



*Corso di Laurea Triennale in Informatica
Università degli Studi della Basilicata*

Reti di Calcolatori

Docente: Ugo Erra

ugo.erra+reti@unibas.it

3° Lezione – Livello di applicazione - I° parte

Sommario



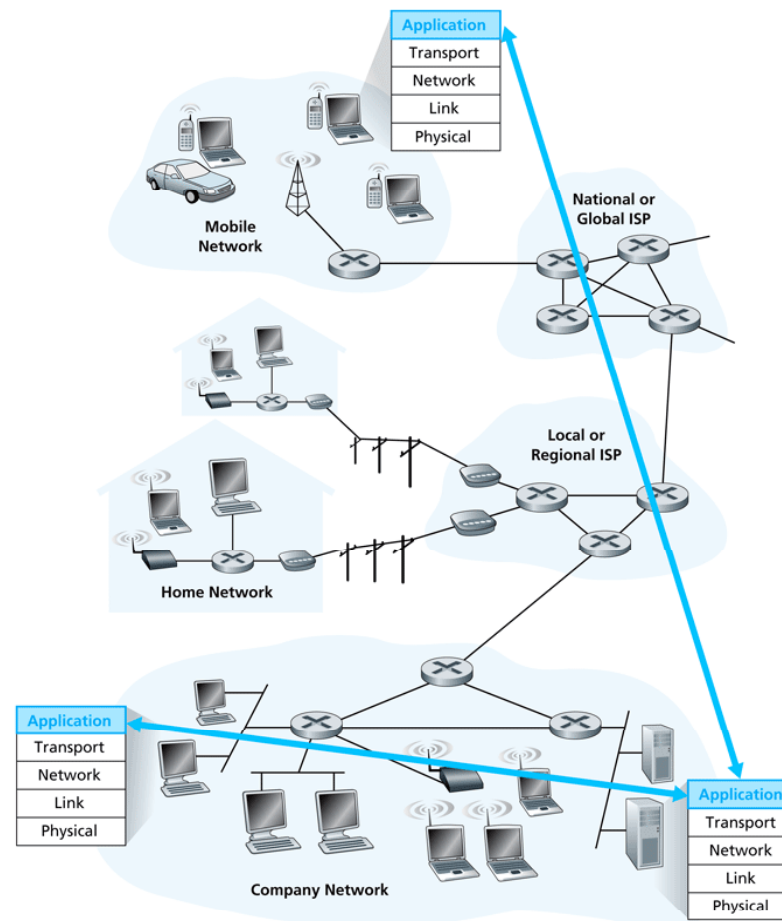
- **Principi delle applicazioni di rete**
 - Architetture delle applicazioni
 - Servizi richiesti dalle applicazioni
- **Web e HTTP**
 - Formato dei messaggi
 - Connessione persistente e non
 - Cookie e web cache

Alcune applicazioni di rete

- Posta elettronica
- Web
- Messaggistica istantanea
- Telefonia via Internet
- Videoconferenza in tempo reale
- Condivisione di file P2P
- Giochi multiutente via rete
- Streaming di video-clip memorizzati
- Autenticazione in un calcolatore remoto (Telnet e SSH)

Applicazioni di rete

- Scrivere una applicazione di rete significa predisporre un processo software che:
 - ▣ Gira su sistemi terminali diversi
 - ▣ Comunicano attraverso la rete
- Non occorre predisporre programmi per i dispositivi del nucleo della rete, quali router o commutatori Ethernet



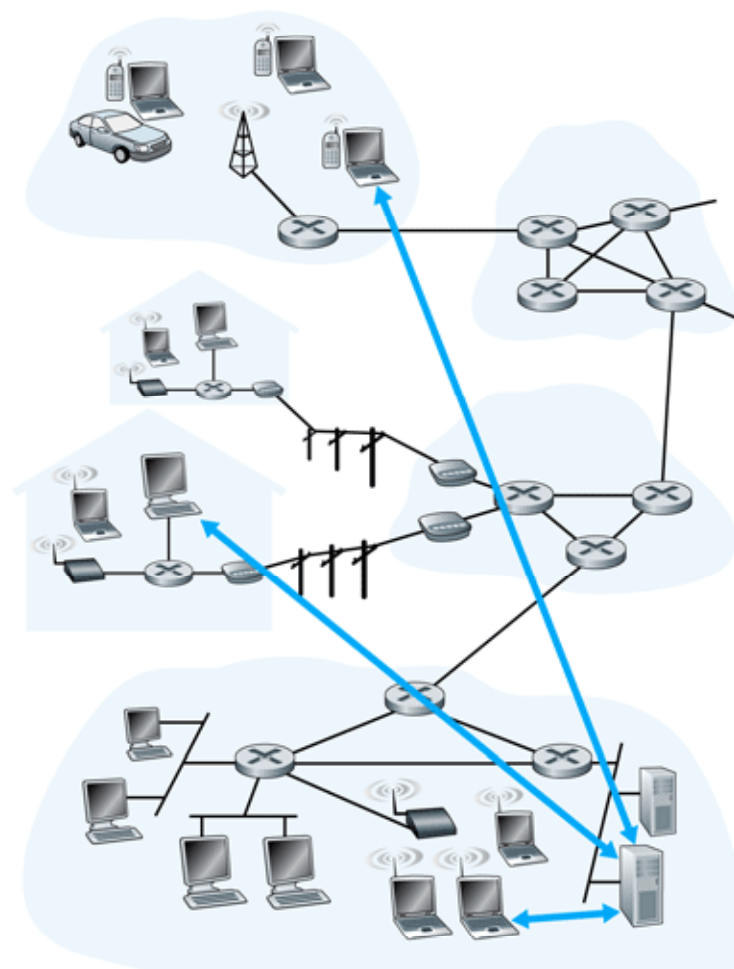
Architetture dell'applicazione



- Sviluppare un'applicazione di rete significa predisporre l'organizzazione del flusso dei dati
- Principali architetture attualmente utilizzate:
 - ▣ Client-server
 - ▣ Peer-to-peer (P2P)
 - ▣ Architetture ibride (client-server e P2P)

Architettura Client-server

- Nell'**architettura client-server** abbiamo un processo *server* ed un processo *client*
- **Server**
 - ▣ Host sempre attivo
 - ▣ Indirizzo IP fisso
- **Client**
 - ▣ Comunica con il server
 - ▣ Può contattare il server in qualunque momento
 - ▣ Può avere indirizzi IP dinamici
 - ▣ Non comunica direttamente con gli altri client



