



**POLITECNICO**  
MILANO 1863



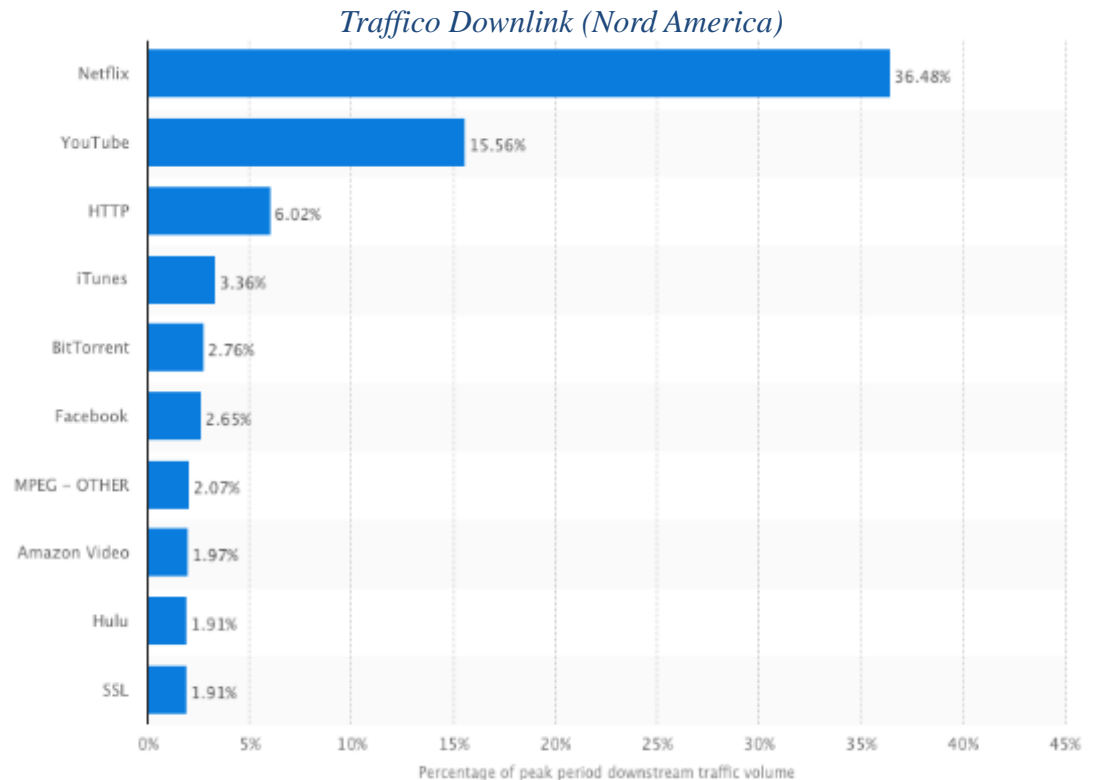
# **Il Livello Applicativo**

## **Capitolo 2**

**Fondamenti di Internet e Reti**

# 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:**
  - Skype, Hangout, ecc..
- **Network games**
- **Video conference**
- **Massive parallel computing**
- **Instant messaging**
- **Remote login:**
  - TELNET
- ...

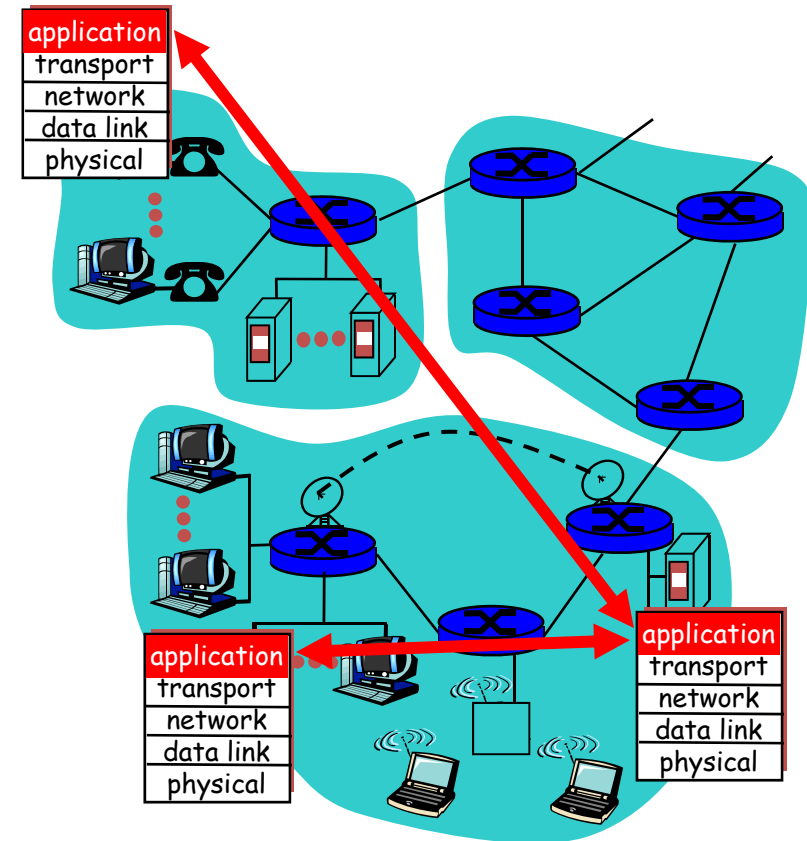


Source: [www.statista.com](http://www.statista.com) (2016)



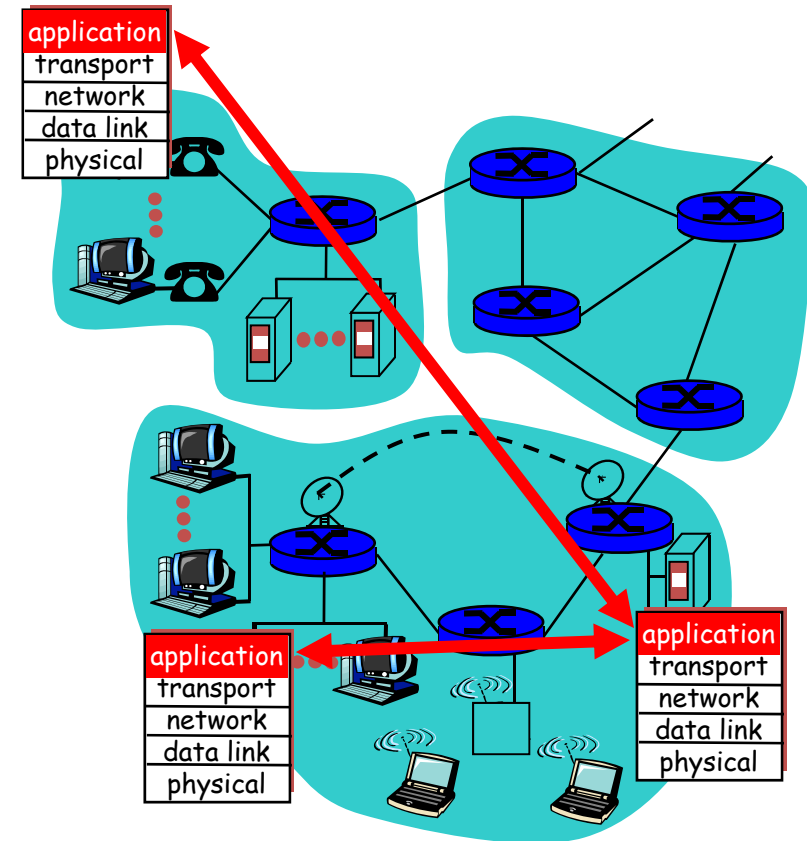
# Creare un'applicazione di rete

- 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!)



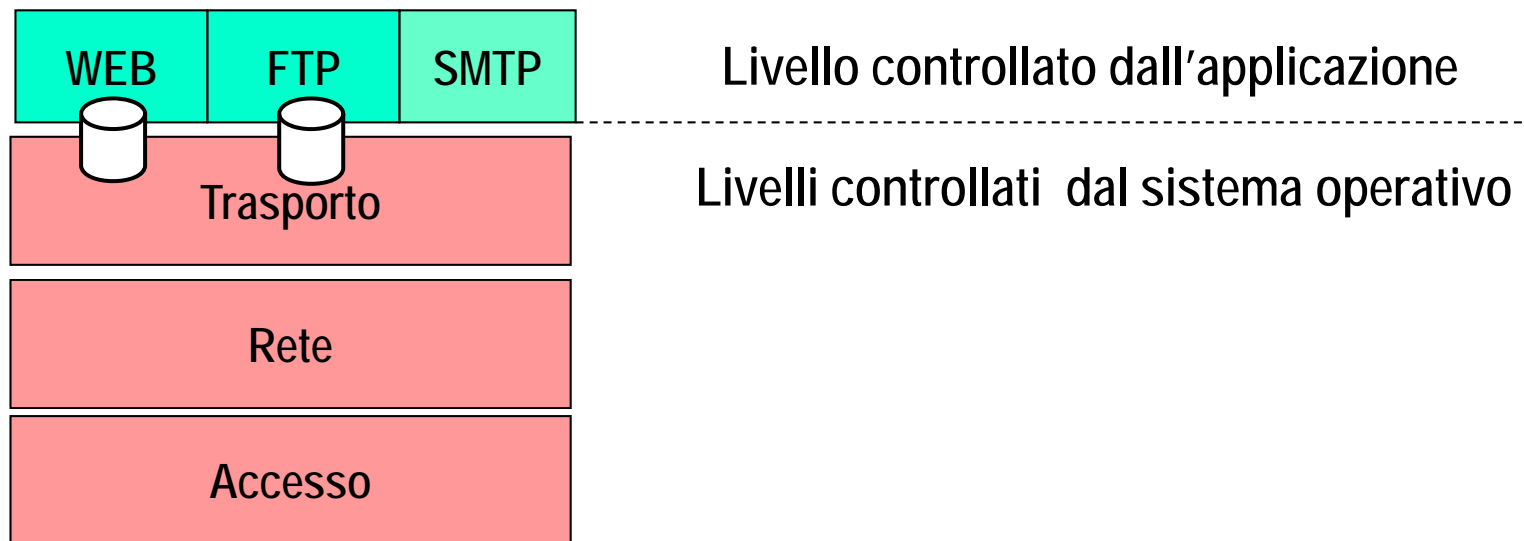
# Ingredienti per la comunicazione tra processi remoti

- 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



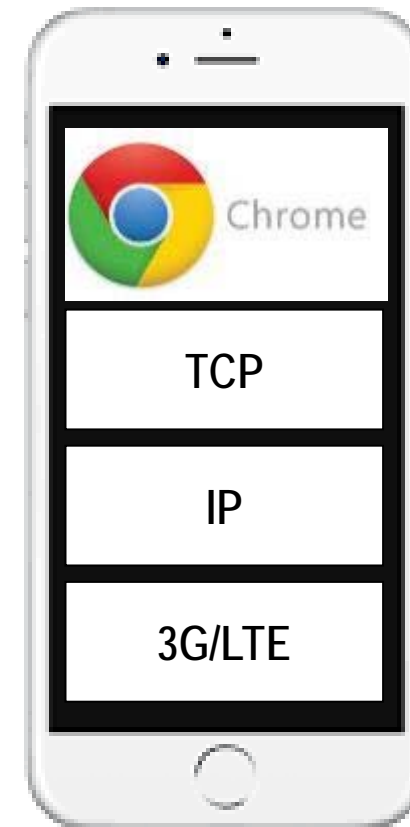
# Indirizzamento di processi applicativi

- 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

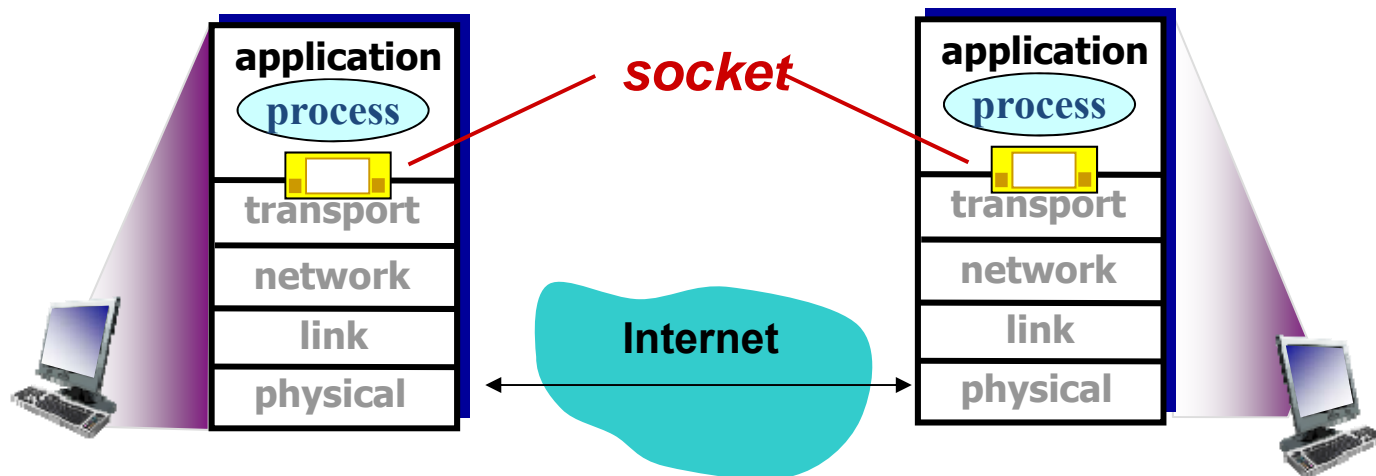
- Cosa serve per identificare il mio browser *Chrome* in esecuzione sul mio *host*?
  - Indirizzo del mio *host*
    - Indirizzo IP univoco di 32 bit (di cui parleremo ampiamente in seguito)
  - Indirizzo del SAP del processo in esecuzione sul mio *host*
    - numero di porta





