



# Dal Web ai Webservices

Prof. Davide Quaglia





# HTTP/HTTPS

- Inventato da Tim Berners-Lee al CERN di Ginevra nel 1989
- Nato per la fruizione di contenuti in rete (World Wide Web)
- Oggi usato anche per l'invocazione di funzionalità remote →  
**Webservice**

# Esempio di HTTP/HTTPS

- Il client si chiama “web browser” (o semplicemente “browser”)
  - Firefox, Chrome, Edge, Safari, Opera, ...
- Il server si chiama “web server”
  - Apache, NGINX, NodeJS
- La comunicazione avviene sul protocollo TCP sulla rete Internet

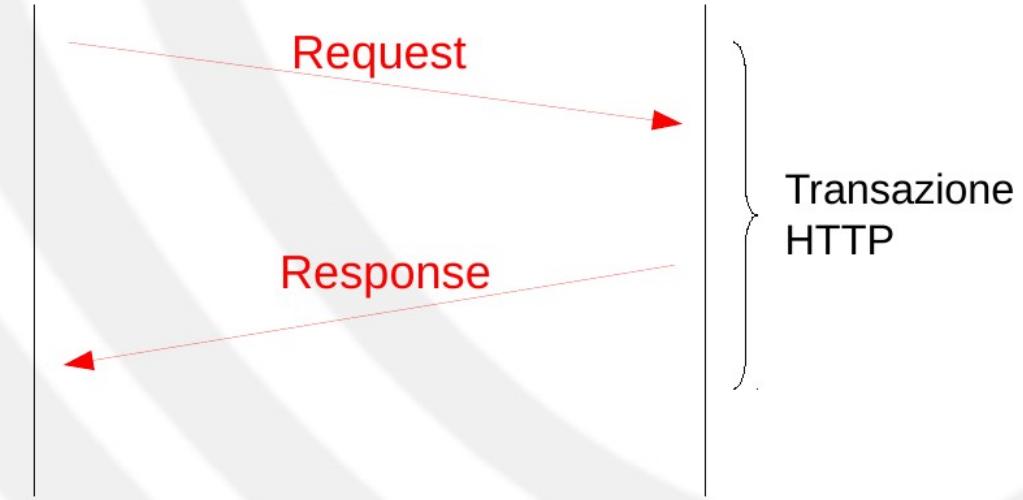


# Protocollo HTTP/HTTPS

- Client/Server
- Testuale
- Fasi
  - 1) Apertura di una connessione TCP
  - 2) [HTTPS] Autenticazione del server e negoziazione di una chiave di cifratura
  - 3) Richiesta e risposta
  - 4) Chiusura della connessione TCP

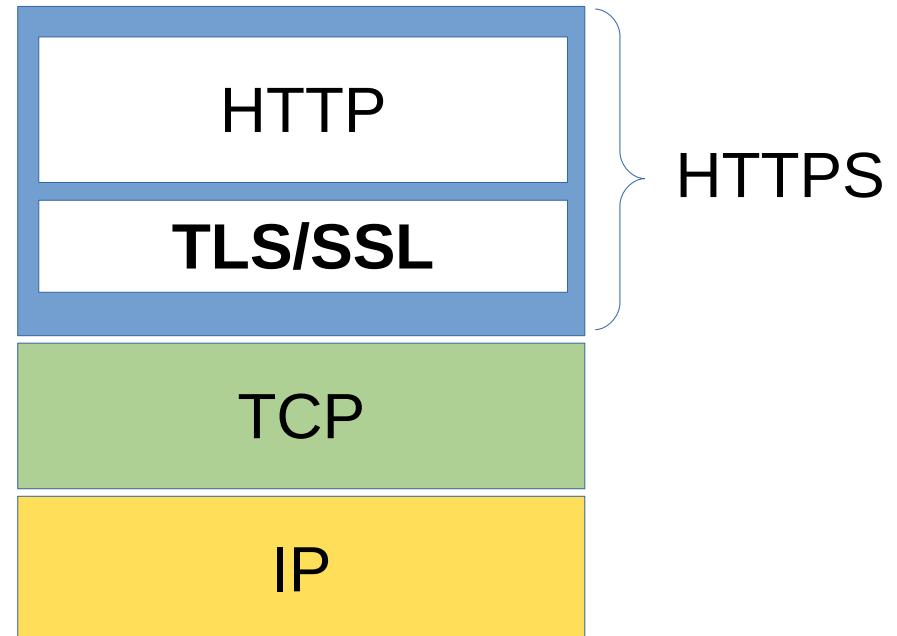
Browser Web (client)  
IP: 157.27.12.5 Porta TCP: 3500

Server Web (server)  
IP: 130.192.16.20 Porta TCP: 80



# HTTPS

- I messaggi che passano nella connessione TCP sono gli stessi dell'HTTP ma vengono sottoposti a
  - Cifratura dei dati in transito
  - Autenticazione del server mediante certificato digitale
- Il server lavora sulla porta 443 invece che 80



# Esempio: richiesta

Metodo HTTP

URL

GET

/it/i-nostri-servizi/servizi-per-studenti

HTTP/1.1

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8)  
Gecko/20100214 Ubuntu/9.10 (karmic) Firefox/3.5.8

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

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

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Connection: keep-alive

!!!Riga vuota!!!

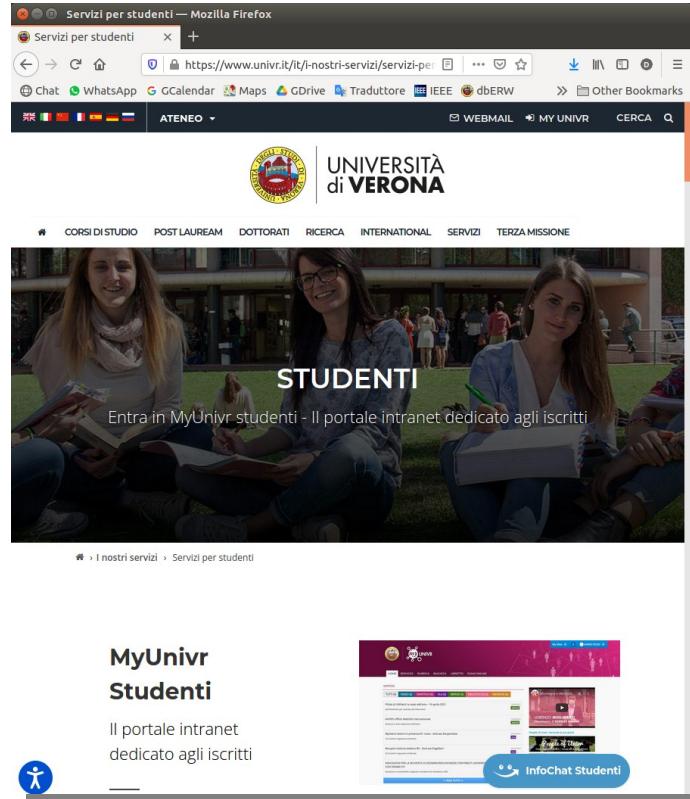
# Esempio: risposta

```
HTTP/1.1 200 OK
Date: Mon, 17 May 2022 16:10:48 GMT
Server: Apache
Last-Modified: Mon, 29 Mar 2022 13:57:17 GMT
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
    !!!Riga vuota!!!
<html>
...
</html>
```

} Intestazione della risposta

} Corpo della risposta

# Esempio: visualizzazione della pagina



- Il corpo della risposta può contenere:
  - HTML puro
  - HTML + codice Javascript
  - Sequenza binaria che rappresenta un'immagine
  - Cascading Style Sheets (CSS)
  - Intere librerie di codice Javascript da eseguire sul browser

# Hyper Text Markup Language (HTML)

- Linguaggio testuale di descrizione di una pagina
- Specializzazione del generico XML (eXtensible Markup Language)
- Basato su “tag” annidati eventualmente contenenti attributi

prova.html nella cartella Esempi-web/

```
<!-- My document -->
<HTML>
<HEAD>
  <TITLE>My Document</TITLE>
</HEAD>
<BODY>
  <H1>Header</H1>
  <P>
    Testo <b>grassetto</b>
  </P>
  <P>
    Testo <i>corsivo</i>
  </P>
</BODY>
</HTML>
```



# Cascading Style Sheets (CSS)

css.html

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  <link rel="stylesheet" href="styles.css">  
  
</head>  
  
<body>  
  
  <h1>This is a heading</h1>  
  
  <p>This is a paragraph.</p>  
  
</body>  
  
</html>
```

styles.css

```
body {  
  background-color: powderblue;  
}  
  
h1 {  
  color: blue;  
}  
  
p {  
  color: red;  
}
```

# Cascading Style Sheets (CSS)

css.html

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

styles.css

```
body {
  background-color: powderblue;
}

h1 {
  color: blue;
}

h1 {
  color: red;
}
```

Provate a togliere questo tag oppure a cambiare il contenuto di styles.css



# HTML: tag per richiamare immagini

## immagini.html

```
<!DOCTYPE html>

<html>
<body>

<h2>I trulli di Alberobello</h2>



</body>
</html>
```

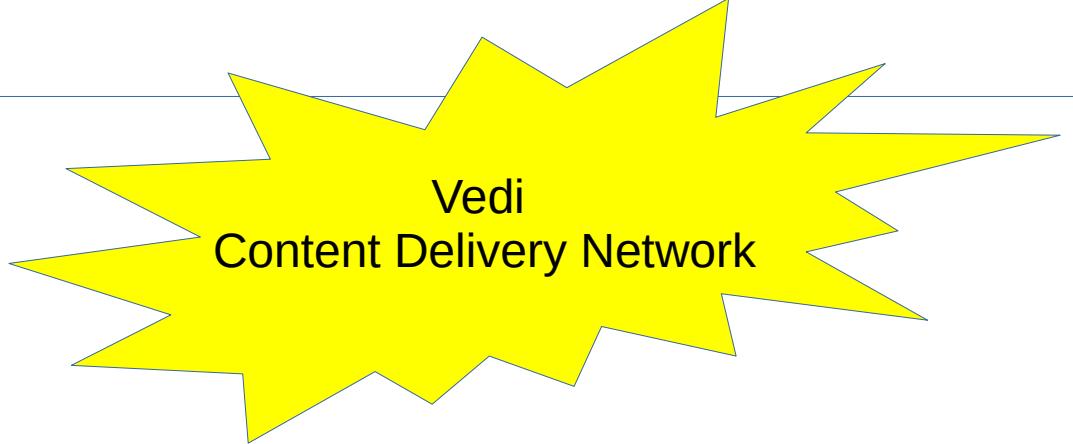
# HTML: tag per richiamare immagini

immagini.html

