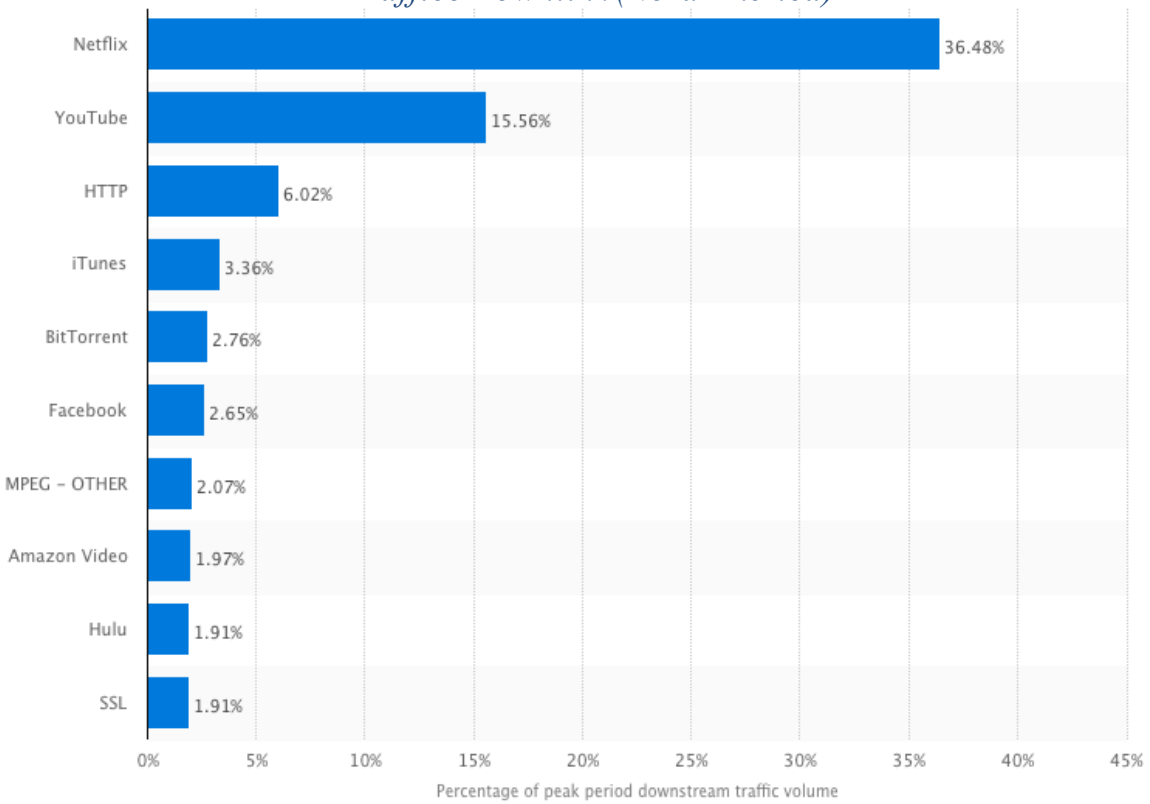
Il Livello Applicativo

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

Traffico Downlink (Nord America)

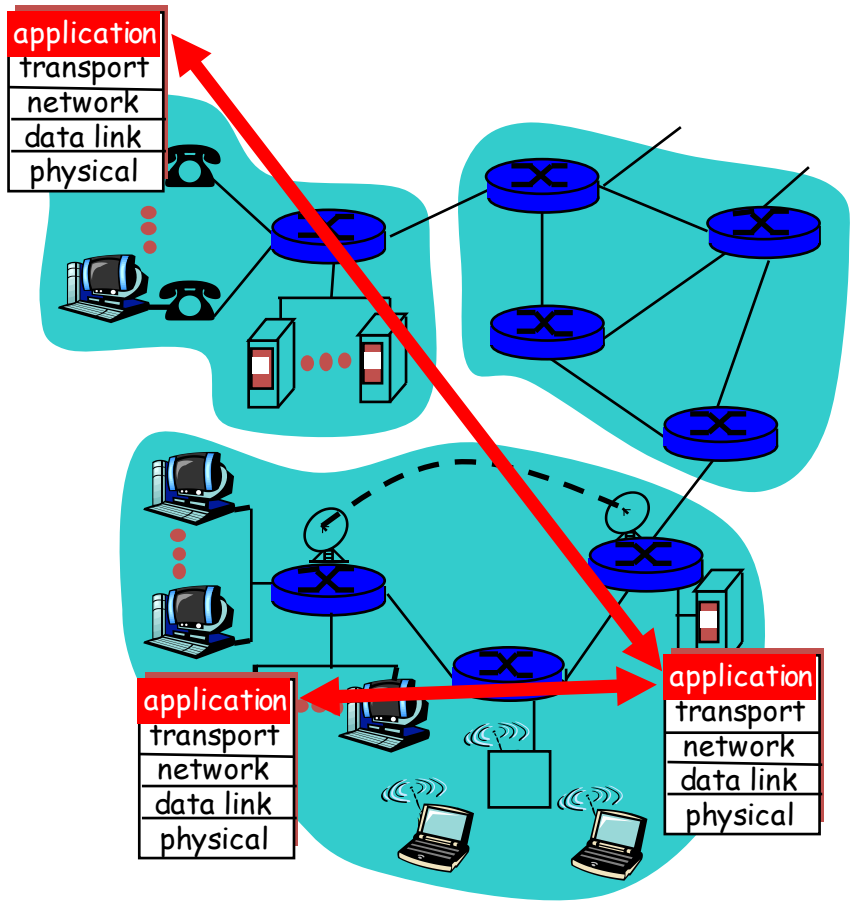


Source: www.statista.com (2016)

Creare un'applicazione di rete

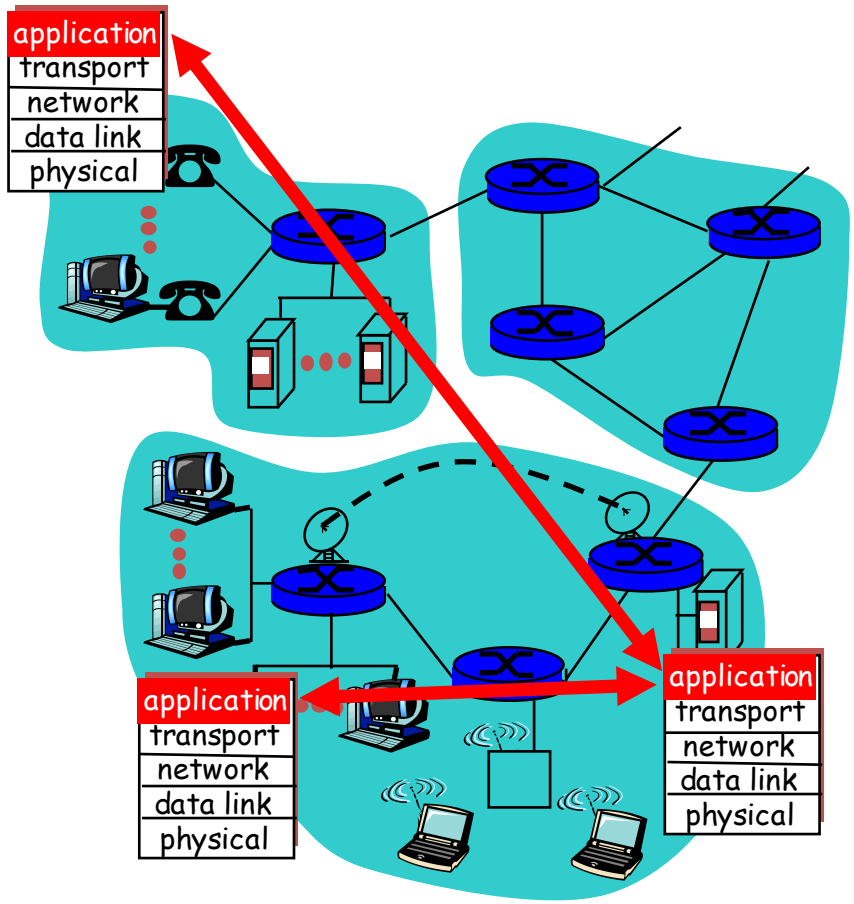
- **Scrivere un software** che:
  - possa essere eseguito su diversi terminali e
  - possa comunicare tramite la rete

- **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, 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!)

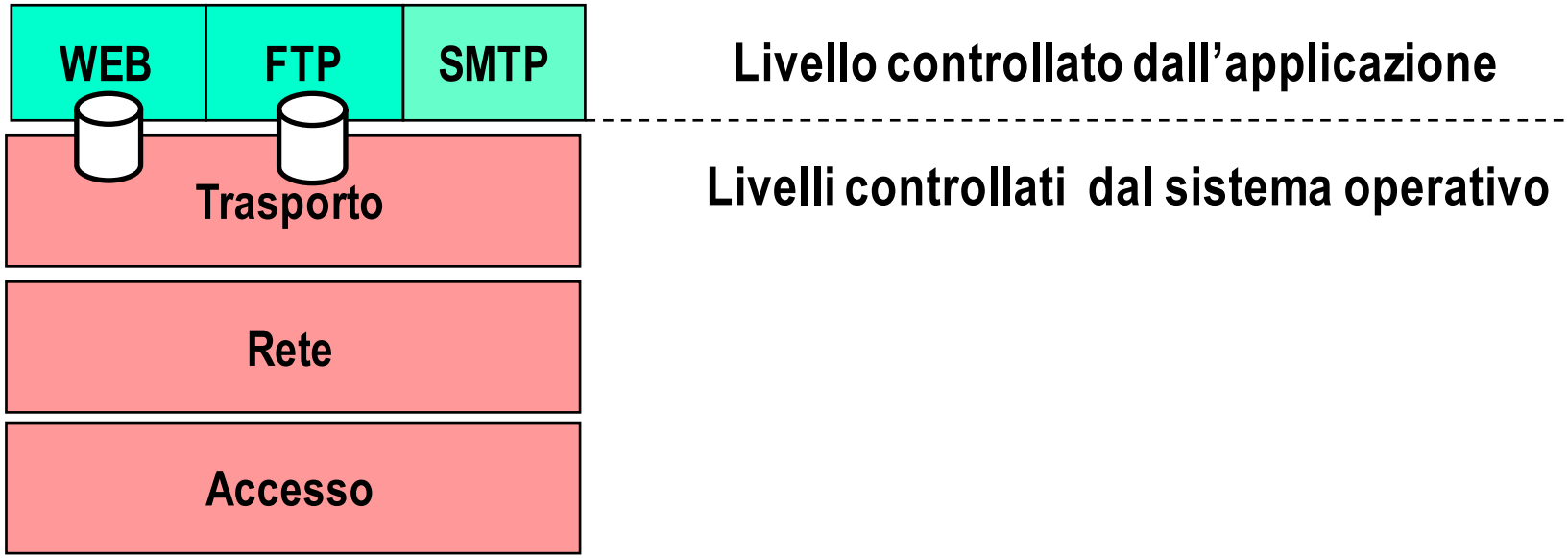
Ingredienti per la comunicazione tra processi remoti

- 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 messagi: Significato dei campi
  - Regole su come e quando inviare e ricevere i messaggi

