



Università  
degli Studi  
della Campania  
*Luigi Vanvitelli*

Reti di Calcolatori e Cybersecurity

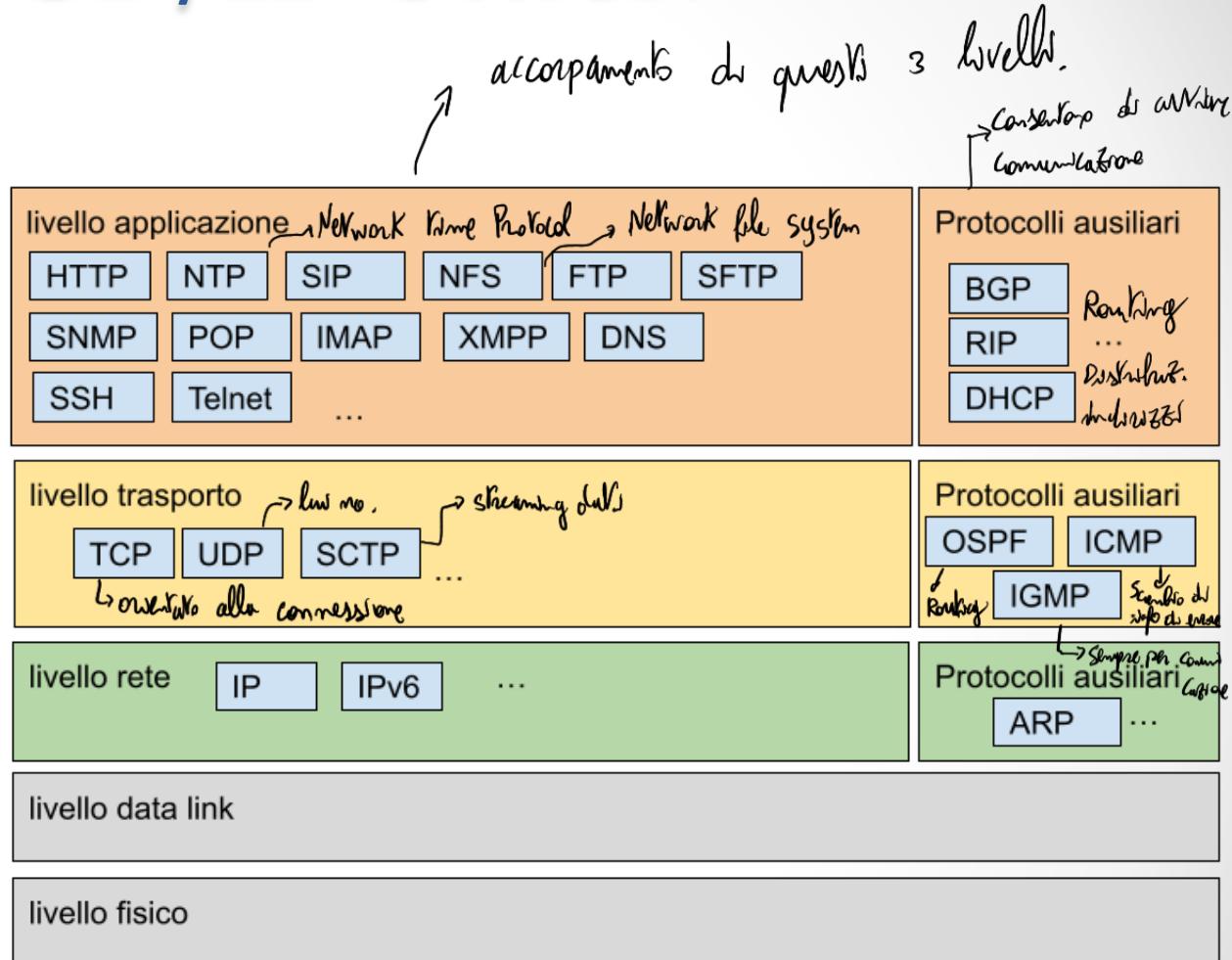
# Internet Protocol Suite

## HTTP

Ing. Vincenzo Abate

# TCP/IP stack

- L'Internet Protocol Suite (Conosciuto anche come stack di protocolli TCP/IP) è il termine usato per riferirsi all'intero insieme di protocolli oggi utilizzati in Internet
- Internet Protocol Suite non considera i livelli al di sotto del livello di rete
- Questo perché il protocollo IP può essere adattato a qualsiasi tecnologia di livello 2



nella def. dello standard è facile adattarlo a qualsiasi livello data link.

# Nomi PDU TCP/IP stack

- A livello generale parliamo di PDU indicando un pacchetto generico formato da header e payload
- Generalmente a seconda del livello la PDU assume nomi specifici

Layer	PDU name
Application	Message
Transport	Segment
Network	Datagram
Data Link	Frame
Physical	Bit

In base al livello, il pacchetto di quel livello assume nome diverso

# HTTP

- È il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) da server a client
- HTTP è l'acronimo di HyperText Transfer Protocol
- Gestisce sia le richieste (URL) inviate al server che le risposte inviate al client (pagine)
- Si basa su TCP
  - 1. Il client apre un socket verso il porto TCP 80 del server (se non diversamente specificato)
  - 2. Il server accetta la connessione
  - 3. Il client manda una richiesta per uno specifico oggetto identificato mediante una URL *Richiesta per ogni oggetto*
  - 4. 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
- Ci sono state diverse versioni particolarmente significative di HTTP: v1.0, v1.1 e v1.1 con pipelining, HTTP/2

# 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 *Cioè che non passa.*
- **Request:** messaggio HTTP di richiesta
- **Response:** messaggio HTTP di risposta
- **Resource:** oggetto di tipo dato univocamente definito
- **URI:** Uniform Resource Identifier – identificatore unico per una risorsa *Identifier ogni risorsa: qualcosa cosa ha che ha. Risorsa: html page*
- **Entity:** rappresentazione di una risorsa, può essere incapsulata in un messaggio, tipicam. di risposta *Quello che ha può valere all'interno di una risposta*

# Uniform Resource Locator (URL)

Dove posso trovare protocollo.

