RETI DI CALCOLATORI

Autunno 2018

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 13:
DNS E HTTP
22/11/2018

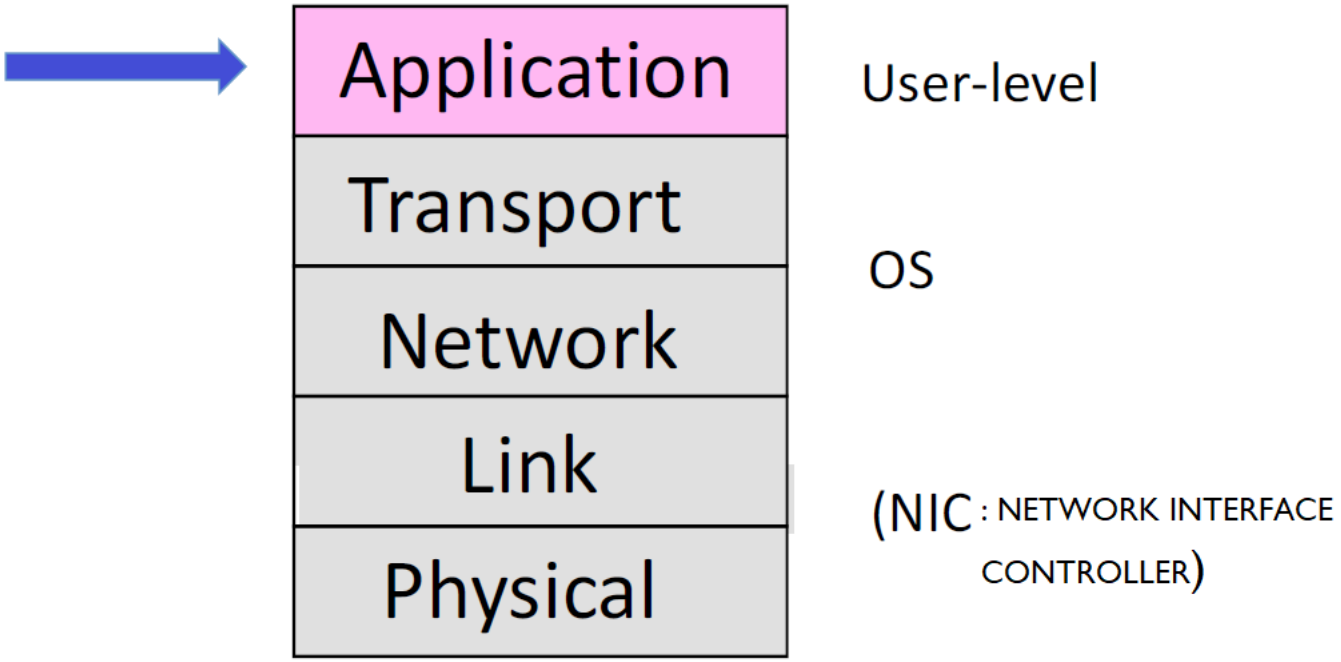
parte di queste slides sono
ricavati da slides pubblicate in corsi
universitari tenuti da colleghi
italiani e stranieri

MATERIALE DIDATTICO

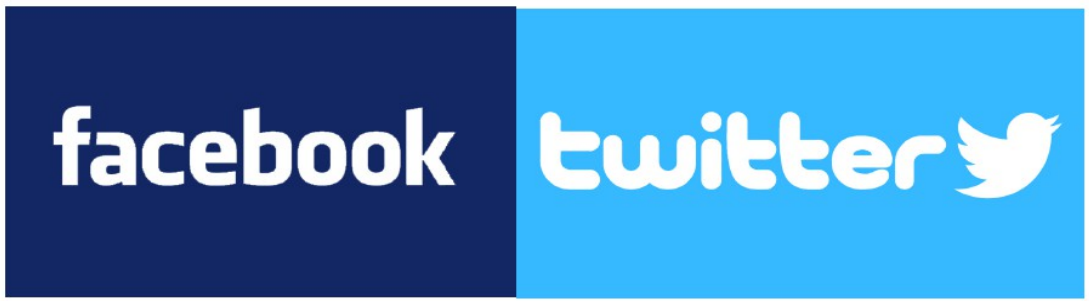
Fourozan

- Paragrafo 2.1
- Paragrafo 2.2
- Paragrafo 2.3.1, 2.3.6

DOVE SIAMO?


Application
Transport
Network
Link
Physical
User-level
OS
(NIC : NETWORK INTERFACE CONTROLLER)

IL LIVELLO DELLE APPLICAZIONI

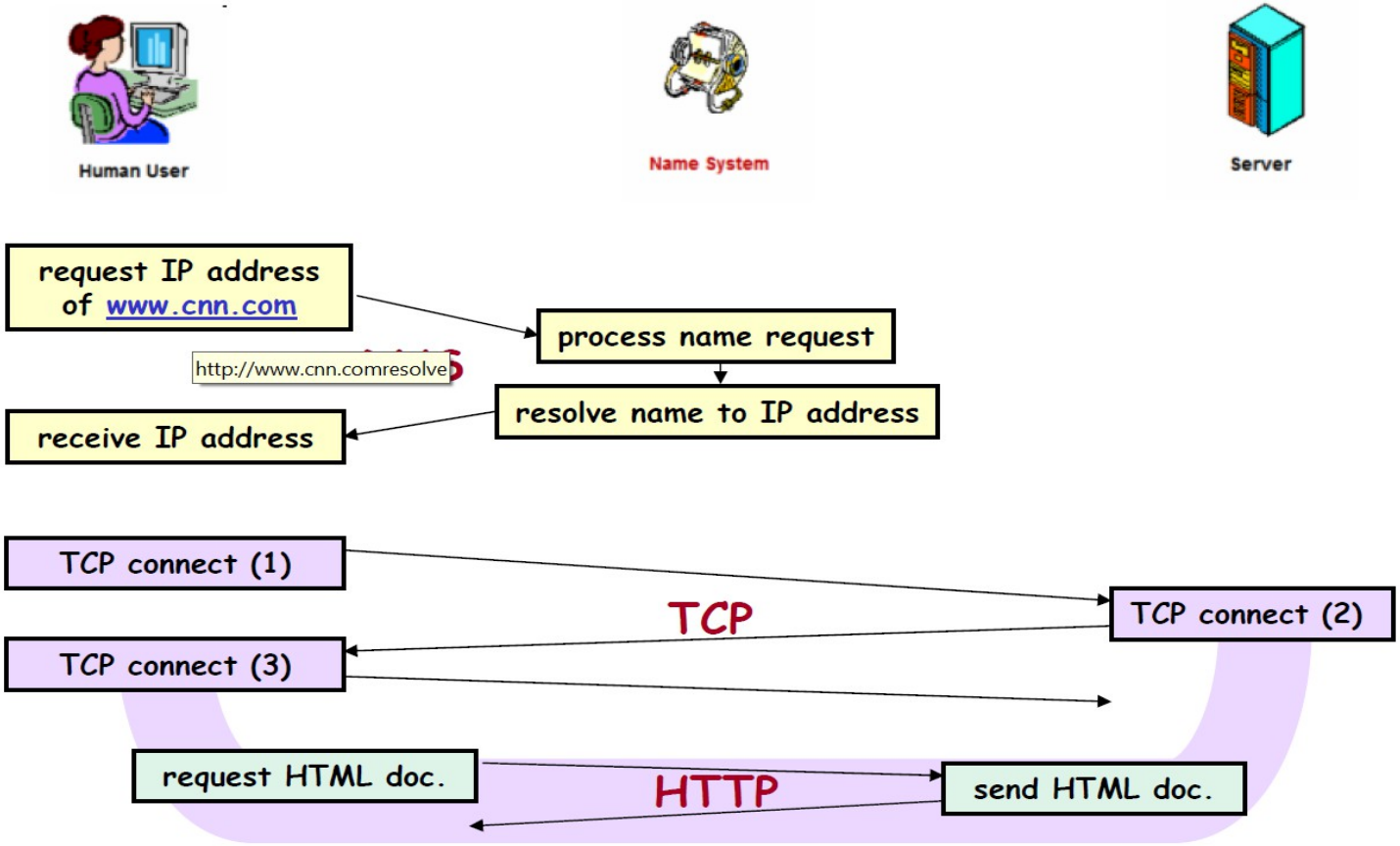


Online search service

Online shopping

Video Streaming

REPERIRE UNA PAGINA WEB SERVIZI E PROTOCOLLI

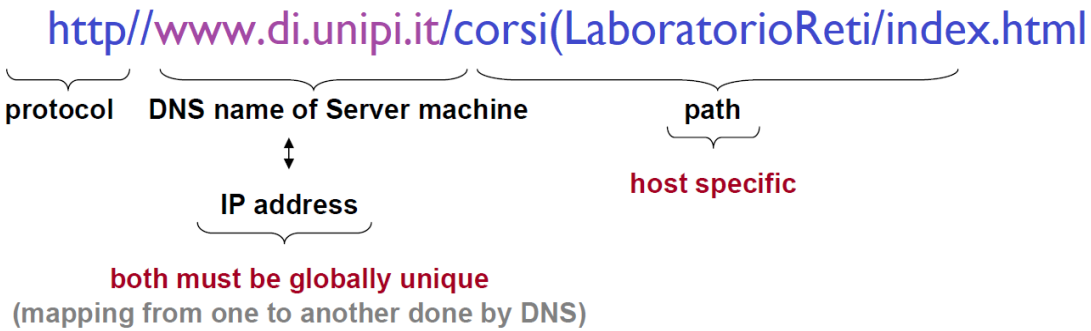


in questa lezione ci occupiamo di

- come risolvere un nome simbolico (DNS)
- come funziona il protocollo client-server alla base del web: HTTP

DOMAIN NAME SYSTEM

- indirizzi di host, es: 72.14.203.99
  - universali ed unici
  - facilmente manipolabili dai routers
  - rispecchiamo la struttura della rete (dove?)
- nomi di host, es. www.google.com
  - adatti all'uso umano
  - rispecchiano la struttura della organizzazione (chi?)
- DNS names ≠URL
  - una tipica URL contiene tre parti: protocol + DNS name + path



- Domain Name System (DNS) mappa nomi simbolici in indirizzi e viceversa

DNS: UN PO' DI STORIA

- inizialmente (prima del 1984) tutti i mapping host-indirizzo si trovavano nel file hosts.txt
  - mantenuto dallo Stanford Research Institute (SRI)
  - modifiche/aggiunte communicate a SRI per e-mail
  - nuove versioni di hosts.txt reperite mediante FTP da SRI
  - l'amministratore poteva scegliere i nomi degli host autonomamente
- cercare /etc/hosts sul vostro PC: contiene la risoluzione di alcuni nomi ed è una eredità dello "storico" hosts.txt
- con la crescita di Internet, questo sistema divenne inaccettabile
  - carico ingestibile da un'unica organizzazione (SRI)
  - nomi non univoci
  - gli host possiedono copie non aggiornate di hosts.txt
- il Domain Name System (DNS) è stato inventato per risolvere questi problemi
- prima implementazione fatta da 4 studenti di UCB!

DOMAIN NAME SYSTEM

- necessaria una soluzione automatizzata e decentralizzata: Domain Name System (DNS)
- ma cosa è un domain name system?:
  - database distribuito memorizzato su molti nodi ed implementato mediante una gerarchia di name servers
  - application layer protocol:
    - permette agli end host di effettuare query sul database distribuito
    - eseguito su UDP (porta 53)
    - a differenza di altri protocolli (HTP, FTP, …), il DNS non è una applicazione con cui gli utenti interagiscono direttamente
    - fornisce un servizio ad altre applicazioni
    - ancora un esempio della filosofia di Internet: concentrare la complessità negli end host ("at the edge of Internet").

DOMAIN NAME SYSTEM: PERCHE' UDP?

- meno overhead
- messaggi corti
- tempo per set-up connessione di TCP lungo
- DNS scambia un unico messaggio tra una coppia di server
- … ma in genere contatta più server
  - se si usasse TCP ogni volta sarebbe necessario creare una nuova connessione
- e se un messaggio non ha risposta entro un timeout?
  - problema risolto a livello applicativo
  - viene riinviato
  - contattare DNS diversi dopo qualche tentativo

DOMAIN NAME SYSTEM: SERVIZI OFFERTI

- traduzione indirizzi simbolici (hostname) in indirizzi IP
  - servizio fornito agli altri protocolli del livello applicazione: HTTP, SMTP, FTP.
- reverse translation IP Address in nomi simbolici
- host aliasing
  - associare ad uno stesso host uno o più nomi
- mail server aliasing: permette l'uso di indirizzi mnemonici (bob@hotmail.com) anche se il nome del mail server di hotmail è più complesso
- load distribution

HOST ALIASING

- un host può avere uno o più sinonimi (alias)
- aliasing: più di un hostname mappato sullo stesso indirizzo IP
- esempio: relay1.west-coast.enterprise.com potrebbe avere due sinonimi, quali enterprise.com e www.enterprise.com
  - relay1.west-coast.enterprise.com è un hostname canonico
  - enterprise.com e www.enterprise.com sono alias
- gli alias sono più facili da ricordare
- Il DNS può essere invocato da un'applicazione usando l'alias per avere l'indirizzo canonico e quindi il suo indirizzo IP

LOAD DISTRIBUTION

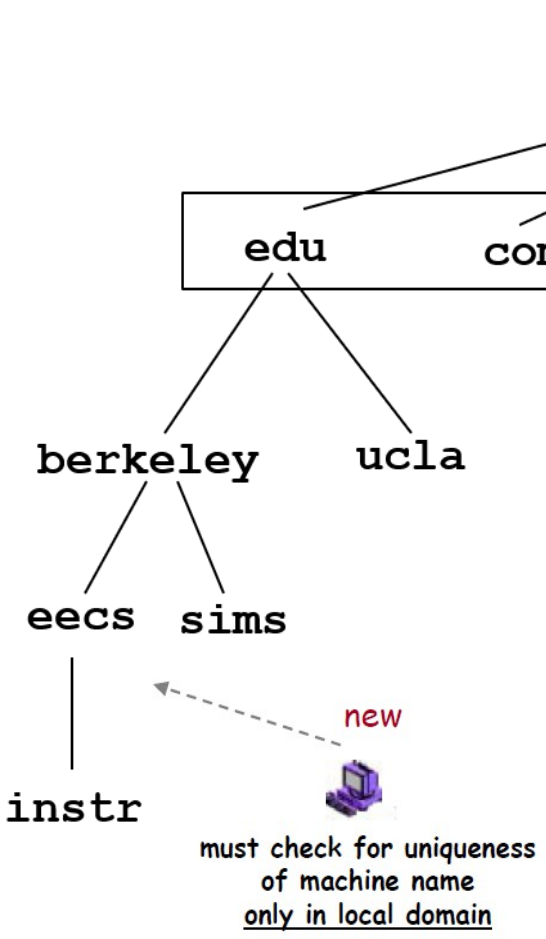
- DNS viene utilizzato per distribuire il carico tra server replicati (es. web server)
- i siti con molto traffico (es. cnn.com) vengono replicati su più server, e ciascuno di questi gira su un end host diverso e presenta un indirizzo IP differente
- Hostname canonico associato a un insieme di indirizzi IP
- il DNS contiene l'insieme di indirizzi IP
- quando un client effettua un richiesta DNS per un nome mappato in un insieme di indirizzi, il server risponde con l'insieme di indirizzi ma variando l'ordinamento a ogni risposta
- la rotazione DNS distribuisce il traffico sui server replicati

DNS: TRE GERARCHIE

- Idea di base per la realizzazione del DNS: strutturazione gerarchica
- 3 gerarchie
- spazio di nomi gerarchico
  - opposto allo spazio "piatto" definito originariamente
- amministrazione gerarchica
  - opposta ad organizzazione centralizzata
- gerarchia di servers che forniscono il servizio
  - opposta ad un unico server centralizzato

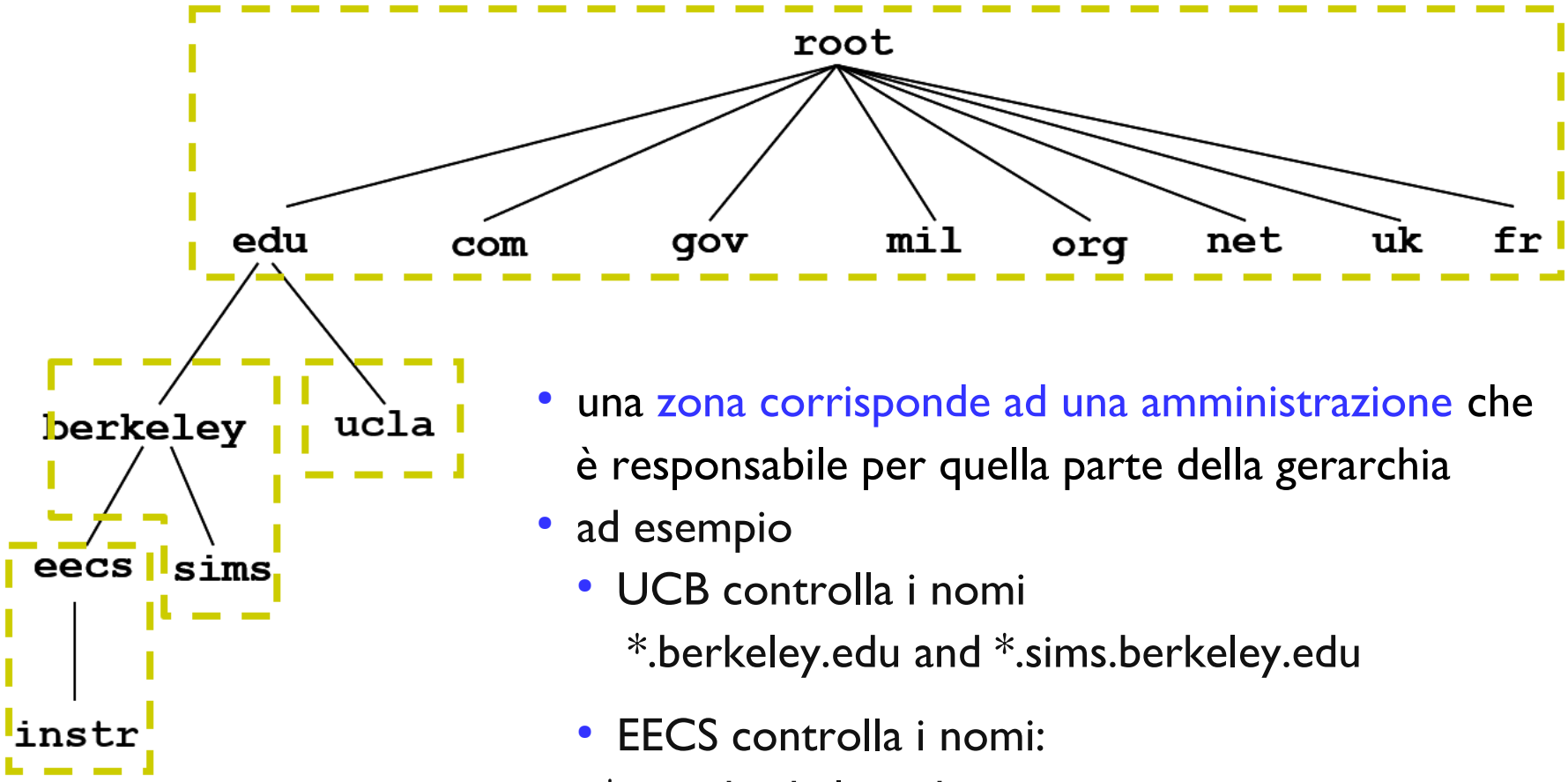
SPAZIO DEI NOMI GERARCHICO

Come può un naming system adattarsi ad una rete la cui dimensione cresce rapidamente senza un sito centrale che amministri l'intero sistema?



