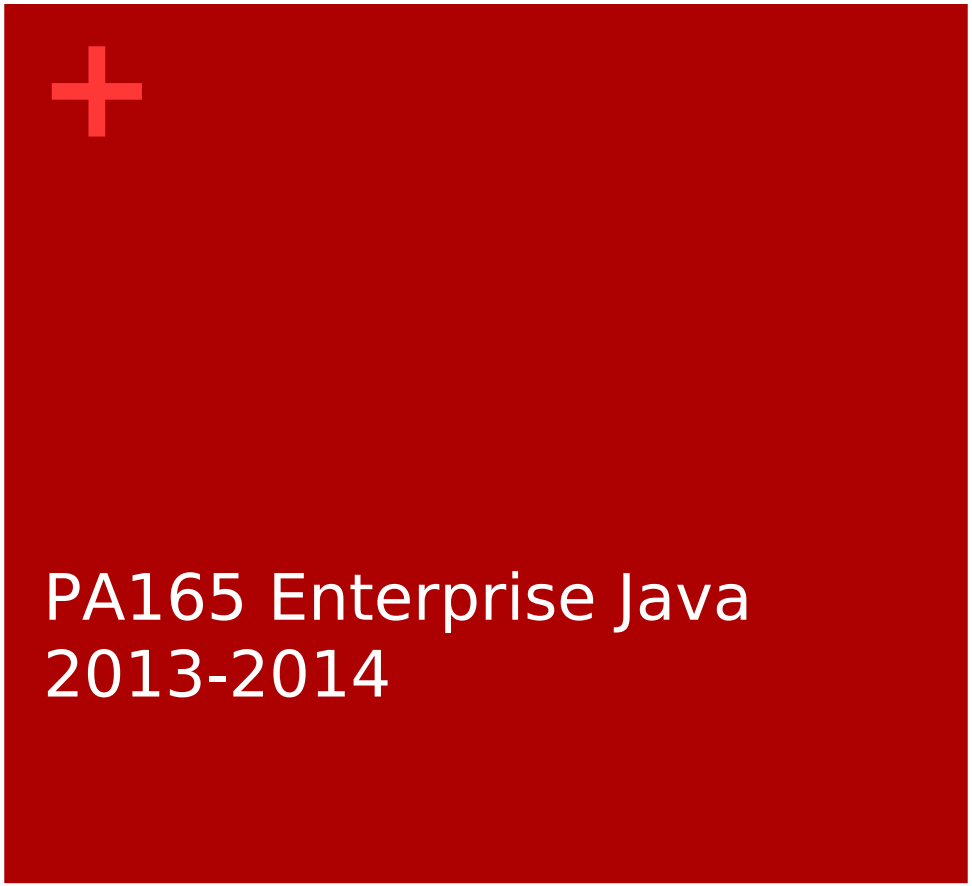


PA165 Enterprise Java
2013-2014

# Intro to Service Oriented Computing (SOC) and Service Oriented Architecture (SOA)

Bruno Rossi & Juha Rikkilä

# Read Queen’s race

Web technologies evolve and diversify rapidly, though standardization create some uniformity.
This introduction presents one selection of topics and technologies.

"*Well, in our country you'd generally get to somewhere else — if you run very fast for a long time, as we've been doing.*” said Alice

"*A slow sort of country!*" said the Queen. "*Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!*"

# Objectives and content of this lecture

## Objectives

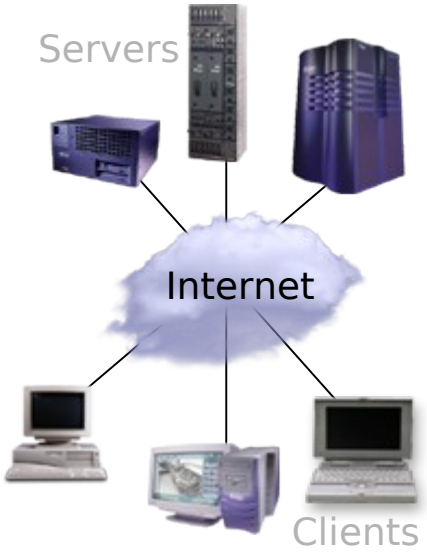
Get “the big picture” of SOA and project it to the current web experience

## Content

- Clients and servers
- SOA, Why and Why not
- Application development view
- Technology stack view
- Basic set of concepts

# Distributed Computing Evolution

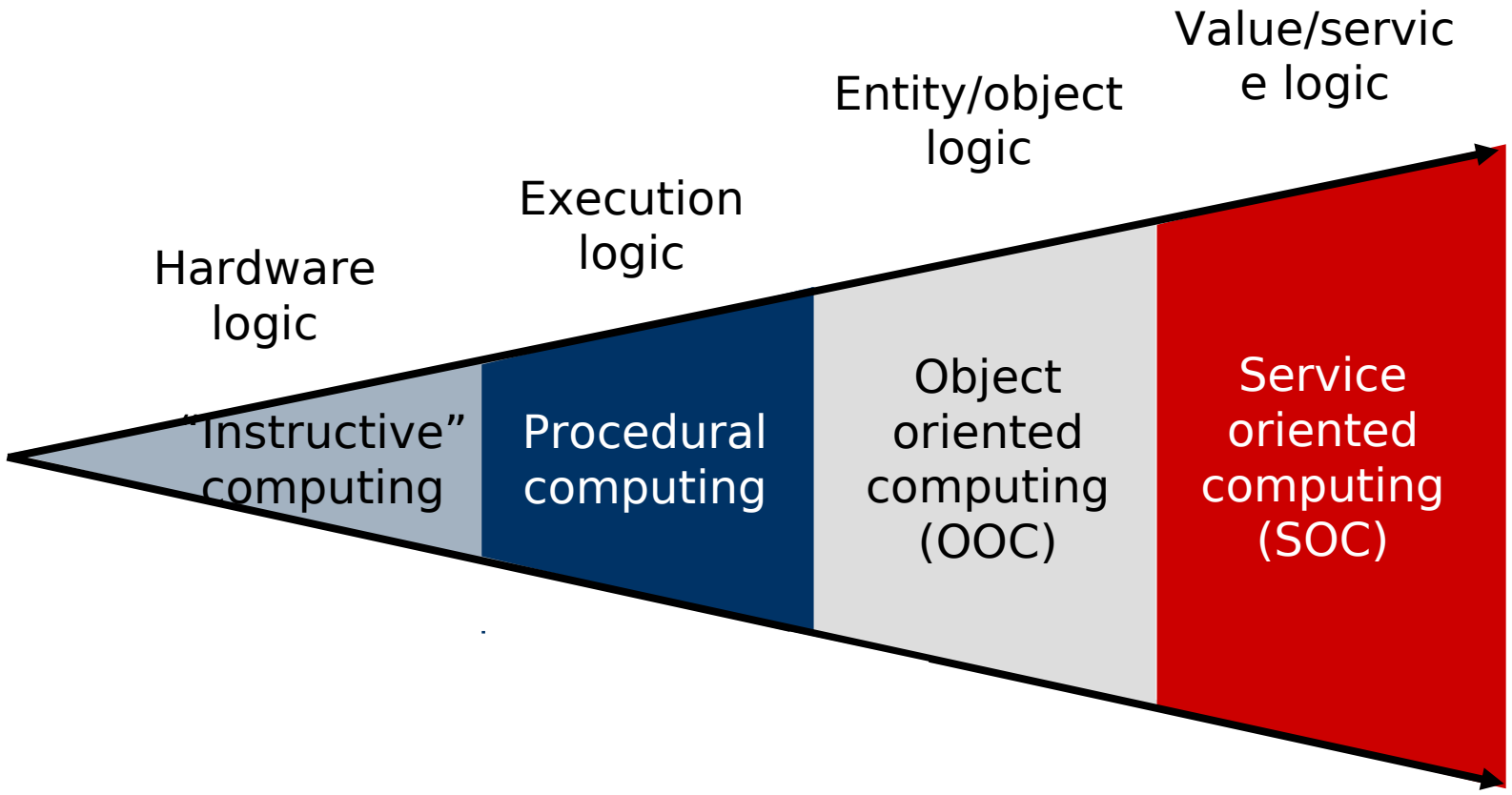

Client-Server(C/S) silos




Web-based computing


Web Services/Peer-to-Peer

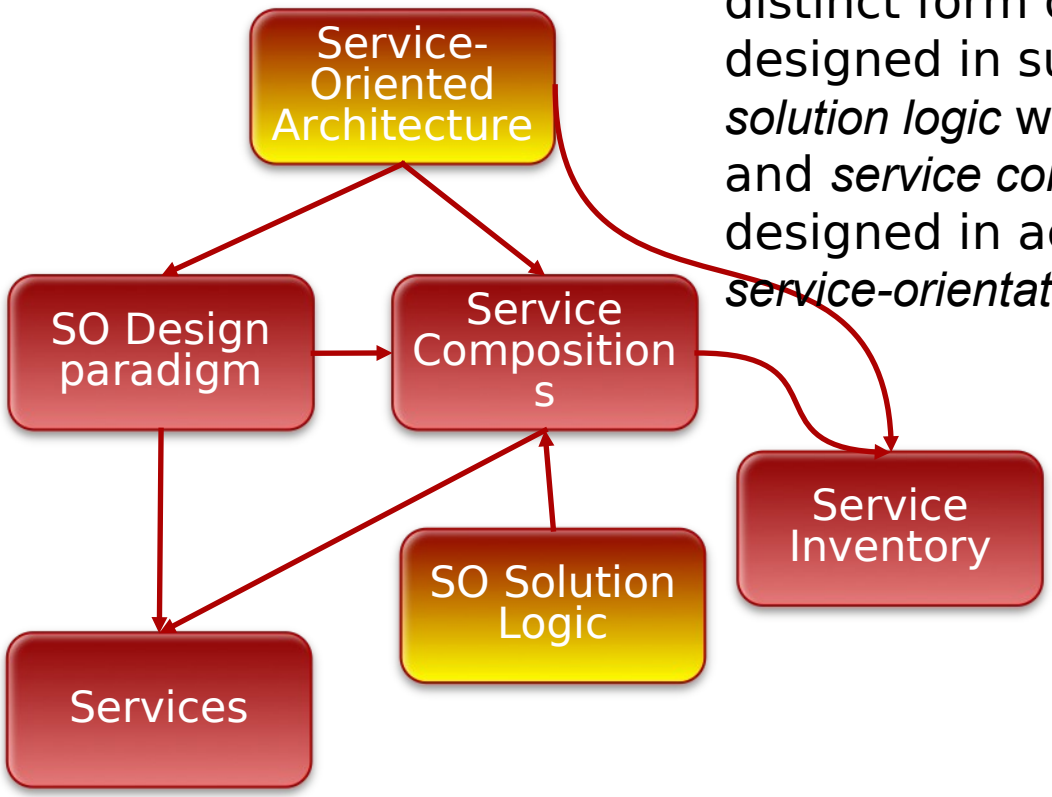
# Evolution of software development /programming



# SOC

- SOC is an emerging cross-disciplinary paradigm for distributed computing that is changing the way software applications are designed, architected, delivered and consumed
- SOC is a new computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments

# Service-Oriented Computing




*Service-oriented architecture* represents a distinct form of technology architecture designed in support of *service-oriented solution logic* which is comprised of *services* and *service compositions* shaped by and designed in accordance with *service-orientation*.