- Un URL HTTP ha la seguente sintassi (RFC 2396):  
`http://host[:port]/path[#fragment][?query]`
  - specifica parte di quella pagina
  - Porta non manca facoltativa
- Host identifica il server, può essere sia un nome simbolico che un indirizzo IP in notazione dotted decimal
  - ↳ quale risorsa voglio su quella macchina
- Port è opzionale; di default è 80 → identifica un'applicazione in esecut. a livello trasporto.
- 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

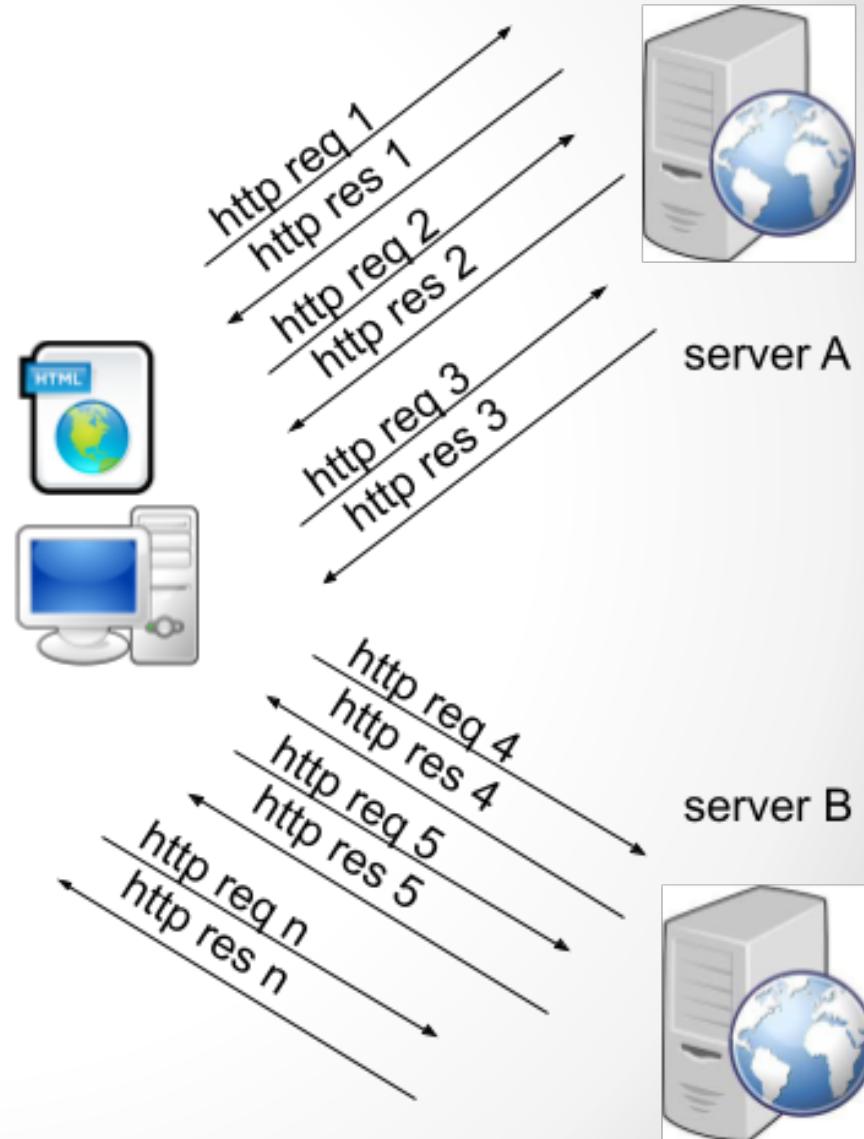
# HTTP e pagine web

- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (Hypertext Markup Language)
- La pagina è identificata mediante un indirizzo, l' 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

→ Ogni richiesta è una connessione da stabilire

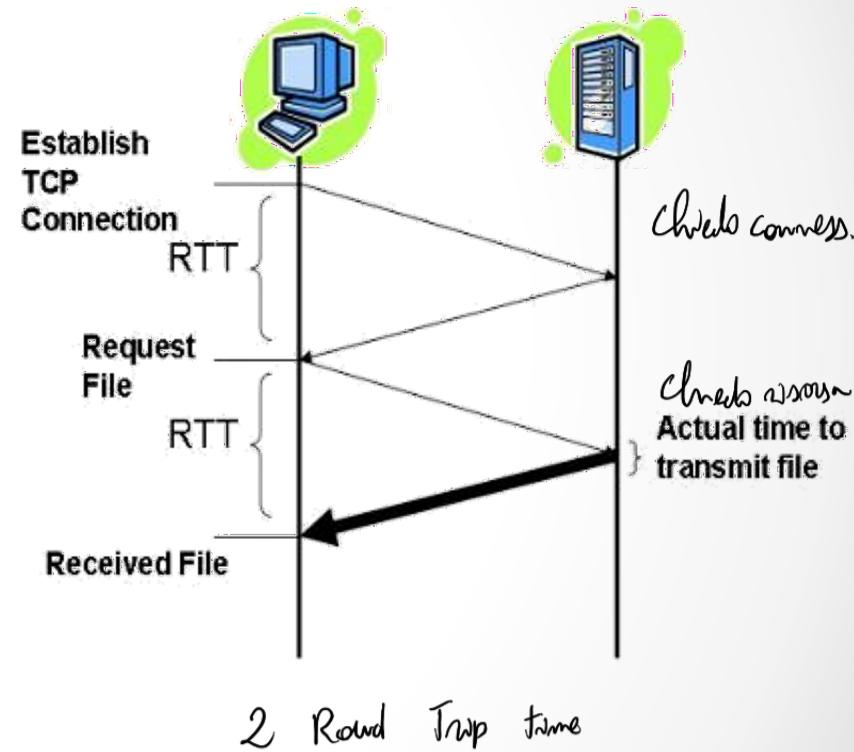
# HTTP v1.0 e pagine web (esempio)

