Professor Hossein Saiedian

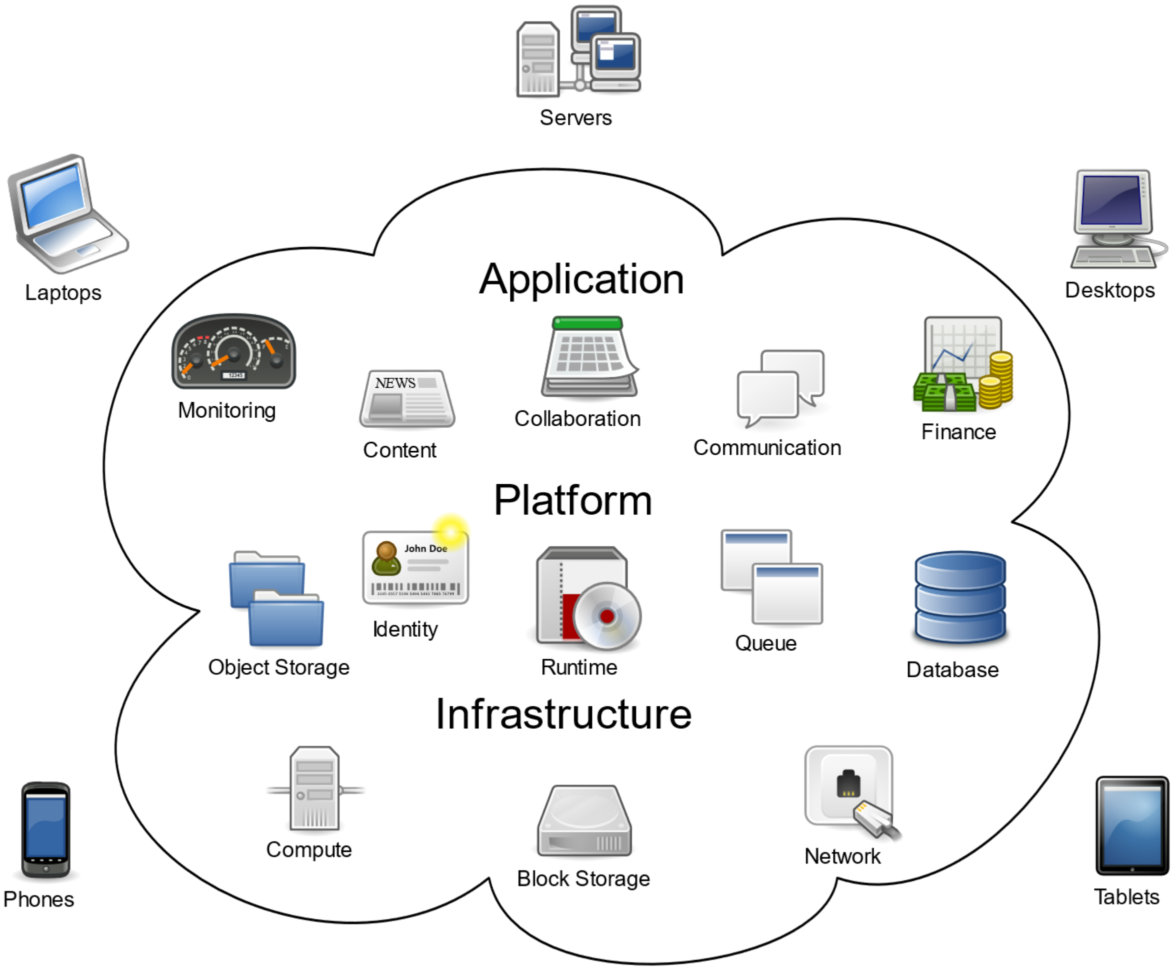
EECS 348: Software Engineering

Spring 2023

KU THE UNIVERSITY OF KANSAS

Cloud computing

- On-demand availability and delivery of computing services
  - Servers
  - Storage
  - Databases
  - Networking
  - Software
  - Analytics



Various things as a service

■ User managed 🟦 Provider managed

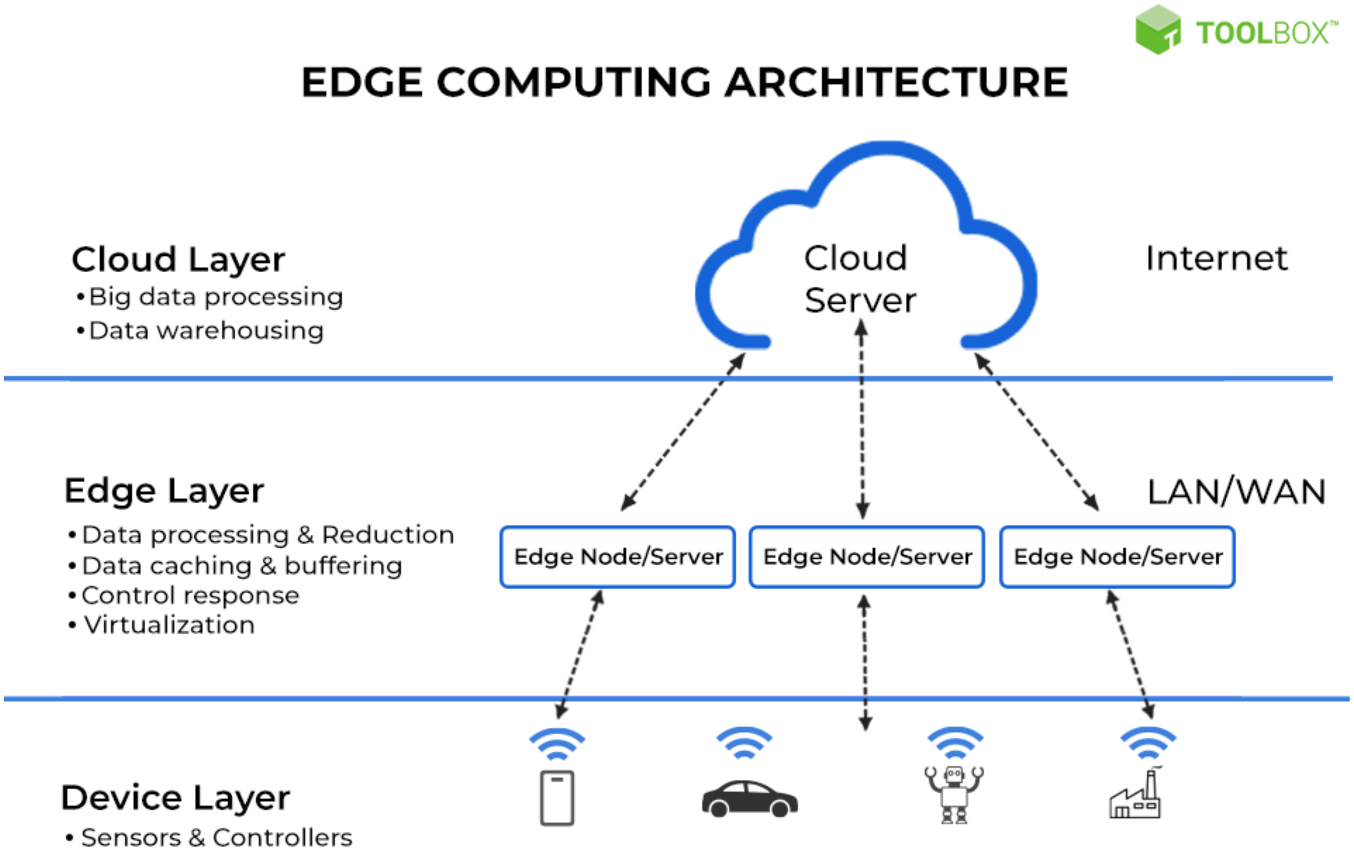
On premises	IaaS	PaaS	SaaS
■ Application	■ Application	■ Application	🟦 Application
■ Data	■ Data	■ Data	🟦 Data
■ Runtime	■ Runtime	🟦 Runtime	🟦 Runtime
■ Middleware	■ Middleware	🟦 Middleware	🟦 Middleware
■ Operating system	■ Operating system	🟦 Operating system	🟦 Operating system
■ Virtualization	🟦 Virtualization	🟦 Virtualization	🟦 Virtualization
■ Networking	🟦 Networking	🟦 Networking	🟦 Networking
■ Storage	🟦 Storage	🟦 Storage	🟦 Storage
■ Servers	🟦 Servers	🟦 Servers	🟦 Servers

Cloud computing related concepts

- On-premise computing
  - Local servers, routers, printers, …
- Edge computing
- Fog computing

Edge computing

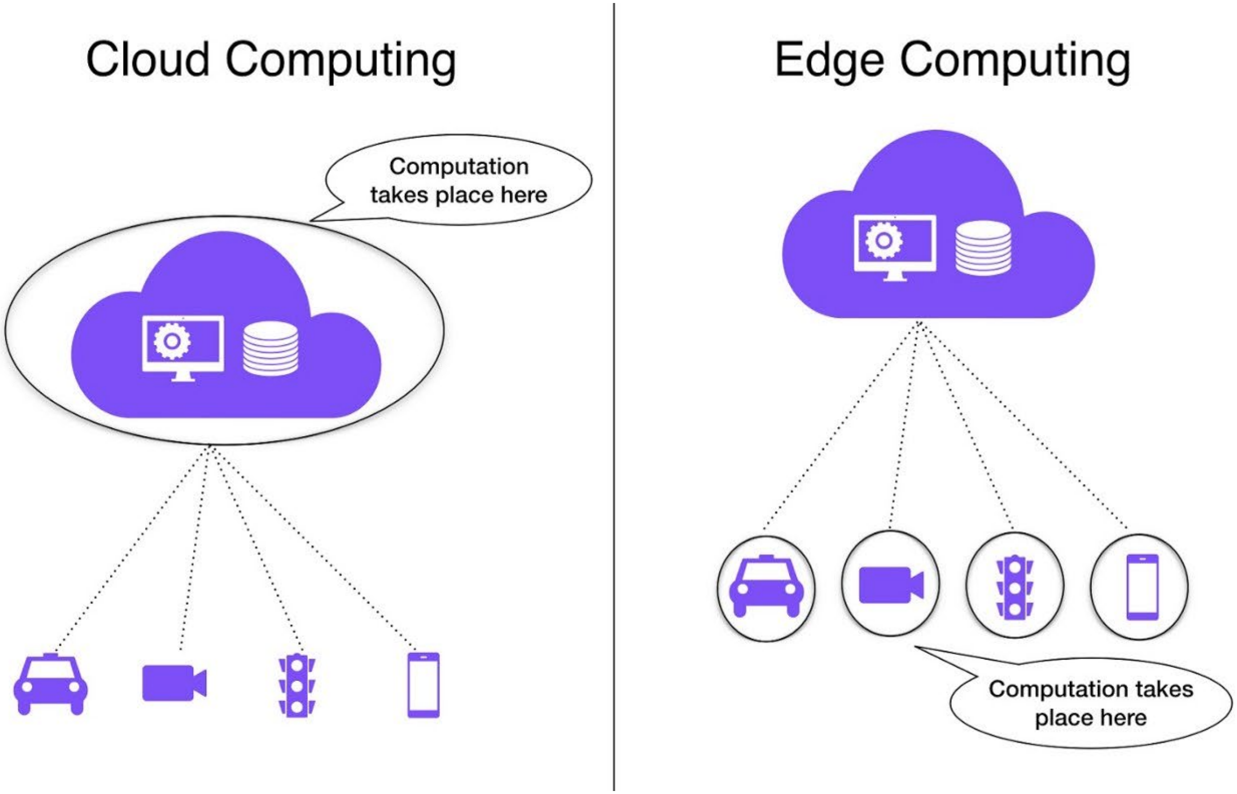
- Related concepts: edge computing



What is edge computing

- Edge is about processing data closer to where it’s being generated, enabling processing at greater speeds and volumes, leading to greater action-led results in real-time
- Collect and process data, share timely insights and if applicable, take appropriate action
  - Examples
  - Wearable on your wrist to the computers parsing intersection traffic flow
  - IoT devices
  - Smart utility grid analysis

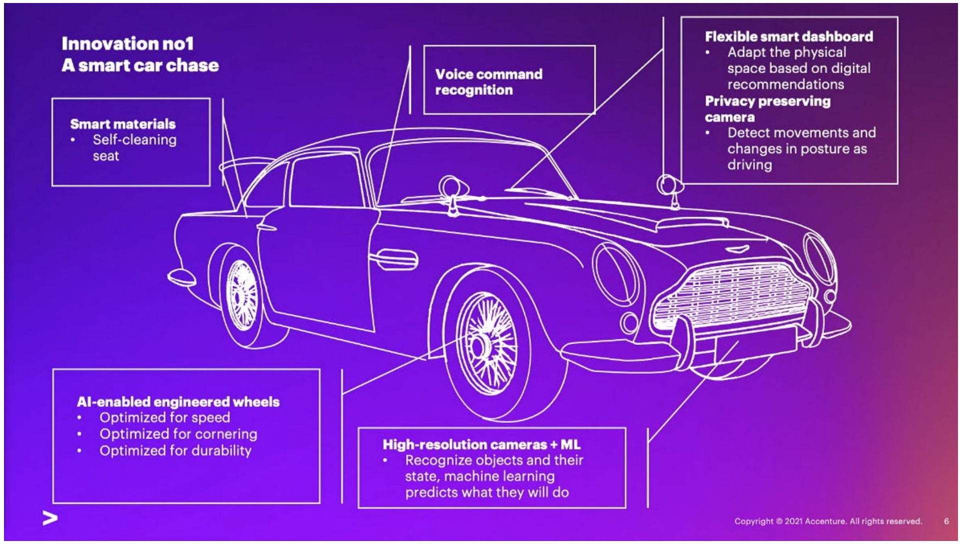
Edge computing vs cloud computing



Edge computing: another view

- Edge computing is transforming how data generated by *billions* of IoT and other devices is stored, processed, analyzed, and transported
  - Before/instead of moving to the cloud
  - Running fewer processes in the cloud
- The practice of moving computing power physically closer to where data is generated, usually an IoT device or sensor
- Internet of Medical Things (IoMT)
  - Monitor glucose levels, blood pressure levels

Examples of edge computing



Innovation no1
A smart car chase

Smart materials
- Self-cleaning seat

Voice command recognition

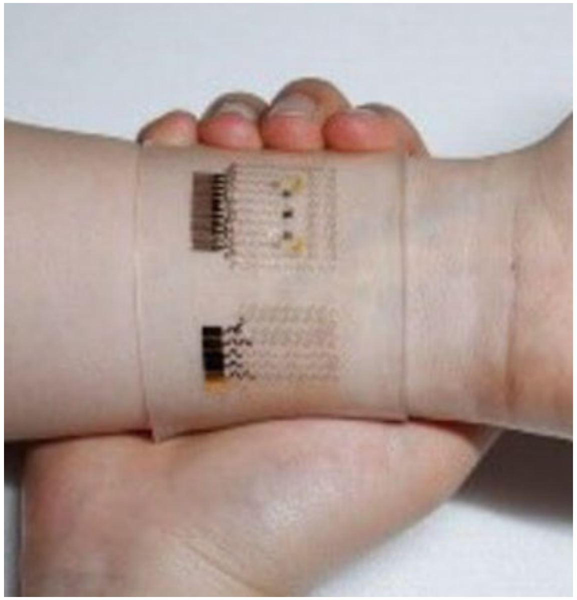
Flexible smart dashboard
- Adapt the physical space based on digital recommendations

Privacy preserving camera
- Detect movements and changes in posture as driving

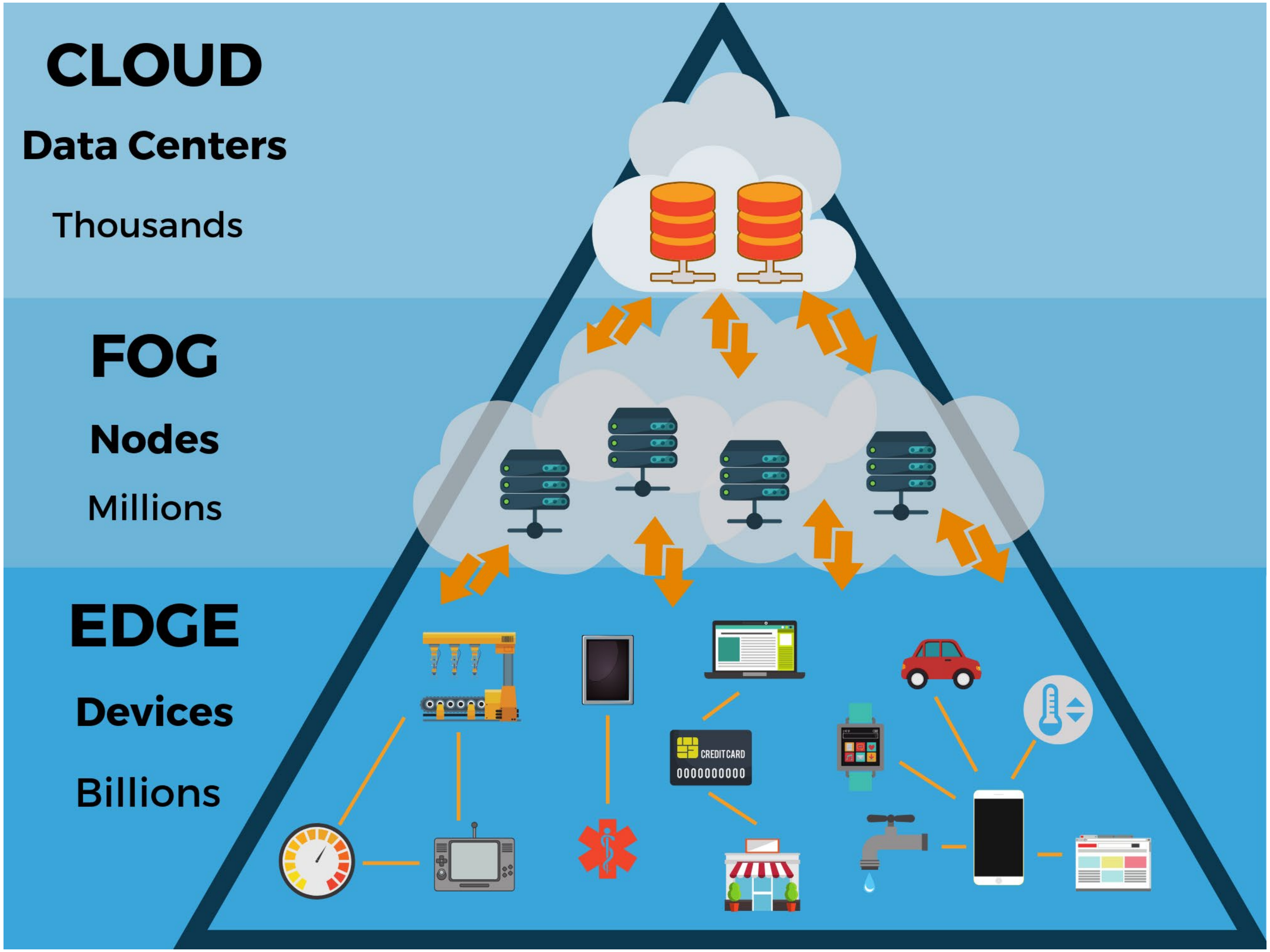
AI-enabled engineered wheels
- Optimized for speed
- Optimized for cornering
- Optimized for durability

