DIPARTIMENTO
DI INFORMATICA
Via Salaria 113 - 00198, Roma

SAPIENZA
UNIVERSITÀ DI ROMA

Livello applicazione: protocollo HTTP, cookie, web cache

Prof.ssa Gaia Maselli

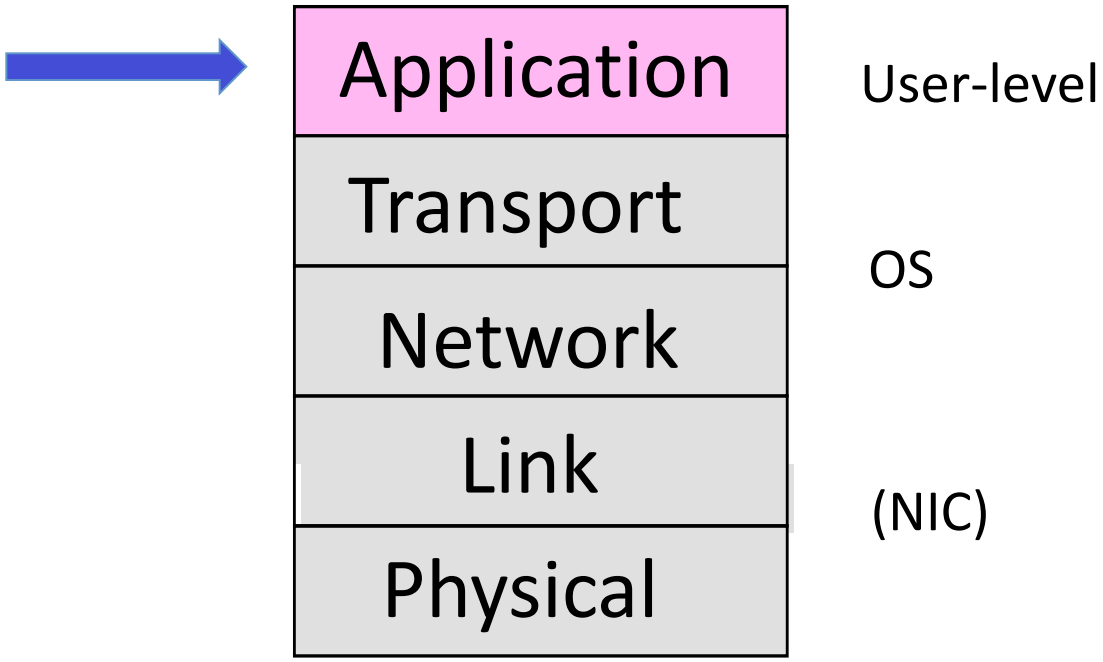
maselli@di.uniroma1.it

Parte di queste slide sono state prese dal materiale associato ai libri:

1) B.A. Forouzan, F. Mosharraf – Reti di calcolatori. Un approccio top-down. Copyright © 2013 McGraw-Hill Education Italy srl. Edizione italiana delle slide a cura di Gabriele D'Angelo e Gaia Maselli

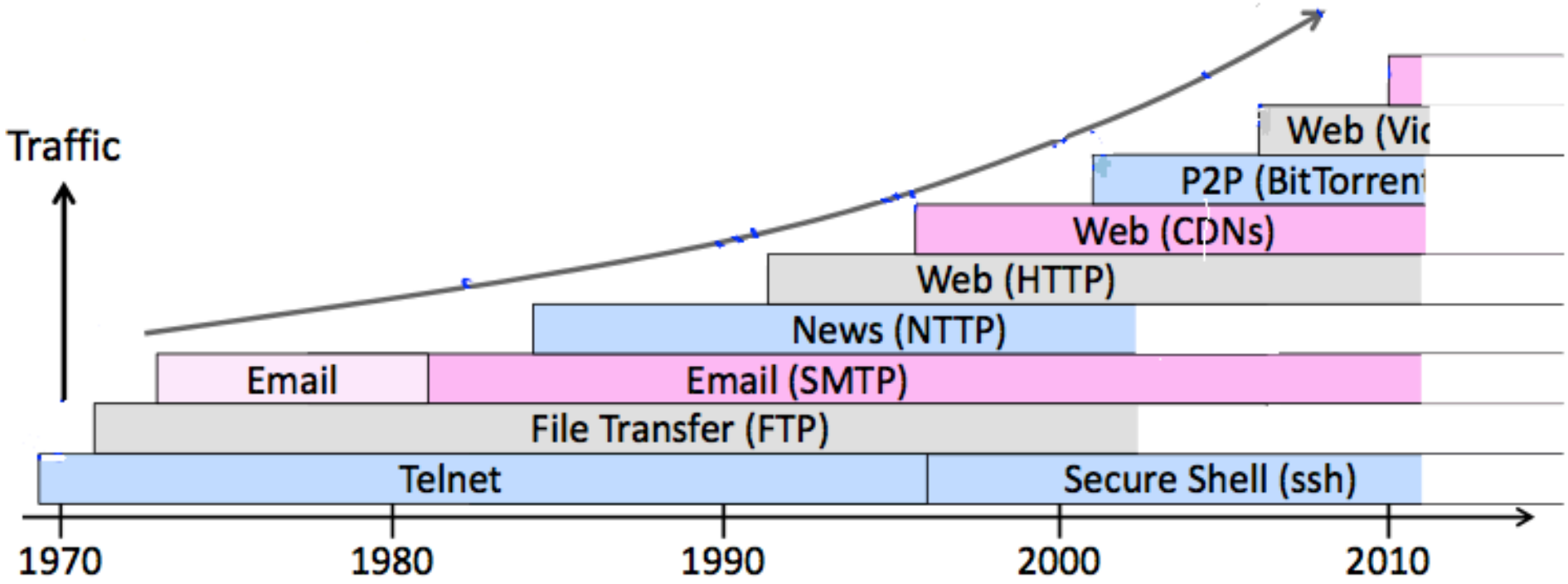
2) Computer Networking: A Top Down Approach , 6th edition. All material copyright 1996-2009 J.F Kurose and K.W. Ross, All Rights Reserved

Dove siamo



Application layer protocols are often part of an “app”

Evoluzione apps Internet



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 utilizzata soprattutto per scopi universitari (1990)
- 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.

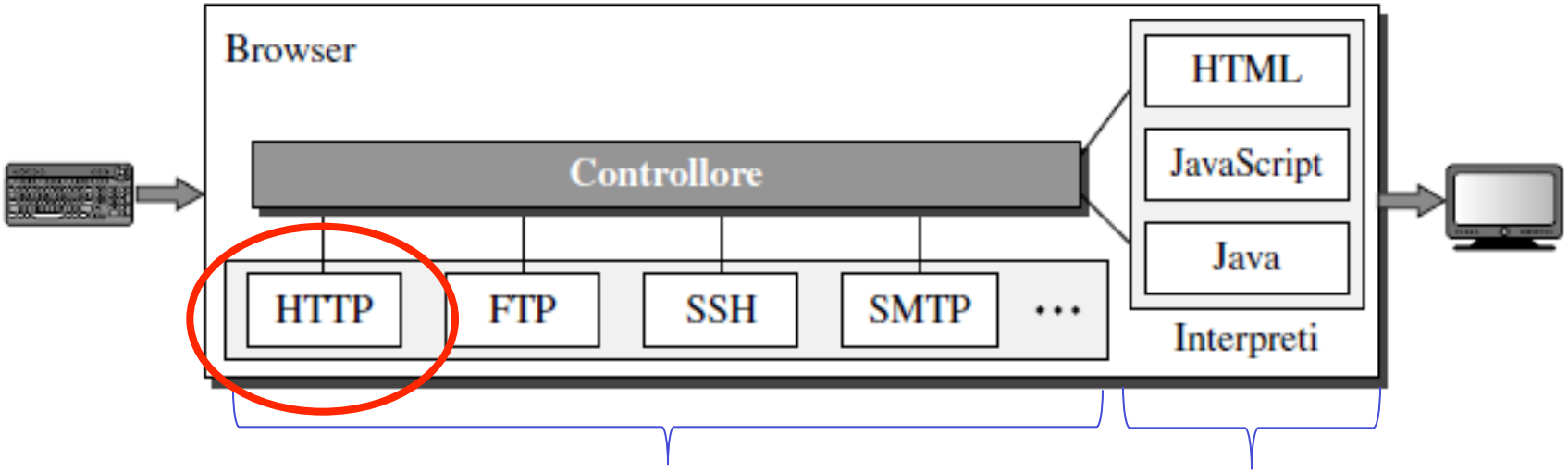
Caratteristiche

- Opera su richiesta (on demand)
- Facile rendere informazioni disponibili
- Collegamenti ipertestuali (hyperlinks) e motori di ricerca permettono di navigare su una grande quantità di siti

The World Wide Web application (WWW o Web)

- Componenti:
  - Web client (es. browser): interfaccia con l'utente
  - Web server (es. Apache)
  - HTML: linguaggio standard per pagine web
  - HTTP: protocollo per la comunicazione tra client e server Web (indipendente dal linguaggio di programmazione usato)

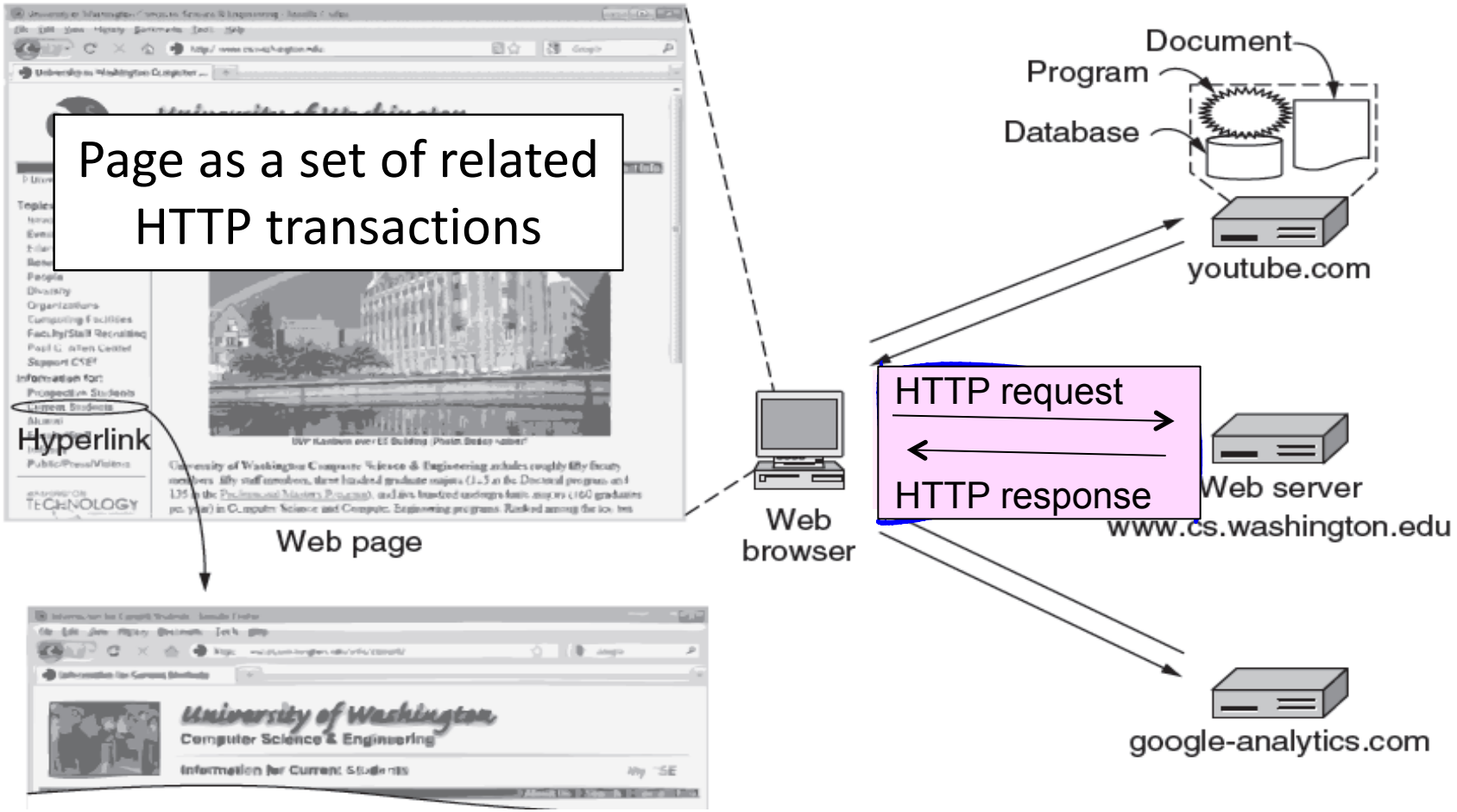
Architettura generale di un browser



Protocolli di comunicazione usati per accedere ai documenti richiesti dall'utente (lato client)