1. Il client http inizia una connessione TCP verso il server http sull'host sulla porta 80
2. Il server http è "in ascolto" sulla porta 80. "Accetta" la richiesta di connessione e ne dà conferma al client
3. Il client http invia un messaggio di richiesta http (request message) contenente URL
4. Il server http riceve il messaggio di richiesta, costruisce un messaggio di risposta (response message) contenente l'oggetto richiesto e lo invia. Ciascun oggetto è identificato dal proprio URL
5. Il client http riceve il messaggio di risposta contenente il file html, visualizza la pagina html. Analizzando il file html, browser trova i riferimenti a n oggetti jpeg
6. Il server http chiude la connessione TCP
7. I passi 1-6 sono ripetuti per ciascuno dei 10 oggetti jpeg



# HTTP v1.0 e RTT

- HTTP 1.0 è «non persistente»
- Per ogni connessione devono essere allocati buffer TCP e le variabili del TCP devono essere conservate nel client e nel server. Questo può sovraccaricare il server Web.
- La richiesta di ogni oggetto richiede 2 Round Trip Time (RTT)
- Ogni richiesta subisce lo slow-start TCP

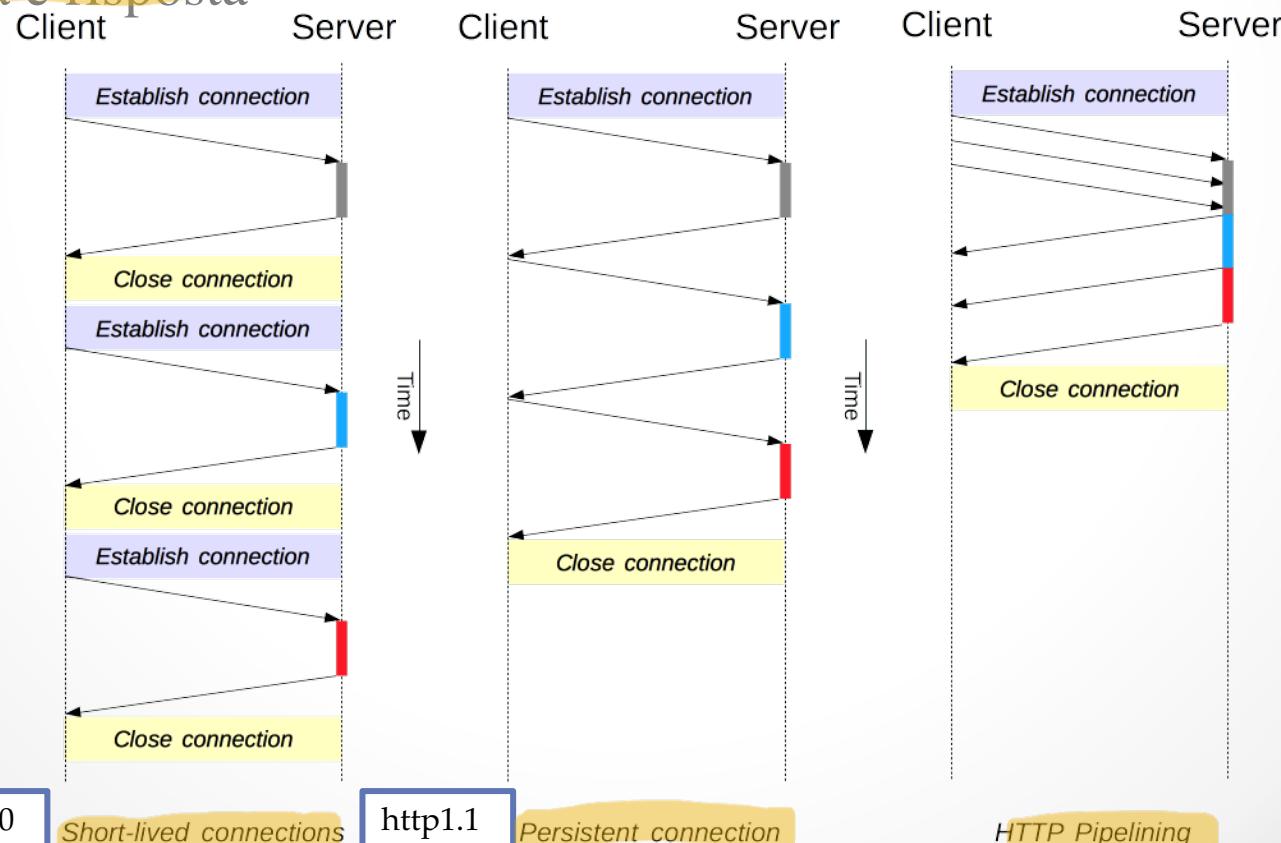


# HTTP v1.0 e v1.1

- HTTP 1.1 è «persistente»
- 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
- Il server lascia aperta la connessione TCP dopo aver spedito la risposta e può quindi ricevere le richieste successive sulla stessa connessione
- Si hanno meno RTT
- Nell'esempio precedente l'intera pagina Web (file HTML + 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)
  - 1 connessione, ma non posso fare un'altra richiesta prima che trovo la soluzione