```
<!DOCTYPE html>
<html>
<body>

<h2>I trulli di Alberobello</h2>


</body>
</html>
```



Vedi  
Content Delivery Network



Provate a sostituire questa URL con  
**<https://cdn.britannica.com/70/20070-050-C2E2045C/Central-Park-Manhattan-New-York-City-apartment.jpg>**

# HTML: tag per il collegamento ipertestuale

link.html

```
<!DOCTYPE html>

<html>
<body>

<h1>HTML Links</h1>

<p><a href="https://www.w3schools.com/">Visit w3Schools.com!</a></p>

</body>
</html>
```

# HTML: tag per il collegamento ipertestuale

link.html

```
<!DOCTYPE html>

<html>
<body>

<h1>HTML Links</h1>

<p><a href="https://www.w3schools.com/">Visit w3Schools.com!</a></p>

</body>
</html>
```

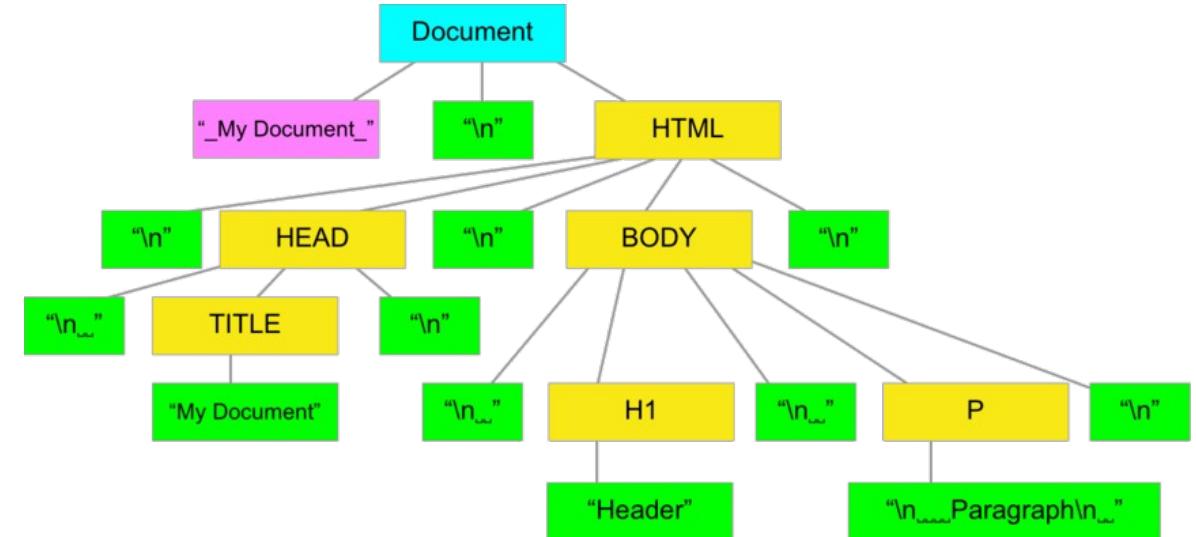


Provate a far diventare il link di colore rosso!

# Document Object Model (DOM)

- rappresentazione della pagina HTML come modello orientato agli oggetti
- standard ufficiale del W3C

```
<!-- My document -->
<HTML>
<HEAD>
  <TITLE>My Document</TITLE>
</HEAD>
<BODY>
  <H1>Header</H1>
  <P>
    Paragraph
  </P>
</BODY>
</HTML>
```





# Javascript

## javascript.html

```
<!DOCTYPE html>
<html>
<body>

<h2>Use JavaScript to Change Text</h2>
<p>This example writes "Hello JavaScript!" into an HTML element with id="demo":</p>

<p id="demo"></p>

<script>
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>

</body>
</html>
```

# Javascript: cosa e dove?

- Codice simil-Java
  - direttamente contenuto nelle pagine HTML attraverso il tag <script>

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
```

- interi file di codice (librerie) richiamati nell'HTML e poi scaricati dal browser

```
<script src="myscripts.js"></script>
```

# Javascript ed eventi

- Il codice Javascript può essere associato a degli eventi che l'utente genera interagendo con la pagina
  - Pressione di un bottone
  - Selezione di una casella
  - ...



# Javascript ed eventi: esempio

javascript-evento.html

```
<!DOCTYPE html>
<html>
<body>

<button
onclick="document.getElementById('demo').innerHTML=Date()"
>Che ora è?
</button>

<p id="demo"></p>

</body>
</html>
```



# Javascript ed eventi: esempio

javascript-evento.html

```
<!DOCTYPE html>
<html>
<body>

<button
onclick="document.getElementById('demo').innerHTML=Date()"
>Che ora è?
</button>

<p id="demo"></p>

</body>
</html>
```

Provate a cliccare più volte sul bottone.  
Cosa succede?



# Javascript e Document Object Model (DOM)

- Il DOM trasforma una pagina web da documento statico a Graphical User Interface (GUI)
- il codice Javascript contenuto nella pagina HTML ed eseguito dal browser può agire sull'oggetto che rappresenta la pagina e modificarla → una pagina si automodifica come reazione a certe azioni scatenate dall'utente → comportamento simile ad una applicazione → web application
- Programmazione “lato client” (o “client side” o “lato front-end”)
- Alternative a Javascript
  - Typescript



# Javascript e DOM

javascript-load.html

```
<!DOCTYPE html>
<html>
<body>

<iframe id="area" height="2000" width="1000"></iframe>

<script>
  document.getElementById("area").src =
"https://www.ansa.it/sito/notizie/topnews/index.shtml";
</script>

</body>
</html>
```



# Javascript e gestione del tempo

javascript-timer.html

```
<!DOCTYPE html>
<html>
<body>

<h1>The Window Object</h1>
<h2>The setInterval() Method</h2>

<p id="demo"></p>

<script>
setInterval(displayHello, 1000);

function displayHello() {
  document.getElementById("demo").innerHTML += "Hello";
}
</script>

</body>
</html>
```



# Javascript e gestione del tempo

javascript-timer2.html

```
<!DOCTYPE html>
<html>
<body>

<h1>The Window Object</h1>
<h2>The setInterval() Method</h2>

<p id="demo"></p>

<script>
setInterval(function() {document.getElementById("demo").innerHTML += "Hello"}, 1000);
</script>

</body>
</html>
```

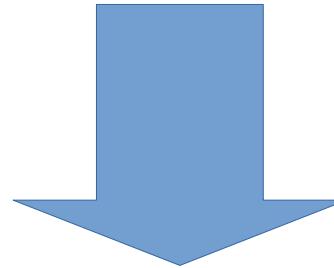


# Esercizio

- Scrivere e provare una pagina HTML che ricarica periodicamente il sito dell'ANSA

# Esercizio

- Scrivere e provare una pagina HTML che ricarica periodicamente il sito dell'ANSA



## APPROCCIO SINCRONO AL REFRESH DEI CONTENUTI

- in gergo informatico si dice “polling”
- non è detto che quel periodo di refresh sia quello giusto
  - refresh inutili con spreco di risorse di rete e calcolo
  - mancato refresh quando davvero serve
- l'alternativa è un approccio **ASINCRONO** o **EVENT-DRIVEN**  
(vedi programmazione mediante Web Socket)



# Uniform Resource Locator (URL)

- Detto anche Universal Resource Locator
- Stringa che permette di identificare in maniera univoca una risorsa HTTP in qualsiasi parte della rete mondiale

<https://www.univr.it/servizi/studenti/carriera>

- Struttura
  - Protocollo utilizzato a livello applicazione → protocollo di livello trasporto + porta utilizzata (HTTP → TCP/80, HTTPS → TCP/443)
  - Nome dell'host (o indirizzo IP) che eroga tale risorsa
  - Nome della risorsa con suo **percorso logico** completo (non necessariamente fisico)
- La porta può essere indicata esplicitamente se non è quella standard

<https://www.univr.it:8000/servizi/studenti/carriera>

# Un semplice web server

- Aprire il file `serverHTTP.c` in `Esempi-web/` e analizzarne il contenuto
- Compilarlo come nell'esercitazione sull'interfaccia socket ed eseguirlo
- Provare ad aprire un secondo terminale nella stessa cartella e a rilanciare lo stesso server. Funziona? Perché?
- Aprire il browser preferito e impostare la URL  
`"http://127.0.0.1:8000/"` Cosa si vede sul browser e sul terminale?
- Aprire il browser preferito e impostare la URL  
`"http://localhost:8000/"` Cosa cambia?