- Lo spazio dei nomi è partizionato al top level, e l'autorità per la gestione dei nomi nelle sottodivisioni è passata ad organizzazioni designate
- Lo spazio dei nomi è poi ulteriormente suddiviso finchè la divisione dello spazio è abbastanza piccola
  - massimo 128 livelli, ma raramente usati più di 4
- la sintassi del nome riflette la delegazione gerarchica
- lo spazio gerarchico dei nomi rende più semplice:
  - assegnare nuovi nomi senza coinvolgere una entità centralizzata

AMMINISTRAZIONE GERARCHICA



- una zona corrisponde ad una amministrazione che è responsabile per quella parte della gerarchia
- ad esempio
  - UCB controlla i nomi
    *.berkeley.edu and *.sims.berkeley.edu
  - EECS controlla i nomi:
    *.eecs.berkeley.edu

DNS DATABASE

- database gerarchico distribuito gestito da un gran numero di servers dislocati in tutto il mondo

- nessun singolo DNS server possiede i mapping per tutti gli hosts
  - mappings divisi e distribuiti tra tutti i DNS servers
  - ogni server memorizza un (piccolo!) sottoinsieme dell'intero database DNS
  - deve conoscere gli altri server che sono responsabili per altre parti della gerarchia

- gerarchia dei server simile a quella dei nomi
  - Root Name Servers
  - Top-level domain (TLD) servers
  - Authoritative DNS servers: memorizzano un "resource records" per tutti i nomi nel dominio su cui hanno autorità

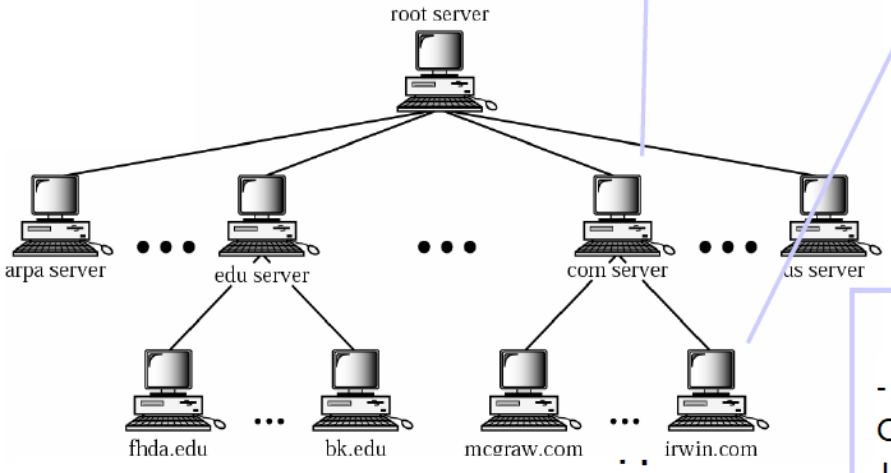
DNS DATABASE

Root DNS server –

- insieme di servers
- in genere non memorizzano alcuna informazione, ma memorizzano un riferimento ai TLD servers

Top-Level Domain (TLD) Servers

responsabili per i domini top-level come : com, org, net, edu e per tutti i top-level domain nazionali come uk, it, fr, ca,...



Authoritative DNS Servers –

- DNS della organizzazione: forniscono mapping tra nome host ed IP per tutti gli host della organizzazione che possono essere acceduti dall'esterno, come server web o gli e-mail servers.

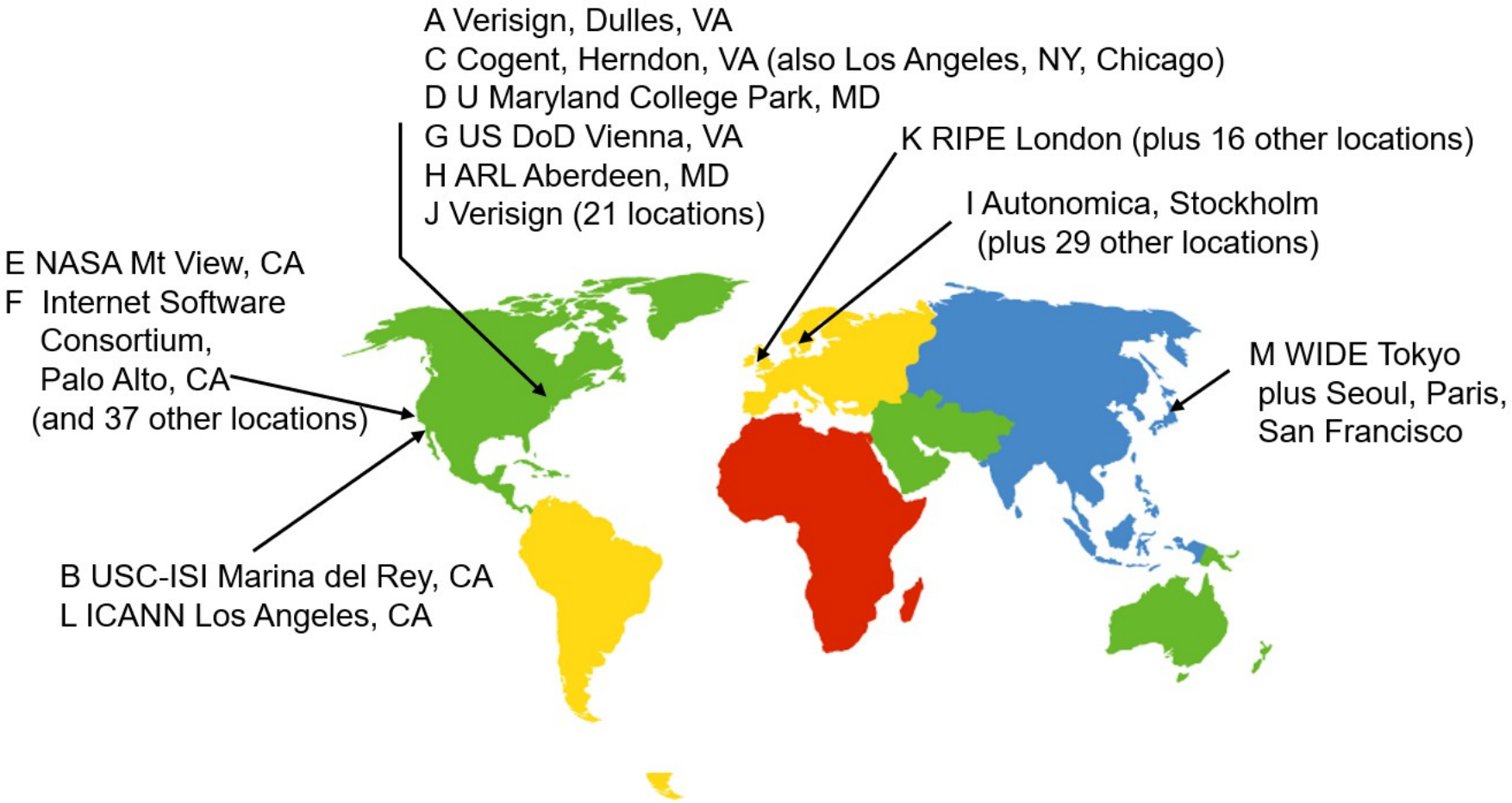
Local DNS Servers –

- non appartengono alla gerarchia, ma alla architettura DNS. Ogni ISP (impresa, università) ne ha uno.
I local DNS servers ricevono queries dagli host ed eventualmente, li propagano nella gerarchia

ROOT NAME SERVER

- 13 root logici i cui indirizzi sono ben noti alla comunità
- in realtà si tratta 376 diversi server fisici
  - gestiti da agenzie autorizzate
  - sono in grado di indicare l'indirizzo di un altro TLD server , corrispondente al nome ricercato
- ad essi si riferiscono direttamente i local server quando non sono in grado di risolvere il processo di risoluzione
- Local Name Server
  - in relatà è client del processo di risoluzione
- Root Name Server
  - Server che restituisce, a sua volta, un riferimento al server da cui iniziare la ricerca

ROOT SERVERS



LOCAL NAME SERVER (LNS)

- non appartengono strettamente alla gerarchia dei servers
  - chiamati anche defaut o caching name server
- ciascun ente (università, compagnia) ne istalla uno
- tutti gli host della organizzazione richiedono al LNS il servizio di risoluzione
  - ogni host ha configurato l'indirizzo del LNS.
  - quando un host effettua una richiesta DNS, la query viene inviata al LNS che opera da proxy per conto del client e, tipicamente, manda la query attraverso la gerarchia dei server
  - l'uso del LNS consente agli end-host di effettuare una sola query verso di essi
    - saranno essi poi a gestire la sequenza di interrogazioni ai server della gerarchia necessarie per la risoluzione del nome
- effettuano caching delle richieste: alcune richieste risolte direttamente dal LNS.

DNS CACHING

- LNS sfrutta il caching per migliorare le prestazioni di ritardo e per ridurre il numero di messaggi DNS che "rimbalzano" in Internet.
- una volta che un server LNS impara la mappatura, la mette nella cache
- le informazioni nella cache vengono invalidate (spariscono) dopo un certo periodo di tempo (es. 2 giorni)
- tipicamente un LNS memorizza nella cache gli indirizzi IP dei server TLD (ma anche quelli di competenza)
  - quindi i server DNS radice non vengono visitati spesso
  - esempio: più utenti in dipartimento che si connettono sul sito dell'università di Berkley

AUTHORITATIVE NAME SERVER

- Server capace di risolvere tutti i nomi all'interno di uno stesso dominio/organizzazione
- Può essere mantenuto dalla organizzazione o da un provider

COSA MEMORIZZA IL DATABASE

- database distribuito che memorizza i record di risorsa o resource record (RR).
- ogni messaggio di risposta DNS trasporta uno o più RR

Formato RR: (Name, Value, Type, TTL)

Tempo residuo di vita

Type=A

Hostname ⇒ IP address

name è il nome dell'host
value è l'indirizzo IP

Es. (relay1.bar.foo.com, 45.37.93.126, A)

Type= CName

Alias ⇒ Canonical Name

name è il nome alias di qualche nome canonico (vero)

value è il nome canonico

Es. (foo.com, relay1.bar.foo.com, CNAME)

Type=NS

Domain name ⇒ Name Server

- name è il dominio (ad esempio foo.com)
- value è il nome dell'host del server di competenza di questo dominio

Es. (foo.com, dns.foo.com, NS)

Type= MX

Alias ⇒ Mail Server Canonical Name

value è il nome canonico del server di posta associato a name

Es. (foo.com, mail.bar.foo.com, MX

ESEMPI

- server di competenza per un hostname
  - contiene un record di tipo A per l'hostname
  - es. (corsi.di.unipi.it, 131.143.22.61, A)
- server non di competenza per un dato hostname
  - contiene un record di tipo NS per il dominio che include l'hostname
  - contiene un record di tipo A che fornisce l'indirizzo IP del server DNS nel campo value del record NS
- Esempio:
  - un server TLD it non è competente per l'host corsi.di.unipi.it
  - contiene (unipi.it, dns.unipi.it, NS)

    (dns.unipi.it, 123.1117.42.222, A)

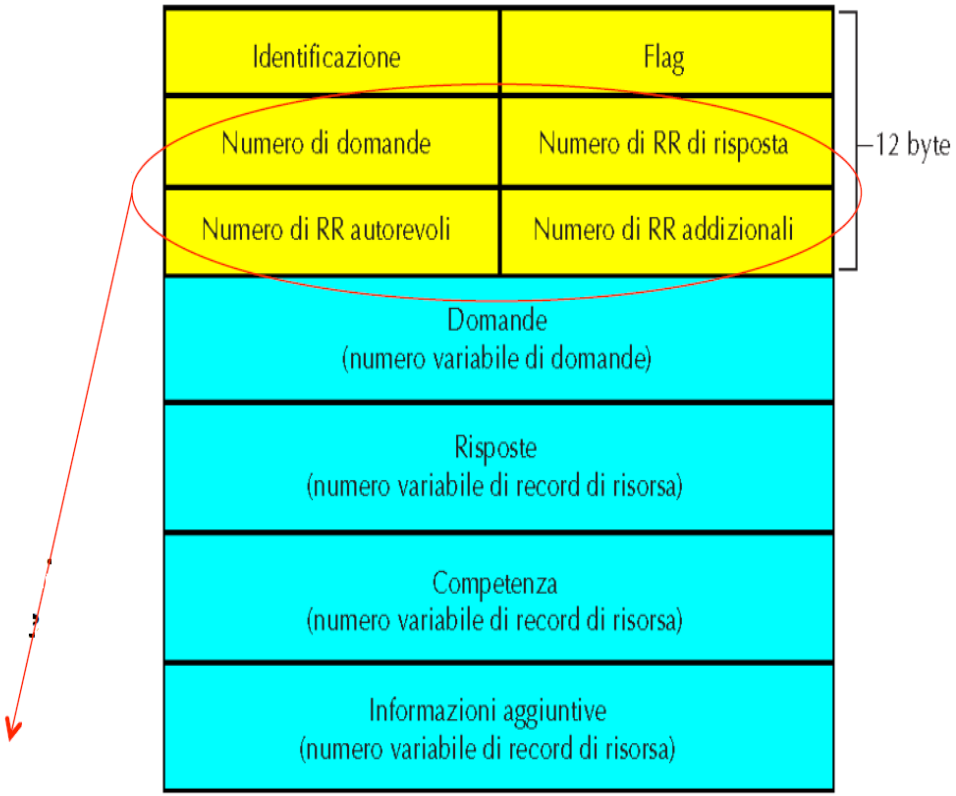
MESSAGGI DNS

- Protocollo DNS: domande (query) e messaggi di risposta, entrambi con lo stesso formato

Intestazione del messaggio

- Identificazione: numero di 16 bit per la domanda; la risposta usa lo stesso numero
- Flag:
  - domanda o risposta
  - richiesta di ricorsione
  - ricorsione disponibile
  - risposta di competenza

  (il server è competente per il nome richiesto)
- Numero di occorrenze delle quattro sezioni di tipo dati che seguono



MESSAGGI DNS

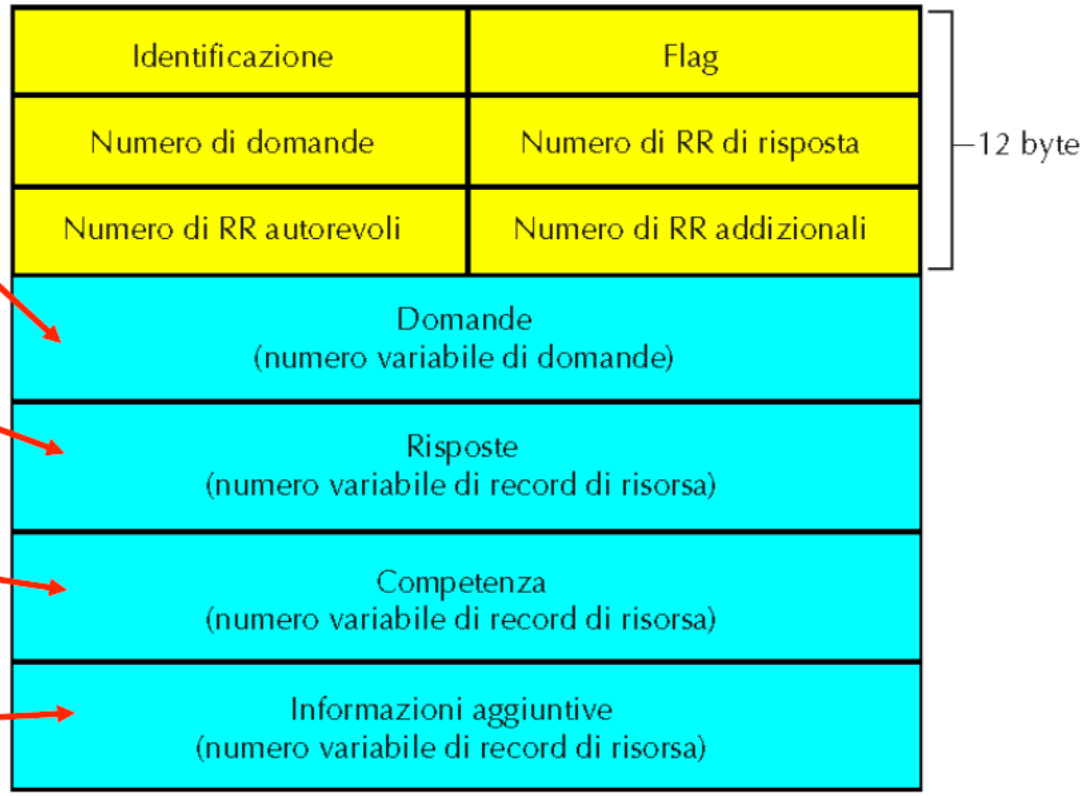
Campi per il nome richiesto e il tipo di domanda (A, MX)

