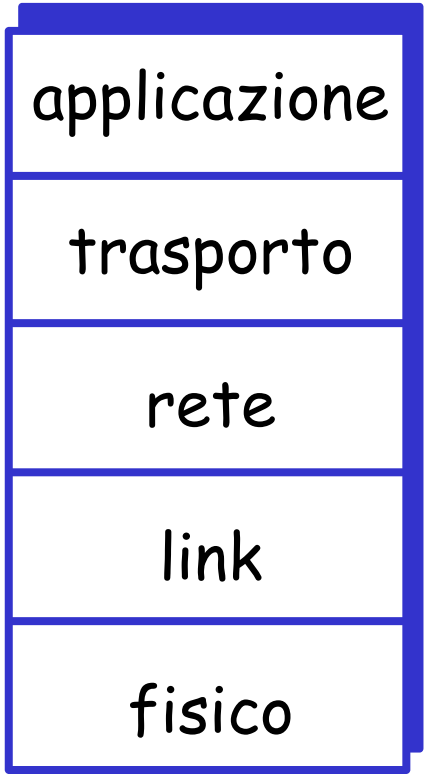
Lo strato Applicativo
URI-HTTP

Reti di Calcolatori
AA. 2019-2020

Docente: Federica Paganelli
Dipartimento di Informatica
federica.paganelli@unipi.i

Pila protocollare di Internet

- applicazione: supporta le applicazioni di rete
  - ftp, smtp, http
- trasporto: trasferimento dati host-host
  - tcp, udp
- rete: instradamento dei datagrammi dalla sorgente alla destinazione
  - ip, protocolli di instradamento
- link: trasferimento dati tra elementi di rete vicini
  - ppp, ethernet
- fisico: bit “sul filo”

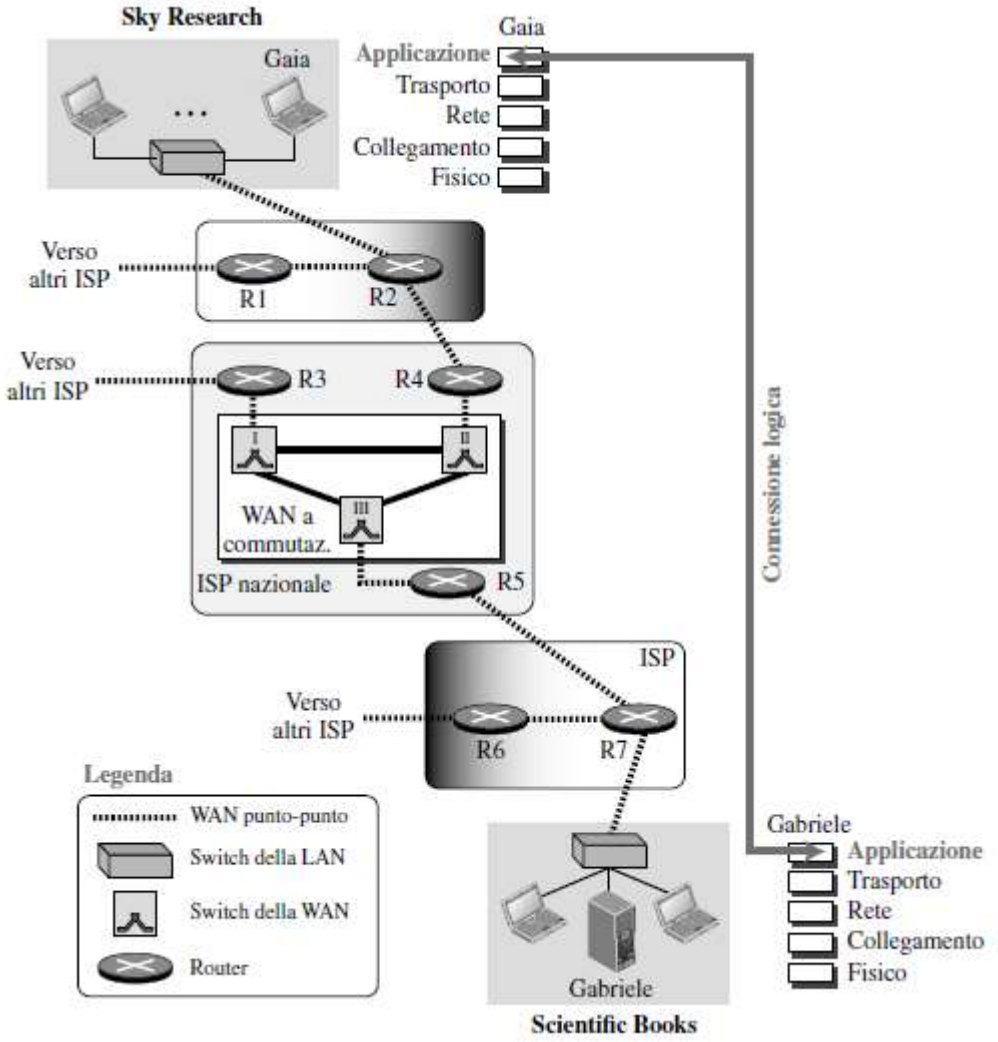


Applicazioni di rete

- Applicazioni formate da processi distribuiti comunicanti
- I processi sono programmi eseguiti dai dispositivi terminali (o host o "end system") di una rete
- All'interno dello stesso host, due processi possono anche comunicare attraverso la comunicazione inter-processo definita dal sistema operativo
- Nella comunicazione a livello applicativo fra due dispositivi terminali diversi di una rete, due o più processi girano su ciascuno degli host comunicanti e si scambiano **messaggi**.

Livello applicazione

- I livelli applicazione nei due lati della comunicazione agiscono come se esistesse un collegamento diretto attraverso il quale poter inviare e ricevere messaggi



Protocollo dello Strato di Applicazione

- Definisce i tipi di messaggi scambiati a livello applicativo (es: di richiesta e di risposta)
- la sintassi dei vari tipi di messaggio (i campi del messaggio)
- la semantica dei campi (significato)
- le regole per determinare quando e come un processo invia messaggi o risponde ai messaggi

Paradigmi del livello applicazione

- Programmi applicativi su host diversi che comunicano tra di loro scambiandosi messaggi
- Es. WEB, gestione di un elaboratore remoto, trasferimento e condivisione file, posta elettronica, comunicazione multimediale

- I due programmi applicativi devono essere entrambi in grado di richiedere e offrire servizi, oppure ciascuno deve occuparsi di uno dei due compiti?

- Paradigmi
  - Client-server:
    - numero limitato di processi server che offrono un servizio e sono in esecuzione, in attesa di ricevere richieste
    - Client: programma che richiede un servizio
  - Peer-to-peer: peer che possono offrire servizi e inviare richieste
  - Misto

