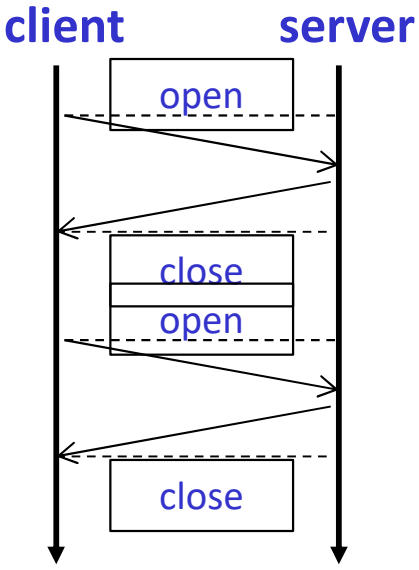
Lo strato Applicativo
HTTP
FORMATO DEL MESSAGGIO

Reti di Calcolatori
AA. 2023-2024

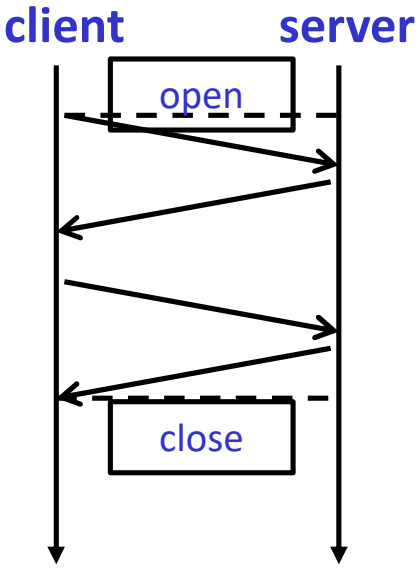
Docente: Federica Paganelli
Dipartimento di Informatica
federica.paganelli@unipi.it

Pipelining

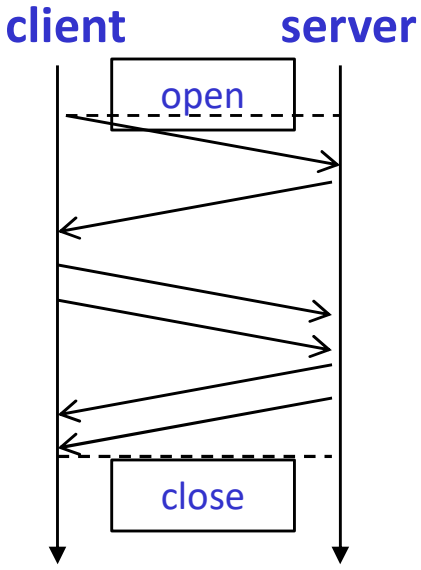
- Serve per migliorare ulteriormente le prestazioni
- Consiste nell'invio da parte del client di molteplici richieste senza aspettare la ricezione di ciascuna risposta
- Il server DEVE inviare le risposte nello stesso ordine in cui sono state ricevute le richieste
- Il client non può inviare in pipeline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



HTTP v1.0



HTTP v1.1



HTTP v1.1 con pipeline

Pipelining e HOLB

- Server web che rispettano HTTP/1.1 devono supportare il pipelining
- Implementato in pochi browser
- Le risposte devono essere inviate in ordine (se una richiesta richiede tempo per essere processata le risposte alle richieste successive sono bloccate (Head of Line Blocking)
- In questi casi l'obiettivo di diminuire il tempo di caricamento di una pagina non è raggiunto

