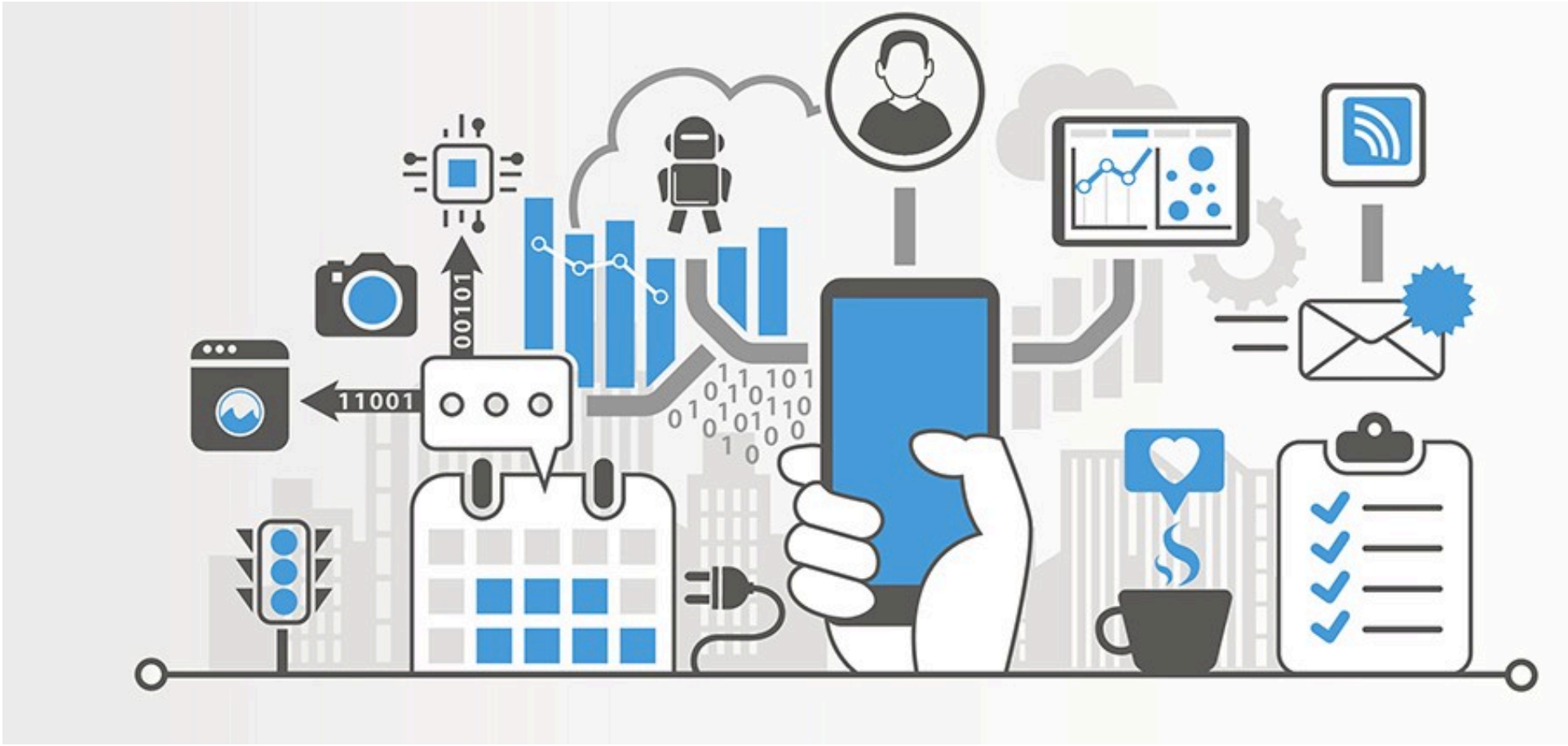
Programming for IoT Applications

Edoardo Patti

Lecture 10



SOFTWARE ARCHITECTURE
DESIGN PATTERNS

Software system

From Wikipedia, the free encyclopedia

A **software system** is a system of intercommunicating components based on software forming part of a computer system (a combination of hardware and software). It "consists of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system".[1]

Software system

From Wikipedia, the free encyclopedia

A **software system** is a system of intercommunicating components based on software forming part of a computer system (a combination of hardware and software). It "consists of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system".[1]

The term "software system" should be distinguished from the terms "computer program" and "software". The term computer program generally refers to a set of instructions (source, or object code) that perform a specific task. However, a software system generally refers to a more encompassing concept with many more components such as specification, test results, end-user documentation, maintenance records, etc.[2]

Monolithic system

From Wikipedia, the free encyclopedia

A software system is called "monolithic" if it has a **monolithic architecture**, in which functionally distinguishable aspects (for example data input and output, data processing, error handling, and the user interface) are all interwoven, rather than containing architecturally separate components.[1]

Monolithic application

From Wikipedia, the free encyclopedia

In software engineering, a **monolithic application** describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.

Monolithic application

From Wikipedia, the free encyclopedia

In software engineering, a **monolithic application** describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.

A monolithic application is self-contained, and independent from other computing applications. The design philosophy is that the application is responsible not just for a particular task, but can perform every step needed to complete a particular function.[1][2]

Monolithic application (ISSUES)

From Wikipedia, the free encyclopedia

In software engineering, a monolithic application describes a software application which is designed without modularity. Modularity is desirable, in general, as it supports reuse of parts of the application logic and also facilitates maintenance by allowing repair or replacement of parts of the application without requiring wholesale replacement.

Monolithic application (ISSUES)

From Wikipedia, the free encyclopedia

In software engineering, a monolithic application describes a software application which is designed without modularity. Modularity is desirable, in general, as it supports reuse of parts of the application logic and also facilitates maintenance by allowing repair or replacement of parts of the application without requiring wholesale replacement.

In its original use, the term "monolithic" described enormous main frame applications with no usable modularity. This – in combination with rapid increase in computational power and therefore rapid increase in the complexity of the problems which could be tackled by software – resulted in unmaintainable systems and the "software crisis".

Event-driven architecture

From Wikipedia, the free encyclopedia

Event-driven architecture (**EDA**) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

Event-driven architecture

From Wikipedia, the free encyclopedia

Event-driven architecture (**EDA**) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

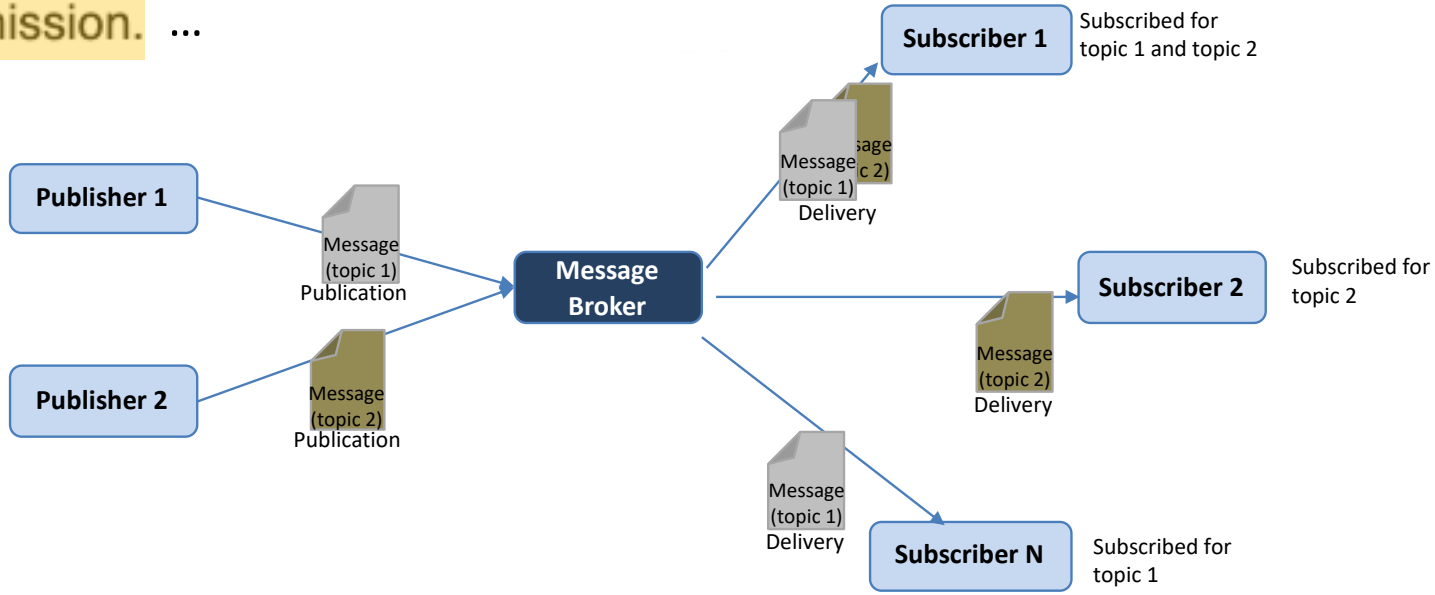
An *event* can be defined as "a significant change in state".[1] For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". A car dealer's system architecture may treat this state change as an event whose occurrence can be made known to other applications within the architecture. From a formal perspective, what is produced, published, propagated, detected or consumed is a (typically asynchronous) message called the event notification, and not the event itself, which is the state change that triggered the message emission. ...

Event-driven architecture

From Wikipedia, the free encyclopedia

Event-driven architecture (**EDA**) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

An *event* can be defined as "a significant change in state".[1] For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". A car dealer's system architecture may treat this state change as an event whose occurrence can be made known to other applications within the architecture. From a formal perspective, what is produced, published, propagated, detected or consumed is a (typically asynchronous) message called the event notification, and not the event itself, which is the state change that triggered the message emission. ...



Event-driven architecture

From Wikipedia, the free encyclopedia

Event-driven architecture (**EDA**) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

An *event* can be defined as "a significant change in state".[1] For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". A car dealer's system architecture may treat this state change as an event whose occurrence can be made known to other applications within the architecture. From a formal perspective, what is produced, published, propagated, detected or consumed is a (typically asynchronous) message called the event notification, and not the event itself, which is the state change that triggered the message emission. ...

... This is due to Event-Driven architectures often being designed atop **message-driven architectures**, where such communication pattern requires one of the inputs to be text-only, the message, to differentiate how each communication should be handled.

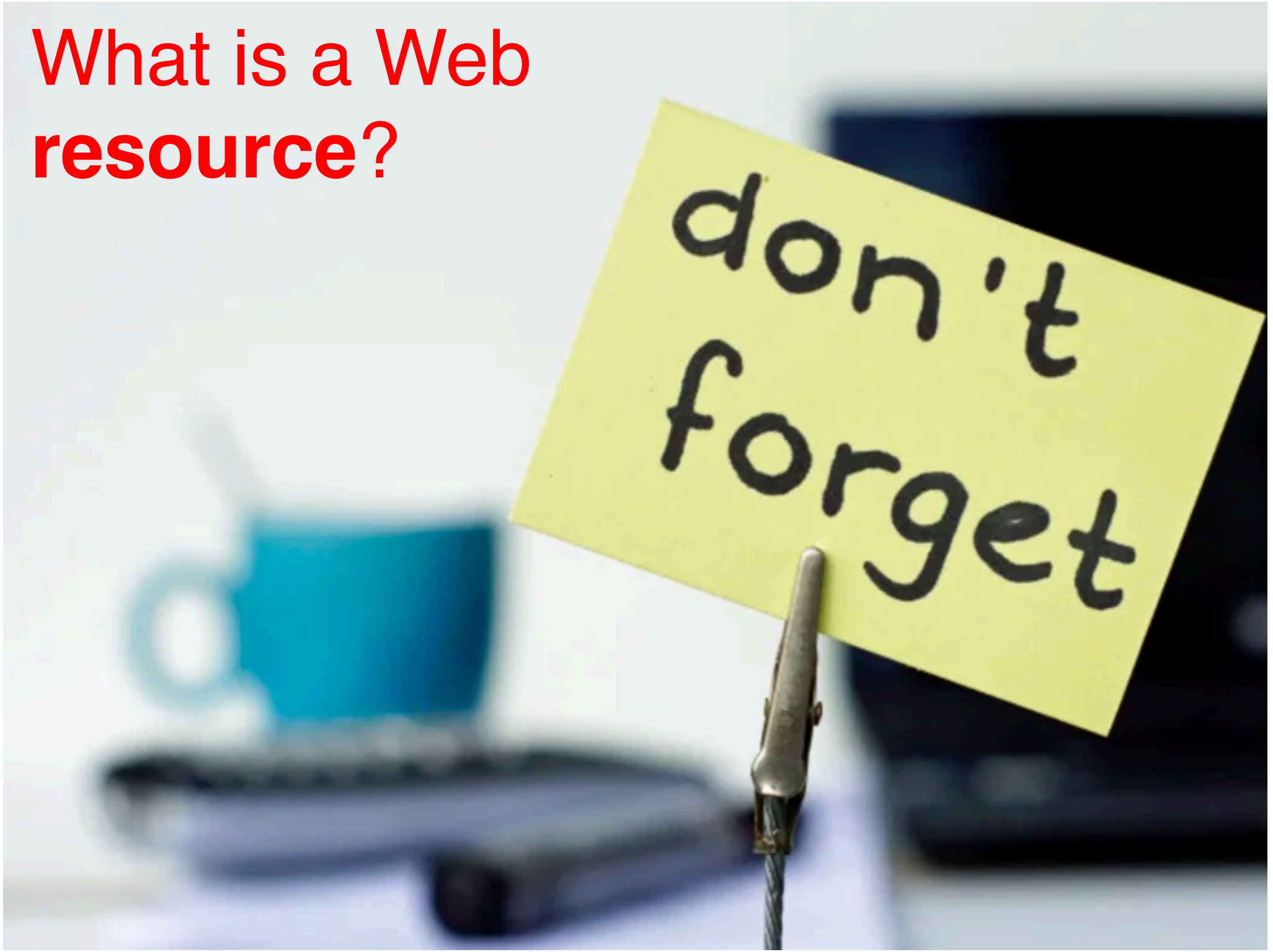
Resource-oriented architecture

From Wikipedia, the free encyclopedia

In software engineering, a **resource-oriented architecture** (**ROA**) is a style of software architecture and programming paradigm for supportive designing and developing software in the form of Internetworking of resources with "RESTful" interfaces. These resources are software components (discrete pieces of code and/or data structures) which can be reused for different purposes. ROA design principles and guidelines are used during the phases of software development and system integration.

REST, or Representational State Transfer, describes a series of architectural constraints that exemplify how the web's design emerged.

What is a Web **resource**?



What is a Web **resource**? (1)

- A **resource** is any information that can be named, such as, a document, an image, etc..
- **A resource has a name and an address represented by a URI**
- A resource is something that can be **stored on a computer and represented as a stream of bits** (e.g. a document, a row in a database, or the result of running an algorithm)

What is a Web **resource**? (2)

- A **representation** is defined as **a sequence of bytes, plus representation meta-data to describe those bytes**.
  - The client receives a representation when it requests a resource or sends one when it wishes to update a resource.

What is a Web **resource**? (3)

- What makes a resource a resource? **It has to have at least one URI.**
  - The **URI is the name and address of a resource**.
  - **URIs uniquely identify a resource**, regardless of its type and representation.
  - **If a piece of information does not have a URI, it is not a resource** and it is not really on the Web

Service-oriented architecture

From Wikipedia, the free encyclopedia

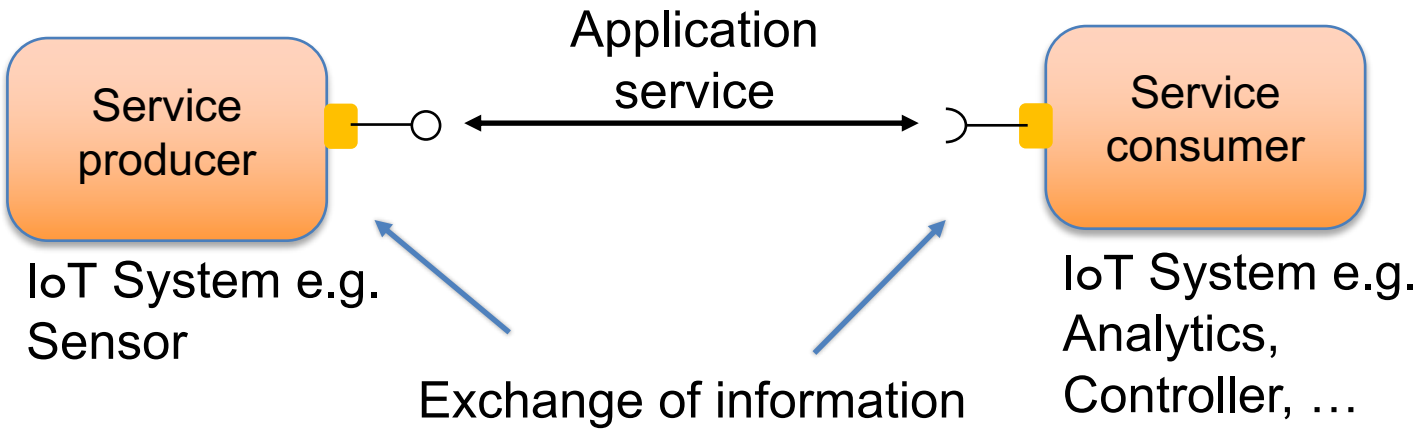
Service-oriented architecture (**SOA**) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies.[1] A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.

What is a **Service**? (1)

The Organization for the Advancement of Structured Information Standards (OASIS) defines a service as *"a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description"*.

What is a **Service**? (2)

- The term **Service** refers to a software functionality or a set of **software functionalities** with a purpose **that different clients can reuse for different purposes**.
- A **Service** can exchange information with other Services through communication interfaces over the Internet



Service-oriented architecture

From Wikipedia, the free encyclopedia

Service-oriented architecture (**SOA**) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies.[1] A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.