# 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



# HTTP/2

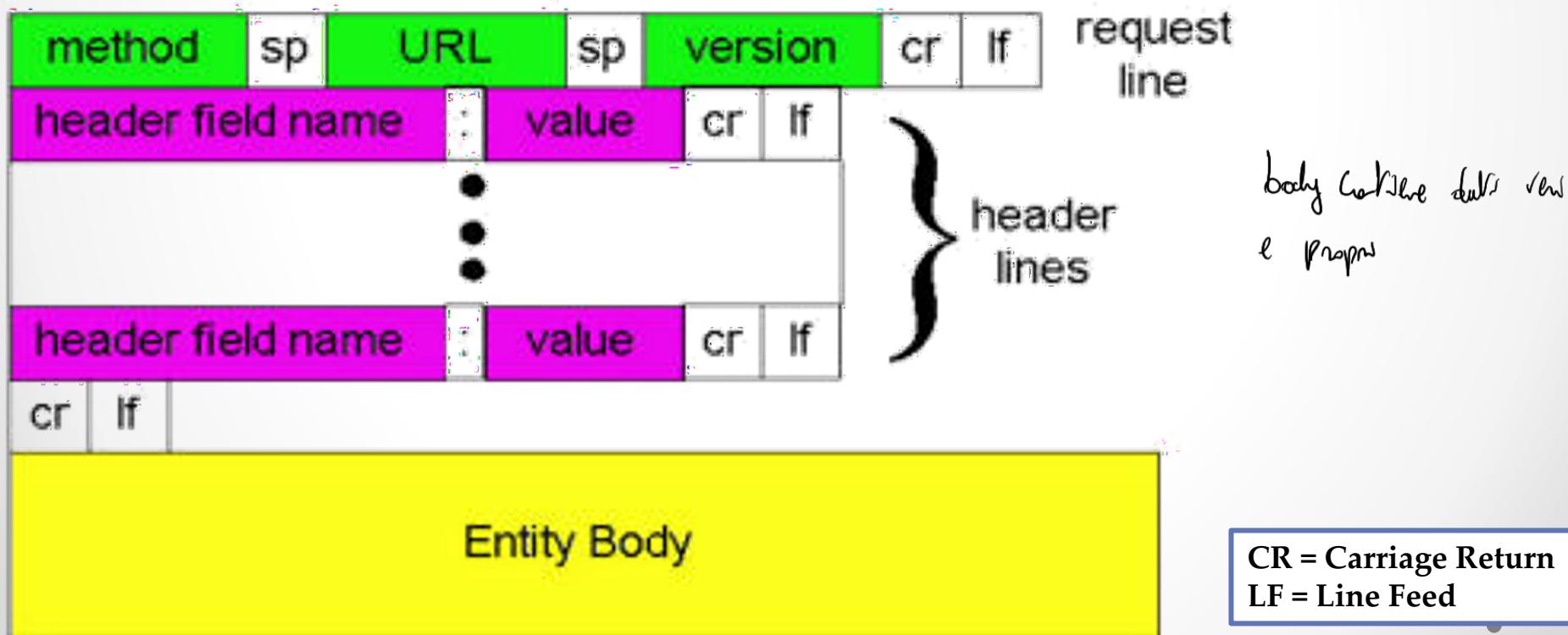
Dopo ampia discussione e lunga battaglia, processo di standardizzazione sta giungendo a compimento HTTP/2 sviluppato dal Working Group Hypertext Transfer Protocol (httpbis) di IETF, pubblicato come RFC 7540 a Maggio 2015

Obiettivo fondamentale di HTTP/2: Miglioramento performance complessiva con full backward compatibility con HTTP 1.1

- request-response multiplexing (per ricevere più oggetti in un'unica sessione)  
*Offrire più risorse da quelle richieste*
- header compression
- server push HTTP/2 permette al server di inviare ("push") più dati di quelli richiesti dal client. Questo consente al server di fornire dati che sa essere necessari ad un web browser per completare la pagina, senza attendere che il browser esamini la prima risposta e senza l'overhead di un ciclo di richiesta addizionale

# HTTP formato richiesta

- HTTP è un protocollo testuale
- I messaggi sono costituiti da sequenze di byte
- Ogni byte identifica un carattere secondo la tabella ASCII
- Il payload dei messaggi può essere comunque anche in formato binario (es. un'immagine GIF, un video, ecc.)



# Header HTTP

- Gli header sono costituiti da insiemi di coppie (nome: valore) che specificano caratteristiche del messaggio trasmesso o ricevuto:
  - Header generali della trasmissione
    - Data, codifica, versione, tipo di comunicazione, ecc.
  - Header relativi all'entità trasmessa
    - Content-type, Content-Length, data di scadenza, ecc.
  - Header riguardo la richiesta effettuata
    - Chi fa la richiesta, a chi viene fatta la richiesta, che tipo di caratteristiche il client è in grado di accettare, quale autorizzazione, ecc.
  - Header della risposta generata
    - Che server dà la risposta, che tipo di autorizzazione è necessaria, ecc.

# HTTP metodi richiesta

- In HTTP/1.1 il campo Method può assumere i seguenti valori:
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
- Un metodo HTTP può essere:
- Sicuro: non genera cambiamenti allo stato interno del server
  - I metodi GET e HEAD sono sicuri
- Idempotente: l'effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta
  - I metodi HTTP sono tutti idempotenti tranne POST

