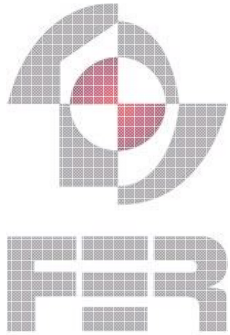
UNIZG-FER 86518

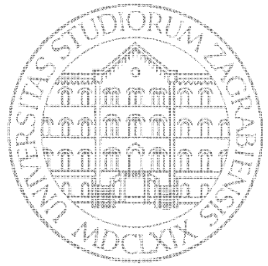
Service-Oriented Computing

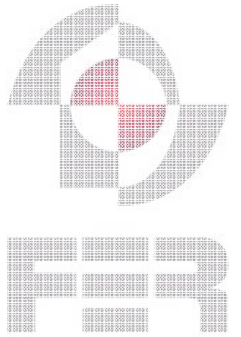


REST

REpresentational State Transfer

Introduction to REST

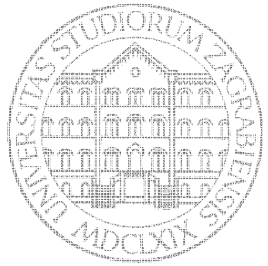


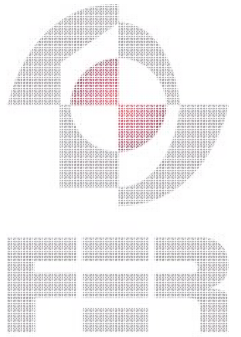


- December of 1990
  - Tim Berners-Lee
  - A non-profit software project called "***WorldWideWeb***"
  - To facilitate the sharing of knowledge

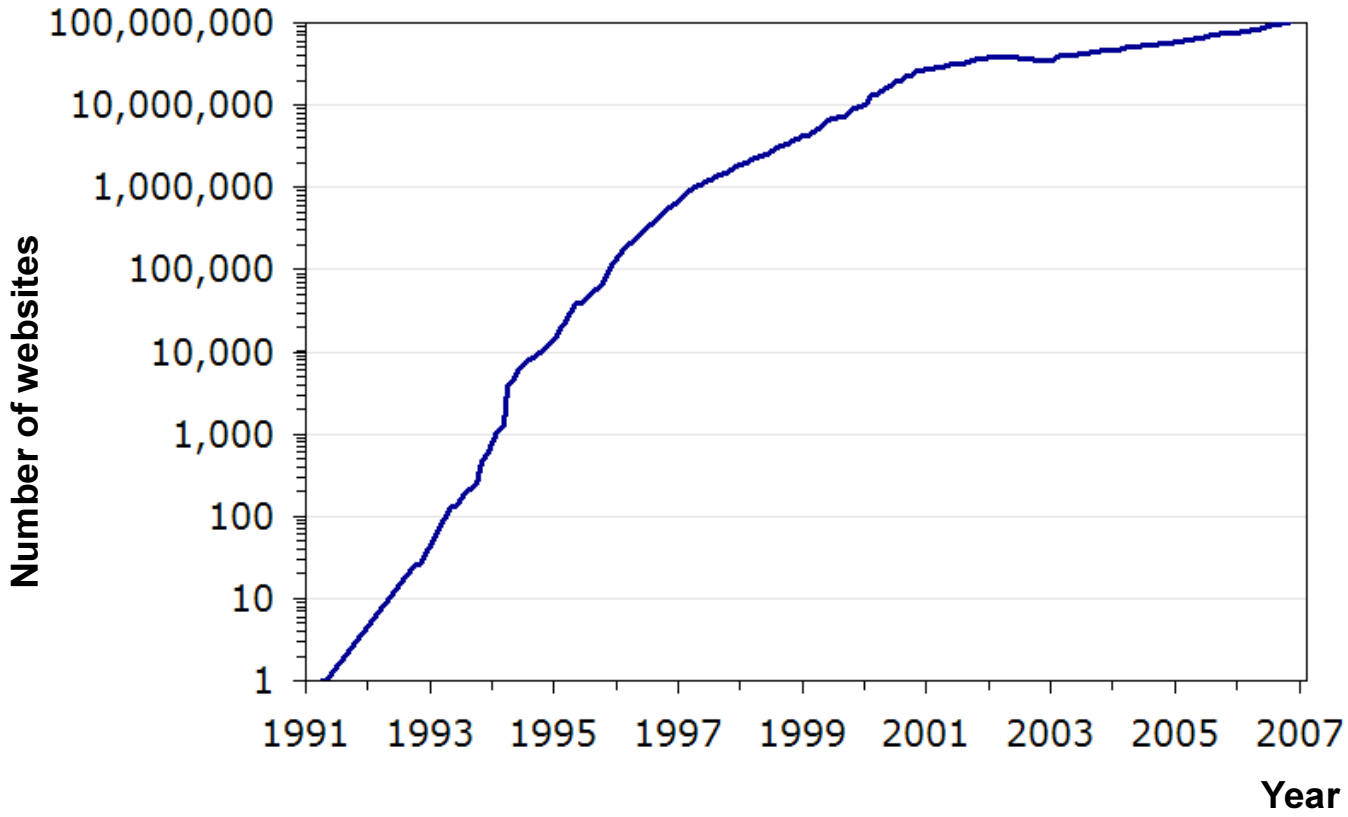
- Basic set of technologies, tools, and applications to make the system usable
  - The Uniform Resource Identifier (URI)
    - A syntax that assigns each web document a unique address
  - The HyperText Transfer Protocol (HTTP)
    - A message-based language that computers use to communicate over the Internet
  - The HyperText Mark-up Language (HTML)
    - To represent informative documents that contain links to related documents
  - The first web server
  - The first web browser
  - The first WYSIWYG (*What You See Is What You Get*) HTML editor
    - built right into the browser

Introduction to REST

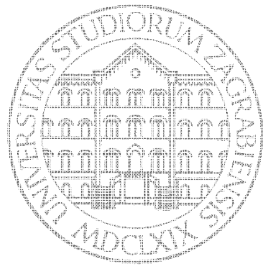


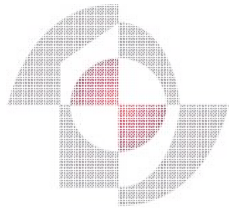


- World Wide Web growth
  - 1991-1997: **Explosive** growth, at a rate of **850%** per year
  - 1998-2001: **Rapid** growth, at a rate of **150%** per year
  - 2002-today: **Maturing** growth, at a rate of **25%** per year



Introduction to REST





- World Wide Web scalability problem
  - The traffic was outgrowing the capacity of the Internet infrastructure
  - The protocols were not uniformly implemented and they lacked support for caches and other stabilizing intermediaries
  - It was unclear if the Web would scale to meet the increasing demand
- Scalability
  - The ability of a system, network, or process to handle a growing amount of work in a capable manner

    or
  - The ability of a system, network, or process to be enlarged to accommodate that growth
- Scalable system
  - A system that is suitably efficient and practical under heavy load (e.g. a large input data set, a large number of users, a large number of nodes)

    or
  - A system whose performance improves after adding components, proportionally to the capacity added

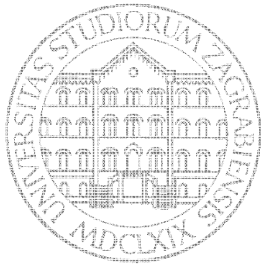
Introduction to REST

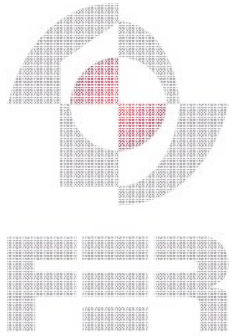
- REST (*Representational State Transfer*)
  - Style of software architecture for distributed systems
  - Architectural style
    - Not an architecture
    - Not a technology
  - Set of design criteria for building **scalable** distributed hypermedia systems

- Roy Thomas Fielding
  - Introduced REST as an architectural style
  - One of the principal authors of HTTP specification

Roy Thomas Fielding: *Architectural Styles and the Design of Network-Based Software Architectures*, Doctoral Dissertation, University of California, Irvine, 2000

Introduction to REST

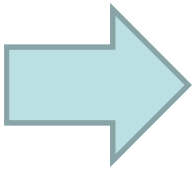




- RESTful
  - A property of system or an architecture that satisfies the principles of REST
- World Wide Web
  - An example of RESTful system
  - The largest implementation of RESTful system

Theoretical design framework

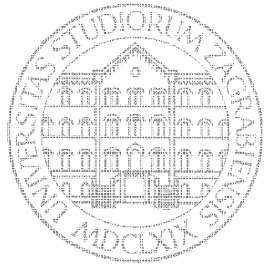
REST
- Constraints
- Properties
- Design criteria

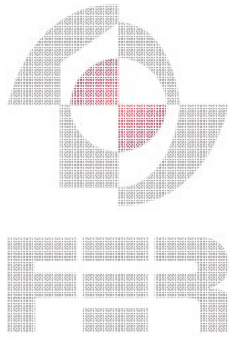


Technology and implementation

WWW
- HTTP
- URI
- Hypermedia

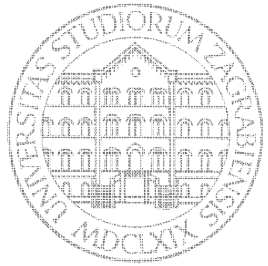
REST Constraints

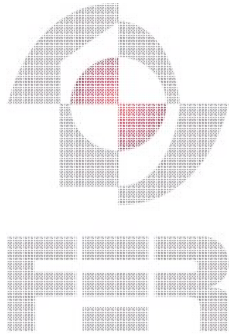




- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

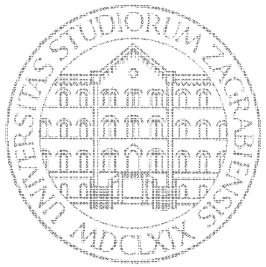
REST Constraints

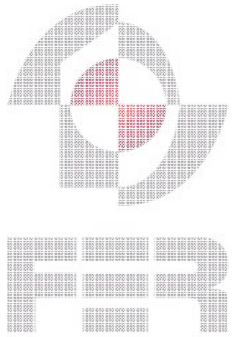




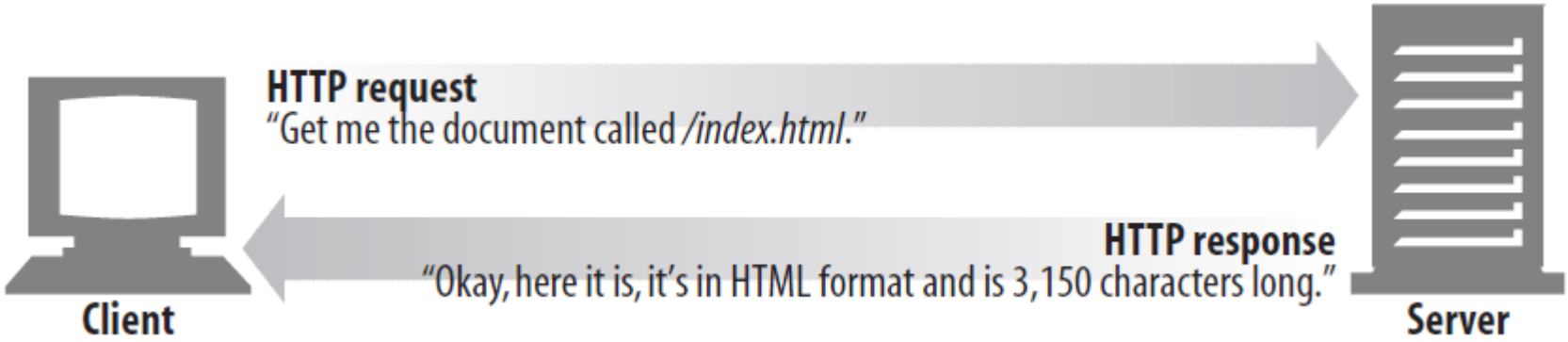
- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

Client-Server Architecture

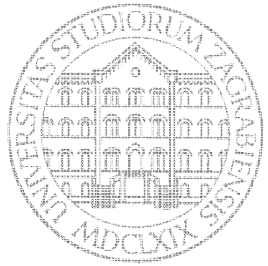


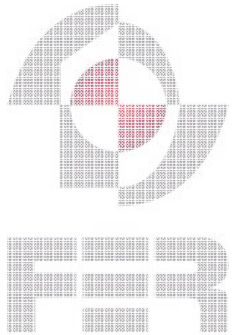


- Client-server
  - Two elementary components of every web-based system
  - The separation of concerns
  - Clients and servers have distinct parts to play
    - Servers host and serve documents
    - Clients format and display documents to the users
  - Clients and servers may be implemented and deployed independently, using any language or technology, so long as they conform to the uniform interface standards



Layered System

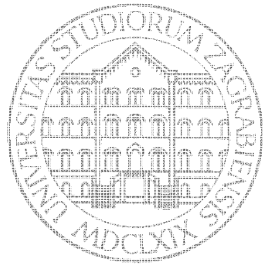


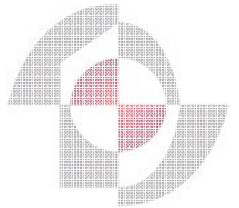


- Layered system
  - Enables network-based intermediaries to be transparently deployed between a client and a server
    - Proxy
    - Gateway
  - Optional components of web-based system
  - Network-based intermediary intercepts client-server communication for a specific purpose
    - Security
    - Caching
    - Load balancing
    - Protocol conversion

Scalability

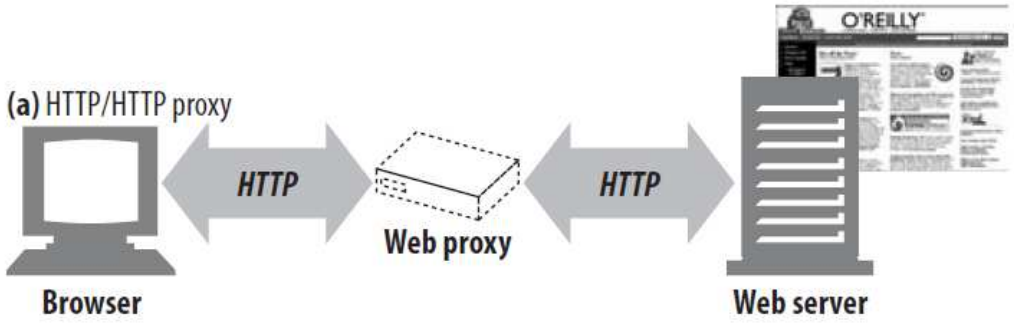
Layered System

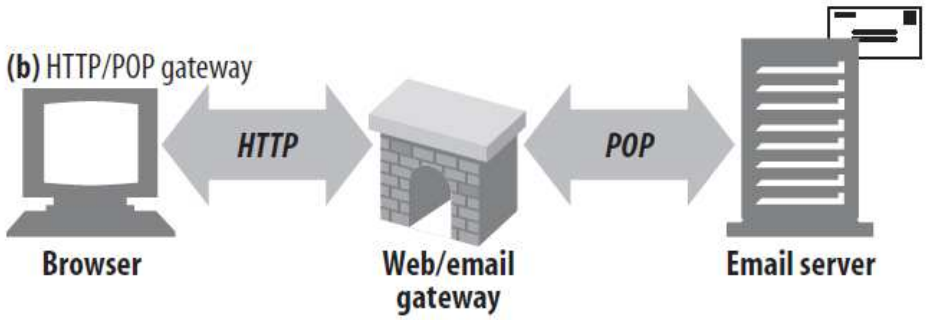




- Layered system
  - Proxy connects two or more applications that speak the same protocol
    - Security
    - Caching
    - Load balancing

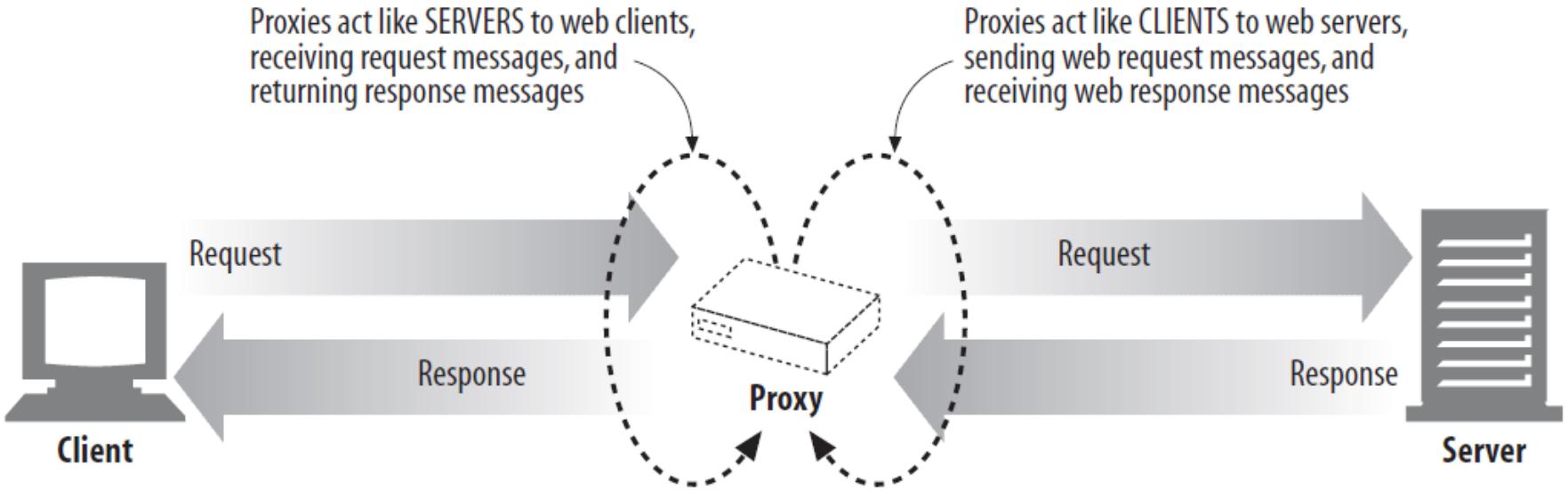
    Scalability
  - Gateway connects two or more applications that speak different protocols
    - Protocol conversion



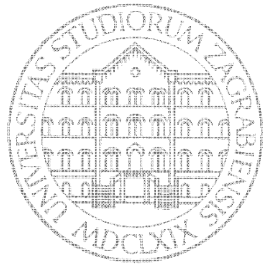


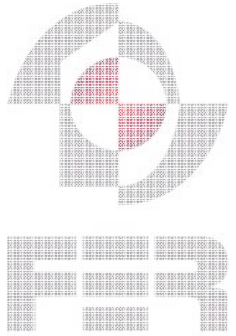
Layered System

- Proxy
  - Sits between a client and a server
  - Receives all of the client's HTTP requests and relays the requests to the server
    - Proxy may modify the request
  - Receives responses from server and forwards them back to the client
    - Proxy may modify the response

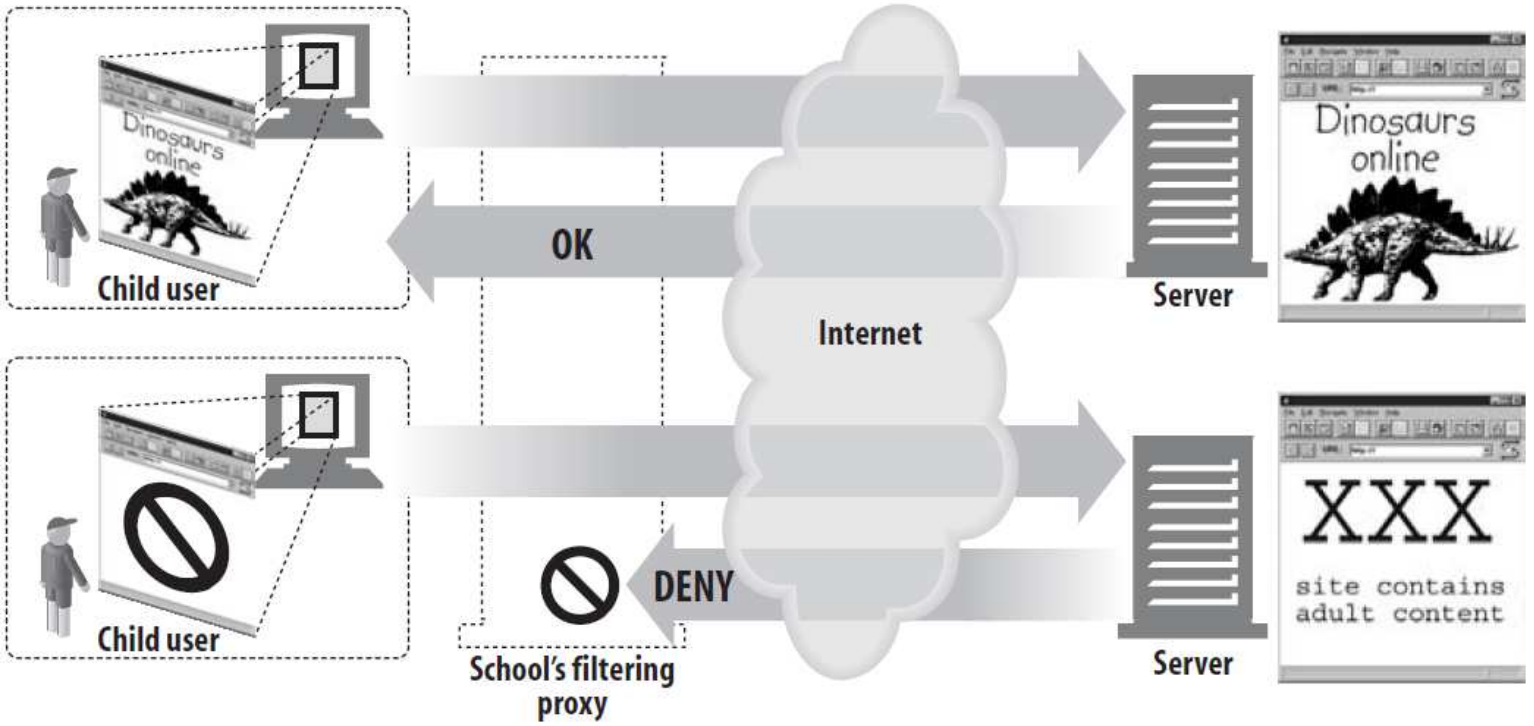


Layered System



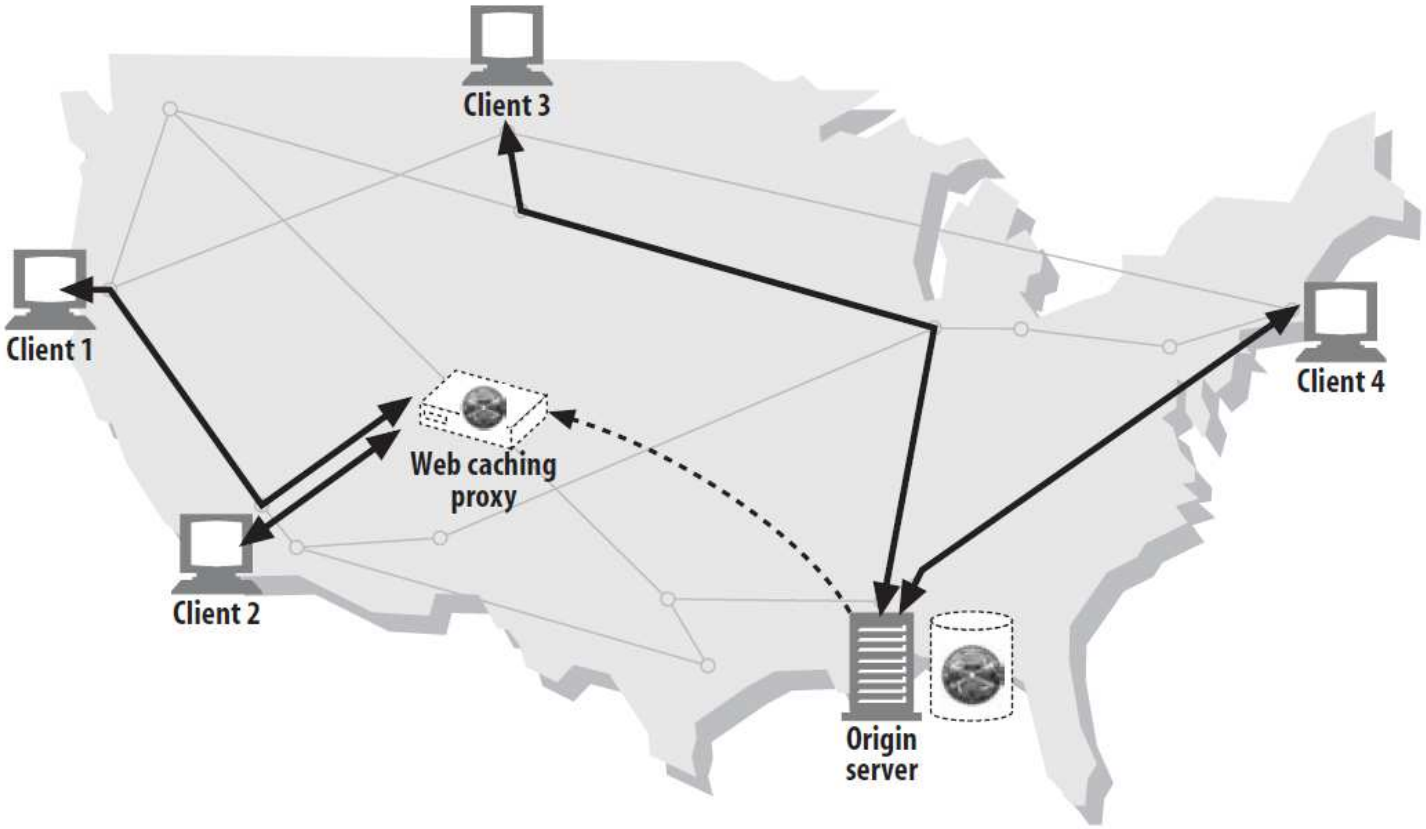


- Proxy usage
  - Security
    - Detection of application viruses in corporate downloads
    - Filtering adult content away from elementary-school students



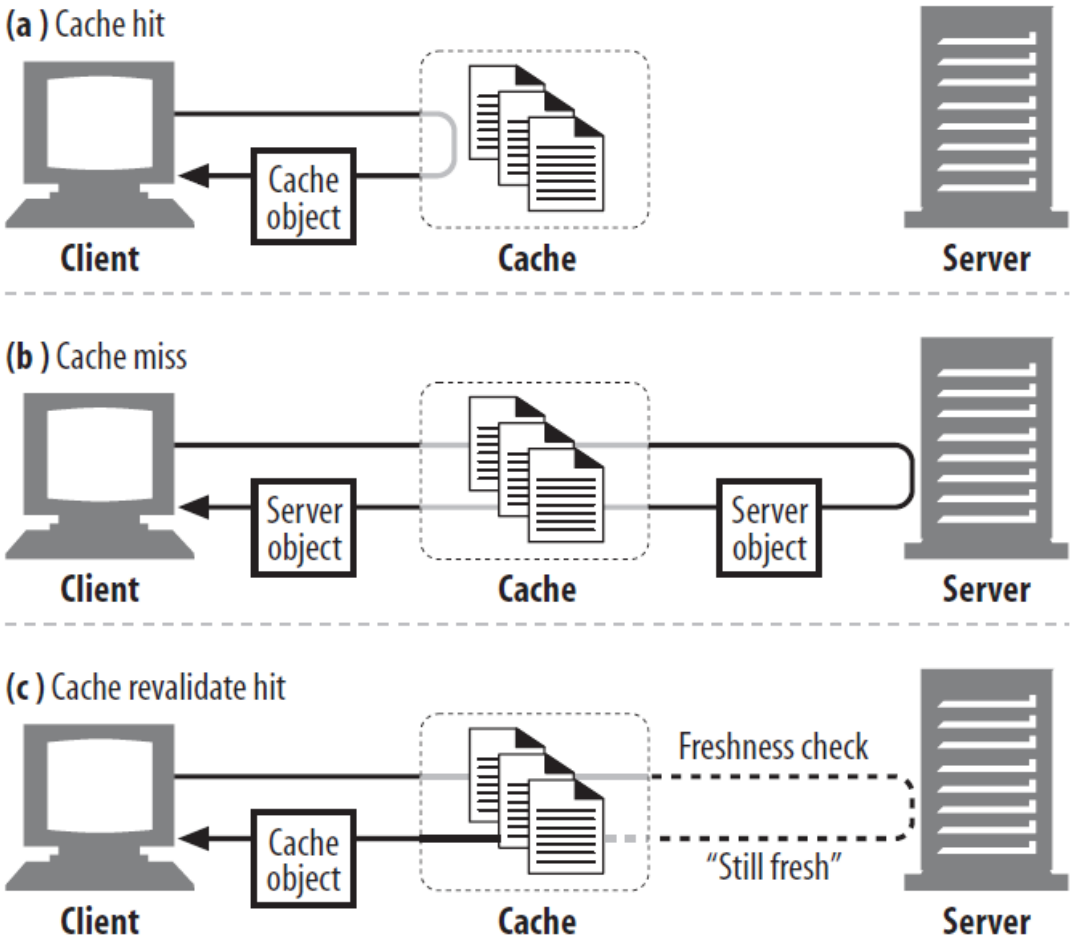
Layered System

- Proxy usage
  - Caching
    - Proxy deployed close to clients
    - Serves server objects stored in cache without contacting server
    - Improves network performance



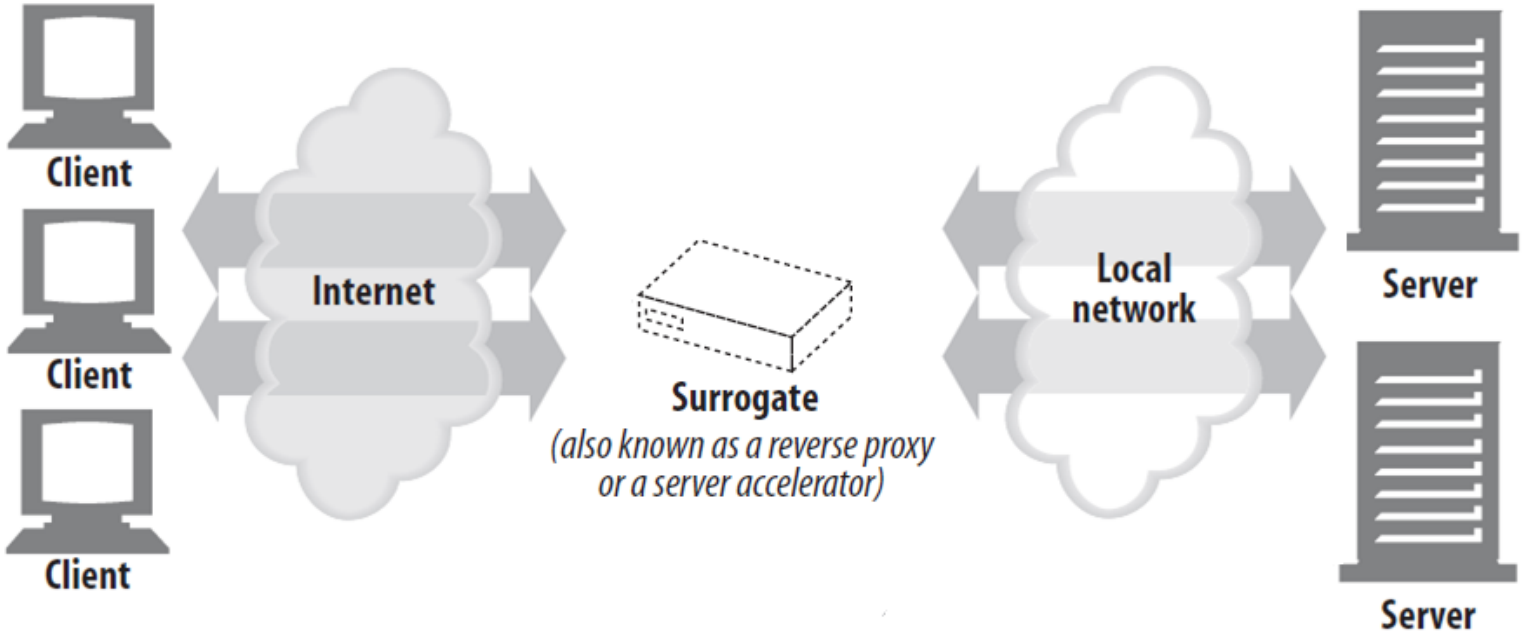
Layered System

- Proxy usage
  - Caching
    - Cache hits and cache misses

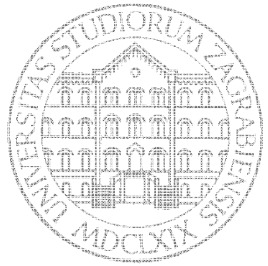


Layered System

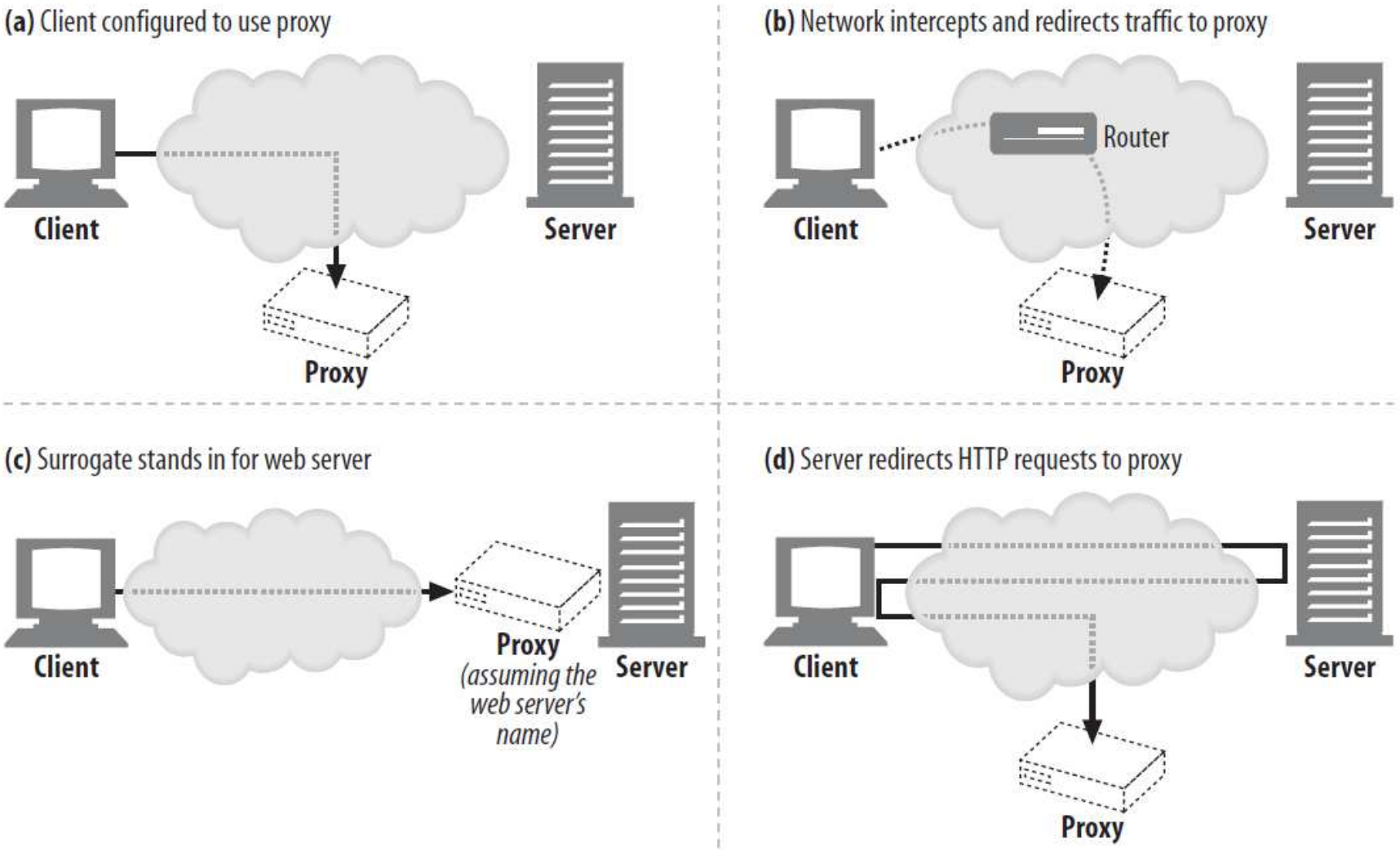
- Proxy usage
  - Load balancing
    - Proxy deployed close to server
    - Balances clients' requests among multiple available servers
    - Improves server performance during heavy load
    - Alternative names
      - Reverse proxy
      - Server accelerator
      - Surrogate



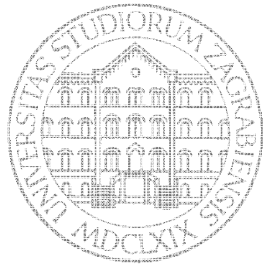
Layered System



- How proxy gets traffic?

