Livello di applicazione: Web e HTTP

World Wide Web (WWW): applicazione Internet nata dalla necessità di scambio e condivisione di informazioni tra ricercatori universitari di varie nazioni

Storia

- Inizialmente una rete di documenti collegati
- Prototipo basato su testo (collegamenti ipertestuali)
- 1993: Primo browser grafico (Mosaic - Netscape)
- 1994: World Wide Web Consortium (W3C), organizzazione per sviluppare ulteriormente il Web, standardizzare protocolli, etc.

Caratteristiche

- Opera su richiesta (on demand)
- Facile rendere informazioni disponibili
- Facile da utilizzare

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 un file base HTML (HyperText Markup Language) che include diversi oggetti referenziati
- Ogni oggetto è referenziato da un URL (Uniform Resource Locator)
- Esempio di URL:

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

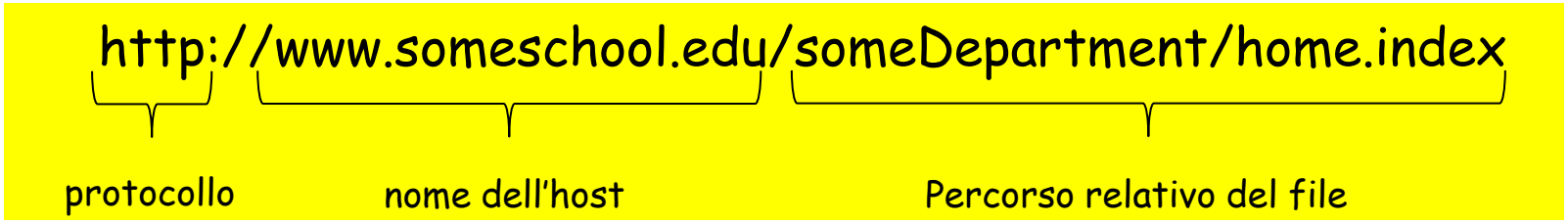
URL

Problema

- Come si chiama la pagina che si vuole visualizzare? Dove si trova? Come ci si può accedere?
- Se ogni pagina avesse un nome univoco, sarebbe possibile trovarla? Dove? (es. codice fiscale)

Soluzione

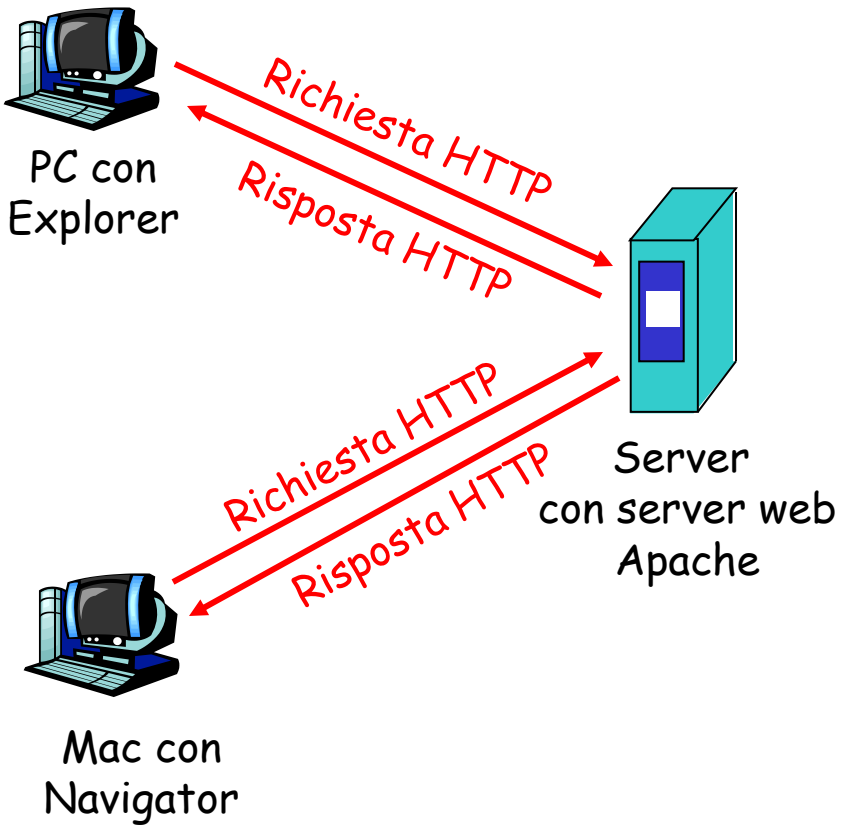
- ***Uniform Resource Locator (URL)**, composto di 3 parti:*
  1. *Il **protocollo***
  2. *Il **nome della macchina** in cui è situata la pagina*
  3. *Il **nome del file** (localmente alla macchina) che indica la pagina specifica*



Panoramica su HTTP (RFC 2616)

HTTP: hypertext transfer protocol

- Protocollo a livello di applicazione del Web
- Modello client/server
  - *client:* il browser che richiede, riceve, "visualizza" gli oggetti del Web
  - *server:* il server web invia oggetti in risposta a una richiesta
- HTTP definisce in che modo i client web richiedono pagine ai server web e come questi le trasferiscono ai client



