Service Oriented Architecture (SOA)

Services and microservices

Distributed software systems

CdLM Informatica - Università di Bologna

Agenda

- Services and APIs
- Service Oriented Architecture
- The REST style
- Microservices

The service economy

- Modern economies rely upon *services*
- This means that some companies **offer support** for activities in the primary (agricolture) or secondary (industry of goods) sectors, or to other companies offering services themselves
- A service company is usually independent (**not owned**) from other companies and their services; however in several cases they should cooperate or at least be **coordinated** in some way

Attributes of physical services

- A service is not owned by its user (compare with *product*)
- It has a well defined, easy-to-use, standardized interface
  - New services can be offered by combining existing services
- (almost) always available but idle until requests come
- "Provision-able" (used by someone only when necessary, then reassigned to someone else)
- A service should be easily accessible and usable readily
  - no "integration" required
  - Self-contained: no visible dependencies to other services
- A service is usually coarse grained
- It is independent from the consumer's context
  - but a service can have a context
- It should have a quantifiable quality of service
  - Services do not compete on "What" but on "How"

Example: mail

- Customers use mail everywhere
- Interfaces: PostOffice, Mail Box, stamp, postman
- Apparently no relationship with other services (eg. transportation, payments)
- Quantifiable quality of service: price, delivery time, lost messages

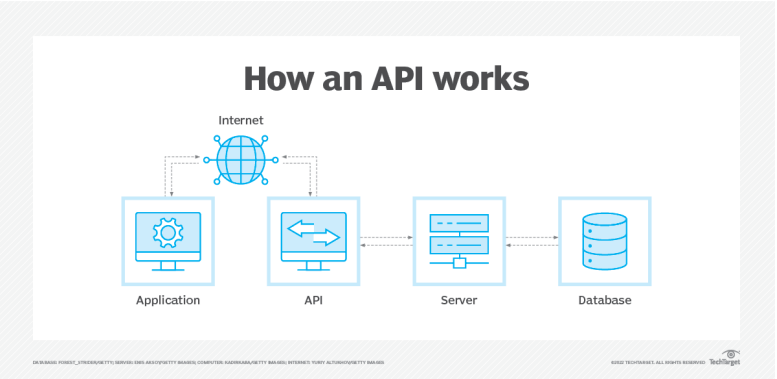
The API economy

No car

No storage

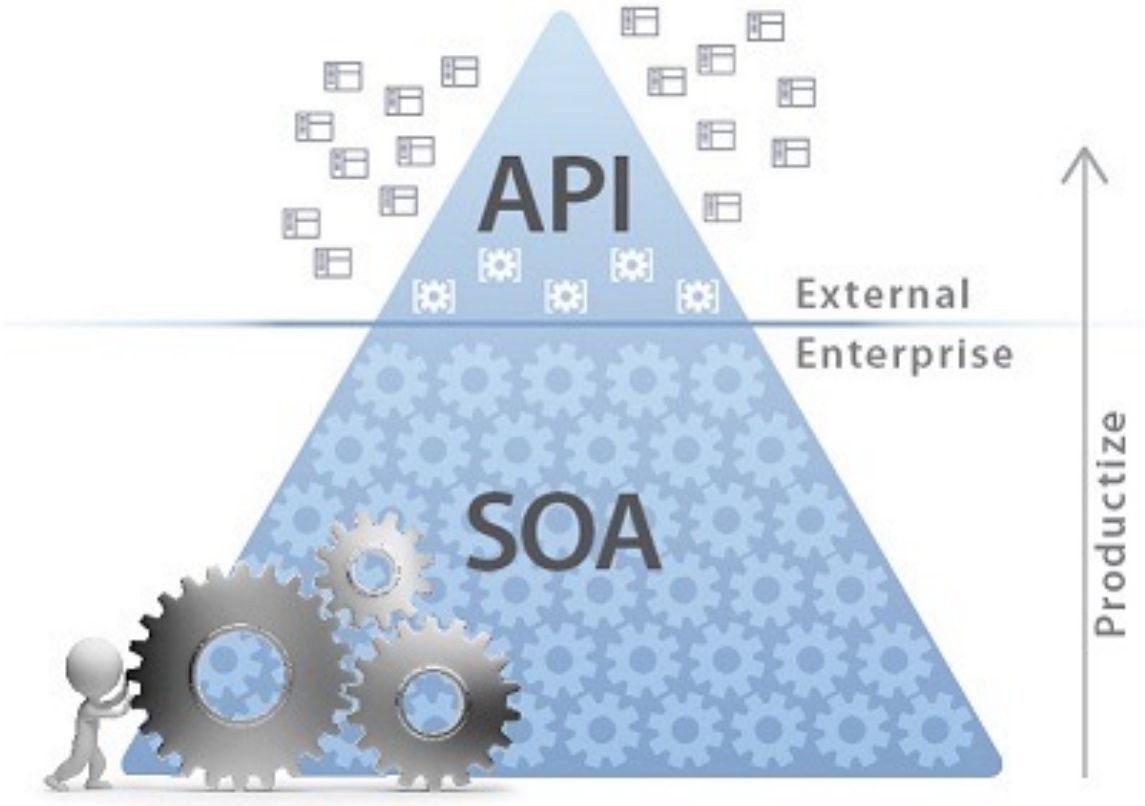
No hotel

Examples of API economy



- **Google Maps API.** Google lets third parties such as Uber use its Maps API. The ride-sharing company then uses the API to power its maps.
- **Slack API.** Slack provides its API to enable integrations with its platform. For example, the cross-platform collaboration software, Mio, uses the Slack API to send and receive messages across Slack and Microsoft Teams.
- **Airbnb API.** The Airbnb API is provided to third-party developers to create integrations and other content around Airbnb. For example, the sales and channel management platform HotelRunner uses the Airbnb API to let hosts list their rooms on Airbnb directly.
- **Square API.** The payment processor Square offers its API to businesses to process payments, manage subscriptions and provide other features such as sending invoices. For example, the food delivery company Postmates uses Square's API to push orders from its ordering app to Square's point-of-sale system.

API and SOA: their relationship



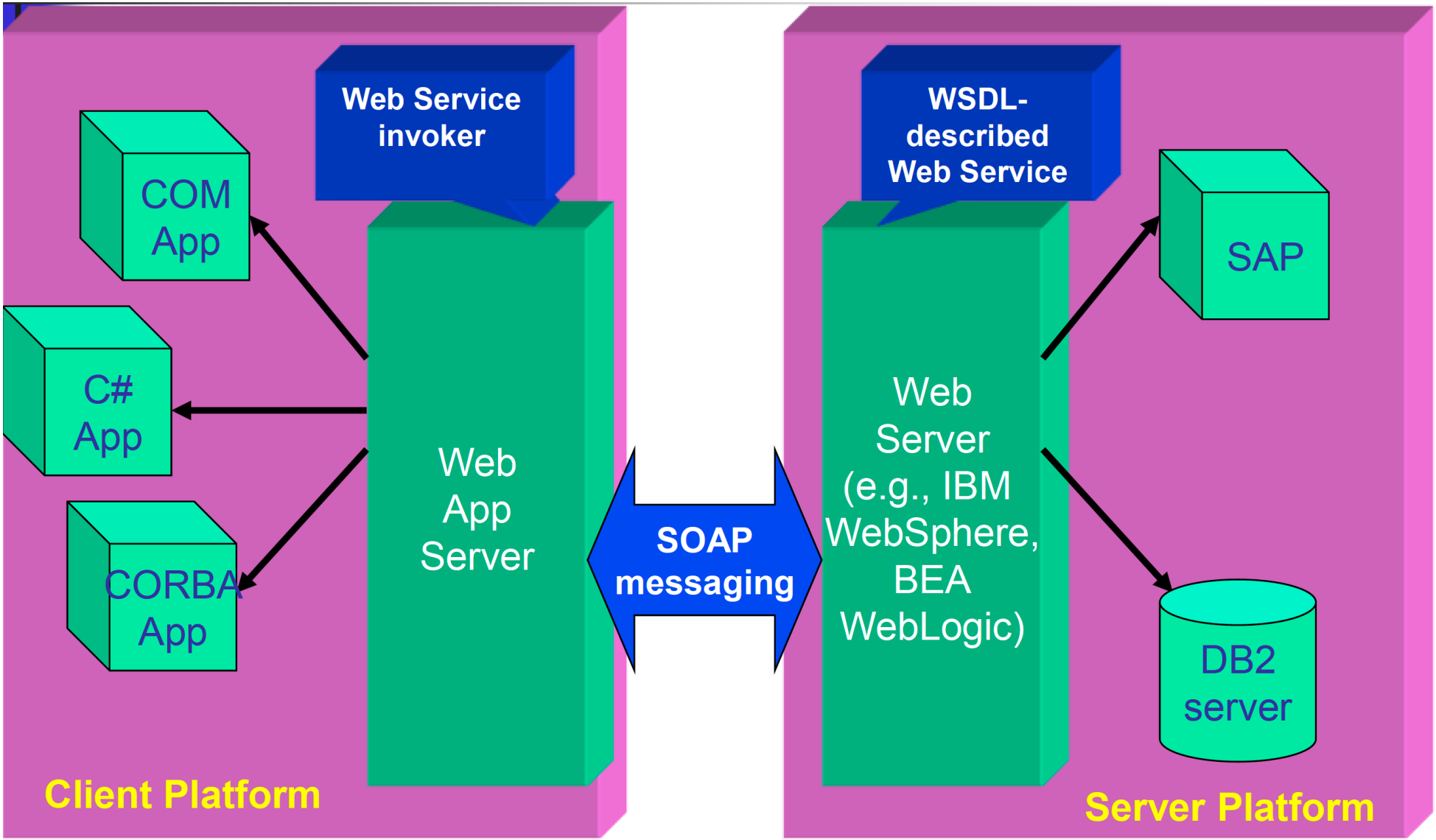
What is a SOA

«Service-Oriented Architecture» is a way of designing, developing, and managing distributed systems, in which

- Services provide reusable business functionality via well-defined interfaces.
- Service consumers are built using functionality from available services.
- A SOA infrastructure enables discovery, composition, and invocation of services.
- Protocols are predominantly, but not exclusively, message-based document exchanges.

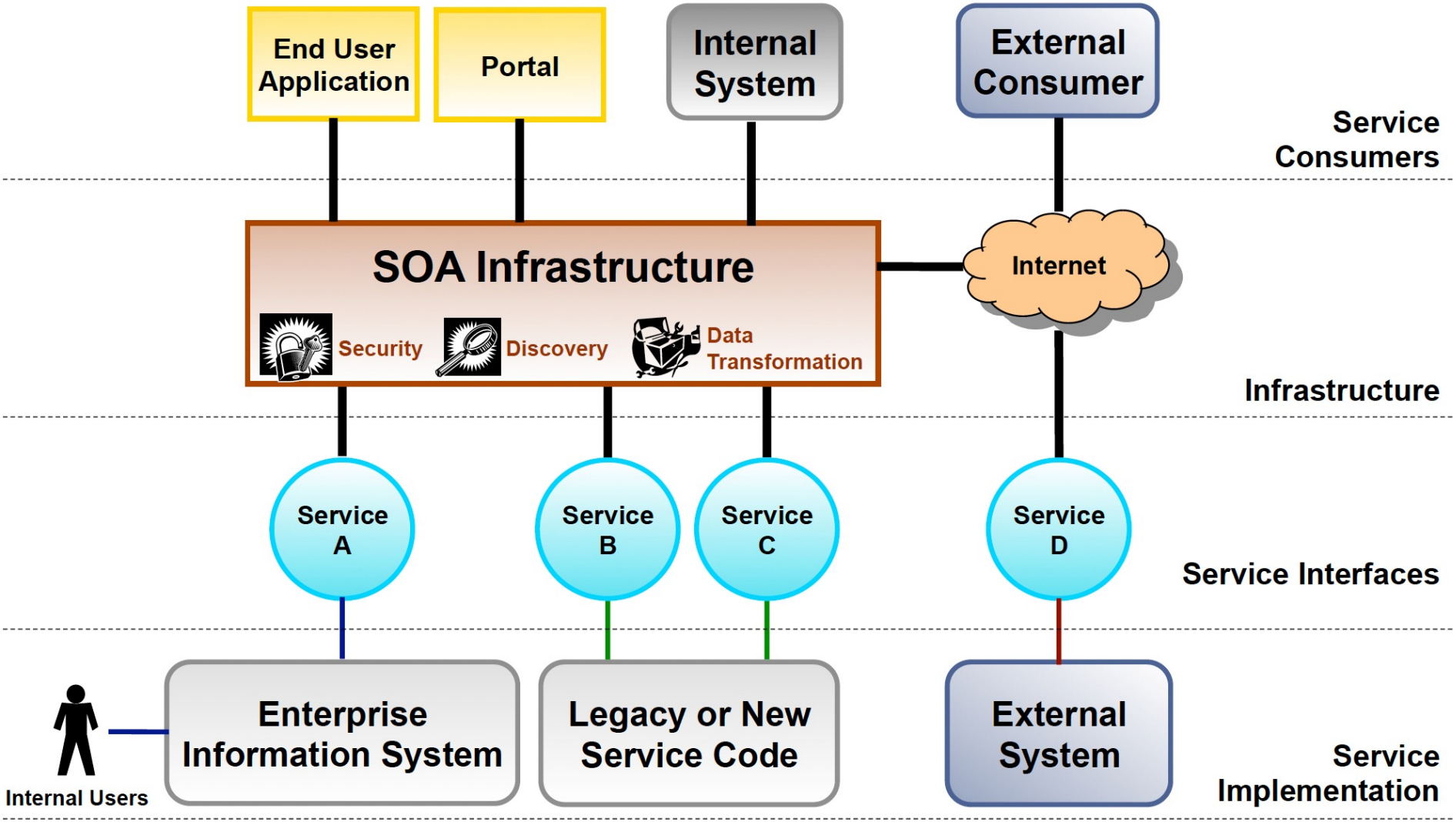
SOA principles: loose coupling, service reusability, and platform independence