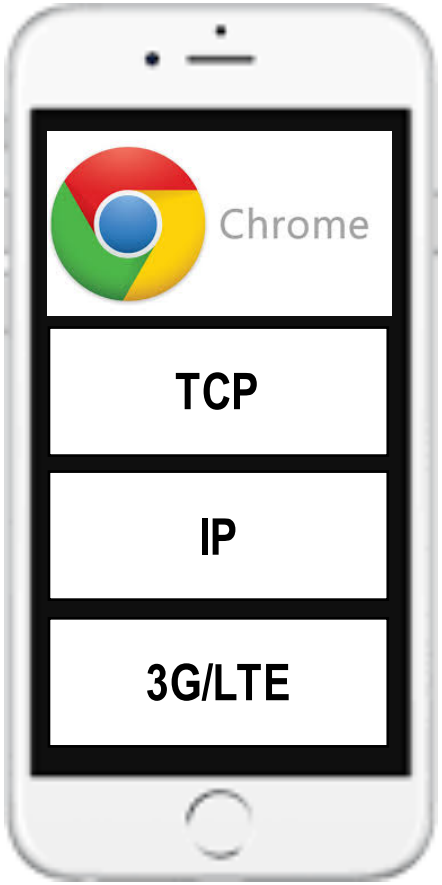
Indirizzamento di processi applicativi

- 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



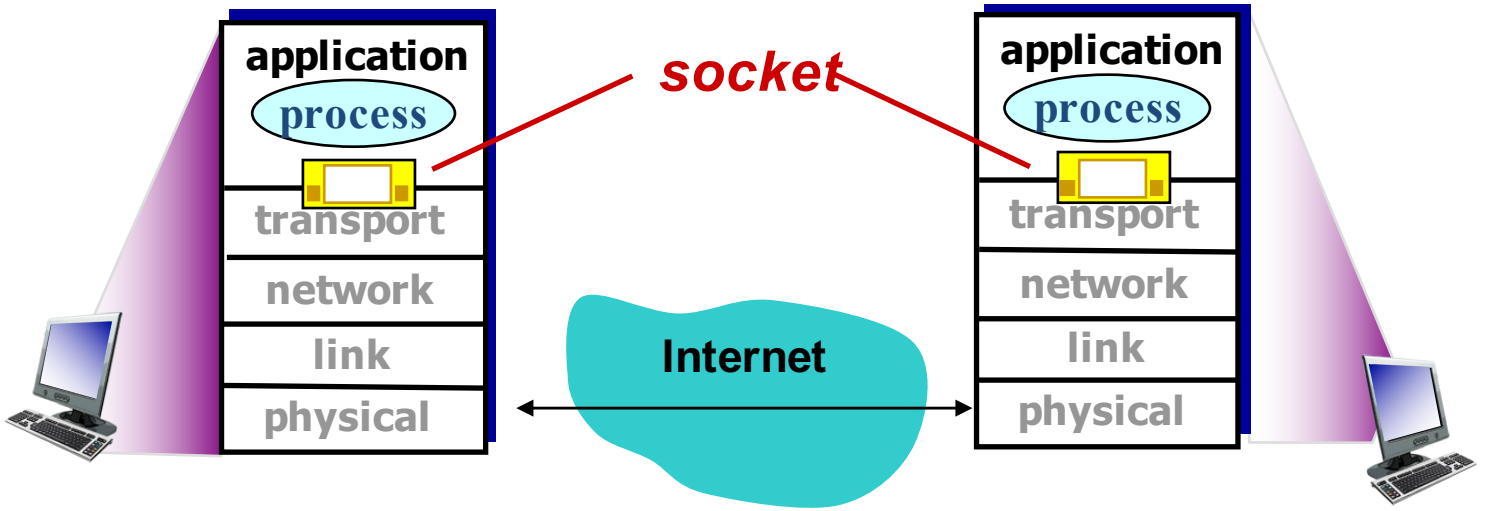
Indirizzamento di processi applicativi

- 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 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 delle 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”

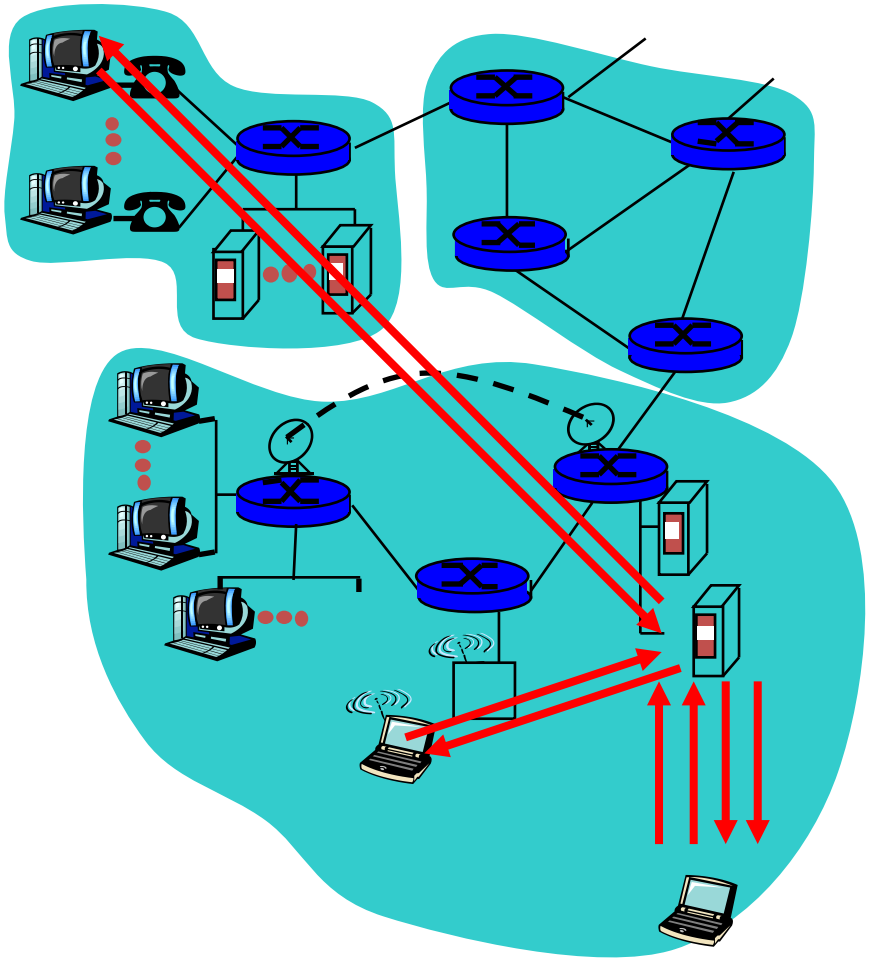
Applicazioni e protocolli di trasporto

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* *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:

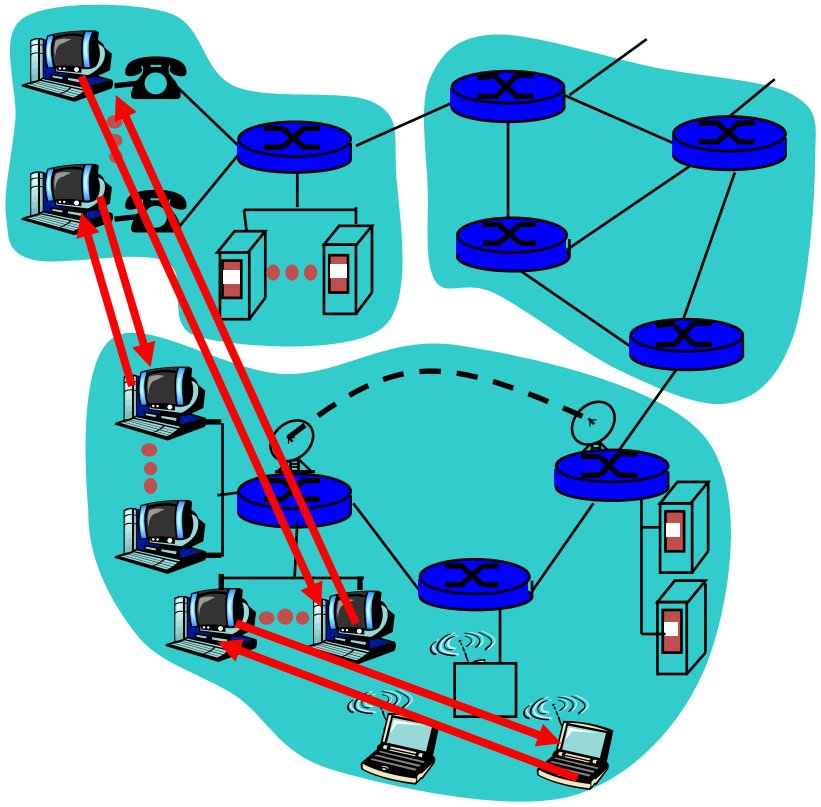
- *Host* sempre attivo
- Indirizzo IP permanente
- Possibilità di utilizzo di macchine in *cluster*
- Possono ricevere richieste da molti *client*

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

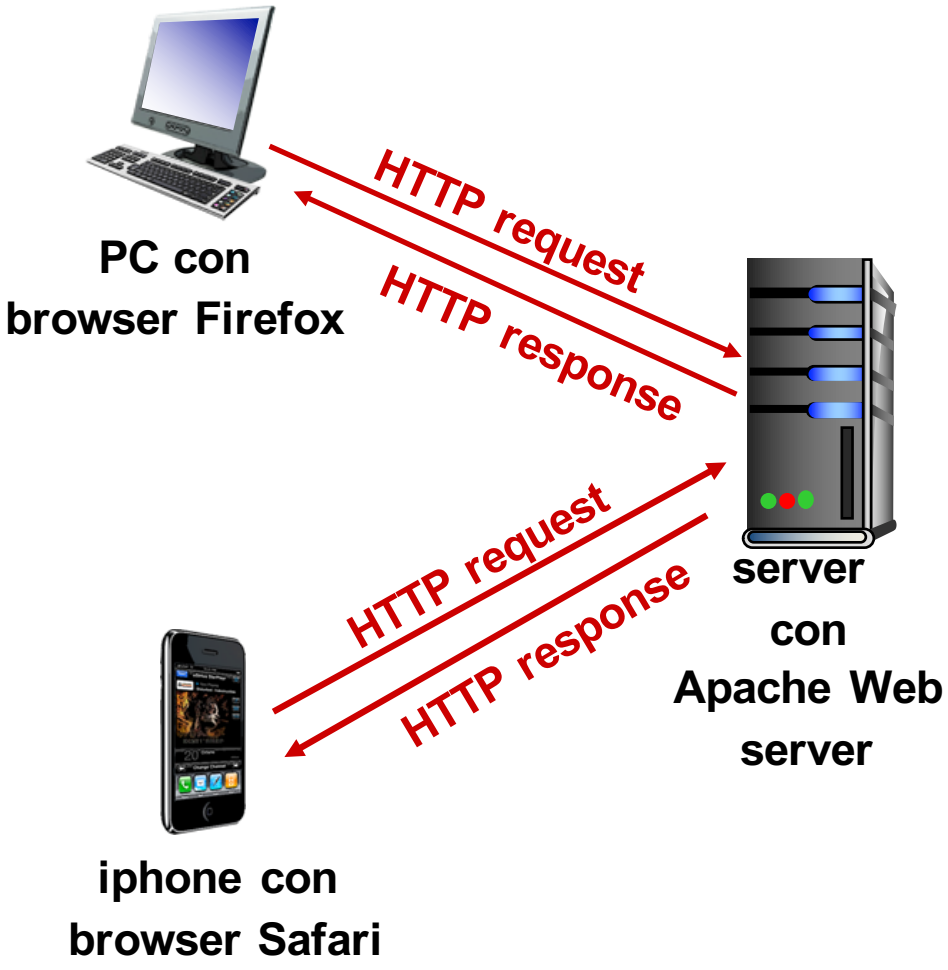
Hyper Text Transfer Protocol (HTTP)
RFC 1945, 2616

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:

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

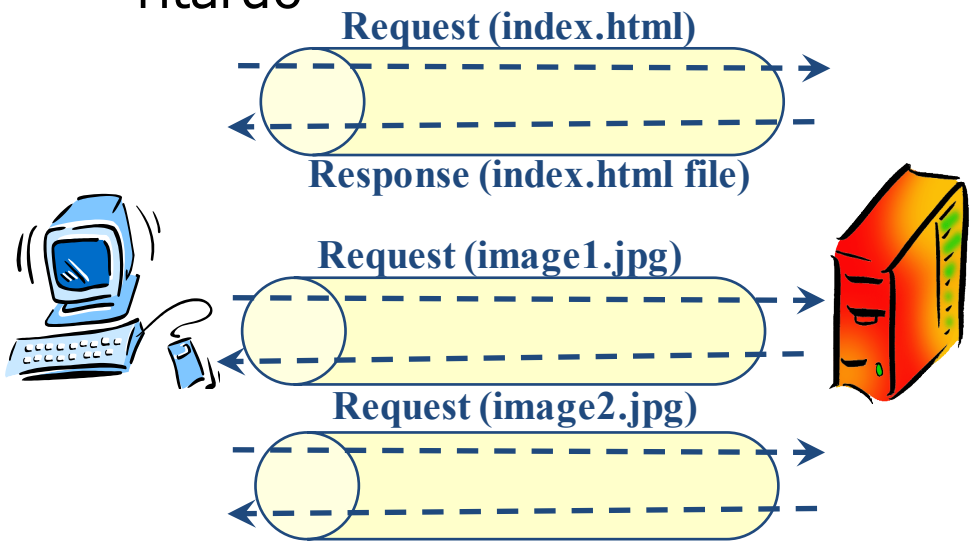
- HTTP si appoggia su TCP a livello di trasporto:
  1. Il client HTTP inizia una connessione TCP verso il server (porta 80)
  2. Il server HTTP accetta connessioni TCP da client HTTP
  3. ***Client* e *Server* HTTP si scambiano informazioni (pagine web e messaggi di controllo)**
  4. La connessione TCP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

