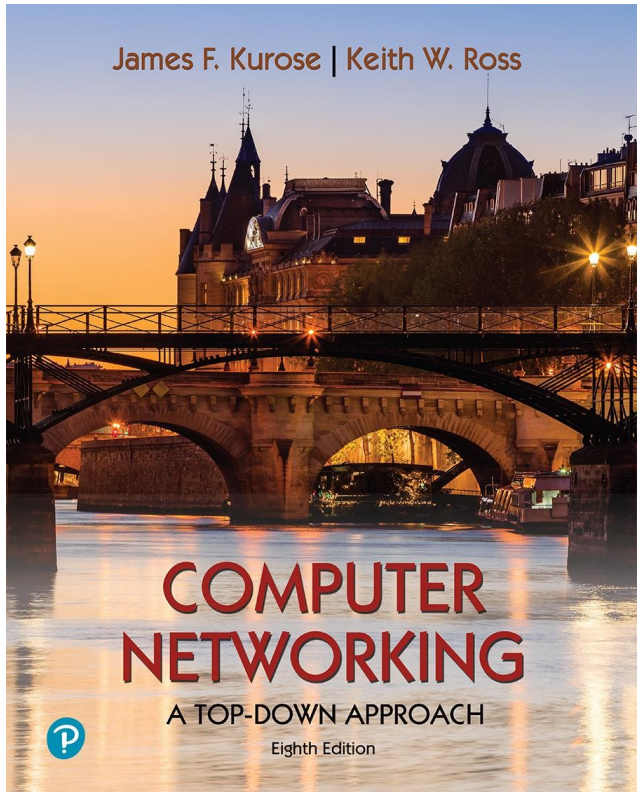
Capitolo 2
Livello di applicazione



Reti di computer: Un approccio dall'alto verso il basso

8th edizione n
Jim Kurose, Keith Ross

Pearson, 2020

Livello applicativo: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Il sistema dei nomi di dominio DNS

- Applicazioni P2P
- reti di streaming video e distribuzione di contenuti
- programmazione di socket con UDP e TCP

Alcune applicazioni di rete

- social network
- Web
- messaggi di testo
- e-mail
- giochi di rete multiutente
- video memorizzati in streaming (YouTube, Hulu, Netflix)
- Condivisione di file P2P

- voce su IP (ad esempio, Skype)
- videoconferenze in tempo reale (ad esempio, Zoom)
- Ricerca su Internet
- accesso remoto
- …

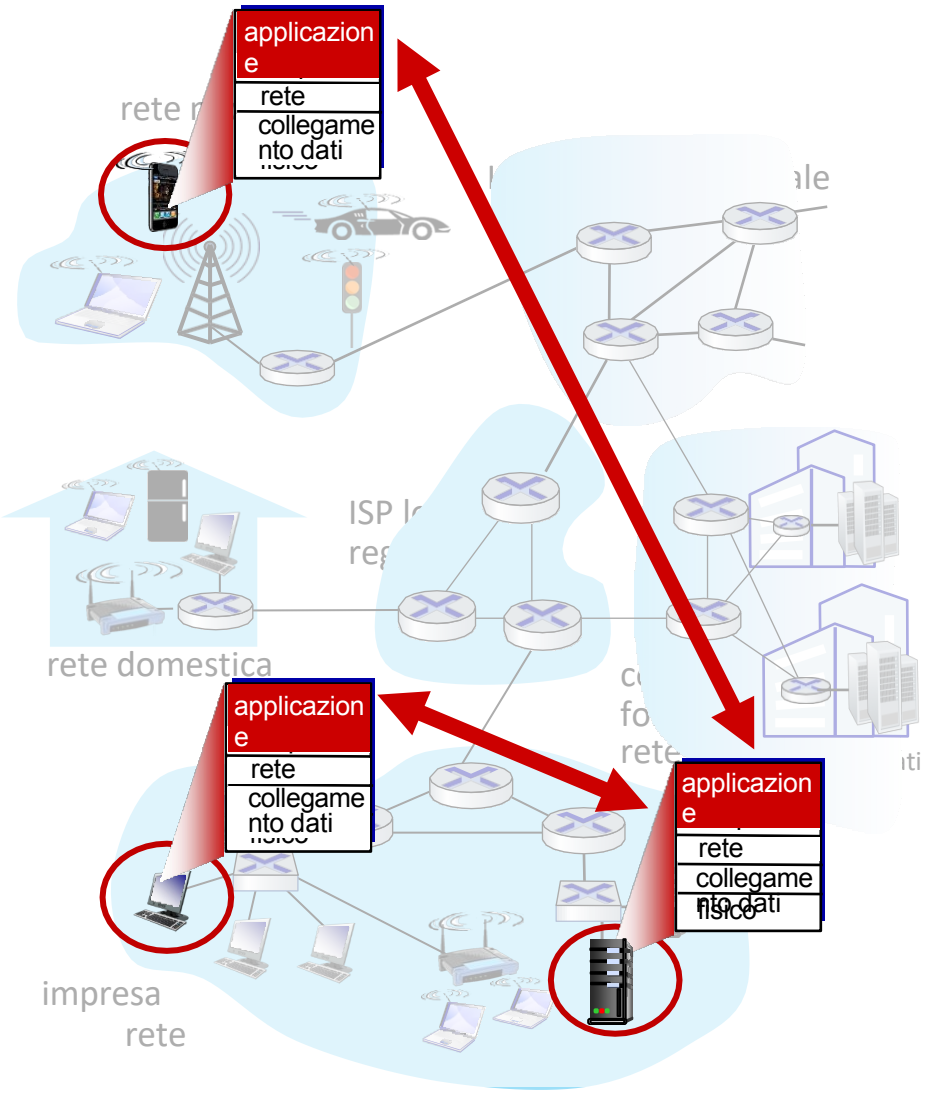
Creazione di un'applicazione di rete

scrivere programmi che:

- eseguiti su sistemi finali (diversi)
- comunicare in rete
- Ad esempio, il software del server Web comunica con il software del browser.

non è necessario scrivere software per i dispositivi network-core

- I dispositivi con nucleo di rete non eseguono applicazioni
- applicazioni sui sistemi finali



permette un rapido sviluppo delle applicazioni, la propagazione

Paradigma client-server

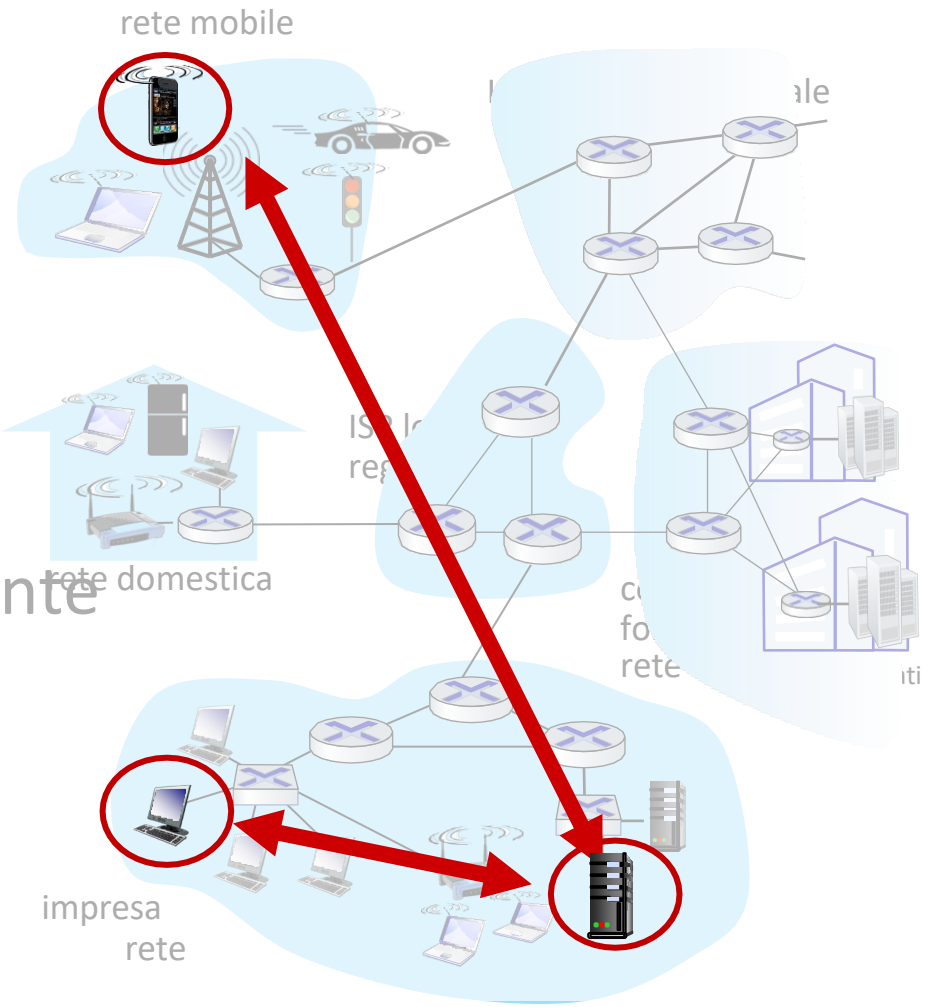
server:

- host sempre attivo
- indirizzo IP permanente
- spesso nei centri dati, per scalare

clienti:

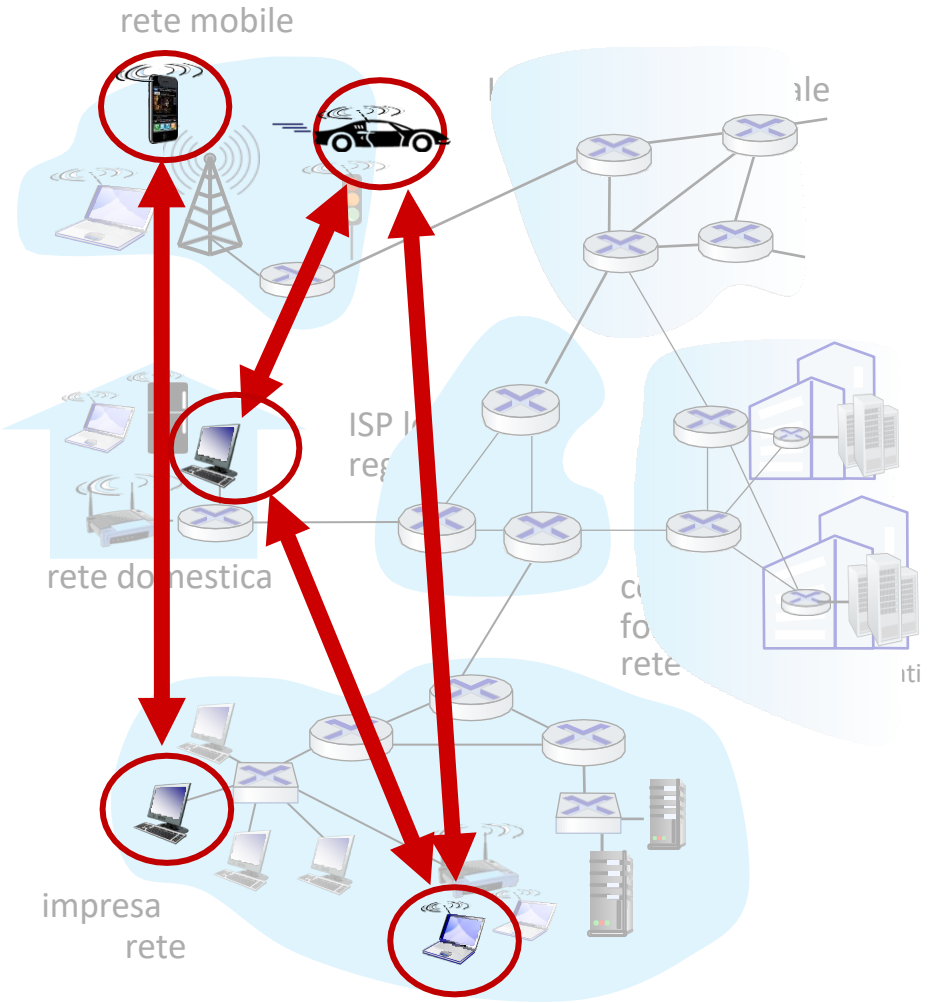
- contattare, comunicare con il server
- può essere collegato in modo intermittente
- possono avere indirizzi IP dinamici
- *non* comunicano direttamente tra loro

- esempi: HTTP, IMAP, FTP



Architettura peer-peer

- *nessun* server sempre attivo
- I sistemi finali arbitrari comunicano direttamente
- I peer richiedono un servizio ad altri peer e forniscono un servizio in cambio ad altri peer.
  - *autoscalabilità* - nuovi peer portano nuova capacità di servizio, così come nuove richieste di servizio
- I peer sono connessi a intermittenza e cambiano indirizzo IP.
  - gestione complessa



- esempio: Condivisione di file P2P [BitTorrent]

Processi che comunicano

processo: programma in esecuzione all'interno di un host

- all'interno dello stesso host, due processi comunicano utilizzando la comunicazione interprocesso (definita dal sistema operativo)
- I processi di host diversi

comunicano scambiandosi messaggi

clienti, server

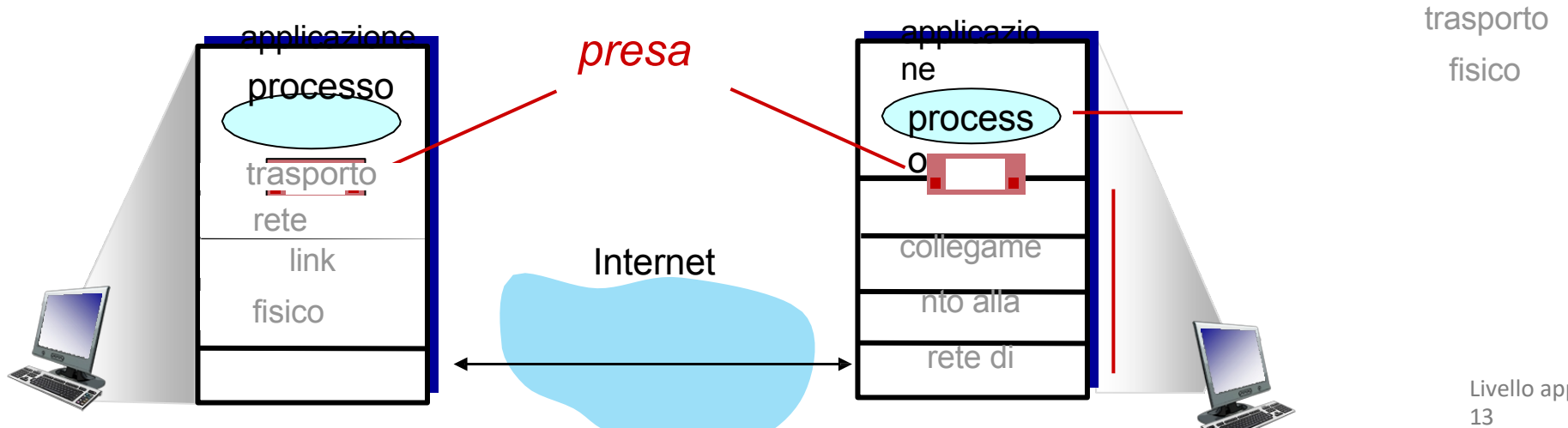
processo del cliente: processo che avvia la comunicazione

processo server: processo che attende di essere contattato

- Nota: le applicazioni con architetture P2P hanno processi client e processi server.

Prese di corrente

- il processo invia/riceve messaggi da/verso il proprio socket
- presa analoga alla porta
  - il processo di invio spinge il messaggio fuori dalla porta
  - Il processo di invio si affida all'infrastruttura di trasporto dall'altra parte della porta per consegnare il messaggio al socket del processo di ricezione.
  - due prese coinvolte: una per lato



controllato dallo sviluppatore dell'app

____controllato dal sistema operativo

Processi di indirizzamento

- per ricevere messaggi, il processo deve avere l'*identificatore*
- il dispositivo host ha un indirizzo IP univoco a 32 bit
- *D: L'*indirizzo IP dell'host su cui gira il processo è sufficiente per identificare il processo?
  - *R:* No, *molti* processi possono essere in esecuzione sullo stesso host.

- *L'identificatore* include sia l'indirizzo IP che i numeri di porta associati al processo sull'host.
- numeri di porta di esempio:
  - Server HTTP: 80
  - server di posta: 25
- per inviare un messaggio HTTP al server web gaia.cs.umass.edu:
  - Indirizzo IP: 128.119.245.12
  - numero di porta: 80
- a breve...

Di quale servizio di trasporto ha bisogno un'app?

integrità dei dati

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

tempistica

- alcune applicazioni (ad esempio, telefonia via Internet, giochi interattivi) richiedono un basso ritardo per essere "efficaci"

rendimento

- alcune applicazioni (ad esempio, quelle multimediali) richiedono una quantità minima di throughput per essere "efficaci"
- altre applicazioni ("applicazioni elastiche") utilizzano il throughput che ottengono.

sicurezza

- crittografia, integrità dei dati, …

Requisiti del servizio di trasporto: applicazioni comuni

applicazione		testo	**perdita di dati**
trasferimento/scaricamento di file	streaming audio/video		nessuna perdita
e-mail	giochi interattivi		nessuna perdita
Documenti web			nessuna
audio/video in tempo reale	messaggi di		perdita tollerante alle

perdite	**rendimento**	lastico	**sensibile al tempo?**
tollerante alle perdite	e	elastic	no
tollerante alle perdite	l	o	no
senza perdite	a	audio: 5Kbps-1Mbps	no
	s	video: 10Kbps-	sì, 10's msec
	t	5Mbps come sopra	
	i	Kbps+	sì, pochi
	c	elastico	secondi sì,
	o		10's msec sì e
	e		no

Servizi dei protocolli di trasporto Internet

Servizio TCP:

- *trasporto affidabile* tra il processo di invio e quello di ricezione
- *controllo del flusso: il* mittente non deve sovraccaricare il ricevitore
- *controllo della congestione:* strozzare il mittente in caso di sovraccarico della rete
- *orientato alla connessione: è* necessaria una configurazione tra i processi client e server
- *non fornisce:* tempistica, garanzia di throughput minimo, sicurezza

Servizio UDP:

- *trasferimento di dati inaffidabile* tra il processo di invio e quello di ricezione
- *non fornisce:* affidabilità, controllo del flusso, controllo della congestione, tempistica, garanzia di throughput, sicurezza o impostazione della connessione.

D: perché preoccuparsi? *Perché* esiste un UDP?

Applicazioni Internet e protocolli di trasporto

applicazione	audio/video	protocollo di livello
trasferimento/scaricamento di file		FTP [RFC 959] SMTP [RFC 5321]
e-mail		HTTP [RFC 7230, 9110]
Documenti web		SIP [RFC 3261], RTP [RFC 3550], o HTTP
Telefonia Internet		proprietario [RFC 7230], DASH
giochi interattivi in streaming		WOW, FPS (proprietario)

protocollo di trasporto

TCP o UDP

TC

T

P UDP o

C

TCP

P

T

C

P

T

C

P

Protezione del TCP

Vanilla TCP e UDP socket:

- nessuna crittografia
- le password in chiaro inviate nel socket attraversano Internet in chiaro (!)

Sicurezza del livello di trasporto (TLS)

- fornisce connessioni TCP crittografate
- integrità dei dati
- autenticazione del punto finale

TLS implementato nel livello applicazione

- Le applicazioni utilizzano le librerie TLS, che a loro volta utilizzano il TCP
- testo in chiaro inviato al "socket" attraversare Internet *in modo criptato*
- di più: Capitolo 8

Livello applicazione: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Il sistema dei nomi di dominio DNS
- Applicazioni P2P
- reti di streaming video e distribuzione di contenuti
- programmazione di socket con UDP e TCP

Web e HTTP

Prima di tutto, un breve ripasso...

- La pagina web è composta da *oggetti,* ognuno dei quali può essere memorizzato su diversi server web.
- L'oggetto può essere un file HTML, un'immagine JPEG, un'applet Java, un file audio...
- La pagina web è costituita da un *file HTML di base* che include *diversi oggetti di riferimento, ciascuno* indirizzabile tramite un *URL, ad* es,

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

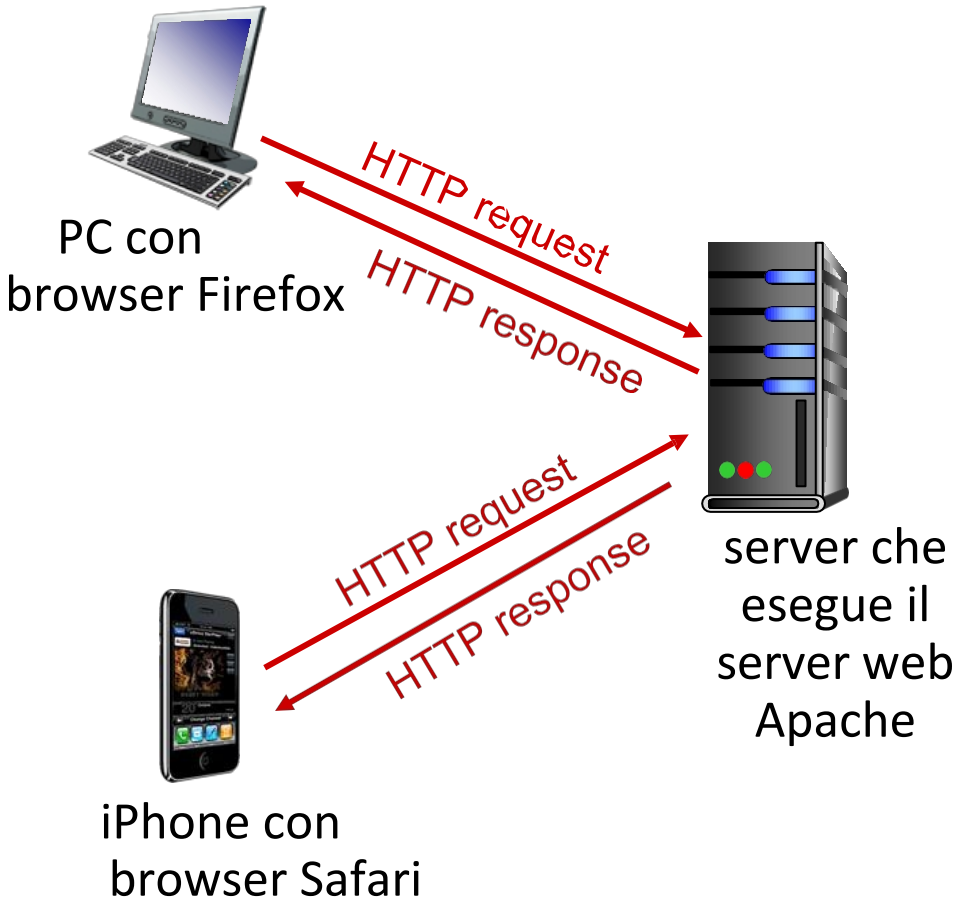
nome dell'host

nome del percorso

Panoramica HTTP

HTTP: protocollo di trasferimento di ipertesti

- Protocollo di livello applicativo del Web
- modello client/server:
  - *client:* browser che richiede, riceve (utilizzando il protocollo HTTP) e "visualizza" gli oggetti Web
  - *server: Il* server web invia (tramite il protocollo HTTP) oggetti in risposta alle richieste.



