Service oriented computing

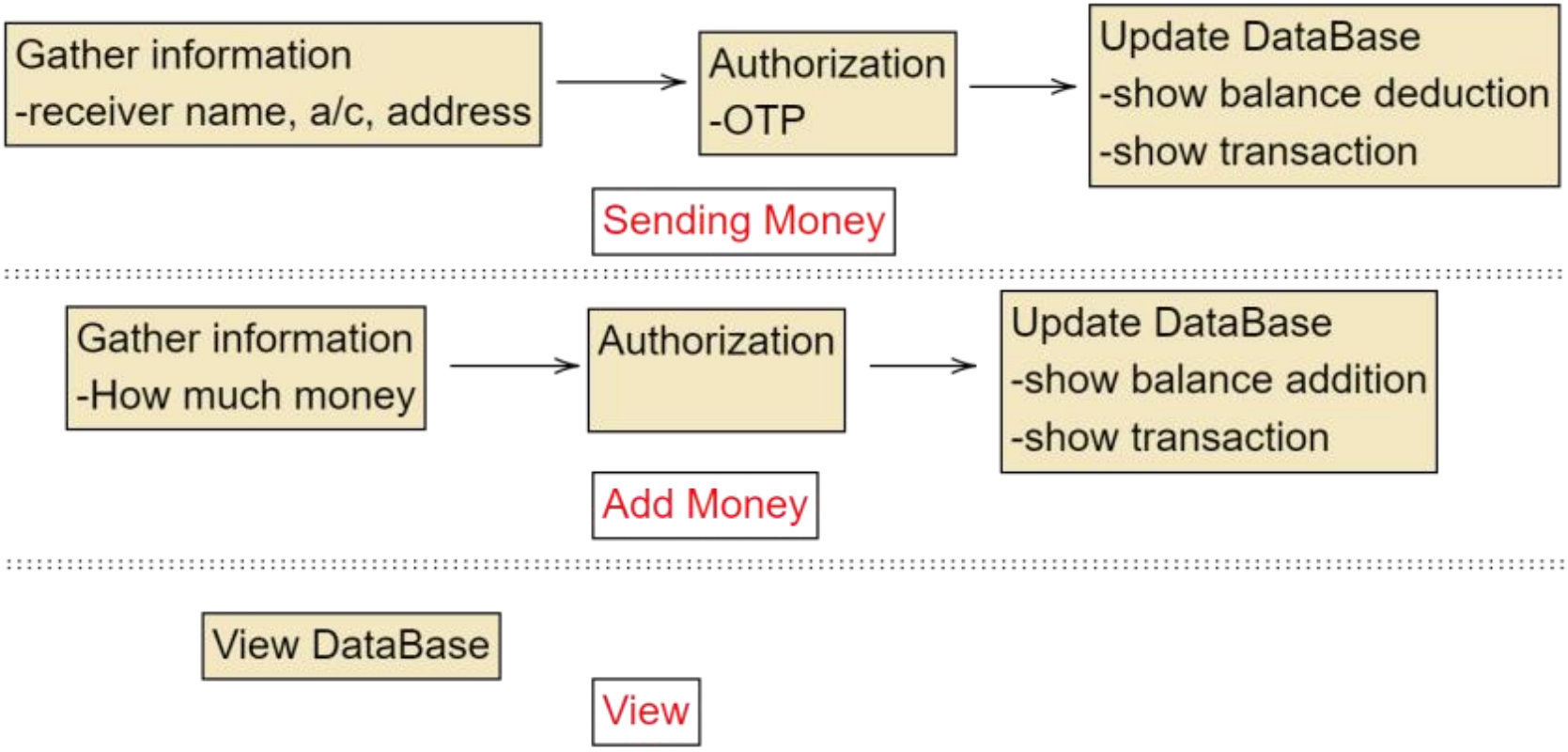
Chapter 3

Outline

Monolithic Application

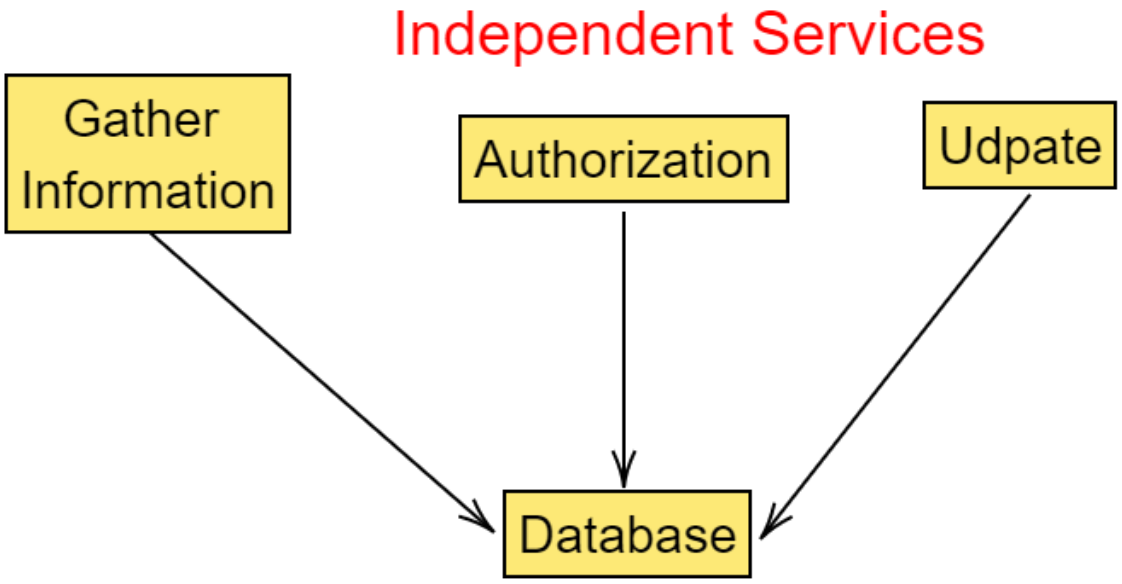
Suppose e-wallet application is design with following feature:

1. Send money
2. Add money
3. View Balance



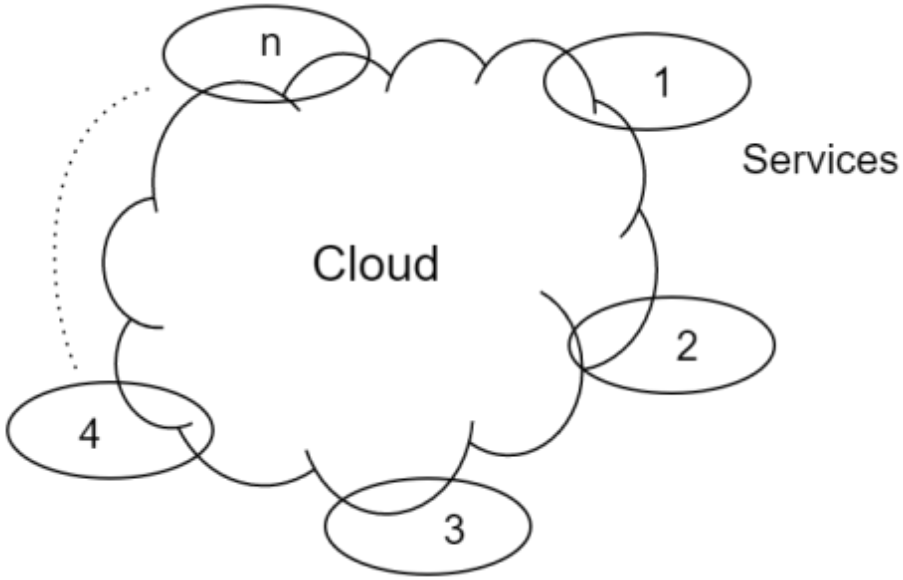
Problem: Monolithic Application

- Need to write code again and again, i.e, **No Code reusability**
- Every block are on chain: if any error occur on block, redeveloped is required in entire system, i.e, **tightly coupled**



Service-oriented computing

- New computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments
- The promise of Service-Oriented Computing is a world of **cooperating services** where application components are assembled with little effort into a network of services that can be **loosely coupled** to create **flexible dynamic business** processes



Service-oriented computing

- There are variety of services or application provide as SaaS
  - User can directly used any service without creating it
- Services have their own structure and process
- If service communicate to each other, i.e, any feature of service can combine with feature of other service to provide user
  - It requires a protocol and standard, i.e, **architecture** called Service-Oriented Architecture (SOA)

What is a service?