## form-get.html

```
<!DOCTYPE html>
<html>
<body>

<h2>The method Attribute</h2>

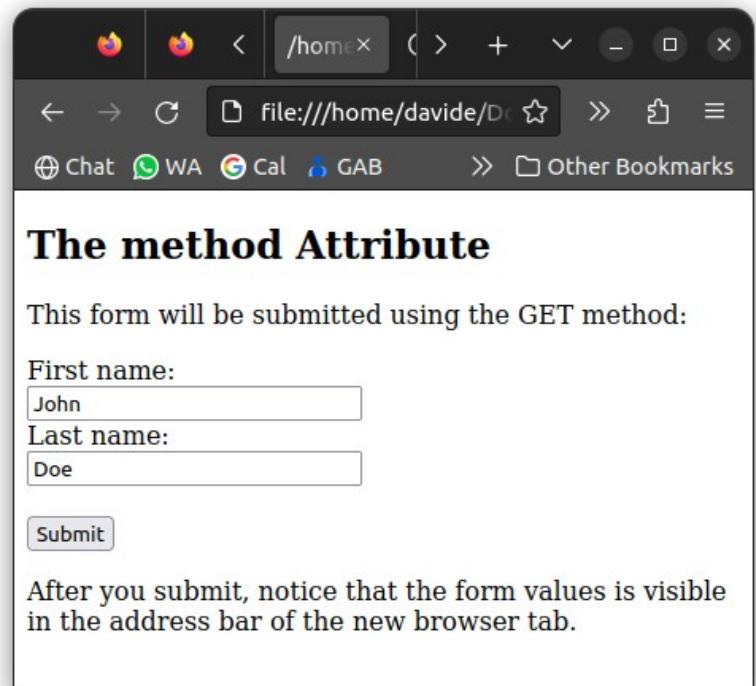
<p>This form will be submitted using the GET method:</p>

<form action="/action" target="_blank" method="get">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>

<p>After you submit, notice that the form values is visible in the
address bar of the new browser tab.</p>

</body>
</html>
```

# Passare dei dati al server web col metodo GET



# Esercizio

- Eseguire il server web `serverHTTP.c` e con il browser preferito aprire il file `form-get.html`
- Cosa si vede?
- Provare ad analizzare il contenuto della connessione TCP con Wireshark.
- Cosa si vede?



# E corrispondente richiesta HTTP

→ Parametri (max 2048 caratteri)

GET /action?fname=John&lname=Doe HTTP/1.1

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8)

Gecko/20100214 Ubuntu/9.10 (karmic) Firefox/3.5.8

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

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

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Connection: keep-alive

!!!Riga vuota!!!

form-post.html

```
<!DOCTYPE html>
<html>
<body>

<h2>The method Attribute</h2>

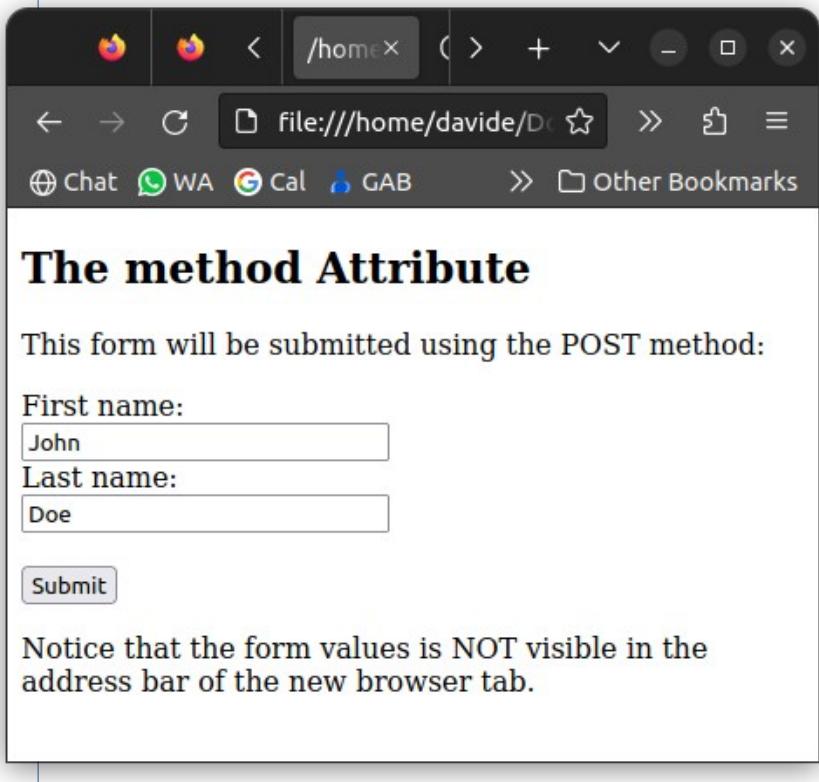
<p>This form will be submitted using the GET method:</p>

<form action="/action" target="_blank" method="post">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>

<p>After you submit, notice that the form values is visible in the
address bar of the new browser tab.</p>

</body>
</html>
```

# Passare dei dati al server web col metodo POST



The screenshot shows a Firefox browser window with two tabs open. The active tab's address bar displays "file:///home/davide/Do". The page content is titled "The method Attribute" and contains the following text:

This form will be submitted using the POST method:

First name:  
John

Last name:  
Doe

Submit

Notice that the form values is NOT visible in the address bar of the new browser tab.

# Esercizio

- Eseguire il server web `serverHTTP.c` e con il browser preferito aprire il file `form-post.html`
- Cosa si vede?
- Provare ad analizzare il contenuto della connessione TCP con Wireshark.
- Cosa si vede?

# E corrispondente richiesta HTTP

```
POST /action HTTP/1.1
```

```
User-Agent: Mozilla/5.0
```

```
Accept: text/html
```

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

```
Accept-Encoding: gzip,deflate
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

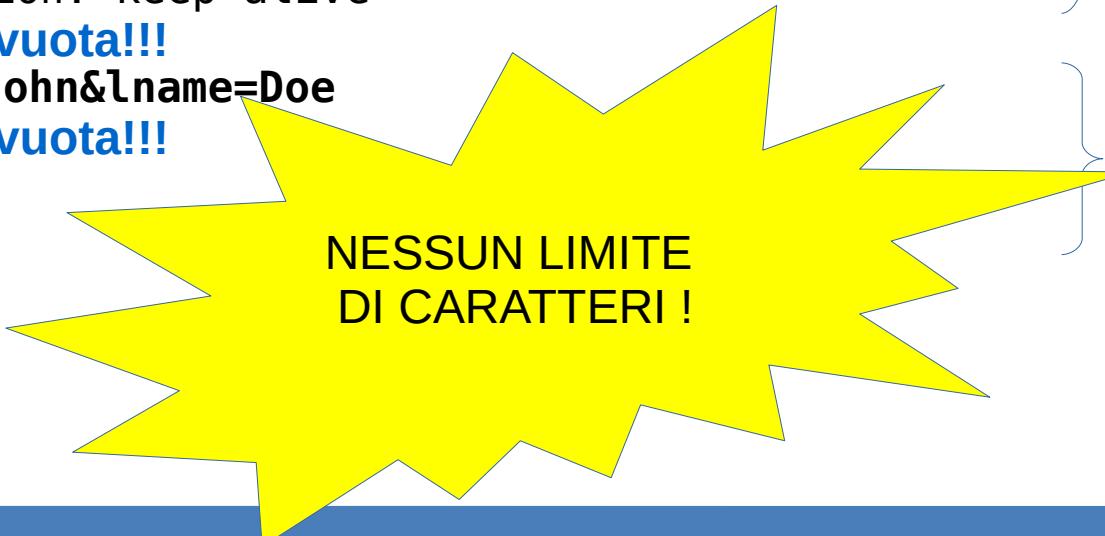
```
Keep-Alive: 300
```

