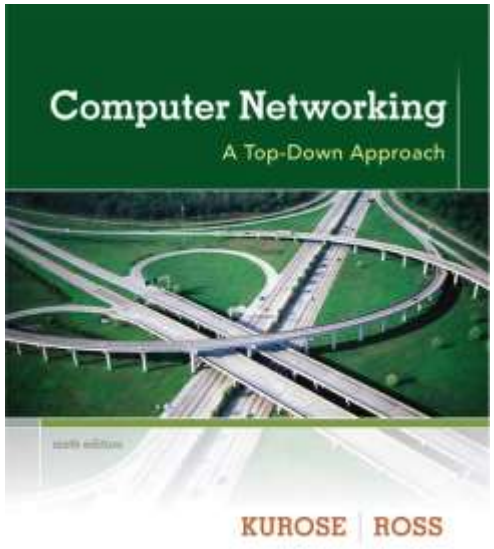
Capitolo 2
Livello di applicazione



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Nota per l'utilizzo:

Abbiamo preparato queste slide con l'intenzione di renderle disponibili a tutti (professori, studenti, lettori). Sono in formato PowerPoint in modo che voi possiate aggiungere e cancellare slide (compresa questa) o modificarne il contenuto in base alle vostre esigenze.

Come potete facilmente immaginare, da parte nostra abbiamo fatto *un sacco* di lavoro. In cambio, vi chiediamo solo di rispettare le seguenti condizioni:

- se utilizzate queste slide (ad esempio, in aula) in una forma sostanzialmente inalterata, fate riferimento alla fonte (dopo tutto, ci piacerebbe che la gente usasse il nostro libro!)
- se rendete disponibili queste slide in una forma sostanzialmente inalterata su un sito web, indicate che si tratta di un adattamento (o che sono identiche) delle nostre slide, e inserite la nota relativa al copyright.

Thanks and enjoy! JFK/KWR

Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

- SMTP, POP3, IMAP

2.5 DNS

2.6 applicazioni P2P

2.7 programmazione delle socket con UDP e TCP

Capitolo 2: livello di applicazione

obiettivi:

- fornire i concetti base e gli aspetti implementativi dei protocolli delle applicazioni di rete
  - modelli di servizio del livello di trasporto
  - paradigma client-server
  - paradigma peer-to-peer
- apprendere informazioni sui protocolli esaminando quelli delle più diffuse applicazioni di rete
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creare applicazioni di rete
  - API delle socket

Alcune applicazioni di rete

- posta elettronica
- web
- messaggistica istantanea
- login remoto
- condivisione di file P2P
- giochi multiutente via rete
- streaming di video (YouTube, Hulu, Netflix)
- voice over IP (es., Skype)
- video conferenza in real-time
- social networking
- ricerche
- …
- …

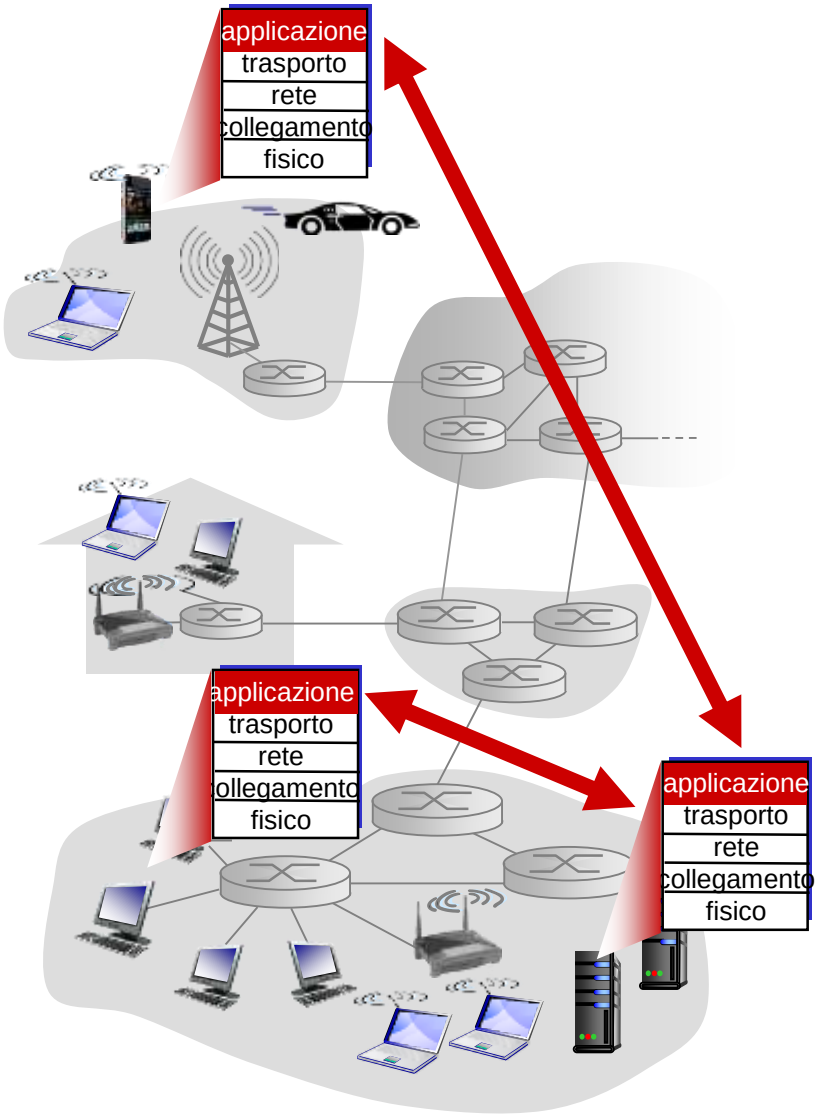
Creare un'applicazione di rete

scrivere programmi che:

- girano su (differenti) *end systems*
- comunicano attraverso la rete
- es., il software di un server web comunica con il software di un browser

non è necessario scrivere software che tenga conto dei dispositivi interni alla rete

- i dispositivi della rete non eseguono applicazioni utente
- pensare le applicazioni per gli end system permette sviluppo e diffusione più rapidi per le applicazioni stesse

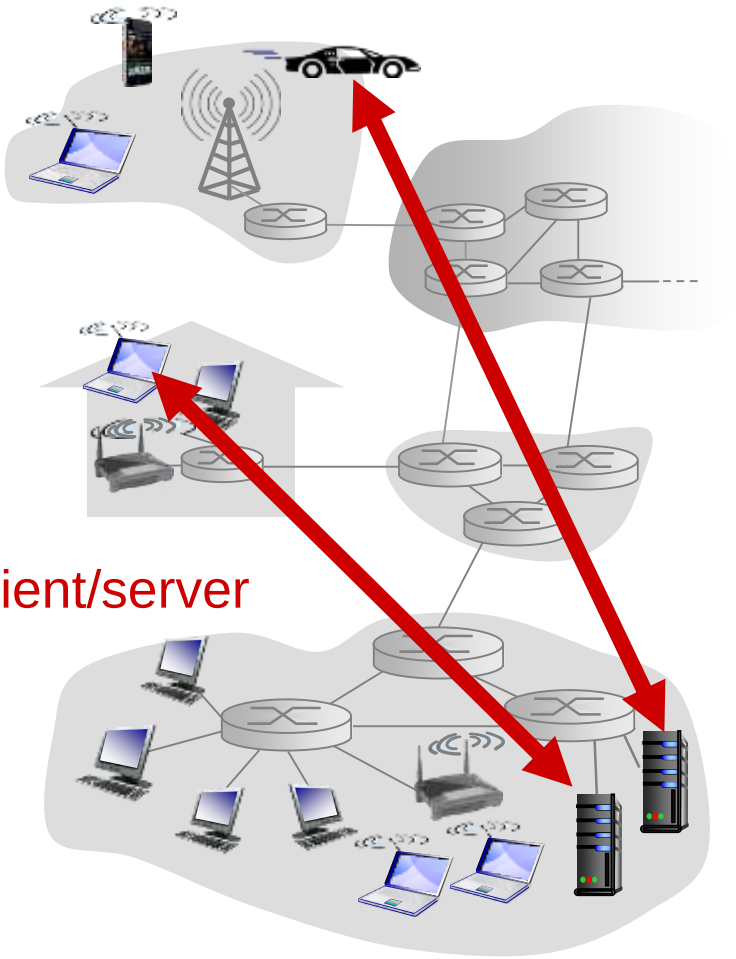


Architetture delle applicazioni

possibili strutture delle applicazioni:

- client-server
- peer-to-peer (P2P)

Architettura client-server



server:

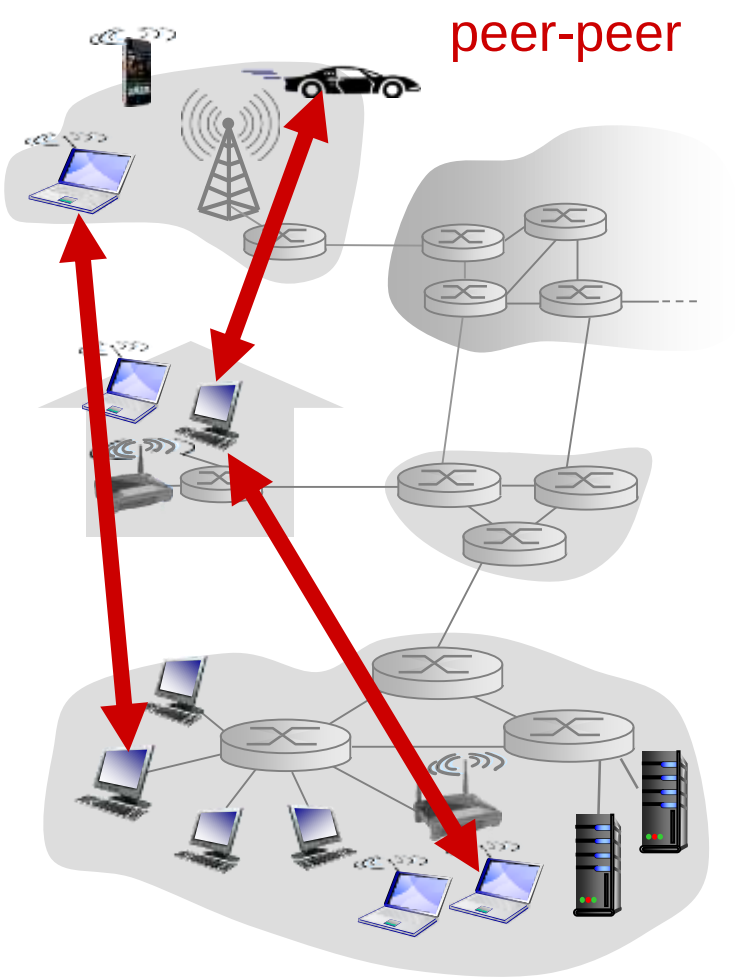
- host sempre attivo
- indirizzo IP fisso
- data center per avere scalabilità

client:

- comunica con il server
- può non essere sempre connesso
- può avere indirizzo IP dinamico
- non comunica direttamente con gli altri client

Architettura P2P

- *non c'*è un server sempre attivo
- coppie arbitrarie di end system comunicano direttamente
- i peer richiedono servizi agli altri peer, e in cambio forniscono servizi agli altri peer
  - *auto scalabilità* – nuovi peer portano maggiore offerta di servizi, così come maggiore domanda
- i peer non sono permanentemente connessi e cambiano indirizzo IP
  - gestione complessa



Processi comunicanti

processo: programma in esecuzione su un host

- all'interno dello stesso host, due processi comunicano usando comunicazioni inter-processo (definiti dal SO)
- processi su host differenti comunicano attraverso lo scambio di messaggi

clients, servers

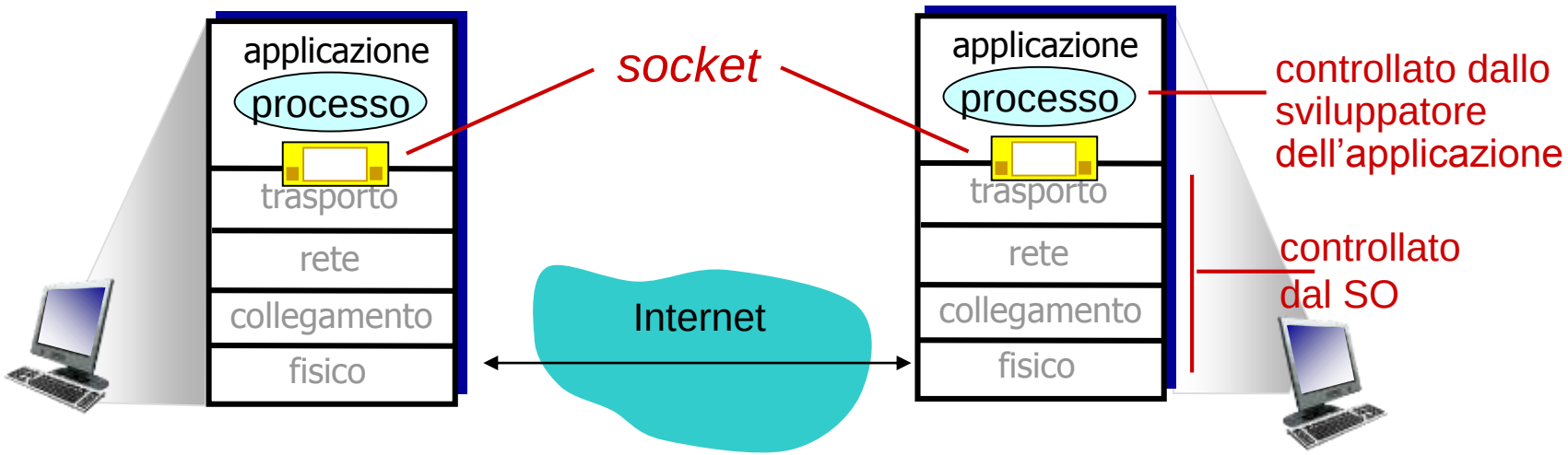
processo client : processo che dà inizio alla comunicazione

processo server : processo che attende di essere contattato

- le applicazioni con architetture P2P hanno processi client e processi server

Socket

- un processo invia/riceve messaggi verso/da la sua socket
- una socket è analoga a una porta
  - il processo inviante immette il messaggio sulla porta
  - il processo inviante fa affidamento su una infrastruttura di trasporto "fuori dalla porta" che si preoccuperà di recapitare il messaggio fino alla porta del processo ricevente



Processi di indirizzamento

- per ricevere messaggi, un processo deve avere un *identificativo*
- un host ha un indirizzo IP univoco a 32 bit
- *D:* l'indirizzo IP dell'host sul quale è in esecuzione il processo è sufficiente a identificare il processo?
- *R:* no, *molti* processi possono essere in esecuzione sullo stesso host

- l'*identificativo* include sia l'indirizzo IP che il numero di porta associato al processo sull'host.
- esempi di numeri di porta:
  - HTTP server: 80
  - mail server: 25
- per inviare messaggi HTTP al server web gaia.cs.umass.edu:
  - IP address: 128.119.245.12
  - port number: 80
- approfondiremo più avanti…

Protocollo a livello di applicazione

- tipi di messaggi scambiati,
  - es., richiesta, risposta
- sintassi dei messaggi:
  - quali sono i campi dei messaggi e come sono strutturati
- semantica dei messaggi
  - significato delle informazioni nei campi
- regole per determinare quando e come un processo invia e risponde ai messaggi

protocolli di dominio pubblico:

- definiti nelle RFC
- consentono l'interoperabilità
- es., HTTP, SMTP

protocolli proprietari:

- es., Skype

Quale servizio di trasporto richiede un'applicazione?

integrità dei dati

- alcune applicazioni (es., trasferimento di file, transazioni web) richiedono un trasferimento dati affidabile al 100%
- altre applicazioni (es., audio) possono tollerare delle perdite

temporizzazione

- alcune applicazioni (es., telefonia Internet, giochi interattivi) richiedono ritardi contenuti per essere "efficaci"

throughput

- alcune applicazioni (e.g., multimedia) richiedono un minimo di throughput per essere "efficaci"
- altre applicazioni ("applicazioni elastiche") utilizzano il throughput disponibile

sicurezza

- crittografia, integrità dei dati, …

Requisiti del servizio di trasporto di applicazioni comuni

applicazione	tolleranza alla perdita di dati	throughput	time sensitive
trasferimento file	no	elastico	no
posta elettronica	no	elastico	no
documenti Web	no	elastico	no
real-time audio/video	sì	audio: 5kbps-1Mbps video:10kbps-5Mbps	sì, 100's msec
stored audio/video	sì	come sopra	sì, pochi secondi
giochi interattivi	sì	fino a pochi kbps	sì, 100's msec
messaggistica istantanea	no	elastico	sì e no

Servizi dei protocolli di trasporto Internet

servizi TCP :

- *trasporto affidabile* tra i processi di invio e di destinazione
- *controllo di flusso:* il mittente non sovraccarica il destinatario
- *controllo della congestione:* rallenta il processo d'invio quando la rete è sovraccarica
- *non fornisce:* temporizzazione, minimo garantito di throughput, sicurezza
- *orientato alla connessione:* è richiesto un setup tra i processi client e server

servizi UDP :

- *trasferimento dati inaffidabile* tra i processi client e server
- *non fornisce:* affidabilità, controllo di flusso, controllo della congestione , temporizzazione, throughput garantito, sicurezza
- *non richiede:* setup della connessione

D: perché esiste UDP?

Applicazioni Internet e relativi protocolli

applicazione	protocollo del livello di applicazione	protocollo di trasporto sottostante
posta elettronica	SMTP [RFC 2821]	TCP
accesso a terminale remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
trasferimento file	FTP [RFC 959]	TCP
streaming	HTTP (es., YouTube), RTP [RFC 1889]	TCP o UDP
telefonia Internet	SIP, RTP, proprietary (e.g., Skype)	TCP o UDP

Rendere più sicuro il TCP

TCP & UDP

- nessuna crittografia
- password inviate in chiaro nelle socket attraversano Internet in chiaro

SSL

- fornisce connessioni TCP crittografate
- integrità dei dati
- autenticazione degli end-point

SSL è nel livello di applicazione

- Applicazioni usano librerie SSL, che "parlano" al TCP

API delle socket SSL

- password inviate in chiaro nelle socket attraversano Internet cifrate
- Vedi Capitolo 7

Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

- SMTP, POP3, IMAP

2.5 DNS

2.6 applicazioni P2P

2.7 programmazione delle socket con UDP e TCP

Web e HTTP

Terminologia

- una *pagina web* è costituita da *oggetti*
- un oggetto può essere un file HTML , un'immagine JPEG, un'applet Java, un file audio,…
- una pagina web è formata da *file HTML di base* che include *diversi oggetti referenziati*
- Ogni oggetto è referenziato da un *URL,* es.,

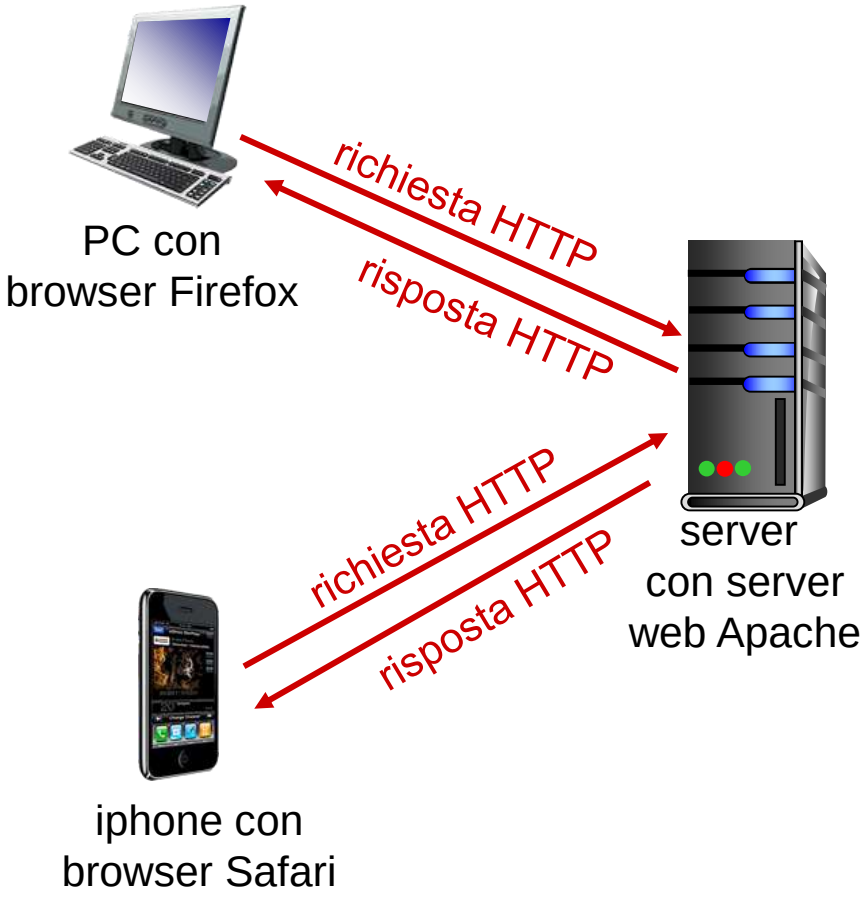
`www.someschool.edu/someDept/pic.gif`

host name | path (percorso)

Panoramica su HTTP

HTTP: hypertext transfer protocol

- protocollo a livello di applicazione del Web
- modello client/server
  - *client:* browser che richiede, riceve, (usando il protocollo HTTP) e "visualizza" gli oggetti Web
  - *server:* server Web che invia (usando il protocollo HTTP) oggetti in risposta a una richiesta



Panoramica su HTTP (continua)

usa TCP:

- il client inizializza la connessione TCP (crea una socket) con il server, sulla porta 80
- il server accetta la connessione TCP dal client
- messaggi HTTP vengono scambiati tra il browser (client HTTP) e il server Web (server HTTP)
- la connessione TCP viene chiusa

HTTP è "stateless" (senza stato)

- il server non mantiene informazioni su precedenti richieste del client

nota

i protocolli che mantengono lo "stato" sono complessi!