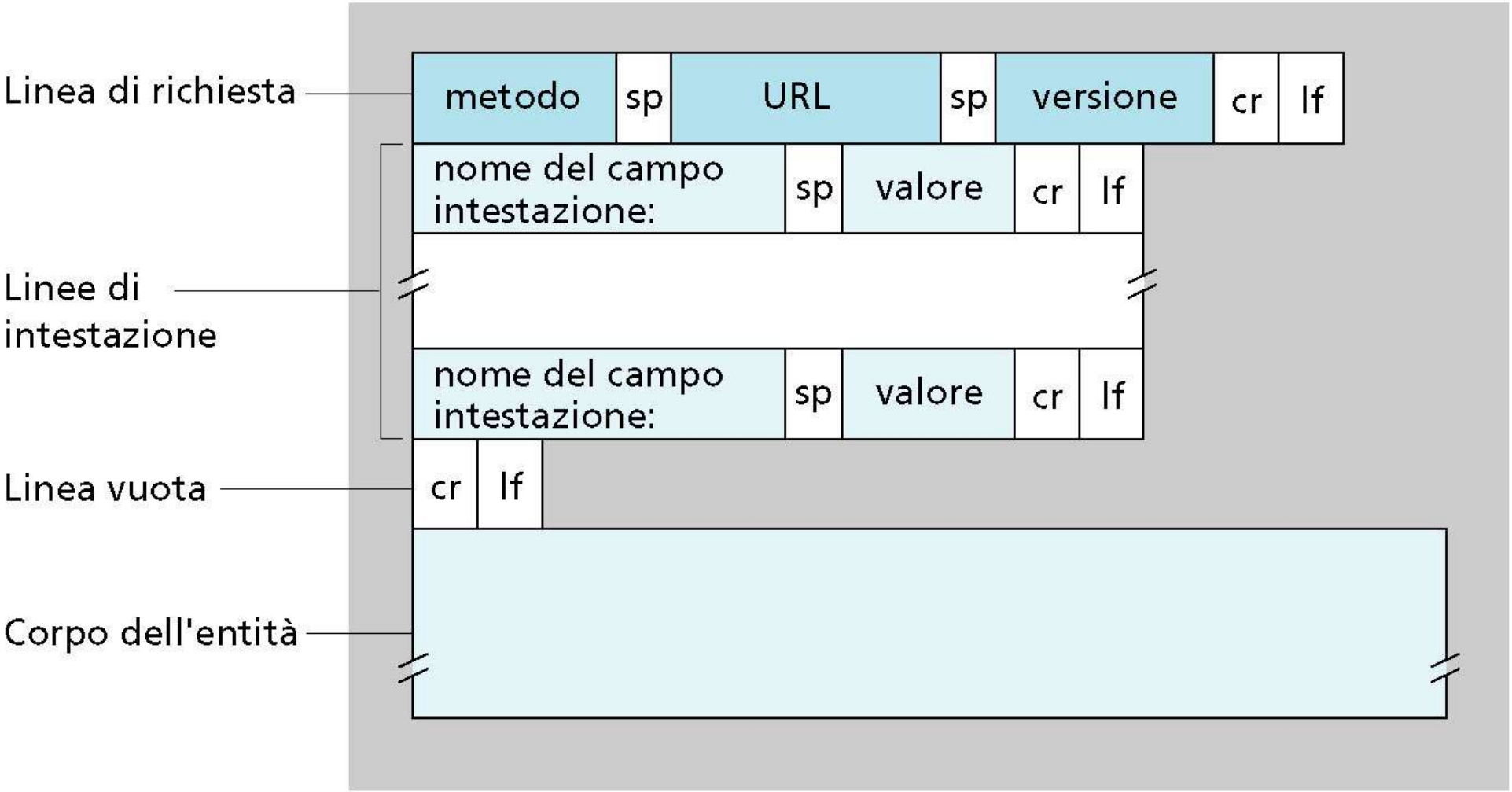
Approfondimento

Come migliorare il tempo di caricamento di una pagina agendo a livello di protocollo? Quali versioni successive di HTTP sono state specificate?

