POLITECNICO DI TORINO

e-Lite

Web Architecture and Technologies

Ambient intelligence: technology and design

Fulvio Corno

Politecnico di Torino, 2014/2015

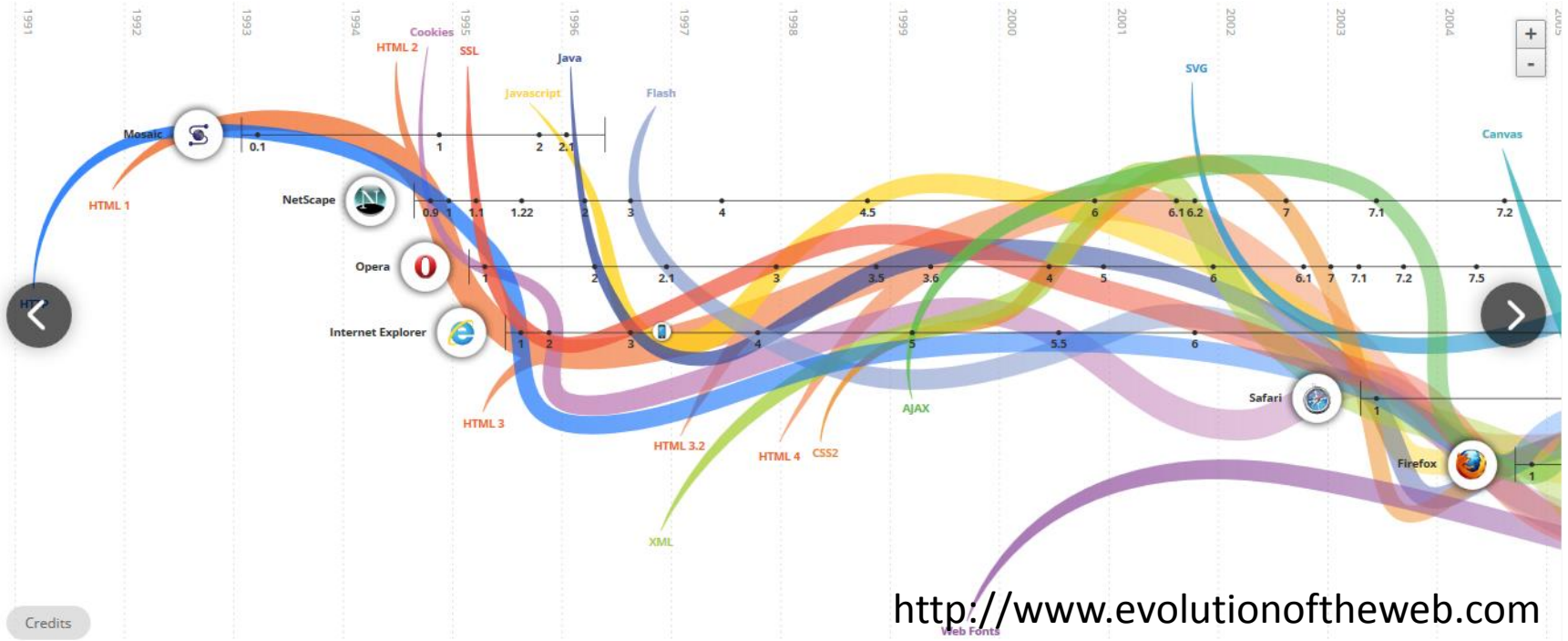
POLITECNICO DI TORINO

Goal

- Understanding Web technologies
  - Adopted for User Interfaces
  - Adopted for Distributed Application Integration
- Understanding main communication protocol (http)
- How to use REST architectures to integrate (call and/or offer) remote services

Summary

- Web Architecture Overview
- HTTP (Hypertext Transfer Protocol)
- REST (Representational State Transfer)
- JSON (JavaScript Object Notation)

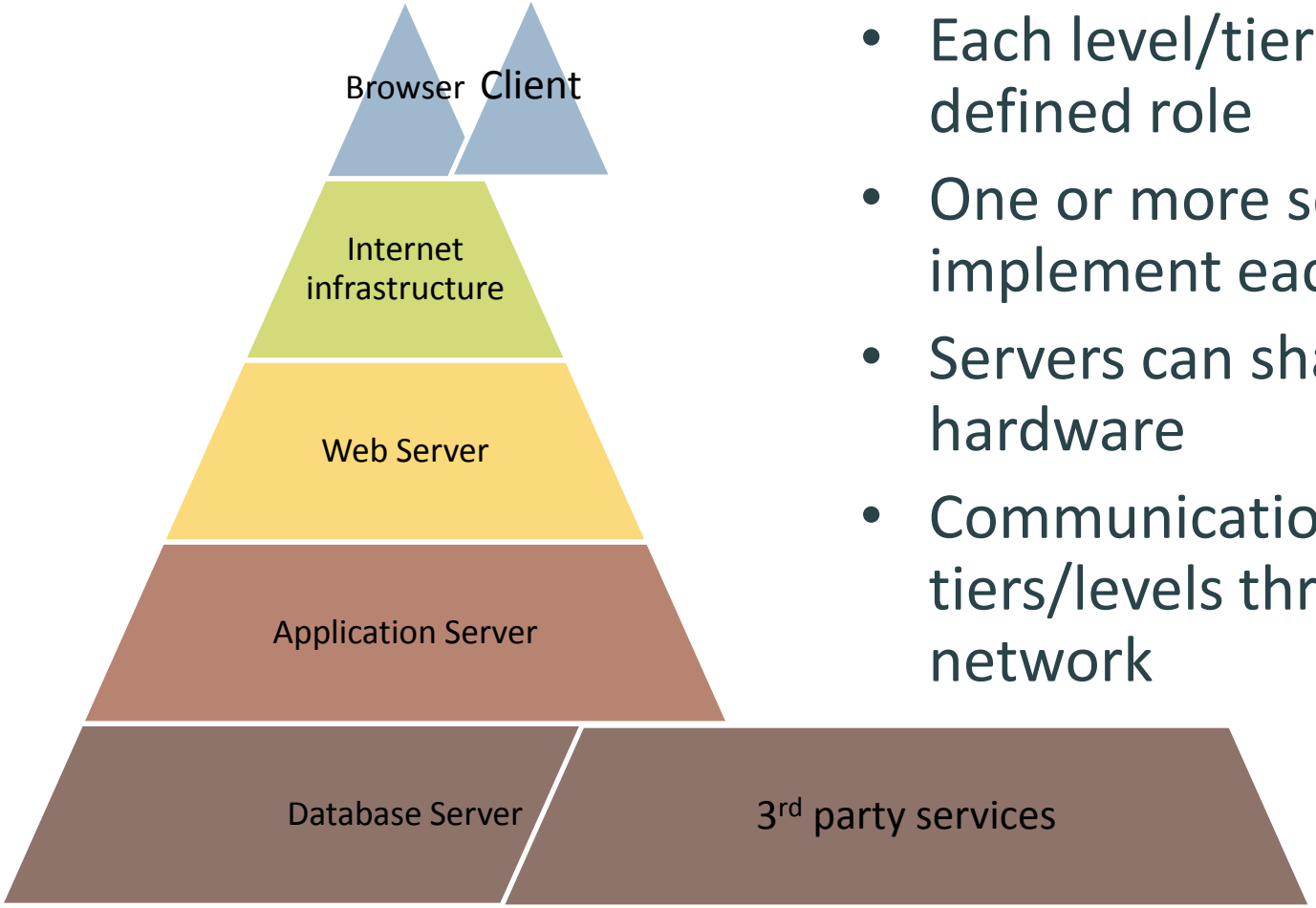


Web Architecture and Technologies

WEB ARCHITECTURE OVERVIEW

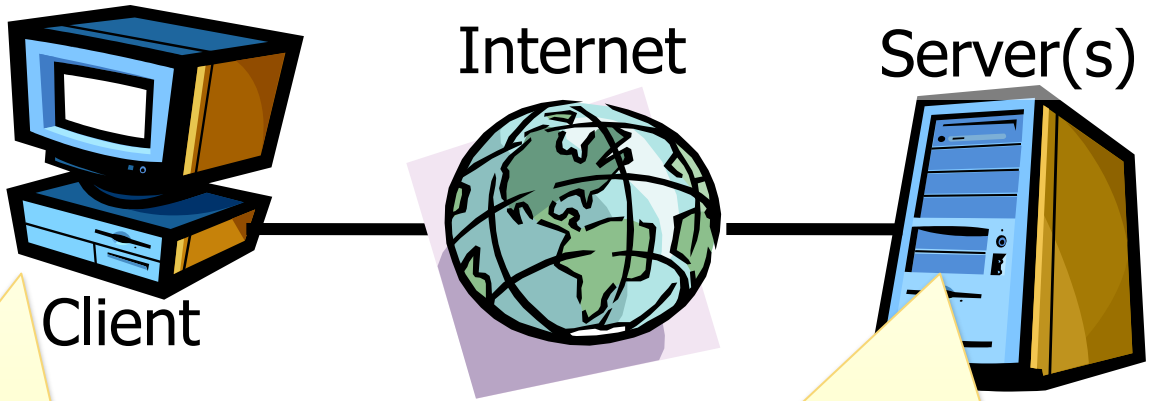
HISTORICAL EVOLUTION

N-tier (N-level) architecture



- Each level/tier has a well defined role
- One or more servers implement each tier/layer
- Servers can share hardware
- Communication between tiers/levels through the network

General Architecture



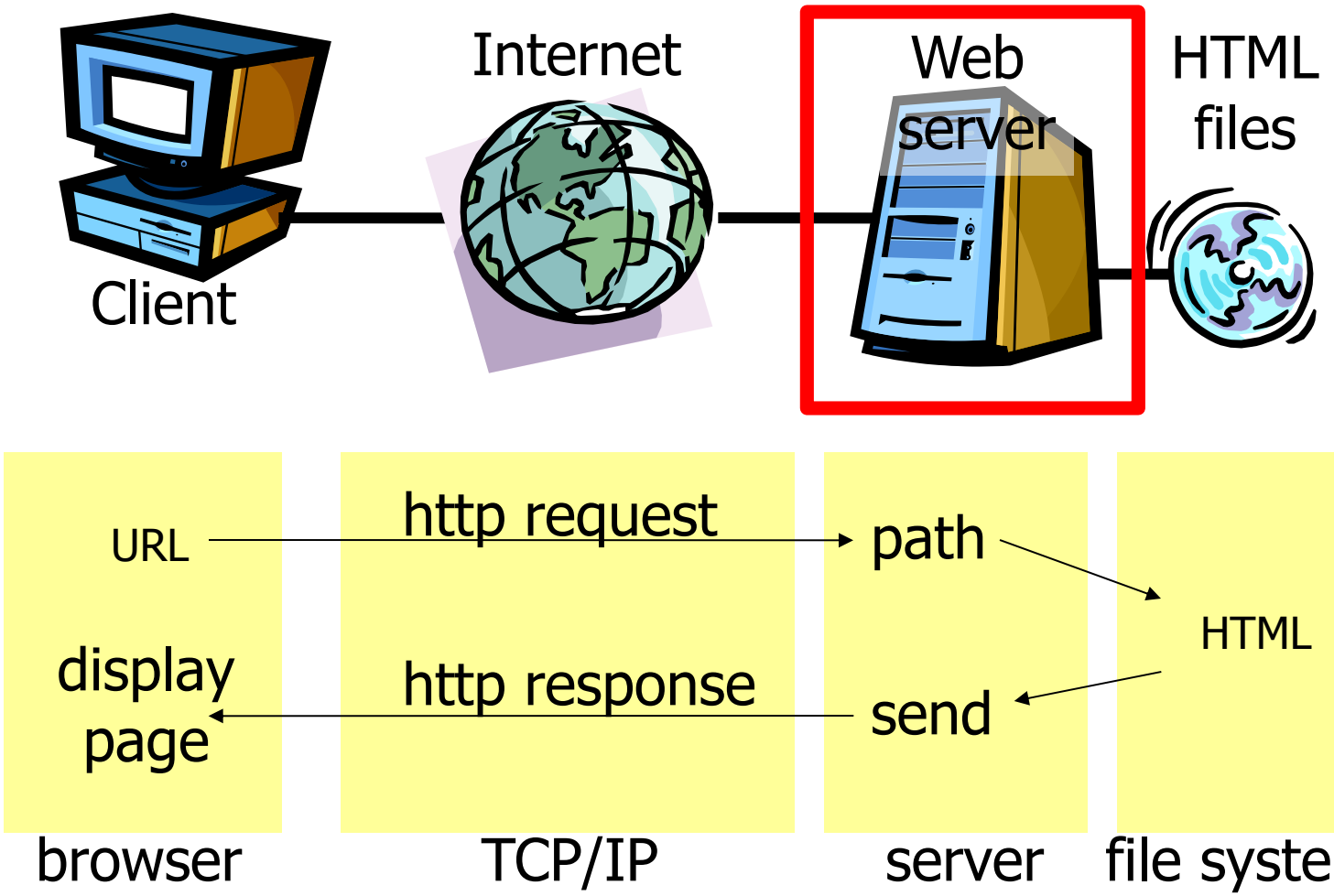
- Historically, a web browser
- But also:
  - Mobile application
  - Desktop application
  - Other server application

- Logical server:
  A process that runs on a host that relays information to a client upon the client sending it a request.
- Physical server :
  A host computer on a network that holds information (eg, Web sites) and responds to requests for information

Web server (logical)

- Manages the HTTP protocol (handles requests and provides responses)
  - Receives client requests
  - Reads static pages/contents from the filesystem
  - Activates the application server for dynamic pages/content generation (server-side)
  - Provides an file (HTML, or other) back to the client
- One HTTP connection for each request
- Multi-process, Multi-threaded or Process pool

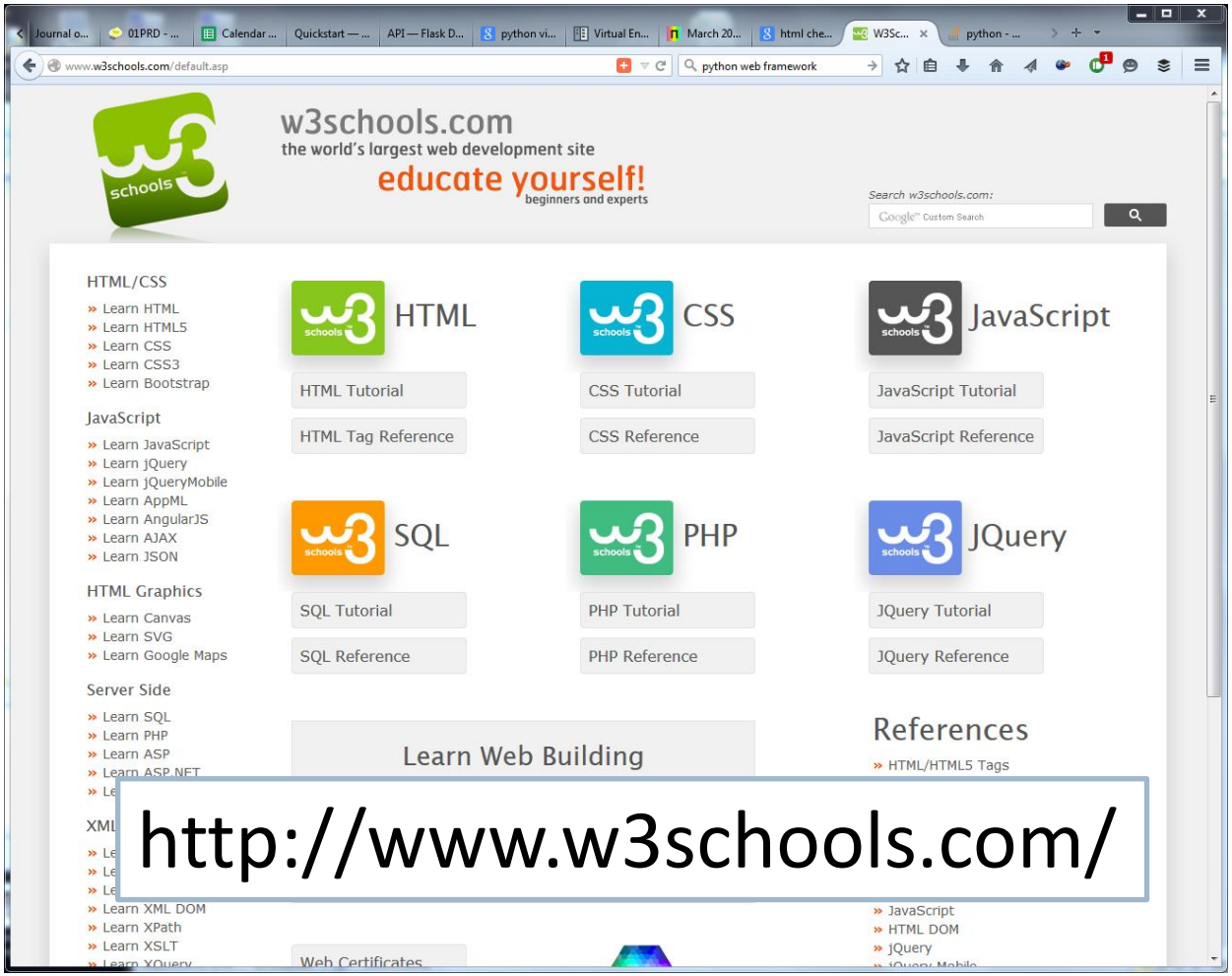
Example


Internet
Web server
HTML files
Client
URL
http request
path
HTML
display page
http response
send
browser
TCP/IP
server
file system

