# EECS 368
# Programming Language Paradigms

David O. Johnson

Fall 2022

# Reminders

- Assignment 5 due: 11:59 PM, Monday, October 31
  - One of our SIs, Soujanya, has come up with a great way for you to create, test, and submit Assignment 5.
  - Please review the StudentVideoDemo at the following link for more information:
  - https://drive.google.com/drive/folders/1n1R5b3YihQcbCVyQwUjVBD1IMGMdEvBE
  - If you have any question, please contact Soujanya Ambati at: saisoujanyaambati@ku.edu
  - Soujanya will also be grading Assignment 5.
- Assignment 6 due: 11:59 PM, Monday, November 14

# Any Questions?

# In-Class Problem Solution

- 23-(10-19) In-Class Problem Solution.pptx

# Any Questions?

# Sources

- https://en.wikipedia.org/wiki/API
- https://en.wikipedia.org/wiki/Software_framework
- https://en.wikipedia.org/wiki/Windows_API
- https://en.wikipedia.org/wiki/XML
- https://blog.axway.com/amplify-products/api-management/different-types-apis
- https://en.wikipedia.org/wiki/Representational_state_transfer
- https://en.wikipedia.org/wiki/SOAP

# What is an API?

- An Application Programming Interface (API) is a type of software interface, offering a service to other pieces of software.
- In contrast to a user interface, which connects a computer to a person, an API connects computers or pieces of software to each other.
- It is not intended to be used directly by a person (the end user) other than a computer programmer who is incorporating it into software.
- An API is often made up of different parts which act as tools or services that are available to the programmer.
- A program that uses one of these parts is said to call that portion of the API.
- The calls that make up the API are also known as subroutines, methods, requests, or endpoints.
- An API specification defines these calls, meaning that it explains how to use or implement them.

# Categories of APIs

APIs roughly fall into 5 categories:

1. Libraries
   - The separation of the API from its implementation can allow programs written in one language to use a library written in another.
   - Applications using libraries make calls to the library to perform certain actions, e.g., sorting a file.
2. Frameworks
   - In a framework, unlike in libraries or in standard user applications, the overall program's flow of control is not dictated by the caller, but by the framework.
   - The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions.
   - In other words, users can extend the framework, but cannot modify its code.
3. Operating systems
   - An API can specify the interface between an application and the operating system.
   - For example, POSIX provides an interface to Unix-like operating systems.
   - WinAPI provides an interface to Microsoft's Windows operating system.

# Categories of APIs

APIs roughly fall into 5 categories (continued):

4. Remote APIs
   - Remote APIs allow developers to manipulate remote resources through specific standards for communication that allow different technologies to work together, regardless of language or platform.
   - For example, the Java remote method invocation API uses the Java Remote Method Protocol to allow invocation of functions that operate remotely, but appear local to the developer.
5. Web APIs
   - Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets.
   - For example, Google has an API that let's another application use its search engine.
   - Twitter has an API that let's another program access Tweets.
   - Alexa has an API that let's another program access its speech recognition software.

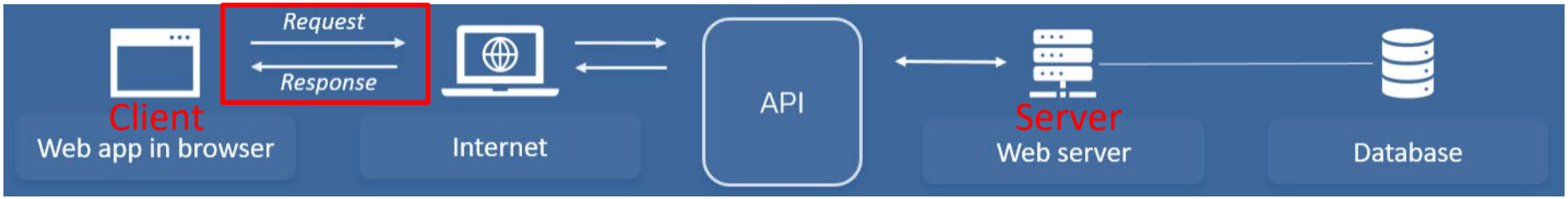
# Any Questions?

