

Web Applications

Il protocollo HTTP

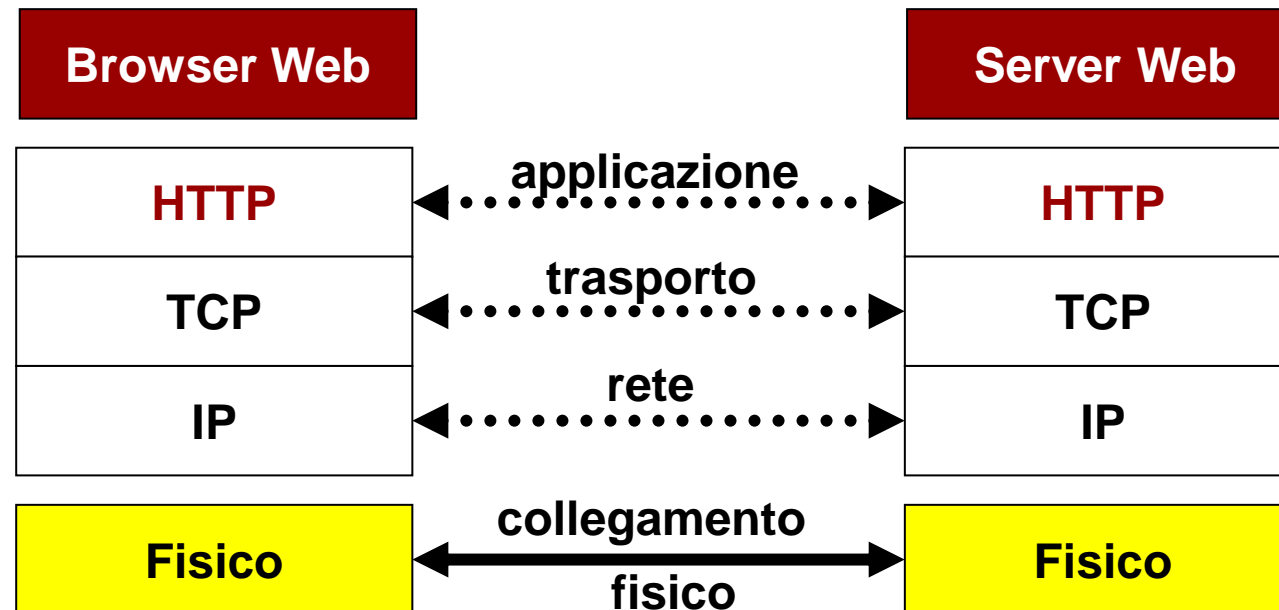
Slide a cura di Giovanni Grasso/Kristian Reale



This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details

HTTP

- HyperText Transfer Protocol
- Protocollo di applicazione usato per trasferire risorse tra client e server
- Gestisce richieste (URL) dal client al server



Terminologia

- Un client e' un qualsiasi applicativo in grado di stabilire una connessione ed inviare richieste
- Il server e' un programma applicativo che accetta connessioni e restituisce specifiche risposte in base alle richieste ricevute
- Una connessione e' un "circuito virtuale" che consente a due applicazioni di comunicare

Terminologia

- Messaggio: e' l'unita' di base di comunicazione HTTP
 - Request, Response: messaggio HTTP di richiesta/risposta
 - Risorsa: qualsiasi cosa univocamente definita (e.g., documento, immagine, servizio web)
 - URI: Uniform Resource Identifier, identifica una risorsa
- Entity
 - Rappresentazione di una risorsa (e.g., pagina HTML,).

Uniform Resource Identifiers (URI)

- URN

- la stringa non descrive il metodo d'accesso ma bensì un nome globalmente unico
- la risorsa può non essere fisicamente accessibile (es: namespace)
- E' come identificare una persona per codice fiscale
- ISBN
 - urn:isbn:0-12345-678-9 (non dice nulla sulla locazione del libro)

- URL

- la risorsa è fisicamente accessibile, la stringa descrive il metodo (primario) per accedere alla risorsa
- E' come identificare una persona per indirizzo o numero telefonico (se univoci)

