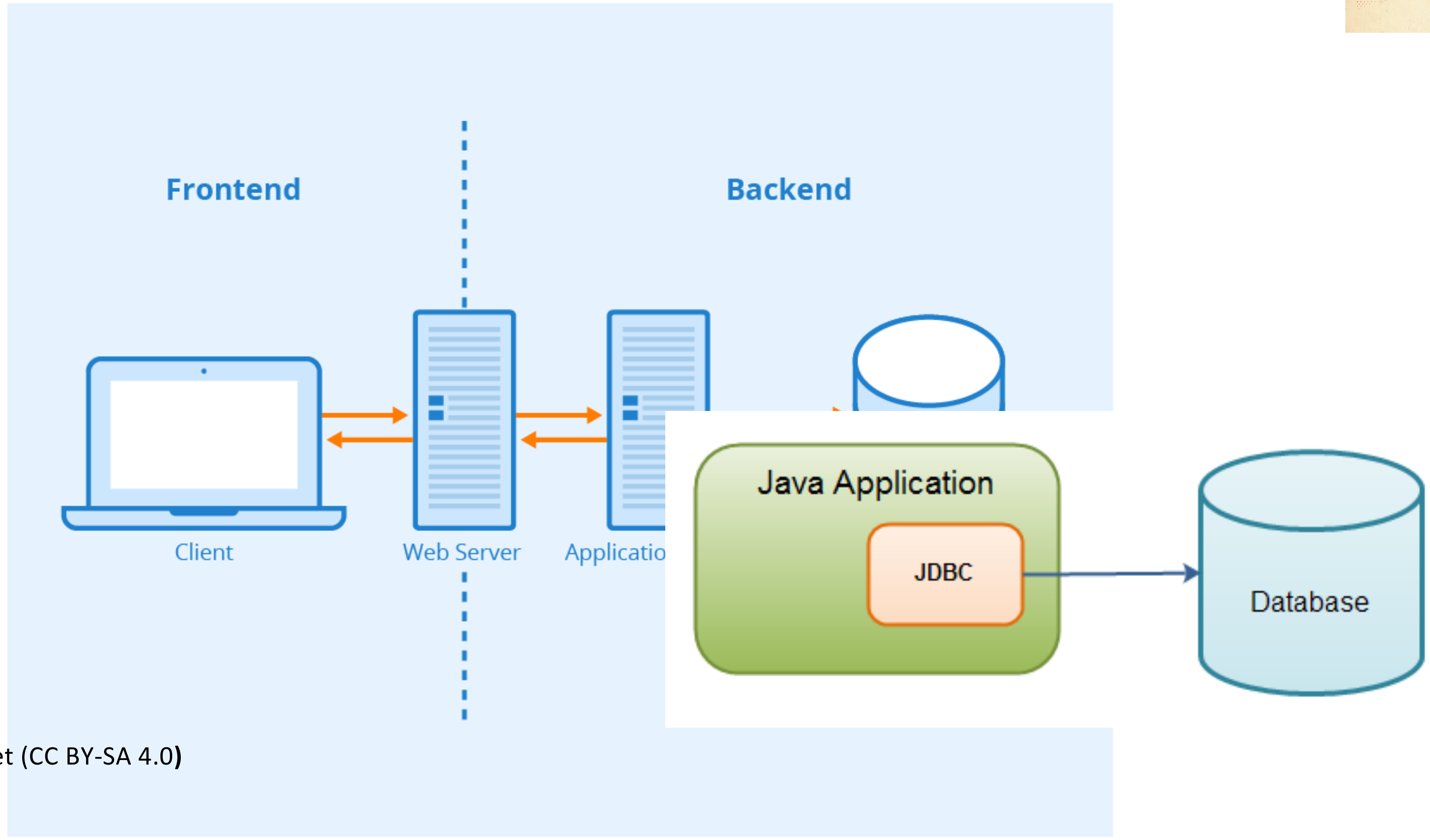




Some interesting insights about the dependency web of the Web:
https://www.synergylabs.org/yuvraj/docs/Kashaf_IMC2020_WebDep.pdf

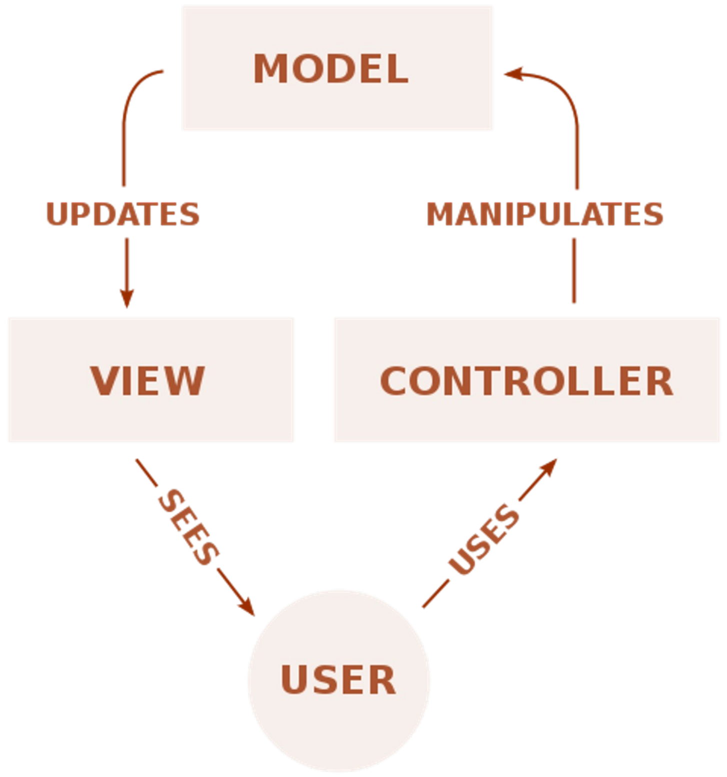
MONOLITHS

Monolithic styles



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

Monolithic styles: MVC Pattern



Monoliths

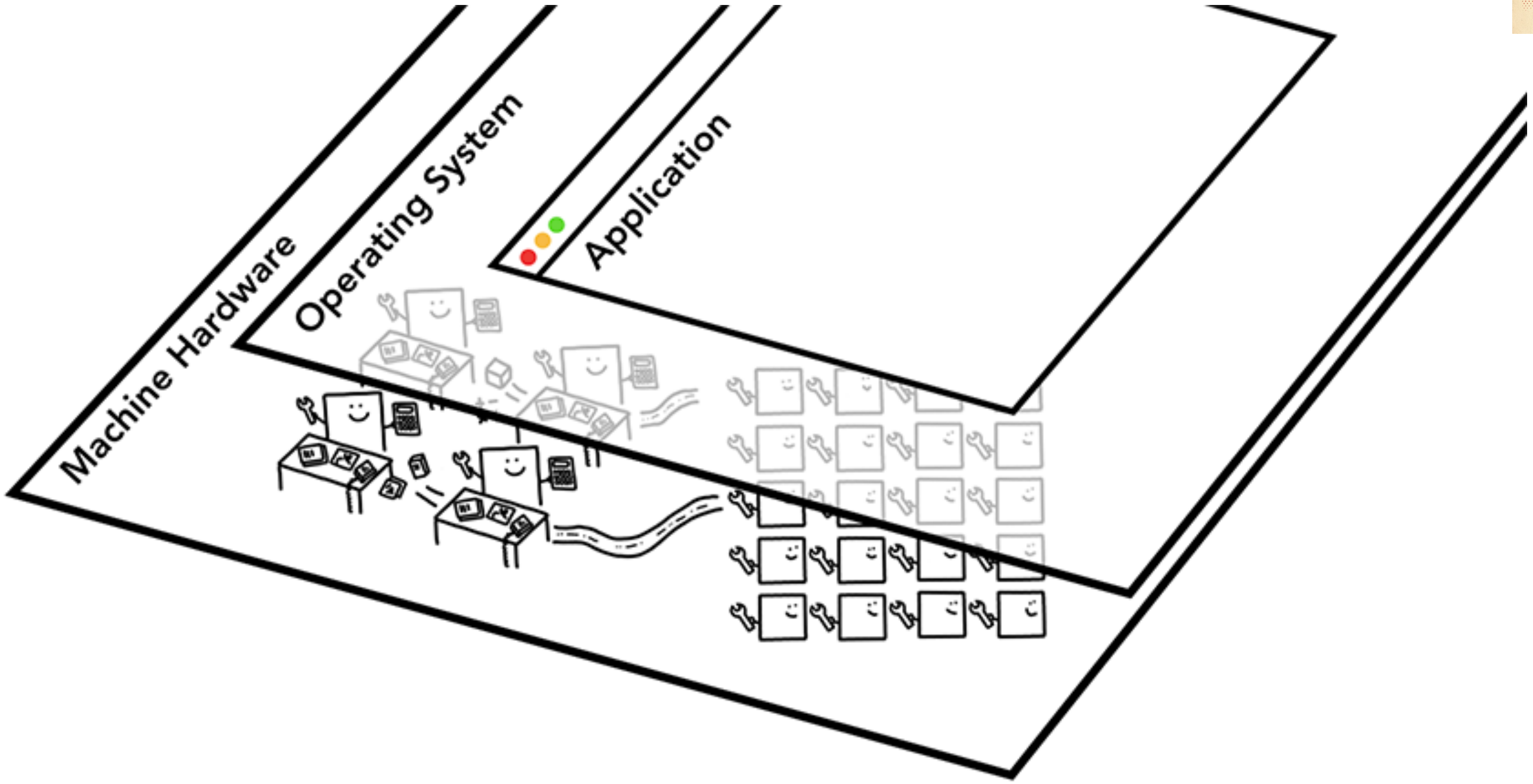
What are the consequences of this architecture? On:

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

Separation of concerns

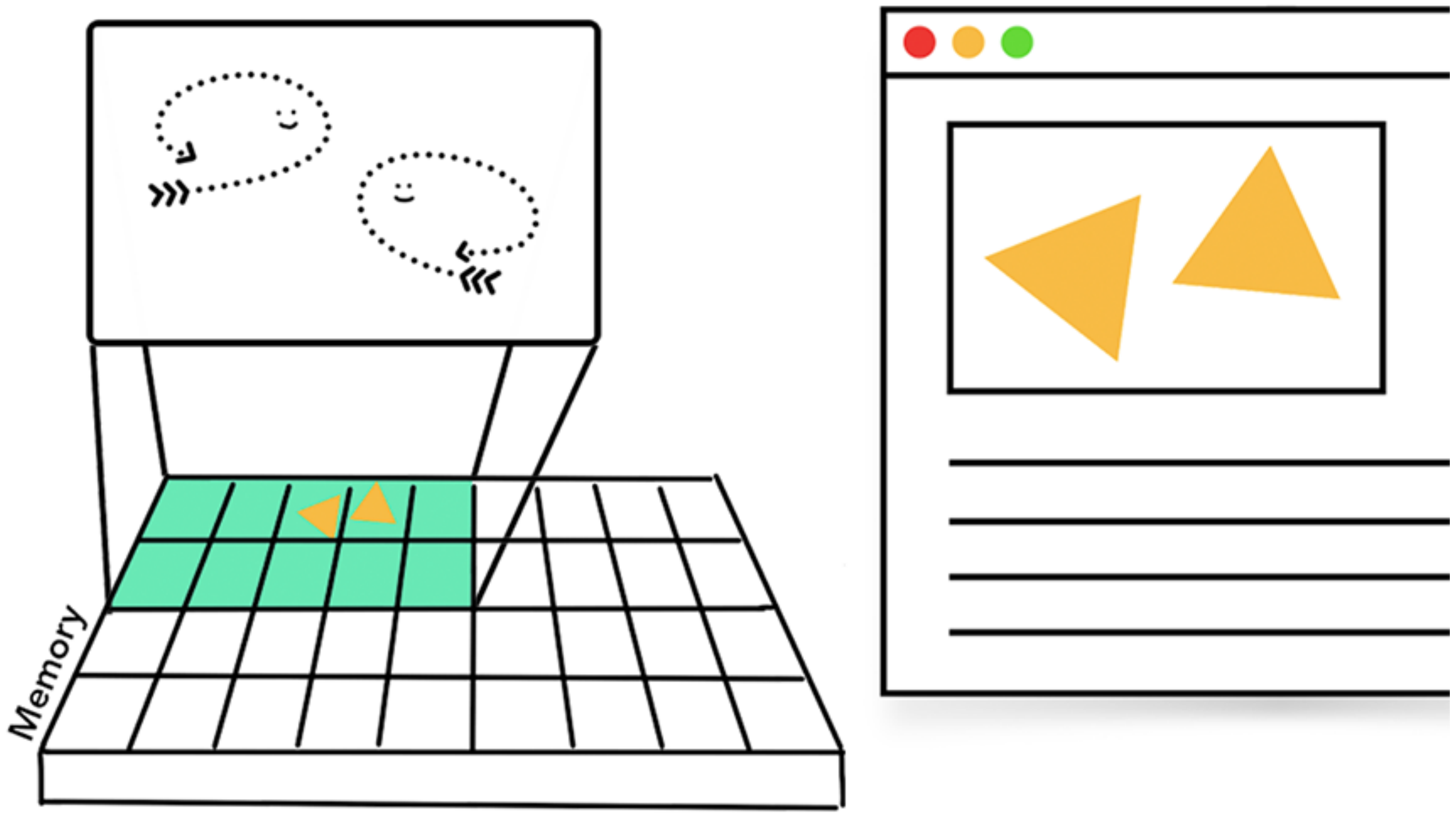
SERVICE-BASED ARCHITECTURE

Web Browsers



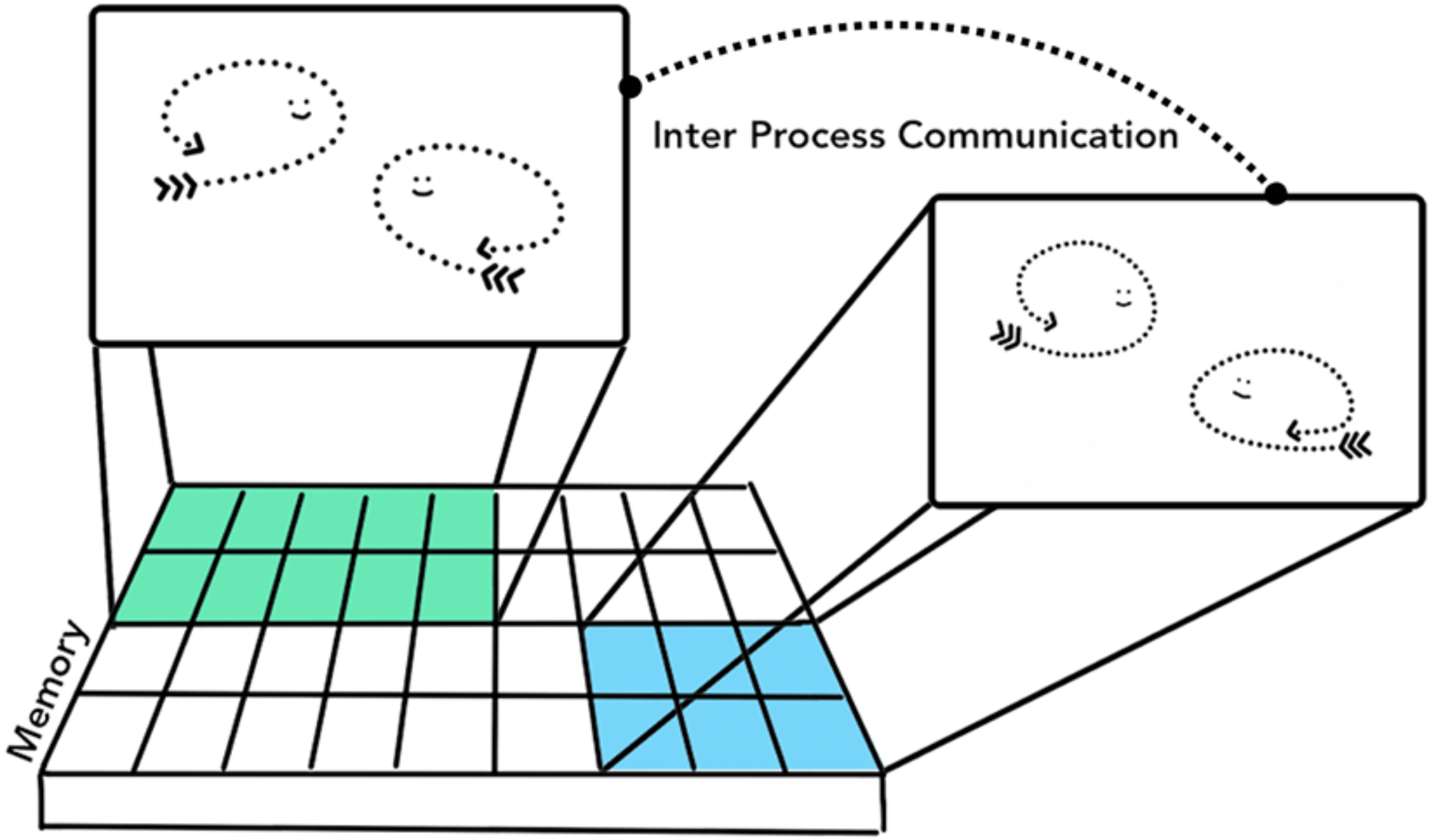
Source: https://developers.google.com/web/updates/2018/09/inside-browser-part1 (CC BY 4.0)

Browser: A multi-threaded process



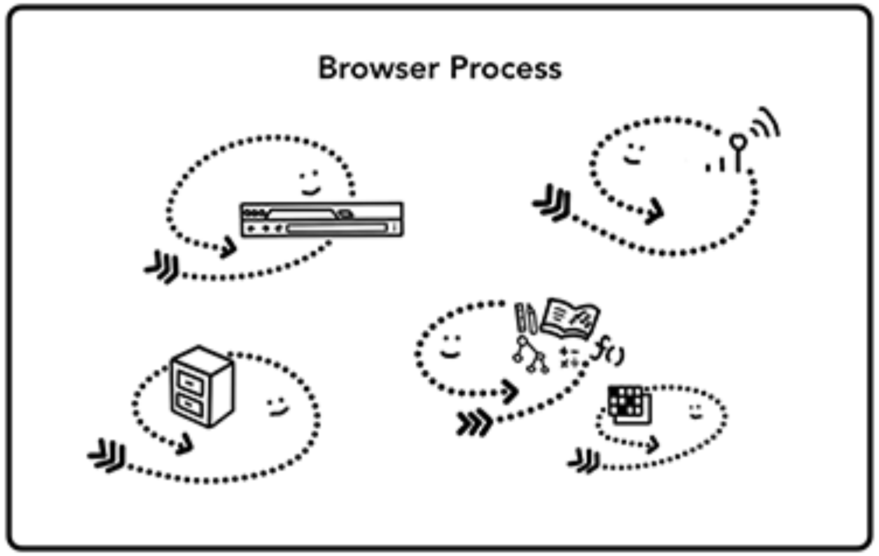
Source: https://developers.google.com/web/updates/2018/09/inside-browser-part1 (CC BY 4.0)

