Linux Avanzato

# Architetture web

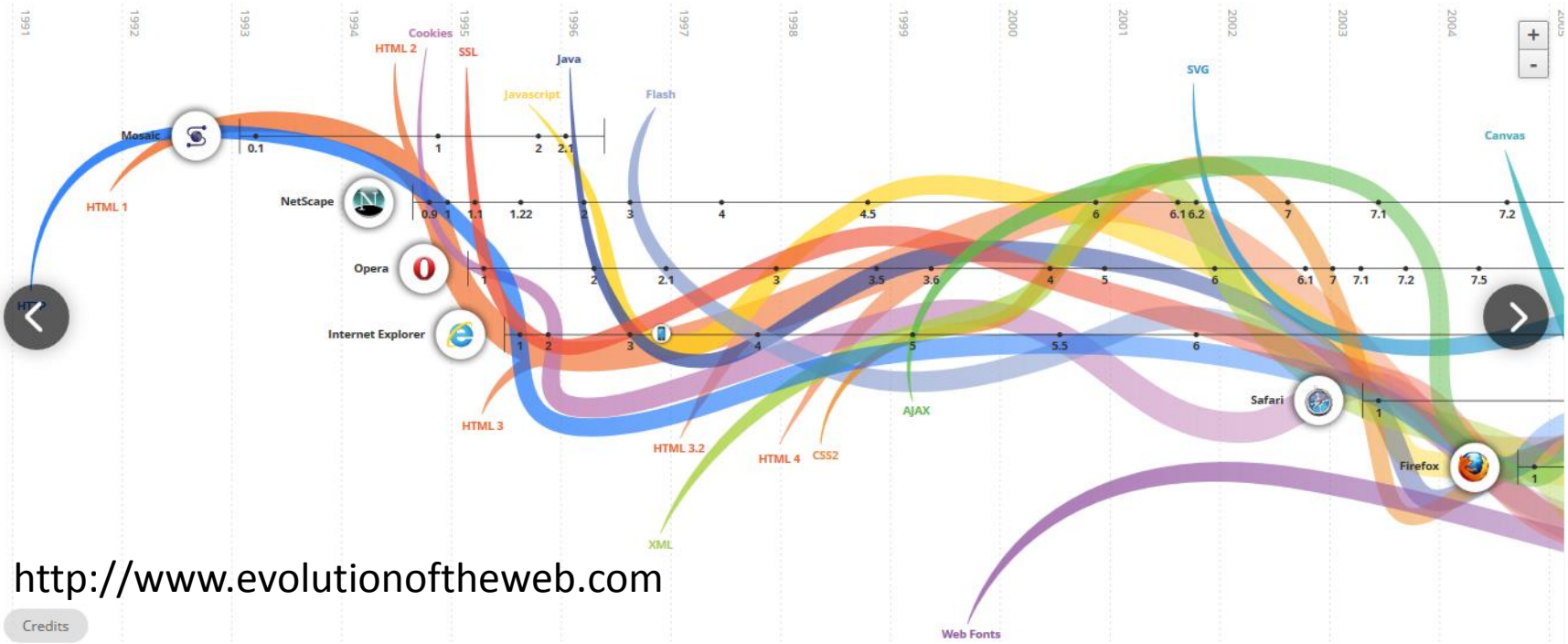
## Linguaggi e standard
## Web server, application server, database server

## ~~Il protocollo HTTP~~

# Argomenti

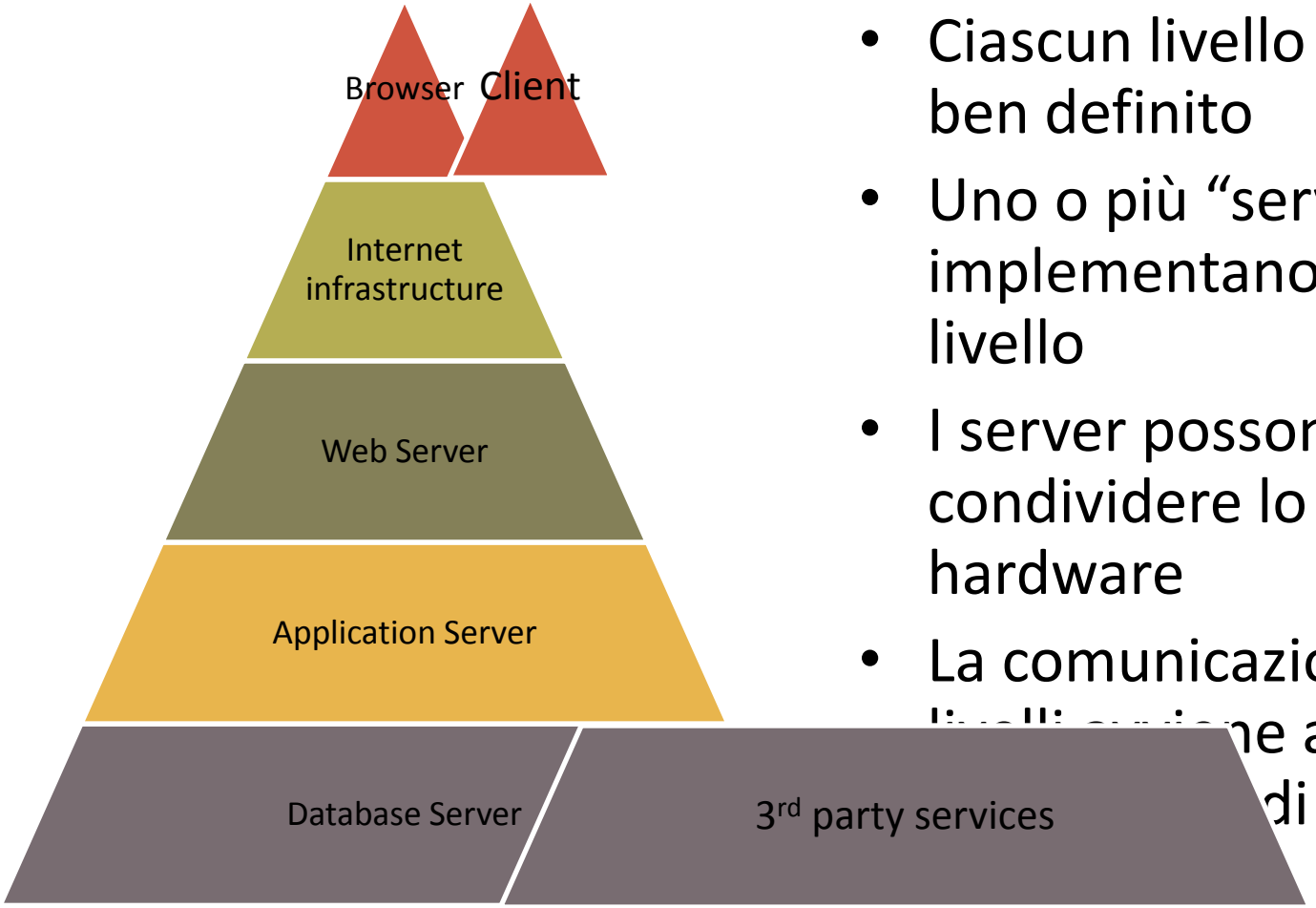
- Architetture web
- Livelli, server, standard, protocolli
- HTTP (Hypertext Transfer Protocol)

# Evoluzione delle architetture web



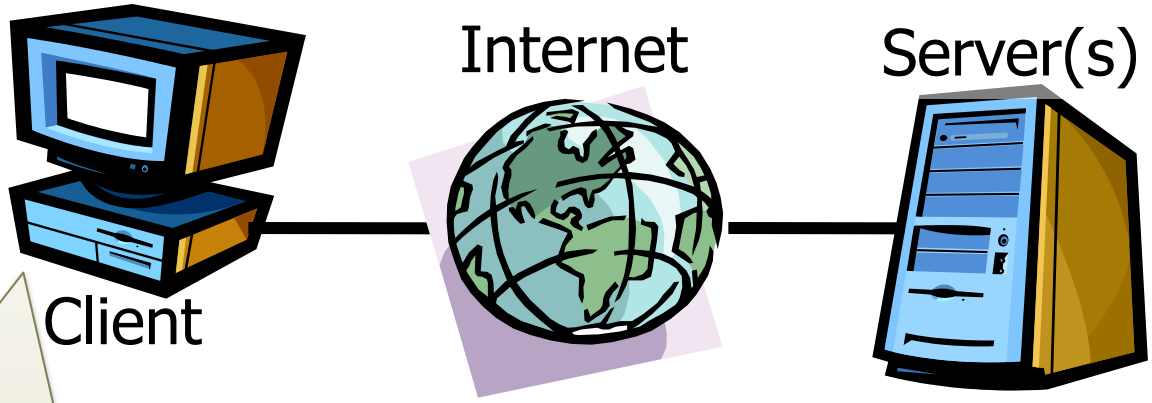

http://www.evolutionoftheweb.com

# Architettura ad N livelli




- Ciascun livello ha un ruolo ben definito
- Uno o più “server” implementano ciascun livello
- I server possono condividere lo stesso hardware
- La comunicazione tra i livelli avviene attraverso [illegible]di rete

# Architettura generale




- Storicamente, un browser web
- Ma anche:
  - Applicazione Mobile
  - Applicazione Desktop
  - Altra applicazione server

# Componenti

- Una o più connessioni ad Internet (attraverso un ISP)
- Uno o più server che implementino ciascun livello dell'architettura
- Una o più reti fisiche per interconnettere I server
- Uno o più apparati di rete (router, firewall, switch) per implementare le politiche di comunicazione e di sicurezza.