Panoramica HTTP (continua)

HTTP utilizza il protocollo TCP:

- il client avvia una connessione TCP (crea un socket) al server, porta 80
- Il server accetta la connessione TCP dal client
- Messaggi HTTP (messaggi di protocollo di livello applicativo) scambiati tra il browser (client HTTP) e il server Web (server HTTP).
- Connessione TCP chiusa

HTTP è "senza stato"

- Il server *non* conserva *alcuna* informazione sulle richieste passate del cliente

a parte i protocolli che mantengono lo "stato" sono complessi!

- la storia passata (stato) deve essere mantenuta
- se il server/client si blocca, il loro Il concetto di "Stato" può essere incoerenti, devono essere riconciliati

Connessioni HTTP: due tipi

HTTP non persistente

1. Connessione TCP aperta
2. al massimo un oggetto inviato tramite connessione TCP
3. Connessione TCP chiusa

Il download di più oggetti richiedeva più connessioni

HTTP persistente

- Connessione TCP aperta a un server
- più oggetti possono essere inviati su *una singola* connessione TCP tra il client e il server.
- Connessione TCP chiusa

HTTP non persistente: esempio

L'utente inserisce l'URL: **`www.someSchool.edu/someDepartment/home.index`**
(contenente testo, riferimenti a 10 immagini jpeg)

someDepartment/home.index

1a. Il client HTTP avvia una connessione TCP al server HTTP (processo) all'indirizzo www.someSchool.edu sulla porta 80.

2. Il client HTTP invia *un messaggio di richiesta* HTTP (contenente un URL) nel socket di connessione TCP. Il messaggio indica che il client desidera l'oggetto

tempo

1b. Il server HTTP dell'host www.someSchool.edu in attesa di una connessione TCP sulla porta 80 "accetta" la connessione, notificando il client

3. Il server HTTP riceve il messaggio di richiesta, forma il *messaggio di risposta* contenente l'oggetto richiesto e invia il messaggio nel suo socket.

HTTP non persistente: esempio (segue)

L'utente inserisce l'URL: **`www.someSchool.edu/someDepartment/home.index`**
(contenente testo, riferimenti a 10 immagini jpeg)

4. Il server HTTP chiude la connessione TCP.

5. Il client HTTP riceve un messaggio di risposta contenente un file html e lo visualizza. Analizzando il file html, trova 10 oggetti jpeg di riferimento.

6. Ripetere le fasi 1-5 per ciascuno di 10 oggetti jpeg

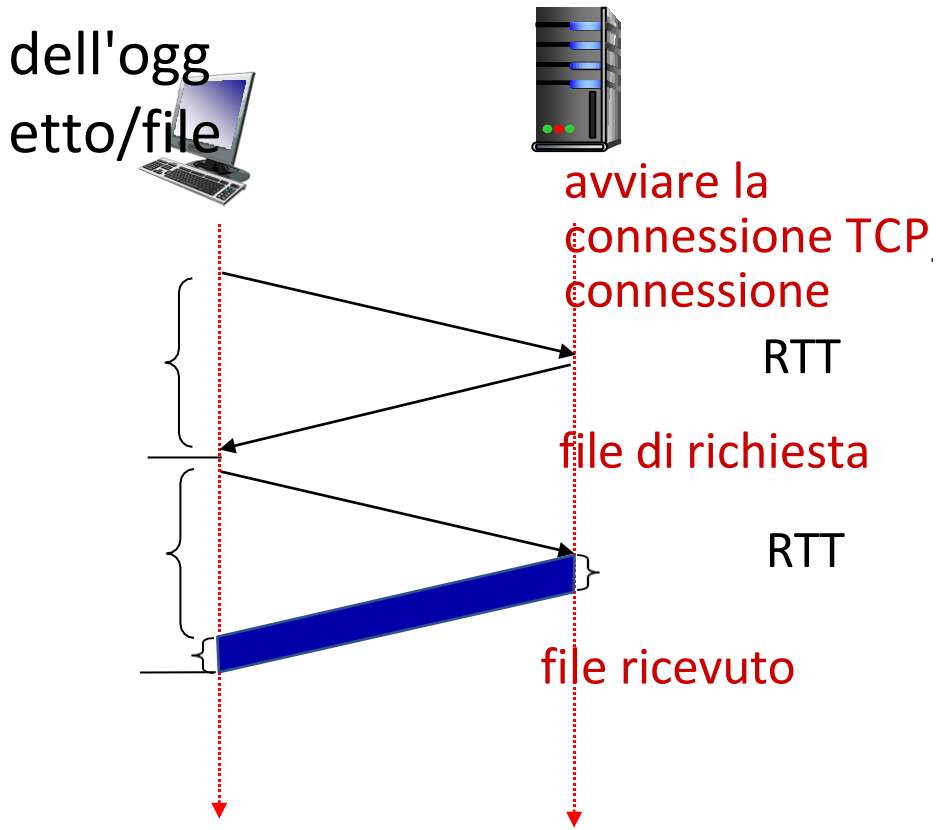
tempo

HTTP non persistente: tempo di risposta

RTT (definizione): tempo di percorrenza di un piccolo pacchetto dal client al server e viceversa.

Tempo di risposta HTTP (per oggetto):

- un RTT per avviare la connessione TCP
- un RTT per la richiesta HTTP e per i primi byte della risposta HTTP da restituire
- tempo di trasmissione



mpo di
trasmiss
ione del
file

te

tempo tempo

Tempo di risposta HTTP non persistente = 2RTT+tempo di trasmissione del file

HTTP persistente (HTTP 1.1)

Problemi HTTP non persistenti:

- richiede 2 RTT per oggetto
- Overhead del sistema operativo per *ogni* connessione TCP
- I browser spesso aprono più connessioni TCP parallele per recuperare gli oggetti di riferimento in parallelo.

HTTP persistente (HTTP1.1):

- il server lascia aperta la connessione dopo l'invio della risposta
- messaggi HTTP successivi tra lo stesso client/server, inviati su una connessione aperta.
- il client invia le richieste non appena incontra un oggetto di riferimento
- un solo RTT per tutti gli

oggetti di riferimento (dimezzando i tempi di risposta)

Messaggio di richiesta HTTP

- due tipi di messaggi HTTP: *richiesta, risposta*
- Messaggio di richiesta HTTP:
  - ASCII (formato leggibile dall'uomo)

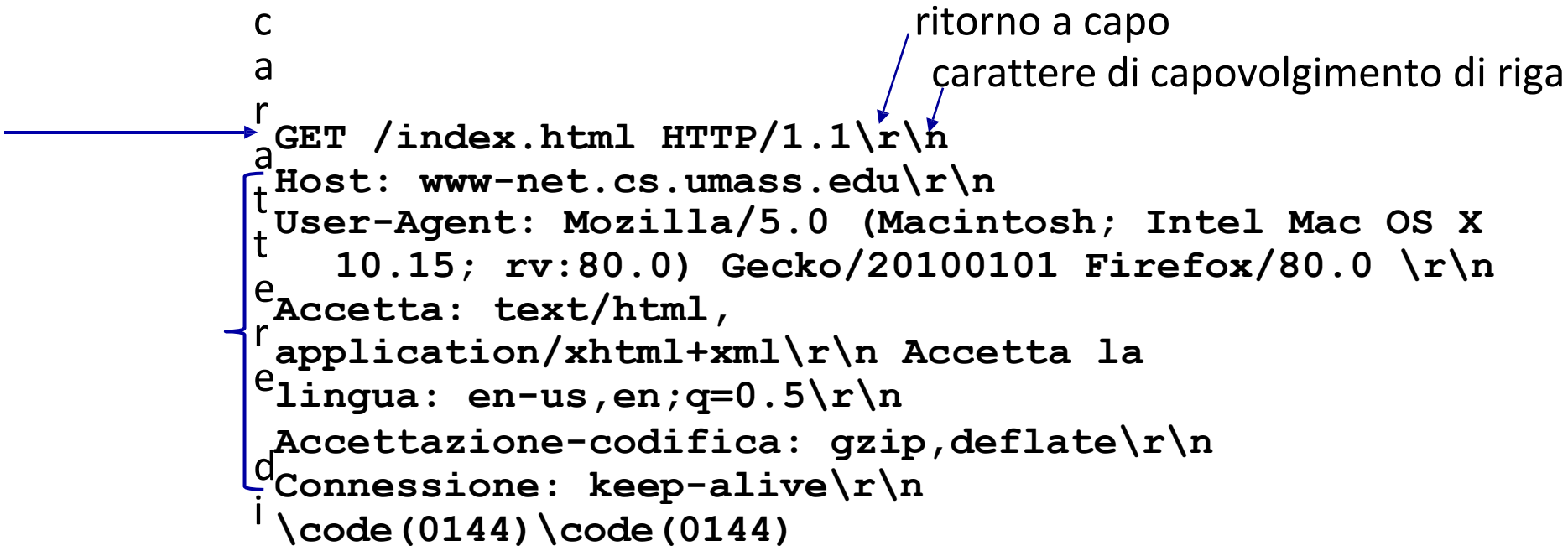
avanzamento di riga all'inizio della riga indica

linea di richiesta (GET,
POST,
Comandi HEAD)

intestazione
linee

ritorno a capo, ⟶

esempi: http://gaia.cs.umass.edu/kurose_ross/interactive/



* Per saperne di più, consultate gli esercizi interattivi online

esempi: http://gaia.cs.umass.edu/kurose_ross/interactive/

Altri messaggi di richiesta HTTP

Metodo POST:

- la pagina web include spesso un modulo di input
- l'input dell'utente inviato dal client al server nel corpo del messaggio di richiesta HTTP POST

```
www.somesite.com/animalsearch?mo
nkeys&banana
```

Metodo GET (per l'invio di dati al server):

- includere i dati dell'utente nel campo URL di HTTP

  Messaggio di richiesta GET (dopo un '?'):

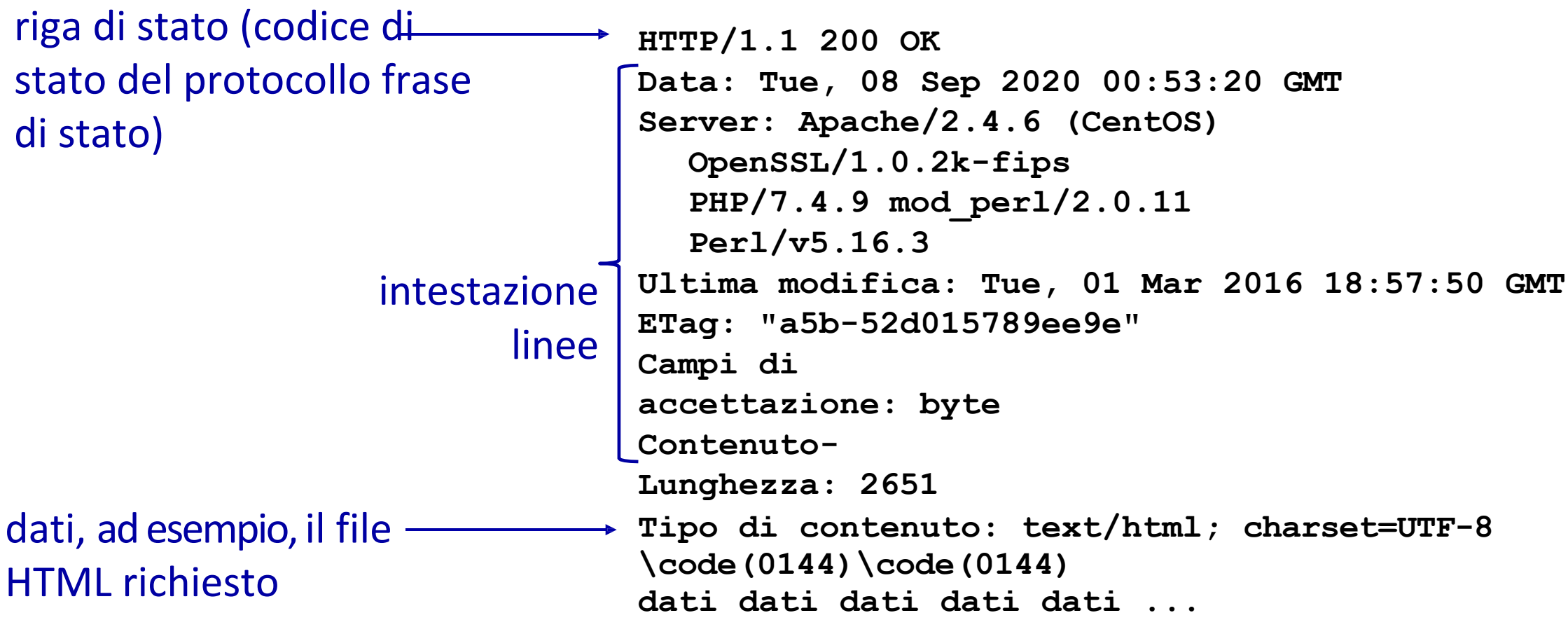
Metodo HEAD:

- richiede le intestazioni (solo) che verrebbero restituite *se* l'URL specificato fosse richiesto con il metodo HTTP GET.

Metodo PUT:

- carica un nuovo file (oggetto) sul server
- sostituisce completamente il file esistente all'URL specificato con il contenuto del corpo dell'entità del messaggio di richiesta HTTP POST.

Messaggio di risposta HTTP



* Per ulteriori esempi, consultare gli esercizi interattivi online: http://gaia.cs.umass.edu/kurose_ross/interactive/

Codici di stato delle risposte HTTP

- Il codice di stato appare nella prima riga del messaggio di risposta da server a client.
- alcuni codici di

esempio: 200 OK

- richiesta riuscita, oggetto richiesto più avanti nel messaggio

301 Trasferito in modo permanente

- oggetto richiesto spostato, nuova posizione specificata più avanti in questo messaggio (nel campo Location:)

400 Richiesta errata

- messaggio di richiesta non compreso dal server

404 Non trovato

- il documento richiesto non è stato trovato su questo server

505 Versione HTTP non supportata

Provare l'HTTP (lato client) per conto proprio

1. netcat al vostro server Web preferito:

% nc -c -v gaia.cs.umass.edu 80

- apre una connessione TCP alla porta 80 (porta predefinita del server HTTP) di gaia.cs.umass.edu.
- Qualsiasi cosa venga digitata sarà inviata alla porta 80 di gaia.cs.umass.edu

2. in una richiesta HTTP GET:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

Digitando questo (premendo due volte il tasto di ritorno a capo), si invia questa richiesta GET minima (ma completa) a HTTP server

3. guardare il messaggio di risposta inviato dal server HTTP!

(o utilizzare Wireshark per esaminare le richieste/risposte HTTP catturate)

Mantenimento dello stato dell'utente/server: cookie

I siti web e i browser dei clienti utilizzano *i cookie* per mantenere alcuni stati tra le transazioni.

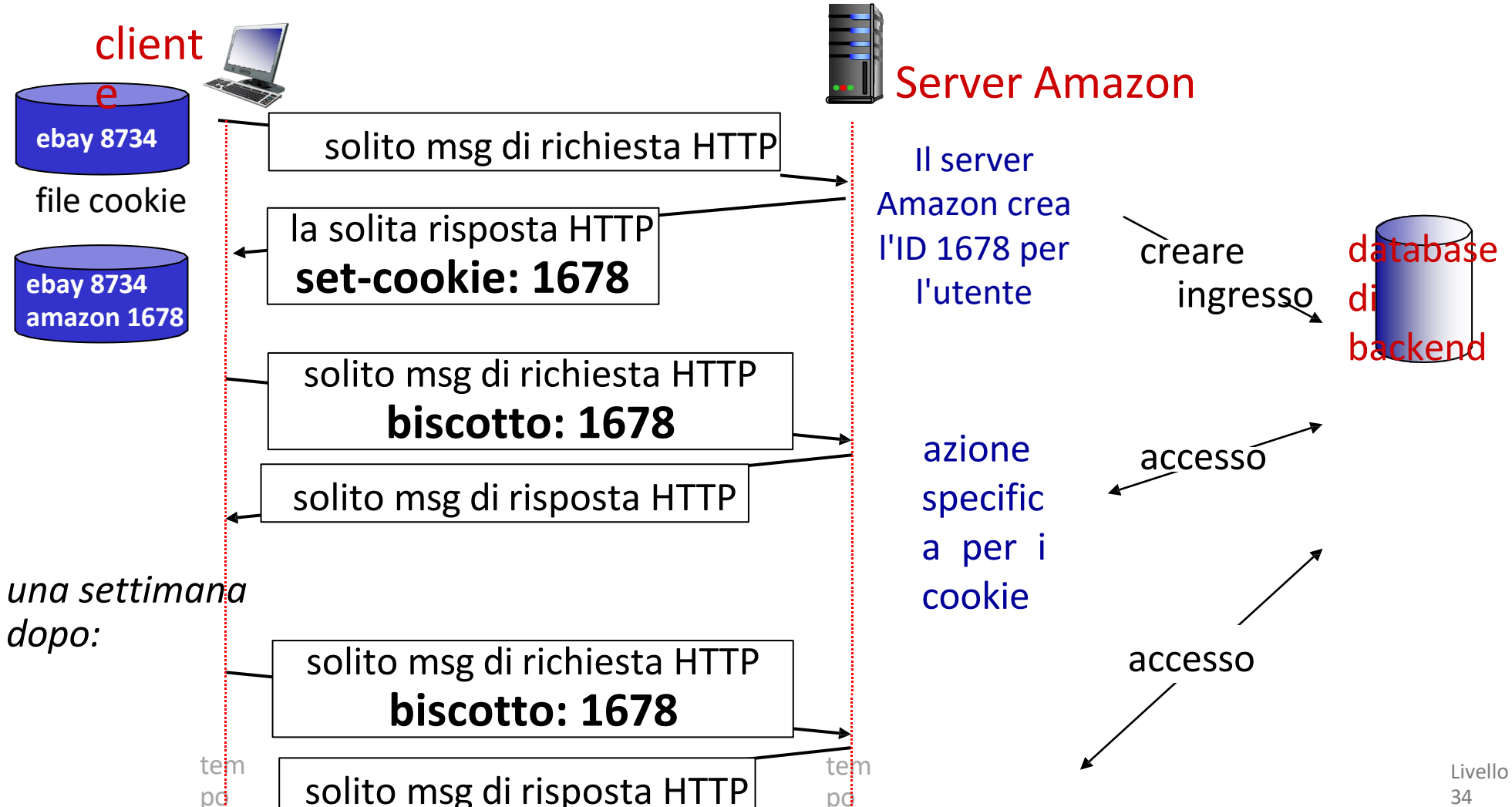
quattro componenti:

1) riga dell'intestazione cookie della *risposta* HTTP messaggio
2) riga dell'intestazione del cookie nel prossimo HTTP messaggio di *richiesta*
3) file cookie conservato sull'host dell'utente, gestito dal browser dell'utente stesso
4) database back-end sul sito Web

Esempio:

- Susan utilizza il browser sul laptop e visita per la prima volta un sito di e-commerce specifico.
- quando la richiesta HTTP iniziale arriva al sito, il sito crea:
  - ID univoco (alias "cookie")
  - nel database di backend per l'ID
- le successive richieste HTTP da parte di Susan a questo sito conterranno il valore ID del cookie, consentendo al sito di "identificare" Susan

Mantenimento dello stato dell'utente/server: cookie



azione specifica per i cookie

tempo

tempo

Cookie HTTP: commenti

Per cosa possono essere utilizzati i cookie:

- autorizzazione
- carrelli della spesa
- raccomandazioni
- stato della sessione utente (e-mail Web)

Sfida: come mantenere lo stato?

- *agli endpoint del protocollo:* mantenere lo stato al mittente/ricevitore per più transazioni
- *nei messaggi: i* cookie nei messaggi HTTP portano con sé lo stato

cookie e privacy:

- I cookie consentono ai siti di *conoscere* molte informazioni sull'utente
- per esempio di terze parti I cookie (tracking cookie) consentono di tracciare un'identità comune (valore del cookie) su più siti web.