Adopted standards

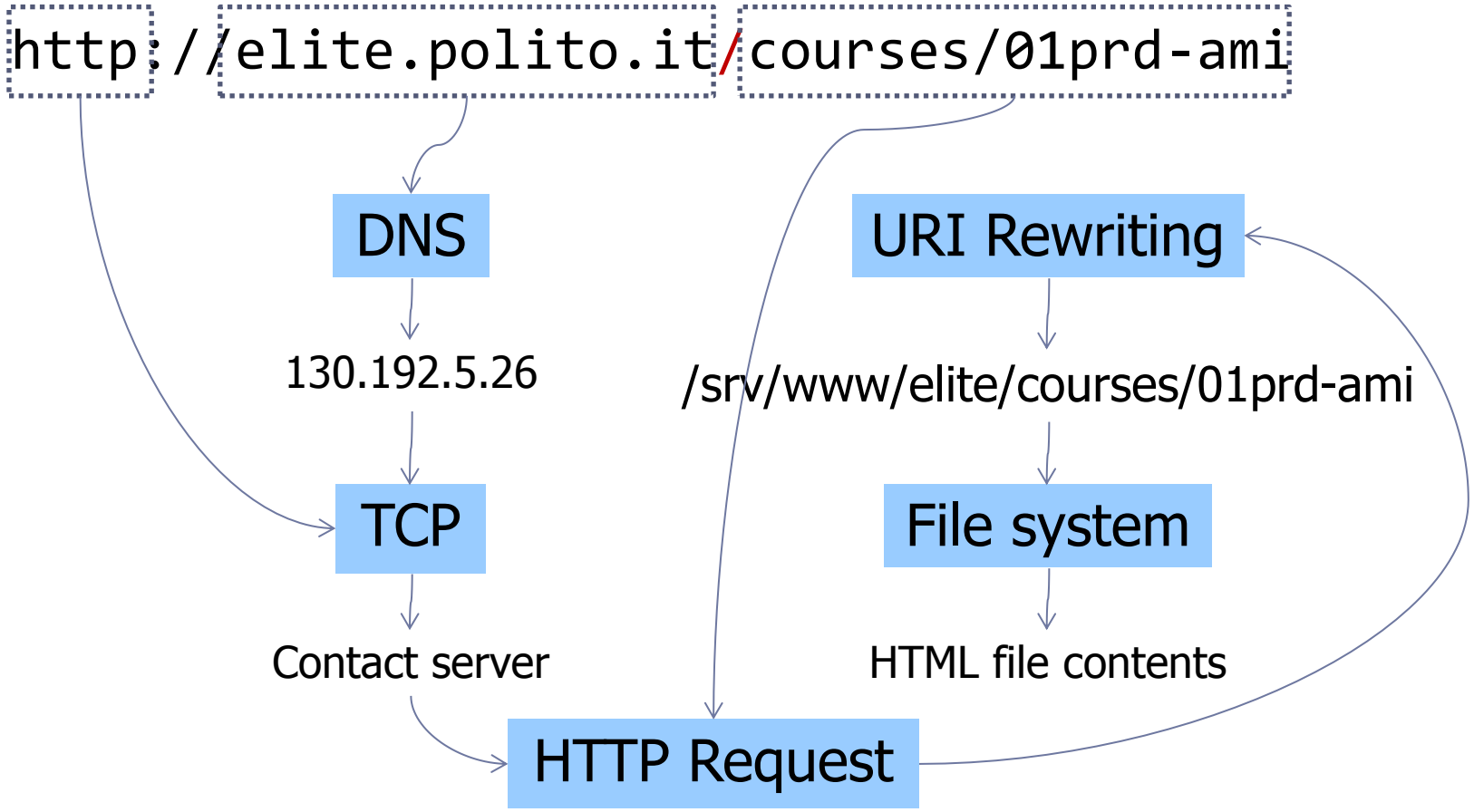
- URL (uniform resource locator) for finding web pages
- HTML (hyper text markup language) for writing web pages
- GIF (graphics interchange format), PNG (portable network graphics), JPEG, … for images
- HTTP (hyper text transfer protocol) for client-server interaction
- TCP/IP (transmission control protocol over internet protocol) for data transfer

HTML in 5 minutes



URL

RFC 2396
http://www.w3.org/Addressing/



URI Basics

- Scheme, Hostname, Query
  http://www.sadev.co.za/users/1/contact
- Scheme, Hostname, Query
  http://www.sadev.co.za?user=1&action=contact
- Scheme, Userinfo, Hostname, Port
  http://rob:pass@bbd.co.za:8044
- Scheme, Hostname, Query, Fragment
  https://bbd.co.za/index.html#about

HTTP protocol

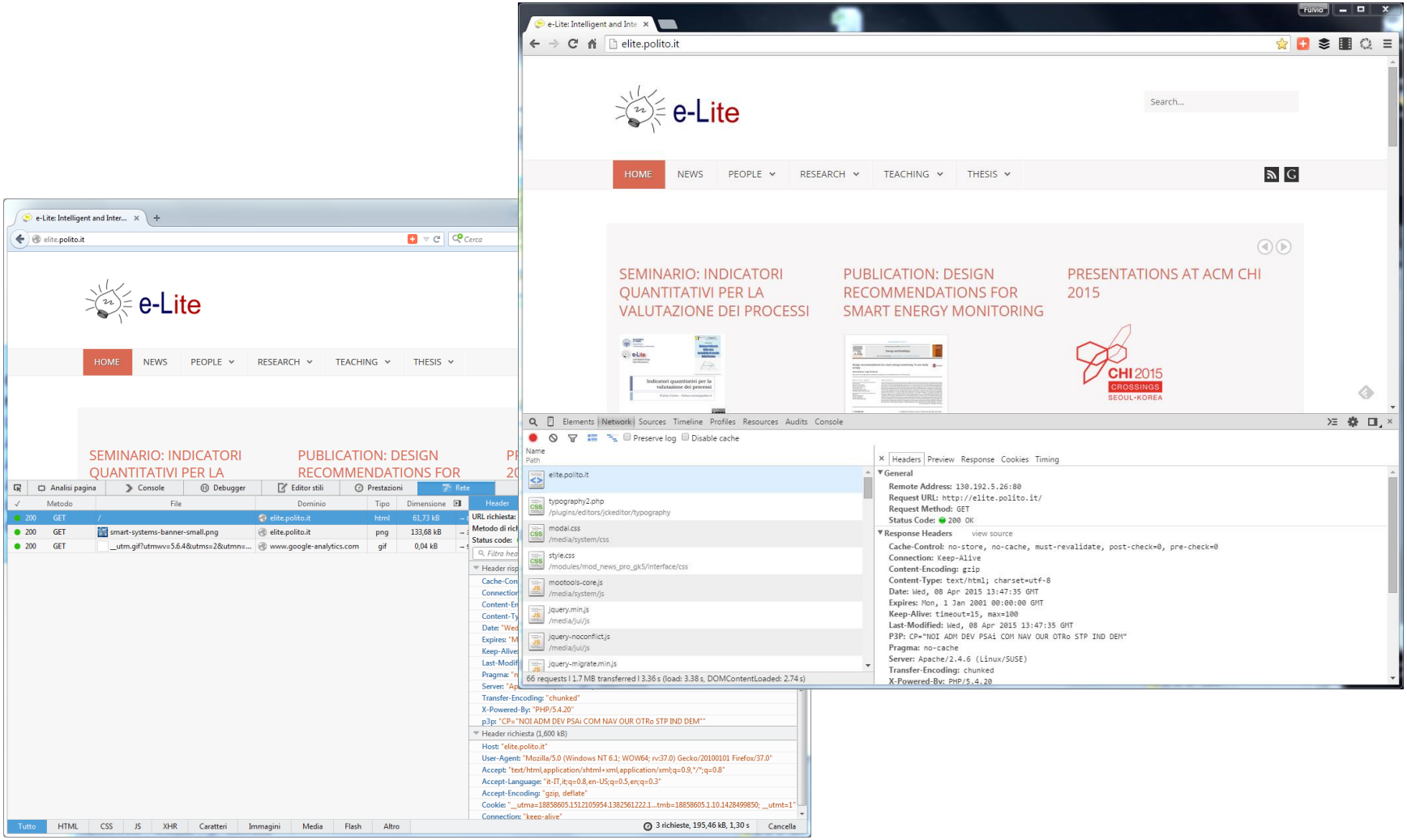
RFC 2616, RFC 2617
http://www.w3.org/Protocols

```
GET / HTTP/1.1
Host: elite.polito.it
User-Agent: Mozilla/5.0
Accept: text/html,appl
Accept-Language: it-I
Accept-Encoding: gzi
Cookie: __utma=1885
Connection: keep-a
```

```
HTTP/1.0 200 OK
Cache-Control: no-store, no-cache, must-revalidate,
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 08 Apr 2015 13:36:24 GMT
Expires: Mon, 1 Jan 2001 00:00:00 GMT
Keep-Alive: timeout=15, max=100
Last-Modified: Wed, 08 Apr 2015 13:36:24 GMT
Pragma: no-cache
Server: Apache/2.4.6 (Linux/SUSE)
Transfer-Encoding: chunked
X-Powered-By: PHP/5.4.20
p3p: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM«

<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-GB">
<head>
. . . . . .
```

Browser developer tools



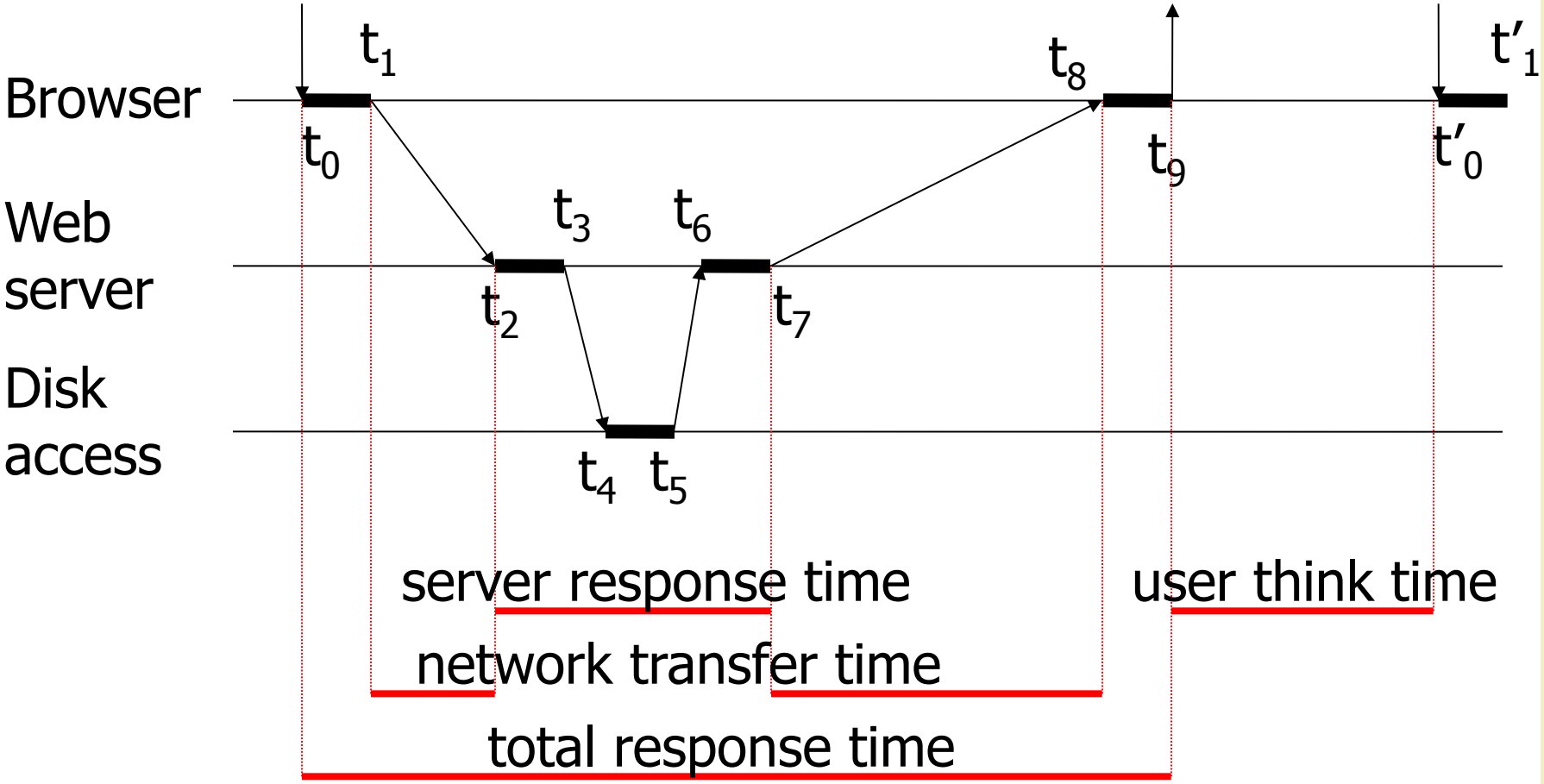
Performance measures

- **Latency**: time required for providing a 0 byte http page. Includes the server activation time, the request decoding time, the file access time, the transmission time and the time for closing the connection.
  - Unit of measure: http/s or s/http
- **Throughput**: maximum speed at which infinite-sized pages can be sent.
  - Unit of measure: Bytes (Mbytes)/s
- **#Requests / s**

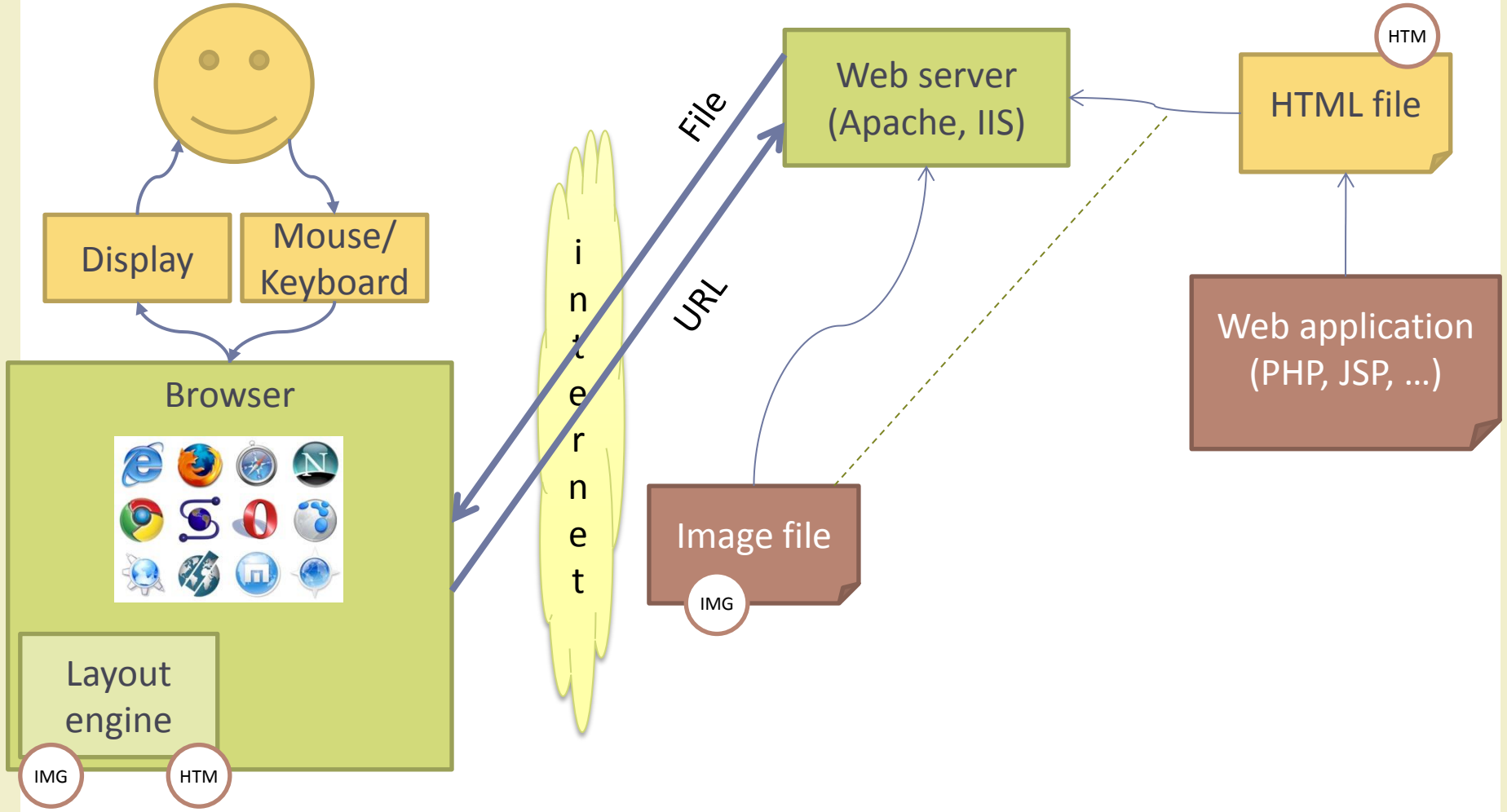
Delay time

- T = Latency + ResponseBytes / Throughput
- This equation is valid if:
  - The other architecture elements (I/O subsystem, network, ...) are not overloaded
  - The web server has not yet reached its maximum workload
- Example:
  - Latency: 0,1s
  - ResponseBytes : 100kBytes
  - Throughput: 800kBytes/s
  - T= 0,1s+ 100KBytes / 800KBytes/s =0,225s

