

SOFTWARE ENGINEERING (Week-5)

USAMA MUSHARAF

*LECTURER (Department of Computer
Science)*

FAST-NUCES PESHAWAR

CONTENTS OF WEEK # 5

- **Distributed Software Architecture (Cont...)**
 - SOA
 - Microservices
 - Cloud Architecture
- **Event Driven Software Architecture**

CATEGORIES OF ARCHITECTURAL STYLES

- Hierarchical Software Architecture
 - Layered
- Data Flow Software Architecture
 - Pipe and Filter
 - Batch Sequential
- Data Centered Software Architecture
 - Black board
 - Shared Repository
- Component-Based Software Architecture
- **Distributed Software Architecture**
 - Client Server
 - Peer to Peer
 - REST
 - SOA
 - Microservices
 - Cloud Architecture
- **Event Based Software Architecture**



SERVICE ORIENTED ARCHITECTURE (SOA)



SERVICE ORIENTED ARCHITECTURE (SOA)

- A service-oriented architecture (SOA) is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network.
- The principles of service-orientation are independent of any product, vendor or technology.

WHAT IS SERVICE?

A service is a self-contained, self-describing and modular piece of software that performs a specific business function such as validating a credit card or generating an invoice.

- The term self-contained implies services include all that is needed to get them working.
- Self-describing means they have interfaces that describe their business functionalities.
- Modular means services can be aggregated to form more complex applications.

EXAMPLE

A single service provides a collection of capabilities, often grouped together within a functional context as established by business requirements.

For example, the functional context of the service below is account. Therefore, this service provides the set of operations associated with a customer's account

Account
<ul style="list-style-type: none">• Balance• Withdraw• Deposit

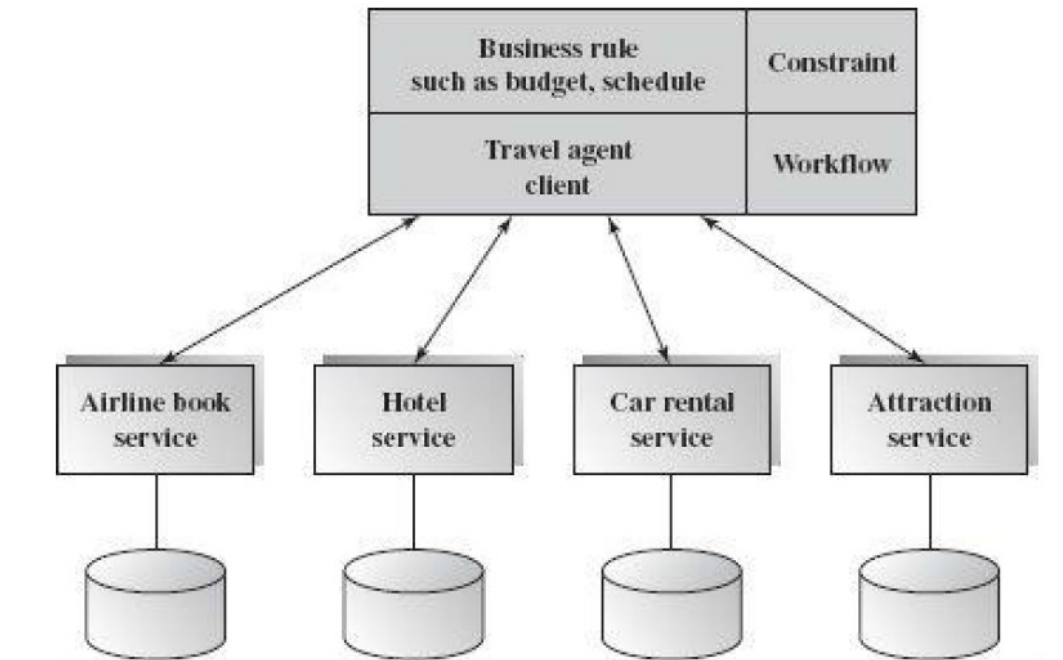
BANKING EXAMPLE

- Imagine that several areas of banking applications will deal with the current balance of an existing customer.
- More than often, the “get current balance” functionality is repeated in various applications within a banking environment.
- This gives rise to a redundant programming scenario.
- The focus should be toward finding this sort of common, reusable functionality and implement it as a service, so
 - that all banking applications can reuse the service as and when necessary.

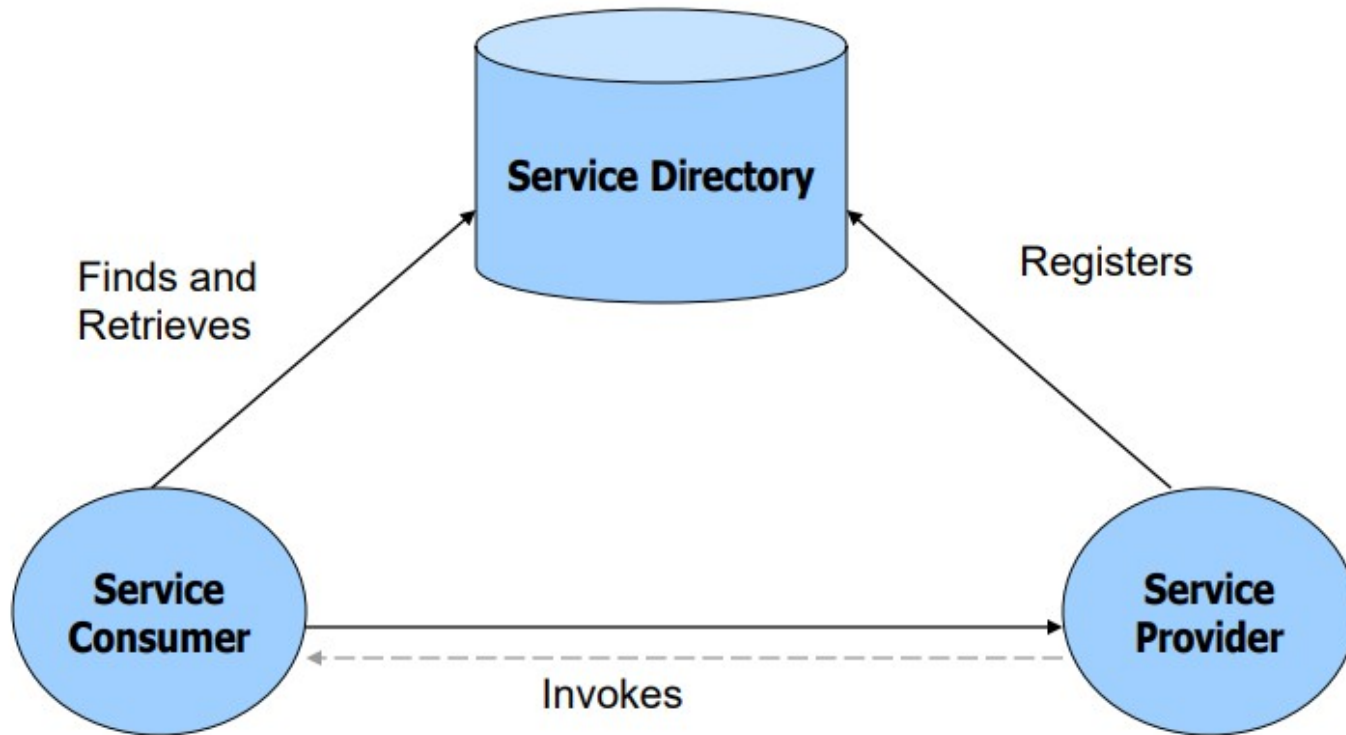
EXAMPLE

An online travel agency system

- It consists of four existing web services:
 - airline reservation,
 - car rental,
 - hotel reservation,
 - and attraction reservation.