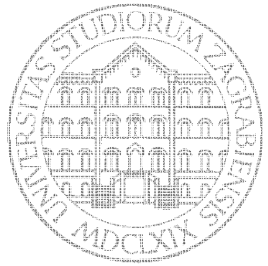


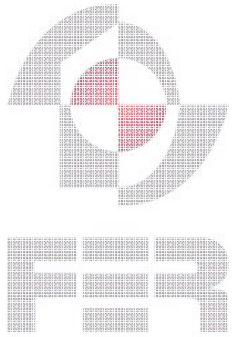
REST Constraints



- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

Resource-Oriented Architecture



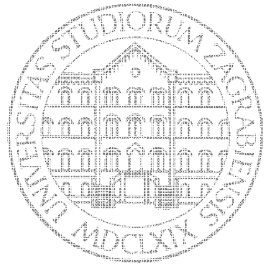


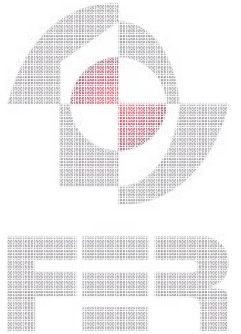
- Resource-oriented architecture (*ROA*)
  - Application exposed to clients as a set of resources
    - Finite set of resources

      or
    - Infinite set of resources

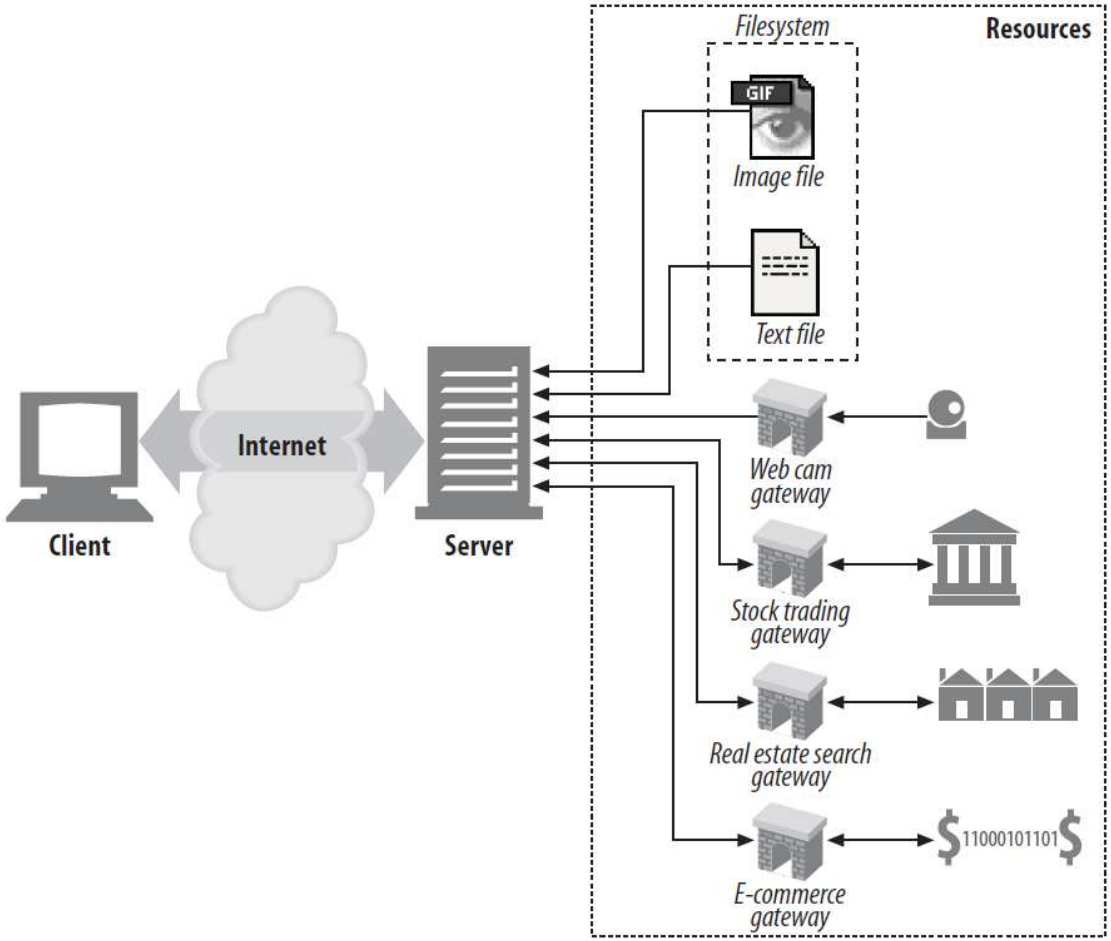
- Resource
  - Basic building block of resource-oriented architecture
  - Anything that is important enough to be referenced by clients
  - Something that can be stored on a computer and represented as a stream of bits
    - Document
    - Image
    - List of documents
    - Collection of images
    - Row in a database
    - Result of running an algorithm
    - . . . .
  - **Well-defined semantics that distinguishes that resource from other resources**

Resource-Oriented Architecture

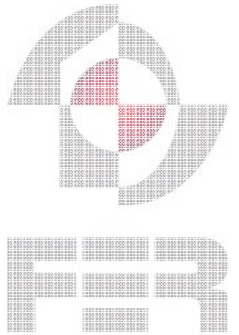




- Resource-oriented architecture (*ROA*)
  - Resources reside on server
  - Clients access resources through HTTP protocol

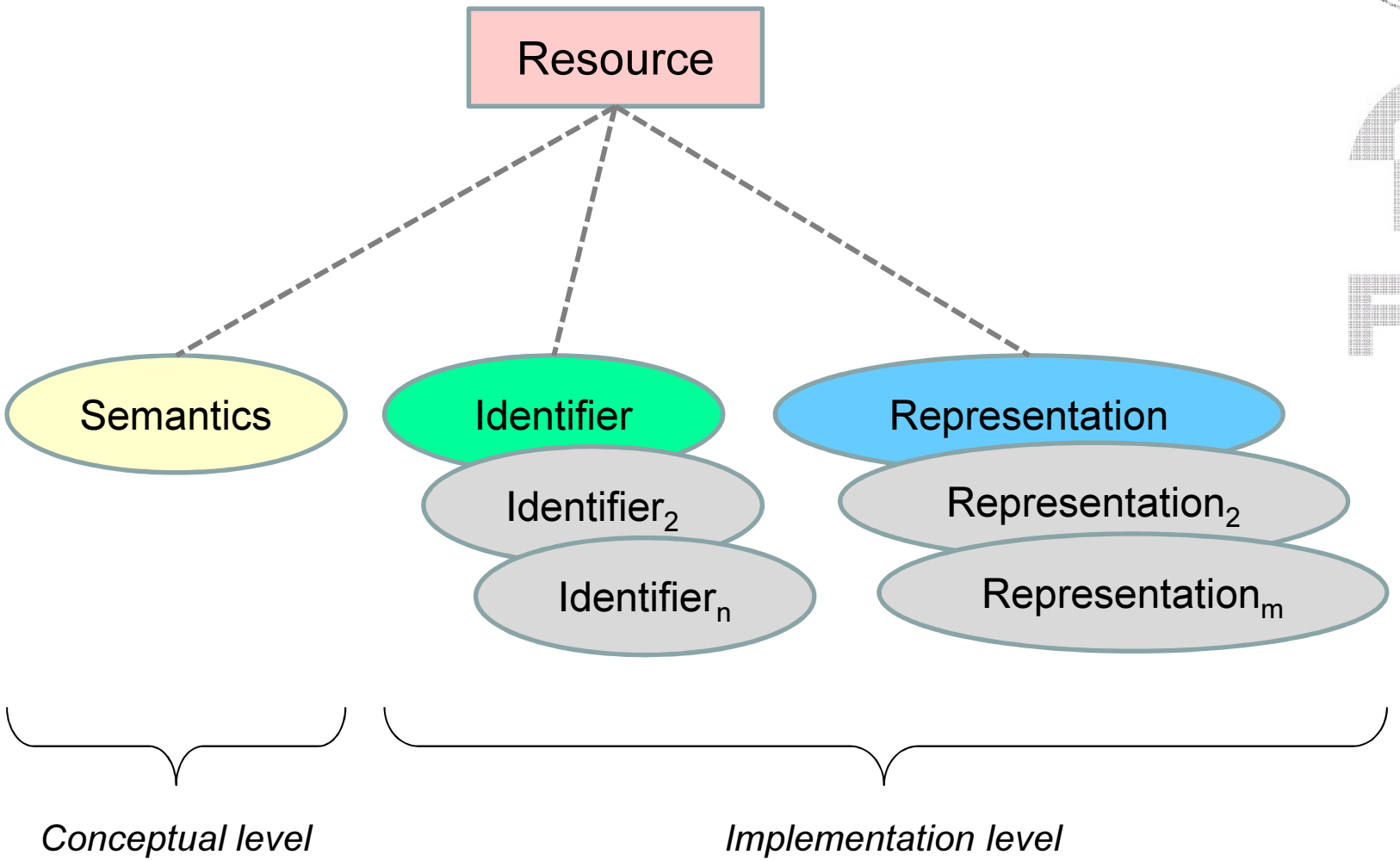


Resource-Oriented Architecture

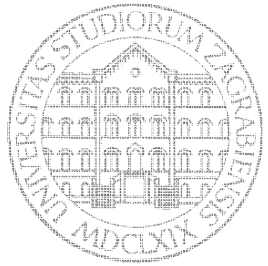


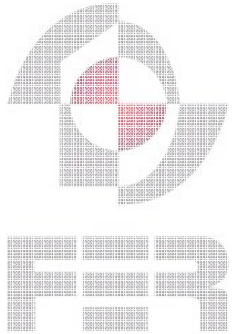
- Example od resources
  - UNIZG-FER-ZEMRIS homepage
  - UNIZG-FER *Service-Oriented Computing* course homepage
  - UNIZG-FER *Service-Oriented Computing* lecturer's homepage
  - List of UNIZG-FER computer science courses
  - List of UNIZG-FER computer science elective courses
  - UNIZG-FER *Service-Oriented Computing* students
  - UNIZG-FER *Service-Oriented Computing* student
  - Version 1.0.3 of the software release
  - The latest version of the software release
  - A list of the open bugs in latest version of the software release
  - The first weblog entry for September 30, 2013
  - A road map of Zagreb, Croatia
  - A satellite map of Zagreb, Croatia
  - The next prime number after 1024
  - The next five prime numbers after 1024
  - The sales numbers for Q4 2013
  - The relationship between two acquaintances, Alice and Bob

Resource-Oriented Architecture



Resource-Oriented Architecture

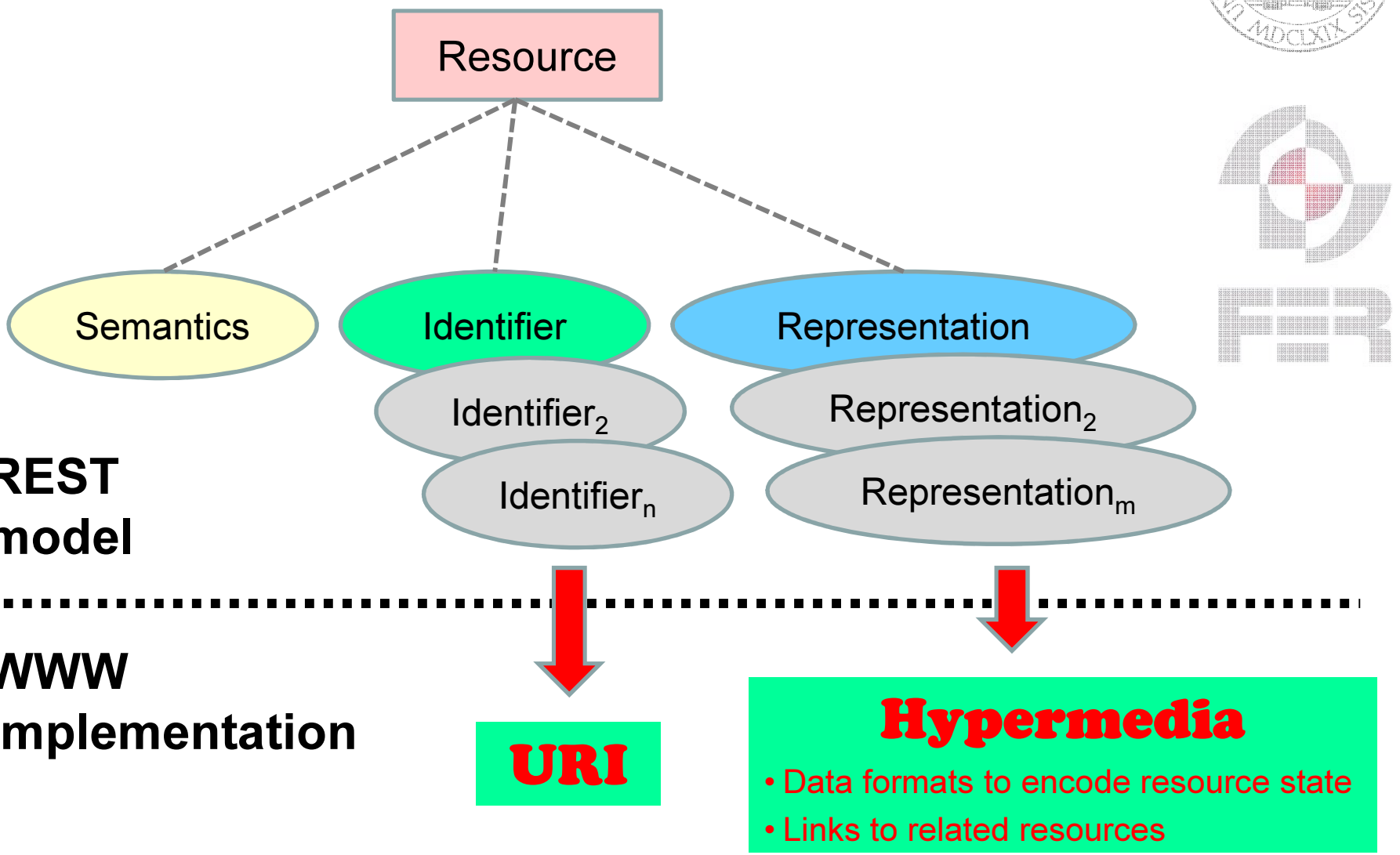




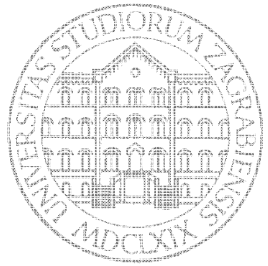
- Resource identifiers
  - Used to reference resources from remote clients
  - Each resource has to have **at least one identifier**
  - Resource may have more than one identifier
    - Permanent identifiers (e.g. *version 1.0.3 of the software release*)
    - Temporary identifiers (e.g. *latest version of the software release*)
  - Each resource identifier is globally unique

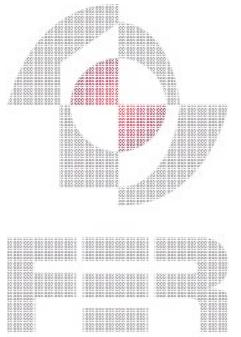
- Resource representation
  - Data format used to represent the resource state and exchange state between client and server
    - Documents
      - HTML
      - XML
      - JSON
      - Plain text
      - PDF
    - Graphics
      - BMP
      - GIF
      - JPEG
      - PNG
      - SVG
    - Sound
      - MP3
      - OGG
      - WAV
    - General-purpose
      - Binary
      - Base-64
      - ZIP
  - Resource may have more than one representation
  - **All resource representations represent the same state**

Resource-Oriented Architecture



URIs



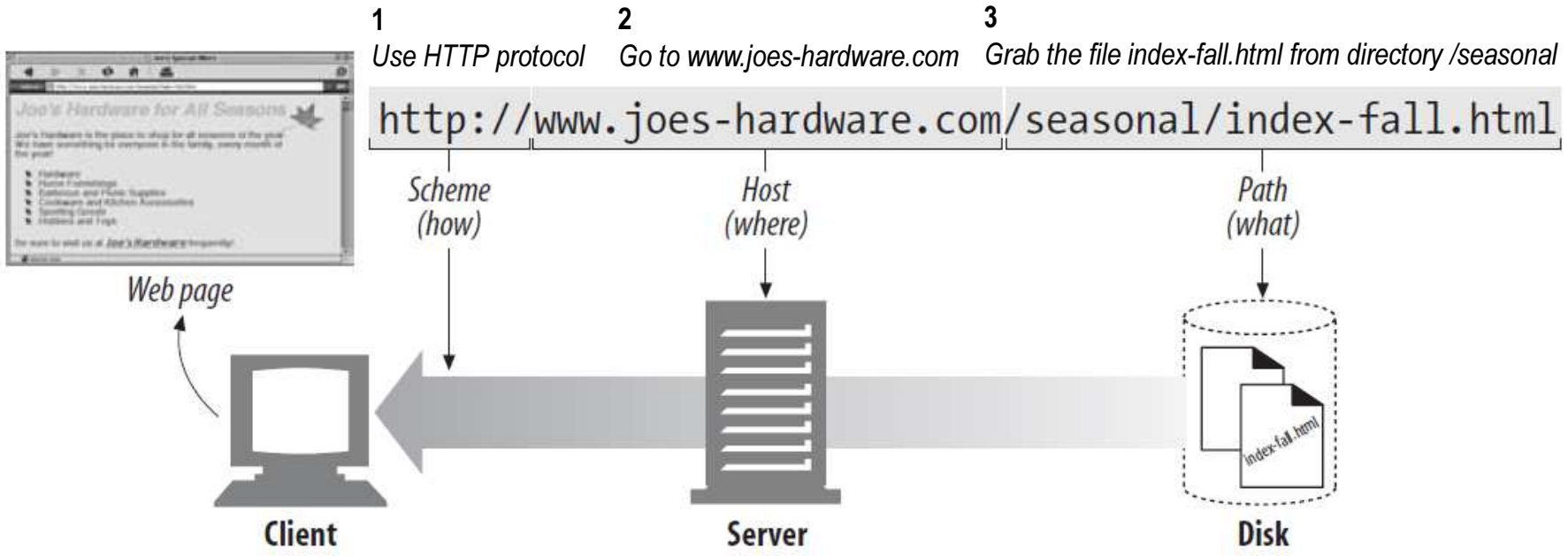


- Uniform Resource Identifier (URI)
  - Globally unique identifiers of the World Wide Web resources

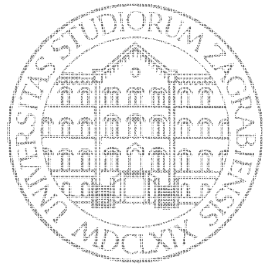
URIs

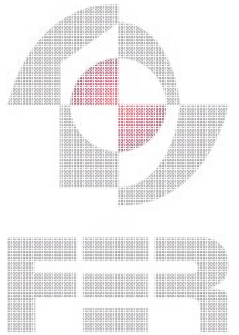
- Uniform Resource Locator (URL)
  - The most common form of URI
  - Describes the specific location of a resource on a particular server
  - Tells exactly how to fetch a resource from a precise, fixed location

Today, almost every URI is a URL



URIs



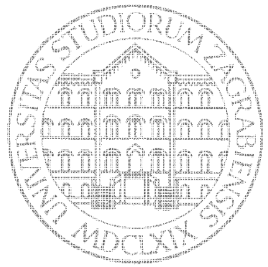


- Uniform Resource Name (URN)
  - A unique name for a particular resource, independent of where the resource currently resides
  - URN is location-independent
  - Allows resources to move from place to place
  - Allows resources to be accessed by multiple protocols
  - The following URN might be used to name the Internet standards document “RFC 2141” regardless of where it resides (it may even be copied in several places)

```
urn:ietf:rfc:2141
```

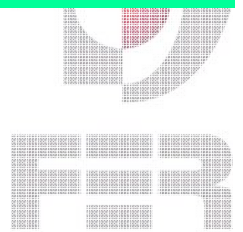
  - URNs are still experimental and not yet widely adopted
  - To work effectively, URNs need a supporting infrastructure to resolve resource locations
  - The lack of such an infrastructure has slowed their adoption

URL Syntax



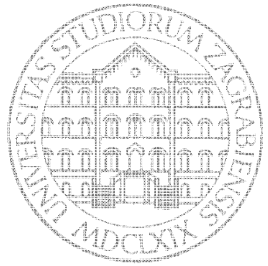
- General URL syntax
  - Almost no URL contains all these components

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```



- URLs provide a means of locating any resource on the Internet
  - **URLs are not tied to World Wide Web only**
- URL for a particular protocol is a subset of a general URL form
- Three most important parts used by World Wide Web
  - Scheme
  - Host
  - Path

URL Syntax



- General URL syntax
  - Almost no URL contains all these components

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```

- World Wide Web

  http://<*host*>:<*port*>/<*path*>?<*query*>#<*fragment*>

  http://www.joes-hardware.com/index.html (default port 80)

  http://www.joes-hardware.com:80/index.html

- Mail

  mailto:<*user*>@<*host*>

  mailto:dejan.skvorc@fer.hr

- FTP

  ftp://<*user*>:<*password*>@<*host*>:<*port*>/<*path*>

  ftp://dejan:mypwd@ftp.fer.hr:21/lectures/slides05.ppt

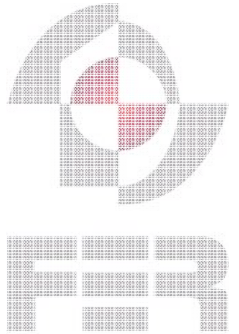
- Local files

  file://<*host*>/<*path*>

  file://localhost/C:/Lectures/Slides05.ppt

URL Syntax

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```

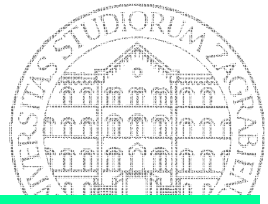


- Scheme
  - Which protocol to use when accessing a server to get a resource
  - Scheme names are case insensitive

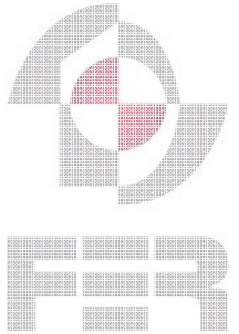
```
http://www.joes-hardware.com
HTTP://www.joes-hardware.com
```

} =

URL Syntax



```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```

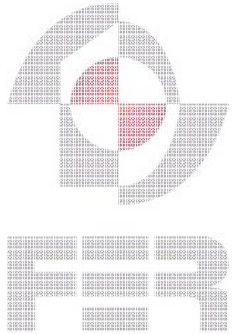


- Many non-public servers require a username and password before you can access data through them
- User
  - The username some schemes require to access a resource
- Password
  - The password that may be included after the username, separated by a colon (:)

```
ftp://ftp.prep.ai.mit.edu/pub/gnu
ftp://anonymous@ftp.prep.ai.mit.edu/pub/gnu          (default username)
ftp://joe:joespwd@ftp.prep.ai.mit.edu/pub/gnu

http://joe:joespwd@www.joes-hardware.com/sales.txt
```

URL Syntax

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```



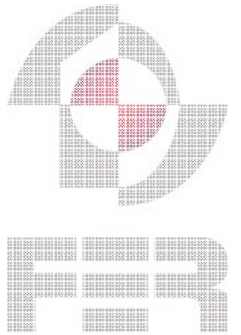
- To locate a resource
- Host
  - The host machine on the Internet that has access to the resource
    - Hostname
    - IP address
- Port
  - The network port on which the server is listening
  - **For HTTP, the default port is 80**

```
http://www.joes-hardware.com:80/index.html
http://161.58.228.45:80/index.html
http://www.joes-hardware.com/index.html    (default port 80)
http://161.58.228.45/index.html            (default port 80)
```

=

URL Syntax

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```

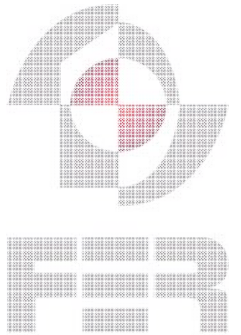


- Path
  - Specifies where on the server machine the resource lives
  - Often resembles a hierarchical file system path
    - Resources on the server are not necessarily organized in file system, but may be generated dynamically
  - Divided into path segments separated by forward slash (/)

```
http://api.soccer.com/                                  root resource (docroot)
-------------------------------------------------------------------------------
http://api.soccer.com/leagues                           subordinate resources
http://api.soccer.com/leagues/seattle
http://api.soccer.com/leagues/seattle/teams
http://api.soccer.com/leagues/seattle/teams/sonic
http://api.soccer.com/leagues/seattle/teams/sonic/players
http://api.soccer.com/leagues/seattle/teams/sonic/players/mike
```

URL Syntax

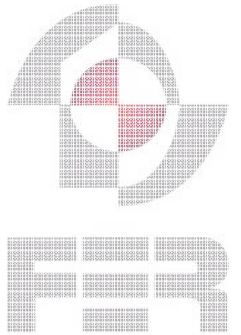
```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```



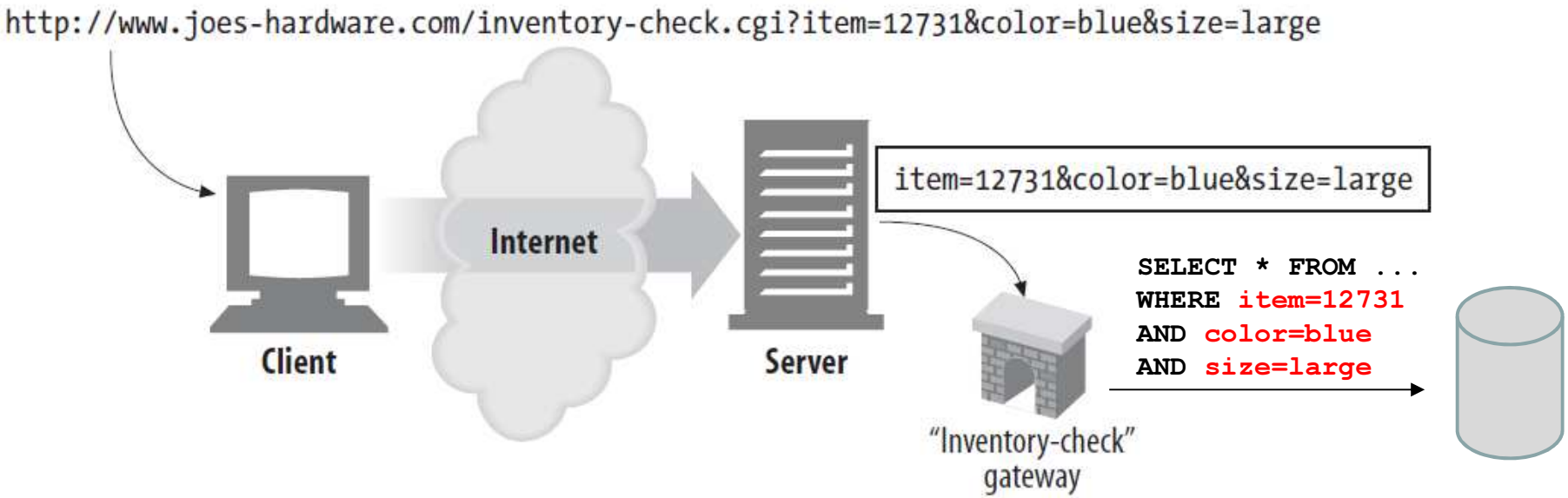
- Parameters
  - Used by some schemes to specify input parameters
  - Parameters are name/value pairs
  - Each path segment can have its own `params` component
  - Rarely used by HTTP

URL Syntax

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```

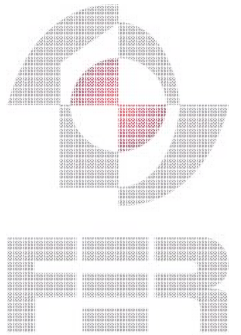


- Query strings
  - Used to pass parameters to active applications (instead of static resources), such as database search, search engines, etc.
  - `name=value` pairs separated by `&` character



URL Syntax

```
<scheme>://<user>:<pass>@<host>:<port>/<path>;<params>?<query>#<fragment>
```



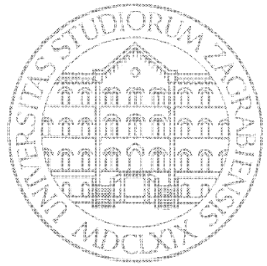
- Some resource types, such as HTML document, can be divided further than just the resource level
  - For example, large text document with sections in it
- Fragment
  - A name for a piece or part of the resource
  - HTTP servers deal only with entire resources (not with fragments of resources)
    - Clients don't pass fragments to servers, but use them only internally
    - Browser gets the entire resource, but displays the `fragment`-marked portion

`http://www.joes-hardware.com/tools.html`
(browser positions the view window at the top of the document)

`http://www.joes-hardware.com/tools.html#drills`
(browser positions the view window where the `drills` section begins)

`http://www.joes-hardware.com/tools.html#chain-saws`
(browser positions the view window where the `chain-saws` section begins)

Addressability

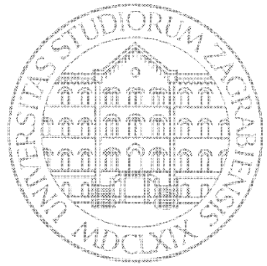


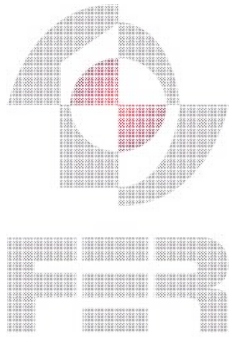
- Resources + URIs ⇒ Addressability

- Addressability
  - Fine granulation of a web service
  - Gives a structure to a web service
  - Every interesting aspect of a service is accessible from outside
    - Has a well-defined semantics
    - Has a URI

- Mashups
  - Clients don't need to manipulate with a service as a whole, but with particular resources
  - Clients can combine resources from many services into new services in ways their creators never imagined

Addressability





Go to **www.business.com**, click **Legal Agreements** at the bottom of the page, then select **Policies** from the menu on the left side of the page, and finally click the **Policy Update**

Not an addressable service

- Hard to automate access to content
- Needs human interpretation of access rules

```
http://www.business.com/legal-docs/agreements/policies/latest
```

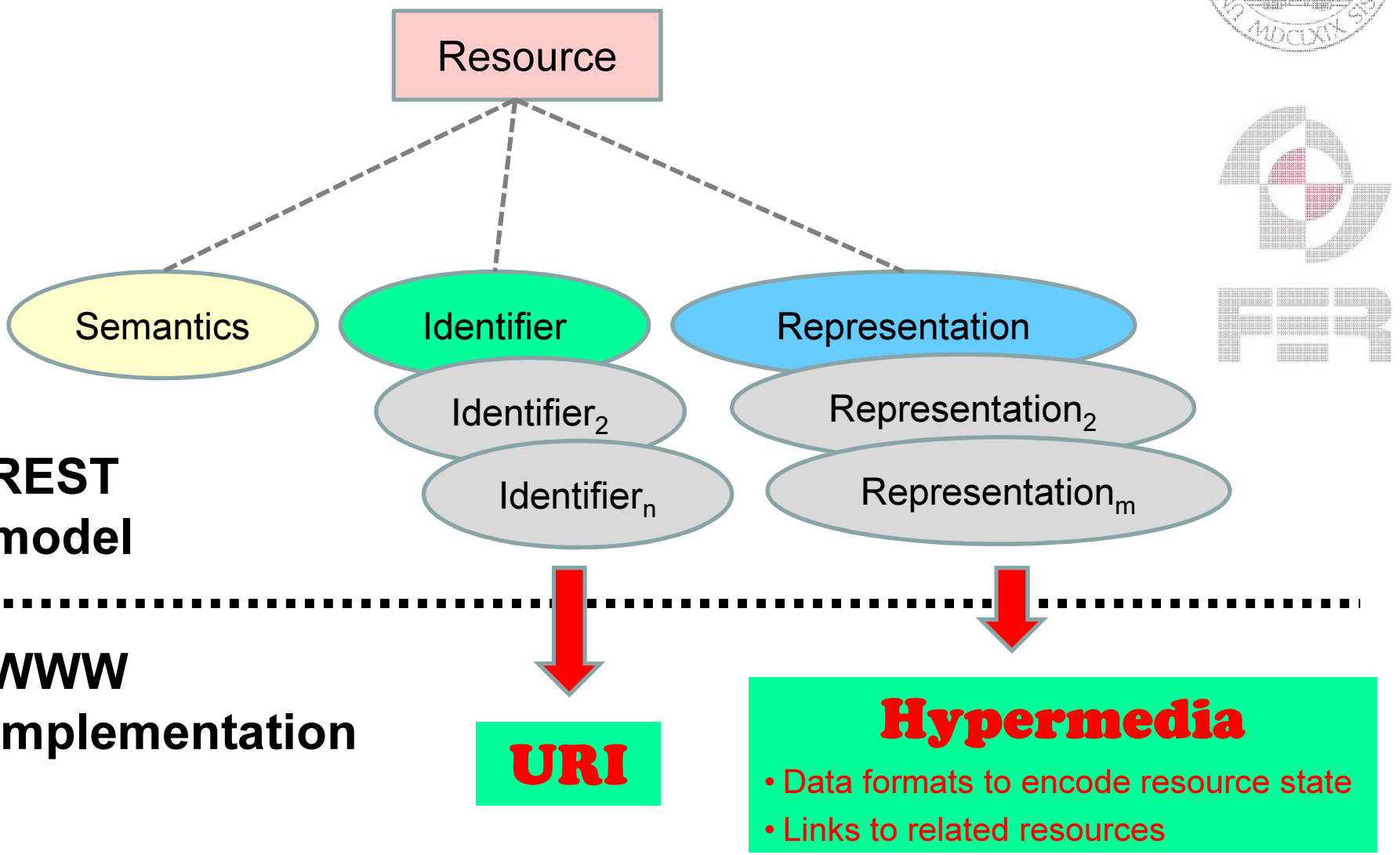
Addressable service

- Easy to build software to access the content

REST Constraints

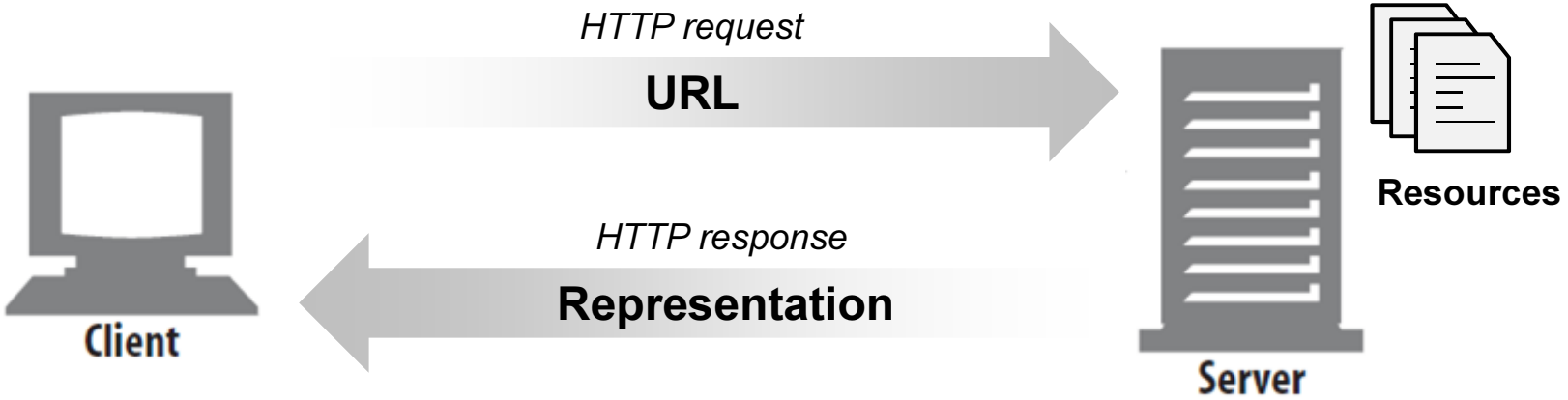
- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