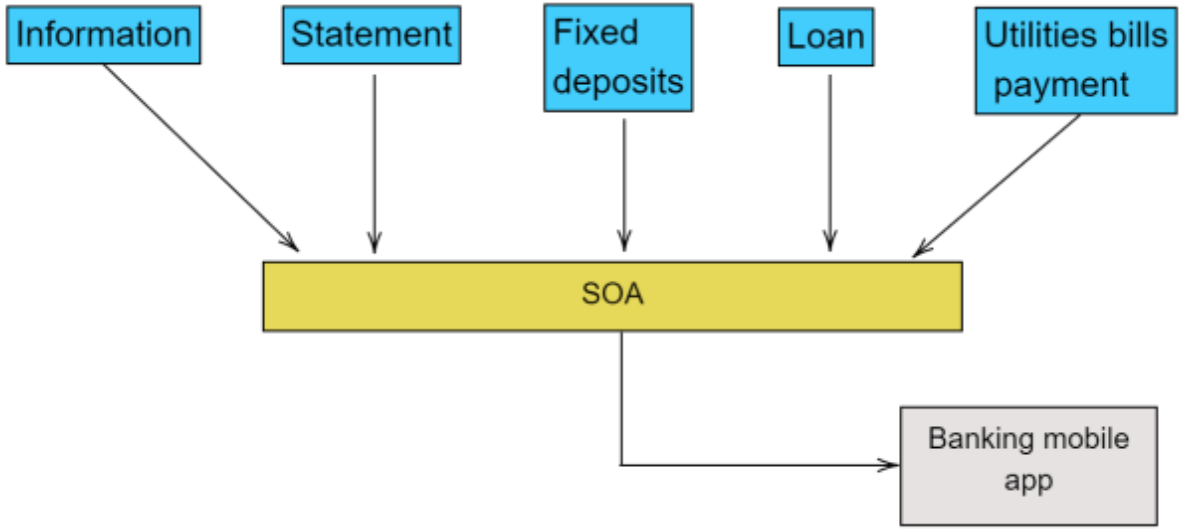
- A service encapsulates as software component that provides a set of coherent and related functionalities
  - that can be **reused** and **integrated** into bigger and more complex applications
- The term service is a general **abstraction** of different technologies and protocols for designing complex and interoperable systems
- Distributed systems are meant to be **heterogeneous**, **extensible**, and **dynamic**
  - By abstracting away from a specific implementation technology and platform, they provide a more efficient way to achieve integration
  - Autonomous components can be more easily reused and aggregated

Service-Oriented Architectures (SOA)

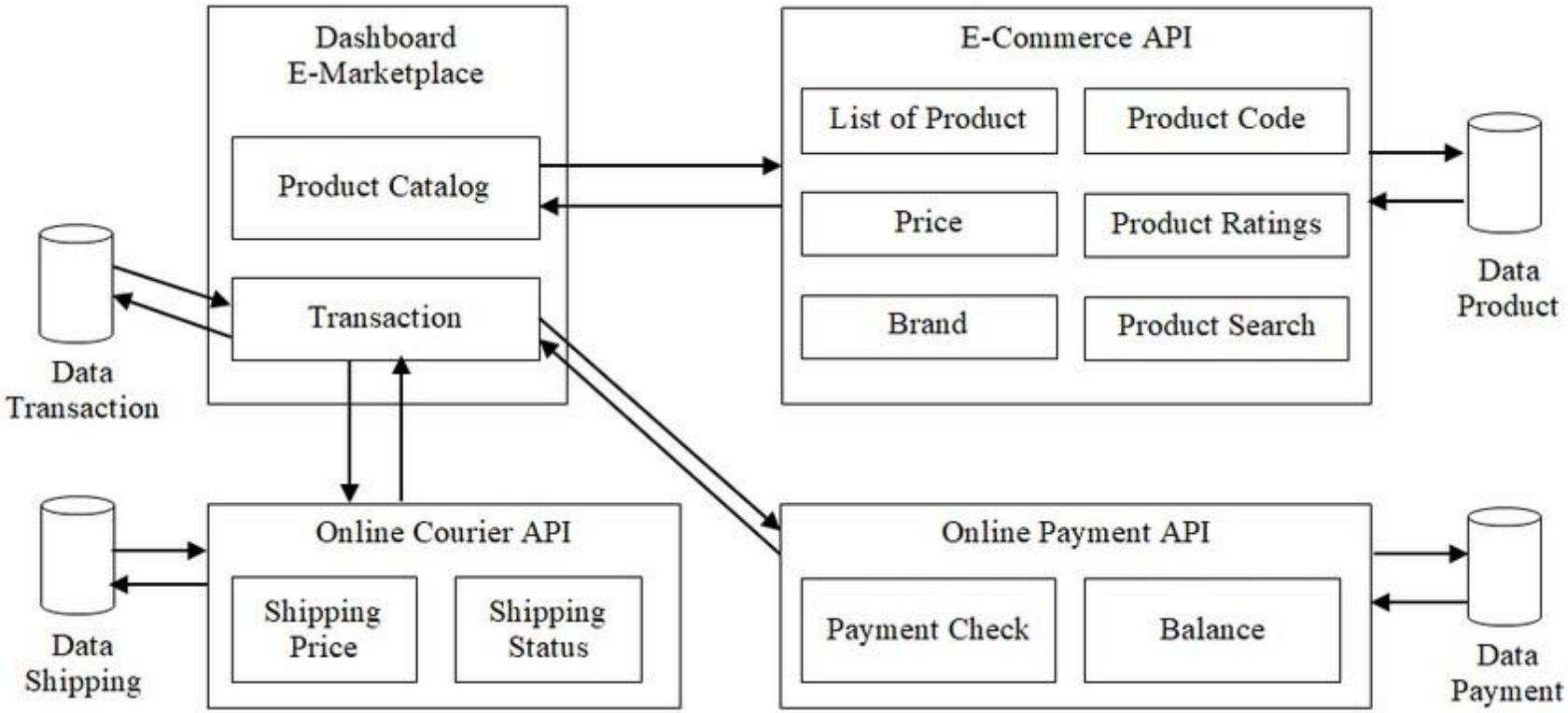
- SOA refers to an architectural approach which includes a collection of interacting services
  - Services are delivered to other application components via communication protocols
- These services are designed to be **loosely coupled** to applications, so they are only used when needed
  - Services are also designed to be easily utilized by software developers, who have to create applications in a consistent way
- SOA is independent of services, vendors, and technologies
  - It is only a concept and not limited to any programming language or platform

SOA: examples

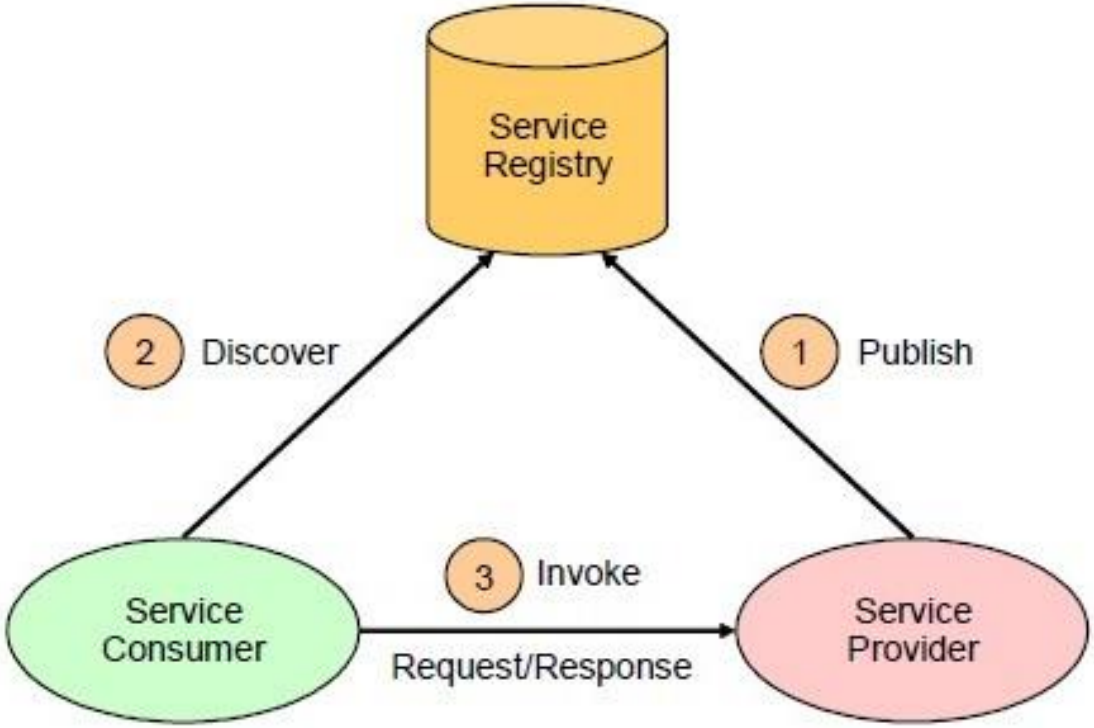
1. **Google Docs**
   - Go to Extensions → click Add-ons → variety of services (Spell checker)
     - **Google Docs** and **Spell checker** can communicate each other via policy agreement
2. **Banking mobile app**



