



**Corso di Laurea in Ingegneria delle Telecomunicazioni**

**Corso di Reti di Calcolatori**

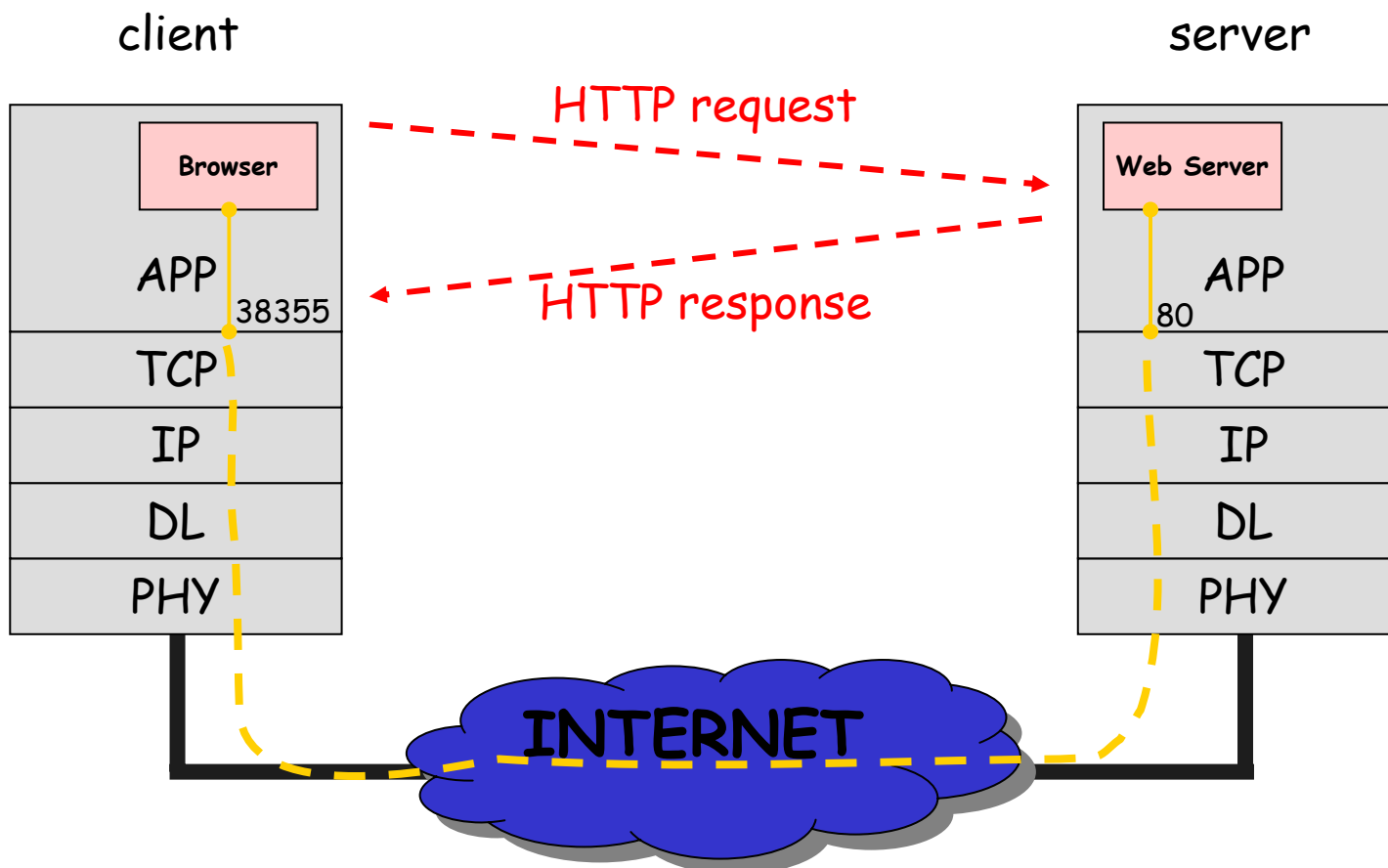
**Docente: Simon Pietro Romano**  
**[spromano@unina.it](mailto:spromano@unina.it)**