Necessari per visualizzare il documento sullo schermo

Web


Page as a set of related HTTP transactions
Hyperlink
Web page
Document
Program
Database
youtube.com
HTTP request
HTTP response
Web browser
Web server
www.cs.washington.edu
google-analytics.com
University of Washington
Computer Science & Engineering

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
```

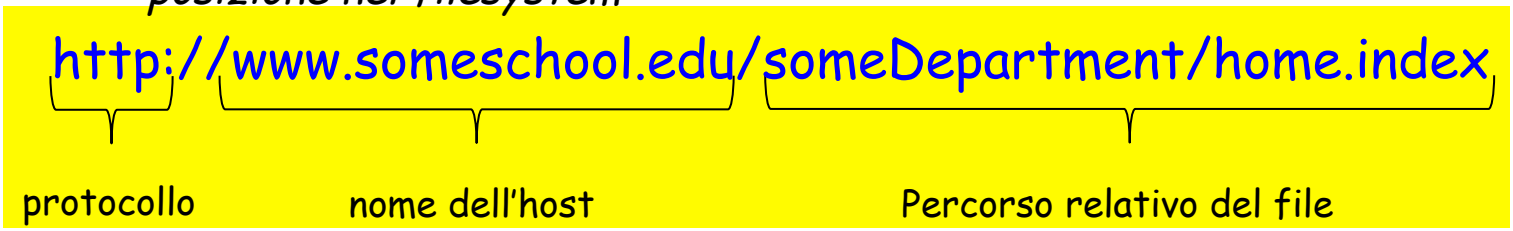
URL

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 **percorso del file** (localmente alla macchina) che indica la pagina il nome del file e la posizione nel filesystem*



protocol://host/path	utilizzato quando la porta è standard
protocol://host:**porta**/path	utilizzato quando è necessario specificare il numero di porta

Documenti Web

- Documento statico
  - Contenuto predeterminato memorizzato sul server
- Documento dinamico
  - Creato dal web server alla ricezione della richiesta (es. *date*)
- Documento attivo
  - Contiene script o programmi che verranno eseguiti nel browser ovvero lato client (es. applet java)

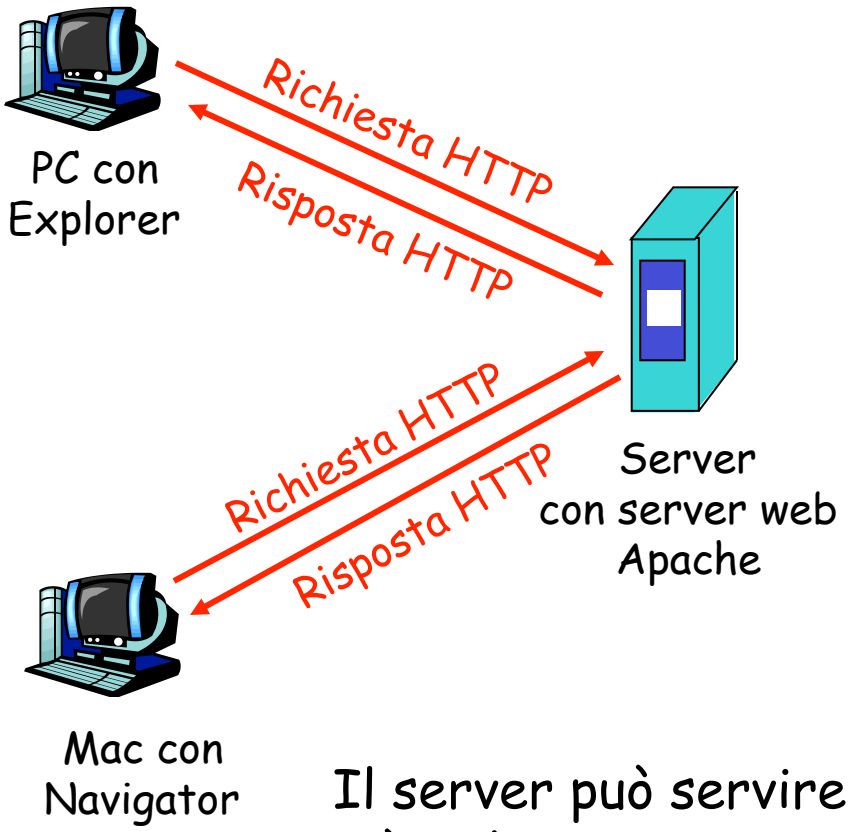
Come vengono recuperati questi documenti?

Come interagiscono client e server?

Panoramica su HTTP (RFC 2616)

HTTP: hypertext transfer protocol

- Protocollo a livello di applicazione del Web
- Modello client/server
  - *client:* il browser 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**



Il server può servire più richieste, provenienti anche da client diversi

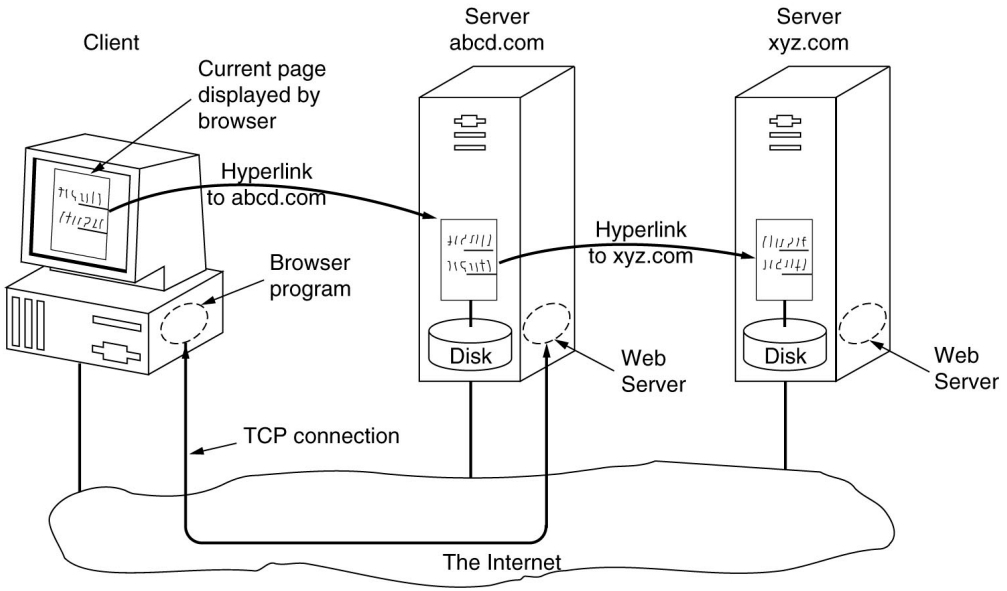
Panoramica su HTTP (continua)

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

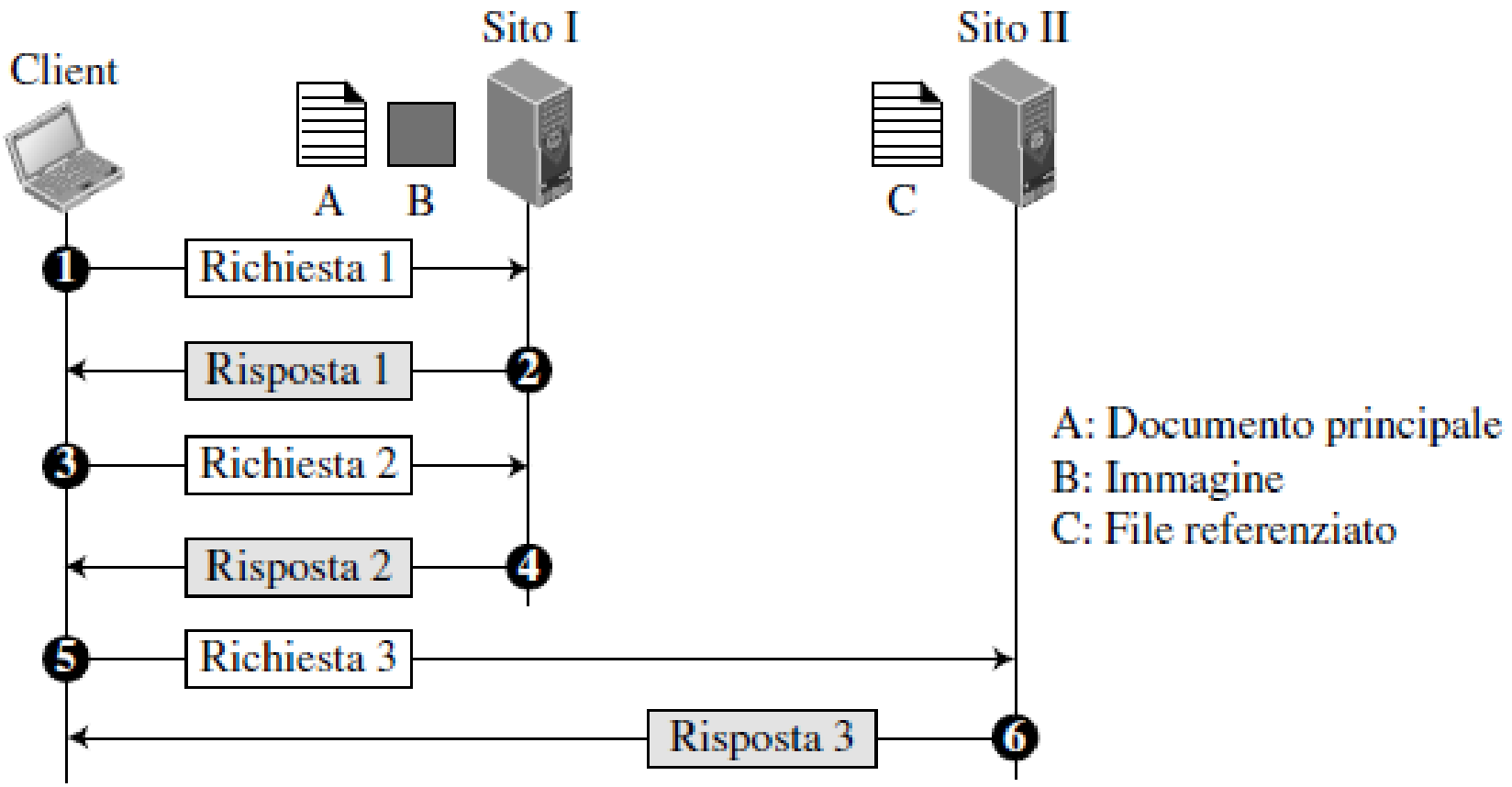
1. Il browser determina l'URL ed estrae host e filename
2. Esegue connessione TCP alla porta 80 dell'host indicato nella URL
3. Invia richiesta per il file