RR nella risposta alla domanda

Più RR nel caso di server replicati

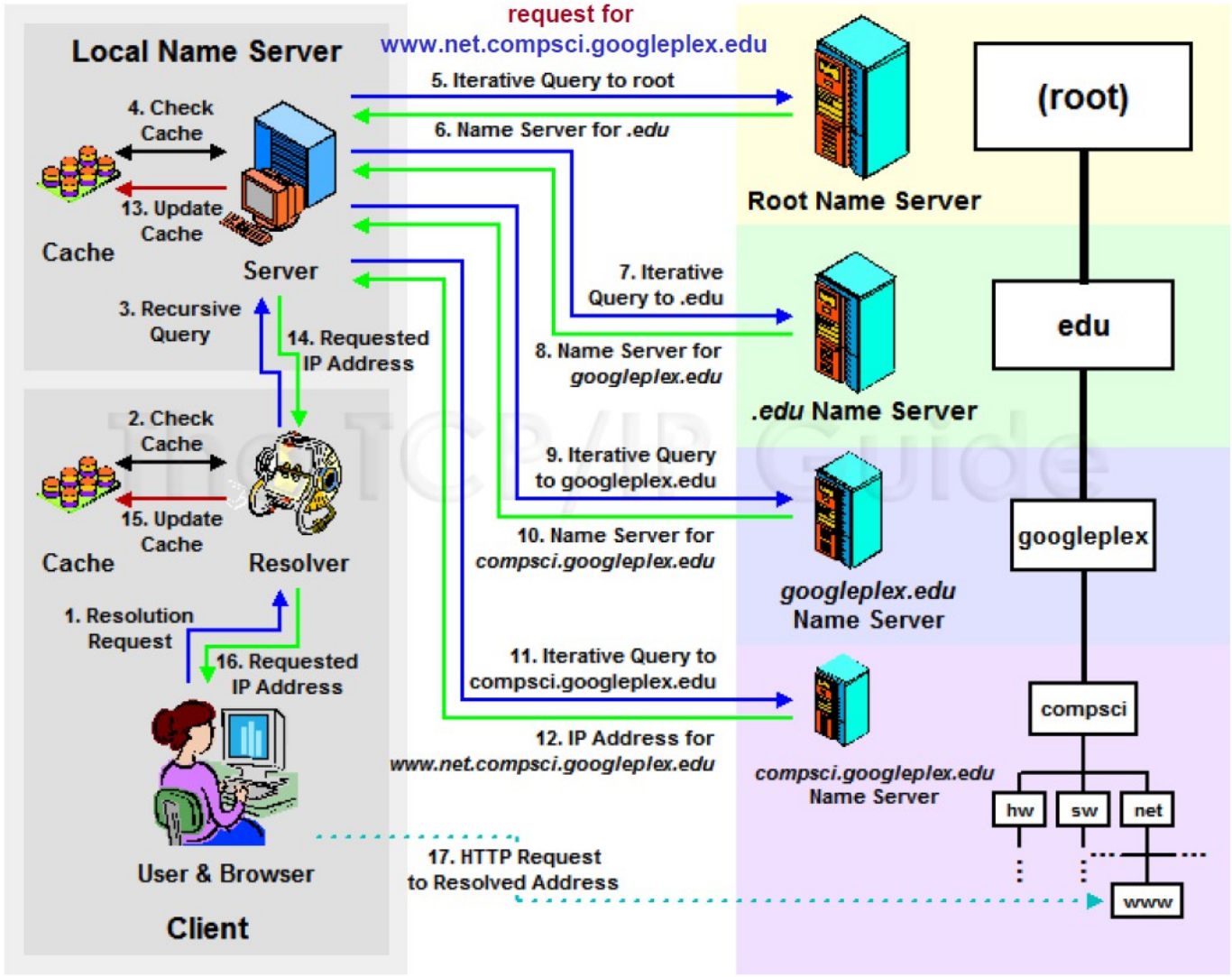
Record per i server di competenza

Informazioni extra che possono essere usate



Nel caso di una risposta MX, il campo di risposta contiene il record MX con il nome canonico del server di posta, mentre la sezione aggiuntiva contiene un record di tipo A con l'indirizzo IP relativo all'hostname canonico del server di posta

RISOLUZIONE ITERATIVA DEI NOMI



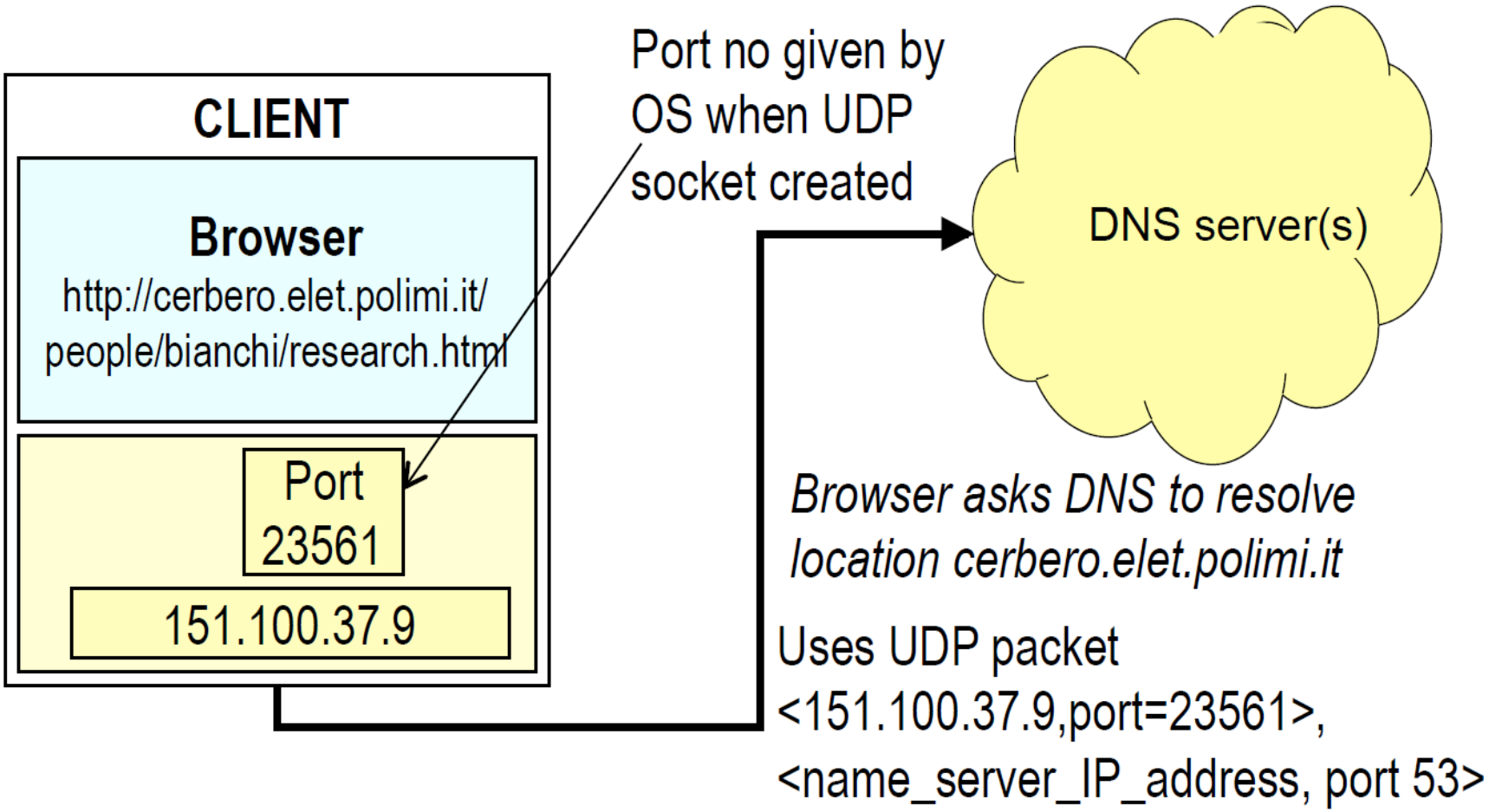
name resolution for www.net.compsci.googleplex.edu

QUERY ITERATIVE E RICORSIVE

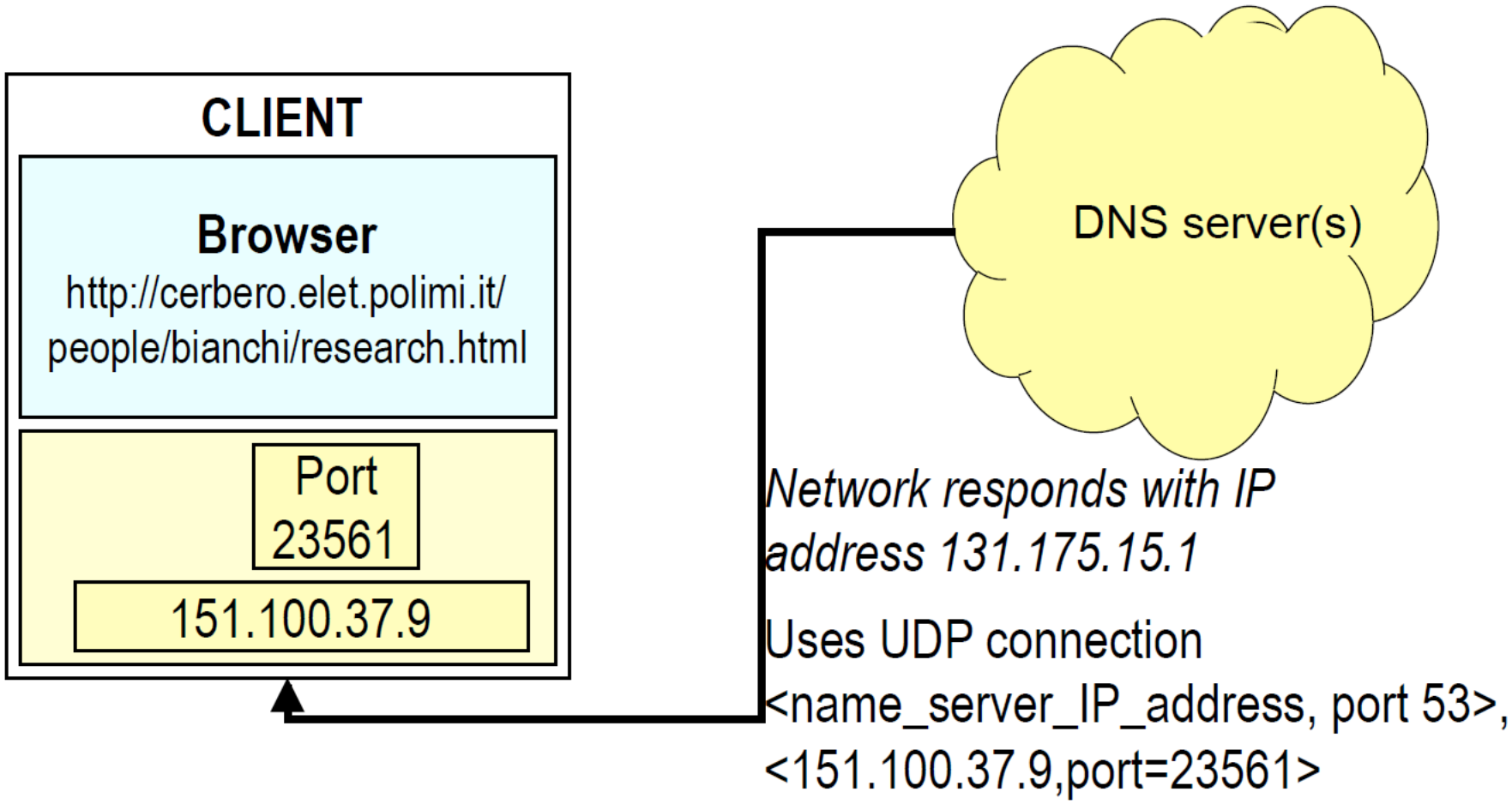
- Query iterative: il server contattato risponde con il nome del Server da contattare
  - la logica: "non conosco questo nome, ma conosco il nome di qualcuno a cui poter chiedere"

- Query ricorsive:
  - affidano il compito di proseguire il compito di traduzione del nome al server DNS contattato
  - sposta il carico della risoluzione dell'indirizzo sul server contattato
  - troppo carico per i root servers
    - in genere processo misto: iterativo per i root servers, può proseguire ricorsivo