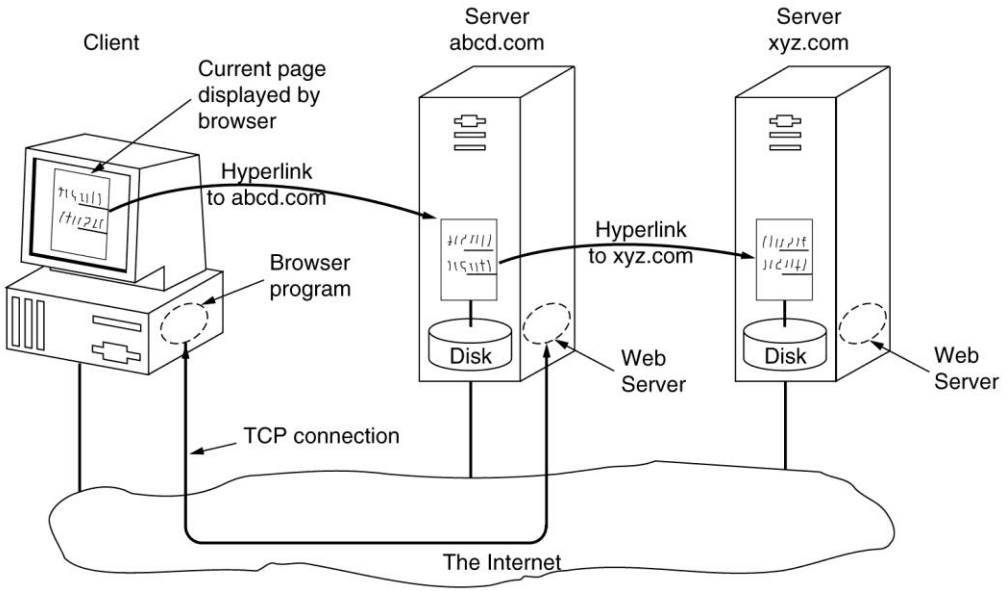
Panoramica su HTTP (continua)

Lato **Client** (Il *browser web* implementa l'*interfaccia* verso l'utente e il protocollo applicativo)

1. Il browser determina l'URL ed estrae host e filename
2. Esegue connessione TCP alla porta 80 dell'host indicato nella URL
3. Invia richiesta per il file
4. Riceve il file dal server
5. Chiude la connessione
6. Visualizza il file

Lato **Server**

1. Accetta una connessione TCP da un client
2. Ottiene il nome del file richiesto
3. Ottiene il file dal disco
4. Invia il file al client
5. Rilascia la connessione



Connessioni HTTP

Connessioni non persistenti

- Almeno un oggetto viene trasmesso su una connessione TCP
- Ciascuna coppia richiesta/risposta viene inviata su una connessione TCP separata

Connessioni persistenti

- Modalità di default
- Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server
- La connessione viene chiusa quando rimane inattiva per un lasso di tempo (timeout) configurabile

Connessioni non persistenti

Supponiamo che l'utente immetta l'URL

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

che contiene testo e riferimenti a 10 immagini jpeg

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

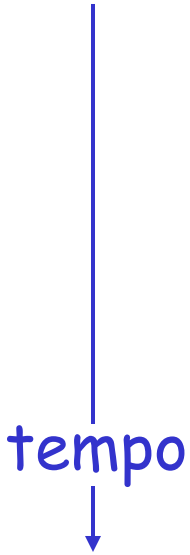
1b. Il server HTTP all'host www.someSchool.edu in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client

2. Il client HTTP trasmette un *messaggio di richiesta* (con 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, forma il *messaggio di risposta* che contiene l'oggetto richiesto e invia il messaggio nella sua socket

tempo

Connessioni non persistenti (continua)



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. Esamina il file html, trova i riferimenti a 10 oggetti jpeg

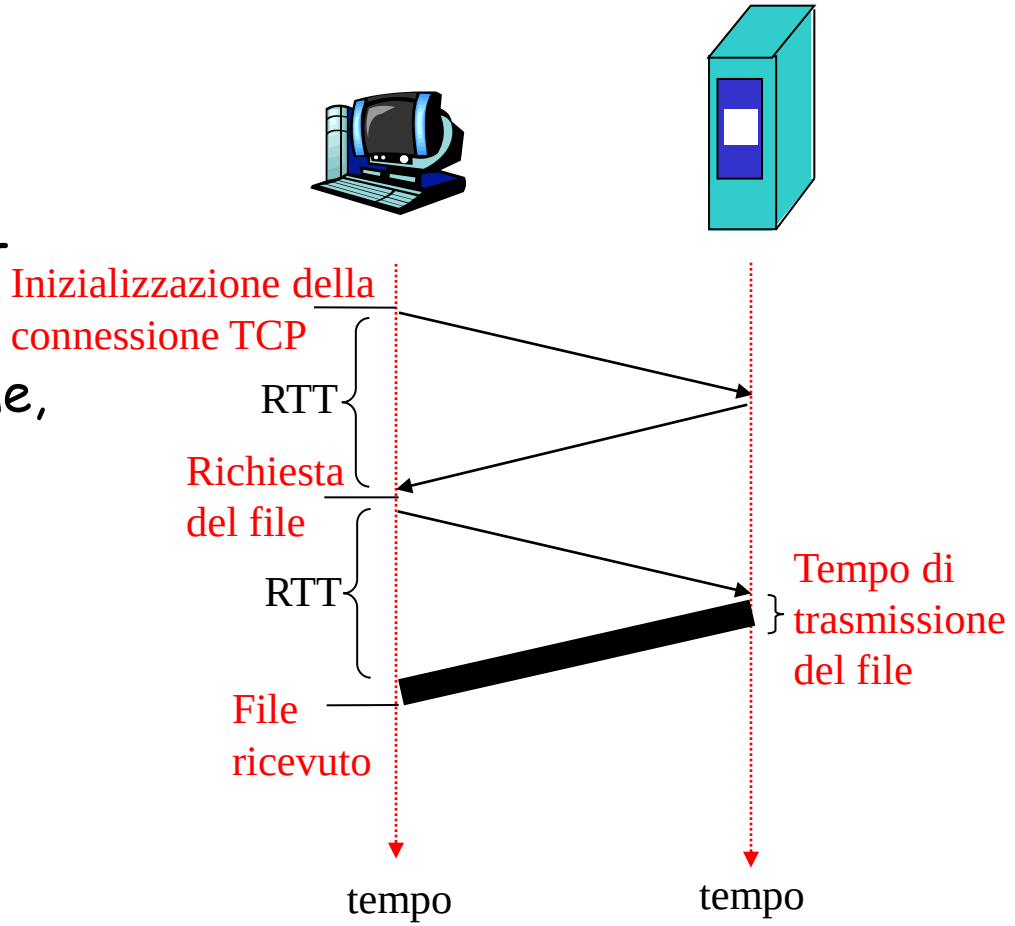
6. I passi 1-5 sono ripetuti per ciascuno dei 10 oggetti jpeg

Schema del tempo di risposta

Definizione di RTT

(Round Trip Time): tempo impiegato da un piccolo pacchetto per andare dal client al server e ritornare al client.

