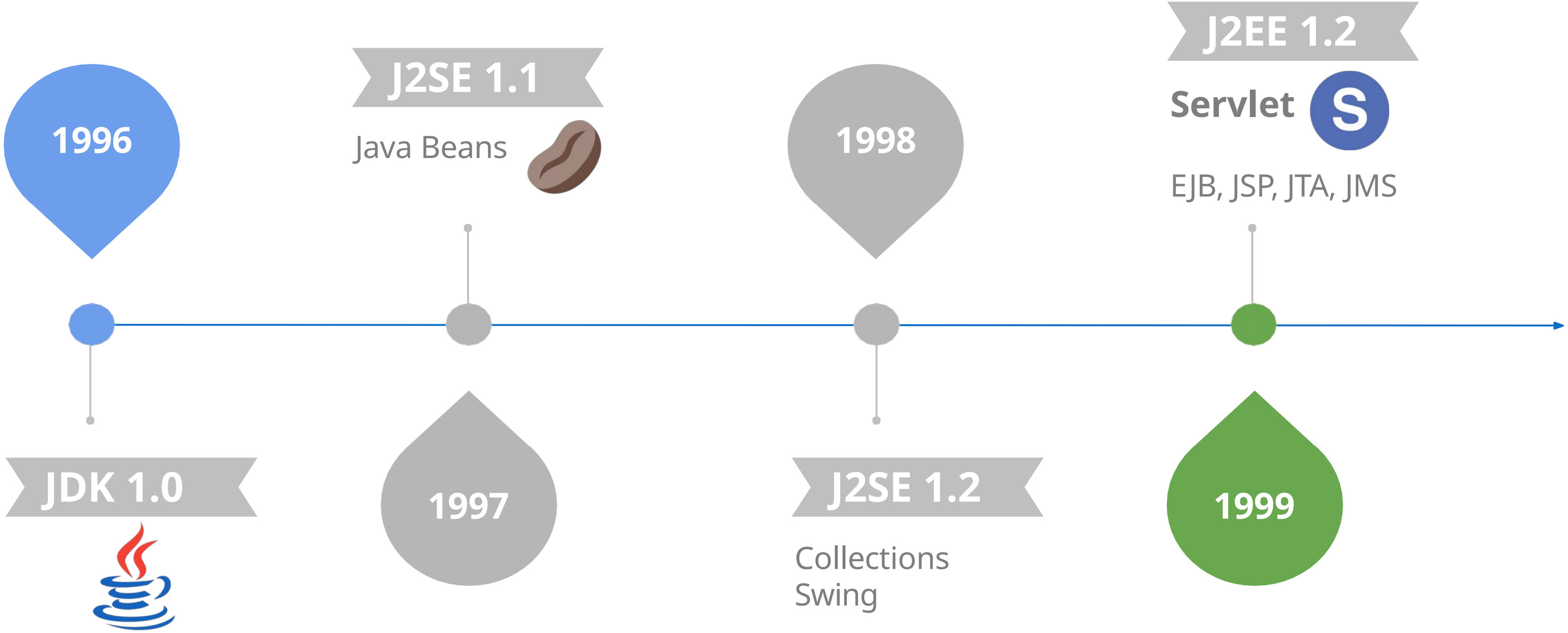
Spring history and web service

Rajeev Gutpa
Java Trainer & Consultant
rgupta.mtech@gmail.com

History



1996

JDK 1.0

1997

J2SE 1.1

Java Beans

1998

J2SE 1.2

Collections
Swing

1999

J2EE 1.2

Servlet S

EJB, JSP, JTA, JMS

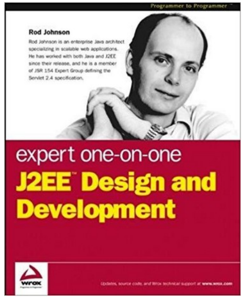
2001

J2EE 1.3

JAXP XML

2002

interface21



Programmer to Programmer™

Rod Johnson

Rod Johnson is an enterprise Java architect specializing in scalable web applications. He has worked with both Java and J2EE since their release, and he is a member of JSR 154 Expert Group defining the Servlet 2.4 specification.