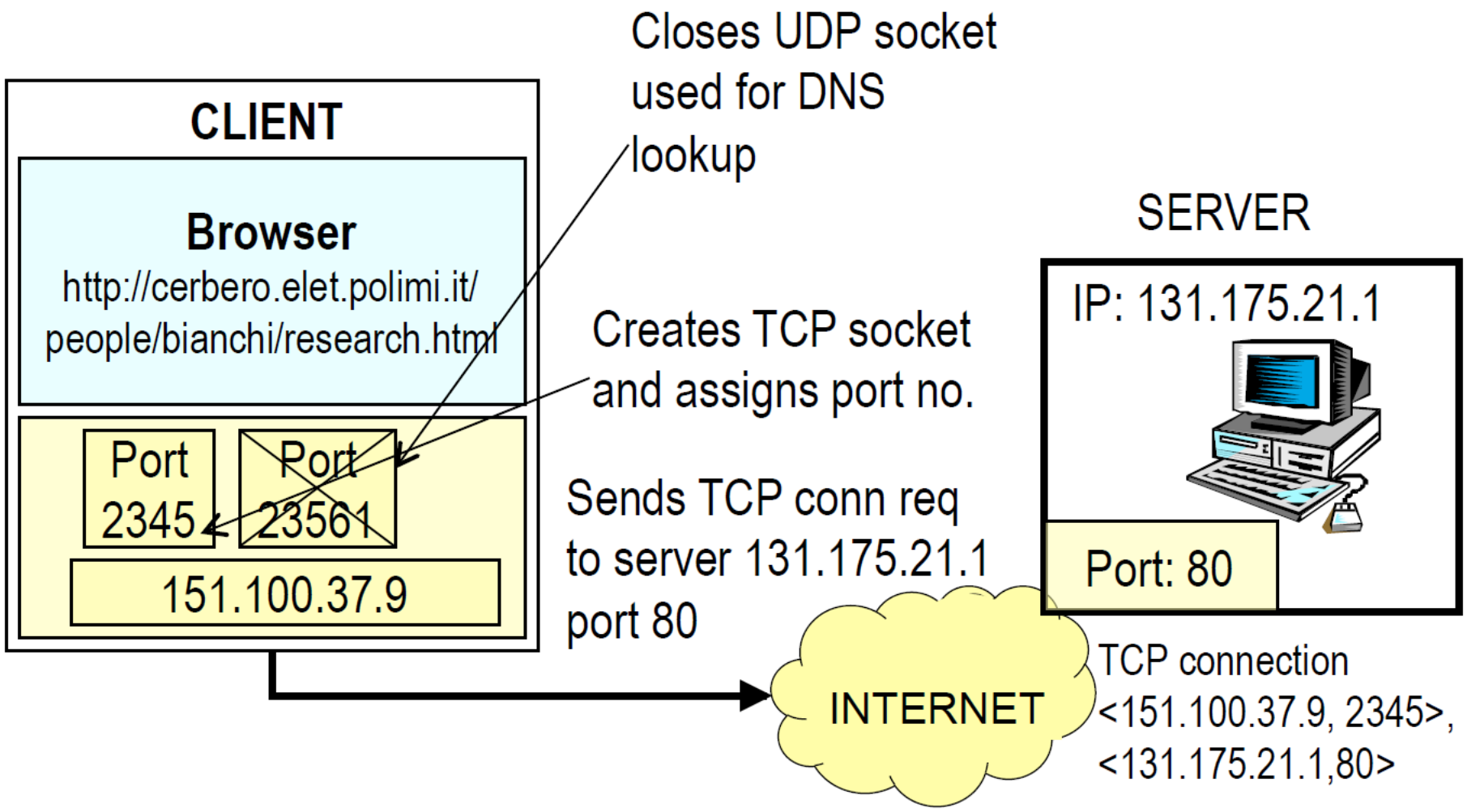
USO DI DNS DA PARTE DI UN CLIENT WEB



USO DI DNS DA PARTE DI UN CLIENT WEB



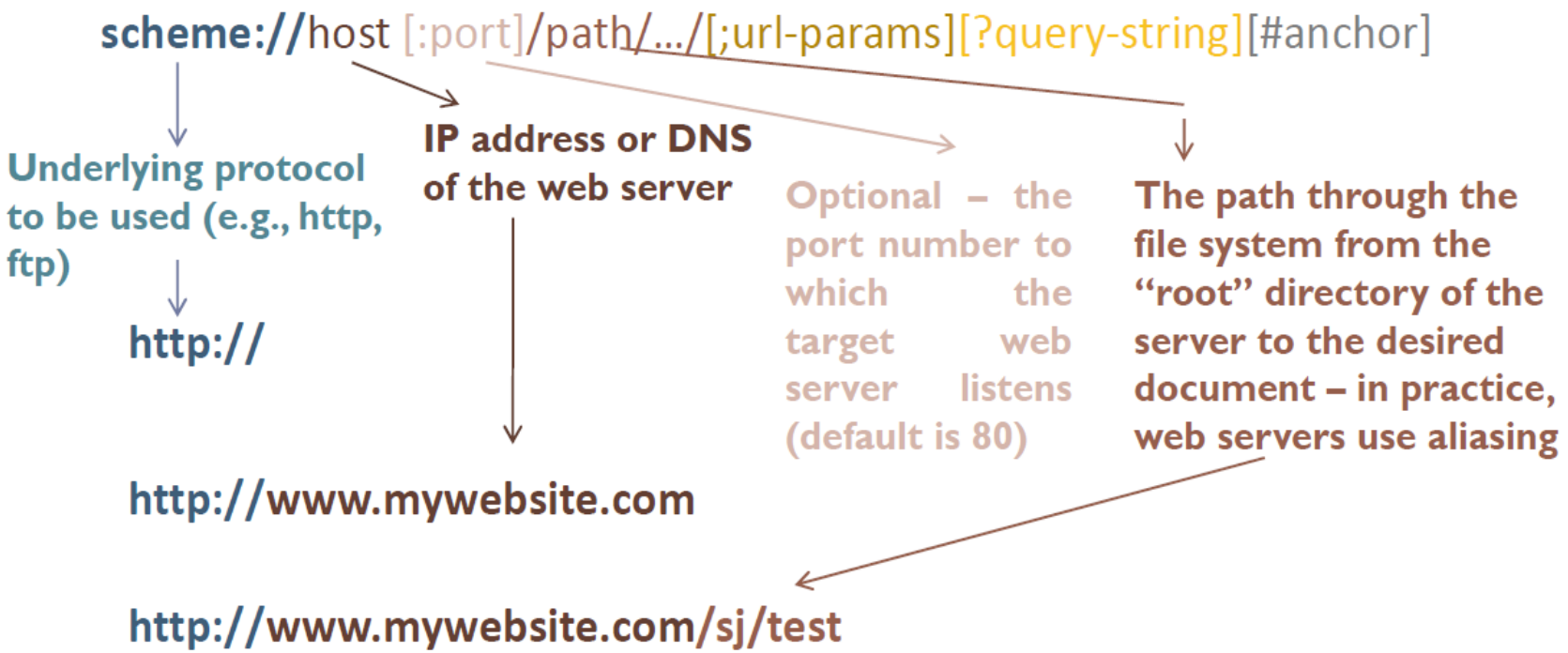
USO DI DNS DA PARTE DI UN CLIENT WEB



COMPONENTI WEB

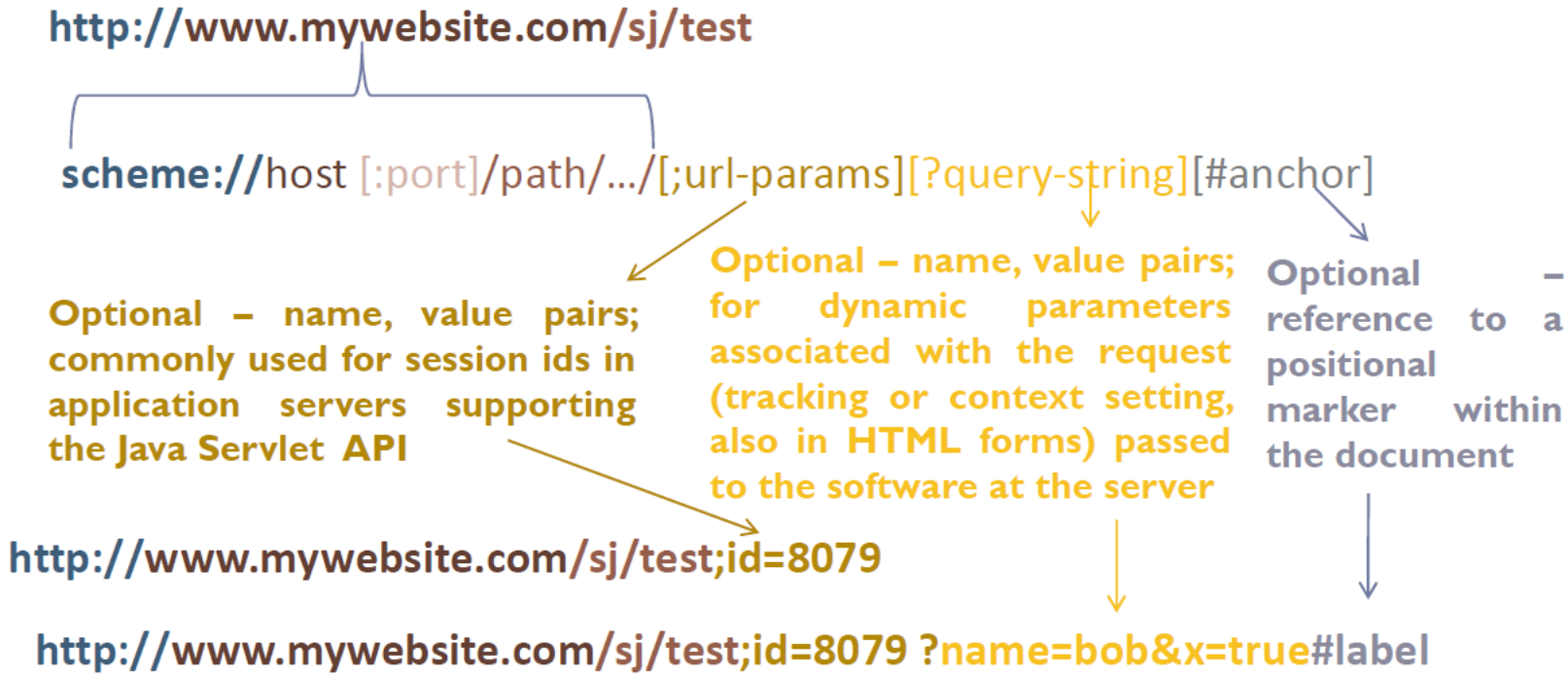
- infrastruttura
  - clients
  - servers
- contenuto
  - URL: per individuare un contenuto
  - HTML: formatting content
  - protocollo per lo scambio dell'informazione: HTTP

COMPONENTI WEB: URL



- nomi di host gerarchici
- estendere l'idea di nomi gerarchici per includere qualsiasi elemento di un file system

COMPONENTI WEB: URL

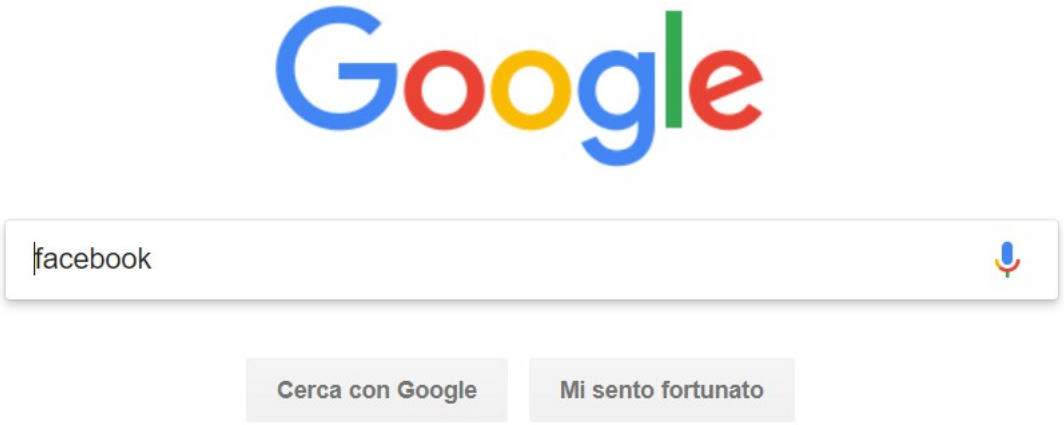


application-protocol://IP-address[:port]path-from-the-root[;par][?dyn-par][#anchor]

URL: QUERY STRING

- Query string della URL può contenere dei parametri da inviare al server
- La query è individuata sempre da un punto interrogativo all'inizio (?) seguito dalla coppia ParameterName=Value.
- esempio:

https://www.google.com?q=facebook



URL: QUERY STRING

Google Maps URLs:

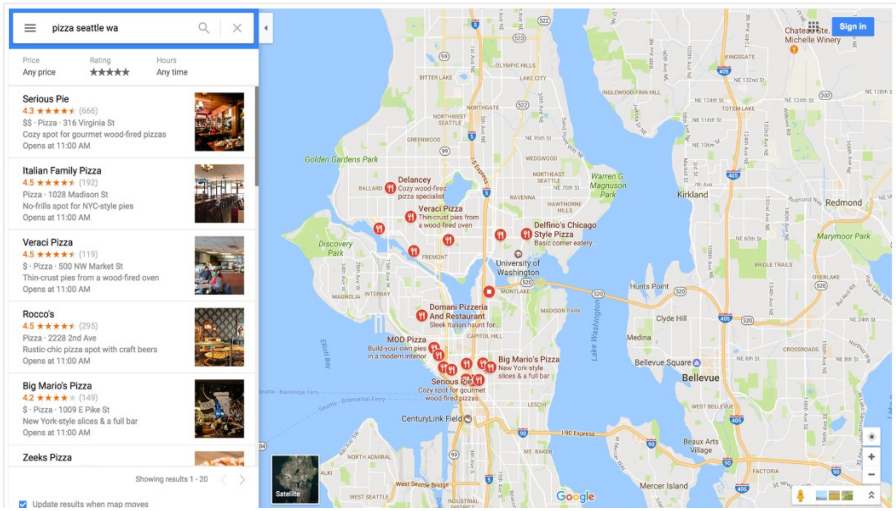
- usano la notazione delle URL, la cui sintassi è universale ed indipendente dalla piattaforma
- permettono di eseguire ricerche, dare direzioni e visualizzzare mappe senza utilizzare la Google API

https://developers.google.com/maps/documentation/urls/guide

Esempio 1:

https://www.google.com/maps/Search/

?api=1&query=pizza+seattle+wa



URL: QUERY STRING

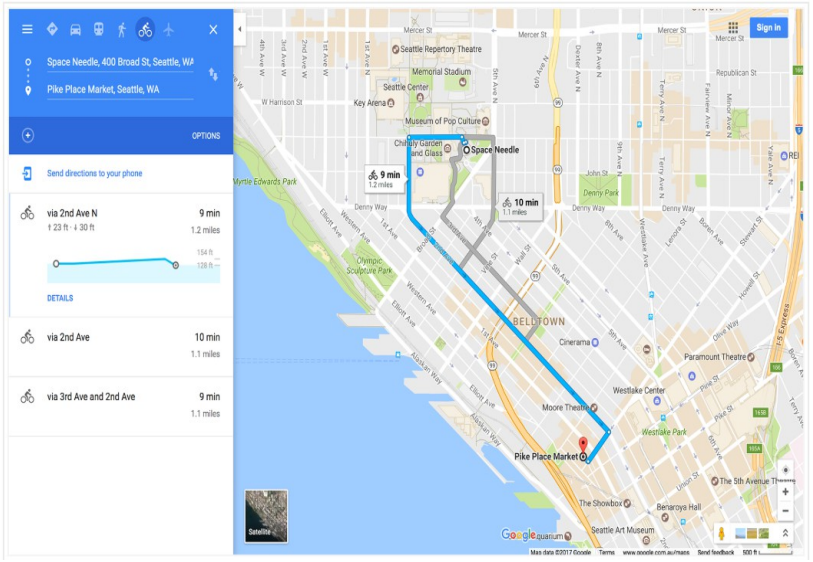
Esempio 2:

https://www.google.com/maps/dir/?api=1

&origin=Space+Needle+Seattle+WA

&destination=Pike+Place+Market+Seattle+WA

&travelmode=bicycling



Esempio 3:

https://www.google.com/maps/@?api=1

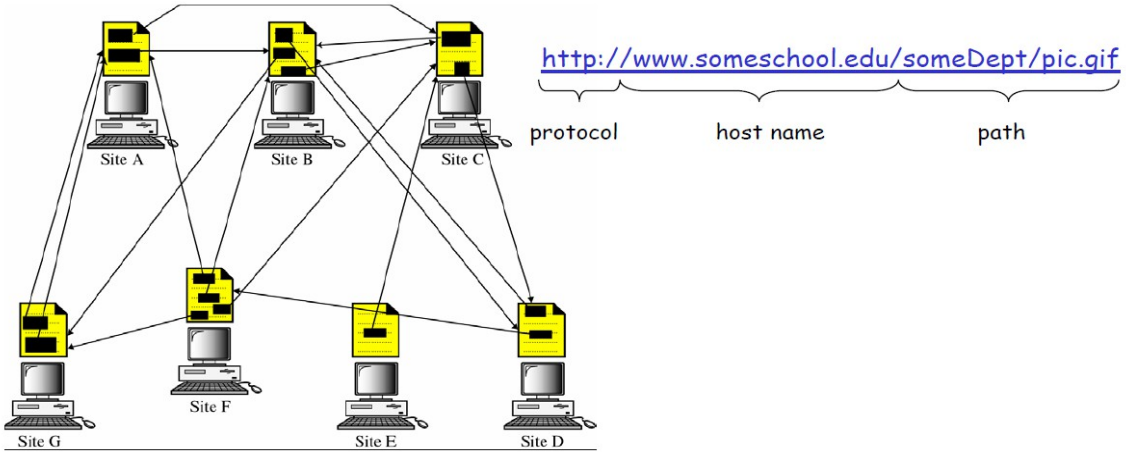
&map_action=map

&center=-33.712206,150.311941

&zoom=12&basemap=terrain

WORLD WIDE WEB

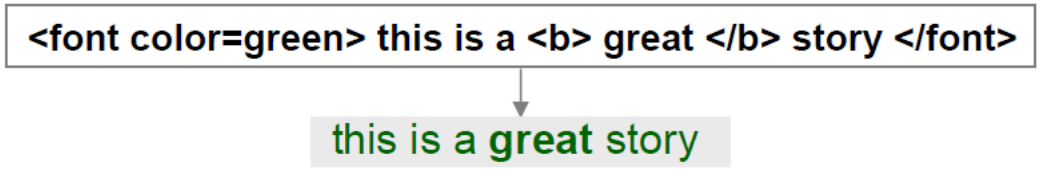
- distributed client-server repository di hypertext / hypermedia objects
- hypertext / hypermedia system system: l'informazione è organizzata come un insieme di documento (oggetti) collegati medianti dei "puntatori",
  - ogni oggetto è individuabile univocamente mediante una Uniform Resource Locator (URL)
- distributed client server system:
  - un client usando un browser può accedere servizi (che forniscono contenuti) offerti da più server distribuiti su più locazioni



WORLD WIDE WEB (WWW)

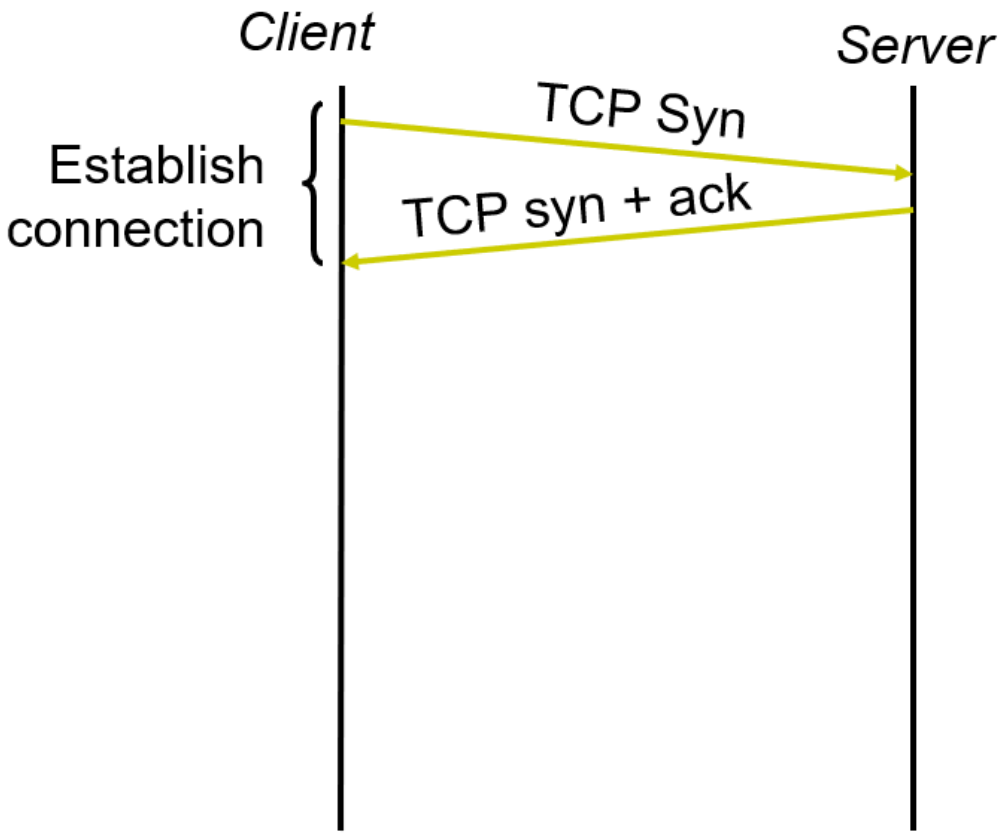
- programma client program che permette all'utente di ritrovare, interagire con oggetti WWW
  - file HTML
  - immagini JPEG
  - file audio
  - applet JAVA

- supporta più linguaggi / protocolli, che includono:
  - HTML (HyperTextMarkupLanguage): un linguaggio utilizzato per definire documenti ipertestuali: pagine Web
  - diversi tag per definire links/proprietà degli oggetti

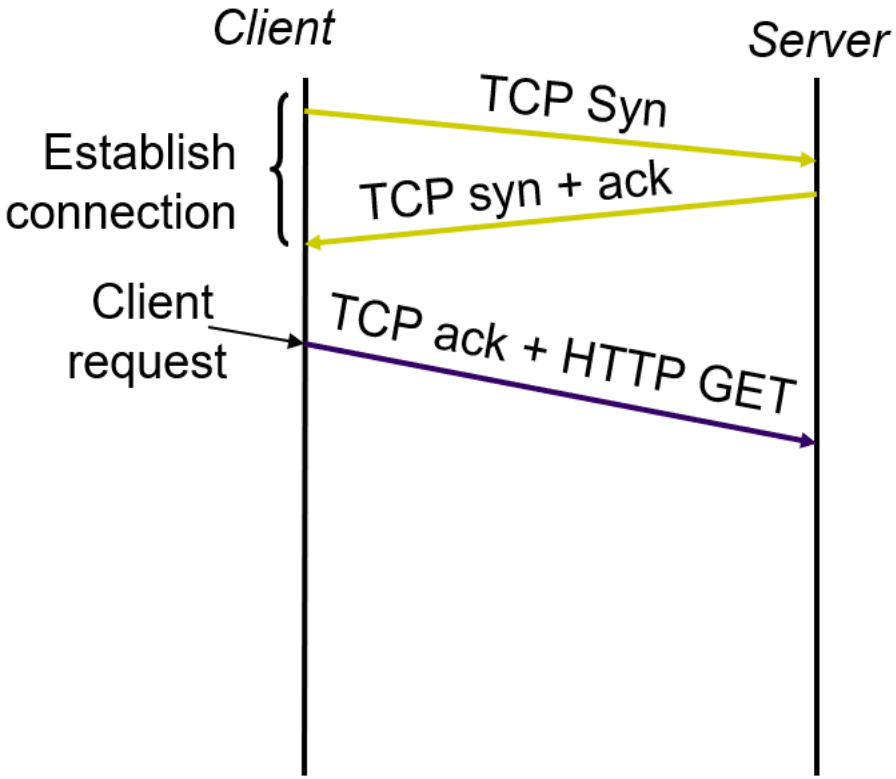


  - HTTP (Hypertext Transfer Protocol): usato per trasferire ipertesti su WWW.

