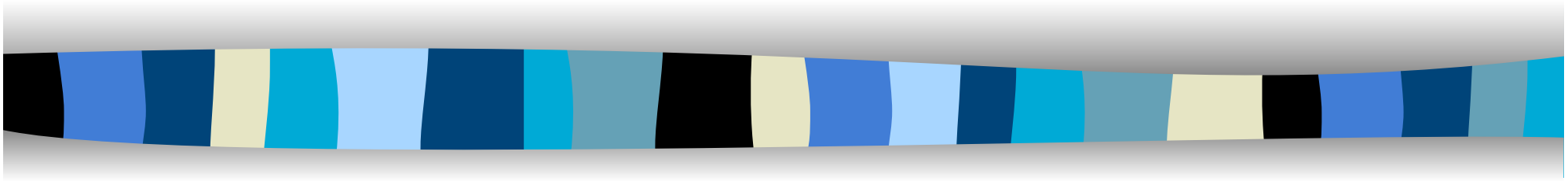
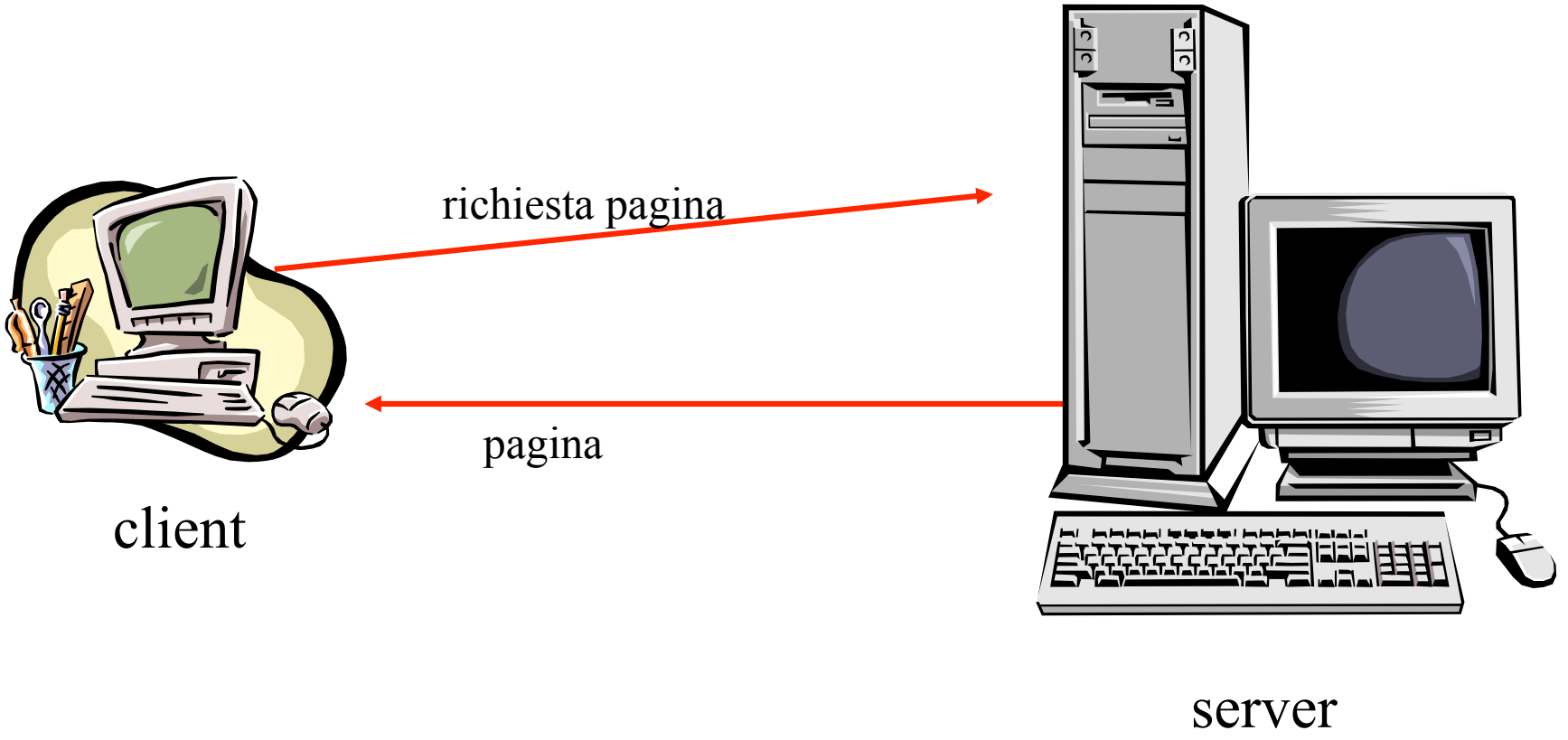
# Il WEB ed HTTP



