

Sistemi informativi su Web

# PANORAMICA DEL CORSO

aa 2024-2025



Paolo Merialdo  
Università degli Studi Roma Tre



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



# Panoramica del corso

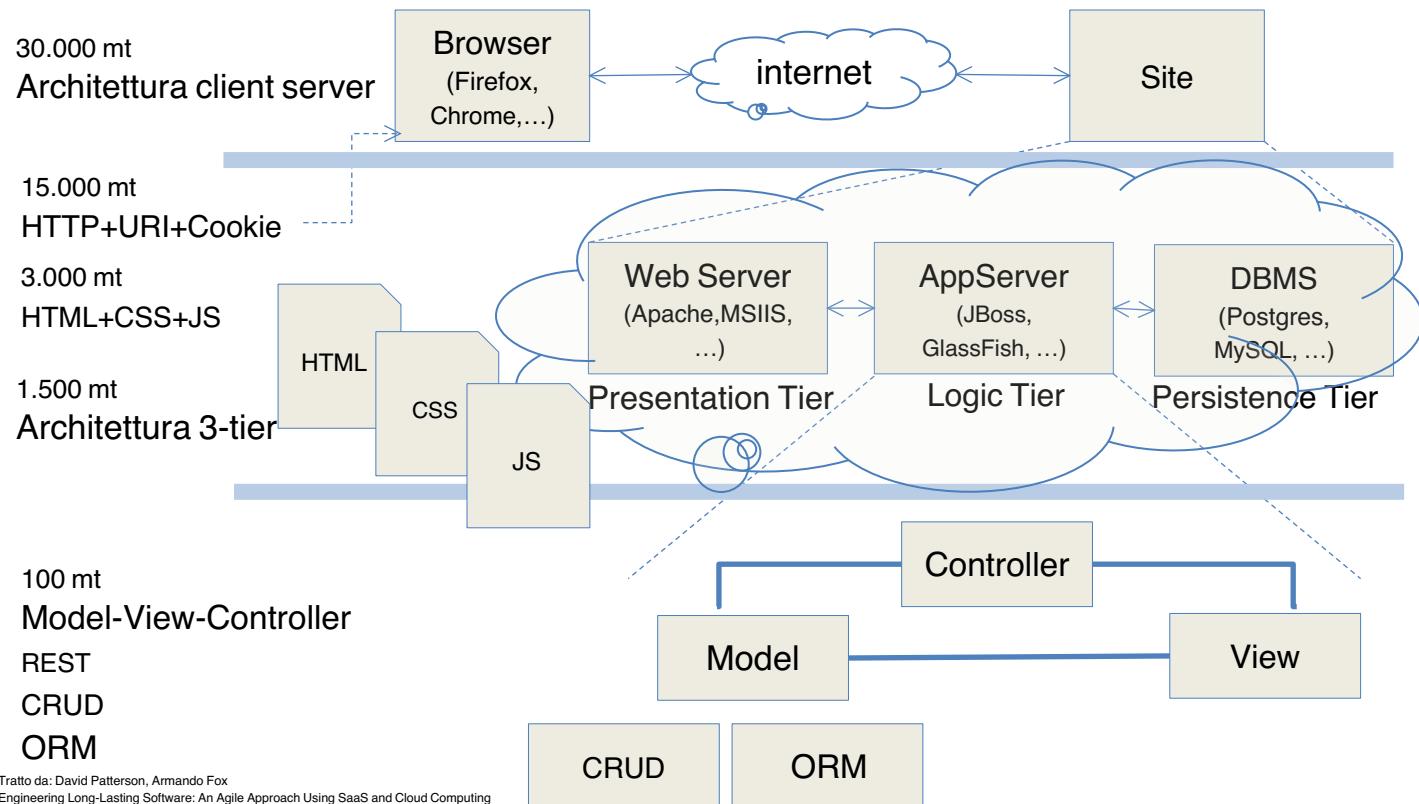
Non è un corso per vecchi

Paolo Merialdo  
[paulo.merialdo@uniroma3.it](mailto:paulo.merialdo@uniroma3.it)