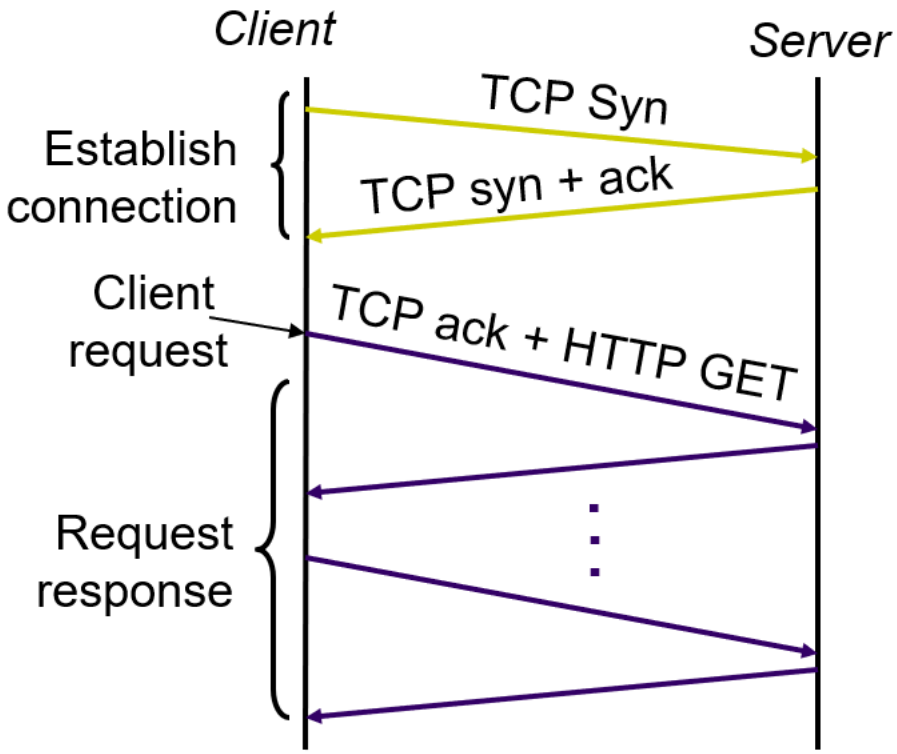
HTTP: RICHIEDERE UNA PAGINA WEB



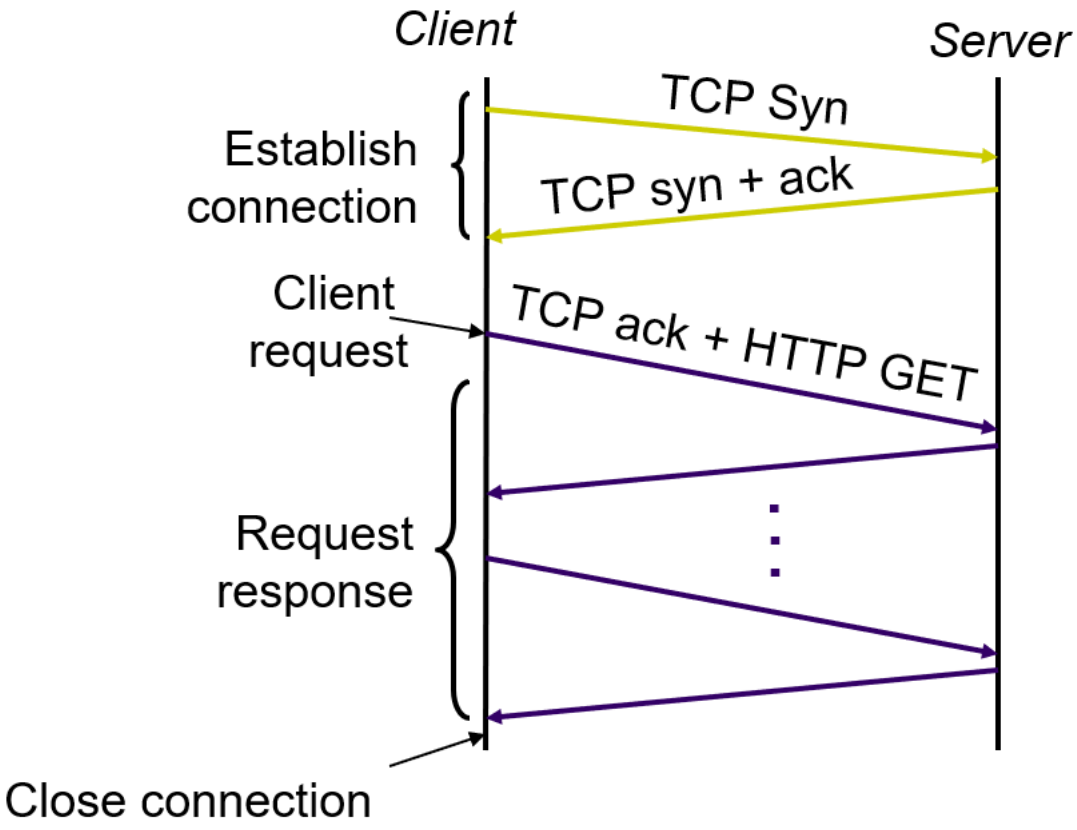
HTTP: RICHIEDERE UNA PAGINA WEB



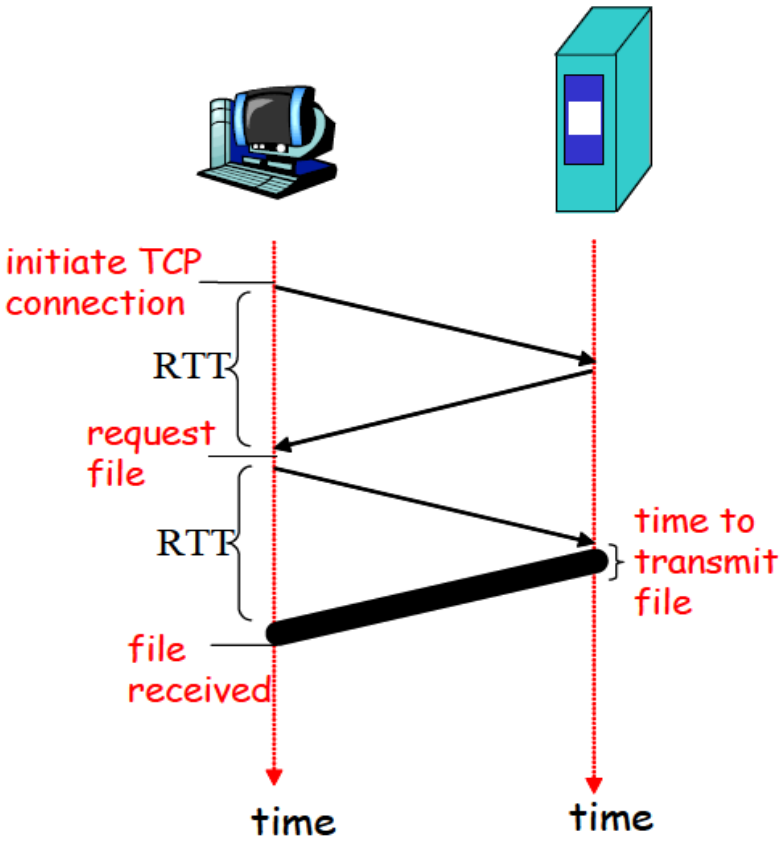
HTTP: RICHIEDERE UNA PAGINA WEB



HTTP: RICHIEDERE UNA PAGINA WEB

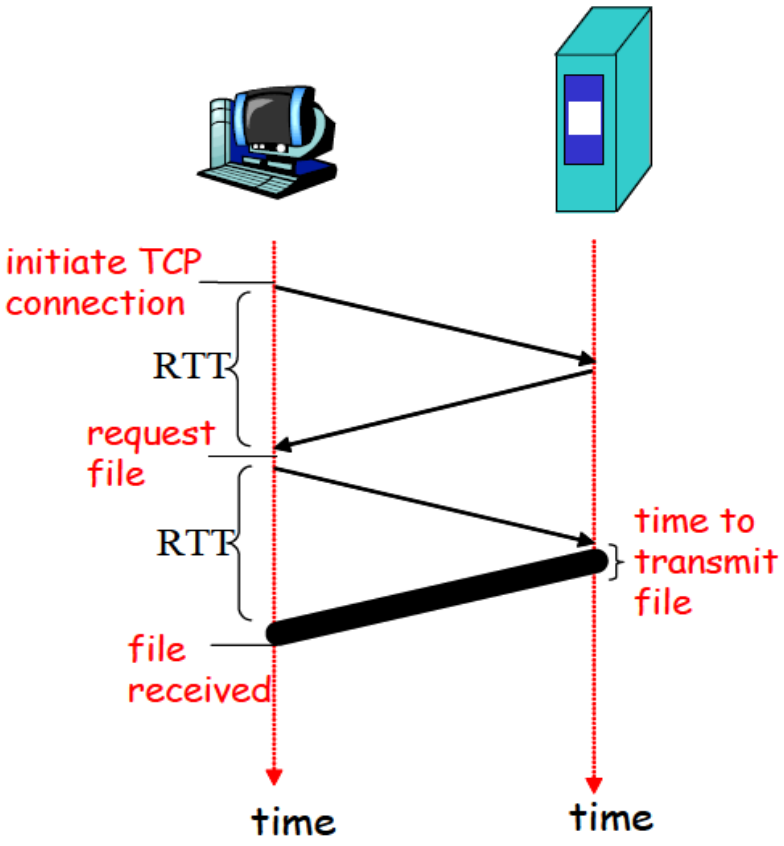


CONNESSIONI TCP NON PERSISTENTI HTTP/1.0



- solo un messaggio di richiesta ed uno di risposta per connessione TCP
- protocollo
  - il cliente apre una connessione TCP ed invia la richiesta
  - il server invia una risposta e chiude la connessione
  - il client esamina l'oggetto ricevuto: se contiene riferimenti ad altri oggetti, i passi precedenti vengono ripetuti per ogni oggetto
  - per reperire una pagina Web con n immagini
    - (n+1) connessioni TCP aperte e chiuse

CONNESSIONI TCP NON PERSISTENTI: HTTP/1.0



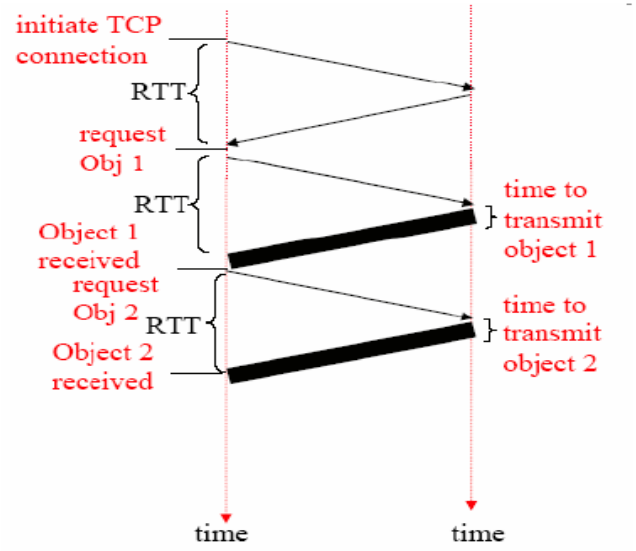
svantaggi

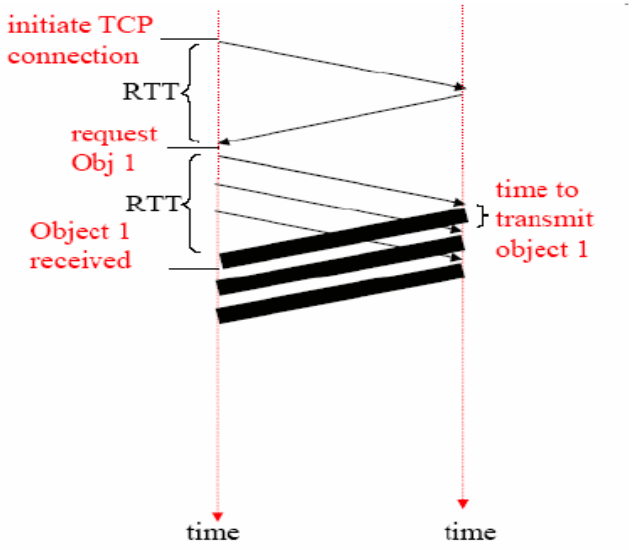
- overhead sul server allocazione e deallocazione dei buffer TCP
- slow start
  - three way end shake ripetuto per ogni connessione
  - appesantimento del tempo di reperimento dei dati

retrieval time per object = 2*RTT + transmission time

RTT = propagation + queueing + processing delay

CONNESSIONI TCP PERSISTENTI: HTTP/1.1





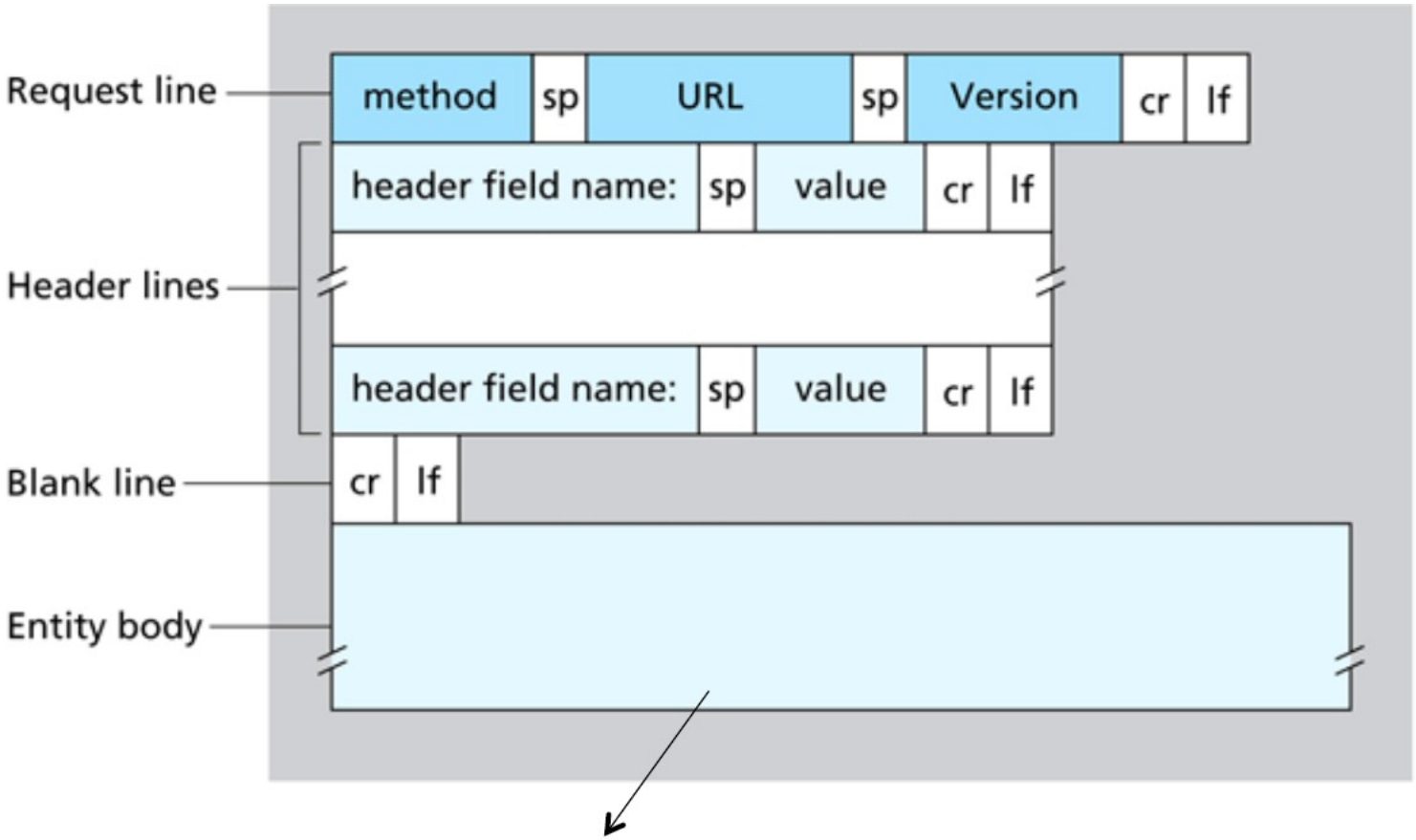
- il server lascia aperta la connessione TCP dopo aver inviato la risposta
- le richieste successive possono essere spedite sulla stessa connessione (default protocol)
- 2 versioni:
  - senza pipelining: il client HTTP manda una nuova richiesta solo quando la precedente è stata ricevuta