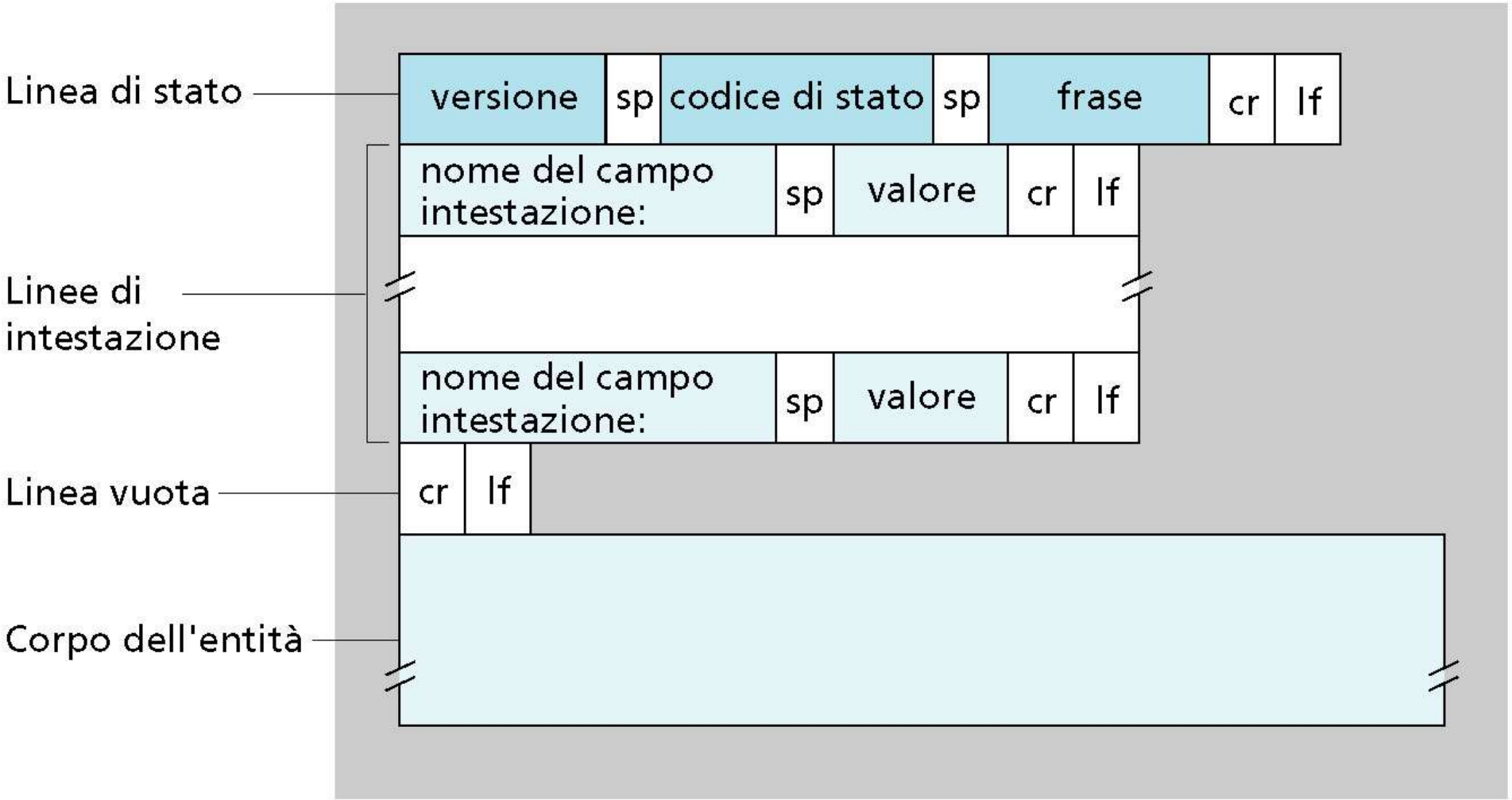
Messaggi HTTP

- Un messaggio HTTP può essere di due tipi: request o response.
- La riga iniziale distingue la richiesta dalla risposta (start –line)
- Seguono una serie di header (coppie nome - valore)
- Un corpo del messaggio (non sempre presente)

HTTP Request message



HTTP Response message



HTTP request

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

Esempio

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

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: operazione che il client richiede venga effettuata sulla risorsa identificata dalla Request-URI. Definizione della semantica dei metodi nella RFC 2616
- HTTP-Version – il mittente indica il formato del messaggio e la sua capacità di comprendere ulteriori comunicazioni HTTP

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

NB. Affrontiamo i metodi dopo aver visto il formato dei messaggi di richiesta e di risposta

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, connessione,
- *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.

Examples

General Headers

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
Connection: close
Transfer-Encoding: chunked
```

Request Headers

```
Accept: text/plain; q=0.5, text/html,
        image/png, application/json
Accept-Charset: iso-8859-5,
                unicode-1-1;q=0.8
Accept-Encoding: compress, gzip
```

- Accept: specifica il formato del corpo del messaggio accettabile per una risposta. “q” indica una sorta di preferenza. Default =1
- Accept-Charset: set di caratteri accettabile per la risposta
- Accept-Encoding: tipo di codifica del contenuto accettabile per la risposta

HTTP response

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

```
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
```

Date: data e ora del messaggio

SERVER: informazioni sul software usato dal server per gestire la richiesta

Status Line

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

Esempio: `HTTP/1.1 200 OK`

- Status-Line: prima riga di un messaggio di risposta
  - Status-Code: intero a tre cifre che rappresenta un codice di risultato dell'operazione richiesta
  - Reason-Phrase: descrizione testuale dello Status code (pensata per l'utente umano)

Status code

- Dallo standard (https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html):

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

Status codes - esempi

"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

- Informazioni relative alla risposta che non possono essere inserite nella Status Line
- Esempi:

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

Server: CERN/3.0 libwww/2.17
```

Location: usato per ridirezionare il ricevente a una URI differente dalla Request-URI per completare la richiesta o per identificare una nuova risorsa.