# 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

- Perdita di dati
  - 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 dello scambio dati
- *Trasporto affidabile* senza perdita di dati
- *Controllo di flusso*: il trasmettitore 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) RTP	TCP, UDP, DASH
Internet telephony	(e.g., Vonage, Dialpad)	typically UDP

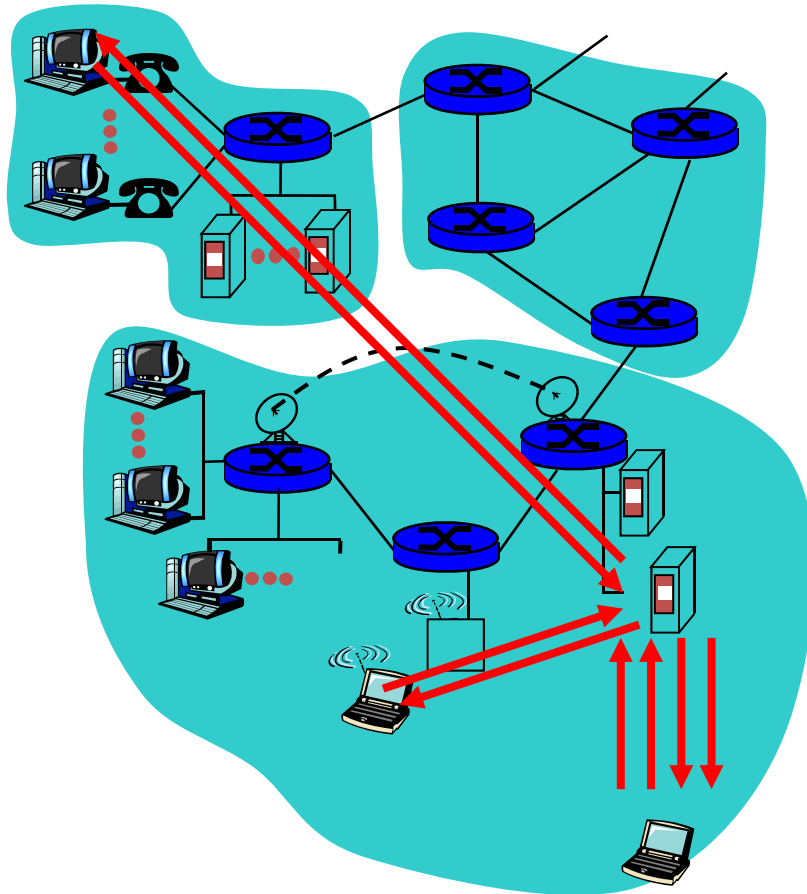


# Architetture applicative

- ***Client-server***
  - I dispositivi coinvolti nella comunicazione implementano o solo il processo *client* o solo il processo *server*
  - I dispositivi *client* e *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*

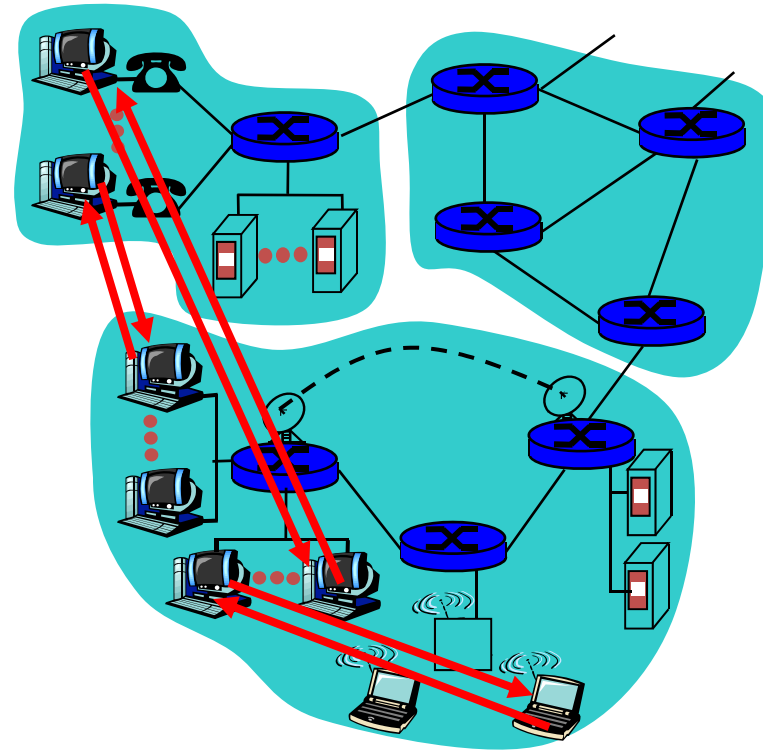
## *Client:*

- 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







**POLITECNICO**  
MILANO 1863



# **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, immagini JPEG, applet Java, file audio, 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

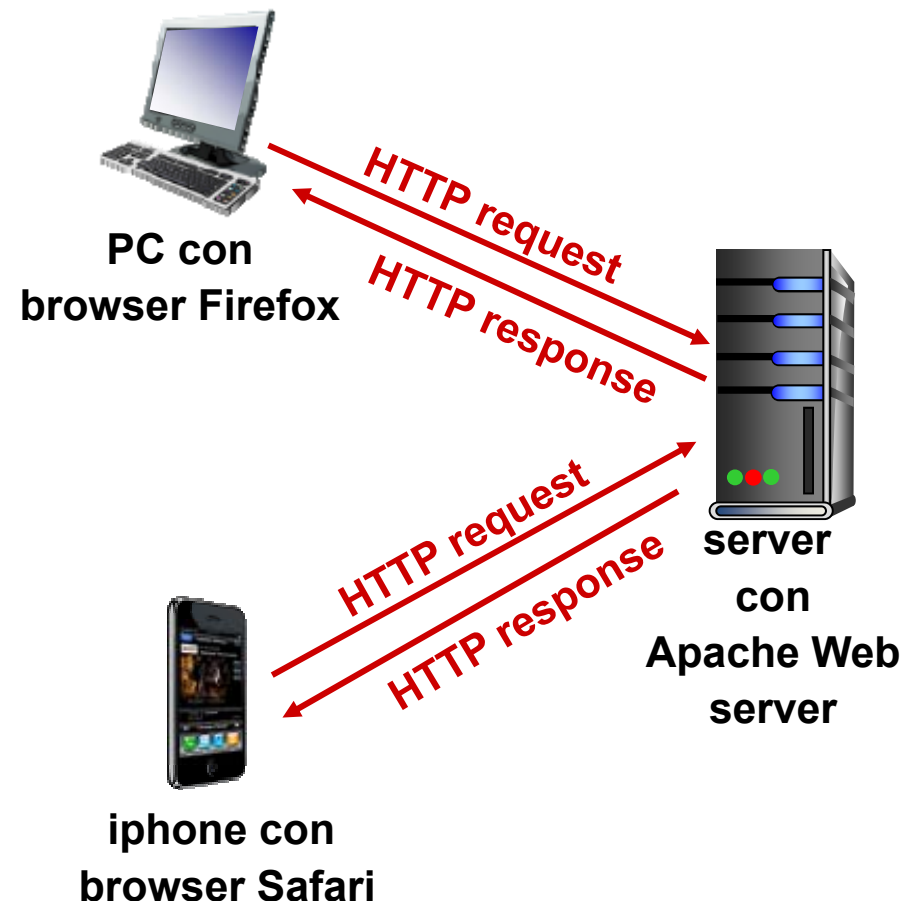
Indica il numero di porta (opzionale)

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

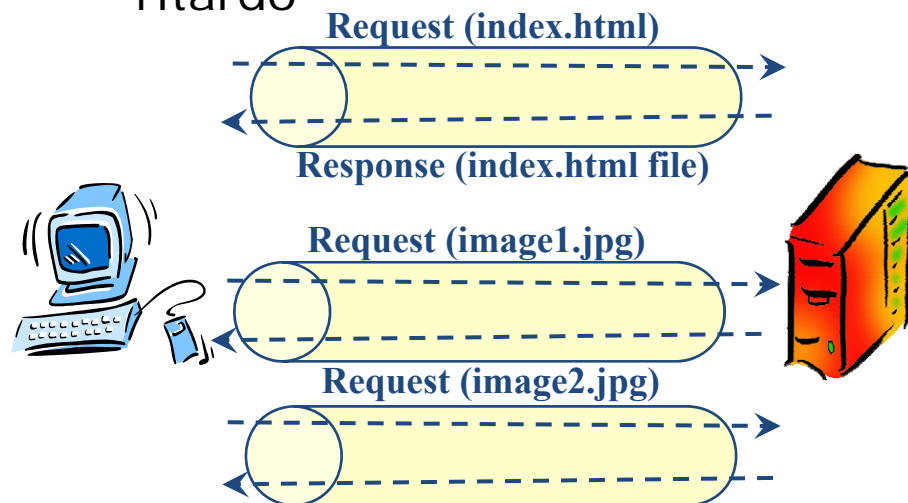
*Vedremo 1, 2 e 4 nei laboratori su socket programming*



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

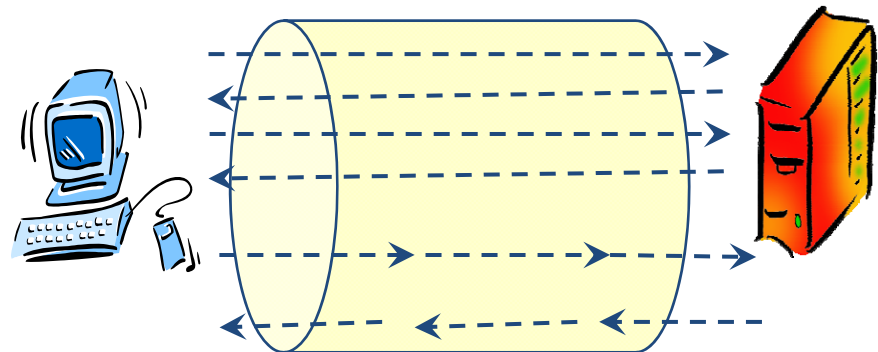
## Connessione non persistente

- 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

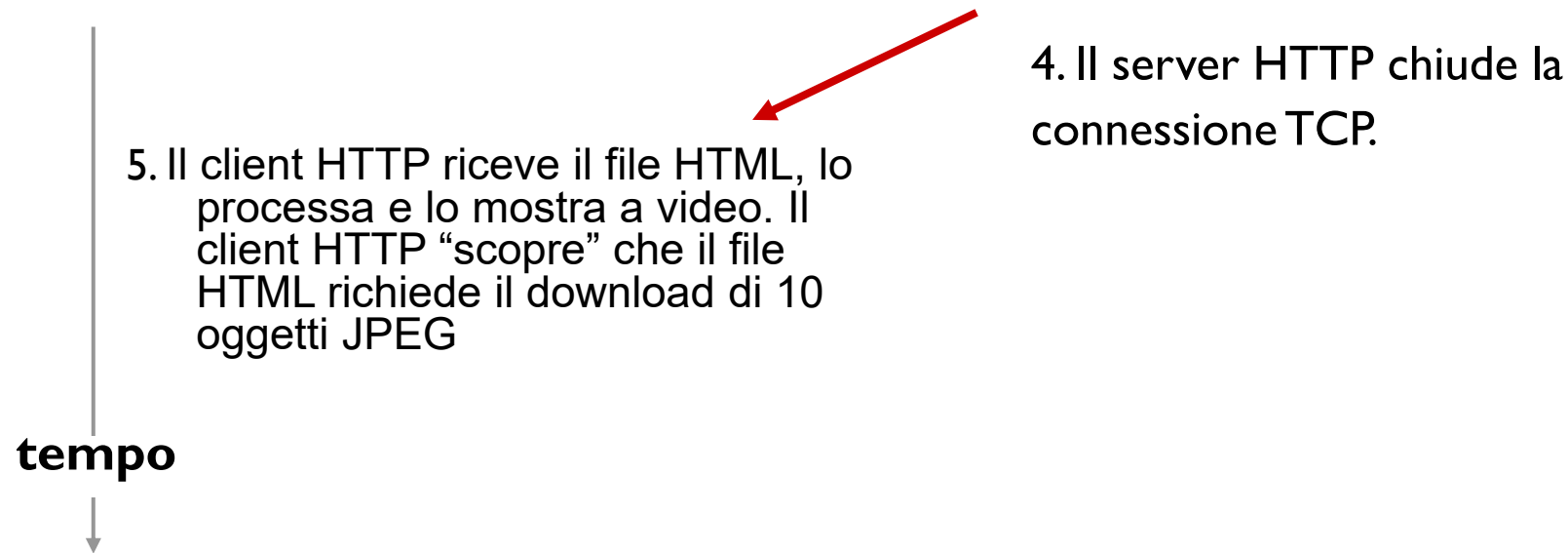
2 il client HTTP invia una richiesta HTTP (contenente la URL) tramite la connessione TCP. La richiesta indica che il client vuole l'oggetto /home/index.html

