



CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

Overview: URI, HTTP, HTML

PROF. SIMONE ROMANO

a.a. 2024-2025

URL

- Ogni risorsa sul web ha un **indirizzo unico**, nel formato URL (**Uniform Resource Locator**)

WWW = URL + HTTP + HTML

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?*
 3. *Quali meccanismi (ad es. in termini di protocollo) possiamo utilizzare per accedere alla risorsa?*
- **La risposta a tutte queste domande sono gli URL**

EXAMPLE OF URL

Protocol: Tells the server which communications protocol (in this case HTTP) will be used.

`http://www.wickedlysmart.com:80/beeradvice/select/beer1.html`

Server: The unique name of the physical server you're looking for. This name maps to a unique IP address. IP addresses are numeric and take the form "xxx.yyy.zzz.aaa". You can specify an IP address here instead of a server name, but a server name is a lot easier to remember.

localhost = 127.0.0.1

Port: This part of the URL is optional. A single server supports many ports. A server application is identified by a port. If you don't specify a port in your URL, then port 80 is the default, and as luck would have it, that's the default port for web servers.

Resource: The name of the content being requested. This could be an HTML page, a servlet, an image, PDF, music, video, or anything else the server feels like serving. If this optional part of the URL is left out, most web servers will look for index.html by default.

Path: The path to the location, on the server, of the resource being requested. Because most of the early servers on the web ran Unix, Unix syntax is still used to describe the directory hierarchies on the web server.

Not shown:

Optional Query String:
Remember, if this was a GET request, the extra info (parameters) would be appended to the end of this URL, starting with a question mark "?", and with each parameter (name/value pair) separated by an ampersand "&"

Partiamo da un concetto più generale: URI

- Gli **URI (Uniform Resource Identifier)** forniscono un meccanismo semplice ed estensibile per identificare una risorsa
- 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
 - 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

Riuscite a farmi un esempio?

U come “UNIFORME”

- Gli URI rispettano una sintassi standard, semplice e regolare
- Gli identificatori sono **uniformi**. L'uniformità ha diversi vantaggi:
 - Convenzioni sintattiche comuni
 - Comune semantica per l'interpretazione
 - Possibilità di usare nello stesso contesto differenti tipologie di identificatori anche con meccanismi (protocolli) di accesso diversi
 - Facilità nell'introduzione di nuovi tipi di identificatori (estensibilità)

SINTASSI DEGLI URI

- Un **identificatore** è un frammento di informazione che fa riferimento ad una **entità** dotata di un'**identità** (risorsa)
- Nel caso degli URI gli identifieri sono stringhe con una sintassi definita, dipendente dallo schema, che può essere espressa nella forma più generale in questo modo:

<scheme>:<scheme-specific-part>

- Per la componente **<scheme-specific-part>** non esiste una struttura o una semantica comune a tutti gli URI
- Esiste però un sottoinsieme di URI che condivide una sintassi comune per rappresentare relazioni gerarchiche in uno spazio di nomi:

<scheme>://<authority><path>?<query>

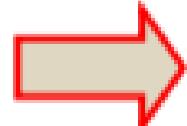
- A parte **<scheme>**, le altre parti possono talora essere omesse, come nei casi in cui non è inclusa la componente **<authority>** o non è inclusa la componente **<query>**

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
 - URL è come indirizzo di casa o numero di telefono (se univoci)

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 procurare il libro



schema



urn:isbn:0-9553010-0-9

↑

Nome univoco nel namespace

URL

WWW = URL + HTTP + HTML

- Un URL tiene conto anche della modalità per accedere alla risorsa
 - Specifica il protocollo necessario per il trasferimento della risorsa stessa (non solo HTTP, quindi...)
 - Tipicamente il nome dello schema corrisponde al protocollo utilizzato
 - La parte rimanente del nome dipende dal protocollo
- Nella sua forma più comune (schema HTTP-like) la sintassi è

```
<protocol>://[<username>:<password>@]<host>[:<port>]  
[/<path>[?<query>][#fragment]]
```

- Questa forma vale per diversi protocolli di uso comune: HTTP, HTTPS, FTP, ...
- Ma non, ad esempio, per la posta elettronica

`mailto:info@example.com?`

`subject=subject&cc=cc@example.com&body=Hello`

COMPONENTI DI UN URL CON SCHEMA HTTP-LIKE

<protocol>://[<username>:<password>@]<host>[:<port>] [<path>[?<query>][#fragment]]

- <**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](#) ma dipende dal server)
- <**query**>: una stringa di caratteri che consente di passare al server uno o più parametri. Di solito ha questo formato:
 - **parametro1=valore1¶metro2=valore2...**
- **fragment**: è una breve stringa di caratteri che si riferisce a una risorsa che è subordinata a un'altra risorsa primaria

ESEMPIO DI URL CON SCHEMA HTTP

Indirizzo IP del server in cui la pagina è memorizzata

Porta di ascolto del server

http://www.di.unisa.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

URL OPACHE E GERARCHICHE

- Le URL sono anche classificate come opache o gerarchiche
 - **URL opaca:** non è soggetta a ulteriori operazioni di parsing
 - `mailto:siromano@unisa.it`
 - **URL gerarchica:** è soggetta a ulteriori operazioni di parsing, per esempio per separare l'indirizzo del server dal percorso all'interno file system
 - `http://di.unisa.it/docs/guide/collections/designfaq.html#28`
 - `../../lab/examples/ant/build.xml`
 - `file:///~/calendar`

OPERAZIONI SULLE URL GERARCHICHE

- **Normalizzazione:** è un processo di trasformazione delle URL in una forma standard per evitare duplicati e garantire che diverse varianti della stessa risorsa siano trattate in modo univoco
- **Risoluzione:** è il processo che a partire da una URL originaria porta all'ottenimento di una URL risultante
 - La URL originaria viene risolta basandosi su una terza URL, detta base URL
 - URL originaria: [docs/guide/collections/designfaq.html#28](#)
 - Base URL: <http://di.unisa.it/>
 - Risultato:
<http://di.unisa.it/docs/guide/collections/designfaq.html#28>
- **Relativizzazione:** è il processo inverso alla risoluzione

RIFERIMENTI BIBLIOGRAFICI

Official Internet Protocol Standards – RFC (<https://www.rfc-editor.org>) -
Request for Comments – Internet Engineering Task Force

- 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>

WWW = URL + HTTP + HTML

HTTP

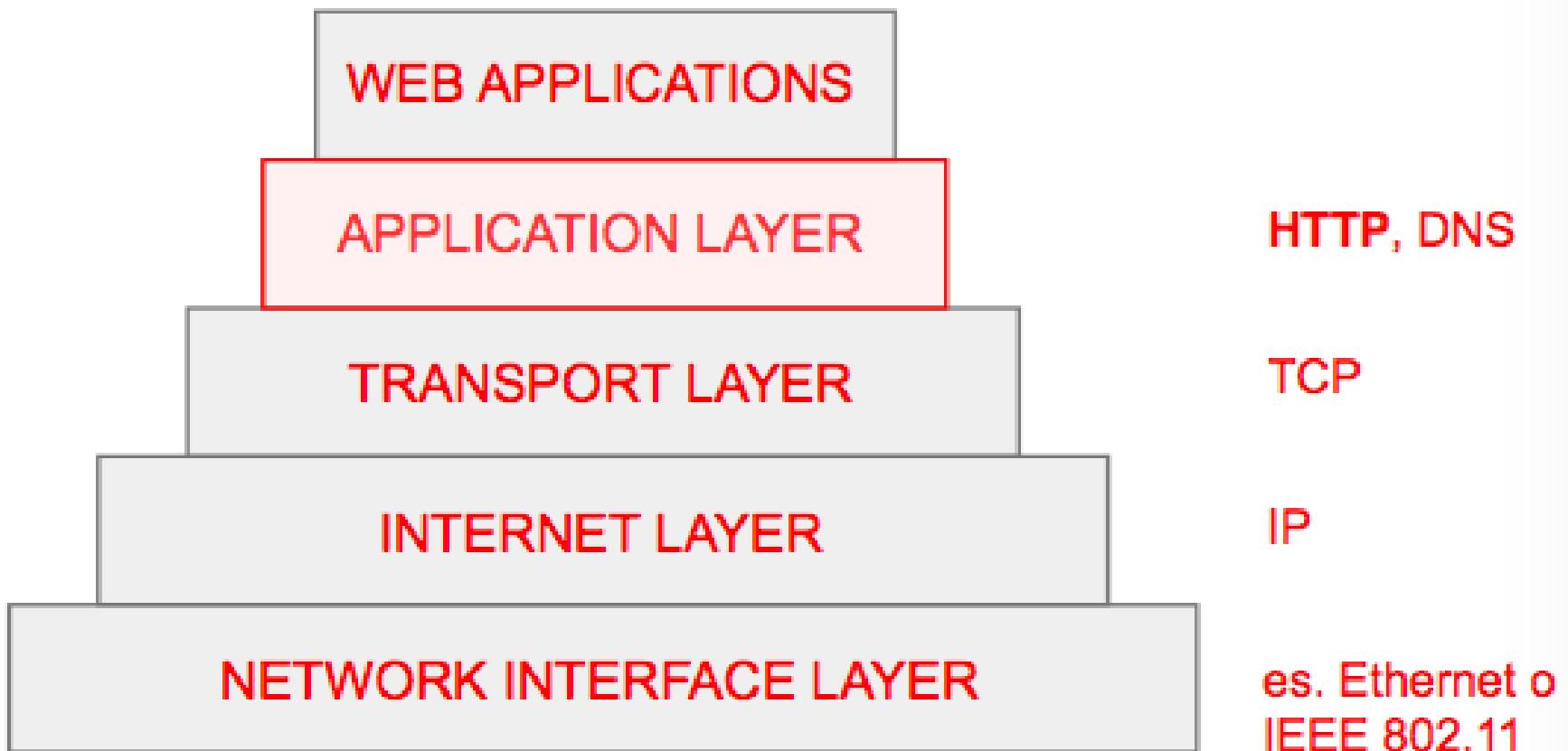
- HTTP è l'acronimo di **HyperText Transfer Protocol**
- È il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) da server a client
- Gestisce sia le richieste (URL) inviate al server che le risposte inviate al client (pagine)
- È un protocollo **stateless**: *né il server né 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
 - **Request:** messaggio HTTP di richiesta (Client - Server)
 - **Response:** messaggio HTTP di risposta (Server - Client)
- **Resource:** oggetto concettuale univocamente definito
- **Entity:** la rappresentazione specifica di quella risorsa trasmessa in un messaggio HTTP

