

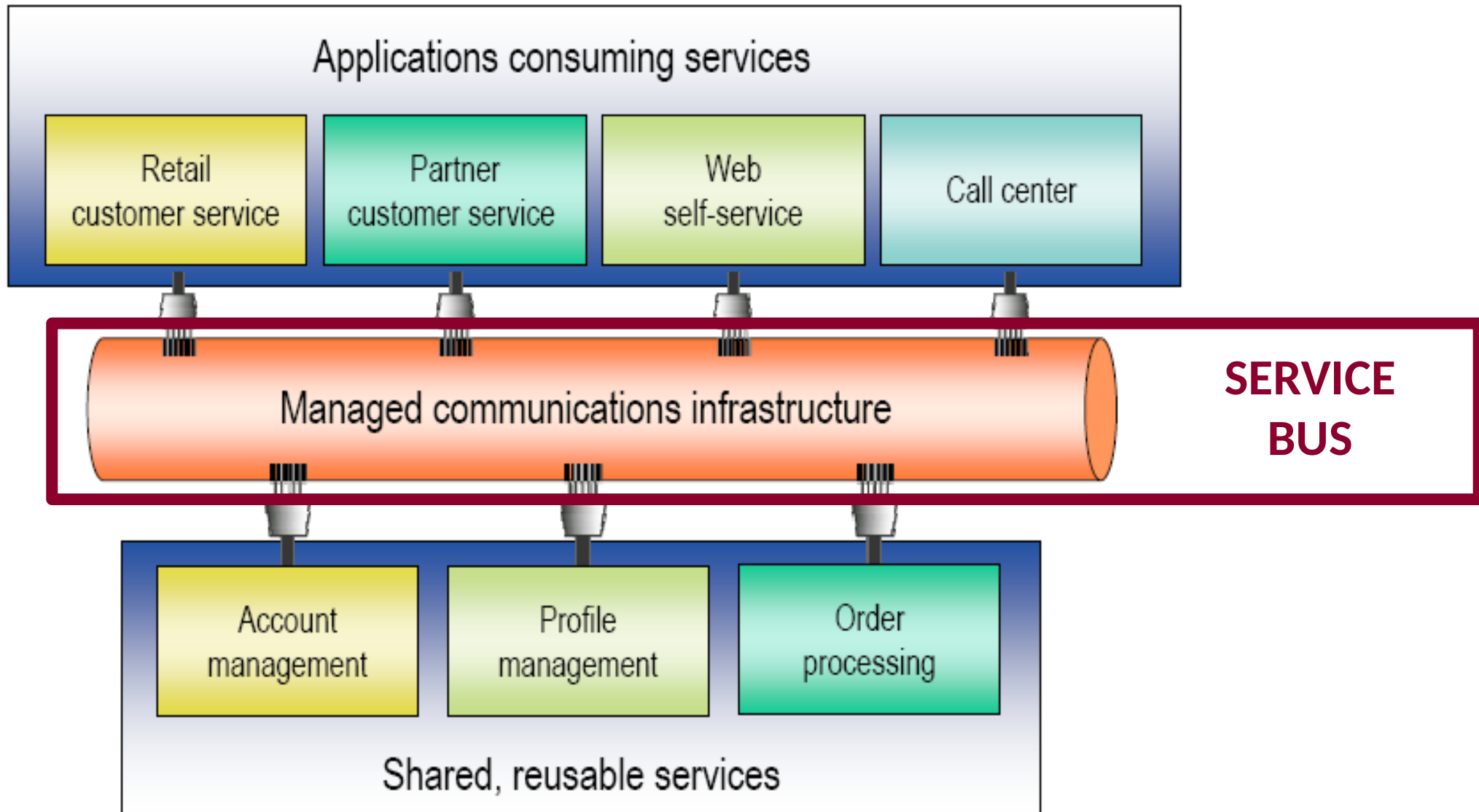
Web Services

José Oramas



Previous: Service-Oriented Architectures

Focus on shared, reusable, loosely coupled services



Today



One building block technology
for constructing SOAs

Using the largest distributed system at hand



Web Services

Services using web technology

Agenda for today



Web Services in theory and practice

- What is a web service?
- Where is it used?