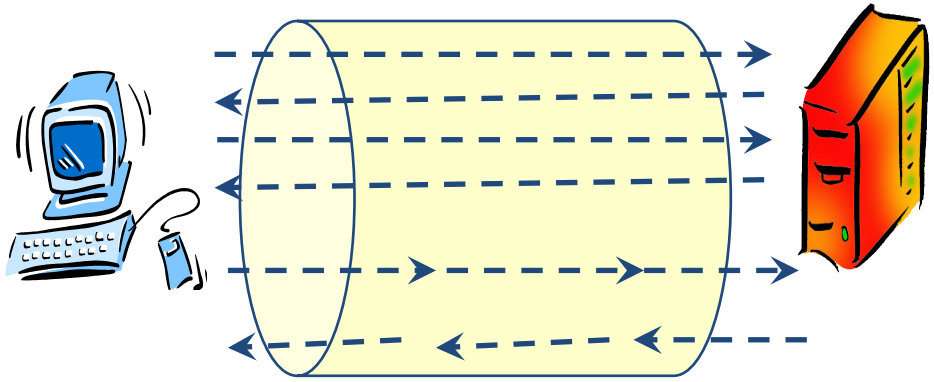
Connessione non persistente

- 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`

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

1a. I client HTTP inizia una connessione TCP verso il server HTTP www.polimi.it sulla porta 80

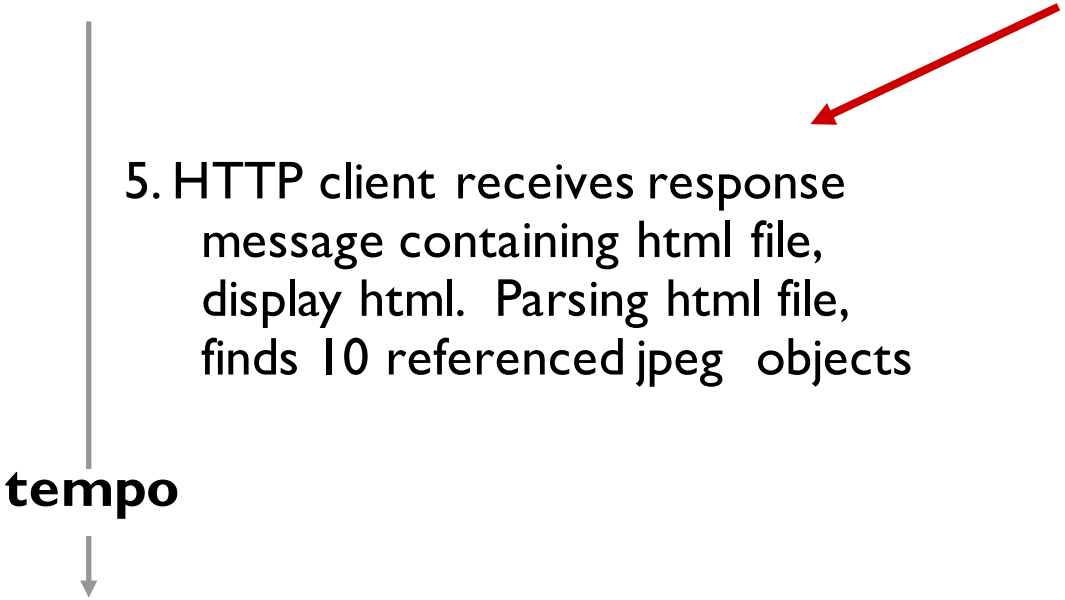
1b. il server HTTP in esecuzione su www.polimi.it in attesa sulla porta 80 accetta la connessione e notifica il client

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

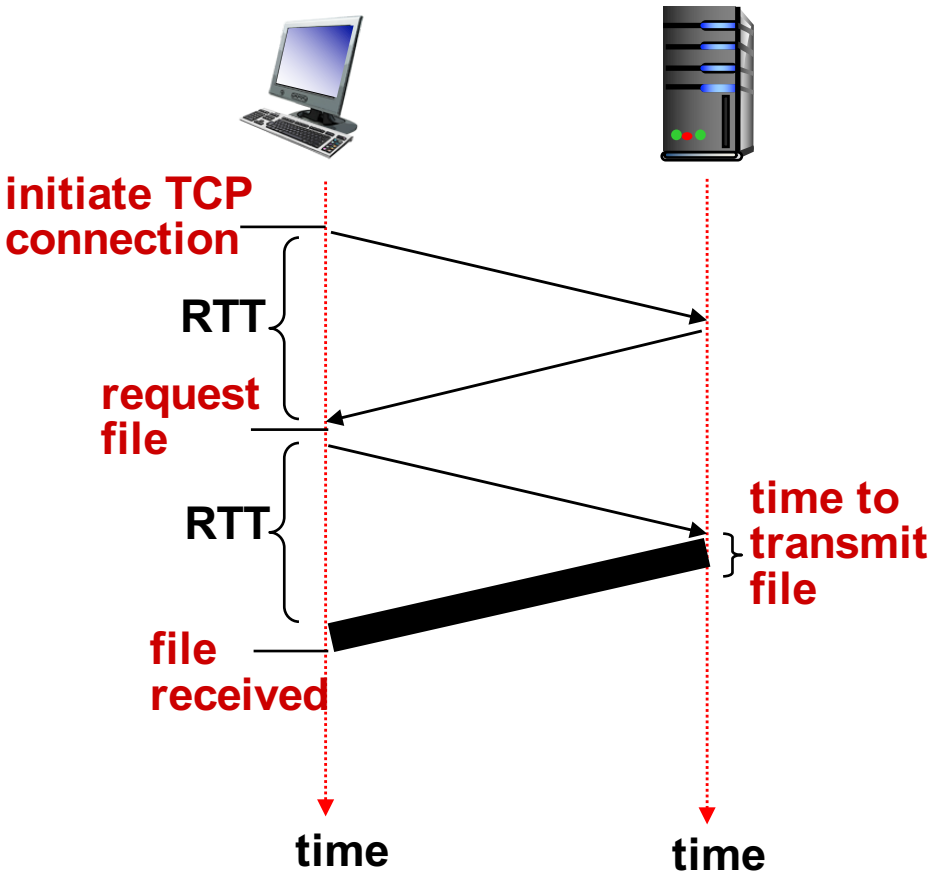
Stima del tempo di trasferimento in HTTP

- *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:

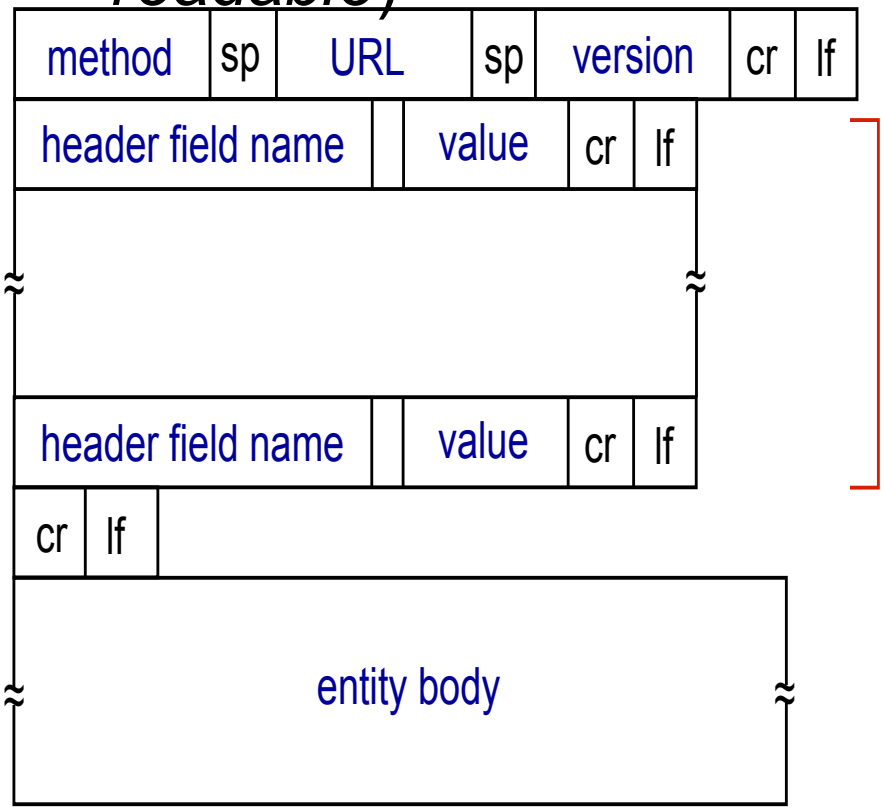
$$T_{nonpers} = \sum_{i=0}^{10}(2RTT + T_i)$$

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



Le richieste HTTP

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



request line

header lines

body

```
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

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

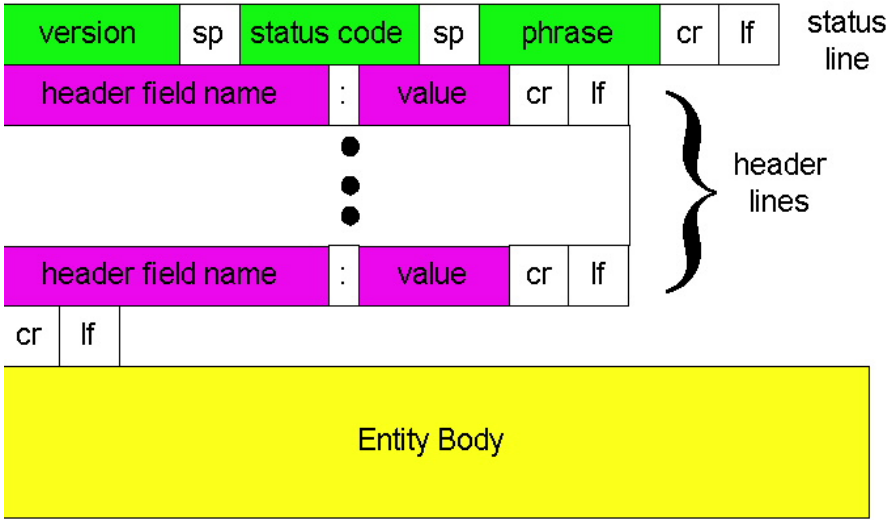
Esempi di *Header* HTTP

- 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




```
TTP/1.1 200 OK\r\n
ate: Sun, 26 Sep 2010 20:09:20 GMT\r\n
erver: Apache/2.0.52 (CentOS)\r\n
ast-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
Tag: "17dc6-a5c-bf716880"\r\n
ccept-Ranges: bytes\r\n
ontent-Length: 2652\r\n
eep-Alive: timeout=10, max=100\r\n
onnection: Keep-Alive\r\n
ontent-Type: text/html; charset=ISO-8859-1\r\n
r\n
ata data data data data ...
```

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

HTTP è testuale (ASCII)

- 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
```

- Risposta

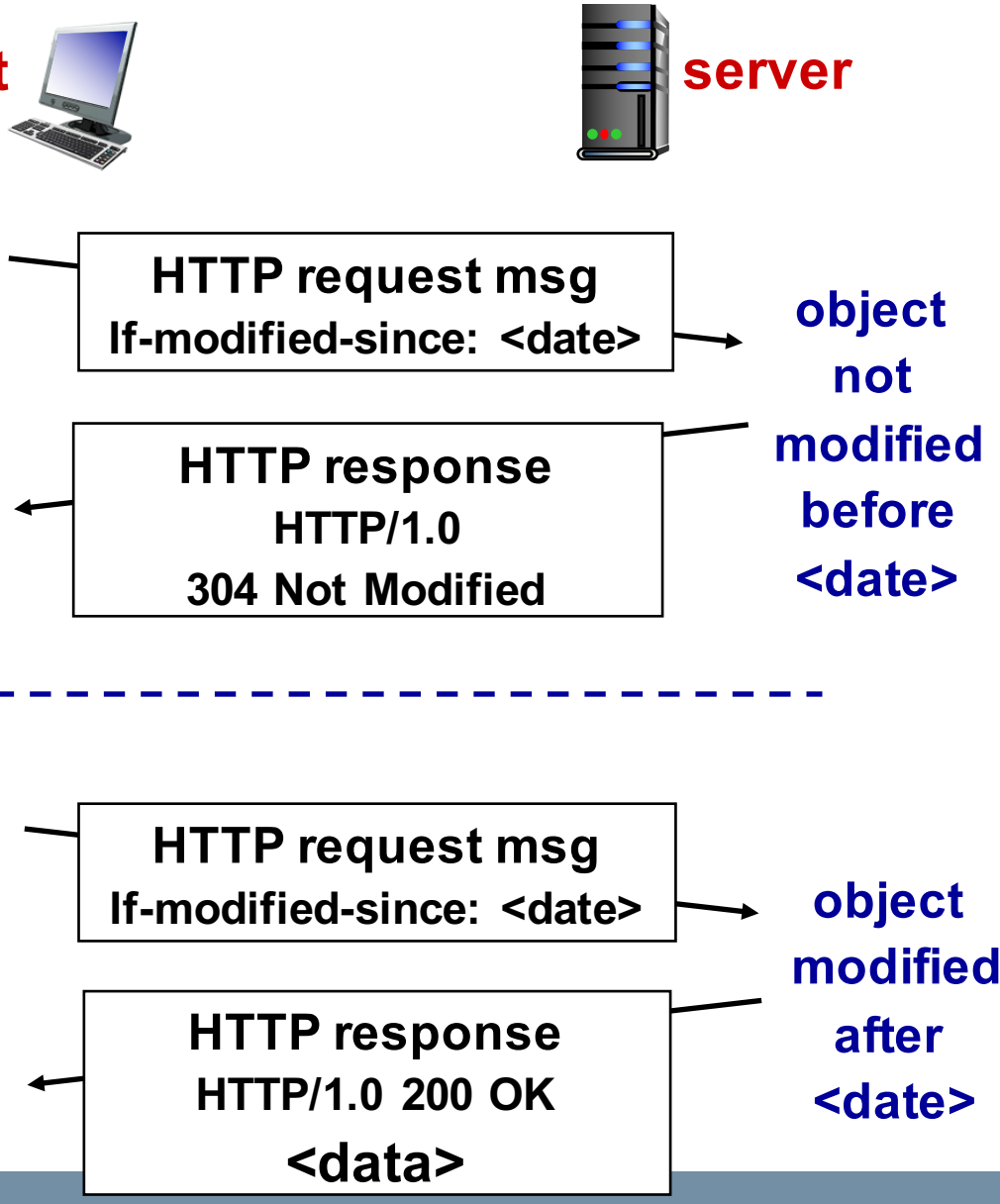
```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
data data data data data ...
```

