

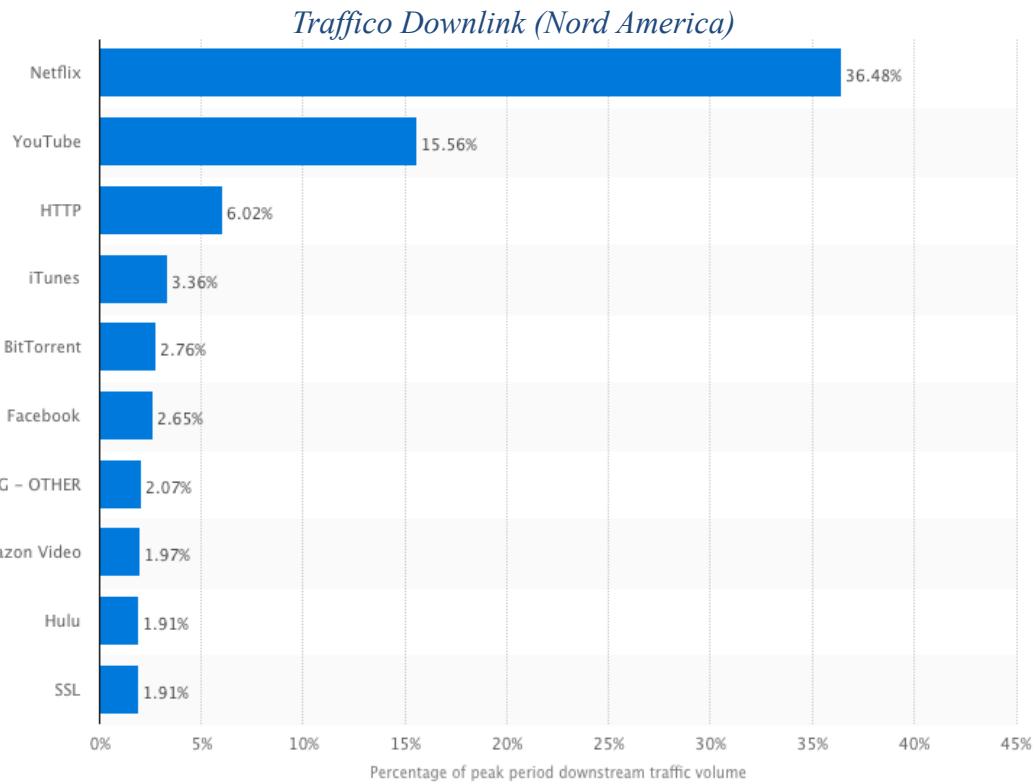


# **Il Livello Applicativo**

**Processi e socket, Web, Mail, DNS,  
peer-to-peer**

# Alcune applicazioni di rete

- **World Wide Web**
  - **HTTP**
- **Posta elettronica:**
  - **SMTP**, Gmail
- **Social networking:**
  - Facebook, Twitter, Instagram, Snapchat, ecc.. (social networking)
- **P2P file sharing:** BitTorrent, eMule, ecc..
- **Video streaming:**
  - NetFlix, YouTube, Hulu
- **Telefonia/videoconf:**
  - Skype, Teams, ecc..
- **Network games**
- **Video conference**
- **Massive parallel computing**
- **Instant messaging**
- **Remote login:**
  - TELNET
- ...

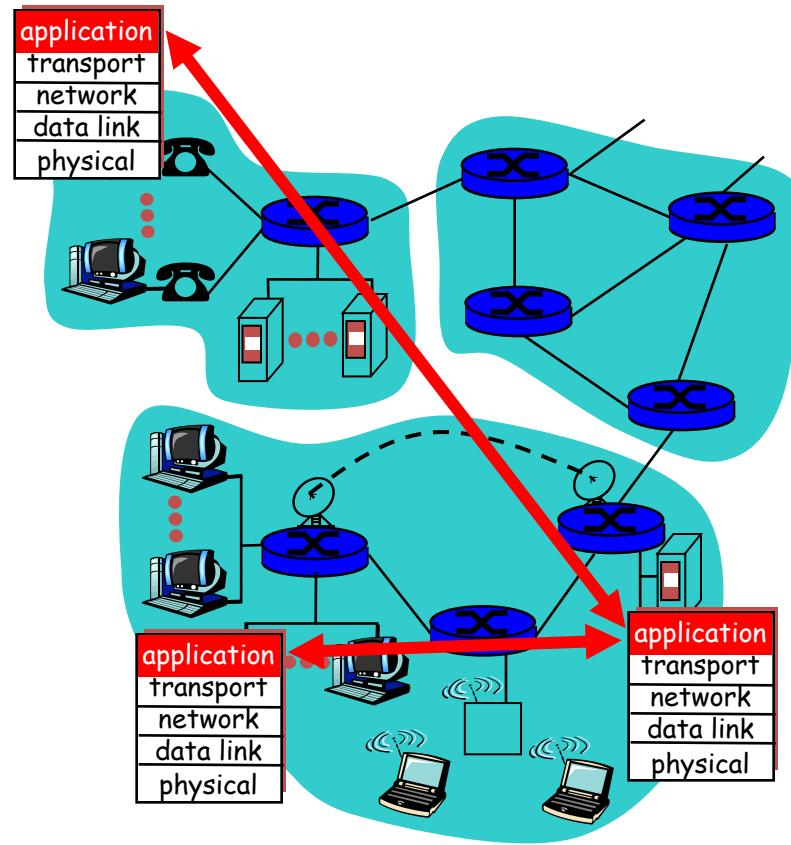


Source: [www.statista.com](http://www.statista.com) (2016)



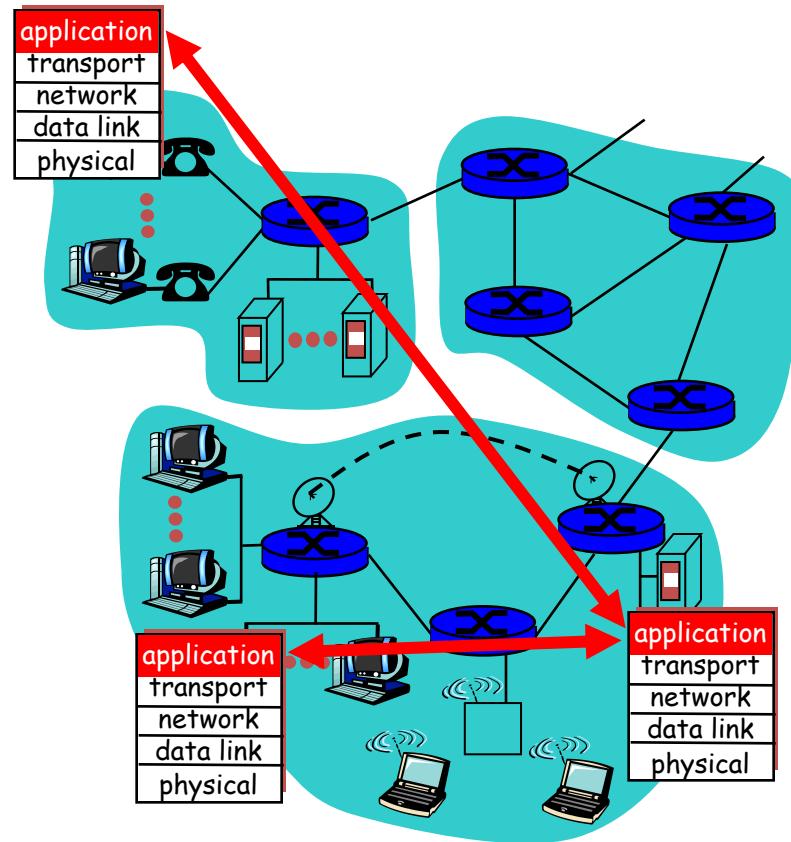
# Creare un'applicazione di rete

- **Scrivere un software che:**
  - possa essere eseguito su diversi terminali e
  - possa comunicare tramite la rete
- **Esempio:** il browser web (*FireFox, Safari, Chrome, ecc..*) è un software “in esecuzione” su un dispositivo che comunica con un software in esecuzione su un server web ([www.google.com](http://www.google.com), [www.amazon.com](http://www.amazon.com), ecc..)



# Creare un'applicazione di rete

- Inventare una nuova applicazione non richiede di cambiare il software della rete
- I nodi della rete non hanno software applicativo
- Le applicazioni sono solo nei terminali e possono essere facilmente sviluppate e diffuse



# Comunicazione tra processi

- **Host:** dispositivo d'utente
  - *Laptop, smartphone, desktop*
- **Processo:** programma software in esecuzione su un *host*
  - Molti processi possono essere in esecuzione simultaneamente sullo stesso *host*
- **Comunicazione inter-processo (IPC):** tecnologie software il cui scopo è di consentire a diversi processi di comunicare scambiandosi informazioni e dati
  - Processi che risiedono sullo stesso *host*
  - **Processi che risiedono su *host* diversi (Serve una Rete!)**



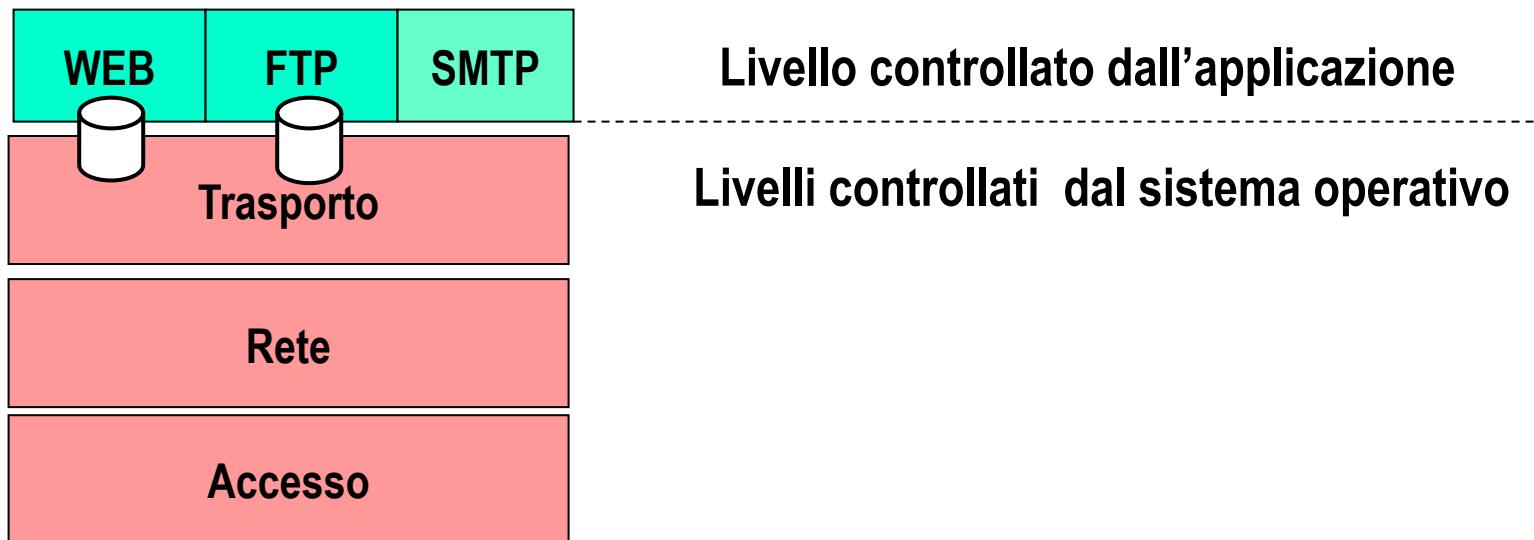
# Ingredienti per la comunicazione tra processi remoti

- **Indirizzamento dei processi** (conoscere il “numero di telefono” dell’interlocutore)
- **Protocollo di scambio dati** (decidere la “lingua” con cui si parla)
  - Tipi di messaggi scambiati: Richieste, risposte
  - Sintassi dei messaggi: Campi del messaggio e delimitatori
  - Semantica dei messaggi: Significato dei campi
  - Regole su come e quando inviare e ricevere i messaggi



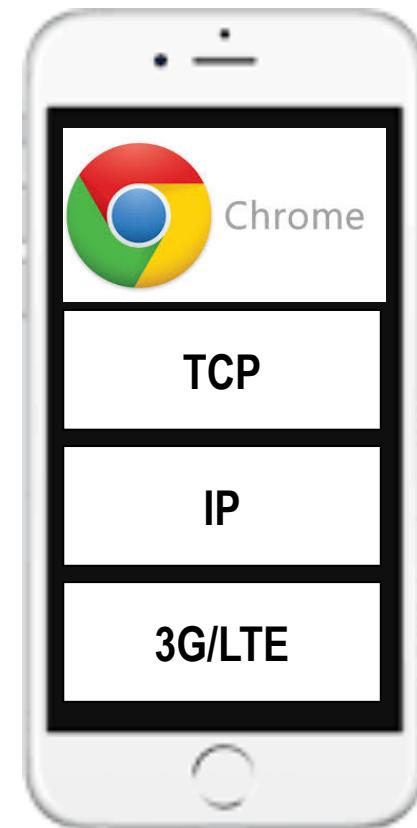
# Indirizzamento di processi applicativi

- Lo scambio di messaggi fra i processi applicativi avviene utilizzando i servizi dei livelli inferiori attraverso i SAP (*Service Access Point*)
- Ogni processo è associato ad un SAP



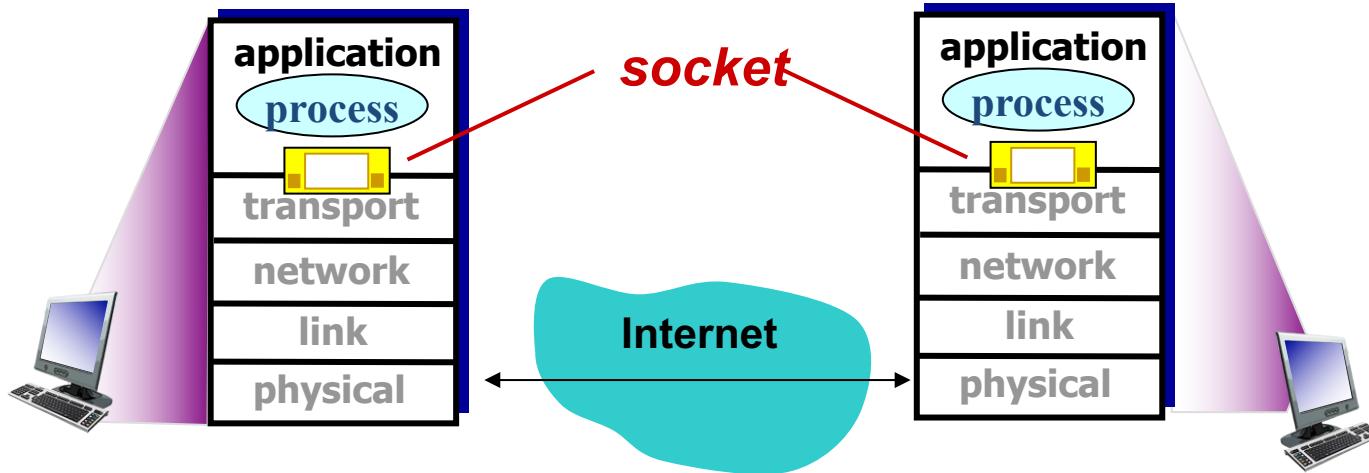
# Indirizzamento di processi applicativi

- Cosa serve per identificare il mio browser *Chrome* in esecuzione sul mio *host*?
  - Indirizzo del mio *host*
    - Indirizzo IP univoco di 32 bit (di cui parleremo ampiamente in seguito)
  - Indirizzo del SAP del processo in esecuzione sul mio *host*
    - numero di porta



# Indirizzamento di processi applicativi

- Indirizzo di un processo in esecuzione = indirizzo IP + numero di porta = **socket**
- Esempio:
  - Il server web del Politecnico [www.polimi.it](http://www.polimi.it) è raggiungibile a: 131.175.12.34/80
- Le **socket** sono delle porte di comunicazione
  - Il processo trasmittente mette il messaggio fuori dalla porta
  - La rete raccoglie il messaggio e lo trasporta fino alla porta del destinatario
- Attività di laboratorio su **socket programming**



# Requisiti delle applicazioni

- **Affidabilità**
  - Alcune applicazioni possono tollerare perdite parziali (ad es. audio)
  - Altre applicazioni richiedono a completa affidabilità (ad es. file transfer, telnet)
- **Ritardo**
  - Alcune applicazioni richiedono basso ritardo (ad es. *Internet telephony, interactive games*)
- **Banda**
  - Alcune applicazioni richiedono un minimo di velocità di trasferimento (ad es. appl. multimediali)
  - Altre applicazioni si adattano alla velocità disponibile (“appl. elastiche”)



# Requisiti delle applicazioni

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100msec
instant messaging	no loss	elastic	yes and no



# Quale servizio di trasporto scegliere

## Servizio TCP:

- *Connection-oriented*: instaurazione connessione prima delle scambio dati
- *Trasporto affidabile* senza perdita di dati
- *Controllo di flusso*: il trasmittitore regola la velocità in base al ricevitore
- *Controllo di congestione*: per impedire di sovraccaricare la rete
- *Non fornisce*: garanzie di ritardo e di banda

## Servizio UDP:

- Trasferimento non affidabile
- senza connessione
- senza controllo sul traffico
- senza garanzie
- Trasferimento “veloce”



# Applicazioni e protocolli di trasporto

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (YouTube, NetFlix) TCP, UDP, DASH RTP	
Internet telephony (e.g., Vonage, Dialpad)		typically UDP