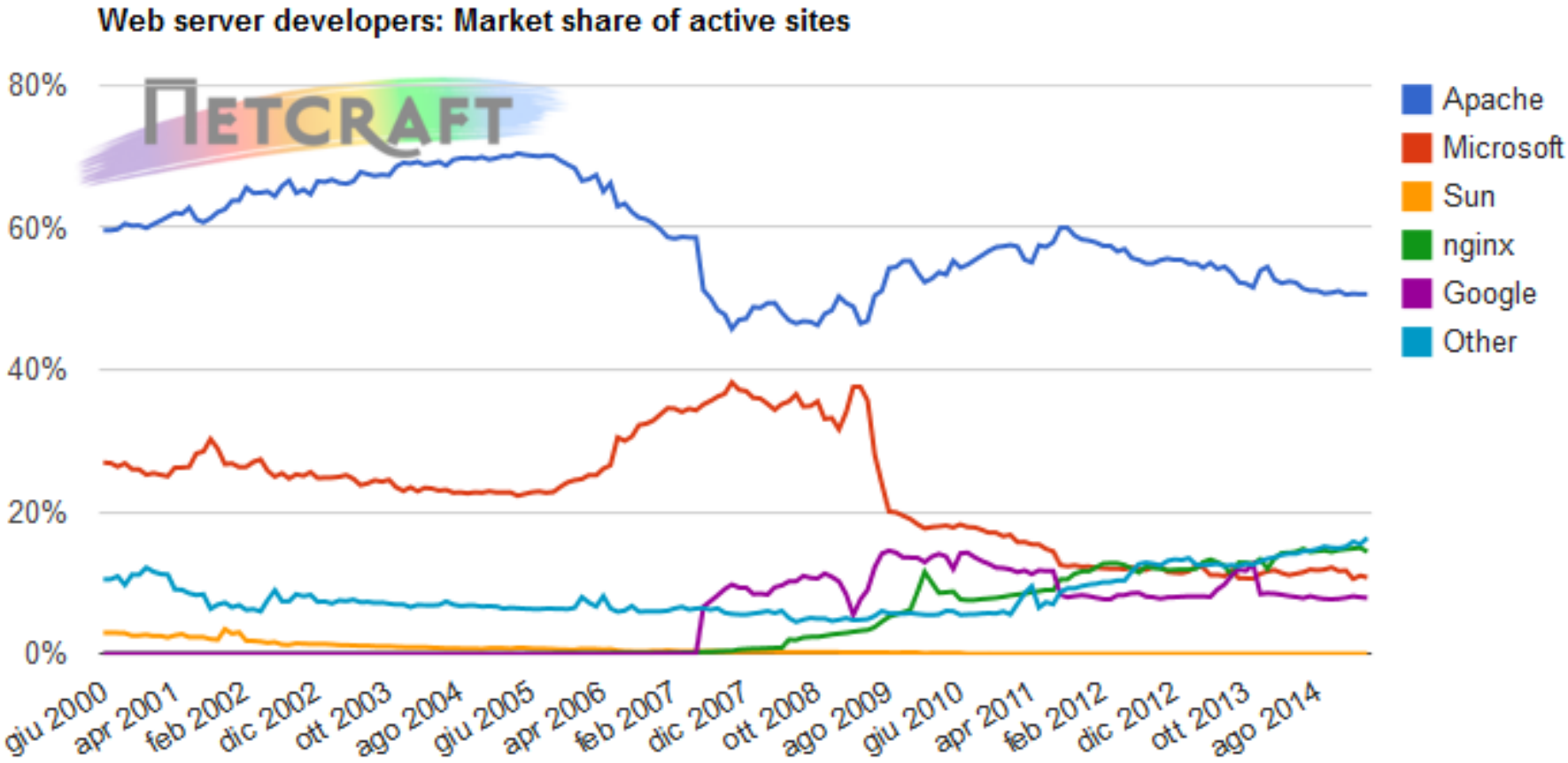
Static web transaction



General Architecture



The most adopted web servers



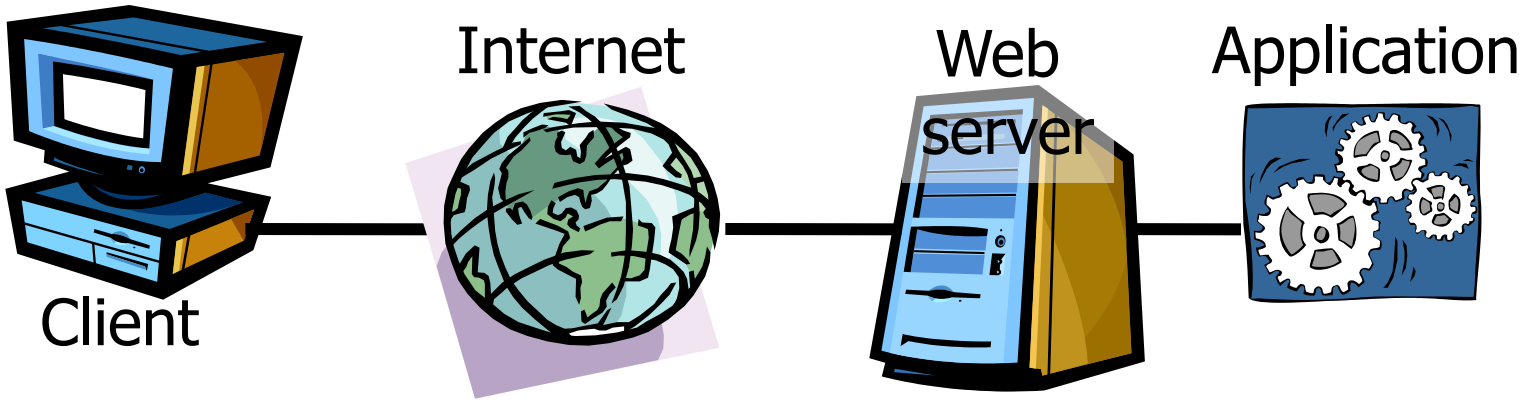
Source: http://news.netcraft.com/

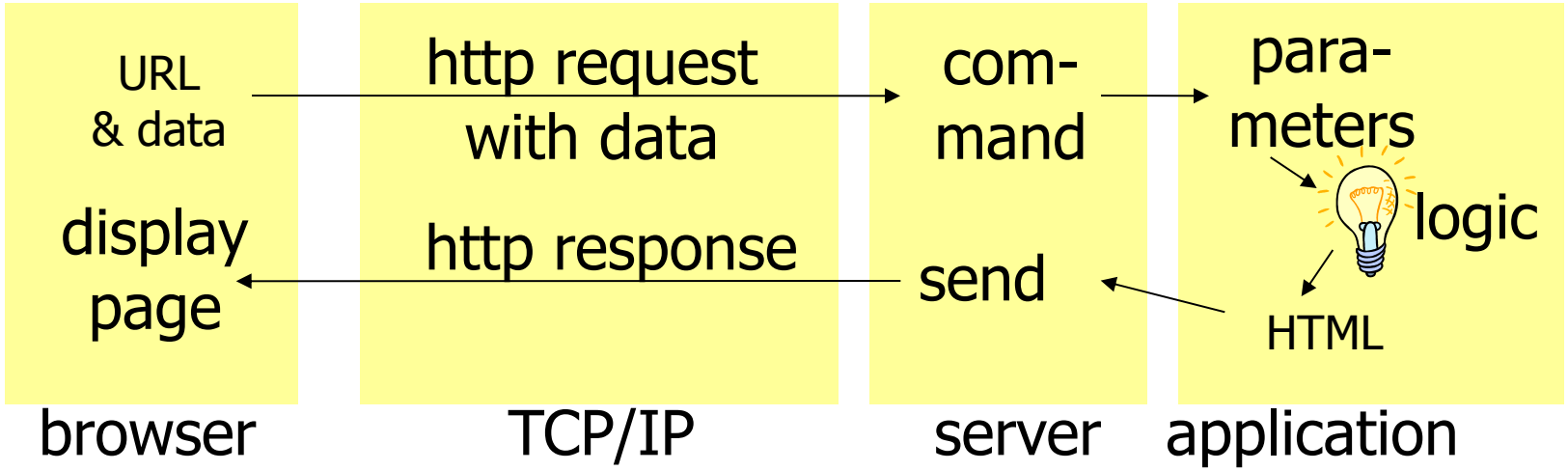
http://news.netcraft.com/archives/2015/03/19/march-2015-web-server-survey.html

Application server

- Dynamic page generation and content generation
- Manages the site business logic
- It’s the middle tier between the client browser and the data residing on a database
- Implements the session mechanisms
- Different technologies and architectures are available

Dynamic web transaction

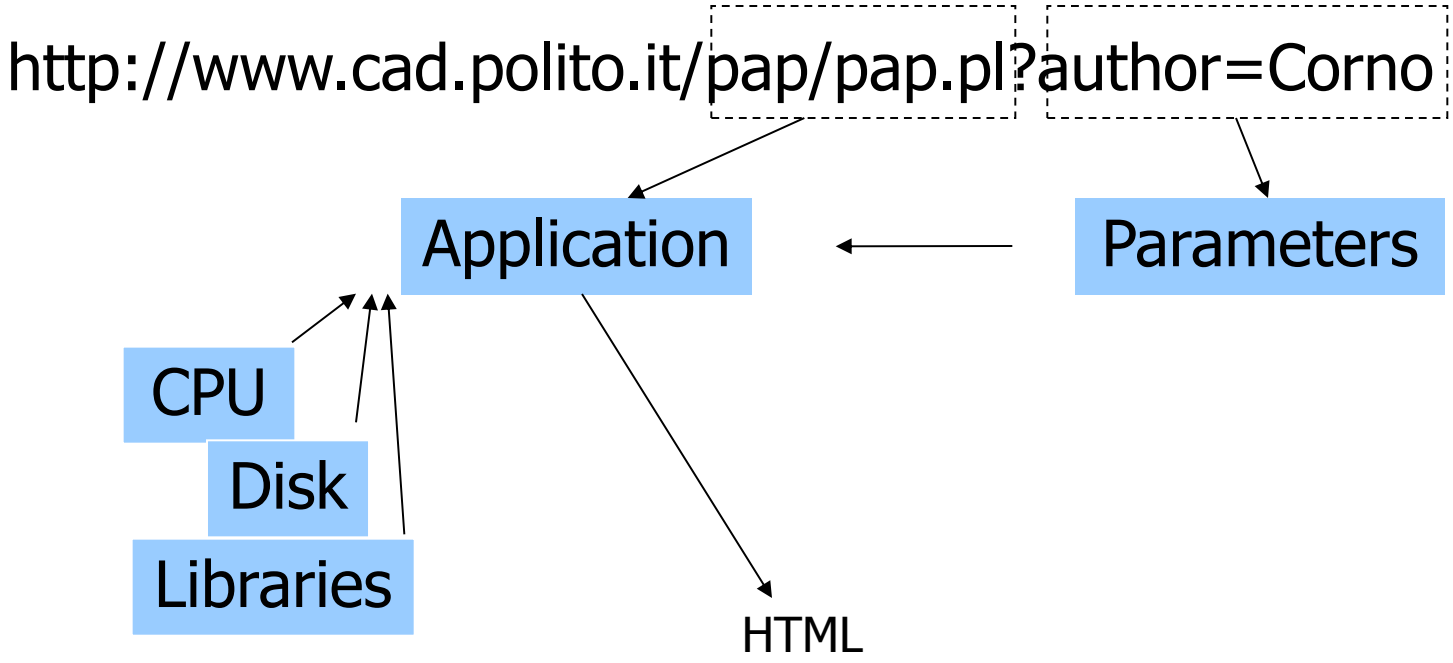




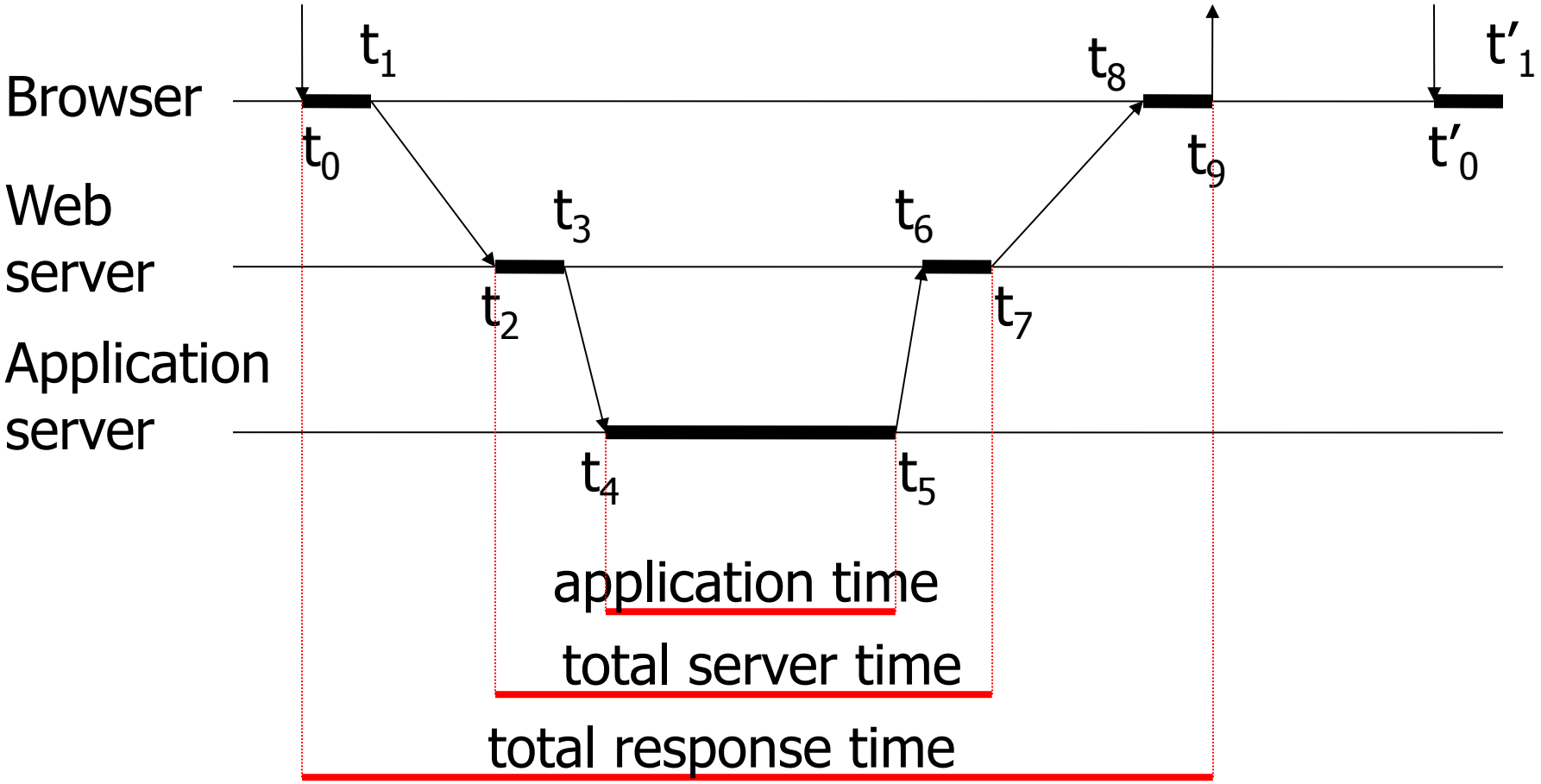
Adopted standards

- HTTP-POST for sending user-specified data
  - In addition to URL-encoding in GET requests
- Technologies for integrating application logic into web servers
  - Obsolete: CGI (common gateway interface), ISAPI (internet information server application programming interface), server-side script
  - java servlets
  - ASP (active server pages), JSP, PHP, PERL, Python as new languages for application development