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

**tempo**



# Esempio di connessione non persistente



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

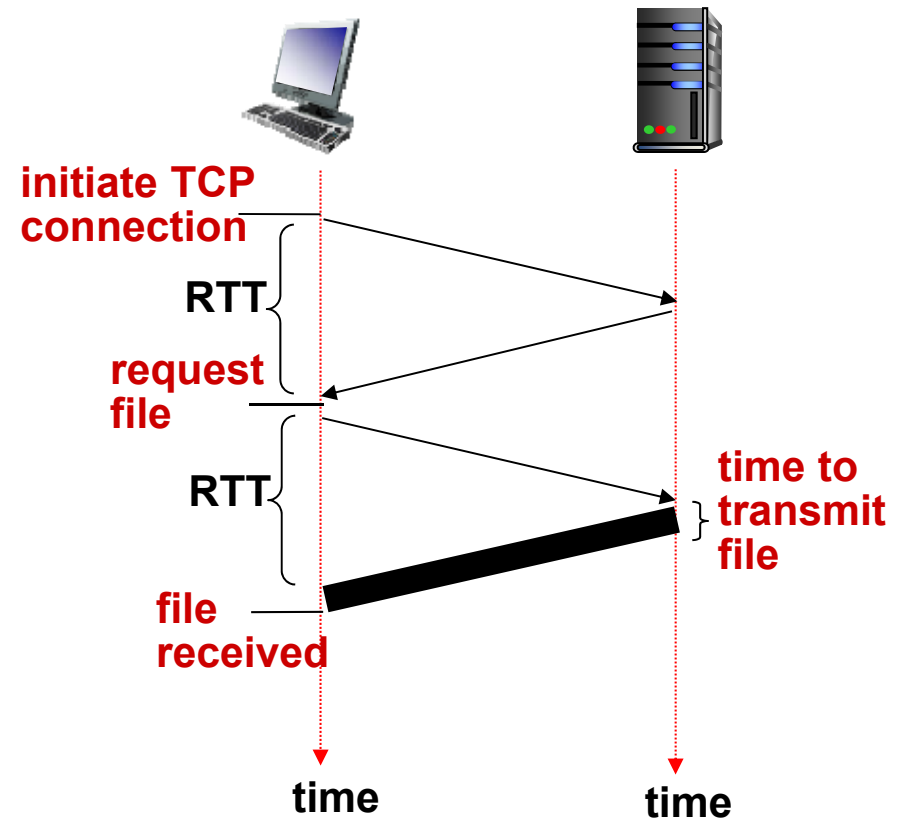


# Stima del tempo di trasferimento in HTTP

- *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..)
- Supponendo che la pagina web sia composta da 11 oggetti (un file HTML e 10 immagini JPEG), il tempo di download dell’intera pagina web è:

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

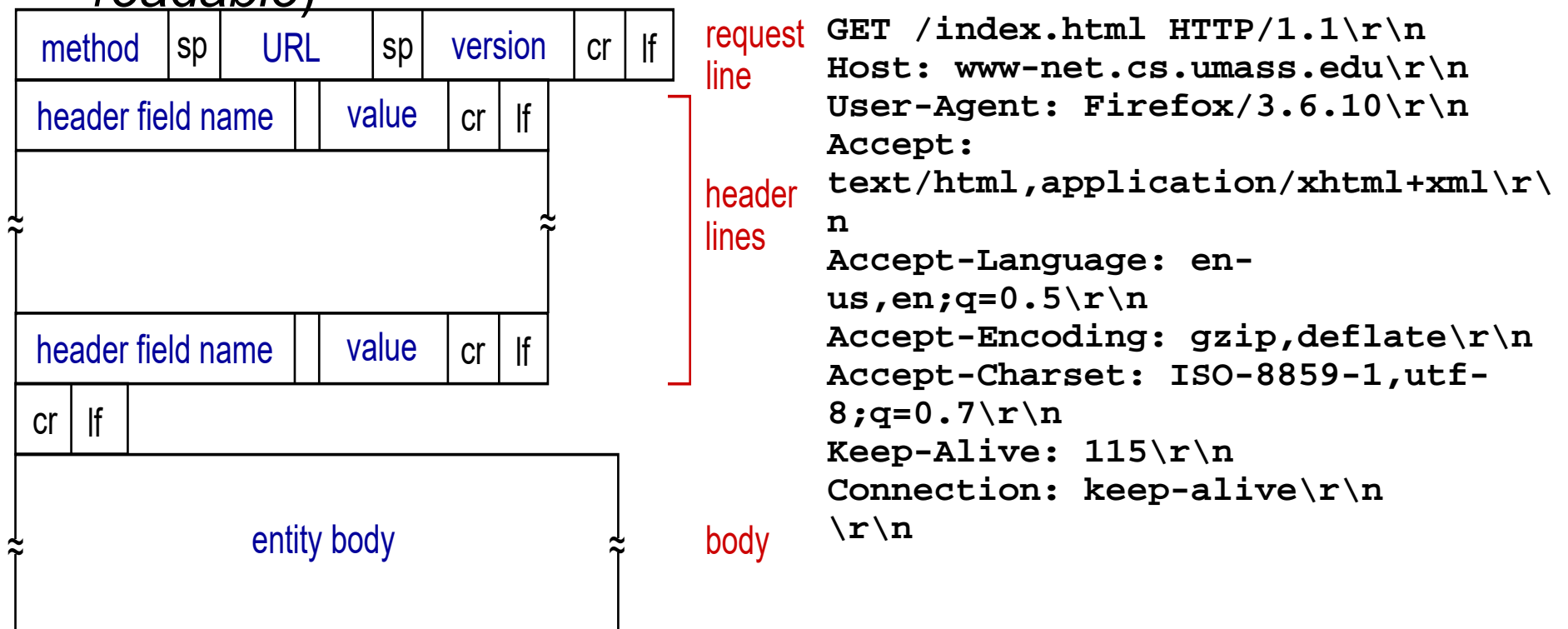
$$T_{pers} = RTT + \sum_{i=0}^{10} (RTT + T_i)$$





# Le richieste HTTP

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



# Esempi di Metodi HTTP

GET	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.
HEAD	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.
POST	E' usato per fornire degli input al server da utilizzare per un particolare oggetto (di solito un applicativo) identificato nell'URL.
PUT	E' utilizzato per memorizzare un documento nel server. Il documento viene fornito nel corpo del messaggio e la posizione di memorizzazione nell'URL.
DELETE	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**

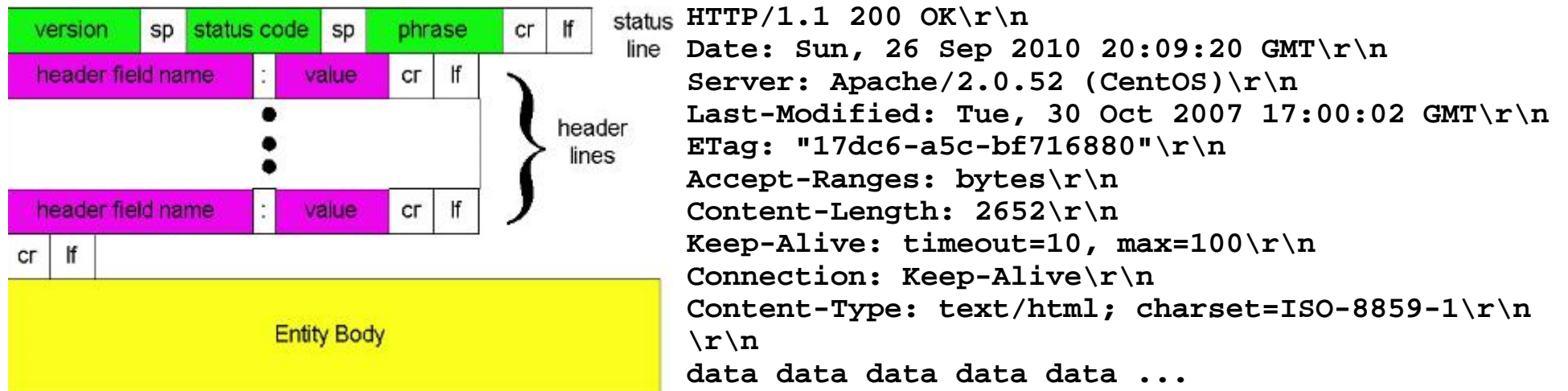
:

**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



# Gli *Status Code* HTTP più comuni

## 200 OK

- richiesta ha avuto successo, l'oggetto richiesto è nel corpo del messaggio di risposta

## 301 Moved Permanently

- l'oggetto richiesto è stato spostato; la nuova posizione (URL) dell'oggetto è inclusa nel messaggio di risposta (Location:)

## 400 Bad Request

- messaggio di richiesta non compreso dal server

## 404 Not Found

- oggetto richiesto non trovato sul server

## 505 HTTP Version Not Supported



# 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)

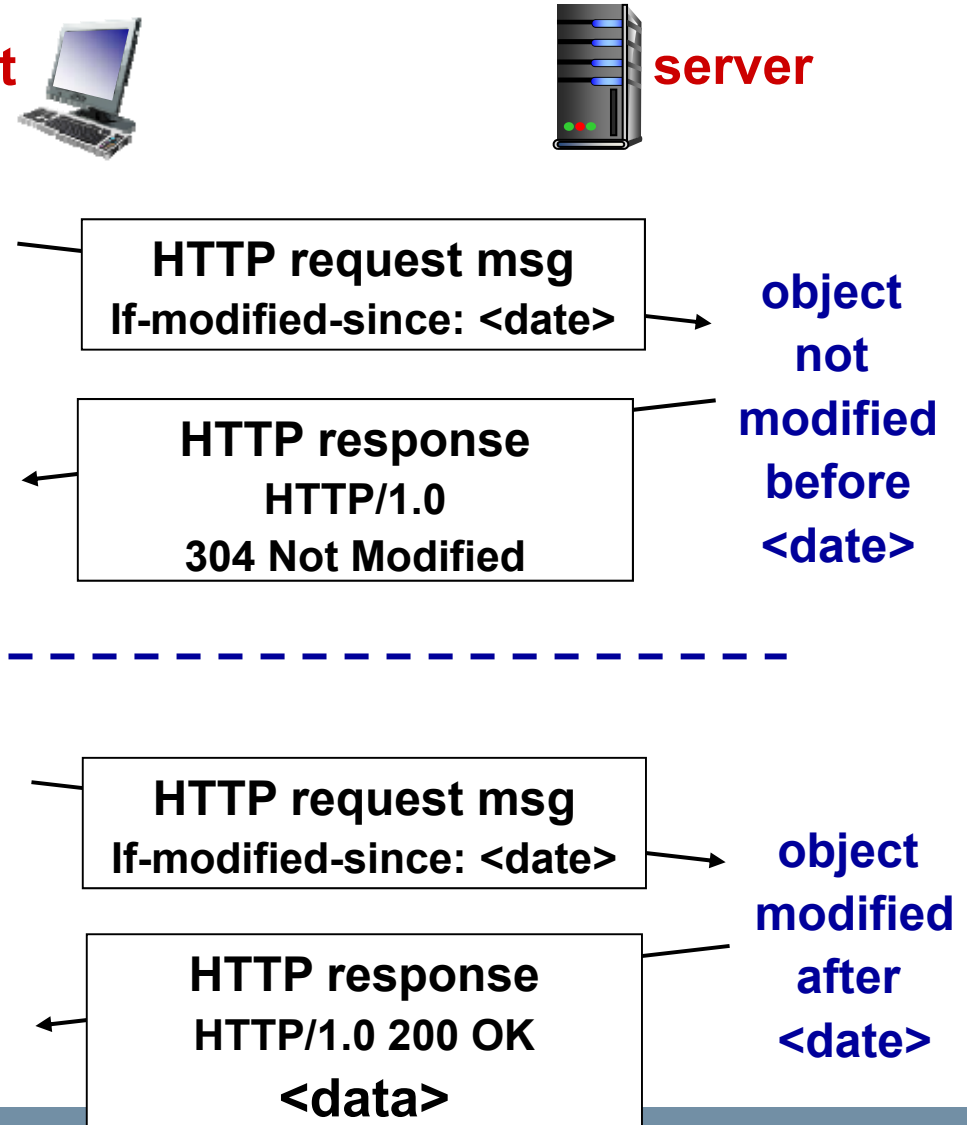
- Risposta

```
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 data ...
```



# Conditional GET

- **Obiettivo:** non inviare un **client** 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*



# Semplici Esperimenti con HTTP - Telnet

- Apriamo una connessione Telnet con un server web

```
telnet www.antlab.polimi.it 80
```

**Apri una connessione TCP  
verso la porta 80 del server  
[www.antlab.polimi.it](http://www.antlab.polimi.it)**

- Inviamo una richiesta HTTP (proviamo GET e HEAD)

```
GET /people/matteo-cesana HTTP/1.1
```

```
Host: www.antlab.polimi.it
```

```
HEAD /people/matteo-cesana HTTP/1.1
```

```
Host: www.antlab.polimi.it
```

- Cosa riceviamo?