Include i ritardi di propagazione, di accodamento e di elaborazione del pacchetto



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

Connessioni persistenti

Svantaggi delle connessioni non persistenti:

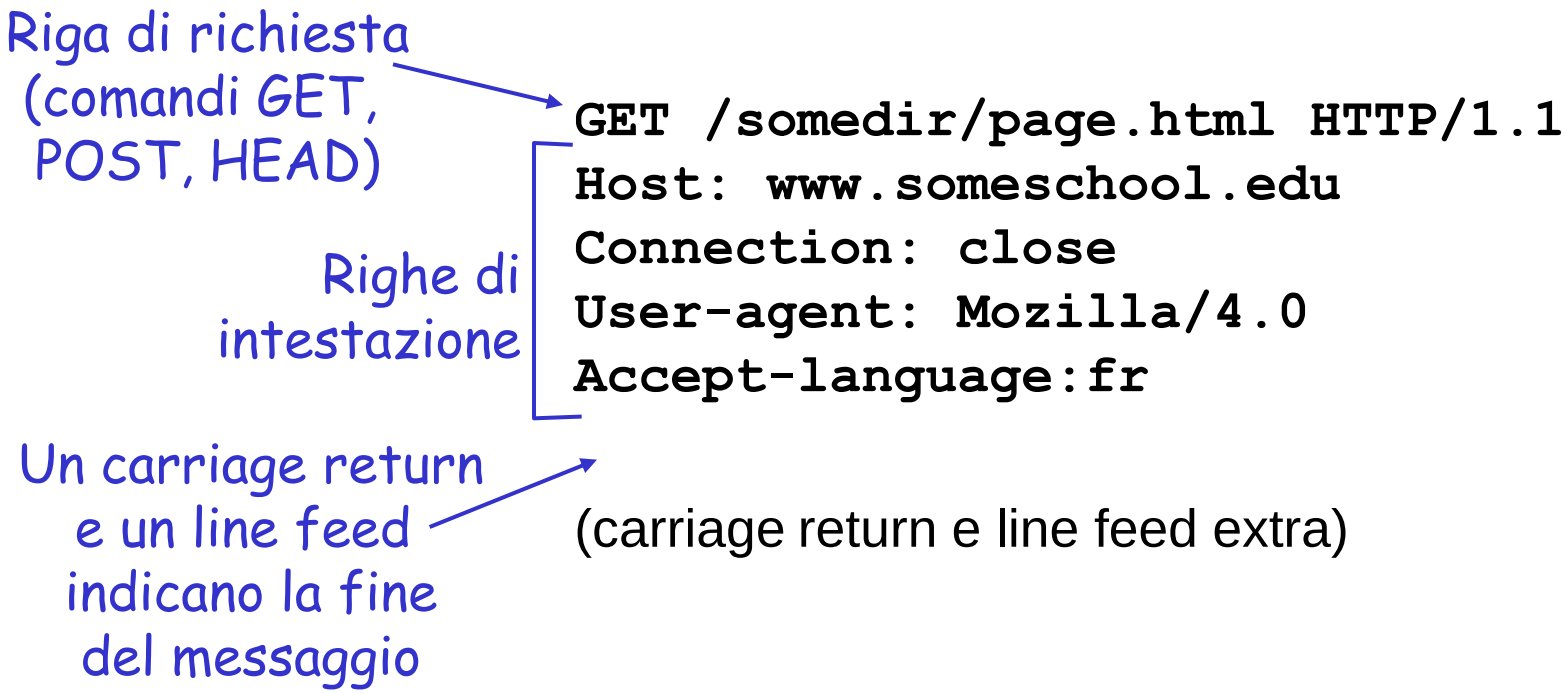
- richiedono 2 RTT per oggetto
- overhead del sistema operativo per *ogni* connessione TCP
- i browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati

Connessioni persistenti

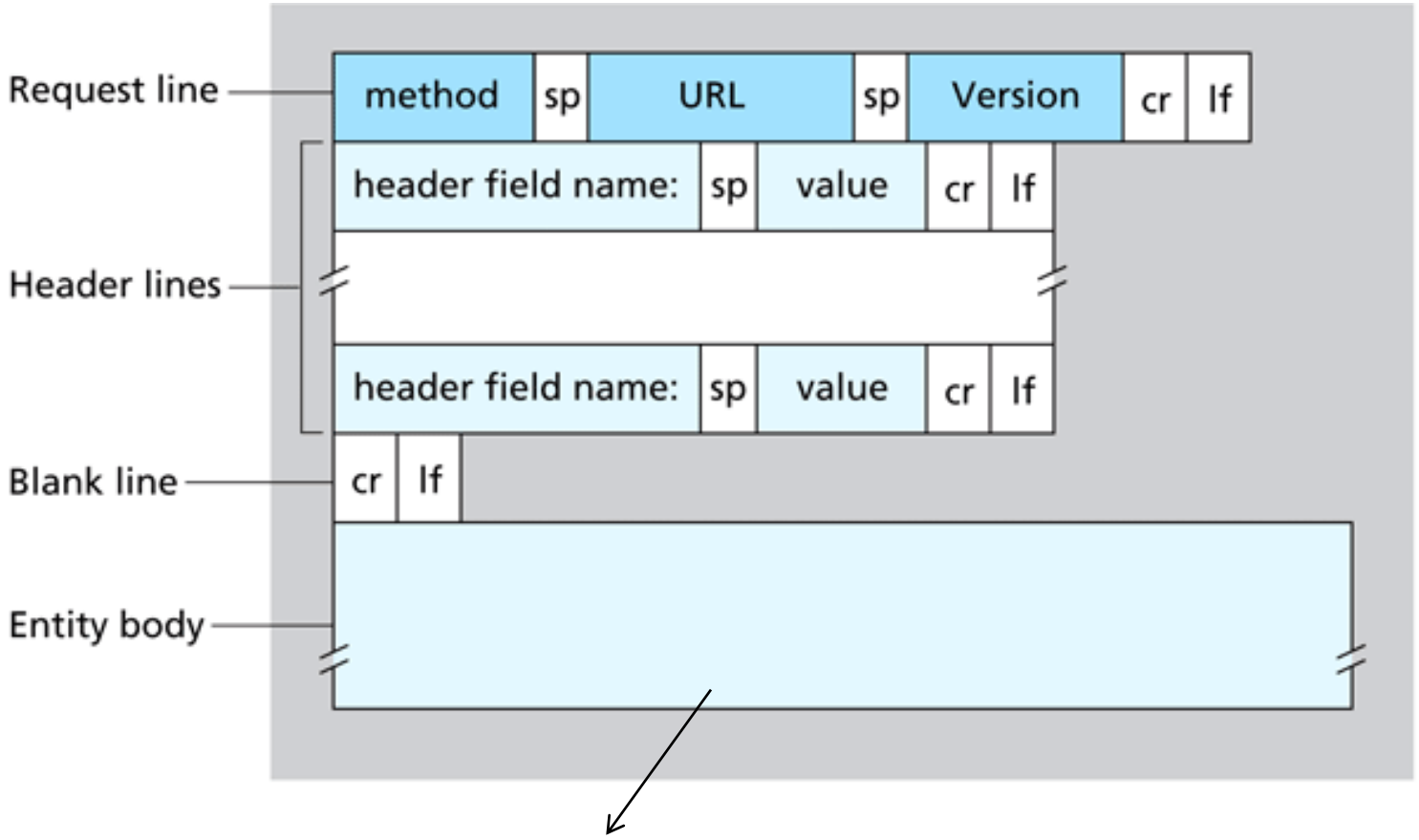
- il server lascia la connessione TCP aperta dopo l'invio di una risposta
- i successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta
- il client invia le richieste non appena incontra un oggetto referenziato
- un solo RTT di connessione per tutti gli oggetti referenziati