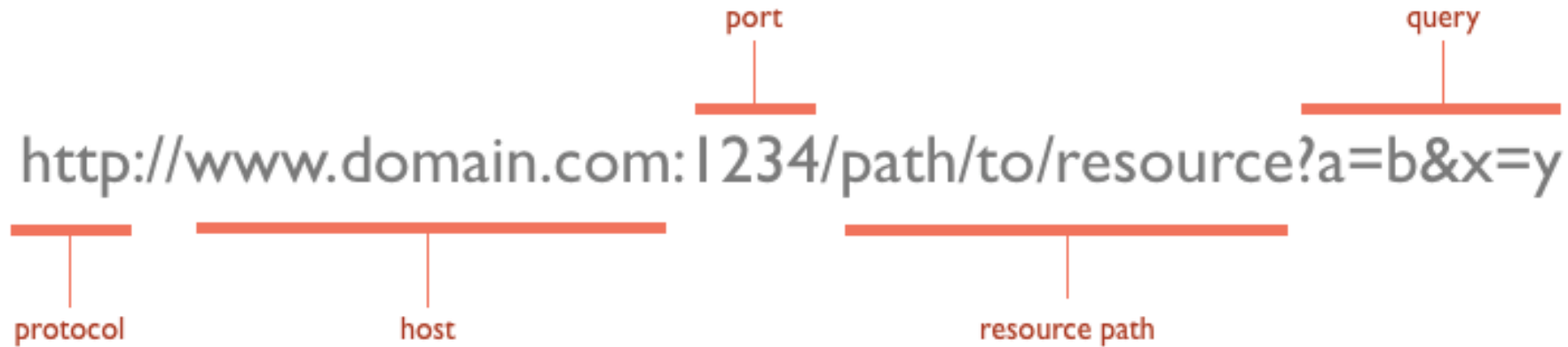
- Forma generale

- <protocollo>:<parte-dipendente-dal-protocollo>

URI

- Principali protocolli
 - http, ftp, mailto, file, news
- Esempi:
 - http://www.unical.it/informatica
 - ftp://lab.unical.it/pub
 - mailto:grasso@mat.unical.it
 - <file:///d:/sites/users/grasso/index.html>
 - news:comp.infosystems.www.servers.unix
 - telnet://melvyl.ucop.edu/

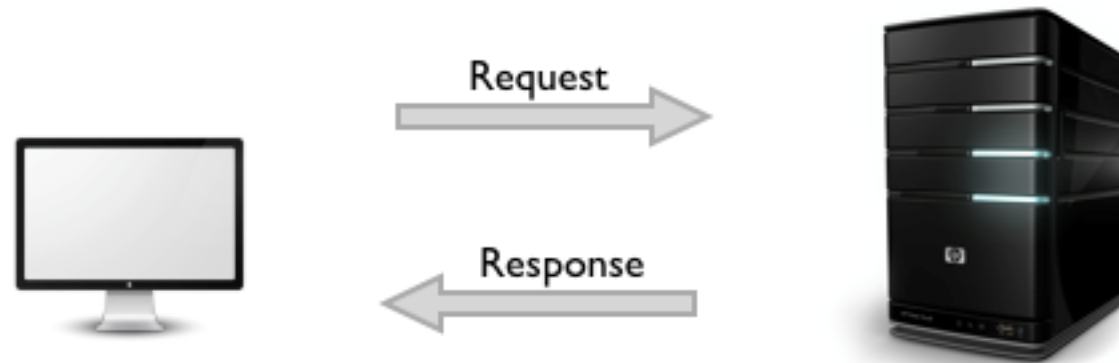
URL



- Protocol
- Host: indirizzo del server, logico o fisico
- Port: default 80
- Resource path: identifica la risorsa nel file system del server
- Query: passaggio dei parametri, nome=valore

HTTP

- E' un protocollo stateless: ne' il client ne' il server "ricordano" nulla dei messaggi precedenti
 - La mancanza di stato influenza la scrittura delle app
- Una transazione HTTP e' uno scambio di messaggi richiesta/risposta tra client e server