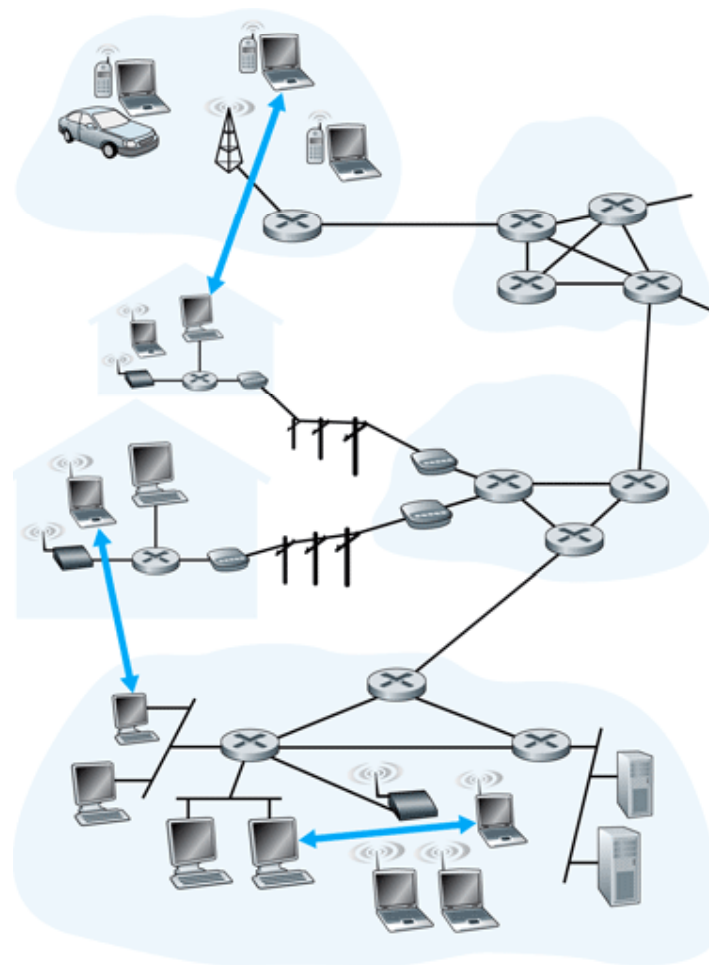
Server farm



- Un singolo server potrebbe non essere in grado di rispondere a tutte le richieste
- Una **server farm** è un cluster di host utilizzati per creare un potente server virtuale
 - ▣ Ad esempio Google, Amazon, etc...

Architettura Peer-to-peer (P2P)

- Nelle **architetture P2P** abbiamo coppie di host chiamati *peer*(pari)
 - ▣ Non c'è un server sempre attivo
 - ▣ I peer comunicano direttamente tra loro
 - ▣ I peer non devono necessariamente essere sempre attivi, e cambiano indirizzo IP
- Il vantaggio principale delle architetture P2P è la **scalabilità**
 - ▣ Un nuovo utente genera carico di lavoro ma offre anche nuove capacità
 - ▣ Difficile da gestire
- Un esempio: Gnutella



Architetture ibride (client-server e P2P)

- Napster
 - ▣ Scambio di file secondo la logica P2P
 - ▣ Ricerca di file centralizzata:
 - I peer registrano il loro contenuto presso un server centrale
 - I peer chiedono allo stesso server centrale di localizzare il contenuto
- Messaggistica istantanea
 - ▣ La chat tra due utenti è del tipo P2P
 - ▣ Individuazione della presenza/location centralizzata:
 - L'utente registra il suo indirizzo IP sul server centrale quando è disponibile online
 - L'utente contatta il server

Processi comunicanti



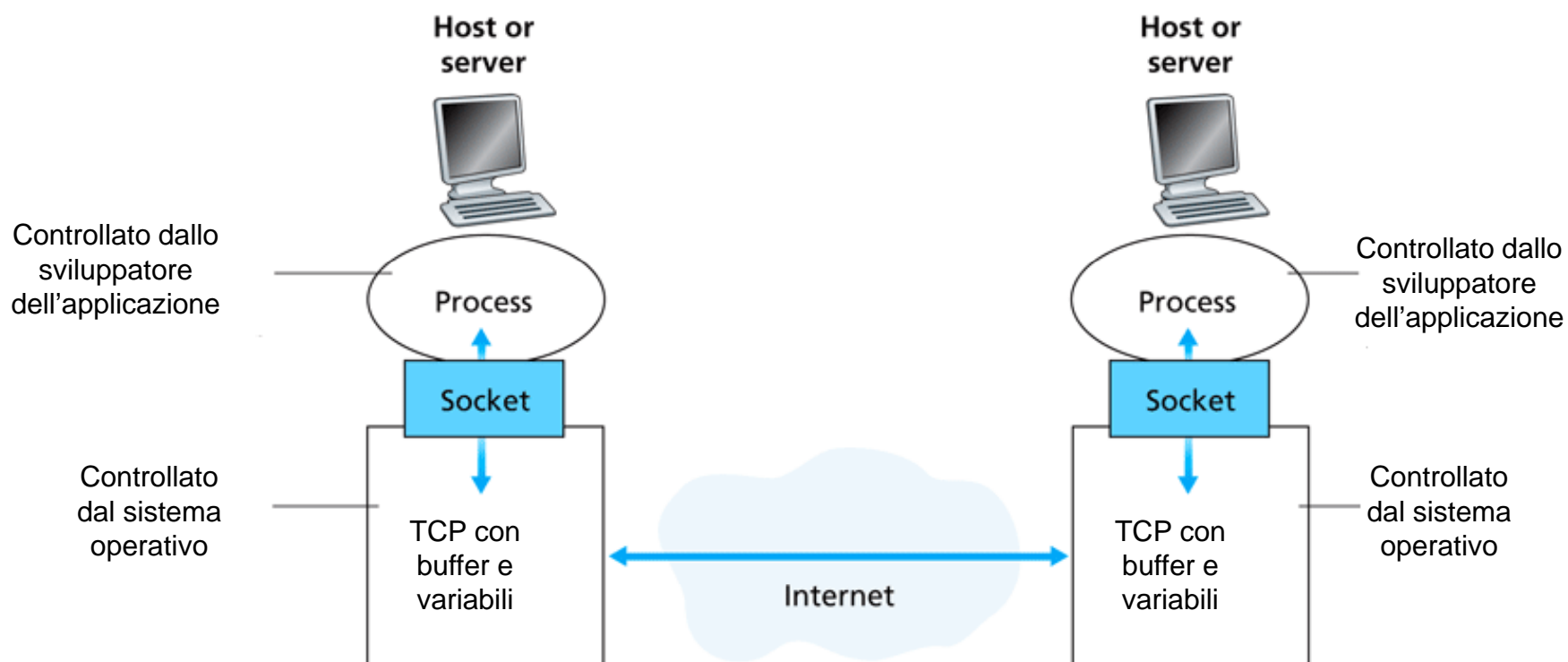
- Un processo è un programma in esecuzione su di un host
 - ▣ All'interno di un host, due processi comunicano utilizzando schemi interprocesso (definiti dal S.O.)
 - ▣ Processi su host differenti comunicano attraverso lo scambio di messaggi
- Un processo client dà inizio alla comunicazione (client side)
- Un processo server attende di essere contattato (server side)

Socket



- Un processo invia/riceve messaggi a/da la sua **socket**
- Una socket è analoga ad una porta:
 - ▣ Un processo client (o server) che vuole inviare (o ricevere) un messaggio, lo fa uscire dalla propria “porta” (socket)
- Lo sviluppatore presuppone l’esistenza di un’infrastruttura esterna che trasporterà il messaggio da una socket all’altra attraverso la rete

Processi e Socket



Application Program Interface



- La socket rappresenta l'interfaccia di programmazione con le applicazioni
- Lo sviluppatore ha il controllo totale sul livello applicativo
- ...ma ha poco controllo sul livello trasporto
 - ▣ Sceglie il protocollo di trasporto (TCP, UDP)
 - ▣ Può determinare la dimensione massima del buffer e del segmento