High-resolution cameras + ML
- Recognize objects and their state, machine learning predicts what they will do

Copyright © 2021 Accenture. All rights reserved. 6

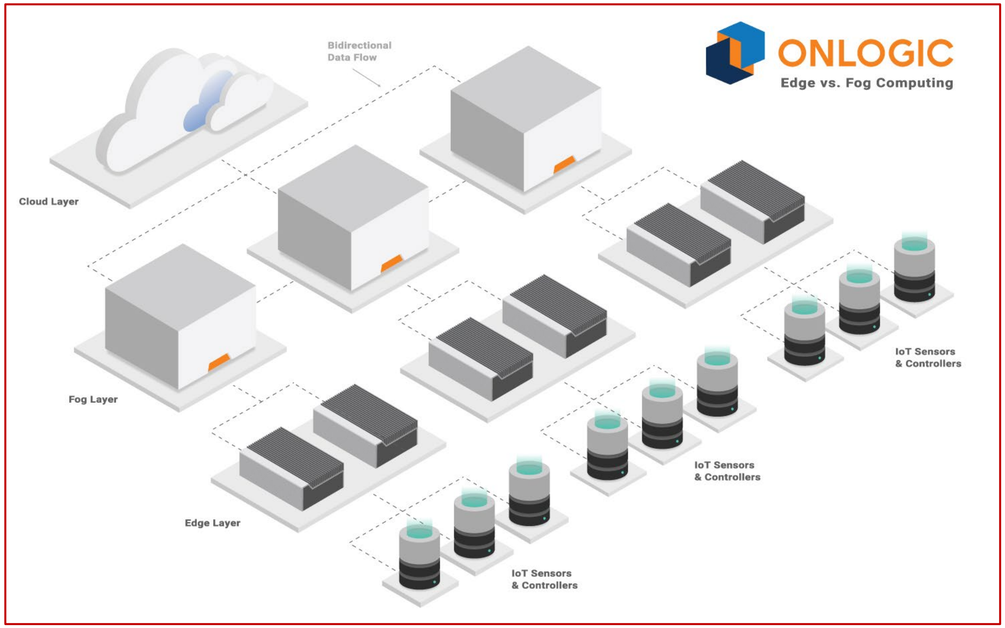


Fog computing



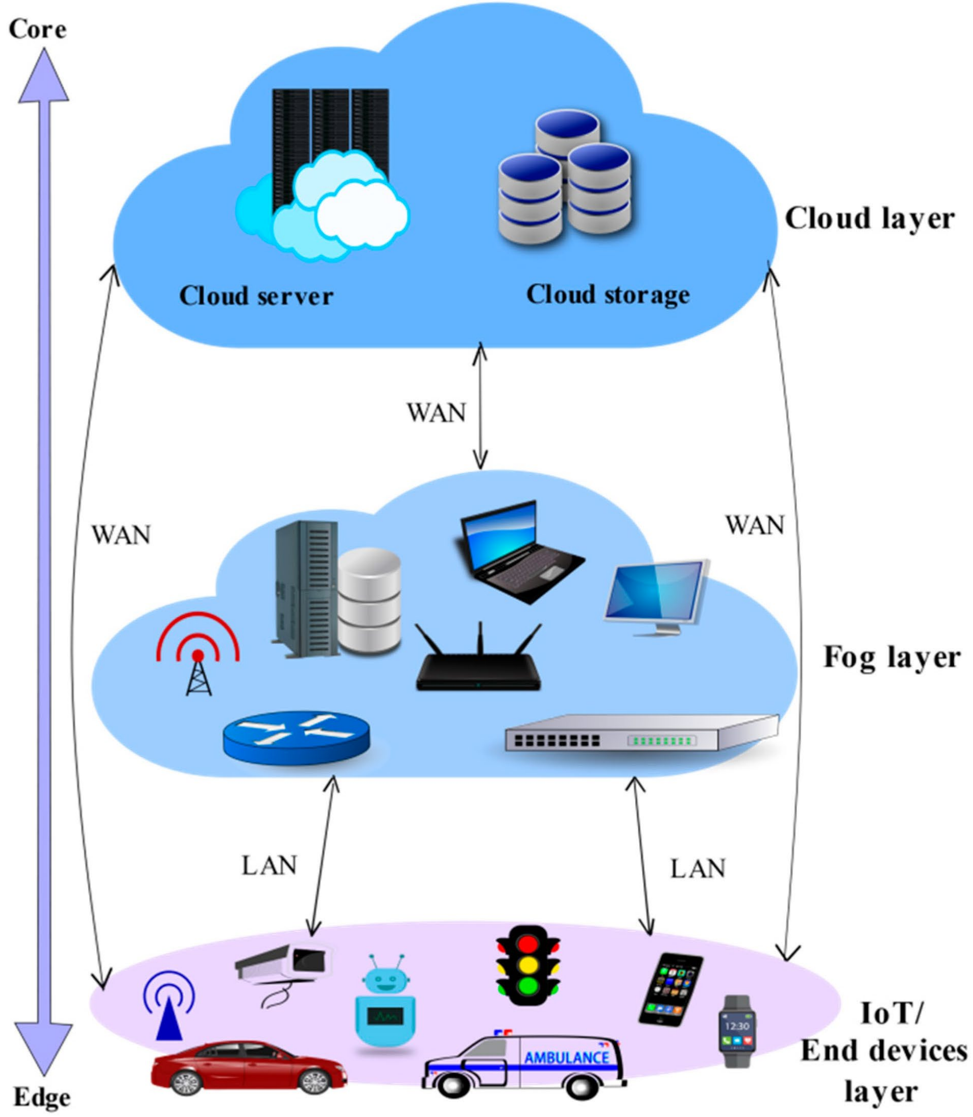
Fog computing benefits

- A compute layer between the edge and cloud
- Receives data from the edge before it reaches the cloud
- Benefits
  - Enables **low-latency** networking connections between devices
  - Minimizes **bandwidth** requirements compared to if that data had to be transferred back to a data center or cloud for analysis



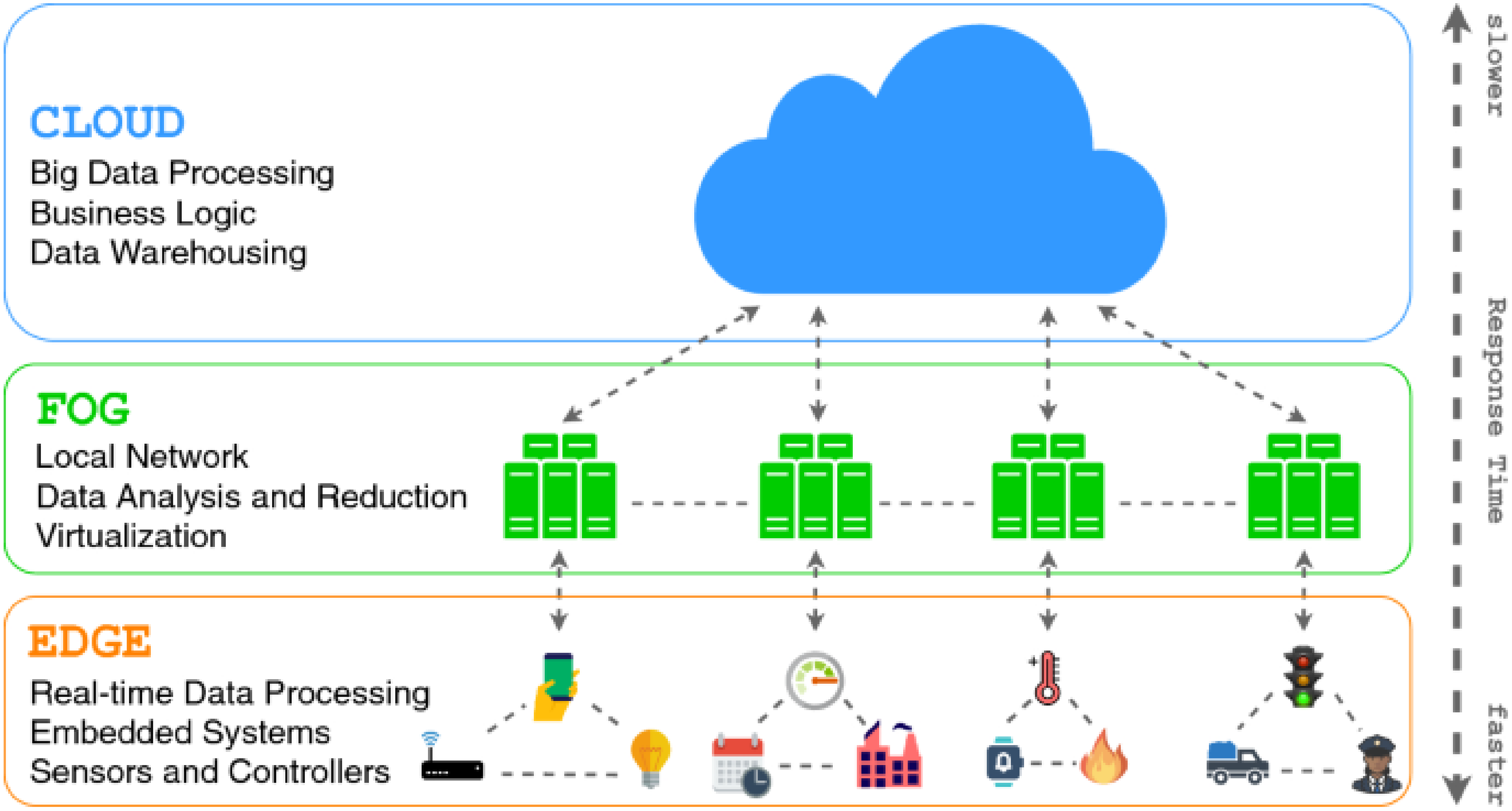
Edge, fog, cloud spectrum

- Additional illustration



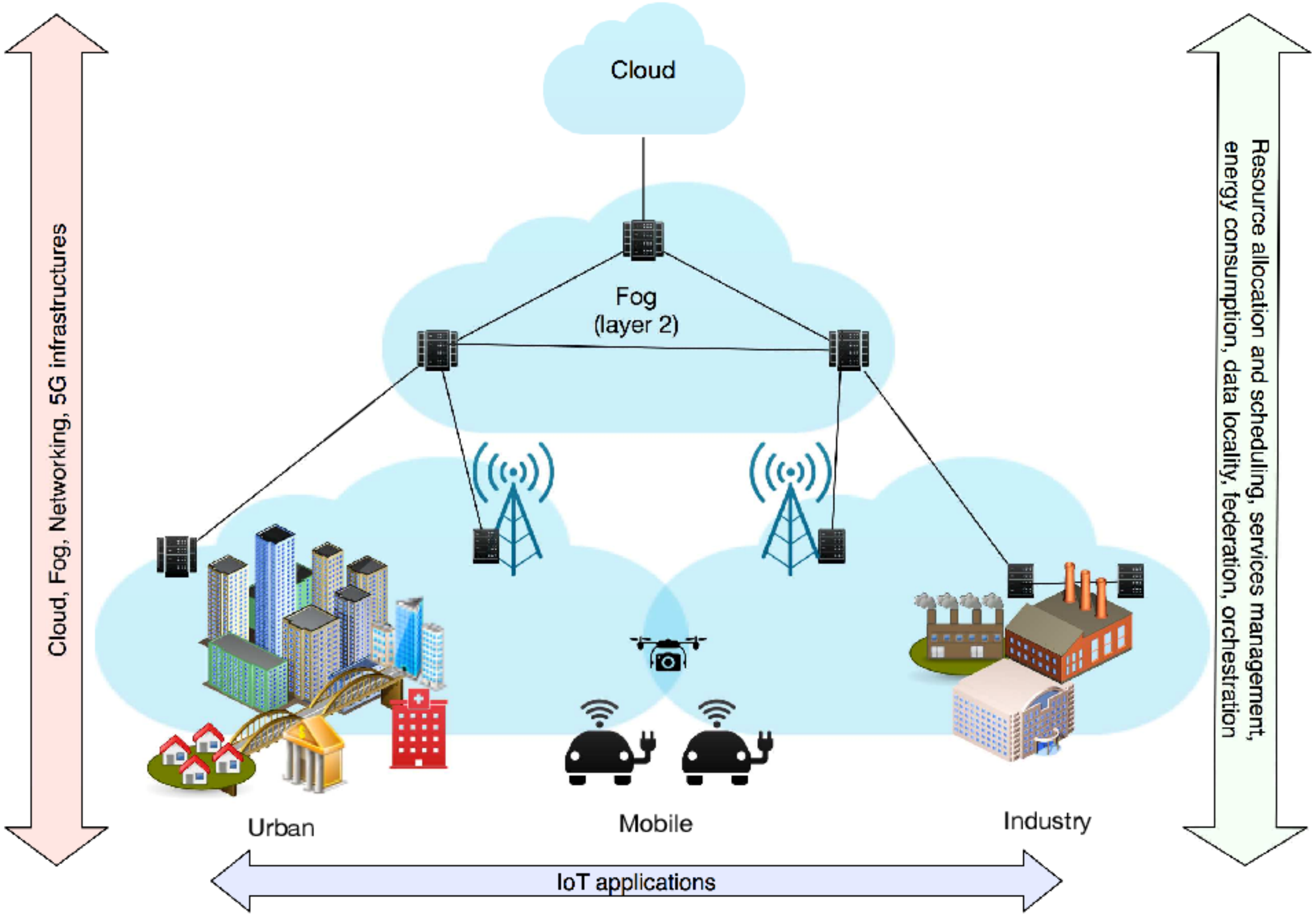
Edge, fog, cloud spectrum

- Additional illustration



Edge, fog, cloud spectrum

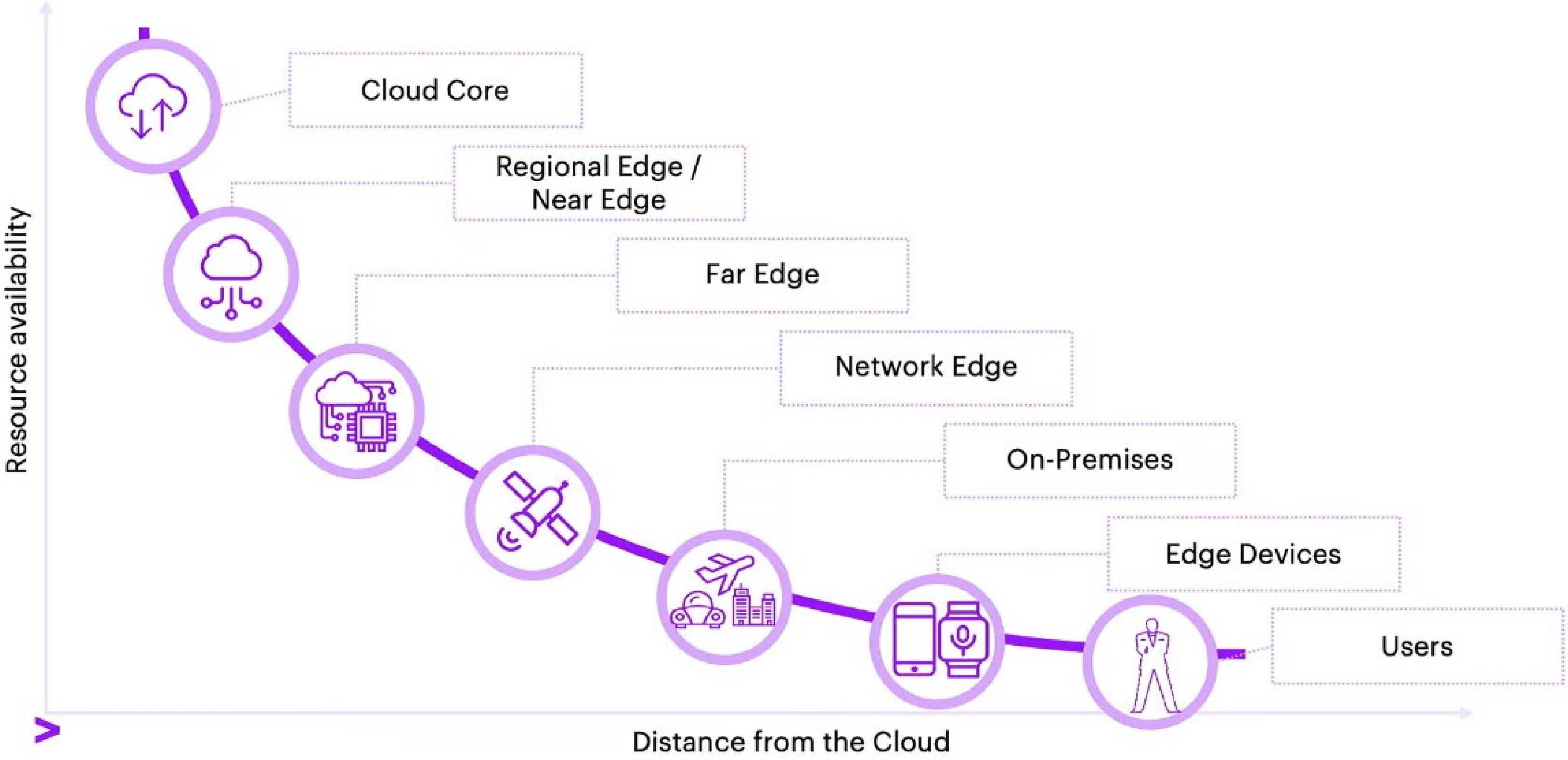
- Additional illustration



Cloud
Fog
(layer 2)
Urban
Mobile
Industry
IoT applications
Cloud, Fog, Networking, 5G infrastructures
Resource allocation and scheduling, services management, energy consumption, data locality, federation, orchestration