expert one-on-one
J2EE™ Design and Development

Updates, source code, and Wrox technical support at www.wrox.com

2003

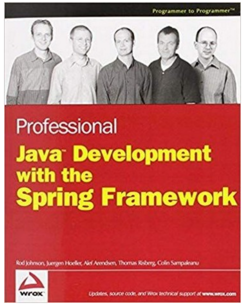
J2EE 1.4

JAX-WS JAX
RPC

SOAP

2004

Spring 1.0

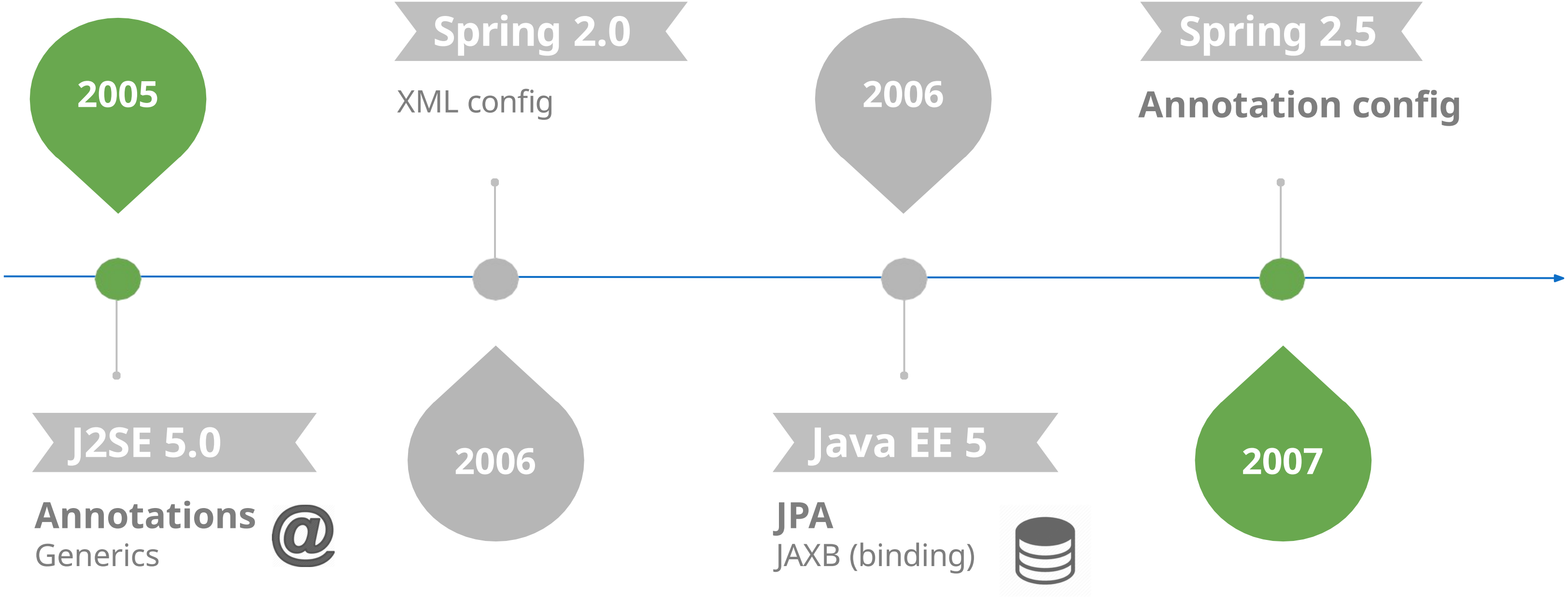


Programmer to Programmer™

Professional
Java™ Development with the Spring Framework

Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu

Updates, source code, and Wrox technical support at www.wrox.com



2005

J2SE 5.0

Annotations

Generics

2006

Spring 2.0

XML config

2006

Java EE 5

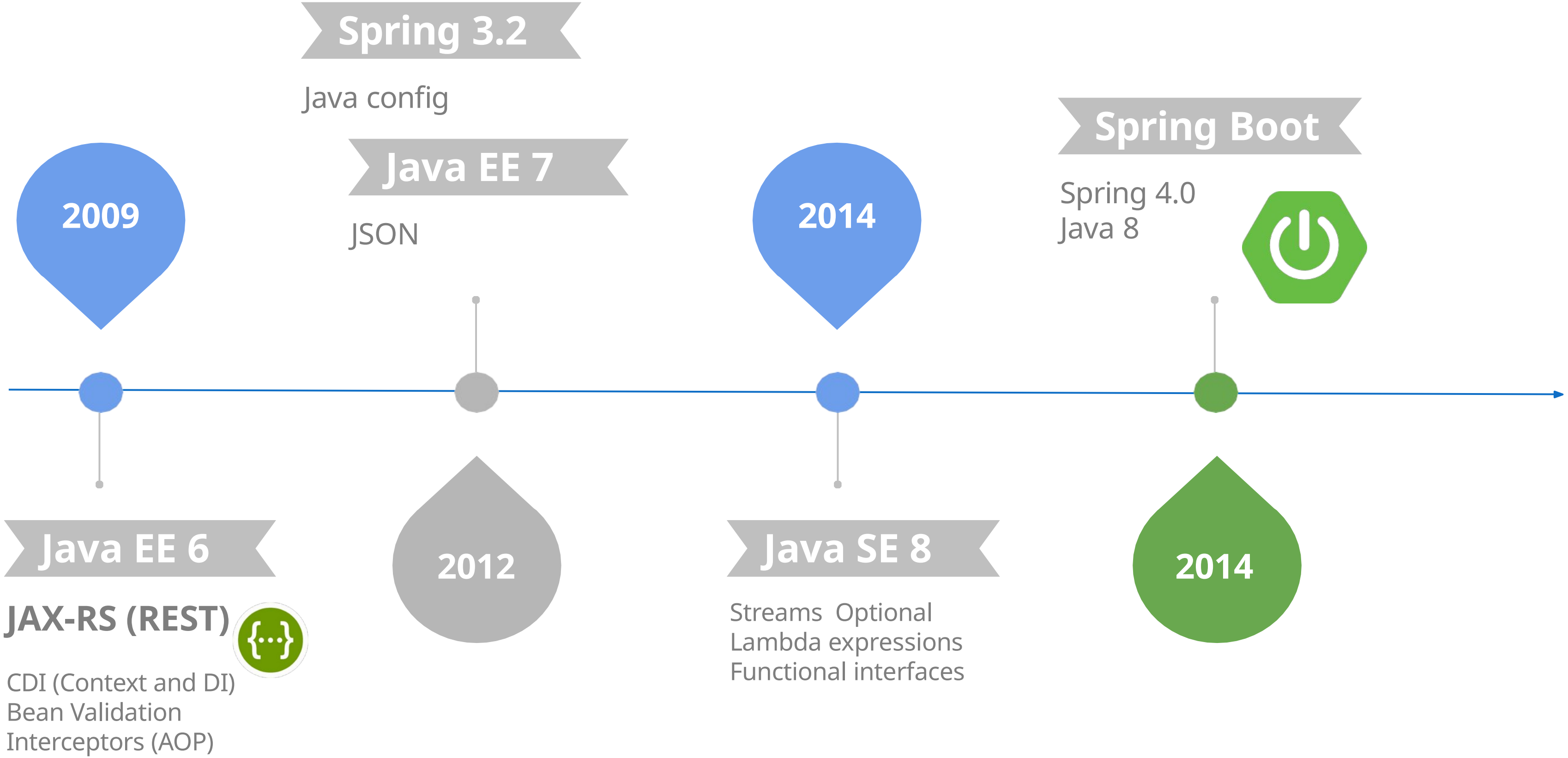
JPA

JAXB (binding)

2007

Spring 2.5

Annotation config



2009

Java EE 6

JAX-RS (REST)

CDI (Context and DI)
Bean Validation
Interceptors (AOP)