- la storia passata (stato) deve essere conservata
- se server o client si bloccano, la loro visione dello "stato" potrebbe non coincidere e sarebbe necessario un riallineamento

Connessioni HTTP

HTTP non persistenti

- al più un oggetto viene trasmesso su una connessione TCP
  - la connessione viene quindi chiusa
- la ricezione di oggetti multipli richiede connessioni multiple

HTTP persistente

- oggetti multipli possono essere inviati su una singola connessione TCP tra client e server

Connessioni non persistenti

supponiamo che l'utente immetta l'URL:
`www.someSchool.edu/someDepartment/home.index`

(contiene testo, e referenzia 10 immagini jpeg)

1a. il client HTTP inizializza una connessione TCP con il (processo) server HTTP www.someSchool.edu sulla porta 80

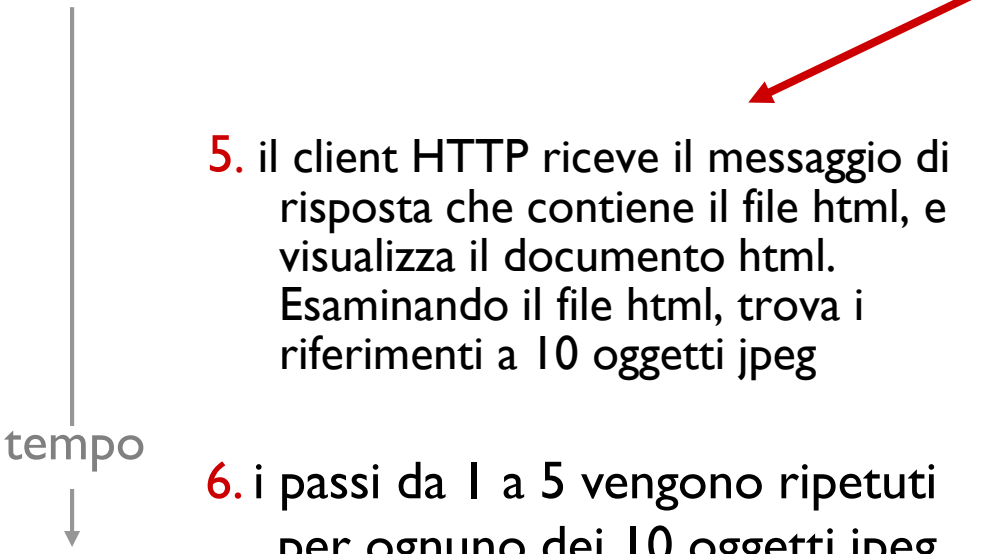
1b. il server HTTP sull'host www.someSchool.edu in attesa di connessioni TCP sulla porta 80. "accetta" la connessione, e avvisa il client

2. il client HTTP invia un *messaggio di richiesta* HTTP (contenente l'URL) nella socket della connessione TCP. Il messaggio indica che il client vuole l'oggetto someDepartment/home.index

3. il server HTTP riceve il messaggio di richiesta, produce il *messaggio di risposta* contenente l'oggetto richiesto, e invia il messaggio nella sua socket

tempo

Connessioni non persistenti (cont.)



4. il server HTTP chiude la connessione TCP

5. il client HTTP riceve il messaggio di risposta che contiene il file html, e visualizza il documento html. Esaminando il file html, trova i riferimenti a 10 oggetti jpeg

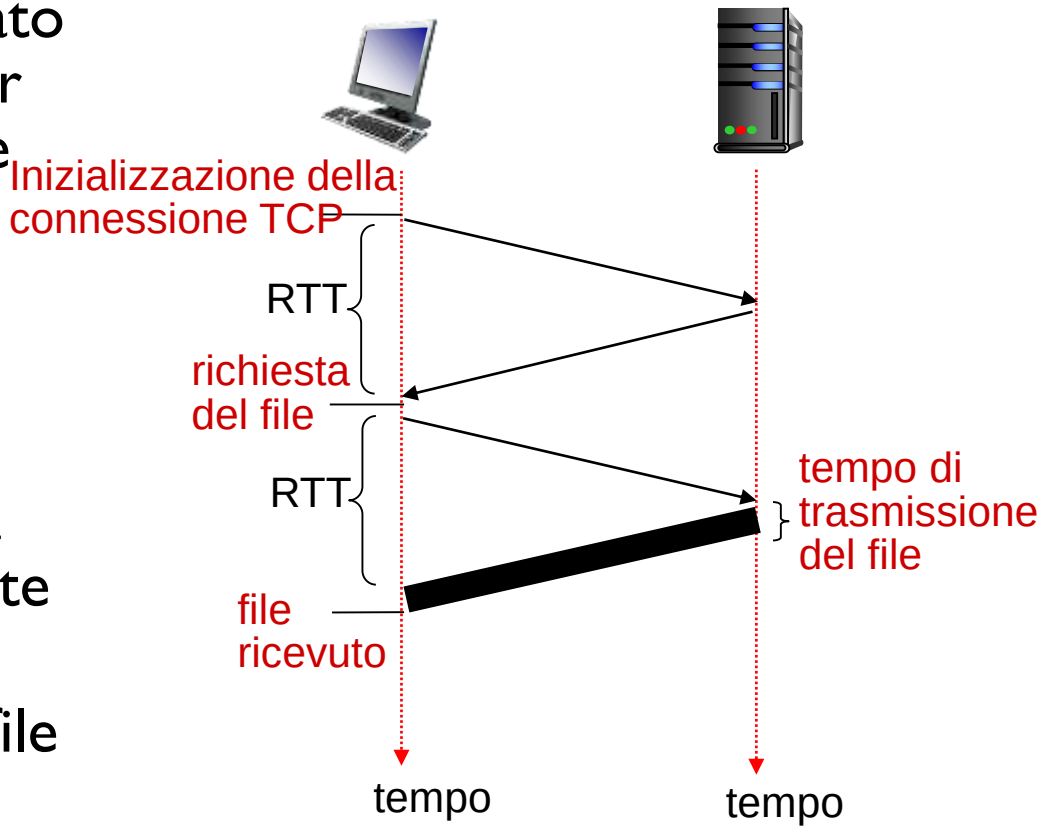
6. i passi da 1 a 5 vengono ripetuti per ognuno dei 10 oggetti jpeg

HTTP non persistente: tempo di risposta

RTT (round-trip time, definizione): tempo impiegato da un piccolo pacchetto per andare dal client al server e ritornare al client

tempo di risposta:

- un RTT per inizializzare la connessione TCP
- un RTT perché ritornino la richiesta HTTP e i primi byte della risposta HTTP
- tempo di trasmissione del file
- totale = 2RTT+ tempo di trasmissione del file



Connessioni persistenti

connessioni non persistenti:

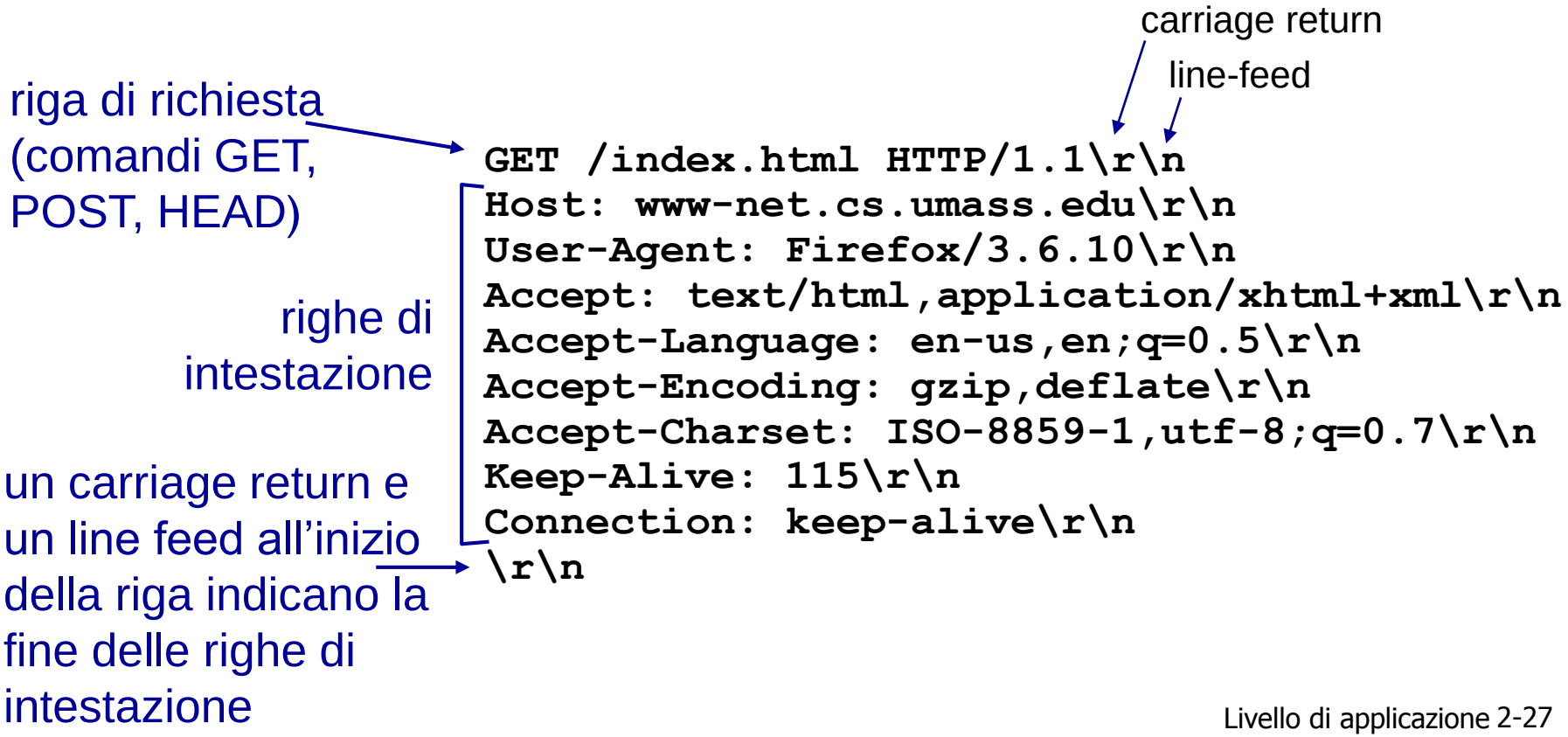
- richiede 2 RTT per oggetto
- overhead del sistema operativo per *ogni* connessione TCP
- i browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati

connessioni persistenti:

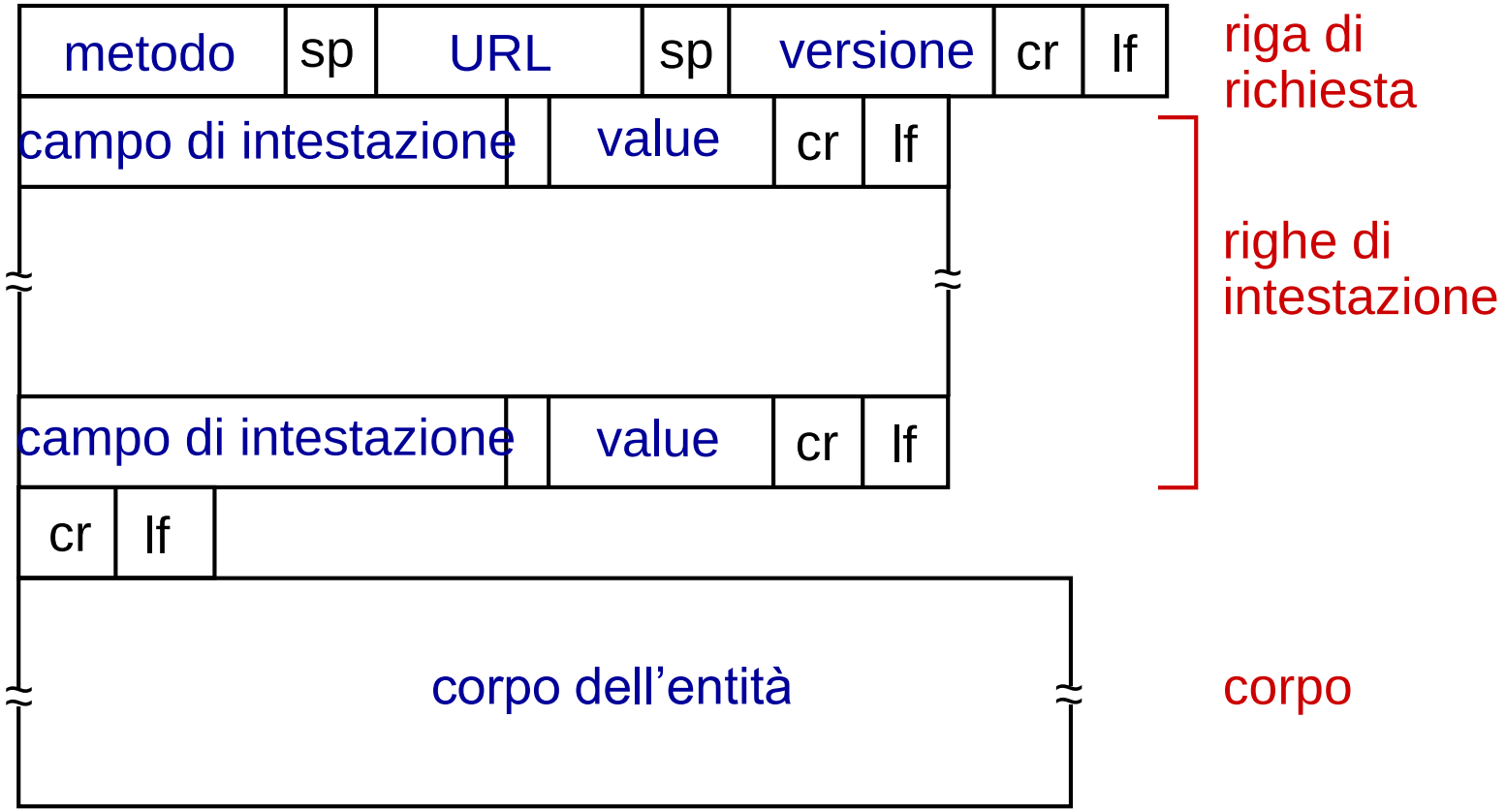
- il server lascia la connessione TCP aperta dopo l'invio di una risposta
- i successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta
- il client invia le richieste non appena incontra un oggetto referenziato
- un solo RTT per tutti gli oggetti referenziati

Messaggi HTTP

- due tipi di massaggi HTTP: *richiesta, risposta*
- messaggio di richiesta HTTP:
  - ASCII (formato human-readable)



Messaggio di richiesta HTTP: formato generale



Upload dell'input di un form

metodo POST :

- la pagina web spesso include un form per l'input dell'utente
- l'input arriva al server nel corpo dell'entità

metodo URL :

- usa il metodo GET
- l'input arriva al server nel campo URL della riga di richiesta

```
www.somesite.com/animalsearch?monkeys&banana
```

Tipi di metodi

HTTP/1.0:

- GET
- POST
- HEAD
  - chiede al server di escludere l'oggetto richiesto dalla risposta

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - include il file nel corpo dell'entità e lo invia al percorso specificato nel campo URL
- DELETE
  - cancella il file specificato nel campo URL

Messaggio di risposta HTTP

riga di stato (protocollo, codice di stato messaggio di stato)

righe di intestazione

dati, es., il file HTML richiesto

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

Codici di stato della risposta HTTP

- il codice di stato compare nella prima riga nel messaggio di risposta server->client
- alcuni codici di stato:

200 OK

- la richiesta ha avuto successo; l'oggetto richiesto viene inviato nella risposta

301 Moved Permanently

- l'oggetto richiesto è stato spostato; la nuova posizione è specificata nell'intestazione Location: della risposta

400 Bad Request

- Il messaggio di richiesta non è stato compreso dal server

404 Not Found

- Il documento richiesto non si trova su questo server

505 HTTP Version Not Supported

Come provare HTTP (lato client)

1. collegatevi via Telnet al vostro server web preferito:

```
telnet cis.poly.edu 80
```

apre una connessione TCP sulla porta 80 (porta di default per un server HTTP) dell'host cis.poly.edu.
tutto ciò che digitate viene trasmesso alla porta 80 di cis.poly.edu

2. digitate una richiesta GET:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

digitando questo (premete due volte il tasto Invio), trasmettete una richiesta GET minima (ma completa) al server HTTP

3. guardate il messaggio di risposta trasmesso dal server HTTP!