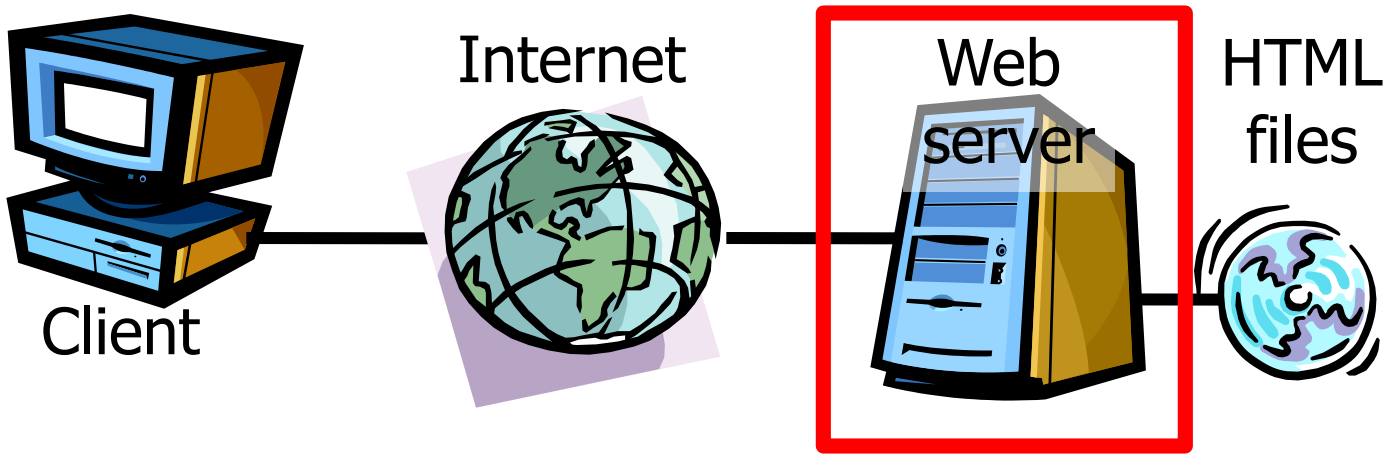
# Definitione

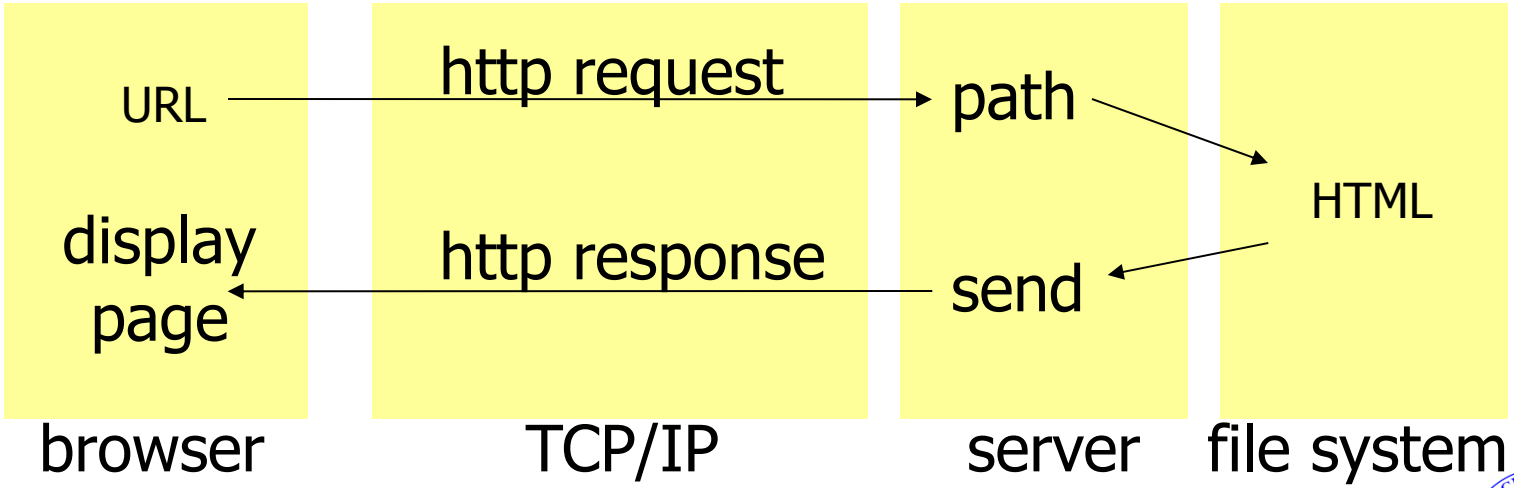
- “Server” si può definire come
  - Definizione logica:
    Un processo, in esecuzione su un computer host, che fornisce informazione ad un client al ricevimento di una richiesta.
  - Definizione fisica:
    Un computer host, collegato in rete, che contiene delle informazioni (es. Siti web) e che risponde alle richieste per tali informazioni

# Server Web

- Gestisce il protocollo HTTP (riceve le richieste e fornisce le risposte)
  - Riceve richieste dai client
  - Legge contenuti e pagine statiche dal filesystem
  - Attiva l'application server per le pagine dimaniche e per la generazione di contenuti dinamici server-side
  - Fornisce al client un file (HTML o altro) come risposta
- Una connessione HTTP per ciascuna richiesta
- Multi-process, Multi-threaded, Process pool

# Esempio


Internet
Web server
HTML files
Client

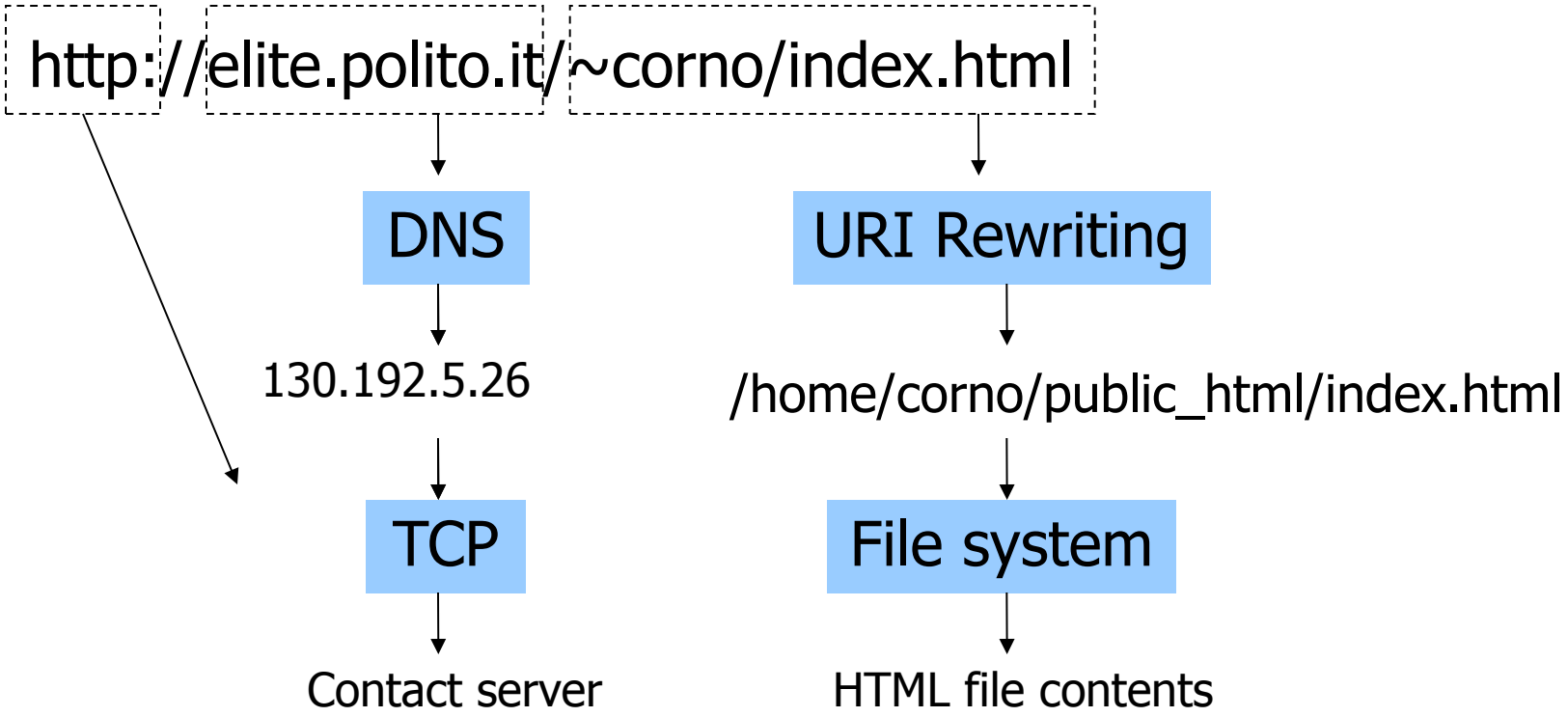

URL
http request
path
HTML
display page
http response
send
browser
TCP/IP
server
file system

# Standard adottati

- URL (uniform resource locator) per identificare le pagine web
- HTML (hyper text markup language) per costruire le pagine web
- GIF (graphics interchange format), PNG (portable network graphics), JPEG, … per le immagini
- HTTP (hyper text transfer protocol) per l'interazione tra client e server
- TCP/IP (transmission control protocol over internet protocol) per il trasferimento di dati

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

# URL



# URI Basics


- http://www.sadev.co.za/users/1/contact


- http://www.sadev.co.za?user=1&action=contact


- http://rob:pass@bbd.co.za:8044


- https://bbd.co.za/index.html#about

# Protocollo HTTP

GET /~corno/index.html HTTP/1.0
Accept: text/html
Accept: image/gif
User-Agent: Firefox/Windows Browser 18.3

HTTP/1.0 200 OK
Date: Monday, 01-Jan-2001 00:00:00 GMT
Server: Apache 1.3.0
MIME-Version: 1.0
Last-Modified: 31-Dec-2000
Content-type: text/html
Content-length: 3021

<HTML> . . .

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

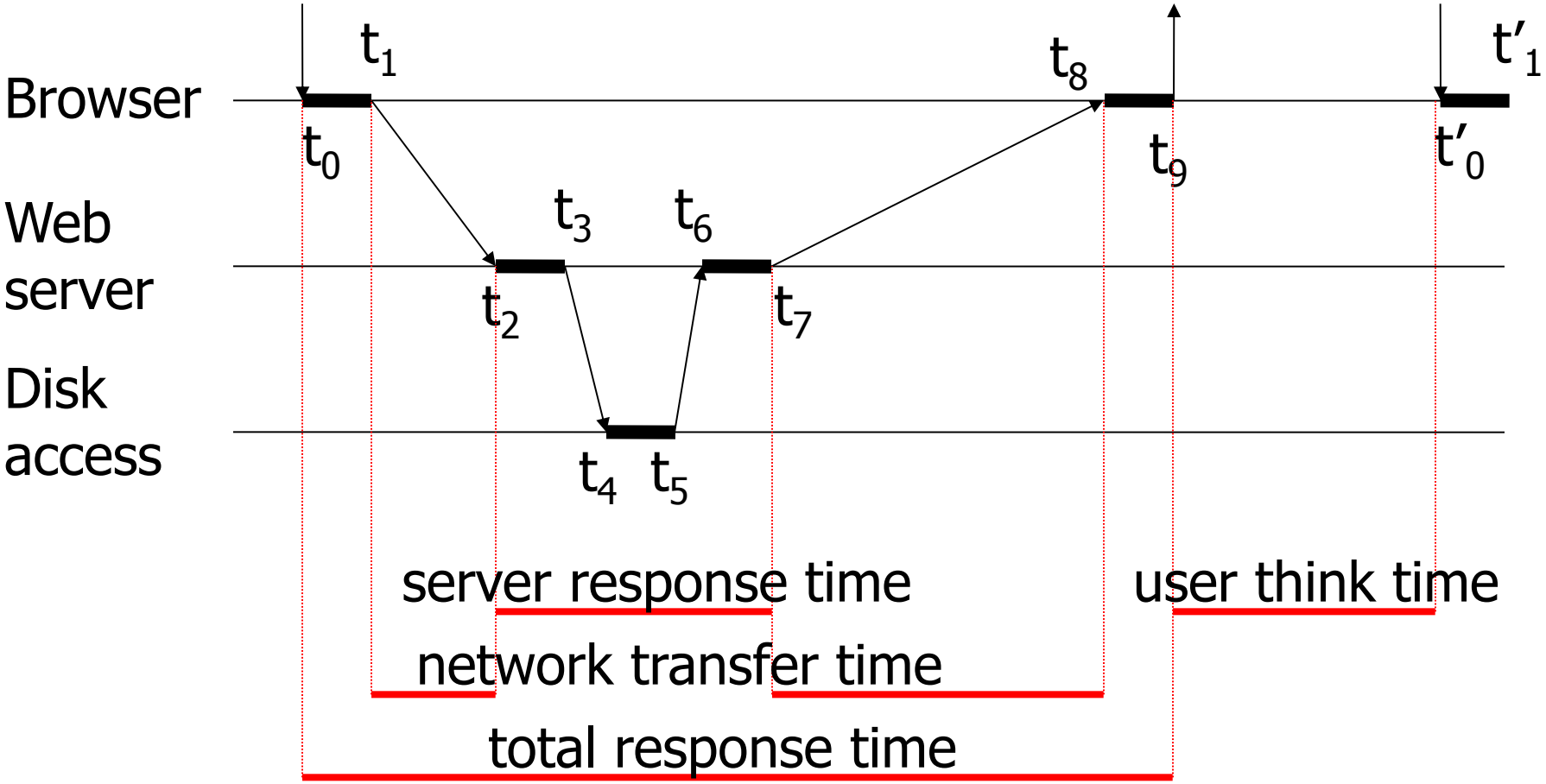
# Misure di prestazione

- Latenza: tempo necessario a fornire una risposta http contenente una pagina di 0 byte. Rappresenta il costo di elaborazione fisso di ciascuna pagina.
  - Misurata in: http/s o s/http
- Throughput: massima velocità a cui una pagna di lunghezza infinita può essere inviata.
  - Misurata in: byte/s, MB/s