Distance from the user to the cloud

- Cloud computing continuum

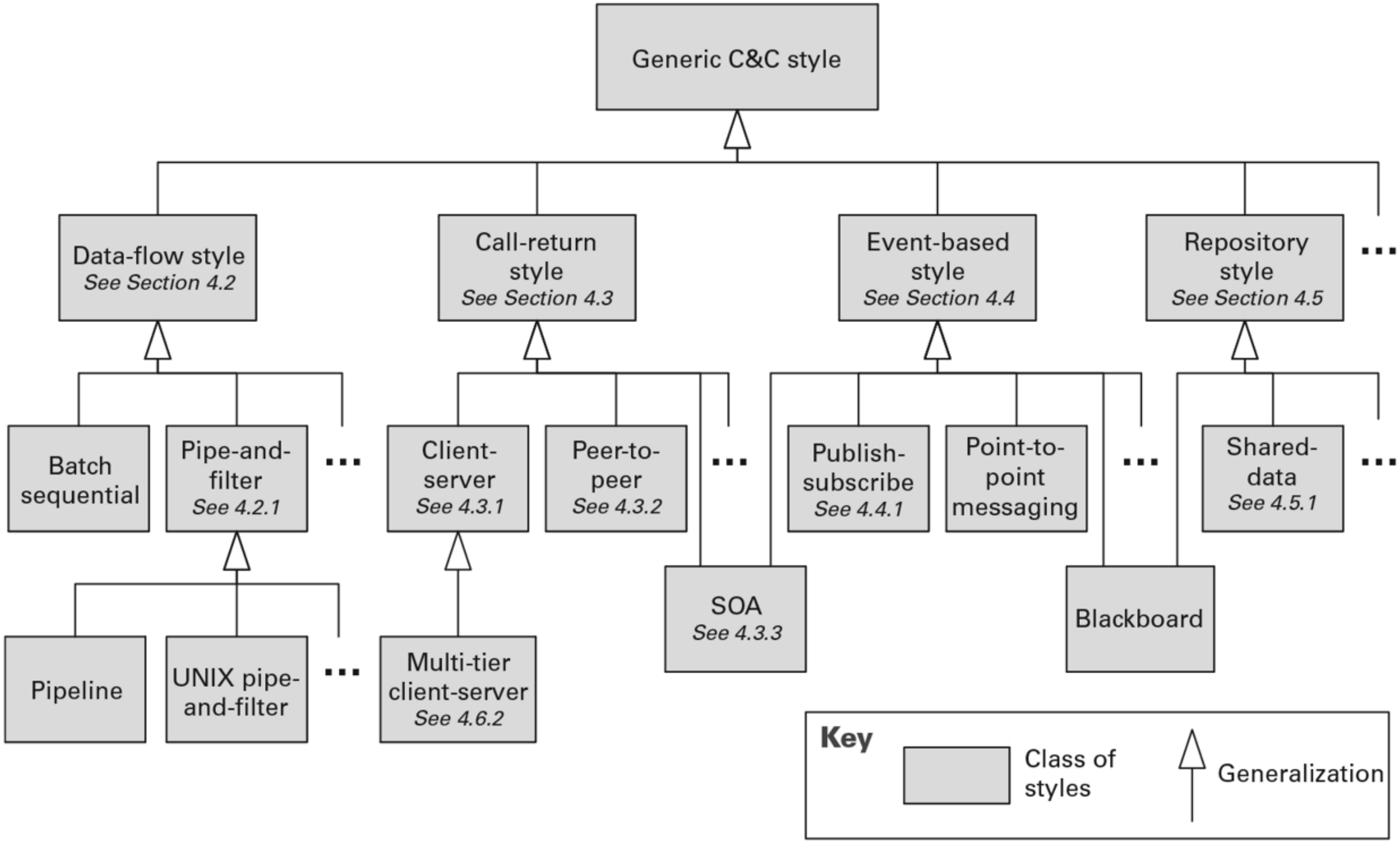


Introduction to microservices

- Microservices is the evolution of service-oriented architecture
- What is service-oriented architecture (SOA)?
  - An architectural style that focuses on discrete services instead of a monolithic design
  - Each service in an SOA embodies the code and data required to execute a complete
    - * Example: checking a customer’s credit

Service-oriented architecture (SOA)

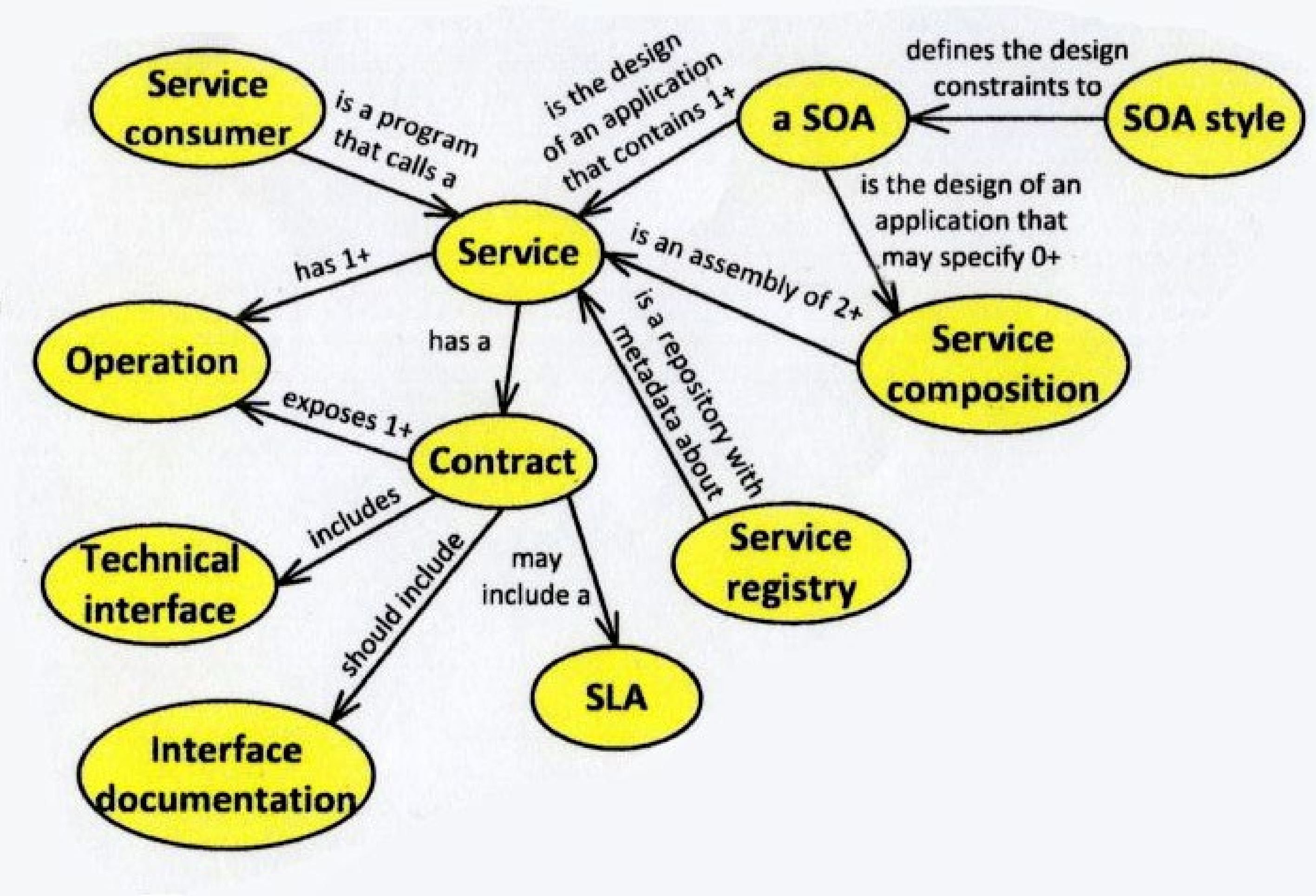
- The SOA architecture style is based on other styles



Software services

- A software service is a software component that can be accessed from remote computers over the Internet
  - Given an input, a service produces a corresponding output, without side effects
  - The service is accessed through its published interface and all details of the service implementation are hidden
  - Services do not maintain any internal state
    - State information is either stored in a database or is maintained by the service requestor

A service properties



Software services

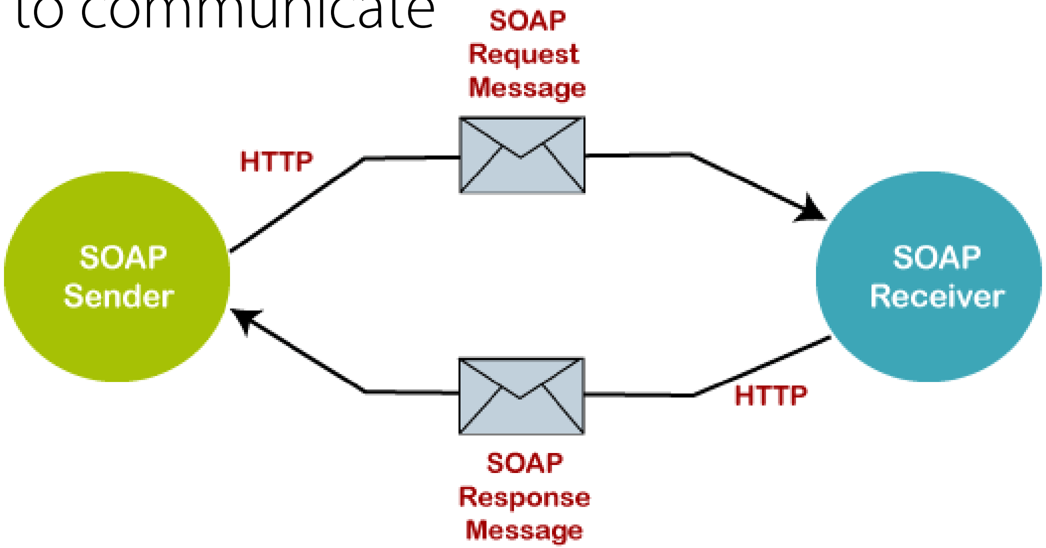
- Services do not maintain any internal state
- When a service request is made, the state information may be included as part of the request, and the updated state information is returned as part of the service result
- As there is no local state, services can be dynamically reallocated from one virtual server to another and replicated across several servers

Related terminology

- SOAP
- XML
- JSON
- WSDL
- RESTful services
- API
- API gateway

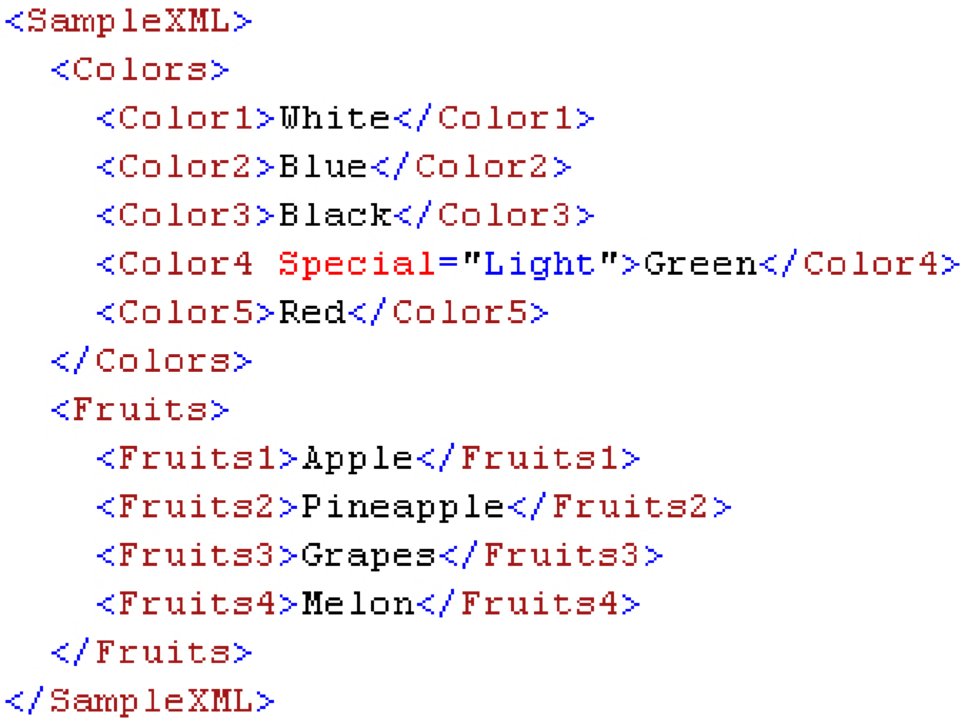
Related terminology

- Web services were initially thought of as implementations of traditional software components that could be distributed over a network
- SOAP (Simple Object Access Protocol)
  - A message protocol that enables the distributed elements of an application to communicate



XML

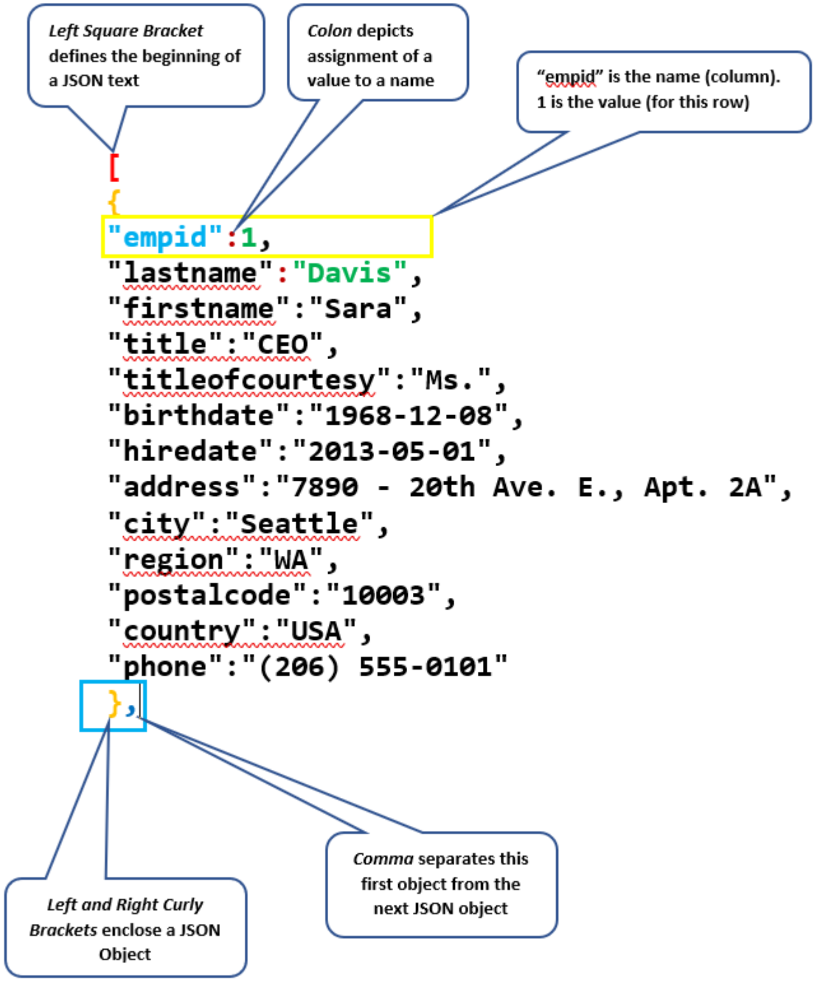
- XML (eXtensible Markup Language)
  - A markup language much like HTML
  - Used for structuring arbitrary data



```
<SampleXML>
  <Colors>
    <Color1>White</Color1>
    <Color2>Blue</Color2>
    <Color3>Black</Color3>
    <Color4 Special="Light">Green</Color4>
    <Color5>Red</Color5>
  </Colors>
  <Fruits>
    <Fruits1>Apple</Fruits1>
    <Fruits2>Pineapple</Fruits2>
    <Fruits3>Grapes</Fruits3>
    <Fruits4>Melon</Fruits4>
  </Fruits>
</SampleXML>
```

JSON

- JSON (JavaScript Object Notation)
  - A lightweight format for storing and transporting data
  - An open standard
  - A common format for electronic data interchange