4. Riceve il file dal server
5. Chiude la connessione
6. Visualizza il file

Lato **Server**

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



Riferimenti ad atri documenti



Connessioni HTTP

Connessioni non persistenti

- Un solo oggetto viene trasmesso su una connessione TCP (un file http, un'immagine, etc.)
- Ciascuna coppia richiesta/ risposta viene inviata su una connessione TCP separata
- Prima di inviare una richiesta al server è necessario stabilire una connessione

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

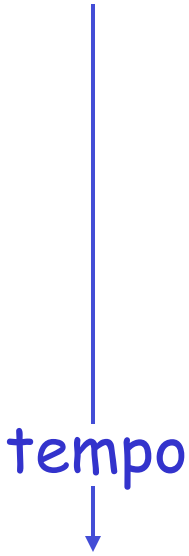
1b. Il server HTTP all'host www.someSchool.edu in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client

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, crea 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

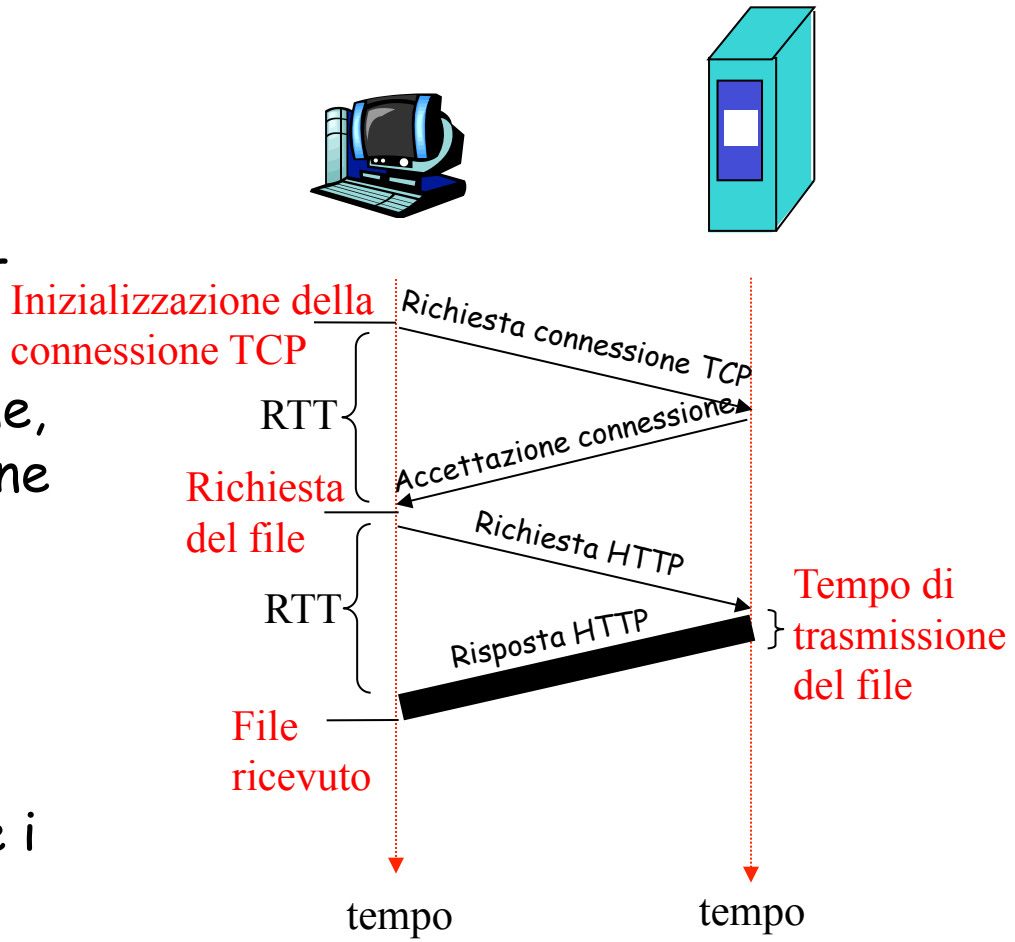
Definizione di RTT

(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 per la richiesta HTTP e i primi byte della risposta HTTP
- tempo di trasmissione del file



totale = 2RTT + tempo di trasmissione

Connessioni persistenti

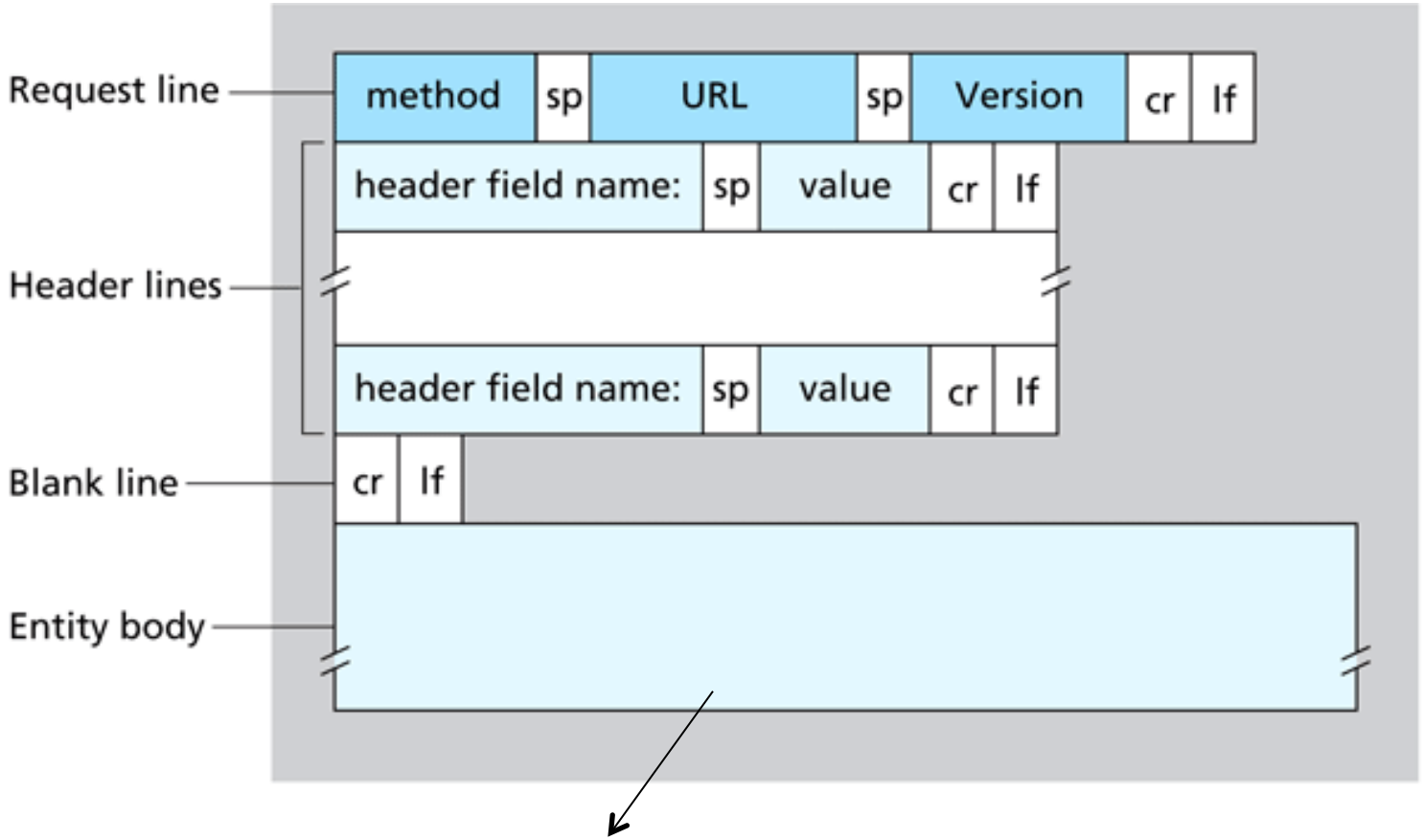
Svantaggi delle connessioni non persistenti:

- 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 (da 5 a 10 connessioni)

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 (+ un RTT per ogni oggetto ricevuto dal server)

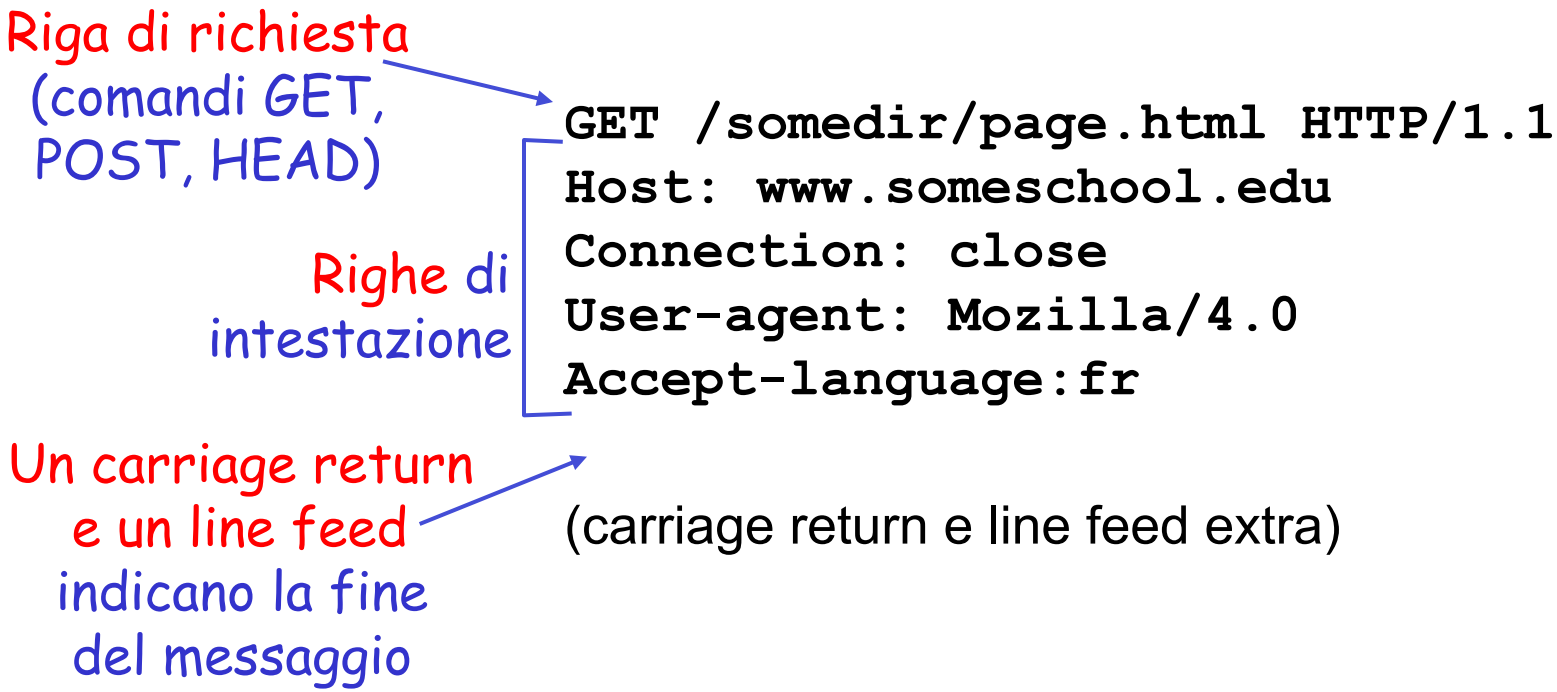
Formato generale dei messaggi di richiesta HTTP:



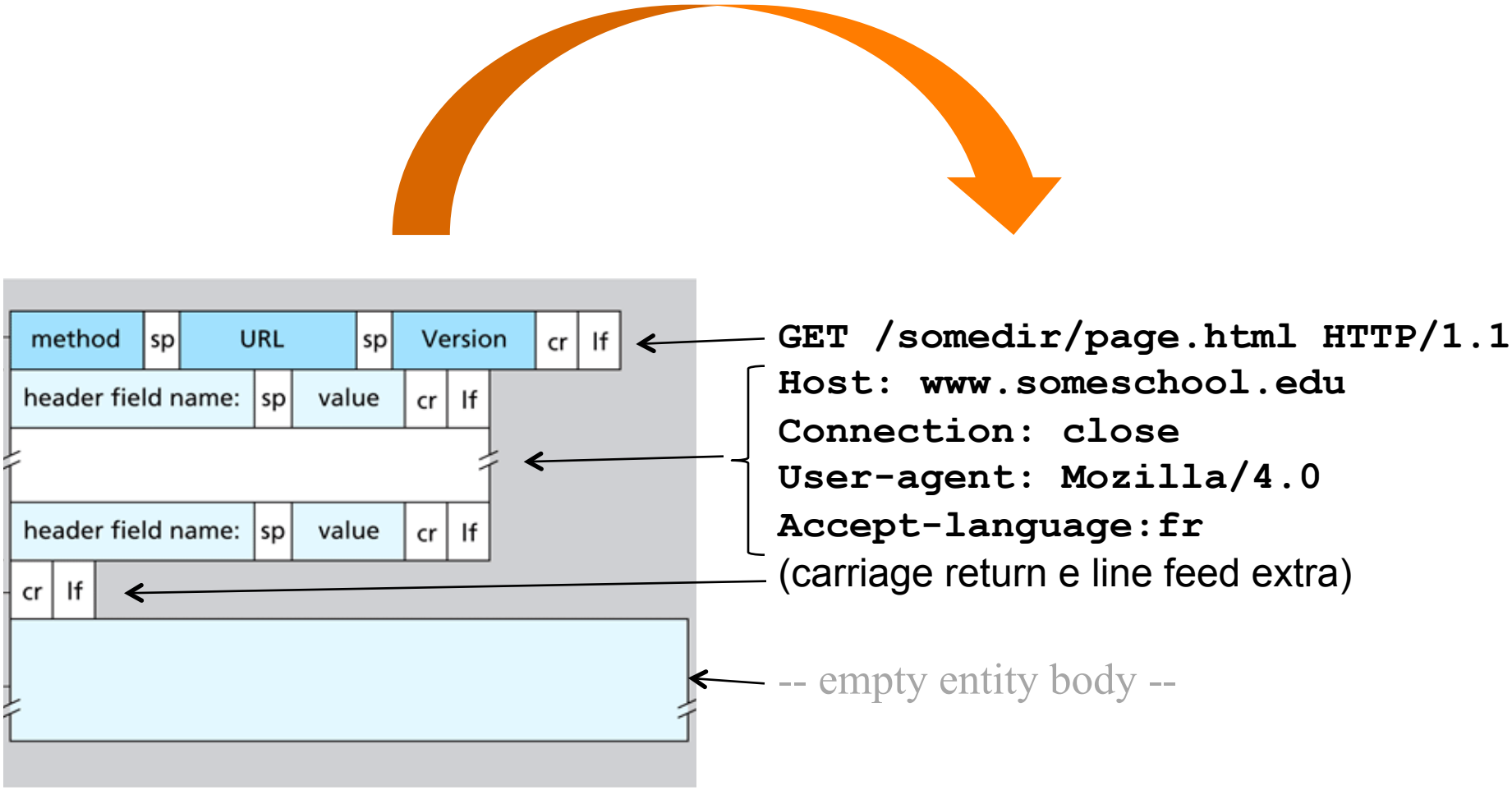
Campo vuoto per il GET, utilizzato per il POST

Esempio richiesta HTTP

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



Formato vs. esempio



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)