Servizi di trasporto disponibili



- Il protocollo a livello trasporto ha il compito di consegnare i messaggi alla porta del processo ricevente
- I servizi che possono essere necessari ad un'applicazione sono:
 - ▣ Trasferimento dati affidabile
 - ▣ Ampiezza di banda
 - ▣ Temporizzazione
 - ▣ Sicurezza

Trasferimento dati affidabile



- ❑ Il **trasferimento di dati affidabile** è un servizio di garanzia nella consegna dei dati
- ❑ I dati sono consegnati correttamente e completamente
- ❑ Alcune applicazioni (ad esempio, audio) possono tollerare qualche perdita
- ❑ Altre applicazioni (ad esempio, trasferimento di file, telnet) richiedono un trasferimento dati affidabile al 100%

Ampiezza di banda

- Le *applicazioni sensibili alla banda* possono richiedere un throughput garantito di r bsp
- Il protocollo di trasporto garantisce almeno r bps
- Alcune applicazioni (ad esempio, quelle multimediali) per essere “efficaci” richiedono un’ampiezza di banda minima
- **Le applicazioni elastiche** utilizzano l’ampiezza di banda che si rende disponibile
 - Ad esempio il Web e la posta elettronica

Temporizzazione



- Alcune applicazioni per essere “realistiche” richiedono piccoli ritardi
- La temporizzazione garantisce un certo ritardo end-to-end tra le applicazioni
 - ▣ Telefonia, ambienti virtuali, giochi multimediali

Sicurezza



- Un protocollo di trasporto può richiedere anche la cifratura di tutti i dati trasmessi
- Anche se i dati vengono intercettati non c'è perdita di riservatezza
- I servizi di sicurezza possono anche essere applicati per garantire l'integrità dei dati e l'autenticazione end-to-end

Requisiti di alcune applicazioni

Applicazioni	Tolleranza alla perdita dei dati	Ampiezza di banda	Sensibilità dal tempo
Trasferimento file	No	Variabile	No
Posta elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/Video in tempo reale	Si	Audio: da 5Kbps a 1Mbps Video: da 10Kbps a 5 Mbps	Si, centinaia di ms
Audio/Video memorizzati	Si	Come sopra	Si, pochi secondi
Giochi interattivi	Si	Fino a pochi K	Si, centinaia di ms
Messaggistica istantanea	No	Variabile	Si e No

Servizi di trasporto di Internet



- Internet fornisce due protocolli di trasporto:
 - ▣ Transmission Control Protocol (TCP)
 - ▣ User Datagram Protocol (UDP)

Servizi di TCP

- *Orientato alla connessione*
 - Prima che i processi si scambino informazioni avviene un setup fra i processi client e server
- *Trasporto affidabile*
 - I dati sono trasportati senza errori e nell'ordine corretto
- *Controllo di flusso*
 - Il mittente non può sovraccaricare il destinatario
- *Controllo della congestione*
 - “Strozza” il processo d'invio quando la rete è sovraccaricata
- Non offre: temporizzazione, ampiezza di banda minima

Servizi di UDP



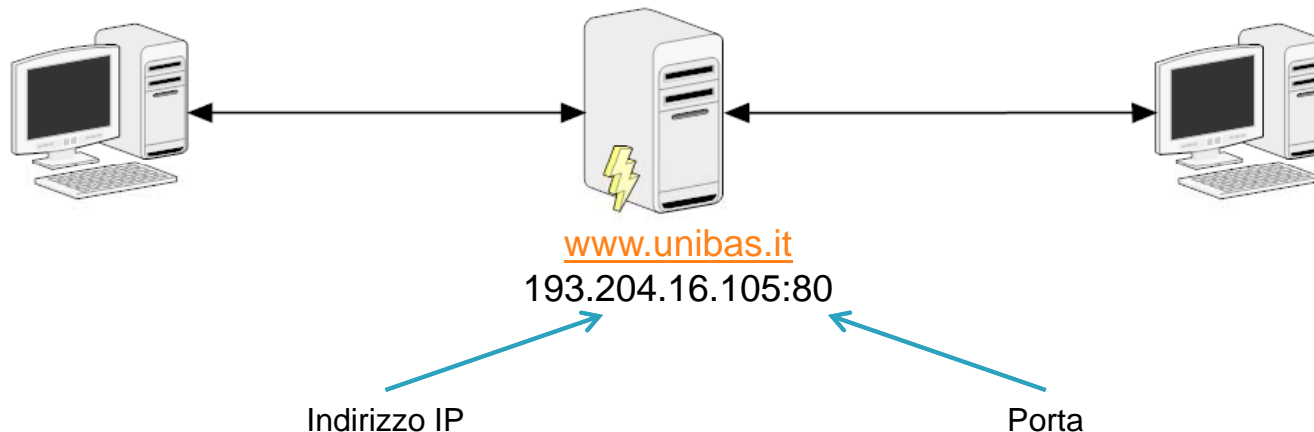
- Trasferimento dati inaffidabile fra i processi d'invio e di ricezione
- Non offre:
 - Setup della connessione affidabilità
 - Controllo di flusso
 - Controllo della congestione
 - Temporizzazione
 - Ampiezza di banda minima

Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto

Applicazione	Protocollo a livello applicazione	Protocollo di trasporto sottostante
Posta elettronica	SMTP [RFC 2821]	TCP
Accesso a terminali remoti	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia in streaming	Proprietario (RealNetworks)	TCP o UDP
Telefonia internet	Proprietario (Vonage, Dialpad)	Tipicamente UDP

Processi di indirizzamento

- Un processo su di un host per poter inviare un messaggio ad un processo su un altro host deve identificare il processo destinatario
- Ogni host possiede un **indirizzo IP** univoco a 32 bit
- Poiché più processi possono girare contemporaneamente ad ogni processo è associata anche una **porta**
- L'*identificatore* comprende sia l'indirizzo IP che i numeri di porta associati al processo in esecuzione su un host



Protocolli a livello applicazione

- Un protocollo a livello applicazione definisce:
 - I tipi di messaggi scambiati
 - Ad esempio messaggi di richiesta e di risposta
 - La sintassi dei messaggi
 - Quali sono i campi nel messaggio e come sono descritti
 - La semantica dei campi
 - Significato delle informazioni nei campi
 - Regole per determinare quando e come un processo invia e risponde ai messaggi
- Esistono fondamentalmente due tipi di protocollo
 - Protocolli di pubblico dominio
 - Definiti nelle RFC
 - Consente l'interoperabilità (HTTP, SMTP, etc...)
 - Protocolli proprietari
 - KaZaA

Sommario



- Principi delle applicazioni di rete
 - ▣ Architetture delle applicazioni
 - ▣ Servizi richiesti dalle applicazioni
- **Web e HTTP**
 - ▣ Formato dei messaggi
 - ▣ Connessione persistente e non
 - ▣ Cookie e web cache

Web e HTTP

- Il World Wide Web è un'applicazione nata agli inizi degli anni '90 da parte di Tim Berners Lee
 - ▣ Ha coniato il nome di World Wide Web
 - ▣ Ha scritto il primo server per il Web, ed il primo programma client
 - ▣ Ha scritto la prima versione del linguaggio di formattazione HTTP
- Il Web opera su richiesta (on demand)



Il computer NeXT Cube utilizzato da Tim Berners-Lee come primo Server Web (*wikipedia*)

