Microservices Architecture

PROF. P. M. JADAV

ASSOCIATE PROFESSOR

COMPUTER ENGINEERING DEPARTMENT

FACULTY OF TECHNOLOGY

DHARMSINH DESAI UNIVERSITY, NADIAD

Monolith E-Commerce Application Architecture

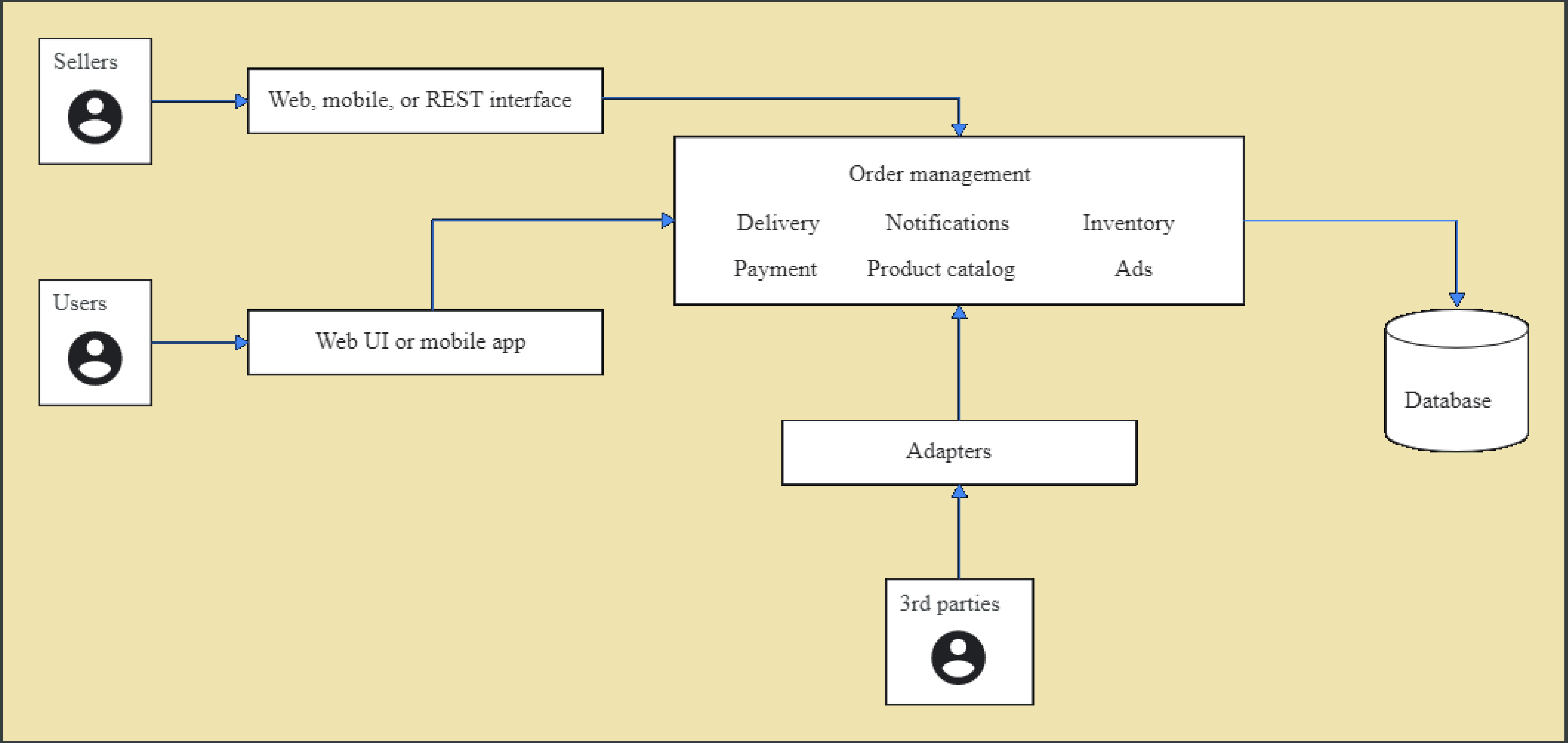


Image Source: https://cloud.google.com/architecture/microservices-architecture-introduction

Monolith Benefits

- End-to-end testing is easy
- Single packaged application can be deployed easily on the server
- Single solution to cross-cutting concerns (logging, caching, security) are easy because application share all the resources.
- Performance advantage

Monolith Drawbacks

- Progressively become harder to build, debug and reason
- Changes are difficult in complex and tightly coupled modules
- Testing process is longer
- Complicated to do continuous integration and deployment CI/CD
- Single update will required redeploying an entire application
- Lot of manual testing required to do regression

Monolith Drawbacks (..cont)

- Difficult to scale
- A bug in one of the module may bring down the entire system
- Difficult to adopt/upgrade to a new technologies/framework

Microservices Architecture

- It is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team
- It enables the rapid, frequent and reliable delivery of large, complex applications
- It also enables an organization to evolve its technology stack

Microservices Architecture Design

- Distributed architecture
  - All the services communicate with the API gateway through REST or RPC. These services can be deployed as multiple instances, and the requests can be distributed to these instances

- Separately deployed components
  - Each component is deployed separately. If one component needs changes, others don’t have to deploy again

Microservices Architecture Design (..cont)

- Service components
  - Services components communicate with each other via service discovery
- Bounded by contexts
  - It encapsulates the details of a single domain, and define the integration with other domains. It is about implementing a business capability

