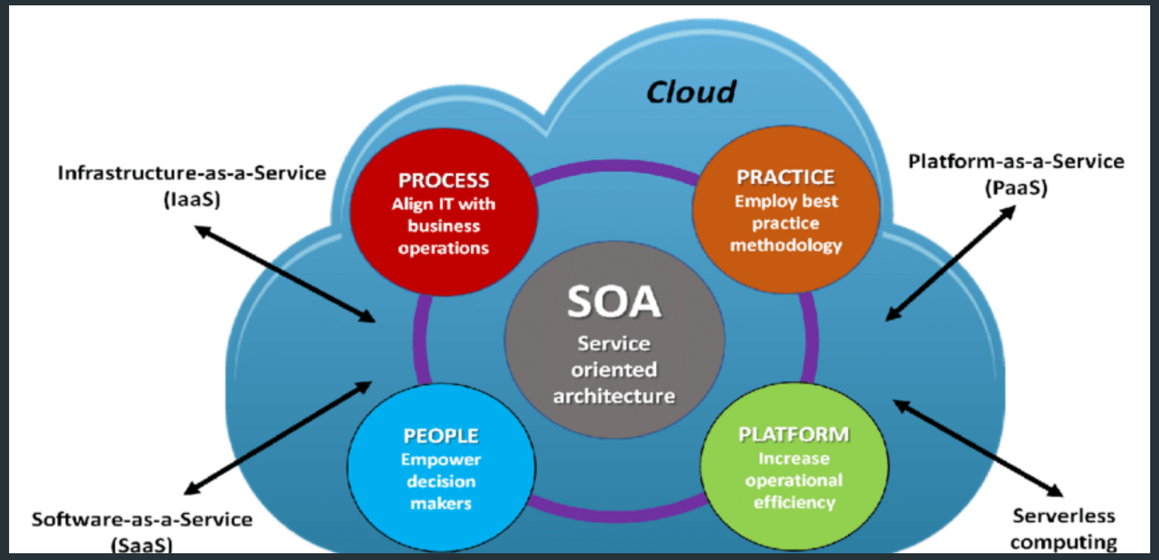
Lecture #7
SOA

WSMT2023

Definition of SOA

- What is Service-Oriented Architecture (SOA)?
- Why is SOA important?
- Examples of SOA implementations

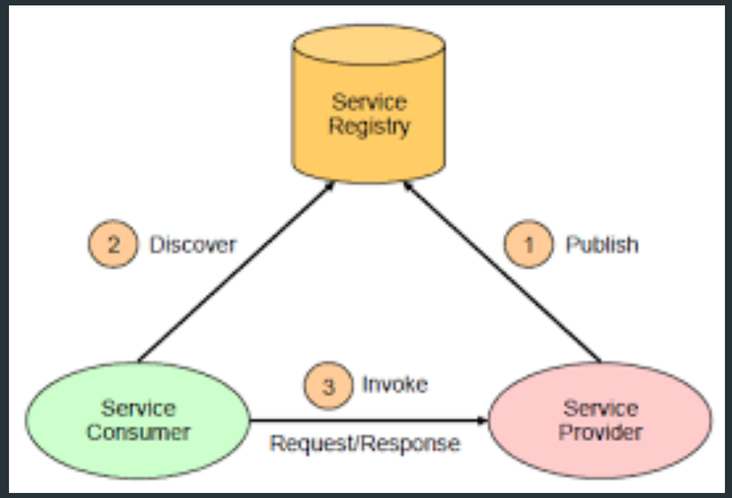


Services, Components, and Contracts

- What are services in SOA?

Services, Components, and Contracts

- What are services in SOA?
- What are components in SOA?



Services, Components, and Contracts

- What are services in SOA?
- What are components in SOA?
- What are contracts in SOA?