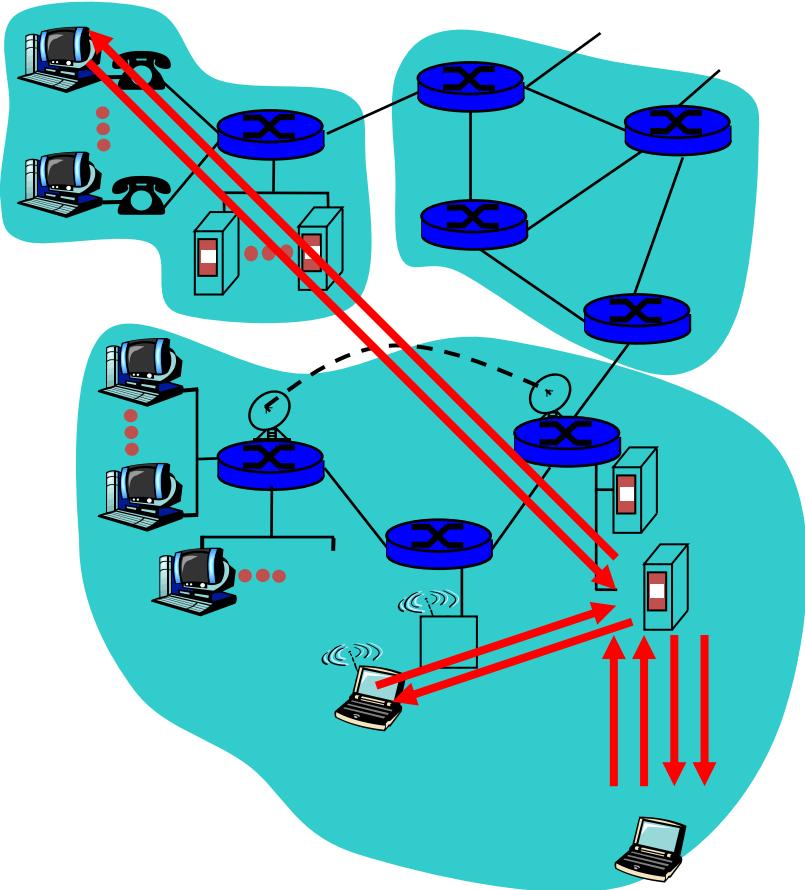


# Architetture applicative

- ***Client-server***
  - I dispositivi coinvolti nella comunicazione implementano o solo il processo *client* o solo il processo *server*
  - I dispositivi *client server* hanno caratteristiche diverse
  - I *client* possono solo eseguire richieste
  - I *server* possono solo rispondere a richieste ricevute
- ***Peer-to-peer (P2P)***
  - I dispositivi implementano tutti sia il processo *client* che quello *server*
- ***Ibrida***



# Architettura *client-server*



## *Server:*

- Host sempre attivo
- Indirizzo IP permanente
- Possibilità di utilizzo di macchine in *cluster*
- Possono ricevere richieste da molti *client*

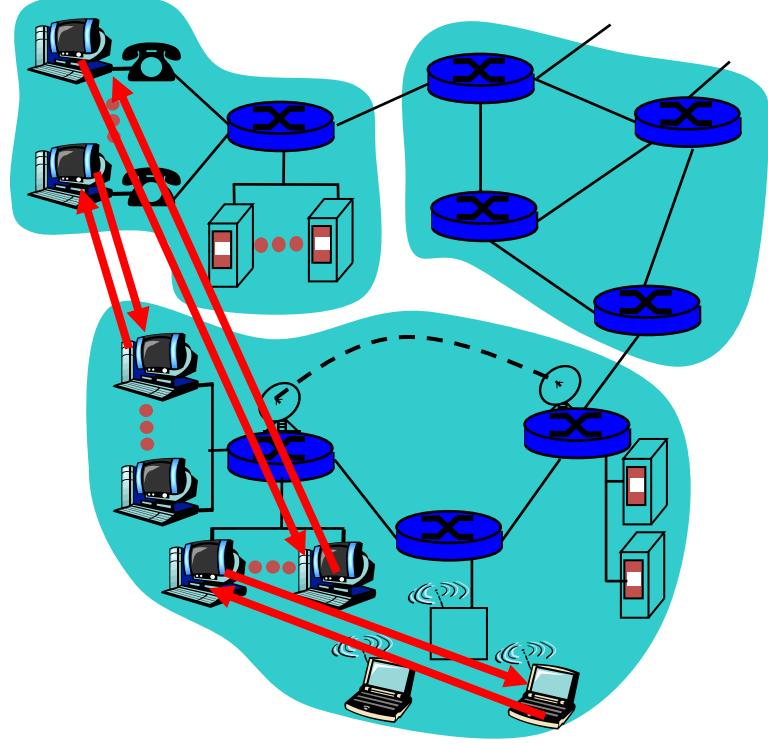
## *Client:*

- Comunicano con il *server*
- Possono essere connessi in modo discontinuo
- Possono cambiare indirizzo IP
- Non comunicano con altri *client*
- Possono inviare molte richieste allo stesso *server*



# Architettura P2P (pura)

- Non ci sono *server* sempre connessi
- Terminali (*peers*) comunicano direttamente
- I *peers* sono collegati in modo intermittente e possono cambiare indirizzo IP
- Esempio: *BitTorrent*



Fortemente scalabile  
ma difficile da gestire





# Il servizio di Web Browsing

Hyper Text Transfer Protocol (HTTP)  
RFC 1945, 2616

# Cosa contengono i messaggi HTTP – le Pagine Web

- Breve ripasso:
  - le *pagine web* sono fatte di *oggetti*
  - gli *oggetti* possono essere file HTML file, immagini JPEG, applet Java, file audio file, file video, collegamenti ad altre pagine web..
  - generalmente le pagine web hanno un file HTML (o *php*) base che “chiama” gli altri *oggetti*
  - ogni *oggetto* è indirizzato da una *Uniform Resource Locator (URL)*, es:

**http://www.polimi.it:80/index.html**

↑  
Indica il protocollo applicativo

↑  
Indica l'indirizzo di rete del server

↑  
Indica il numero di porta (opzionale)

↑  
Indica la pagina web richieste



# La comunicazione HTTP

- Architettura *client/server*:
  - **client**: browser che effettua richieste HTTP di pagine web (oggetti), le riceve e le mostra all'utente finale
  - **server**: Web server inviano gli oggetti richiesti tramite risposte HTTP
- Nessuna memoria sulle richieste viene mantenuta nei server sulle richieste passate ricevute da un client (protocollo *stateless*)



# La comunicazione HTTP

- HTTP si appoggia su TCP a livello di trasporto:
  1. Il client HTTP inizia una connessione TCP verso il server (porta 80)
  2. Il server HTTP accetta connessioni TCP da client HTTP
  3. ***Client e Server HTTP si scambiano informazioni (pagine web e messaggi di controllo)***
  4. La connessione TCP tra client e server HTTP viene chiusa

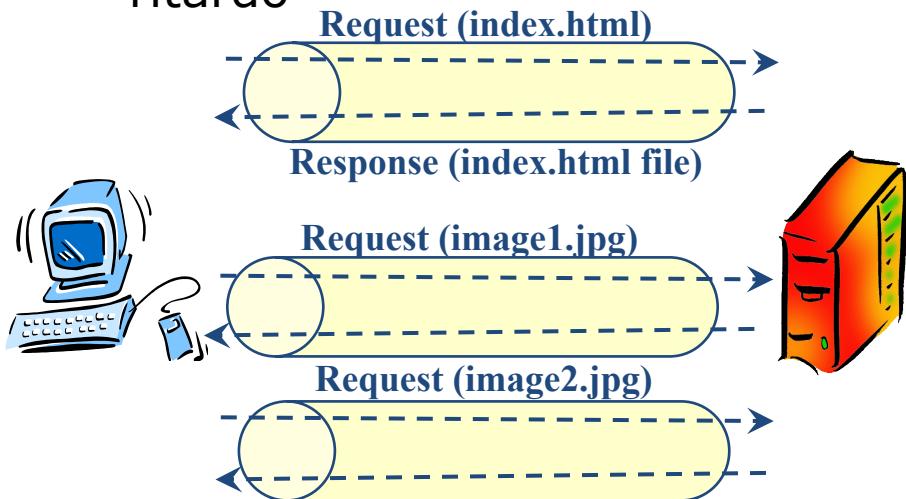
*Vedremo 1, 2 e 4 nei laboratori su socket programming*



# Modalità di connessione tra *client* e *server* HTTP

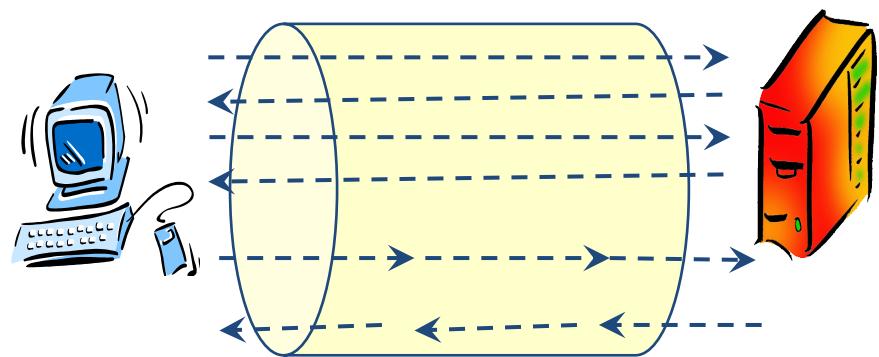
## Connessione non persistente

- Una connessione TCP per una sola sessione richiesta-risposta; inviato l'oggetto il server chiude la connessione TCP
- La procedura viene ripetuta per tutti i file collegati al documento HTML base
- Le connessioni TCP per più oggetti possono essere aperte in parallelo per minimizzare il ritardo



## Connessione persistente

- La connessione TCP rimane aperta e può essere usata per trasferire più oggetti della stessa pagina web o più pagine web
  - *without pipelining*: richieste HTTP inviate in serie
  - *with pipelining*: richieste HTTP inviate in parallelo (default mode HTTP v1.1)



# Esempio di connessione non persistente

L'utente digita nel browser la URL:

[www.polimi.it/home/index.html](http://www.polimi.it/home/index.html)

(l'HTML contiene testo e riferimenti a 10 immagini jpeg)

1a. Il client HTTP inizia una connessione TCP verso il server HTTP [www.polimi.it](http://www.polimi.it) sulla porta 80

1b. il server HTTP in esecuzione su [www.polimi.it](http://www.polimi.it) in attesa sulla porta 80 accetta la connessione e notifica il client

2 il client HTTP invia una richiesta HTTP (contenente la URL) tramite la connessione TCP. La richiesta indica che il client vuole l'oggetto /home/index.html

3 Il server HTTP riceve la richiesta HTTP ed invia una risposta HTTP contenente il file HTML

*tempo*

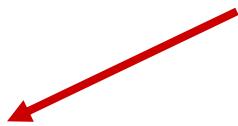


# Esempio di connessione non persistente

tempo  
↓

4. Il server HTTP chiude la connessione TCP.

5. HTTP client riceve il messaggio di risposta contenente il file html e visualizza la pagina. Analizzando la pagina html scopre gli altri oggetti jpeg collegati.

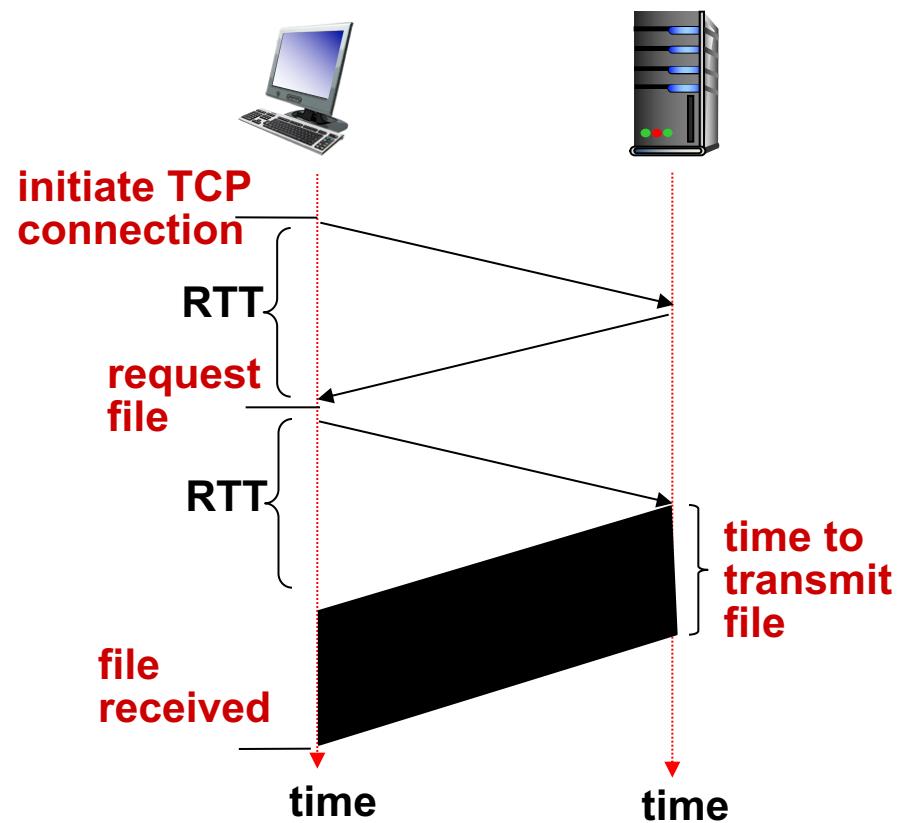


I passi da 1 a 5 sono ripetuti per ognuna delle 10 immagini JPEG indicate dal file HTML



# Stima del tempo di trasferimento in HTTP

- *Round trip Time (RTT)*: tempo per trasferire un messaggio “piccolo” dal *client* al *server* e ritorno
- Tempo di risposta di HTTP:
  - un RTT per iniziare la connessione TCP
  - un RTT per inviare i primi byte della richiesta HTTP e ricevere i primi byte di risposta
  - tempo di trasmissione dell’oggetto (file HTML, immagine, ecc..)

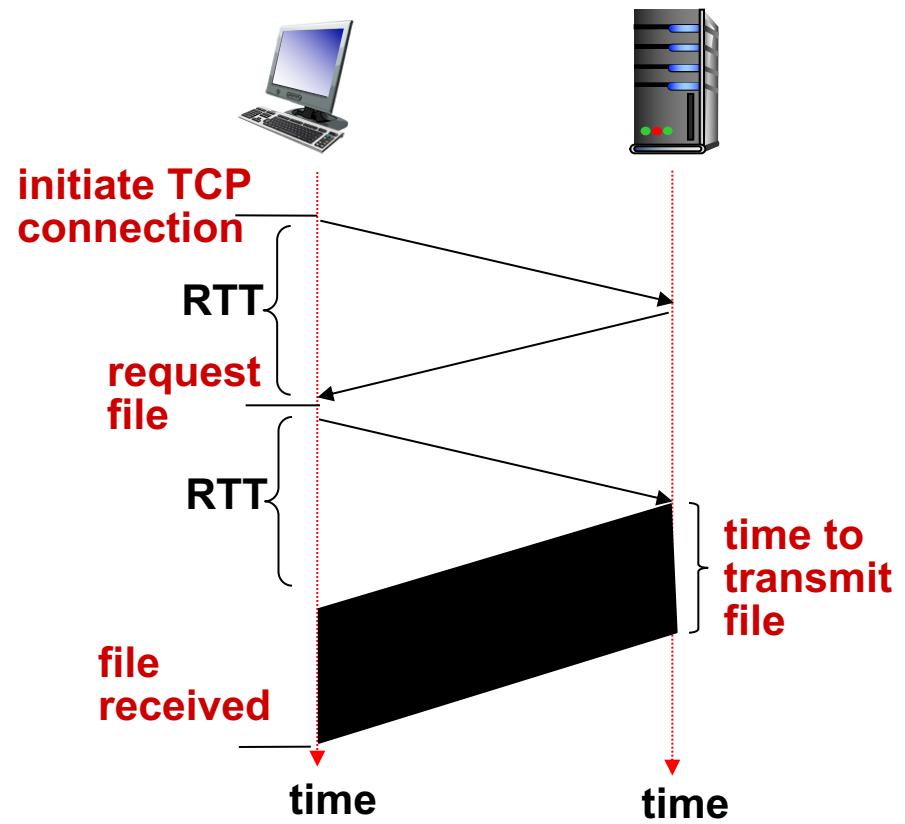


# Stima del tempo di trasferimento in HTTP

- Supponendo che la pagina web sia composta da 10 oggetti (un file HTML e 9 immagini JPEG), il tempo di download dell'intera pagina web è:

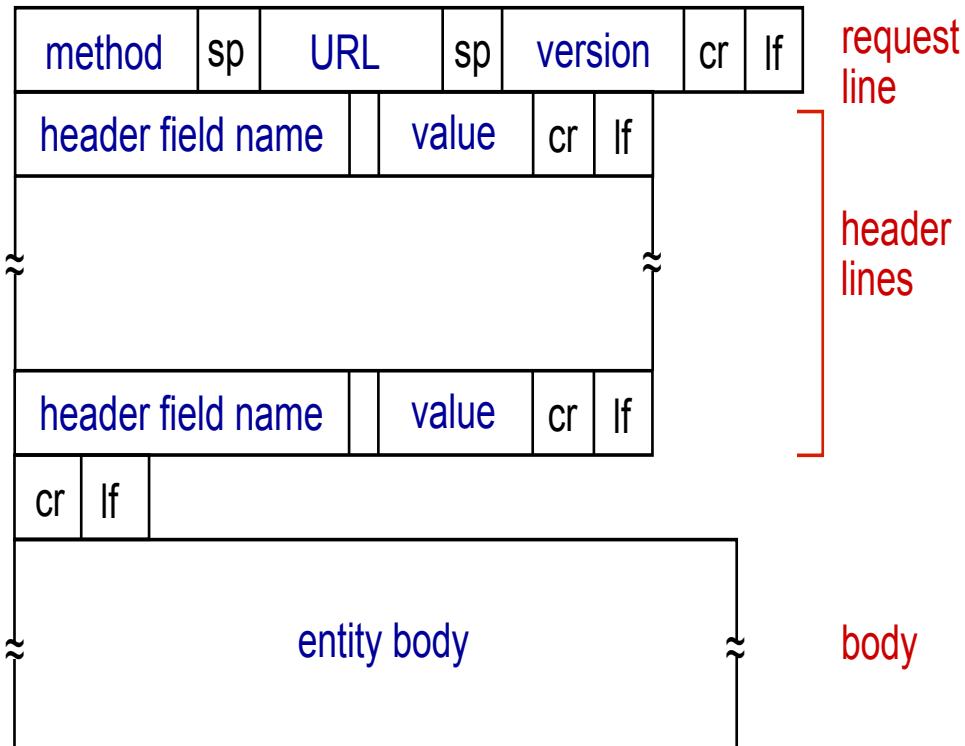
$$T_{nonpers} = \sum_{i=0}^{10} (2RTT + T_i)$$

$$T_{pers} = RTT + \sum_{i=0}^{10} (RTT + T_i)$$



# Le richieste HTTP

- I messaggi HTTP sono codificati in ASCII (*human-readable*)



```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```



# Esempi di Metodi HTTP

<b>GET</b>	E' usato quando il client vuole scaricare un documento dal server. Il documento richiesto è specificato nell'URL. Il server normalmente risponde con il documento richiesto nel corpo del messaggio di risposta.
<b>HEAD</b>	E' usato quando il client non vuole scaricare il documento ma solo alcune informazioni sul documento (come ad esempio la data dell'ultima modifica). Nella risposta il server non inserisce il documento ma solo degli header informativi.
<b>POST</b>	E' usato per fornire degli input al server da utilizzare per un particolare oggetto (di solito un applicativo) identificato nell'URL.
<b>PUT</b>	E' utilizzato per memorizzare un documento nel server. Il documento viene fornito nel corpo del messaggio e la posizione di memorizzazione nell'URL.
<b>DELETE</b>	cancella il documento specificato nella URL



# Esempi di *Header* HTTP

- Gli *header* servono per scambiare informazione di servizio aggiuntiva
- E' possibile inserire più linee di *header* per messaggio

Header name

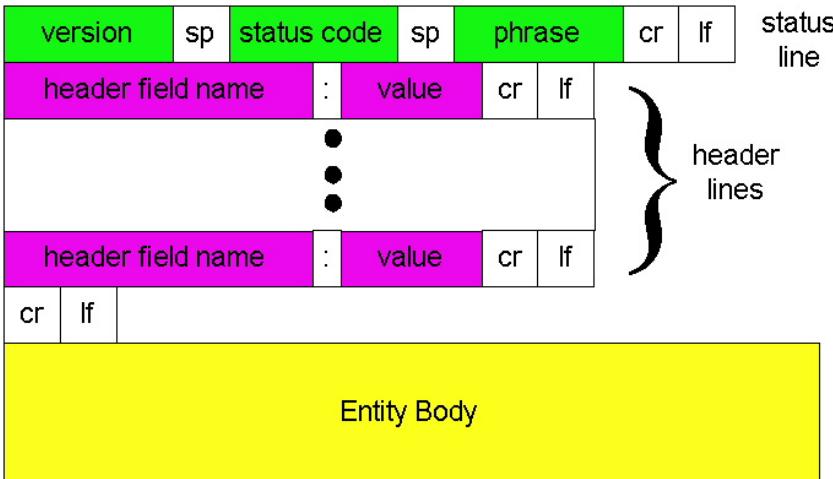
:

Header value

Cache-control	Informazione sulla cache
Accept	Formati accettati
Accept-language	Linguaggio accettato
Authorization	Mostra i permessi del client
If-modified-since	Invia il doc. solo se modificato
User-agent	Tipo di user agent



# Le risposte HTTP