SOA: examples



Component of SOA



Component of SOA

Service Provider

- The service provider is the maintainer of the service and the organization that makes available one or more services for others to use
- The service provider publishes the interface and the access details with the Service Registry
- The provider must publish their services using
  1. Web services description language (WSDL): XML language
  2. Universal Description, Discovery, and Integration (UDDI)

Service Registry

- The registry **publish** the service together with a service contract that specifies the nature of the service, how to use it, the requirements for the service, and the fees charged

Component of SOA

Service Consumer

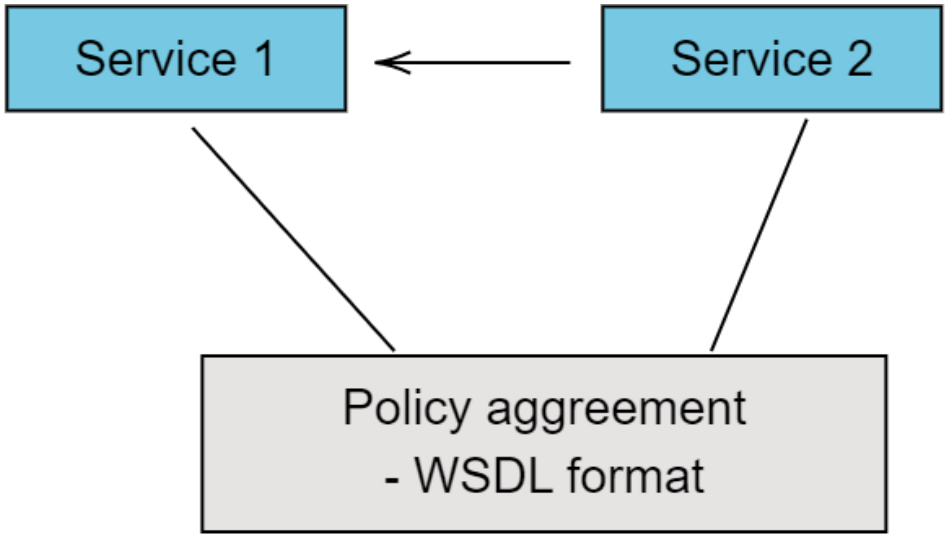
- The Consumer ensures that it's possible to locate the service (**discover**) that is registered in the service registry and
- It **binds** to the service provider is obtained and **invoke** the service that is defined.

SOA is typically implemented with web services such as simple object access protocol (SOAP) and web services description language (WSDL)

Service-Oriented Architecture (SOA) Principles

SOA provides a reference model for architecting several software systems. The following guiding principles characterize SOA platforms:

1. Standardized service contract
   - Services adhere (follow) to a given communication agreement, which is specified through one or more service description documents



Application design using service 1 along with part of service 2 using the policy agreement

Service-Oriented Architecture (SOA) Principles

2. Loose coupling
   - Services are designed as self-contained components, maintain relationships that minimize dependencies on other services, and only require being aware of each other
   - Service contracts will enforce the required interaction among services.
   - This simplifies the flexible aggregation of services and enables a more agile (improve) design strategy that supports the evolution of the enterprise business.
3. Abstraction
   - A service is completely defined by service contracts and description documents
   - They hide their logic, which is encapsulated within their implementation
     - The service shouldn't show how it performs its functionality
4. Reusability
   - Designed as components, services can be reused more effectively, thus reducing development time and the associated costs

Service-Oriented Architecture (SOA) Principles

5. Autonomy
   - Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation
6. Lack of state
   - Services should stay stateless
     - services should not keep data from one state to the other
   - This would be required to be done from each client application
7. Discoverability
   - Services can be discovered (usually in a service registry)
   - Service discovery provides an effective means for utilizing third-party resources
8. Composability
   - Using services as building blocks, sophisticated and complex operations can be implemented
     - It breaks large problems into tiny problems

Characteristics of Service-Oriented Architecture

- supports loose coupling
- supports interoperability (operate in conjunction with each other)
- supports vendor diversity
- increases the quality of service
- location-transparent

SOA Advantages

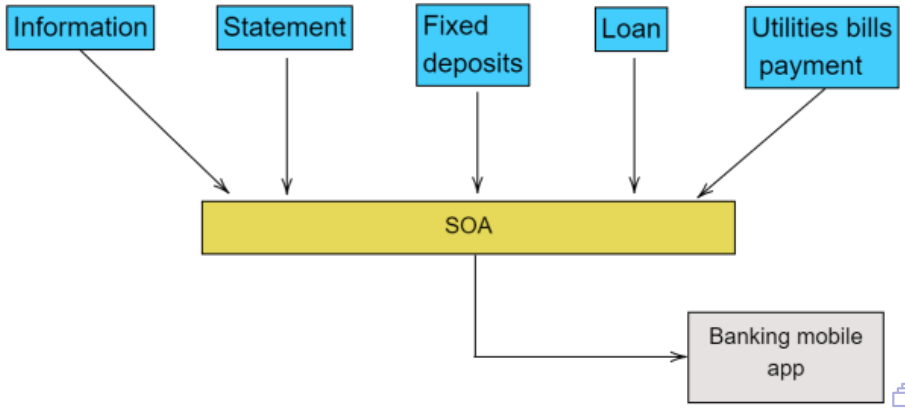
- **Service Reusability:** Applications are built from existing services. The services can be re-used to create many other applications
- **Easy Maintenance:** As services are independent of each other, they can be updated and transformed easily without harming other services
- **Availability:** These facilities are effortlessly available to anyone on demand
- **Platform independent:** SOA allows making a complex application by combining services picked from different sources, independent of the platform
- **Reliability:** These applications are extra secure because it is simple to test short code rather than large codes
- **Scalability:** Services can work on various servers within an environment, this improves scalability

SOA Disadvantages

- **Standards dependency:** depends on the implementation of standards
- **High overhead:** A validation of input parameters of services is done whenever services interact this decreases performance as it increases load and response time.
- **High investment:** A huge initial investment is required for SOA.
- **Complex service management:** When services interact they exchange messages to tasks. The number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.

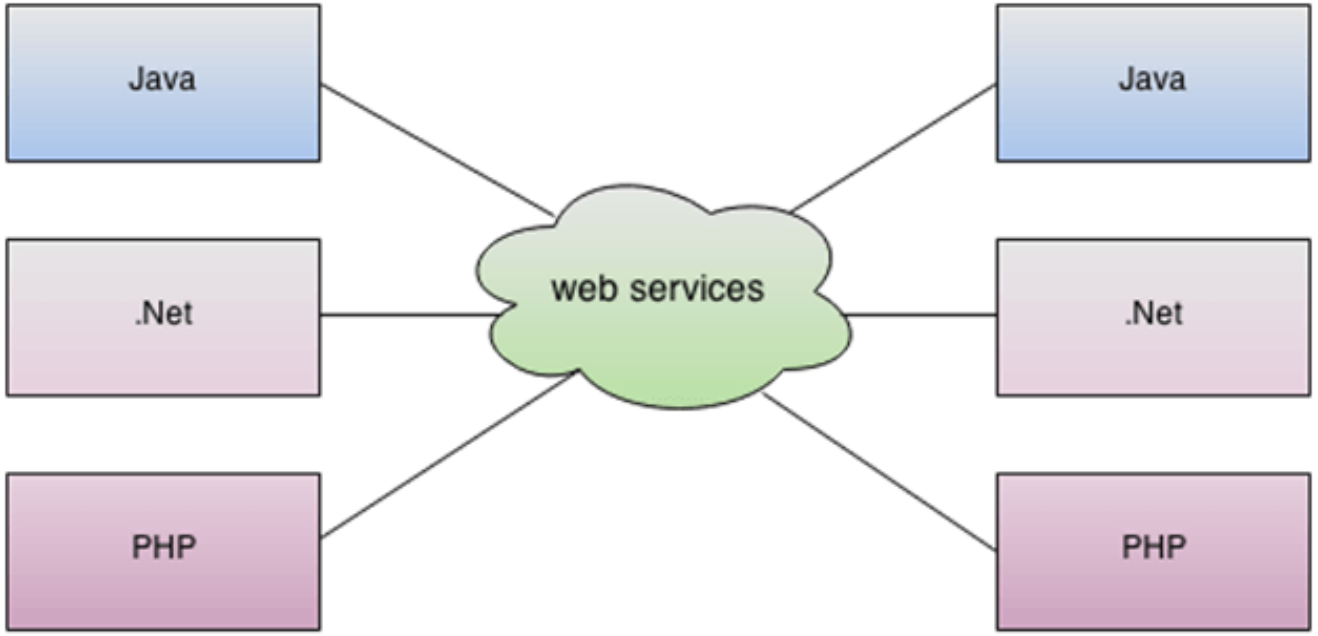
Applications of SOA

- To manage a data (e.g: university students, banking data)
- To consolidate different services (e.g: banking, healthcare)
- To manage workflow (e.g: top-level, middle-level, low-level management)
- To improve customer services (e.g: different services at single platform)
- For more effective partnership (e.g: payment gateway from mobile banking)
- To trim cost
- To increase agility of services



