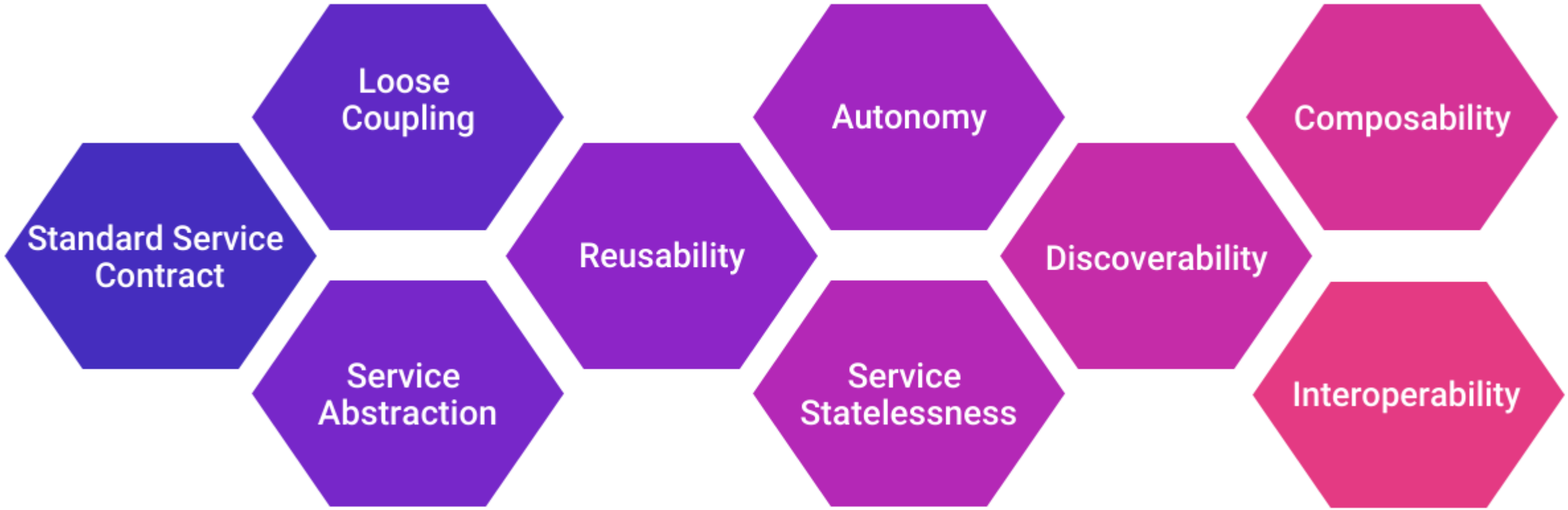
Basic Components of SoA

DR. VIPIN PAL

Service-Oriented Architecture Principles



spiceworks

Services

In the context of Service-Oriented Architecture (SOA), a service is a discrete unit of functionality that can be accessed remotely and acted upon independently. Services are self-contained, meaning they perform specific business functions and are designed to be reusable in different applications or contexts.

Characteristics:

Autonomous: Services operate independently without requiring other services to function.

Discoverable: They can be discovered and invoked by other components or services within the architecture.

Composability: Services can be combined or orchestrated to create more complex business processes.

Example: A payment processing service that can be used by multiple e-commerce platforms to handle transactions.

Examples

Payment Processing Service- Example: **PayPal or Stripe**

When you purchase something online, the e-commerce site calls a payment processing service like PayPal or Stripe to handle the transaction. The service ensures that funds are transferred from the customer’s bank or credit card to the merchant's account.

Authentication Service- Example: OAuth or Firebase Authentication

Weather Information Service- Example: OpenWeatherMap or Weather.com API

A travel app that shows you the current weather at your destination uses a weather information service to pull that data. It sends a request to the weather service with the location as a parameter and receives the weather details in response.

Components

Components are modular building blocks within an application that encapsulate specific functionality or data. In SOA, a component often implements the logic of a service, providing the operational behavior required to fulfill a service's contract.

Characteristics:

Reusability: Components can be reused across different parts of an application or across multiple applications.

Encapsulation: Components hide their internal details and expose functionality through well-defined interfaces.

Interchangeability: Components can often be replaced with others as long as they conform to the same interface.

Example: A user authentication component that handles login and session management for an application.

Service Contract

A contract in SOA is a formal agreement that specifies the terms under which a service operates, including the input and output data types, the operations available, and any constraints or policies that apply. Contracts define the obligations and expectations between service providers and consumers.