# WEB: Architettura Client – Server



# *Le Pagine Web*

- Consistono di *istruzioni* (marcatori) HTML
- Memorizzate su computer detti *server web*
- Visualizzare da computer detti client usando un *browser*
  - Google Chrome
  - Firefox
  - Safari
  - Opera
  - Internet Explorer

# Il browser – 1

- Il browser legge un documento scritto in linguaggio HTML (+ CSS + Javascript) interpretandone i comandi e visualizzandolo
- Quando si clicca su un collegamento ipertestuale, il browser utilizza il protocollo HTTP per inviare ad un server Web una richiesta del nuovo documento (risorsa)

# Il browser – 2

- Il server Web risponde alla richiesta, utilizzando il protocollo HTTP ed invia il documento richiesto

- Il browser legge e interpreta l'informazione scritta in HTML (+CSS + Javascript) e la presenta (*quasi sempre*) nel formato corretto

# URL (*Uniform Resource Locator* )

- Per poter individuare ed accedere a risorse sul WEB è necessario un meccanismo per poterle identificare (tramite un nome) e localizzare (tramite un indirizzo)
- Una URL indica la collocazione reale di una risorsa accessibile mediante uno dei protocolli (HTTP) attualmente in uso su Internet

# HTTP

- "Protocollo di livello applicativo per sistemi di informazione distribuiti, collaborativi ed ipermediali"
  - Esso viene utilizzato dal web server e dal client (*user agent*) per comunicare
  - Si colloca al di sopra di TCP/IP
  - Permette di costruire sistemi di accesso all'informazione indipendenti dal tipo dell'informazione stessa

# Apertura della connessione

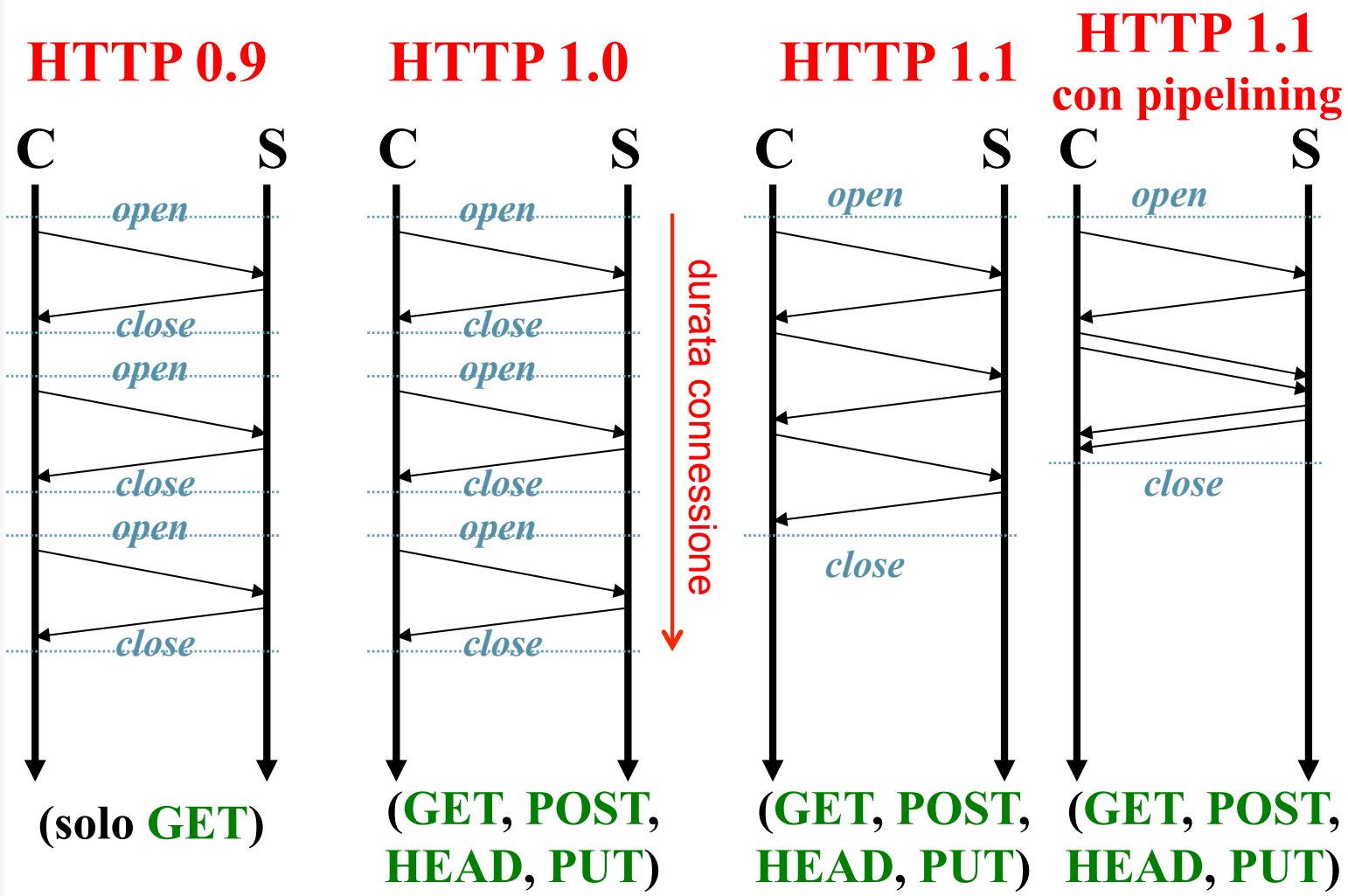
- Il browser (*user agent*) analizza l'URL e ne estrae il dominio
- Il browser consulta il DNS per ottenere l'indirizzo IP corrispondente al dominio
- Il browser apre una connessione TCP con il web server (di default usa la porta 80)
- I messaggi HTTP sono di 2 tipi:
  - **request / response**
- Una volta stabilita la connessione, il browser richiede una risorsa al web server (request) ed il server risponde inviando o la risorsa richiesta oppure un messaggio di errore (response)