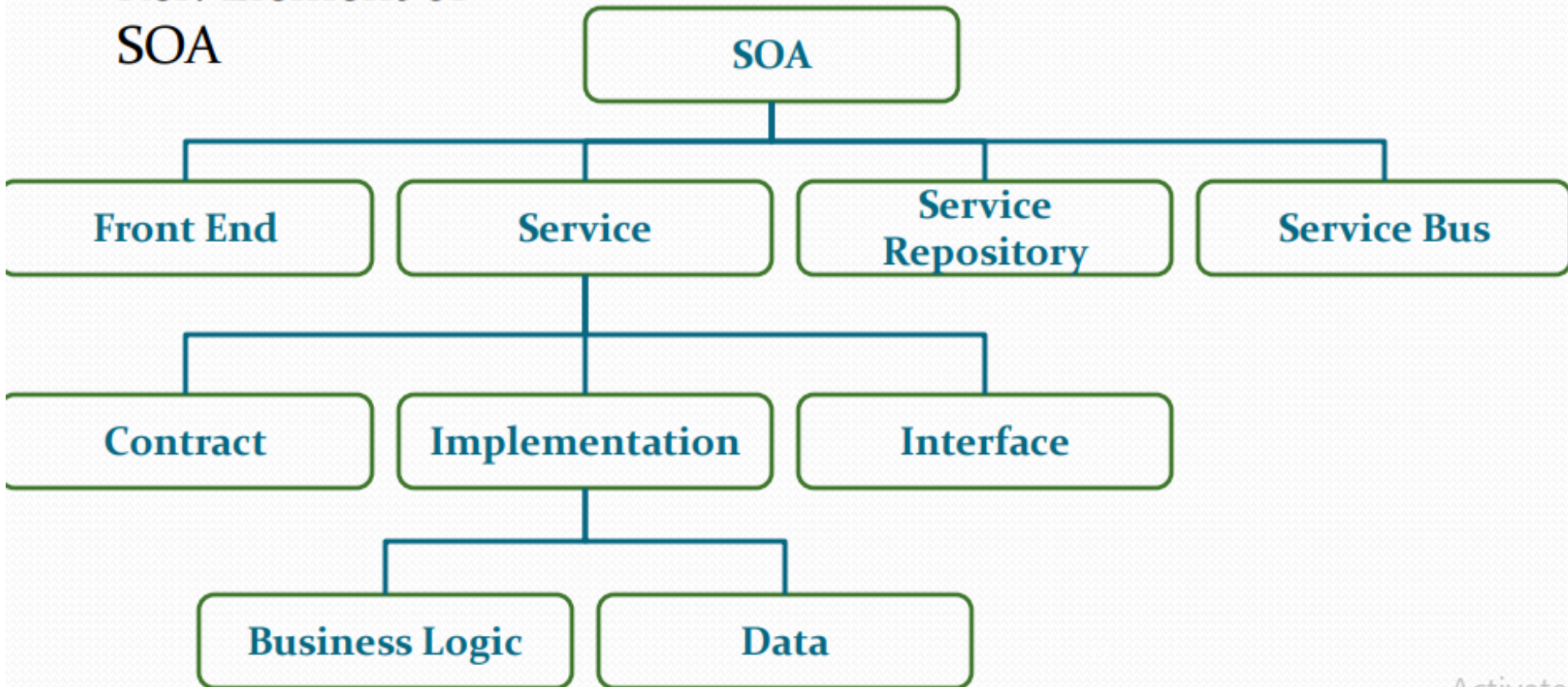


SOA ARCHITECTURE



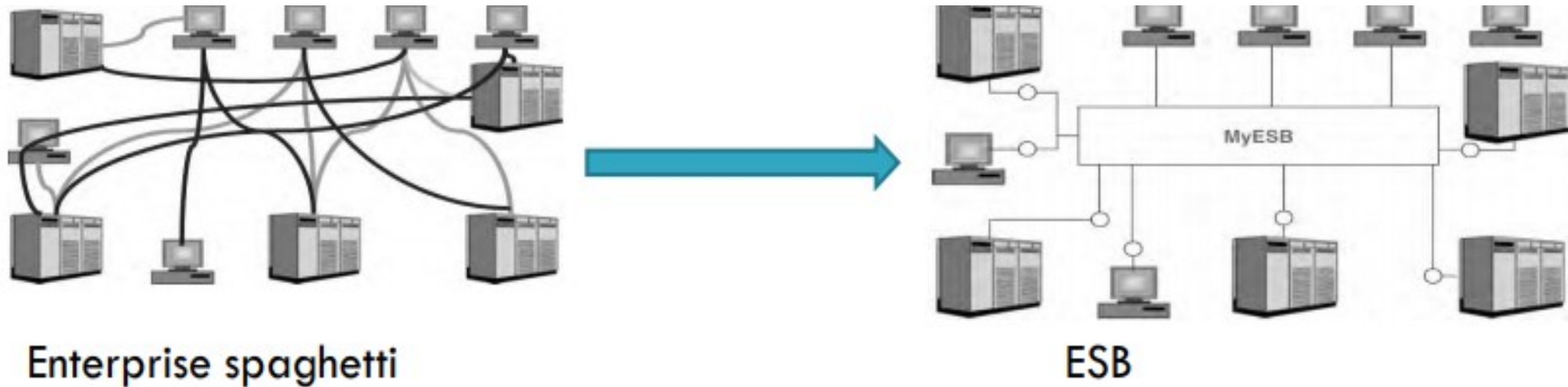
COMPONENTS OF SOA

Ref: Element of
SOA



ESB

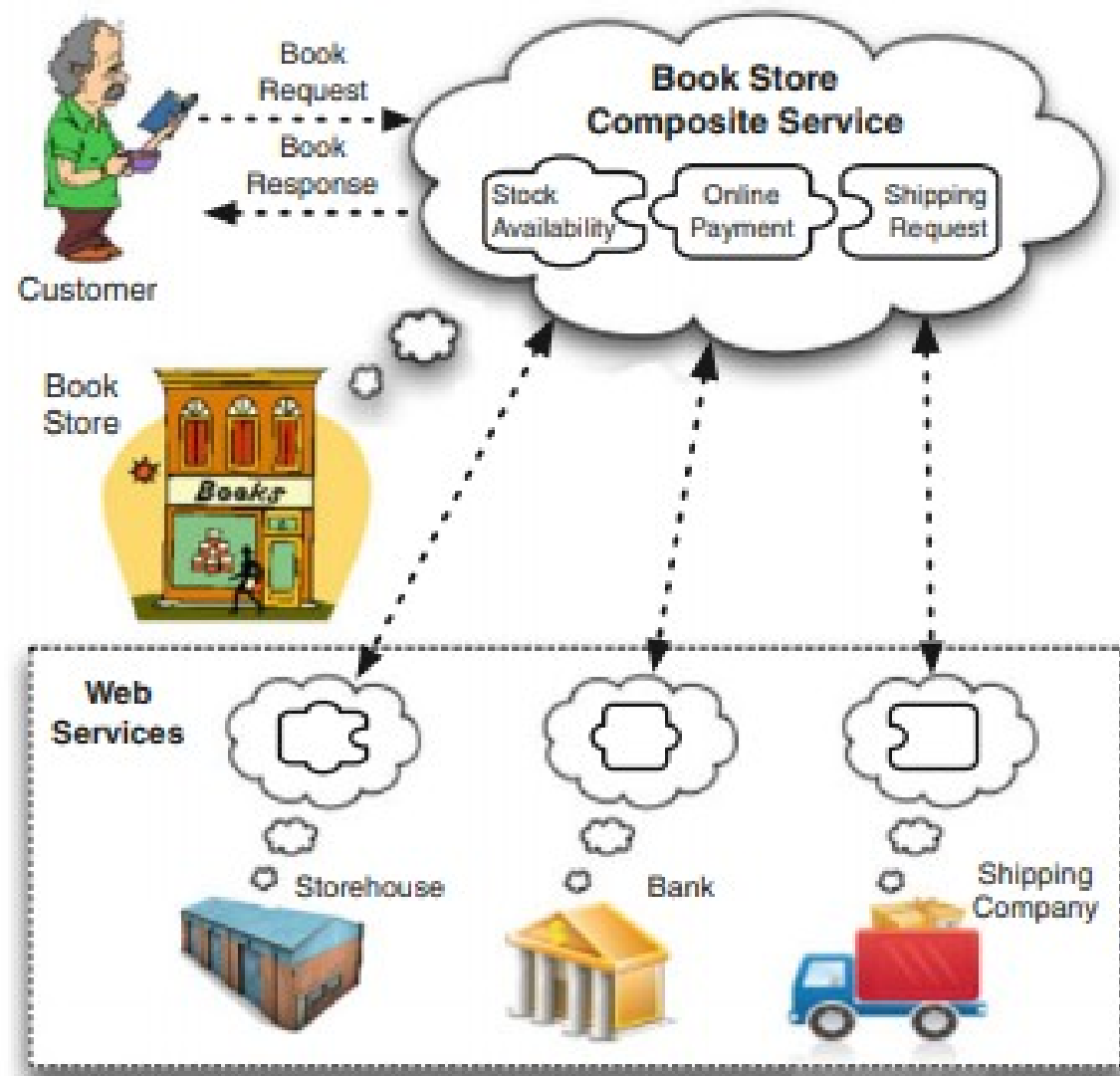
- Enterprise Service Buses (ESBs) build on MOM (message-oriented middleware) to provide a flexible, scalable, standards-based integration technology for building a loosely coupled, highly-distributed SOA