Example of Service and Contract

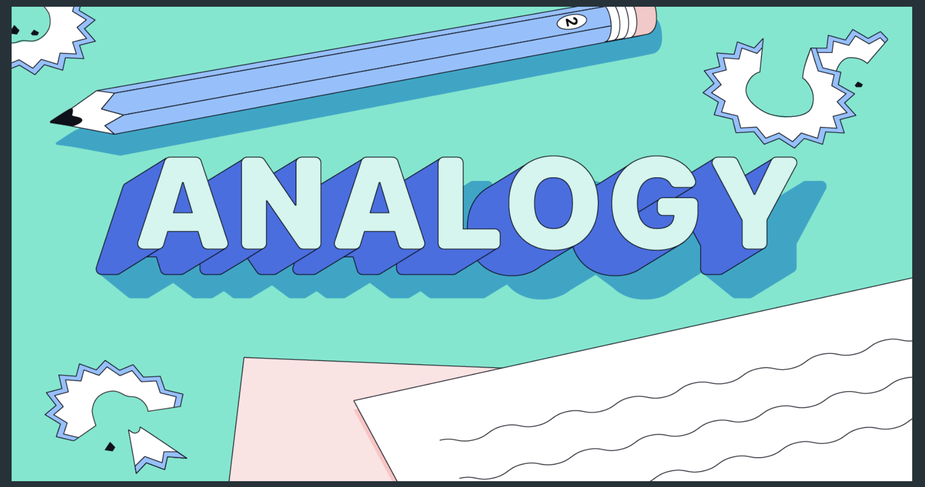
```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.example.com/currencyConverter/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
name="currencyConverter"
targetNamespace="http://www.example.com/currencyConverter/">
    <message name="convertToUSDRequest">
        <part name="amount" type="xsd:float"/>
    </message>
    <message name="convertToUSDResponse">
        <part name="result" type="xsd:float"/>
    </message>
    <message name="convertToEURRequest">
        <part name="amount" type="xsd:float"/>
    </message>
    <message name="convertToEURResponse">
        <part name="result" type="xsd:float"/>
    </message>
    <portType name="currencyConverterPortType">
        <operation name="convertToUSD">
            <input message="tns:convertToUSDRequest"/>
            <output message="tns:convertToUSDResponse"/>
```

```
<portType name="currencyConverterPortType">
    <operation name="convertToUSD">
        <input message="tns:convertToUSDRequest"/>
        <output message="tns:convertToUSDResponse"/>
    </operation>
    <operation name="convertToEUR">
        <input message="tns:convertToEURRequest"/>
        <output message="tns:convertToEURResponse"/>
    </operation>
</portType>
<binding name="currencyConverterBinding" type="tns:currencyConverterPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="convertToUSD">
        <soap:operation soapAction="http://www.example.com/currencyConverter/convertToUSD"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="convertToEUR">
        <soap:operation soapAction="http://www.example.com/currencyConverter/convertToEUR"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
```

```
    <binding name="currencyConverterBinding" type="tns:currencyConverterPortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="convertToUSD">
            <soap:operation soapAction="http://www.example.com/currencyConverter/convertToUSD"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="convertToEUR">
            <soap:operation soapAction="http://www.example.com/currencyConverter/convertToEUR"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="currencyConverterService">
        <port binding="tns:currencyConverterBinding" name="currencyConverterPort">
            <soap:address location="http://localhost:8080/currencyConverter"/>
        </port>
    </service>
</definitions>
```

A Service-Oriented Analogy

- Understanding SOA through a real-life analogy
- Importance of clear communication in SOA
- Implications of the analogy for web services development



The Analogy

- The story of a restaurant
- Understanding restaurant roles as services
- Mapping to SOA concepts

Clear Communication

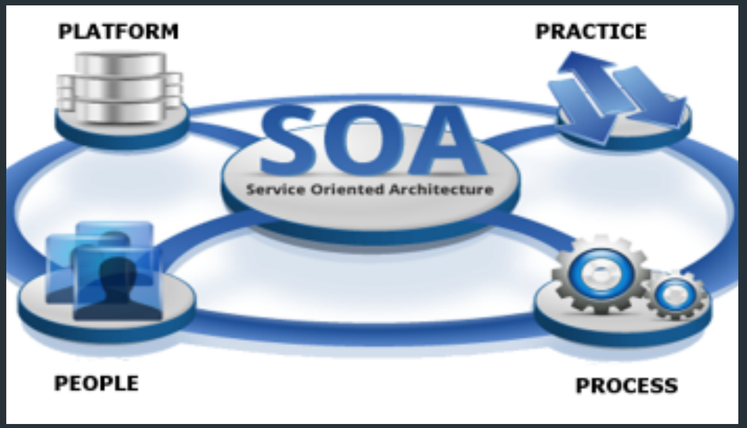
- Importance of clear communication in SOA
- Communication challenges in the restaurant analogy
- How to improve communication in SOA

Implications for Web Services

- Applying the analogy to web services development
- Service roles in web services
- Benefits of clear communication for web services

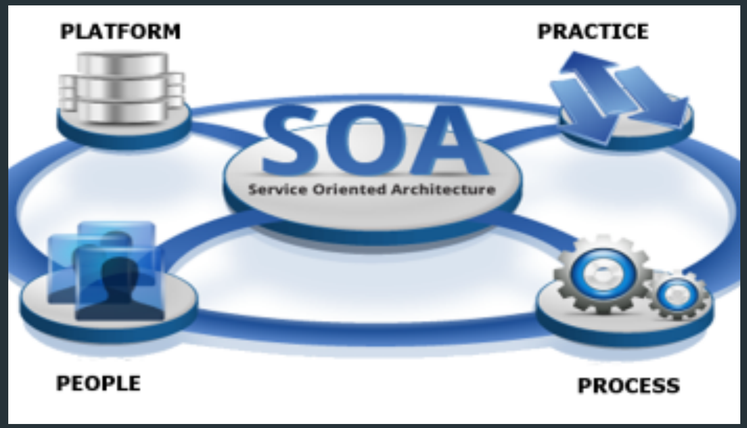
Understanding Encapsulation

- What is encapsulation in SOA
- Benefits of encapsulation
- Encapsulation in the context of services



