



CORSO DI LAUREA IN INFORMATICA

# TECNOLOGIE SOFTWARE PER IL WEB

Overview: URI, HTTP


a.a 2020-2021



DI COSA PARLIAMO OGGI

**WWW = URL + HTTP + HTML**

Cosa succede quando clicchiamo su un link  
presente in una pagina all'interno di un  
browser?



# URL

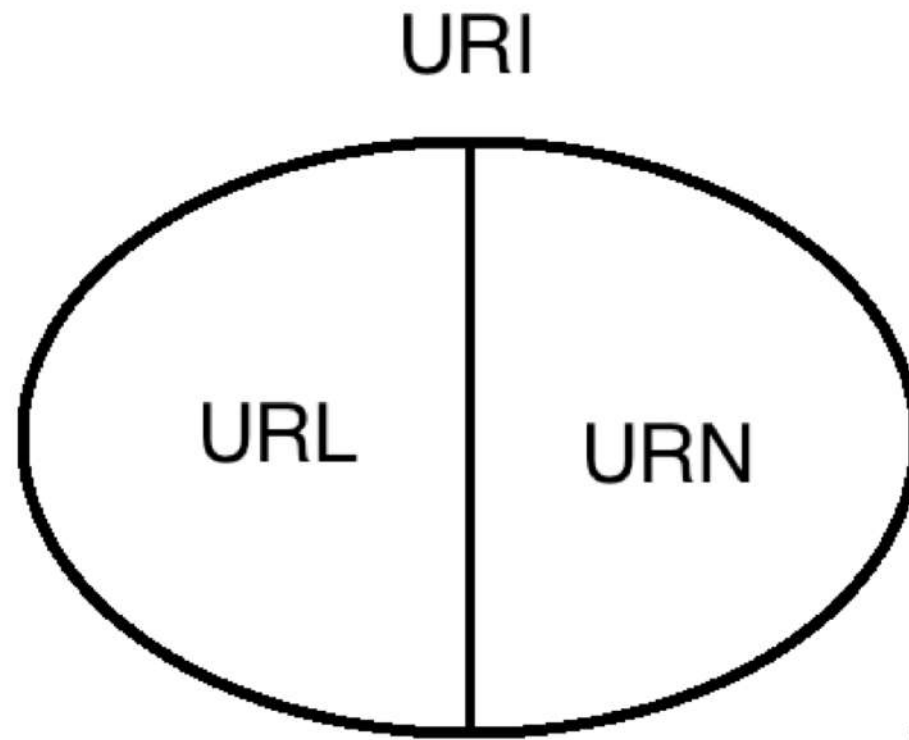
- Ogni risorsa sul web ha un **indirizzo unico**, nel formato URL (Uniform Resource **Locators**)
  1. *Come identifichiamo il **server** in grado di fornirci un elemento dell'ipertesto (una pagina o una risorsa all'interno della pagina)?*
  2. *Come identifichiamo la **risorsa** (elemento dell'ipertesto) a cui vogliamo accedere?*
- **Gli URL sono un particolare tipo di URI (Uniform Resource Identifier)**

# URI

- Gli **URI** (*Uniform Resource Identifier*) forniscono un meccanismo semplice ed estensibile per identificare una risorsa. Un URI è una stringa di caratteri.
- Con il termine **risorsa** intendiamo qualunque entità abbia una **identità**: un documento, un'immagine, un servizio, una collezione di altre risorse
- Caratteristiche di un URI:
  - È un concetto generale: non fa riferimento necessariamente a risorse accessibili tramite HTTP o ad entità disponibili in rete (quando lo fa, diventa URL)
  - È un mapping concettuale ad una entità: non si riferisce necessariamente ad una particolare versione dell'entità esistente in un dato momento
    - Il mapping può rimanere inalterato anche se cambia il contenuto della risorsa

# URN E URL

- Esistono due specializzazioni del concetto di URI:
  - **Uniform Resource Name (URN)**: identifica una risorsa per mezzo di un “nome” che deve essere **globalmente unico** e restare valido anche se la risorsa diventa non disponibile o cessa di esistere
  - **Uniform Resource Locator (URL)**: identifica una risorsa per mezzo del suo meccanismo di **accesso** primario (es. locazione nella rete) piuttosto che sulla base del suo nome o dei suoi attributi

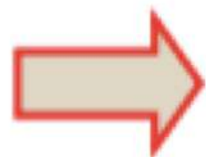


Applicando questi concetti ad una persona:

- URN è come identificazione basata su nome+cognome, o meglio codice fiscale (la persona potrebbe essere ovunque, ed il suo URN non cambia)
- URL è come indirizzo di casa o numero di telefono (se univoci). L'identificativo si basa sulla locazione della persona.

# URN

- Un URN identifica una risorsa mediante un nome in un particolare dominio di nomi (**namespace**)
  - Deve essere unico e duraturo
  - Consente di “parlare” di una risorsa **prescindendo** dalla **sua ubicazione** e dalle **modalità con cui è possibile accedervi**
  - Un esempio molto noto è il codice ISBN (International Standard Book Number) che identifica a livello internazionale in modo univoco e duraturo un libro o una edizione di un libro di un determinato editore
  - Non ci dice nulla su come e dove procurarci il libro



schema

urn:isbn:0-9553010-9

Nome univoco nel namespace

# ESEMPIO DI URL CON SCHEMA HTTP

Indirizzo IP del server in cui la pagina è memorizzata

Porta di ascolto del server

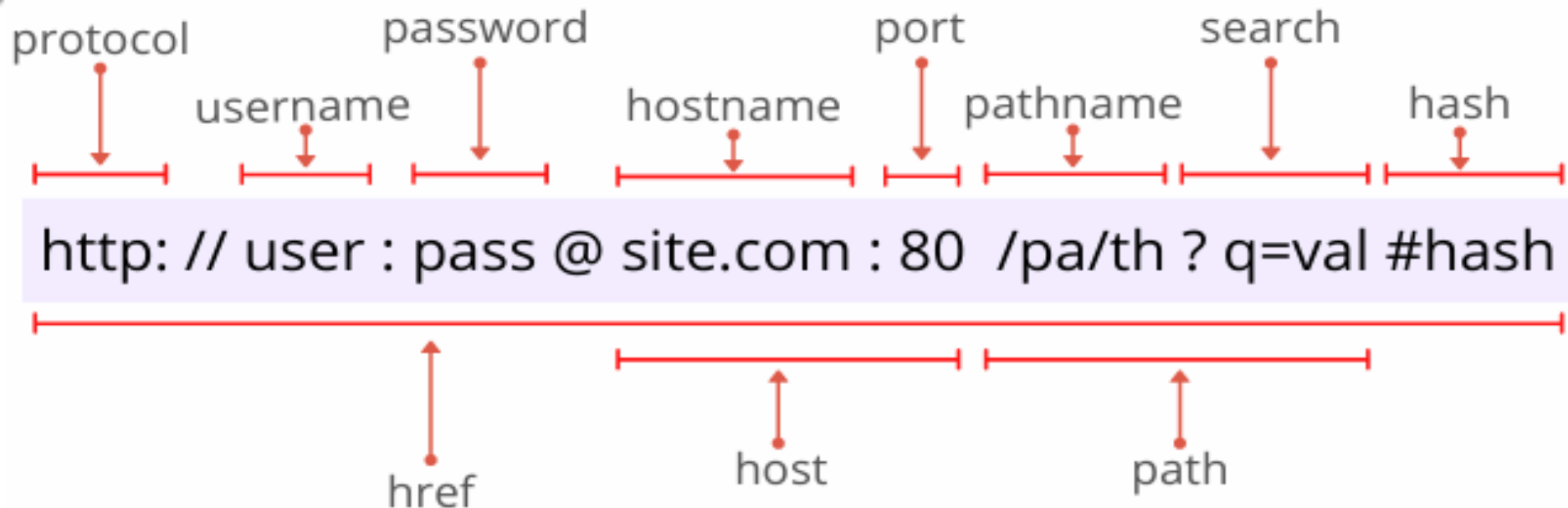
**http://lia.deis.unibo.it:8080/Courses/index.html**