```
Connection: keep-alive
```

!!!Riga vuota!!!

fname=John&lname=Doe

!!!Riga vuota!!!



NESSUN LIMITE  
DI CARATTERI !



Intestazione della richiesta



Corpo della richiesta



# Esercizio

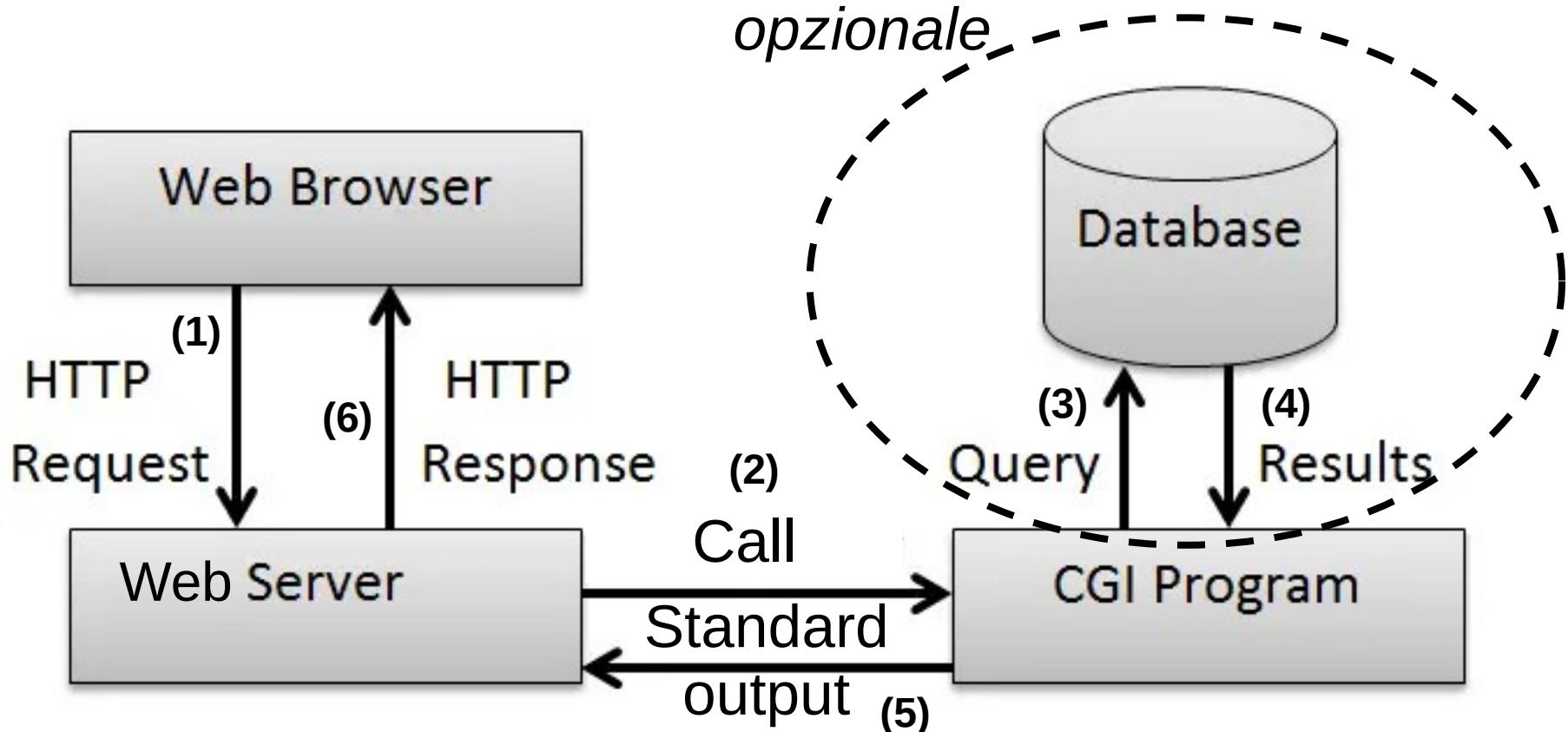
- Modificare il file `serverHTTP.c` in modo che, invece di restituire sempre la solita pagina web di prova, restituisca una delle pagine html usate dal lucido 9 in poi scelta attraverso l'uso del browser.
- Suggerimenti:
  - Provare a fare la nuova richiesta col browser usando il `serverHTTP.c` in modo da vedere la richiesta HTTP per capire dove si trova la stringa con il nome del file nella richiesta che fa il browser (aiutarsi anche con Wireshark)
    - Cosa devo scrivere nella barra del browser?
  - Capire come recuperare la stringa con il nome del file dalla richiesta HTTP (si veda `esempio-parser.c`)
  - Per costruire la risposta riciclare parte del codice usato nell'esercizio del trasferimento di file
- Prova finale: cosa succede se chiedo al server di restituire i file `form-get.html` e `form-post.html` ?



# Esercizio per casa

- Modificare il server web `serverHTTP.c` in modo che accetti con il metodo POST un intero file da salvare nella cartella del server (il cosiddetto “upload sul server”).
- Suggerimenti:
  - utilizzare il file `form-file.html` con `serverHTTP.c` non modificato ed analizzare il contenuto della connessione TCP con Wireshark in modo da capire nella richiesta HTTP
    - Dove si trova il nome del file
    - Dove si trova il contenuto del file
  - Nella lettura della richiesta HTTP sul server aggiungere il codice che salva il file prendendo spunto dal codice usato nell'esercizio del trasferimento di file
  - Invece la risposta HTTP può essere molto statica come nella versione originale di `serverHTTP.c`

# Common Gateway Interface (CGI)





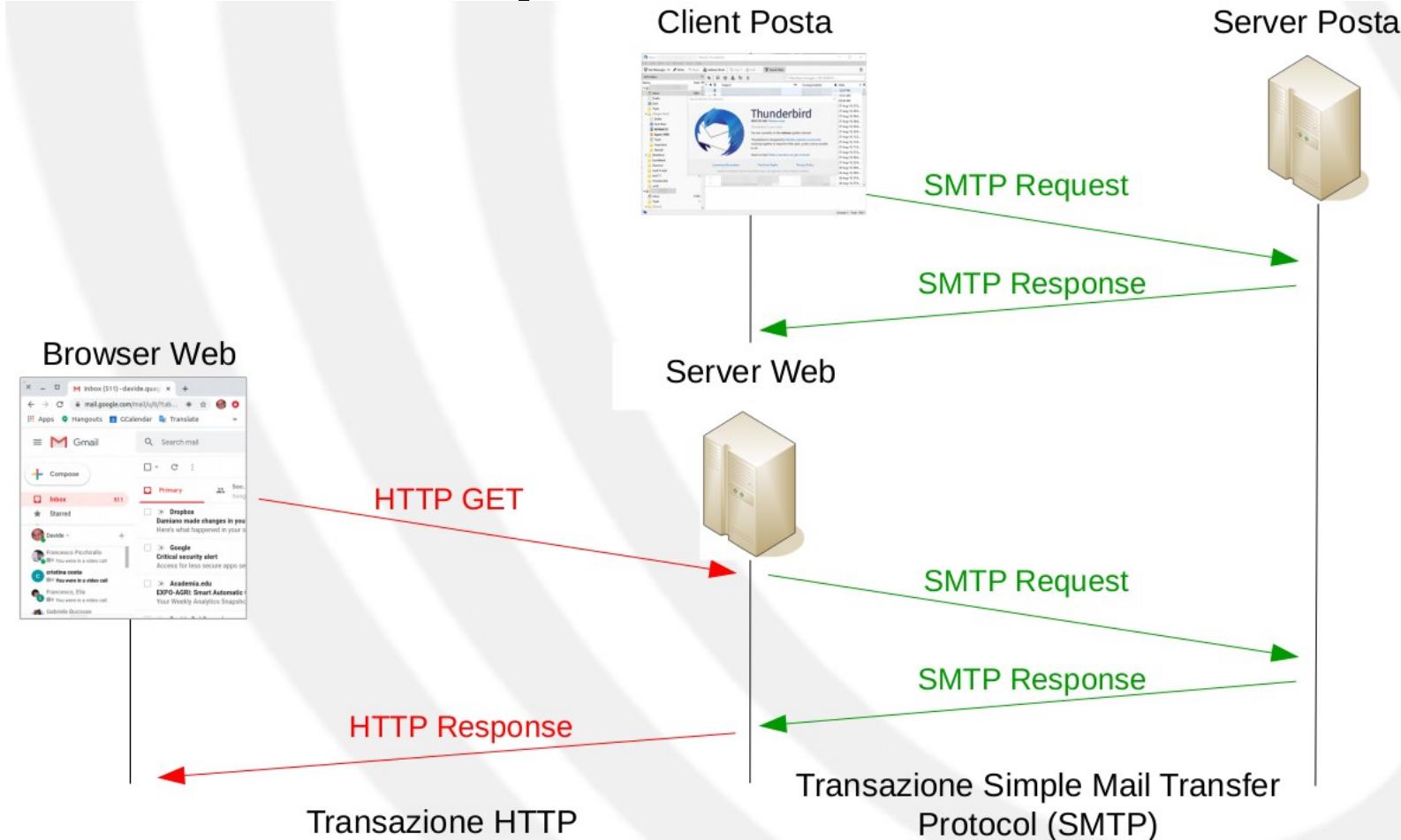
# Esempio di web server esteso con gestione CGI

- Aprire il file `serverHTTP-CGI.c` in `Esempi-web/` e analizzarne il contenuto
- Compilarlo come al solito ed eseguirlo
- Aprire il browser preferito e il file `sommatrice-web.html`
  - Cosa si vede sul browser?
  - NOTA: Provare con numeri positivi, negativi, con parte decimale...
- A cosa corrisponde il secondo parametro della funzione `sommatrice()` ?

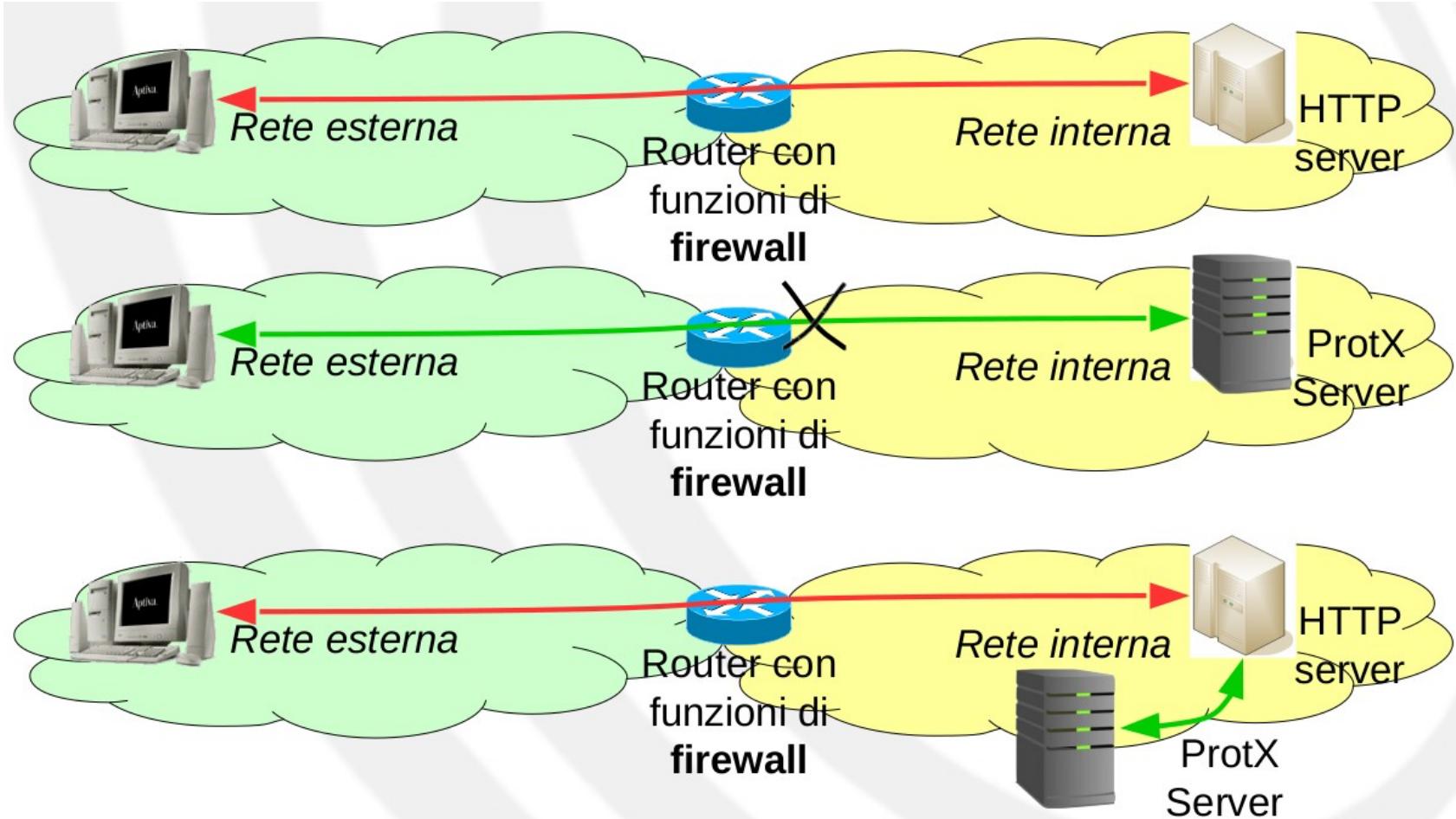
# CGI e web dinamico

- Il risultato della richiesta del browser non è più un doc statico
- Il programma CGI viene eseguito “lato server” (o “server side” o “lato back-end”)
  - Eseguibile (ad esempio client di posta elettronica)
  - PHP
  - Perl
  - Java, JSP, Servlet
