CORSO DI LAUREA IN INFORMATICA

PROGRAMMAZIONE WEB

Overview: URI, HTTP, HTML

a.a 2017-2018

URL

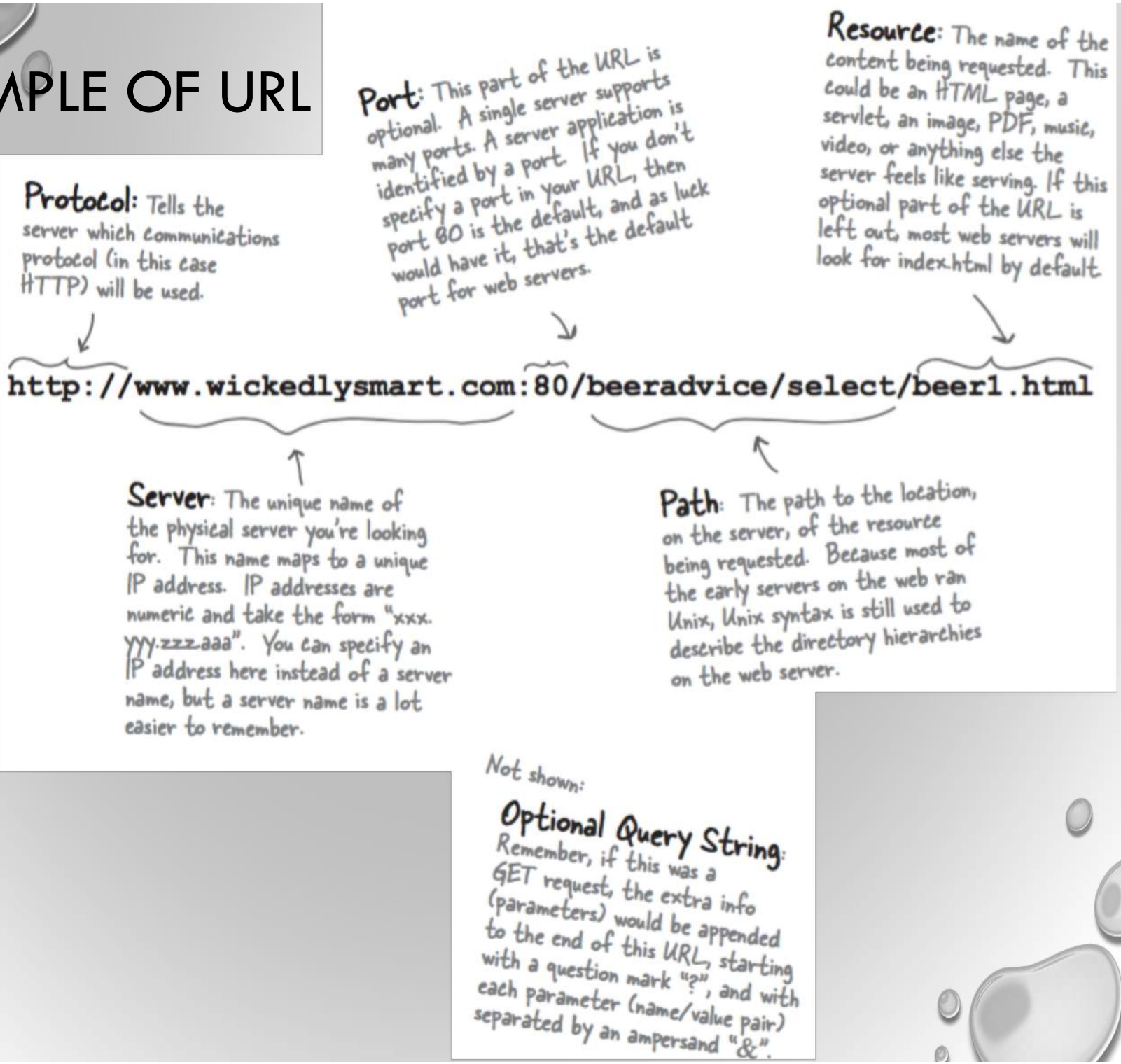
- Every resource on the web has its own **unique** address, in the URL (Uniform Resource Locators) format

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 URI

EXAMPLE OF URL



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

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 identificatori 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>

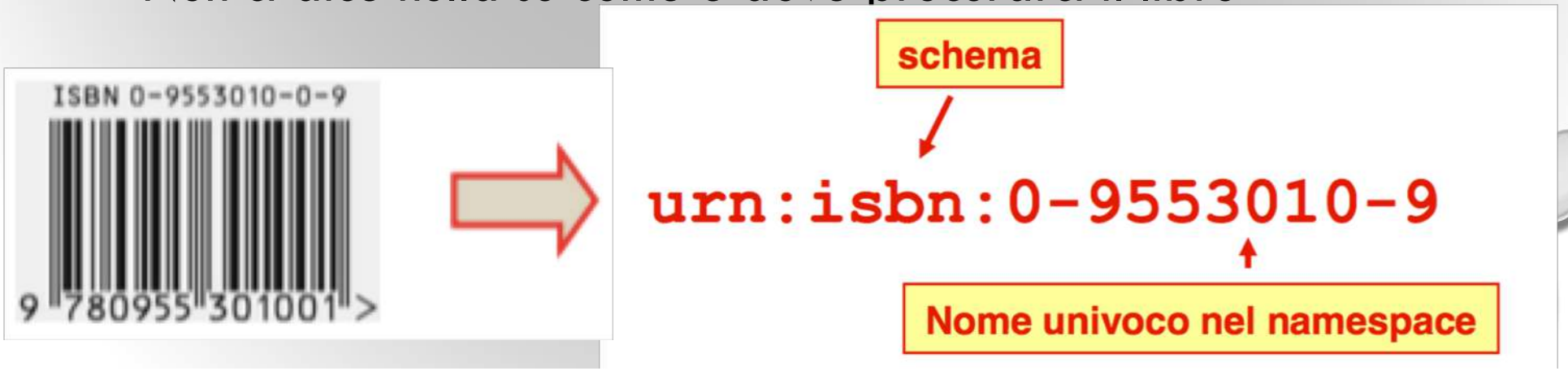
- A parte **<scheme>**, le altre parti possono talora essere omesse, come nei casi in cui non è inclusa la componente **<authority>** o non è inclusa la componente **<query>**

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 procurarci il libro



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>]

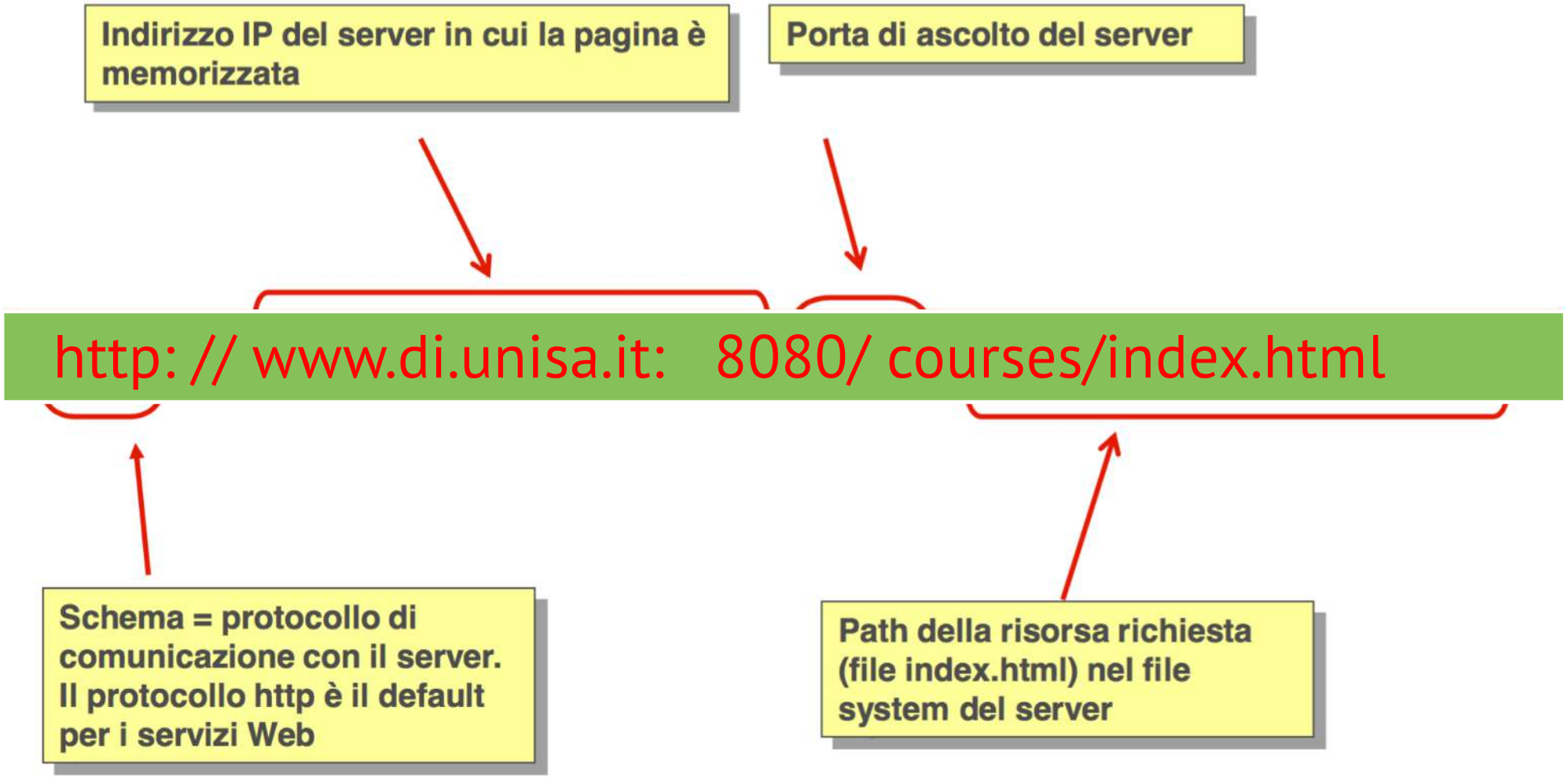
  [/<path>[?<query>][#fragment]]
- Questa forma vale per diversi protocolli di uso comune: HTTP, HTTPS, FTP, WAP, ...
- Ma non, ad esempio, per la posta elettronica

  mailto:info@example.com?
  subject=subject&cc=cc@example.com&body=Hello

COMPONENTI DI UN URL CON SCHEMA HTTP-LIKE

- **<protocol>**: 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:** 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