2012

Spring 3.2

Java config

Java EE 7

JSON

2014

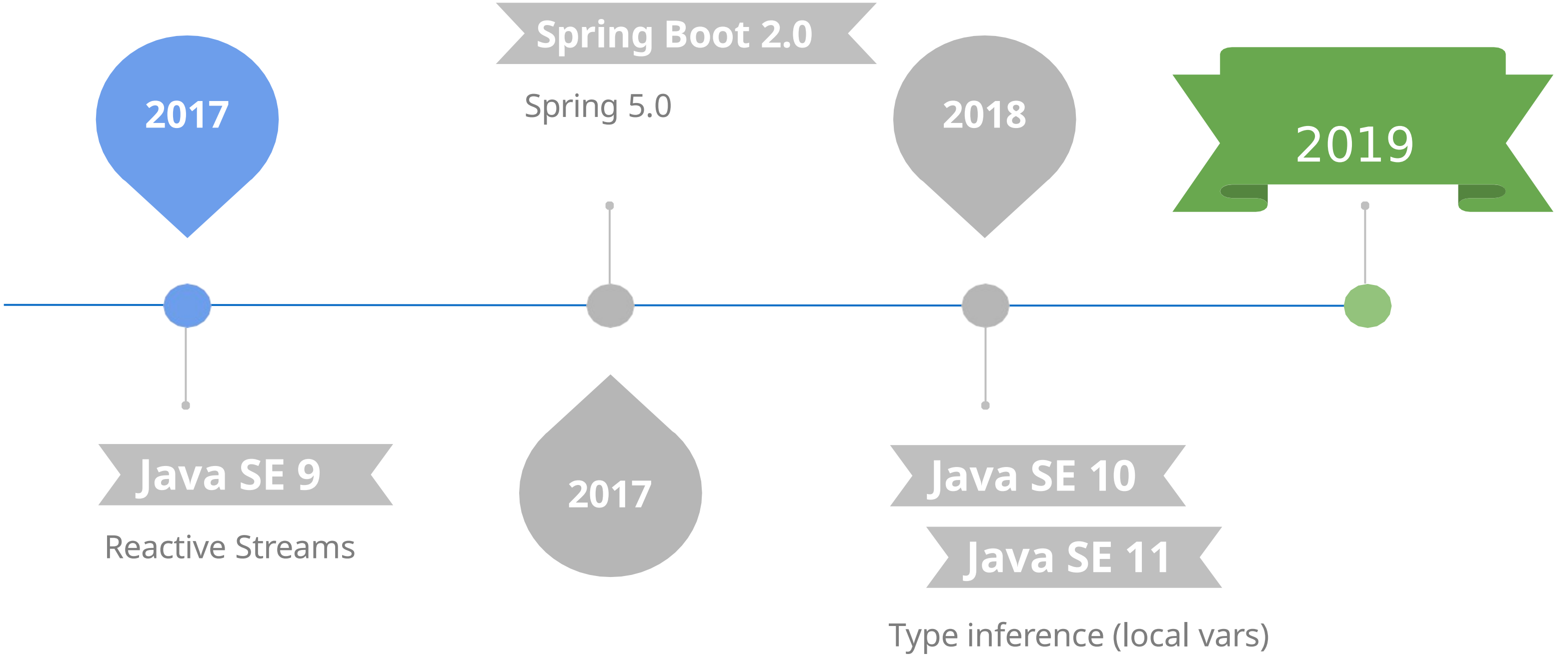
Java SE 8

Streams Optional
Lambda expressions
Functional interfaces

2014

Spring Boot

Spring 4.0
Java 8



2017

Java SE 9

Reactive Streams

2017

Spring Boot 2.0

Spring 5.0

2018

Java SE 10

Java SE 11

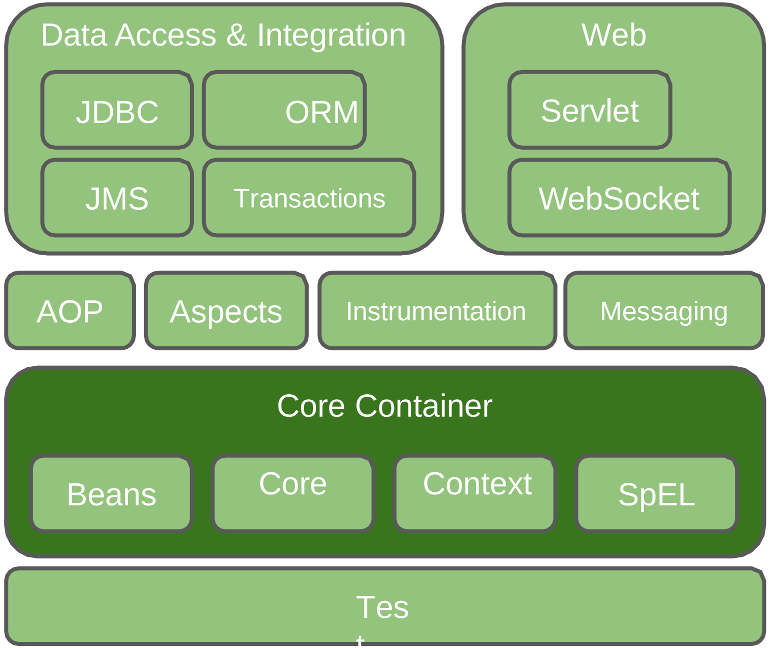
Type inference (local vars)

2019

Spring Framework

- OS application framework
- Modular architecture
    - Containers: application context, bean factory
    - IoC: bean lifecycle management and DI
    - AOP: cross-cutting concerns
    - MVC: web applications and RESTful services
    - Data access: JDBC, ORM, no-SQL
    - Transaction management
    - Security: authentication and authorization
    - Messaging
    - Testing