  - Active Server Page (ASP)
  - NodeJS
  - Fastify
  - ... qualsiasi cosa la moda si inventerà ...
- Il browser web diventa il client di molte applicazioni di rete:
  - Posta elettronica
  - Wiki
  - Content Management Systems (CMS)

# Web e posta elettronica

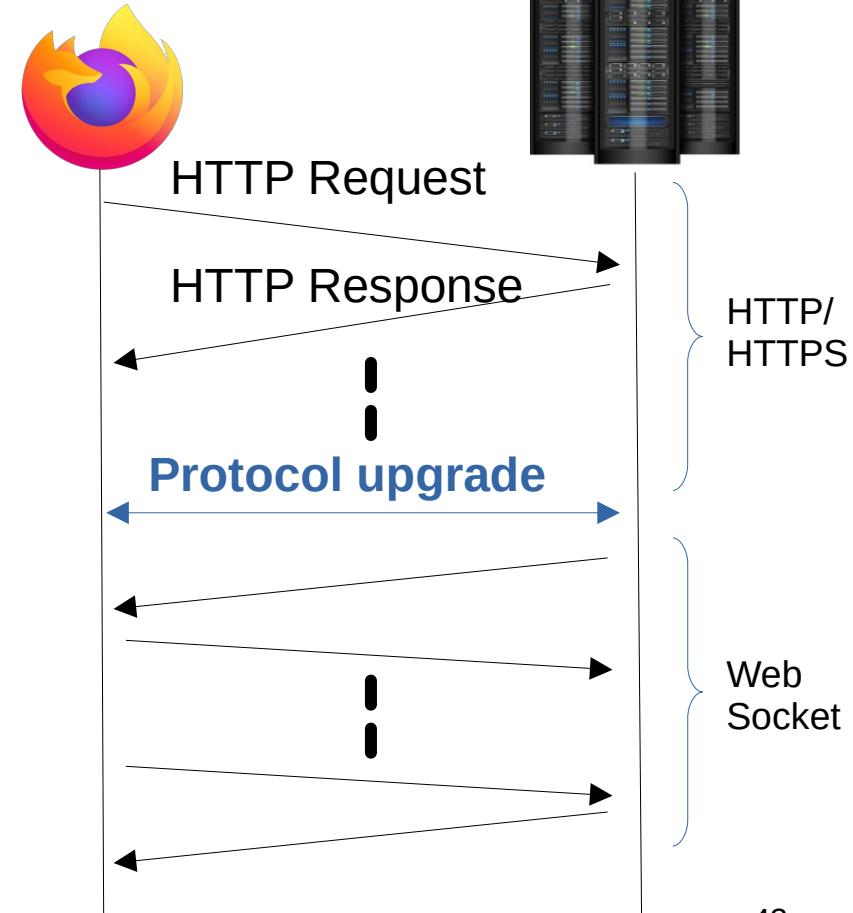


# Perché tutto passa per il web?



# Web socket

- Protocollo di livello applicazione ALTERNATIVO a HTTP/HTTPS
- Ammette la comunicazione SIMMETRICA tra il web browser e il web server
- Nasce da una sessione HTTP/HTTPS attraverso una operazione di “Protocol upgrade”
- Da quel momento in poi entrambi i processi possono “prendere l'iniziativa” per mandare dei dati dall'altra parte
  - Utilizzato per il REFRESH ASINCRONO di una pagina web attraverso il PUSH di una nuova pagina da parte del server





# Protocol upgrade

GET / HTTP/1.1

Host: server.example.com

**Upgrade: websocket**

**Connection: Upgrade**

Sec-WebSocket-Key: dGh1IHNhbXBsZSSub25jZQ==

Origin: http://example.com

Sec-WebSocket-Version: 13



HTTP/1.1 101 Switching Protocols

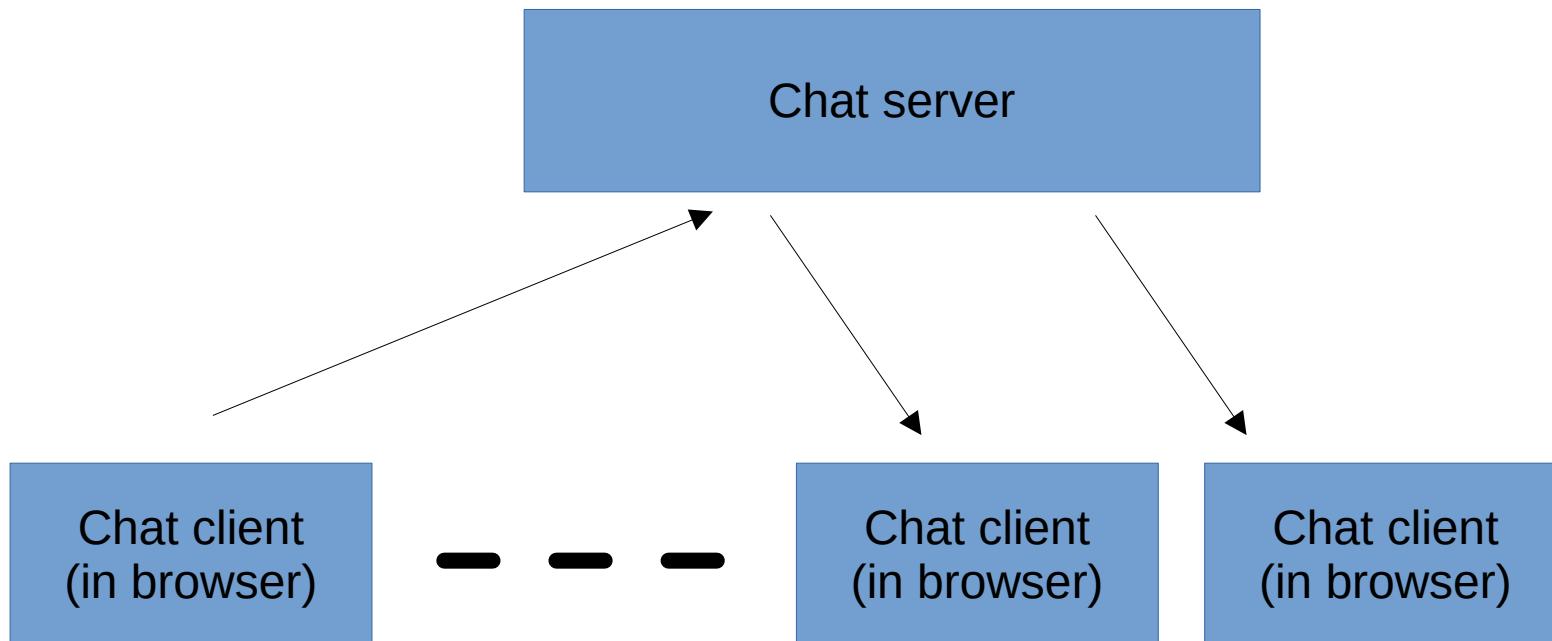
**Upgrade: websocket**

**Connection: Upgrade**

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

# Esercizio su Web Socket

- Realizzazione di una chat eseguita dentro un browser web
  - Fare riferimento al materiale apposito condiviso con gli studenti





# ARCHITETTURE ORIENTATE AI SERVIZI

e

## WEB SERVICES

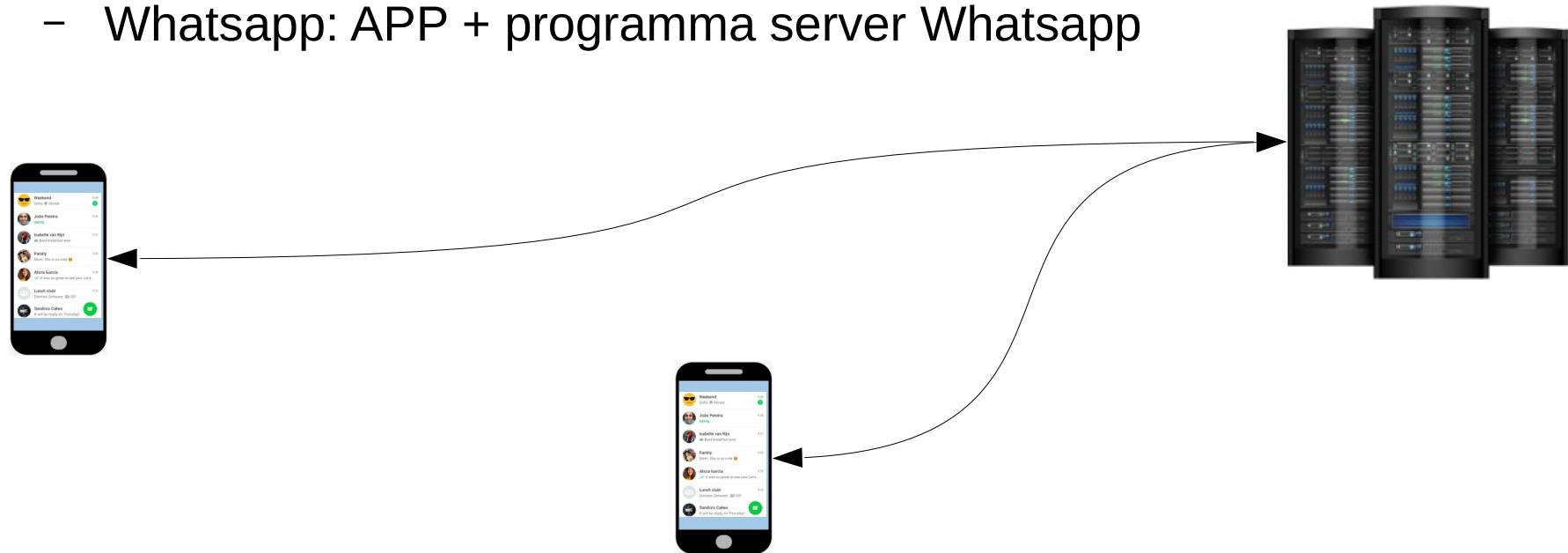
# Scrittura di applicazioni che usano la rete

- Programmi in esecuzione su host diversi che comunicano tra loro usando la rete, ad es:
  - Web: browser + programma sul sito (server) web



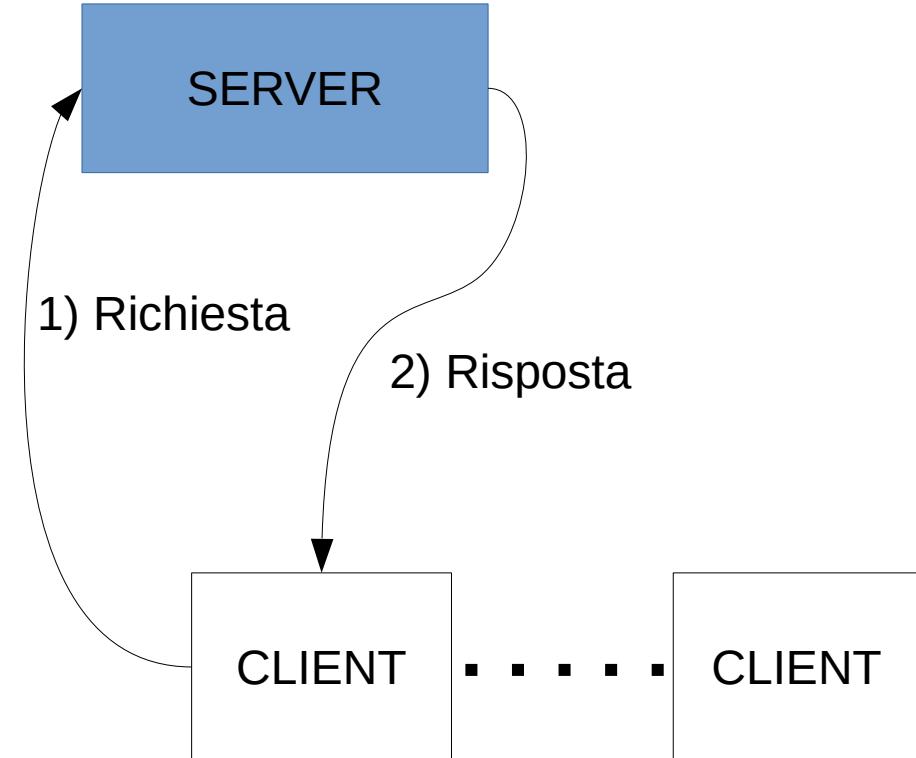
# Scrittura di applicazioni che usano la rete (cont.)

- Programmi in esecuzione su host diversi che comunicano tra loro usando la rete, ad es:
  - Whatsapp: APP + programma server Whatsapp



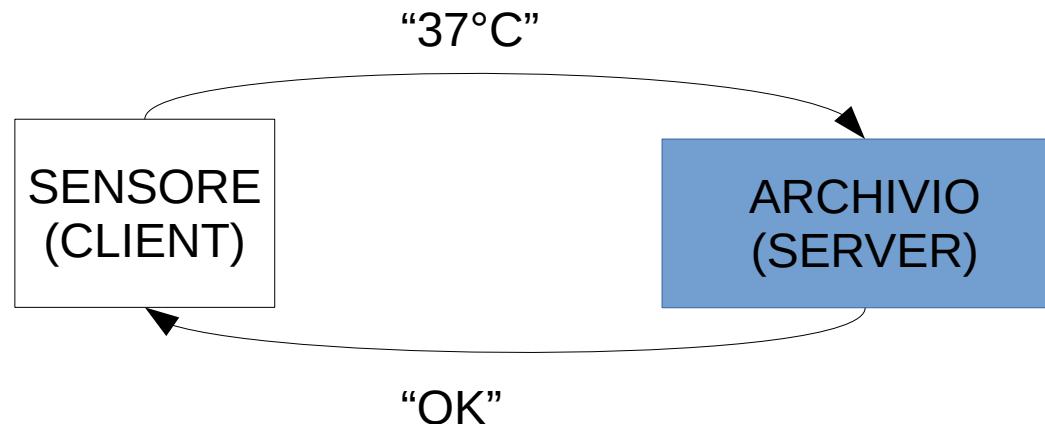
# Modello client/server

- E' la situazione più frequente
- Il client fa sempre il primo passo con una **richiesta**
- Il server manda la **risposta** e poi si rimette in attesa di altre richieste
- NOTA: La richiesta del client può essere una richiesta di un dato oppure la trasmissione di un dato (cioè la richiesta di prendere in consegna un certo dato)



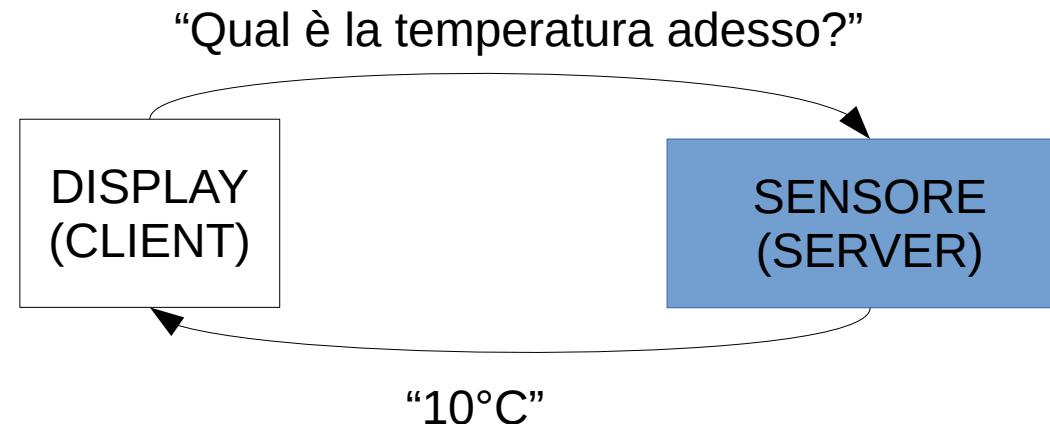
# Esempio di applicazione client/server

- Il sensore di temperatura corporea funge da client e manda al server una temperatura
- Il server risponde con un “OK”



# Altro esempio di applicazione client/server

- Il display funge da client e chiede al server una temperatura
