Università degli Studi di Roma "Tor Vergata"
Laurea in Informatica

Sistemi Operativi e Reti
(modulo Reti)
a.a. 2023/2024

Livello di applicazione (parte2)

dr. Manuel Fiorelli

manuel.fiorelli@uniroma2.it

https://art.uniroma2.it/fiorelli

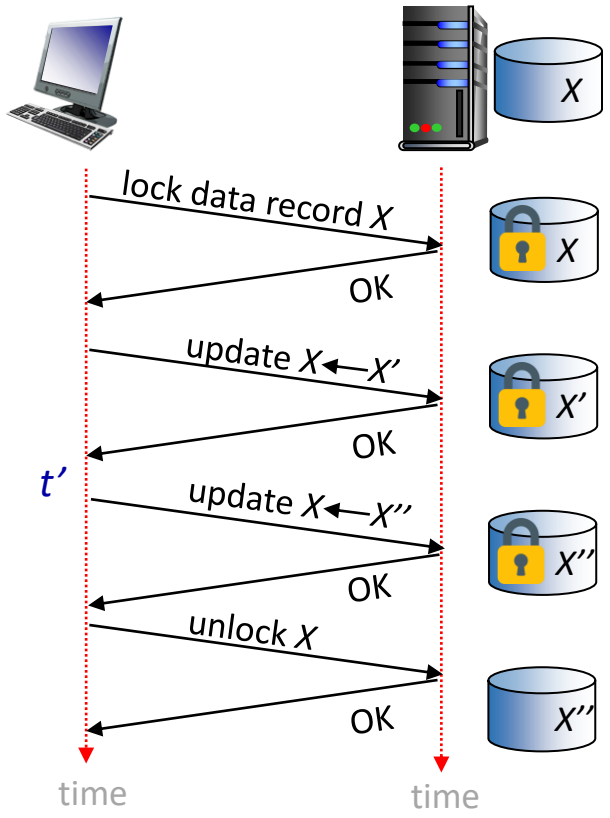
Basate sulle slide del libro di testo:
https://gaia.cs.umass.edu/kurose_ross/ppt.php

Mantenere stato utente/server: i cookie

Ricordate: l'interazione HTTP GET/risposta è *senza stato (stateless)*

- nessuna nozione di scambio di messaggi HTTP in più fasi per completare una "transazione" Web
  - non è necessario che il client o il server tengano traccia dello "stato" dello scambio in più fasi
  - tutte le richieste HTTP sono indipendenti l'una dall'altra
  - non è necessario che il client né il server siano in grado di “recuperare” da una transazione quasi completa ma mai completata

un protocollo con stato: il client fa due modifiche a X, o nessuna



Q: che succede se la connessione di rete o il client si blocca al tempo *t'* ?

Mantenere stato utente/server: i cookie

I siti web e il browser client usano i *cookie* per mantener dello stato tra le transazioni

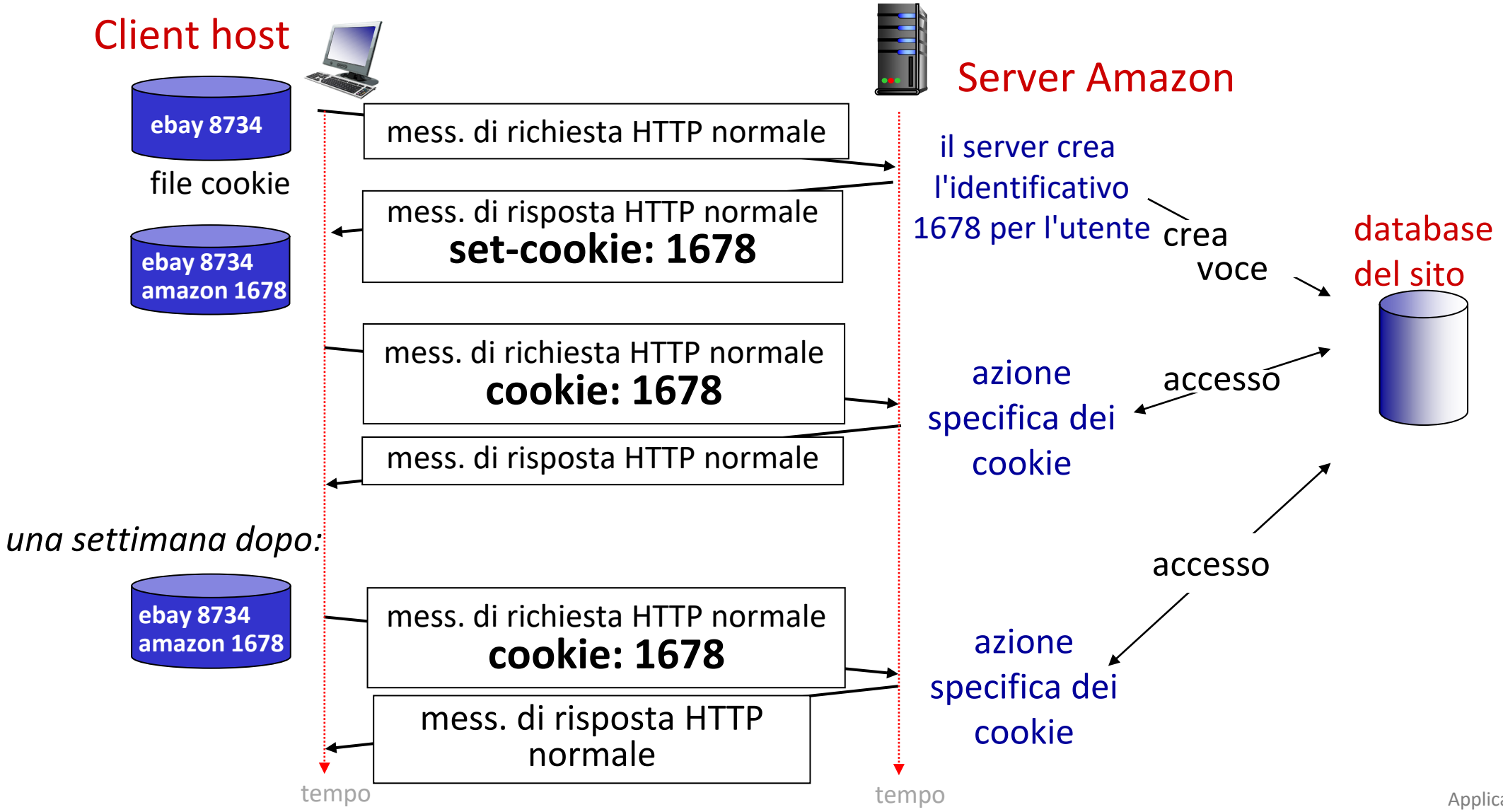
quattro componenti:

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 usa il browser dal portatile, visita uno specifico siti di commercio elettronico per la prima volta
- quando la richiesta HTTP iniziale arriva al sito, il sito crea:
  - un identificativo unico
  - una voce nel proprio database, indicizzata dal numero identificativo
- Il server ritorna una risposta che include l'intestazione Set-cookie, che contiene l'identificativo unico e che sarà aggiunto al file dei cookie
- le successive richieste del browser di Susan per questo sito conterranno l'identificativo in una intestazione cookie

Mantenere stato utente/server: i cookie



Cookie HTTP: commenti

I cookie possono essere usati per:

- autorizzazione
- carello degli acquisti
- raccomandazioni
- stato della sessione dell'utente (e-mail Web)

Sfida: Come mantenere lo stato?

- *presso gli endpoint del protocollo:* mantenere lo stato presso il trasmettitore e il ricevitore attraverso multiple transazioni
- *nei messaggi:* i cookie trasportano lo stato nei messaggi HTTP

nota

cookie e privacy:

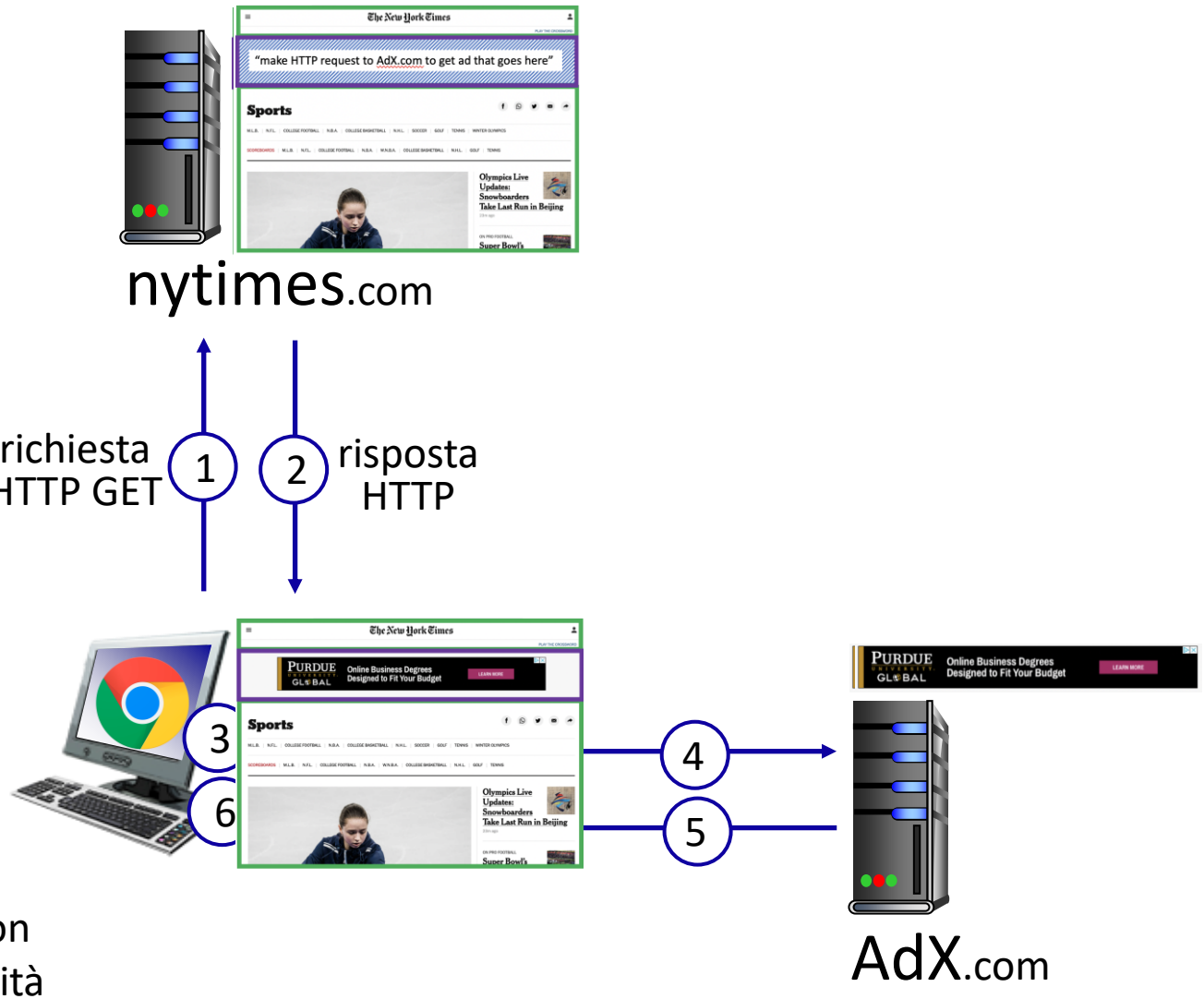
- I cookie consentono ai siti di *imparare* molto di voi
- cookie persistenti di terze parti (cookie di tracciamento, *tracking cookies*) consentono il tracciamento di una identità comune (valore del cookie) attraverso siti web multipli

Esempio: visualizzare una pagina web del NY Times

1 2 invia una GET per il file HTML di base da nytimes.com

4 5 recupera l'annuncio da AdX.com

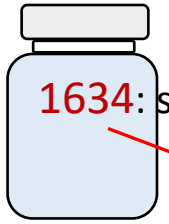
7 mostra la pagina composta



pagina del NY con mostrate pubblicità incorporate

Cookie: tracciare il comportamento di navigazione di un utente

nytimes.com (sports)



1634: sports, 2/15/22

HTTP GET

risposta HTTP
Set cookie: 1634

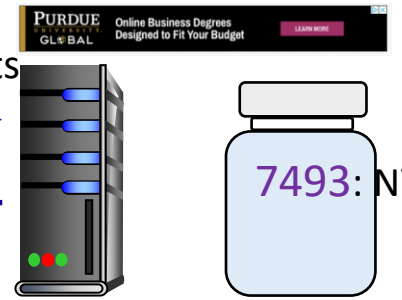
Cookie di "prima parte" ("first party") – da siti che hai scelto di visitare (fornisce il file HTML di base)

HTTP GET
Referrer: NY Times Sports

4

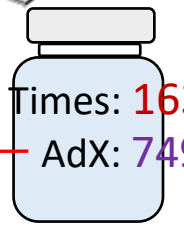
5

HTTP reply
Set cookie: 7493



7493: NY Times sports, 2/15/22

AdX.com

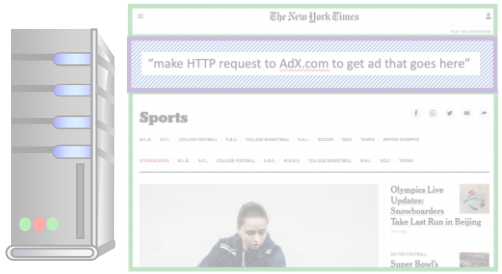


NY Times: 1634
AdX: 7493

cookie di "terze parti" ("third party") cookie – da siti web che <u>non</u> hai scelto di visitare

Cookie: tracciare il comportamento di navigazione

socks.com



nytimes.com

HTTP reply 2

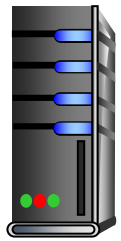
HTTP GET 1

HTTP GET
Referrer: socks.com, cookie: 7493

4

5

HTTP reply
Set cookie: 7493



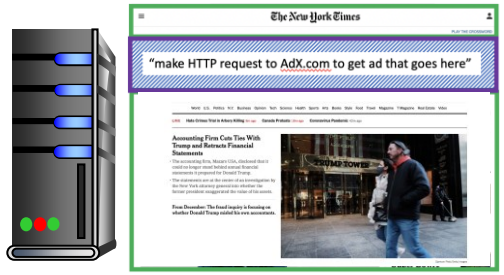
AdX.com

AdX:

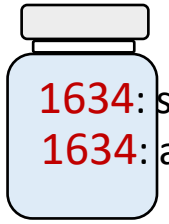
- *traccia la mia navigazione* su siti con annunci AdX
- può restituire annunci mirati in base alla cronologia di navigazione

Cookie: tracciare il comportamento di navigazione di un utente

(un giorno dopo)



nytimes.com (arts)



HTTP GET
cookie: 1634

HTTP reply
Set cookie: 1634

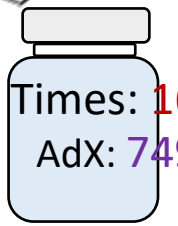
HTTP GET
Referrer: nytimes.com, cookie: 7493

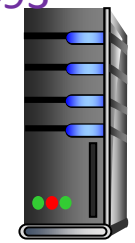
4

5

HTTP reply
Set cookie: 7493

Restituito annuncio per I calzini!





AdX.com

Cookie: tracciare il comportamento di navigazione di un utente

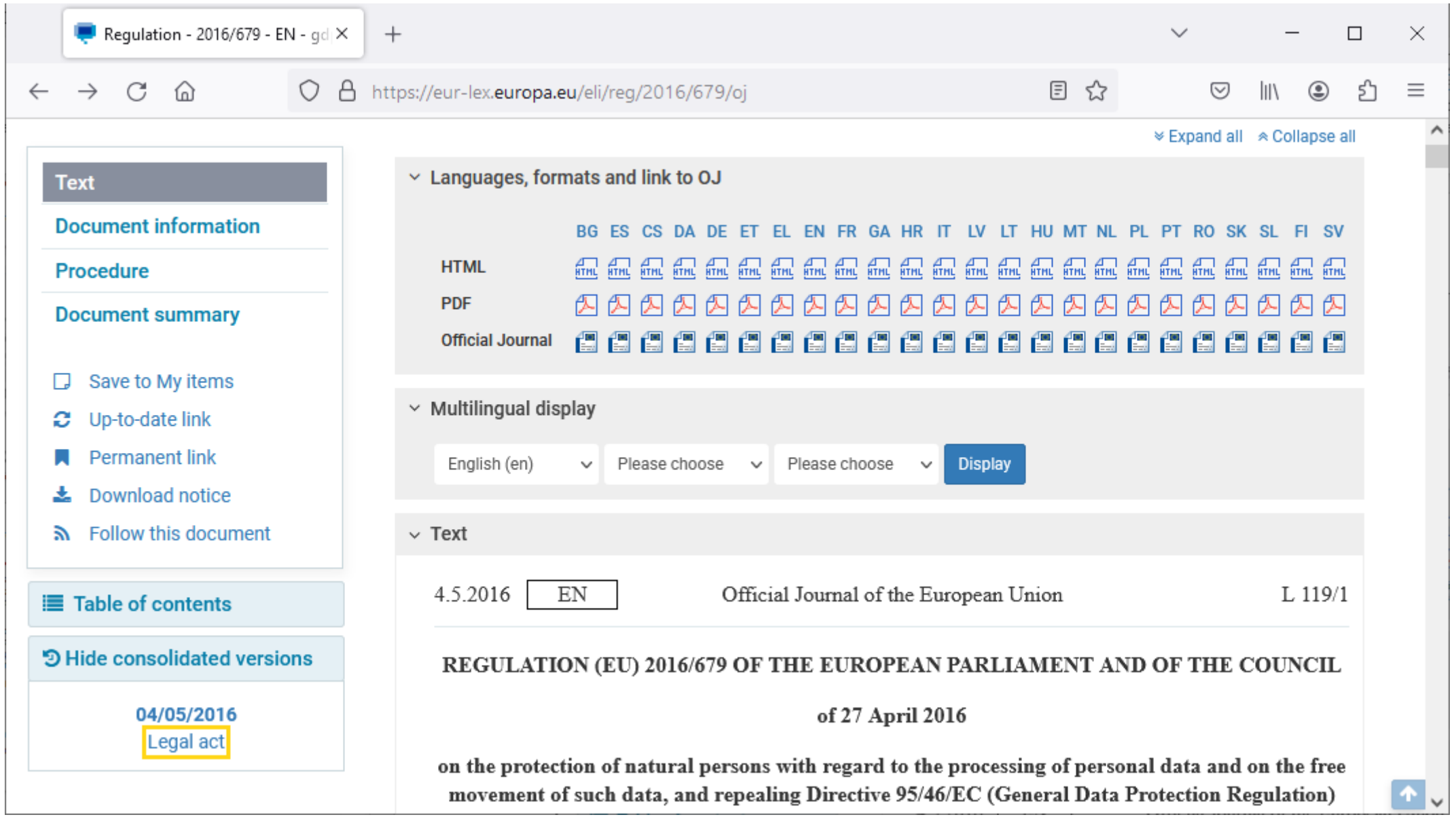
I cookie posso essere usati per:

- tracciare (*track*) il comportamento degli utenti su un dato sito (cookie di prima parte)
- tracciare il comportamento degli utenti su più siti (cookie di terze parti) senza neppure che l'utente abbia mai scelto di visitare il sito del tracker (!)
- il tracciamento può essere *invisibile* all'utente:
  - piuttosto che un annuncio visualizzato che attiva HTTP GET al tracker, potrebbe essere un collegamento invisibile

tracciamento di terze parti tramite cookie:

- disabilitato per impostazione predefinita nei browser Firefox e Safari
- eliminazione graduale dei cookie di terze parti nel browser Chrome, inizialmente bloccati per l'1% degli utenti a partire da Gennaio 2024, con l'obiettivo di estendere il blocco a tutti nel terzo trimestre del 2024

GDPR (EU General Data Protection Regulation)



GDPR (EU General Data Protection Regulation) e i cookie

"Le persone fisiche possono essere associate a identificativi online [..], quali gli indirizzi IP, a marcatori temporanei (cookies) o a identificativi di altro tipo, [...].

Tali identificativi possono lasciare tracce che, in particolare se combinate con identificativi univoci e altre informazioni ricevute dai server, possono essere utilizzate per creare profili delle persone fisiche e identificarle.."

GDPR, recital 30 (May 2018)

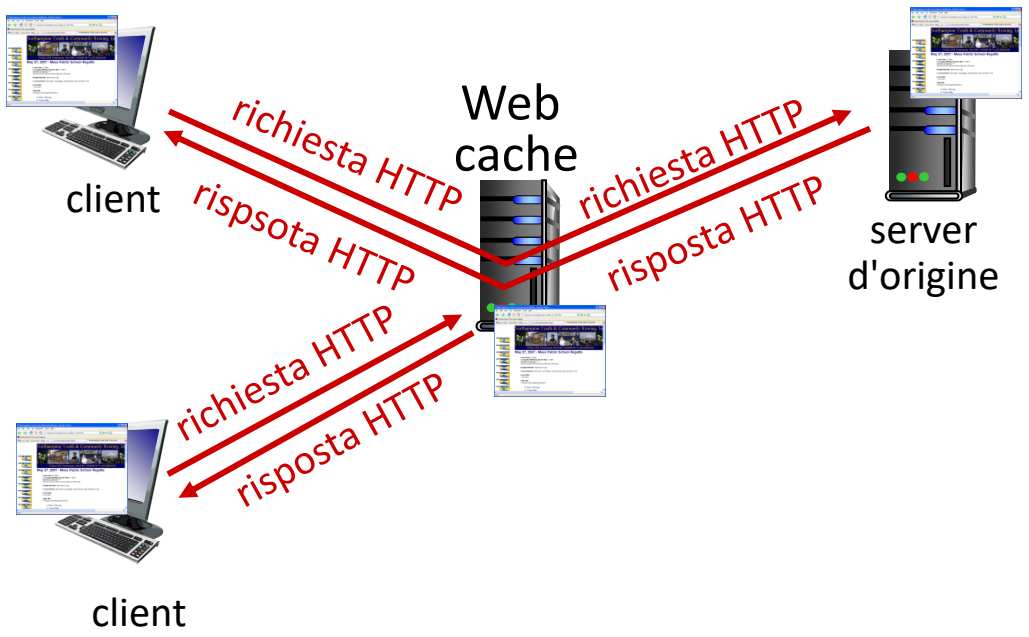
quando i cookie possono identificare un individuo, i cookie sono considerati dati personali, soggetti alla normativa GDPR sui dati personali

L'utente ha un controllo esplicito sull'autorizzazione o meno dei cookie.

Web cache

Obiettivo: soddisfare la richiesta del client senza coinvolgere il server d'origine (*origin server*)

- l'utente configura il browser per usare una *Web cache* (locale)
- Il browser trasmette tutte le richieste HTTP alla cache
  - *se* l'oggetto è nella cache: la cache fornisce l'oggetto al client
  - *altrimenti* la cache richiede l'oggetto al server d'origine, memorizza ("cache") l'oggetto ricevuto, e infine lo restituisce al client



Web cache (server proxy)

- la cache opera come client (per il server d'origine) e come server (per il client originale)
- Il server comunica alla cache la cache consentita dell'oggetto nell'intestazione della risposta:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

Queste righe di intestazione possono essere usate anche nelle richieste con un significato analogo.

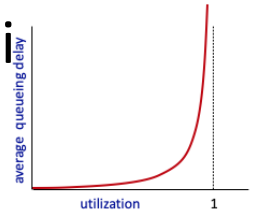
Perché il web caching?

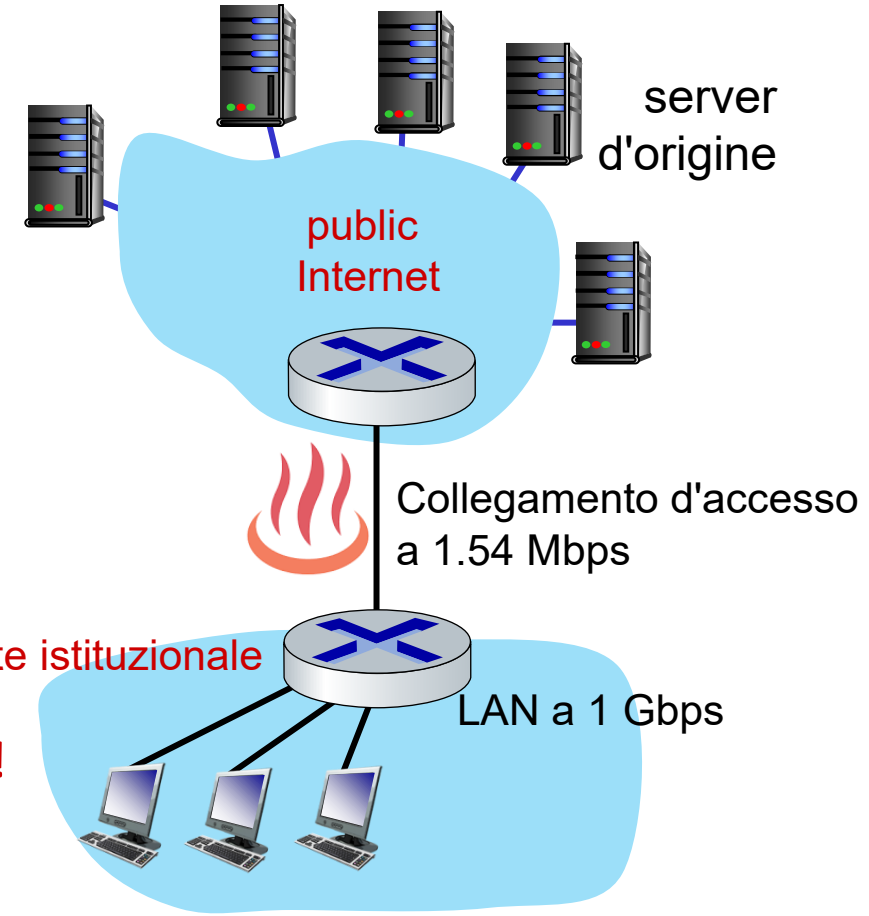
- riduce i tempi di risposta alle richieste dei client
  - la cache è più vicina ai client
- riduce il traffico sul collegamento di accesso a Internet istituzionale
- Internet è ricca di cache
  - consente ai provider "scadenti" di fornire dati con efficacia