URL OPACHE E GERARCHICHE

- Le URL sono anche classificate come opache o gerarchiche
  - **URL opaca**: non è soggetta a ulteriori operazioni di parsing
    - **mailto:francese.rita@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:** processo di rimozione dei segmenti "." e ".. " (e altri caratteri speciali) dal path di una URI gerarchica
  - Normalizzazione si applica solo a URI gerarchiche, su URI opache non ha effetto
- **Risoluzione:** è il processo che a partire da una URI originaria porta all'ottenimento di una URI risultante
  - La URI originaria viene risolta basandosi su una terza URI, detta base URI
- **Relativizzazione** è il processo inverso alla risoluzione
- Esempio:
  - 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**

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

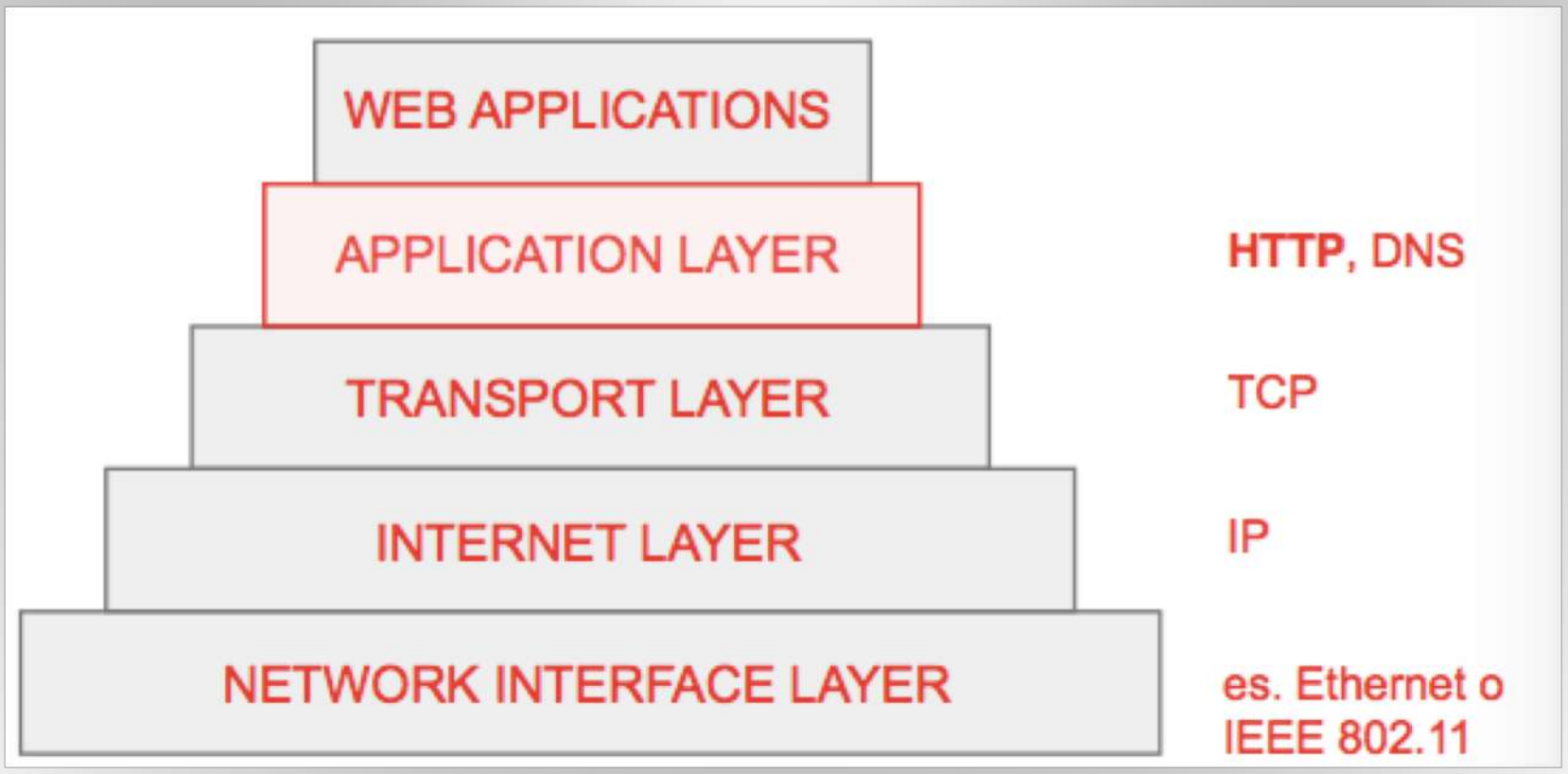
- HTTP è l'acronimo di HyperText Transfer Protocol
- È il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) da server a client
- Gestisce sia le richieste (URL) inviate al server che le risposte inviate al client (pagine)
- È un protocollo **stateless**: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

HTTP NELLO STACK TCP/IP

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



WHAT IS THE HTTP PROTOCOL?

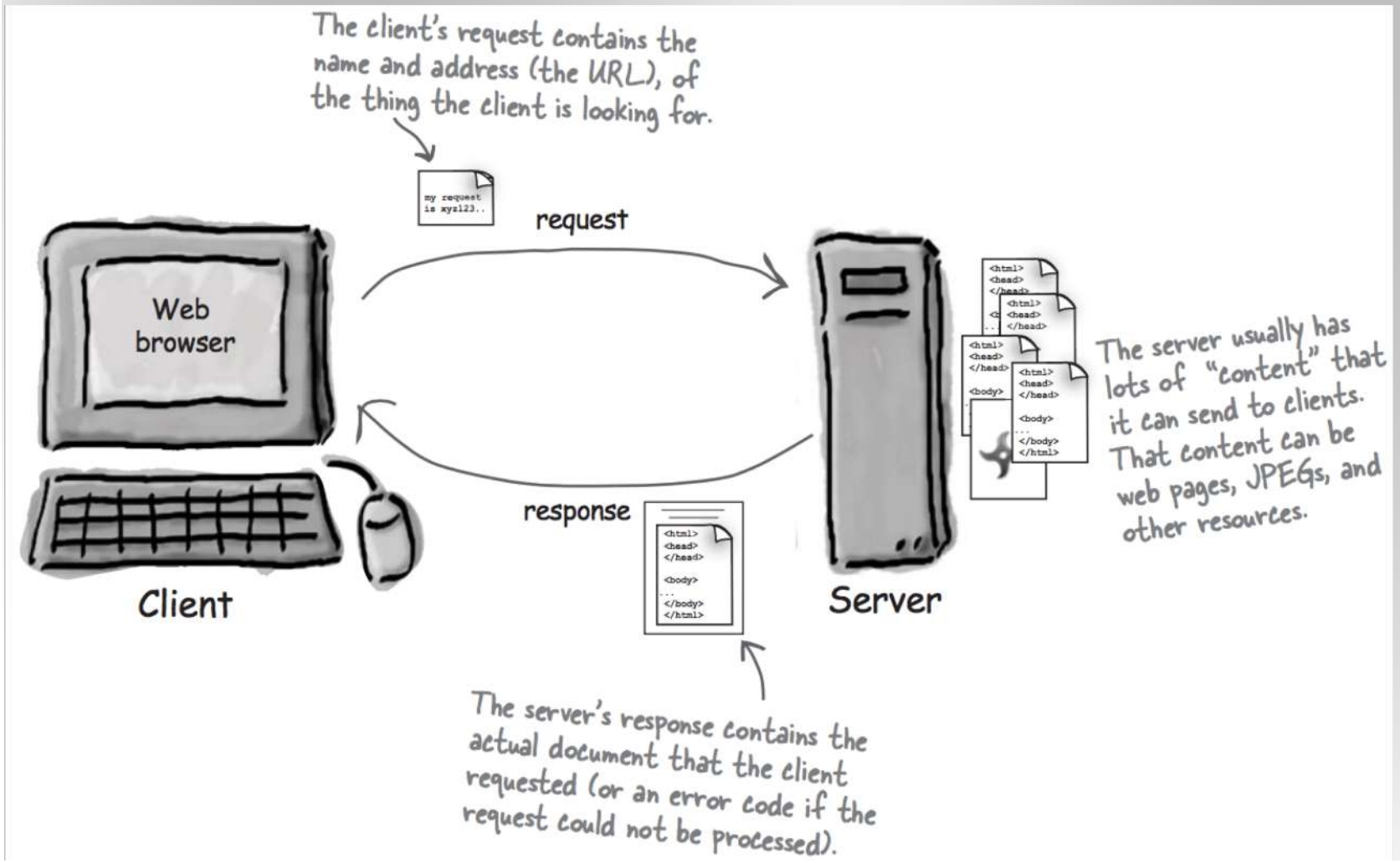
- **TCP** is responsible for making sure that a file sent from one network node to another ends up as a complete file at the destination, even though the file is split into chunks when it's sent
- **IP** is the underlying protocol that moves/routes the chunks (packets) from one host to another on their way to the destination
- **HTTP** is another network protocol that has Web-specific features, but it depends on TCP/IP to get the complete request and response from one place to another