Left Square Bracket defines the beginning of a JSON text

Colon depicts assignment of a value to a name

"empid" is the name (column). 1 is the value (for this row)

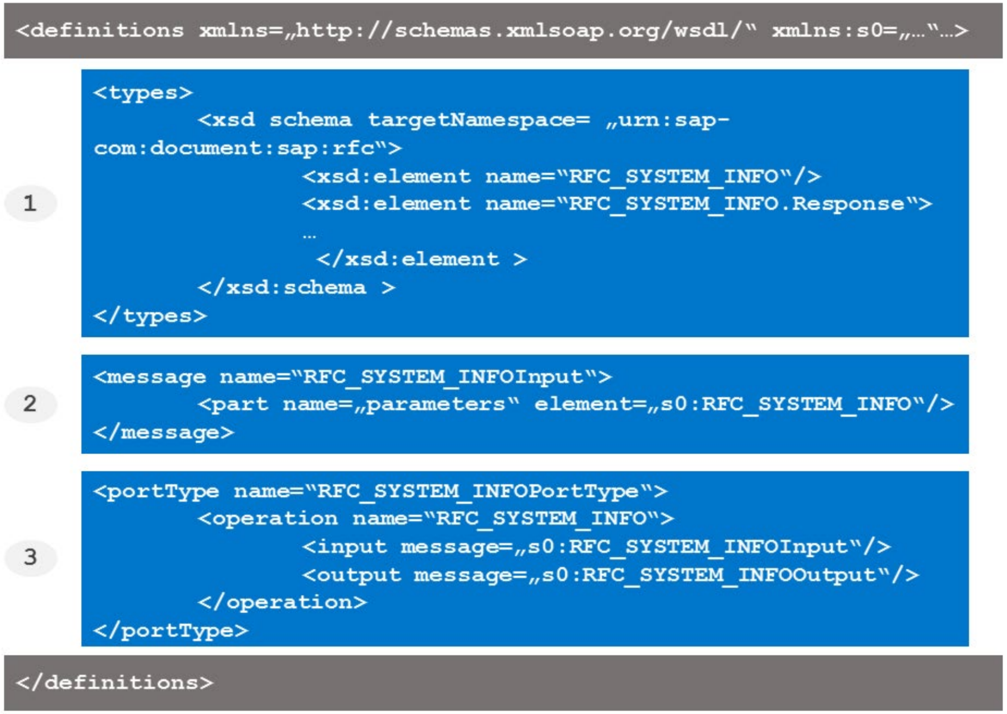
```
[
{
"empid":1,
"lastname":"Davis",
"firstname":"Sara",
"title":"CEO",
"titleofcourtesy":"Ms.",
"birthdate":"1968-12-08",
"hiredate":"2013-05-01",
"address":"7890 - 20th Ave. E., Apt. 2A",
"city":"Seattle",
"region":"WA",
"postalcode":"10003",
"country":"USA",
"phone":"(206) 555-0101"
},
```

Left and Right Curly Brackets enclose a JSON Object

Comma separates this first object from the next JSON object

WSDL

- WSDL (Web Service Description Language) is an XML based definition language
- It is used to describe functionality of a Web service



```
<definitions xmlns=„http://schemas.xmlsoap.org/wsdl/‟ xmlns:s0=„…‟…>
```

1

```
<types>
        <xsd schema targetNamespace= „urn:sap-
com:document:sap:rfc‟>
                <xsd:element name=‟RFC_SYSTEM_INFO‟/>
                <xsd:element name=‟RFC_SYSTEM_INFO.Response‟>
                …
                 </xsd:element >
        </xsd:schema >
</types>
```

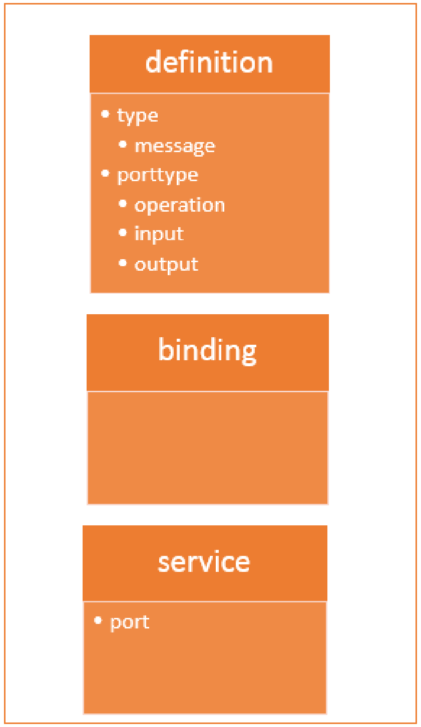
2

```
<message name=‟RFC_SYSTEM_INFOInput‟>
        <part name=„parameters‟ element=„s0:RFC_SYSTEM_INFO‟/>
</message>
```

3

```
<portType name=‟RFC_SYSTEM_INFOPortType‟>
        <operation name=‟RFC_SYSTEM_INFO‟>
                <input message=„s0:RFC_SYSTEM_INFOInput‟/>
                <output message=„s0:RFC_SYSTEM_INFOOutput‟/>
        </operation>
</portType>
```

```
</definitions>
```



definition

- type
  - message
- porttype
  - operation
  - input
  - output

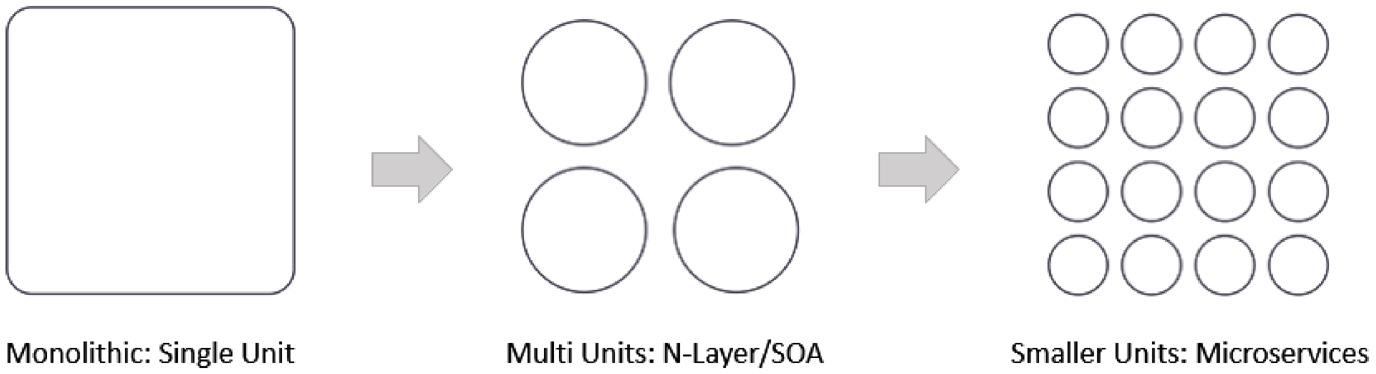
binding

service

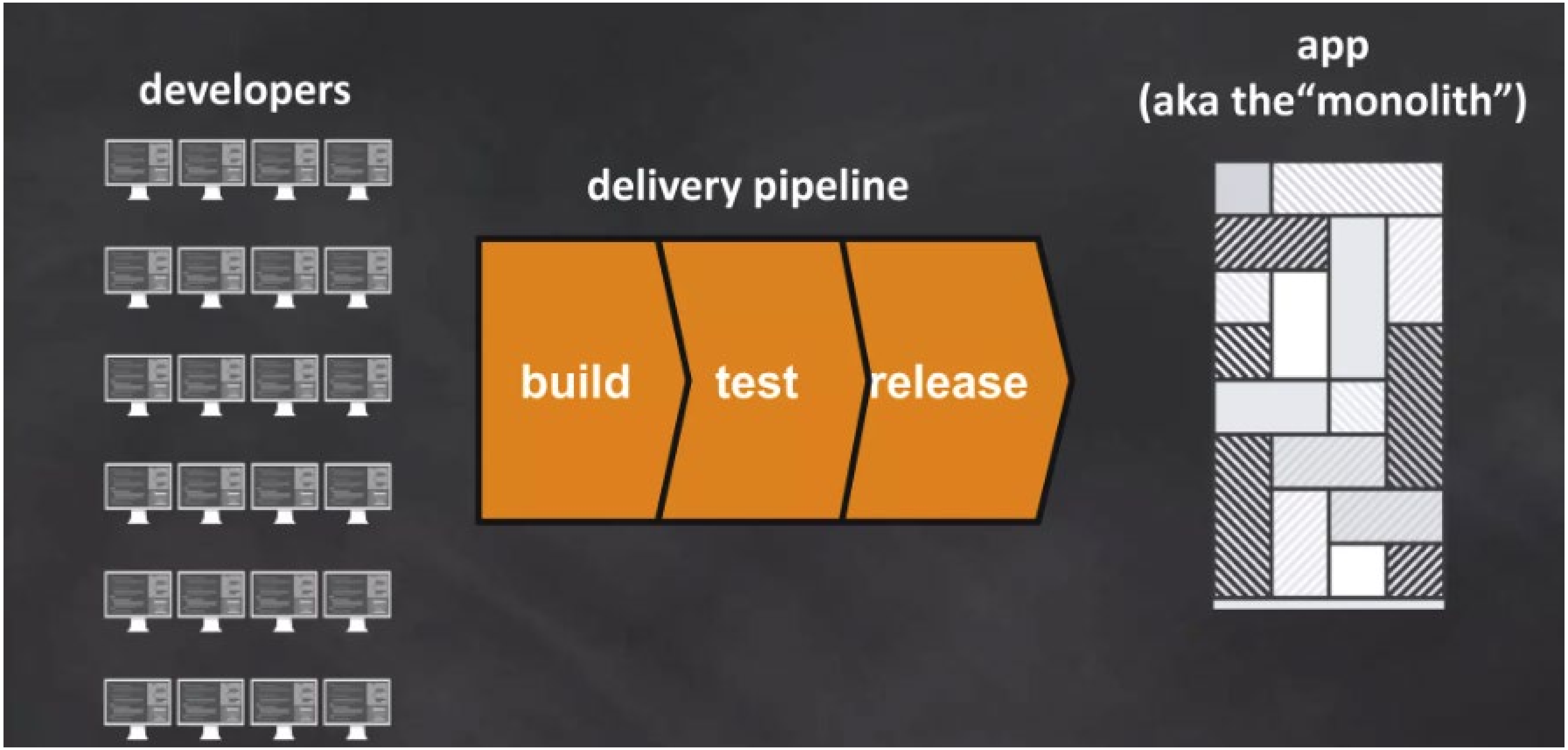
- port

Why microservices

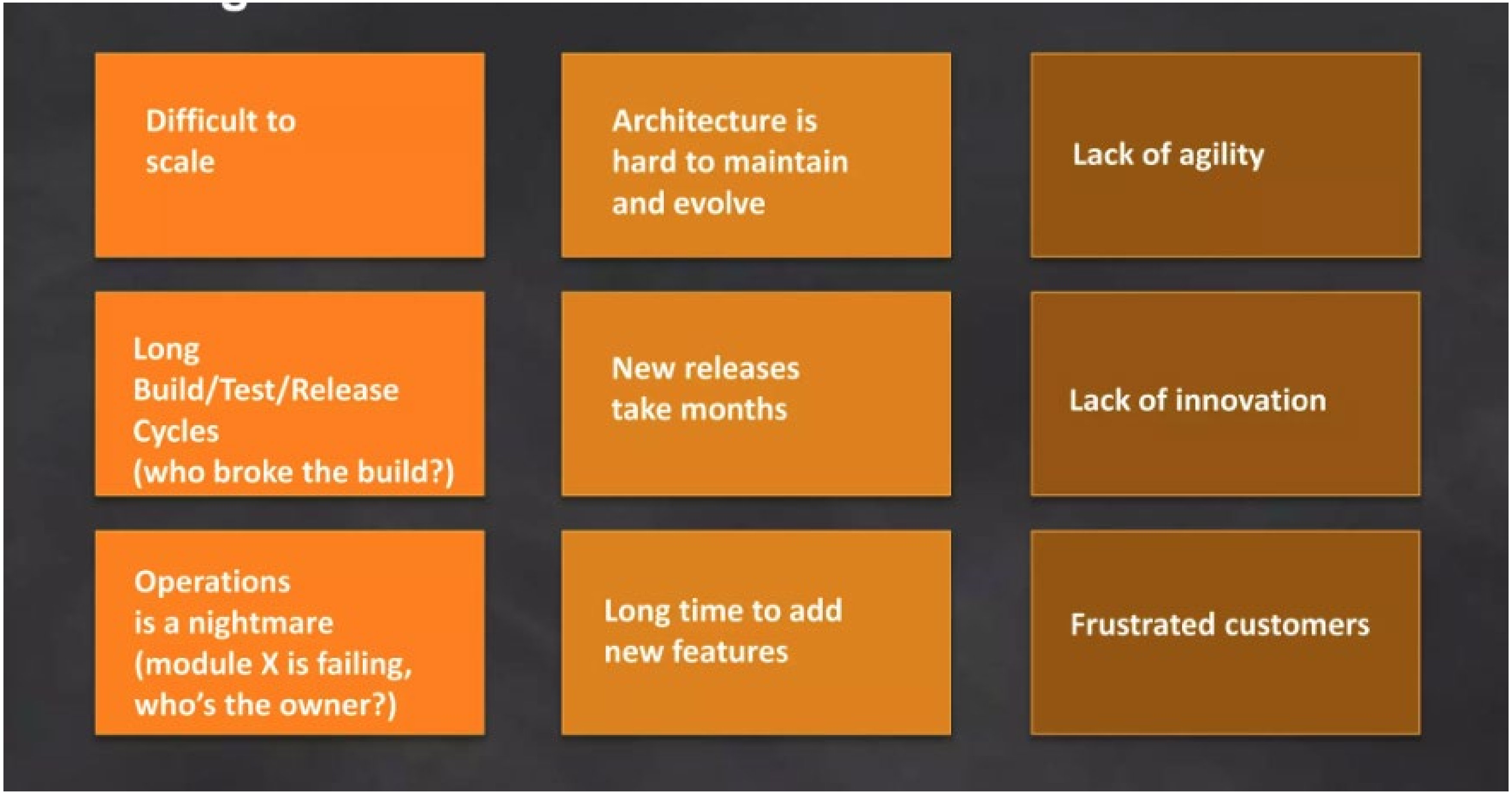
- Microservices are small and independent services that work together
  - Decentralized
  - Deploy independently
  - Modeled around a business domain
  - Isolate failure
  - Hide internal details (reduce coupling)



Monolith software development lifecycle



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

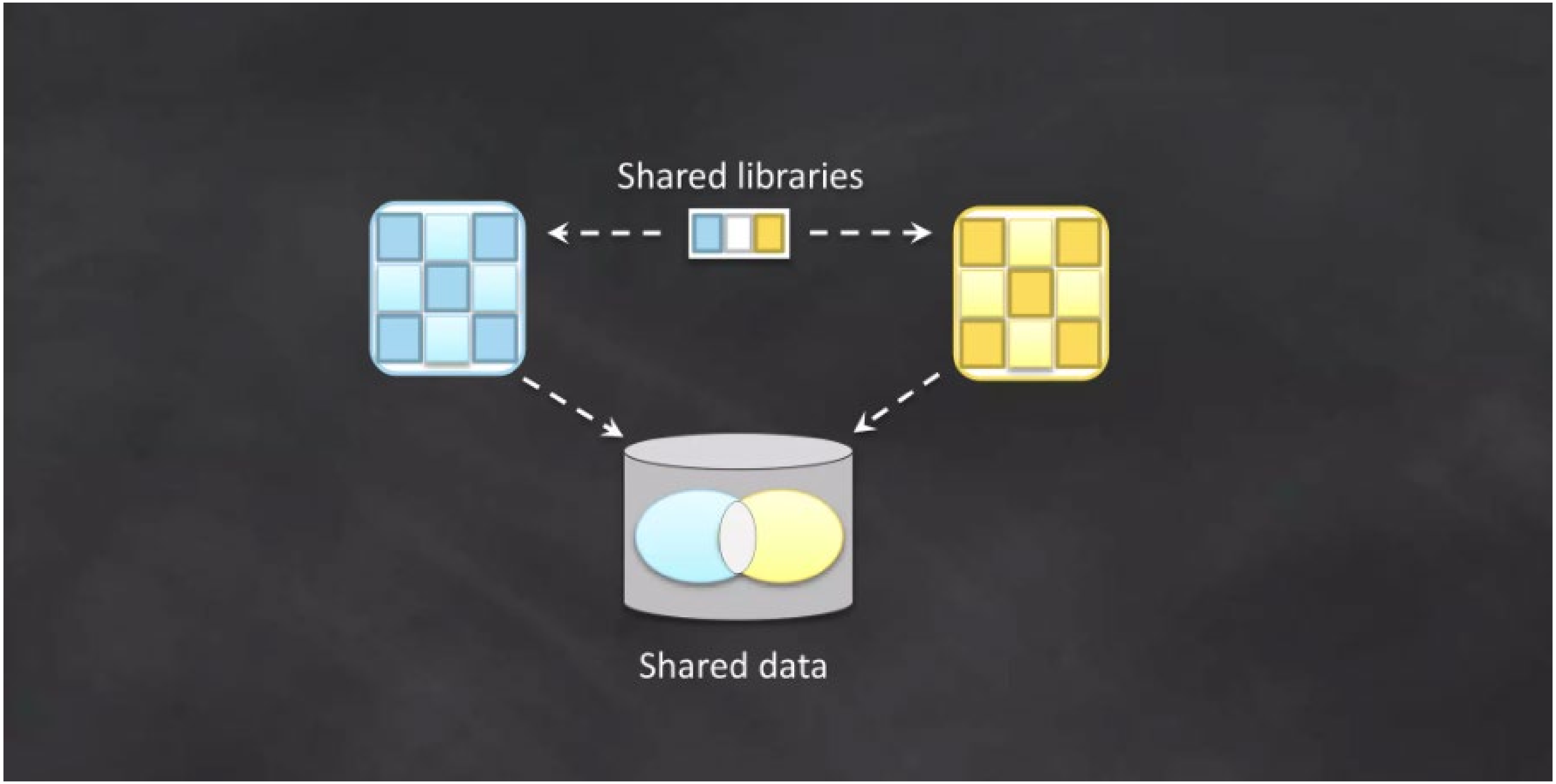
Lack of innovation

Operations is a nightmare (module X is failing, who's the owner?)