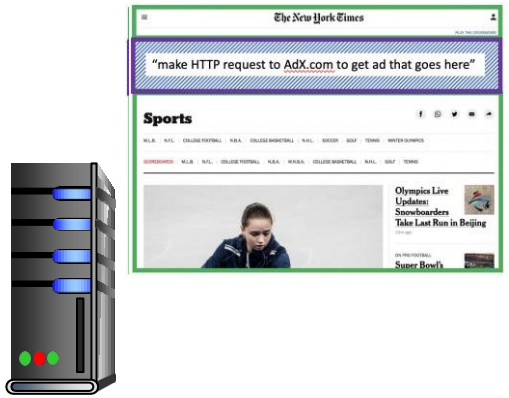
da parte sul loro sito

Esempio: visualizzazione di una pagina web del NY Times

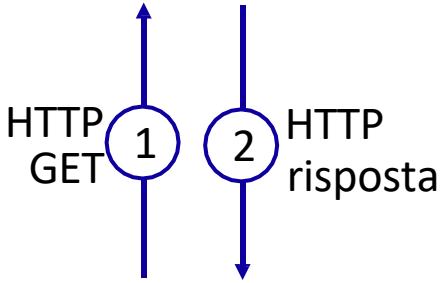
1 2 GET file html di base da nytimes.com

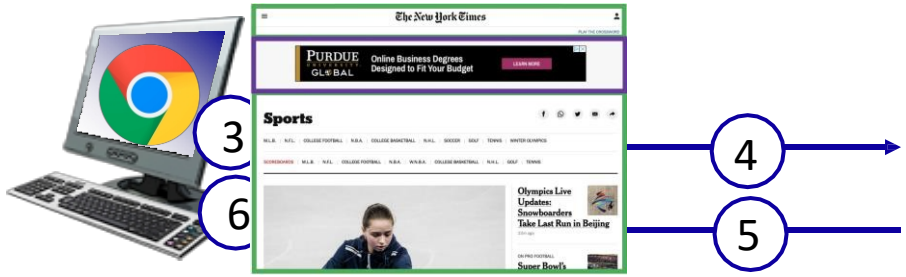
4 5 recuperare l'annuncio da AdX.com

7 visualizzare la pagina composta



nytimes.com

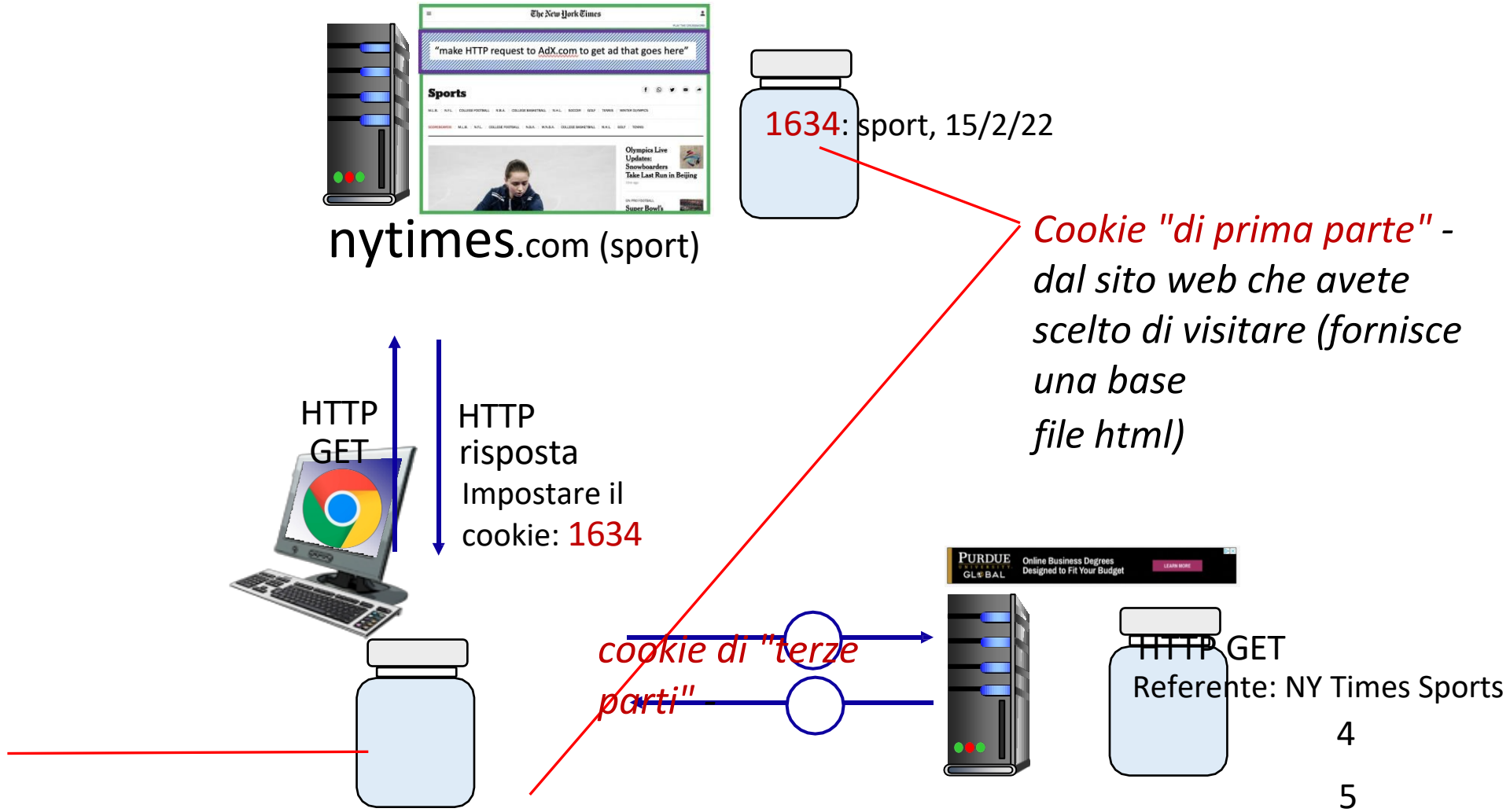




Pagina del NY Times con annuncio incorporato visualizzato

AdX.com

Cookie: tracciamento del comportamento di navigazione dell'utente



Risposta HTTP

dal sito web <u>non</u>
scegliere di visitare

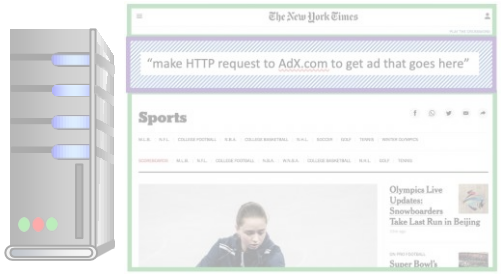
NY Times: 1634
AdX: 7493

7493: NY Times sport, 15/2/22
Impostare il cookie: 7493

AdX.com

Cookie: tracciamento del comportamento di navigazione dell'utente

calzini.com
o

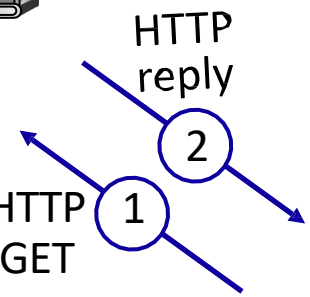


nytimes.com

sport, 15/2/22

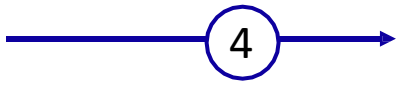
AdX:

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

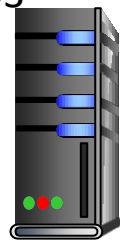


HTTP GET

Referente: socks.com, cookie: 7493



Risposta HTTP



7493: NY Times sport, 15/2/22
7493: socks.com, 16/02/22

NY Times: 1634

AdX: 7493

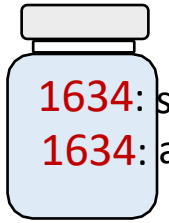
Impostare il cookie: 7493

AdX.com

Cookie: tracciamento del comportamento di navigazione dell'utente (un giorno dopo)

calzini.com

nytimes.com (arte)



HTTP GET
biscotto: 1634

HTTP risposta
Impostare il cookie: 1634

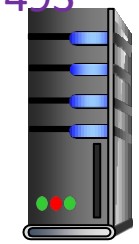
HTTP GET
Referente: nytimes.com, cookie: 7493

4

5

Risposta HTTP

NY Times: 1634



A

X: 7493

Impostare il cookie: 7493

Annuncio restituito per i calzini!

AdX.com

Cookie: tracciamento del comportamento di navigazione dell'utente

I cookie possono essere utilizzati per:

- tracciare il comportamento degli utenti su un determinato sito web (cookie di prima parte)
- tracciare il comportamento dell'utente su più siti web (cookie di terze parti) senza che l'utente scelga mai 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 link invisibile

tracciamento da parte di terzi tramite cookie:

- disabilitato per impostazione predefinita nei browser Firefox e Safari
- da disabilitare nel browser Chrome nel 2023

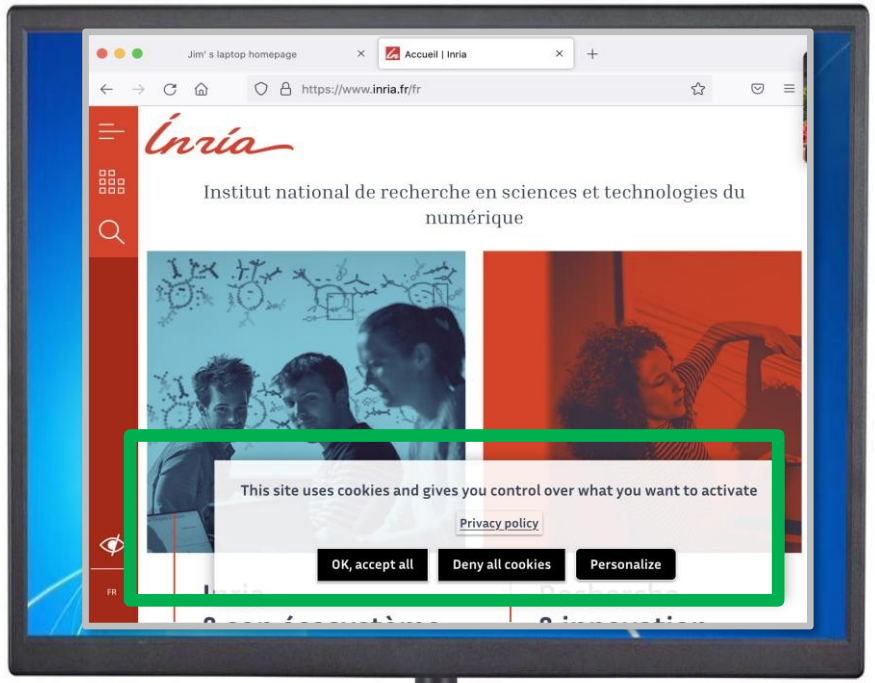
GDPR (Regolamento generale sulla protezione dei dati dell'UE) e cookie

"Le persone fisiche possono essere associate a identificatori online [...] come indirizzi di protocollo internet, identificatori di cookie o altri identificatori [...].

Ciò può lasciare tracce che, in particolare se combinate con identificatori unici e altre informazioni ricevute dai server, possono essere utilizzate per creare profili delle persone fisiche e identificarle."

GDPR, considerando 30 (maggio 2018)

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

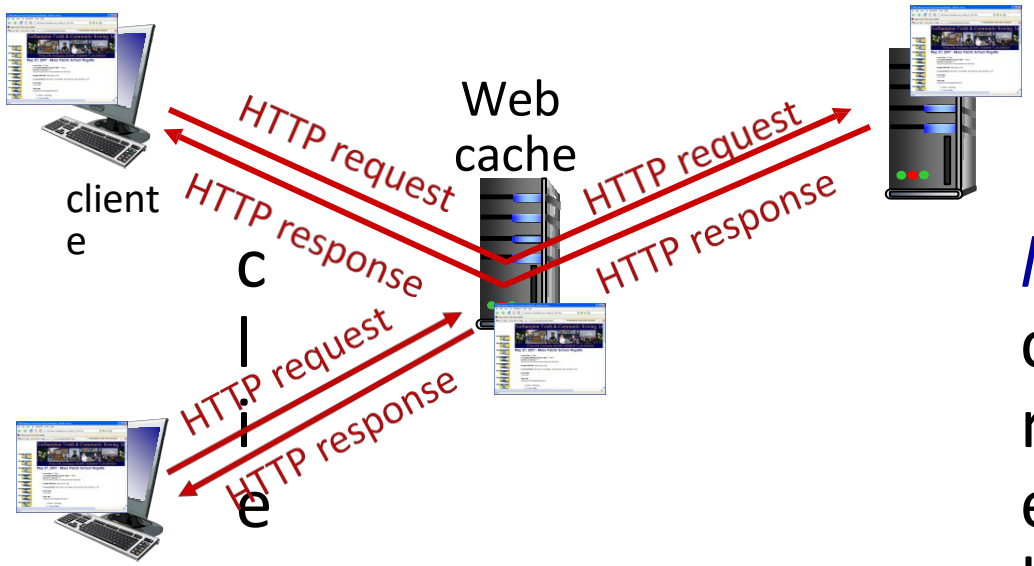


L'utente ha un controllo esplicito sull'autorizzazione o meno dei cookie.

Cache web

Obiettivo: soddisfare le richieste dei clienti senza coinvolgere il server di origine

- l'utente configura il browser per puntare a una *cache Web* (locale)
- il browser invia tutte le richieste HTTP alla cache
  - *se l'*oggetto è nella cache: la cache restituisce l'oggetto al client
  - *e lse* cache richiede l'oggetto al server



di origine, memorizza nella cache l'oggetto ricevuto e lo restituisce al cliente

server di origine

client e

Cache web (alias server proxy)

- La cache web agisce sia come client che come
  - server per il client richiedente originale
  - dal client al server di origine
- Il server comunica alla cache dell'oggetto è consentita la cache in

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

intestazione della risposta:

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

Perché il web caching?

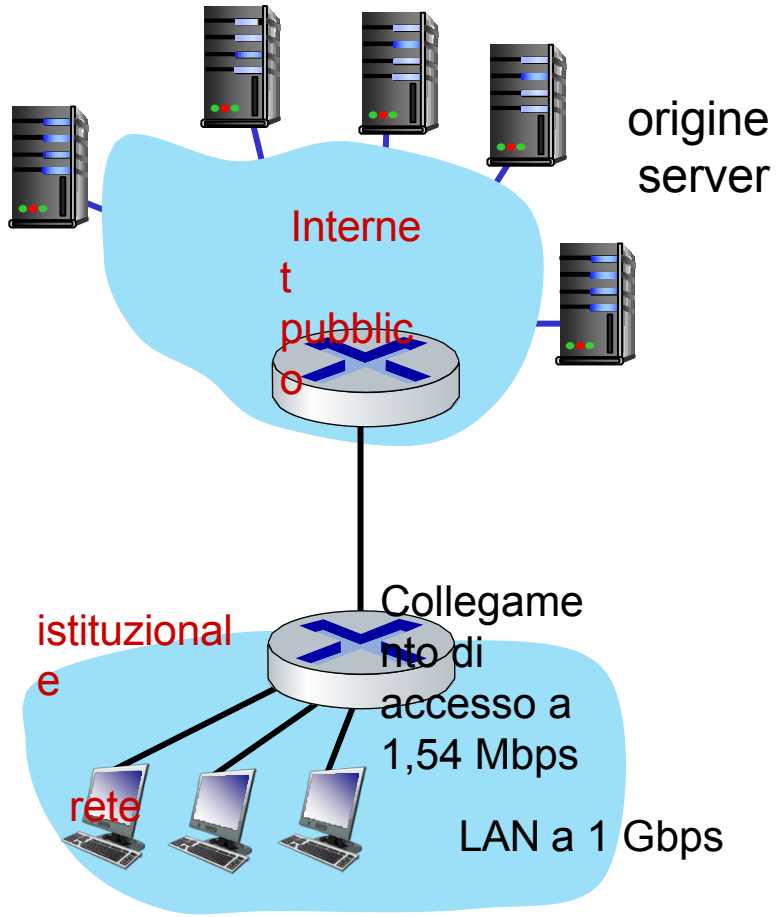
- ridurre i tempi di risposta alle richieste dei clienti
  - la cache è più vicina al client
- ridurre il traffico sul collegamento di accesso di un'istituzione
- Internet è densa di cache
  - consente ai fornitori di contenuti "poveri

per distribuire in modo più efficace i contenuti

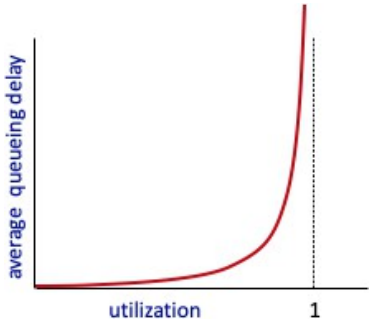
Esempio di caching

Scenario:

- velocità di accesso: 1,54 Mbps
- RTT dal router istituzionale al server: 2 sec
- Dimensione oggetto web: 100K bit
- velocità media di richiesta dai browser ai server di origine: 15/sec
  - velocità media di trasmissione dei dati ai browser: 1,50 Mbps



Esempio di caching



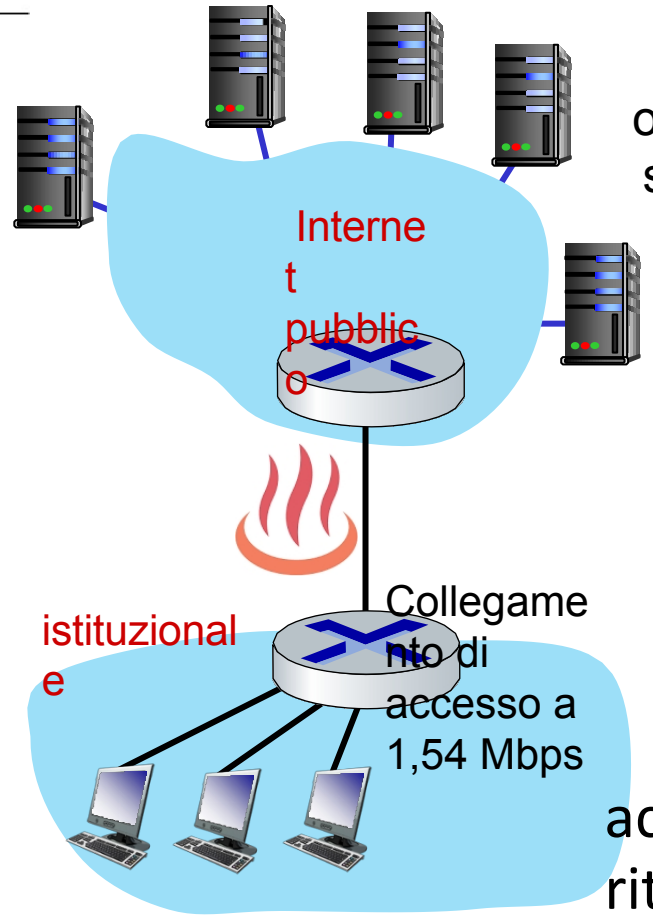
Scenario:

- velocità di accesso: 1,54 Mbps
- RTT dal router istituzionale al server: 2 sec
- Dimensione oggetto web: 100K bit
- velocità media di richiesta dai browser ai server di origine: 15/sec
  - velocità media di trasmissione dei dati ai browser: 1,50 Mbps

Prestazioni:

- utilizzo del collegamento di accesso = .97

- Utilizzo della LAN: .0015
- ritardo finale = ritardo Internet + ritardo del collegamento di accesso + ritardo della LAN



= 2 sec + minuti + usec

rete

LAN a 1 Gbps

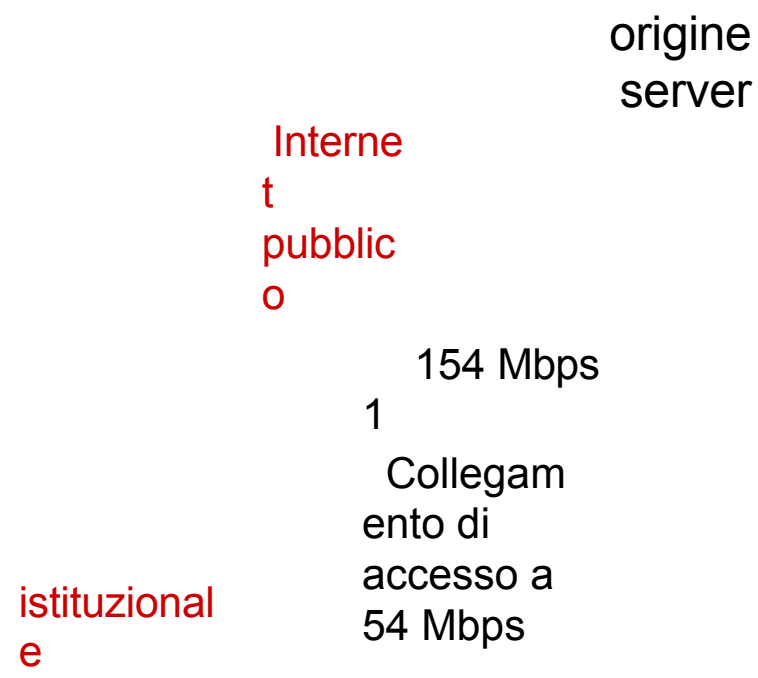
Opzione 1: acquistare un collegamento di accesso più veloce

Scenario:

- velocità di accesso: 1,54 Mbps
- RTT dal router istituzionale al server: 2 sec
- Dimensione oggetto web: 100K bit
- velocità media di richiesta dai browser ai server di origine: 15/sec
  - velocità media di trasmissione dei dati ai browser: 1,50 Mbps

Prestazioni:

- utilizzo del collegamento di accesso = .97
- Utilizzo della LAN: .0015



- ritardo finale = ritardo

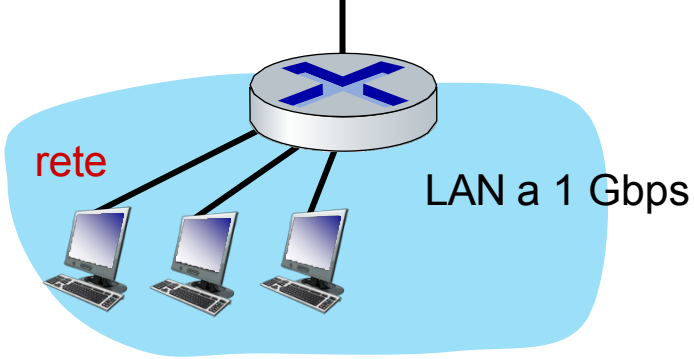
Internet +

ritardo del collegamento di accesso + ritardo della LAN

= 2 sec + minuti + usec

msec

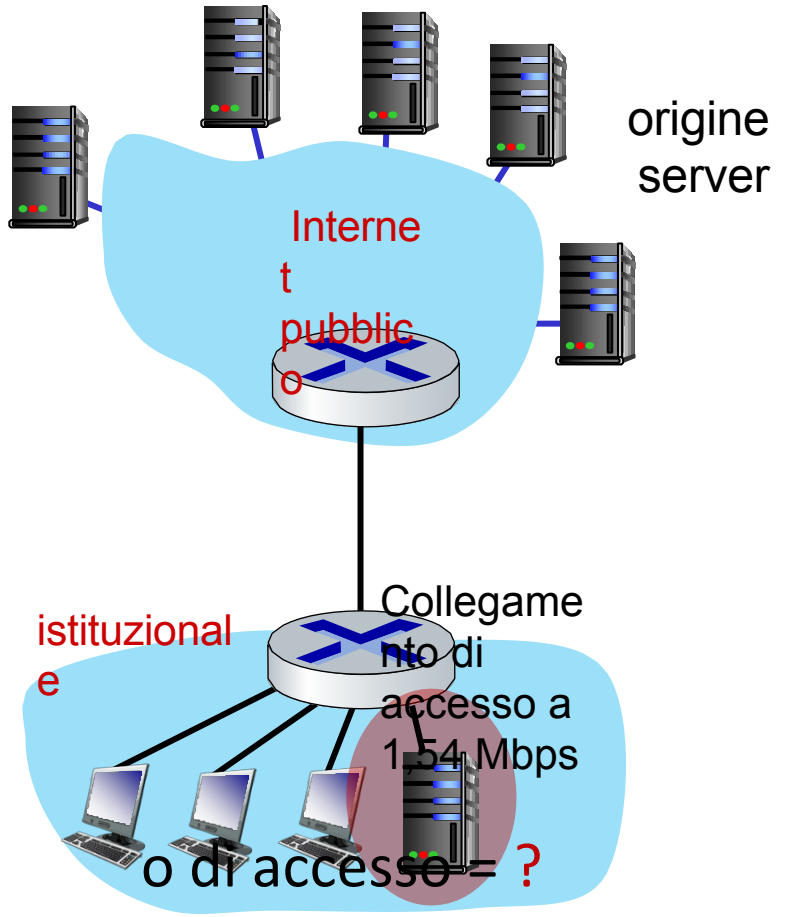
Costo: collegamento di accesso più veloce (costoso!)