E-Commerce Application with MSA

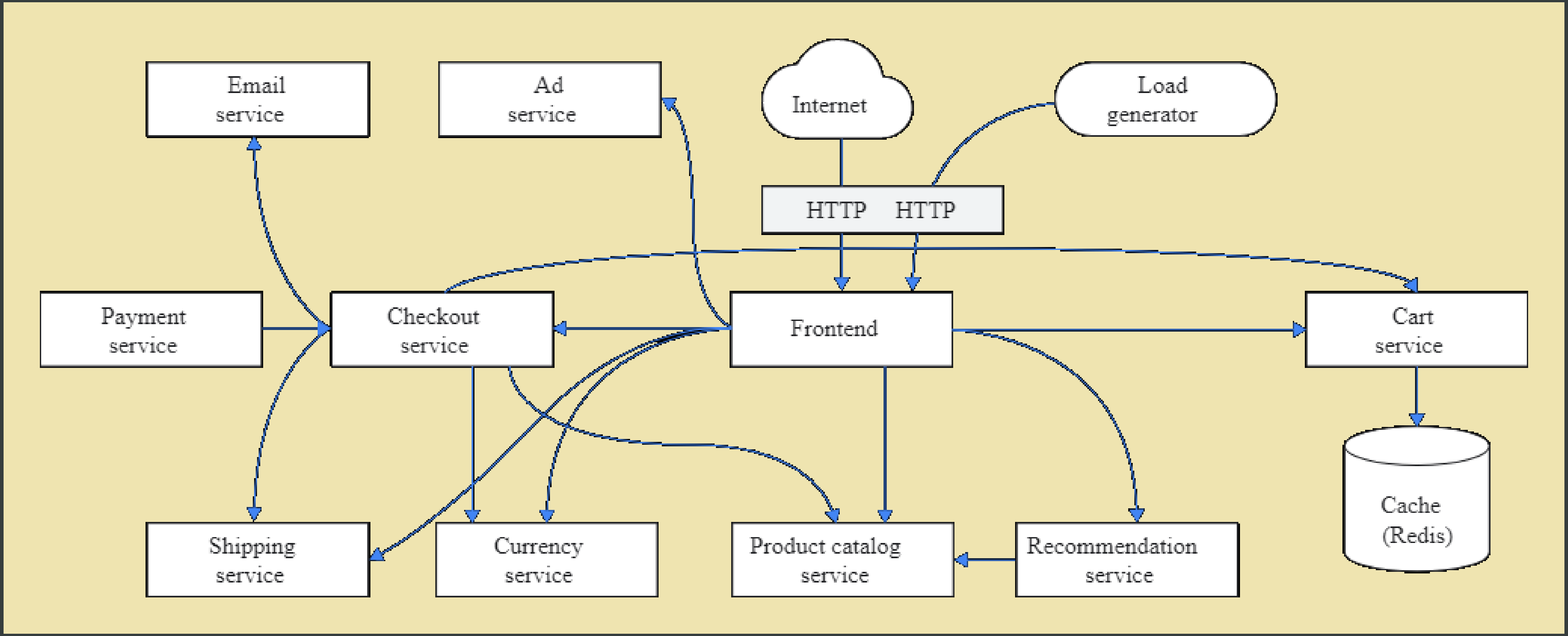


Image Source: https://cloud.google.com/architecture/microservices-architecture-introduction

Each Service in MSA has its own database

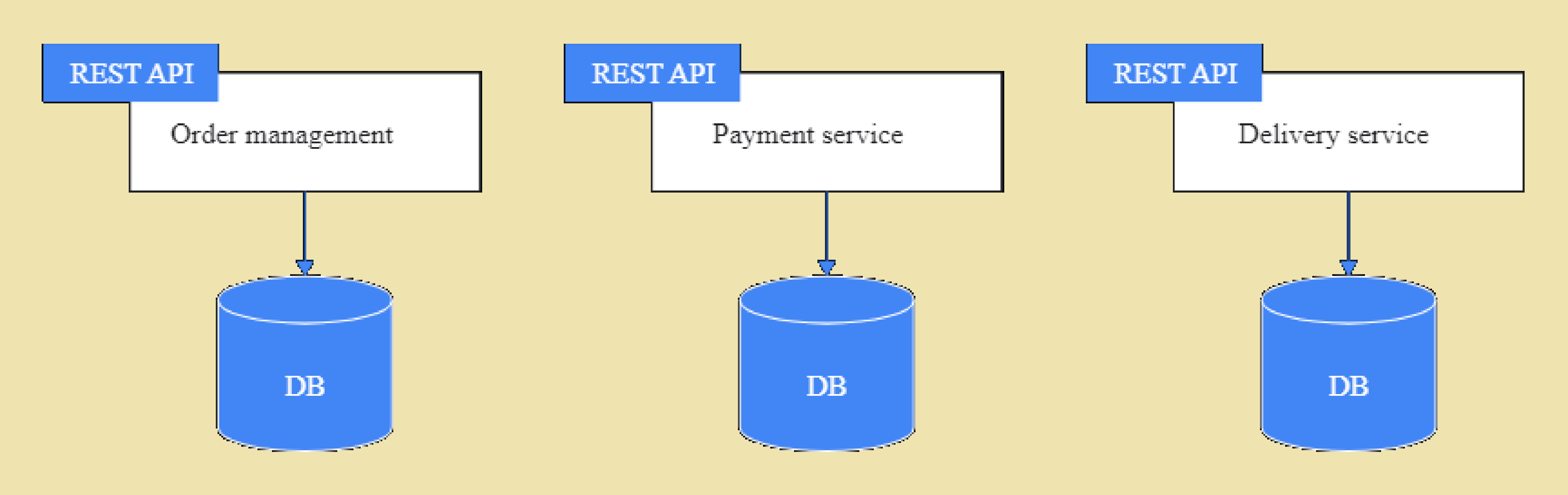


Image Source: https://cloud.google.com/architecture/microservices-architecture-introduction

Characteristics of Microservices Architecture

- Small in size
- Messaging enabled
- Bounded by contexts
- Autonomously developed
- Independently deployable
- Decentralized
- Built and released with automated processes

Benefits of Microservices Architecture

- Asynchronicity
- Integration & Disintegration
- Independent Deployments
- Evolutionary Architecture
- Components are deployed
- Features are released
- Applications consist of routing
- Easier to understand the code – It is easy to distinguish one small service and flow of the whole service rather than one big code base