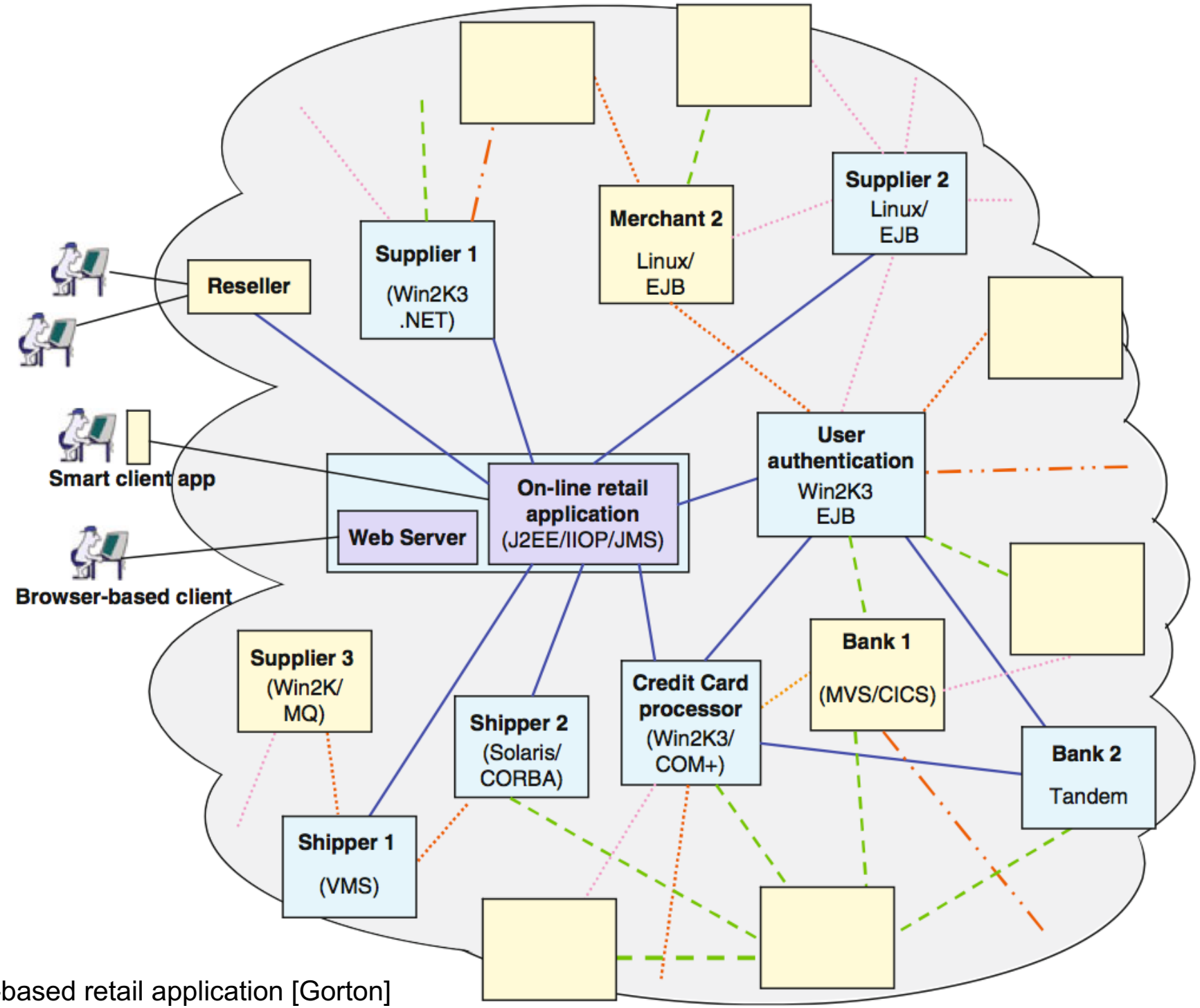
Entering a SOA via a Web Service



Example: https://developer.amazonservices.com

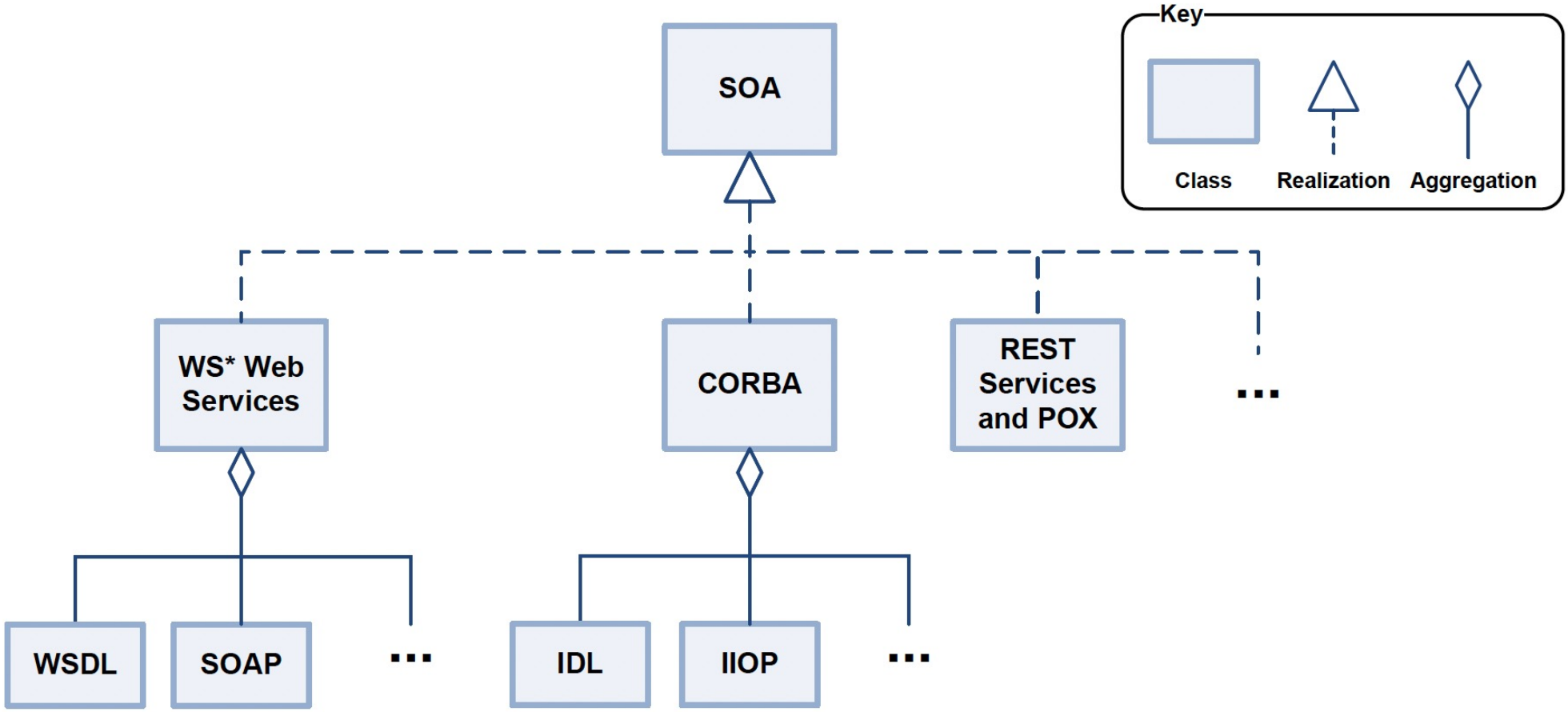
Components of a SOA





Service-based retail application [Gorton]

SOA is a generic term

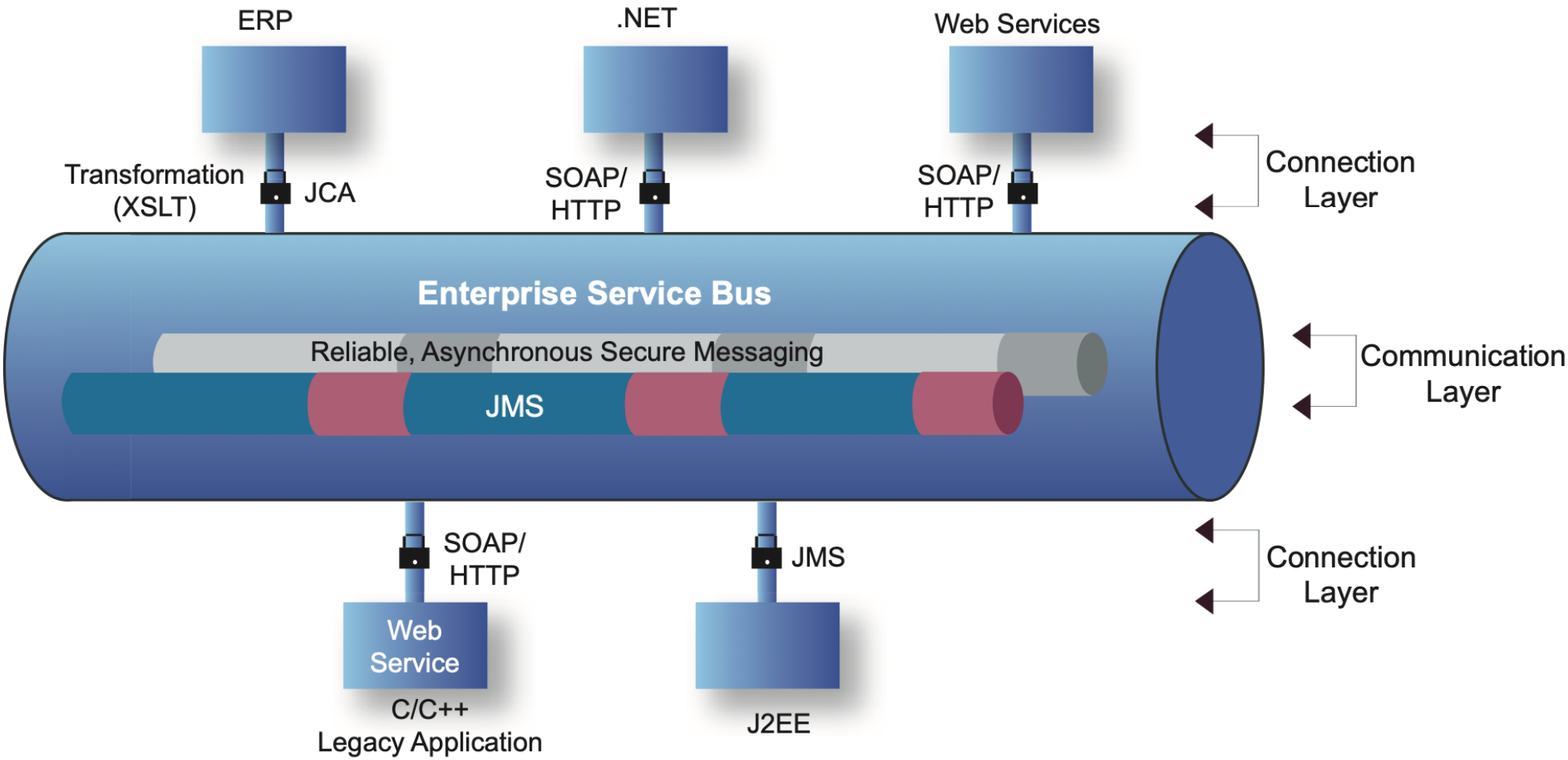


SOA is an architectural pattern
Systems that implement the SOA pattern are called *service oriented*
Web Services is one technology for SOA implementation

The essence of SOA

- **Use other services as RPC servers for your app**
- Web 1.0: large sites organized this way *internally*
  - Yahoo!, Amazon, Google, …: *external* "Services" available, but complex: Doubleclick ads, Akamai
- XML based Web Services msgs and descriptions
- Web 2.0: public service API's
  - Services: Google API, Amazon CloudFront...
  - Platforms: Facebook, Google Maps, ...
  - Mashups, e.g. housingmaps.com
  - User-composable services, e.g. Yahoo Pipes

How to build a SOA (example)

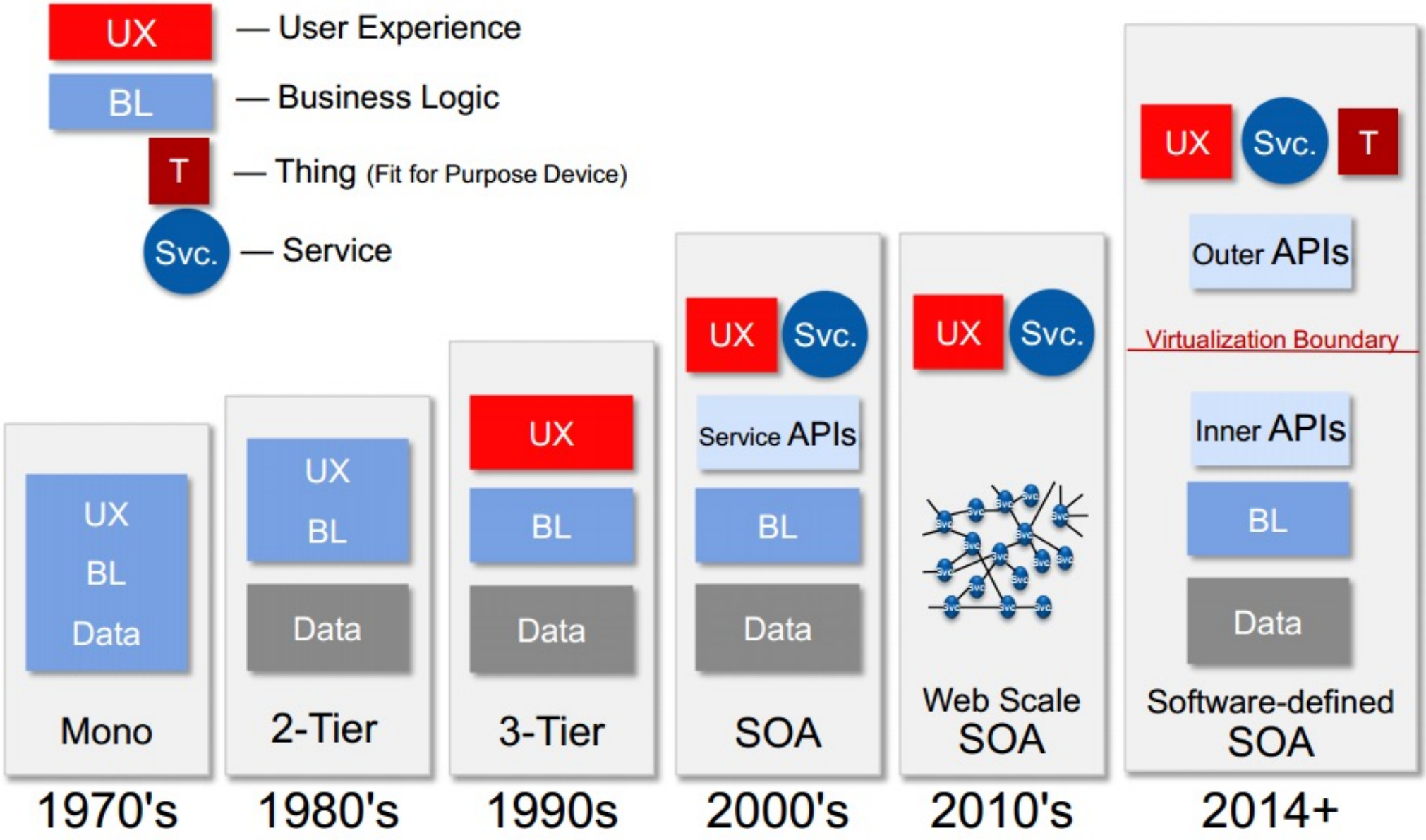


JMS for communication
Web services, J2EE, and .NET for connectivity to various systems
Extensible Stylesheet Language Transformation (XSLT) and Xquery for transformation
Lightweight directory access protocol (LDAP), secure sockets layer (SSL), and others for security