Il paradigma client-server

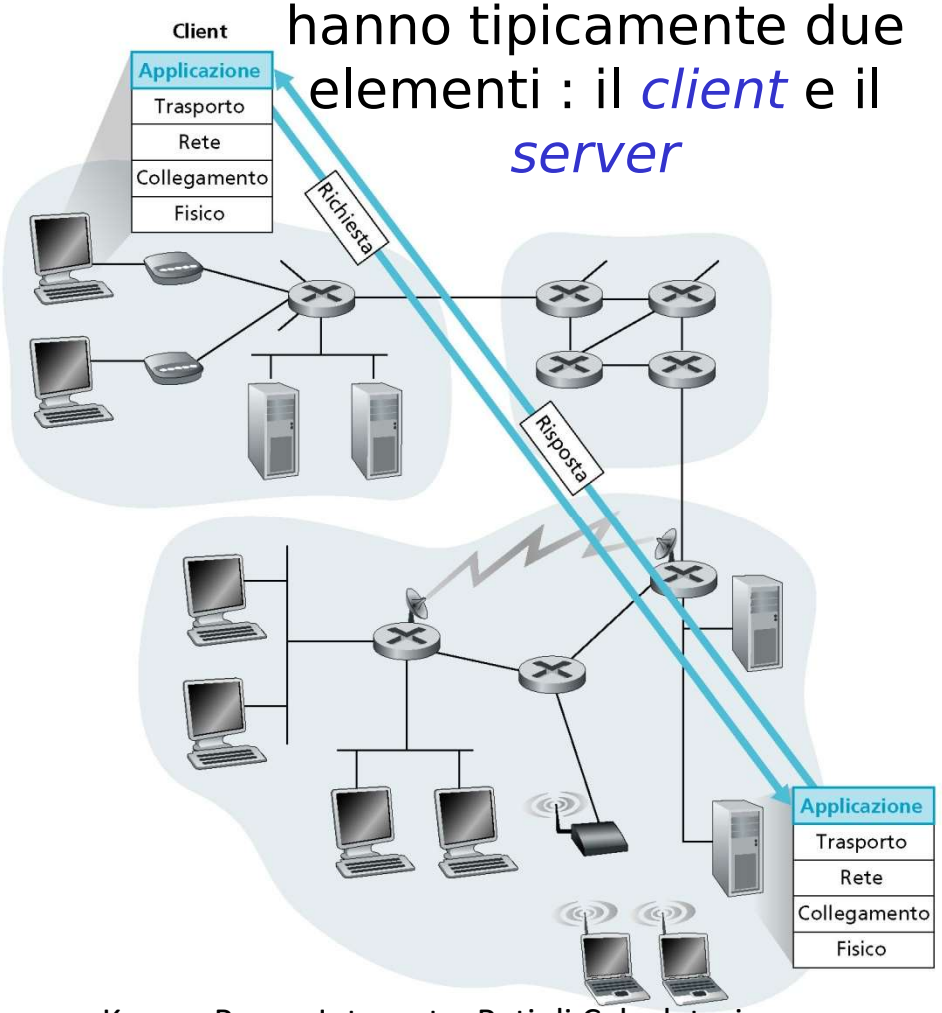
Client:

- inizia il contatto con il server (“parla per primo”)
- tipicamente richiede il servizio al server
- es.: per il Web, il client è implementato nel browser; per l’e-mail, nel mail reader

Server:

- fornisce al client il servizio richiesto
- Sempre attivo
- es.: i Web server inviano le pagine Web richieste, il mail server invia le e-mail

Le applicazioni di rete hanno tipicamente due elementi : il *client* e il *server*



Kurose Ross – Internet e Reti di Calcolatori

I componenti di un'applicazione di rete: due esempi

Web

- Browser sul client
- Server Web
- Standard per il formato dei documenti (risorse)
- Protocollo HTTP

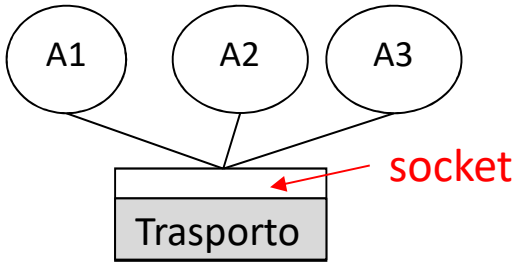
Posta elettronica

- Standard per il formato dei messaggi
- Programmi di lettura/scrittura sul client
- Server di posta di Internet
- Protocolli SMTP, POP3, ecc.

Applicazioni di rete: terminologia

API: application programming interface

- Insieme di regole che un programmatore deve rispettare per utilizzare delle risorse.

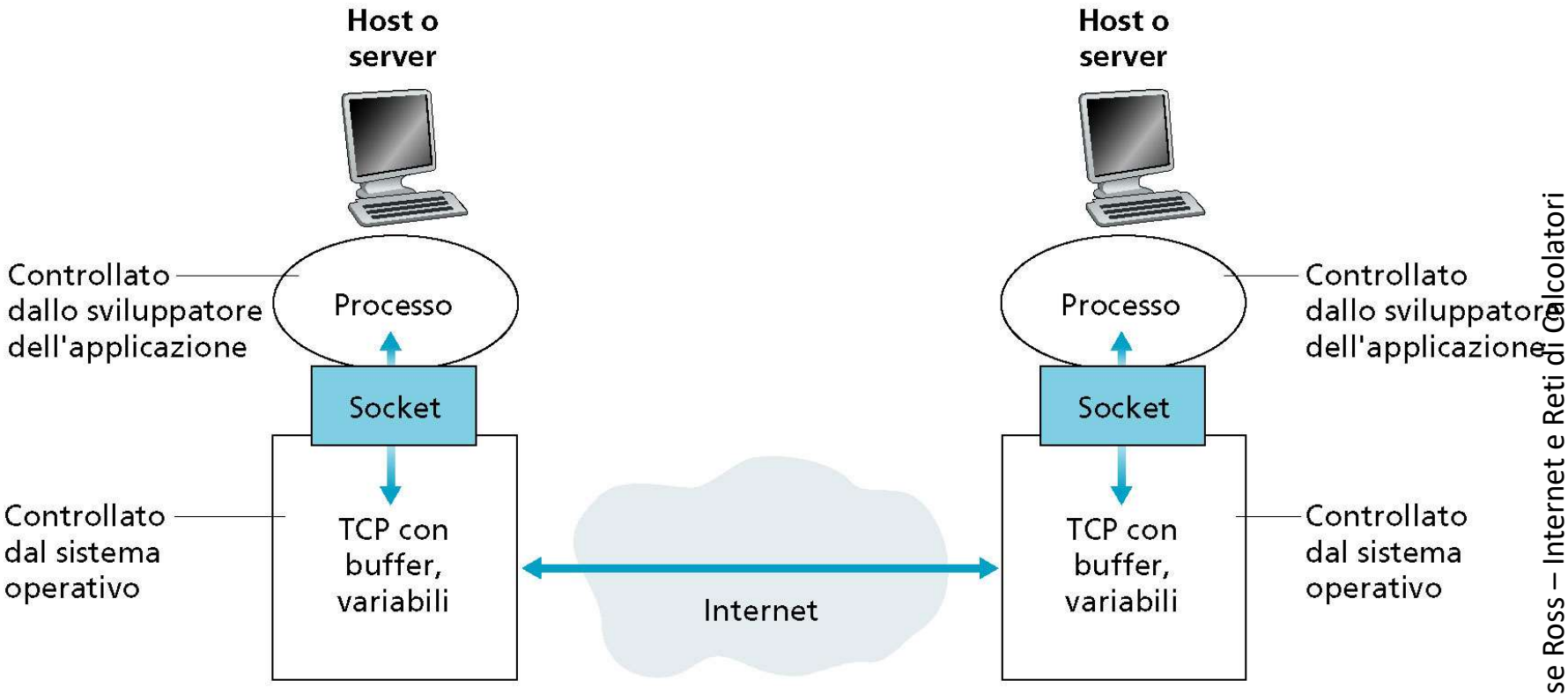


Interfaccia Socket:

API che funge da interfaccia tra gli strati di applicazione e di trasporto

- E' la API di Internet per eccellenza
- due processi comunicano mandando dati alla socket, e leggendoli da questa
- Connessione logica
- ->invio/ricezione dei dati responsabilità del sistema operativo e TCP/IP

Identificazione di un processo

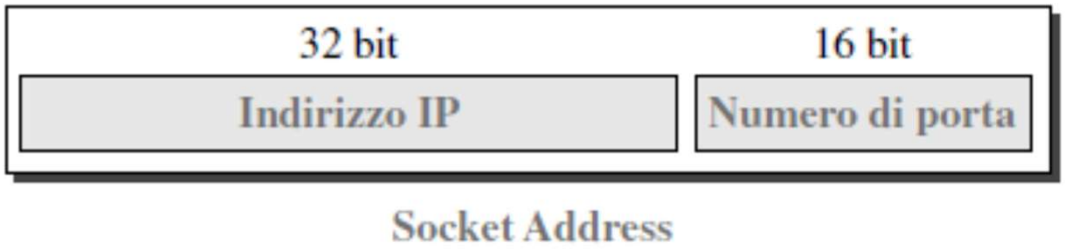


Identificazione di un processo

- servizi trasporto offerti a livello application mediante API
- ogni servizio transport usato (simultaneamente) da più processi application
- Q: Come identificare processi di livello application (di host diversi)?