GET	E' usato quando il client vuole scaricare un documento dal server. Il documento richiesto è specificato nell'URL. Il server normalmente risponde con il documento richiesto nel corpo del messaggio di risposta.
HEAD	E' usato quando il client non vuole scaricare il documento ma solo alcune informazioni sul documento (come ad esempio la data dell'ultima modifica). Nella risposta il server non inserisce il documento ma solo degli header informativi.
POST	E' usato per fornire input al server (contenuto dei campi di un form). L'input arriva al server nel corpo dell'entità.
PUT	E' utilizzato per memorizzare un documento nel server. Il documento viene fornito nel corpo del messaggio e la posizione di memorizzazione nell'URL.

Per inviare info al server

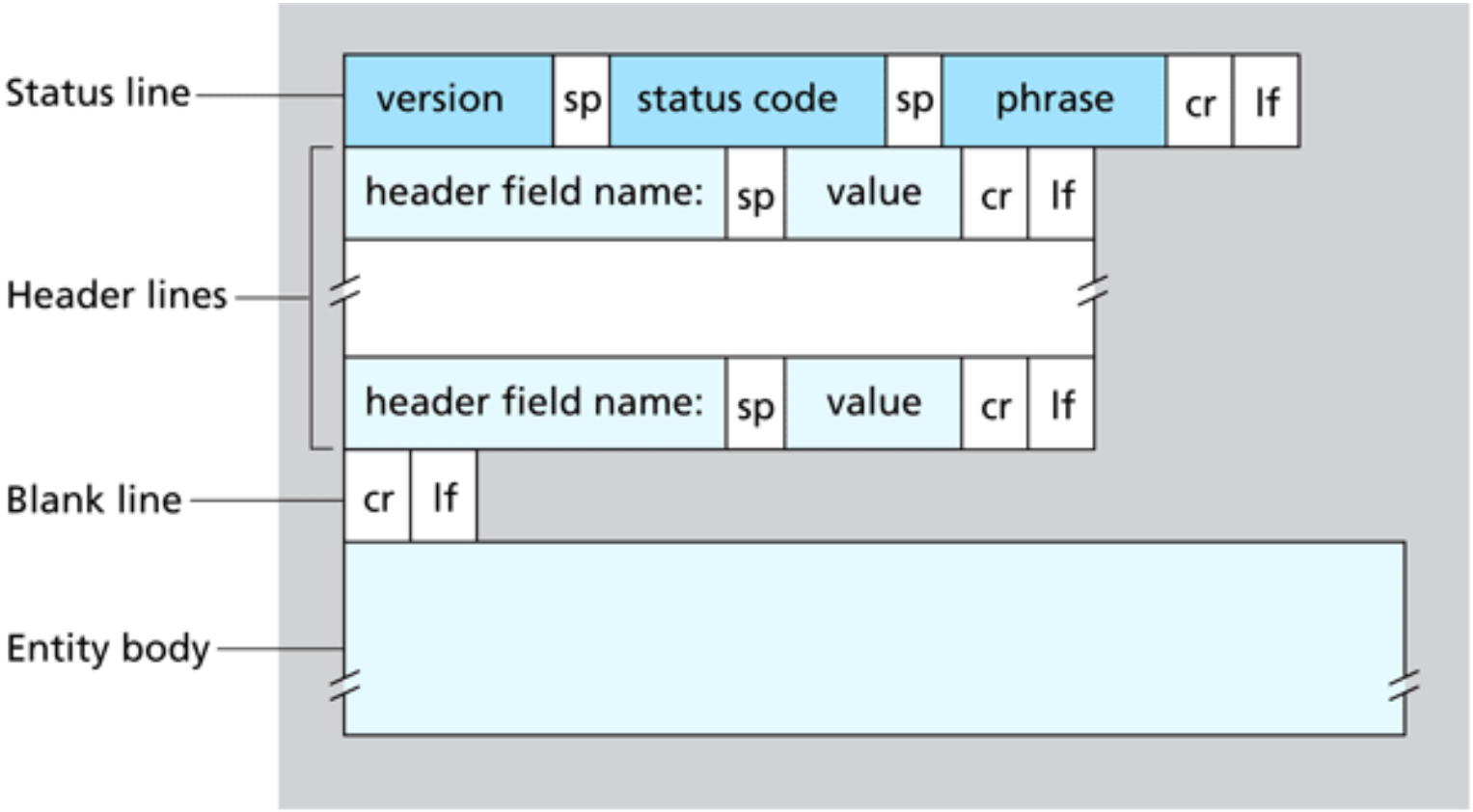
E' possibile anche usare GET

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

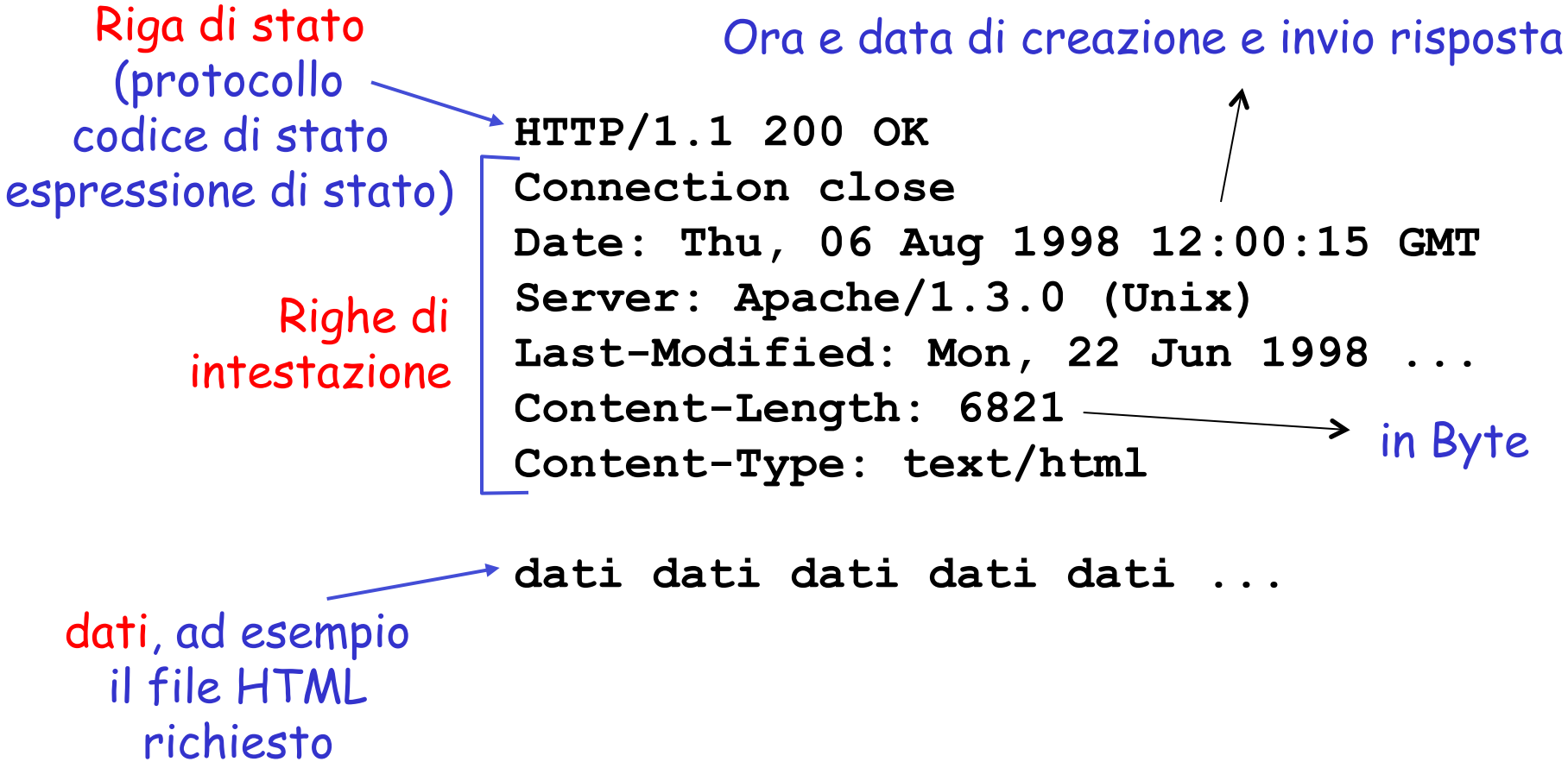
Intestazioni di richiesta

Intestazione	*Descrizione*
User-agent	Indica il programma client utilizzato
Accept	Indica il formato dei contenuti che il client è in grado di accettare
Accept-charset	Famiglia di caratteri che il client è in grado di gestire
Accept-encoding	Schema di codifica supportato dal client
Accept-language	Linguaggio preferito dal client
Authorization	Indica le credenziali possedute dal client
Host	Host e numero di porta del client
Date	Data e ora del messaggio
Upgrade	Specifica il protocollo di comunicazione preferito
Cookie	Comunica il cookie al server (verrà spiegato successivamente)
If-Modified-Since	Invia il documento solo se è più recente della data specificata

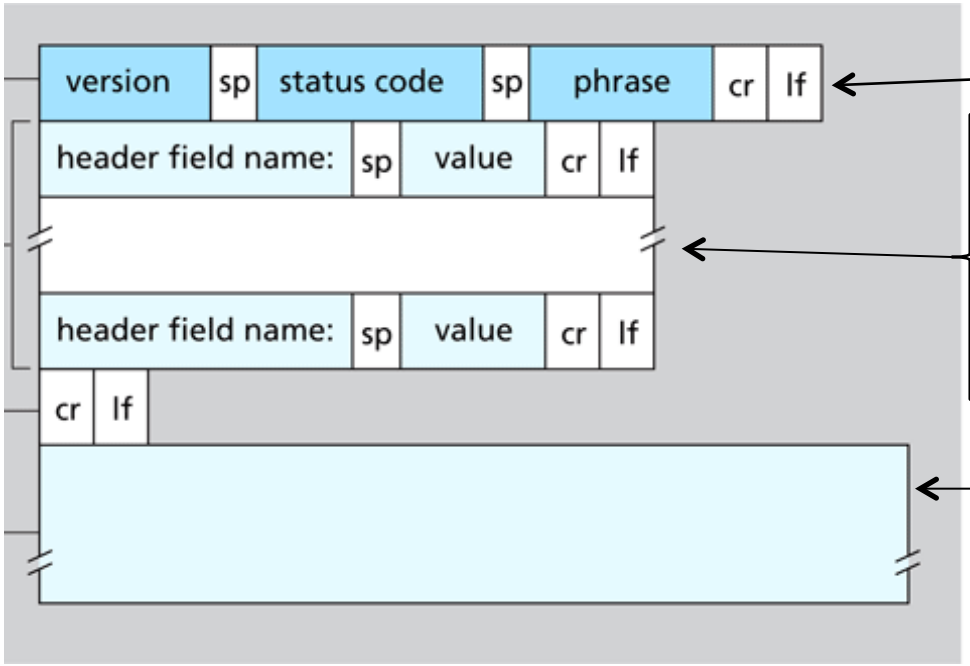
Formato generale dei messaggi di risposta HTTP



Messaggio di risposta HTTP



Formato vs. esempio




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

dati dati dati dati dati ...
```

