UNIVERSITÀ DI PARMA

DIPARTIMENTO DI SCIENZE MATEMATICHE, FISICHE ED INFORMATICHE
Corso di Laurea in Informatica

# Il Livello Applicativo – Parte C
## WWW

RETI DI CALCOLATORI - a.a. 2023/2024

Roberto Alfieri

# Livello Applicativo: sommario

**PARTE A**

- Applicativi UDP: TFTP e DNS

**PARTE B**

- I servizi di posta elettronica: SMTP, POP e IMAP.

**PARTE C**

- Il World Wide Web

**PARTE D**

- Multimedia

# World Wide Web

**WWW** (World Wide Web) è una architettura client/server per la consultazione di documenti multimediali e ipertestuali distribuiti in rete.

L'architettura è nata nel 1989 al CERN di Ginevra e dal 1994 il suo sviluppo è gestito dal consorzio **W3C** (accordo CERN-MIT).

**HTML** (HyperText Markup Language) è il formato con cui vengono descritti gli ipertesti.

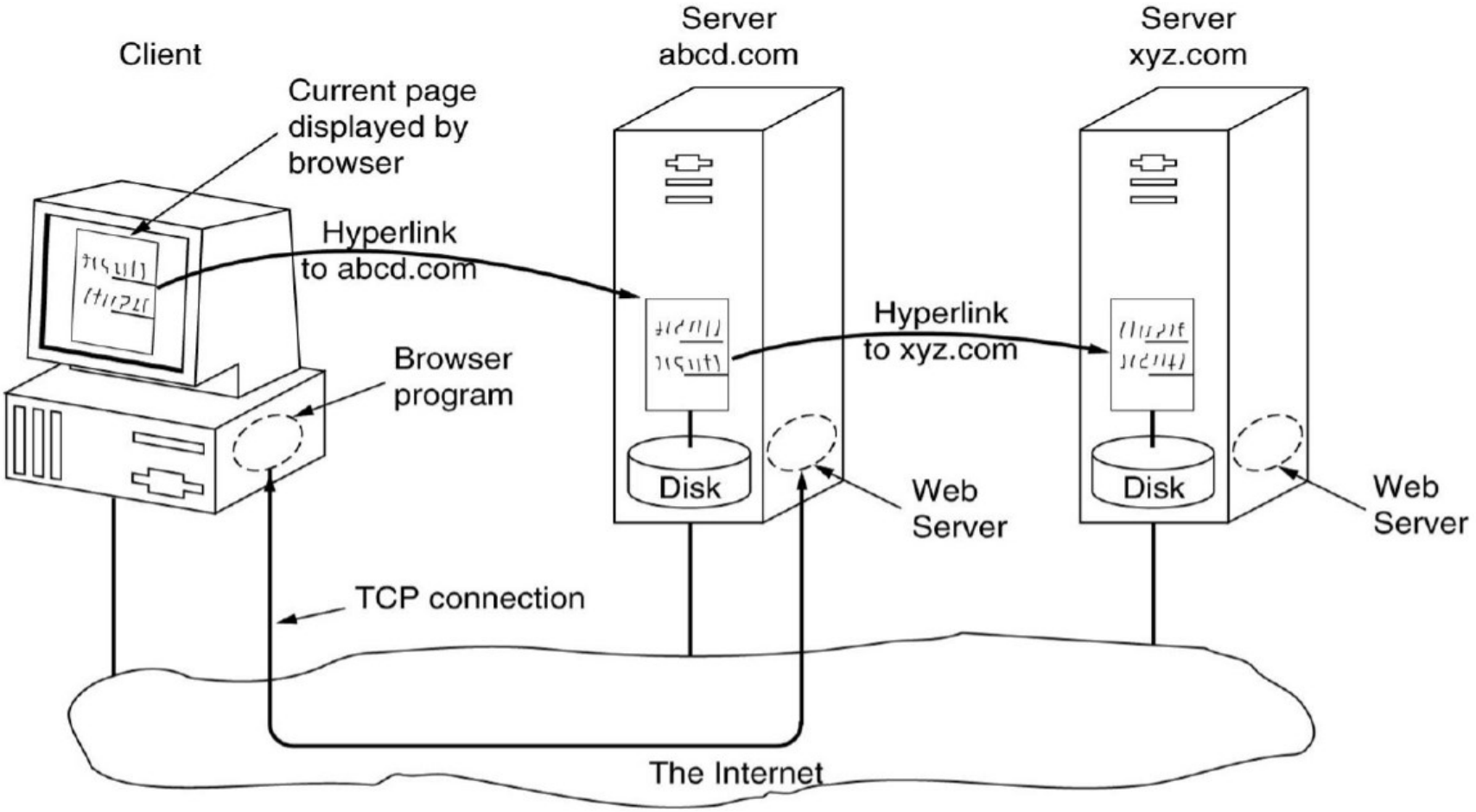
**HTTP** (HyperText Transfer Protocol) è il protocollo principale per la comunicazione tra **client** e **server**, anche se l'architettura WWW consente l'utilizzo di protocolli diversi.