**Richiede l'oggetto**

**[/people/matteo-cesana](http://www.antlab.polimi.it/people/matteo-cesana)**

**In Windows 10:**

- Abilitare Telnet
- Aprire cmd
- telnet
- set LOCALECHO
- open [www.antlab.polimi.it](http://www.antlab.polimi.it) 80





# 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
- Vediamo qualche esempio usando *Chrome Dev Tools*  
<https://developers.google.com/web/tools/chrome-devtools/>



# 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:
  - Ottenere una pagina web:  
`curl www.antlab.polimi.it`
  - E se vogliamo solo l'intestazione della pagina web?  
`curl --head www.antlab.polimi.it`
  - Monitoriamo lo scambio di messaggi tra client e server `curl -trace-ascii filename.txt 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`

In Windows 10:

```
curl www.antlab.polimi.it --trace-ascii  
C:\Users\maier\log.txt --trace-time
```



# Mantenere uno “stato” in HTTP: i cookies

## *Ingredienti dei cookies:*

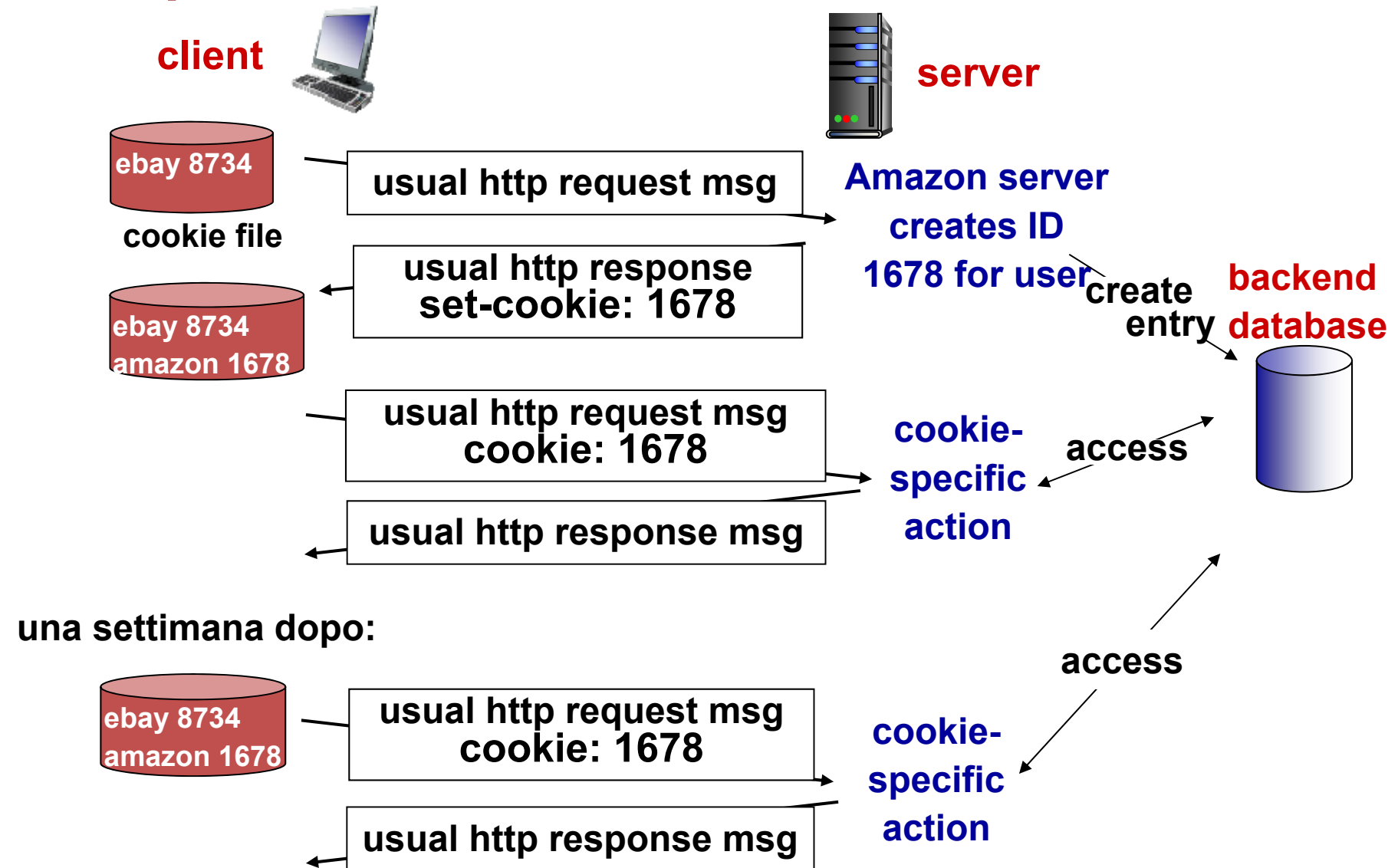
- 1) Un header cookie nelle risposte HTTP
- 2) Un header cookie nella prossima richiesta HTTP
- 3) Una lista di cookie mantenuta sull'host (dal browser)
- 4) Un data base di cookie mantenuto dal sito web

## *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

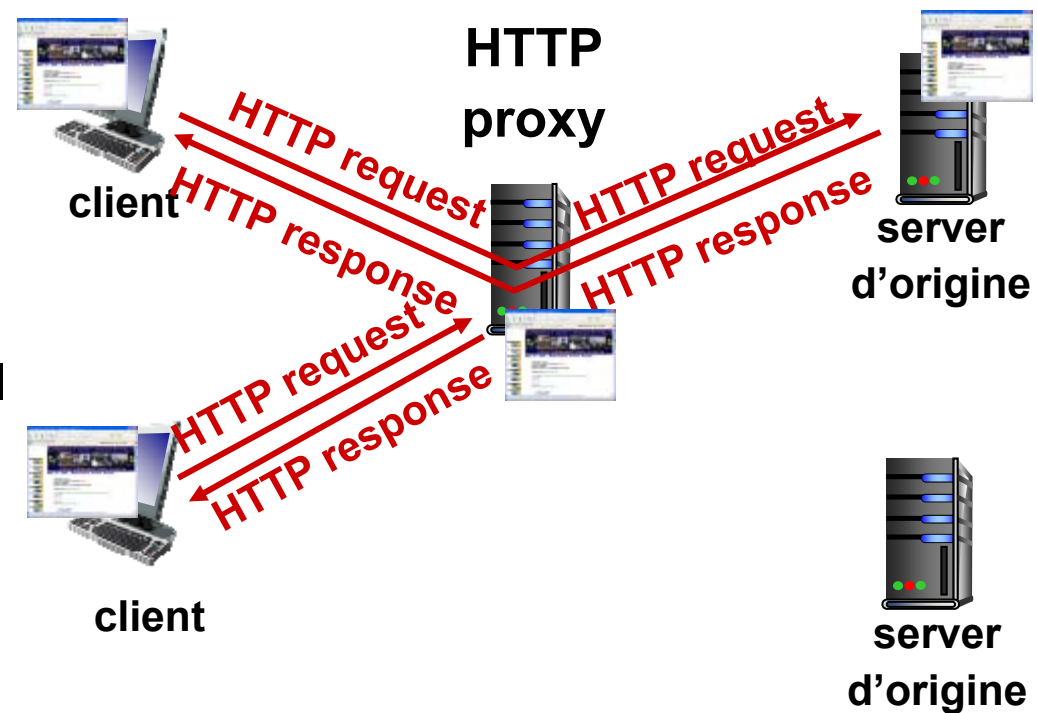


# Esempio di uso dei cookie



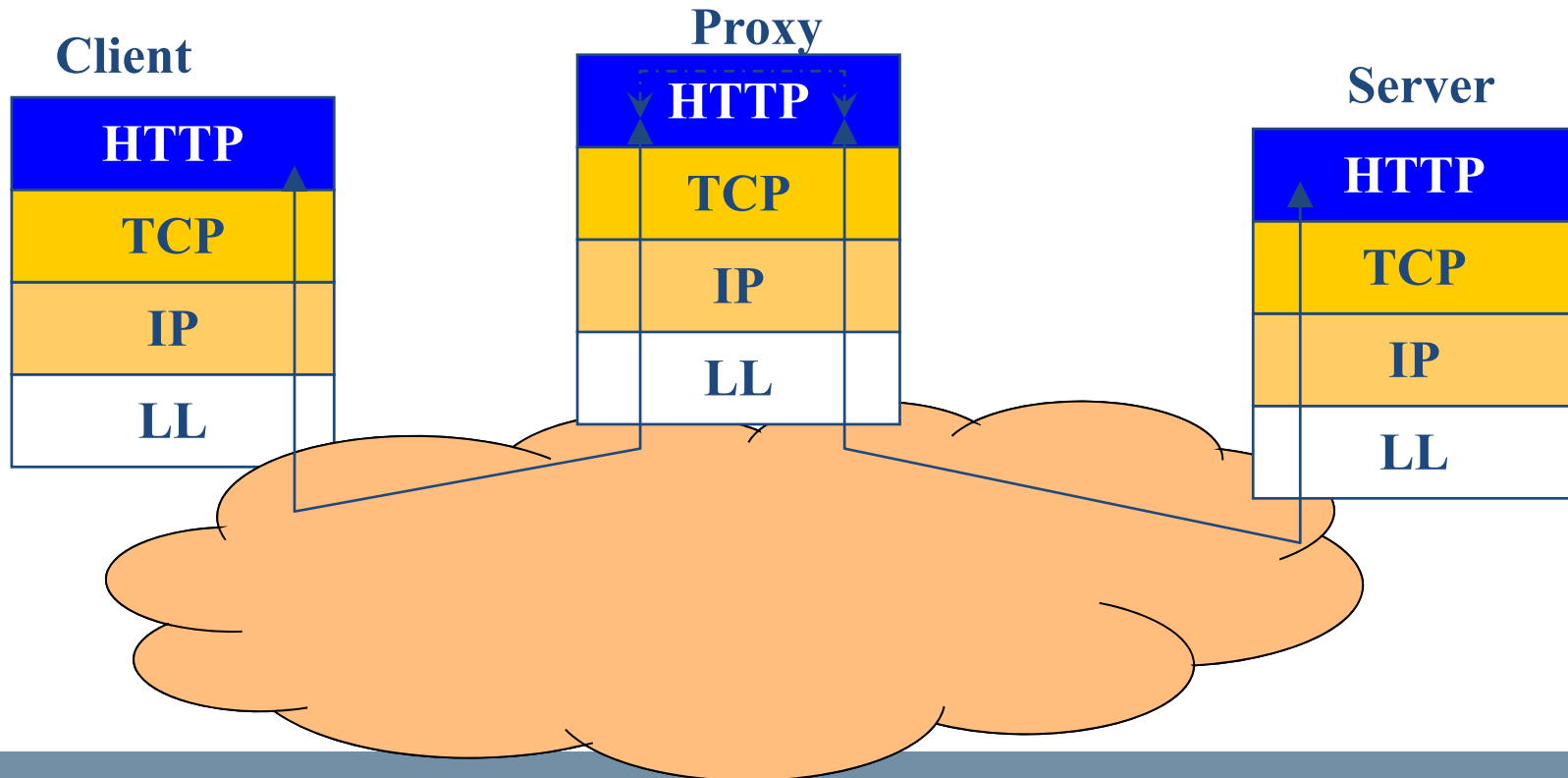
# I proxy HTTP: Cache di rete

- 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*)



# Verifichiamo il funzionamento dei *proxy* HTTP

- Accediamo alla pagina [www.google.com](http://www.google.com)
- Verifichiamo a quale indirizzo IP corrisponde il nome [www.google.com](http://www.google.com) (come possiamo fare?)
- Impostiamo ora il *proxy* HTTP del Politecnico
- Accediamo alla pagina [www.google.com](http://www.google.com)
- Verifichiamo tramite WS che le richieste HTTP vengono “servite” dal proxy del Politecnico
- Quale è l'indirizzo IP del proxy del Politecnico?



\*Connessione alla rete locale (LAN) [Wireshark 1.12.6 (v1.12.6-0-gee1fce6 from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
35	0.576354000	fe80::3cb2:c74d:3585:d542	ff02::c	SSDP	514	NOTIFY * HTTP/1.1
36	0.577284000	fe80::3cb2:c74d:3585:d542	ff02::c	SSDP	569	NOTIFY * HTTP/1.1
37	0.578306000	fe80::3cb2:c74d:3585:d542	ff02::c	SSDP	557	NOTIFY * HTTP/1.1
38	0.579250000	fe80::3cb2:c74d:3585:d542	ff02::c	SSDP	505	NOTIFY * HTTP/1.1
39	0.579414000	fe80::3cb2:c74d:3585:d542	ff02::c	SSDP	585	NOTIFY * HTTP/1.1
40	0.580248000	fe80::3cb2:c74d:3585:d542	ff02::c	SSDP	571	NOTIFY * HTTP/1.1
139	2.069926000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
403	6.071713000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
583	9.071103000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
792	12.082506000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
932	13.919086000	10.79.0.205	216.58.208.36	HTTP	1210	GET / HTTP/1.1
938	13.933348000	216.58.208.36	10.79.0.205	HTTP	525	HTTP/1.1 302 Found (text/html)
939	13.936340000	10.79.0.205	216.58.208.35	HTTP	1384	GET /?gfe_rd=cr&ei=-N_mvonbDizb8AemhpP4Dg HTTP/1.1
943	13.991294000	216.58.208.35	10.79.0.205	HTTP	628	HTTP/1.1 302 Found (text/html)
1221	16.082956000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
1451	19.095089000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1

Frame 932: 1210 bytes on wire (9680 bits), 1210 bytes captured (9680 bits) on interface 0

Ethernet II, Src: Dell\_Ea:03:8a (18:03:73:ea:03:8a), Dst: All-HSRP-routers\_c0 (00:00:0c:07:ac:c0)

Internet Protocol Version 4, Src: 10.79.0.205 (10.79.0.205), Dst: 216.58.208.36 (216.58.208.36)

Transmission Control Protocol, Src Port: 52593 (52593), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 1156

Hypertext Transfer Protocol

GET / HTTP/1.1\r\n

[Expert Info (chat/Sequence): GET / HTTP/1.1\r\n]

Request Method: GET

Request URI: /

Request Version: HTTP/1.1

Host: www.google.com\r\n

Connection: keep-alive\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8\r\n

Upgrade-Insecure-Requests: 1\r\n

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36\r\n

X-Client-Data: cKa2yQEIXLbJAQj9lcoB\r\n

Accept-Encoding: gzip, deflate, sdch\r\n

Accept-Language: it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4\r\n

[truncated]Cookie: CONSENT=YES+IT.it+20150719-14-0; SID=DQAAACGBAABT-oBaphsJE63vLbwdCTyHOZG9CvsF5e449t\_7jJdraws3\_yHAMpwkjjw0mjN7Bmrsvr5P8EYbo1w392wyAGXqAg3QaDUXZA61Nm--\_wb5QWHDjo9xpcqVBo50-9Eok

0040 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e /1.1..Host: www.