Diamo un'occhiata ad una semplice applicazione:  
<http://sinai.inf.uniroma3.it:8080>



# Anatomia di un sistema informativo su Web



# Client-Server

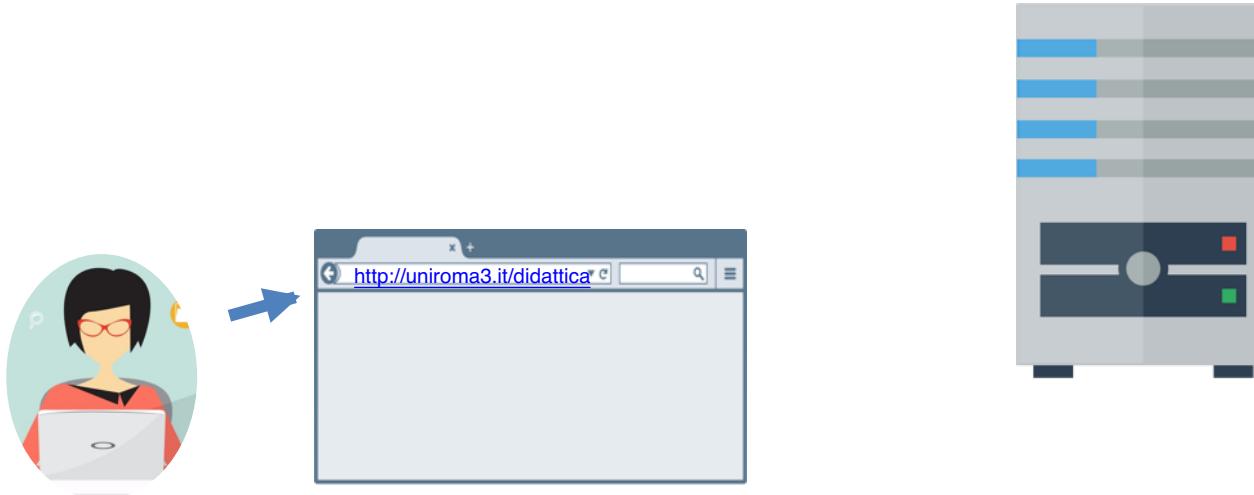
- Consideriamo il Portale dello studente
- È un esempio di sistema su architettura Client-server
- Il browser che usiamo per accedere al sistema è un esempio di un client: un programma specializzato a chiedere informazioni ad un server e a permettere all'utente di interagire con tali informazioni
- Il programma che risponde alle richieste è un esempio di un server: un programma specializzato ad attendere richieste dai client, e a fornire una risposta
- La risposta del server è generata da un'applicazione sotto il controllo (diretto o indiretto) del server

# Client-Server

- Il client fa un richiesta
- Il server invoca una applicazione per soddisfare la richiesta e manda in risposta al client l'output della applicazione
- Il client visualizza la risposta