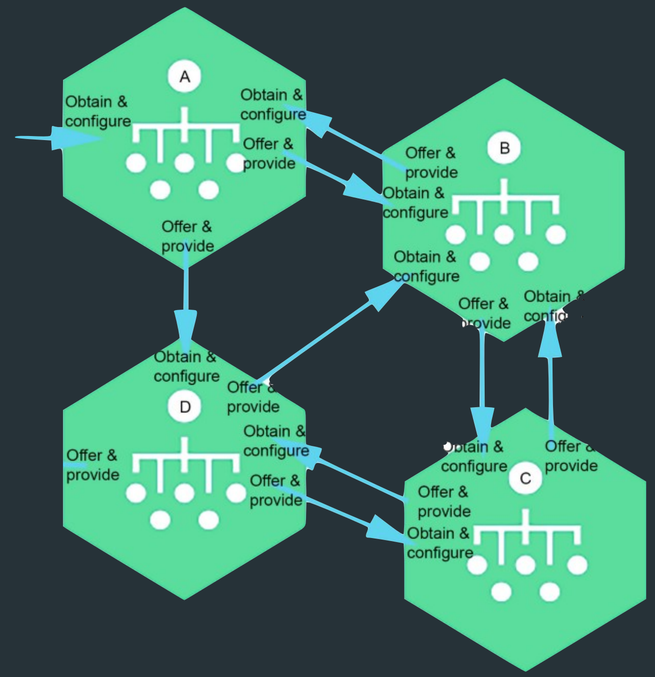
Advantages of Encapsulation

- Improved maintainability and scalability
- Increased reusability
- Better security and reliability



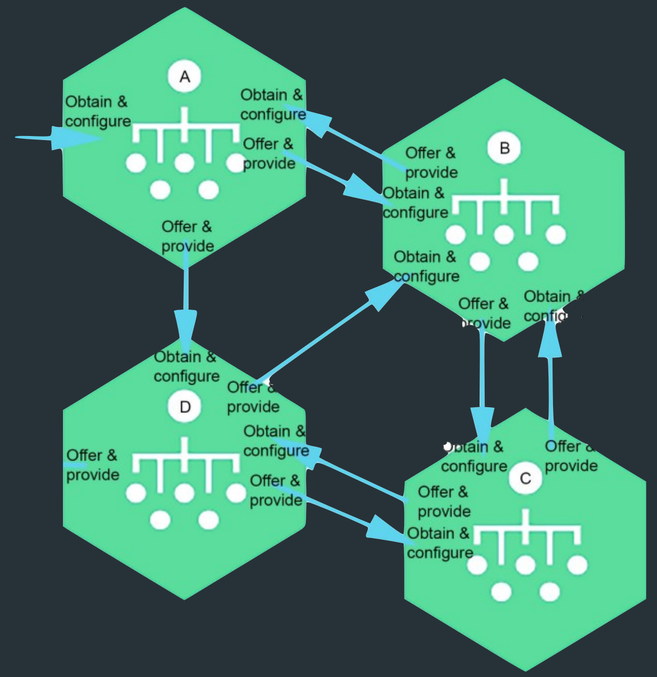
Understanding Service Relationships

- What are service relationships in SOA
- Different types of service relationships
- Characteristics of service relationships