Opzione 2: installare una cache web

Scenario:

- velocità di accesso: 1,54 Mbps
- RTT dal router istituzionale al server: 2 sec
- Dimensione oggetto web: 100K bit
- velocità media di richiesta dai browser ai server di origine: 15/sec
  - velocità media di trasmissione dei dati ai browser: 1,50 Mbps



Costo: cache web (economica!)

Prestazioni:

- Utilizzo della LAN: .?
- utilizzo del collegament
- o di accesso = ?

rete

LAN a

1 Gbps

Come calcolare il collegamento utilizzo, ritardo?

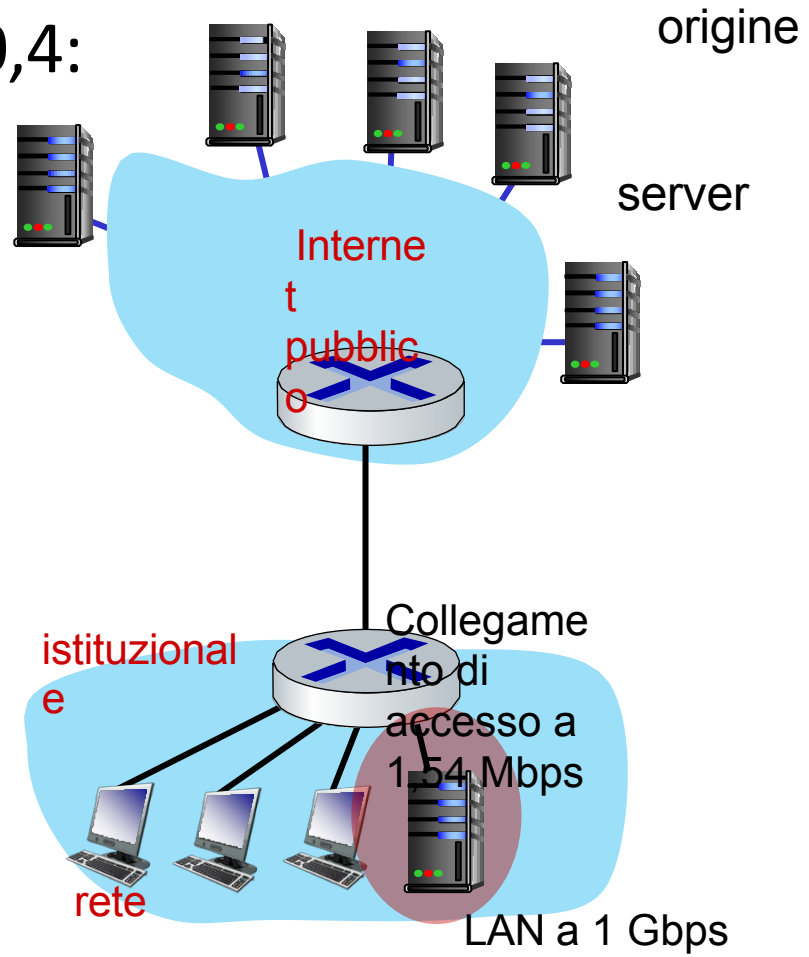
- ritardo medio finale = ?

cache web locale

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

supponiamo che il tasso di hit della cache sia 0,4:

- 40% di richieste servite dalla cache, con un basso (msec) ritardo
- 60% di richieste soddisfatte all'origine
  - tasso ai browser attraverso il collegamento di accesso
    = 0,6 * 1,50 Mbps = .9 Mbps
  - utilizzo del collegamento di accesso = 0,9/1,54 = .58 significa basso (msec) ritardo di accodamento sul collegamento di accesso

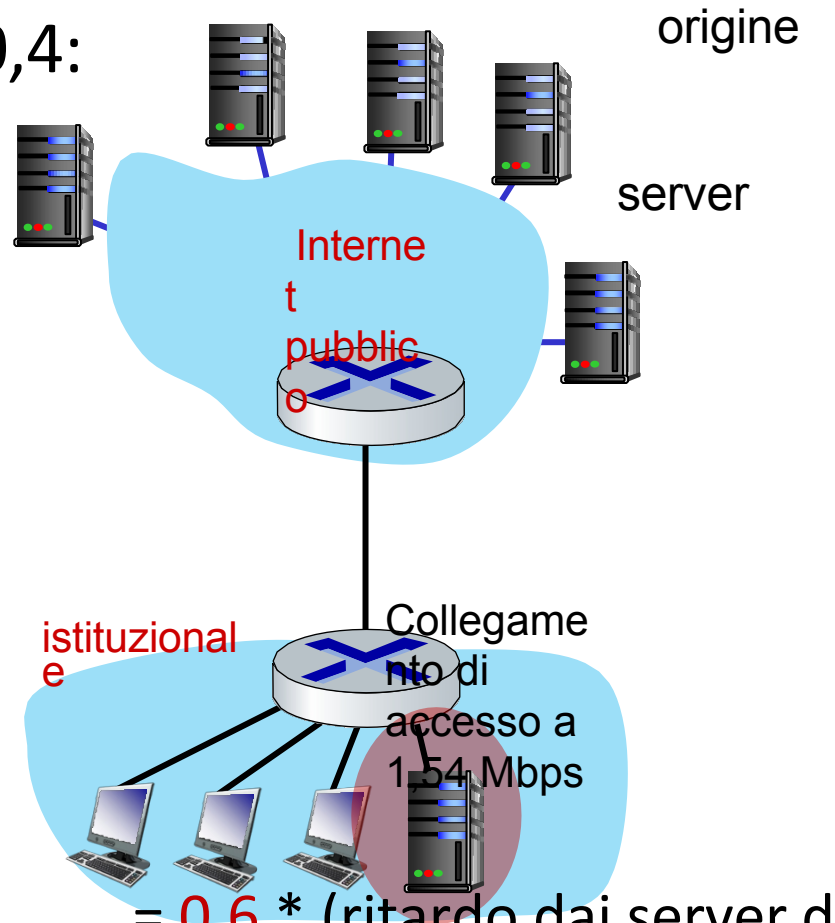


cache web locale

Calcolo dell'utilizzo del collegamento di accesso, ritardo finale con la cache:

supponiamo che il tasso di hit della cache sia 0,4:

- 40% di richieste servite dalla cache, con un basso (msec) ritardo
- 60% di richieste soddisfatte all'origine
  - tasso ai browser attraverso il collegamento di accesso
    = 0,6 * 1,50 Mbps = .9 Mbps
  - utilizzo dei collegamenti di accesso = 0,9/1,54 = .58 significa basso (msec) ritardo di accodamento sul collegamento di accesso
- ritardo medio finale:



= 0,6 * (ritardo dai server di

origine)

+ 0,4 * (ritardo quando viene soddisfatta la cache)

= 0,6 (2,01) + 0,4 (~msec) = ~ 1,2 secondi

rete

LAN a 1 Gbps

cache web locale

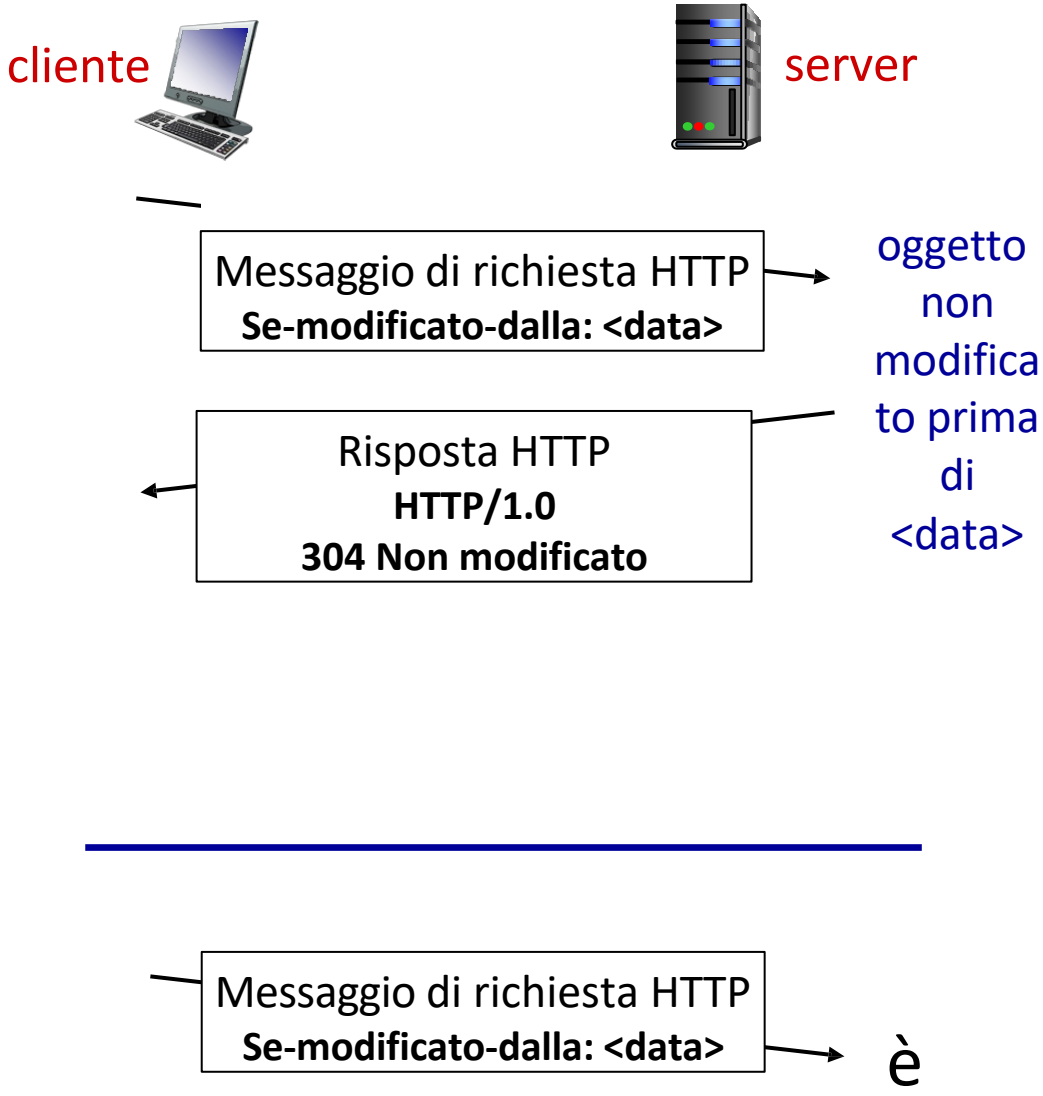
ritardo medio finale inferiore rispetto al collegamento a 154 Mbps (e anche più economico!)

Caching del browser: GET condizionale

Obiettivo: non inviare l'oggetto se il browser ha una versione aggiornata in cache

- nessun ritardo di trasmissione dell'oggetto (o utilizzo di risorse di rete)

- *client:* specificare la data del browser copia in cache nella richiesta HTTP
  Se-modificato-dalla: <data>
- *server: la* risposta non contiene se la copia in cache del browser è



aggiornata:

HTTP/1.0 304 Non modificato

HTTP/2

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

HTTP1.1: introduzione di GET multipli e in pipeline su una singola connessione TCP

- il server risponde *in ordine* (FCFS: first-come-first-served scheduling) alle richieste GET
- con FCFS, gli oggetti piccoli possono dover attendere la trasmissione (blocco della linea di testa (HOL)) dietro agli oggetti grandi
- il recupero delle perdite (ritrasmissione dei segmenti TCP

persi) blocca la trasmissione degli oggetti

HTTP/2

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

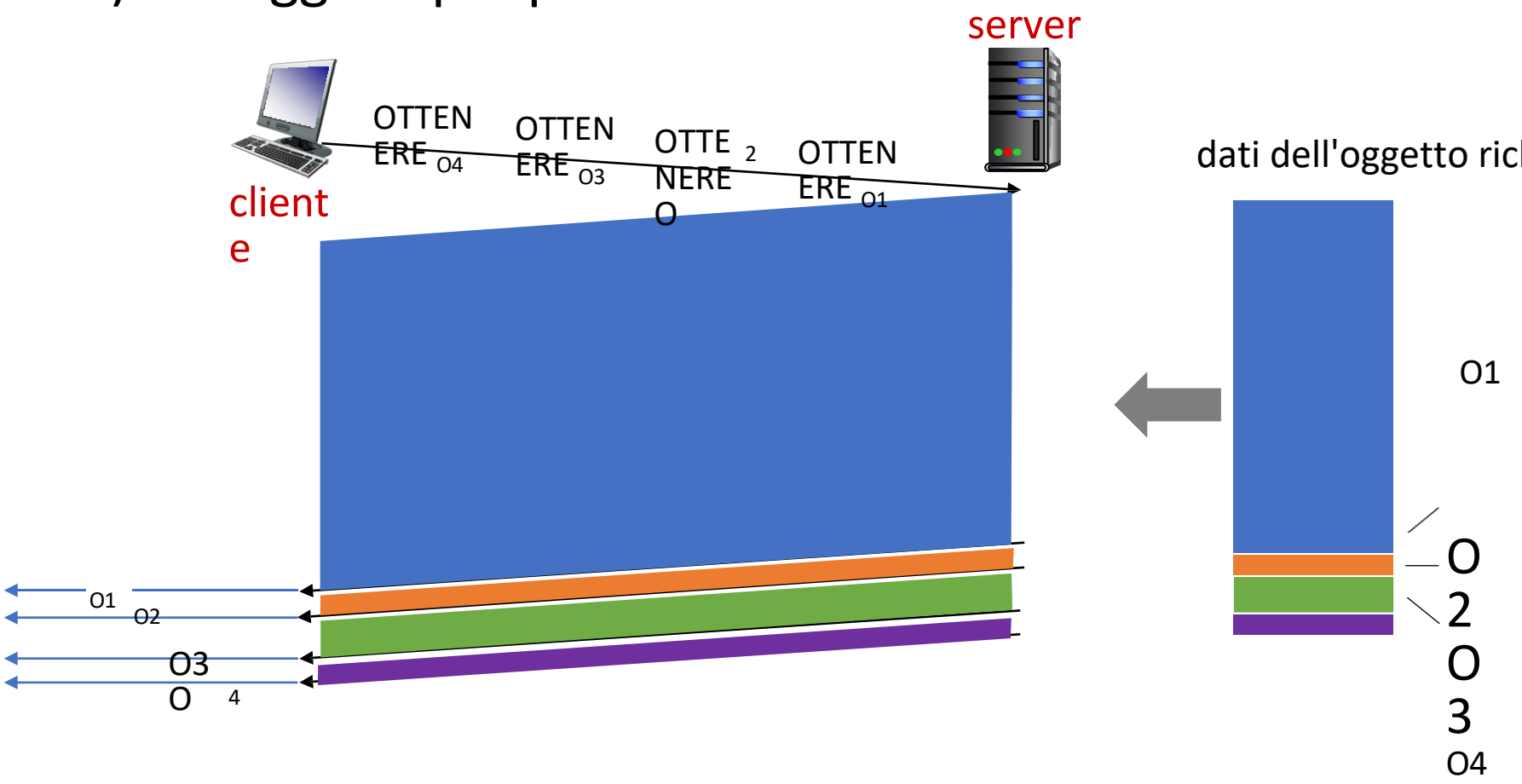
HTTP/2: [RFC 7540, 2015] maggiore flessibilità del *server* nell'invio di oggetti al client:

- metodi, codici di stato e la maggior parte dei campi di intestazione invariati rispetto a HTTP 1.1
- ordine di trasmissione degli oggetti richiesti in base alla priorità degli oggetti specificata dal cliente (non necessariamente FCFS)
- *spingere gli* oggetti non richiesti al client
- dividere gli oggetti in fotogrammi, programmare i fotogrammi per

attenuare il blocco HOL

HTTP/2: mitigazione del blocco HOL

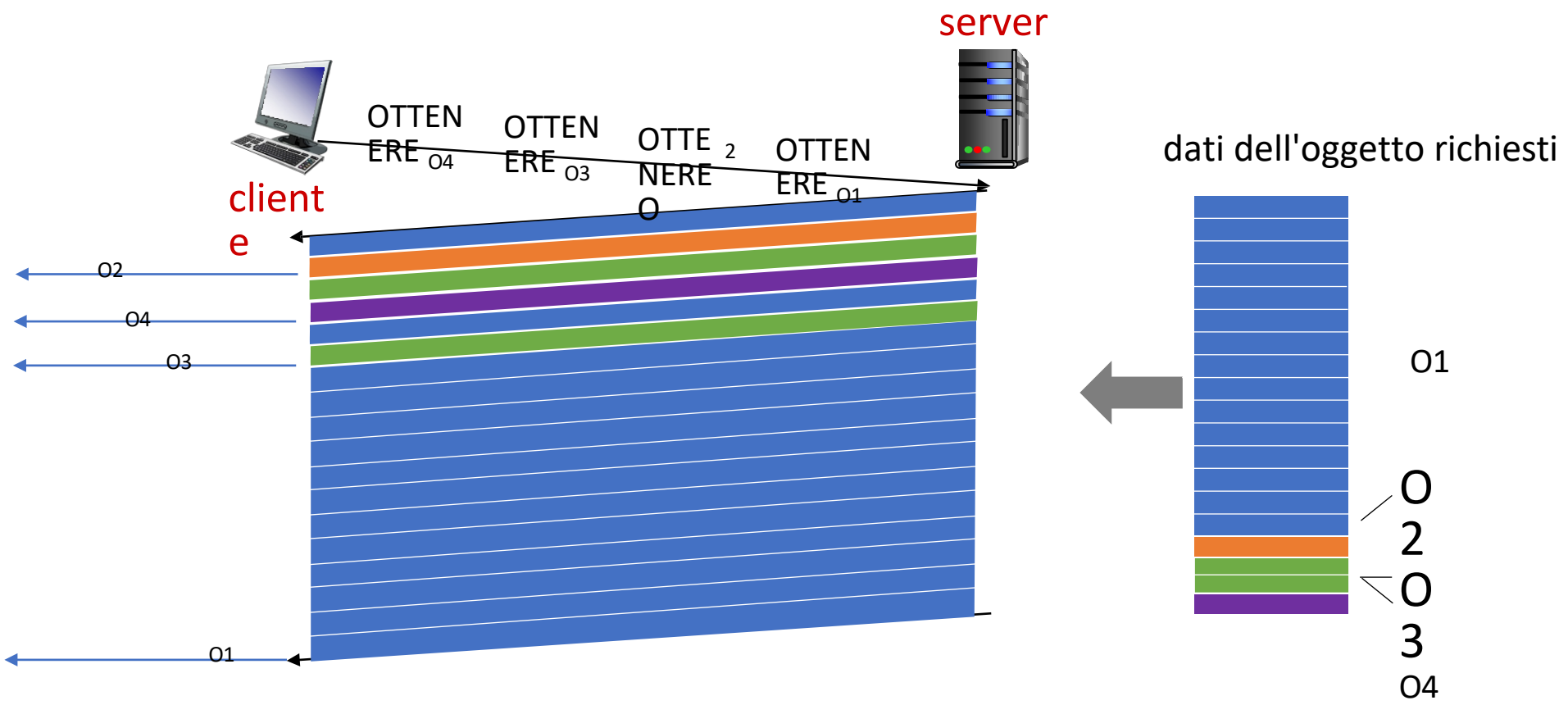
HTTP 1.1: il client richiede 1 oggetto di grandi dimensioni (ad esempio, un file video) e 3 oggetti più piccoli.