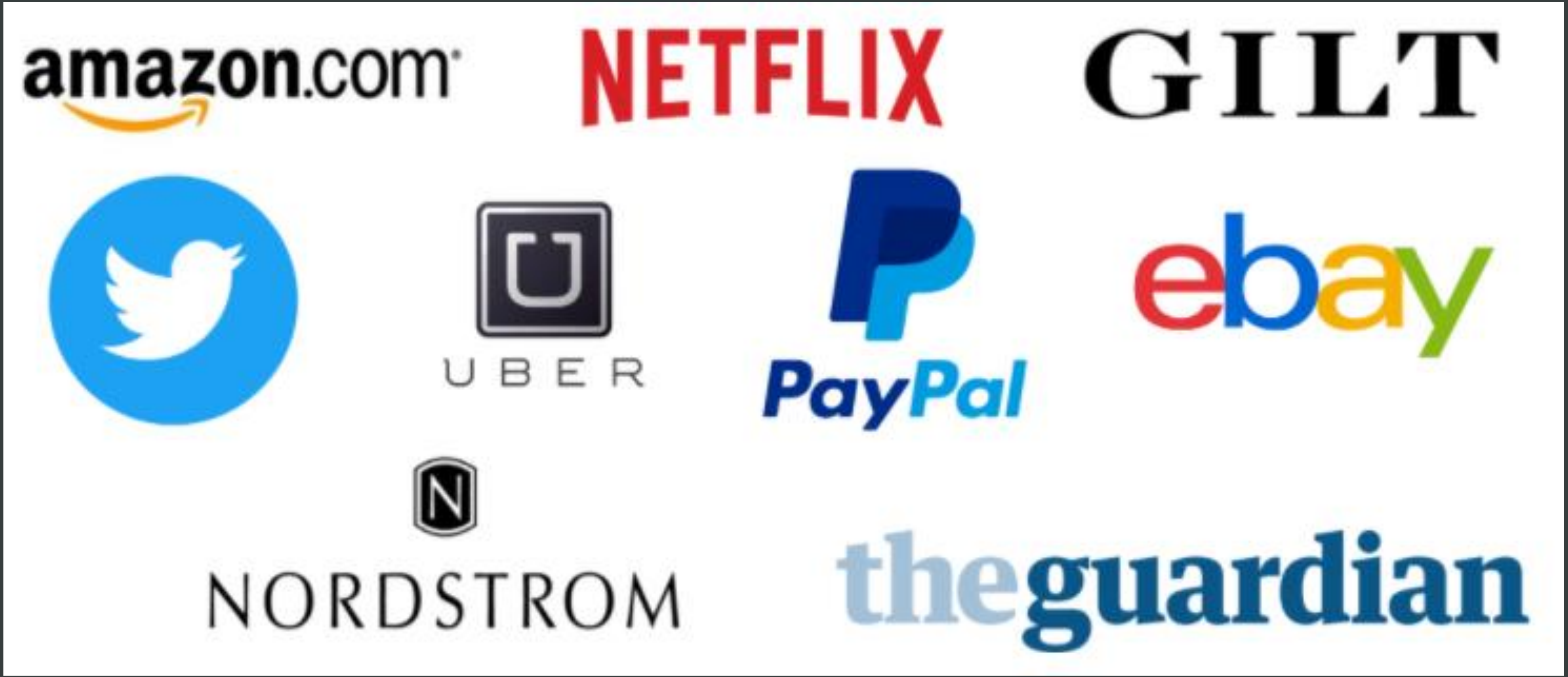
Benefits of Microservices Architecture (..cont)

- Fast Software delivery – Each service can be developed by different developers and in many different languages
- Efficient debugging – Don’t have to jump through multiple layers of an application and in essence better fault isolation
- Reusable – Since it is an independent service it can be used in other projects also

Benefits of Microservices Architecture (..cont)

- Scalability
- Horizontal scaling
- Workload partitioning
- Don’t have to scale the whole project, only need to scale up that component which needs to scale up
- Deployment – Need only to deploy that service which has been changed not the whole project again