SERVICE COMPOSITION

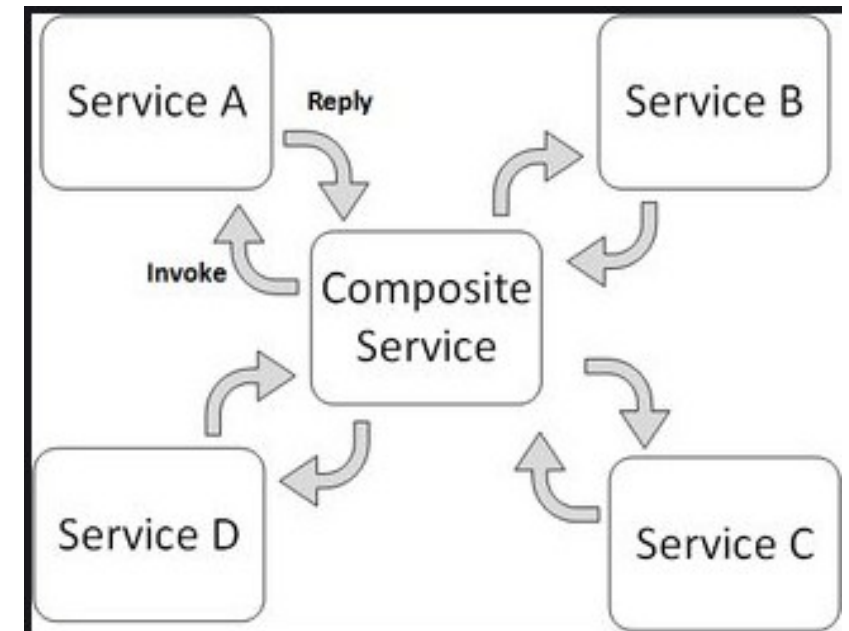




An example of Web service composition

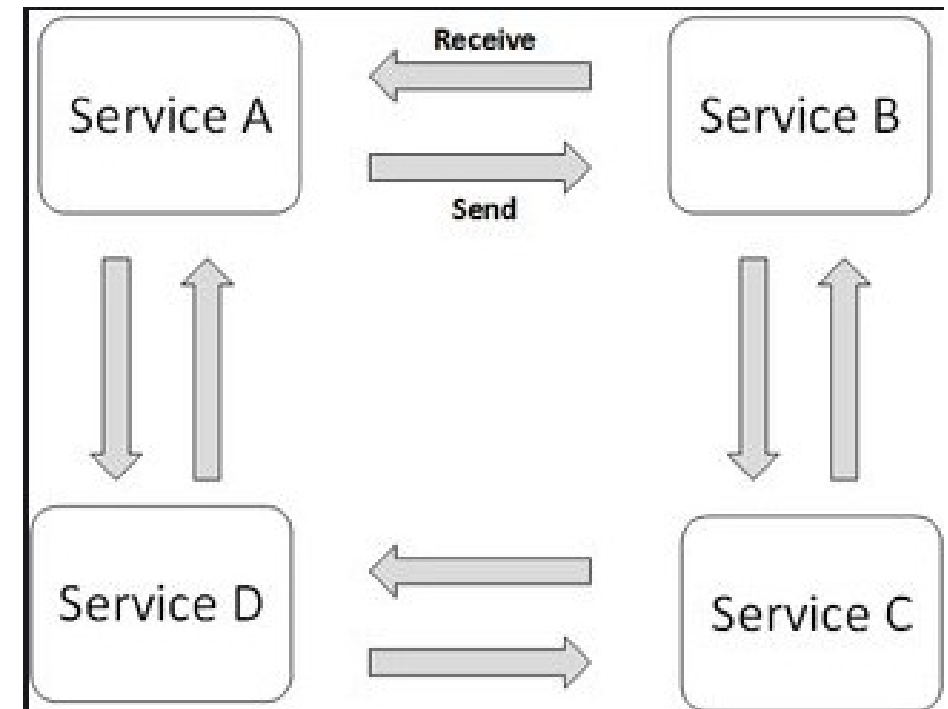
SERVICE ORCHESTRATION VS SERVICE CHOREOGRAPHY

- Orchestration comes from orchestras; in an orchestra there is a conductor leading it, and there are musicians who play their instruments following the instructions of the conductor.
- The musicians know their role; they know how to play their instruments, but not necessarily directly interact with each other. That is, in an orchestra, there is a central system (the conductor) that coordinates and synchronizes the interactions of the components to perform a coherent piece of music.



SERVICE ORCHESTRATION VS SERVICE CHOREOGRAPHY

- Choreography is related to dance; on a dancing stage there are just dancers, but they know what they have to do and how to interact with the other dancers; each dancer is in charge of being synchronized with the rest.



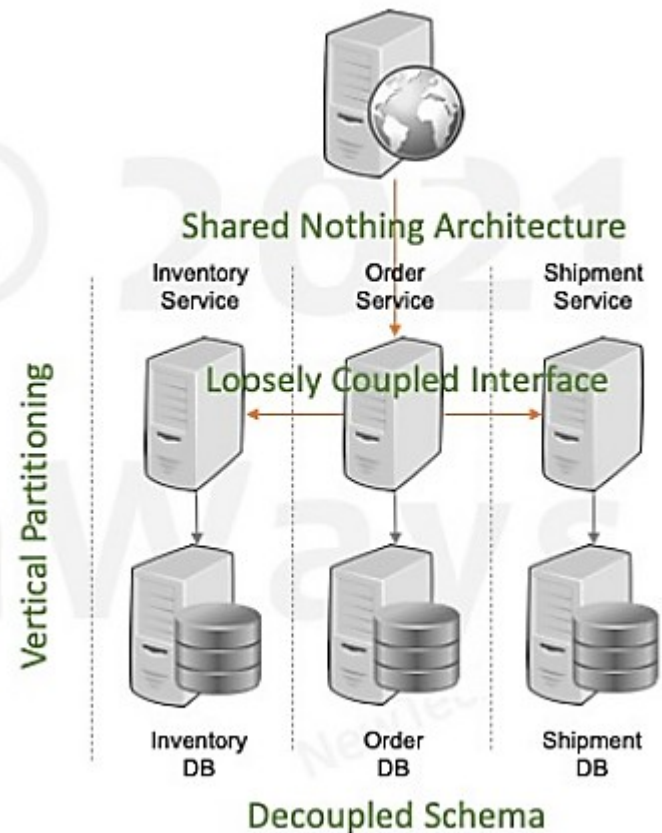


MICROSERVICES



MICROSERVICES

- Microservices are small, individually deployable services performing different operations.
- Variant of SOA



Frequent Deployment

Independent Deployment

Independent Development

Independent Services

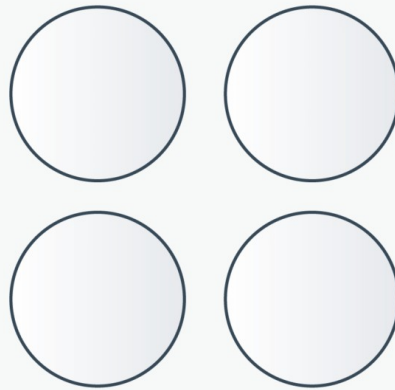
SOA VS MICROSERVICES

Monolithic vs. SOA vs. Microservices



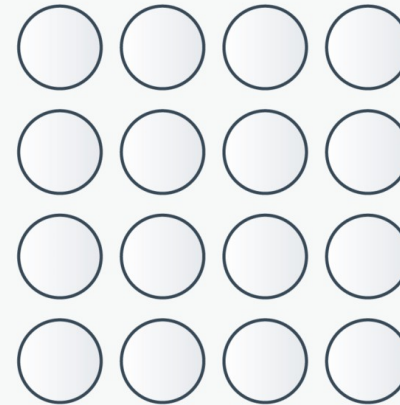
Monolithic

Single Unit



SOA

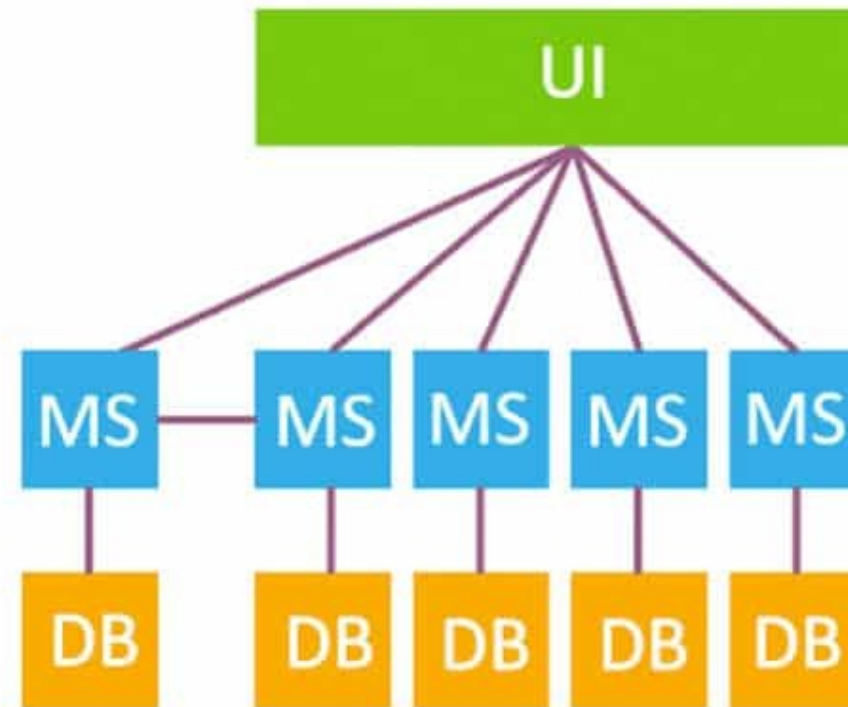
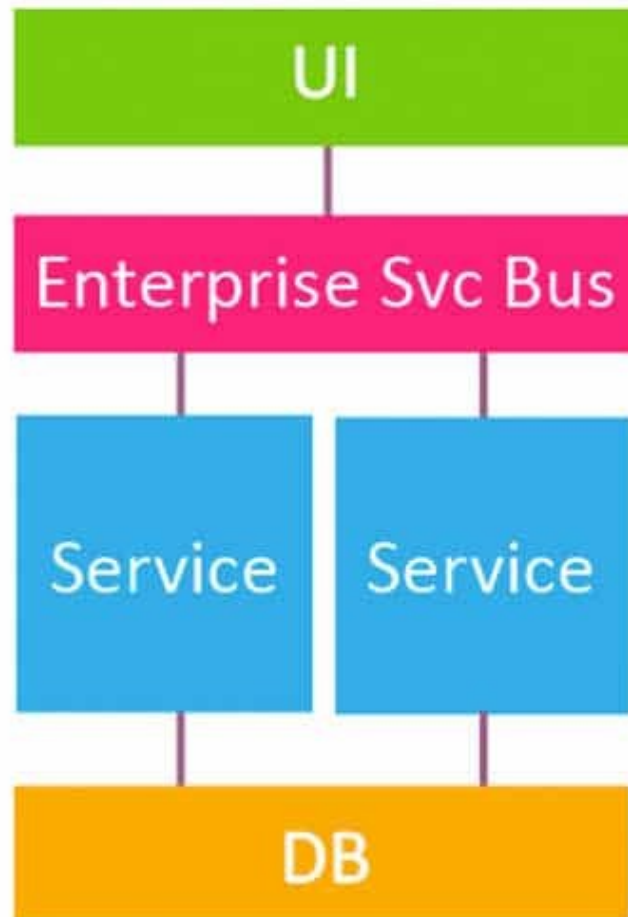
Coarse-grained