Conditional GET

- Obiettivo: non inviare un 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
```

Apre una connessione TCP verso la porta 80 del server www.antlab.polimi.it

- Inviamo una richiesta HTTP (proviamo GET e HEAD)

```
GET /people/matteo-cesana HTTP/1.1
Host: www.antlab.polimi.it
```

Richiede l'oggetto `/people/matteo-cesana`

- Cosa riceviamo?

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
```

Mantenere uno "stato" in HTTP: i *cookies*

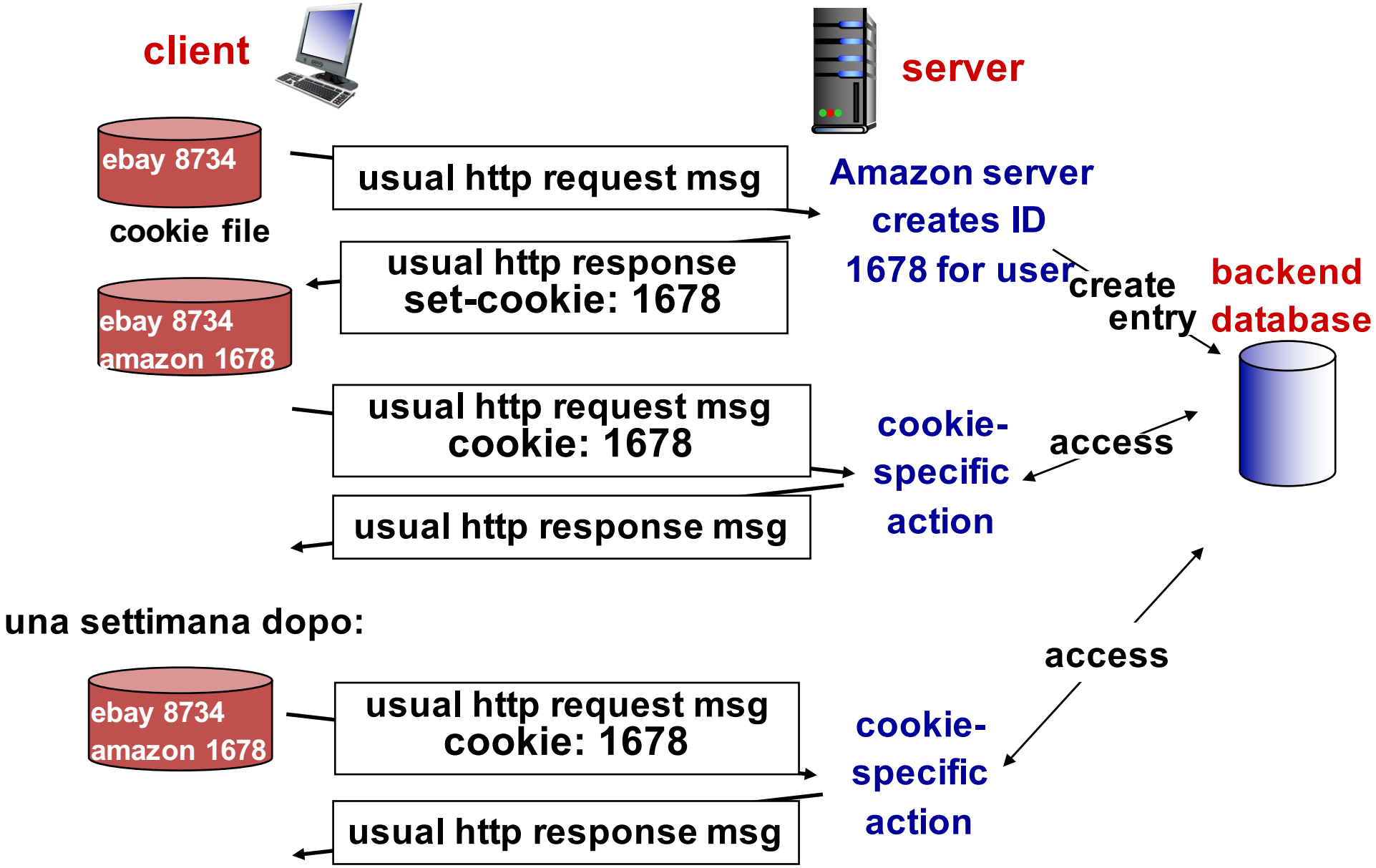
Ingredienti dei cookies:

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:

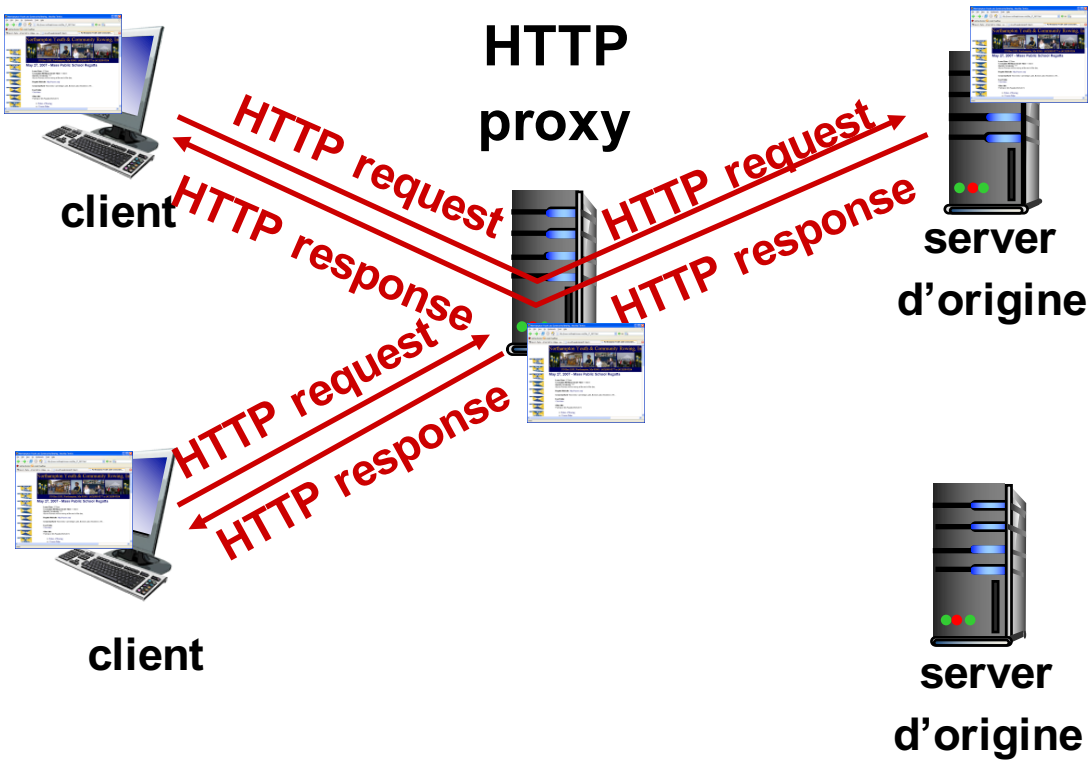
- Matteo visita un sito di e-commerce per la prima volta
- Quando il sito riceve la prima richiesta HTTP, il sito genera:
  - un ID unico
  - una entry nel data base locale che corrisponde all'ID creato