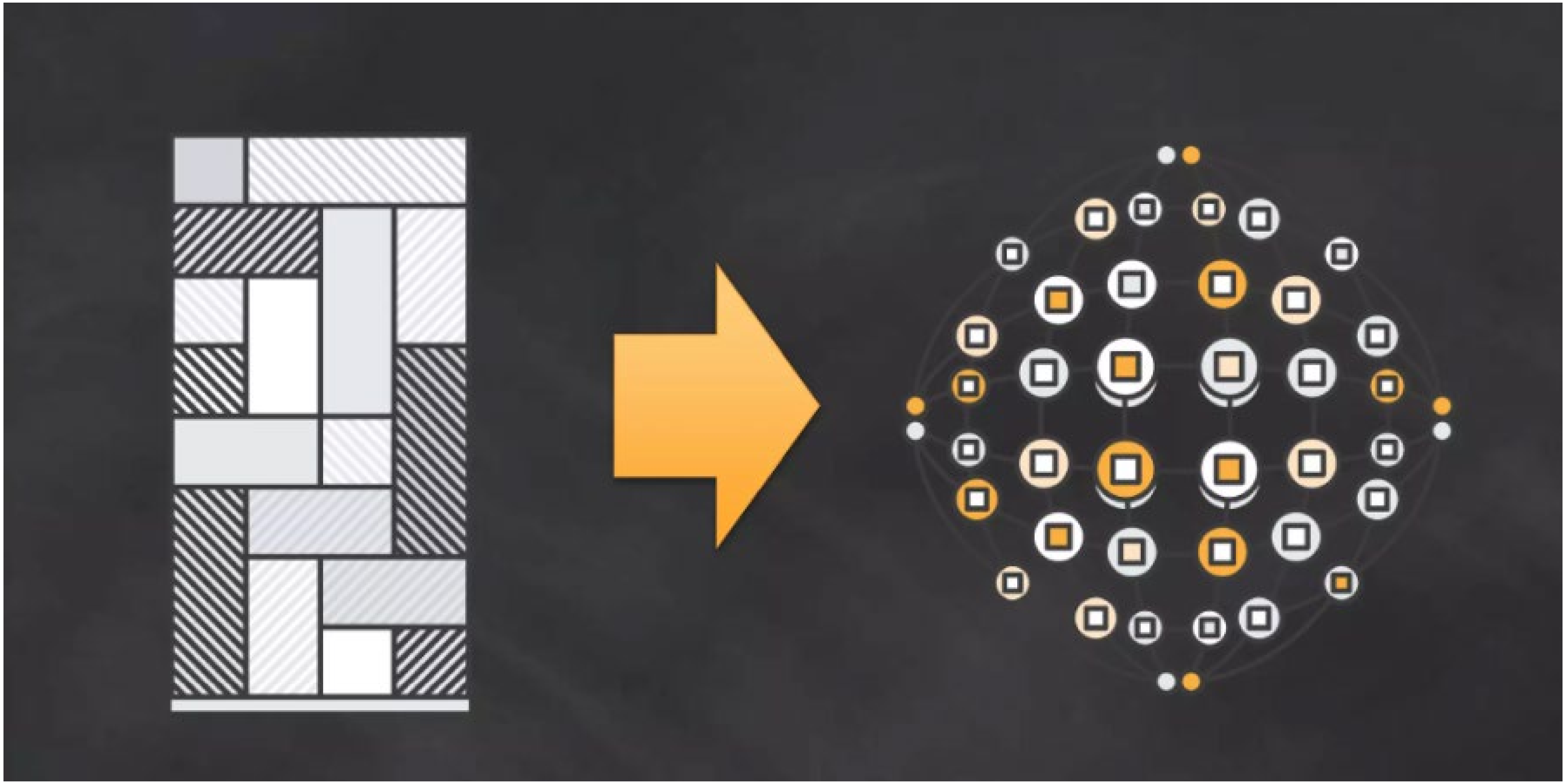
Long time to add new features

Frustrated customers

Too much component coupling

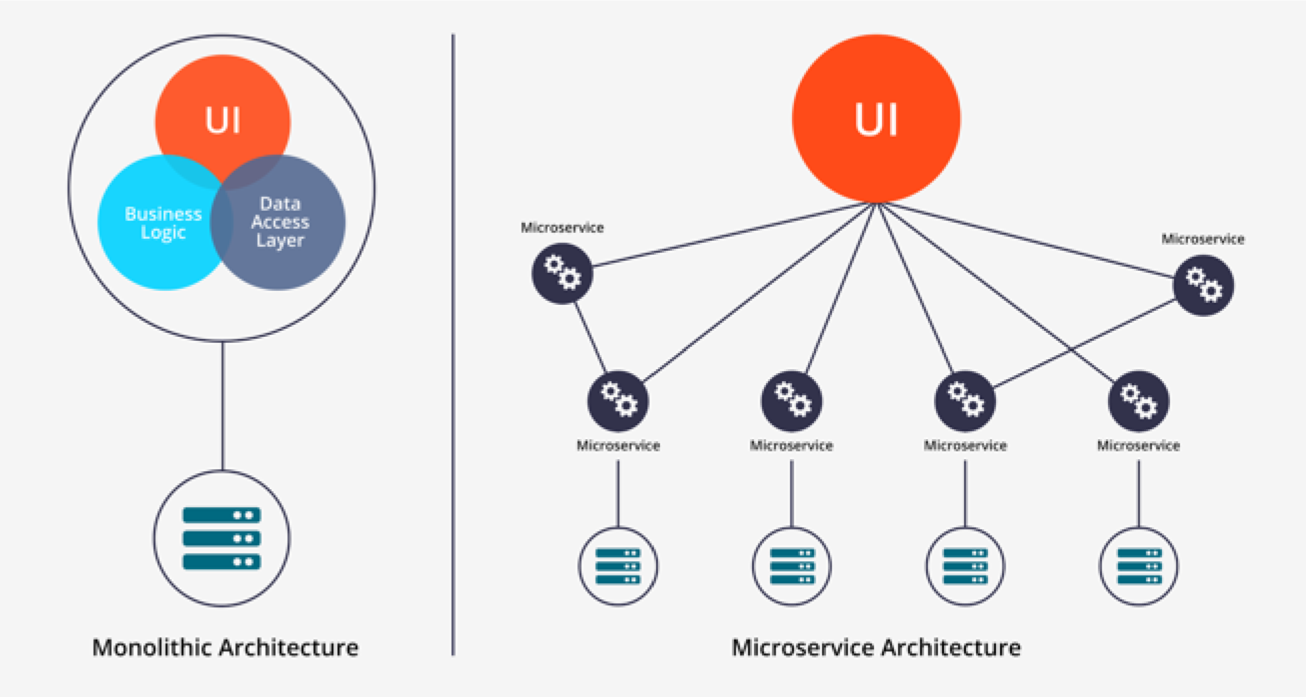


Microservices idea



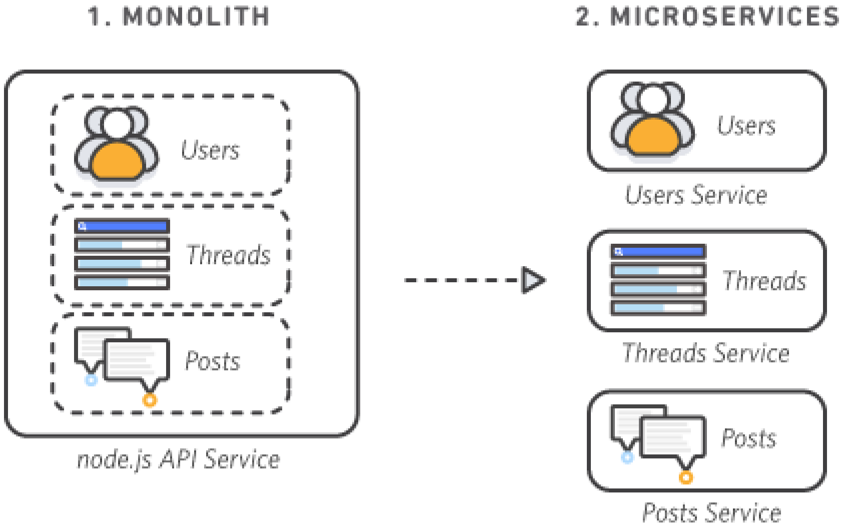
Microservices idea

- Monolithic app: everything is built in one app
  - UI components, business logic, … and deployed on one server node



Microservices: definition

- Microservices are an architectural and organizational approach to software development where
  - Software is composed of loosely-coupled small independent services that communicate over well-defined APIs
- Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features



Characteristics of microservices

- **Autonomous**: Each component service in a microservices architecture can be developed, deployed, operated, and scaled without affecting the functioning of other services
  - Any communication between individual components happens via well-defined APIs.
  - Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features

Characteristics of microservices

- **Specialized**: Each service is designed for a set of capabilities and focuses on solving a specific problem
- If developers contribute more code to a service over time and the service becomes complex, it can be broken into smaller services.

Characteristics of microservices

- **Self-contained**: Code can be updated without knowing anything about internals of other microservices

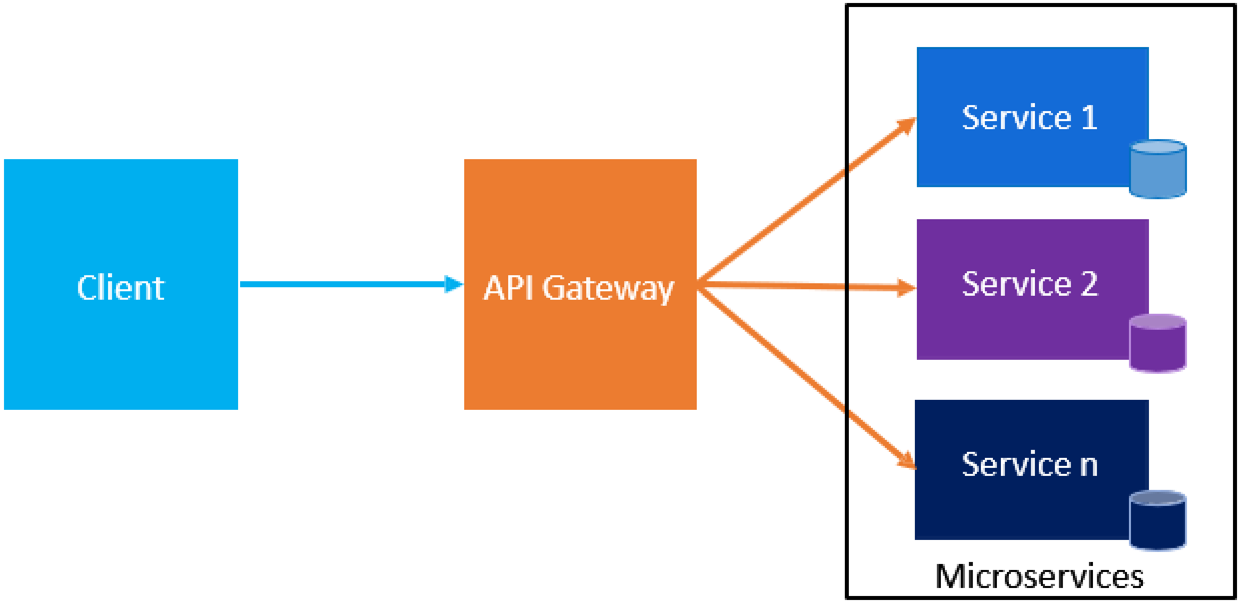
Characteristics of microservices

- **Highly cohesive**
  - Similar to the concept of cohesive modules during the design
  - Cohesive with respect to a business service
  - Do one thing: a business service
- **Loosely coupled**
  - Self-contained: Code can be updated independently without knowing anything about internals of other microservices
  - Updating one service doesn't require changing other services

Microservices: Yet another explanation

- Avoid this
- Do one thing

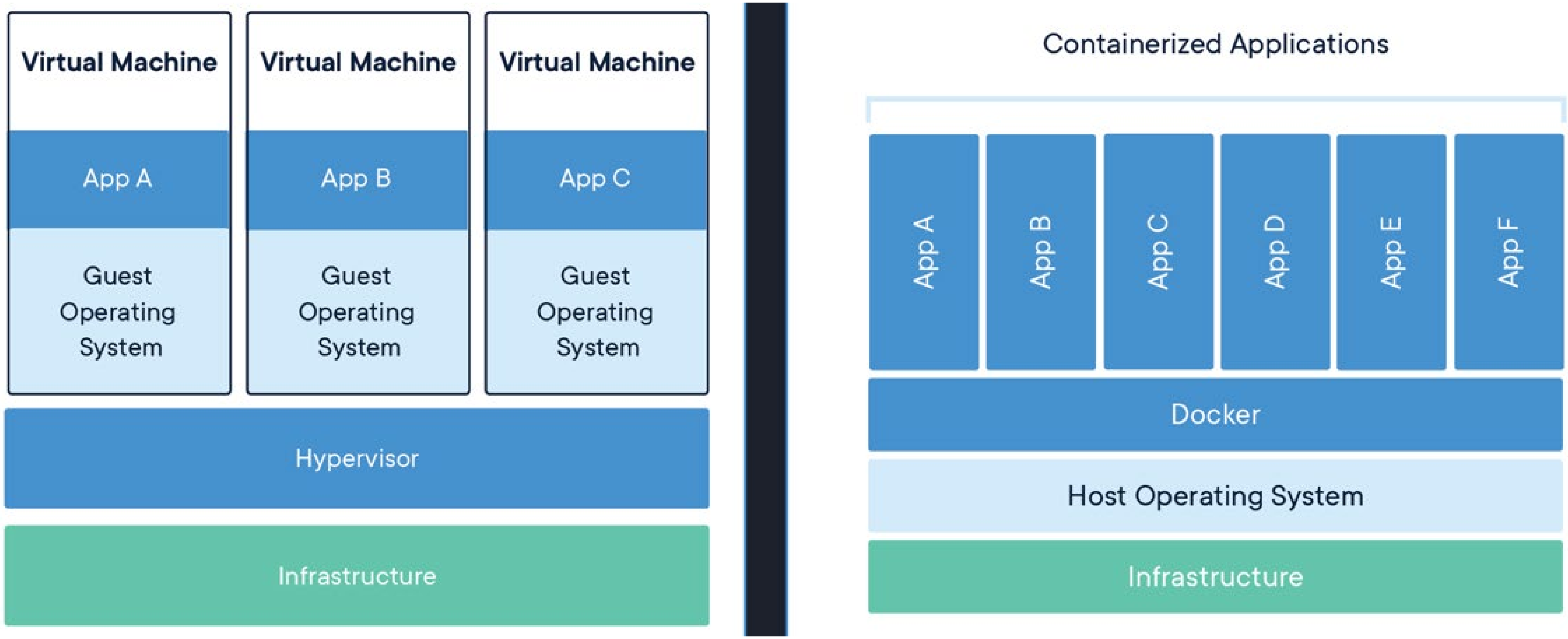
Microservices architecture (oversimplified)



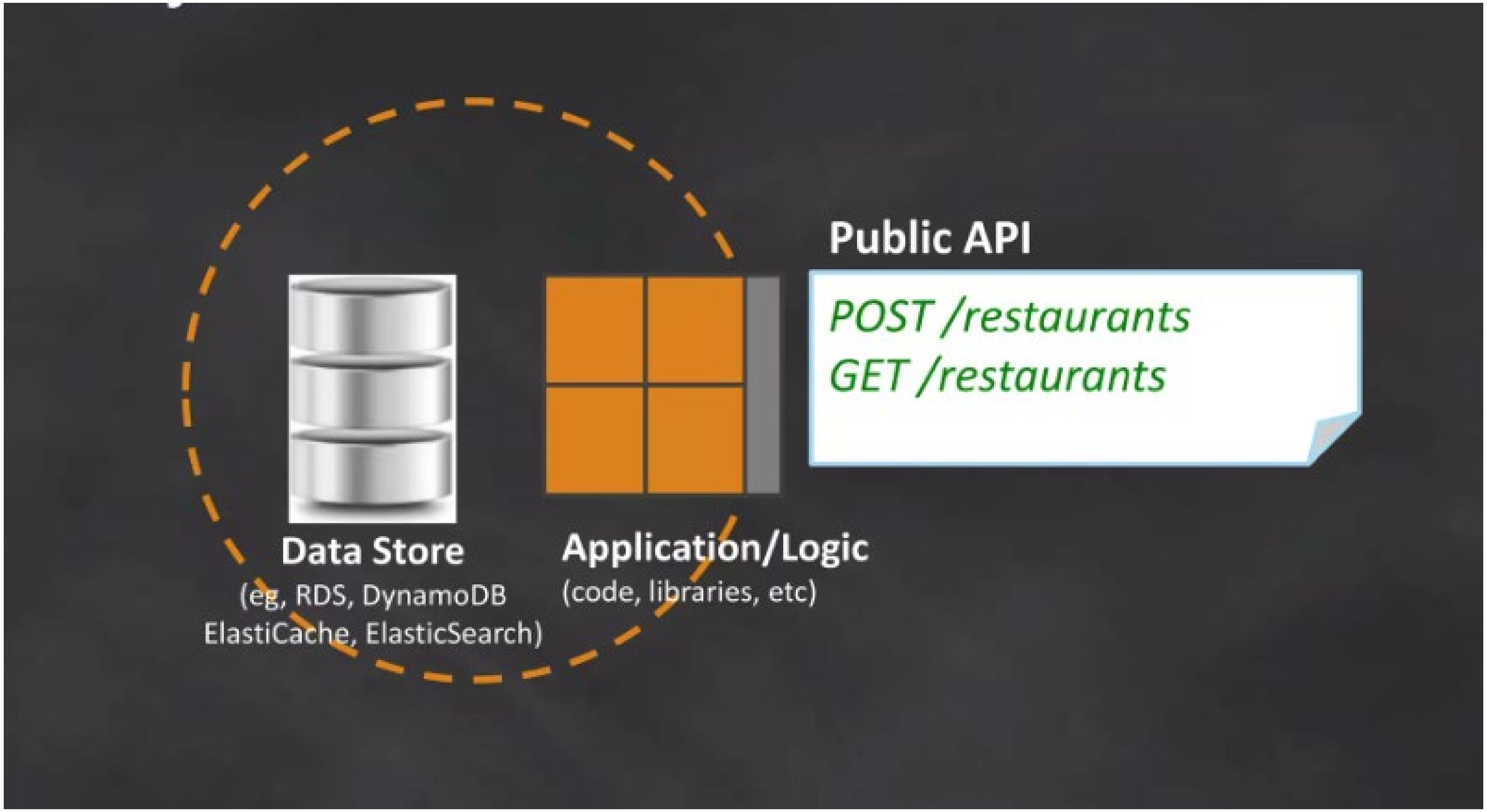
- An API gateway is an API management tool that sits between a client and a collection of services
- API: the way for two programs to communicate
  - An interface specification
  - Based on some standards: SOAP, RPC, REST