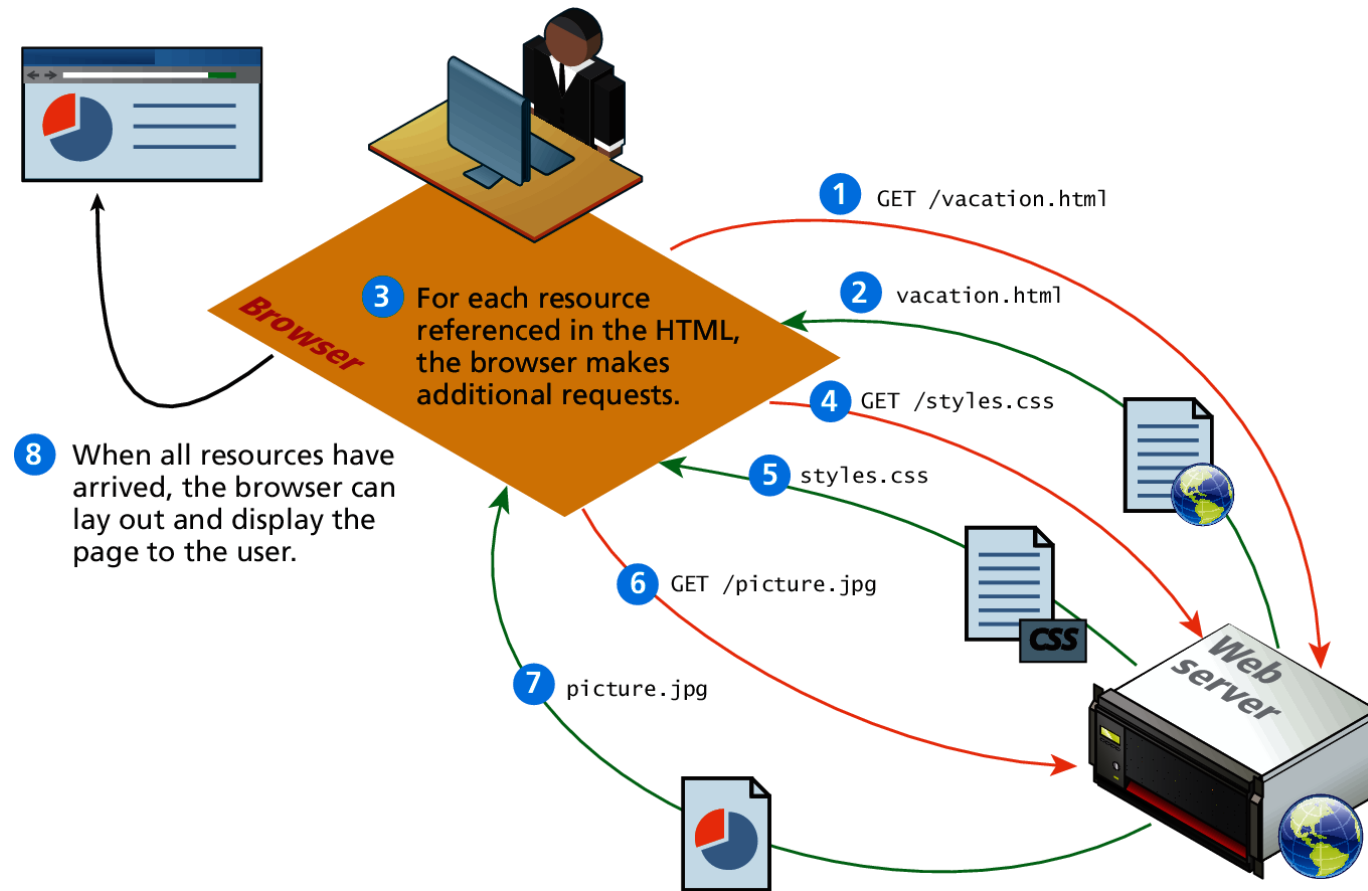


Transazione HTTP

- Il server e' un ascolto su una porta (80 default)
- Il client si connette al server
 - Per esempio sul browser l'utente chiede `www.example.com`
 - Tramite DNS il browser risolve il nome in indirizzo IP
 - Viene richiesta una connessione all'Ip sulla porta 80
 - Il server accetta e' conferma (ACK) la connessione al client
- Il client manda un request message
 - `GET /index.html HTTP/1.0`
- Il server riceve la richiesta e restituisce un response message che contiene la risorsa richiesta.
- Il client visualizza la risorsa, il server chiude la connessione

Richieste HTTP

- Le richieste HTTP sono solitamente multiple
 - Una volta ottenuta la pagina index.html il browser richiede tutte le risorse aggiuntive (immagini, files)