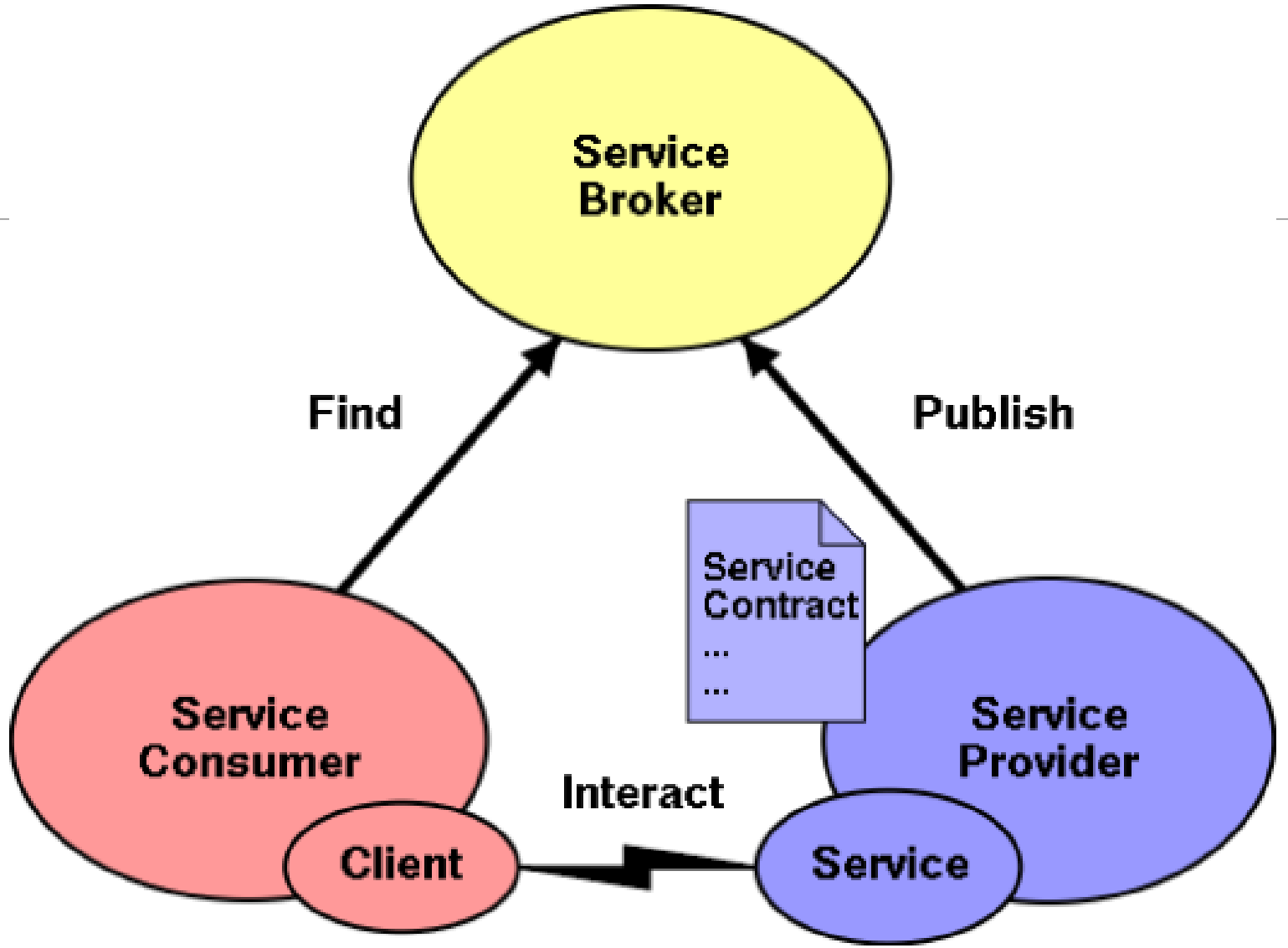
Characteristics:

Explicit: Contracts are explicitly defined, often using languages like WSDL (Web Services Description Language) in the case of web services.

Versioned: Contracts can be versioned to allow for updates without breaking existing consumers.

Binding: The contract is binding, meaning both service providers and consumers must adhere to the agreed-upon terms.

Example: A service contract for a weather API that specifies the required parameters (e.g., location) and the format of the returned weather data.


Service
Broker
Find
Publish
Service
Contract
...
...
Service
Consumer
Service
Provider
Interact
Client
Service

Importance of contracts in ensuring service consistency

Standardization of Interactions

Clear Boundaries and Expectations

Ensuring Interoperability

Error Handling and Fault Management

Version Control and Compatibility

Security and Compliance

Documentation and Transparency

Loose Coupling

Loose coupling refers to the degree to which services and components are interconnected yet independent of one another.