RESTful Web Services

- One particular type of Web Service
- Very popular



SOAP Web Services

- “Older” web services
- Still used a lot

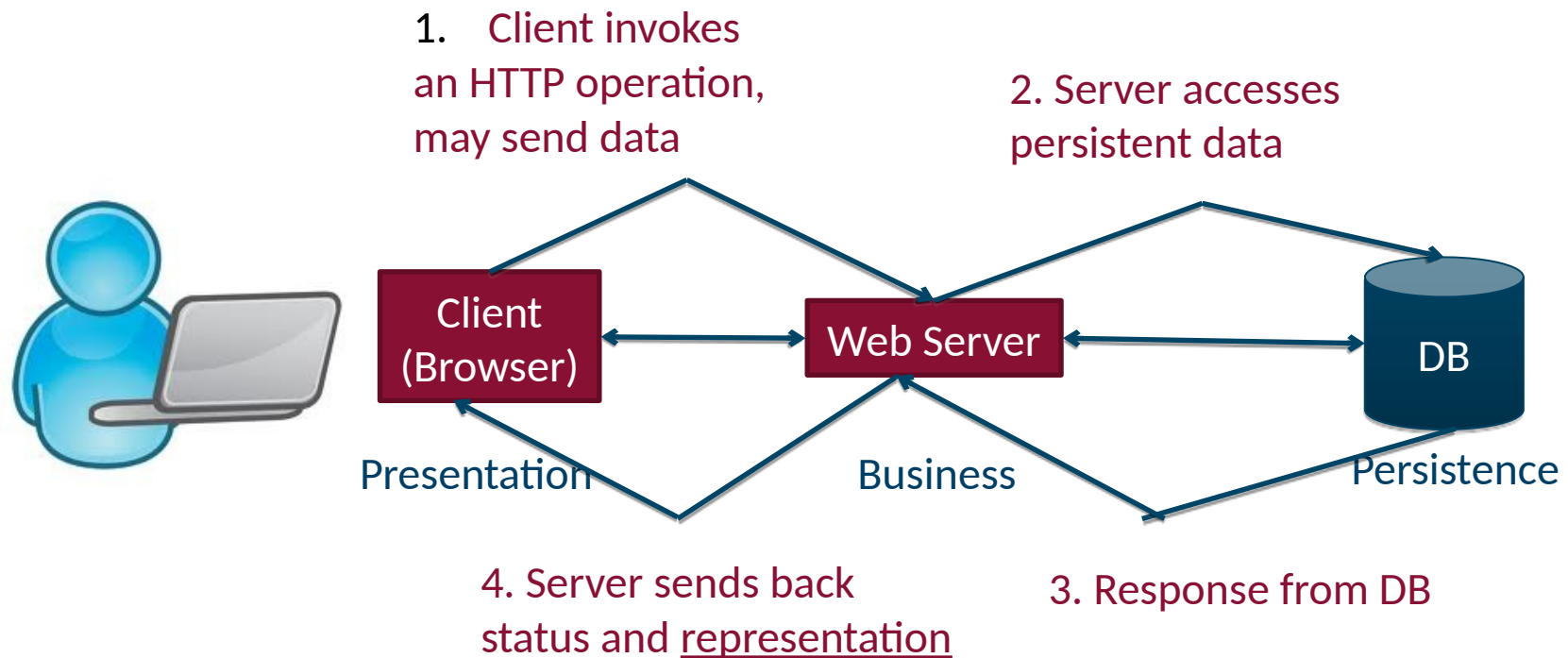
So far: Anatomy of Web Apps

3-tier architecture

Again presentation, business, persistence

Still used a lot in small-scale websites...

... but every “layer” is a process

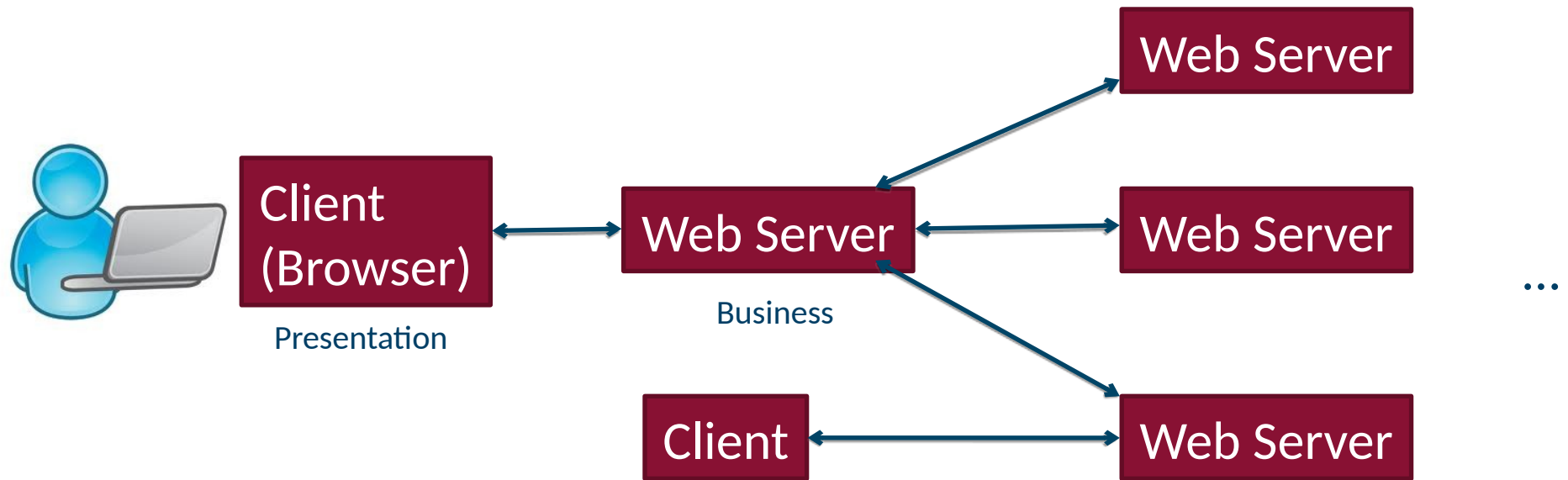


More: Anatomy of Web Apps



What happens if we increase the scale?

SOA approach: composition of services



“Web Services”

Where do we see web services?