Ogni documento WWW o singolo oggetto multimediale (file audio, immagine, ..) è identificato mediante un indirizzo univoco in Internet (**URL**) e può contenere riferimenti ipertestuali ad altri documenti.

I documenti possono essere statici (già presenti sul server al momento della richiesta) o dinamici (generati al momento della richiesta dal server o dal client)

L'utente accede ai documenti fornendo l'URL al programma client (Web Browser).

# Architettura WWW



# URL

Gli URL (Uniform Resource Locator) sono identificatori univoci di documenti WWW.

Sono composti da 4 parti  schema://NomeServer:port/NomeLocale

- **schema** è il  protocollo per raggiungere il documento
- **NomeServer** è il nome DNS del server che contiene il documento
- **Port** è la porta di ascolto del server
- **NomeLocale** è l'identificatore del documento sul server

Ad esempio:  http://www.company.com:81/a/b/c.html

http è il protocollo più utilizzato per l'accesso a documenti WWW.

Altri schema diffusi sono:

- ✓ https  è la versione cifrata del protocollo http
- ✓ ftp  ( vedi ad esempio qui: https://www.gnu.org/prep/ftp.html  )
- ✓ file (esempio  file:///C:/  identifica i file in  C: )

Riferimenti:

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

# Tipi Mime

Oltre ai documenti ipertestuali HTML l'architettura WWW supporta un numero sempre crescente di altri formati, denominati MIME-Types

Elenco di Mime-Type comuni:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

Per ogni Mime-Type il Browser avrà associata la modalità di gestione

Per alcuni formati il Browser conterrà l'interprete necessario per visualizzarli, per altri formati si appoggerà a componenti software esterne, i Plug-in o gli helper:

- ✓ - Un plug-in è un modulo esterno che il browser installa come estensione di se stesso
- ✓ - L'Helper (o applicazione di supporto) è un programma autonomo eseguito come processo separato a cui il browser passa il file da visualizzare.

# I Web Client

Prevalentemente **browser grafici** (Firefox, Chrome, IExplorer, ecc)
Dispongono di una Cache su disco per i documenti visitati di recente
Sequenza di operazioni:

- Input dell'URL
- Verifica se il documento è presente in Cache
- Risoluzione del nome DNS nell'indirizzo IP
- Apertura della connessione TCP
- Invio della richiesta al server mediante il protocollo indicato nello schema
- Download del documento
- Parsing del documento
- eventuale richiesta di documenti collegati
- Visualizzazione del documento direttamente o mediante viewer esterni
- Rilascio della connessione TCP (se non vi sono altre richieste entro breve termine)

Un client Web può essere utilizzato per il **download di documenti**.
Esempi in ambiente Linux o MacOsX: wget e curl

# I Web Server

E' un programma che si occupa di fornire, su richiesta del client browser, una pagina WWW.

**The Apache Software Foundation** è il nome di un gruppo di lavoro ( http://www.apache.org/ ) che sta portando avanti diversi progetti Open Source tra cui due tra i più diffusi Server Web: **Apache** e **Tomcat**.

Altri server moto diffusi: **NGINX**

Sequenza di operazioni base del server:

- Accetta la connessione TCP da un client
- Determina dall'URL il percorso del documento richiesto
- Accede al documento su disco
- Invio al client di intestazione (Mime-Type, .. ) e contenuto del documento
- Rilascio della connessione TCP

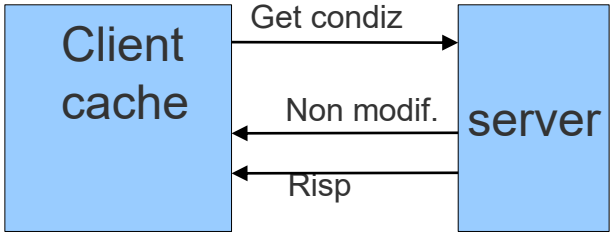
# Web Caching

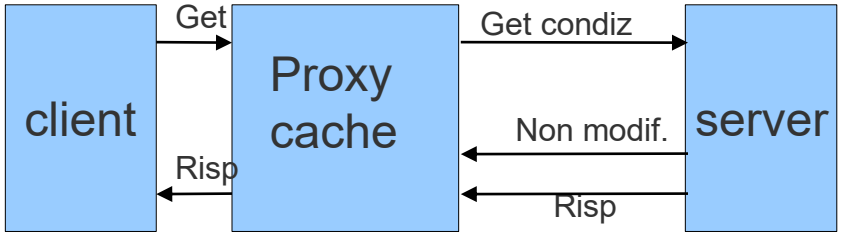
Fare Caching significa memorizzare temporaneamente le pagine in un punto più vicino al client per una **visualizzazione più rapida** e per **ridurre il carico del server.**

I **Browser** operano automaticamente caching sul disco locale dei documenti visitati.