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)validità temporale del contenuto

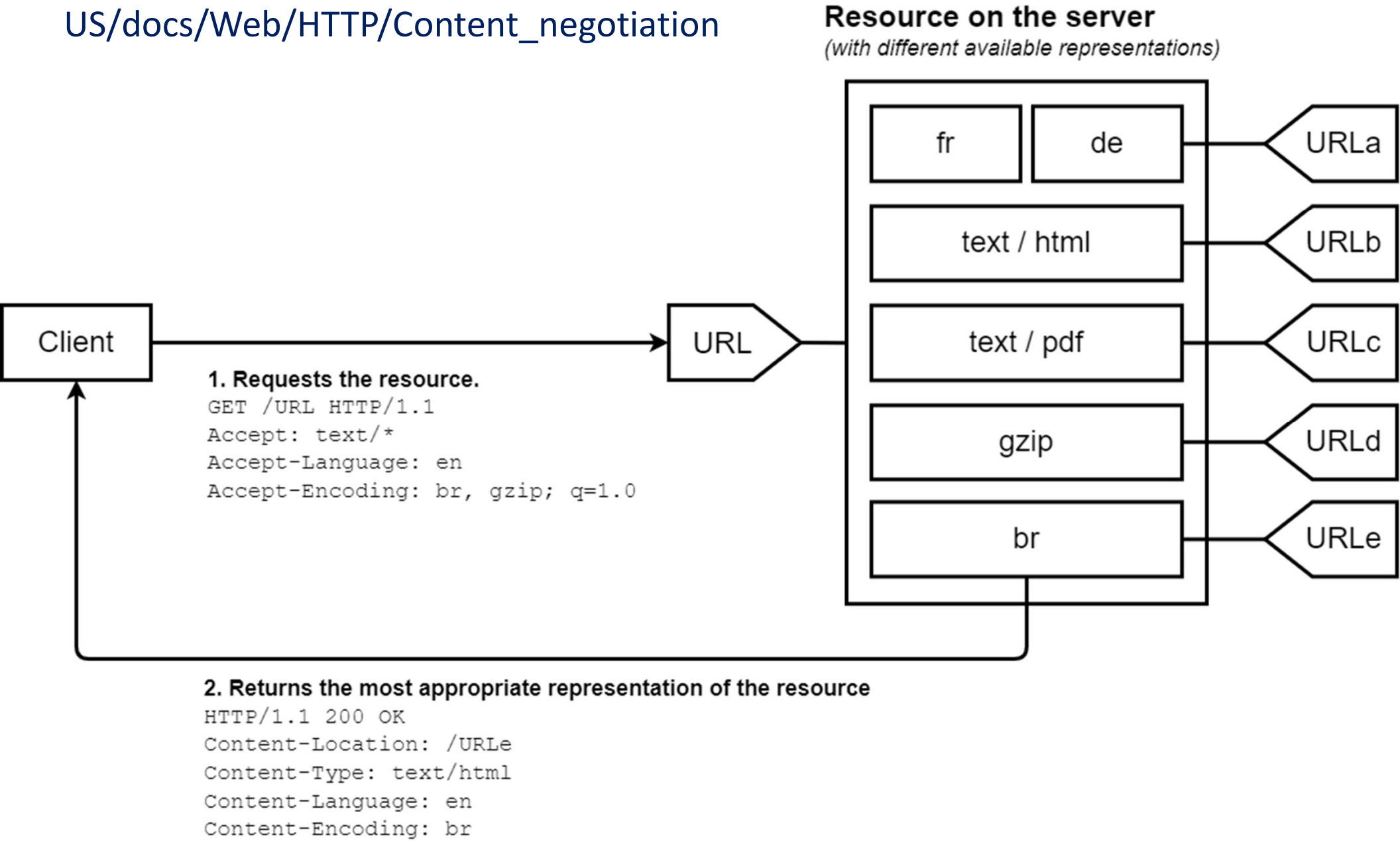
Last-Modified
(utile per caching)data dell'ultima modifica sul server

Content Negotiation

- Le risorse possono essere disponibili in più rappresentazioni (lingua, formato di dati, dimensione, ecc..)
- **Content negotiation**
  - meccanismo per selezionare la rappresentazione appropriata quando viene servita una richiesta (uso di Request e Entity headers)

Content Negotiation

https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation



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>
```

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> ...
```