SOA and Web Services

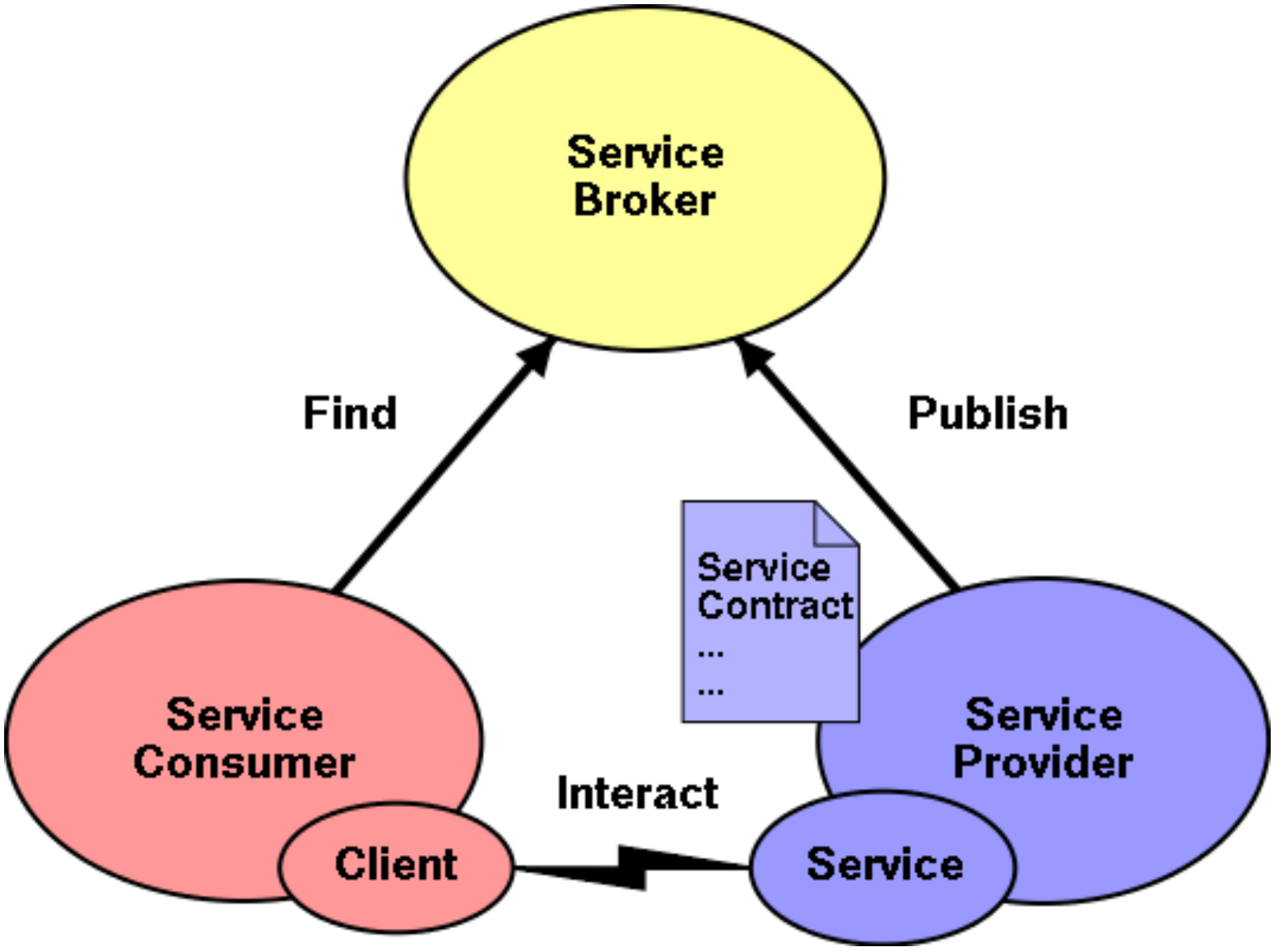
- Web services define a web technology that can be used to build distributed applications that can send /receive messages using SOAP over HTTP
- SOA is an architectural model for implementing loosely coupled service based applications.
- Web services can be used to implement SOA applications.
- Even if the web service approach to SOA has become very popular, it is not the unique method of implementing SOA, that can also be implemented using any other service-based technology (e.g. CORBA and REST).

Software-defined Applications on the Application Architecture Road Map

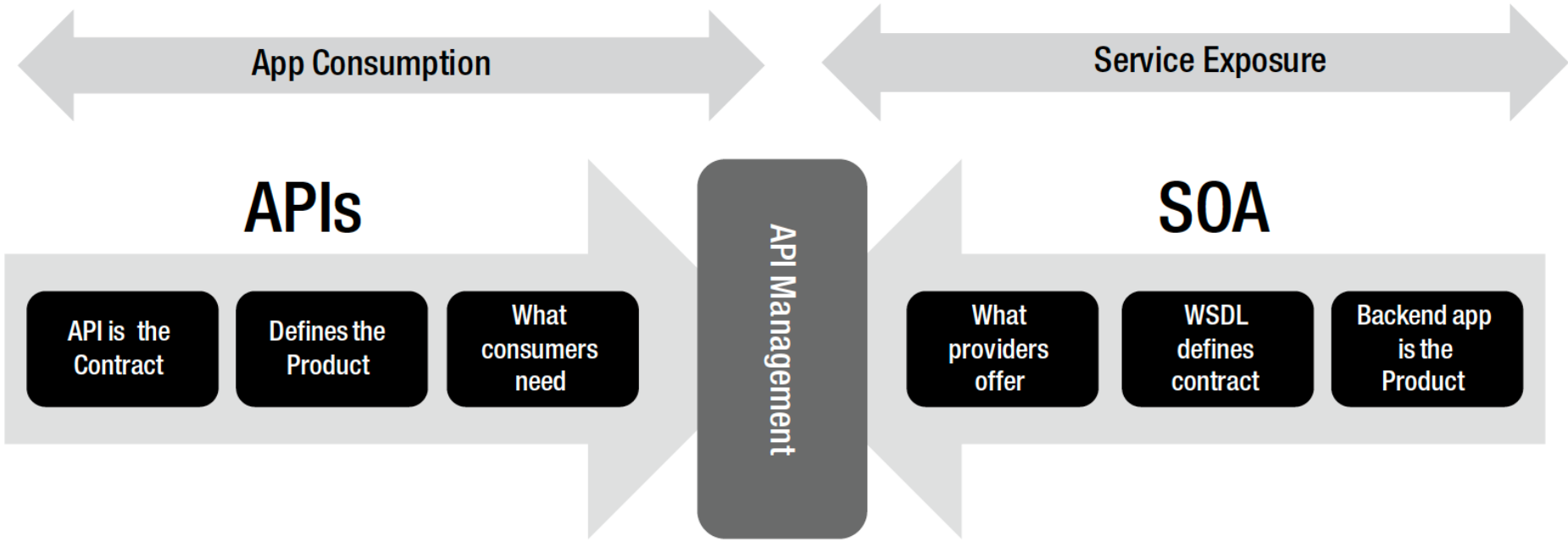


Gartner

The SOA style

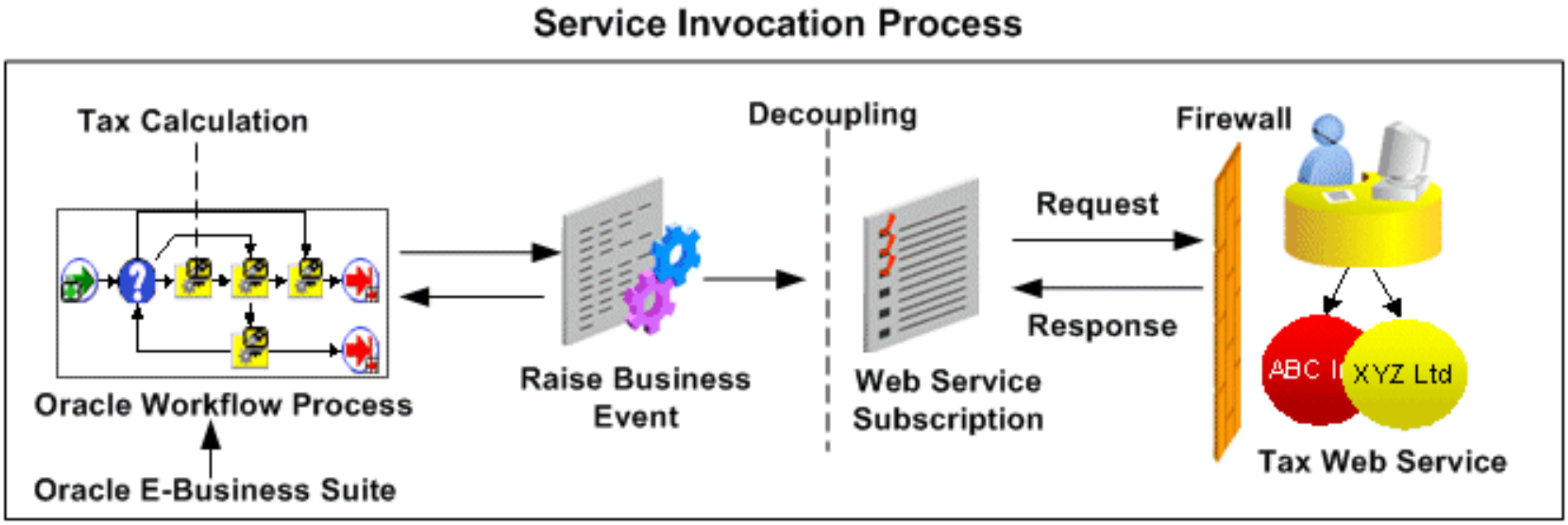


API vs SOA



- API technology focuses on the consumption of the back-end services created using SOA principles.
- APIs can be thought of as an evolution of SOA: creating and exposing reusable services.
- The main difference between them is that APIs are focused more on making consumption easier, whereas SOA is focused on control and has an extensive and well-defined description language

SOA: example


Service Invocation Process
Tax Calculation
Oracle Workflow Process
Oracle E-Business Suite
Raise Business Event
Decoupling
Web Service Subscription
Request
Response
Firewall
XYZ Ltd
Tax Web Service

Main SOA Standards (the SOA soup)

- **XML** (Extensible Markup Language): a data markup language for Web services
- **SOAP** (Simple Object Access Protocol): a W3C-approved standard for exchanging information among applications
- **WSDL** (Web Service Description Language): a W3C-approved standard for using XML to define Web services
- **WADL** (Web ApplicationDescription Language): is the REST equivalent of WSDL, WADL is based on XML and models the resources provided by a service and the relationships between them
- **UDDI** (Universal Description, Discovery, and Integration): an OASIS-approved standard specification for defining Web service registries
- **WS-Reliability** (Web Services Reliability): a SOAP-based protocol for exchanging SOAP messages, with delivery and message-ordering guarantees
- **WS-Security** (Web Services Security): a SOAP-based protocol that addresses data integrity, confidentiality, and authentication in Web services
- **JEE**: the Java Platform, Enterprise Edition, with APIs for deploying and managing Web services
- **WSIF** (Web Services Invocation Framework): an open source standard for specifying, in WSDL, EJB implementations for the Web server
- **WSRP** (Web Services for Remote Portlets): an OASIS standard for integrating remote Web services into portals
- **BPEL** (Business Process Execution Language): a standard for assembling sets of discrete services into an end-to-end business process

Standardizing bodies for SOAs

- **OMG** (1989)
- **W3C** (established 1994)
- **OASIS** (1993), consortium of former GML providers, deals with applications using XML
  - **WS-I** (2002), promotes interoperability among the stack of Web Services specifications http://www.ws-i.org

Typical issues in SOA

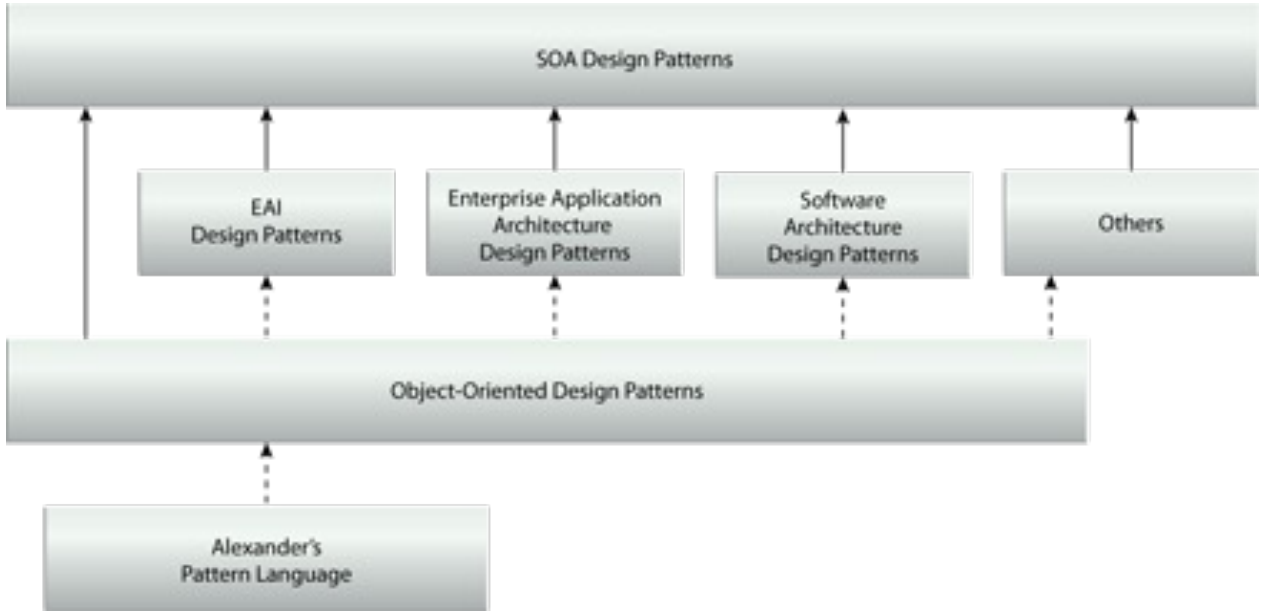
- Model, design, and implement a SOA
- Automate business processes by mapping them to the architectural model
- Orchestrate services and execute processes with the Business Process Execution Language (BPEL)
- Choreography describes a global protocol governing how individual participants interact with one another
- Achieve interoperability within a SOA using proven standards and best practices
- Secure and govern an enterprise SOA

Elements of SOA Design

- Business Modeling
- Service Oriented Architectural Modeling and Design
- Model Driven assumptions (loose coupling)
- Distributed objects and MOM (Message Oriented Middleware) for component-based sw systems