Se il documento richiesto è presente sul disco locale viene fatto un GET condizionale (if-modified-since) in cui si chiede se esiste una versione più recente del documento.

Esempio: If-Modified-Since: Tue, 17 Mar 2015 07:00:00 GMT




Una LAN può organizzare un servizio di caching disponibile per tutti gli utenti della rete. Per utilizzare il servizio l'utente deve attivarlo esplicitamente.

Questo servizio di cache viene denominato **Proxy** poiché l'accesso al server che contiene il documento richiesto viene realizzato da Proxy/Cache server.

Il **Reverse Proxy** è un tipo di Proxy che recupera i contenuti per conto di un client da uno o più server. Questi contenuti sono poi trasferiti al client come se provenissero dallo stesso Reverse Proxy, che quindi appare al client come un Web server.

# Cache Expiration

Alcuni documenti HTML possono contente un'intestazione “Expires” che indica il tempo di validità del documento  e verrà utilizzata per decidere se utilizzare la copia o recuperare il documento dal server.

```
<head>
<META HTTP-EQUIV="expires" CONTENT="Mon, 24 Mar 2008 08:21:57 GMT">
</head>
```

Altri documenti usano l'intestazione “no-cache” per impedire che il documento venga inserito nella cache (tipicamente pagine dinamiche).

```
<head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
</head>
```

# Content Delivery Network (CDN)

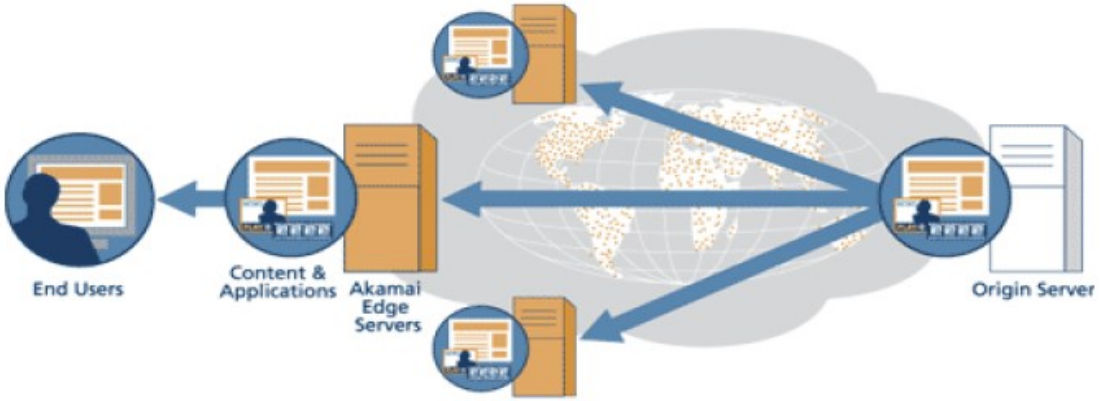
Per contenuti che devono essere distribuiti su scala globale (downloads, streaming, ..) esistono 2 approcci: Web caching e CDN.

Con il Web caching è il client a attivare le copie mentre con CDN è il provider che distribuisce copie dei contenuti in un insieme di nodi in differenti posizioni e redirige i client in modo che usino un nodo a lui vicino.

Possibili architetture per realizzare una rete CDN:

- **DNS redirection**: Il name server della CDN è gestito dalla CDN stessa. Il client ottiene dal DNS l'indirizzo Unicast della copia più vicina a lui.

- **Routing Anycast:** i mirror server hanno gli stessi indirizzi Anycast e il compito di trovare il server più vic




Una delle reti CDN più note è Akamai, con oltre 29000 server sparsi su circa 70 nazioni.

Riferimenti: https://www.akamai.com/it/our-thinking/cdn/what-is-a-cdn

# HTML

La maggior parte dei documenti WWW sono scritti in HTML.

HTML è un linguaggio di markup, ovvero contiene direttive di formattazione.

I contenuti sono testo formattato (font, colore, liste, tabelle, ecc) e eventuali riferimenti a oggetti esterni (immagini, video, suoni, hyper-link ad altri documenti)

Le direttive per la formattazione sono dette TAG e sono racchiuse tra parentesi angolari (esempio  testo evidenziato )

Il documento HTML è delimitato dal TAG <html>..</html> e comprende una intestazione <head> .. </head> e un corpo <body> .. </body>

Esempio:

```
<html>
 <head>
 <title> Le informazioni in Head non appaiono nel documento </title>
 </head>
 <body>
 <b> Nel Body viene scritto il testo formattato del documento </b> <p>
 <a href="http://www.unipr.it"> Questo e' un collegamento ipertestuale </a>
<p>
Le immagini vengono incluse il un documento nel seguente modo: <p>
 <img src="./immagine.png">
 </body>
</html>
```

# Standard HTML

HTML continua ad evolversi.