Bangkok Hotels


02/22/2014 02/23/2014 **Change dates** **Clear dates**

All Bangkok hotels (740) **Just for you** **BETA** **Best Value (73)** **Family (200)** **Luxury (193)**

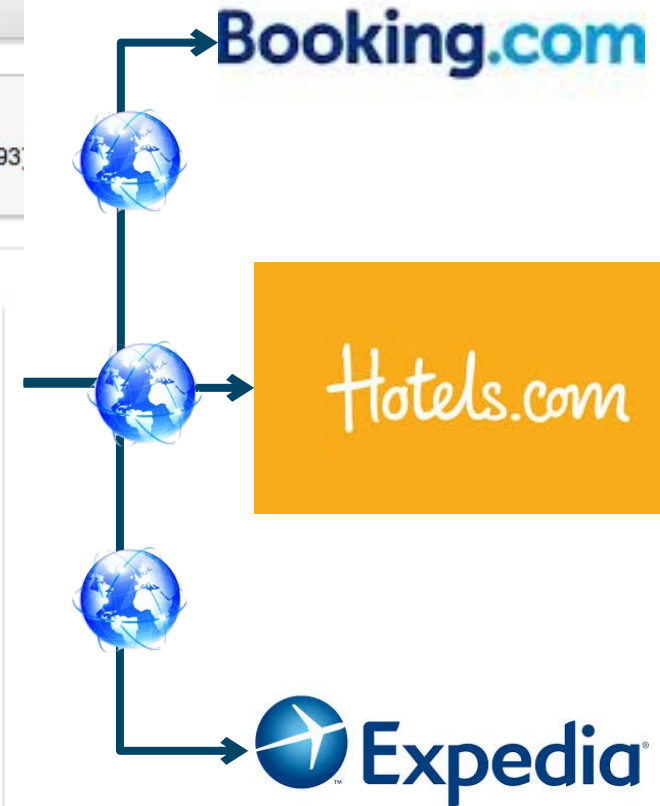
Price ▾ Rating ▾ Neighborhood ▾ More ▾

740 of 740 hotels shown **Sorted by** Availability ▾

Mandarin Oriental, Bangkok ★★★★★
🏆 Travelers' Choice® 2014 Winner Top Hotels | Luxury
📌 Special Offer High Speed Internet On Us

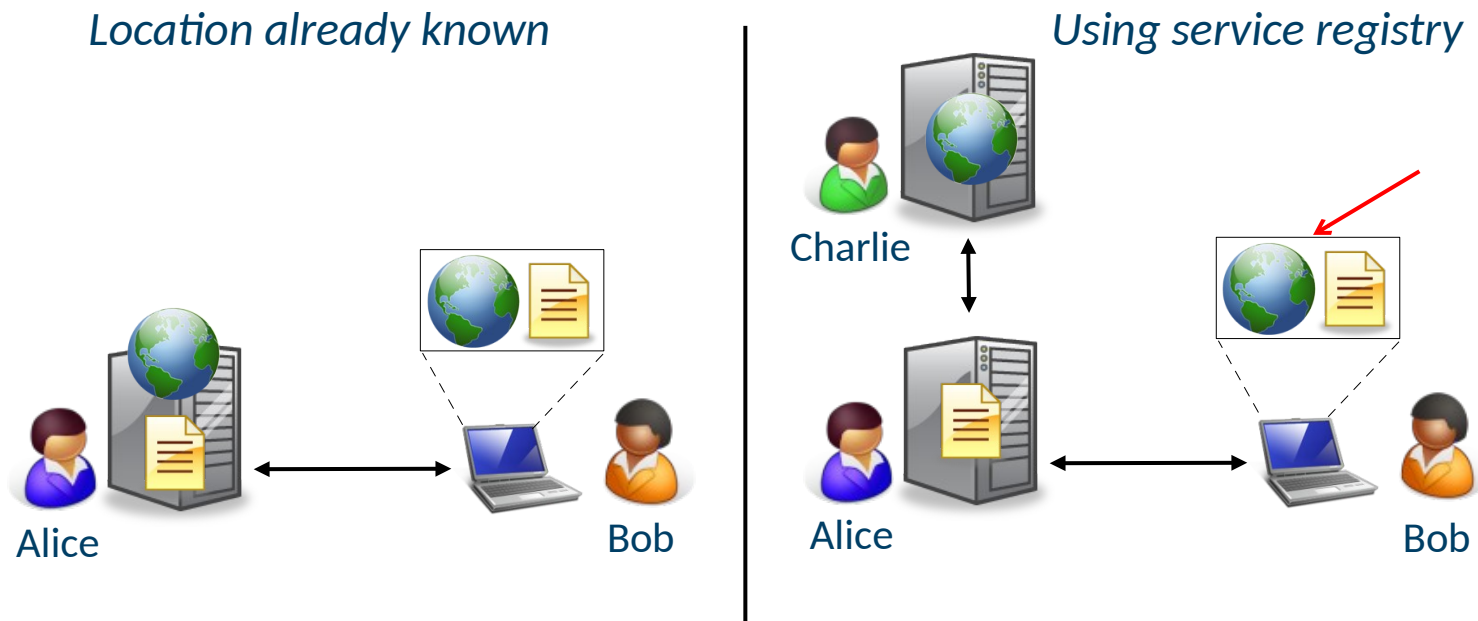
 **#1 of 740 hotels in Bangkok**
🟢🟢🟢🟢🟢 1,895 reviews
"Service perfection!" 02/17/2014
"World class." 02/16/2014
Professional photos | Traveler photos (1143) | Map

Booking.com
€ 317* per night >
Mandarin Ori... € 319*
Expedia.be € 320*
Hotels.com € 320*
See all 7



Web Services

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network” [W3C2004]



Intuition: access a service using the web

Interaction through **messages** with a standardized format (HTTP/XML)

Focus is on exposing functionality to third parties (read & writes)

No one-size-fits-all underlying technology: SOAP, REST, etc.