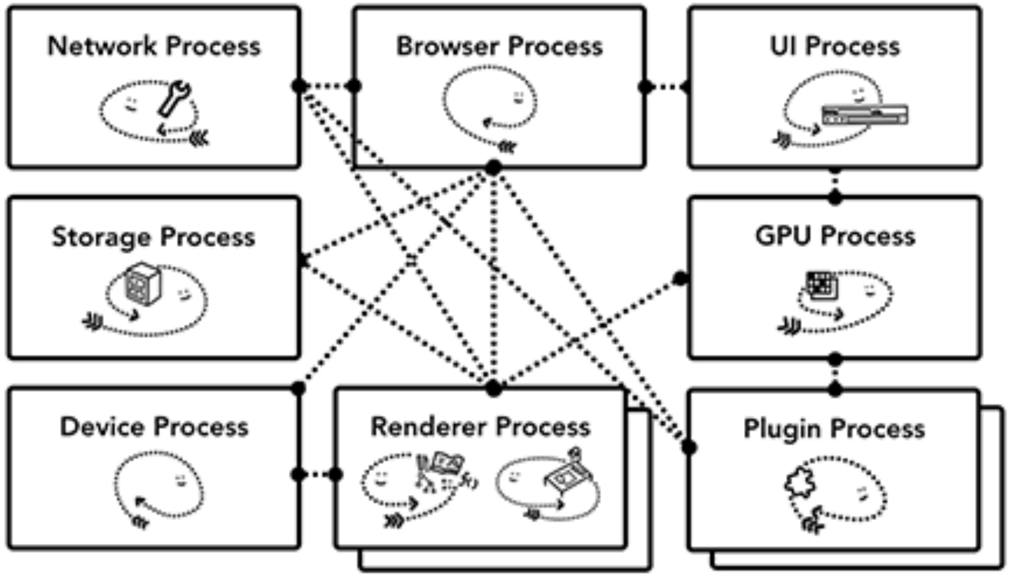
Multi-process browser with IPC



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

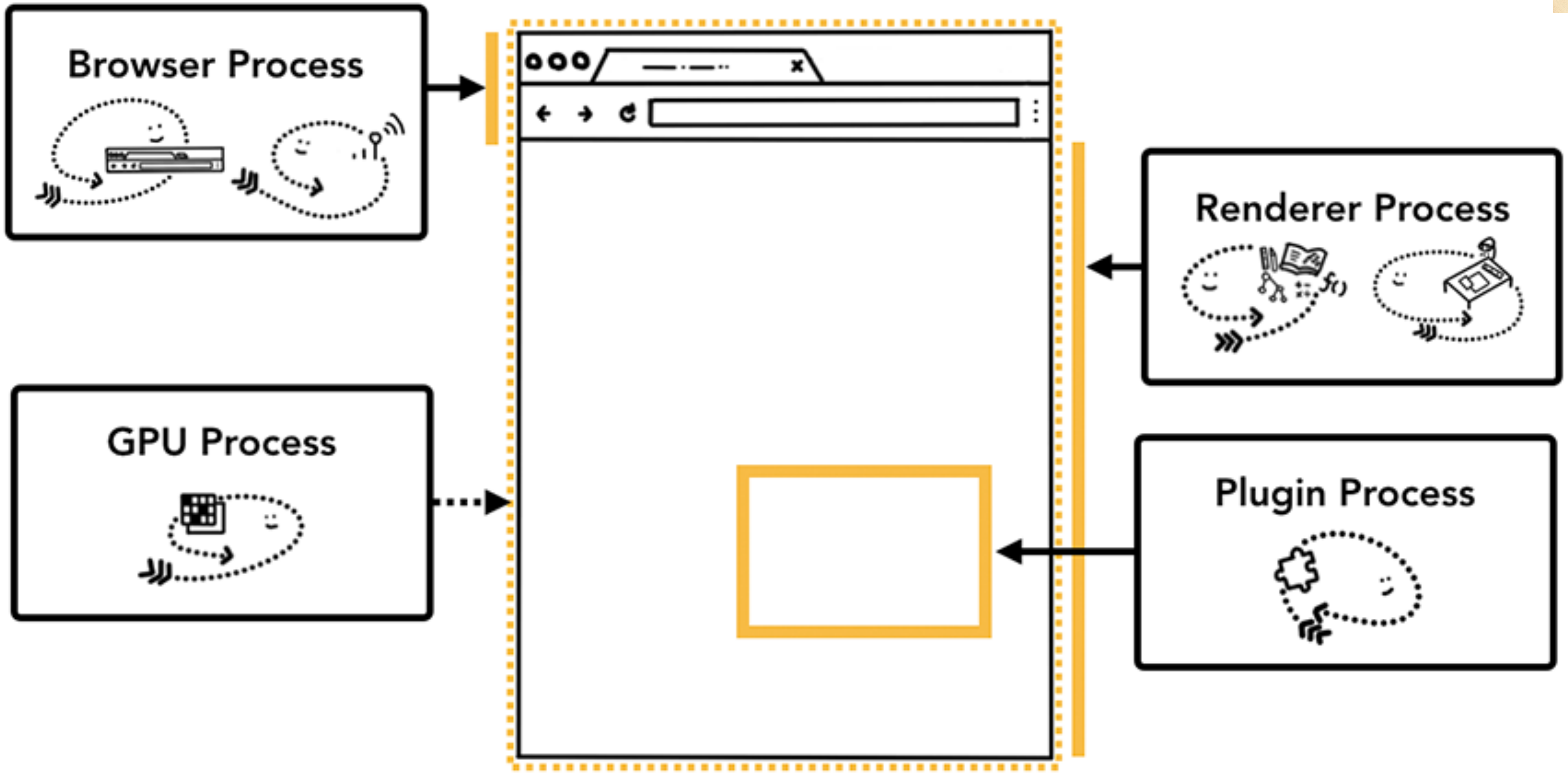
Browser Architectures





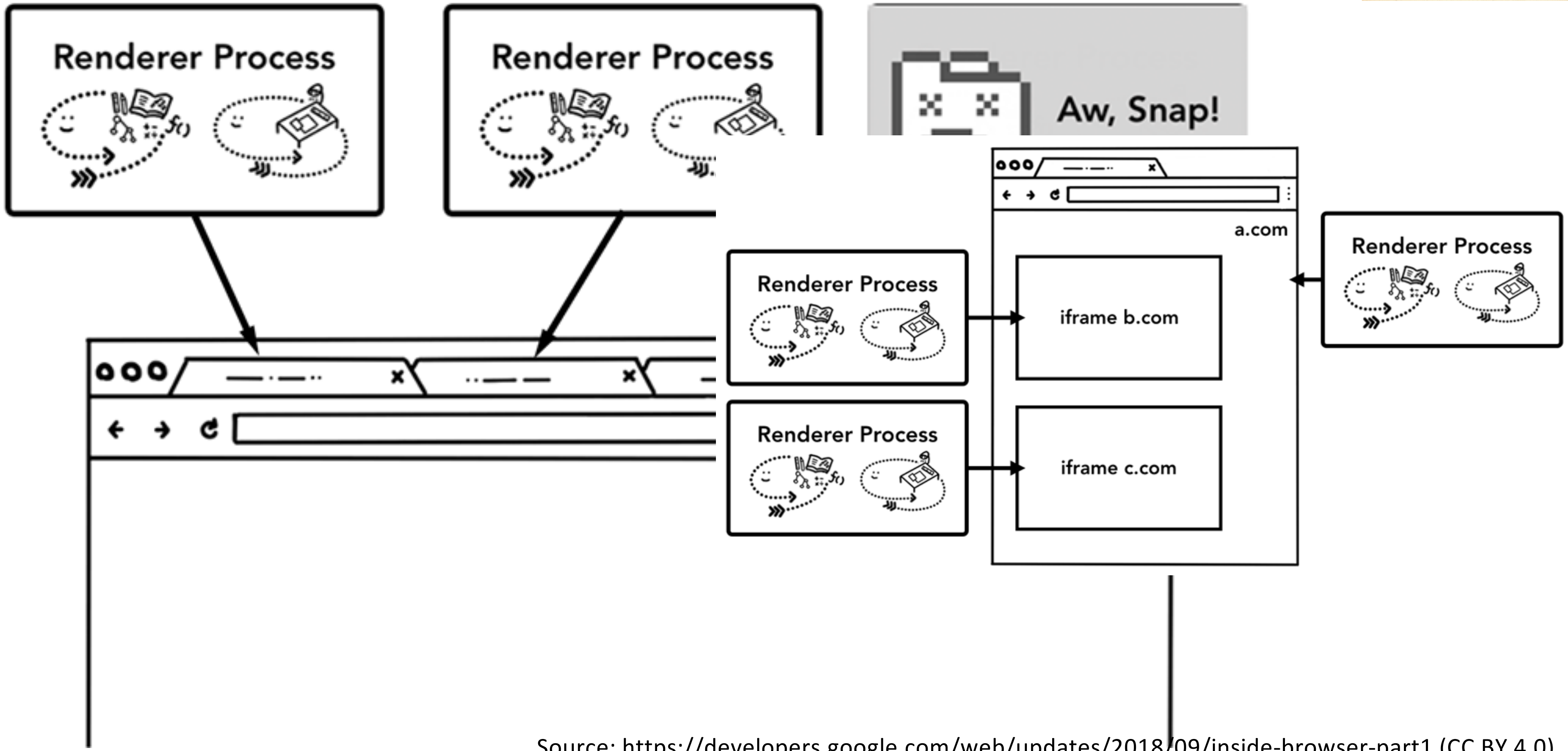
Source: https://developers.google.com/web/updates/2018/09/inside-browser-part1 (CC BY 4.0)

Service-based browser architecture



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

Service-based browser architecture

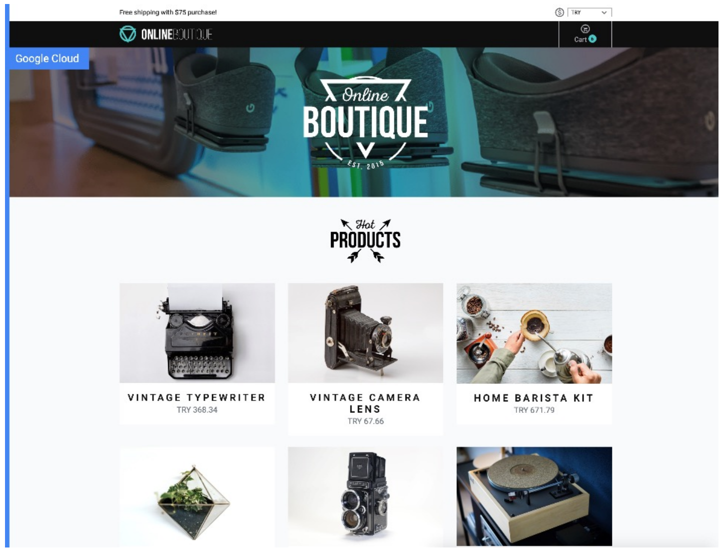


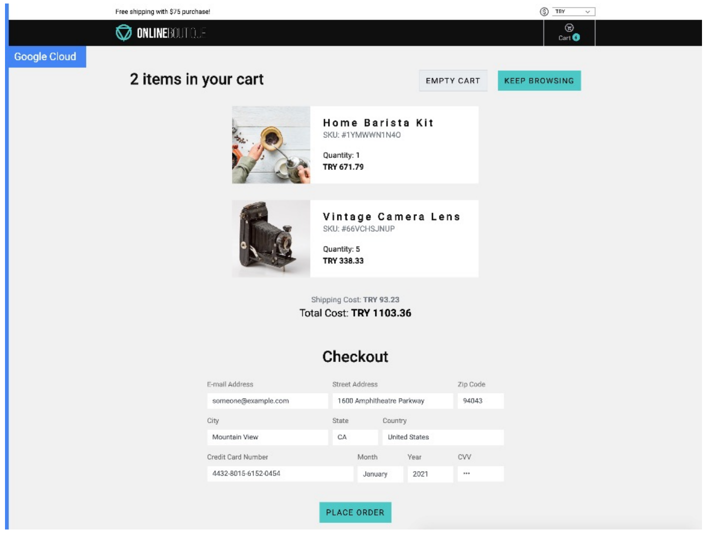
Source: https://developers.google.com/web/updates/2018/09/inside-browser-part1 (CC BY 4.0)

Taking it further

MICROSERVICES

Hipster Shop User Interface





2 items in your cart

Home Barista Kit
SKU: #1YMWWN1N4O
Quantity: 1
TRY 671.79

Vintage Camera Lens
SKU: #66VCHSJNUP
Quantity: 5
TRY 338.33

Shipping Cost: **TRY 93.23**
Total Cost: **TRY 1103.36**

Checkout

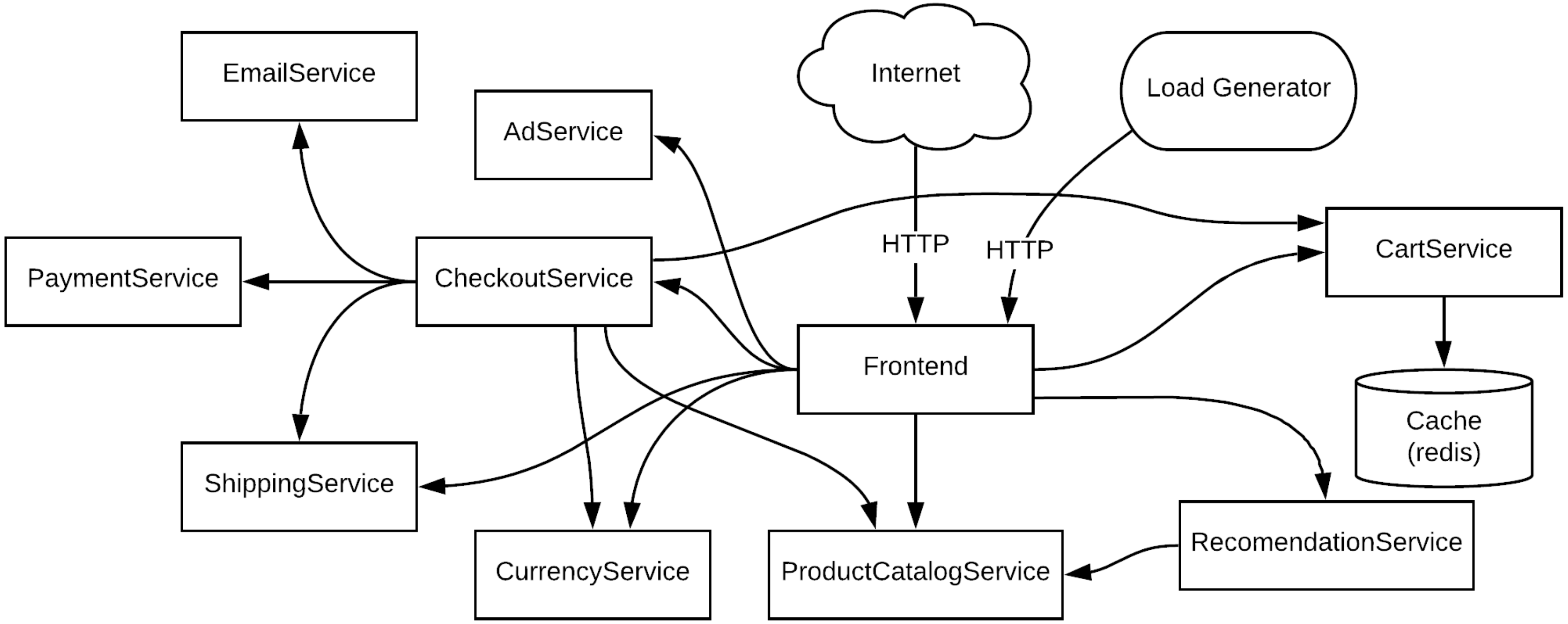
E-mail Address	Street Address	Zip Code
someone@example.com	1600 Amphitheatre Parkway	94043