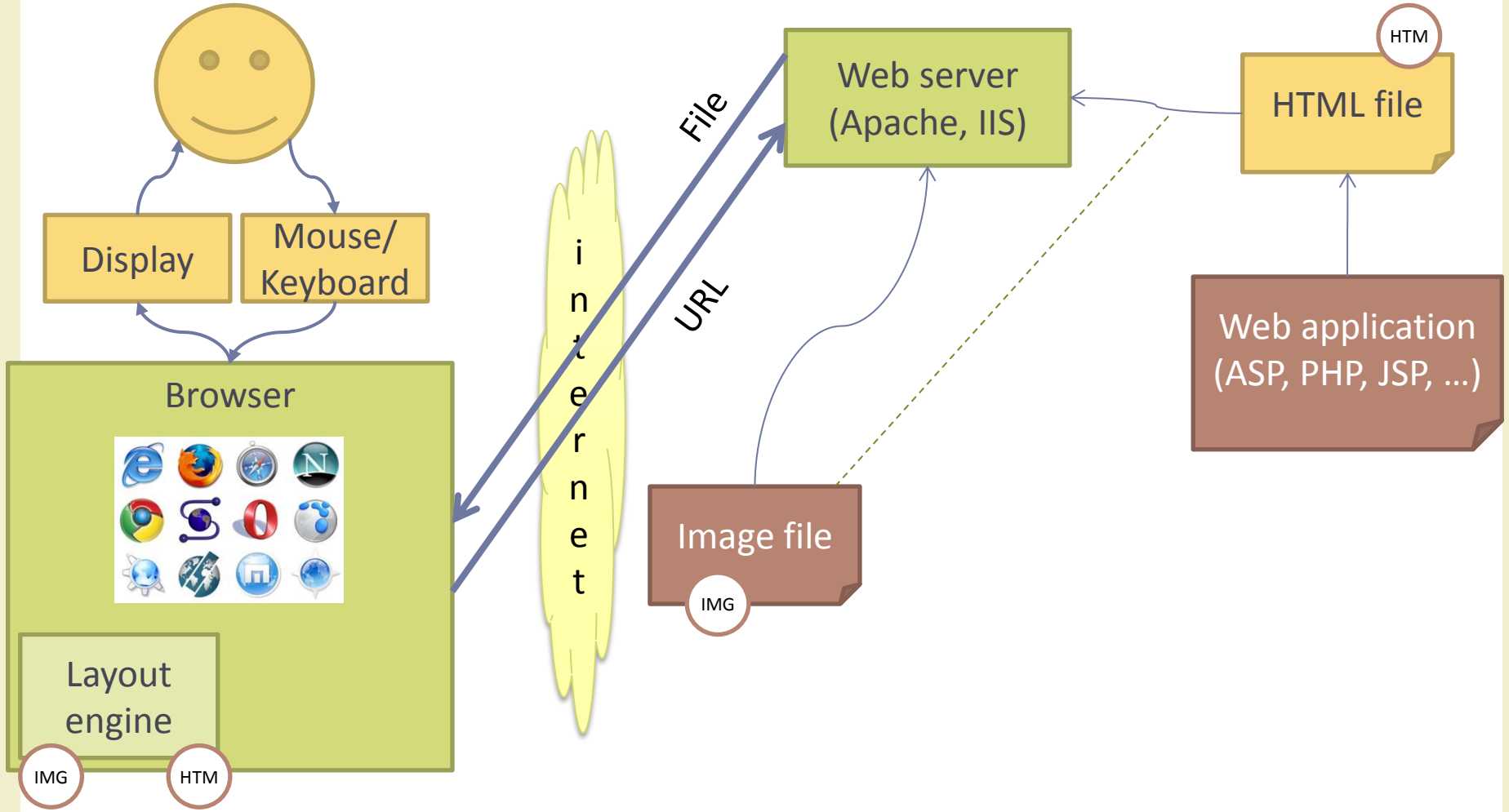
URL (HTTP GET)


http://www.cad.polito.it/pap/pap.pl?author=Corno
Application
Parameters
CPU
Disk
Libraries
HTML

Dynamic web transaction



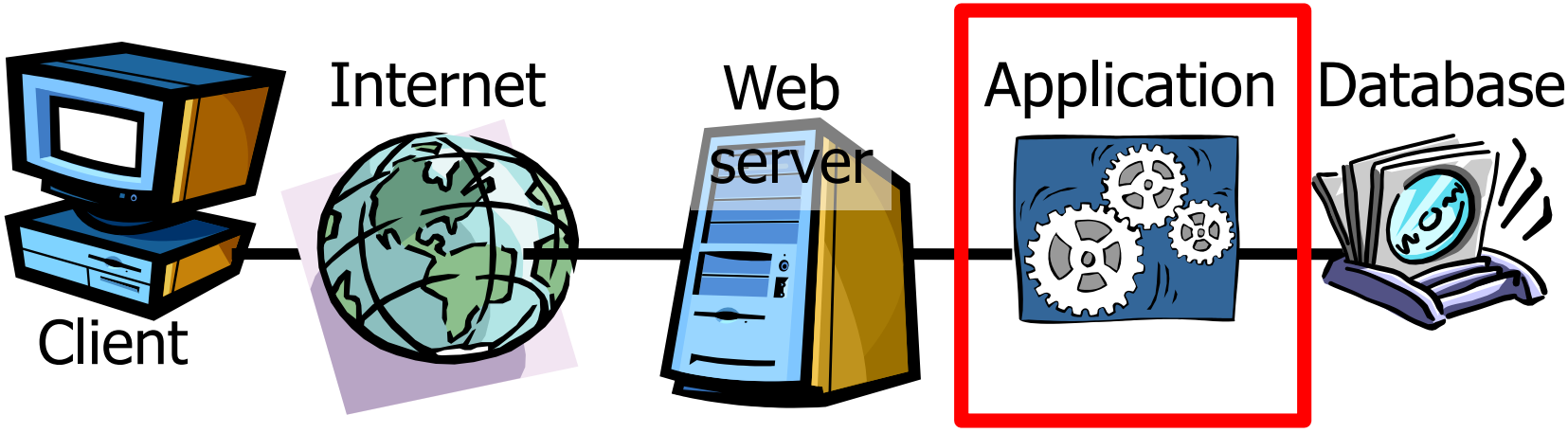
General Architecture

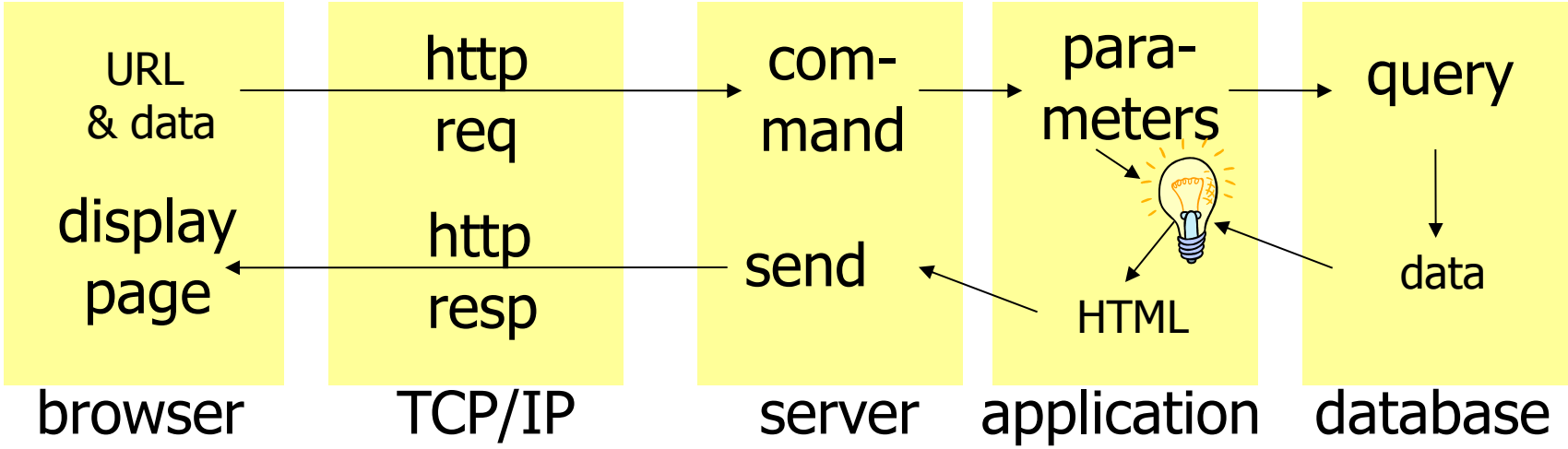


Database server

- Stores the data on which the application server works.
- Executes the queries issued by the application server:
  - Updates the stored data
  - Inserts new data
  - Provides back query results
- The most frequent/complex queries can be implemented internally as stored procedures (pre-compiled queries with parameters)

Example


Internet
Web server
Application
Database
Client


URL & data
http req
com-mand
para-meters
query
display page
http resp
send
data
HTML
browser
TCP/IP
server
application
database

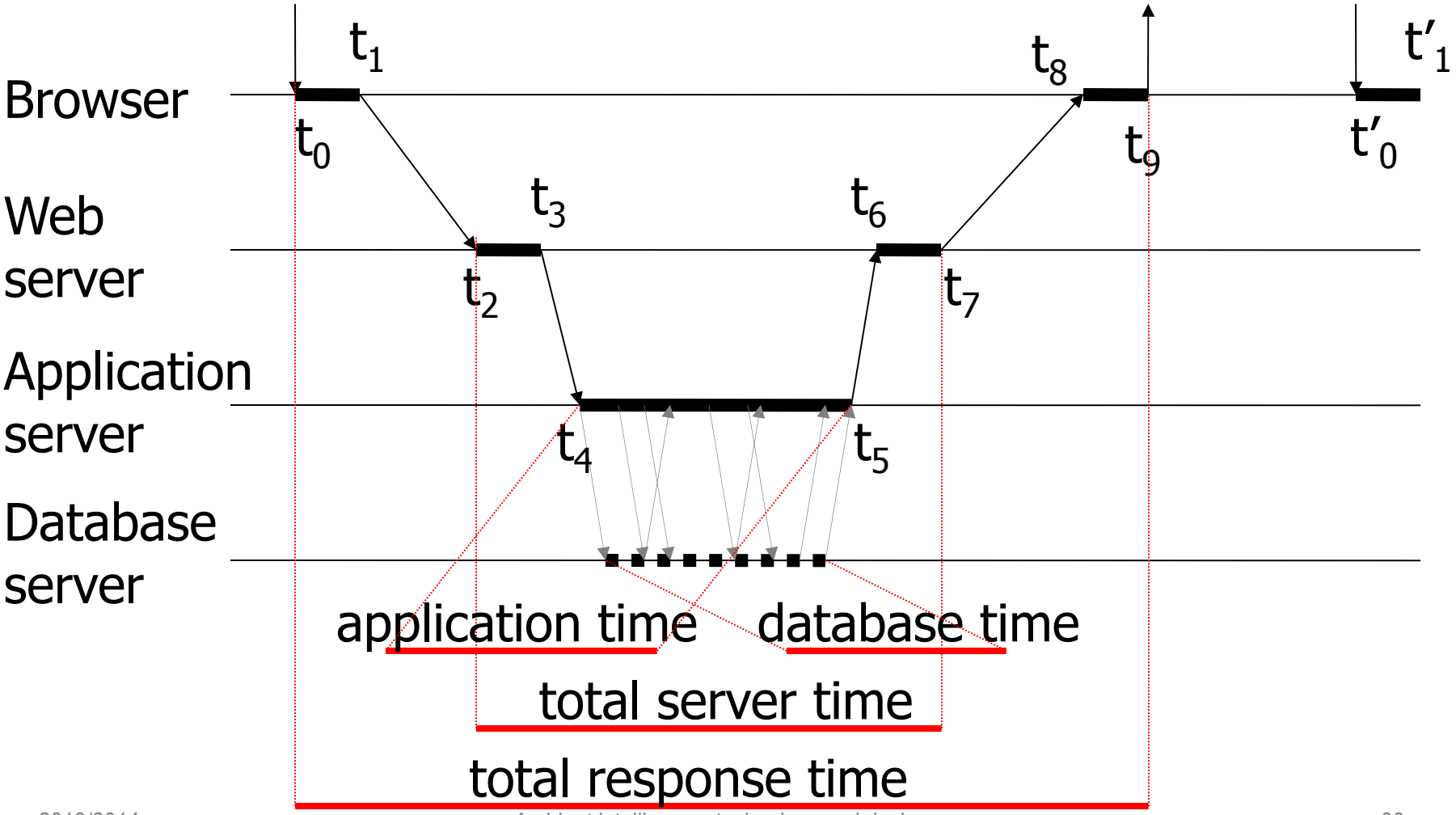
Adopted standards

- Cookies for storing the state of a session
- Java, JavaScript, ActiveX, Flash to program the user interface on the browser
- SQL (structured query language), ODBC (open database connectivity) to access data bases

Database server

- Queries are almost always in SQL
  - SELECT * FROM table;
  - ….
- Often adopts the relational database model
  - Other models can be used
    - Object model
    - Triple model
- The most advanced/complete solutions are called Transaction servers

Database-driven transaction



Example (PHP)

The application composes the query

```
<?php
$query = “SELECT doc_id FROM key_doc_index, keywords WHERE
key_doc_index.key_id = keywords.id AND keywords.key =
$_REQUEST[“query”];”;
```

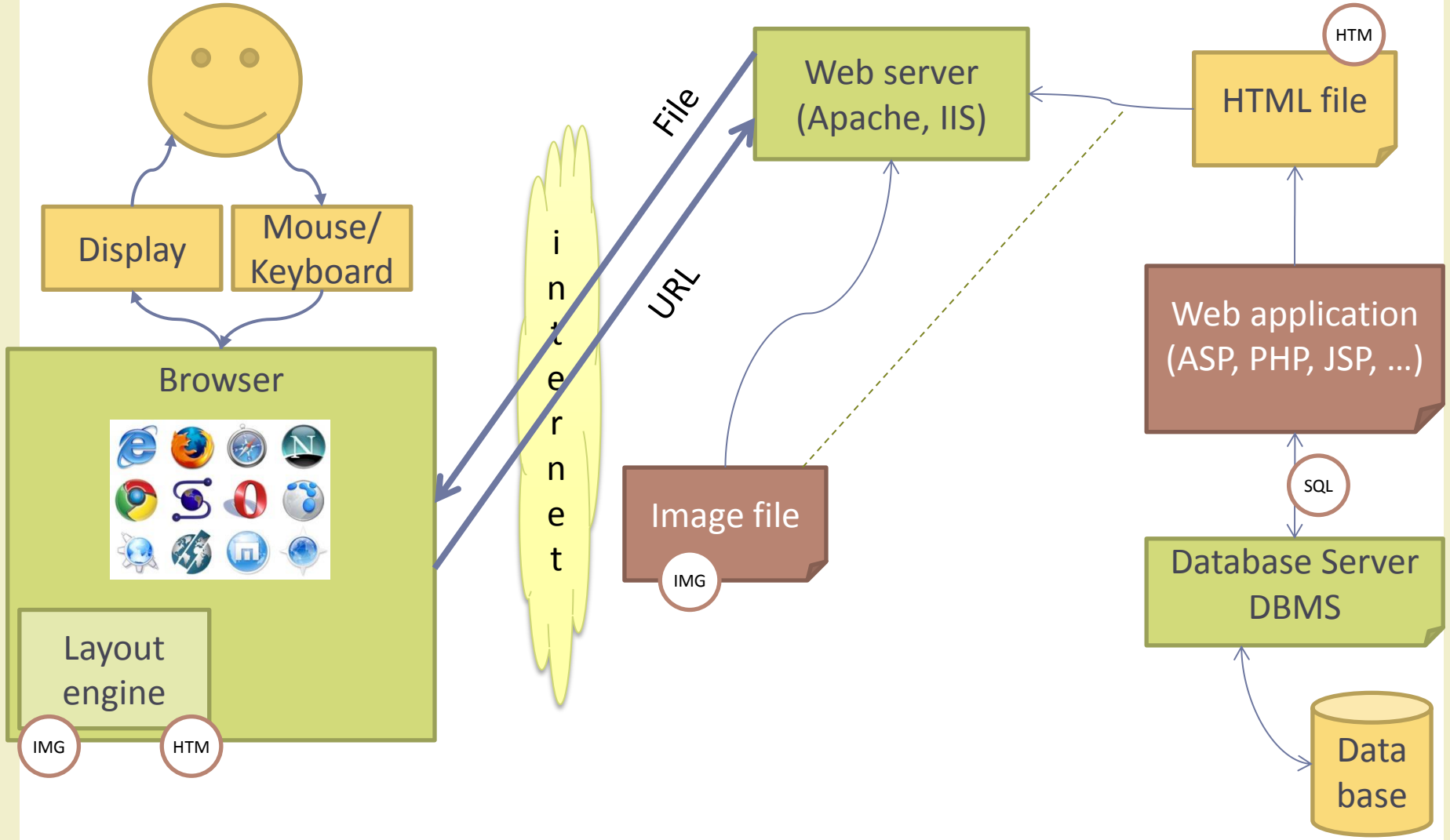
The query is sent to the db-server and a rowset containing the results is returned

```
$rowset = mysql_query($query);
```