```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS) \r\nLast-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-1\r\n\r\ndata data data data data ...
```

I messaggi contenuti nella **status line** sono identificati da un codice<sup>1</sup>:

**1xx:** informazione

**2xx:** successo

**3xx:** redirezione (richiesta è corretta, è stata rediretta ad un altro server)

**4xx:** errore lato client (richiesta errata)

**5xx:** errore lato server (c'è un problema interno del server)

I messaggi sono accompagnati da un testo “*human readable*”

<sup>1</sup>L'elenco completo è definito nell'RFC 2616

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>



# Scambio di messaggi: un esempio

- Richiesta oggetto

```
GET /ntw/index.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language:it
```

*HTTP è testuale (ASCII)*

- Risposta

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
data data data data ...
```



# HTTP

## ***Download di una pagina via HTTP 1.0***

- Scarichiamo una pagina col protocollo HTTP usando *telnet* dal server web a disposizione sulla macchina virtuale.
- 1) Ci connettiamo al server web sulla porta 80

```
myhost$ telnet localhost 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

- 2) Richiediamo una pagina web (è necessaria una riga vuota alla fine della richiesta: dare 2 [INVIO] )

```
GET /testpage.html HTTP/1.0
```

- 3) La pagina viene scaricata e la connessione viene chiusa



# HTTP

## *Analisi della risposta*

HTTP/1.1 200 OK

Date: Wed, 26 Mar 2008 08:53:44 GMT

Server: Apache/2.2.3 (Debian)

Last-Modified: Wed, 26 Mar 2008 08:49:41 GMT

ETag: "9c3f1-417-26541340"

Accept-Ranges: bytes

Content-Length: 1047

Connection: close

Content-Type: text/html; charset=UTF-8

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

[...]

</html>



# HTTP

## Download di una pagina via HTTP 1.0

- È stata scaricata la pagina, ma non gli elementi *non HTML* che la compongono: in questo caso, mancano le immagini!
- Dal codice HTML leggiamo che si chiamano `img/logo_poli_small.png` e `img/logo-antlab.png`
- 4) Ci connettiamo ancora al server web