A service has four properties according to one of many definitions of SOA:[2]

1. It logically represents a business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers.
4. It may consist of other underlying services.[3]

Different services can be used in conjunction to provide the functionality of a large software application,[4] a principle SOA shares with modular programming. Service-oriented architecture integrates distributed, separately maintained and deployed software components. It is enabled by technologies and standards that facilitate components' communication and cooperation over a network, especially over an IP network.

What are the differences?

Monolithic

What are the differences?

Monolithic

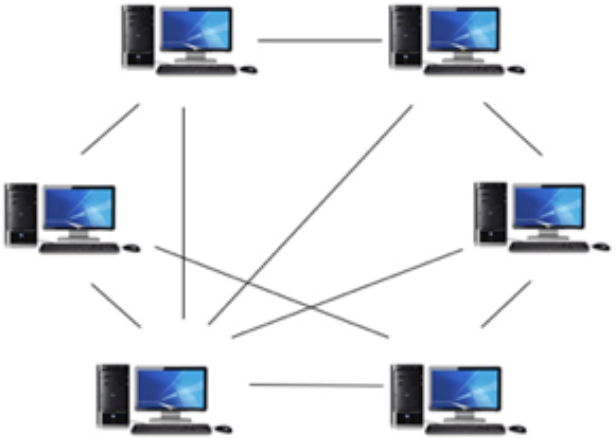
What are the differences?

Monolithic

Service Oriented

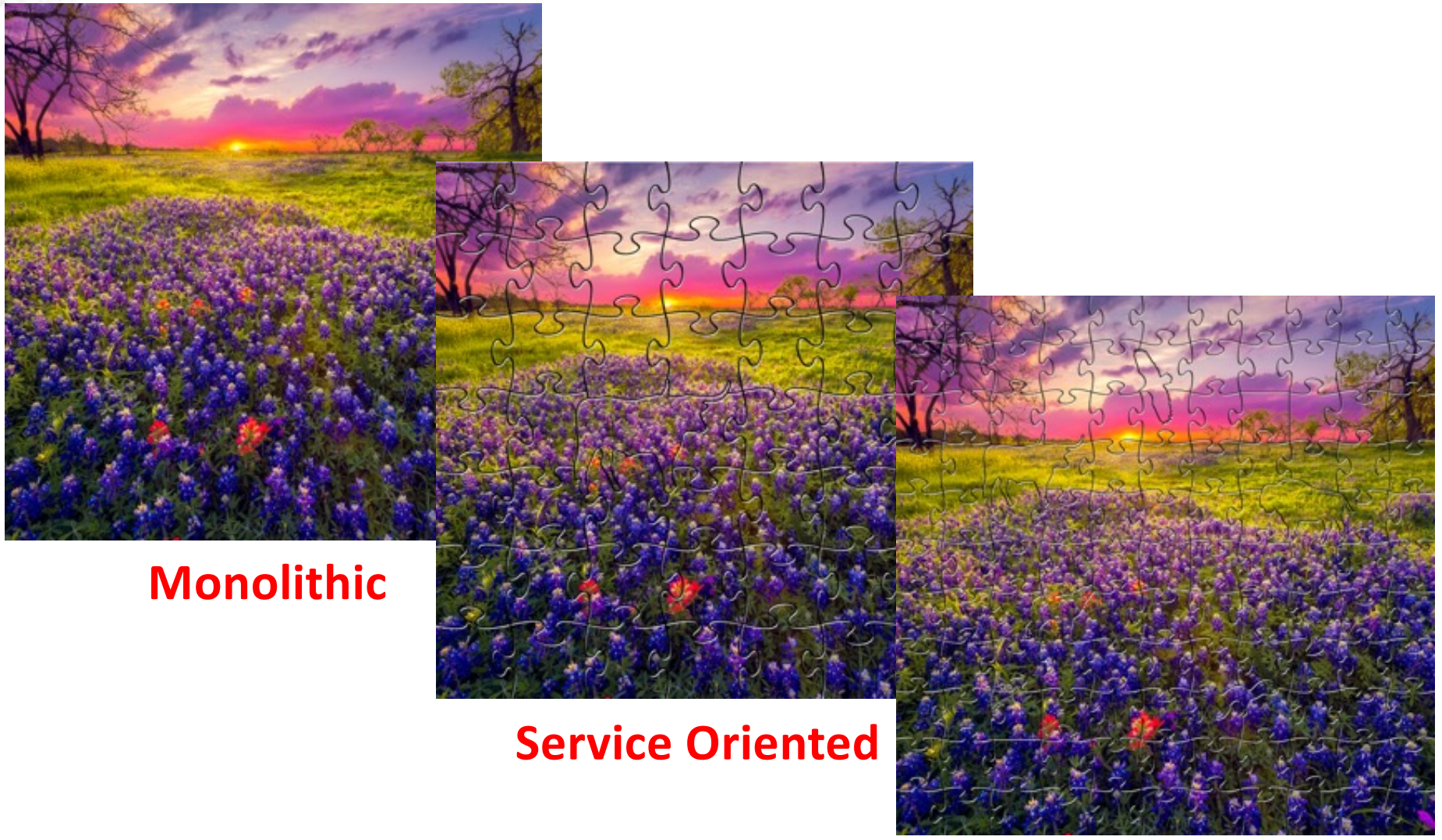
What are the differences?

Monolithic



Service Oriented

What are the differences?



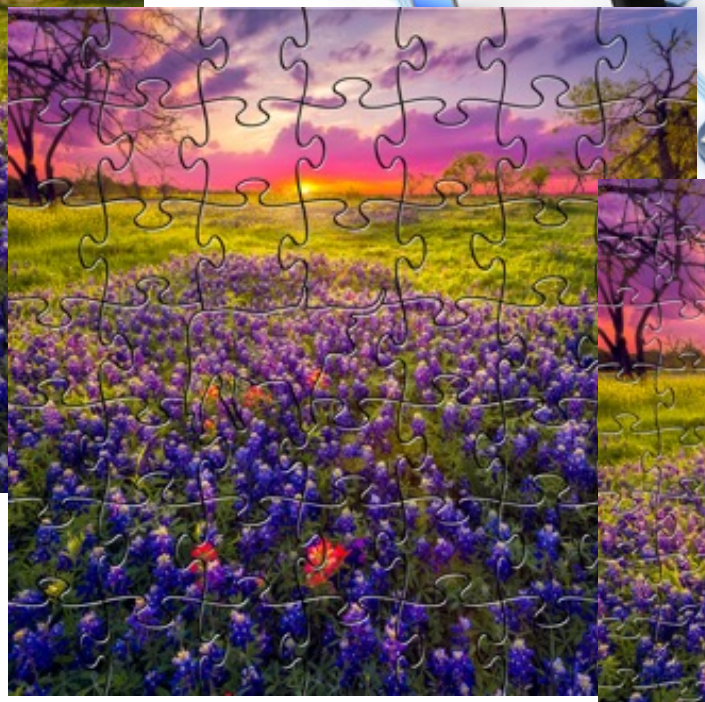
Monolithic

Service Oriented

Microservices

What are the differences?

Monolithic



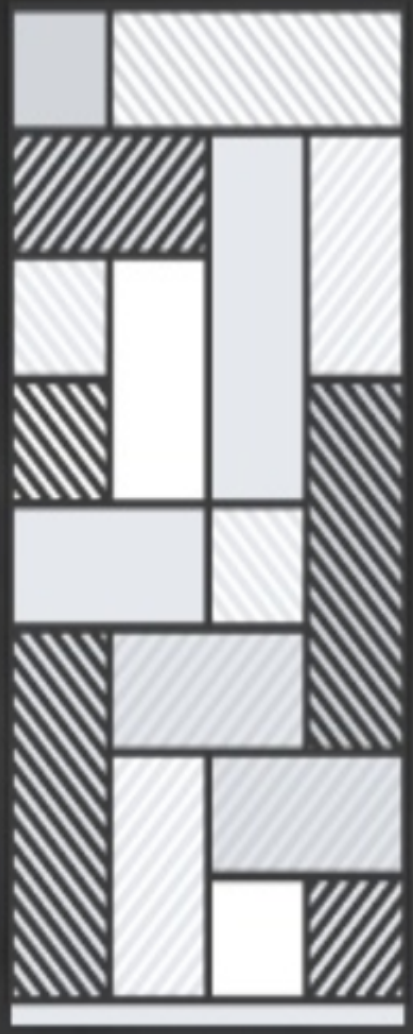
Service Oriented

Microservices

MICROSERVICES DESIGN PATTERN

“The Monolith”

Clip slide



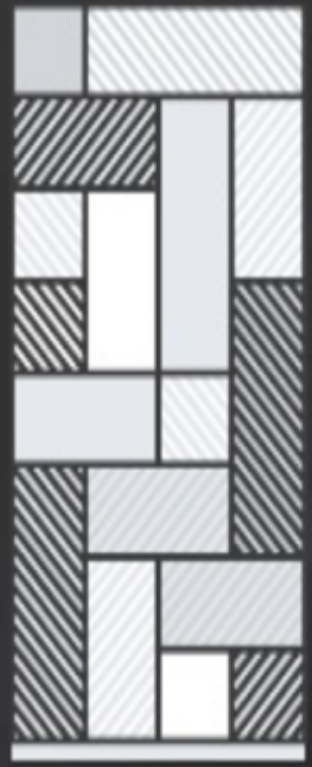
Monolith development lifecycle

Clip slide

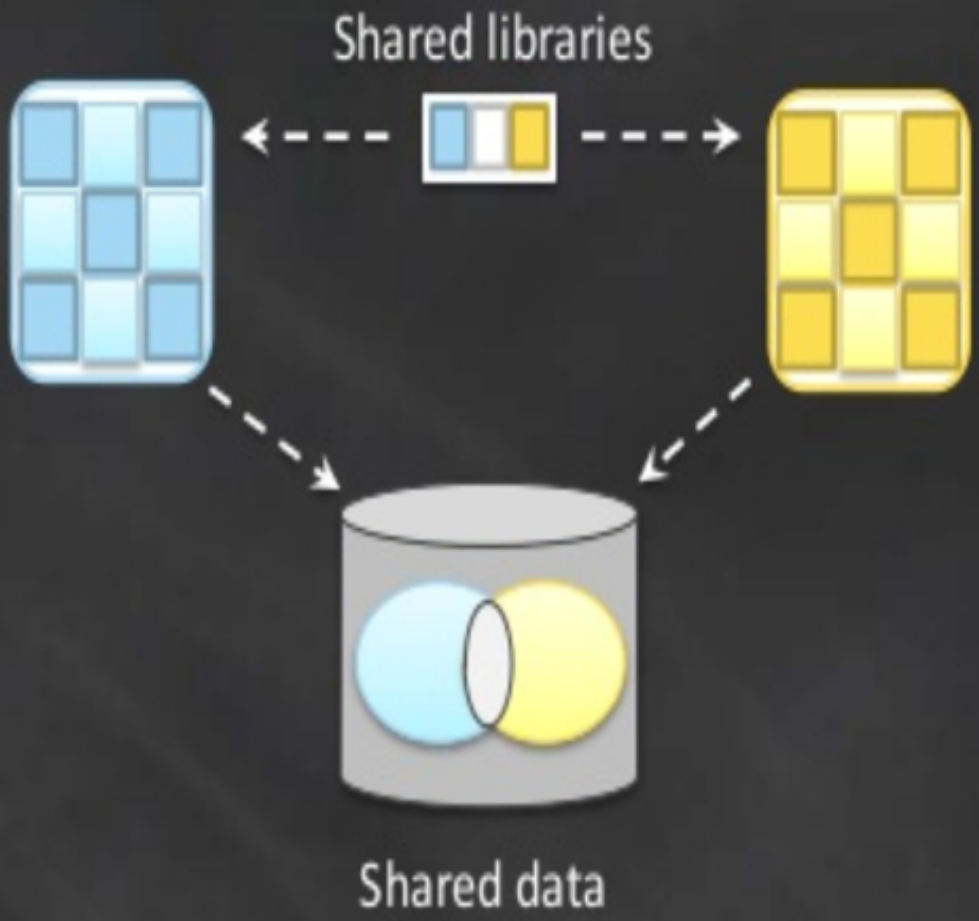
developers

delivery pipeline

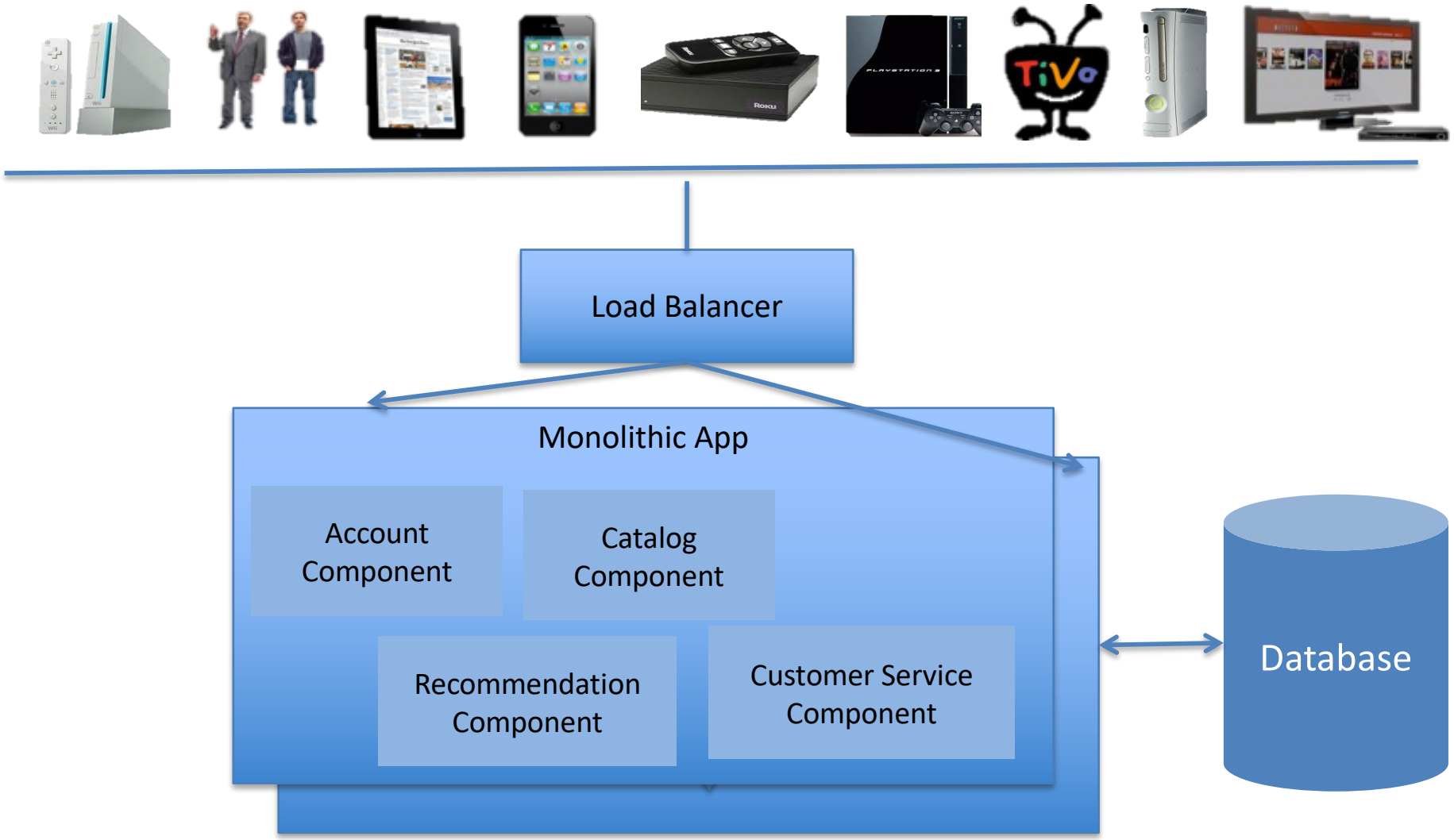
app
(aka the“monolith”)



Too much software coupling



Monolithic Architecture



Characteristics

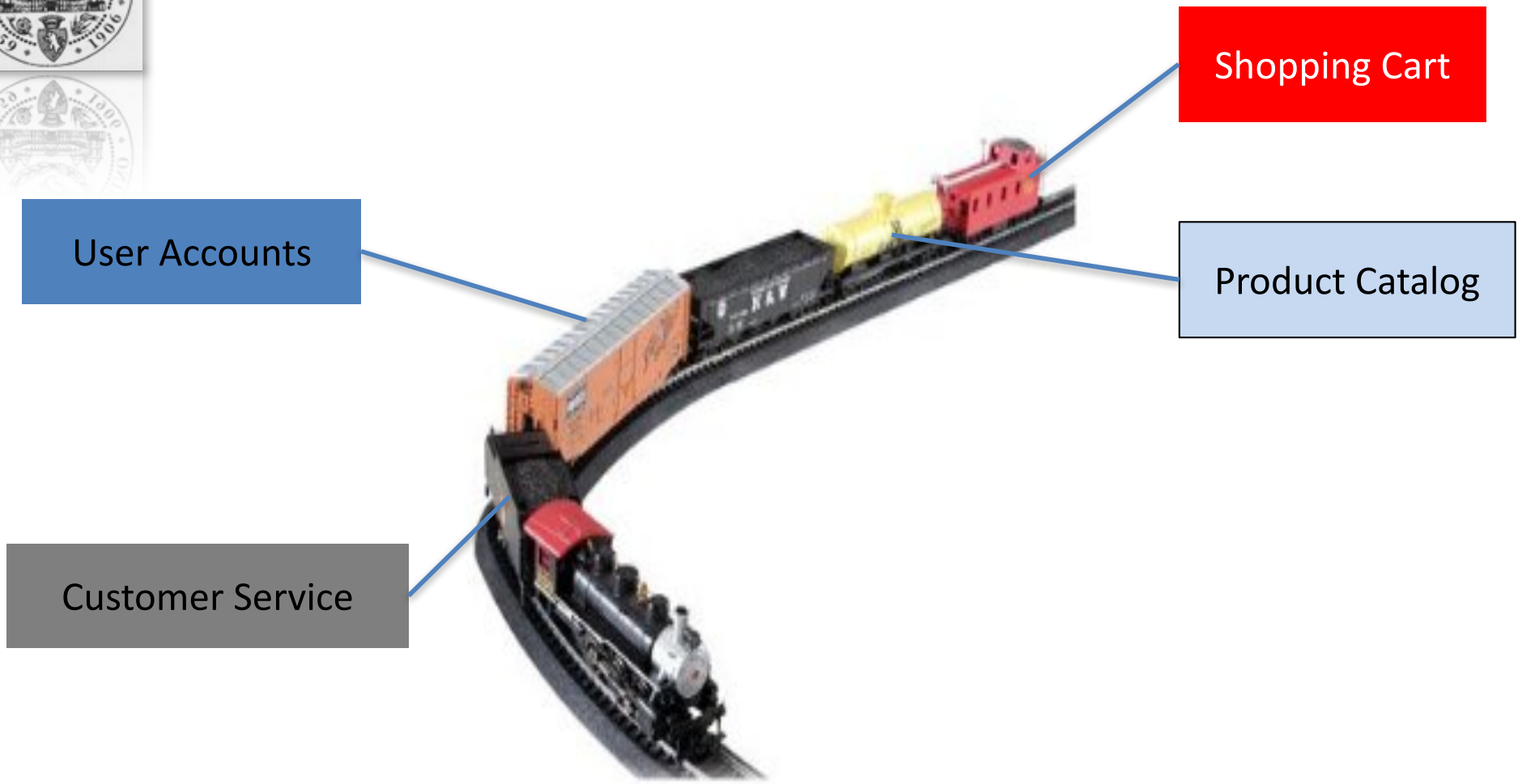
- Large Codebase
- Many Components, no clear ownership
- Long deployment cycles