0050 6f 6f 6f 6f 6c 65 2e 63 6f 6d 0d 0a 43 6f 6e 6e google.c om..Conn

0060 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 ection: keep-ali

0070 76 65 0d 0a 41 63 63 65 70 74 3a 20 74 65 78 74 ve..Acce pt: text

0080 2f 68 74 6d 6c 2c 61 70 70 6c 69 63 61 74 69 6f /html,ap plicatio

HTTP Host (http.host), 22 bytes Packets: 1524 · Displayed: 16 (1,0%) · Dropped: 0 (0,0%) Profile: Default

Skype™ - fmusumeci85 Google - Google Chr... \*Connessione alla ret... Proprietà - Internet senza\_proxy1 - Paint

ITA 17:03 14/03/2016



\*Connessione alla rete locale (LAN) [Wireshark 1.12.6 (v1.12.6-0-gee1fce6 from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1914	22.765552000	fe80::8fe:84b1:4fa9:4e79	ff02::c	SSDP	581	NOTIFY * HTTP/1.1
1955	23.123308000	fe80::8fe:84b1:4fa9:4e79	ff02::c	SSDP	567	NOTIFY * HTTP/1.1
1988	23.267418000	fe80::8fe:84b1:4fa9:4e79	ff02::c	SSDP	565	NOTIFY * HTTP/1.1
2003	23.445027000	fe80::8fe:84b1:4fa9:4e79	ff02::c	SSDP	501	NOTIFY * HTTP/1.1
2035	23.920579000	fe80::8fe:84b1:4fa9:4e79	ff02::c	SSDP	510	NOTIFY * HTTP/1.1
2069	24.248149000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
2148	25.186728000	fe80::8fe:84b1:4fa9:4e79	ff02::c	SSDP	553	NOTIFY * HTTP/1.1
2212	25.755122000	fe80::f0ca:4d0c:a983:d518	ff02::c	SSDP	181	M-SEARCH * HTTP/1.1
2213	25.756071000	fe80::f0ca:4d0c:a983:d518	ff02::c	SSDP	179	M-SEARCH * HTTP/1.1
2227	25.832264000	10.79.0.205	131.175.12.65	HTTP	1237	GET http://www.google.com/ HTTP/1.1
2234	25.847967000	131.175.12.65	10.79.0.205	HTTP	312	HTTP/1.1 302 Found (text/html)
2237	25.857736000	10.79.0.205	131.175.12.65	HTTP	1358	GET http://www.google.it/?gfe_rd=cr&ei=mehmvo_eDq-P8QfnhZmACw HTTP/1.1
2249	25.917898000	131.175.12.65	10.79.0.205	HTTP	328	HTTP/1.1 302 Found (text/html)
2335	26.063672000	131.175.12.65	10.79.0.205	HTTP	1414	Continuation
2463	27.262833000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
2868	31.253680000	fe80::11c1:391e:a855:e190	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1

Frame 2227: 1237 bytes on wire (9896 bits), 1237 bytes captured (9896 bits) on interface 0

Ethernet II, Src: Dell\_ea:03:8a (18:03:73:ea:03:8a), Dst: All-HSRP-routers\_c0 (00:00:0c:07:ac:c0)

Internet Protocol Version 4, Src: 10.79.0.205 (10.79.0.205), Dst: 131.175.12.65 (131.175.12.65)

Transmission Control Protocol, Src Port: 52620 (52620), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 1183

Hypertext Transfer Protocol

GET http://www.google.com/ HTTP/1.1\r\n

[Expert Info (Chat/Sequence): GET http://www.google.com/ HTTP/1.1\r\n]

Request Method: GET

Request URI: http://www.google.com/

Request Version: HTTP/1.1

Host: www.google.com\r\n

Proxy-Connection: keep-alive\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8\r\n

Upgrade-Insecure-Requests: 1\r\n

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36\r\n

X-Client-Data: CKA2yQEIXLbJAQj9lcoB\r\n

Accept-Encoding: gzip, deflate, sdch\r\n

Accept-Language: it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4\r\n

[truncated]Cookie: CONSENT=YES+IT.it+20150719-14-0; SID=DQAAACGBAABT-oBaphsje63VLbwdCTyH0ZG9cvsF5e449t\_7jjdraws3\_yHAMpwkjjw0mjN7Bmrsvr5P8EYbo1w392wyAGXqag3QaDUXZA61Nm--\_wb5QWHDjo9xpcqVBo50-9Eok

0000 00 00 0c 07 ac c0 18 03 73 ea 03 8a 08 00 45 00 ..... S.....E.

0010 04 c7 79 b1 00 00 80 06 00 00 0a 4f 00 cd 83 af .....Y.....O...

0020 0c 41 cd 8c 1f 90 1b 2b e2 eb 2b e5 2a 1c 50 18 .....A.....+..\*..P.

0030 ff 00 9f c5 00 00 47 45 54 20 68 74 74 70 3a 2f .....GE T http://

0040 2f 77 77 2e 67 6f 6f 67 6c 65 2e 63 6f 6d 2f ...../www.goo gle.com/

File: "C:\Users\FRANCE~1\AppData\Local\Temp\wireshark\_... Packets: 2892 · Displayed: 66 (2,3%) · Dropped: 0 (0,0%)

Profile: Default

Skype™ - fmusumeci85 Google - Google Chr... \*Connessione alla ret... Proprietà - Internet

ITA 17:09 14/03/2016

\*Connessione alla rete locale (LAN) [Wireshark 1.12.6 (v1.12.6-0-gee1fce6 from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
942	6.553998000	10.79.0.205	131.175.12.65	HTTP	277	CONNECT v2.auc.avira.com:443 HTTP/1.1
947	6.573584000	131.175.12.65	10.79.0.205	HTTP	93	HTTP/1.1 200 Connection established
948	6.573828000	10.79.0.205	131.175.12.65	TLSv1.2	571	Client Hello
949	6.595538000	131.175.12.65	10.79.0.205	TLSv1.2	1414	Server Hello
962	6.604409000	131.175.12.65	10.79.0.205	TLSv1.2	1221	Certificate
963	6.607197000	10.79.0.205	131.175.12.65	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
966	6.626116000	131.175.12.65	10.79.0.205	TLSv1.2	296	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
968	6.694288000	10.79.0.205	131.175.12.65	TLSv1.2	944	Application Data
971	6.718277000	131.175.12.65	10.79.0.205	TLSv1.2	406	Application Data
972	6.720996000	10.79.0.205	131.175.12.65	TLSv1.2	944	Application Data
973	6.750225000	131.175.12.65	10.79.0.205	TLSv1.2	406	Application Data
975	6.752350000	10.79.0.205	131.175.12.65	HTTP	1347	GET http://www.corriere.it/ HTTP/1.1
1005	6.810202000	131.175.12.65	10.79.0.205	HTTP	534	GET http://tags.tiqcdn.com/utag/rcsmedia/corriere/prod/utag.sync.js HTTP/1.1
1008	6.816067000	131.175.12.65	10.79.0.205	HTTP	421	HTTP/1.1 304 Not Modified
1024	6.820356000	131.175.12.65	10.79.0.205	HTTP	787	HTTP/1.1 200 OK (text/html)
1026	6.821293000	10.79.0.205	131.175.12.65	HTTP	529	GET http://tags.tiqcdn.com/utag/rcsmedia/corriere/prod/utag.js HTTP/1.1

Transmission Control Protocol, Src Port: 52643 (52643), Dst Port: 8080 (8080), Seq: 1361, Ack: 1, Len: 1293

[2 Reassembled TCP Segments (2653 bytes): #974(1360), #975(1293)]

Hypertext Transfer Protocol

GET http://www.corriere.it/ HTTP/1.1\r\n

[Expert Info (Chat/Sequence): GET http://www.corriere.it/ HTTP/1.1\r\n]

Request Method: GET

Request URI: http://www.corriere.it/

Request Version: HTTP/1.1

Host: www.corriere.it\r\n

Proxy-Connection: keep-alive\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8\r\n

Upgrade-Insecure-Requests: 1\r\n

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36\r\n

Accept-Encoding: gzip, deflate, sdch\r\n

Accept-Language: it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4\r\n

[truncated]Cookie: cpmt\_xa=5334; \_\_ric=5334%3AMon%20Jul%2027%202015%2018%3A15%3A38%20GMT+0200%2028ora%20legale%20Europa%20occidentale%29%7C; rccsLocalPref=milano%7Cmilano%7C015146; fbm\_203568\r\n

[Full request URI: http://www.corriere.it/http://www.corriere.it/]

[HTTP request 1/3]

0000 00 00 0c 07 ac c0 18 03 73 ea 03 8a 08 00 45 00 .....S....E.

0010 05 35 7b c3 40 00 80 06 00 00 0a 4f 00 cd 83 af .5{.@... ..O...

0020 0c 41 cd a3 1f 90 cd bb 08 52 5c bf b9 3e 50 18 .A.....R...>P.

Frame (1347 bytes) Reassembled TCP (2653 bytes)

File: "C:\Users\FRANCE-1\AppData\Local\Temp\wireshark..." Packets: 2109 · Displayed: 250 (11,9%) · Dropped: 0 (0,0%) Profile: Default

Skype™ - fmsumeci85 Corriere della Sera - G... \*Connessione alla rete... Proprietà - Internet

17:11 14/03/2016



**POLITECNICO**  
MILANO 1863



# **Il servizio di posta elettronica**

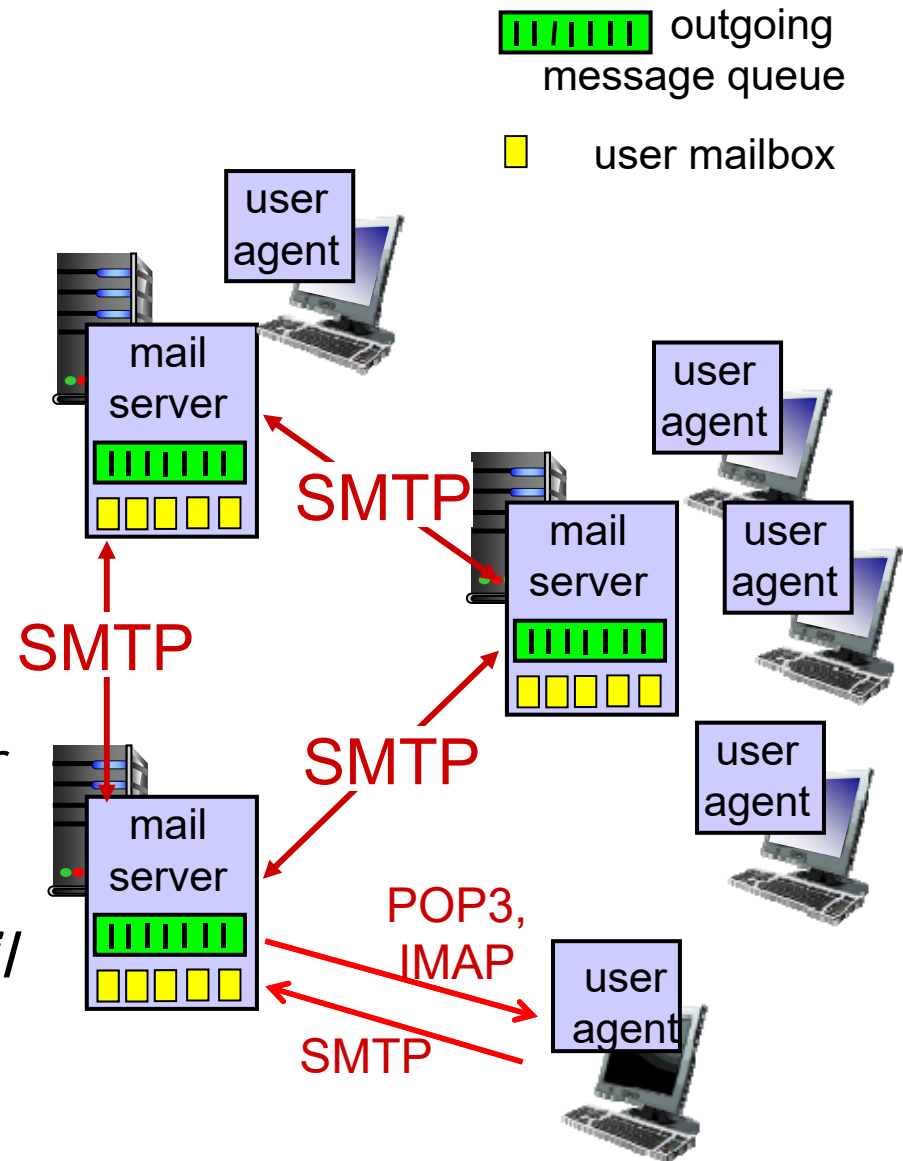
**Simple Mail Transfer Protocol (SMTP) RFC 5321**