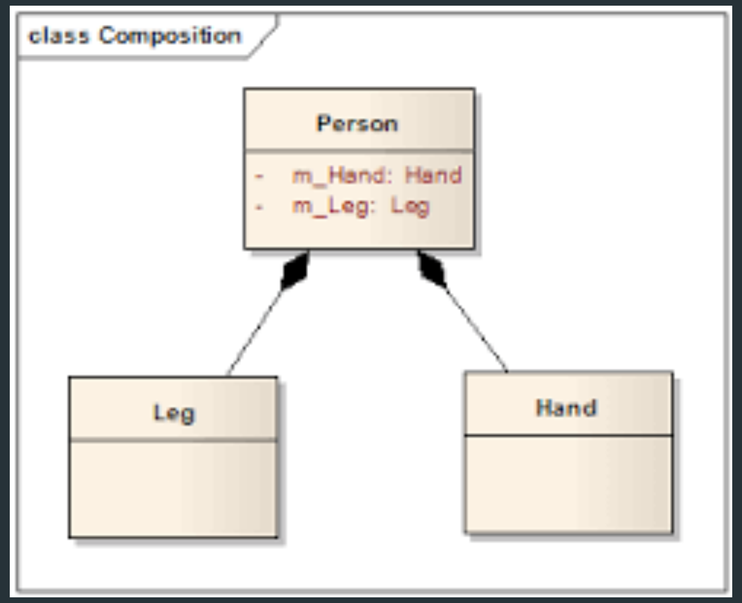


Types of Service Relationships

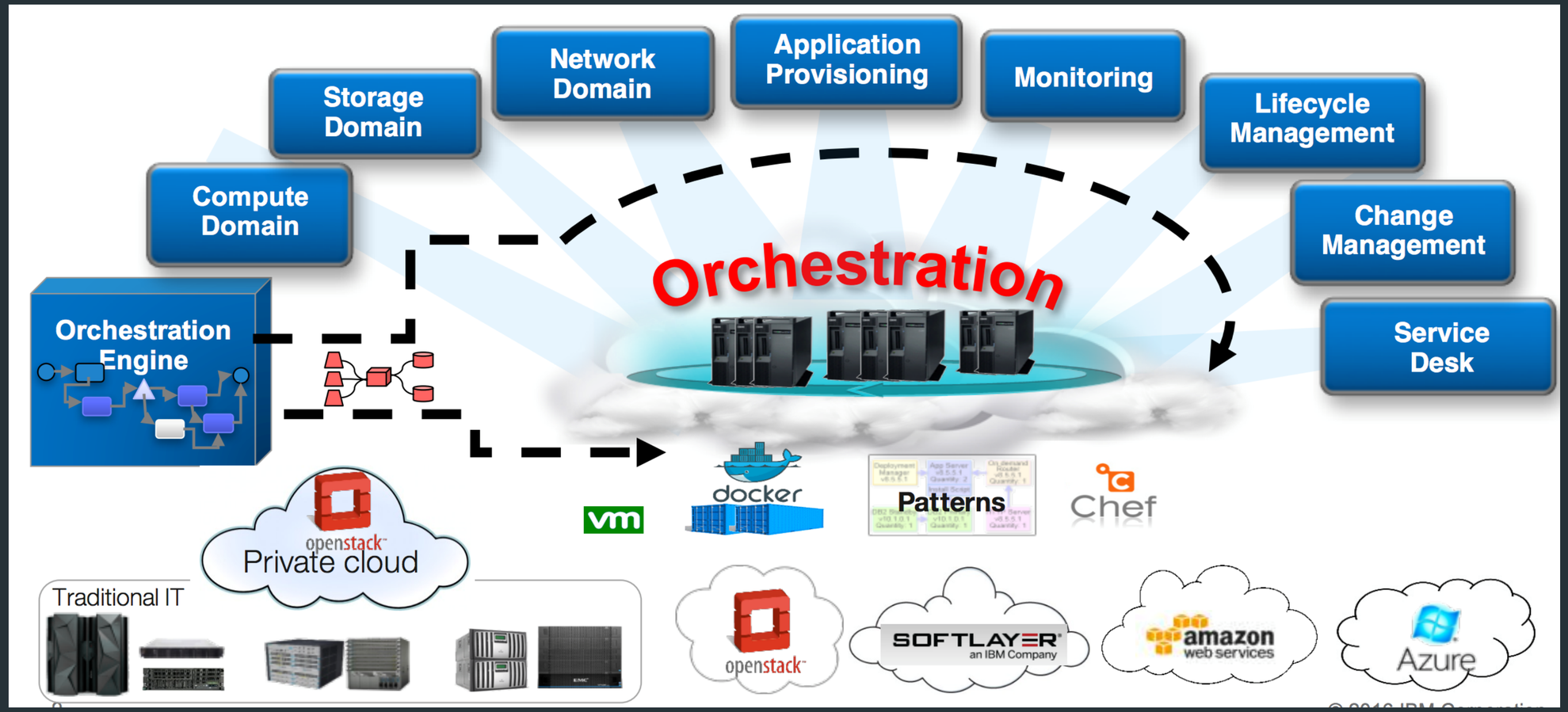
- Composition
- Orchestration
- Choreography