Resource-Oriented Architecture

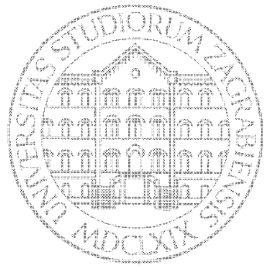


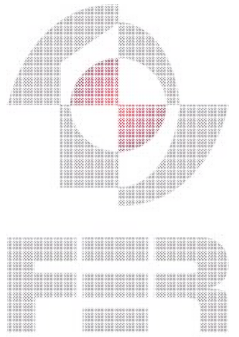
Representations

- Resource representation
  - Data used to represent the resource state and exchange state between client and server
    - Documents
      - HTML
      - XML
      - JSON
      - Plain text
      - PDF
    - Graphics
      - BMP
      - GIF
      - JPEG
      - PNG
      - SVG
    - Sound
      - MP3
      - OGG
      - WAV
    - General-purpose
      - Binary
      - Base-64
      - ZIP



Representations





- Resource may have more than one representation

- **All resource representations represent the same state**

Representations

- Different representations of the same resource state
  - Data formats

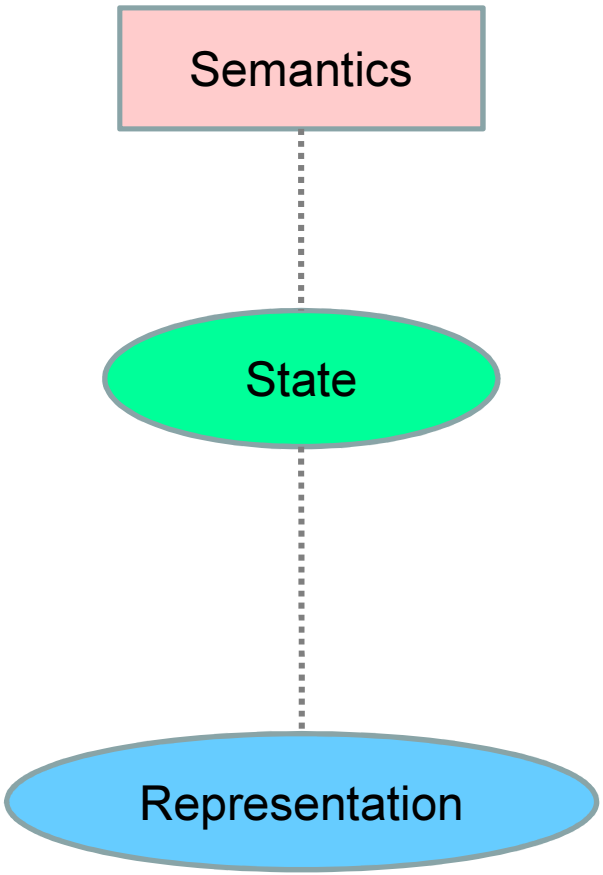
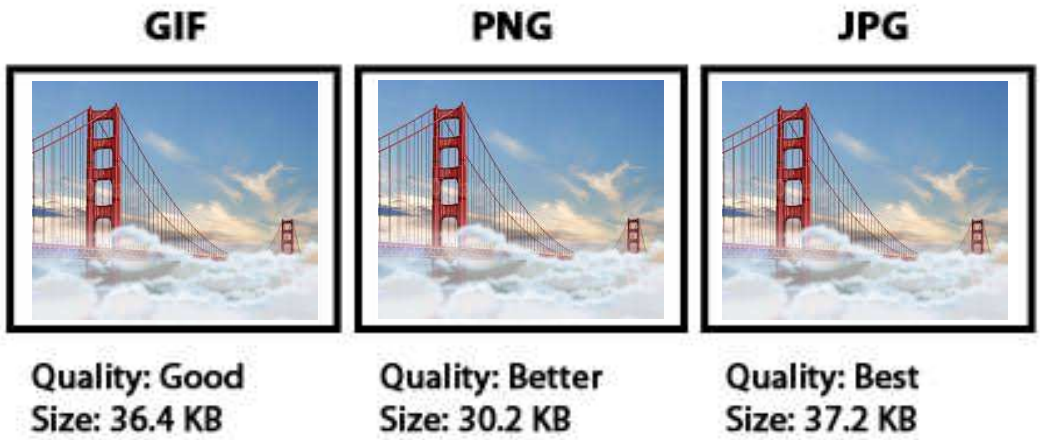


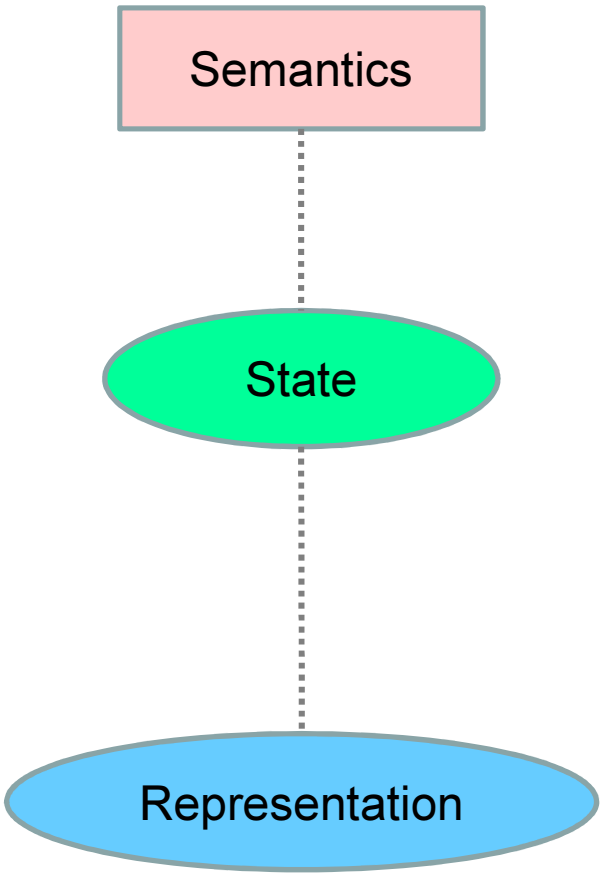
Photo of the Golden Gate Bridge in Fog

Graphics to be displayed on the screen



Representations

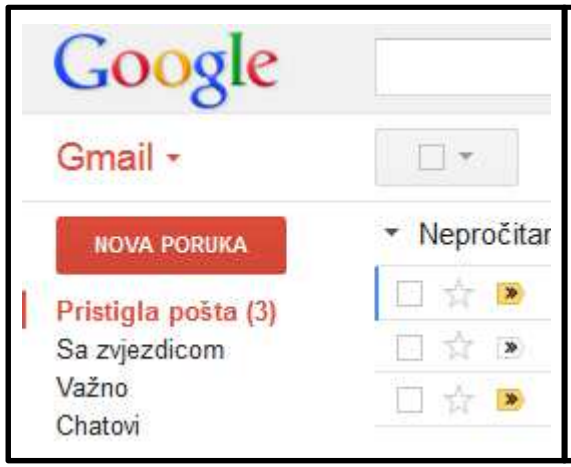
- Different representations of the same resource state
  - Language (localization of resources)



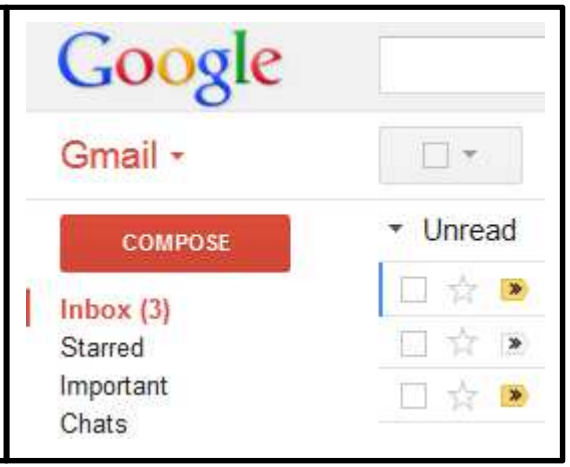
Gmail Inbox

List of received email messages

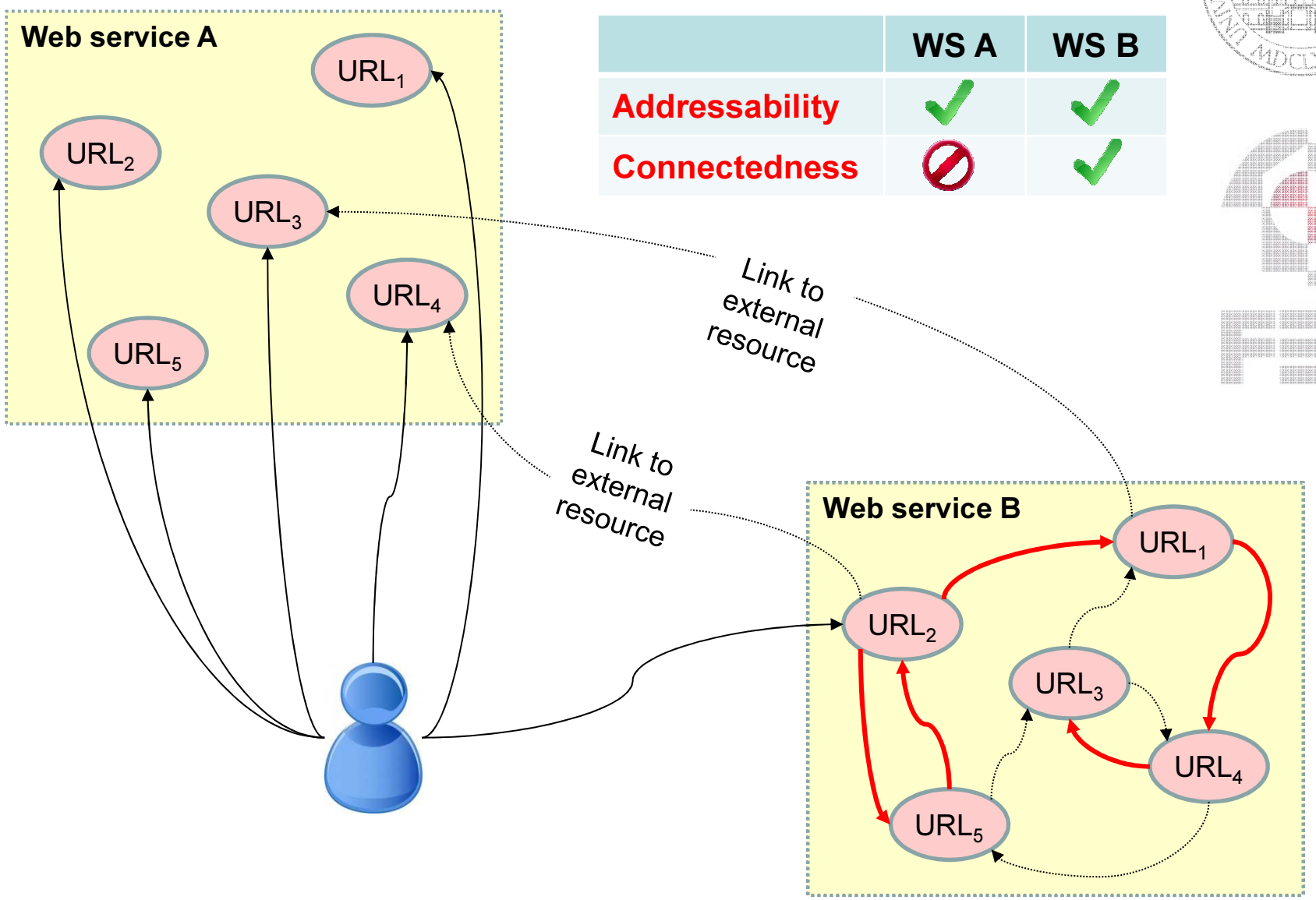
HR



EN



Connectedness



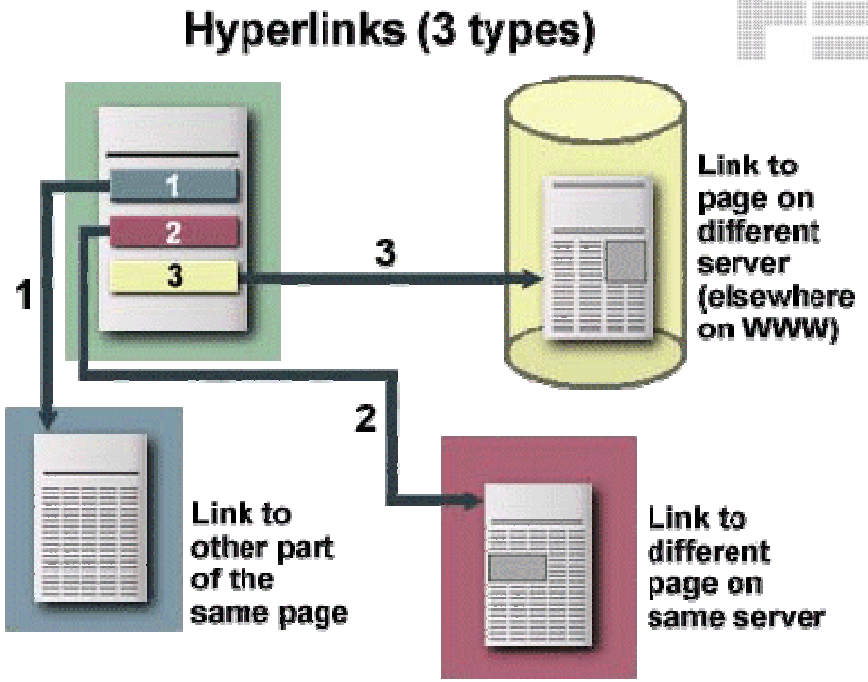
	WS A	WS B
Addressability	✓	✓
Connectedness	⊘	✓

Hypermedia

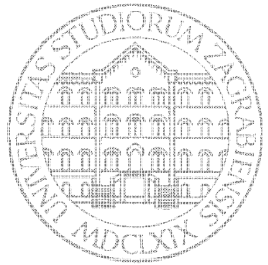
- How to achieve connectedness?

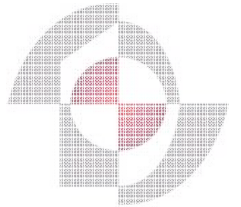
Hypermedia-based resource representations

- Hypermedia
  - Non-linear data representation system that enables compilation of the multimedia content in an interconnected manner
  - Augments the essential content with links to related content
    - Resource state
    - Links to related resources (hyperlinks)



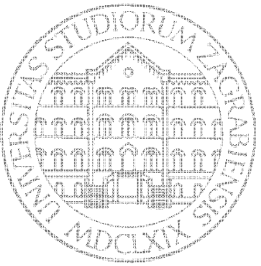
Hypermedia

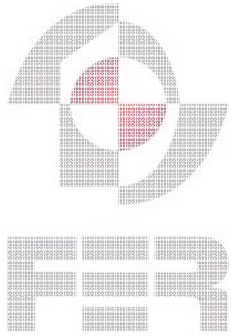




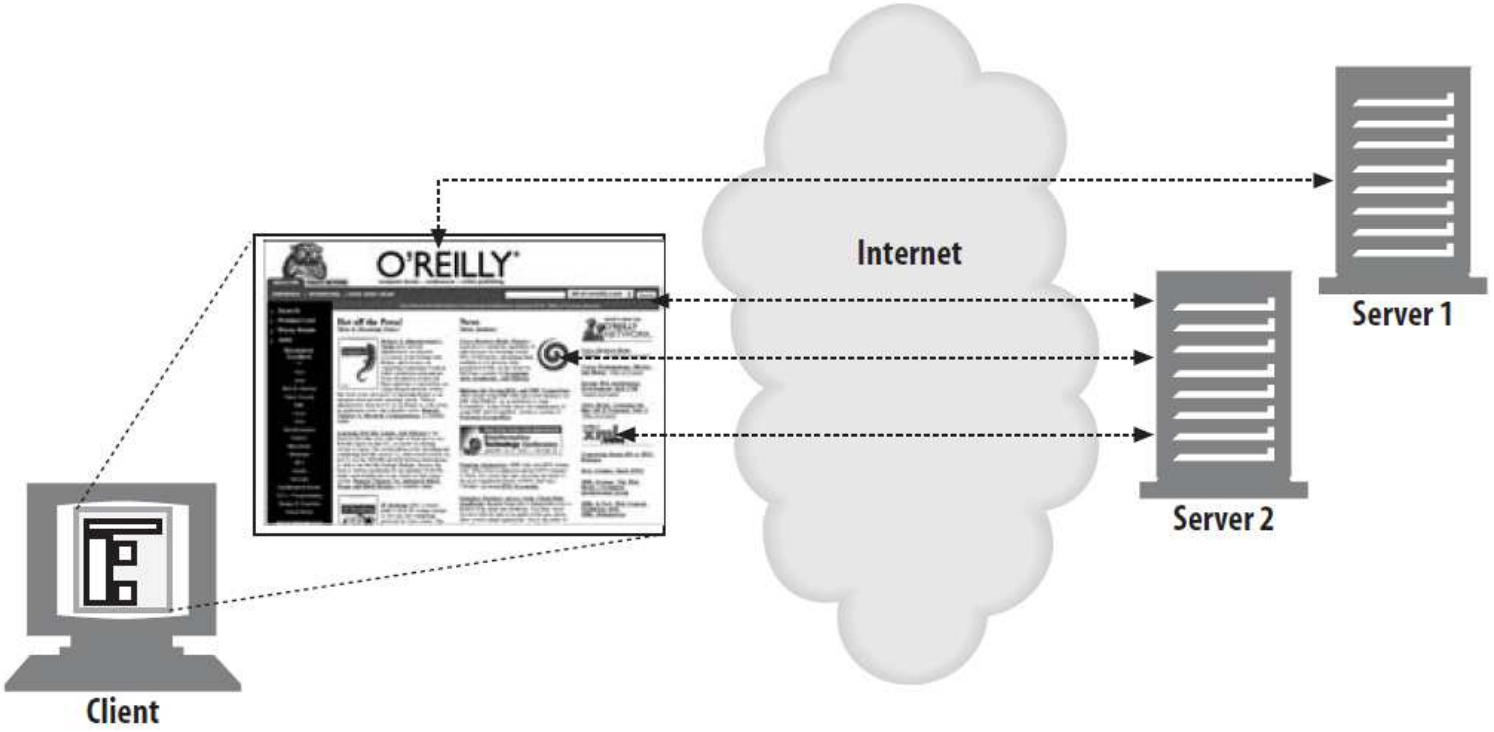
- Hypermedia
  - Enables to combine various media types into single document
    - Text
    - Graphics
    - Sound
    - Video

Hypermedia

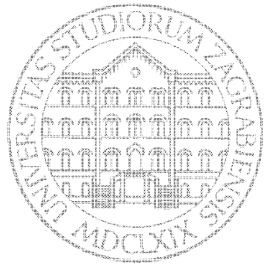


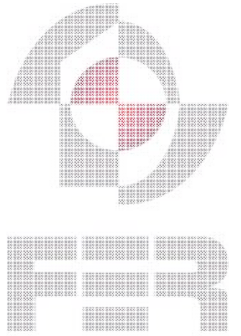


- Hypermedia
  - Enables to combine resources from various sources



Hypermedia





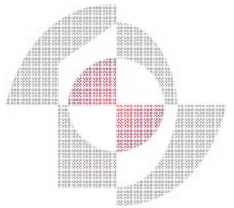
- Hypermedia
  - Enables to traverse the resources in a non-linear manner
    - Jump from one resource to another and vice versa
    - No predefined order/path when accessing resources
    - No need to access all the referenced resources

- Hypermedia technologies

Human Web

Programmable Web

HTML



- **H**yper**T**ext **M**arkup **L**anguage
  - Main markup language for creating web pages
  - Designed to carry information to be displayed in a web browser
    - GUI-based web applications

- HTML documents
  - Content (pure content + hyperlinks)
  - Display and formating instructions

```
<html>
 <head></head>
 <body>
  <h1>Title</h1>
  <h2>Subtitle</h2>
  <p>Text paragraph</p>
  <img src="http://www.c.com/image.gif" />
  <a href="http://www.b.com/page2.html">Another page</a>
  <input type="text" />
  <input type="submit" />
 </body>
</html>
```

Title

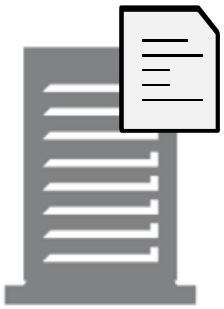
Subtitle

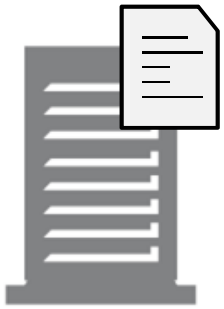
Text paragraph

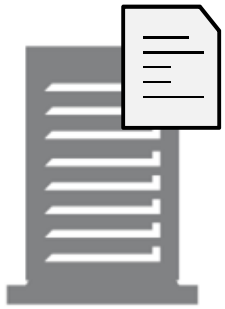
Another page

Submit Query

HTML







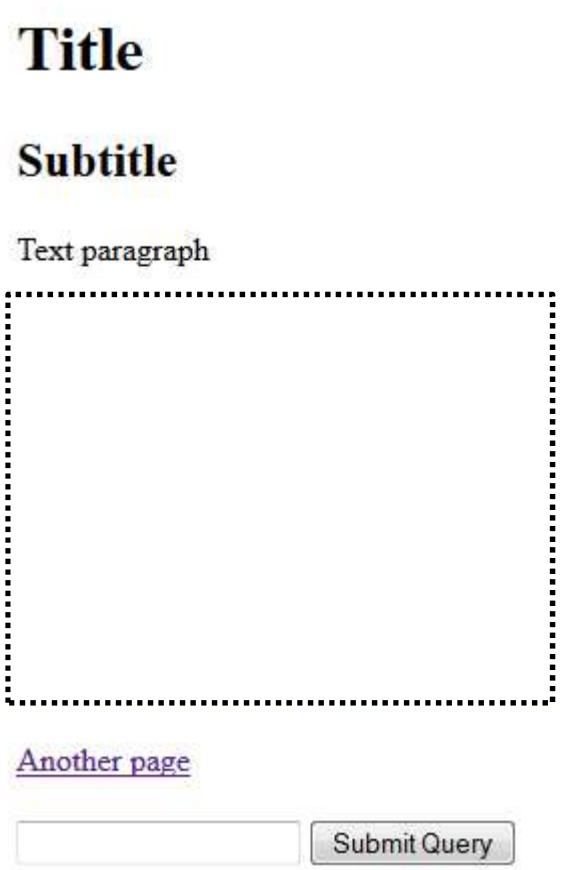
HTTP request

```
GET /page1.html HTTP/1.1
Host: www.a.com
```

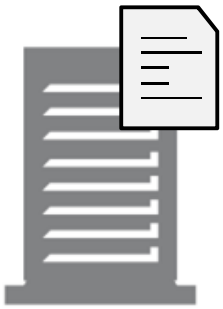
HTML

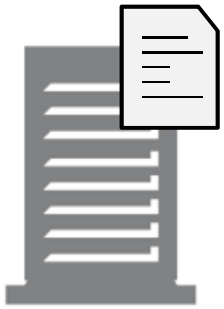
HTTP response

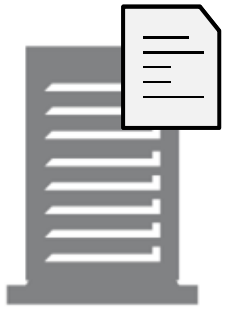
```
HTTP/1.1 200 OK
Content-Length: 123
Content-Type: text/html

<html>

 <head></head>

 <body>
  <h1>Title</h1>
  <h2>Subtitle</h2>
  <p>Text paragraph</p>
  <img src="http://www.c.com/image.gif" />
  <a href="http://www.b.com/page2.html">Another page</a>
  <form>
   <input type="text" />
   <input type="submit" />
  </form>
 </body>

</html>
```



HTML







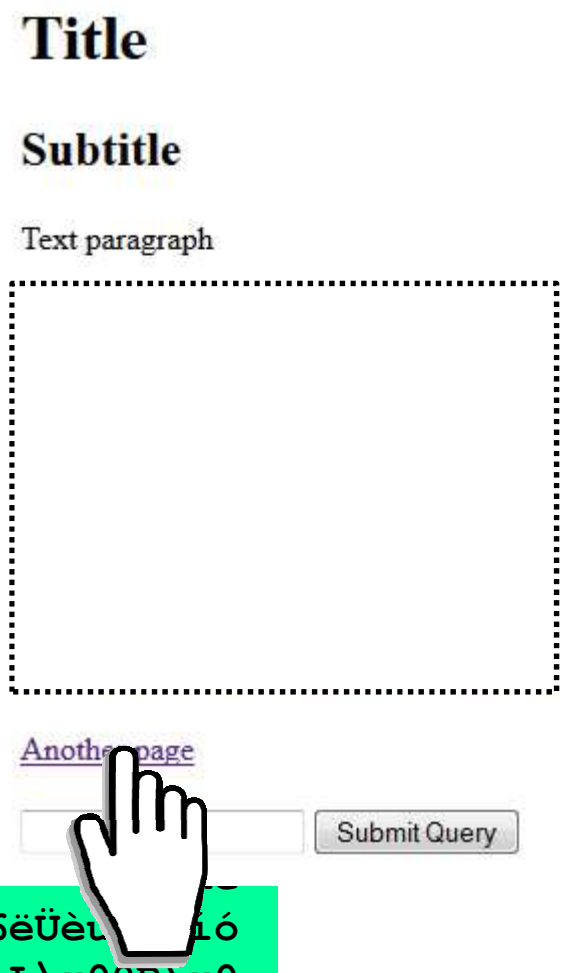
HTTP request

```
GET /image.gif HTTP/1.1
Host: www.c.com
```

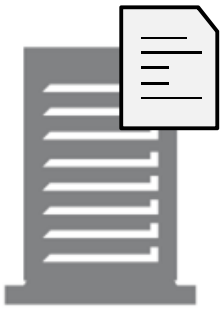
HTML

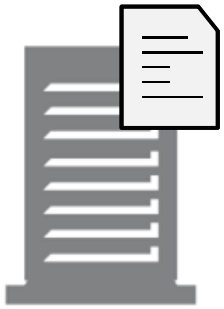
HTTP response

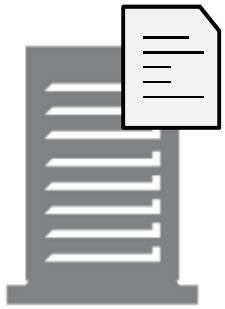
```
HTTP/1.1 200 OK
Content-Length: 4123
Content-Type: image/gif

GIF89aX\x002X\x002⌡,\x000ΣΣ\x000∞∞\x000⌠⌠\x000
\x000\x000\x000\x000\x000\x000\x000\x000\x000\x000\
x000\x000\x000\x000\x000\x000\x000\x000\x000\x000\x
000\x000\x000\x000\x000\x000\x000\x000\x000\x000\x0
00\x000\x000\x000\x000\x000\x000\x000\x000\x000\x00
0\x000\x000\x000\x000\x000\x000\x000\x000\x000\x000
!·\x004\t\x002\t@File source: http://www.c.com
\x000\x000\x000X\x002X\x002\x000\x006@ûpH,\x01
¿tJ¡Z»F╟&si0\x00Cïà┴P(\x00B╬Ç≤╢\\\x018ï┴ìKfπ└┌∩
âäàG\x00E\x00C\x00B_èdh\x000ÅÉæÆôöòÅj\x002\x006ëÜè
úñÑª°¿X\x019_\x005\x002¡û▒▓│ö««è\x019╒╟╗╝╜┐ ┴┴L\x00B\x0
0E\r¡ÄπëÑ═¢8°\x015+│°Oè╧jµ\x01CJ┤¿\x00C\x00B^²\╩tαìFúJ¥·
─╓╖ªX│å;├á\x001┌»`Ñ\x013╒┤¼Yx\x002\x016x\r╥╢mGcd┴╩¥δφ╠Z╖
x≤óC╢.┐ÇôÑ]\x000U»ß├╝\x01C┐ \x00B╟╤ ▓zê#K\x00E⌡╧▒╖\x000\x
001\x006\x00Cü\x002\x008\x011"LáPí┤à╨¿Qùª0!4ä\x007\t\x00
E\x010╩î┘≥▓Lw'δ}G1Oö6wV┴┬\x007
```



HTML







HTTP request

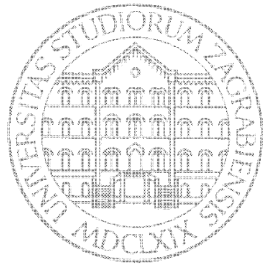
```
GET /page2.html HTTP/1.1
Host: www.b.com
```

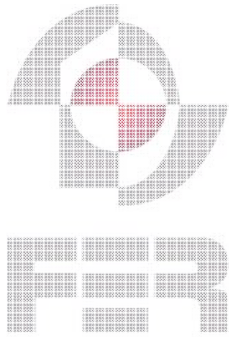
HTML

HTTP response

```
HTTP/1.1 200 OK
Content-Length: 85
Content-Type: text/html

<html>

 <head></head>

 <body>
  <h1>Hello again!</h1>
  <h3>This is another document</h3>
  <a href="http://www.a.com/page1.html">Back to first page</a>
  <br/>
  <a href="http://www.d.com/page3.html">Next page</a>
 </body>

</html>
```

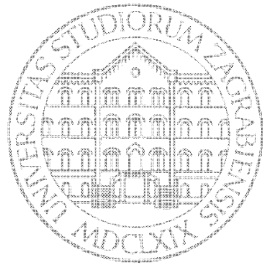
XML

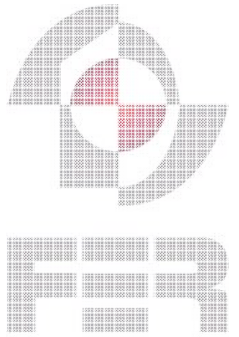




- E**X**tensible **M**arkup **L**anguage
  - Defines a set of rules for encoding documents in a format that is both human-readable and machine-readable
  - W3C open standard
  - Textual data format
  - General-purpose data format for representation of (semi)structured data
    - Markup + content
    - Unlike HTML, XML markup has no predefined semantics
  - Data formats derived from XML
    - XHTML
    - SOAP
    - RSS
    - Atom
    - Office Open XML (Microsoft Office)
    - OpenDocument (LibreOffice)

XML





- Tag
  - A markup construct that begins with < and ends with >
- Tags come in three flavors
  - Start tag

    ```
    <section>
    ```
  - End tag

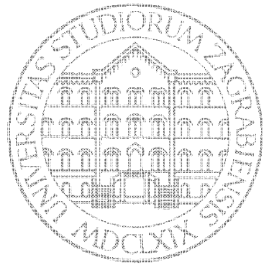
    ```
    </section>
    ```
  - Empty tag

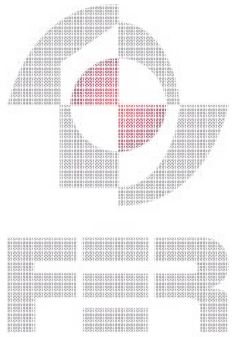
    ```
    <section />
    ```
    (same as `<section></section>`)
- Element
  - Start tag + end tag + content in between

    ```
    <section>This is a simple element</section>
    ```
  - Element can nest another elements

    ```
    <section>
      <subsection>This is a nested element</subsection>
    </section>
    ```

XML

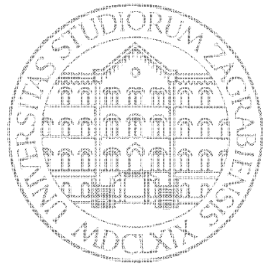


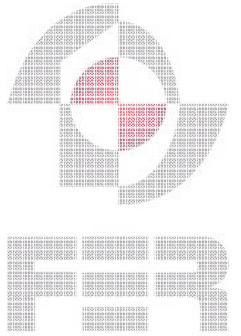


- Attribute
  - A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag

```
<step number="3">Connect A to B</step>

<step number="4" />
```