gli oggetti vengono consegnati nell'ordine richiesto: O_2, O_3, O_4 aspettano dietro O_1

HTTP/2: mitigazione del blocco HOL

HTTP/2: oggetti divisi in frame, trasmissione dei frame interleaved



O_2, O_3, O_4 *sono stati consegnati rapidamente,* O_1 *con un leggero ritardo.*

Da HTTP/2 a HTTP/3

HTTP/2 su 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 degli errori e della congestione per oggetto (più pipelining) su UDP
  - Ulteriori informazioni su HTTP/3 nel livello di trasporto

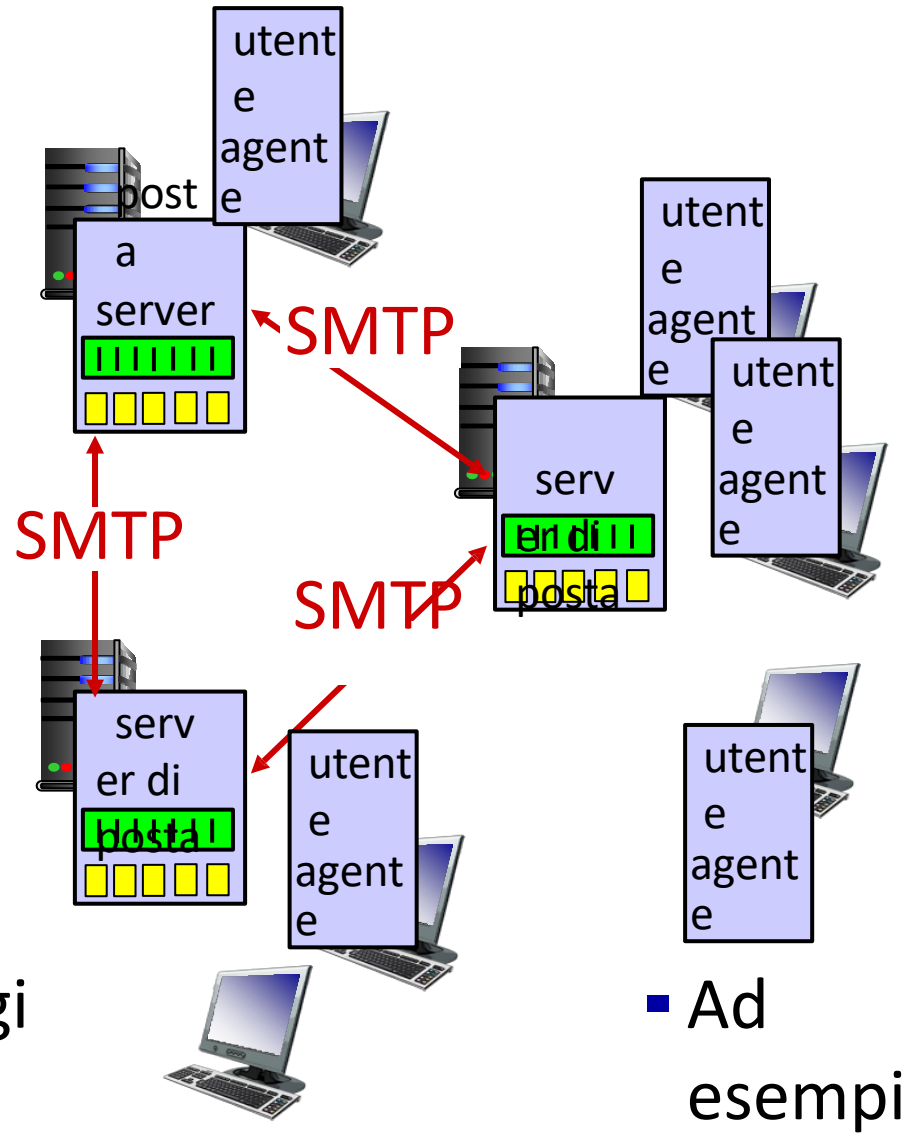
Livello applicativo: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Il sistema dei nomi di dominio DNS
- Applicazioni P2P
- reti di streaming video e distribuzione di contenuti
- programmazione di socket con UDP e TCP

E-mail

Tre componenti principali:

- agenti utente
- server di posta
- protocollo di trasferimento semplice della posta: SMTP



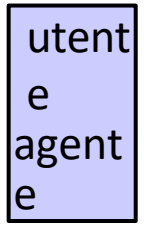
Agente utente

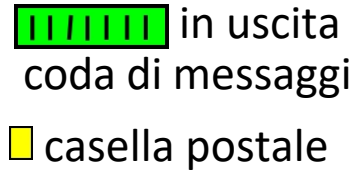
- alias "lettore di posta".
- comporre, modificare e leggere i messaggi di posta elettronica
- Ad esempio,

Outlook, client di posta per iPhone.

- messaggi in uscita e in entrata memorizzati su server

dell'utente





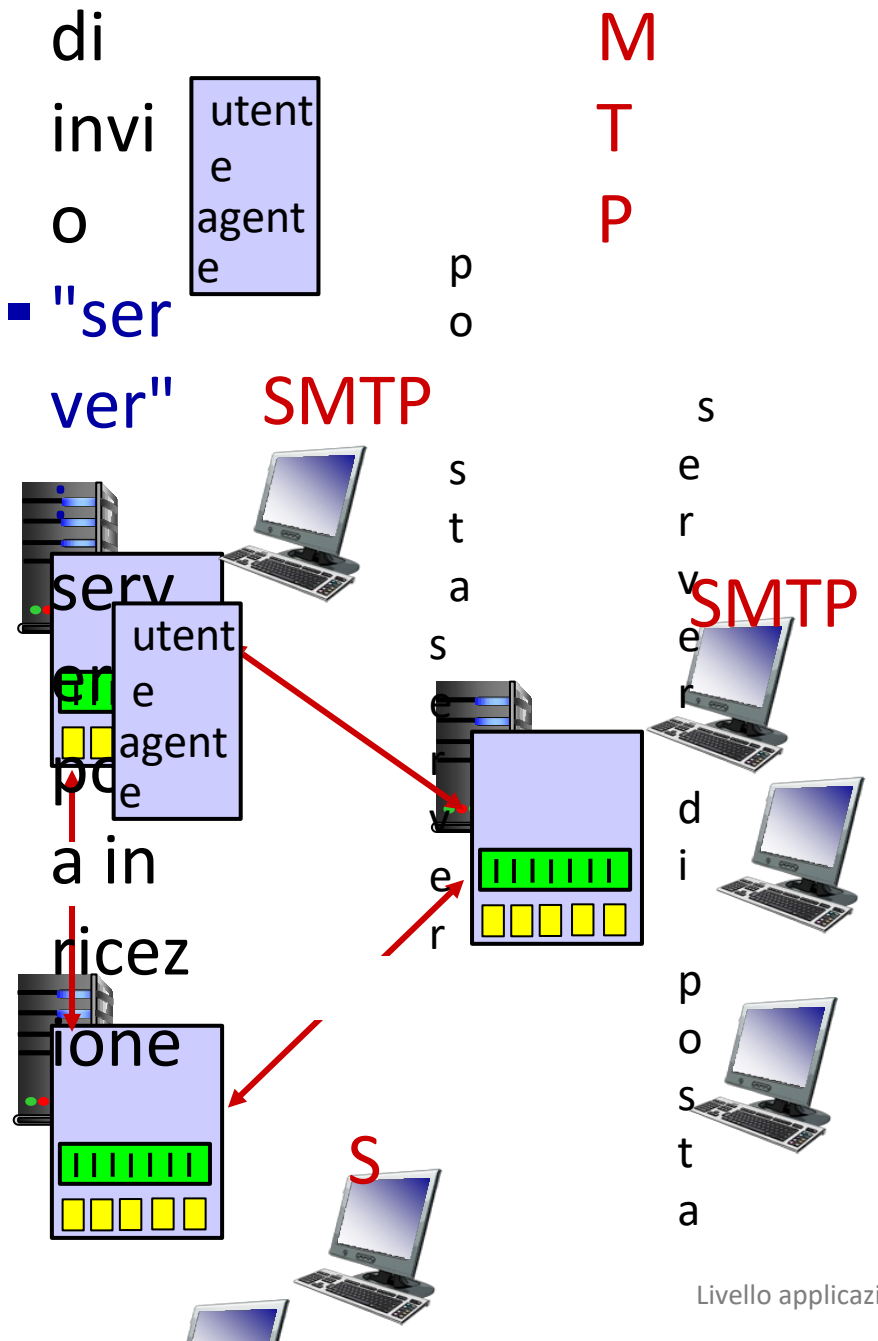
Posta elettronica: server di posta

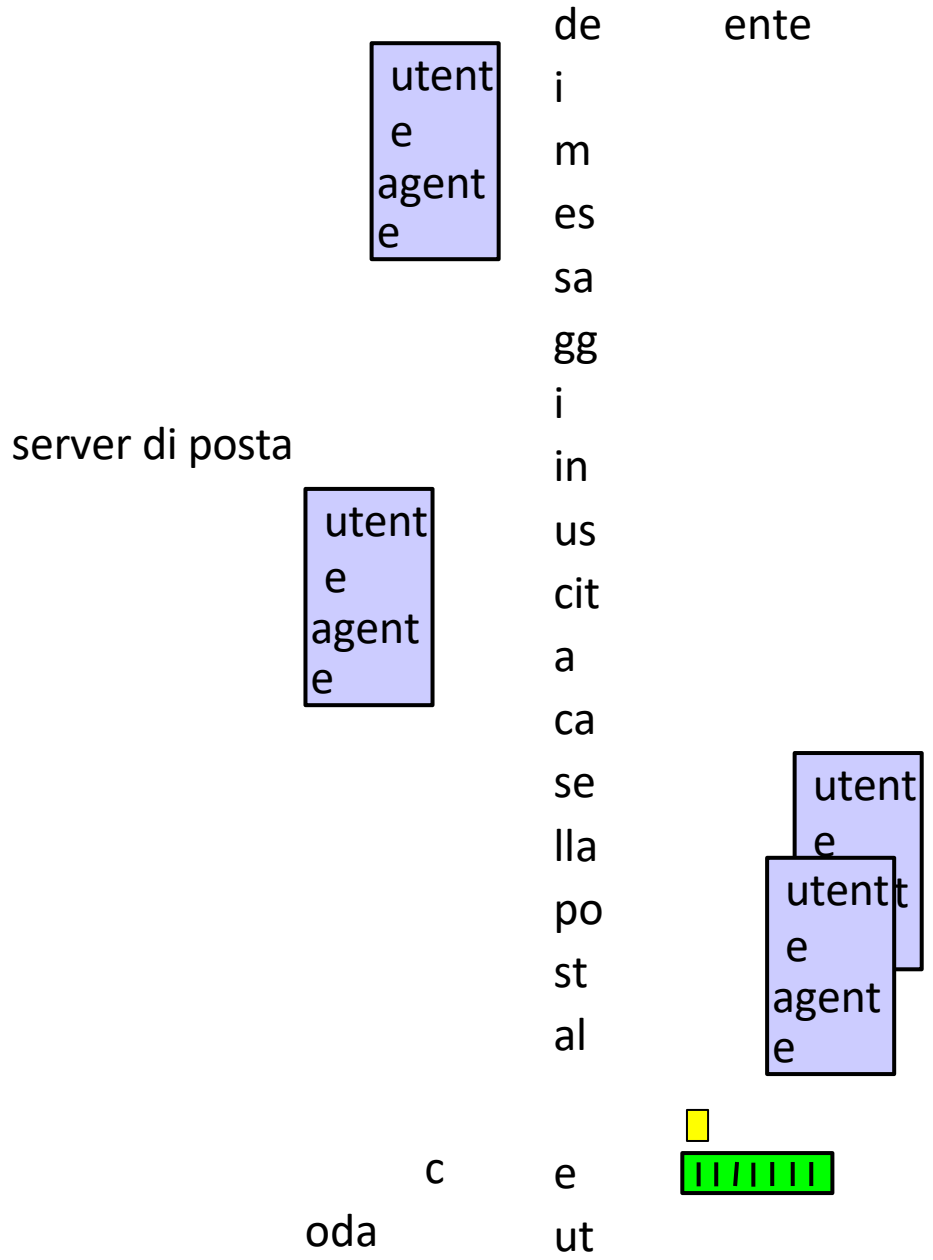
server di posta:

- *La mailbox* contiene i messaggi in arrivo per l'utente
- *coda di messaggi* di posta in uscita (da inviare)

Protocollo SMTP tra server di posta per l'invio di messaggi di posta elettronica

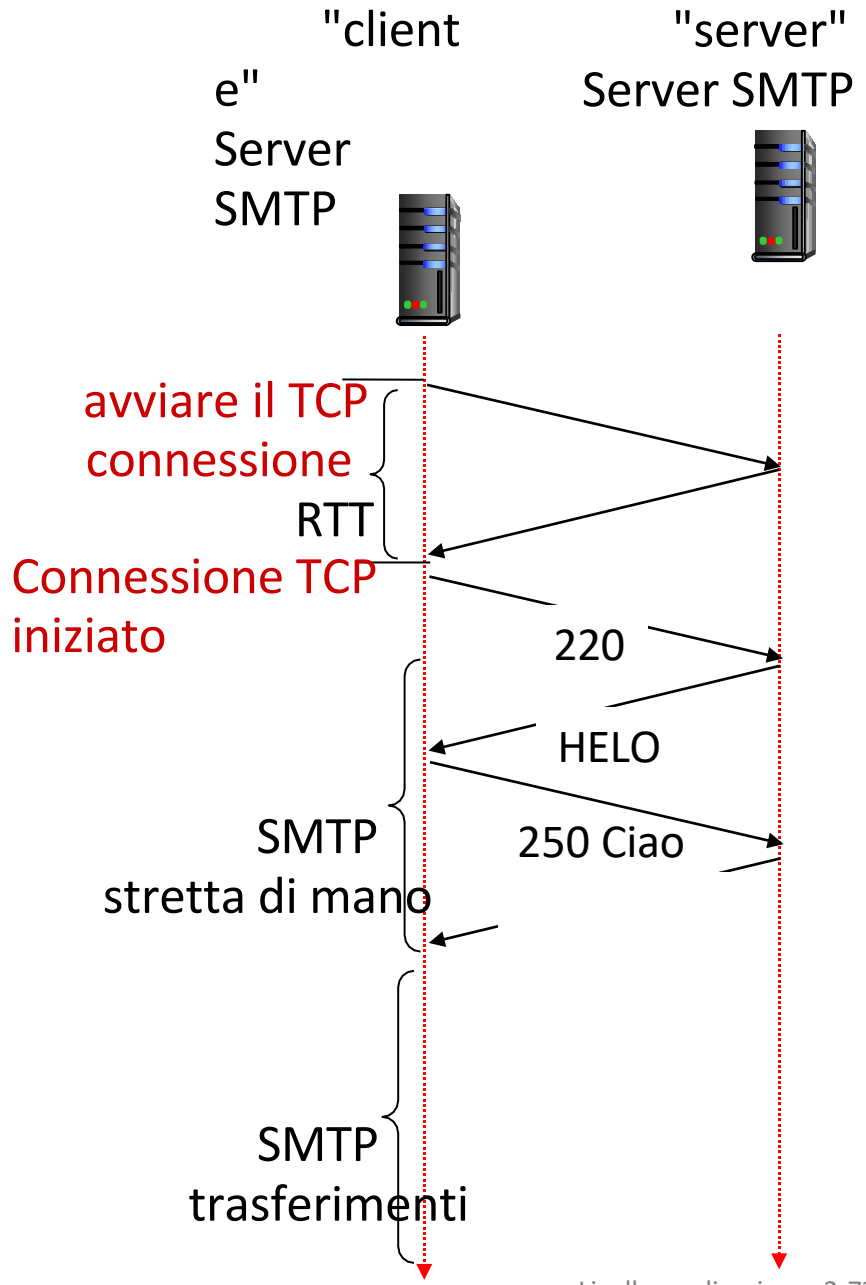
- client: server di posta elettronica




de ente
i
m
es
sa
gg
i
in
us
cit
a
ca
se
lla
po
st
al
utent
e
agent
e
server di posta
utent
e
agent
e
utent
e
utentt
e
agent
e
c e
oda ut

SMTP RFC (5321)

- utilizza TCP per trasferire in modo affidabile i messaggi di posta elettronica dal client (server di posta che avvia la connessione) al server, porta 25
  - trasferimento diretto: server di invio (che agisce come un client) al server ricevente
- tre fasi di trasferimento
  - SMTP handshaking (saluto)
  - Trasferimento di messaggi SMTP
  - Chiusura SMTP
- interazione comando/risposta (come HTTP)
  - comandi: Testo ASCII



- risposta: codice di stato e frase

tempo

Scenario: Alice invia un'e-mail a Bob

1) Alice utilizza UA per comporre le e-mail
messaggio "a" bob@someschool.edu

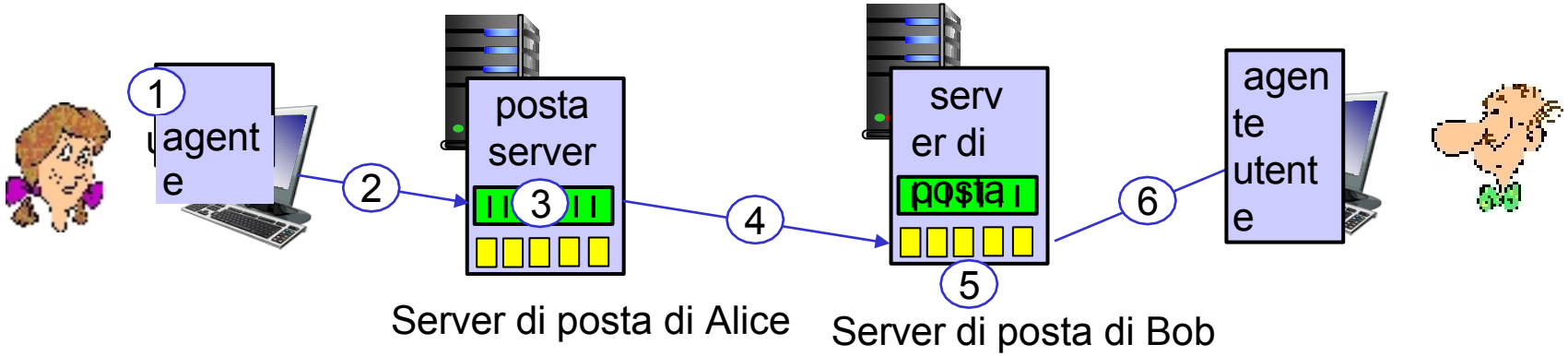
2) L'UA di Alice invia il messaggio al suo server di posta elettronica utilizzando SMTP; il messaggio viene inserito nella coda dei messaggi.

3) lato client di SMTP al server di posta apre la connessione TCP con il server di posta di Bob

4) Il client SMTP invia il messaggio di Alice attraverso la connessione TCP.