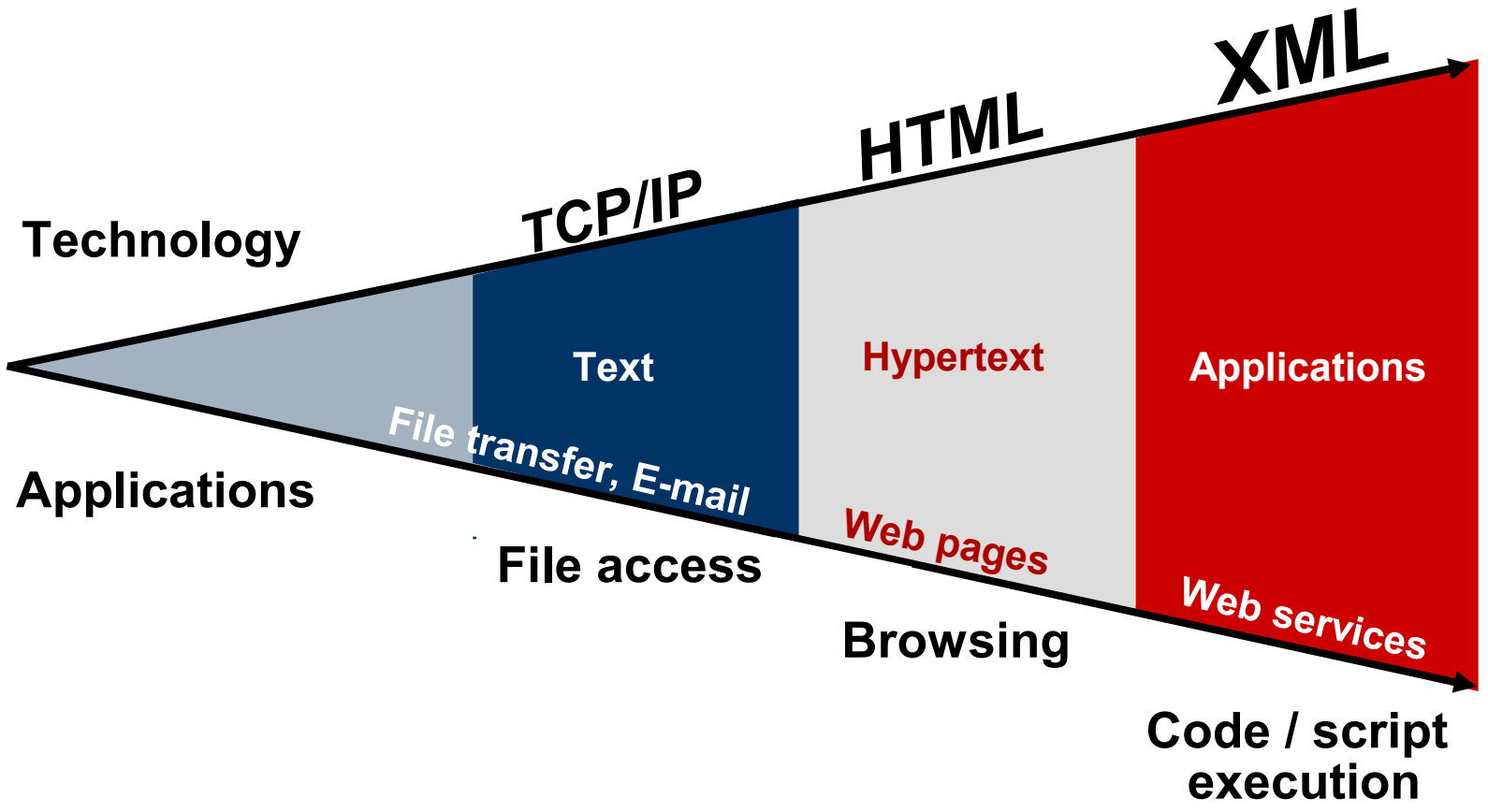
*Service-orientation* is a design paradigm comprised of *service-orientation design principles*. When applied to units of solution logic, these principles create *services* with distinct design characteristics that support the overall goals and vision of *service-oriented computing*.

*Service-oriented computing* represents a new generation computing platform that encompasses the *service-orientation paradigm* and *service-oriented architecture* with the ultimate goal of creating and assembling one or more *service inventories*

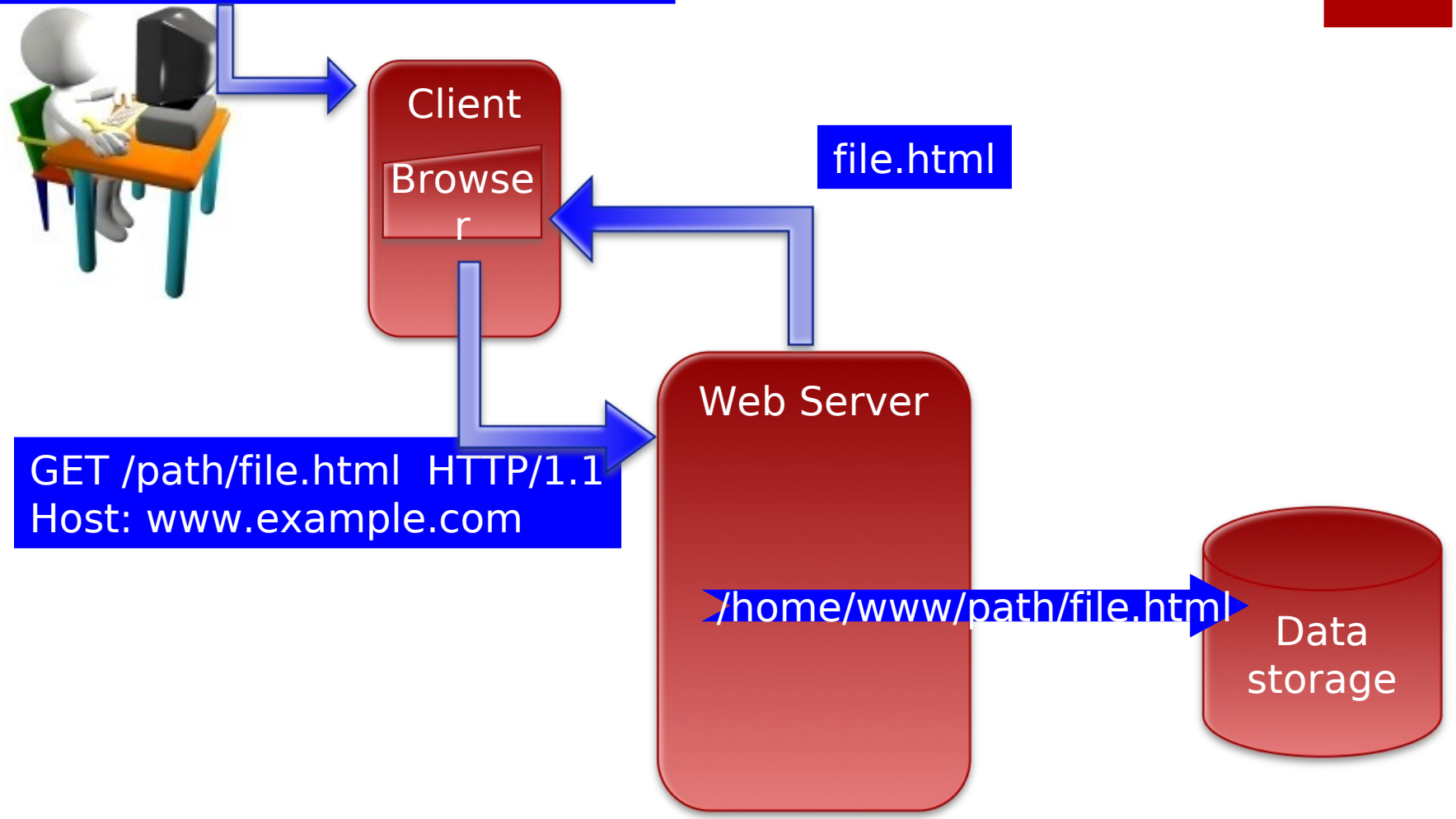
# SOC elements, an implementation perspective

- *Service-oriented solution logic* is implemented as *services* and *service compositions* designed in accordance with *service-orientation design principles*
- A *service composition* is comprised of *services* that have been assembled to provide the functionality required to automate a specific business task or process
- Because *service-orientation* shapes many *services* as agnostic enterprise resources, one *service* may be invoked by multiple consumer programs, each of which can involve that same *service* in different *service composition*
- A collection of standardized *services* can form the basis of a *service inventory* that can be independently administered with its own physical development environment
- Multiple business processes can be automated by the creation of *service compositions* that draw from a pool of existing agnostic *services* that reside within a *service inventory*
- *Service-oriented architecture* is a form of technology architecture optimized in support of *services*, *service compositions* and *service inventories*