HTTP NELLO STACK TCP/IP

- HTTP è a livello **application** nello stack TCP/IP



WHAT IS THE HTTP PROTOCOL?

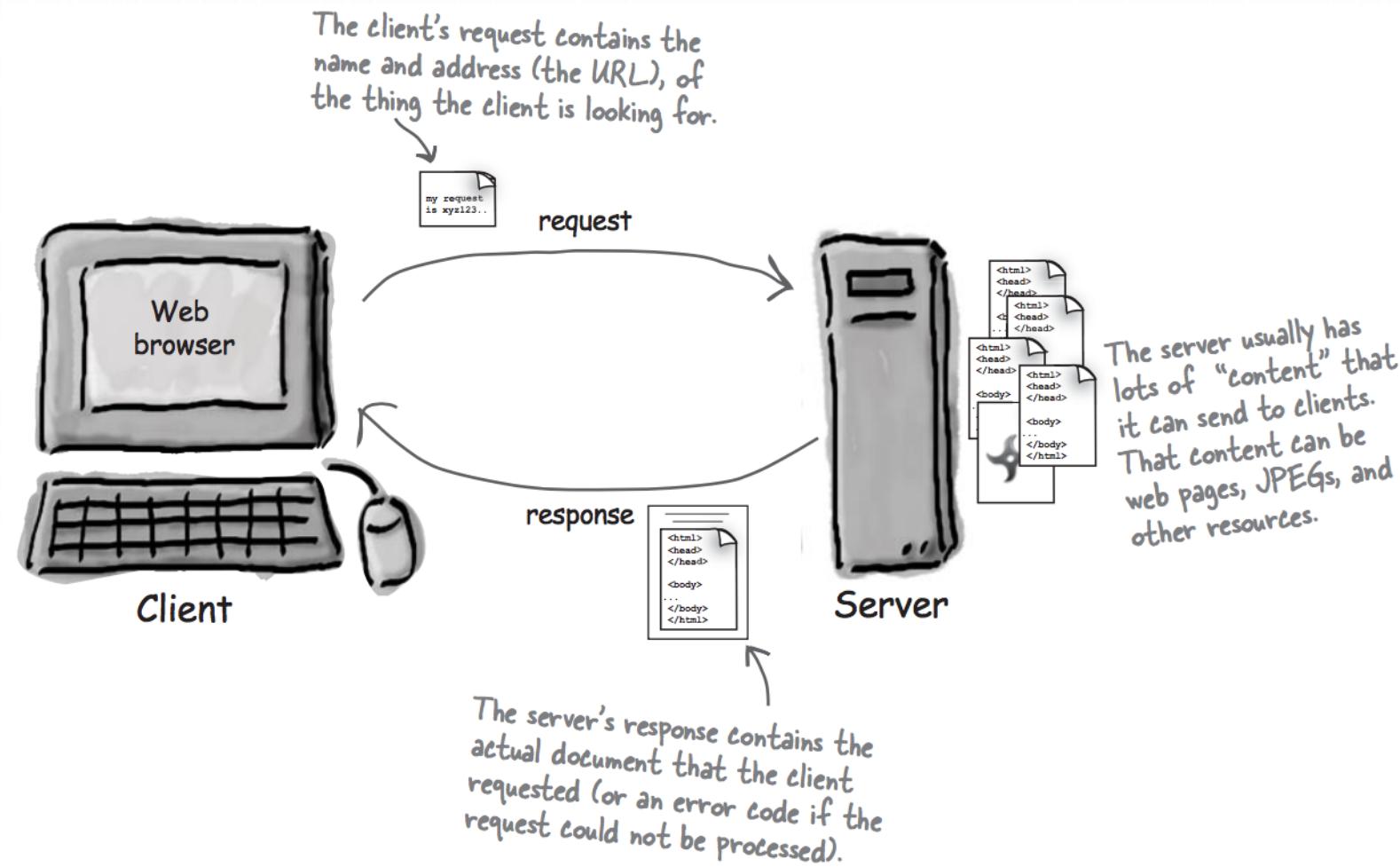
- **HTTP** è il protocollo che ha caratteristiche specifiche del Web, ma dipende da TCP/ IP per ottenere la richiesta completa e la risposta da un luogo all'altro
- **TCP** è responsabile di assicurarsi che un file inviato da un nodo di rete a un altro sia completamente inviato, anche se il file viene diviso in blocchi quando viene inviato
- **IP** è il protocollo sottostante che sposta/instrada i blocchi (pacchetti) da un host all'altro nel loro cammino verso la destinazione

HTTP

- HTTP 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
- 

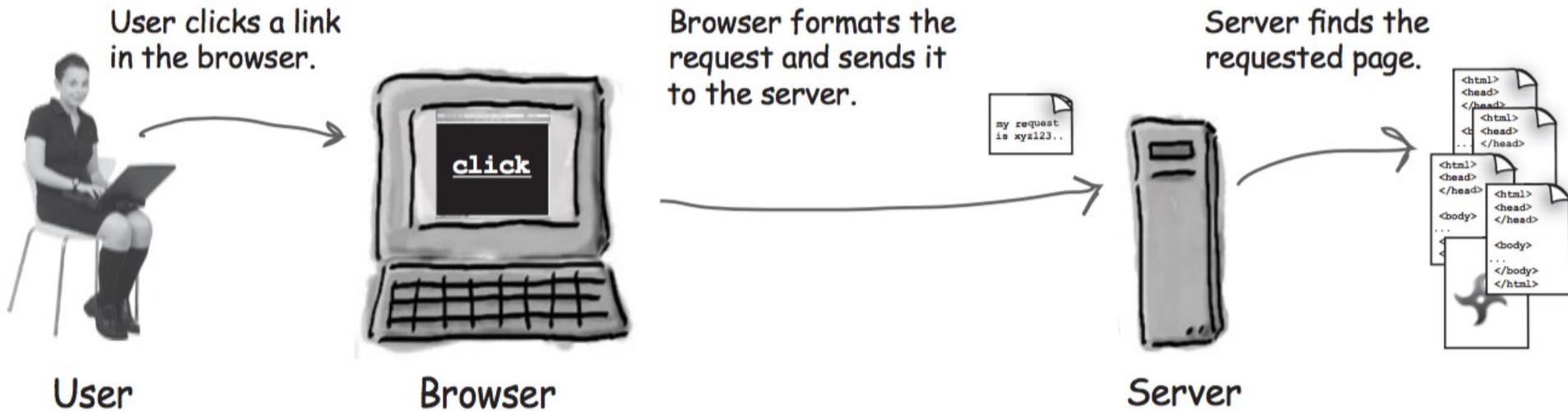
WHAT DO CLIENT AND WEB SERVER DO?