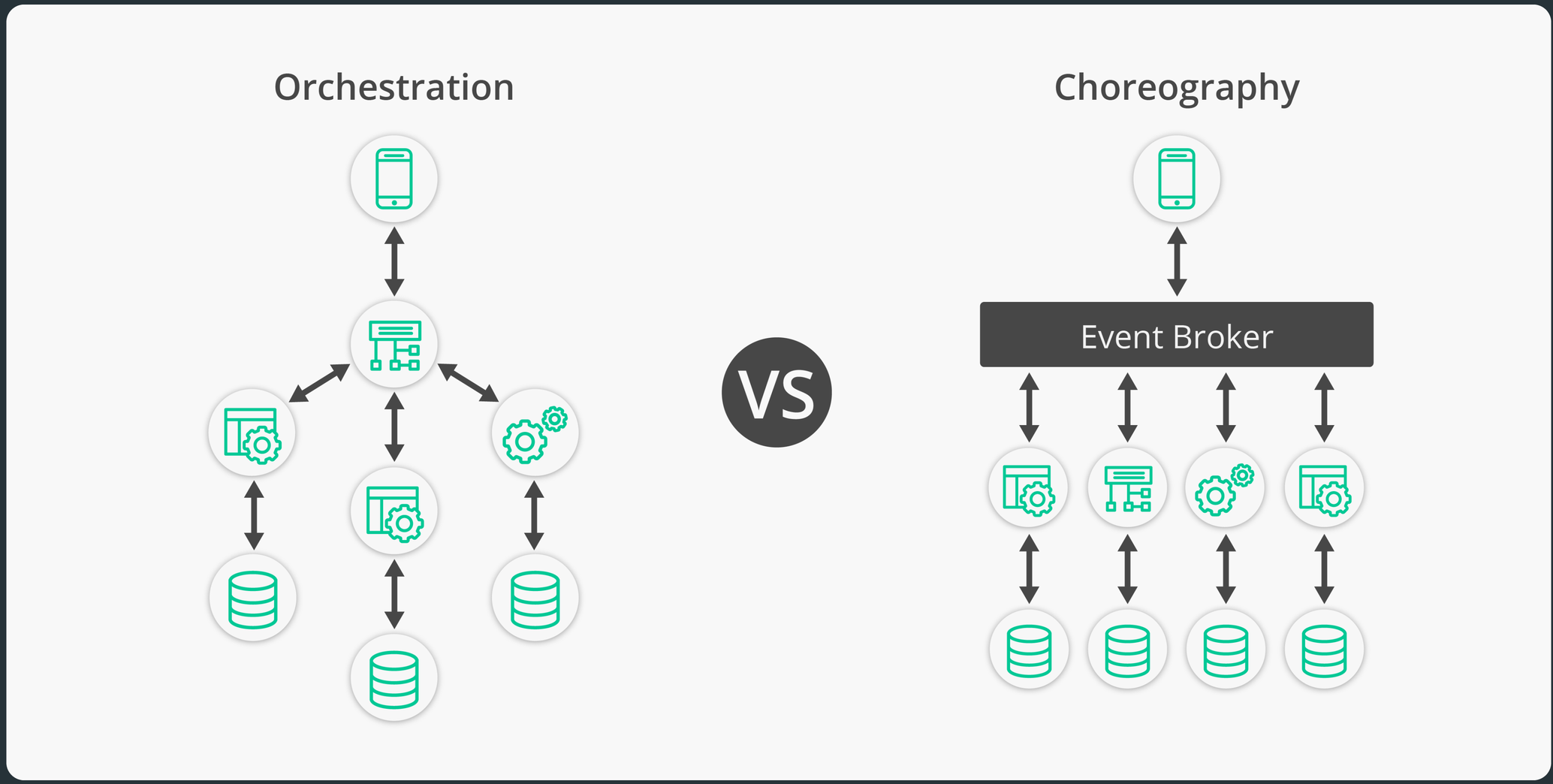
Composition



Orchestration

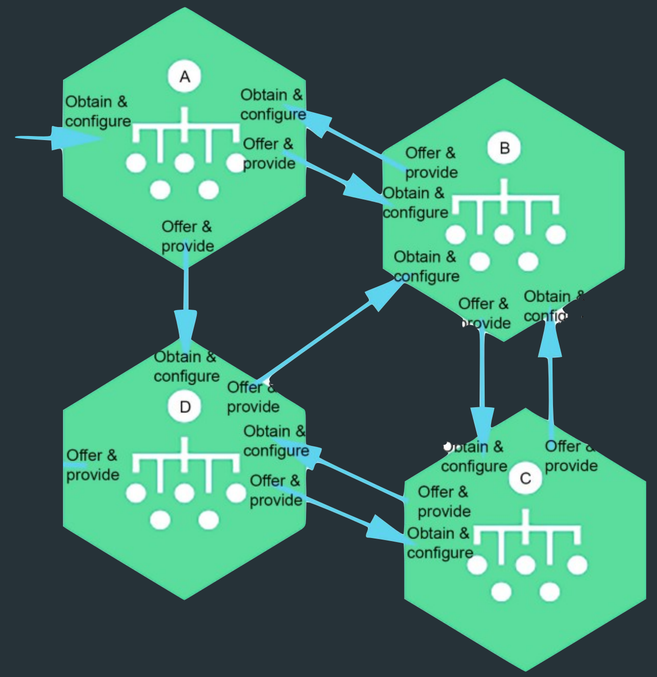


Choreography



Importance of Service Relationships

- Enabling service reuse
- Improving scalability and flexibility
- Simplifying system integration



How Services Communicate

- Define service communication and its role in SOA
- Explain the types of communication that can occur between services, including synchronous and asynchronous communication
- Discuss the benefits and challenges of each type of communication

Examples of Service Communication

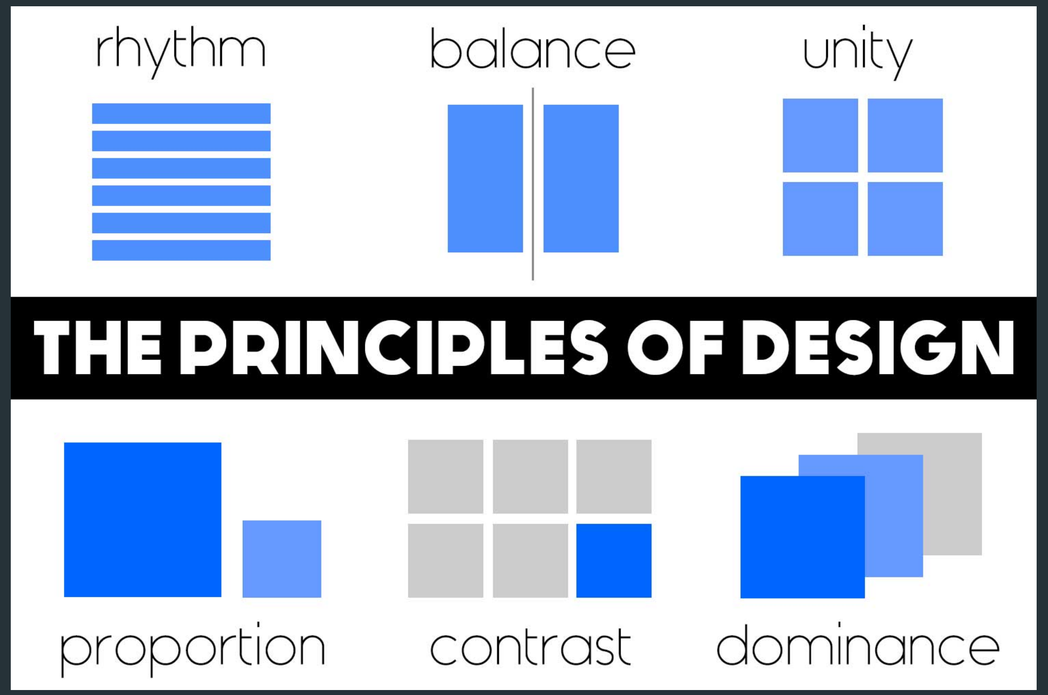
```
// Service 1 sends a request to Service 2
GET /api/orders/123 HTTP/1.1
Host: service2.example.com
Accept: application/json

// Service 2 responds with a JSON message
HTTP/1.1 200 OK
Content-Type: application/json

{
  "orderId": 123,
  "customerName": "John Doe",
  "status": "shipped"
}
```