Esempio di caching

Scenario:

- velocità collegamento d'accesso: 1.54 Mbps
- RTT dal router istituzionale al server: 2 s
- dimensione di un oggetto: 100K bits
- frequenza media di richieste dai browser istituzionali al server d'origine: 15/s
  - velocità media di trasmissione dei dati ai browser: 1.50 Mbps





Prestazioni:

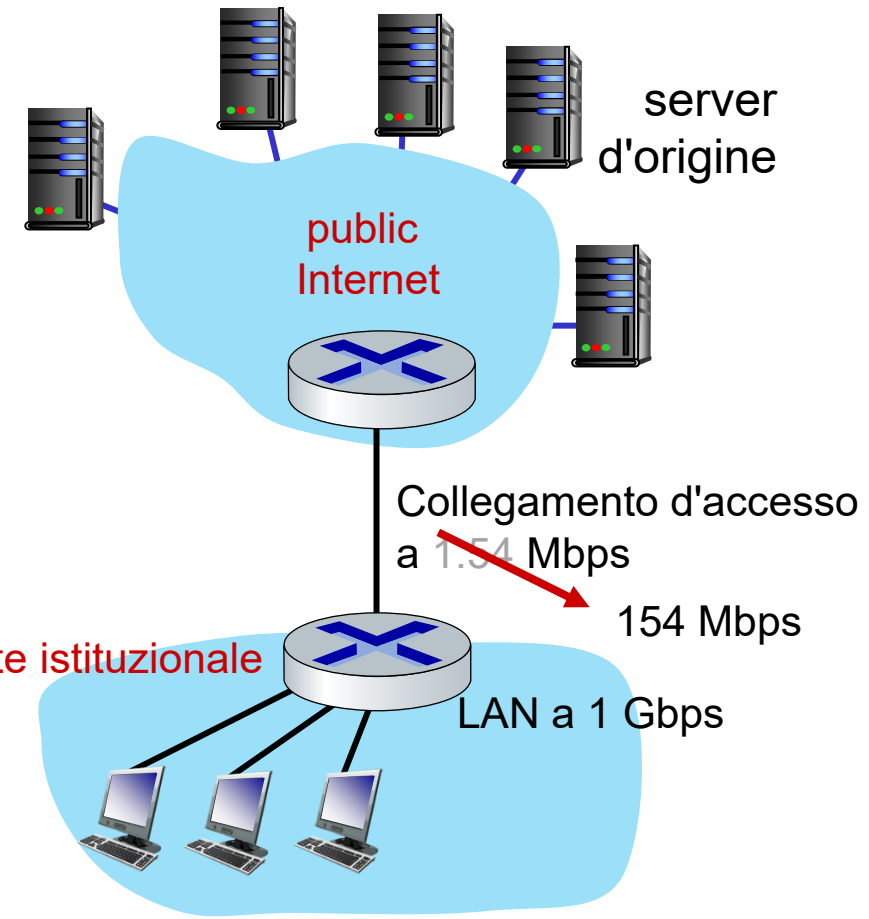
- utilizzazione del collegamento d'accesso = .97
- utilizzazione della LAN: .0015
- end-end delay = ritardo di Internet + ritardo del collegamento d'accesso + ritardo della LAN
  = 2 s + minuti + microsecondi

problema: ritardo d'accodamento elevato con elevata utilizzazione!

Opzione 1: collegamento d'accesso più veloce

Scenario:

- velocità collegamento d'accesso: 1.54 Mbps → 154 Mbps
- RTT dal router istituzionale al server: 2 s
- dimensione di un oggetto: 100K bits
- frequenza media di richieste dai browser istituzionali al server d'origine: 15/s
  - velocità media di trasmissione dei dati ai browser: 1.50 Mbps



Prestazioni:

- utilizzazione del collegamento d'accesso = .97 → .0097
- utilizzazione della LAN: .0015
- end-end delay = ritardo di Internet + ritardo del collegamento d'accesso + ritardo della LAN
  = 2 s + minuti + microsecondi → msecs

Costo: collegamento d'accesso più veloce (costoso!)

Opzione 2: installare un web cache

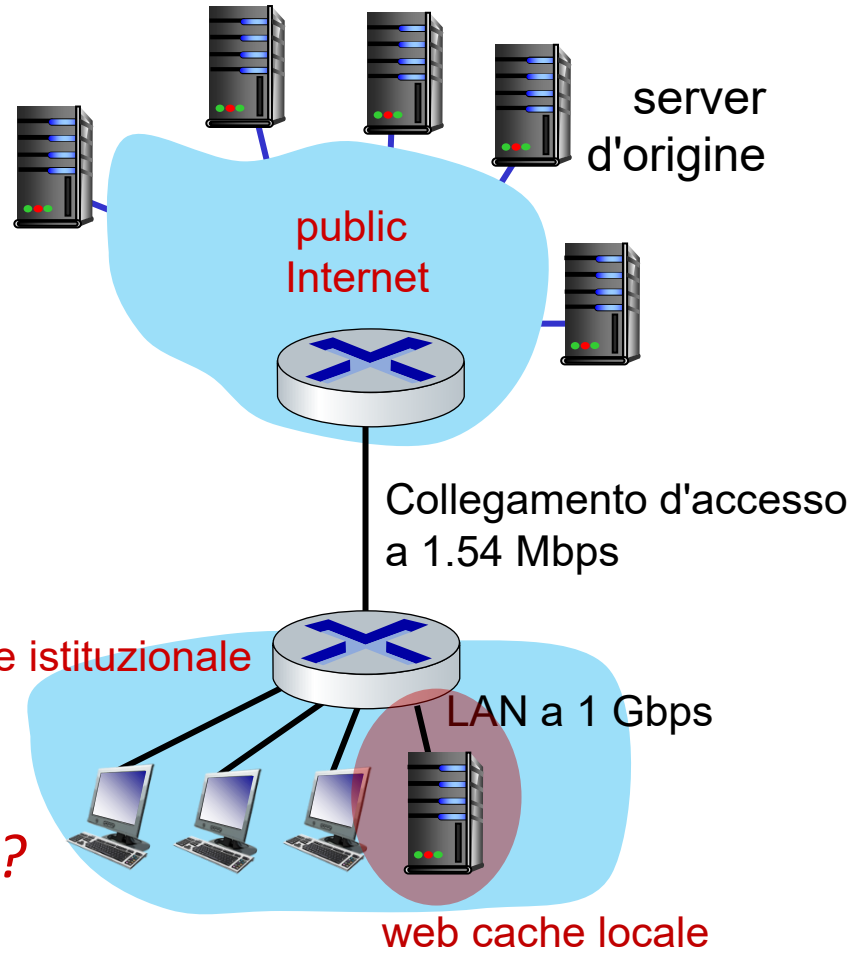
Scenario:

- velocità collegamento d'accesso: 1.54 Mbps
- RTT dal router istituzionale al server: 2 s
- dimensione di un oggetto: 100K bits
- frequenza media di richieste dai browser istituzionali al server d'origine: 15/s
  - velocità media di trasmissione dei dati ai browser: 1.50 Mbps

Costo: web cache (economica!)

Prestazioni:

- utilizzazione LAN: .?
- utilizzazione del link di accesso = ? *come calcolarli?*
- ritardo end-end medio = ?



Calcolo dell'utilizzo del collegamento di accesso e del ritardo end-end con la cache:

supponiamo una percentuale di successo (hit rate) pari a 0.4:

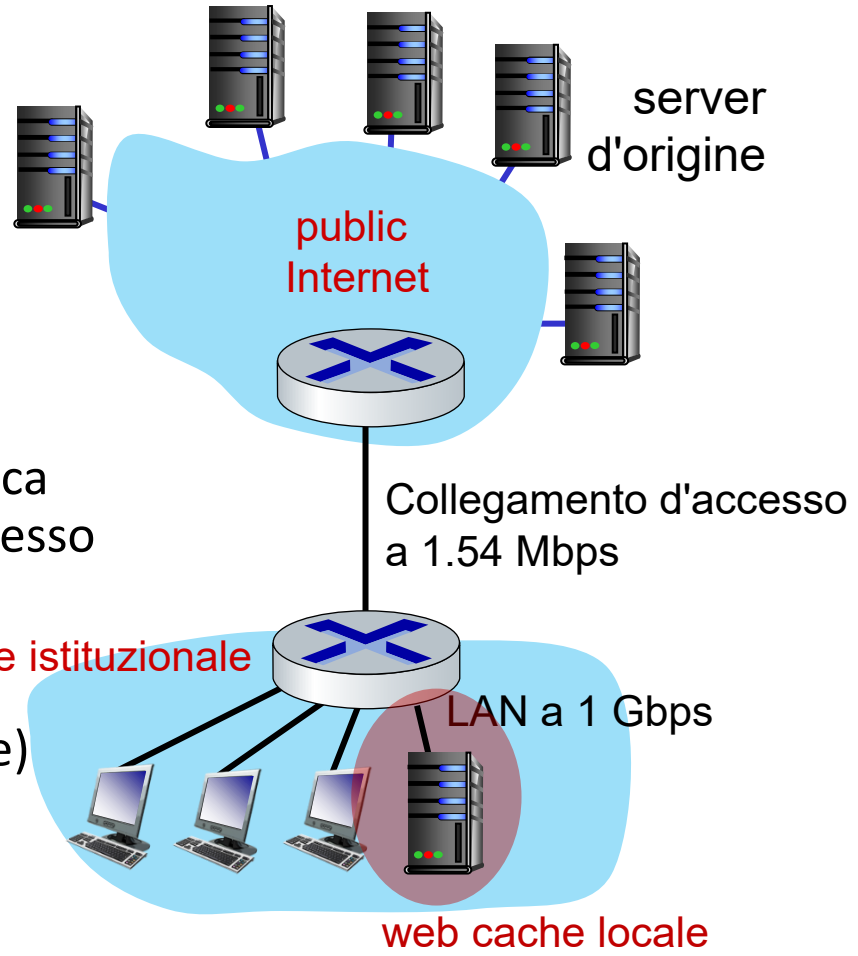
- il 40% delle richieste sarà soddisfatto dalla cache, con ritardo basso (msec)
- 60% delle richieste sarà soddisfatto dal server d'origine
  - tasso di trasmissione sul collegamento d'accesso

    = 0.6 * 1.50 Mbps = .9 Mbps
  - utilizzazione collegamento d'accesso = 0.9/1.54 = .58 significa basso (msec) ritardo di accodamento al collegamento d'accesso
- ritardo end-end medio:

  = 0.6 * (ritardo dai server d'origine)

  + 0.4 * (ritardo quando richiesta soddisfatta dalla cache)

  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs



ritardo medio end-end inferiore che con un collegamento a 154 Mbps (e meno costoso!)

GET condizionale

Obiettivo: non inviare un oggetto se la cache ha una copia aggiornata dell'oggetto

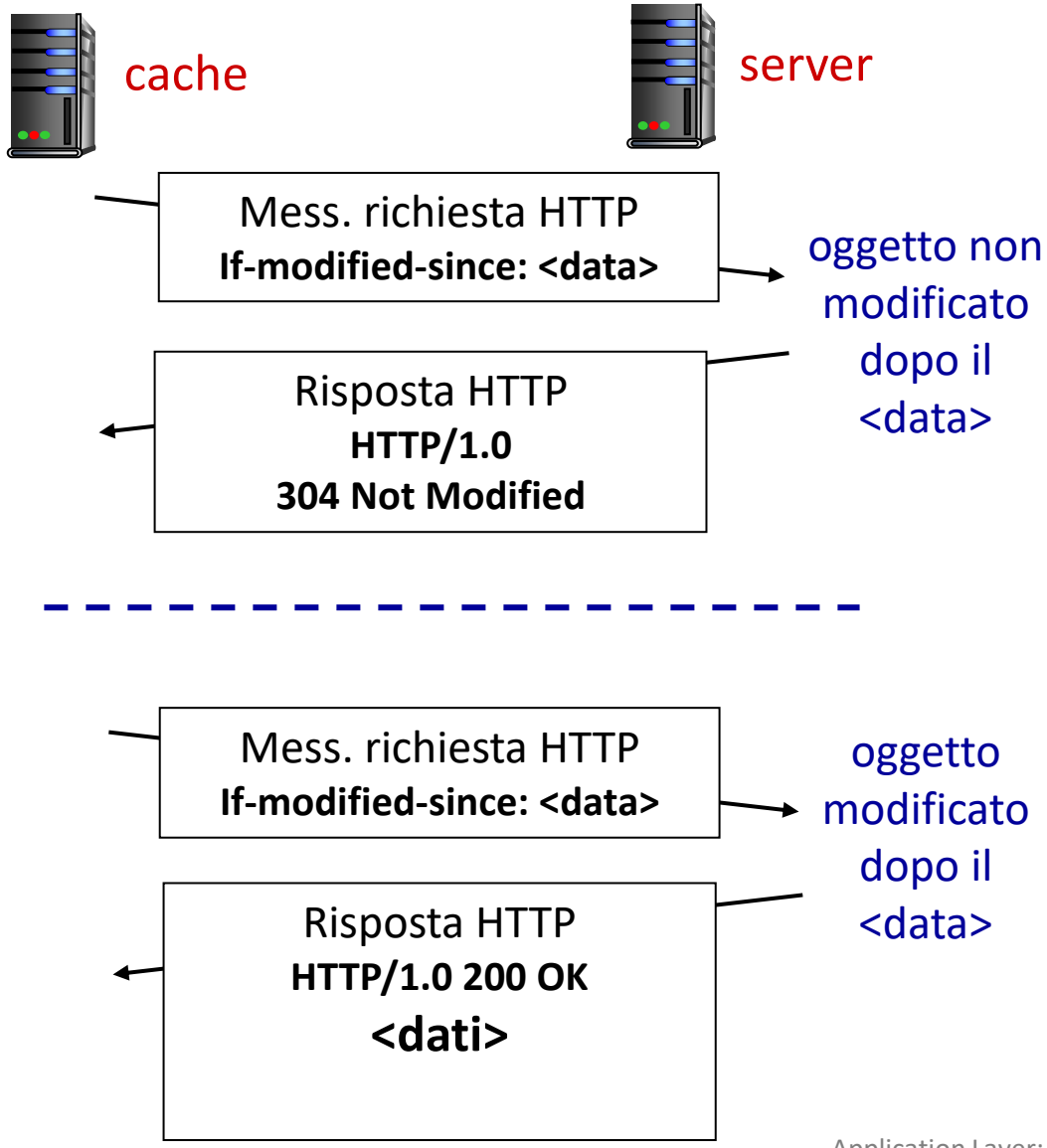
- Nessun ritardo di trasmissione dell'oggetto (o uso delle risorse di rete)

- *client:* 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



Nota sul caching

Il caching può essere effettuato da:

- una web cache, ossia uno speciale tipo di proxy, cui il browser invia le richieste invece che indirizzarle all'origin server.
- oppure, dal browser stesso, che conserva una copia degli oggetti richiesti in precedenza

In entrambi i casi, occorre prestare attenzione al problema dell'aggiornamento degli oggetti: vedi riga di intestazione *Cache-Control* e *GET condizionale*.

HTTP/2

Obiettivo principale: diminuzione del ritardo nelle richieste HTTP a più oggetti

HTTP1.1: ha introdotto GET multiple in pipeline su una singola connessione TCP

- il server risponde *in ordine* (FCFS: first-come-first-served scheduling) alle richieste GET
- con FCFS, oggetti piccoli possono dover aspettare per la trasmissione (head-of-line (HOL) blocking [blocco in testa alla coda]) dietro a uno o più oggetti grandi
- il recupero delle perdite (ritrasmissione dei segmenti TCP persi) blocca la trasmissione degli oggetti

HTTP/2

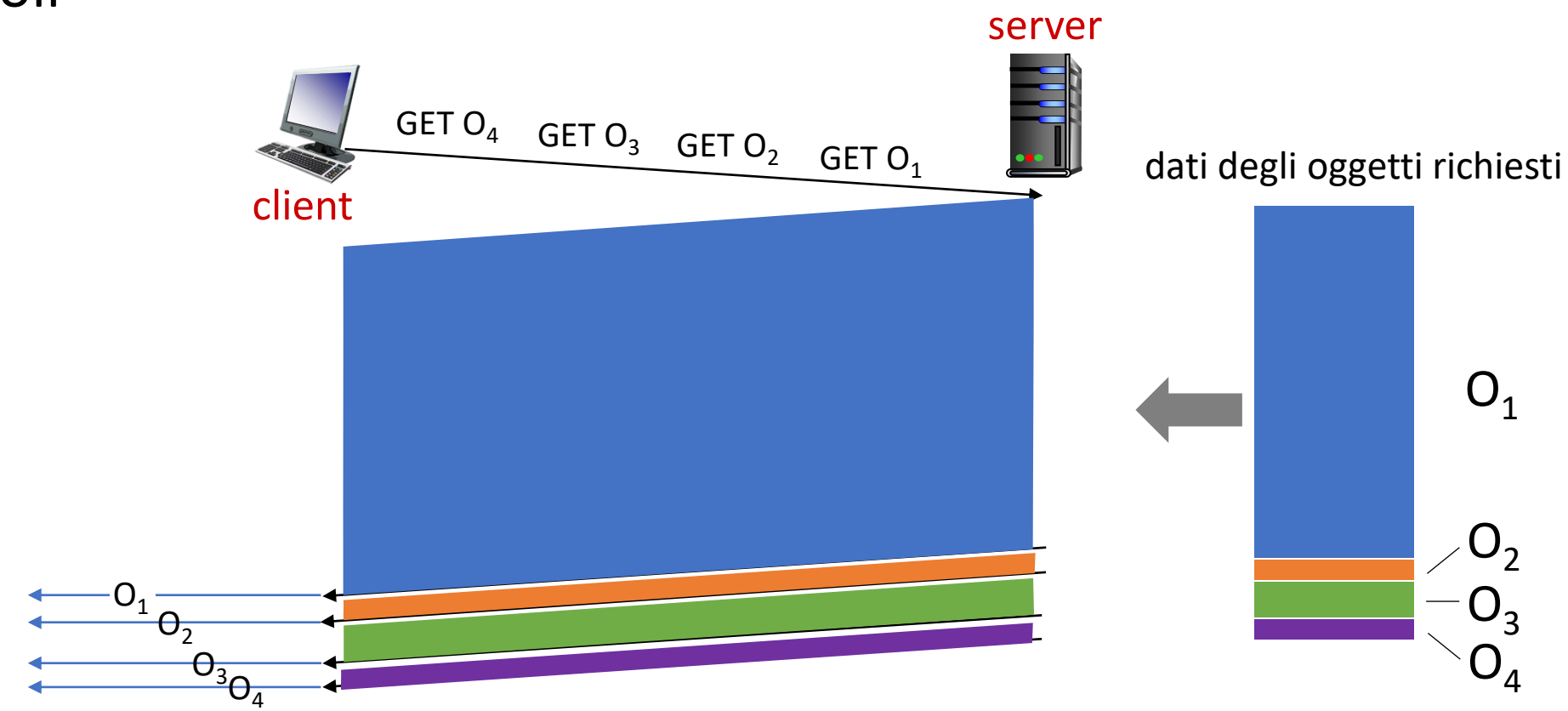
Obiettivo principale: diminuzione del ritardo nelle richieste HTTP a più oggetti