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

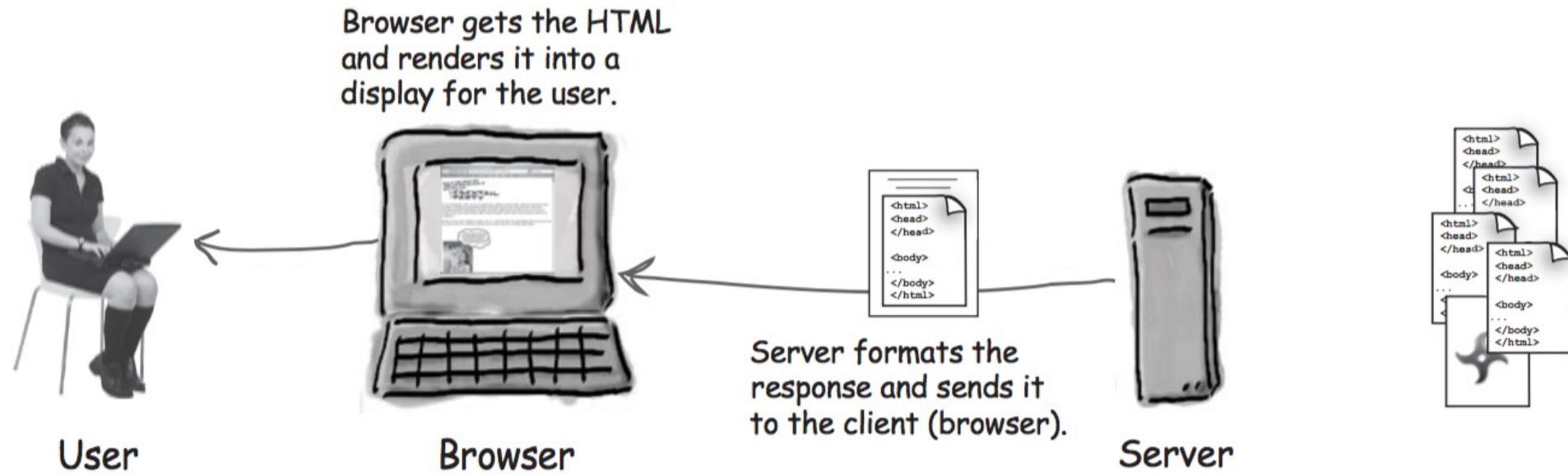


WHAT DOES A WEB CLIENT DO?

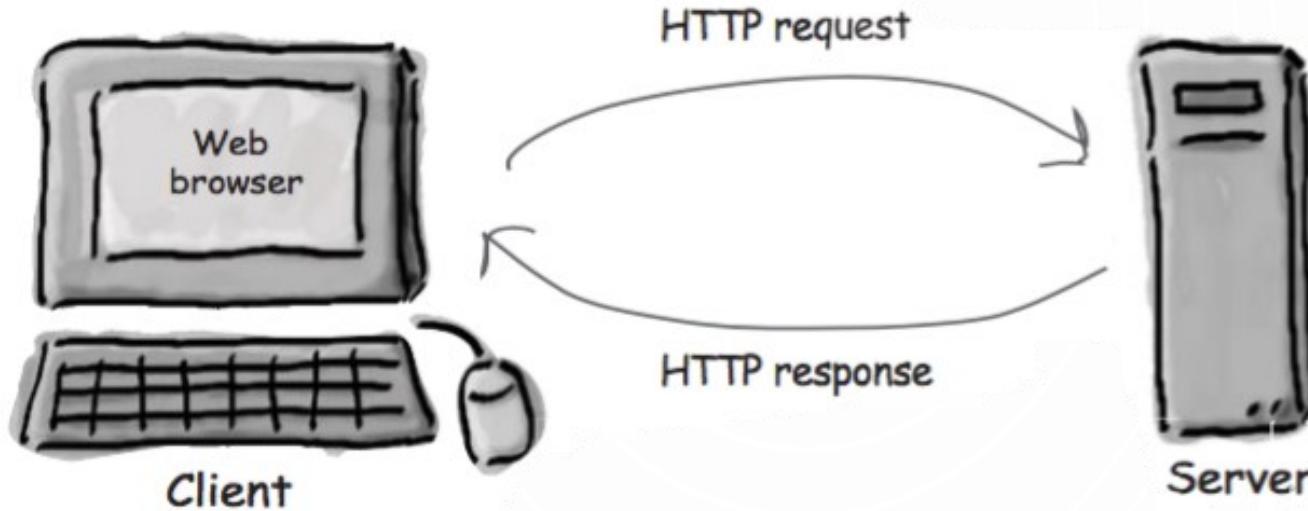
A web client lets the user request something on the server, and shows the user the result of the request



WHAT DOES A WEB SERVER DO?



THE REQUEST/RESPONSE SEQUENCE



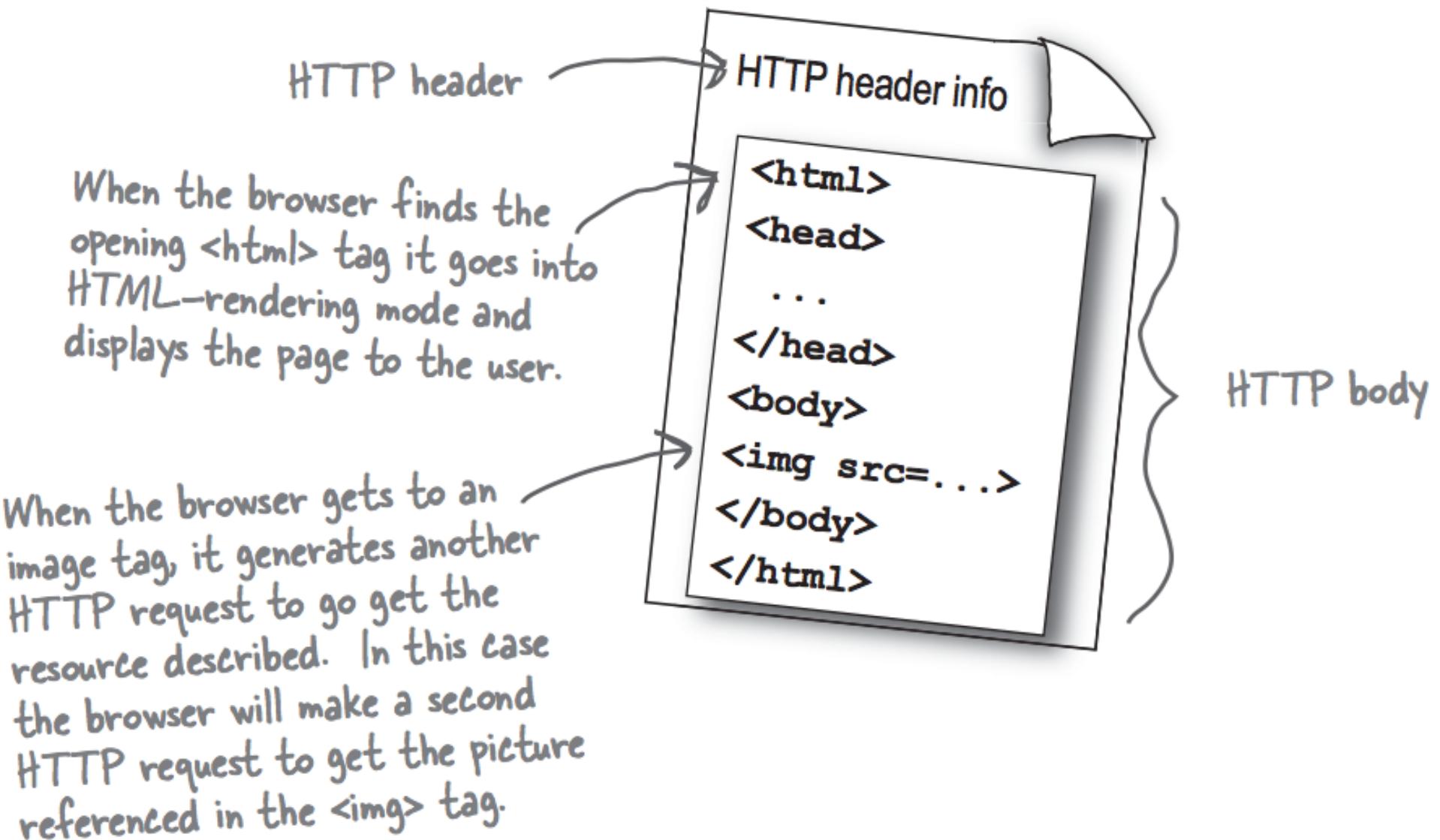
Key elements of the **request** stream:

- 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



ESEMPIO HTTP V1.0: REQUEST-RESPONSE, STATELESS, ONE-SHOT

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



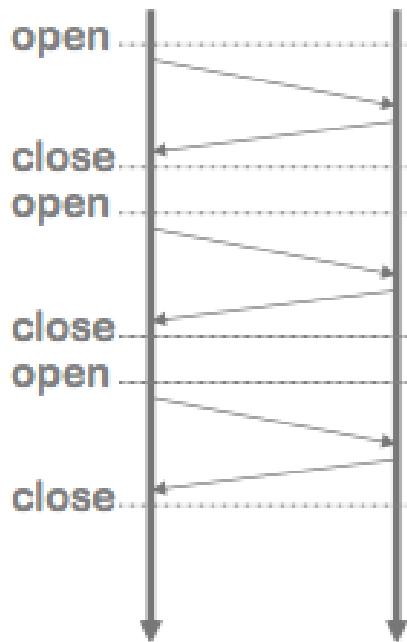
DIFFERENZE FRA HTTP V1.0 E V1.1

- 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 della versione 1.0 di HTTP sono dette **non persistenti** mentre quelle della versione 1.1 sono 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 (**timeout**)

HTTP V1.1 E PIPELINING

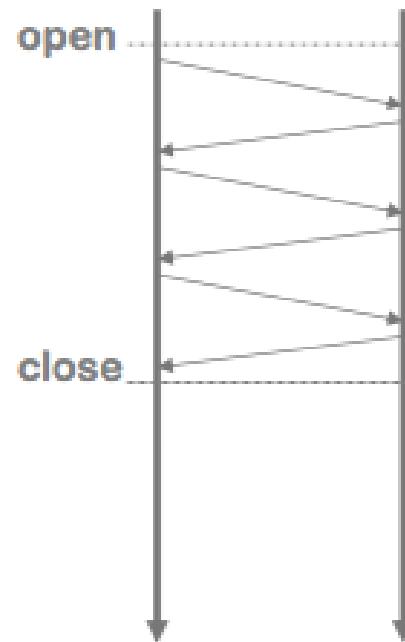
- Per migliorare **ulteriormente le prestazioni** si può usare la tecnica del **pipelining**
- Il pipelining consiste nell'**invio di molteplici richieste da parte del client prima di terminare la ricezione delle risposte**
- **Le risposte debbono però essere date nello stesso ordine delle richieste**, poiché non è specificato un metodo esplicito di associazione tra richiesta e risposta (si pensi al funzionamento di TCP al sottostante livello di trasporto)

client server



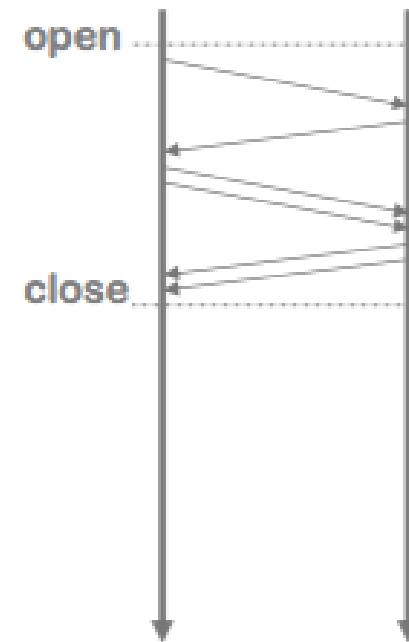
HTTP v1.0

client server



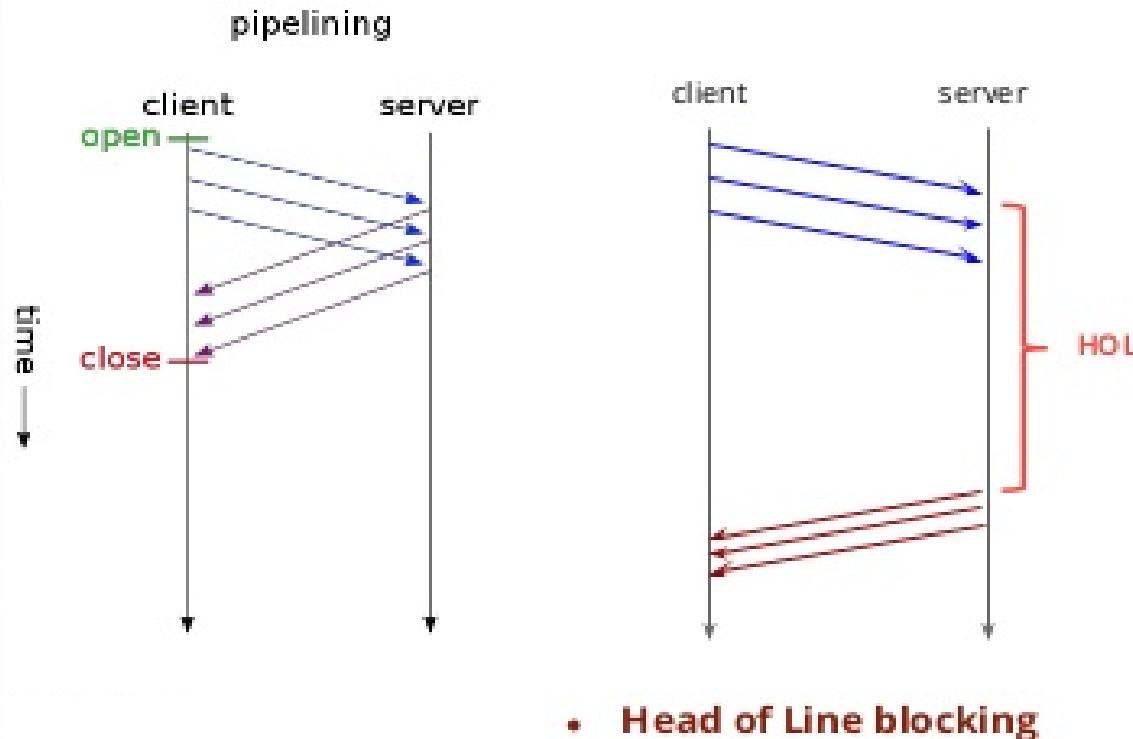
HTTP v1.1

client server



**HTTP v1.1
con pipelining**

HEAD OF LINE BLOCKING



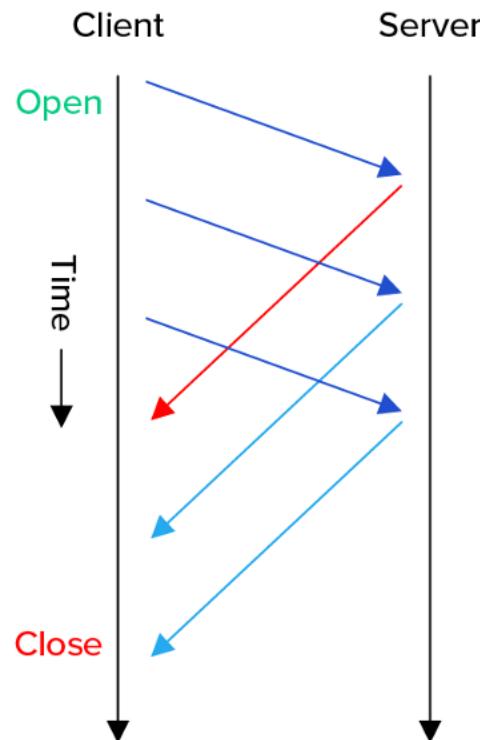
WIKI Pedia: Head-of-line blocking (HOL blocking) in computer networking is a performance-limiting phenomenon that occurs when a line of packets is held up in a queue by a first packet. Examples include input buffered network switches, out-of-order delivery and multiple requests in HTTP pipelining.

HTTP 2.0 o HTTP 2

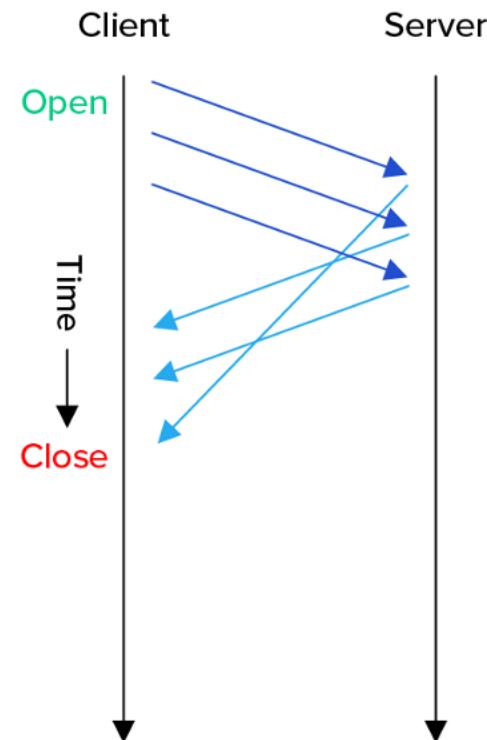
- Compatibilità di alto livello con HTTP 1.1 (per esempio con i metodi, codici di stato, e URI, e la maggior parte campi di intestazione)
- **Ridurre la latenza** per migliorare la velocità di caricamento delle pagine nei browser Web considerando:
 - La **compressione** dei dati degli header HTTP
 - Le tecnologie **push** lato server (ricezione in modo asincrono dei dati)
 - tecnologia che permette al server di fornire le risorse di una pagina ancora prima che il client ne faccia richiesta
 - La **pipeline** delle richieste
 - Soluzioni al problema del blocco head-of-line (HOL bloking) in HTTP 1
 - ossia della lunga attesa dovuta a una risorsa che occupa molto tempo a essere servita dal server al client
 - Il **caricamento in parallelo** degli elementi di una pagina su una singola connessione TCP (multiplexed request-response)
 - il browser non resta **nulla-facente** nei periodi di attesa, ma si occupa di **renderizzare** ogni risorsa della pagina man a mano che questa si rende disponibile, ri-chiedendo di continuo le nuove risorse di cui ha necessità

TIPI DI CONNESSIONE

Pipelining
(Head-of-line Blocking)



HTTP Multiplexing



WHAT'S IN THE REQUEST?