-> Serve un identificativo che identifichi sia l'host che il processo

Coppie <Indirizzo IP + numero di porta>



Socket Address

Esempio di API: TCP

connection **TCPopen** (IPaddress, int)	//per aprire una connessione
void **TCPsend** (connection, data)	//per spedire dati su una connessione
data **TCPreceive** (connection)	//per ricevere dati su una connessione
void **TCPclose** (connection)	//per chiudere una connessione

int **TCPbind** (int)	//per rich. assegnaz. porta su cui attendere rich. di conn.
void **TCPunbind** (int)	//per liberare una porta
connection **TCPaccept** (int)	//per attendere richieste di connessione

Note: (1) connection identificata da una quadrupla
(2) astraiamo da possibili eccezioni sollevate (e da loro trattamento)

Uso dei servizi di trasporto

- Una coppia di processi fornisce servizi agli utenti di Internet, siano questi persone o applicazioni.
- La coppia di processi, tuttavia, deve utilizzare i servizi offerti dal livello trasporto per la comunicazione, poiché non vi è una comunicazione fisica a livello applicazione.

- Nel livello trasporto della pila di protocolli TCP/IP sono previsti due protocolli principali
  - TCP – servizio connection-oriented
  - UDP – servizio connection-less

Concetti generali

Applicazioni di rete realizzate “sopra” servizi Internet di trasporto dati

*Servizio **TCP***

- Connection-oriented: setup richiesto tra client e server
- Trasporto affidabile tra processo mittente e destinatario
- Controllo del flusso: il mittente non “inonderà” di dati il destinatario
- Controllo di congestione: “strangola” il mittente quando la rete è sovraccarica
- Non offre garanzie di timing né di ampiezza minima di banda

*Servizio **UDP***

- Non orientato alle connessioni
- Trasporto NON affidabile
- NO controllo di flusso
- NO controllo congestione
- No garanzie timing né ampiezza minima di banda

Ma allora quali applicazioni usano UDP? (e perché?)

Che tipo di trasporto è richiesto da un'applicazione?

Throughput/Banda

- la frequenza alla quale il processo mittente può inviare i bit al processo ricevente
- alcune applicazioni (es. multimedia) richiedono una banda minima per essere efficaci
- altre apps ("elastic apps") usano la banda che trovano a disposizione
- Velocità di trasferimento ≠ velocità di propagazione

Perdita dei dati

- alcune applicazioni (es., audio) possono tollerare alcune perdite
- altre apps (es., file transfer, telnet) richiedono un trasferimento dati affidabile al 100%

Timing

- alcune applicazioni (es., teleconferenza, giochi interattivi) richiedono un basso ritardo per essere efficaci

Trasporto richiesto da applicazioni comuni

Applicazione	Tolleranza alla perdita di dati	Throughput	Sensibilità al tempo
Trasferimento file	No	Variabile	No
Posta elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/video in tempo reale	Sì	Audio: da 5 Kbps a 1 Mbps Video: da 10 Kbps a 5 Mbps	Sì, centinaia di ms
Audio/video memorizzati	Sì	Come sopra	Sì, pochi secondi
Giochi interattivi	Sì	Fino a pochi Kbps	Sì, centinaia di ms
Messaggistica istantanea	No	Variabile	Sì e no

Applicazioni Internet

Applicazione	Protocollo a livello applicazione	Protocollo di trasporto sottostante
Posta elettronica	SMTP [RFC 2821]	TCP
Accesso a terminali remoti	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia in streaming	HTTP (es. YouTube) RTP [RFC 1889]	TCP o UDP
Telefonia Internet	SIP, RTP, proprietario (es. Skype)	Tipicamente UDP

Applicazione

Web e HTTP

Il Web: terminologia

- Pagina Web:
  - consiste di “oggetti” indirizzati da un URL (Uniform Resource Locator)
- Le pagine Web sono solitamente formate da:
  - pagine Web (HTML, javascript,..)
  - diversi oggetti referenziati (altre pagine, immagini)
- Lo User agent per il Web è chiamato browser:
  - Firefox
  - Chrome
  - MS Internet Explorer
- Il server per il Web è chiamato Web Server:
  - Apache (open)
  - MS Internet Information Server

UNIFORM RESOURCE IDENTIFIER

Uniform Resource Identifier (URI)

- Una URI è una forma generale per identificare una risorsa presente sulla rete (vedi IETF rfc 2396)
  - *A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource.*
- *The URI syntax was designed with global transcriptability as one of its main concerns. A URI is a sequence of characters from a very limited set, i.e. the letters of the basic Latin alphabet, digits, and a few special characters. A URI may be represented in a variety of ways: e.g., ink on paper, pixels on a screen, or a sequence of octets in a coded character set. The interpretation of a URI depends only on the characters used and not how those characters are represented in a network protocol*.

Uniform Resource Identifier (URI)

- Uniform: uniformità della sintassi dell'identificatore anche se i meccanismi per accedere alle risorse possono variare
- Resource: qualsiasi cosa abbia un'identità (!)
  - Documento, servizio, immagine, collezione di altre risorse
- Identifier: oggetto che può agire da riferimento verso qualcosa che ha identità

Uniform Resource Identifier (URI)

- Ci sono due tipi di URI:
- Uniform Resource Locator (URL): *subset of URI that identify resources via a representation of their primary access* mechanism (e.g., their network "location"),
- Uniform Resource Name (URN): *subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.*

Esempi:

URN: urn:oasis:names:specification:docbook:dtd:xml:4.1.2:

URN: urn:doi:10.1109/LCN.1988.10239

URL: https://doi.org/10.1109/LCN.1988.10239
URL: ftp://ftp.is.co.za/rfc/rfc1808.txt
URL: http://www.apple.com/index.html

Uniform Resource Identifiers

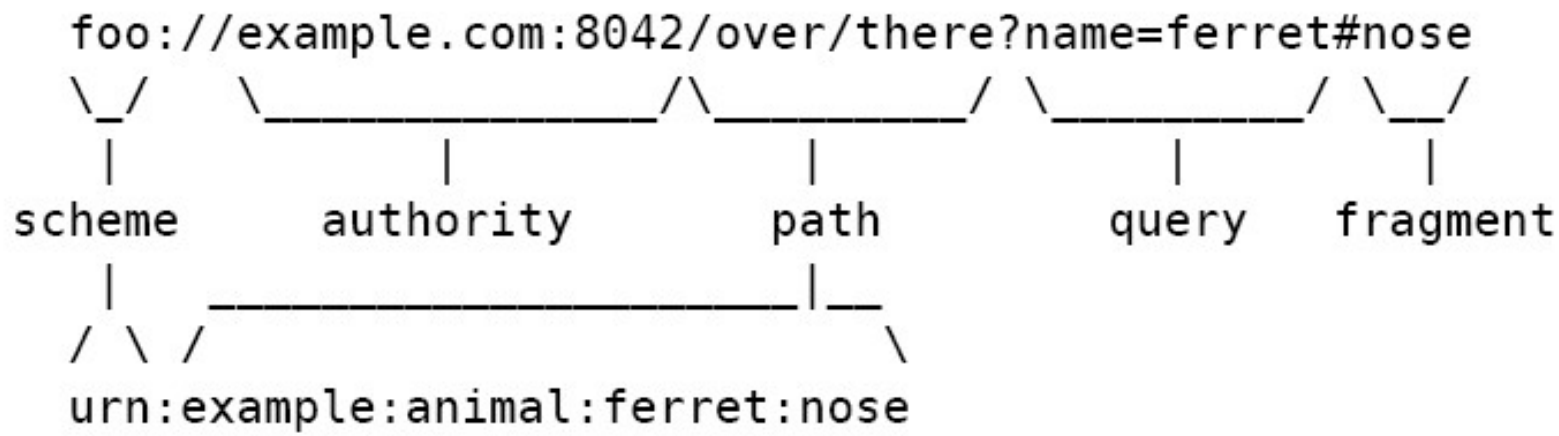
- *The URI syntax is organized hierarchically, with components listed in order of decreasing significance from left to right*.
- Una URI assoluta può essere formata da quattro componenti

<scheme>://<authority><path>?<query>

  - <scheme> obbligatorio, schema per identificare la risorsa. The URI scheme (Section 3.1) defines the namespace of the URI, and thus may further restrict the syntax and semantics of identifiers using that scheme. Although many URL schemes are named after protocols, this does not imply that the only way to access the URL's resource is via the named protocol
  - <authority> *hierarchical element for a naming authority so that governance of the name space defined by the remainder of the URI is delegated to that authority*. Il nome di dominio di un host (reg_name) o il suo indirizzo IP in notazione decimale puntata (server)

    authority = [userinfo "@"] host [":" port]
  - <path> contiene dati specifici per l'authority (o scheme) e identifica la risorsa nel

Esempi



scheme authority path query

http://maps.google.it/maps/place?q=largo+bruno+pontecorvo+pisa&hl=it

URI Assolute e Relative

- Le URI possono essere assolute o relative

- URI assoluta: identifica una risorsa indipendentemente dal contesto in cui è usata

- URI relativa: informazioni per identificare una risorsa in relazione ad un'altra URL (è priva dello schema e della authority).