(o usate Wireshark per guardare le richieste/risposte catturate)

Stato utente-server: i cookie

molti dei più importanti siti web usano i cookie

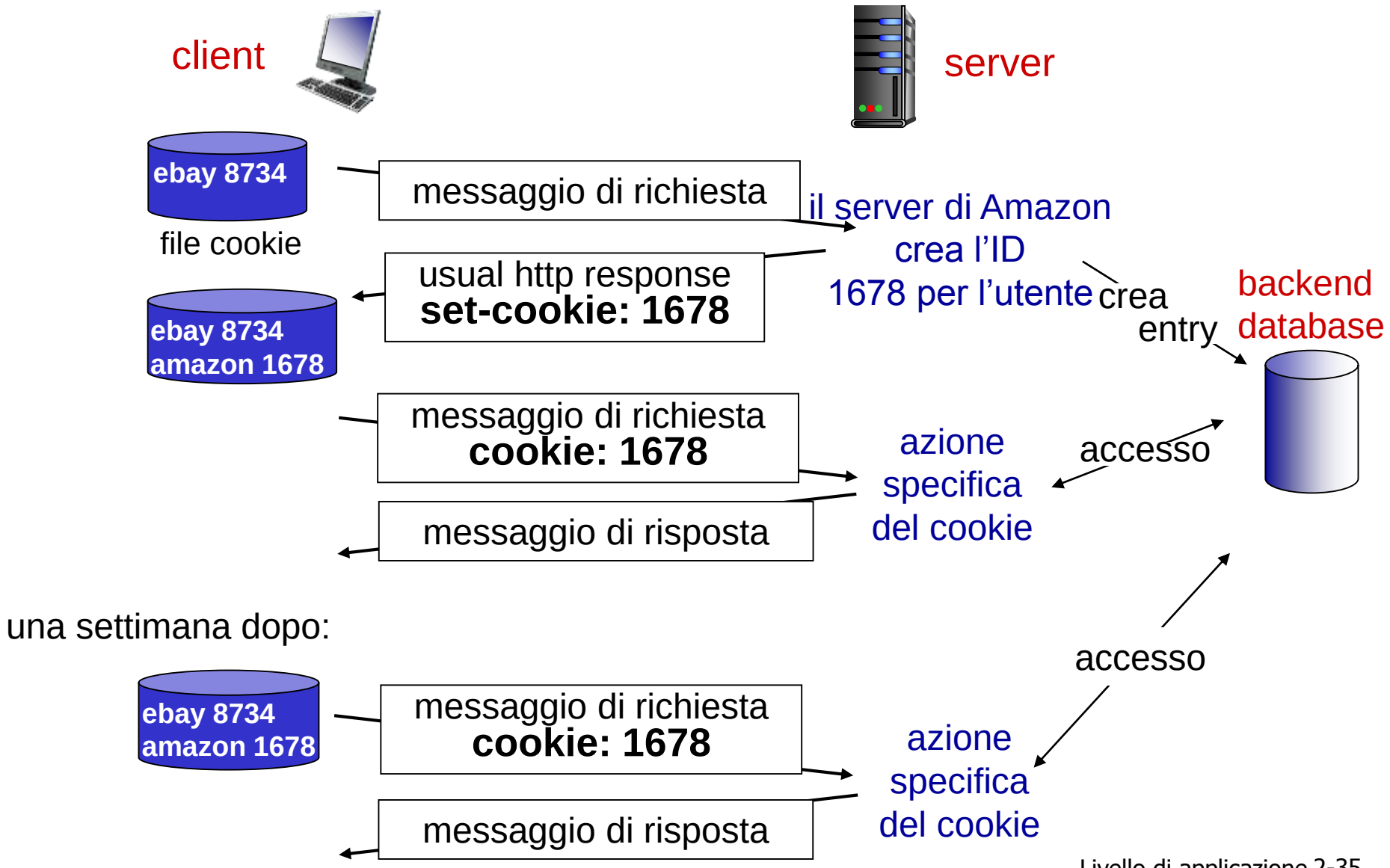
quattro componenti:

1) una riga di intestazione nel messaggio di *risposta* HTTP
2) una riga di intestazione cookie nel successivo messaggio di richiesta HTTP
3) un file cookie mantenuto sull'host dell'utente e gestito dal browser dell'utente
4) un database sul sito Web

esempio:

- Susan accede a Internet sempre dallo stesso PC
- visita per la prima volta un particolare sito di commercio elettronico
- quando la richiesta HTTP iniziale giunge al sito, il sito crea:
  - un'identificatico unico (ID)
  - una entry nel database per quell'ID

Cookie (continua)



Cookie (continua)

per cosa possono essere usati i cookie:

- autorizzazione
- carrelli della spesa
- suggerimenti
- stato della sessione dell'utente (Web e-mail)

nota

cookie e privacy:

- i cookie permettono ai siti di imparare molte cose sugli utenti
- l'utente può fornire al sito il nome e l'indirizzo e-mail

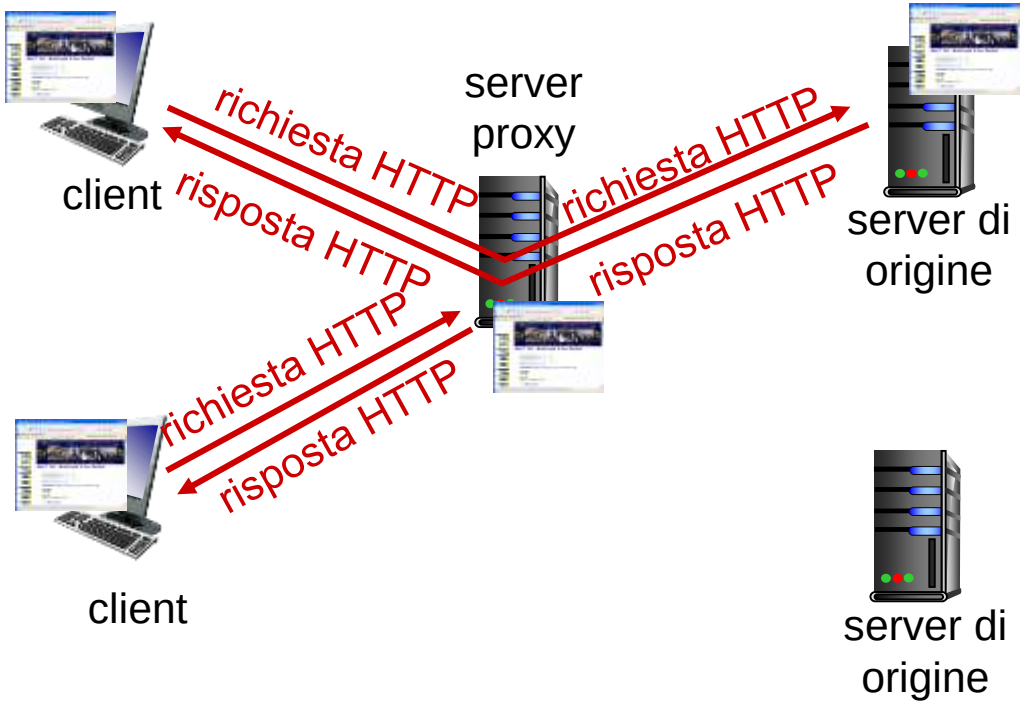
come mantenere lo "stato":

- endpoint del protocollo: mantengono lo stato tra sender e receiver dopo ogni transazione
- cookies: trasportano informazioni di stato nei messaggi http

Cache web (server proxy)

obiettivo: soddisfare la richiesta del client senza coinvolgere il server d'origine

- l'utente configura il browser: accesso al Web tramite la cache
- il browser trasmette tutte le richieste HTTP alla cache
  - oggetto nella cache: la cache fornisce l'oggetto
  - altrimenti la cache richiede l'oggetto al server d'origine e poi lo inoltra al client



Cache Web (continua)

- la cache opera come client e come server
  - server per il client che fa la richiesta
  - client per il server d'origine
- tipicamente la cache è installata da un ISP (università, aziende o ISP residenziali)

perché il Web caching?

- riduce i tempi di risposta alle richieste dei client
- riduce il traffico sul collegamento di accesso a Internet
- Internet arricchita di cache consente ai provider "scadenti" di fornire dati con efficacia (agisce così anche la condivisione di file P2P)

Esempio di caching

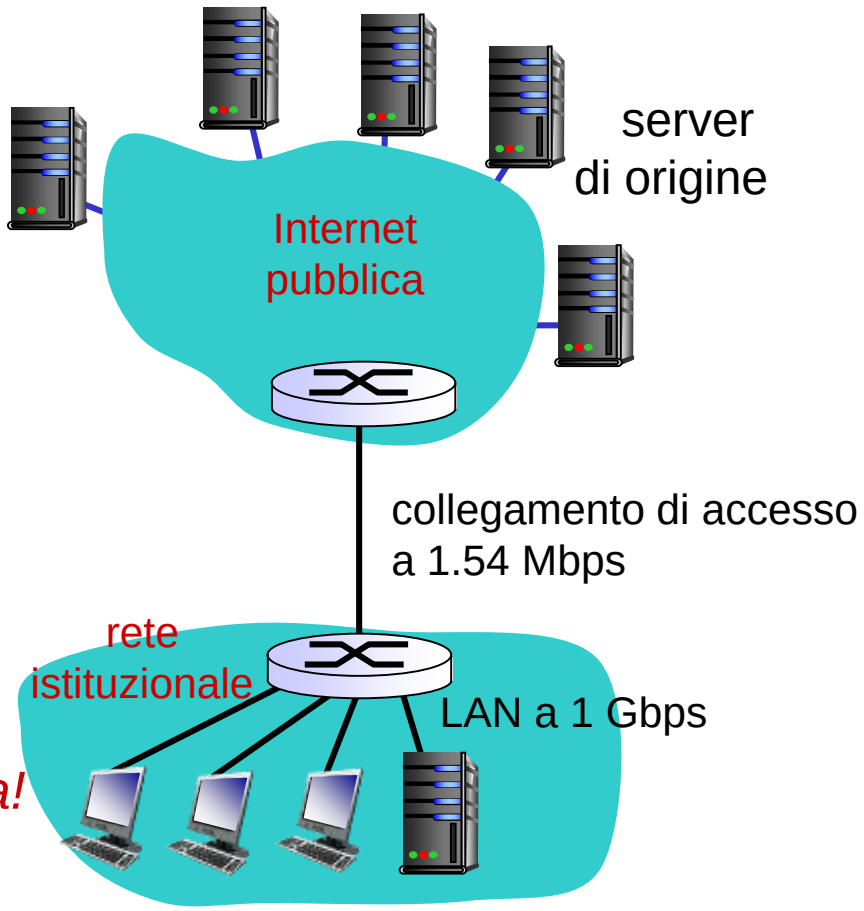
ipotesi:

- dimensione media di un oggetto: 100K bit
- frequenza media di richieste dai browser ai server di origine: 15/sec
- data rate medio verso i browser: 1.50 Mbps
- RTT dal router istituzionale verso ogni server d'origine: 2 sec
- rate del collegamento di accesso: 1.54 Mbps

conseguenze:

- utilizzo della LAN: 15%
- utilizzo del link di accesso = 99%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
  = 2 sec + minuti + millisecondi

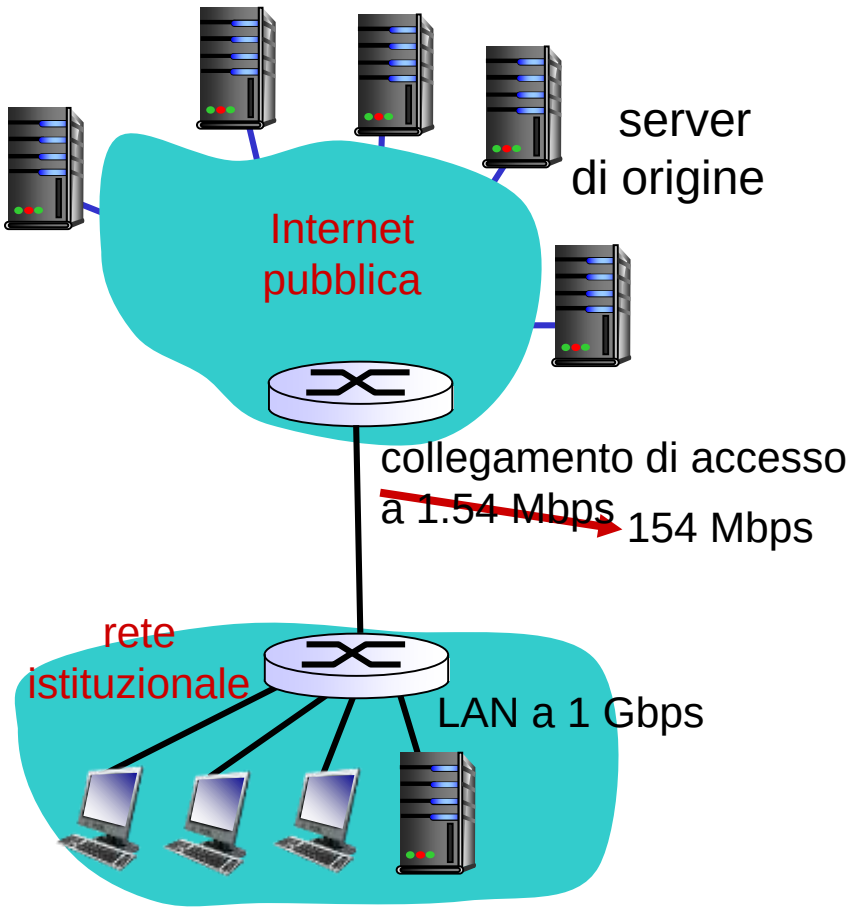
problema!



Esempio di caching (continua)

ipotesi:

- dimensione media di un oggetto: 100K bit
- frequenza media di richieste dai browser ai server di origine: 15/sec
- data rate medio verso i browser: 1.50 Mbps
- RTT dal router istituzionale verso ogni server d'origine : 2 sec
- rate del collegamento di accesso: ~~1.54 Mbps~~ 154 Mbps



conseguenze:

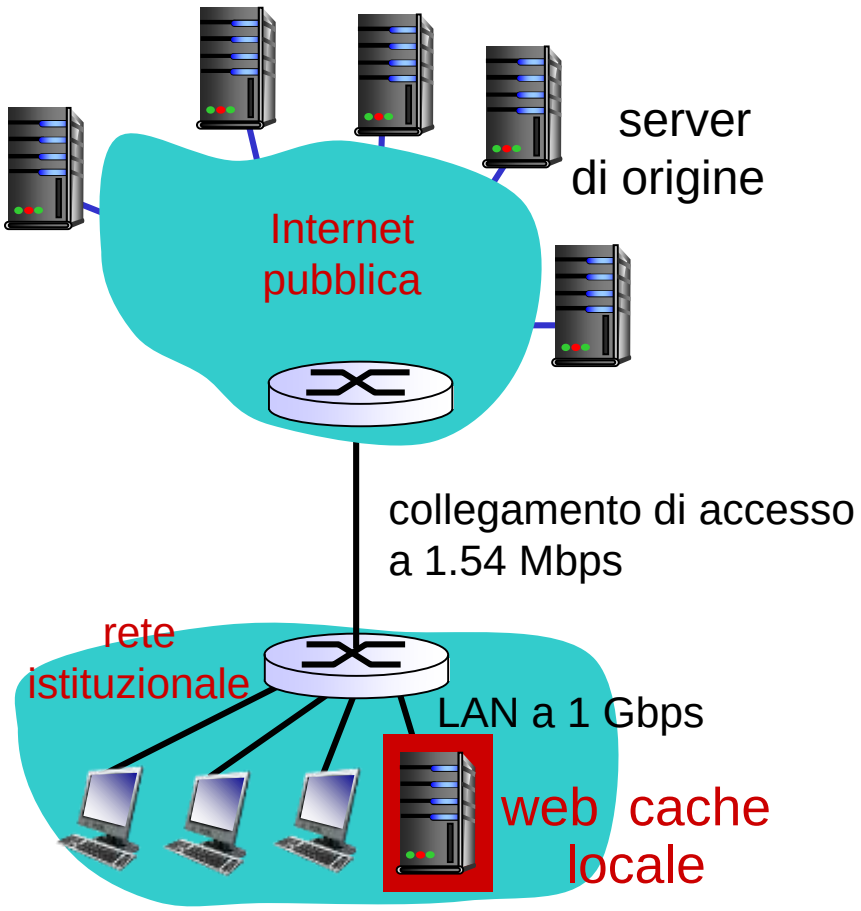
- utilizzo della LAN : 15%
- utilizzo del link di accesso = ~~99%~~ 9.9%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
  = 2 sec + ~~minuti~~ millisecondi + microsecondi

risultato ottenuto con un aumento della velocità del link (molto costoso!)

Caching example: cache locale

ipotesi:

- dimensione media di un oggetto: 100K bit
- frequenza media di richieste dai browser ai server di origine: 15/sec
- data rate medio verso i browser: 1.50 Mbps
- RTT dal router istituzionale verso ogni server d'origine: 2 sec
- rate del collegamento di accesso: 1.54 Mbps



conseguenze:

- utilizzo della LAN: 15%
- access link utilization = ?
- total delay = ?

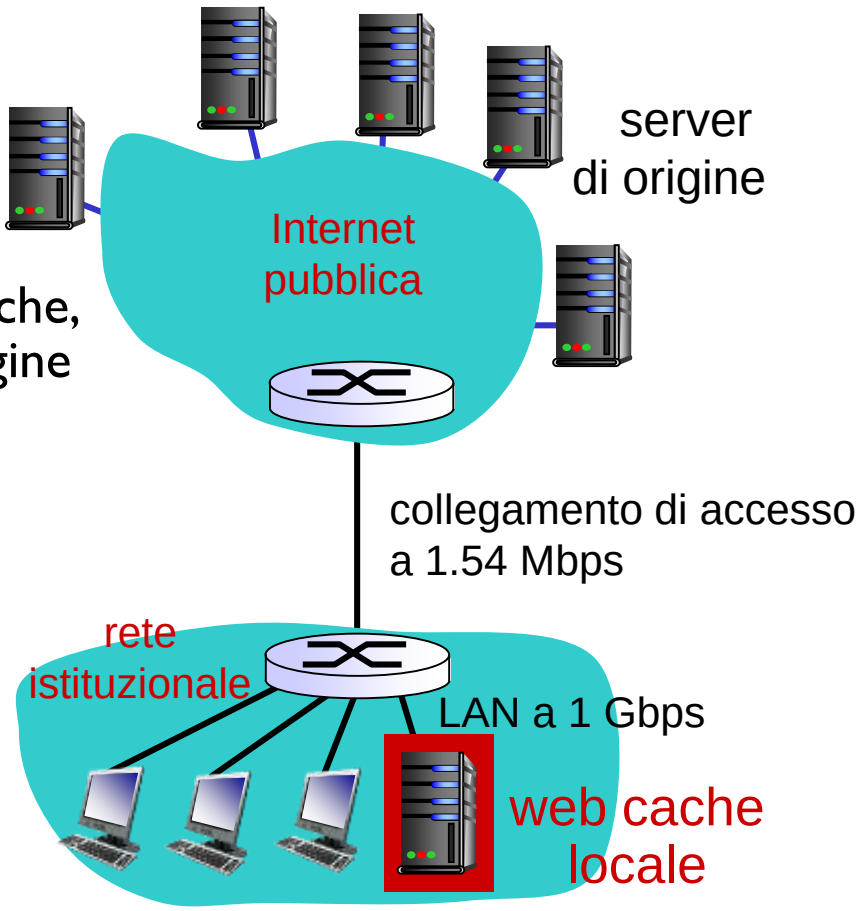
Come calcolare utilizzo e delay?

implementazione della web cache (economico!)

Caching example: cache locale

Calcolo dell'utilizzo del link e del delay con la cache:

- supponiamo una percentuale di successo (hit rate) di 0.4
  - 40% delle richieste soddisfatte dalla cache, 60% delle richieste soddisfatte dall'origine
- utilizzo del link di accesso:
  - 60% delle richieste usano il link
- data rate sul link = 0.6*1.50 Mbps = .9 Mbps
  - utilizzo = 0.9/1.54 = .58
- ritardo totale
  - = 0.6 * (ritardo dall'origine) +0.4 * (ritardo della cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - meno del link a154 Mbps link (e più economico!)