Pros

- Single codebase
  - Easy to develop/debug/deploy
  - Good IDE support
- A Central Ops team can efficiently handle

Monolithic App – Evolution

- As codebase increases …
  - Tends to increase “tight coupling” between components
    - Just like the cars of a train
  - All components have to be coded in the same language



Evolution of a Monolithic App

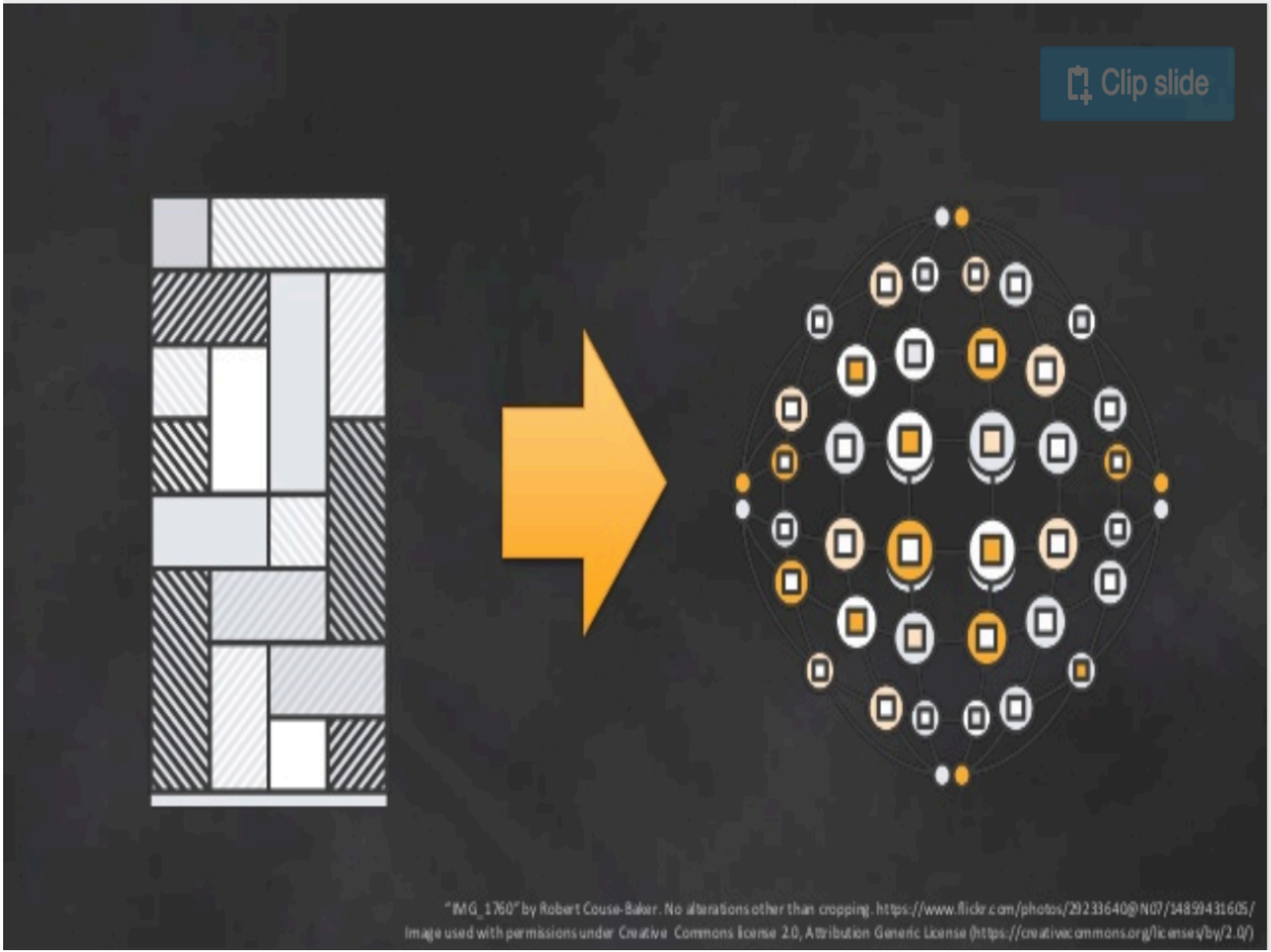
Monolithic App - Scaling

- Scaling is “undifferentiated”
  - Can't scale “Product Catalog” differently from “Customer Service”

Challenges with monolithic software

Difficult to scale	Architecture is hard to maintain and evolve	Lack of agility
Long Build/Test/Release Cycles (who broke the build?)	New releases take months	Lack of innovation
Operations is a nightmare (module X is failing, who's the owner?)	Long time to add new features	Frustrated customers

Monolithic Apps – Failure & Availability



What is the Microservices approach?

Microservices is an emerging software designing pattern that can be defined as *an approach to develop a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms. These services are small, highly decoupled and focus on doing a small task*.

It increases **flexibility** and **maintainability**.

What is the Microservices approach?

Microservices

From Wikipedia, the free encyclopedia
(Redirected from Microservice)

Microservices is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services should be fine-grained and the protocols should be lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test. It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.[1] It also allows the architecture of an individual service to emerge through continuous refactoring.[2] Microservices-based architectures enable continuous delivery and deployment.[3]

"service-oriented
architecture
composed of
loosely coupled
elements
that have
bounded contexts"

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

"service-oriented architecture

composed of

loosely coupled elements

that have

bounded contexts"

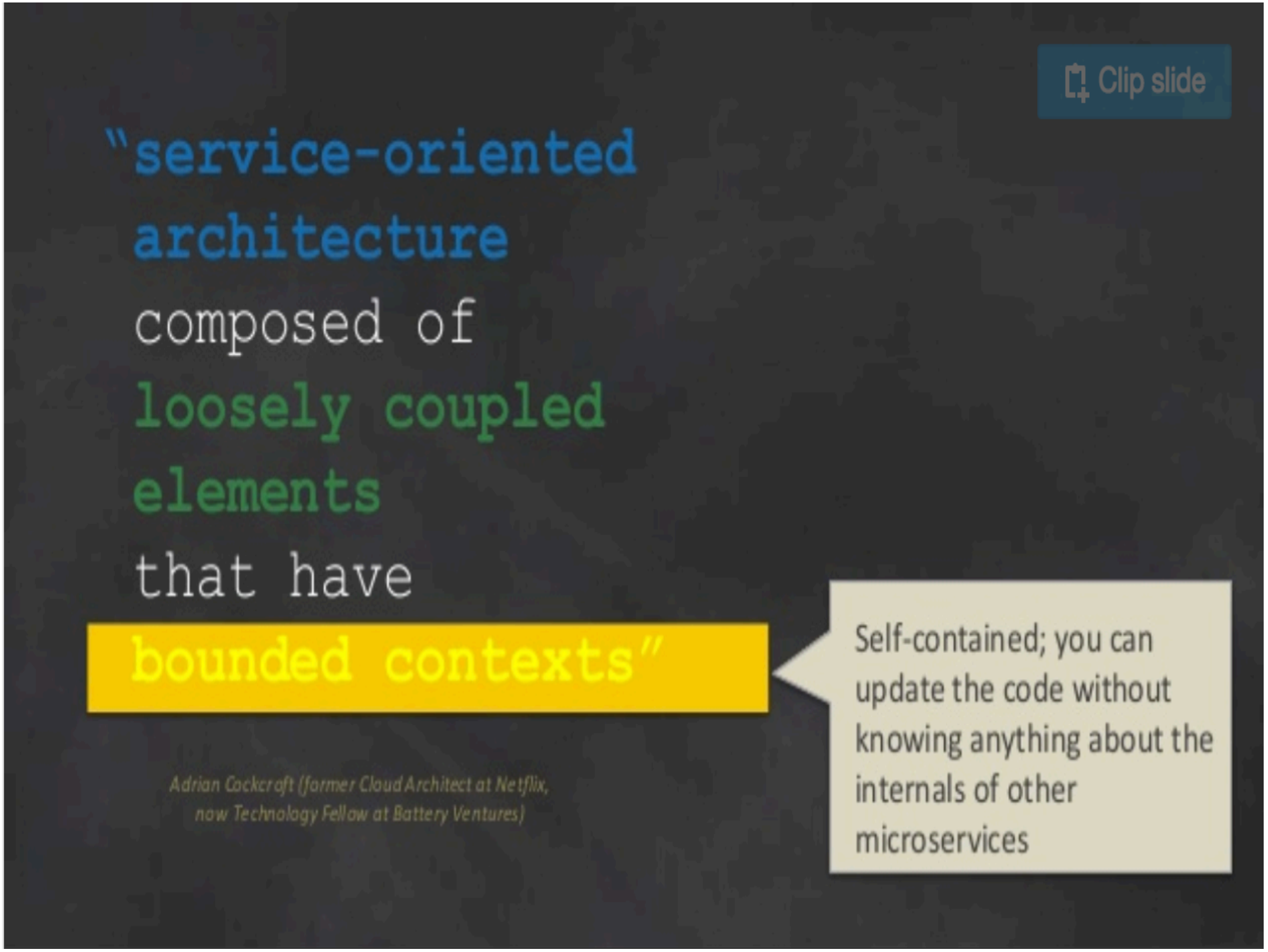
Services communicate with each other over the network

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

"service-oriented
architecture
composed of
loosely coupled
elements
that have
bounded contexts"

You can update the services independently; updating one service doesn't require changing any other services.

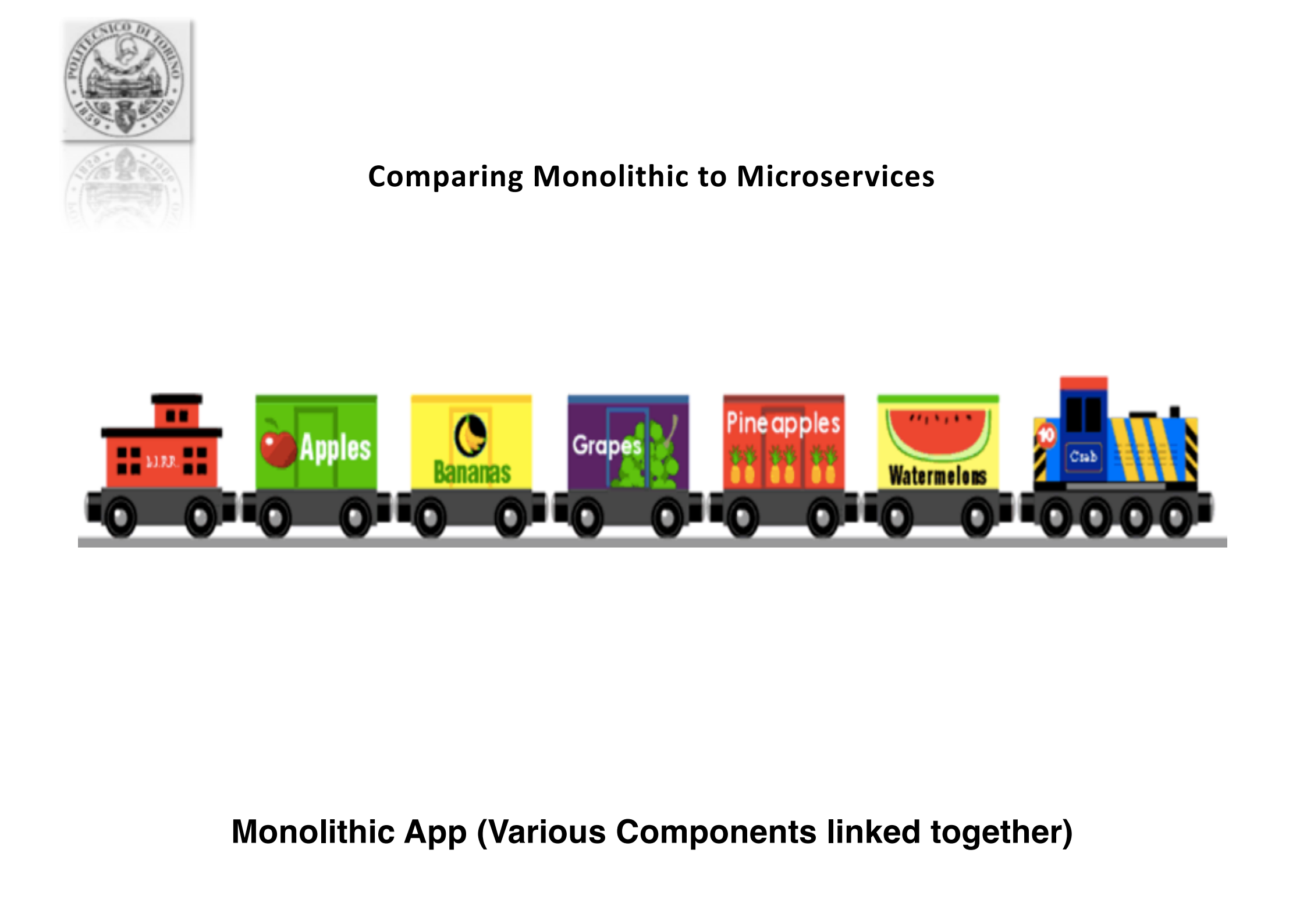
Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)


Clip slide
"service-oriented
architecture
composed of
loosely coupled
elements
that have
bounded contexts"
Adrian Cockcroft (former Cloud Architect at Netflix,
now Technology Fellow at Battery Ventures)
Self-contained; you can update the code without knowing anything about the internals of other microservices

Characteristics

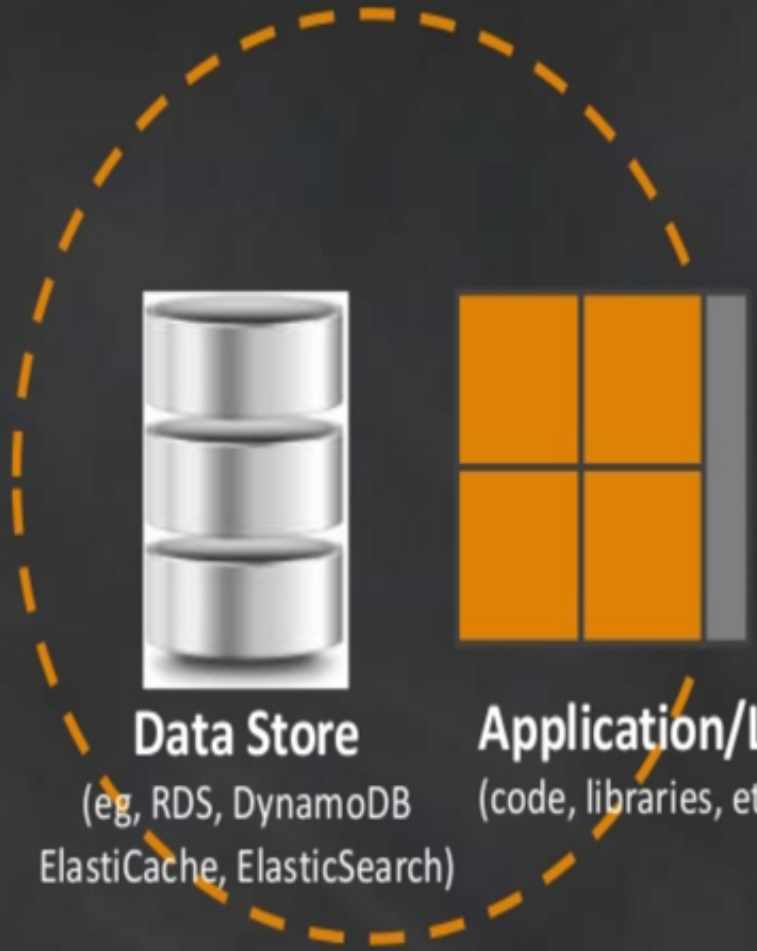
- Many smaller (fine grained), clearly scoped services
  - Single Responsibility Principle
  - Domain Driven Development
  - Bounded Context
  - Independently Managed
- Clear ownership for each service
  - Typically need/adopt the “DevOps” model

Attribution: Adrian Cockroft, Martin Fowler …


Comparing Monolithic to Microservices
Apples
Bananas
Grapes
Pineapples
Watermelons
Monolithic App (Various Components linked together)