**HTML 2.0** è il primo vero standard di HTML rilasciato nel 1995 da **W3C**

**HTML 3.2** , rilasciato nel Gennaio 1997, è stato adottato in diversi browser.

**HTML 4.01**, rilasciato nel 1999 da W3C, è un'altra release con molte implementazioni

HTML 5, rilasciato da W3C nel 2014.

I Browser e Server Web hanno generalmente implementato le varie release HTML in modo più o meno rigoroso, sorvolando spesso su errori di formattazione e in alcuni casi, aggiungendo TAG non standard, funzionanti solo su alcune piattaforme.

**XHTML** è un linguaggio di markup che associa alcune proprietà dell'XML nell'HTML.

Un file XHTML è un pagina HTML scritta in conformità con lo standard XML.

# Style Sheet

HTML è un linguaggio di Markup che mescola il contenuto alla formattazione.

Infatti alcuni TAG descrivono il contenuto del documento, indipendentemente dalla sua rappresentazione finale (esempio <img src=..>  oppure <a href=.. > ), mentre altri TAG descrivono il modo in cui il documento dovrà apparire al lettore

(esempio <font color=”red”, face=”arial”> )

Con la crescente complessità e varietà di utilizzo dei documenti Web è sorta la necessità di separare i 2 aspetti del documento. Per questo motivo sono stati introdotti i “Fogli di Stile” (Style Sheet) che sono file associati ad un documento in cui vengono confinate le informazioni di formattazione.

CSS (Cascading Style Sheet) sono i fogli di stile supportati da HTML introdotti nel 1996 da W3C.

Esempio: mystyle.css

```
.piccolo {font-style:normal; font-size:12px; }
```

Esempio: mydoc.html

```
<html>
<head> <link rel="Stylesheet" type="text/css"  href="mystyle.css"> </head>
<body> <div class="piccolo"> ciao </div> </body>
</html>
```

# Il protocollo HTTP

E' un protocollo testuale di tipo request/response che utilizza il servizio 80/TCP.

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

Prima versione V0.9 rilasciata dal HTTP Working Group nel 1991 (obsoleto)
Versini successive: HTTP/1.0 rilasciato nel 1996 (obsoleto),

HTTP/1.1 del1997 , HTTP/2 del 2015 e HTTP/3 del 2022

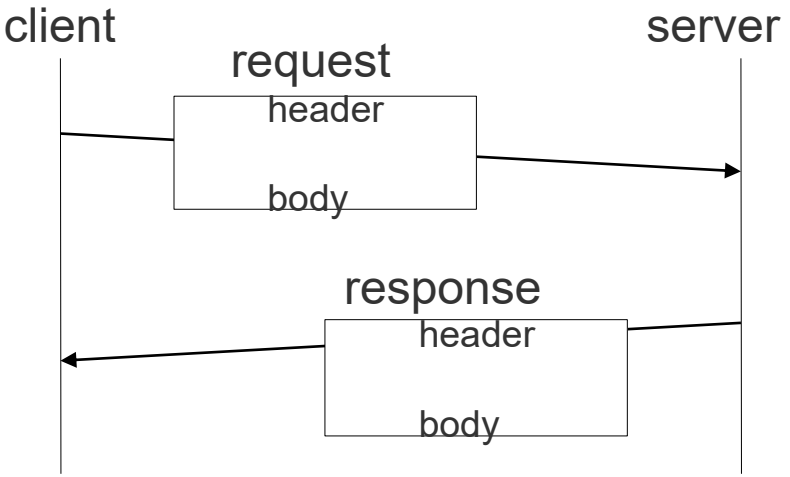
Deve trasportare un messaggio di richiesta dal client al server ed un messaggio di risposta dal server al client con il documento richiesto. Ogni messaggio è formato da una intestazione (Header) ed un corpo (Body) separati da riga vuota (CR LF).

```
<METHOD> <URI>  HTTP/1.0
<Header>: <Value>
<Header>: <Value>

<BODY>
```

```
HTTP/1.0 <STATUS_CODE><REASON>
<Header>: <Value>
<Header>: <Value>

<BODY>
```



# Metodi di richieste HTTP

**GET** : richiede tutte le informazioni disponibili per un determinato URL.
Il body del messaggio non è utilizzato.

```
GET /index.html HTTP/1.0
```

**HEAD**: Richiede solo l'header, senza la risorsa (il file HTML, l'immagine, ecc.).
Usato soprattutto per diagnostica.

```
HEAD /index.html  HTTP/1.0
```

**POST**: è stato concepito in origine per inviare al server molte informazioni (nel Body della richiesta) , senza un limite sulla quantità di dati da trasmettere e sul tipo, ed in modo non visibile da URL.

```
POST /prog.cgi  HTTP/1.0
Content-length: 10

01234566789
```

# Metodi di richieste HTTP

**OPTIONS:** Richiede l'elenco dei metodi permessi dal server

```
OPTIONS * HTTP/1.0
[riga vuota]
…
Allow: GET,HEAD,POST,OPTIONS,TRACE
```

**TRACE:** Traccia una richiesta, visualizzando come viene trattata dal server.

**DELETE:** Cancella una risorsa (file) sul server. L'utente con cui gira il web server deve poter avere permessi in scrittura sul file indicato e il server deve essere configurato per poterlo fare.