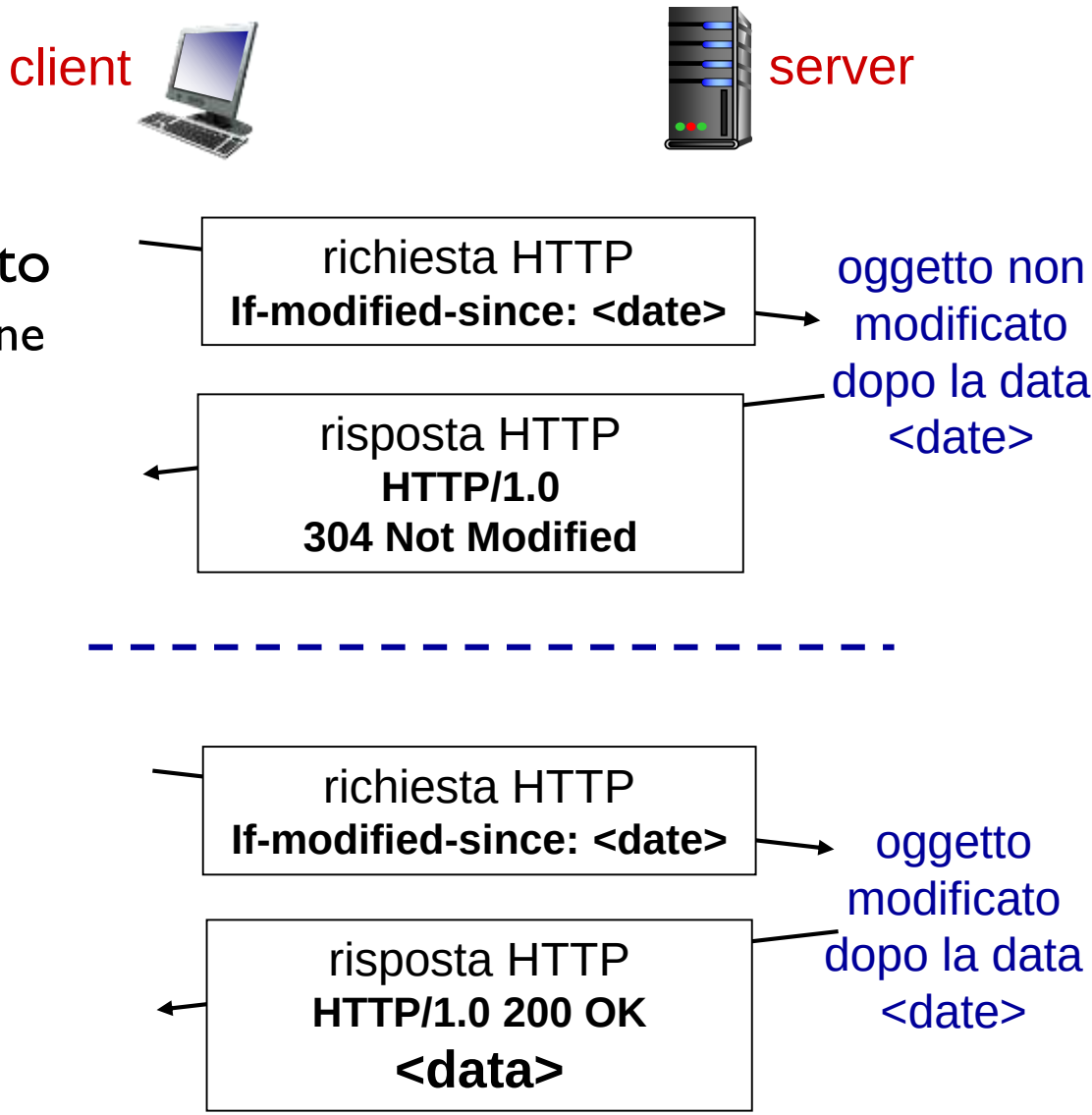
GET condizionale

- *Obiettivo:* non inviare un oggetto se la cache ha una copia aggiornata dell'oggetto
  - nessun ritardo di trasmissione
  - minor utilizzo del link
- *cache:* specifica la data della copia dell'oggetto nella richiesta HTTP

```
If-modified-since:
  <date>
```

- *server:* la risposta non contiene l'oggetto se la copia nella cache è aggiornata:

```
HTTP/1.0 304 Not
  Modified
```



Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

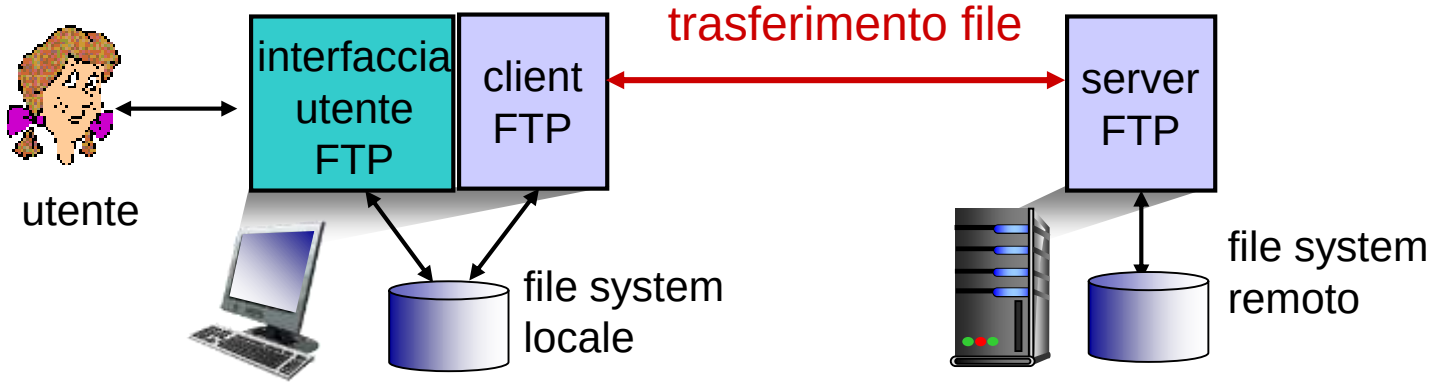
- SMTP, POP3, IMAP

2.5 DNS

2.6 applicazioni P2P

2.7 programmazione delle socket con UDP e TCP

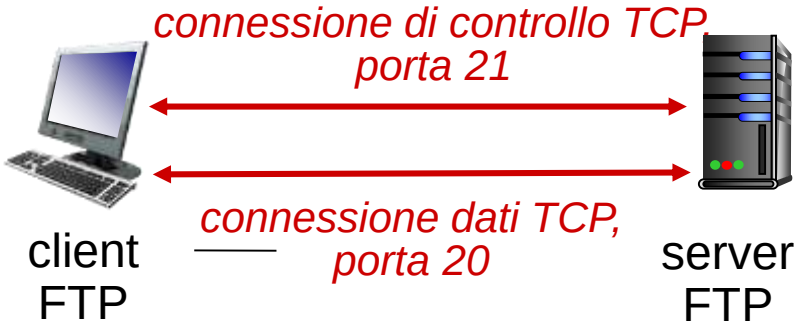
FTP: file transfer protocol



- trasferimento file da/verso un host remoto
- modello client/server
  - *client:* chi richiede il trasferimento (da/verso un host remoto)
  - *server:* host remoto
- ftp: RFC 959
- server ftp : porta 21

FTP: controllo separato, connessioni dati

- il client FTP contatta il server FTP alla porta 21, specificando TCP come protocollo di trasporto
- il client ottiene l'autorizzazione sulla connessione di controllo
- il client sceglie la directory remota inviando i comandi sulla connessione di controllo
- quando il server riceve un comando per trasferire un file, il *server* apre una 2^ connessione TCP verso il client per i dati (file)
- dopo il trasferimento di un file, il server chiude la connessione dati



- il server apre un'altra connessione dati TCP per trasferire un altro file
- connessione di controllo: *"out of band"*
- il server FTP mantiene lo "stato": directory corrente, autenticazione effettuata

Comandi e risposte FTP

alcuni comandi:

- inviati come testo ASCII sulla connessione di controllo
- **`USER`** ***`username`***
- **`PASS`** ***`password`***
- **`LIST`** ritorna l'elenco dei file della directory corrente
- **`RETR filename`** recupera (get) un file dalla directory corrente
- **`STOR filename`** memorizza (put) un file sull'host remoto

alcuni codici di ritorno:

- codice di stato ed espressione (come in HTTP)
- **`331 Username OK, password required`**
- **`125 data connection already open; transfer starting`**
- **`425 Can't open data connection`**
- **`452 Error writing file`**

Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

- SMTP, POP3, IMAP

2.5 DNS

2.6 applicazioni P2P

2.7 programmazione delle socket con UDP e TCP

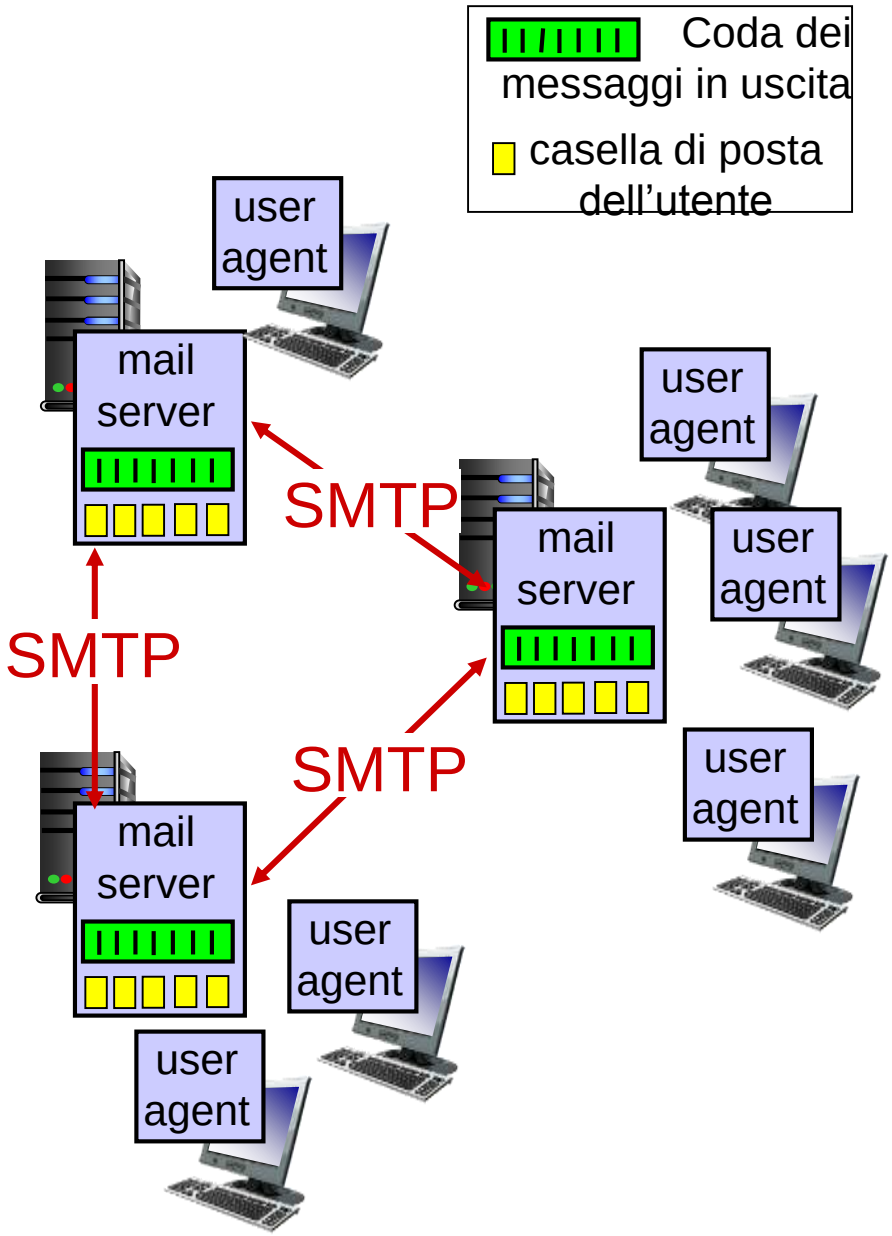
Posta elettronica

Tre component principali:

- user agent
- server di posta (mail server)
- simple mail transfer protocol: SMTP

User Agent

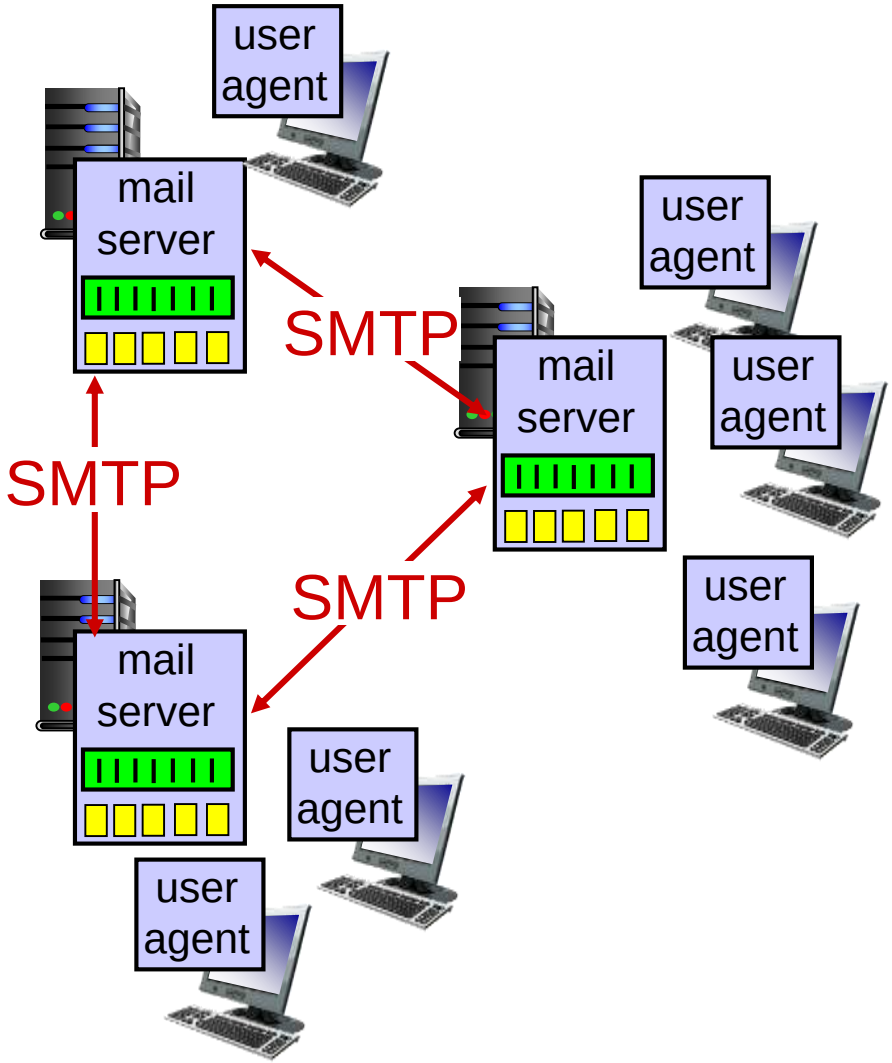
- il "mail reader"
- composizione, editing, lettura dei messaggi di posta elettronica
- es., Outlook, Thunderbird, iPhone mail client
- i messaggi in uscita o in arrivo sono memorizzati sul server



Posta elettronica: mail server

mail server:

- *mailbox* (casella di posta) contiene i messaggi in arrivo per l'utente
- *coda dei messaggi* da trasmettere
- *protocollo SMTP* tra mail server per inviare messaggi di posta elettronica
  - client: mail server che invia
  - "server": mail server che riceve

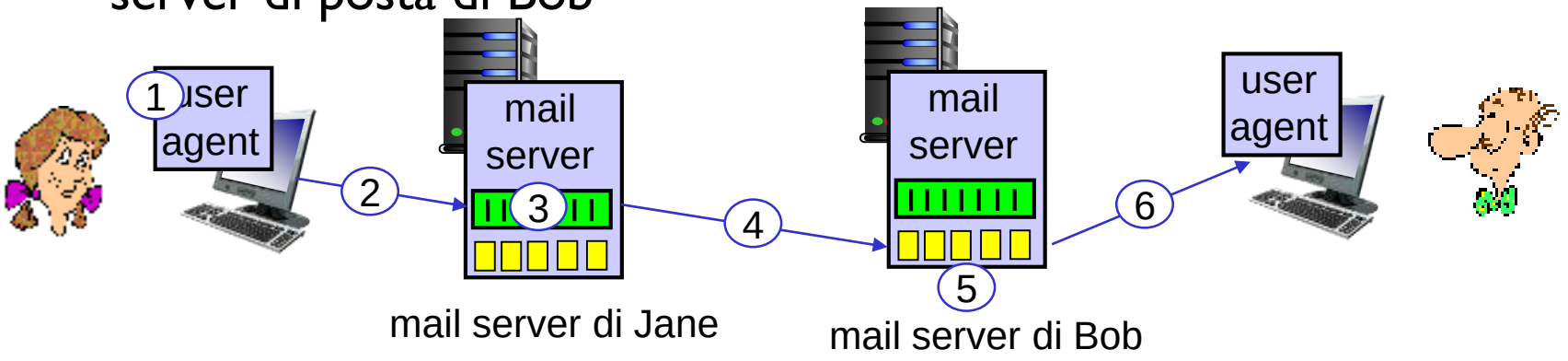


Posta elettronica: SMTP [RFC 2821]

- usa TCP per trasferire in modo affidabile i messaggi di posta elettronica dal client al server, porta 25
- trasferimento diretto: dal server trasmittente al server ricevente
- tre fasi per il trasferimento
  - handshaking (saluto)
  - trasferimento dei messaggi
  - chiusura
- interazione comando/risposta (come HTTP e FTP)
  - comandi: testo ASCII
  - risposta: codice di stato ed espressione
- i messaggi devono essere nel formato ASCII a 7 bit

Scenario: Jane invia un messaggio a Bob

1) Jane usa il suo user agent per comporre il messaggio da inviare "a" `bob@someschool.edu`
2) l'UA di Jane invia un messaggio al server di posta di Jane; il messaggio è posto nella coda di messaggi
3) il lato client di SMTP apre una connessione TCP con il server di posta di Bob
4) il client SMTP invia il messaggio di Jane sulla connessione TCP
5) il server di posta di Bob pone il messaggio nella casella di posta di Bob
6) Bob invoca il suo user agent per leggere il messaggio



Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <jane@crepes.fr>
S: 250 jane@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Per provare un'interazione SMTP:

- **`telnet servername 25`**
- riceverete la risposta 220 dal server
- digitate i comandi HELO, MAIL FROM, RCPT TO, DATA, QUIT

questo consente di inviare messaggi di posta elettronica senza usare il client di posta

SMTP: note finali

- SMTP usa connessioni persistenti
- SMTP richiede che il messaggio (intestazione e corpo) sia nel formato ASCII a 7 bit
- il server SMTP usa `CRLF.CRLF` per determinare la fine del messaggio

confronto con HTTP:

- HTTP: pull
- SMTP: push
- entrambi hanno un'interazione comando/risposta in ASCII, codici di stato
- HTTP: ogni oggetto è incapsulato nel suo messaggio di risposta
- SMTP: più oggetti vengono trasmessi in un unico messaggio

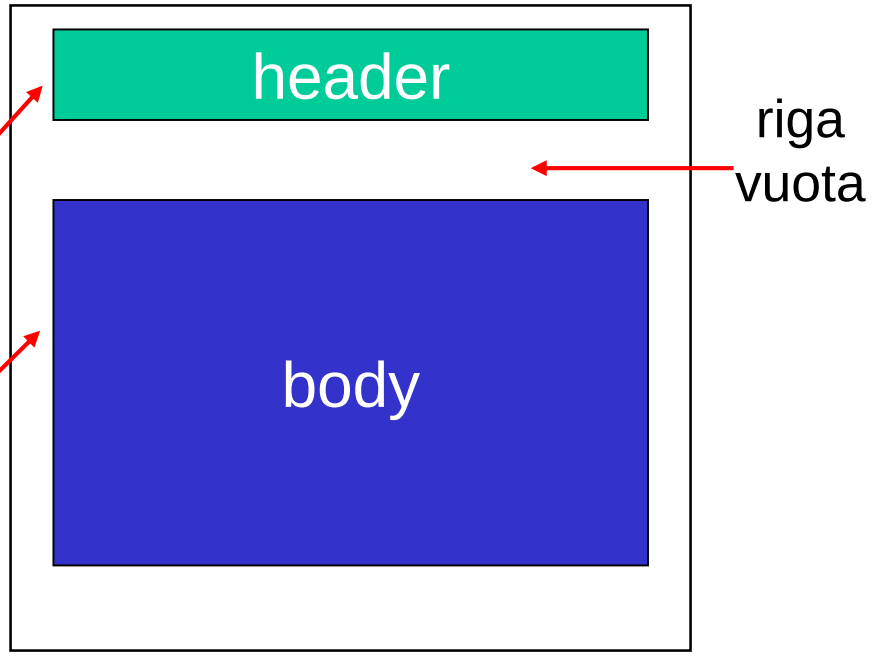
Formato dei messaggi

SMTP: protocollo per scambiare messaggi di posta elettronica

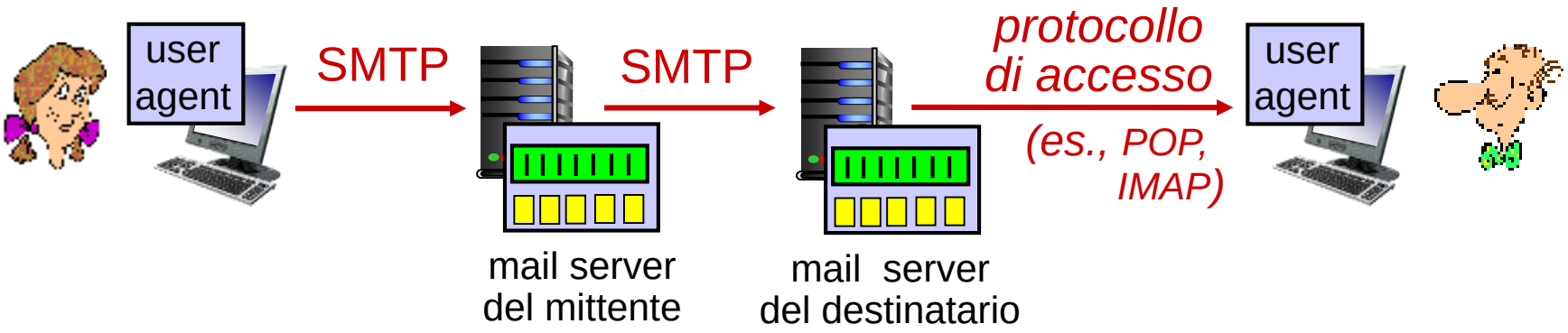
RFC 822: standard per il formato dei messaggi di testo:

- Righe di intestazione, es.,
  - To:
  - From:
  - Subject:

  differenti dai comandi SMTP MAIL FROM, RCPT TO: !
- Body: il "messaggio"
  - solo caratteri ASCII



Protocolli di accesso alla posta



- SMTP: consegna/memorizzazione sul server del destinatario
- protocollo di accesso alla posta: ottenere i messaggi dal server
  - POP: Post Office Protocol [RFC 1939]: autorizzazione e download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: più funzioni, inclusa la manipolazione di messaggi memorizzati sul server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Protocollo POP3

fase di autorizzazione

- comandi client:
  - **`user`**: dichiara lo username
  - **`pass`**: password
- risposte del server
  - **`+OK`**
  - **`-ERR`**

fase delle transazioni client:

- **`list`**: elenca i numeri dei messaggi
- **`retr`**: ottiene i messaggi per numero
- **`dele`**: cancella
- **`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
```

POP3 (altro) e IMAP

ancora su POP3

- il precedente esempio usa la modalità POP3 "scarica e cancella"
  - Bob non può rileggere le e-mail se cambia client
- modalità "scarica e mantieni": permette di copiare i messaggi su più client
- POP3 è un protocollo senza stato tra le varie sessioni

IMAP

- mantiene sincronizzati i messaggi sul client e sul server
- consente all'utente di organizzare i messaggi in cartelle
- IMAP conserva lo stato dell'utente tra le varie sessioni:
  - i nomi delle cartelle e l'associazione tra identificatori dei messaggi e nomi delle cartelle

Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

- SMTP, POP3, IMAP

2.5 DNS

2.6 applicazioni P2P

2.7 programmazione delle socket con UDP e TCP

DNS: domain name system

persone: molti identificatori:

- CF, nome, # passaporto

host e router di Internet:

- indirizzo IP (32 bit) - usato per indirizzare i datagrammi
- "nome", es., www.yahoo.com - usato dagli esseri umani

D: come associare un indirizzo IP a un nome, e viceversa?

Domain Name System:

- *database distribuito* implementato in una gerarchia di numerosi *name server*
- *Protocollo a livello di applicazione:* host, name server comunicano per *risolvere* i nomi (traduzione indirizzo/nome)
  - nota: funzione critica di Internet implementata come protocollo a livello di applicazione
  - complessità nelle parti periferiche della rete

DNS: servizi, struttura

servizi DNS

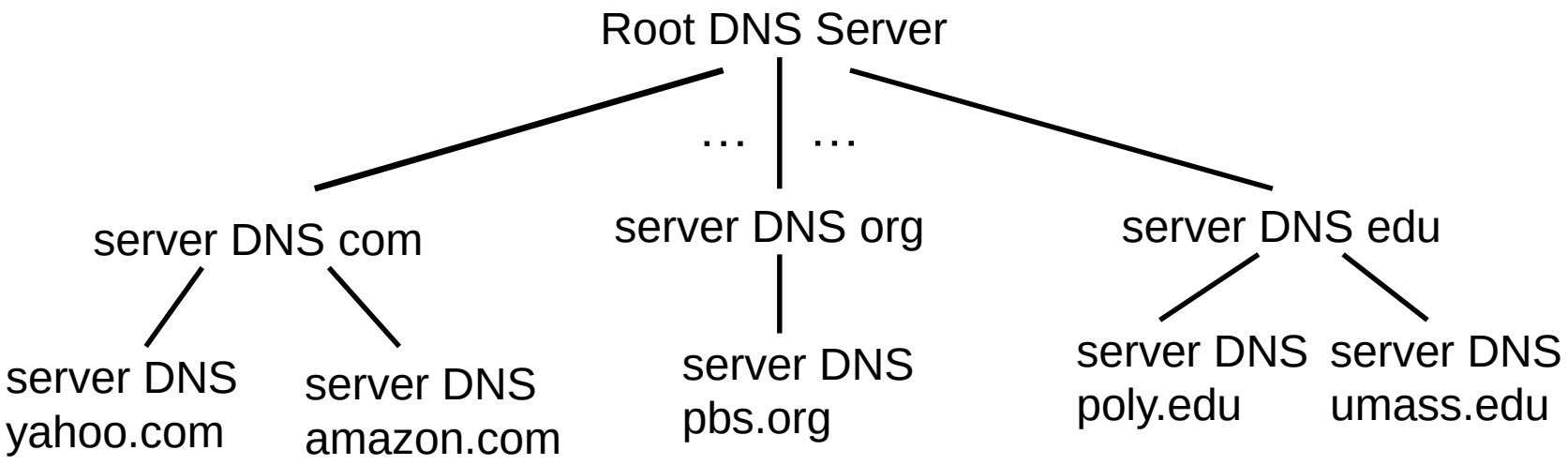
- traduzione degli hostname in indirizzi IP
- host aliasing
  - un host può avere un nome canonico più vari alias
- mail server aliasing
- distribuzione del carico
  - server web replicati: insieme di indirizzi IP con un nome canonico

D: perché non centralizzare DNS?

- single point of failure
- volume di traffico
- database centralizzato distante
- manutenzione

R: non è scalabile!

DNS: un database distribuito e gerarchico

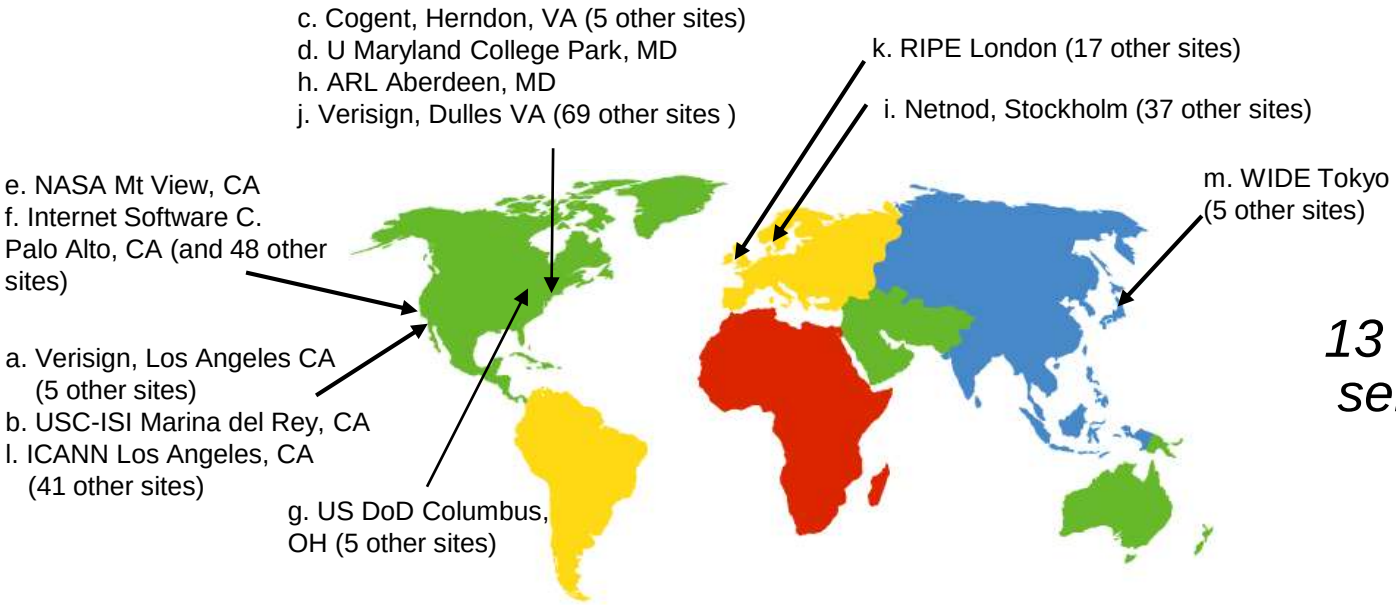


il client vuole l'IP di www.amazon.com; 1^ approssimazione:

- il client interroga il root server per trovare il server DNS com
- il client interroga il server DNS com per ottenere il server DNS amazon.com
- il client interroga il server DNS amazon.com per ottenere l'indirizzo IP di www.amazon.com

DNS: root name server

- contattato da un server DNS locale che non può tradurre un nome
- root name server:
  - contatta un server DNS autoritativo se non ha il mapping del nome
  - ottiene il mapping
  - restituisce il mapping al server DNS locale



13 root name server nel mondo

TLD, server autoritativi

server top-level domain (TLD) :

- responsabili dei domini com, org, net, edu, aero, jobs, museums, e di tutti i top-level country domain, es.: uk, fr, ca, jp
- Network Solutions gestisce i servers TLD per il dominio .com
- Educause quelli per il dominio .edu

server DNS autoritativi:

- DNS server propri di un'organizzazione, che forniscono autoritativamente il mapping hostname-IP per gli host (che necessitano di un nome) dell'organizzazione
- possono essere mantenuti dall'organizzazione stessa o da un service provider

DNS server locale

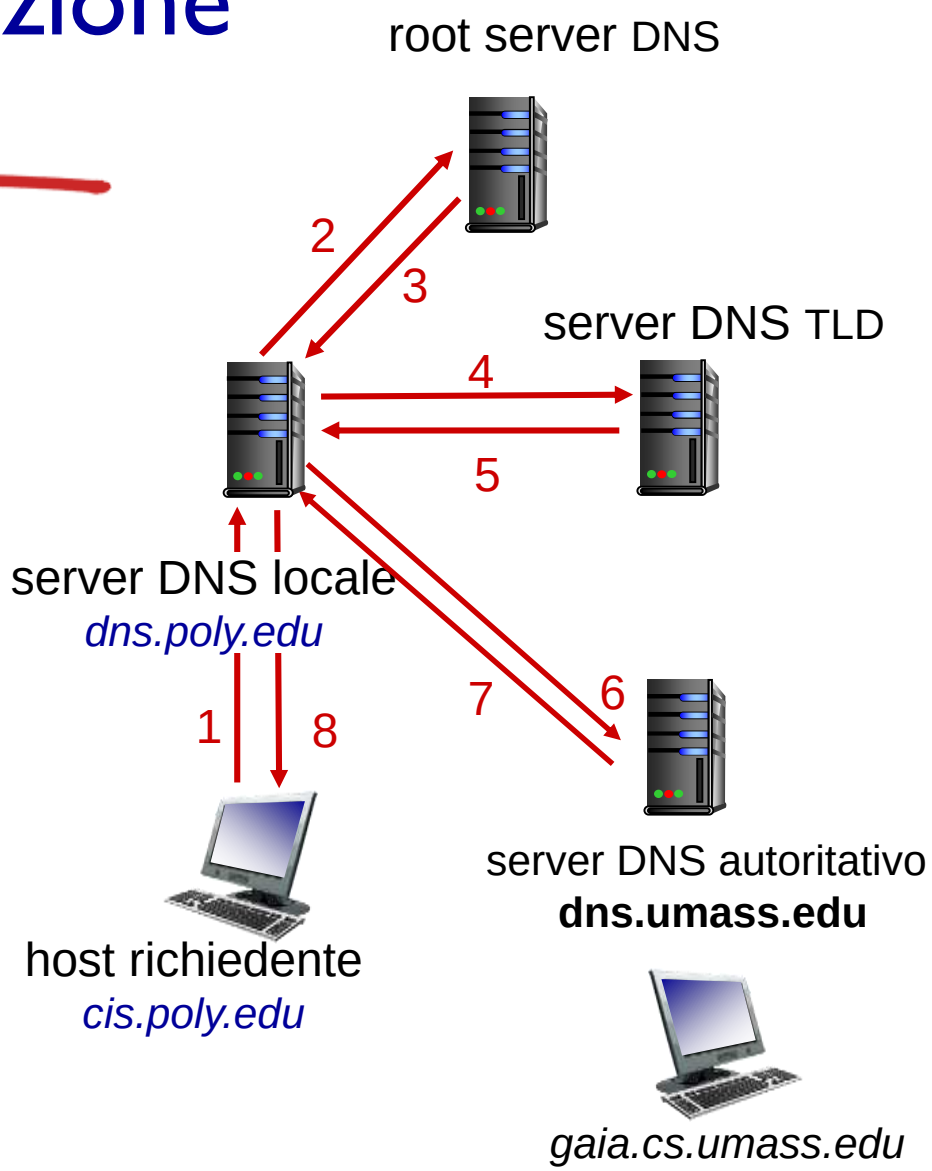
- non appartiene strettamente alla gerarchia dei server
- ogni ISP (ISP residenziale, azienda, università) ne ha uno
  - detto anche "default name server"
- quando un host effettua una richiesta DNS, la query viene inviata al suo server DNS locale
  - o risolve il nome tramite la sua cache locale che contiene le recenti traduzioni name-to-address (che potrebbero essere non più valide!)
  - o agisce come un proxy, inoltrando la richiesta lungo la gerarchia dei server DNS

Esempio di risoluzione DNS di un nome

- un host nel dominio cis.poly.edu vuole l'indirizzo IP di gaia.cs.umass.edu

query iterativa:

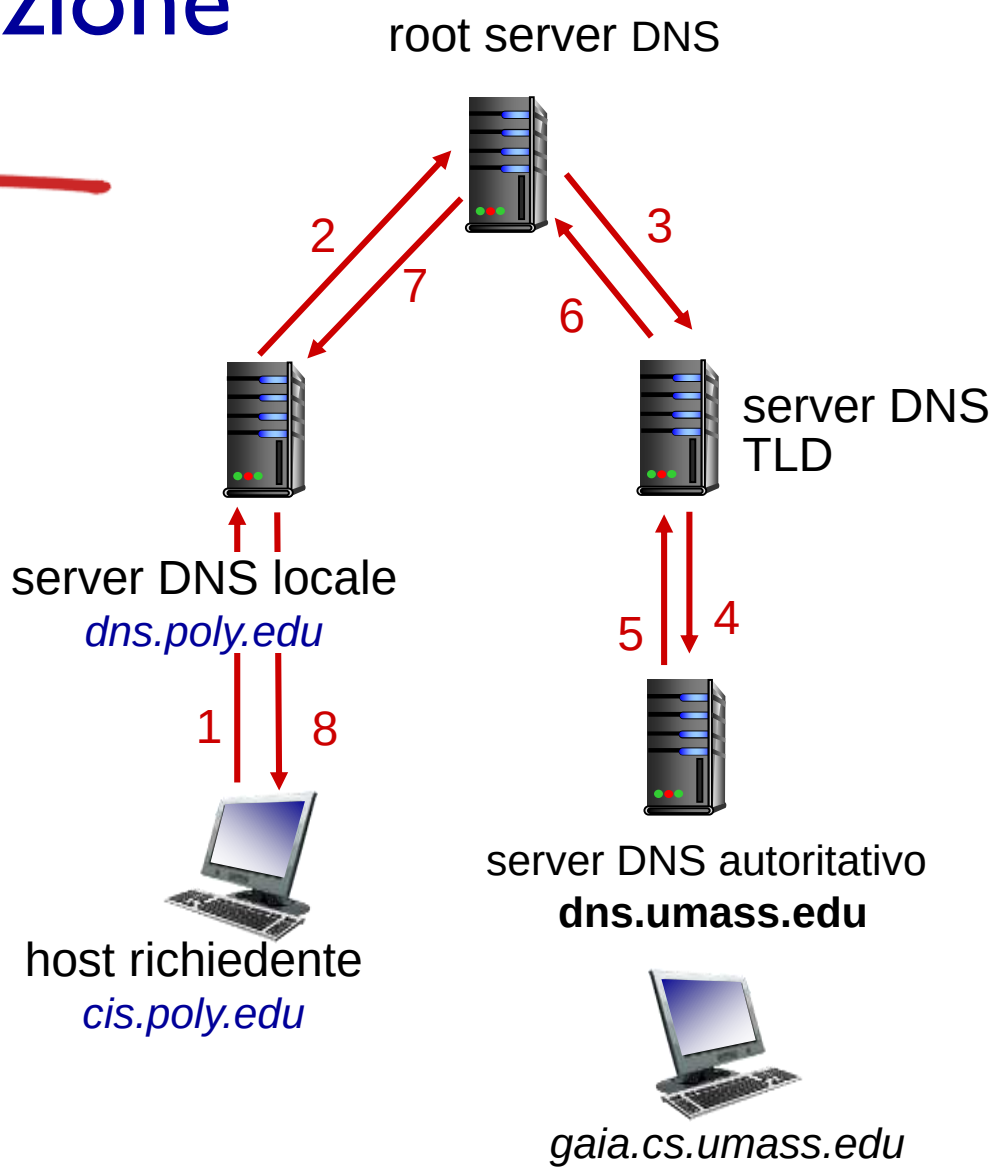
- il server contattato risponde con il nome cercato
- "Non so il nome, chiedi a questo server"



Esempio di risoluzione DNS di un nome

query ricorsiva:

- affida il compito di tradurre il nome al server DNS contattato
- grosso carico per gli alti livelli della gerarchia



DNS: cache e aggiornamento dei record

- una volta che un server DNS impara un mapping, lo mette nella *cache*
  - le informazioni nella cache vengono invalidate (spariscono) dopo un certo periodo di tempo (TTL)
  - tipicamente un server DNS locale memorizza nella cache gli indirizzi IP dei server TLD
    - quindi i root name server non vengono visitati spesso
- i dati nella cache potrebbero essere *out-of-date* (scaduti)
  - se un host cambia indirizzo IP, il cambiamento potrebbe non essere appreso subito da tutta Internet fino a quando tutti i TTL si esauriscono