Esempio di uso dei *cookie*



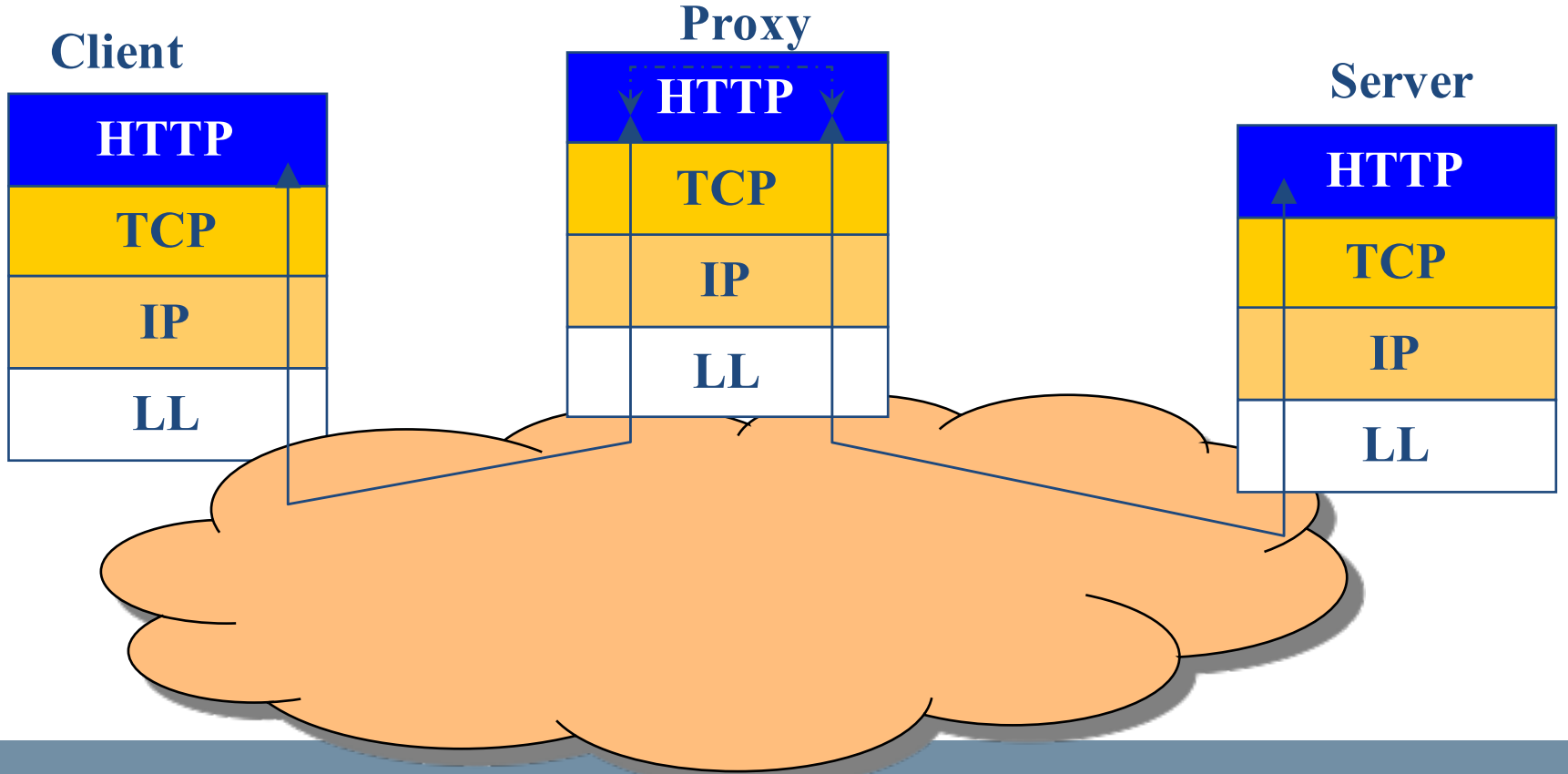
I *proxy* HTTP: *Cache* di rete

- 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

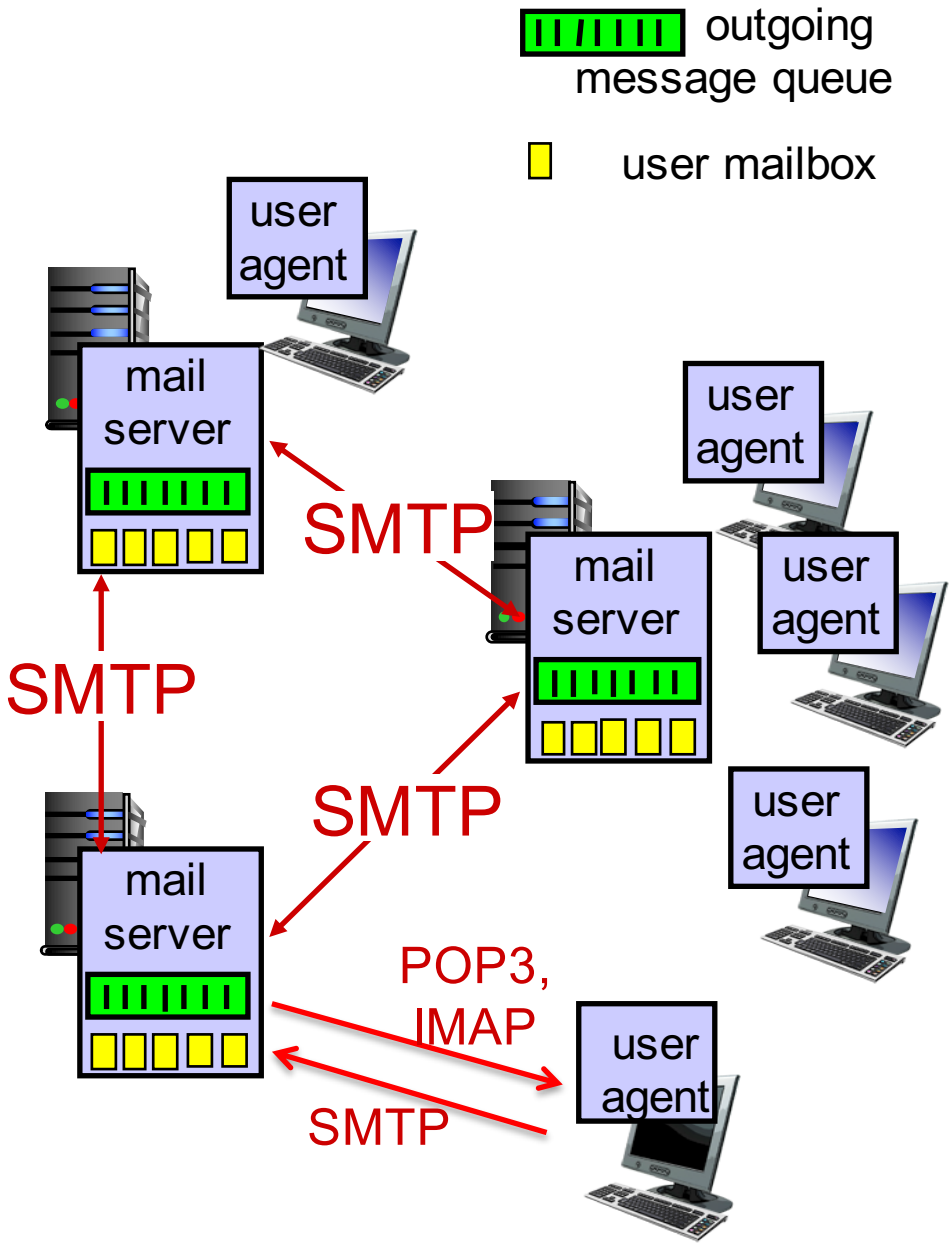
- Accediamo alla pagina www.google.com
- Verifichiamo a quale indirizzo IP corrisponde il nome www.google.com (come possiamo fare?)
- Impostiamo ora il *proxy* HTTP del Politecnico
- Accediamo alla pagina www.google.com
- Verifichiamo tramite che le richieste HTTP vengono "servite" dal proxy del Politecnico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

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

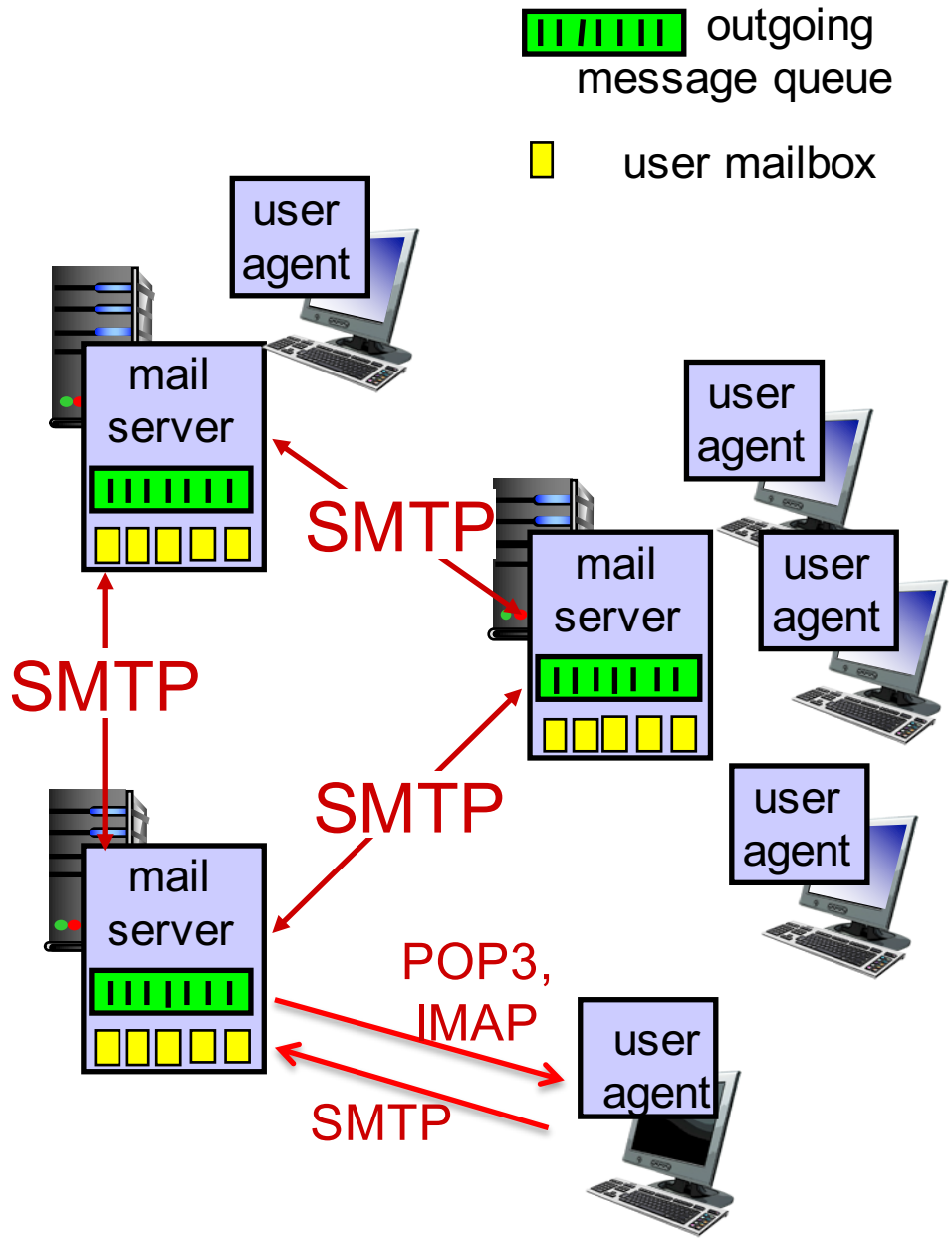
Il servizio di E-mail

- **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

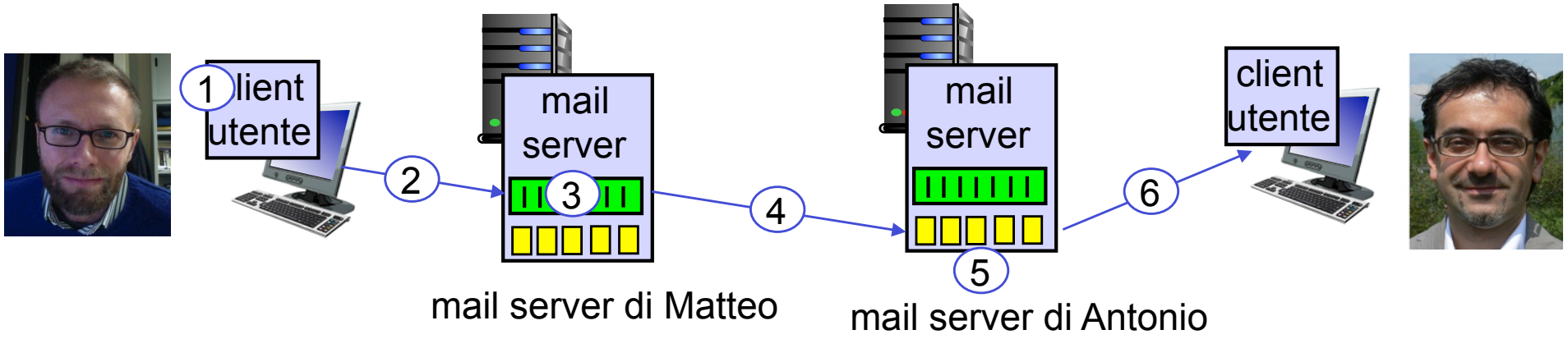


SMTP

- 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
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>
```

invio

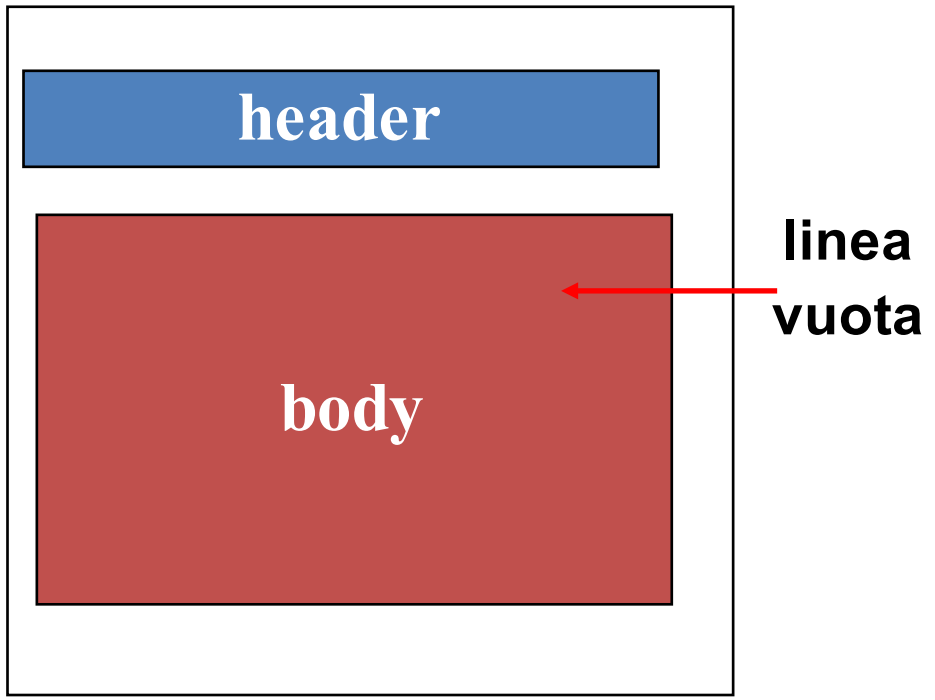
```
S: 250 matteo@matteomailserver.com... Sender ok
C: RCPT TO: <antonio@antoniomailserver.com>
S: 250 antonio@antoniomailserver.com ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Oggi corri al Giuriati?
C: .
S: 250 Message accepted for delivery
```

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:

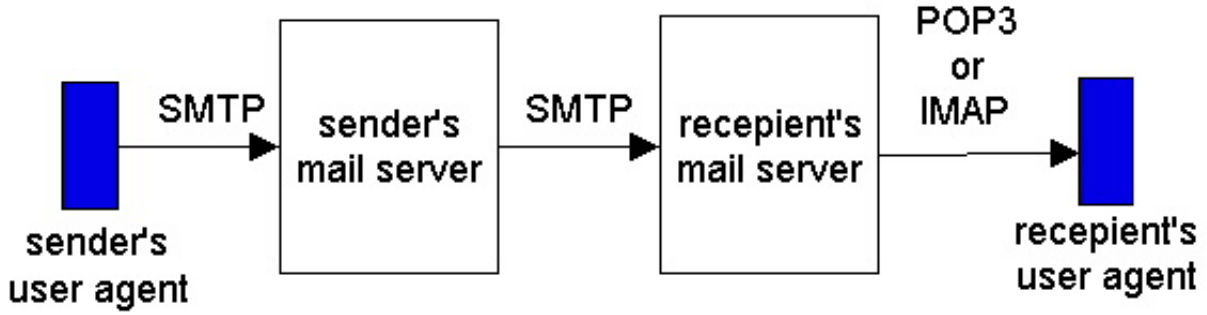
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe with commentary
MIME-Version: 1.0
Content-Type: multipart/mixed; Boundary=StartOfNextPart
--StartOfNextPart
Dear Bob,
Please find a picture of an absolutely scrumptious crepe.

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

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

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
```