---

**Protocolli HTTP ed FTP**



# Web: interazione Client→Server

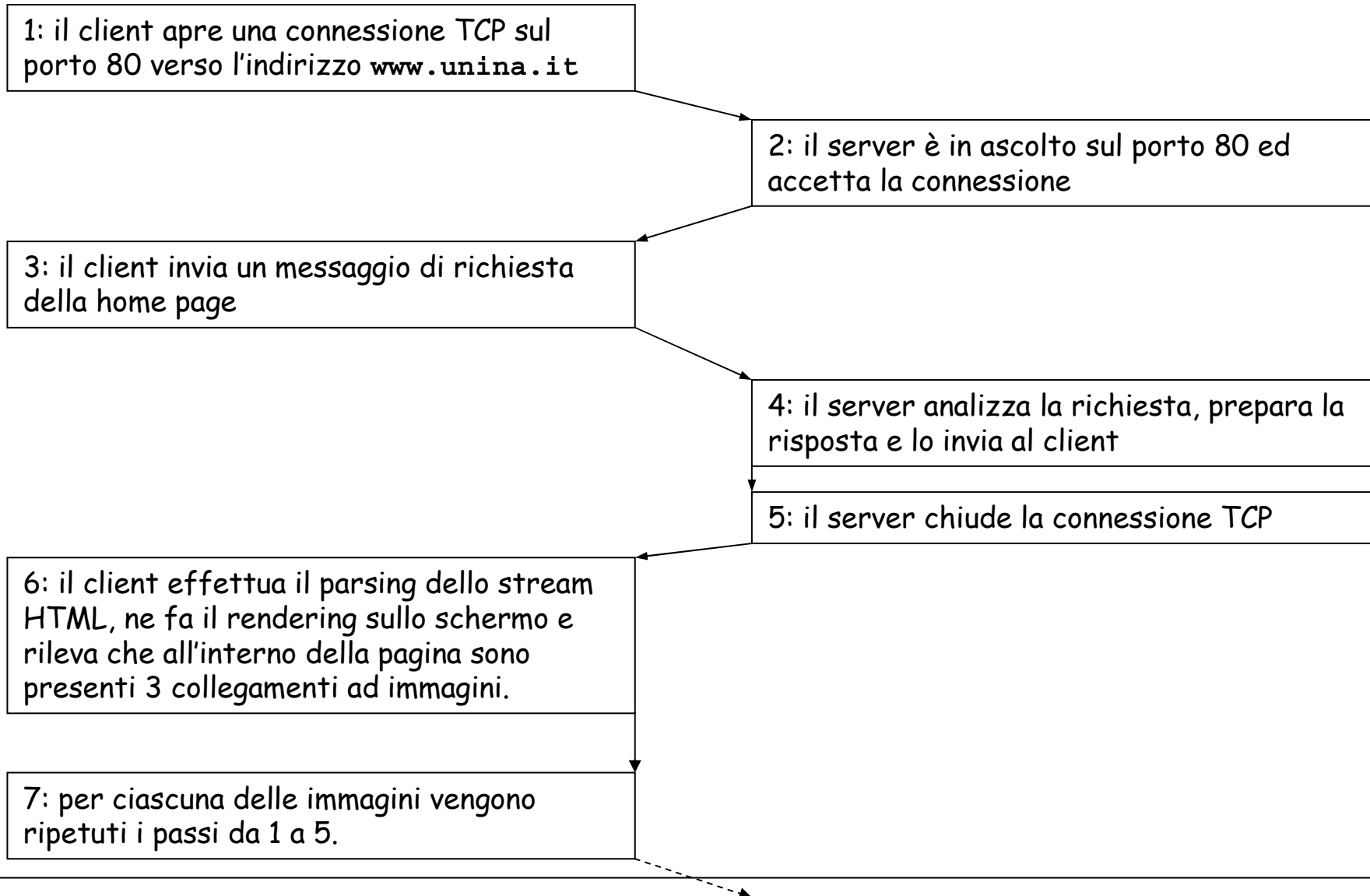




# Il protocollo HTTP

- Si basa su TCP
- Il client apre una socket verso il porto 80 (se non diversamente specificato) del server
- Il server accetta la connessione
- Il client manda una richiesta
- Il server risponde e chiude la connessione
  
- Il protocollo HTTP è stateless: né il server né il client mantengono a livello HTTP informazioni relative ai messaggi precedentemente scambiati

# Es: richiesta di una pagina contenente immagini



# Connessioni persistenti e non persistenti



## non persistente

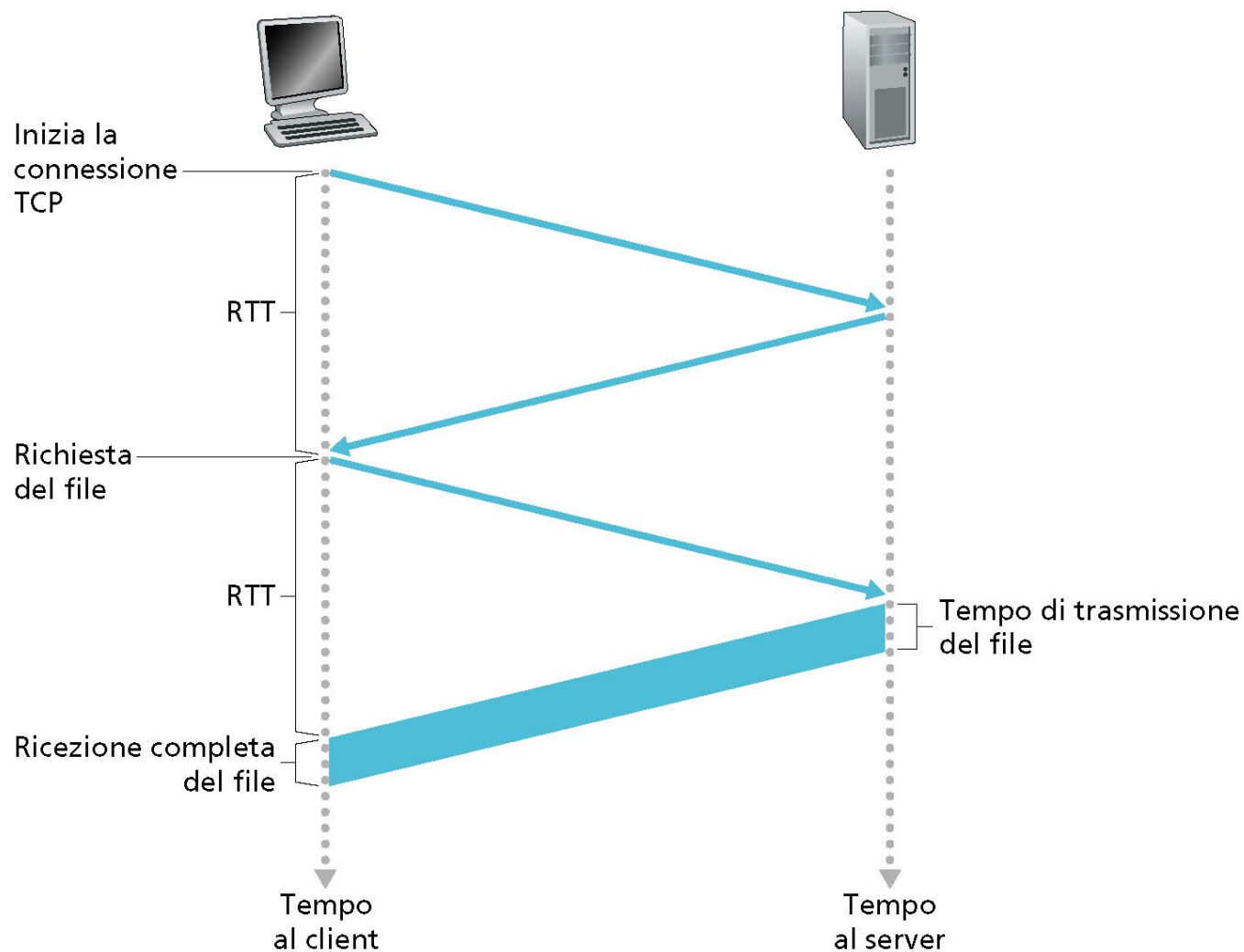
- HTTP/1.0
- Il server analizza una richiesta, la serve e chiude la connessione
- 2 Round Trip Time (RTT) per ciascuna richiesta
- Ogni richiesta subisce lo slow-start TCP

## persistente

- HTTP/1.1
- Sulla stessa connessione il server analizza tutte le richieste e le serve
- Il client riceve la pagina iniziale e invia subito tutte le altre richieste
- Si hanno meno RTTs ed un solo slow-start



# Round Trip Time e connessioni HTTP





# Il protocollo HTTP

- Per effettuare richieste, specificare cosa si richiede, rispondere alle richieste, si rende necessario un opportuno protocollo.
- Su Internet si usa il protocollo HTTP
- E' un protocollo testuale
- I messaggi sono costituiti da sequenze di byte
- Ogni byte identifica un carattere secondo la tabella ASCII
- In certi casi, il payload dei messaggi può essere comunque anche in formato binario.