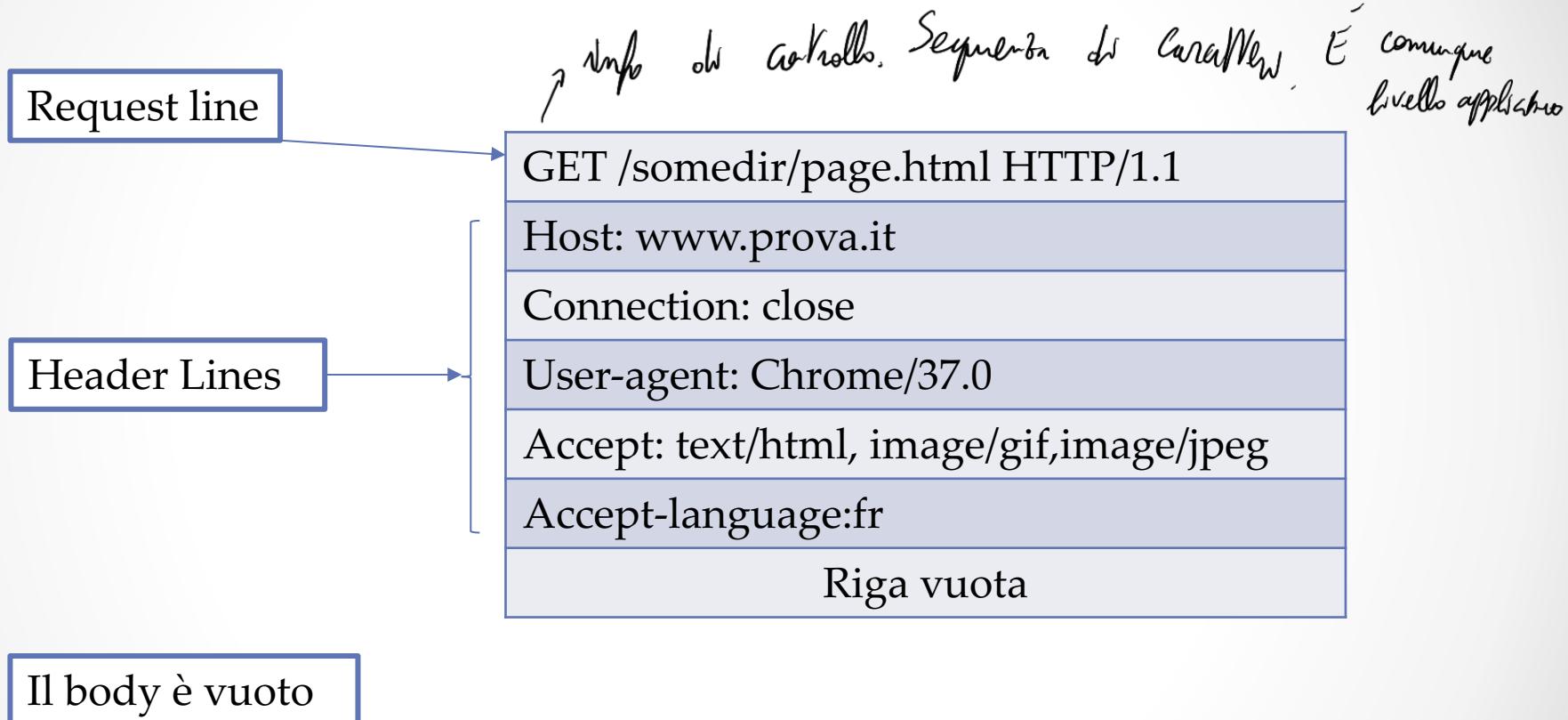
↳ es. immagine GET

# HTTP metodi richiesta: **GET**

## GET

- Serve per richiedere una risorsa ad un server
- È il metodo più usato: 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
- GET è un metodo sicuro ed idempotente
- 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

# HTTP metodi richiesta: GET



# HTTP metodi richiesta: HEAD

- Il **metodo HEAD** è simile al metodo GET, ma il **server deve rispondere soltanto con gli header relativi, senza body**  
↳ solo con header della risposta
- Viene usato per **verificare**:
  - **Validità** di un URL e per apprendere le caratteristiche della risorsa rappresentate nell'header del messaggio di risposta
  - **Coerenza** di una copia della risorsa già disponibile nella cache del browser rispetto all'originale sul server
  - **Accessibilità**: se è richiesta autenticazione

# HTTP metodi richiesta: POST

## 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 CGI sul server
- POST è un metodo non sicuro e non idempotente
  - 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 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

*Posso aggiungere anche risorse sul server, e questo può avere risposte diverse se per esempio crea qualcosa che già c'è.*

# HTTP invio dati GET e POST

L'invio di dati da un client ad un server ad esempio con una form può avvenire sia usando il metodo GET che il metodo POST

→ Parametri vanno nel body

```
<form action="/action_page.php" method="post"  
target="_blank">  
    <label for="fname">First name:</label>  
    <input type="text" id="fname" name="fname"><br><br>  
    <label for="lname">Last name:</label>  
    <input type="text" id="lname" name="lname"><br><br>  
    <input type="submit" value="Submit">  
</form>
```

Il campo method può essere sia get che post

Si può verificare la differenza al link:

[https://www.w3schools.com/tags/tryit.asp?filename=tryhtml\\_form\\_method](https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_form_method)



# HTTP metodi richiesta: PUT e 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
- PUT è un metodo non sicuro ma idempotente

## DELETE

- Richiede la cancellazione della risorsa riferita dall'URL specificato

Sono normalmente disabilitati sui server pubblici!