**PUT:** Upload di un file sul server con il nome indicato e i contenuti specificati nel body.

# Principali intestazioni HTTP

| Header | Type | Contents |
|---|---|---|
| User-Agent | Request | Information about the browser and its platform |
| Accept | Request | The type of pages the client can handle |
| Accept-Charset | Request | The character sets that are acceptable to the client |
| Accept-Encoding | Request | The page encodings the client can handle |
| Accept-Language | Request | The natural languages the client can handle |
| Host | Request | The server's DNS name |
| Authorization | Request | A list of the client's credentials |
| Cookie | Request | Sends a previously set cookie back to the server |
| Date | Both | Date and time the message was sent |
| Upgrade | Both | The protocol the sender wants to switch to |
| Server | Response | Information about the server |
| Content-Encoding | Response | How the content is encoded (e.g., gzip) |
| Content-Language | Response | The natural language used in the page |
| Content-Length | Response | The page's length in bytes |
| Content-Type | Response | The page's MIME type |
| Last-Modified | Response | Time and date the page was last changed |
| Location | Response | A command to the client to send its request elsewhere |
| Accept-Ranges | Response | The server will accept byte range requests |
| Set-Cookie | Response | The server wants the client to save a cookie |
| **If-modified-since** | **Request** | **Allows a 304 Not Modified to be returned if content is unchanged.** |

http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

# Risposte del Server

La prima riga della risposta del server contiene un codice che classifica la 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 |

# Esempio di richiesta/risposta con telnet

**Richiesta del Client:**

telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /index.html HTTP/1.0
[riga vuota]

**Risposta del server:**

HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 24 Nov 2022 11:34:55 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Wed, 23 Nov 2022 08:24:08 GMT
Connection: close
ETag: "637dd8a8-264"
Accept-Ranges: bytes

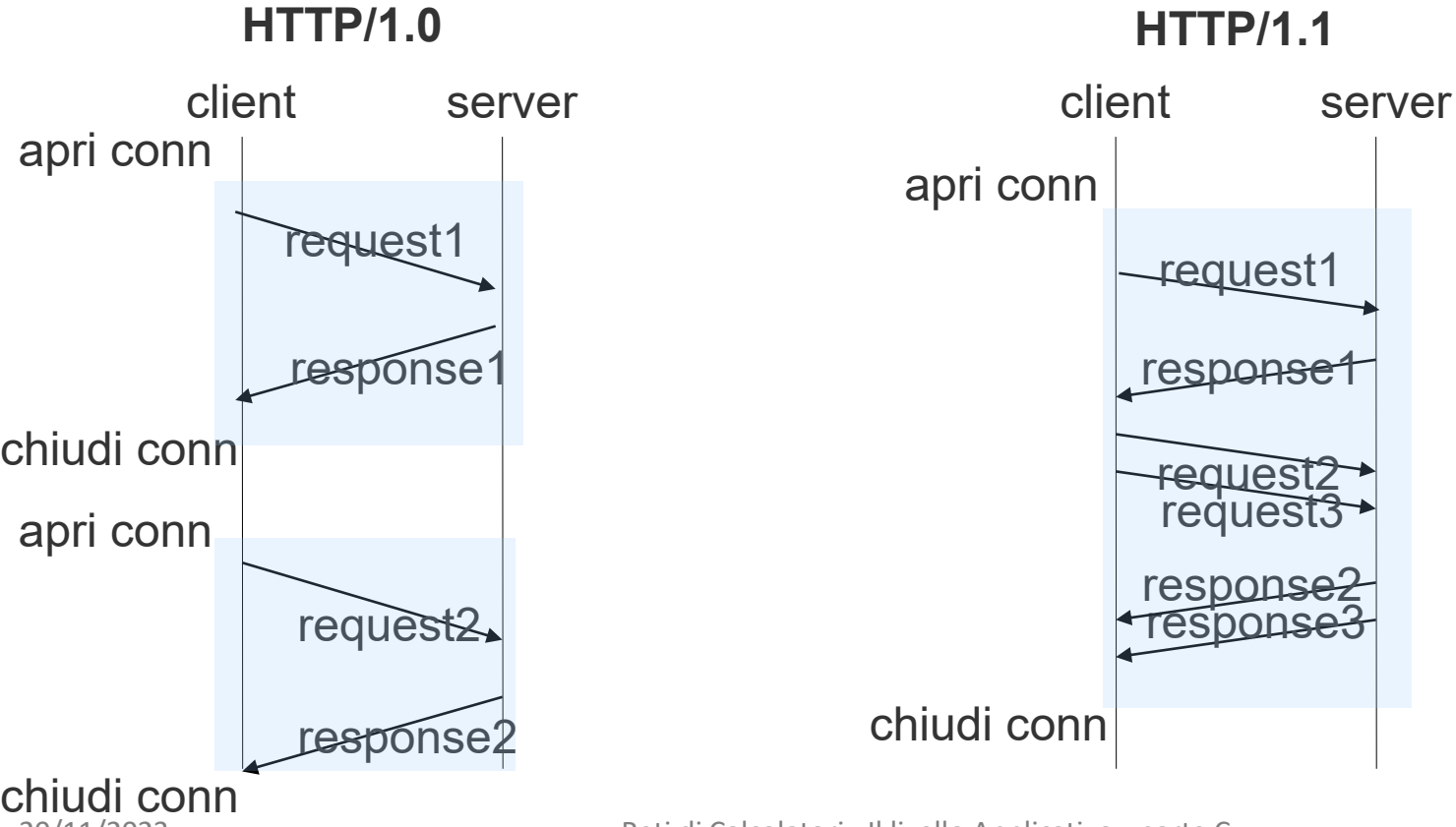
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body>
<h1>Welcome to nginx!</h1>
If you see this page, the nginx web server is successfully ins
working. Further configuration is required.