A more detailed definition

*"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 (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP or RESTful messages, typically conveyed using HTTP with an XML or JSON serialization **in conjunction with other Web-related standards**."*

Key elements:

Machine-to-machine interaction

Interoperable (with other applications and services)

Machine-processable format

Key technologies:

SOAP, REST: used for communication

WSDL (Web Services Description language; XML-based)

Benefits of Web Service



Loose Coupled Design

Each service exists **independently** of the other services that make up the application. Individual pieces of the application to be **modified without impacting** unrelated areas.



Ease of Integration

Data is isolated between applications . Web Services **act as glue** between these and enable easier communications within and across organizations.



Service Reuse

Takes code reuse a step further. A specific function within the domain is only ever **coded once and used over and over** again by consuming applications.

Difference with SOAs?

Web Services examples



Google Direction API

- Get routing instructions from one location to the other
- <https://developers.google.com/maps/documentation/directions/start>

The Facebook logo, consisting of the word 'facebook' in white lowercase letters on a blue rectangular background.

facebook

Facebook Graph API

- Query Facebook content using a web service
- <https://developers.facebook.com/docs/graph-api/>

- Following a specific API: need to know the syntax
- Different Web Services give different output formats.
- XML, JSON, etc.
- Sometimes you have a choice, sometime you don't

Break

[See you in 10 minutes]

REST WEB SERVICES



This guy changed the distributed computing world twice

Roy Fielding

Senior Principal Scientist at Adobe Systems



- Inventor / co-author of HTTP
- Co-founder of Apache Web Server
- Got a PhD in 1998 describing **RE**presentational **S**tate **T**ransfer
- Capture **key** characteristics and **principles of the Web** in an architectural style

Introduction to REST

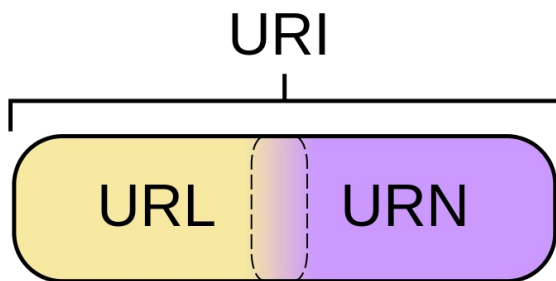
Representational state transfer (REST) is a **style** of software architecture for distributed systems such as the World Wide Web

It's an architectural style not a protocol

Resource is a central concept

= Any item of interest identified by some identifier

- National Product of Belgium in 2012, Maximum Flow algorithm, Kill Bill 2
- <http://www.gambling.com/bets?horse=bigbrown&jockey=kent>



Identifier is a Uniform Resource Identifier (URI)

- Can be URL, URN or both
- **Uniform Resource Locator:**
The location where you can find the resource, method for finding it
- **Uniform Resource Name**
Defines an item identity

REST URI design

Remember: REST is an architectural principle – not a protocol

Anyone can claim that his API is RESTful

A few design principles you need to stick to

REST URI design: how are REST resources described?

- Describe resources, not applications
- Non-restful URIs:
 - `/admin/updatebook.jsp?book=5`
 - `/bookview.jsp?book=5`
 - `/bookandreviews.jsp?book=5`
- Make it short: <https://www.flickr.com/services/api/rest>
- Hackable up the tree:
 - `uantwerp/courses/computer-science/master/distributed-computing`
 - `Uantwerp/courses/`
- Meaningful:
 - <http://ua.ac.be/main.aspx?c=jose.oramas>
<https://www.uantwerpen.be/en/staff/jose-oramas/>
- Predictable, human-readable
- Nouns, not verbs
- Permanent
- Query arguments are only for parameters
- Avoid extensions

REST and RESTful and HTTP



REST principles were used for designing HTTP 1.1
REST was initially described with HTTP in mind
REST is often linked with HTTP
But REST on top of application layers is also possible

RESTful API:

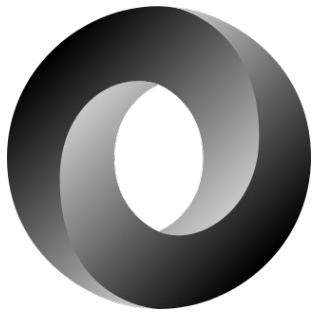
Web Service API that adheres to the REST style, meaning

- It has a base URI, such as <http://example.com/resources/>
- It has an Internet media type for the data.
- It uses standard HTTP methods
- It uses hypertext links to reference state
- It uses hypertext links to reference related resources

It has an Internet media type for the data

This can be everything: XML, Atom, images, etc.

Often used: JSON



JSON: JavaScript Object Notation

- Open Standard
- Human-Readable text
- Alternative to XML
- Language-independent
- Support in many programming languages