Companies Using Microservices

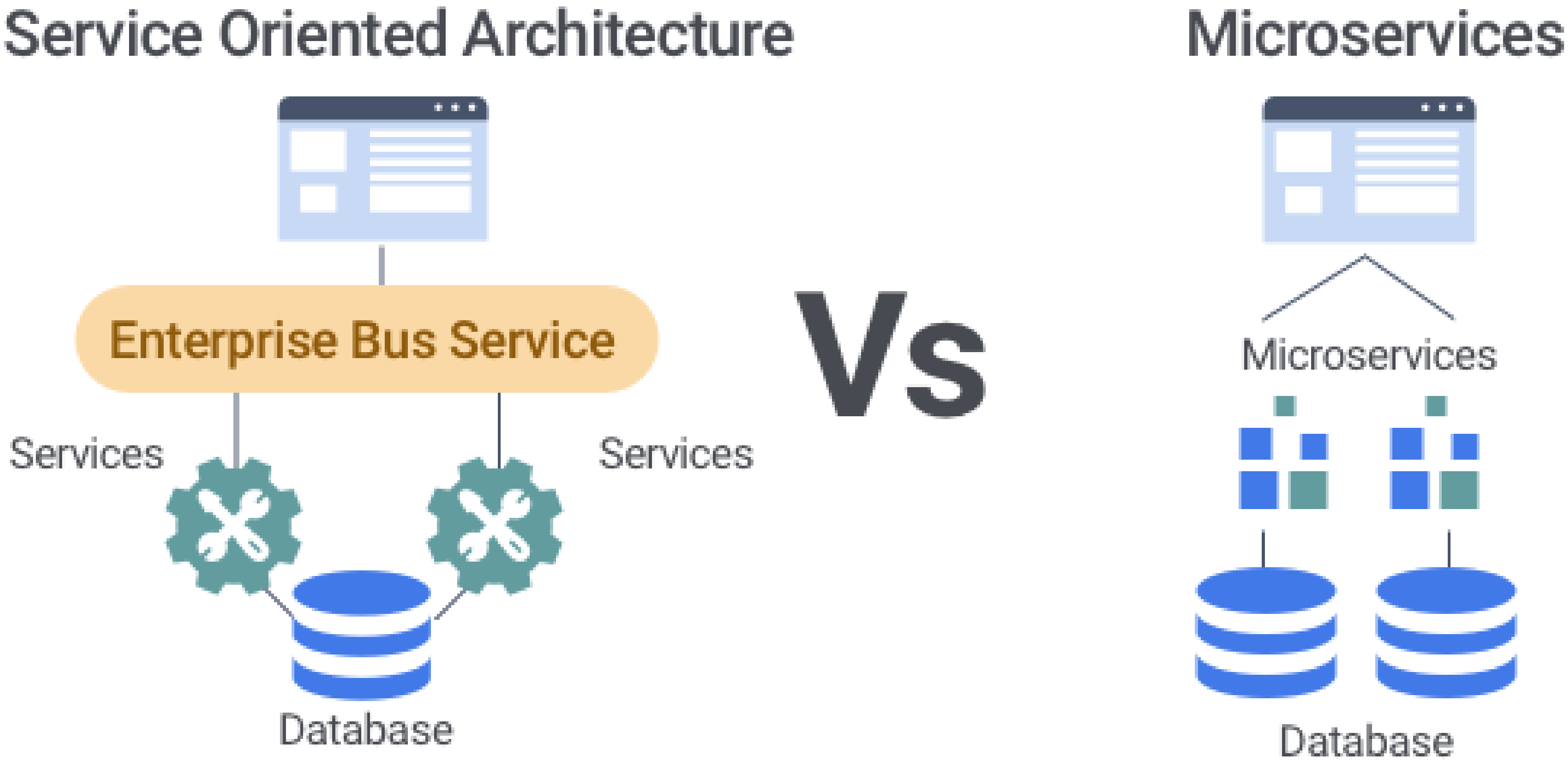


Challenges of Microservices Architecture

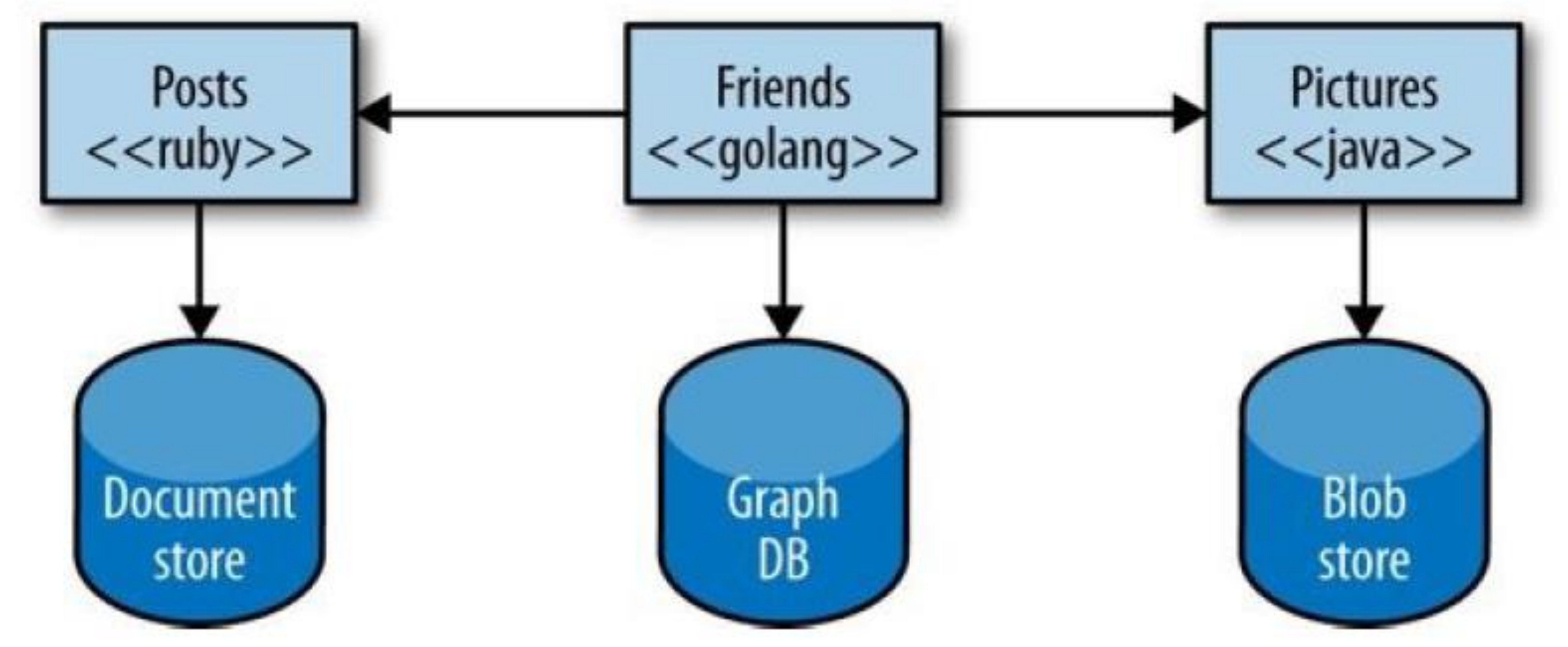
- Microservices has all the associated complexities of the distributed system
- There is a higher chance of failure during communication between different services
- Difficult to manage a large number of services
- The developer needs to solve the problem, such as network latency and load balancing
- Complex testing over a distributed environment

SOA vs Microservices

XENONSTACK



Technology Heterogeneity (Social Network App)