Microservices – separate single purpose services

Anatomy of a Micro-service



Data Store
(eg, RDS, DynamoDB
ElastiCache, ElasticSearch)

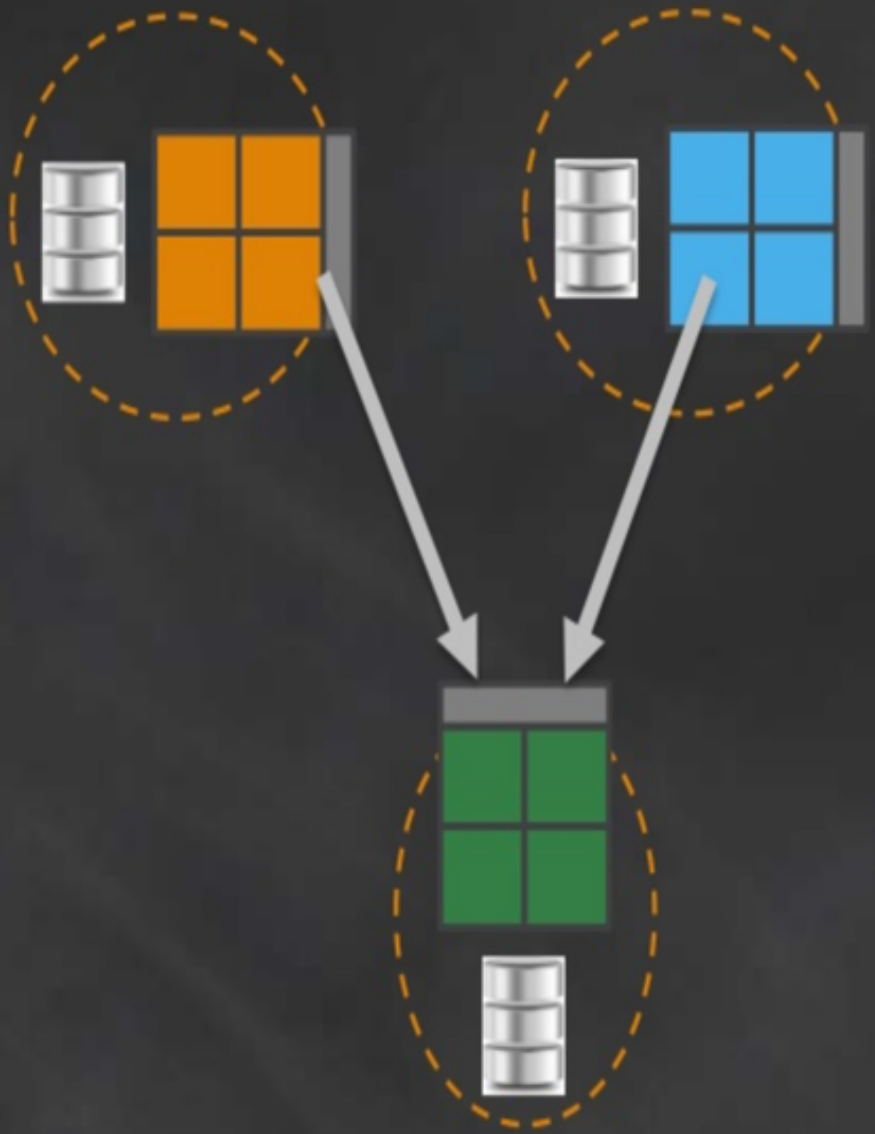
Application/Logic
(code, libraries, etc)