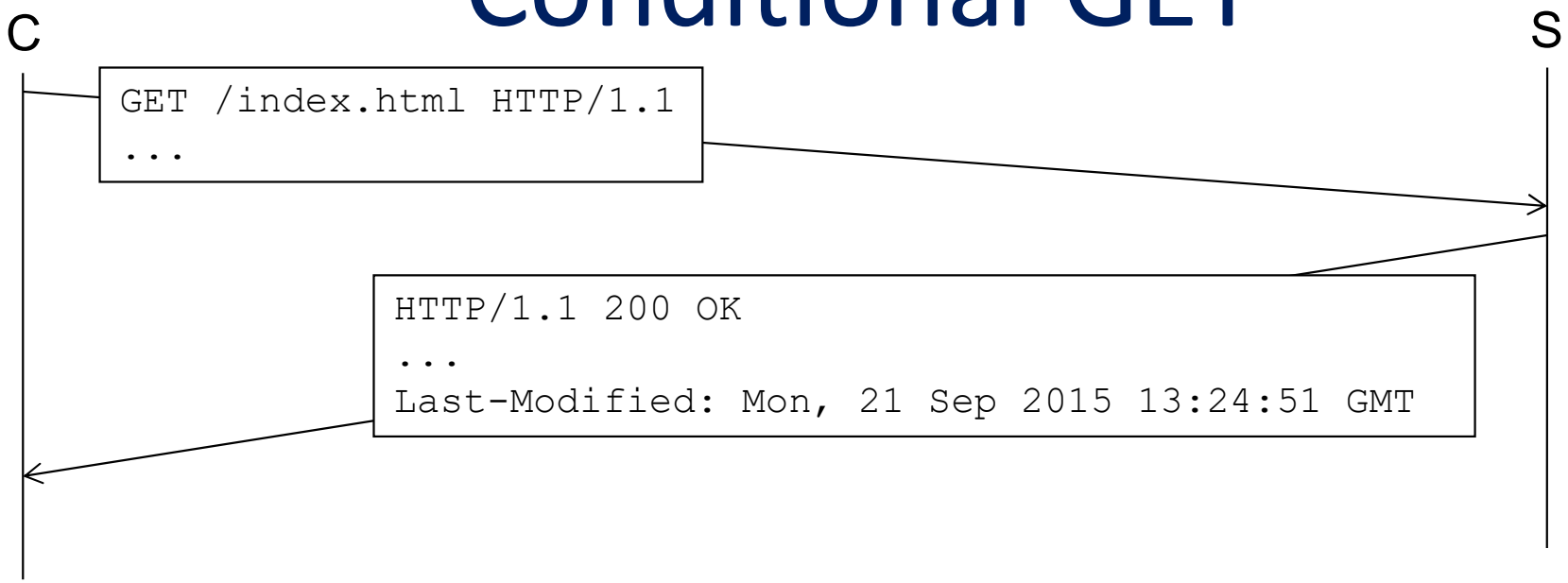
GET condizionale

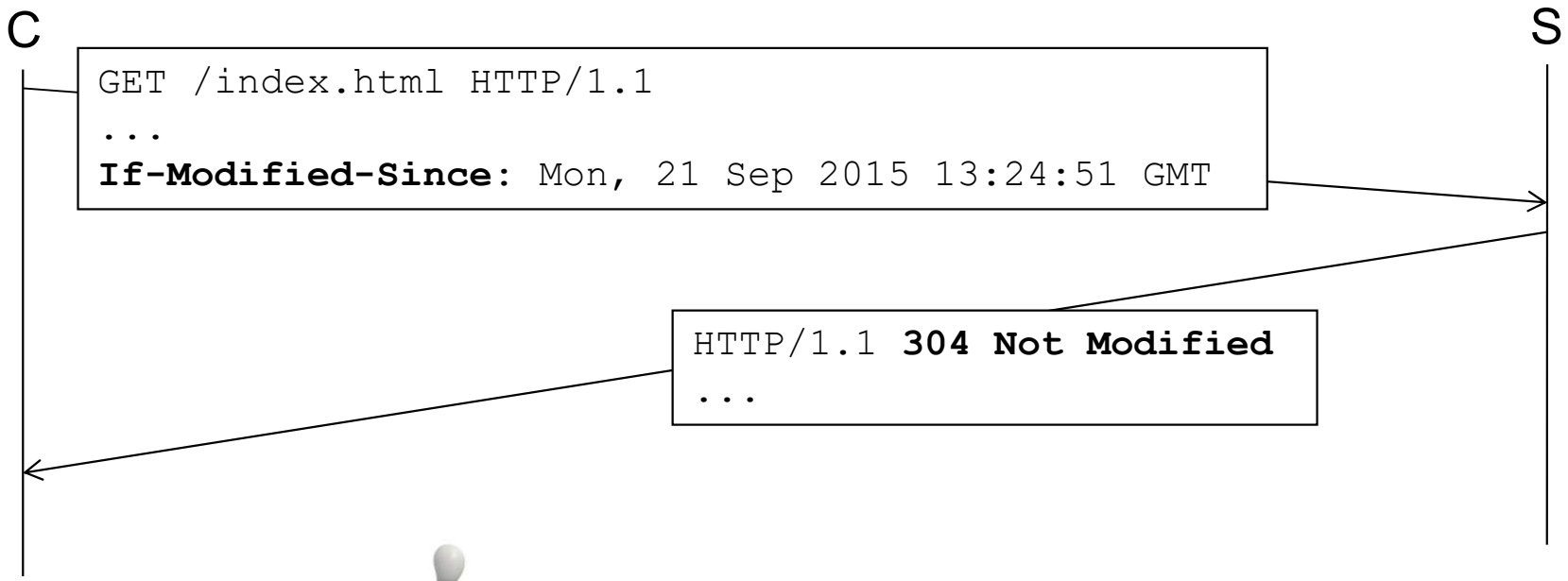
```
GET http://192.168.11.66 HTTP/1.1
Host: 192.168.11.66
If-Modified-Since: Tue, 21 Sep 1999 14:46:36 GMT
```

```
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)

Conditional GET





Numero messaggi HTTP invariato?

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…

  Il metodo POST è usato per chiedere che il server accetti l'entità (risorsa) nel corpo della richiesta come una nuova subordinata della risorsa identificata dalla Request-URI nella Request-Line.

- In pratica…

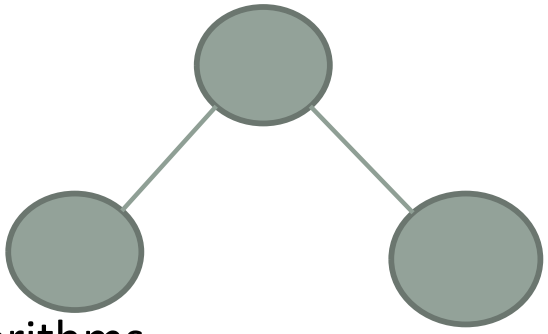
  La funzione effettiva eseguita dal POST è determinata dal server e dipendente tipicamente dalla Request-URI.

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

  NB con le API REST vengono rispettate le specifiche HTTP…

Request method - POST

www.unipi.it/courses



www.unipi.it/courses/Algorithms

www.unipi.it/courses/ComputerNetworks

POST www.unipi.it/courses HTTP/1.1

…