```
myhost$ telnet localhost 80 [...]
```

- 5) scarichiamo la prima immagine

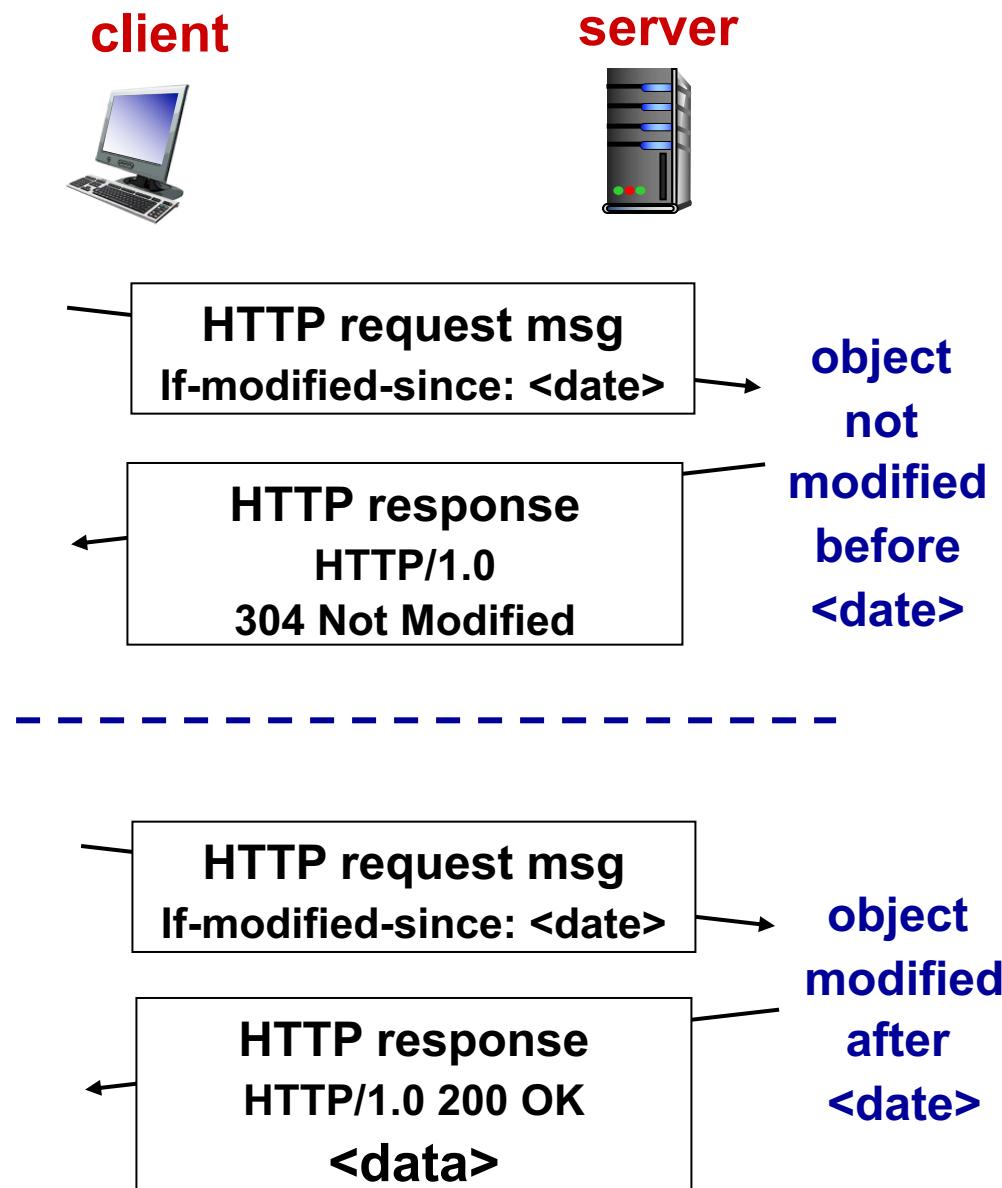
```
GET /img/logo_poli_small.png HTTP/1.0
```

*(ovviamente l'immagine è un file binario, per cui compariranno a monitor un insieme di caratteri)*



# Conditional GET

- **Obiettivo:** non inviare un oggetto richiesto se già presente presso il *client*
- Si inserisce nella richiesta HTTP la data dell'oggetto presente in *cache* locale tramite l'*header*  
*If-modified-since: <date>*
- La risposta HTTP non contiene l'oggetto richiesto se la copia presente al client è aggiornata  
*HTTP/1.0 304 Not Modified*



# Compiti a casa

- Vedere su YouTube video su:
  - come richiedere pagine web con HTTP/1.1  
<https://youtu.be/astJTu59BfY>
  - come inviare GET condizionate  
<https://youtu.be/YfIILUI5S0E>



# Analisi del traffico HTTP

- Molti *browser* (*Chrome*, *Safari*, *Firefox*) offrono strumenti interessanti per la visualizzazione, l'analisi e la valutazione delle prestazioni di traffico HTTP



# Compiti a casa

- Vedere video su canale YouTube su come usare *Chrome Dev Tools*  
<https://developers.google.com/web/tools/chrome-devtools/>  
<https://youtu.be/BdkwJaMjG8k>
- Sfruttando quello che avete imparato nel video, analizzare le seguenti pagine web:
  - [www.polimi.it](http://www.polimi.it)
  - [www.inter.it](http://www.inter.it)
  - [www.cnn.com](http://www.cnn.com)
- In particolare, scoprire il tempo medio di *loading*, il numero di oggetti che compongono la pagina web e la dimensione complessiva della stessa.



# Giocare con HTTP da linea di comando: cURL

- *cURL* è un *tool* da linea di comando per la manipolazione ed il trasferimento di dati basato su URL (vedi <http://curl.haxx.se/>)
- Proviamo qualcosa:
  - Ottenerne una pagina web:  
curl [www.antlab.polimi.it](http://www.antlab.polimi.it)
  - E se vogliamo solo l'intestazione della pagina web?  
curl --head [www.antlab.polimi.it](http://www.antlab.polimi.it)
  - Monitoriamo lo scambio di messaggi tra client e server  
curl --trace-ascii filename.txt [www.antlab.polimi.it](http://www.antlab.polimi.it)
  - E se ci interessano i tempi di richiesta e risposta?  
curl --trace-ascii filename.txt -trace-time [www.antlab.polimi.it](http://www.antlab.polimi.it)



# Compiti a casa

- Vedi video YouTube su come usare cURL per analisi del tempo di *loading* di una pagina web
  - <https://youtu.be/wklvcxY9iTA>
  - <https://youtu.be/ZpTv5lc28m8>
- Usando lo script descritto nel video, valutare e confrontare il tempo di *loading* delle pagine web:
  - [www.polimi.it](http://www.polimi.it)
  - [www.canterbury.ac.nz](http://www.canterbury.ac.nz)
  - [www.stanford.edu](http://www.stanford.edu)



# Mantenere uno “stato” in HTTP: i cookies

## Ingredienti dei cookies:

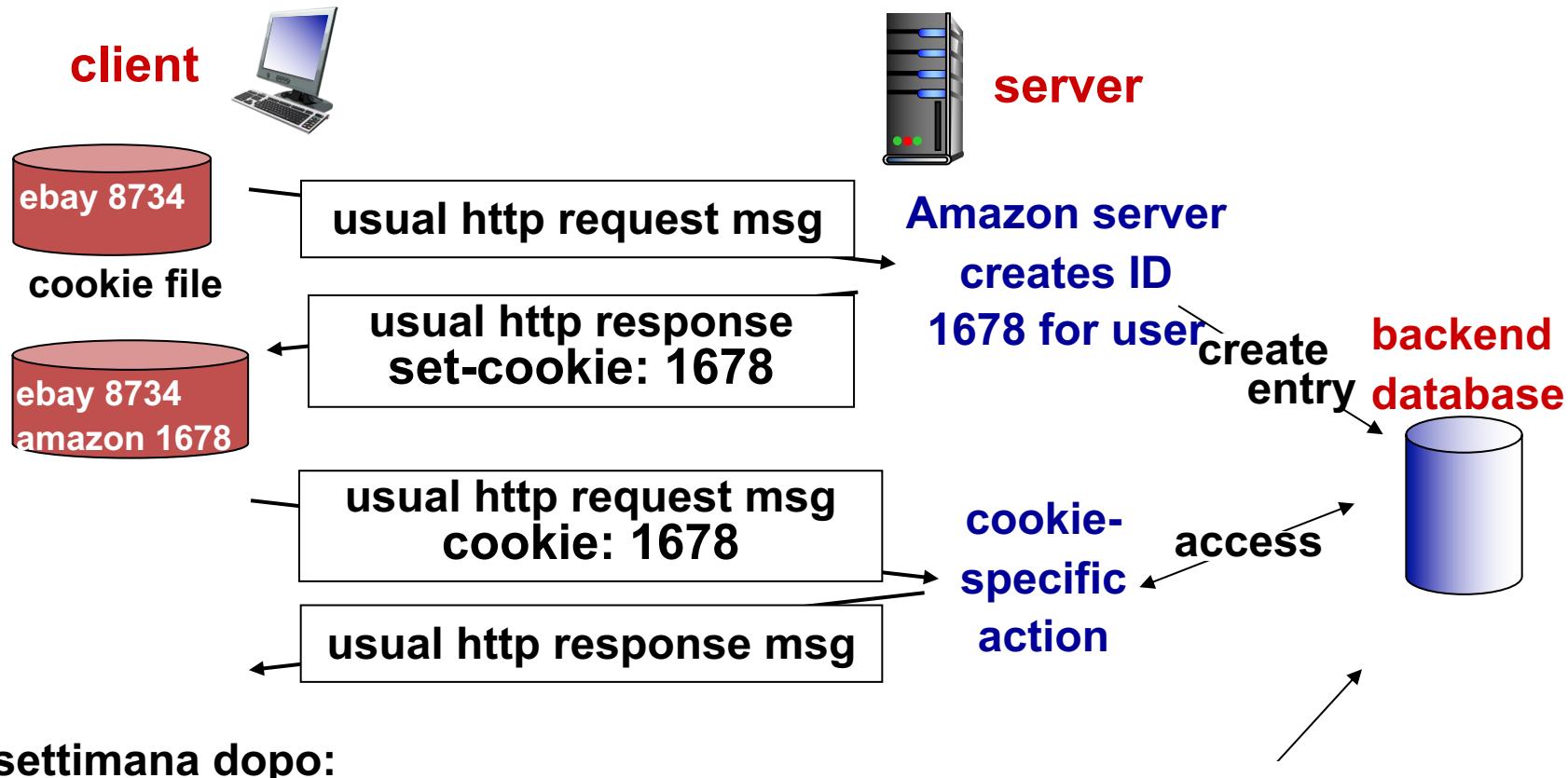
- 1) Un *header cookie* nelle risposte HTTP
- 2) Un *header cookie* nella prossima richiesta HTTP
- 3) Una lista di *cookie* mantenuta sull'*host* (dal *browser*)
- 4) Un data base di *cookie* mantenuto dal sito web

## esempio:

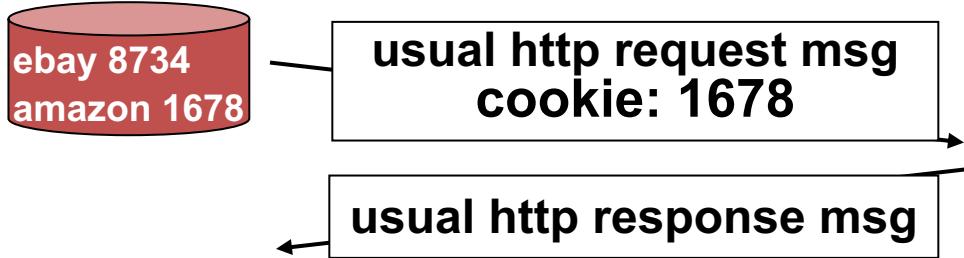
- Matteo visita un sito di e-commerce per la prima volta
- Quando il sito riceve la prima richiesta HTTP, il sito genera:
  - un ID unico
  - una *entry* nel data base locale che corrisponde all'ID creato



# Esempio di uso dei cookie

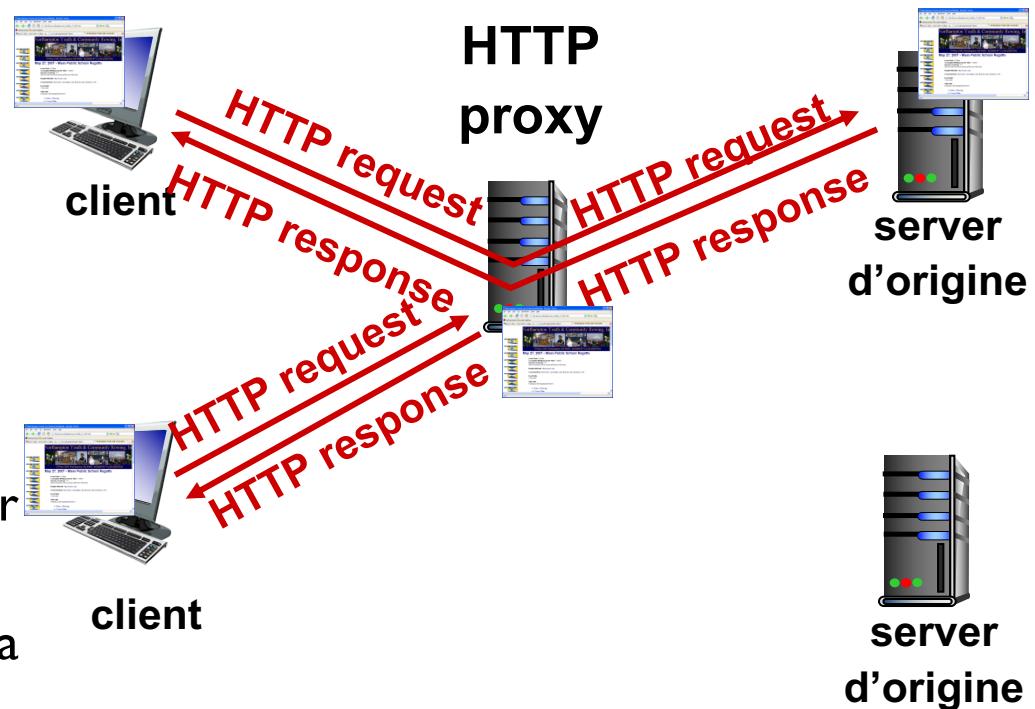


una settimana dopo:



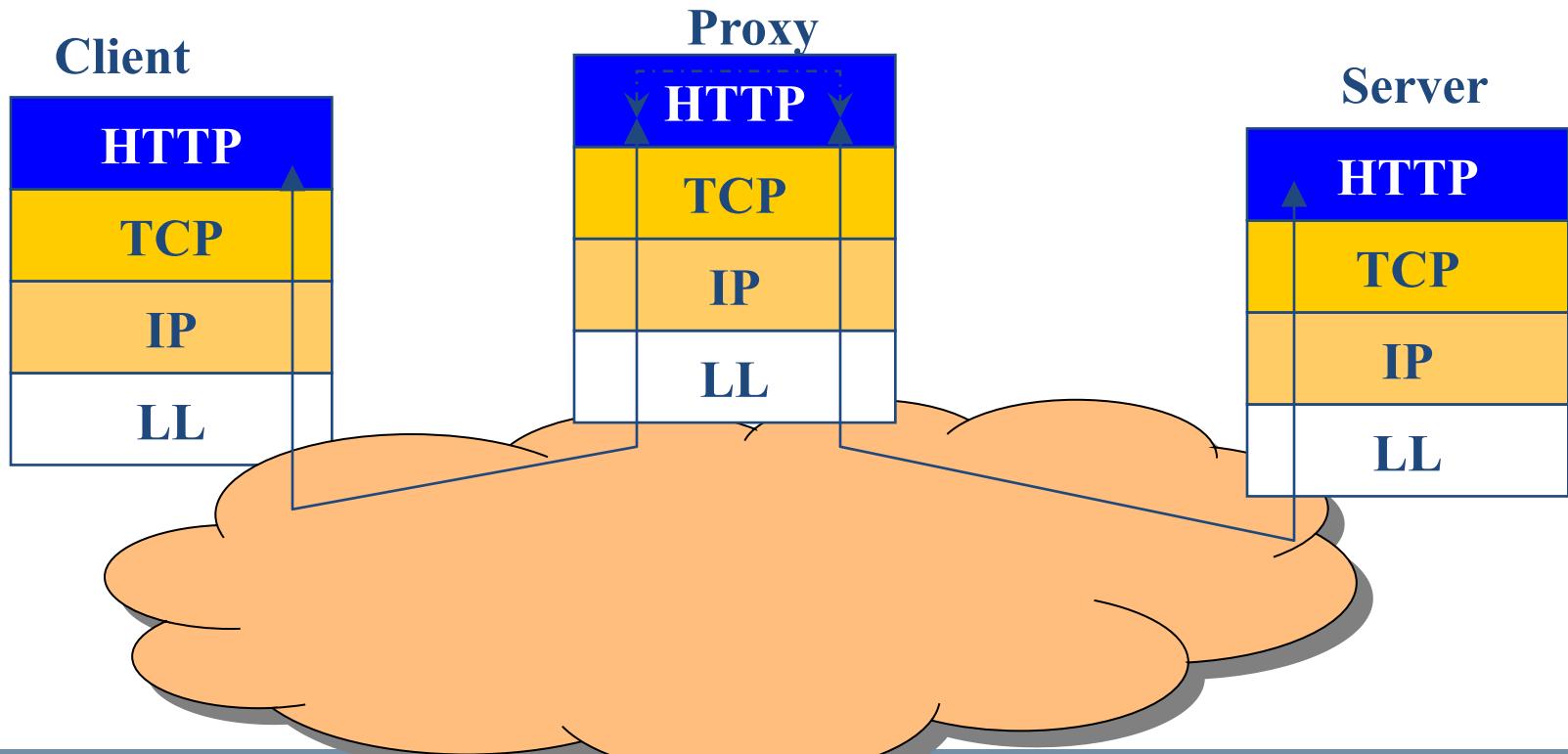
# I proxy HTTP: Cache di rete

- Obiettivo: **rispondere alle richieste HTTP senza coinvolgere il server HTTP**
  - Il client HTTP invia **tutte** le richieste HTTP ad un proxy HTTP:
    - se l'oggetto richiesto è disponibile nella cache del proxy server, il proxy server risponde con l'oggetto
    - altrimenti il proxy recupera l'oggetto dal server d'origine e lo restituisce al client



# I proxy HTTP: Cache di rete

- I proxy sono degli *application gateway*, ovvero degli instradatori di messaggi di livello applicativo
- Devono essere sia *client* (verso i server d'origine) che *server* (verso i *client*)
- Il server vede arrivare tutte le richieste dal proxy (mascheramento degli utenti del proxy)



# HTTP/2 e HTTP/1.1: Differenze in breve

- Obiettivo
  - Ridurre la latenza (o *loading time*) di pagine web
  - Risolvere alcuni problemi di HTTP/1.1



- Il sito [www.gazzetta.it](http://www.gazzetta.it) *include 209 oggetti*
  - HTTP/1.0 consente una connessione per oggetto -> 209 connessioni TCP richieste
  - HTTP/1.1 usa connessioni TCP persistenti ma "seriali" -> se un oggetto è "lento", blocca gli altri (*Head of Line Problem*)



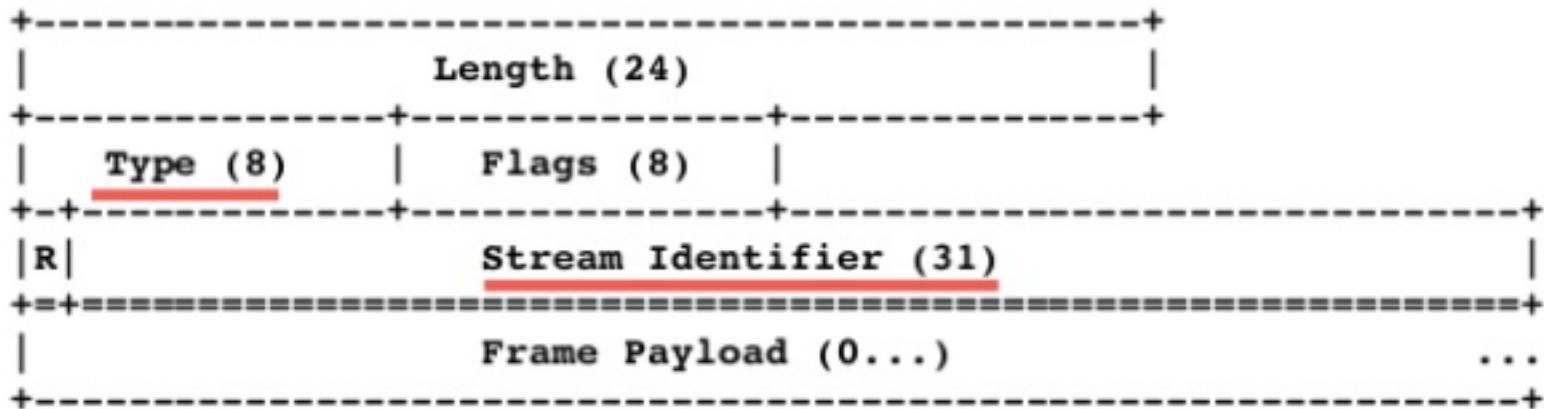
# Caratteristiche di HTTP/2

- HTTP/2 è in formato binario: trasferisce *frame*
- Multiplazione: una connessione TCP per *stream* multipli
- Compressione degli header
- Servizio di *server push*
- Servizio di controllo di flusso (flow control) a livello applicativo
- Si appoggia su TLS (disponibile anche versione in chiaro)

*Quanto si risparmia? Demo <https://http2.akamai.com/demo>*



# HTTP/2 Frames



- Type:
  - DATA: trasporta dati di uno stream
  - HEADERS: usata per aprire uno stream
  - PRIORITY: specifica priorità di uno stream
  - RST\_STREAM: per terminare uno stream
  - SETTINGS: trasporta parametri di configurazione
  - PUSH PROMISE: gestisce il servizio di PUSH
  - PING, GOAWAY, WINDOW\_UPDATE, CONTINUATION:



# HTTP/2: compressione degli header

▼ Request Headers    view parsed  
GET / HTTP/1.1  
Host: www.gazzetta.it  
Connection: keep-alive  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
Accept-Encoding: gzip, deflate, sdch  
Accept-Language: it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4  
Cookie: \_\_gads=ID=7412cf258e0b4ea9:T=1426953830:S=ALNI\_MZyiuEjznRnrtBlttWZ1qNe4MNw; is\_returning=1; \_\_ric=5295%3ASat%20May%2030%202015%2010%3A14%3A18%20GMT+0200%20%28CEST%29%7C; dnaddnt=0; mUserID=5hejoxJCBTnQ; \_\_qca=P0-538237289-1453997156314; \_cb\_ls=1; cpmt\_xa=5295,5498; \_sg\_b\_n=1461756783562; widgetGazzettathirdColumn\_METADATA=%7B%22layoutType%22%3A%22single%22%20%22currentGroup%22%3A%2C%22totalDeals%22%3A3%7D; widgetGazzetta\_USER\_DATA=%7B%22userId%22%3A%22ba10be1b1a6347dc92cd2e2f4003d5f0%22%7D; gvsC>New; incognitoMode=false; GEO\_PLAYLIST\_ACTIVITY=W3sidSI6IjZialUiLCJ0c2wi0jE000MyMDg3NTisIm52IjoxLCJ1cHQi0jE00DMyMDc3NTgsImx0IjoxNDgzMjA4NzUyfV0.; channel=Natural Search|https://www.google.it/|Google - Italy|No Keyword; userid=7D345911-D6EA-A6E2-153A-70B57DFBC241; s\_sq=%5B%BB%5D%5D; utag\_main=v\_id:014f8a939b03000c34f86cdefc3f06079001707100838\$\_sn:579\$ss:1\$\_st:1486131591176\$p\_n:1%3Bexp-session\$ses\_id:1486129791176%3Bexp-session; OAS\_SC1=1486129791328; \_\_vrf=1486129791350gUfvWL6rYGybFmH3ZqBoEsypv58VqcPd; TSstop=NA|1486129791436; testcookie=true; s\_fid=7C687C1454C159A8-0E27B74F8356438A; s\_fbcr=1; s\_nr=1486129793025-Repeat; gpv\_sect=homepage; SC\_LNK\_GZ=%5B%5B%5D%5D; gpv\_page=GAZ%2F; s\_cc=true; \_ga=GA1.2.761966839.1426317491; \_sg\_b\_p=%2F; \_ceg.s=oksx35; \_ceg.u=oksx35; \_gat=1; ch\_CBT=tracked; s\_ppvl=GAZ%2F%2C5%2C5%2C30%2C1121%2C302%2C2560%2C1440%2C1%2C; \_cb\_zieHgC\_UlsSBlZQXv; \_chartbeat2=1426317498862.1486129797169.000000000000101.C3a4tLCKWncwC1n\_lKP88pYDbZ1m6; \_cb\_svref=null; dtPC=-; gazzettaNotifications=%7B%22creationDate%22%3A1426317498004%20%22skipModal%22%3Afalse%20%22articles%22%3A%5B%5D%7D; dtLatC=-9876; dtCookie=|\_default|0|Gazzetta|1; \_sg\_b\_v=633%3B3739777%3B1486129793; s\_ppv=GAZ%2F%2C5%2C5%2C336%2C1399%2C332%2C2560%2C1440%2C1%2C; \_chartbeat4=t=BAxExDH0c7TBt6mwQCVC2xB914a\_E=0&EE=0&c=0.81&y=10184&w=332

