INTERNET SYSTEMS

WEB SERVICES

TOMASZ PAWLAK, PHD
MARCIN SZUBERT, PHD
POZNAN UNIVERSITY OF TECHNOLOGY, INSTITUTE OF COMPUTING SCIENCE

PRESENTATION OUTLINE

- History & Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

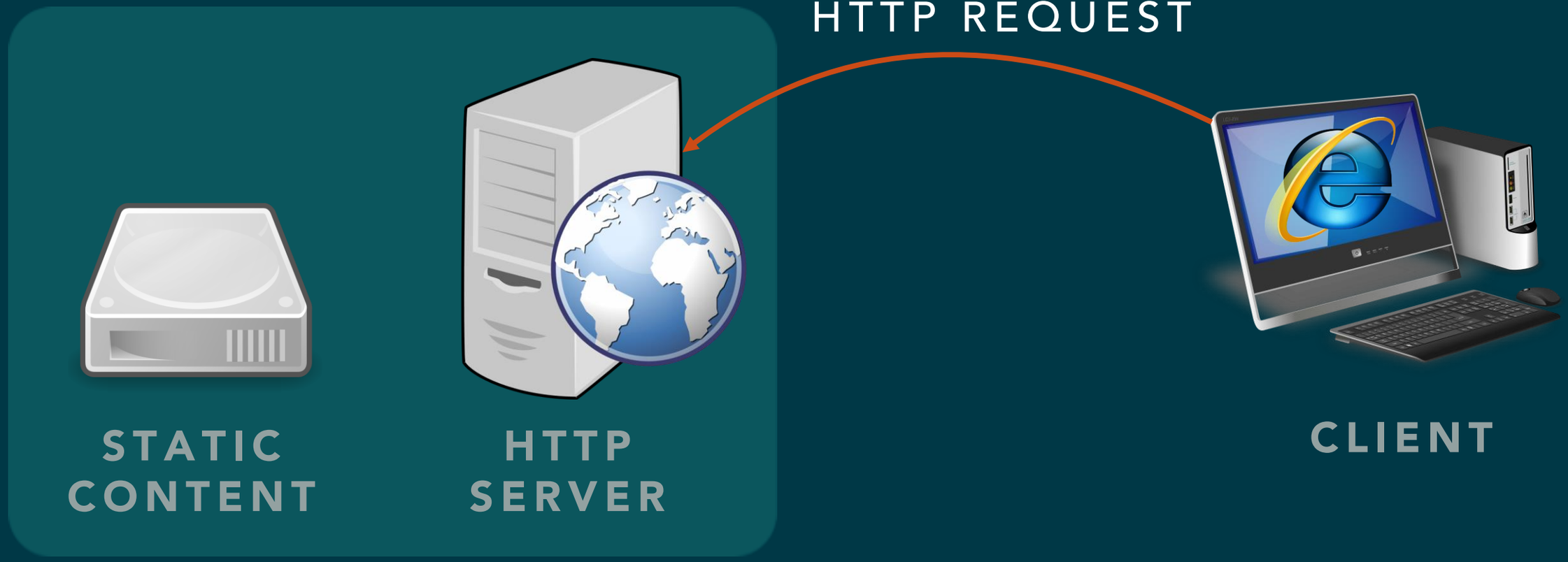
WEB DEVELOPMENT APPROACHES
BEFORE WEB APPS — STATIC WEBSITES

STATIC CONTENT

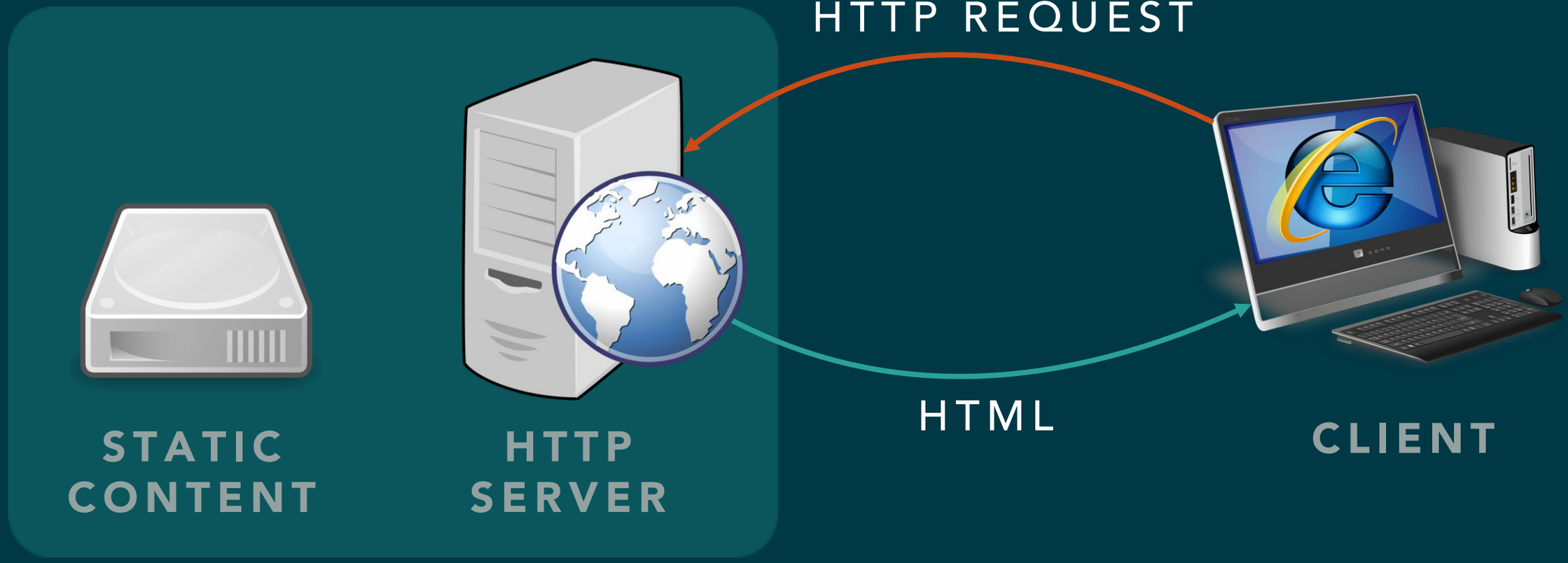
HTTP SERVER

CLIENT

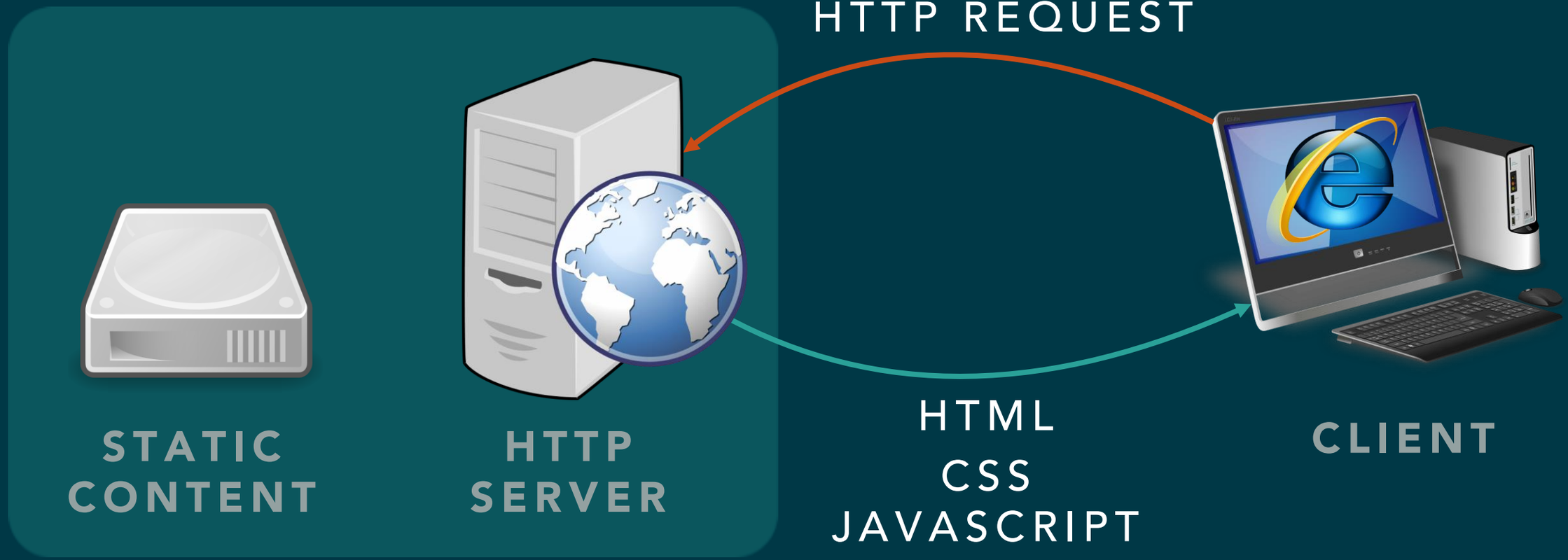
WEB DEVELOPMENT APPROACHES
BEFORE WEB APPS — STATIC WEBSITES



WEB DEVELOPMENT APPROACHES
BEFORE WEB APPS — STATIC WEBSITES



WEB DEVELOPMENT APPROACHES
BEFORE WEB APPS — STATIC WEBSITES



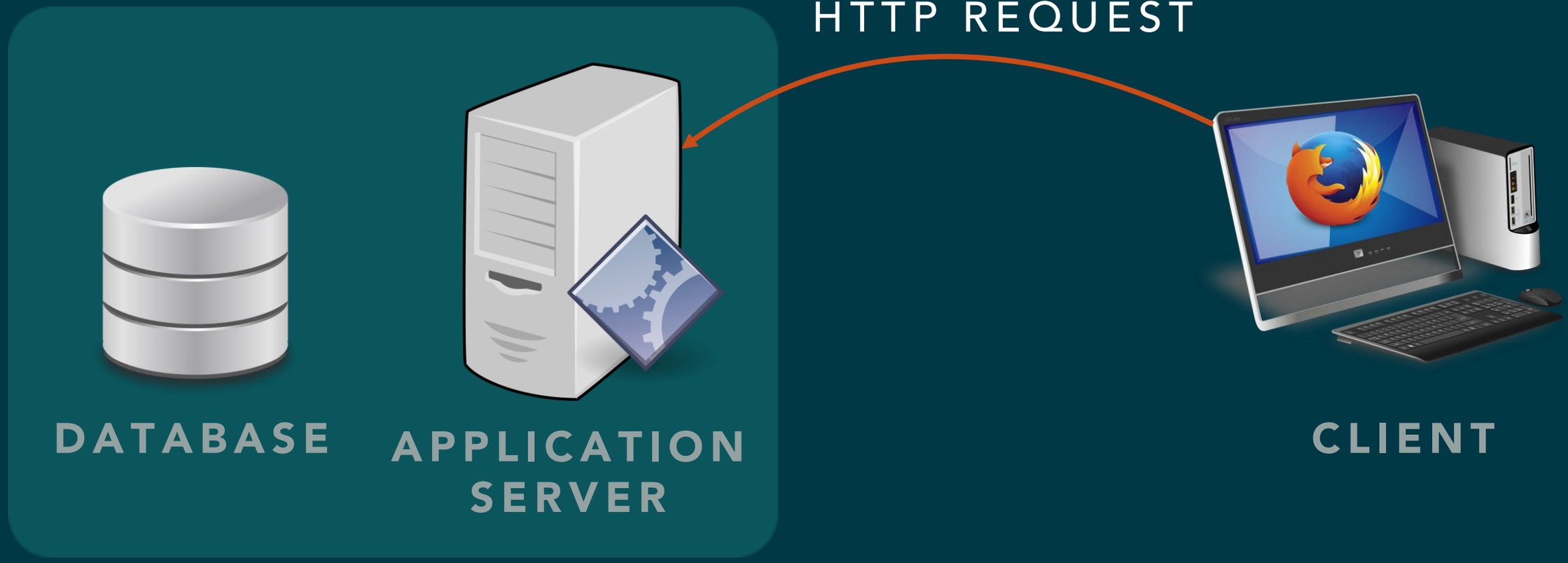
WEB DEVELOPMENT APPROACHES
CLASSIC WEB APPLICATIONS

DATABASE

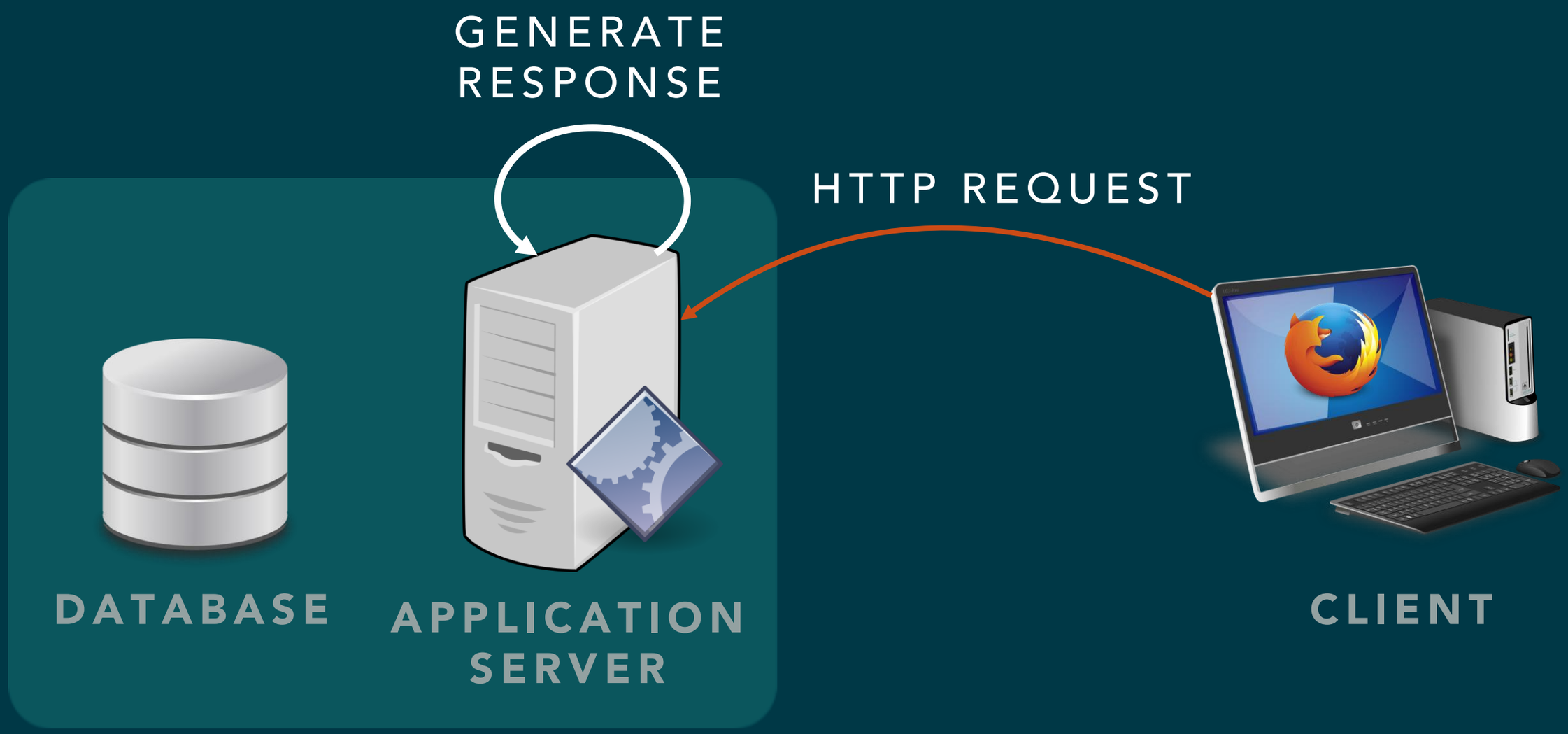
APPLICATION SERVER

CLIENT

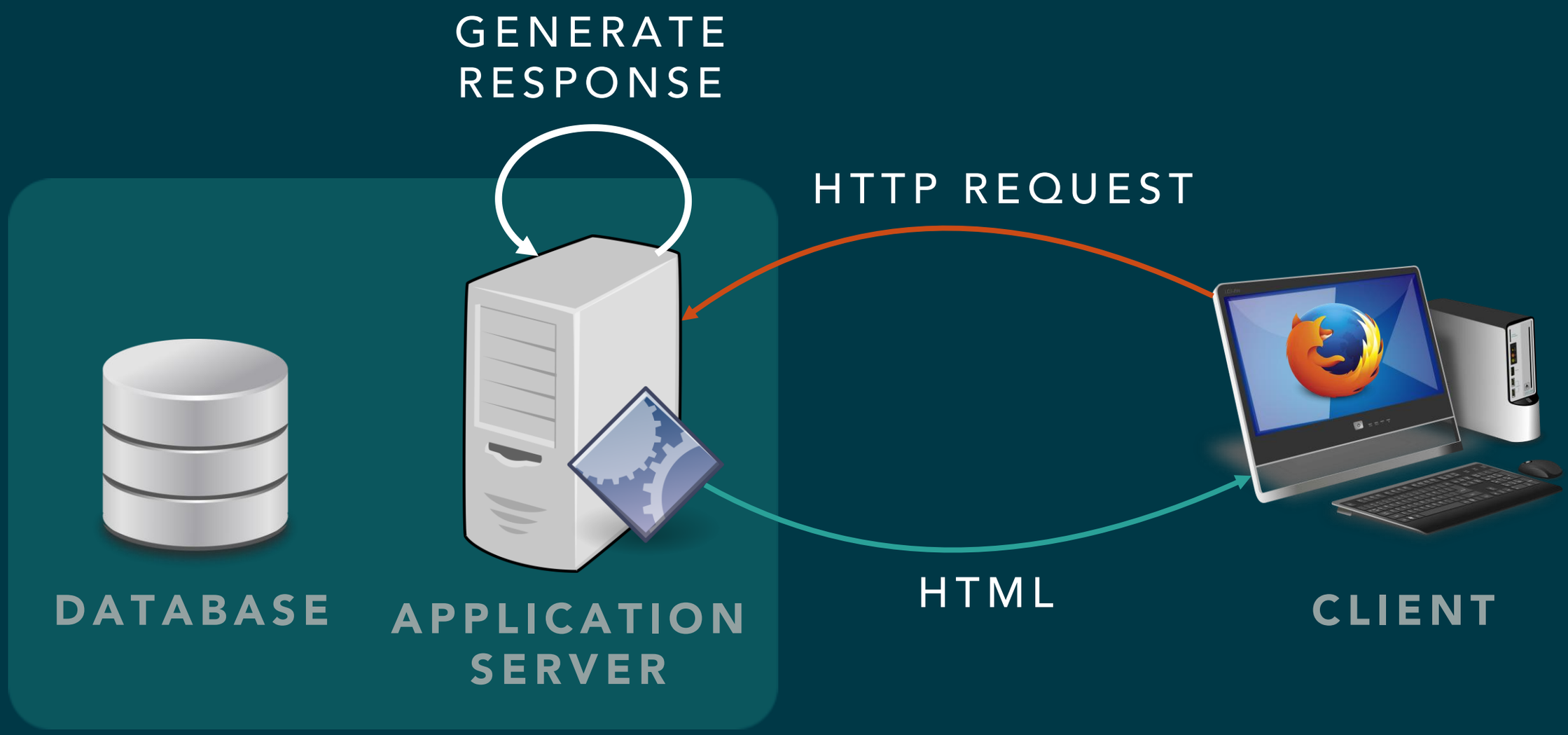
WEB DEVELOPMENT APPROACHES
CLASSIC WEB APPLICATIONS



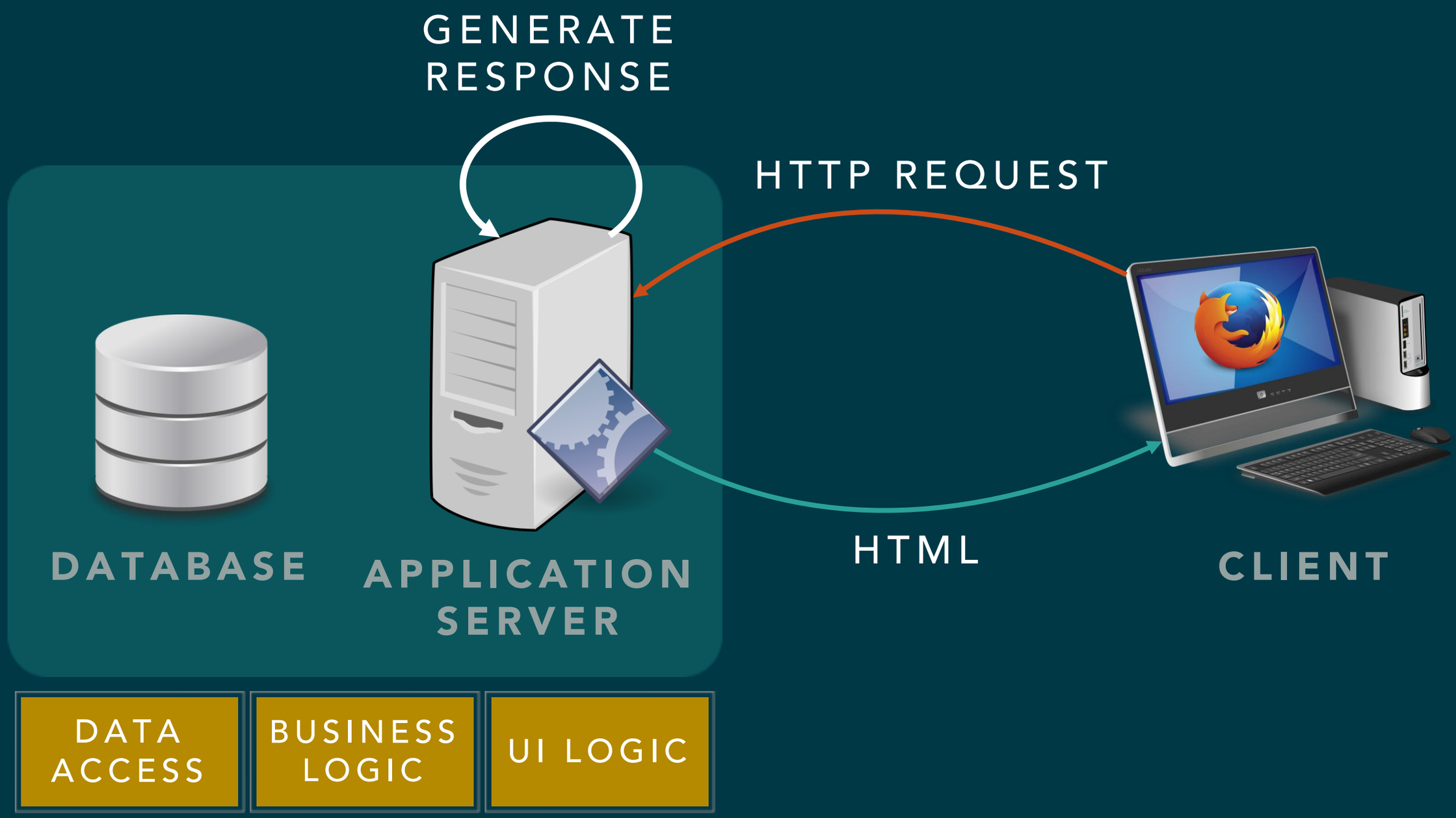
WEB DEVELOPMENT APPROACHES
CLASSIC WEB APPLICATIONS



WEB DEVELOPMENT APPROACHES
CLASSIC WEB APPLICATIONS



WEB DEVELOPMENT APPROACHES
CLASSIC WEB APPLICATIONS



DATA ACCESS | BUSINESS LOGIC | UI LOGIC

MOTIVATION

- **Web services** have existed for years, but the shift to using them as a **foundational design element** of web applications did not occur immediately.