Schema = protocollo di comunicazione con il server. Il protocollo http è il default per i servizi Web

Path della risorsa richiesta (file index.html) nel file system del server



# STRUTTURA DI UN URL



Per l'http sono obbligatori il protocol e l'hostname, tutti gli altri possono essere assenti.  
Ad esempio, `http://unisa.it` è un URL corretto.

Altri esempi,

`https://it.wikipedia.org/wiki/Napoli#Clima`

`https://www.google.com/search?q=form&safe=off`

`http://localhost:8080/esercizio1/login.jsp`

Questa forma vale per diversi protocolli di uso comune: HTTP, HTTPS, FTP, WAP, ..

# COMPONENTI DI UN URL CON SCHEMA HTTP-LIKE

- **<protocol>**: Descrive il protocollo da utilizzare per l'accesso al server (HTTP, HTTPS, FTP, MMS, ...)
- **<username>:<password>@**: credenziali per l'autenticazione
- **<host>**: indirizzo server su cui risiede la risorsa. Può essere un indirizzo IP logico o fisico (*www.unisa.it* o *193.205.160.20*)
- **<port>**: definisce la porta da utilizzare (TCP come protocollo di trasporto per HTTP). Se non viene indicata, si usa porta standard per il protocollo specificato (per HTTP è 80)
- **<path>**: percorso (pathname) che identifica la risorsa nel file system del server. Se manca, tipicamente si accede alla risorsa predefinita (es. home page → *index.html*)
- **<query>**: una stringa di caratteri che consente di passare al server uno o più parametri. Di solito ha questo formato:

**parametro1=valore&parametro2=valore2...**

- **Fragment o hash**: è una breve stringa di caratteri che si riferisce a una risorsa che è subordinata a un'altra risorsa primaria.

# RIFERIMENTI BIBLIOGRAFICI

- RFC2396, “Uniform Resource Identifiers (URI): Generic Syntax”,  
<http://www.ietf.org/rfc/rfc2396.txt>
- RFC1738, “Uniform Resource Locators (URL)”,  
<http://www.ietf.org/rfc/rfc1738.txt>

# HTTP

WWW = URL + HTTP + HTML

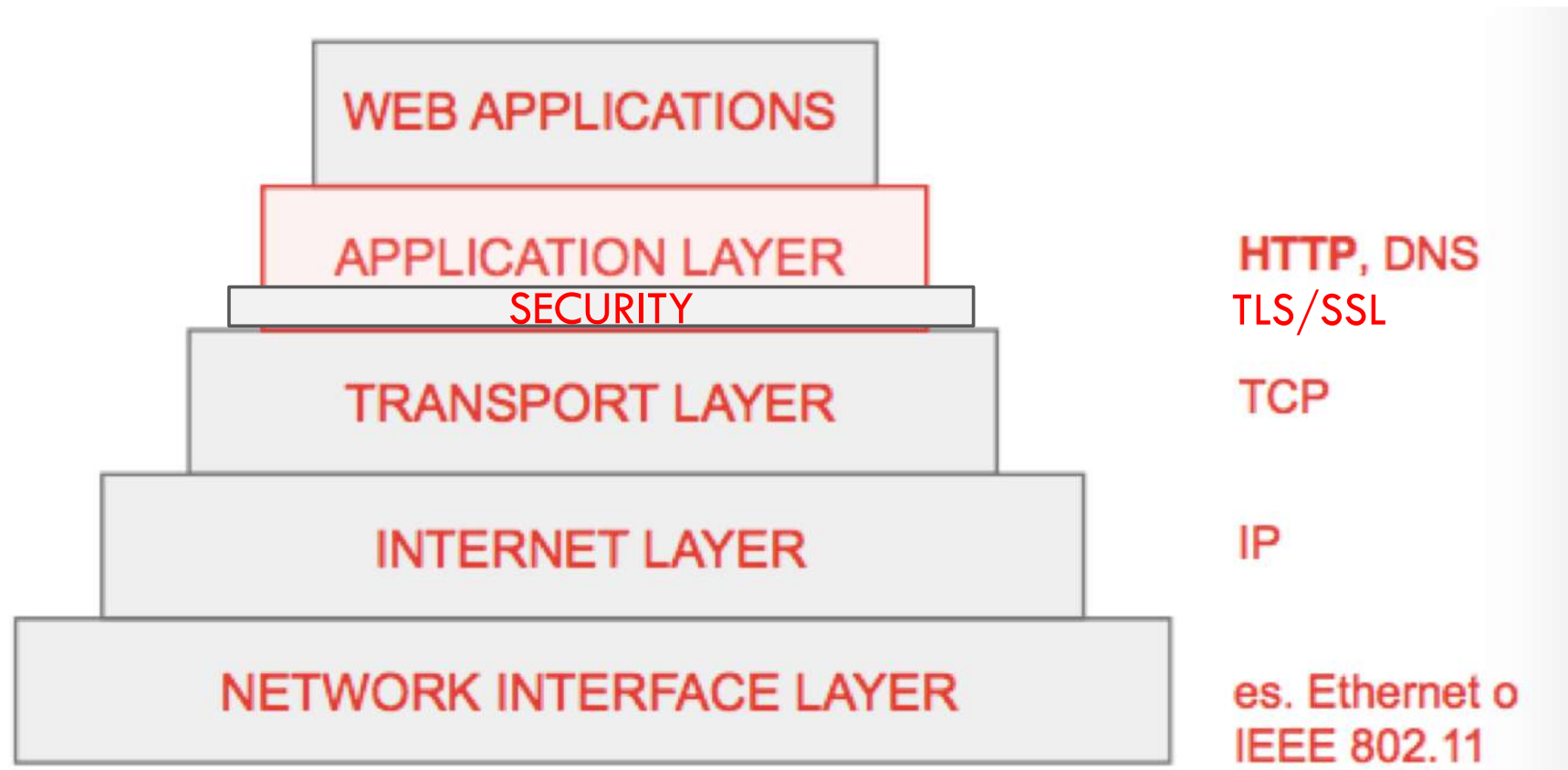
- HTTP è l'acronimo di HyperText Transfer Protocol
- È il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) da server a client
- Gestisce sia le **richieste** (URL) inviate al server che le **risposte** inviate al client (pagine)
- È un protocollo **stateless**: nè il server nè il client mantengono, a livello di protocollo, informazioni relative ai messaggi precedentemente scambiati

# HTTP: TERMINOLOGIA

- **Client:** programma applicativo che stabilisce una connessione al fine di inviare delle richieste
- **Server:** programma applicativo che accetta connessioni al fine di ricevere richieste ed inviare specifiche risposte con le risorse richieste
- **Connessione:** circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione
- **Messaggio:** è l'unità base di comunicazione HTTP, è definita come una specifica sequenza di byte concettualmente atomica
  - **Request:** messaggio HTTP di richiesta (Client → Server)
  - **Response:** messaggio HTTP di risposta (Server → Client)
- **Resource:** oggetto di tipo dato univocamente definito
- **URI:** Uniform Resource Identifier – identificatore unico per una risorsa
- **Entity:** rappresentazione di una risorsa, può essere incapsulata in un messaggio, tipicamente di risposta
- Una **risorsa** può essere una pagina X, mentre una sua corrispondente **entità** è il testo contenuto in essa. Se cambia il testo la risorsa resta comunque la pagina X.