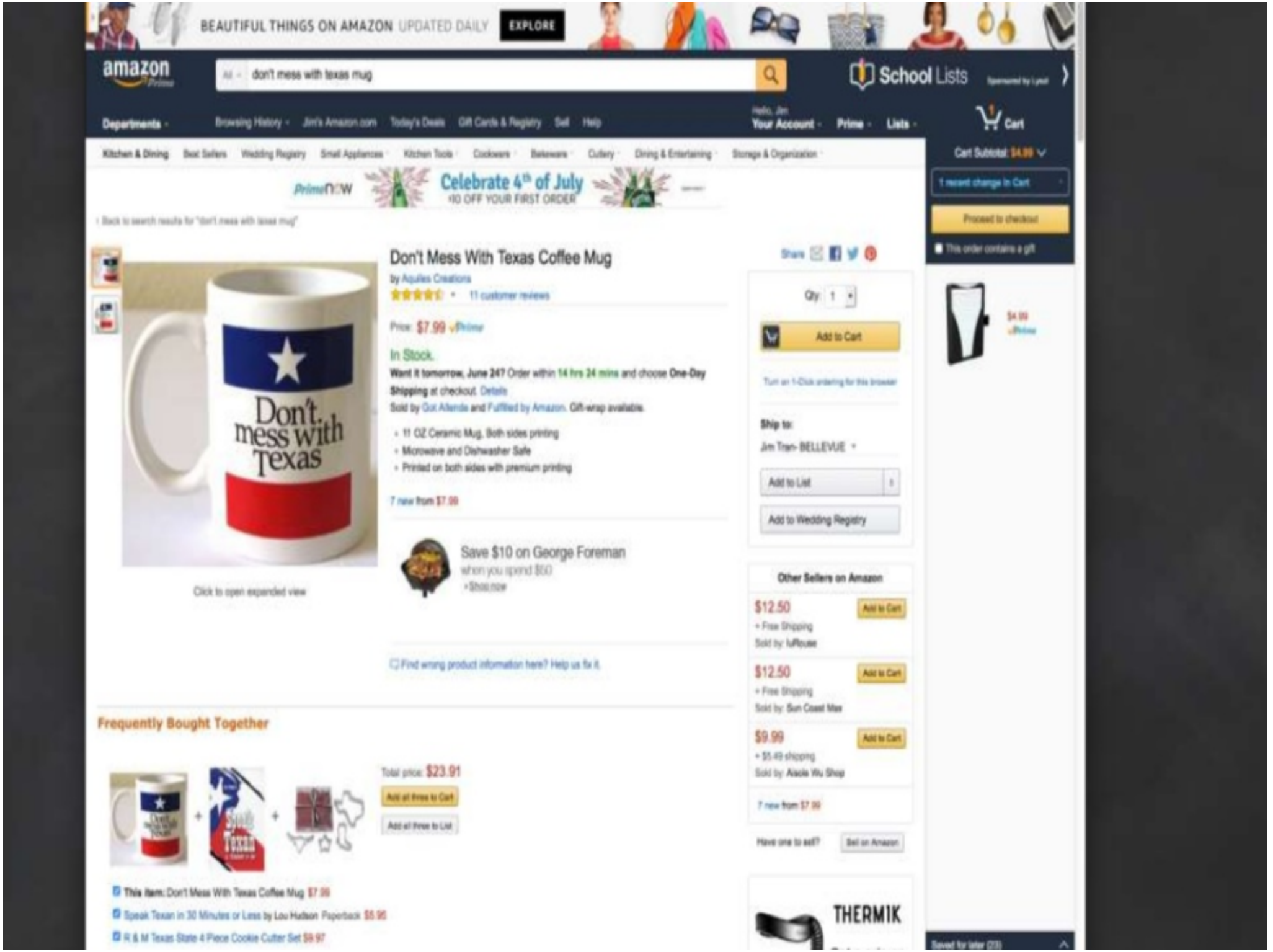
Public API

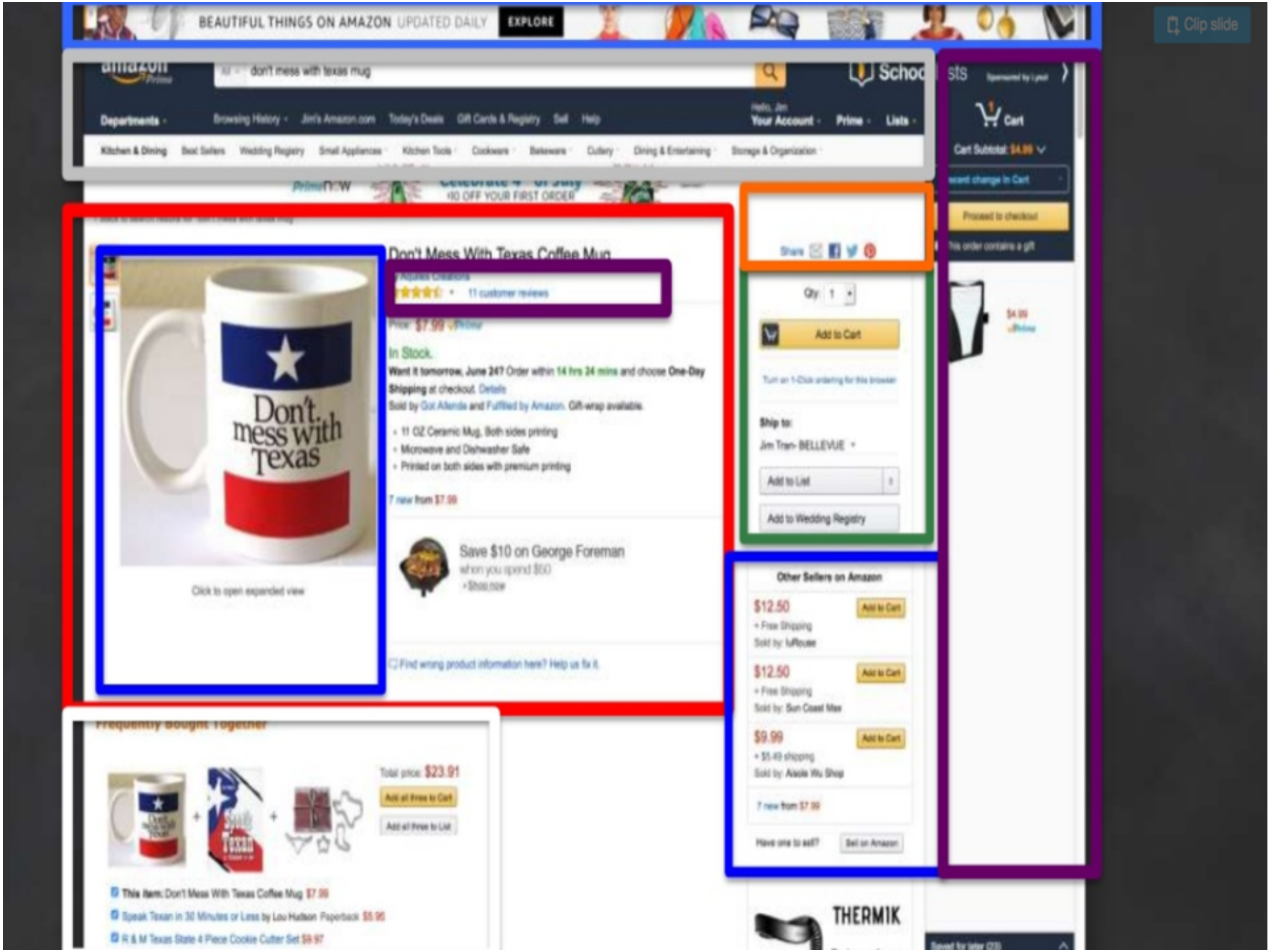
POST /restaurants
GET /restaurants

Publish /mytopic/restaurants
Subscribe /mytopic/restaurants

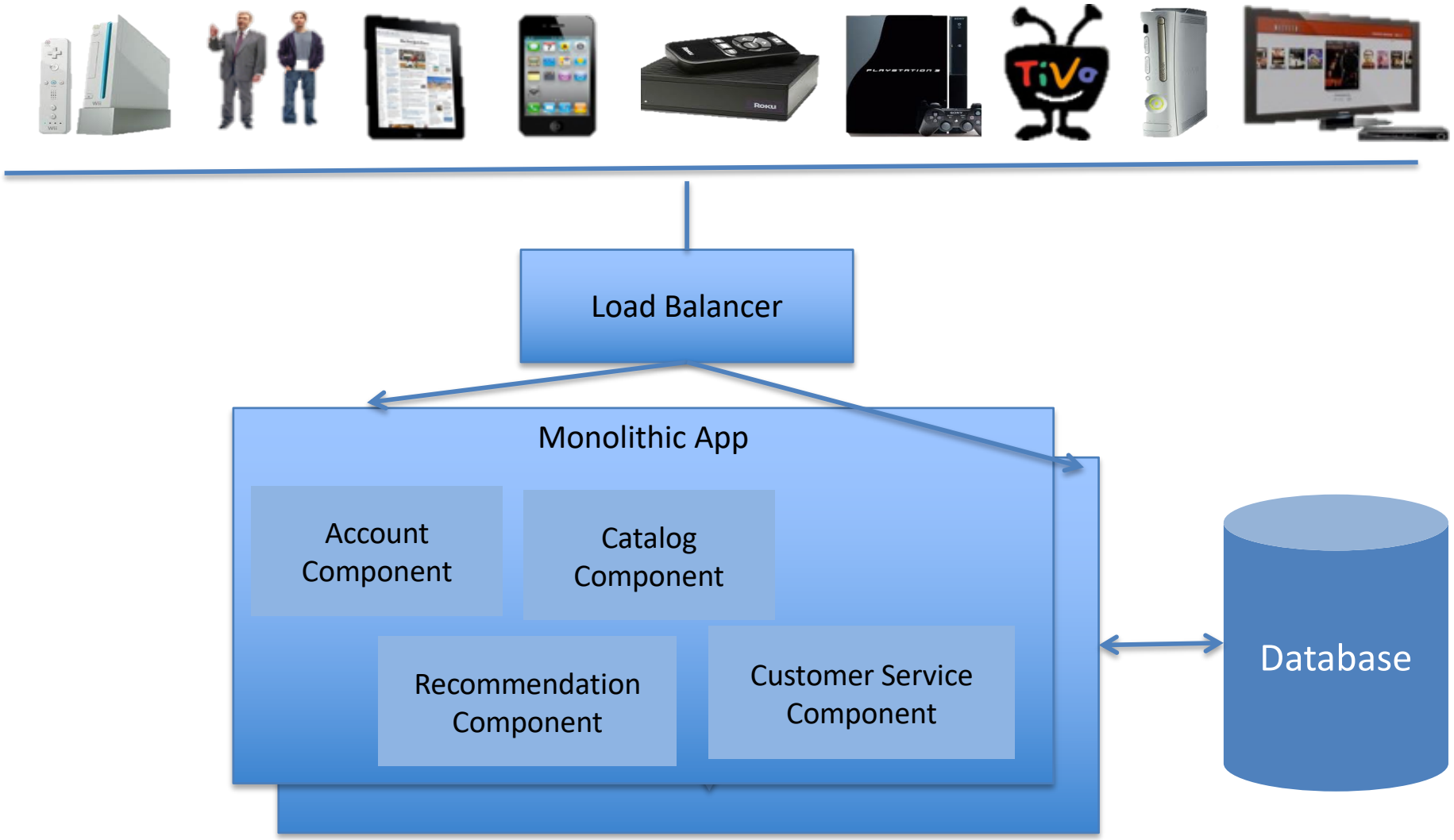
Avoid Software Coupling



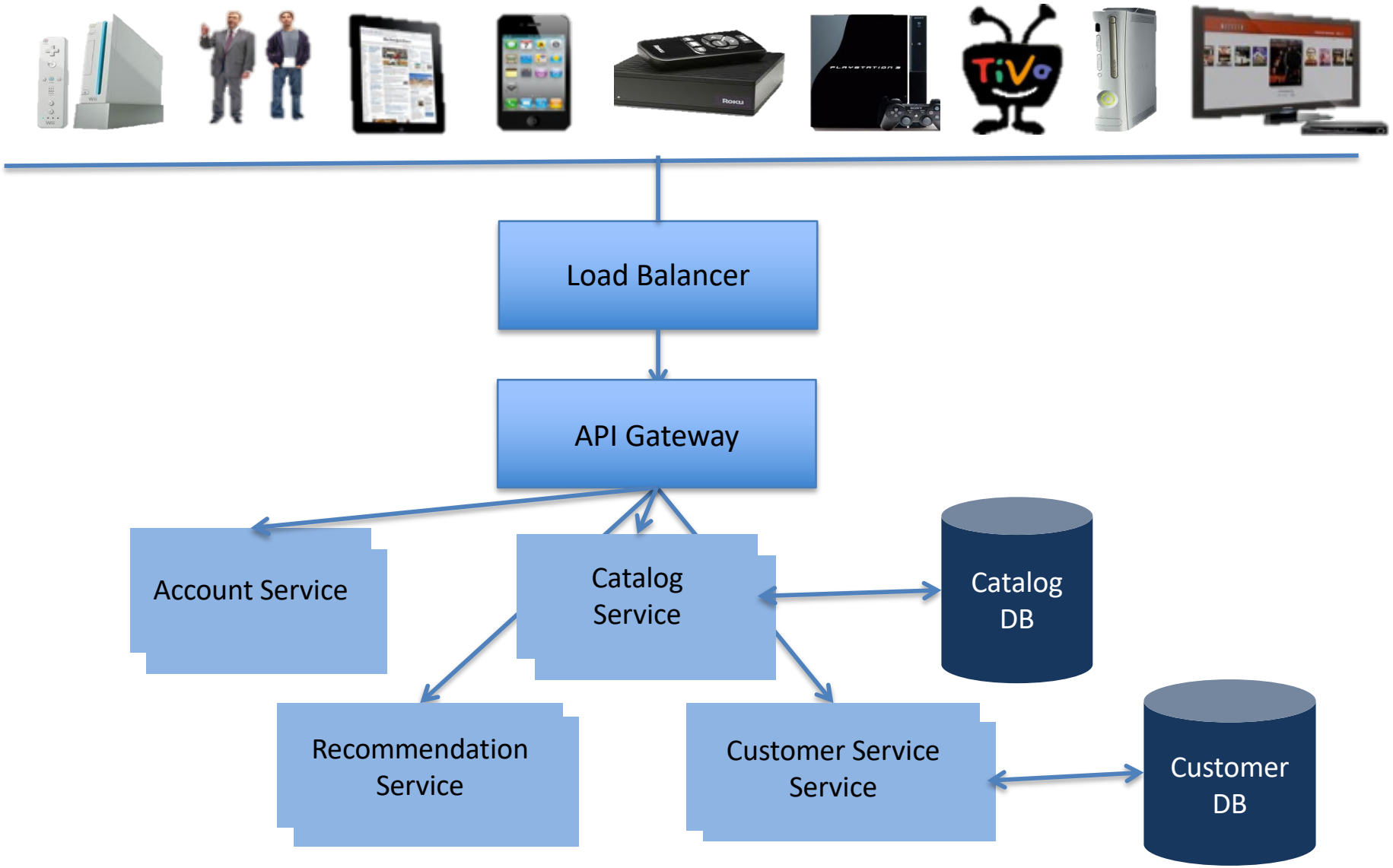

BEAUTIFUL THINGS ON AMAZON UPDATED DAILY
EXPLORE
amazon
don't mess with texas mug
School Lists
Departments
Browsing History
Jim's Amazon.com
Today's Deals
Gift Cards & Registry
Sell
Help
Your Account
Prime
Lists
Cart
Kitchen & Dining
Best Sellers
Wedding Registry
Small Appliances
Kitchen Tools
Cookware
Bakeware
Cutlery
Dining & Entertaining
Storage & Organization
PrimeNOW
Celebrate 4th of July
$10 OFF YOUR FIRST ORDER
Don't Mess With Texas Coffee Mug
by Aquiles Creations
11 customer reviews
Price: $7.99 Prime
In Stock.
Want it tomorrow, June 24? Order within 14 hrs 24 mins and choose One-Day Shipping at checkout. Details
Sold by Got Allanda and Fulfilled by Amazon. Gift-wrap available.
11 OZ Ceramic Mug, Both sides printing
Microwave and Dishwasher Safe
Printed on both sides with premium printing
7 new from $7.99
Don't mess with Texas
Click to open expanded view
Save $10 on George Foreman
when you spend $60
Shop now
Find wrong product information here? Help us fix it.
Share
Qty: 1
Add to Cart
Turn on 1-Click ordering for this browser
Ship to:
Jim Tran- BELLEVUE
Add to List
Add to Wedding Registry
Other Sellers on Amazon
$12.50
+ Free Shipping
Sold by: luRouse
Add to Cart
$12.50
+ Free Shipping
Sold by: Sun Coast Max
Add to Cart
$9.99
+ $5.49 shipping
Sold by: Alaska Wu Shop
Add to Cart
7 new from $7.99
Have one to sell?
Sell on Amazon
Frequently Bought Together
Total price: $23.91
Add all three to Cart
Add all three to List
This item: Don't Mess With Texas Coffee Mug $7.99
Speak Texan in 30 Minutes or Less by Lou Hudson Paperback $5.95
R & M Texas State 4 Piece Cookie Cutter Set $9.97
THERMIK
Cart Subtotal: $4.99
1 recent change in Cart
Proceed to checkout
This order contains a gift
$4.99
Saved for later (23)


BEAUTIFUL THINGS ON AMAZON UPDATED DAILY
EXPLORE
don't mess with texas mug
Departments
Your Account
Prime
Lists
Don't Mess With Texas Coffee Mug
11 customer reviews
Price: $7.99
In Stock.
Want it tomorrow, June 24? Order within 14 hrs 24 mins and choose One-Day Shipping at checkout. Details
Sold by Gol Allanda and Fulfilled by Amazon. Gift-wrap available.
11 OZ Ceramic Mug, Both sides printing
Microwave and Dishwasher Safe
Printed on both sides with premium printing
7 new from $7.99
Don't mess with Texas
Click to open expanded view
Save $10 on George Foreman
when you spend $50
Shop now
Find wrong product information here? Help us fix it.
Add to Cart
Qty: 1
Add to List
Add to Wedding Registry
Ship to:
Jim Tran- BELLEVUE
Other Sellers on Amazon
$12.50
+ Free Shipping
Sold by: luRouse
$12.50
+ Free Shipping
Sold by: Sun Coast Max
$9.99
Sold by: Alaole Wu Shop
7 new from $7.99
Have one to sell?
Sell on Amazon
Cart
Cart Subtotal:
Proceed to checkout
This order contains a gift
Frequently Bought Together
Total price: $23.91
This item: Don't Mess With Texas Coffee Mug $7.99
Speak Texan in 30 Minutes or Less by Lou Hudson Paperback $5.95
R & M Texas State 4 Piece Cookie Cutter Set $9.97
THERMIK

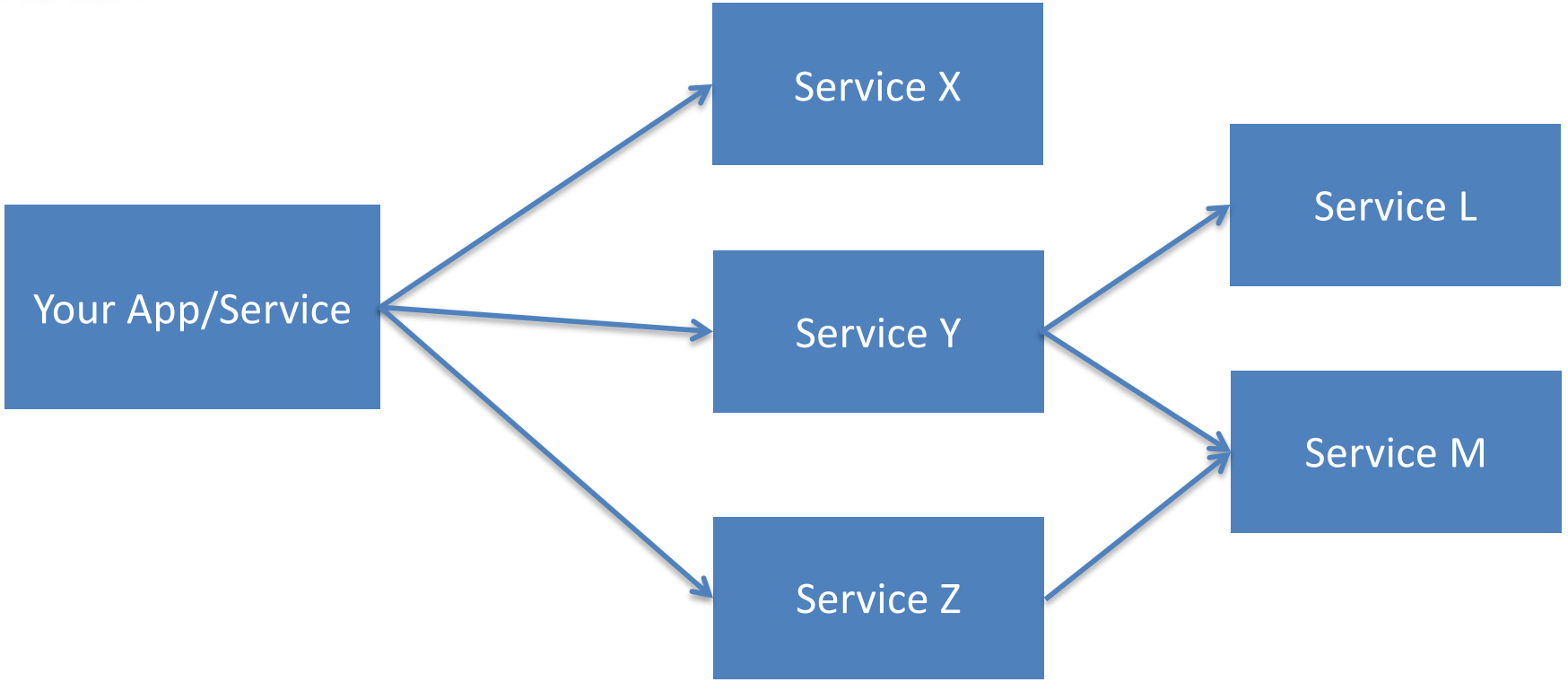
Monolithic Architecture (Revisiting)



Microservices Architecture



Concept -> Service Dependency Graph



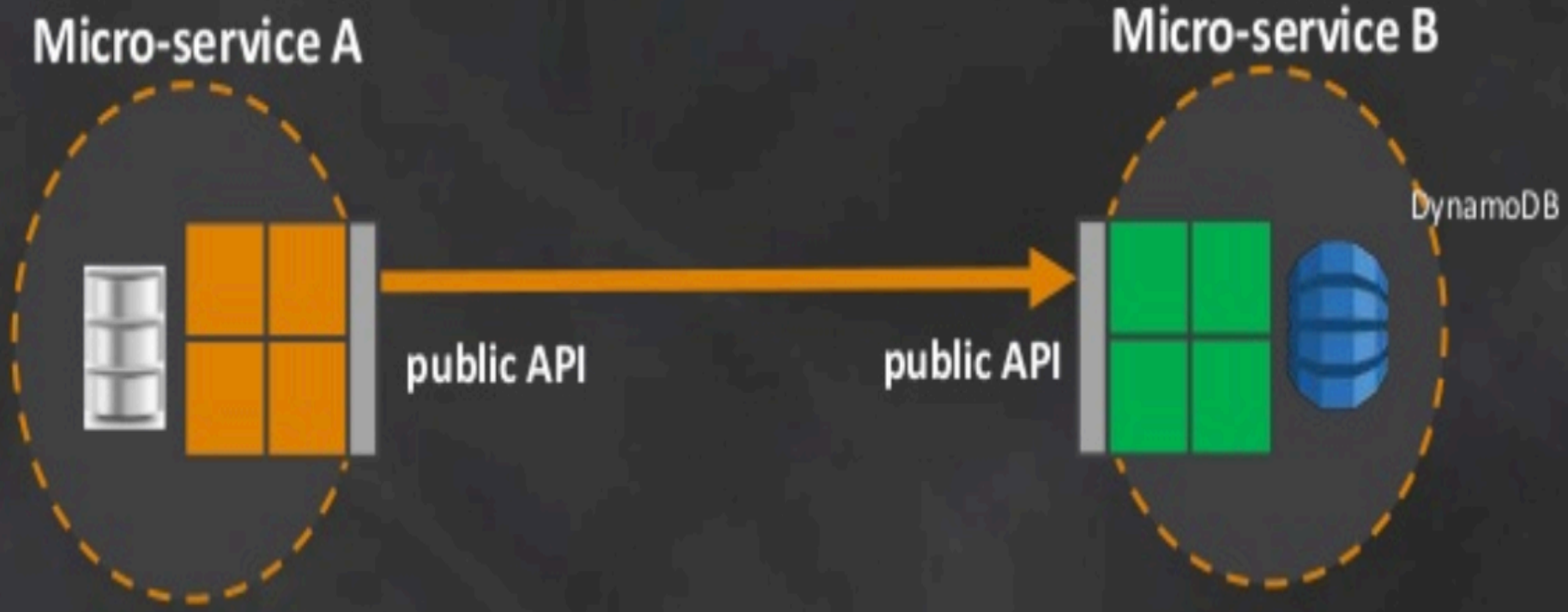
Why?

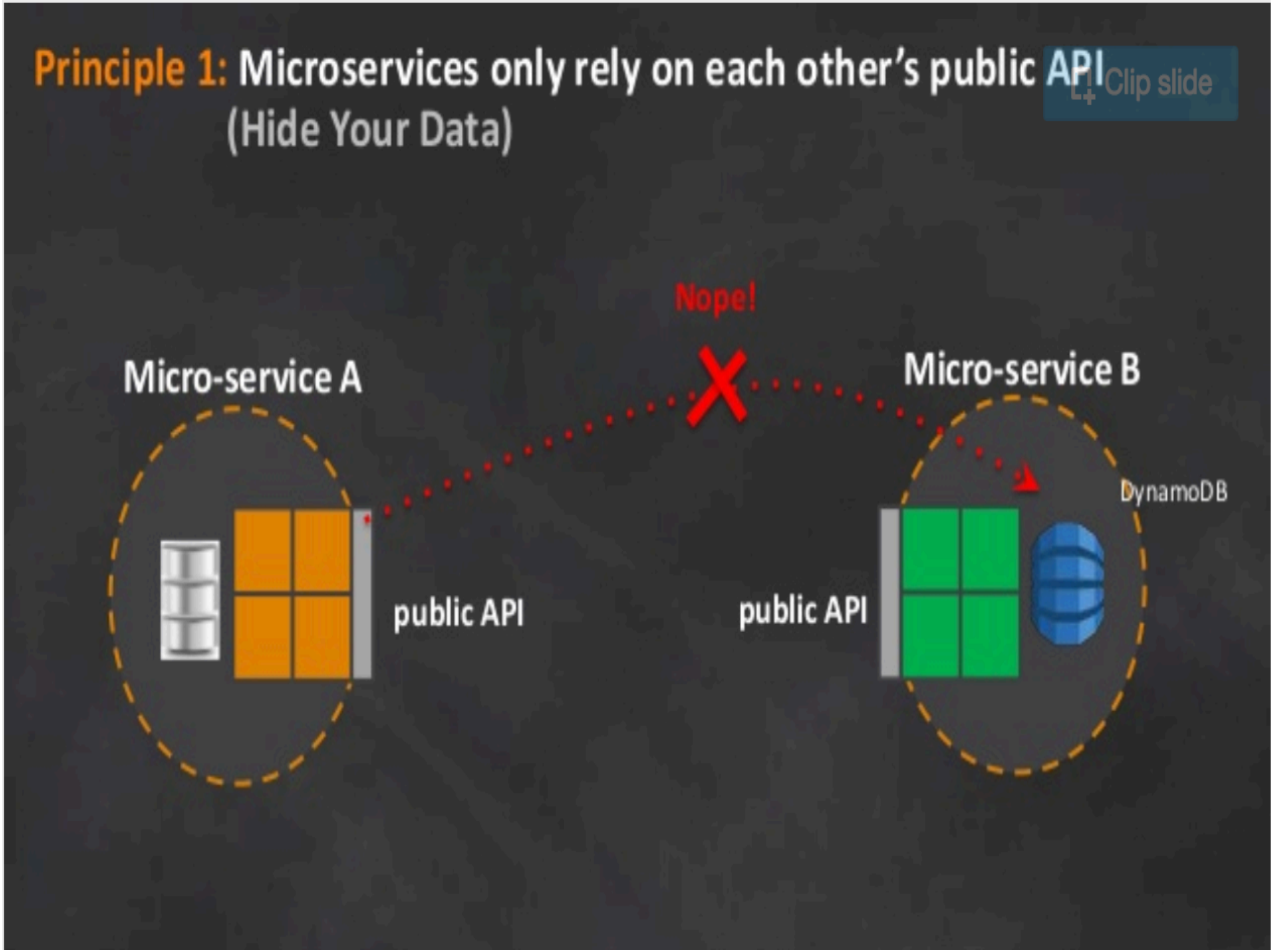
- Faster and simpler deployments and rollbacks
  - Independent Speed of Delivery (by different teams)
- Right framework/tool/language for each domain
  - Recommendation component using Python, Catalog Service in Java ..
- Greater Resiliency
  - Fault Isolation
- Better Availability
  - If architected right ☺

Principle 1

Micro-services only rely on each other's public API

Principle 1: Microservices only rely on each other's public API




Principle 1: Microservices only rely on each other's public API
(Hide Your Data)
Clip slide
Nope!
Micro-service A
Micro-service B
public API
public API
DynamoDB

Principle 1: Microservices only rely on each other's public API (Evolve API in backward-compatible way...and document!)