# Il messaggio HTTP/1.0 request

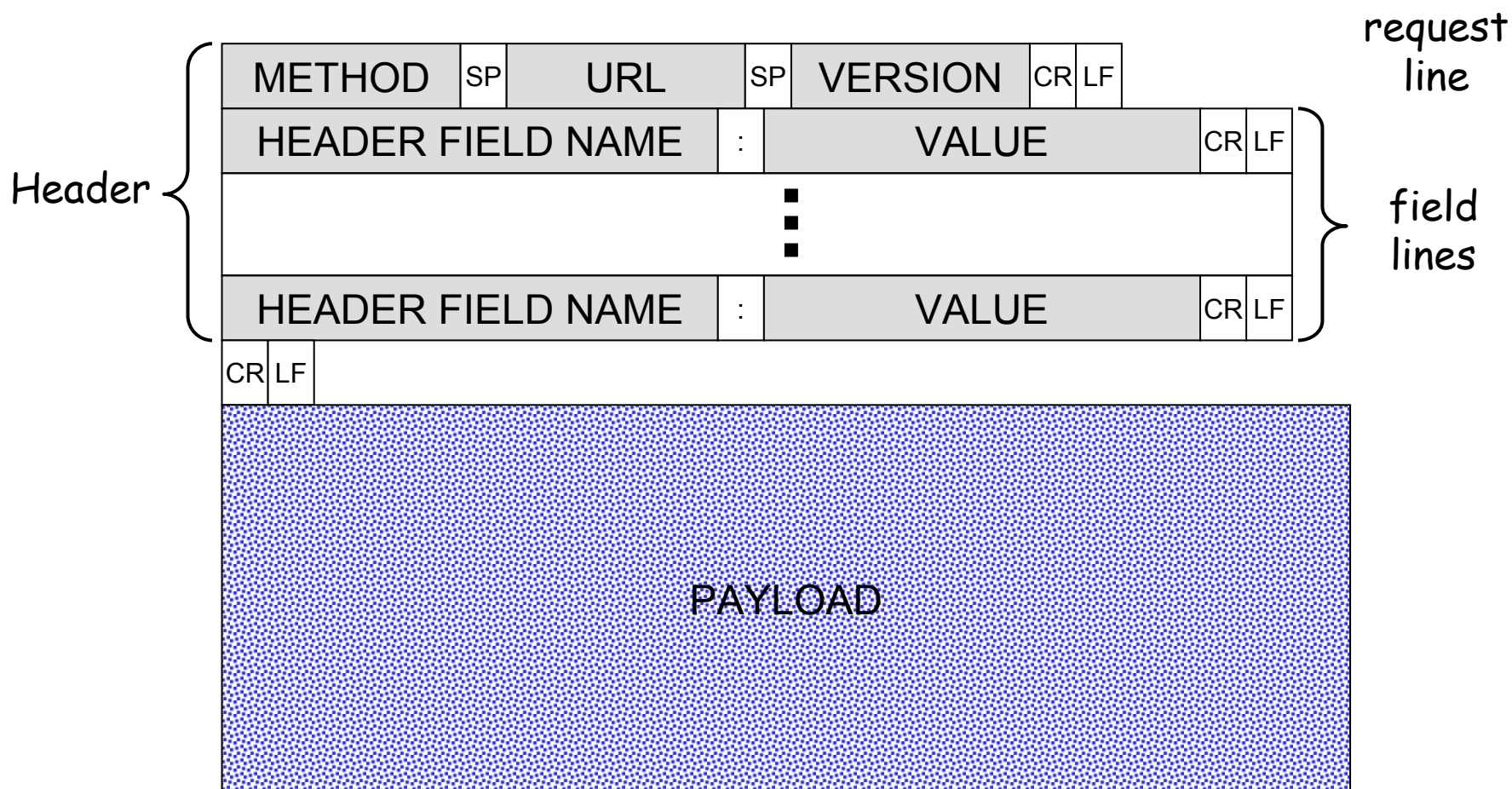
Un esempio di messaggio GET

GET /path/pagename.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: it
RIGA VUOTA

indica la fine del messaggio



# Il messaggio HTTP/1.0 request (cont)





# Il messaggio HTTP/1.0 response

codice di stato

HTTP/1.0 200 OK
Date: Mon, 16 Dec 2002 14:00:22 GMT
Server: Apache/1.3.24 (Win32)
Last-Modified: Fri, 13 Dec 2002 08:06:44 GMT
Content-Length: 222
Content-Type: text/html
RIGA VUOTA
PAYLOAD



# Esempi di codici di stato

## 200 OK

- **Successo:** l'oggetto richiesto si trova più avanti nel messaggio

## 301 Moved Permanently

- **L'oggetto richiesto è stato spostato.**  
Il nuovo indirizzo è specificato più avanti nel messaggio  
( Location: )

## 400 Bad Request

- **Richiesta incomprensibile al server**

## 404 Not Found

- **Il documento non è presente sul server**

## 505 HTTP Version Not Supported

- **La versione del protocollo HTTP usata non è supportata dal server**

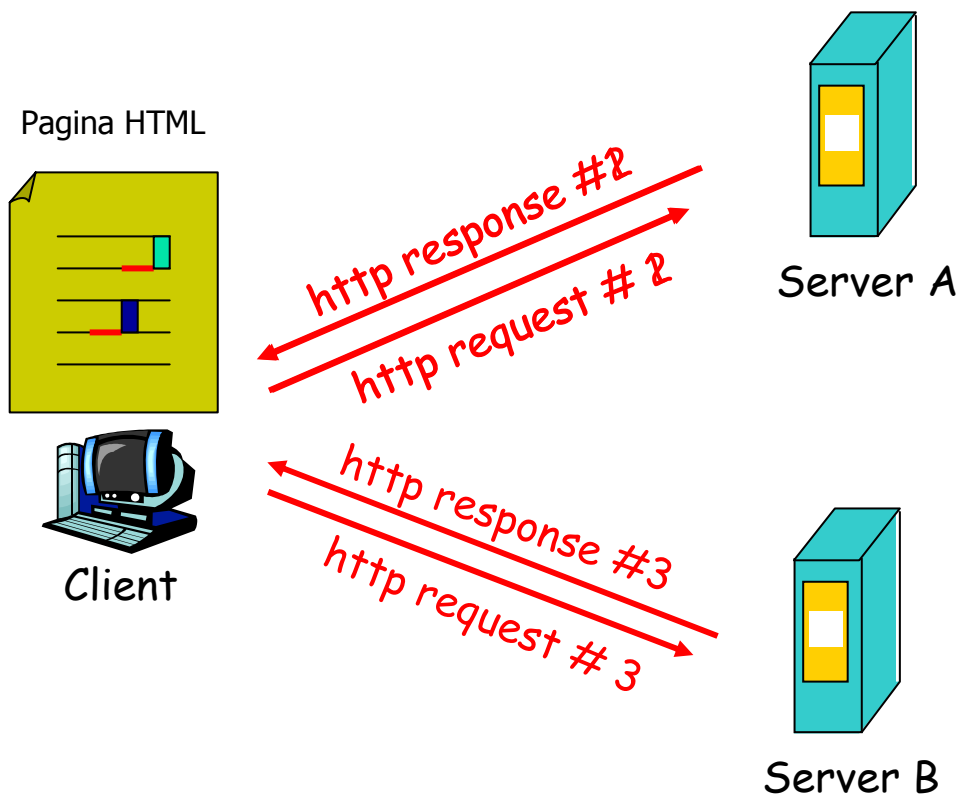


## HTTP per il trasferimento di pagine web

- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (*Hypertext Markup Language*)
- La pagina è identificata mediante un indirizzo, detto URL
- Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici
  - Es. sfondo, immagini, ecc.
- Ciascun oggetto è identificato dal proprio URL
- Questi oggetti possono trovarsi anche su server web diversi
- Una volta ricevuta la pagina HTML, il browser estrae i riferimenti agli altri oggetti che devono essere prelevati e li richiede attraverso una serie di connessioni HTTP