SOA design patterns http://soapatterns.org

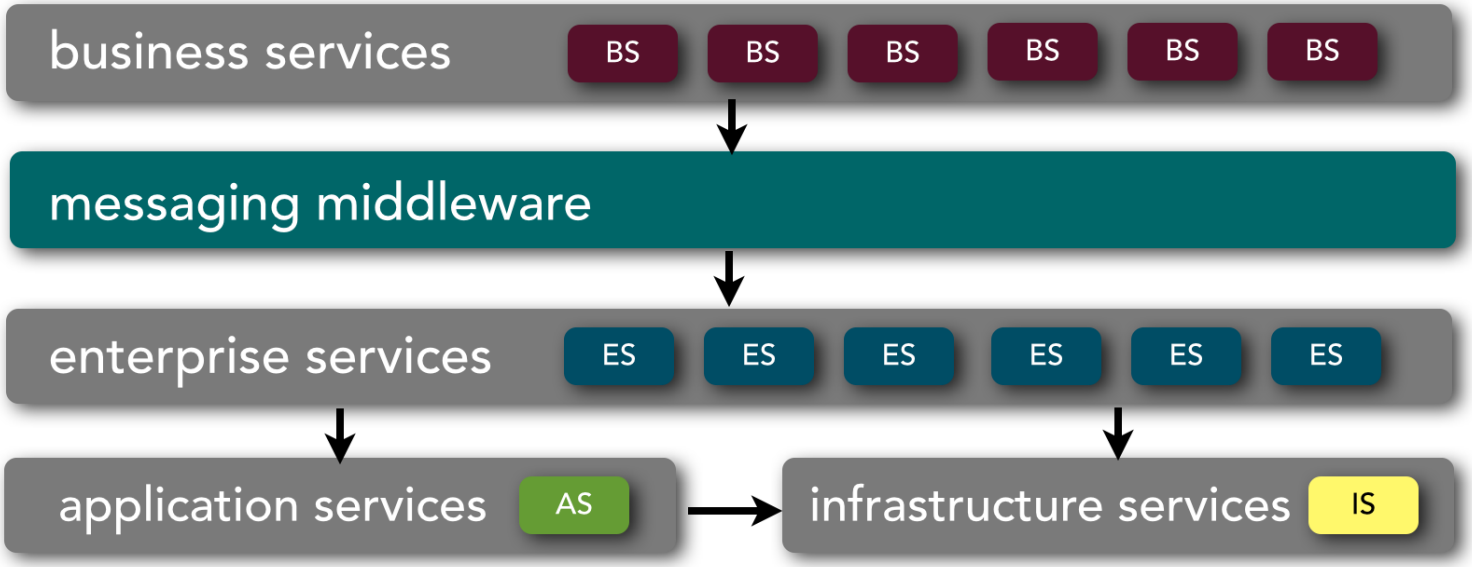
- Service-orientation has deep roots in distributed computing platforms,
- Many SOA design patterns can be traced back to established design concepts, approaches, and previously published design pattern catalogs



SOA patterns related to other patterns

Services and SOAs

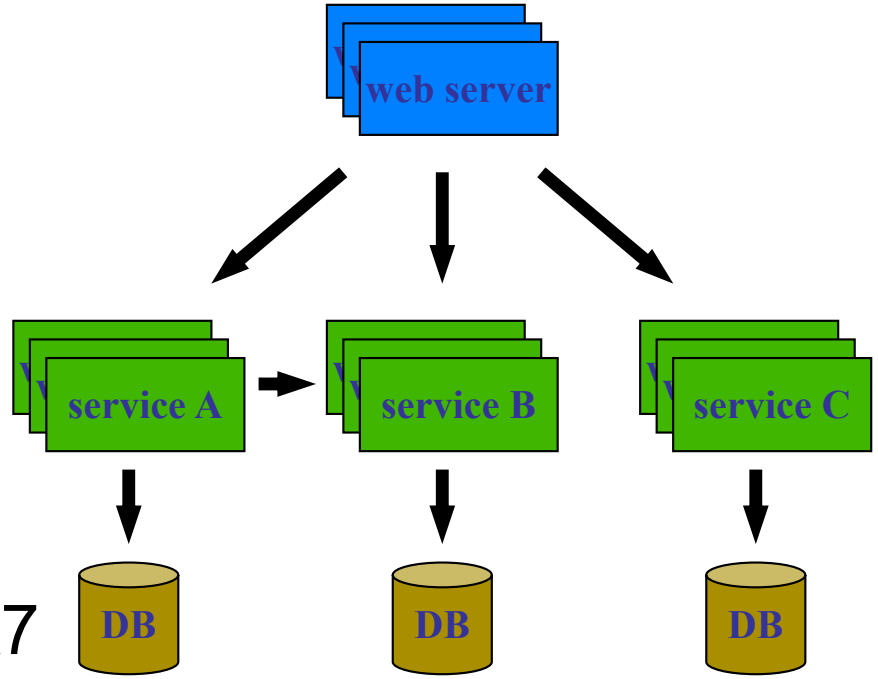
- A **service** is a program interacting via message exchanges
  - Web Services: all messages and service descriptions are written in XML
- A SOA is a set of **deployed** services cooperating in a given task



https://www.oreilly.com/radar/microservices-vs-service-oriented-architecture/

Example Web 1.0 SOA: amazon.com

- ~50 “two-pizza” teams of “developer/operators”
- ~10 operators
  - monitor the whole site
  - page the resolvers on alarm
- ~1000 resolvers
  - 10-15 per team, 1 on-call 24x7
  - monitor own service, fix problems
- Over 140 code change commits/month
- SOA (like Yahoo, Google, others)



P. Bodík et al., *Advanced Tools for Operators at Amazon.com,* Proc. ICAC 2005

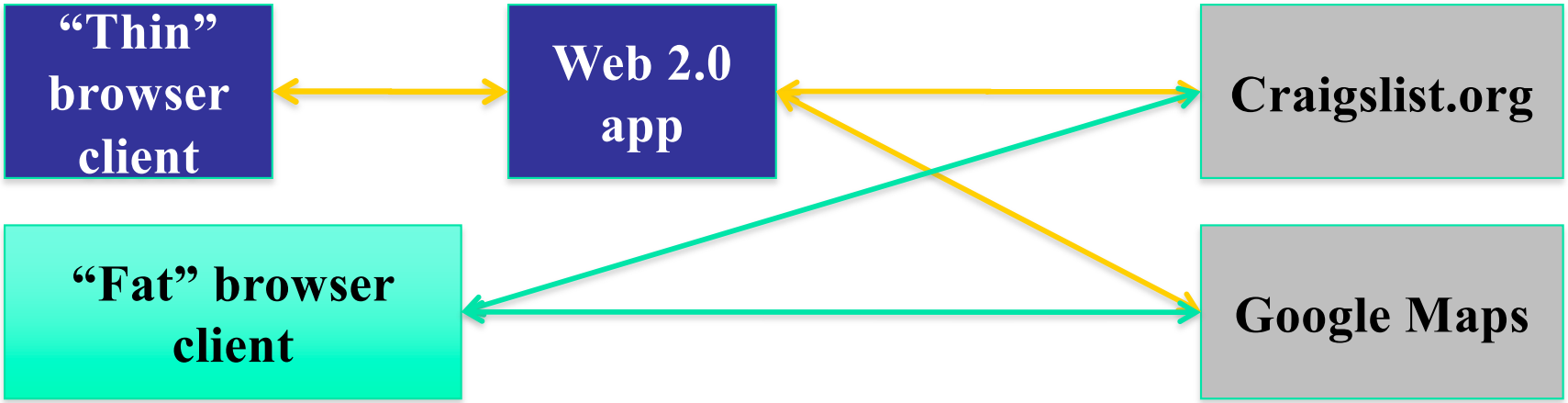
SOA based on RPC

- Transport: HTTP(S)
- Data interchange:
  - XML DTD (e.g., RSS)
  - JSON (Javascript Object Notation)
- Request protocol:
  - SOAP (Simple Object Access Protocol)
  - JSON-RPC
- See also WebHooks www.webhooks.org

AJAX vs SOA

- AJAX: client ⇔ server
  - A client makes async requests to a HTTP server
  - client-side JavaScript upcall receives reply and decides what to do
  - response includes XHTML/XML to update page, or JavaScript to execute
- SOA: server ⇔ server or client ⇔ server
  - An initiator makes (sync or async) requests to an HTTP server
  - In the past, initiator was a server running some app
  - today, JavaScript clients can exploit this approach

Two ways to do it: thin or fat clients



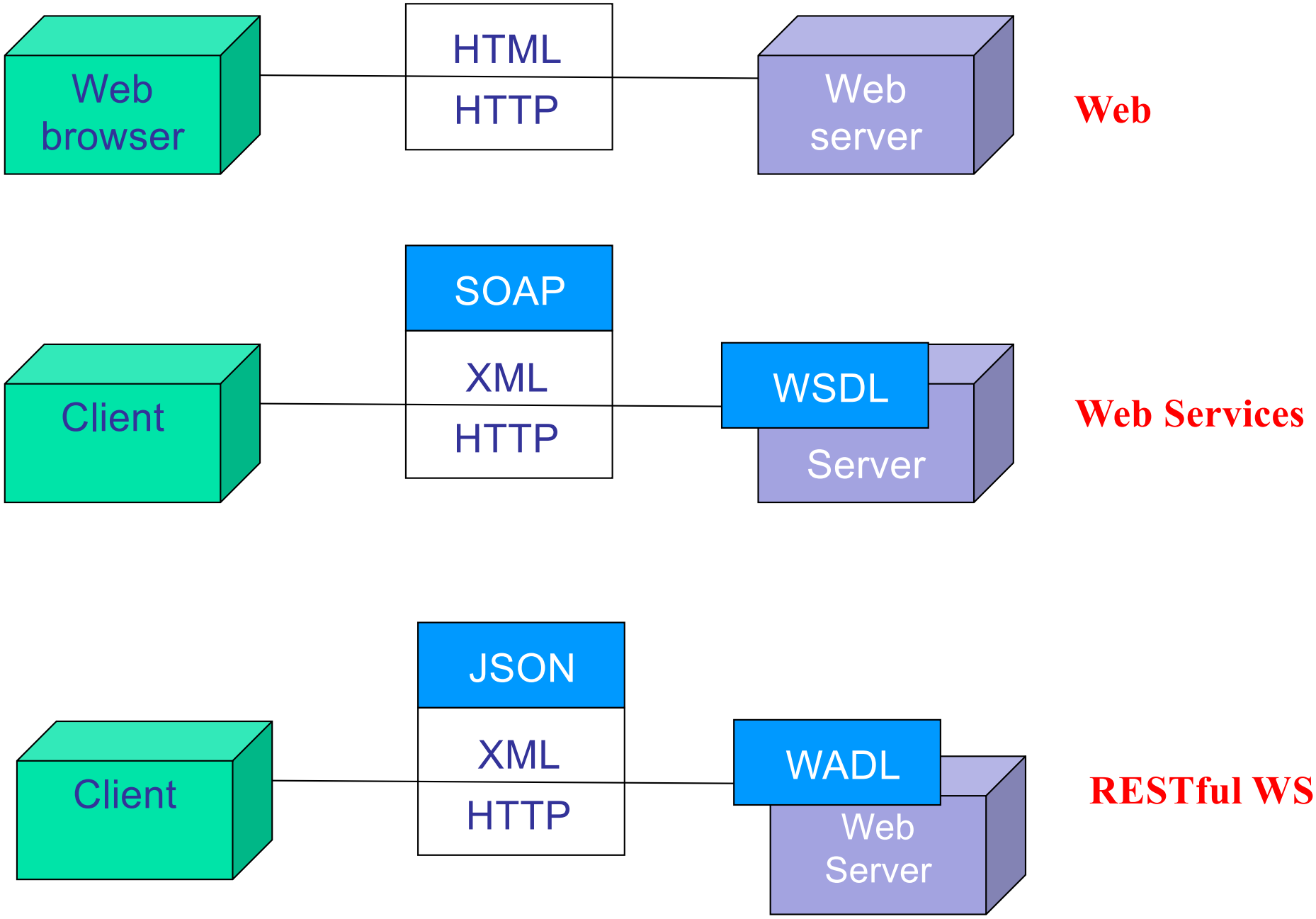
+ Client portability

+/– Client performance (both app download & JavaScript execution)

+ Availability of utility libraries for app development

– Privacy/trustworthiness of aggregator app

– Caching


Web browser
HTML
HTTP
Web server
Web
Client
SOAP
XML
HTTP
WSDL
Server
Web Services
Client
JSON
XML
HTTP
WADL
Web Server
RESTful WS

REST (Representational State Transfer) style

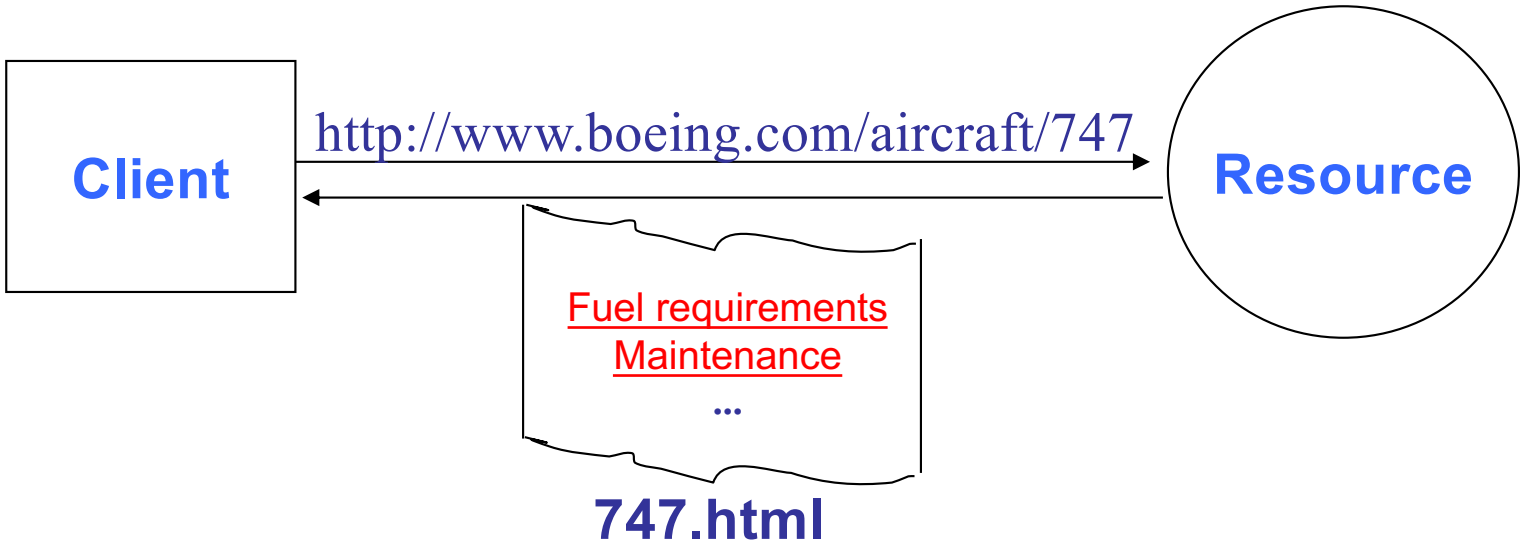
- Architectural style:
  - Client-server, stateless, cache
  - description of properties that made Web 1.0 successful by constraining SOA interactions
- In context of SOA for Web 2.0
  - HTTP is transport; HTTP methods (get, put, etc.) are the only commands
  - URI names are a resource
  - Client has resource ⇔ has enough info to request *modification* of the resource on server
  - A cookie can encode part of transferred state
- If an app is RESTful, it is easy to "SOA"-ify

REST style

- Representation State Transfer (REST) was introduced by R.Fielding to describe an **architectural style** of networked software systems
- REST prescribes how a well-designed Web application behaves: a net of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for use."