- Il server risponde con il dato



# Architetture orientate ai servizi Service-oriented architecture (SOA)

- Tradizionali applicazioni “monolitiche”:
  - l’interfaccia utente richiama delle **funzionalità** fornite da una serie di **librerie** linkate in un unico programma che “gira” sulla macchina dell’utente
- SOA: realizzazione di applicazioni complesse attraverso la **combinazione di diversi programmi attraverso la rete**
  - L’interfaccia utente + qualche funzionalità di base “girano” sull’host dell’utente
  - Le funzionalità principali dell’applicazione sono fornite da programmi che girano su uno o più server

# SOA: vantaggi

- Potenza di calcolo e memoria sono delegate ai server
- Protezione della proprietà intellettuale su algoritmi strategici
- Annullamento della necessità di distribuire aggiornamenti del software quando le modifiche riguardano solo il codice dei server
- Nuovo modello economico: pay per use
- Eliminazione della pirateria

# SOA: requisiti

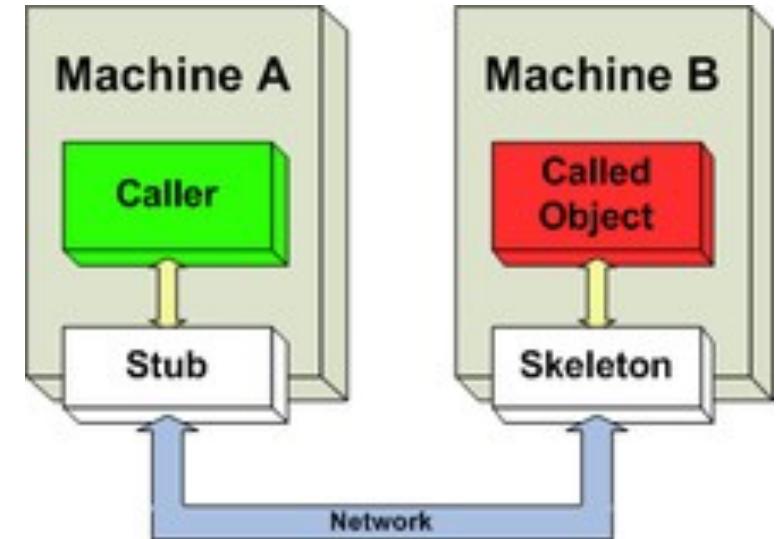
- L'infrastruttura di rete diventa un elemento essenziale
- Se “manca la rete” l'applicazione non funziona

# SOA: tecnologie

- **Servizio** = funzionalità = chiamata a funzione = metodo (per gli ambienti orientati agli oggetti)
  - È il cuore di SOA
- Componenti di una funzione:
  - Nome → ci dice cosa fa la funzione
  - Tipo e numero di parametri in ingresso
  - Tipo di valore restituito
  - Implementazione
- **Interfaccia** = descrizione di nome, parametri e valore di ritorno delle funzioni erogate da una libreria che è locale in caso di applicazioni tradizionali oppure remota in caso di SOA
- Application program interface (**API**) = insieme delle funzioni/metodi esposte da una certa libreria locale o da un server remoto
- Il cuore di SOA è la **chiamata di funzione remota** perfettamente conforme al modello client/server

# Chiamata di funzione remota

- Il componente server espone una API che descrive una serie di funzioni che il componente client (che **NON è un web browser**) può invocare
- L'implementazione delle funzioni sta nel server
- Il codice che chiama la funzione sul client e quello che implementa la funzione sul server possono essere scritti con linguaggi differenti e girare su architetture di calcolo molto differenti
  - Entrambe le funzioni implementano la **STESSA INTERFACCIA**
- La funzione chiamata dal client apparentemente realizza la funzionalità; in realtà CODIFICA i parametri per essere trasmessi in rete e DE-CODIFICA il valore di ritorno
  - Tale funzione si chiama **STUB**