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    }
  ]
}
```

JSON has...

- The ability to define a schema for validation
- Its own RPC system (web service)

It uses standard HTTP methods

REST uses HTTP methods GET, PUT, POST and DELETE to modify / access resources

According to CRUD: all major functions that are implemented in databases

- **Create** ☒ HTTP PUT or **HTTP POST**
- **Read** ☒ HTTP GET
- **Update** ☒ HTTP PUT
- **Delete** ☒ HTTP DELETE

HTTP verb / URI	Intended meaning
POST employees	Create a new employee from the requested data
GET employees	Return a list of all employees
GET employees/21	Return employee number 21
GET employees/new	Get the form to create a new employee
GET employees/21/edit	Get the form to edit employee number 21
PUT employees	Update the employee with the requested data
DELETE employees	Delete all employees
DELETE employees/21	Delete employee number 21

REST architectural constraints

①

Interface uniformity between components

They need to speak the same language and know what they expect in order to understand each other.

②

Client-server model (separation of concerns)

Client and server are fully decoupled and can be developed independently

③

Stateless client-server communication

Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored session state on the server

REST architectural constraints

4

Caching

Clients can cache responses to requests, improving scalability and performance. The fact that responses are not cacheable should be implicitly or explicitly defined.

5

Layering

A client cannot tell if it is directly connected to a server or to some intermediary (proxy, cache, tunnel, firewalls, ...). This allows for load-balancing, fail-over and data transformation.

6

Code-on-demand (optional)

Client functionality can be dynamically extended through the transfer of executable code (e.g. Javascript)

REST vs RPC

RPC (and Distributed Object Systems, SOAP Web Services)

- Many operations, few URI
- Address **software components**
- Procedure name and parameters are transferred

REST

- Few operations, many URI
- Address **resources**
- Resource representations are transferred

Break

[See you in 10 minutes]

SOAP WEB SERVICES



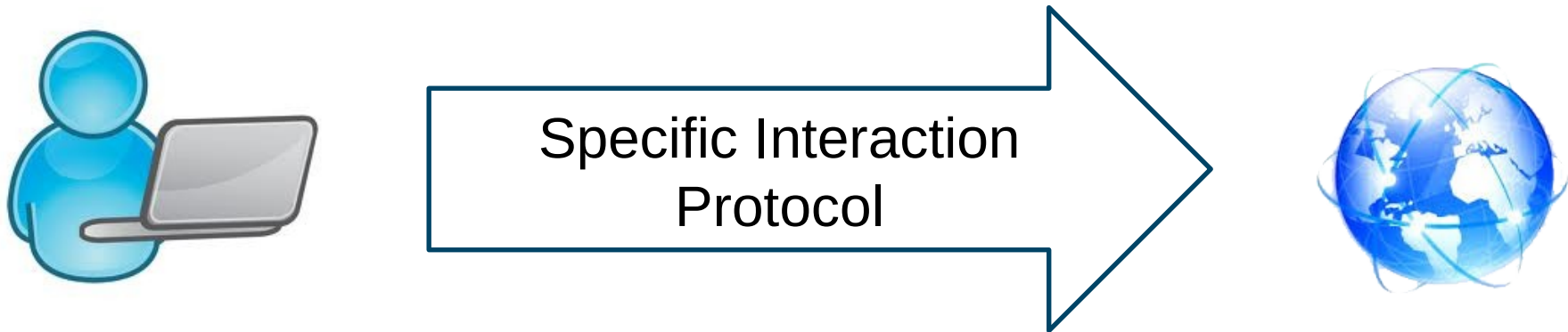
What we already saw: REST Web Service



HTTP POST, PUT, GET,
DELETE & UPDATE



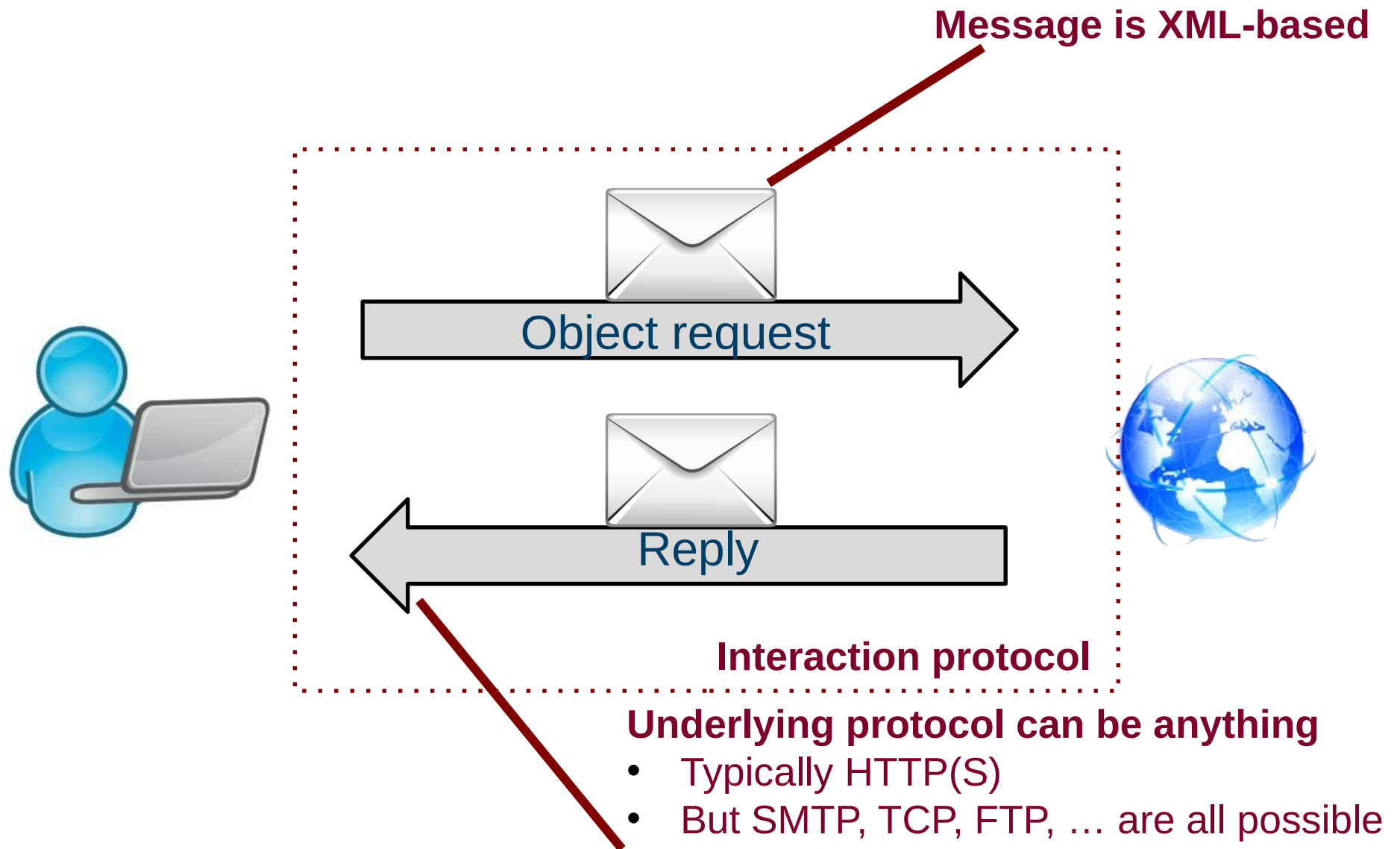
Now: SOAP Web Service



Requirements:

- Text protocol: needs to be readable on the wire
- Object oriented: needs to describe components of a service

The SOAP principle



Interaction protocol = SOAP

Simple Object Access Protocol

W3C Standard (originated from Microsoft)

Promoted as right style for Web Services

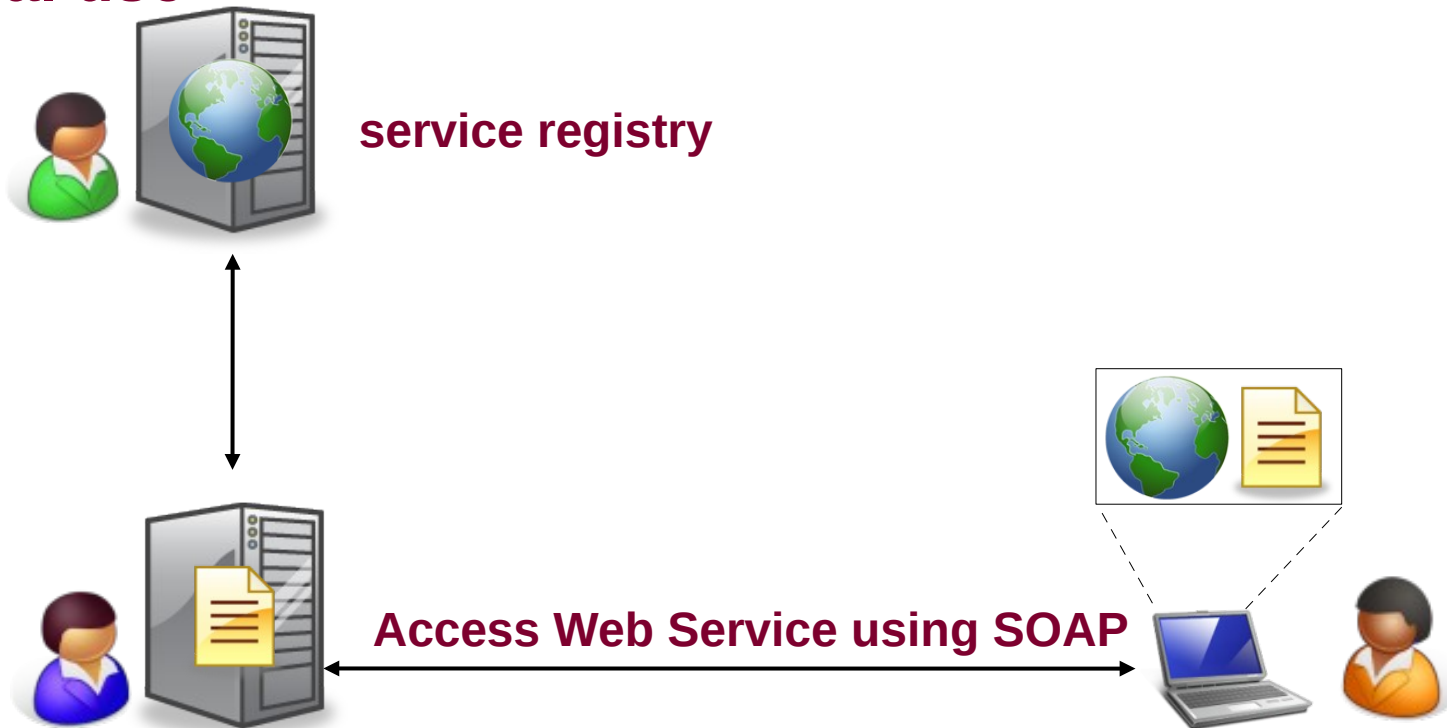
Rich feature set

Older than REST

Still very popular



Typical use



Unlike REST, SOAP is only linked with XML

Message **format** and **processing rules**

Stateless and one-way

Can be used to develop more complex “conversations”

Significant overhead

XML-is “**human readable**” argument?

Why XML?

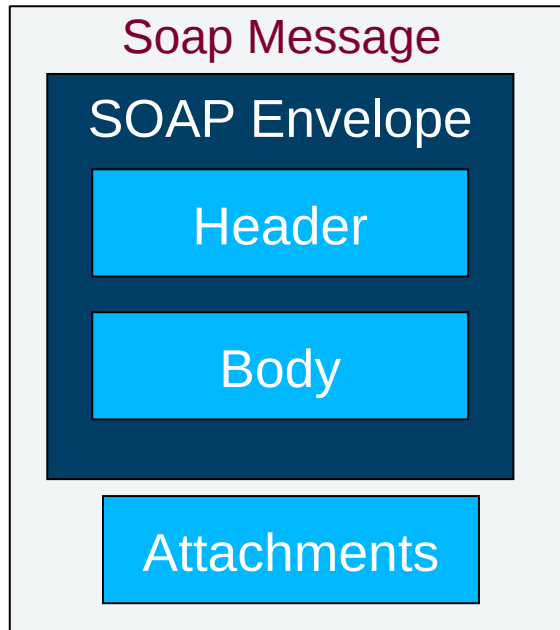
One level above XML RPC

- Adds processing and extensibility models to the story
- Style for exchanging arbitrary data

It was the de facto standard in human readable text back then



SOAP Messages