Relationship with containers

- Faster deployment
- Flexible scaling
- Lesser resources

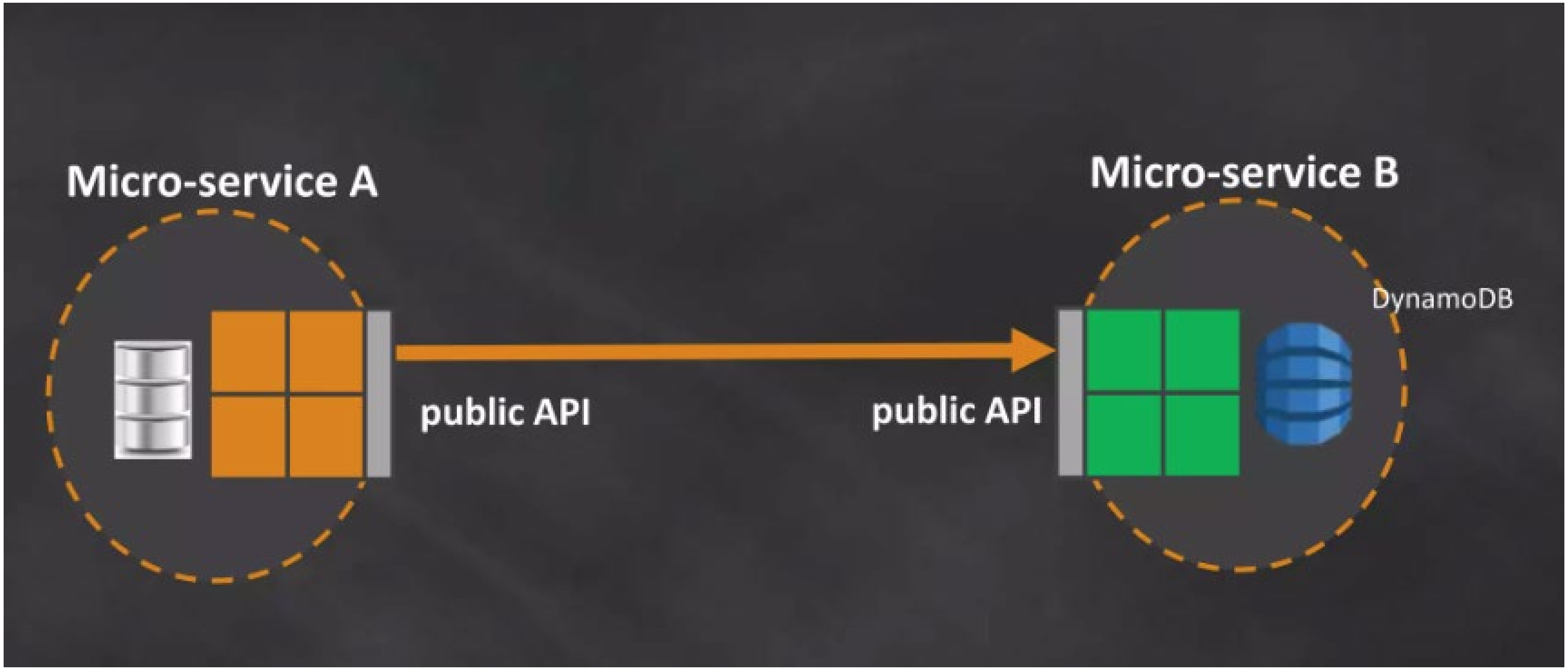


Anatomy of a microservice



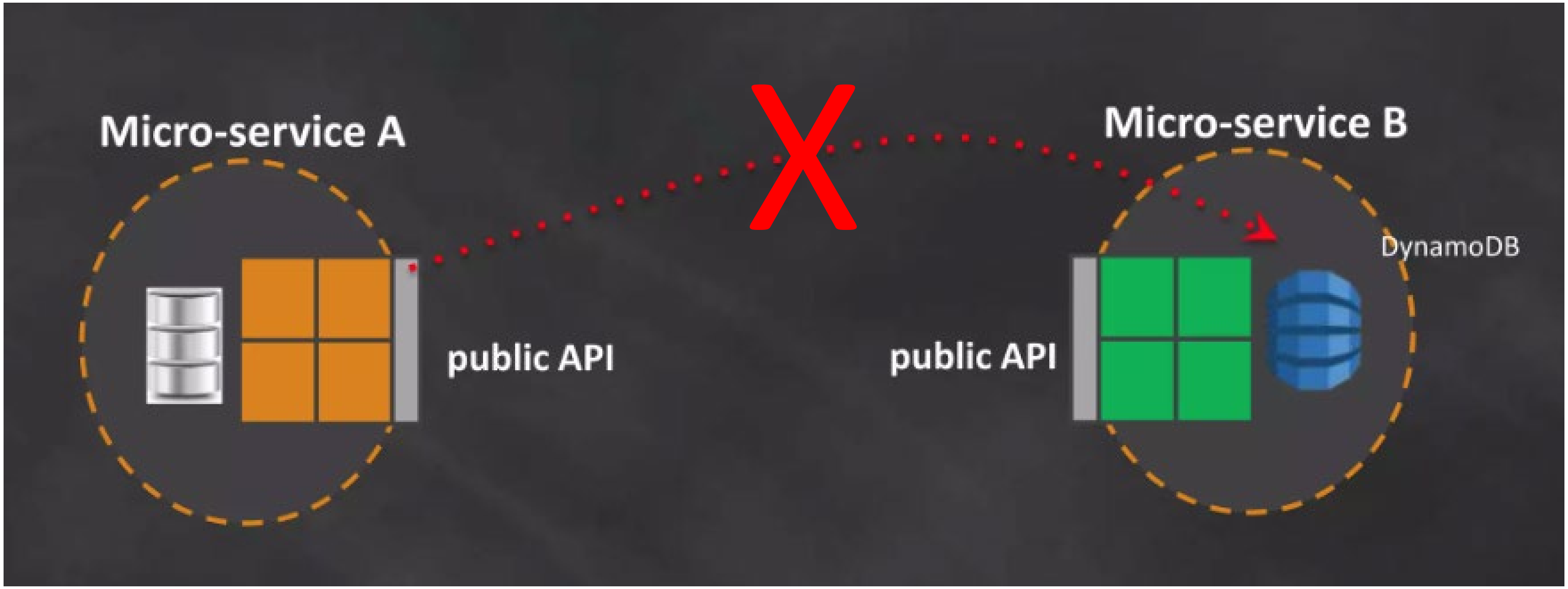
Rely on APIs

- Microservices only rely on each other’s public API



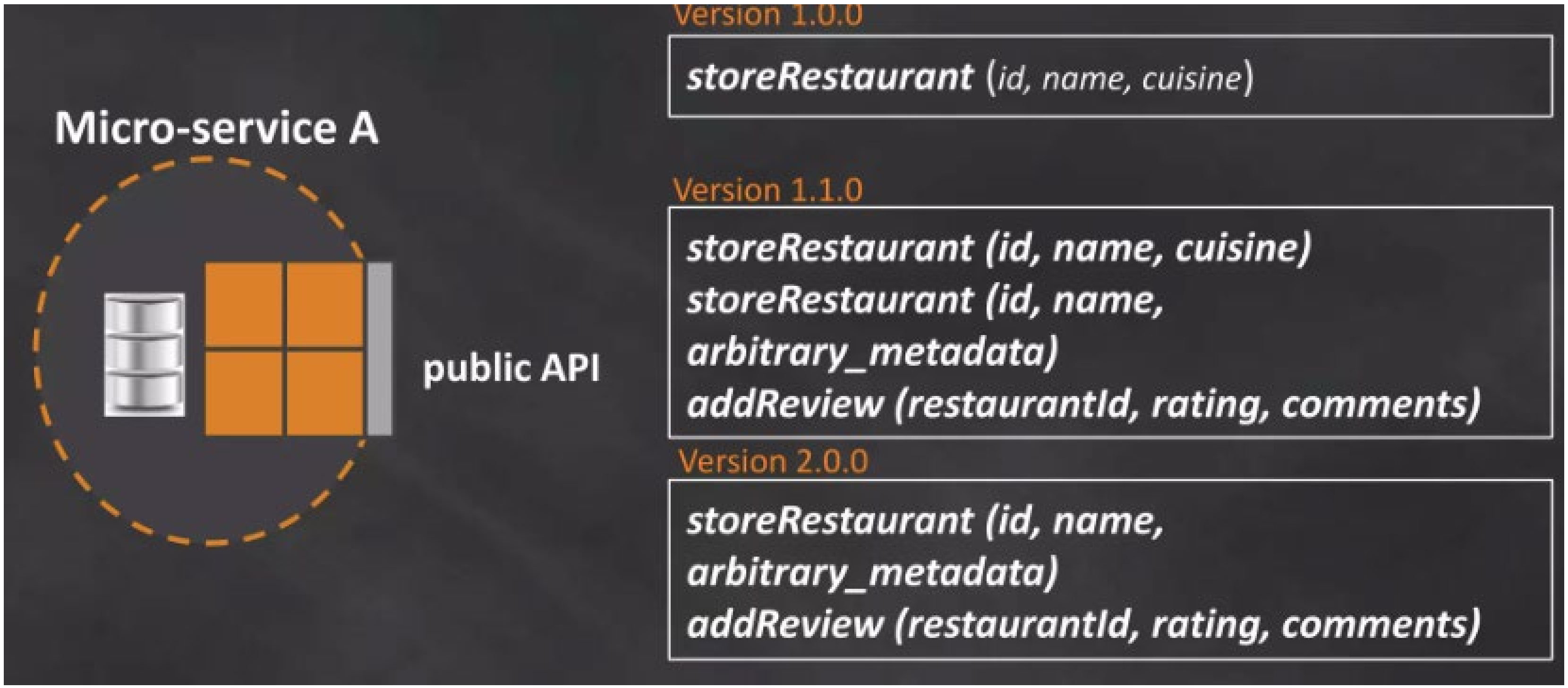
Avoid component coupling

- Communicate only thru a well-defined API



Avoid component coupling

- API interfaces may evolve but must remain compatible



Version 1.0.0

storeRestaurant (id, name, cuisine)

Version 1.1.0

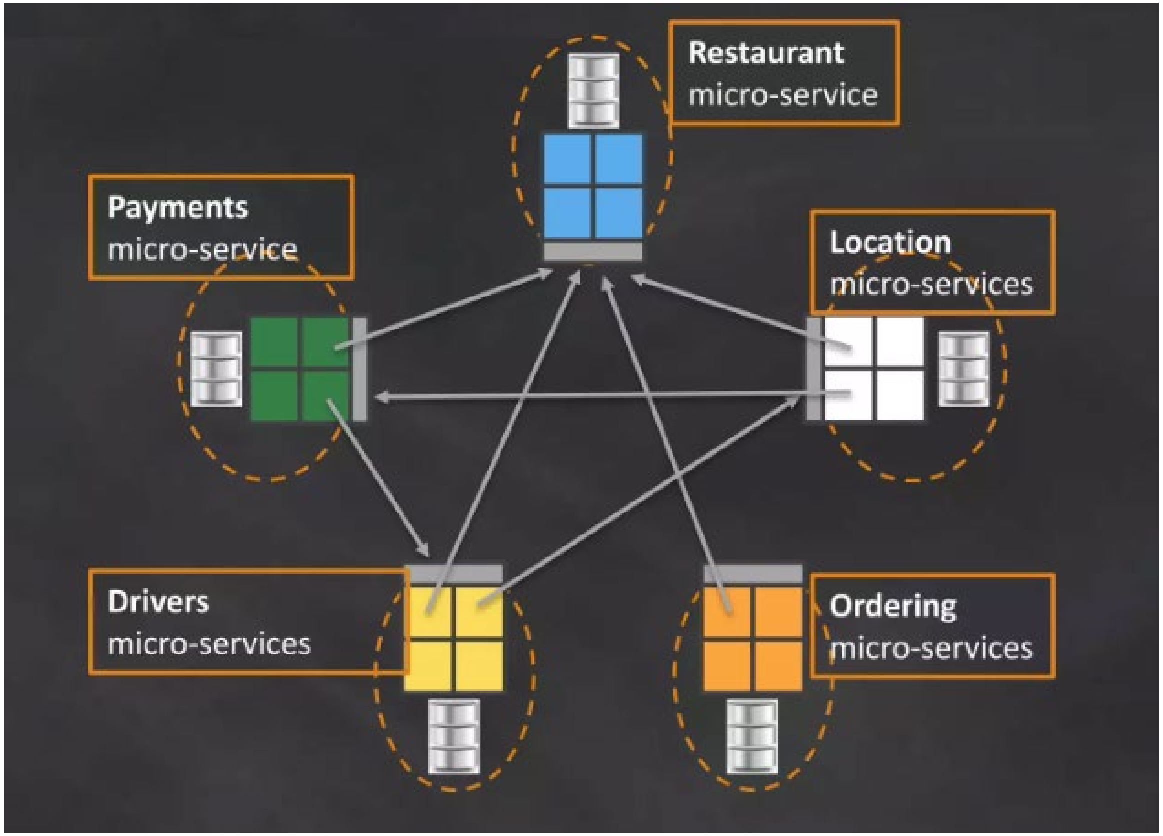
storeRestaurant (id, name, cuisine)
storeRestaurant (id, name, arbitrary_metadata)
addReview (restaurantId, rating, comments)

Version 2.0.0

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

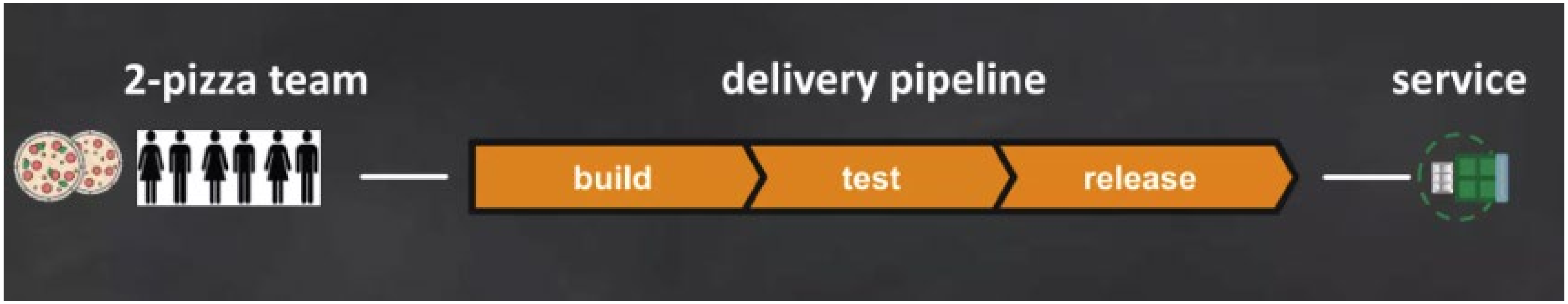
Echosystem of a delivery restaurant

- If one service fails, others are still operational
- In the future can add other services (e.g., Review)



How big should a microservice be?

- Avoid line of code size
- Amazon’s “two rule”
  - – Develop (code, test, deploy) in **two weeks**
  - – By a team that can be fed by **two pizza**
  - – Bozo: "We try to create teams that are no larger than can be fed by two pizzas.“
    - * Can be completed by six developers



Microservices definition

- 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
- Need to create cloud-based software products that are adaptable, scalable, and resilient?
  - Use a microservices architecture.