- i meccanismi di aggiornamento/notifica sono progettati da IETF
  - RFC 2136

Record DNS

DNS: database distribuito che memorizza i record di risorsa (RR)

formato RR : **(name, value, type, ttl)**

type=A

- **name** è il nome dell'host
- **value** è l'indirizzo IP

type=NS

- **name** è il dominio (es., foo.com)
- **value** è il nome dell'host del server autoritativo di quel dominio

type=CNAME

- **name** è un alias di qualche nome "canonico" (nome vero)
- **www.ibm.com** è in realtà **servereast.backup2.ibm.com**
- **value** è il nome canonico

type=MX

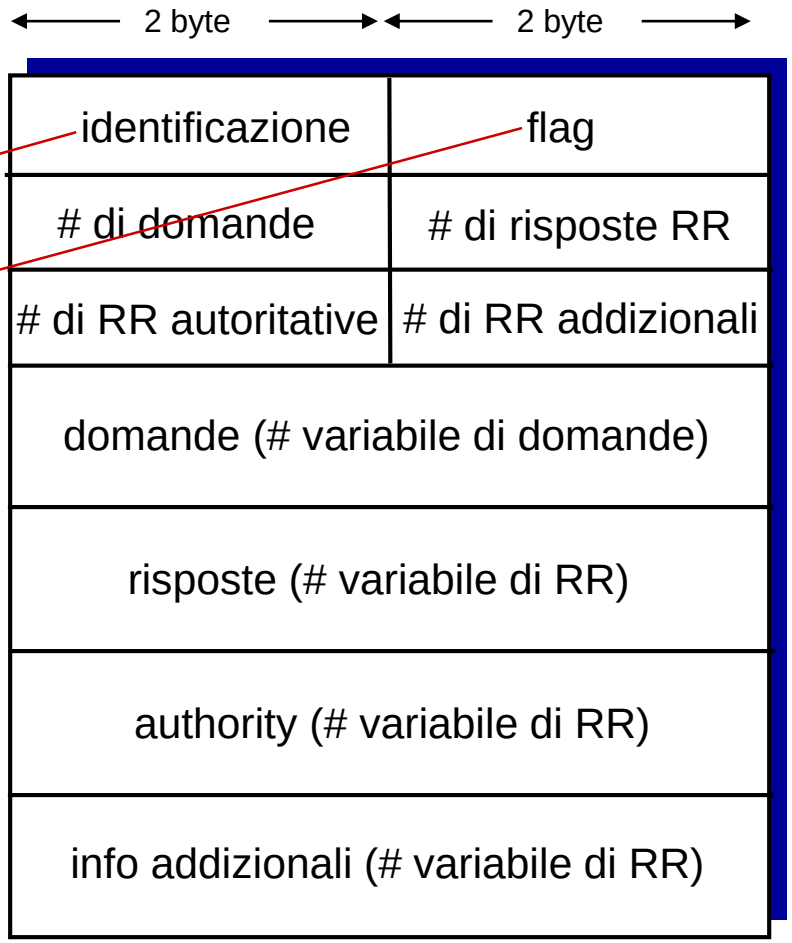
- **value** è il nome del server di posta associato a **name**

Protocollo e messaggi DNS

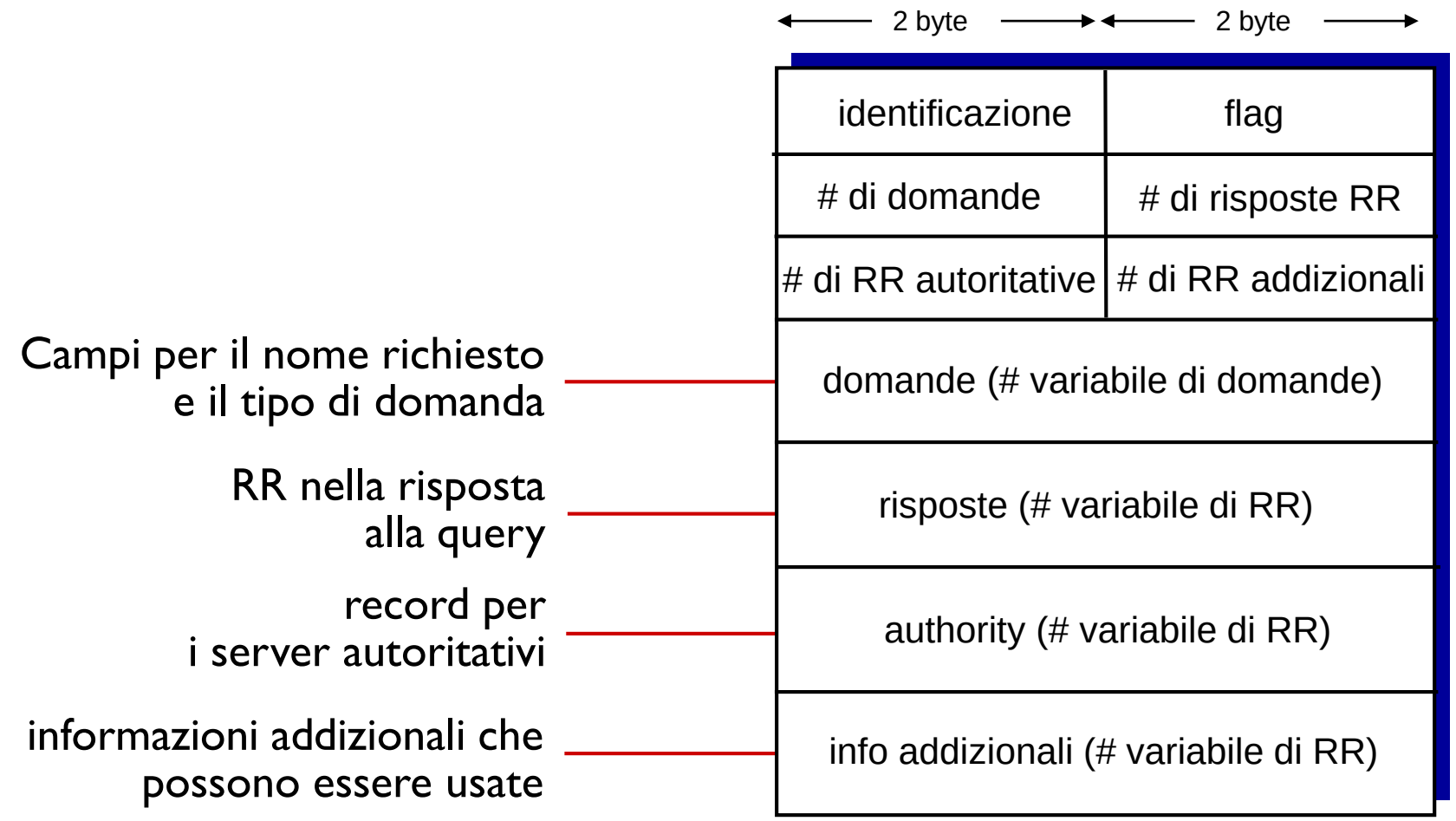
- messaggi di *query* e di *reply*, entrambi con lo stesso *formato*

intestazione del messaggio

- identificazione: numero di 16 bit per la query; il reply usa lo stesso numero
- flag:
  - query o reply
  - richiesta di ricorsione
  - ricorsione disponibile
  - il reply è autoritativo



Protocollo e messaggi DNS



Inserire record nel DNS

- esempio: c'è una nuova startup "Network Utopia"
- registriamo il nome networkuptopia.com presso un *DNS registrar* (es., Network Solutions)
  - forniamo nomi e indirizzi IP dei name server autoritativi (primario e secondario)
  - il registrar inserisce due RR nei TLD server di .com :
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- inseriamo nel server autoritativo un record tipo A per www.networkuptopia.com e un record tipo MX per networkutopia.com

Attacchi ai DNS

attacchi DDoS

- bombardare i root server con traffico
  - non di particolare successo grazie al filtraggio del traffico e al fatto che i server DNS locali "cachano" gli IP dei server TLD, consentendo di bypassare i root server
- bombardare i server TLD
  - potenzialmente più pericoloso

attacchi di ridirezionamento

- man-in-middle
  - intercetta le query
- DNS poisoning (avvelenamento)
  - invia risposte false al server DNS, che le mette in cache

sfruttare il DNS per DDoS

- si inviano query aventi come indirizzo IP richiedente quello dell'host da attaccare
- richiede amplificazione

Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

- SMTP, POP3, IMAP

2.5 DNS

2.6 applicazioni P2P

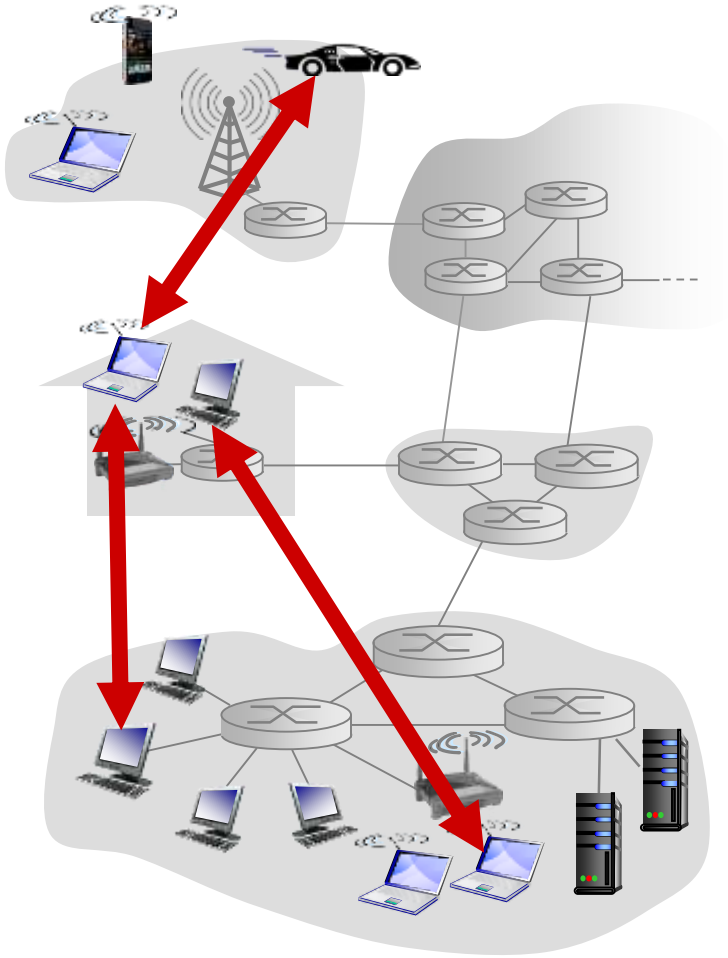
2.7 programmazione delle socket con UDP e TCP

Architettura P2P pura

- *nessun* server always-on
- end system arbitrari comunicano direttamente
- i peer non sono connessi permanentemente e cambiano indirizzo IP

esempi:

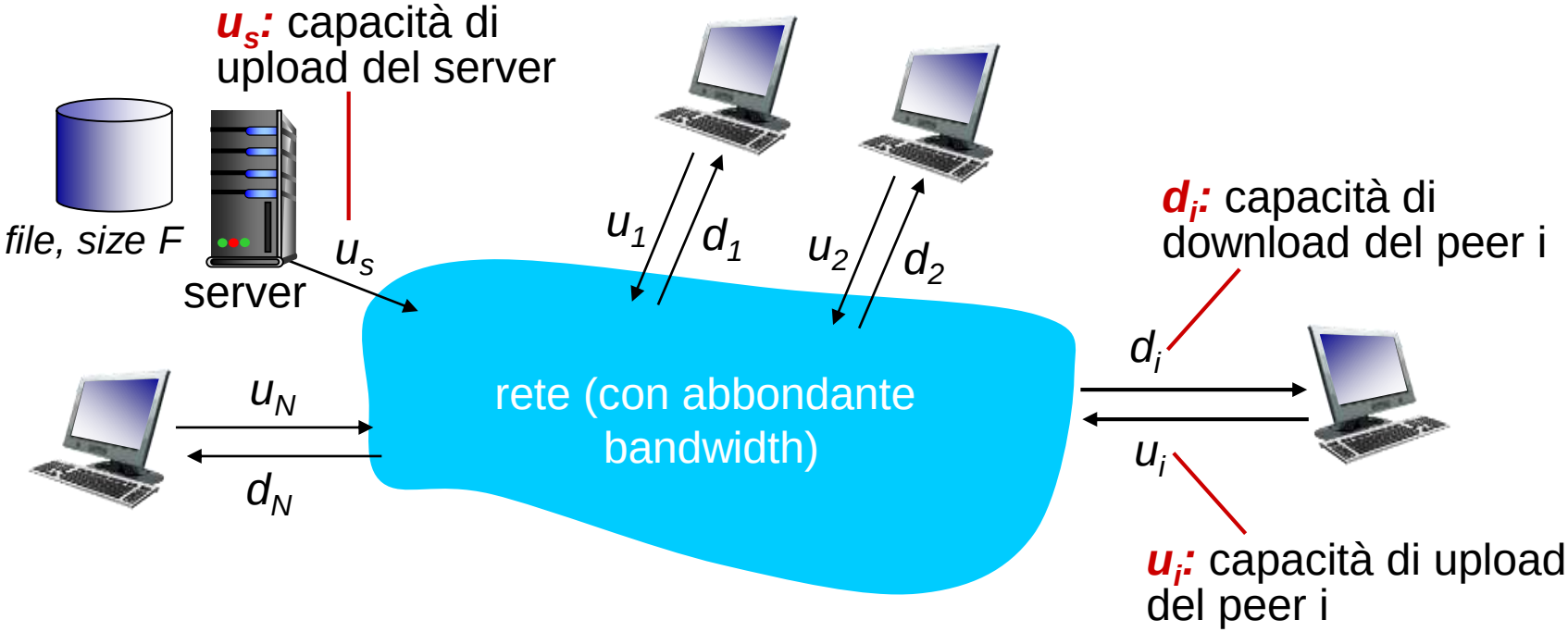
- distribuzione di file (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



Distribuzione di file : client-server vs P2P

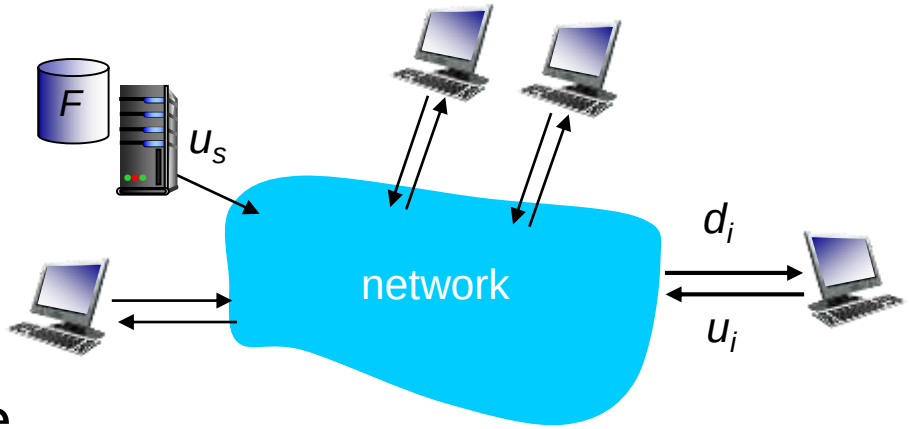
Domanda: quanto tempo occorre per distribuire un file (di dimensione *F*) da un server a *N* *peer*?

- la capacità di upload/download di un peer è una risorsa limitata



Tempo di distribuzione di file: client-server

- *trasmissione del server :* deve inviare in sequenza (upload) *N* copie del file :
  - tempo per una copia: F/u_s
  - tempo per N copie: NF/u_s



- *client:* ogni client deve scaricare (download) una copia del file
  - d_{min} = download rate più basso tra i client
  - tempo minimo di download del client: F/d_{min}

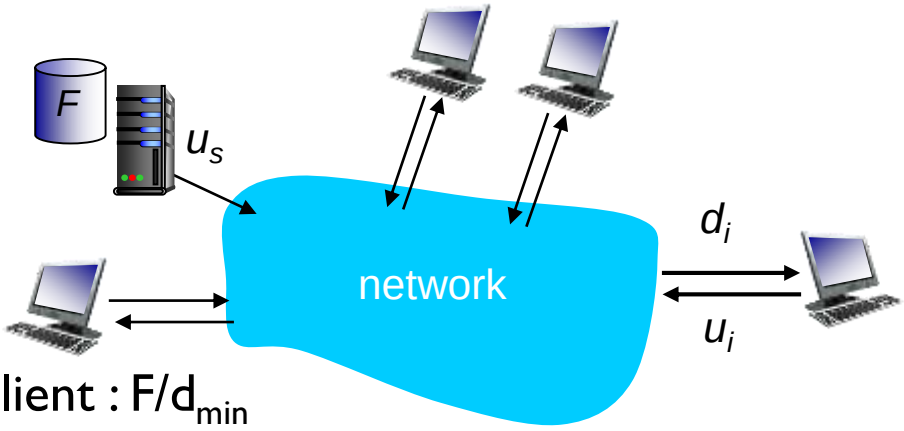
tempo per distribuire F a N client usando l'approccio client-server

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

cresce linearmente con N

Tempo di distribuzione di file : P2P

- *trasmissione del server:* deve upload-are almeno una copia
  - tempo per inviare una copia: F/u_s
- *client:* ogni client deve scaricare una copia del file
  - tempo minimo di download del client : F/d_{min}
- *tutti i client:* aggregati devono scaricare *NF* bit
  - rate massimo di upload (che è un limite per il massimo download rate) è $u_s + \Sigma u_i$



tempo per distribuire F a N client usando l'approccio P2P

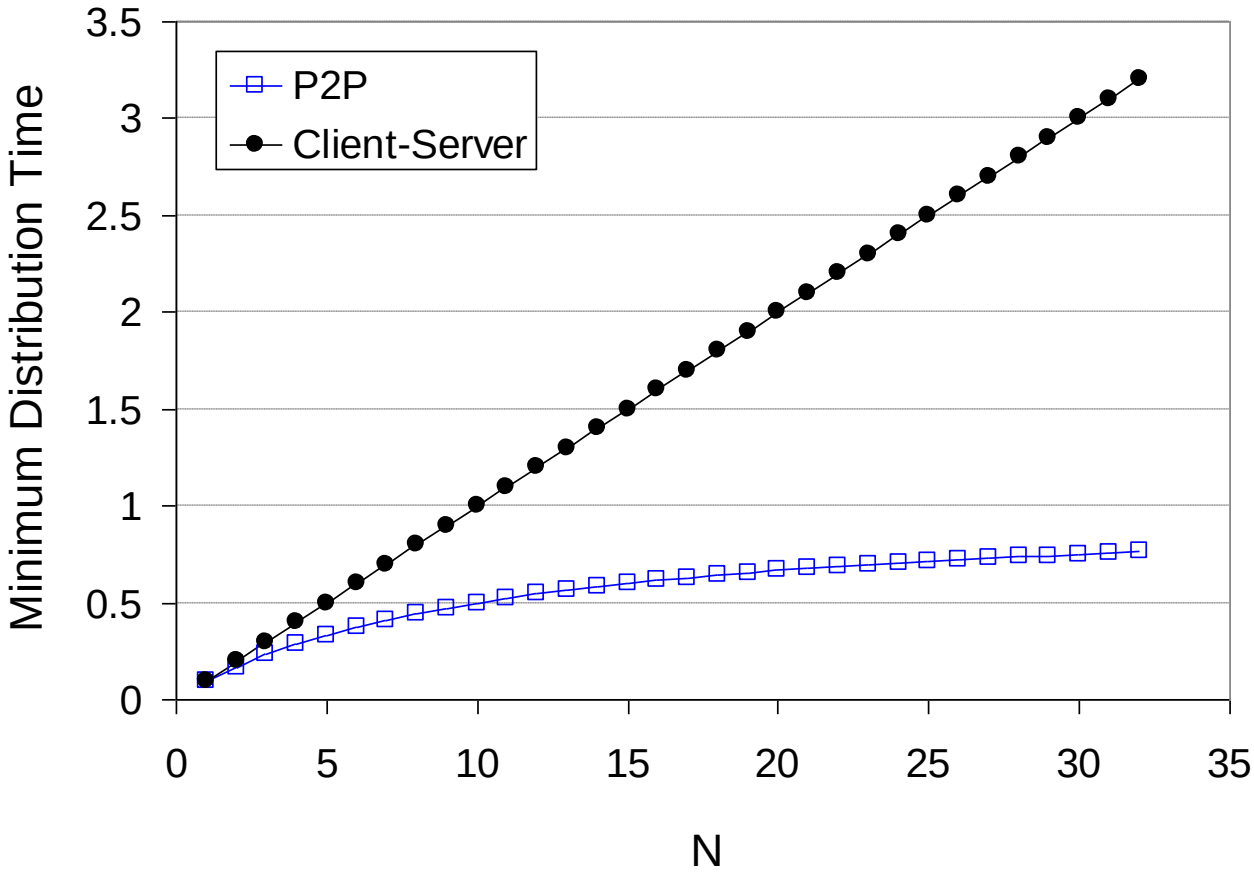
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

cresce linearmente con *N* …

… ma anche questo, poiché ogni peer aggiunge capacità al servizio

Client-server vs. P2P: example

upload rate del client = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

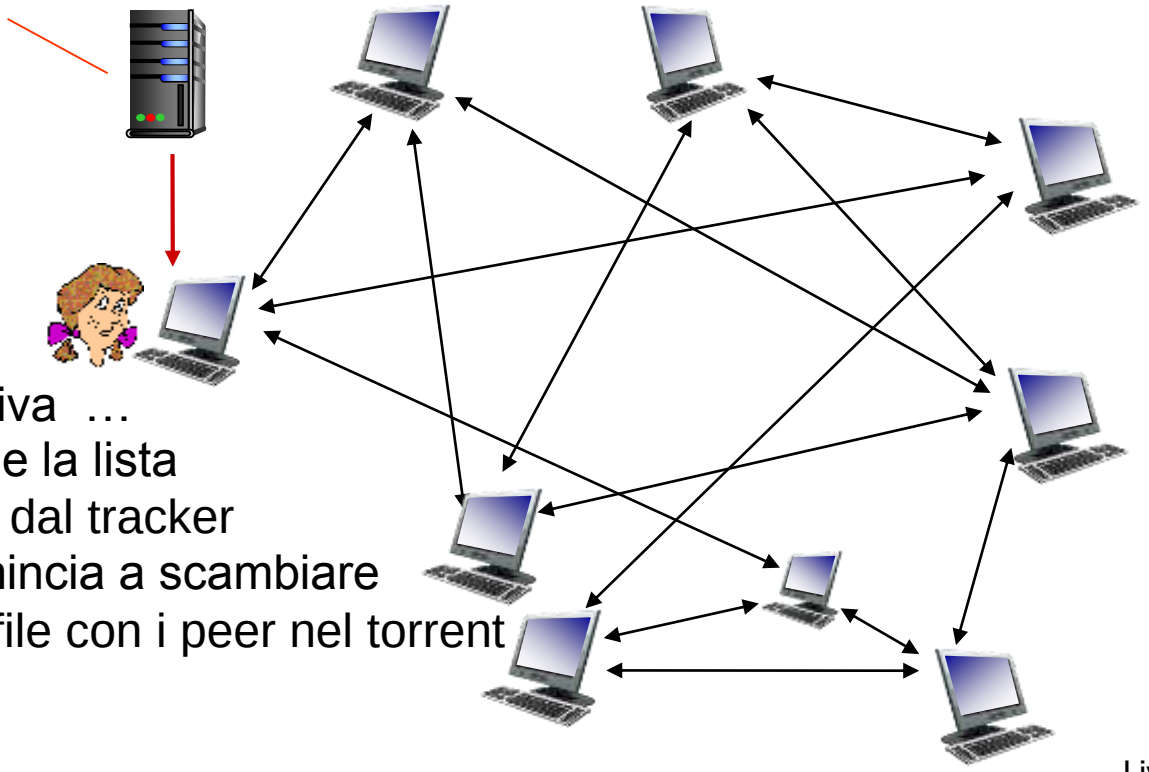


Distribuzione file P2P: BitTorrent

- file diviso in pezzi (chunk) da 256Kb
- il peer nel torrent invia/riceve pezzi di file

tracker: tiene traccia dei peer che partecipano al torrent

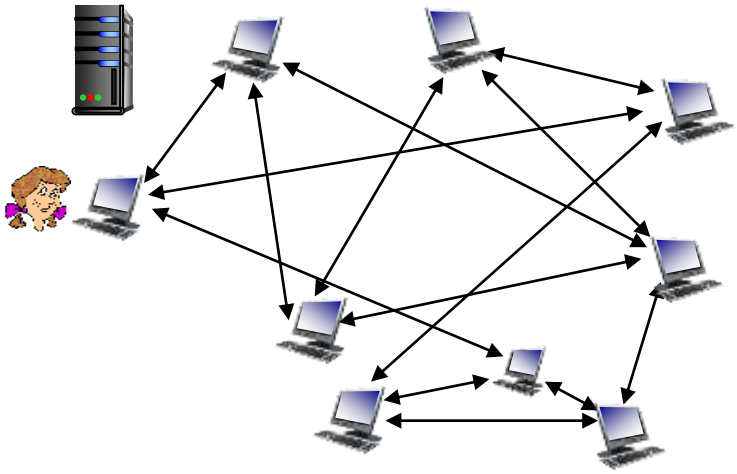
torrent: gruppo di peers che si scambiano pezzi di file



Jane arriva …
… ottiene la lista dei peer dal tracker
… e comincia a scambiare pezzi di file con i peer nel torrent

Distribuzione file P2P: BitTorrent

- i peer che si uniscono al torrent:
  - non hanno pezzi, ma li accumulano via via dagli altri peer
  - si registrano sul tracker per avere la lista dei peer, e si connettono a un sottoinsieme di peers ("neighbors")



- durante il download, i peer upload-ano pezzi agli altri peers
- un peer può cambiare i peer con i quali scambia pezzi
- *churn:* i peer possono andare e venire
- una volta che un peer ha l'intero file, può (egoisticamente) lasciare o (altruisticamente) restare nel torrent

BitTorrent: richiesta e invio di file chunks

richiesta di chunk:

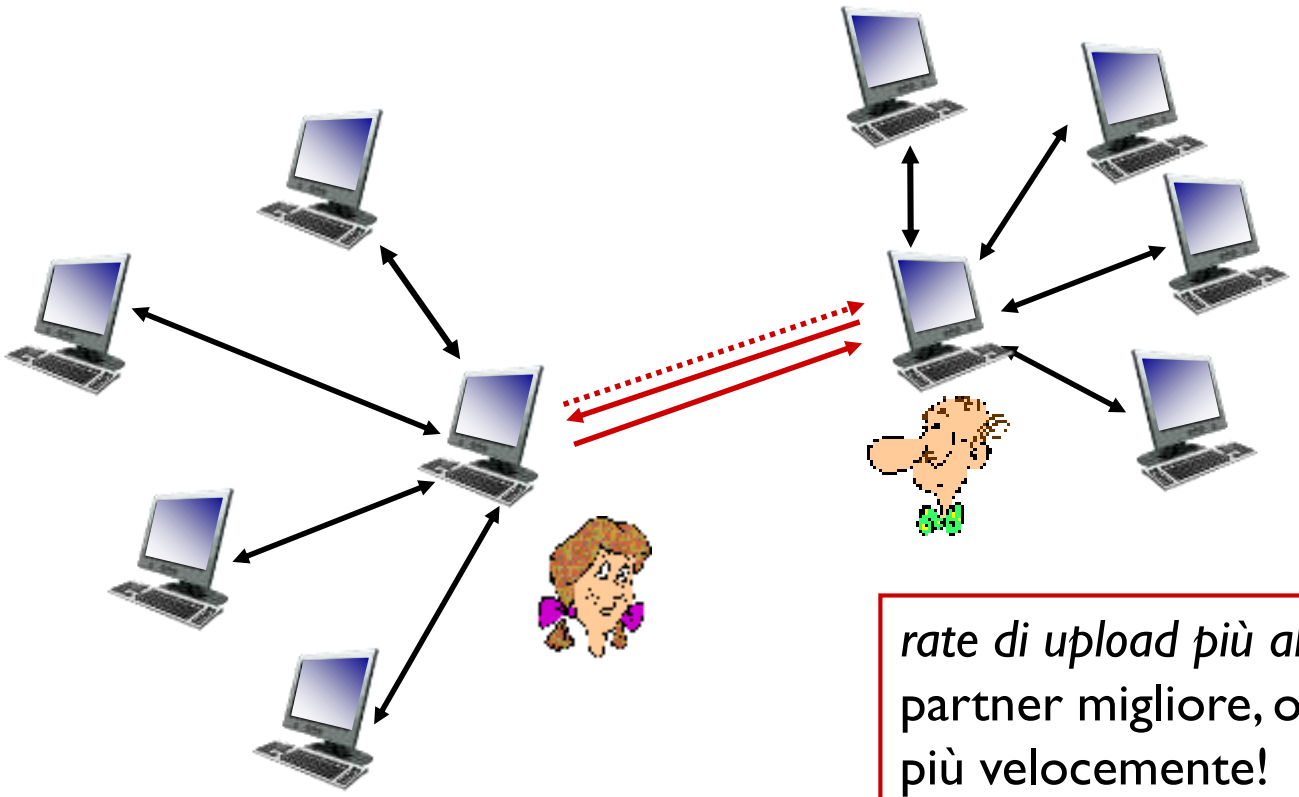
- ad ogni determinato istante, i diversi peer hanno differenti sottoinsiemi di pezzi di file
- periodicamente, Jane chiede a ogni peer la lista dei pezzi che hanno