- **HTTP method name**
 - indica al server **il tipo di richiesta** che viene effettuata e come sarà formattato il resto del messaggio
- Il protocollo HTTP ha diversi method name, i più utilizzati sono **GET** and **POST**

GET

User clicks a link to a new page.

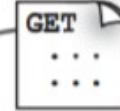


User

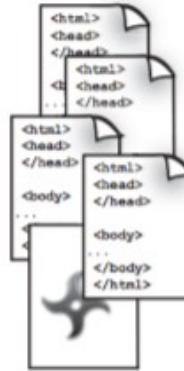


Browser

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



Server



POST

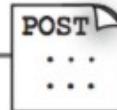
User types in a form and hits the Submit button.



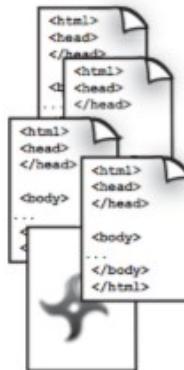
User

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)
- La lunghezza massima di un URL è **limitata**

POST

- **Serve per richiedere una risorsa al server**
- A differenza di GET, i dettagli per identificazione ed elaborazione della risorsa stessa non sono nell'URL, ma sono **contenuti nel body 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 vedrete presto... 😊*)

GET & POST: DIFFERENCES

- **GET** is the simplest HTTP method
 - Chiede al server di recuperare una risorsa e inviarla indietro. Questa risorsa potrebbe essere una pagina HTML, un JPEG, un PDF, ecc.
- **POST** is a more powerful request
 - E' possibile richiedere qualcosa e allo stesso tempo inviare i dati del modulo al server



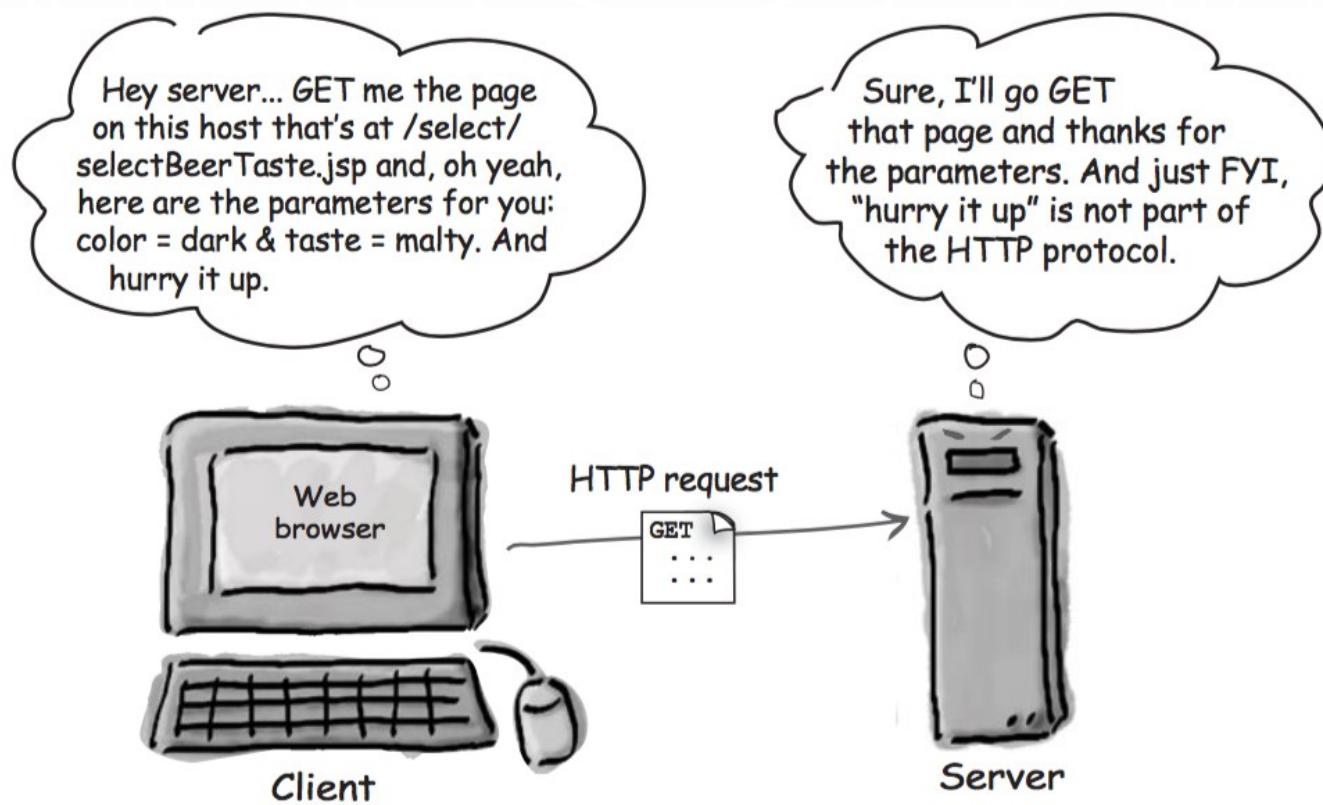
SEND A LITTLE DATA WITH HTTP GET

- La quantità totale di caratteri in un GET è davvero limitata
Se l'utente digita un lungo messaggio in una casella di "ricerca", il GET potrebbe non funzionare
- I dati inviati con GET vengono aggiunti all'URL nella barra del browser, in modo che qualsiasi messaggio inviato sia visibile
 - *Better not put a password or some other sensitive data as part of a GET!*
- Examples:
 - <http://rubrica.unisa.it/persone?nome=scanniello>
 - The original URL before the extra parameter:
 - <http://rubrica.unisa.it/persone>

DETAILS ON HTTP GET REQUEST

- Il percorso della risorsa e tutti i parametri aggiunti all'URL sono tutti inclusi nella “request line”

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



HTTP GET REQUEST

The Request line.

The HTTP Method.

The path to the resource on the web server.

In a GET request, parameters (if there are any) are appended to the first part of the request URL, starting with a "?". Parameters are separated with an ampersand "&".

The protocol version that the web browser is requesting.

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

HTTP POST REQUEST

The Request line.

The Request headers.

The message body, sometimes called the "payload"

The HTTP Method.

The path to the resource on the web server.

The protocol version that the web browser is requesting.

POST /advisor/selectBeerTaste.do HTTP/1.1

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

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.