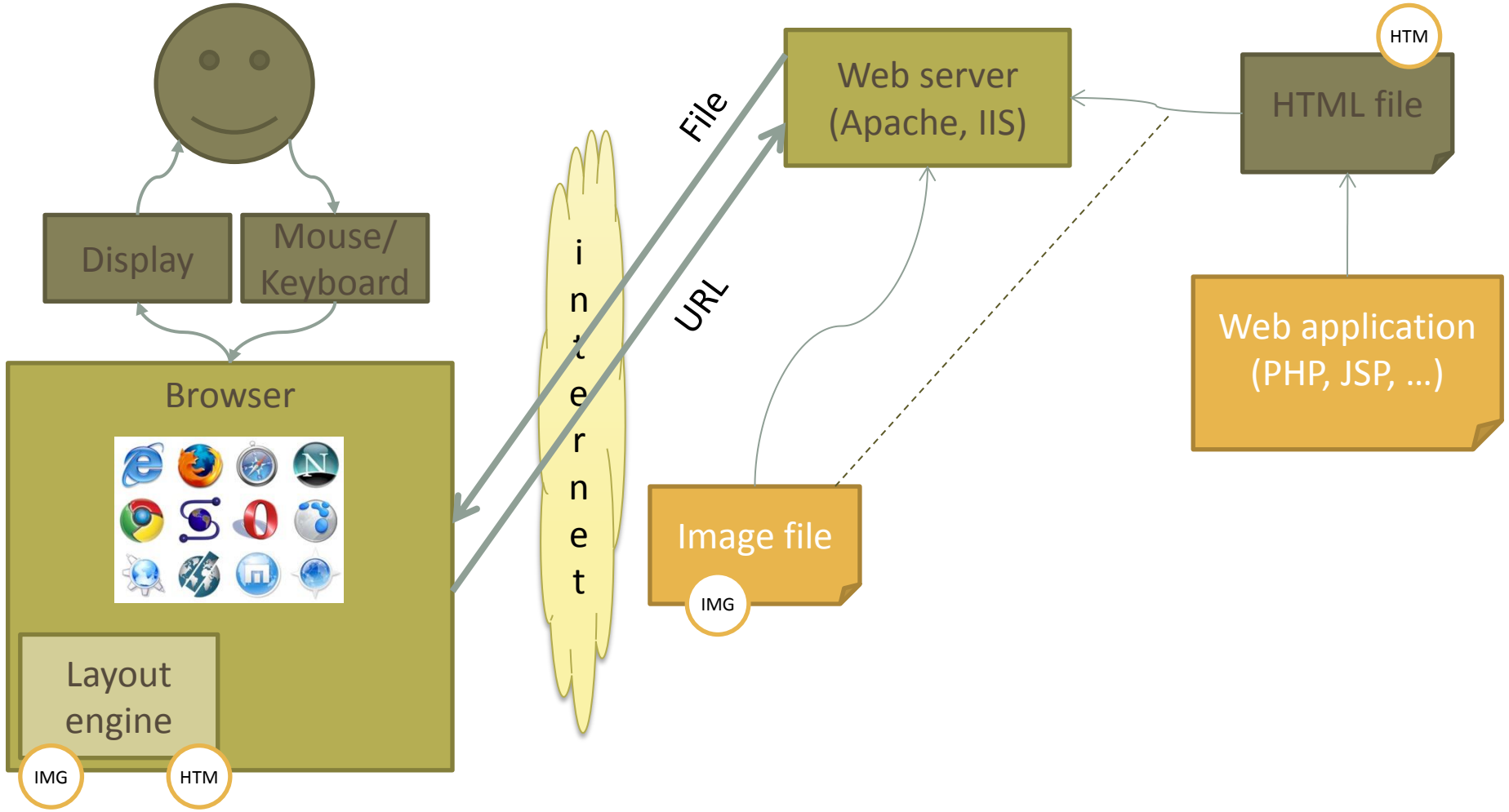
# Tempo di risposta

- T = Latency + ResponseBytes / Throughput
- Equazione valida se:
  - Gli altri elementi architetturali (I/O, reti, ...) non sono sovraccarichi
  - Il server web non ha ancora raggiunto il massimo carico sopprtabile
- Esempio:
  - Latency: 0,1s
  - ResponseBytes : 100kBytes
  - Throughput: 800kBytes/s
  - T= 0,1s+ 100KBytes / 800KBytes/s =0,225s