# HTTP 1.1: connessioni persistenti e parallele

HTTP 1.0 richiede la disconnessione TCP tra 2 richieste successive.

HTTP 1.1 supporta connessioni persistenti (riutilizzo di una conn. per diverse richieste)

HTTP 1.1 supporta anche richieste parallele



# HTTP 1.1: Chunked Transfering Encoding

In HTTP 1.0 chi spedisce un dato include nell'intestazione la dimensione in byte del dato stesso, mediante l'intestazione "**Content-length: nn**".

Chunked Transfer Encoding è una modalità di trasferimento introdotta in HTTP 1.1 in cui i dati vendono inviati in una serie di "chunk".

Viene utilizzata per la trasmissione di dati generati dinamicamente, di cui non conosciamo la lunghezza prima di iniziare la trasmissione, come ad esempio gli eventi in steaming Si utilizza "**Transfer-Encoding: chunked**" invece di "Content-length: nn"

La dimensione di ogni chunk viene inviata appena prima del chunk stesso in modo che il ricevente possa capire quando ha finito di ricevere il chunk.

Il trasferimento termina con un chunk finale pari a 0.

Riferimento:

http://en.wikipedia.org/wiki/Chunked_transfer_encoding

```
HTTP/1.1 200 OK
Date: Mon, 22 Mar 2004 11:15:03 GMT
Content-Type: text/html
Transfer-Encoding: chunked

29
<html><body><p>The file you requested is
5
3,400
23
bytes long and was last modified:
1d
Sat, 20 Mar 2004 21:12:00 GMT
13
.</p></body></html>
0
```

# Mancanza di stato e Cookie

Il Web è privo di stato: se un Browser richiede più documenti da un server ogni richiesta è indipendente; il server non ricorda i contatti precedenti.

In alcuni casi però  sarebbe utile avere “memoria”. Ad esempio se l’accesso ai documenti richiede autenticazione , autorizzazione, ecc.

I Cookie sono stati introdotti da Netscape e formalizzati in RFC 2109 per risolvere il problema.

I Cookie sono generati dal server e scaricati assieme al documento.

Il browser li memorizza in una opportuna directory, ma volendo li può disabilitare.

Informazioni contenute in un Cookie:

| Domain | Path | Content | Expires | Secure |
|---|---|---|---|---|
| toms-casino.com | / | CustomerID=497793521 | 15-10-02 17:00 | Yes |
| joes-store.com | / | Cart=1-00501;1-07031;2-13721 | 11-10-02 14:22 | No |
| aportal.com | / | Prefs=Stk:SUNW+ORCL;Spt:Jets | 31-12-10 23:59 | No |
| sneaky.com | / | UserID=3627239101 | 31-12-12 23:59 | No |

# Contenuto dei Cookie

**Contenuto** (Nome/valore) è una variabile ed un campo obbligatorio.

**Expire** (Scadenza) è un attributo opzionale che permette di stabilire la data di scadenza del cookie. Può essere espressa come data, come numero massimo di giorni oppure come Now (adesso) (implica che il cookie viene eliminato subito dal computer dell'utente in quanto scade nel momento in cui viene creato) o Never (mai) (implica che il cookie non è soggetto a scadenza e questi sono denominati persistenti).

**Dominio e Percorso** definiscono l'ambito di visibilità del cookie, indicano al browser che il cookie può essere inviato al server solo per il dominio e il percorso indicati. Se non specificati, come predefiniti prendono il valore del dominio e del percorso che li ha inizialmente richiesti.

**Secure** (Sicuro) indica se il cookie debba essere trasmesso criptato con HTTPS.

# Come funzionano i Cookie

La prima volta che viene richiesto un URL il server invia i cookie al client inserendoli nell'intestazione, assieme al documento. Ad esempio:

```
HTTP/1.0 ….
Set-Cookie: tuocodice=1234567; expires=Tue, 18-Mar-08 18:43:09 GMT
Set-Cookie: tuonome=Mario; expires=Tue, 18-Mar-08 18:43:09 GMT
```

Il client memorizza i cookie ricevuti.