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

Codici di risposta

Yes! →

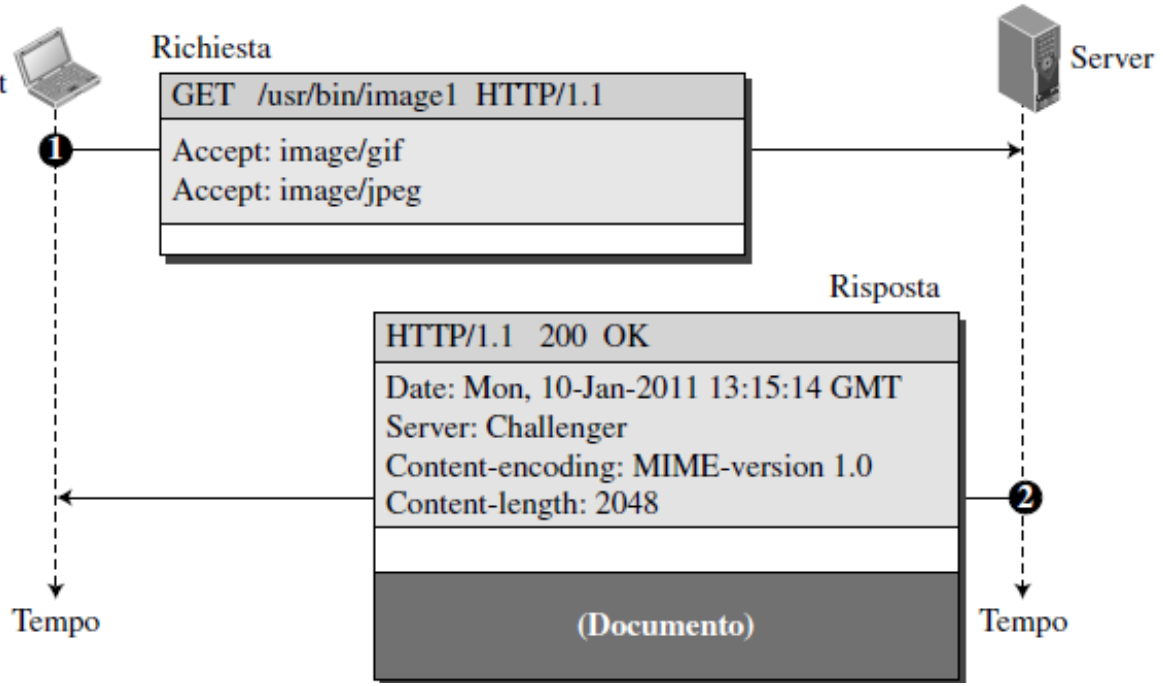
Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

Intestazioni di risposta

Intestazione	*Descrizione*
Date	Data corrente
Upgrade	Specifica il protocollo preferito
Server	Indica il programma server utilizzato
Set-Cookie	Il server richiede al client di memorizzare un cookie
Content-Encoding	Specifica lo schema di codifica
Content-Language	Specifica la lingua del documento
Content-Length	Indica la lunghezza del documento
Content-Type	Specifica la tipologia di contenuto
Location	Chiede al client di inviare la richiesta a un altro sito
Last-modified	Fornisce data e ora di ultima modifica del documento

Esempio GET

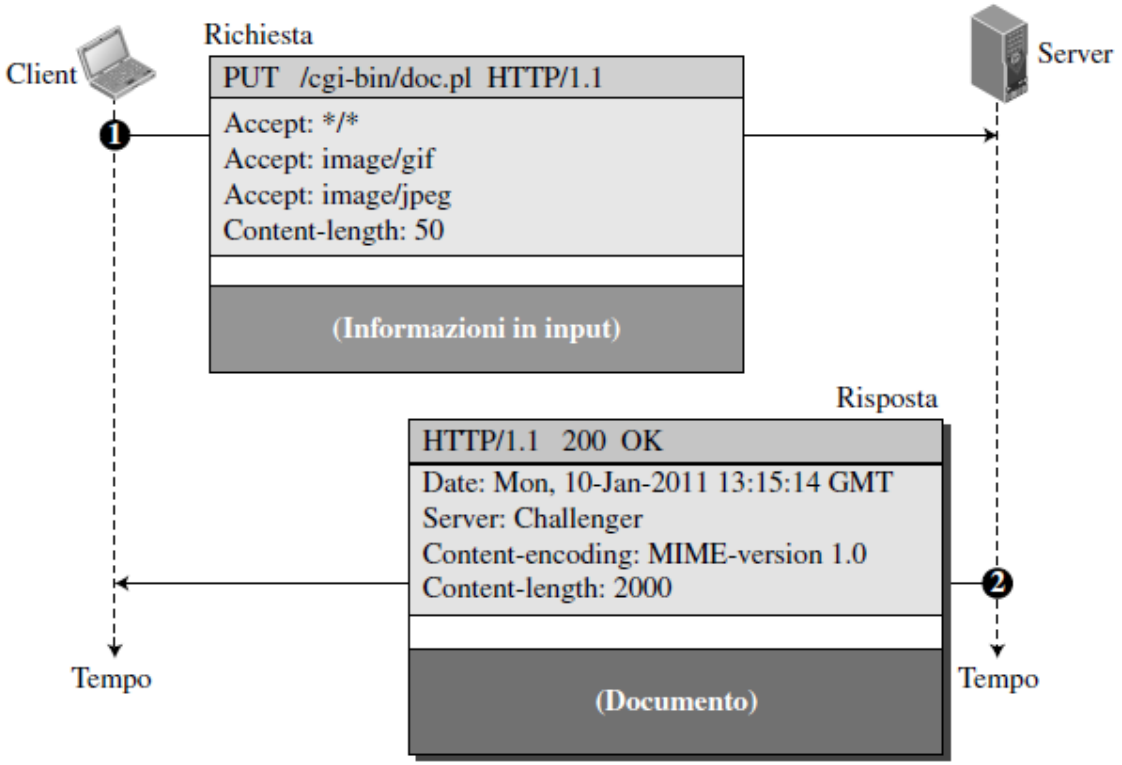
- Il client preleva un documento: viene usato il metodo GET per ottenere l'immagine individuata dal percorso /usr/bin/image1.



- La riga di richiesta contiene il metodo (GET), l'URL e la versione (1.1) del protocollo HTTP. L'intestazione è costituita da due righe in cui si specifica che il client accetta immagini nei formati GIF e JPEG. Il messaggio di richiesta non ha corpo.
- Il messaggio di risposta contiene la riga di stato e quattro righe di intestazione che contengono la data, il server, il metodo di codifica del contenuto (la versione MIME, argomento che verrà descritto nel paragrafo dedicato alla posta elettronica) e la lunghezza del documento.
- Il corpo del messaggio segue l'intestazione.

Esempio PUT

- Il client spedisce al server una pagina Web da pubblicare. Si utilizza il metodo PUT.



- La riga di richiesta contiene il metodo (PUT), l'URL e la versione (1.1) del protocollo HTTP. L'intestazione è costituita da quattro righe d'intestazione. Il corpo del messaggio di richiesta contiene la pagina Web inviata.
- Il messaggio di risposta contiene la riga di stato e quattro righe di intestazione.
- Il documento creato, un documento CGI, è incluso nel corpo del messaggio di risposta.