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Why is it called "Representation State Transfer"?



The Client references a Web resource using a URL. A representation of the resource is returned (in this case as an HTML document).
The representation (e.g., Boeing747.html) places the client application in a state.
The result of the client traversing a hyperlink 747.html is another resource is accessed.
The new representation places the client application into yet another state.
Thus, the client application changes (transfers)
state with each resource representation --> Representation State Transfer!

Motivation for REST

"The motivation for developing REST was to create
an architectural model for how the Web should work,
such that it could serve as a framework for the Web protocol standards.

REST has been applied to describe the desired Web architecture,
help identify existing problems, compare alternative solutions,
and ensure that protocol extensions
would not violate the core constraints that make the Web successful."

- Roy Fielding

REST with HTTP examples

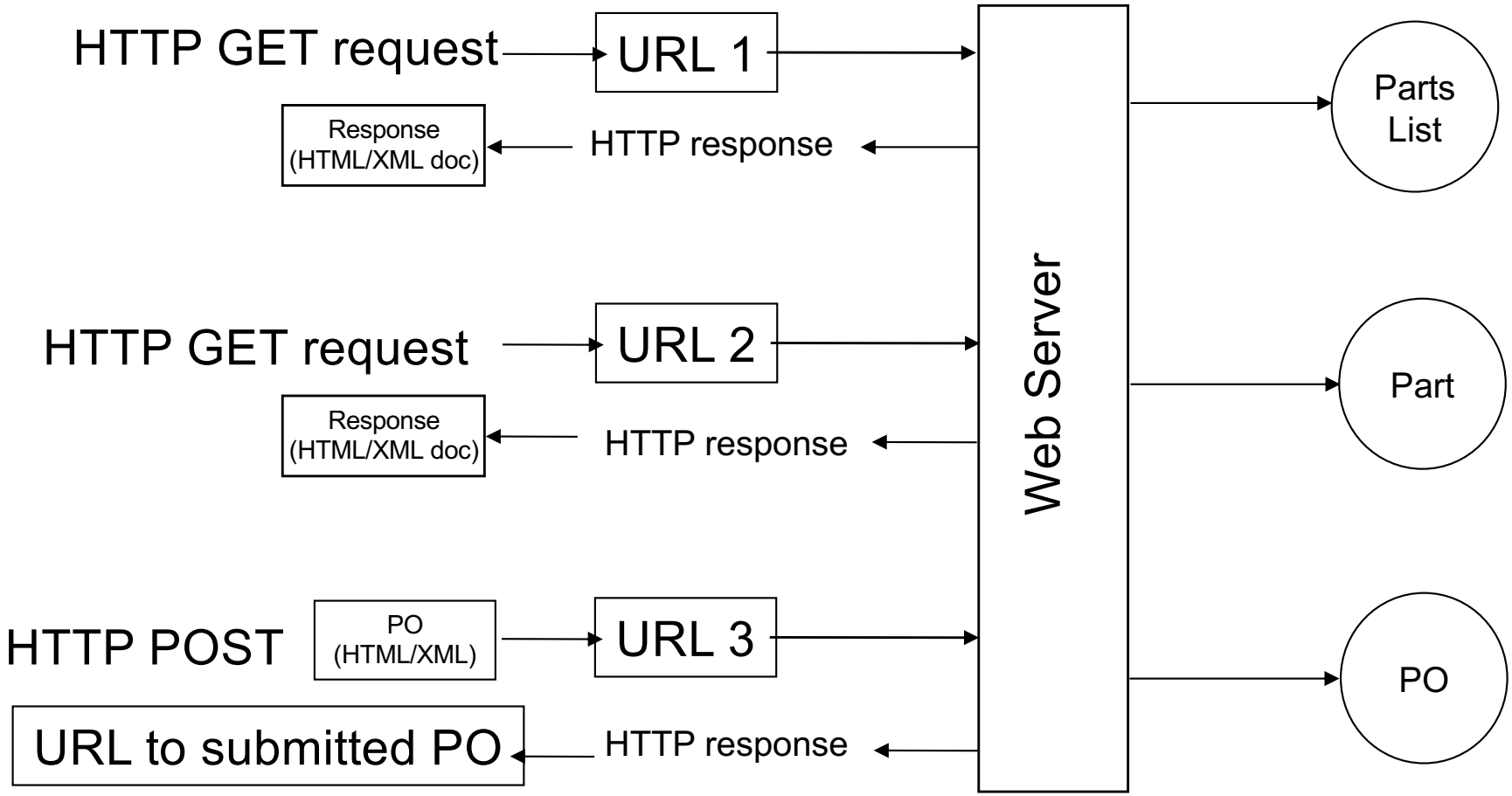
HTTP GET	HTTP PUT	HTTP POST	HTTP DELETE
Collection URI, such as `http://example.com/customers/257/orders`			
List the members of the collection, complete with their member URIs for further navigation	Replace the entire collection with another collection	Create a new entry in the collection. The ID created is usually included as part of the data returned by this operation.	delete the entire collection
HTTP GET	**HTTP PUT**	**HTTP POST**	**HTTP DELETE**
Element URI, such as `http://example.com/resources/7HOU57Y`			
Retrieve a representation of the addressed member of the collection in an appropriate MIME type	Update (or create) the addressed member of the collection	Treats the addressed member as a collection in its own right and creates a new subordinate of it.	Delete the addressed member of the collection.

REST vs SOAP: example

- A company deploying 3 Web services to enable its customers to:
  - get a list of parts
  - get detailed information about a particular part
  - submit a Purchase Order (PO)

- the REST solution first, then the SOAP solution

The REST way of Implementing Web Services



Implementing a Web Service using SOAP

- Service: Get detailed information about a particular part
  - The client creates a SOAP document that specifies the procedure desired, along with the part-id parameter.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>

      <p:getPart xmlns:p="http://www.parts-depot.com">
           <part-id>00345</part-id>
      </p:getPart>