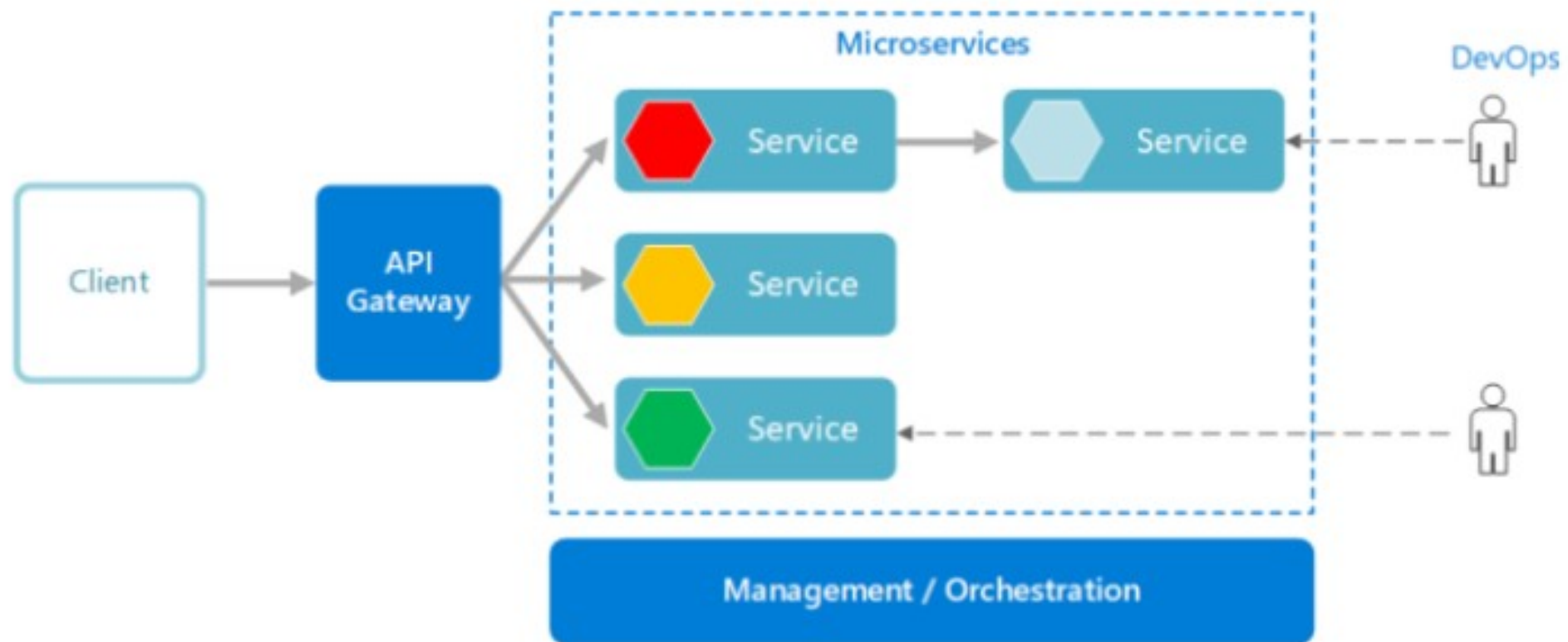
Microservices

Fine-grained

SOA VS MICROSERVICES



HOW DOES MICROSERVICES ARCHITECTURE WORK?



COMMUNICATION IN MICROSERVICES

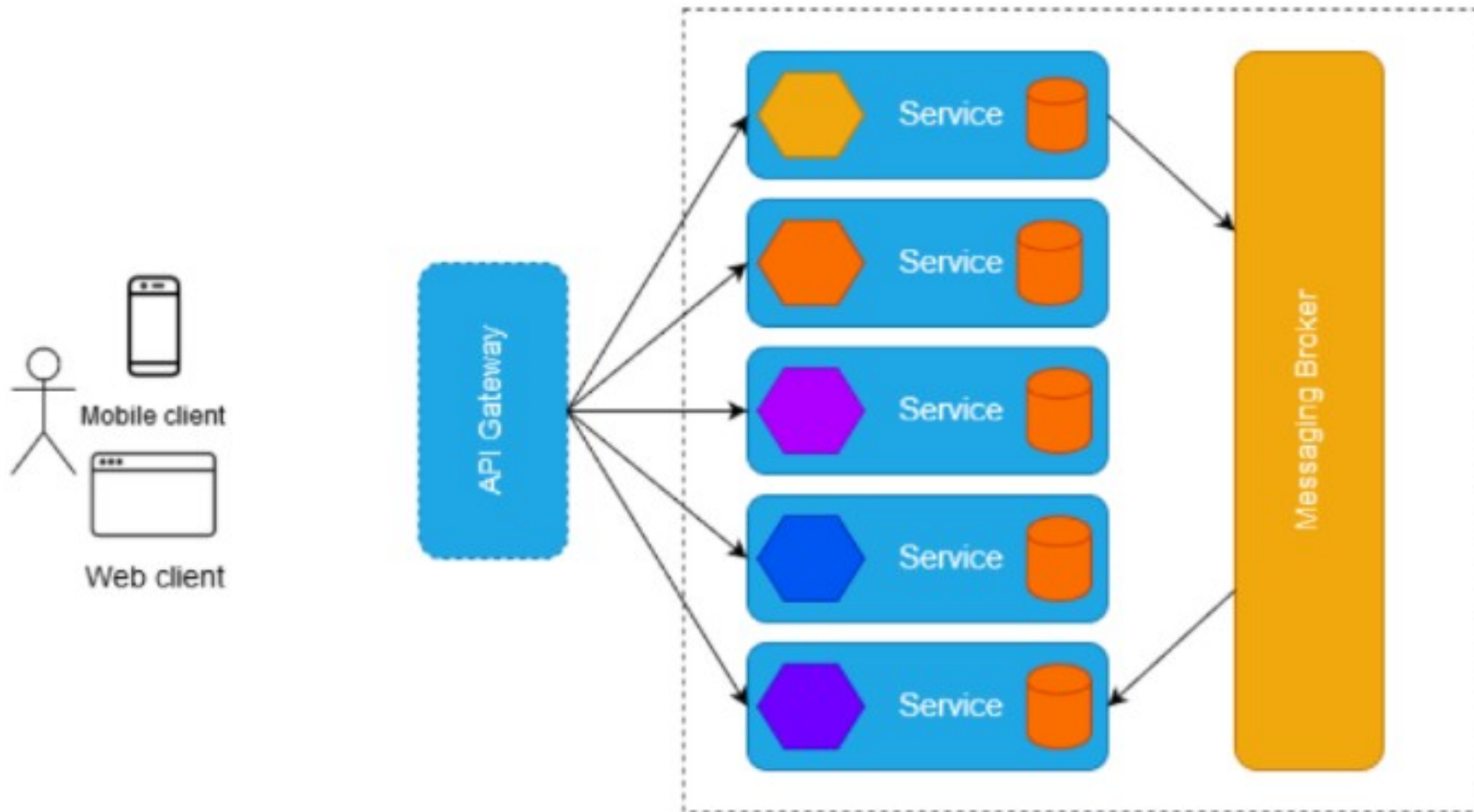
Synchronous Messages:

In the situation where clients wait for the responses from a service, Microservices usually tend to use **REST (Representational State Transfer)** as it relies on a stateless, client-server, and the **HTTP protocol**. This protocol is used as it is a distributed environment each and every functionality is represented with a resource to carry out operations

Asynchronous Messages:

In the situation where clients do not wait for the responses from a service, Microservices usually tend to use protocols such as **AMQP, MQTT**.