L'idea

- 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 un **file base HTML** che include diversi oggetti referenziati
- Ogni oggetto è referenziato da un **Uniform Resource Locator** o **URL**
- Esempio di URL:

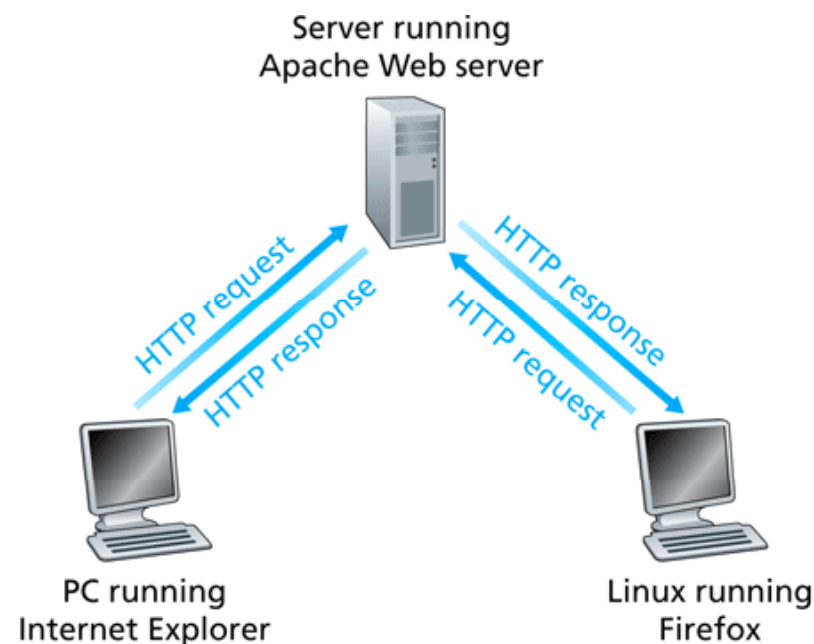
www.someschool.edu/someDept/pic.gif

Nome dell'host

Nome del percorso

Il protocollo HTTP - 1

- L' **hypertext transfer protocol** (HTTP) è il protocollo a livello di applicazione del Web
- Utilizza un modello client/server
 - ▣ *client*: un browser che richiede, riceve e “visualizza” gli oggetti del Web
 - ▣ *server*: il server web invia oggetti in risposta a una richiesta
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



Il protocollo HTTP - 2

- Il protocollo HTTP usa TCP
 - ▣ Il client inizializza la connessione TCP (crea un socket) con il server usando la porta 80
 - ▣ Il server accetta la connessione TCP dal client
 - ▣ Il client HTTP (browser) ed il server HTTP (server web) si scambiano messaggi HTTP
 - ▣ Al termine la connessione TCP è chiusa
- HTTP è un **protocollo senza stato** (*stateless protocol*)
 - ▣ Il server non mantiene informazioni sulle richieste fatte dal client

Tipi di connessioni HTTP

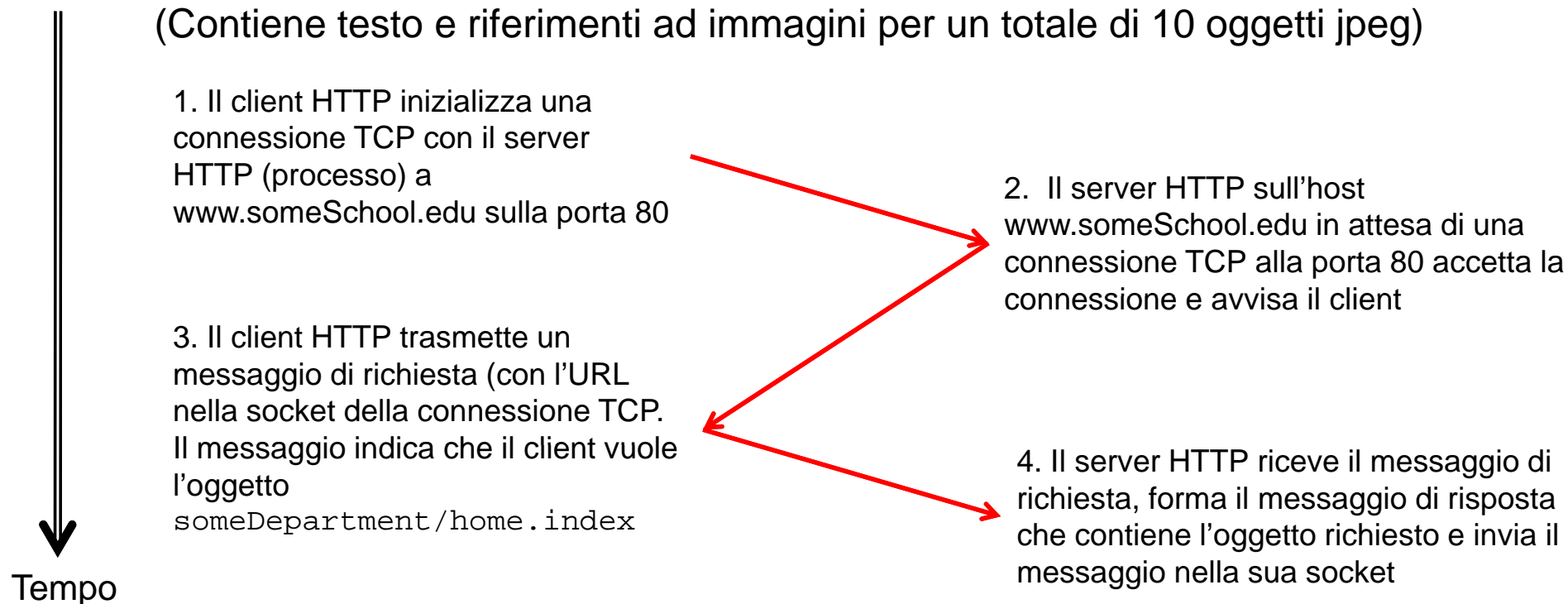
- ❑ **Connessioni non persistenti**
 - ▣ Almeno un oggetto viene trasmesso su una stessa connessione TCP
 - ▣ HTTP/1.0 usa connessioni non persistenti
- ❑ **Connessioni persistenti**
 - ▣ Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server
 - ▣ HTTP/1.1 usa connessioni persistenti nella modalità di default