```
<env:Envelope
```

```
xmlns:env="http://www.w3.org/2003/05/soap-  
envelope">
```

```
  <env:Header>
```

```
    ...
```

```
  </env:Header>
```

```
  <env:Body>
```

```
    ...
```

```
  </env:Body>
```

```
</env:Envelope>
```

Envelope is root element of a SOAP message and contains

- An optional SOAP header: for extensions (see later)
- SOAP body: can be anything

SOAP RPC message : request example

`stock.GetStockPrice("IBM")`

`POST /InStock HTTP/1.1`

`Host: www.stock.org`

`Content-Type: application/soap+xml; charset=utf-8`

`Content-Length: nnn`

**HTTP Protocol
binding**

`<?xml version="1.0"?>`

`<soap:Envelope`

`xmlns:soap="http://www.w3.org/2001/12/soap-envelope"`

`soap:encodingStyle=`

`"http://www.w3.org/2001/12/soap-encoding">`

**Specify
namespaces
and encoding
rules**

`<soap:Body xmlns:m="http://www.stock.org/stock">`

`<m:GetStockPrice>`

`<m:StockName>IBM</m:StockName>`

`</m:GetStockPrice>`

`</soap:Body>`

`</soap:Envelope>`

**RPC-
specification**

SOAP RPC message : request example

`stock.GetStockPrice("IBM")`

SMTP protocol binding

From: me@my.com
To: you@your.com
Subject: Greeting

Specify namespaces and encoding rules

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

RPC-specification

```
<soap:Body xmlns:m="http://www.stock.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

HTTP SOAP Reply

HTTP/1.1 200 OK

Content-Type: application/soap; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-  
envelope"
```

```
soap:encodingStyle="http://www.w3.org/2001/12/soap-  
encoding">
```

```
  <soap:Body xmlns:m="http://www.stock.org/stock">
```

```
    <m:GetStockPriceResponse>
```

```
      <m:Price>34.5</m:Price>
```

```
    </m:GetStockPriceResponse>
```

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

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

SOAP message : fault example

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

  <soap:Body xmlns:m="http://www.stock.org/stock">
    <faultcode>soap:Server</faultcode>
    <faultstring>Stock not found</faultstring>
    <detail>
      <m:StockError>
        <m:stockName>IBM</m:stockName>
      </m:StockError>
    </detail>
  </soap:Body>
</soap:Envelope>
```

WSDL

Web Services Description Language

- XML **grammar to specify collection of “access end points”**
(1 URL specifies a single access end point)
- designed to **automate application-to-application interaction** (or B2B interaction)
- defines the ***communication protocol* to be used at runtime**
 - message format
 - methods to be invoked
 - parameter lists, return types
 - ...
- WSDL descriptions can be automatically generated for existing code
- stub classes can be generated from WSDL descriptions

Let's zoom out...

What we've seen so far...