# API Protocols

- The last two categories of APIs are related to the Client-Server architecture and generally operate over the Internet:
  - Remote APIs
  - Web APIs

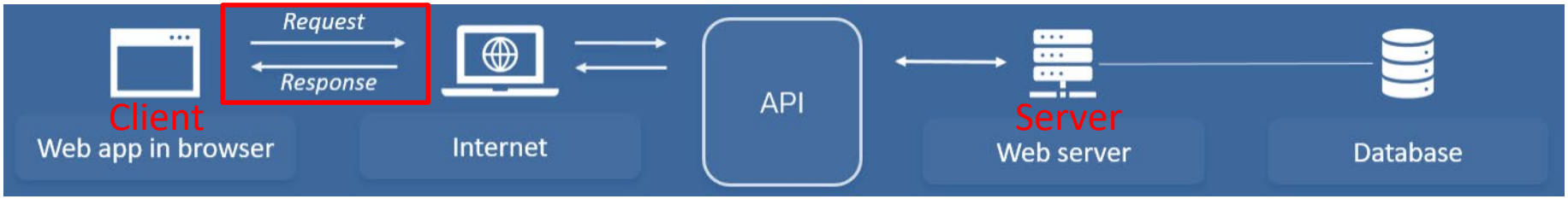
- To leverage these categories of APIs, we must follow certain protocols.
- A protocol provides defined rules for API calls.
- It specifies the accepted data types and commands.

- Let's look at the major types of protocols for APIs:
  - REST (REpresentational State Transfer)
  - SOAP (Simple Object Access Protocol)
  - RPC (Remote Procedural Call)



# API Protocols & XML

- All of the API protocols use XML.
- What is XML?
- Extensible Markup Language (XML) is a tag-based (e.g., , ) markup language and file format for storing, transmitting, and reconstructing arbitrary data.
- It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- Specified in the World Wide Web Consortium's XML 1.0 Specification of 1998 and several other related specifications—all of them free open standards.
- XML has come into common use for the interchange of data over the Internet.
- Hundreds of document formats using XML syntax have been developed (e.g., .docx, .xlsx).