Connessioni non persistenti - 1

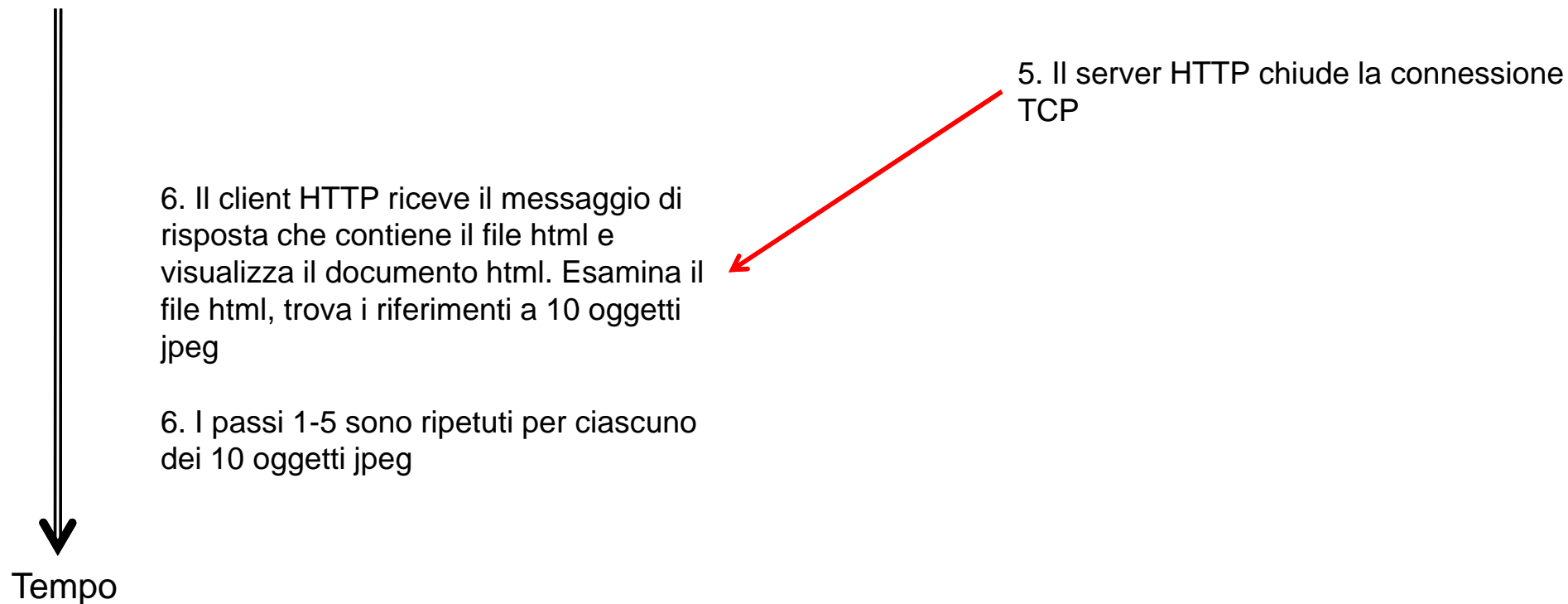
□ Supponiamo che l'utente immetta l'URL

`www.someSchool.edu/someDepartment/home.index`

(Contiene testo e riferimenti ad immagini per un totale di 10 oggetti jpeg)

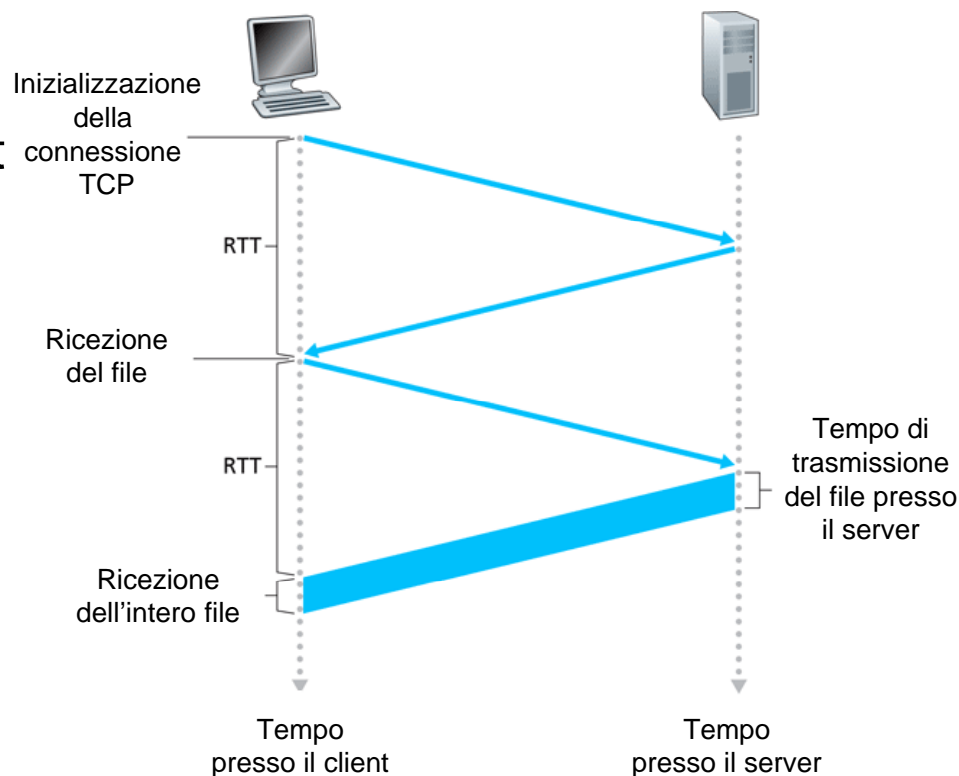


Connessioni non persistenti - 2



Schema del tempo di risposta

- Il **round-trip time (RTT)** è il tempo impiegato da un piccolo pacchetto per andare dal client al server e ritornare al client
- Il tempo di risposta totale comprende:
 - ▣ Un RTT per inizializzare la connessione TCP
 - ▣ Un RTT perché ritornino la richiesta HTTP di connessione e i primi byte della risposta HTTP
 - ▣ Tempo di trasmissione del file
- Totale = $2RTT +$ tempo di trasmissione



Svantaggi connessioni non persistenti



- Richiede 2 RTT per oggetto
- Overhead del sistema operativo per ogni connessione TCP
 - ▣ Inizializzazione strutture dati, allocazione buffer, etc...
- I browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati

Connessioni persistenti

- Nelle connessioni persistenti il server lascia aperta la connessione TCP dopo l'invio di una risposta
 - ▣ I successivi messaggi vengono spediti sulla stessa connessione
- Le connessioni persistenti possono essere di due tipi:
 - ▣ Una di seguito all'altra (*back-to-back*)
 - ▣ Senza aspettare la risposte pendenti (*pipelining*)