- L'*header* delle richieste HTTP può avere dimensione non trascurabile perché può contenere: molti *cookie*, molte *header line* per autenticazione, specifica della transazione, etc.
- L'*header* di richieste HTTP consecutive (verso lo stesso server) contiene informazione ridondante



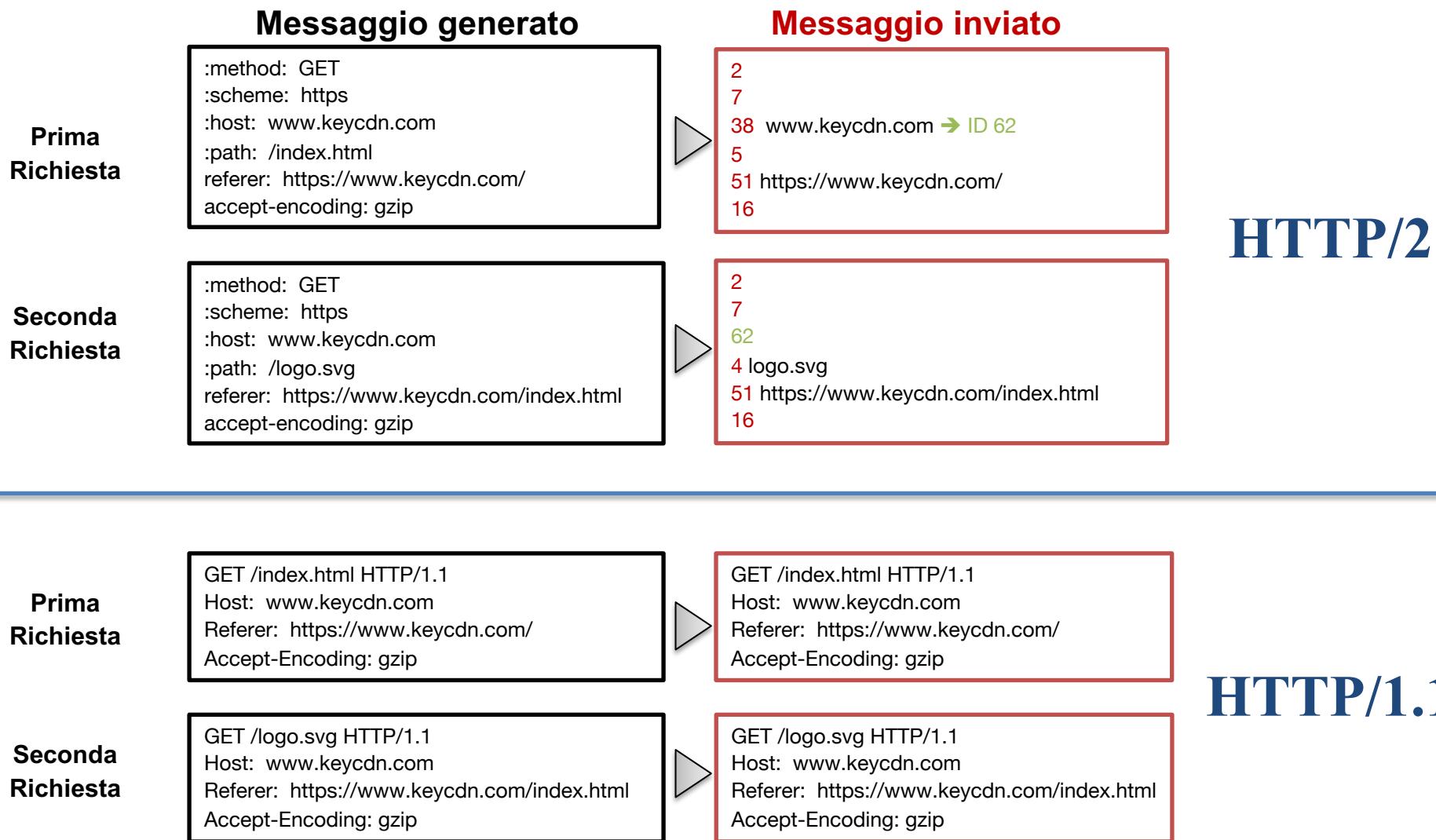
# HTTP/2: HPACK compressione degli header

<sup>1</sup>RFC7541, <https://tools.ietf.org/html/rfc7541>

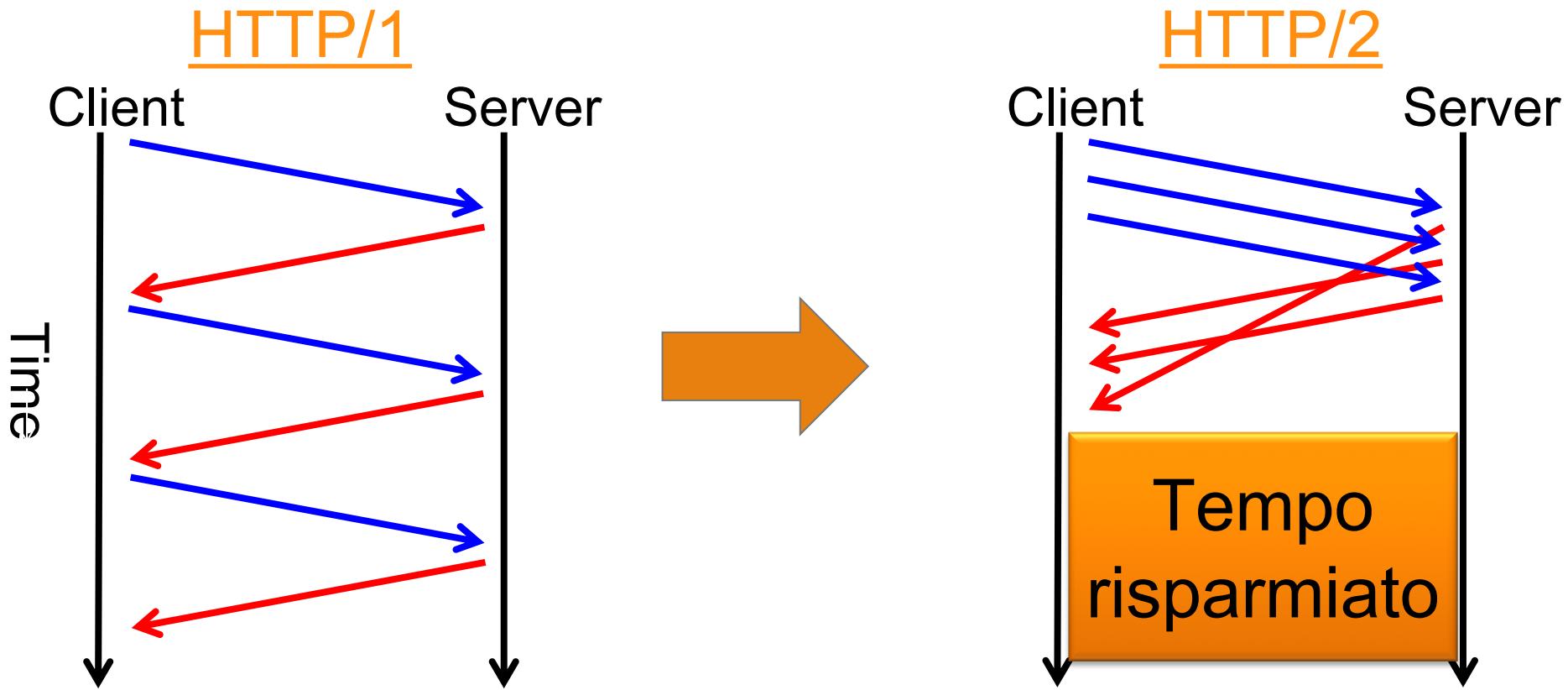
- Codifica di *Huffman*: assegno stringhe binarie a simboli più comuni,
  - es: a-101, c-0, e-111, p-110, t-100, la parola *accept* (6 byte codificata in ASCII) è inviata come *101 0 0 111 110 100* (2 byte)
- *Indexing*: assegno un indice a *header line* più comuni ed invio nei messaggi solo l'indice
- *Codifica differenziale*: l'*header* di richieste successive riporta per esteso solo la differenza con l'*header* delle richieste precedenti



# HTTP/2: HPACK compressione degli header

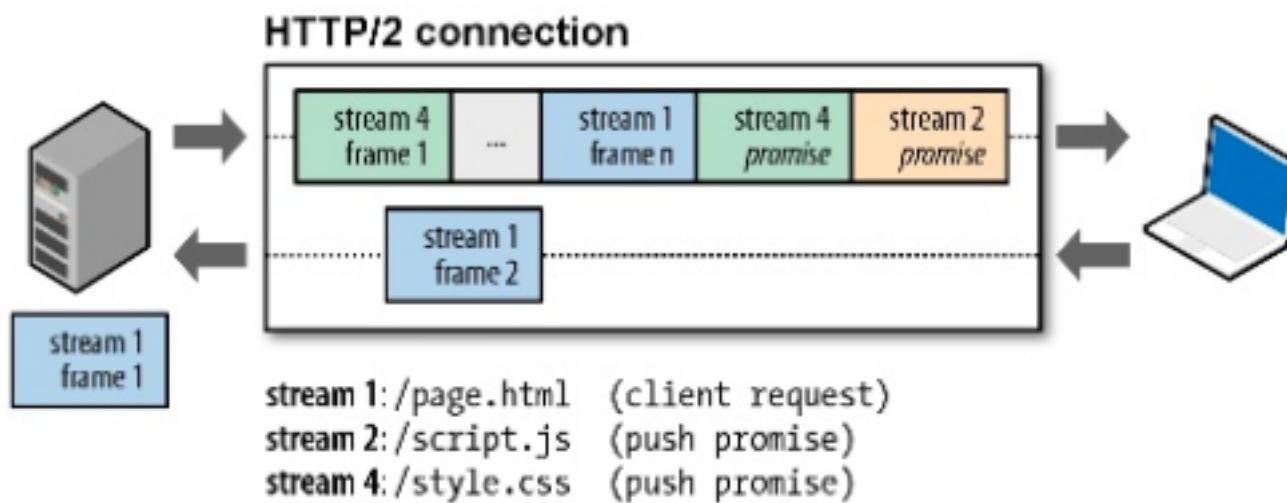


# Multiplazione (1)



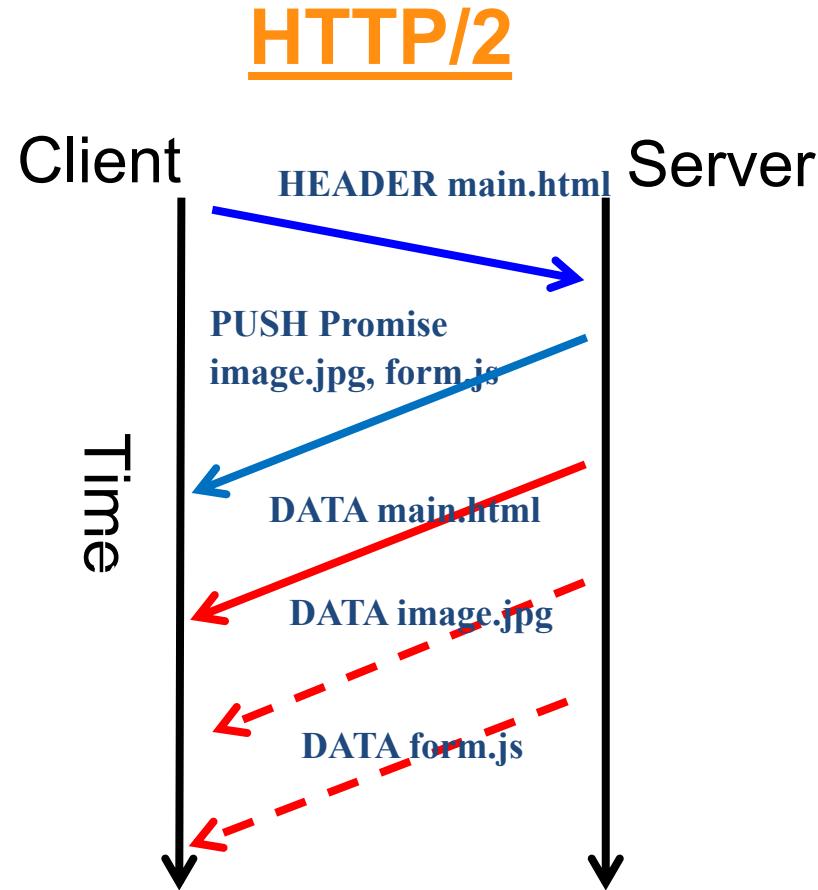
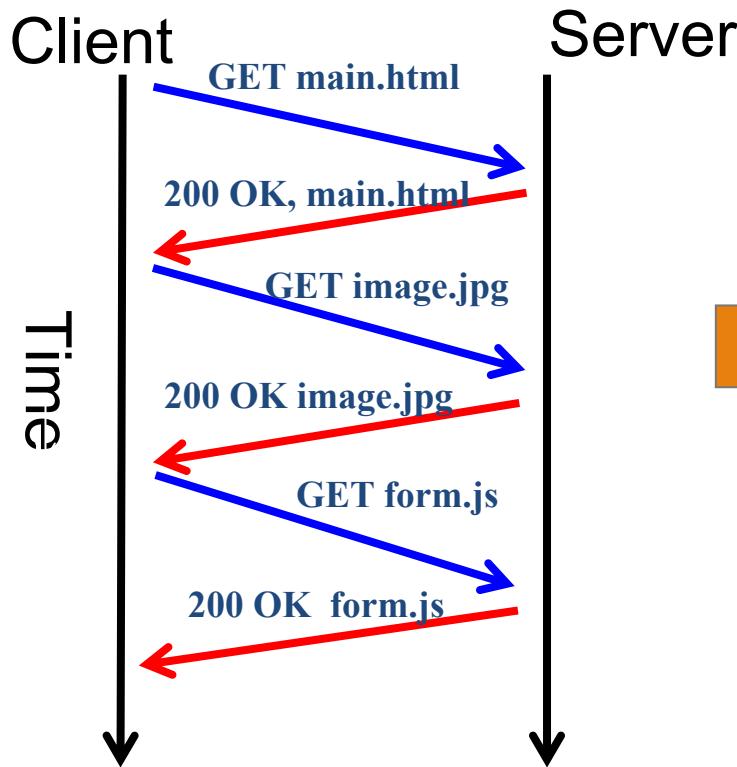
# Multiplazione (2)

- Lo scambio di frame tra *client* e *server* è organizzato in *stream*
- Uno *stream* è una sequenza logica di *frame*
- Ogni *stream* ha una priorità (impostata dal *browser*)



# Server Push

## HTTP/1



- Il *server* può inviare informazione “utile” al *client* senza che il *client* ne faccia esplicita richiesta
- La funzionalità è richiesta dal *client*



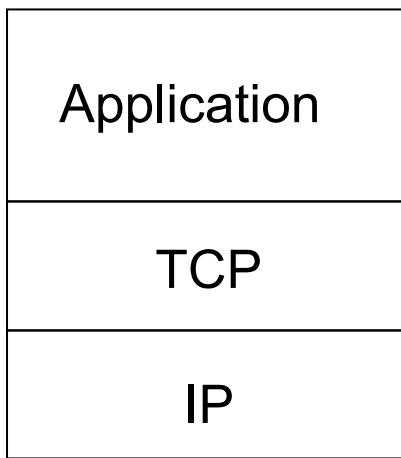
# Rendere sicuro HTTP: HTTPs

- Cosa potrebbe succedere se gli acquisti con Amazon fossero “trasportati” da HTTP?
  - Un malintenzionato potrebbe catturare i messaggi HTTP che contengono i dati della carta di credito usata (nessuna **confidenzialità** dei dati)
  - Un malintenzionato potrebbe modificare i messaggi HTTP che richiedono l’acquisto facendo acquistare merce diversa, più merce, ecc... (nessuna **integrità** dei dati)
  - Un malintenzionato potrebbe fingersi Amazon e truffare l’acquirente e rubargli informazione (nessuna **autenticazione** di server e client)

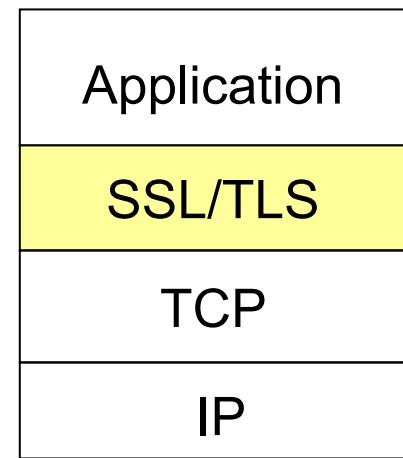


# Soluzioni

- *Secure Socket Layer (SSL) e Transport Layer Security (TLS) aggiungono **confidenzialità, integrità ed autenticazione** alle connessioni TCP*



*senza sicurezza*



*con sicurezza*



# Connessioni SSL/TLS

- **Handshake:**

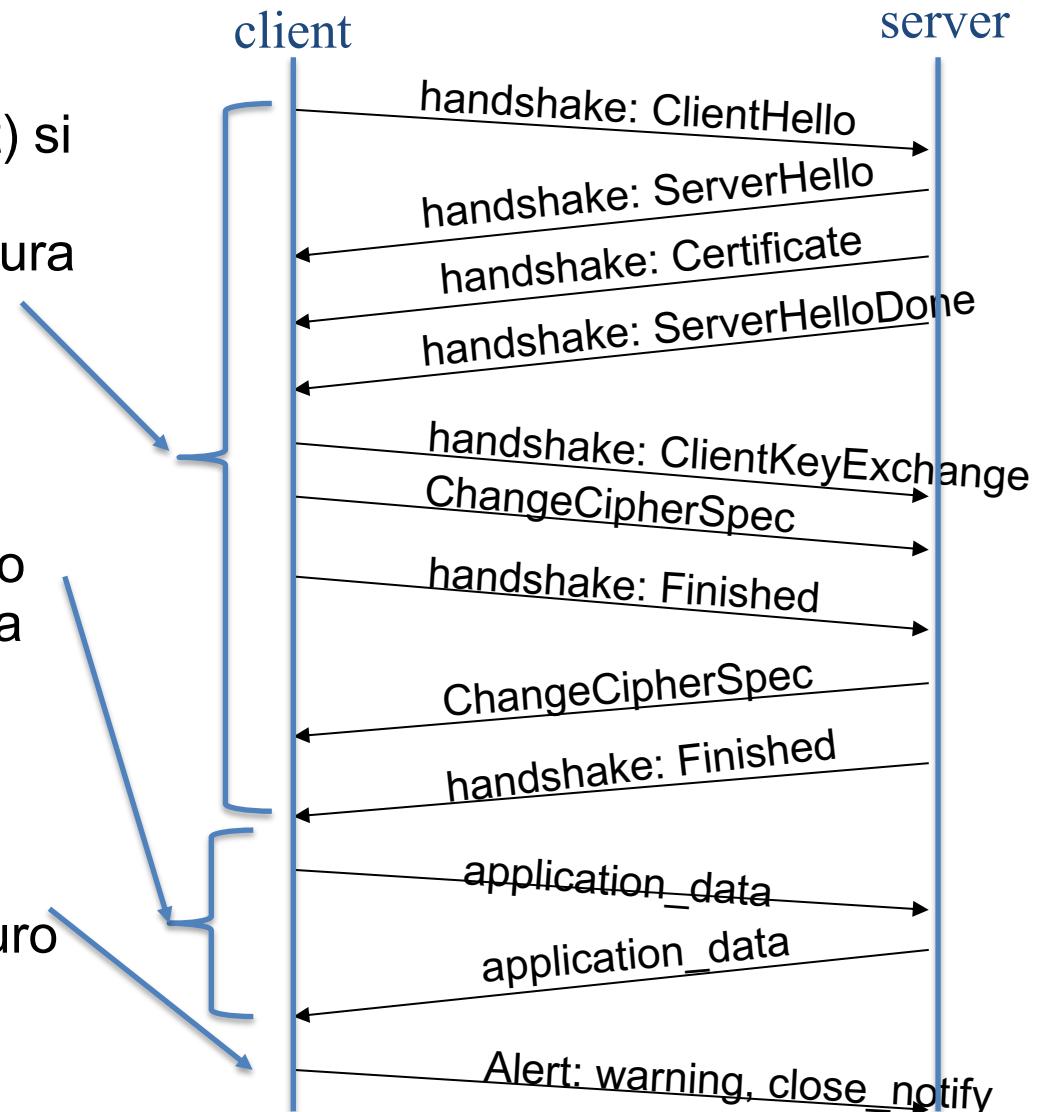
- fase in cui server (e client) si autenticano e si mettono d'accordo sul tipo di cifratura da applicare ai dati

- **Trasferimento dati**

- I dati applicativi sono suddivisi in *record* (PDU) ciascuno dei quali è cifrato con l'algoritmo scelto nella fase precedente

- **Chiusura connessione**

- un messaggio speciale è usato per chiudere la connessione in modo sicuro



# Fase di *Handshake*

- Scambio del *certificato* tra server e client (e viceversa) che certifica l'identità del server (client)
  - il certificato è generato da una *Certification Authority* (CA) e contiene:
    - la *chiave pubblica* dell'entità certificata
    - informazioni aggiuntive (indirizzo IP, nome, etc)
    - firma digitale della CA
- Generazione e scambio delle *chiavi simmetriche* per la cifratura del trasferimento dati
- Lo scambio delle chiavi simmetriche avviane su una connessione a sua volta cifrata con chiavi asimmetriche

**Ne volete sapere di più su concetti “magici” come:  
certificati, chiavi pubbliche, chiavi simmetriche?**

- Laurea Triennale: Sicurezza delle Reti (5CFU) a scelta 3° Anno
- Laurea Magistrale: diversi corsi su crittografia e Computer Security





# Il servizio di posta elettronica

Simple Mail Transfer Protocol (SMTP) RFC 5321

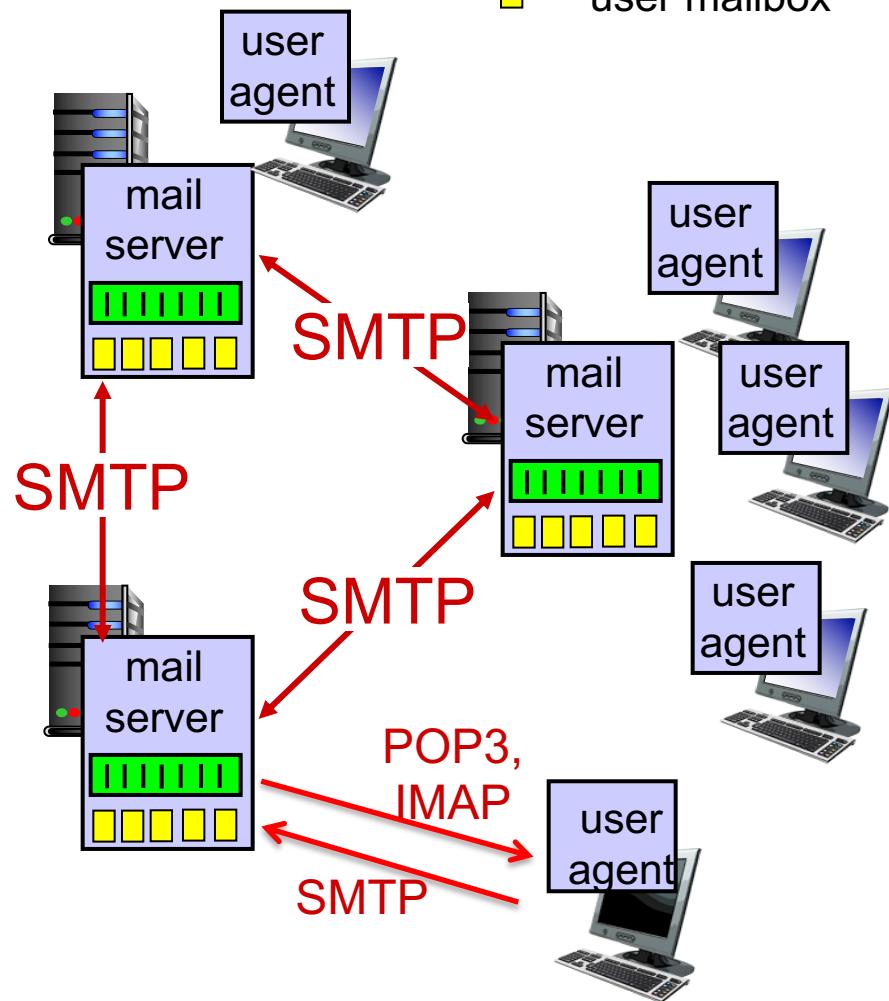
Post Office Protocol (POP3) RFC 1939

Internet Mail Control Protocol (IMAP) RFC 3501

# Il servizio di E-mail

||||| outgoing message queue  
█ user mailbox

- **Client d'utente aka User Agent (OutLook, Thunderbird, etc.)**
- **Mail Server**
- **Simple Mail Transfer Protocol SMTP:** per trasferire email dal client d'utente fino al mail server del destinatario
- **Protocolli di accesso ai mail server:** per “scaricare” email dal proprio mail server (POP3, IMAP)

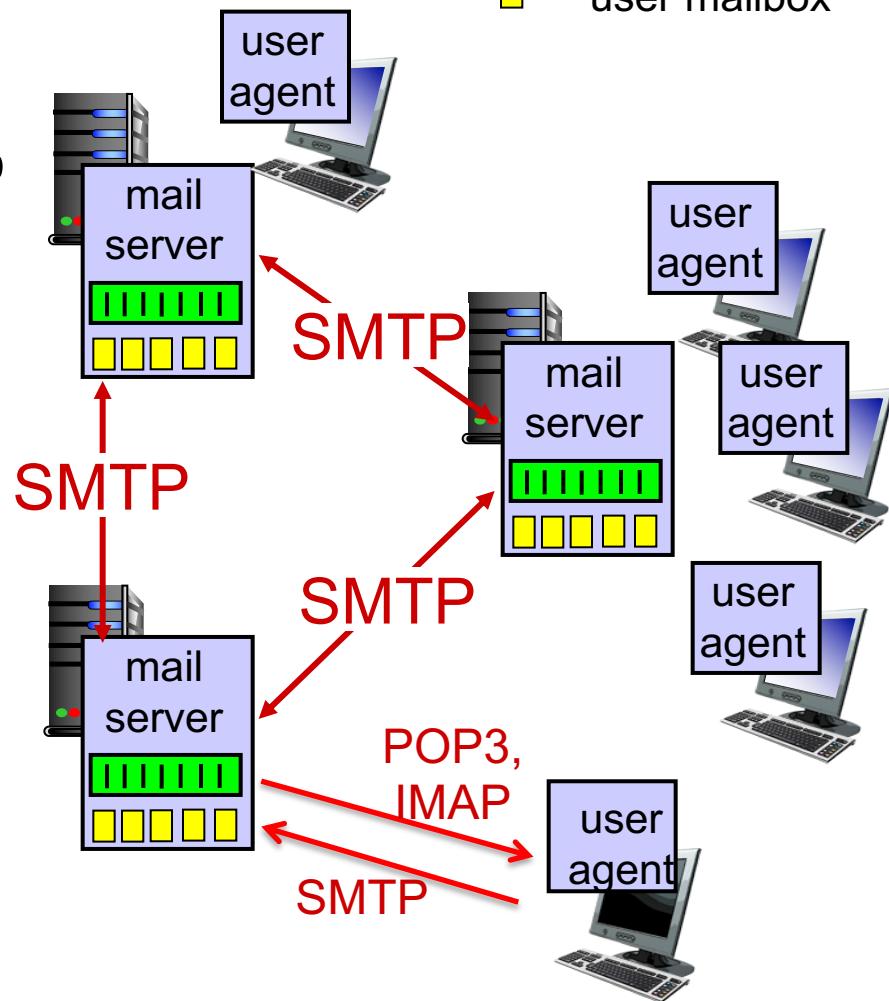


# I Mail Server

 outgoing message queue

 user mailbox

- I mail server contengono per ogni client controllato:
  - una coda di email in ingresso (**mailbox**)
  - una **coda di email in uscita**
- I mail server
  - Ricevono le mail in uscita da tutti i client d'utente che “controllano”
  - Ricevono da altri mail server tutte le mail destinate ai client d'utente controllati
- I mail server “parlano”
  - **SMTP** con altri mail server e con i client d'utente in uplink
  - **POP3/IMAP** con i client d'utente in downlink



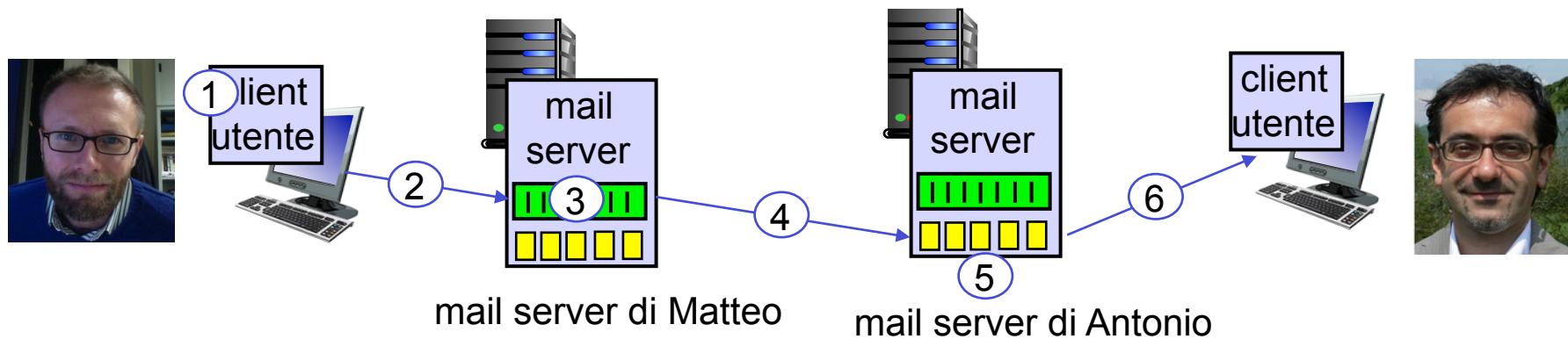
# SMTP

- E' un protocollo applicativo *client-server*
- quando un *mail server* riceve un messaggio da un client d'utente
  - mette il messaggio in una coda
  - apre una connessione TCP con la porta 25 del *mail server* del destinatario
  - trasferisce il messaggio
  - chiude la connessione TCP
- L'interazione tra *client* SMTP e *server* SMTP è di tipo comando/risposta
- Comandi e risposte sono testuali
- Richiede che anche il corpo dei messaggi sia ASCII
  - i documenti binari devono essere convertiti in ASCII 7-bit



# Esempio di trasferimento SMTP

1. Matteo compone una *email* destinata ad Antonio [antonio@miomailserver.com](mailto:antonio@miomailserver.com)
2. Il *client d'utente* di Matteo invia la *mail* al proprio *mail server*
3. Il *mail server* di Matteo si comporta come *client SMTP* ed apre una connessione TCP (porta 25) con il *mail server* di Antonio
4. Il *client SMTP* (*mail server* di Matteo) invia la *email* sulla connessione TCP
5. Il *mail server* di Antonio memorizza la *mail* nella *mailbox* di Antonio
6. Antonio (in modo asincrono) usa il proprio *client d'utente* per leggere la *mail*



# Colloquio tra client e server SMTP

apertura