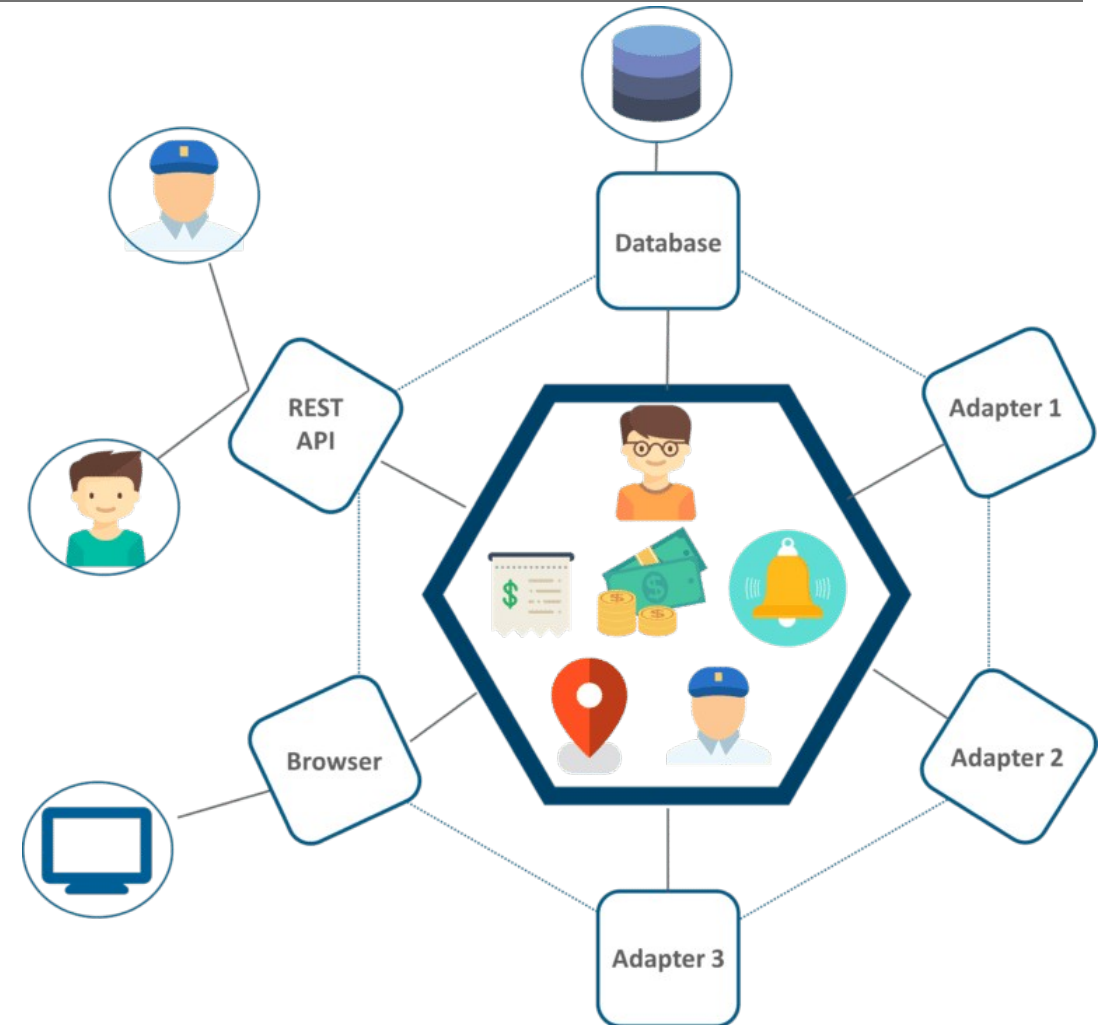
REFERENCE ARCHITECTURE OF MICROSERVICES



UBER'S PREVIOUS ARCHITECTURE

- A REST API is present with which the passenger and driver connect.
- Three different adapters are used with API within them, to perform actions such as billing, payments, sending emails/messages that we see when we book a cab.
- A MySQL database to store all their data.

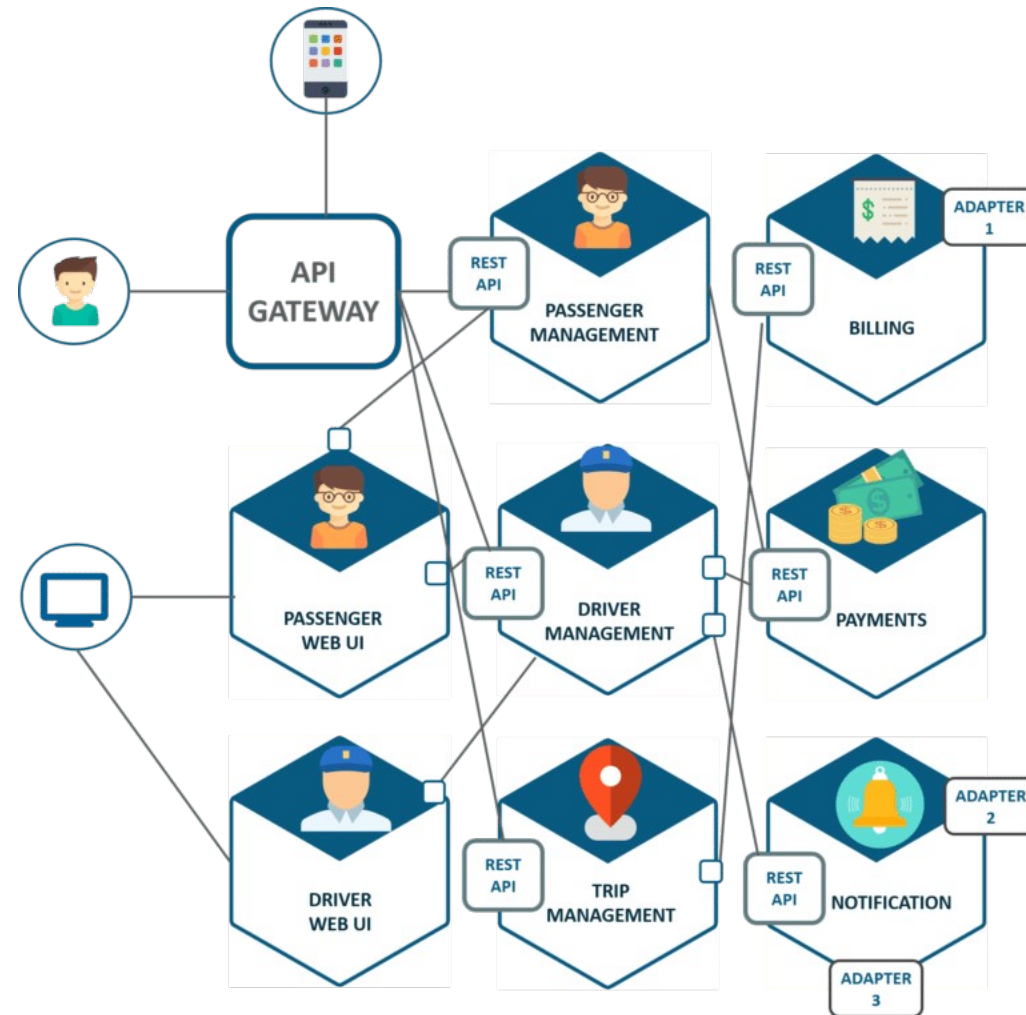
All the features such as passenger management, billing, notification features, payments, trip management, and driver management were composed within a single framework.



PROBLEM IN UBER'S ARCHITECTURE

- All the features had to be re-built, deployed and tested again and again to update a single feature.
- Fixing bugs became extremely difficult in a single repository as developers had to change the code again and again.
- Scaling the features simultaneously with the introduction of new features was quite tough to be handled together.

SOLUTION IS MICROSERVICES ARCHITECTURE



SOLUTION

- The units are individual separate deployable units performing separate functionalities.
- For Example: If you want to change anything in the billing Microservices, then you just have to deploy only billing Microservices and don't have to deploy the others.
- All the features were now scaled individually i.e. The interdependency between each and every feature was removed.

DECOMPOSITION OF MICROSERVICES

There are some Prerequisite of decomposition of microservices.

Services must be cohesive.

- A service should implement a small set of strongly related functions.

Services must be loosely coupled

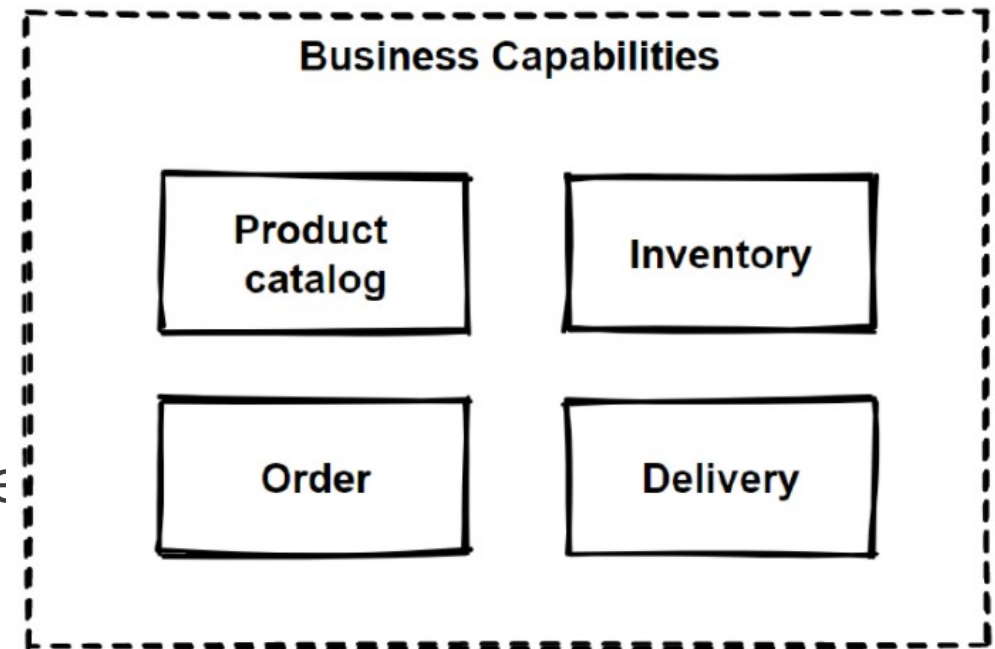
- Each service as an API that encapsulates its implementation.

DECOMPOSITION OF MICROSERVICES

Decompose by Business Capability” pattern offers that;

- Define services corresponding to business capabilities.
- A business capability is a concept from business architecture modeling.
- A business service should generate value.

For example; Order Management is responsible for orders, Customer Management is responsible for customers.





CLOUD ARCHITECTURE



CLOUD COMPUTING

- Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user.