MOTIVATION

- **Web services** have existed for years, but the shift to using them as a **foundational design element** of web applications did not occur immediately.

- Evolution of web application architecture:
  - classic web applications
  - server-side MVC
  - client-side MVC

MOTIVATION

- **Web services** have existed for years, but the shift to using them as a **foundational design element** of web applications did not occur immediately.

- Evolution of web application architecture:
  - classic web applications
  - server-side MVC
  - client-side MVC

- Web services follow the web apps evolution:
  - shift of responsibility from the server to the client,
  - trend towards separating content from presentation.

WEB DEVELOPMENT APPROACHES
CLASSIC WEB APPLICATIONS

- **Programmatic (code-centric)** approaches:
  - Generating response by **executing application** written in a scripting or high-level programming language.
  - HTML and other formatting constructs are embedded within the application logic and produced using output statements.

- **Template** (**document-centric**) approaches:
  - Generating response by **interpreting a template** file.
  - Templates are essentially HTML files with additional **'tags'** that allow for inserting dynamically generated content.
  - The **inverse** of programmatic approaches — application logic is embedded within page formatting structures.

PROGRAMMATIC APPROACHES
JAVA SERVLET EXAMPLE

A servlet is a Java class used to extend the capabilities of servers that host applications accessed by means of a ***request-response*** *programming model (Java Servlet technology defines HTTP-specific servlet classes).*

```
1 public class Hello extends HttpServlet {
2    public void doGet(HttpServletRequest rq, HttpServletResponse rsp) {
3       rsp.setContentType("text/html");
4       try {
5          PrintWriter out = rsp.getWriter();
6          String user = rq.getParameter("user");
7          out.println("<HTML>");
8          out.println("<HEAD><TITLE>Welcome</TITLE></HEAD>");
9          out.println("<BODY>");
10         out.println("<H3>Welcome "+((user==null) ? "" : user)+"!</H3>");
11         out.println("<P>Today is "+new Date()+".</P>");
12         out.println("</BODY>");
13         out.println("</HTML>");
14      } catch (IOException ioe) {// (error processing)
15      }
16   }
17 }
```

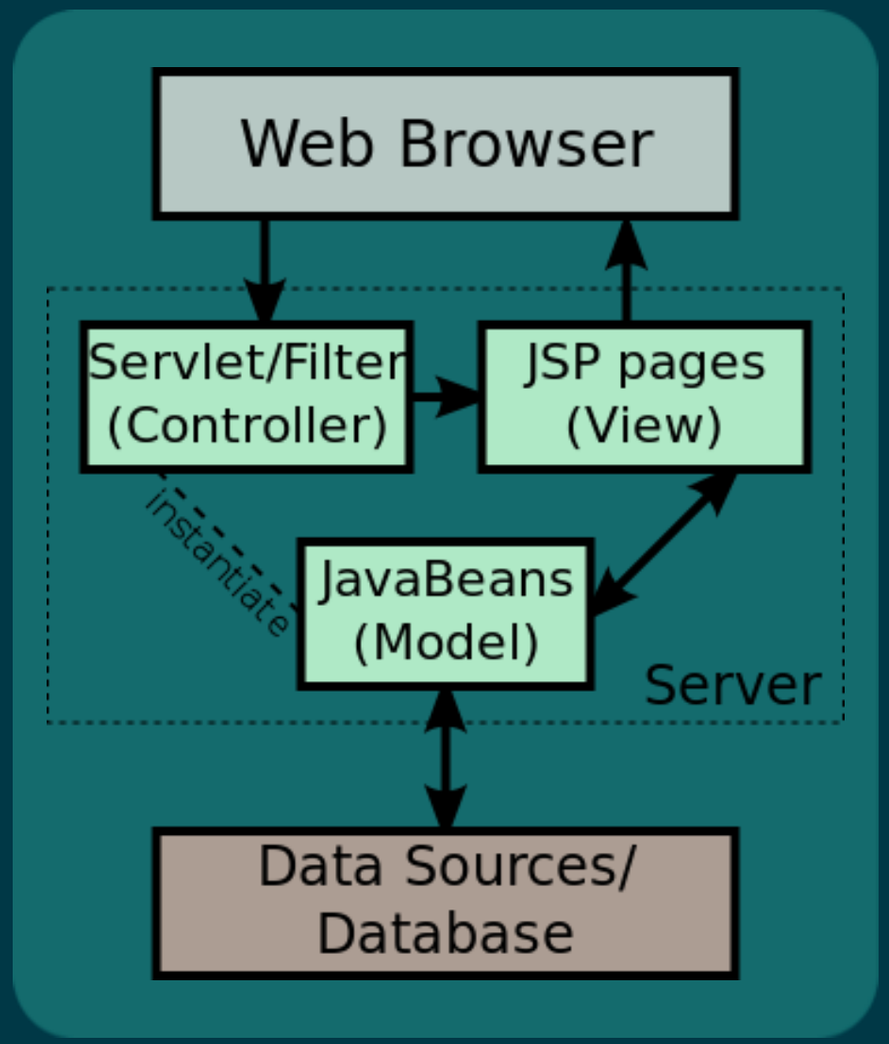
TEMPLATE APPROACHES
JAVA SERVER PAGES EXAMPLE

- JavaServer Pages (JSP)
  - Built on top of the Servlet API
  - JSP pages are **translated into servlets at runtime**
  - A more natural approach with support of all the capabilities of Java Servlets
- A JSP page is a text document that contains two types of text:
  - **Static** data – any text-based format (such as HTML),
  - **JSP elements**, which construct **dynamic** content.

```
1 <HTML>
2 <HEAD><TITLE>Welcome</TITLE></HEAD>
3 <BODY>
4 <% String user=request.getParameter("user"); %>
5 <H3>Welcome <%= (user==null) ? "" : user %>!</H3>
6 <P>Today is <%= new Date() %>.</P>
7 </BODY>
8 </HTML>
```

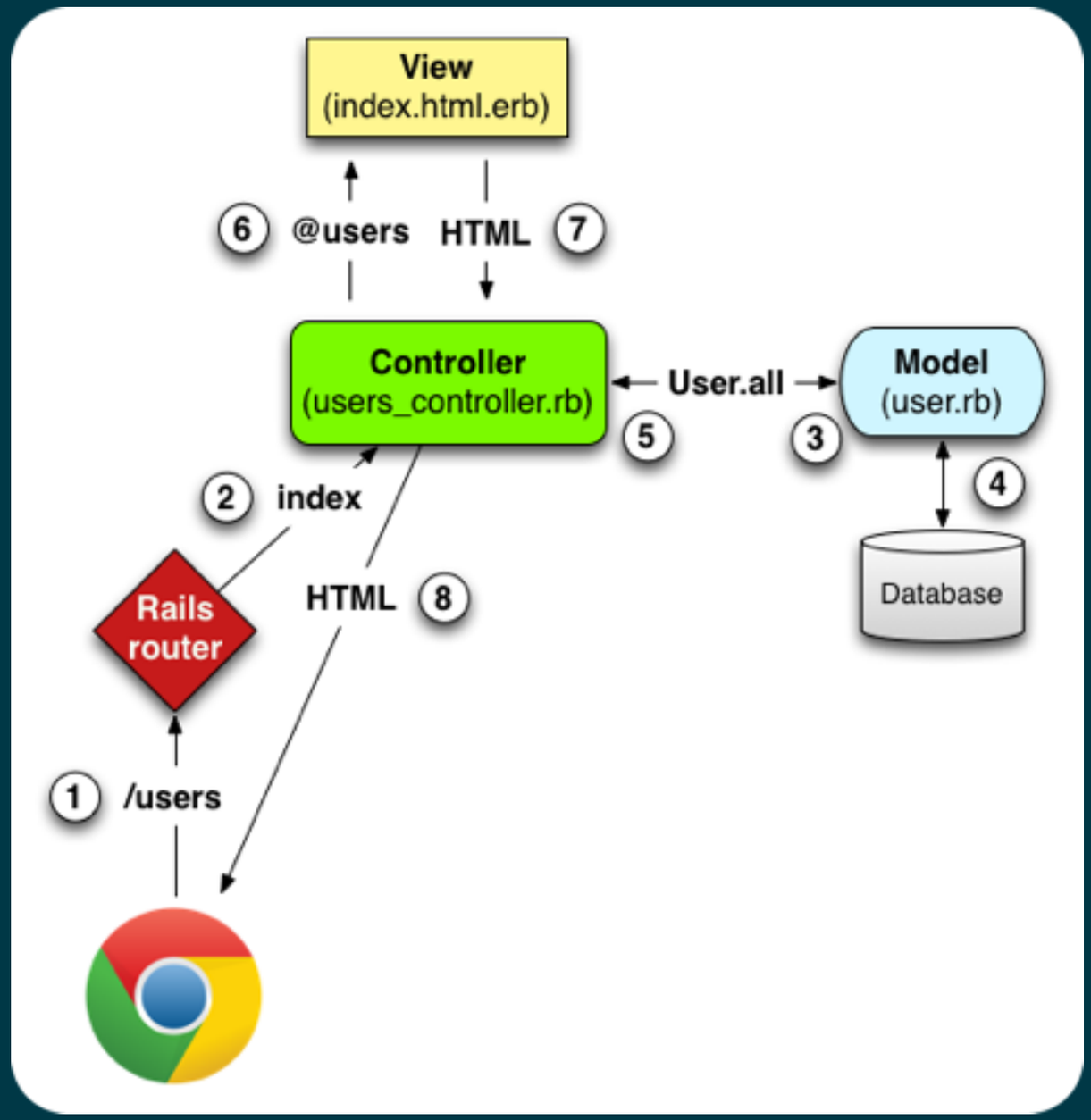
WEB DEVELOPMENT APPROACHES
SERVER-SIDE MVC

- Model-View-Controller — software architecture used by modern web frameworks (but not limited to web):
  - **Model**: encapsulating application data, data access, business logic;
  - **View**: output representation of data, generating user interface (in HTML);
  - **Controller**: handling user interactions, processing user requests, building model and passing it to the view.



WEB DEVELOPMENT APPROACHES
SERVER-SIDE MVC FRAMEWORKS

- Apache Struts
- Java Server Faces
- Spring MVC
- Ruby on Rails
- Django
- ASP.NET MVC



MODERN WEB APPLICATION
CLIENT-SIDE MVC + WEB API

DATABASE

SERVER

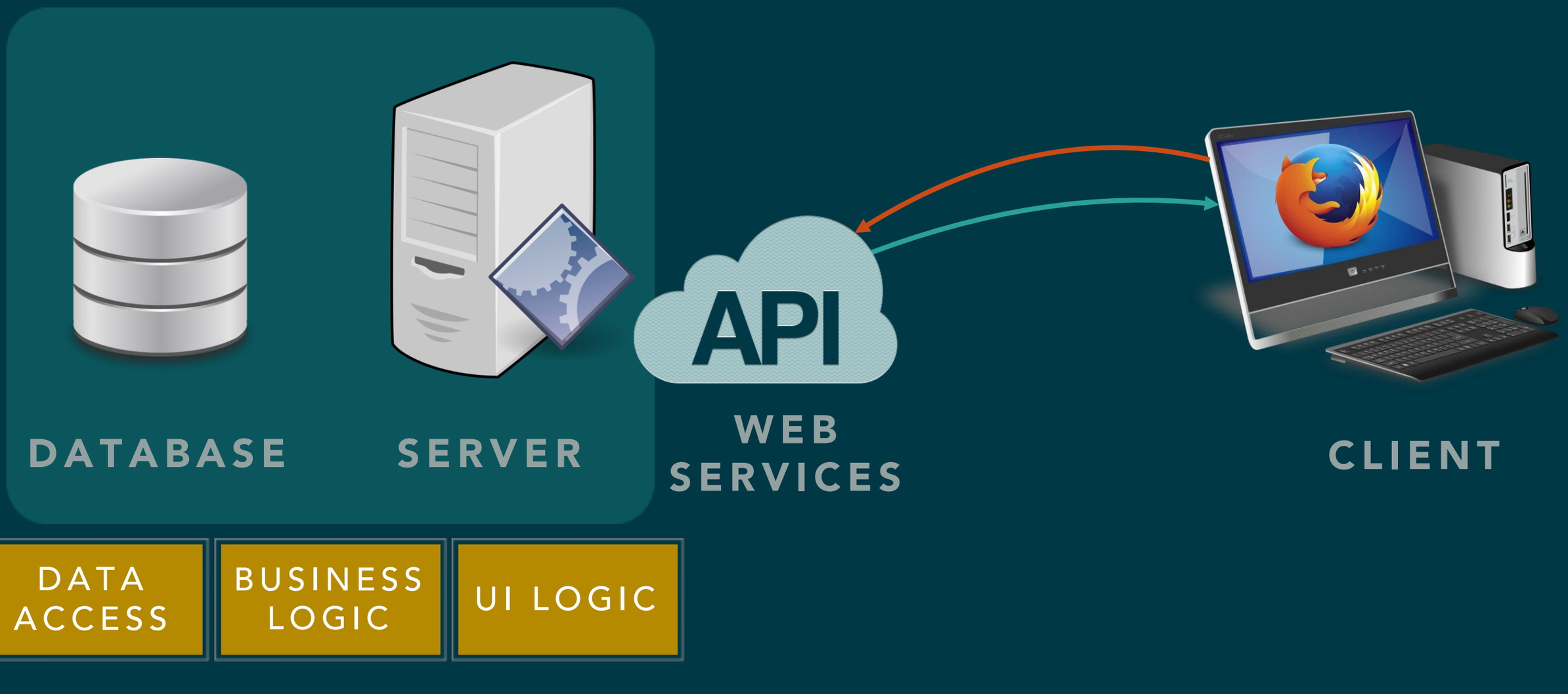
CLIENT

DATA ACCESS

BUSINESS LOGIC

UI LOGIC

MODERN WEB APPLICATION
CLIENT-SIDE MVC + WEB API



DATA ACCESS | BUSINESS LOGIC | UI LOGIC

MODERN WEB APPLICATION
CLIENT-SIDE MVC + WEB API

DATA ACCESS | BUSINESS LOGIC | UI LOGIC

MODERN WEB APPLICATION
CLIENT-SIDE MVC + WEB API

- Heterogeneous clients using the same back-end code.
- **Loose coupling** between back-end and front-end:
  - The back-end is completely unaware on what consumes its service as long as it sends valid requests.
  - As long as the response is valid, and the service does as promised, the front-end doesn't care what the back-end is.
- Improving web development **productivity**.
- More responsive, rich Web applications.

SINGLE PAGE APPLICATIONS

- **Single Page Applications** (SPA) — a web application that requires only a single page load in a web browser; the goal is to provide a more fluid user experience.

- **Thin Server Architecture** — SPA moves logic from the server to the client; the role of the web server is limited to a pure data API (i.e. a **web service**).

- JavaScript frameworks for SPA development:
  - AngularJS
  - Ember.js
  - Backbone.js

PRESENTATION OUTLINE

- History & Motivation
- **What are web services?**
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

WHAT ARE WEB SERVICES?

- *Web services are **client and server applications** that communicate over the HyperText Transfer Protocol and provide a standard means of **interoperating** between applications running on a variety of platforms.*

— JAVA PLATFORM, ENTERPRISE EDITION: THE JAVA EE TUTORIAL

WHAT ARE WEB SERVICES?

- *Web services are **client and server applications** that communicate over the HyperText Transfer Protocol and provide a standard means of **interoperating** between applications running on a variety of platforms.*

— JAVA PLATFORM, ENTERPRISE EDITION: THE JAVA EE TUTORIAL

- *A Web service is a **collection of functions** that are published to the network for use by other programs.*

— *THE WEB SERVICES (R)EVOLUTION*, GRAHAM GLASS

WHAT ARE WEB SERVICES?

- *Web services are **client and server applications** that communicate over the HyperText Transfer Protocol and provide a standard means of **interoperating** between applications running on a variety of platforms.*

— JAVA PLATFORM, ENTERPRISE EDITION: THE JAVA EE TUTORIAL

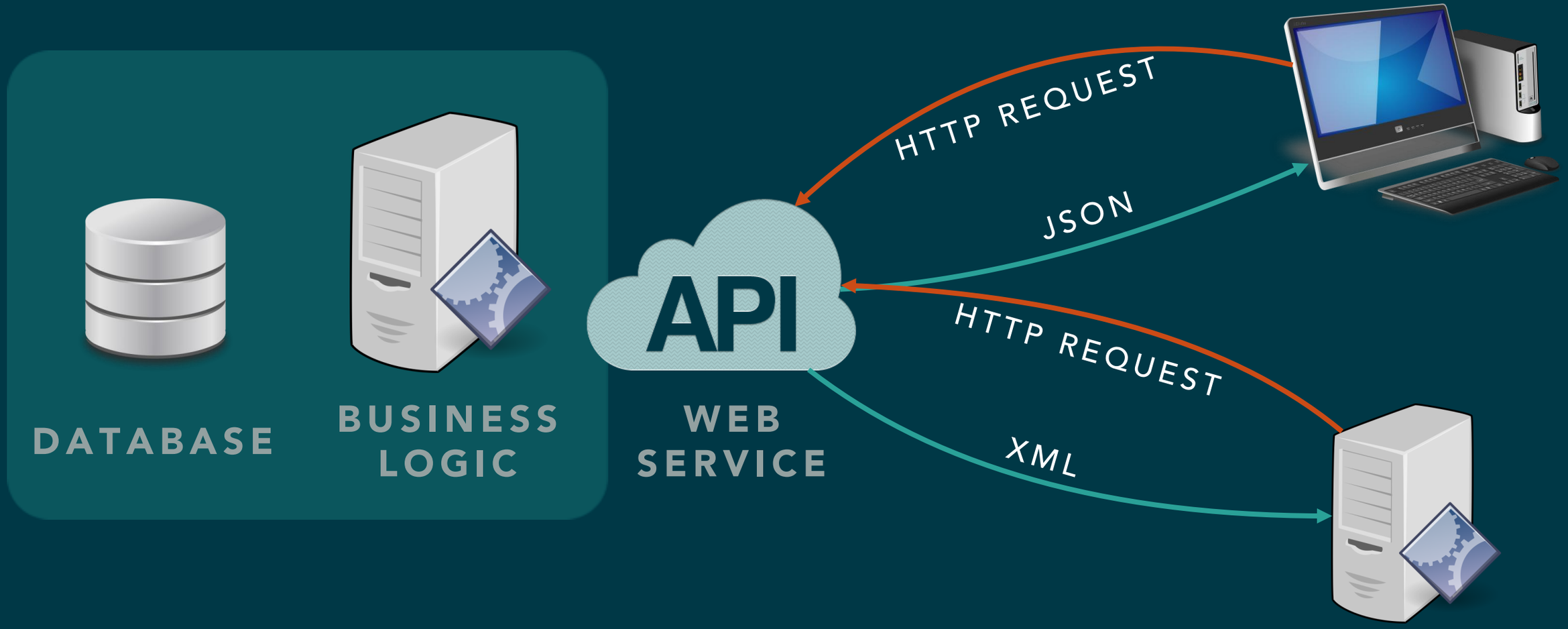
- *A Web service is a **collection of functions** that are published to the network for use by other programs.*

— *THE WEB SERVICES (R)EVOLUTION*, GRAHAM GLASS

- The aim of web services is to provide a way for **non-human** clients to interact with your web application.