HTTP/2: [RFC 7540, 2015] maggiore flessibilità del server nell'invio di oggetti al client:

- metodi, codice di stato, maggior parte dei campi di intestazione inalterati rispetto a HTTP 1.1
- ordine di trasmissione degli oggetti richiesti basata su una priorità degli oggetti specificata dal client (non necessariamente FCFS)
- invio *push* al client di oggetti aggiuntivi, senza che il client li abbia richiesti
- dividere gli oggetti in frame, intervallare i frame per mitigare il blocco HOL

HTTP/2: mitigazione del blocco HOL

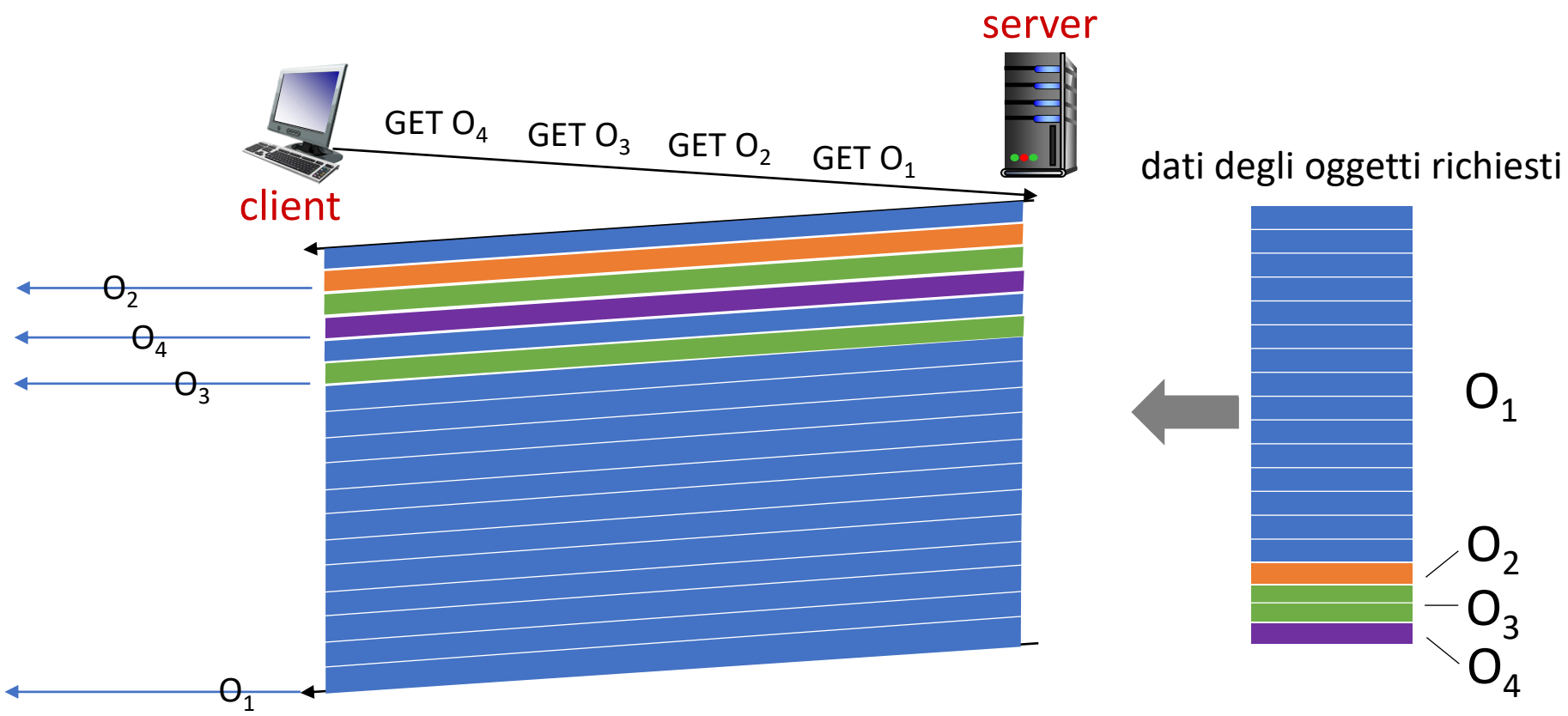
HTTP 1.1: il client richiede 1 oggetto grande (es., file video) e 3 oggetti più piccoli



oggetti consegnati nell'ordine in cui sono stati richiesti: O_2, O_3, O_4 aspettano dietro O_1

mitigazione

HTTP/2: oggetti divisi in frame, trasmissione de frame interlacciata



O_2, O_3, O_4 consegnati rapidamente, O_1 leggermente ritardato

Da HTTP/2 a HTTP/3

HTTP/2 su una singola connessione TCP significa:

- il recupero dalla perdita di pacchetti blocca comunque tutte le trasmissioni di oggetti
  - come in HTTP 1.1, i browser sono incentivati ad aprire più connessioni TCP parallele per ridurre lo stallo e aumentare il throughput complessivo
- nessuna sicurezza su una connessione TCP semplice
- HTTP/3: aggiunge sicurezza, controllo di errore e congestione per oggetto (più pipelining) su UDP
  - ulteriori informazioni su HTTP/3 trattando il livello di trasporto

Livello di applicazione: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- E-mail, SMTP, IMAP
- DNS: il servizio di directory di Internet
- Applicazioni P2P
- Streaming video e reti di distribuzione di contenti
- Programmazione delle socket programming con UDP e TCP