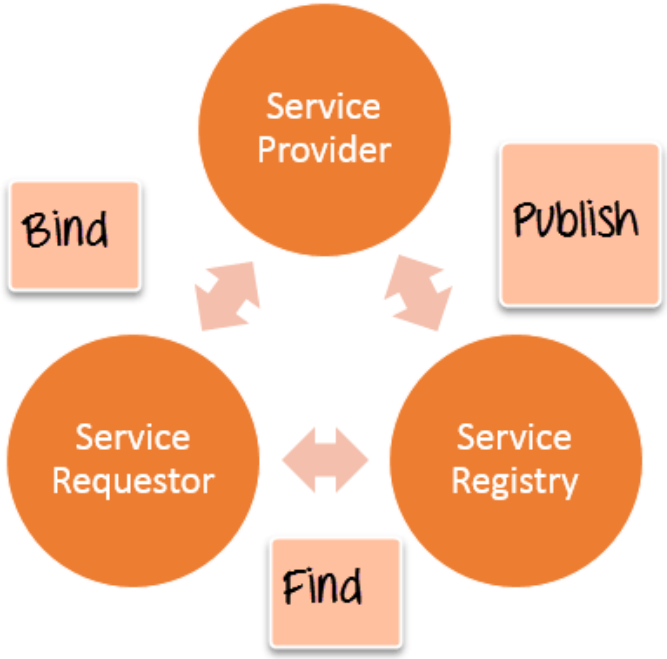
Web Services

- Web services are applications that allow for communication between devices over the internet
- Usually independent of the technology or language the devices are built on as they use standardised eXtensible Markup Language (XML) for information exchange
- A client or user is able to invoke a web service by sending an XML message and then in turn gets back and XML response message.



Web Services

- Collection of open protocols and standards used for exchanging data between application or system
- To communicate one service with another
- Web service can be searched for over the internet and can also be invoked accordingly



Web service Characteristics

1. **XML-based:**
   - A web service's information representation and record transport layers employ XML
   - Web offering-based applications are highly interactive
2. **Loosely Coupled:**
   - the client and the web service are not bound to each other
     - if the web service changes over time, it should not change the way the client calls the web service
3. **Synchronous or Asynchronous functionality:**
   - In synchronous operations, the client will actually wait for the web service to complete an operation
     - If data is read from one database and subsequently written to another, then the operations have to be done in a sequential manner
   - Asynchronous operations allow a client to invoke a service and then execute other functions in parallel

Web service Characteristics

4. **Ability to support Remote Procedure Calls (RPCs):**
   - Web services enable clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol
   - Remote procedures expose input and output parameters that a web service must support
5. **Heterogeneous languages and environments:**
   - Enables web services that are built on various programming languages to communicate with each other

Web Service Components

Three major web service components.

1. Simple Object Access Protocol (SOAP)
   - SOAP is a messaging protocol for exchanging information between two computers based on XML over the internet
     - that's why independent of platform and language
2. Web Services Description Language (WSDL)
   - XML based standard file to tell the client application what does the web service do
3. Universal Description, Discovery and Integration (UDDI)
   - XML based standard for describing, publishing and find web service
   - Platform independent open framework
   - use WSDL to describe interface to web services

Web Service Components

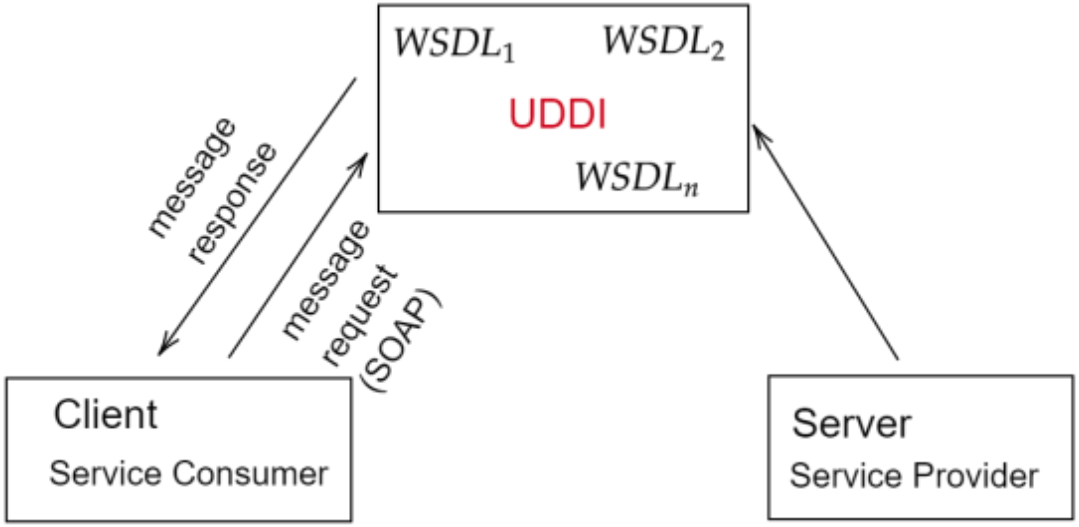


Figure: Web Service Components

For example

- UDDI : Youtube
- WSDL : Videos (Cloud Computing)
  You can get multiple cloud computing video (WSDL) in Youtube (UDDI)

How Web Services Work?

- A web service takes the help of:
  - XML: to tag the data
  - SOAP: to transfer a message
  - WSDL: to describe the availability of service
  - UDDI: publish and find services

How Web Services Work?

1. Client would request (invoke) a series of web services calls via requesting to server through Remote Procedure Call (RPC)
2. The data which is transferred between client and server is inform of XML
3. Web services use SOAP for sending the XML data between client applications and web services
4. SOAP does not pose any restriction on transport
   - Different standards like HTTP, TCP/IP, SMTP, JASON can be used for sending message over internet
5. The data which is sent from the web services to the application is called SOAP message (an XML document)

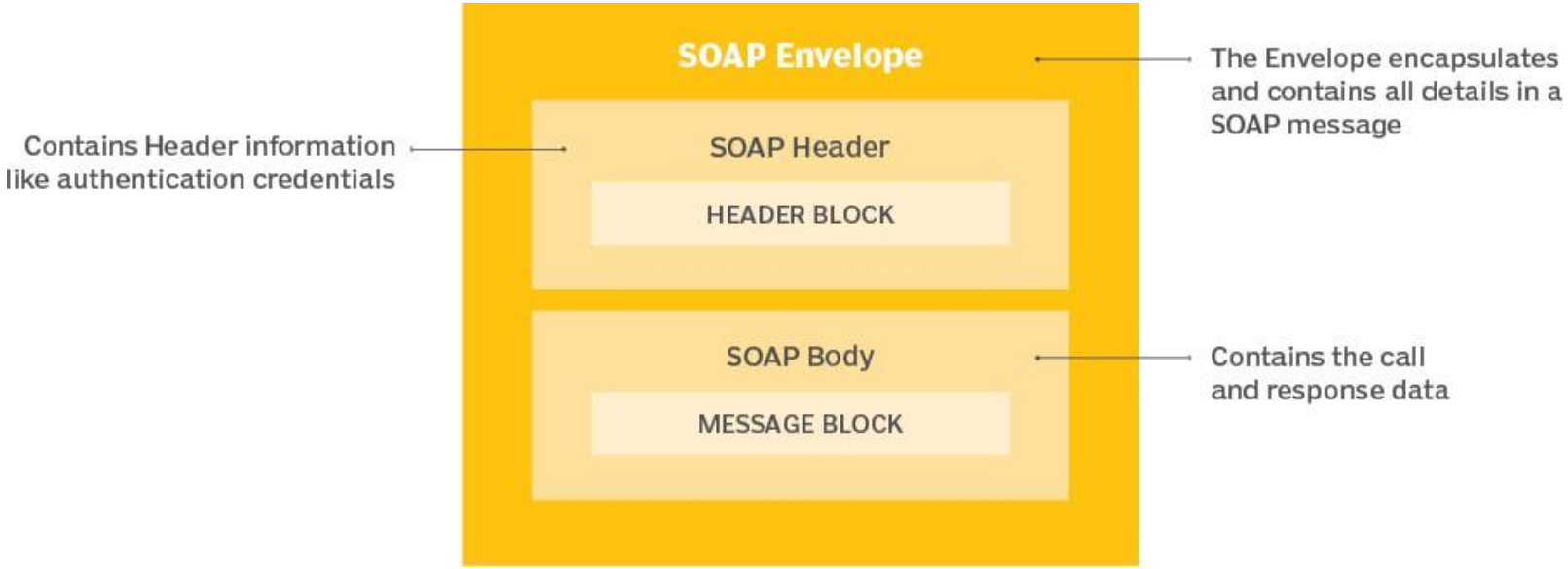
Type of Web Service

1. SOAP web services
2. RESTful web services

SOAP web services

- Simple Object Access Protocol (SOAP): is a protocol for exchanging data between Web services over HTTP (Hypertext transfer protocol) or SMTP (simple mail transfer protocol)
- SOAP messages use XML Information Set for their formatting.
- Three characteristics of SOAP:
  - neutrality, independence, and extensibility.
- Each SOAP document needs to have a root element known as the <Envelope> element
- The "envelope" is in turn divided into 2 parts.
  - header
  - body