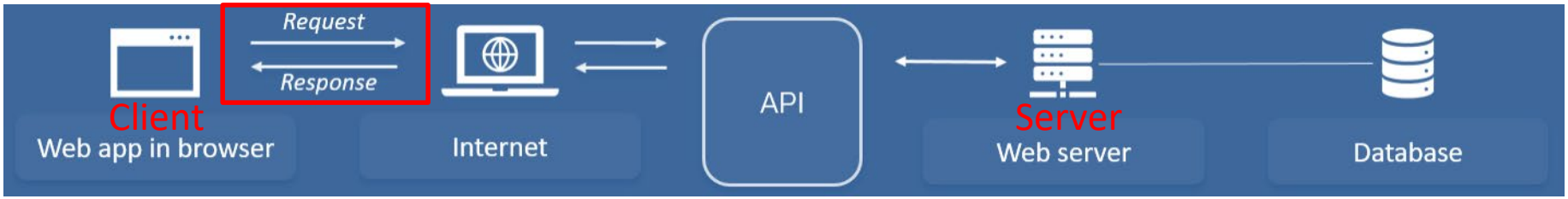


# API Protocols & XML

**HTML and XML are related to each other:**

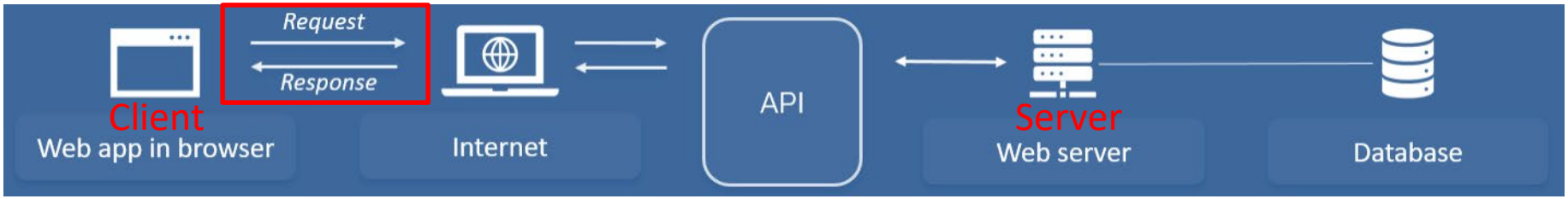
- HTML displays data and describes the structure of a webpage.
- XML stores and transfers data.

- HTML is a simple predefined language.
- XML is a standard language that defines other languages.



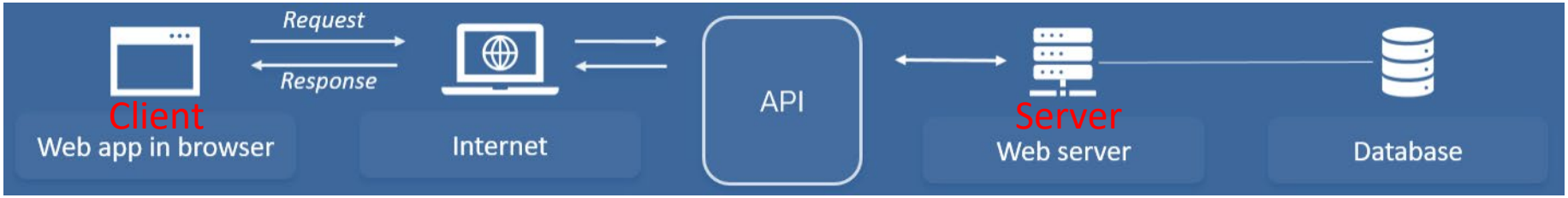
# REST

- REST APIs are a key part of modern web applications, including Netflix, Uber, Amazon, and many others.
- REST APIs are typically based on:
  - HTTP methods (e.g., GET, POST, PUT, and DELETE) to access resources via URL-encoded parameters.
  - Use of JSON or XML to transmit data.
- The REST architectural style defines six guiding constraints:
  - Client-Server architecture
  - Statelessness
  - Cacheability
  - Layered system
  - Uniform interface
  - Code on demand (optional)
- An API that follows these styles is said to be RESTful.



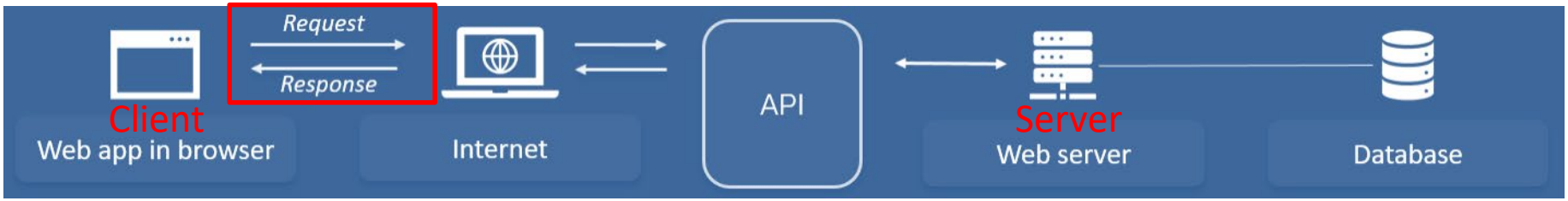
# REST (Client-Server Architecture)

- The Client-Server Architecture enforces the design principle of separation of concerns:
  - separating the user interface concerns from the data storage concerns
- Portability of the user interface is thus improved.
- In the case of the Web, a plethora of web browsers have been developed for most platforms without the need for knowledge of any server implementations.
- Separation also simplifies the server components:
  - Improving scalability
  - Allowing components to evolve independently, which is necessary in an Internet-scale environment that involves multiple organizational domains.