URI Relativa

Sia http://a/b/c/d;p?q il documento di partenza, allora:

```
g       =   http://a/b/c/g
/g      =   http://a/g
//g     =   http://g
?y      =   http://a/b/c/?y
#s          =   (current document)#s
g;x?y#s         =   http://a/b/c/g;x?y#s
..      =   http://a/b/
../../g     =   http://a/g
```

Le URI Relative…

- NON viaggiano sulla rete
- Sono interpretate dal browser in relazione al documento di partenza

HYPERTEXT TRANSFER PROTOCOL

HTTP: HyperText Transfer Protocol

- Usato dal 1990 come protocollo di trasferimento per il World Wide Web, è definito nel seguente modo:
  - "protocollo di livello applicazione per sistemi di informazione distribuiti, collaborativi ed ipermediali" (RFC 2068 and RFC 2616)
  - *It is a **generic**, **stateless**, **object-oriented** protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing of data representation, allowing systems to be built independently of the data being transferre*d
  - permette di costruire sistemi di accesso all'informazione indipendenti dal tipo dell'informazione stessa**.**

HTTP URL

- Lo schema "http" è usato per accedere alla risorsa attraverso il protocollo HTTP
- Sintassi per URL http:

http_URL = "http:" "//" host [":" port] [path]

  - host = A legal Internet host domain name or IP address (in dotted-decimal form),
  - port = *DIGIT If the port is empty or not given, port 80 is assumed.
- La risorsa è localizzata nel server in ascolto per connessioni TCP su quella porta di quell'host. Il path specifica la Request-URI (vedi messaggio di richiesta in seguito)

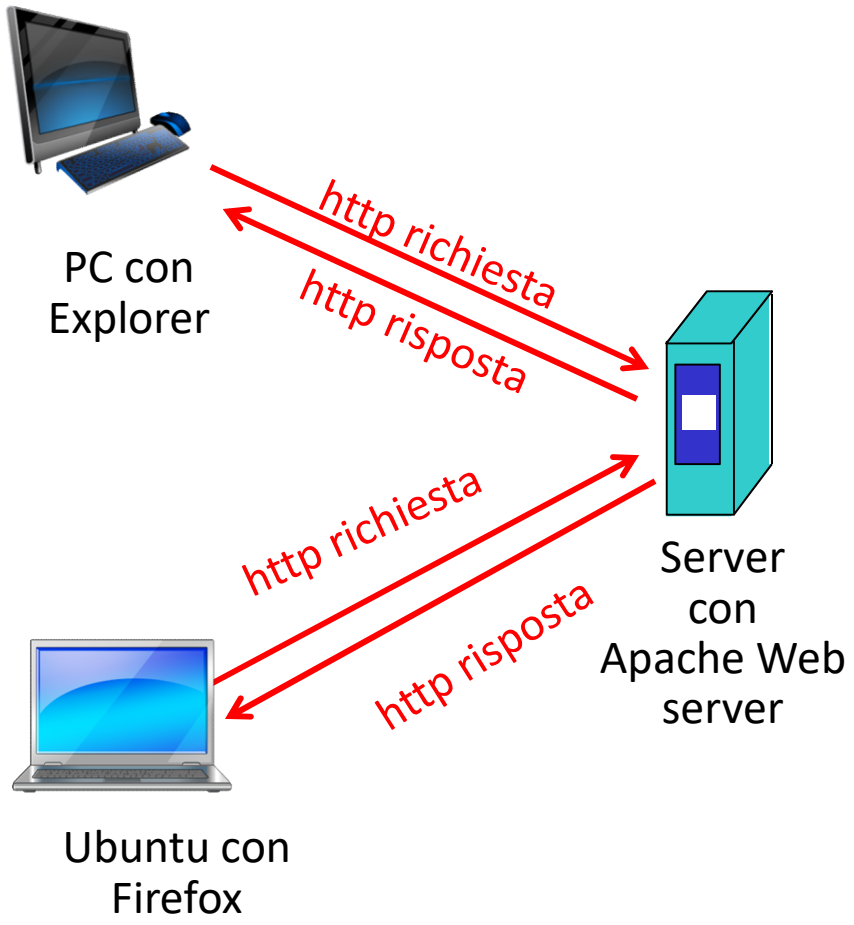
Il protocollo http

- E’ un protocollo di tipo request/response
  - La connessione viene iniziata dal client, che invia un messaggio di request.
  - Il server risponde con una response
- Protocollo generico e stateless
  - le coppie richiesta/risposta sono indipendenti
- HTTP 1.0/1.1

  La connessione viene terminata (HTTP 1.0), oppure (HTTP 1.1) si procede con un’altra coppia di request/response

Il protocollo http

- modello client/server
- *client:* browser che richiede, riceve e visualizza gli oggetti Web
- *server:* Web server che invia oggetti in risposta ad una richiesta



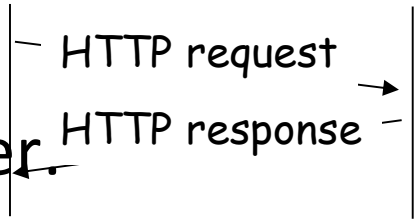
- *A client establishes a connection with a server and sends a request to the server in the form of a request method, URI, and protocol version, followed by a message.*
- *The server is an application program that accepts connections in order to serve requests by sending back responses.*

Connections

- *Connection*: Un circuito logico di livello trasporto stabilito tra due programmi applicativi per comunicare tra loro.
- *Non-persistent connection* (http1.0: RFC 1945): a separate TCP connection is established to fetch each URL
  - increasing the load on HTTP servers
  - causing congestion on the Internet (e.g. The use of inline images and other associated data often requires a client to make multiple requests of the same server in a short amount of time).
- *Persistent connection* (http1.1: RFC 2616): unless otherwise indicated, the client may assume that the server will maintain a persistent connection.
  - The standard specifies a mechanism by which a client and a server can signal the close of a TCP connection (the Connection header field).
  - Once a close has been signaled, the client MUST not send any more requests on that connection.

HTTP

- HyperText Transfer Protocol
  - tipica interazione HTTP client-server.
  - **HTTP utilizza TCP**



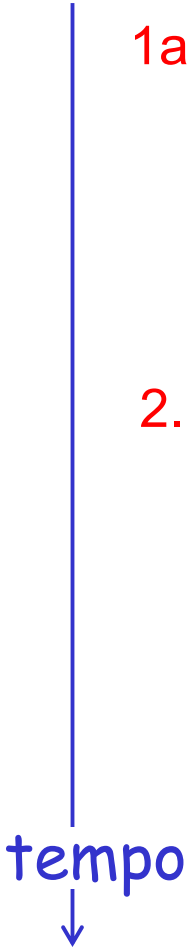

```
//esempio client
c = TCPopen(“131.115.7.24”, 80);
TCPsend(c,”GET /index.html”);
d = TCPreceive(c);
```

```
//esempio server
p = TCPbind(80);
d = TCPaccept(p);
 r = TCPreceive(d)
          ...
TCPsend(d,pag)
TCPclose(d)
```

Esempio http

Supponiamo che l'utente digiti la URL
www.someSchool.edu/someDepartment/home.index

(contiene testo, e riferimenti a 10 immagini jpg)



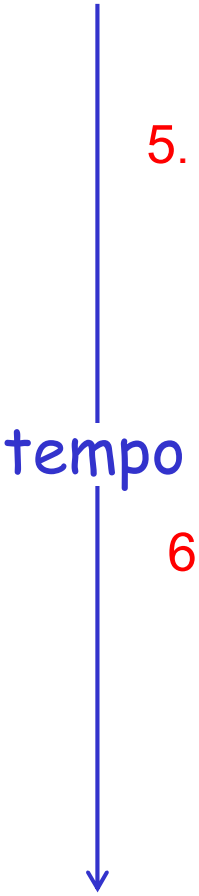
1a. Il client http inizia la connessione TCP verso il server http al www.someSchool.edu. Socket Porta 80 è default per il server http.