# HTTP per il trasferimento di pagine web (2)





# URL

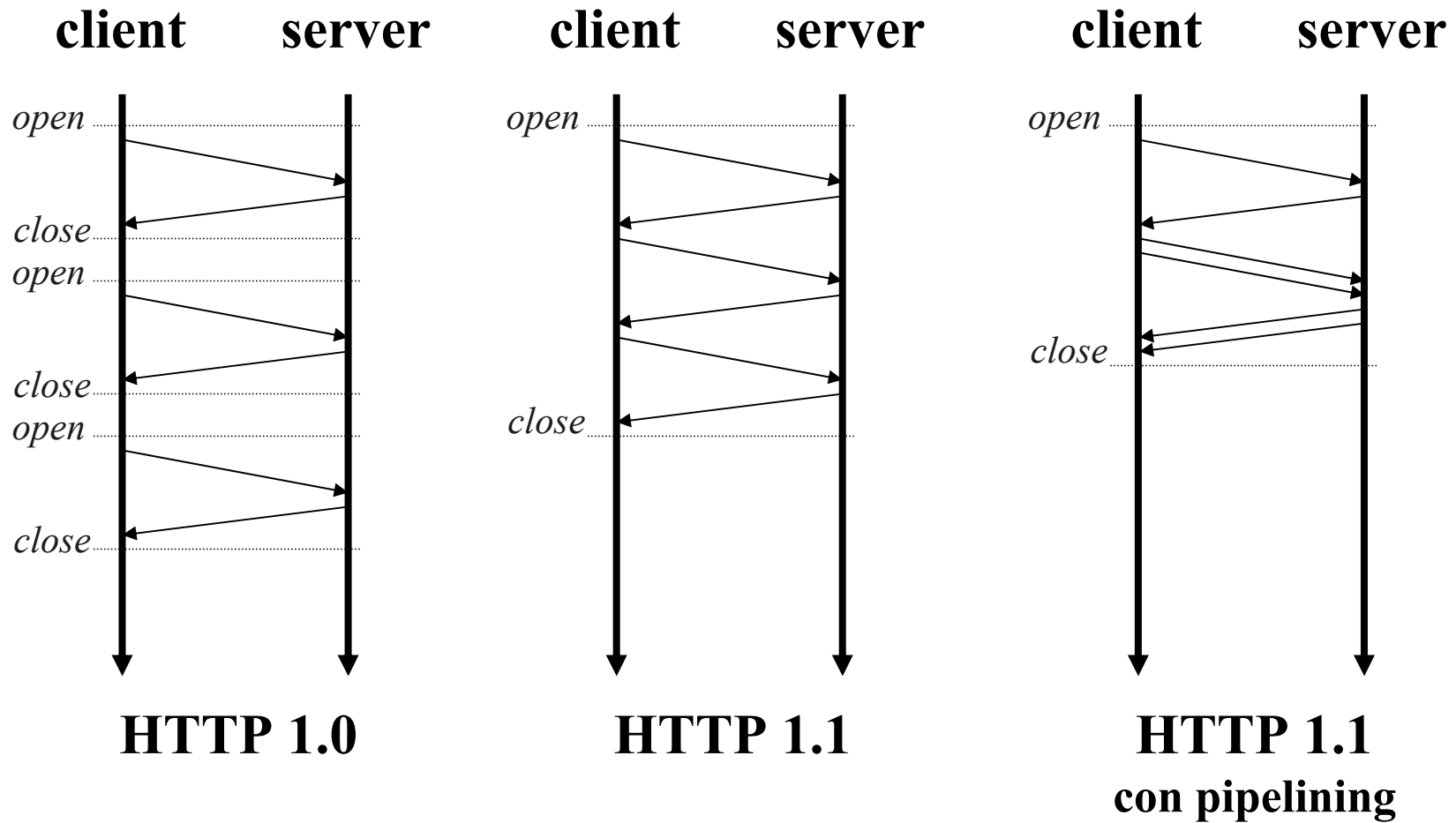
- Un URL HTTP ha la seguente sintassi:

`http://host[:port]/path[#fragment][?query]`

- Host identifica il server
  - Può essere sia un nome simbolico che un indirizzo IP in notazione dotted decimal
- Port è opzionale; di default è 80
- Path identifica la risorsa sul server
  - es: images/sfondo.gif
- #fragment identifica un punto preciso all'interno di un oggetto
  - es: #paragrafo1
- ?query è usato per passare informazioni dal client al server
  - es: dati inseriti nei campi di una form



# La connessione HTTP





# Il metodo GET

- Uno dei più importanti metodi di HTTP è GET
- Usato per richiedere una risorsa ad un server
- Questo è il metodo più frequente, ed 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
- GET può essere:
  - **assoluto**
    - la risorsa viene richiesta senza altre specificazioni
  - **condizionale**
    - si richiede la risorsa se è soddisfatto un criterio indicato negli header **If-match**, **If-modified-since**, **If-range**, ecc.
  - **parziale**
    - si richiede una sottoparte di una risorsa memorizzata



# Il metodo HEAD

- Simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza il corpo
- Usato per verificare:
  - **la validità di un URI**
    - la risorsa esiste e non è di lunghezza zero
  - **l'accessibilità di un URI**
    - la risorsa è accessibile presso il server, e non sono richieste procedure di autenticazione del documento
  - **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



# Il metodo POST

- Il metodo POST serve per trasmettere delle informazioni dal client al server, ma senza la creazione di una nuova risorsa
- POST viene usato per esempio per sottomettere i dati di una form HTML ad un'applicazione sul server
- I dati vengono trasmessi nel body della richiesta
- Il server può rispondere positivamente in tre modi:
  - 200 Ok: dati ricevuti e sottomessi alla risorsa specificata; è stata data risposta
  - 201 Created: dati ricevuti, la risorsa non esisteva ed è stata creata
  - 204 No content: dati ricevuti e sottomessi alla risorsa specificata; non è stata data risposta



# Il metodo PUT

- Il metodo PUT serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata
  - Esempio: upload di un file
- In generale, l'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso nome



# HTTP: Response

- La risposta HTTP è un messaggio testuale formato da una riga iniziale, da header facoltativi ed eventualmente un body (corpo)

**Version status-code reason-phrase CRLF**

**[Header]**

**CRLF**

**Body**

dove:

[...] indica un elemento opzionale

CRLF indica la sequenza di caratteri di codice ASCII

13 (base 16) = 19 (base 10) → CR = Carriage Return

10 (base 16) = 16 (base 10) → LF = Line Feed



# HTTP: Response (2)

- Esempio:

```
HTTP/1.1 200 OK
```

```
Date: Thu, 10 Apr 2003 11:46:53 GMT
```

```
Server: Apache/1.3.26 (Unix) PHP/4.0.3pl1
```

```
Last-Modified: Wed, 18 Dec 2002 12:55:37 GMT
```

```
Accept-Ranges: bytes
```

```
Content-Length: 7394
```

```
Content-Type: text/html
```

```
<HTML> ... </HTML>
```



# Status code

- 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
  - **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 interno