# Caratteristiche di HTTP

- HTTP è esistito in tre versioni:
  - 0.9
  - 1.0 (RFC 1945)
  - 1.1 (RFC 2616)
- La differenza principale tra HTTP 1.0 e 1.1 è stata la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione

# Connessione HTTP

Esempio: richiesta di una pagina contenente due immagini



# Ulteriori caratteristiche di HTTP

- HTTP è “stateless” La percezione dell’utente è diversa….
  - Il server non mantiene alcuna informazione sulle richieste precedenti del client
  - I protocolli che conservano lo “stato” sono molto complessi
    - La storia passata (**stato**) deve essere mantenuta
  - Se il server (il client) interrompe la connessione, allora la *visione* dello stato di server e client possono essere inconsistenti e deve essere riconciliata

Insiemi di valori condivisi

# Messaggi HTTP

- Il protocollo definisce il formato dei messaggi
- request e response hanno lo stesso formato e seguono la struttura (RFC 822) di un messaggio di email (sono testuali non binari)

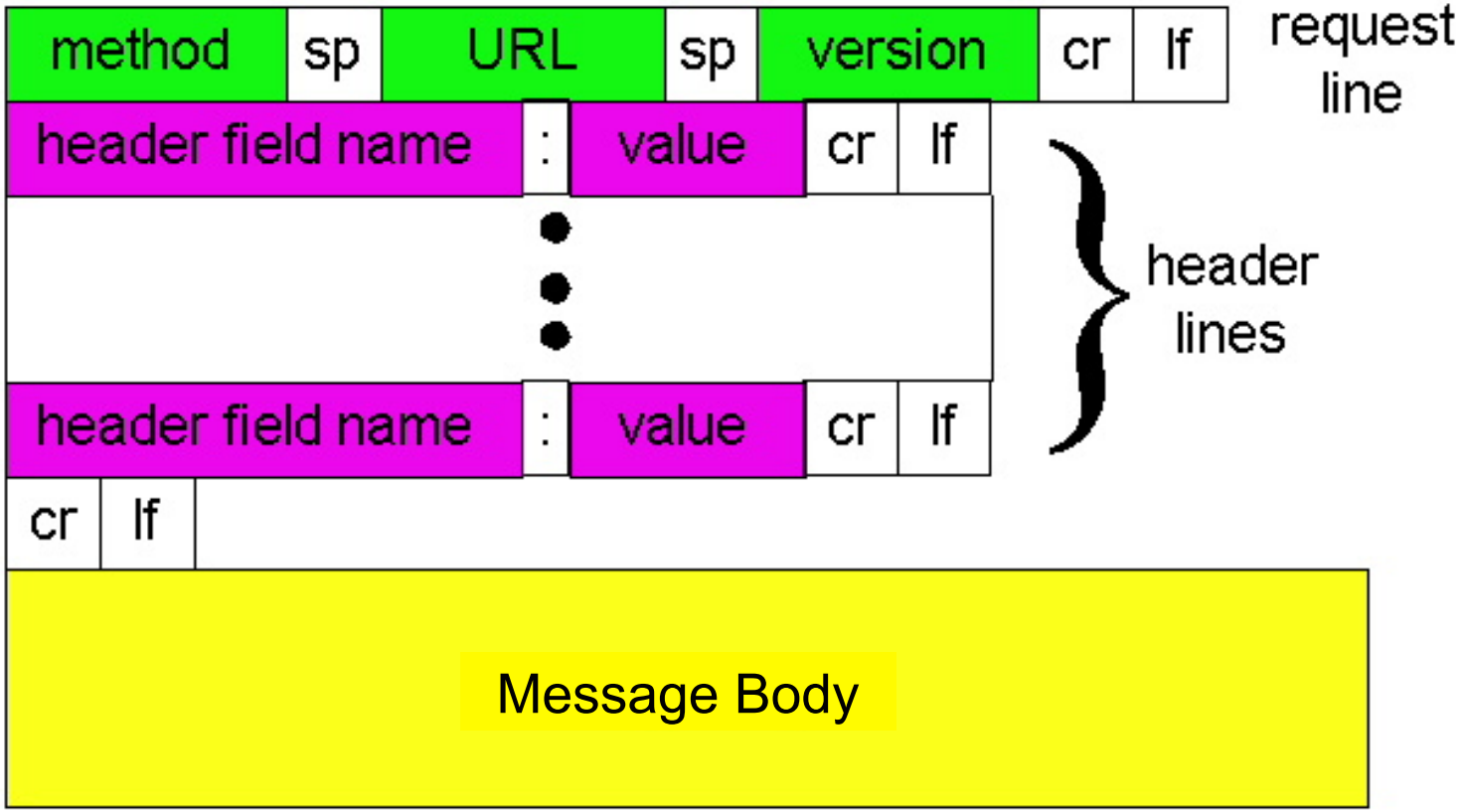
  start-line
  - può essere o una request-line o una status-line

  HEADER (contiene informazioni sull'host e sul corpo del messaggio – coppie nome/valore)
  - Content-length
  - Content-type

  <Linea Vuota>

  BODY *(può mancare)*

# Formato generale di *request*



# Esempio di request

request line (GET, POST, HEAD, PUT)

header lines

```
GET /TSW/esami.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:it
```

(ulteriore carriage return, line feed)

carriage return + line feed indicano la fine dell'intestazione, può iniziare il corpo (body) del messaggio – parte opzionale

# Metodi HTTP (*comandi*)

- Un metodo HTTP può essere
  - **Sicuro**: non genera cambiamenti allo stato di una risorsa
  - **Idempotente**: l'effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta
- Analizzeremo solo:
  - GET (Richiesta di dati/risorse)
  - HEAD (Richiesta al server di header HTTP contenenti informazioni)

  } lettura
  - POST (Invio di dati all'URL indicato)
  - PUT (Colloca i dati contenuti nel BODY all'URL indicato)

  } scrittura

# Il metodo GET

- È il metodo più utilizzato, si attiva quando
  - Si segue un link
  - Si scrive un URL nella barra indirizzi del browser
- GET è sicuro ed idempotente, e può essere:
  - *assoluto* (utilizzo di default, la risorsa viene richiesta senza specificare altro)
  - *condizionale* (se la risorsa corrisponde ad un criterio indicato negli header If-match, If-range If-modified-since, etc.)
  - *parziale* (se la risorsa richiesta è una sottoparte di una risorsa memorizzata sul server).