Semplici esperimenti con SMTP

- Visualizzare gli header SMTP

- Proviamo ad inviare una mail tramite *telnet*

```
telnet smtp.gmail.com 25
```

Risoluzione di nomi simbolici

DNS

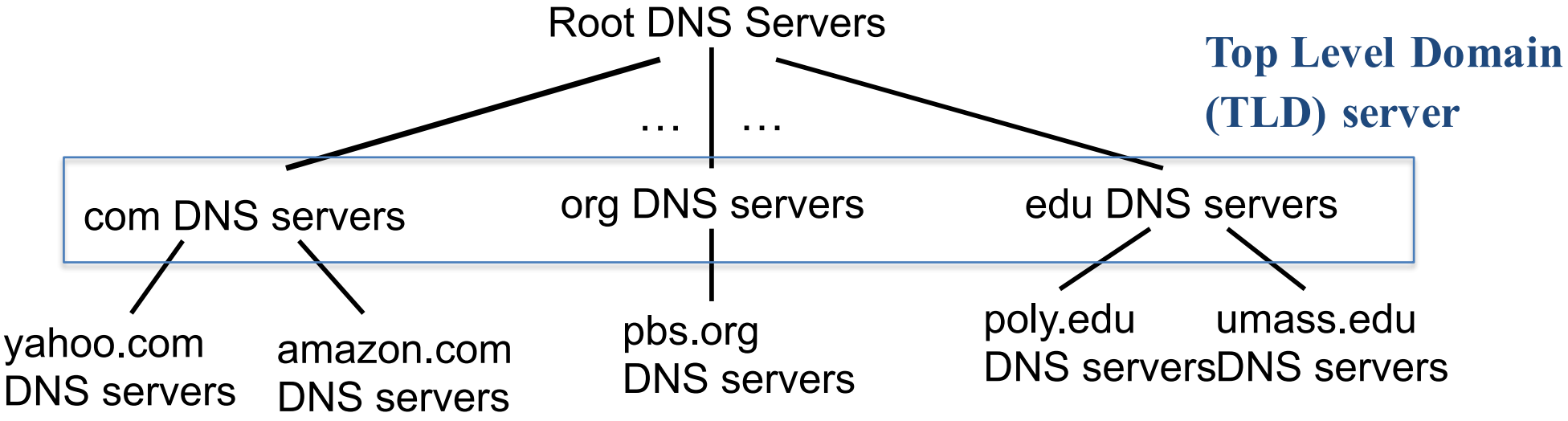
Domain Name System (DNS)

- 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)

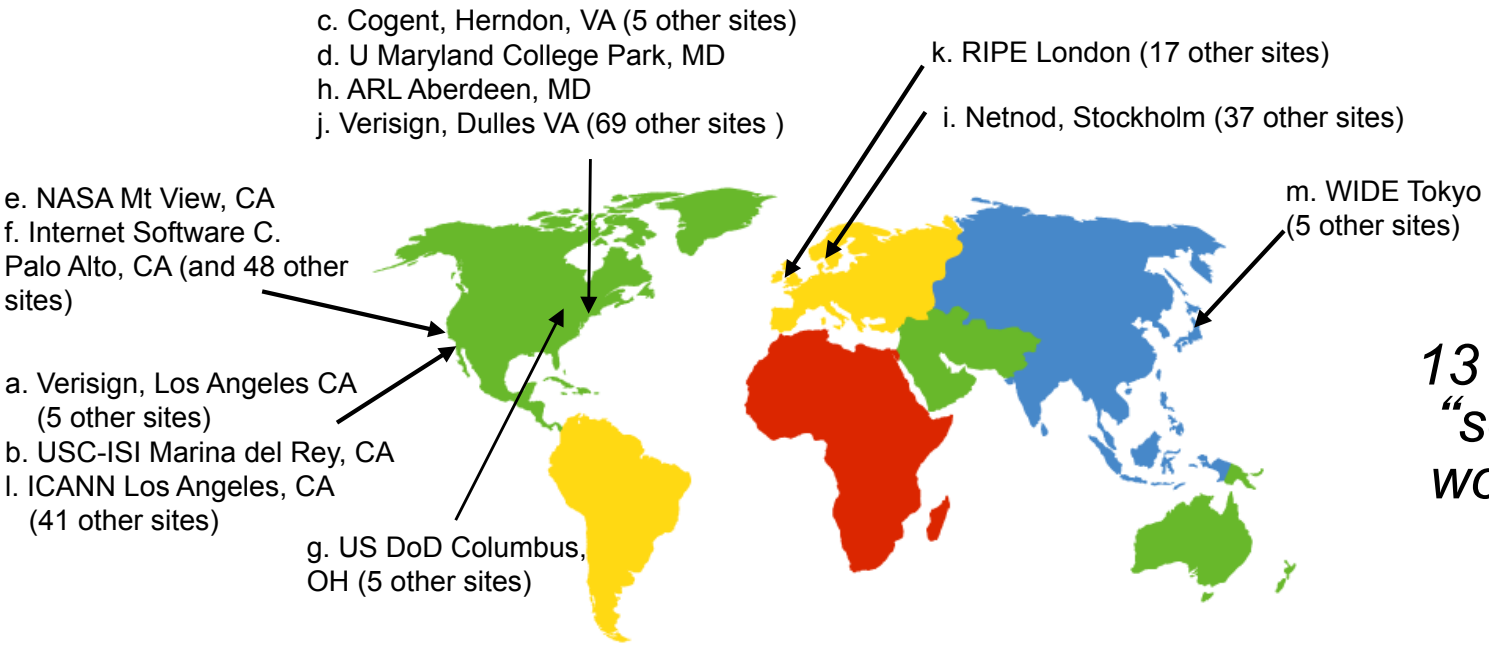
- Ingredienti
  - Database distribuito costituito da molti *name servers* con organizzazione gerarchica
  - Protocollo applicativo basato su UDP tra *name server* e *host* per risolvere nomi simbolici (tradurre nomi simbolici in indirizzi IP)
- Servizi aggiuntivi
  - host aliasing
  - Mail server aliasing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



13 root name "servers" worldwide

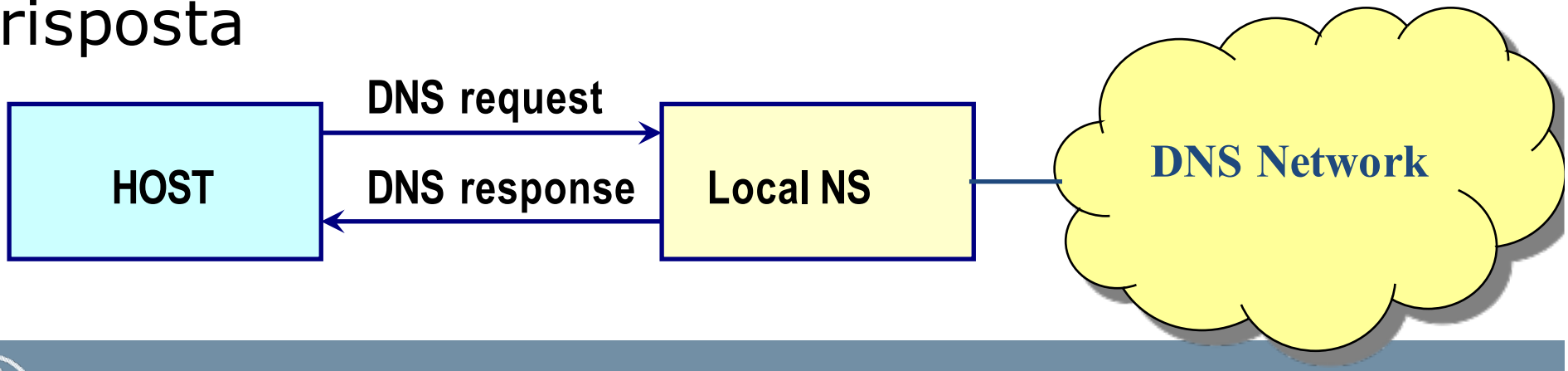
Altri tipi di NS

- *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

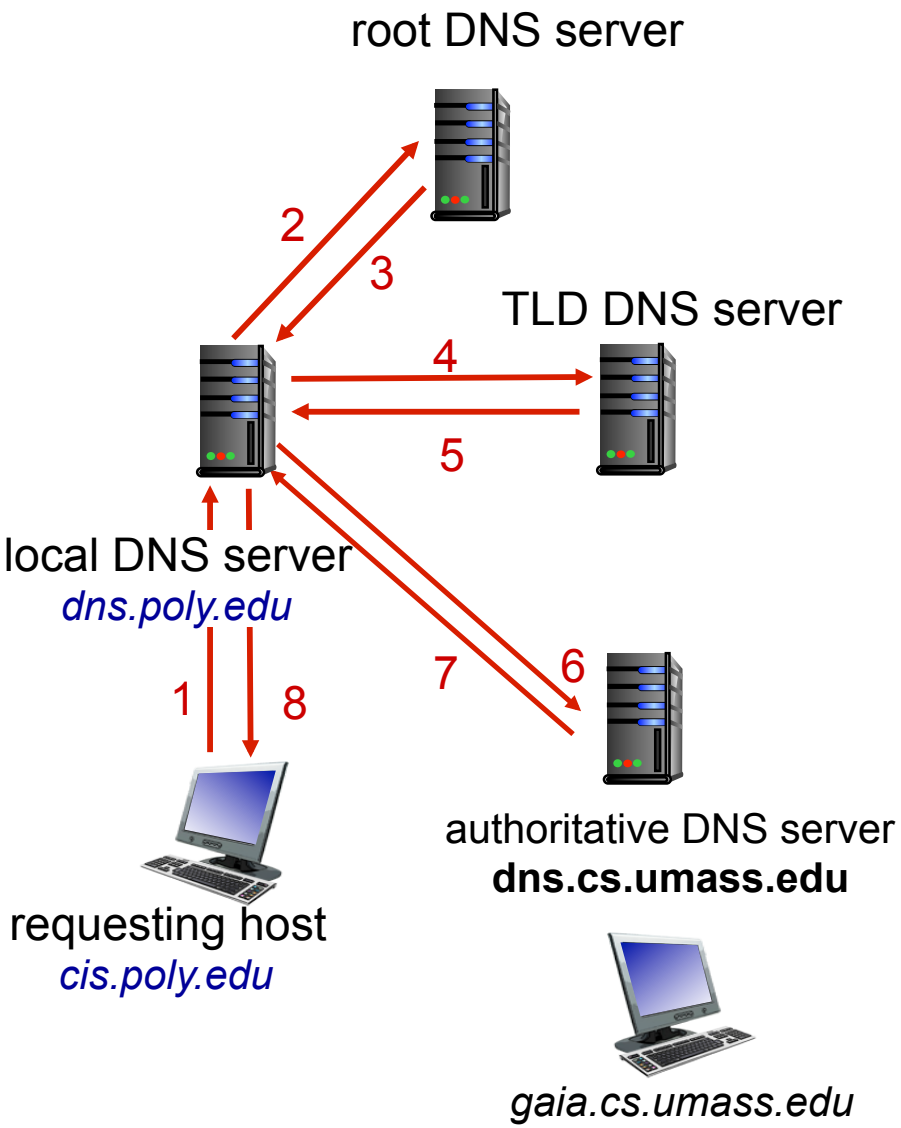
- *Authoritative Name Servers*
  - NS "responsabile" di un particolare *hostname*