Service Contracts and Communication

- Define service communication and its role in SOA
- Explain the types of communication that can occur between services, including synchronous and asynchronous communication
- Discuss the benefits and challenges of each type of communication

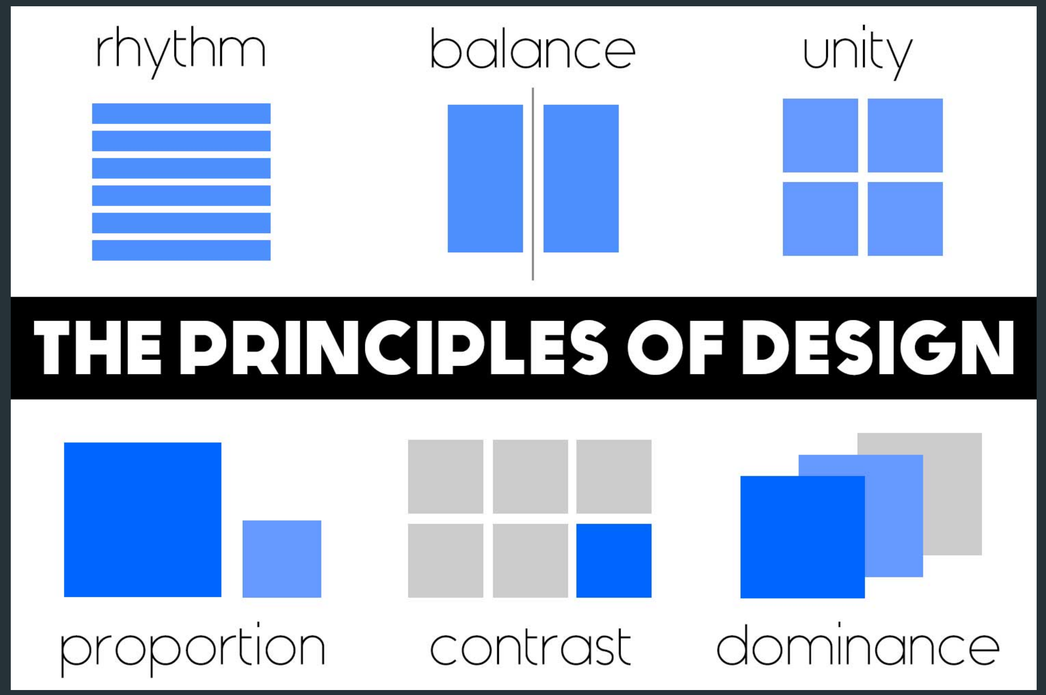
Service Design Principles

- Granularity and Cohesion
- Loose coupling and High cohesion
- Reusability



Service Design Considerations

- Service Contract
- Service Interface
- Service Implementation



Code Example

```
// Service contract
public interface AdditionService {
    int add(int a, int b);
}

// Service interface
@WebService
public class AdditionServiceImpl implements AdditionService {
    public int add(int a, int b) {
        return a + b;
    }
}
```

Service Building Blocks

- Service Registry and Repository
- Service Contract
- Service Endpoint

Service Implementation Technologies

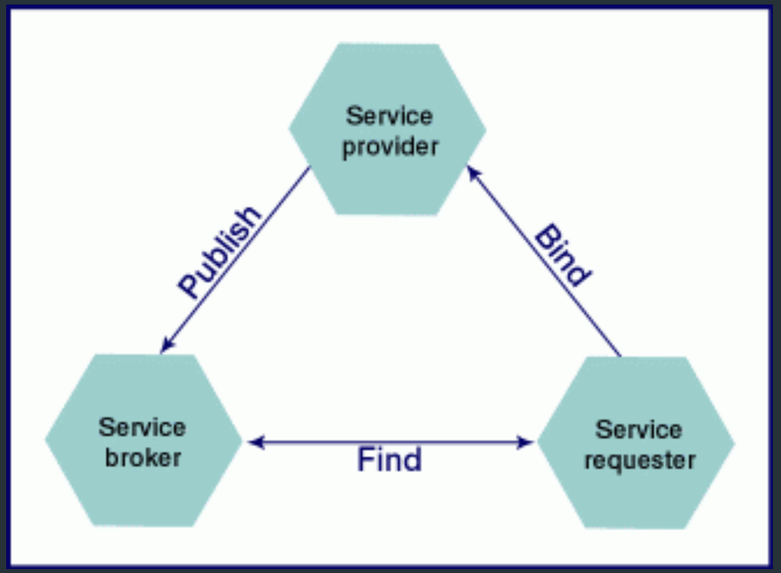
- SOAP
- REST
- Message Queues

Service Implementation Technologies

```
// Service contract
const express = require('express');
const app = express();
app.get('/users/:userId', (req, res) => {
  const userId = req.params.userId;
  const user = getUser(userId);
  res.send(user);
});

// Service endpoint
app.listen(3000, () => console.log('Server started on port 3000'));

// Service implementation
function getUser(userId) {
  // Retrieve user information from database
  return {
    name: 'John Doe',
    age: 30,
    email: 'johndoe@example.com'
  };
}
```