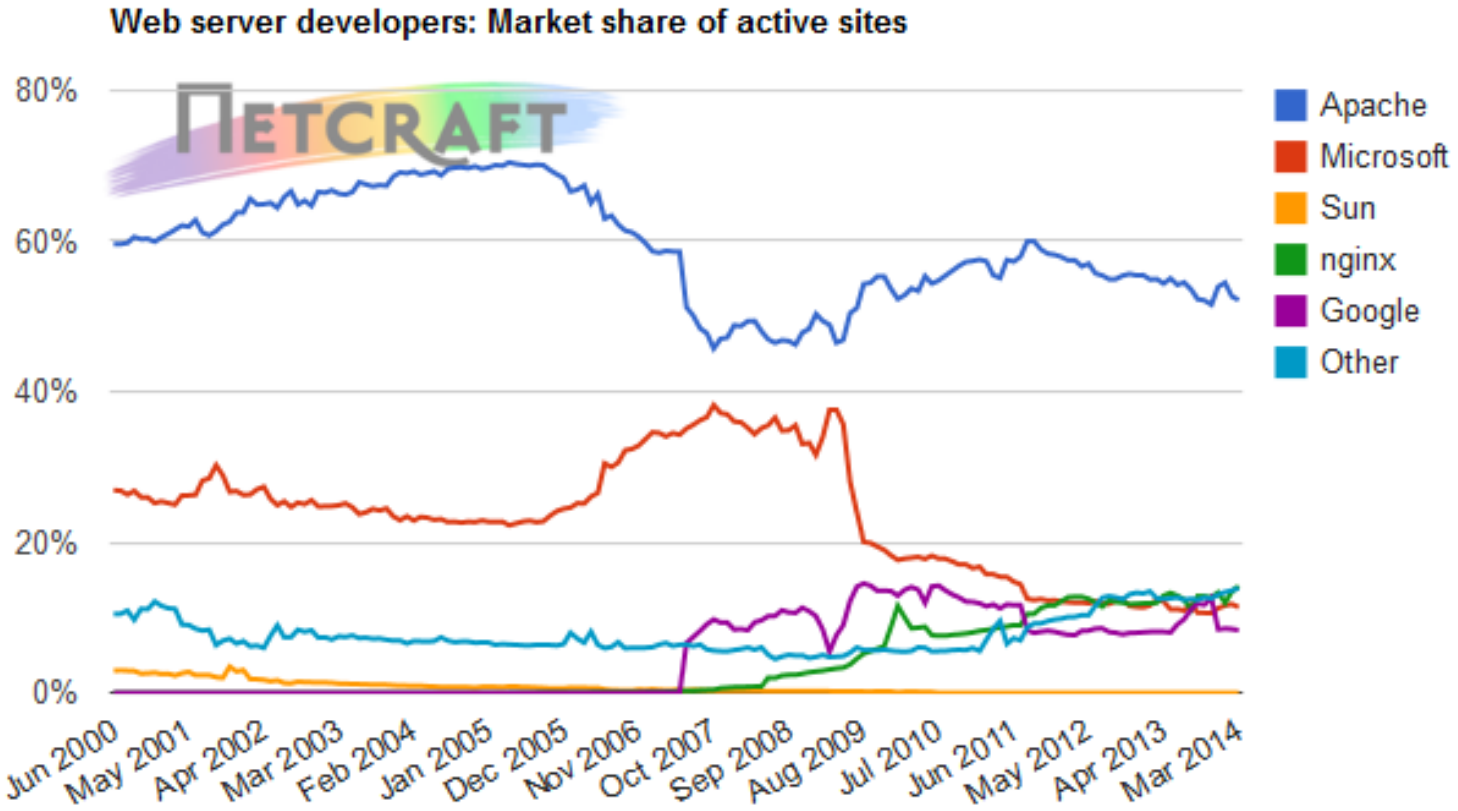
# Transazione web statica



# General Architecture



# The most adopted web servers



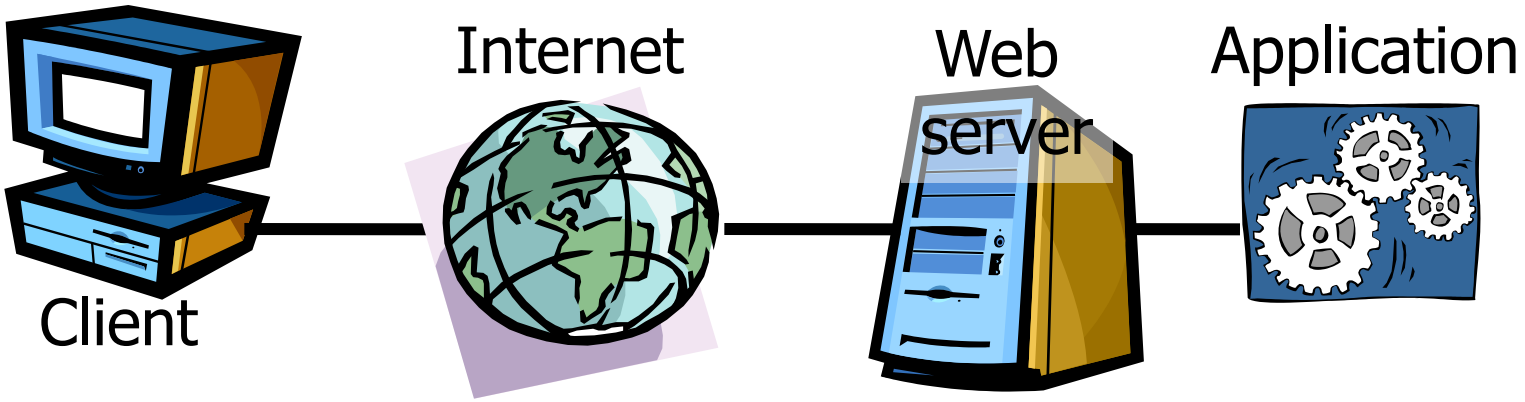

Source: http://news.netcraft.com/

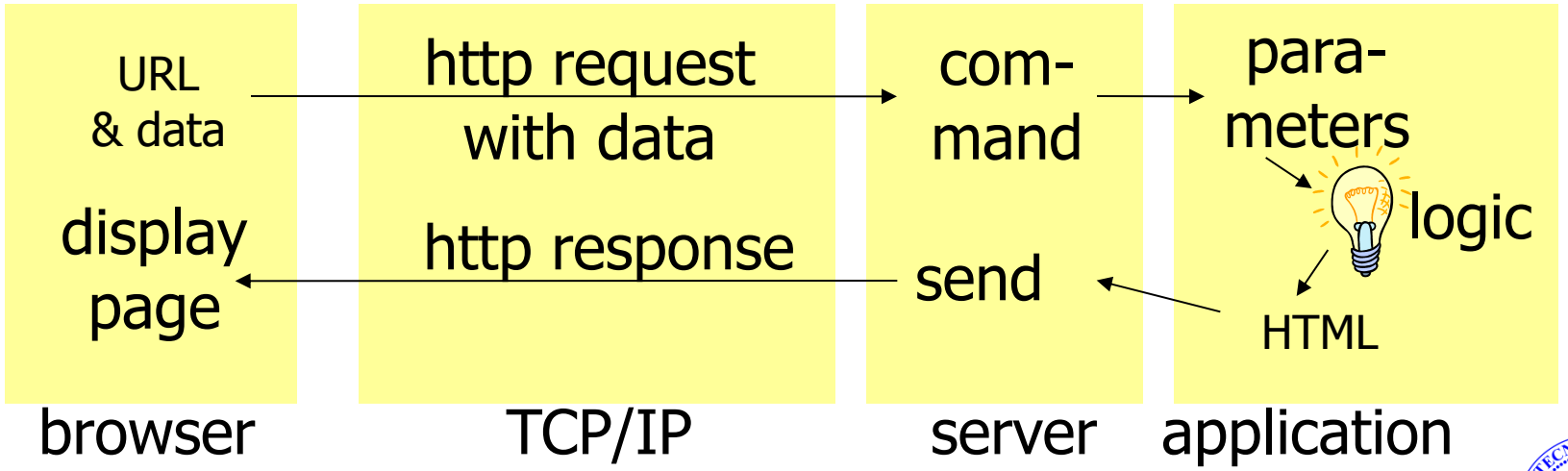
http://news.netcraft.com/archives/2014/03/03/march-2014-web-server-survey.html

# Application server

- Generazione delle pagine e dei contenuti dinamici
- Gestisce la logica operativa e funzionale (business logic) del sito
- Livello intermedio tra il front-end (web) ed il back-end (database)
- Implementa i meccanismi di sessione (gestione cookie)
- Realizzabile con diverse tecnologie, architetture e linguaggi di programmazione

# Transazione web dinamica

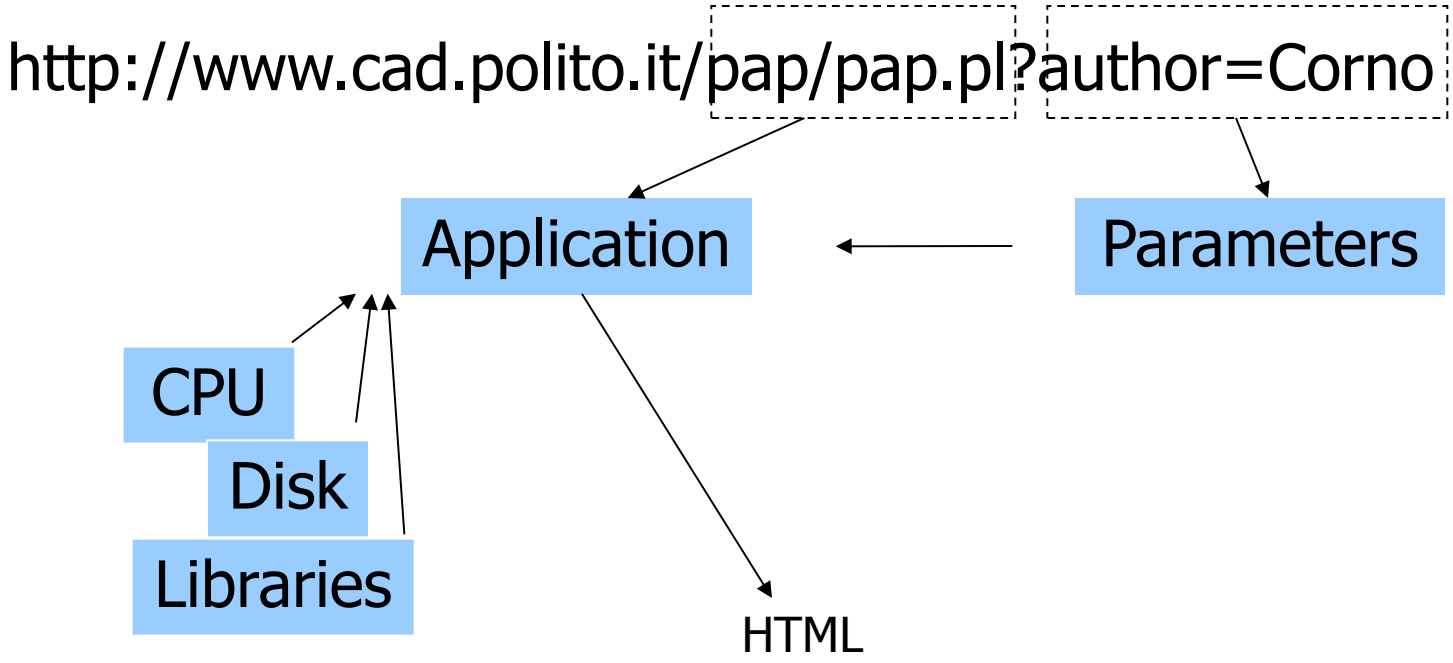


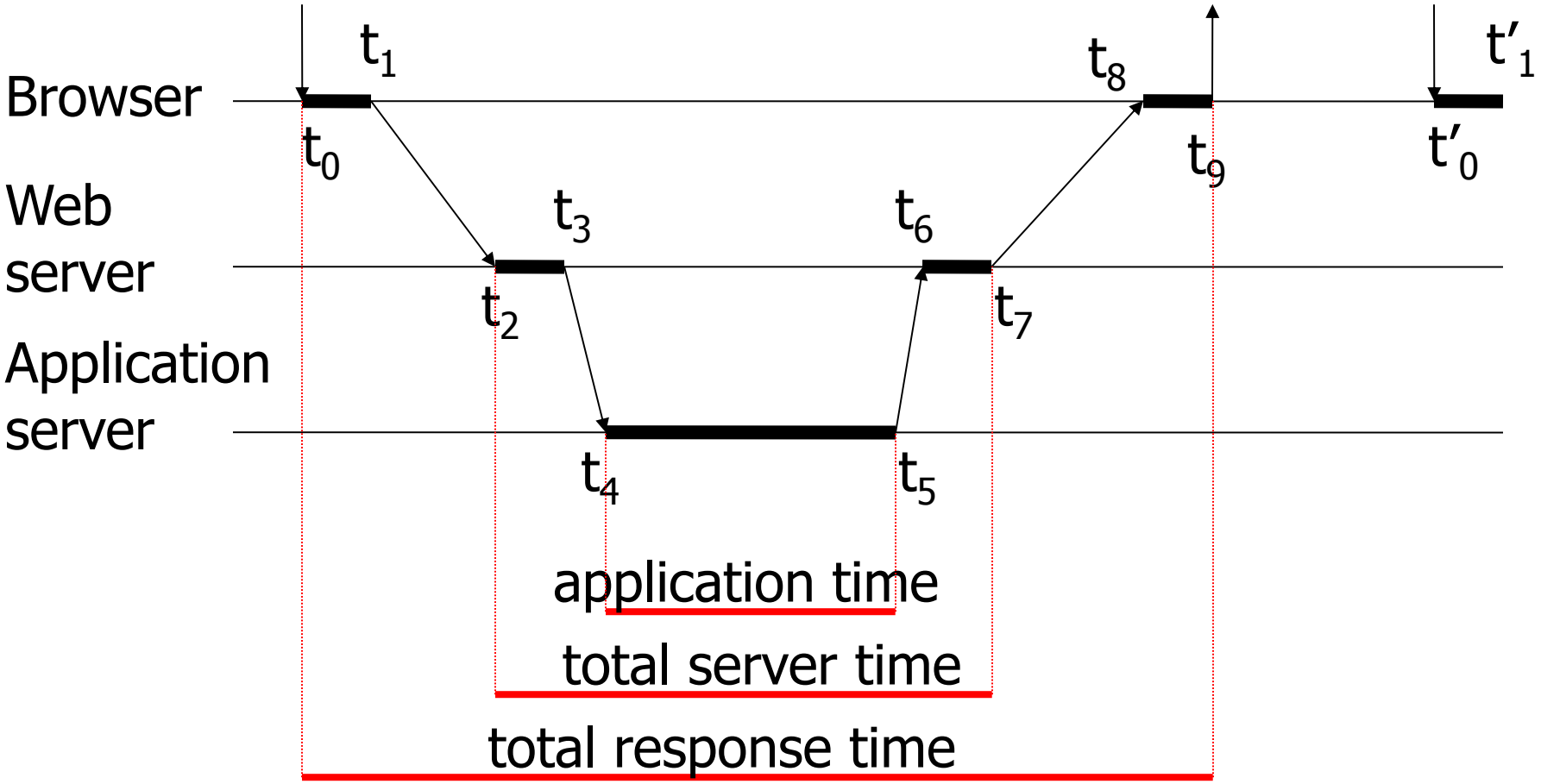
# Standard adottati

- Estensioni HTTP per inviare dati
  - HTTP-POST
  - URL-encoding in HTTP-GET requests
- Tecnologie per integrare codice applicativo nel server web
  - java servlets
  - ASP (active server pages), JSP, PHP, PERL, Python as new languages for application development
- Cookie per gestire lo stato di una sessione