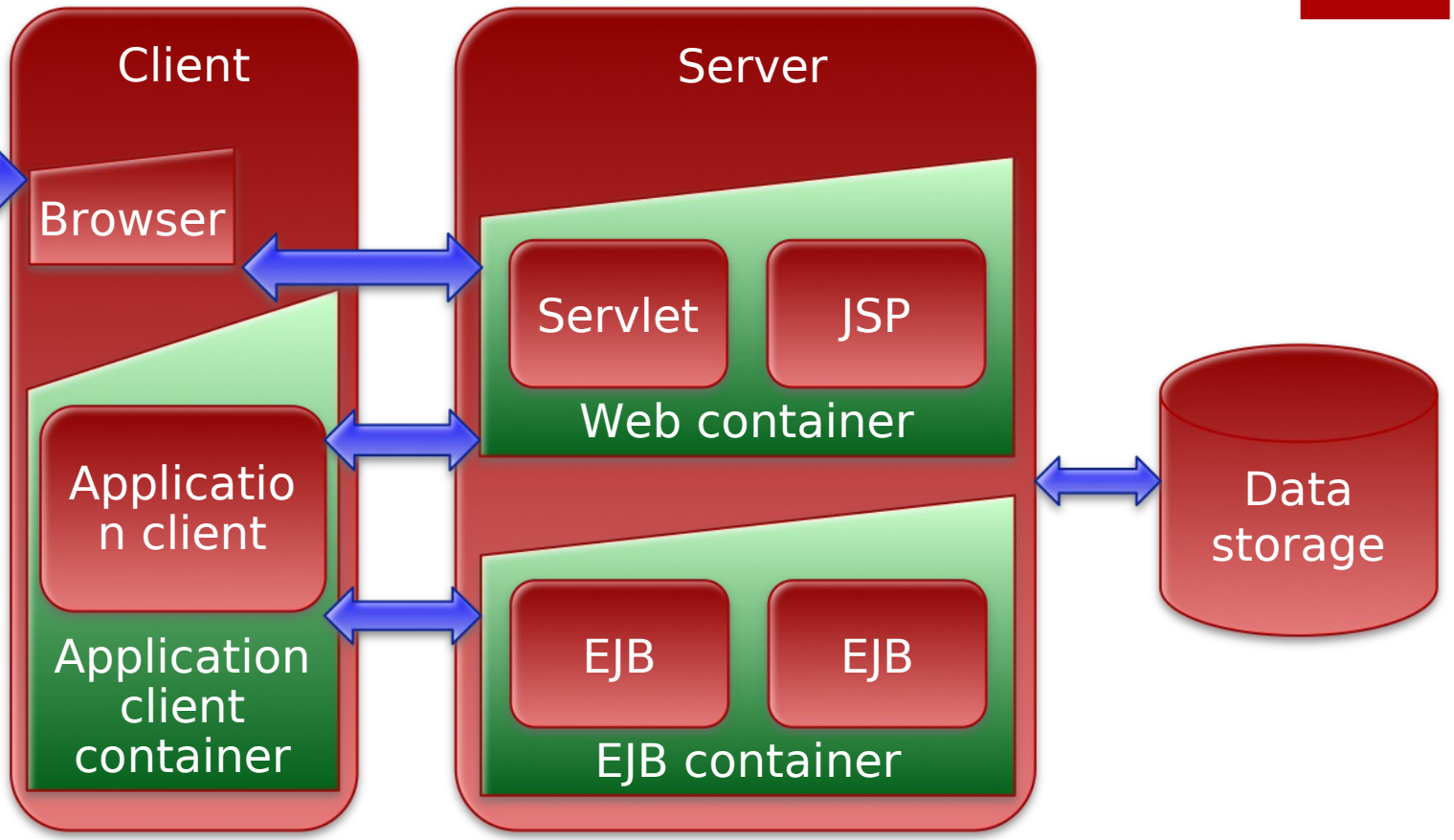
# Internet evolution



# Browsing

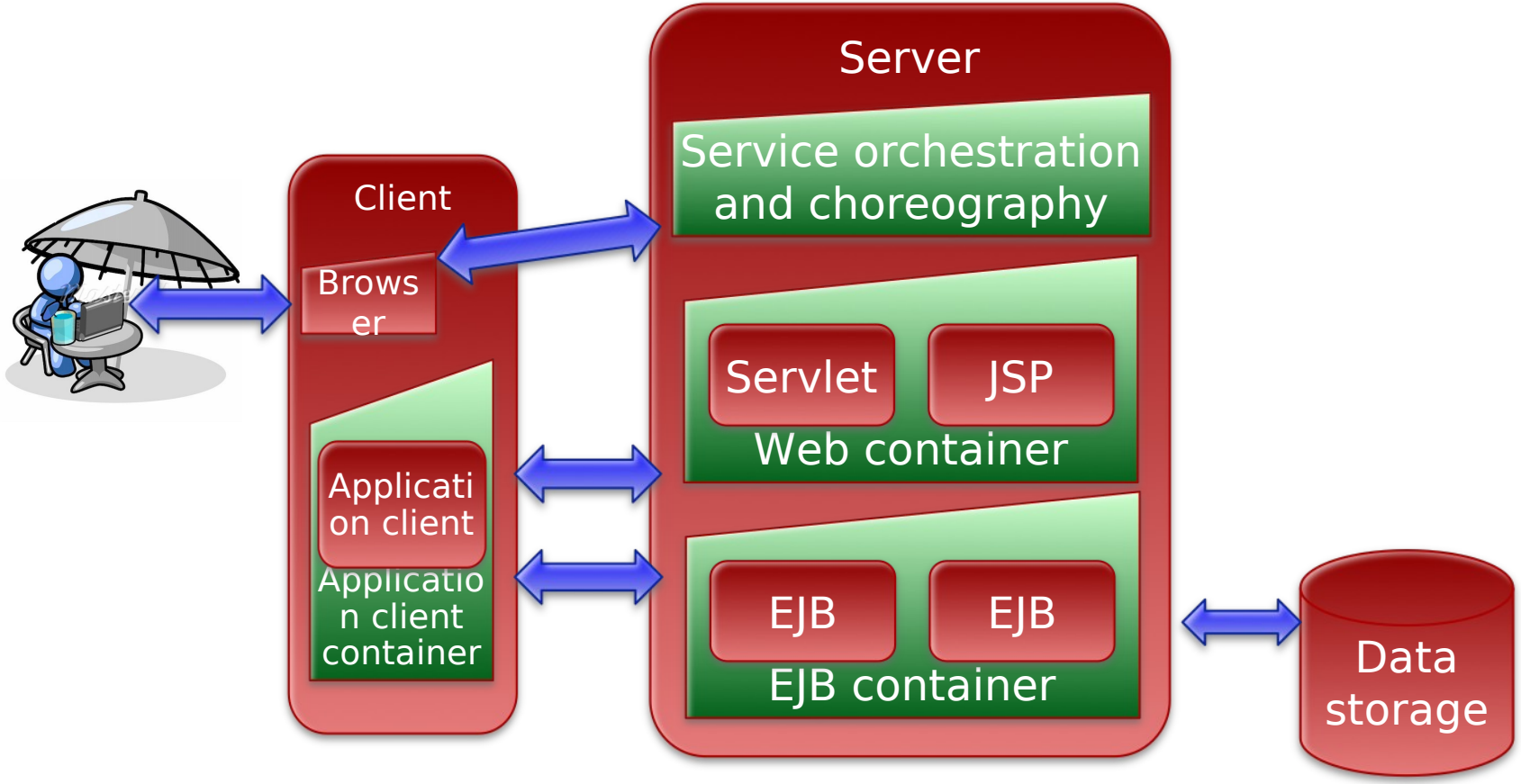

http://www.example.com/path/file.html
Client
Browse
r
file.html
Web Server
GET /path/file.html HTTP/1.1
Host: www.example.com
/home/www/path/file.html
Data
storage