Prova pratica: HTTP (lato client)

1. Collegatevi via Telnet al server web:

```
telnet reti.dsi.uniroma1.it 80
```

Apre una connessione TCP alla porta 80 (porta di default per un server HTTP) dell'host reti.dsi.uniroma1.it
Tutto ciò che digitate viene trasmesso alla porta 80 di reti.dsi.uniroma1.it

2. Digitate una richiesta GET:

```
GET /eng/maselli/ HTTP/1.1
Host: reti.dsi.uniroma1.it
```

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!

```
> telnet reti.dsi.uniroma1.it 80
Trying 151.100.17.39...
Connected to ccalcolo.di.uniroma1.it.
Escape character is '^]'.
GET /eng/maselli/ HTTP/1.1
Host: reti.dsi.uniroma1.it

HTTP/1.1 200 OK
Date: Tue, 28 Feb 2017 10:20:24 GMT
Server: Apache
Last-Modified: Wed, 19 Oct 2016 07:40:28 GMT
ETag: "5e8400-134b-ec3eb300"
Accept-Ranges: bytes
Content-Length: 4939
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
        ...
```

COOKIE

Mancanza di stato (stateless protocol)

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
  - Offrire contenuto personalizzato in base alle preferenze dell'utente
  - 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 (esempio in caso di NAT)
- Soluzione: Cookie (RFC 6265)
  - Consentono ai siti di tener traccia degli utenti

Cookie: meccanismo per mantenere lo stato

- Il meccanismo dei Cookie rappresenta un modo per creare una sessione di richieste e risposte HTTP "con stato" (stateful)
- La sessione rappresenta un contesto più largo rispetto alla richiesta/risposta.
- Questo contesto o sessione può essere utilizzato per creare per esempio "shopping cart", in cui le selezioni dell'utente possono essere aggregate, o un giornale online può presentare contenuti personalizzati in base alle letture precedenti dell'utente
- Per informazioni dettagliate su cookie e sessioni guardare RFC 2109 obsoleted by RFC 2965, obsoleted by RFC 6265

Cookie e Sessione

- Ci possono essere diversi tipi di sessione in base al tipo di informazioni scambiate e la natura del sito.
- Caratteristiche generali di una sessione:

1. Ogni sessione ha un inizio e una fine.
2. Ogni sessione ha un tempo di vita relativamente corto.
3. Sia il client che il server possono chiudere la sessione.
4. La sessione è **implicita** nello scambio di informazioni di stato.

Sessione vs. connessione

N.B. Per "sessione" NON si intende connessione persistente, ma una sessione logica creata da richieste e risposte HTTP. Una sessione può essere creata su connessioni persistenti e non persistenti.

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 server

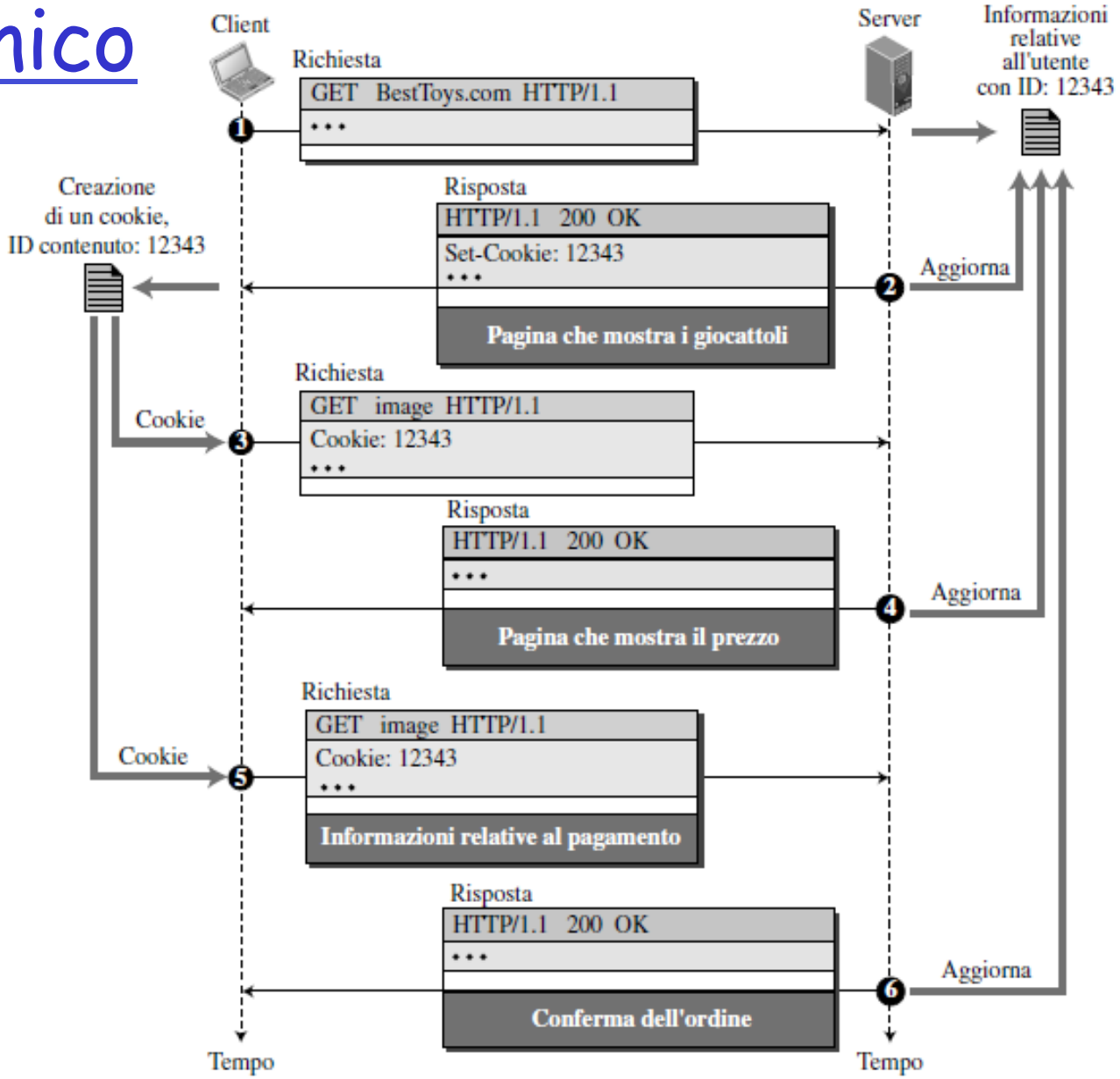
Esempio:

- L'utente A accede sempre a Internet dallo stesso PC

(non necessariamente con lo stesso IP)

- 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
- L'utente A invierà ogni futura richiesta inserendo l'ID nella richiesta

Esempio di utilizzo cookie in un negozio elettronico



Cookie

- Il server mantiene tutte le informazioni riguardanti il client su un file e gli assegna un identificatore (cookie) che viene fornito al client
- Il cookie inviato al client è un **identificatore di sessione (SID)**
- Per evitare che il cookie sia utilizzato da utenti "maligni" l'identificatore è composto da una stringa di numeri
- Esempio

```
== Server -> User Agent ==
Set-Cookie: SID=31d4d96e407aad42
```

- Il client ogni volta che manda una richiesta al server fornisce il suo identificatore (cookie): il browser consulta il file cookie, estrae il numero di cookie per il sito che si vuole visitare e lo inserisce nella richiesta http
- Esempio

```
== User Agent -> Server ==
Cookie: SID=31d4d96e407aad42
```

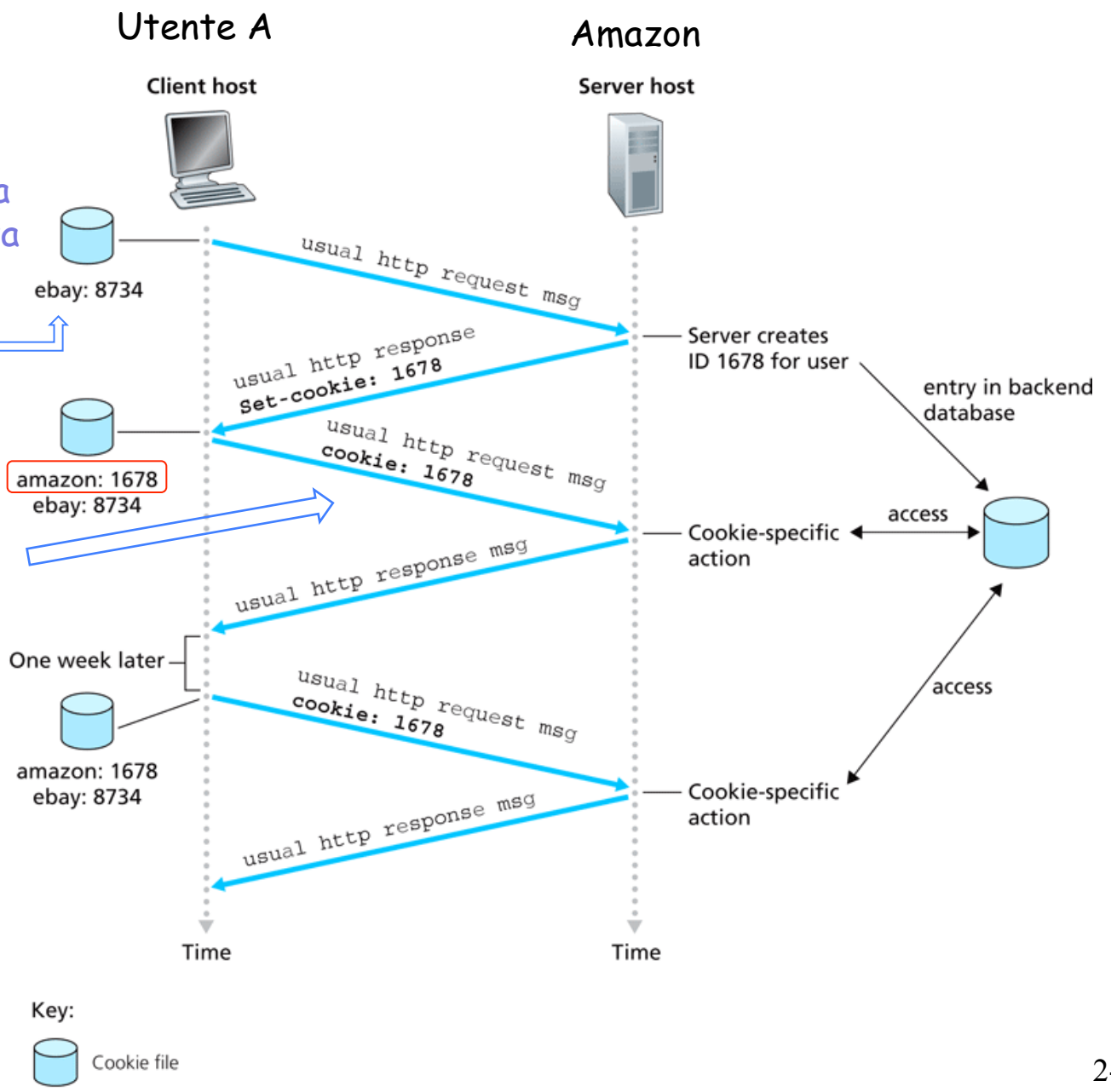
- Il server mediante il cookie fornito dal client accede al relativo file e fornisce risposte personalizzate

Cookie

Utente A usa sempre lo stesso browser, ha già visitato ebay, e ora visita Amazon per la prima volta

File cookie gestito dal browser

Nella successiva richiesta verso Amazon viene inserito il numero di cookie



Durata di un cookie

- Il server chiude una sessione inviando al client una intestazione `Set-Cookie` nel messaggio con

  `Max-Age=0`

- `Max-Age=delta-seconds`

  L'attributo `Max-Age` definisce il tempo di vita in secondi di un cookie. Dopo delta secondi il client dovrebbe rimuovere il cookie. Il valore zero indica che il cookie deve essere rimosso subito.