Deploying an eCommerce Website on-premises (aka the old way)

Assumes you don't have a private cloud, or don't have enough capacity

Activity:

Timeline:

1) Purchase hardware

4-12 weeks

2) Install and build

4-8 weeks

3) Acceptance testing

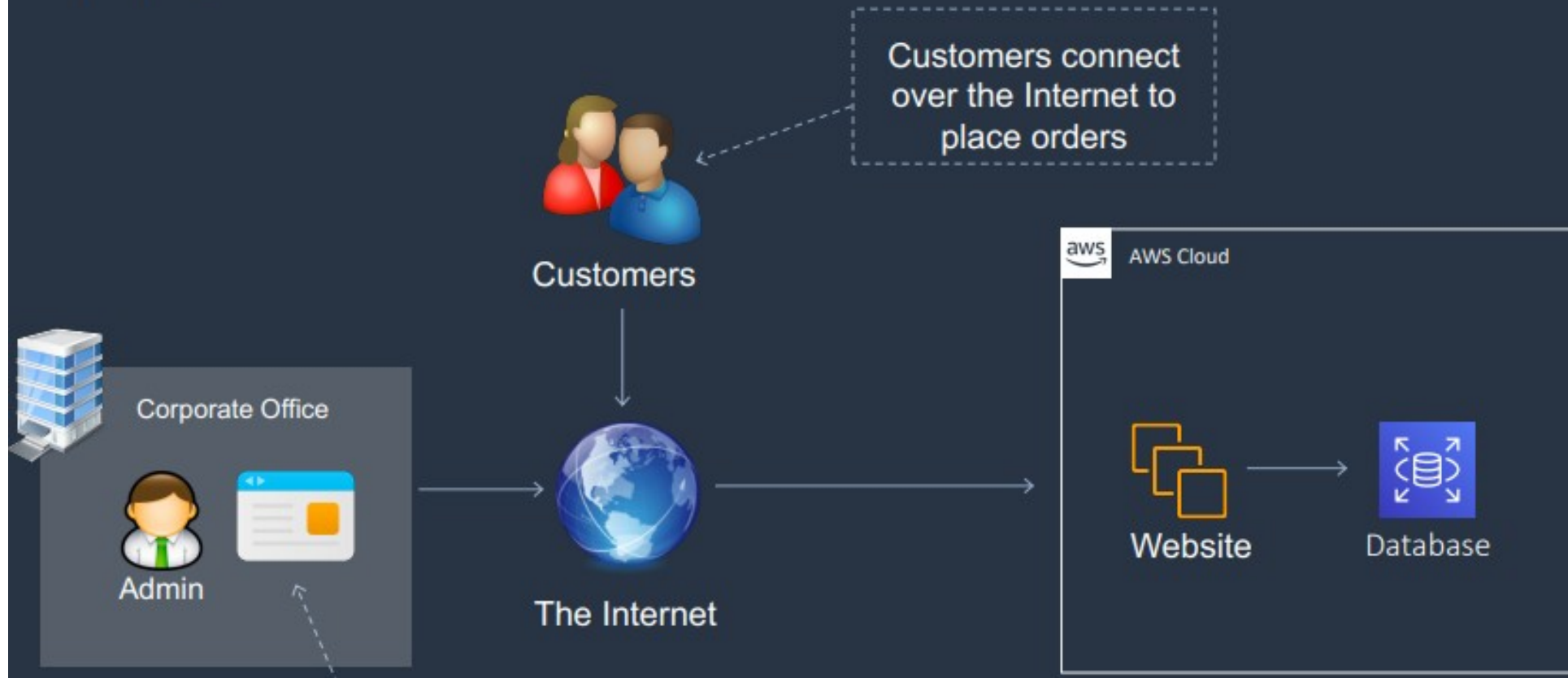
2-4 weeks

1) Handover to operations

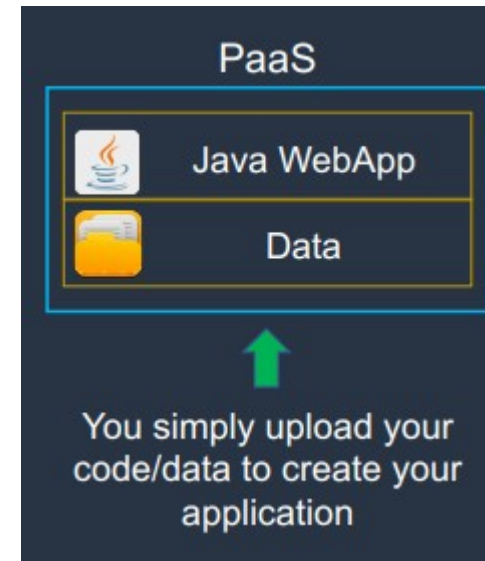
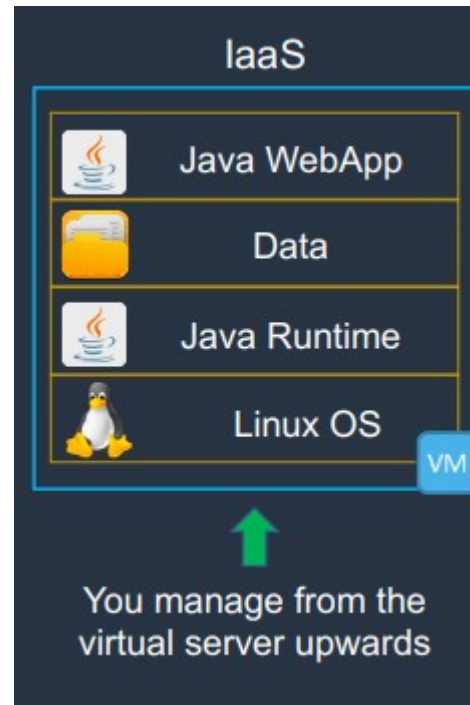
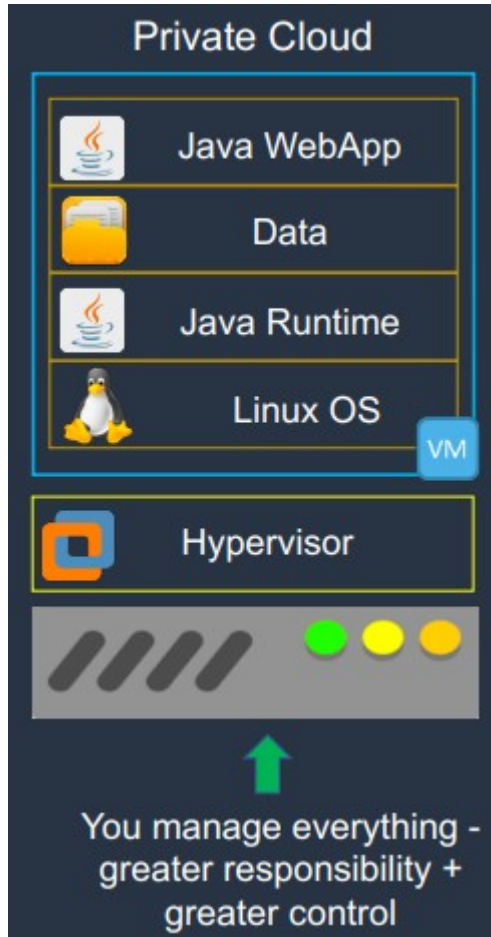
1-2 weeks

3-6 months !

Deploying an eCommerce Website in the Cloud



CLOUD SERVICE MODELS: COMPARISON





EVENT BASED SOFTWARE ARCHITECTURE



EVENT DRIVEN ARCHITECTURE


- Event-driven architecture refers to a system that exchange information between each other through the production and consumption of events.
- The main purpose of this type of communication architecture is to provide a decoupling between the event/message, the publishers/producers, and the subscribers/customers.
- These are very popular architectures in distributed applications.

EXAMPLE


- Uber's
- Fire Alarming System
- Used to enforce integrity constraints in **database management systems** (called triggers).