XML

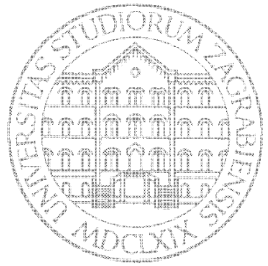


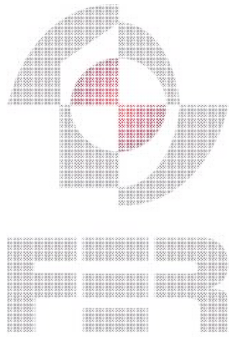


- Standard representation based on XML element content

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>
      <type>home</type>
      <number>212 555-1234</number>
    </phoneNumber>
    <phoneNumber>
      <type>fax</type>
      <number>646 555-4567</number>
    </phoneNumber>
  </phoneNumbers>
</person>
```

XML

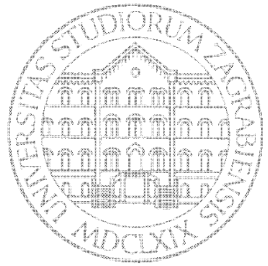


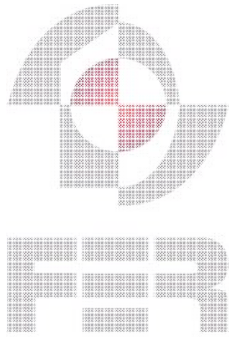


- Compact representation based on XML attributes

```
<person firstName="John"
        lastName="Smith"
        age="25">
  <address streetAddress="21 2nd Street"
           city="New York"
           state="NY"
           postalCode="10021" />
  <phoneNumbers>
     <phoneNumber type="home" number="212 555-1234"/>
     <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
</person>
```

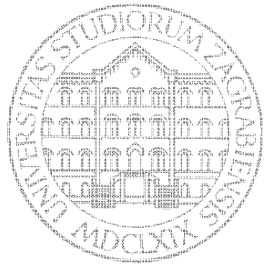
JSON

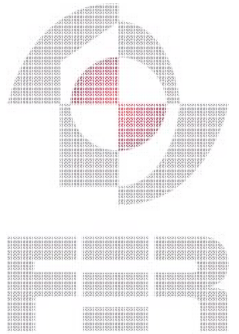




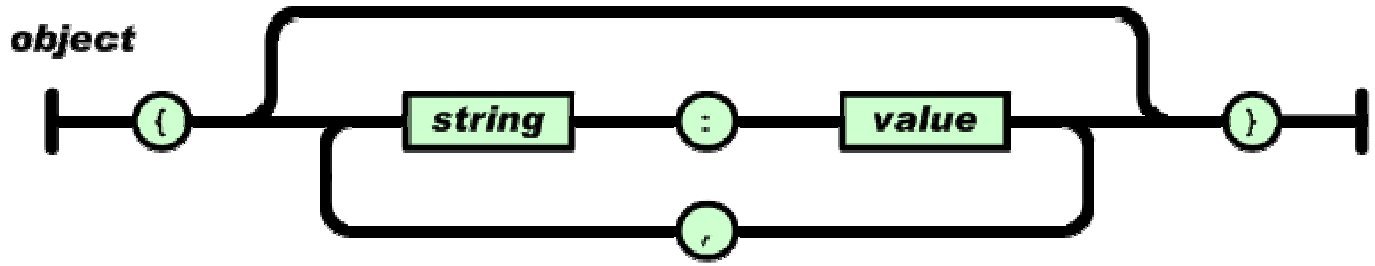
- **J**ava**S**cript **O**bject **N**otation
  - A language for representing simple data structures and associative arrays, called objects
  - Text-based open standard
  - The fat-free alternative to XML
  - Easy for humans to read and write
  - Easy for machines to parse and generate
  - Derived from the JavaScript scripting language
  - Despite its relationship to JavaScript, JSON is language-independent
  - Parsers available for many languages
  - Data types
    - Object
    - Array
    - Number, String, Boolean, null

JSON

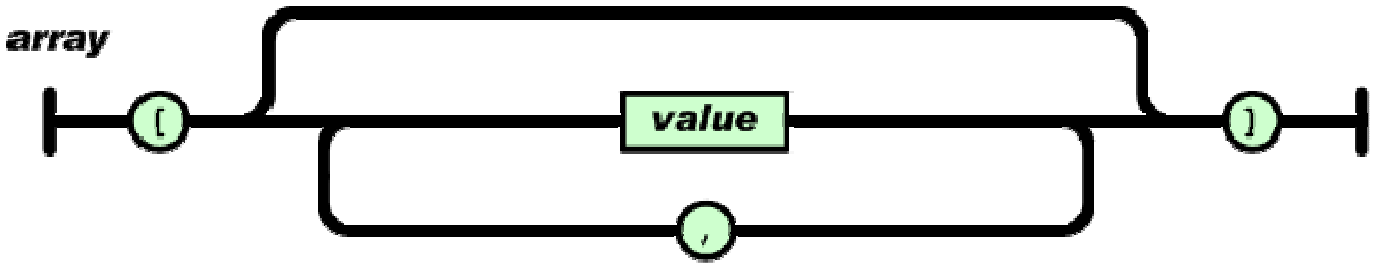




- Object
  - Unordered set of name/value pairs
  - An object begins with { and ends with }

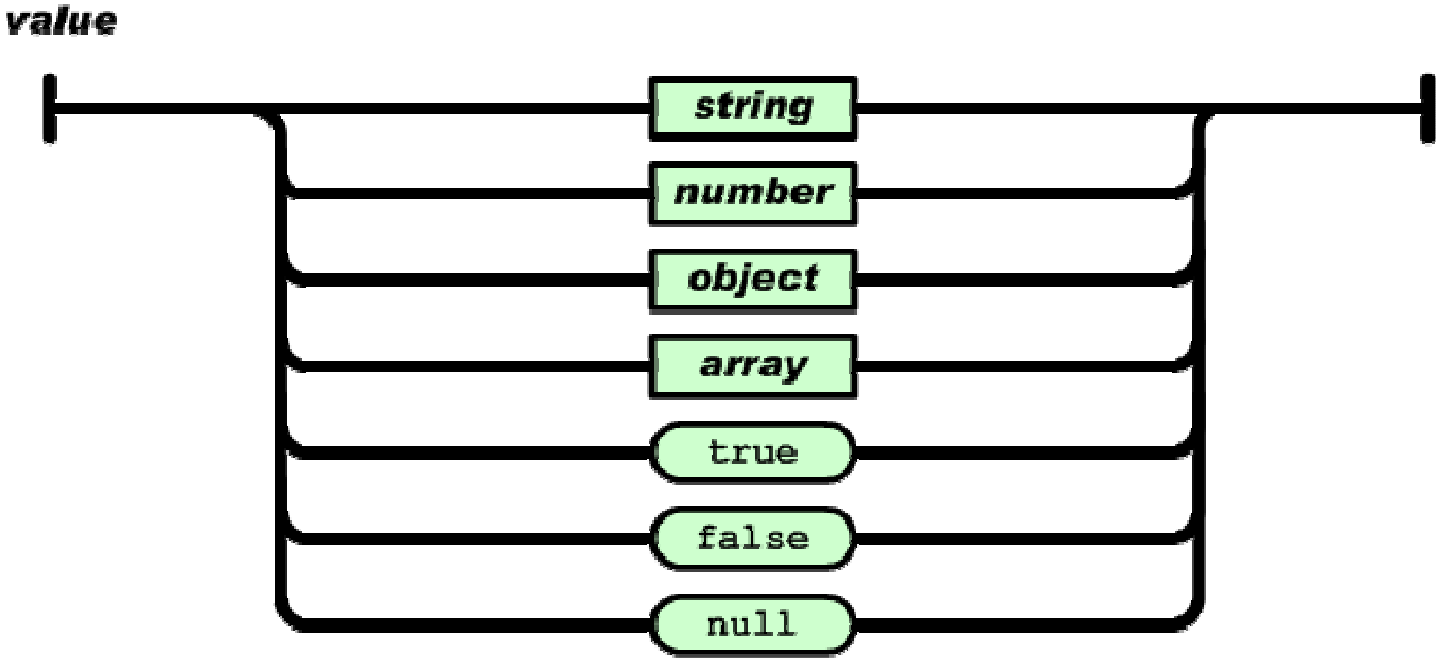


- Array
  - Ordered collection of values
  - An object begins with [and ends with]



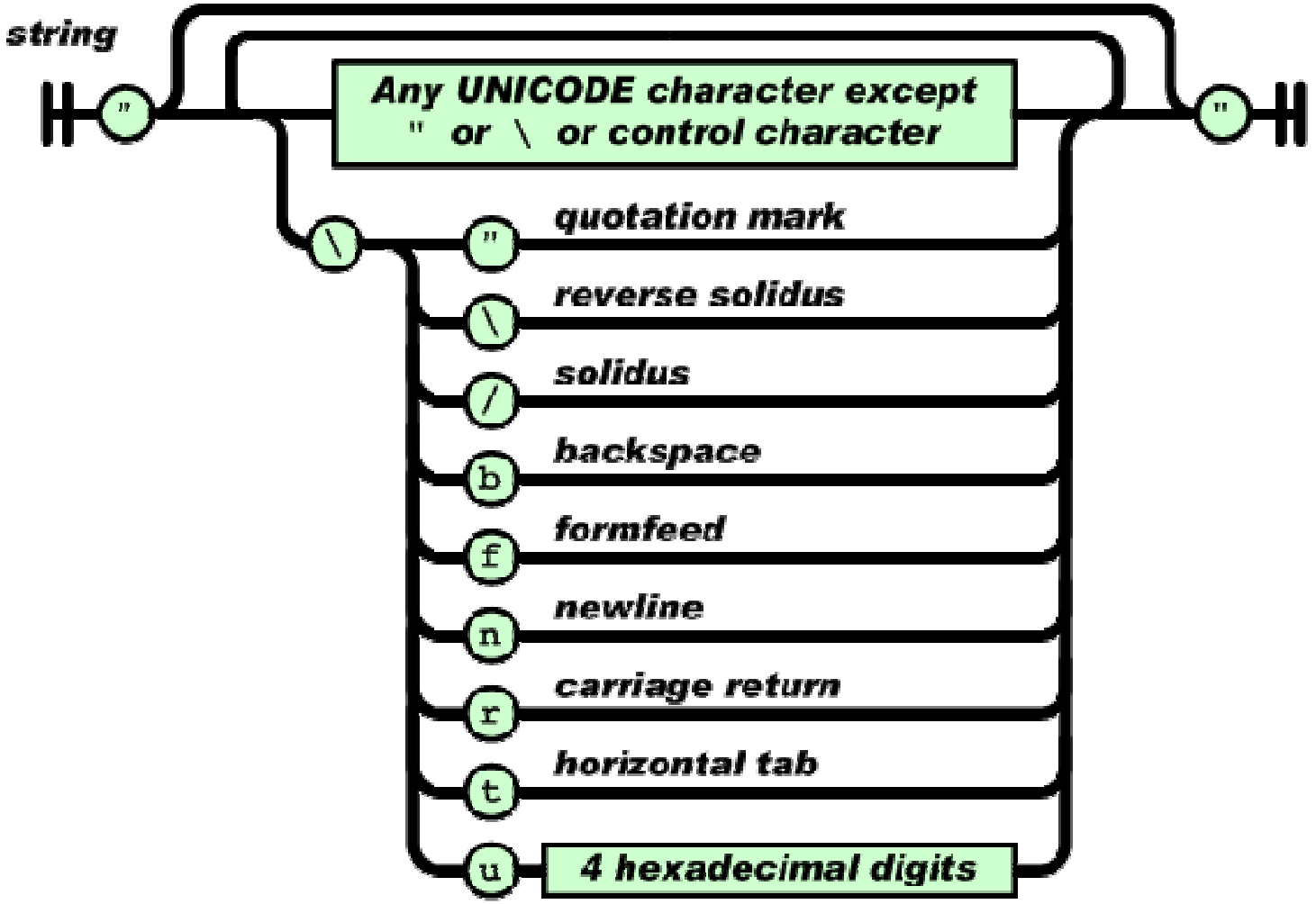
JSON

- Value



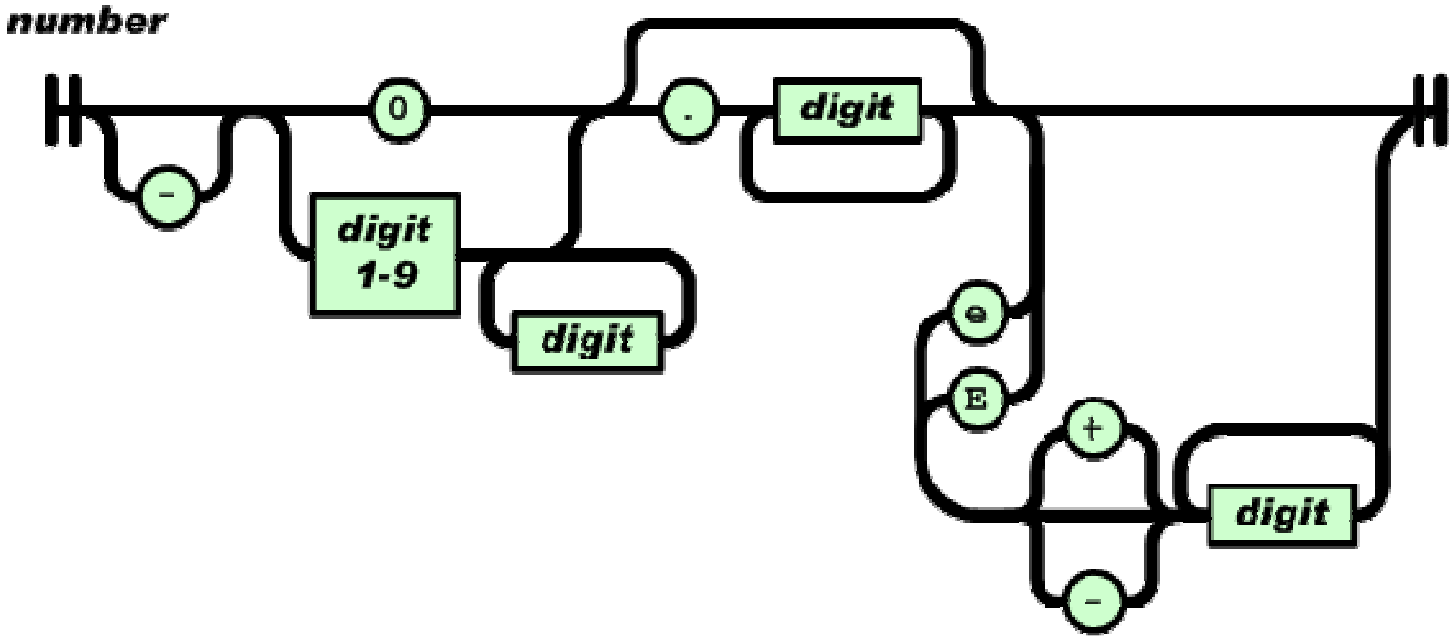
JSON

- String

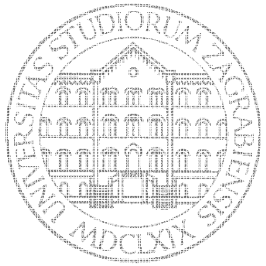


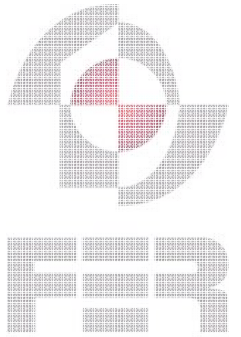
JSON

- Number



JSON



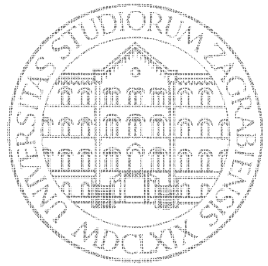


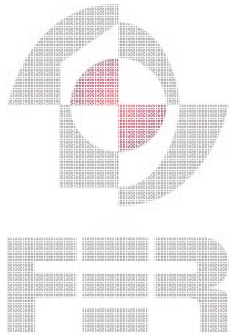
```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Native support in JavaScript

```
var p = JSON.parse(contact);

p.firstName
p.lastName
p.address.city
p.phoneNumbers[0].number
```

HTML, XML, JSON and Hypermedia



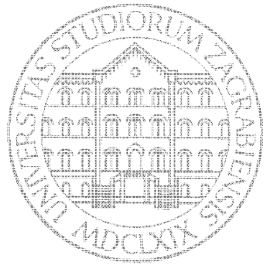


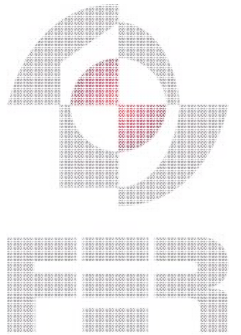
- Embedding links to resource representations
  - HTML: standardized markup semantics
  - XML, JSON: application-specific markup semantics

HTML

```
<html>
 <head>
  <script src="validate-form.js"/>
 </head>
 <body>
  <p>During the
   <a href="http://en.wikipedia.org
            /wiki/Renaissance">
            Renaissance
   </a> ...
  </p>
  <img src="/images/galileo.jpg"/>
 <body>
</html>
```

HTML, XML, JSON and Hypermedia





- Embedding links to resource representations
  - HTML: standardized markup semantics
  - XML, JSON: application-specific markup semantics

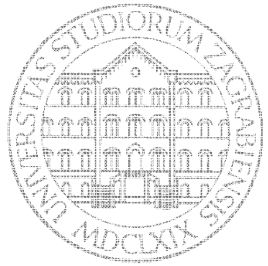
HTML

```
<html>
 <head>
  <script src="validate-form.js"/>
 </head>
 <body>
  <p>During the
   <a
    </a
  </p>
  <img
 <body
</html
```

XML

```
<person>
 <name>John Smith</name>
 <date-of-birth>1970-10-11</date-of-birth>
 <employer href="http://www.example.org/companies/
                 acme.xml">ACME Widgets, Inc.</employer>
 <country href="http://www.example.org/
                countries/ca.xml">Canada</country>
</person>
```

HTML, XML, JSON and Hypermedia



- Embedding links to resource representations
  - HTML: standardized markup semantics
  - XML, JSON: application-specific markup semantics

HTML

```
<html>
 <head>
  <script src="valida
 </head>
 <body>
  <p>During the
   <a
    </a
  </p>
  <img
 <body
</html
```

XML

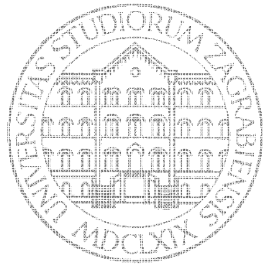
```
<person>
 <name>John Sm
 <date-of-birt
 <employer hre

 <country href

</person>
```

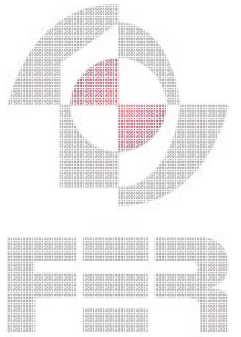
JSON

https://graph.facebook.com/625589/friends

```
{
  "data": [
    {
      "name": "John Smith",
      "id": "https://graph.facebook.com/670874"
    },
    {
      "name": "Tom Hanks",
      "id": "https://graph.facebook.com/281205"
    },
    {
      "name": "Morgan Freeman",
      "id": "https://graph.facebook.com/281288"
    },
  ]
}
```

HTML vs XML vs JSON



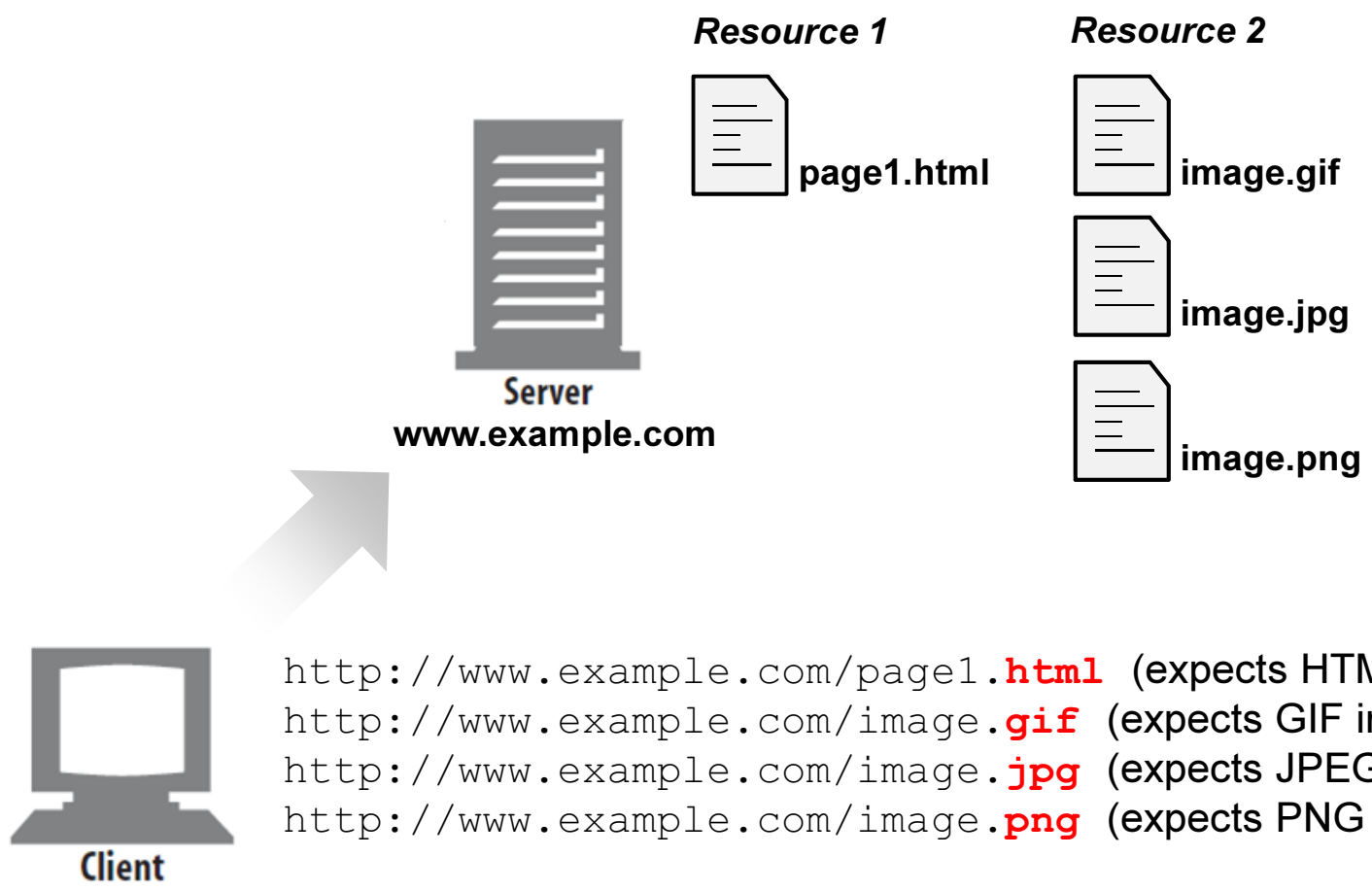


HTML	XML, JSON
Human web	Programmable web
Predefined and standardized markup semantics	Application-specific markup semantics

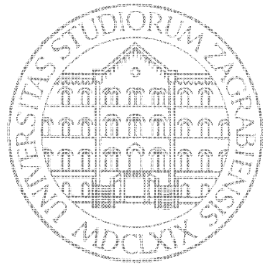
XML	JSON
Bulky representations due to extensive markup	Compact representations
Lack of data types	Native mapping to almost any programming language's data type system
Generic and unconstrained	Somewhat limited with a predefined set of data types

Media Types

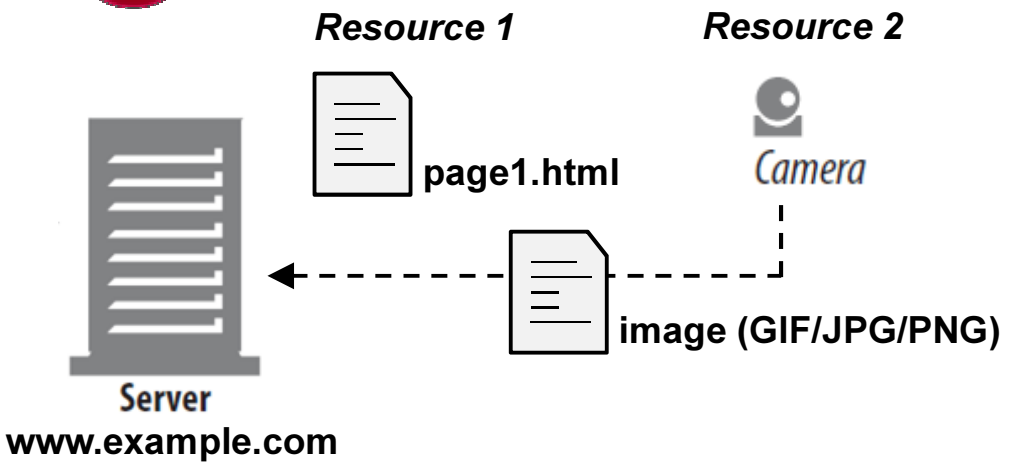
- How to tell which resource representation is in use?
  - File extension ?



Media Types



- How to tell which resource representation is in use?
  - File extension ?



`http://www.example.com/page1.html` (expects HTML document)

`http://www.example.com/image` (no file extension in URL)

What to expect?

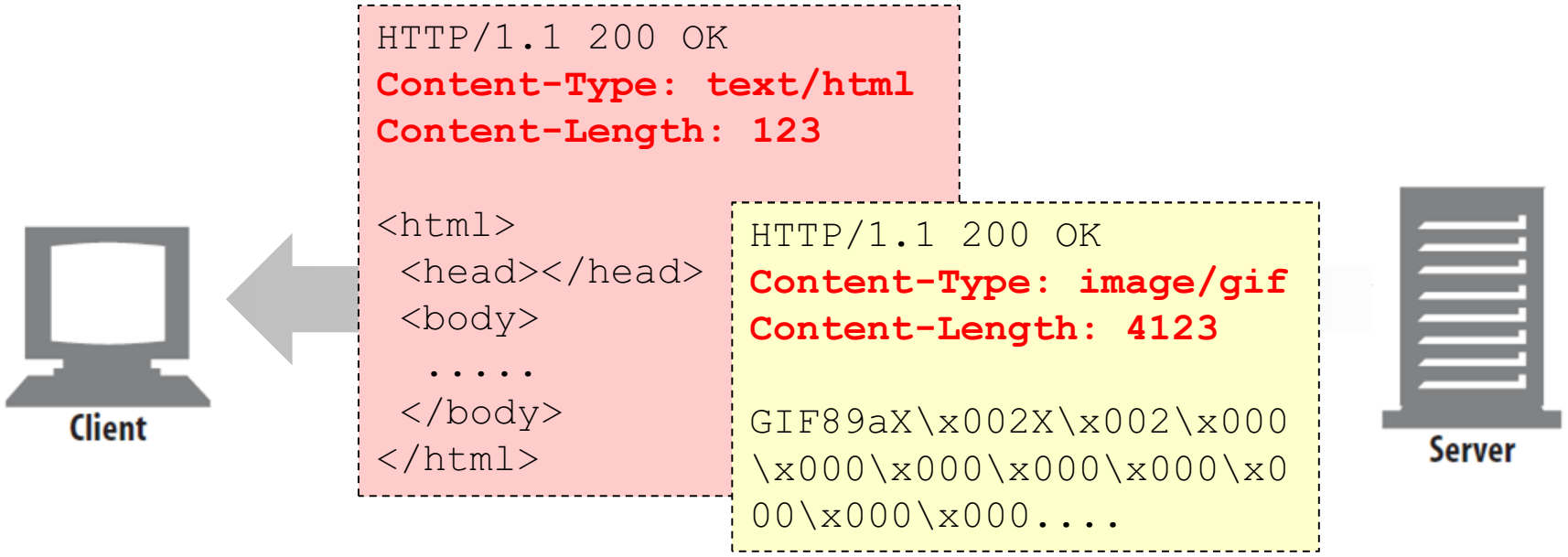
HTTP relies on media types, not on file extensions

Media Types

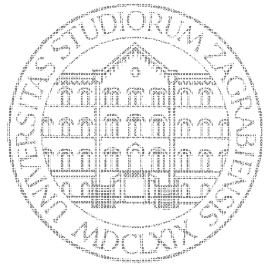
- How to tell which resource representation is in use?
  - HTTP metadata in request/response headers

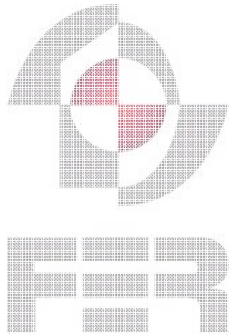
`Content-Type`
- Tells how to interpret the data in request/response body

`Content-Length`
- Specifies the length of the data in request/response body



Media Types





- Media type syntax (MIME types)

```
type / subtype ; [parameter]
```

  - `type`
    - The media category

      `text, image, audio, video, message, model, application, multipart`

  - `subtype`
    - Exact data type under the media category

      `text/html, text/plain, ...`

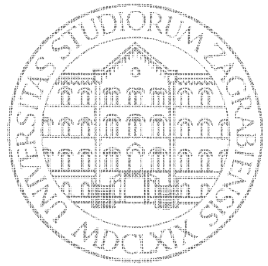
      `image/gif, image/jpeg, image/png, ...`

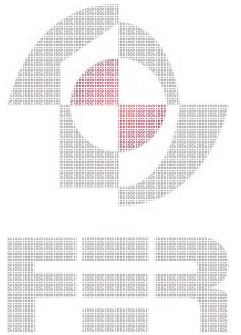
  - `parameter`
    - Optional part that may follow the `type/subtype` in the form of `attribute=value` pairs

      `Content-Type: text/html; charset=ISO-8859-4`

      `Content-Type: text/html; charset="us-ascii"`

Media Types





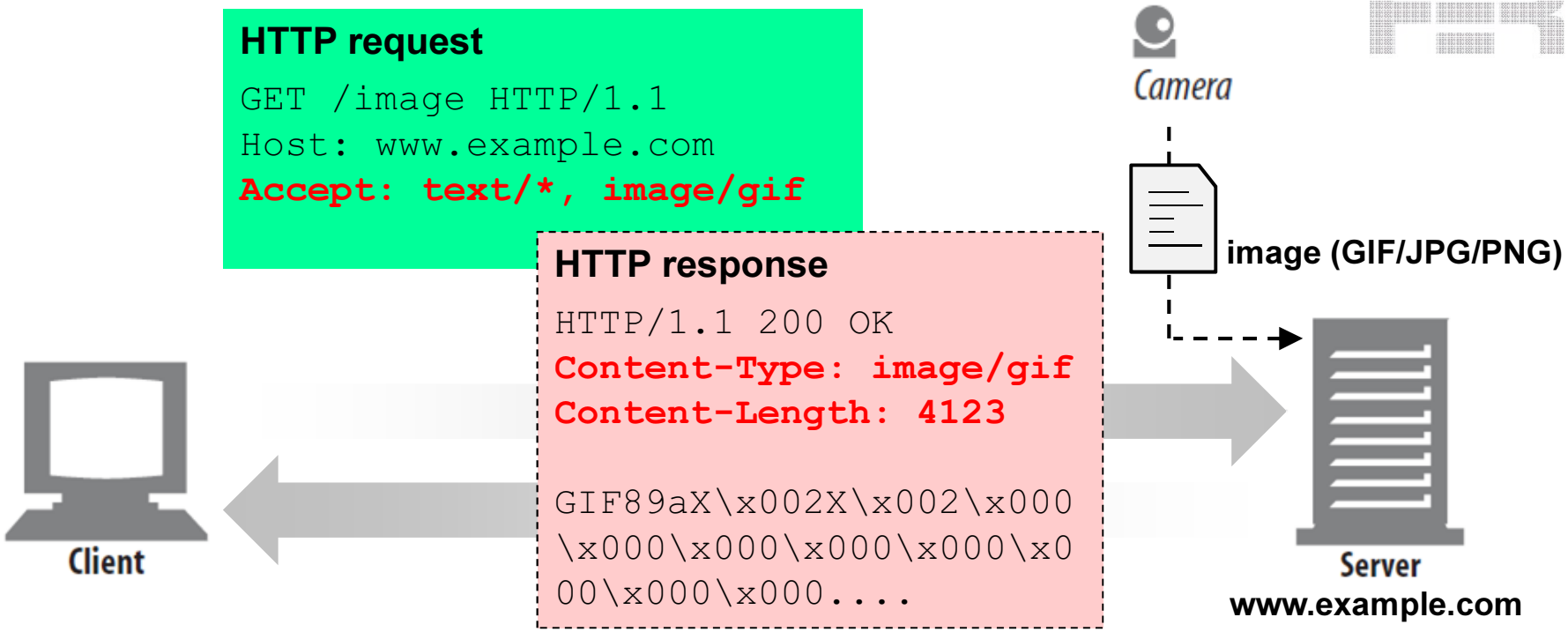
- The *Internet Assigned Numbers Authority* (IANA) governs the set of registered media types

  `http://www.iana.org/assignments/media-types`

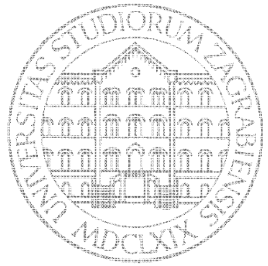
`text/plain`	A plain text
`text/html`	Text formatted as HTML
`application/xml`	Text formatted as XML
`application/json`	Text formatted as JSON
`application/javascript`	Source code written in JavaScript
`application/pdf`	PDF documents
`image/jpeg`	JPEG lossy compressed image
`image/gif`	GIF lossless compressed image
`audio/mpeg`	MP3 or other MPEG audio
`video/mpeg`	MPEG-1 video
`video/mp4`	MPEG-4 video
`application/octet-stream`	Arbitrary binary data
`application/zip`	ZIP archive file

Media Types

- How to communicate preferred resource representation if multiple are available?
- Content negotiation
  - Client specifies what representations it is ready to accept
  - Server responds with one of the acceptable representations



Media Types



- Content negotiation HTTP headers
  - Request headers
  - Client initiates content negotiation

Request header	Description

`Accept`	Tells the server what media types are okay to send
	`Accept: */*`
	`Accept: text/*, image/*`
	`Accept: text/*, image/gif, image/jpeg`
	`Accept: image/gif;q=0.5, image/jpeg;q=1, image/bmp;q=0`

`Accept-Charset`	Tells the server what charsets are okay to send
	`Accept-Charset: *`
	`Accept-Charset: iso-latin-1`

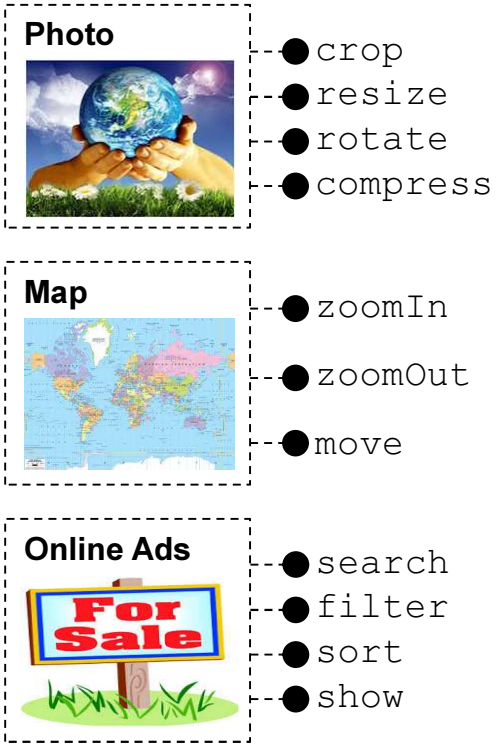
`Accept-Encoding`	Tells the server what encodings are okay to send `Accept-Encoding:` (unencoded content only) `Accept-Encoding: gzip`
`Accept-Language`	Tells the server what languages are okay to send `Accept-Language: en;q=0.5, fr;q=0.0, nl;q=1.0, tr;q=0.0`

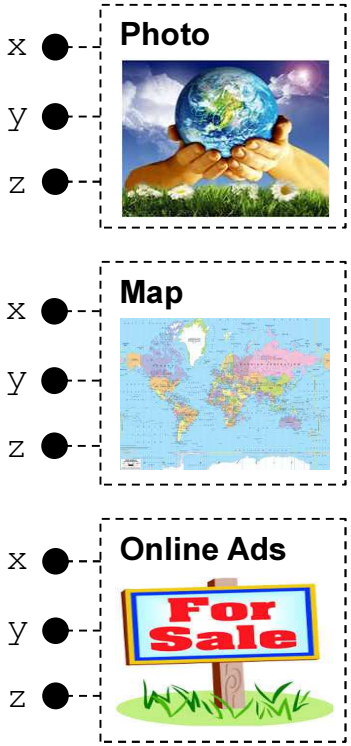
REST Constraints

- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

Uniform Interface

- Resource manipulation
  - Operations exposed to clients





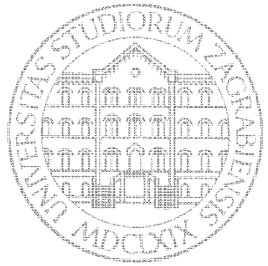
Resource-specific interface

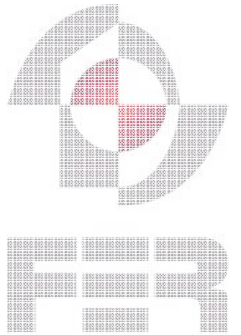
- Each resource has its own set of operations
- Different resources may have different number of operations

Uniform interface

- Different resources
- The same set of operations
- The finite set of operations

Uniform Interface





- Why the uniform interface matters?