# HTTP NELLO STACK TCP/IP

- HTTP si situa a livello **application** nello stack TCP/IP

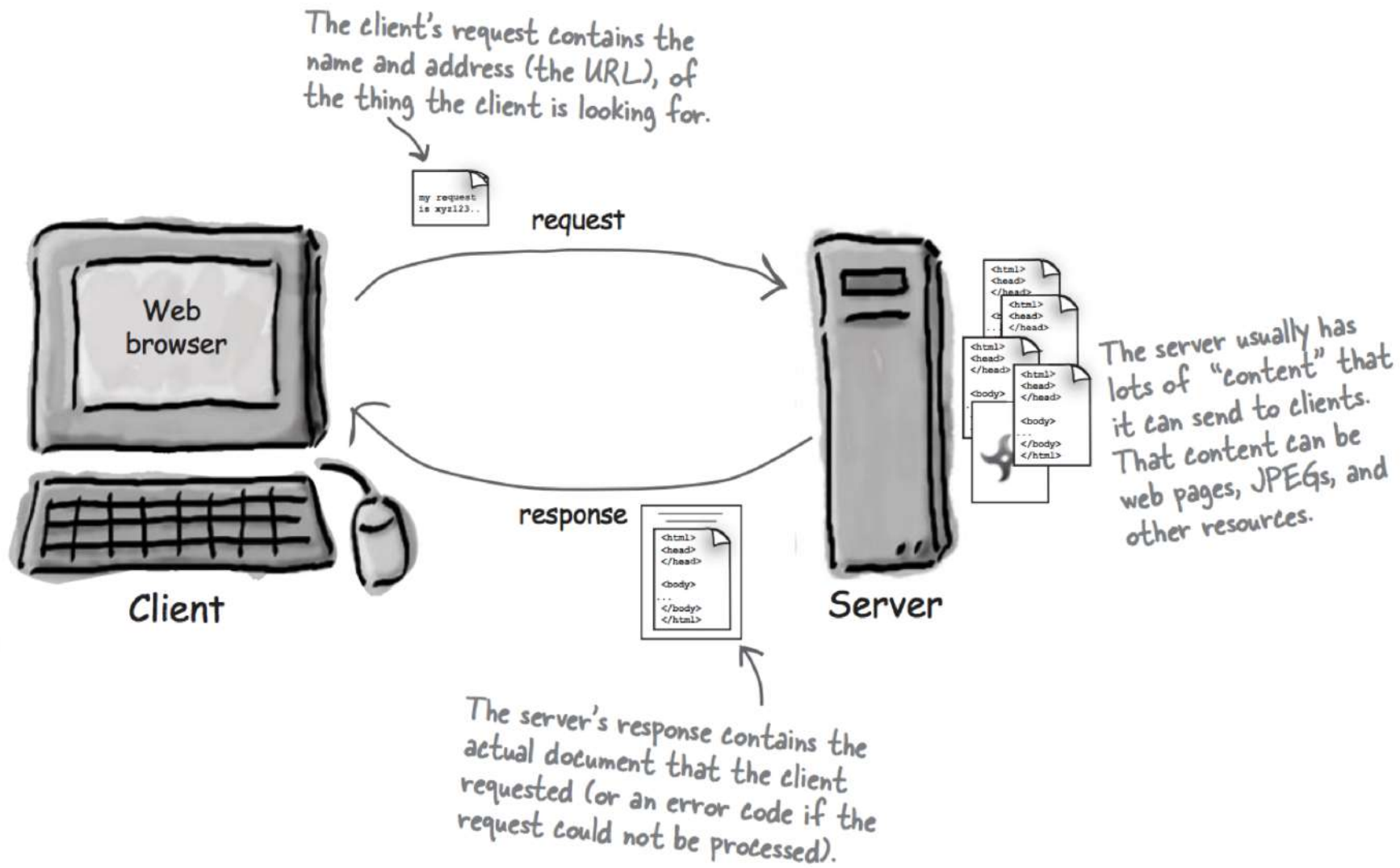


# HTTP

- Almeno per v1.0: **protocollo request-response, stateless, one-shot**
- Sia le richieste al server che le risposte ai client sono trasmesse usando stream TCP
- Segue uno schema di questo tipo:
  - **server** rimane in ascolto (server passivo), tipicamente sulla porta 80
  - **client** apre una connessione TCP sulla porta 80
  - **server** accetta la connessione (possibili più connessioni in contemporanea?)
  - **client** manda una richiesta
  - **server** invia la risposta e chiude la connessione

# WHAT DOES YOUR WEB SERVER DO?

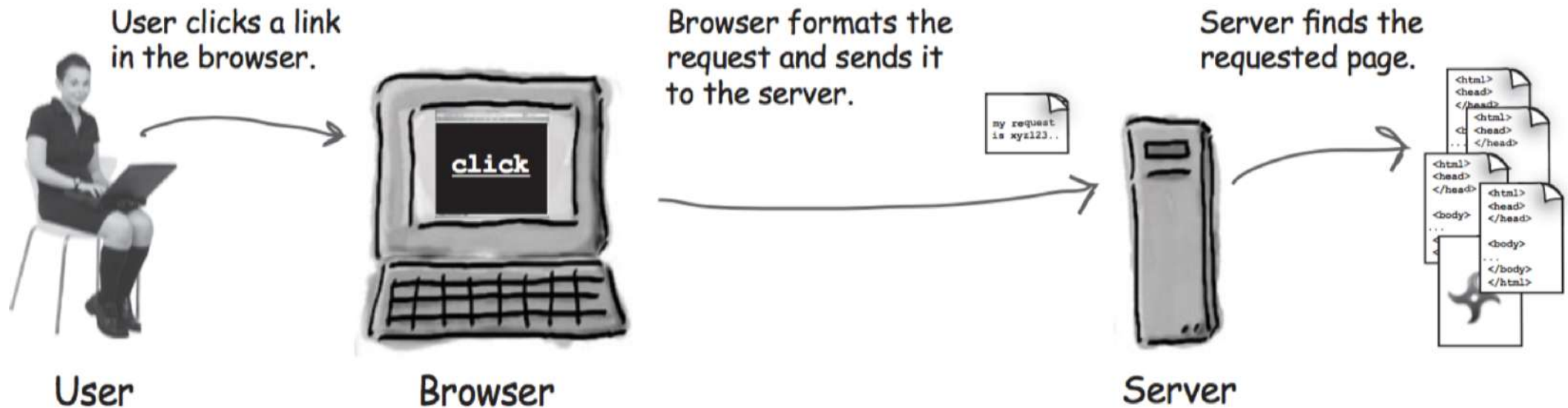
**A web server takes a client request and gives something back to the client**





# WHAT DOES A WEB CLIENT DO?

**A web client lets the user request something on the server, and shows the user the result of the request**



# WHAT DOES A WEB SERVER DO?

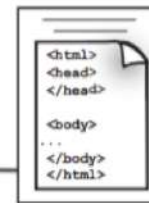
Browser gets the HTML  
and renders it into a  
display for the user.