SYNCHRONOUS VS ASYNCHRONOUS

- Synchronous
- Asynchronous



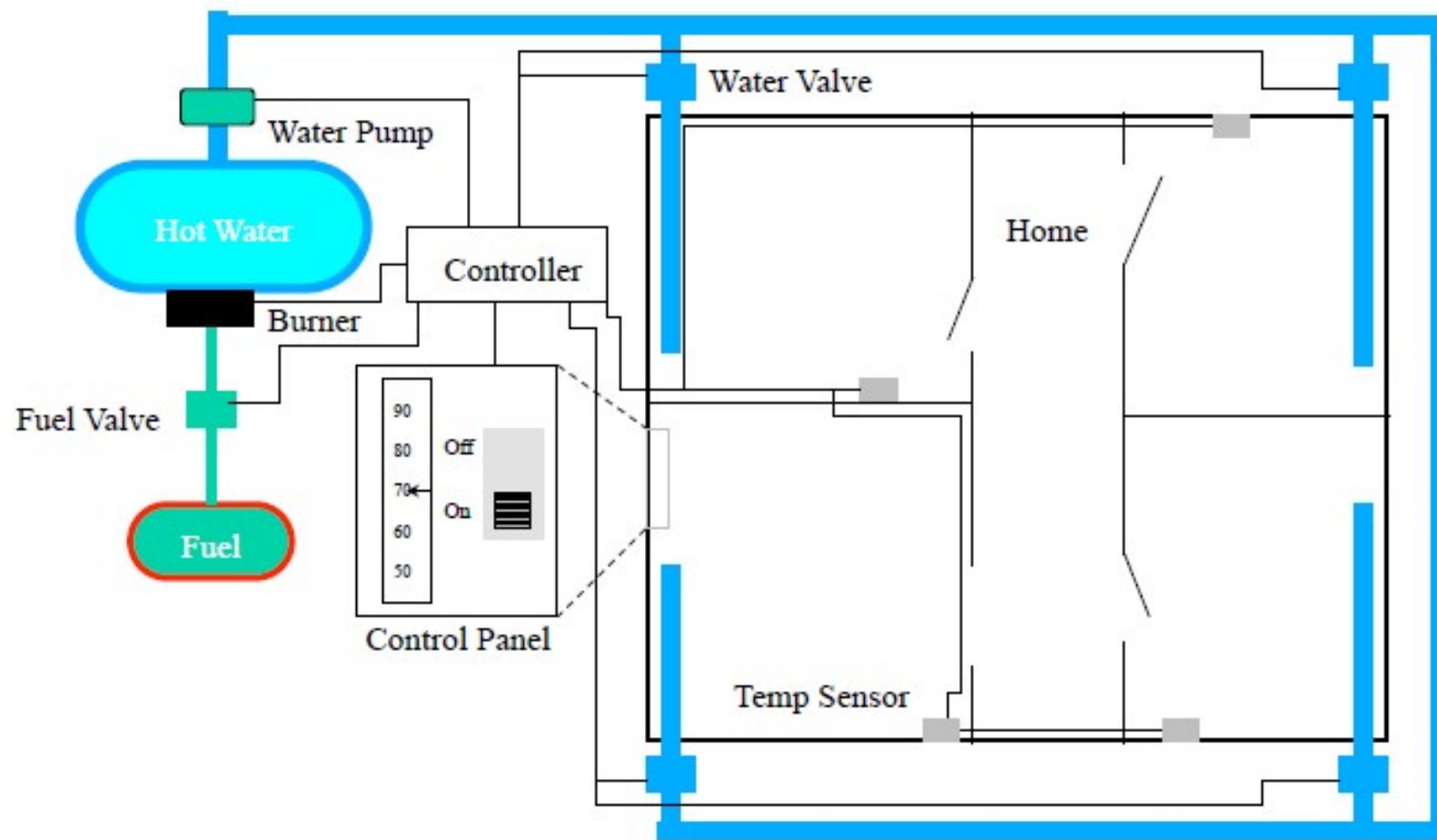
IMPLICIT ASYNCHRONOUS COMMUNICATION SOFTWARE ARCHITECTURE



IMPLICIT ASYNCHRONOUS COMMUNICATION SOFTWARE ARCHITECTURE

Instead of invoking a procedure directly

- A **component** can announce (or broadcast) one or more events.
- When an event is announced, the broadcasting system (**connector**) itself invokes all of the procedures that have been registered for the event.



IMPLICIT ASYNCHRONOUS COMMUNICATION SOFTWARE ARCHITECTURE

- Non-buffered Event-Based Implicit Invocations
- Buffered Message-Based Software Architecture



NON-BUFFERED EVENT-BASED IMPLICIT INVOCATIONS



NON-BUFFERED EVENT-BASED IMPLICIT INVOCATIONS

The non-buffered event-based implicit invocation architecture breaks the software system into two partitions:

- Event sources and
 - Event listeners.
-
- The event registration process connects these two partitions. There is no buffer available between these two parties.

NON-BUFFERED EVENT-BASED IMPLICIT INVOCATIONS

- In the event-based implicit invocations (non-buffered) each object keeps its own dependency list.
- Any state changes of the object will impact its dependents.

BENEFITS:

- Reusability of components: It is easy to plug in new event handlers without affecting the rest of the system.
- System maintenance and evolution: Both event sources and targets are easy to update.
- Parallel execution of event handlings is possible.

LIMITATIONS:

- It is difficult to test and debug the system since it is hard to predict and verify responses and the order of responses from the listeners.
 - The event trigger cannot determine when a response has finished or the sequence of all responses.
- There is tighter coupling between event sources and their listeners than in message queue-based or message topic-based implicit invocation.



BUFFERED MESSAGE-BASED SOFTWARE ARCHITECTURE



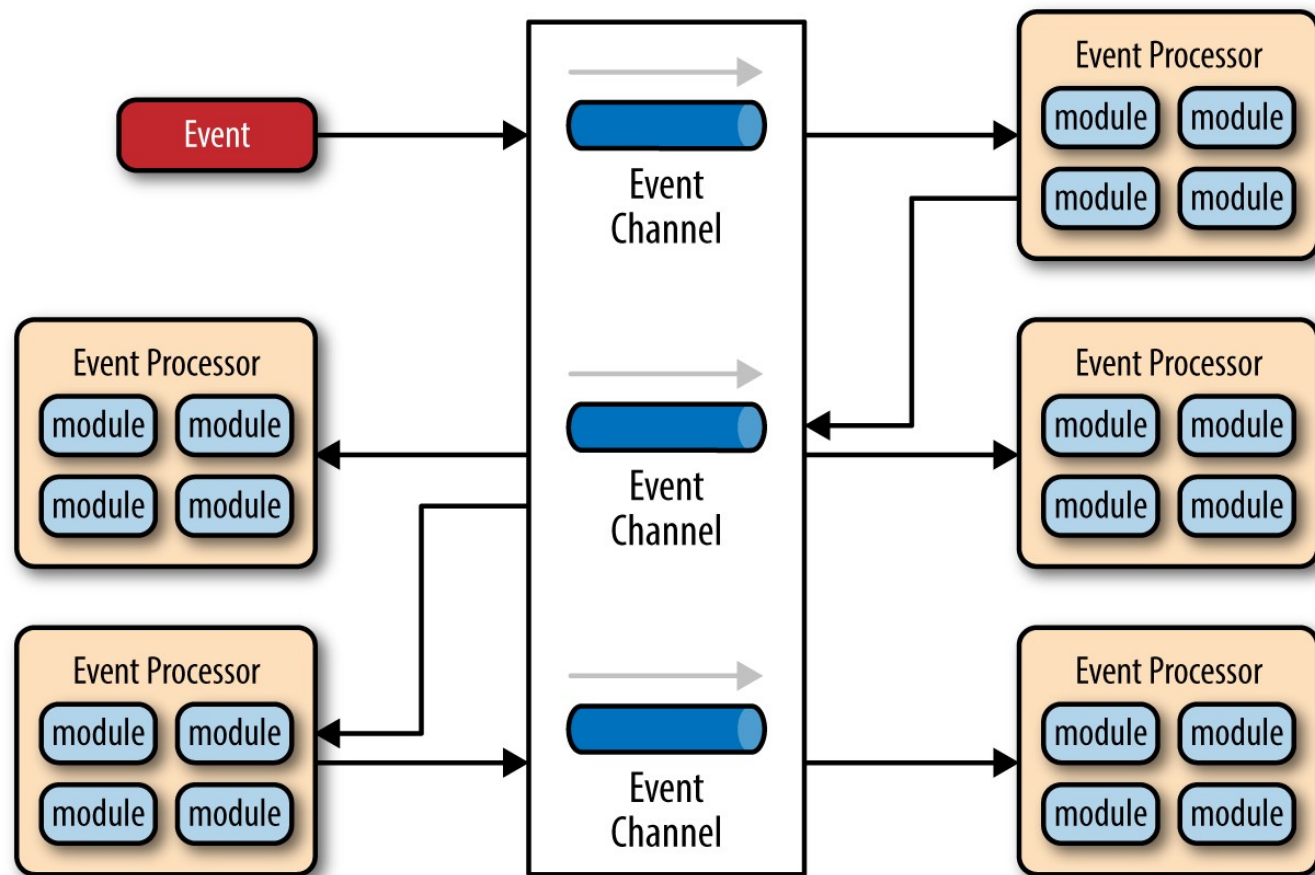
THE BUFFERED MESSAGE-BASED SOFTWARE ARCHITECTURE

It breaks the software system into three partitions:

- message producers,
- Message consumers, and
- message service providers.

They are connected asynchronously by either a message queue or a message topic.

THE BUFFERED MESSAGE-BASED SOFTWARE ARCHITECTURE



THE BUFFERED MESSAGE-BASED SOFTWARE ARCHITECTURE

- A messaging client can produce and send messages to other clients, and can also consume messages from other clients.
- Each client must register with a messaging destination in a connection session provided by a message service provider for creating, sending, receiving, reading, validating, and processing messages.

APPLICABLE DOMAINS OF MESSAGE-BASED ARCHITECTURE:

- Suitable for a software system where the communication between a producer and a receiver requires buffered message-based asynchronous implicit invocation for performance and distribution purposes.
- The provider wants components that function independently of information about other component interfaces so that components can be easily replaced.

BENEFITS:

- Anonymity: provides high degree of anonymity between message producer and consumer.
- Concurrency: supports concurrency both among consumers and between producer and consumers.
- Scalability

BENEFITS

- Provides strong support for **reuse** since any component can be introduced into a system simply by registering it for the events of that system.
- **Eases system evolution** since components may be replaced by other components without affecting the interfaces of other components in the system.