- Jane richiede i pezzi mancanti dai peer, chiedendo per primi i meno diffusi (rarest first)

invio di chunks tit-for-tat

- Jane invia pezzi ai 4 peer che gli stanno inviando pezzi *con il rate più alto*
  - gli altri peer non vengono riforniti (choked) da Jane
  - rivaluta i top 4 ogni 10 sec
- ogni 30 sec: seleziona a caso un altro peer
  - "optimistically unchoke" questo peer
  - il nuovo peer scelto può unirsi ai top 4

BitTorrent: tit-for-tat

(1) Jane dà fiducia a ("optimistically unchokes") Bob
(2) Jane diventa uno dei top 4 fornitori di Bob; Bob ricambia
(3) Bob diventa uno dei top 4 fornitori di Jane



rate di upload più alto: cerca il partner migliore, ottiene il file più velocemente!

Distributed Hash Table (DHT)

- Hash table
- Paradigma DHT
- DHT circolari e overlay network
- Peer churn

Semplice Database

Semplice database con coppie (key, value):

- key: nome; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
.....	.....
Lisa Kobayashi	177-23-0199

- key: titolo di un film; value: indirizzo IP

Hash Table

- E' più conveniente mantenere e cercare rappresentazioni numeriche della chiave
- key = hash(key originale)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....		.....
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

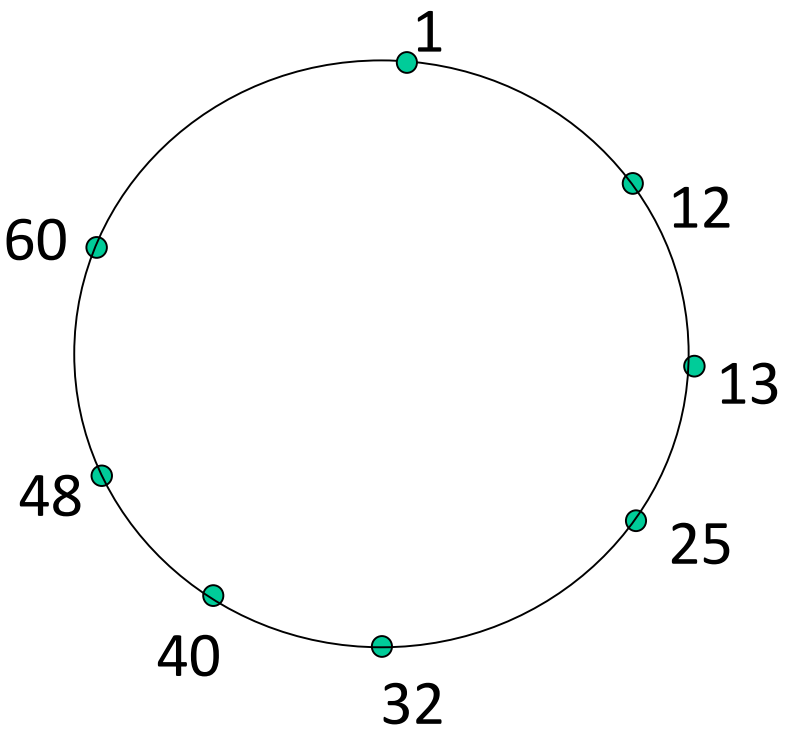
- Le coppie (key, value) sono distribuite su milioni di peer
  - le coppie sono equamente distribuite sui peer
- Ogni peer può interrogare (query) il database con una key
  - il database restituisce il value per la key
  - per eseguire la query, un piccolo numero di messaggi viene scambiato tra i peer
- Ogni peer è a conoscenza di un piccolo numero di peer
- Robusto alle entrate e uscite dei peer (churn)

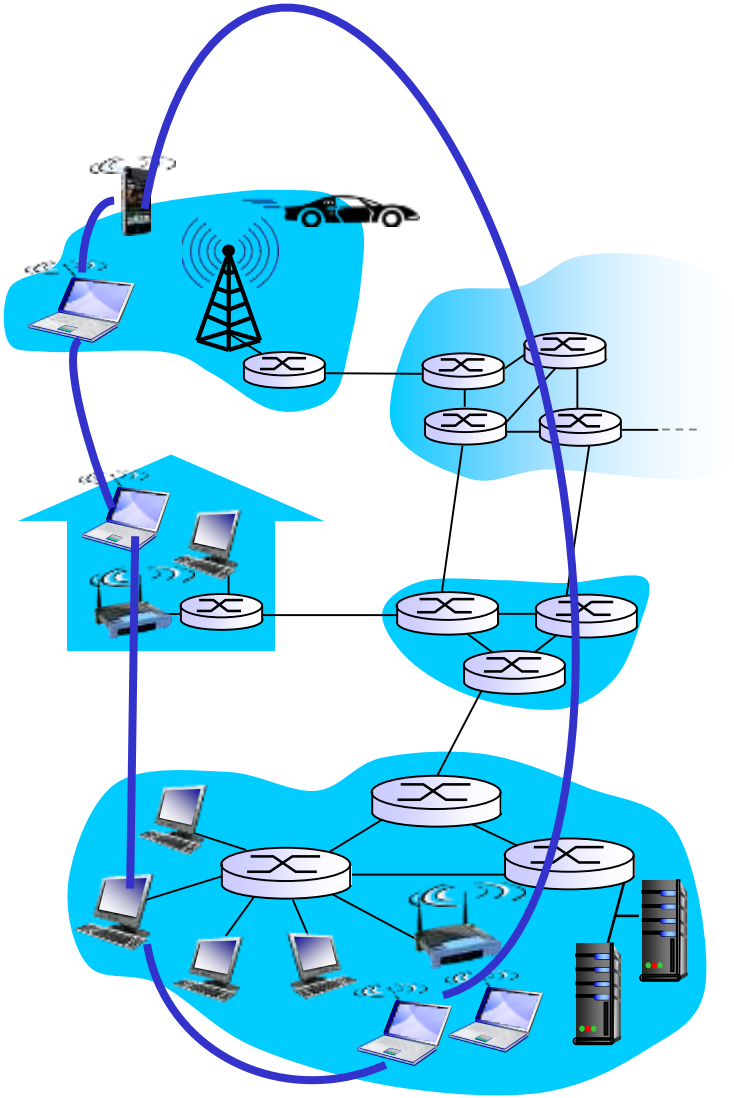
Assegnazione di coppie key-value ai peer

- regola: si assegna la coppia key-value al peer che ha l'ID *più vicino*.
- convenzione: per più vicino s'intende l'*immediato successore* della key.
- es., lista ID {0,1,2,3,…,63}
- supponiamo 8 peer: 1,12,13,25,32,40,48,60
  - Se key = 51, allora assegnala al peer 60
  - Se key = 60, allora assegnala al peer 60
  - Se key = 61, allora assegnala al peer 1

DHT circolare

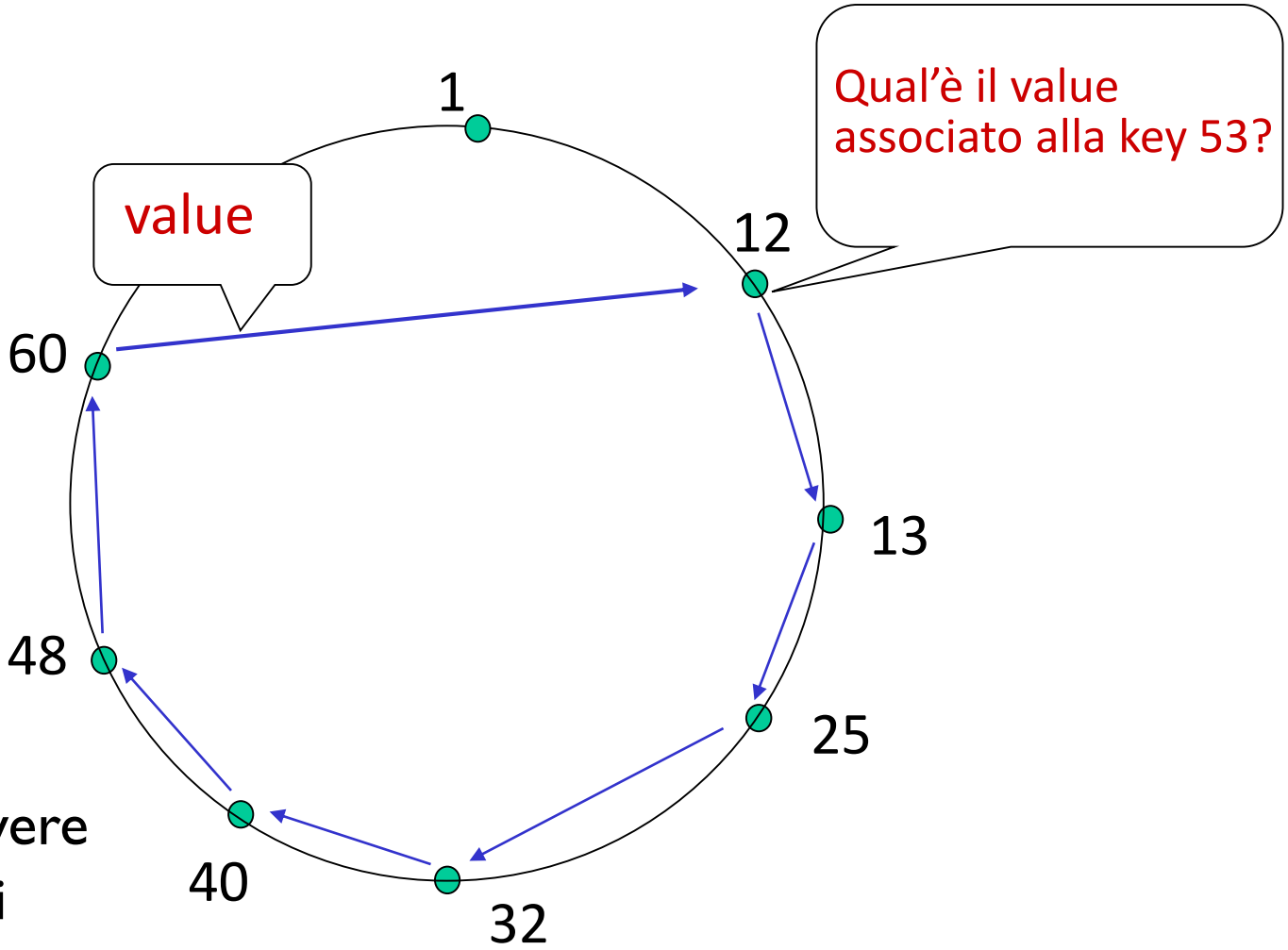
- ogni peer è a conoscenza *solo* degli immediati successore e predecessore.





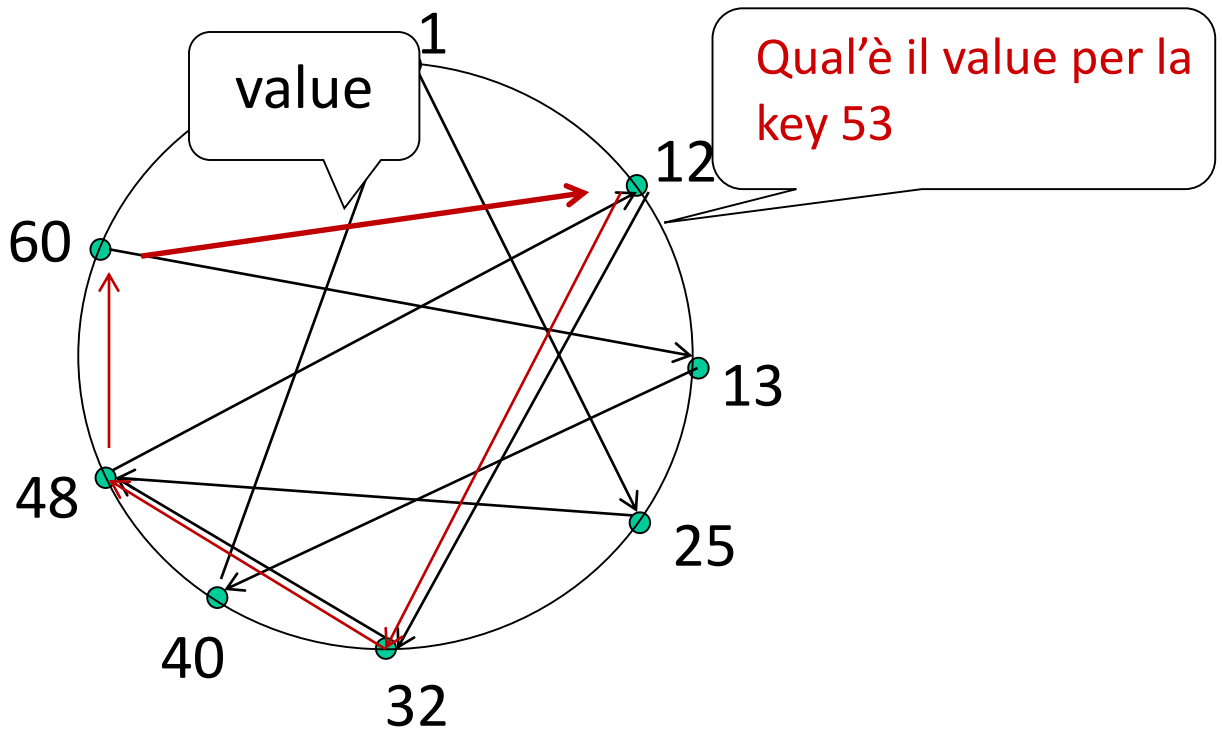
"rete di copertura"

Risolvere una query



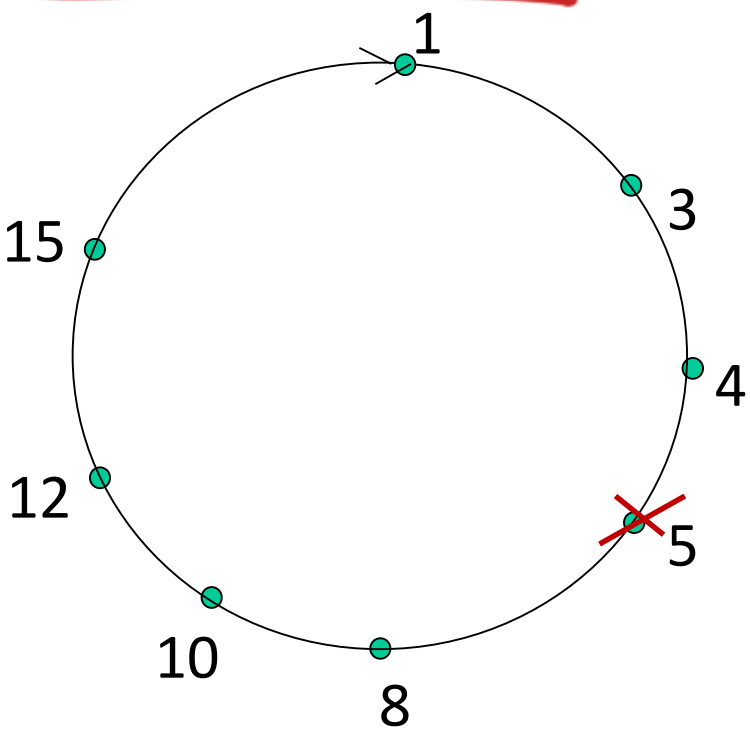
O(N) messaggi in media per risolvere la query, quando ci sono *N* peer

DHT circolare con shortcuts



- ogni peer tiene traccia degli indirizzi IP del predecessore, successore, e qualche altro (shortcut).
- messaggi ridotti da 6 a 3.
- è possibile tracciare shortcuts con *O(log N)* neighbor, *O(log N)* messaggi nella query

Peer churn

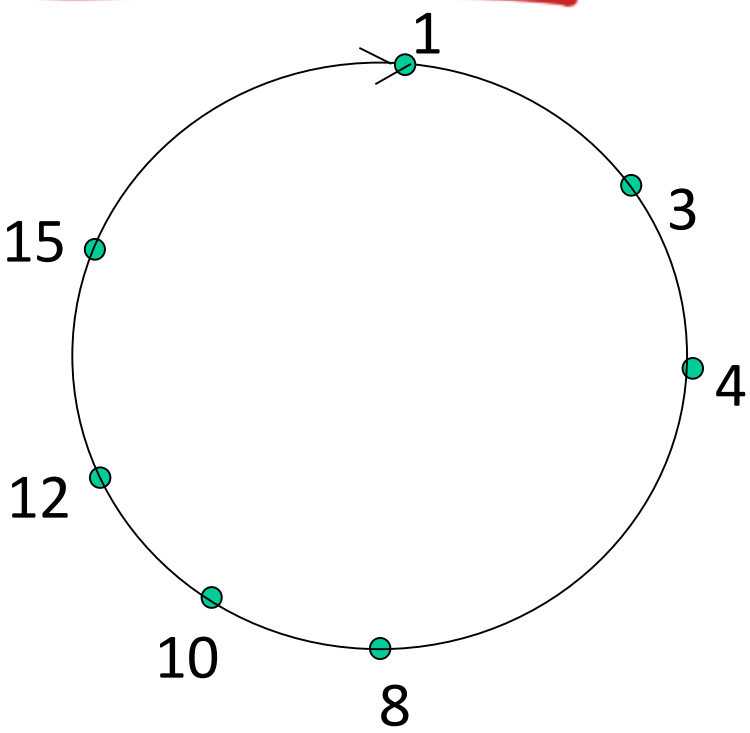


gestione del peer churn:

- i peer possono andare e venire (churn)
- ogni peer sa l'indirizzo dei suoi due successori
- ogni peer periodicamente pinga i suoi due successori per vedere se sono attivi
- se l'immediato successore lascia, scegli il successore seguente come nuovo immediato successore

esempio: il peer 5 lascia bruscamente

Peer churn



gestione del peer churn:

- i peer possono andare e venire (churn)
- ogni peer sa l'indirizzo dei suoi due successori
- ogni peer periodicamente pinga i suoi due successori per vedere se sono attivi
- se l'immediato successore lascia, scegli il successore seguente come nuovo immediato successore

esempio: il peer 5 lascia bruscamente

- il peer 4 rileva la partenza del peer 5; prende l'8 come immediato successore
- 4 chiede a 8 chi è il suo immediato successore; prende l'immediato successore di 8 come secondo successore.

Capitolo 2: livello di applicazione

2.1 principi delle applicazioni di rete

2.2 Web e HTTP

2.3 FTP

2.4 posta elettronica

- SMTP, POP3, IMAP

2.5 DNS

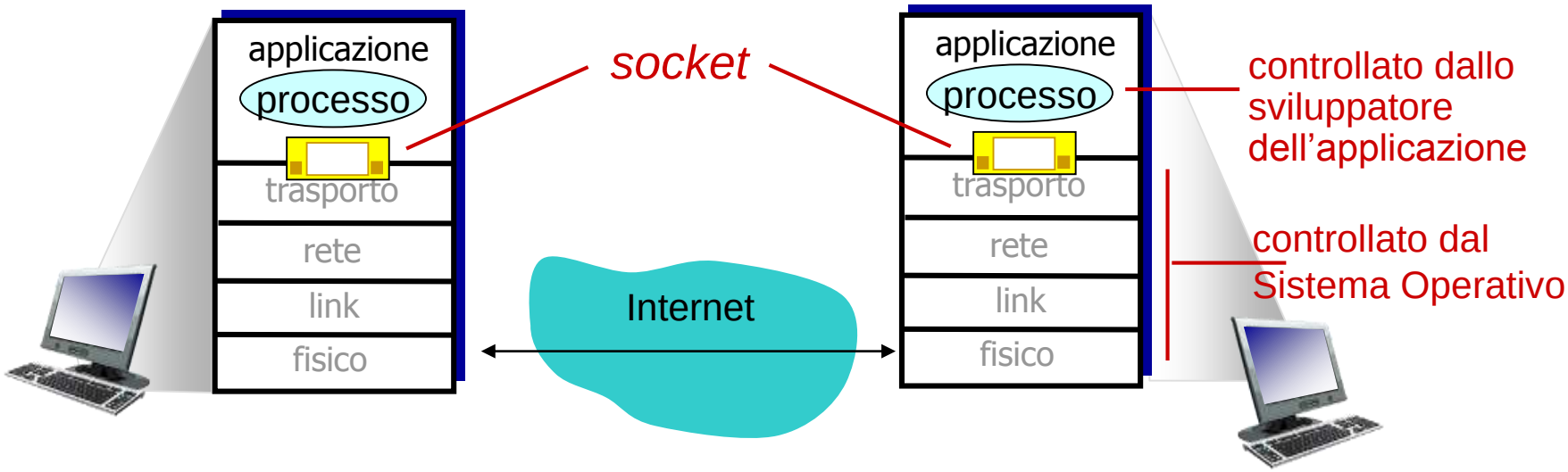
2.6 applicazioni P2P

2.7 programmazione delle socket con UDP e TCP

Programmazione delle socket

obiettivo: imparare a costruire applicazioni client/server che comunicano utilizzando le socket

socket: una porta tra il processo di un'applicazione e il protocollo di trasporto end to end



Programmazione delle socket

Due tipi di socket per due servizi di trasporto:

- *UDP:* trasferimento non affidabile di gruppi di byte ("datagrammi")
- *TCP:* trasferimento affidabile di un flusso di byte da un processo all'altro

Esempio di applicazione:

1. Il client legge una linea di caratteri (dati) dalla tastiera e la invia al server.
2. Il server riceve I dati e converte i caratteri in maiuscolo.
3. Il server invia i dati modificati al client.
4. Il client riceve i dati modificati e mostra la linea sullo schermo.

Programmazione delle socket *con UDP*

UDP: non c'è "connessione" tra client & server

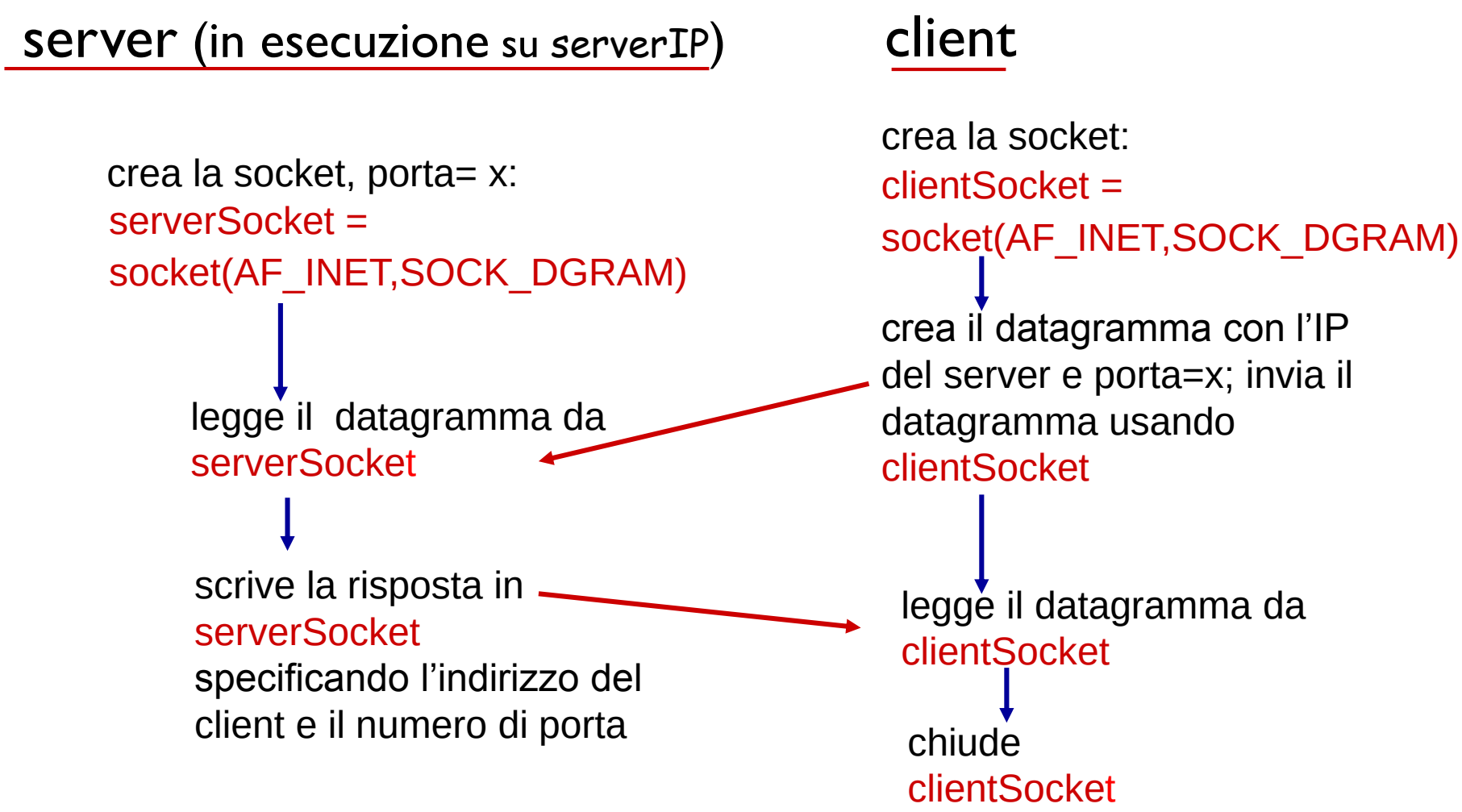
- non c'è handshaking prima dell'invio di dati
- il mittente indica esplicitamente in ogni pacchetto l'indirizzo IP e la porta di destinazione
- chi riceve estrae l'indirizzo IP e la porta del mittente dal pacchetto ricevuto

UDP: i dati trasmessi possono perdersi o arrivare a destinazione in un ordine diverso da quello d'invio

Punto di vista dell'applicazione:

- UDP fornisce un trasferimento *inaffidabile* di gruppi di byte ("datagrammi") tra client e server

Interazione delle socket client/server: UDP



Esempio: client UDP

UDPClient in Python

include le librerie socket Python →

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

crea una socket client UDP →