```
S: 220 antoniomailserver.com  
C: HELO matteomailserver.com  
S: 250 Hello matteomailserver.com, pleased to meet you
```

```
C: MAIL FROM: <matteo@matteomailserver.com>
```

```
S: 250 matteo@matteomailserver.com... Sender ok
```

```
C: RCPT TO: <antonio@antoniomailserver.com>
```

invio

```
S: 250 antonio@antoniomailserver.com ... Recipient ok
```

```
C: DATA
```

```
S: 354 Enter mail, end with "." on a line by itself
```

```
C: Oggi corri al Giuriati?
```

```
C: .
```

```
S: 250 Message accepted for delivery
```

chiusura

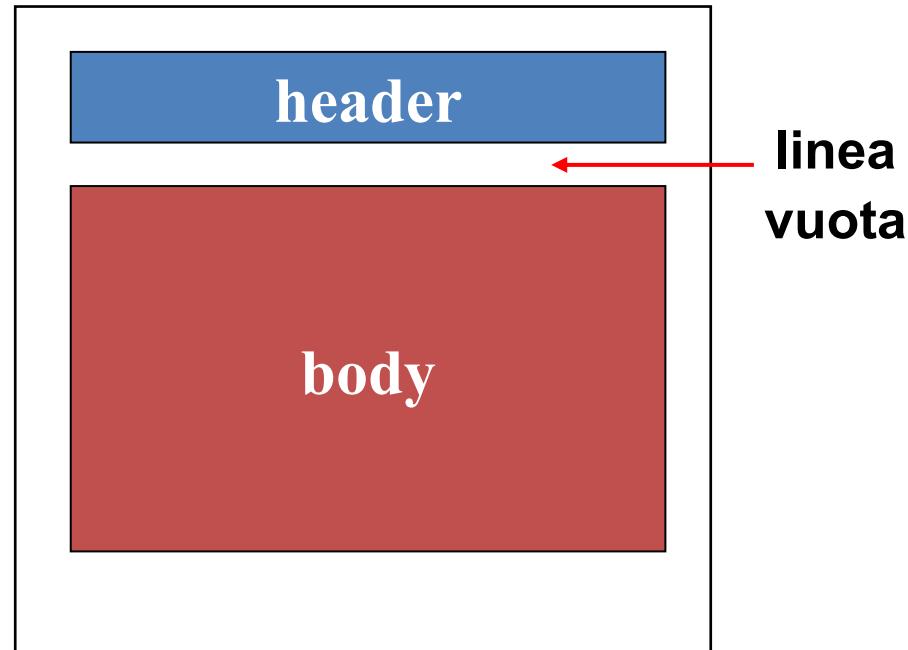
```
C: QUIT
```

```
S: 221 antoniomailserver.com closing connection
```



# Il formato delle email (RFC 822)

- Il formato dei messaggi inviati (tutto ciò che segue il comando SMTP DATA) è specificato
- *Header:*
  - *To:*
  - *From:*
  - *Subject:*
- Body: il contenuto dell'email (deve essere ASCII!)



# Multipurpose Internet Mail Extensions (MIME)

## RFC 2045- 2046

- Estende il formato dei messaggi email (RFC 822) per supportare contenuti multimediali (non ASCII 7-bit)
- Definisce *header* per specificare il tipo di contenuto (Content-Type) ed il tipo di codifica (base64, quoted printable)

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
-----
.....base64 encoded data
.
```



# Multipurpose Internet Mail Extensions (MIME)

## RFC 2045- 2046

- MIME consente anche il trasferimento di più oggetti come parti di uno stesso messaggio:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe with commentary
MIME-Version: 1.0
Content-Type: multipart/mixed; Boundary=StartOfNextPart
--StartOfNextPart
Dear Bob,
Please find a picture of an absolutely scrumptious crepe.

--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data ......

--StartOfNextPart
Let me know if you would like the recipe.
.
```



# Sessione d'esempio

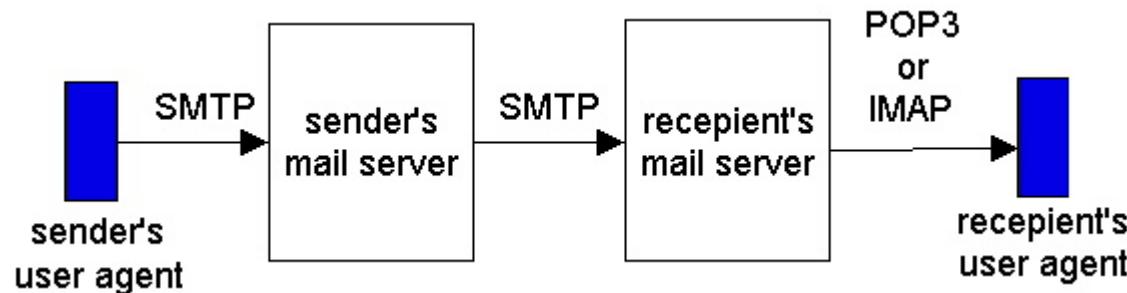
```
albertux@antlab101:~$ telnet localhost 25
220 localhost ESMTP Postfix (Debian/GNU)
HELO studente.it
250 localhost
MAIL FROM:<matr666999@studente.it>
250 2.1.0 Ok
RCPT TO:<user@labfir.lan>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Subject: Verifica SMTP
From: matr666999@studente.it
To: professore@labfir.it
Buongiorno
Questa mail serve a dimostrare che ho imparato SMTP.

.
250 2.0.0 Ok: queued as F41CC9F6EE
QUIT
221 2.0.0 Bye
```



# Protocolli di accesso al mailbox

- Diversi protocolli sono stati sviluppati per il colloquio tra *user agent* e *server* in fase di lettura dei messaggi presenti nel *mailbox*
  - POP3 (*Post Office Protocol versione 3, RFC 1939*): download dei messaggi
  - IMAP (*Internet Mail Access Protocol, RFC 1730*): download dei messaggi, gestione della *mailbox* sul *mail server*
  - HTTP



# POP3

## Fase di autorizzazione

- Comandi del client:
  - **user**: username
  - **pass**: password
- Risposte del server:
  - **+OK**
  - **-ERR**

## Fase di transazione, client:

- **list**: elenca numero mess.
- **retr**: recupera messaggio
- **dele**: cancella messaggio
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```



# Comandi

Login in chiaro:

**USER <username>**  
**PASS <password>**

Risposte del server:

-ERR  
+OK

Operazioni comuni:

● **STAT**

info sullo stato mbox

● **LIST**

elenca il # messaggi

● **RETR *n***

leggi messaggio *n*

● **DELE *n***

cancella messaggio *n*

● **RSET**

annulla cancellazioni

● **QUIT**

esce

● **CAPA**

mostra le capabilities del server



# Sessione d'esempio

```
albertux@antlab101:~$ telnet localhost 110
+OK Dovecot ready.
USER user
+OK
PASS labfir
+OK Logged in.
LIST
+OK 1 messages:
1 598

RETR 1
+OK 598 octets
Return-Path: <matr666999@studente.it>
X-Original-To: userlab@localhost
Delivered-To: userlab@localhost
Received: from studente.it (ip254 [192.168.0.254])
    by localhost (Postfix) with SMTP id F41CC9F6EE
    for <labrci001@labrci.lan>; Wed, 26 Mar 2008 13:09:43 +0100 (CET)
Message-Id: <20080326121008.F41CC9F6EE@localhost>
Date: Wed, 26 Mar 2008 13:09:43 +0100 (CET)
From: matr666999@studente.it
To: undisclosed-recipients:;

Subject: Verifica SMTP
From: matr666999@studente.it
To: professore@labrci.it
Buongiorno
Questa mail serve a dimostrare che ho imparato SMTP.