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

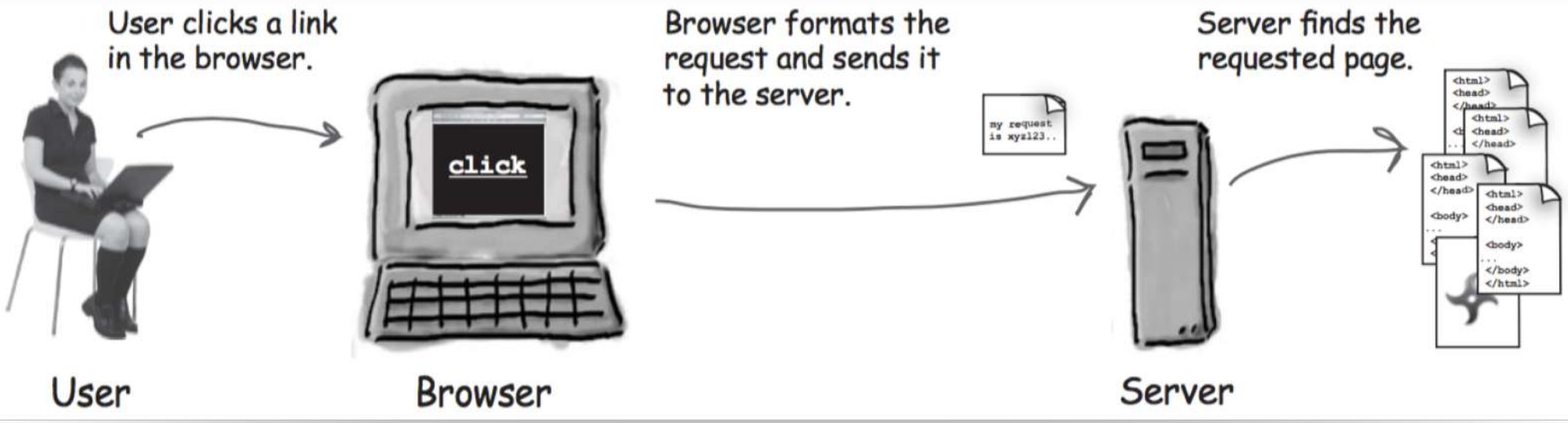
WHAT DOES YOUR WEB SERVER DO?

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

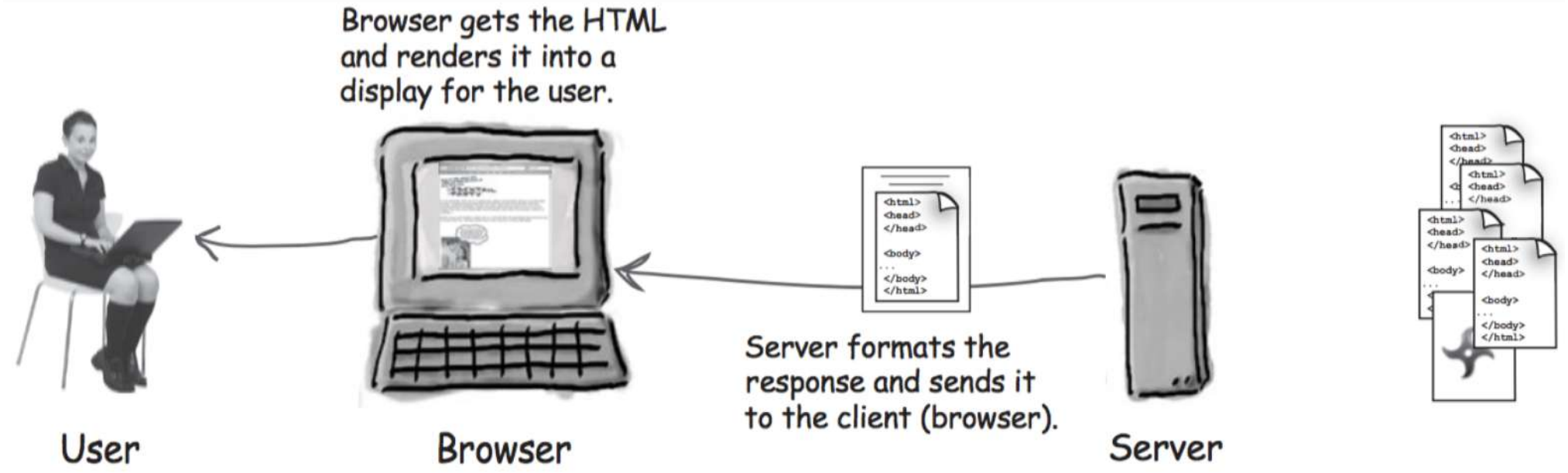


WHAT DOES A WEB CLIENT DO?

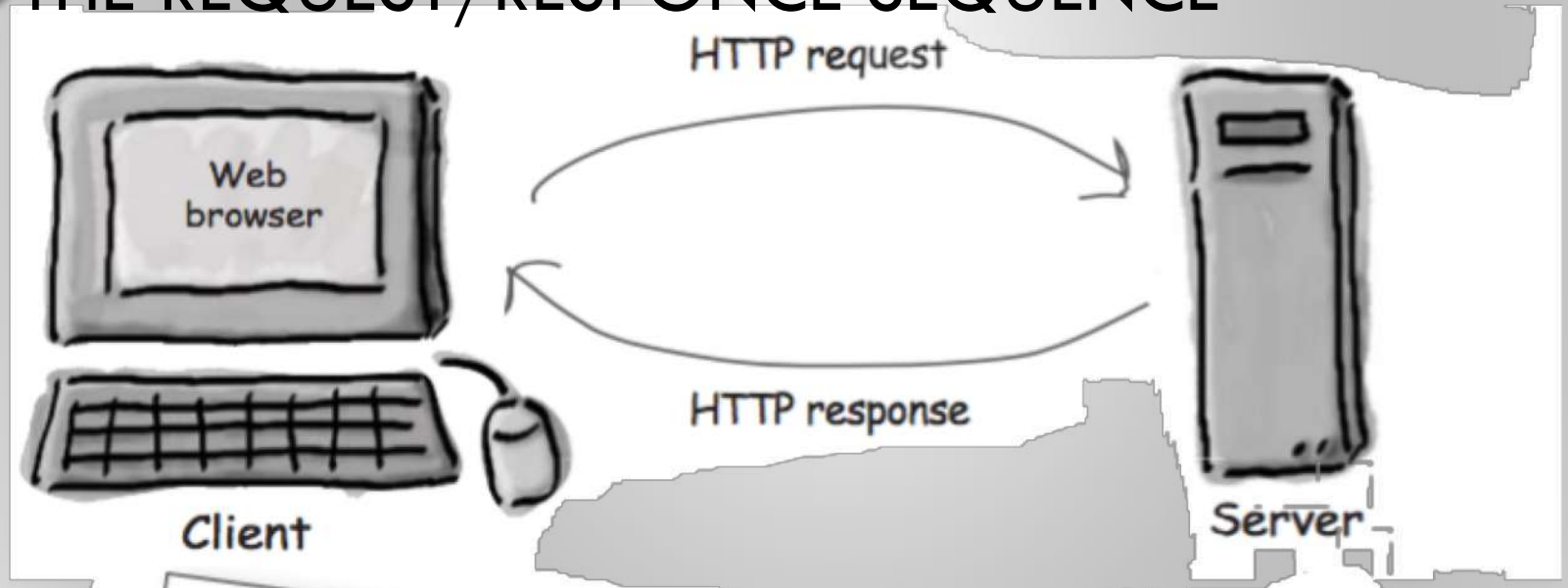
A web client lets the user request something on the server, and shows the user the result of the request



WHAT DOES A WEB SERVER DO?



THE REQUEST/RESPONCE SEQUENCE



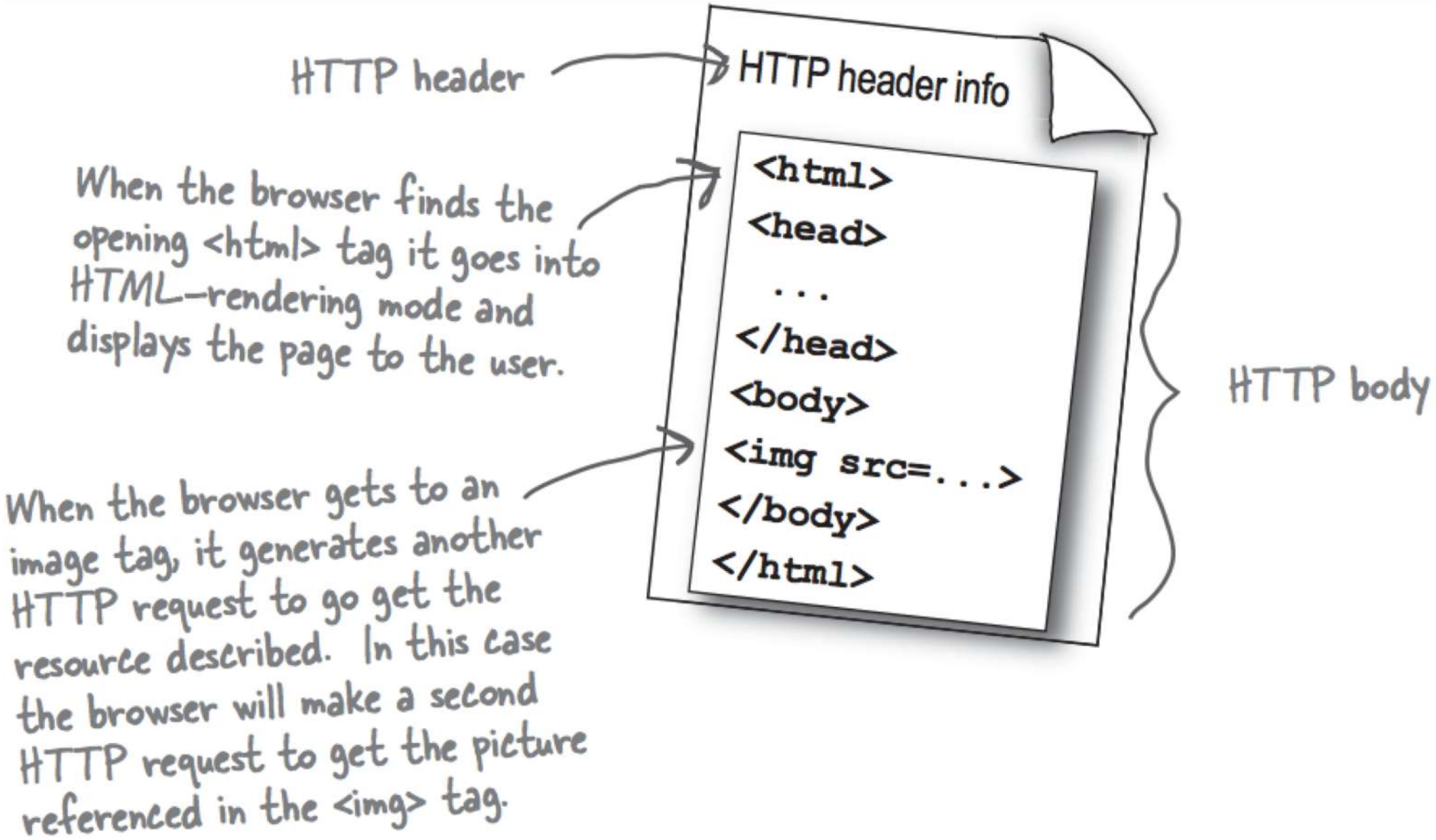
Key elements of the **request** stream:

- 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:

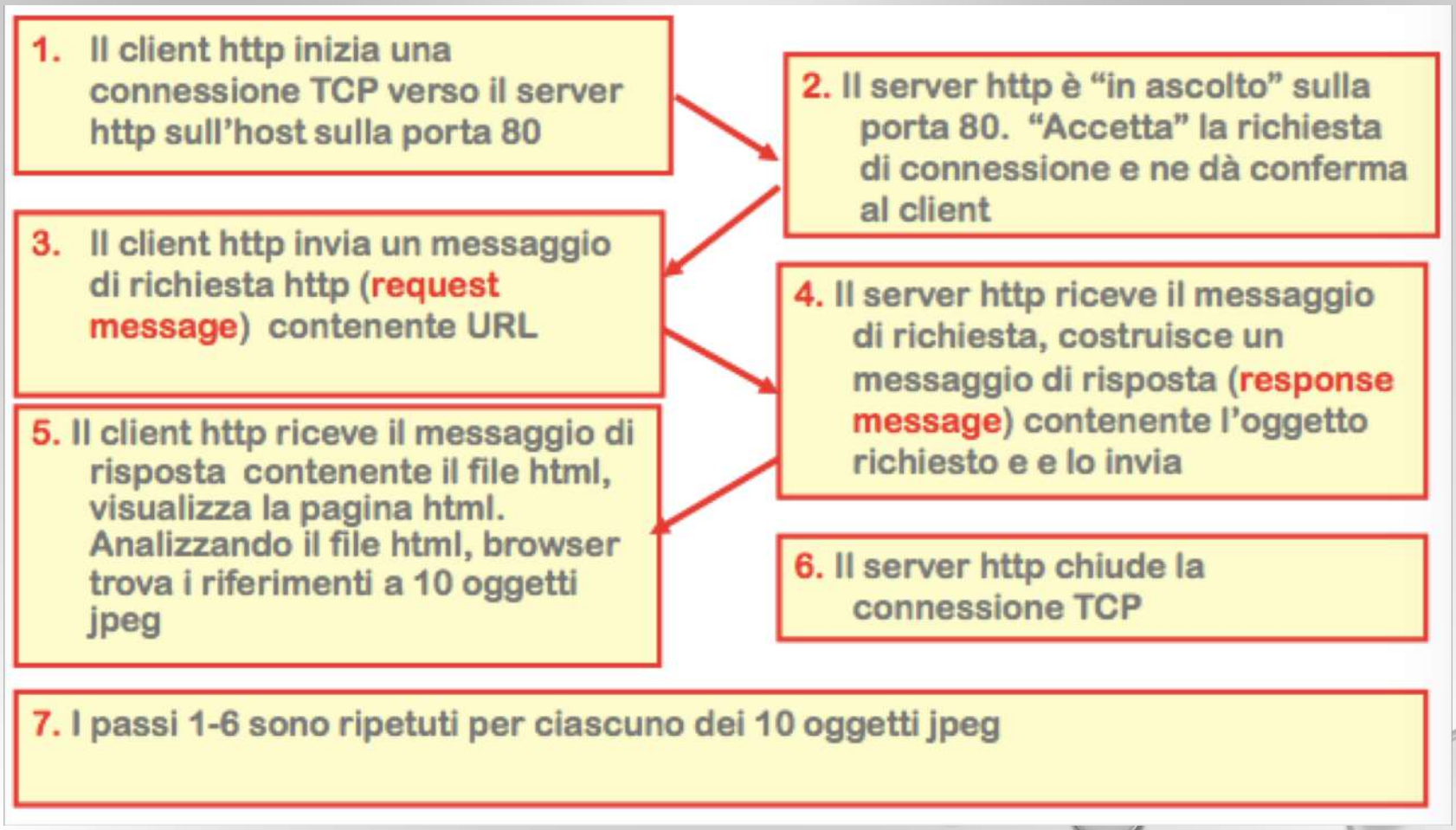
- A status code (for whether the request was successful)
- Content-type (text, picture, HTML, etc.)
- The content (the actual HTML, image, etc.)

HTML IS PART OF THE HTTP RESPONSE



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

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



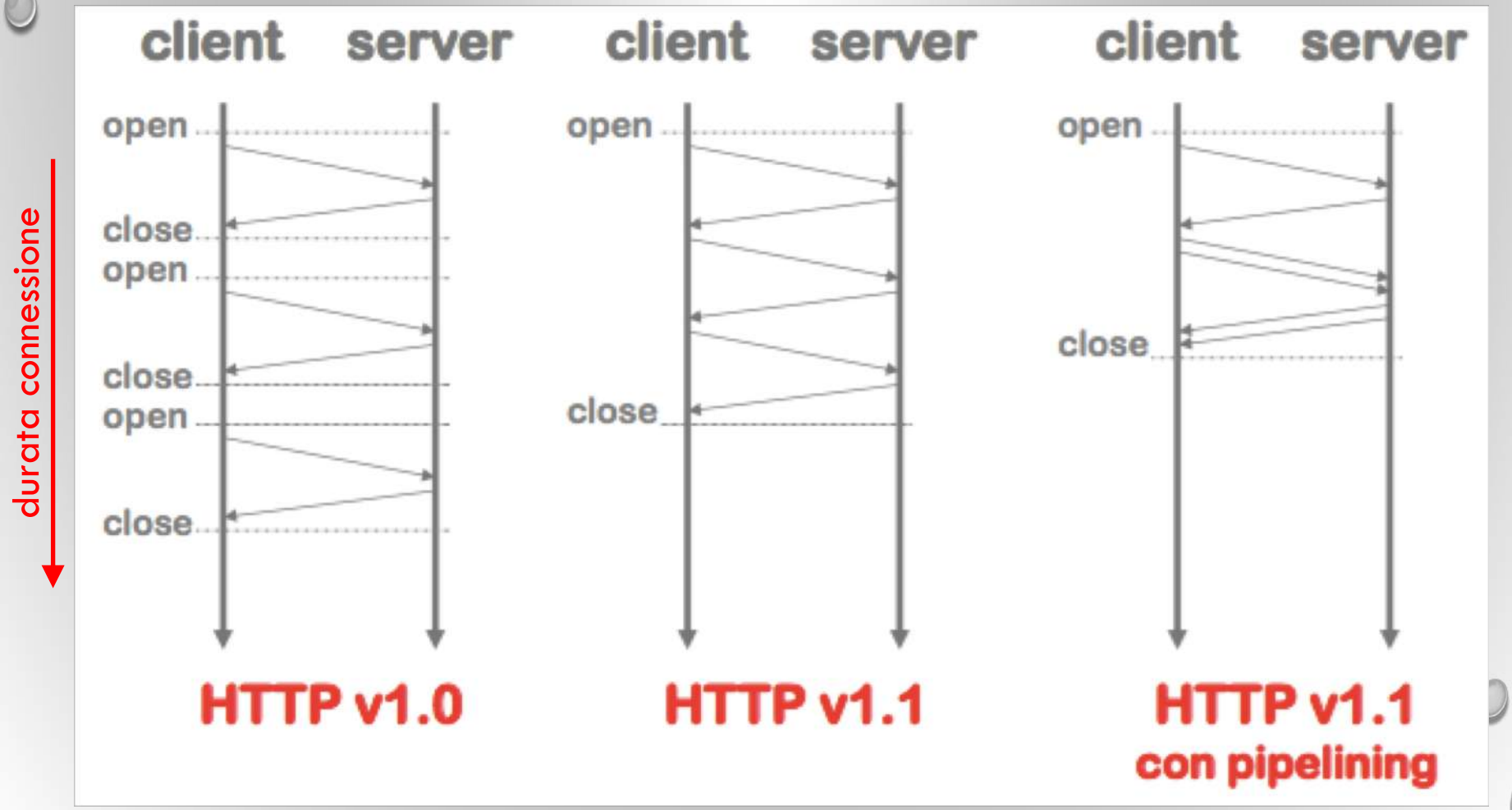
DIFFERENZE FRA HTTP V1.0 E V1.1

- 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 usa 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)