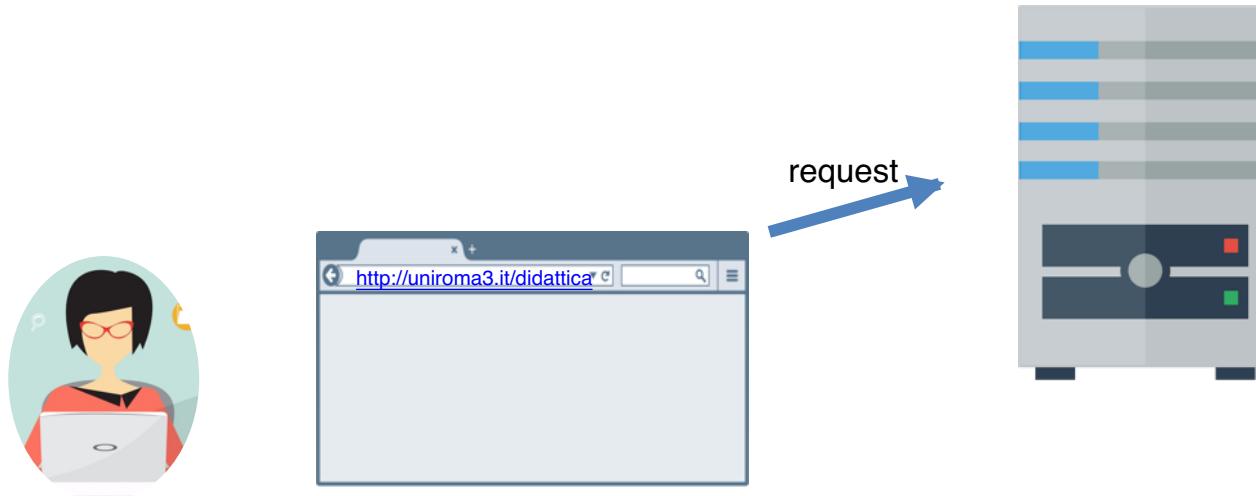
# Come funziona una richiesta Web

L'utente fa una richiesta attraverso il client (un browser)