City	State	Country
Mountain View	CA	United States

Credit Card Number	Month	Year	CVV
4432-8015-6152-0454	January	2021	•••

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

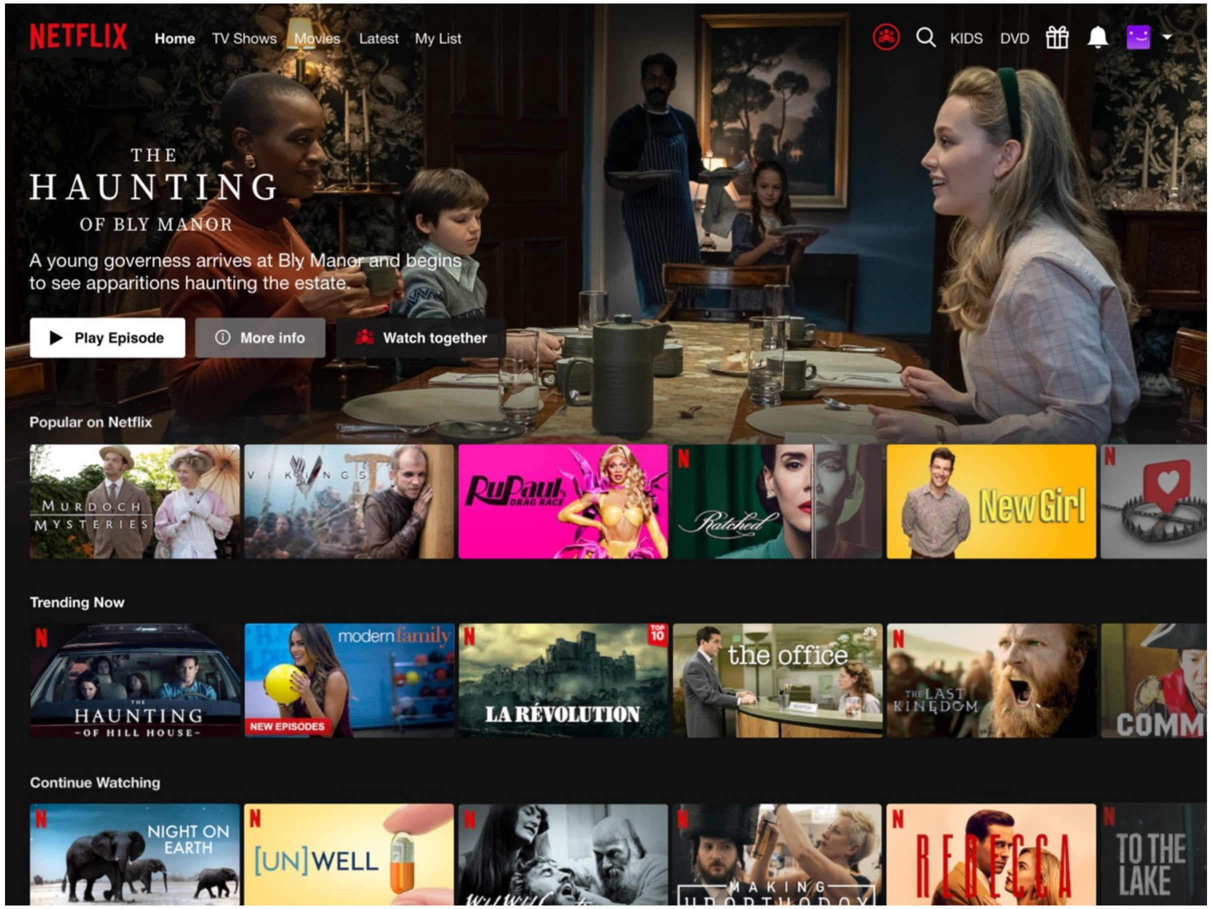
Hipster Shop Microservice Architecture



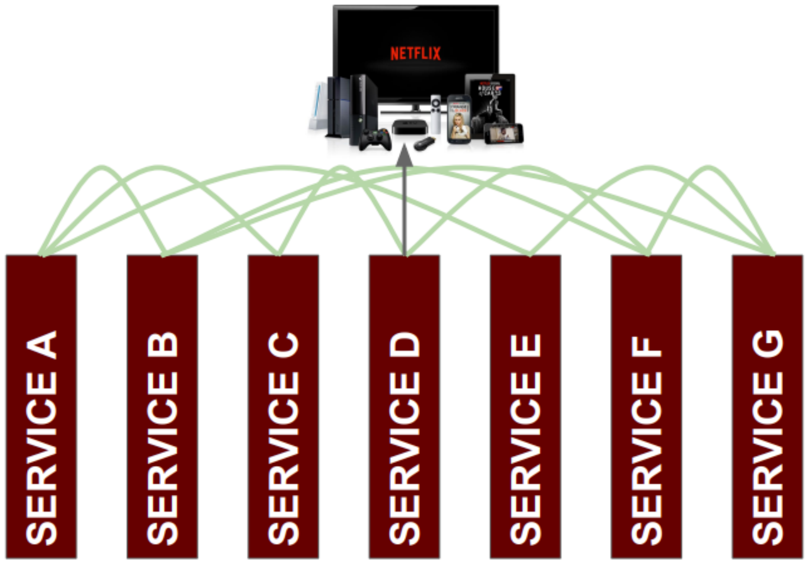
https://github.com/GoogleCloudPlatform/microservices-demo

Netflix

17-313
Foundations of
SOFTWARE
ENGINEERING


NETFLIX
Home
TV Shows
Movies
Latest
My List
KIDS
DVD
THE HAUNTING OF BLY MANOR
A young governess arrives at Bly Manor and begins to see apparitions haunting the estate.
Play Episode
More info
Watch together
Popular on Netflix
Trending Now
Continue Watching

AppBoot



Bookmarks

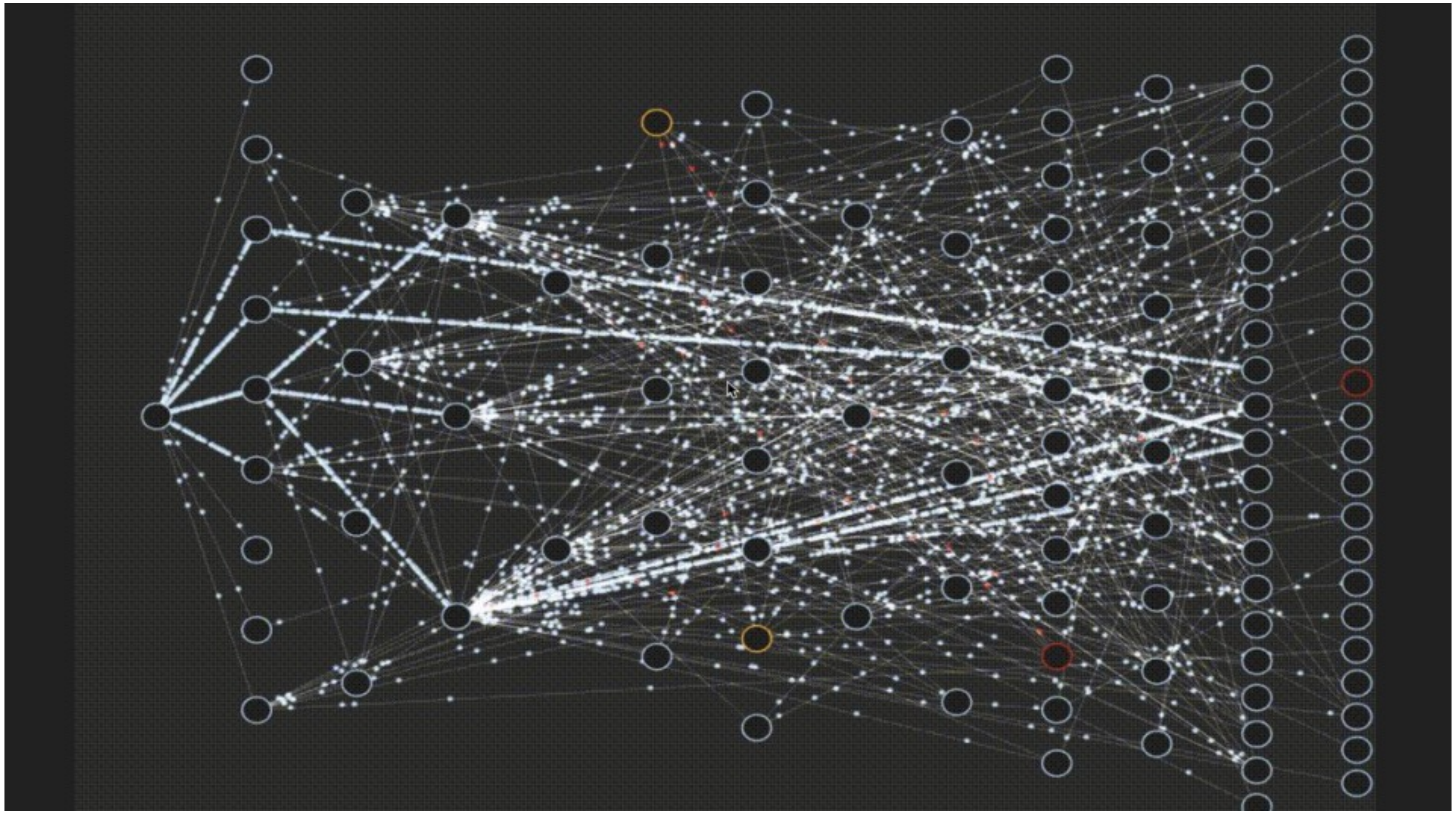
Recommendations

My List

Metrics

(as of 2016)

Netflix Microservices



(as of 2016)

https://www.youtube.com/watch?v=CZ3wIuvmHeM

Who uses Microservices?

17-313 Foundations of SOFTWARE ENGINEERING

COMCAST

NETFLIX

ebay

UBER

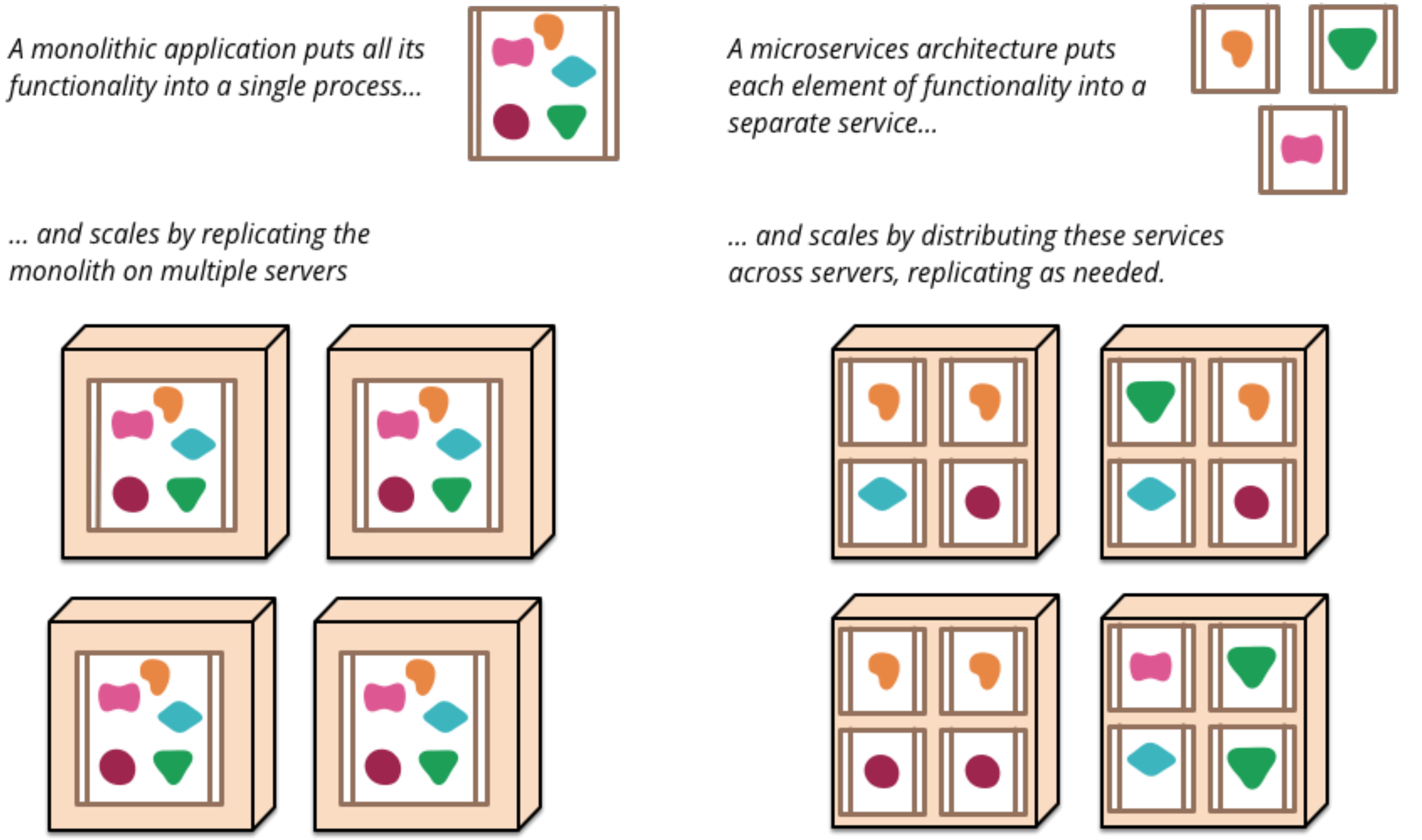
GROUPON

Microservices

What are the consequences of this architecture? On:

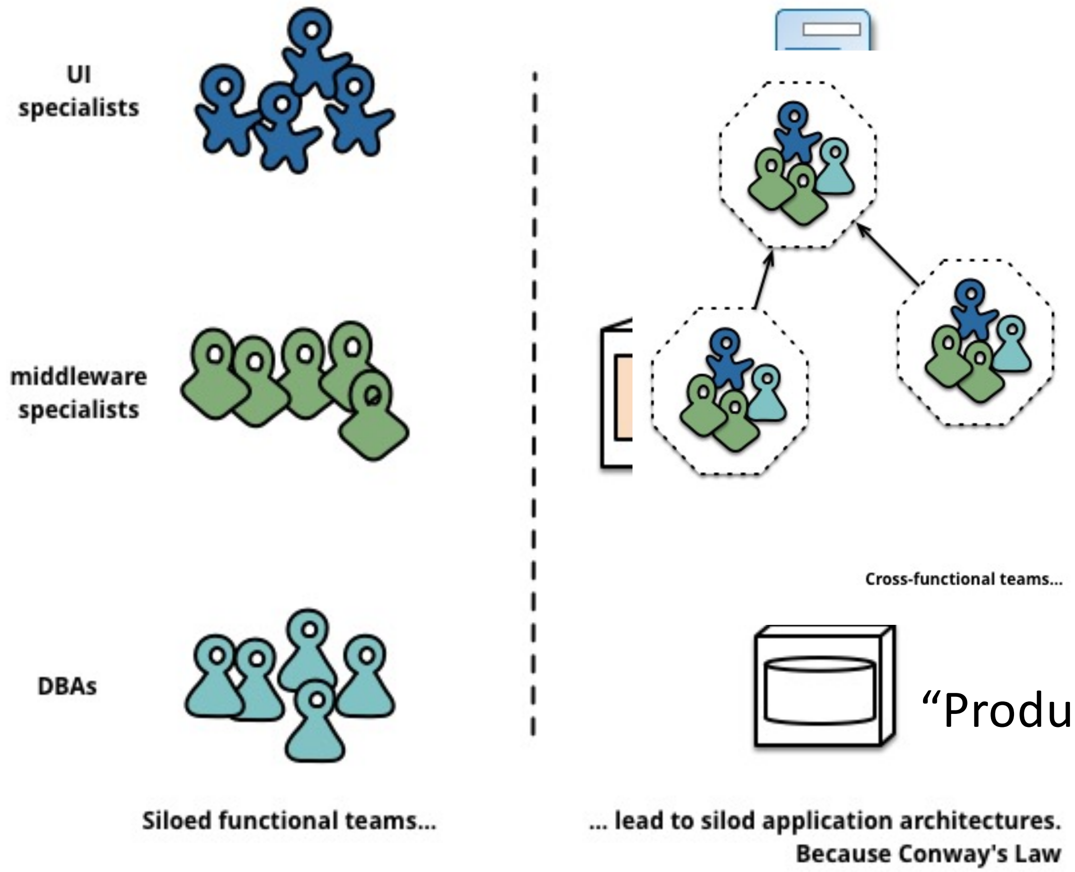
- Scalability
- Reliability
- Performance
- Development
- Maintainability
- Evolution
- Testability
- Ownership
- Data Consistency

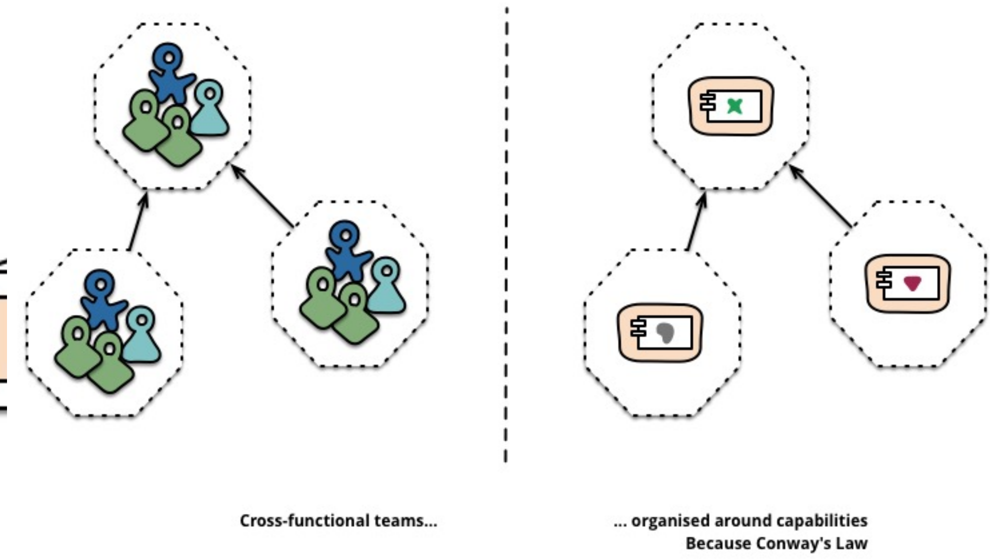
Scalability


A monolithic application puts all its functionality into a single process...
A microservices architecture puts each element of functionality into a separate service...
... and scales by replicating the monolith on multiple servers
... and scales by distributing these services across servers, replicating as needed.