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	<p>GET is less secure compared to POST because data sent is part of the URL</p> <p>Never use GET when sending passwords or other sensitive information!</p>	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

PUT AND DELETE

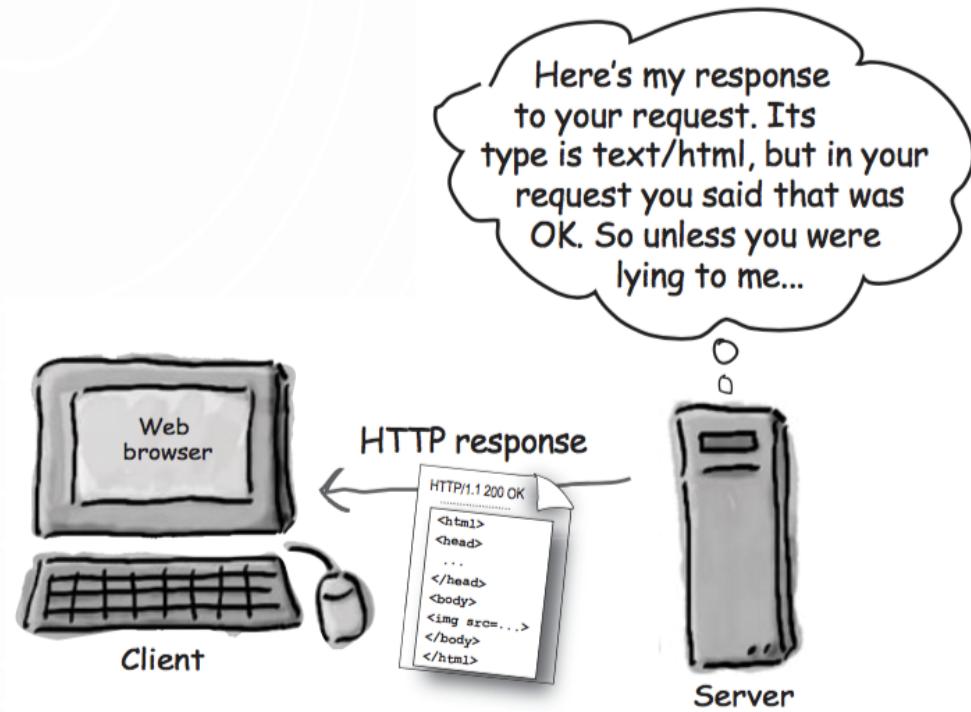
- **PUT**
 - Chiede la memorizzazione sul server di una risorsa all'URL specificato
 - Il metodo PUT serve quindi per trasmettere delle informazioni dal client al server
 - A differenza del POST però si ha la creazione di una risorsa (o la sua sostituzione se esisteva già)
- **DELETE**
 - Richiede la cancellazione della risorsa riferita dall'URL specificato
- **Sono normalmente disabilitati sui server pubblici**

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

- Una risposta HTTP ha sia un header sia un body
- Le informazioni dell'**header** dicono al browser sul protocollo utilizzato, se la richiesta ha avuto successo e che tipo di contenuto è incluso nel corpo
- Il **body** include i contenuti (ad esempio, HTML) da visualizzare nel browser



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

<html>

...

</html>

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.

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

MIME: MULTIPURPOSE INTERNET MAIL EXTENSIONS

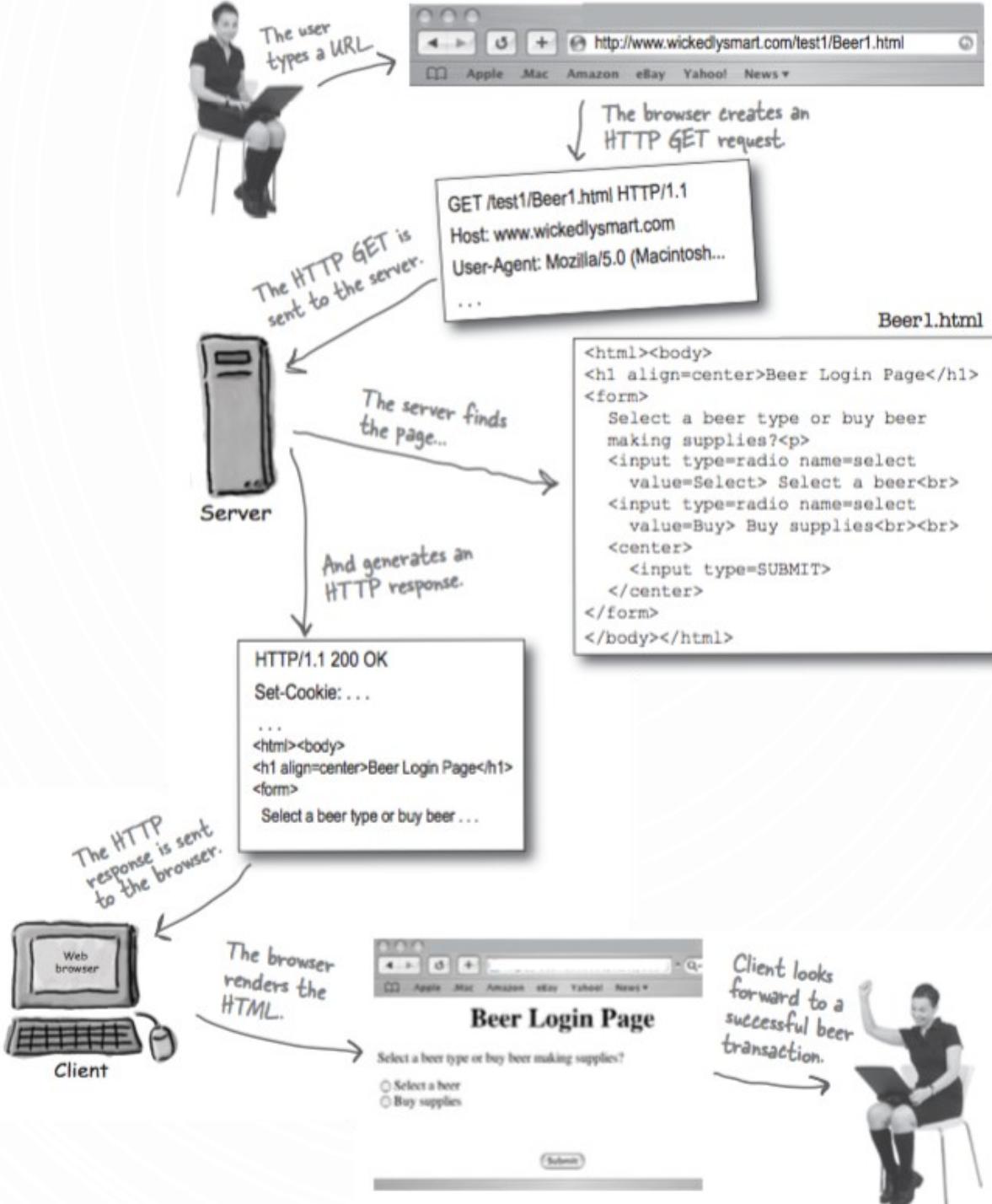
- Inizialmente sviluppato per la posta elettronica
- Usato per specificare al browser la forma di un file inviato dal server, è inserito dal server all'inizio del documento
- Specifica del tipo
 - Formato: **tipo/sottotipo**
- Esempi
 - text/plain, text/html, image/gif, image/jpeg
- È possibile estendere 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

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.** Il server ha ricevuto la richiesta e la sta processando
 - **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 o di applicazioni CGI (Common Gateway Interface))

ESEMPI DI CODICI DI STATO

- **100 Continue** (la richiesta è stata ricevuta dal server e il server è pronto a continuare a elaborare la richiesta)
- **200 Ok** (la richiesta ha avuto successo)
- **201 Created** (la richiesta ha avuto successo e la risorsa è stata creata come richiesto)
- **301 Moved permanently** (URL non valida, il server conosce la nuova posizione)
- **400 Bad request** (errore sintattico nella richiesta)
- **401 Unauthorized** (manca l'autorizzazione, il client si deve autenticare per ottenere la risorsa)
- **403 Forbidden** (richiesta non autorizzabile)
- **404 Not found** (URL errato)
- **500 Internal server error** (il server ha incontrato una situazione che non è in grado di gestire)
- **501 Not implemented** (il metodo richiesto non è supportato)



I COOKIE

- Parallelamente alle sequenze **request/response**, il protocollo prevede una struttura dati che si muove come un **token**, **dal client al server e viceversa**: “**i cookie**”
- I cookie sono generati dai server e vengono memorizzati nei browser
- Dopo la loro creazione vengono sempre passati ad ogni trasmissione di request e response
- Hanno come scopo quello di fornire un supporto per **il mantenimento di stato** in un protocollo come HTTP che è essenzialmente **stateless**

STRUTTURA DEI COOKIE

- I cookie non sono altro che una collezione di stringhe
- I principali attributi di un cookie sono:
 - **Nome=Valore**: il nome identifica univocamente un cookie all'interno di un ambito definito da **domain e path** mentre il valore rappresenta il dato effettivamente memorizzato nel cookie (è una stringa di max 255 caratteri)
 - **Domain**: quale dominio può accedere al cookie
 - **Path**: definisce il percorso che può accedere al cookie
 - **Max-age**: numero di secondi di vita del cookie (in alternativa si può usare **Expires** per definire quando il cookie cesserà di esistere)
 - **Secure**: garantisce che il cookie venga inviato solo tramite HTTPS
 - **HttpOnly**: Impedisce l'accesso al cookie da parte di JavaScript (utile contro attacchi XSS (Cross-site scripting))
 - **SameSite**: Controlla il comportamento nelle richieste cross-site (utile contro attacchi CSRF (Cross-site request forgery))

HEADER DEI COOKIE

- I cookies usano due header, uno per la risposta, ed uno per richieste successive:
 - **Set-Cookie:** header della risposta (**response**), il client può memorizzarlo e rispedirlo alla prossima richiesta
 - **Cookie:** header della richiesta (**request**). Il client decide se spedirlo sulla base del nome del documento, dell'indirizzo IP del server, e dell'età del cookie