Uniform interface allows clients to access every RESTful web service in the same manner, using a small set of interaction rules

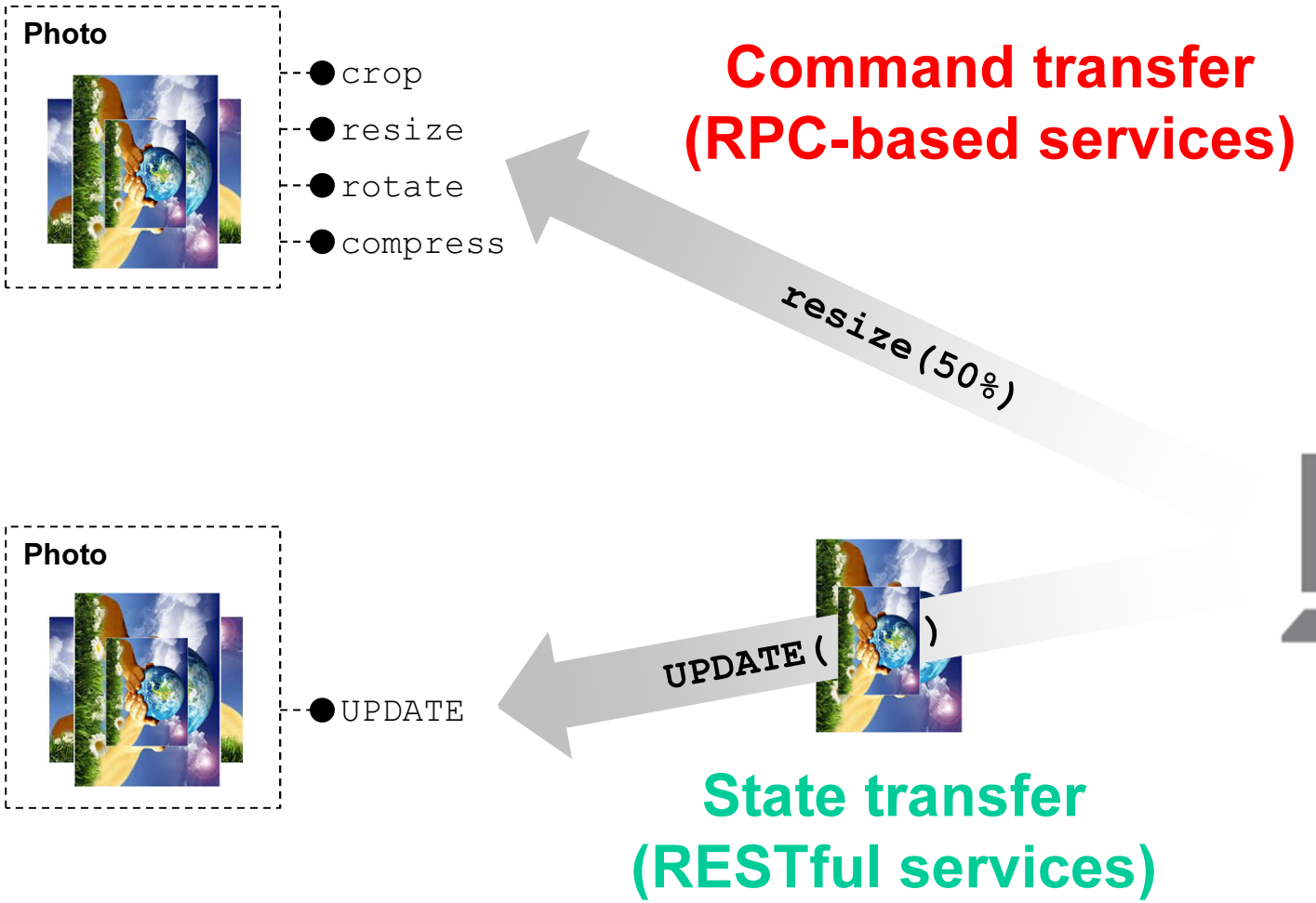
UNIVERSAL CLIENTS

- Teach your client (computer program) how to use the interface once, apply the same client to use every RESTful web service
  - No interface description languages
  - No interface handshaking prior to actual interaction
  - No client upgrades with every service change

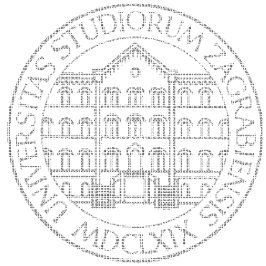
- Given a URI of a resource, there is no question how to interact with the resource

Uniform Interface

- How to achieve uniform interface?

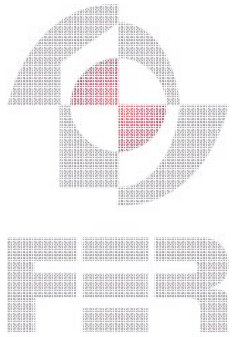


Uniform Interface



- How to achieve uniform interface?

Do not bind the operation to the semantics of the particular resource, but to the representation-level manipulation



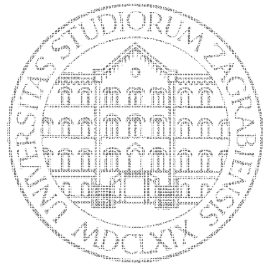
- Separation of concerns (client-server model)
  - Client interprets the resource semantics and prepares modifications of resource representation
  - Server performs just basic manipulations with representation

- CRUD operations

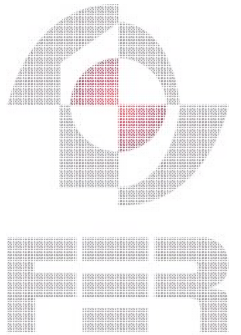
Operation	SQL	HTTP
Create	INSERT	PUT, POST
Read	SELECT	GET, HEAD, OPTIONS
Update	UPDATE	PUT
Delete	DELETE	DELETE

HTTP methods (HTTP verbs)

Uniform Interface



- **RE**presentational **S**tate **T**ransfer



Request

```
GET /music/artists/magnum/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

Verb

Noun

Response

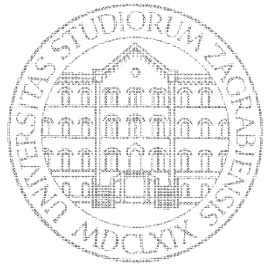
Status code

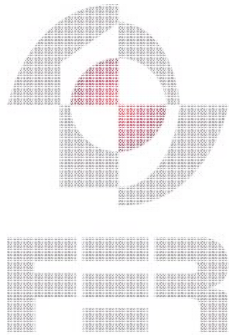
```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

State transfer

Representation

HTTP Methods

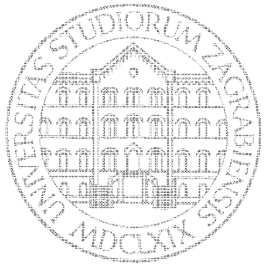


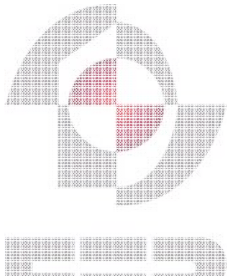


- HTTP methods overview

Method	Description	Message Body
`GET`	Get a resource representation from the server	No
`PUT`	Store data specified in the request body to the server as a new resource representation	Yes
`POST`	Send data to the server for processing	Yes
`DELETE`	Remove a resource from the server	No
`HEAD`	Get just the headers for a resource representation from the server, not the representation itself	No
`OPTIONS`	Determine what methods are available to perform over a resource	No

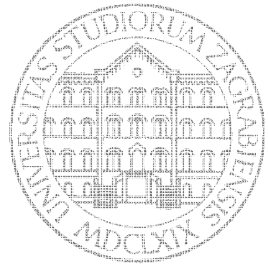
HTTP Methods



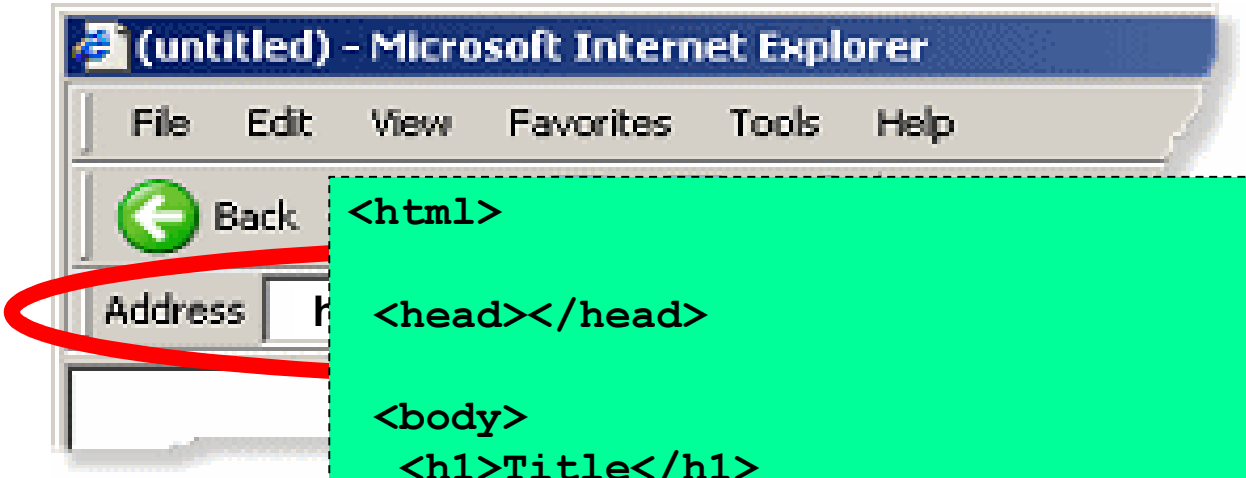


- GET
  - The most common method in HTTP
  - Used to ask a server to send a resource representation
  - The server sends back a representation in response body (response entity body)
  - **Client's request never contains a message body**

HTTP Methods

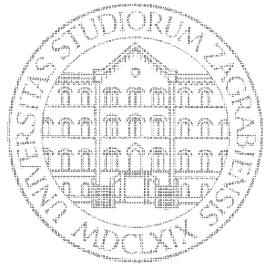


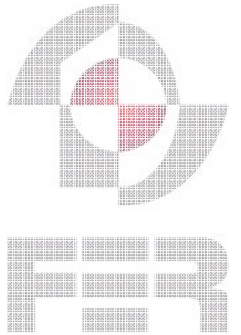
- GET




```
<html>

 <head></head>

 <body>
  <h1>Title</h1>
  <h2>Subtitle</h2>
  <p>Text paragraph</p>
  <img src="http://www.c.com/image.gif" />
  <a href="http://www.b.com/page2.html">Another page</a>
  <form>
   <input type="text" />
   <input type="submit" />
  </form>
 </body>

</html>
```

HTTP Methods





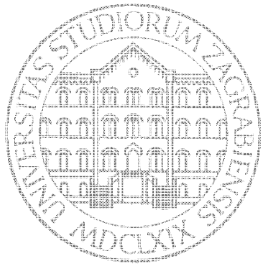
- GET
- Request URI may refer to
  - Static resource stored in file
    - File content is returned in response body
  - Static or dinamic resource stored in database
    - Data retrieved from a database are returned in the response body
  - Data-producing process
    - Server executes the process associated with the given URI
    - Data produced by that executable process are returned in the response body (instead of the process itself, for example, process source code or process executable code)

Reminder:

HTTP relies on media types, not on file extensions

- From the client's perspective, URI may always be considered to refer to a data-producing process
  - In its simplest form, data-producing process is a server code that reads a file content from the disk and writes it into the response body

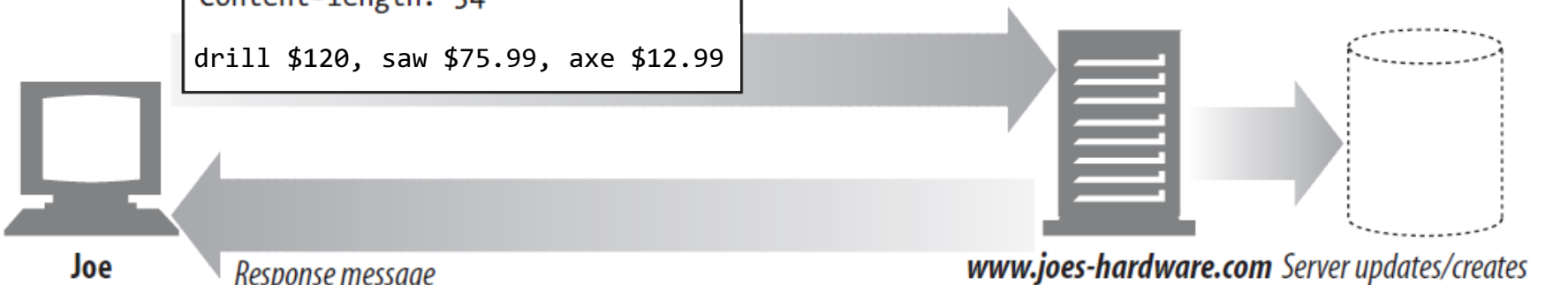
HTTP Methods



- PUT
  - Writes resource representations to a server
    - The inverse of the way that GET reads representations from a server
  - **Almost always** contains a resource representations in client's request body

Request message

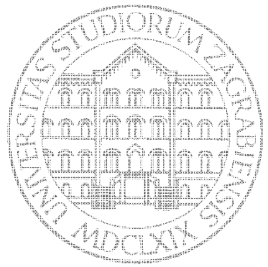
```
PUT /product-list.txt HTTP/1.1
Host: www.joes-hardware.com
Content-type: text/plain
Content-length: 34

drill $120, saw $75.99, axe $12.99
```

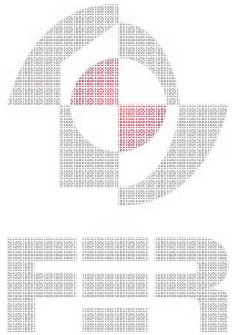


Response message

```
HTTP/1.1 201 Created
Location: http://www.joes-hardware.com/product-list.txt
Content-Type: text/plain
Context-Length: 47

http://www.joes-hardware.com/product-list.txt
```

HTTP Methods



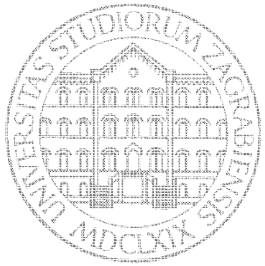


- PUT

- Creation of a new resource
  - Client makes a PUT request to a **non-existent URI**
  - Server creates a new resource and **assigns it the client-provided URI**
  - Server constructs a resource state based on representation provided in client's request
  - Request body may be empty if
    - There is no additional resource state, except the resource URI
    - Resource representation can be fully specified in URI query
  - Server responds with `201 Created` with `Location` header set to the URL of new resource

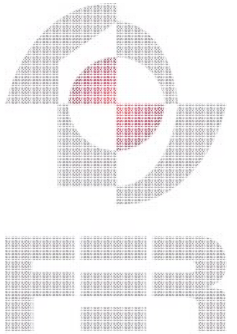
- Update of an existing resource
  - Client makes a PUT request to **an existing URI**
  - Server replaces existing resource representation with a new representation provided in client's request
  - Server responds with `200 OK`

HTTP Methods



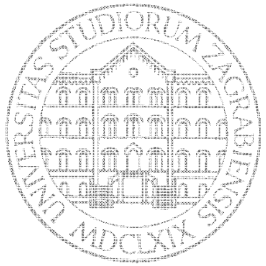
- POST

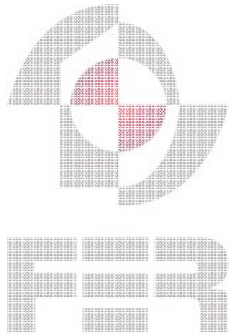
- Creation of a subordinate resources within collections
  - Client makes a POST request to a **collection URI**
  - Server creates a new resource within given collection and **assigns it the server-generated URI**
  - Server constructs a resource state based on representation provided in client's request
  - Server responds with `201 Created` with `Location` header set to the URL of new resource



	PUT to a new resource	PUT to an existing resource	POST
`/weblogs`	N/A (resource already exists)	No effect	Create a new weblog
`/weblogs/myweblog`	Create this weblog	Modify this weblog's settings	Create a new weblog entry
`/weblogs/myweblog/entries/1`	N/A (how would you get this URI?)	Edit this weblog entry	Post a comment to this weblog entry

HTTP Methods





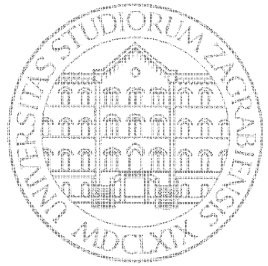
- POST
- Execution of algorithmic resources for which no CRUD method is appropriate
  - Used to send input data to algorithms that execute on server
    - e.g. sending email or SMS message, making alert
- Tunneling of other CRUD methods when client does not support them
  - **This is not a RESTful way of doing things and should be avoided**
  - In practice often used to support HTML forms in web browsers
    - Browsers DO NOT support entire set of HTTP operations
    - GET and POST are typically supported
    - PUT, DELETE, HEAD, and OPTIONS are not supported
    - POST is used to emulate non-supported HTTP methods

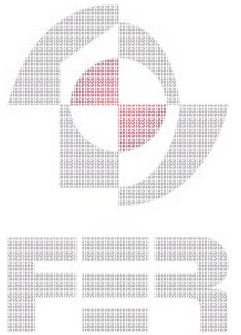
HTTP Methods

- POST
  - Sending HTML form to a server for processing

```
<html>

 <head></head>

 <body>
  <h1>Title</h1>
  <h2>Subtitle</h2>
  <p>Text paragraph</p>
  <img src="http://www.c.com/image.gif" />
  <a href="http://www.b.com/page2.html">Another page</a>
  <form method="POST" action="http://api.sms.com/sendsms">
   <input type="text" name="phonenumber" />
   <input type="text" name="msgtext" />
   <input type="submit" />
  </form>
 </body>

</html>
```

HTTP Methods





- POST
  - Sending HTML form to a server for processing

HTTP request

```
POST /sendsms HTTP/1.1
Host: api.sms.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 55

phonenumber=00385981234567&msgtext=Hello%2C%20students
```

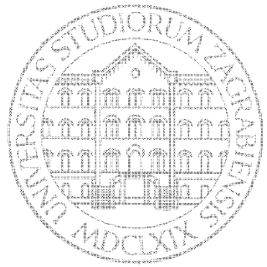
  - Input parameters are encoded into request body (URL-encoding)
    - `name=value` pairs delimited with `&`
    - `name` – name of input parameter (e.g. `phonenumber`)
    - `value` – value of input parameter (e.g. `00385981234567`)
  - Characters not allowed in URLs are percent-encoded
    - `%HH` (`%` = prefix, `HH` = hexadecimal code of given character)
      e.g. comma (decimal ASCII code 44, hex ASCII code 2C) = `%2C`
      e.g. space (decimal ASCII code 32, hex ASCII code 20) = `%20`

HTTP Methods

- POST
  - Using GET instead of POST to send HTML form

```
<html>

 <head></head>

 <body>
  <h1>Title</h1>
  <h2>Subtitle</h2>
  <p>Text paragraph</p>
  <img src="http://www.c.com/image.gif" />
  <a href="http://www.b.com/page2.html">Another page</a>
  <form method="GET" action="http://api.sms.com/sendsms">
   <input type="text" name="phonenumber" />
   <input type="text" name="msgtext" />
   <input type="submit" />
  </form>
 </body>

</html>
```

HTTP Methods



- POST
  - Using GET instead of POST to send HTML form

HTTP request

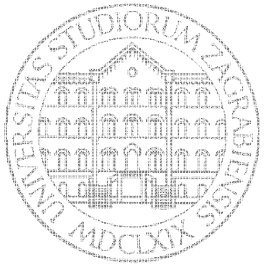
```
GET /sendsms?phonenumber=00385981234567&msgtext=Hello%2C%20students HTTP/1.1
Host: api.sms.com
```

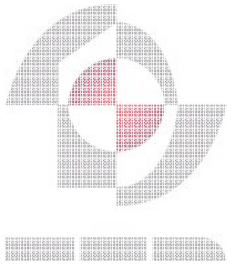
  - Input parameters are encoded as URL query
    - The same way of encoding as in POST method, except that parameters are placed in URL query instead of request body
  - Advantage of POST over GET
    - Servers often limit the length of URL (typically 4 kB or 8 kB), but do not limit the length of request body
      - `GET` – to send small amount of data to server
      - `POST` – to send large amount of data to server
  - Advantage of GET over POST
    - Requests may be bookmarked by clients

Using **GET** method for operation that **DOES NOT retrieve** a resource from a server

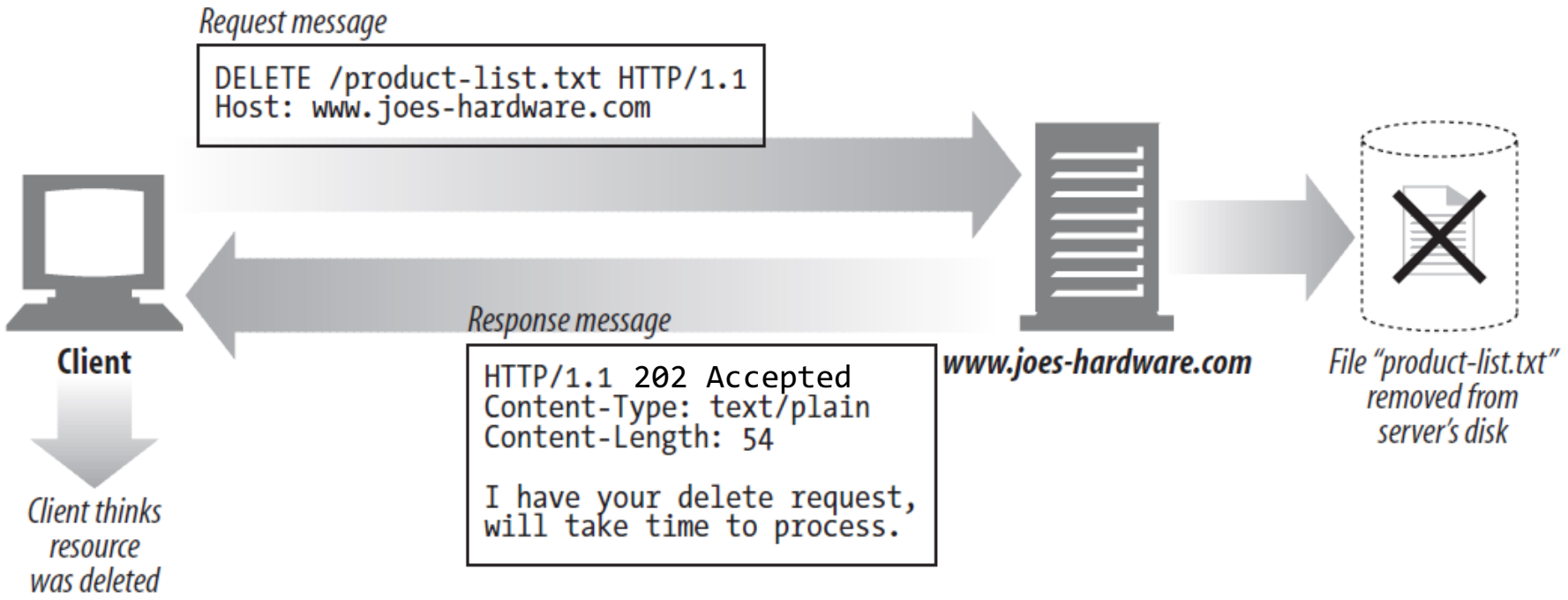
Not RESTful, avoid this whenever possible!

HTTP Methods





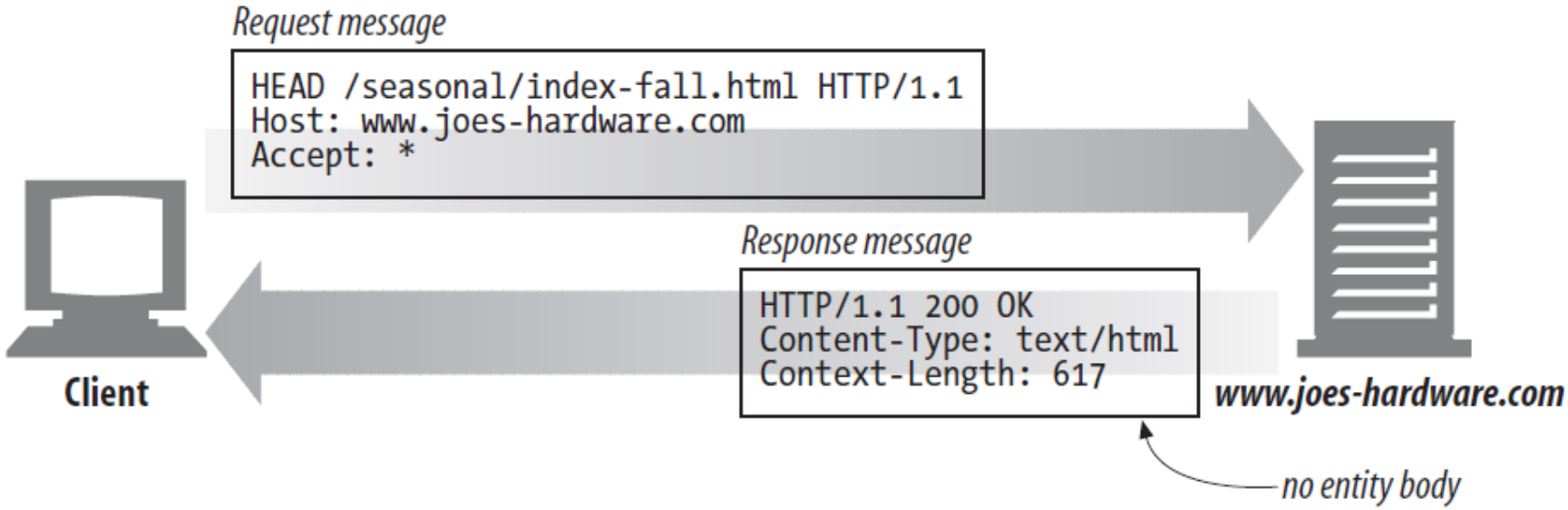
- DELETE
  - Used to ask the server to delete the resource specified by the request URI
  - Server returns `200 OK` if it deletes the resource immediately
  - Server returns `202 Accepted` if it postpones the deletion or deletion takes time



HTTP Methods

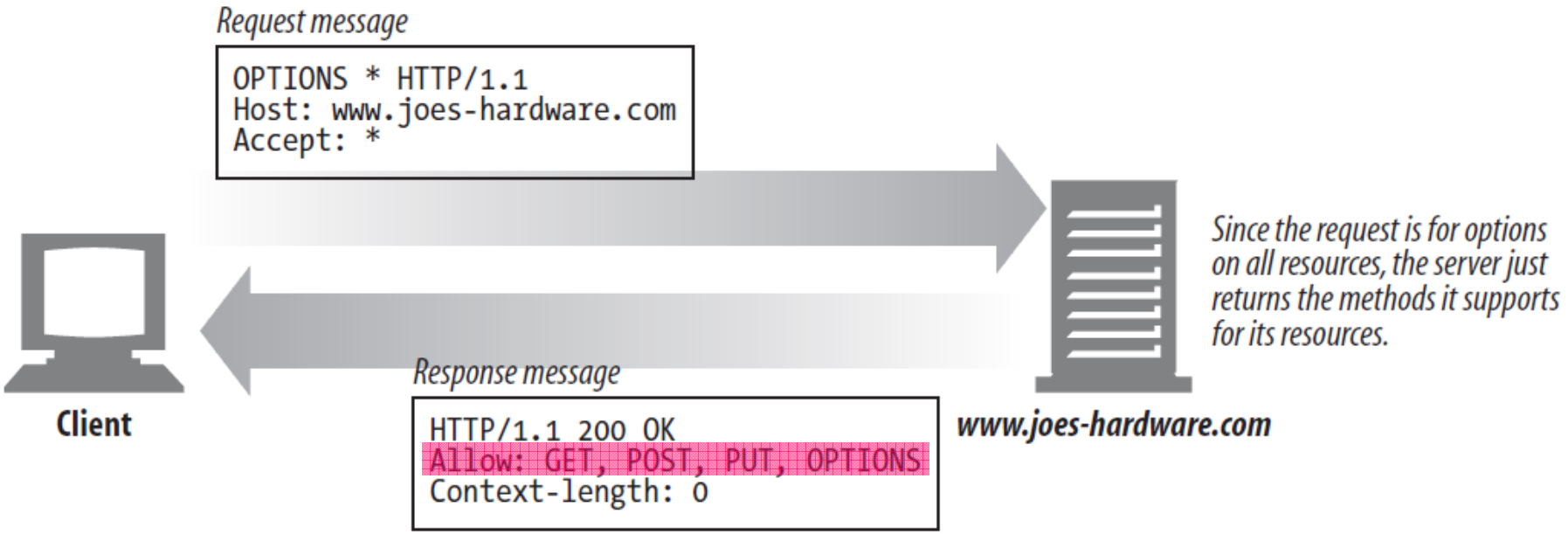
- HEAD
  - Behaves exactly like the GET, but the server returns only the headers in the response, no entity body is ever returned
  - This allows a client to

- Find out about a resource (e.g., determine its type and size) without getting it
- See if an object exists, by looking at the status code of the response **Scalability**
- Test if the resource has been modified, by looking at the headers

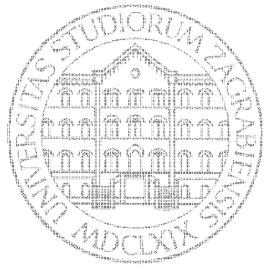


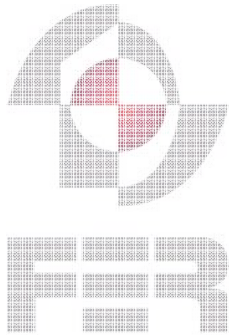
HTTP Methods

- OPTIONS
  - Uset to inspect resource's uniform interface
  - Asks the server to tell about what methods it supports
    - In general (using * as an URI)
    - For particular resources (using particular resource URI)
  - This provides a means for client applications to determine how best to access various resources without actually having to access them

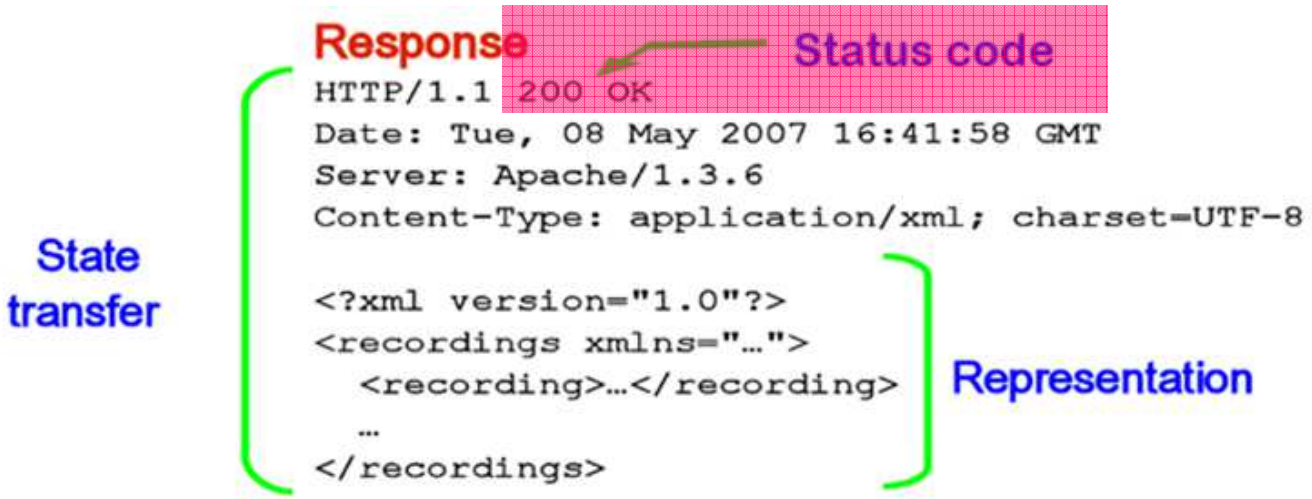


Status Codes





- Mandatory part of the response line
- Provide a server's feedback for clients to understand the results of their transactions



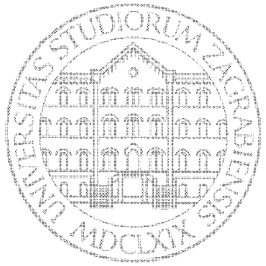
numeric value

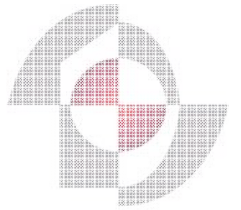
3-digit number that reflects the outcome of the request.
Informs the client whether the request is served successfully, some error occurred, etc. Used for machine interpretation.