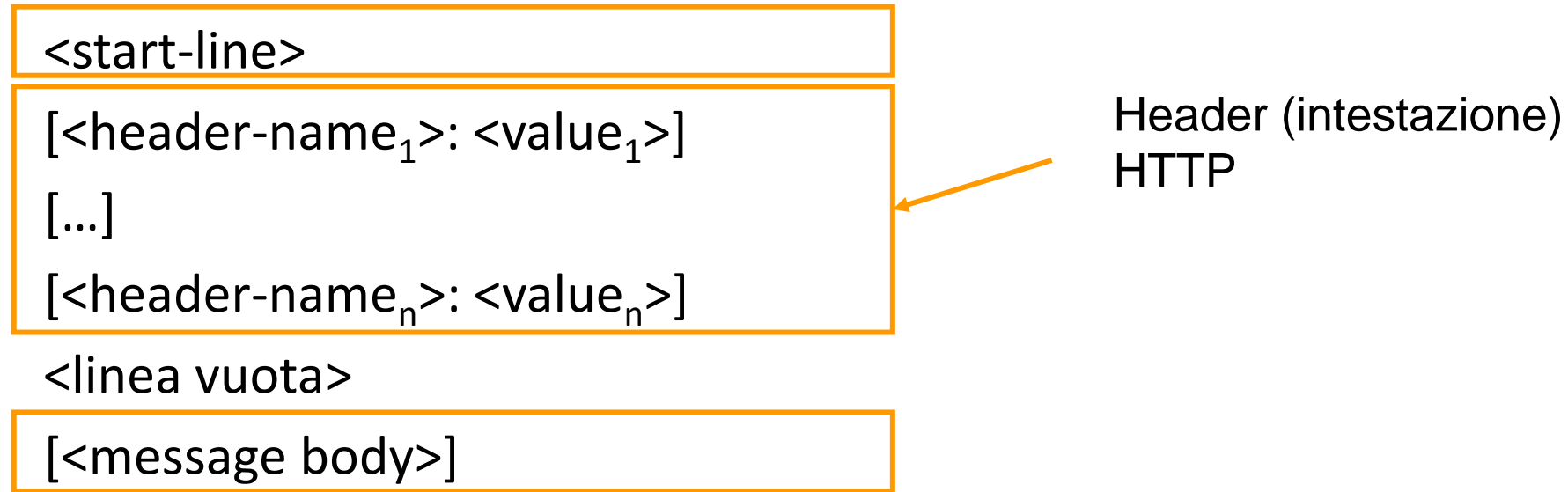


Connessioni in HTTP 1.0 e 1.1

- HTTP 1.0: Connessioni non persistenti
 - una coppia richiesta/risposta per connessione
 - Se supportato dal server, il client può effettuare connessioni parallele
- HTTP 1.1: Connessioni Persistenti (default)
 - Il server lascia aperta la connessione dopo la prima risposta per poter ricevere richieste successive
 - La connessione è chiusa dal client con un messaggio specifico oppure, se inutilizzata, dal server dopo un time-out
 - Minimizza l'overhead di TCP
- Connessioni parallele e persistenti sono largamente diffuse

HTTP Formato dei Messaggi

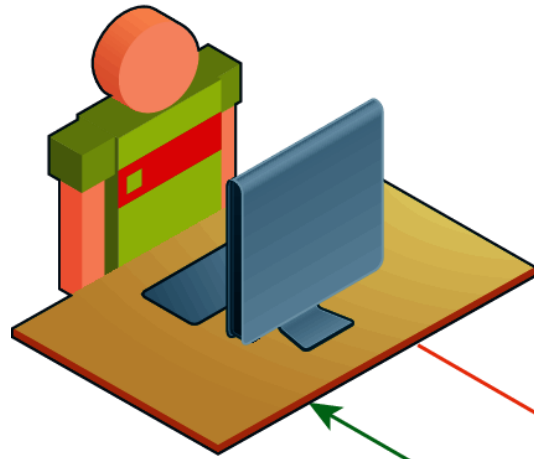
- Richiesta e risposta hanno la stessa struttura



HTTP Formato dei Messaggi

- Start-line
 - nella richiesta contiene l'URI ed il “verbo” HTTP (metodo)
 - nella risposta contiene l'esito della richiesta (codice)
- Message-body
 - nella richiesta è vuoto o contiene la query
 - nella risposta contiene la risorsa richiesta, codificata in un formato MIME specificato nell'header
- Message-headers
 - Metadati per identificare il messaggio





Request



GET /index.html HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1

Accept: text/html,application/xhtml+xml

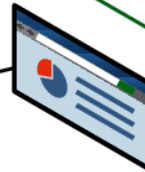
Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

Cache-Control: max-age=0

Response



HTTP/1.1 200 OK

Date: Mon, 22 Oct 2012 02:43:49 GMT

Server: Apache

Vary: Accept-Encoding

Content-Encoding: gzip

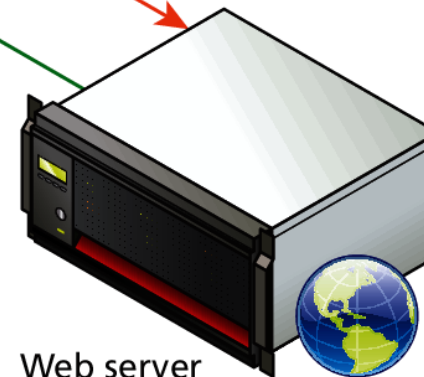
Content-Length: 4538

Connection: close

Content-Type: text/html; charset=UTF-8

<html>

<head> ...