**Post Office Protocol (POP3) RFC 1939**

**Internet Mail Control Protocol (IMAP) RFC 3501**

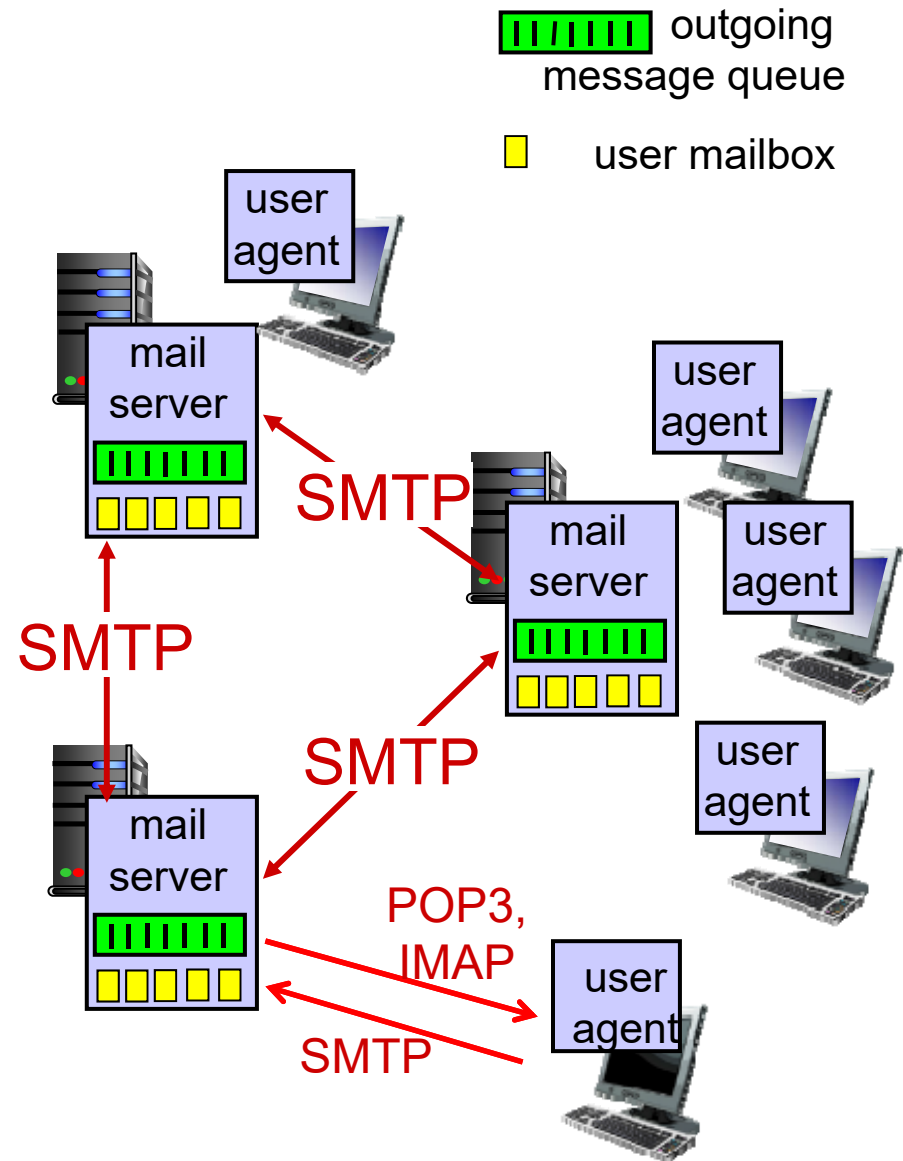
# Il servizio di E-mail

- *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

- 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*



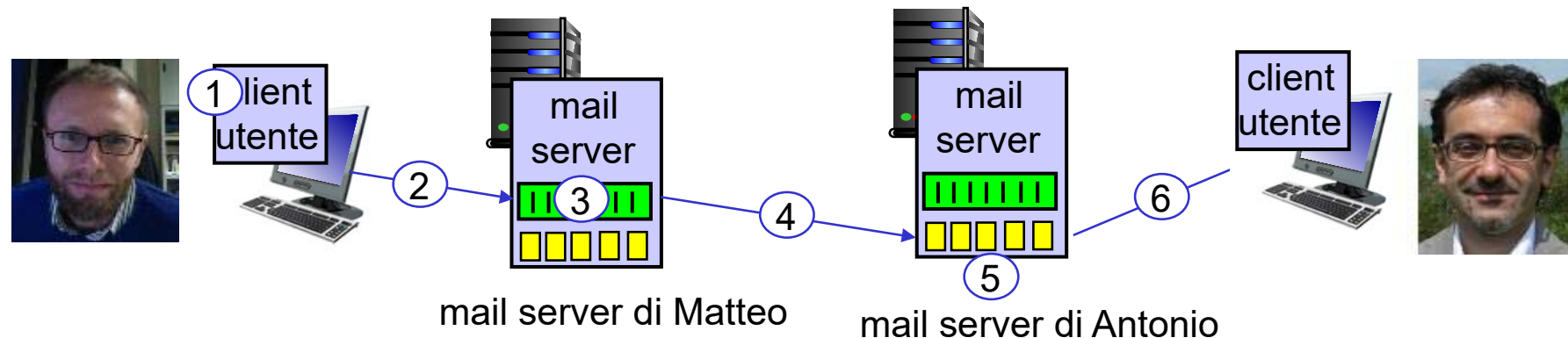
# SMTP

- 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 e 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

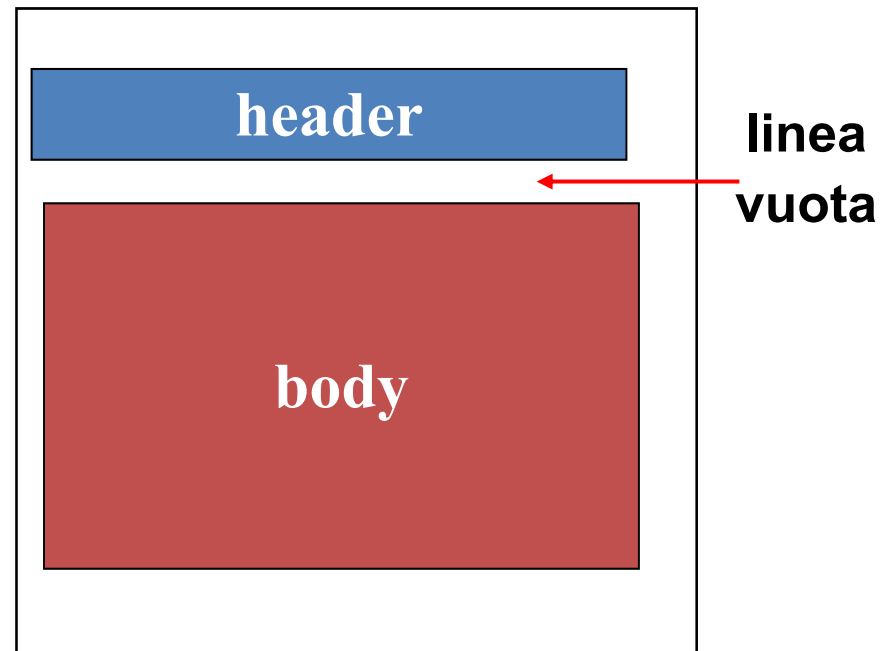
apertura	S: 220 antonimailserver.com
	C: HELO matteomailserver.com
	S: 250 Hello matteomailserver.com, pleased to meet you
	C: MAIL FROM: <matteo@matteomailserver.com>
invio	S: 250 matteo@matteomailserver.com... Sender ok
	C: RCPT TO: <antonio@antonimailserver.com>
	S: 250 antonio@antonimailserver.com ... Recipient ok
	C: DATA
	S: 354 Enter mail, end with "." on a line by itself
	C: Oggi corri al Giuriati?
	C: .
chiusura	S: 250 Message accepted for delivery
	C: QUIT
	S: 221 antonimailserver.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.

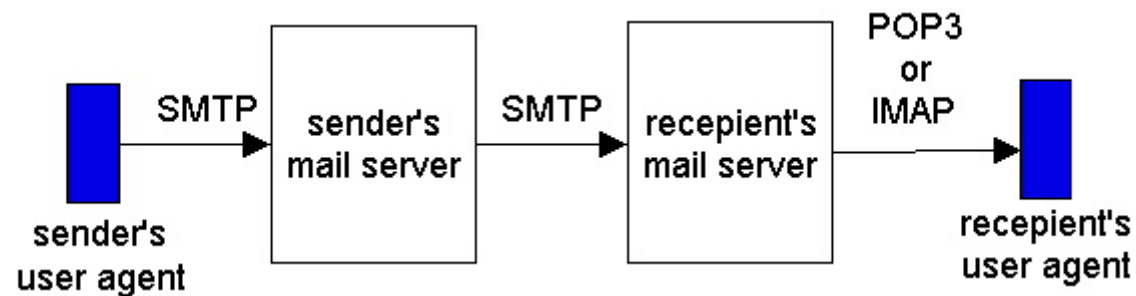
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....

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



# 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**

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

## Fase di transazione, client:

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

```
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
```



# Semplici esperimenti con SMTP

- Visualizzare gli header SMTP
- Proviamo ad inviare una mail tramite *telnet*

telnet smtp.gmail.com 25





**POLITECNICO**  
MILANO 1863



# **Risoluzione di nomi simbolici**

# 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](http://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)



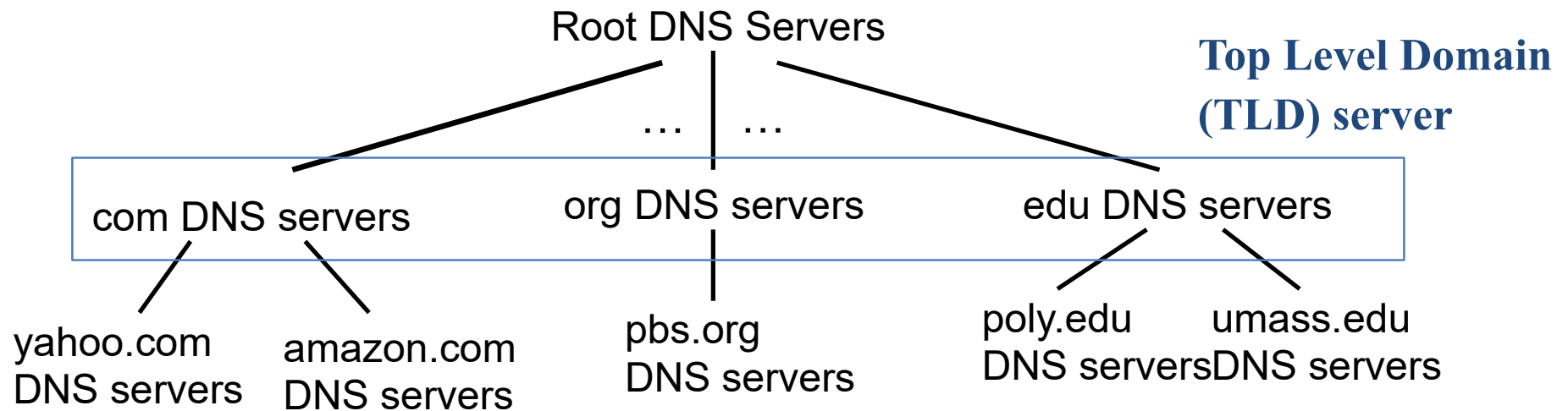


# Domain Name System (DNS)

- 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



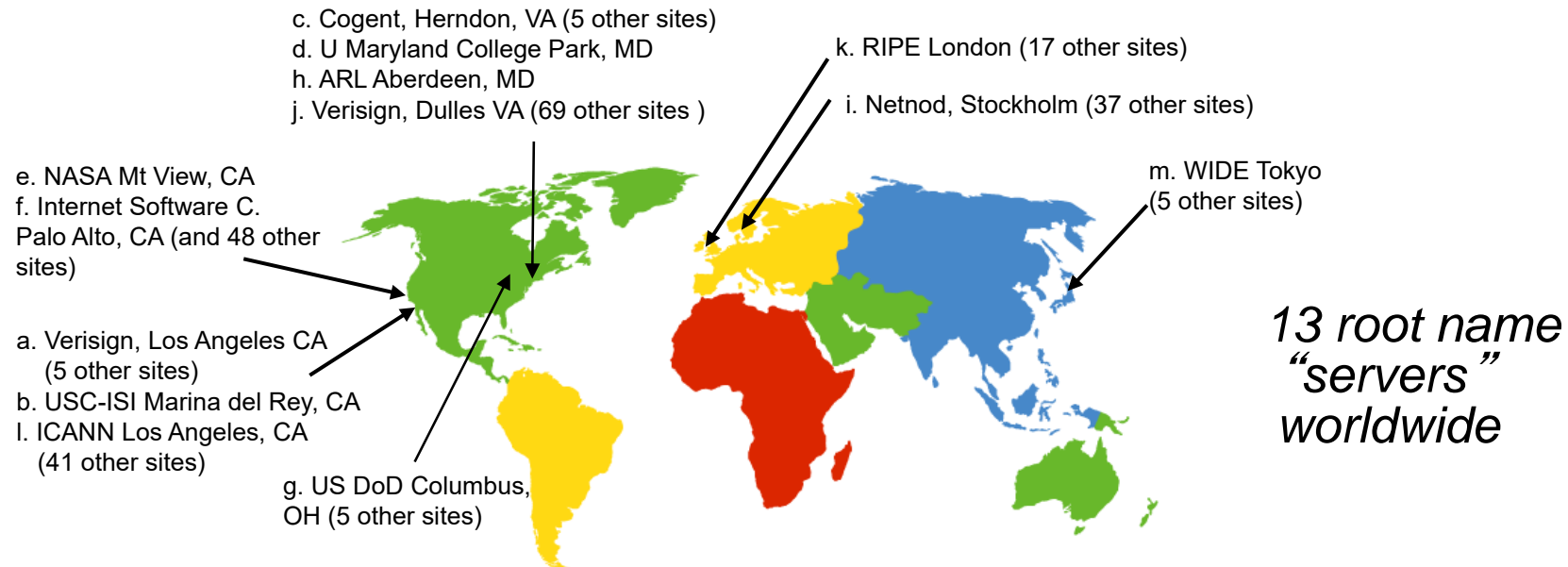
# Database distribuito e gerarchico



- 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 di che gestiscono i domini .com*
  - *I name server .com sanno come trovare i name server che gestisce il dominio google.com*
  - *I name server google.com sanno risolvere il nome simbolico `www.google.com`*