# URL (HTTP GET)


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

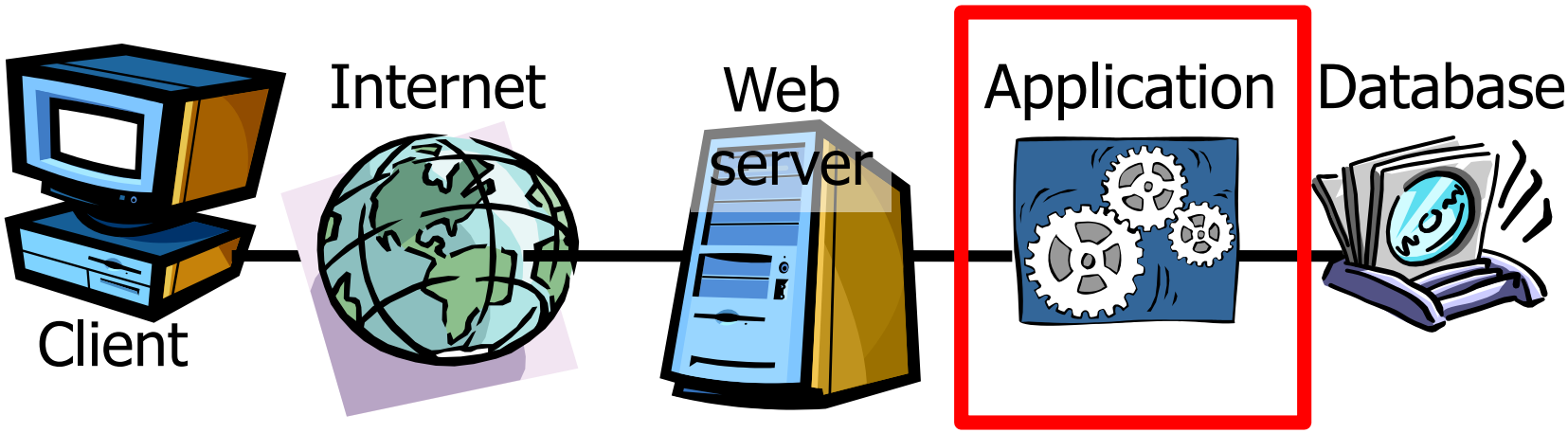
# Transazione web dinamica

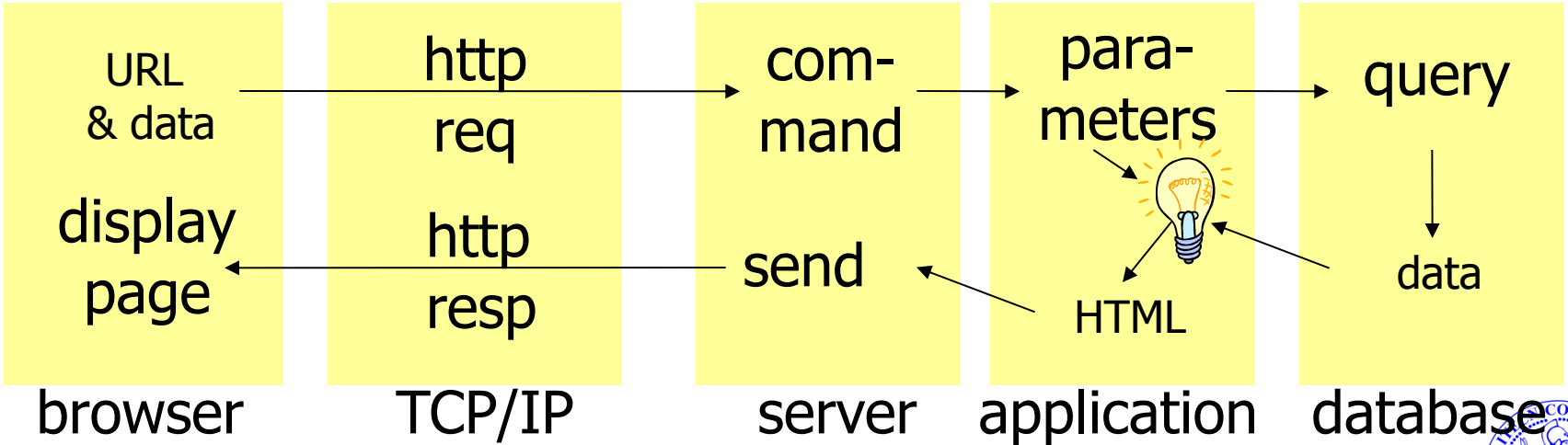


# Server database

- Memorizza i dati su cui lavora l'application server
- Esegue le interrogazioni (query) richieste dall'application server:
  - Aggiorna i dati memorizzati
  - Inserisce nuovi dati
  - Restituisce i risultati delle ricerche
- Le query più complesse o più frequenti possono essere implementate all'interno del database, per mezzo di «stored procedure» e/o query parametriche.

# Esempio


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

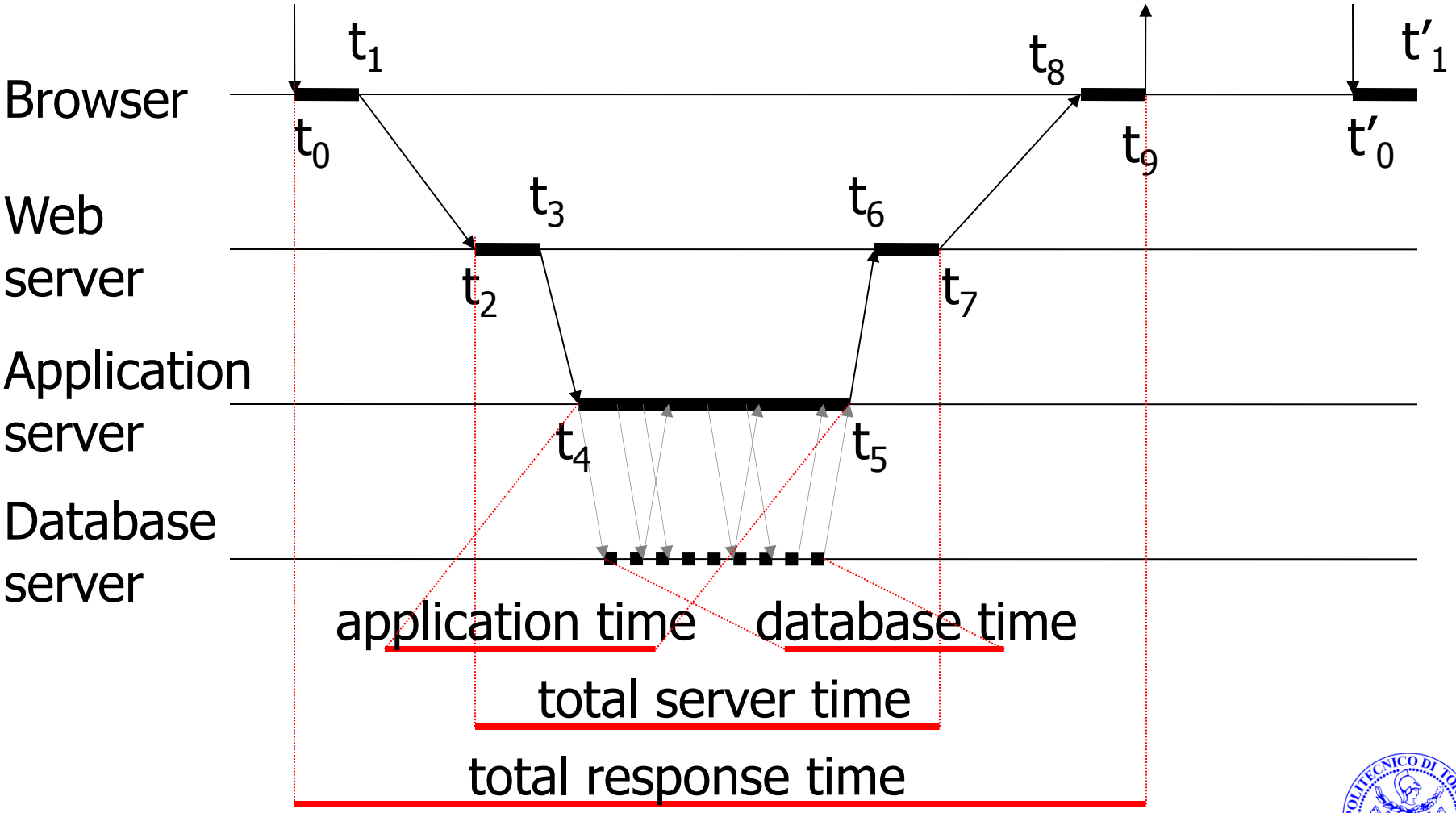
# Standard adottati

- Basi di dati relazionali (RDMBS)
- SQL (structured query language)
- JDBC/ODBC per accedere alle basi dati