Web server

Esempio di Richiesta

```
GET /informatica HTTP/1.1
Host: www.mat.unical.it
Connection: keep-alive
Accept: text/html,application/xhtml+xml
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X)
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8,it;q=0.6
Cache-Control: max-age=0
```

Start-line: Metodo Oggetto Versione HTTP

headers

Message Body VUOTO

- Metodi
 - GET, POST, PUT, DELETE
 - HEAD, TRACE, OPTION

Metodo GET

- Metodo standard per richiedere una risorsa al server
 - E' quello che accade specificando una URL nel browser o quando si fa click su un link
 - Viene specificato l'URI della risorsa
 - Il corpo della richiesta è vuoto
 - La lunghezza massima dell'URI è limitata
 - Eventuale passaggio di parametri tramite la query
 - visibili pubblicamente

GET /index.html HTTP/1.1

GET /informatica/InformazioniCorsi HTTP/1.1

GET /search.php?nome=Pinco&cogn=Pallo HTTP/1.1

Metodo POST

- Utilizzato per colloquiare con un servizio web
 - Per esempio per sottomettere una form o per fare update di una risorsa (e.g., carrello spesa)
 - Viene specificato l'URI della risorsa senza parametri
 - Utile per dati privati o di una certa lunghezza
 - I parametri contenuti nel corpo del messaggio

POST /search.php? HTTP/1.1

nome=Pinco&cogn=Pallo

Altri Metodi

- HEAD
 - Simile al GET ma la risposta non contiene body ma solo headers. Usato per debugging/validita' url
- PUT e DELETE
 - Creazione e cancellazione di una risorsa.
 - Solitamente disabilitati su server pubblici
 - Usati in REST API (vedremo in seguito)
- TRACE
 - Per diagnostica (numero di hops round-trip)
- OPTIONS
 - Per chiedere le opzioni disponibili per la comunicazione

Esempio di Risposta

Status-line: protocollo codice stato stato leggibile

HTTP/1.1 200 OK

Date: Thu, 25 Feb 2016 12:00:15 GMT

Server: Apache/2.2.16 (Debian)

Content-Length: 6821

Content-Encoding: gzip

Content-Type: text/html; charset=utf-8

Keep-Alive: timeout=15, max=100

Connection: Keep-Alive

headers

<!DOCTYPE html>

<html>...</html>

La risorsa
richiesta

Codice di Stato

- 3 cifre, la prima indica la classe, le restanti la risposta
- 1xx: Messaggio informativo temporaneo (deprecato in 1.1)
- 2xx: Successo. La richiesta e' stata soddisfatta
- 3xx: Redirezione. La richiesta e' ok ma occorrono nuove azioni da parte del client per soddisfare la richiesta
- 4xx: Errore sul client. La richiesta contiene errori e non e' autorizzata
- 5xx: Errore sul server. Problema interno per cui non si puo' soddisfare la richiesta.

Esempi di codici di risposta

- *100 Continue*
 - Il server chiede al client di mandare il resto della richiesta (per esempio il body) o di ignorare se già mandata
- *200 Ok* (la più popolare risposta ad una GET)
- *201 Created* (una PUT ha creato una risorsa con successo)
- *301 Moved permanently*
 - URL non valida ma il server conosce la nuova posizione
- *404 Not Found* (URL invalida, la risorsa non esiste sul server)
- *400 Bad request* (Errore sintattico nella richiesta)
- *401 Unauthorized, 403 Forbidden*
- *500 Internal server error*
- *501 Not implemented* (metodo non conosciuto)

Headers

- Generali

- Si applicano sia alla richiesta che alla risposta.

- Cache-control Date Connection Transfer-Encoding

- Specifici per la richiesta

- User-agent Host Accept-* Authorization If-*

- Specifici per la risposta

- Age Location Server WWW-Authenticate Accept-Ranges

- Specifici per la entity

- Allow Expires Last-Modified Content-*

- Custom

Headers di Richiesta

Host: `www.mat.unical.it`

← Lo stesso IP puo' servire piu' siti

User-Agent: `Mozilla/5.0 (Macintosh; Intel Mac OS X)`

Connection: `keep-alive`

← Il client supporta la connessione persistente

Accept: `text/html,application/xhtml+xml`

← Formato MIME

Accept-Encoding: `gzip, deflate`

Accept-Language: `en-US,en;q=0.8,it;q=0.6`

“Preferisco inglese americano, altrimenti qualsiasi inglese o anche italiano”