# Root NS



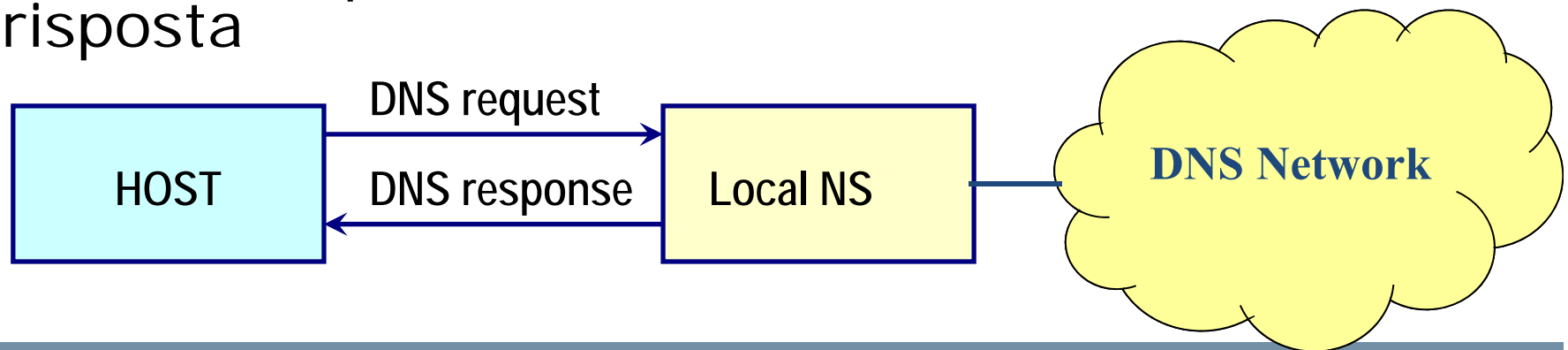
## Altri tipi di NS

- *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*

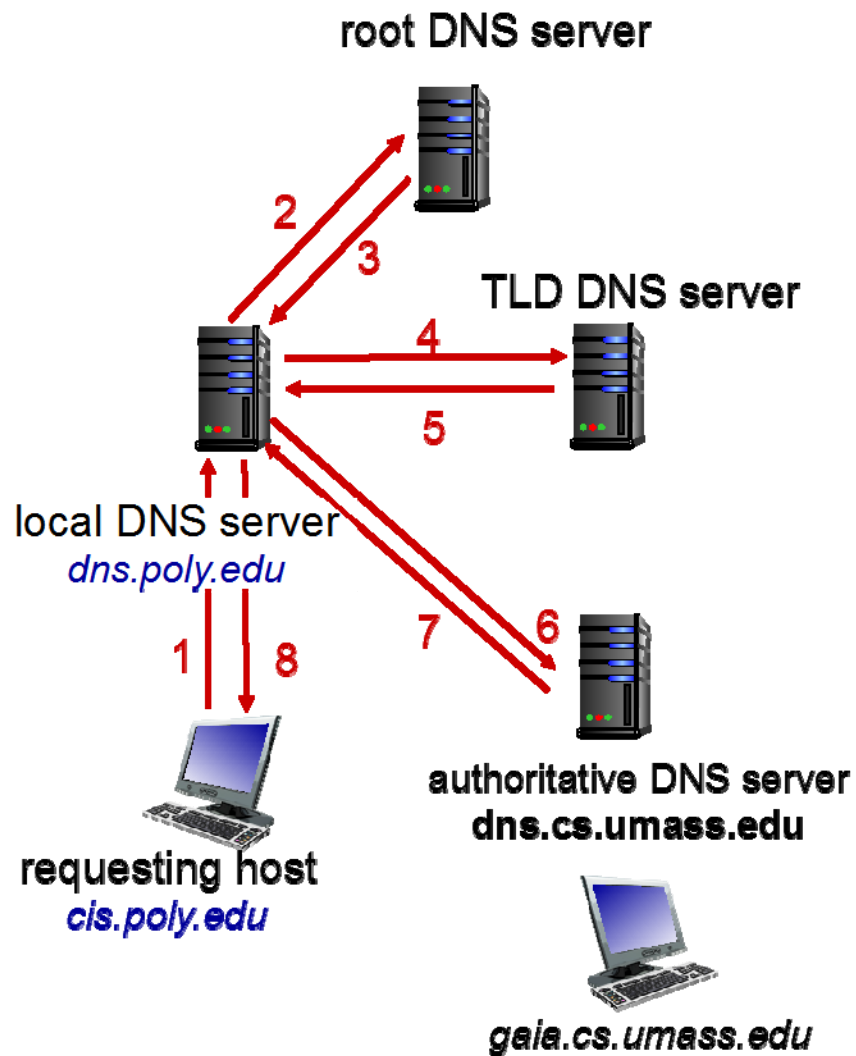


## Come ottenere un mappaggio

- 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

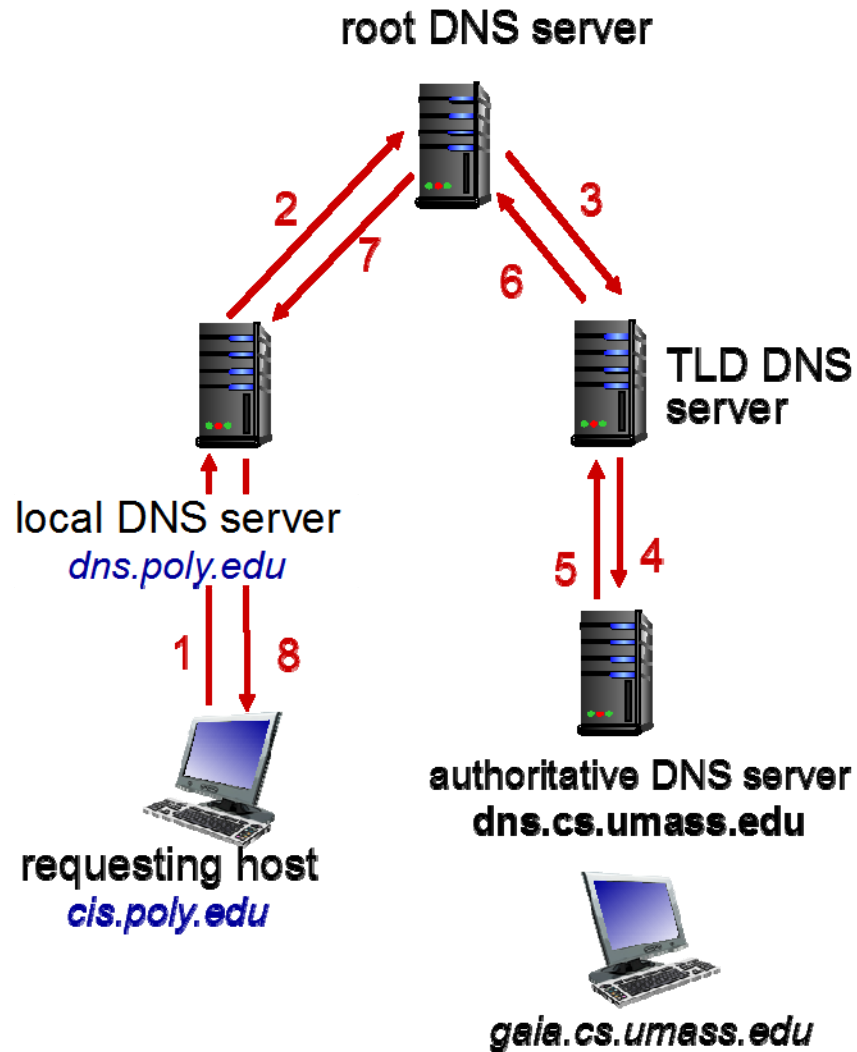


Un *host* presso `cis.poly.edu` vuole risolvere l'indirizzo simbolico `gaia.cs.umass.edu`

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. Il *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: *Name* è il nome di un *host* e *Value* è il suo indirizzo IP  
(morgana.elet.polimi.it, 131.175.21.1, A, TTL)
  - NS: *Name* è un *domain* e *Value* è il nome di un *server* che può ottenere le informazioni relative  
(elet.polimi.it, morgana.elet.polimi.it, NS, TTL)
  - CNAME: *Name* è un nome alternativo (alias) per un *host* il cui nome canonico è in *Value*  
(www.polimi.it, zephyro.rett.polimi.it, CNAME, TTL)
  - MX: *Name* è dominio di *mail* o un alias di mail e *Value* è il nome del *mail server*  
(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 information*: 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, dsn1.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`

**Da soli:** 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](http://www.polimi.it)

Ad esempio, provare:

<http://www.geektools.com/digtool.php>

(non richiede di installare il programma)