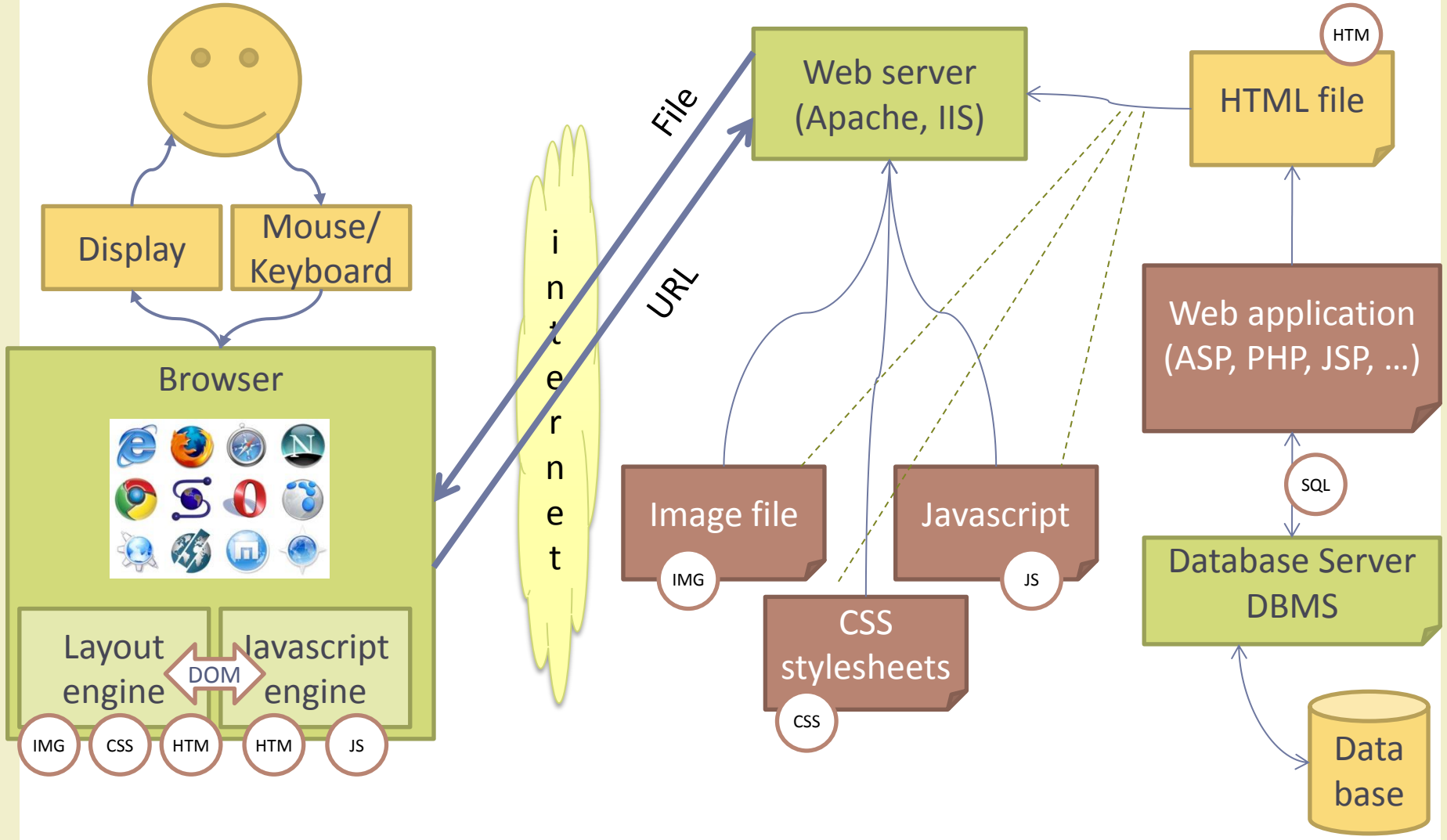
```
while($row = mysql_fetch_row($rowset))
{
//elaborate data
}
?>
```

The application elaborates the data

General Architecture



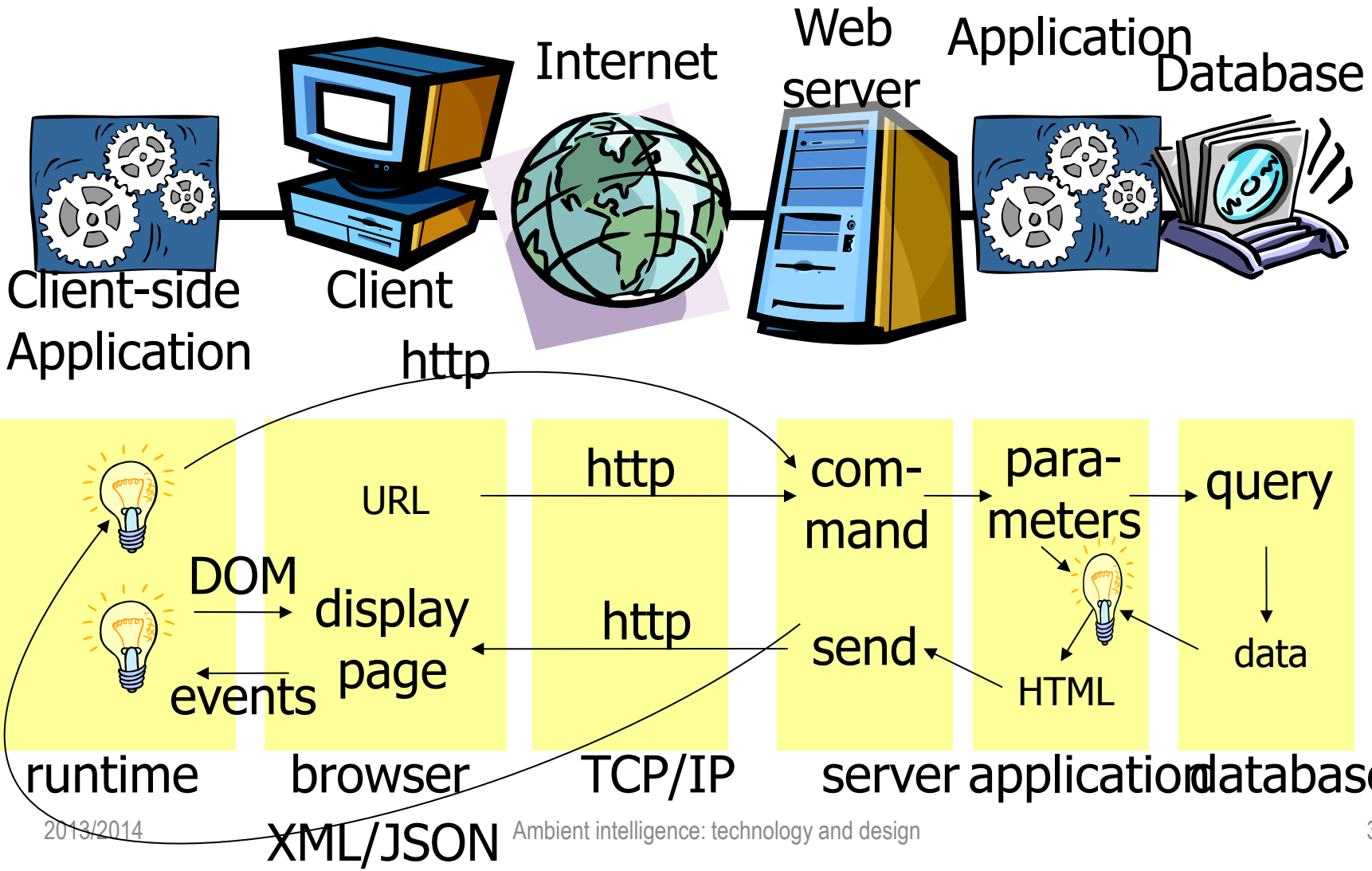
General Architecture



Web 2.0

- Web applications support social interaction models
- Peer exchange and user-contributed content instead of rigid publisher/reader pattern
  - Online communities
- Rich, dynamic, interactive user interfaces
- Integration of contents across web sites (mashups)

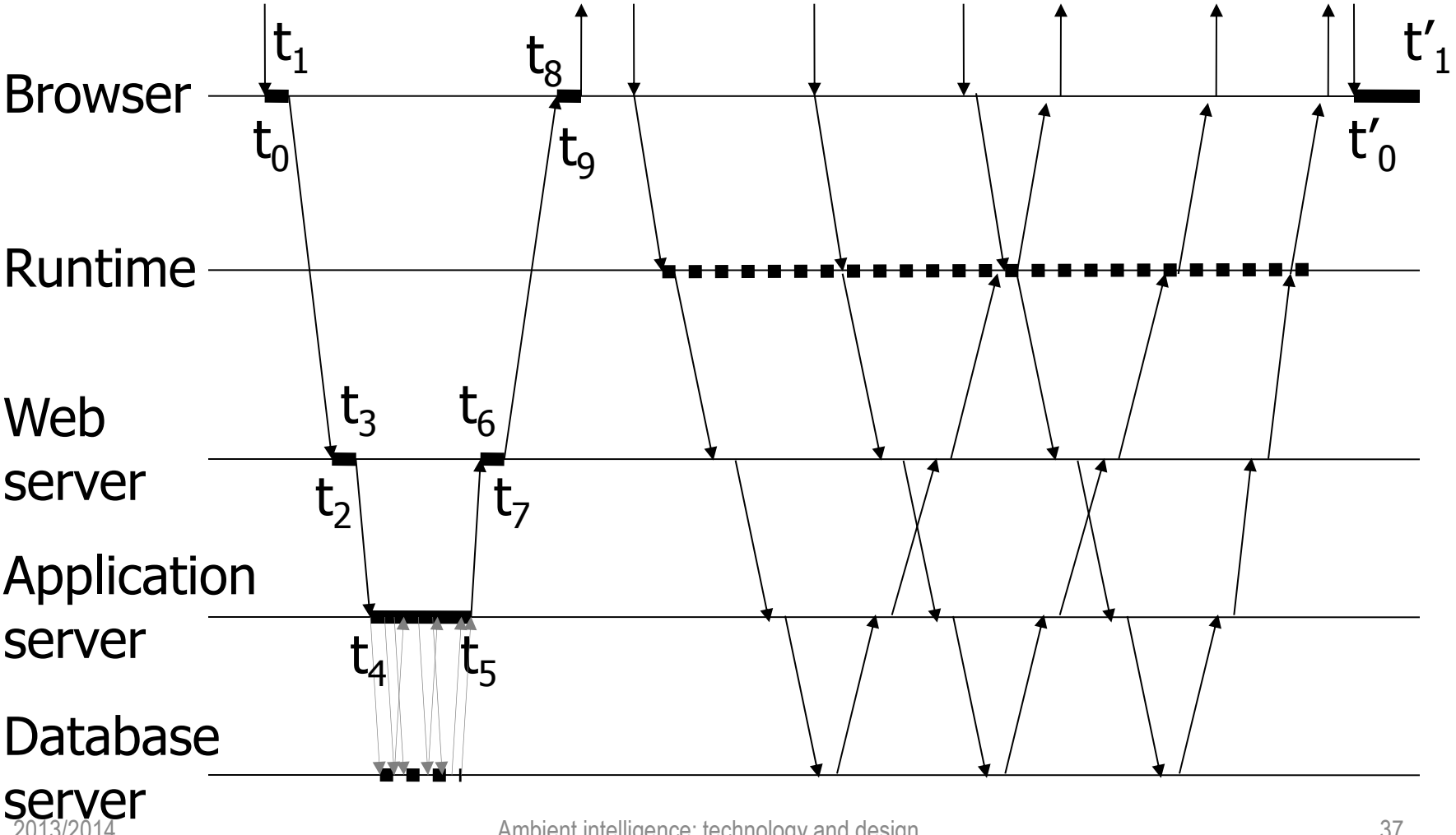
Rich-Client Asynchronous Transactions



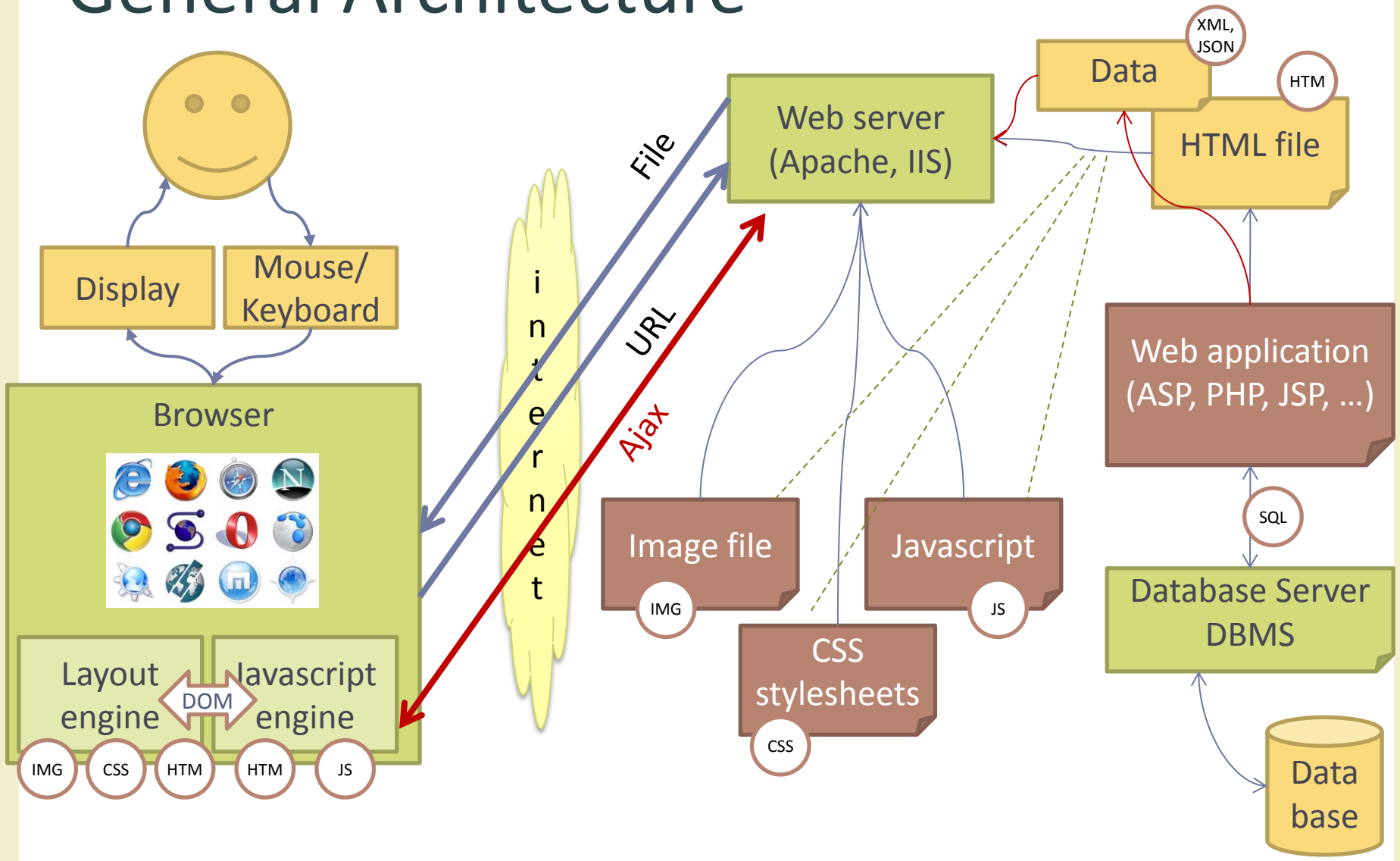
Adopted standards

- Dynamic HTML: DOM, Javascript, CSS
  - JavaScript, Flash to handle a runtime environment on the browser
  - DOM (XHTML Document Object Model) to allow on-the fly modification of the web page
  - CSS 2.1 to modify attribute and handle objects
- AJAX: Asynchronous Javascript and XML
  - XMLHttpRequest for asynchronous communication to the server
  - Data transfer formats: JSON, XML, RDF, RSS, Atom, FOAF, ...
- Mash-up technology