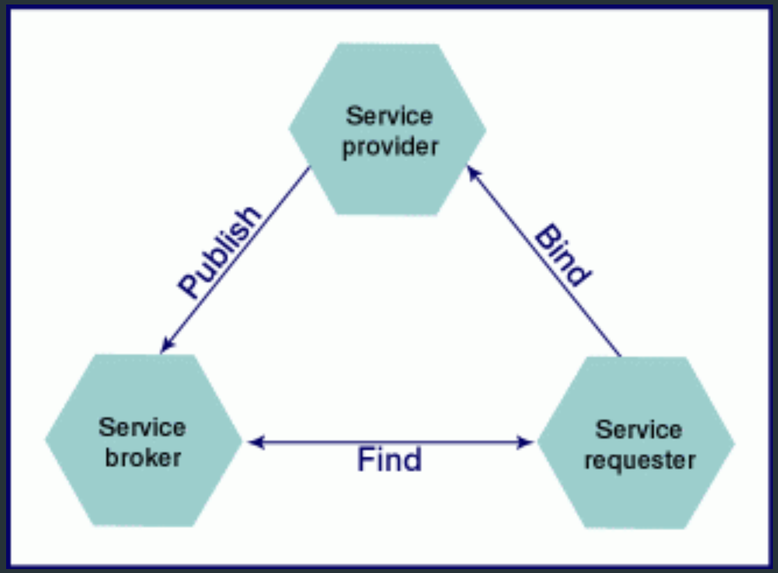
What is Primitive SOA?

- Simple and basic SOA approach
- Uses XML and SOAP
- Focuses on data exchange and interoperability



Advantages of Primitive SOA

- Easy to implement and understand
- Promotes interoperability
- Can be used with legacy systems



SOAP message in Java

```
// Create a SOAP message factory
SOAPMessageFactory factory = SOAPMessageFactory.newInstance();

// Create a new SOAP message
SOAPMessage message = factory.createMessage();

// Create a SOAP part and add it to the message
SOAPPart part = message.getSOAPPart();
SOAPEnvelope envelope = part.getEnvelope();
SOAPBody body = envelope.getBody();
SOAPElement element = body.addBodyElement(
  envelope.createName("HelloWorld", "ns1", "http://example.com"));
element.addChildElement(envelope.createName("name")).addTextNode("John");

// Create a client and send the message
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(HelloWorld.class);
factory.setAddress("http://localhost:8080/HelloWorld");
HelloWorld client = (HelloWorld) factory.create();
SOAPMessage response = client.sayHello(message);
```

What is Service-Orientation?

- Set of design principles for building services
- Loosely coupled, reusable, and interoperable
- Service contracts, abstraction, and autonomy

Service Contract in WSDL

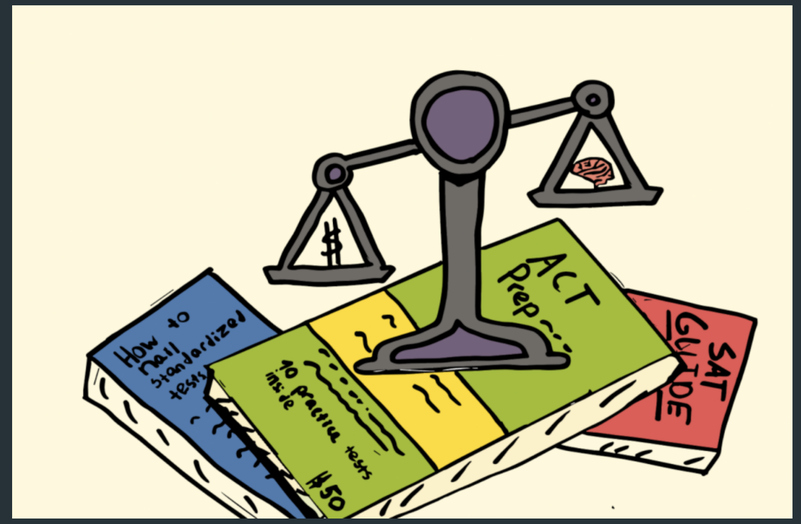
```
<definitions targetNamespace="http://example.com/weather"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
  <service name="WeatherService">
    <port name="WeatherPort" binding="tns:WeatherBinding">
      <soap:address location="http://example.com/weather"/>
    </port>
  </service>

  <portType name="WeatherPortType">
    <operation name="GetWeather">
      <input message="tns:GetWeatherRequest"/>
      <output message="tns:GetWeatherResponse"/>
    </operation>
  </portType>

  <binding name="WeatherBinding" type="tns:WeatherPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetWeather">
      <soap:operation soapAction="http://example.com/weather/GetWeather"/>
      <input>
        <soap:body use="literal"/>
      </input>
```

```
    <operation name="GetWeather">
      <input message="tns:GetWeatherRequest"/>
      <output message="tns:GetWeatherResponse"/>
    </operation>
  </portType>

  <binding name="WeatherBinding" type="tns:WeatherPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetWeather">
      <soap:operation soapAction="http://example.com/weather/GetWeather"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <message name="GetWeatherRequest">
    <part name="City" type="string"/>
  </message>

  <message name="GetWeatherResponse">
    <part name="Temperature" type="float"/>
  </message>
</definitions>
```