MIME Type

- Multipurpose Internet Mail Extensions
 - Internet Media Type
- Usato per descrivere il formato di una risorsa e come deve essere gestito
- E' composta da tipo/sottotipo e parametri opzionali
 - text/html; charset=UTF-8 → HTML
 - Top Level types
 - application, audio, example, image, message, model, multipart, text, video
 - Subtype: di solito il nome di un tipo

Tipi di Contenuto (“Content-Types”)

- text
 - text/plain
 - text/html
 - text/xml ...
- image
 - image/gif
 - image/jpeg
 - image/png ...
- application
 - application/pdf
 - application/zip
 - application/msword
 - application/X-...
- video
 - video/mpeg ...
- audio
- multipart
 - multipart/form-data ...
- message
- model

Headers di Risposta

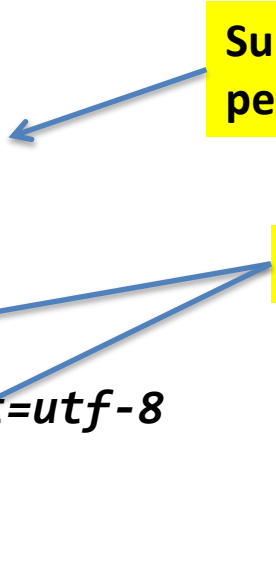
Date: Thu, 25 Feb 2016 12:00:15 GMT

Server: Apache/2.2.16 (Debian)

Keep-Alive: timeout=15, max=100

Connection: Keep-Alive

Supporta connessione
persistente



Content-Length: 6821

Content-Encoding: gzip

Content-Type: text/html; charset=utf-8

Descrivono la Risorsa

<!DOCTYPE html>

<html>...</html>

Applicazioni Stateful: Cookie

- Problema: Mantenere lo stato tra le diverse richieste/risposte nella “sessione” di lavoro
- Idea fondamentale
 - Utilizzare le intestazioni HTTP per "nascondere" informazioni che vengono scambiate tra client e server
- Usa due intestazioni HTTP
 - Set-Cookie: nelle risposte HTTP del server
 - Cookie: nelle richieste HTTP del client

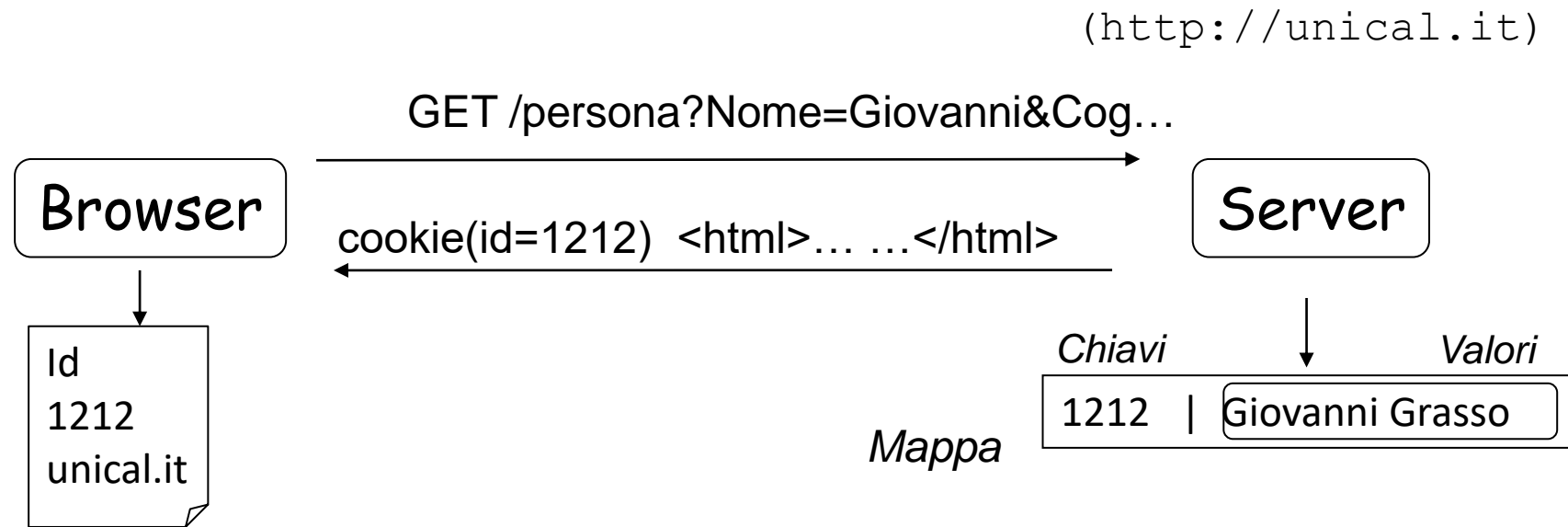
Il contenuto di un cookie

- Name
 - identificatore univoco all'interno di un domain:path
- Valore
 - stringa
- Domain
 - Il dominio in cui e' stato generato
- Path
 - posizione nel sito a cui e' associato
- Expiry/Max-age
 - Validita'