SOAP Message Building Blocks



SOAP Message Building Blocks

SOAP Envelope

- An Envelope element that identifies the XML document as a SOAP message
- Used to encapsulate all the details in the SOAP message
- Root element in the SOAP message

SOAP Header

- A Header element that contains header information –
  - authentication credentials which can be used by the calling application
- Also contain multiple header blocks
  - Header block1: username & password to run application

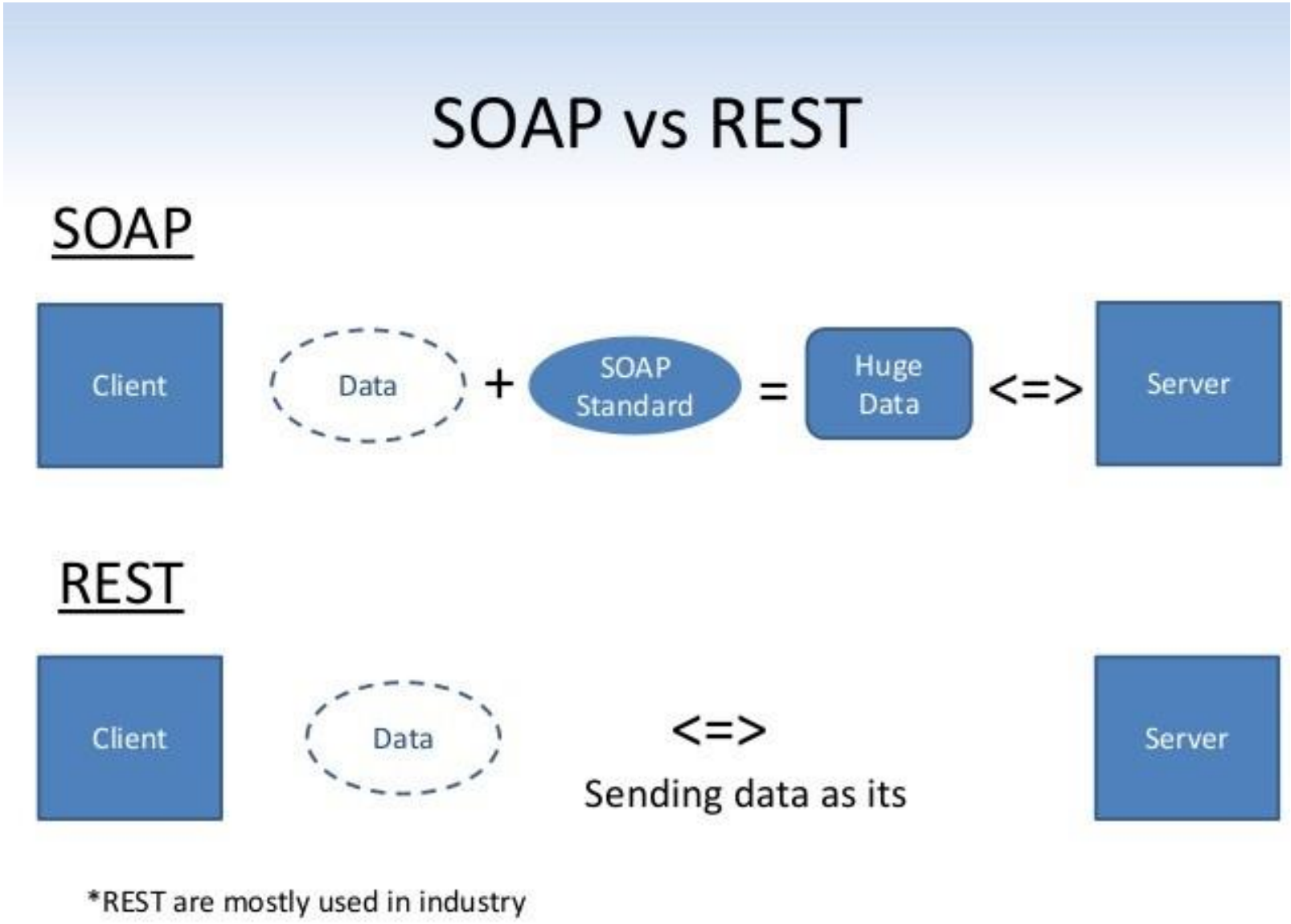
SOAP Body

- A body element includes the details of the actual message that need to be sent from the web service to the calling application
- Data includes call and response information

RESTful web services

- Representational State Transfer (REST) is an architectural style for designing loosely coupled application over HTTP
  - guideline to build a light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications
- In a client-server communication, REST suggests to create an object of the data requested by the client and send the values of the object in response to the user
  - For example, if the user is requesting for a movie in Kathmandu at a certain place and time, then you can create an object on the server-side.
- Web services which follow REST architecture style are RESTful Web services
- In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web

RESTful web services



RESTful web services

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI:** Resources are identified by URIs, which provide a global addressing space for resource and service discovery.
- **Uniform interface:** Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.
  - PUT creates a new resource
  - DELETE to remove existing resource
  - GET retrieves the current state of a resource in some representation
  - POST transfers a new state onto a resource
- **Self-descriptive messages:** Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others

RESTful web services

- **Stateful interactions through hyperlinks:** Every interaction with a resource is stateless; i. e., request messages are self-contained
  - The concept of stateless means that it's up to the client to ensure that all the required information is provided to the server
    - This is required so that server can process the response appropriately
- **Cache:** Each server client request is independent in nature, sometimes the client might ask the server for the same request again
  - This request will go to the server, and the server will give a response
  - This increases the traffic across the network
  - The cache is a concept implemented on the client to store requests which have already been sent to the server
    - if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information
    - this saves the amount of to and fro network traffic from the client to the server

RESTful web services

Cache

Uniform interface

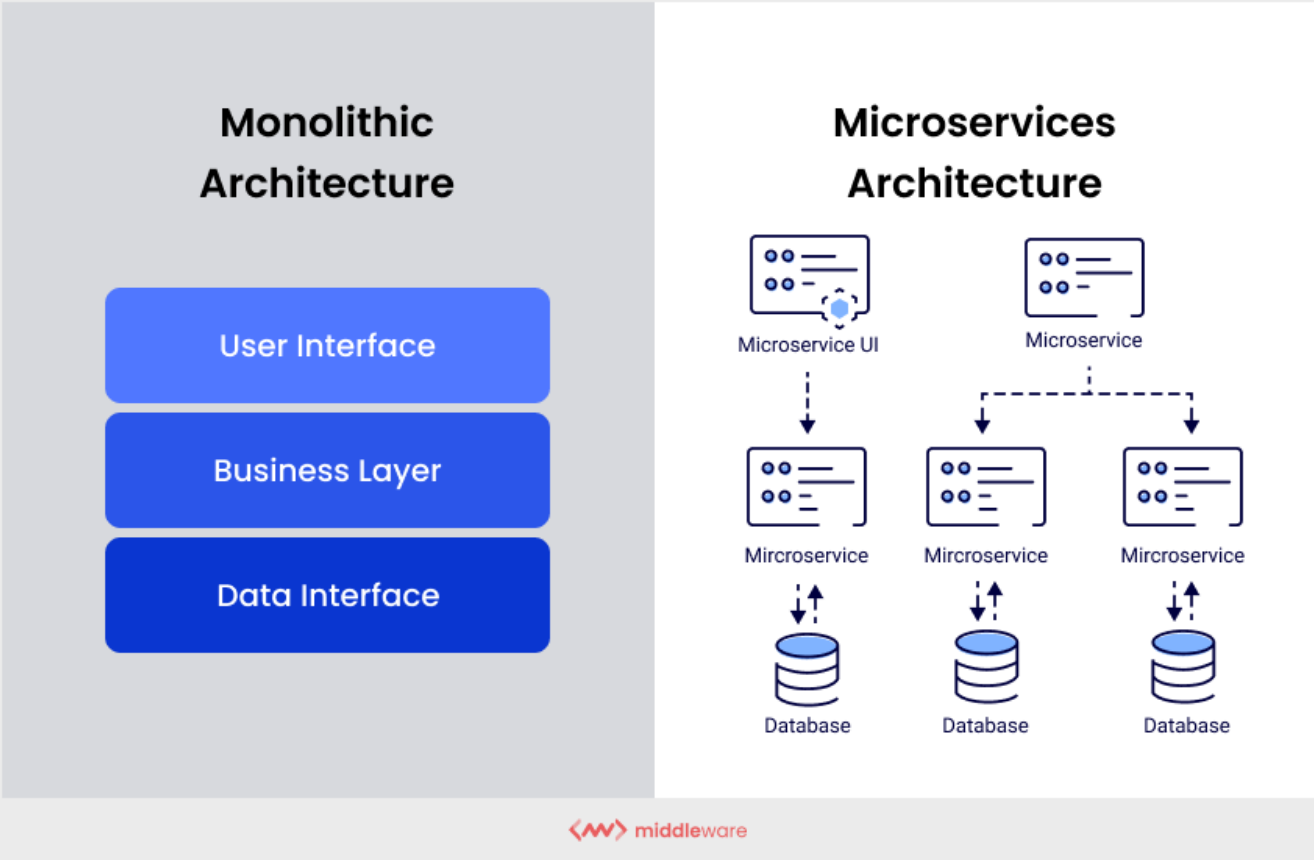
Resource	POST	GET
/Employee /Employee/1	**Create** **Create a new employee**	Read Get All Employee details