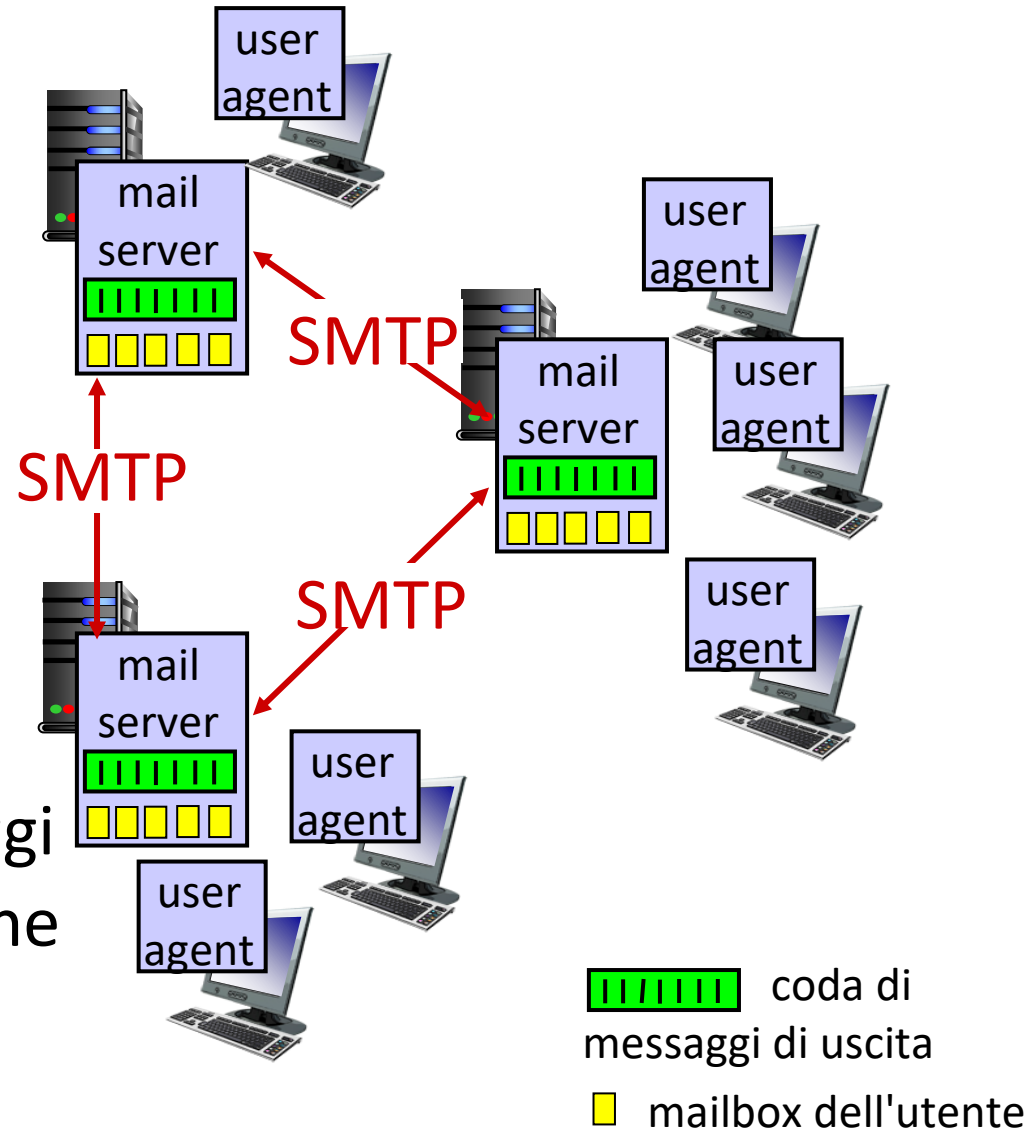
E-mail

Tre componenti principali:

- user agents (o *agenti utenti*)
- mail servers (o *server di posta*)
- simple mail transfer protocol: SMTP

User Agent

- detto anche “mail reader”
- composizione, editing, lettura dei messaggi
- esempi: Outlook, client di posta dell'iPhone
- i messaggi in uscita o in arrivo sono memorizzati sul server



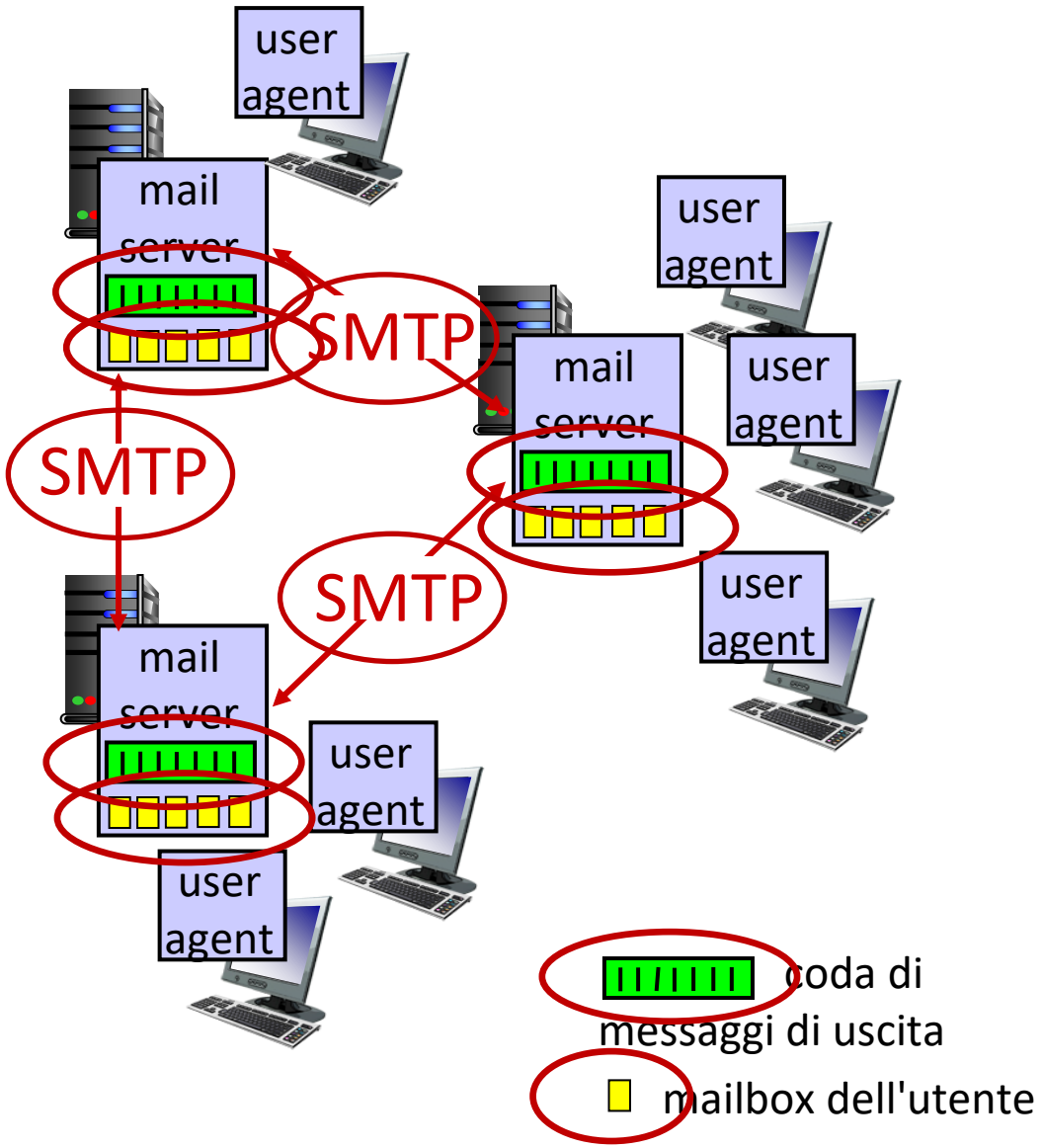
E-mail: mail servers

mail server:

- *mailbox* (casella di posta) contiene i messaggi in arrivo per l'utente
- *coda di messaggi* da trasmettere

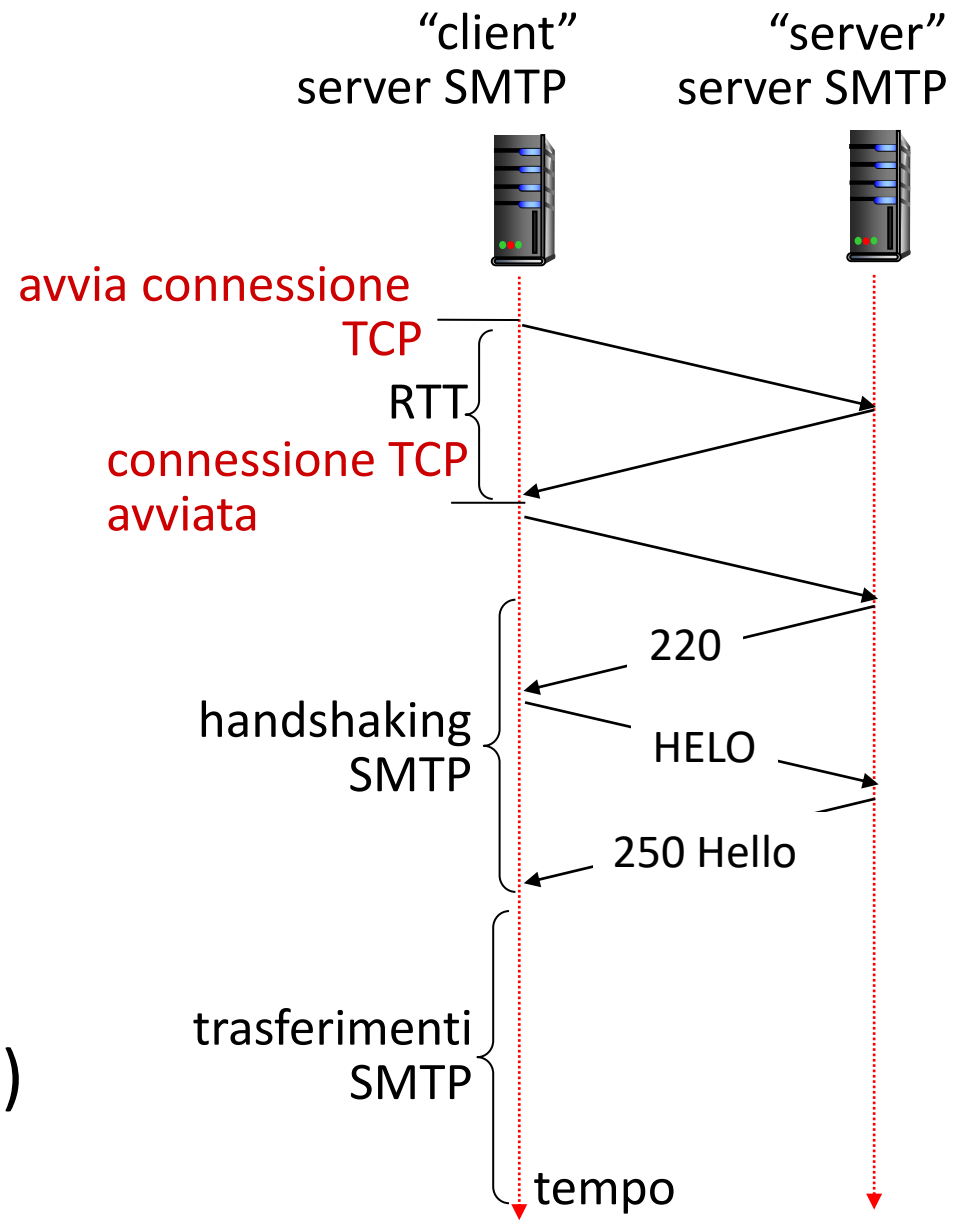
protocollo SMTP tra mail server per inviare messaggi email