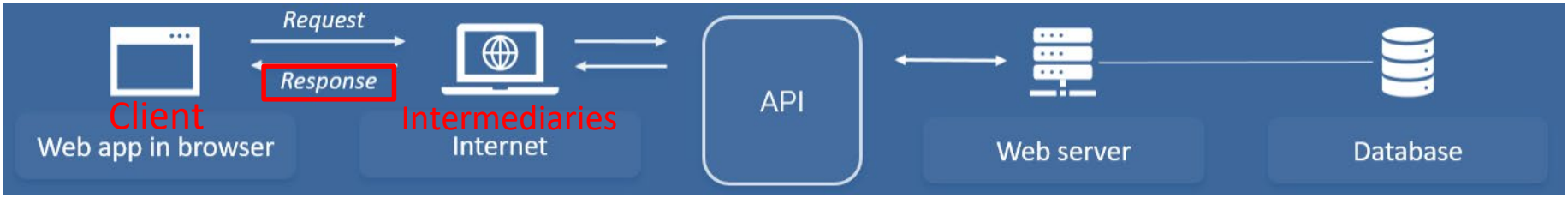
# REST (Statelessness)

- A stateless protocol is a communications protocol in which no session information is retained by the server.
- Relevant session data is sent to the server by the client in such a way that every packet of information transferred can be understood in isolation, …
- without context information from previous packets in the session.
- This property of stateless protocols makes them ideal in high volume applications, …
- increasing performance by removing server load caused by retention of session information.



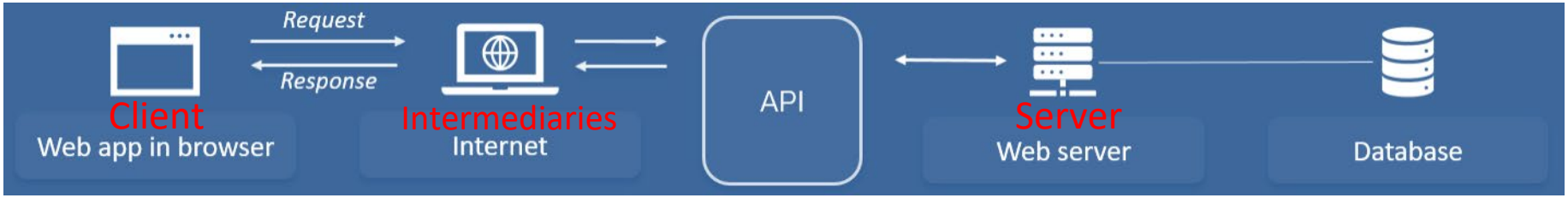
# REST (Cacheability)

- On the World Wide Web, clients and intermediaries can cache responses.
- Responses must, implicitly or explicitly, define themselves as either cacheable or non-cacheable to prevent clients from providing stale or inappropriate data to the user.
- Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance (i.e., speed).



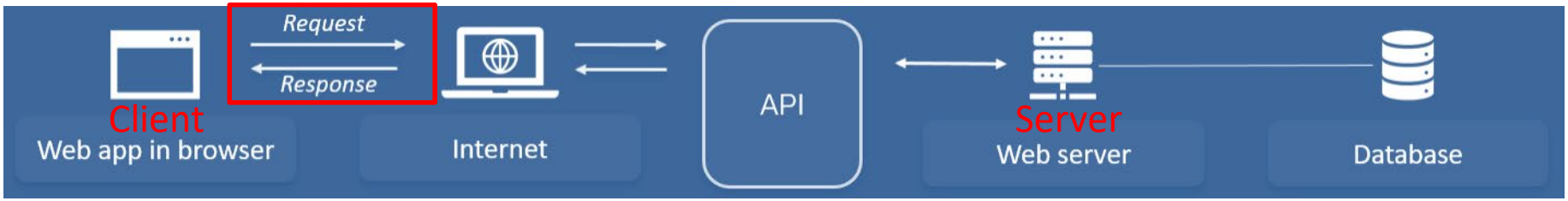
# REST (Layered System)