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

Team Organization (Conway's Law)

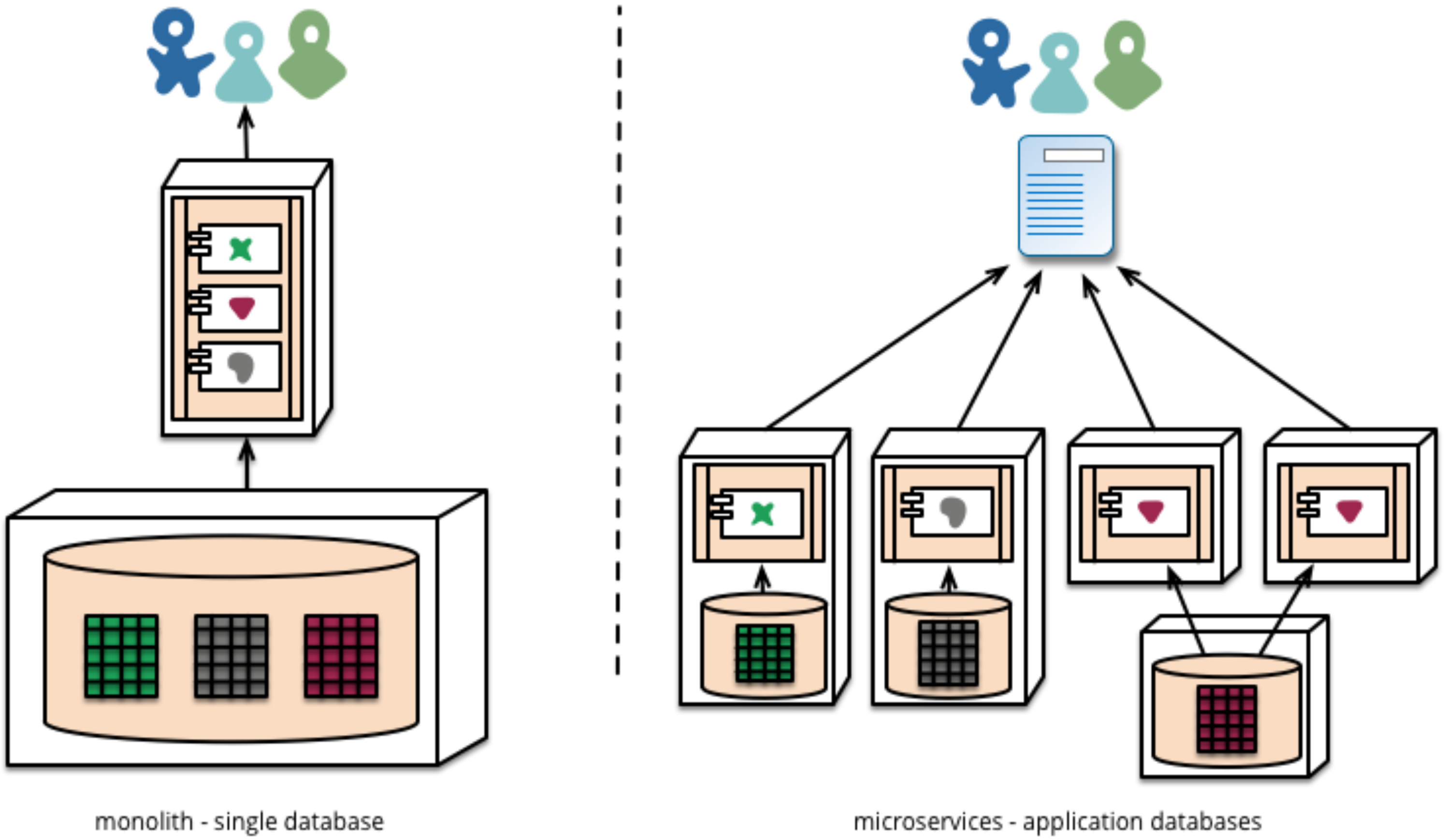

UI specialists
middleware specialists
DBAs
Siloed functional teams...
... lead to silod application architectures.
Because Conway's Law



"Products" not "Projects"

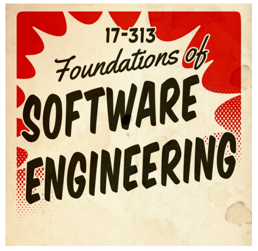
Source: http://martinfowler.com/articles/microservices.html

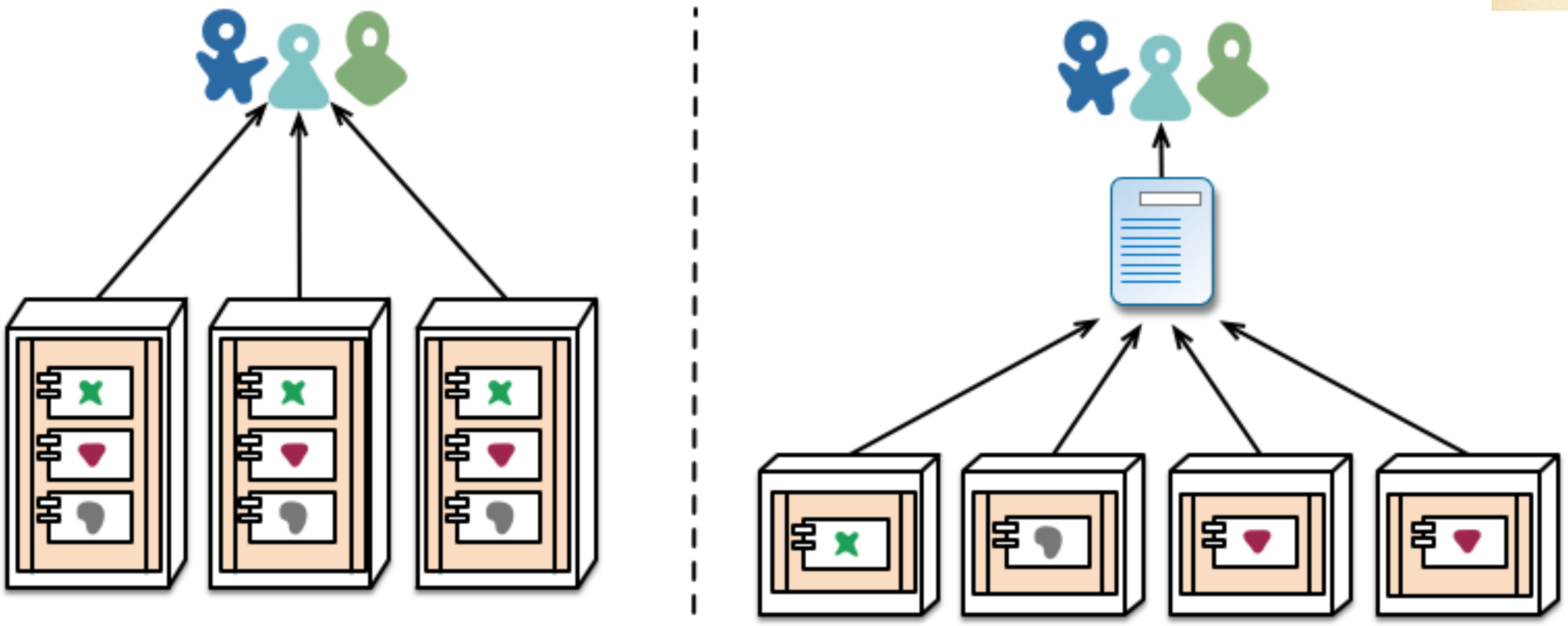
Data Management and Consistency


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

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
  - Schemas?

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

Taking it to the extreme

SERVERLESS

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

Microservices

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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 architecture.

A microservice example

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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

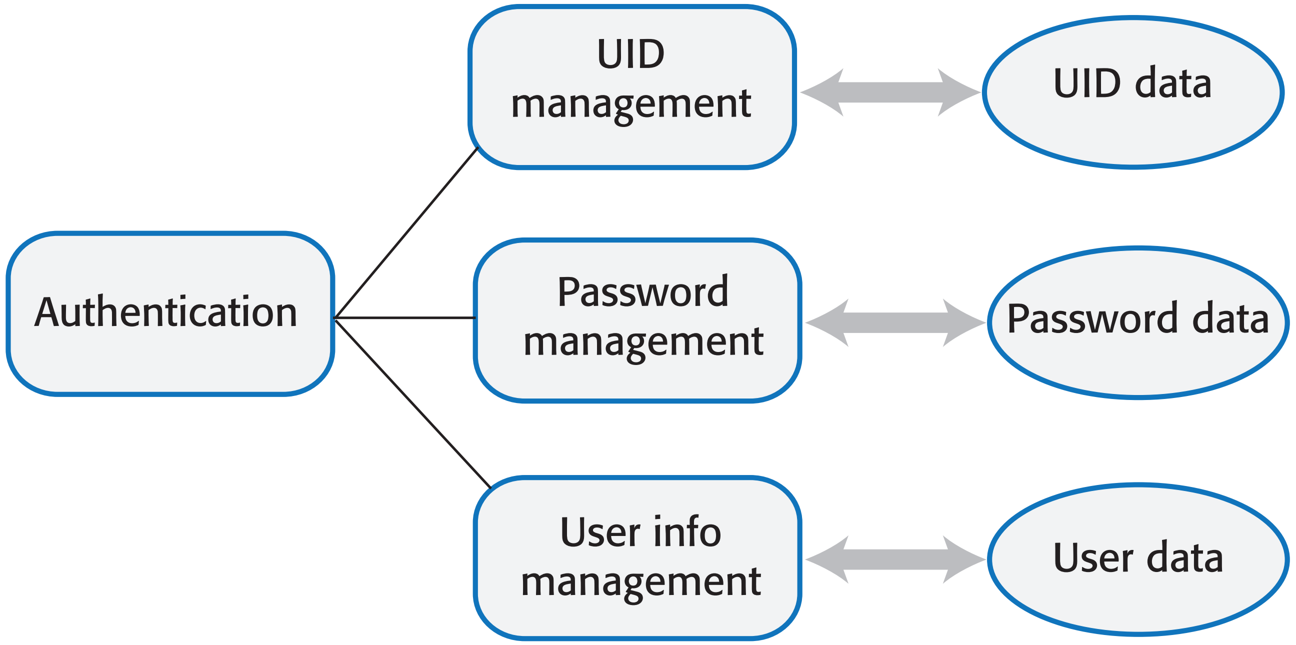
- Setup new login id
- Setup new password
- Setup password recovery information
- Setup two-factor authentication
- Confirm registration

Authenticate using UID/password

- Get login id
- Get password
- Check credentials
- Confirm authentication

Authentication microservices

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



Characteristics of microservices

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- **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

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A well-designed microservice should have high cohesion and l 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.

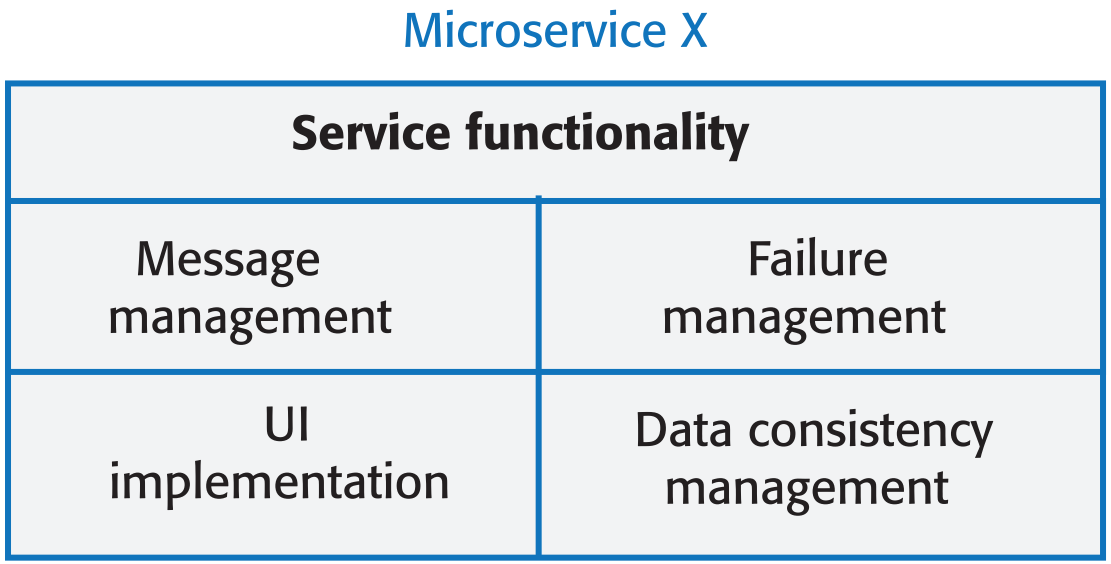
Password management functionality

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

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



RESTful services

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- ***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 delays, road closures 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

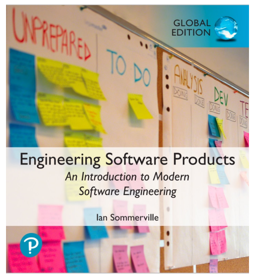
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

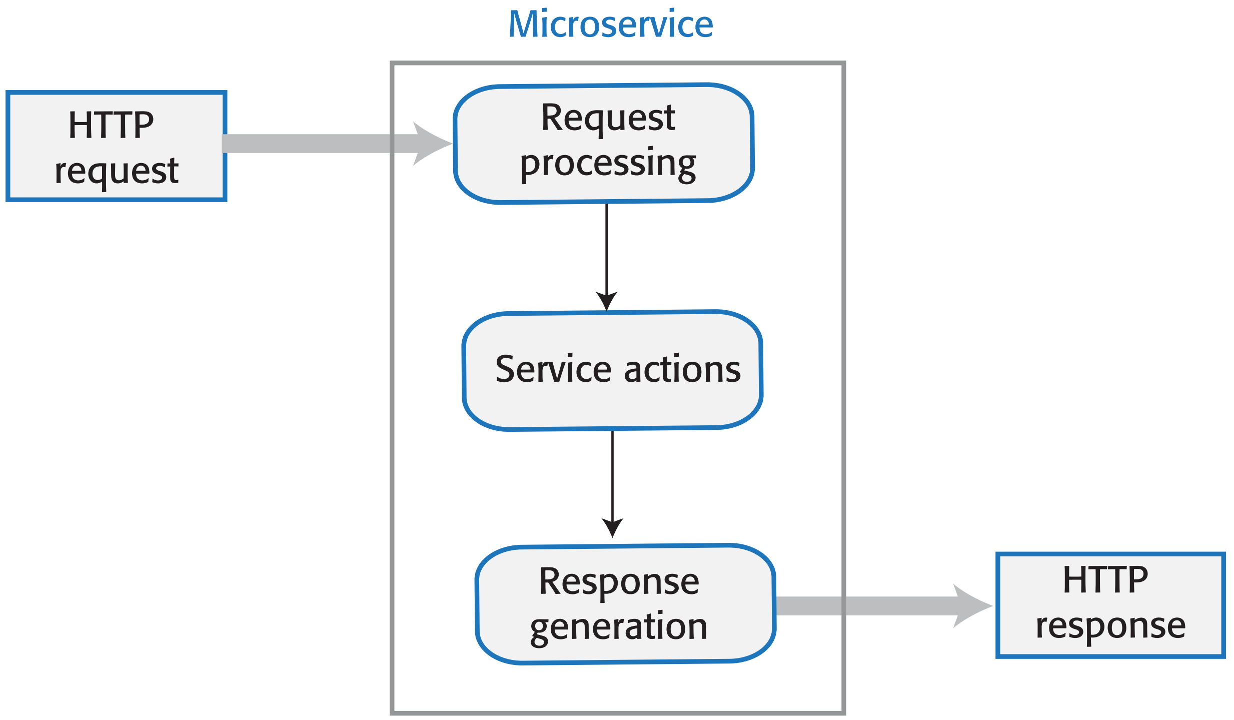
- Incident ID: A90N17061714391
- Date: 17 June 2017
- Time reported: 1439
- Severity: Significant
- Description: Broken-down bus on north carriageway. One lane closed. Expect delays of up to 30 minutes

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 processing


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



HTTP request and response message organization

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

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

UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

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

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

REQUEST

```
GET | incidents/A90/stonehaven/ | HTTP/1.1

Host: trafficinfo.net
...
Accept: text/json, text/xml, text/plain
Content-Length: 0
```

RESPONSE

```
HTTP/1.1 | 200

...
Content-Length: 461
Content-Type: text/json

{
   “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”
}
```

Service deployment

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

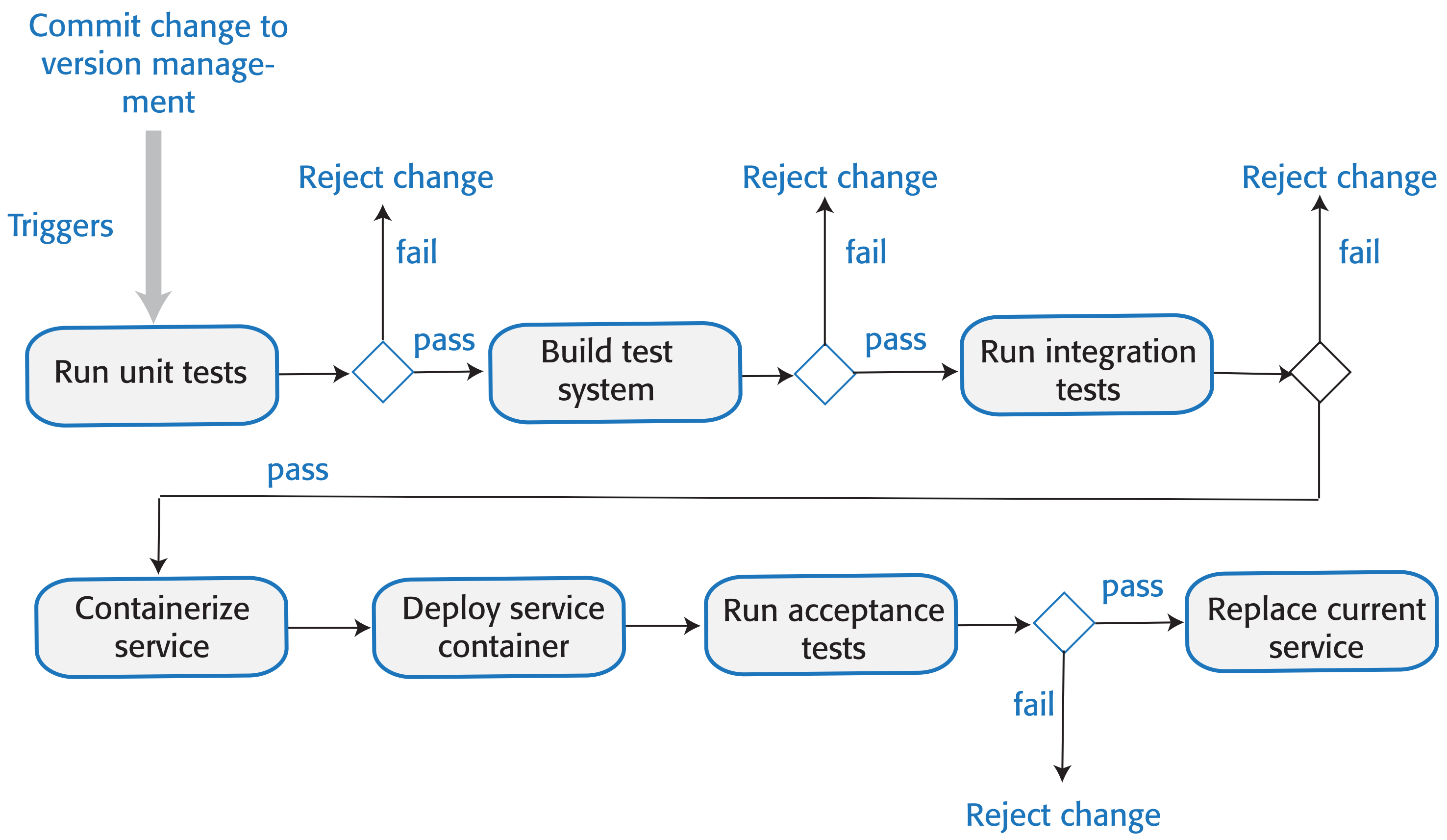
- After a system has been developed and delivered, it has to be deployed on servers, monitored for problems and updated as new versions become available.
- When a system is composed of tens or even hundreds of microservices, deployment of the system is more complex than for monolithic systems.
- The 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