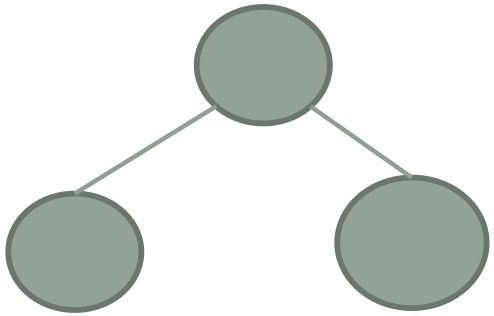
…entity body representing the resource Computer Networks…

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 una 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

Request method - PUT

www.unipi.it/courses



www.unipi.it/courses/Algorithms

www.unipi.it/courses/ComputerNetworks

PUT www.unipi.it/courses/ComputerNetworks HTTP/1.1

…

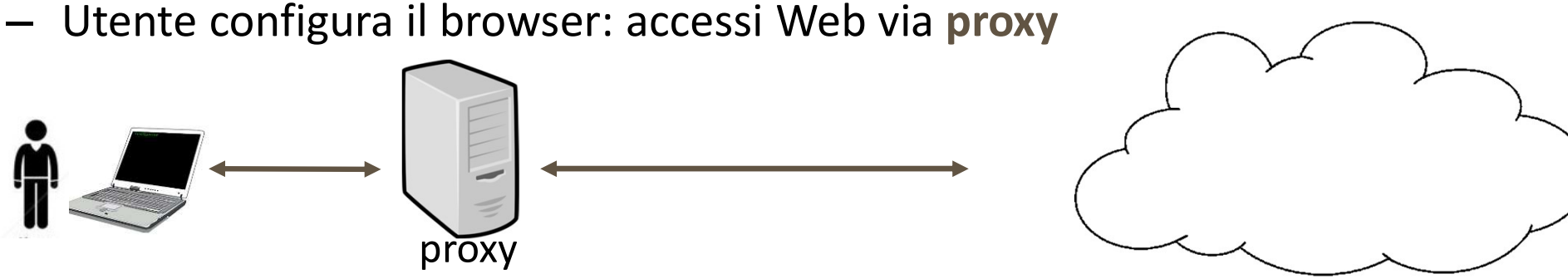
…entity body representing the resource Computer Networks…

Metodi sicuri e idempotenti

- Safe Methods
  - Metodi che non hanno effetti “collaterali”, es. non modificano la risorsa
  - GET, HEAD, OPTIONS, TRACE
- Idempotent Methods
  - I metodi possono avere la proprietà di “idempotenza” negli effetti collaterali se N > 0 richieste identiche hanno lo stesso effetto di una richiesta singola.
  - GET, HEAD, PUT, DELETE, OPTIONS, TRACE

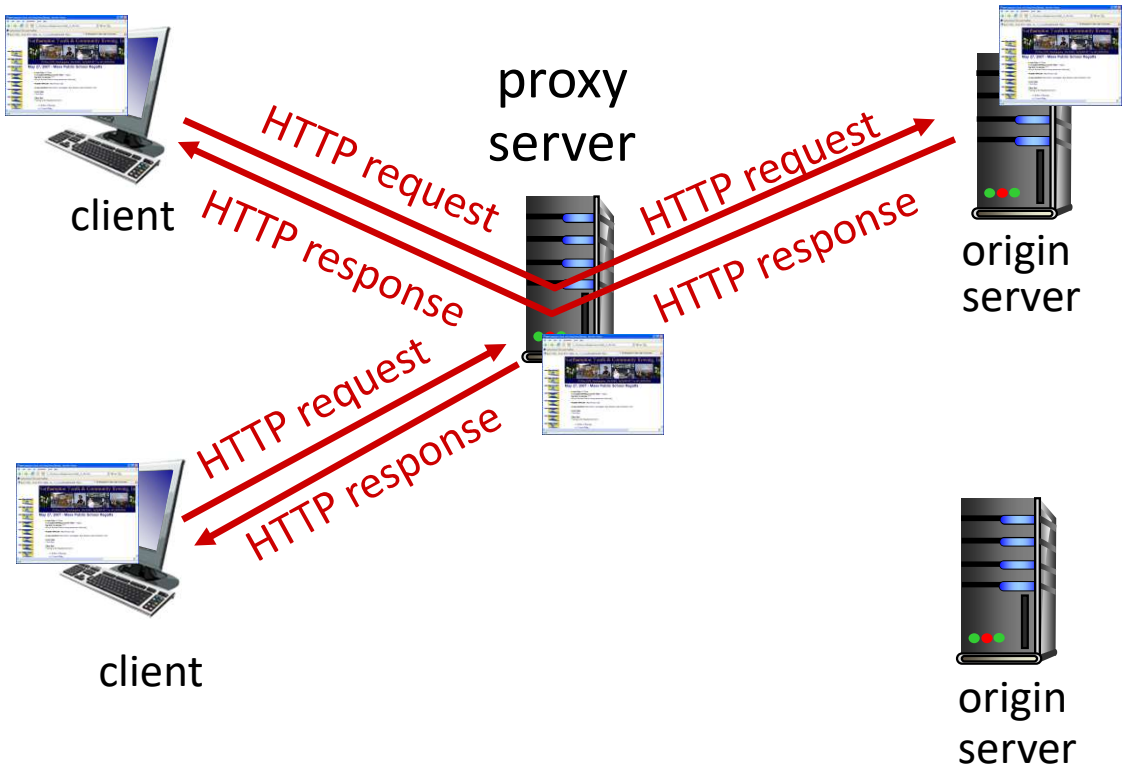
Web Caching

- 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 diminuire 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**