Common Principles of Service-Orientation

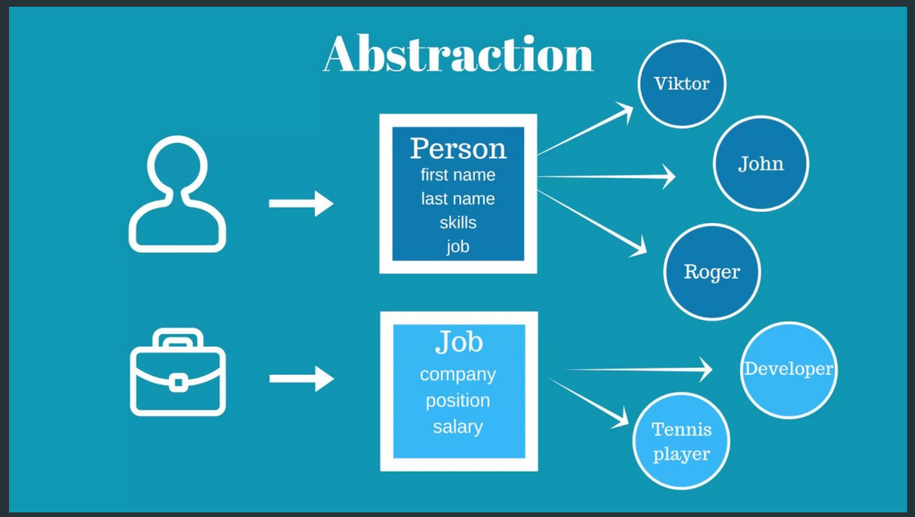
Principle 1 - Standardized Service Contracts

- Importance of standardized service contracts
- Designing service contracts
- Benefits of standardized service contracts



Principle 2 - Service Abstraction

- Importance of service abstraction
- Designing service abstractions
- Benefits of service abstraction

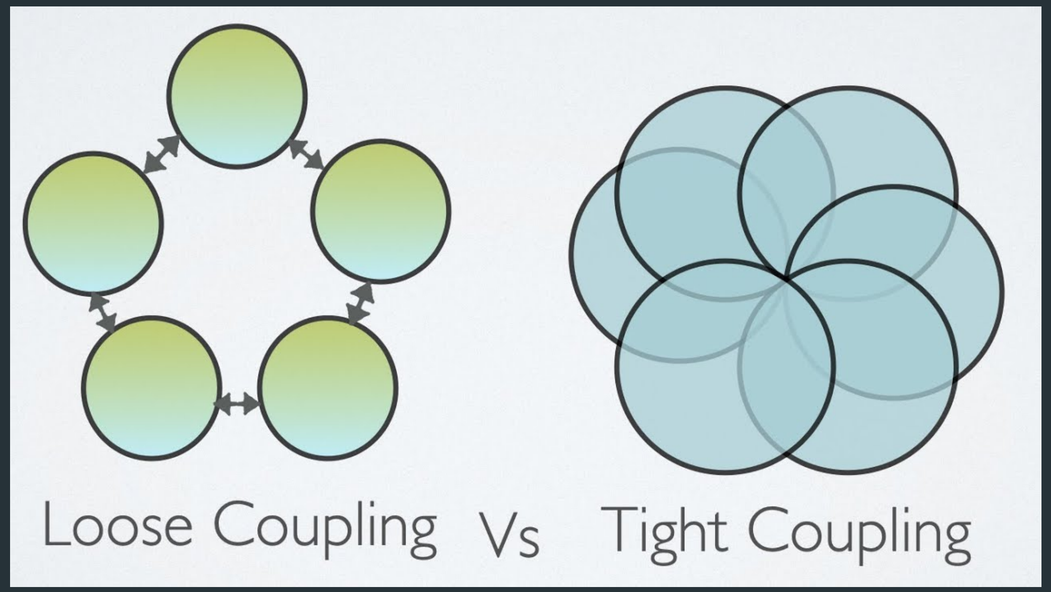


Principle 3 - Service Reusability

- Importance of service reusability
- Designing reusable services
- Benefits of service reusability

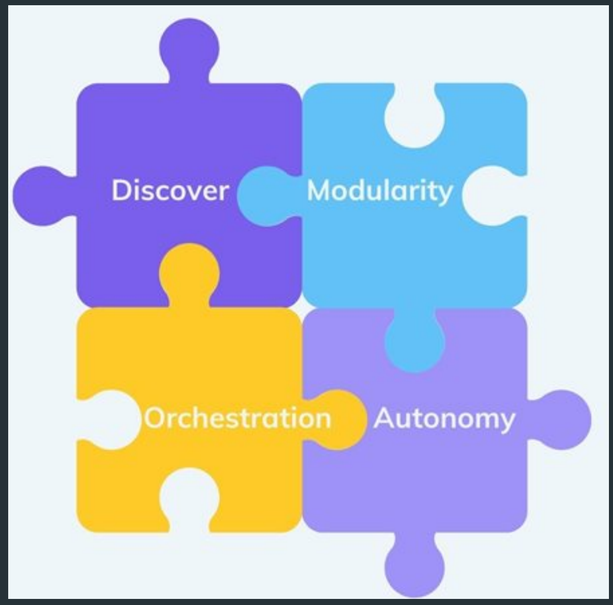
Principle 4 - Service Loose Coupling

- Importance of service loose coupling
- Designing loosely coupled services
- Benefits of service loose coupling



Principle 5 - Service Composability

- Importance of service composability
- Designing composable services
- Benefits of service composability



Lecture outcomes

- SOA
  - Fundamentals
  - Analogy
  - Encapsulation
  - Relationships
  - Communication
  - Design
  - Built
- Primitive SOA
- Service-Orientation