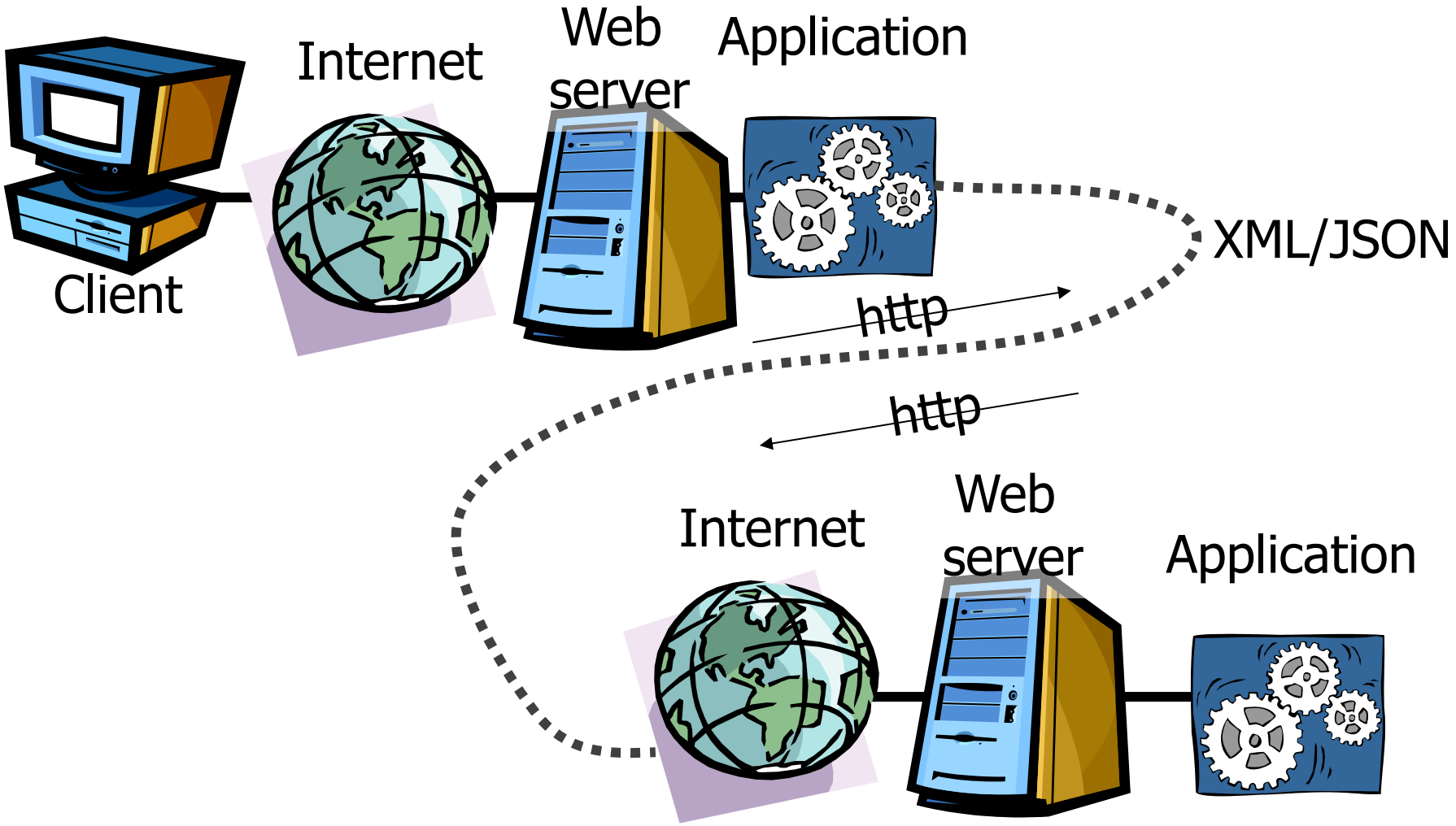
Rich-client transaction



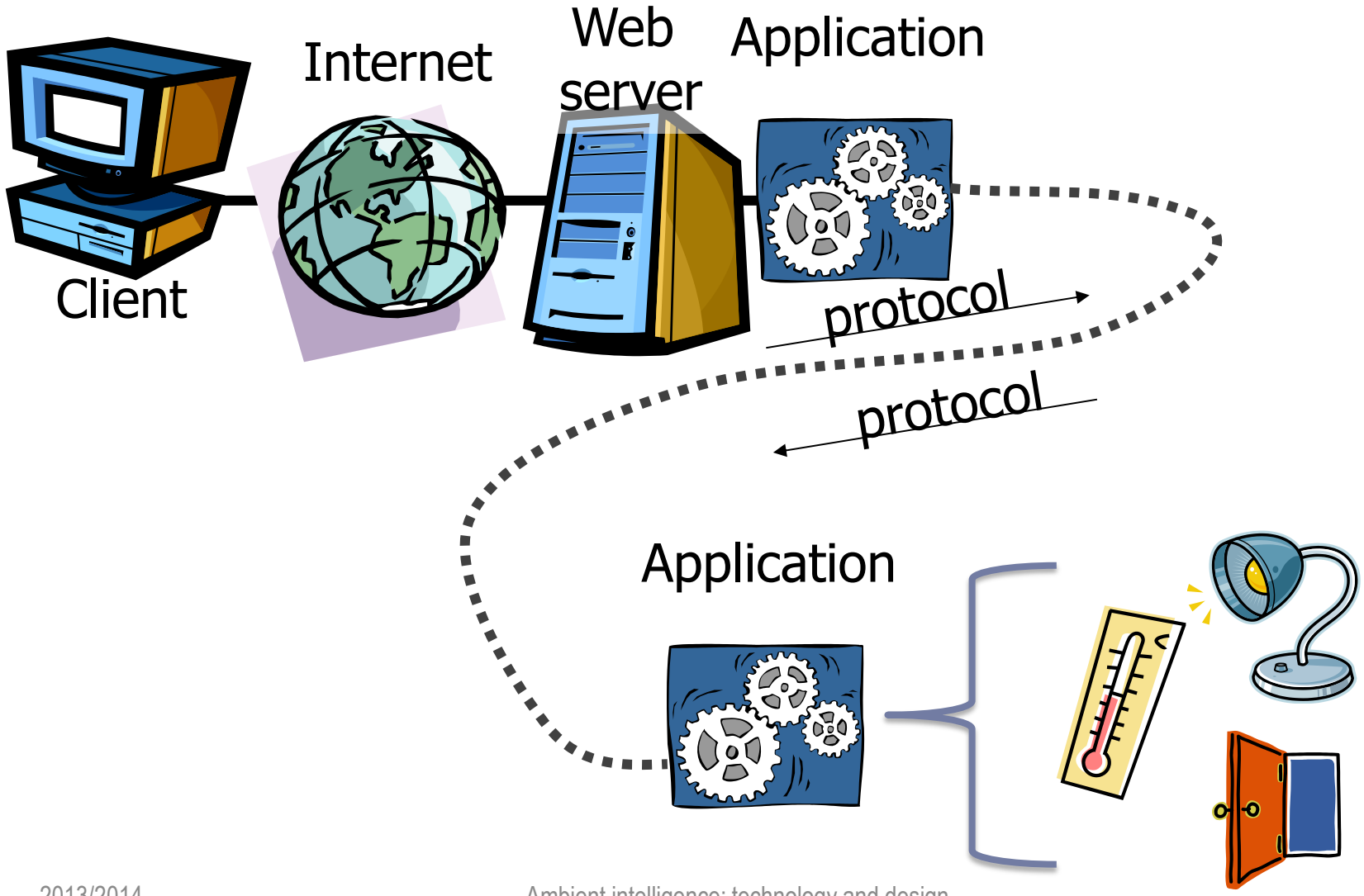
General Architecture



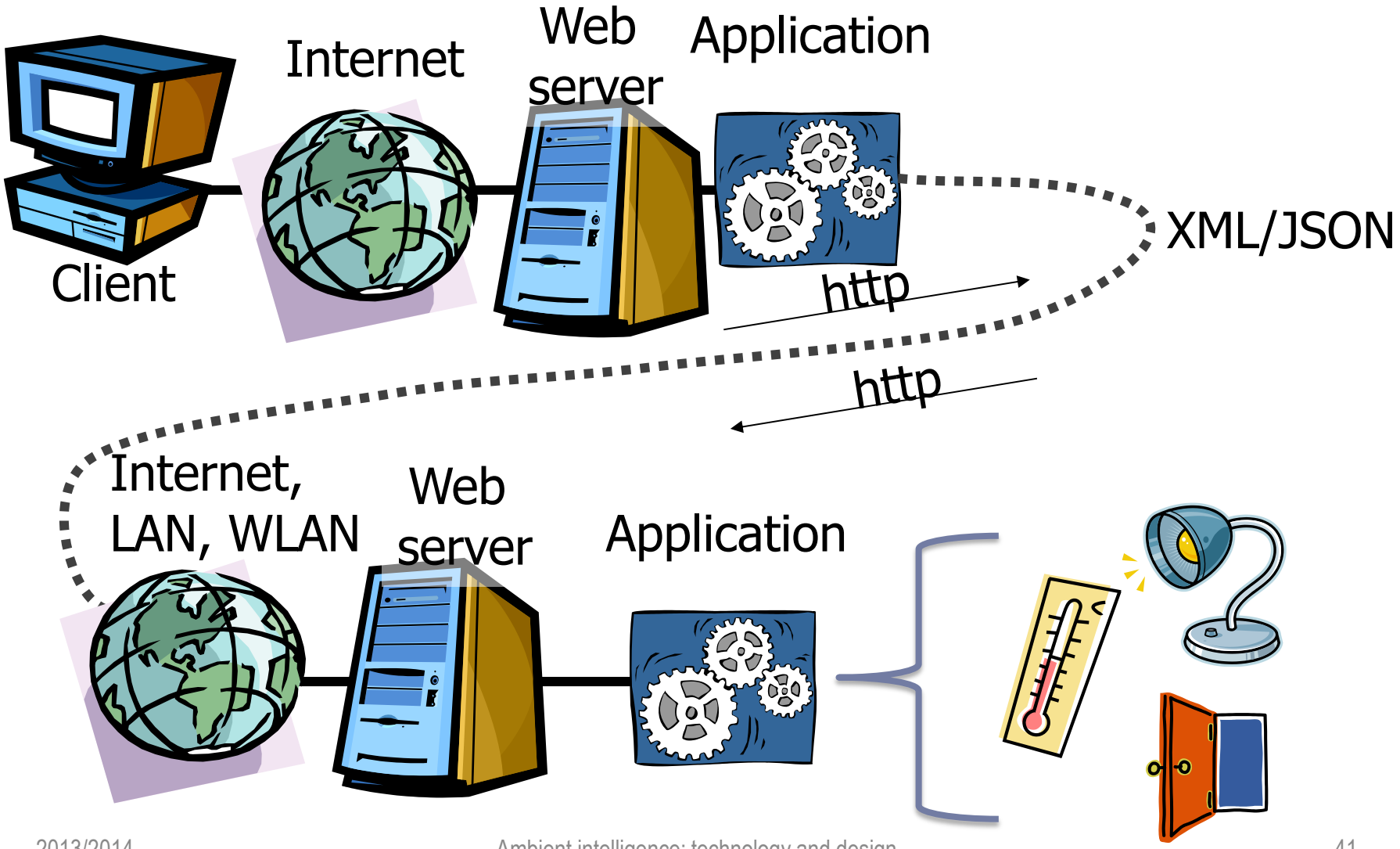
Distributed transactions



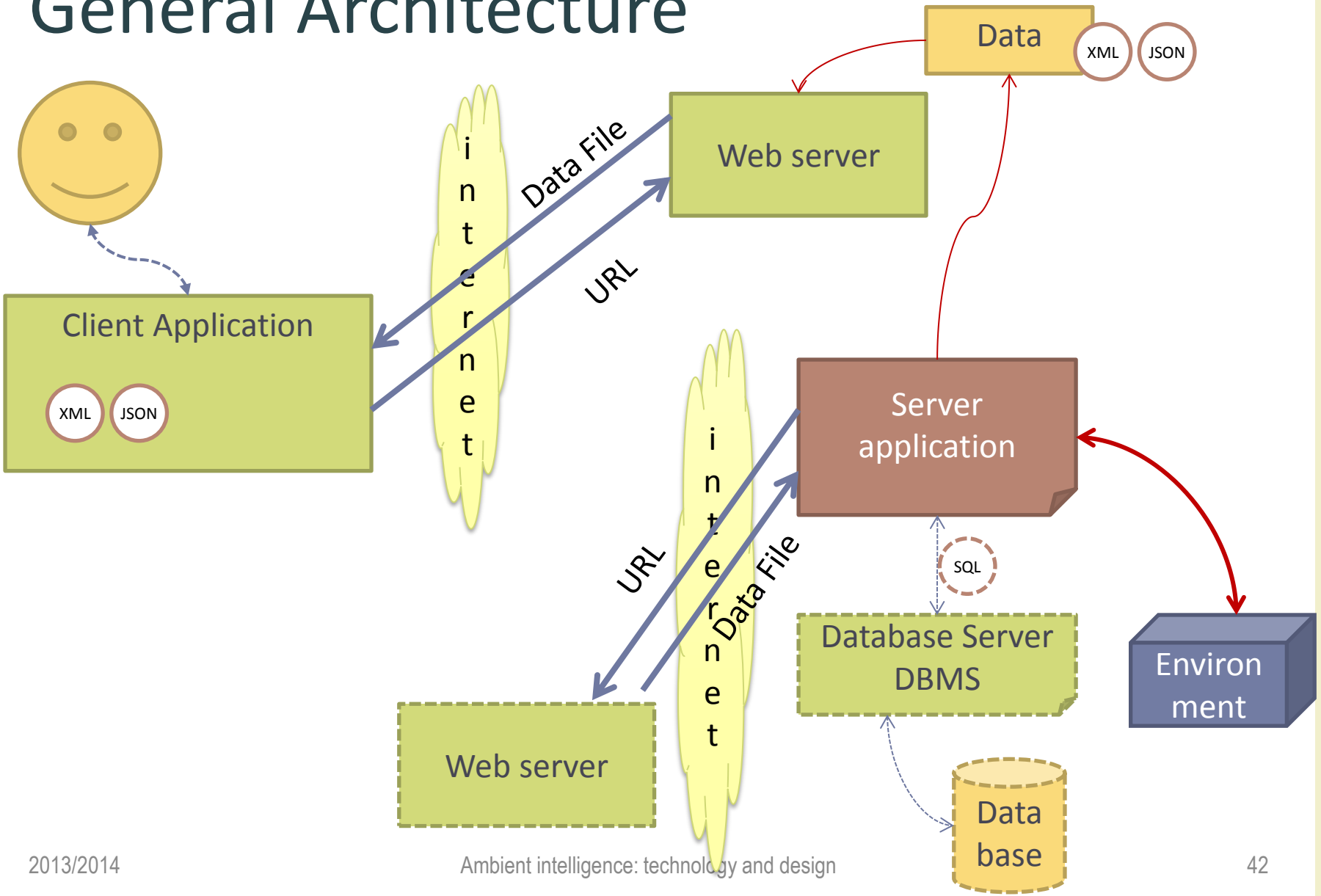
AmI control


Client
Internet
Web server
Application
protocol
protocol
Application

AmI control via http



General Architecture



Web Architecture and Technologies

HTTP

HYPERTEXT TRANSFER PROTOCOL

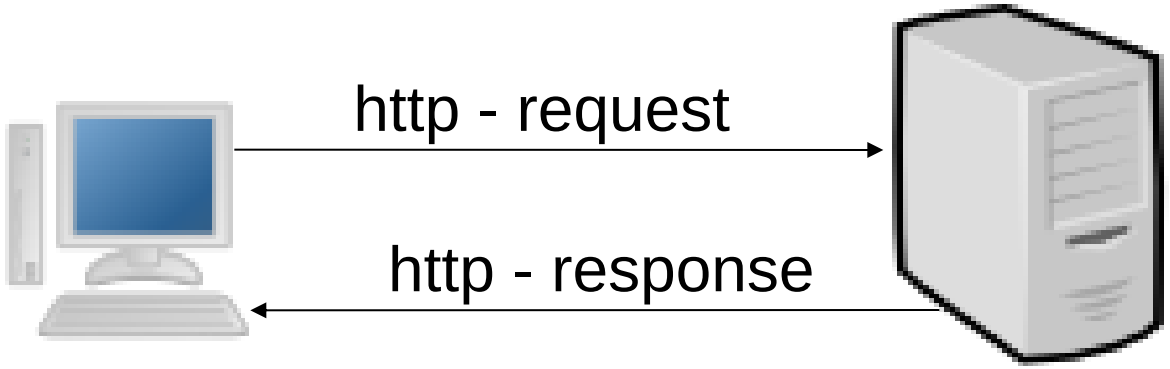
What is HTTP?

- HTTP stands for Hypertext Transfer Protocol
- It is the network protocol used to delivery virtually all data over the WWW:
  - Images
  - HTML files
  - Query results
  - Etc.
- HTTP takes places over TCP/IP connections

http://www.ietf.org/rfc/rfc2616.txt

HTTP clients and servers

- A browser is an HTTP client because it sends requests to an HTTP server, which then sends responses back to the client.
- The standard port for HTTP servers to listen on is 80, though they can use any port.