- A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way.
- If a proxy or load balancer is placed between the client and server, …
- it won't affect their communications, and …
- there won't be a need to update the client or server code.
- Intermediary servers can improve system scalability by enabling load balancing and by providing shared caches.
- Also, security can be added as a layer on top of the web services, …
- separating business logic from security logic.
- Adding security as a separate layer enforces security policies.
- Finally, intermediary servers can call multiple other servers to generate a response to the client.



# REST (Uniform Interface)

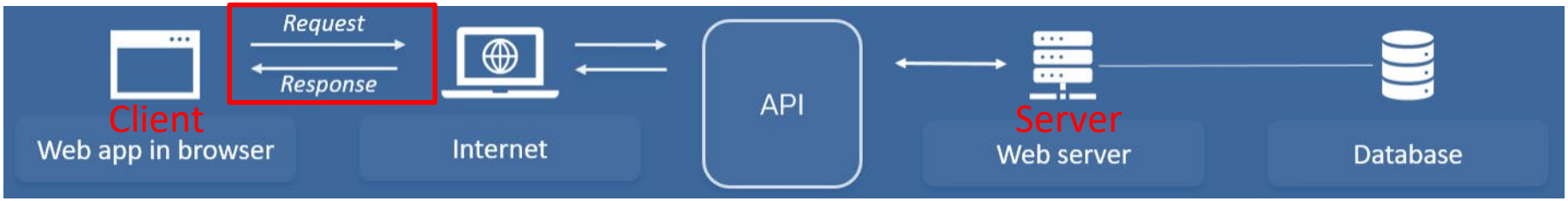
- The uniform interface constraint is fundamental to the design of any RESTful system.
- It simplifies and decouples the architecture, which enables each part to evolve independently.



# REST (Uniform Interface)

The four constraints for the uniform interface are:

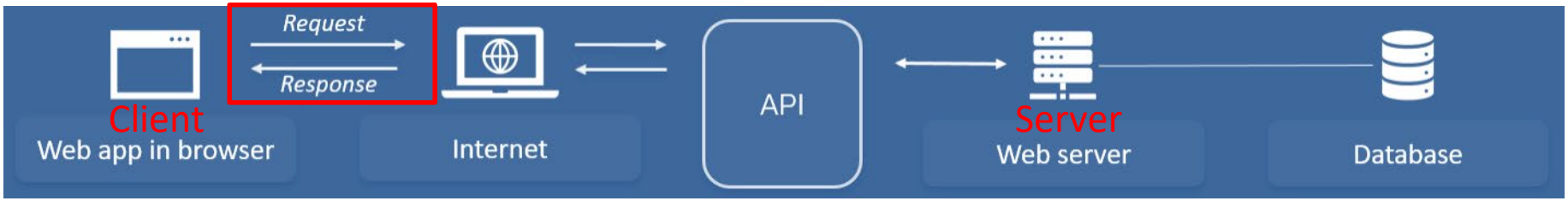
1. Resource identification in requests:
   - Individual resources are identified in requests, for example using URIs in RESTful Web services.
   - The resources themselves are conceptually separate from the representations that are returned to the client.
   - For example, the server could send data from its database as HTML, XML or JSON, none of which are the server's internal representation.
2. Resource manipulation through representations:
   - When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.