You know what SOAP is

You know how SOAP looks like

You know how to write it

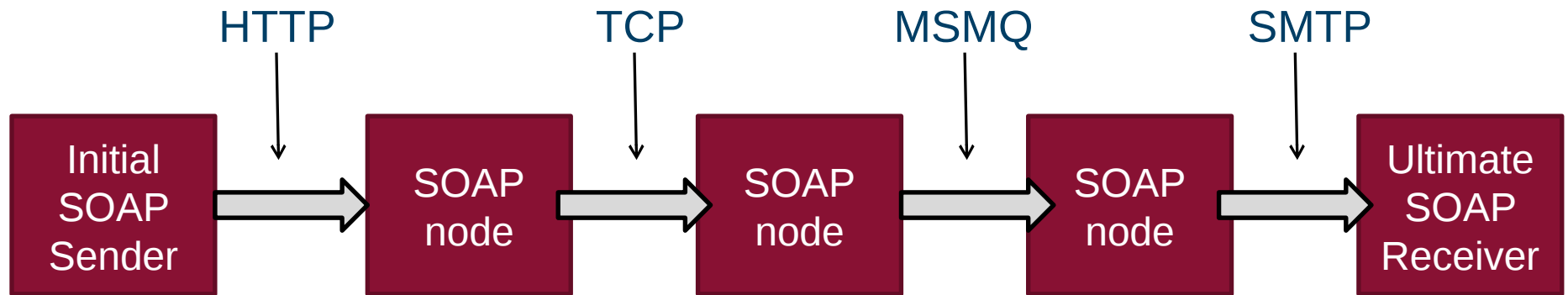
You know about WSDL

SOAP Message Model

SOAP Processing Model

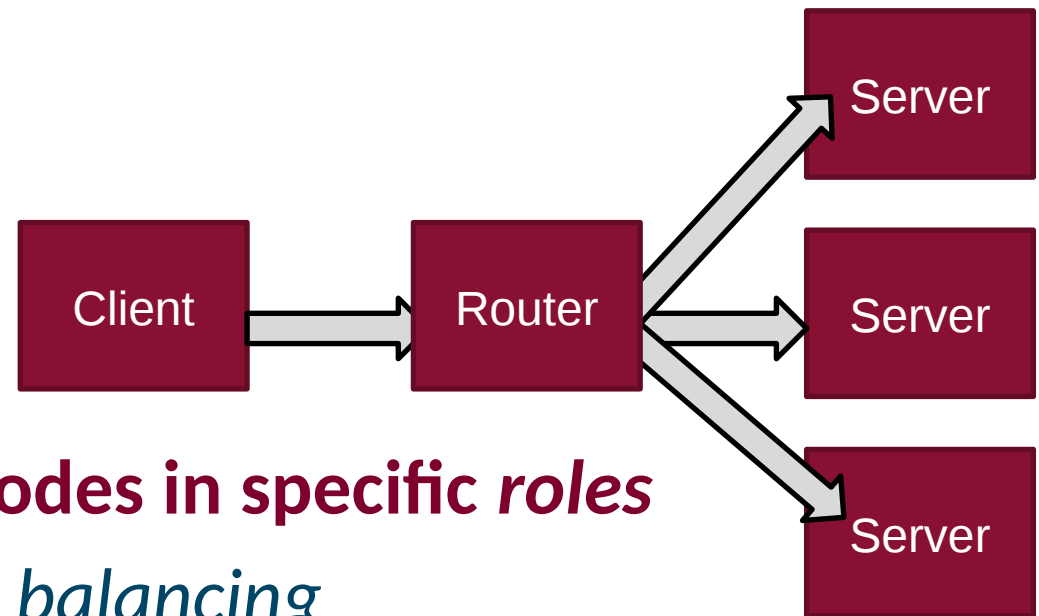
SOAP Extensibility Model & Extensions

SOAP Processing Model



A SOAP app is made of communicating *nodes*

- Sender
- Receiver
- Initial Sender (Originator)
- Intermediary
- Ultimate Receiver



SOAP messages may target nodes in specific *roles*

For example, to allow for load balancing

SOAP Extensibility Model

Suppose you want to add credentials to a SOAP message: two options

Option 1) Add it in the SOAP body

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <x:TransferFunds xmlns:x="urn:examples-org:banking">
      <from>22-342439</from>
      <to>98-283843</to>
      <amount>100.00</amount>
      <!-- security credentials -->
      <credentials>
        <username>dave</username>
        <password>evad</password>
      </credentials>
    </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

Credentials need to be encoded in every parameter

SOAP Extensibility Model

Option 2) Use the optional SOAP header

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!-- security credentials -->
    <s:credentials xmlns:s="urn:examples-org:security">
      <username>dave</username>
      <password>evad</password>
    </s:credentials>
  </soap:Header>
  <soap:Body>
    <x:TransferFunds xmlns:x="urn:examples-org:banking">
      <from>22-342439</from>
      <to>98-283843</to>
      <amount>100.00</amount>
    </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

Header blocks can be annotated with `mustUnderstand` attribute. This means that receiving body must be able interpret this header.

SOAP is much bigger!

WS-Negotiation

Negotiating about the SLAs

WS-Agreement

Defining an SLA (contract) on the performance of the web service



Huge range of extensions, all with particular purpose

WS-Discovery

Automated search & detection of web services

WS-Security

WS-Addressing

WS-Transactions

REST vs SOAP

Advantages of REST

- Uniform interface is immutable
- Conceptually simpler
- Lower protocol overhead (for stateless services)
- Can improve server scalability
- Inherent support for request caching
- Less reliant on tool support and heavyweight libraries
- No need for additional messaging layer
- Complexity at the endpoints instead of the transport layer level

Disadvantages of REST

- Service model not always easily mappable to REST verbs, uniform interface tailored to large-grain hypermedia data transfer
- Looser in terms of defining the exposed service API and its semantics
- Does not cover all WS standards (security, transactions, addressing, coordination, policy, reliable messaging, ...)

When to use SOAP / REST?

SOAP's popularity is clearly decreasing

General rule of thumb: "Unless you have a definitive reason to use SOAP use REST"

Then, what's a good reason to use SOAP?

SOAP exposes application logic as services

Need more formal ways of invoking services (e.g., WSDL)

Need stateful information

If you need one of the WS-* extensions

- Security
- Transactional logic
- Load balancing
- Etc.

Pay Attention To...



Web Services in theory and practice

- What is a web service?
- Where is it used?
- Positioning w.r.t. Service Oriented Architectures



RESTful Web Services

- Main principles behind REST
- Architectural constraints
- What makes an API RESTful
- Comparison with standard RPC



SOAP Web Services

- Description
- Comparison w.r.t. REST

Questions?

Further Reading

- **Web services**

<http://www.w3.org/TR/ws-gloss/#webservice>

<https://restfulapi.net/rest-api-design-tutorial-with-example/>

- **Flickr RESTful API** (some examples)

<https://www.flickr.com/services/api/rest>

- **REST API and hypertext**

<https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

<https://symfonycasts.com/screencast/rest/rest>

- **Chapter 5 of Fielding's PhD:**

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- **SOAP**

<https://www.guru99.com/soap-simple-object-access-protocol.html>

<https://www.w3.org/TR/SOAP-attachments/>

- **WSDL**

<https://www.w3.org/TR/wsdl.html>

Web Services

José Oramas