# Status code: esempi

- **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 programma in esecuzione sul server ha generato errore
- **501 Not implemented**
  - metodo non conosciuto dal server



# Gli header di risposta

- Gli header della risposta sono posti dal server per specificare informazioni sulla risposta e su se stesso al client
  - **Server:** una stringa che descrive il server: tipo, sistema operativo e versione
  - **Accept-ranges:** specifica che tipo di range può accettare (valori previsti: byte e none)



# Gli header generali

- Gli header generali si applicano solo al messaggio trasmesso e si applicano sia ad una richiesta che ad una risposta, ma non necessariamente alla risorsa trasmessa
- **Date**: data ed ora della trasmissione
- **MIME-Version**: la versione MIME usata per la trasmissione (sempre 1.0)
- **Transfer-Encoding**: il tipo di formato di codifica usato per la trasmissione
- **Cache-Control**: il tipo di meccanismo di caching richiesto o suggerito per la risorsa
- **Connection**: il tipo di connessione da usare
  - Connection: Keep-Alive → tenere attiva dopo la risposta
  - Connection: Close → chiudere dopo la risposta
- **Via**: usato da proxy e gateway



# Gli header dell'entità

- Gli header dell'entità danno informazioni sul body del messaggio, o, se non vi è body, sulla risorsa specificata
- **Content-Type**: il tipo MIME dell'entità acclusa
  - Specifica se è un testo, se un'immagine GIF, un'immagine JPG, un suono WAV, un filmato MPG, ecc...
  - Obbligatorio in ogni messaggio che abbia un body
- **Content-Length**: la lunghezza in byte del body
  - Obbligatorio, soprattutto se la connessione è persistente
- **Content-Base, Content-Encoding, Content-Language, Content-Location, Content-MD5, Content-Range**: l'URL di base, la codifica, il linguaggio, l'URL della risorsa specifica, il valore di digest MD5 e il range richiesto della risorsa
- **Expires**: una data dopo la quale la risorsa è considerata non più valida (e quindi va richiesta o cancellata dalla cache)
- **Last-Modified**: la data e l'ora dell'ultima modifica
  - Serve per decidere se la copia posseduta (es. in cache) è ancora valida o no
  - Obbligatorio se possibile

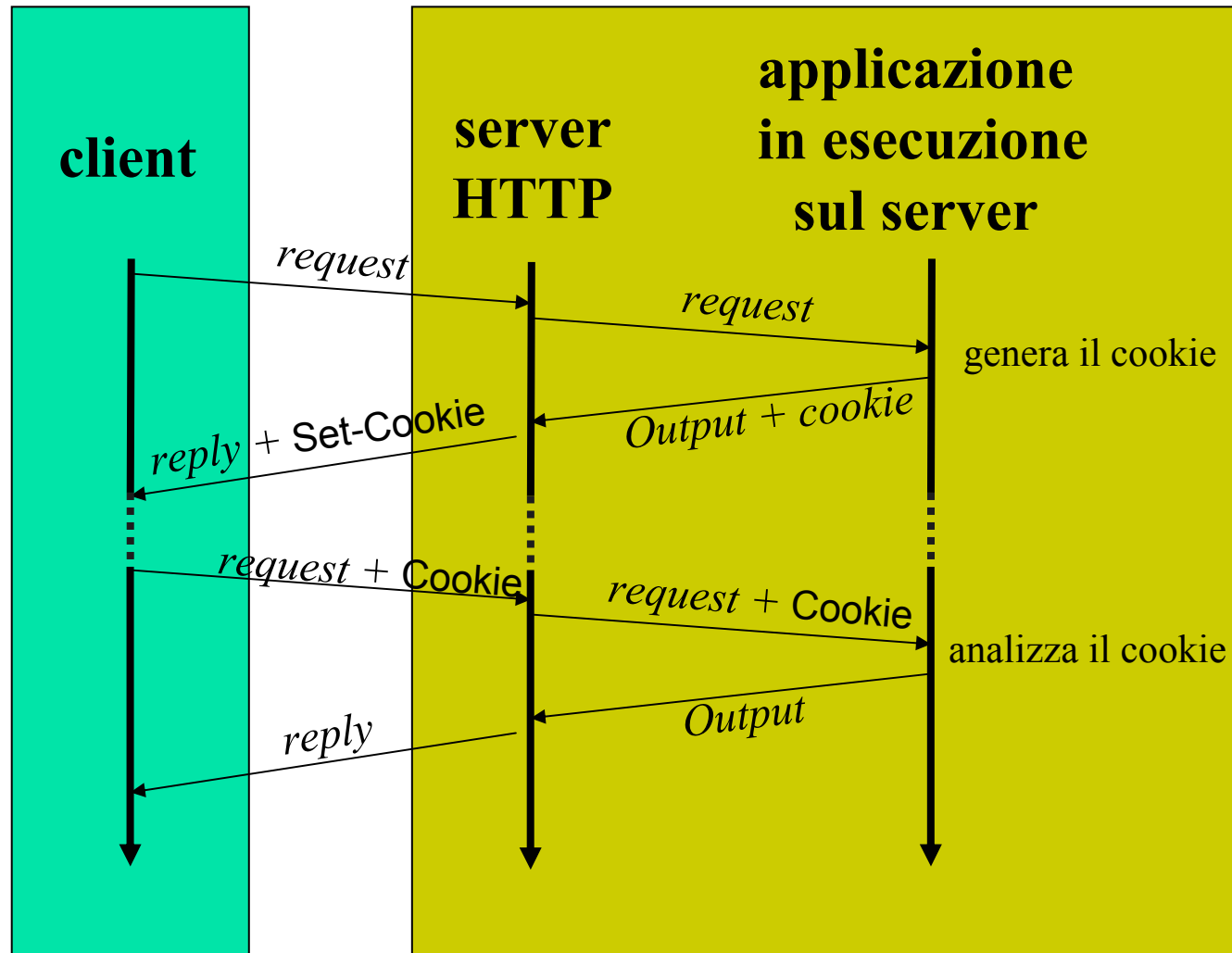


# I cookies

- **HTTP è stateless**: il server non è tenuto a mantenere informazioni su connessioni precedenti
- Un cookie è una breve informazione scambiata tra il server ed il client
- Tramite un cookie il client mantiene lo stato di precedenti connessioni, e lo manda al server di pertinenza ogni volta che richiede un documento
- Esempio: tramite un cookie si viene rediretti sulla pagina in Italiano tutte le volte che ci si ricollega allo stesso server (es. [www.google.com](http://www.google.com))
- I cookies sono definiti in RFC 2108 (su proposta di Netscape)