Core Components



Data Access & Integration

JDBC

ORM

JMS

Transactions

Web

Servlet

WebSocket

AOP

Aspects

Instrumentation

Messaging

Core Container

Beans

Core

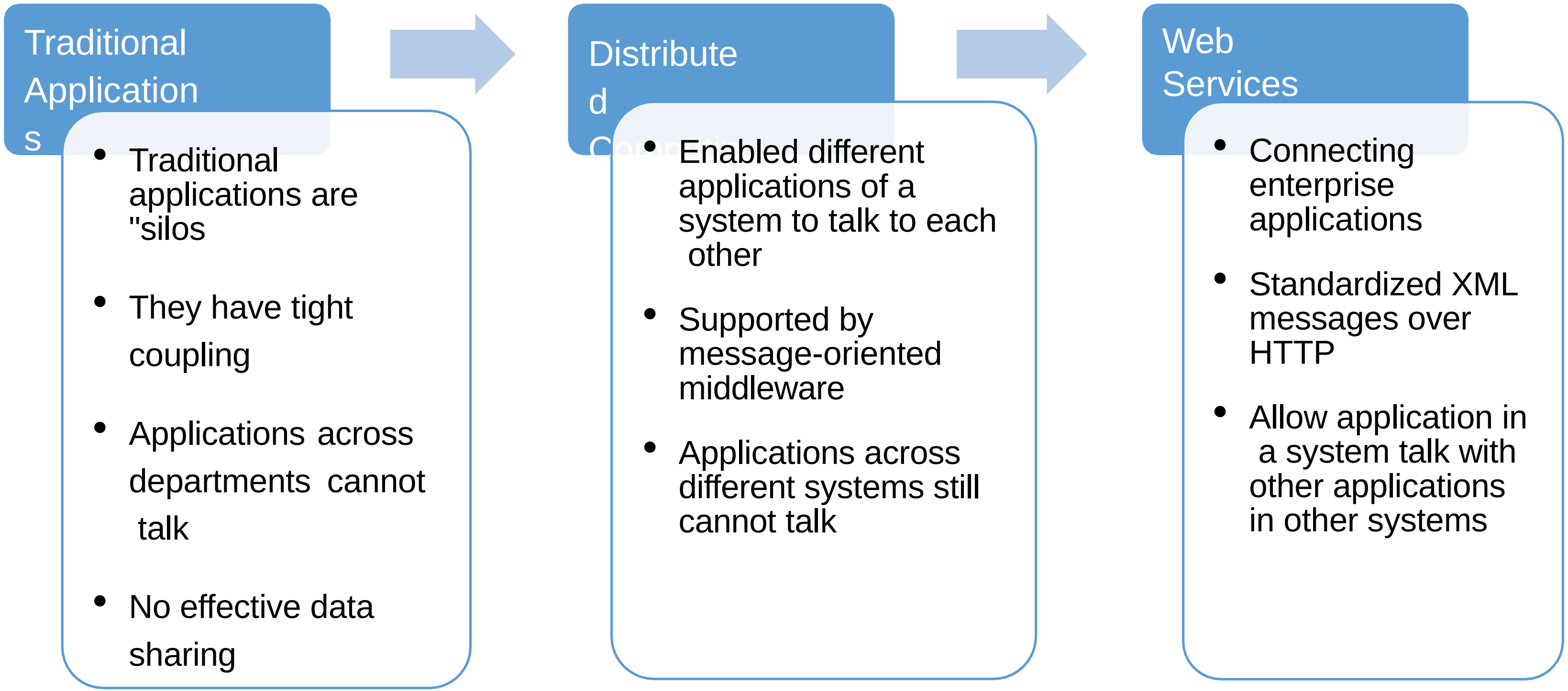
Context

SpEL

Tes
t

- Most of Organizational structures today silo'ed.
- Most enterprise applications also operate as silos.
- Enterprise Data is scattered and "trapped" in the application silos.
- Need a way to move beyond these Application Silos.
- This is the key motivation for **Service Oriented Architecture**.

Traditional Enterprise Applications



Traditional Applications

- Traditional applications are "silos
- They have tight coupling
- Applications across departments cannot talk
- No effective data sharing

Distributed Computin

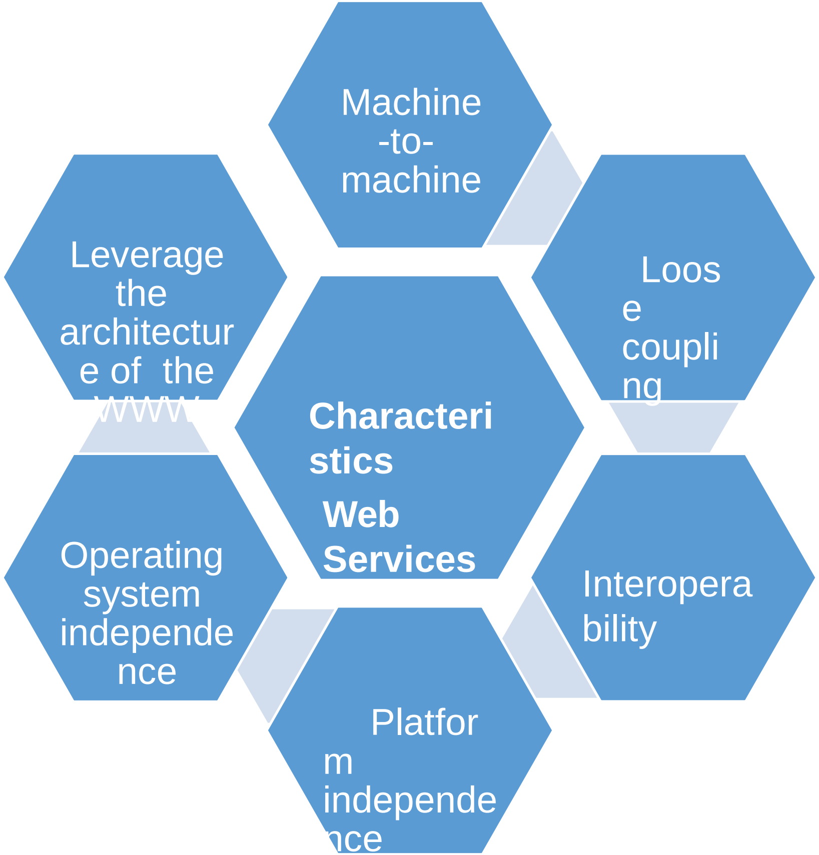
- Enabled different applications of a system to talk to each other
- Supported by message-oriented middleware
- Applications across different systems still cannot talk

Web Services

- Connecting enterprise applications
- Standardized XML messages over HTTP
- Allow application in a system talk with other applications in other systems

- A **software system** designed to support interoperable **machine- to-machine** interaction over a **network**.
    - › Self-contained, modular, distributed, dynamic
  - › Standardized messaging system
    - › Can be described, published, located and/ or invoked over the network.
  - › Language-agnostic
  - › Vendor and transport neutral