Web cache (server proxy)

- l'utente configura il browser in modo che punti a una cache Web il browser invia tutte le richieste HTTP alla cache
- se l'oggetto è nella cache: la cache restituisce l'oggetto al client
- Altrimenti la cache richiede l'oggetto al server di origine, memorizza nella cache l'oggetto ricevuto, quindi restituisce l'oggetto al client



Web cache (server proxy)

- La cache Web funge sia da client che da server:
  - Server per il client richiedente
  - Client verso il server di origine
- Web cache può essere fornita installata dall'ISP (università, azienda, ISP residenziale)

Vantaggi

Web cache (server proxy)

- La cache Web funge sia da client che da server:
  - Server per il client richiedente
  - Client verso il server di origine
- Web cache può essere fornita installata dall'ISP (università, azienda, ISP residenziale)

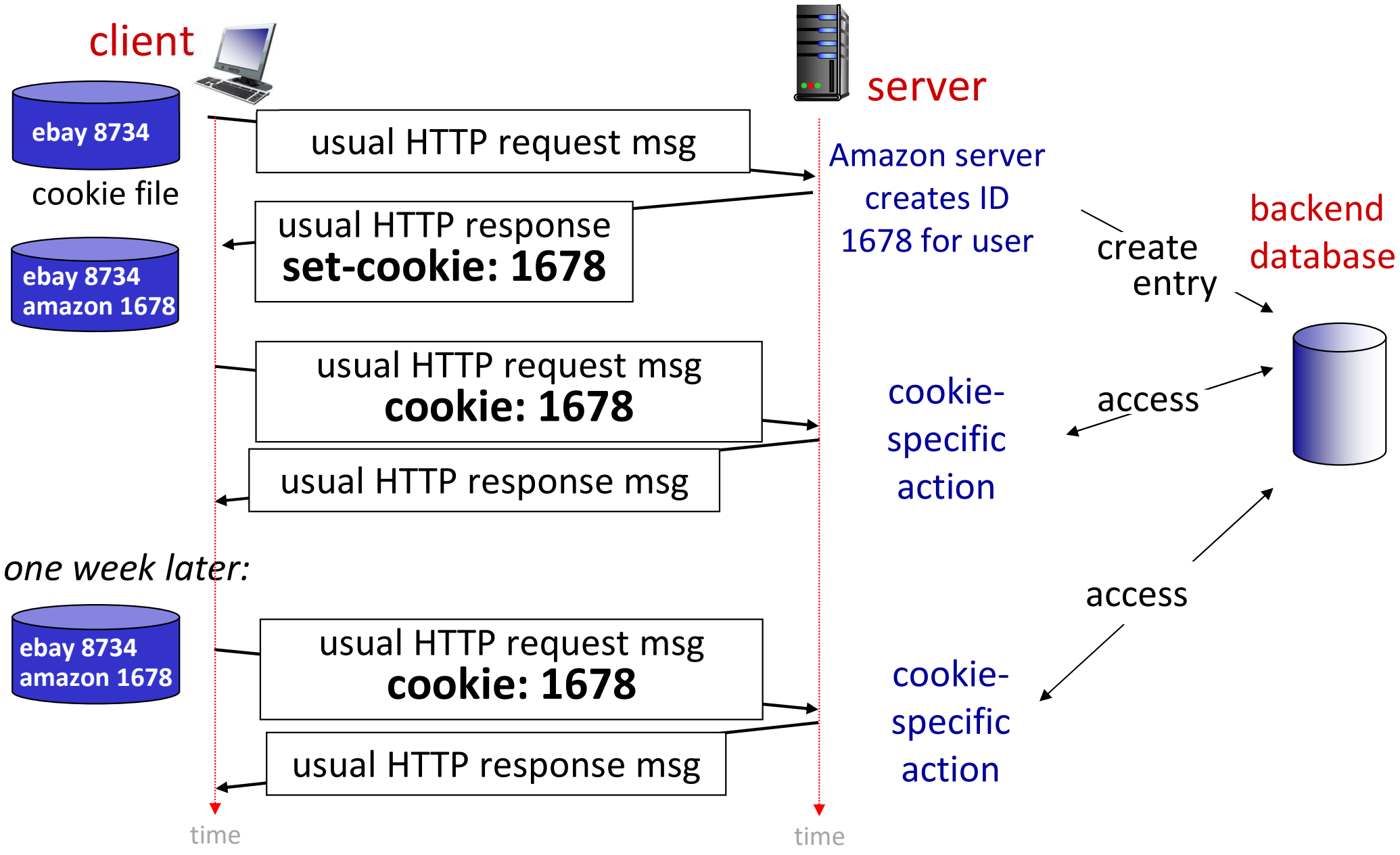
Vantaggi

- ridurre i tempi di risposta
  - la cache è più vicina al client
- ridurre il traffico sul collegamento di accesso di un istituto
- Internet è denso di cache
  - consente ai «piccoli» fornitori di contenuti di fornire contenuti in modo più efficace

I "cookie"

- HTTP è state-less (server HTTP non mantengono informazioni sui clienti)
- Problema:
  - Come riconoscere un 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"


client
server
ebay 8734
cookie file
ebay 8734
amazon 1678
usual HTTP request msg
usual HTTP response
set-cookie: 1678
Amazon server
creates ID
1678 for user
create
entry
backend
database
usual HTTP request msg
cookie: 1678
usual HTTP response msg
cookie-
specific
action
access
one week later:
ebay 8734
amazon 1678
usual HTTP request msg
cookie: 1678
usual HTTP response msg
access
cookie-
specific
action
time
time

I "cookie"

- Funzionamento:
  - cliente C invia a server S normale richiesta HTTP
  - server invia a cliente normale risposta HTTP + linea **Set-cookie: 1678**
  - cliente memorizza cookie in un file (associandolo a S) e lo aggiunge con una linea **cookie: 1678** 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**