# Code / script / application execution

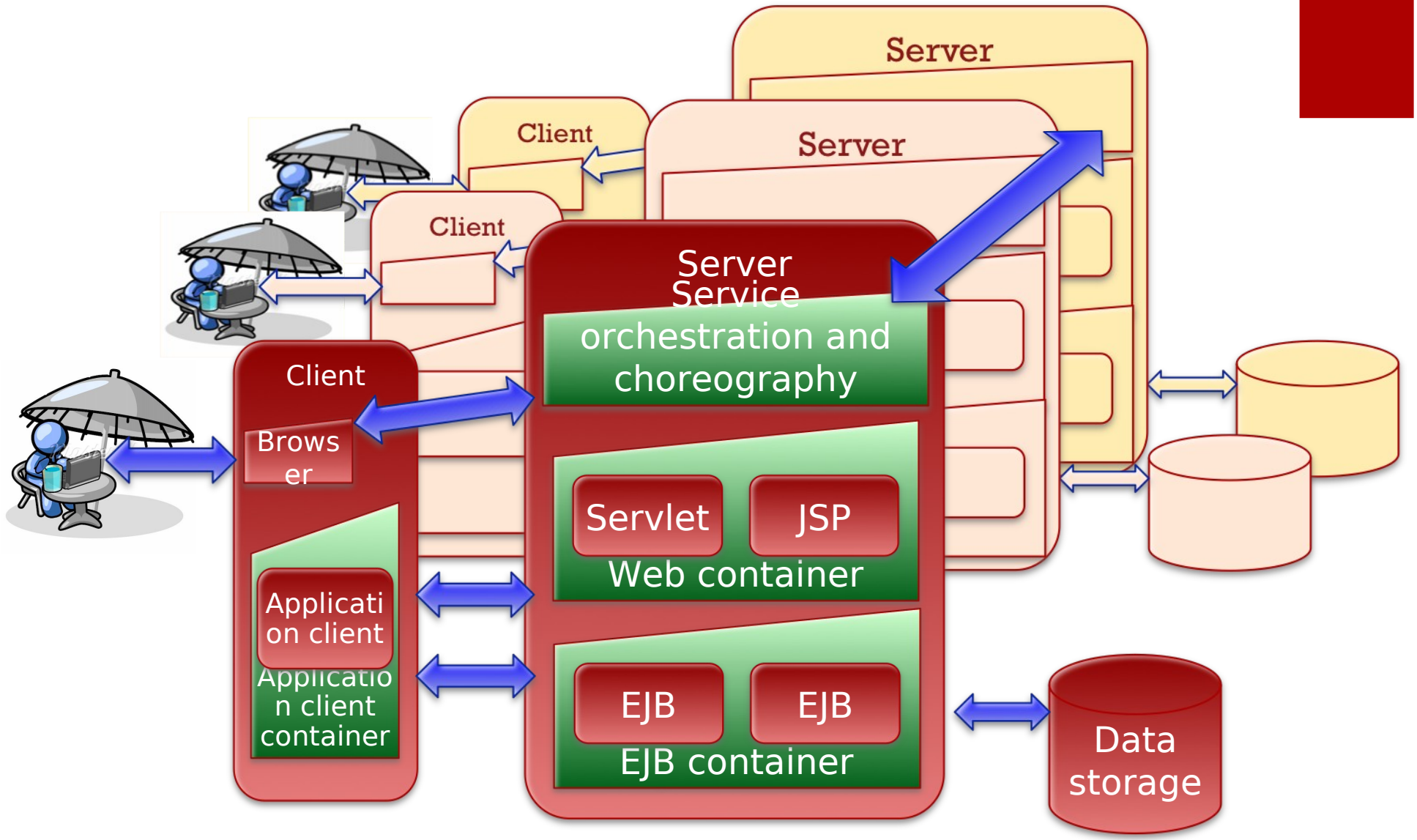



JSP = JavaServer Pages
EJB = Enterprise Java Beans

# Service execution (1/2)



# Service execution (2/2)



# Some SOA definitions (1/2)

A **Service-Oriented Architecture (SOA)** facilitates the creation of flexible, re-usable assets for enabling end-to-end business solutions. *(Open Group Standard: SOA Reference Architecture, 2011)*

Contemporary **SOA** represents an open, agile extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services. *(Erl, T., Service-oriented Architecture: Concepts, Technology and Design, 2005)*

**Service-Oriented Architecture** is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that can be combined and reused quickly to meet business needs. (*BEA white paper, 2005 -> 2008 Oracle)*

**SOA** is a conceptual business architecture where business functionality, or application logic, is made available to SOA users, or consumers, as shared, reusable services on an IT network. “Services” in an SOA are modules of business or application functionality with exposed interfaces, and are invoked by messages. *(Marks, E.A., Bell, M., Service Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology, 2006)*

# Some SOA definitions

**Service-oriented architecture (SOA)** is a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functionalities that are built as software components (discrete pieces of code and/or data structures) that can be reused for different purposes. SOA design principles are used during the phases of systems development and integration. *(Wikipedia, accessed 2012)*

**SOA** is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. *(O'Reilly XML.COM, accessed 2012)*

# What is SOA

SOA is an architectural style,

realized as a collection of collaborating agents,

each called a service,

whose goal is to **manage complexity** and **achieve architectural resilience and robustness** through ideas such as **loose coupling**, **location transparency**, and **protocol independence**.

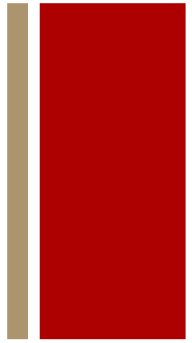
# SOA, no single definition –
## "SOA is different things to different people"

- A set of services that a business wants to expose to their customers and partners, or other portions of the organization.
- An architectural style which requires a service provider, a service requestor (consumer) and a service contract (a.k.a. client/server).
- A set of architectural patterns such as enterprise service bus, service composition, and service registry, promoting principles such as modularity, layering, and loose coupling to achieve design goals such as separation of concerns, reuse, and flexibility.
- A programming and deployment model realized by standards, tools and technologies such as Web services and Service Component Architecture (SCA).

# Service

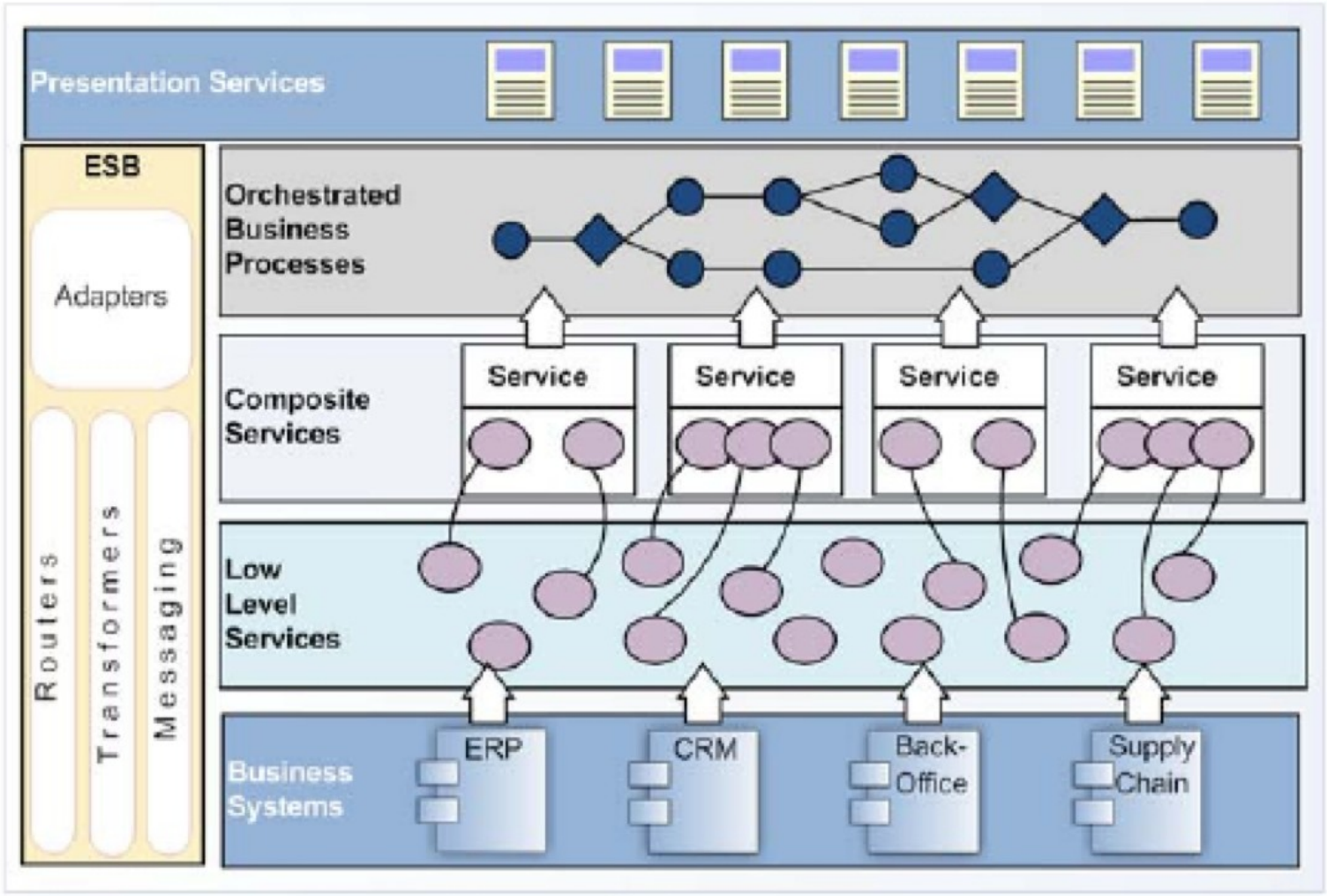
- A service is an entity that has a description, and that is made available for use through a published interface that allows it to be invoked by a service consumer.
- A service in SOA is an exposed piece of functionality with three properties:
  - The interface contract to the service is platform-independent.
  - The service can be dynamically located and invoked.
  - The service is self-contained. That is, the service maintains its own state.

# Principles of SOA

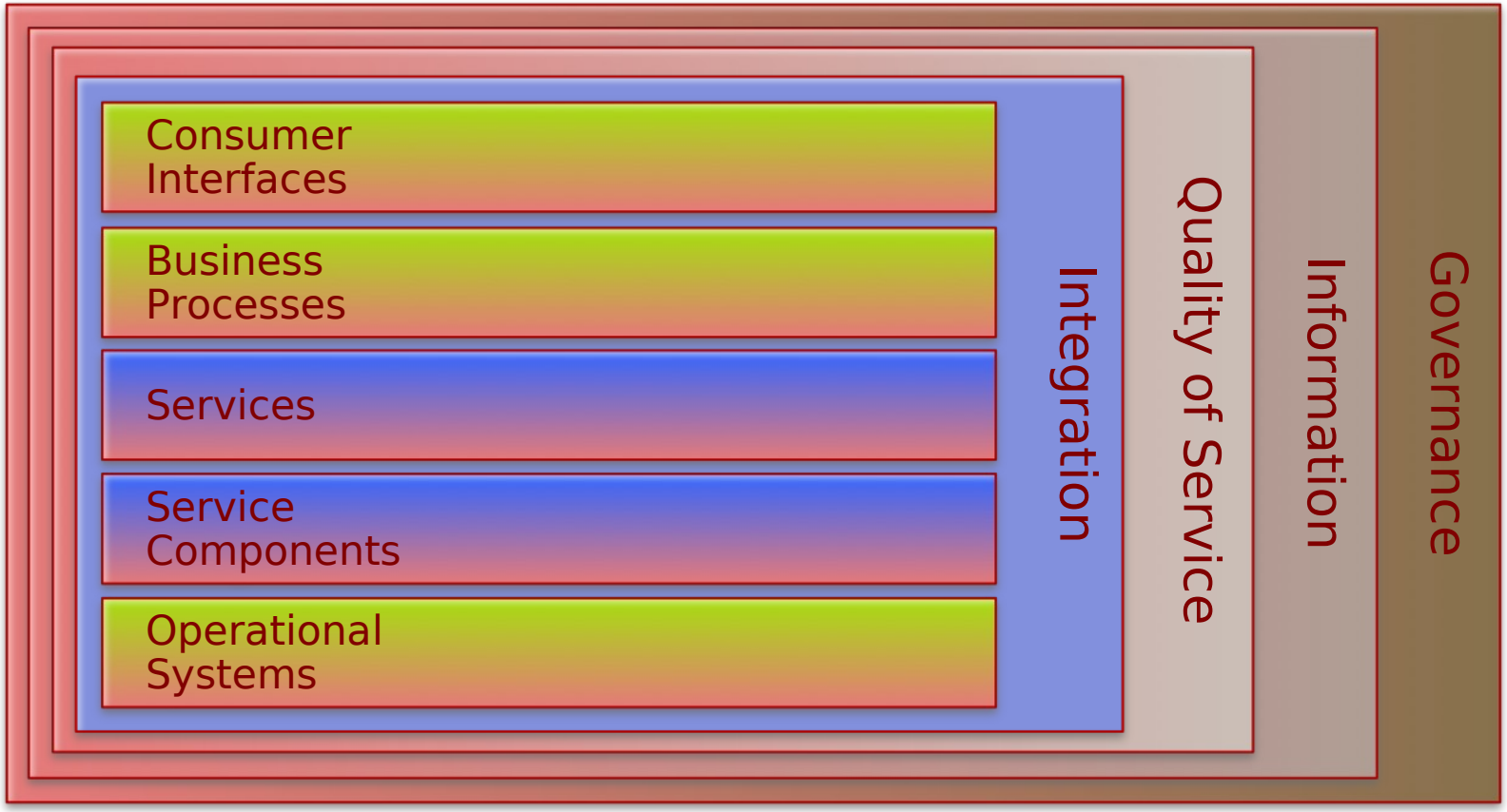


- Services
  - Share a formal contract
  - Are loosely coupled
  - Abstract underlying logic
  - Are composable
  - Are reusable
  - Are autonomous
  - Are stateless
  - Are discoverable