# HTTP metodi richiesta: OPTIONS e TRACE

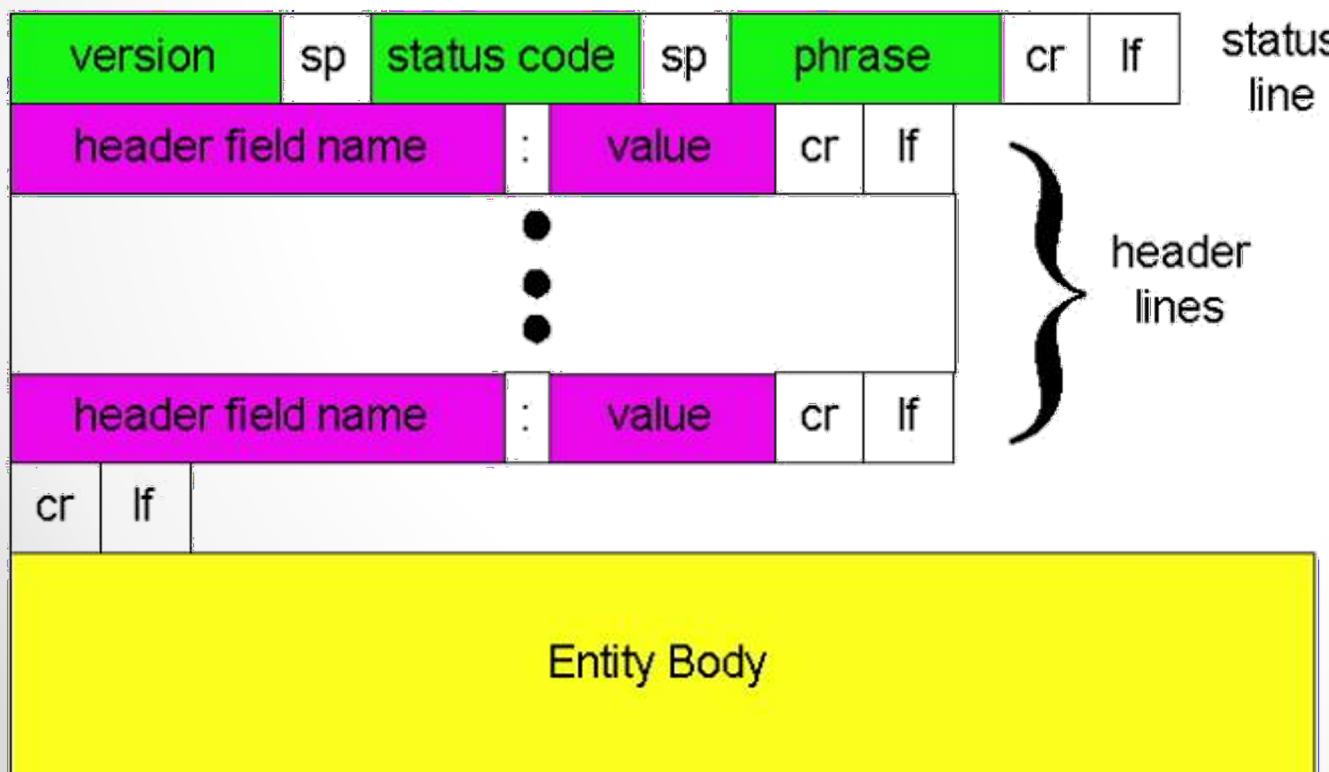
OPTIONS: serve per richiedere informazioni sulle opzioni disponibili per la comunicazione

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 formato risposta

- HTTP è un protocollo testuale
- I messaggi sono costituiti da sequenze di byte
- Ogni byte identifica un carattere secondo la tabella ASCII
- Il payload dei messaggi può essere comunque anche in formato binario (es. un'immagine GIF, un video, ecc.)



CR = Carriage Return  
LF = Line Feed

# HTTP formato risposta

HTTP/1.1 200 OK

Connection: close

Date: Thu, 06 Aug 2008 12:00:15 GMT

Server: Apache/2.3.0 (Unix)

Last-Modified: Mon, 22 Jun 2008

Content-Length: 6821

Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.51  
Transitional//EN"> <html>...</html>

# HTTP formato risposta: 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

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 interno

# Status code: esempi

**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** (metodo non conosciuto dal server)

**501 Not implemented** (metodo non conosciuto dal server)

# 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) *tipo di info che sta trasportando*
- **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

# 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)

# Header di entità

Gli header dell'entità danno informazioni sul body del messaggio, o, se non vi è body, sulla risorsa specificata

- **Content-Type:** oggetto/formato *Cos'è lo header*
  - Ogni coppia oggetto/formato costituisce un 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

# I Cookie

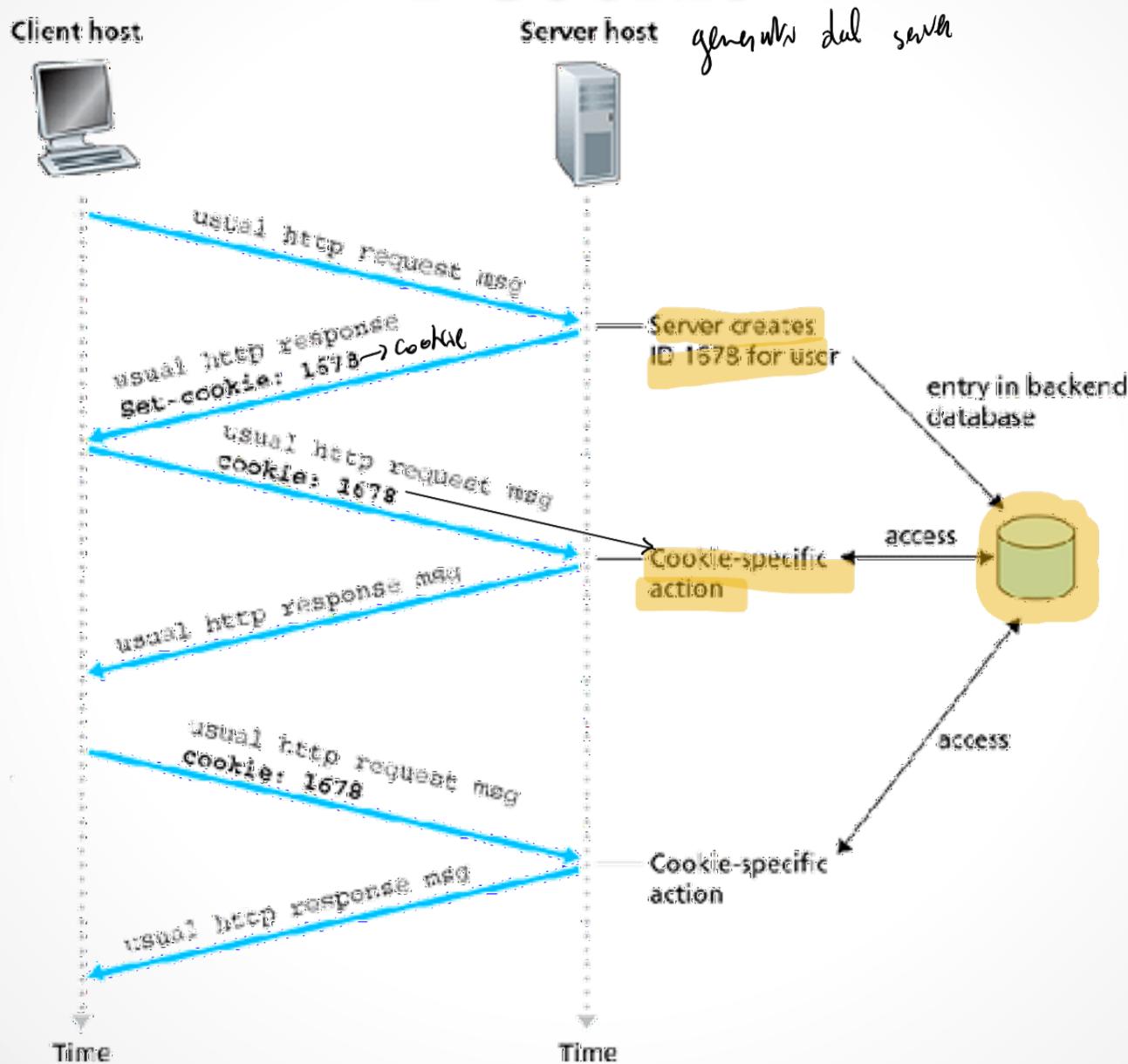
Parallelamente alle sequenze request/response, il protocollo prevede una struttura dati che si muove come un token, dal client al server e viceversa: **i cookie**