WHAT ARE WEB SERVICES?

- Web service — a network accessible **interface** to application functionality that can be invoked remotely using existing **web** infrastructure (over **HTTP**) and supports **machine-to-machine** interaction.



KEY FEATURES OF WEB SERVICES

KEY FEATURES OF WEB SERVICES

- **Web services are platform independent** — they are based on a concise set of open standards (HTTP, XML, JSON) designed to promote **interoperability** between a Web service and clients across a variety of computing platforms and programming languages.

KEY FEATURES OF WEB SERVICES

- **Web services are platform independent** — they are based on a concise set of open standards (HTTP, XML, JSON) designed to promote **interoperability** between a Web service and clients across a variety of computing platforms and programming languages.

- **Web services are self-contained** — no additional software is required, a programming language with XML/JSON and HTTP client support is enough to get started.

KEY FEATURES OF WEB SERVICES

- **Web services are platform independent** — they are based on a concise set of open standards (HTTP, XML, JSON) designed to promote **interoperability** between a Web service and clients across a variety of computing platforms and programming languages.

- **Web services are self-contained** — no additional software is required, a programming language with XML/JSON and HTTP client support is enough to get started.

- **Web services are self-describing** — a public interface to the service is available; definitions of message format travel with the messages.

KEY FEATURES OF WEB SERVICES

- **Web services are platform independent** — they are based on a concise set of open standards (HTTP, XML, JSON) designed to promote **interoperability** between a Web service and clients across a variety of computing platforms and programming languages.

- **Web services are self-contained** — no additional software is required, a programming language with XML/JSON and HTTP client support is enough to get started.

- **Web services are self-describing** — a public interface to the service is available; definitions of message format travel with the messages.

- **Web services are modular** — simple web services can be aggregated to form more complex Web services (e.g. **mashups**), e.g. https://www.programmableweb.com/category/all/apis

TWO TYPES OF WEB SERVICES

TWO TYPES OF WEB SERVICES

- **Big web services:**
  - Simple Object Access Protocol (SOAP)
  - Web Service Description Language (WSDL)
  - Service-Oriented Architecture (SOA)
  - JAX-WS (JSR 224) — *Metro* is the reference implementation

TWO TYPES OF WEB SERVICES

- **Big web services:**
  - Simple Object Access Protocol (SOAP)
  - Web Service Description Language (WSDL)
  - Service-Oriented Architecture (SOA)
  - JAX-WS (JSR 224) — *Metro* is the reference implementation

- **RESTful web services:**
  - Lightweight infrastructure that require minimal tooling.
  - Representational State Transfer — HTTP as an API
  - Resource-Oriented Architecture (ROA)
  - JAX-RS (JSR 339) — *Jersey* is the reference implementation

PRESENTATION OUTLINE

- Motivation
- What are web services?
- Big web services:
  - **Brief history of web services: RPC, XML-RPC**
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

HISTORICAL PERSPECTIVE

- **1976**: Description of the **RPC** principle in RFC 707
- **1986**: RPC/XDR by Sun (RFC 1057)
- **1997**: Java RMI — object-oriented RPC
- **1998**: XML-RPC — the birth of web services
- **1998**: SOAP — turning point when web services started to become more prevalent among enterprises
- **2001**: WSDL & UDDI

REMOTE PROCEDURE CALL

- **RPC** — inter-process communication mechanism that allows a client applications to call procedures in **another address space** without the programmer explicitly coding the details for this remote interaction, just as it were a local procedure (**location transparency**).

- Generating client and server artifacts (**stubs** and **skeletons**) from an **IDL** (Interface Definition Language) document.

```
 1 [uuid(2d6ead46-05e3-11ca-7dd1-426909beabcd), version(1.0)]
 3 interface echo {
 4     const long int ECHO_SIZE = 512;
 5     void echo(
 6        [in]             handle_t h,
 7        [in, string]     idl_char from_client[ ],
 8        [out, string]    idl_char from_server[ECHO_SIZE]
 9     );
10 }
```

REMOTE PROCEDURE CALL

SERVER

IDL

CLIENT

REMOTE PROCEDURE CALL

SERVER

IDL

RPC
COMPILER

CLIENT

REMOTE PROCEDURE CALL

SERVER

IDL

CLIENT

SERVER STUB (SKELETON)

RPC COMPILER

CLIENT STUB

REMOTE PROCEDURE CALL

SERVER

SERVER STUB (SKELETON)

IDL

RPC COMPILER

CLIENT

CLIENT STUB

REMOTE PROCEDURE CALL

SERVER

SERVER STUB (SKELETON)

IDL

RPC COMPILER

CLIENT

CLIENT STUB

NETWORK ROUTINES

LOCAL OS

REMOTE PROCEDURE CALL

SERVER

IDL

CLIENT

SERVER STUB (SKELETON)

RPC COMPILER

CLIENT STUB

NETWORK ROUTINES

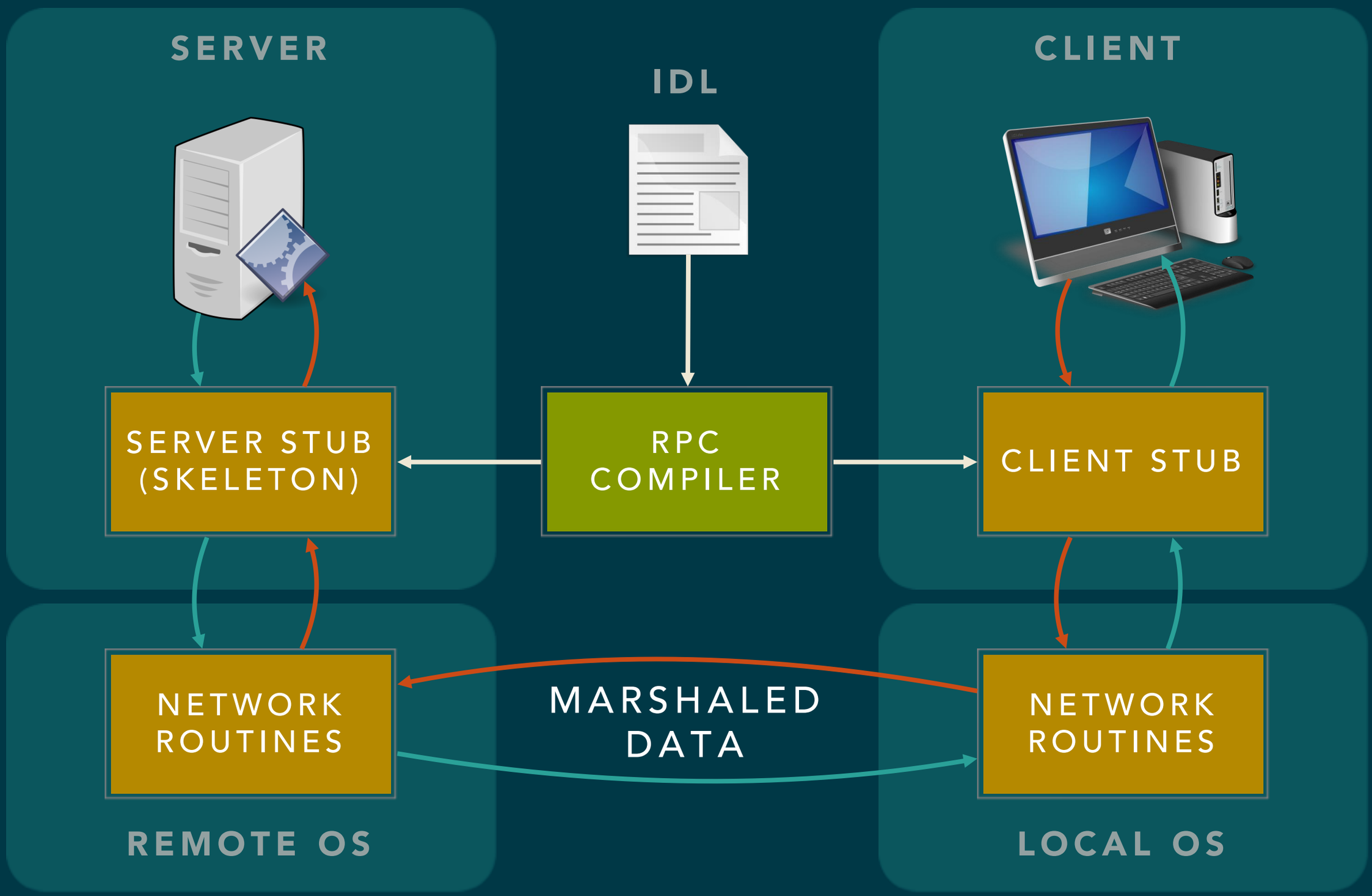
MARSHALED DATA

NETWORK ROUTINES

REMOTE OS

LOCAL OS

REMOTE PROCEDURE CALL



XML-RPC

- XML-RPC — a very **lightweight** RPC protocol with support for elementary data types.
  - **XML** marshaling to achieve language neutrality — messages are easy to inspect and process with standard tools.
  - **HTTP** for transport, instead of proprietary system — alleviates the traditional firewall issues of having to open additional ports for RPC.

XML-RPC

- XML-RPC — a very **lightweight** RPC protocol with support for elementary data types.
  - **XML** marshaling to achieve language neutrality — messages are easy to inspect and process with standard tools.
  - **HTTP** for transport, instead of proprietary system — alleviates the traditional firewall issues of having to open additional ports for RPC.
- Request-response paradigm:
  - The client sends a HTTP request — the body is an XML document specifying a single call to a method (method name + parameters).
  - The server replies with a response — body also contains XML.

XML-RPC

- XML-RPC — a very **lightweight** RPC protocol with support for elementary data types.
  - **XML** marshaling to achieve language neutrality — messages are easy to inspect and process with standard tools.
  - **HTTP** for transport, instead of proprietary system — alleviates the traditional firewall issues of having to open additional ports for RPC.
- Request-response paradigm:
  - The client sends a HTTP request — the body is an XML document specifying a single call to a method (method name + parameters).
  - The server replies with a response — body also contains XML.
- E.g. http://codex.wordpress.org/XML-RPC_WordPress_API

XML-RPC EXAMPLE

```
POST /xmlrpc HTTP/1.1
Content-Type: text/xml

<?xml version="1.0"?>
<methodCall>
   <methodName>countCharacters</methodName>
      <params>
         <param>
            <value><string>test</string></value>
         </param>
      </params>
</methodCall>
```

XML-RPC EXAMPLE

```
POST /xmlrpc HTTP/1.1
Content-Type: text/xml

<?xml version="1.0"?>
<methodCall>
   <methodName>countCharacters</methodName>
      <params>
         <param>
            <value><string>test</string></value>
         </param>
      </params>
</methodCall>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><int>4</int></value>
      </param>
   </params>
</methodResponse>
```

XML-RPC FAULT EXAMPLE

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
  <fault>
     <value>
        <struct>
           <member>
              <name>faultCode</name>
              <value>
                 <int>4</int>
              </value>
           </member>
           <member>
              <name>faultString</name>
              <value>
                 <string>Too many parameters.</string>
              </value>
           </member>
        </struct>
     </value>
  </fault>
</methodResponse>
```

XML-RPC SPECIFICATION

- XML-RPC is a simple specification without ambitious goals — stub generation, interface description and service lookup are not in the protocol.

- *We wanted a clean, extensible format that's very simple. It should be possible for an HTML **coder** to be able to look at a file containing an XML-RPC procedure call, **understand** what it's doing, and be able to **modify** it and have it work on the first or second try.*

—HTTP://XMLRPC.SCRIPTING.COM/SPEC

PRESENTATION OUTLINE

- Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - **Web services protocol stack: SOAP, WSDL**
  - Java API for XML Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

SOAP-BASED WEB SERVICES

- *A Web service is a software system designed to support **interoperable machine-to-machine** interaction over a network. It has an interface described in a machine-processable format (**WSDL**).*

- *Other systems interact with the Web service in a manner prescribed by its description using **SOAP**-messages, typically conveyed using **HTTP** with an **XML** serialization in conjunction with other Web-related standards.* —WEB SERVICES GLOSSARY, WWW.W3.ORG, 2004

SOAP-BASED WEB SERVICES

- *A Web service is a software system designed to support **interoperable machine-to-machine** interaction over a network. It has an interface described in a machine-processable format (**WSDL**).*

- *Other systems interact with the Web service in a manner prescribed by its description using **SOAP**-messages, typically conveyed using **HTTP** with an **XML** serialization in conjunction with other Web-related standards.* —WEB SERVICES GLOSSARY, WWW.W3.ORG, 2004

- Web services **protocol stack**:
  - messaging protocol — XML, SOAP
  - transport protocol — HTTP, SMTP, JMS
  - service description protocol — WSDL
  - service discovery protocol — UDDI

SOAP-BASED WEB SERVICES

SERVICE REGISTRY
(UDDI)

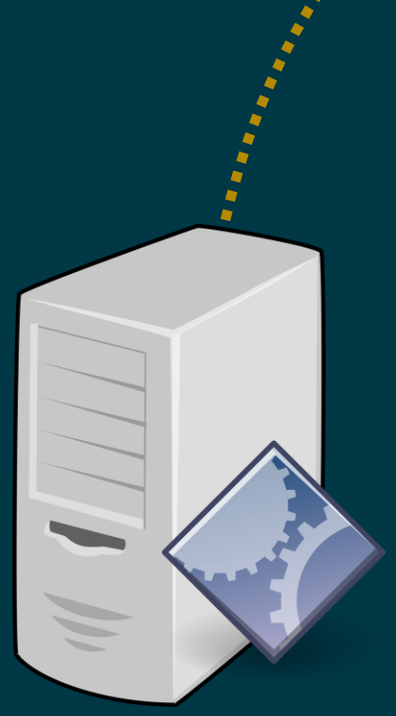
SERVICE
PROVIDER

SERVICE
CONSUMER

SOAP-BASED WEB SERVICES

SERVICE REGISTRY
(UDDI)

1. WSDL



SERVICE
PROVIDER

SERVICE
CONSUMER

SOAP-BASED WEB SERVICES

SERVICE REGISTRY
(UDDI)

1. WSDL

2. WSDL

SERVICE
PROVIDER

SERVICE
CONSUMER

SOAP-BASED WEB SERVICES

SERVICE REGISTRY
(UDDI)

1. WSDL

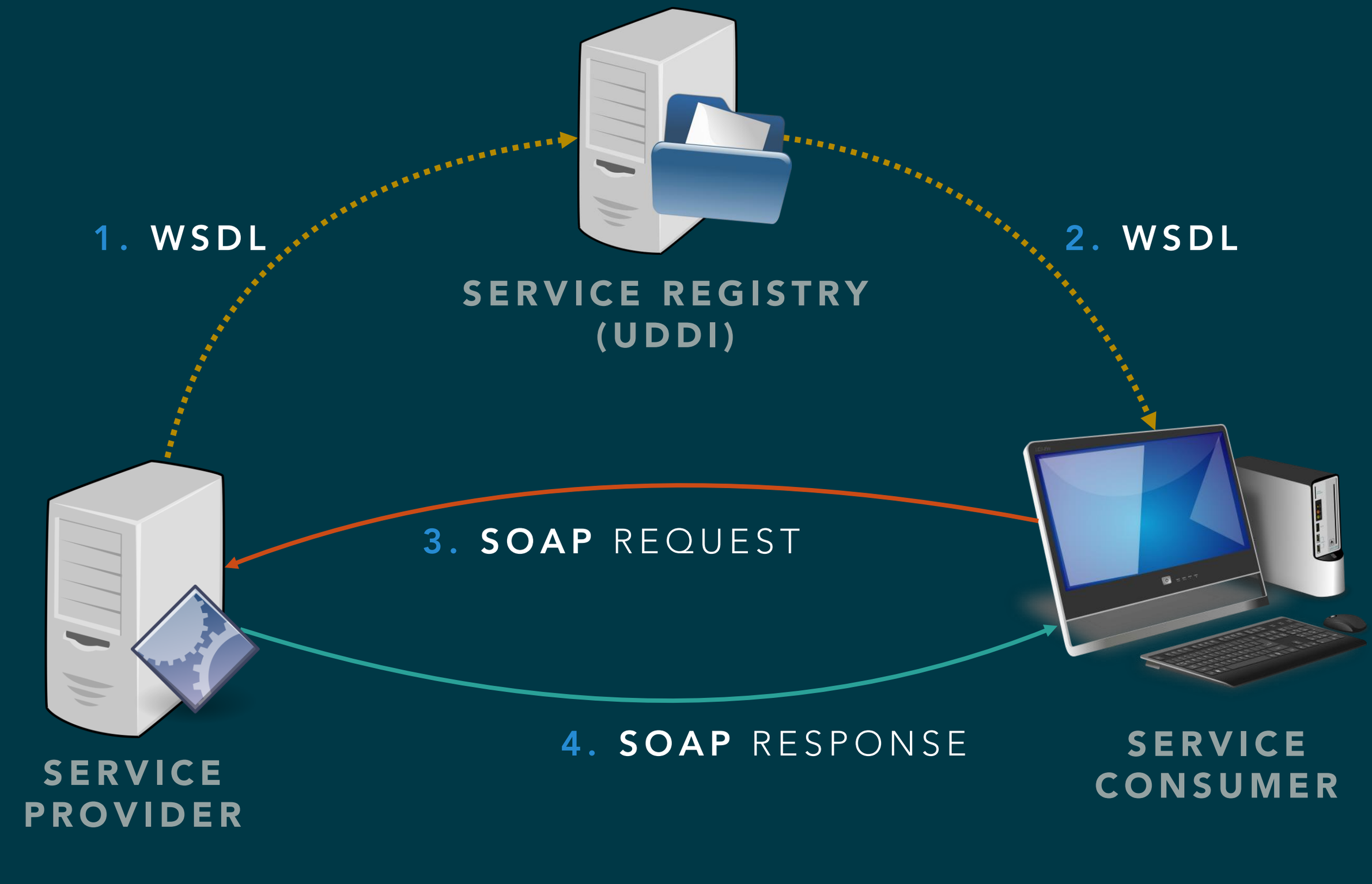
2. WSDL

SERVICE
PROVIDER

3. **SOAP** REQUEST

SERVICE
CONSUMER

SOAP-BASED WEB SERVICES



SOAP PROTOCOL

SOAP PROTOCOL

- SOAP is a XML-based protocol for exchange of information in a decentralized, distributed environment.

SOAP PROTOCOL

- SOAP is a XML-based protocol for exchange of information in a decentralized, distributed environment.

- SOAP — foundation of the web services protocol stack, providing a basic **messaging framework**:
  - XML-based structure of SOAP messages.
  - processing model for transferring messages.
  - guidance on how to transport SOAP messages.

SOAP PROTOCOL

- SOAP is a XML-based protocol for exchange of information in a decentralized, distributed environment.

- SOAP — foundation of the web services protocol stack, providing a basic **messaging framework**:
  - XML-based structure of SOAP messages.
  - processing model for transferring messages.
  - guidance on how to transport SOAP messages.

- SOAP has three major characteristics:
  - **extensibility** — multiple WS-* standards built on top of
  - **neutrality** — SOAP can operate over any transport protocol
  - **independence** — SOAP allows for any programming model

SOAP MESSAGE STRUCTURE

- A SOAP message is an XML document that consists of an **envelope**, an optional **header**, a **body** and optional **fault**.

SOAP MESSAGE STRUCTURE

- A SOAP message is an XML document that consists of an **envelope**, an optional **header**, a **body** and optional **fault**.

SOAP MESSAGE EXAMPLE

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

SOAP RESPONSE EXAMPLE

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

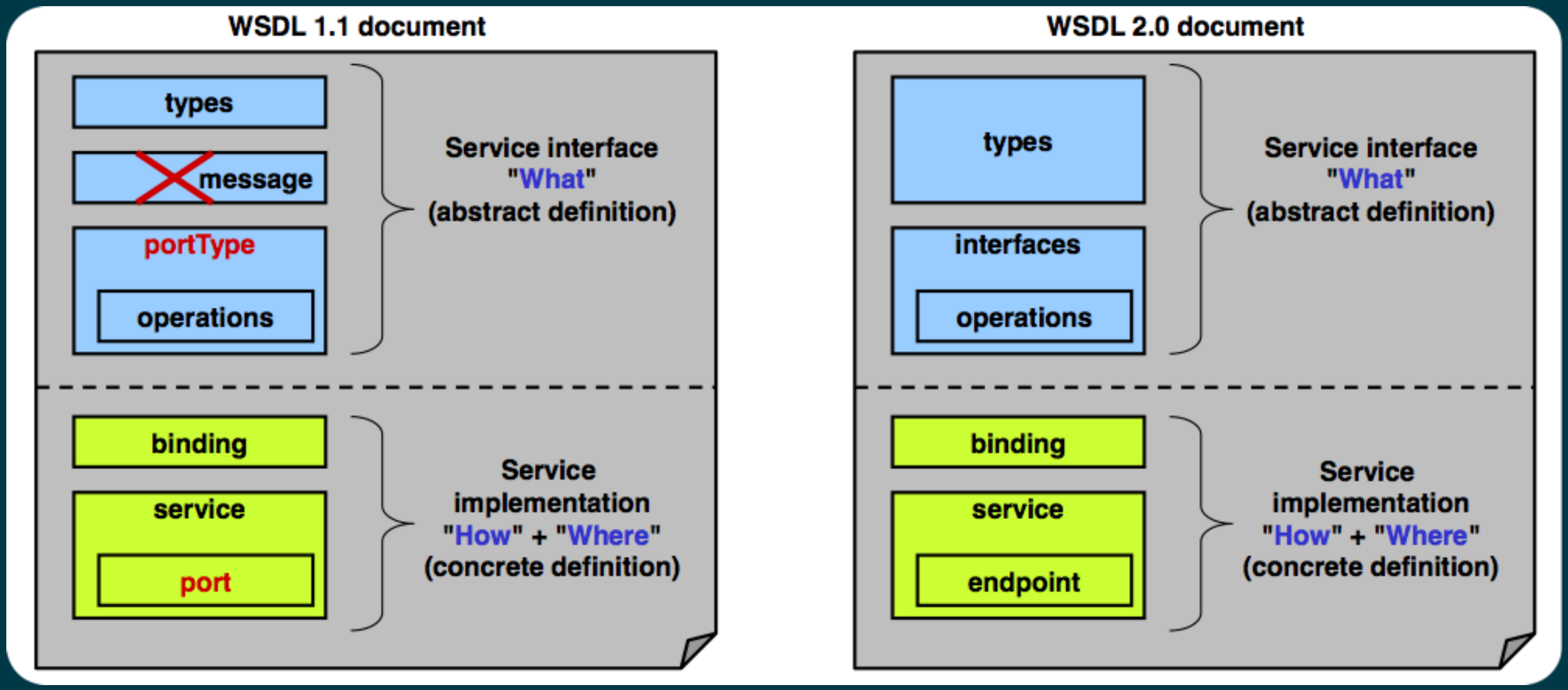
WEB SERVICE DESCRIPTION LANGUAGE

- WSDL — an XML-based **interface definition language** used for describing the functionality of a web service.

- WSDL provides a **machine-readable** description of:
  - what operations are available in the service,
  - what data are required and returned by service operations,
  - what message-exchange pattern is employed,
  - how to connect to the service provider,
  - where to find the service endpoint.

- WSDL acts as a **service contract** that users should observe and the web service promises to abide by.

WEB SERVICE DESCRIPTION LANGUAGE



- Address book service example: http://goo.gl/aBiMTZ

WSDL IN PRACTICE

- Many frameworks ship with built-in support
  - wsimport in JDK 1.6
  - IntelliJ IDEA Ultimate
  - wsld.exe in .NET Framework Tools
  - Visual Studio
- WSDL documents, as service contracts, should be **publishable** and **discoverable**, as are the services that they describe.
- Universal Description Discovery and Integration (**UDDI**) registry
  - Stores published WSDL
  - Clients can discover the document and consume the web service that it describes
- *Public* UDDIs have not been as widely adopted — Microsoft, IBM and SAP shut down their UDDI servers in 2006
- Private UDDIs are still in use

PRESENTATION OUTLINE

- History & Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - **Java API for XML Web Services (JAX-WS)**
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

JAVA API FOR XML WEB SERVICES

- JAX-WS is a standard (part of the Java EE platform) for building web services and clients that communicate using **SOAP**
- JAX-WS is the successor to JAX-RPC derived from XML-RPC
- JAX-WS specification (JSR 229) defines the **mapping between WSDL 1.1 and Java**:
  - defines how different WSDL constructs are mapped to Java.
  - defines how Java packages, classes, interfaces, methods, parameters, and other parts of a web service endpoint are mapped to WSDL

JAX-WS ON THE SERVER SIDE

- On the server side, the web service methods are defined by an interface or a class written in Java
- JAX-WS relies on simple **annotations** added to plain Java classes to define **service endpoint interfaces**:

```
 1 @WebService
 2 public class Hello {
 3     private final String message = "Hello, ";
 4
 5     public Hello() {
 6     }
 7
 8     @WebMethod
 9     public String sayHello(@WebParam(name = "name") String name) {
10         return message + name + ".";
11     }
12 }
```

JAX-WS ON THE CLIENT SIDE

- A client creates a **proxy** (a local object representing the service) and then simply invokes methods on the proxy
- Client proxies can be automatically generated from WSDL using the **wsimport** tool which follows the WSDL-to-Java mapping
- The **JAX-WS runtime** system converts the API calls and responses to and from SOAP messages

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

CLIENT

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

SERVICE ENDPOINT INTERFACE

CLIENT

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

SERVICE ENDPOINT INTERFACE → WSGEN TOOL

CLIENT

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

SERVICE ENDPOINT INTERFACE

WSGEN TOOL

XSD

WSDL

CLIENT

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

SERVICE ENDPOINT INTERFACE

XSD

WSDL

WSGEN TOOL

WSIMPORT TOOL

CLIENT

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

SERVICE ENDPOINT INTERFACE

WSGEN TOOL

XSD

WSDL

WSIMPORT TOOL

CLIENT

CLIENT PROXY

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

SERVER

SERVICE ENDPOINT INTERFACE

WSGEN TOOL

XSD

WSDL

WSIMPORT TOOL

CLIENT

CLIENT PROXY

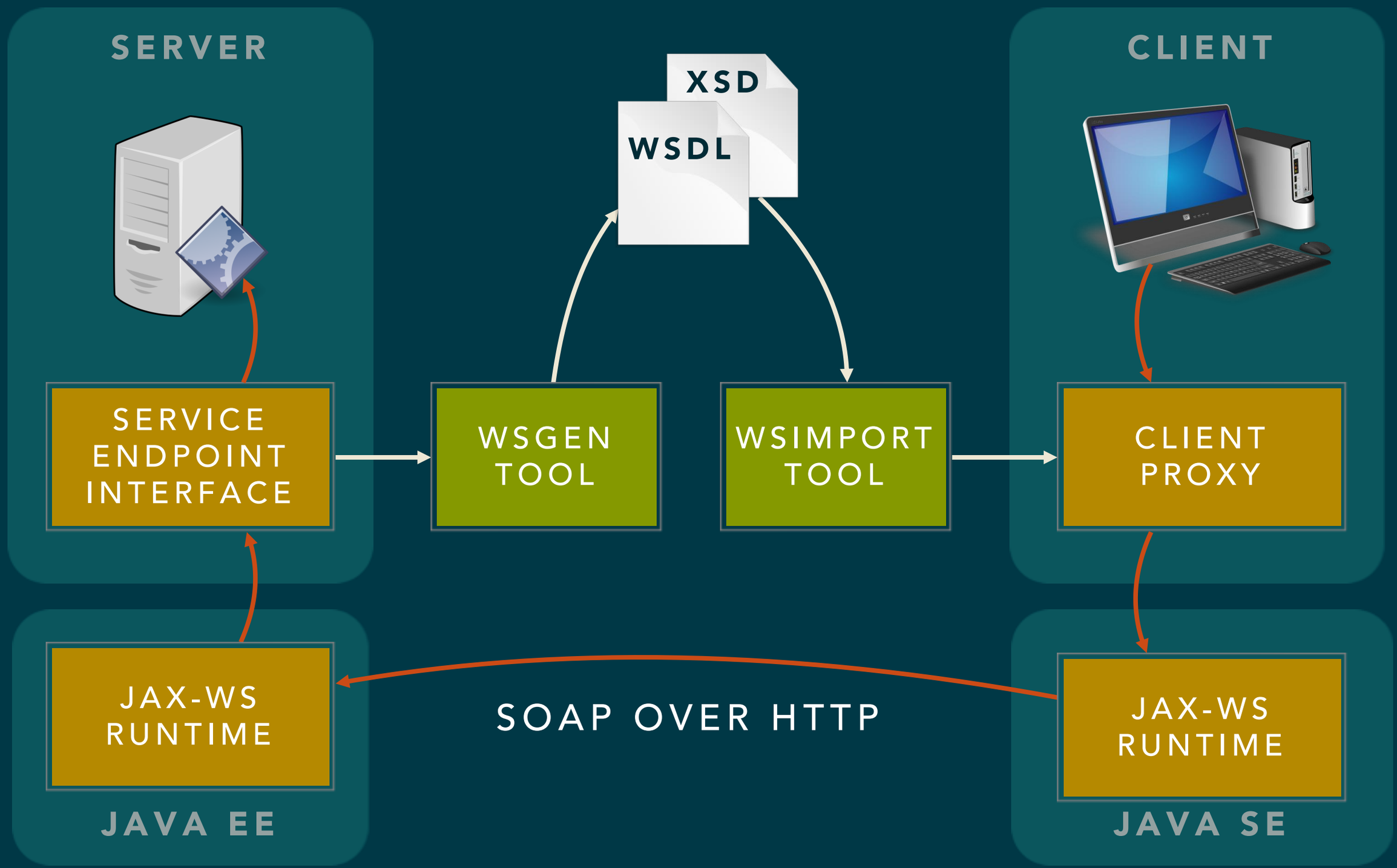
JAX-WS RUNTIME

JAVA EE

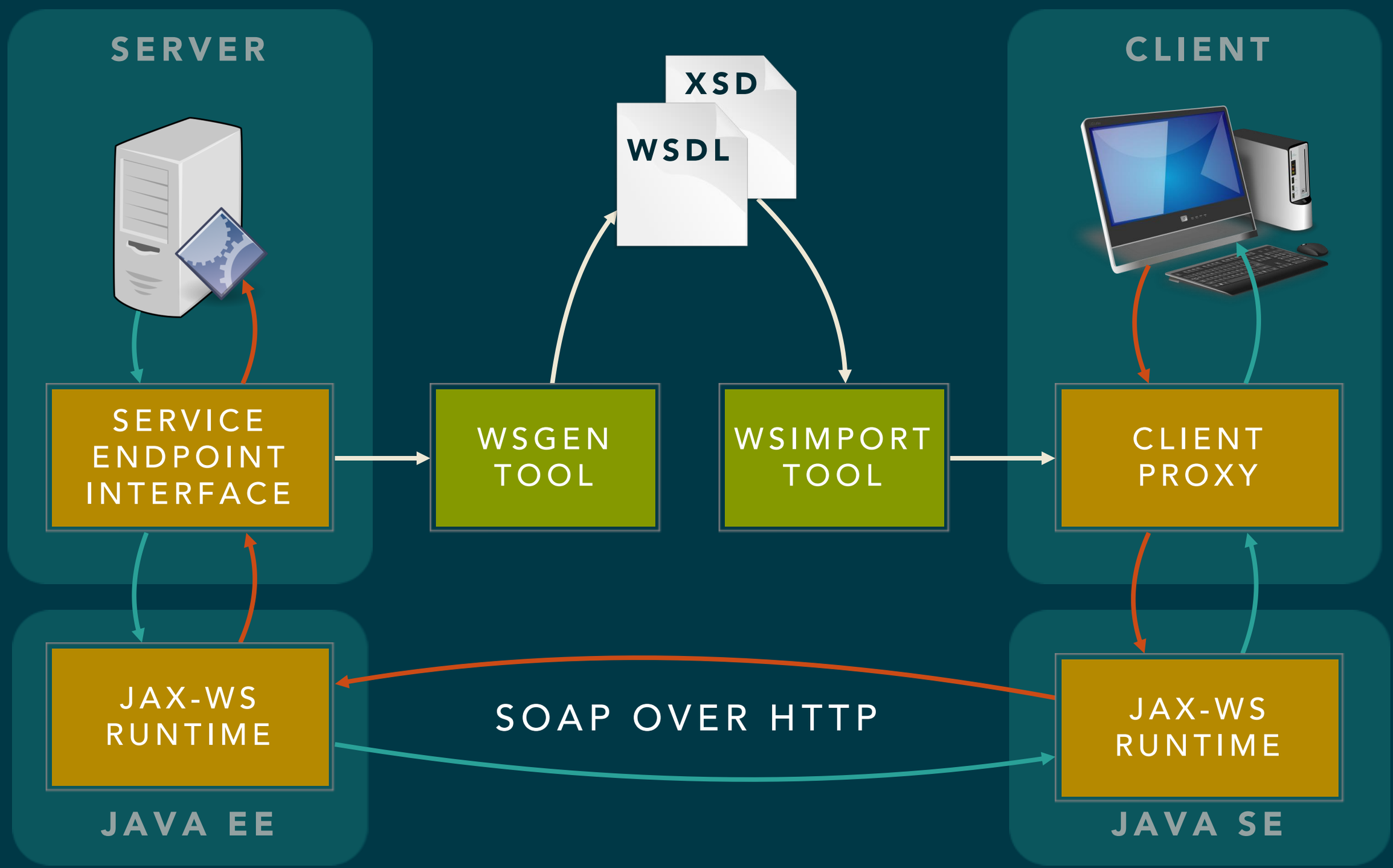
JAX-WS RUNTIME

JAVA SE

JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

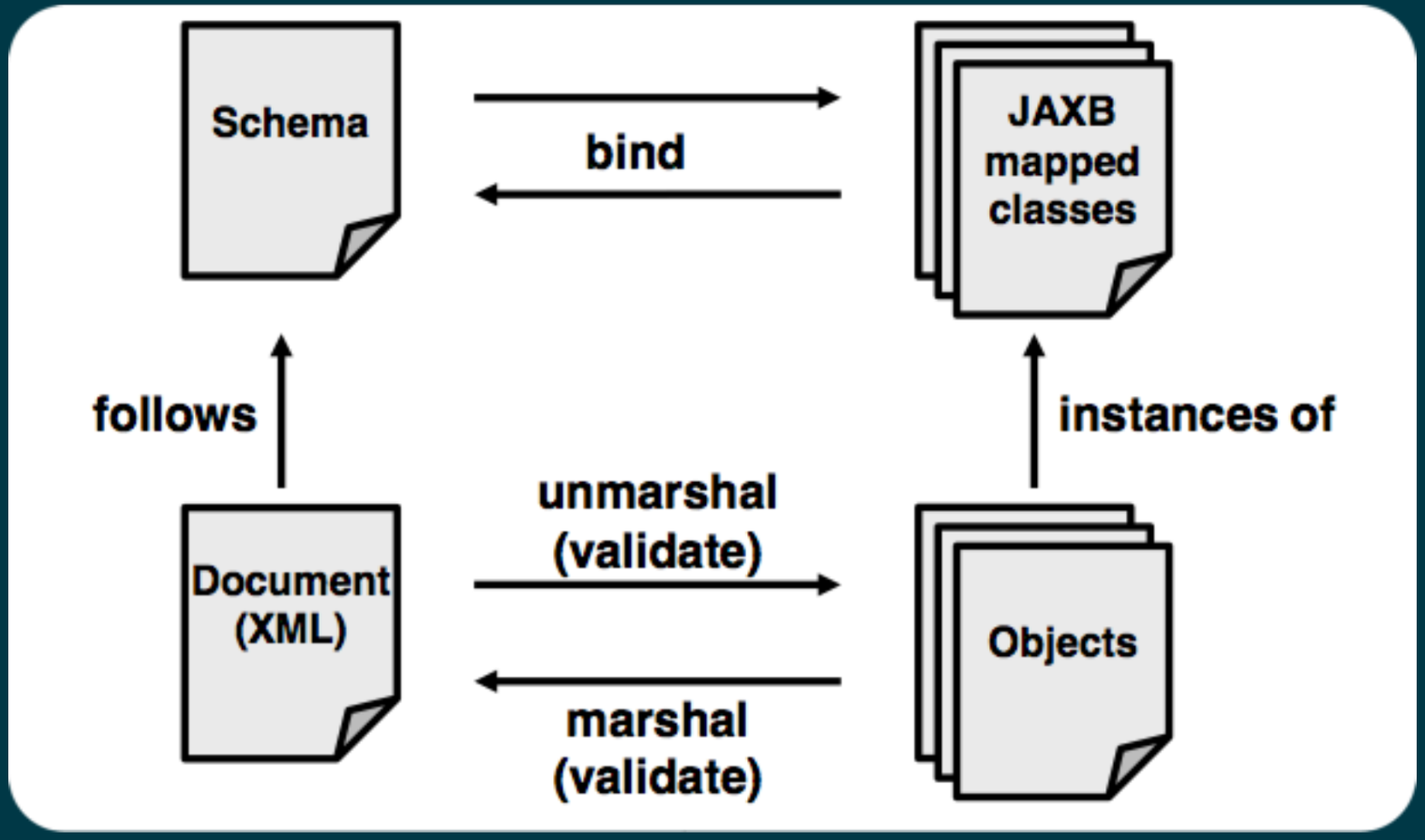


JAX-WS IN ACTION
CODE FIRST DEVELOPMENT

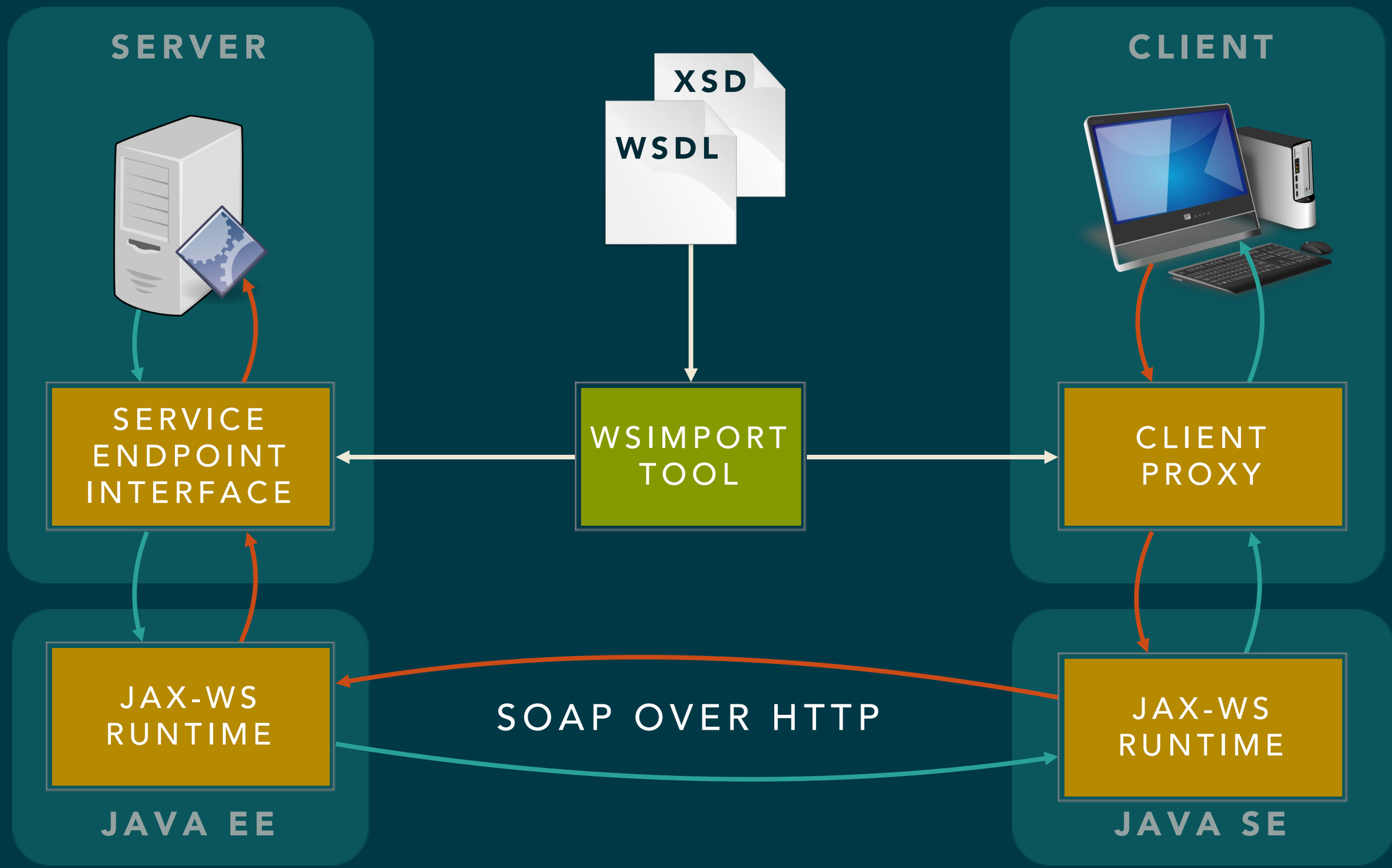


JAVA ARCHITECTURE FOR XML BINDING

- JAX-WS delegates the mapping of Java types to and from XML definitions to **JAXB** — annotations can be used to customize the mapping.
- Business methods that are exposed to web service clients must have JAXB-compatible parameters and return types.



JAX-WS IN ACTION
CONTRACT FIRST DEVELOPMENT

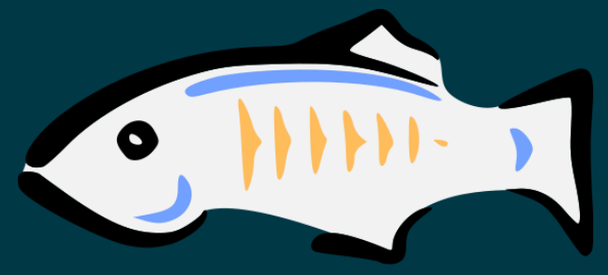


CODE FIRST VS. CONTRACT FIRST

- Code-first approach is dominant beacause it is easier.
- However, it has some drawbacks, including:
  - **fragility** — implementation code typically changes at a faster rate than the interface (or a formal **service contract**); changes in the back-end can break any existing clients
  - **tight-coupling** — interfaces are not separated from implementations, internal models are **exposed**
  - **language-dependence** — different WS stacks generate different service contract; Object/XML impedance **mismatch**
  - **performance** — automatically generated XMLs can be bloated

JAVA WEB SERVICES FRAMEWORKS

- **Metro** — a high-performance, extensible, easy-to-use web service stack which is developed as a part of the open source **GlassFish** project.



- Components of Metro include:
  - **JAX-WS RI** — reference implementation of the JAX-WS
  - **JAXB RI** — data binding used in every Java WS framework
  - **WSIT** — support for **quality of service** features (reliability, security, transactions) and **interoperability** with .NET
- Metro is bundled with Java SE and GlassFish server.

JAVA WEB SERVICES FRAMEWORKS

- Apache Axis2 — Apache Extensible Interaction System
  - support for wide range of XML-Java **binding frameworks** (XMLBeans, JiXB) and **transport protocols** (Jabber, UDP)
  - not fully compliant with JAX-WS
- Apache CXF — Celtix + XFire
  - simple to **integrate** CXF into existing systems, intuitive, easy to use
  - fully compliant with JAX-WS and **JAX-RS** specifications
- Spring Web Services:
  - emphasis on **contract-first** development
  - best aligned with **Spring** technology stack (Spring Annotations, Spring Security), does not implement JAX-WS.

PRESENTATION OUTLINE

- History & Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
  - **.NET API for Web Services (WCF)**
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

WINDOWS COMMUNICATION FOUNDATION (WCF)

- WCF
  - Infrastructure
  - Set of APIs for SOA applications
  - Uniform interfaces for multiple communication technologies
    - Code development is separated from how communication is performed
    - A mode of communication is fully configurable
- Support for
  - SOAP
  - REST
  - MSMQ
  - Named pipes

WCF – CODE FIRST APPROACH

- Define data contract