Pipelining



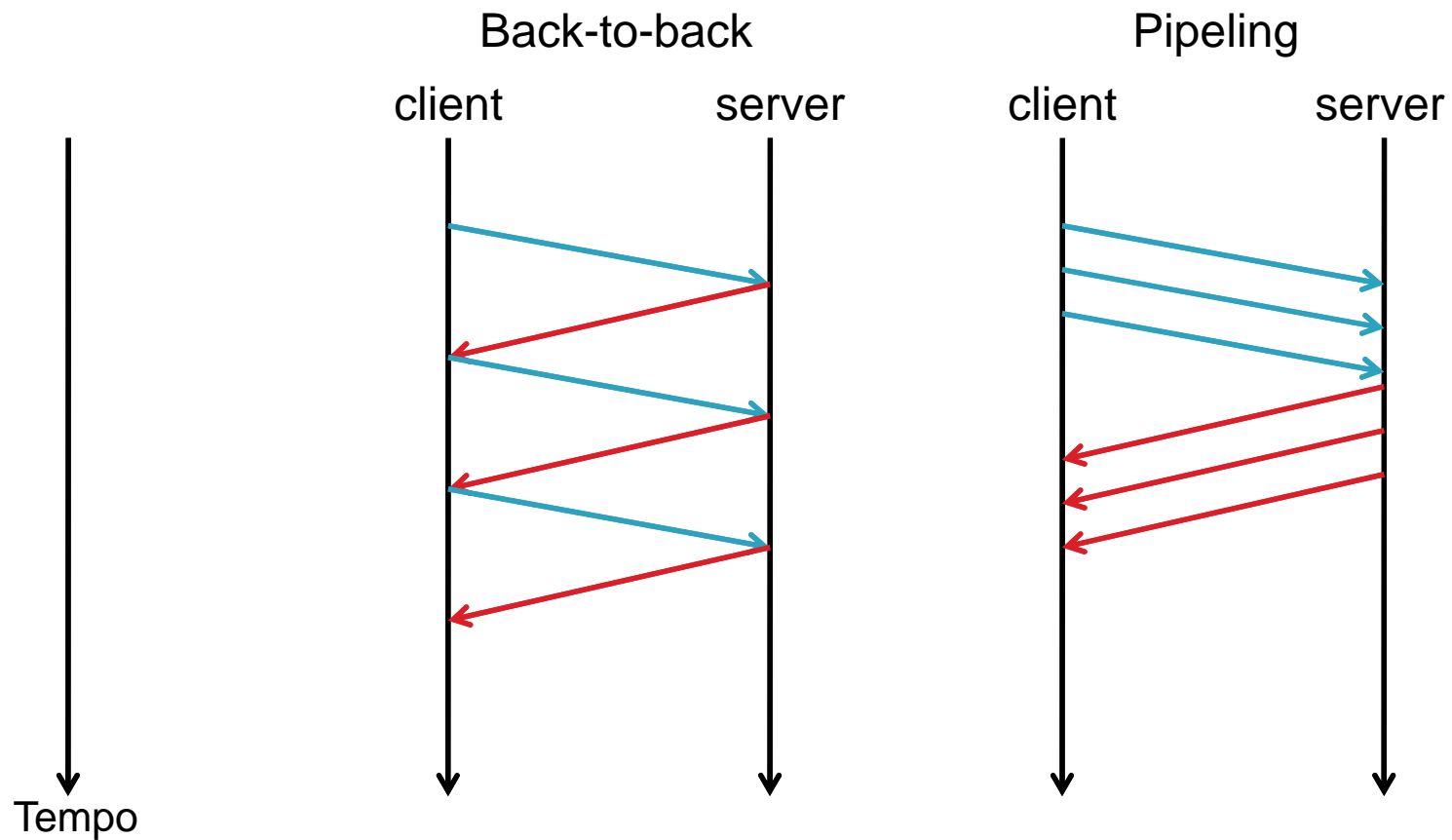
- Il client invia una nuova richiesta solo quando ha ricevuto la risposta precedente
- E' necessario un RTT per ogni oggetto referenziato

Back-to-back



- Il client invia una richiesta non appena incontra un oggetto referenziato
- Un solo RTT per ogni oggetto referenziato
- Modalità di default in HTTP/1.1

Pipelining/Back-To-Back



Formato messaggi HTTP

- I tipi di messaggi HTTP possono essere suddivisi in:
 - ▣ Messaggi di richiesta
 - ▣ Messaggi di risposta
- Un messaggio di richiesta è scritto in testo ASCII

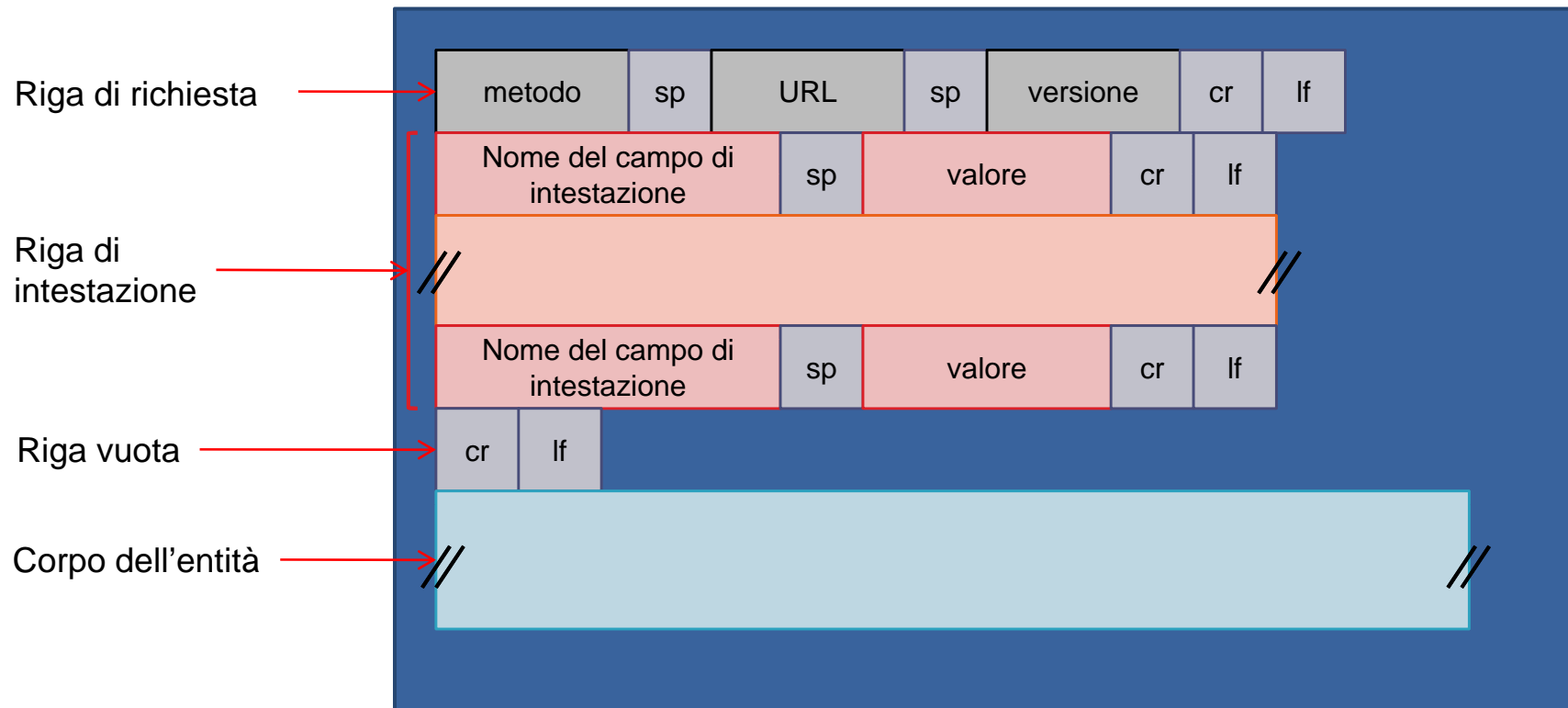
Riga di richiesta
(comandi GET,
POST, HEAD)

Righe di intestazione

Un carriage return e
un line feed indicano la
fine del messaggio

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

Formato generale dei messaggi di richiesta



Upload dell'input di un form

□ Metodo URL

- ▣ Usa il metodo GET
- ▣ L'input arriva al server nel campo URL della riga di richiesta
- ▣ Ad esempio:

`www.somesite.com/animalsearch?scimmie&banane`

□ Metodo POST

- ▣ Una pagina web può permettere ad un utente di inserire delle informazioni attraverso una form
- ▣ L'input al server nel corpo dell'entità