PUT	DELETE
Update Update relevant Employee details	Delete Delete relevant Employee details

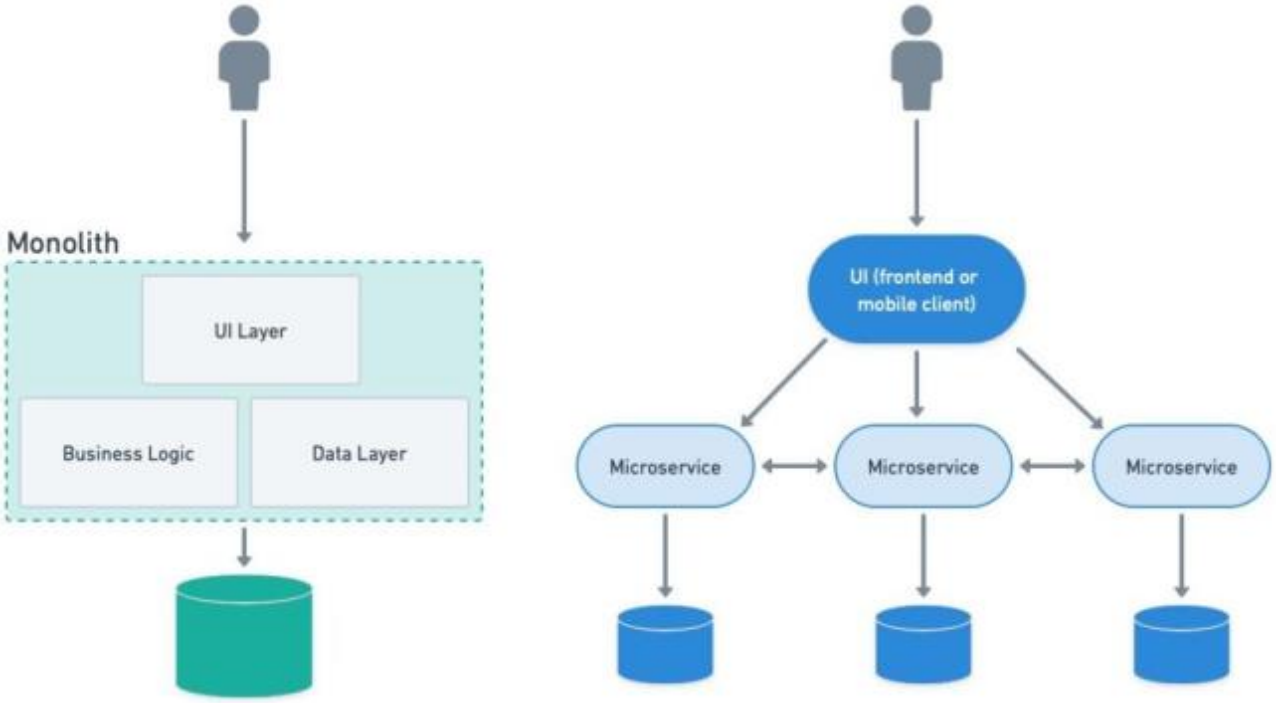
Microservices

- Microservice architecture is a software design approach that decomposes an application into small independent services that communicate over well-defined APIs
- Microservices allow companies to keep teams small and agile
- The idea is to decompose the application into small services that can be autonomously developed and deployed by tightly-knitted teams
- Each service can be developed and maintained by autonomous teams, it is the most scalable method for software development

Microservice



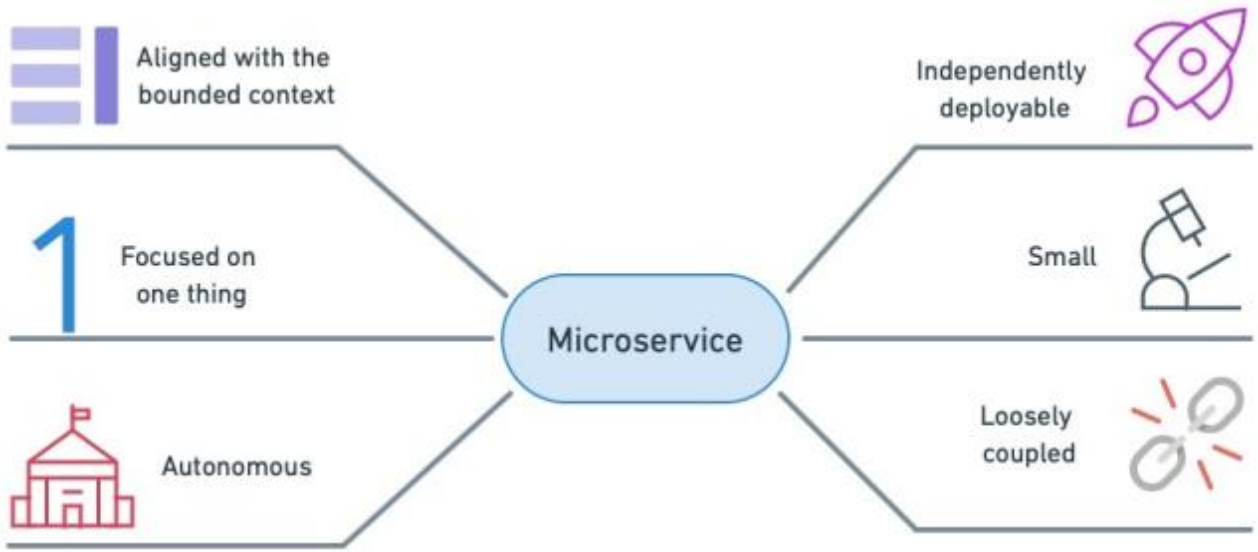
Microservice vs. monolith architectures



Benefits of microservices

1. **Scalability:** Services can be developed and released independently without arranging large-scale coordination efforts within the organization
2. **Fault isolation:** A benefit of having a distributed system is the ability to avoid single failure points
3. **Smaller teams:** With microservices, the development team can stay small and cohesive. The smaller the group, the less communication overhead and the better the collaboration
4. **Freedom to choose the tech stack:** each microservice can use the tech stack that is most appropriate for solving the task at hand. Thus, the team can pick the best tool for the job and their skills
5. **More frequent releases:** The development and testing cycle is shorter as small teams iterate quicker. And, because they can also deploy their updates at any time, microservices can be deployed much more frequently than a monolith

Microservice properties



Microservice architecture design challenges

- **Small:** applies both to the team size and the codebase
  - A microservice must be small enough to be entirely understood by one person
  - Your microservice is too big if it would take you more than a sprint to rewrite it from scratch
- **Focused on one thing:** a service must focus on one aspect of the problem or perform only one task
- **Autonomous:** autonomy allows a team to choose the most appropriate language stack and data model
  - This usually means that each microservice has its own database or persistence layer that is not shared with other services
- **Aligned with the bounded context:** in software, we create models to represent the problem we want to solve
  - A bounded context represents the limits of a given model
  - Contexts are natural boundaries for services, so finding them is part of designing a good microservice architecture

Microservice architecture design challenges

- **Loosely-coupled:** while microservices can depend on other microservices, we must be careful about how they communicate
  - Each time a bounded context is crossed, some level of abstraction and translation is needed to prevent behaviour changes in one service from affecting the others
- **Independently deployable:** being autonomous and loosely coupled, a team can deploy their microservice with little external coordination or integration testing
  - Microservices should communicate over well-defined APIs and use translation layers to prevent behaviour changes in one service from affecting the others

Microservice: example (Food-to-Go)