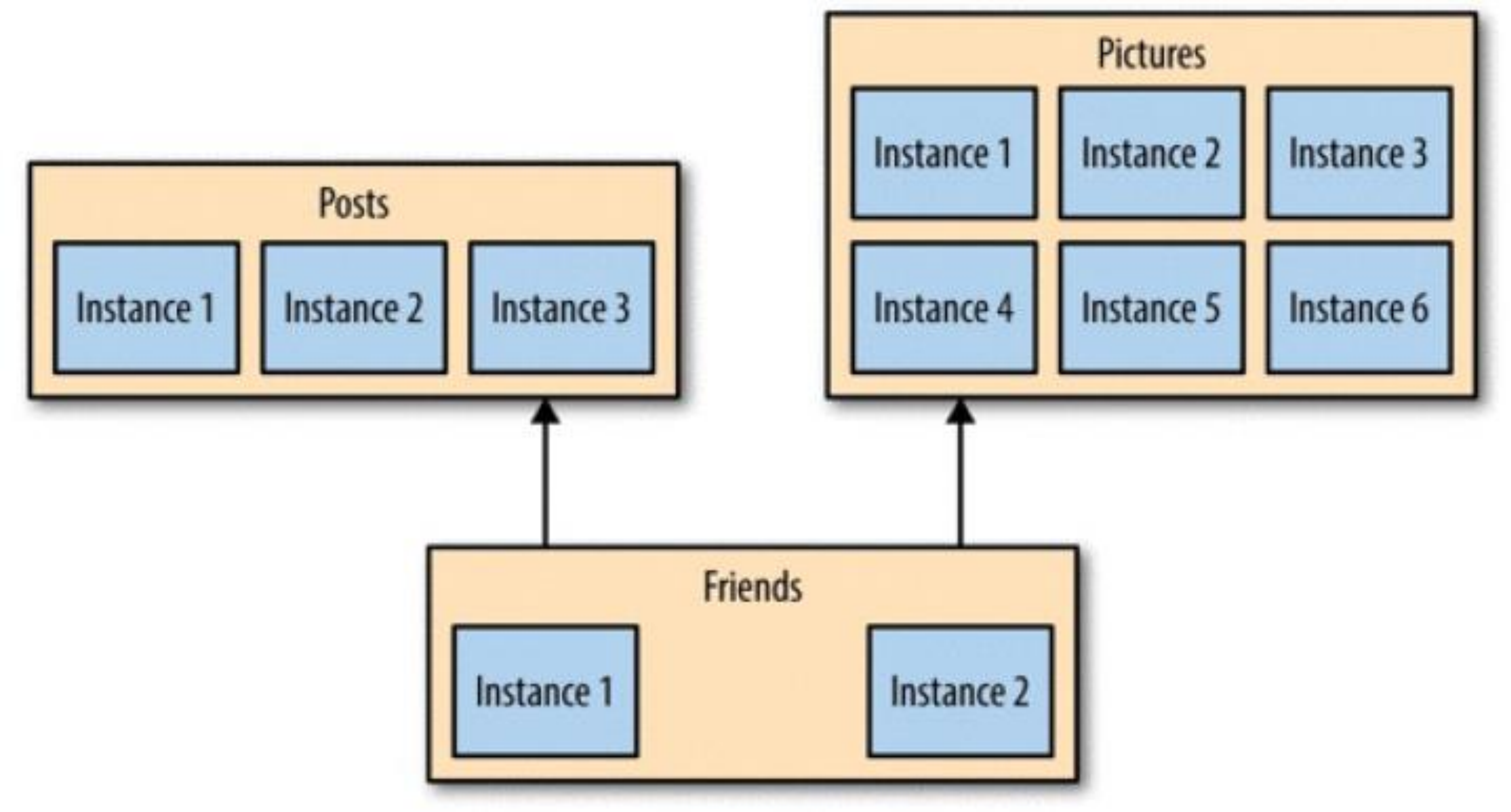
Resilience

- If one component of a system fails, but that failure doesn’t cascade, you can isolate the problem and the rest of the system can carry on working
- In a monolithic service, if the service fails, everything stops working
- With a monolithic system, we can run on multiple machines to reduce our chance of failure
- With microservices, we can build systems that handle the total failure of services and degrade functionality accordingly
- With Microservices we need to understand the new sources of failure that distributed systems have to deal: network/machines failure

Scaling

- With a large, monolithic service, we have to scale everything together
- One small part of our overall system is constrained in performance, but if that behavior is locked up in a giant monolithic application, we have to handle scaling everything as a piece
- With smaller services, we can just scale those services that need scaling
- It allows us to run other parts of the system on smaller, less powerful hardware

Scaling


Posts
Instance 1
Instance 2
Instance 3
Pictures
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Friends
Instance 1
Instance 2

Ease of Deployment

- A one-line change to a million-line-long monolithic application requires the whole application to be deployed in order to release the change
- That could be a large-impact, high-risk deployment
- With microservices, we can make a change to a single service and deploy it independently of the rest of the system
- This allows us to get our code deployed faster
- If a problem does occur, it can be isolated quickly to an individual service, making fast rollback easy to achieve
- It also means we can get our new functionality out to customers faster

Organizational Alignment

- There are problems associated with large teams and large codebases
- These problems can be exacerbated when the team is distributed
- Smaller teams working on smaller codebases tend to be more productive
- Microservices allow us to better align our architecture to our organization
- It help us minimize the number of people working on any one codebase as per the team size and productivity

Composability

- With a monolithic application, we have one coarse-grained seam that can be used from the outside
- With microservices, we allow for our functionality to be consumed in different ways for different purposes
- Web, native application, mobile web, tablet app, or wearable device

Optimizing for Replaceability

- Microservices being small in size, the cost to replace them with a better implementation, or even delete them altogether, is much easier to manage
- The barriers to rewriting or removing services entirely are very low

Enterprise Service Bus (ESB)

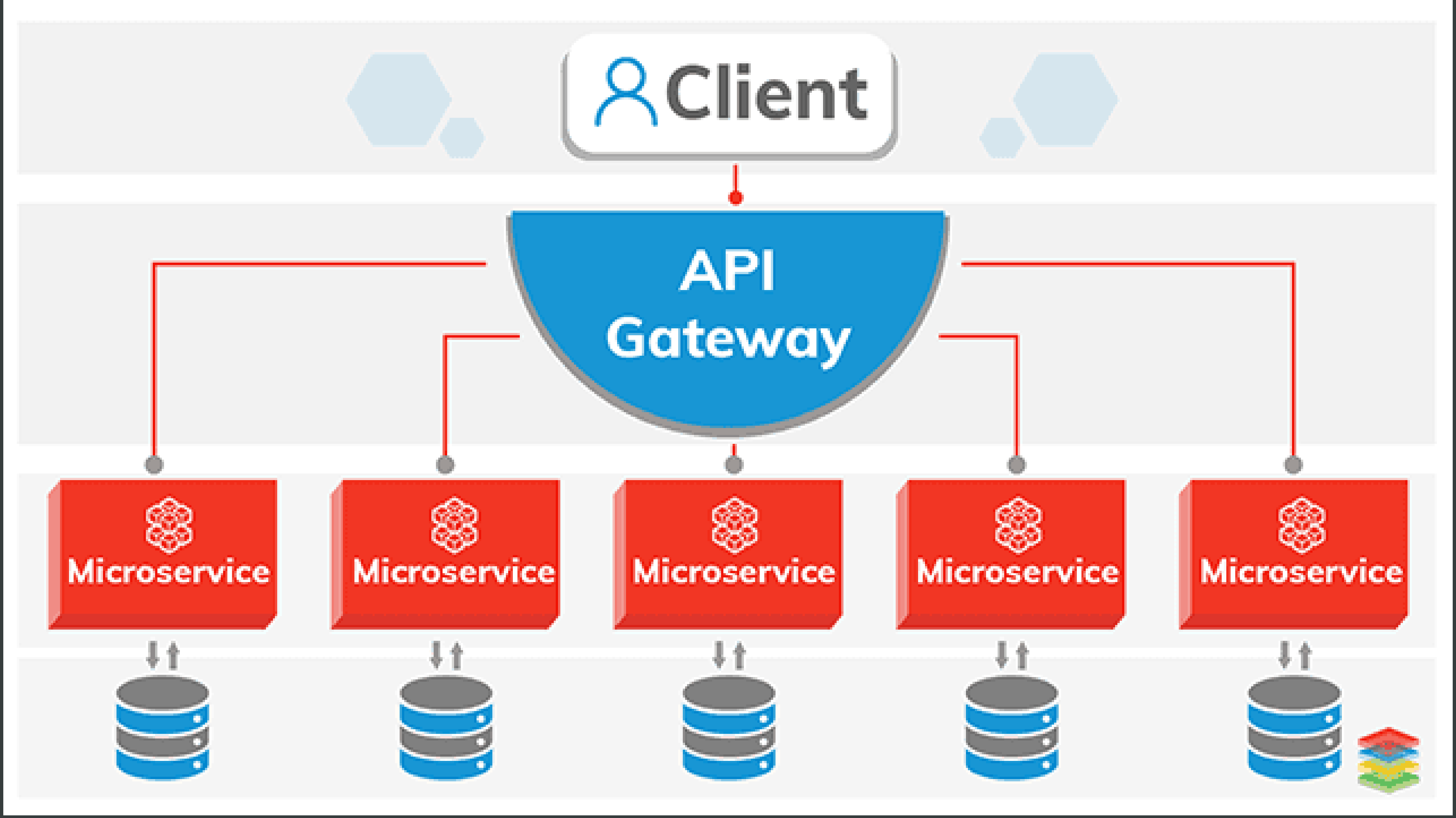
- It is a middleware tool used to distribute work among connected components of an application
- Designed to provide a uniform means of moving work, offering applications the ability to connect to the bus and subscribe to messages based on simple structural and business policy rules
- It's a tool that has use in both distributed computing and component integration
- Visualize it as a set of switches that can direct a message along a specific route between application components based on message contents and implementation or business policies