HEADER DEI COOKIE

- Il server invia un header **Set-Cookie** in una risposta HTTP:

```
HTTP/1.1 200 OK
```

```
Set-Cookie: user=JohnDoe; Path=/; HttpOnly; Secure
```

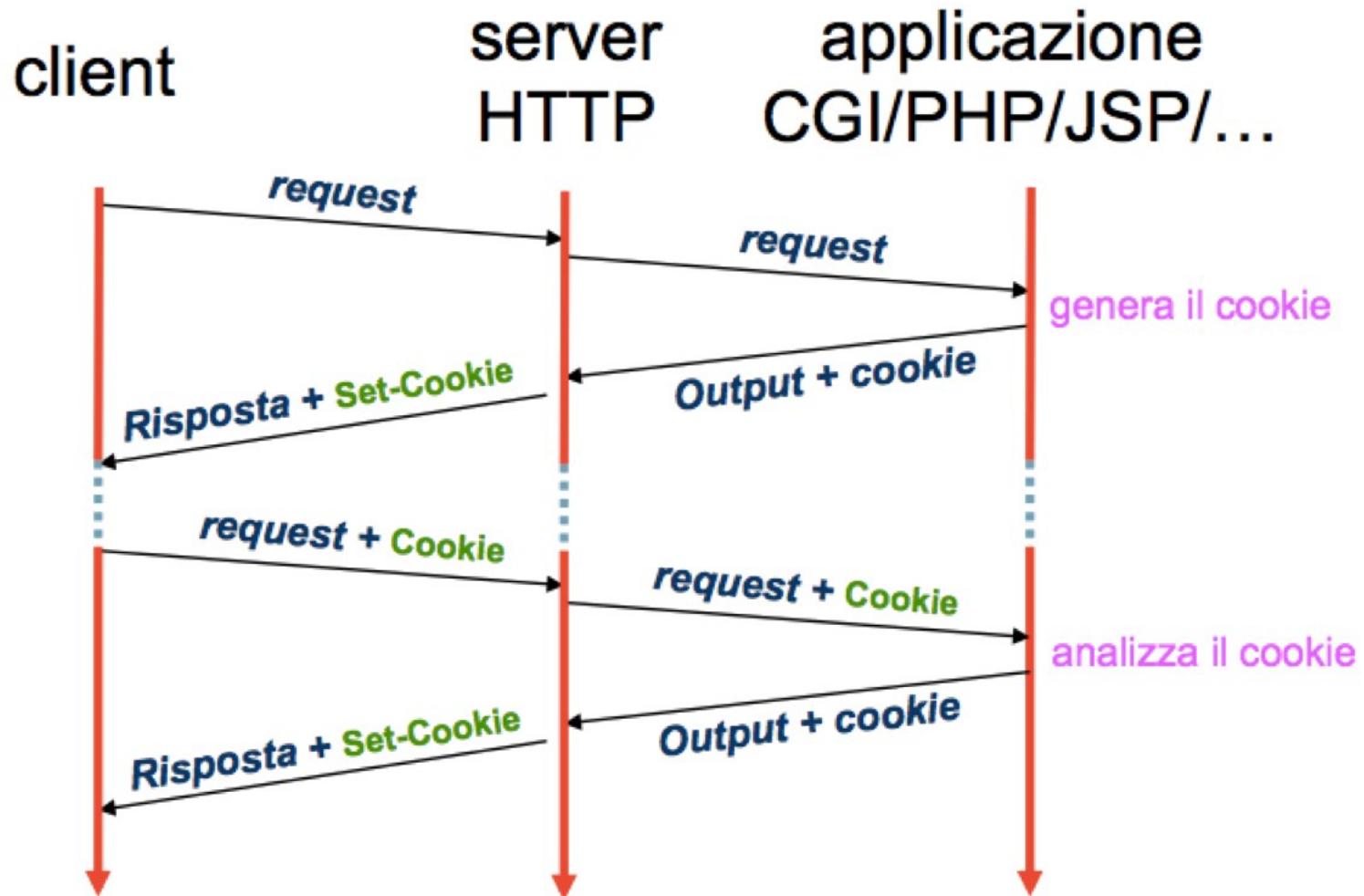
- Nelle richieste successive allo stesso dominio, il browser include automaticamente il cookie nell'header **Cookie**:

```
GET /dashboard HTTP/1.1
```

```
Host: example.com
```

```
Cookie: user=JohnDoe
```

INTERAZIONE CON I COOKIE



AUTENTICAZIONE

- Esistono situazioni in cui si vuole restringere l'accesso alle risorse ai soli utenti abilitati
- Tecniche comunemente utilizzate
 - Filtro su set di indirizzi IP
 - HTTP Basic
 - HTTP Digest
 - Form per la richiesta di username e password

AUTENTICAZIONE

- Basare l'autenticazione sull'indirizzo IP del client è una soluzione che presenta vari svantaggi:
 - Non funziona se l'indirizzo non è pubblico (vedi esempio dei NAT (Network Address Translation))
 - Non funziona se l'indirizzo IP è assegnato dinamicamente (es. DHCP)
 - Esistono tecniche che consentono di presentarsi con un IP fasullo (spoofing)
- L'autenticazione HTTP Digest è caduta in disuso negli ultimi anni (invio di password dopo la codifica con una funzione hash (es. MD5))
- Normalmente si usano
 - **HTTP Basic**
 - **Form**

AUTENTICAZIONE HTTP BASIC

- **Challenge (da parte del server):**

HTTP/1.1 401 Authorization Required

WWW-Authenticate: Basic realm="Secure Area"

- **Response (da parte del cliente):**

GET /private/index.html HTTP/1.1

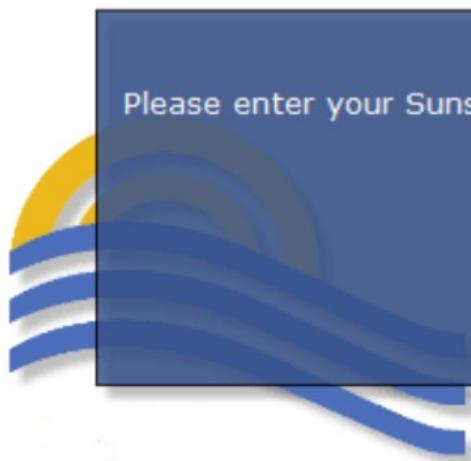
Host: localhost

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==



AUTENTICAZIONE FORM

- Normalmente si usa il metodo POST



Please enter your Sunshine Connections Username and Password below:

Username:

Password:

SICUREZZA (dei DATI)

- Proprietà desiderabili:

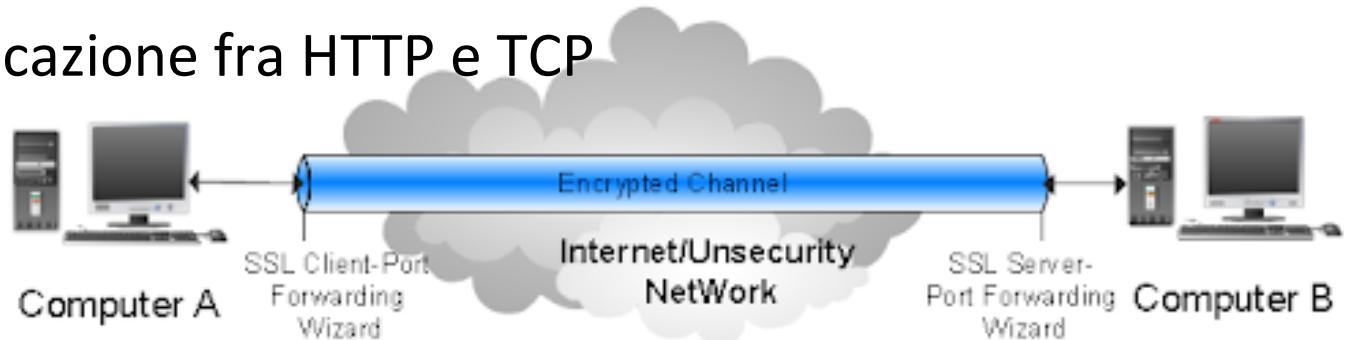
- Confidenzialità (i dati e le risorse siano preservati dal possibile utilizzo o accesso da parte di soggetti non autorizzati)
- Integrità (mantenere la veridicità dei dati e delle risorse e garantire che non siano in alcun modo modificate o cancellate–se non da soggetti autorizzati)
- Autenticità (il sistema deve garantire la paternità dei dati)
- Non ripudio
 - Non ripudio della sorgente: prova chi è il mittente dei dati in una transazione
 - Non ripudio della destinazione: prova che i dati sono arrivati ad uno specifico destinatario