Quando l'utente torna a visitare la pagina il Browser cerca tra i Cookie che ha memorizzato un Cookie (non scaduto) con lo stesso dominio dell'URL.

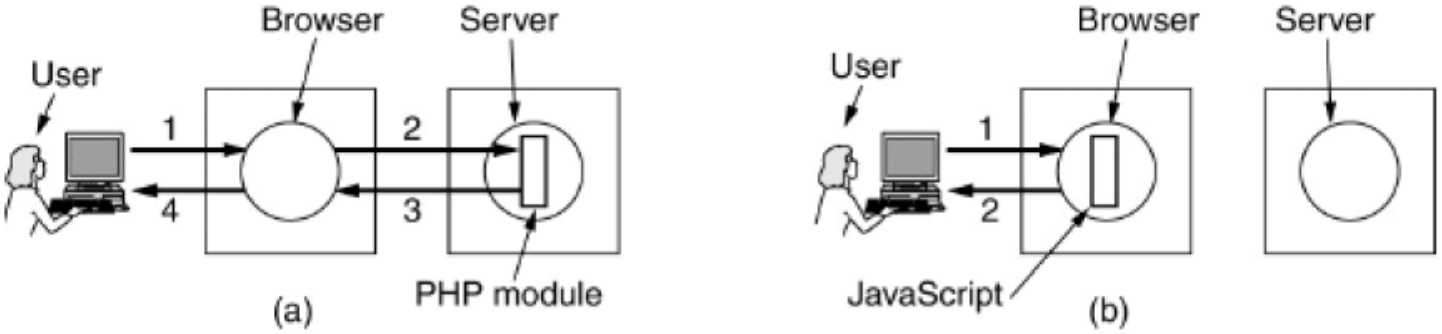
Se esiste viene fatto Upload del Cookie nell'Header assieme alla richiesta:

```
GET ….  HTTP/1.0
Cookie:  nome1=valore1 ; nome2=valore2
```

# Pagine Web statiche e dinamiche

- Le **pagine statiche** sono file sul disco del server che vengono spediti al client assieme ad una intestazione (content-type ecc)

- **Pagine dinamiche**: il documento viene generato in tempo reale, su richiesta. La generazione è eseguita da un programma che può essere eseguito
  - **dal server**
    - via CGI (programmi esterni richiamati dal server)
    - scripting PHP, JSP, ASP (codice incorporato in HTML interpretato dal server)
  - **dal client (pagine attive)**
    - Javascript
    - Java Applet (richiede JVM sul client)
    - ActiveX (tecnologia Microsoft, codice compitato per Intel)



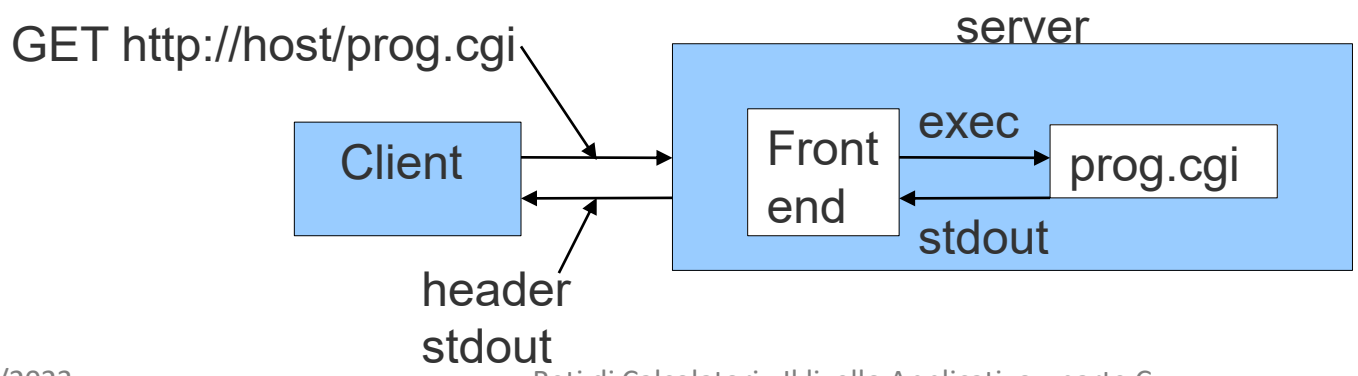
# Pagine dinamiche con CGI

Il protocollo **CGI** (Common Gateway Interface, RFC 3875) consente di mettere in esecuzione un programma eseguibile sul server e di redirigere lo standard output del programma verso il client il quale lo interpreterà come una normale risposta http.

Per attivare un programma il client utilizza lo stesso modello URL utilizzato per il riferimento alle pagine statiche, con i metodi GET o POST.