Web
Service



Characteristics Web Services

Machine-to-machine

Loose coupling

Interoperability

Platform independence

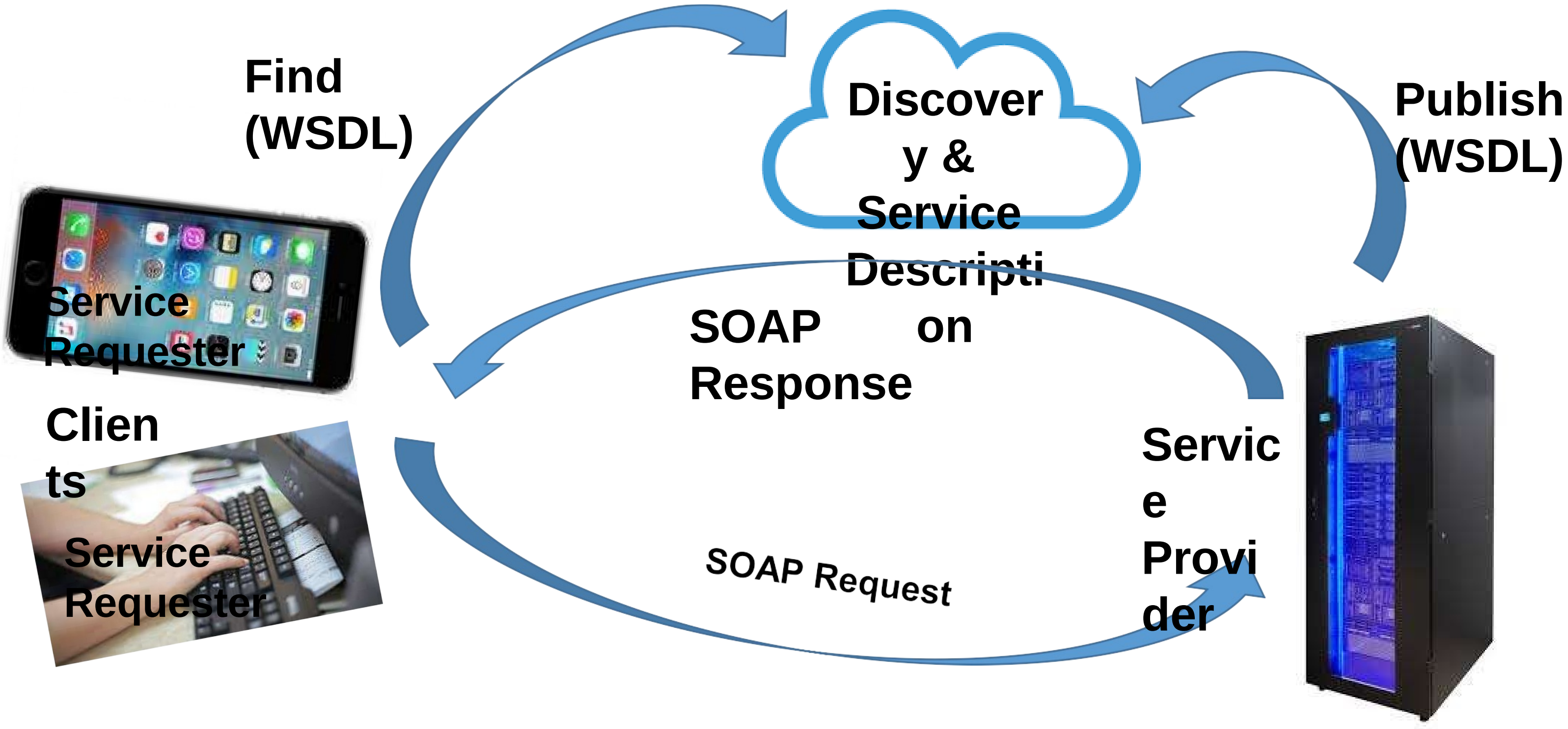
Operating system independence

Leverage the architecture of the WWW

- **Glue application data silos**: enable easier communications within and across organizations
- **Expose the functionality** of existing applications e over the network – without application changes.
- **Loosely Coupled:** Each service exists independently of the other services.
- **Service Reuse:** A function coded once and used over and over
- **Low Cost of Communication:** HTTP over existing internet

SOAP Web Services

- An industry accepted W3C specification for XML distributed computing infrastructure.
- Acronym for **S**imple **O**bject **A**ccess **P**rotocol (acronym not used now)
- Extends HTTP for XML messaging and provides data transport for Web services
- Enables client applications to connect to remote services and invoke remote methods
- Driven by standard specifications
  - Basic: UDDI, WSDL, SOAP, XML & Namespaces
  - Extended: WS-Security, WS-Policy, WS-I (Interoperability) etc.



Find (WSDL)

Discovery & Service Description

Publish (WSDL)

Service Requester

SOAP Response

Clients

Service Requester

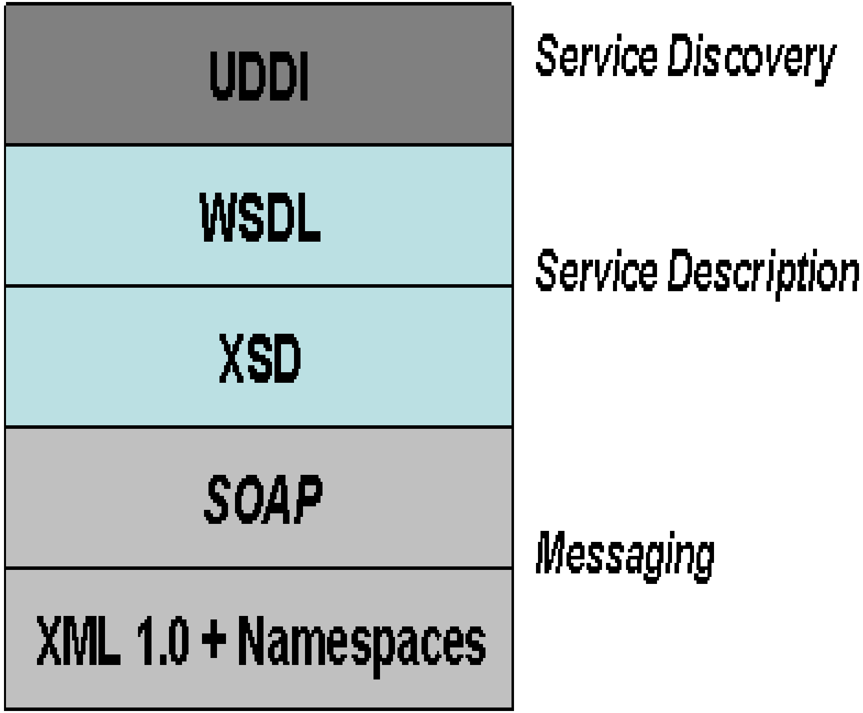
SOAP Request

Service Provider

- **UDDI** (Universal Description, Discovery, and Integration): Platform-independent way of describing and discovering Web services and Web service providers.