- Sul server esiste un componente chiamato **SKELETON** che DE-CODIFICA i parametri di input, li passa alla funzione che contiene l'implementazione vera e propria (**BUSINESS LOGIC**), prende il risultato, lo CODIFICA e lo spedisce al client
- Ci vuole un protocollo di rete per il trasporto dei dati codificati
  - La codifica a volte si definisce **SERIALIZZAZIONE**



# Tecnologie/standard per la chiamata di funzione remota

- Remote procedure call (RPC)
  - C su TCP
- Java Remote Method Invocation (JAVA RMI)
  - Java su TCP
- Common Object Request Broker Architecture (CORBA)
  - standard indipendente dai linguaggi di programmazione e dal protocollo di liv. trasporto
- **Webservice**
  - HTTP/HTTPS come protocollo di trasporto degli elementi della funzione
  - Formato dei dati
    - Protocollo XML SOAP: pesante e datato
    - Metodologia **REST** (attualmente la più usata)

# Webservice basati su REST

- Utilizzo di HTTP/HTTPS come protocollo che veicola la chiamata
- Mapping del **nome della funzione** sulla URL
- Passaggio dei **parametri** sulla URL (metodo GET o DELETE) oppure dopo l'header (metodo POST o PUT)
- Scelta del metodo HTTP in funzione della **semantica** della funzione
  - POST: usato per funzioni che creano un NUOVO oggetto sul server
  - PUT: funzioni che aggiornano un oggetto ESISTENTE sul server
  - GET: funzioni che recuperano info di un oggetto ESISTENTE sul server
  - DELETE: funzioni che distruggono un oggetto esistente sul server
- **Valore di ritorno** nel corpo della risposta
  - Sostituzione di HTML con
    - Testo puro
    - JSON

# Metodi nella richiesta HTTP

- GET
- POST
- PUT
- DELETE
- HEAD
- CONNECT
- OPTIONS
- TRACE
- PATCH

# Webservice: vantaggi

- L'infrastruttura Internet è già predisposta all'uso di HTTP/HTTPS
  - Firewall
  - NAT
- L'uso di contenuti testuali nelle transazioni facilita il debugging delle applicazioni SOA
- Lo stesso insieme di servizi può essere richiamato sia da un programma client sia da un browser web
  - Esempio: webservice associati ad una applicazione di Internet Banking usati sia da un sito web sia dall'APP.

# JSON

- Formato di dato testuale nato con Javascript ma oggi supportato in tutti i linguaggi di programmazione
- Struttura gerarchica facilmente leggibile da un umano e parserizzabile da un programma
- Elemento base: coppia attributo : valore
- Tipi base: stringa (da indicare sempre tra " "), numero, boolean, null
- Composizione di elementi omogenei: array [...]
- Composizione di elementi eterogenei: struttura dati {...}



# Esempi di formattazione JSON