# Cookies (2)





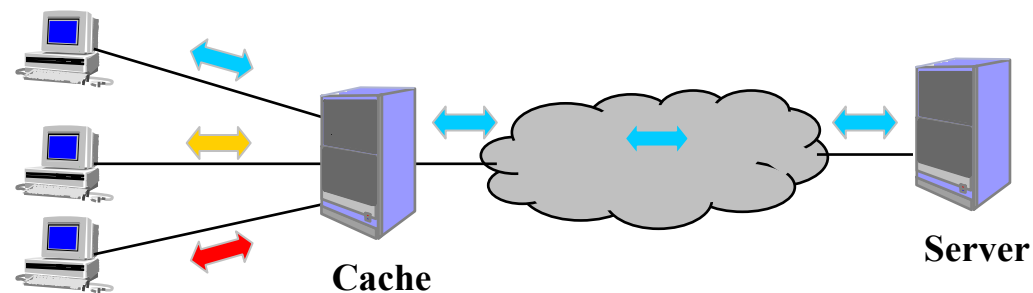
# Cookies: header specifici

- I cookies dunque usano due header: uno per la risposta, ed uno per le richieste successive:
  - **Set-Cookie**: header della risposta
    - il client può memorizzarlo (se vuole) e rispedito alla prossima richiesta
  - **Cookie**: header della richiesta
    - il client decide se spedito sulla base del nome del documento, dell'indirizzo IP del server, e dell'età del cookie
- Un browser può essere configurato per accettare o rifiutare i cookies
- Alcuni siti web richiedono necessariamente la capacità del browser di accettare i cookies



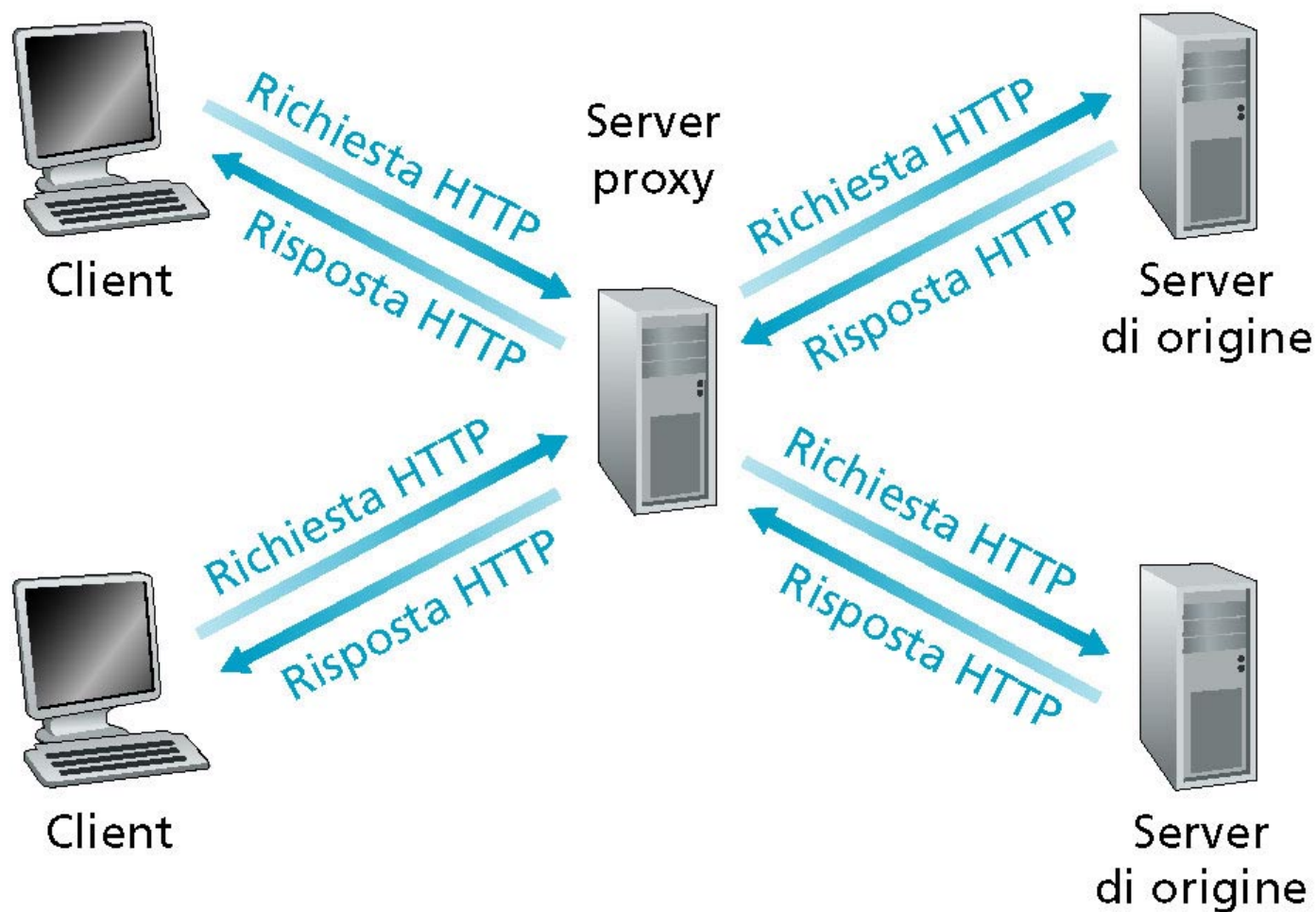
# Web caching

- Si parla genericamente di Web caching quando le richieste di un determinato client non raggiungono il Web Server, ma vengono intercettate da una cache
- Tipicamente, un certo numero di client di una stessa rete condivide una stessa cache web, posta nelle loro prossimità (es. nella stessa LAN)
- Se l'oggetto richiesto non è presente nella cache, questa lo richiede *in vece* del client conservandone una copia per eventuali richieste successive
- Richieste successive alla prima sono servite più rapidamente
- Due tipi di interazione HTTP: client-cache e cache-server