Tipi di metodi

- HTTP/1.0
 - ▣ GET
 - ▣ POST
 - ▣ HEAD
 - L'oggetto non è incluso nella risposta del server
- HTTP/1.1
 - ▣ GET, POST, HEAD
 - ▣ PUT
 - Include un 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
espressione di stato)

Righe di intestazione

dati, ad
esempio il file
HTML richiesto

```
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 Righe di, 22 Jun
1998
Content-Length: 6821
Content-Type: text/html
```

```
dati dati dati dati dati ...
```

Codici di stato nella risposta HTTP

- **200 OK**
 - ▣ La richiesta ha avuto successo; l'oggetto richiesto viene inviato nella risposta
- **301 Moved Permanently**
 - ▣ L'oggetto richiesto è stato trasferito; 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**
 - ▣ Il server non ha la versione di protocollo HTTP

Interazione utente-server: i cookie

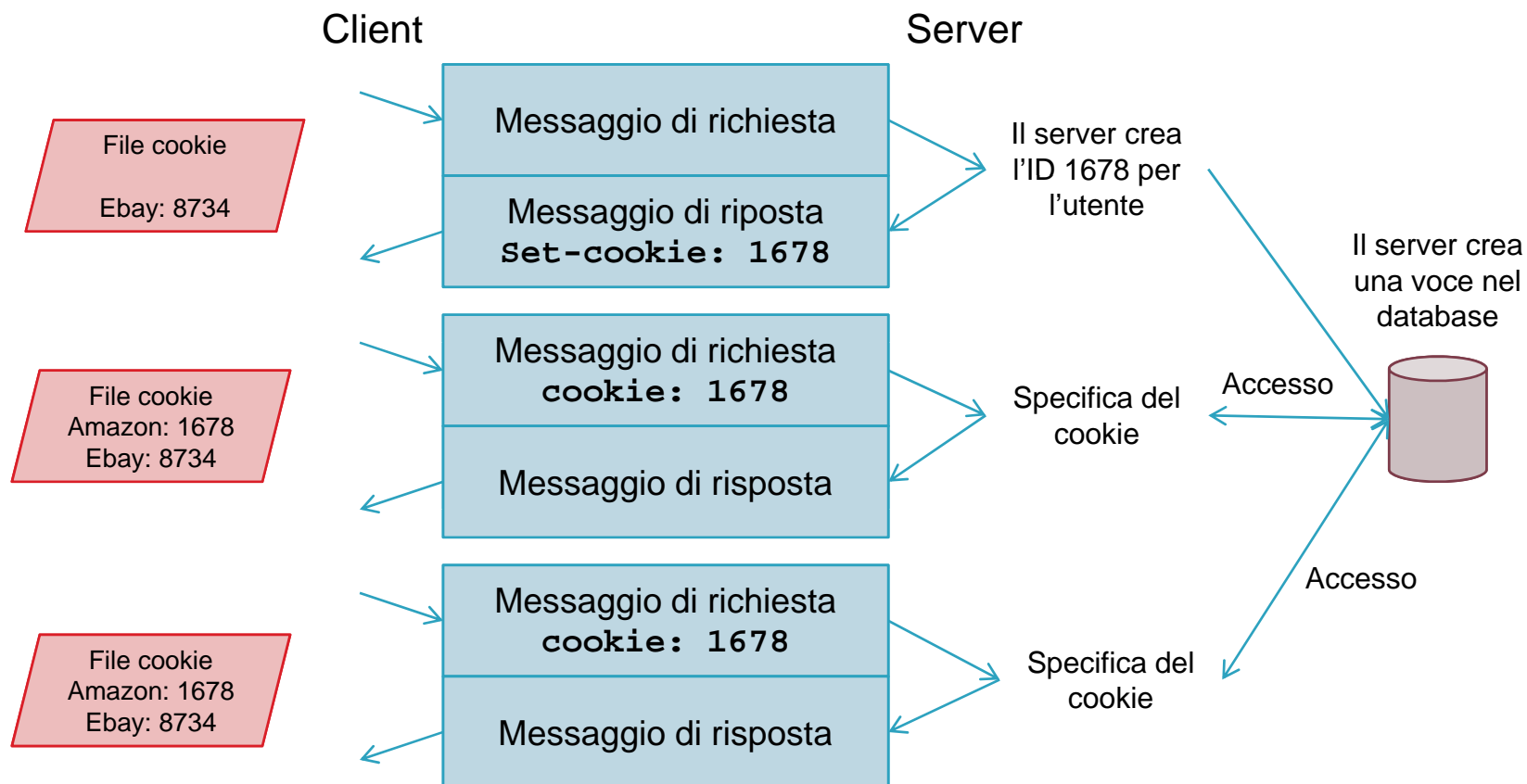
- I cookie permettono di rimediare parzialmente alla mancanza di stato di HTTP
- Per poter usare i cookie abbiamo bisogno di:
 - ▣ Una riga di intestazione nel messaggio di risposta HTTP
 - ▣ Una riga di intestazione nel messaggio di richiesta HTTP
 - ▣ Un file cookie mantenuto sul sistema terminale dell'utente e gestito dal browser dell'utente
 - ▣ Un database sul sito

Esempio di cookie



- Susan accede sempre a Internet 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 identificativo unico (ID) e una entry nel database ID

Esempio di cookie

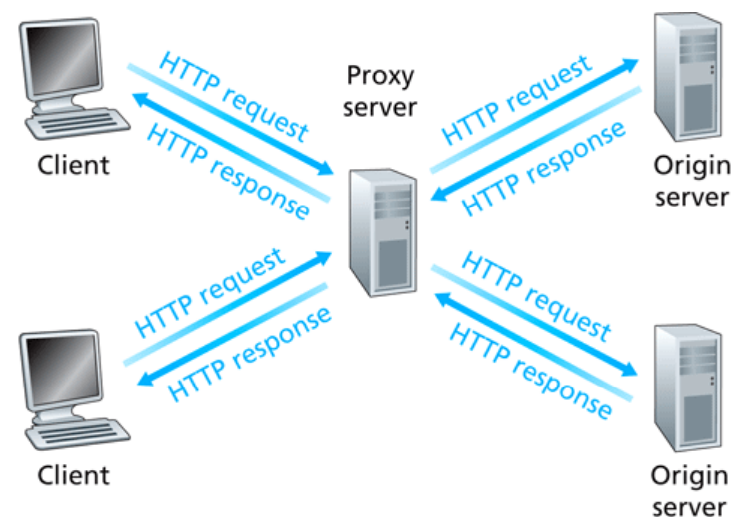


Cookie e privacy

- Cosa possono contenere i cookie
 - ▣ Autorizzazione
 - ▣ Numero di carta di credito
 - ▣ Raccomandazioni
 - ▣ Stato della sessione dell'utente (email)
- I cookie sono spesso criticati per via della privacy
 - ▣ Permettono ai siti di tracciare le abitudini degli utenti
 - L'utente può fornire al sito il nome e l'indirizzo e-mail
 - ▣ I motori di ricerca usano il reindirizzamento e i cookie per saperne ancora di più
 - ▣ Le agenzie pubblicitarie ottengono informazioni dai siti

Cache web - 1

- Una **cache web** o **server proxy** permette di soddisfare le richieste del client senza coinvolgere il server originale
- L'utente configura il browser impostando un accesso al Web d'origine tramite la cache
- Il browser trasmette tutte le richieste HTTP alla cache
 - ▣ Se l'oggetto è presente 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 - 2