User



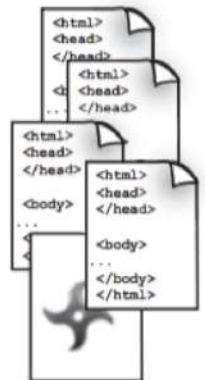
Browser



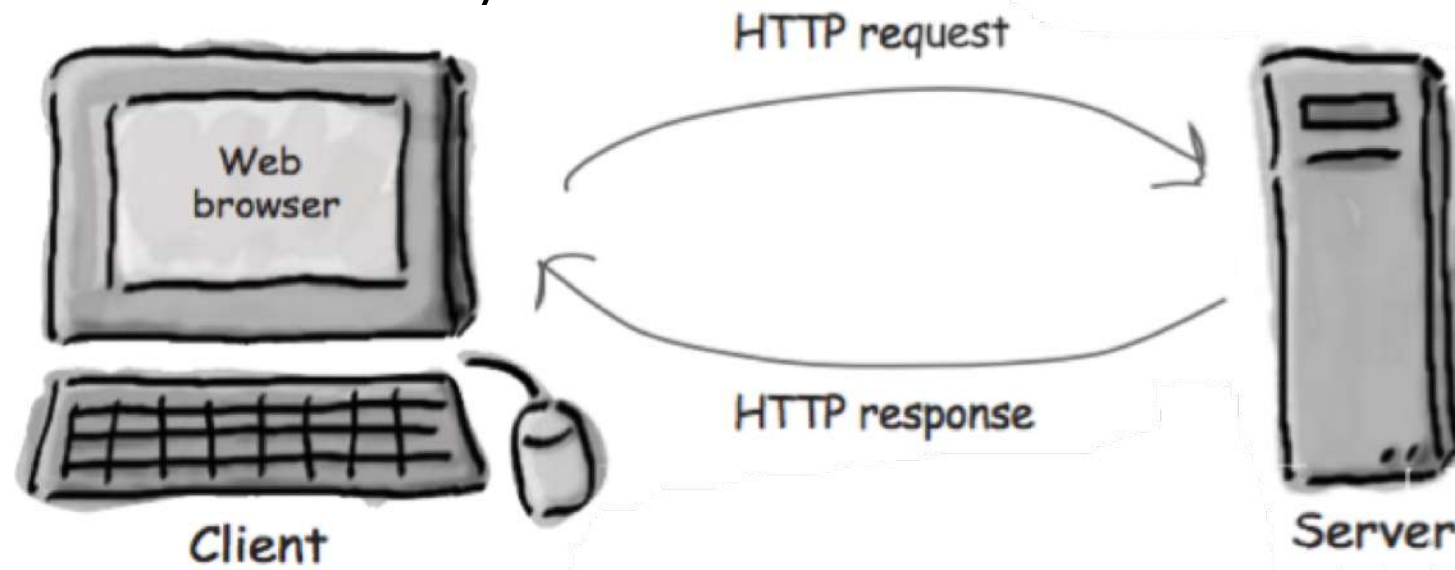
Server formats the  
response and sends it  
to the client (browser).



Server



# THE REQUEST/RESPONSE SEQUENCE



## Key elements of the **request** stream:

- HTTP method (the action to be performed)
- The page to access (a URL)
- Form parameters (like arguments to a method)

## Key elements of the **response** stream:

- A status code (for whether the request was successful)
- Content-type (text, picture, HTML, etc.)
- The content (the actual HTML, image, etc.)

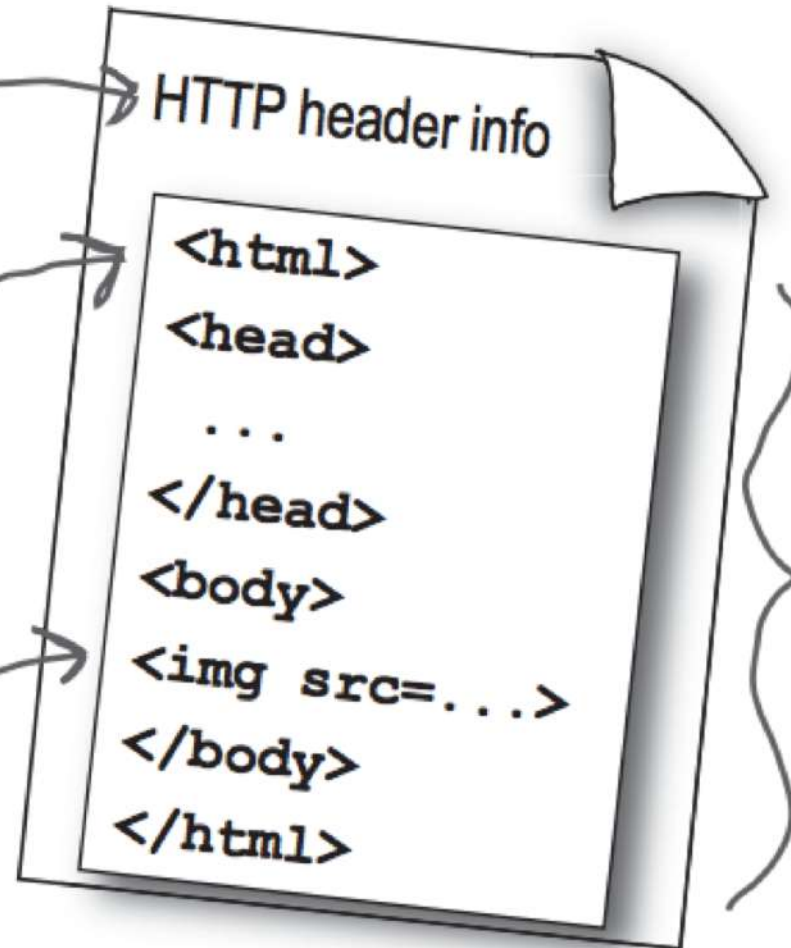
# HTML IS PART OF THE HTTP RESPONSE

HTTP header

HTTP header info

When the browser finds the opening `<html>` tag it goes into HTML-rendering mode and displays the page to the user.

When the browser gets to an image tag, it generates another HTTP request to go get the resource described. In this case the browser will make a second HTTP request to get the picture referenced in the `<img>` tag.