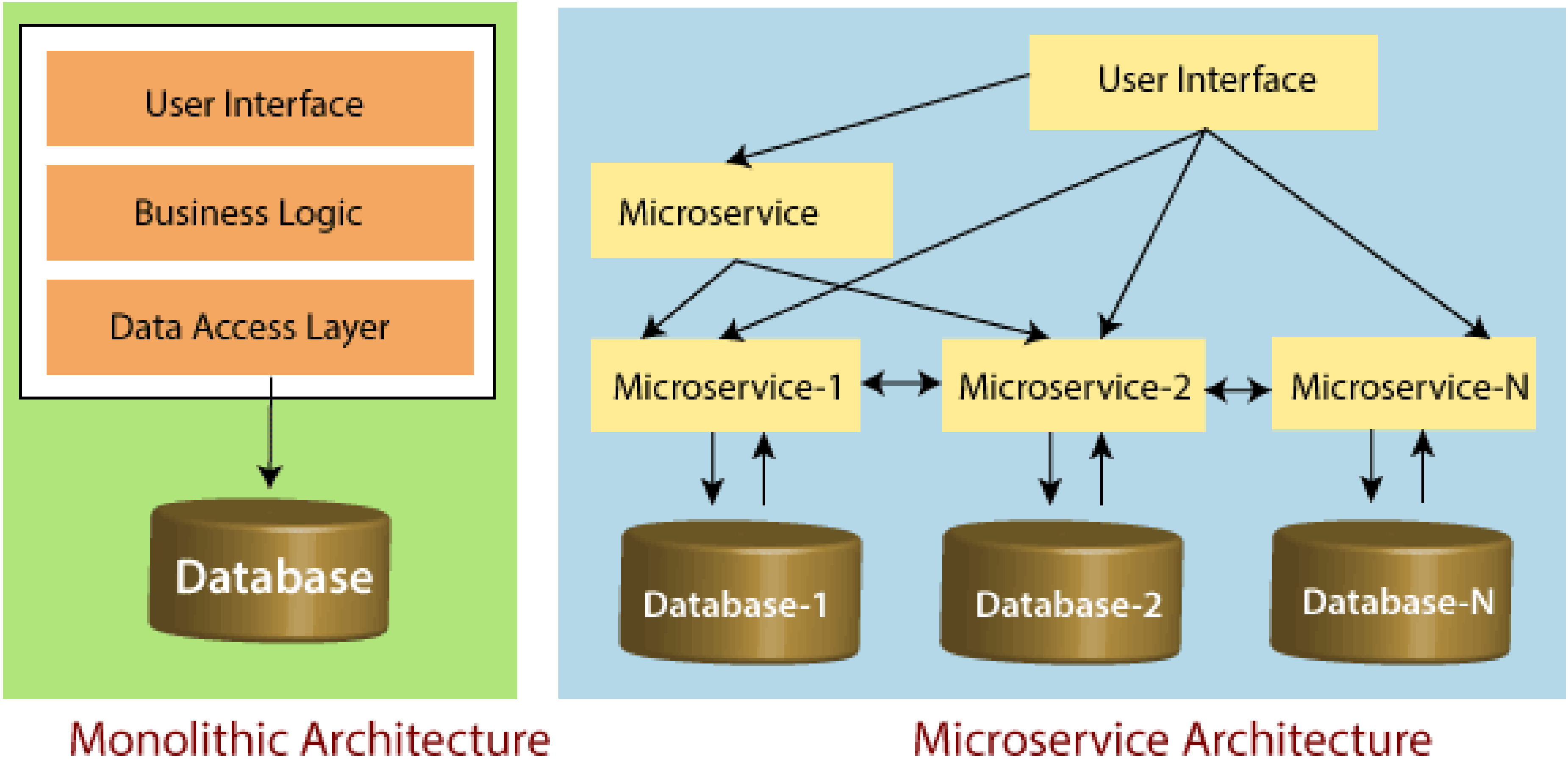
Enterprise Service Bus (ESB)

- It is the centre of application workflow
- It is, in effect, a message queue that handles information exchanges throughout the application
- It does not dictate whether components that use the bus are local to it or remote, nor does it enforce any specific requirements for programming languages
- It acts to unify the various ways in which components can receive or send information to other application elements

Enterprise Service Bus (ESB)

- it makes it easy to change components or add additional components to an application
- It also makes for a convenient place to enforce security and compliance requirements, log normal or exception conditions and even handle transaction performance monitoring
- It also provides load balancing in which multiple copies of a component can be instantiated to improve performance
- It can also often provide failover support should a component or its resources fail


Client
API
Gateway
Microservice
Microservice
Microservice
Microservice
Microservice