# 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



# Esempio (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”];”;
```

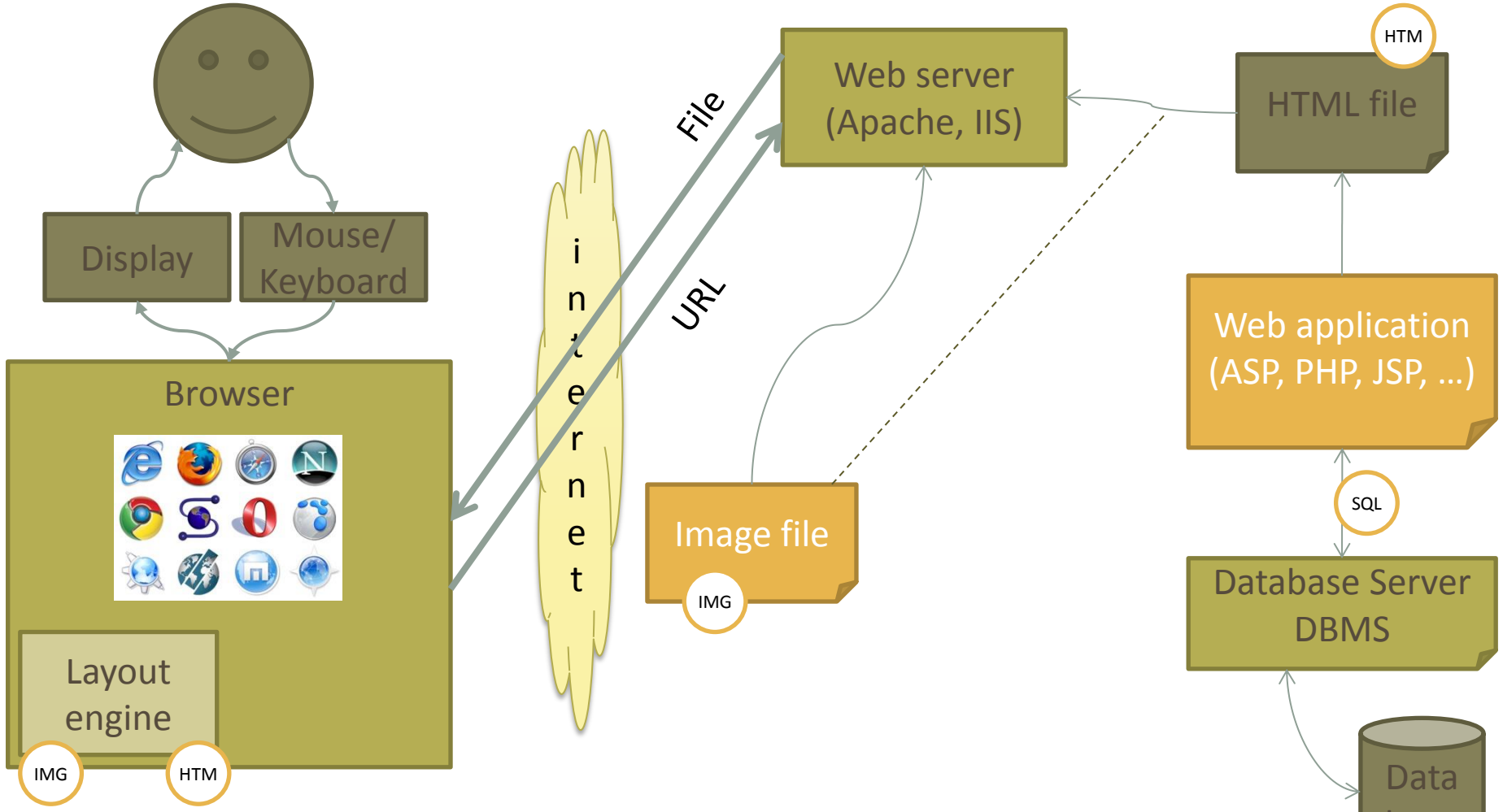
The query is sent to the db-server and a rowset containing the results is returned

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

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

The application elaborates the data

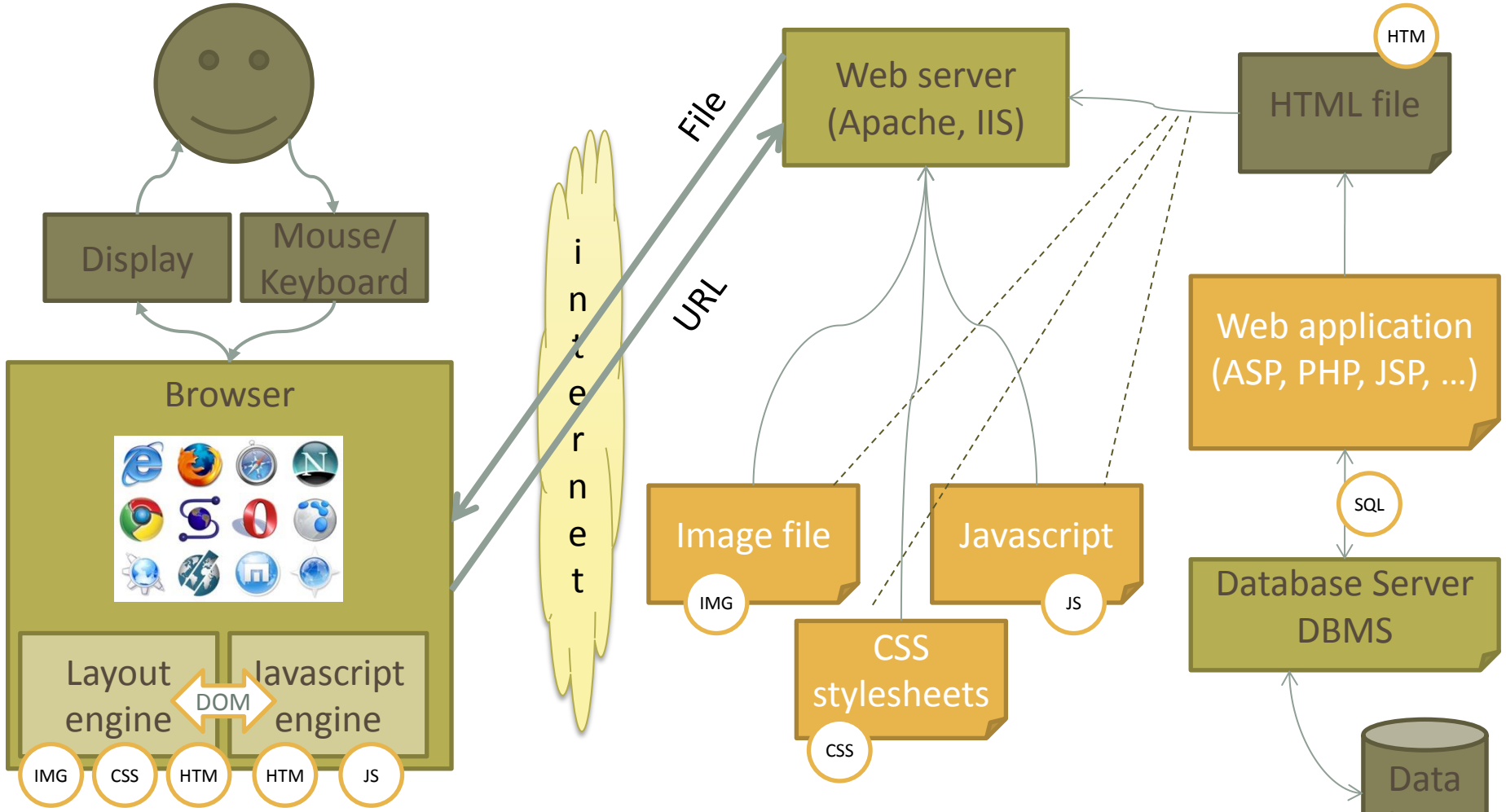
# General Architecture



# Interazione lato client

- Gli standard W3C regolano l'interazione client-server, ma gli utenti vogliono maggiore interazione sul client
- Nuovi standard e linguaggi permettono dei comportamenti dinamici all'interno di una pagina caricata, senza coinvolgere il server

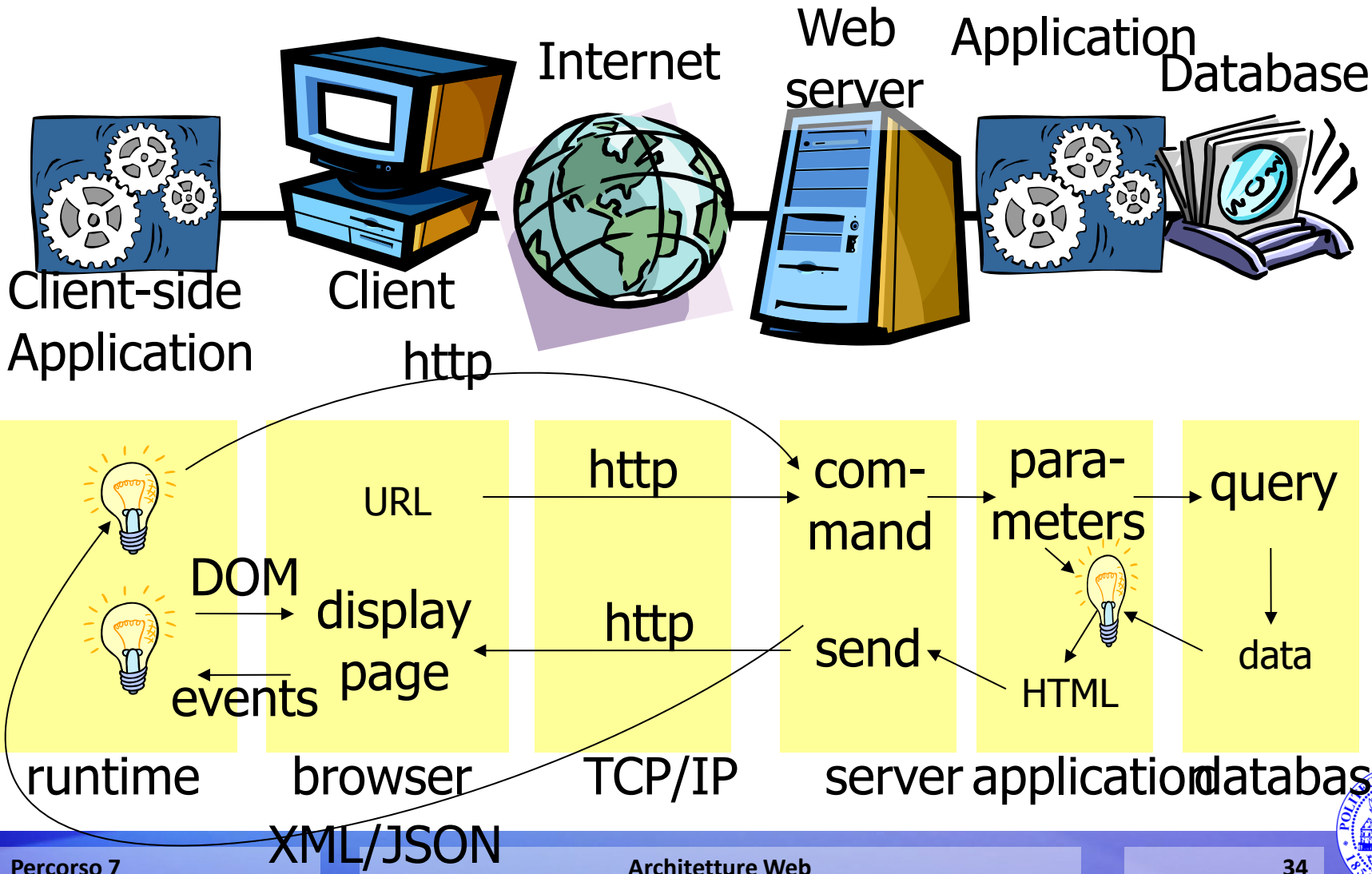
# General Architecture



# 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)

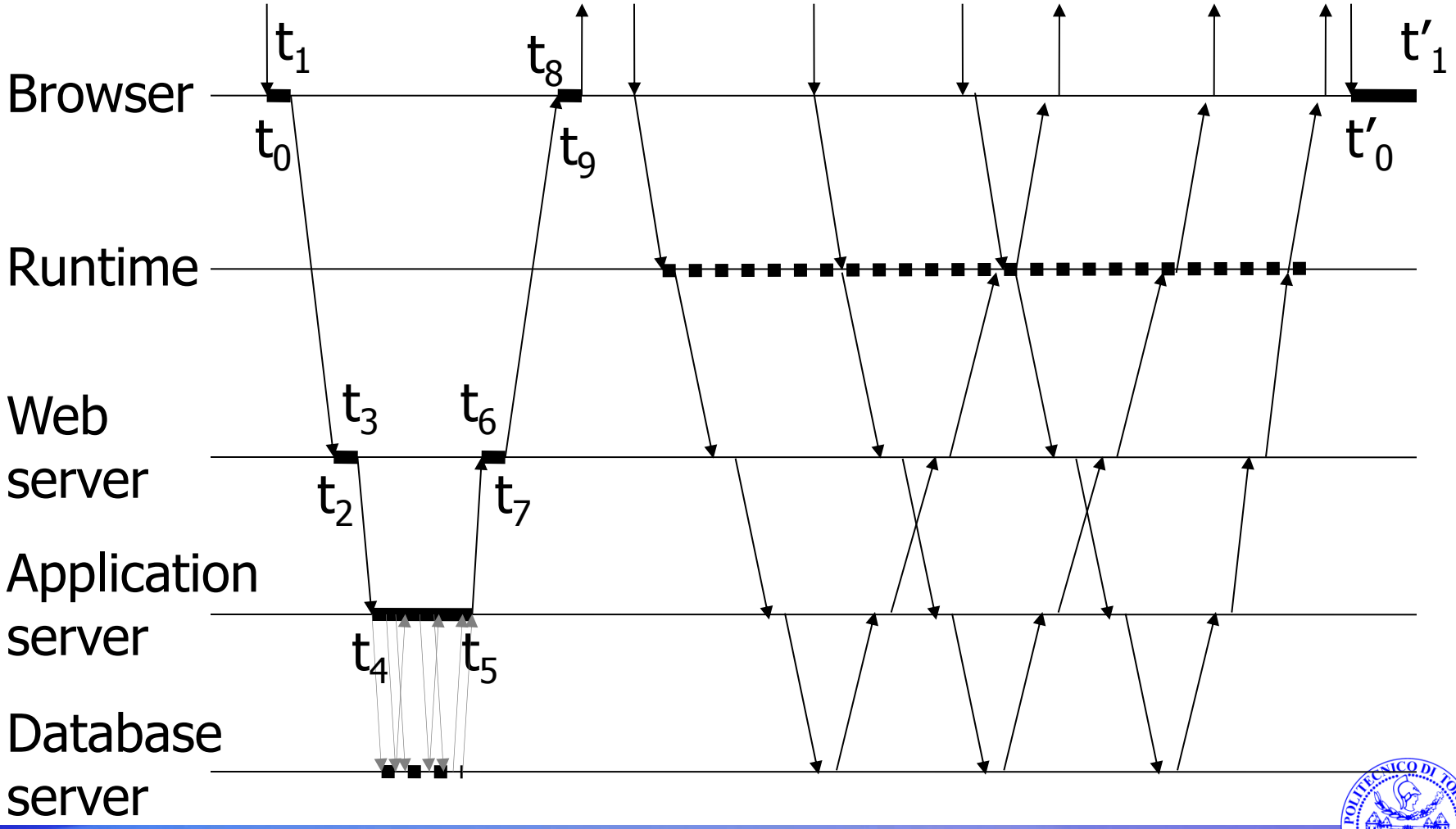
# Rich-Client Asynchronous Transactions



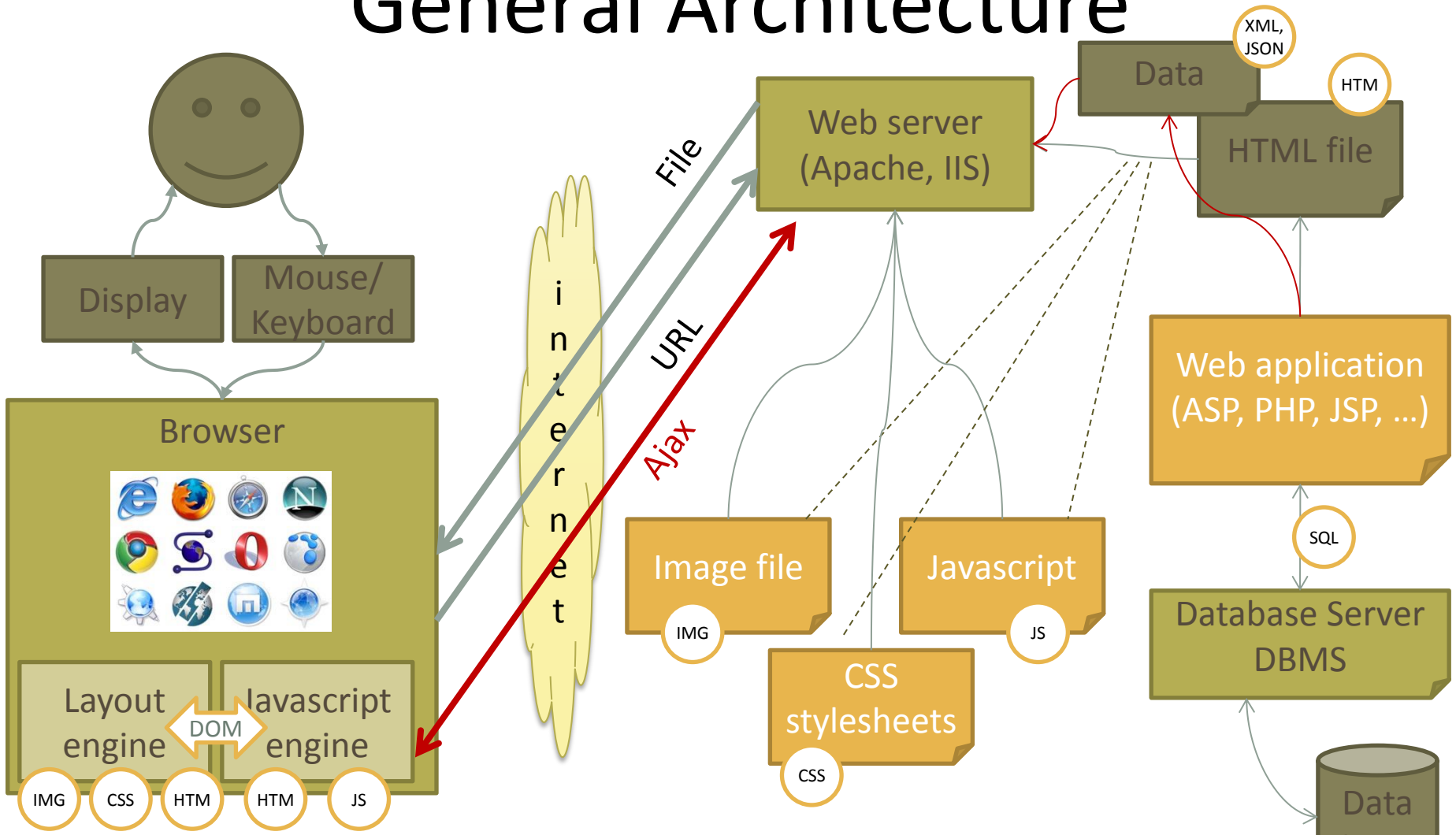
# Adopted standards

- 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

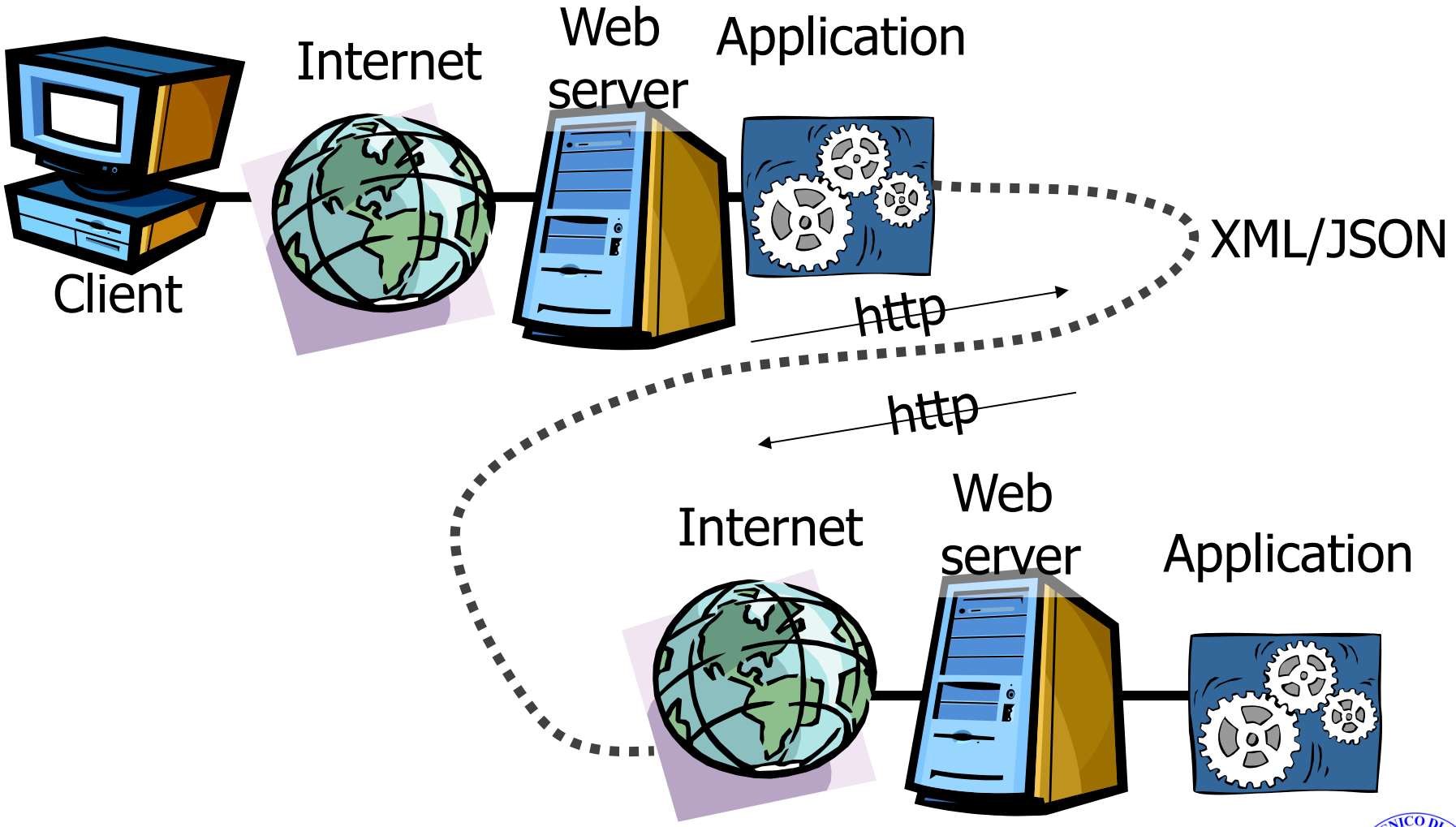


# General Architecture



# Architettura generale del web

# Distributed transactions



# Requisiti del mondo “reale”…

- The users
- Functionality
- Flexibility
- Portability
- Reliability
- Security
- Integrity
- Maintenance
- Performance
- Scalability

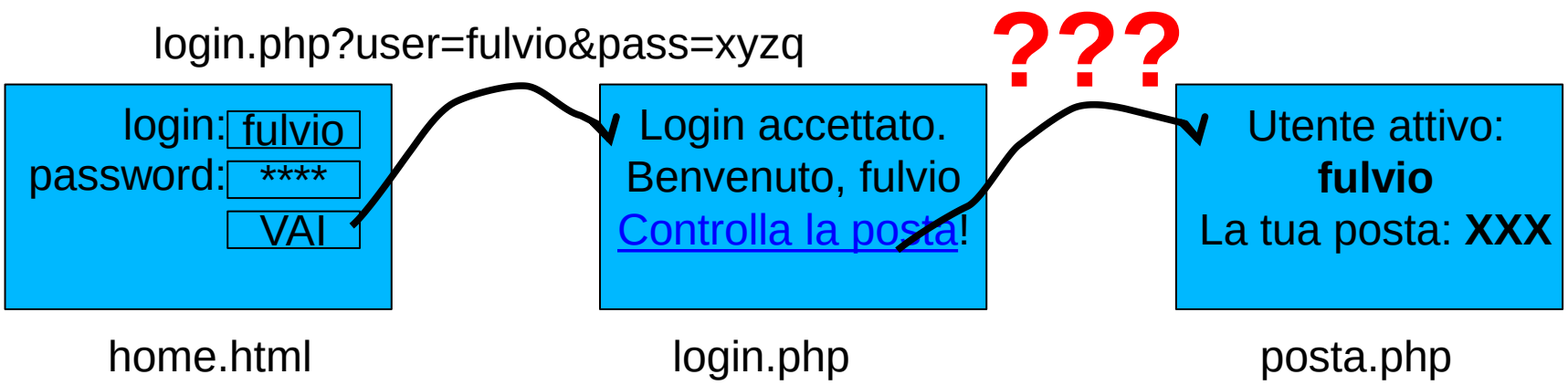
- Costs
- Maintenance
- Development times