```
{"nome": "Ugo", "età": 30, "laurea": null, "abbonato": false}
```

```
{"impiegati": [
    { "nome": "Giovanni", "cognome": "Rossi" },
    { "nome": "Anna", "cognome": "Bianchi" },
    { "nome": "Pietro", "cognome": "Verdi" }
]}
```

# Esercizio

- Considerare la cartella Webservice/
- Aprire il file serverHTTP-REST.c e analizzarne il contenuto
- Compilarlo come al solito ed eseguirlo
- Aprire il file clientREST-GET.c e analizzarne il contenuto
- Compilarlo come al solito ed eseguirlo
  - Che parametri devo passare in linea di comando?
  - Cosa si può vedere analizzando lo scambio di dati tramite Wireshark?
  - Qual è la signature della funzione calcolaSomma() sul server e sul client?  
Perché ha senso che siano uguali?

# Esercizio 2

- Prendere in considerazione il file ClientREST.java
  - Dopo aver installato l'ambiente base di Java (già presente in Lab Delta) si può compilare con javac ClientREST.java
  - Ed eseguire con java ClientREST
- Il fatto che il server sia fatto in C e il client in Java è un problema o un'opportunità? Perché?

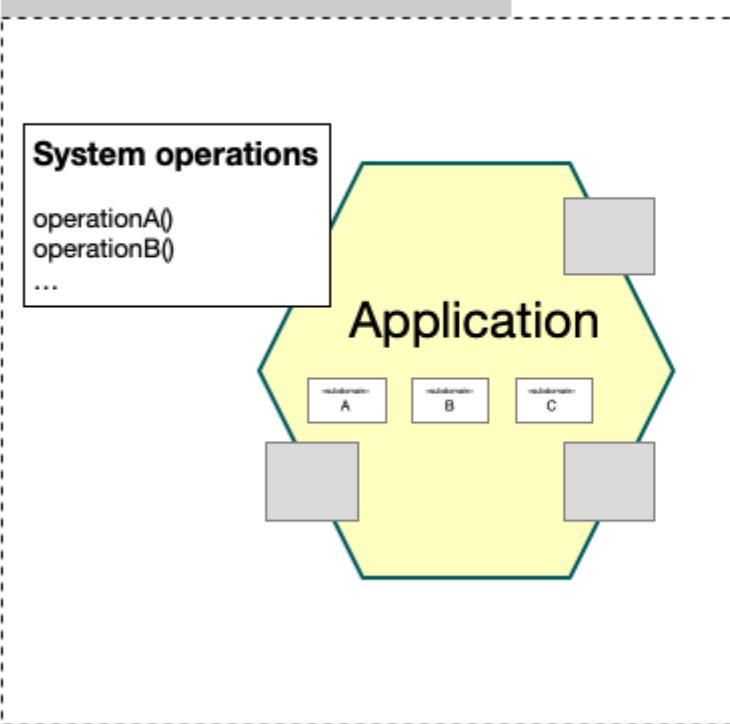


# Esercizio 3

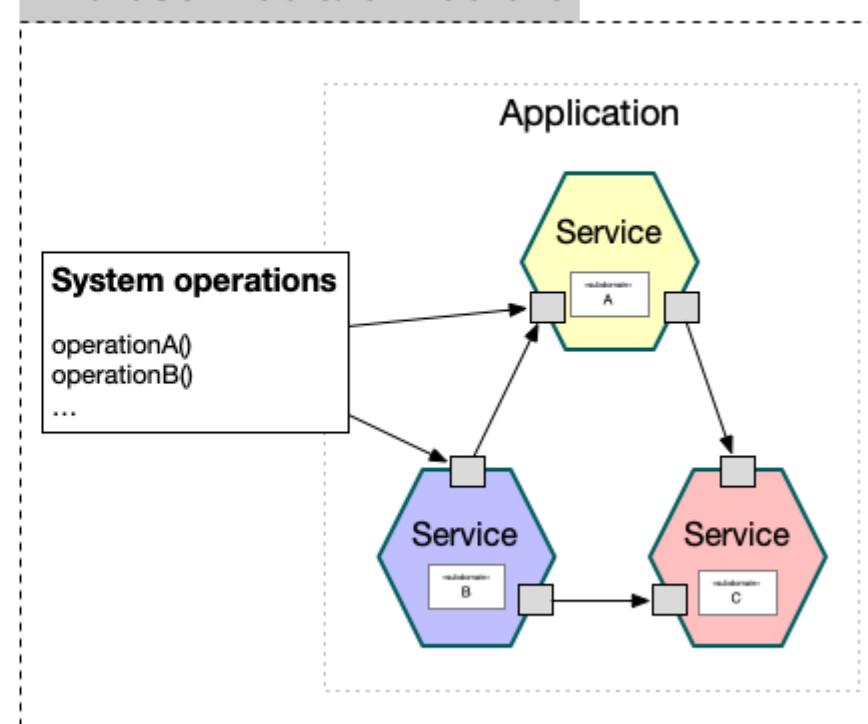
- Estendere il webservice `serverHTTP-REST.c` in modo che esponga un secondo servizio relativo al calcolo dei numeri primi compresi nell'intervallo  $[min, max]$ 
  - Si tragga spunto al programma `prime-number-interval.c`
- Estendere il file `ClientREST.java` in modo da poter chiamare, a scelta, entrambe le funzionalità della nuova API
- Provare il client con il calcolo dei numeri primi compresi nell'intervallo  $[1, 1000000]$ 
  - Quanto tempo ci mette?
  - E' stato necessario tradurre l'algoritmo dei numeri primi in Java? Perché?

# Architettura SOA a microservizi

Monolithic architecture



Microservice architecture



# Pro e contro dei microservizi

- Posso rendere modulare la parte di backend
  - assegnare moduli software a macchine differenti per aumentare la scalabilità a fronte di aumento di richieste dei client (vedere il concetto di LOAD BALANCING più avanti)
  - maggiore facilità di sviluppo e manutenzione
    - programmare interventi di manutenzione e sviluppo su certi moduli lasciando attivi gli altri
    - ridurre il rischio di effetti collaterali
- Se suddivido le tabelle della base dati su più database differenti si elimina la possibilità di JOIN ottimizzato dei DBMS relazionali e si deve implementare manualmente l'incrocio delle tabelle
  - occorre dividere la base date in punti dove l'esigenza di incrocio è minima

# Esercizio finale

- Prendere in considerazione il file `clientThreadREST.java`
  - Esso invoca `calcolaSomma()` in 3 thread concorrenti
  - Però la macchina su cui gira il server chiamato è la stessa e quindi non ci guadago in prestazioni
  - Come si potrebbe modificare il codice in modo che le 3 invocazioni finiscano su 3 macchine diverse?
  - Bisogna modificare anche il codice del server?
- Si riconsideri consideri il servizio che calcola i numeri primi nell'intervallo  $[min, max]$  costruito nell'esercizio precedente
  - Si trovi un modo efficiente, sfruttando diversi server in rete, per calcolare il numeri primi tra 1 e 1000000
  - Di quanto migliorano le prestazioni?
  - Devo modificare anche il codice del server?

# Progettazione del backend

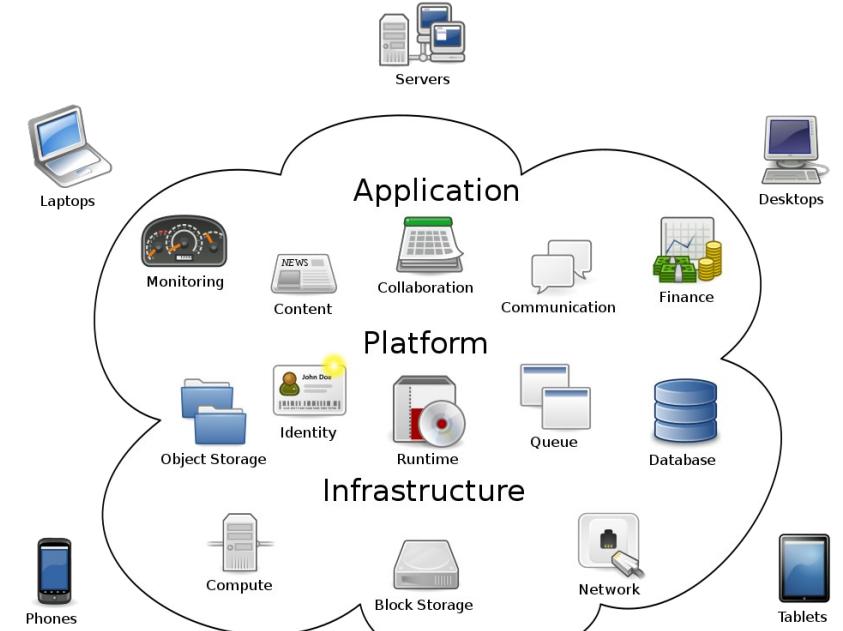
- Definire l'insieme dei servizi in cui partizionare l'applicazione
- Per ciascun servizio:
  - decidere il metodo HTTP tra GET, POST, PUT, DELETE
  - decidere la forma dell'URL con relativi parametri
  - definire il formato del JSON restituito
- Implementare i servizi con le librerie software necessarie per creare i server (Apache, NodeJS, Flask, ecc...)
- Istanziare gli applicativi server sulle varie macchine
- Ottimizzare l'esecuzione dei server sulle varie macchine

# Ottimizzazione del backend

- Posso istanziare più server dello stesso tipo e distribuire le richieste degli utenti sulle varie istanze → LOAD BALANCING
  - la containerizzazione (es. Docker) permette di replicare facilmente le istanze
  - L'orchestrazione (es. Kubernetes) permette di variare dinamicamente il numero di istanze in funzione del carico e di riavviare istanze in crash
- Diversi modi per fare load balancing
  - servizi diversi vanno su istanze diverse (non è un vero e proprio load balancing)
  - presenza di un server unico con IP corrispondente al nome di host presente nelle varie URL con il solo ruolo di smistare le richieste alle varie istanze che stanno dietro
    - può tenere conto dell'identità dell'utente e mandare richieste successive dello stesso utente sulla stessa istanza
  - il DNS risolve il nome dell'host presente nelle URL fornendo IP diversi corrispondenti alle diverse istanze
    - Problema del caching delle informazioni del DNS nella rete del client

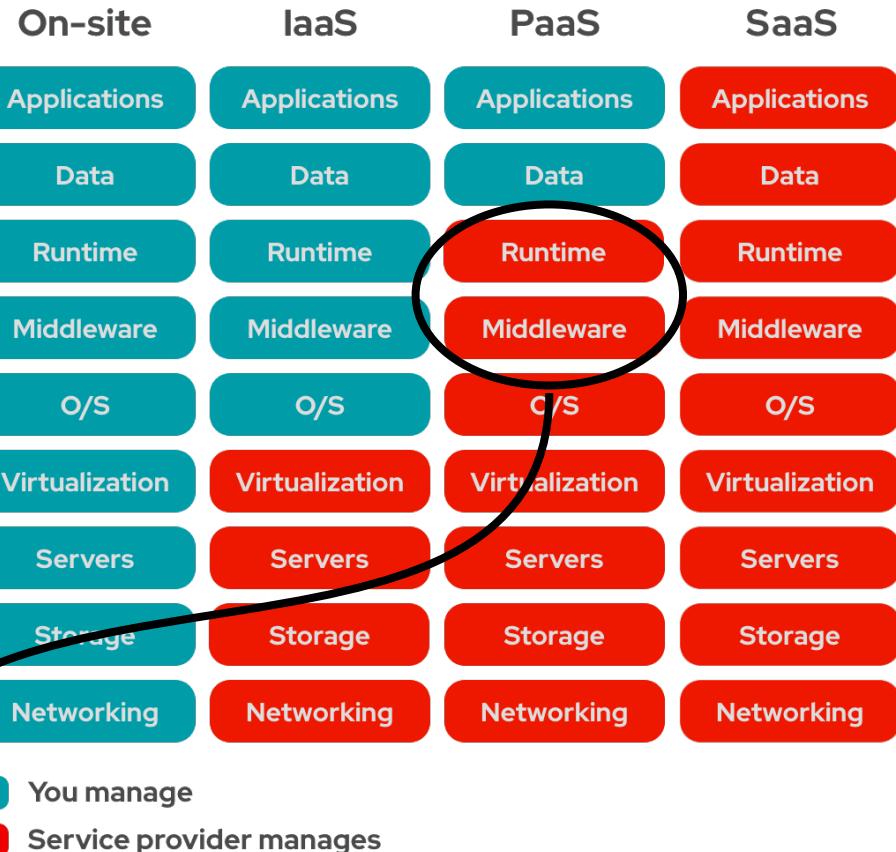
# Cloud computing

- Gruppo di host in rete che forniscono servizi di calcolo e memorizzazione senza essere indirizzati o gestiti individualmente dagli utenti
  - Amazon Web Service
  - Google Cloud
  - Microsoft Azure
  - Dropbox
- L'intero insieme di hardware e software è gestito dal fornitore del cloud che si fa pagare con una politica a consumo oppure forfettaria
- Tecnologie abilitanti per il cloud:
  - Utenti perennemente connessi in rete
  - Virtualizzazione per ottimizzare l'uso delle risorse HW
  - Webservice per sfruttare le risorse di calcolo da remoto



# Cloud: servizi offerti

- **On-site** (oppure **on-premise**): l'hardware è mio e si trova a casa mia; devo gestire tutto io.
- **Infrastructure as a Service (IaaS)**: affitto una macchina virtuale “vuota” su cui installo il mio sistema operativo e sopra tutte le mie applicazioni.
- **Platform as a Service (PaaS)**: affitto l'uso di un ambiente in cui c'è già il S.O. e posso installare direttamente le mie applicazioni (ad es. con Docker)
- **Software as a Service (SaaS)**: affitto l'uso di applicazioni già installate, ad es.
  - Web server
  - Database server
  - Broker per pub/sub



Sistema di containerizzazione (ad es. Docker) Fonte: [www.redhat.com](http://www.redhat.com)



# Function-as-a-Service (FaaS)

- Nuova offerta cloud simile a SaaS
- Possibilità di creare direttamente delle **funzioni**
  - Scegliendo il linguaggio preferito
  - Vengono associate a degli eventi
  - Vengono eseguite allocando al volo le risorse di calcolo necessarie
  - Il committente non deve creare neanche il processo server che le fornisce
- Servizi commerciali
  - AWS Lambda.
  - Google Cloud Functions.
  - IBM Cloud Functions based on Apache OpenWhisk.
  - Microsoft Azure Functions.
  - Oracle Cloud Functions.

# Function-as-a-Service (FaaS)

```
void resizeImage(Image img)  
{  
    resize(img);  
    ...  
}
```

- Granularità di suddivisione del calcolo ancora più fine rispetto all'architettura a microservizi
- anche conosciuto come “serverless computing” anche se questo è un nome poco significativo