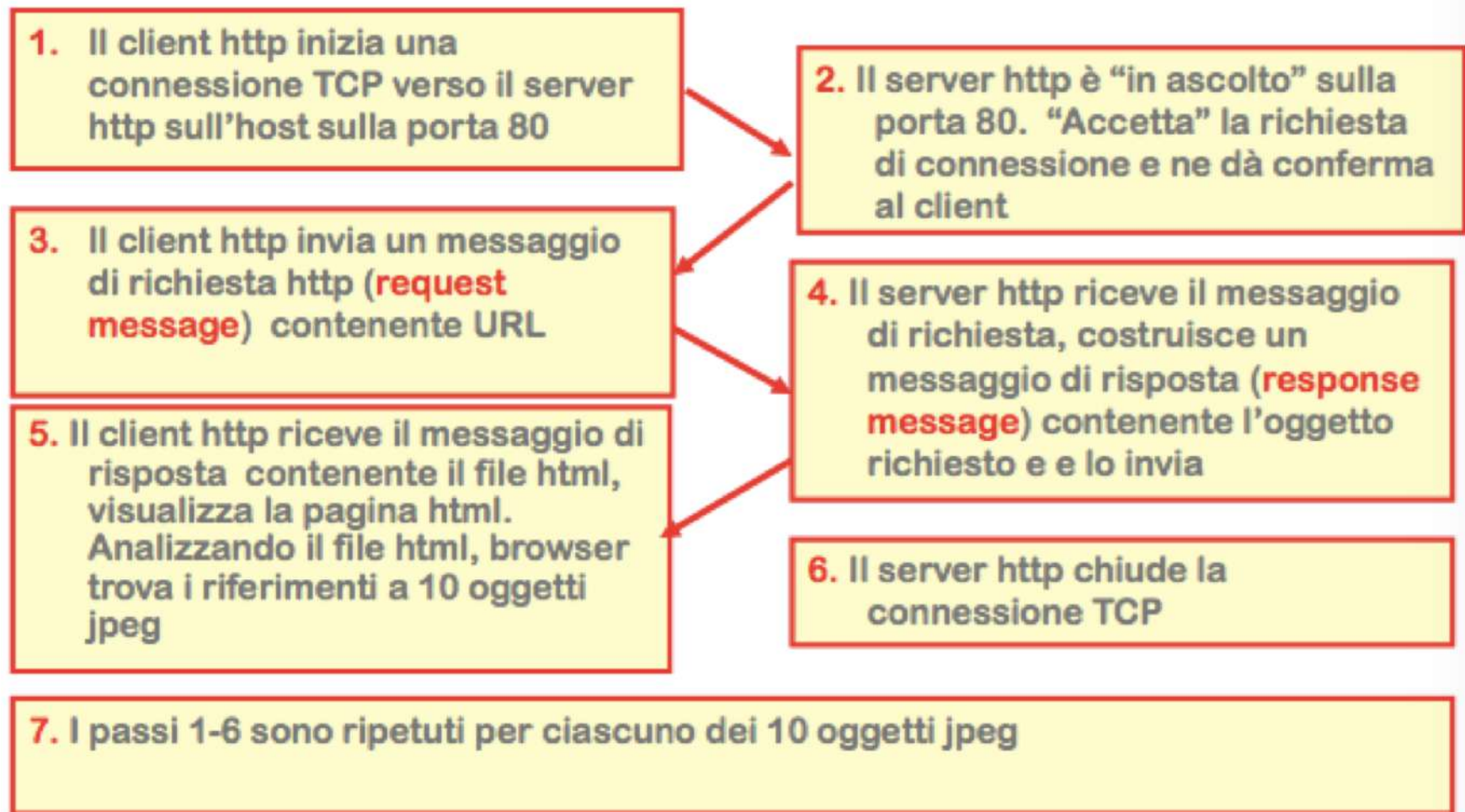


HTTP body



## ESEMPIO HTTP 1.0: REQUEST-RESPONSE, STATELESS, ONE-SHOT

- Ipotizziamo di volere richiedere una pagina composta da un file HTML e 10 immagini JPEG:



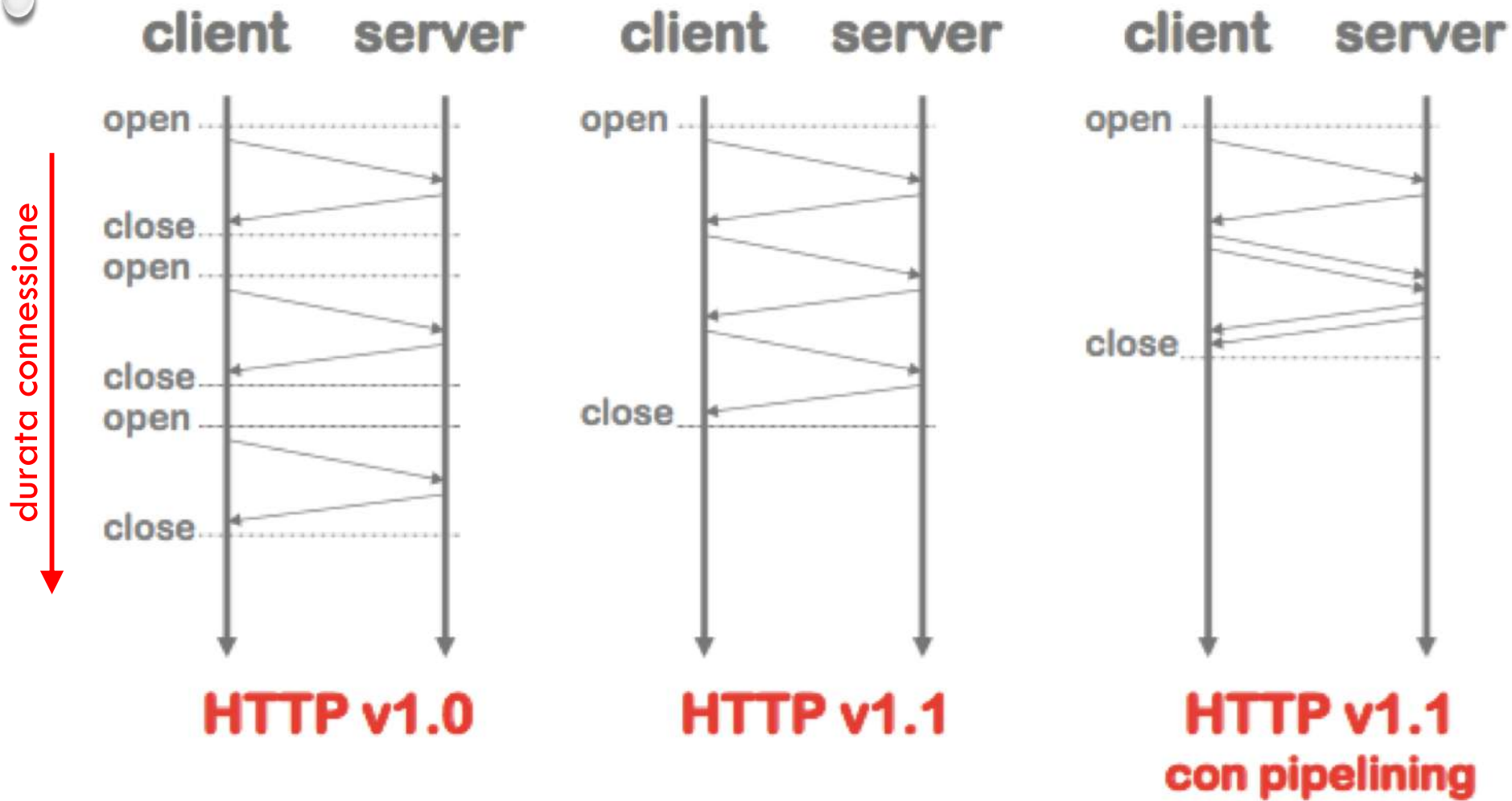
# DIFFERENZE FRA HTTP V1.0 E V1.1

- La stessa connessione HTTP **può essere utilizzata per una serie di richieste e una serie corrispondente di risposte**
- La differenza principale tra HTTP 1.0 e 1.1 è la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione
- Le connessioni 1.0 vengono dette **non persistenti** mentre quelle 1.1 vengono definite **persistenti**
- Il server lascia aperta la connessione TCP dopo aver spedito la risposta e può quindi ricevere le richieste successive sulla stessa connessione
  - Nell'esempio precedente l'intera pagina Web (file HTML + 10 immagini ) può essere inviata sulla stessa connessione TCP
- Il server HTTP chiude la connessione quando viene specificato nell'header del messaggio (desiderata da parte del cliente) oppure quando non è usata da un certo tempo (**time out**)