- **WSDL**: Defines services as collections of network endpoints.

- **SOAP** Simple and lightweight mechanism for exchanging structured and typed information.
- XML + Namespaces



UDDI

WSDL

XSD

SOAP

XML 1.0 + Namespaces

Service Discovery

Service Description

Messaging

Web Service.

An XML format for describing information needed to invoke and communicate with a

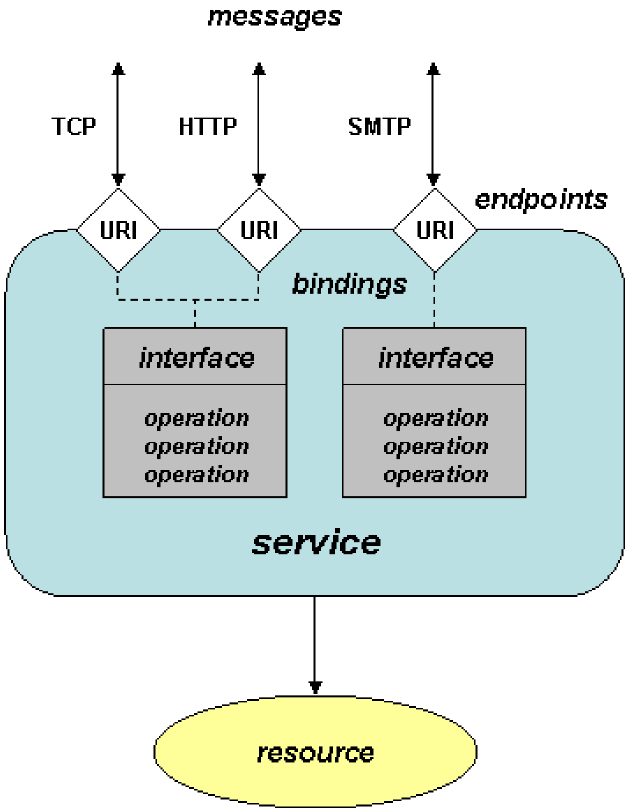
It gives the answers to thequestions›

Who? What?, Where?, Why? How?

- It describes the complete contract for service

› **Functional Description**

› **Nonfunctional Description**:

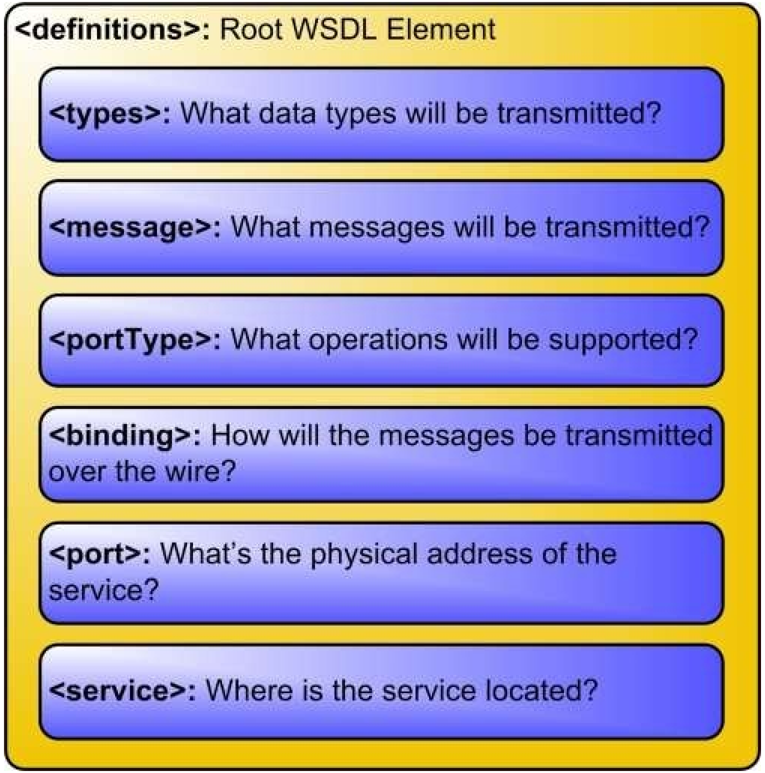


A WSDL Document is a set of definitions with a single root element. Services can be defined using the following XML

elements:

- **Type:** Used to define custom message types

  (Data Type)
- **Message:** Abstraction of request and response messages that my client and service need to communicate (Methods)
- **PortType:** Contains a set of operations (Interfaces) which organize WSDL messages.
- **Binding**: Binds the portType to a specific protocol (typically SOAP over http) (Encoding Scheme)
- **Port**: provides physical address of the Service.
- **Service:** Gives one or more URLs for the service many URLs



<definitions>: Root WSDL Element

<types>: What data types will be transmitted?

<message>: What messages will be transmitted?

<portType>: What operations will be supported?

<binding>: How will the messages be transmitted over the wire?

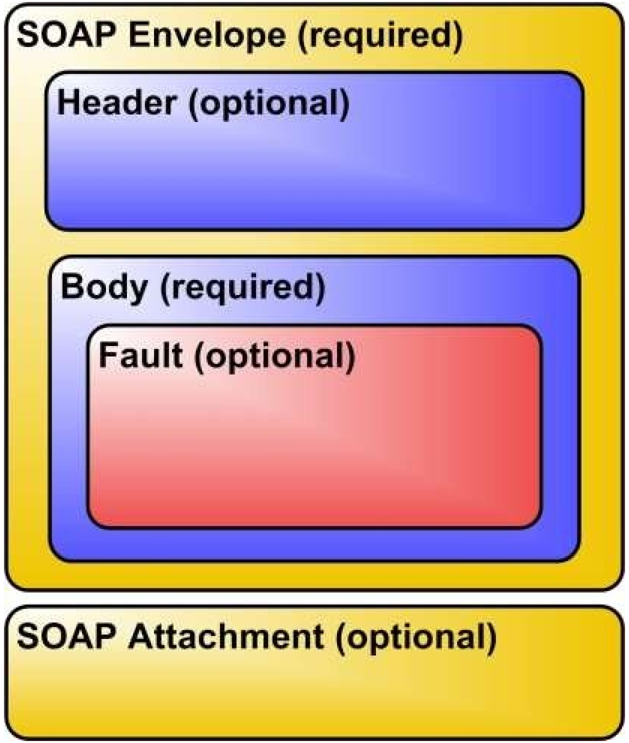
<port>: What's the physical address of the service?