DELE 1
+OK Marked to be deleted.
QUIT
+OK Logging out, messages deleted.
```



# Esercizi

- Vedere video su *YouTube* su SMTP e POP3

<https://youtu.be/v4Zpywa8ag0>

- Inviare una mail da un mittente falso  
han.solo@MFalcon.glx al proprio indirizzo di posta elettronica

**Oggetto:** *Fake Rules!*

**Corpo del messaggio:**

*Il corso di Fondamenti di Internet e Reti è fantastico!*





# Risoluzione di nomi simbolici

DNS

# Domain Name System (DNS)

- Gli indirizzi IP (32 bit) sono poco adatti ad essere usati dagli applicativi
- E' più comodo utilizzare indirizzi simbolici:
  - E' meglio www.google.com o 74.125.206.99?
- Occorre una mappatura fra indirizzi IP (usati dalle macchine di rete) ed i nomi simbolici (usati dagli esseri umani)

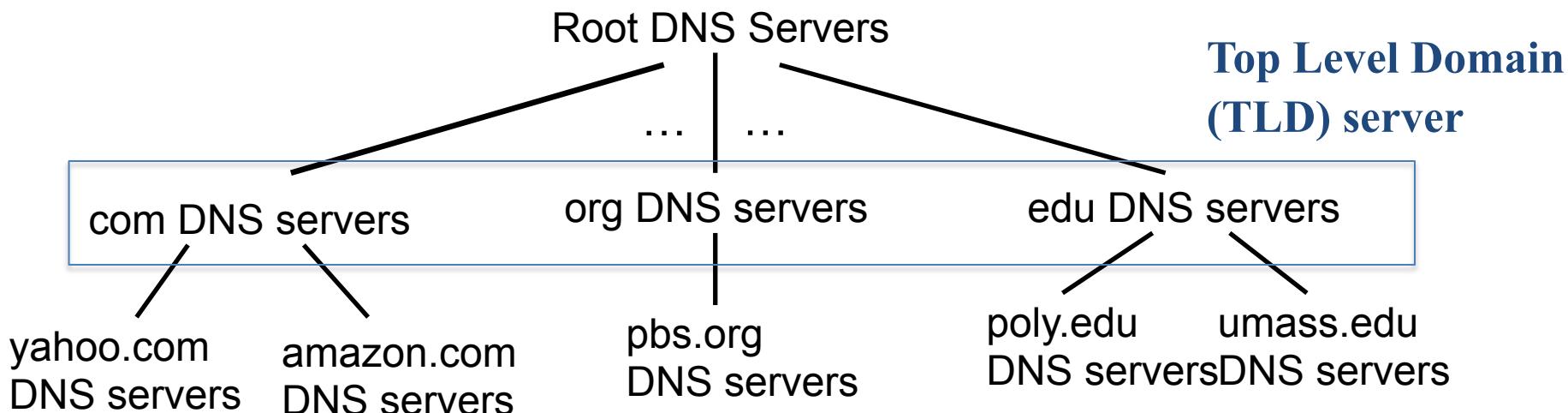


# Domain Name System (DNS)

- Ingredienti
  - Database distribuito costituito da molti *name servers* con organizzazione gerarchica
  - Protocollo applicativo basato su UDP tra *name server* e *host* per **risolvere** nomi simbolici (tradurre nomi simbolici in indirizzi IP)
- Servizi aggiuntivi
  - *host aliasing*
  - *Mail server aliasing*
  - *Load distribution*



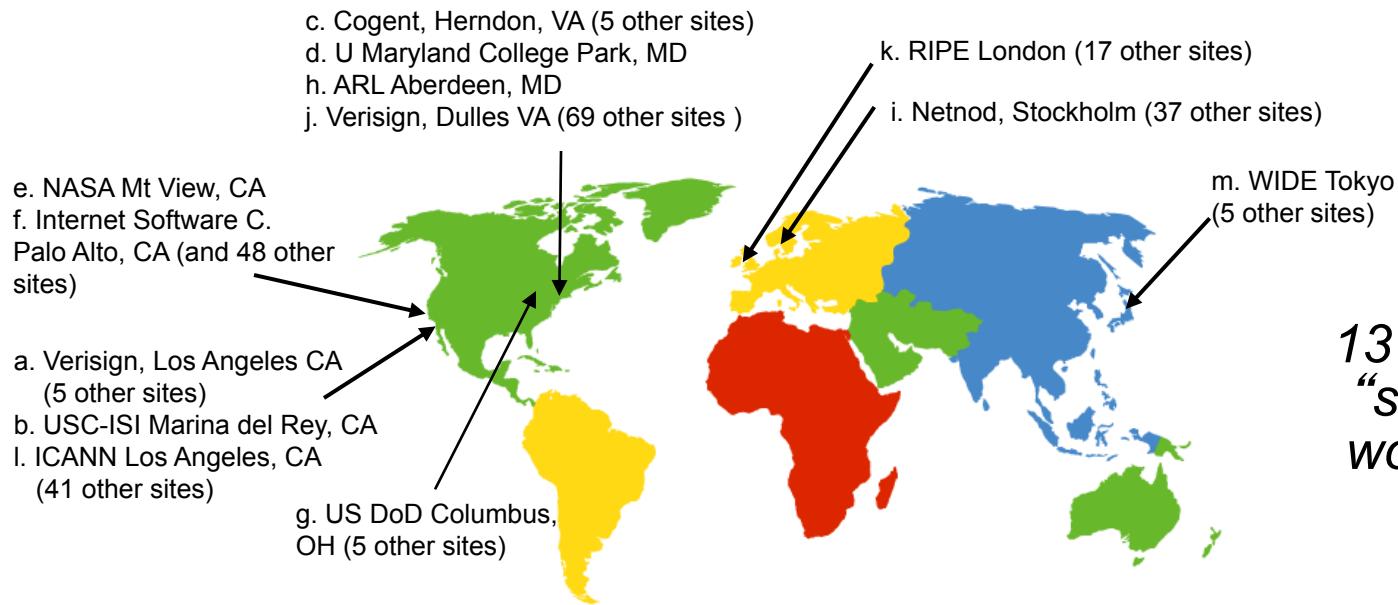
# Database distribuito e gerarchico



- Ogni livello nella gerarchia ha diversa “profondità” di informazione
- Esempio: un utente vuole l’IP di **www.google.com**
  - *Root name server sanno come “trovare” i name server che gestiscono i domini .com*
  - *I name server .com sanno come trovare il name server che gestisce il dominio google.com*
  - *I name server google.com sanno risolvere il nome simbolico www.google.com*



# Root NS



*13 root name  
“servers”  
worldwide*



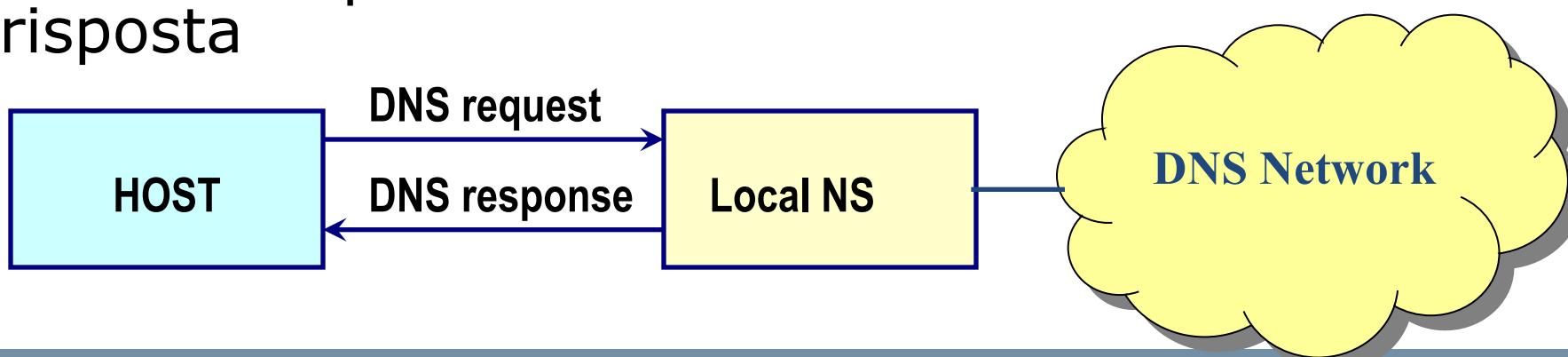
# Altri tipi di NS

- *Local Name Servers*
  - Ogni ISP (residenziale, università, compagnia) ha un NS locale
  - Direttamente collegati agli *host*
  - Tutte le volte che un *host* deve risolvere un indirizzo simbolico contatta il Local Name Server
  - Il *Local Name Server* (eventualmente) contatta i Root Name Server nella gerarchia
- *Authoritative Name Servers*
  - NS “responsabile” di un particolare *hostname*

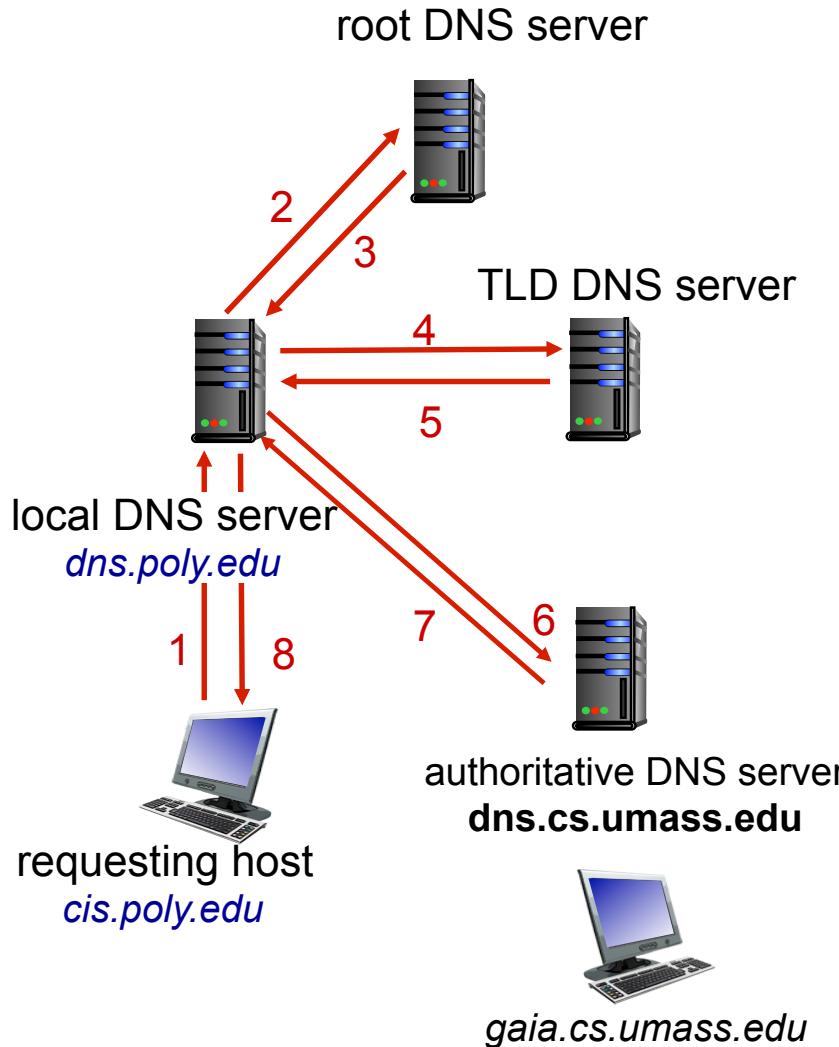


# Come ottenere un mappaggio

- Ogni *host* ha configurato l'indirizzo del LNS
- Le applicazioni che richiedono un mappaggio (browser, ftp, etc.) usano le funzioni del DNS
- Una richiesta viene inviata al server DNS usando UDP come trasporto
- Il server reperisce l'informazione e restituisce la risposta



# Esempio di risoluzione di un nome simbolico: modalità iterativa

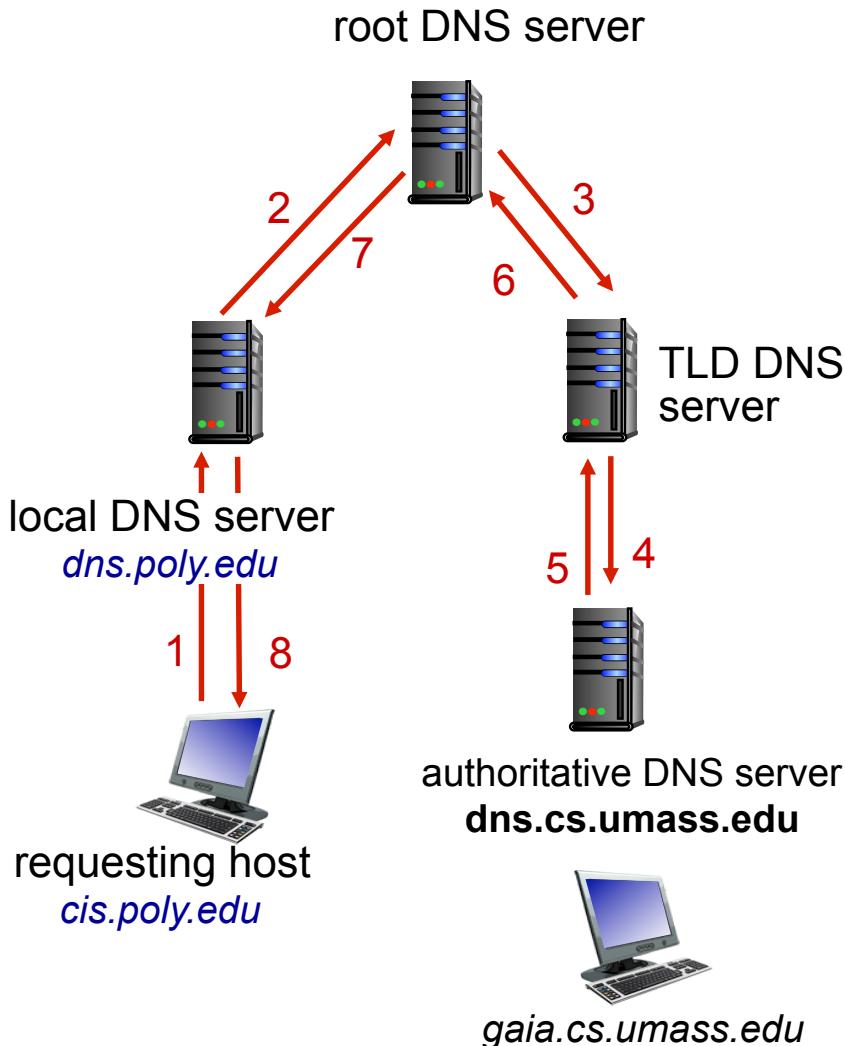


Un *host* presso `cis.poly.edu` vuole risolvere l'indirizzo simbolico `gaia.cs.umass.edu`

1. Il *client* DNS sull'*host* contatta il LNS
2. Il LNS contatta il Root NS
3. Il Root NS segnala al LNS il TLD server responsabile del dominio `.edu`
4. Il LNS contatta il TLD server responsabile del dominio `.edu`
5. Il TLD server segnala al LNS il server autoritativo per il nome simbolico `gaia.cs.umass.edu`
6. il LNS contatta il *name server* autoritativo
7. il server autoritativo segnala al LNS l'indirizzo IP che corrisponde a `gaia.cs.umass.edu`



# Esempio di risoluzione di un nome simbolico: modalità ricorsiva



Un host presso *cis.poly.edu* vuole risolvere l'indirizzo simbolico *gaia.cs.umass.edu*

1. Il *client DNS* sull'*host* contatta il LNS
2. Il LNS contatta il Root NS
3. Il Root NS contatta il TLD server responsabile del dominio .edu
4. Il TLD contatta il *server autoritativo* per il nome simbolico *gaia.cs.umass.edu*
5. I *server autoritativo* segnala al TLD server l'indirizzo IP che corrisponde a *gaia.cs.umass.edu*
6. 7. 8. l'informazione segue il percorso inverso



# Caching

- Un *server*, dopo aver reperito un'informazione su cui non è *authoritative* può memorizzarla temporaneamente
- All'arrivo di una nuova richiesta può fornire l'informazione senza risalire sino al *server authoritative*
- Il TTL è deciso dal *server authoritative* ed è un indice di quanto stabile nel tempo è l'informazione relativa
- I TLD *server* sono generalmente “memorizzati” nei *Local Name Servers*
- I *server non-authoritative* usano il TTL per decidere un time-out



# Informazioni memorizzate

- | Resource Record  | Name, Value, Type, TTL |
|--|------------------------|
| • Type   |                        |
| – A: <i>Name</i> è il nome di un <i>host</i> e <i>Value</i> è il suo indirizzo IP<br>(morgana.elet.polimi.it, 131.175.21.1, A, TTL)  |                        |
| – NS: <i>Name</i> è un <i>domain</i> e <i>Value</i> è il nome di un <i>server</i> che può ottenere le informazioni relative<br>(elet.polimi.it, morgana.elet.polimi.it, NS, TTL) |                        |
| – CNAME: <i>Name</i> è un nome alternativo (alias) per un <i>host</i> il cui nome canonico è in <i>Value</i><br>(www.polimi.it, zephyro.rett.polimi.it, CNAME, TTL)              |                        |
| – MX: <i>Name</i> è dominio di <i>mail</i> o un alias di <i>mail</i> e <i>Value</i> è il nome del <i>mail server</i><br>(elet.polimi.it, mailserver.elet.polimi.it, MX, TTL)     |                        |



# Formato dei messaggi DNS

- identification: identificativo coppia richiesta/risposta
- flag: richiesta/risposta, authoritative/non auth., iterative/recursive
- number of: relativo al numero di campi nelle sez. successive
- questions: nome richiesto e tipo (di solito A o MX)
- answers: resource records completi forniti in risposta
- authority: contiene altri record forniti da altri server
- additional infor.: informazione addizionale, ad es. il record con l'IP ADDR. per il MX fornito in answers

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

12 bytes



# Come aggiungere un dominio alla “rete” DNS

- una nuova startup *I-Like-Networking* vuole registrare il dominio *I-Like-Networking.com* (supponiamo che il dominio sia disponibile)
- *I-Like-Networking* registra il dominio presso uno dei **DNS Registrars**
  - *I-Like-Networking* deve fornire al **DNS registrar** i nomi simbolici ed i relativi indirizzi IP dei name server autoritativi
  - Il DNS registrar inserisce due RR nel TLD server .com
    - I-Like-Networking, dns1.I-Like-Networking.com, NS
    - dns1.I-Like-Networking.com, 212.212.212.1, A
  - Il **DNS registrar** eventualmente scrive un record di tipo MX per *I-Like-Networking.com*



# Semplici esperimenti con DNS - nslookup

- Usiamo il comando nslookup che consente di inviare richieste DNS a server specificati
- Vediamo come funziona  
`man nslookup`
- Risolviamo un nome simbolico  
`nslookup www.antlab.polimi.it`
- Troviamo i name server autoritativi per un certo dominio  
`nslookup -type=NS polimi.it`

**Compiti a casa:** provate a ottenere una risposta autoritativa per il nome simbolico `www.google.com`



# Semplici esperimenti con DNS - dig

- Il comando `dig` (simile a `nslookup`) fornisce più dettagli sui messaggi del protocollo DNS
- Proviamo una semplice query

```
dig www.polimi.it
```



# Semplici esperimenti con DNS - dig

```
[wMacBook-Pro-di-Matteo:~ teo1$ dig www.polimi.it
```

```
; <>> Dig 9.8.3-P1 <>> www.polimi.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 10838
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 2
```

```
;; QUESTION SECTION:
www.polimi.it.          IN      A
```

```
;; ANSWER SECTION:
www.polimi.it.      134     IN      A      131.175.187.72
```

```
;; AUTHORITY SECTION:
polimi.it.           1152    IN      NS      ns.polimi.it.
polimi.it.           1152    IN      NS      dns.cineca.it.
polimi.it.           1152    IN      NS      ns2.polimi.it.
```

```
;; ADDITIONAL SECTION:
ns2.polimi.it.       2488    IN      A       131.175.12.2
ns.polimi.it.        1546    IN      A       131.175.12.1
```

```
;; Query time: 3 msec
;; SERVER: 10.248.17.11#53(10.248.17.11)
;; WHEN: Wed Jan 20 10:30:56 2016
;; MSG SIZE  rcvd: 139
```