# Il metodo HEAD utilizzato dai proxy

- Simile al GET, ma non viene restituita una risorsa
- Si usa per ottenere informazioni sulla risorsa
- HEAD è sicuro ed idempotente, e viene usato per verificare:
  - *la validità di un URI*: la risorsa esiste e non è di lunghezza zero,
  - *l'accessibilità di un URI*: per accedere alla risorsa presso il server non sono richieste procedure di autenticazione
  - *la coerenza di cache di un URI*: la risorsa non è stata modificata nel frattempo, non ha cambiato lunghezza, valore hash o data di modifica

# GET ed HEAD

- GET e HEAD devono essere implementati dal server
- Gli altri metodi sono facoltativi

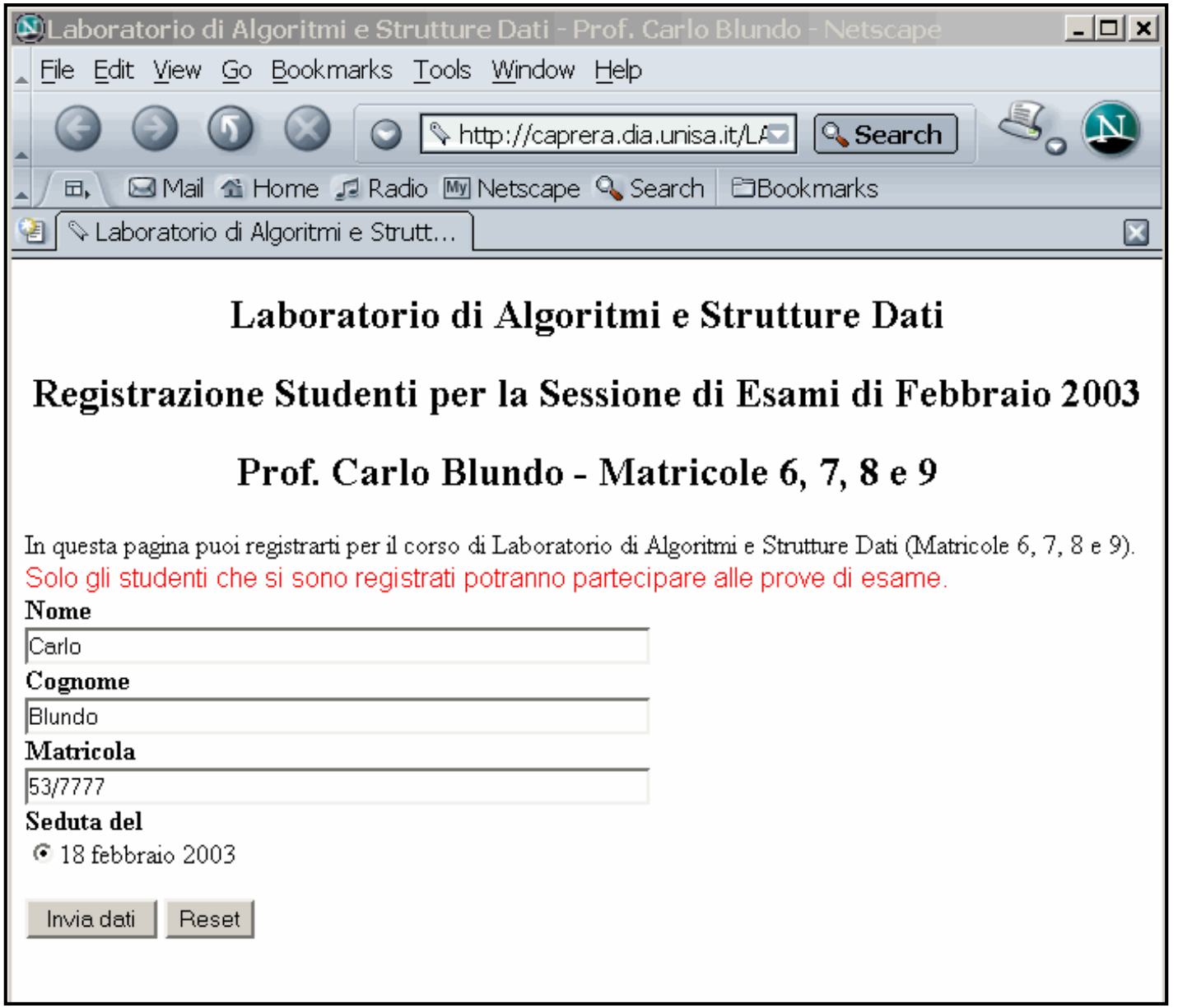
# Il metodo POST

- POST, pur essendo una *request*, contiene un *message body*
- POST serve per inviare una risorsa a un URL senza crearne una nuova
- POST non è sicuro né idempotente
- Ad esempio, si utilizza per:
  - Inviare un messaggio a un indirizzo di posta elettronica
  - Inviare il contenuto di un modulo ad un programma che risiede sul server