TIPI DI CONNESSIONE



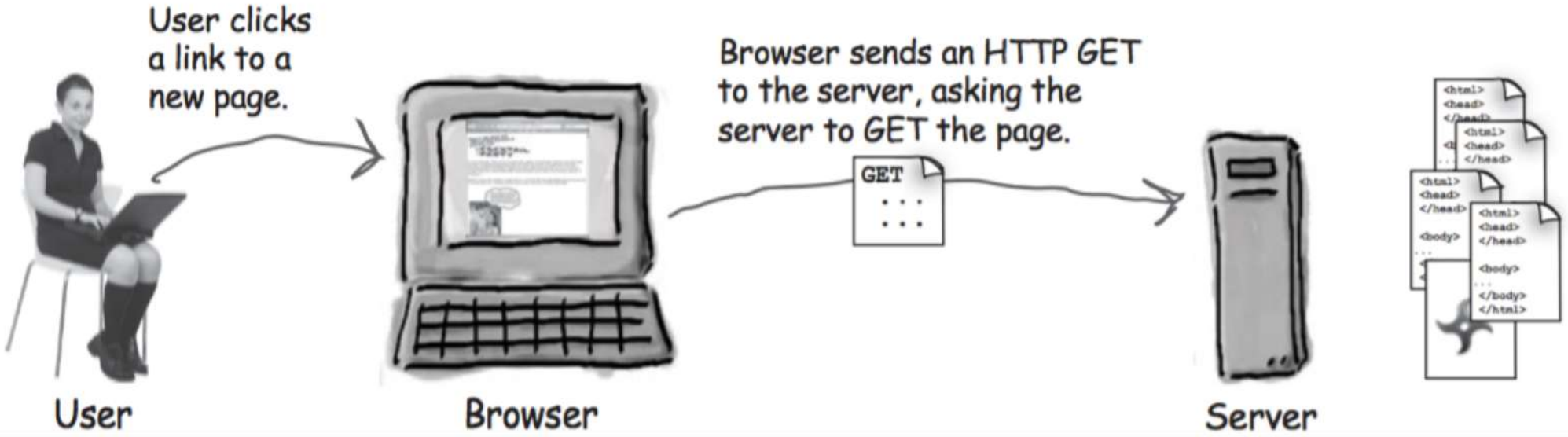
HTTP 2.0 O HTTP 2

- 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
    - 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 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, richiedendo di continuo le nuove risorse di cui ha necessità

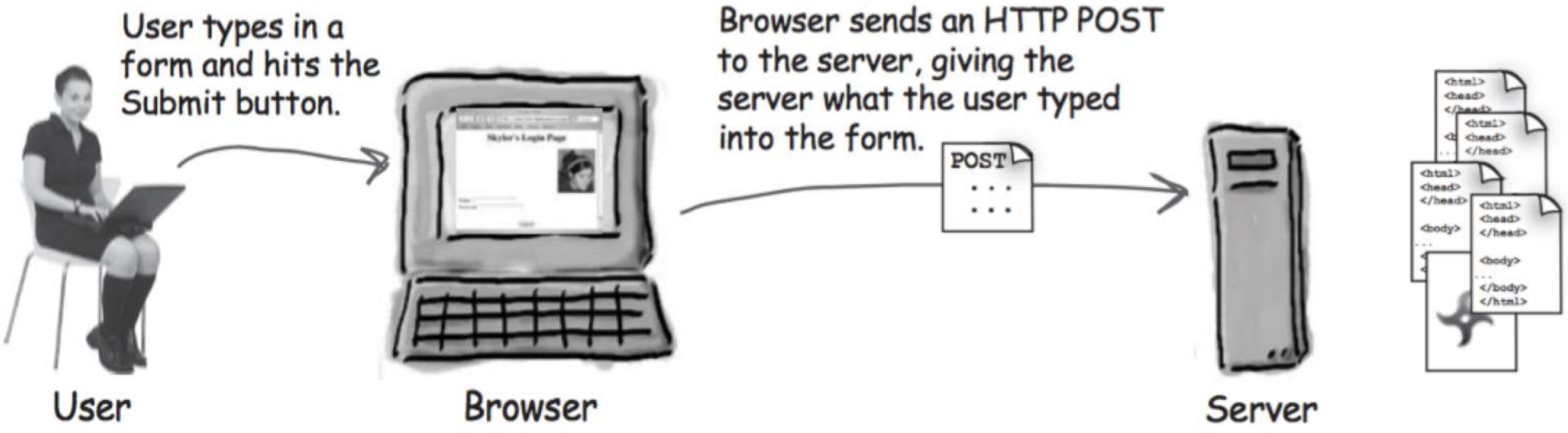
WHAT'S IN THE REQUEST?

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

POST


User types in a form and hits the Submit button.
Browser sends an HTTP POST to the server, giving the server what the user typed into the form.
POST
User
Browser
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

- 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 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 vedremo presto…)

- *Si ha una trasmissione di informazioni client → server che però non porta alla creazione di una risorsa sul server*

GET & POST: DIFFERENCIES

- **GET** is the simplest HTTP method
  - It asks the server to get a resource and send it back. That resource might be an HTML page, a JPEG, a PDF, etc.
- **POST** is a more powerful request
  - With POST, you can request something and at the same time send form data to the server

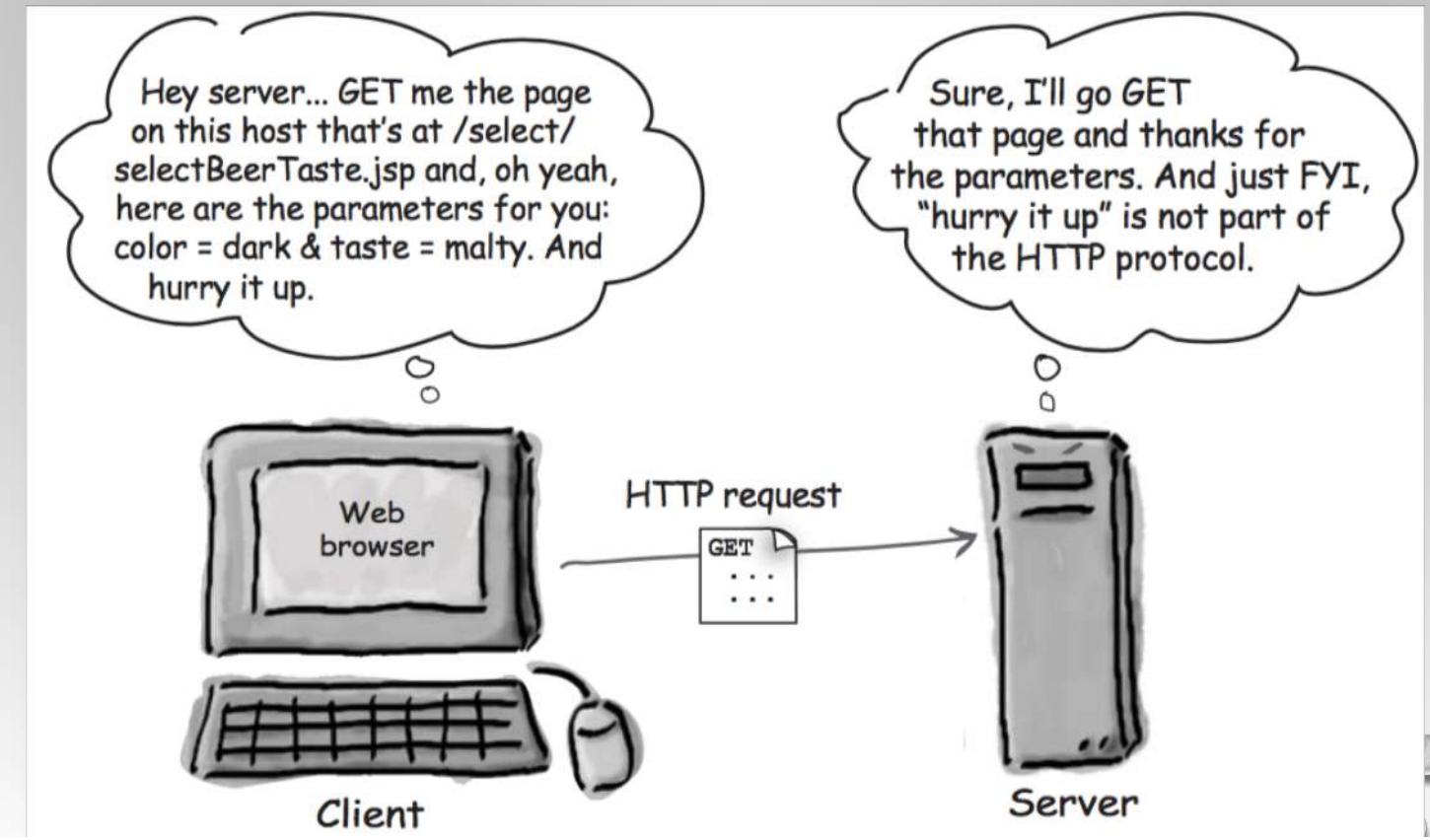
SEND A LITTLE DATA WITH HTTP GET

- 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/persone?nome=francese
  - The original URL before the extra parameter:
    - http://rubrica.unisa.it/persone
  - http://httpbin.org/get