Photo printing system

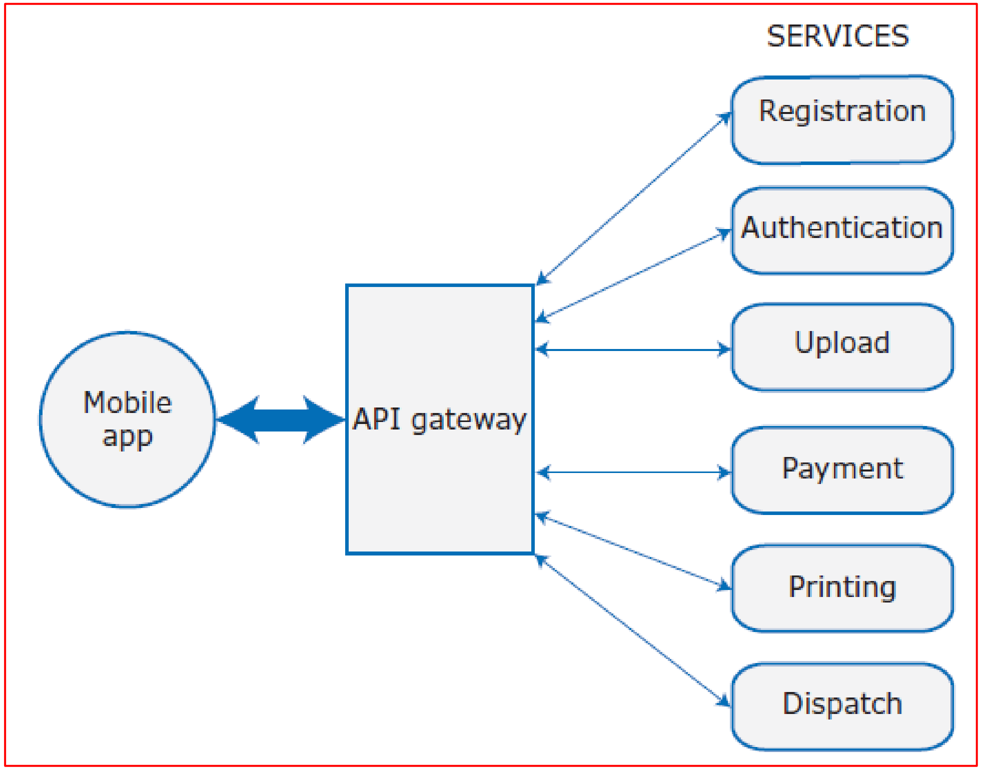
- Imagine that you are developing a photo printing service for mobile devices
  - Users can upload photos to your server from their phone or specify photos from their Instagram account
  - Users can choose print size and print medium
    - May decide to print a picture onto a mug or a T-shirt
  - Users can choose a shipping option
  - Users choose a payment option (CC, PP, or Android/Apple pay)

Photo printing system

- A monolithic architecture solution: the whole system has to be rebuilt, retested, and redeployed when changes are made
  - Slow
  - Frequent updates are not practical
  - When demands increases, the whole system has to be scaled
  - Larger servers must be used

Photo printing system

- A microservices architecture solution
  - API gateway: a single point of contact and translates service requests from the app into calls to the microservices
  - App need not know what service protocol to use

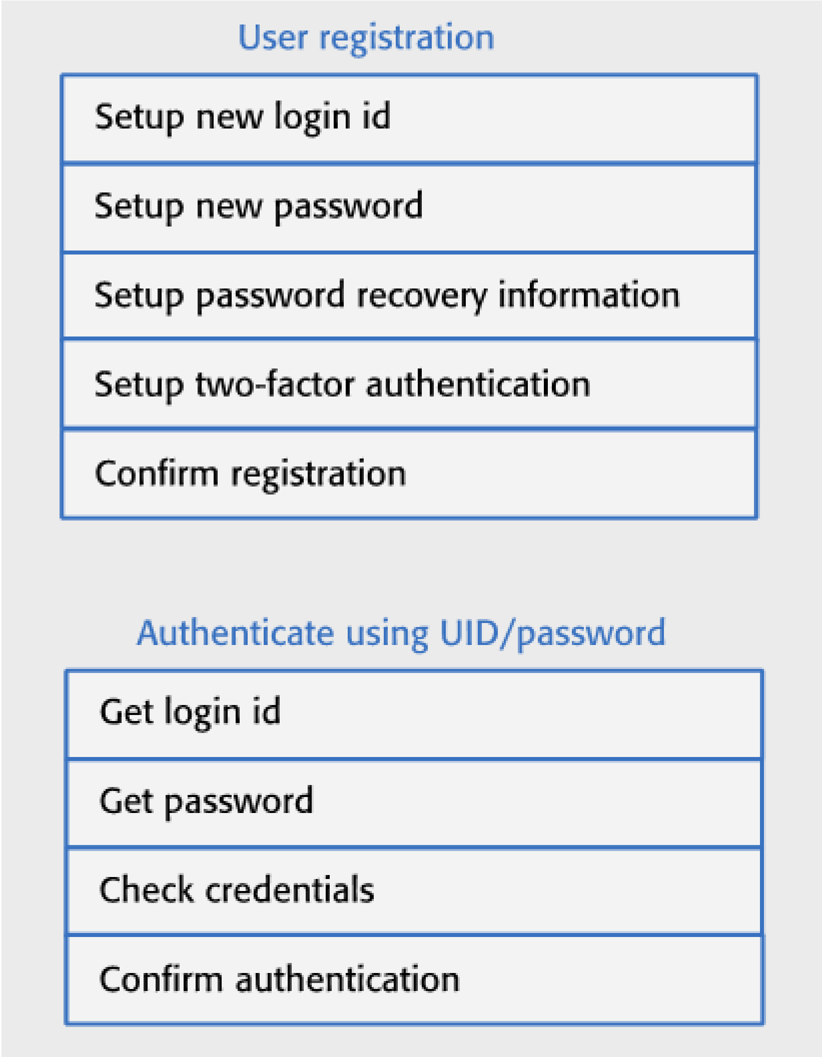


Another microservice example

- System authentication
  - **User registration**, where users provide information about their identity, security information, mobile (cell) phone number and email address.
  - **Authentication** using UID/password.
  - **Two-factor authentication** using code sent to mobile phone.
  - **User information management** e.g. changing a password or mobile phone number
  - **Reset** forgotten password
- To identify the microservices:
  - Break down into smaller functions

Another microservice example

- System authentication smaller functions
- Acceptable?
  - No!
  - Each has to have its own data
  - Each has be a business function



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

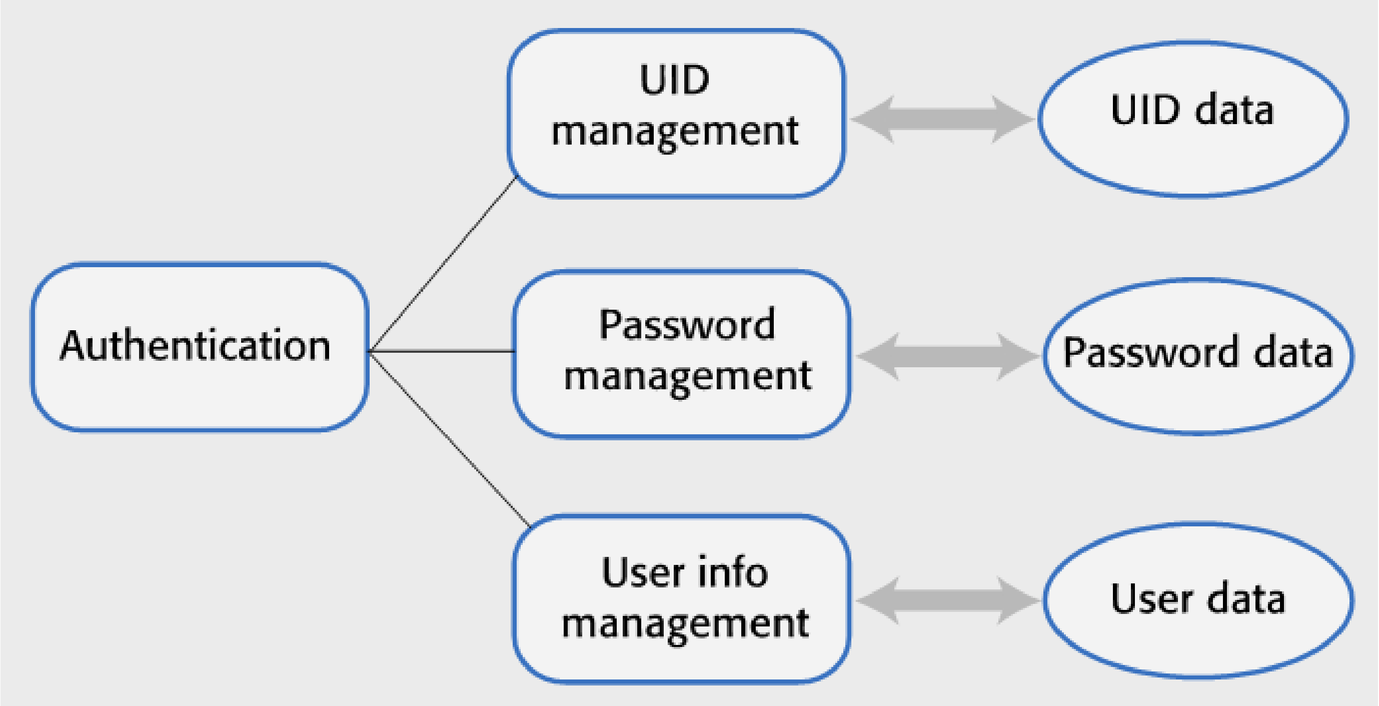
Get password

Check credentials

Confirm authentication

Another microservice example

- System authentication smaller functions



Characteristics of microservices

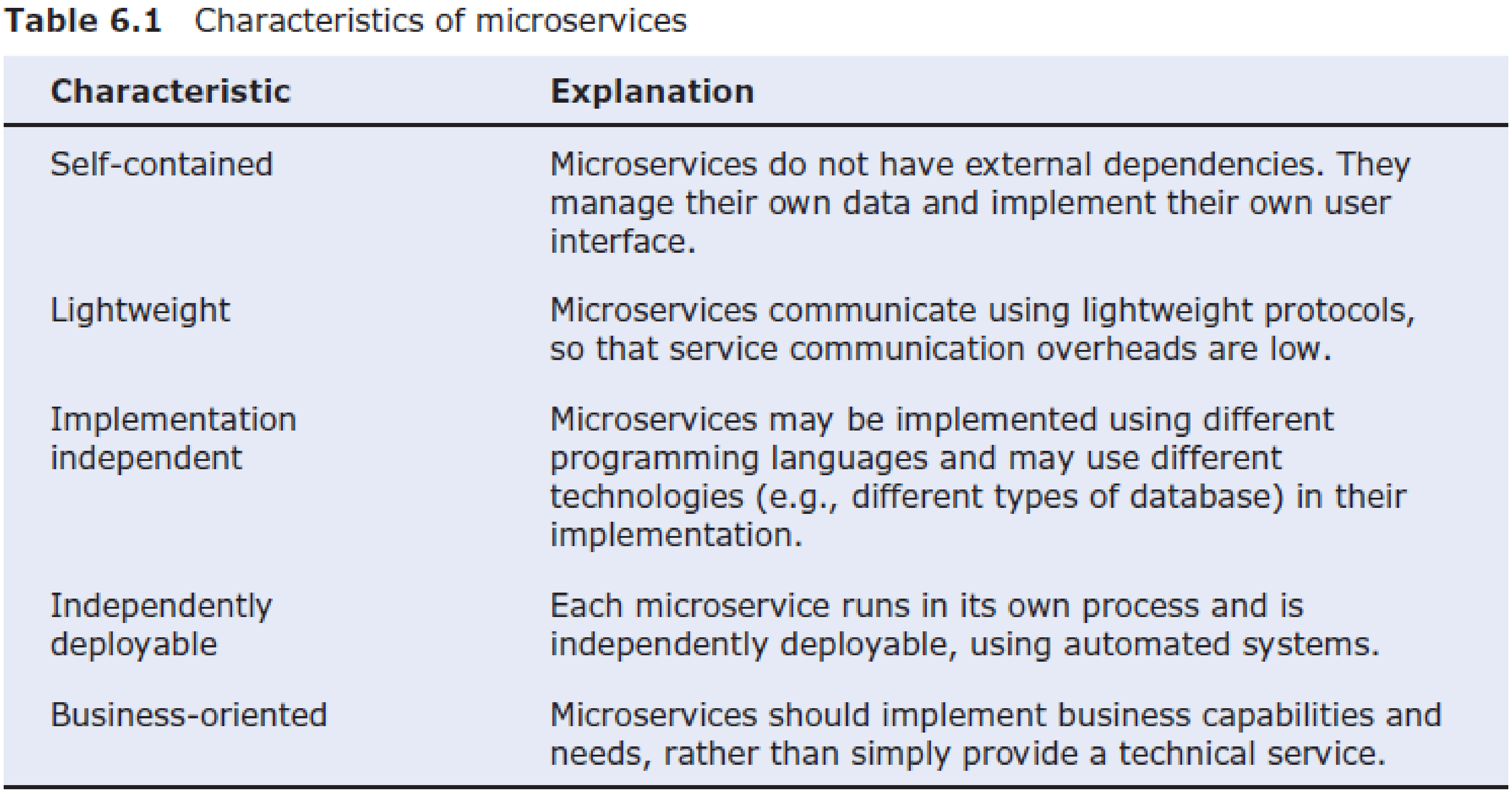


Table 6.1 Characteristics of microservices

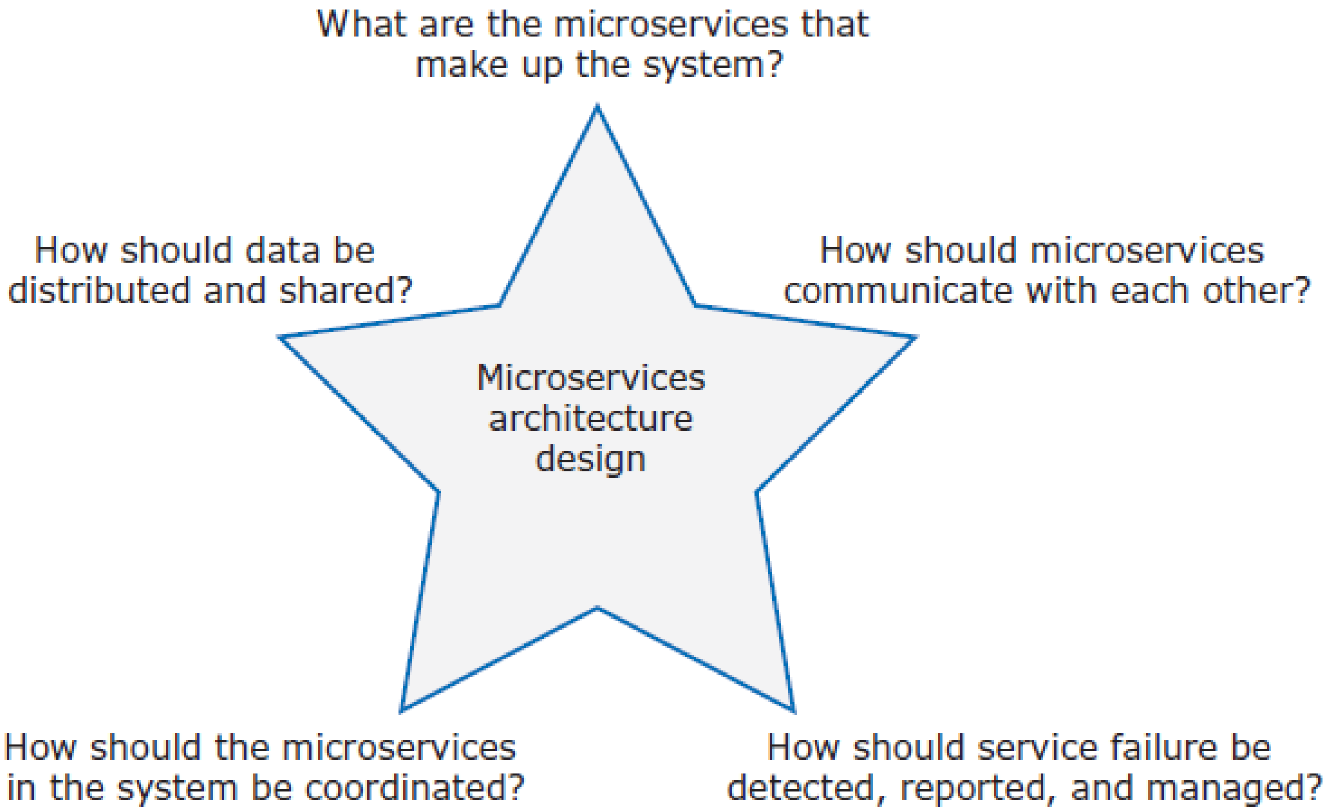
Characteristic	Explanation
Self-contained	Microservices do not have external dependencies. They manage their own data and implement their own user interface.
Lightweight	Microservices communicate using lightweight protocols, so that service communication overheads are low.
Implementation independent	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.

Microservices size revisited

- Remember the size criteria
- Question: a small function; why 6-8 people in two weeks?
  - Code
  - Test
  - UI code
  - Security
  - …
  - Because a microservices will have to be completely independent

Microservices architecture design questions

Figure 6.6 Key design questions for microservices architecture



What are the microservices that make up the system?

How should data be distributed and shared?

How should microservices communicate with each other?