- Interactions with existing systems
- Interactions with the “physical” world

# Il problema delle sessioni

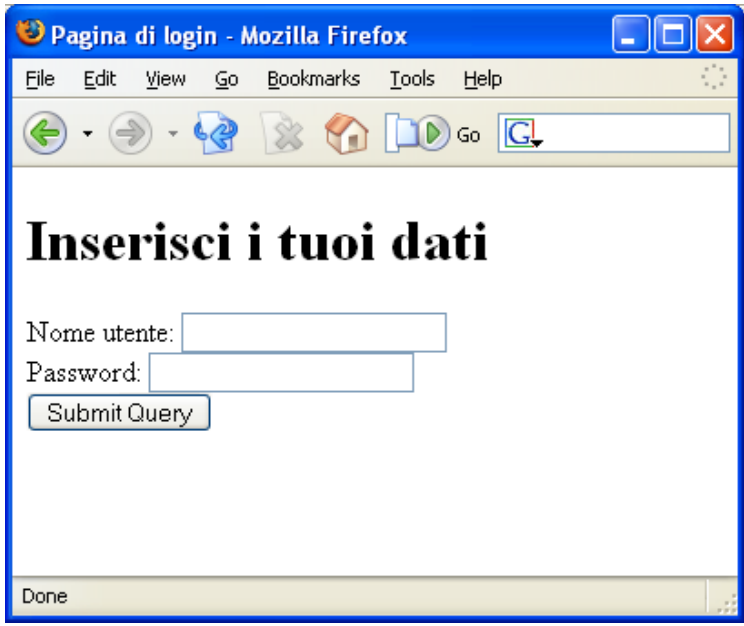
- Il protocollo HTTP è “senza memoria” (stateless)
- Ogni richiesta è indipendente dalle altre, e non può “ricordare” il percorso precedente di un utente
- Le variabili di input di ogni pagina possono dipendere solamente dai FORM nella pagina precedente
- Gli utenti, invece, vorrebbero avere l’impressione di una navigazione continua, in cui una scelta compiuta in una pagina venga “ricordata” in tutte le pagine successive.
- È necessario costruire una astrazione di una sessione di navigazione
- Sessione = sequenza di pagine, visitata sullo stesso sito dallo stesso utente con lo stesso browser nella stessa seduta di navigazione, che presentino una consequenzialità logica

# Esempio

- L'utente inserisce i propri username e password sulla prima pagina che visita
- La pagina successiva sarà necessariamente una pagina dinamica che "valida" il login
- Tutte le pagine successive dovranno conoscere il nome dell'utente



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>
<title>Pagina di login</title>
<meta http-equiv="content-type" content="text/html;
</head>
<body>

<h1>Inserisci i tuoi dati</h1>

<form method="GET" action="login.php">

Nome utente: <input type="text" name="user" /> <br />
Password: <input type="password" name="pass" /> <br />

<input type="submit" />

</form>

</body>
</html>
```