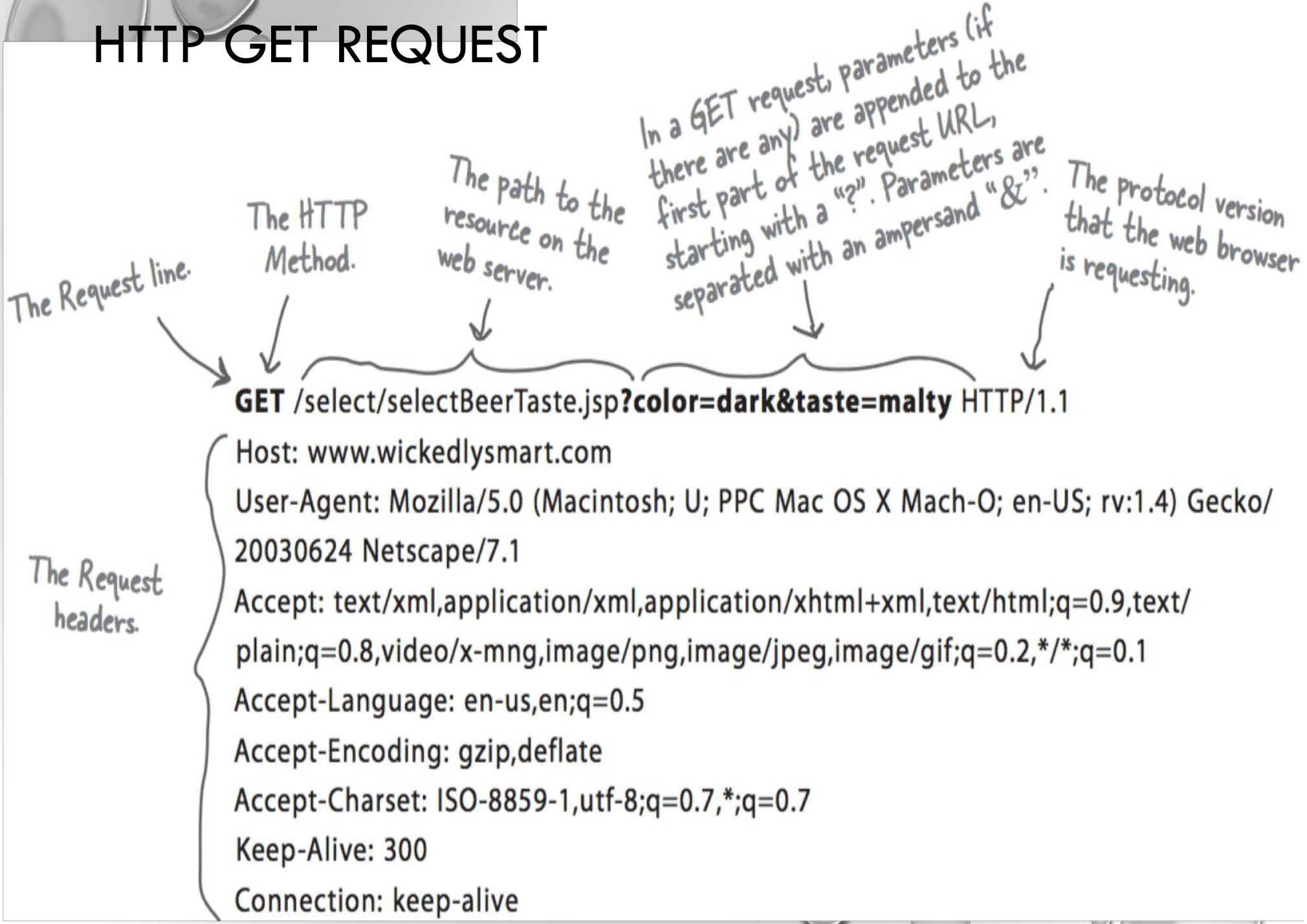
DETAILS ON HTTP GET REQUEST

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

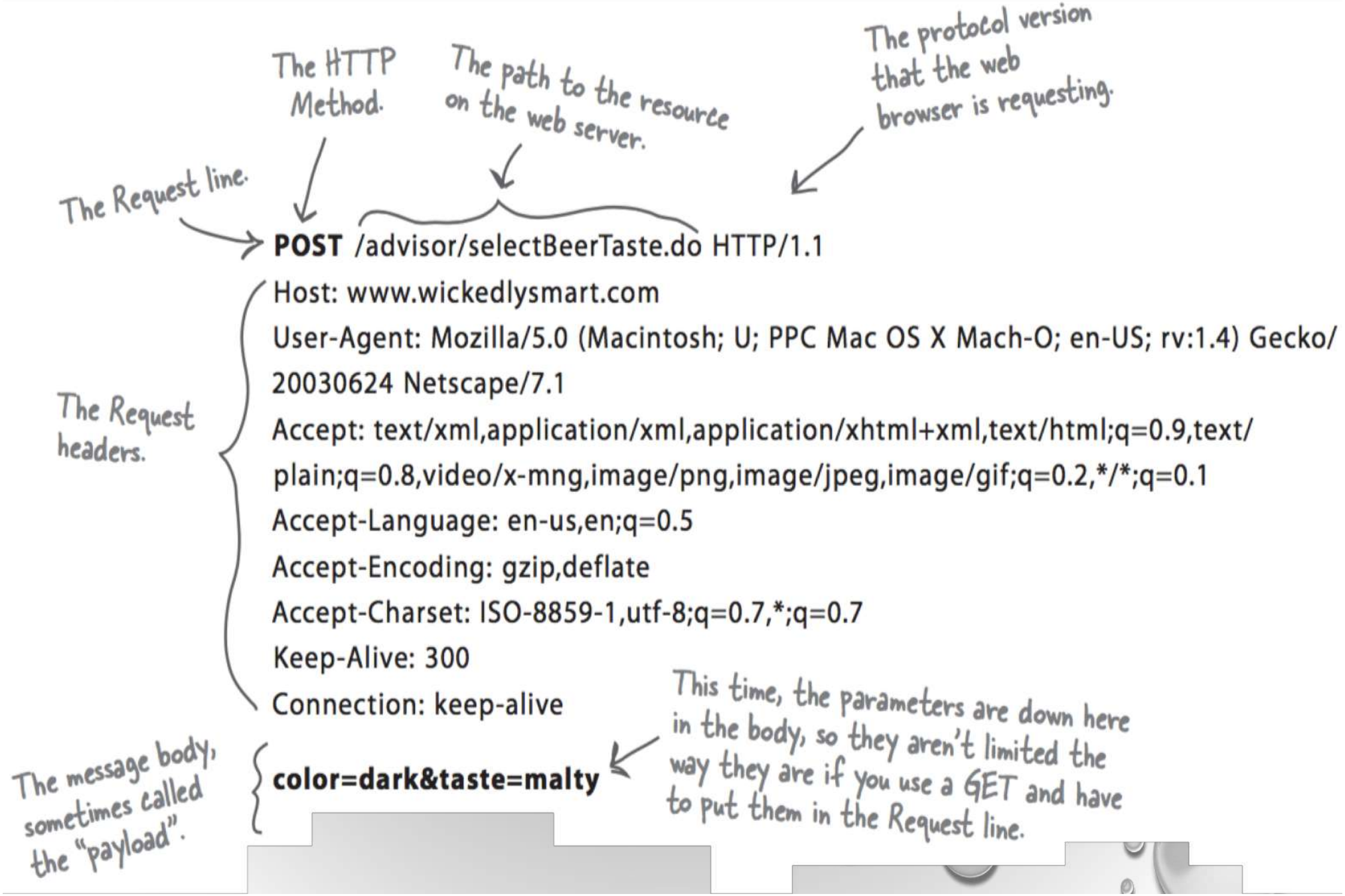
GET /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1



HTTP GET REQUEST



HTTP POST REQUEST



	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	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	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)
  - 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

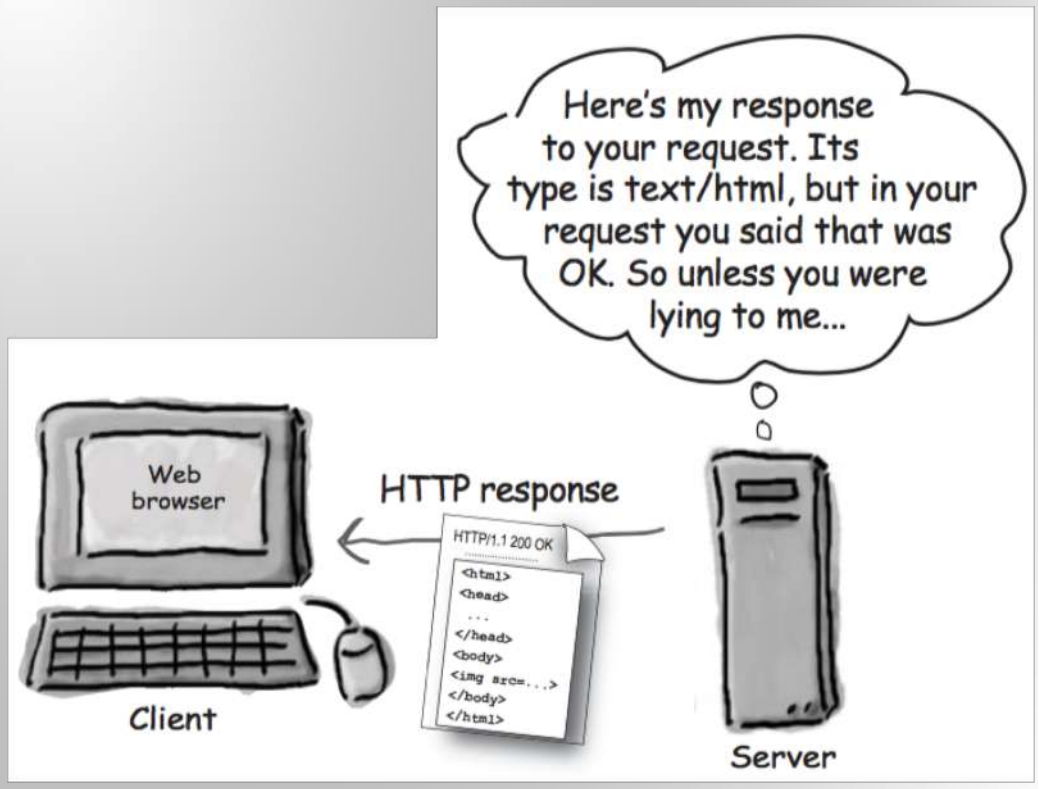
- **Sono normalmente disabilitati sui server pubblici**