reason phrase

Short textual explanation of the status code for human interpretation

Status Codes

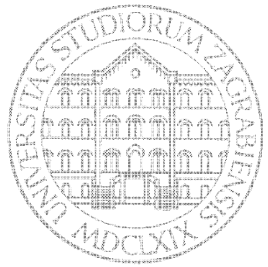


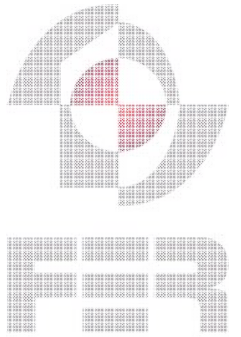


- HTTP status codes are classified into five broad categories
- Each category defines a set of specific codes that describe particular circumstances under the category

Overall range	Defined range	Category
100-199	100-101	Informational
200-299	200-206	Successful
300-399	300-305	Redirection
400-499	400-415	Client error
500-599	500-505	Server error

Status Codes





- Examples of most commonly used status codes

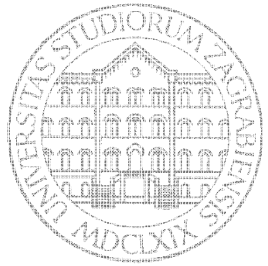
`200 OK`	Request served successfully, response contains the body with resource representation
`301 Moved Permanently`	The resource from the requested URL has been moved to the new URL
`304 Not Modified`	Resource has not changed since the last request, client can use a representation from a local cache
`400 Bad Request`	Tells the client that it sent a malformed request
`401 Unauthorized`	Server requires from the client to authenticate itself before it can gain access to the resource
`403 Forbidden`	The request was refused by the server, for example when client has no privileges to access the resource
`404 Not Found`	The server cannot find the requested URL
`500 Internal Server Error`	The server encountered an error that prevented it from servicing the request
`503 Service Unavailable`	The server cannot currently service the request but will be able to in the future

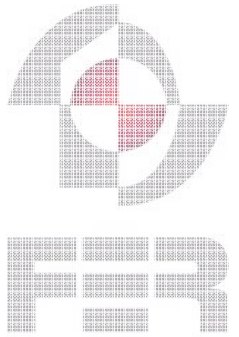
For a complete list see:

1) D. Gourley, B. Totty, M. Sayer, A. Aggarwal, S. Reddy: **HTTP The Definitive Guide, Appendix B**

2) http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Safety and Idempotence

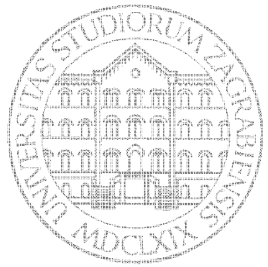




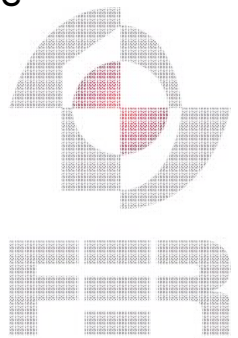
- Safety
  - Operation is **SAFE** if its execution has the same effect on resource as if it was not executed at all
  - The execution of safe operation has no visible effects on server
- Idempotence
  - Operation is **IDEMPOTENT** if its multiple execution over the same resource has the same effect as a single execution

Method	Safe	Idempotent
GET	✓	✓
PUT	⊘	✓
POST	⊘	⊘
DELETE	⊘	✓
HEAD	✓	✓
OPTIONS	✓	✓

Safety and Idempotence

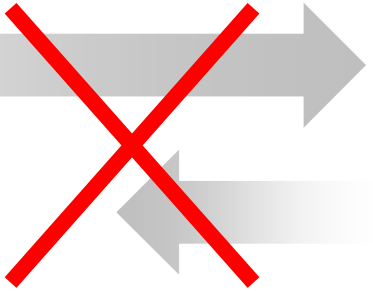


- Why it matters?
  - Reliable HTTP requests over an unreliable network
    - If initial request was not successful for any reason, client just makes another one



HTTP request

```
GET /value HTTP/1.1
Host: www.example.com
```



HTTP request – retry

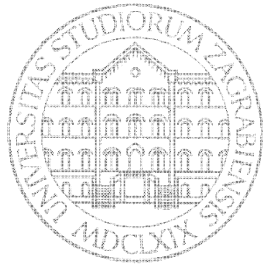
```
GET /value HTTP/1.1
Host: www.example.com
```

`value=10`

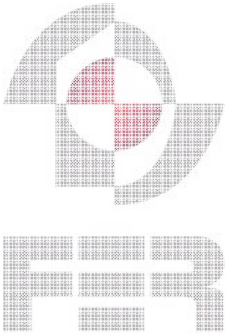
HTTP response

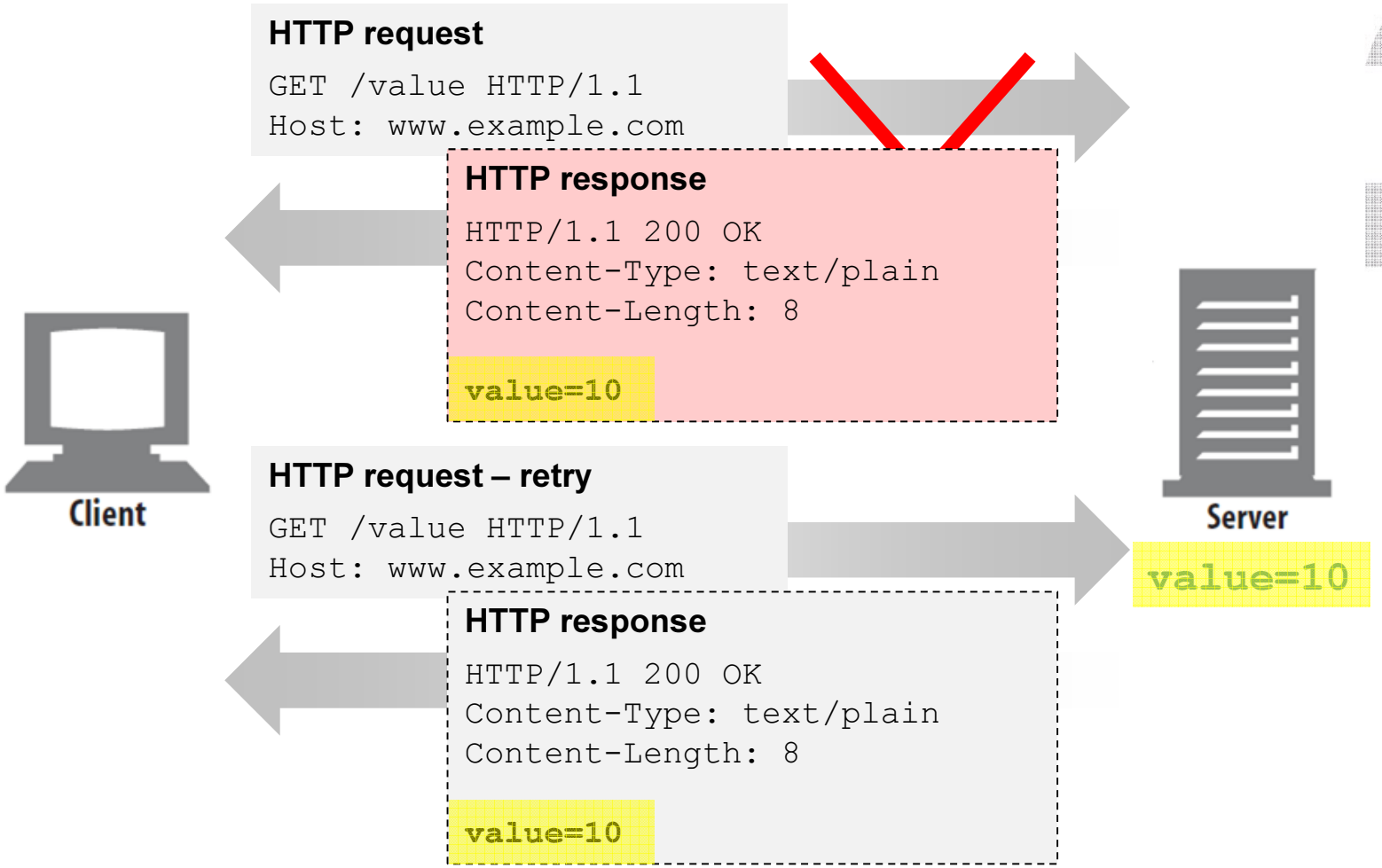
```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 8

value=10
```

Safety and Idempotence

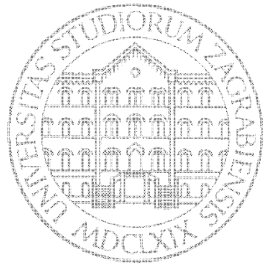


- Why it matters?
  - Reliable HTTP requests over an unreliable network
    - If initial request ever went through, it has **NO REAL effect** on server

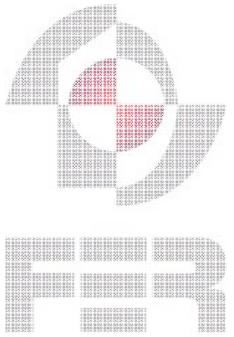




Safety and Idempotence

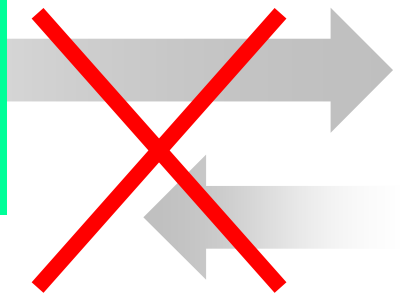


- Why it matters?
  - Reliable HTTP requests over an unreliable network
    - If initial request ever went through, it has **THE SAME effect** as retry request



HTTP request

```
PUT /value HTTP/1.1
Host: www.example.com

value=11
```



HTTP request – retry

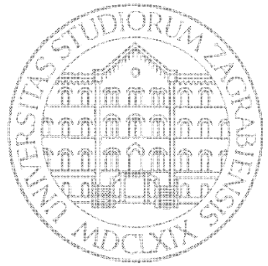
```
PUT /value HTTP/1.1
Host: www.example.com

value=11
```

`value=11`

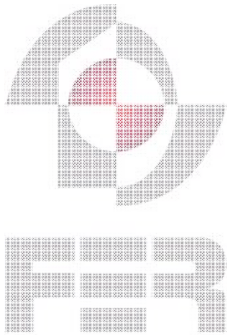
HTTP response

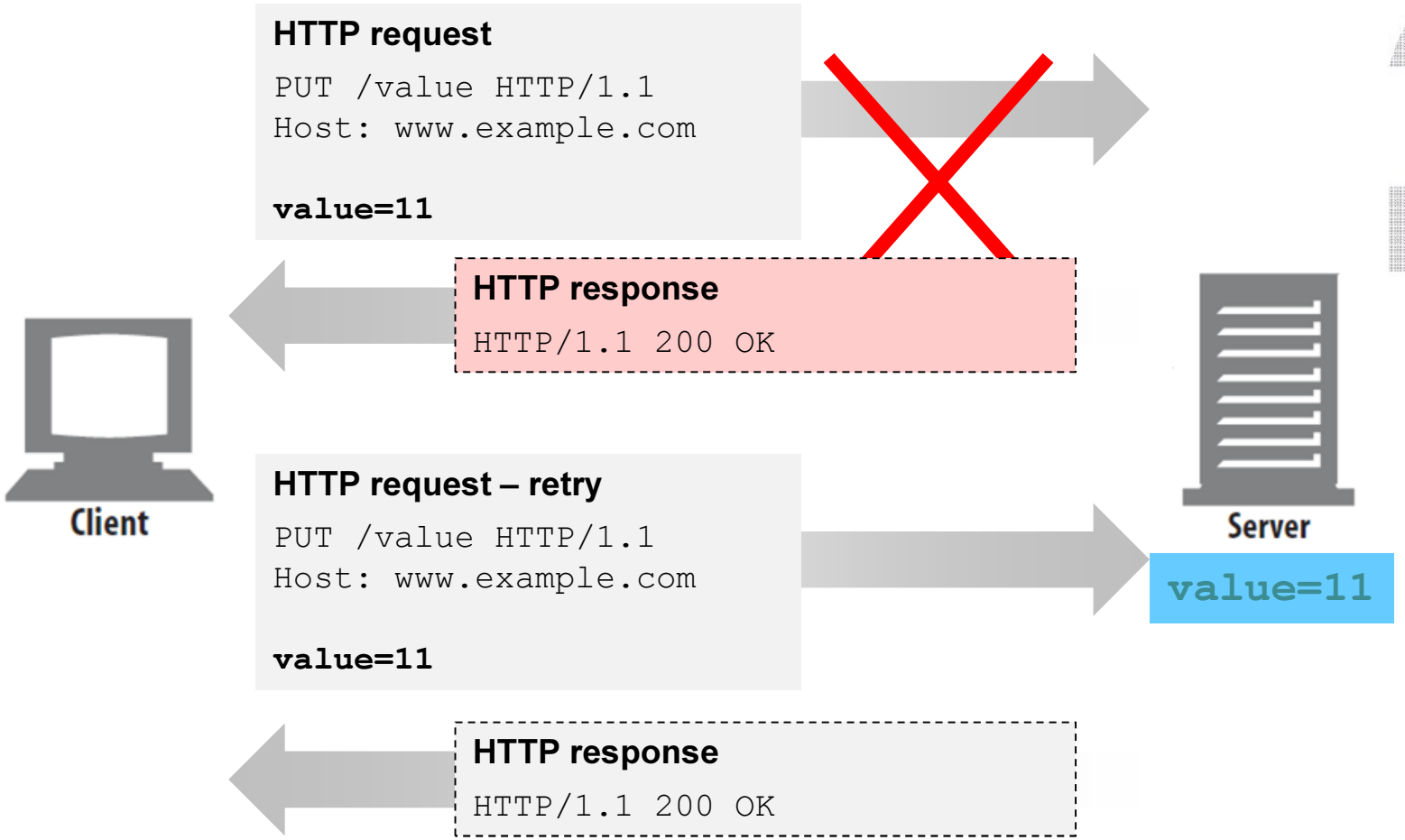
```
HTTP/1.1 200 OK
```

Safety and Idempotence



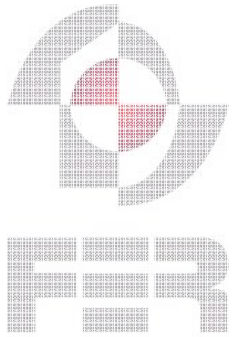
- Why it matters?
  - Reliable HTTP requests over an unreliable network
    - If initial request ever went through, it has **THE SAME effect** as retry request

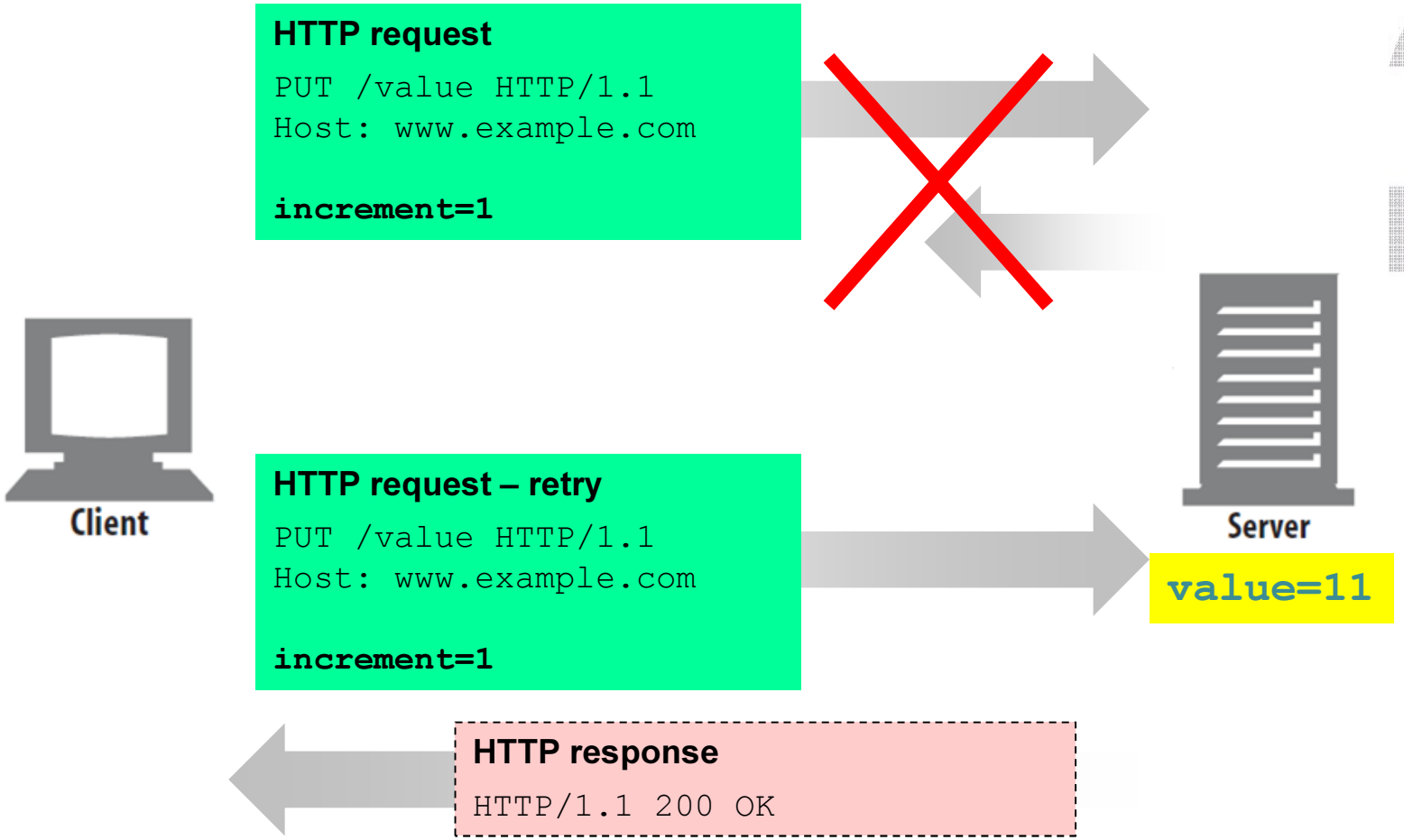




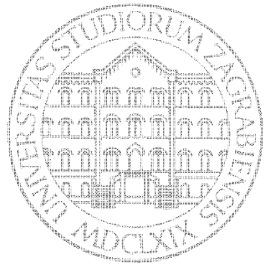
Safety and Idempotence

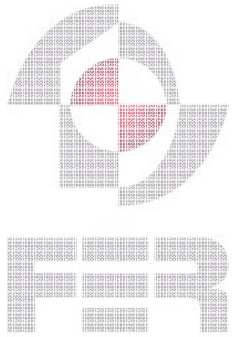
- DO NOT design methods to change resource's state in relative terms





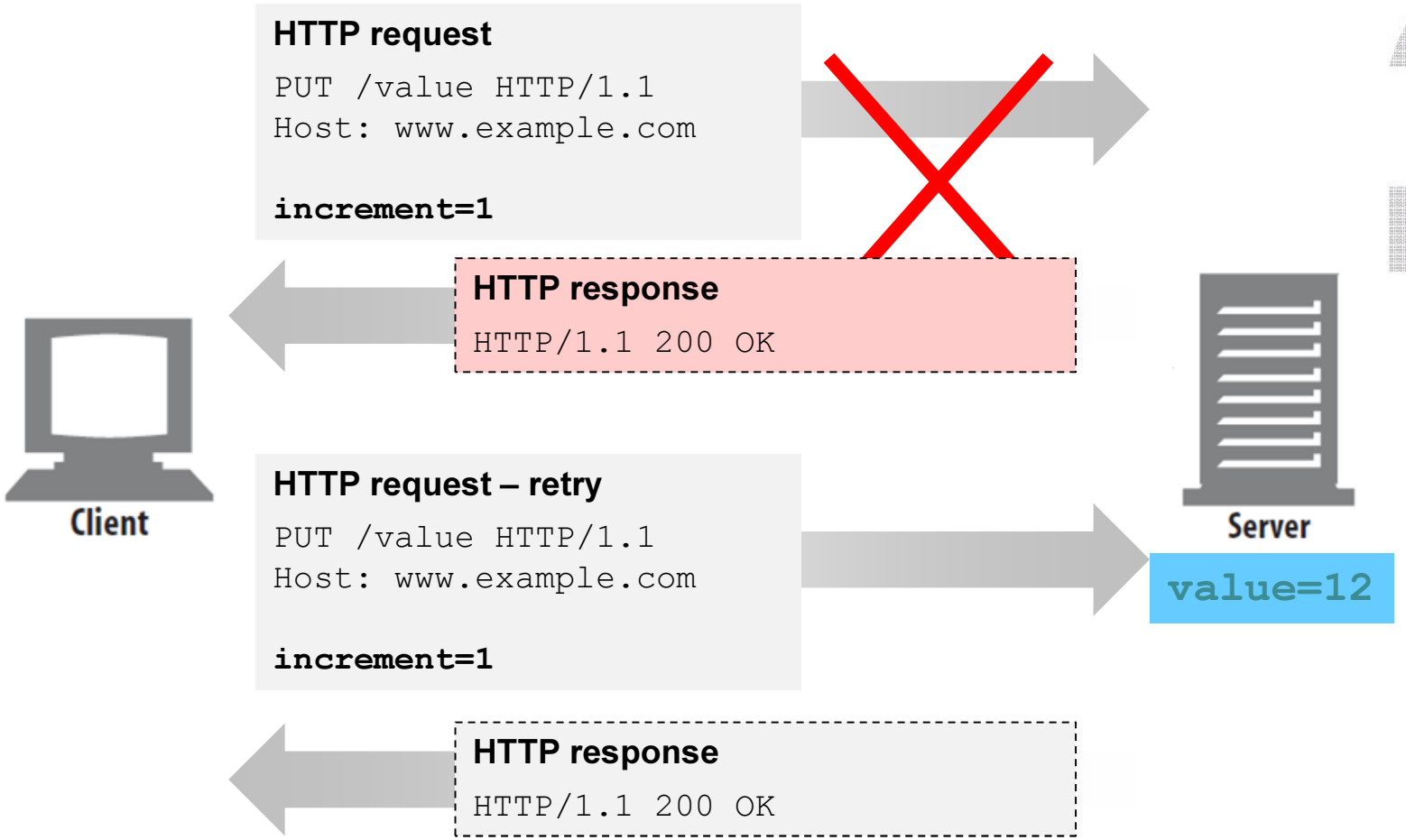
Safety and Idempotence



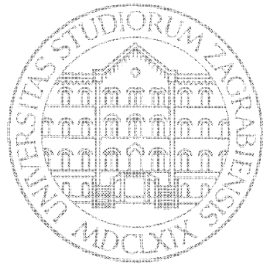


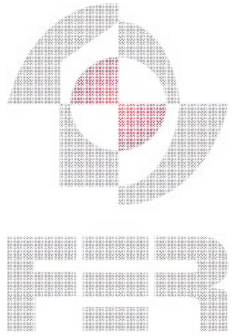
- DO NOT design methods to change resource's state in relative terms

NOT IDEMPOTENT



HTTP Server Programming

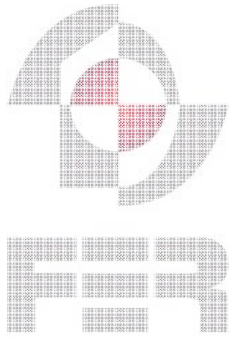




- HTTP server programming using Java servlets
  - HTTP server
    - Accepting TCP connections
    - Request dispatching
    - Multithreading
  - HTTP servlets
    - Request handlers
    - Subclassing the `HttpServlet` class
    - Servlets handle requests by overriding `HttpServlet` methods

```
doGet
doPost
doPut
doDelete
doHead
doOptions
```

HTTP Server Programming



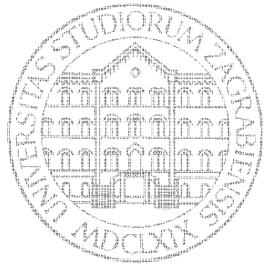
- REST API design
  - Resources and URLs

Resource	URL
Person	`http://hostname/demo/person`
Dog	`http://hostname/demo/dog`

  - Methods and resource representations

Resource	Method	Resource Representation
Person	GET	Firstname: **John** Lastname: **Smith** Age: **25**
	PUT	Age: **34**

HTTP Server Programming



- REST API design
  - Methods and resource representations

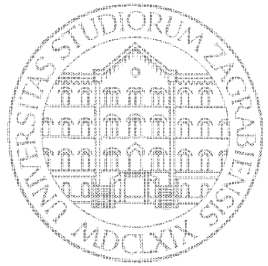
Resource	Method	Resource Representation

Dog	GET	Name: Kai
		Breed: Royal Poodle
		
		`<html><body>`
		`<h1 style="color:blue">`
		`Name: Kai</h1>`
		`<h2 style="color:red">`
		`Breed: Royal Poodle</h2>`
		`</body></html>`

		`<dog>`
		`<name>Kai</name>`
		`<breed>Royal Poodle</breed>`
		`</dog>`

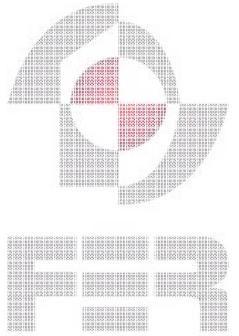
		`{`
		`"name": "Kai",`
		`"breed": "Royal Poodle"`
		`}`

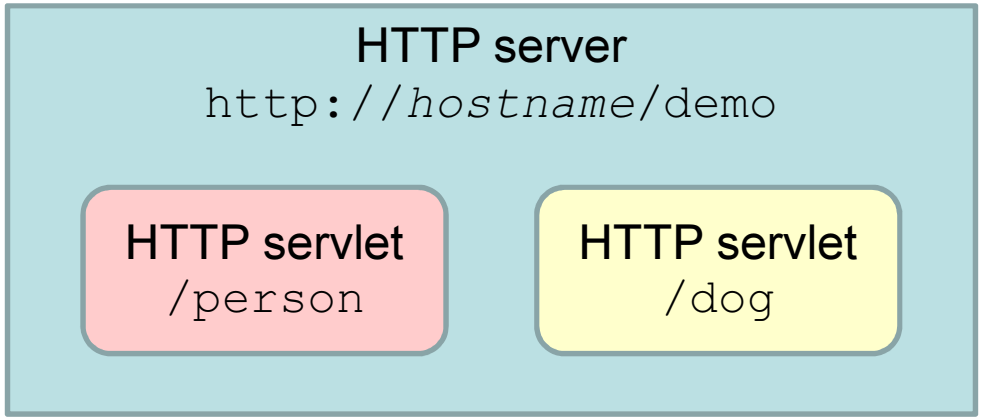
HTTP Server Programming



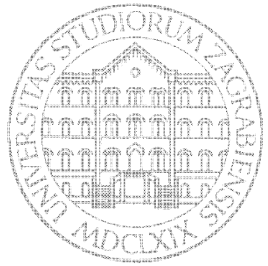
- REST API implementation

Resource	URL
Person	`http://hostname/demo/person`
Dog	`http://hostname/demo/dog`

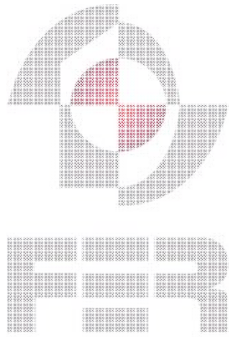




HTTP Server Programming

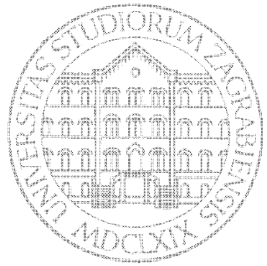


- HTTP server implementation (`SimpleHttpServer.java`)
  - Based on *Jetty* servlet engine (http://www.eclipse.org/jetty/)



```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletHolder;
import org.eclipse.jetty.webapp.WebAppContext;

public class SimpleHttpServer {
  public static void main(String[] args) throws Exception {
    // configure the root directory and root URL
    WebAppContext servlets = new WebAppContext();
    servlets.setResourceBase(".");
    servlets.setContextPath("/demo");

    // register servlets for dynamic resources and assign them URLs
    servlets.addServlet(new ServletHolder(new Person()), "/person");
    servlets.addServlet(new ServletHolder(new Dog()), "/dog");

    // configure the server
    Server simpleServer = new Server(80);
    simpleServer.setHandler(servlets);

    // start the server
    simpleServer.start();
    simpleServer.join();
  }
}
```

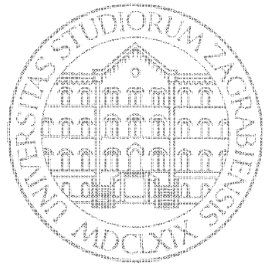
HTTP Server Programming



- HTTP servlet skeleton – subclassing the `HttpServlet`

```
public class Person extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response) {
  }

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response) {
  }

  @Override
  protected void doPut(HttpServletRequest request, HttpServletResponse response) {
  }

  @Override
  protected void doDelete(HttpServletRequest request, HttpServletResponse response) {
  }

  @Override
  protected void doHead(HttpServletRequest request, HttpServletResponse response) {
  }

  @Override
  protected void doOptions(HttpServletRequest request, HttpServletResponse response) {
  }
}
```

HTTP Server Programming



- `Person` servlet implementation (`Person.java`)

```
public class Person extends HttpServlet {
  // part of resource state that users can modify
  private int age = 25;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    response.setStatus(HttpServletResponse.SC_OK);
    response.setContentType("text/plain");
    response.getWriter().println("Firstname: John\r\nLastname: Smith\r\nAge: " + age);
  }

  @Override
  protected void doPut(HttpServletRequest request, HttpServletResponse response) {
    byte[] requestBody = new byte[request.getContentLength()];
    request.getInputStream().read(requestBody);
    String[] requestBodyParts = (new String(requestBody)).split("=");
    age = Integer.parseInt(requestBodyParts[1]);
  }

  @Override
  protected void doOptions(HttpServletRequest request, HttpServletResponse response) {
    response.setStatus(HttpServletResponse.SC_OK);
    response.addHeader("Allow", "GET, PUT, OPTIONS");
  }
}
```

HTTP Basics

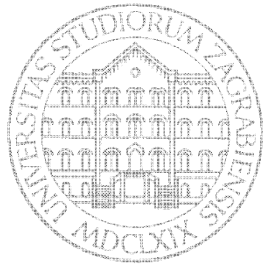
- HTTP server programming using Java servlets
  - Compile program

```
javac -cp lib/* SimpleHttpServer.java Person.java Dog.java
```

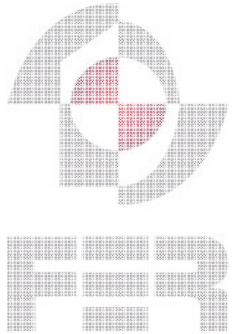
  - Start program

```
java -cp lib/*;. SimpleHttpServer
```

HTTP Basics



- HTTP server programming using Java servlets
  - Test program



```
curl –X OPTIONS http://localhost/demo/person -i

curl –X GET http://localhost/demo/person -i
```
Web browser: `http://localhost/demo/person`

```
curl –X PUT http://localhost/demo/person -d age=56 -i

curl –X GET http://localhost/demo/person -i
```
Web browser: `http://localhost/demo/person`

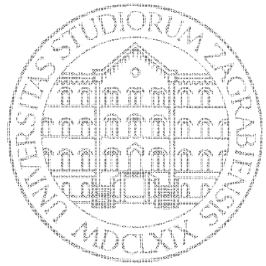
`curl` options
- `-X` specify request method to use
- `-i` include protocol headers in the output
- `-d` specify request body data

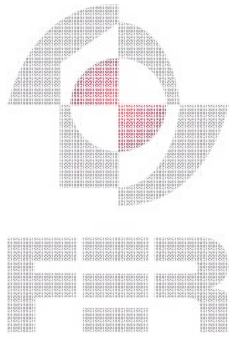
HTTP Server Programming

- `Dog` servlet implementation (`Dog.java`)

```
public class Dog extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    String acceptHeader = request.getHeader("Accept");
    if (acceptHeader == null || acceptHeader.contains("*/*") ||
                                acceptHeader.contains("text/html")) {
      response.setStatus(HttpServletResponse.SC_OK);
      response.setContentType("text/html");
      response.getWriter().println("<html><body>");
      response.getWriter().println("<h1 style=\"color:blue\">Name: Kai</h1>");
      response.getWriter().println("<h2 style=\"color:red\">Breed: Royal Poodle</h2>");
      response.getWriter().println("</body></html>");
    }
    else if (acceptHeader.contains("application/xml")) {
      response.setStatus(HttpServletResponse.SC_OK);
      response.setContentType("application/xml");
      response.getWriter().println("<dog><name>Kai</name><breed>Royal Poodle</breed>
                                    </dog>");
    }
    else if (acceptHeader.contains("application/json")) {
      response.setStatus(HttpServletResponse.SC_OK);
      response.setContentType("application/json");
      response.getWriter().println("{\"name\": \"Kai\", \"breed\": \"Royal Poodle\"}");
    }
    else { response.setStatus(HttpServletResponse.SC_BAD_REQUEST); }
  }
}
```

HTTP Basics





- HTTP server programming using Java servlets
  - Test program

```
curl –X GET http://localhost/demo/dog -i
```

Web browser: `http://localhost/demo/dog`

```
curl –X GET http://localhost/demo/dog
     -H Accept:application/xml -i
```

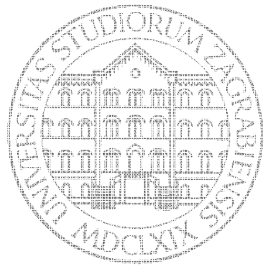
```
curl –X GET http://localhost/demo/dog
     -H Accept:application/json -i
```

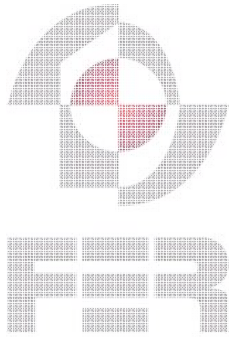
`curl` options

- `-X` specify request method to use
- `-i` include protocol headers in the output
- `-H` define custom header to pass to server

REST Triangle

- Three key elements of a RESTful service



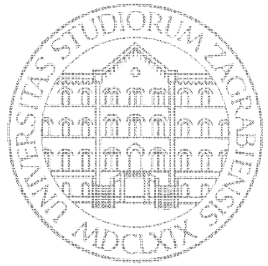


URIs (Nouns)
(unconstrained)
E.g. http://api.example.com/
http://api.example.com/users
http://api.example.com/users/john
http://api.example.com/photos
....