HTTP messages

- The format of the request and response messages are similar.
  - An initial line
  - Zero or more header lines
  - A blank line (CRLF)
  - An optional message body

```
Initial line
header1: value1
header2: value2
header3: value3

message body...
```

Header Example

```
HEAD /index.html HTTP/1.1
Host: www.example.com
```

Request

Response

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

HTTP request – initial line

- The initial line is different for the request and the response.
- A **request** initial line has three parts separated by white spaces:
  - A method name
  - The local path of the requested resource
  - The version of the HTTP being used

- `GET /path/to/file/index.html HTTP/1.0`

HTTP request – initial line

- The method name is always in upper case.
- There are several methods for a HTTP request
  - GET (most commonly used)
  - POST (used for sending form data)
  - HEAD
  - ...
- The path is the part of the URL after the host name
  - http://www.tryme.com/examples/example1.html

HTTP Method Basics

HEAD	Gets just the HTTP header
GET	Gets HTTP head & body
POST	Submits data in the body to the server
PUT	Uploads a resource
DELETE	Deletes a resource
TRACE	Echo’s back the request
OPTIONS	Gets a list of supported methods
CONNECT	Converts to a TCP/IP tunnel for HTTPS
PATCH	Apply partial modifications to a resource

HTTP request – initial line

- The HTTP version is always in the form
  - HTTP/x.x (uppercase)
- The versions currently in use are:
  - HTTP/1.0
  - HTTP/1.1

HTTP response – initial line

- The **response** initial line is usually called status line and has also 3 parts separated by spaces:
  - The HTTP version
  - The response status code
  - An English phrase describing the status code
- Example:
  - `HTTP/1.0 200 OK`
  - `HTTP/1.0 404 Not Found`

Response Status Codes

- 1xx – Informational
- 2xx – Success
- 3xx – Redirection
- 4xx – Client Error
- 5xx – Server Error

Response Status Codes

- 1xx – Informational
- 2xx – Success
- 3xx – Redirection
- 4xx – Client Error
- 5xx – Server Error

- 100 = Continue
- 102 = Processing
- 200 = OK
- 201 = Created
- 204 = No Content
- 206 = Partial Content
- 301 = Moved Permanently
- 302 = Found (Moved Temp)
- 307 = Temp Redirect
- 400 = Bad Request
- 401 = Unauthorised
- 402 = Payment Required
- 403 = Forbidden
- 404 = Not Found
- 405 = Method Not Allowed
- 409 = Conflict
- 450 = Blocked by Windows Parental Controls
- 500 = Internal Server Error
- 501 = Not Implemented

HTTP msg – header lines

- Header lines provide information about the request/response or about the object sent in the message body
- The header lines are in the following format:
  - One line per header
  - Form: “Header-Name: value”
- HTTP/1.0 defines 16 headers (none required); HTTP/1.1 defines 46 headers and 1 is required in requests:
  - `Host:`

Request headers

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization;
- Expect
- From
- **Host**
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

Response Headers

- Accept-Ranges
- Age
- Etag
- Location
- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

General (request & response) headers

- `Cache-Control`
- `Connection`
- `Date`
- `Pragma`
- `Trailer`
- `Transfer-Encoding`
- `Upgrade`
- `Via`
- `Warning`

Message body

- An HTTP message may have a **body** of data sent after the header lines.
- In a **response** the body contains the resource returned to the client
  - Images
  - text/plain, text/html
  - …
- In a **request** it may contain the data entered by the user in a form or a file to upload, etc.

Content Type

- Proper name: Internet Media Type
  - Also known as MIME type
- Parts: Type, SubType, Optional Parameters
- `x-` prefix for nonstandard types or subtypes
- `vnd.` prefix for vendor specific subtypes

Content Type Examples

Content-Type	File
text/plain	Plain text
text/xml	XML
text/html	HTML
image/png	PNG image
audio/basic	Wave audio
audio/mpeg	MPEG audio (MP3)
video/quicktime	Quicktime Video
application/pdf	Adobe PDF document
application/javascript	JavaScript
application/vnd.ms-powerpoint	PowerPoint file
application/x-rar-compressed	RAR file

Message body

- Some HTTP headers are used to describe the body content:
  - `Allow`
  - `Content-Encoding`
  - `Content-Language`
  - `Content-Length`
  - `Content-Location`
  - `Content-MD5`
  - `Content-Range`
  - `Content-Type`
  - `Expires`
  - `Last-Modified`
  - `extension-header n`

HTTP Authentication

- Basic Authentication
  - Easy to do, but plain text. Easy to reverse engineer. Less of an issue when used with SSL.
- Digest Authentication
  - Harder to do, still plain text. Hard (impossible?) to reverse engineer because of hashing.
- NTLM Authentication
  - Hard to do, Windows specific. Hard (impossible?) to reverse engineer.
- Note: usually, authentication is dealt at the application level, and http mechanisms are not used

HTTP methods: HEAD

- The HEAD method is like the GET except it asks the server to return the **response headers, only**. Is useful for checking the characteristics of a resource without actually downloading it.
- The response to a HEAD request **never** contains a message body, only the initial line and the headers.

HTTP methods: POST

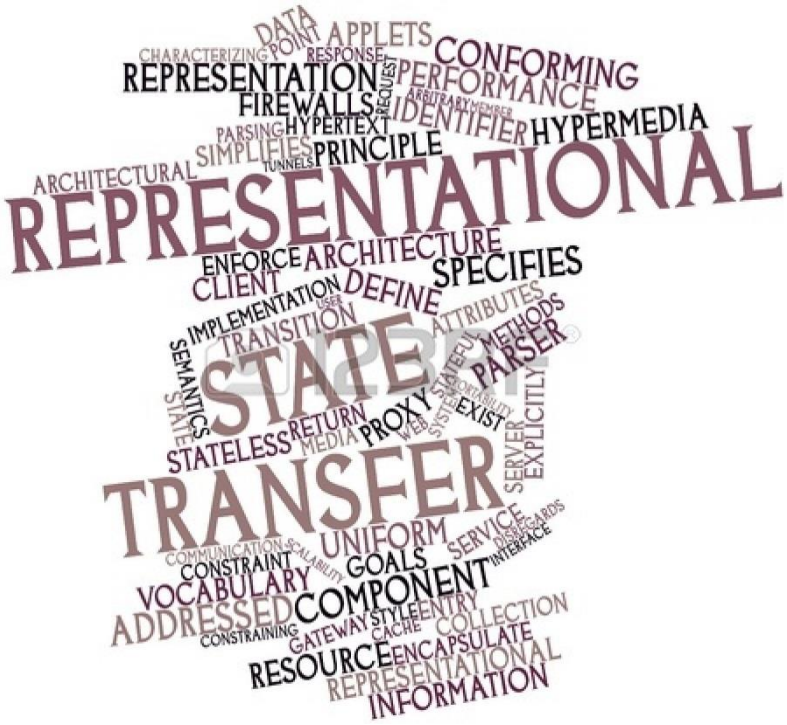
- Used to send data to the server
- A POST request is different from the GET request as:
  - There's a block of data sent with the request in the request message body
  - The request URI is not a resource to retrieve, it's usually a program or a server page that handles the sent data
  - The HTTP response is usually not-static (generated depending on the received data)

GET vs POST

- The most common use of the POST method is to submit data gathered from user forms
- Also the GET can be used to submit form data however, the data is encoded in the request URI
  - http://www.example.com/example.html?var=This+is+a+simple+%26+short+test
- GET requests should be **idempotent**, i.e., may be repeated without changing the state of the application

HTTP as transport layer

- HTTP is used as “transport” for many resources / protocols
- Protocols:
  - SOAP (Simple Object Access Protocol)
  - XML-RPC
  - WebDAV
- Resources:
  - Text (plain, HTML, XHTML, ...)
  - Images (gif, jpeg, ...)
  - ....



Web Architecture and Technologies

REST

REPRESENTATIONAL STATE TRANSFER

REST

Roy T. Fielding

Senior Principal Scientist, Adobe
Co-founder, Apache HTTP Server Project
Director, The Apache Software Foundation
Ph.D., Information and Computer Science, UC Irvine

- @fielding; Blog: Untangled
- Email: fielding at (choose **one** of) gbiv.com, adobe.com, apache.org

- **Representational State Transfer**
- A style of software architecture for distributed systems
- Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else),
- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP)
- Can easily be used in the presence of firewalls.

What is a Resource?

- A resource can be anything that has identity
  - a document or image
  - a service, e.g., “today’s weather in Seattle”
  - a collection of other resources
  - non-networked objects (e.g., people)
- The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity that corresponds to that mapping at any particular point in time!

Main principles

- Resource: source of specific information
- Mapping: Resources ⇔ URIs
- Client and server exchange *representations* of the resource
  - The same resource may have different representations
  - E.g., XML, JSON, HTML, RDF, …
- Operations on the Resource is done by means of HTTP methods
  - GET, POST, PUT, DELETE

Main Types of Resources

- **Collection** resource
  - Represents a set (or list) of items
  - Format: /resource
  - E.g., `http://api.polito.it/students`
    `http://api.polito.it/courses`
- **Element** (Item) resource
  - Represents a single item, and its properties
  - Format: /resource/identifier
  - E.g., `http://api.polito.it/students/s123456`
    `http://api.polito.it/courses/01PRD`

Best practice

- Nouns (not verbs)
- Plural nouns
- Concrete names (not abstract)
  - /courses, not /items

Actions use HTTP Methods

- GET
  - Retrieve the representation of the resource (in http response body)
  - Collection: the list of items
  - Element: the properties of the element
- POST
  - Create a new resource (data in http request body)
  - Use a URI for a Collection
- PUT
  - Update an existing element (data in http request body)
  - Mainly for Elements' properties
- DELETE

Actions on Resources - Example

Resource	POST create	GET read	PUT update	DELETE delete
/dogs	Create a new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/1234	Error	Show Bo	If exists update Bo If not error	Delete Bo

Relationships

- A given Element may have a (1:1 or 1:N) relationship with other Element(s)
- Represent with: /resource/identifier/resource
- E.g.,
  `http://api.polito.it/students/s123456/courses`
  `http://api.polito.it/courses/01PRD/students`

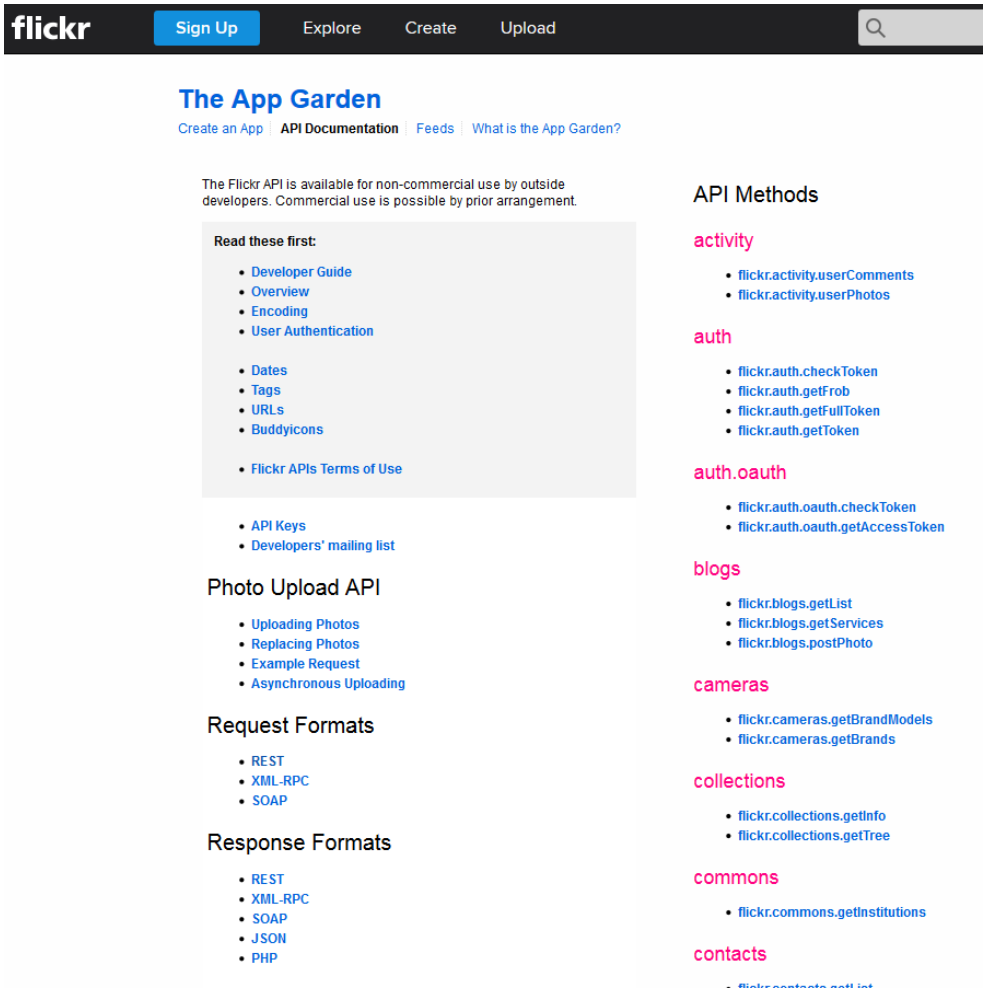
Representations

- Returned in GET, sent in PUT/POST
- Different formats are possible
- Mainly: XML, JSON
  - But also: SVG, JPEG, TXT, …
  - In POST: URL-encoding
- Format may be specified in
  - Request headers
    - `Accept: application/json`
  - URI extension
    - `http://api.polito.it/students/s123456.json`
  - Request parameter
    - `http://api.polito.it/students/s123456?format=json`

Real Life: Flickr API

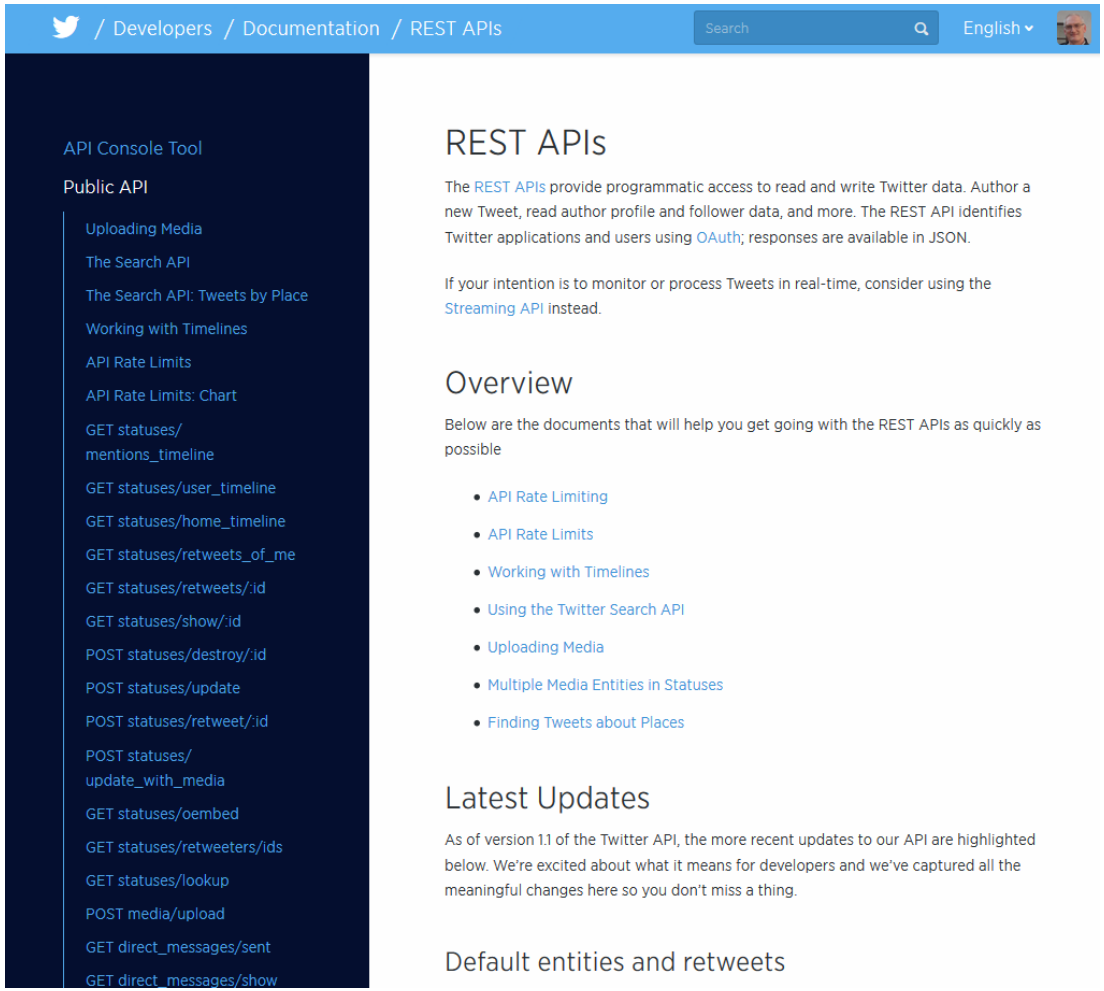
- Resource: Photos
- Where:
  - `http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg`
  - `http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}_[mstb].jpg`
  - `http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)`
- What: JPEG, GIF or PNG (defined in the URL)
  - `http://farm1.static.flickr.com/2/1418878_1e92283336_m.jpg`