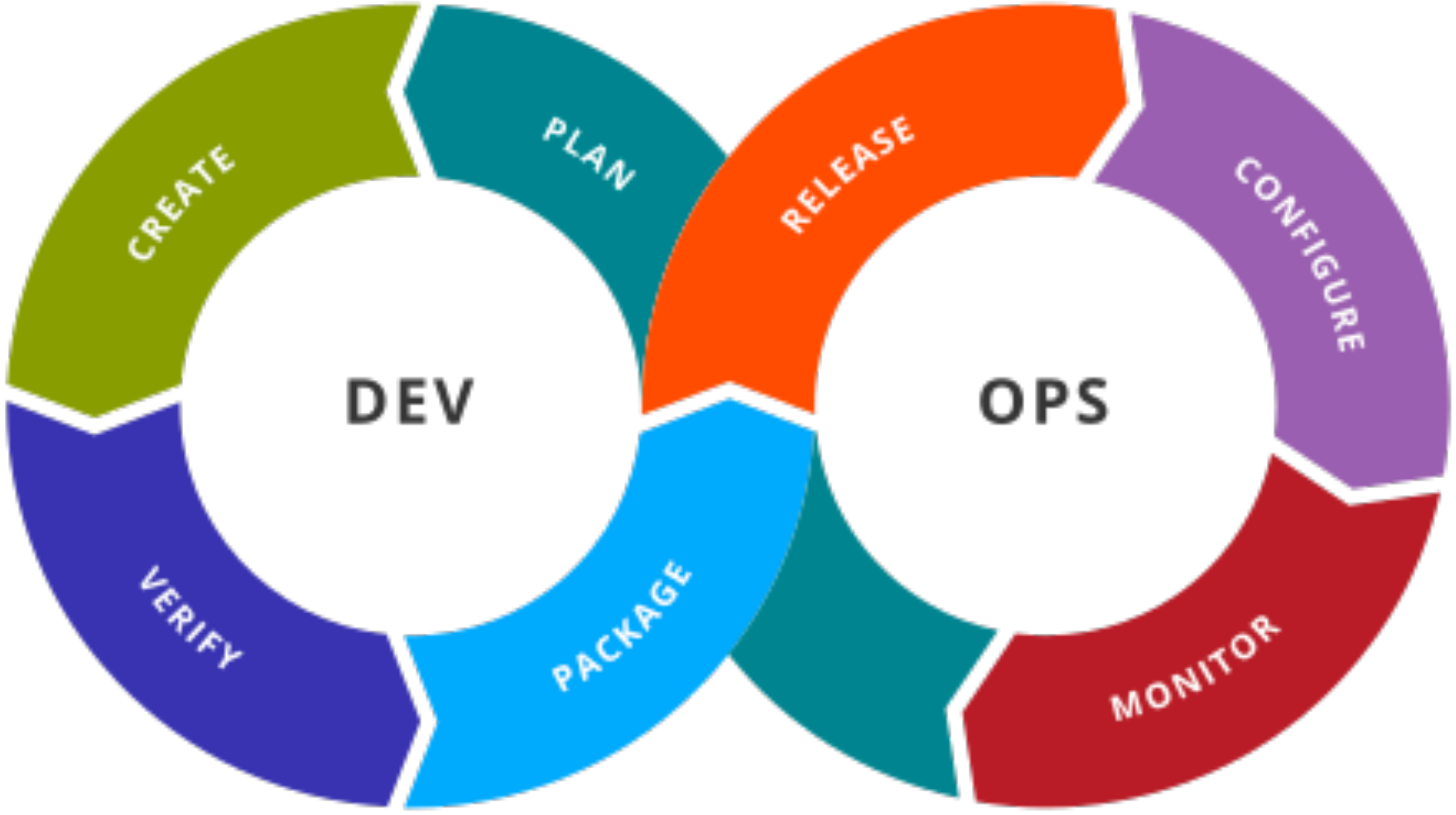
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- 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



Next Week: DevOps



Key Points

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Software architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- The architecture of a software system has a significant influence on non-functional system properties such as reliability, efficiency and security.
- Architectural design involves understanding the issues that are critical for your product and creating system descriptions that shows components and their relationships.
- The principal role of architectural descriptions is to provide a basis for the development team to discuss the system organization. Informal architectural diagrams are effective in architectural description because they are fast and easy to draw and share.
- System decomposition involves analyzing architectural components and representing them as a set of finer-grain components.

Key Points

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- To minimize complexity, you should separate concerns, avoid functional duplication and focus on component interfaces.
- Web-based systems often have a common layered structure including user interface layers, application-specific layers and a database layer.
- The distribution architecture in a system defines the organization of the servers in that system and the allocation of components to these servers.
- Multi-tier client-server and service-oriented architectures are the most commonly used architectures for web-based systems.
- Making decisions on technologies such as database and cloud technologies are an important part of the architectural design process.

Key Points

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- A microservice is an independent and self-contained software component that runs in its own process and communicates with other microservices using lightweight protocols.
- Microservices in a system can be implemented using different programming languages and database technologies.
- Microservices have a single responsibility and should be designed so that they can be easily changed without having to change other microservices in the system.
- Microservices architecture is an architectural style in which the system is constructed from communicating microservices. It is well-suited to cloud based systems where each microservice can run in its own container.
- The two most important responsibilities of architects of a microservices system are to decide how to structure the system into microservices and to decide how microservices should communicate and be coordinated.

Key Points

- Communication and coordination decisions include deciding on microservice communication protocols, data sharing, whether services should be centrally coordinated, and failure management.
- The RESTful architectural style is widely used in microservice-based systems. Services are designed so that the HTTP verbs, GET, POST, PUT and DELETE, map onto the service operations.
- The RESTful style is based on digital resources that, in a microservices architecture, may be represented using XML or, more commonly, JSON.
- Continuous deployment is a process where new versions of a service are put into production as soon as a service change has been made. It is a completely automated process that relies on automated testing to check that the new version is of 'production quality'.
- If continuous deployment is used, you may need to maintain multiple versions of deployed services so that you can switch to an older version if problems are discovered in a newly-deployed service.

Key Points

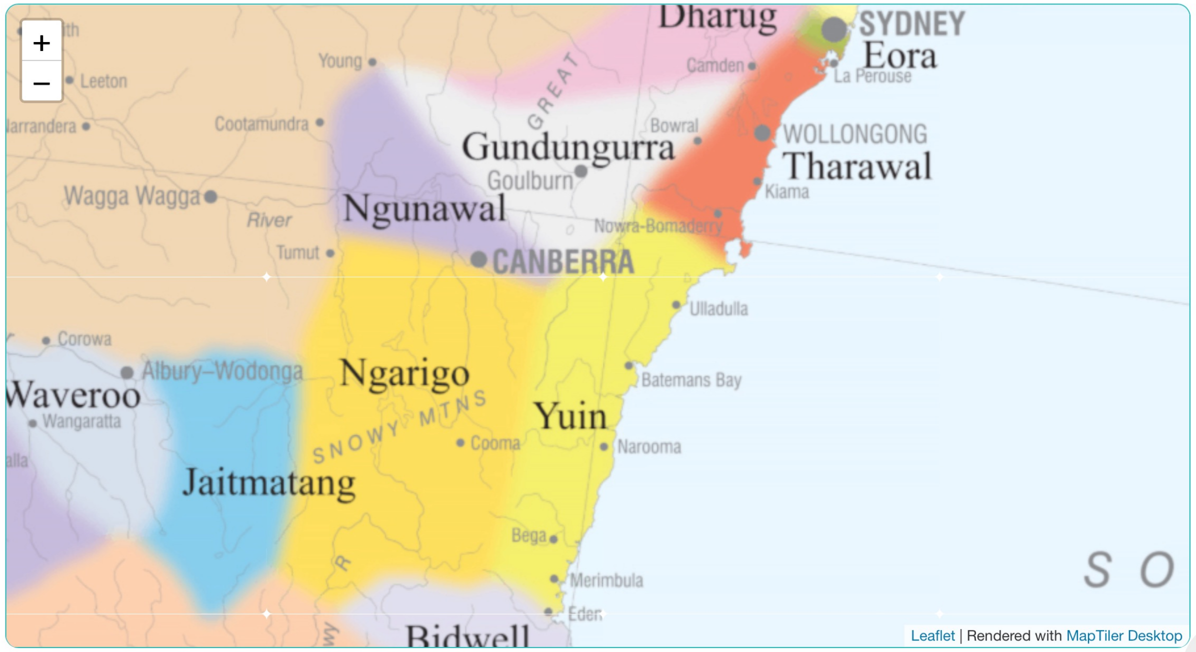
- Contrast the monolithic application design with a modular design based on microservices.
- Reason about how architectural choices affect software quality and process attributes.
- Reason about tradeoffs of microservices architectures.

- **Inspirations**:
- Martin Fowler (http://martinfowler.com/articles/microservices.html)
- Josh Evans @ Netflix (https://www.youtube.com/watch?v=CZ3wIuvmHeM)
- Matt Ranney @ Uber (https://www.youtube.com/watch?v=kb-m2fasdDY)
- Christopher Meiklejohn & Filibuster (http://filibuster.cloud)

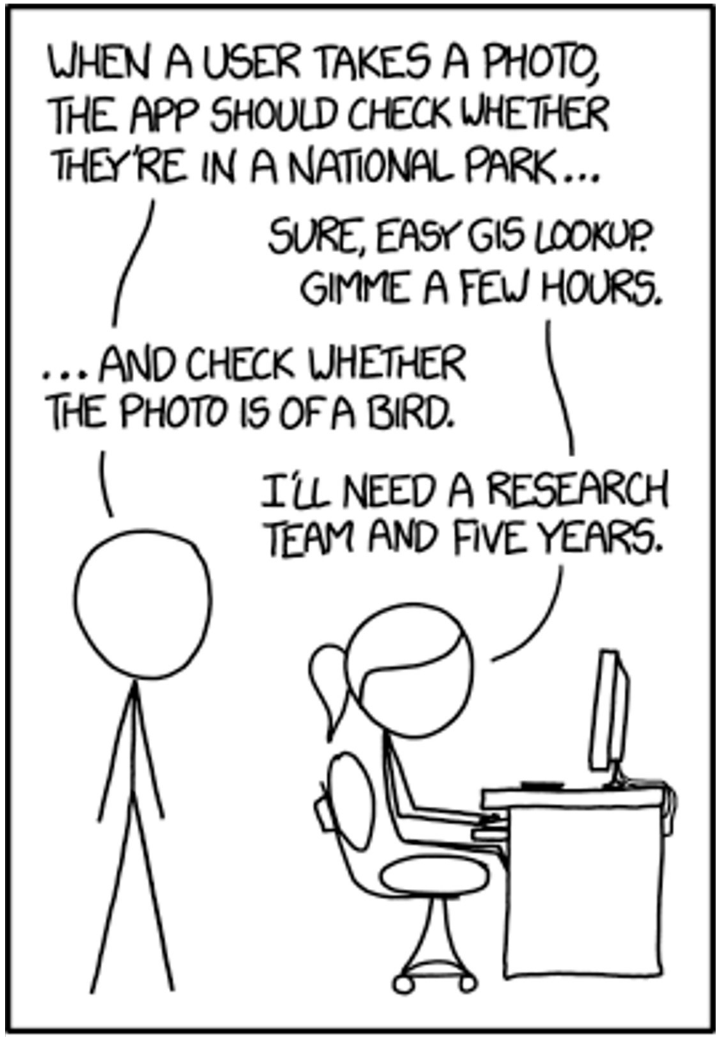
End of Monday Lecture/Start of Tuesday Lecture

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."



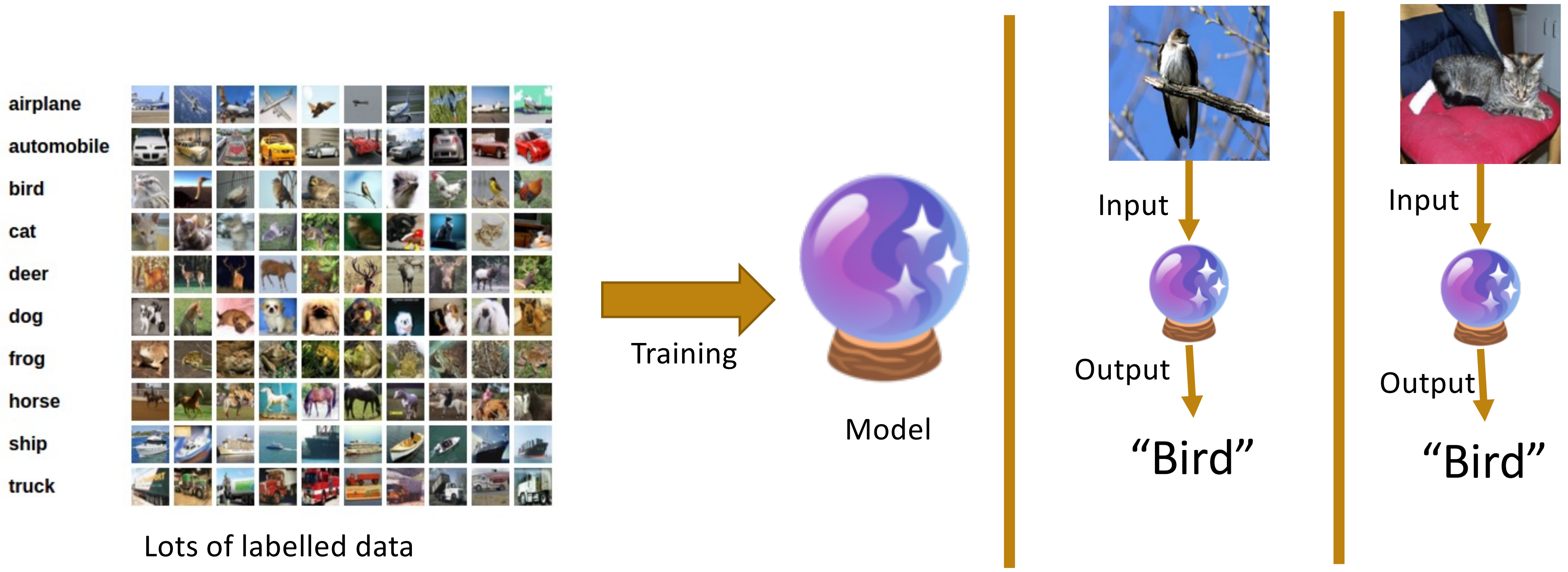
https://aiatsis.gov.au/explore/map-indigenous-australia


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.

Source: https://xkcd.com/1425/

Machine Learning in One Slide

(Supervised)

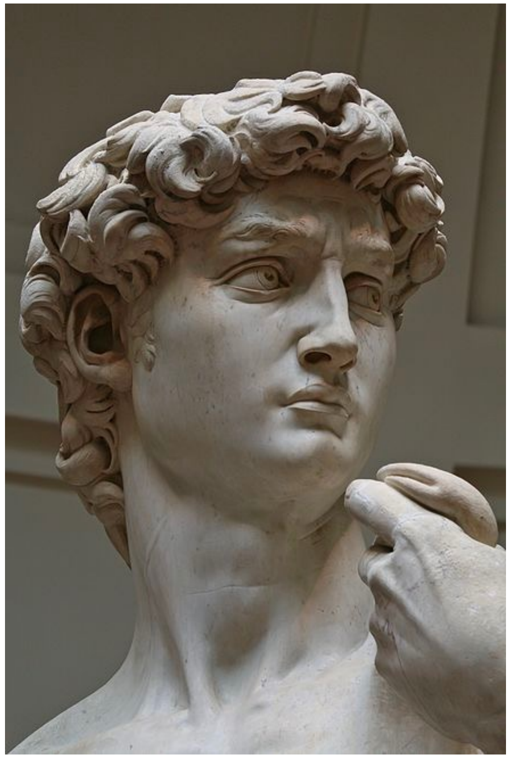


Lots of labelled data
(Inputs, outputs)

Traditional Software Development

17-313 Foundations of SOFTWARE ENGINEERING

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



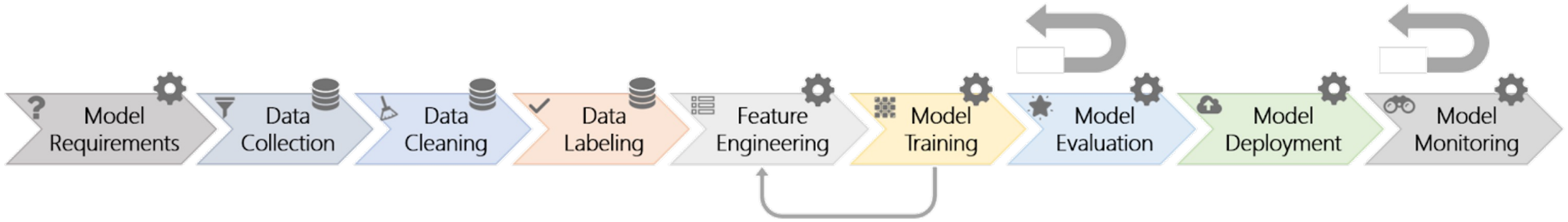
ML Development

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

Black-box View of Machine Learning

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Microsoft’s view of Software Engineering for ML



Source: “Software Engineering for Machine Learning: A Case Study” by Amershi et al. ICSE 2019