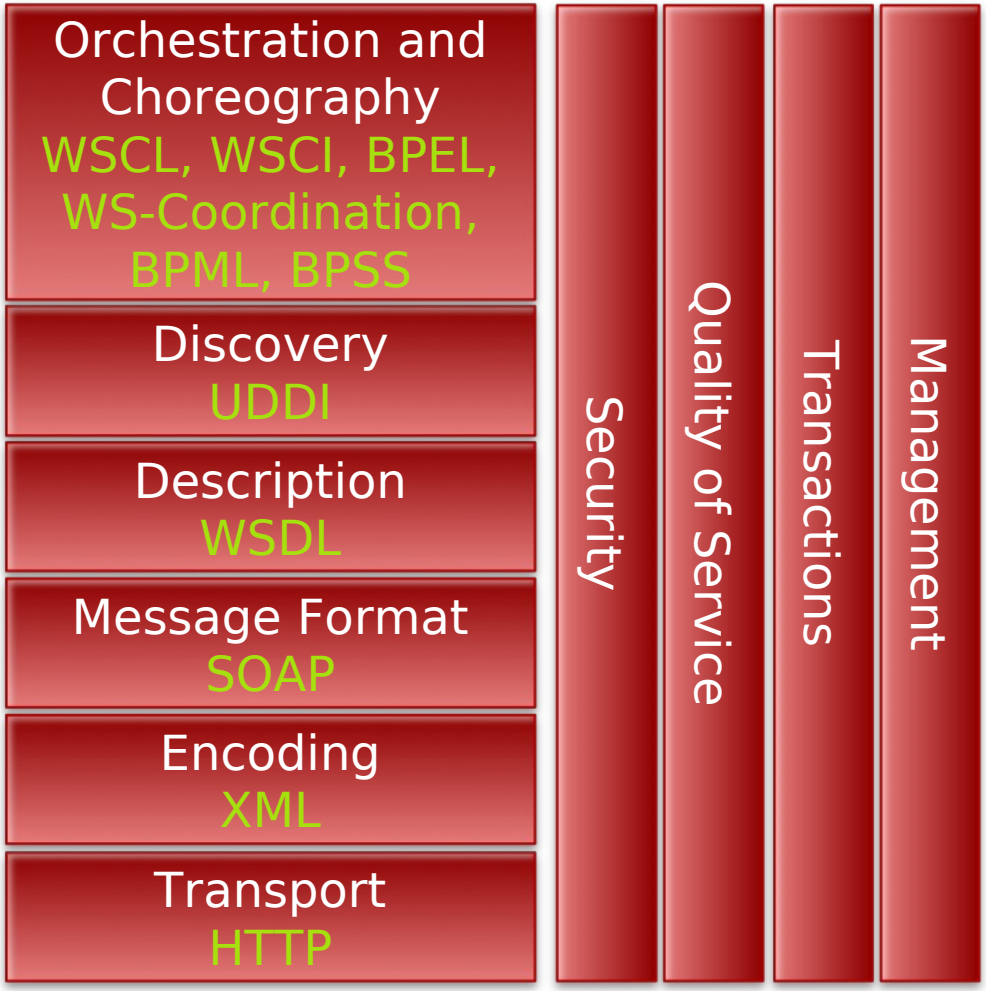
# A SOA Characterization



# A SOA Application development view: SOA Reference Architecture



# A SOA Technology view: WS* Protocol Stack




| | |
|---|---|
| WSCL | Web Services Conversation Language |
| WSCI | Web Service Choreography Interface |
| BPEL | Business Process Execution Language |
| WS | Web Services |
| BPML | Business Process Modeling Language |
| BPSS | Business Process Specification Schema |
| UDDI | Universal Description, Discovery and Integration |
| WSDL | Web Services Description Language |
| SOAP | Simple Object Access Protocol |
| XML | eXtensible Markup Language |
| HTTP | Hypertext Transfer Protocol |

# Why

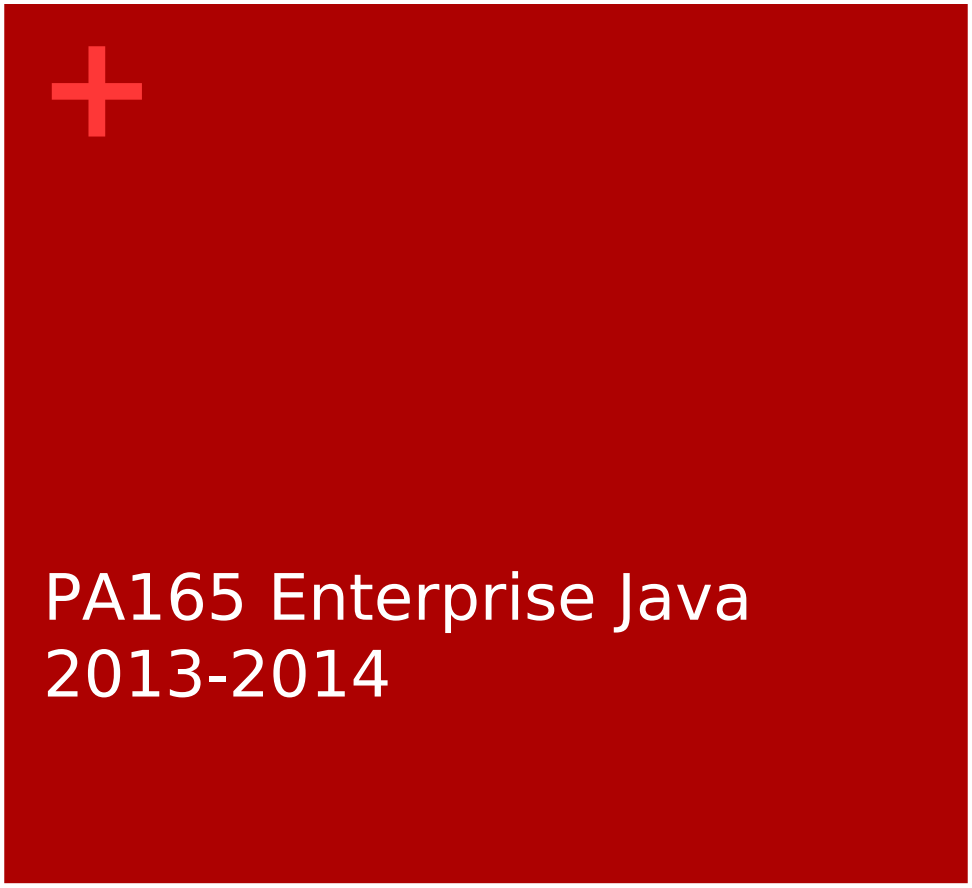
- “The quest is to find a solution that simplifies development and implementation, supports effective reuse of software assets, and leverages the enormous and low-cost computing power now at our fingertips. While some might claim that service-oriented architecture (SOA) is just the latest fad in this illusive quest, tangible results have been achieved by those able to successfully implement its principles”

- “companies that have embraced SOA have eliminated huge amounts of redundant software, reaped major cost savings from simplifying and automating manual processes, and realized big increases in productivity”

  *(Open Source SOA, Jeff Davis)*

# Summary

- The purpose of this introduction is to give an overall impression of SOC and SOA, somewhat compromising the technical accuracy in favor of getting a global view
- SOC is rapidly expanding to new areas and applications of computing
- SOA standardization pins down commonly accepted principles, but different solutions and proposals expand the area continuously
- Service orientation is establishing itself as “next generation of application development approach”