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



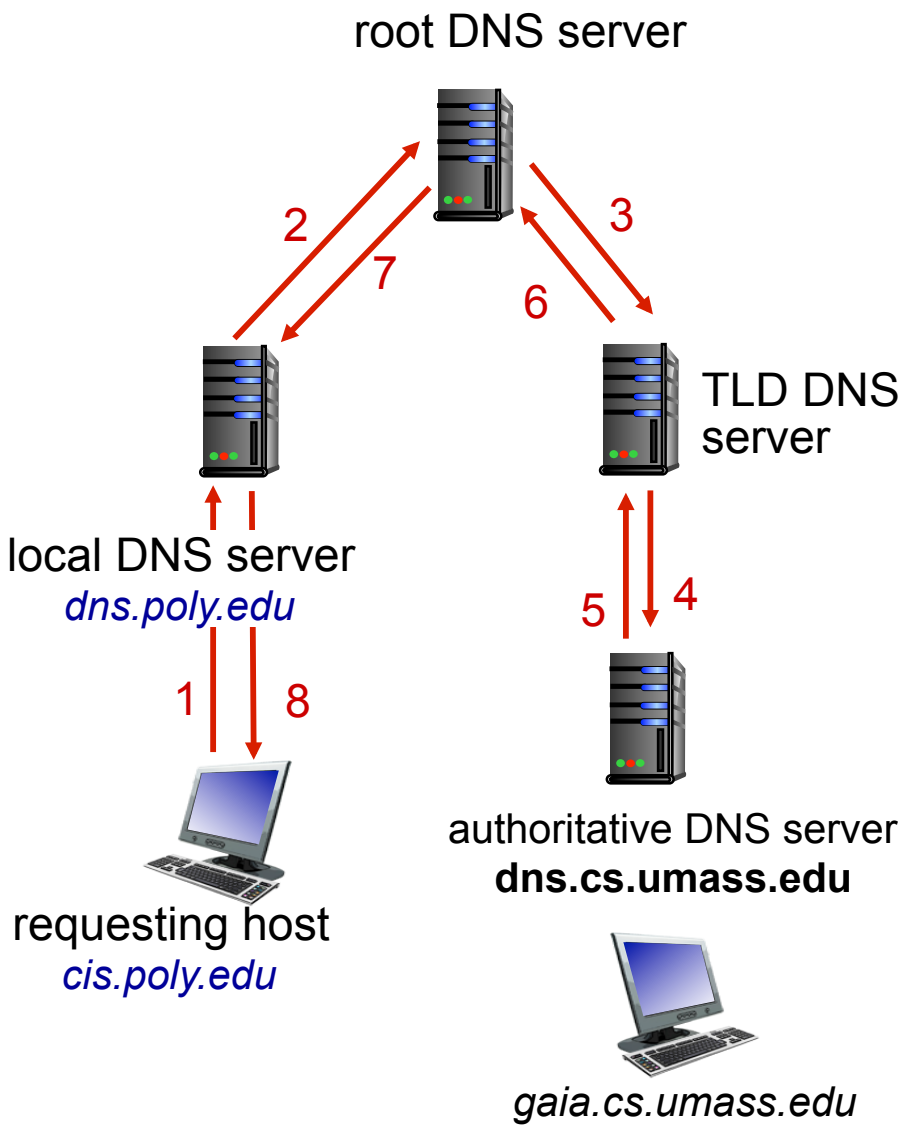
Esempio di risoluzione di un nome sibolico: modalità iterativa



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 sibolico: 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: *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

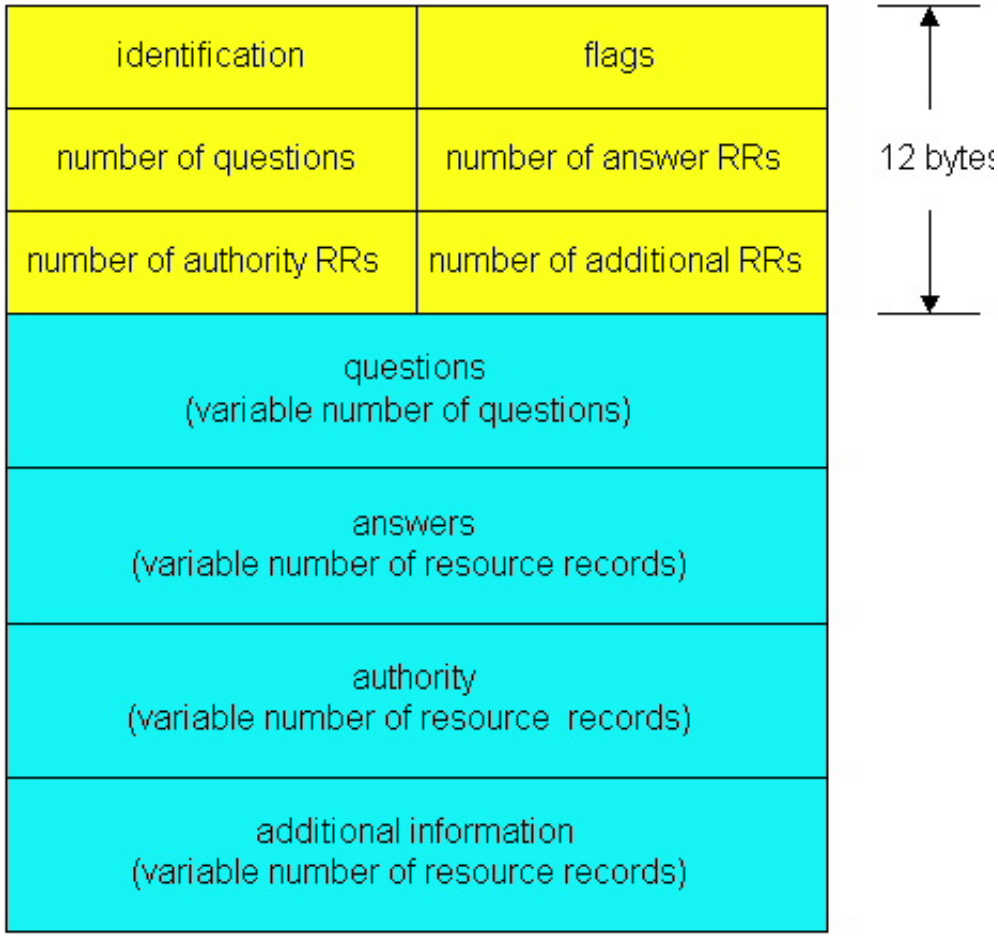
    (elet.polimi.it, morgana.elet.polimi.it, NS, TTL)
  - CNAME: *Name* è un nome alternativo (alias) per un *host* il cui nome canonico è in *Value*

    (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 infor.: informazione addizionale, ad es. il record con l'IP ADDR. per il MX fornito in answers



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

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

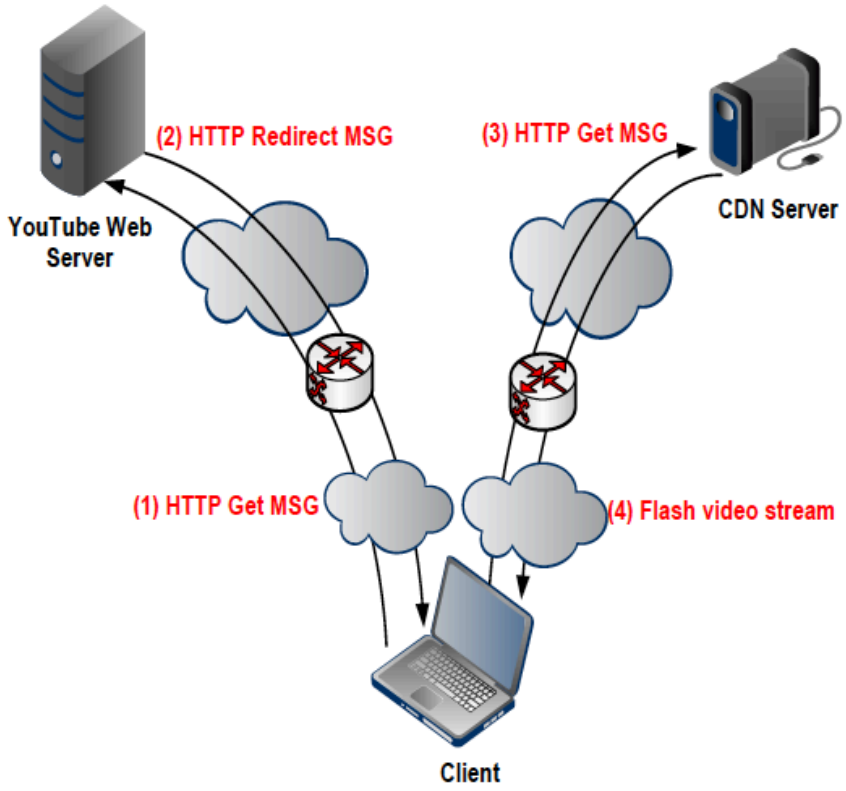
Attività proposta – DNS e Wireshark

- 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??)

Applicazioni Peer-to-Peer

File sharing, architettura, search

P2P file sharing

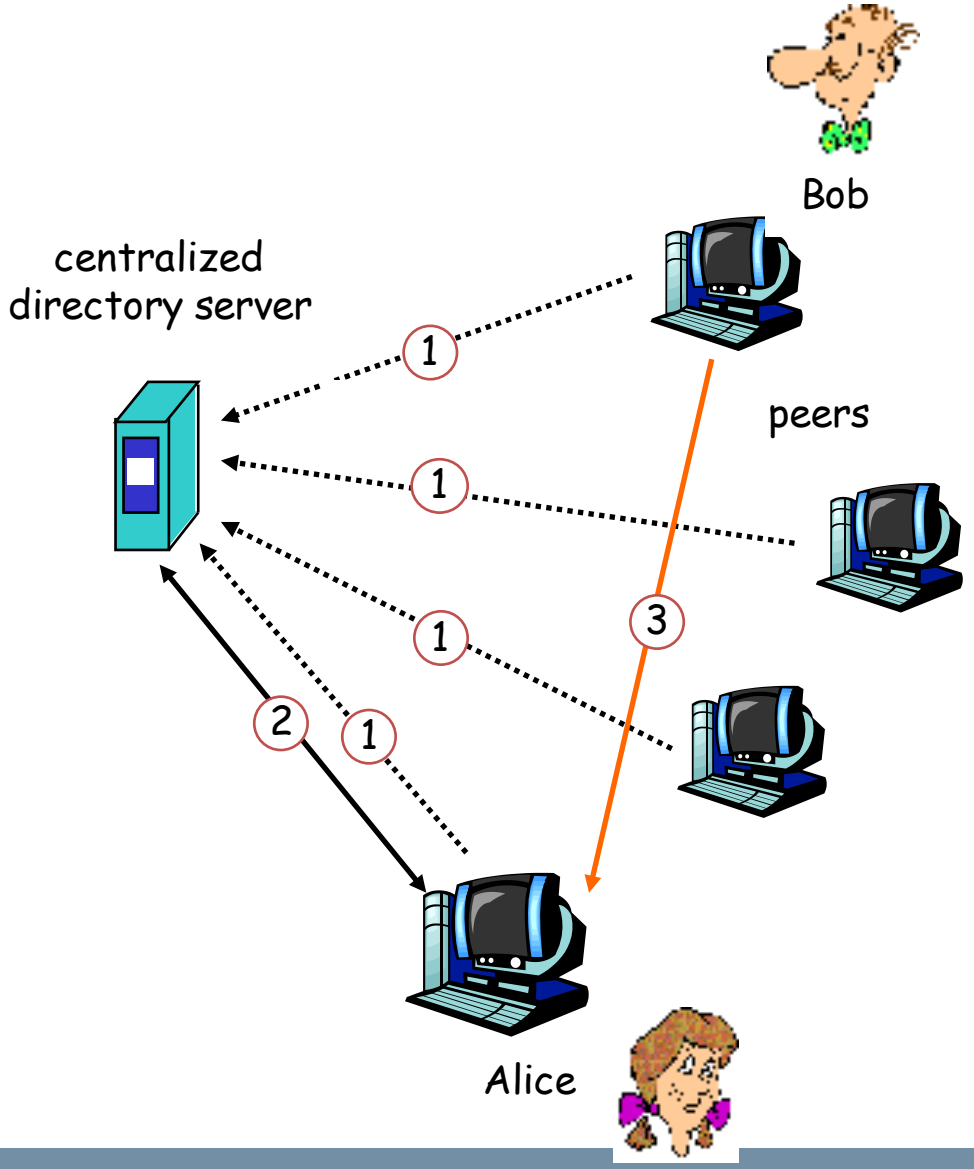
- Gli utenti utilizzano il software P2P sul proprio PC
- Si collegano in modo intermittente a Internet prendendo indirizzi IP diversi ogni volta
- Se un utente cerca un file l'applicazione trova altri utenti che lo hanno
- L'utente sceglie da chi scaricarlo
- Il file è scaricato usando un protocollo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