1b. Il server http dell'host www.someSchool.edu aspetta le richieste di connessione TCP connection alla porta 80. "Accetta" la connessione, e lo notifica al client

2. Il client http invia un *messaggio di richiesta http* (che contiene la URL) alla socket di connessione con lo strato di trasporto (TCP)

3. Il server http riceve il msg di richiesta, compila un *messaggio di risposta* che contiene l'oggetto richiesto (someDepartment/home.index), manda il messaggio alla socket

Esempio http (cont.)



4. Il server http chiude la connessione TCP

5. Il client http riceve il messaggio di risposta che contiene il file html e lo visualizza. Percorrendo il file trova il riferimento a 10 oggetti jpg

6. Ripete i passaggi da 1-5 per ognuno dei 10 oggetti jpg

Il protocollo http è "stateless"

ogni risposta è collegata solo alla richiesta che l'ha generata, indipendentemente dalle richieste precedenti

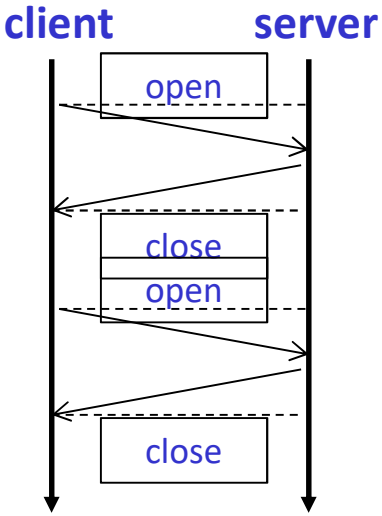
Connessione persistente (HTTP 1.1)

- La stessa connessione HTTP può essere utilizzata per una serie di richieste e una serie corrispondente di risposte
- Il server lascia aperta la connessione TCP dopo aver spedito la risposta e può quindi ricevere le richieste successive sulla stessa connessione
- Il server HTTP chiude la connessione quando viene specificato nell'header del messaggio (desiderata da parte del cliente) oppure quando non riceve richieste per un certo intervallo di tempo (time out)

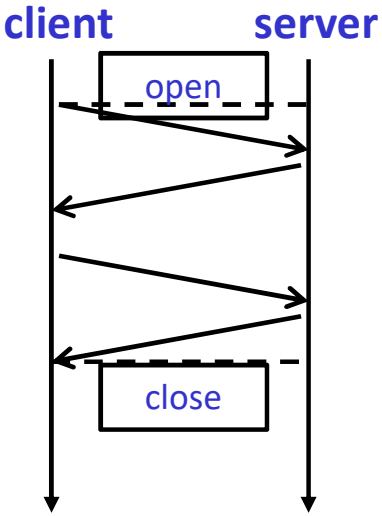
Come cambia il diagramma il flusso richieste/risposte nella slide precedente?

Pipelining

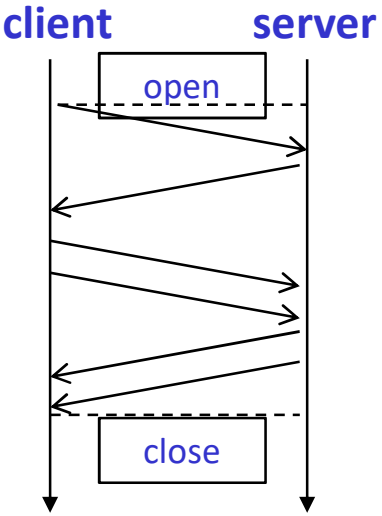
- Serve per migliorare ulteriormente le prestazioni
- Consiste nell'invio da parte del client di molteplici richieste senza aspettare la ricezione di ciascuna risposta
- Il server DEVE inviare le risposte nello stesso ordine in cui sono state ricevute le richieste
- Il client non può inviare in pipeline richieste che usano metodi HTTP non idem-potenti