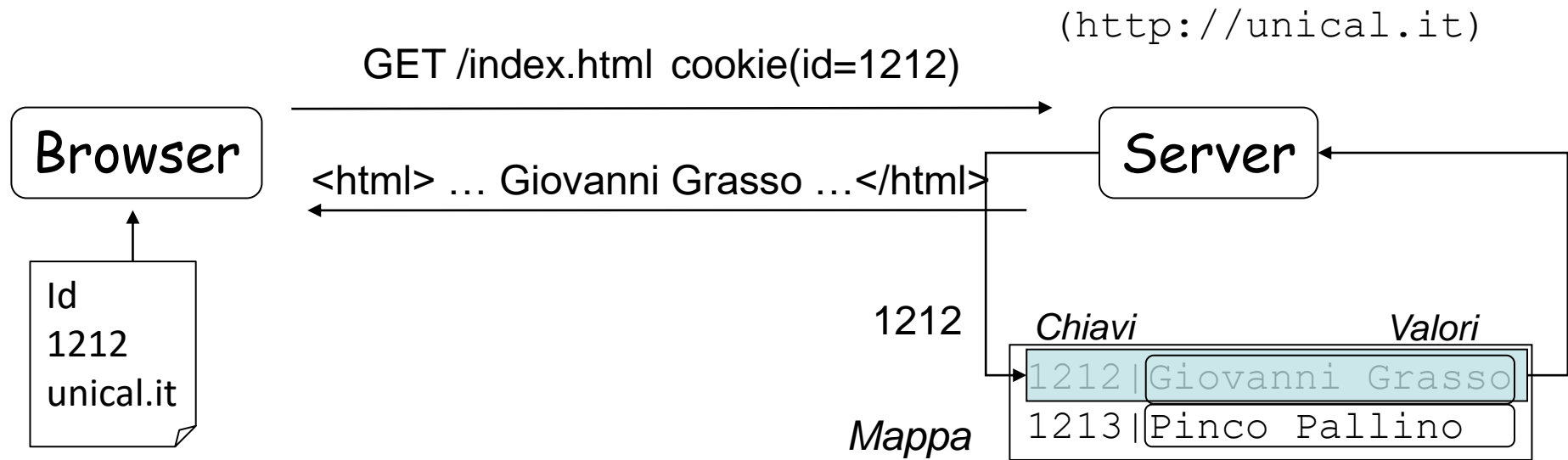
Cookies: funzionamento

- Quando il server riceve una richiesta mette un cookie nella risposta
 - attraverso l'intestazione HTTP "Set-Cookie"
- Il client può accettare o rifiutare il cookie
- Se lo accetta
 - lo salva sul disco locale
 - si impegna a re-inviarlo al server in ogni richiesta HTTP (per tutto il tempo specificato nella validità del cookie)
 - attraverso l'intestazione HTTP "Cookie"

Esempio



Esempio



Gestione della sessione

- Il trucco consiste nell'uso di una informazione "nascosta" scambiata tra server e client
 - il client si impegna a ritornare questa informazione al server ad ogni richiesta
 - il server usa questa informazione per ricostruire ai propri fini la storia delle interazioni con il client

Nota

- Il modello di riferimento è *client pull*
(o *request-reply*)
 - il server risponde ad una richiesta del client
- Si stanno diffondendo standard (HTML5+WebSocket) che permettono un approccio *server push*
 - il server può aggiornare i dati sul client