In SOA, loose coupling is a key principle that allows services to be developed, deployed, and managed independently while still interacting effectively.

Characteristics:

Minimal Dependencies: Services have minimal dependencies on each other, reducing the impact of changes in one service on others.

Flexibility: Loose coupling enables flexibility in service composition, allowing different services to be combined in various ways.

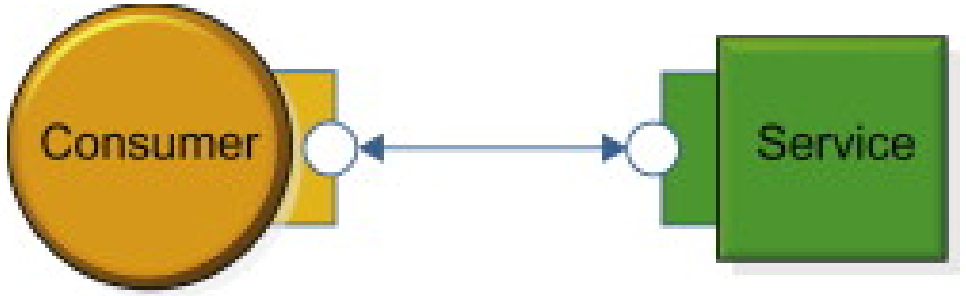
Resilience: Systems with loosely coupled services are more resilient to failures, as the failure of one service does not cascade to others.

Example: An order processing system that can function even if the inventory service is temporarily unavailable, using a fallback mechanism.

Consider an instance wherein you have created two classes in a program: A and B.

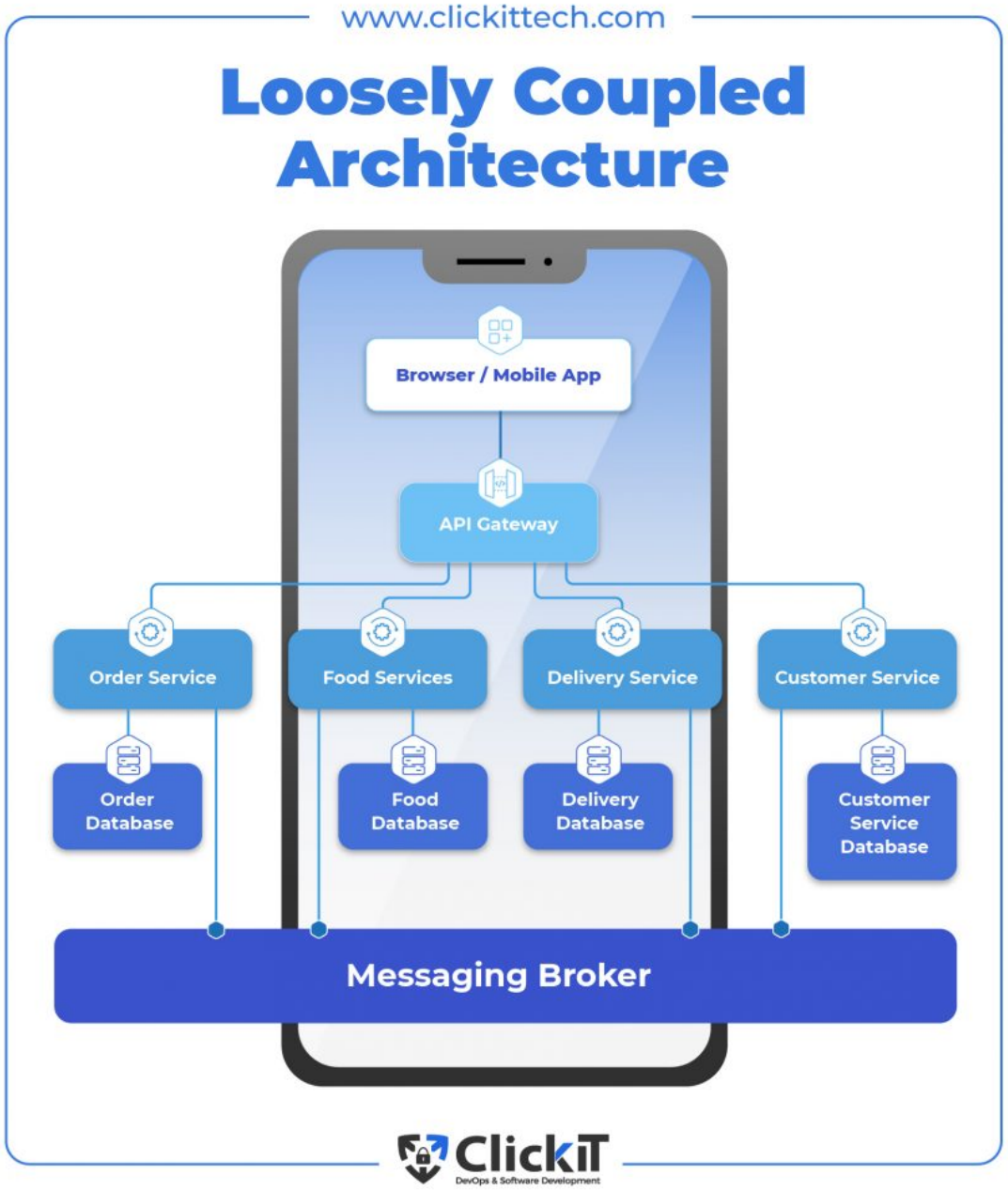
When a method of Class A calls the method of Class B or uses variable instances defined in Class B, both classes are tightly coupled.