# REST (Uniform Interface)

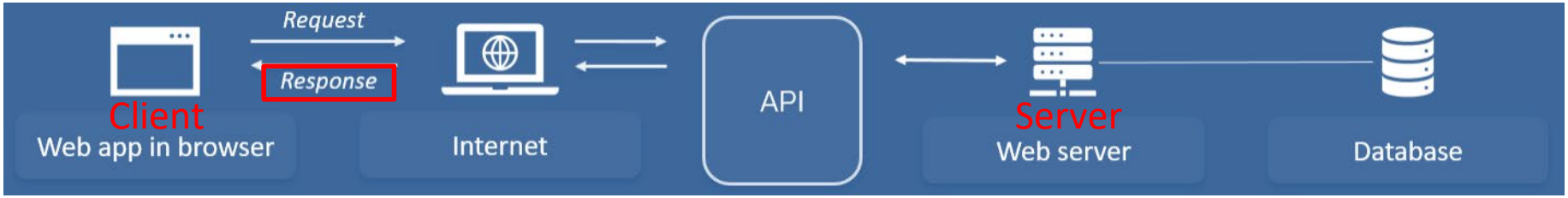
The four constraints for this uniform interface are (continued):

3. Self-descriptive messages:
   - Each message includes enough information to describe how to process the message.
   - For example, which parser to invoke can be specified by a media type.
4. Hypermedia As The Engine Of Application State (HATEOAS):
   - Having accessed an initial URI for the REST application…
   - a REST client should then be able to use server-provided links dynamically (i.e., hyperlinks) to discover all the available resources it needs.
   - As access proceeds, the server responds with text that includes hyperlinks to other resources that are currently available.
   - There is no need for the client to be hard-coded with information regarding the structure or dynamics of the application.



# REST (Code on Demand)

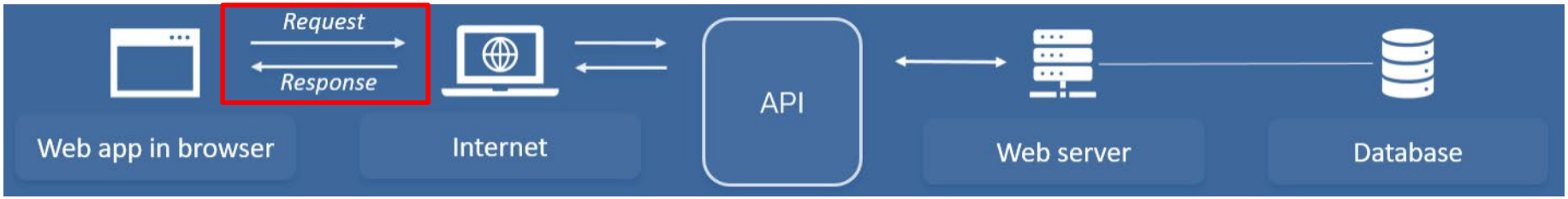
- Optionally, servers can temporarily extend or customize the functionality of a client by transferring executable code.
- For example, compiled components such as Java applets, or client-side scripts such as JavaScript.



# Any Questions?

# SOAP

- SOAP is a well-established protocol similar to REST in that it's a type of Web API.
- SOAP has been leveraged since the late 1990s.
- SOAP was the first to standardize the way applications should use network connections to manage services.
- SOAP is an Application Layer protocol that relies on other Application Layer protocols (most commonly HTTP) to transport it.



# SOAP in the OSI Protocol Model

| OSI model | | | |
|---|---|---|---|
| Layer | | | Function |
| Host layers | 7 | Application | High-level APIs, including resource sharing, remote file HTTP s SOAP |
| | 6 | Presentation | Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption |
| | 5 | Session | Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes |
| | 4 | Transport | Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing TCP |
| Media layers | 3 | Network | Structuring and managing a multi-node network, including addressing, routing and traffic control |
| | 2 | Data link | Reliable transmission of data frames between two nodes connected by a physical layer |
| | 1 | Physical | Transmission and reception of raw bit streams over a physical medium |

# SOAP

It is an XML-based protocol consisting of three parts:

An envelope, which defines the message structure and how to process it.