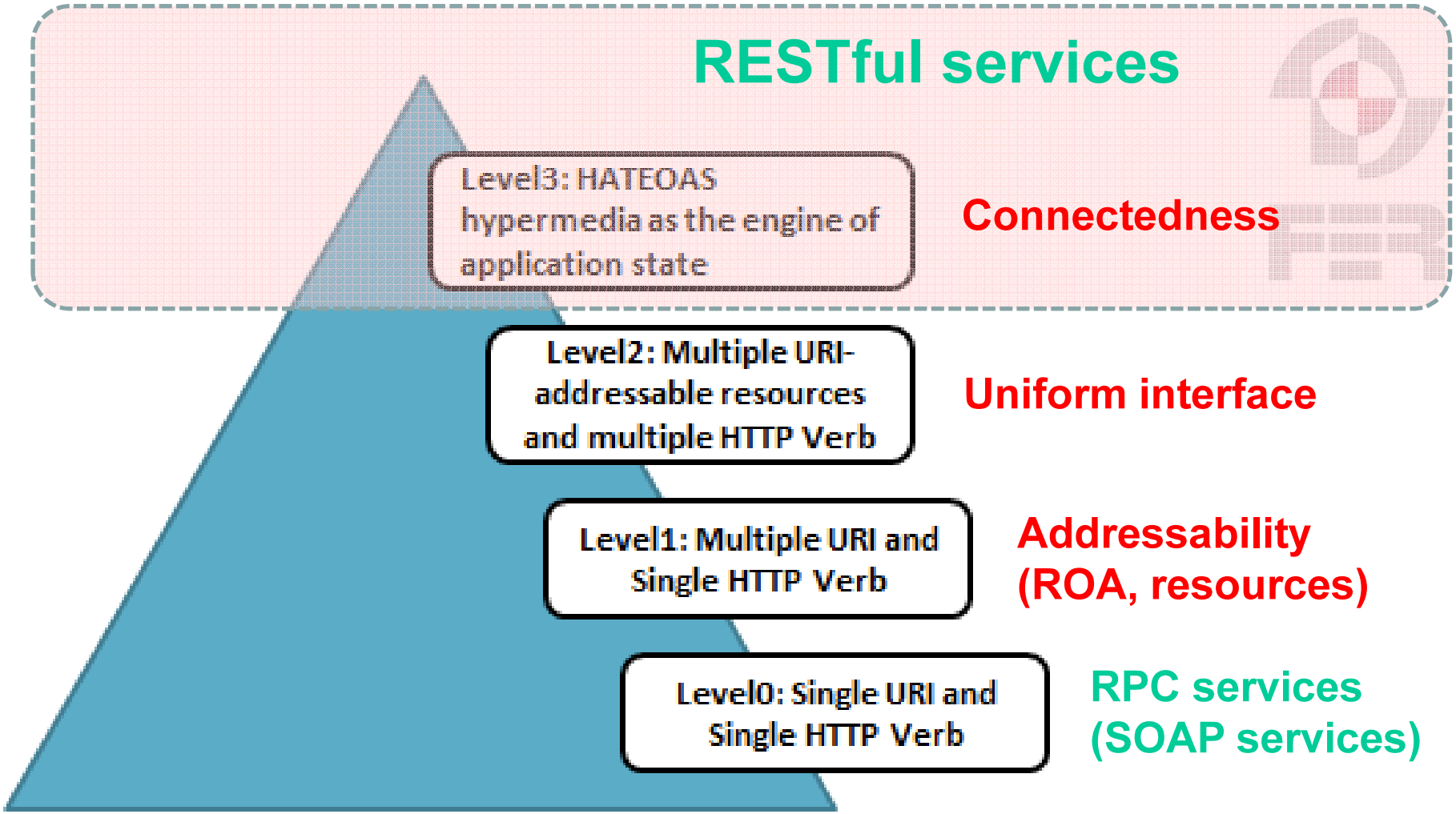
Methods (Verbs)
(constrained)
GET, PUT, POST, DELETE, HEAD, OPTIONS

Media types
(semiconstrained)
HTML, XML, JSON
+
other standard data types

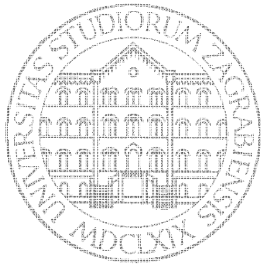
REST Maturity Model

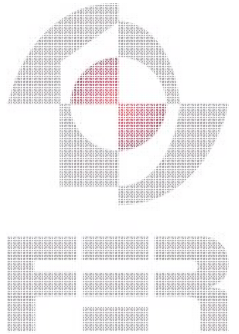


- Richardson's REST maturity model
  - Evaluation to what extent an API conforms to the REST principles

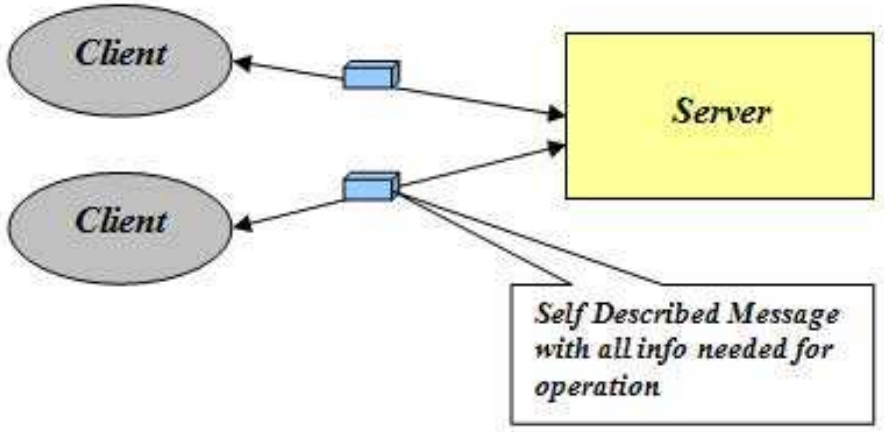


REST Constraints

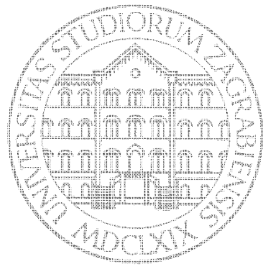




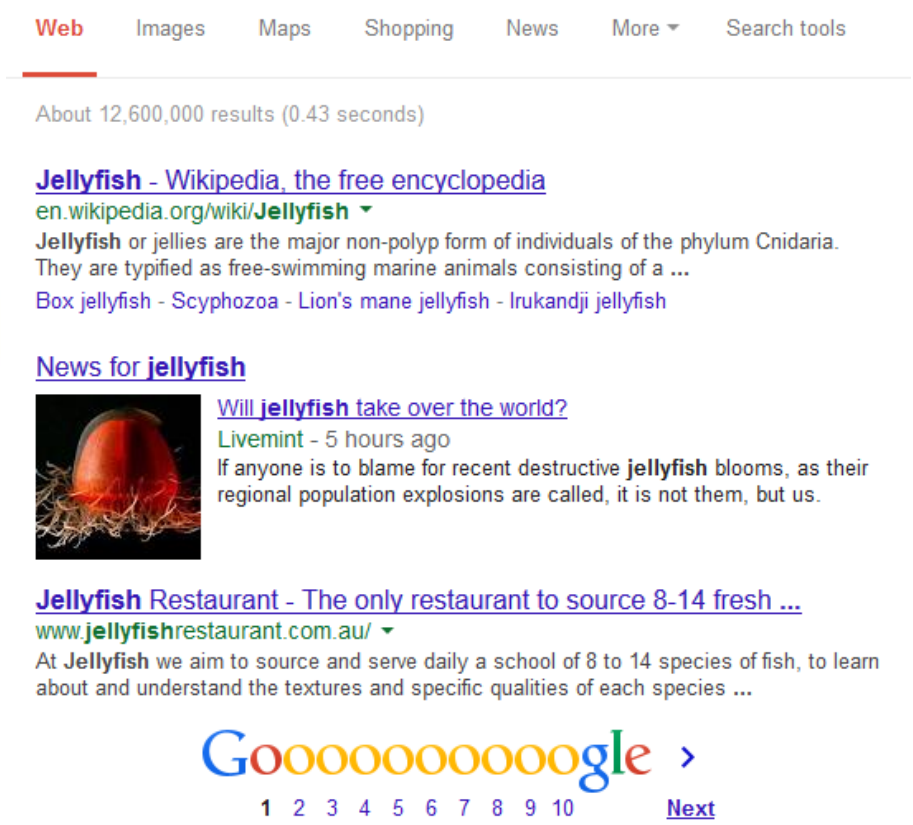
- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand



Statelessness

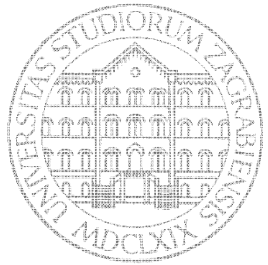


Step 1
`http://www.google.com/`



Step 2
`http://www.google.com/search?q=jellyfish`

Statelessness



Step 3
`http://www.google.com/next`

Web Images Maps Shopping More Search tools

Page 2 of about 12,300,000 results (0.20 seconds)

Images for **jellyfish** - Report images

Jellyfish Pictures
www.**jellyfish**pictures.co.uk/
Jellyfish are a BAFTA award winning visual effects studio based in Soho London specialising in computer graphics animation.

Jellyfish - Enchanted Learning Software
www.enchantedlearning.com/subjects/.../**jellyfish**/**Jellyfish**coloring.shtml
Printable chart and fact sheet show the life cycle of the **jellyfish**. From EnchantedLearning.com.

‹ Goooooooooogle ›
Previous 1 2 3 4 5 6 7 8 9 10 Next

Step 4
`http://www.google.com/next`

Web Images Maps Shopping More Search tools

Page 3 of about 12,300,000 results (0.19 seconds)

They're Taking Over! by Tim Flannery | The New York Review of Books
www.nybooks.com/articles/archives/2013/.../**jellyfish**-theyre-taking-over/
Sep 26, 2013 - A moon **jellyfish** and cross **jellyfish** floating in a remote channel near Vancouver Island, British Columbia; photograph by David Hall from ...

Meduzot (2007) - IMDb
www.imdb.com/title/tt0807721/
Meduzot (the Hebrew word for **Jellyfish**) tells the story of three very different Israeli women living in Tel Aviv whose intersecting stories weave an unlikely portrait ...

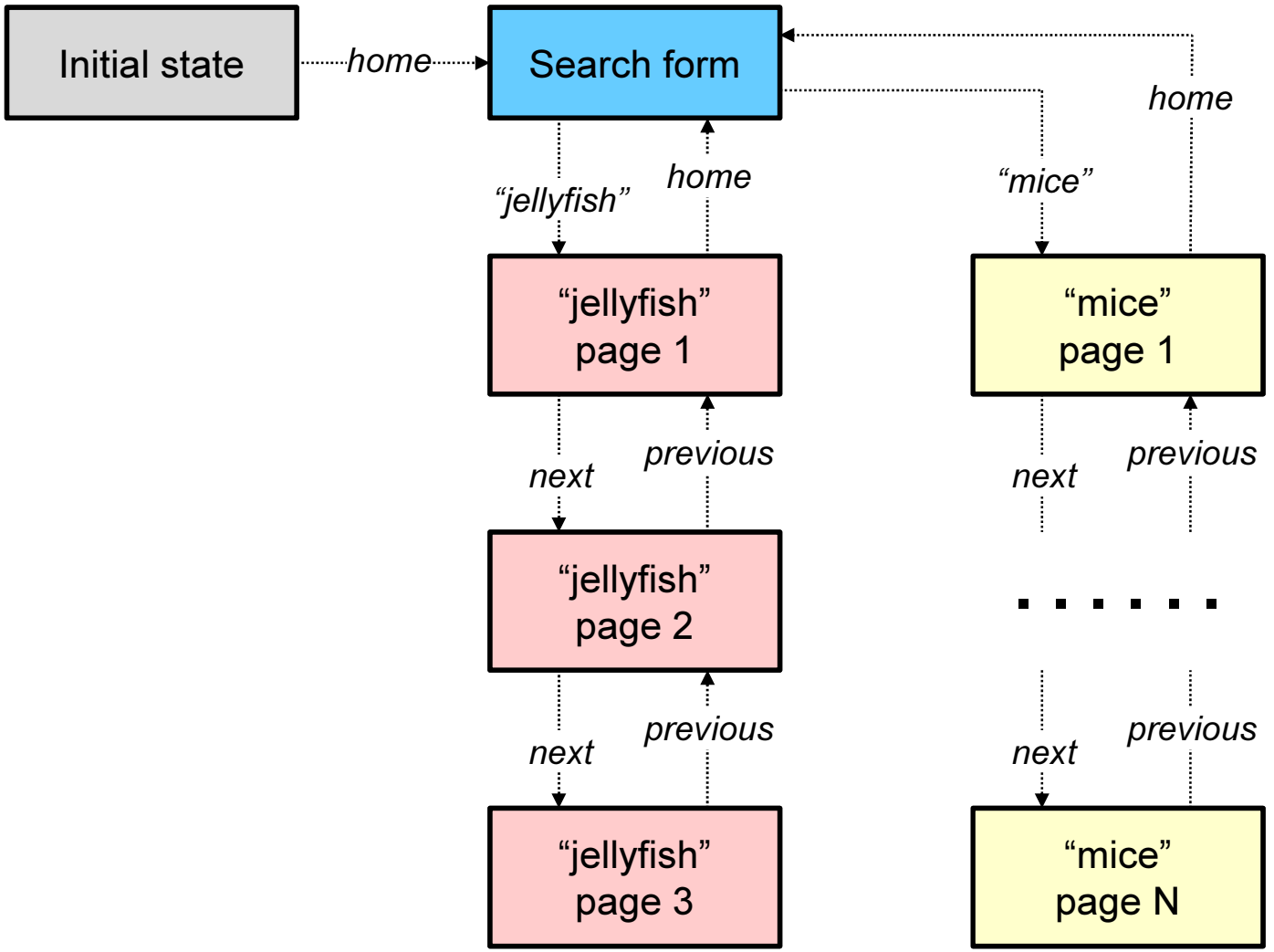
Jellyfish EIGHT
www.**jellyfish**magazine.org/

‹ Goooooooooogle ›
Previous 1 2 3 4 5 6 7 8 9 10 Next

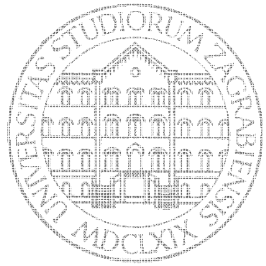
Step 5
`http://www.google.com/previous`

Statelessness

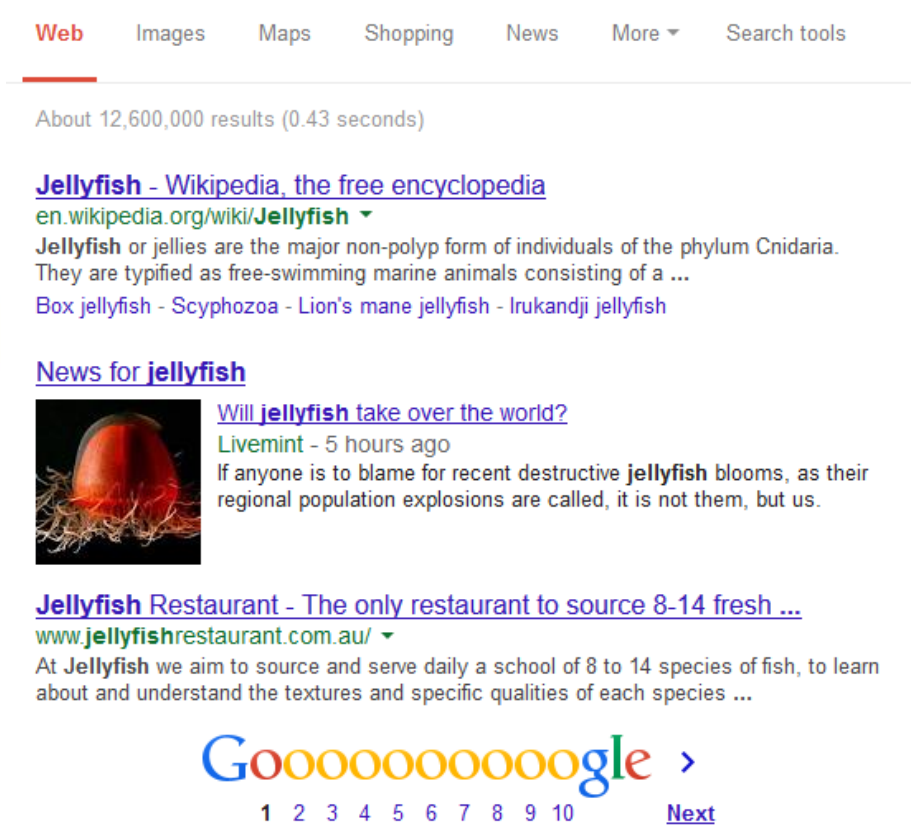
- Stateful service



Statelessness

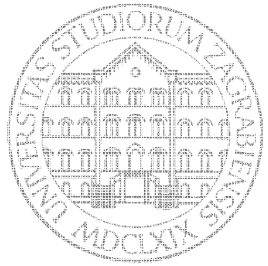


Step 1
`http://www.google.com/`



Step 2
`http://www.google.com/search?q=jellyfish`

Statelessness



Step 3
`http://www.google.com/search?q=jellyfish&start=10`

Web Images Maps Shopping More Search tools

Page 2 of about 12,300,000 results (0.20 seconds)

Images for **jellyfish** - Report images

Jellyfish Pictures
www.**jellyfish**pictures.co.uk/
Jellyfish are a BAFTA award winning visual effects studio based in Soho London specialising in computer graphics animation.

Jellyfish - Enchanted Learning Software
www.enchantedlearning.com/subjects/.../**jellyfish**/**Jellyfish**coloring.shtml
Printable chart and fact sheet show the life cycle of the **jellyfish**. From EnchantedLearning.com.

‹ Goooooooooogle ›
Previous 1 2 3 4 5 6 7 8 9 10 Next

Web Images Maps Shopping More Search tools

Page 3 of about 12,300,000 results (0.19 seconds)

They're Taking Over! by Tim Flannery | The New York Review of Books
www.nybooks.com/articles/archives/2013/.../**jellyfish**-theyre-taking-over/
Sep 26, 2013 - A moon **jellyfish** and cross **jellyfish** floating in a remote channel near Vancouver Island, British Columbia; photograph by David Hall from ...

Meduzot (2007) - IMDb
www.imdb.com/title/tt0807721/
Meduzot (the Hebrew word for **Jellyfish**) tells the story of three very different Israeli women living in Tel Aviv whose intersecting stories weave an unlikely portrait ...

Jellyfish EIGHT
www.**jellyfish**magazine.org/

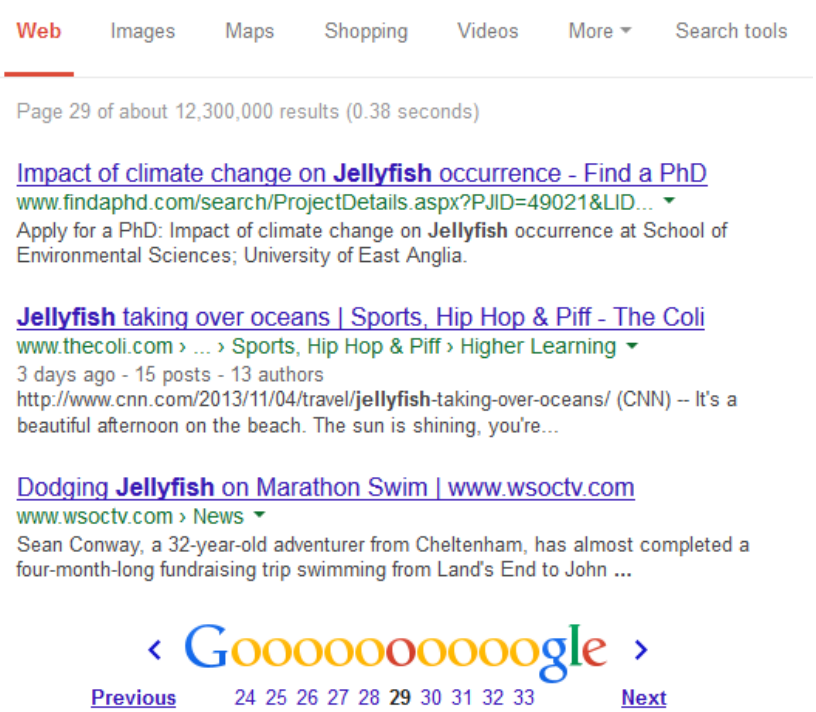
‹ Goooooooooogle ›
Previous 1 2 3 4 5 6 7 8 9 10 Next

Step 4
`http://www.google.com/search?q=jellyfish&start=20`

Statelessness

Any step

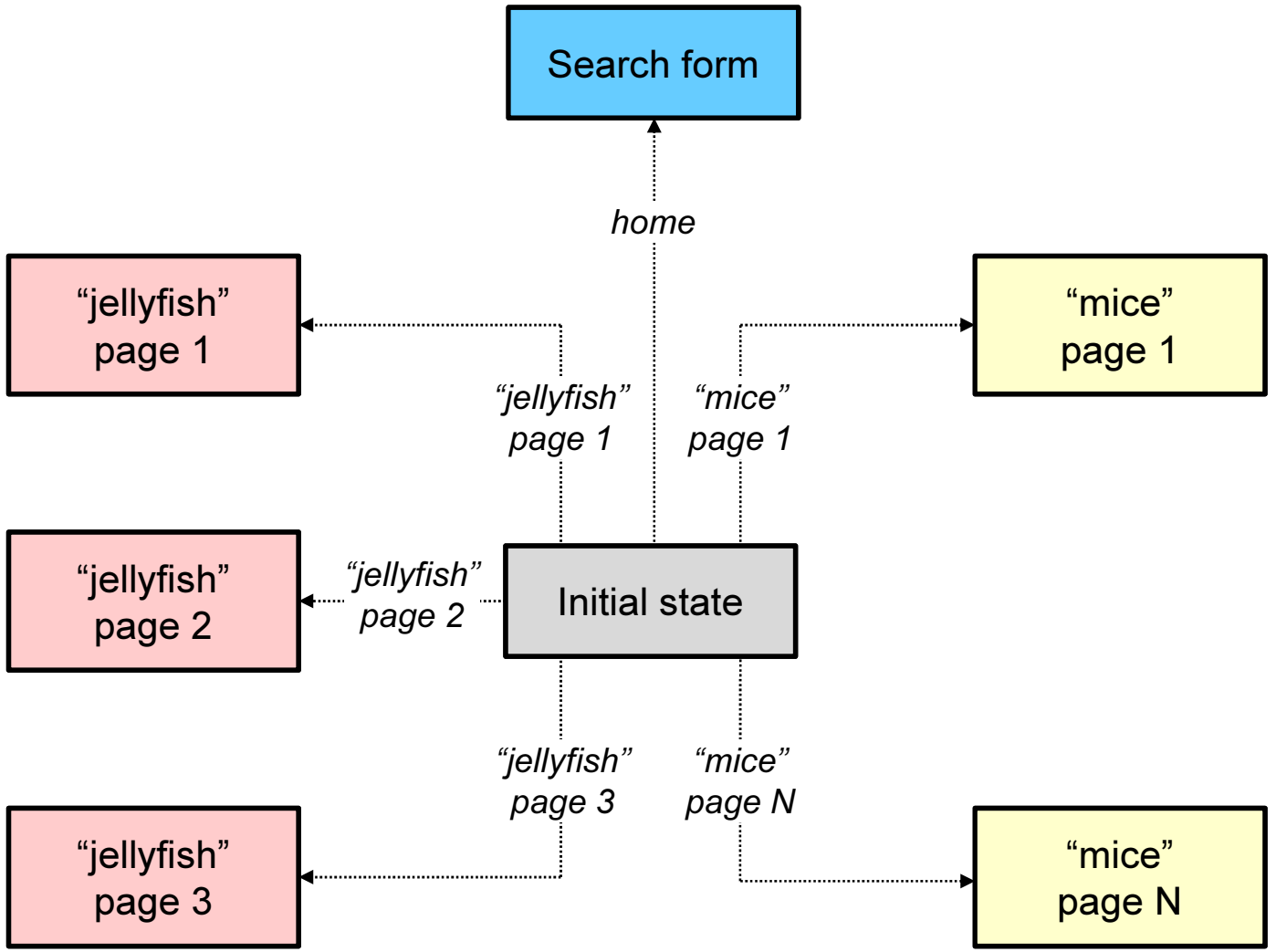
`http://www.google.com/search?q=jellyfish&start=280`



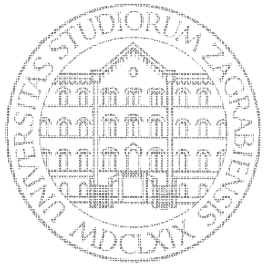
Direct access to any resource at any step no matter what resources were you used before

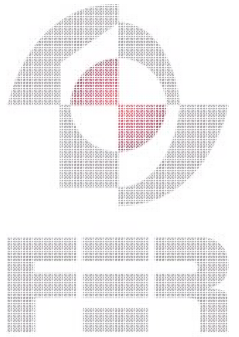
Statelessness

- Stateless service



Statelessness

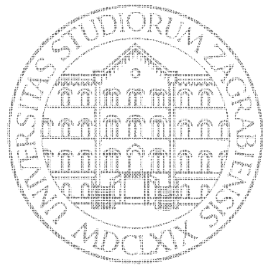


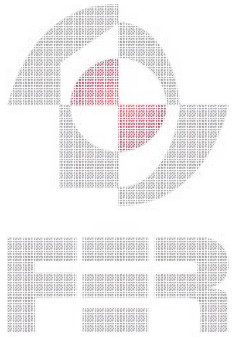


- Every HTTP request happens in complete isolation
  - When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request
  - The server never relies on information from previous requests
  - If information from previous request was important, the client would have sent it again in new request

The client should not have to coax the server into a certain state to make it receptive to a certain request

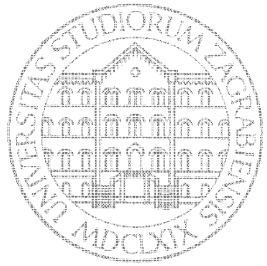
Statelessness

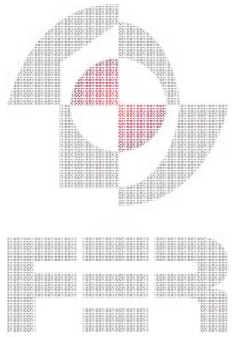




- By applying statelessness constraint
  - Session state is kept entirely on the client (***application state***)
  - Server maintains only ***resource state***
  - **Scalability** is improved due to not having to allocate resources for storing application state
    - Server does not have to manage resource usage across requests
  - **Visibility** is improved since a monitoring system does not have to look beyond a single request
  - **Reliability** is improved due to easier recoverability from partial failures
- Statelessness constraint has following tradeoffs
  - Reduced network performance
    - Larger requests
  - Reduced server control over application state consistency

Statelessness





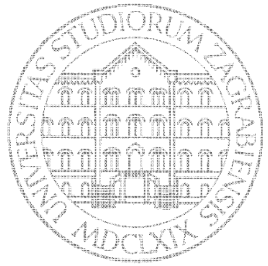
- Why statelessness matters?

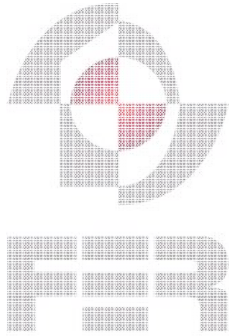
- Benefits for servers
  - Distribution of a stateless application across load-balanced servers
    - Since no two requests depend on each other, they can be handled by two different servers that never coordinate with each other
    - Scaling up is as simple as plugging more servers into the load balancer
  - Caching
    - A decision whether or not to cache the result of an HTTP request can be made just by looking at that one request
    - There's no nagging uncertainty that state from a previous request might affect the cacheability of this one

- Benefits for clients
  - A URI that works when a user is hours deep into an HTTP session will work the same way as the first URI sent in a new session

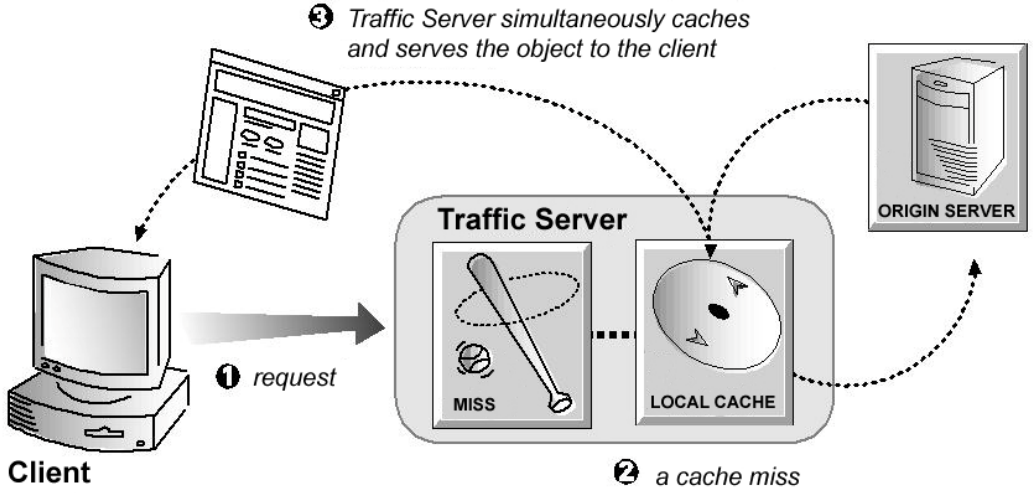
    *compare this with accessing linked list vs. indexed arrays

REST Constraints

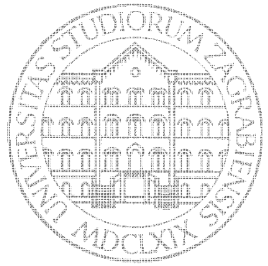


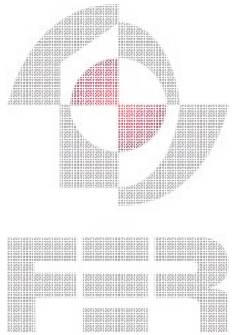


- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

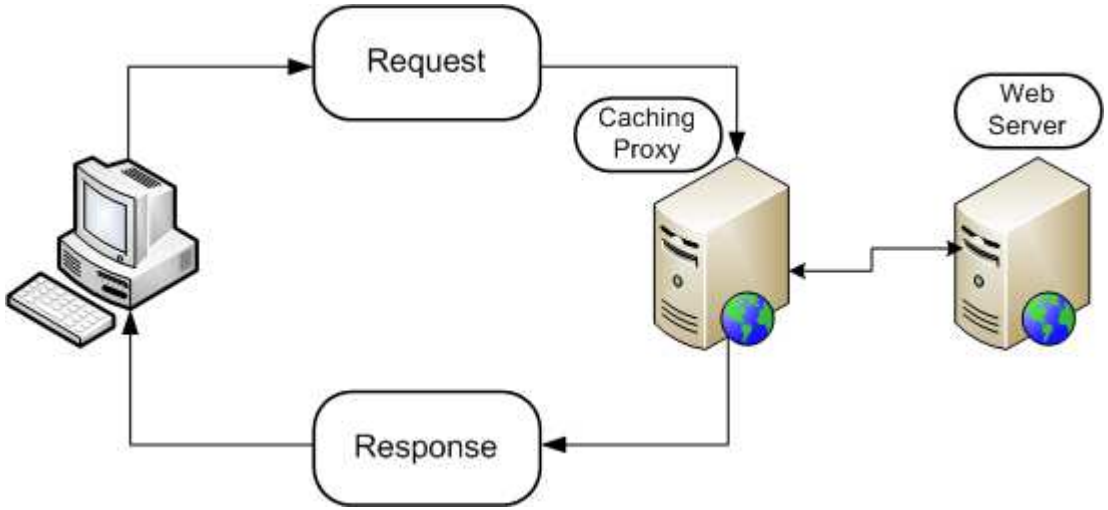


Cache



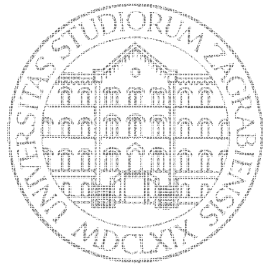


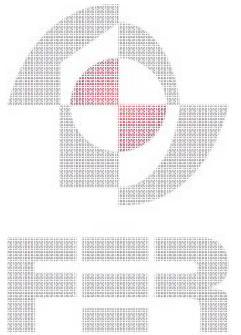
- Web caches
  - HTTP devices or software that automatically keep copies of popular resources



- When a HTTP request arrives at a cache, if a local “cached” copy is available, the resource is served from the local storage instead of from the origin server

Cache

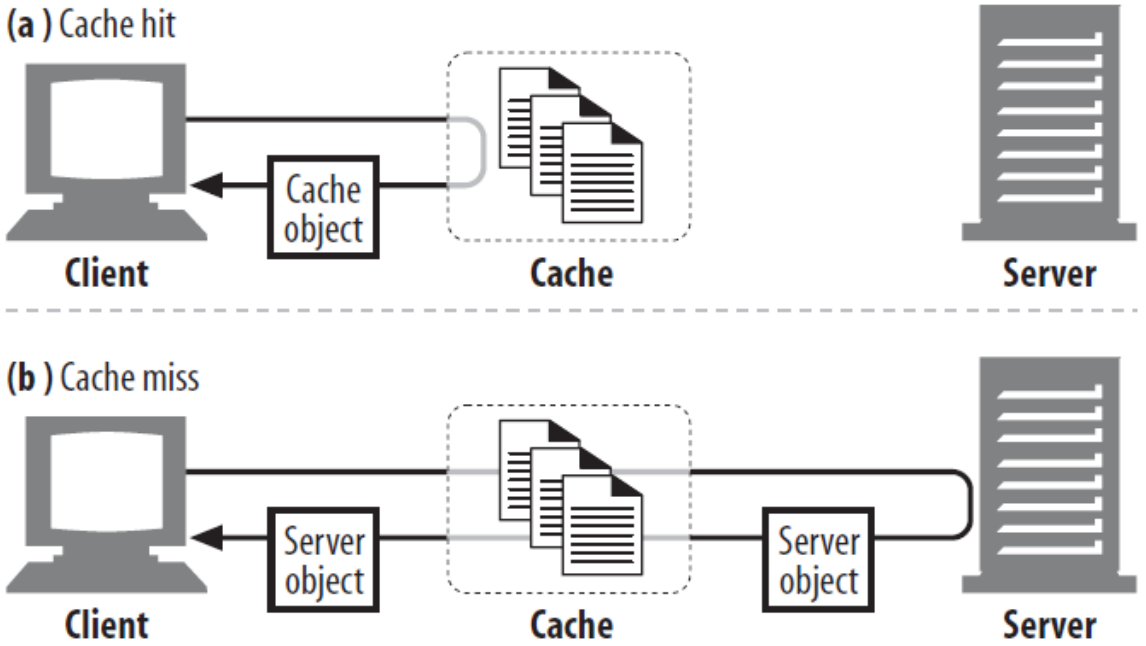




- Why web caches matter?
  - *Reduces redundant data transfers*
  - Saving money in network charges
  - *Reduces network bottlenecks*
  - Pages load faster without more bandwidth
  - *Reduces demand on origin servers*
  - Servers reply faster and avoid overload
  - *Reduce distance delays*
  - Pages load slower from farther away

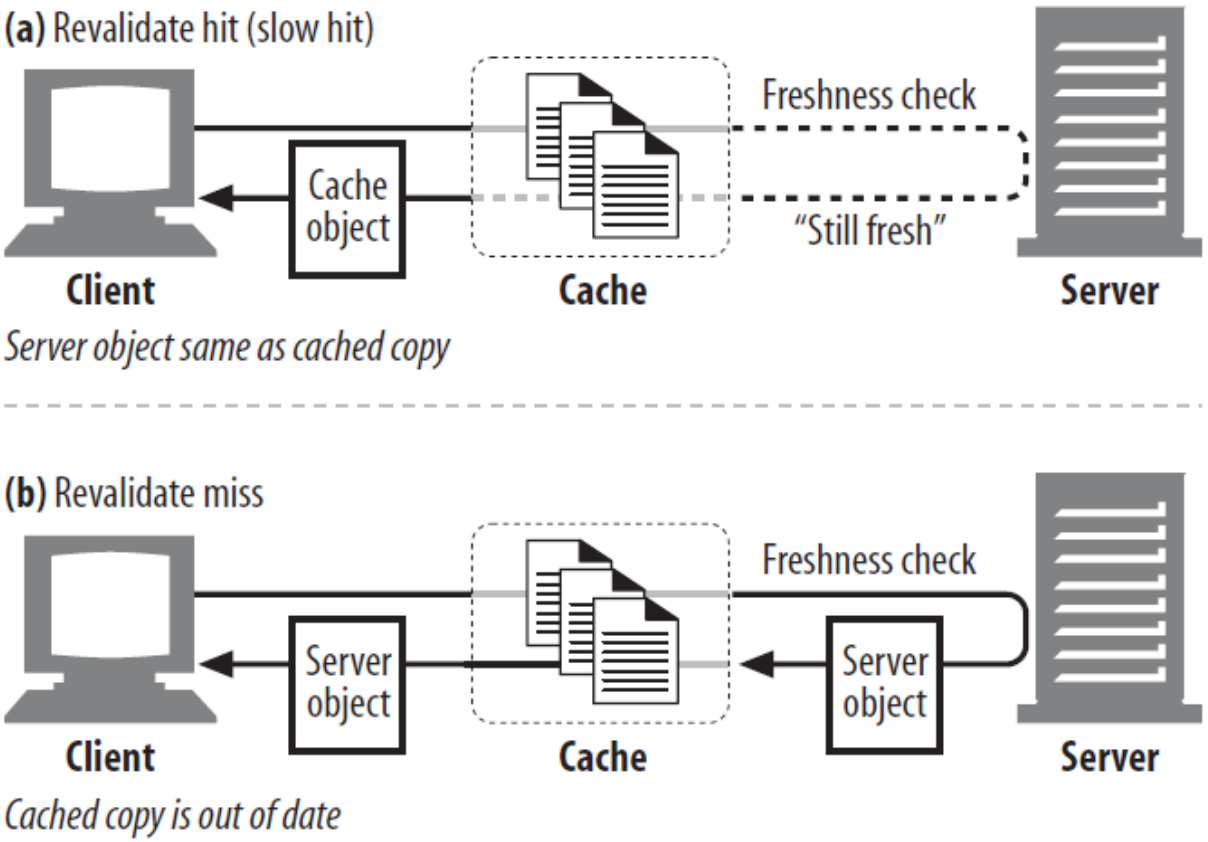
Cache

- Cache miss
  - First request for a given resource
  - Resource is retrieved from the origin server, forwarded to client, but also stored in a local cache
- Cache hit
  - Subsequent requests for the same resource
  - Resource is delivered from a local cache, instead of origin server

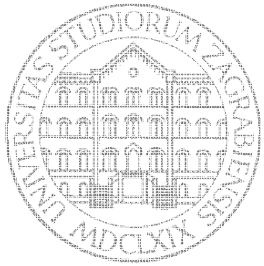


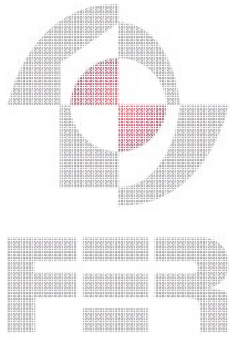
Cache

- Cache revalidation
  - Checking with the server if cached copies are still up-to-date
  - HTTP defines special requests that can quickly check if content is still fresh, without fetching the entire object from the server



Resource Expiration





- HTTP lets an origin server attach an “expiration date” to each resource

`Expires`

Response header to specify an absolute expiration date. If the expiration date is in the past, the document is no longer fresh.