# GET e POST

- Anche GET può essere utilizzato per inviare dati al server
- La risorsa richista con GET è un programma che estrae i dati dal messaggio di request e li utilizza in qualche modo
- POST invia i dati nel corpo del messaggio di richiesta
- GET invia i dati come parte dell'URL

# Esempio


Laboratorio di Algoritmi e Strutture Dati - Prof. Carlo Blundo - Netscape
File Edit View Go Bookmarks Tools Window Help
http://caprera.dia.unisa.it/LA
Search
Mail Home Radio Netscape Search Bookmarks
Laboratorio di Algoritmi e Strutt...
Laboratorio di Algoritmi e Strutture Dati
Registrazione Studenti per la Sessione di Esami di Febbraio 2003
Prof. Carlo Blundo - Matricole 6, 7, 8 e 9
In questa pagina puoi registrarti per il corso di Laboratorio di Algoritmi e Strutture Dati (Matricole 6, 7, 8 e 9).
Solo gli studenti che si sono registrati potranno partecipare alle prove di esame.
Nome
Carlo
Cognome
Blundo
Matricola
53/7777
Seduta del
18 febbraio 2003
Invia dati
Reset

# Cosa riceve il server con POST

POST /cgi-bin/registra.php HTTP/1.1

Host: caprera.dia.unisa.it:12345
[. . .]
Connection: keep-alive

Referer: http://caprera.dia.unisa.it/LASD/prenota.html
Content-Type: multipart/form-data; boundary= ----------------------------41184676334
Content-Length: 438

----------------------------41184676334
Content-Disposition: form-data; name="nome"

carlo
----------------------------41184676334
Content-Disposition: form-data; name="cognome"

blundo
----------------------------41184676334
Content-Disposition: form-data; name="matricola"

53/7777
----------------------------41184676334

# Cosa riceve il server con GET

GET
/cgi-bin/registra.php?nome=carlo&cognome=blundo&
matricola=53%2F7777&seduta=2 HTTP/1.1

Host: caprera.dia.unisa.it:12345

[...]