A set of encoding rules for expressing instances of application-defined datatypes.

A convention for representing procedure calls and responses.

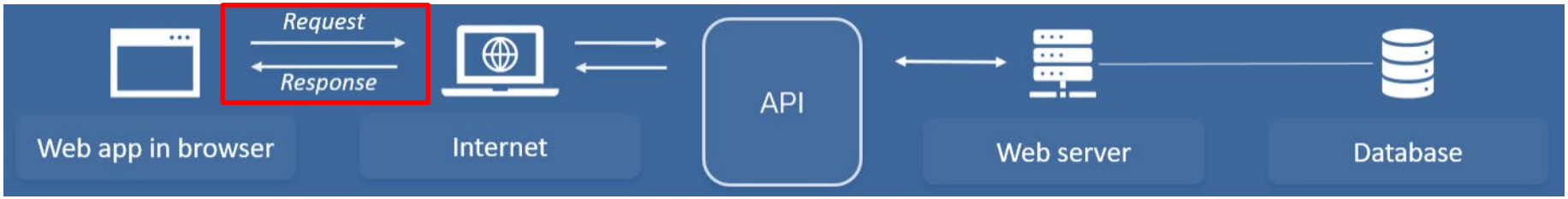
```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

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

# Any Questions?

# RPC

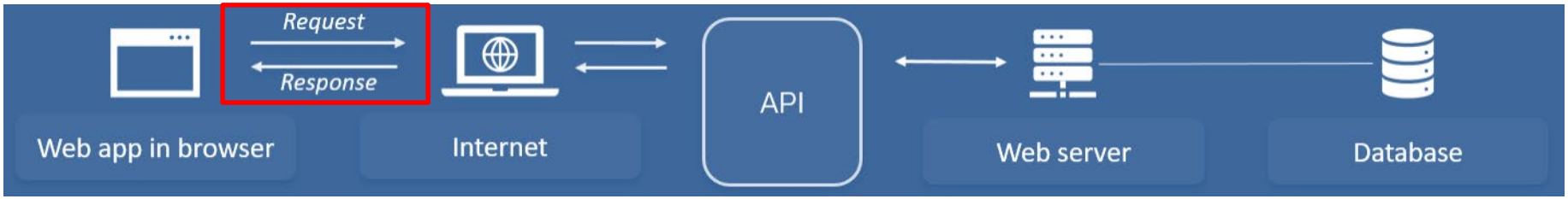
- An RPC is a remote procedural call protocol.
- They are the oldest and simplest types of APIs.
- The goal of an RPC is for the client to execute code on a server.
- XML-RPC uses XML to encode its calls, …
- while JSON-RPC uses JSON for the encoding.
- Both are simple protocols.
- Though similar to REST, there are a few key differences.
- RPC APIs are very tightly coupled, so this makes it difficult to maintain or update them.
- To make any changes, a new developer has to go through various RPC's documentation to understand how one change could affect the other.



# Any Questions?

# API Protocols Summary

- APIs play a key role in the development of any application.
- And REST has become the preferred standard for building applications that communicate over the network.
- REST fully leverages all the standards that power the World Wide Web.
- REST is simpler than traditional SOAP-based web services.
- Unlike RPC, it allows for a loosely coupled layered architecture to maintain easily or update them.



# Any Questions?

# In-Class Problem

Considering the API between the client browser and the file server that we talked about in previous lectures and that you are writing for Assignment 5:

1. Is the API an RPC? Explain why.
2. Is the API SOAP? Explain why.
3. Does the API follow the REST client-server architecture style? Explain why.
4. Does the API follow the REST stateless architectural style? Explain why.
5. Does the API follow the REST cacheability architectural style? Explain why.
6. Does the API follow the REST layered system architectural style? Explain why.
7. Does the API follow the optional REST code on demand architectural style? Explain why.
8. Is it a RESTful API? Explain why.