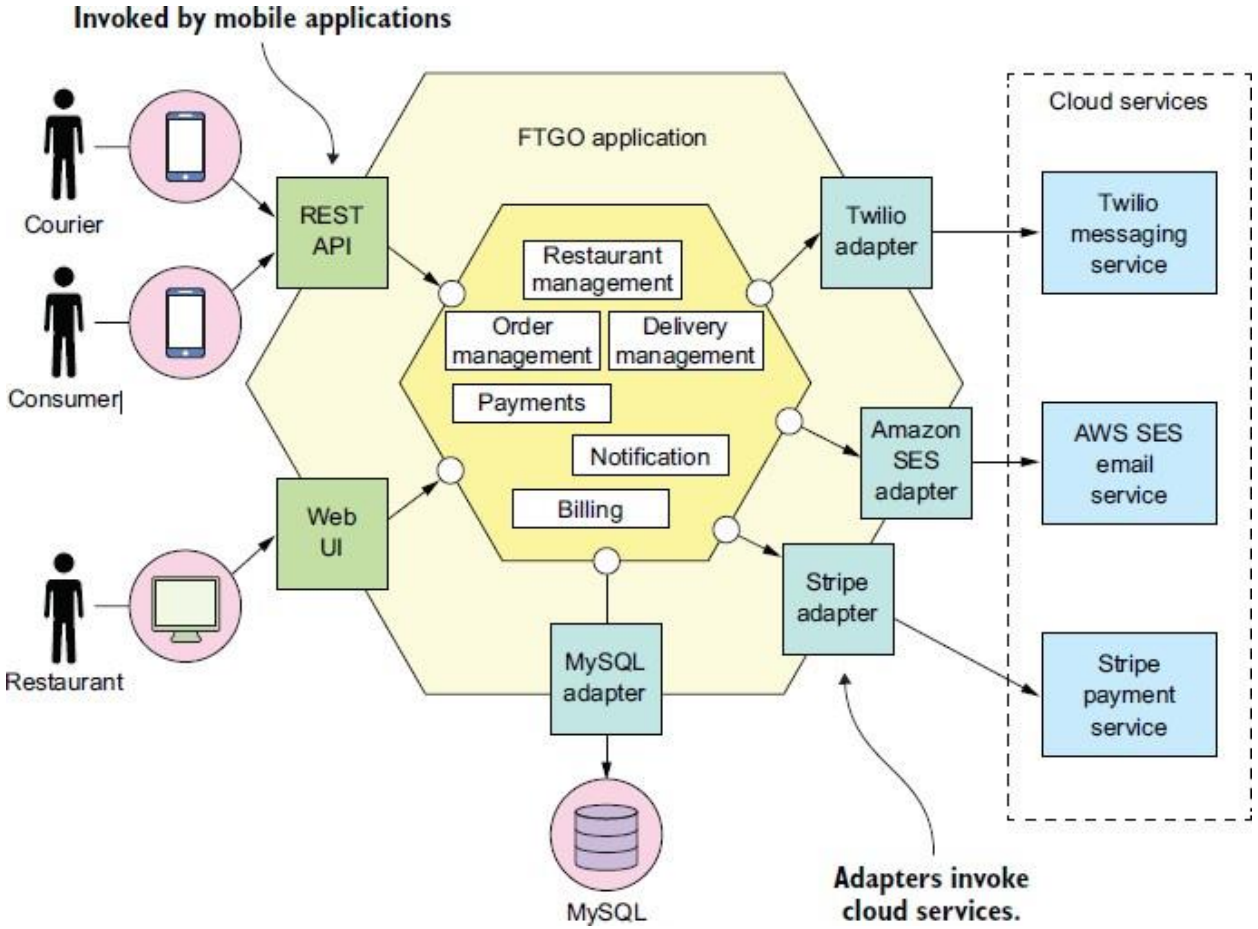


Figure: Food-to-Go (FTGO) monolithic architecture

Microservice: example (Food-to-Go)

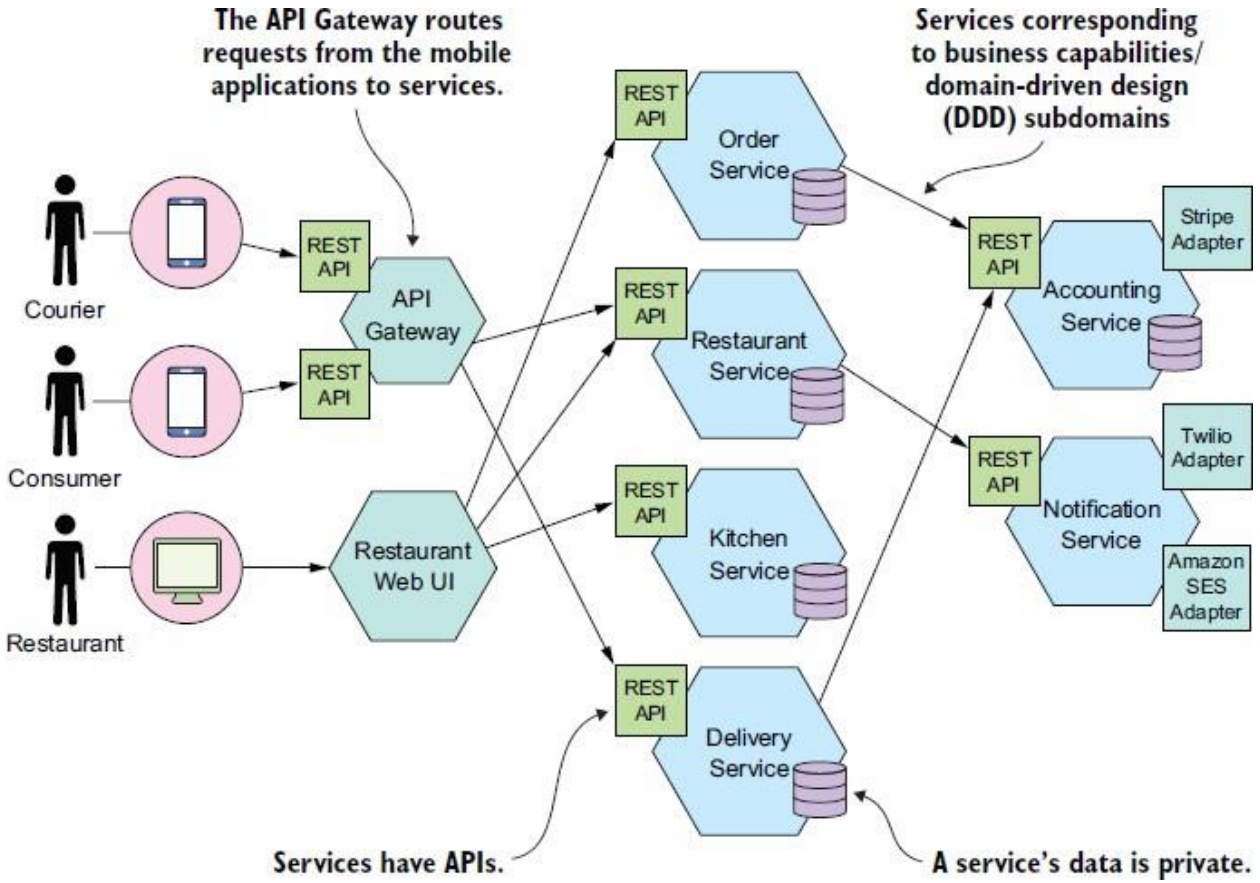


Figure: Food-to-Go (FTGO) microservices architecture

Microservice: example (Food-to-Go)

Each backend service has a REST API and its own private data store. The backend services include the following:

- **Order Service:** Manages orders
- **Delivery Service:** Manages delivery of orders from restaurants to consumers
- **Restaurant Service:** Maintains information about restaurants
- **Kitchen Service:** Manages the preparation of orders
- **Accounting Service:** Handles billing and payments

Microservice: example (Food-to-Go)