```
[DataContract]
4 references
public class StringData
{
    [DataMember]
    public string OriginalString;


    [DataMember]
    public string FlippedCaseString;

}
```

WCF – CODE FIRST APPROACH

- Define service contract

```
[ServiceContract]
1 reference
public interface IFlipCaseService
{

    [OperationContract]
    1 reference
    StringData FlipTheCase(StringData sd);
}
```

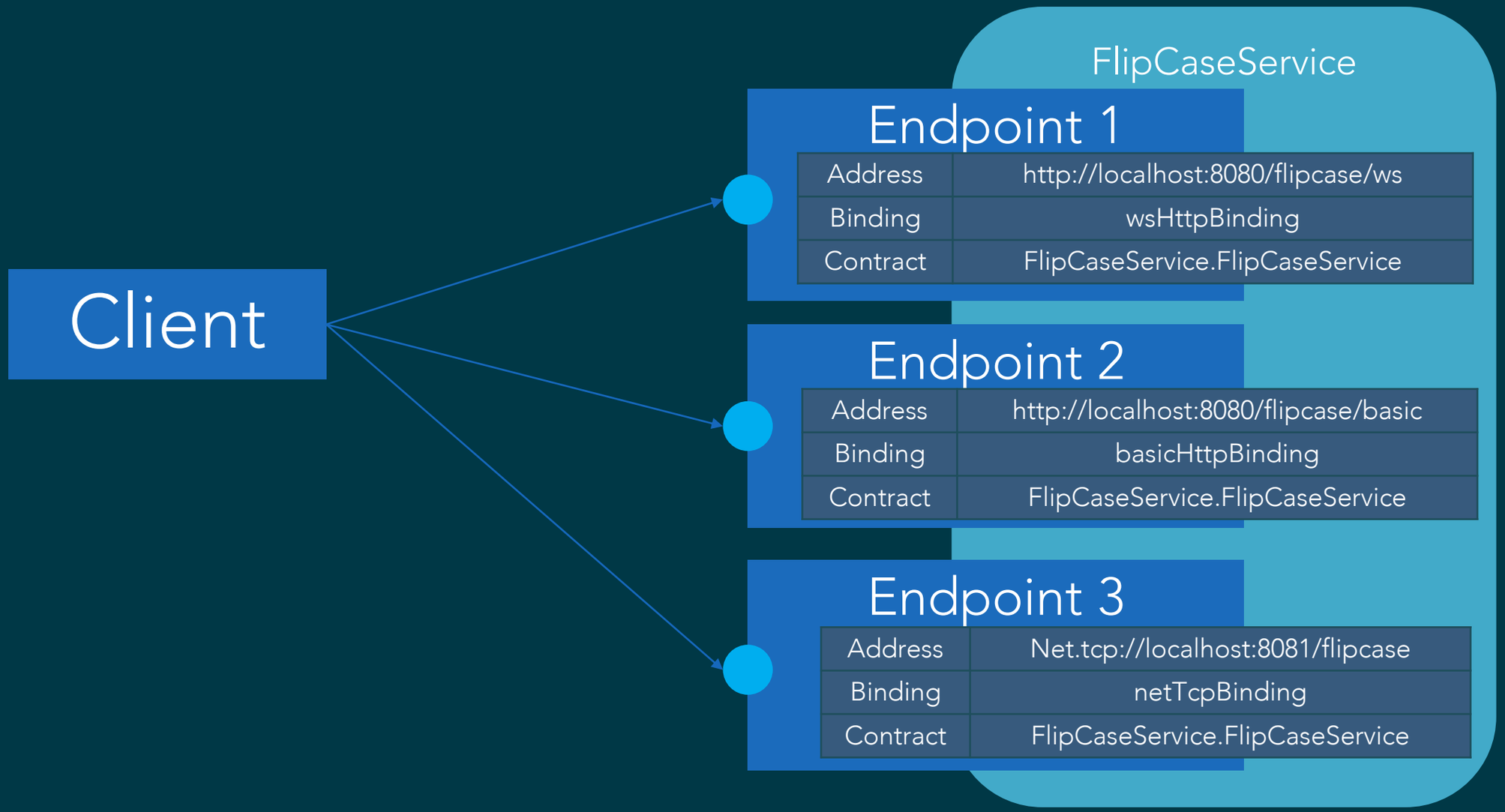
WHY USE INTERFACE?

- Interfaces does not change with implementation
- Interfaces can be deployed as separate assembly and shared
- Interface is a minimal amount of knowledge required to use a service

CONFIGURATION OF ENDPOINTS

- Endpoint
  - Address of service
  - Binding of service (e.g., protocol)
  - Contract of service
  - Behavior of service (e.g., RPC style)

MULTIPLE ENDPOINTS OF A SERVICE



EXEMPLARY CONFIGURATION (APP.CONFIG/WEB.CONFIG)