Il server può riconoscere un programma cgi in base alla sua estensione (e.g. .cgi) o alla sua posizione (es /cgi-bin/.. )

Normalmente l'output è in formato HTML, ma può assumere anche altre forme (immagini, dati binari, istruzioni particolari per il browser, ..)



# Passaggio dei parametri

L'esecuzione di una pagina dinamica deve prevedere la possibilità di passare parametri o dati all'eseguibile.

Questo può avvenire con 2 metodi alternativi: GET e POST.

-**Il metodo GET** codifica i parametri all'interno della stringa URL

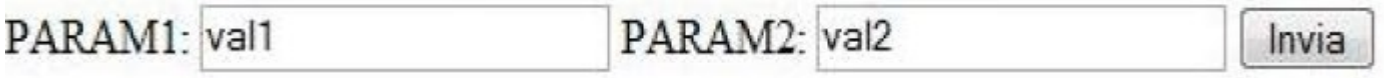
Esempio: http://host/prog.cgi?param

-**Il metodo POST** utilizza la parte Body della richiesta.

L'HTML fornisce diversi TAG per la codifica di parametri nella forma NOME=valore da passare con GET o POST

Ad esempio:

```
<FORM ACTION="http://host/prog.cgi " METHOD=GET>
PARAM1: <INPUT TYPE="text"  NAME="param1">
PARAM2: <INPUT TYPE="text"  NAME="param2" >
<INPUT TYPE="submit" VALUE="Invia">
</FORM>
```




Genera la URL: http://host/prog.cgi?param1=val1&param2=val2

# Passaggio di dati con il metodo GET

Con il metodo GET la stringa di query (input) è inserita in coda alla URI del documento preceduta dal carattere “?”

**GET http://host/prog.cgi?param1=val1&param2=val2 HTTP/1.0**
**header..**
**header..**
**<cr><lf>**

Il programma riceve la stringa attraverso la variabile di ambiente QUERY_STRING:
QUERY_STRING=”param1=val1&param2=val2”

# Passaggio di dati con il metodo Post

I parametri passati con il metodo POST vengono inseriti nel Body della richiesta HTTP e il programma li riceve attraverso lo Standard Input.

Ad esempio, se scegliamo il metodo POST nella FORM dell'esempio precedente:

<FORM ACTION="http://host/prog.cgi " METHOD=POST> ...

Otteniamo una richiesta HTTP del tipo:

**POST http://host/prog.cgi HTTP/1.0**

**header..**

**Content-length: 23**

**<cr><lf>**

**param1=val1&param2=val2**

Questo metodo ha 2 vantaggi nel passaggio dei parametri rispetto al metodo GET:

- ✓ i parametri non compaiono nella URI (e quindi non vengono tracciati nei file di log)
- ✓ è possibile trasferire non solo parametri, ma anche dati

# Pagine dinamiche lato server con PHP

I server web supportano anche la possibilità di incorporare piccoli script all'interno del codice HTML che verrà eseguito al momento della consultazione della pagina.

Questo consente di realizzare documenti in cui solo una parte è dinamica.

Un linguaggio molto utilizzato per questo scopo è il PHP.

Esempio:

```
<html>
<head>
</head>
<body>
<FORM  ACTION="http://host/prog.php " METHOD=GET>
PARAM1: <INPUT TYPE="text"  NAME="param1">
PARAM2: <INPUT TYPE="text"  NAME="param2" >
<INPUT TYPE="submit" VALUE="Invia">
</FORM>
</body>
</html>
```

```
<html>
<body>
<h1>Dati inseriti:</h1>
<? php echo $param1; ?>
<? php echo $param2; ?>
</body>
</html>
```

# Pagine dinamiche lato client: javascript

In altre applicazioni è utile che il codice venga eseguito lato client.

Anche in questo caso viene  incorporato codice di script nella pagina HTML.

Il linguaggio più popolare lato client è JavaScript.  Esempio:

```
<html>
<head>

<script language=”javascript” type=”text/javascript”>
Function response (test_form) {
...
}
</script>

</head>
<body>
<form>
PARAM1: <INPUT TYPE="text"  NAME="param1">
PARAM2: <INPUT TYPE="text"  NAME="param2" >
<input type=”button” value=”submit” onclic=”response(this.form)”>
</form>
</body>
</html>
```

# Pagine dinamiche lato client: Applet

Un altro metodo molto diffuso è l'utilizzo di Applet Java.

E' necessario che il browser includa una JVM (quasi tutti i Browser)

Le Applet sono più portabili perché la JVM è la stessa su diverse piattaforme, mentre il supporto JavaScript può differire da una Browser all'altro.

Le Applet possono essere incorporate nelle pagine HTML:<applet> … </applet>

Esempio:

```
<html>
<body>
<applet code="PrimoApplet.class">
</applet>
</body>
</html>
```

```
import java.applet.*;
import java.awt.*;

public class PrimoApplet extends Applet
 {
   public void paint (Graphics g)
   {
   g.drawString("Ciao, io sono il primo applet.",0,50);
   }
 }
```