# HTTP V1.1 E PIPELINING

- Per migliorare ulteriormente le prestazioni si può usare la tecnica del **pipelining**
- Il pipelining consiste nell'invio **di molteplici richieste da parte del client prima di terminare la ricezione delle risposte**
- Le risposte debbono però essere date nello stesso ordine delle richieste

# TIPI DI CONNESSIONE





# HTTP/2

Obiettivo fondamentale di HTTP/2:

miglioramento performance complessiva con full backward compatibility con HTTP 1.1

- request-response multiplexing (è possibile per il server non inviare risposte nello stesso ordine delle richieste)
- header compression (compressione in binario dei dati – trasferimento di dati non in formato ASCII)
- server push (invio di pagine anche non richieste dal client ma che sicuramente lo saranno successivamente - prefetch)

# WHAT'S IN THE REQUEST?

- An **HTTP method name**
  - The method name tells the server the kind of request that's being made, and how the rest of the message will be formatted
- The HTTP protocol has several methods, the most often used are **GET** and **POST**

## GET

User clicks  
a link to a  
new page.

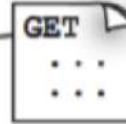


User

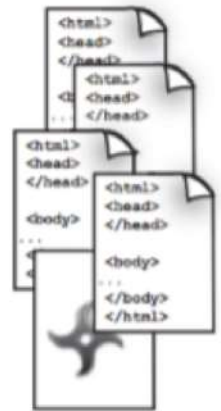


Browser

Browser sends an HTTP GET  
to the server, asking the  
server to GET the page.



Server



## POST

User types in a  
form and hits the  
Submit button.

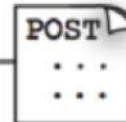


User

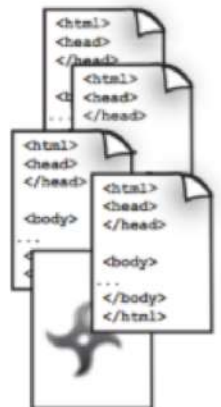


Browser

Browser sends an HTTP POST  
to the server, giving the  
server what the user typed  
into the form.



Server



# GET

- Serve per richiedere una risorsa ad un server
- È il metodo più frequente: è quello che viene attivato facendo click su un link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo di un browser
- È previsto il passaggio di parametri (la parte `<query>` dell'URL)
- Il numero di parametri è **limitato** dalla lunghezza massima di un URL

# POST

- Progettato come il messaggio per richiedere una risorsa
- A differenza di GET, i dettagli per identificazione ed elaborazione della risorsa stessa non sono nell'URL, ma sono contenuti nel corpo (body) del messaggio
- **Non ci sono limiti di lunghezza nei parametri di una richiesta**
- POST viene usato per esempio per sottomettere i dati di una form HTML ad un'applicazione sul server (lo vedremo presto...)

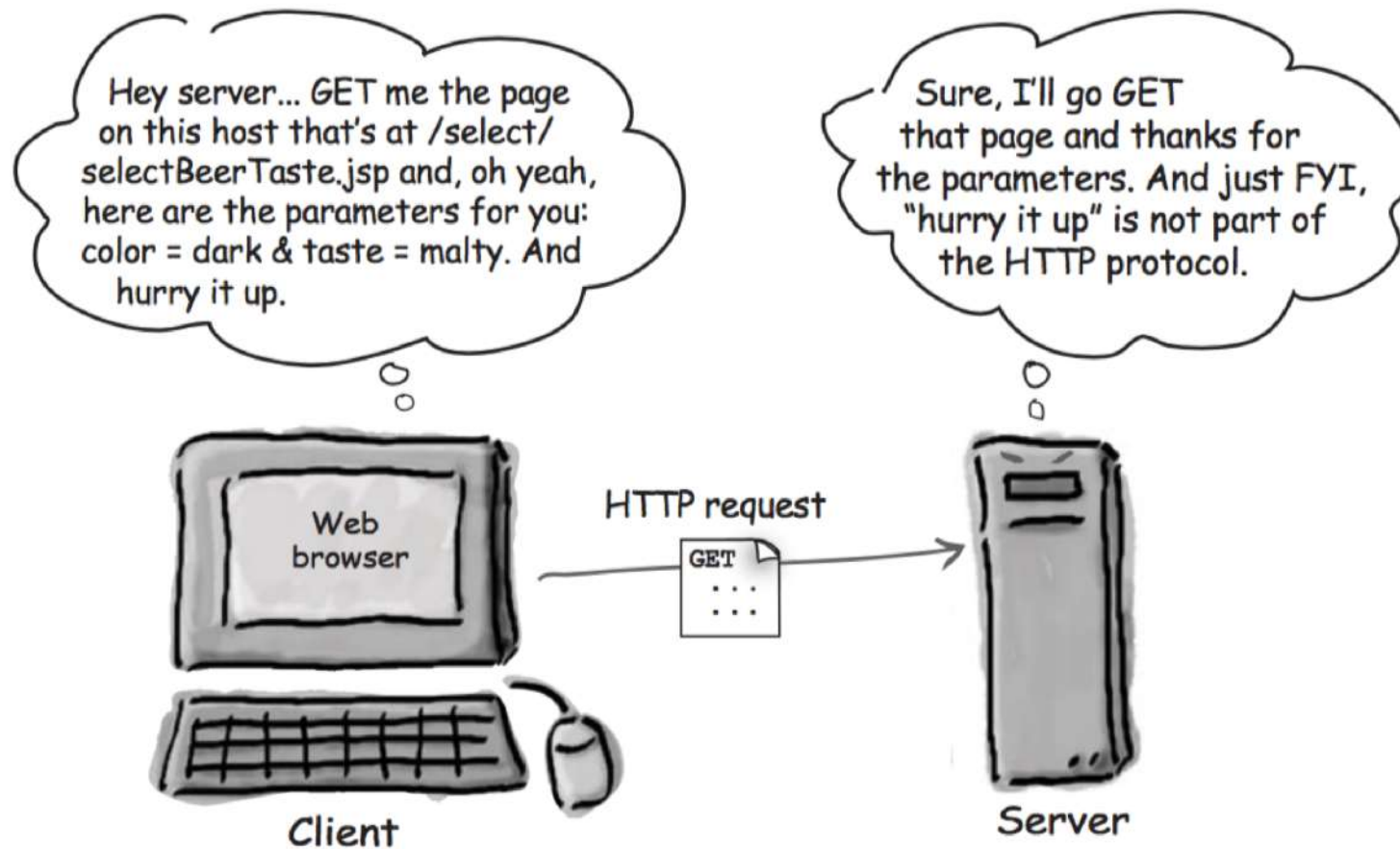
# SEND A LITTLE DATA WITH HTTP GET

- The total amount of characters in a GET is **really limited** If the user types a **long passage** into a “search” input box, the GET **might not work**
- The data you send with the GET is appended to the URL up in the browser bar, so whatever you send is exposed
  - *Better **not put a password** or some other sensitive data as part of a GET!*
- Examples:
  - <http://rubrica.unisa.it/persona?nome=pippo>
  - The original URL before the extra parameter:
    - <http://rubrica.unisa.it/persona>