Clip slide

Micro-service A

public API

Version 1.0.0

storeRestaurant (*id, name, cuisine*)

Version 1.1.0

storeRestaurant (id, name, cuisine)
storeRestaurant (id, name, arbitrary_metadata)
addReview (restaurantId, rating, comments)

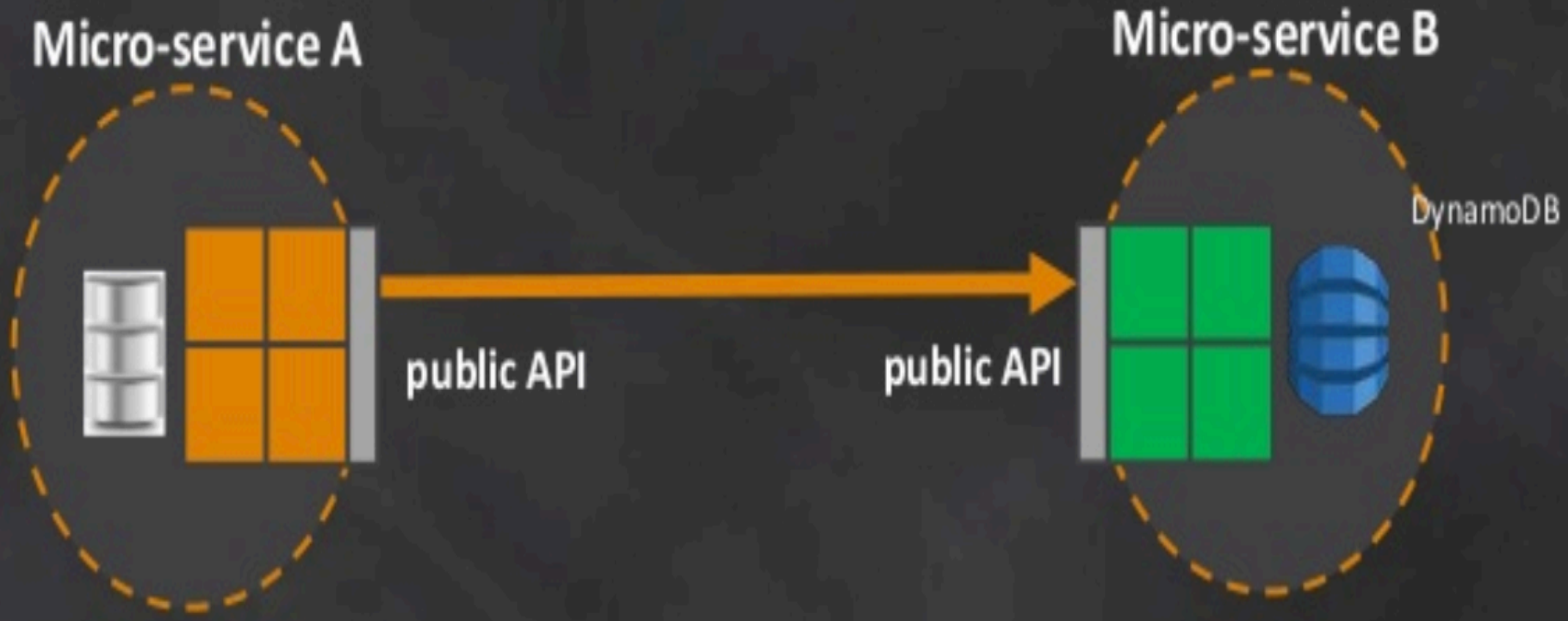
Version 2.0.0

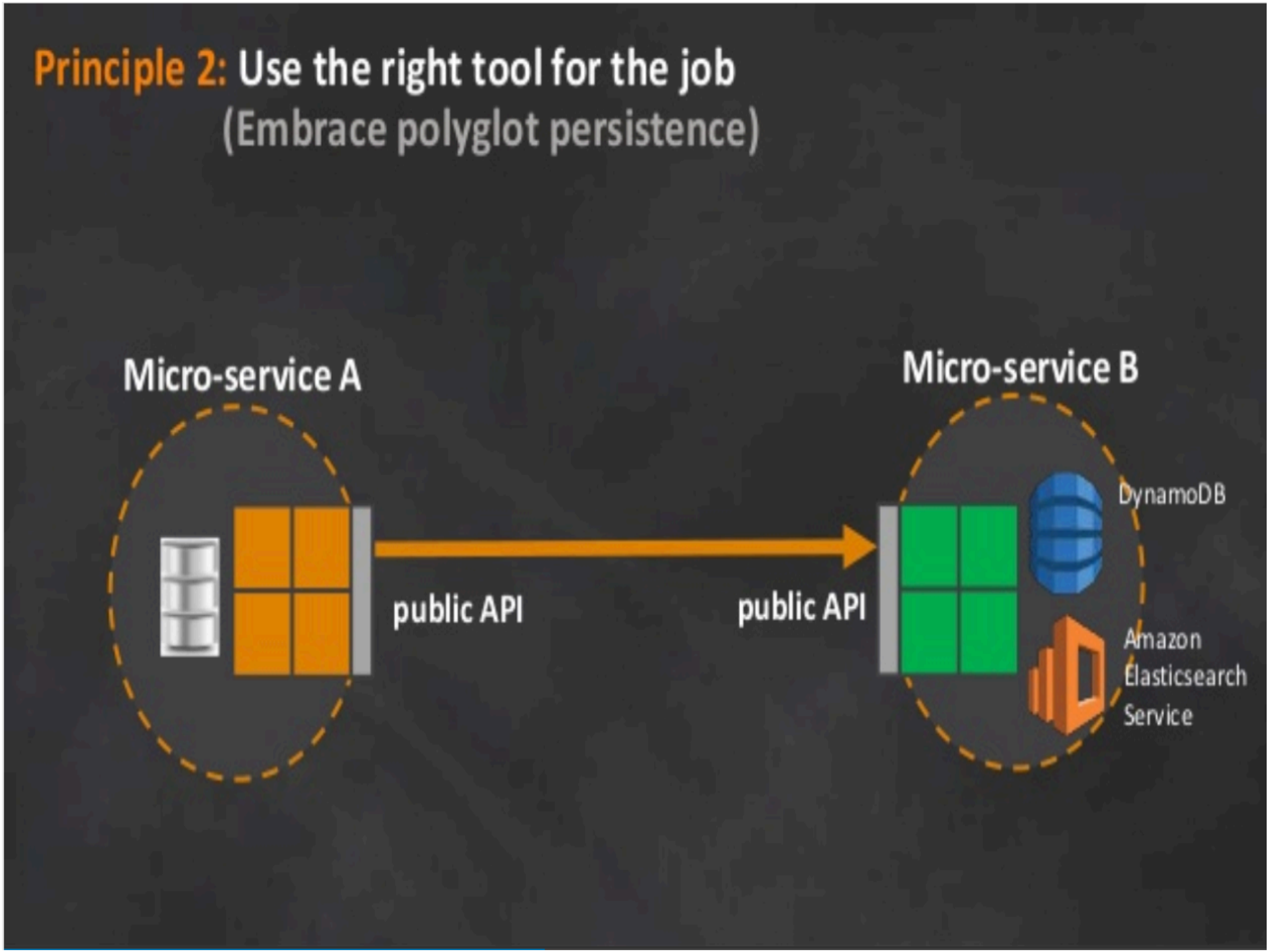
storeRestaurant (id, name, arbitrary_metadata)
addReview (restaurantId, rating, comments)

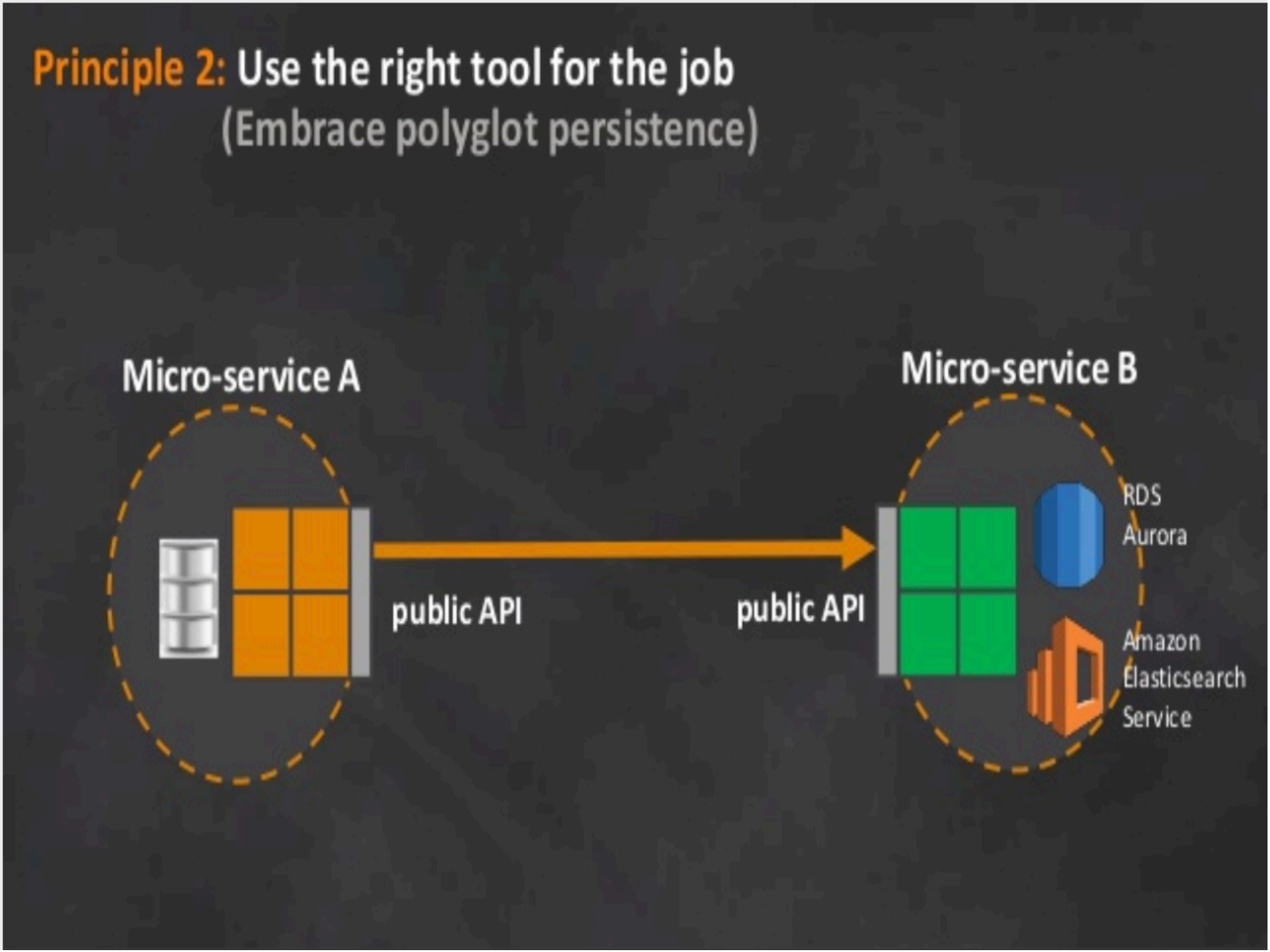
Principle 2

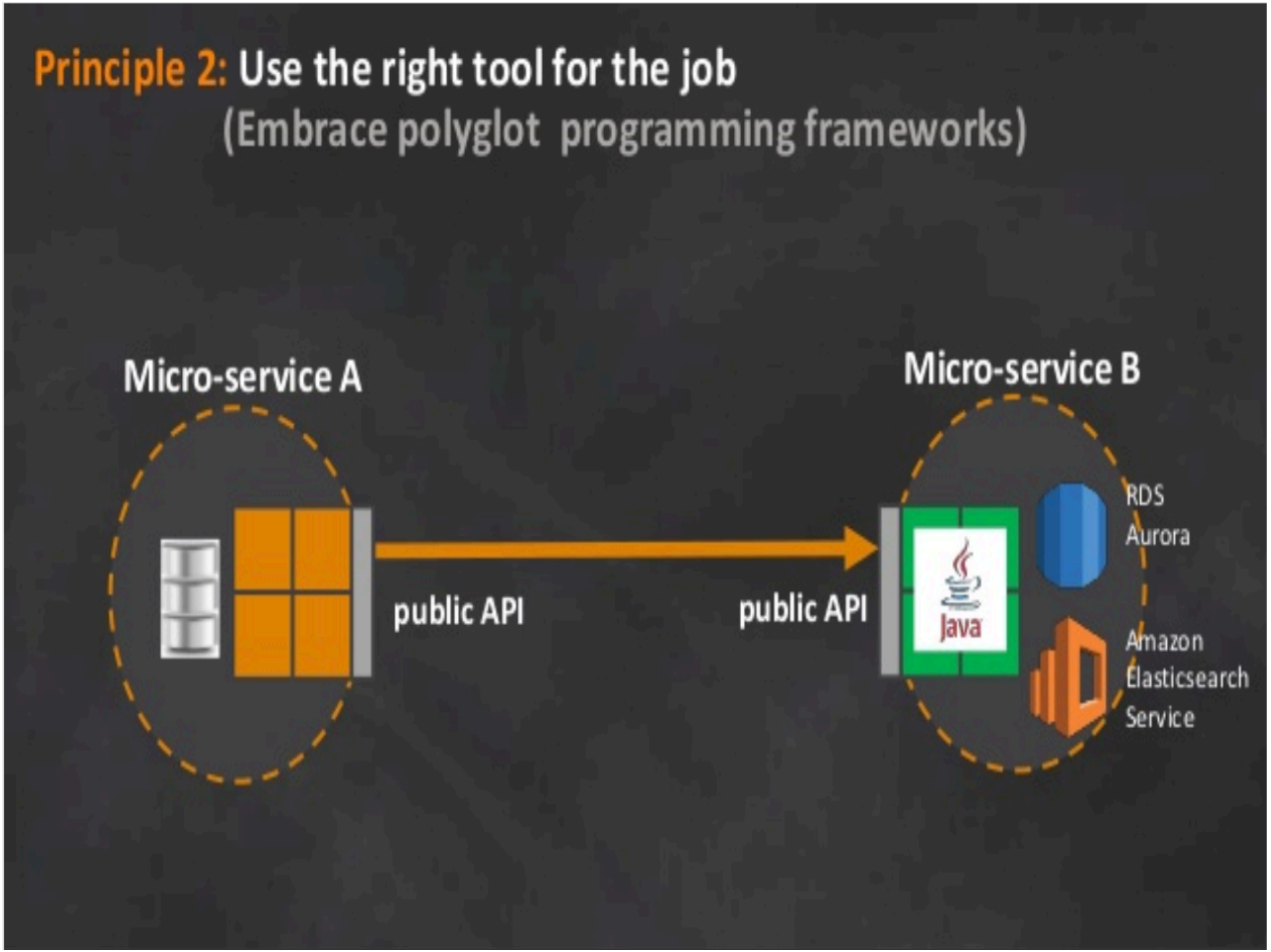
Use the right tool for the job

Principle 2: Use the right tool for the job
(Embrace polyglot persistence)



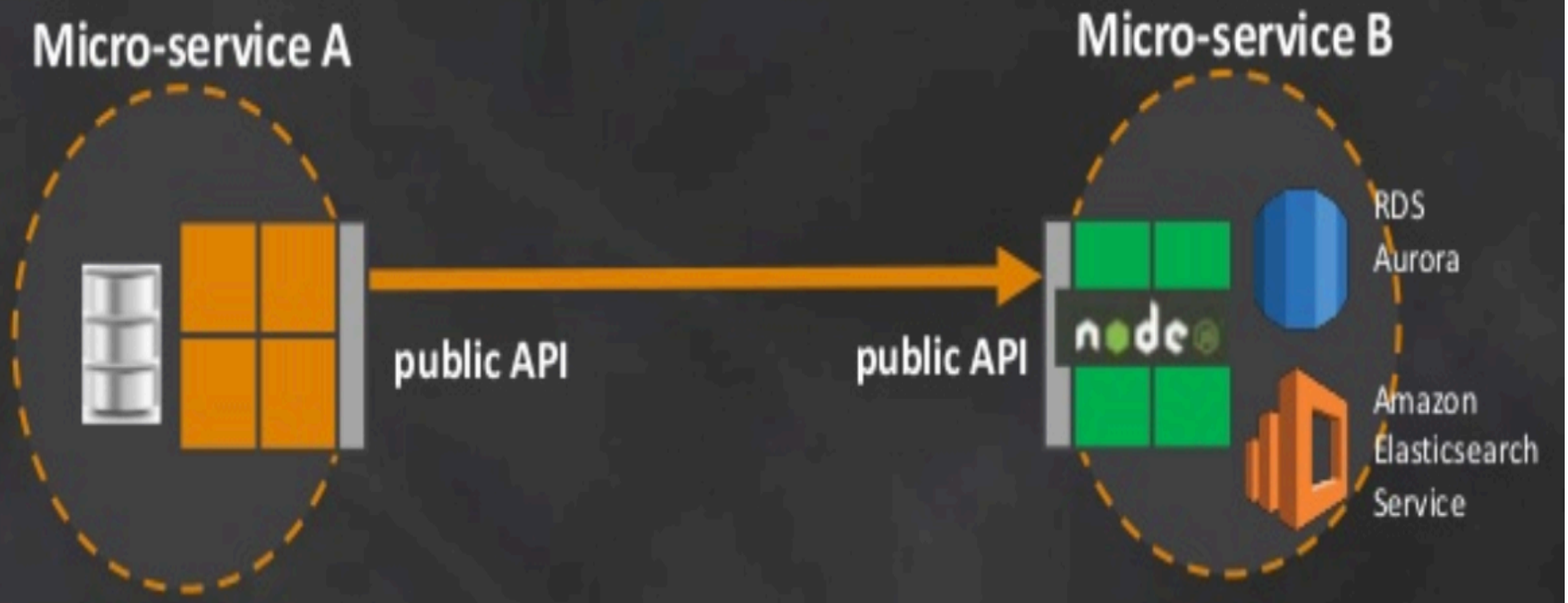

Principle 2: Use the right tool for the job
(Embrace polyglot persistence)
Micro-service A
Micro-service B
public API
public API
DynamoDB
Amazon
Elasticsearch
Service


Principle 2: Use the right tool for the job
(Embrace polyglot persistence)
Micro-service A
Micro-service B
public API
public API
RDS
Aurora
Amazon
Elasticsearch
Service

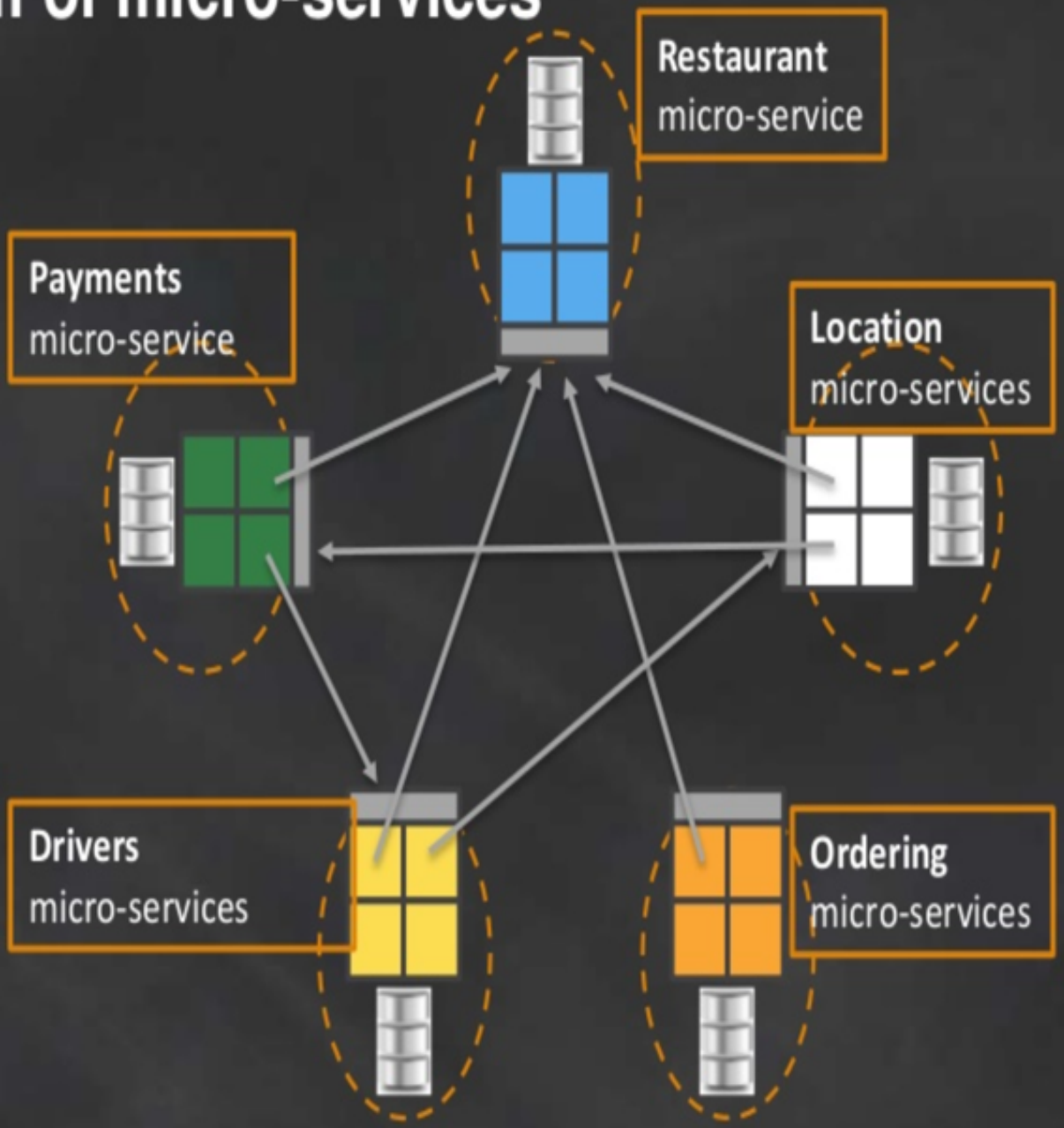

Principle 2: Use the right tool for the job
(Embrace polyglot programming frameworks)
Micro-service A
Micro-service B
public API
public API
Java
RDS
Aurora
Amazon
Elasticsearch
Service