retrieval time per object = RTT + transmission time

  - con pipeling: il client HTTP invia una richiesta appena trova un riferimento ad un nuovo oggetto

one RTT for all objects

FORMATO GENERALE MESSAGGI HTTP

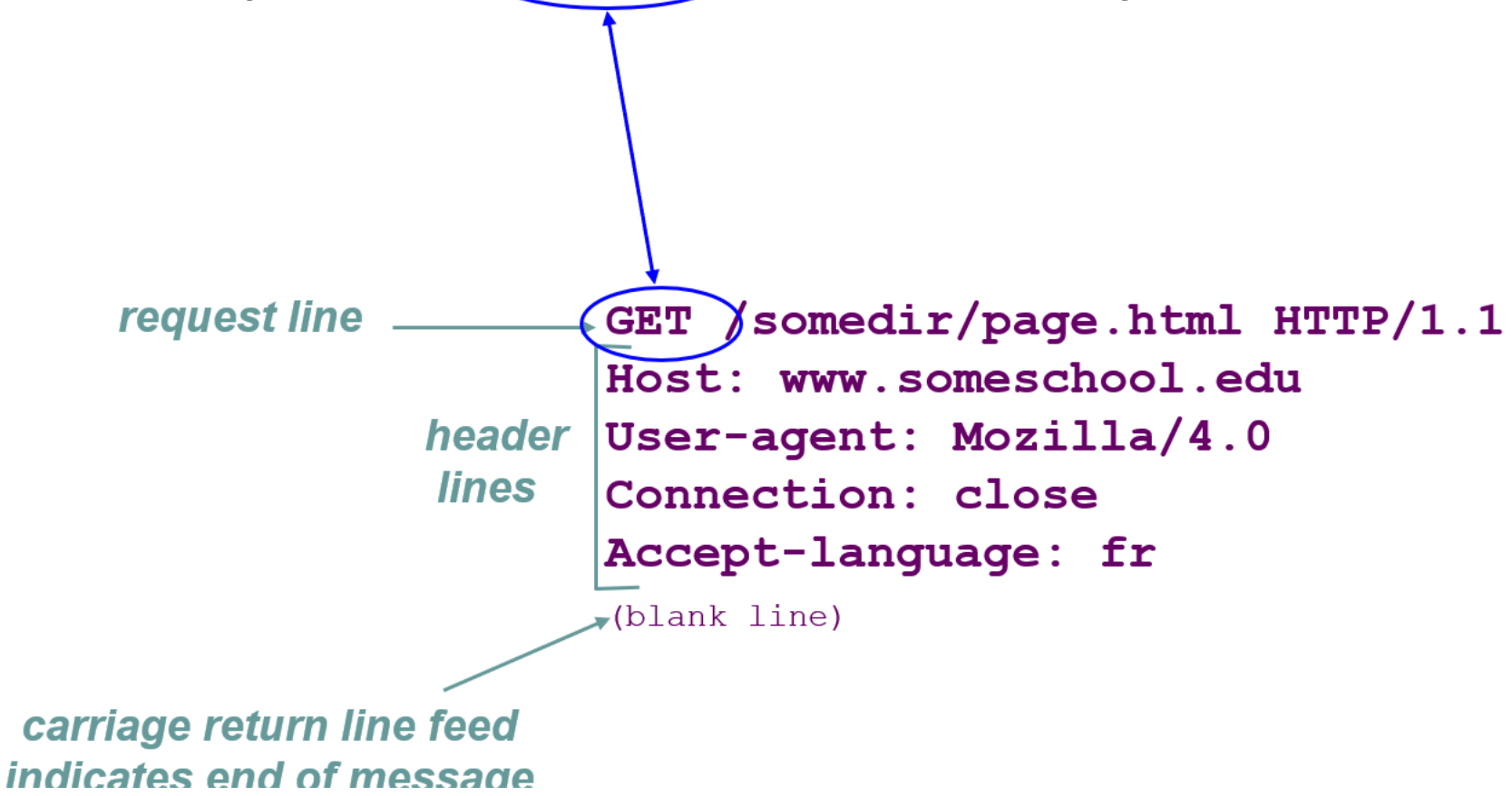


Campo vuoto per il GET, utilizzato per il POST

ASCII: formato human readable

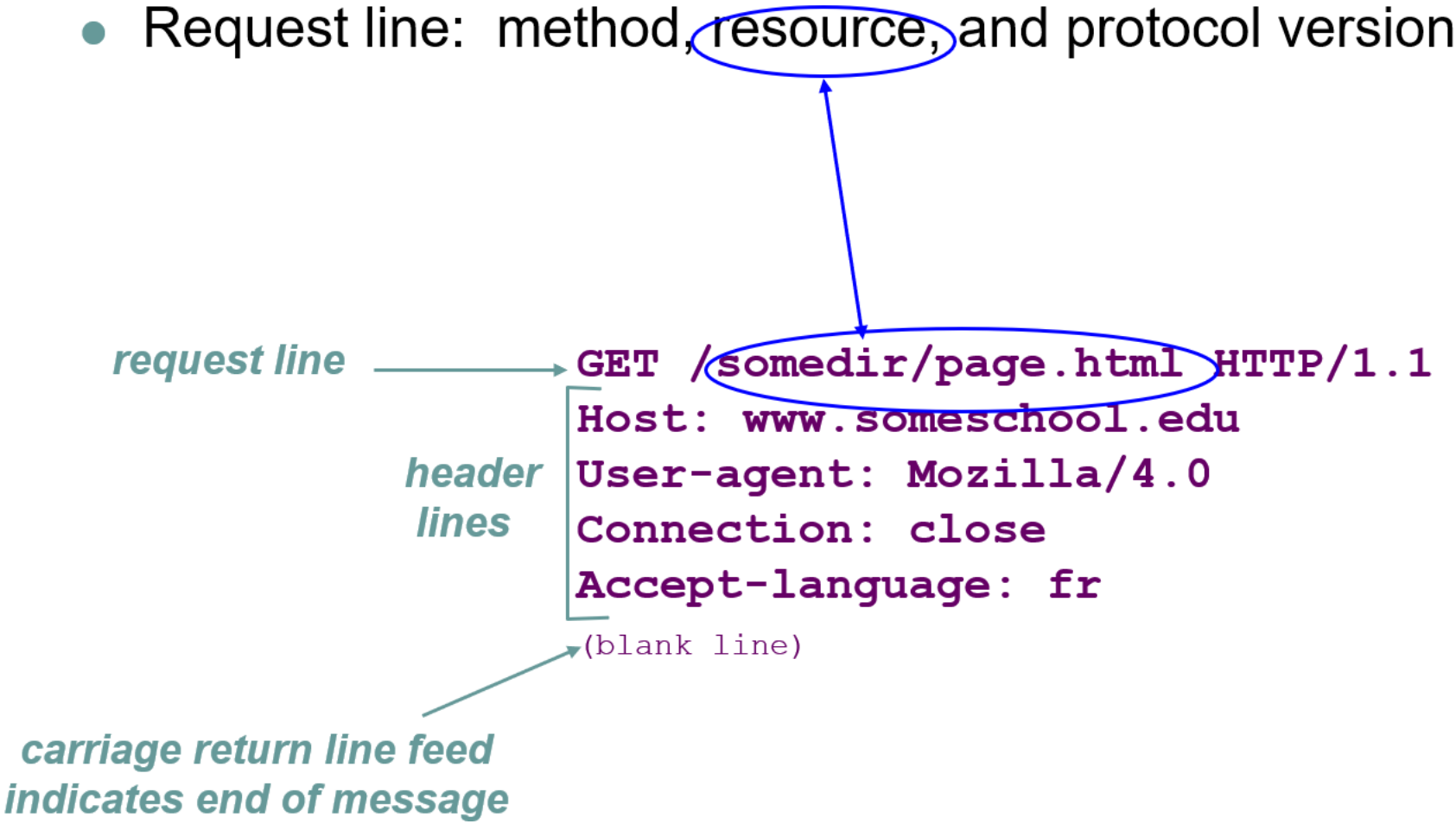
RICHIESTA HTTP

- HTTP Request Message
  - Request line: method, resource, and protocol version



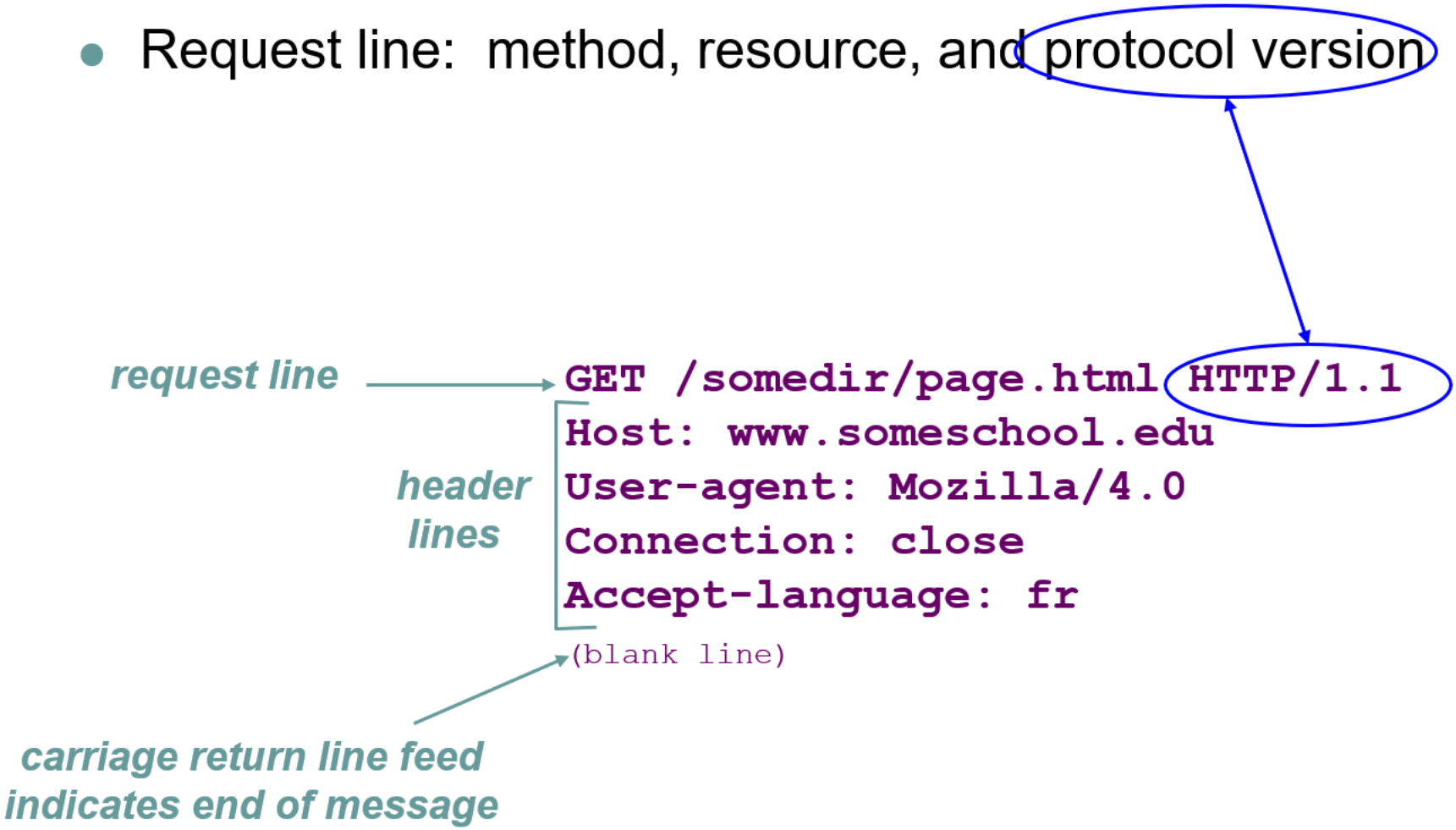
RICHIESTA HTTP

- HTTP Request Message
  - Request line: method, resource, and protocol version



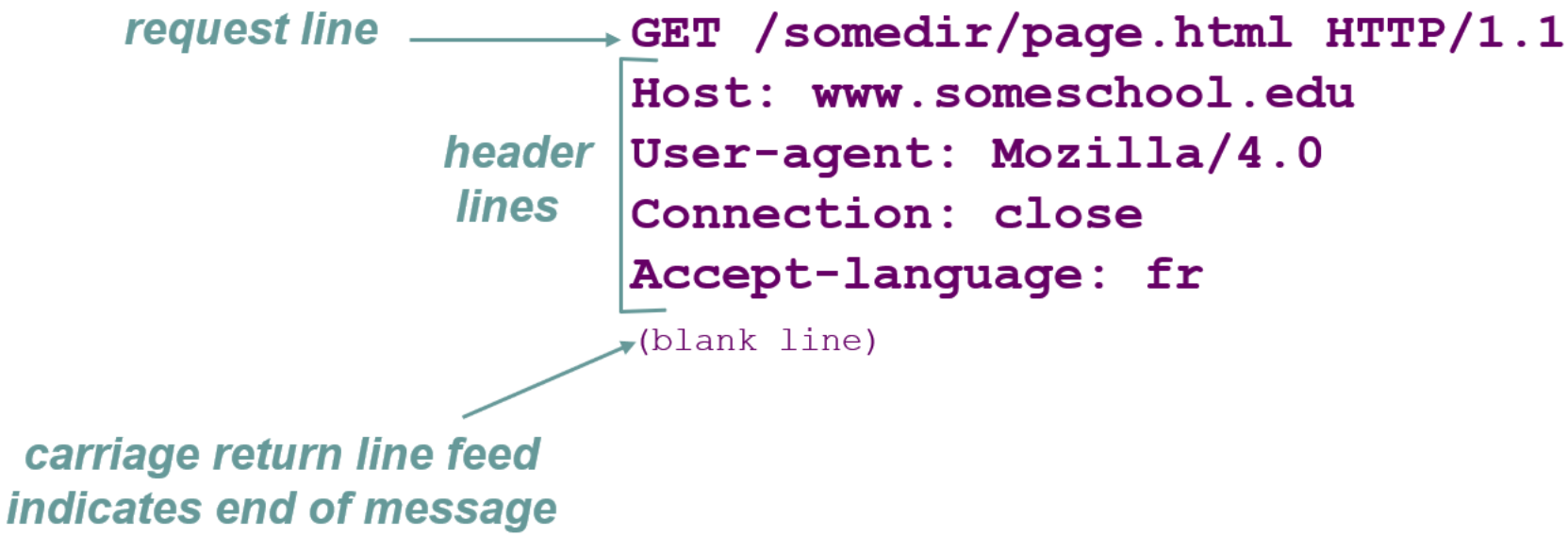
RICHIESTA HTTP

- HTTP Request Message
  - Request line: method, resource, and protocol version

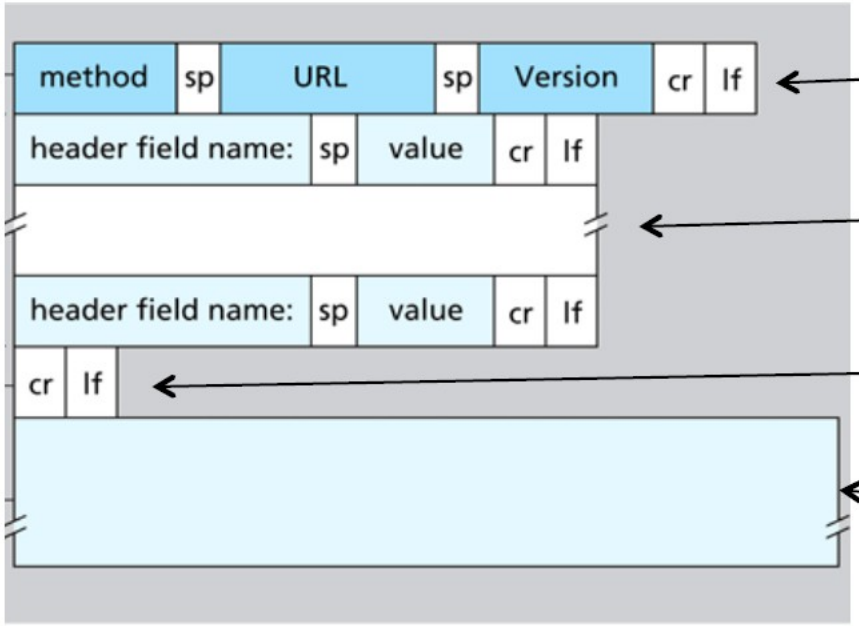


RICHIESTA HTTP

- Request line: method, resource, and protocol version
- Request headers: provide information or modify request



RICHIESTA E FORMATO MESSAGGI




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

(carriage return e line feed extra)