However, when Class A depends on the interface of Class B instead of the methods defined in Class B, both classes are loosely coupled.

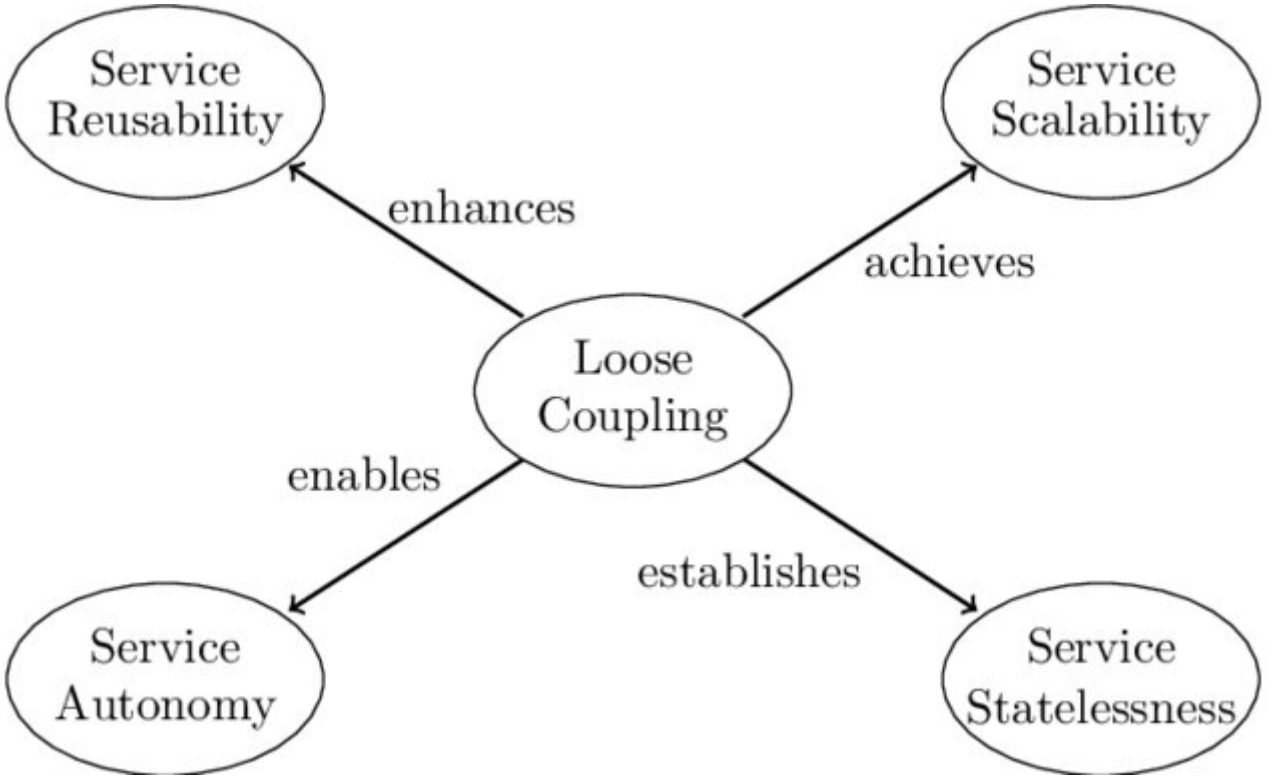


Tightly Coupled
(consumer interacts directly with the service)