Formato dei messaggi HTTP

- Messaggio di richiesta HTTP inviato dal client:
  - ASCII (formato leggibile dall'utente)



Formato generale dei messaggi di richiesta HTTP:



Campo vuoto per il GET, utilizzato per il POST

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

Alternativa

- Si usa il metodo GET inserendo del campo URL l'input richiesto

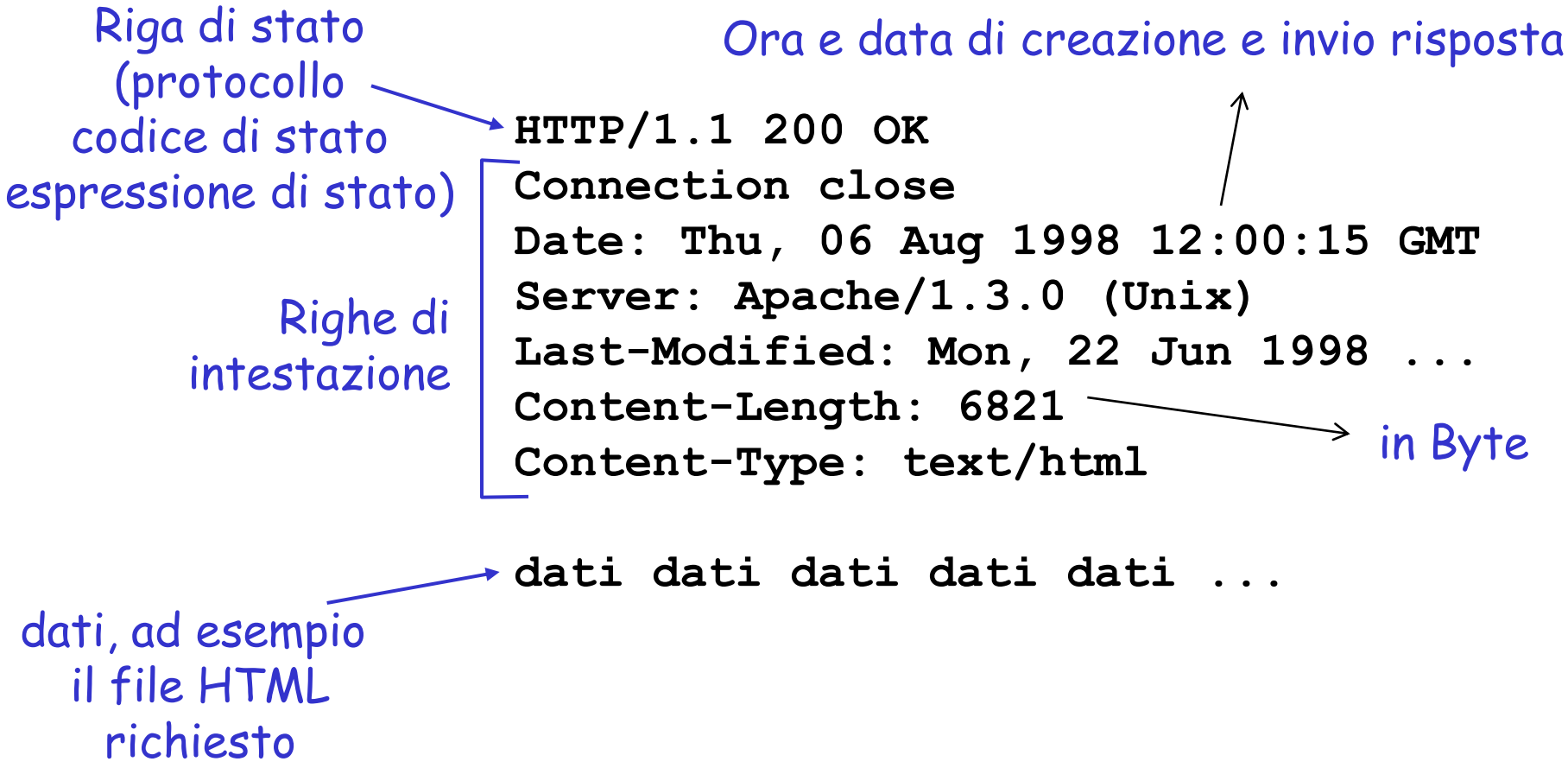
```
www.somesite.com/animalsearch?monkeys&banana
```