```
<system.serviceModel>
   <behaviors>
     <serviceBehaviors>
       <behavior>
         <!–
            To avoid disclosing metadata information, set the values below to false before deployment
         -->
         <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
         <!– To receive exception details in faults for debugging purposes, set the value below to
             true.  Set to false before deployment to avoid disclosing exception information -->
         <serviceDebug includeExceptionDetailInFaults="false"/>
       </behavior>
     </serviceBehaviors>
   </behaviors>
   <protocolMapping>
       <add binding="basicHttpsBinding" scheme="https" />
   </protocolMapping>
   <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />
</system.serviceModel>
```

WS-STANDARDS OVERVIEW

- **WS-*** — standards associated with **big web services** that extend the basic WS protocol stack, including:
  - *WS-Security* — specifies how **integrity** and **confidentiality** can be enforced on messages.
  - *WS-ReliableMessaging* — allows SOAP messages to be **reliably** delivered between distributed applications in the presence of software component, system, or network failures.
  - *WS-Addressing* — a standardized way of including message routing data within SOAP headers.
  - *WS-Transaction* — defines protocols to achieve **transactional** behavior

Interoperability Issues

Basic Profile 1.1 · WS-I · Final Specification

▲ **Basic Profile** – The Basic Profile 1.1 provides implementation guidelines for how related set of non-proprietary Web Service specifications should be used together for best interoperability.

Basic Profile 1.2 · WS-I · Working Group Draft

▲ **Basic Profile** – The Basic Profile 1.2 builds on Basic Profile 1.1 by incorporating Basic Profile 1.1 errata, requirements from Simple SOAP Binding Profile 1.0, and adding support for WS-Addressing and MTOM.

Basic Profile 2.0 · WS-I · Working Group Draft

▲ **Basic Profile** – The Basic Profile 2.0 is an update of WS-I BP that includes a profile of SOAP 1.2.

Attachments Profile 1.0 · WS-I · Final Specification

▲ **Attachments Profile** – The Attachment Profile 1.0 complements the Basic Profile 1.1 to add support for interoperable SOAP Messages with attachments-based Web Services.

Simple SOAP Binding Profile 1.0 · WS-I · Final Specification

▲ **Simple SOAP Binding Profile** – The Simple SOAP Binding Profile consists of those Basic Profile 1.0 requirements related to the serialization of the envelope and its representation in the message.

Basic Security Profile 1.0 · WS-I · Board Approval Draft

▲ **Basic Security Profile** defines the WS-I Basic Security Profile 1.0, based on a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

REL Token Profile 1.0 · WS-I · Working Group Draft

▲ **REL Token Profile** is based on a non-proprietary Web services specification, along with clarifications and amendments to that specification which promote interoperability.

SAML Token Profile 1.0 · WS-I · Working Group Draft

▲ **SAML Token Profile** is based on a non-proprietary Web services specification, along with clarifications and amendments to that specification which promote interoperability.

Conformance Claim Attachment Mechanism (CCAM) 1.0 · WS-I · Final Specification

▲ **Conformance Claim Attachment Mechanism (CCAM)** catalogues mechanisms that can be used to attach WS-I Profile Conformance Claims to Web services artefacts (e.g., WSDL descriptions, UDDI registries).

Reliable Asynchronous Messaging Profile (RAMP) 1.0 · WS-I · Working Draft

▲ **Reliable Asynchronous Messaging Profile (RAMP)** is a profile, in the fashion of the WS-I profiles, that enables, among other things, basic B2B integration scenarios using Web services technologies.

Standards Bodies

OASIS The **Organization for the Advancement of Structured Information Standards (OASIS)** is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. The consortium produces more Web services standards than any other organization along with standards for security, e-business, and standardization efforts in the public sector and for application-specific markets. Founded in 1993, OASIS has more than 4,000 participants representing over 600 organizations and individual members in 100 countries.

W3C The **World Wide Web Consortium (W3C)** was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has over 350 Member organizations from all over the world and has earned international recognition for its contributions to the growth of the Web. W3C is designing the infrastructure, and defining the architecture and the core technologies for Web services. In September 2000, W3C started the XML Protocol Activity to address the need for an XML-based protocol for application-to-application messaging. In January 2002, the Web Services Activity was launched, subsuming the XML Protocol Activity and extending its scope.

WS-I The **Web Services Interoperability Organization (WS-I)** is an open industry organization chartered to promote Web services interoperability across platforms, operating systems and programming languages. The organization's diverse community of Web services leaders helps customers to develop interoperable Web services by providing guidance, recommended practices and supporting resources. Specifically, WS-I creates, promotes and supports generic protocols for the interoperable exchange of messages between Web services.

IETF The **Internet Engineering Task Force (IETF)** is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

Business Process Specifications

Business Process Execution Language for Web Services 1.1 (BPEL4WS) · 1.1 · BEA Systems, IBM, Microsoft, SAP, Siebel Systems · OASIS-Standard

▲ **Business Process Execution Language for Web Services 1.1** (BPEL4WS) provides a language for the formal specification of business processes and business interaction protocols using Web Services.

WS-Choreography Model Overview 1.0 · W3C · Working Draft

▲ **WS-Choreography Model Overview** defines the format and structure of the (SOAP) messages that are exchanged, and the sequence and conditions in which the messages are exchanged.

Web Service Choreography Interface (WSCI) · 1.0 · W3C · Sun Microsystems, SAP, BEA Systems and Intalio · Note

▲ **Web Service Choreography Interface** (WSCI) describes how Web Service operations can be choreographed in the context of a message exchange in which the Web Service participates.

Web Service Choreography Description Language (CDL4WS) · 1.0 · W3C · Candidate Recommendation

▲ **Web Service Choreography Description Language** (CDL4WS) is to specify a declarative, XML based language that defines from a global viewpoint the common and complementary observable behaviour, where message exchanges occur, and when the jointly agreed ordering rules are satisfied.

Business Process Execution Language for Web Services 2.0 (BPEL4WS) · 2.0 · OASIS, BEA Systems, IBM, Microsoft, SAP, Siebel Systems · Committee Draft

▲ **Business Process Execution Language for Web Services 2.0** (BPEL4WS) provides a language for the formal specification of business processes and business interaction protocols using Web Services.

Business Process Management Language (BPML) 1.1 · BPMI.org · Final Draft

▲ **Business Process Management Language** (BPML) provides a meta-language for expressing business processes and supporting entities.

XML Process Definition Language (XPDL) 2.0 · Final

▲ **XML Process Definition Language** (XPDL) provides an XML file format that can be used to interchange process models between tools.

Management Specifications

Management Using Web Services (WSDM-MUWS) 1.0 · OASIS · OASIS-Standard

▲ **Web Service Distributed Management:** Management Using Web Services (WSDM-MUWS) defines how an IT resource connected to a network provides manageability interfaces such that the IT resource can be managed locally and from remote locations using Web services technologies.

Management Of Web Services (WSDM-MOWS) 1.0 · OASIS · OASIS-Standard

▲ **Web Service Distributed Management:** Management Of Web Services (WSDM-MOWS) addresses management of the components that form the network, the Web services endpoints, using Web services protocols.

WS-Management · AMD, Dell, Intel, Microsoft and Sun Microsystems · Published Specification

▲ **WS-Management** describes a general SOAP-based protocol for managing systems such as PCs, servers, devices, Web services and other applications, and other manageable entities.

Service Modeling Language · IBM, BEA, BMC, Cisco, Dell, HP, Intel, Microsoft, Sun · Draft Specification

▲ **Service Modeling Language** (SML) is used to model complex IT services and systems, including their structure, constraints, policies, and best practices.

Presentation Specifications

Web Services for Remote Portlets (WSRP) 2.0 · OASIS · Committee Draft

▶ **Web Services for Remote Portlets** (WSRP) defines a set of interfaces and related semantics which standardize interactions with components providing user-facing markup, including the processing of user interactions with that markup.

Metadata Specifications

WS-Policy 1.5 · W3C · Working Draft

▲ **WS-Policy** describes the capabilities and constraints of the policies on intermediaries and endpoints (e.g. business rules, required security tokens, supported encryption algorithms, privacy rules).

WS-PolicyAssertions 1.1 · BEA Systems, IBM, Microsoft, SAP · Public Draft

▲ **WS-PolicyAssertions** provides an initial set of assertions to address some common needs of Web Services applications.

WS-PolicyAttachment 1.2 · W3C · W3C Member Submission

▲ **WS-PolicyAttachment** defines two general-purpose mechanisms for associating policies with the subjects to which they apply; the policies may be defined as part of existing metadata about the subject or the policies may be defined independently and associated through an external binding to the subject.

WS-Discovery · Microsoft, BEA Systems, Canon, Intel and webMethods · Draft

▲ **WS-Discovery** defines a multicast discovery protocol for dynamic discovery of services on ad-hoc and managed networks.

WS-MetadataExchange 1.1 · BEA Systems, Computer Associates, IBM, Microsoft, SAP, Sun Microsystems and webMethods · Public Draft

▲ **WS-MetadataExchange** enables a service to provide metadata to others through a Web services interface. Given only a reference to a Web service, a user can access a set of WSDL/SOAP operations to retrieve the metadata that describes the service.

Universal Description, Discovery and Integration (UDDI) 3.0.2 · OASIS · OASIS-Standard

▲ **Universal Description, Discovery and Integration (UDDI)** defines a set of services supporting the description and discovery of businesses, organizations, and other Web services providers, the Web services they make available, and the technical interfaces which may be used to access those services.

Web Service Description Language 2.0 SOAP Binding 2.0 · W3C - Working Draft

▲ **Web Service Description Language SOAP Binding** describes the concrete details for using WSDL 2.0 in conjunction with SOAP 1.1 protocol.

Web Service Description Language 2.0 Core 2.0 · W3C · Candidate Recommendation

▲ **Web Service Description Language 2.0 Core** is an XML-based language for describing Web services and how to access them. It specifies the location of the service and the operations (or methods) the service exposes.

Web Service Description Language 1.1 1.1 · W3C · Note

▲ **Web Service Description Language** 1.1 is an XML-based language for describing Web services and how to access them. It specifies the location of the service and the operations (or methods) the service exposes.

Reliability Specifications

WS-ReliableMessaging 1.1 · OASIS · Committee Draft

▲ **WS-ReliableMessaging** describes a protocol that allows Web services to communicate reliable in the presence of software component, system, or network failures. It defines a SOAP binding that is required for interoperability.

WS-Reliable Messaging Policy Assertion (WS-RM Policy) 1.1 · OASIS · Committee Draft

▲ **Web Services ReliableMessaging Policy Assertion** (WS-RM Policy) describes a domain-specific policy assertion for WS-ReliableMessaging that that can be specified within a policy alternative as defined in WS-Policy framework.

WS-Reliability 1.1 · OASIS · OASIS-Standard

▲ **WS-Reliability** is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering. WS-Reliability is defined as SOAP header extensions and is independent of the underlying protocol. This specification contains a binding to HTTP.

Security Specifications

WS-Security 1.1 · OASIS · OASIS-Standard

▲ **WS-Security** is a communications protocol providing a means for applying security to Web Services.

WS-SecurityPolicy 1.1 · IBM, Microsoft, RSA Security, VeriSign · Public Draft

▲ **WS-SecurityPolicy** defines how to describe policies related to various features defined in the WS-Security specification.

WS-Security: SOAP Message Security 1.1 · OASIS · Public Review Draft

▲ **WS-Security:** SOAP Message Security describes enhancements to SOAP messaging to provide message integrity and confidentiality. Specifically, this specification provides support for multiple security token formats, trust domains, signature formats and encryption technologies. The token formats and semantics for using these are defined in the associated profile documents.

WS-Security: Username Token Profile 1.1 · OASIS · Public Review Draft

▲ **WS-Security: Username Token Profile** describes how a Web Service consumer can supply a Username Token as a means of identifying the requestor by username, and optionally using a password (or shared secret, etc) to authenticate that identity to the Web Service producer.

WS-Security: Kerberos Binding 1.0 · Microsoft, IBM, OASIS · Working Draft

▲ **WS-Security: Kerberos Binding** defines how to encode Kerberos tickets and attach them to SOAP messages. As well, it specifies how to add signatures and encryption to the SOAP message, in accordance with WS-Security, which uses and references the Kerberos tokens.

WS-Federation 1.0 · IBM, Microsoft, BEA Systems, RSA Security, and VeriSign · Initial Draft

▲ **WS-Federation** describes how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.

WS-Security: SAML Token Profile 1.1 · OASIS · Public Review Draft

▲ **WS-Security: SAML Token Profile** defines the use of Security Assertion Markup Language (SAML) v1.1 assertions in the context of WSS: SOAP Message Security including for the purpose of securing SOAP messages and SOAP message exchanges.

WS-Trust · BEA Systems, Computer Associates, IBM, Layer 7 Technologies, Microsoft, Netegrity, Oblix, OpenNetwork, Ping Identity Corporation, Reactivity, RSA Security, VeriSign and Westbridge Technology · Initial Draft

▲ **WS-Trust** describes a framework for trust models that enables Web Services to securely interoperate. It uses WS-Security base mechanisms and defines additional primitives and extensions for security token exchange to enable the issuance and dissemination of credentials within different trust domains.

WS-Security: X.509 Certificate Token Profile 1.1 · OASIS · Public Review Draft

▲ **WS-Security: X.509 Certificate Token Profile** describes the use of the X.509 authentication framework with the WS-Security: SOAP Message Security specification.

WS-SecureConversation · BEA Systems, Computer Associates, IBM, Layer 7 Technologies, Microsoft, Netegrity, Oblix, OpenNetwork, Ping Identity Corporation, Reactivity, RSA Security, VeriSign and Westbridge Technology · Public Draft

▲ **WS-SecureConversation** specifies how to manage and authenticate message exchanges between parties including security context exchange and establishing and deriving session keys.

Transaction Specifications

WS-Coordination 1.1 · OASIS · Working Draft

▲ **WS-Coordination** describes an extensible framework for providing protocols that coordinate the actions of distributed applications.

WS-Business Activity 1.1 · OASIS · Working Draft

▲ **WS-Business Activity** provides the definition of the business activity coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification.

WS-Atomic Transaction 1.1 · OASIS · Committee Draft

▲ **WS-Atomic Transaction** defines protocols that enable existing transaction processing systems to wrap their proprietary protocols and interoperate across different hardware and software vendors.

WS-Composite Application Framework (WS-CAF) 1.0 · Arjuna Technologies, Fujitsu, IONA, Oracle and Sun Microsystems · Committee Specification

▲ **WS-Composite Application Framework** (WS-CAF) is a collection of three specifications aimed at solving problems that arise when multiple Web Services are used in combination. It proposes standard, interoperable mechanisms for managing shared context and ensuring business processes achieve predictable results and recovery from failure.

WS-Context (WS-CTX) 1.0 · Arjuna Technologies, Fujitsu, IONA, Oracle and Sun Microsystems · Committee Draft

▲ **WS-Context** (WS-CTX) is intended as a lightweight mechanism for allowing multiple Web Services to share a common context.

WS-Coordination Framework (WS-CF) 1.0 · Arjuna Technologies, Fujitsu, IONA, Oracle and Sun Microsystems · Committee Draft

▲ **WS-Coordination Framework** (WS-CF) allows the management and coordination in a Web Services interaction of a number of activities related to an overall application.

WS-Transaction Management (WS-TXM) 1.0 · Arjuna Technologies, Fujitsu, IONA, Oracle and Sun Microsystems · Committee Draft

▲ **WS-Transaction Management** (WS-TXM) defines a core infrastructure service consisting of a Transaction Service for Web Services.

Resource Specifications

Web Services Resource Framework (WSRF) 1.2 · OASIS · OASIS-Standard

▲ **Web Services Resource Framework** (WSRF) defines a family of specifications for accessing stateful resources using Web Services.

WS-BaseFaults (WSRF) 1.2 · OASIS · Working Draft

▲ **WS-BaseFaults** (WSRF) defines a base set of information that may appear in fault messages. WS-BaseFaults defines an XML schema type for base faults, along with rules for how this base fault type is used and extended by Web Services.

WS-ServiceGroup (WSRF) 1.2 · OASIS · Working Draft

▲ **WS-ServiceGroup** (WSRF) defines a means by which Web Services and WS-Resources can be aggregated or grouped together for a domain specific purpose.

WS-ResourceProperties 1.2 · OASIS · Working Draft

▲ **WS-ResourceProperties** specifies the means by which the definition of the properties of a WS-Resource may be declared as part of the Web Service interface. The declaration of the WS-Resource properties represents a projection of or a view on the WS-Resource state.

WS-ResourceLifetime 1.2 · OASIS · Working Draft

▲ **WS-ResourceLifetime** is to standardize the terminology, concepts, message exchanges, WSDL and XML needed to monitor the lifetime of, and destroy WS-Resources. Additionally, it defines resource properties that may be used to inspect and monitor the lifetime of a WS-Resource.

WS-Transfer · W3C · W3C Member Submission

▲ **WS-Transfer** describes a general SOAP-based protocol for accessing XML representations of Web service-based resources.

Resource Representation SOAP Header Block (RRSHB) · W3C · Recommendation

▲ **Resource Representation SOAP Header Block** (RRSHB) complements MTOM by defining mechanisms for describing and conveying non-XML resource representations in a SOAP 1.2 message.

Messaging Specifications

WS-Notification 1.3 · OASIS · OASIS-Standard

▶ **WS-Notification** is a family of related white papers and specifications that define a standard Web services approach to notification using a topic-based publish/subscribe pattern.

WS-BrokeredNotification 1.3 · OASIS · OASIS-Standard

▶ **WS-BrokeredNotification** defines the interface for the NotificationBroker. A NotificationBroker is an intermediary, which, among other things, allows publication of messages from entities that are not themselves service providers.

WS-BaseNotification 1.3 · OASIS · OASIS-Standard

▶ **WS-BaseNotification** standardizes the terminology, concepts, operations, WSDL and XML needed to express the basic roles involved in Web services publish and subscribe for notification message exchange.

WS-Eventing · W3C · Public Draft

▶ **WS-Eventing** defines a baseline set of operations that allow Web services to provide asynchronous notifications to interested parties.

WS-Addressing – Core 1.0 · W3C · Recommendation

▶ **WS-Addressing – Core** provides transport-neutral mechanisms to address Web services and messages. This specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

WS-Enumeration · Systinet, Microsoft, Sonic Software, BEA Systems and Computer Associates · Public Draft

▶ **WS-Enumeration** describes a general SOAP-based protocol for enumerating a sequence of XML elements that is suitable for traversing logs, message queues, or other linear information models.

WS-Topics 1.3 · OASIS · OASIS-Standard

▶ **WS-Topics** defines three topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system.

WS-Addressing – WSDL Binding 1.0 · W3C · Candidate Recommendation

▶ **WS-Addressing – WSDL Binding** defines how the abstract properties defined in Web Services Addressing – Core are described using WSDL.

WS-Addressing – SOAP Binding 1.0 · W3C · Recommendation

▶ **WS-Addressing – SOAP Binding** provides transport-neutral mechanisms to address Web services and messages.

SOAP

SOAP 1.2 · W3C · Recommendation

SOAP Message Transmission Optimization Mechanism (MTOM) 1.0 · W3C · Recommendation

▶ **SOAP Message Transmission Optimization Mechanism** describes an abstract feature for optimizing the transmission and/or wire format of a SOAP message.

SOAP 1.1 · W3C · Note

▶ SOAP is a lightweight, XML-based protocol for exchange of information in a decentralized, distributed environment.

XML Specifications

XML 1.1 1.1 · W3C · Recommendation

▶ **XML** – Extensible Markup Language is a pared-down version of SGML, designed especially for Web documents. It allows one to create own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

XML 1.0 1.0 · W3C · Recommendation

▶ **XML** – Extensible Markup Language is a pared-down version of SGML, designed especially for Web documents. It allows one to create own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

Namespaces in XML 1.1 · W3C · Recommendation

▶ **Namespaces in XML** provides a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by IRI references.

XML Information Set 1.0 · W3C · Recommendation

▶ **XML Information Set** is an abstract data set to provide a consistent set of definitions for use in other specifications that need to refer to the information in a well-formed XML document.

XML Schema 1.1 · W3C · Working Draft

▶ **XML Schema** – XML Schema Definition Language is an XML language for describing and constraining the content of XML documents.

XML binary Optimized Packaging (XOP) 1.0 · W3C · Recommendation

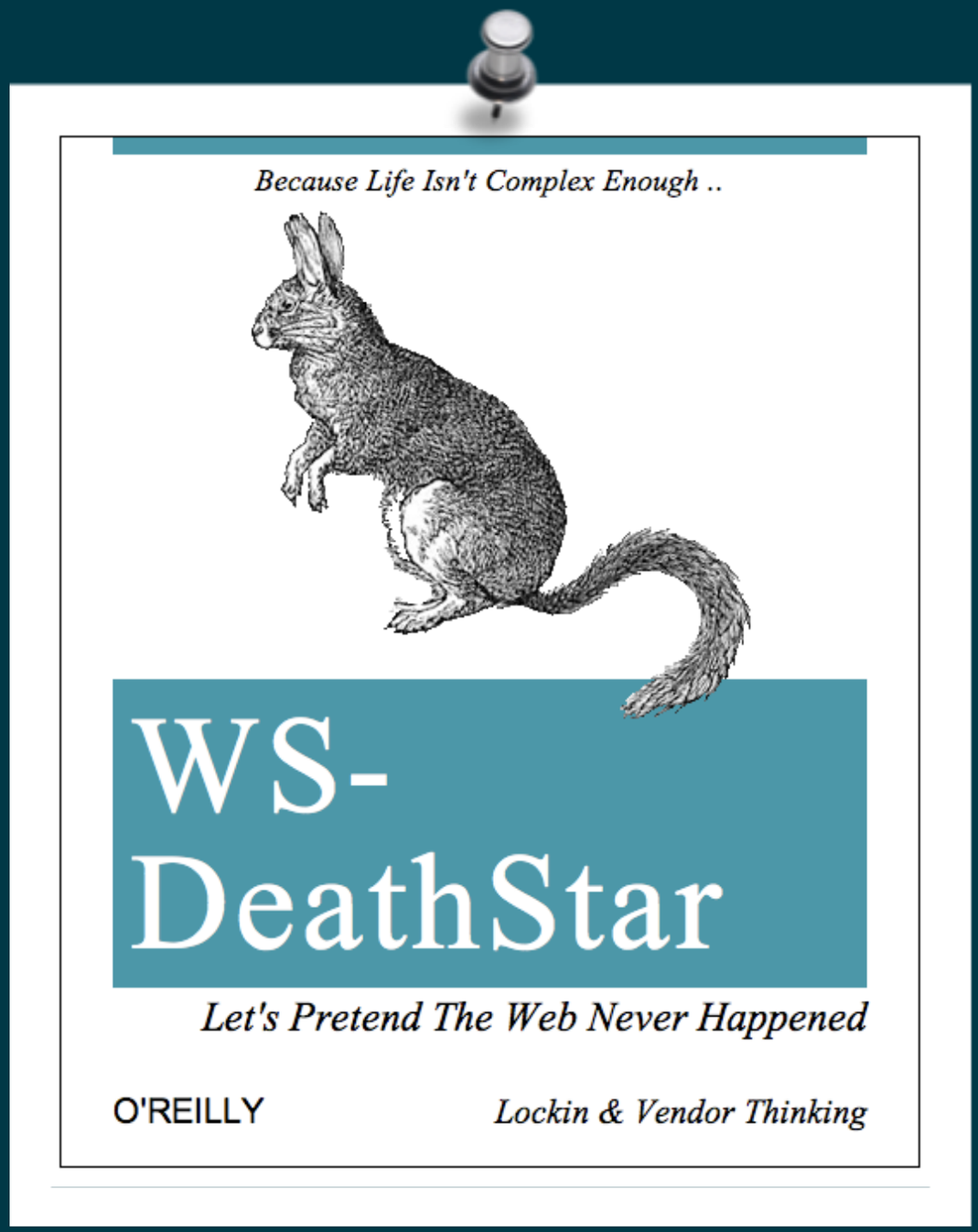
▶ **XML binary Optimized Packaging** (XOP) is an XML language for describing and constraining the content of XML documents.

Describing Media Content of Binary Data in XML (DMCBDX) · W3C · Note

▶ **Describing Media Content of Binary Data in XML** (DMCBDX) specifies how to indicate the content-type associated with binary element content in an XML document and to specify, in XML Schema, the expected content-type(s) associated with binary element content.

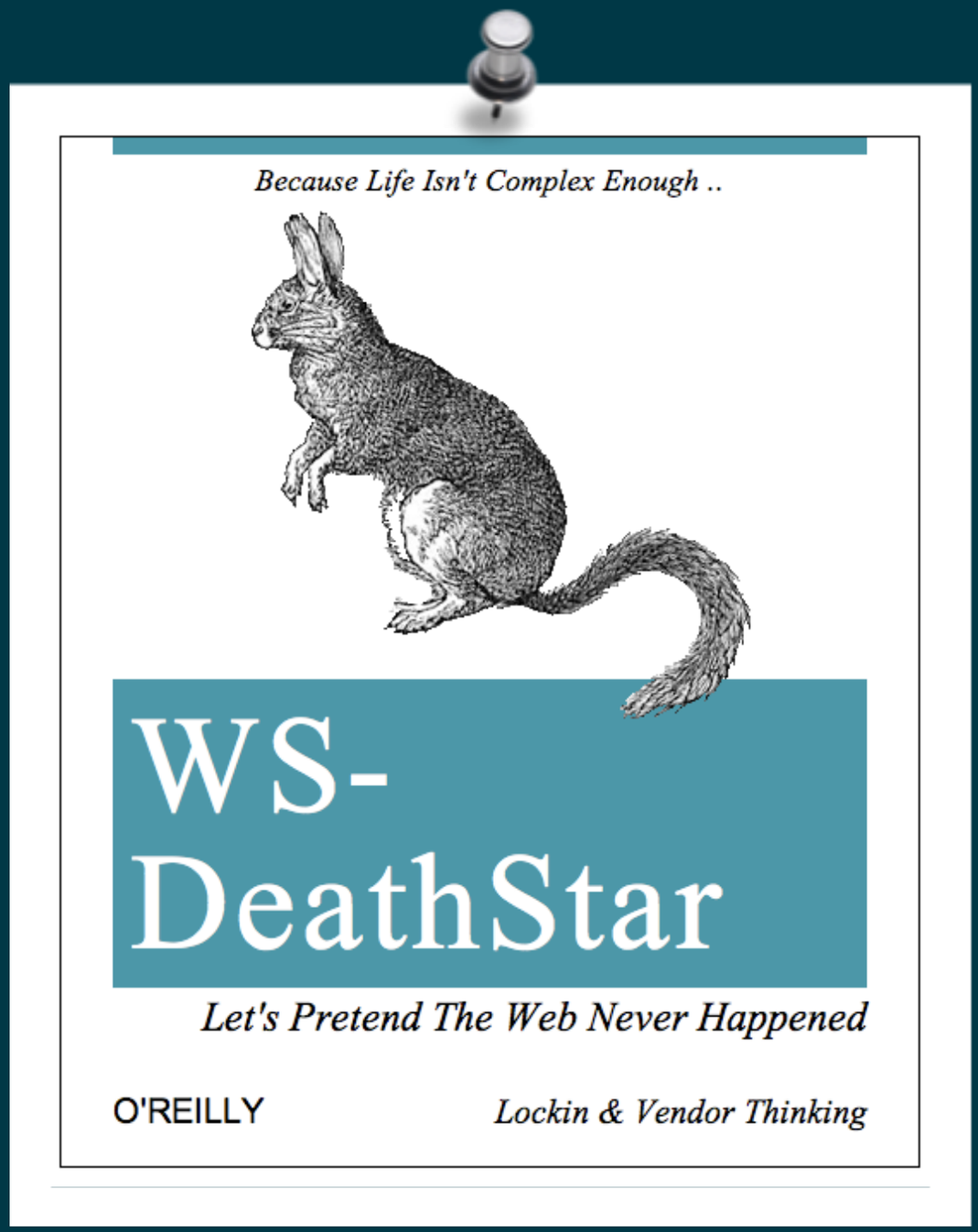
WS-* CRITICISM

- SOAP
  - verbosity
  - overhead
- Fragility
- Complexity



WS-* CRITICISM

- SOAP
  - verbosity
  - overhead
- Fragility
- Complexity



- But.. WS-* are still heavily used in the B2B integration.

PRESENTATION OUTLINE

- Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - **REST — Representational State Transfer**
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

WHY THE WEB IS SO SUCCESSFUL?

- Why is the Web such a **successful** information-sharing platform?
- How has it grown from a simple network of researchers and academics to an interconnected **worldwide community**?
- What makes the Web **scale**?

WHY THE WEB IS SO SUCCESSFUL?

- Why is the Web such a **successful** information-sharing platform?
- How has it grown from a simple network of researchers and academics to an interconnected **worldwide community**?
- What makes the Web **scale**?
- Can we follow the same **principles** driving the **human-centric** Web for computer-to-computer scenarios?

WHY THE WEB IS SO SUCCESSFUL?

- Why is the Web such a **successful** information-sharing platform?
- How has it grown from a simple network of researchers and academics to an interconnected **worldwide community**?
- What makes the Web **scale**?
- Can we follow the same **principles** driving the **human-centric** Web for computer-to-computer scenarios?

Architectural Styles and the Design of Network-based Software Architectures, **Roy Fielding**, 2000
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

ARCHITECTURE OF THE WEB

- Web architecture is based on three *technologies* **URL**, **HTTP**, and **HTML** — the Fielding dissertation explains the **principles** behind the design of these *technologies*.

- Underlying the three web *technologies* are two essential concepts: **resources** and **representations**.

- Understanding web *technologies* on a deep level is the key to understanding the **REST constraints**, how those constraints drive the success of the Web, and how you can exploit those constraints in your own APIs.

THINKING IN RESOURCES

- Resources are the fundamental building blocks of web-based systems, to the extent that the Web is often referred to as being **resource-oriented**.

- A resource is anything we expose to the Web, from a document or video clip to a business processor device.

- A resource is anything with which a consumer **interacts** while progressing toward some goal.

- To use a resource we need to:
  - identify it on the network
  - have some means of manipulating it

THINKING IN RESOURCES

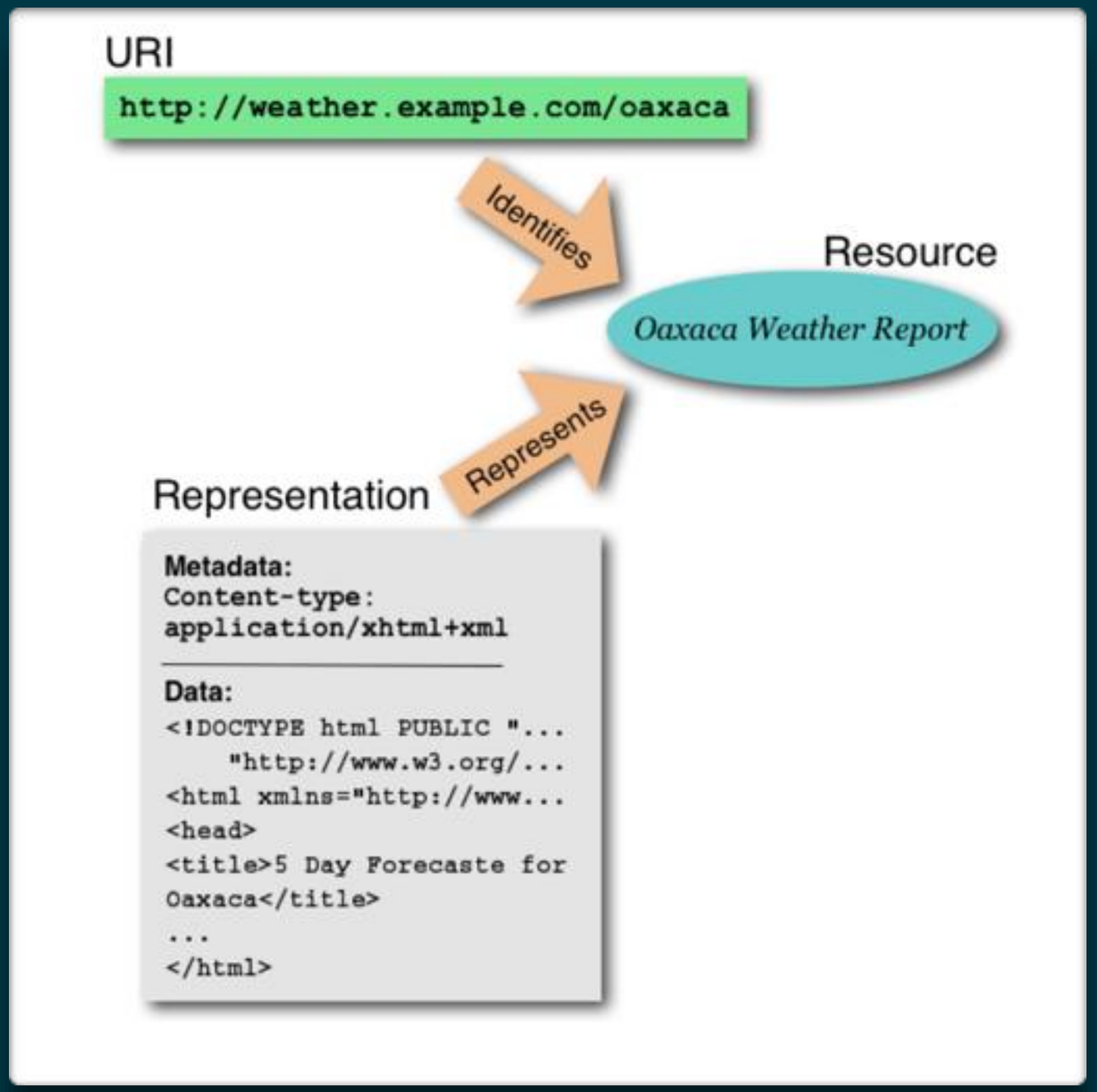
- Resources are the fundamental building blocks of web-based systems, to the extent that the Web is often referred to as being **resource-oriented**.

- A resource is anything we expose to the Web, from a document or video clip to a business processor device.

- A resource is anything with which a consumer **interacts** while progressing toward some goal.

- To use a resource we need to:
  - identify it on the network — the Web provides URI
  - have some means of manipulating it — the Web employs HTTP

THINKING IN RESOURCES



ARCHITECTURE OF THE WORLD WIDE WEB, VOLUME ONE

HTTP://WWW.W3.ORG/TR/WEBARCH/

RESOURCE REPRESENTATION

- A **representation** is a view on the **state** of the resource at an instant in time, e.g. JSON, PDF.

- A resource can be represented in **multiple formats**, defined by a media types; client and server employ **content negotiation** to agree on the transfer format.

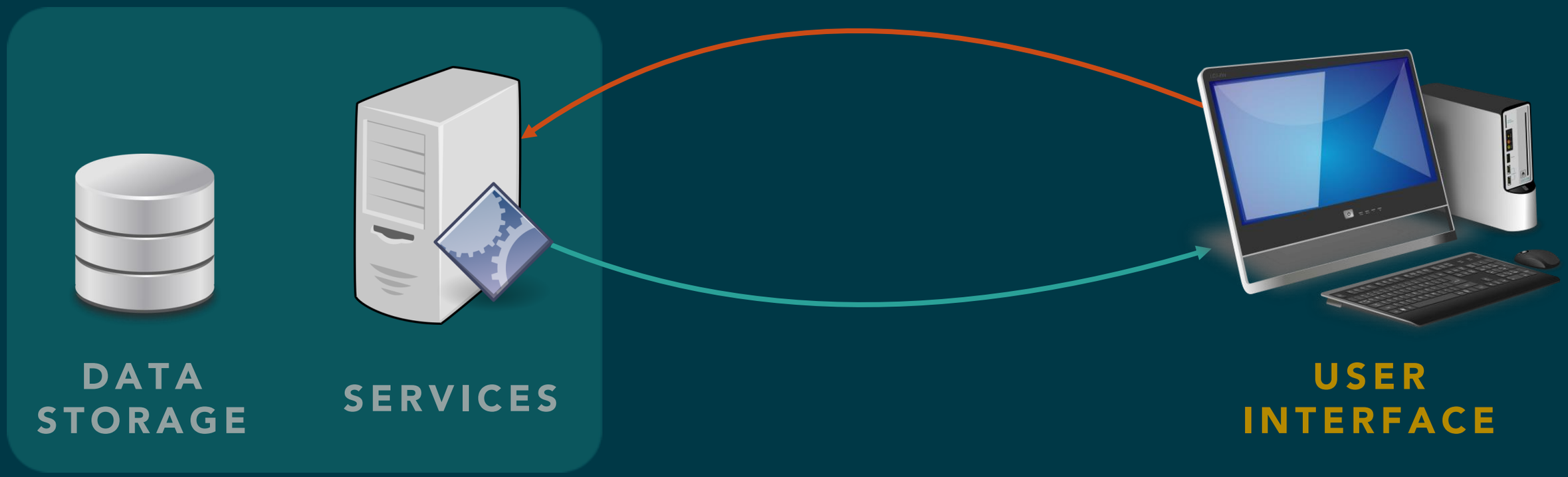
- Clients and servers exchange representations:
  - **GET** — retrieves a representation of the current state of the addressed resource; the client never sees a resource directly.
  - **POST** — passes a representation of the resource to the server so that the underlying resource's state can change.

REPRESENTATIONAL **S**TATE **T**RANSFER

REST is a set of architectural principles (**design constraints**) describing how to build distributed systems (e.g. web apps) to achieve **scalability** and **reliability**:

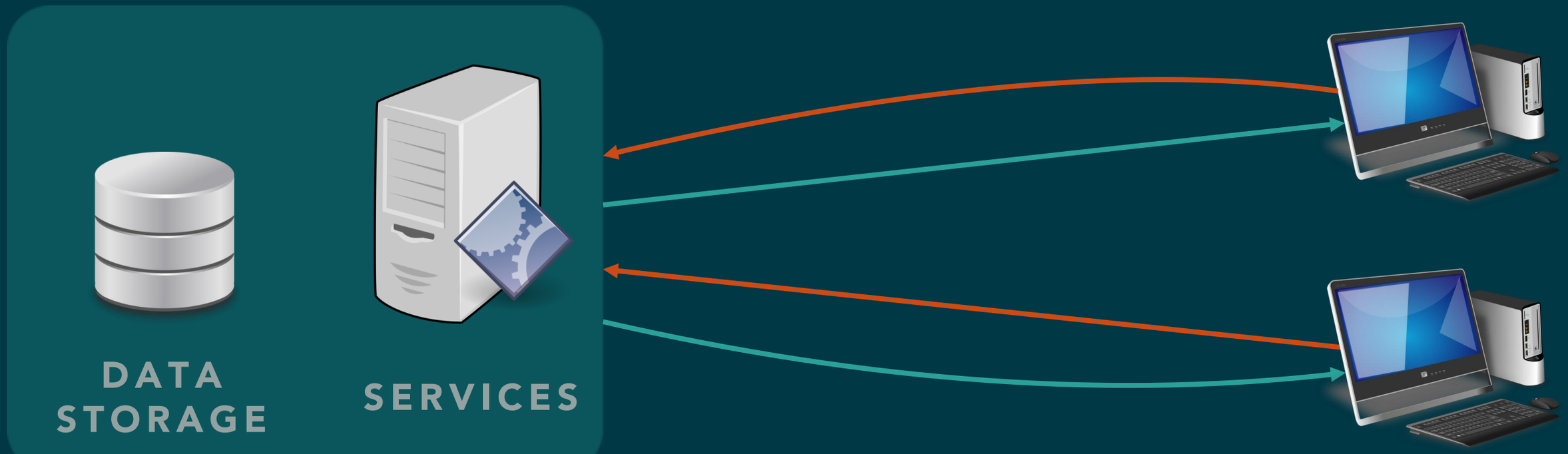
1. Client-Server
2. Statelessness
3. Caching
4. Uniform Interface:
    I. Identification of resources — addressability
    II. Manipulation of resources through representations
    III. Self-descriptive messages
    IV. Hypermedia As The Engine Of Application State — HATEOAS
5. Layered System
6. Code-On-Demand

REST ARCHITECTURAL PRINCIPLES
1. CLIENT-SERVER



- Immediate **separation of concerns**:
  - **Simplifying** the server component in order to improve scalability.
  - **Moving** all of the **user interface** functionality to the client.
  - **Independent** development and evolution of components.

REST ARCHITECTURAL PRINCIPLES
2. STATELESSNESS

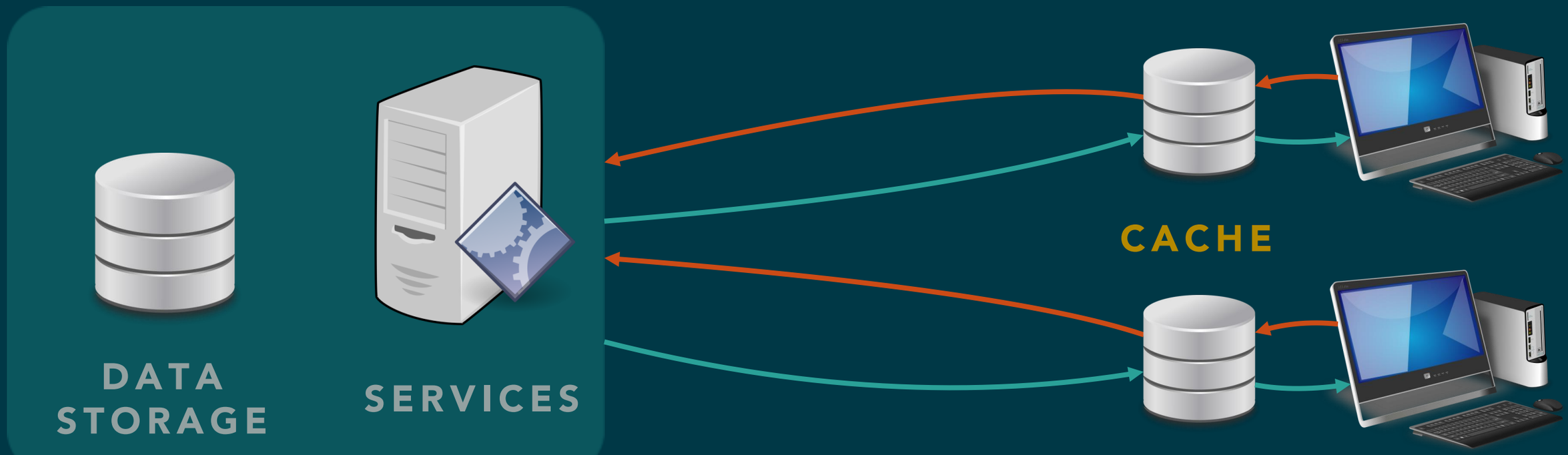


STATE OF RESOURCES

STATE OF APPLICATION

- Each request must be **context-free** and **self-contained**
- **Increased**:
  - **Visibility** – work to be done can be determined based on a single request
  - **Reliability** – it eases the task of recovering from partial failures
  - **Scalability** – no session means that the server may free up once a request is responded
- **Decreased** network performance by sending repetitive data

REST ARCHITECTURAL PRINCIPLES
3. CACHING



- Server's responses should be labeled as **cacheable** or **non-cacheable**.
- If a response is cacheable, then a client cache is given the right to **reuse** that response data for later, equivalent requests.
- Caching may **increase** performance but also introduces the standard complexities associated with proper cache **invalidation**.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: RESOURCES

I. **Addressability** — every resource in the system is reachable through a **unique** identifier
   - the Web provides addressability by using URI.