User Interface

Business Logic

Data Access Layer

Database

Monolithic Architecture

User Interface

Microservice

Microservice-1

Microservice-2

Microservice-N

Database-1

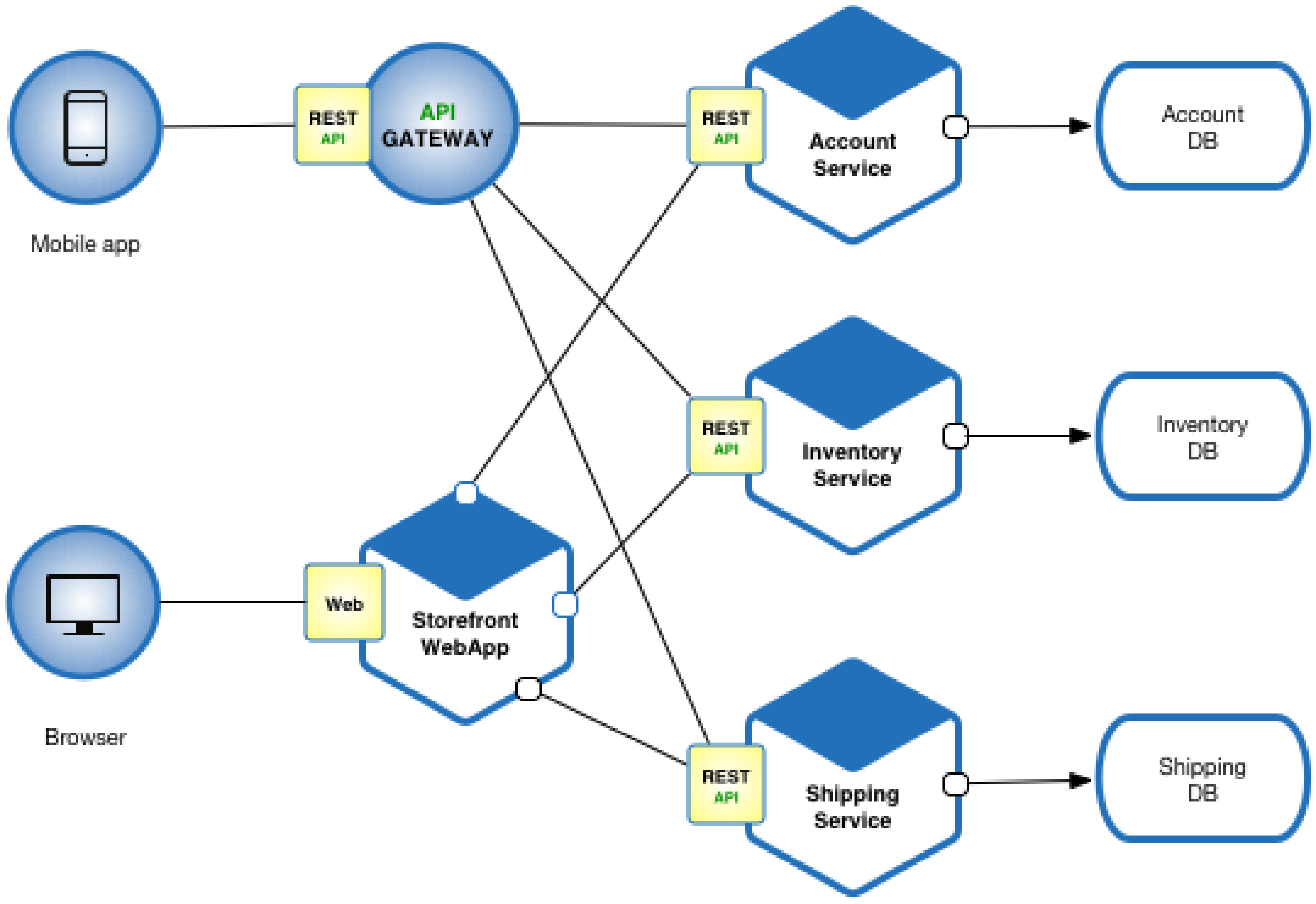
Database-2

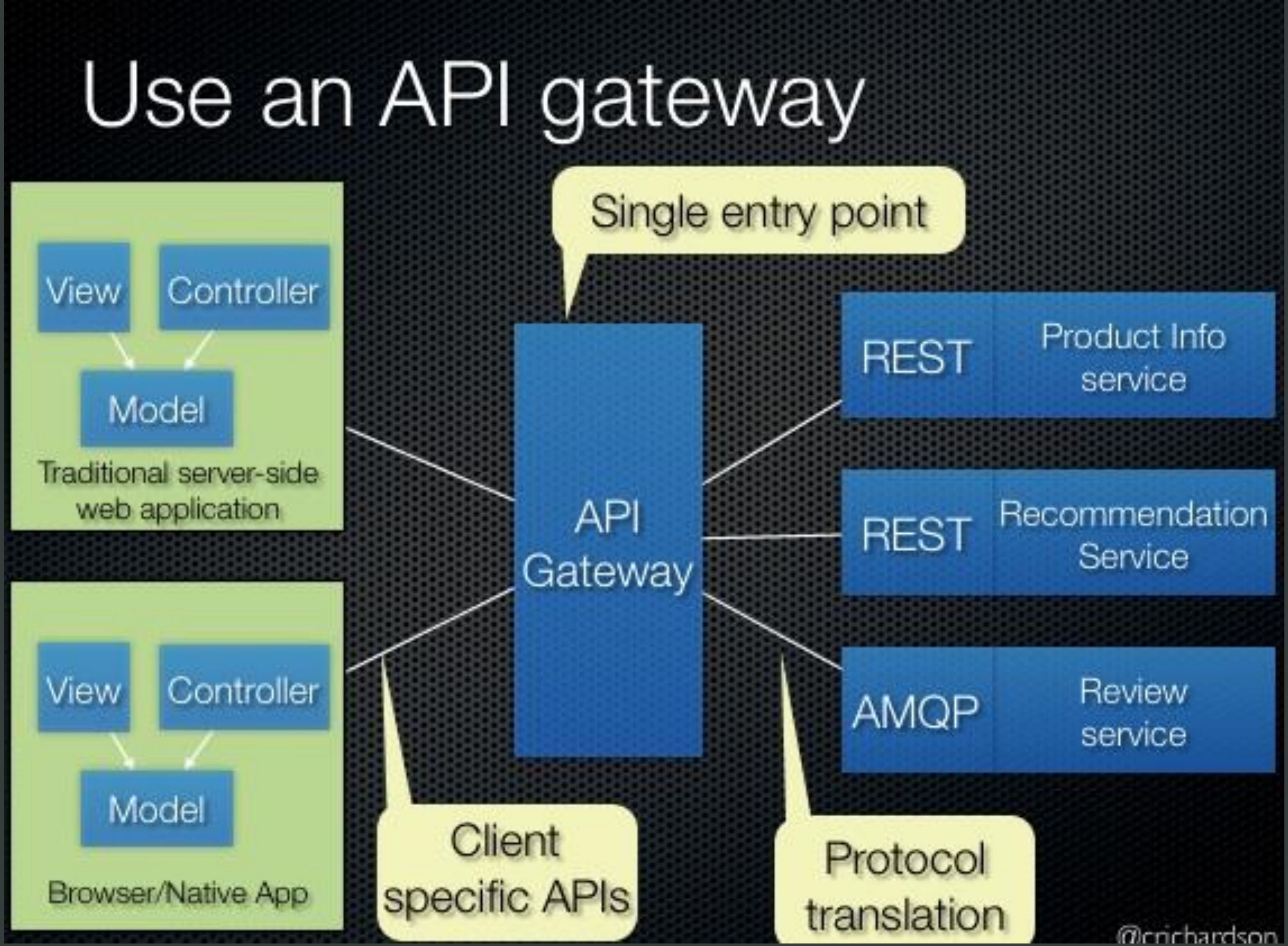
Database-N

Microservice Architecture

Monolithic vs Microservice Architecture

E-Commerce Application





Use an API gateway

@crichardson

An API Gateway

- An API gateway is an API management tool that sits between a client and a collection of backend services
- An API gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfil them, and return the appropriate result

Why use an API Gateway?