Principle 2: Use the right tool for the job

(Embrace polyglot programming frameworks)



Ecosystem of micro-services

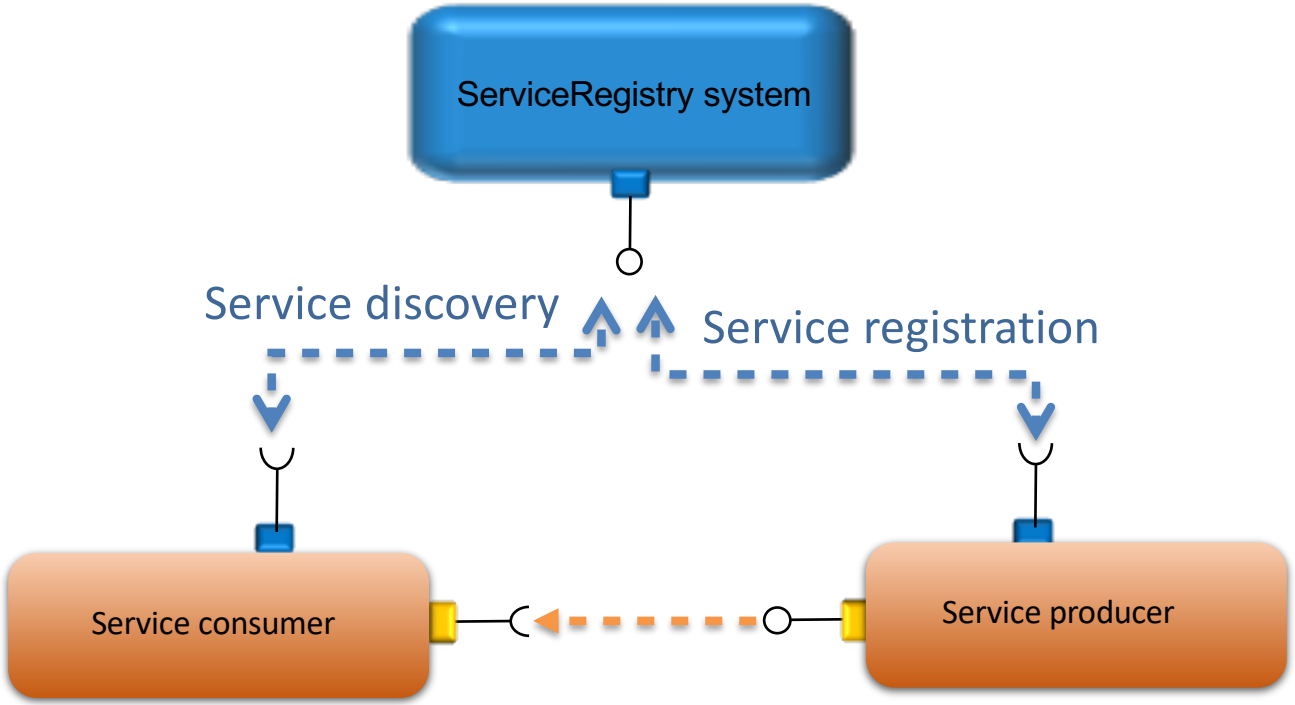


Properties of the Microservices pattern

- **Loosely coupling**
  - Two Microservices systems do not need to know about each other at design time to allow a run time data exchange.
  - The identification of available system services is established at run-time making use of **service registry, device registry and discovery mechanisms.**
  - A new **Microservice will register** itself in the **service registry** upon which it will be discoverable by any other service in the network.
  - A new **device will register** itself in the **device registry** upon which it will be discoverable by any other service in the network.

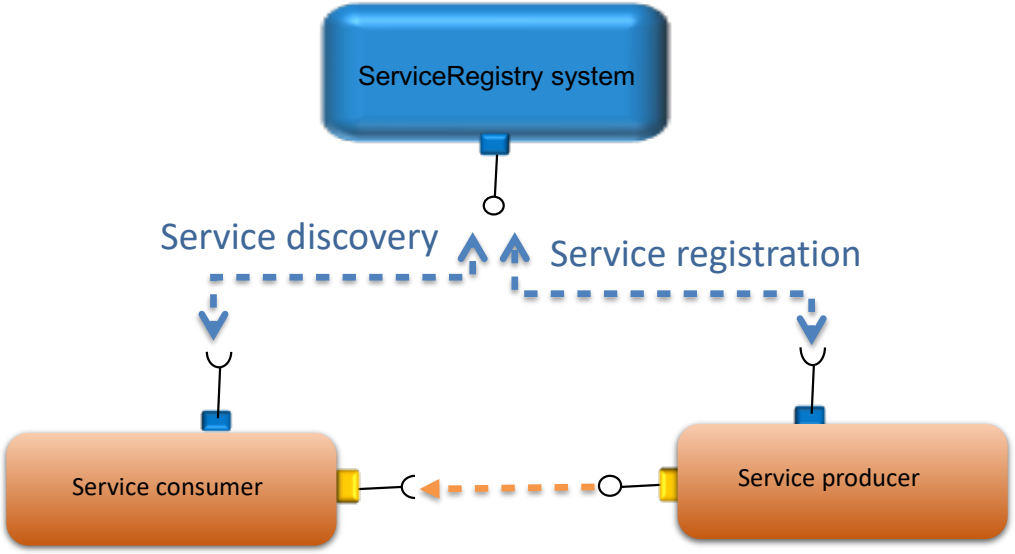
Properties of the Microservices pattern

- **Loosely coupling**



Properties of the Microservices pattern

- **Loosely coupling**
  - The objective of a **service registry system** is to provide all active microservices registered within a distributed platform and enable the discovery of them.
  - A **service registry** is an independent system that provides one service and (usually) does not consume any other services



Properties of the Microservices pattern

- **Loosely coupling**
  - The objective of a **device registry system** is to register devices and keep track of those that are active and connected to the Internet
  - A **device registry system** shall store metadata about the device, addressing non-functional information such as software revision, deployment info, etc.
  - A **device registry** is an independent system that provides one service and could not consume any other services
  - Devices can register themselves through a specific **Gateway** or **Device Connector**

Properties of the Microservices pattern

- **Late binding**
  - In a Microservices system the exchange of data between two systems is established in runtime.
  - The run-time coupling can be initiated by an **orchestration mechanism** that provides mechanisms for distributing orchestration rules and and remote routines (e.g. web services) consumption patterns. (Sometimes, it can even provide some synchronization mechanisms).
  - Sometimes **orchestration mechanisms** help in avoiding caos

Properties of the Microservices pattern

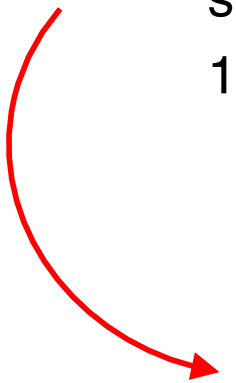
- **Autonomy**
  - **Each** device and related **software system can act on its own regardless of other systems**. Thus it is responsible for its own data and functionalities. Once a microservice exchange is set up between two systems this exchange may go on without further involvement of any supporting services/functionalities.

Properties of the Microservices pattern

- **Pull and push behavior** (1)
  - In a Microservices environment the data exchange can be initiated by a service consumer requesting data - a **pull behavior**.
    - A pull behavior can for example be controlled by a timer at the service consumer, thus creating data pulling of a sensor every 100ms as an example.

Properties of the Microservices pattern

- **Pull and push behavior** (1)
  - In a Microservices environment the data exchange can be initiated by a service consumer requesting data - a **pull behavior**.
    - A pull behavior can for example be controlled by a timer at the service consumer, thus creating data pulling of a sensor every 100ms as an example.



- **Request/Response**
- **REST**

Properties of the Microservices pattern

- **Pull and push behavior** (2)
  - The data exchange can also be initiated by a producer that knows about conditional data request - a **push behavior**.
    - This is initiated by a data subscription under certain criteria. For example, a pressure sensor will push its pressure reading service to a consumer whenever the pressure reading is higher than 2 bar. Data is then pushed from the producer to the consumer.
  - **REMEMBER:** a Message Broker is needed

Properties of the Microservices pattern

- **Pull and push behavior** (2)
  - The data exchange can also be initiated by a producer that knows about conditional data request - a **push behavior**.
    - This is initiated by a data subscription under certain criteria. For example, a pressure sensor will push its pressure reading service to a consumer whenever the pressure reading is higher than 2 bar. Data is then pushed from the producer to the consumer.
  - **REMEMBER:** a Message Broker is needed
    - **Publish/Subscribe**
    - **Event driven**

To Recap

- **Properties:**
  - Loosely coupling
  - Late binding
  - Autonomy
  - Pull and push behavior

To Recap

- **Properties:**
  - Loosely coupling
  - Late binding
  - Autonomy
  - Pull and push behavior
- **Main software entities…**
  - service registry system
  - device registry system
  - gateways or device connectors
  - orchestrator
  - Message Broker
- **…and your application components**

To Recap

- **Properties:**
  - Loosely coupling
  - Late binding
  - Autonomy
  - Pull and push behavior
- **Main software entities…**
  - service registry system
  - device registry system
  - gateways or device connectors
  - orchestrator
  - Message Broker
- **…and your application components**

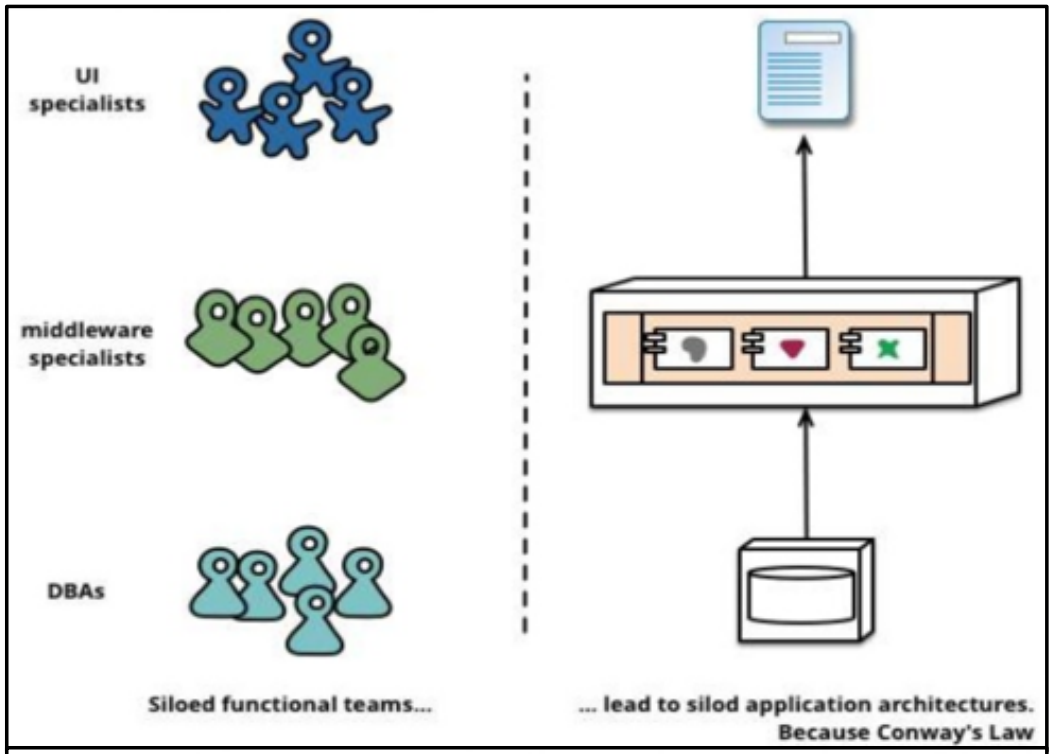
Both properties and software entities are also provided by Service Oriented Architecture

Conway's Law

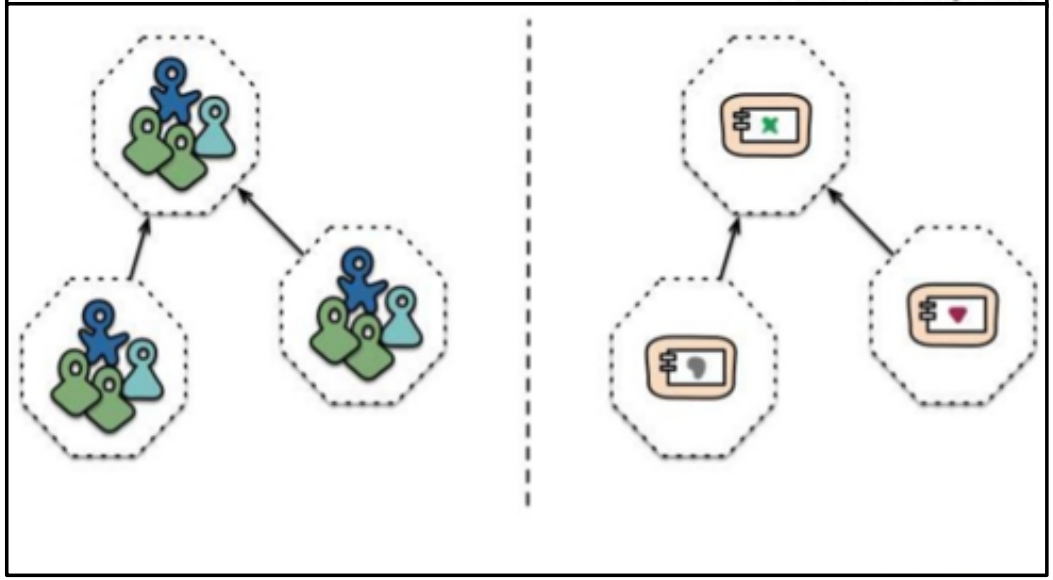
"Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."