- Sicurezza del canale di trasporto:

- **SSL** (Secure Sockets Layer)
- **TLS** (Transport Layer Security)
 - Sostituisce SSL
 - È alla base di HTTPS

SSL/TSL

- Viene aggiunto un livello che si occupa della gestione di *confidenzialità, autenticità ed integrità* della comunicazione fra HTTP e TCP



- Si accede tramite <https://...>
- Basato su crittografia a chiave pubblica:
 1. private key + public key
 2. certificato (in genere usato per autenticare il server)

USER AGENT CACHE

- Lo **user agent** (tipicamente il browser) mantiene una cache delle pagine visitate dall'utente
- L'uso delle user agent cache era molto importante in passato quando gli utenti non avevano accesso a connessioni di rete a banda larga

HTTP AND CACHE

- HTTP definisce vari meccanismi per la gestione delle cache
 - **Freshness:** il server indica per quanto tempo una risposta HTTP nella cache è da considerarsi «fresca», fintantoché la risposta è fresca il client può riusarla
 - **Validation:** le risposte HTTP obsolete in cache non vengono immediatamente scartate dal client. HTTP ha un meccanismo, detto validation, per trasformare una risposta obsoleta in una nuova «fresca»
 - **Invalidation:** è il processo di rimozione/riaggiornamento di un risposta HTTP nella cache quando diventa obsoleta prima della scadenza naturale

HTML

WWW = URL + HTTP + HTML

- The server usually sends some type of content to the browser so that the browser can display it
- Servers often send the browser a set of instructions written in HTML, the **HyperText Markup Language**
- The HTML tells the browser how to present the content to the user
- HTML has dozens of **tags** and hundreds of tag **attributes**. The goal of HTML is to take a text document and add tags that tell the browser how to format the text

Tag	Description
<!-- -->	where you put your <i>comments</i>
<a>	<i>anchor</i> - usually for putting in a hyperlink
<align>	<i>align</i> the contents left, right, centered, or justified
<body>	define the boundaries of the document's <i>body</i>
 	a <i>line break</i>
<center>	<i>center</i> the contents
<form>	define a <i>form</i> (which usually provides input fields)
<h1>	the first level <i>heading</i>
<head>	define the boundaries of the document's <i>header</i>
<html>	define the boundaries of the HTML <i>document</i>
<input type>	defines an <i>input widget</i> to a form
<p>	a new <i>paragraph</i>
<title>	the HTML document's <i>title</i>

```
<html>  
  <!-- Some sample HTML -->
```

An HTML comment

(A) <head>
 <title>A Login Page</title>

</head>

<body>

(B) <h1 align="center">Skyler's Login Page</h1>

<p align="right">

(C)
</p>

<form action="date2"> The servlet to send
 the request to.

(D) Name: <input type="text" name="param1"/>

 Password: <input type="text" name="param2"/>
>

<center>
 <input type="SUBMIT"/>

(E) </center>
</form>

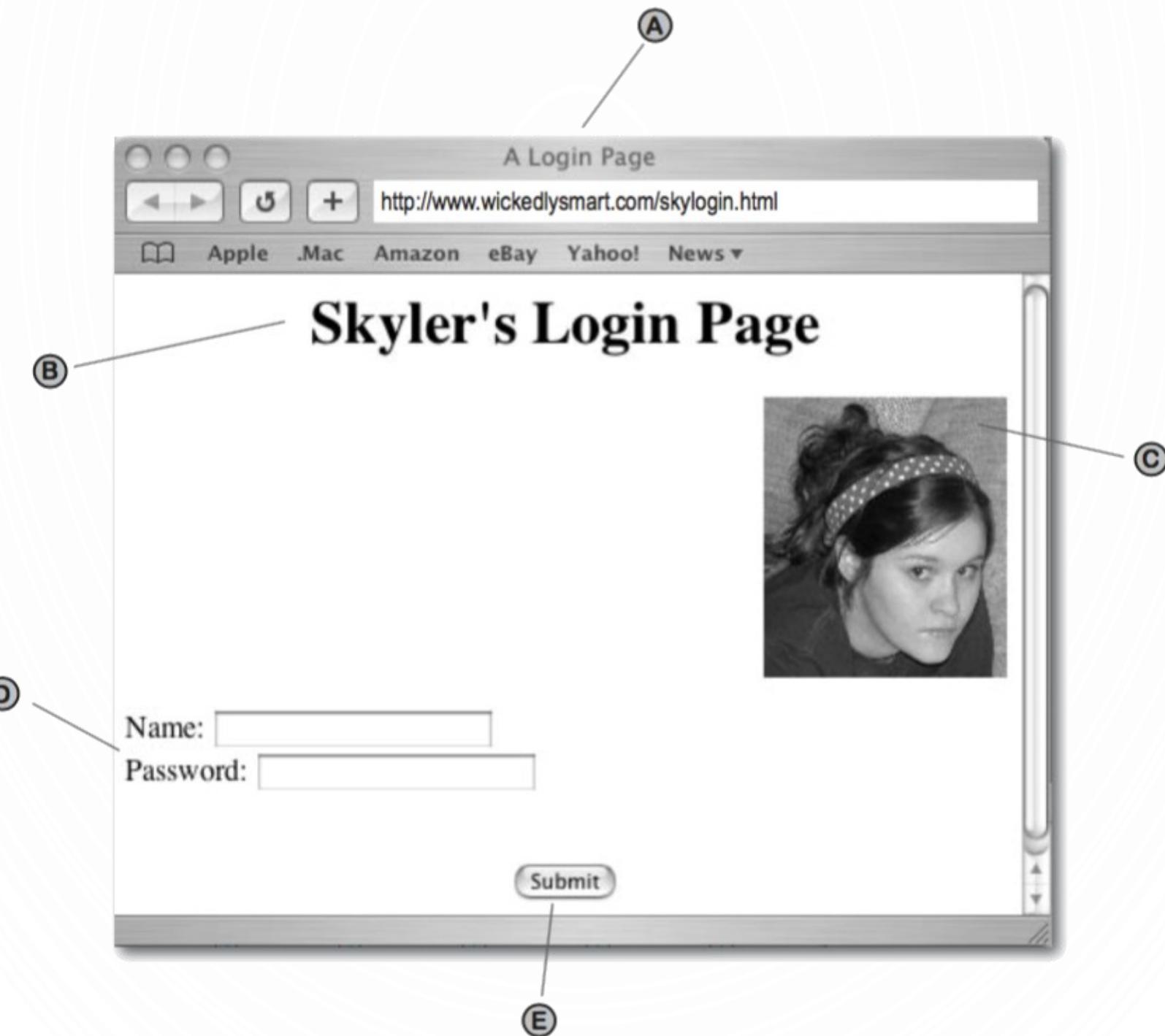
</body>

</html>

We'll talk more about
forms later, but briefly,
the browser can collect
the user's input and
return it to the server.

The "submit" button
in the form.

The
 tags
cause line breaks.





GREAT JOURNEY BEGINS WITH A SMALL STEP

Make Your Life Better and Bright! You must trip with Us!

[LEARN MORE](#)[HOTELS](#)[TOURS](#)[TICKETS](#)

PLACE / HOTEL NAME

EXAMPLE FRANCE

CHECK IN

CHECK OUT

MM/DD/YY



MM/DD/YY



ROOMS

ADULT

CHILD

[SEARCH](#)[ADVANCED SEARCH OPTIONS](#)

ANDRASSY THAI HOTEL
LOCATION: THAILAND

152\$
AVG/NIGHT

TODAY TUESDAY WEDNESDAY

15 M/S 5 M/S 3 M/S

FIND OUR SPECIAL OFFERS FOR
BROOKLYN

GREAT JOURNEY BEGINS WITH A SMALL STEP

Make Your Life Better and Bright! You must trip with Us!

[LEARN MORE](#)

< >

HOTELS TOURS TICKETS

PLACE / HOTEL NAME

EXAMPLE:FRANCE

CHECK IN

CHECK OUT

MM/DD/YY

MM/DD/YY

ROOMS

ADULT

CHILD

--
--
--

--
--
--

--
--
--

ANDRASSY THAI HOTEL

152\$

SEARCH

ADVANCED SEARCH OPTIONS



GREAT JOURNEY BEGINS WITH A SMALL STEP

Make Your Life Better and Bright! You must trip
with Us!

[LEARN MORE](#)

HOTELS

TOURS

TICKETS

PLACE / HOTEL NAME

EXAMPLE:FRANCE

CHECK IN

CHECK OUT

MM/DD/YY



MM/DD/YY



ROOMS

ADULT

--



--



CHILD

--

[SEARCH](#)

ADVANCED SEARCH OPTIONS