Loosely Coupled
(consumer and service interact via messaging and the ESB)


www.clickittech.com
Loosely Coupled Architecture
Browser / Mobile App
API Gateway
Order Service
Food Services
Delivery Service
Customer Service
Order Database
Food Database
Delivery Database
Customer Service Database
Messaging Broker
ClickIT
DevOps & Software Development

Relation between coupling and other SOA principles



Benefits of loose coupling

Enhanced Flexibility and Agility

Improved Scalability

Increased Reusability

Improved Reliability and Resilience

Better Interoperability

Easier Testing and Debugging

Cost Efficiency

Interoperability

Interoperability is the ability of different services, systems, or components to work together seamlessly, regardless of their underlying technologies, platforms, or implementations.

In SOA, interoperability ensures that services can interact with one another across diverse environments.

Characteristics:

Cross-Platform Compatibility: Services are designed to work across different operating systems, programming languages, and network protocols.

Standardization: The use of standardized protocols (e.g., SOAP, REST) and data formats (e.g., XML, JSON) facilitates interoperability.

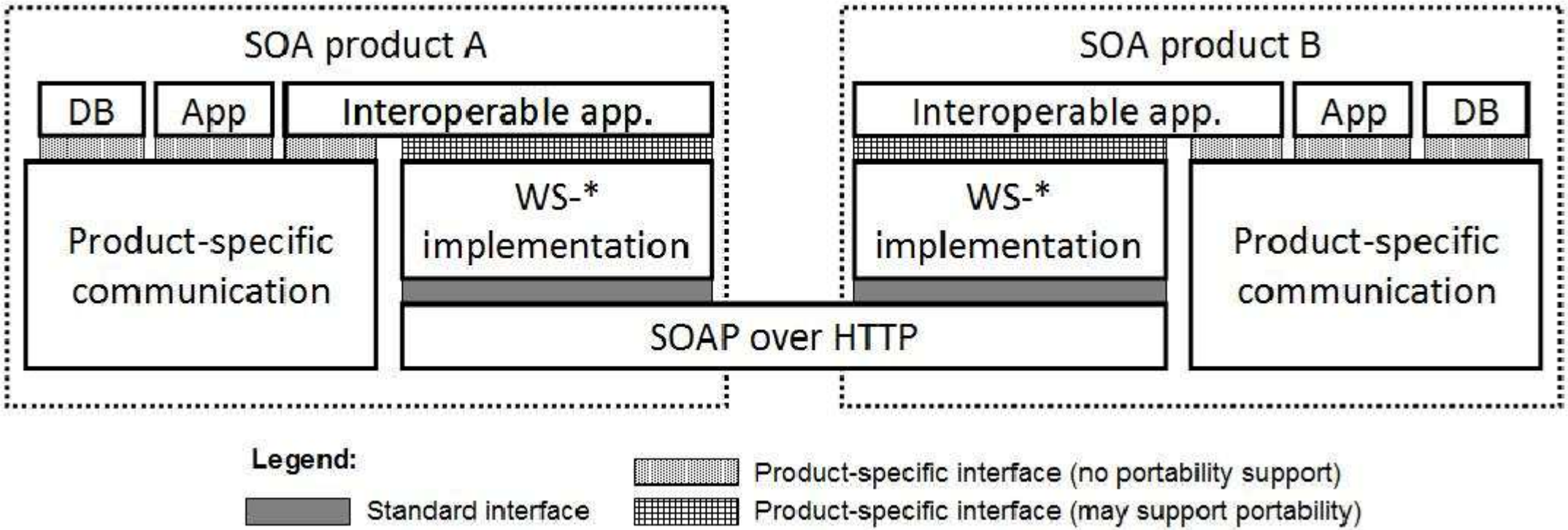
Scalability: Interoperable services can be scaled and extended to work with new systems as they are introduced.

Example: A CRM system that integrates with various third-party applications like email marketing tools, social media platforms, and analytics services, regardless of the technologies they use.

Interoperability in heterogeneous environments is crucial because it enables diverse systems, applications, and technologies to work together seamlessly.

In today’s complex IT landscapes, organizations often use a mix of legacy systems, modern applications, cloud services, and third-party platforms.

Interoperability ensures that these disparate systems can communicate, share data, and collaborate effectively.


SOA product A
DB
App
Interoperable app.
Product-specific communication
WS-* implementation
SOA product B
Interoperable app.
App
DB
WS-* implementation
Product-specific communication
SOAP over HTTP
Legend:
Standard interface
Product-specific interface (no portability support)
Product-specific interface (may support portability)