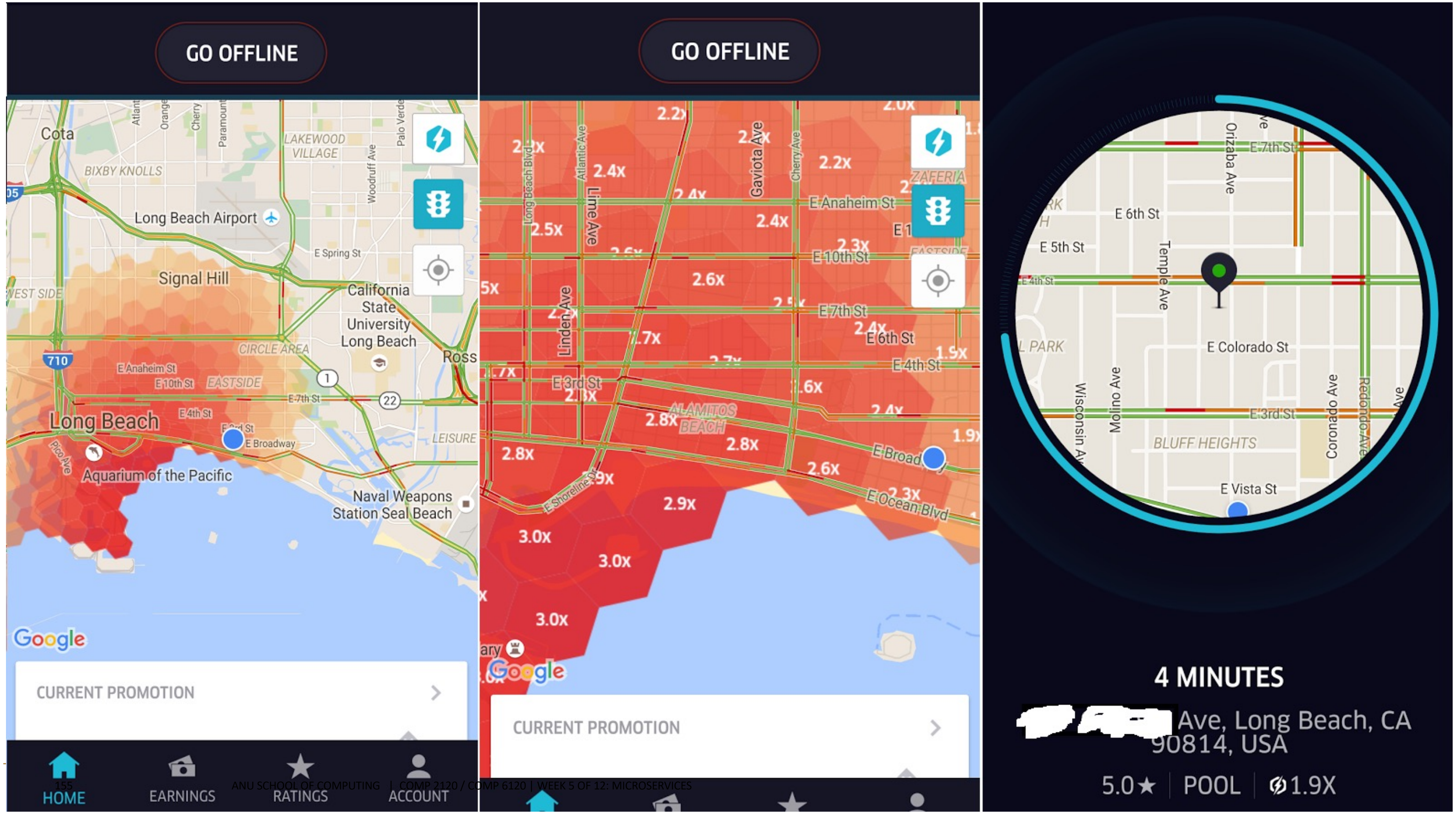
Three Fundamental Differences:

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

WHAT CHALLENGES ARE THERE IN BUILDING AND DEPLOYING ML?

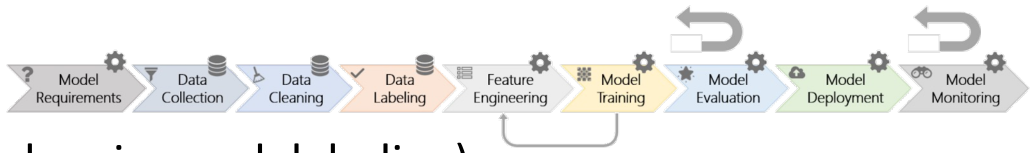
現金のみ
Japanese
English
cash only

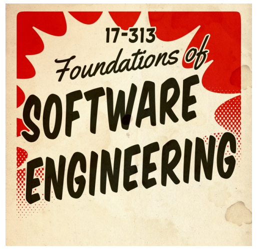
PhoDUCK
명동교자
아하 노래방
POTALA
TIBETAN INDIAN NEPALI FOOD
POTALA RESTAURANT
MINI BEER PUB
빤닭빤닭
DAKGALBI
명동본점 2F
정성본
칼국수
日本語歌完全具備
六千曲收
김밥
노래방


GO OFFLINE
CURRENT PROMOTION
HOME
EARNINGS
RATINGS
ACCOUNT
GO OFFLINE
CURRENT PROMOTION
4 MINUTES
[redacted] Ave, Long Beach, CA 90814, USA
5.0★ | POOL | 1.9X

MACHINE LEARNING PIPELINE

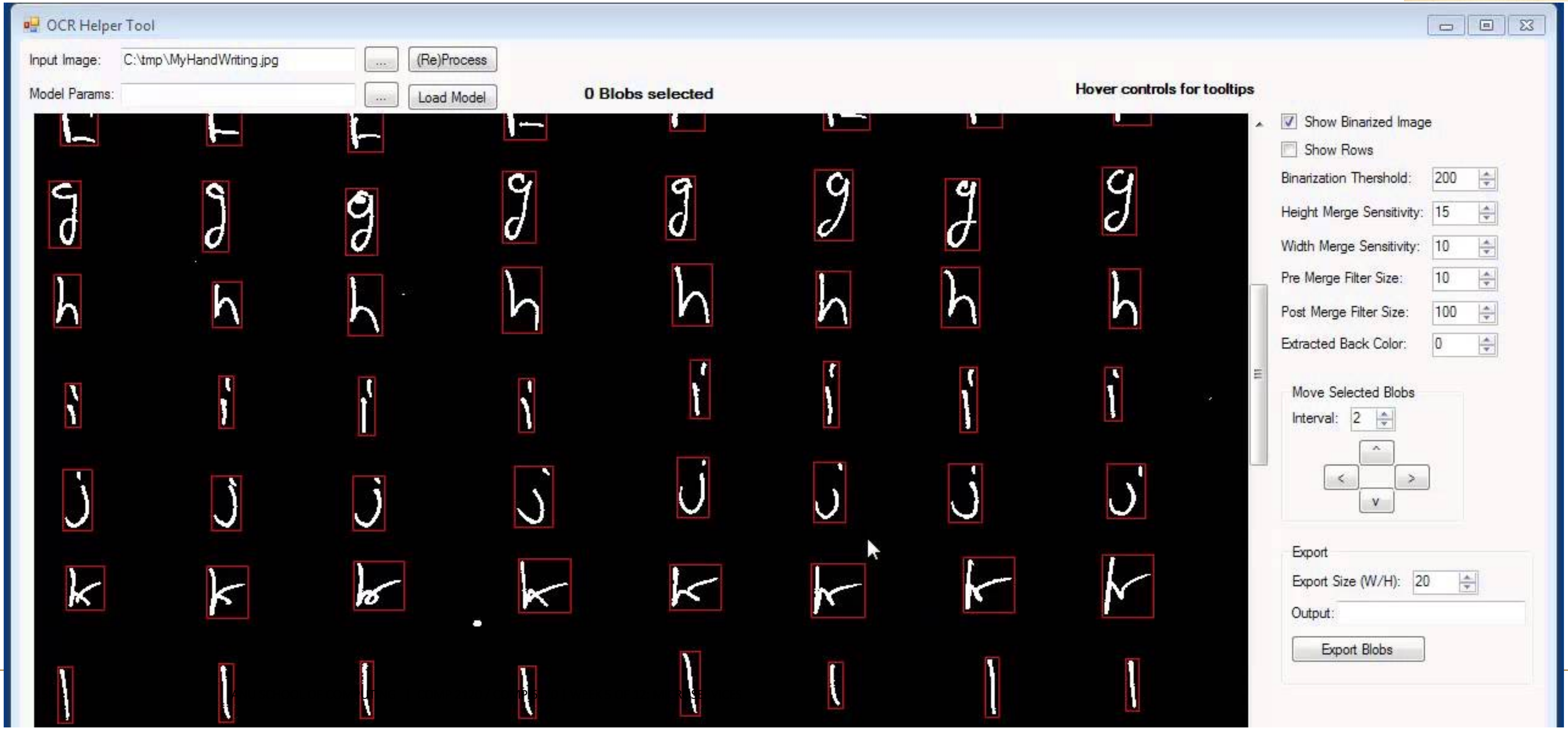
Typical ML Pipeline





- 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

Model Params:

0 Blobs selected

Hover controls for tooltips

[✓] Show Binarized Image

[ ] Show Rows

Binarization Thershold: 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

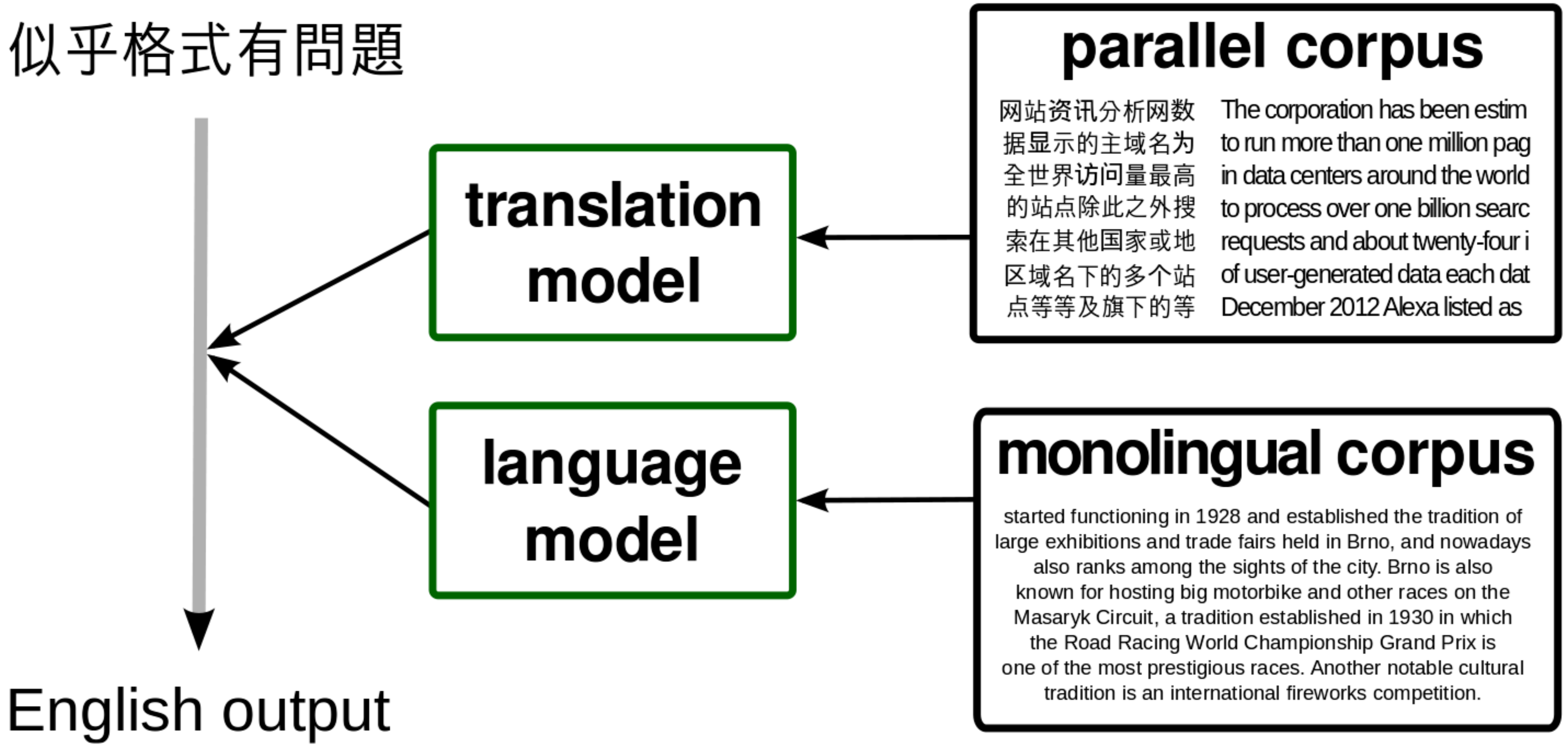
Interval: 2

Export

Export Size (W/H): 20

Output:

Learning Data


似乎格式有問題
translation model
language model
English output
parallel corpus
网站资讯分析网数据显示的主域名为全世界访问量最高的站点除此之外搜索在其他国家或地区域名下的多个站点等等及旗下的等
The corporation has been estim to run more than one million pag in data centers around the world to process over one billion searc requests and about twenty-four i of user-generated data each dat December 2012 Alexa listed as
monolingual corpus
started functioning in 1928 and established the tradition of large exhibitions and trade fairs held in Brno, and nowadays also ranks among the sights of the city. Brno is also known for hosting big motorbike and other races on the Masaryk Circuit, a tradition established in 1930 in which the Road Racing World Championship Grand Prix is one of the most prestigious races. Another notable cultural tradition is an international fireworks competition.

Feature Engineering

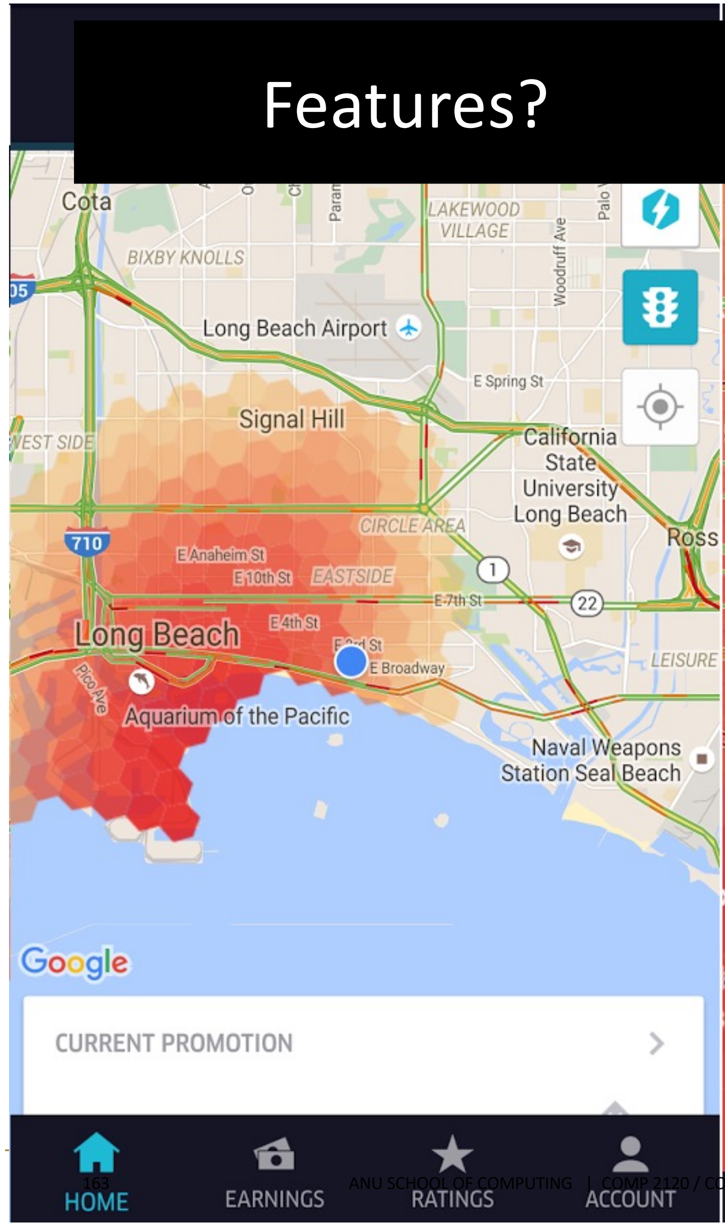
- Identify parameters of interest that a model may learn on
- Convert data into a useful form
- Normalize data
- Include context
- Remove misleading things

Features?