Tipi di metodi

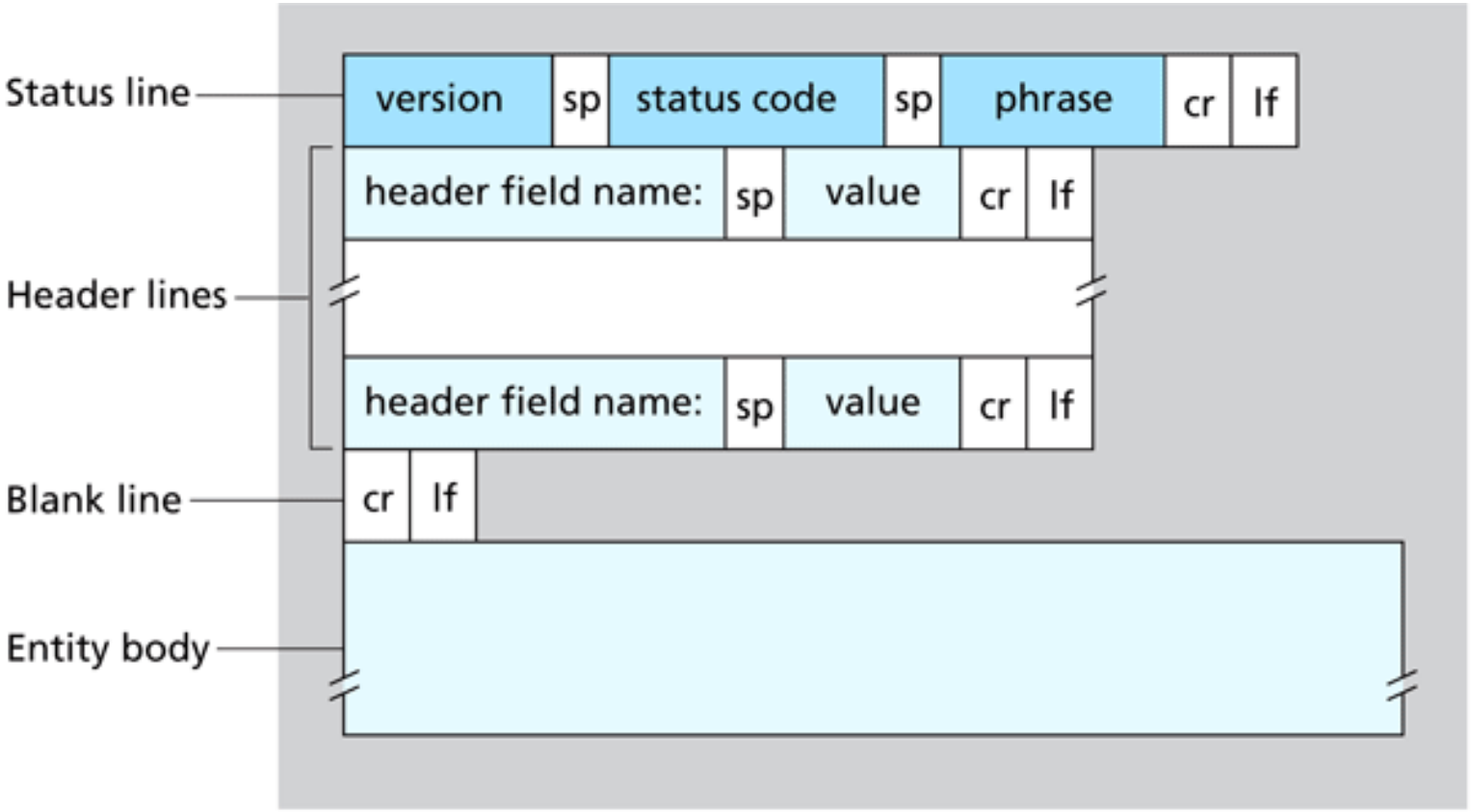
HTTP/1.1 (metodi principali)

- GET: Richiede di leggere una pagina Web
- POST: accoda alla risorsa indicata
- HEAD: richiede di leggere l'intestazione di una pagina Web
- PUT: richiede di memorizzare una pagina Web
- DELETE: rimuove la pagina Web

Messaggio di risposta HTTP



Formato generale dei messaggi di risposta HTTP



Codici di stato della risposta HTTP

Nella prima riga nel messaggio di risposta server->client.

Alcuni codici di stato e relative espressioni:

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

Alcune intestazioni dei msg HTTP

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Client

Client e Server

Server

Prova pratica: HTTP (lato client)

1. Collegatevi via Telnet al vostro server web preferito:

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

Apre una connessione TCP alla 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!

Mancanza di stato

HTTP è un protocollo "senza stato" (stateless)

- Il server una volta servito il client se ne dimentica (non mantiene informazioni sulle richieste fatte)

nota

I protocolli che mantengono lo "stato" sono complessi!

- La storia passata (stato) deve essere memorizzata
- Se il server e/o il client si bloccano, le loro viste dello "stato" potrebbero essere contrastanti e dovrebbero essere riconciliate

Necessità di tenere traccia dell'utente

- Ci sono molti casi in cui il server ha bisogno di ricordarsi degli utenti
  - Siti che richiedono agli utenti di registrarsi. Come possono i server distinguere tra le richieste degli utenti registrati e quelle di altri utenti?
  - Mantenere il carello nei siti di commercio elettronico
- Gli indirizzi IP degli host non sono adatti
  - gli utenti possono lavorare su computer condivisi
  - Molti ISP assegnano lo stesso IP ai pacchetti in uscita provenienti da tutti gli utenti
- Soluzione: Cookie (RFC 2109)
  - Consentono ai siti di tener traccia degli utenti