# Semplici esperimenti con DNS - dig

```
MacBook-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



## Attività proposta – DNS e Wireshark

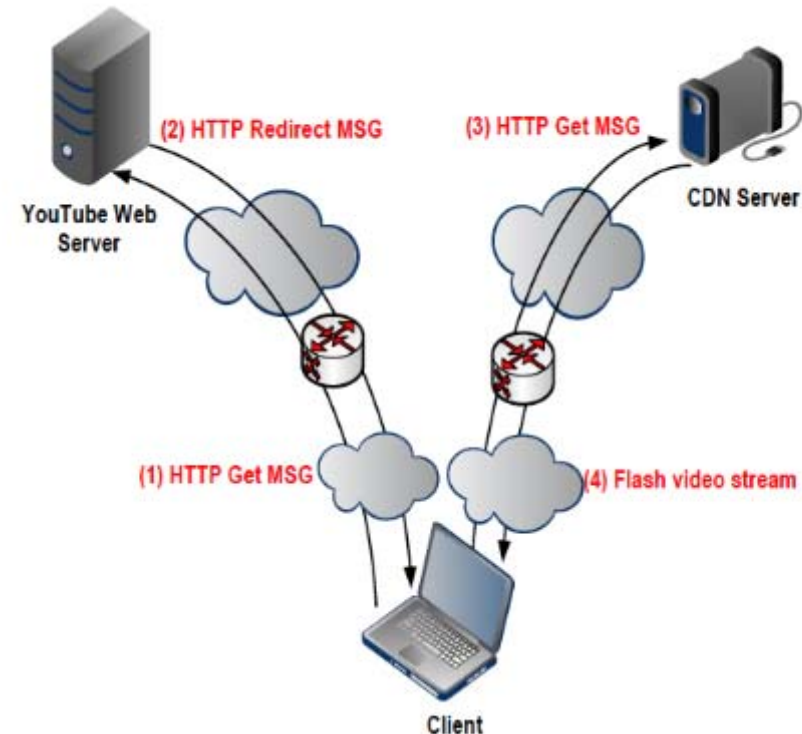
- Cattura con Wireshark di una sessione DNS (o apri la traccia fornita)
  - usa ipconfig (windows) o ifconfig (Linux) per svuotare la cache DNS sul vostro dispositivo
  - Apri il *browser* e svuota la *cache* del *browser*
  - Apri *Wireshark* ed imposta un filtro sul tuo indirizzo IP `ip.addr==vostroIP` (in questo modo catturate solo il traffico da/per il vostro dispositivo)
  - Inizia una cattura *Wireshark*
  - Visita una pagina *web* qualsiasi
  - Termina la cattura
- Trova ed esamina le richieste/risposte DNS rispondendo alle seguenti domande:
  - quale è la porta di destinazione dei messaggi di richiesta DNS?
  - che tipo di richiesta DNS viene effettuata?
  - quante risposte sono contenute nel messaggio di risposta?





# HTTP e video streaming: YouTube

- Architettura di *YouTube* si basa su:
  - HTTP per la distribuzione di contenuti video
  - Una (o più) *Content Distribution Network (CDN)* per lo *storage* distribuito dei contenuti video
- Da soli: analizzate con *Wireshark* una sessione *YouTube* verificando lo scambio di messaggi in figura (funziona??)



Esempio: provare  
<https://www.youtube.com/watch?v=w1f6o1HQBvg>





**POLITECNICO**  
MILANO 1863



# Applicazioni Peer-to-Peer

# P2P file sharing

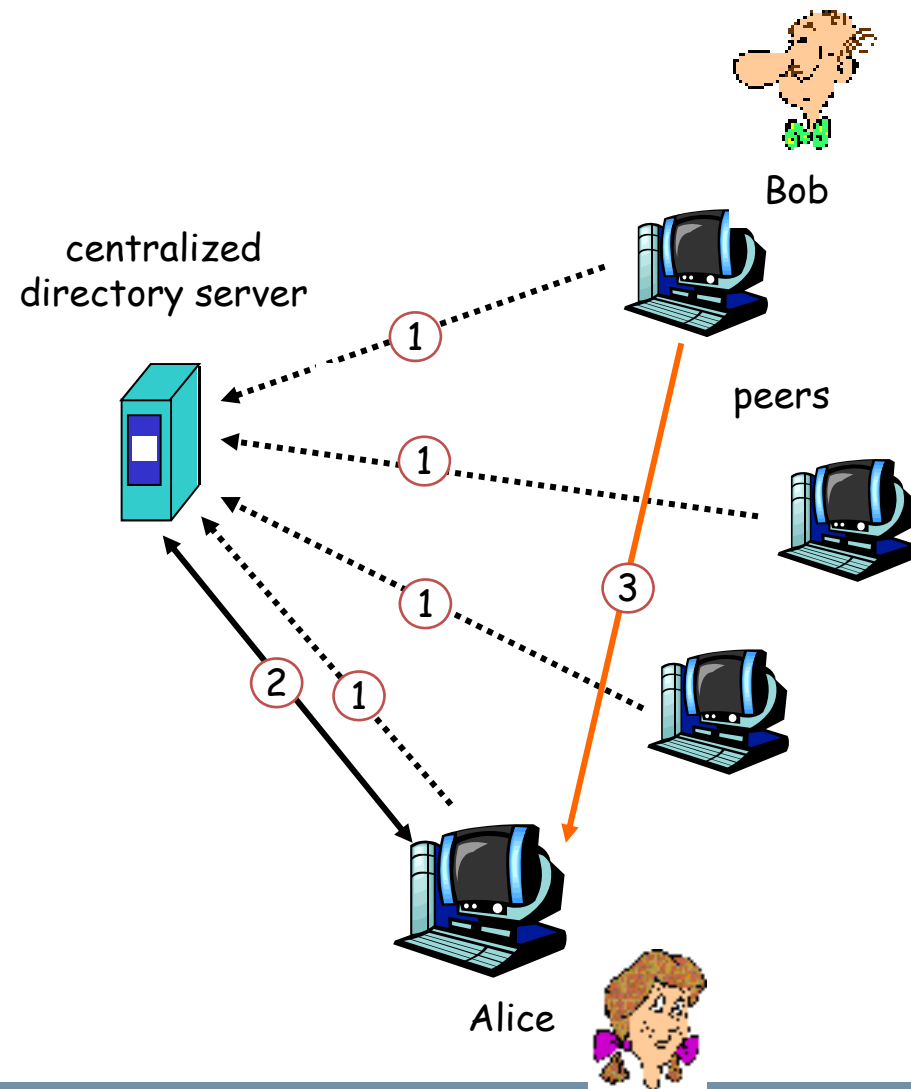
- Gli utenti utilizzano il software P2P sul proprio PC
  - Si collegano in modo intermittente a Internet prendendo indirizzi IP diversi ogni volta
  - Se un utente cerca un file l'applicazione trova altri utenti che lo hanno
  - L'utente sceglie da chi scaricarlo
  - Il file è scaricato usando un protocollo 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

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

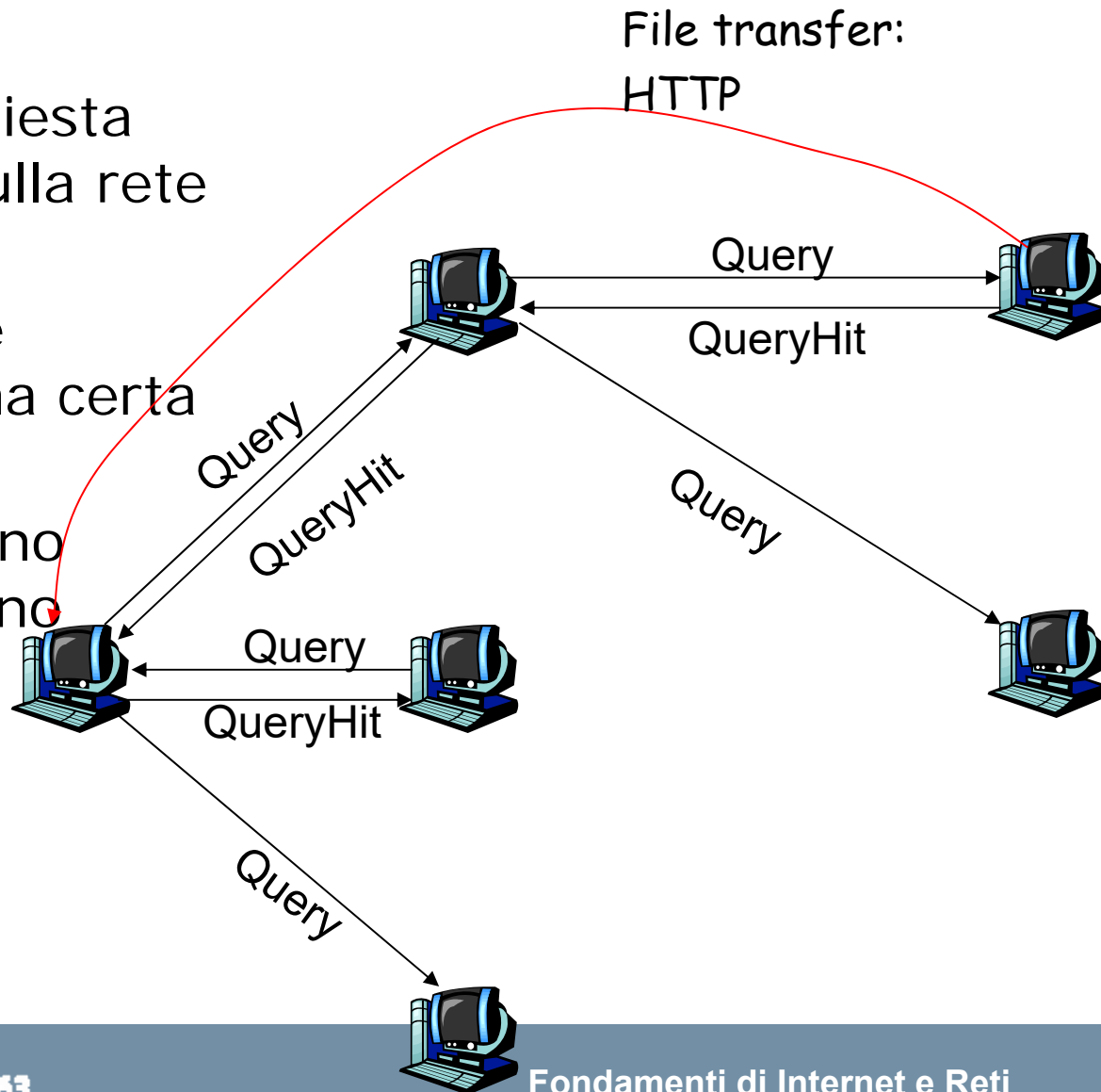
- I *peer* si attivano e si collegano ad un numero ( $<10$ ) di altri vicini
- La ricerca dei vicini è distribuita
- I vicini nella rete *overlay* possono essere fisicamente distanti



# 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*





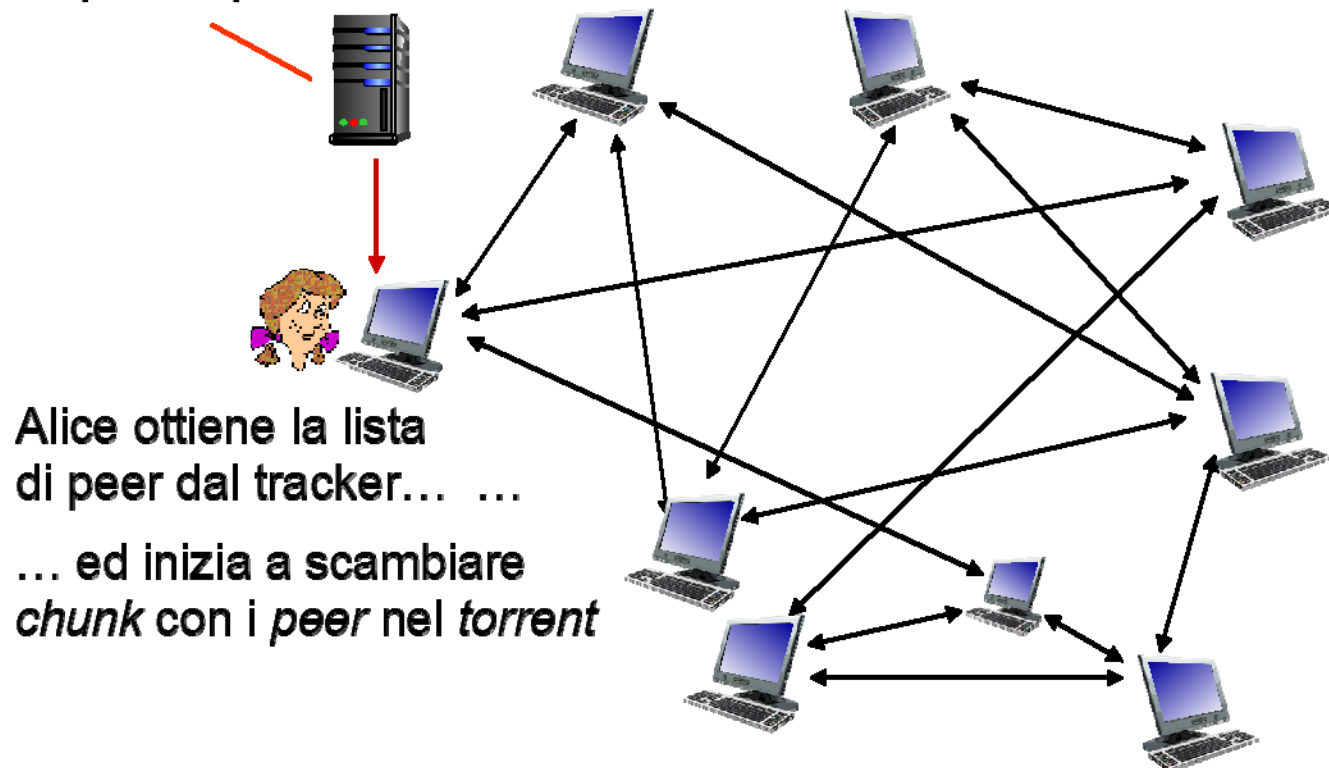
# BitTorrent

Oltre al metodo basato sul tracker, esiste anche un altro meccanismo alternativo basato su Distributed Hash Table (DHT)

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

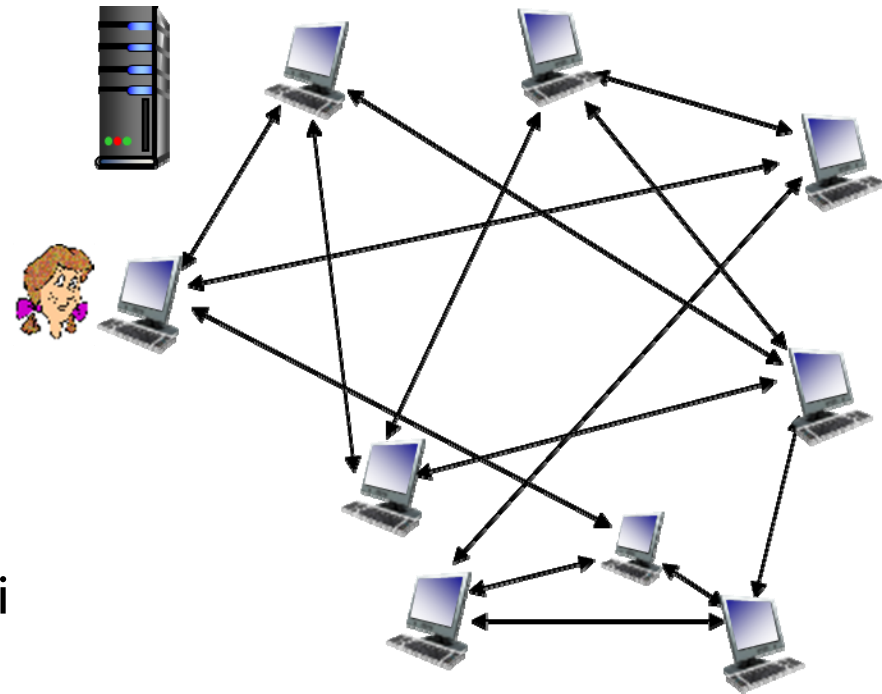
**tracker:** tiene traccia dei peer che partecipano ad un torrent

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



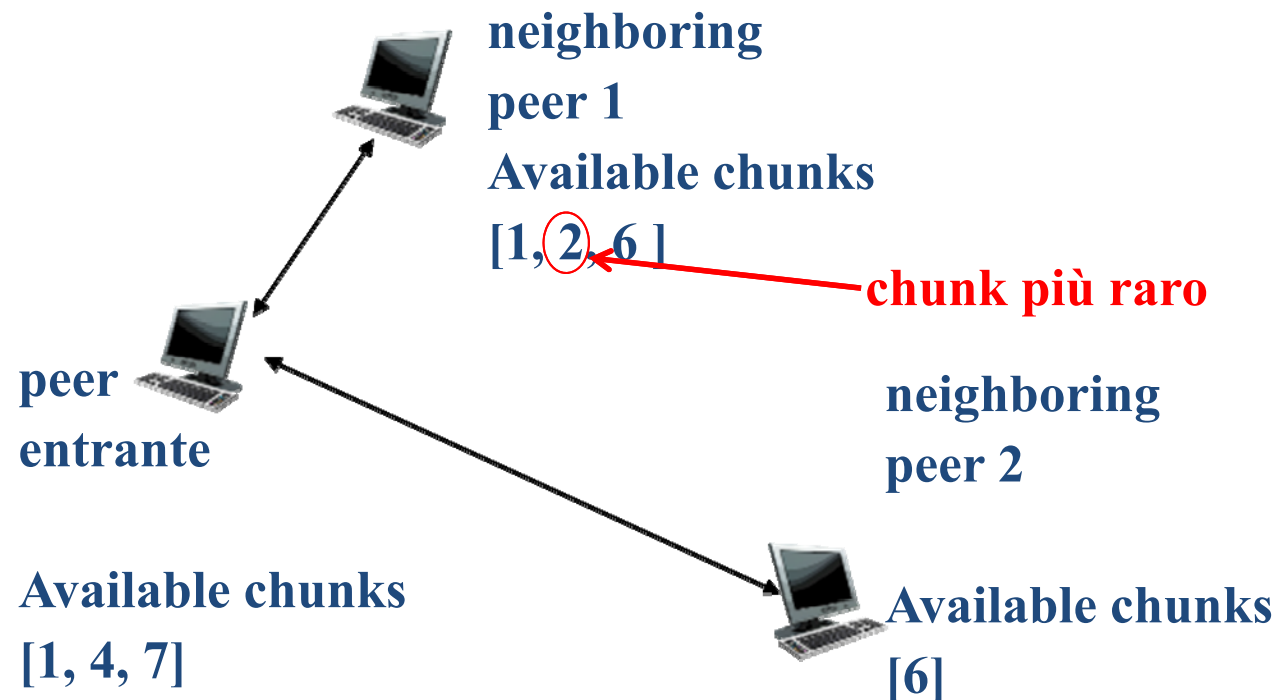
# BitTorrent – entrare nel *torrent*

- 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 sottoinsieme 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*)