`Cache-Control: max-age`

Response header to define the maximum age of the resource – the maximum legal elapsed time (in seconds) from when a resource representation is first generated to when it can no longer be considered fresh enough to serve from cache

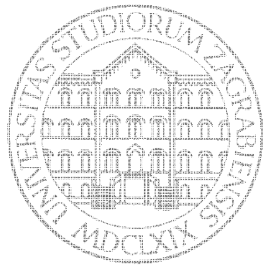
```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Expires: Fri, 05 Jul 2002, 05:00:00 GMT

Independence Day sale at Joe's Hardware
Come shop with us today!
```

(a) Expires header

```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Cache-Control: max-age=484200

Independence Day sale at Joe's Hardware
Come shop with us today!
```

(b) Cache-Control: max-age header

Resource Expiration



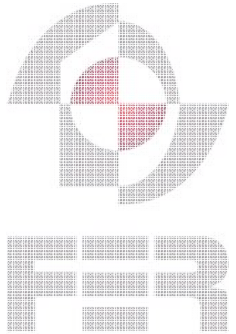
- Another option is to attach a “version identifier” to the resource

`ETag`

Entity tags are arbitrary labels (quoted strings) attached to the resource

- Serial number
- Version name
- Checksum or other fingerprint of the resource content

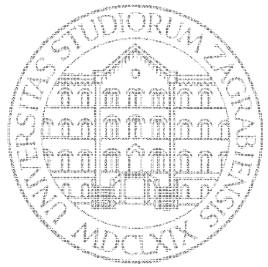
When the publisher makes a resource change, he changes the document’s entity tag to represent this new version.

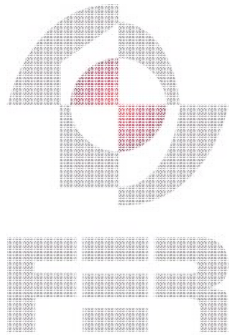


```
HTTP/1.0 200 OK
Date: Wed, 03 Jul 2002, 19:18:23 GMT
ETag: "v2.6"
Expires: Fri, 05 Jul 2002, 05:00:00 GMT
```

Response

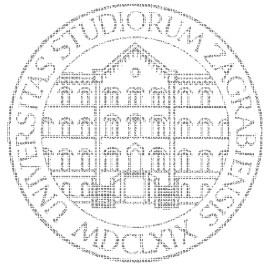
Resource Expiration

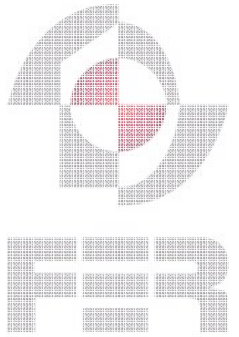




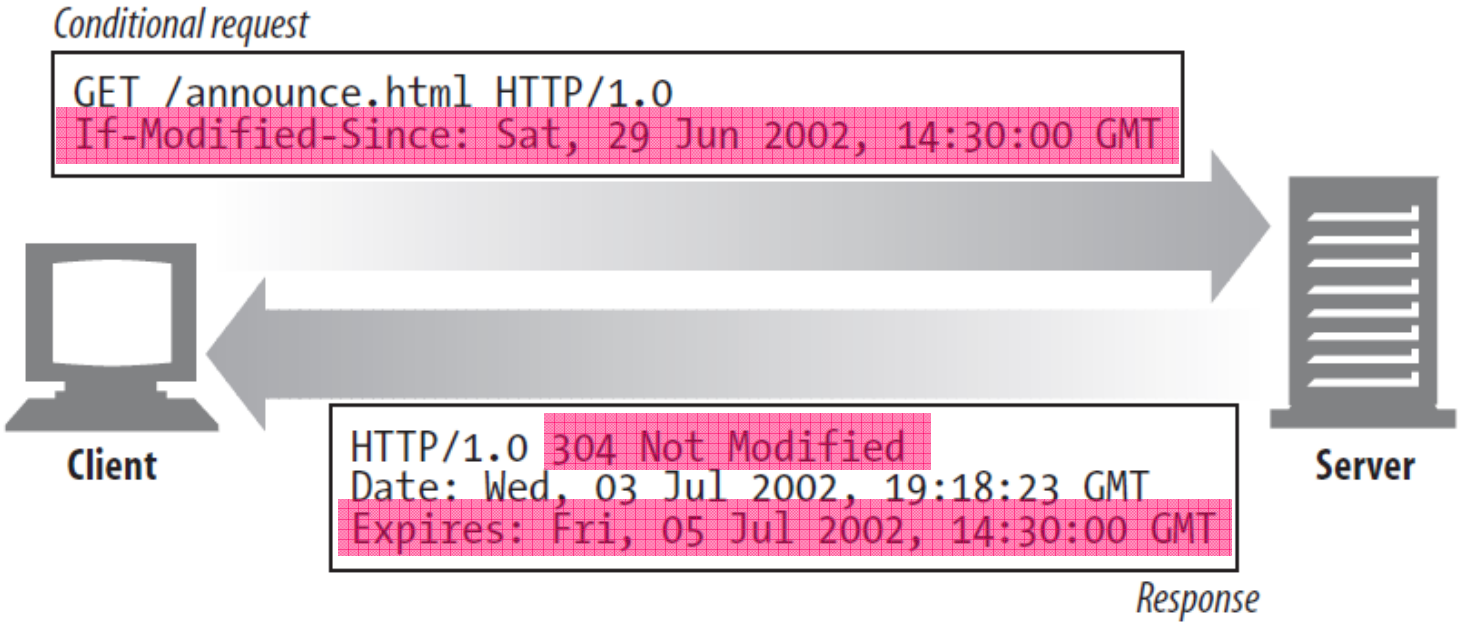
- Just because a cached resource has expired doesn't mean it is actually different from what's living on the origin server
- It just means that it's time to check
- This is called "cache revalidation," meaning the cache needs to ask the origin server whether the resource has changed:
  - If revalidation shows the content has changed, the cache gets a new copy of the resource representation, stores it in place of the old data, and sends the resource representation to the client
  - If revalidation shows the content has not changed, the cache only gets new headers, including a new expiration date, updates the headers in the cache, and sends previously cached resource representation to the client

Cache Revalidation

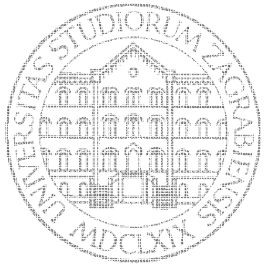


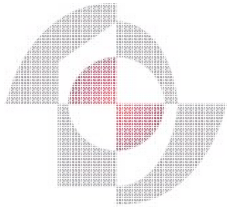


- Conditional GET request
  - Date revalidation
  - Revalidate hit
    - If the server object isn't modified, the server responds with a small HTTP `304 Not Modified` response, with no content

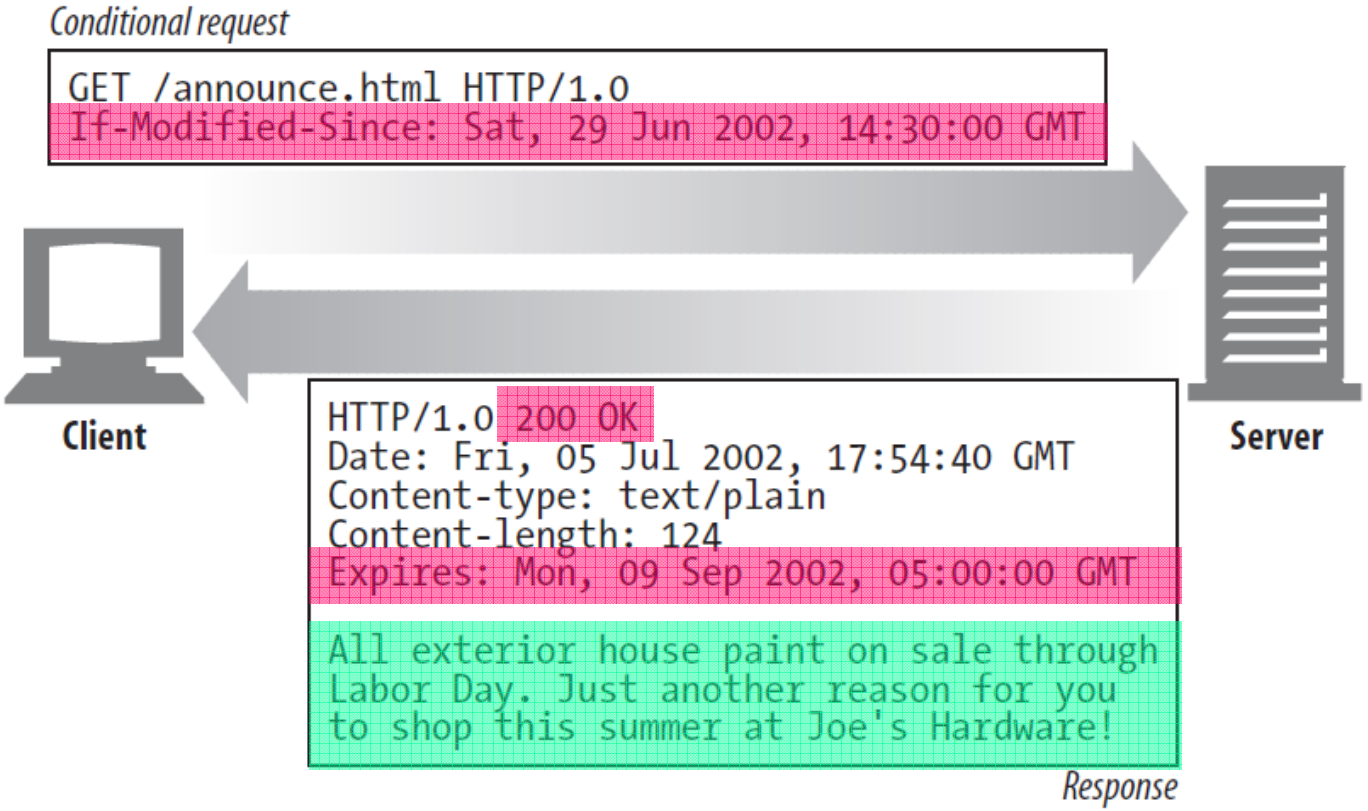


Cache Revalidation

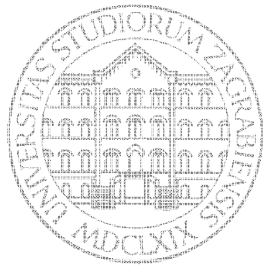


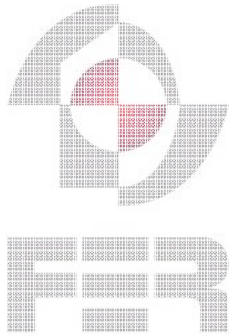


- Conditional GET request
  - Date revalidation
  - Revalidate miss
    - If the server object is different from the cached copy, the server sends the client a normal HTTP `200 OK response`, with the full content



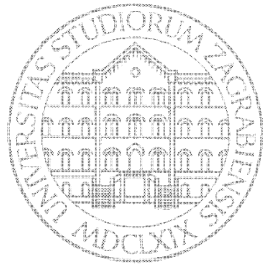
Cache Revalidation

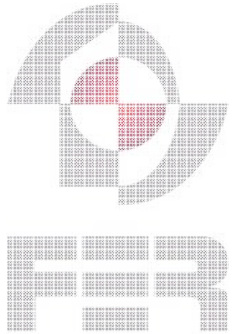




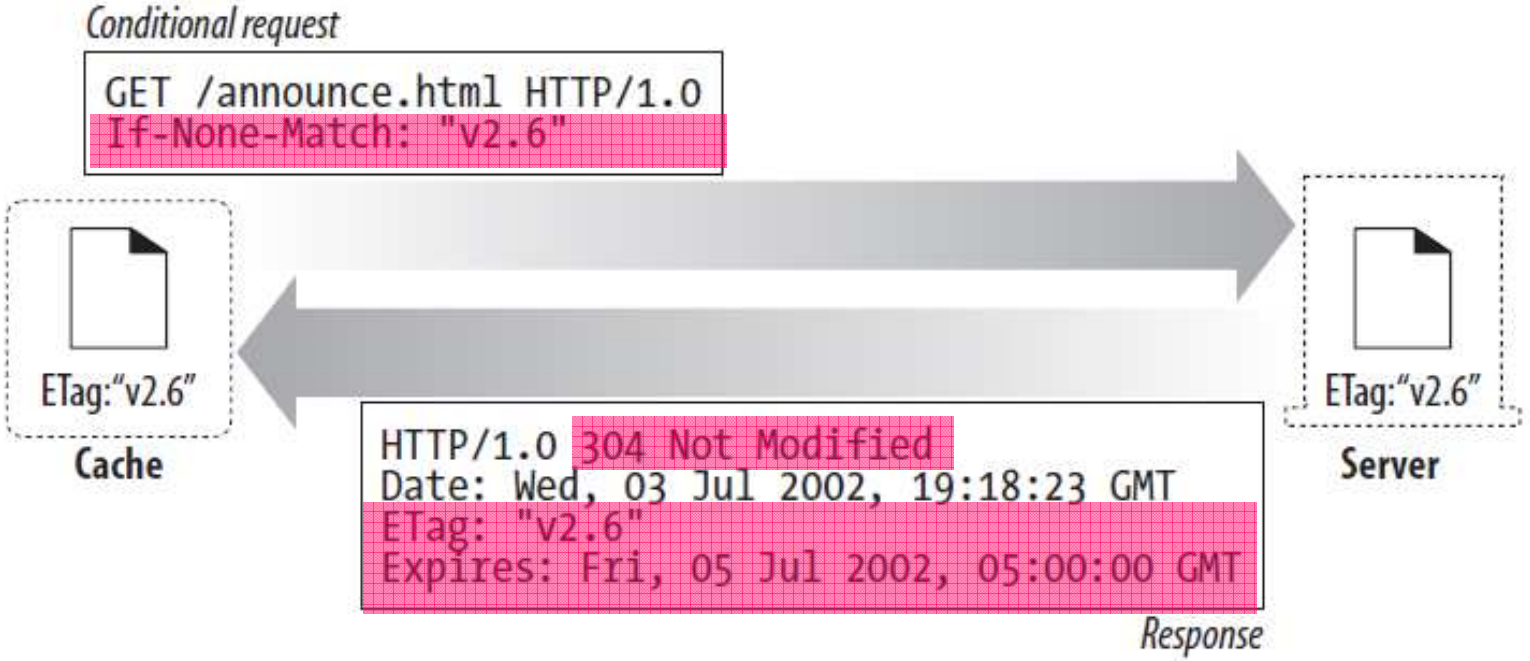
- Conditional GET request
  - Date revalidation
  - Resource deleted
    - If the server object has been deleted, the server sends back a `404 Not Found` response, and the cache deletes its copy

Cache Revalidation



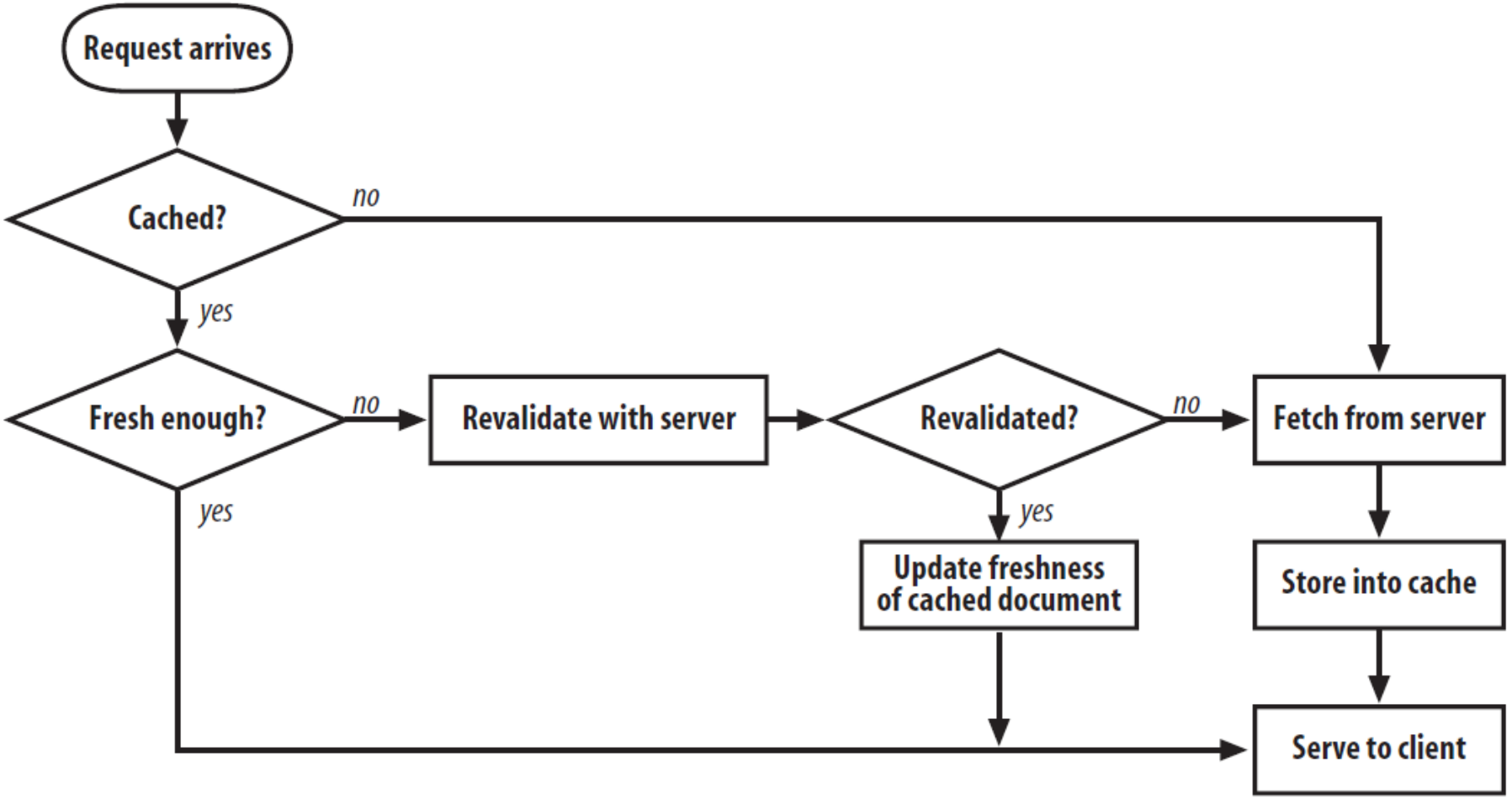


- Conditional GET request
  - Entity tag revalidation
  - Revalidate hit
    - If the server object has been deleted, the server sends back a `404 Not Found` response, and the cache deletes its copy

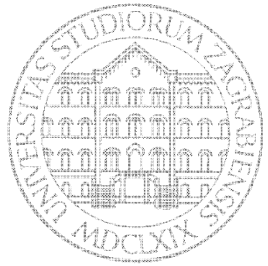


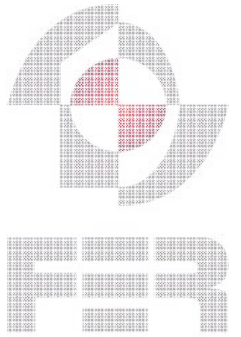
Cache Revalidation

- Cache processing flowchart



Cache





- Hit rate
  - The fraction of requests that are served from cache
  - The hit rate ranges from 0 to 1 (0% to 100%)
    - 0% means that every request was a miss (had to get the document across the network)
    - 100% means that every request was a hit (had a copy in the cache)

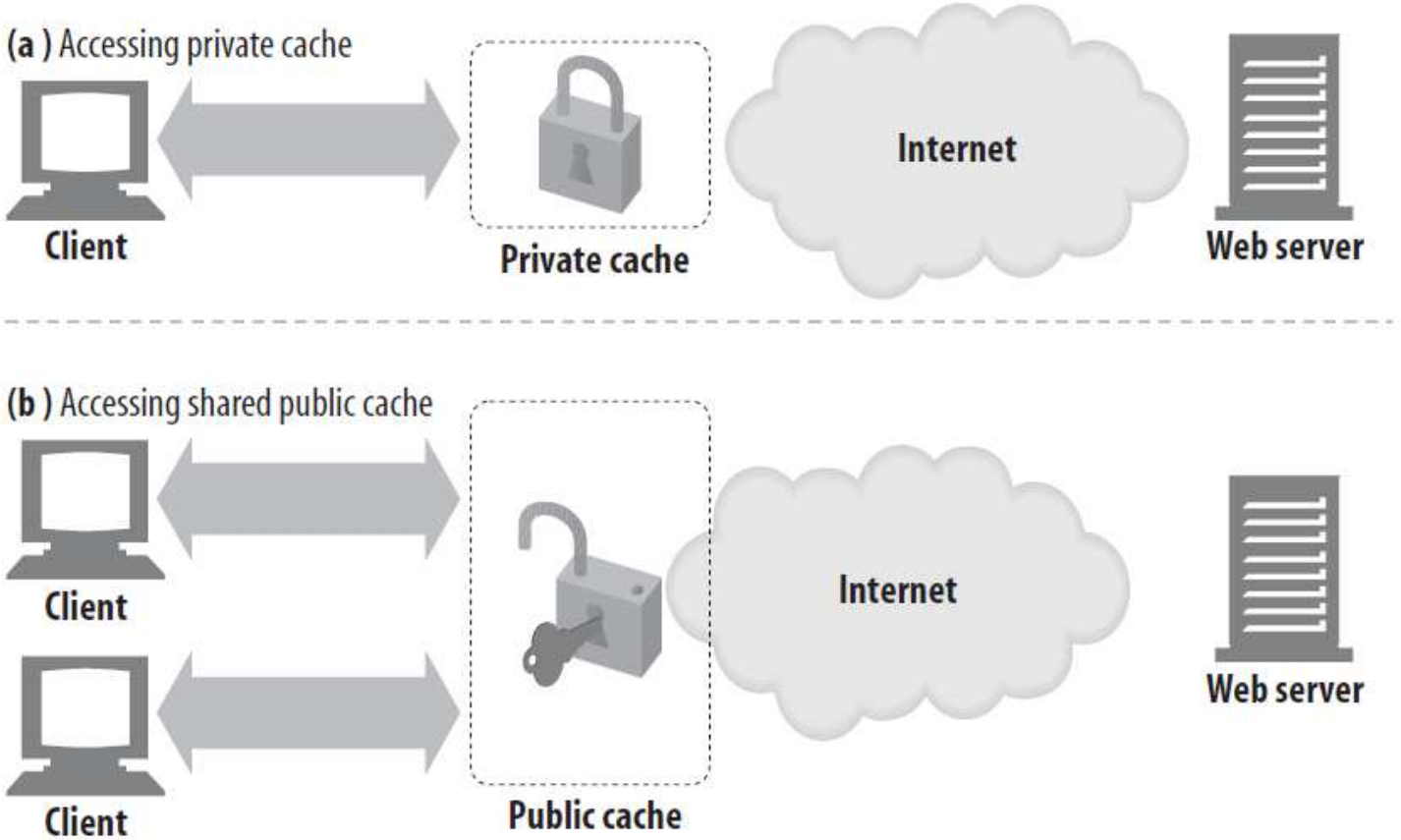
$$hit\ rate = \frac{number\ of\ cache\ hits}{total\ number\ of\ requests}$$

- Byte hit rate
  - The fraction of all bytes transferred that were served from cache
  - Resources are not all the same size
  - Large objects might be accessed less often but contribute more to overall data traffic

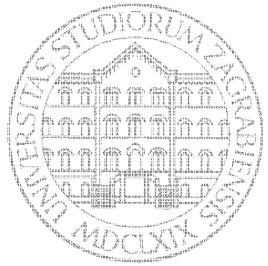
$$byte\ hit\ rate = \frac{number\ of\ bytes\ delivered\ from\ cache}{total\ size\ of\ all\ responses}$$

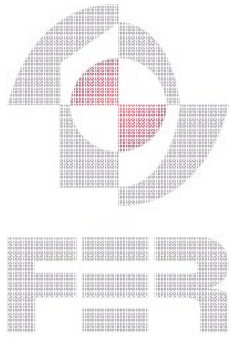
Cache Topologies

- Private caches and public caches



Cache Topologies

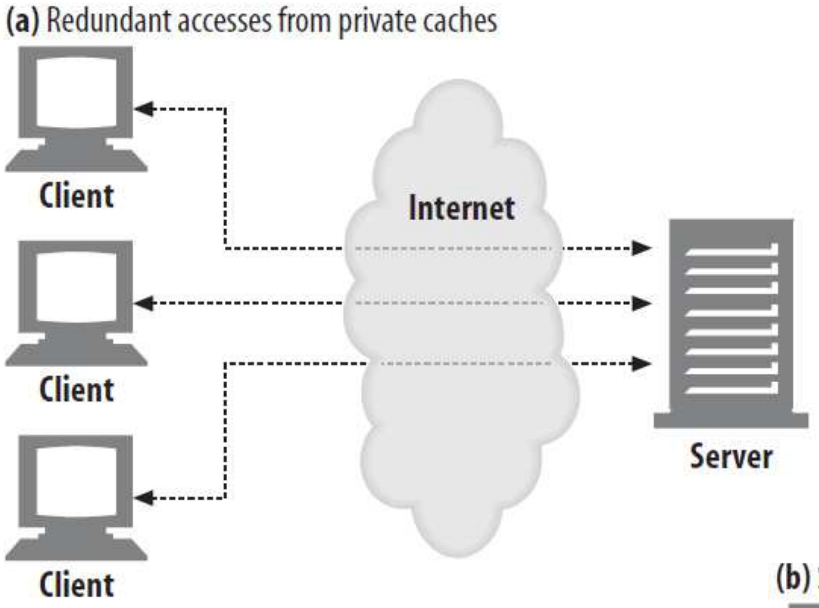




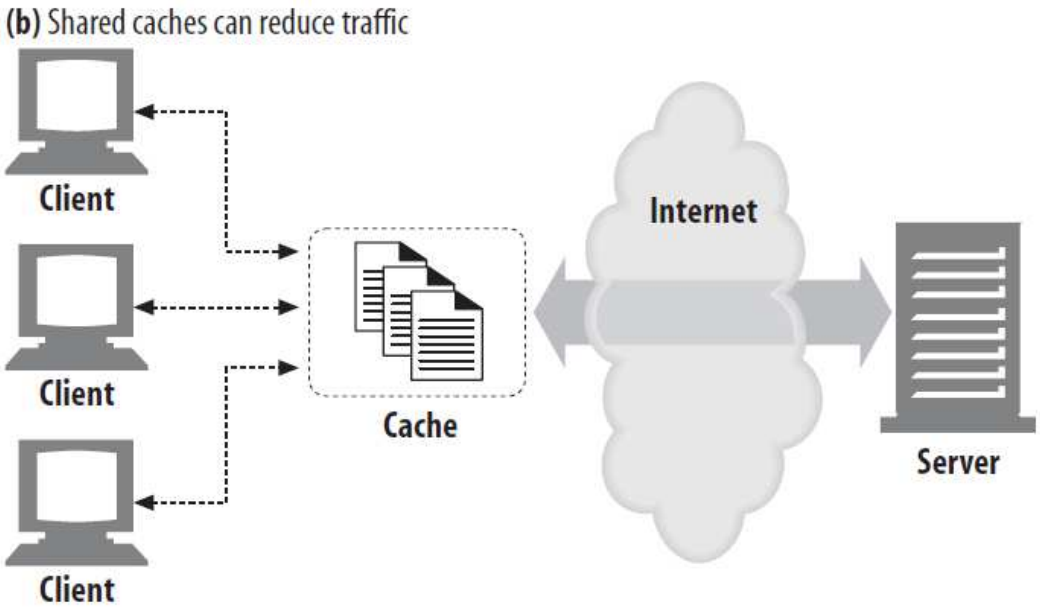
- Private cache
  - Personal cache
  - Dedicated to a single user
  - Do not need much computing power or storage space
  - Small and cheap
  - Web browsers have private caches built in
    - Most browsers cache popular documents in the disk and memory of user's personal computer

- Public cache
  - Shared between user community
  - Caching proxy servers (proxy caches)
  - More expensive than private cache
  - Because a public cache receives accesses from multiple users, it has more opportunity to eliminate redundant traffic

Cache Topologies

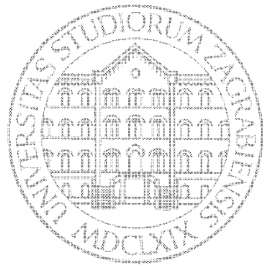


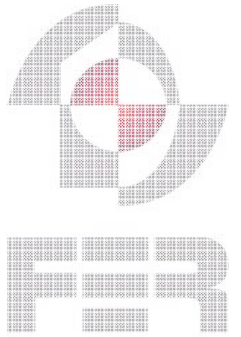
Each client accesses the same object (not yet in the private cache). Each private cache fetches the same object, crossing the network multiple times.



The public cache needs to fetch the popular object only once. It uses the shared copy to service all other clients, reducing network traffic.

REST Constraints

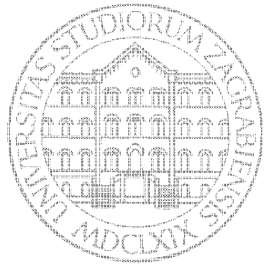


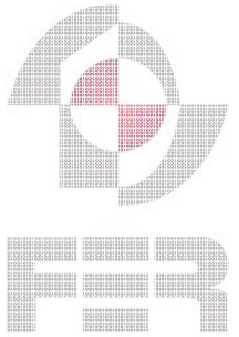


- Network constraints
  - Client-server
  - Layered system
- Content constraints
  - Addressability
  - Connectedness
- Interaction constraints
  - Uniform interface
  - Statelessness
  - Cache
  - Code on demand

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, stringData;
        TcpClient server;
        try{
            server = new TcpClient(" . . . . ", port);
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int recv = ns.Read(data, 0, data.Length);
        stringData  = Encoding.
            ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        while(true){
            input = Console.ReadLine();
            if (input == "exit") break;
                newchild.Properties["ou"].Add
                ("Auditing Department");
                newchild.CommitChanges();
                newchild.Close();
```

Code on Demand





- The only optional constraint in REST

- Allows client functionality to be extended by downloading and executing code in the form of applets or scripts
  - Java applets
  - Javascript scripts
  - Flash applications

- Using code on demand reduces visibility, which is why this constraint is optional
  - Security
    - Prevents inspecting and filtering interactions at network firewalls (CRUD level security enforcement)
    - Possibility to download a vulnerable or malicious software
  - Caching
    - The code itself is cacheable, but the results of its execution aren't