```
MacBook-Pro-di-Matteo:~ teo1$
```

Header del messaggio DNS

Descrizione della richiesta

Risposta

Server autoritativi per il dominio richiesto

Informazioni aggiuntive

Informazioni di performance sulla richiesta



# Semplici esperimenti con DNS - dig

- Se si vuole solo l'elenco dei record NS  
`dig -t NS polimi.it +noall +answer`
- Se si vuole solo l'elenco dei record MX  
`dig -t MX polimi.it +noall +answer`
- Se si vuole l'elenco di tutti i record disponibili  
`dig -t ANY polimi.it +noall +answer`
- dig consente anche di analizzare la sequenza di richieste DNS per ogni query  
`dig -t A polimi.it +noall +answer +trace`

Da soli: leggere il manuale di `dig`, cambiare la modalità di risoluzione di un nome simbolico e verificare il comportamento di `dig` con l'opzione `+trace`



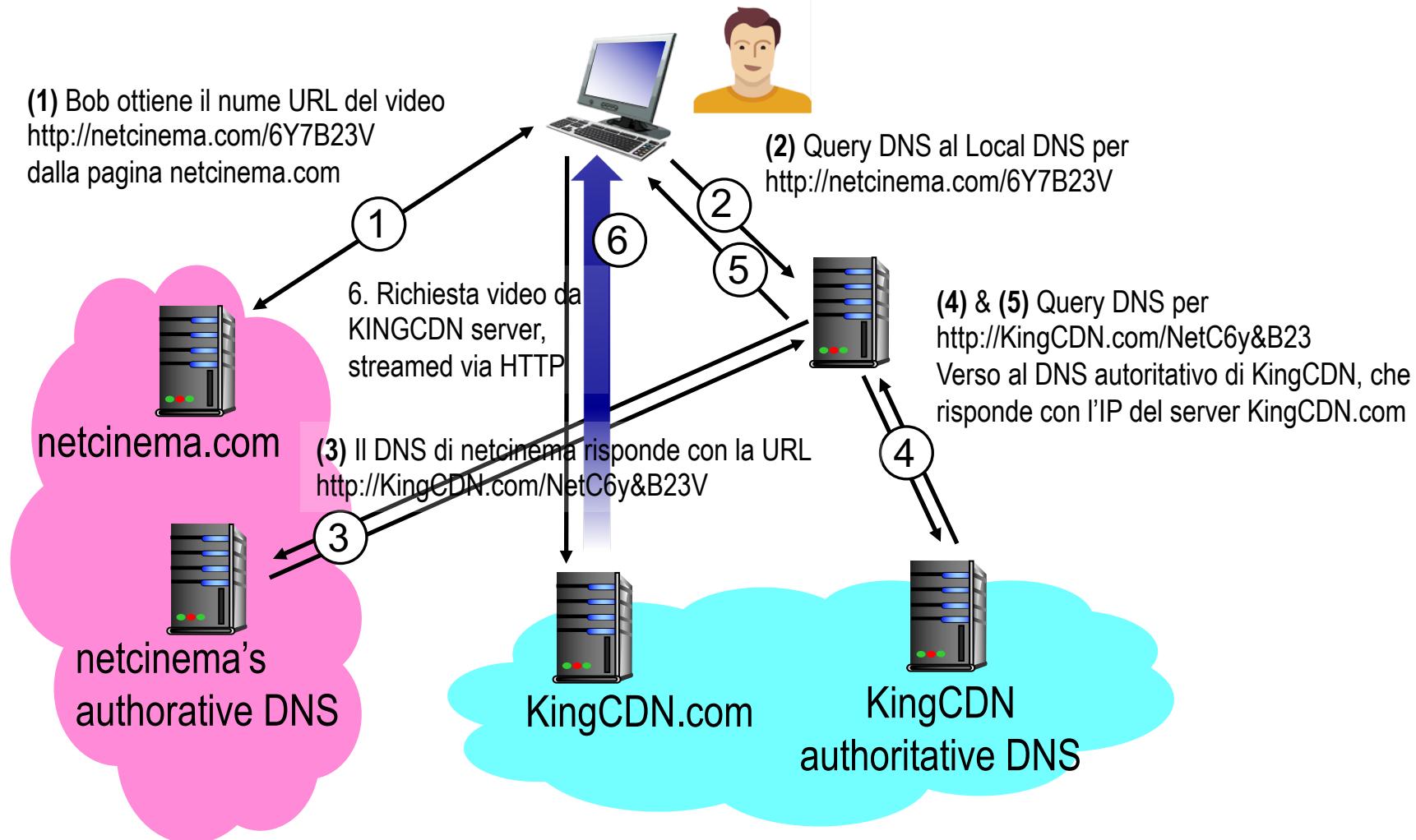
# Content Distribution Networks (CDNs)

- *Problema:*
  - come distribuire in maniera efficiente moltissimi contenuti contemporaneamente a moltissimi utenti lontanissimi gli uni dagli altri
- *Soluzione:*
  - creare una rete di server geograficamente distribuita che “ospita” copie dei contenuti richiesti (simile ad una “mega Cache” distribuita)
  - *la rete di server (CDN) può essere di proprietà di chi offre il servizio (Google, Netflix, Facebook) o di terze parti (Akamai, Limelight, KCDN)*



# CDN: Esempio di accesso ai contenuti

L'azienda NetCinema si appoggia ad una CDN gestita da KingCDN  
Bob (client) richiede un video <http://netcinema.com/6Y7B23V>  
Il video si trova in CDN a <http://KingCDN.com/NetC6y&B23V>



# Scelta del **miglio Server**

- **Più vicino**: scegli il server più vicino “geograficamente” al *client*
- **Percorso più corto**: scegli il *server* con il percorso più corto (numero di hop più piccolo) verso il *client*
- **Lascia decidere all’utente**: fornisce all’utente una lista di *server* possibili e l’utente sceglie il migliore (*Netflix*)





# Applicazioni Peer-to-Peer

File sharing, architettura, search

# P2P file sharing

- Gli utenti utilizzano il software P2P sul proprio PC
  - Si collegano in modo intermittente a Internet prendendo indirizzi IP diversi ogni volta
  - Se un utente cerca un file l'applicazione trova altri utenti che lo hanno
  - L'utente sceglie da chi scaricarlo
  - Il file è scaricato usando un protocollo come HTTP
  - Altri utenti potranno (in seguito o in contemporanea) scaricare il file dall'utente
  - L'applicazione P2P è sia client che server.
- Elevata scalabilità!**



# P2P: directory centralizzata

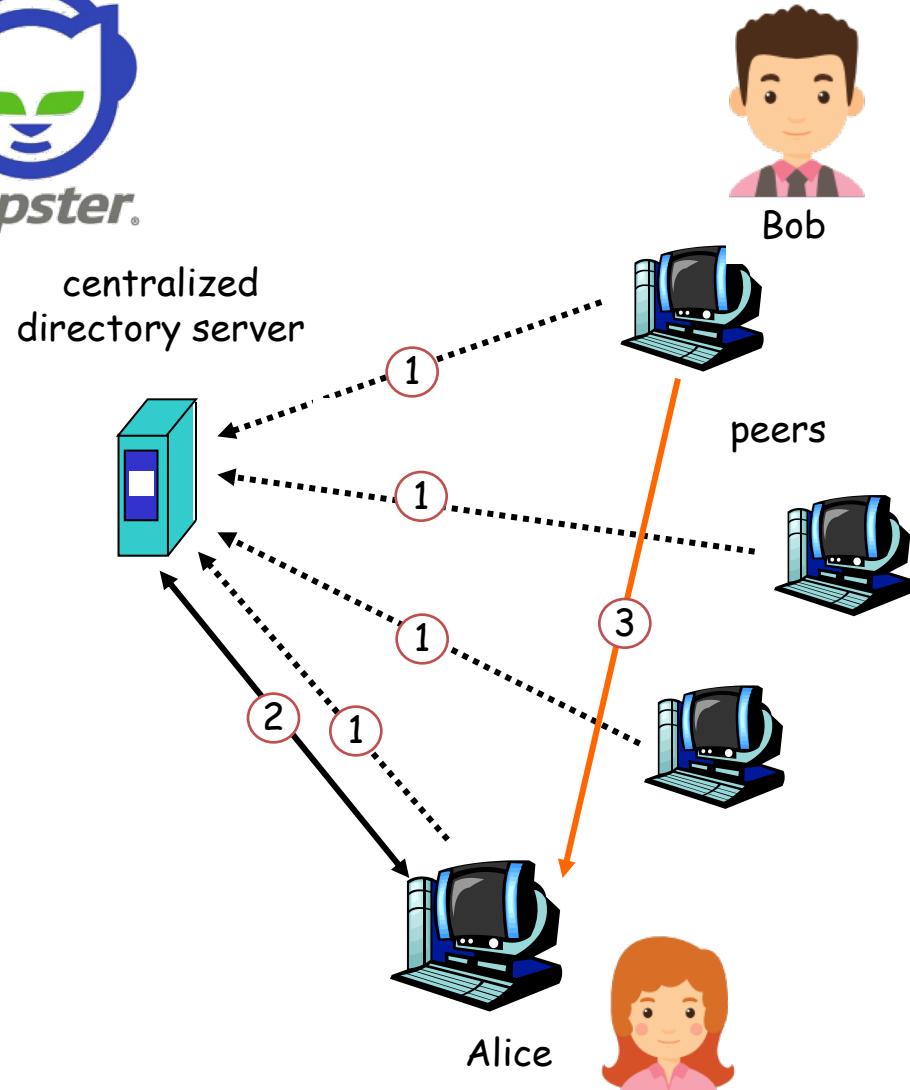
Meccanismo di  
“Napster”

1) Quando i *peer* si connettono, informano il server centrale:

- Indirizzo IP
- File condivisi

2) Il *peer* interroga il server centrale per uno specifico file

3) Il file viene scaricato direttamente



# P2P: directory centralizzata

- Problemi dell'architettura centralizzata
  - Se il *server* si rompe il sistema si blocca
  - Il *server* è un collo di bottiglia per il sistema
  - Chi gestisce il *server* può essere accusato di infrangere le regole sul *copyright*

Il trasferimento file è distribuito, ma la ricerca dei contenuti è fortemente centralizzata



# P2P: completamente distribuita

## Meccanismo di *Gnutella*

- Nessun *server* centrale
- Protocollo di pubblico dominio
- Molti *software* diversi basati sullo stesso protocollo



**GNUTELLA**  
search and find multiple files



## Basato su una rete (grafo) di *overlay*

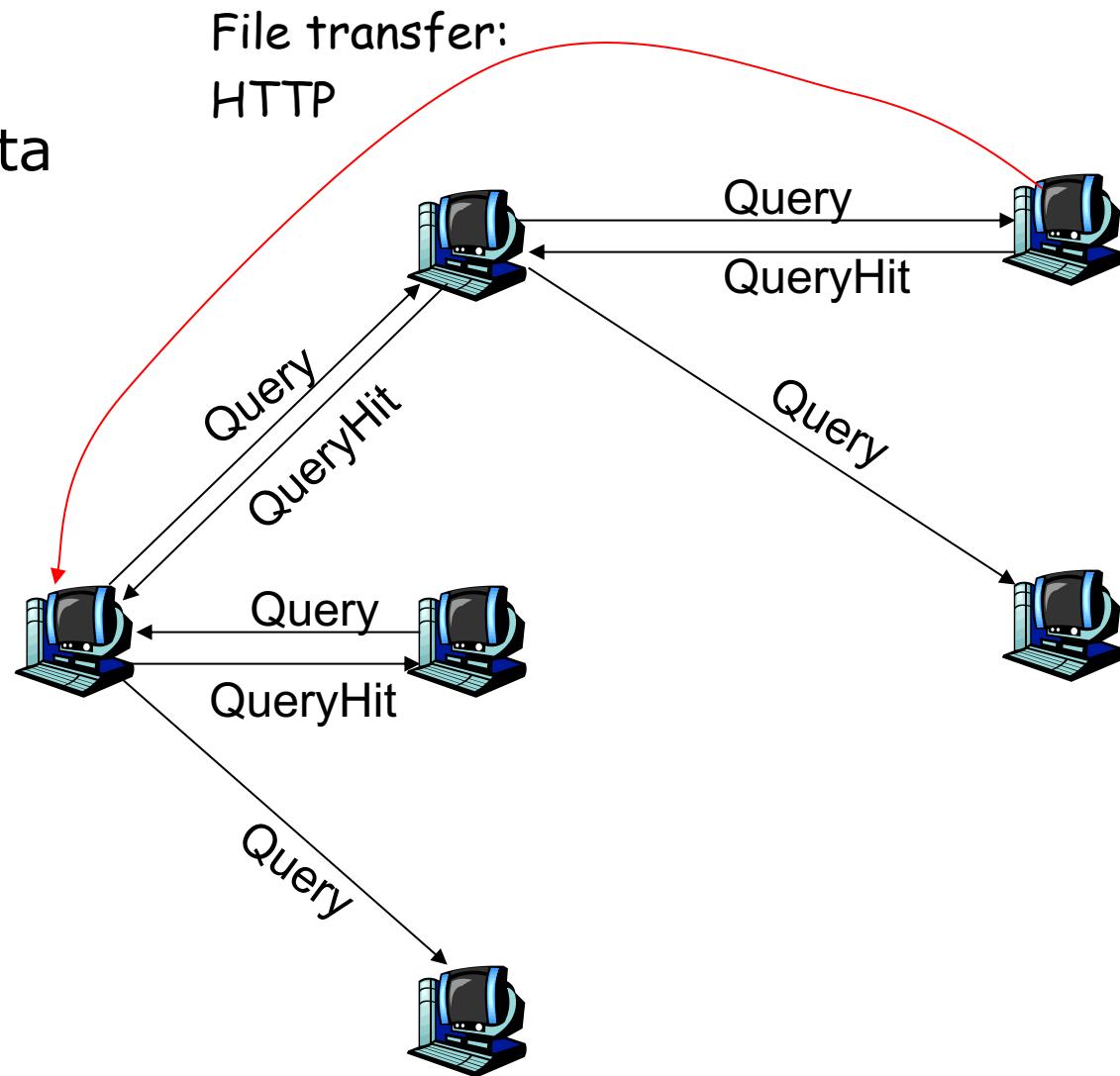
- I *peer* si attivano e si collegano ad un numero (<10) di altri vicini
- La ricerca dei vicini è distribuita
- I vicini nella rete *overlay* possono essere fisicamente distanti



# P2P: completamente distribuita

## Ricerca di un file

- I messaggi di richiesta vengono diffusi sulla rete di *overlay*
- I *peer* inoltrano le richieste fino a una certa distanza
- Le risposte vengono inviate sul cammino opposto



# P2P: completamente distribuita

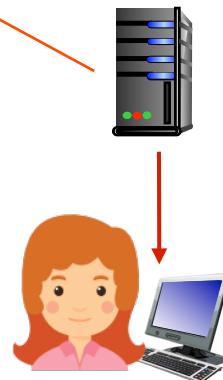
- Accesso alla rete
  - Per iniziare il processo di accesso il *peer* X deve trovare almeno un altro *peer*; la ricerca si basa su liste note
  - X scandisce la lista fino a che un *peer* Y risponde
  - X invia un messaggio di Ping a Y; Y inoltra il Ping nella rete di *overlay*.
  - Tutti i *peer* che ricevono il *Ping* rispondono a X con un messaggio di *Pong*
  - X riceve molti messaggi di *Pong* e può scegliere a chi connettersi aumentando il numero dei suoi vicini nella rete di *overlay*



# BitTorrent

- I file sono divisi in *chunk* di 256kbyte

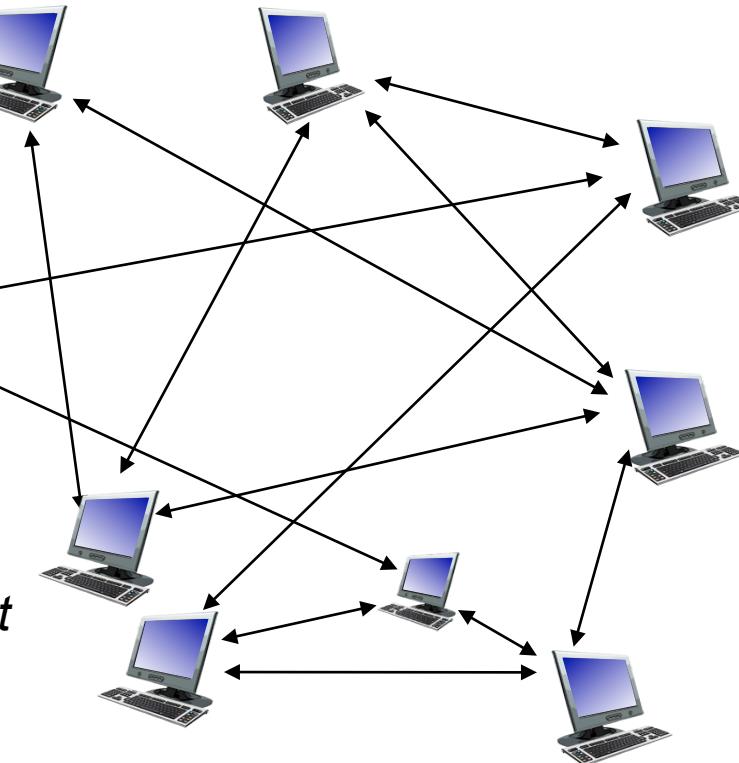
**tracker:** tiene traccia dei peer che partecipano ad un torrent



Alice ottiene la lista di peer dal tracker... ...

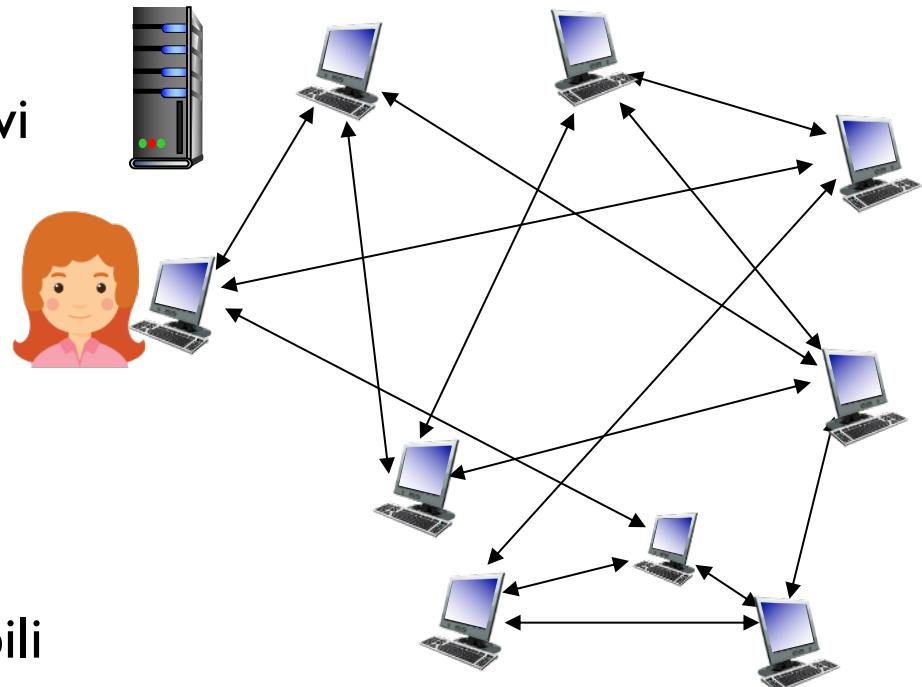
... ed inizia a scambiare *chunk* con i peer nel *torrent*

**torrent:** gruppo di peer che si scambiano chunk di un file



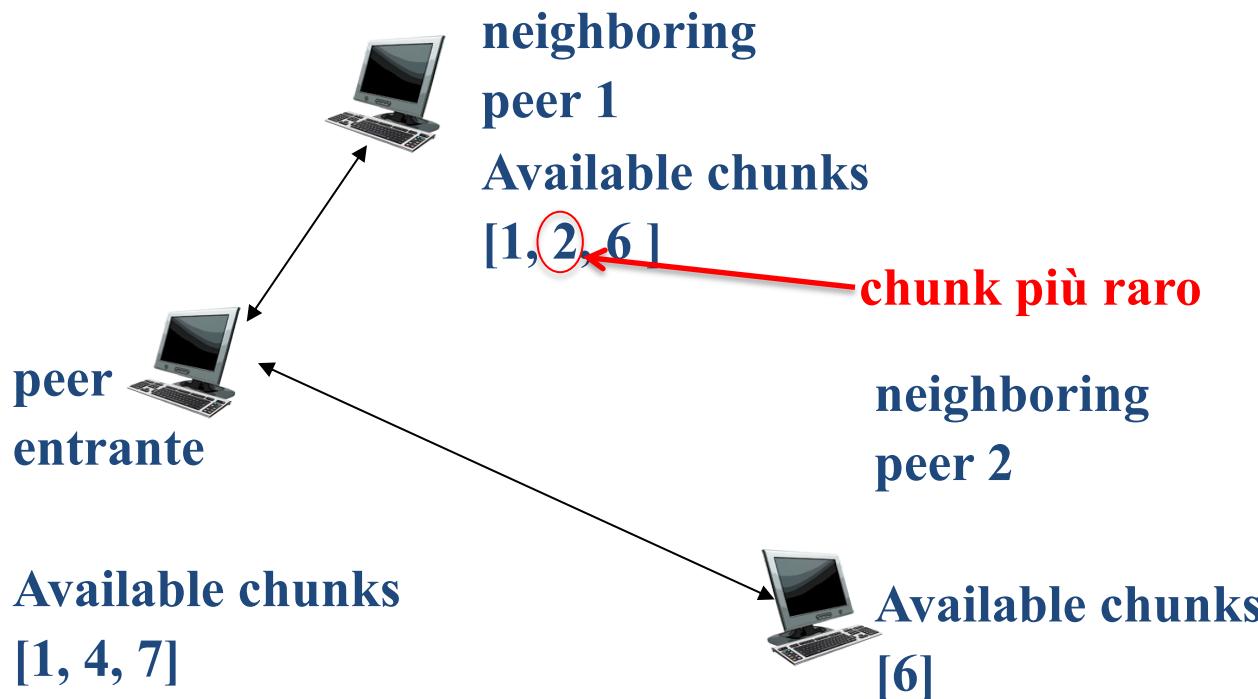
# BitTorrent – entrare nel *torrent*

- I *peer* che entrano in un *torrent* si registrano presso un *tracker* per ottenere una lista di *peer* “attivi”
- Il *tracker* invia una lista di *peer* attivi su un *torrent* (indirizzi IP)
- Il *peer* entrante stabilisce connessioni TCP con un sottinsieme dei *peer* nella lista (*neighboring peers*)
- I *neighboring peers* inviano al *peer* entrante la lista dei *chunk* disponibili
- Il *peer* entrante sceglie quale *chunk* scaricare e da quale *peer* scaricare *chunk* secondo meccanismi euristici



# Meccanismo di richiesta di *chunk*

- Il principio del *Rarest First*
  - Il peer entrante, tra tutti i *chunk* mancanti, scarica prima i *chunk* più rari nelle liste di *chunk* ricevute da tutti i *neighboring peer*



# Meccanismo di invio di *chunk*

- Il *peer* entrante risponde a richieste che provengono dagli x *peer* che inviano *chunk* al massimo *rate*
- Tutti gli altri *peer* sono strozzati (*choked*)
- I migliori x *peer* sono ricalcolati periodicamente (10[s])
- Ogni 30[s] un nuovo *peer* viene scelto casualmente per l'invio di *chunk* (*optimistic unchoking*)