HTTP v1.0



HTTP v1.1



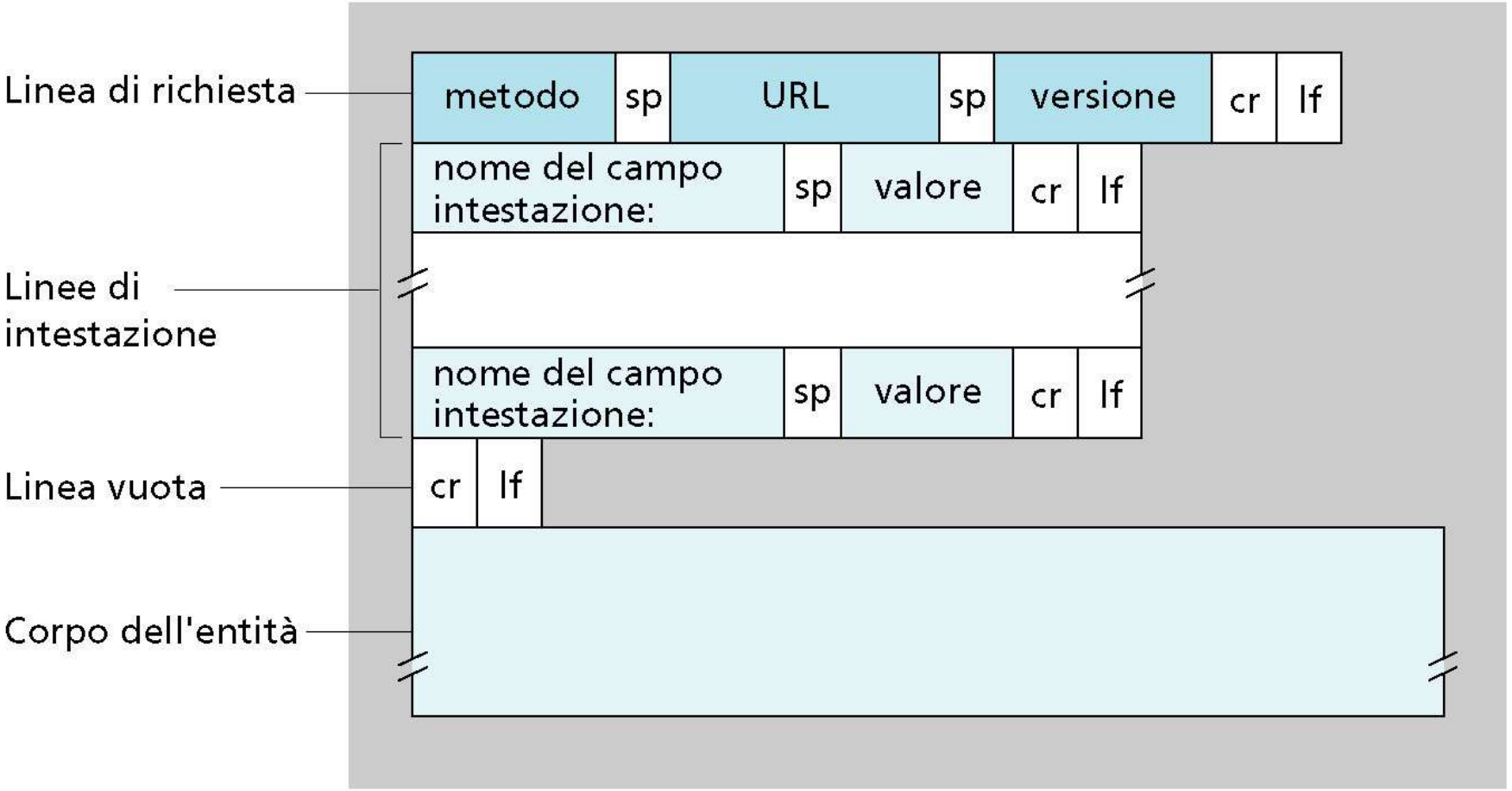
HTTP v1.1 con pipeline

Messaggi HTTP

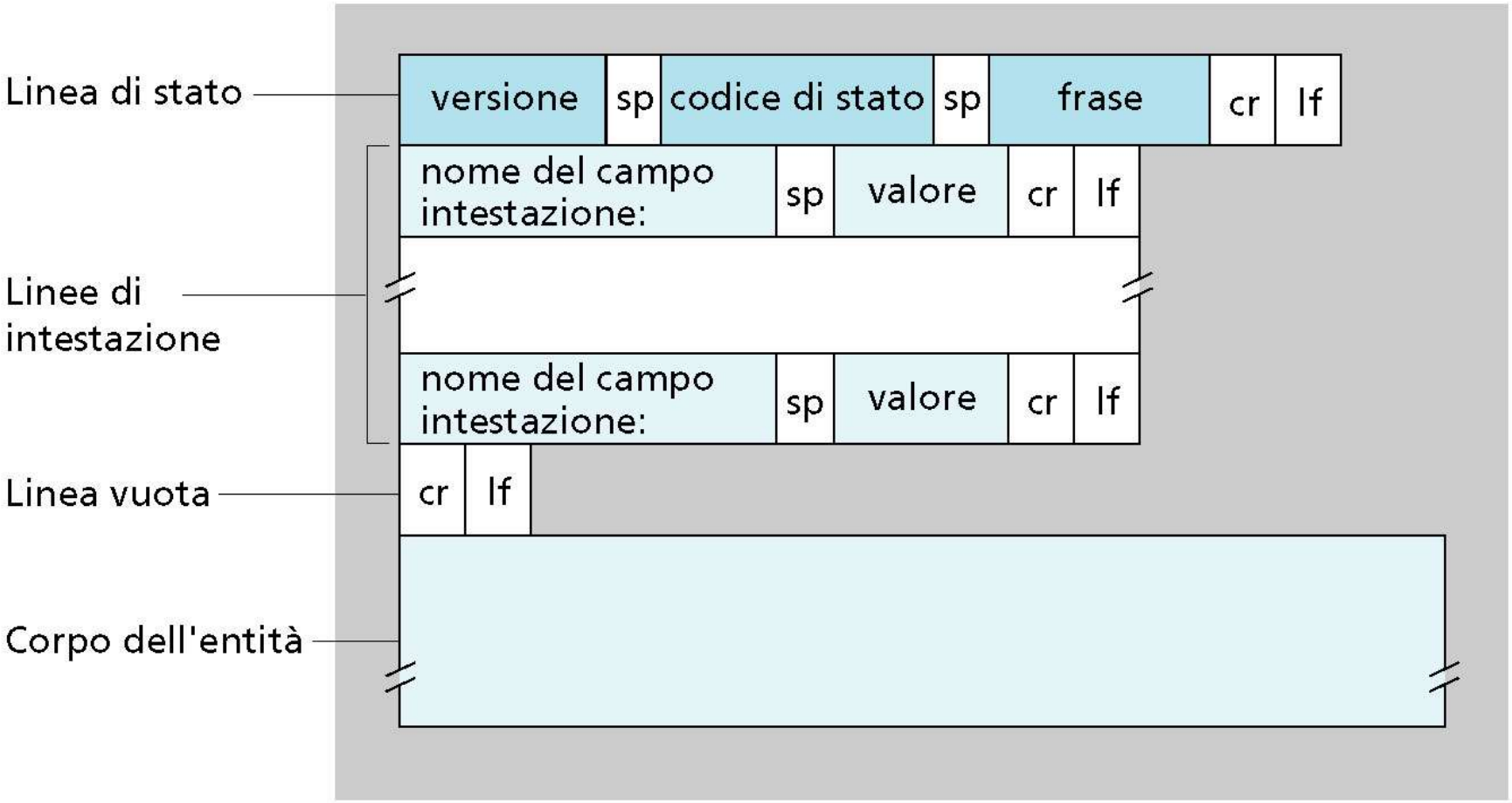
```
generic-message = start-line
                  *message-header
                  CRLF
                  [ message-body ]


start-line      = Request-Line |
                      Status-Line
```

NOTA: La start line distingue la request dalla response

HTTP Request message



HTTP Response message



HTTP request

```
Request  = Request-Line
           *( general-header
            | request-header
            | entity-header )
           CRLF
           [ message-body ]
```

Un esempio

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
Connection: close
User Agent: Mozilla/4.0
Accept-language: it

(Body)
```

HTTP request line

```
Request-Line = Method SP
               Request-URI SP
               HTTP-Version CRLF
```

```
GET http://www.w3.org/pub/WW/TheProject.html HTTP/1.1
```

```
Method  = "OPTIONS" | "GET"
        | "HEAD"    | "POST"
        | "PUT"     | "DELETE"
        | "TRACE"   | extension-method
```

HTTP request line

- Method: The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.
- HTTP-Version - The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication
- Uniform Resource Identifiers are simply formatted strings which identify a network resource.

HTTP request line (GET)

```
GET http://www.w3.org/pub/
        WWW/TheProject.html HTTP/1.1


GET /pub/WWW/TheProject.html HTTP/1.1

(Host: www.w3.org)
```

The exact resource identified by an Internet request is determined by examining both the Request-URI and the Host header field

RFC 2616

Header

- Gli header sono insiemi di coppie (nome: valore) che specificano alcuni parametri del messaggio trasmesso o ricevuto:
- *General Header* – relativi alla trasmissione
  - Es. Data, codifica, connection,
- *Entity Header* - relativi all'entità trasmessa
  - Content-type, Content-Length, data di scadenza, ecc.
- *Request Header* – relativi alla richiesta
  - Chi fa la richiesta, a chi viene fatta la richiesta, che tipo di caratteristiche il client è in grado di accettare, autorizzazione, ecc.
- *Response Header* – nel messaggio di risposta
  - Server, autorizzazione richiesta, ecc.

General Headers

- *Date*: date and time at which the message was originated
- *Connection*: allows the sender to specify options that are desired for that particular connection. “close” connection option for the sender to signal that the connection will be closed after completion of the response.
- *Transfer-encoding*: indicates what (if any) type of transformation has been applied to the message body in order to safely transfer it between the sender and the recipient (e.g. chunked, gzip)

General Headers

- *Cache Control*
  - Public: Indicates that the response is cachable by any cache
  - Private: Indicates that all or part of the response message is intended for a single user and MUST NOT be cached by a shared cache. A private (non-shared) cache may cache the response.
  - no-cache: Indicates that all or part of the response message MUST NOT be cached anywhere.

General headers

```
general-header = Cache-Control
               | Connection
               | Date
               | Pragma
               | Transfer-Encoding
               | Upgrade
               | Via
```

Nota: questi headers si applicano a tutto il messaggio

General headers

```
Esempi:

Date: Tue, 15 Nov 1994 08:12:31 GMT

Connection: close

Transfer-Encoding: chunked
```

Request headers

```
request-header = Accept               | Accept-Charset
               | Accept-Encoding      | Accept-Language
               | Authorization
               | Proxy-Authorization
               | From                 | Host
               | If-Modified-Since
               | If-Unmodified-Since
               | If-Match             | If-None-Match
               | If-Range
               | Max-Forwards         | Range
               | Referer              | User-Agent
```

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.

Request headers

- Accept: to specify certain media types which are acceptable for the response.
  - "q" parameter for indicating a relative quality factor. Default =1
- Accept-Charset: quale set di caratteri accettabile per la risposta
- Accept-Encoding: trasformazione sul contenuto (e.g. compressione)

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
Accept-Charset: iso-8859-5,
                unicode-1-1;q=0.8
Accept-Encoding: compress, gzip
```

(Some) Request methods

- OPTIONS
- GET
- HEAD
- POST
- DELETE
- PUT

RFC 2616: definizione della semantica dei metodi

Request method - OPTIONS

```
OPTIONS http://192.168.11.66/manual/index.html
      HTTP/1.1
host: 192.168.11.66
Connection: close

HTTP/1.1 200 OK
Date: Sun, 14 May 2000 19:52:12 GMT
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
Connection: close
```

Richiede solo le opzioni di comunicazione associate ad un URL o al server stesso (le sue capacità, metodi esposti, ecc.)