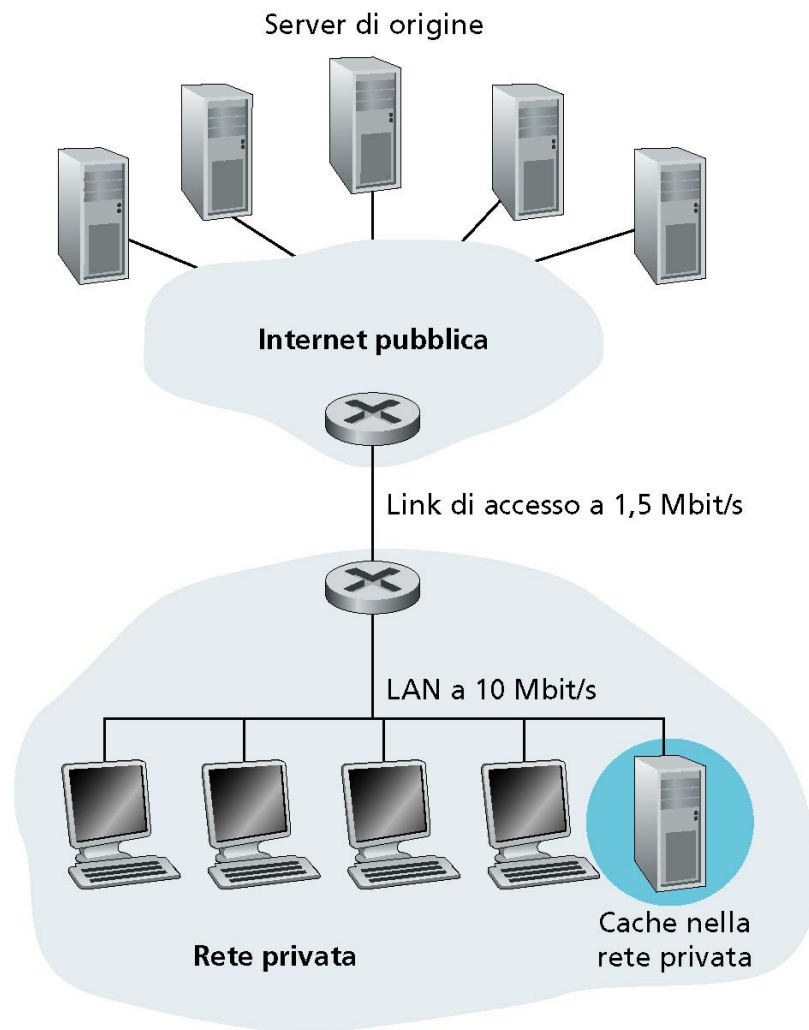


# Server proxy: schema logico





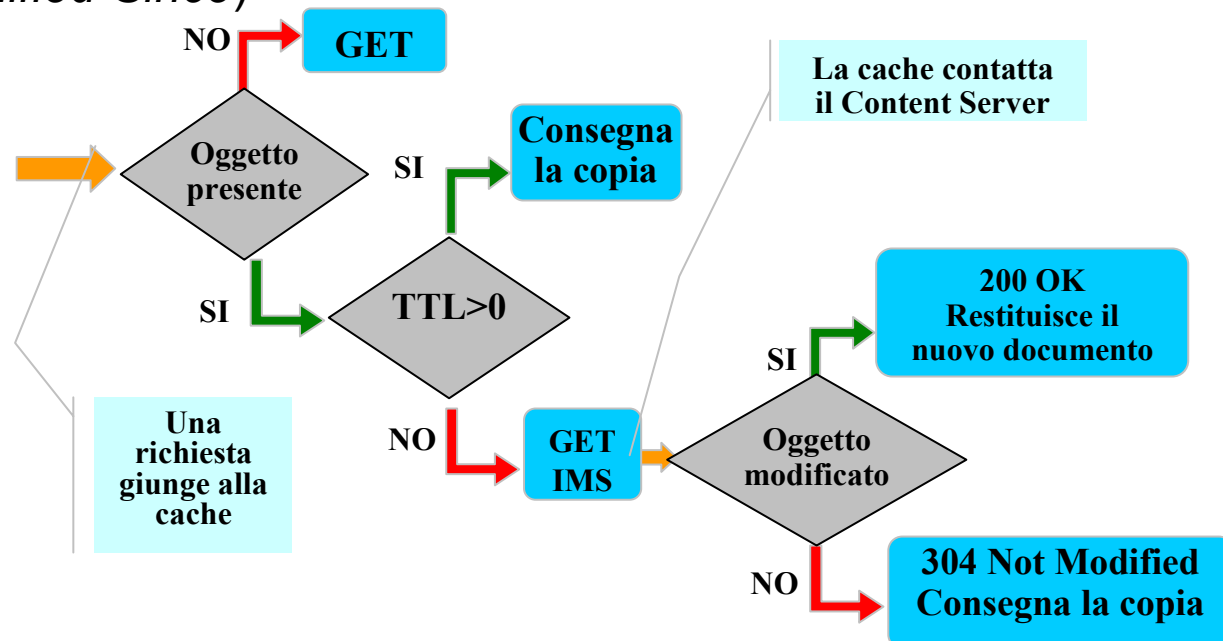
# Server proxy in una rete di accesso





# Gestione della coerenza

- Problema: cosa succede se l'oggetto presente nel server è aggiornato ?
- La copia in cache deve essere aggiornata per mantenersi uguale all'originale
- HTTP fornisce due meccanismi per la gestione della coerenza:
  - TTL (Time To Live) : il server quando fornisce un oggetto dice anche quando quell'oggetto "scade" (header *Expires*)
    - Quando TTL diventa  $< 0$ , non è detto in realtà che l'oggetto sia stato realmente modificato
  - Il client può fare un ulteriore controllo mediante una GET condizionale (*If-Modified-Since*)