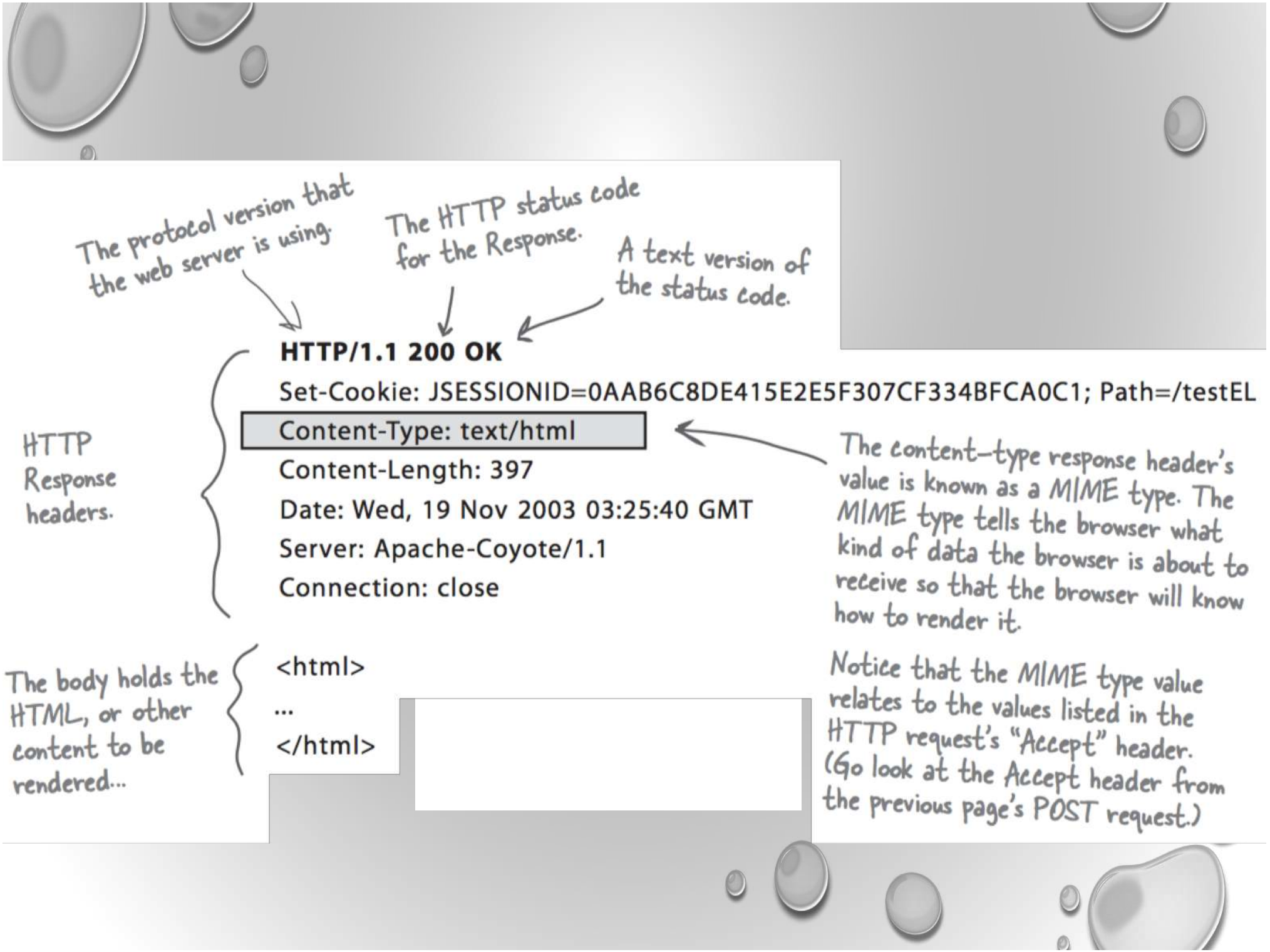
HEAD, OPTIONS AND TRACE

- **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




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

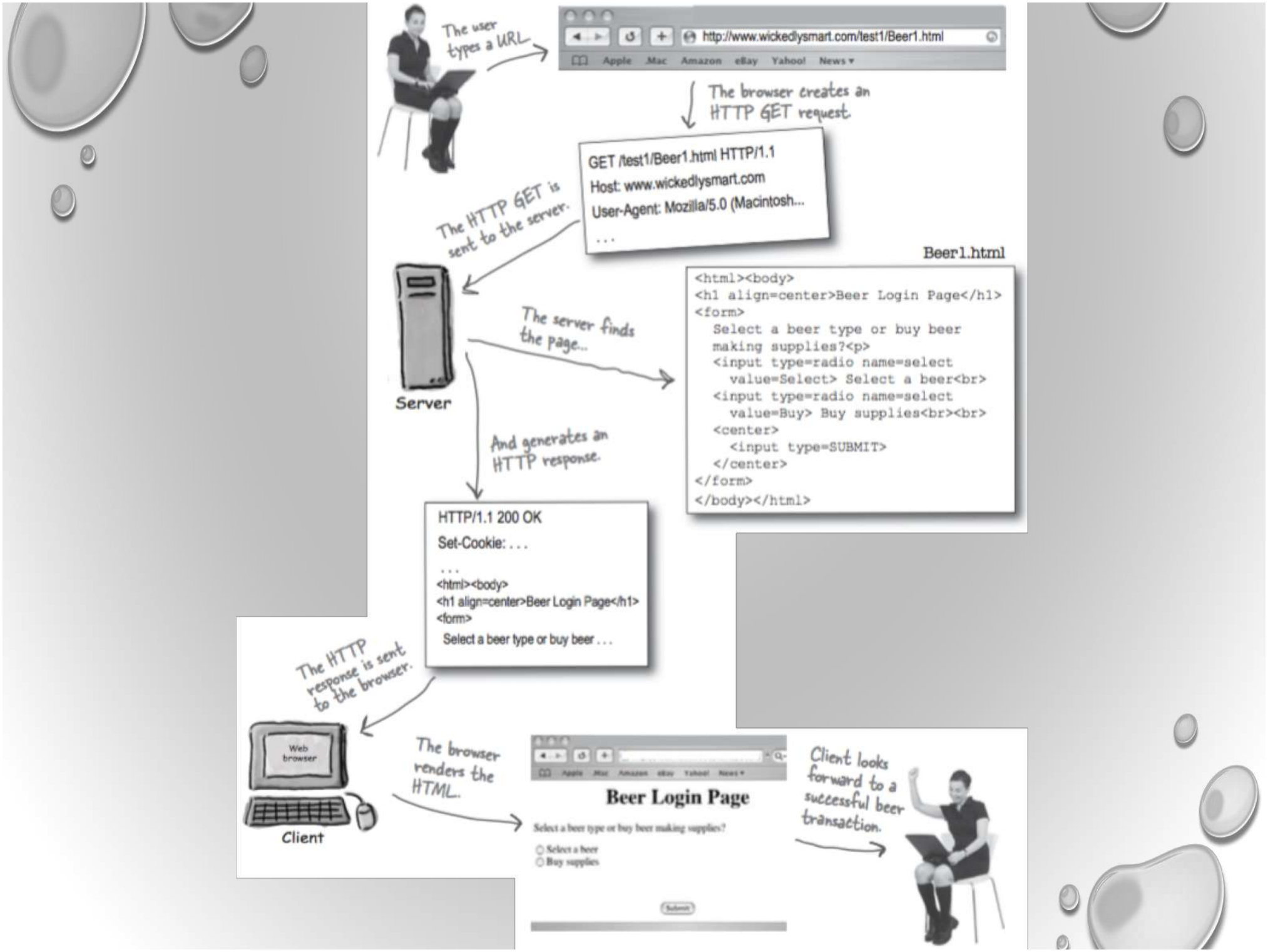
- È possibile estendere i tipi MIME esistenti con dei tipi sperimentali
- Il tipo o sottotipo inizia con una x-
  - video/x-nuovoformato
  - audio/x-wav
- I tipi sperimentali richiedono che il server invii al browser un'applicazione o plug-in per poter far interpretare correttamente i nuovi tipi di dati al browser

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 o di applicazioni CGI (Common Gateway Interface))

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)
- **400 Bad request** (errore sintattico nella richiesta)
- **401 Unauthorized** (manca l'autorizzazione)
- **403 Forbidden** (richiesta non autorizzabile)
- **404 Not found** (URL errato)
- **500 Internal server error** (tipicamente un CGI mal fatto)
- **501 Not implemented** (metodo non conosciuto dal server)


The user types a URL.
http://www.wickedlysmart.com/test1/Beer1.html
Apple Mac Amazon eBay Yahoo! News
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.
Server
The server finds the page...
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
<input type=radio name=select value=Buy> Buy supplies

<input type=SUBMIT>

</form>
</body></html>
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 ...
The HTTP response is sent to the browser.
Web browser
Client
The browser renders the HTML.
Beer Login Page
Select a beer type or buy beer making supplies?
Select a beer
Buy supplies
Submit
Client looks forward to a successful beer transaction.

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 possono essere generati sia dal client che dal server
- 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**