Request method – GET (request)

```
GET http://192.168.11.66 HTTP/1.1
host: 192.168.11.66
Connection: close
```

Metodo che richiede il trasferimento di una risorsa identificata da una URL o operazioni associate all'URL stessa.

Sono possibili **conditional get** (header "If-…")

- Header: If-Modified-Since, If-Unmodified-Since (date); If-Match, If-None-Match (ETag); or If-Range

partial get

- Header: Range

Request method – GET (response)

```
HTTP/1.1 200 OK
Date: Sun, 14 May 2000 19:57:13 GMT
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT
ETag: "f2fc-799-37e79a4c"
Accept-Ranges: bytes
Content-Length: 1945
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2
       Final//EN">
<HTML> ...
```

GET condizionale

```
GET http://192.168.11.66 HTTP/1.1
Host: 192.168.11.66
If-Modified-Since: Tue, 21 Sep 1999
14:46:36 GMT
```

```
HTTP/1.1 304 Not Modified
Date: Wed, 22 Sep 1999 15:06:36 GMT
Server: Apache/1.3.9 (Unix)  (Red
Hat/Linux)
```

Altri meccanismi:
Etag (If-None-Match, If-Match)

Request method – HEAD (request)

```
HEAD http://192.168.11.66 HTTP/1.1
host: 192.168.11.66
Connection: close
```

Simile al GET, ma il server non trasferisce il message body nella risposta.
Utile per controllare lo stato dei documenti (validità, modifiche, cache refresh).

Request method – HEAD (response)

```
HTTP/1.1 200 OK
Date: Sun, 14 May 2000 20:02:41 GMT
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT
ETag: "f2fc-799-37e79a4c"
Accept-Ranges: bytes
Content-Length: 1945
Connection: close
Content-Type: text/html
```

Notare la mancanza del message body.

Request method - POST

- Il metodo POST serve per inviare dal client al server informazioni inserite nel body del messaggio.
- In teoria lo standard dice che…

  The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
- In pratica…

  The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI.

  Esempi: Annotazioni ad URL esistenti, FORMs, Posting a message boards, newsgroups, mailing list, etc., Scrittura su database

Request method – PUT & DELETE

- PUT: il client chiede al server di creare/modificare una risorsa
  - Il client specifica nella Request URI l'identificativo della risorsa (facendo un GET poi posso recuperare la risorsa)
- DELETE: il client chiede di cancellare una risorsa identificata dalla Request URI
- Metodi normalmente non abilitati sui server web pubblici

Metodi sicuri e idempotenti

- Safe Methods
  - Metodi che non hanno effetti “collaterali”, es.non modificano la risorsa
  - GET, HEAD, OPTIONS, TRACE
- Idempotent Methods
  - Methods can also have the property of “idempotence” in the side-effects of N > 0 identical requests is the same as for a single request.
  - GET, HEAD, PUT, DELETE, OPTIONS, TRACE

HTTP response

```
Response = Status-Line
           *( general-header
            | response-header
            | entity-header )
           CRLF
           [ message-body ]
```

Un esempio

HTTP/1.1 200 OK

Date: Sun, 14 May 2000 23:49:39 GMT

Server: Apache/1.3.9 (Unix) (Red Hat/Linux)

Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT

Status Line

```
Status-Line
    HTTP-Version SP
    Status-Code SP
    Reason-Phrase CRLF
```

Esempio: `HTTP/1.1 200 OK`

- Status-Line The first line of a Response message
  - The Status-Code: 3-digit integer result code of the attempt to understand and satisfy the request.
  - The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user.

HTTP response

1xx: Informational - Request received, continuing process
2xx: Success - The action was successfully received, understood, and accepted
3xx: Redirection - Further action must be taken in order to complete the request
4xx: Client Error - The request contains bad syntax or cannot be fulfilled
5xx: Server Error - The server failed to fulfill an apparently valid request

NOTA: Le risposte sono su una singola linea

Status codes

"100" - Continue	"101" - Switching Protocols
"200" - OK	"201" - Created
"202" - Accepted	"203" - Non-Authoritative Information
"204" - No Content	"205" - Reset Content
"206" - Partial Content	
"300" - Multiple Choices	"301" - Moved Permanently
"302" - Moved Temporarily	"303" - See Other
"304" - Not Modified	"305" - Use Proxy
"400" - Bad Request	"401" - Unauthorized
"402" - Payment Required	"403" - Forbidden
"404" - Not Found	"405" - Method Not Allowed
"406" - Not Acceptable	"407" - Proxy Authentication Required
"408" - Request Time-out	"409" - Conflict
"410" - Gone	"411" - Length Required
"412" - Precondition Failed	"413" - Request Entity Too Large
"414" - Request-URI Too Large	"415" - Unsupported Media Type
"500" - Internal Server Error	"501" - Not Implemented
"502" - Bad Gateway	"503" - Service Unavailable
"504" - Gateway Time-out	"505" - HTTP Version not supported

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Response headers

```
response-header = Age
                | Location
                | Proxy-Authenticate
                | Public
                | Retry-After
                | Server
                | Vary
                | Warning
                | WWW-Authenticate
```

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Response headers

```
Age: 150
     // età del doc. se tramite Proxy

Location: http://www.w3.org/pub/
                    WWW/People.html

Server: CERN/3.0 libwww/2.17
```

AGE: the Age response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. In secondi

The Location response-header field is used to redirect the recipient to a location other than the Request-URI for completion of the request or identification of a new resource

SERVER: The Server response-header field contains information about the software used by the origin server to handle the request.

Content Negotiation

- Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

- Content negotiation: The mechanism for selecting the appropriate representation when servicing a request
- Ogni entity è constituita da un entity body e da una serie di entity headers che ne definiscono contenuto e proprietà.
  - Nota: Gli entity headers sono “informazioni sulle informazioni”, ovvero metadati

Entity headers

```
entity-header  = Allow
               | Content-Base
               | Content-Encoding
               | Content-Language
               | Content-Length
               | Content-Location
               | Content-MD5
               | Content-Range
               | Content-Type
               | ETag
               | Expires
               | Last-Modified
               | extension-header
```

Entity headers

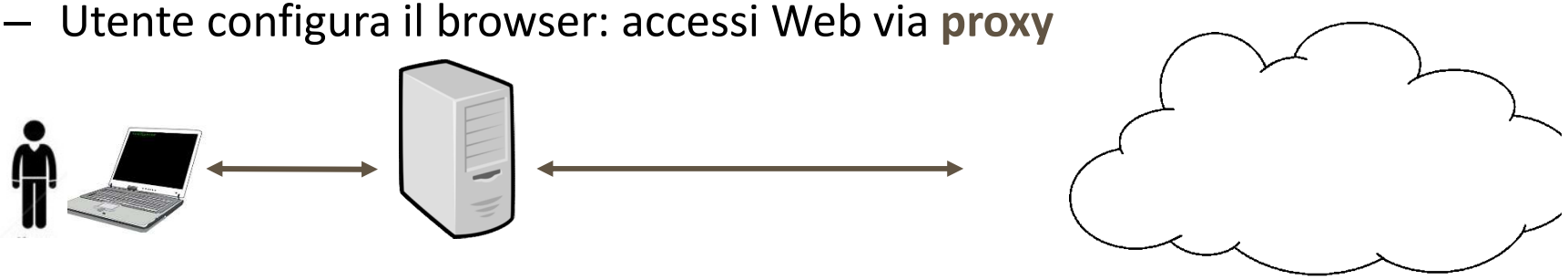
```
Content-Base
  URI assoluta da usare per risolvere le URL
  relative contenute nell'entity body
Content-Encoding
  codifica dell'entity body (es: gzip)
Content-Language
  lingua dell'entity body (es: en, it)
Content-Type
  tipo dell'entity body (es: text/html)
Expires                (utile per caching)
  val. tempor. dell'entity body
Last-Modified          (utile per caching)
  data dell'ultima modifica sul server
```