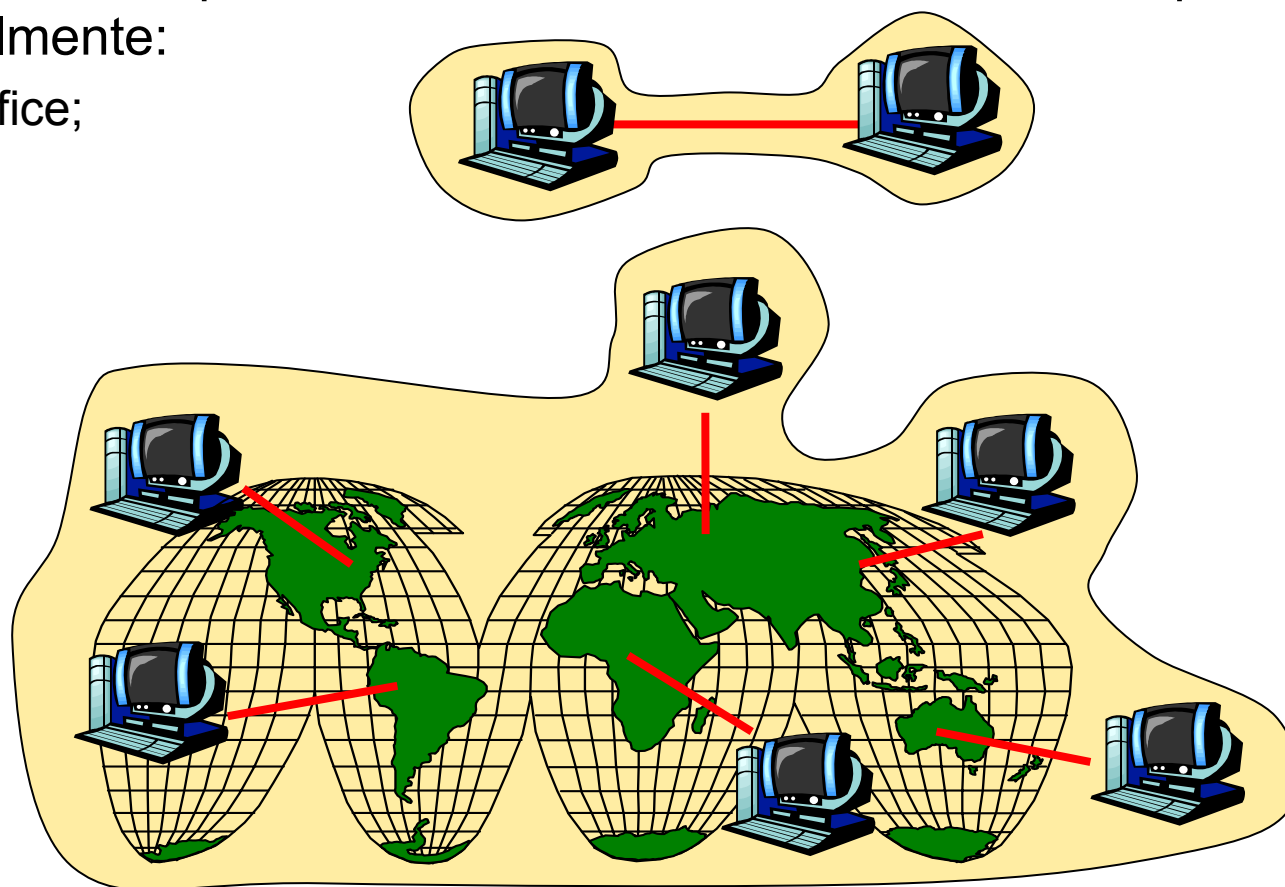


## File Transfer Protocol (FTP)

- Internet oggi si presenta come una rete ad estensione globale che connette molti milioni di macchine sparse su tutto il globo.
- Spesso sorge l'esigenza di copiare un file da una macchina ad un'altra per poterlo utilizzare localmente:

- un documento di Office;
- un file eseguibile;
- un file di testo;
- etc...

- Ciò può accadere sia tra macchine molto distanti tra di loro che tra macchine direttamente connesse, presenti nello stesso locale.



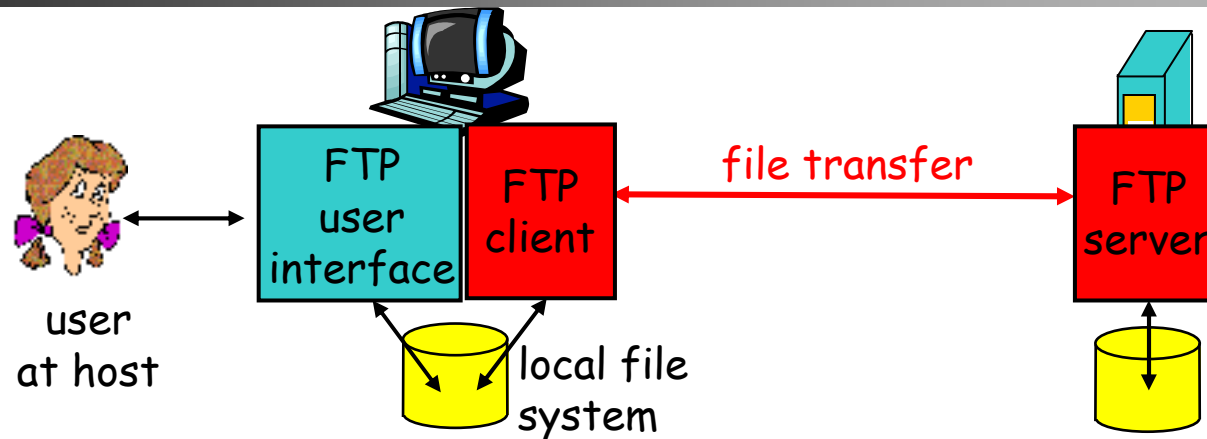


## Il protocollo FTP

- Un apposito protocollo è stato definito a questo scopo.
- Si chiama File Transfer Protocol (FTP)
- Attraverso di esso è possibile trasferire uno o più files di qualsiasi tipo tra due macchine.
- Il protocollo FTP è descritto in RFC959
  
- NOTA: un RFC (Request For Comment) è un documento pubblico sottoposto alla comunità Internet al fine di essere valutato. Ciò che è un RFC rappresenta uno standard *de facto* nella comunità Internet. Tutti gli RFC possono essere reperiti al sito dell'Internet Engineering Task Force (<http://www.ietf.org>)



# Come funziona FTP

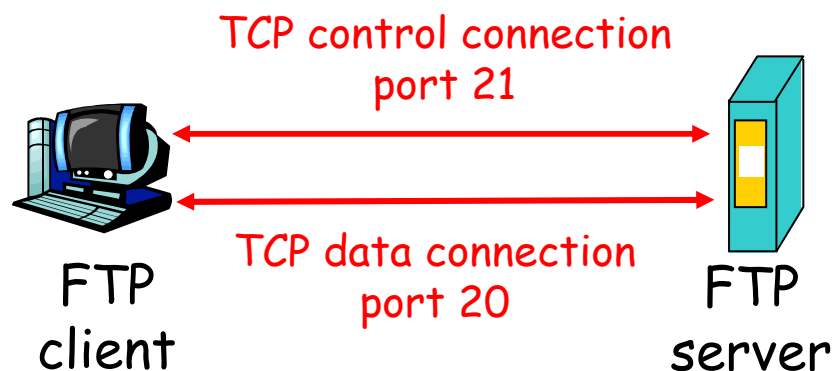


- trasferisce files da o verso una macchina remota
- usa il modello client/server
  - **client**: è l'entità che dà luogo al trasferimento (sia in un senso che nell'altro)
  - **server**: è l'entità remota che è in continua attesa di connessioni FTP da parte di altre entità
- ftp server: numero di porto 21



# Le connessioni di una sessione FTP

- Il client ftp contatta il server ftp al porto 21;
- vengono aperte due connessioni parallele:
  - **controllo**: scambio di comandi, messaggi di risposta tra il client e il server  
**controllo "out of band" (fuori banda)**
  - **dati**: file che fluiscono dal client al server o viceversa
- un server ftp mantiene uno stato:
  - la directory corrente;
  - i dati dell'autenticazione.





# Scambio delle informazioni

- I comandi vengono inviati come testo ASCII sulla connessione di controllo
- Anche le risposte sono costituite da testo ASCII
- *NOTA: il testo ASCII è una sequenza di caratteri testuali stampabili*



# Esempi di comandi e codici

## Esempi di comandi:

- **USER** *username*
- **PASS** *password*
- **LIST**  
restituisce la lista dei files  
presenti nella directory  
corrente
- **GET filename**  
preleva il file dalla macchina  
remota
- **PUT filename**  
invia il file alla macchina  
remota

## Esempi di codici di stato:

- 331 Username OK,  
password required
- 125 data connection  
already open;  
transfer starting
- 425 Can't open data  
connection
- 452 Error writing  
file



# Cosa è un server FTP

- Non è possibile per un client stabilire una connessione FTP verso una qualsiasi macchina
- Il tipo di paradigma adottato (client/server) presuppone infatti che il server debba essere stato opportunamente configurato per accettare connessioni
- Normalmente non tutte le macchine sono originariamente configurate per accettare connessioni di tipo FTP; se si tenta di stabilire una connessione verso una macchina non abilitata la sessione fallisce e nessun trasferimento risulta possibile
- Ad esempio le macchine dotate di Win98, WinME, WinXP e Win2000Professional non accettano automaticamente connessioni FTP ma devono essere opportunamente configurate perché ciò possa avvenire
- Le macchine Linux o Win2000Server, invece, spesso hanno il servizio FTP già attivo, senza che esso debba essere esplicitamente installato
- I client FTP sono invece disponibili pressoché su tutti i sistemi operativi