   </soap:Body>
</soap:Envelope>
```

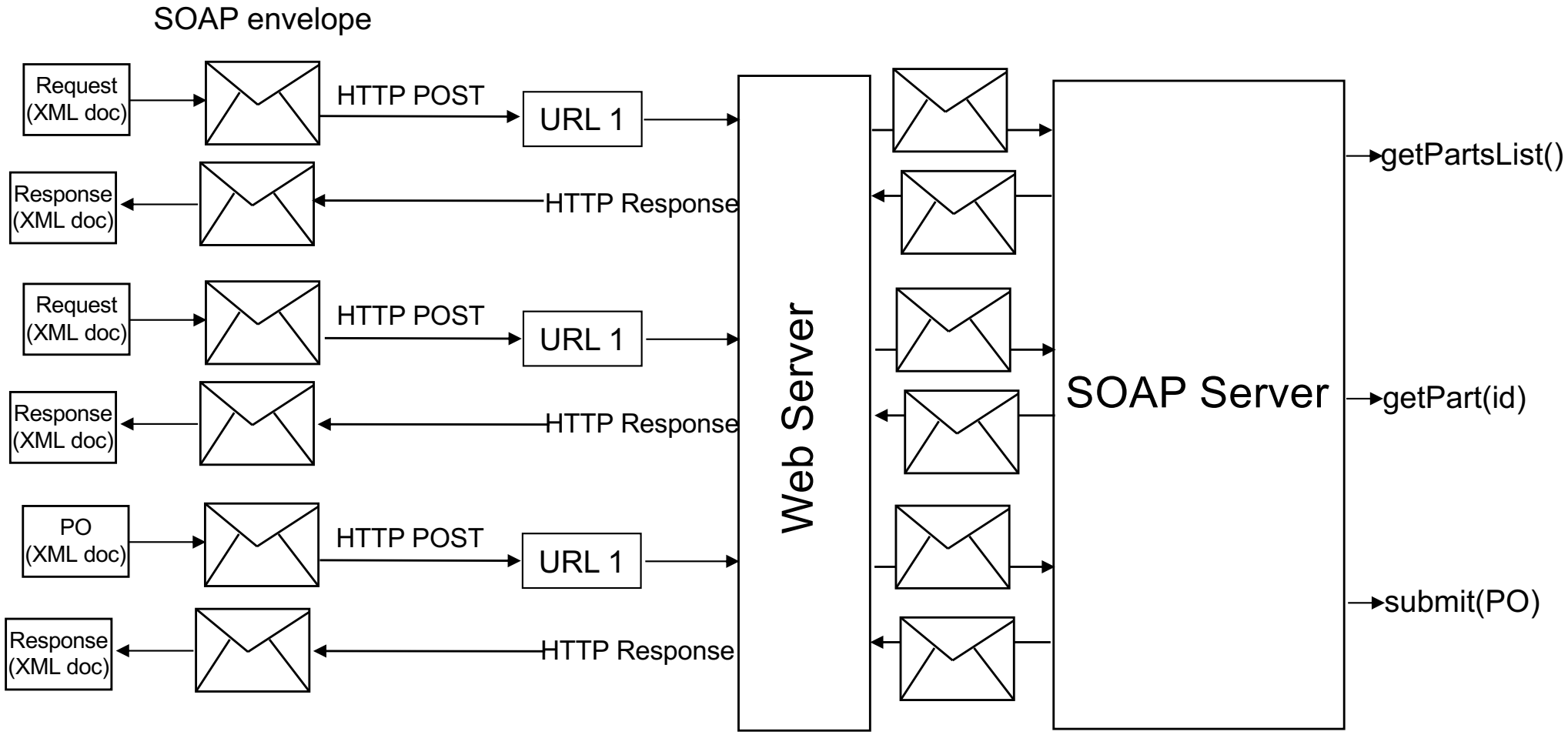
the client will HTTP POST this document to the SOAP server at:

`http://www.parts-depot.com/soap/servlet/messagerouter`

Note that this is the same URL as was used when requesting the parts list.

The SOAP server peeks into this document to determine what procedure to invoke.

Implementing the Web Services using SOAP



Note the use of the same URL (URL 1) for all transactions.
The SOAP Server parses the SOAP message to determine which method to invoke.
All SOAP messages are sent using an HTTP POST.

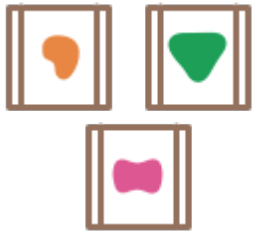
Microservices

- *"Microservice Architecture" describes a particular way of designing software applications as suites of independently deployable services.*
- *While there is no precise definition of this architectural style, there are certain common characteristics around organization, business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data*

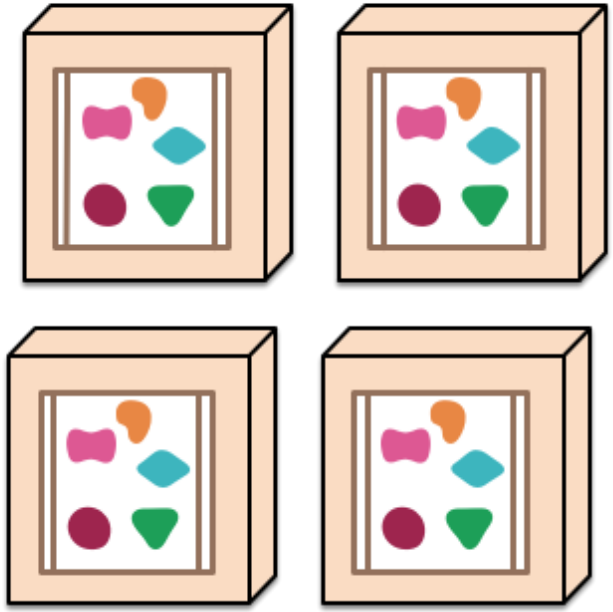
http://martinfowler.com/articles/microservices.html

A monolithic application puts all its functionality into a single process...

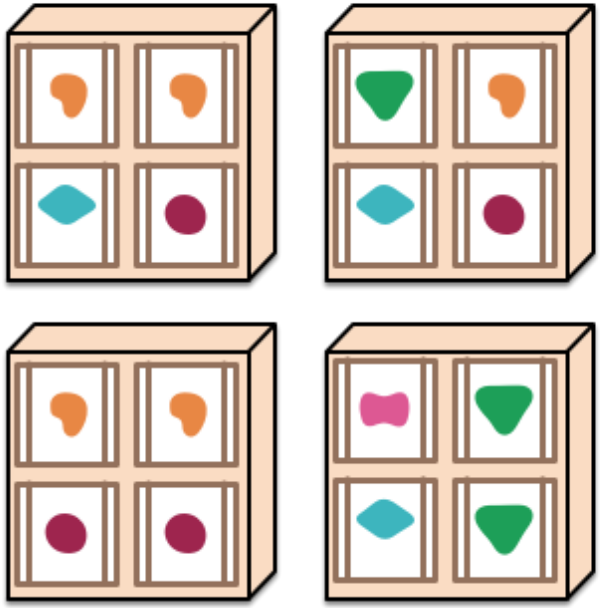
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers

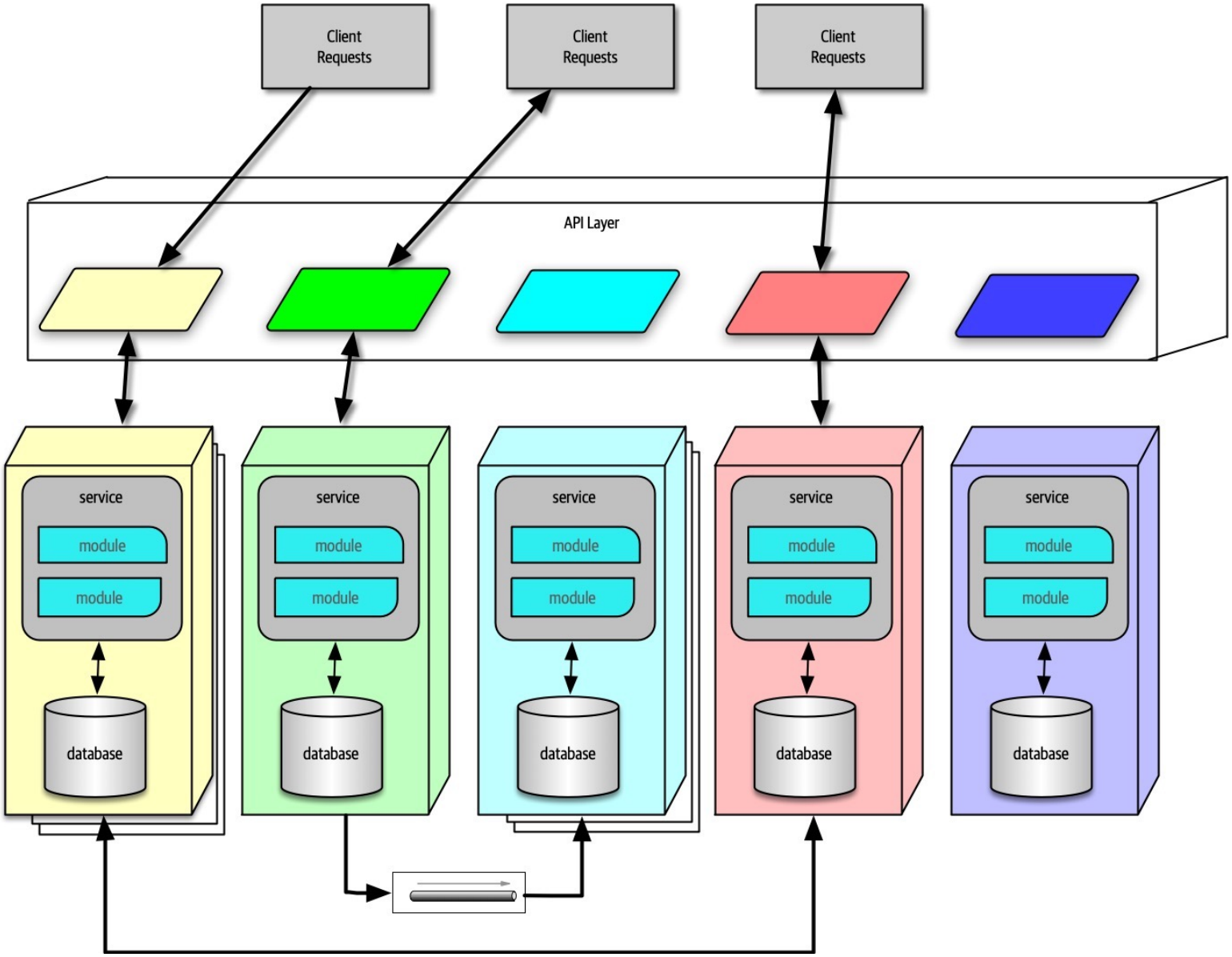


... and scales by distributing these services across servers, replicating as needed.

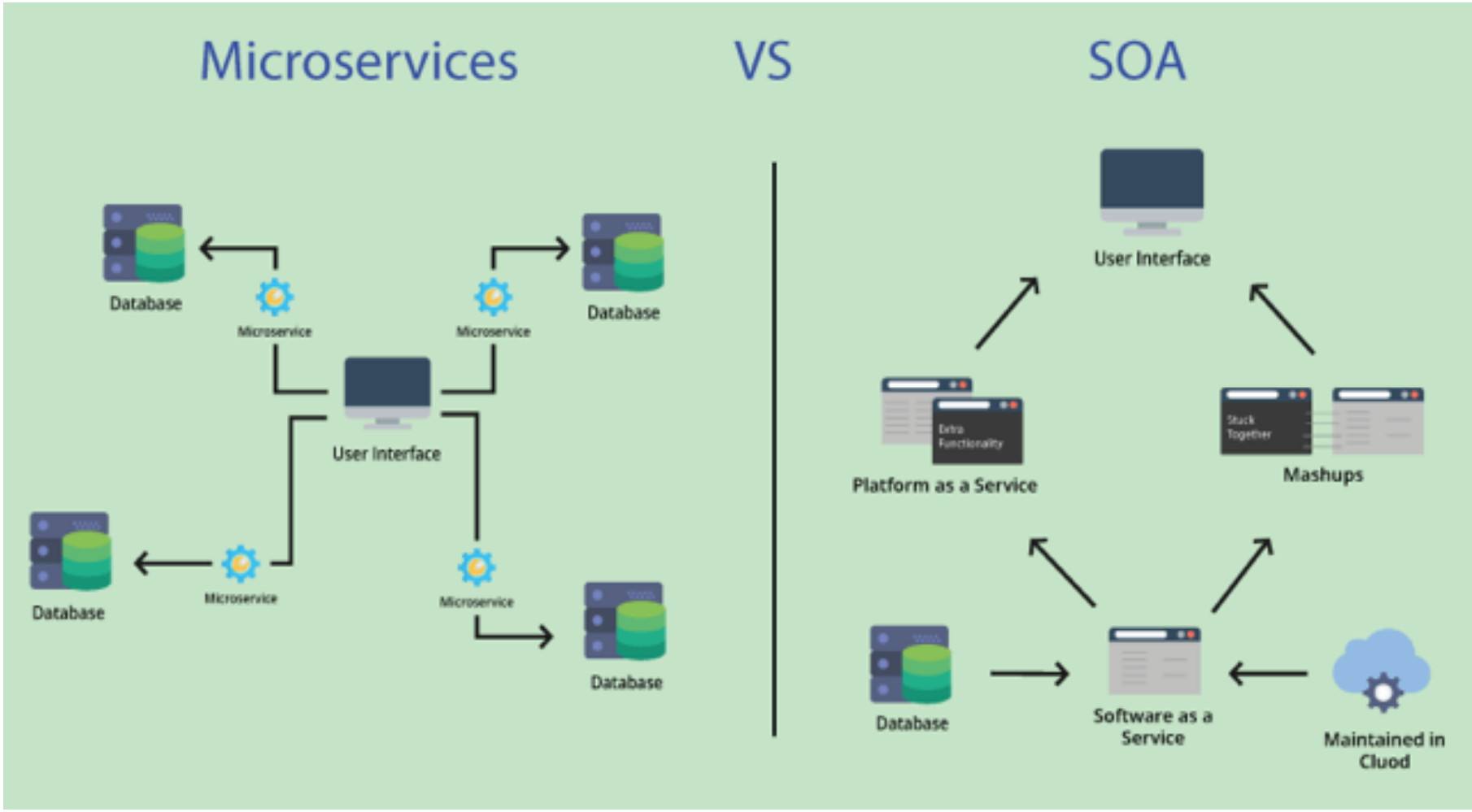


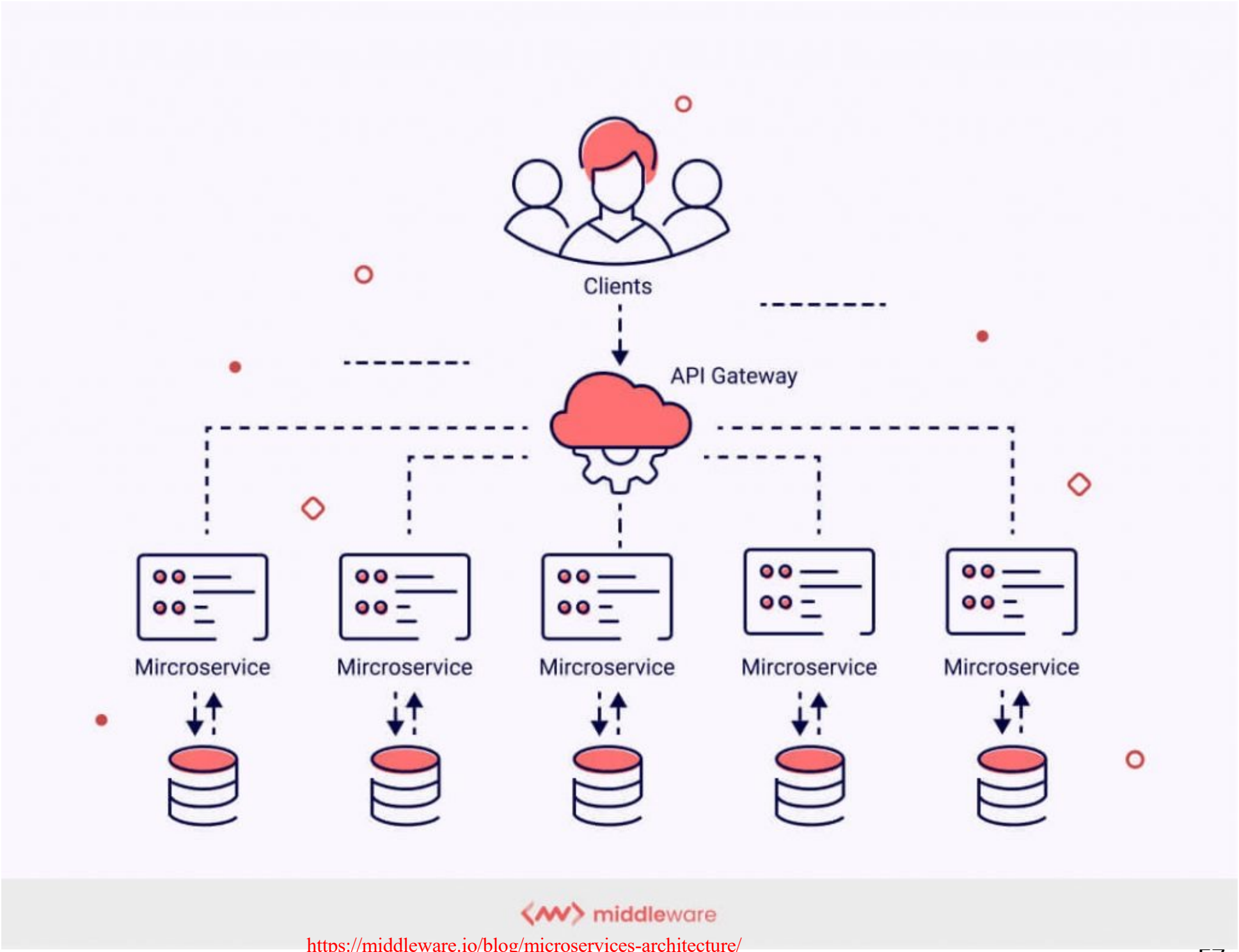
https://martinfowler.com/articles/microservices.html

Microservice architecture



Microservices vs SOA

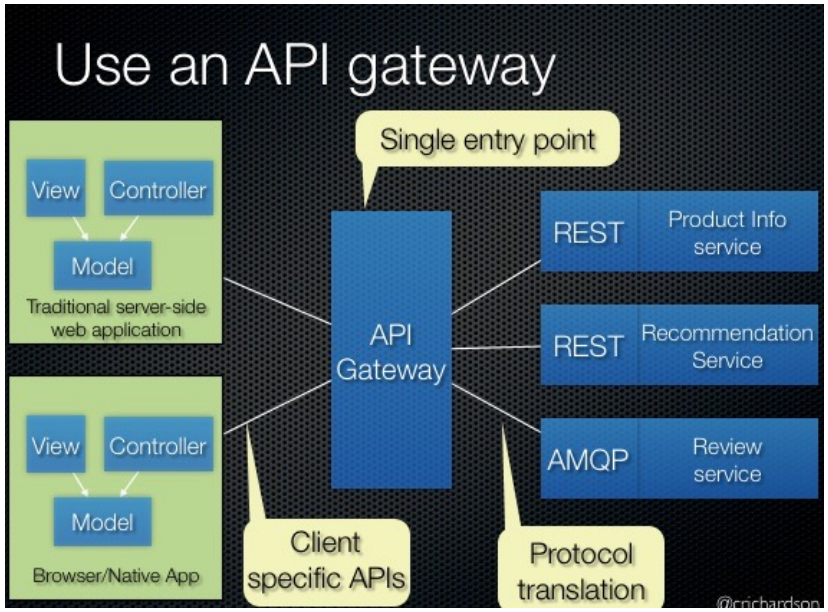




https://middleware.io/blog/microservices-architecture/

API gateway

How do the clients of a microservices-based application access the individual services?
an API gateway that is the single entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.



https://microservices.io/patterns/apigateway.html

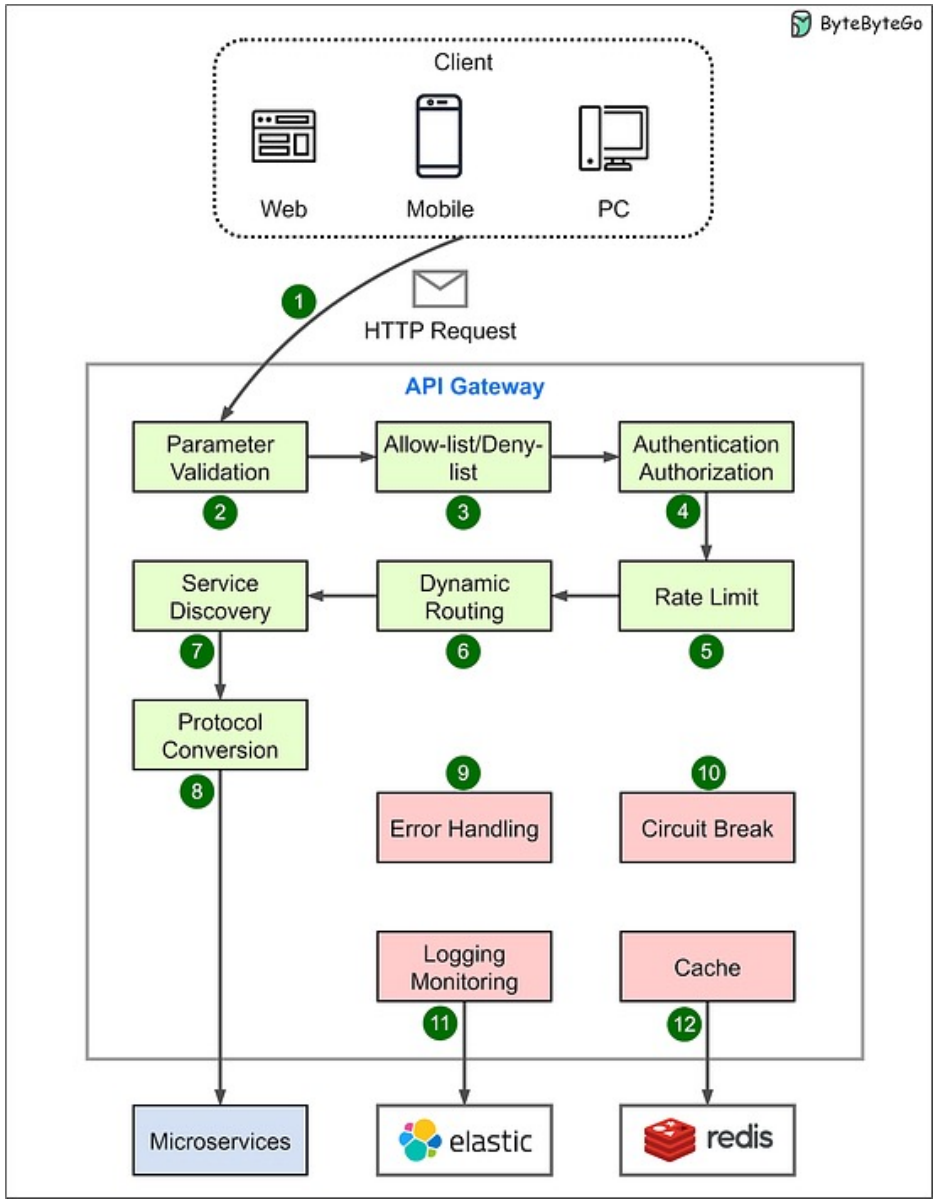
API gateway

In a microservices architecture, an API gateway acts as a single entry point for client requests.

The API gateway is responsible for request routing, composition, and protocol translation.

It also provides additional features like authentication, authorization, caching, and rate limiting

The API gateway is different from a load balancer. While both handle network traffic, the API gateway operates at the application layer, mainly handling HTTP requests; the load balancer mostly operates at the transport layer, dealing with TCP/UDP protocols



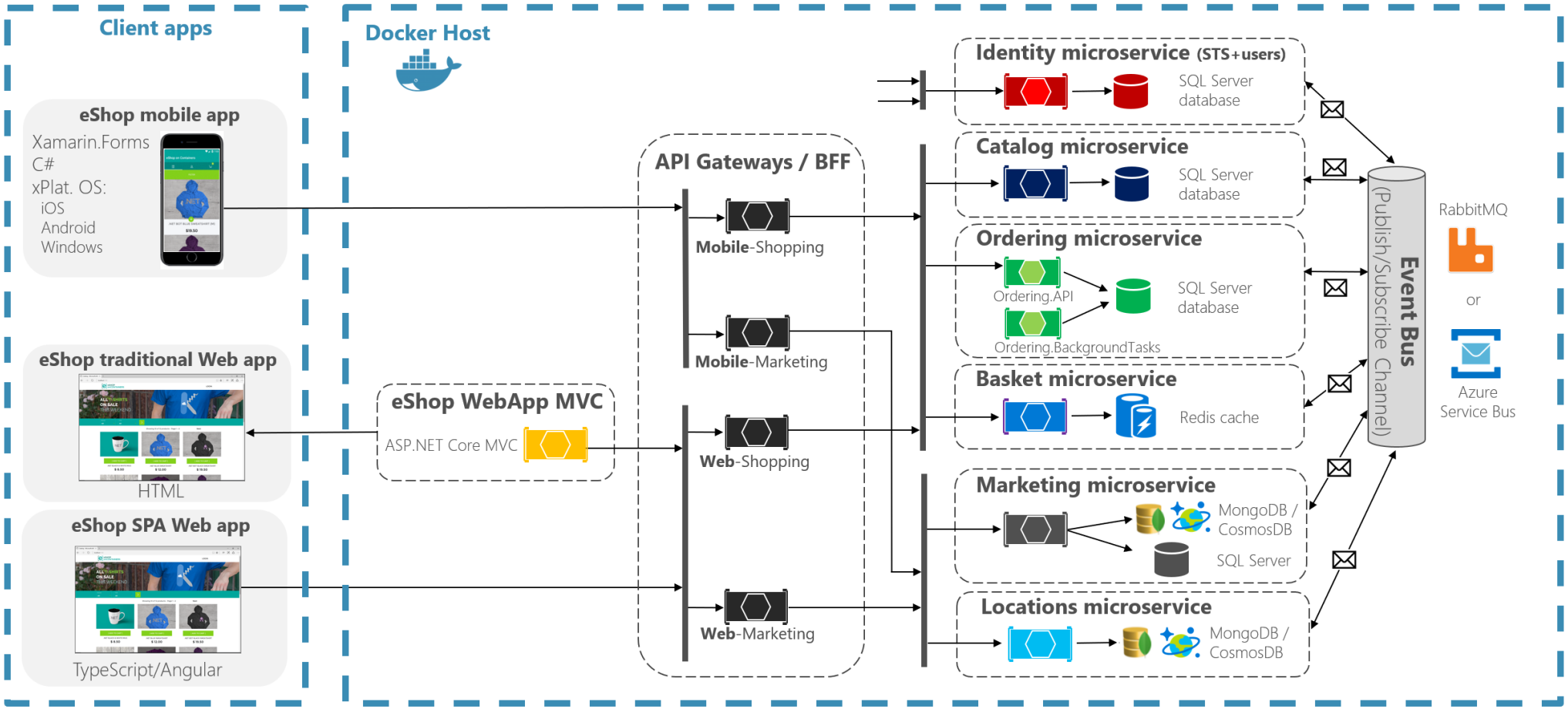
Example: microservices in Instagram

- User: Register account, close account
- Authentication: Generate token from username/password
- Photo: Store photos, resize photos, remove photos
- Location: Store locations, find location by latitude/longitude
- Tag: Store tags, create tags, remove tags
- Timeline: List photos
- E-mail: Send e-mail
- Search: List photos based on tags

Microservice example: eShop on containers

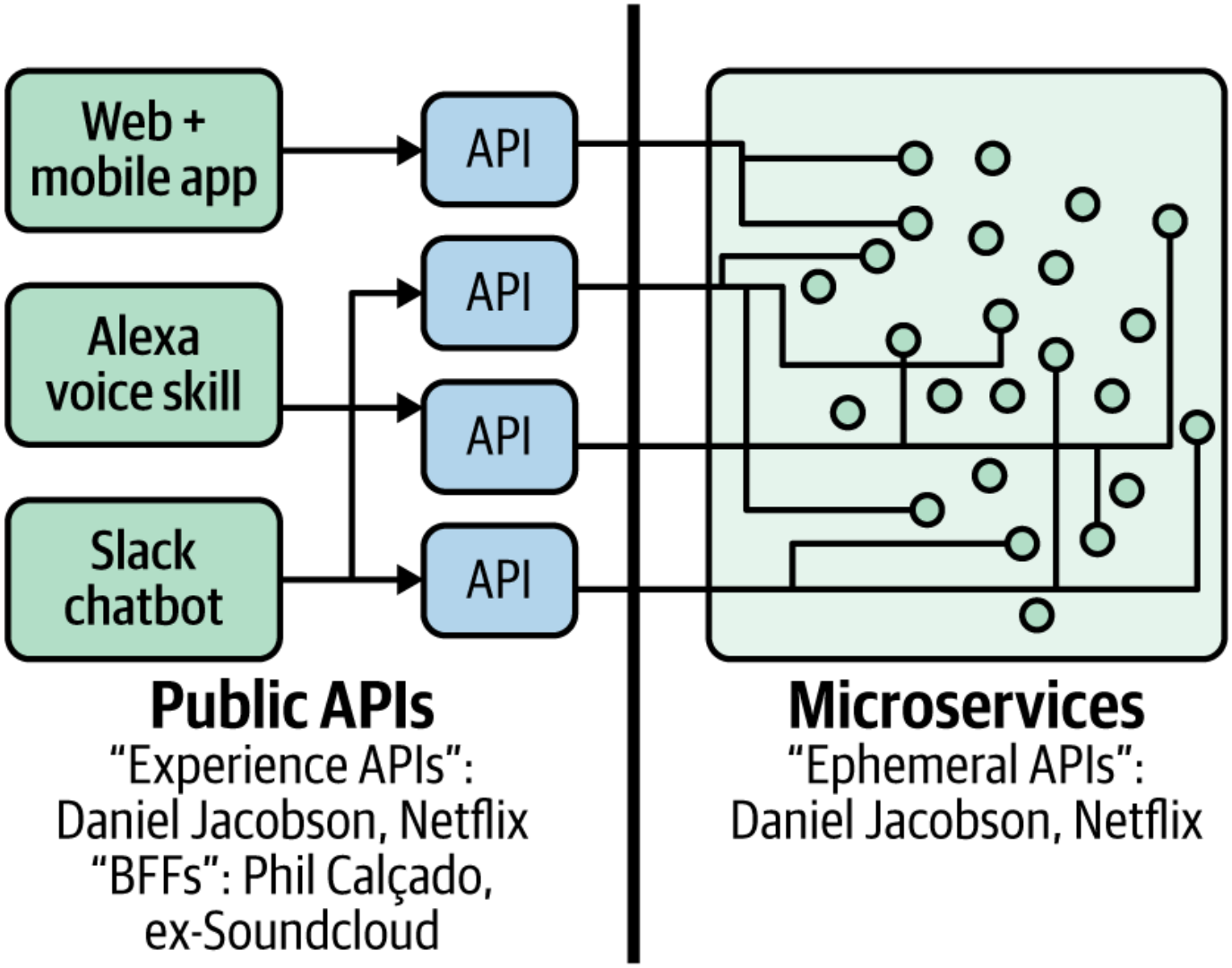
eShopOnContainers reference application

(Development environment architecture)

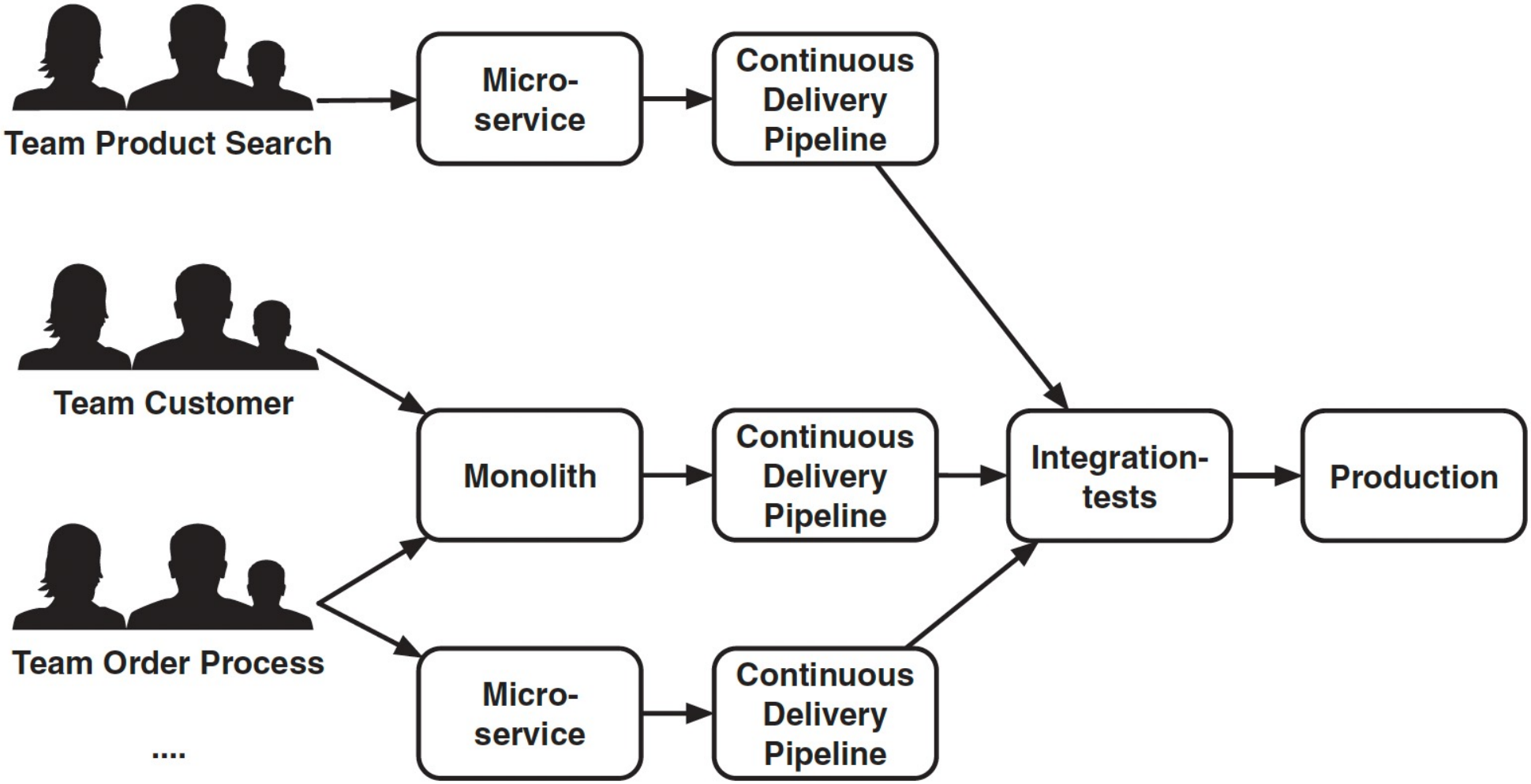


docs.microsoft.com/it-it/dotnet/architecture/microservices/multi-container-microservice-net-applications/microservice-application-design

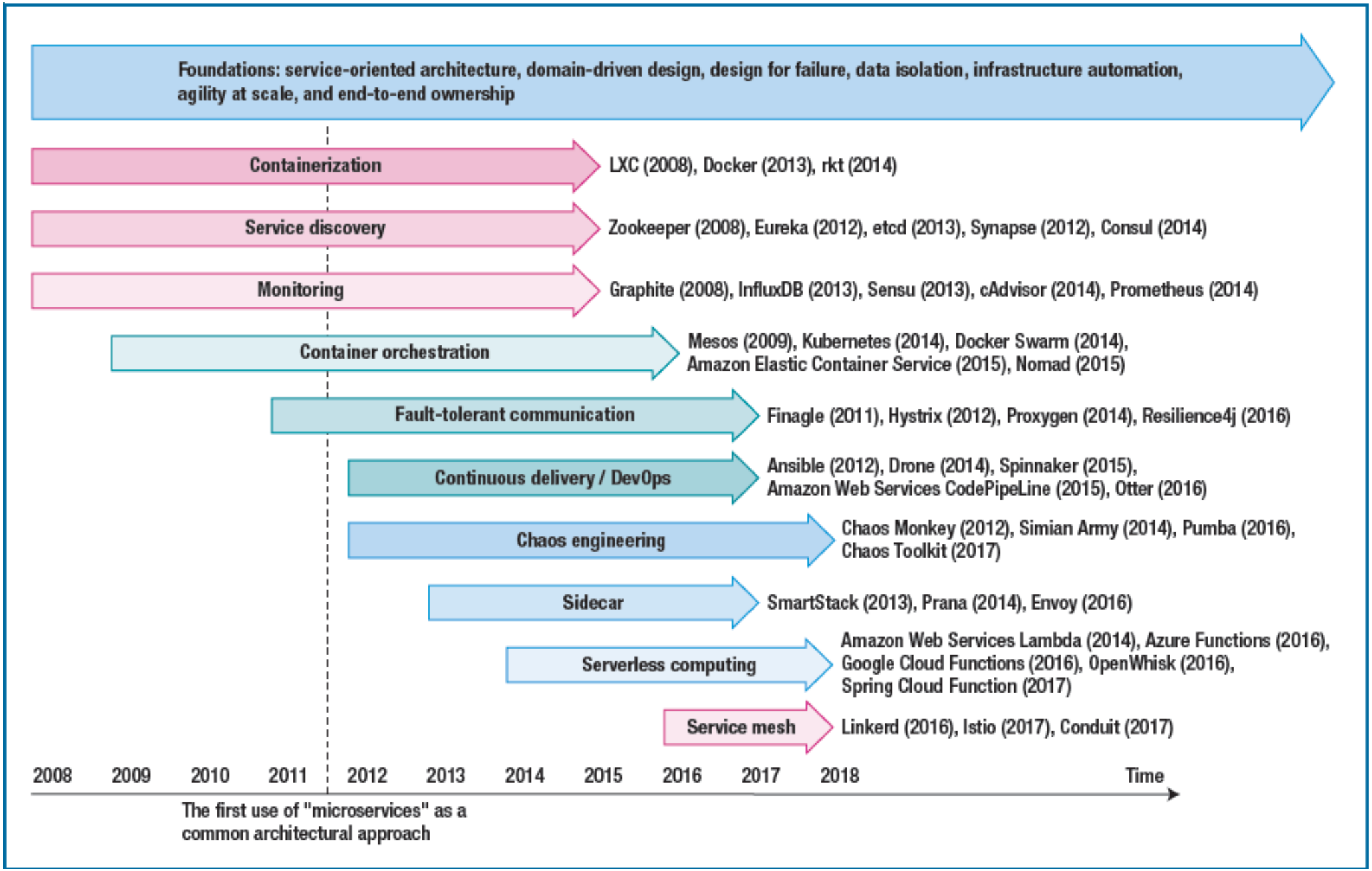
Relationship between API and microservices



Microservices and DevOps



Microservice technologies timeline



Summary: SOA principles

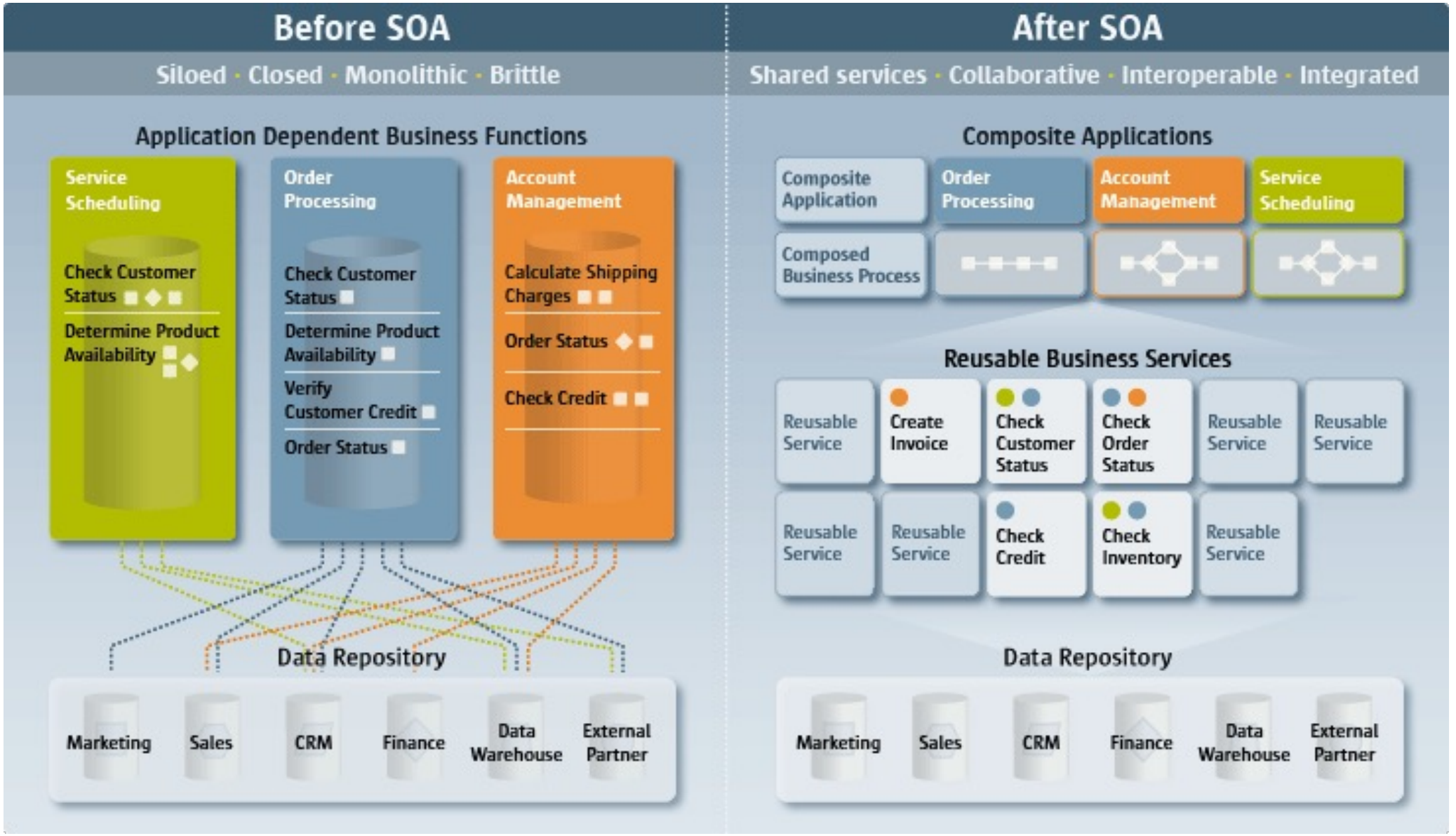