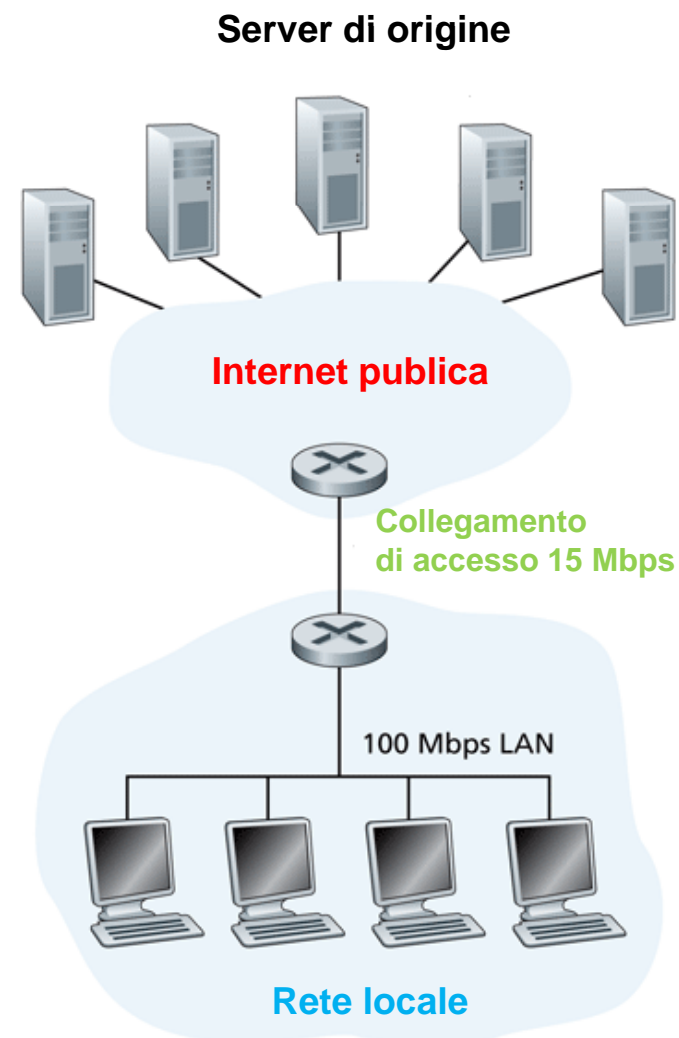


- La cache opera come un client ed un server
- Tipicamente la cache è installata da un ISP (università, aziende o ISP residenziali)
- Vantaggi del caching
 - ▣ Riduce i tempi di risposta alle richieste dei client
 - ▣ Riduce il traffico sul collegamento di accesso a Internet
 - ▣ Internet arricchita di cache consente ai provider “scadenti” di fornire dati con efficacia

Esempio di caching - 1

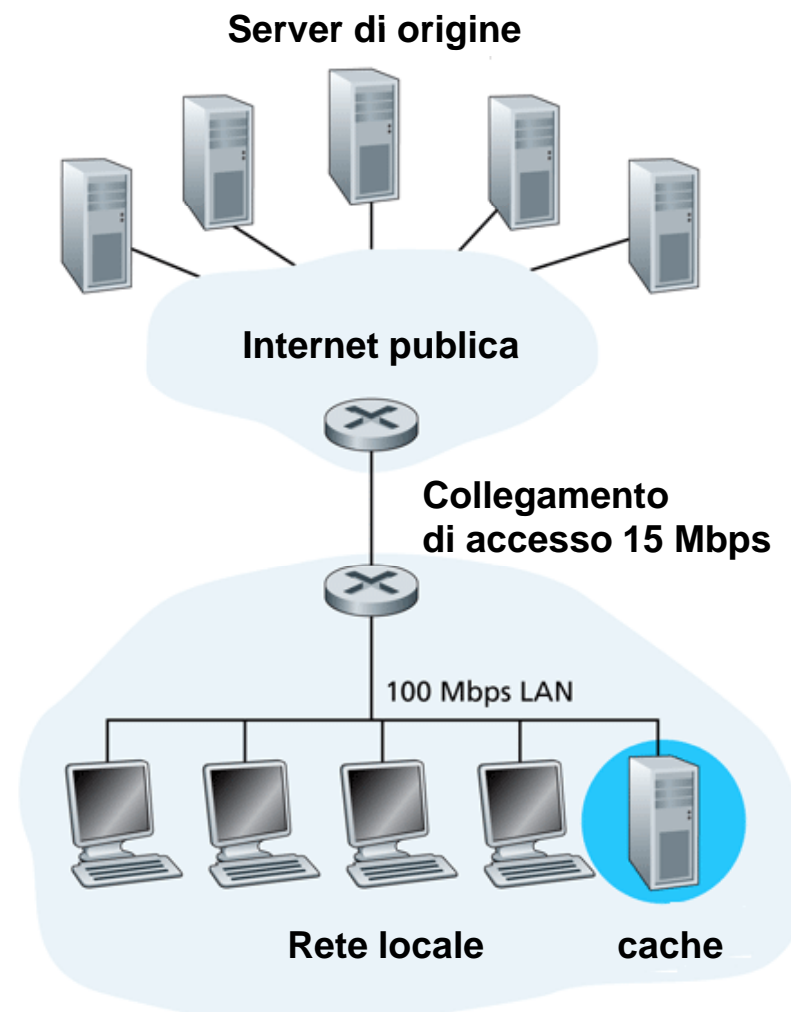
Intensità di traffico: L_a/R

- Assumiamo che:
 - ▣ La dimensione media degli oggetti è di 1 Mbps
 - ▣ La frequenza media delle richieste al server è di 15/sec
- Il ritardo medio del router locale verso il router lato Internet è di 2 sec
- Intensità di traffico rete locale
 - ▣ $(1 \text{ Mbit/richiesta}) \times (15 \text{ richieste/sec}) / (100 \text{ Mbps}) = 0.15$
 - ▣ Utilizziamo solo il 15% della LAN
 - ▣ In millisecondi quindi trascurabile
- Intensità di traffico sul collegamento d'accesso
 - ▣ $(1 \text{ Mbit/richiesta}) \times (15 \text{ richieste/sec}) / (15 \text{ Mbps}) = 1$
 - ▣ Utilizziamo il 100% del link di accesso
 - ▣ Il ritardo tende a 1 quindi cresce senza limiti
- Ritardo totale =
 - ▣ **Internet pubblica** + **accesso** + **LAN**
 - ▣ 2 sec + minutes + milliseconds



Esempio di caching - 1

- Adoperiamo una web cache con un hit rate 0.4
 - ▣ Il 40% delle richieste viene soddisfatto immediatamente
 - ▣ Il 60% devono arrivare al server di origine
 - L'intensità di traffico sul collegamento di accesso passa da 1 a 0.6
- Ritardo totale medio
 - = Internet + collegamento + LAN
 - = $0.6 \cdot (2.01) \text{ secs} + 0.4 \cdot \text{milliseconds} < 1.4 \text{ secs}$



GET condizionale

- La GET condizionale permette di inviare un oggetto se la cache ha una copia aggiornata dell'oggetto

- Cache: specifica la data della copia dell'oggetto nella richiesta HTTP

`If-modified-since: <data>`

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

`HTTP/1.0 304 Not Modified`