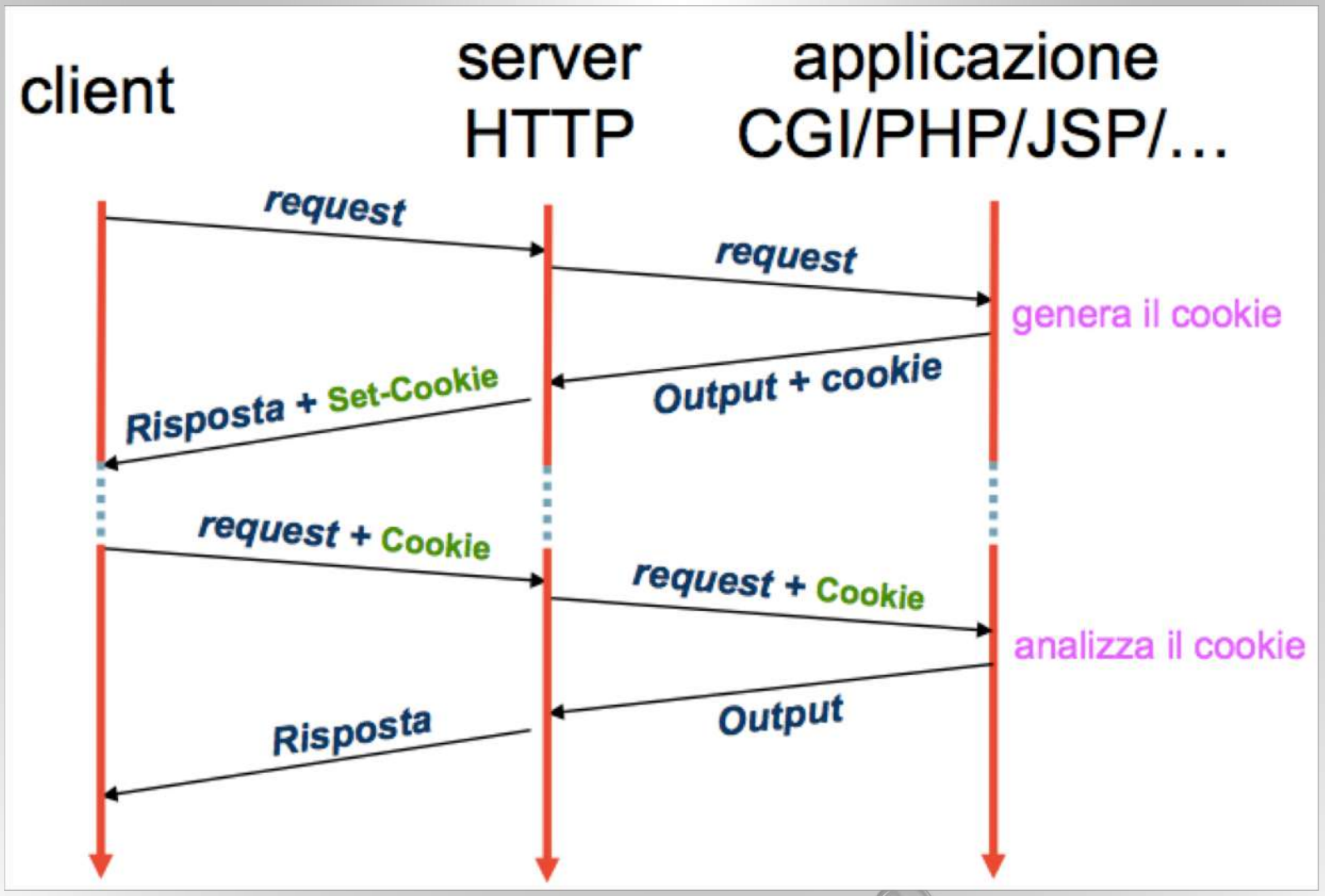
STRUTTURA DEI COOKIE

- I cookie sono una collezione di stringhe:
  - **Key**: identifica univocamente un cookie all'interno di un ***domain:path***
  - **Value**: valore associato al cookie (è una stringa di max 255 caratteri)
  - **Path**: posizione nell'albero di un sito al quale è associato (di default /)
  - **Domain**: dominio dove è stato generato
  - **Max-age**: (opzionale) numero di secondi di vita
  - **Secure**: (opzionale) non molto usato. Questi cookie vengono trasferiti se e soltanto se il protocollo è sicuro (**https**)
  - **Version**: identifica la versione del protocollo di gestione dei cookie

HEADER DEI COOKIE

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

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 Digest
  - Form per la richiesta di username e password
  - HTTP Basic

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 codifica con una funzione hash (es. MD5))

- Normalmente si usano
  - Form
  - HTTP Basic

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==
```

Testo in chiaro (username:password) codificato in Base64

Man is *TWFu*. Encoded in ASCII, the characters *M*, *a*, and *n* are stored as the bytes `77`, `97`, and `110`, v

AUTENTICAZIONE FORM

- Normalmente si usa il metodo POST

SICUREZZA

- Proprietà desiderabili:
  - Confidenzialità
  - Integrità — SSL/TLS
  - Autenticità
  - 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 **http**s://...

- Basato su crittografia a chiave pubblica:
  - private key + public key
  - 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
- Questo modello di caching è ora molto rilevante per i dispositivi mobili al fine di consentire agli utenti di lavorare con connettività intermittente

HTTP AND CACHE

- HTTP definisce vari meccanismi per la gestione delle cache
  - **Freshness**: controllata lato server da Expires response header e lato cliente da direttiva Cache-Control: max-age
  - **Validation**: può essere usato per controllare se un elemento in cache è ancora corretto, ad es. nel caso in cui sia in cache da molto tempo
  - **Invalidation**: è normalmente un effetto collaterale di altre request che hanno attraversato la cache
    - Se per esempio viene mandata una POST, una PUT o una DELETE a un URL il contenuto della cache deve essere e viene automaticamente invalidato

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

(Technically, the and <align> tags have been deprecated in HTML 4.0, but we're using them in some of our examples because it's simpler to read than the alternative, and you're not here to learn HTML anyway.)

```
<html>
<!-- Some sample HTML -->
<head>
  <title>A Login Page</title>
</head>
<body>
<h1 align="center">Skyler's Login Page</h1>

<p align="right">
  <img src="SKYLER2.jpg" width="130" height="150"/>
</p>

<form action="date2">
      Name: <input type="text" name="param1"/><br/>
  Password: <input type="text" name="param2"/><br/><br/><br/>

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

</body>
</html>
```

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

An HTML comment

The <img> tag is nested inside a paragraph <align> tag in order to place the image roughly where we want it. (Remember, <align> is deprecated, but we're using it because it's simple to read.)

The servlet to send the request to.

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

The
 tags cause line breaks.

The "submit" button in the form.

You need only the most basic HTML knowledge.


A
A Login Page
http://www.wickedlysmart.com/skylogin.html
Apple
.Mac
Amazon
eBay
Yahoo!
News
Skyler's Login Page
B
C
D
Name:
Password:
Submit
E

SPARROW

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

MM/DD/YY

CHECK OUT

MM/DD/YY

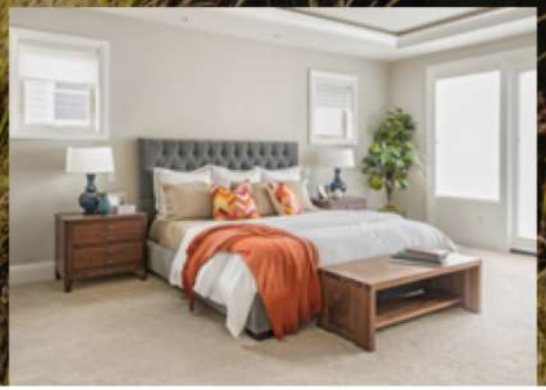
ROOMS

ADULT

CHILD

SEARCH

ADVANCED SEARCH OPTIONS

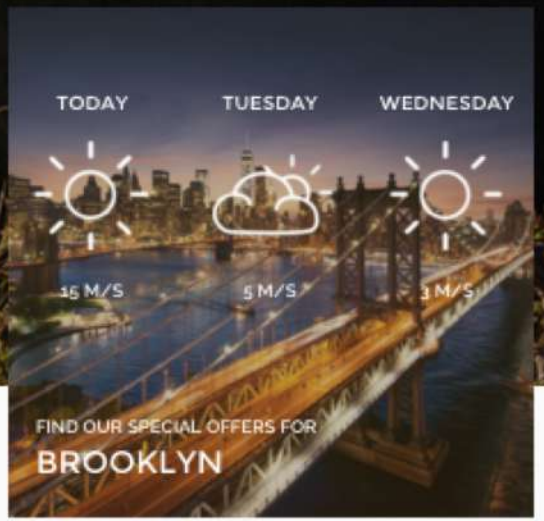


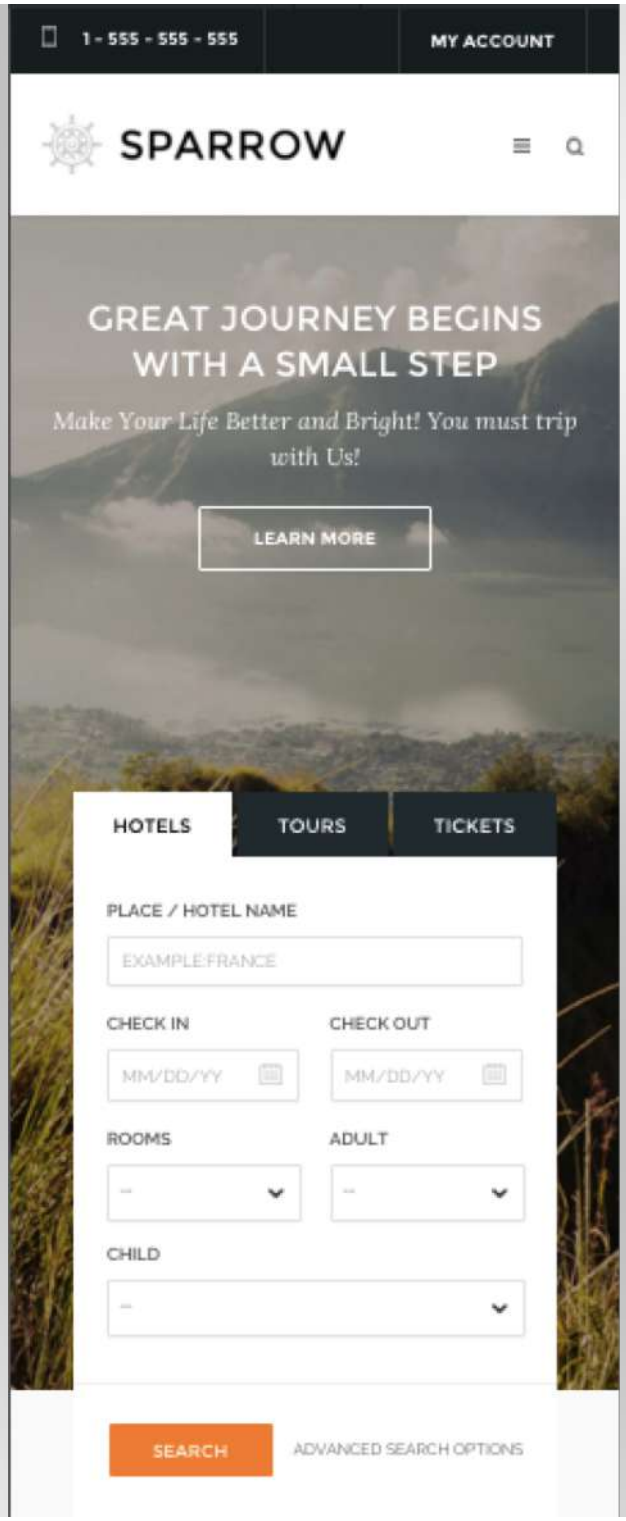
ANDRASSY THAI HOTEL

LOCATION:THAILAND

152$

AVG/NIGHT




1 - 555 - 555 - 555
MY ACCOUNT
SPARROW
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