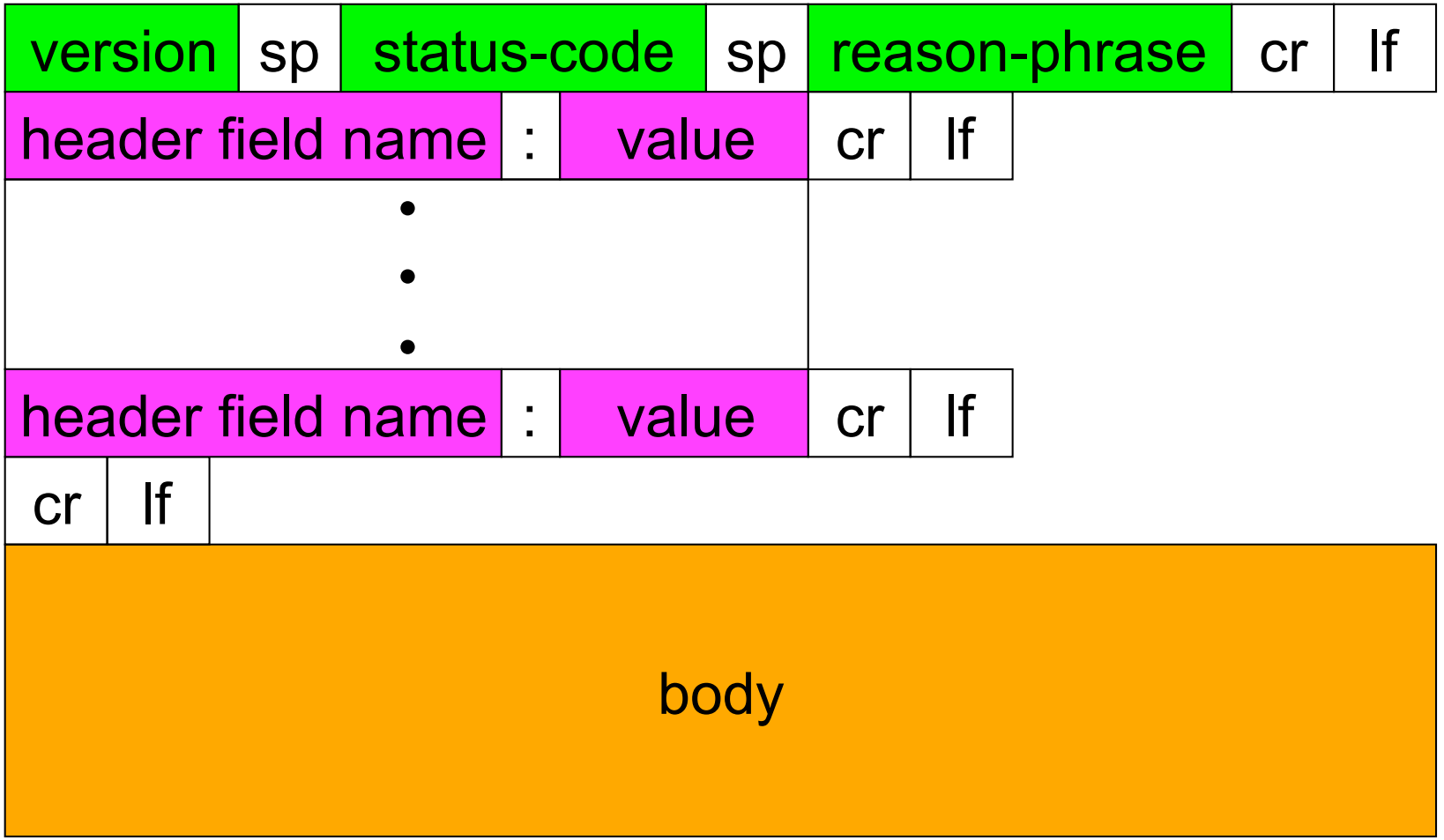
Keep-Alive: 300
Connection: keep-alive

Referer: http://caprera.dia.unisa.it/LASD/prenota.html

# Il metodo PUT – non implementato

- Il metodo PUT serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata.
- In generale, l'argomento del metodo PUT è la risorsa che ci si aspetta, in seguito, di ottenere eseguendo un GET con lo stesso argomento
- PUT è idempotente ma non sicuro, e comunque non offre nessuna garanzia di controllo degli accessi o locking.

# Formato generale di *response*



# Esempio di response

HTTP/1.1 200 OK
Date: Wed, 12 Feb 2003 12:43:01 GMT
Server: Apache/1.3.22 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 16 Oct 2002 09:47:40 GMT
ETag: "dbca-2-3dad35bc"
Accept-Ranges: bytes
Content-Length: 43
Connection: close
Content-Type: text/html

<HTML> <BODY>
Ciao mondo!
</BODY></HTML>

# Status Code – 1

- 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. Esistono le seguenti classi:
  - **1xx: Informational**
    - Una risposta temporanea alla richiesta, durante il suo svolgimento.
  - **2xx: Successful**
    - Il server ha ricevuto, capito e accettato la richiesta.

# Status Code – 2

- **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 (errore) interno (suo o di applicazioni CGI).