- client: mail server trasmittente
- "server": mail server ricevente



SMTP RFC (5321)

- usa TCP per trasferire un modo affidabile i messaggi di posta elettronica dal client (mail server che avvia la connessione) al server, porta 25
  - Trasferimento diretto: il server trasmittente al server ricevente
- Tre fasi per il trasferimento
  - handshaking (saluto)
  - trasferimento dei messaggi
  - chiusura
- Interazione comando/risposta (come HTTP)
  - comandi: testo ASCII a 7 bit
  - risposta: codice di stato e espressione



Scenario: Alice invia un'e-mail a Bob

1) Alice usa il suo user agent per comporre il messaggio da inviare “a” ("to") bob@someschool.edu

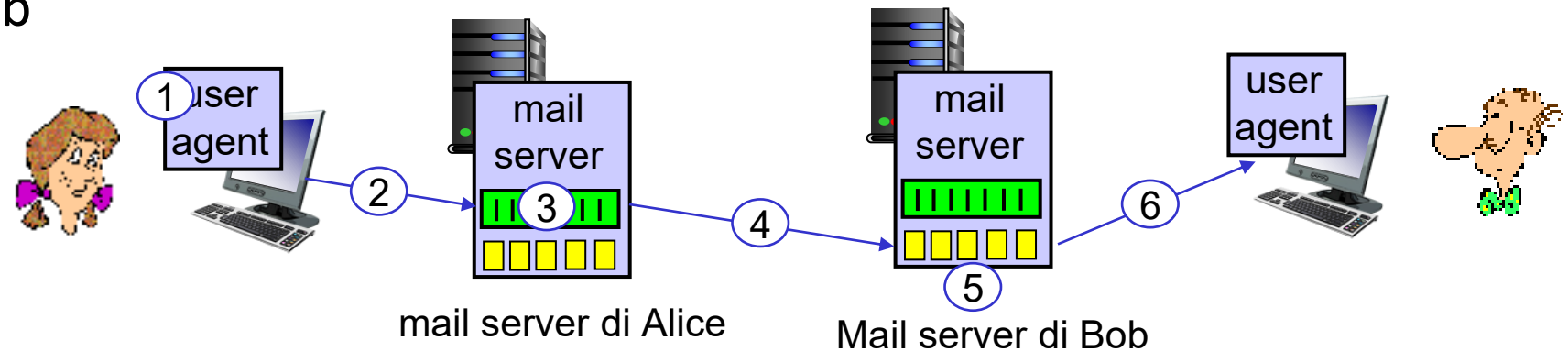
2) lo user agent di Alice invia un messaggio al server di posta di Alice; il messaggio è posto nella coda di messaggi

3) il lato client di SMTP apre una connessione TCP con il mail server di Bob

4) il client SMTP invia il messaggio di Alice sulla connessione TCP

5) il mail server di Bob pone il messaggio nella casella di posta di Bob

6) Bob invoca il suo user agent per leggere il messaggio



Esempio di interazione SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@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
```

SMTP: note finali

confronto con HTTP:

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

- SMTP usa connessione 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

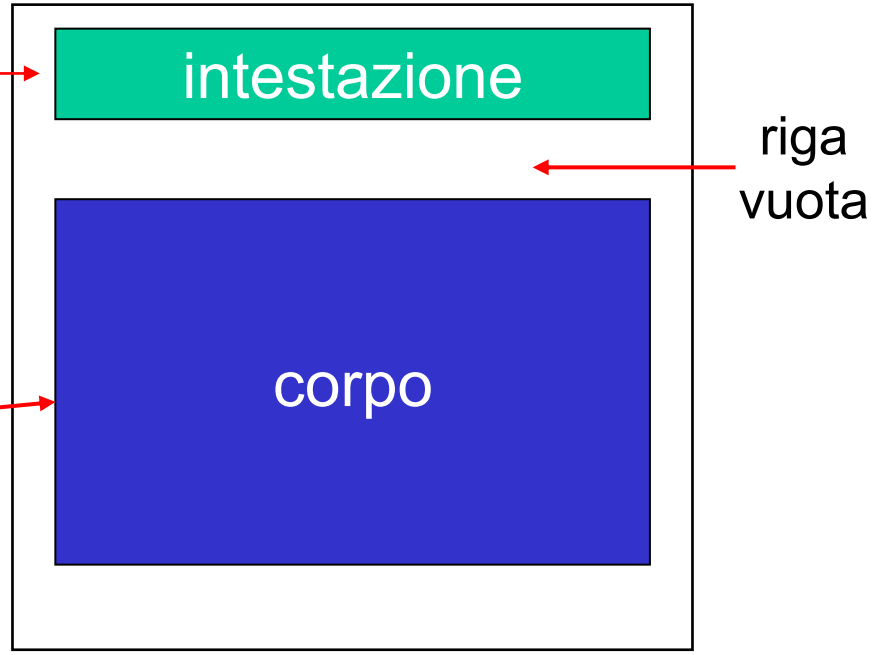
Formato dei messaggi di posta elettronica

SMTP: protocollo per scambiare messaggi di posta elettronica, definito nell'RFC 5321 (come RFC 7231 definisce HTTP)

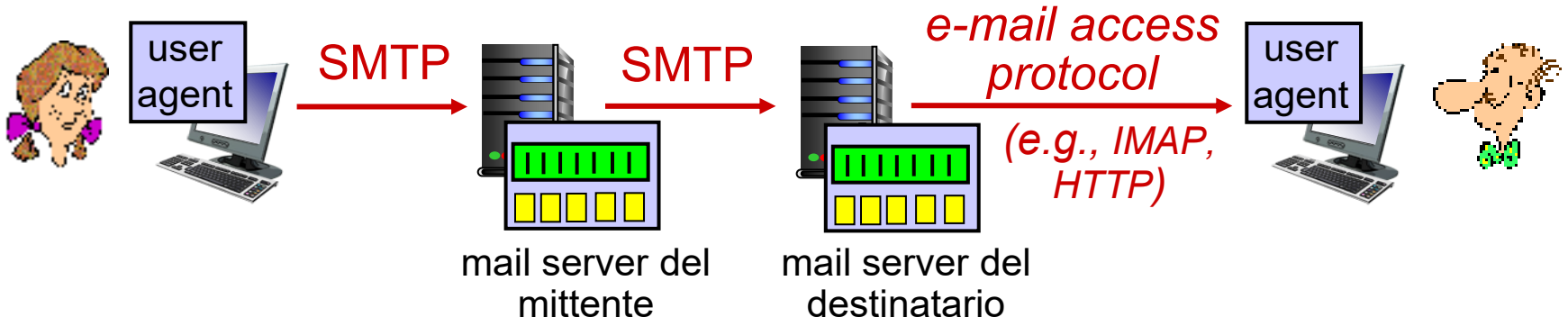
RFC 2822 definisce la *sintassi* dei messaggi di posta elettronica (come HTML definisce la sintassi per i documenti web)

- Righe di intestazione, per esempio.,
  - To/A:
  - From/Da:
  - Subject/Oggetto:

  differenti da comandi SMTP MAIL FROM:, RCPT TO:!
- corpo: il “messaggio” , soltanto caratteri ASCII



Protocolli di accesso alla posta



- SMTP: consegna/memorizzazione sul server del destinatario
- protocollo di accesso alla posta: ottenere i messaggi dal server
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messaggi memorizzati sul server, IMAP consente di recuperare, cancellare e archiviare i messaggi memorizzati sul server.
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. consente interfaccia web sopra a STMP (per l'invio) e IMAP (o POP) per il recupero delle email.