# REpresentational State Transfer (REST)

Bruno Rossi & Juha Rikkilä

# Objectives and content

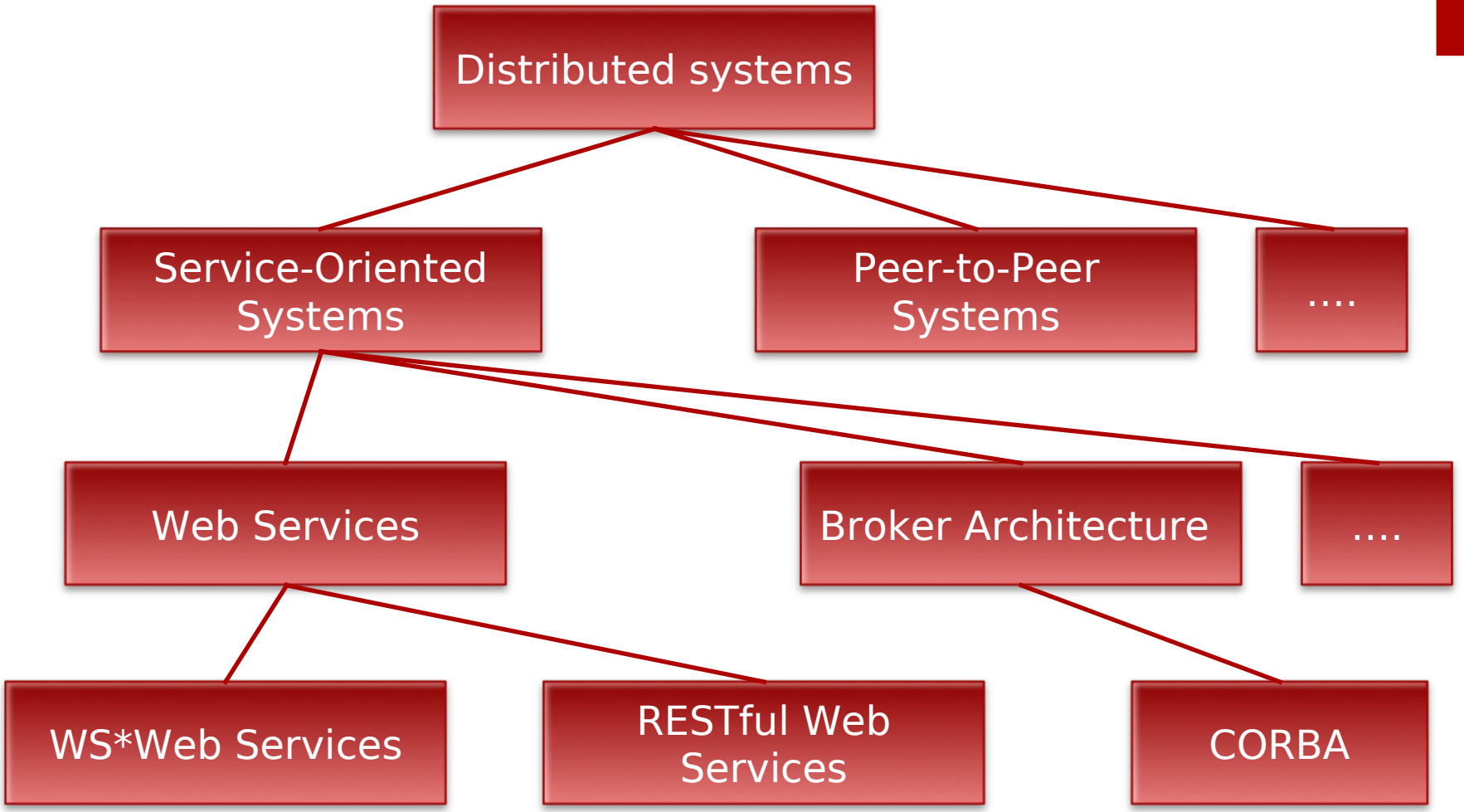
## Objectives

Obtain overall understanding of the REST architectural style and its implementation in web.

## Content

- Distributed systems
- REST, RESTFUL
- URI
- HTTP, HTTP methods
- Cache, Proxy, Gateway
- Security
- Summary, the six constraints, the principles of the uniform interface

# Distributed Systems




REST=Representational State Transfer

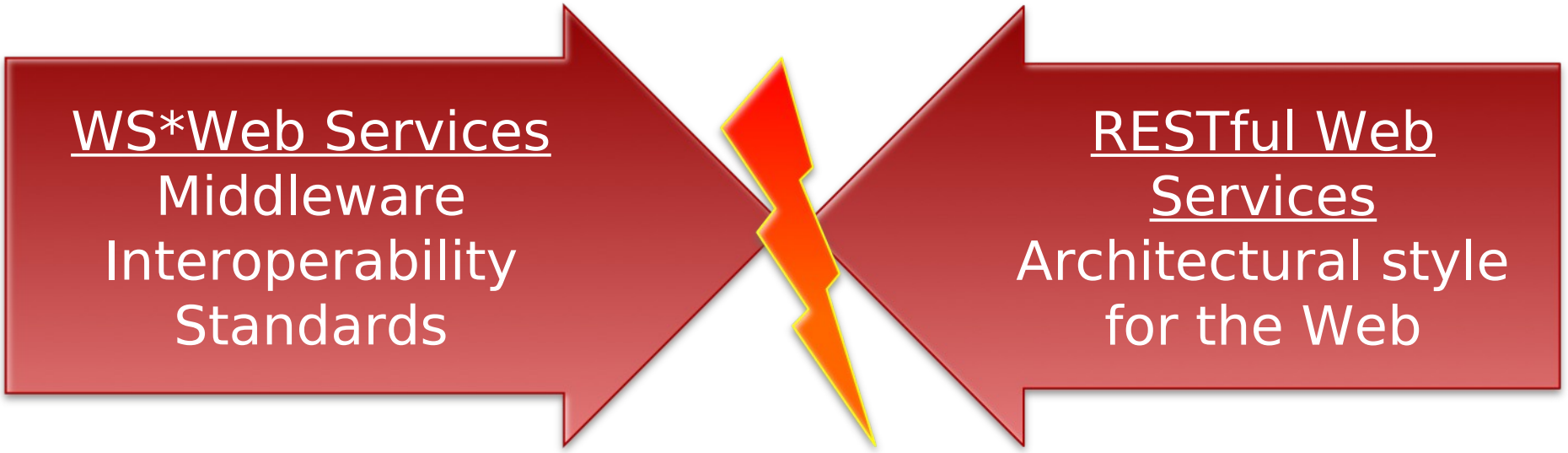
# REST

- REpresentational State Transfer
  - Name by Roy Fielding in his Ph.D thesis*
- Architectural Style for the Web

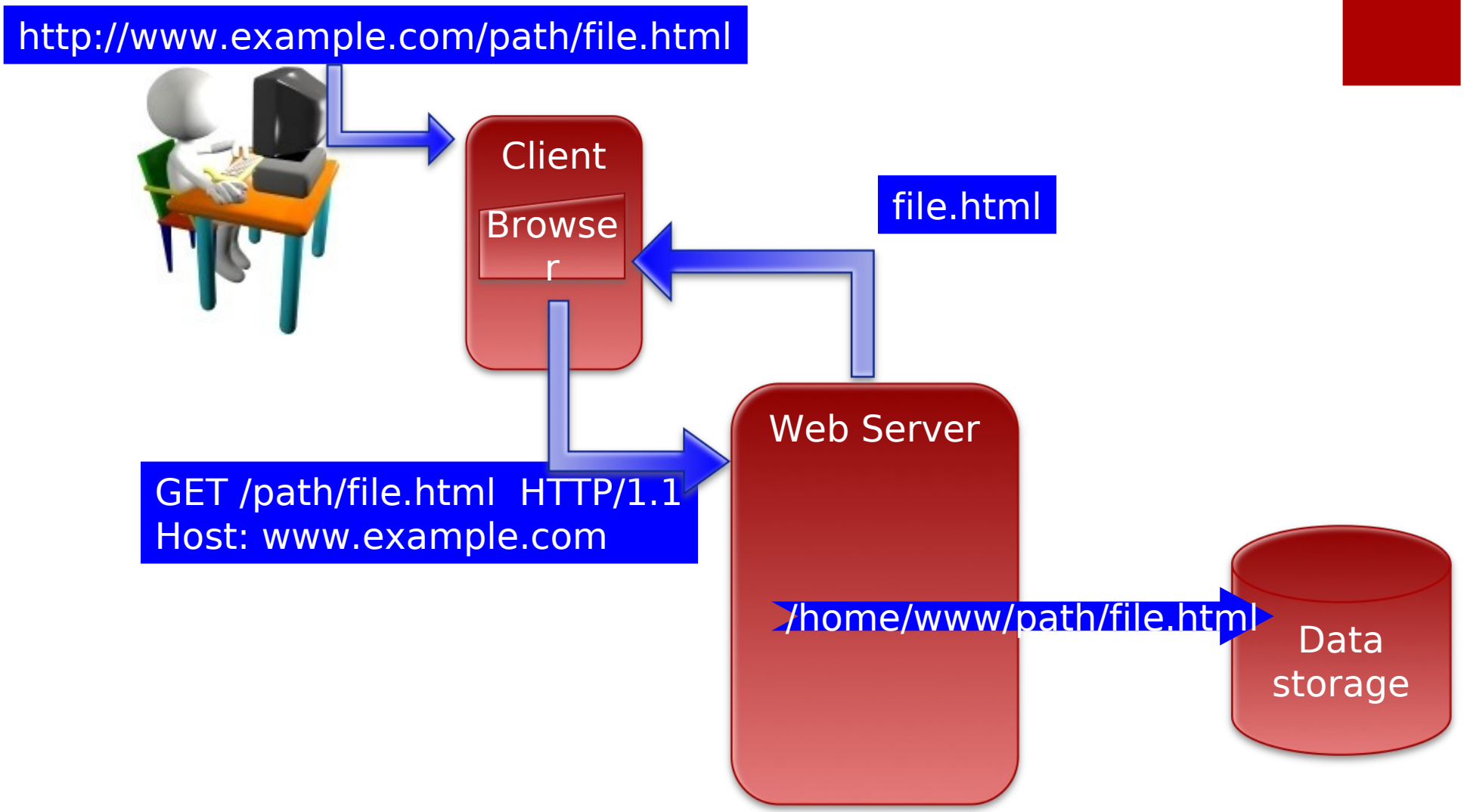
* http://ics.uci.edu/~fielding/pubs/dissertation/top.htm

- Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: presented with a network of Web pages (a virtual state-machine), the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for his use. Fielding's PhD thesis, section 6.1
- REST was initially described in the context of HTTP, but it is not limited to that protocol.