```
scheme://host[:port]/path/…/[?query-string][#anchor]
```

   - unique URIs make the available resources **linkable**.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: RESOURCES

II. **Manipulation of resources through representations** — clients and servers manipulate resources by sending representations back and forth using HTTP methods

- HTTP has a fixed number of **methods** with well-defined semantics that are sufficient to meet the requirements of most distributed applications:
  - **GET**
  - **PUT**
  - **DELETE**
  - **POST**
  - **PATCH**
  - **HEAD**
  - **OPTIONS**
- HTTP defines a set of **response codes** that specify semantics of the result of the requested method, e.g., 200 OK, 201 Created, 404 Not Found
- HTTP methods and response codes mean the same for all resources (universal semantics).

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: MESSAGES

III. Self-Descriptive Messages:

- **Interaction is stateless between requests** — in a stateless system, a server can handle a client's request without having to remember how it handled all that client's previous requests.
- **Standard methods and media types are used to indicate semantics and exchange information** — an HTTP response includes the `Content-Type` header to inform the client how to parse the body.
- **Responses explicitly indicate cacheability** — the server conveys caching information by adding a header to the very HTTP response that might be cached.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: HATEOAS

IV. HATEOAS
— Hypermedia As The Engine Of Application State

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: HATEOAS

IV. HATEOAS
— Hypermedia As The Engine Of Application State

1. All application state is kept on the client side. Changes to application state are the client's responsibility.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: HATEOAS

IV. HATEOAS
— Hypermedia As The Engine Of Application State

1. All application state is kept on the client side. Changes to application state are the client's responsibility.
2. The client can only change its application state by making an HTTP request and processing the response.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: HATEOAS

IV. HATEOAS
— Hypermedia As The Engine Of Application State

1. All application state is kept on the client side. Changes to application state are the client's responsibility.

2. The client can only change its application state by making an HTTP request and processing the response.

3. How does the client know which requests it can make next?
   — by looking at the hypermedia controls in the representations it has received so far.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: HATEOAS

IV. HATEOAS
— Hypermedia As The Engine Of Application State

1. All application state is kept on the client side. Changes to application state are the client's responsibility.

2. The client can only change its application state by making an HTTP request and processing the response.

3. How does the client know which requests it can make next?
— by looking at the hypermedia controls in the representations it has received so far.

4. Therefore, **hypermedia controls are the driving force behind changes in application state.**

REST ARCHITECTURAL PRINCIPLES
HATEOAS EXAMPLE

```
GET /account/12345 HTTP/1.1
```

REST ARCHITECTURAL PRINCIPLES
HATEOAS EXAMPLE

```
GET /account/12345 HTTP/1.1

HTTP/1.1 200 OK
```

```
<?xml version="1.0"?>
<account>
   <account_number>12345</account_number>
   <balance currency="usd">100.00</balance>
   <link rel="deposit" href="/account/12345/deposit"/>
   <link rel="withdraw" href="/account/12345/withdraw"/>
   <link rel="transfer" href="/account/12345/transfer"/>
   <link rel="close" href="/account/12345"/>
</account>
```

REST ARCHITECTURAL PRINCIPLES
HATEOAS EXAMPLE

```
GET /account/12345 HTTP/1.1
```

REST ARCHITECTURAL PRINCIPLES
HATEOAS EXAMPLE

```
GET /account/12345 HTTP/1.1

HTTP/1.1 200 OK
```

```
<?xml version="1.0"?>
<account>
   <account_number>12345</account_number>
   <balance currency="usd">-25.00</balance>
   <link rel="deposit" href="/account/12345/deposit"/>
</account>
```

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: HATEOAS

IV. HATEOAS:

- HATEOAS improves **discoverability**, providing a way of making the application more **self-documenting**.
- A REST client needs **no prior knowledge** about how to interact with any particular application or server beyond a generic understanding of hypermedia (in contrast to knowing the IDL)
- HATEOAS allows a client to **automatically adapt** to changes on the server side.
- HATEOAS allows a server to change its underlying implementation **without breaking** all of its **clients**.

REST ARCHITECTURAL PRINCIPLES
4. UNIFORM INTERFACE: **BENEFITS**

- **Familiarity** — no need for interface description language to describe which methods are available.
- **Interoperability** — with REST over HTTP there is usually no need to install vendor-specific libraries.
- **Scalability** — predictable behavior of interface methods can bring large performance benefits.
  - method properties (safe and idempotent)
  - caching semantics
  - statelessness

REST ARCHITECTURAL PRINCIPLES
5. LAYERED SYSTEM

- *Layered-client-server adds proxy and gateway components to the client-server style... These additional mediator components can be added in multiple layers to add features like load balancing and security checking to the system.*

- Layered systems reduce **coupling** across multiple layers by hiding the inner layers from all except the adjacent outer layer, thus improving **extensibility** and **reusability**.

- A client cannot tell whether it is connected directly to the end server, or to an intermediary along the way — **transparency**.

REST ARCHITECTURAL PRINCIPLES
6. CODE-ON-DEMAND

- Optional constraint — may be disabled within some contexts.
- Client functionality to be extended by downloading and executing code in the form of **applets** or **scripts**.
- This simplifies clients by reducing the number of features required to be pre-implemented.
- Allowing features to be downloaded after deployment improves system **extensibility** and **configurability**, and provides for better **user-perceived performance.**

PRESENTATION OUTLINE

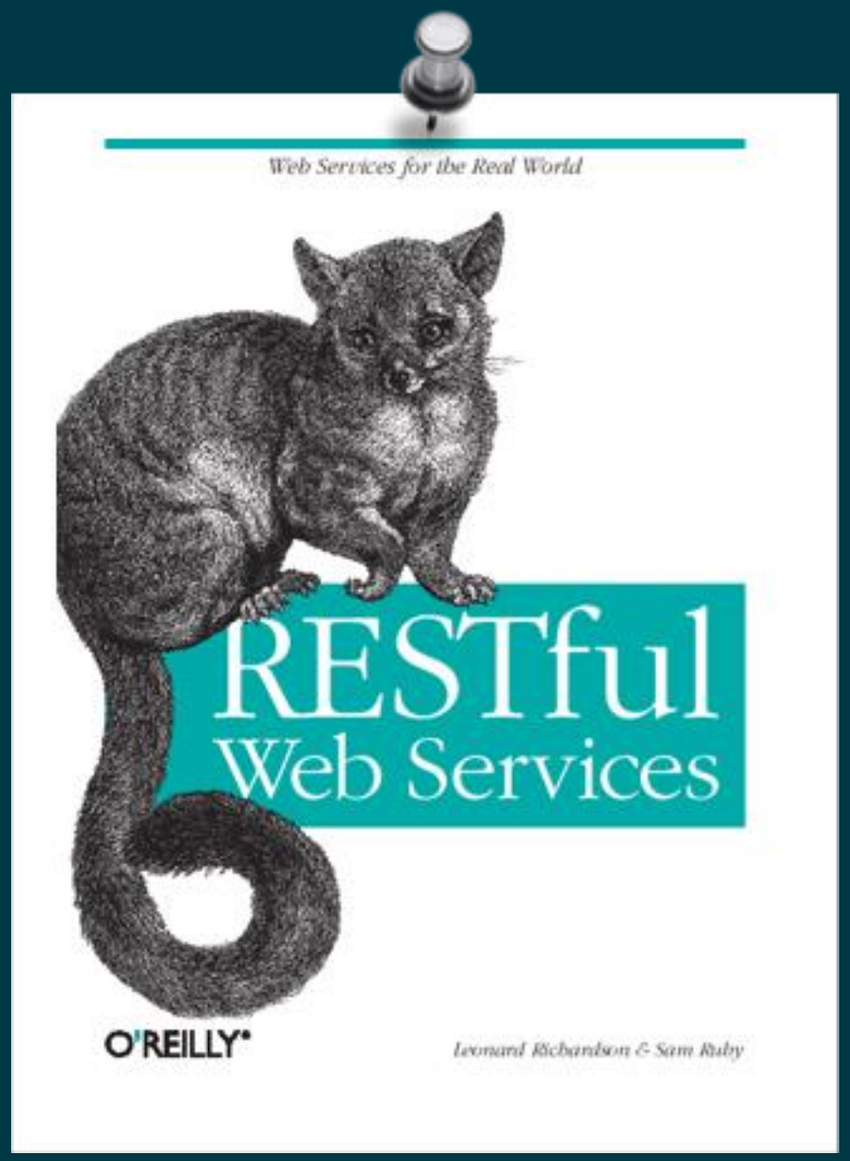
- Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - **RESTful web API in practice**
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

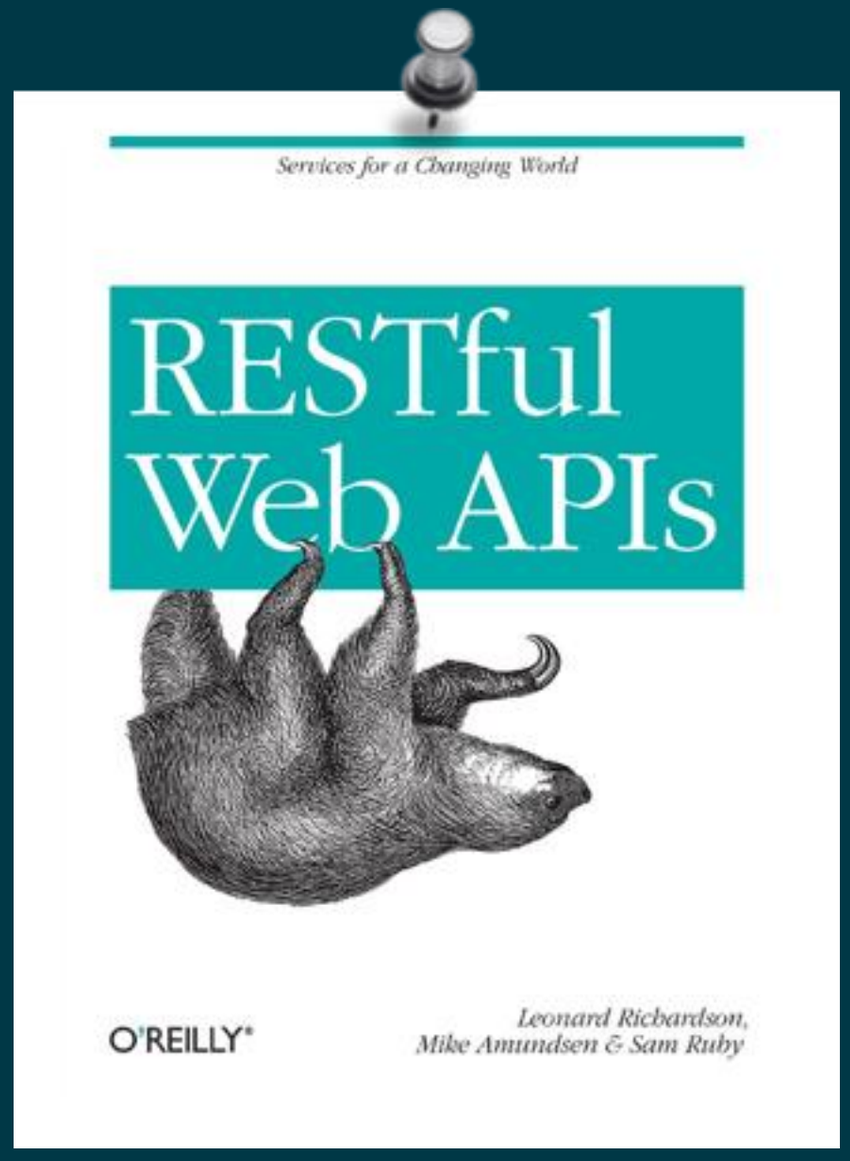
REST IN PRACTICE

- REST captures the fundamental principles of the Web.
- REST constraints provide **abstract** guidance on how modern web applications should be designed to promote their **longevity** and **independent evolution**.
- In particular, REST focuses on the **interface** between the server and the client — **RESTful web services**.
- HATEOAS, while theoretically compelling, has proved difficult to implement consistently (e.g. in JSON APIs).

RESTFUL WEB API

- **Web API** — web services with emphasis on REST principles (in contrast to SOAP-based WS).





RESTFUL WEB API GUIDELINES

*The key abstraction of information in REST is a **resource**. Any **information** that can be named can be a resource: a document or image, a temporal service (e.g. "weather in Los Angeles"), a collection of other resources, and so on.* —ROY FIELDING

1. Identify and model the resources exposed by the service:
   - name resources as **nouns** as opposed to **verbs** or actions
   - each resource has at least one **URI**, which should follow a predictable, **hierarchical** structure to enhance **understandability** and **usability**.
   - a resource can be a **singleton / instance** or a **collection**, e.g.
     - a user of the system — https://api.github.com/users/mszubert
     - repositories — https://api.github.com/users/mszubert/repos
     - repo contents — https://api.github.com/repos/mszubert/2048/contents

RESTFUL WEB API GUIDELINES

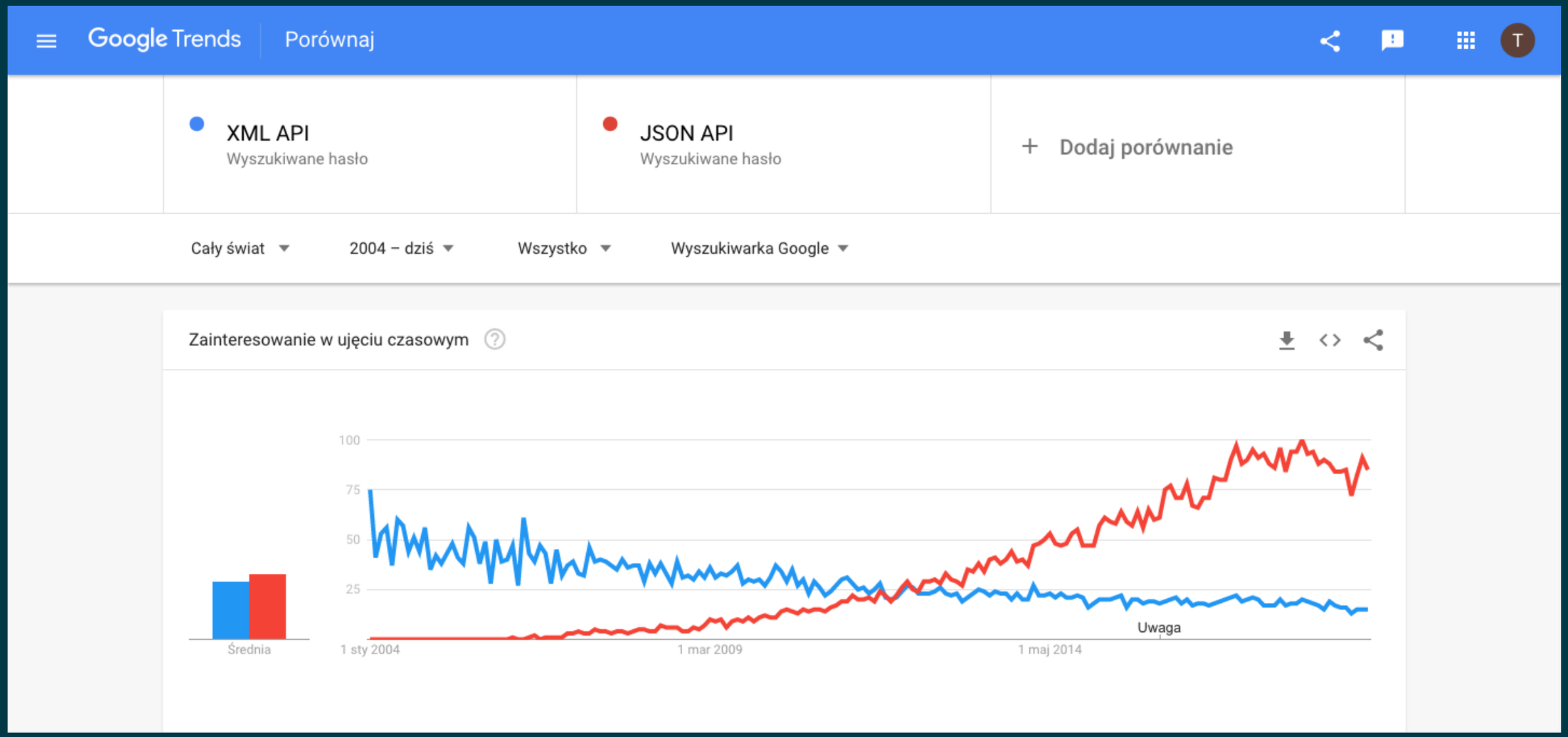
2. Use HTTP **verbs** and **response codes** to model interactions with resources:
   - **GET** = **read** a representation of a specific resource (by an identifier) or a collection of resources without side-effects, e.g. `GET /users/mszubert/repos`
   - **DELETE** = **delete** a specific resource by an identifier e.g. `DELETE /repos/mszubert/2048`
   - **POST** = **create** a new resource by appending to the existing collection resource — server generates URIs `POST /repos/mszubert/2048/forks`
   - **PUT** = **update** an existing resource or **create** a new one by sending its complete representation — client specifies URIs `PUT /repos/mszubert/2048/contents/:path`

RESTFUL WEB API GUIDELINES
HTTP VERBS PROPERTIES

HTTP METHOD	SAFE	IDEMPOTENT
GET	YES	YES
POST	NO	NO
PUT	NO	YES
DELETE	NO	YES
PATCH	NO	OPTIONAL
HEAD	YES	YES
OPTIONS	YES	YES

RESTFUL WEB API GUIDELINES

3. Specify possible representations of resources:
   - As the default representation, the recommendation is **JSON**, but services should allow clients to get alternatives (e.g. **XML**)



Source: Google Trends, https://trends.google.com

JAVASCRIPT OBJECT NOTATION

- JSON (RFC 4627) — a data interchange text-based format derived from JavaScript (a leightweight alternative to XML):
  - easy for humans to read and write
  - easy for machines to parse and generate
  - good interoperability with the client (browser)

- Commonly used in RESTful Web API as a **resource representation format**.

- JSON Schema specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control.

JSON EXAMPLE — BASIC TYPES

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

- Number — a signed decimal number that may contain a fractional part.
- String — a sequence of zero or more Unicode characters.
- Boolean — either of the values true or false.
- Array — an ordered list of zero or more values, each of which may be of any type.
- Object — an unordered collection of key/value pairs where the keys are strings.
- null — An empty value, using the word null

JSON AND REST/HATEOAS

- JSON is not a **hypermedia** format — there is no predefined way to deal with link discovery in JSON.

- Although a browser running JavaScript is consistent with the design of the Web and the formal definition of REST, the use of JSON as a data interchange format is not.

JSON AND REST/HATEOAS

- JSON is not a **hypermedia** format — there is no predefined way to deal with link discovery in JSON.

- Although a browser running JavaScript is consistent with the design of the Web and the formal definition of REST, the use of JSON as a data interchange format is not.

- **JSON-LD**, a W3C Recommendation, is a specification for encoding meaning into otherwise meaningless JSON documents:

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

- **Hypertext Application Language (HAL)** is another standard for storing links in JSON

PRESENTATION OUTLINE

- Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - **Richardson Maturity Model**
  - Java API for RESTful Services (JAX-RS)
  - .NET API for RESTful Services (WCF)

RICHARDSON MATURITY MODEL

"Steps toward the glory of REST"

3. HYPERMEDIA

2. HTTP VERBS

1. RESOURCES

0. PLAIN OLD XML

HTTP://MARTINFOWLER.COM/ARTICLES/
RICHARDSONMATURITYMODEL.HTML

RICHARDSON MATURITY MODEL
LEVEL 0 — PLAIN OLD XML

- **POX** — Plain Old XML over HTTP.
- Services use a single URI, single a HTTP verb (typically POST) and a single response status code (cf. XML-RPC)
- HTTP is used only as a synchronous, firewall-friendly transport system for remote interactions (**remote procedure calls**) based on transferring XML.
- POX-based approach **ignores** the web as a platform.

RICHARDSON MATURITY MODEL
LEVEL 0 EXAMPLE