5) Il server di posta di Bob inserisce il messaggio nella casella di posta di Bob

6) Bob invoca il suo interprete per leggere il messaggio


1
agent
e
2
posta
server
3
4
serv
er di
posta
5
6
agen
te
utent
e
Server di posta di Alice
Server di posta di Bob

Esempio di interazione SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Ciao crepes.fr, piacere  per incontrarvi
C: MAIL DA: <alice@crepes.fr>
S: 250 alice@crepes.fr...          ok
       Mittente
C: RCPT A: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Destinatario
ok C: DATI
S: 354 Inserire la posta, terminare con "." su una
       riga a sé stante
C: Farti piace il ketchup?
   e
C: Com sui sottaceti?
   e
C: .
```

```
S: 250 Messaggio accettato per la consegna
C: CHIUDERE
S: 221 hamburger.edu chiude la connessione
```

SMTP: osservazioni

confronto con l'HTTP:

- HTTP: client pull
- SMTP: client push
- entrambi hanno un'interazione comando/risposta ASCII, codici di stato
- HTTP: ogni oggetto incapsulato nel proprio messaggio di risposta
- SMTP: invio di più oggetti in un messaggio multiparte
- SMTP utilizza connessioni persistenti
- SMTP richiede che il messaggio (intestazione e corpo) sia in formato ASCII a 7 bit.
- Il server SMTP utilizza CRLF.CRLF per determinare la fine del messaggio

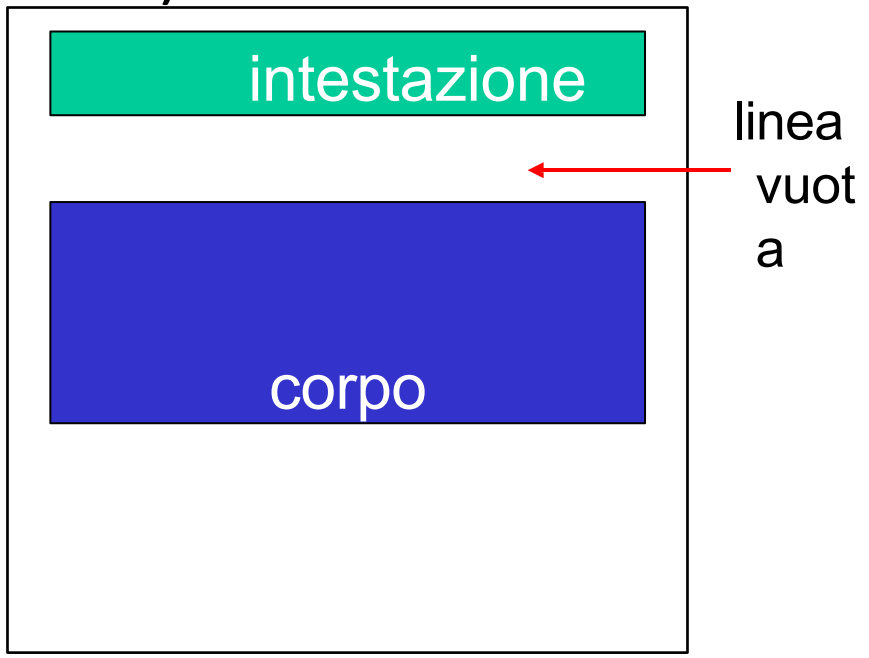
Formato del messaggio di posta

SMTP: protocollo per lo scambio di messaggi di posta elettronica, definito in RFC 5321 (come RFC 7231 definisce HTTP).

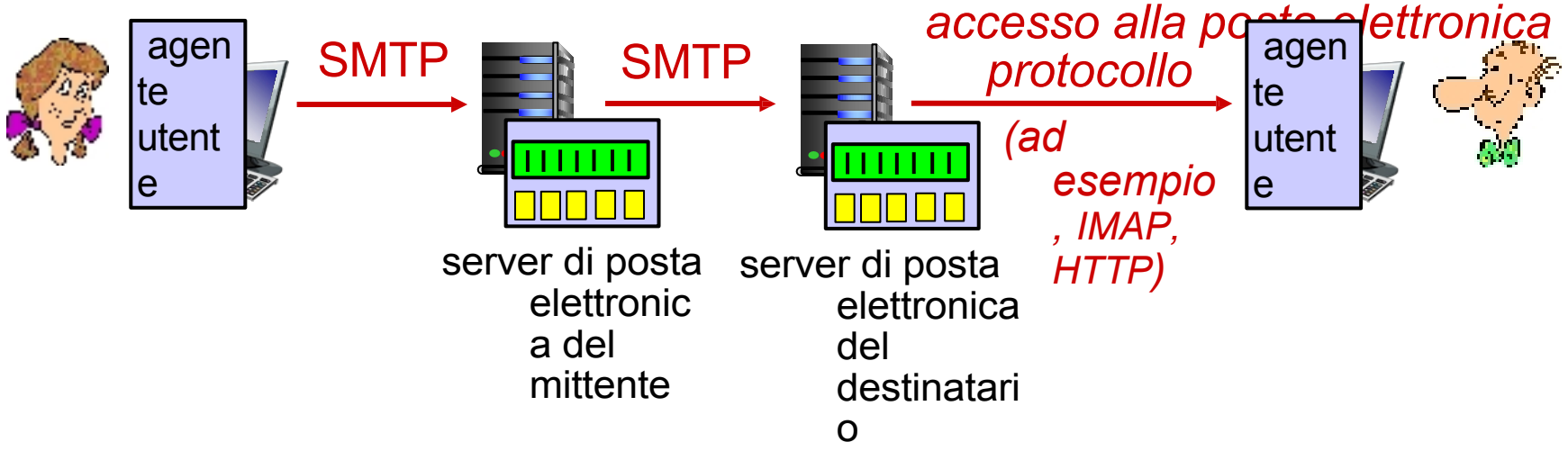
L'RFC 2822 definisce la *sintassi* del messaggio di posta elettronica (come l'HTML definisce la sintassi dei documenti web).

- righe di intestazione, ad es,
  - A:
  - Da:
  - Oggetto:

  queste righe, all'interno del corpo del messaggio di posta elettronica diverso dai comandi SMTP MAIL FROM:, RCPT TO:!
- Corpo: il "messaggio", solo con caratteri ASCII.



Recupero delle e-mail: protocolli di accesso alla posta elettronica



- SMTP: consegna/memorizzazione dei messaggi di posta elettronica al server del destinatario
- protocollo di accesso alla posta: recupero 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, ecc. forniscono un'interfaccia basata sul web in aggiunta a STMP (per l'invio) e IMAP (o POP) per il recupero dei messaggi di posta elettronica.

Strato applicativo: Panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Il sistema dei nomi di dominio DNS
- Applicazioni P2P
- reti di streaming video e distribuzione di contenuti
- programmazione di socket con UDP e TCP

DNS: Sistema dei nomi di dominio

persone: molti identificatori:

- SSN, nome, numero di passaporto

Host Internet, router:

- Indirizzo IP (32 bit) - utilizzato per indirizzare i datagrammi
- "nome", ad esempio, cs.umass.edu. utilizzato dagli esseri umani

D: come mappare tra indirizzo IP e nome e viceversa?

Sistema dei nomi di dominio (DNS):

- *database distribuito* implementato in una gerarchia di numerosi *server di nomi*
- *protocollo di livello applicativo:* host, server DNS comunicano per *risolvere i* nomi (traduzione indirizzo/nome)
  - *nota:* funzione centrale di Internet, implementata come protocollo di livello applicativo
  - complessità ai "margini" della rete

DNS: servizi, struttura

Servizi DNS:

- traduzione da nome host a indirizzo IP
- aliasing dell'host
  - canonici, nomi di alias
- aliasing del server di posta
- distribuzione del carico
  - server web replicati: molti indirizzi IP corrispondono a un nome

D: Perché non centralizzare il DNS?

- singolo punto di guasto
- volume di traffico
- database centralizzato distante
- manutenzione

R: non è scalabile!

- Solo i server DNS di Comcast: 600B query DNS al giorno
- Solo i server DNS Akamai:

2,2T query DNS al giorno

Pensare al DNS

un enorme database distribuito:

- ~ miliardi di record, ognuno semplice

gestisce molti *trilioni* di interrogazioni al giorno:

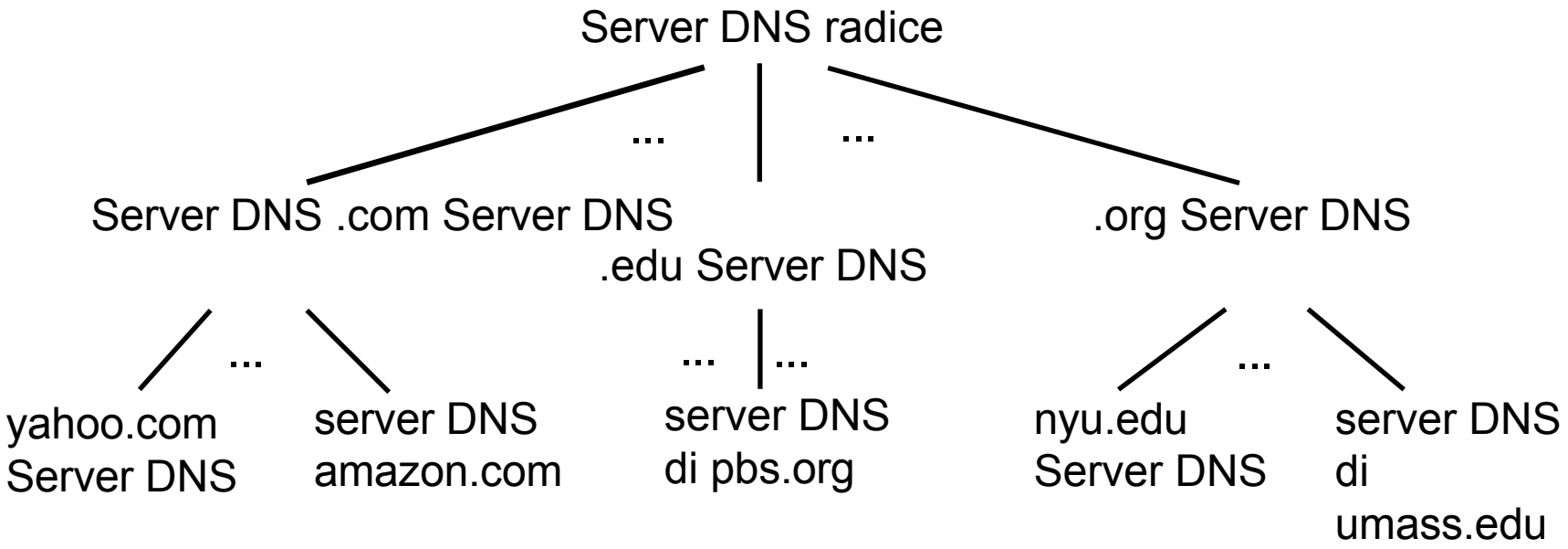
- *molte* più letture che scritture
- *Le prestazioni sono importanti:* quasi tutte le transazioni Internet interagiscono con il DNS - i msec contano!

organizzativamente e fisicamente decentralizzato:

- milioni di organizzazioni diverse responsabili dei loro archivi

"A prova di proiettile": affidabilità, sicurezza

DNS: un database distribuito e gerarchico



Radice

Dominio di primo livello

Autorevole

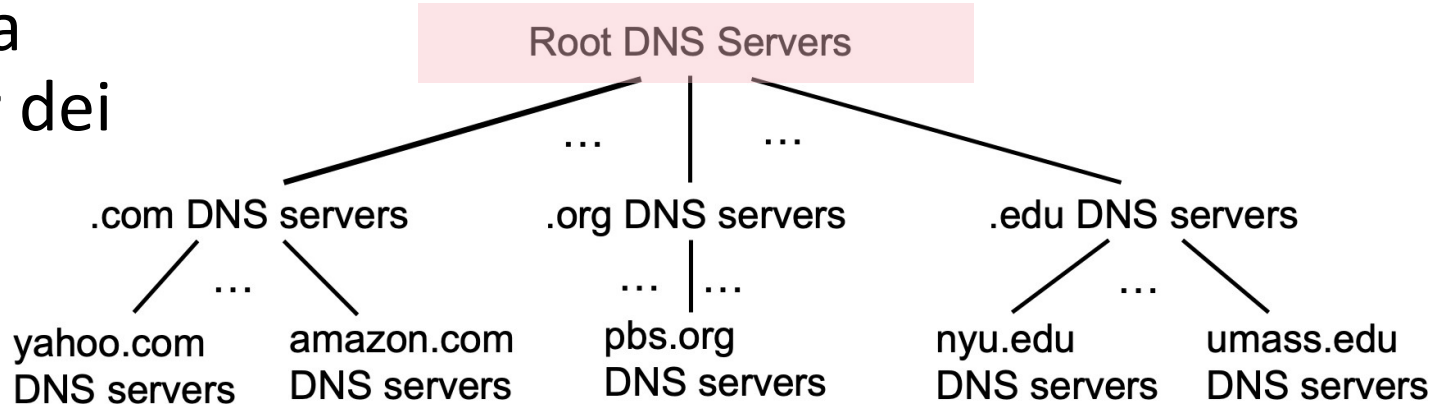
Il cliente vuole un indirizzo IP per www.amazon.com; 1st approssimazione:

- il client interroga il server root per trovare il server DNS .com
- il client interroga il server DNS .com per ottenere il server DNS amazon.com

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

DNS: server dei nomi radice

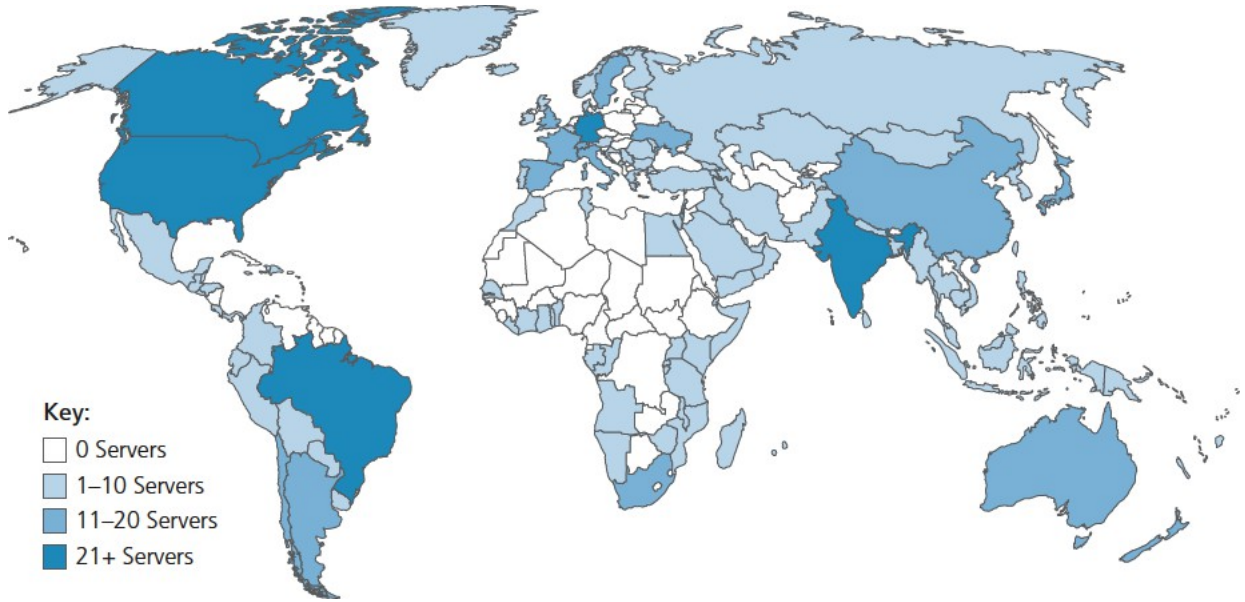
- ufficiale, contatto di ultima istanza da parte dei server dei nomi che non riescono a risolvere il nome



DNS: server dei nomi radice

- ufficiale, contatto di ultima istanza da parte dei server dei nomi che non riescono a risolvere il nome
- funzione Internet *incredibilmente importante*
  - Internet non potrebbe funzionare senza!
  - DNSSEC - fornisce sicurezza (autenticazione, integrità del messaggio)
- L'ICANN (Internet Corporation for Assigned Names and Numbers) gestisce i domini DNS radice

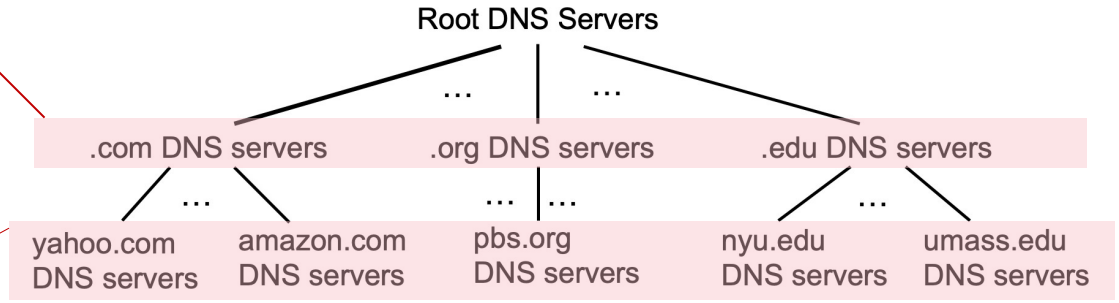
13 nomi di radici logiche "server" in tutto il mondo ogni "server" replicato
molte volte (~200 server negli Stati Uniti)



Dominio di primo livello e server autoritativi

Server dei domini di primo livello (TLD):

- responsabile di .com, .org, .net, .edu, .aero, .jobs, .museums e di tutti i livelli superiori. domini nazionali, ad esempio: .cn, .uk, .fr, .ca, .jp
- Network Solutions: registro autorevole per i TLD .com e .net
- Educause: TLD .edu



server DNS autorevoli:

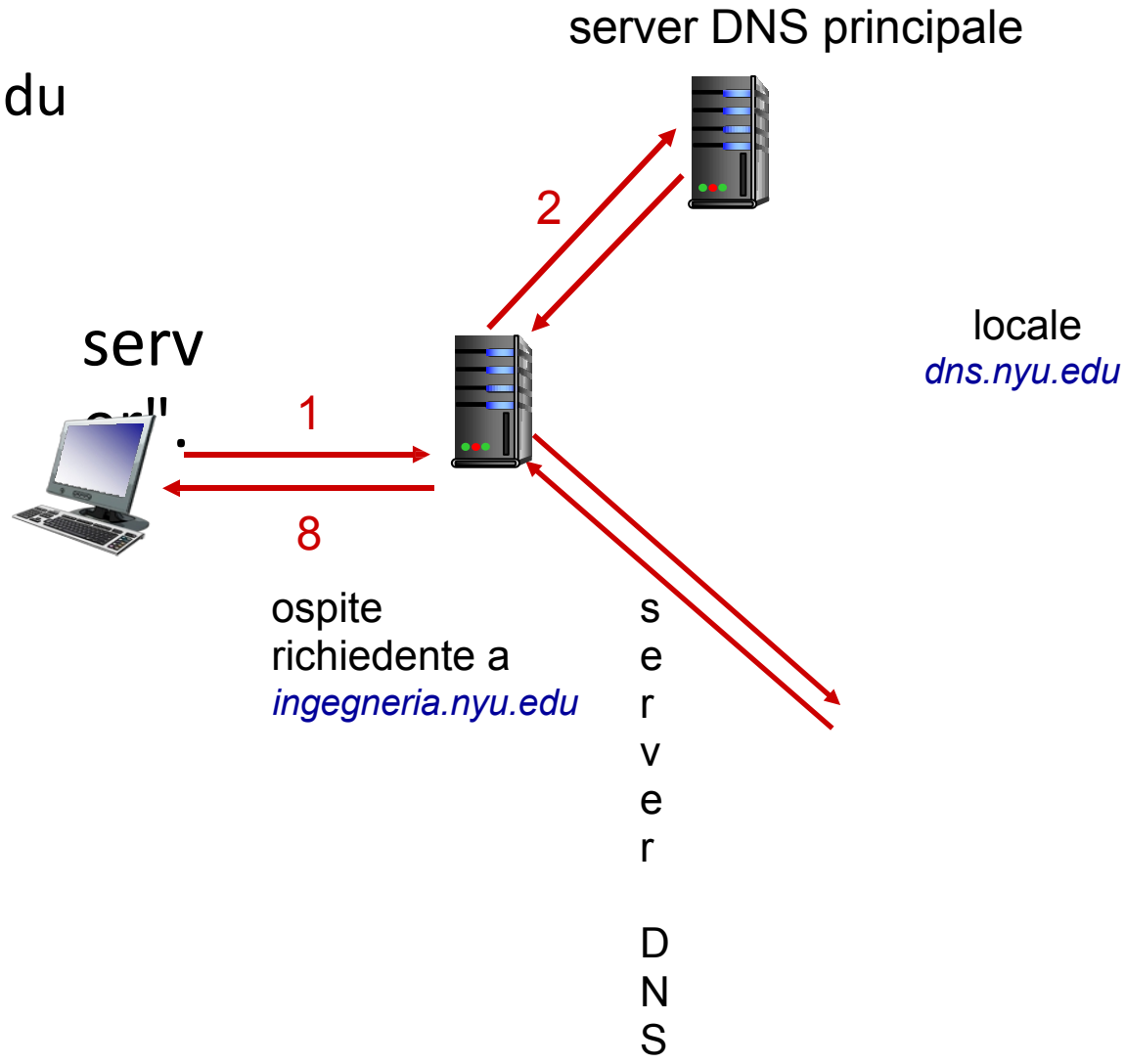
- server DNS dell'organizzazione, che fornisce il nome host autorevole per l'IP mappature per gli host nominati dell'organizzazione
- può essere mantenuto dall'organizzazione o dal fornitore di servizi

Server dei nomi DNS locali

- Quando l'host effettua una query DNS, questa viene inviata al suo server DNS *locale.*
  - Il server DNS locale restituisce la risposta, rispondendo:
    - dalla sua cache locale di coppie di traduzione nome-indirizzo recenti (forse non aggiornate!).
    - inoltrare la richiesta nella gerarchia DNS per la risoluzione.
  - Ogni ISP dispone di un server DNS locale; per trovare il vostro:
    - MacOS: `% scutil --dns`
    - Windows: `>ipconfig /all`
- il server DNS locale non appartiene strettamente alla gerarchia

Risoluzione dei nomi DNS: interrogazione iterata

Esempio: host at engineering.nyu.edu vuole l'indirizzo IP per gaia.cs.umass.edu

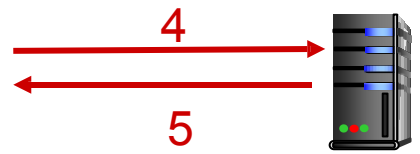


Query iterata:

- il server contattato risponde con il nome del server da contattare
- "Non conosco questo nome,
  ma chiedete a questo

3

Server DNS
TLD



gai
a.c
s.u
ma
ss.
ed
u

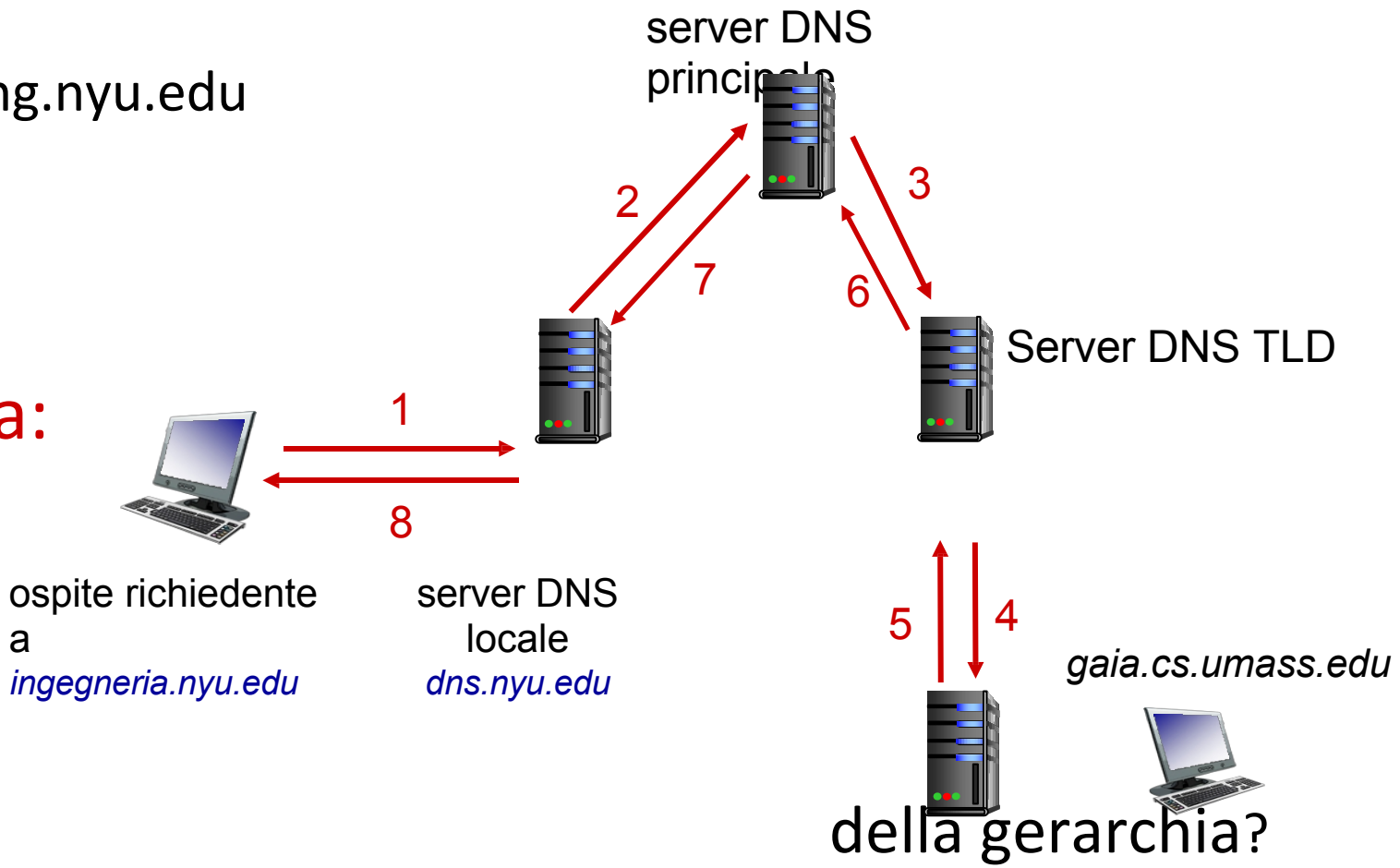
7 6

Risoluzione dei nomi DNS: interrogazione ricorsiva

Esempio: host at engineering.nyu.edu vuole l'indirizzo IP per gaia.cs.umass.edu

Interrogazione ricorsiva:

- mette il peso del nome risoluzione sul server dei nomi contattato
- carico pesante ai livelli superiori



server DNS autoritario
dns.cs.umass.edu

Caching delle informazioni DNS

- una volta che (qualsiasi) server dei nomi apprende la mappatura, la *memorizza nella* cache e la restituisce *immediatamente* in risposta a una query.
  - La cache migliora i tempi di risposta
  - le voci della cache vanno in timeout (scompaiono) dopo un certo tempo (TTL)
  - I server TLD sono in genere memorizzati nella cache dei server dei nomi locali.
- le voci nella cache potrebbero essere *obsolete*
  - se l'host nominato cambia indirizzo IP, potrebbe non essere conosciuto in Internet fino alla scadenza di tutti i TTL!

- *traduzione nome-indirizzo al massimo sforzo!*

Registrazioni DNS

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

Formato RR: (`nome, valore, tipo, ttl`)

tipo=A

- `il nome` è il nome dell'host
- `il valore` è l'indirizzo IP

tipo=NS

- `il nome` è il dominio (ad esempio, pippo.com)
- `il valore` è il nome host

del server dei nomi autoritario per questo dominio

tipo=NOME

- `Il nome` è un alias per un nome "canonico".
  (il vero) nome
- www.ibm.com è in realtà servereast.backup2.ibm.com
- `il valore` è il nome canonico

tipo=MX

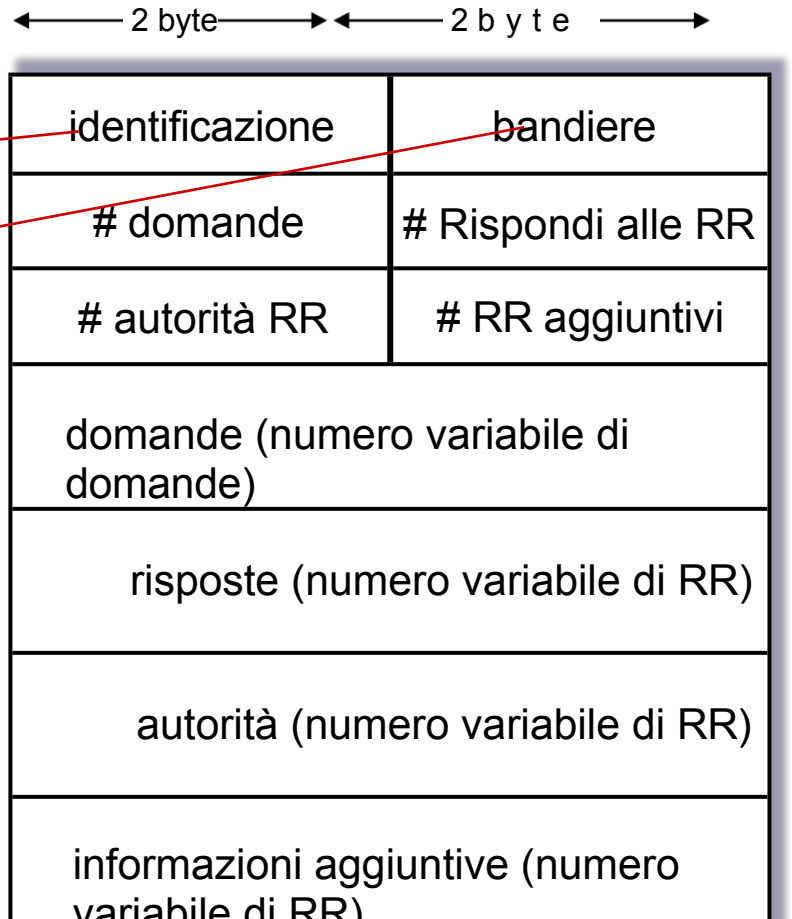
- `il valore` è il nome della posta SMTP server associato al `nome`

Messaggi del protocollo DNS

I messaggi di *richiesta* e *risposta* DNS hanno entrambi lo stesso *formato:*

intestazione del messaggio:

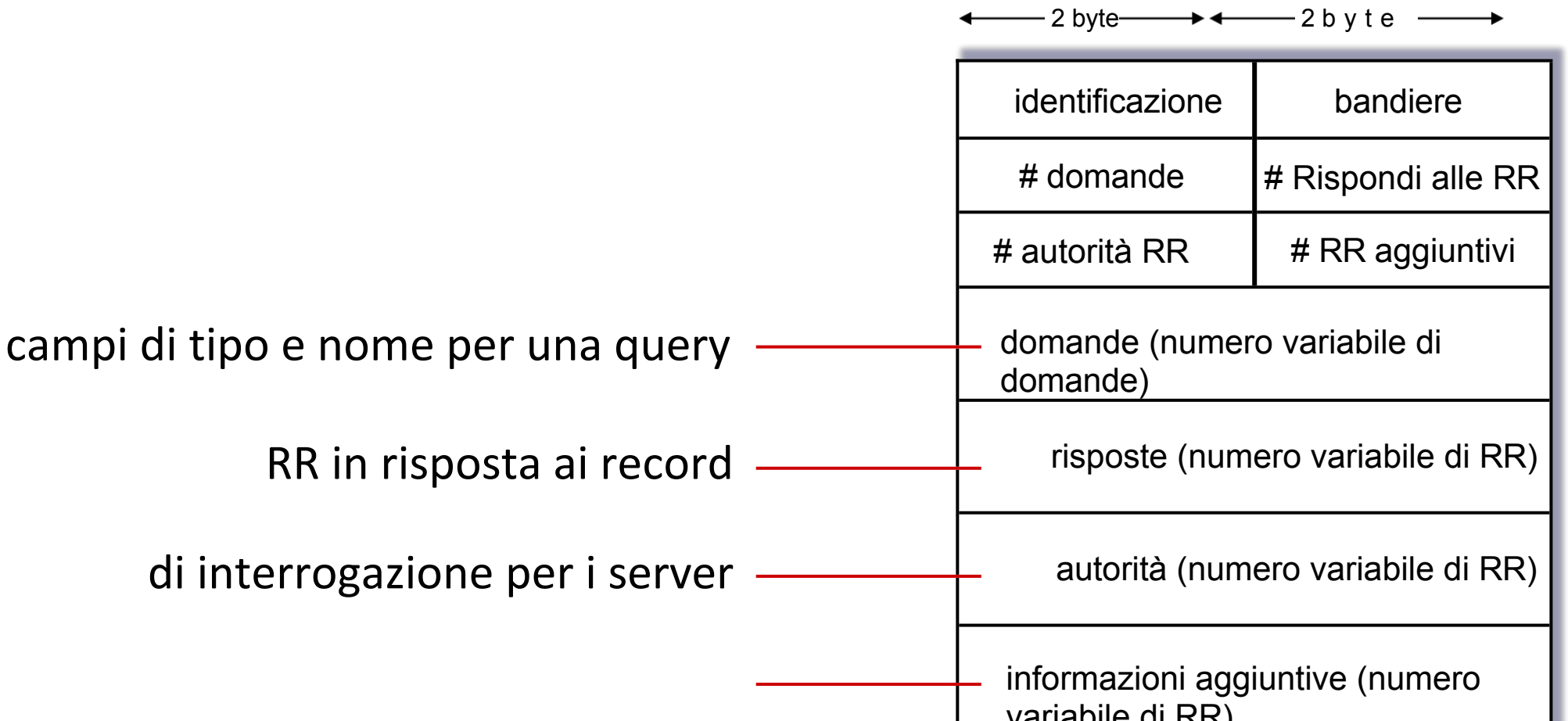
- identificazione: 16 bit # per l'interrogazione, la risposta all'interrogazione usa lo stesso #
- bandiere:
  - domanda o risposta
  - ricorsione desiderata
  - ricorsione disponibile



- la risposta è autorevole

Messaggi del protocollo DNS

I messaggi di *richiesta* e *risposta* DNS hanno entrambi lo stesso *formato:*



autoritari

ulteriori informazioni "utili" che possono essere utilizzato

Inserire le informazioni nel DNS

esempio: nuova startup "Utopia della rete".

- registrare il nome networkuptopia.com presso un *registrar DNS* (ad esempio, Network Solutions).
  - fornire i nomi, gli indirizzi IP dei server dei nomi autoritativi (primario e secondario)
  - Il registrar inserisce i RR NS e A nel server TLD .com:
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- creare un server autoritario locale con indirizzo IP `212.212.212.1`
  - record di tipo A per www.networkuptopia.com
  - tipo di record MX per networkutopia.com

Sicurezza DNS

Attacchi DDoS

- bombardare di traffico i server root
  - non ha avuto successo fino ad oggi
  - filtraggio del traffico
  - I server DNS locali memorizzano nella cache gli IP dei server TLD, consentendo di aggirare il server root.
- bombardare i server TLD