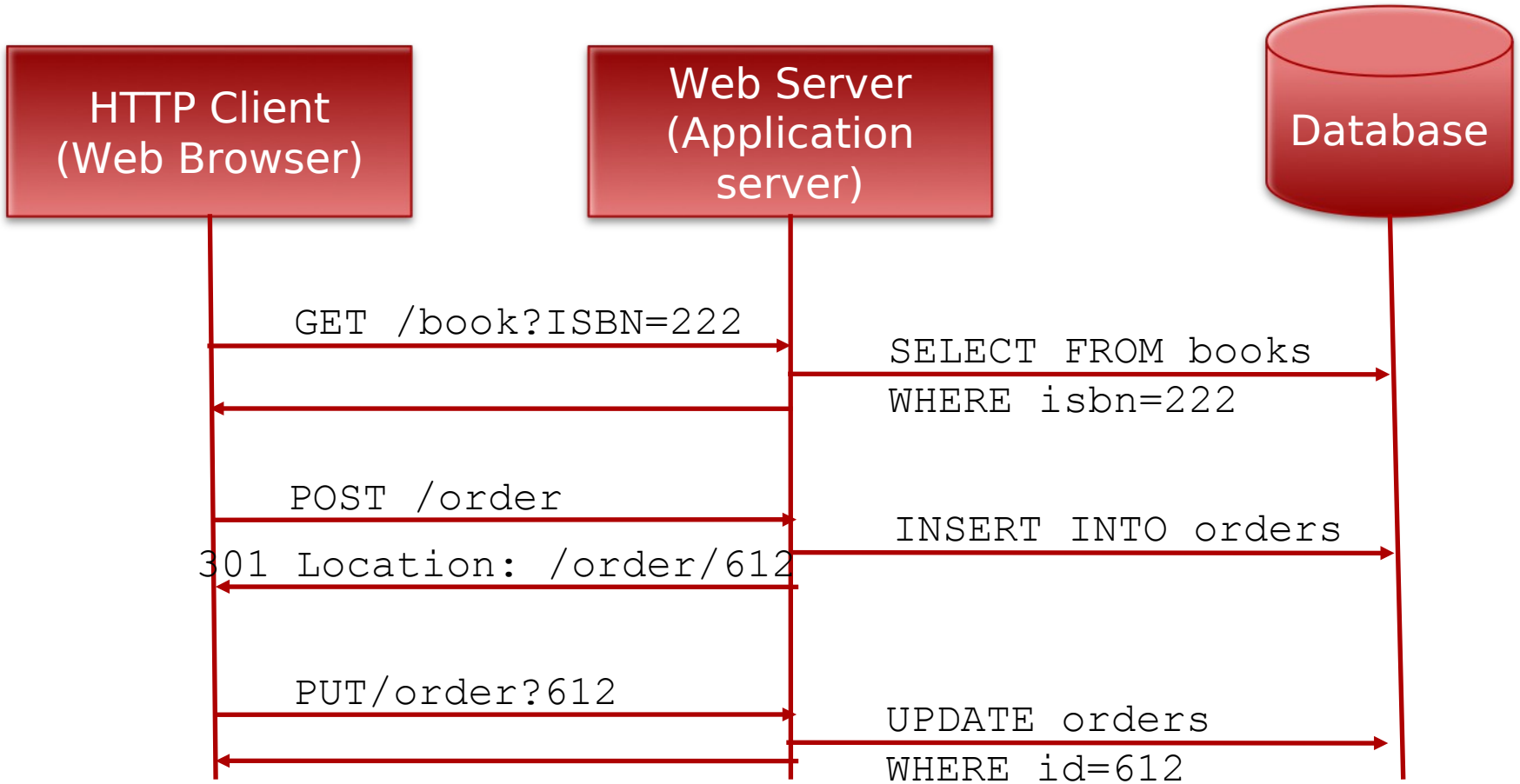
# WS* vs. RESTful Web services



# Browsing


http://www.example.com/path/file.html
Client
Browse
r
file.html
Web Server
GET /path/file.html HTTP/1.1
Host: www.example.com
/home/www/path/file.html
Data
storage

# An example


HTTP Client
(Web Browser)
Web Server
(Application
server)
Database
GET /book?ISBN=222
SELECT FROM books
WHERE isbn=222
POST /order
INSERT INTO orders
301 Location: /order/612
PUT/order?612
UPDATE orders
WHERE id=612

# Characteristics of REST

- In REST system, resources are manipulated through the exchange of representations of the resources
  - For example, a purchase order resource is represented by an XML document.
  - Within a RESTful purchasing system, a purchase order might be updated by posting an XML document containing the changed purchase order to its URI
- REST-based architectures communicate primarily through the transfer of representations of resources
  - State is maintained within a resource representation

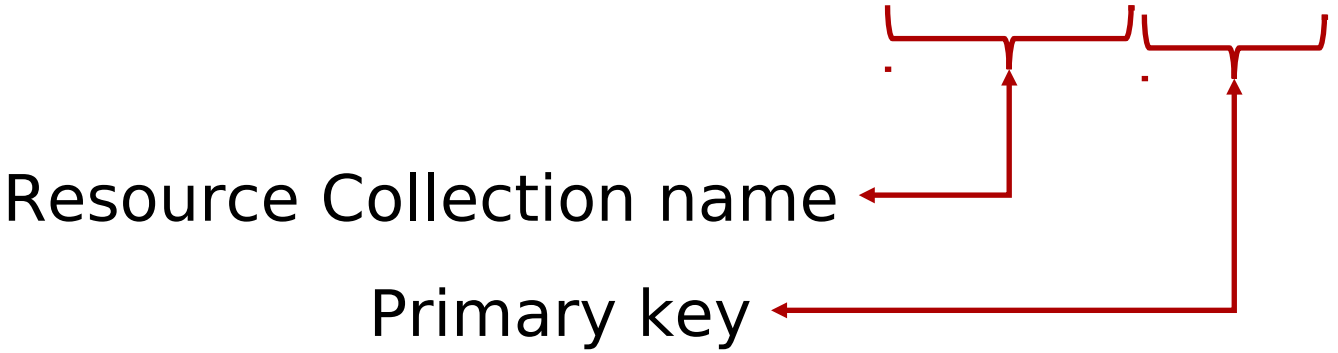
# Characteristics of REST

- RESTful services are stateless
  - Each request from client to server must contain all the information necessary to understand the request
- RESTful services have a uniform interface
  - GET, POST, PUT, and DELETE.
- REST-based architectures are built from resources (pieces of information) that are uniquely identified by URIs
  - In a RESTful purchasing system, each purchase order has a unique URI

# URI, two examples

1.

`http://www.sun.com/servers/blades/t6300`

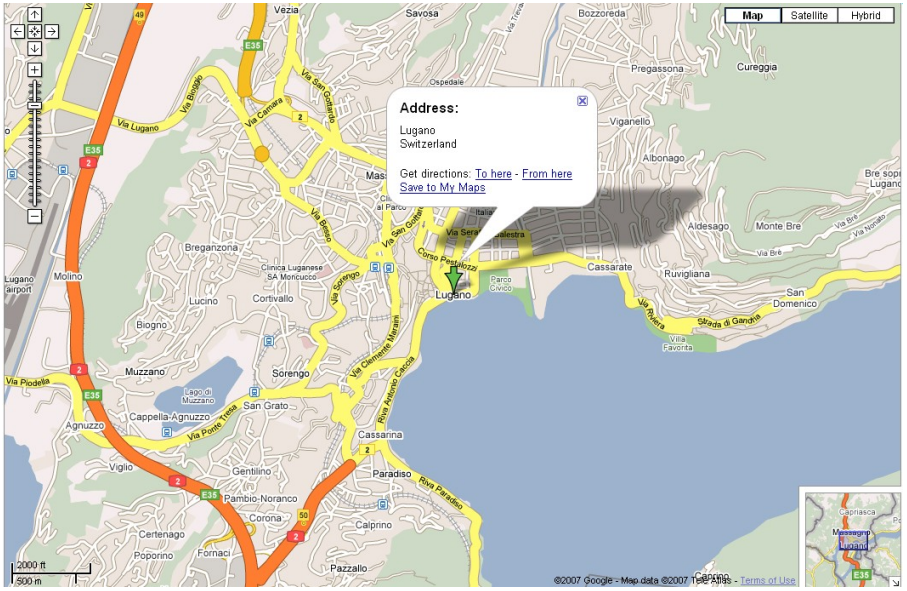



2.

http://localhost:8080/MyWebservice/rest/hello

# URI, another example

http://maps.google.com/lugano




http://maps.google.com/maps?f=q&hl=en&q=lugano,
+switzerland&layer=&ie=UTF8&z=12&om=1&iwloc=addr

# URI templates

Template:

- http://www.myservice.com/order/{oid}/item/{iid}
  - http://www.myservice.com/order/XYZ/item/12345

Template:

- http://www.google.com/search?{-join|&|q,num}
  - http://www.google.com/search?q=REST&num=10