Real Life: flickr



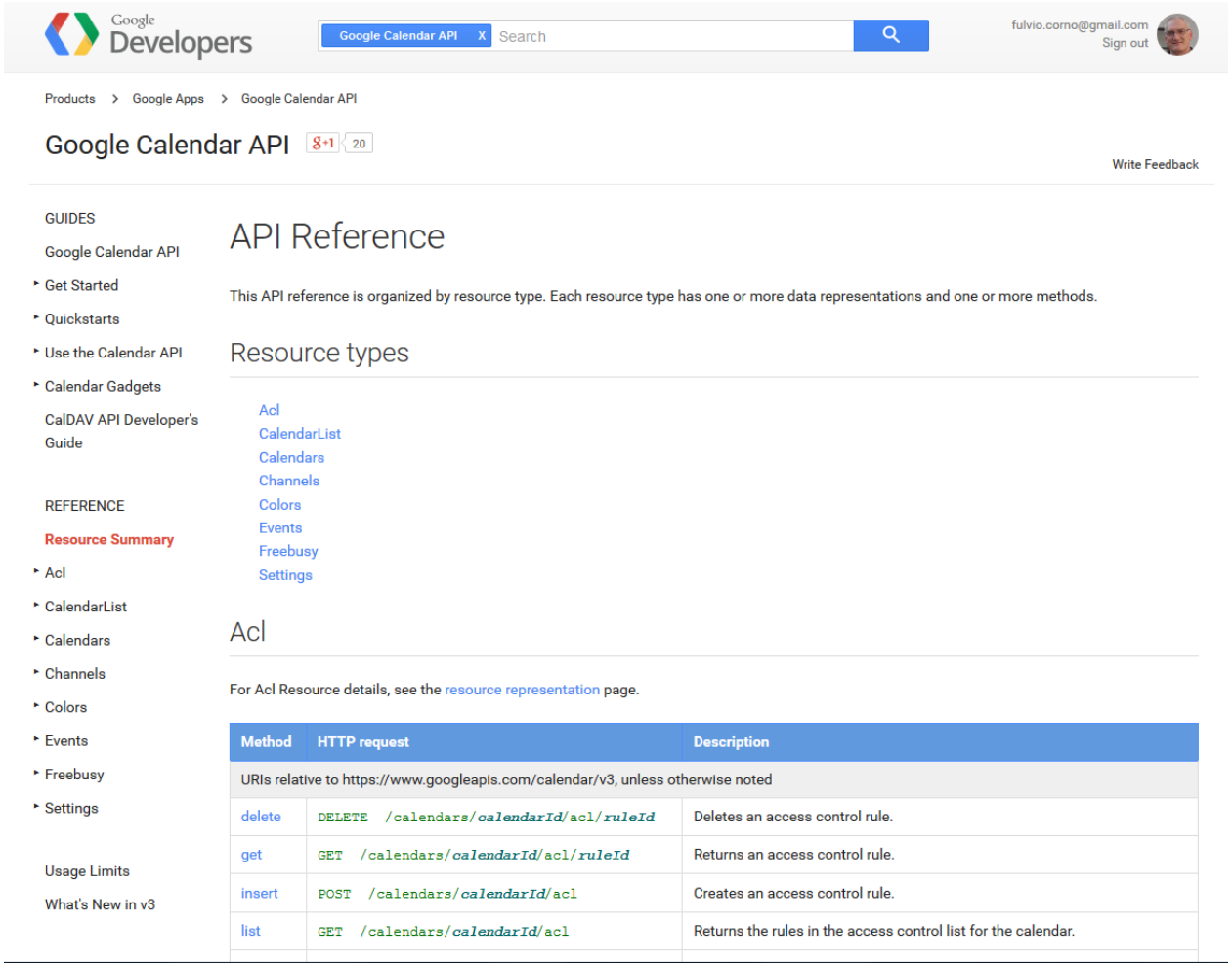
https://www.flickr.com/services/api/

Real Life: Twitter API



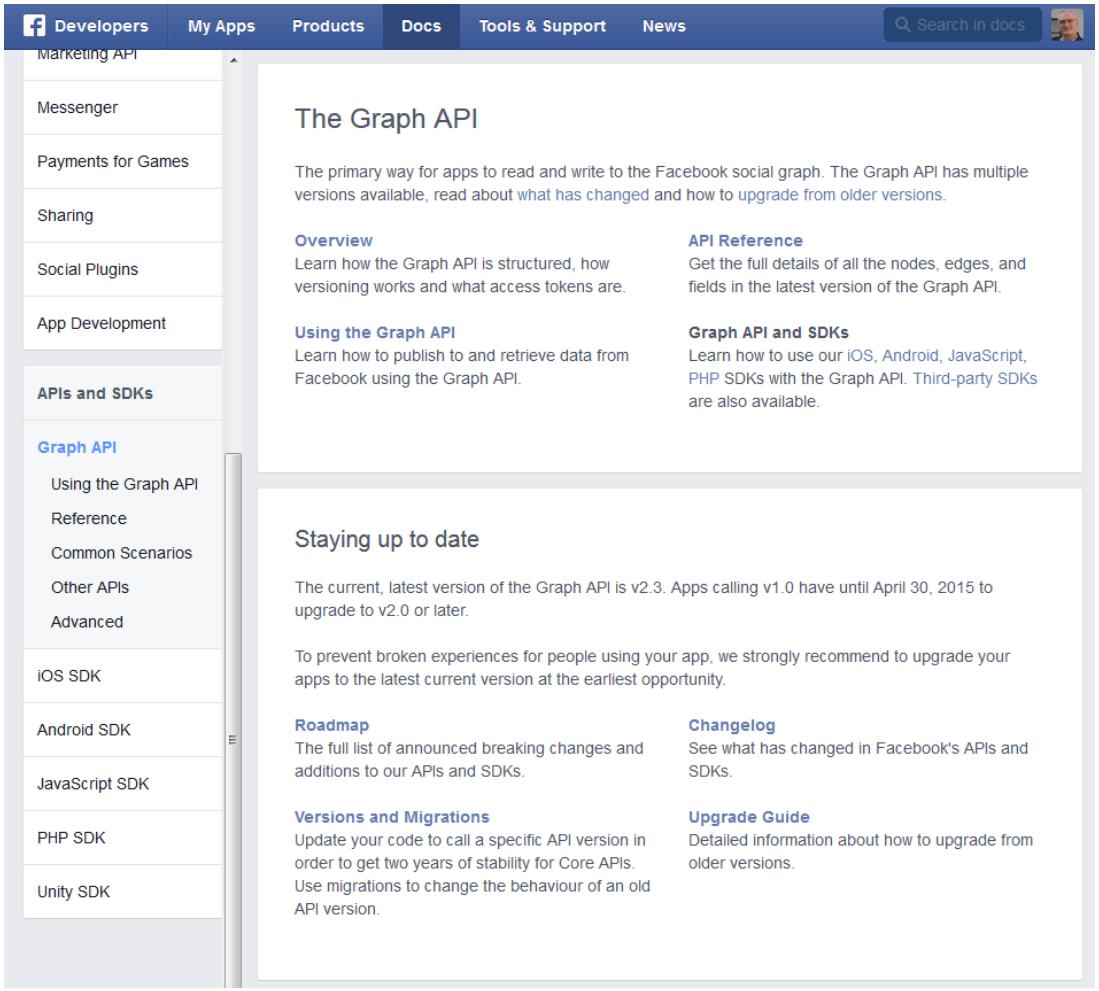
https://dev.twitter.com/rest/public

Real Life: Google Calendar API



https://developers.google.com/google-apps/calendar/v3/reference/

Real life: Facebook Graph API



https://developers.facebook.com/docs/graph-api

Complex resource search

- Use `?`*`parameter`*`=`*`value`* for more advanced resource filtering (or search)
  - E.g., `https://api.twitter.com/1.1/statuses/user_timeline.json`**`?screen_name=twitterapi&count=2`**

Errors

- When errors or exceptions are encountered, use meaningful HTTP Status Codes
  - The Response Body may contain additional information (e.g., informational error messages)

```
{
   "developerMessage" : "Verbose, plain language description of
the problem for the app developer with hints about how to fix
it.",
   "userMessage":"Pass this message on to the app user if
needed.",
   "errorCode" : 12345,
   "more info": "http://dev.teachdogrest.com/errors/12345"
}
```

Authentication

Twitter Streaming API

```
Authorization: Basic aWhlYXJ0OmFwaXM=
```

Amazon Web Services API

```
Authorization: AWS
AKIAIOSFODNN7EXAMPLE:frJIUNo//yllqDzg=
```

Google API

```
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
```

Guidelines

- Design with standards in mind – for example RSS & ATOM
- Create should return URIs not resources
- Use the right HTTP methods for the right actions
- You are on HTTP – use the infrastructure.
  - Proxy, Caching, Etag, Expires

Guidelines (1/2)

URL Design	
Plural nouns for collections	`/dogs`
ID for entity	`/dogs/1234`
Associations	`/owners/5678/dogs`
4 HTTP Methods	`POST GET PUT DELETE`
Bias toward concrete names	`/dogs (not animals)`
Multiple formats in URL	`/dogs.json` `/dogs.xml`
Paginate with limit and offset	`?limit=10&offset=0`
Query params	`?color=red&state=running`
Partial selection	`?fields=name,state`
Use medial capitalization	`"createdAt": 1320296464` `myObject.createdAt;`
Use verbs for non-resource requests	`/convert?from=EUR&to=CNY&amount=100`
Search	`/search?q=happy%2Blabrador`
DNS	`api.foo.com` `developers.foo.com`

Guidelines (2/2)

Versioning	
Include version in URL	`/v1/dogs`
Keep one previous version long enough for developers to migrate	`/v1/dogs` `/v2/dogs`

Errors	
8 Status Codes	`200 201 304 400 401 403 404 500`
Verbose messages	`{"msg": "verbose, plain language hints"}`

Client Considerations	
Client does not support HTTP status codes	`?suppress_response_codes=true`

Client does not support HTTP methods	`GET /dogs?method=post`
	`GET /dogs`
	`GET /dogs?method=put`
	`GET /dogs?method=delete`

Complement API with SDK and code libraries	1. JavaScript 2. … 3. …

Web Architecture and Technologies

JSON

JAVASCRIPT OBJECT NOTATION

JSON – What is it?

- “JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate” – JSON.org

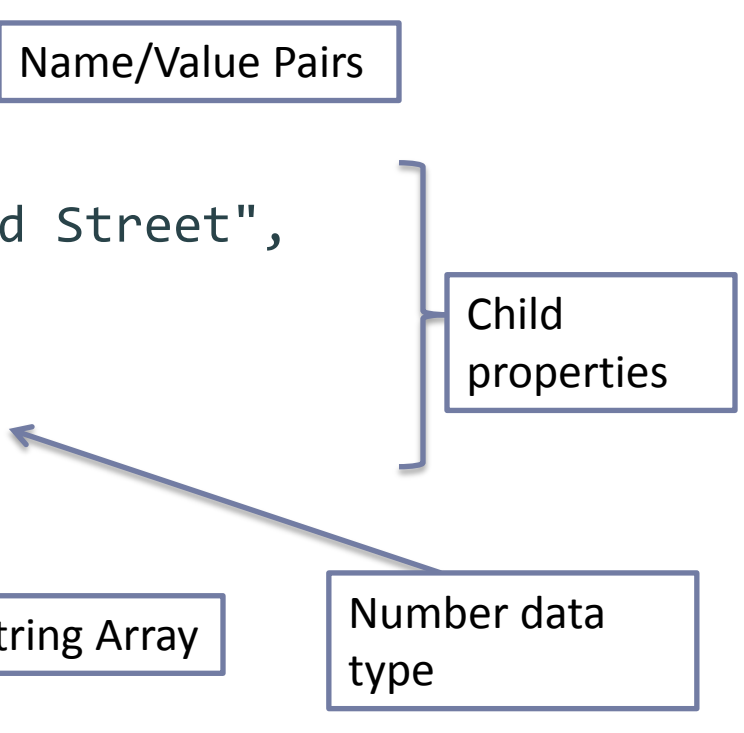
- Important: JSON is a subset of JavaScript

Json Logical Structure

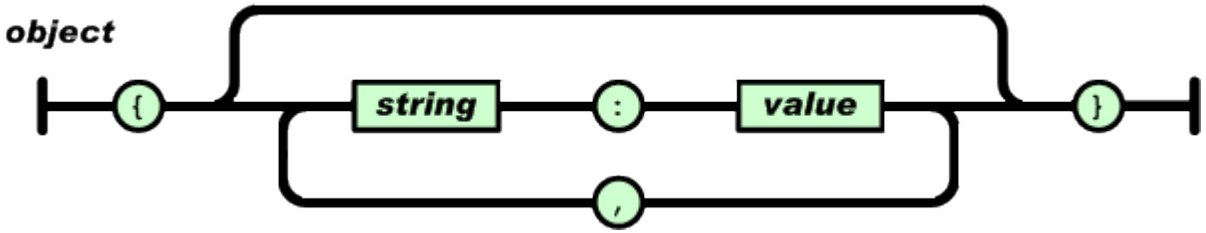
- JSON is built on two structures:
  - A **collection** of name/value pairs. In various languages, this is realized as an ***object***, record, struct, dictionary, hash table, keyed list, or associative array. **{ … }**
  - An **ordered list** of values. In most languages, this is realized as an ***array***, vector, list, or sequence. **[…]**

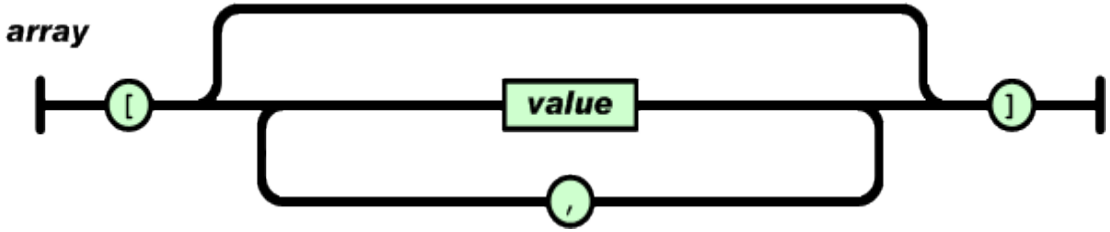
JSON – What does it look like?

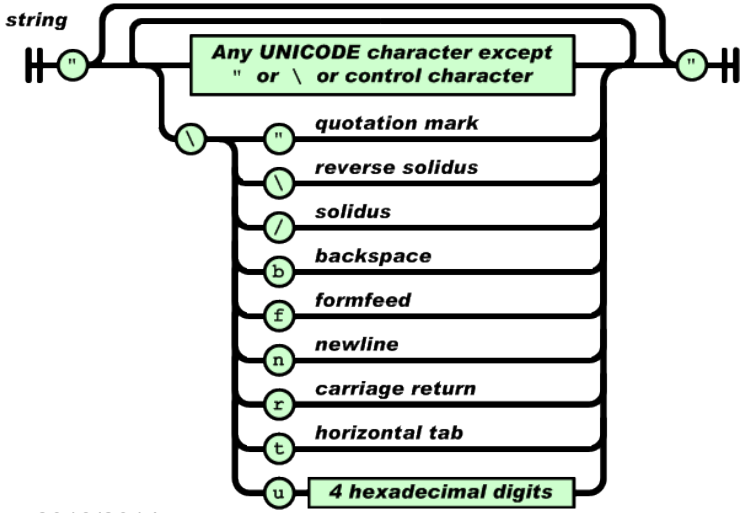
```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

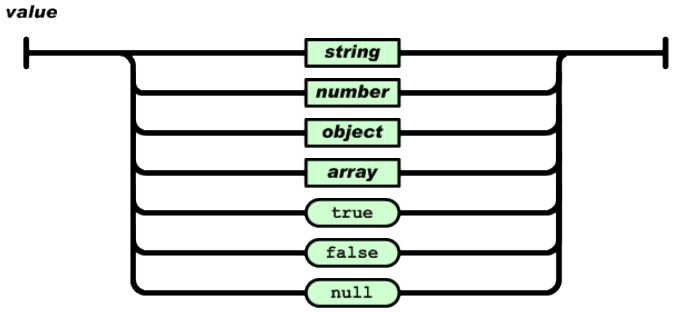


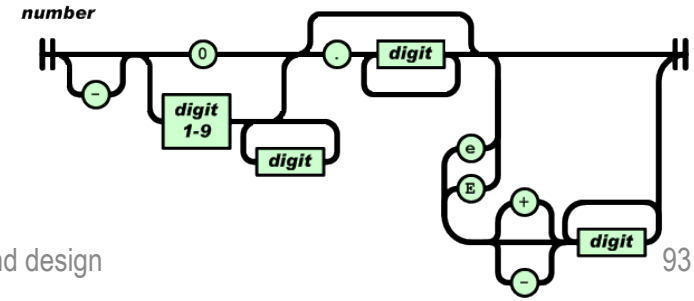
JSON Data Structures











Resources

- HTTP
  - http://www.w3.org/Protocols/
  - Hypertext Transfer Protocol -- HTTP/1.1: http://tools.ietf.org/html/rfc2616
- REST
  - http://en.wikipedia.org/wiki/Representational_state_transfer
  - R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
  - Learn REST: A Tutorial: http://rest.elkstein.org/
  - https://pages.apigee.com/ebook-web-api-design-registration.html
  - http://www.slideshare.net/apigee/api-design-3rd-edition
  - groups.google.com/group/api-craft

Resources

- JSON
  - http://json.org
  - ECMA-404 The JSON Data Interchange Standard. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

License

- These slides are distributed under a Creative Commons license “Attribution – NonCommercial – ShareAlike (CC BY-NC-SA) 3.0”
- You are free to:
  - Share — copy and redistribute the material in any medium or format
  - Adapt — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- Under the following terms:
  - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - NonCommercial — You may not use the material for commercial purposes.
  - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- http://creativecommons.org/licenses/by-nc-sa/3.0/