I cookie possono essere generati sia dal client che dal server  
Tramite un cookie il client mantiene lo stato di precedenti connessioni

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

# I Cookie



# I Cookie: Header

Il meccanismo dei cookie dunque definisce due nuovi possibili header: uno per la risposta, ed uno per le richieste successive

- Set-Cookie: header della risposta uno che fa un set cookie
  - il client può memorizzarlo (se vuole) 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
- Un browser può essere configurato per accettare o rifiutare i cookie
- Alcuni siti web richiedono necessariamente la capacità del browser di accettare i cookies

# I Cookie: struttura

I cookie sono una collezione di stringhe:

- **Key:** identifica univocamente un cookie all'interno di un dominio:<sup>un id</sup>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 (permette la scadenza di una sessione)
- **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

Un cookie per il quale non è specificata la durata è detto un **session cookie** e dura finché il browser non è chiuso

# Web Caching

Si trovano sempre  
dal letto mosso

- 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
- Una Web cache memorizza i documenti che la attraversano
- Tipi di Web cache
  - User Agent Cache
  - Proxy Cache

# 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 comunque ora molto rilevante per i dispositivi mobili al fine di consentire agli utenti di lavorare con connettività intermittente ma anche per ridurre latenze dovute a caricamento di elementi statici (icone, sfondi, ...)

Anche strumenti addizionali, es. Google Gears, si basano su questo concetto

# Proxy Cache

✓ separato dal client

## Forward Proxy Cache

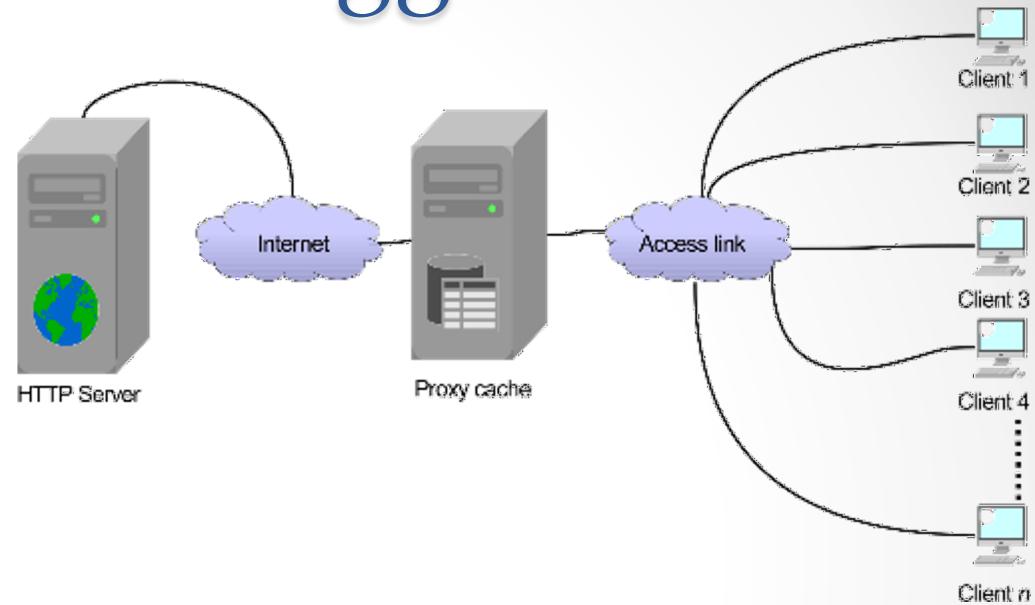
- Servono per ridurre le necessità di banda
- Es. rete locale aziendale, Università, ecc.
- Il proxy intercetta il traffico e mette in cache le pagine
- Successive richieste non provocano lo scaricamento di ulteriori copie delle pagine al server

## Reverse (o server-side) Proxy Cache

- E' presente una Gateway cache
- Operano per conto del server e consentono di ridurre il carico computazionale delle macchine
- I client non sono in grado di capire se le pagine arrivano dal server o dal gateway
- Internet Caching Protocol per il coordinamento fra diverse cache.