# DETAILS ON HTTP GET REQUEST

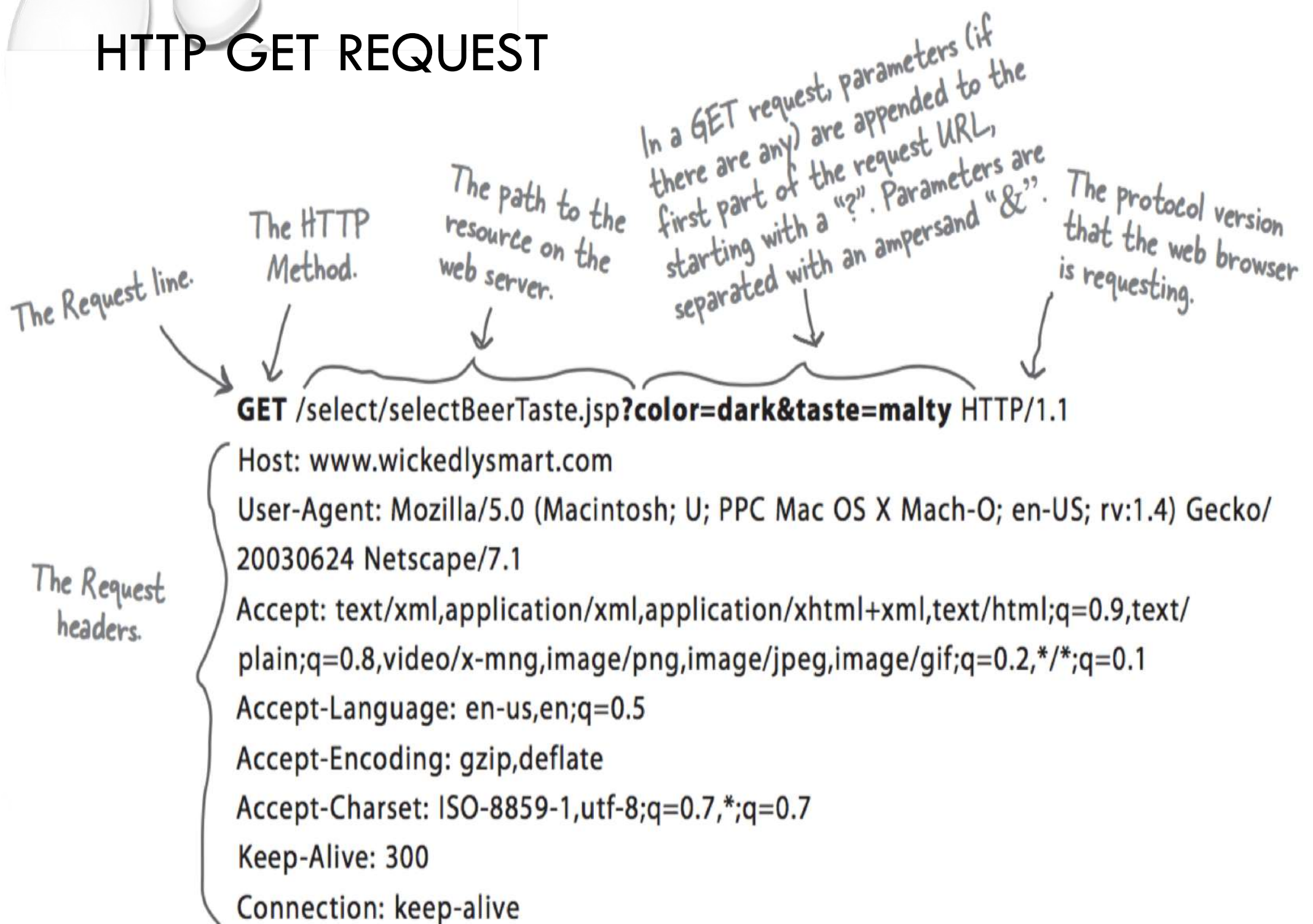
- The path to the resource, and any parameters added to the URL are all included on the “request line”

**GET** /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1





# HTTP GET REQUEST





# HTTP POST REQUEST

The HTTP Method.

The path to the resource on the web server.

The protocol version that the web browser is requesting.

The Request line.

**POST** /advisor/selectBeerTaste.do HTTP/1.1

The Request headers.

Host: www.wickedlysmart.com

User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/20030624 Netscape/7.1

Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,\*/\*;q=0.1

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Connection: keep-alive

**color=dark&taste=malty**

The message body, sometimes called the "payload".

This time, the parameters are down here in the body, so they aren't limited the way they are if you use a GET and have to put them in the Request line.

# HTTP REQUEST: ALTRI METODI

## *PUT AND DELETE*

- **PUT**

- Chiede la memorizzazione sul server di una risorsa all'URL specificato
- Il metodo PUT serve quindi per trasmettere delle informazioni dal client al server
- A differenza del POST però si ha la creazione di una risorsa (o la sua sostituzione se esisteva già)
- L'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET con lo stesso nome in seguito

- **DELETE**

- Richiede la cancellazione della risorsa riferita dall'URL specificato
- **Sono normalmente disabilitati sui server pubblici**