  - potenzialmente più pericoloso

Attacchi di spoofing

- intercettare le query DNS, restituendo risposte fasulle
  - Avvelenamento della cache DNS
  - RFC 4033: DNSSEC servizi di autenticazione

Livello applicazione: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Il sistema dei nomi di dominio DNS

- Applicazioni P2P
- reti di streaming video e distribuzione di contenuti
- programmazione di socket con UDP e TCP

Streaming video e CDN: contesto

- traffico video in streaming: il principale consumatore di larghezza di banda Internet
  - Netflix, YouTube, Amazon Prime: 80% del traffico ISP residenziale (2020)
- *sfida:* scala - come raggiungere ~1B utenti?
- *sfida:* eterogeneità
  - utenti diversi hanno capacità diverse (ad esempio, cablati o mobili; ricchi di larghezza di banda o poveri di larghezza di banda)

- *soluzione:* infrastruttura distribuita a livello di applicaz

Multimedia: video

- video: sequenza di immagini visualizzate a velocità costante
  - ad esempio, 24 immagini/sec
- immagine digitale: matrice di pixel
  - ogni pixel rappresentato da bit
- codifica: utilizzare la ridondanza *all'interno* e *tra le* immagini per ridurre il numero di bit utilizzati per la codifica dell'immagine
  - spaziale (all'interno dell'immagine)
  - temporale (da un'immagine a

Esempio di codifica spaziale: invece di inviare *N* valori dello stesso colore (tutti viola), inviare solo due valori: il valore del colore (*viola) e il numero di valori ripetuti (*N).

telaio *i*

pros
simo
)

esempio di codifica temporale: invece di inviare il fotogramma completo a i+1,

inviare solo le differenze dal fotogramma i

fotogramma *i+1*

Multimedia: video

- CBR: (constant bit rate): velocità di codifica video fissa
- VBR: (variable bit rate): la velocità di codifica video cambia al variare della quantità di codifica spaziale e temporale.
- esempi:
  - MPEG 1 (CD-ROM) 1,5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (spesso utilizzato in

Esempio di codifica spaziale: invece di inviare *N* valori dello stesso colore (tutti viola), inviare solo due valori: il valore del colore (*viola) e il numero di valori ripetuti (*N).

telaio *i*

esempio di codifica temporale: invece di inviare il fotogramma completo a i+1,

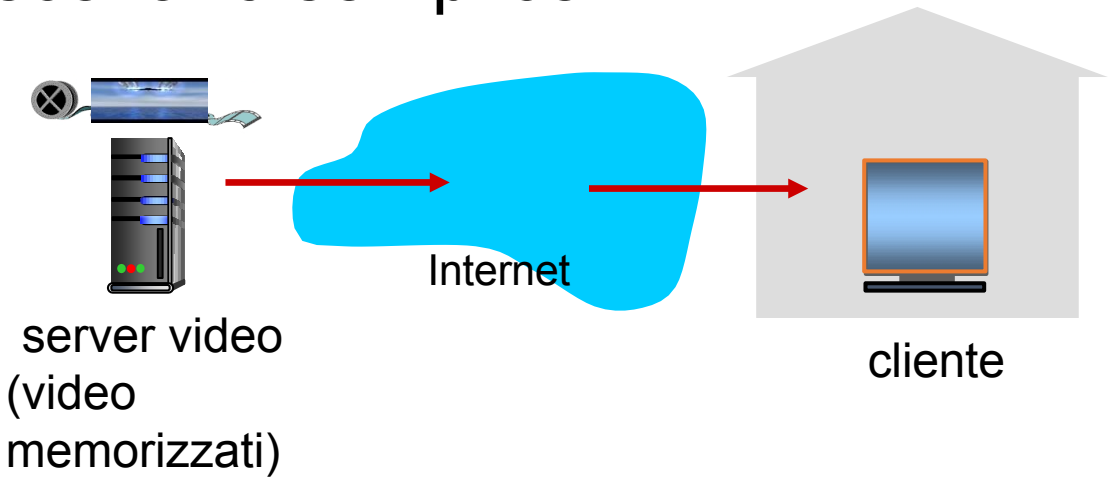
Internet, 64Kbps - 12 Mbps)

inviare solo le differenze
dal fotogramma i

fotogramma *i+1*

Streaming di video memorizzati

scenario semplice:

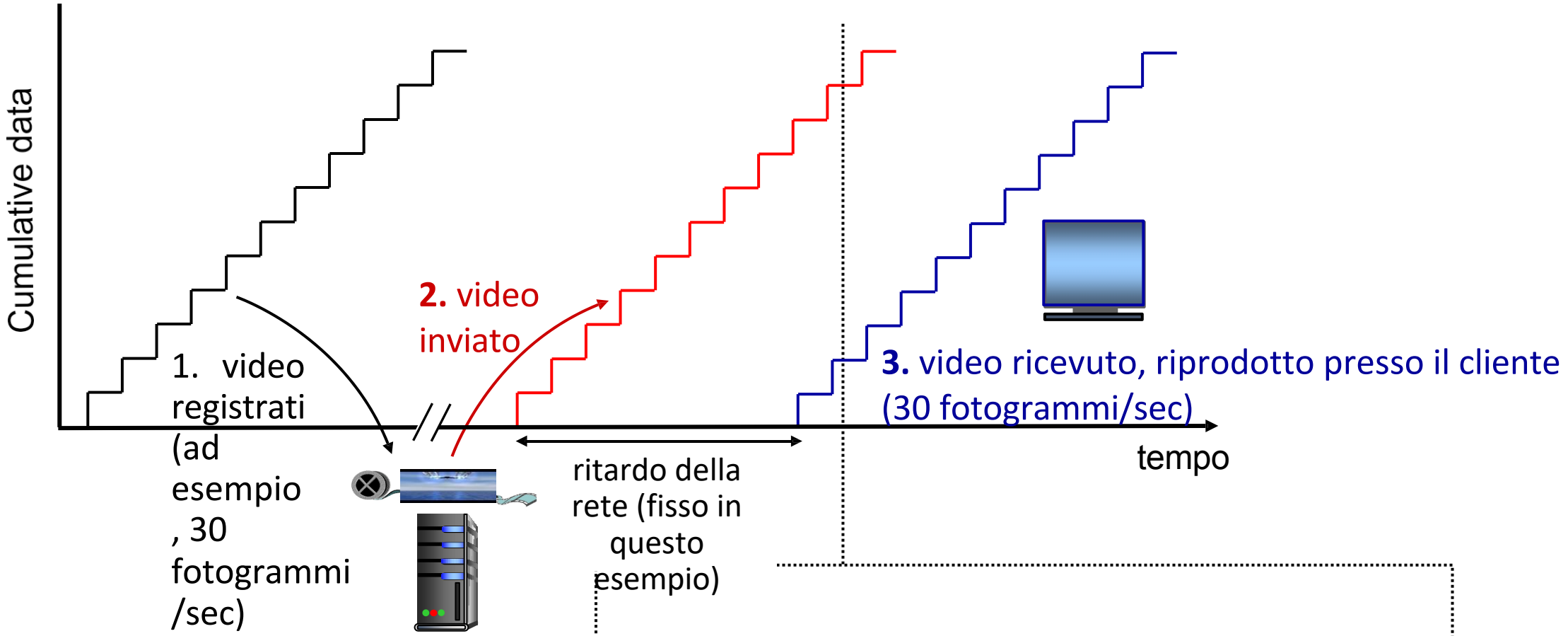


Sfide principali:

- la larghezza di banda da server a cliente *varia* nel tempo, con il variare dei livelli di congestione della rete (in house, rete di accesso, nucleo della rete, server video)
- La perdita di pacchetti, i ritardi dovuti alla congestione ritardano

la riproduzione o comportano una scarsa qualità video.

Streaming di video memorizzati

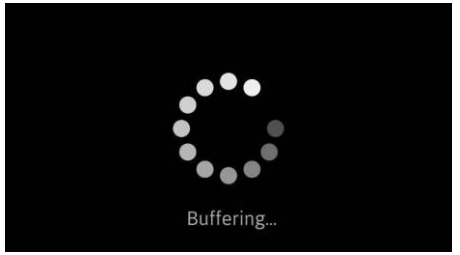


streaming: in questo momento, il client riproduce la parte iniziale del video, mentre il

server invia ancora la parte successiva del video

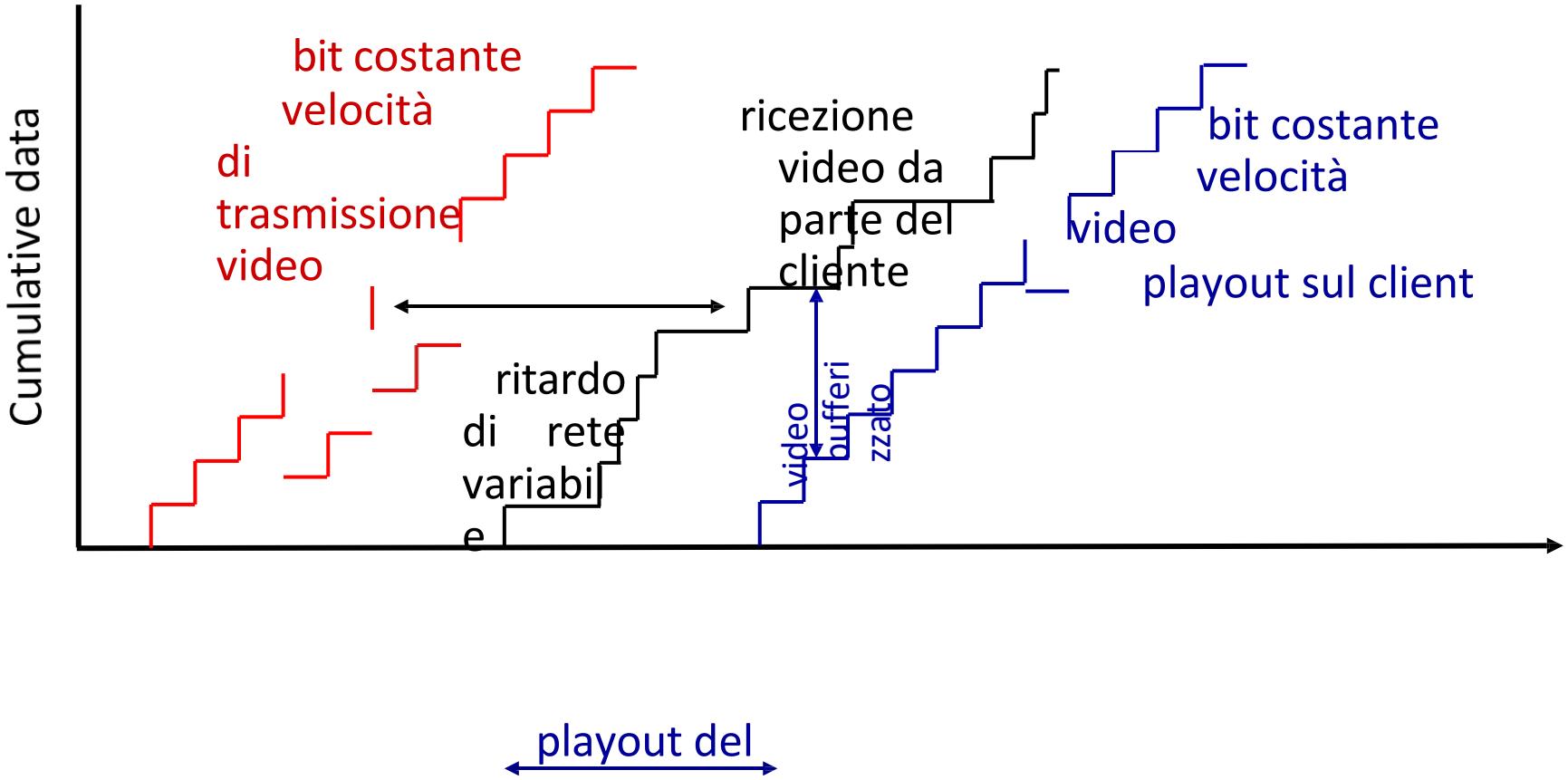
Streaming di video memorizzati: le sfide

- Vincolo di riproduzione continua: durante la riproduzione del video client, la tempistica di riproduzione deve corrispondere alla tempistica originale.
  - ... ma i ritardi della rete sono variabili (jitter), quindi sarà necessario un buffer lato client per soddisfare i vincoli di riproduzione continua.
- altre sfide:
  - interattività del cliente: pausa, avanzamento veloce, riavvolgimento, salto nel video



- i pacchetti video possono essere persi, ritrasmessi

Streaming di video memorizzati: buffering di playout



iente
ritardo

tempo

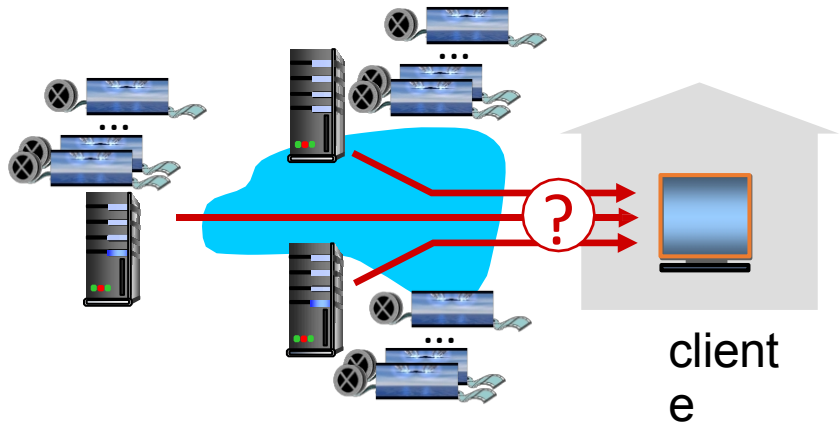
- *buffering lato client e ritardo di riproduzione:* compensazione del ritardo aggiunto dalla rete, jitter del ritardo

Streaming multimediale: DASH

Dinamico, adattivo
Streaming su HTTP

server:

- divide il file video in più parti
- ogni brano codificato a più velocità diverse
- codifiche di velocità diverse memorizzate in file diversi
- file replicati in vari nodi CDN
- *file manifest:* fornisce gli URL per i diversi chunk



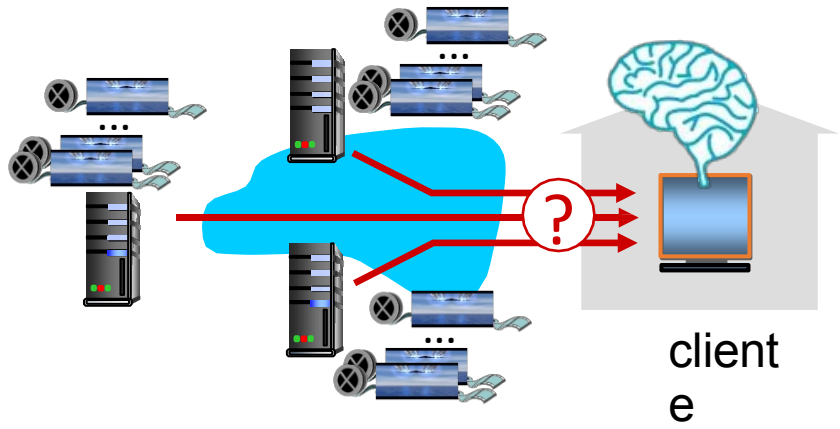
cliente:

- stima periodicamente la larghezza di banda da server a cliente
- consultazione del manifesto, richiesta di un pezzo alla volta

- sceglie la massima velocità di codifica sostenibile data la larghezza di banda corrente
- può scegliere velocità di codifica diverse in momenti diversi (a seconda della larghezza di banda disponibile in quel momento), e da server diversi

Streaming multimediale: DASH

- *"intelligenza"* al cliente: il cliente determina
  - *quando* richiedere il chunk (in modo che non si verifichi un buffer starvation o un overflow)
  - *quale velocità di codifica* richiedere (qualità più elevata quando è disponibile una maggiore larghezza di banda)
  - *dove* richiedere il chunk (può essere richiesto da un server URL che è "vicino" al client o che ha un'elevata larghezza di banda disponibile)



Streaming video = codifica + DASH + buffering di playout

Reti di distribuzione dei contenuti (CDN)

sfida: come trasmettere contenuti (selezionati tra milioni di video) a centinaia di migliaia di utenti *simultanei*?

- *opzione 1:* singolo, grande "mega-server"
  - singolo punto di guasto
  - punto di congestione della rete
  - percorso lungo (e possibilmente congestionato) verso i clienti lontani

....quasi semplicemente: questa soluzione *non è scalabile*

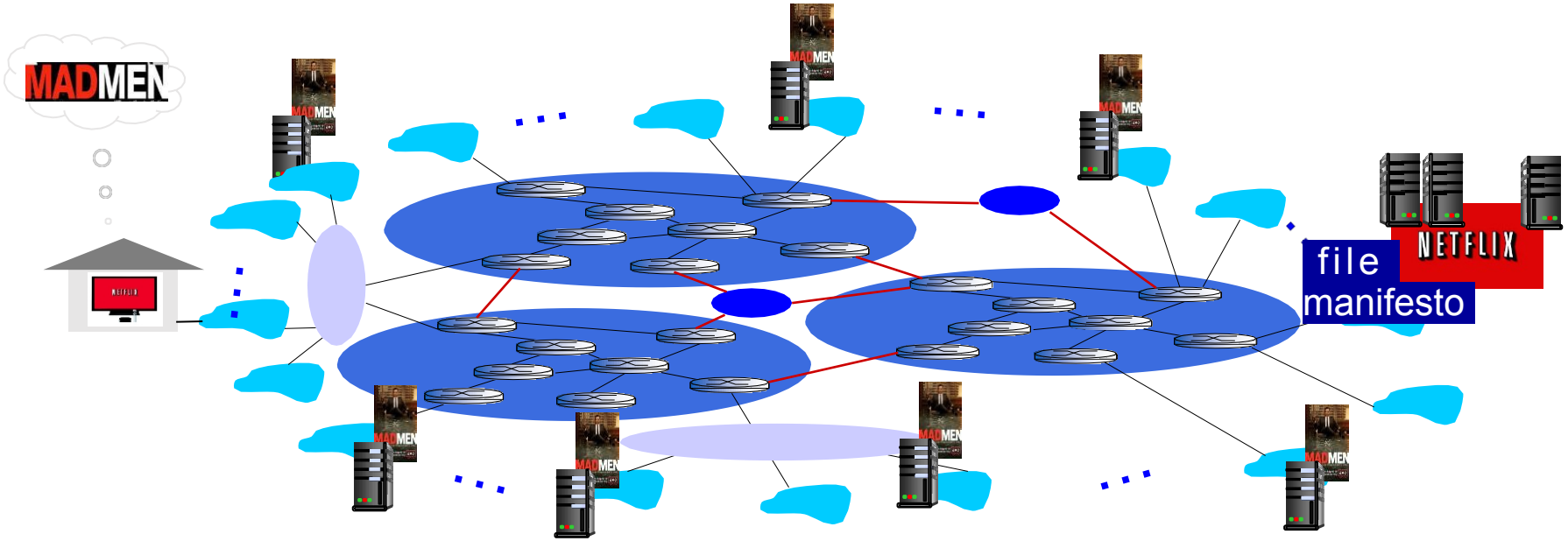
Reti di distribuzione dei contenuti (CDN)

sfida: come trasmettere contenuti (selezionati tra milioni di video) a centinaia di migliaia di utenti *simultanei*?

- *opzione 2:* memorizzare/conservare più copie di video in più siti geograficamente distribuiti *(CDN)*
  - *entrare in profondità:* spingere i server CDN in profondità in molte reti di accesso
    - vicino agli utenti
    - Akamai: 240.000 server distribuiti in > 120 paesi (2015)
  - *portare a casa: un* numero minore (da 10) di cluster più grandi nei POP vicino alle reti di accesso
    - utilizzato da Limelight

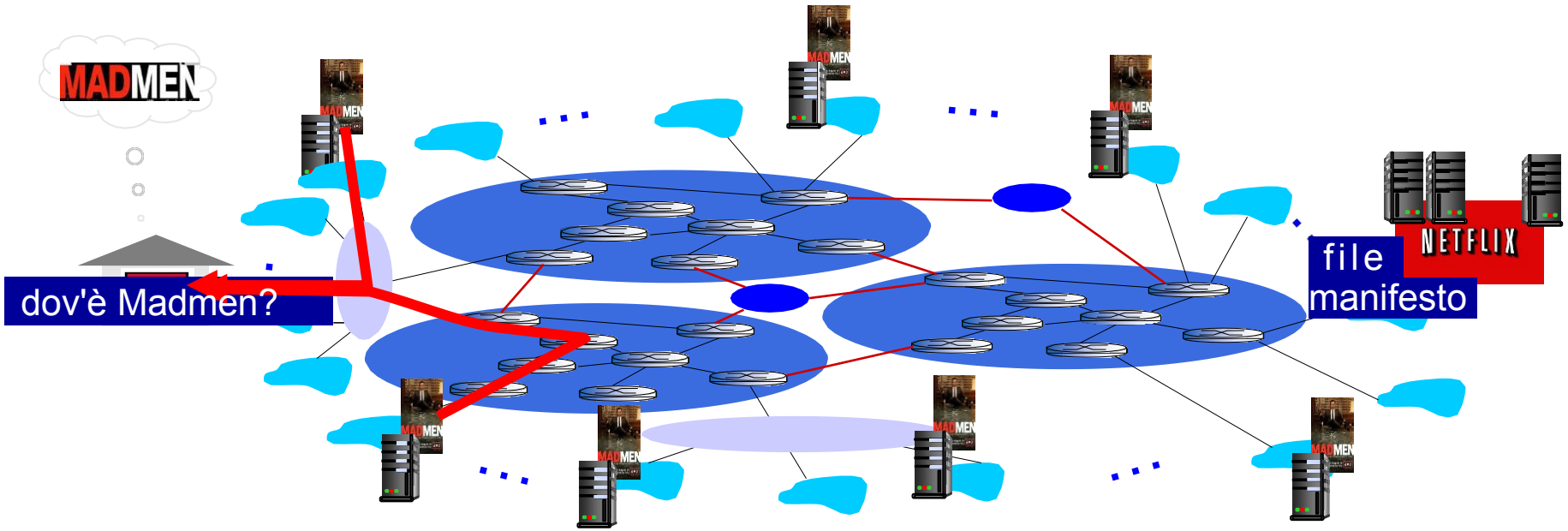
Come funziona Netflix?

- Netflix: memorizza copie di contenuti (ad esempio, MADMEN) nei suoi nodi CDN OpenConnect (in tutto il mondo).
- l'abbonato richiede il contenuto, il fornitore di servizi restituisce il manifesto
  - utilizzando il manifest, il client recupera i contenuti alla massima velocità supportata
  - può scegliere una velocità diversa o una copia se il percorso di rete è congestionato


MADMEN
NETFLIX
file
manifesto

Come funziona Netflix?

- Netflix: memorizza copie di contenuti (ad esempio, MADMEN) nei suoi nodi CDN OpenConnect (in tutto il mondo).
- l'abbonato richiede il contenuto, il fornitore di servizi restituisce il manifesto
  - utilizzando il manifest, il client recupera i contenuti alla massima velocità supportata
  - può scegliere una velocità diversa o una copia se il percorso di rete è congestionato


MADMEN
dov'è Madmen?
file
manifesto
NETFLIX

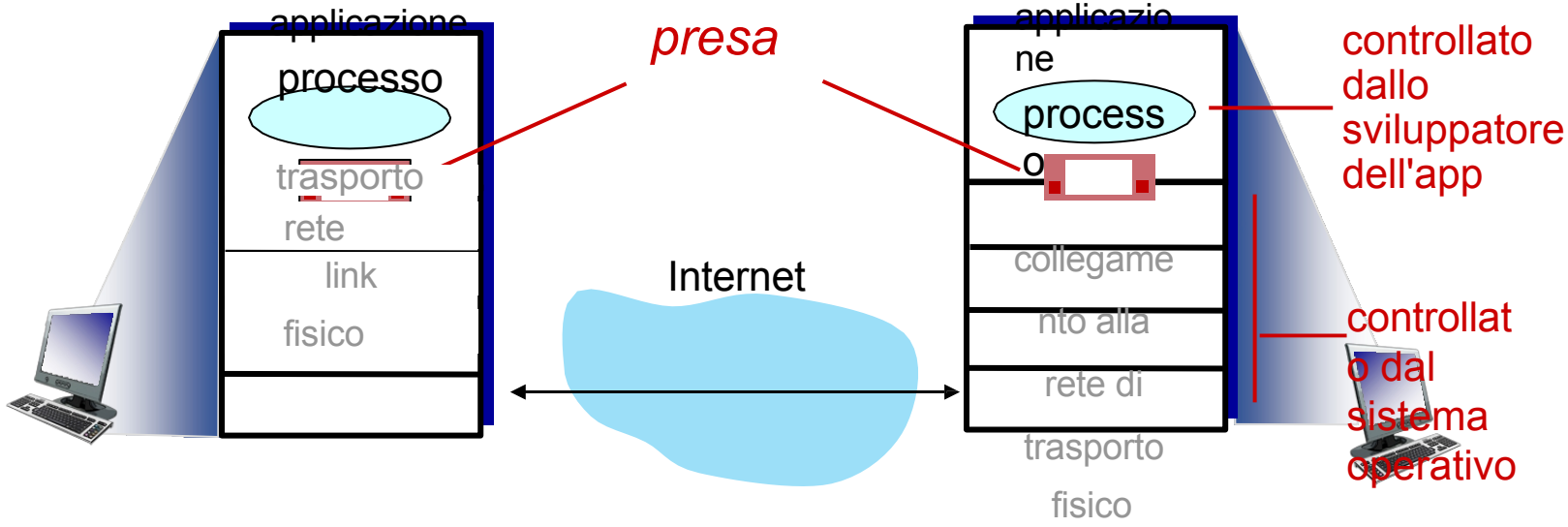
Strato applicativo: Panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Il sistema dei nomi di dominio DNS
- Applicazioni P2P
- reti di streaming video e distribuzione di contenuti
- programmazione di socket con UDP e TCP

Programmazione delle prese

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

socket: porta tra il processo applicativo e il protocollo di trasporto finale



Programmazione delle prese di corrente

Due tipi di socket per due servizi di trasporto:

- *UDP:* datagramma inaffidabile
- *TCP:* affidabile, orientato al flusso di byte

Esempio di applicazione:

1. il client legge una riga di caratteri (dati) dalla sua tastiera e la invia al server
2. Il server riceve i dati e converte i caratteri in maiuscolo.
3. Il server invia i dati modificati al client
4. il client riceve i dati modificati e li visualizza sul proprio schermo

Programmazione di socket con UDP

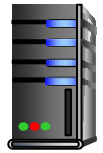
UDP: nessuna "connessione" tra client e server:

- nessun handshake prima dell'invio dei dati
- Il mittente allega esplicitamente l'indirizzo di destinazione IP e la porta # a ciascun pacchetto.
- Il ricevitore estrae l'indirizzo IP del mittente e il numero di porta dal pacchetto ricevuto.

UDP: i dati trasmessi possono essere persi o ricevuti in modo non ordinato Punto di vista dell'applicazione:

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

Interazione socket client/server: UDP



server (in esecuzione su serverIP)

cliente 

creare una presa:
porta

creare un socket, porta= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

leggere il datagramma da
serverSocket

scrivere una
risposta a
serverSocket
specificando
l'indirizzo del
client e il
numero di

clientSocket =
socket(AF_INET,SOCK_DGRAM)

Creare un datagramma con
indirizzo IP del server e porta=x;
inviare il datagramma tramite
clientSocket

leggere il datagramma da
clientSocket

chiudere
clientSocket

Esempio di applicazione: Client UDP

Python UDPClient

includere la libreria socket di Python →
```
da socket import * serverName
= 'hostname' serverPort =
12000 clientSocket =
```
creare un socket UDP →
```
socket(AF_INET,
                    SOCK_DGRAM)
```
ottenere l'input da →
```
message = input('Input lowercase sentence:')
```
tastiera dell'utente allegare il nome del server, la porta →
```
clientSocket.sendto(message.encode(),
```
al messaggio; inviare nel socket
```
                    (serverName, serverPort))
```
→
```
modifiedMessage, serverAddress =
```
leggere i dati di risposta (byte) dal socket
```
                    clientSocket.recvfrom(2048)
```
→
```
print(modifiedMessage.decode())
```
stampa la stringa ricevuta e chiude il socket
```
clientSocket.close()
```

Nota: questo aggiornamento del codice (2023) a Python 3

Esempio di applicazione: Server UDP

UDPServer Python