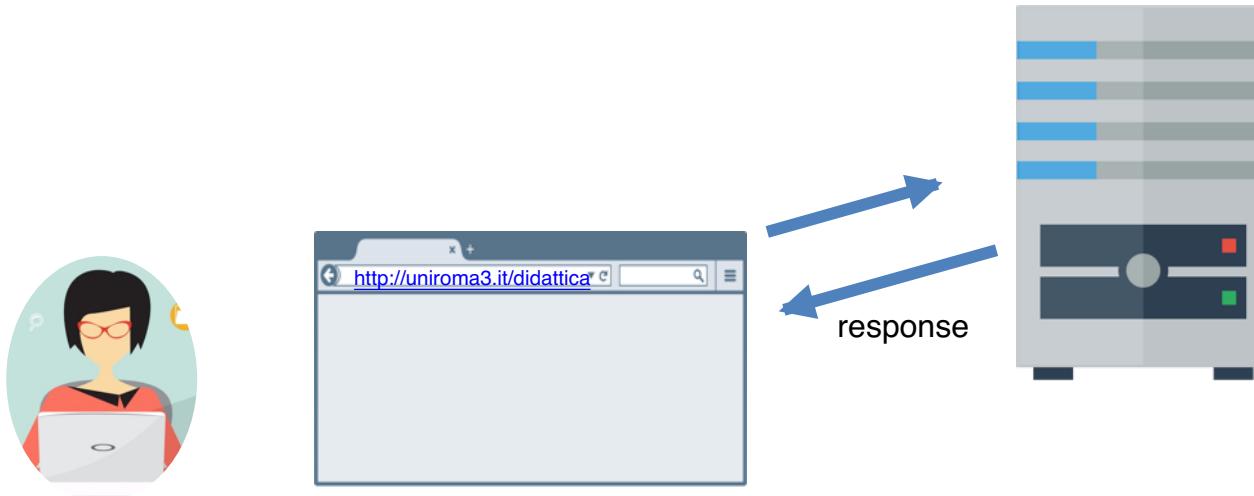
# Come funziona una richiesta Web

Il browser invia una richiesta al server



# Come funziona una richiesta Web

Il server costruisce la risposta e la invia al client



# Client-Server

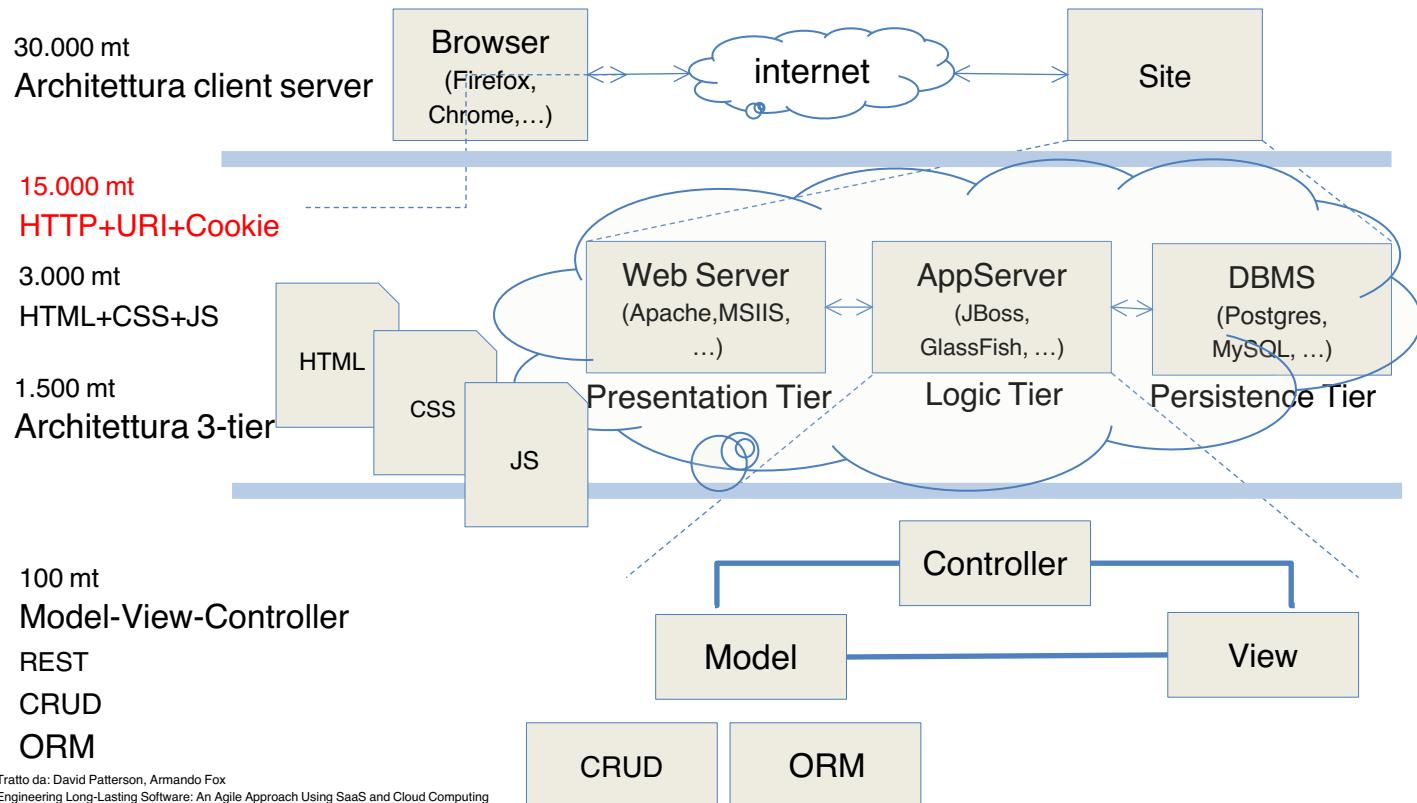
- La distinzione tra client e server permette a ciascun tipo di programma di essere specializzato nei suoi compiti
  - Client: deve realizzare un'interfaccia utente efficace
  - Server: deve rispondere in modo efficiente a molti client simultaneamente
- Esempi client
  - Firefox, Chrome, Safari, Internet Explorer
- Esempi server
  - Apache http, Apache Tomcat, MS-IIS

# Client-Server

- Prima che fossero proposti (e affermati) gli standard del Web, c'era la necessità di installare client proprietari per ogni servizio internet  
(es. Eudora per email, pgadmin per Postgresql)
- Oggi, i browser Web hanno soppiantato i client proprietari e rappresentano un "client universale"
  - NB: questa tendenza sta nuovamente cambiando con le app mobili