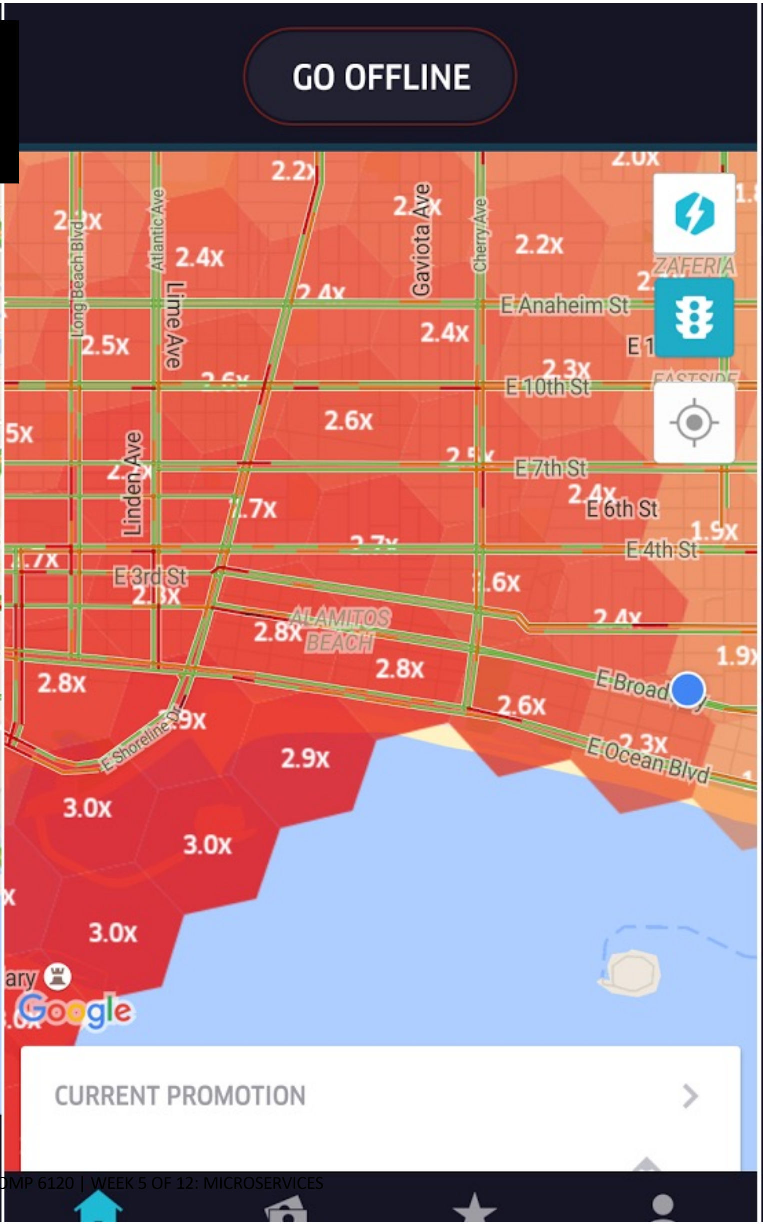
Feature Extraction

- 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)

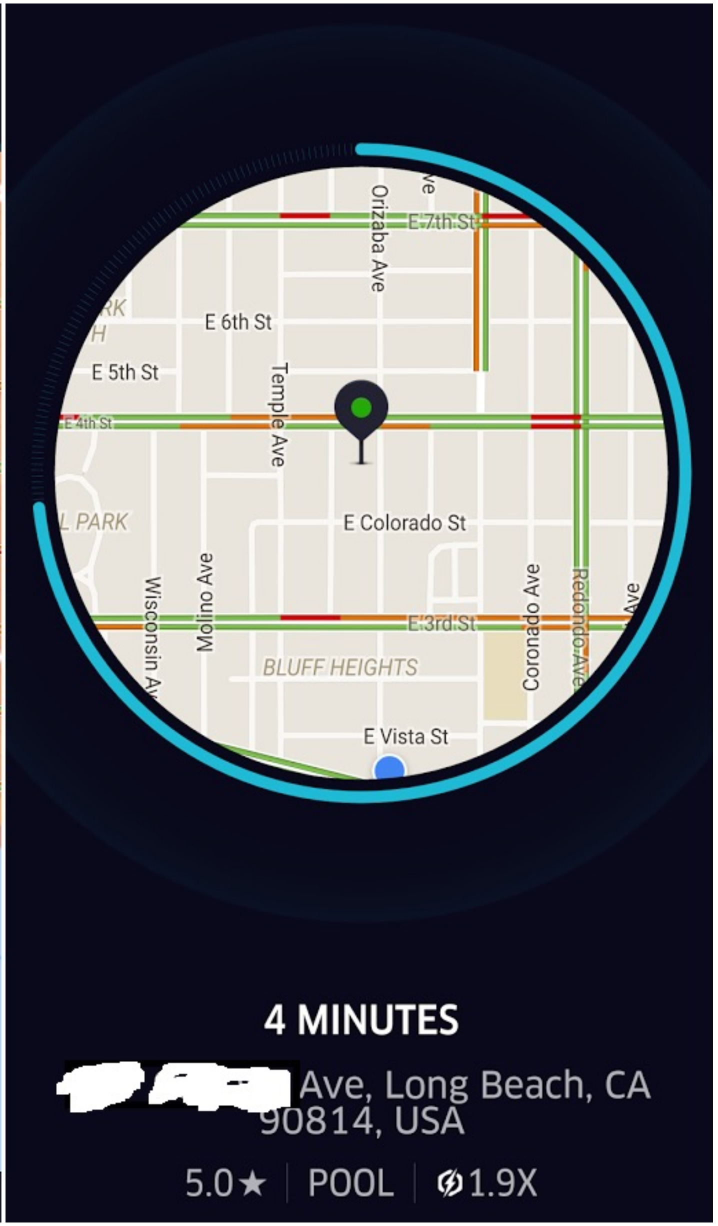
Features?



CURRENT PROMOTION



CURRENT PROMOTION



4 MINUTES

[redacted] Ave, Long Beach, CA 90814, USA

5.0★ | POOL | 1.9X

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

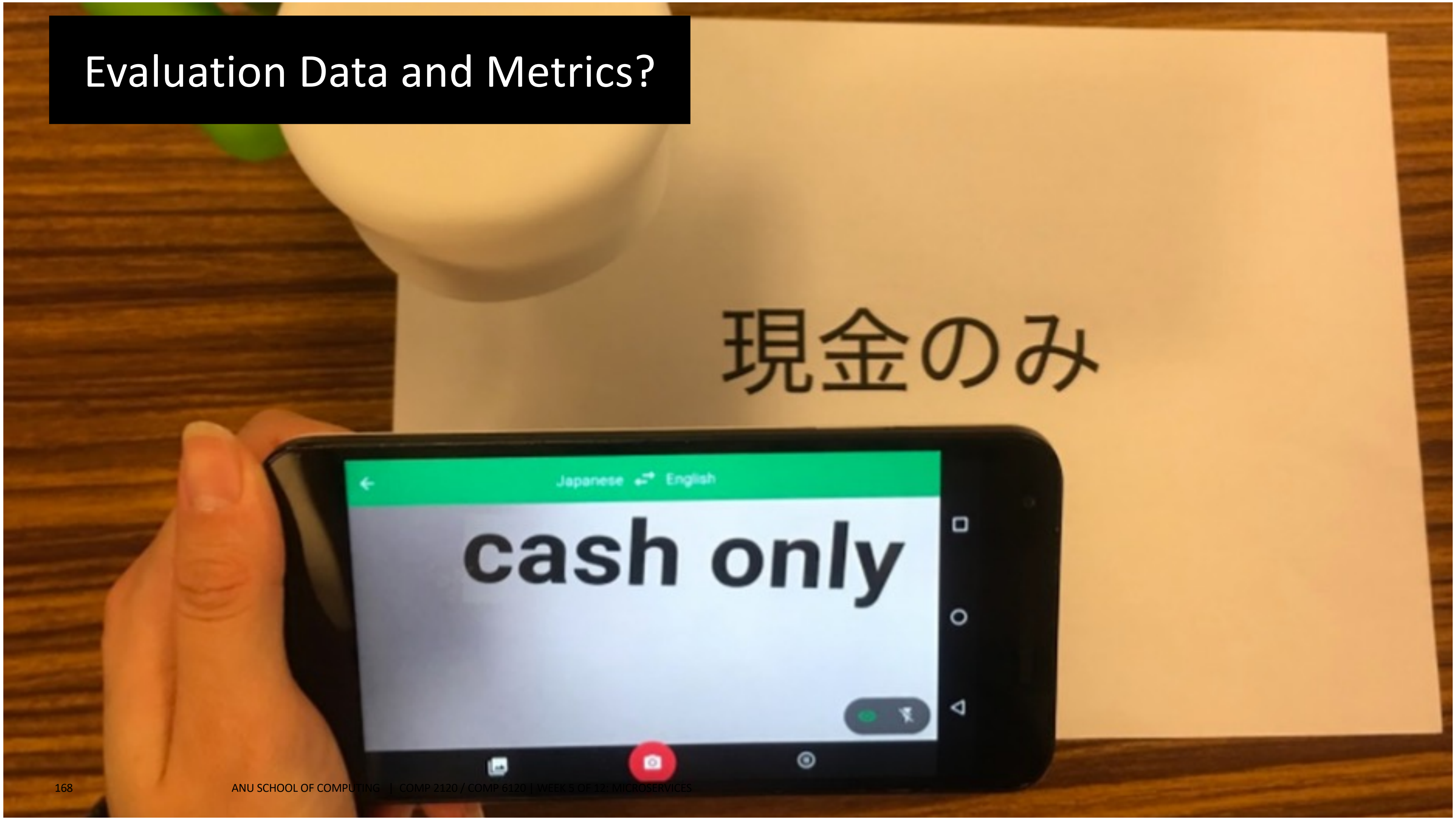
- Build a predictor that best describes an outcome for the observed features

Evaluation

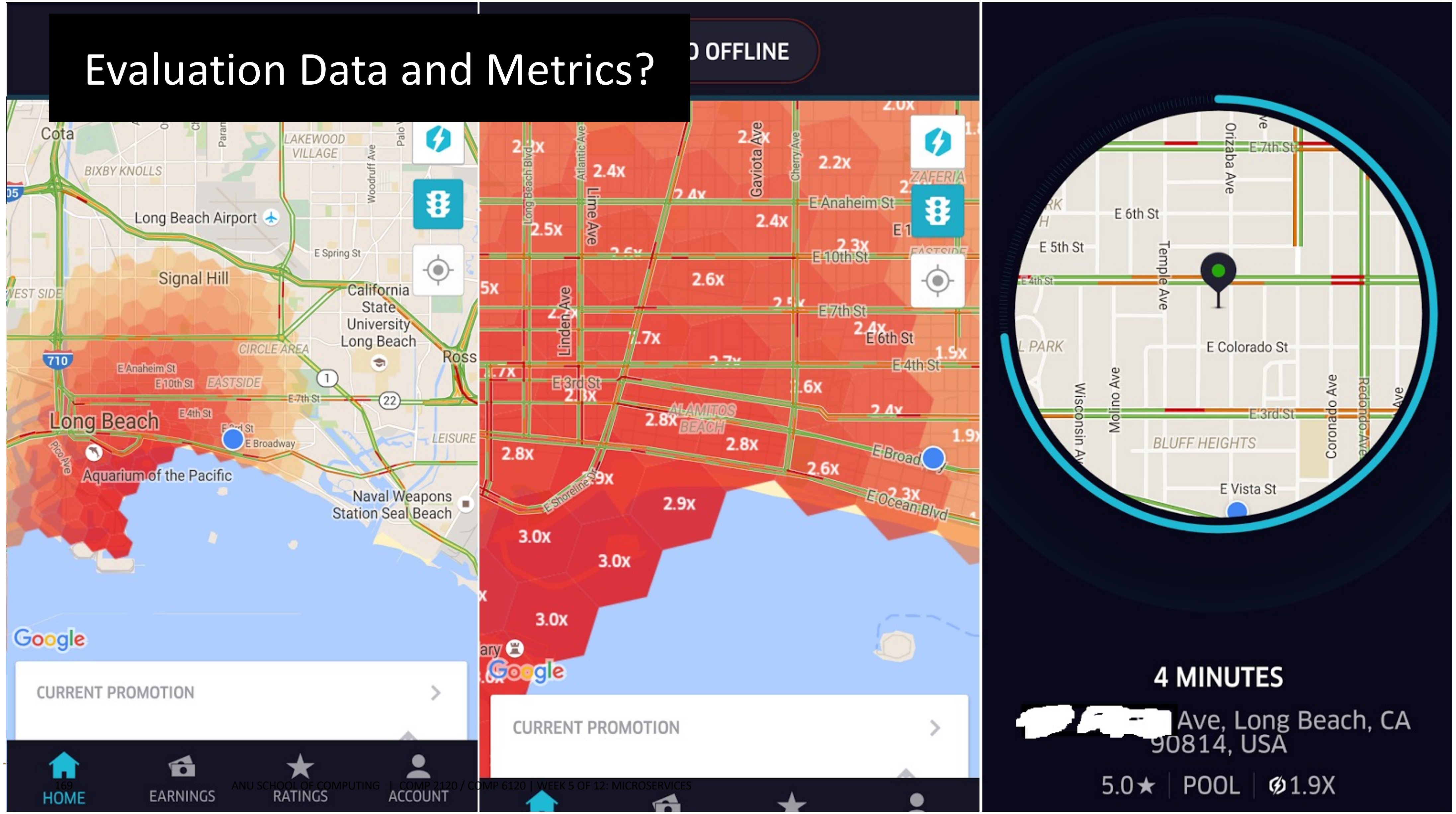
- Prediction accuracy on learned data vs
- Prediction accuracy on unseen data
  - Separate learning set, not used for training

- 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?



Evaluation Data and Metrics?



Learning and Evaluating in Production

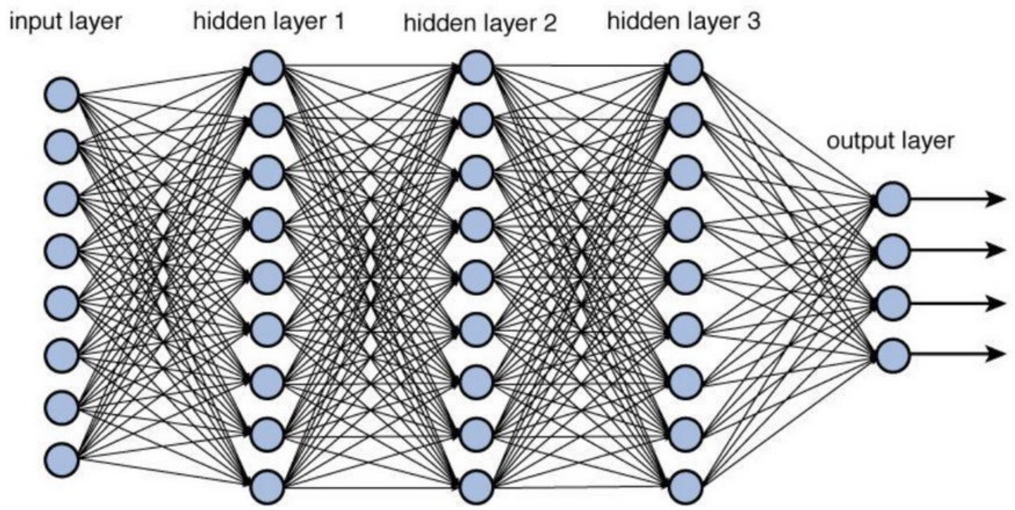
- Beyond static data sets, **build telemetry**
- Design challenge: identify mistakes in practice

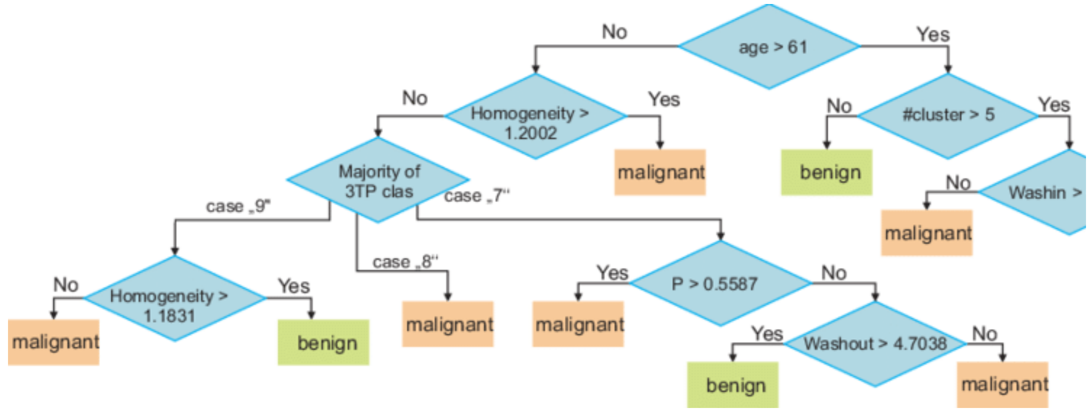
- Use sample of live data for evaluation
- Retrain models with sampled live data regularly
- Monitor performance and intervene

TRADEOFFS IN ML MODELS

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?
- …

SYSTEM ARCHITECTURE CONSIDERATIONS

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 analysists 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

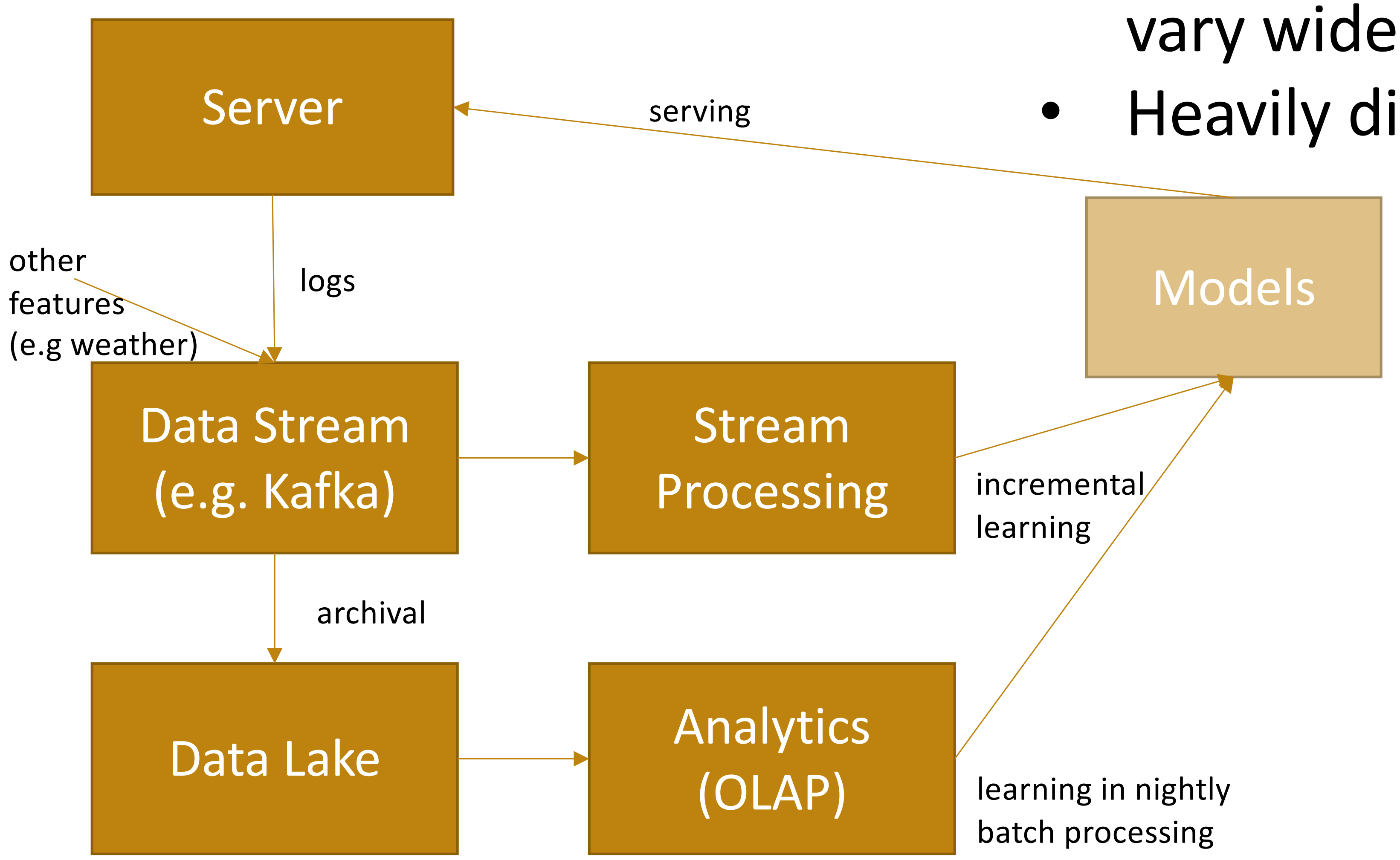
- What steps to take?
- What information to collect?