# Esempi di status code

**100** **Continue** (se il client non ha ancora mandato il body)

**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)

**Status code reason-phrase**

**Codice di stato motivazione**

# I cookie – 1

- HTTP è stateless
  - Il server non mantiene alcuna informazione sulle precedenti richieste del client
- Un cookie è una stringa scambiata tra il server ed il client
  - Il client mantiene con un cookie lo stato di precedenti connessioni, e lo invia al server di pertinenza ogni volta che richiede un documento.
  - Non è previsto in HTTP, ma è un'estensione di Netscape, proposta nell'RFC 2109
  - Per questioni di sicurezza/privacy si possono spedire cookies solo al server che li ha settati
    - **Vincolo facilmente aggirabile** same origin policy

# I cookie – 2

- I cookies usano due *extension-header*[1], uno per la risposta, ed uno per richieste successive:
  - **Set-Cookie**: header della risposta, il client può memorizzare il valore del cookie indicato e rispedirlo alla prossima richiesta.
  - **Cookie**: header della richiesta contenente il valore del cookie.
    - Il client decide se spedirlo sulla base del nome del documento, dell'indirizzo del server, e dell'età del cookie

[1] Non fanno parte degli header standard HTTP, ma sono stati aggiunti ad HTTP

# Interazione con cookie