Microservices architecture design

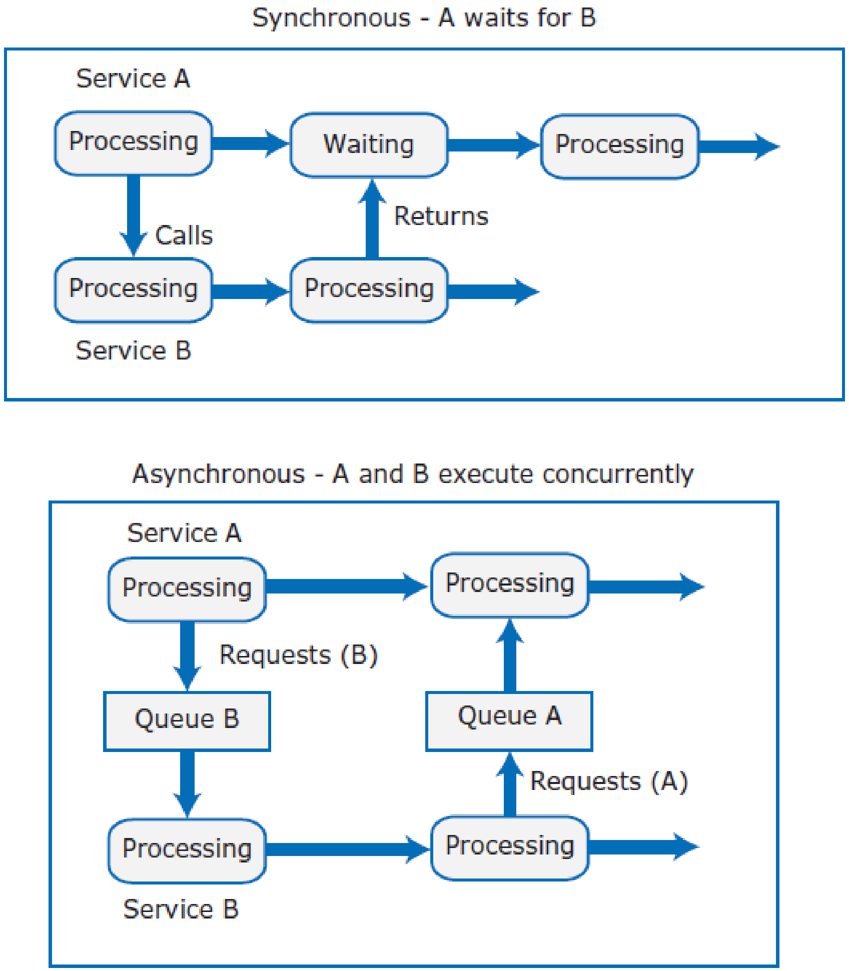
How should the microservices in the system be coordinated?

How should service failure be detected, reported, and managed?

Service communications

- Synchronous vs asynchronous
- Asynchronous communication is more challenging
- Recommendation
  – Start with synchronous
- Direct communication
  – Need the service URI

Figure 6.7 Synchronous and asynchronous microservice interaction



Synchronous - A waits for B

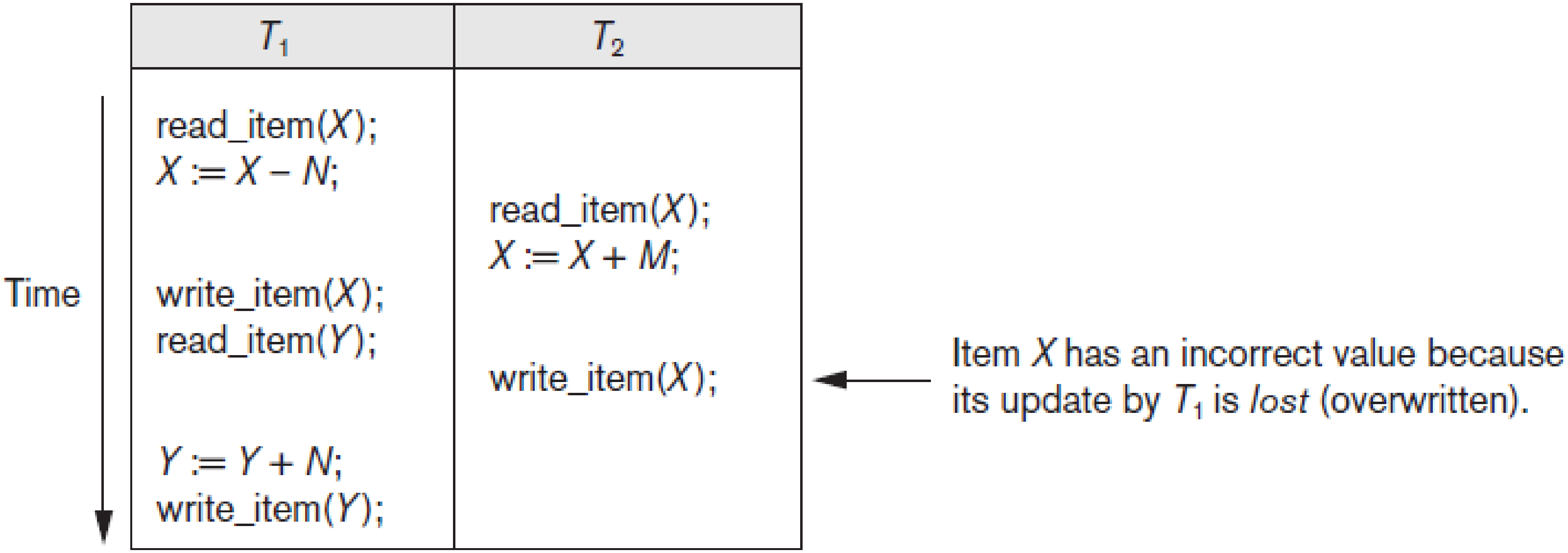
Asynchronous - A and B execute concurrently

Data distribution and sharing

- A general rule of microservice development: each microservice should manage its own data
- Complete data independence is impossible
  - Minimize data sharing
  - Limit to read-only
  - If data is replicated in database, keep them consistent
- Database *transaction* ACID rule apply
  - Atomicity, consistency, isolation, and durability
  - Either all updates are completed or none of them are
  - Database is always consistent

Why ACID is important

- **Lost update** problem occurs if an otherwise successful update of a data item by a transaction is overwritten by another transaction that wasn't "aware" of the first



T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;

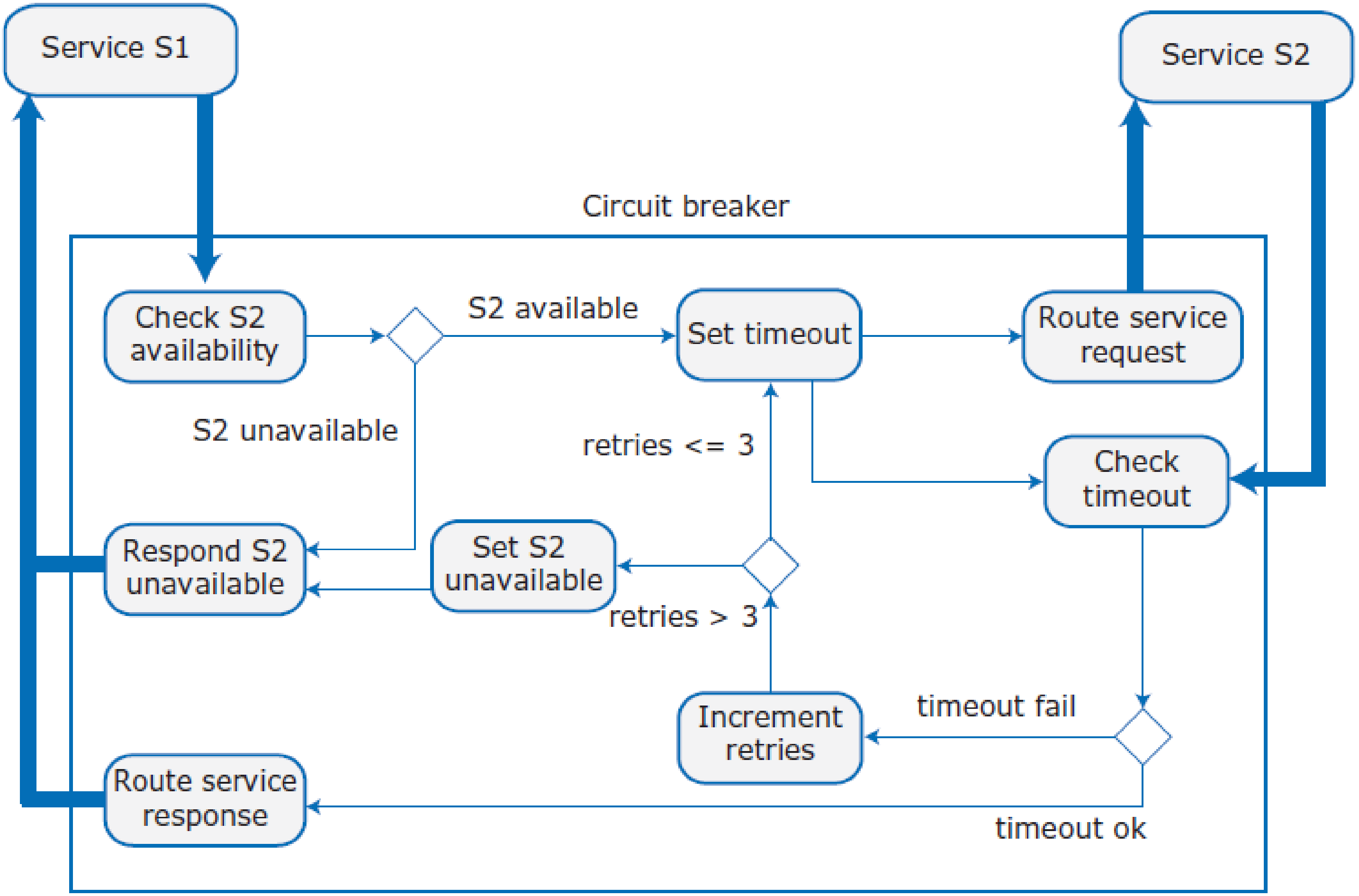
write_item(X);
read_item(Y);	

	write_item(X);
$Y := Y + N$; write_item(Y);	

Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

Coping with service failure

Figure 6.12 Using a circuit breaker to cope with service failure



RESTful services protocol

- Use HTTP verbs
  - GET, PUT, POST, DELETE
- Stateless services
  - Services never maintain internal state
- URI addressable
  - All resources have a URI
- Use XML or JSON
  - Resource should be in JSON or XML or both

XML, JSON representation

- An incident description

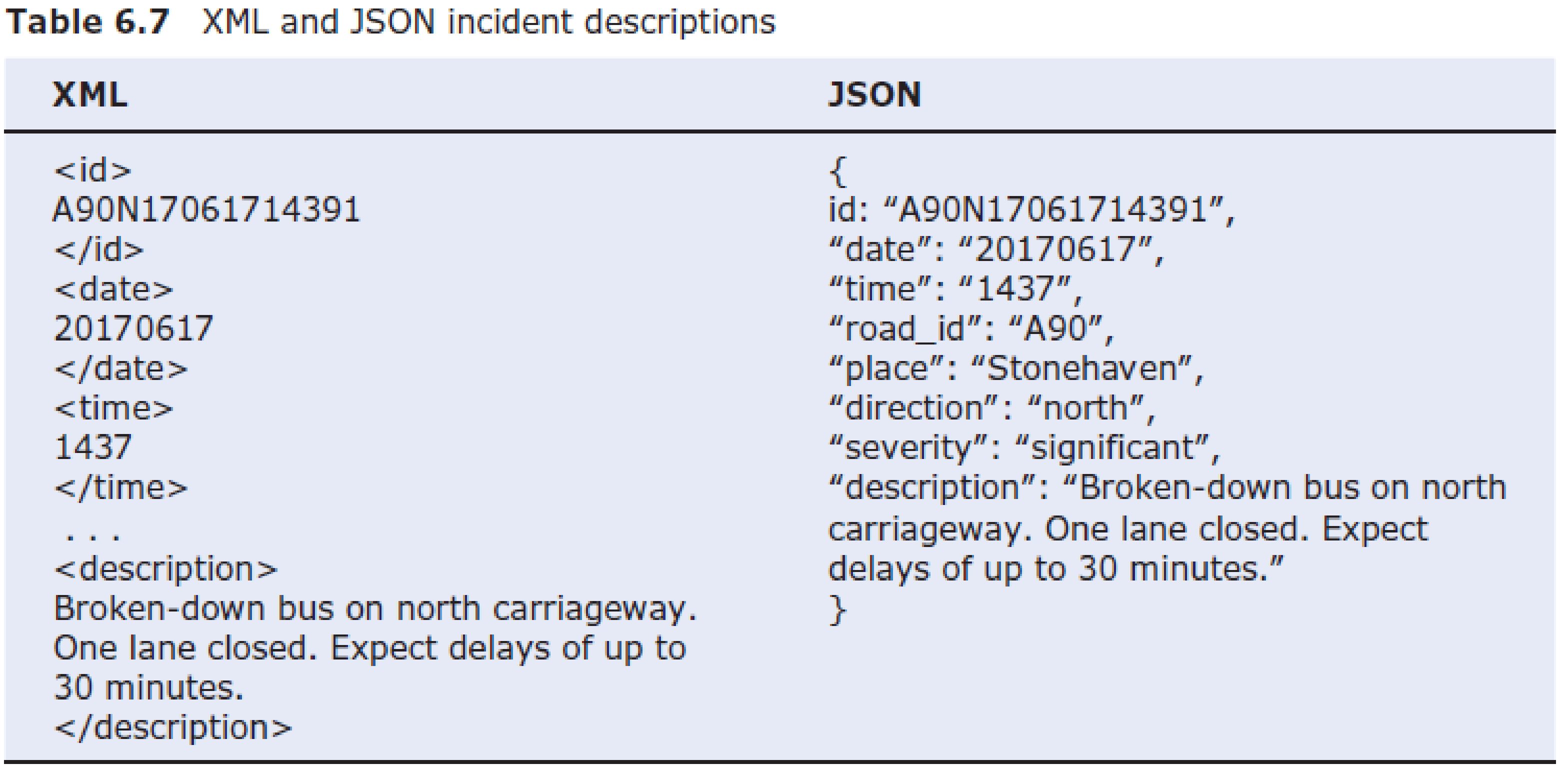


Table 6.7 XML and JSON incident descriptions

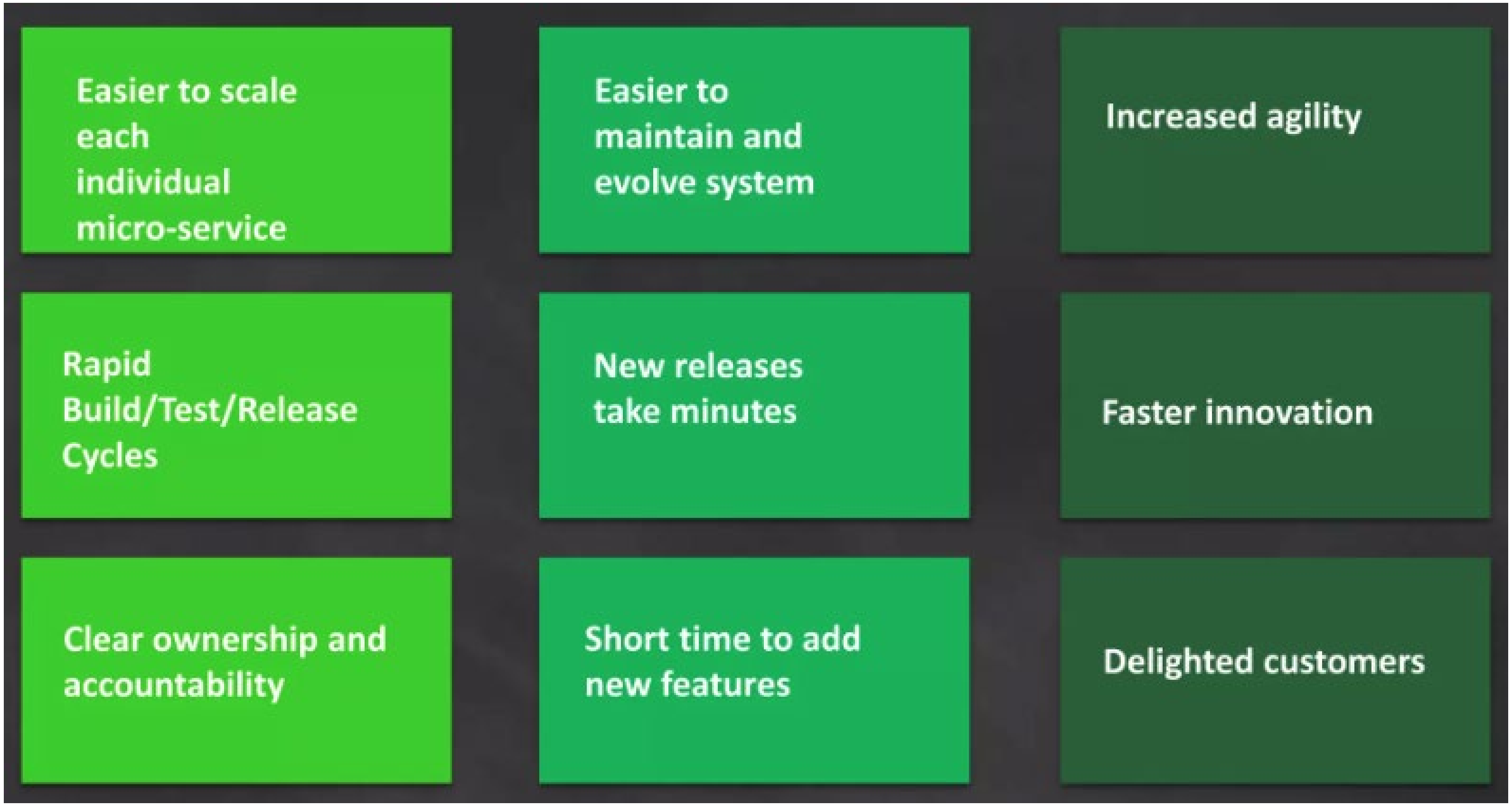
XML	JSON

<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>
	{
	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.”
	}

Summary and key points*

- A microservice is an independent and self-contained software component
  - Runs in its own process
  - 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

Summary: 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

References

- Peter Dalbhanjan, Introduction to Microservices, AWS