-- empty entity body --

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

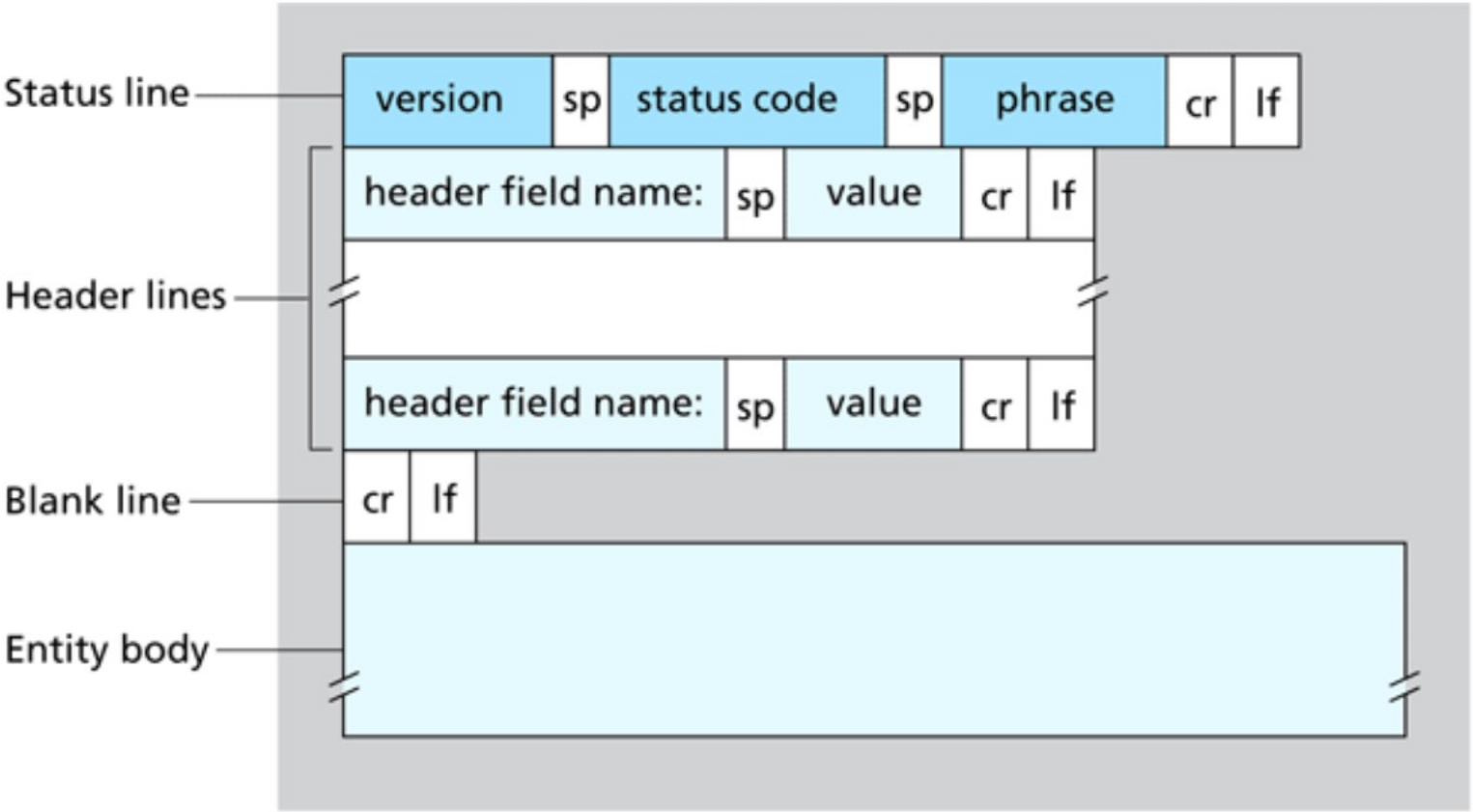
E' possibile anche usare GET

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

LE RIGHE DI INTESTAZIONE RICHIESTA

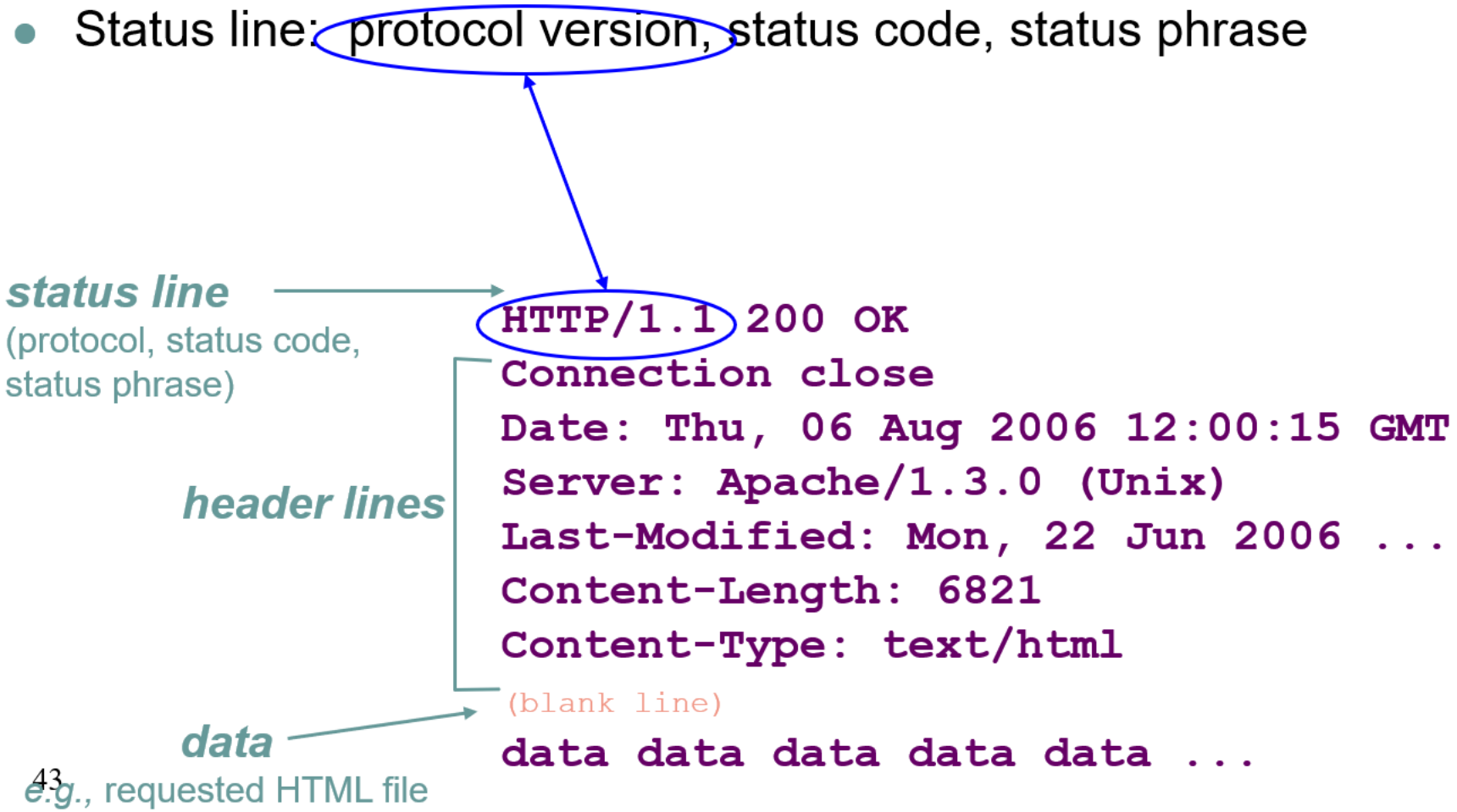
Intestazione	*Descrizione*
User-agent	Indica il programma client utilizzato
Accept	Indica il formato dei contenuti che il client è in grado di accettare
Accept-charset	Famiglia di caratteri che il client è in grado di gestire
Accept-encoding	Schema di codifica supportato dal client
Accept-language	Linguaggio 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

MESSAGGIO DI RISPOSTA: FORMATO

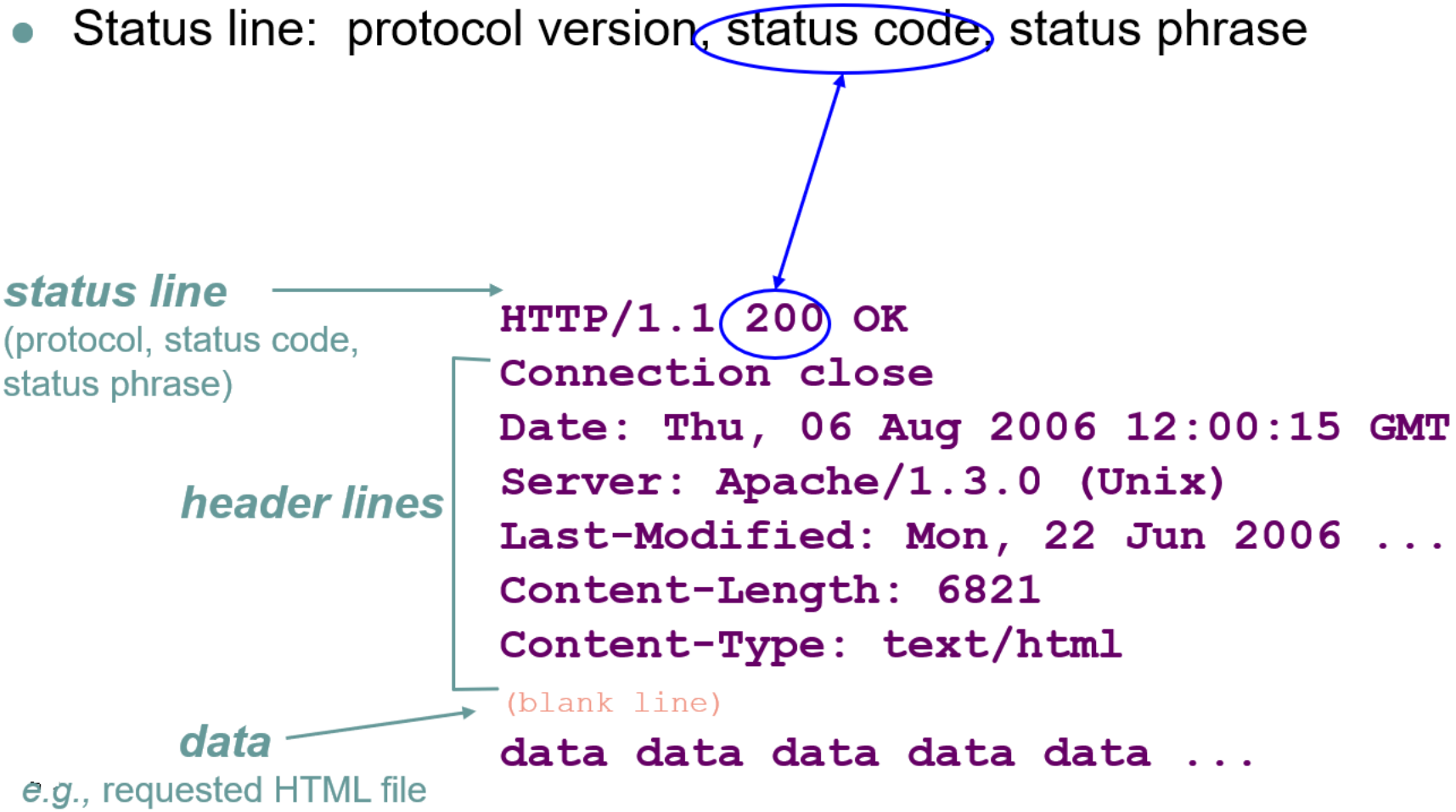


RISPOSTA HTTP

- HTTP Response Message
  - Status line: protocol version, status code, status phrase

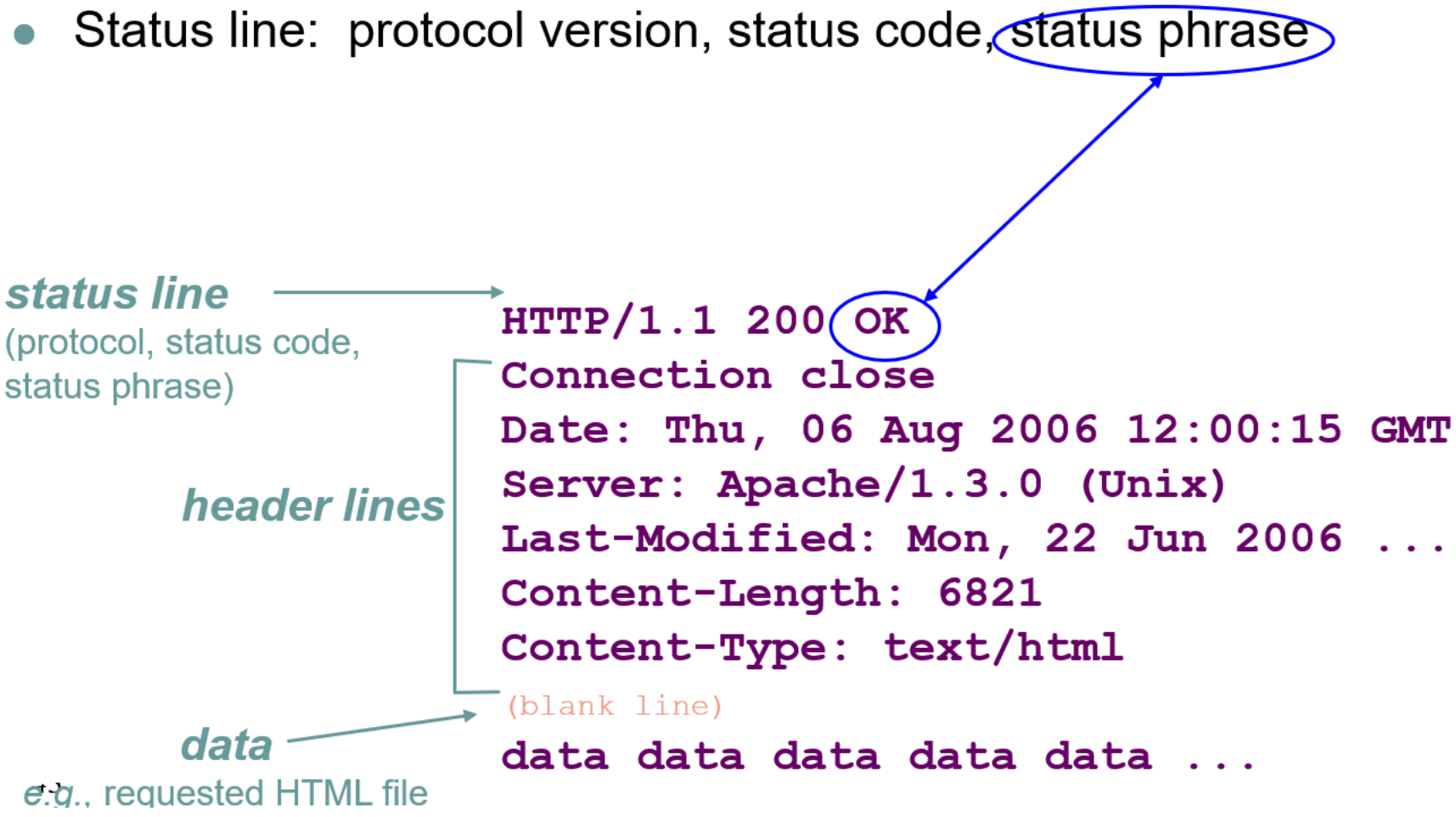


- HTTP Response Message
  - Status line: protocol version, status code, status phrase

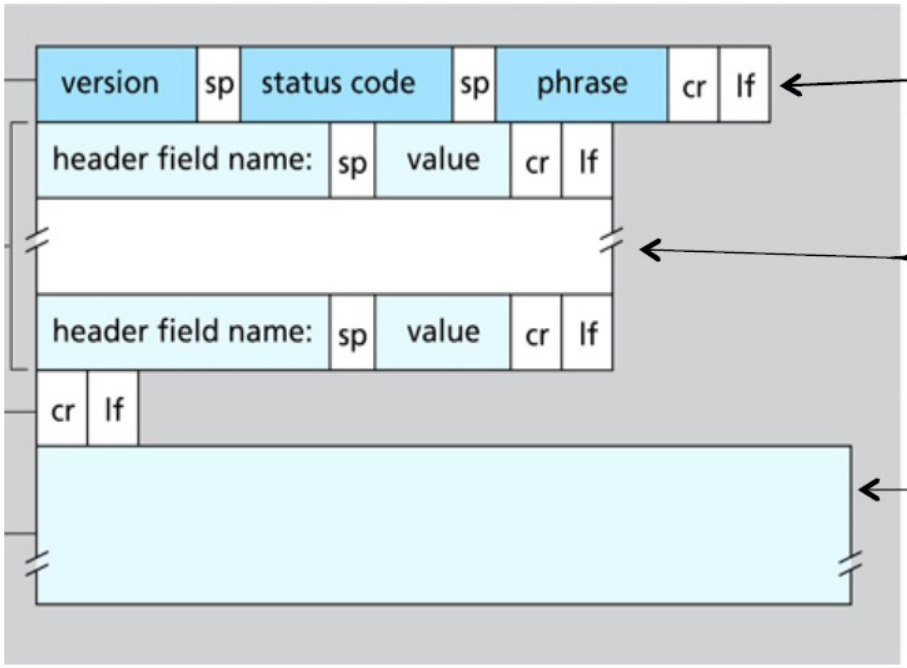


COMUNICAZIONE SERVER/CLIENT

- HTTP Response Message
  - Status line: protocol version, status code, status phrase



RISPOSTA E FORMATO MESSAGGI




```
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 ...
```