Melvin E. Conway, 1967

Teams in monolithic infrastructure



Teams in microservices infrastructure



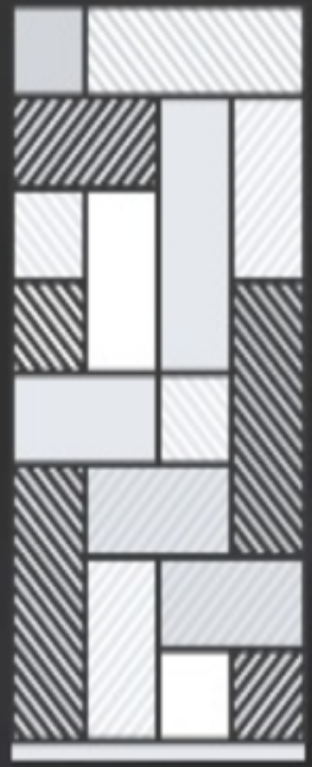
Monolith development lifecycle

Clip slide

developers

delivery pipeline

app
(aka the“monolith”)

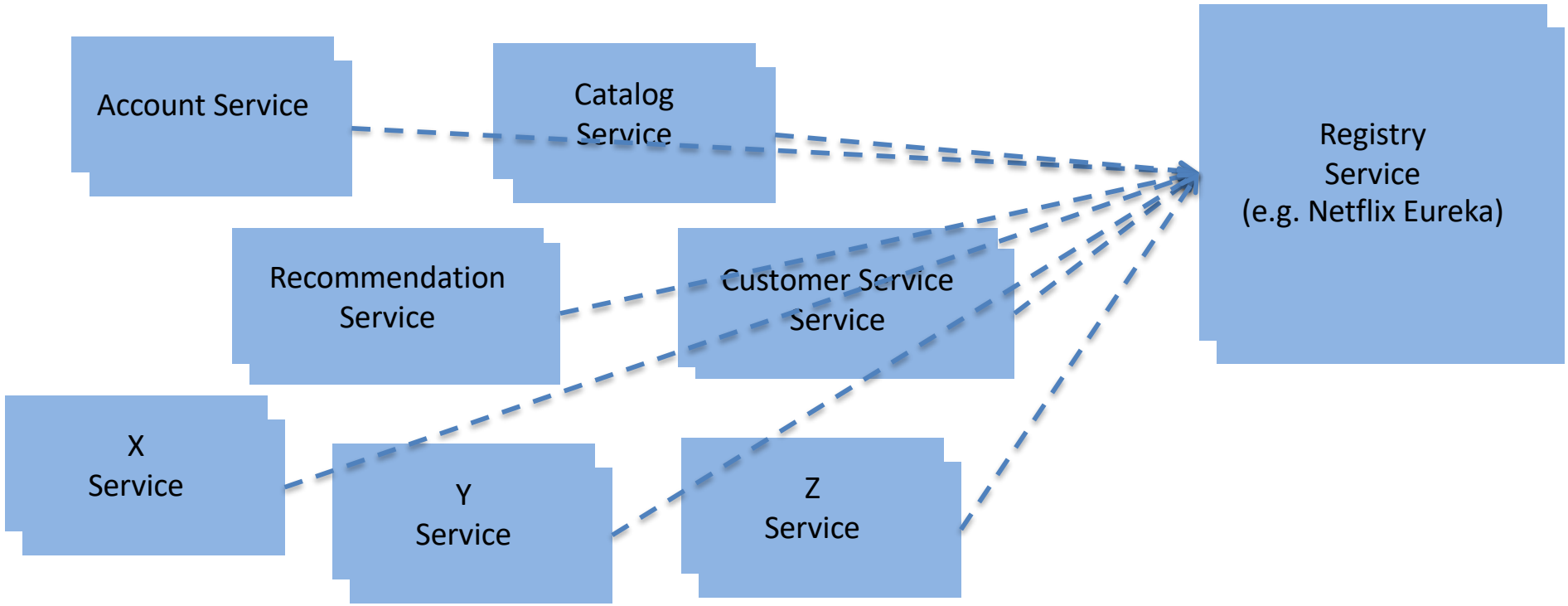


Microservices deployment lifecycle


Clip slide
Thousands of teams
× Microservice architecture
× Continuous delivery
× Multiple environments
= 50 million deployments a year
In Amazon
(5708 per hour, or every 0.63 second)

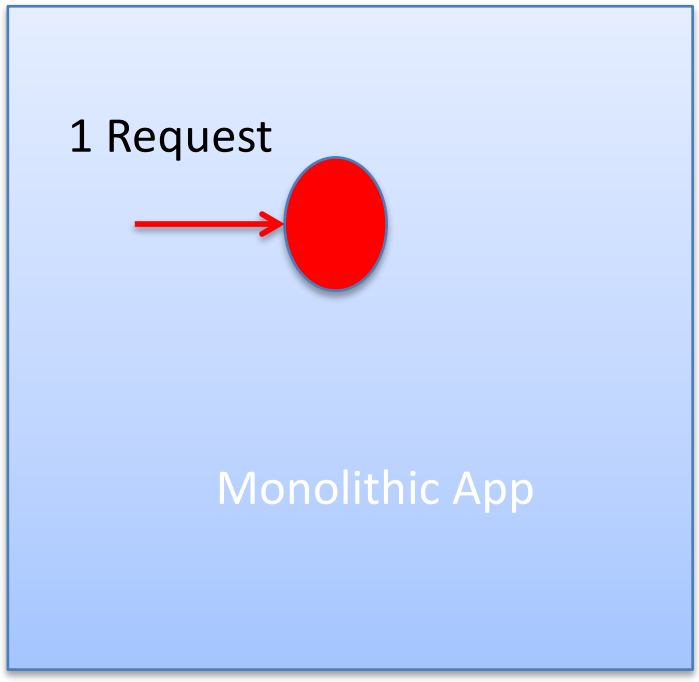
Service Registry and Discovery

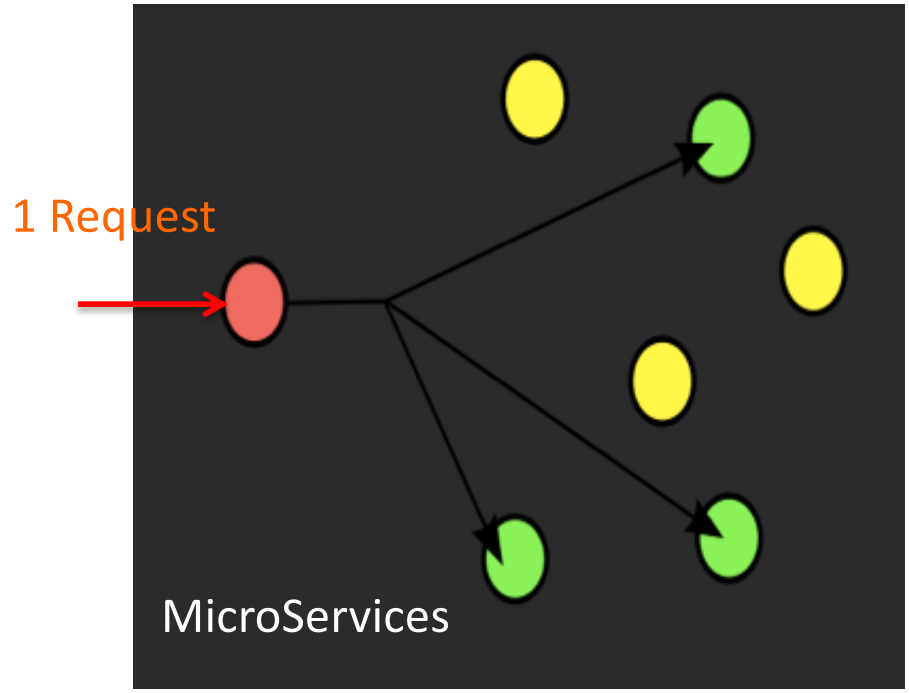
- 100s of Microservices
  - Need a Service Metadata Registry (Discovery Service)



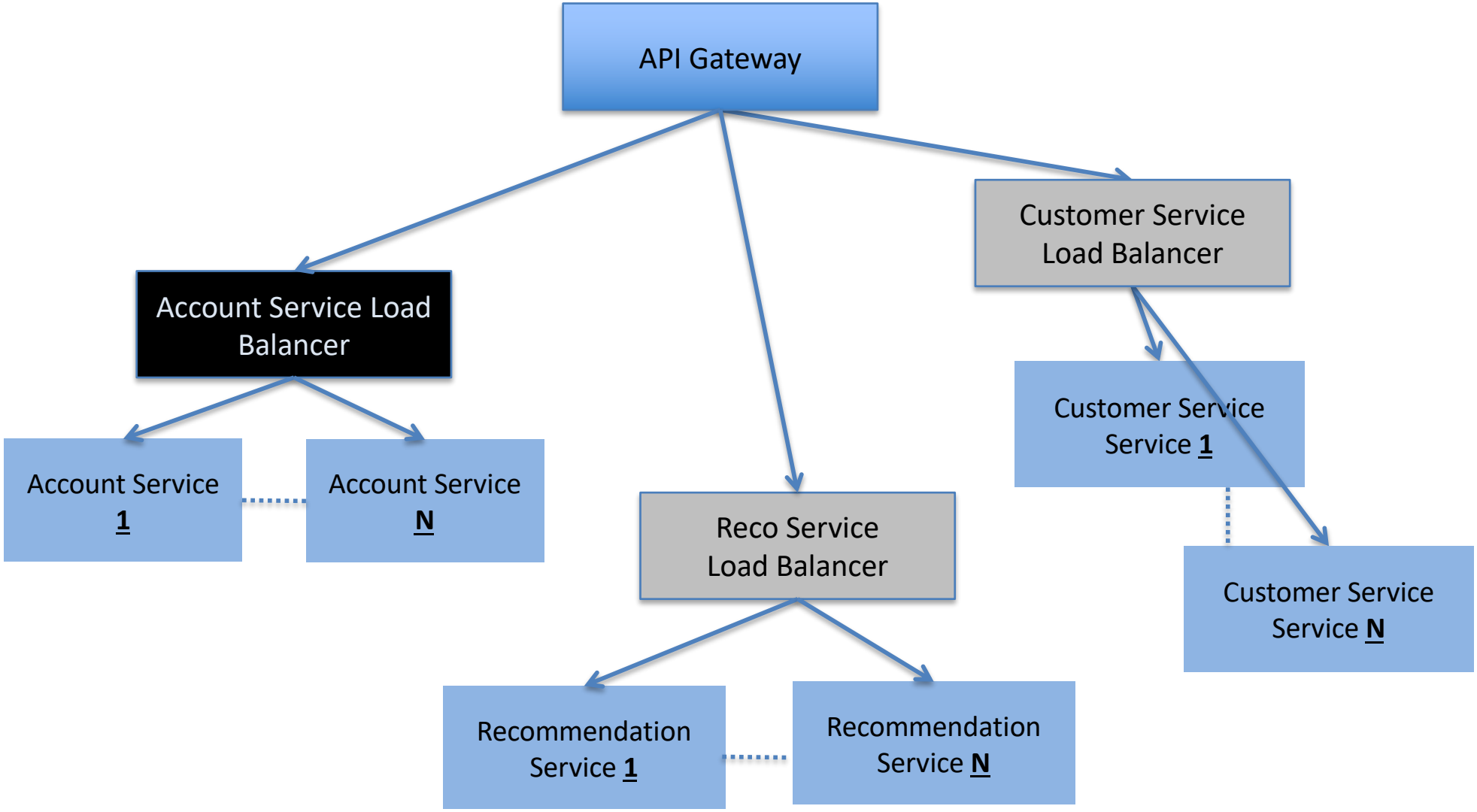
Chattiness (and Fan Out)

- ~2 Billion Requests per day on Edge Service
- Results in ~20 Billion Fan out requests in ~100 MicroServices

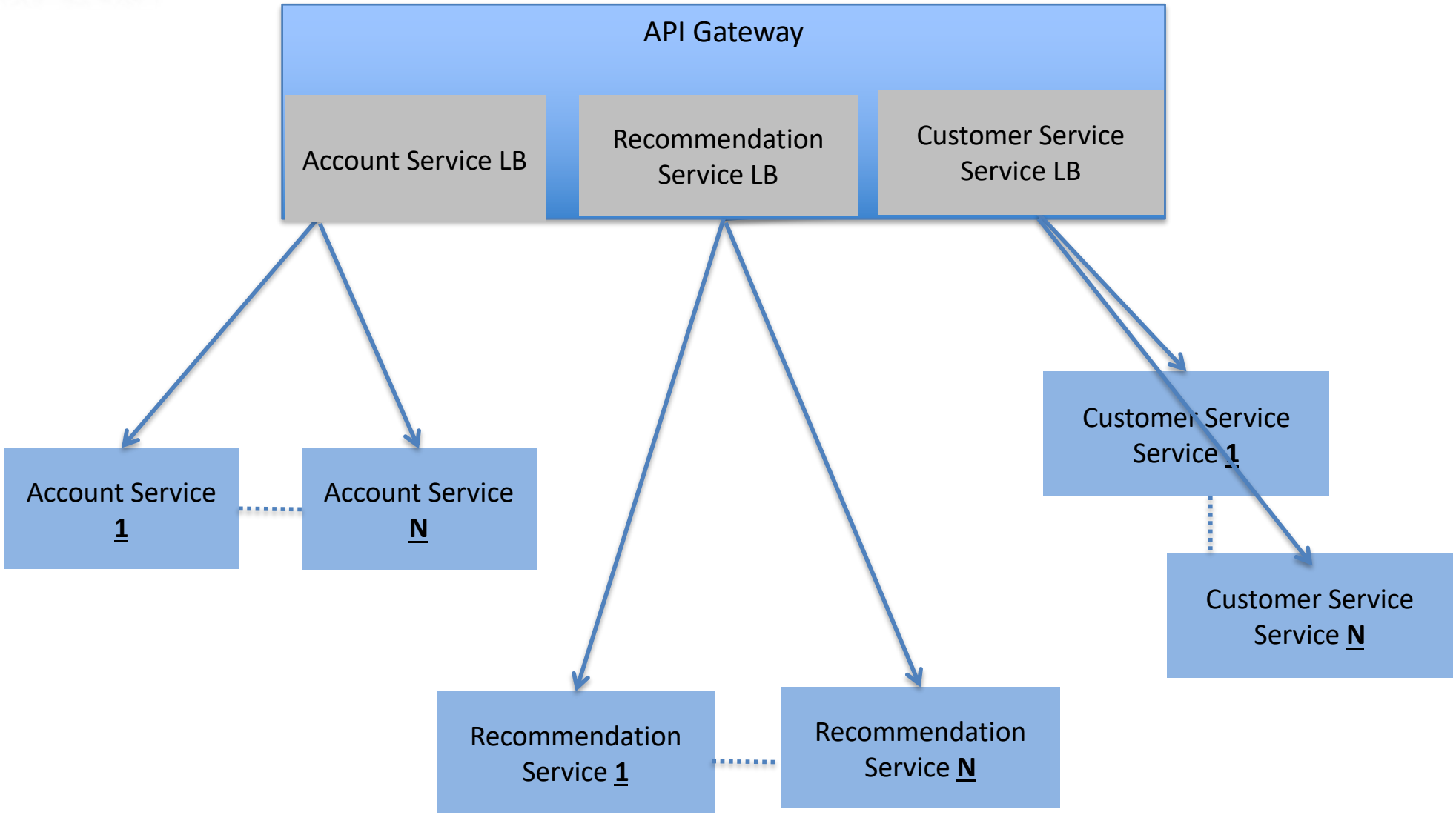




Central (Proxy) Loadbalancer



Client Loadbalancer



Challenges with monolithic software

Difficult to scale	Architecture is hard to maintain and evolve	Lack of agility
Long Build/Test/Release Cycles (who broke the build?)	New releases take months	Lack of innovation
Operations is a nightmare (module X is failing, who's the owner?)	Long time to add new features	Frustrated customers

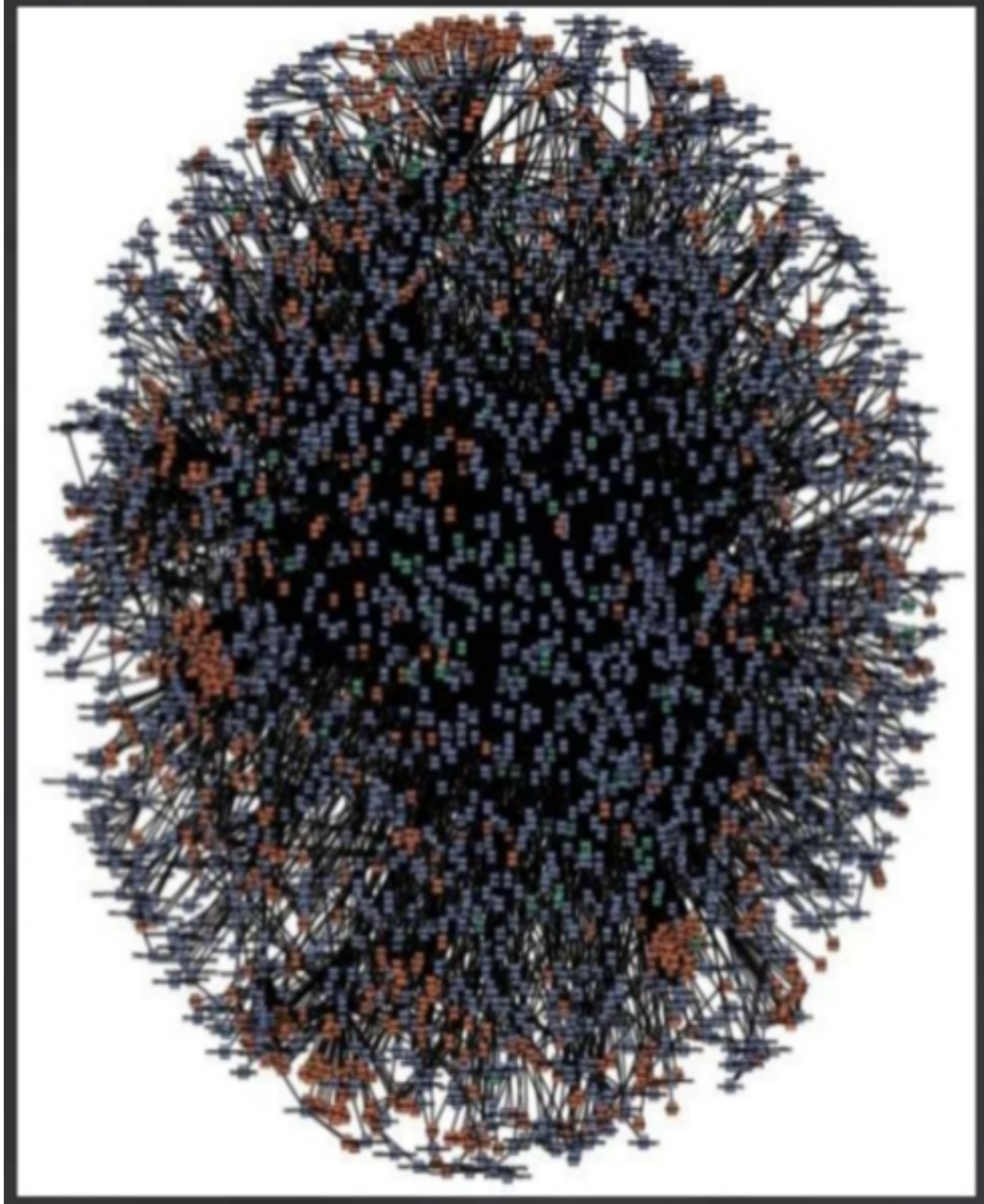
Benefits of microservices

Easier to scale each individual micro-service	Easier to maintain and evolve system	Increased agility
Rapid Build/Test/Release Cycles	New releases take minutes	Faster innovation
Clear ownership and accountability	Short time to add new features	Delighted customers

Challenges

Can lead to chaos if not designed right …

It's a journey…



Expect challenges along the way…

- Understanding of business domains
- Coordinating txns across multiple services
- Eventual Consistency
- Service discovery
- Lots of moving parts requires increased coordination
- Complexity of testing / deploying / operating a distributed system
- Cultural transformation

See also….

https://www.youtube.com/watch?v=wgdBVIX9ifA