# Cache: vantaggi

1. Il browser stabilisce una connessione TCP con la cache e invia una richiesta http alla cache
2. La cache controlla se ha una versione della risorsa localmente
3. In caso positivo invia l'oggetto richiesto all'interno di una risposta http al client
4. In caso negativo apre una connessione TCP con il server, invia una richiesta http per richiedere l'oggetto
5. Il server risponde con l'oggetto
6. La cache archivia una copia e risponde al client con l'oggetto

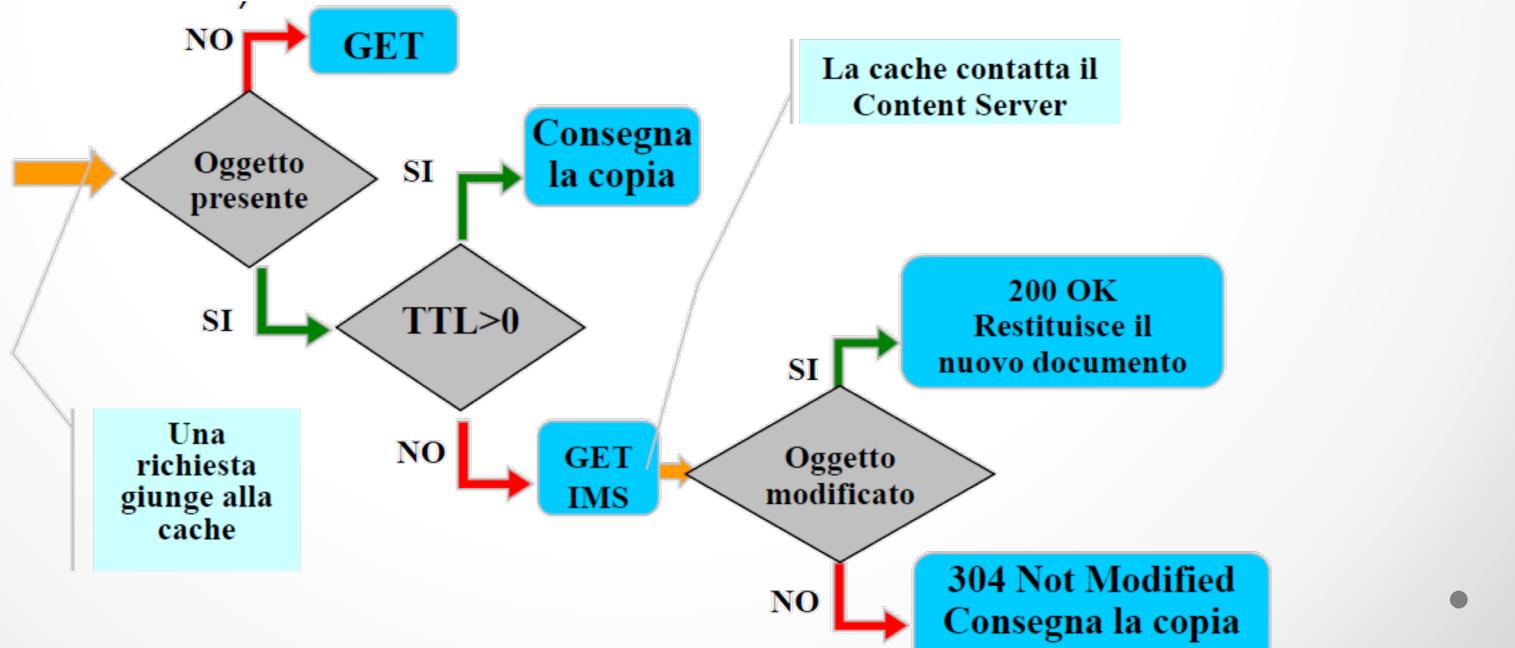


## Vantaggi:

1. Si può ridurre il tempo di risposta a una richiesta del client, soprattutto se il collo di bottiglia è la larghezza di banda tra client e server
2. La cache può ridurre il traffico su un link di accesso internet di una istituzione
3. Si può ridurre il traffico di Internet migliorando le performance di tutte le applicazioni

# 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)



# Caching cooperativo

- Utilizzando cache web multiple, poste in diversi punti della rete internet, è possibile cooperare incrementando le performance generale.
- Per esempio, la cache di una istituzione può essere configurata per inviare le richieste http a una cache in una colonna portante di ISP a livello nazionale. In questo caso, se la cache dell'istituzione non contiene l'oggetto richiesto, essa invia la richiesta http alla cache nazionale.
- La cache nazionale invia poi l'oggetto (in una risposta http) alla cache dell'istituzione, che a sua volta la inoltra al browser richiedente.
- Passare attraverso una cache di più alto livello migliora le possibilità perché ha un numero di utilizzatori e quindi maggiori hit rate
  - ↳ Possibilità di became

# Browser Web

→ Da parte dello user agent  
lavora al browser

Esistono oggi una molteplicità di browser

- Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, Opera, In passato NCSA Mosaic, Internet Explorer, Netscape Navigator, ...
- Un browser è costituito da:
  - un browser engine
  - un'interfaccia utente (UI)

Un browser engine è, a sua volta, composto da

- Un layout engine che decodifica e visualizza il documento HTML e gli oggetti multimediali in essa contenuti, tenendo in conto le indicazioni contenute in file CSS che determinano l'aspetto grafico (stile)
- Un JavaScript engine che esegue il codice JavaScript encapsulato nel documento HTML o contenuto in altri file esterni (file .js)

# HTTP senza Browser

- In varie situazioni il ruolo di client HTTP può essere assunto da programmi che devono interagire con un server per ottenere informazioni tramite il protocollo HTTP
- Un client HTTP attivabile da command-line: wget
  - Utilizzabile sia in Linux/Unix che Windows
  - <http://gnuwin32.sourceforge.net/packages/wget.htm> (versione Win)
  - <https://shapeshed.com/unix-wget/>
  - wget <URL> [OPZIONI]
- Ci sono opzioni che permettono di scaricare ricorsivamente tutti i file puntati dalla prima URL (crawling)
- In questo modo si può ricavare localmente una copia di un intero sito web

# HTTP non solo WEB

- HTTP non è utilizzato solo per il Web
- Ad esempio:
  - Web Services e SOA (Service Oriented Architecture)
  - Video streaming
    - DASH: Dynamic Adaptive Streaming over HTTP
- Peer-to-peer