MESSAGGI DI RISPOSTA

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

Yes! →

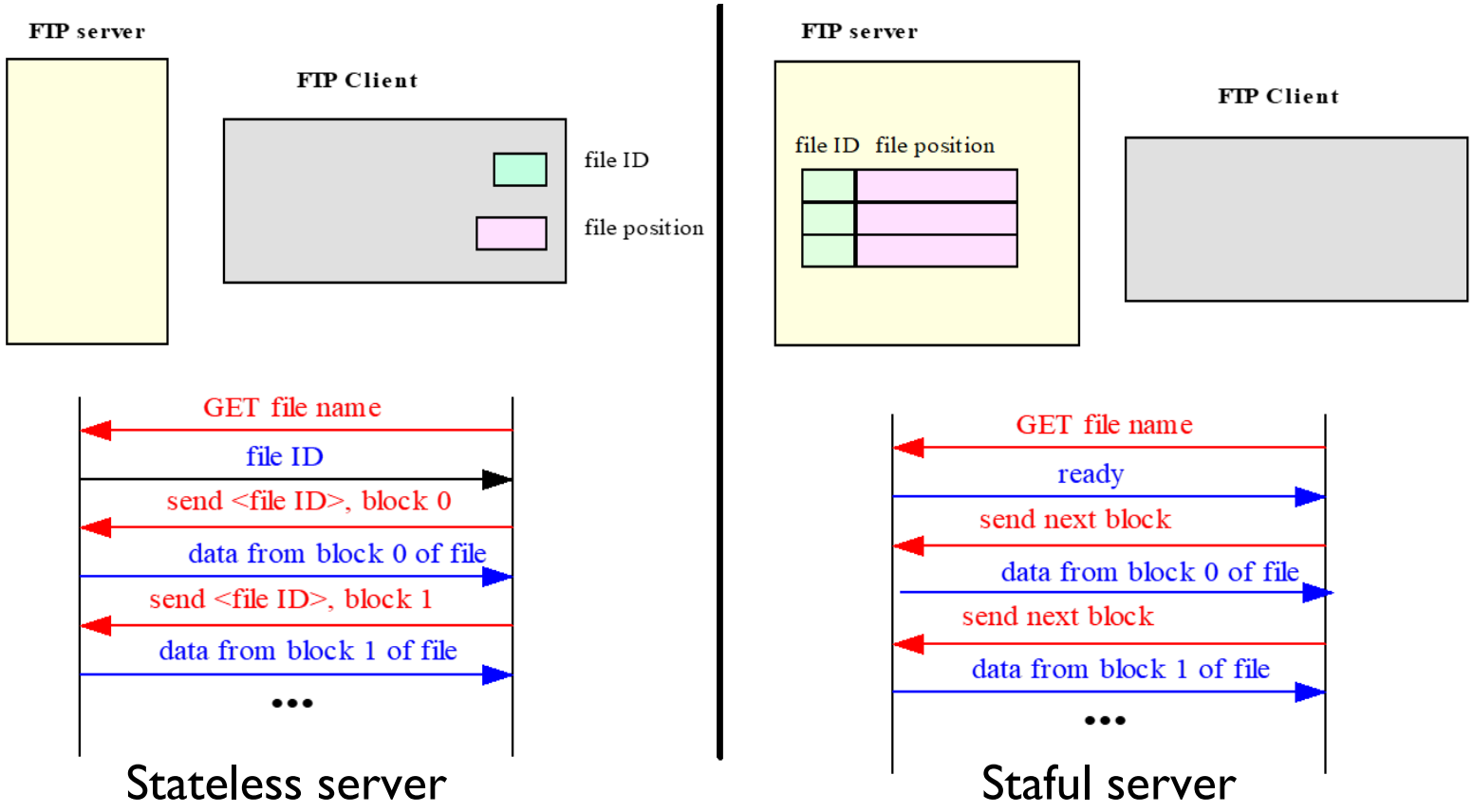
STATEFUL O STATELESS PROTOCOL?

- stateful protocol: il nodo (server) 'ricorda' informazioni da una connessione all'altra con uno stesso nodo
  - crea un "contesto" basato sulla precedente richiesta del client
    - ad esempio il server ricorda quando è avvenuta l'ultima transazione effettuata da un client, il tipo della transazione
  - vantaggi: meno informazione da inviare sulla rete, le richieste non devono essere "self contained"
  - svantaggi:
    - tutta la storia passata delle interazioni deve essere mantenuta dal nodo
    - se il nodo va in crash, lo stato può diventare inconsistente e deve essere "ricostruito"

- HTTP è un protocollo stateless

FTP: UN PROTOCOLLO STATEFUL

- il server mantiene informazioni su ogni client attivo
- mantenere lo stato può ridurre la quantità di dati scambiati e, di conseguenza, tempo di risposta



STATELESS PROTOCOL: HTTP

- nessuna informazione memorizzata dal server dopo che una transazione viene elaborata
  - vantaggio: semplicità + scalabilità + non è necessario mantenere traccia delle sessioni dei client
  - svantaggio: bandwidth overhead: ogni transazione completamente self-contained
- ci sono molti casi in cui un server ha bisogno di ricordarsi degli utenti
  - offrire contenuto personalizzato in base alle preferenze dell'utente
  - mantenere il carrello 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

STATELESS PROTOCOL: HTTP

- 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 una "shopping cart", in cui le selezioni dell'utente possono essere aggregate,
  - un giornale online può presentare contenuti personalizzati in base alle letture precedenti dell'utente.
- per informazioni dettagliate su cookie e sessioni:

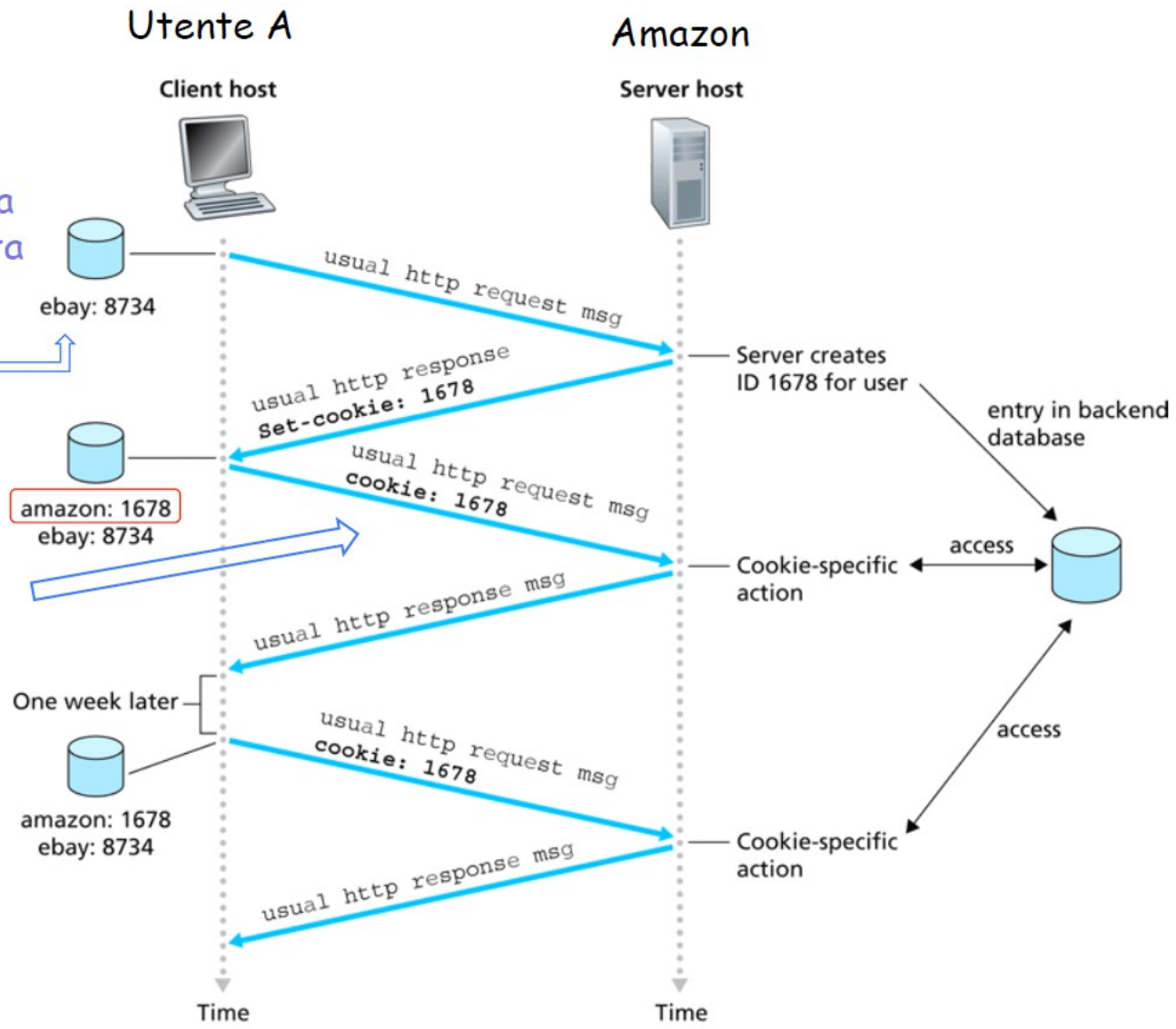
  RFC 2109 obsoleted by RFC 2965, obsoleted by RFC 6265

COOKIES

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



WEB CACHING

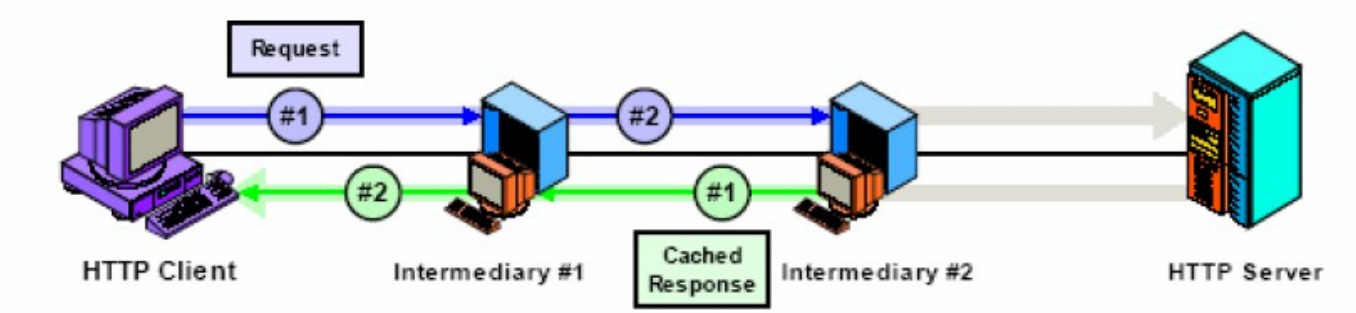
- un web cache è allocato tra il web server ed i client
  - analizza le richieste che arrivano e memorizza le risposte del server
  - quando riceve una richiesta per la solita URL, il web cache può recuperare la risposta memorizzata precedentemente e la può inviare al client
  - evita di richiedere nuovamente il contenuto al web server
- vantaggi del web caching:
  - riduce notevolmente la latenza
  - riduce il consumo di banda
    - cache degli oggetti vicino al client
    - risposte alla stessa richiesta non devono essere inviate al server
  - riduce il carico sul server: alcune delle richieste non devono essere gestite dal server
    - conseguenza: migliorare il tempo di risposta per i contenuti non reperiti nella cache

CATEGORIE DI WEB CACHING: BROWSER

- molti web browser supportano il caching
- quando una pagina web viene ritrovata per la prima volta, il browser salva quella pagina sul disco locale
- se la pagina viene richiesta di nuovo o, più frequentemente, se l'utente clicca il tasto “back”, può essere utilizzata la pagina nella cache
  - talvolta il browser contatta il web server remoto per verificare che la pagina web sia ancora valida. Se lo è, viene usata la versione locale
- Esempi: Firefox, IE

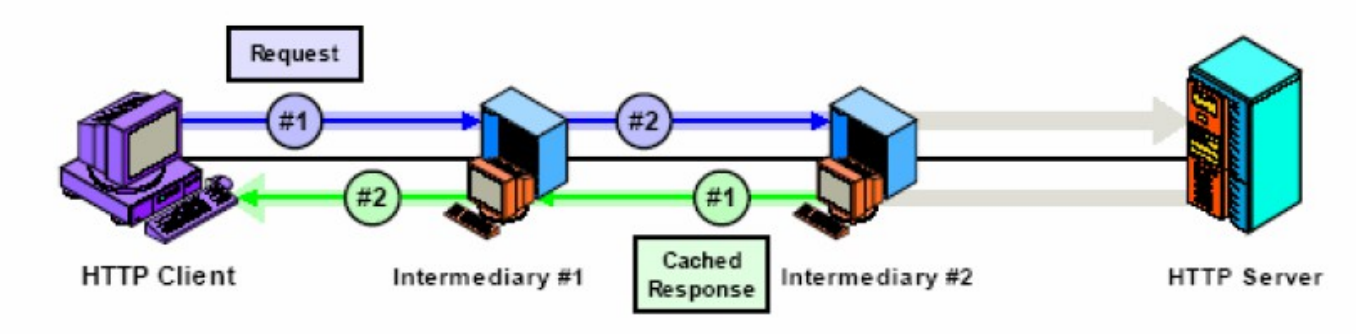
CATEGORIE DI WEB CACHING: PROXY SERVER

- una entità intermediaria che soddisfa le richieste HTTP per conto del Web server.
- il browser dell'utente deve essere configurato in modo che tutte le richieste HTTP siano prima dirette al Web cache locale.
- il Web cache controlla se possiede una copia dell'oggetto richiesto
  - in caso positivo, invia l'oggetto al client
  - in caso negativo, lo richiede al server originario



CATEGORIE DI WEB CACHING:PROXY SERVER

- vantaggi del proxy caching: ridurre
  - il tempo di risposta per le richieste del client
  - il traffico su il link di accesso ad una certa istituzione
- svantaggi del caching
  - diminuzione della performance se l'oggetto non è nella cache
  - costo di aggiunta di un ulteriore livello alla struttura



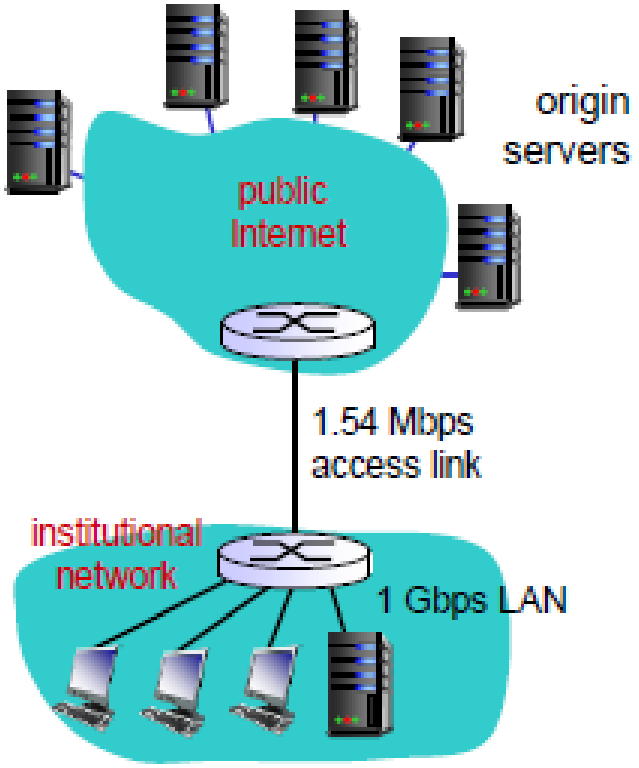
USO DI PROXY: UN ESEMPIO

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 99% *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs



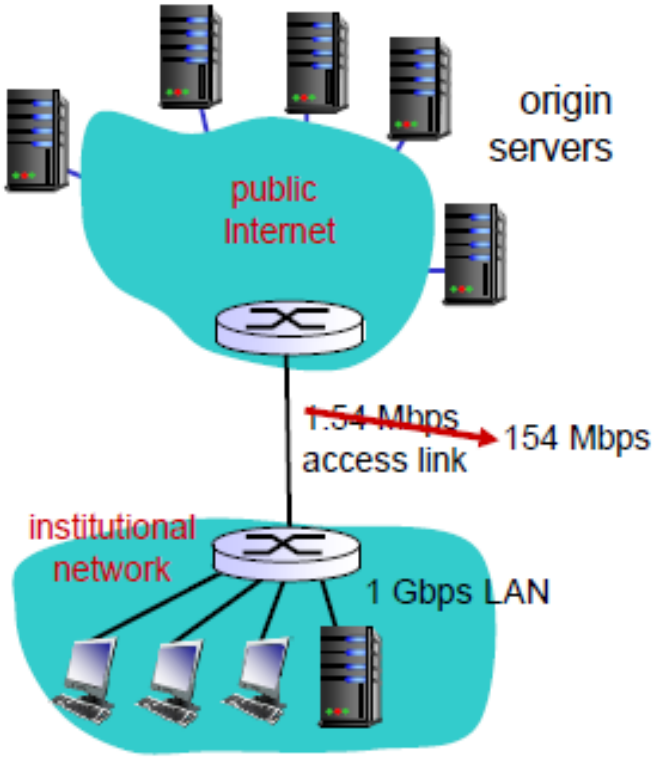
USO DI PROXY: AUMENTARE LA BANDA DEL LINK

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs
  msecs



Cost: increased access link speed (not cheap!)

USO DI PROXY: INSTALLARE UN CACHE LOCALE

assumptions:

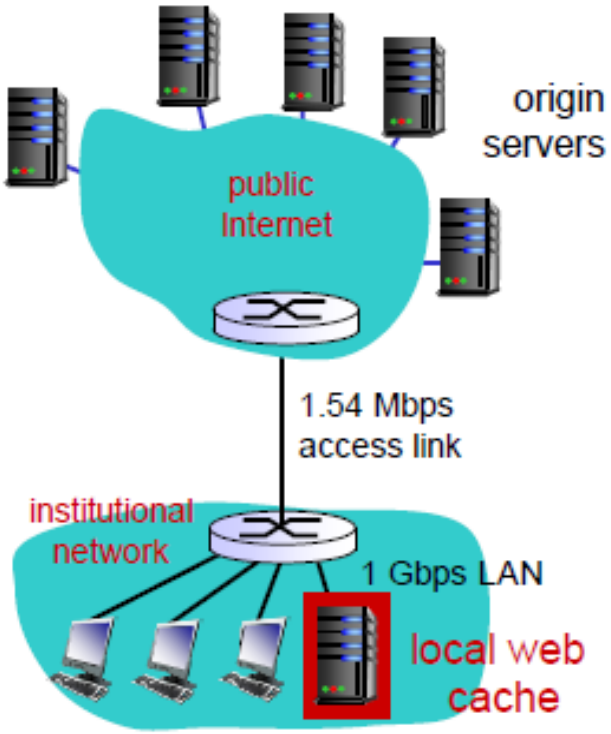
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



USO DI PROXY: UN ESEMPIO

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link
  = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)