Cookie (continua)

Cosa possono contenere i file cookie:

- autorizzazione
- carta per acquisti
- Preferenze dell'utente
- 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

Altra soluzione per mantenere lo stato

Per mantenere lo stato e quindi creare una sessione

- Il client mantiene tutte le informazioni sullo stato della sessione e le inserisce in ogni richiesta inviata al server
  - Metodo POST
  - Inserendole nella URL

Vantaggi

- facile da implementare
- non richiede l'introduzione di particolare funzionalità sul server

Svantaggi

- può generare lo scambio di grandi quantità di dati
- le risorse del server devono essere re-inizializzate ad ogni richiesta

Web caching

Caching

Obiettivo: migliorare le prestazioni dell'applicazione web

- 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

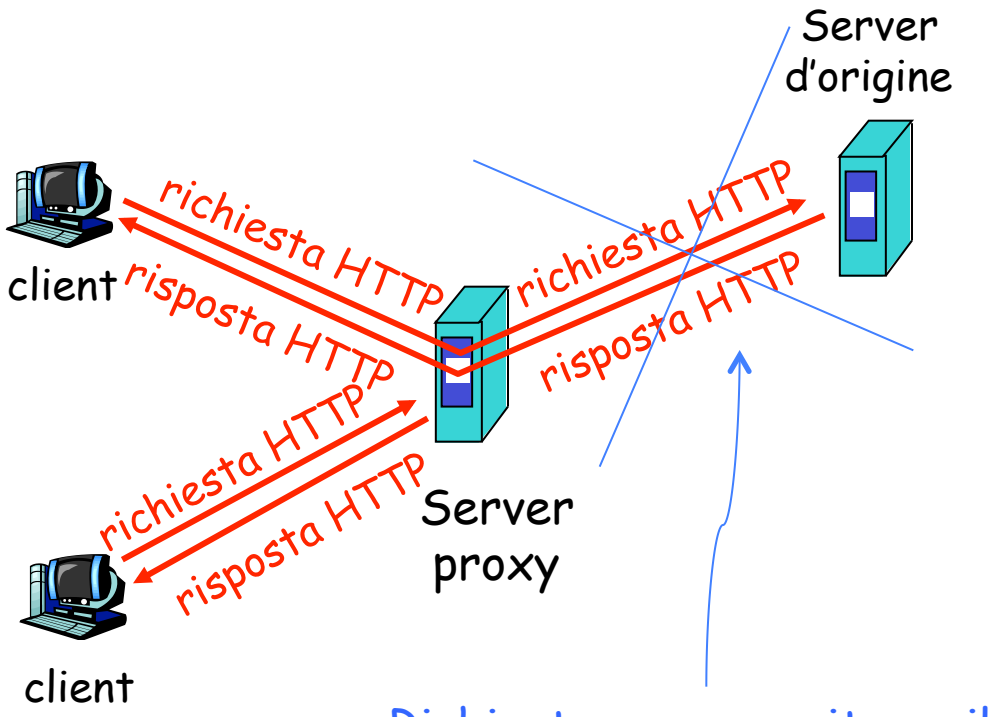
Browser Caching

- 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

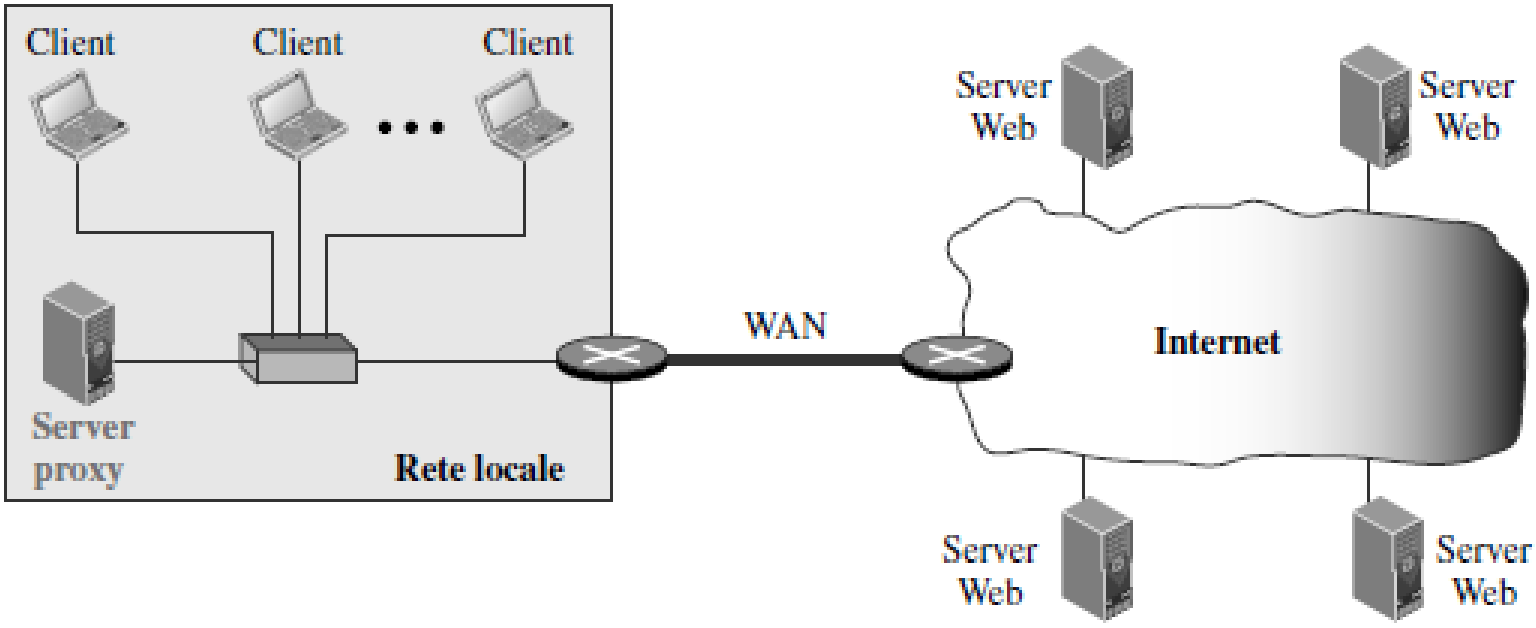
Web Caching: server proxy

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

- Il server proxy ha una memoria per mantenere copie della pagine visitate
- Il browser può essere configurato per inviare le richieste dell'utente alla cache
- Il browser trasmette tutte le richieste HTTP alla cache
  - oggetto presente nella cache: la cache fornisce l'oggetto
  - altrimenti la cache richiede l'oggetto al server d'origine e poi lo inoltra al client



Il server proxy in una LAN



Anche gli ISP possono mantenere un proxy per le richieste dei vari utenti

Cache web (continua)

- La cache opera come client e come server
- Tipicamente la cache è installata da un ISP (università, aziende o ISP residenziali)

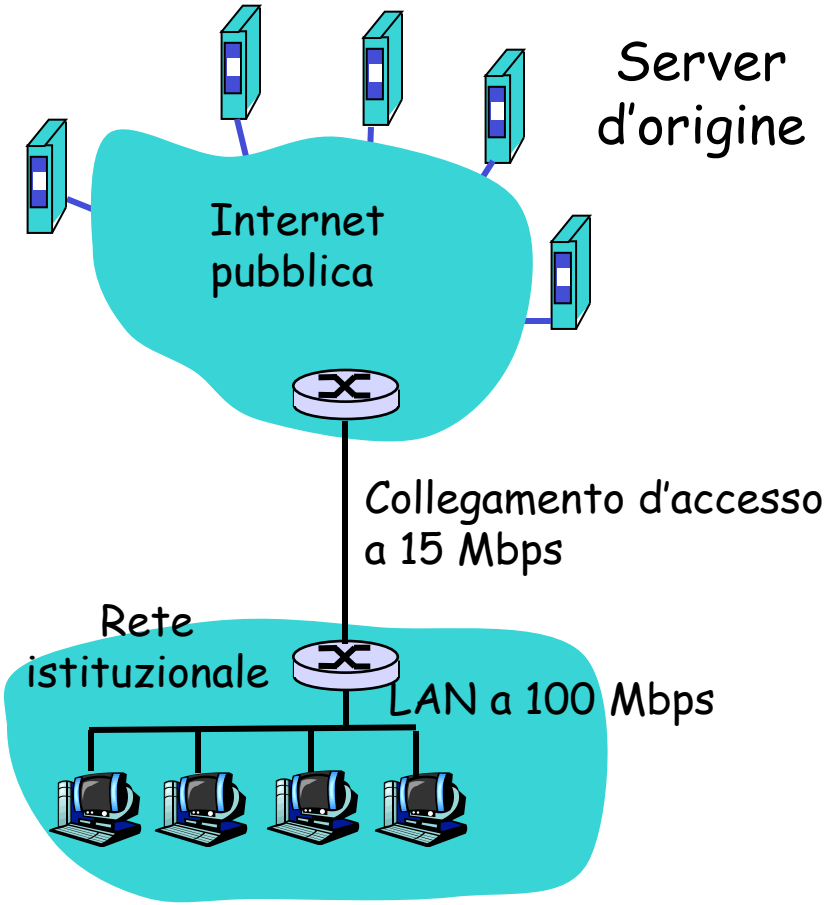
Perché il caching web?