Esempio Richiesta/Risposta

```
GET http://192.168.11.66/ HTTP/1.1
host: 192.168.11.66
Connection: close

HTTP/1.1 200 OK
Date: Sun, 14 May 2000 23:49:39 GMT
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT
ETag: "f2fc-799-37e79a4c"
Accept-Ranges: bytes
Content-Length: 1945
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD> <TITLE>Test Page for Red Hat Linux's Apache Installation</TITLE> </HEAD>
<H1 ALIGN="CENTER">It Worked!</H1>
<P>
If you can see this, it means that the installation of the  <A
HREF="http://www.apache.org/"  >Apache</A>  software on this <a
href="http://www.redhat.com/">Red Hat Linux</a> system was successful. You
may now add content to  this directory and replace this page.
</P>
</BODY>
</HTML>
```

Web Caching

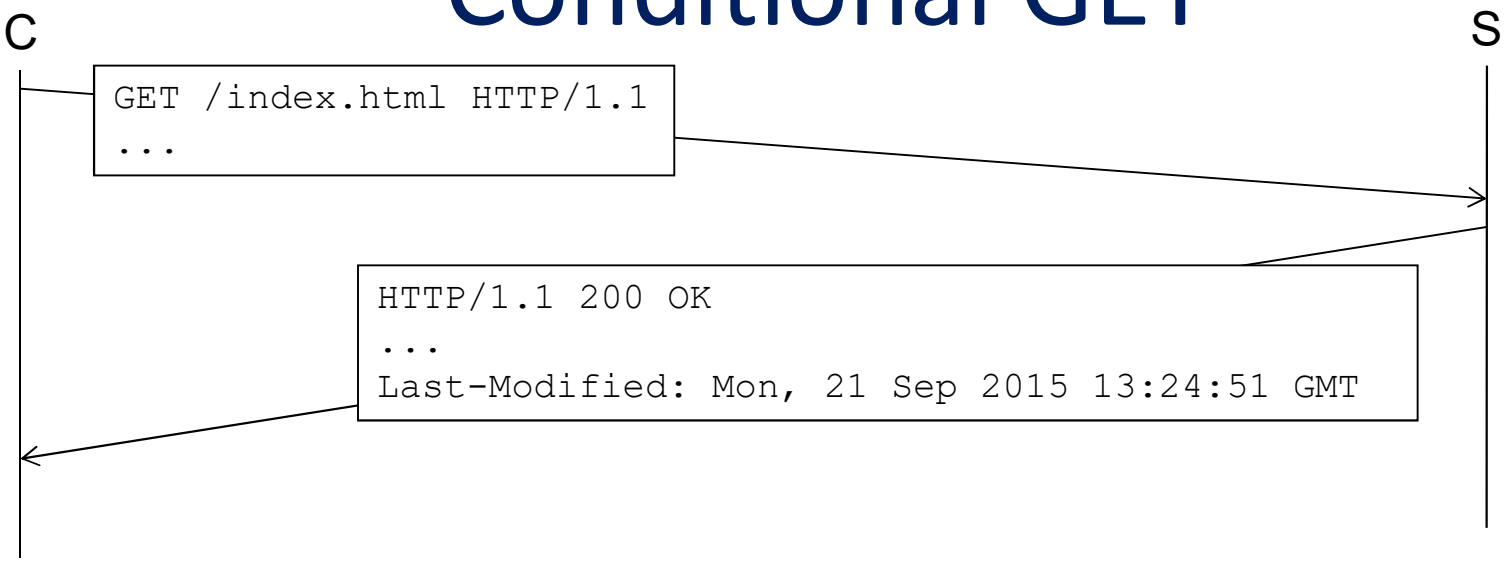
- Obiettivo: soddisfare richiesta del cliente senza contattare server
- Memorizzare copie temporanee di risorse Web (es. pagine HTML, immagini) e servirle al client per ridurre l'uso di risorse (e.g. banda, workload sul server) e diminuite tempo di risposta al client
- User Agent Cache: Lo user agent (il browser) mantiene una copia delle risorse visitate dall'utente
- Proxy Cache
  - Il proxy intercetta il traffico e mette in cache le risposte. Successive richieste alla stessa Request-URI possono essere servite dal proxy senza inoltrare la richiesta al server
  - Utente configura il browser: accessi Web via **proxy**

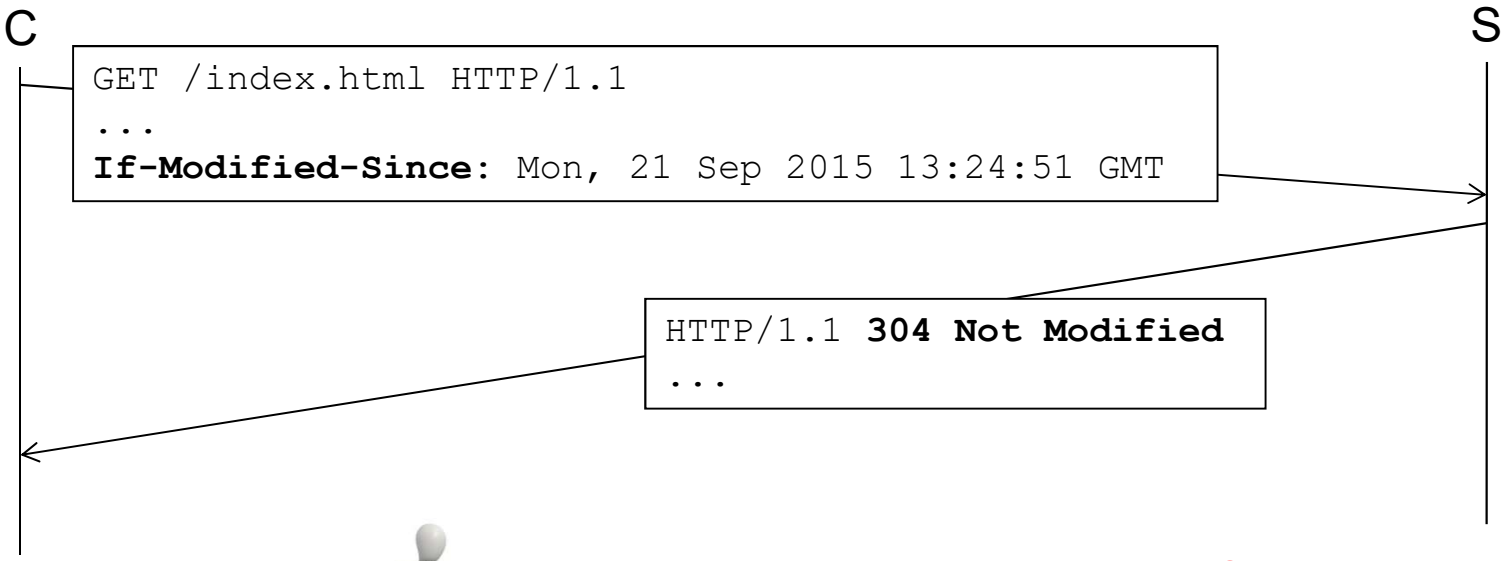


Proxy

- PROXY: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers.
  - A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or parts of the message, and forwarding the reformatted request toward the server identified by the URI.

Conditional GET





Numero messaggi HTTP invariato?

I "cookie" 

- HTTP è state-less (server HTTP non mantengono info sui clienti)
- Problema:
  - Come riconoscere cliente di un'applicazione Web? (p.e. Amazon)
  - Più in generale, come realizzare applicazioni Web con stato? (p.e. shopping cart)
  - Tipicamente **utente si connette ogni volta con un indirizzo (IP e porta) diverso!**
- Soluzione: "numerare" i clienti e obbligarli a "farsi riconoscere" ogni volta presentando un "cookie"

I "cookie"

- Funzionamento:
  - cliente C invia a server S normale richiesta HTTP
  - server invia a cliente normale risposta HTTP + linea **Set-cookie: 1678453**
  - cliente memorizza cookie in un file (associandolo a S) e lo aggiunge con una linea **cookie: 1678453** a tutte le sue successive richieste a quel sito
  - server confronta cookie presentato con l'informazione che ha associato a quel cookie
- Utilizzo dei cookie per
  - autenticazione
  - ricordare profilo utente, scelte precedenti (cfr. “carte-soci”)
  - in generale creare sessioni sopra un protocollo stateless (es. shopping carts, Webmail)
  - ... “non accettare dolci dagli sconosciuti” ... visitate **cookiecentral.com**