- **Services are autonomous**
- **Services are distributable**
- **Services are loosely coupled**
- **Services share schema and contract, not class**
- **Compatibility is based on policy**

Common examples of service-oriented applications include sharing information, handling multistep processes such as reservation systems and online stores, exposing specific data or services over an extranet, and creating mashups that combine information from multiple sources

SOA benefits: summary

- **Domain alignment**. Reuse of services with standard interfaces increases business and technology opportunities and reduces costs
- **Abstraction**. Services are autonomous and accessed through a formal contract, which provides loose coupling and abstraction
- **Discoverability**. Services expose descriptions that allow other services to locate them and automatically determine the interface
- **Interoperability**. Because the protocols and data formats are based on industry standards, the provider and consumer of the service can be built and deployed on different platforms
- **Rationalization**. Services can be granular in order to provide specific functionality, rather than duplicating the functionality in number of applications, which removes duplication

Before and after SOA



https://herbertograca.com/2017/11/09/service-oriented-architecture-soa/

API architectural styles

REST
Proposed in 2000, REST is the most used style. It is often used between front-end clients and back-end services. It is compliant with 6 architectural constraints. The payload format can be JSON, XML, HTML, or plain text.

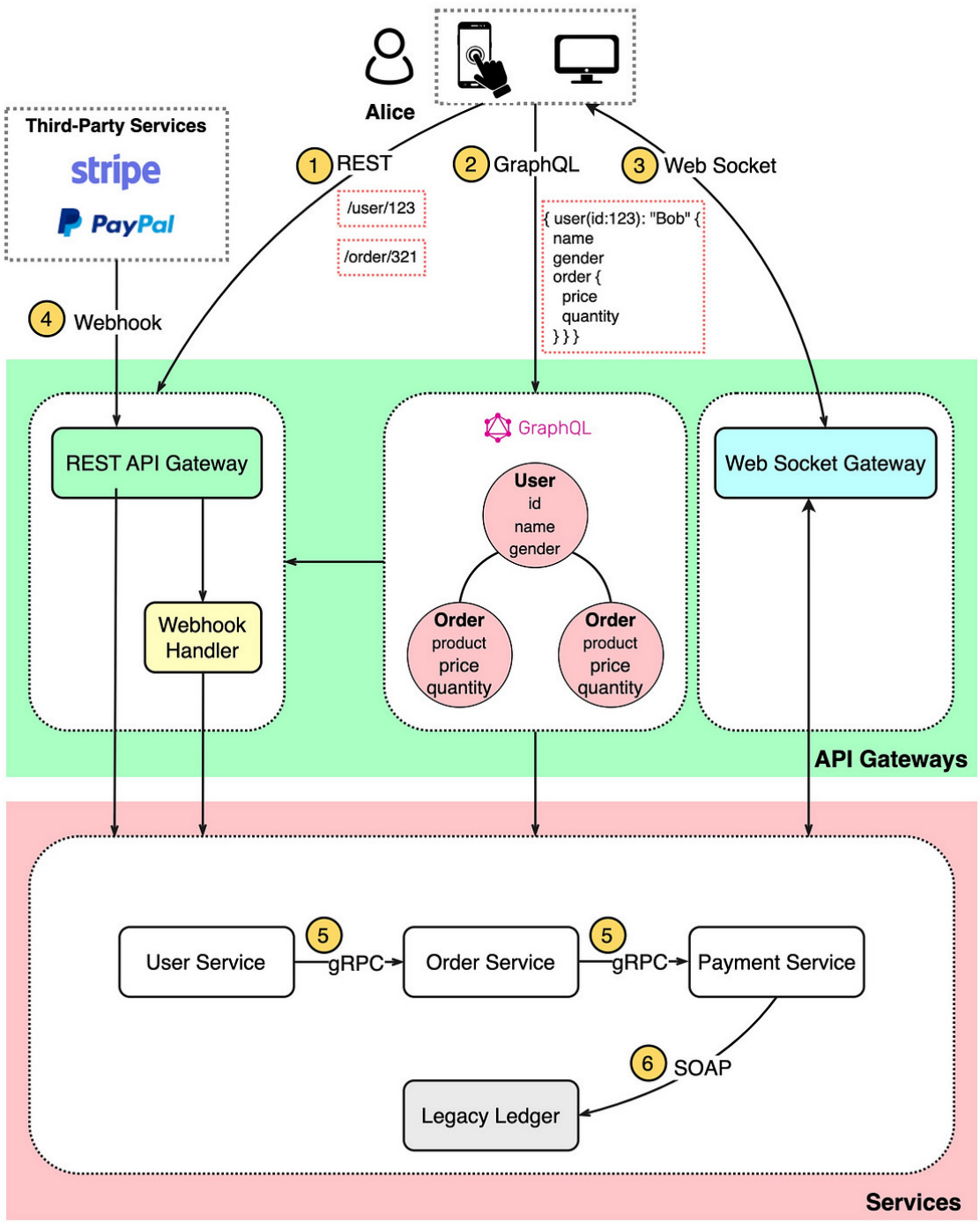
GraphQL
GraphQL was proposed in 2015 by Meta. It provides a schema and type system, suitable for complex systems where the relationships between entities are graph-like. For example, in the diagram below, GraphQL can retrieve user and order information in one call, while in REST this needs multiple calls. GraphQL is not a replacement for REST. It can be built upon existing REST services.

Web socket
Web socket is a protocol that provides full-duplex communications over TCP. The clients establish web sockets to receive real-time updates from the back-end services. Unlike REST, which always "pulls" data, web socket enables data to be "pushed".

Webhook
Webhooks are usually used by third-party asynchronous API calls. In the diagram below, for example, we use Stripe or Paypal for payment channels and register a webhook for payment results. When a third-party payment service is done, it notifies the payment service if the payment is successful or failed. Webhook calls are usually part of the system's state machine.

gRPC
Released in 2016, gRPC is used for communications among microservices. gRPC library handles encoding/decoding and data transmission.

SOAP
SOAP stands for Simple Object Access Protocol. Its payload is XML only, suitable for communications between internal systems.



API styles

There are several API architectural styles, some of the main ones include:

- RESTful API: Representational State Transfer (REST) is a popular architectural style used for building web services. RESTful APIs use HTTP methods like GET, POST, PUT, and DELETE to perform operations on resources identified by URIs.
- GraphQL API: GraphQL is a query language for APIs that was developed by Facebook. It allows clients to request only the data they need and enables the server to send back only the requested data, improving performance and reducing network traffic.