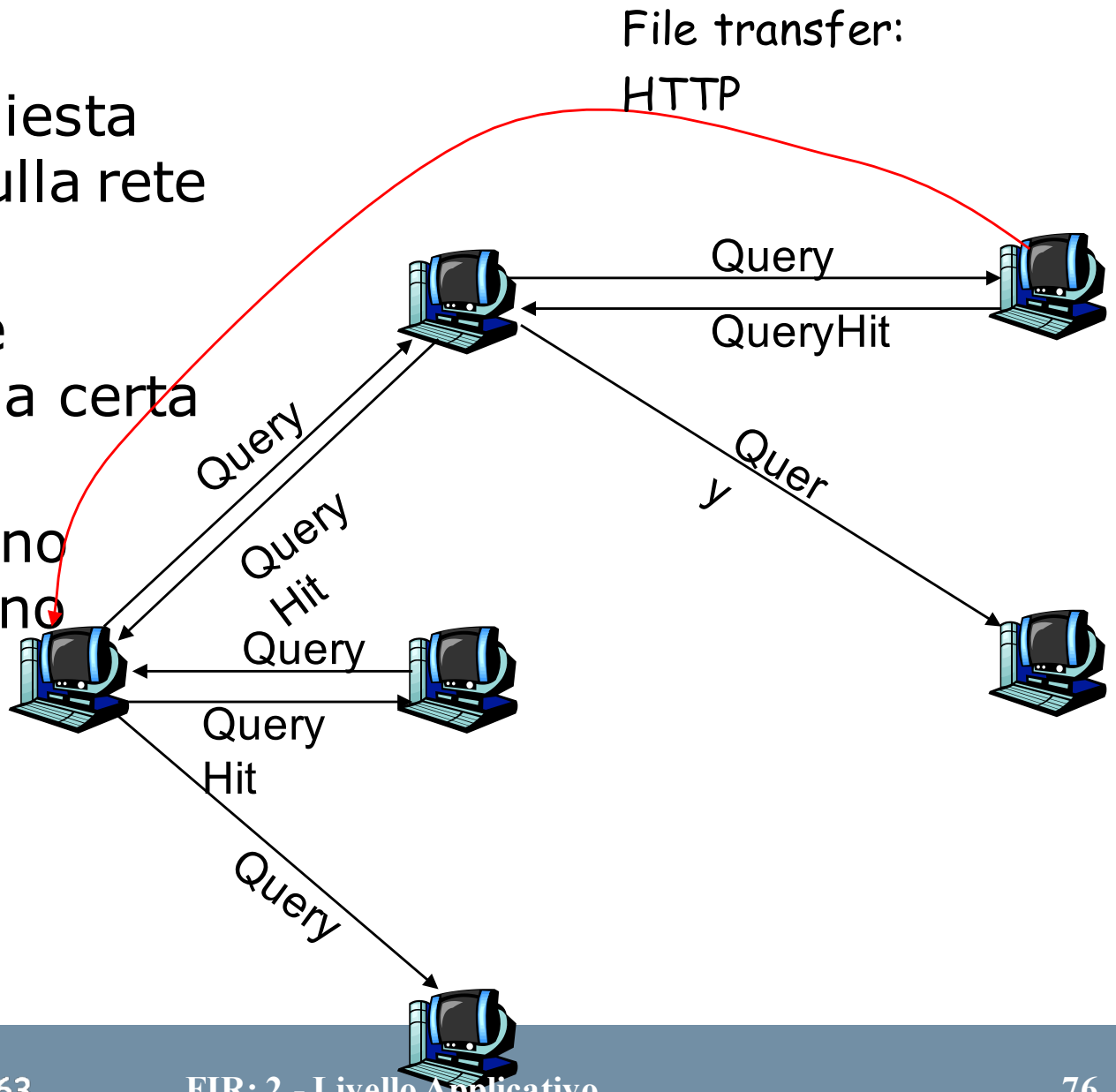
Basato su una rete (grafo) di *overlay*

- 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

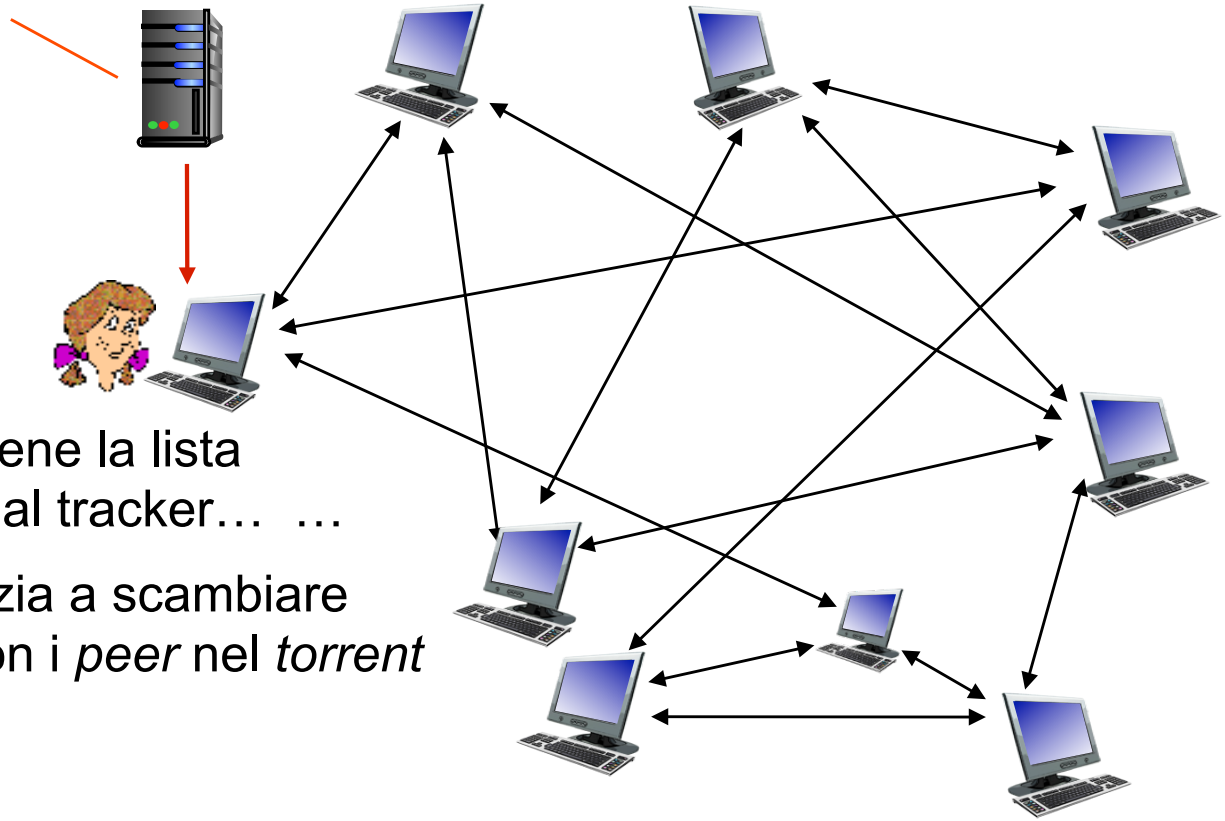
- Accesso alla rete
  - Per iniziare il processo di accesso il *peer* X deve trovare almeno un altro *peer*; la ricerca si basa su liste note
  - X scandisce la lista fino a che un *peer* Y risponde
  - X invia un messaggio di Ping a Y; Y inoltra il Ping nella rete di *overlay*.
  - Tutti i *peer* che 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

- 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

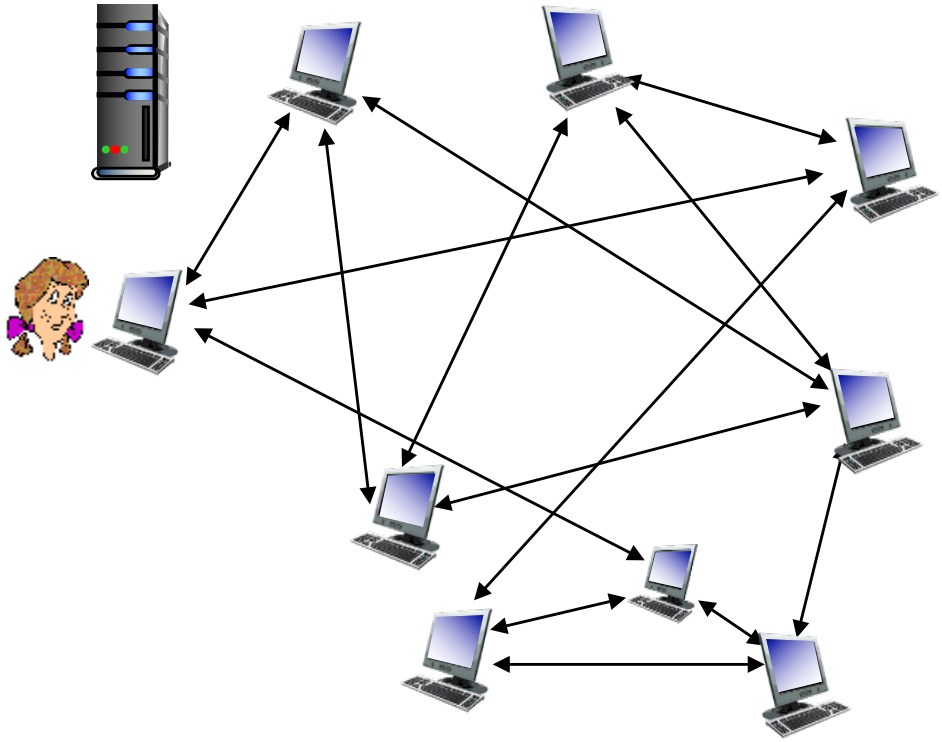


Alice ottiene la lista di peer dal tracker… …

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

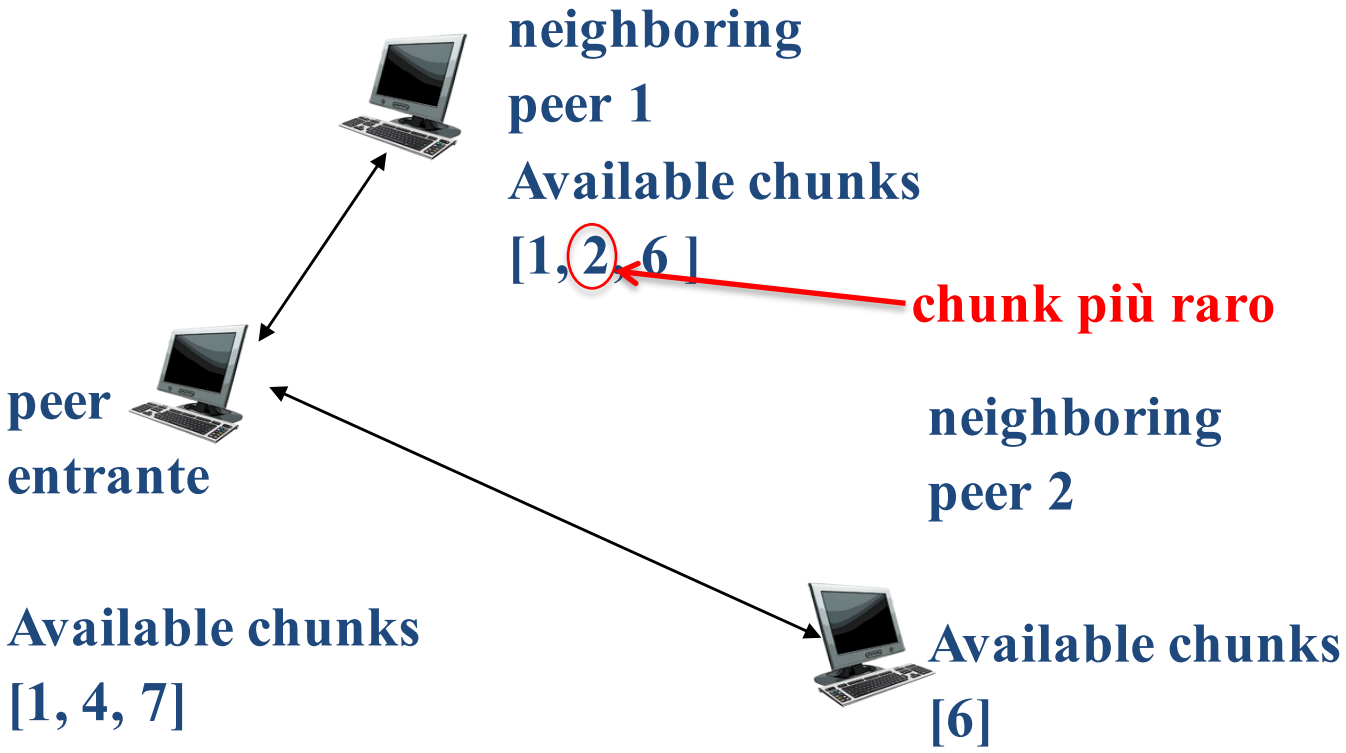
BitTorrent – entrare nel *torrent*

- 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*)