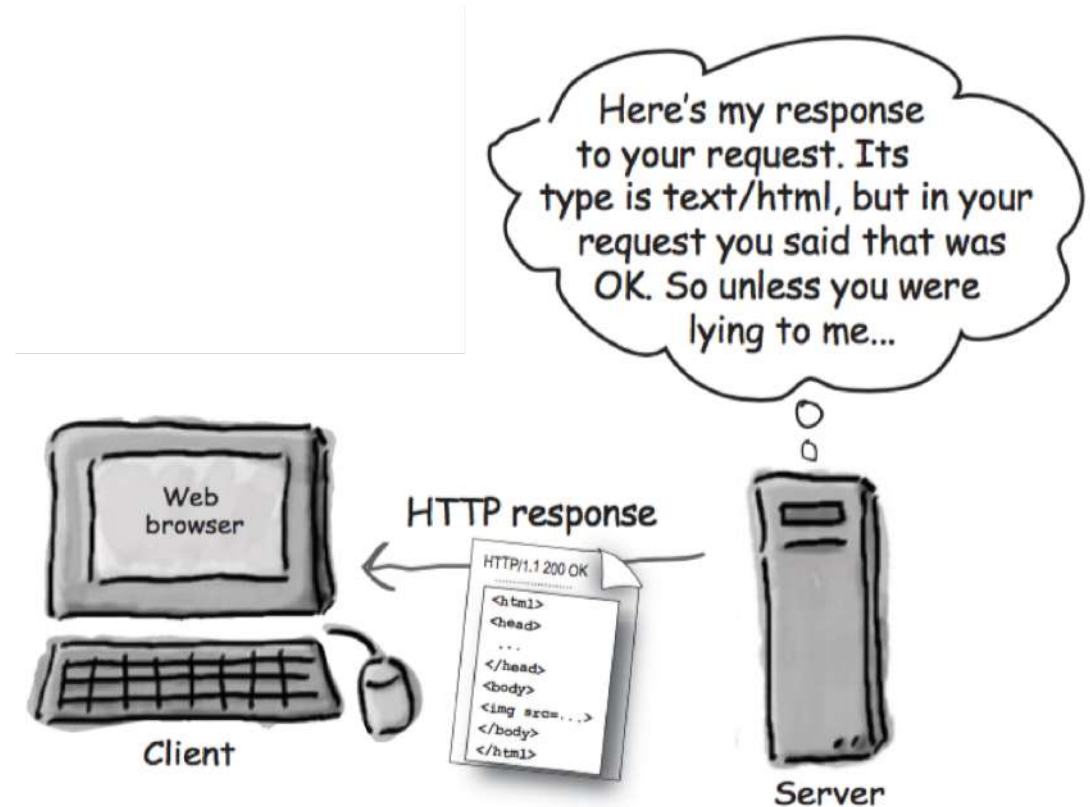
# HTTP REQUEST: ALTRI METODI

## *HEAD, OPTIONS AND TRACE*

- **HEAD:** è simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza body
  - Viene usato per verificare un URL
  - **Validità:** la risorsa esiste e non è di lunghezza zero
  - **Accessibilità:** non è richiesta autenticazione
- **OPTIONS:** serve per richiedere informazioni sulle opzioni disponibili per la comunicazione
- **TRACE:** è usato per invocare il loop-back remoto a livello applicativo del messaggio di richiesta
  - Consente al client di vedere che cosa è stato ricevuto dal server: viene usato nella diagnostica e nel testing dei servizi Web

# HTTP RESPONSE

- An HTTP response has both a header and a body
- The **header** info tells the browser about the protocol being used, whether the request was successful, and what kind of content is included in the body
- The **body** contains the contents (for example, HTML) for the browser to display



# HTTP response

The protocol version that the web server is using.

The HTTP status code for the Response.

A text version of the status code.

**HTTP/1.1 200 OK**

Set-Cookie: JSESSIONID=0AAB6C8DE415E2E5F307CF334BFCA0C1; Path=/testEL

**Content-Type: text/html**

Content-Length: 397

Date: Wed, 19 Nov 2003 03:25:40 GMT

Server: Apache-Coyote/1.1

Connection: close

HTTP  
Response  
headers.

The content-type response header's value is known as a *MIME* type. The *MIME* type tells the browser what kind of data the browser is about to receive so that the browser will know how to render it.

The body holds the HTML, or other content to be rendered...

```
<html>
...
</html>
```

Notice that the *MIME* type value relates to the values listed in the HTTP request's "Accept" header. (Go look at the Accept header from the previous page's POST request.)

# MIME MULTIPURPOSE INTERNET MAIL EXTENSIONS

- Inizialmente sviluppato per la posta elettronica
- Usato per specificare al browser la forma di un file inviato dal server, è inserito dal server all'inizio del documento
- Specifica del tipo
  - Formato: **Tipo/sottotipo**
- Esempi
  - text/plain, text/html, image/gif, image/jpeg
- È possibile estendere i tipi MIME esistenti con dei tipi sperimentali
- Il tipo o sottotipo inizia con una x-
  - video/x-nuovoformato
  - audio/x-wav
- I tipi sperimentali richiedono che il server invii al browser un'applicazione o plug-in per poter far interpretare correttamente i nuovi tipi di dati al browser



# HTTP RESPONSE: *CODICI DI STATO*

- Lo status code è un numero di tre cifre, di cui la prima indica la classe della risposta e le altre due la risposta specifica
- Ci sono 5 classi:
  - **1xx: Informational.** Una risposta temporanea alla richiesta, durante il suo svolgimento (sconsigliata a partire da HTTP 1.0)
  - **2xx: Successful.** Il server ha ricevuto, capito e accettato la richiesta
  - **3xx: Redirection.** Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta
  - **4xx: Client error.** La richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata)
  - **5xx: Server error.** La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema (suo)

# HTTP RESPONSE

## ESEMPI DI CODICI DI STATO

- **100 Continue** (se il client non ha ancora mandato il body, deprecated da HTTPv1.0)
- **200 Ok** (GET con successo)
- **201 Created** (PUT con successo)
- **301 Moved permanently** (URL non valida, il server conosce la nuova posizione)
- Problemi del Client (la richiesta ha problemi)
  - **400 Bad request** (errore sintattico nella richiesta)
  - **401 Unauthorized** (manca l'autorizzazione)
  - **403 Forbidden** (richiesta non autorizzabile)
  - **404 Not found** (URL errato)
- Problemi del Server (l'esecuzione lato server ha problemi)
  - **500 Internal server error** (tipicamente un'applicazione su server mal fatta)
  - **501 Not implemented** (metodo non conosciuto dal server)





The user types a URL



The browser creates an HTTP GET request.

```
GET /test1/Beer1.html HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh...
```

The HTTP GET is sent to the server.



The server finds the page...

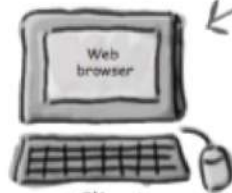
And generates an HTTP response.

```
HTTP/1.1 200 OK
Set-Cookie: ...
...
<html><body>
<h1 align=center>Beer Login Page</h1>
<form>
  Select a beer type or buy beer ...
```

Beer1.html

```
<html><body>
<h1 align=center>Beer Login Page</h1>
<form>
  Select a beer type or buy beer
  making supplies?<p>
  <input type=radio name=select
    value=Select> Select a beer<br>
  <input type=radio name=select
    value=Buy> Buy supplies<br><br>
  <center>
    <input type=SUBMIT>
  </center>
</form>
</body></html>
```

The HTTP response is sent to the browser.



Client

The browser renders the HTML.



Client looks forward to a successful beer transaction.



- SI PROVI A D ESEGUIRE IL SEGUENTE COMANDO DA TERMINALE

**`curl -v http://google.com/`**

- L'output sarà:

```
* Trying 216.58.205.46...
* Connected to google.com (216.58.205.46) port 80 (#0)
> GET / HTTP/1.1
> Host: google.com
> User-Agent: curl/7.49.1
> Accept: */*
>
< HTTP/1.1 302 Found
< Cache-Control: private
< Content-Type: text/html; charset=UTF-8
< Location: http://www.google.it/?gfe_rd=cr&ei=8-03WJbzMK7BXuSHhvgP
< Content-Length: 256
< Date: Thu, 02 Mar 2017 09:20:51 GMT
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.it/?gfe_rd=cr&ei=8-03WJbzMK7BXuSHhvgP">here</A>.
</BODY></HTML>
* Connection #0 to host google.com left intact
```

- Cosa è successo?

(su piattaforma Windows, se non presente, il comando curl può essere scaricato all'indirizzo:

<https://curl.haxx.se/dlwiz/?type=bin&os=Win64&flav=-> )