<service>: Where is the service located?

- **XML message format** for exchanging

- Envelope : Root element of a SOAP

- Attachment: MIME encoded for exchanging



- SOAP Pros
  - › Standard protocol for exchanging information in a decentralized and distributed environment.

› Platform independent & Vendor neutral.

› Simple compared to RMI, CORBA, and DCOM etc

› Decouples the encoding and communications protocol.

› Anything that can generate XML can communicate through SOAP.

› Additional Security in addition to HTTP authentication or HTTPS.

› Supported by most languages and tools.

SOAP Cons

› Complex compared to RESTful Services

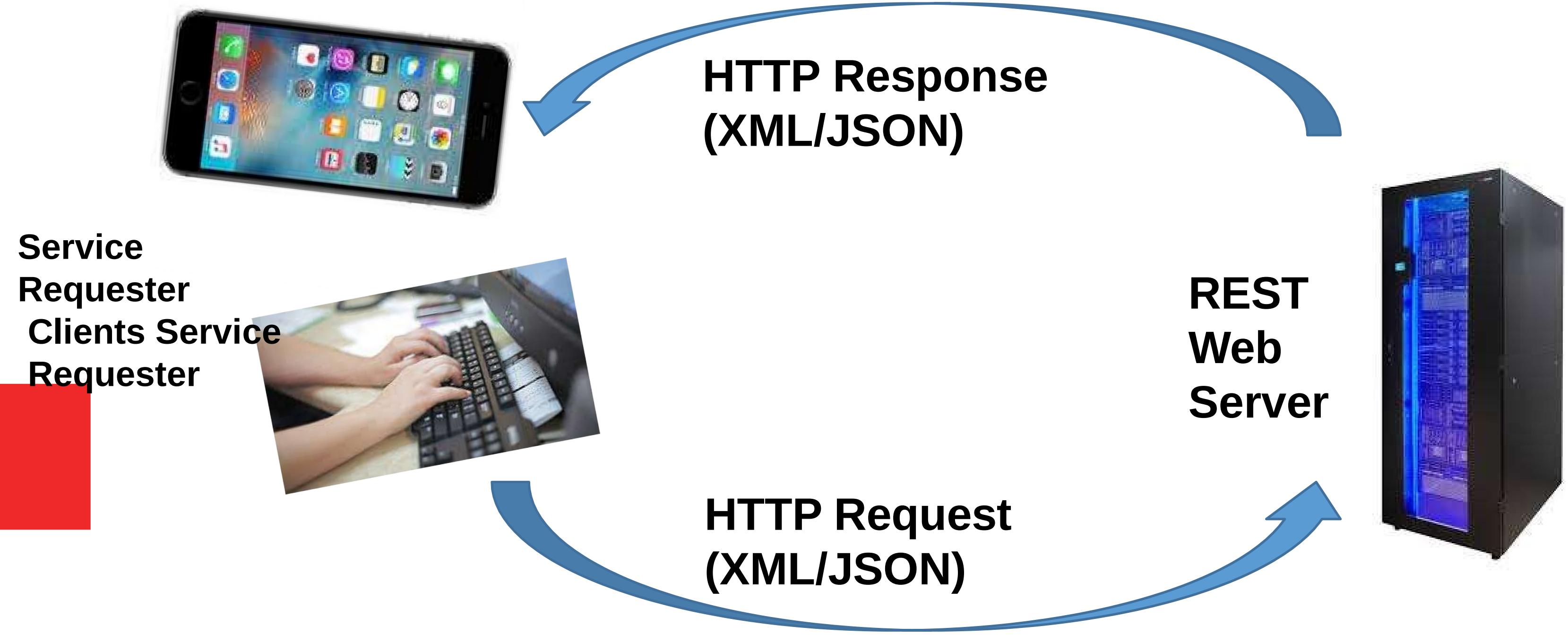
› Higher learning curve

› Being protocol heavy may lead to performance issues

- **Re**presentational **S**tate **T**ransfer
- REST is an architecture all about the Client-Server communication
- Guided by REST constraints (design rules).

- Based on Resource Oriented Architecture
  - A network of web pages where the client progresses through an application by selecting links
  - Requests/responses relate to representations of states of a resource
  - When client traverses link, accesses new resource (i.e., transfers state)

- Uses simple HTTP protocol and service methods:
  - GET: Return data, nothing is changed on server
  - POST: Create, update, or delete data on server
  - PUT: Replace referenced resource(s)
  - DELETE: Delete referenced resource(s)