```
da socket importa *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort)) print('Il
server è pronto a ricevere') while True:
    messaggio, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

creare un socket UDP

legare il socket alla porta locale numero 12000

ciclo per sempre

Leggere dal socket UDP nel messaggio, ottenendo indirizzo del client (IP e porta del client)

inviare la stringa maiuscola a questo client

Nota: questo aggiornamento del codice (2023) a Python 3

Programmazione di socket con TCP

Il client deve contattare il server

- Il processo del server deve essere in esecuzione
- Il server deve aver creato un socket (porta) che accolga il contatto del cliente

Il client contatta il server tramite:

- Creare un socket TCP, specificando l'indirizzo IP e il numero di porta del processo server.

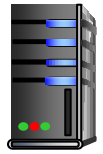
- *quando il client crea il socket: il* client TCP stabilisce la connessione con il server TCP

- quando viene contattato dal client, il *server TCP crea un nuovo socket* per il processo server per comunicare con quel particolare client
  - consente al server di parlare con più client
  - porta sorgente del client # e indirizzo IP utilizzati per distinguere i client (maggiori informazioni nel capitolo 3)

Punto di vista dell'applicazione

TCP fornisce un trasferimento affidabile e ordinato di flussi

Interazione socket client/server: TCP



server (in esecuzione su hostid)

cliente 

crea un socket,
`porta=x`, per le
richieste in arrivo:
serverSocket = socket()

attendere la
richiesta di
connessione in
arrivo
connectionSocket
=
serverSocket.accept()

TCP
impostazione
della
connessione

creare una presa,
connettersi a **`hostid`**, **`porta=x`**
clientSocket = socket()

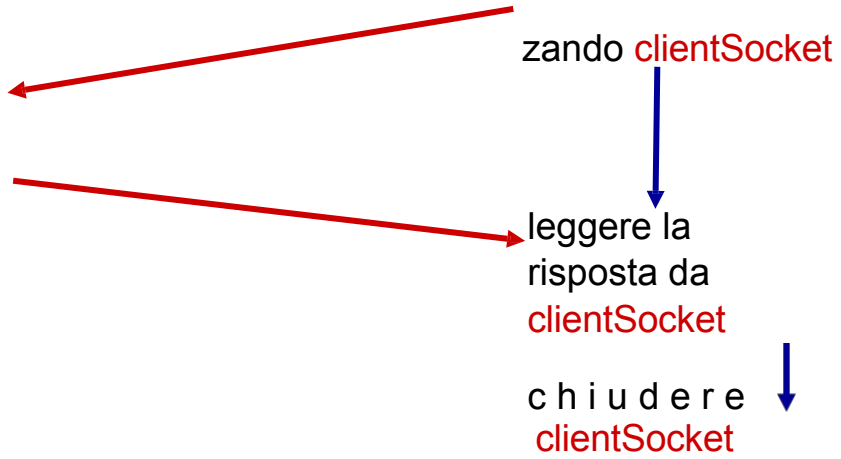
leggere la
richiesta da
connectionSocket

scrivere risposta
a

connectionSocket

chiudere connectionSocket

i n v i a r e l a r i c h i e s t a u t i l i z



Esempio di applicazione: Client TCP

Python TCPClient

```
da socket import * serverName
= 'servername' serverPort =
12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('Dal server:', modifiedSentence.decode())
clientSocket.close()
```

creare un socket TCP per il server, porta remota 12000

Non è necessario allegare il nome del server, la porta

Nota: questo aggiornamento del codice (2023) a Python 3

Esempio di applicazione: Server TCP

Server TCPS in Python

```
 da socket importa *
serverPort = 12000
serverSocket =
socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print('Il server è pronto a ricevere') while
True:
    connectionSocket, addr = serverSocket.accept()

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

creare un socket TCP accogliente

il server inizia ad ascoltare le richieste TCP in arrivo

loop per sempre

il server attende su accept() le richieste in arrivo, al ritorno viene creato un nuovo socket

leggere i byte dal socket (ma non l'indirizzo come in UDP)

Nota: questo aggiornamento del codice (2023) a Python 3

chiudere la connessione a questo cliente (ma *non* presa di benvenuto)

```
connectionSocket.close()
```

Nota: questo aggiornamento del codice (2023) a Python 3