```
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
```

legge l'input da tastiera dell'utente →

```
message = raw_input('Input lowercase sentence:')
```

specifica nome e porta del server; invia nella socket →

```
clientSocket.sendto(message,(serverName, serverPort))
```

legge i caratteri nella risposta dalla socket e li mette in una stringa →

```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

visualizza la stringa ricevuta e chiude la socket →

```
print modifiedMessage
clientSocket.close()
```

Esempio: server UDP

UDPServer in Python

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

crea la socket UDP

associa la socket alla porta locale numero 12000

loop continuo

legge i caratteri dalla socket e li mette in una stringa, (ottenendo anche IP e porta del client)

invia la stringa upper case al client

Programmazione delle socket *con TCP*

il client deve contattare il server

- il processo server deve essere in esecuzione
- il server deve aver creato delle socket (porte) di benvenuto per il client

il client contatta il server:

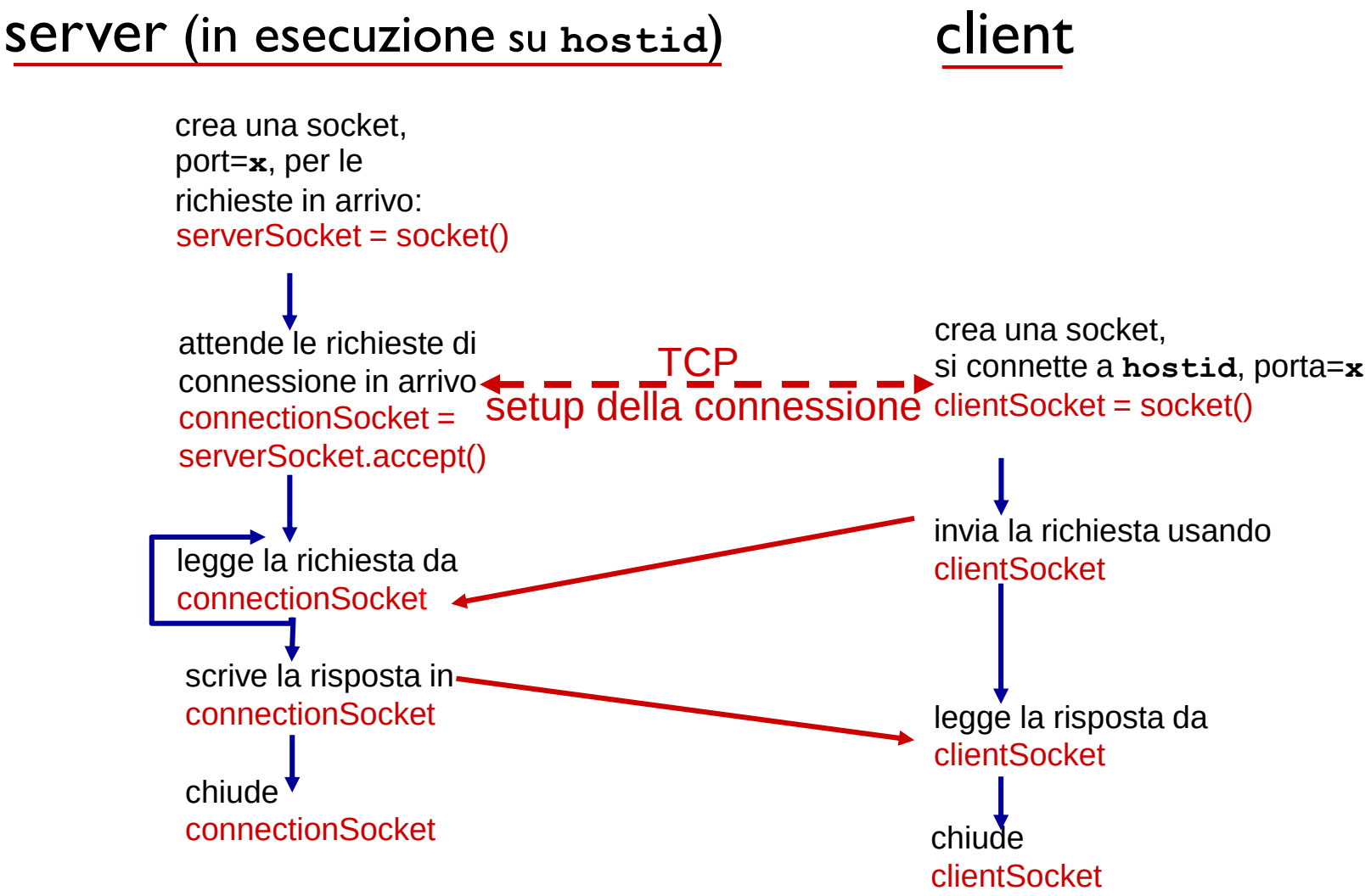
- creando una socket TCP, specificando indirizzo IP, numero di porta del processo server
- *quando il client crea la socket:* il client TCP stabilisce una connessione con il server TCP

- quando viene contattato dal client, *il server TCP crea una nuova socket* per far comunicare il processo server con quel particolare client
  - ciò consente al server di parlare con più client
  - il numero di porta sorgente viene usato per distinguere i client (Cap 3)

punto di vista dell'applicazione:

il TCP fornisce un trasferimento affidabile di un flusso ordinato di byte ("pipe") tra client e server

Interazione delle socket client/server : TCP



Esempio: client TCP

TCPClient in Python

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

crea una socket TCP, porta remota 12000

non serve specificare il nome e la porta server

Esempio: server TCP

TCPServer in Python

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

crea una socket TCP di benvenuto

il server comincia ad ascoltare richieste TCP in arrivo

loop continuo

il server usa la accept() per le richieste in arrivo, una nuova socket viene creata

legge i byte dalla socket (non l'indirizzo come in UDP)

chiude la connessione a questo client (ma *non* il socket di benvenuto)

Capitolo 2: riassunto

il nostro studio sulle applicazioni di rete è ora completo!

- architetture delle applicazioni
  - client-server
  - P2P
- requisiti dei servizi delle applicazioni:
  - affidabilità, bandwidth, ritardo
- Modelli di servizio di trasporto di Internet
  - connection-oriented, affidabile: TCP
  - non affidabile, datagrammi: UDP
- protocolli specifici:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- programmazione delle socket: socket TCP e UDP

Capitolo 2: riassunto

molto importante: abbiamo conosciuto i protocolli!

- tipico scambio di messaggi di richiesta/risposta:
  - il client richiede informazioni o servizi
  - il server risponde con dati e codici di stato
- formati dei messaggi:
  - intestazioni: campi che forniscono informazioni sui dati
  - dati: informazioni da comunicare

argomenti importanti :

- messaggi di dati e di controllo
  - in-band, out-of-band
- architetture centralizzate o decentralizzate
- protocolli stateless e stateful
- trasferimento di messaggi affidabile e non affidabile
- "complessità presenti nei confini della rete"