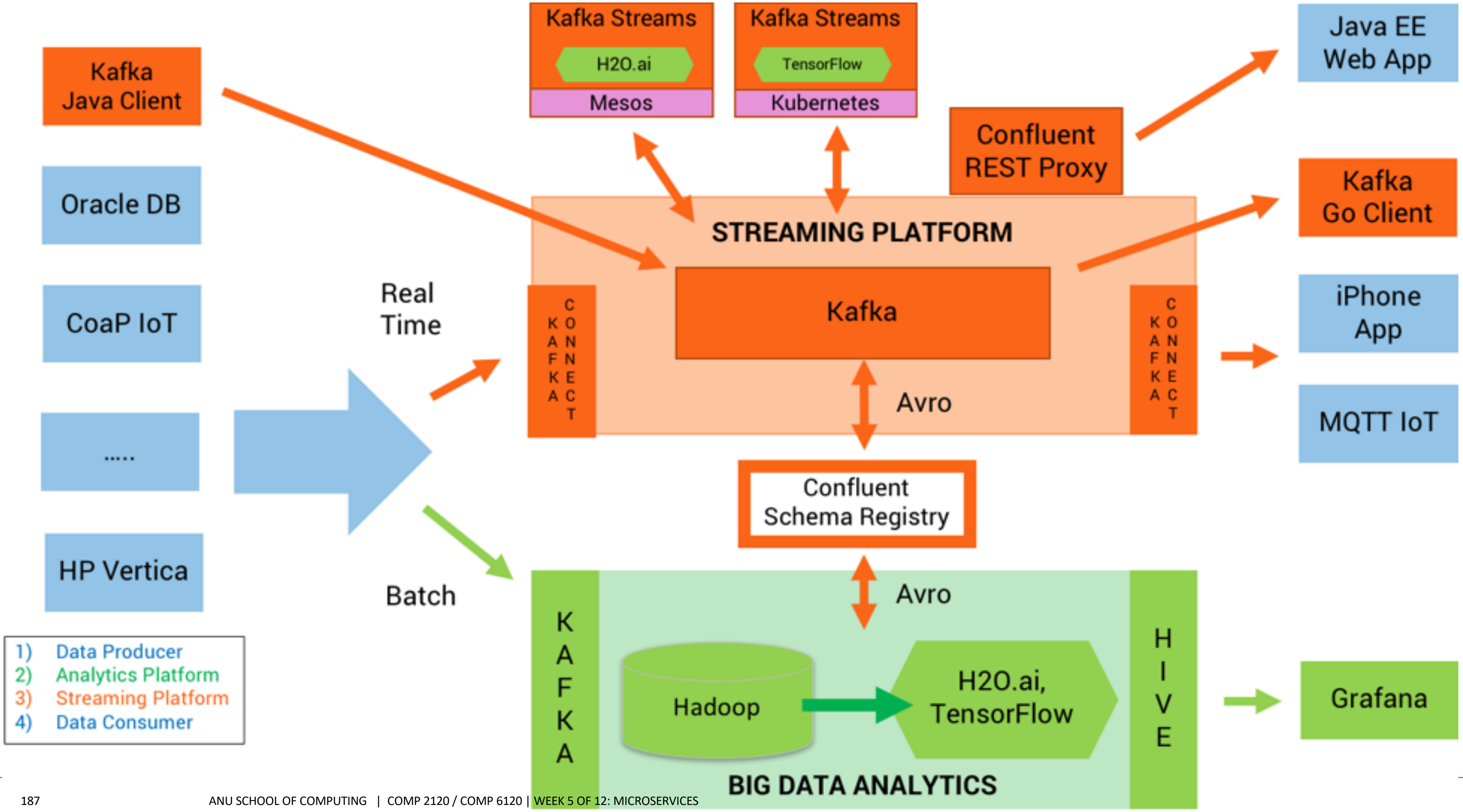
UPDATING MODELS

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 automation vary widely
- Heavily distributed

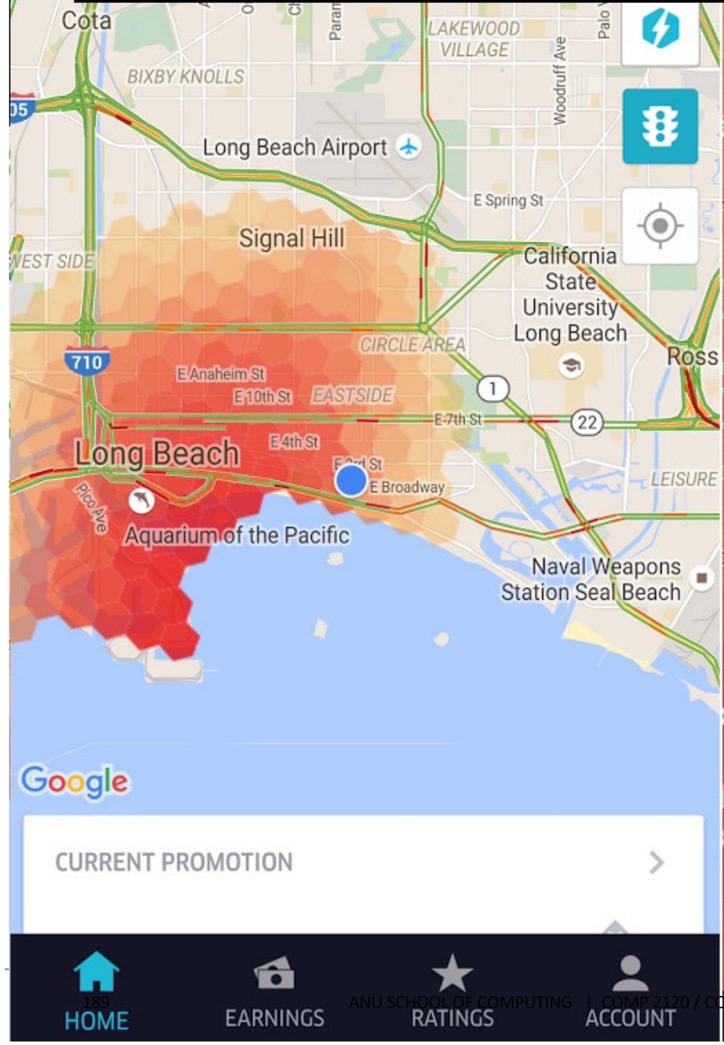


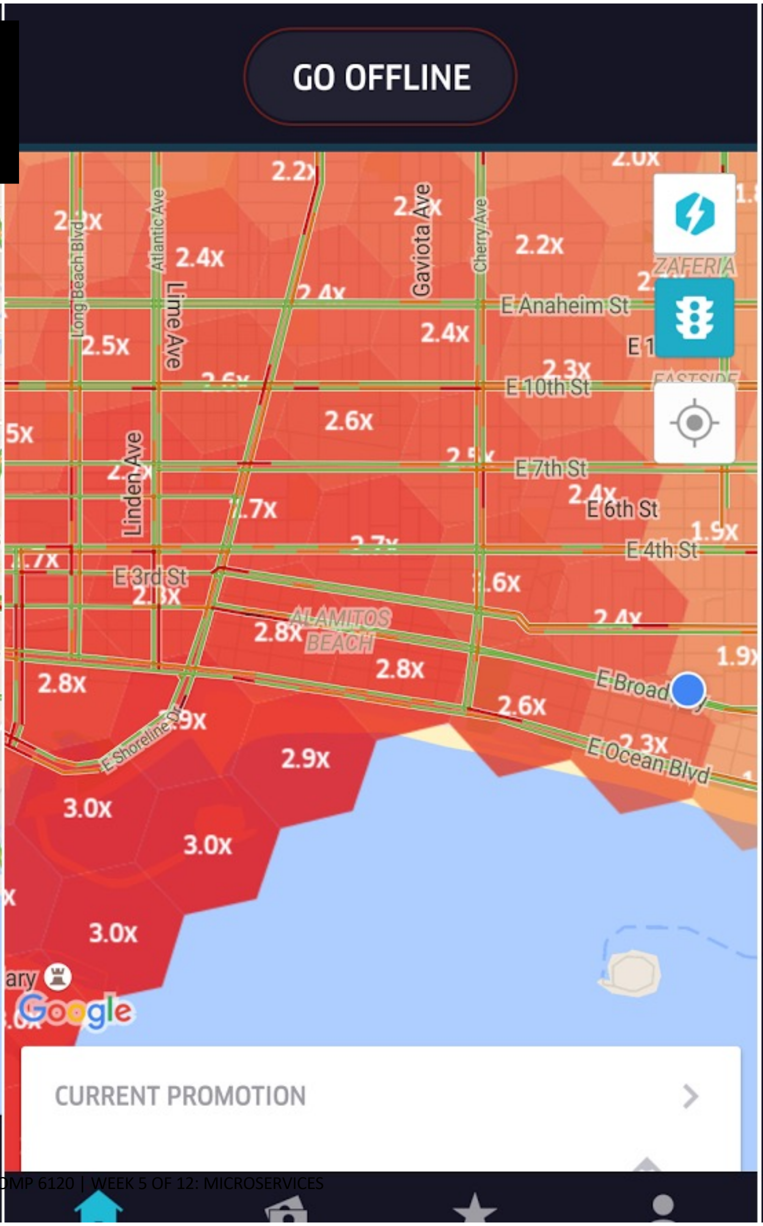


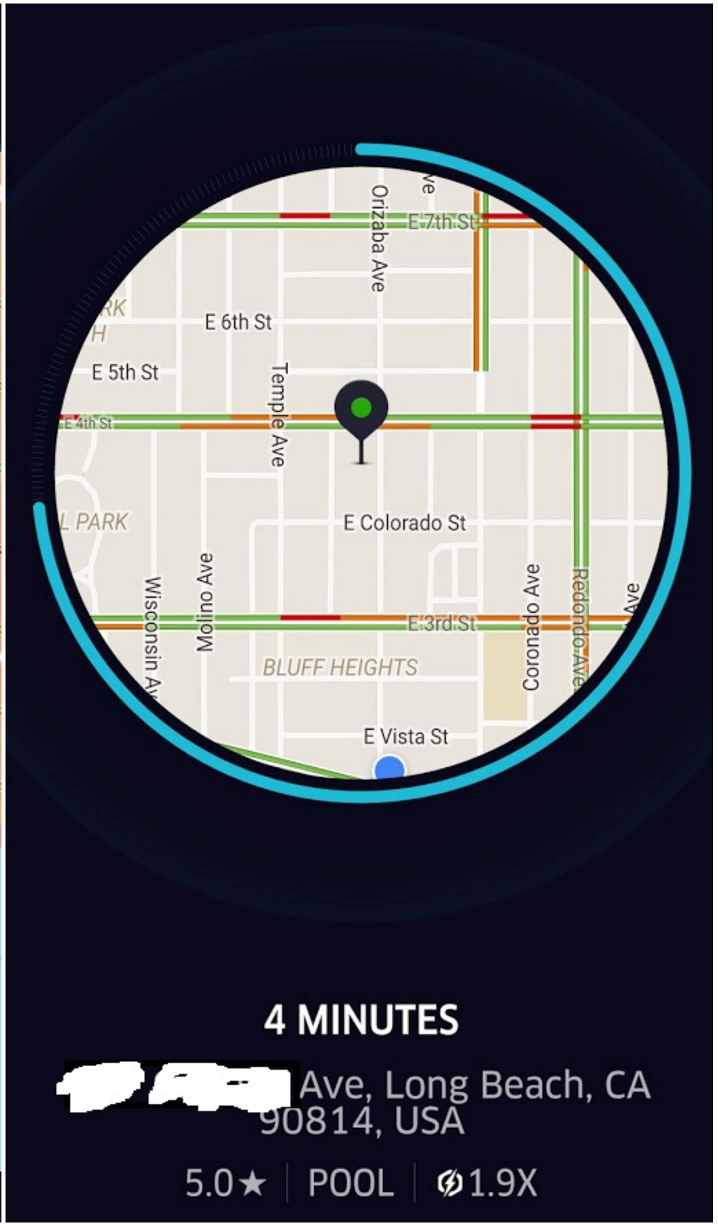
Kafka Java Client
Oracle DB
CoaP IoT
.....
HP Vertica
Real Time
Batch
Kafka Streams
H2O.ai
Mesos
Kafka Streams
TensorFlow
Kubernetes
Confluent REST Proxy
STREAMING PLATFORM
KAFKA CONNECT
Kafka
KAFKA CONNECT
Avro
Confluent Schema Registry
Avro
KAFKA
Hadoop
H2O.ai, TensorFlow
HIVE
BIG DATA ANALYTICS
Java EE Web App
Kafka Go Client
iPhone App
MQTT IoT
Grafana
1) Data Producer
2) Analytics Platform
3) Streaming Platform
4) Data Consumer

Update Strategy?

Update Strategy?

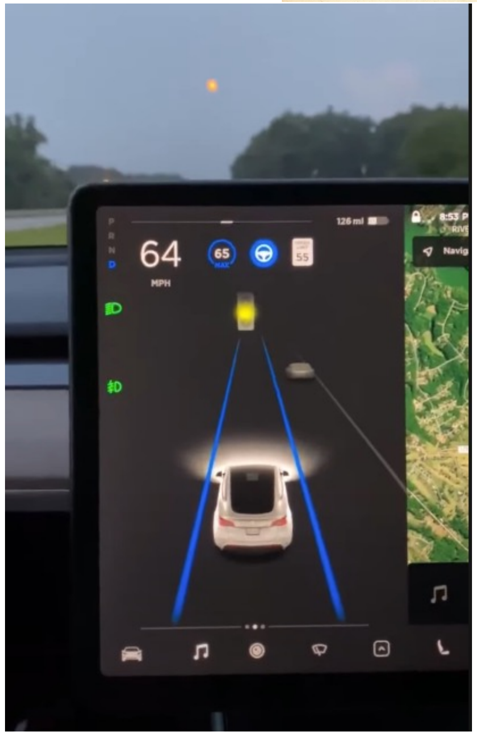






PLANNING FOR MISTAKES

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

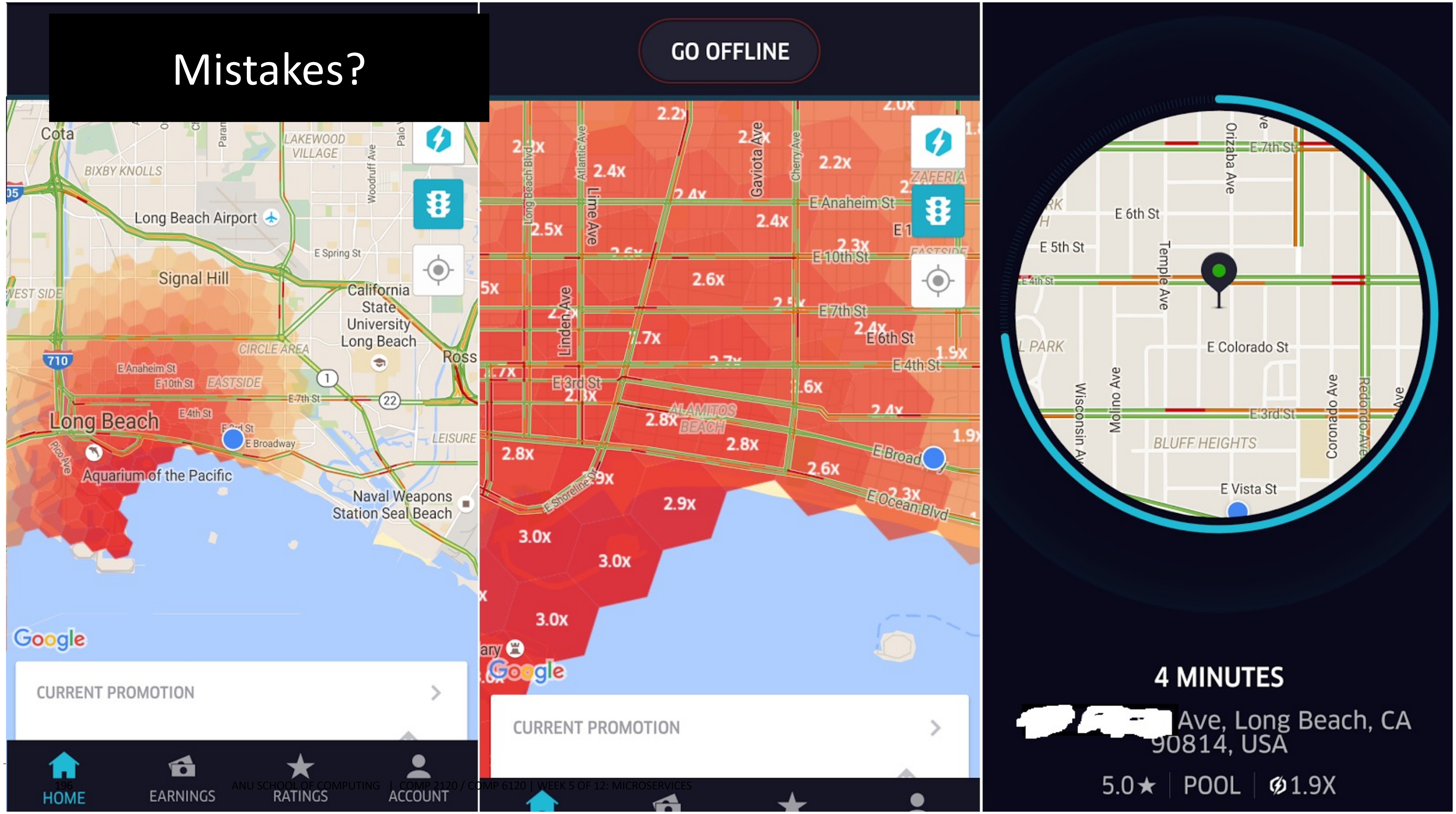
- Worst thing that can happen?
- Backup strategy? Undoable? Nontechnical compensation?

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?

Mistakes?



4 MINUTES

[redacted] Ave, Long Beach, CA 90814, USA

5.0★ | POOL | 1.9X

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

Key Points

- Identify differences between traditional software development and development of ML systems.
- Understand the stages that comprise the typical ML development pipeline.
- Identify challenges that must be faced within each stage of the typical ML development pipeline.

Key Points

- Machine learning in production systems is challenging
- Many tradeoffs in selecting ML components and in integrating them in larger system
- Plan for updates
- Manage mistakes, plan for telemetry

Quotes from this wonderful paper suggested by Mary Shaw:

- https://web.engr.oregonstate.edu/~burnett/Reprints/vlhcc16-ML-trialsTribulations.pdf