Interazione utente-server: i cookie

Molti dei più importanti siti web usano i cookie

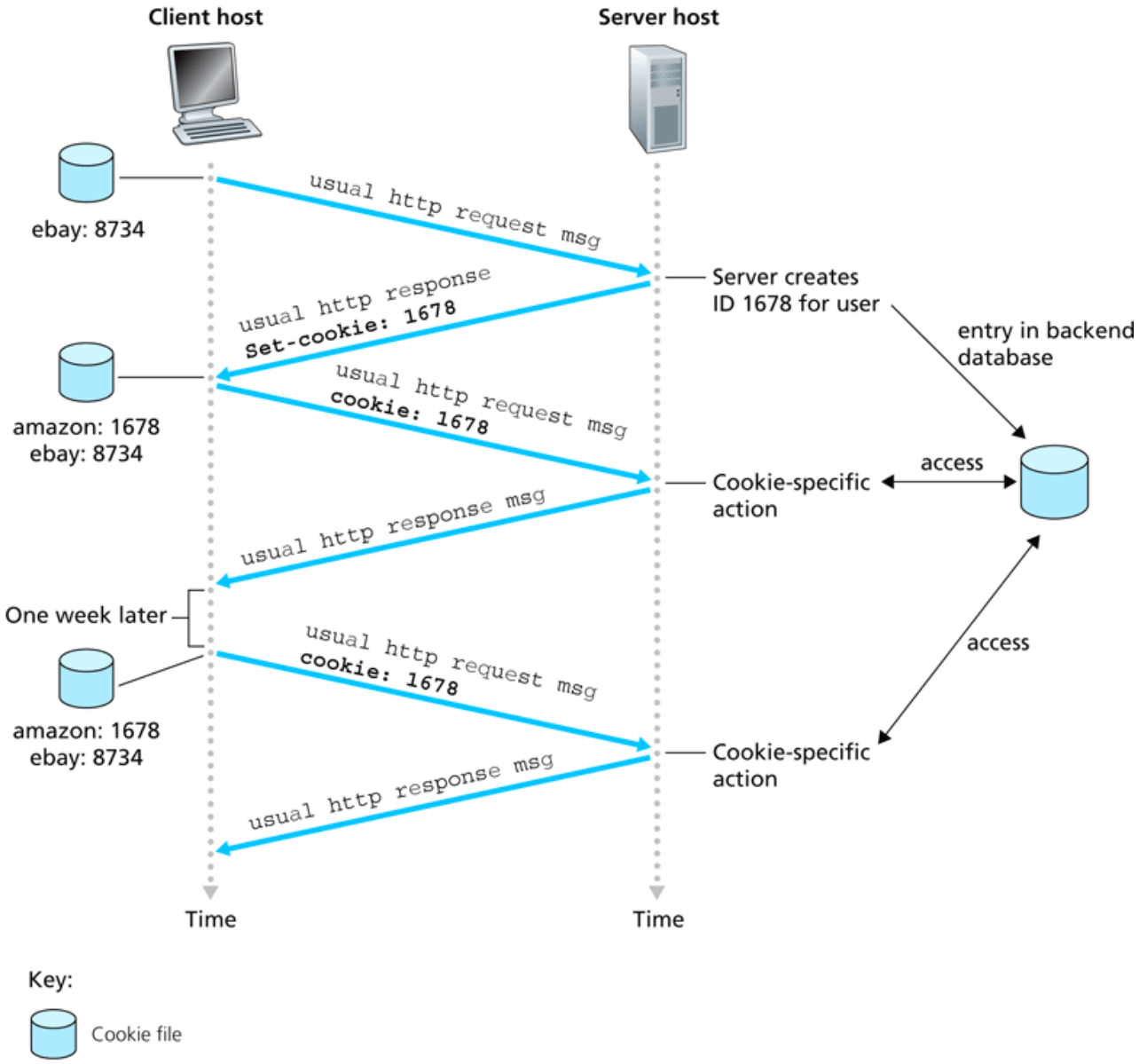
Quattro componenti:

1) Una riga di intestazione nel messaggio di *risposta* HTTP
2) Una riga di intestazione nel messaggio di *richiesta* HTTP
3) Un file cookie mantenuto sul sistema terminale dell'utente e gestito dal browser dell'utente
4) Un database sul sito

Esempio:

- 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 per ID

Cookie (continua)



Cookie (continua)

Cosa possono contenere i cookie:

- autorizzazione
- carta per acquisti
- raccomandazioni
- stato della sessione dell'utente (e-mail)

Lo "stato"

- Mantengono lo stato del mittente e del ricevente per più transazioni
- I messaggi http trasportano lo stato

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

Caching

Obiettivo: migliorare le prestazioni

- Un modo semplice consiste nel salvare le pagine richieste per riutilizzarle in seguito senza doverle richiedere al server
- Tecnica efficiente con pagine che vengono visitate molto spesso
- L'accumulo delle pagine per un utilizzo successivo è definito caching
- Il caching può essere eseguito da
  - Browser
  - Proxy