# Le soluzioni possibili

- URL rewriting
- Variabili nascoste nei FORM
- Cookie
- Session management

# Cookie

- Un cookie è una (piccola) quantità di informazione che il server può memorizzare sul browser
- Nella risposta http il server aggiunge un comando di intestazione “set cookie”
- Il browser si impegna a restituire al server il cookie ad ogni nuova richiesta http che farà
- Il cookie è composto da
  - Nome
  - Valore
  - Scadenza
  - Dominio
- Esempio: Nome: “user”, valore: “fulvio”, scadenza: “1200”, dominio: “miosito.it”

# Cookie

- Quando login.php valida la password, può settare un cookie sul browser
- La pagina posta.php può leggere il cookie, che il server riceverà nuovamente dal browser
- Accesso ai cookie da PHP:
  - Per scrivere: setcookie("nome", "valore")
    - bool setcookie ( string name [, string value [, int expire [, string path [, string domain [, int secure]]]]] )
    - Deve essere chiamata prima di ogni contenuto HTML (anche spazi!)
  - Per leggere: $_COOKIE["nome"]
  - Per verificare: if( isset($_COOKIE["nome"]) ) ...
  - Per cancellare: setcookie("nome", "")

# Pratica consigliata

- Tra le alternative viste, la più conveniente è l'utilizzo dei cookie, poiché:
  - È sufficiente fare una sola setcookie all'inizio
  - È possibile settare più cookie relative a diverse variabili
  - Non occorre modificare né i link esistenti né i form esistenti
- Svantaggi:
  - Non è possibile memorizzare informazioni voluminose
  - Non è "educato" memorizzare molti cookie diversi
  - Non è consigliabile fidarsi dei dati memorizzati sul browser dell'utente
  - È possibile che il browser rifiuti di restituire i cookie memorizzati

# Session management

- È possibile memorizzare più variabili con un solo cookie (breve):
- Il cookie memorizza un “numero magico”, diverso per ogni nuova sessione
- nome="session" value="123456"
- Il server memorizza, da qualche parte, le variabili di sessione associate al numero magico
- Session=123456 ➔ "user" = "fulvio"
- Può essere memorizzato nella memoria del web server, oppure in un file privato del server (es. session.123456.variables), oppure in una tabella di database (Session, Name, Value)
- Ad ogni nuova pagina, il cookie ricevuto può “sbloccare” le variabili opportune e renderle disponibili nella pagina.

# Session management in PHP

- Attivare la sessione:
  - session_start()
    - Nessun parametro, va chiamata prima di ogni altra cosa
    - Crea una nuova sessione o recupera quella esistente
    - Crea un cookie di nome simile a PHPSESSID123456
  - Il vettore $_SESSION["nome"] può essere usato per memorizzare le variabili associate con la sessione:
    - $_SESSION["user"] = "fulvio"
    - echo $_SESSION["user"]
    - If ( isset( $_SESSION["user"] ) ) ...
- Chiudere la sessione:
  - Scade automaticamente dopo 20-30 minuti
  - Scade automaticamente chiudendo il browser
  - Può essere “resettata” con session_destroy()

```php
<?php

    session_start() ;

    $user = $_REQUEST["user"] ;
    $pass = $_REQUEST["pass"] ;

    if($user == "fulvio" && $pass ==
"xyz")
    {
        // login OK
        $_SESSION["user"] = $user ;
        $login = 1 ;
    }
    else
    {
        // login errato
        $login = 0 ;
    }
?>
```

```php
<html>
<head>
<title>Controllo login</title>
</head>
<body>
<?php

if($login == 1)
{

  echo "<p>Benvenuto, $user</p>\n";
  echo "<p>Controlla la tua <a
href=\"posta.php\">posta</a></p>\n" ;
}
else
{
  echo "<p>ERRORE: login non valido, <a
href=\"home.html\">riprova</a></p>\n" ;
}
//echo "<p>Password: $pass</p>\n" ;



?>
</body>
</html>
```

# Esempio: posta.php

```
<?php
session_start() ;
?>

<html>
<head>
<title>La tua posta</title>
</head>
<body>
<?
   if(isset($_SESSION["user"]))
   {
        $user = $_SESSION["user"] ;
        echo "<p>Ecco la tua posta, $user</p>\n" ;
        /*  stampa la posta... */
   }
   else
   {
        echo "<p>ERRORE: devi prima fare il <a
href=\"home.html\">login</a></p>\n" ;
   }
?>
</body>
</html>
```

Web Architecture and Technologies

# HTTP

# Hypertext Transfer Protc

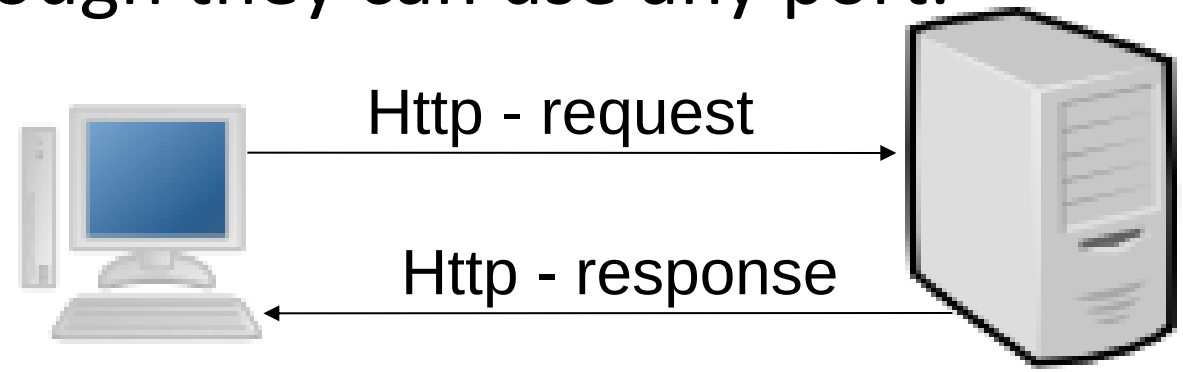
# What is HTTP?

- 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 (English-oriented). They consist of:
  - 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

Request

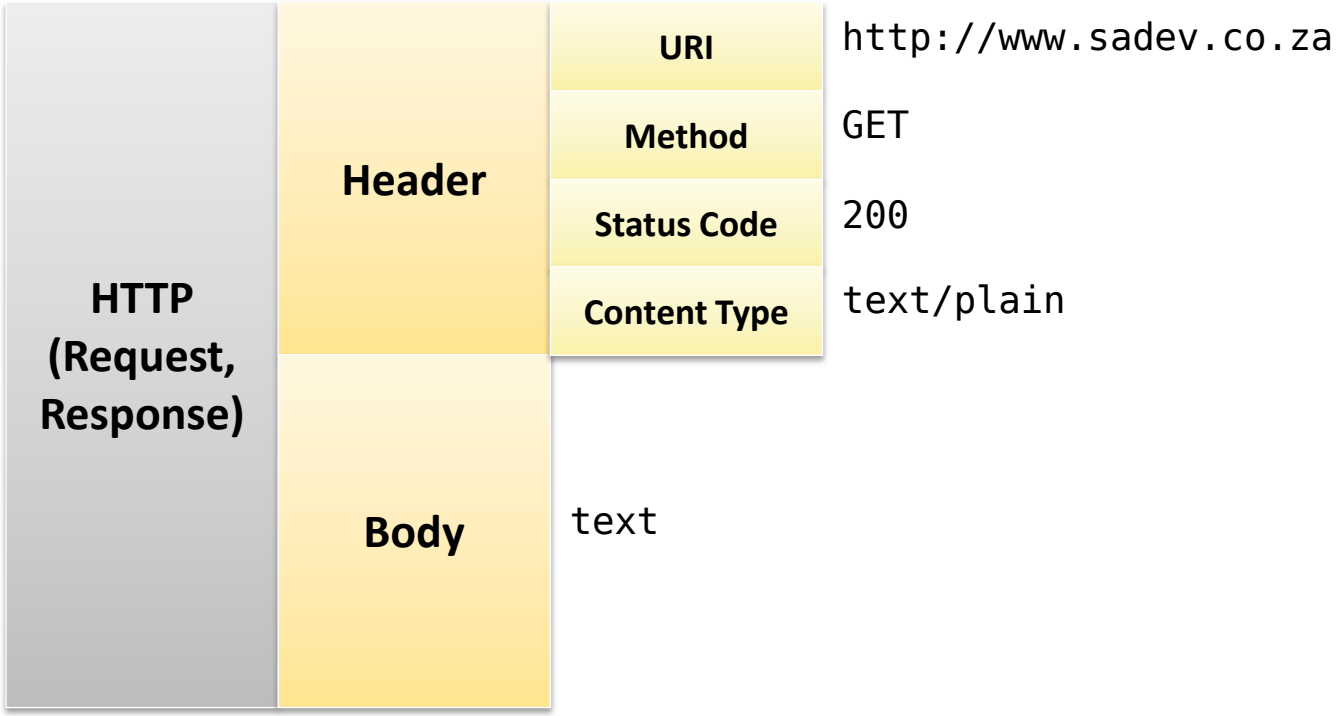
- HEAD /index.html HTTP/1.1
- Host: www.example.com

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 Basics

- HTTP is not HTML
- HTTP is stateless



# HTTP msg – 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 msg – 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 msg – 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 msg – 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 header

- 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 Header

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

# General 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.

# 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.

# 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, ...)
  - ....

- These slides are licensed under a Creative Commons
- Attribution
  Non Commercial
  Share Alike
  4.0 International
- To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/
- Versione in Italiano:
- https://creativecommons.org/licenses/by-nc-sa/4.0/deed.it