- SOAP API: Simple Object Access Protocol (SOAP) is an XML-based messaging protocol used to exchange structured data between applications over the internet. SOAP APIs can use different transport protocols, such as HTTP and SMTP.
- RPC API: Remote Procedure Call (RPC) is a protocol for calling a function or a procedure on a remote server. It is a client-server model that allows the client to invoke a method on a server without knowing the underlying implementation.
- Webhooks: Webhooks are a way for applications to send real-time notifications to other applications or services. Instead of constantly polling for new data, the webhook approach allows for a more efficient and streamlined process, where the server sends a notification to a specific URL when certain events occur.
- Microservices: Microservices architecture is a style of building software systems that focuses on breaking down an application into small, independent services that communicate with each other through APIs. Each service is responsible for a specific business capability and can be developed, deployed, and scaled independently.

Exercise - Building and Deploying a Simple Microservices System

Setup a dev environment with Docker and a code editor (e.g., Visual Studio Code) installed.

1. Define two microservices with specific functionalities. For example, one service could be a "User Service" responsible for managing user data, and the other could be an "Order Service" responsible for managing orders.
2. Implement Services. Develop the two microservices using your preferred programming language and framework. You can use any technology stack, but ensure that they expose RESTful APIs for communication.
3. Containerize Services. Create a Dockerfile for each microservice. These Dockerfiles should specify the necessary dependencies and configurations for running each service.
4. Deploy Services. Use Docker Compose to define a deployment configuration for both microservices. Ensure that they can communicate with each other.
5. Test the System. Run the Docker Compose configuration to start both microservices. Write a simple client or use a tool like Postman to test the interaction between the two services. For example, the "User Service" might create a user, and the "Order Service" could place an order for that user.
6. Discuss how to monitor and scale these services in a production environment.

Hints: While implementing services, consider using a framework like Spring Boot (Java), Flask (Python), or Express (Node.js); Use Docker Compose for creating the service orchestration; Test your services thoroughly, and ensure they can communicate effectively.

Self test

- Which are the consequences of defining software “a *service*” (instead of “a *good*”)?
- What are the main features of a SOA architectural style?
- What are the main features of the REST architectural style?
- Discuss the difference between REST and SOAP-based architectures.
- How can we see that a site is RESTful?
- Which architectural issues and patterns are typical of systems based on SOA technologies?

References

- Erl, *SOA design patterns*, Prentice Hall, 2008
- RotemGalOz, *SOA patterns*, Manning, 2012
- Richards, *Microservices vs SOA*, O'Reilly, 2015
- Wolff, *Microservices*, AW, 2017
- Daniel & Matera, *Mashups*, Springer, 2014
- Mitra, *Microservices up & running*, O'Reilly, 2020

Relevant sites

- martinfowler.com/articles/microservices.html
- https://aws.amazon.com/microservices/

Questions?