Caching

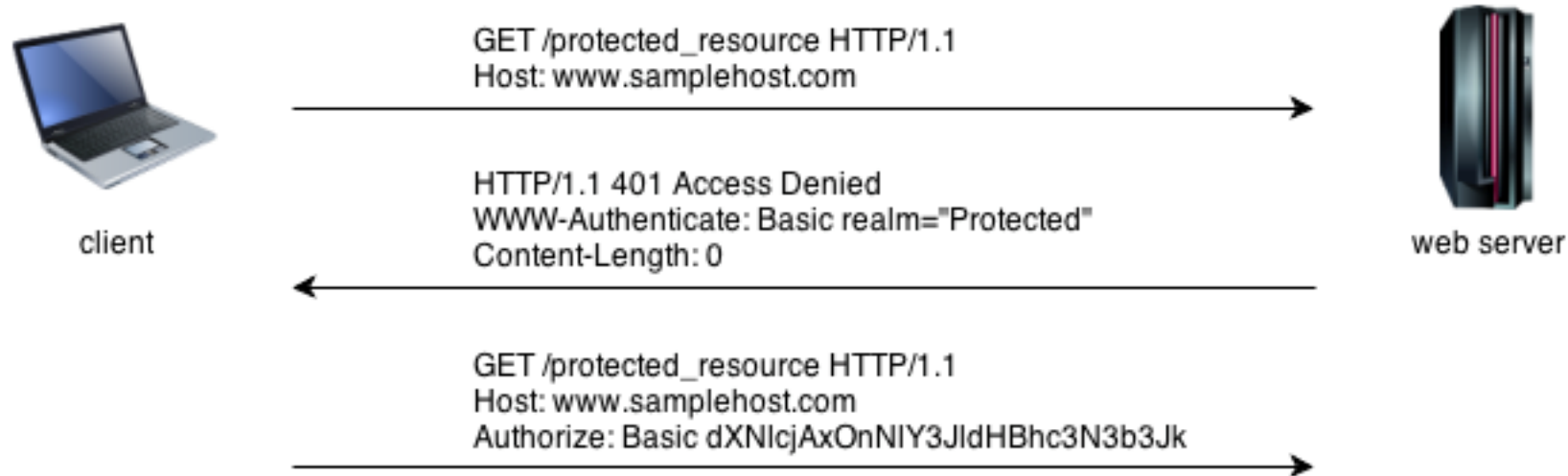
- Migliorare user-experience e ottimizzare risorse rete/server
 - Memorizzare contenuto che cambia poco frequentemente
 - Proxy Cache
- HTTP Cache
 - Implementato in tutti i browser moderni
 - Direttiva Cache-Control
 - Definisce la policy per ogni risorsa individualmente
 - Specifica chi puo' mettere in cache, a che condizioni, e per quanto tempo

Autenticazione

- Restringere l'accesso ad alcune risorse solo ad utenti abilitati
- Possibili tecniche
 - Filtro su IP
 - Vari svantaggi (NAT,DHCP, Spoofing)
 - HTTP Digest Authentication
 - Basata su hashing, caduta in disuso
 - Form di login con user / password
 - Le credenziali sono inviate IN CHIARO nel corpo del messaggio POST
 - HTTP Basic authentication
 - Le credenziali sono inviate in CHIARO ma codificate

Autenticazione Basic

- Il server chiede al client di autenticarsi per il reame
- Il client chiede nome utente e password all'utente
- Il client invia nome utente e password in chiaro al server (codificati come base64)



Sicurezza

- HTTP Basic e' insicuro perche' in chiaro
- HTTP Digest e' leggermente meglio ma vulnerabile
- Form Authentication e' vulnerabile (network sniffing)
- E' perciò necessario accoppiare l'autenticazione basic o form-based ad un canale di trasporto sicuro come SSL/TSL (crittografato) → HTTPS (Secure HTTP)

Risorse online

- <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- <https://www.jmarshall.com/easy/http/>

Prossima Lezione

Prima della successiva lezione:

- Scaricare i seguenti tool **senza installarli** (lo faremo insieme)
 - Scaricare **IntelliJ IDEA** (o qualunque altro IDE preferito).
 - Le esercitazioni saranno comunque espletate tramite IntelliJ
 - Scaricare la versione community: <https://www.jetbrains.com/edu-products/download/#section=idea>
 - Scaricare **Apache Tomcat 10** (<https://tomcat.apache.org/download-10.cgi>)
 - Scaricare **Postgres** (<https://www.postgresql.org/>)
 - Scaricare **Dbeaver** (<https://dbeaver.io/>)
 - In alternativa è possibile scaricare **pgAdmin** (<https://www.pgadmin.org/>)

Termini della licenza

- Questo lavoro si basa su:
 - materiale del W3C
 - materiale del prof. G. Mecca dell'Università degli Studi della Basilicata
 - materiale del prof. P. Meriando dell'Università degli Studi Roma Tre
 - materiale del prof. G. Grasso dell'Università della Calabria
- e viene concesso in uso secondo i termini della licenza “Attribution-ShareAlike” di Creative Commons.