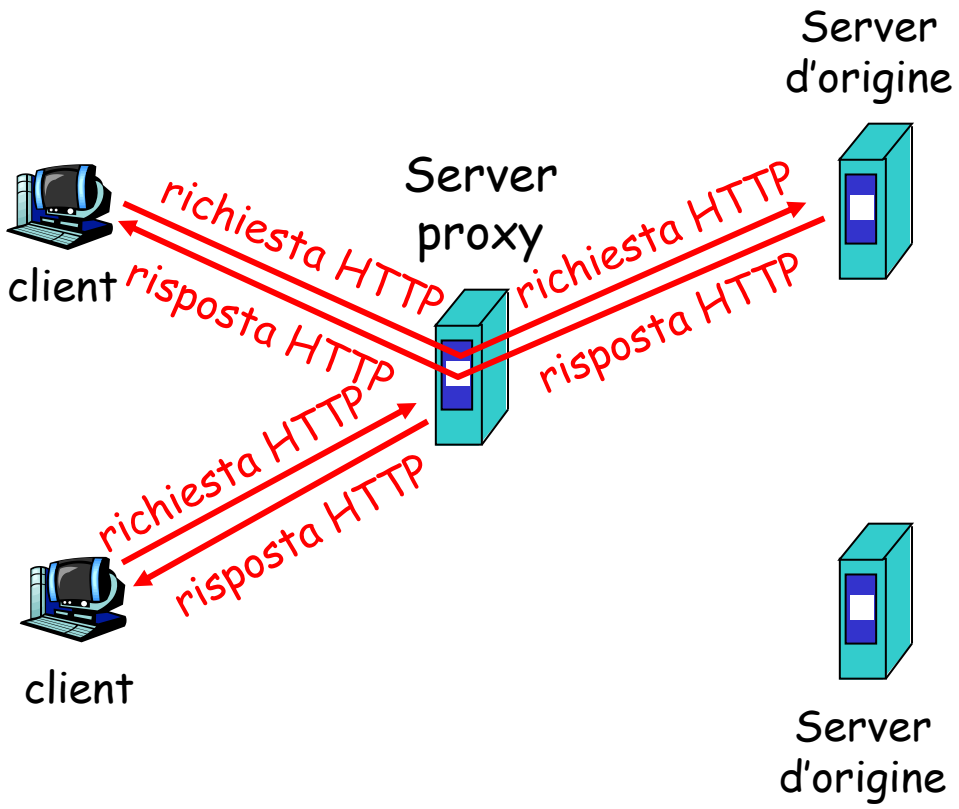
Browser Caching

- Il browser può mantenere una cache delle pagine visitate
- La cache è personalizzabile dall'utente
- Esistono vari meccanismi per la gestione della cache locale
  - L'utente può impostare il numero di giorni dopo i quali i contenuti della cache vengono cancellati e l'eventuale gestione
  - La pagina può essere mantenuta in cache in base alla sua ultima modifica (es. modificata un'ora prima -> mantenuta per un'ora, oppure un giorno, un mese, etc.)
  - Si possono utilizzare informazioni nei campi intestazione dei messaggi per gestire la cache
    - Es: campo Expires specifica la scadenza dopo la quale la pagina è considerata obsoleta (stale)
    - Non sempre rispettato dai browser

Web Caching (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
- Tipicamente la cache è installata da un ISP (università, aziende o ISP residenziali)

Perché il caching web?

- 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

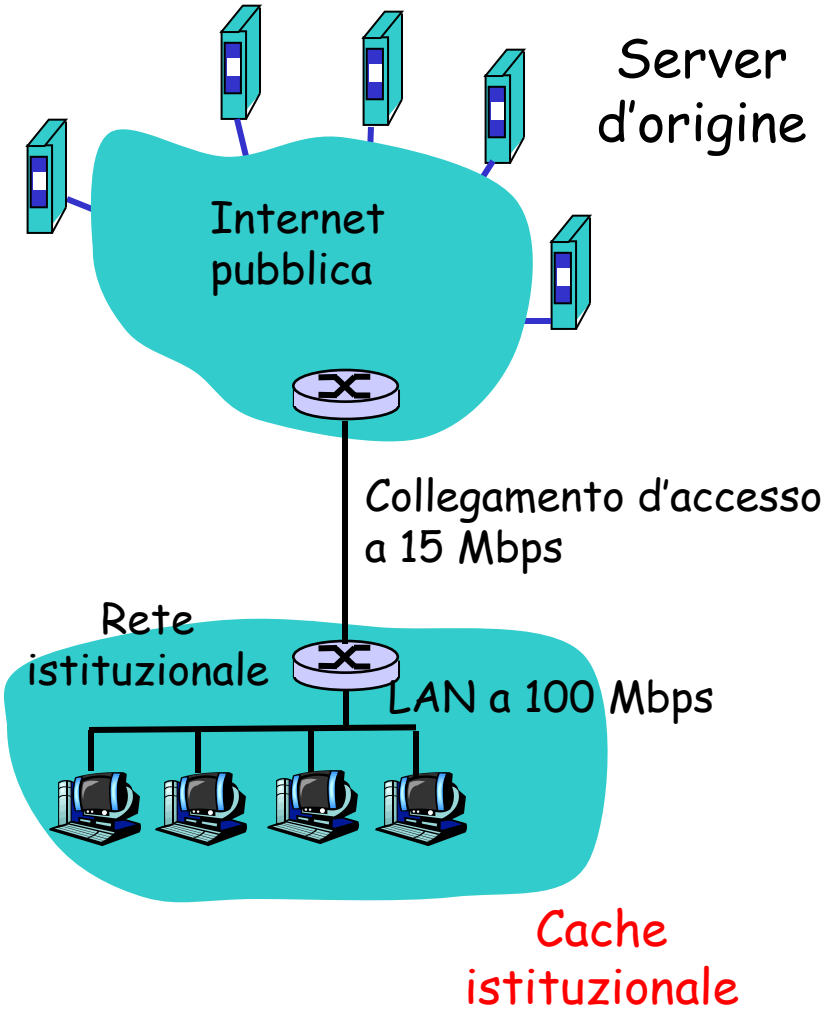
Ipotesi

- Dimensione media di un oggetto = 1Mbps
- Frequenza media di richieste dai browser istituzionali ai server d'origine = 15/sec

Conseguenze

- Valutiamo l'intensità di traffico La/R
- utilizzazione sulla LAN = (15*1Mbps)/100Mbps = 15%
- utilizzazione sul collegamento d'accesso = (15*1Mbps)/15Mbps = 100%
- La/R -> 1: il ritardo si fa consistente
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN

= 2 sec + minuti + millisecondi



Esempio di caching (continua)