From http://bitworking.org/projects/URI-Templates/

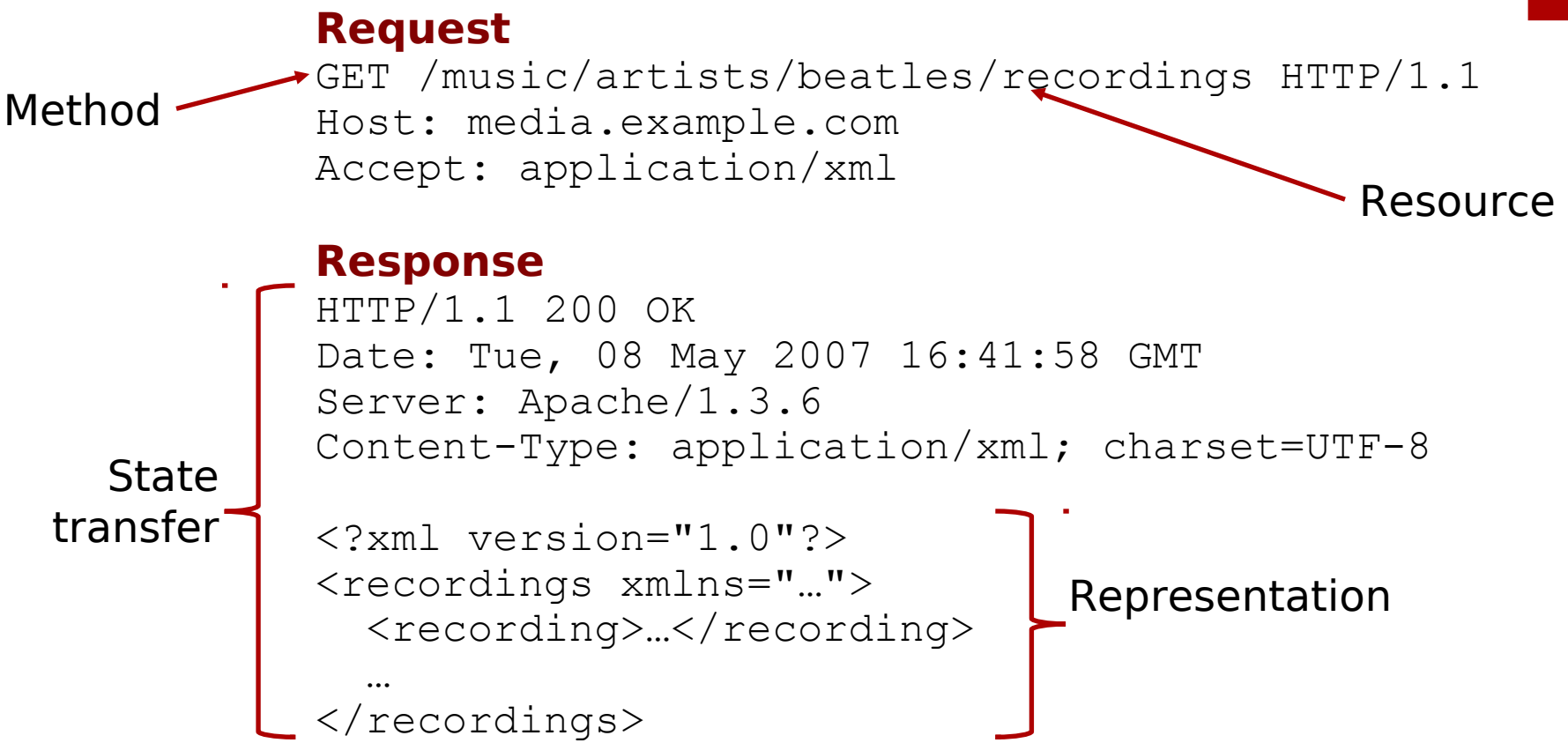
# Multiple Represenations

- Data in a variety of formats
  - XML
  - JSON (JavaScript Object Notation)
  - (X)HTML
- Content negotiation
  - Accept header

    ```
    GET /foo
    Accept: application/json
    ```
  - URI-based

    ```
    GET /foo.json
    ```

# HTTP Request/Response As REST



# HTTP Methods, for both collection and single item

## GET
- to retrieve information
- Retrieves a given URI
- Idempotent, should not initiate a state change
- Cacheable

## PUT
- to update information
- Full entity create/replace used when you know the “id”

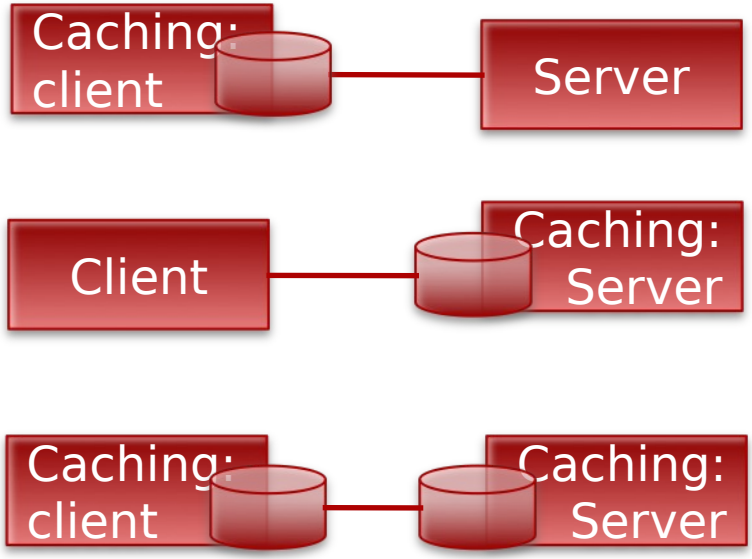
## POST
- to add new information
- Add the entity as a subordinate/append to the POSTed resource

## DELETE
- to remove (logical) an entity

# Caching

## Basic setup


## Caching options



# Proxy / Gateway

- Proxy / gateway forward (and may cache/manipulate/control) requests and responses
- May be used for composing services

A proxy is chosen by the Client (for caching, or access control)

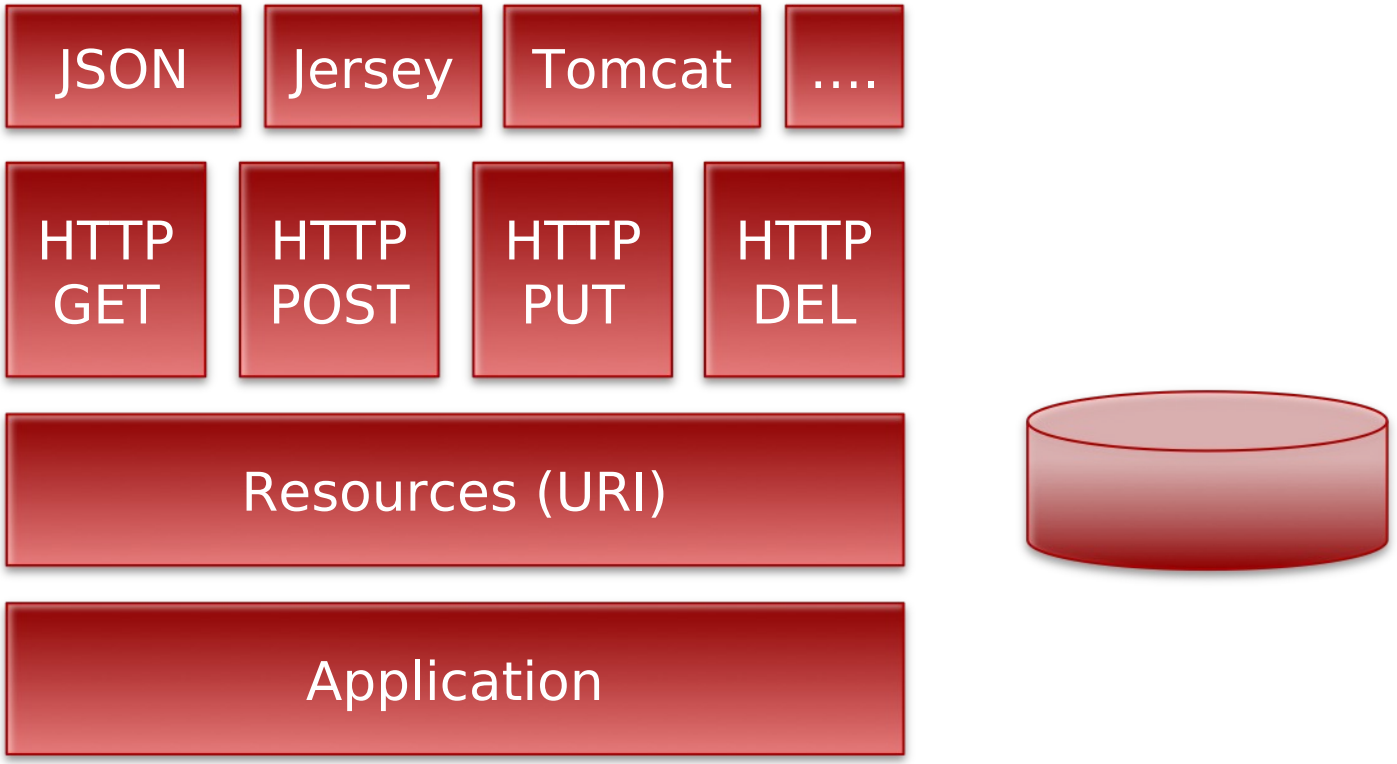

The use of a gateway is imposed by the server

# REST security

- HTTPS (HTTP + SSL/TLS)
- HTTP Basic Authentication
- when using XML content
  - XML Encryption
  - XML Signature
- Secure, point to point communication
  - (Authentication, Integrity and Encryption)

# Summary

- “The Web is the universe of globally accessible information” (Tim Berners Lee)
- Applications publish their data on the Web through URI



# The six constraints (1/3)

Summarizing, the REST architectural style describes the six constraints applied to the architecture:

1. Client–server

A uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, or servers are not concerned with the user interface or user state

2. Stateless

- No client context is stored on the server between requests. Each request from any client contains all of the information necessary to service the request, and any session state is held in the client.

Wikipedia, Oct. 2012

# The six constraints (2/3)

3. Cacheable

- Clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests.

4. Layered system

- A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. They may also enforce security policies.

Wikipedia, Oct. 2012

# The six constraints (3/3)

5. Code on demand (optional)
   - Servers are able temporarily to extend or customize the functionality of a client by the transfer of executable code. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

6. Uniform interface
   - The uniform interface between clients and servers simplifies and decouples the architecture, which enables each part to evolve independently.
   - The four guiding principles of this interface on the next slides.

Wikipedia, Oct. 2012

# The uniform interface (1/3)

- The uniform interface that any REST interface must provide

1. Identification of resources
   - Individual resources are identified in requests, for example using URIs in web-based REST systems. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server does not send its database, but some HTML, XML or JSON that represents some database records expressed in some indicated language and encoding, depending on the details of the request and the server implementation.

Wikipedia, Oct. 2012

# The uniform interface (2/3)

2. Manipulation of resources through these representations
   - When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.

3. Self-descriptive messages
   - Each message includes enough information to describe how to process the message. For example, the parser may be specified by an Internet media type, the cacheability of responses.

Wikipedia, Oct. 2012

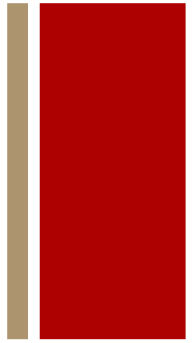
# The uniform interface (3/3)

4. Hypermedia as the engine of application state (aka HATEOAS)

- Clients make state transitions only through actions that are dynamically identified within hypermedia by the server (e.g., by hyperlinks within hypertext). In general a client does not assume that any particular action is available for any particular resources beyond those described in representations previously received from the server.

Wikipedia, Oct. 2012

# Let's dig into the details: Oracle Tutorials on RESTful Services with JAX-RS



Web Services:

http://docs.oracle.com/javaee/7/tutorial/doc/partwebsvcs.htm

Building RESTful Web Services with JAX-RS:

http://docs.oracle.com/javaee/7/tutorial/doc/jaxrs.htm#GIEPU

Accessing REST Resources with the JAX-RS Client API:

http://docs.oracle.com/javaee/7/tutorial/doc/jaxrs-client.htm#BABEIGIH