```
1 POST /appointmentService HTTP/1.1
2 Content-Type: application/xml
3 [various other headers]
4
5 <openSlotRequest date = "2010-01-04" doctor = "mjones"/>
```

RICHARDSON MATURITY MODEL
LEVEL 0 EXAMPLE

```
POST /appointmentService HTTP/1.1
Content-Type: application/xml
[various other headers]

<openSlotRequest date = "2010-01-04" doctor = "mjones"/>
```

```
HTTP/1.1 200 OK
Content-Type: application/xml
[various other headers]

<openSlotList>
  <slot start = "1400" end = "1450">
    <doctor id = "mjones"/>
  </slot>
  <slot start = "1600" end = "1650">
    <doctor id = "mjones"/>
  </slot>
</openSlotList>
```

RICHARDSON MATURITY MODEL
LEVEL 0 EXAMPLE

```
1 POST /appointmentService HTTP/1.1
2 [various other headers]
3
4 <appointmentRequest>
5   <slot doctor = "mjones" start = "1400" end = "1450"/>
6   <patient id = "jsmith"/>
7 </appointmentRequest>
```

RICHARDSON MATURITY MODEL
LEVEL 0 EXAMPLE

```
POST /appointmentService HTTP/1.1
[various other headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

RICHARDSON MATURITY MODEL
LEVEL 0 EXAMPLE

```
POST /appointmentService HTTP/1.1
[various other headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 200 OK
[various headers]

<appointmentRequestFailure>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <reason>Slot not available</reason>
</appointmentRequestFailure>
```

RICHARDSON MATURITY MODEL
LEVEL 1 — RESOURCES

- Rather than making all requests to a singular service endpoint, reference specific **individual resources**.
- **URI templates** (e.g. `/doctors/{name}`):
  - provide a way to parameterize URIs with variables that can be substituted at runtime,
  - allow to automate the way clients bind to services,
  - provide human- and machine-readable service documentation.

RICHARDSON MATURITY MODEL
LEVEL 1 EXAMPLE

```
1 POST /doctors/mjones HTTP/1.1
2 [various other headers]
3
4 <openSlotRequest date = "2010-01-04"/>
```

RICHARDSON MATURITY MODEL
LEVEL 1 EXAMPLE

```
1 POST /doctors/mjones HTTP/1.1
2 [various other headers]
3
4 <openSlotRequest date = "2010-01-04"/>

1 HTTP/1.1 200 OK
2 [various headers]
3
4 <openSlotList>
5   <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
6   <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
7 </openSlotList>
```

RICHARDSON MATURITY MODEL
LEVEL 1 EXAMPLE

```
POST /doctors/mjones HTTP/1.1
[various other headers]

<openSlotRequest date = "2010-01-04"/>
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

RICHARDSON MATURITY MODEL
LEVEL 2 — HTTP VERBS AND CODES

- Level 2 introduces a standard set of **verbs** to handle similar situations in the same way, removing unnecessary variation.
- **GET** is crucial to support caching and improve web performance.
- **POST** and **PUT** are not strict equivalent of create and update:
  - **PUT** — create or overwrite a resource **completely** through client-generated URI.
  - **POST** — create a resource identified by a service-generated URI.
    — append a resource to a collection identified by service-generated URI.
- **PATCH** (RFC 5789) can be used to partial update of resources.
- Use **response status codes** to indicate the status.

RICHARDSON MATURITY MODEL
LEVEL 2 EXAMPLE

```
1 GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
2 [various other headers]
```

RICHARDSON MATURITY MODEL
LEVEL 2 EXAMPLE

```
1 GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
2 [various other headers]
```

```
1 HTTP/1.1 200 OK
2 [various headers]
3
4 <openSlotList>
5   <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
6   <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
7 </openSlotList>
```

RICHARDSON MATURITY MODEL
LEVEL 2 EXAMPLE

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
[various other headers]
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

RICHARDSON MATURITY MODEL
LEVEL 2 EXAMPLE

```
HTTP/1.1 201 Created
Location: /slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

```
HTTP/1.1 409 Conflict
[various headers]

<openSlotList>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

RICHARDSON MATURITY MODEL
LEVEL 3 — HYPERMEDIA / HATEOAS

- Level 3 introduces **discoverability**, providing a way of making a service more **self-documenting**.
- **Hypermedia** controls tell us what we can do next, and the URI of the resource we need to manipulate to do it.
- No need to know where to post our appointment request — the hypermedia controls in the response describes it.
- One obvious benefit of hypermedia controls is that it allows the server to change its URI scheme without breaking clients.

RICHARDSON MATURITY MODEL
LEVEL 3 EXAMPLE

```
1 GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
2 [various other headers]
```

RICHARDSON MATURITY MODEL
LEVEL 3 EXAMPLE

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
[various other headers]
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

RICHARDSON MATURITY MODEL
LEVEL 3 EXAMPLE

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
[various other headers]
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

RICHARDSON MATURITY MODEL
LEVEL 3 EXAMPLE

```
HTTP/1.1 201 Created
Location: /slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
        uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime"
        uri = "/doctors/mjones/slots?date=20100104&status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
        uri = "/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help"
        uri = "/help/appointment"/>
</appointment>
```

PRESENTATION OUTLINE

- Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - **Java API for RESTful Services (JAX-RS)**
  - .NET API for RESTful Services (WCF)

RESTFUL WEB SERVICES IN JAVA

- **Servlets** and **JSP**
  - uncomplicated but powerful
  - provide convenient wrappers around HTTP requests and responses; allow for filtering of requests by HTTP verb

- **JAX-RS** (Java API for RESTful Web Services)
  - takes full advantage of **annotations** to advertise the RESTful aspects of implemented services
  - integrates well with **JAXB** technologies to automate the conversion of Java types into **XML** and **JSON** documents.
  - mimics the routing idioms of **Rails** and **Sinatra**

- **JAX-WS** (Java API for XML Web Services)
  - `@WebServiceProvider` annotation provides lower-level API

JAX-RS — SERVICE-SIDE API

- Annotating plain Java classes allow to implement the standard principles of REST:
  - identifying a resource as a URI,
  - exposing a well-defined set of methods to access the resource,
  - providing multiple representation formats of a resource.

```java
@Path("hello/{name}")
public class HelloResource {

 private final String message = "Hello, ";

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayHello(@PathParam("name") String name) {
        return message + name + ".";
    }
}
```

JAX-RS — CLIENT-SIDE API

- JAX-RS Client API makes it easy to consume a RESTful Web service exposed over HTTP by encapsulating the key REST constraint — **Uniform Interface**:
  - every resource is identified by a URI;
  - clients interact with the resource using a fixed set of HTTP verbs
  - different representations (media types) can be returned

```
1 String entity = client.target("http://example.com/rest")
2                 .path("resource/helloworld")
3                 .queryParam("greeting", "Hi World!")
4                 .request(MediaType.TEXT_PLAIN_TYPE)
5                 .header("some-header", "true")
6                 .get(String.class);
```

PRESENTATION OUTLINE

- Motivation
- What are web services?
- Big web services:
  - Brief history of web services: RPC, XML-RPC
  - Web services protocol stack: SOAP, WSDL
  - Java API for XML Web Services (JAX-WS)
  - .NET API for Web Services (WCF)
- RESTful web services:
  - REST — Representational State Transfer
  - RESTful web API in practice
  - Richardson Maturity Model
  - Java API for RESTful Services (JAX-RS)
  - **.NET API for RESTful Services (WCF)**

CREATE A SERVICE CONTRACT

```
[ServiceContract]
public interface IService
{
    [OperationContract]
    [WebGet]
    string EchoWithGet(string s);

    [OperationContract]
    [WebInvoke(Method = "POST",
        RequestFormat = WebMessageFormat.Json,
        ResponseFormat = WebMessageFormat.Json,
        UriTemplate = "OverwrittenUri")]
    string EchoWithPost(string s);
}
```

IMPLEMENT A SERVICE

```
public class RESTService : IService
{
   public string EchoWithGet(string s)
   {
      return s;
   }

   public string EchoWithPost(string s)
   {
      return s;
   }
}
```

CONFIGURE (APP.CONFIG/WEB.CONFIG)

```
<system.serviceModel>
    <services>
      <service name="WcfService2.RESTService">
        <endpoint binding="webHttpBinding" contract="WcfService2.IService" behaviorConfiguration="REST"/>
      </service>
    </services>
    <behaviors>
      <endpointBehaviors>
        <behavior name="REST">
          <webHttp/>
        </behavior>
      </endpointBehaviors>
      <serviceBehaviors>
        <behavior>
          <!-
          To avoid disclosing metadata information, set the values below to false before deployment
          -->
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes, set the value below to
          true.  Set to false before deployment to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true"/>
</system.serviceModel>
```

CREATE SERVICE CLIENT

```
class RESTClient : ClientBase<IService>, IService
{
    public string EchoWithGet(string s)
    {
        return this.Channel.EchoWithGet(s);
    }

    public string EchoWithPost(string s)
    {
        return this.Channel.EchoWithPost(s);
    }
}
```

CONFIGURE SERVICE CLIENT

```
<system.serviceModel>
   <behaviors>
     <endpointBehaviors>
       <behavior name="REST">
         <webHttp/>
       </behavior>
     </endpointBehaviors>
   </behaviors>
   <client>
     <endpoint address="http://localhost:8088/RESTService.svc"
               name="RESTService"
               binding="webHttpBinding"
               contract="WcfService2.IService"
               behaviorConfiguration="REST"/>
   </client>
</system.serviceModel>
```

CONFIGURE SERVICE CLIENT

```
<system.serviceModel>
   <behaviors>
     <endpointBehaviors>
       <behavior name="REST">
         <webHttp/>
       </behavior>
     </endpointBehaviors>
   </behaviors>
   <client>
     <endpoint address="http://localhost:8088/RESTService.svc"
               name="RESTService"
               binding="webHttpBinding"
               contract="WcfService2.IService"
               behaviorConfiguration="REST"/>
   </client>
</system.serviceModel>
```

And run:

```
var client = new RESTClient();
client.EchoWithPost("abc");
```

WADL — WEB APPLICATION DESCRIPTION LANGUAGE

- WADL — machine-readable XML description of HTTP-based web applications (typically REST web services).
- REST equivalent of WSDL used mainly in big web services.
- WADL is not widely adopted (alternative: **swagger.io**).

```xml
<?xml version="1.0"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
  <resources base="http://localhost:8080/myapp/">
     <resource path="hello/{name}">
        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="name"
                    style="template" type="xs:string"/>
        <method id="sayHello" name="GET">
           <response>
              <representation mediaType="text/plain"/>
           </response>
        </method>
     </resource>
  </resources>
</application>
```

REST-STYLE (MICRO)FRAMEWORKS

- DropWizard
  - Jetty for HTTP
  - Jersey for JAX-RS
  - Jackson for JSON
- RestEasy
- Restlet
- Spark
- Sinatra
- Express.js
- Flask

Restlet

node.js™

Flask

web development,
one drop at a time

REST Easy

CONCLUSIONS

- Two types of web services:
  - SOAP-based
  - REST-style / RESTful
- SOAP-based web services are one approach to provide data API for heterogeneous clients.
- JAX-WS and WCF are programmer-friendly web service technologies.
- WS-* standards make big web services suitable for enterprise application–integration scenarios that have advanced quality-of-service requirements

CONCLUSIONS

- Web API form a foundation of modern web applications — next step towards separating content from presentation.
- *REST vs SOAP:*
  - REST is minimalistic, SOAP needs a stack of protocols
  - SOAP is out of sync with web architecture
  - neither Android nor iOS support SOAP natively
  - WS-* standards make SOAP services suitable for enterprise application–integration scenarios
    - Advances requirements for quality-of-service, distributed transactions, reliability, discoverability etc.

REFERENCES

- *RESTful Service Best Practices: Recommendations for Creating Web Services* — Todd Fredrich, Pearson eCollege, 2013, available at: http://www.restapitutorial.com/resources.html

- RESTful Web Services — Leonard Richardson and Sam Ruby, O'Reilly Inc., 2007, available at: http://restfulwebapis.org/rws.html

- Richardson Maturity Model — Martin Fowler, 2010 http://martinfowler.com/articles/richardsonMaturityModel.html

- *Architectural Styles and the Design of Network-based Software Architectures* — Roy Fielding, PhD thesis, University of California, Irvine, 2000 http://www.ics.uci.edu/~fielding/pubs/dissertation/

REFERENCES

- *REST in Practice* — Jim Webber, Savas Parastatidis, and Ian Robinson, O'Reilly Media, Inc., 2010
- *Java Web Services: Up and Running* — Martin Kalin, O'Reilly Media, Inc., 2013
- *RESTful Java with JAX-RS 2.0* — Bill Burke, O'Reilly Media, Inc., 2014
- *Client-Server Web Apps with JavaScript and Java* — Casimir Saternos, O'Reilly Media, Inc., 2014
- *Java Platform, Enterprise Edition: The Java EE Tutorial* https://docs.oracle.com/javaee/7/tutorial
- *Microsoft Virtual Academy, Web Services and Windows Azure*