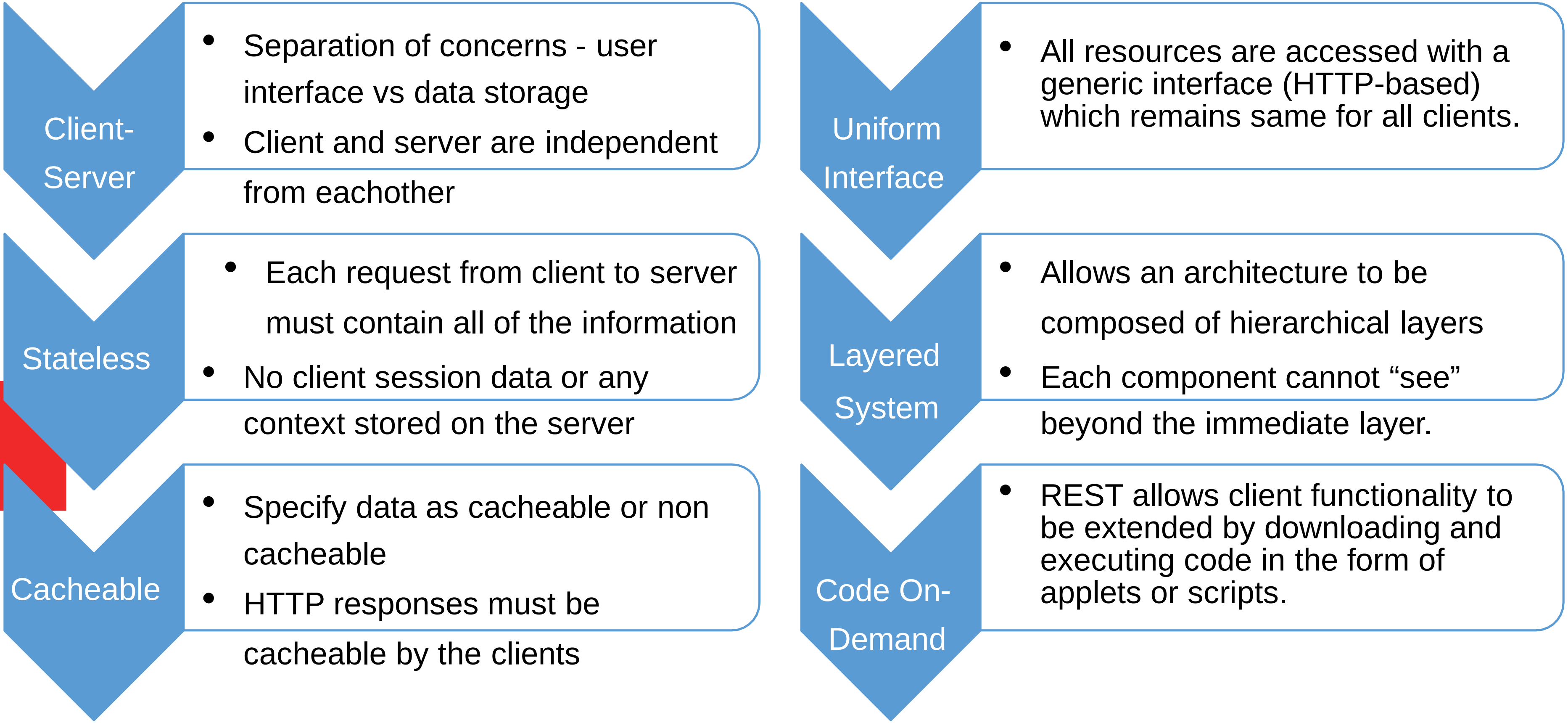


HTTP Response
(XML/JSON)

Service
Requester
Clients Service
Requester

REST
Web
Server

HTTP Request
(XML/JSON)



Client-Server

- Separation of concerns - user interface vs data storage
- Client and server are independent from eachother

Stateless

- Each request from client to server must contain all of the information
- No client session data or any context stored on the server

Cacheable

- Specify data as cacheable or non cacheable
- HTTP responses must be cacheable by the clients

Uniform Interface

- All resources are accessed with a generic interface (HTTP-based) which remains same for all clients.

Layered System

- Allows an architecture to be composed of hierarchical layers
- Each component cannot “see” beyond the immediate layer.

Code On-Demand

- REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts.

HTTP Method	URI	CRUD	Request Stream	Response Stream	Response Code
POST	/customers	Create	Customer without id	customer	201 / 404 / 409
GET	/customers	Read	n/a	Customers collection	200 / 404
GET	/customers/{id}	Read	n/a	Customer	200 / 404
PUT	/customers/{id}	Update	Customer	n/a	200 / 204 / 404
DELETE	/customers/{id}	Delete	n/a	n/a	200 / 404
OPTIONS	/customers/	Available Methods	n/a	Available Methods	200 / 204

Steps for designing RESTful Web Service

- Identifying resources the service will expose over the network.
- Designing the URI Templates – map URIs to resources
- Applying the Uniform HTTP Interface – options available on each resource for different user groups.
- Security Considerations – Authentication and authorization
- Designing the Resource Representations – XML/JSON.
- Supporting alternate Representations – XML or JSON based on filters
- Providing Resource Metadata – Ability to discover resources and options
- Avoiding RPC Tendencies

RESTful Service Implementation Considerations

- Parse the incoming request to
  - › Use URI to identify the resource.
  - › Identify URI variables (and map them to resource variables)
  - › HTTP method used in the request (and whether it’s allowed for the resource).
- › Read the resource representation
- Authenticate and authorize the user.
- Use all of this information to perform the underlying service logic.
- Generate an appropriate HTTP response, including
  - › Proper status code
  - › Description
  - › Outgoing resource representation in the response entity body (if any).

- RESTful Pros
  - Simple interface (URI based)
  - Uses HTTP service methods (GET, POST, ...)
  - Caching can be leveraged for better performance
  - Small learning curve
  - Simple to test (browser compatible)
  - Less reliance on tools
  - No standard
- RESTful Cons
  - Not yet well integrated into IDE’s (but getting better)
  - Security relies on HTTP authentication
  - Less reliance on tools
  - No standard

SOAP or RESTful?

SOAP	RESTful
XML based Messaging Protocol	REST is an architectural style
Uses WSDL for communication between Consumer	Uses XMl or JSON to send or receive data Provider
SOAP is Service Oriented – Invokes services by calling RPC methods	REST is Resource Oriented - uses (generally) URI and methods like (GET, PUT, POST, DELETE) to expose resources
SOAP supports for stateful implementation	REST follows stateless model
Transfer is over HTTP as well as other protocols such as SMTP, FTP, etc	REST is over only HTTP
SOAP is Distributed Computing style implementation	REST is Web Style (Client Server) Implementation
SOAP can be called from JavaScript but difficult to implement.	Easy to call from JavaScript.

- Choosing between SOAP and RESTful – Architectural Decision, not just a matter of simplicity or performance.

When to use SOAP

- Complex applications
- Stateful operations
- Formal Contracts
- Asynchronous processing and invocation
- Distributed applications in peer relationship
- Additional security via WS Security

When to use RESTful

- Relatively Simpler Applications
- Stateless operations
- Limited bandwidth and resources
- Caching situations
- Client Server mode
- Only HTTPS security

Thank you