- 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 meno efficienti di fornire dati con efficacia

Esempio in assenza di cache

Ipotesi

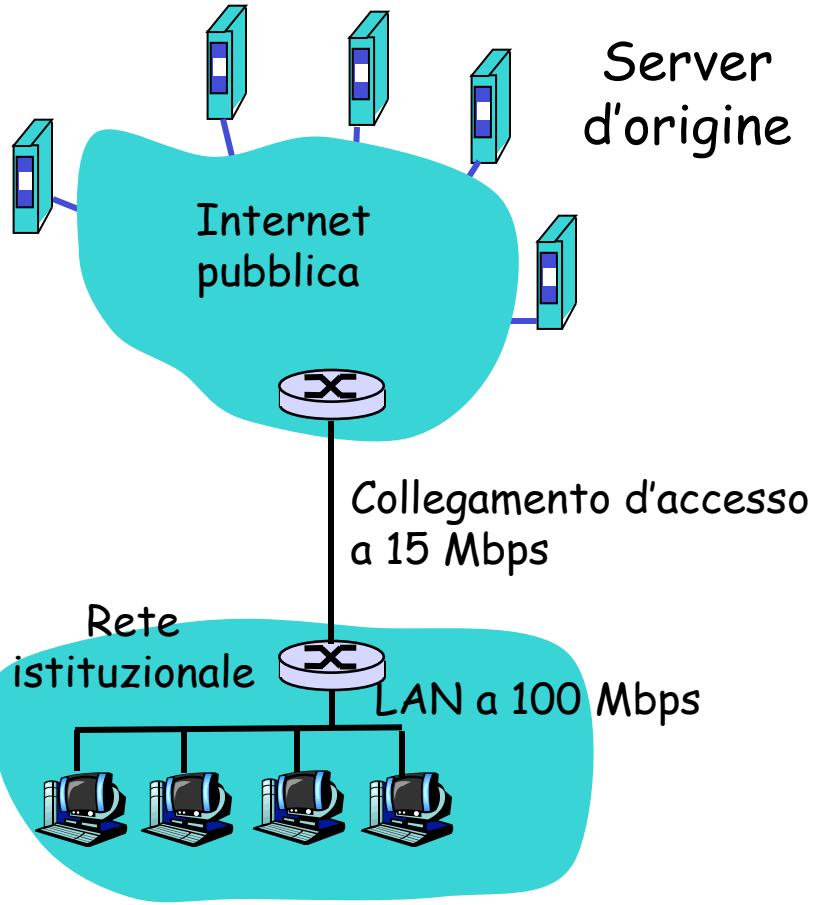
- Dimensione media di un oggetto = 1Mbits
- Frequenza media di richieste dai browser istituzionali ai server d'origine = 15 richieste al secondo
- Ritardo per recuperare un oggetto sulla rete internet (Internet delay): 2 sec
- Il tempo totale di risposta (total delay) =LAN delay + access delay+ Internet delay



Esempio in assenza di cache

Stima del tempo di risposta

- Valutiamo l'intensità di traffico La/R
- utilizzo sulla LAN =
  (15req/s*1Mb/req)/100Mbps = 15%
  (corrisponde a un ritardo nell'ordine delle decine di ms)
- utilizzo sul collegamento d'accesso = (15req/s*1Mb)/15Mbps = 100%
  (ritardo nell'ordine dei minuti)
- Intensità -> 1: il ritardo si fa consistente
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
  = 2 sec + minuti + millisecondi



Esempio in assenza di cache (continua)

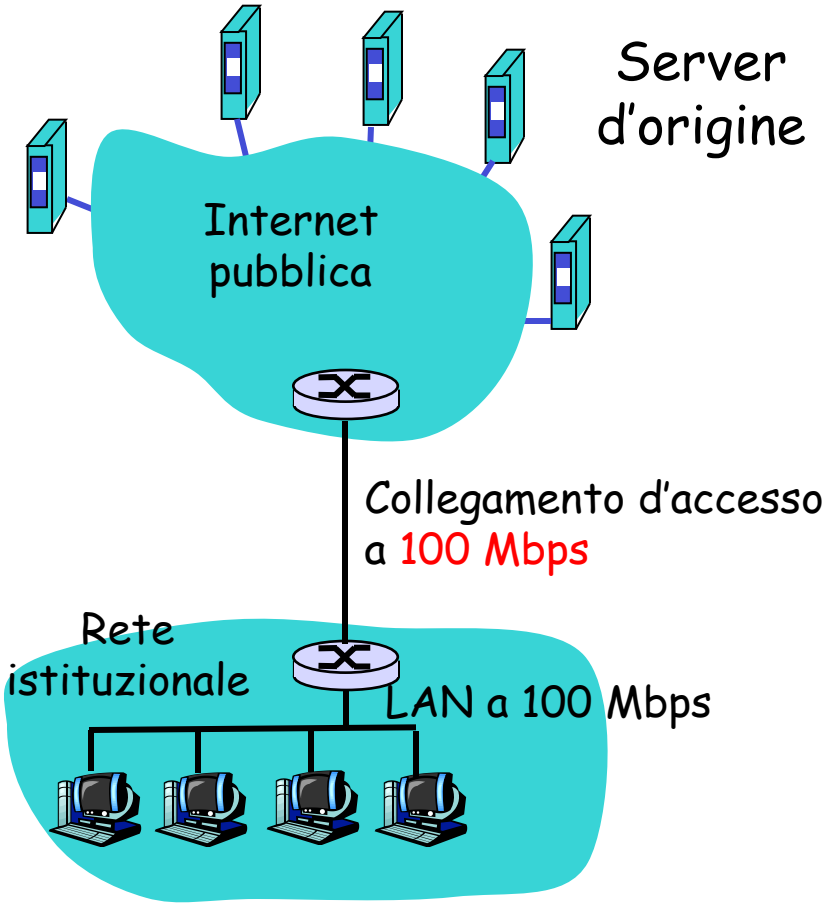
Soluzione possibile

- 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 costoso aggiornare il collegamento



Esempio in presenza di cache

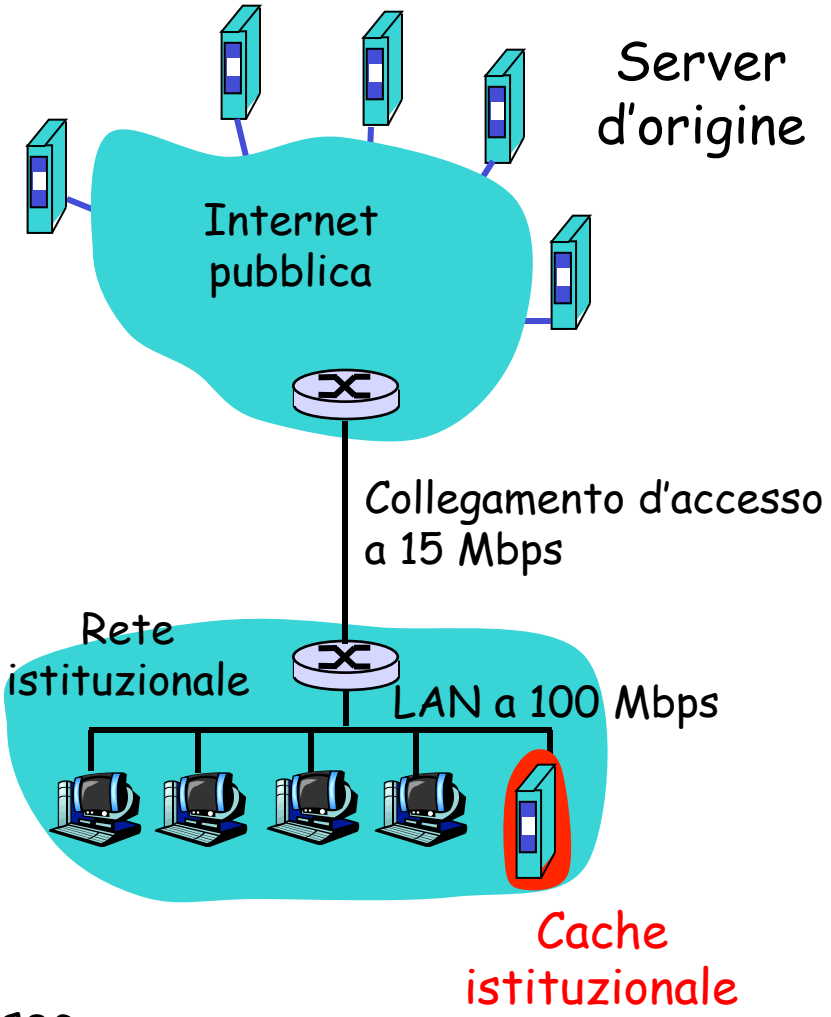
Soluzione possibile: installare la cache

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

Conseguenze

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

0,6*(2,01) sec + 0.4*(0.01) sec ~ 1,2 sec



Inserimento di un oggetto in cache

- Il **Client** invia messaggio di richiesta HTTP alla cache

```
GET /page/figure.gif
Host: www.sito.com
```

- La cache non ha l'oggetto
- La cache invia richiesta HTTP al **Server**
- Il server invia risposta HTTP alla cache

```
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
- La cache invia la risposta al client

Validazione dell'oggetto

- Il **Client** invia messaggio di richiesta HTTP alla cache

```
GET /page/figure.gif
Host: www.sito.com
```

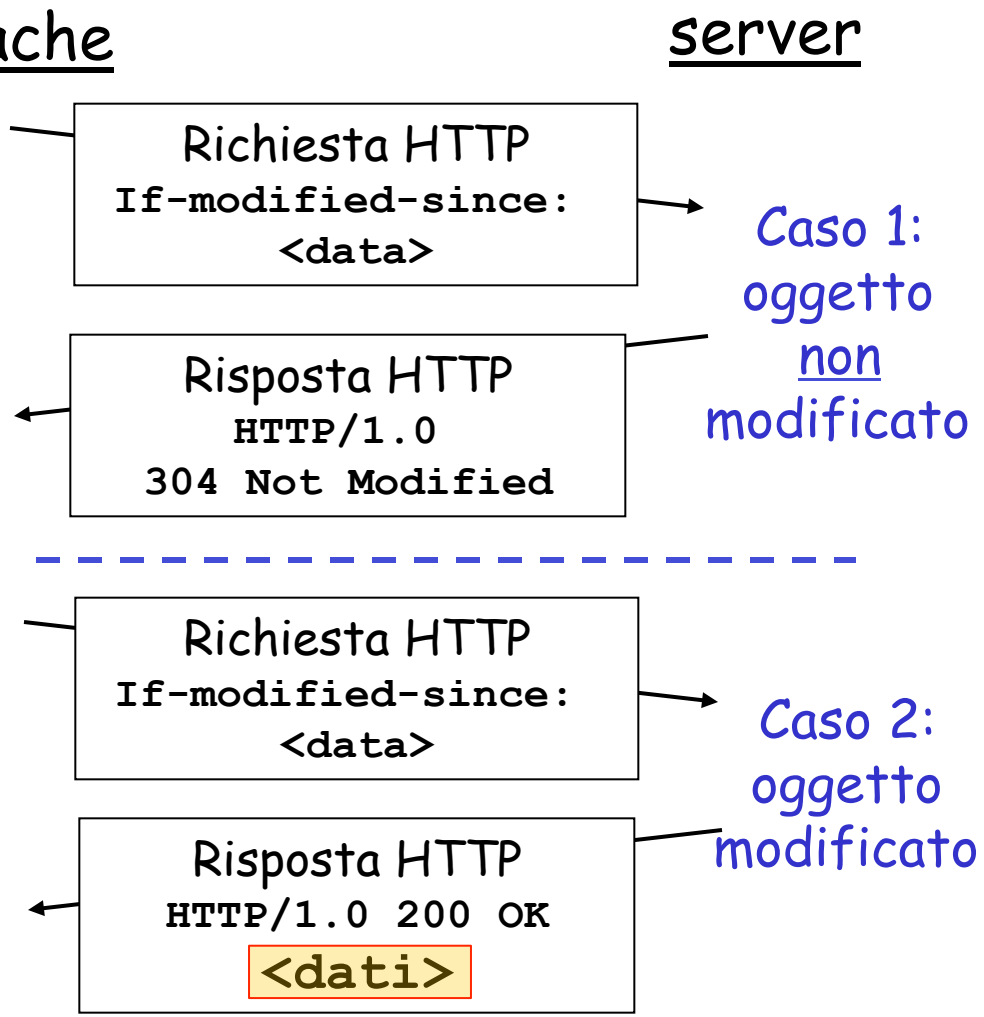
- La cache ha l'oggetto
- La cache prima di inviare l'oggetto deve verificare che non sia **scaduto** (**modificato sul server di origine**)
- La cache esegue una richiesta verso il Web server che mantiene l'oggetto, per verificarne la validità mediante il metodo

GET condizionale

  - Usa metodo GET
  - Include una riga di intestazione If-Modified-Since

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`



Riassunto

- Protocollo HTTP
  - Client-server
  - Connessioni persistenti e non
  - Formato messaggi
- Cookie
  - Mantenimento dello stato
- Web cache (server proxy)
  - GET condizionale