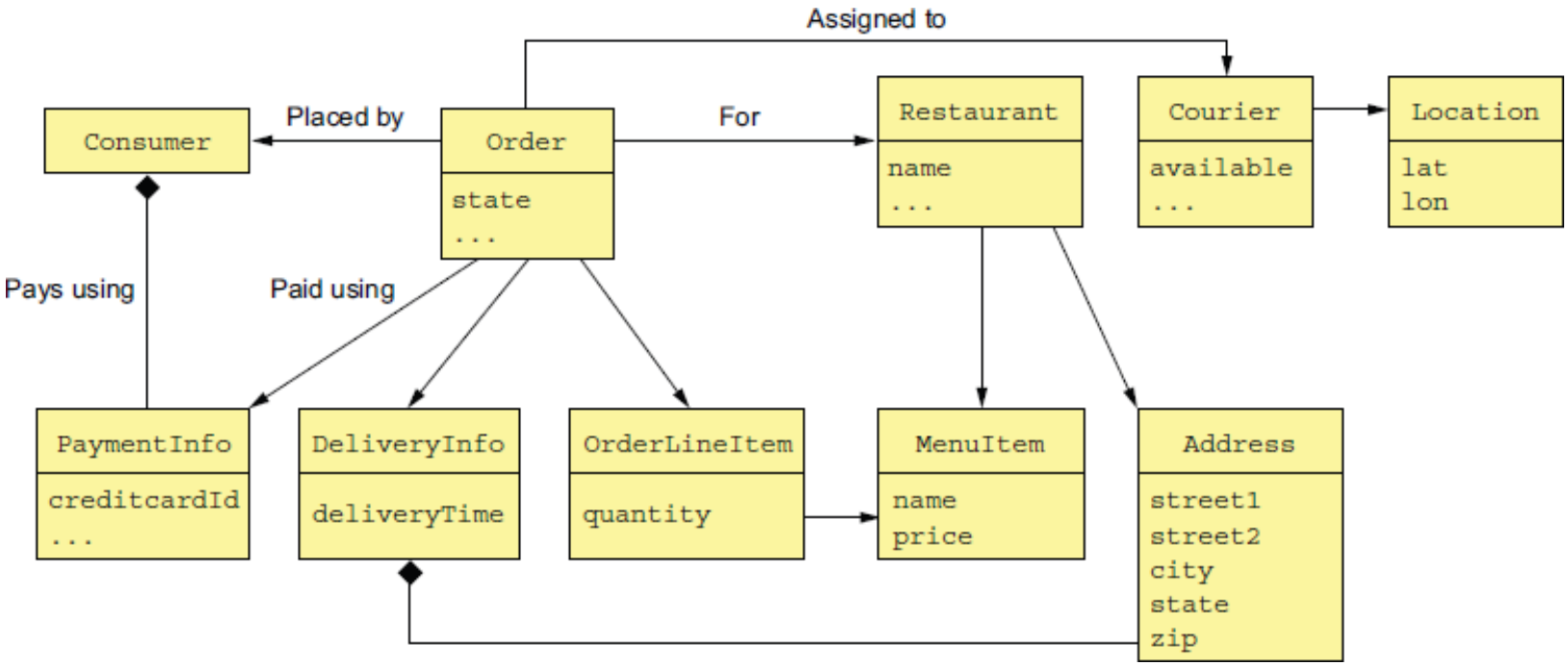


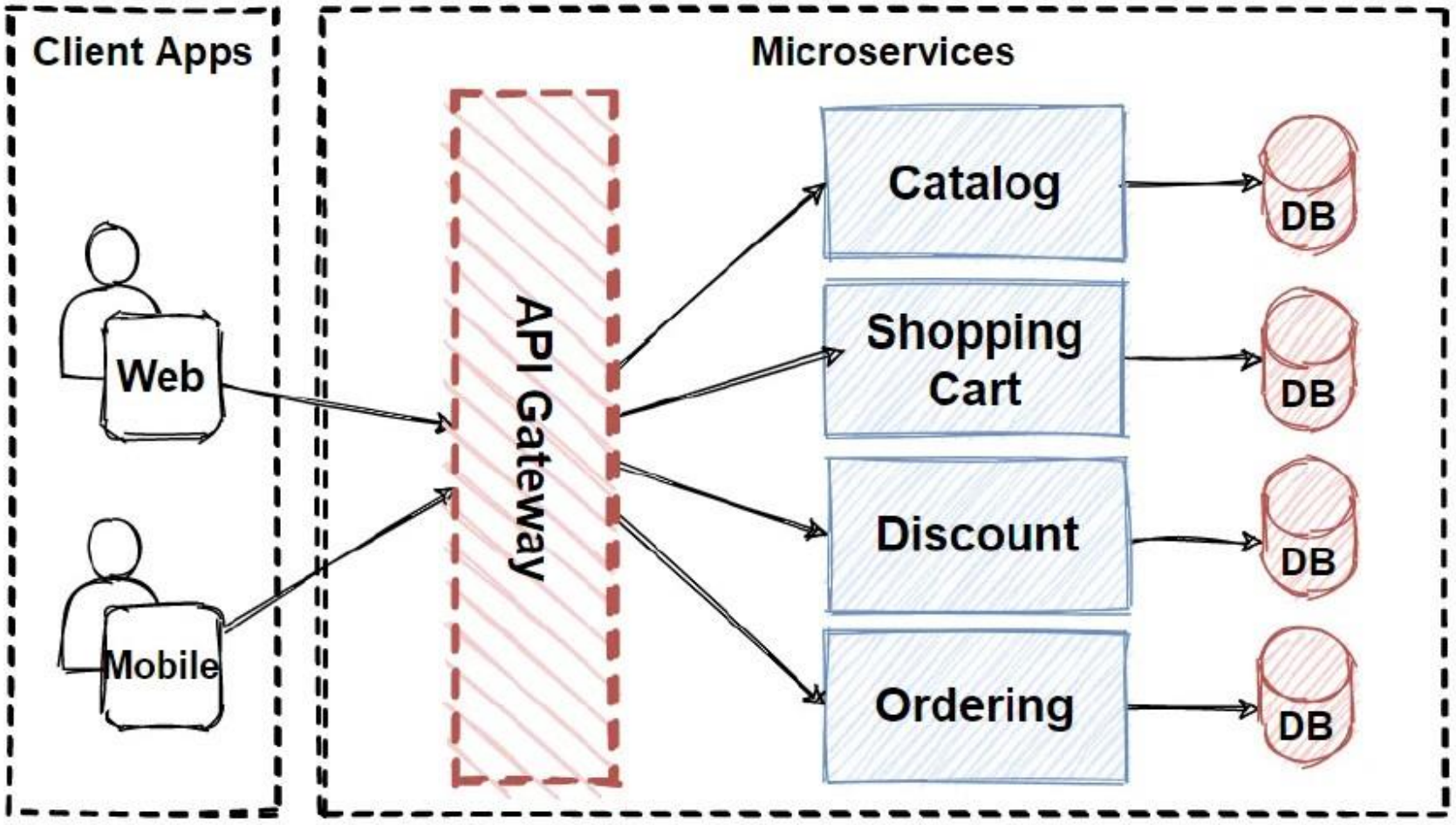
Figure: FTGO class diagram with key classes

Microservice: example (Food-to-Go)

The responsibilities of each class are as follows:

- **Consumer:** A consumer who places orders
- **Order:** An order placed by a consumer. It describes the order and tracks its status
- **OrderLineItem:** A line item of an Order
- **DeliveryInfo:** The time and place to deliver an order
- **Restaurant:** A restaurant that prepares orders for delivery to consumers
- **MenuItem:** An item on the restaurant's menu
- **Courier:** A courier who delivers orders to consumers. It tracks the availability of the courier and their current location
- **Address:** The address of a Consumer or a Restaurant
- **Location:** The latitude and longitude of a Courier

Microservice: example (e-commerce website)

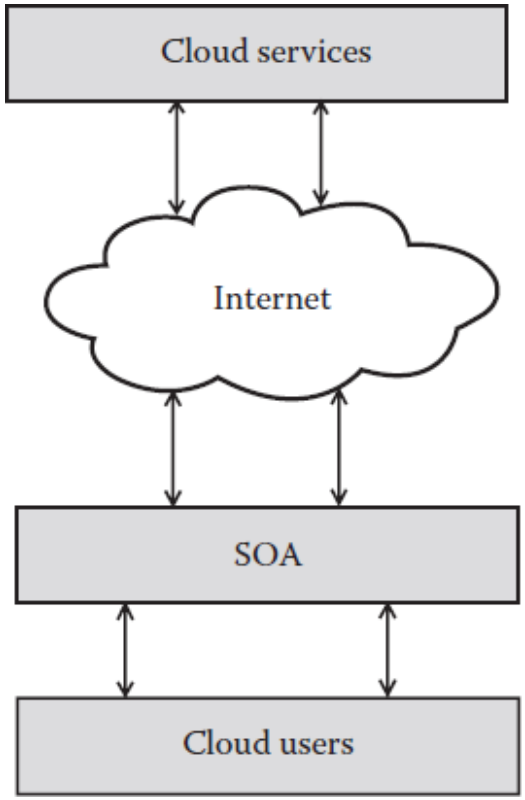


SOA and Cloud

- SOA is a flexible set of design principles and standards used for systems development and integration
  - SOA-based system provides a loosely coupled set of services that can be used by the service consumers for meeting their service requirements within various business domains
- Cloud computing is a service delivery model in which shared services and resources are consumed by the users across the Internet just like a public utility on an on-demand basis
- SOA is used by enterprise applications, and cloud computing is used for availing the various Internet-based services
  - When used with cloud computing, SOA helps to deliver IT resources as a service over the Internet and to mix and match the resources to meet the business requirements
- SOA using cloud computing architecture provides agility: it could easily be changed to incorporate the business needs since it uses services that are configured through a configuration or process layer

SOA and Cloud

- Cloud and SOA are focused on delivering services to the business with increased agility, speed, and cost-effectiveness



Cloud computing open architecture (CCOA)

CCOA: architecture for the cloud environment that incorporates the SOA. The goals of the CCOA are as follows

- To develop an architecture that is reusable and scalable
  - in future, the architecture should incorporate any further changes without the need for replacing the entire architecture
- To develop a uniform platform for the cloud application development
  - cloud users to switch between the CSPs without the need to make significant changes in the application
- To enable businesses to run efficiently
  - the CSPs to make more money by delivering quality services successfully