LIMITATIONS:

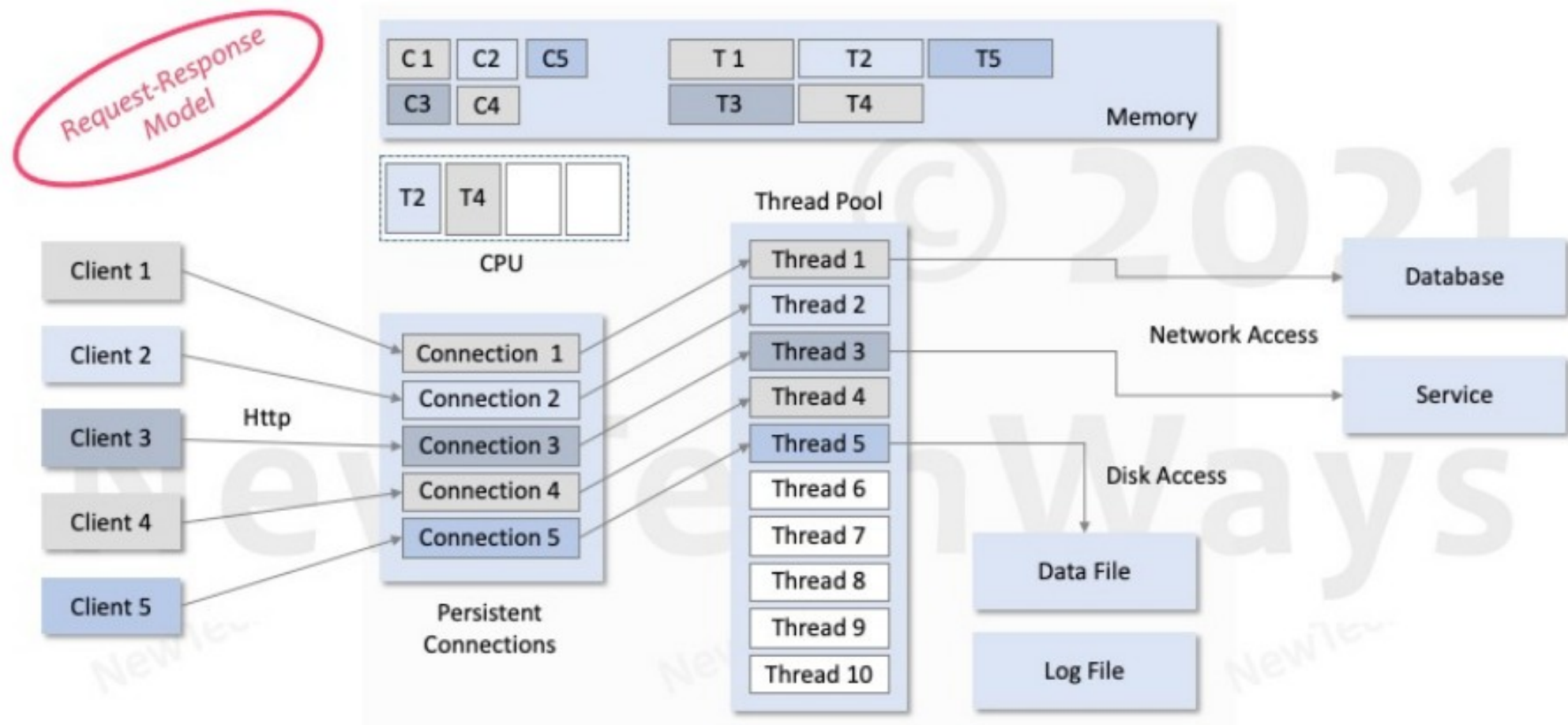
- Capacity limit of message queue:
- Increased complexity of the system design and implementation.

LIMITATIONS

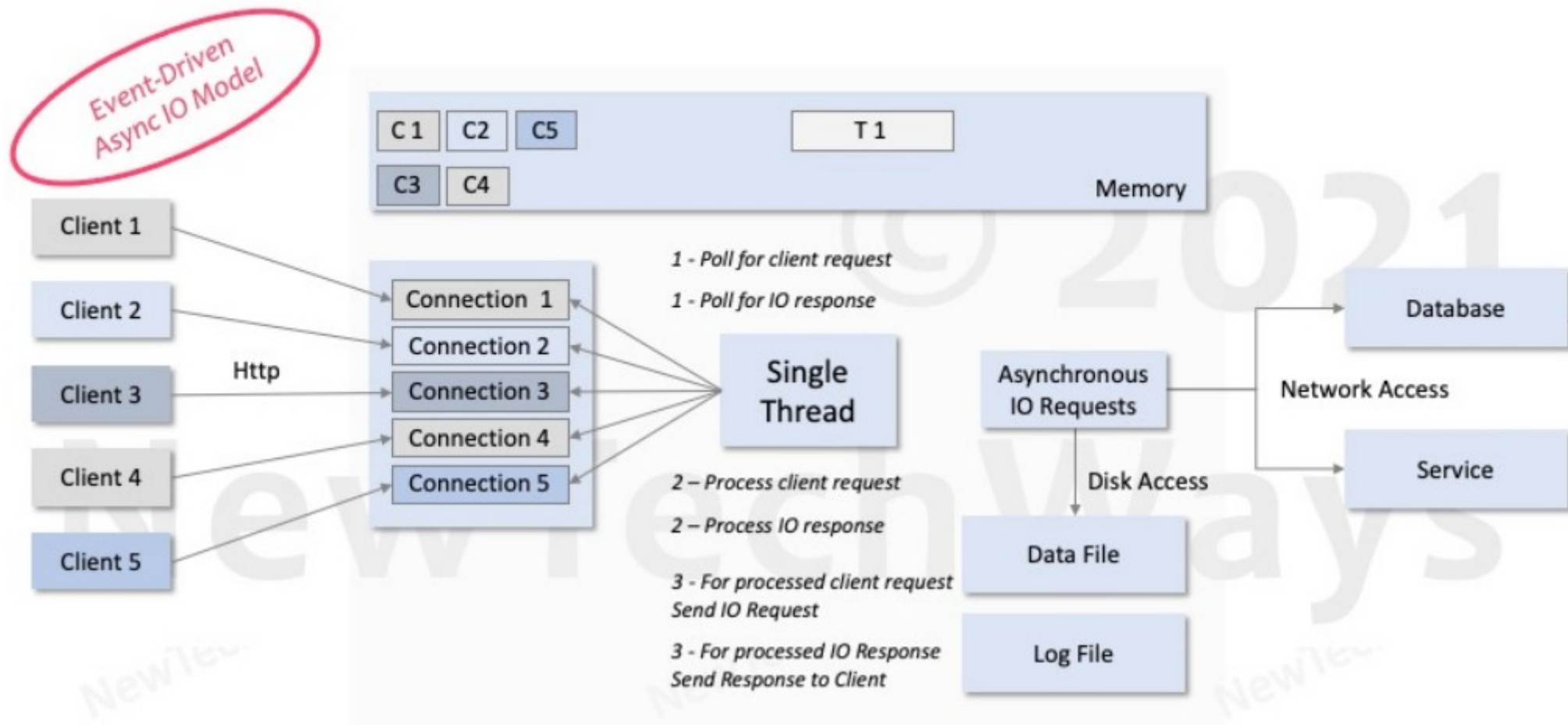
When a component announces an event:

- it has no idea how other components will respond to it,
- it cannot rely on the order in which the responses are invoked
- it cannot know when responses are finished.

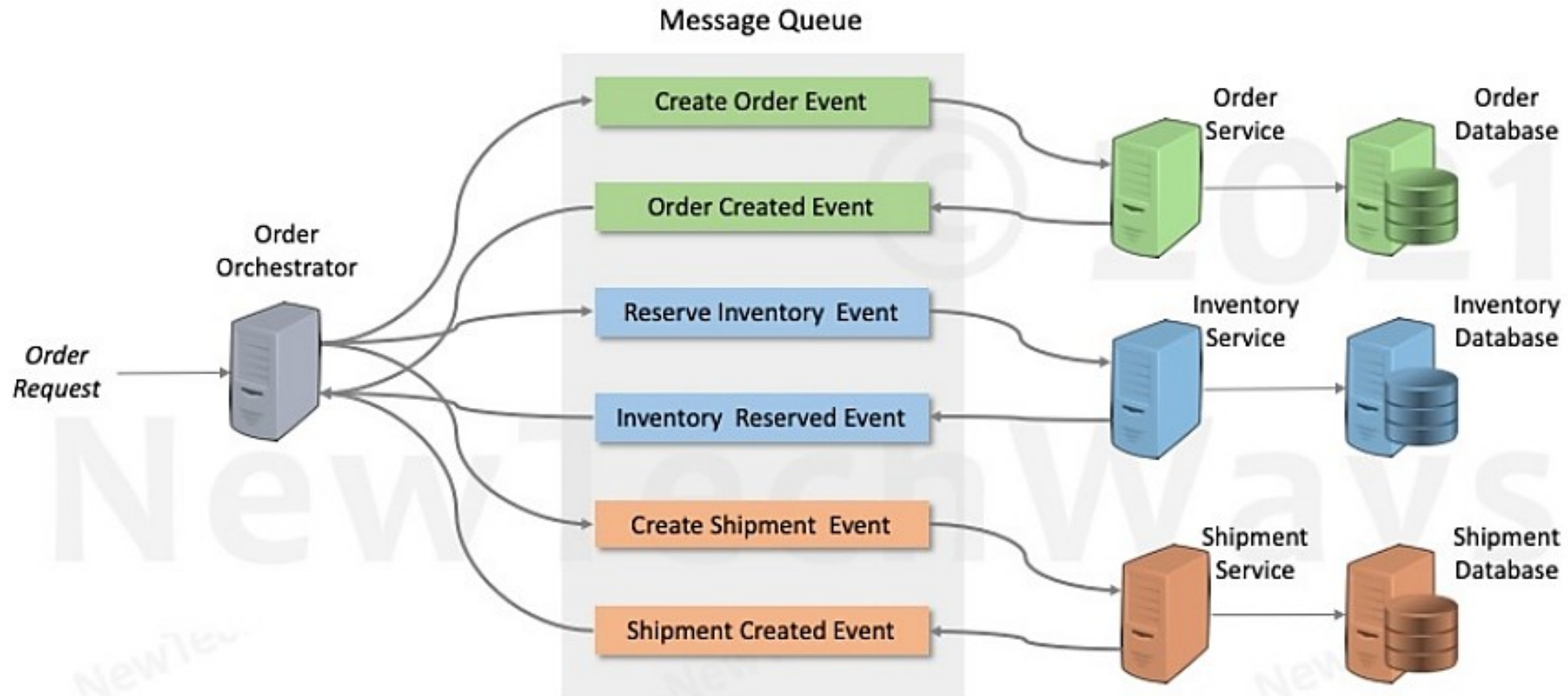
Apache Webserver Architecture



Nginx Architecture



Micro-Services Event Driven Transactions





HAVE A GOOD DAY!