- Abstracting the clients from the complexity of the backend system
- Handling Authentication and Authorization of users in a single place for the entire system
- Transforming the requests and responses according to the format needed
- Rate limiting for each of the APIs to prevent the load on backend services
- Throttling the API requests made by a single user to prevent the DDOS attack
- Reducing the load on the services by Caching the responses for the repeated requests

Why use an API Gateway?

- Supports different protocols
- Metering individual APIs to make billing the clients based on usage
- Logging and tracing the API calls into the system in a single place
- Error handling
- Aggregating the data across microservices and respond the data to the client on a single API call. This will avoid multiple API calls the clients to need to make

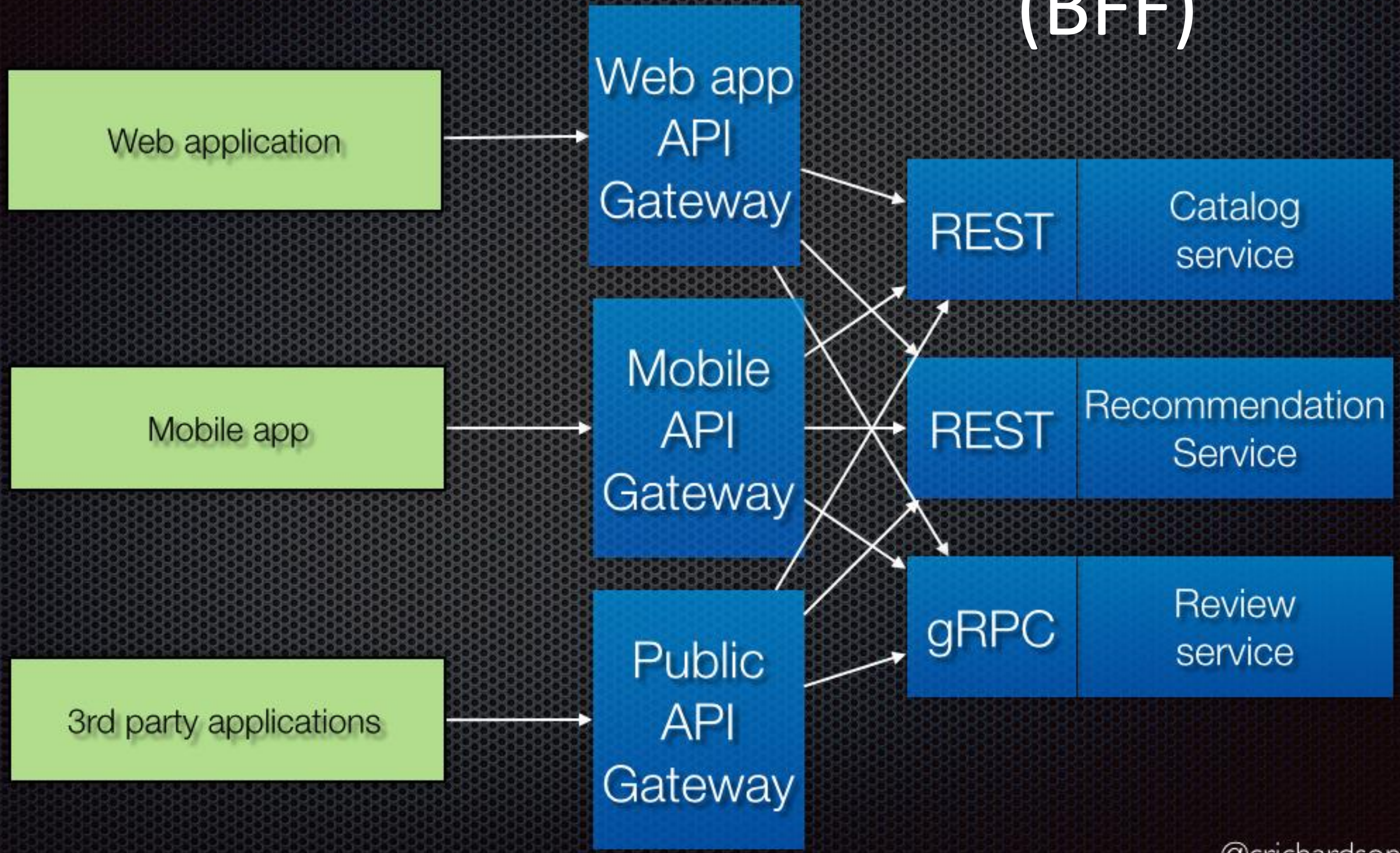
Drawbacks of an API Gateway

- It can become a single point of failure
- It could cause performance issues and increase the latency of API calls if we perform too many things in the API Gateway
- If we have too many microservices, managing too many routes could get complicated
- There is always an extra hop on API calls from the client to the server
- Extra cost because of the additional server component involved

Backend for Frontend (BFF)

- The BFF pattern is an architectural paradigm, a variant of the API gateway pattern
- It comprises of multiple back-ends that are designed to meet the demands of specific front-end applications, such as desktop, browser, and native-mobile applications, IOT devices etc.

Variation: Backends for frontends (BFF)



@crichardson

Microservice Based Architecture	Service-Oriented Architecture
Uses **protocols** such as **REST**, and **HTTP**, etc.	SOA supports **multi-message protocols**.
It focuses on **decoupling**.	It focuses on application service **reusability**.
It uses a **simple messaging system** for communication.	It uses **Enterprise Service Bus** (ESB) for communication.
follows "**share as little as possible**" architecture approach.	follows "**share as much as possible architecture**" approach.
much better in **fault tolerance**	not better in fault tolerance in comparison to MSA.
Each microservice have an **independent** database.	SOA services share the **whole** data storage.
MSA used **modern** relational databases.	SOA used **traditional** relational databases.
MSA tries to **minimize** sharing through bounded context (the coupling of components and its data as a single unit with minimal dependencies).	SOA **enhances** component sharing.
It is better suited for the **smaller** and **well portioned**, web-based system.	It is better for a **large** and **complex** business application environment.

References (Web)

- https://www.xenonstack.com/insights/service-oriented-architecture-vs-microservices/
- https://www.xenonstack.com/insights/microservices/
- https://www.tutorialspoint.com/microservice_architecture/microservice_architecture_introduction.htm
- https://www.javatpoint.com/microservices
- https://microservices.io/index.html
- https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do

References (Web)

- https://searchapparchitecture.techtarget.com/definition/Enterprise-Service-Bus-ESB
- https://www.edureka.co/blog/what-is-microservices/
- https://cloud.google.com/architecture/microservices-architecture-introduction

References (Books)

- Building Microservices: Designing Fine-Grained Systems by Sam Newman, Publisher: O’Reilly