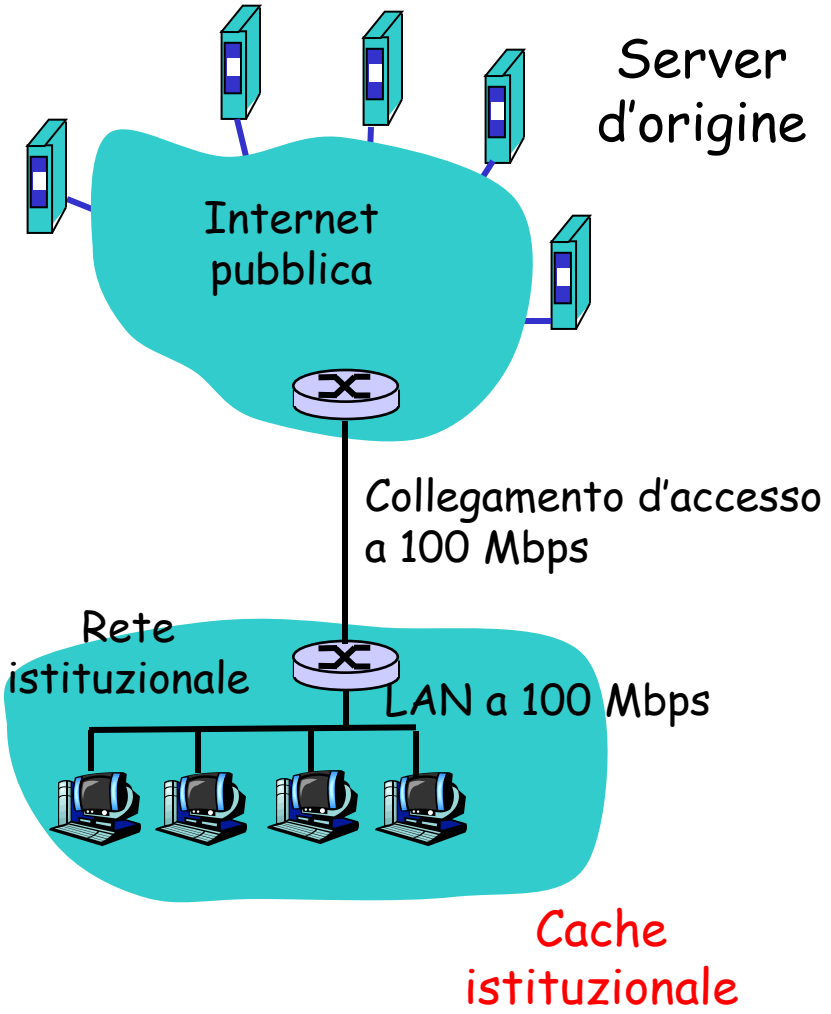
Soluzione possibile

- aumentare l'ampiezza di banda del collegamento d'accesso a 100 Mbps, per esempio

Conseguenze

- utilizzo sulla LAN = 15%
- utilizzo sul collegamento d'accesso = 15%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN

  = 2 sec + msecs + msecs
- Non sempre attuabile e comunque costo aggiornare il collegamento



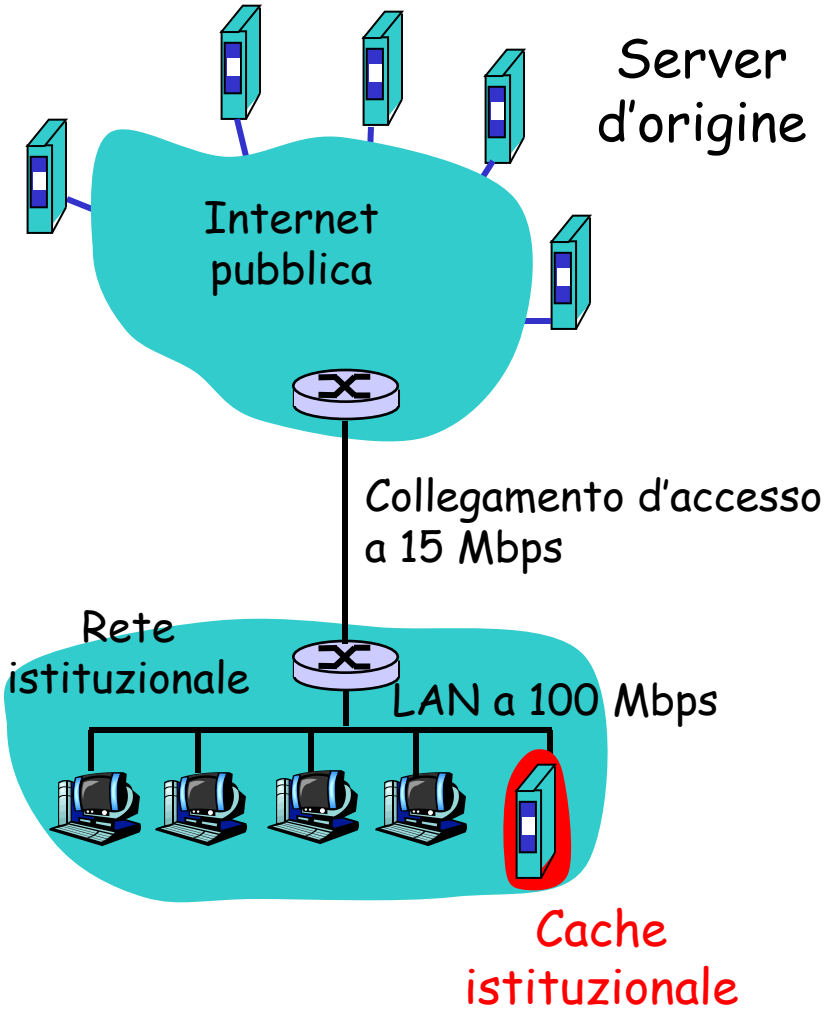
Esempio di caching (continua)

Soluzione possibile: installare la cache

- supponiamo una percentuale di successo (*hit rate*) pari a 0,4

Conseguenze

- il 40% delle richieste sarà soddisfatto quasi immediatamente
- il 60% delle richieste sarà soddisfatto dal server d'origine
- l'utilizzazione del collegamento d'accesso si è ridotta al 60%, determinando ritardi trascurabili (circa 10 msec)
- ritardo totale medio = ritardo di Internet + ritardo di accesso + ritardo della LAN =



0,6*(2,01) sec + millisecondi < 1,4 sec

Verifica della cache

- La copia di un oggetto che risiede in cache potrebbe essere scaduta (pagina modificata sul server)
- HTTP presenta un meccanismo per verificare se gli oggetti in cache sono aggiornati
- **GET Condizionale**:
  - Usa metodo GET
  - Include una riga di intestazione `If-Modified-Since`
- Esempio:
  - **Client** invia messaggio di richiesta

    ```
    GET /page/figure.gif
    Host: www.xxx.com
    ```
  - **Server** invia il messaggio di risposta

    ```
    HTTP/1.1 200 OK
    Date: …
    …
    Last-Modified: Wed, 2 Jul 2008 09:23:24
    (data data …)
    ```
  - La cache memorizza la pagina per richieste future, mantenendo la data di ultima modifica

GET condizionale

- Obiettivo: non 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`