# Anatomia di un sistema informativo su Web



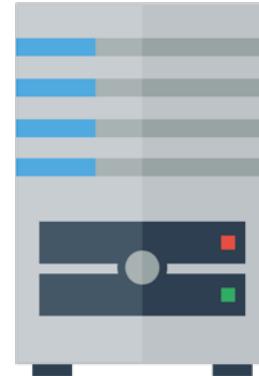
# HTTP

- Client e server comunicano attraverso un protocollo di rete (insieme di regole condivise tra gli agenti coinvolti)
- Browser e server Web comunicano con il protocollo HTTP (HyperText Transfer Protocol)
  - HTTP si basa su TCP/IP (Transmission Control Protocol/Internet Protocol) che permette a una coppia di agenti di comunicare sequenze ordinate di byte

# Come funziona una richiesta Web

L'utente fa un richiesta **http**:

- digitando un URI sulla barra del browser
- oppure, seguendo un link in una pagina

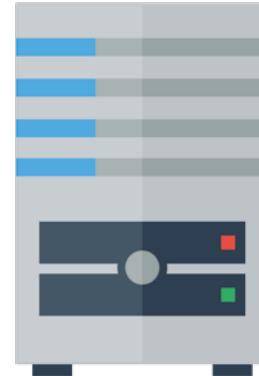
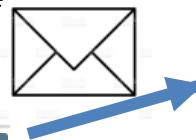


# Come funziona una richiesta Web

Il browser invia un **messaggio HTTP** al server

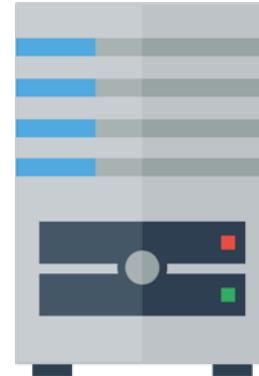
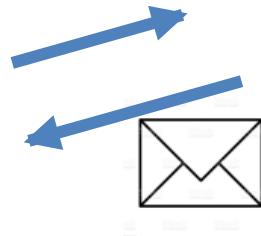
*"dear server http://uniroma3.it,*

*please GET me the didattica resource"*



# Come funziona una richiesta Web

Il server costruisce la risposta e la invia al browser  
in un messaggio HTTP



# HTTP

- In una rete TCP/IP, ogni computer (host) ha un indirizzo IP (quattro byte) ed una porta
  - Es. 193.205.219.57:8080
  - Il numero IP identifica un computer sulla rete
    - Di solito non usiamo l'indirizzo IP, ma un nome: un server specializzato (Domain Name Server, DNS) mantiene una mappa nome → numero IP. Quando il browser contatta un server per nome (es. www.uniroma3.it), automaticamente chiede ad un DNS il numero IP ad esso associato
  - La porta identifica uno specifico programma server
    - Tipicamente un server web è impostato sulla porta :80

# HTTP

Le regole di comunicazione tra client e server (richiesta client e risposta server) possono essere riassunte come segue:

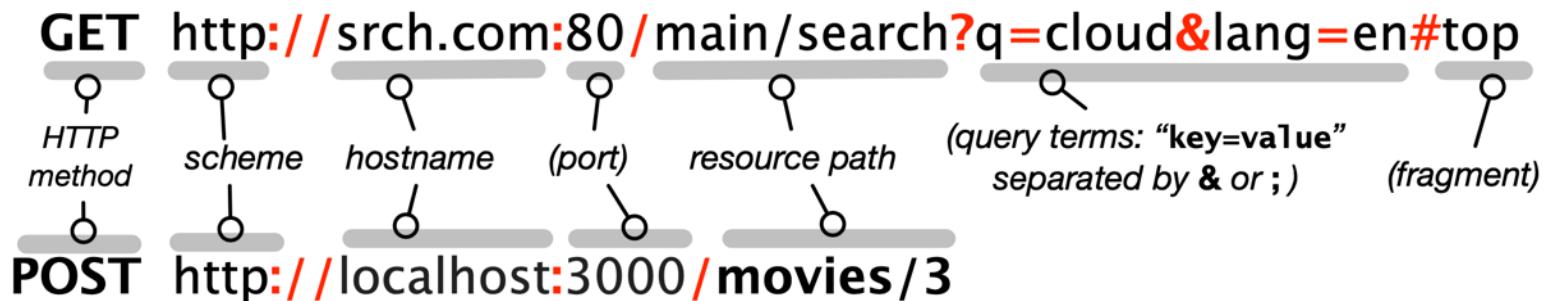
1. Il client avvia una connessione TCP/IP a un server specificando l'indirizzo IP e il numero di porta (default 80). Se il computer a quell'indirizzo IP non ha un processo server HTTP in ascolto sulla porta specificata, il client rileva un errore, che la maggior parte dei browser riporta come "Impossibile raggiungere questo sito" o "Connessione rifiutata".
2. Altrimenti, se la connessione riesce, il client invia una richiesta HTTP che descrive la sua intenzione di eseguire un'operazione su una risorsa. Una risorsa è qualsiasi entità manipolata dall'app server: una pagina Web, un'immagine e l'invio di un modulo che crea un nuovo account utente sono esempi di risorse.
3. Il server fornisce una risposta HTTP soddisfacendo la richiesta del client o segnalando eventuali errori che hanno impedito il successo della richiesta.

# Richiesta HTTP

- Una richiesta HTTP è costituita da:
  - metodo HTTP (GET, POST, PUT, PATCH o DELETE)
  - URL (Uniform Resource Locator)
  - zero o più intestazioni

\* In alcuni contesti (ad es., Ruby on Rails), metodo + URL sono chiamati "*route*"

# Esempio richiesta HTTP (non sono mostrate le intestazioni)



Tra parentesi componenti opzionali

# Uniform Resource Locator

- La stringa

`http://uniroma3.it/didattica`

è un Uniform Resource Locator (URL)

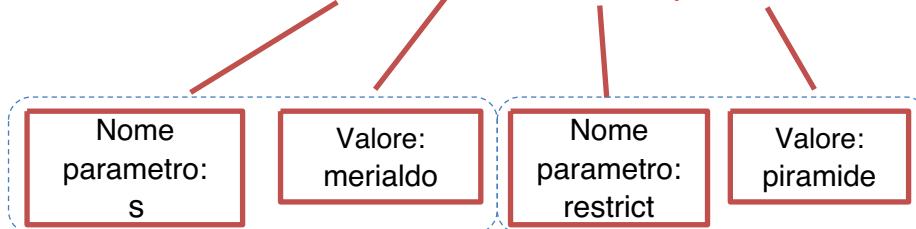
- Nome del protocollo (`http://`)
- Hostname (`www.uniroma3.it`)
- Port number, opzionale: se non specificata si usa il default associato al protocollo (:80 per HTTP)
- Risorsa richiesta all'host (`/didattica`)
  - Immagine, pagina HTML, etc.
  - Ogni applicazione ha le sue regole per interpretare il nome della risorsa

# Query String

- La risorsa può contenere una query string:
  - sequenza di coppie nome=valore
- Es: <https://www.uniroma3.it/?s=merialdo&restrict=piramide>
- Serve a trasmettere parametri al server
- Sintassi query string:

?<nome-parametro>=<valore>(&<nome-parametro>=<valore>)\*

Es. <http://www.uniroma3.it/?s=merialdo&restrict=piramide>



# Cookie

- Il protocollo HTTP è privo di stato
  - Ogni richiesta non ha memoria delle precedenti
  - Serve un meccanismo per mantenere traccia dello stato della "sessione" tra una richiesta e l'altra
- Cookies
  - meccanismo per tenere traccia dello stato della "sessione" tra una richiesta e l'altra
    - Permette di mantenere memoria nella comunicazione tra client e server
  - IETF RFC 2109 – HTTP State

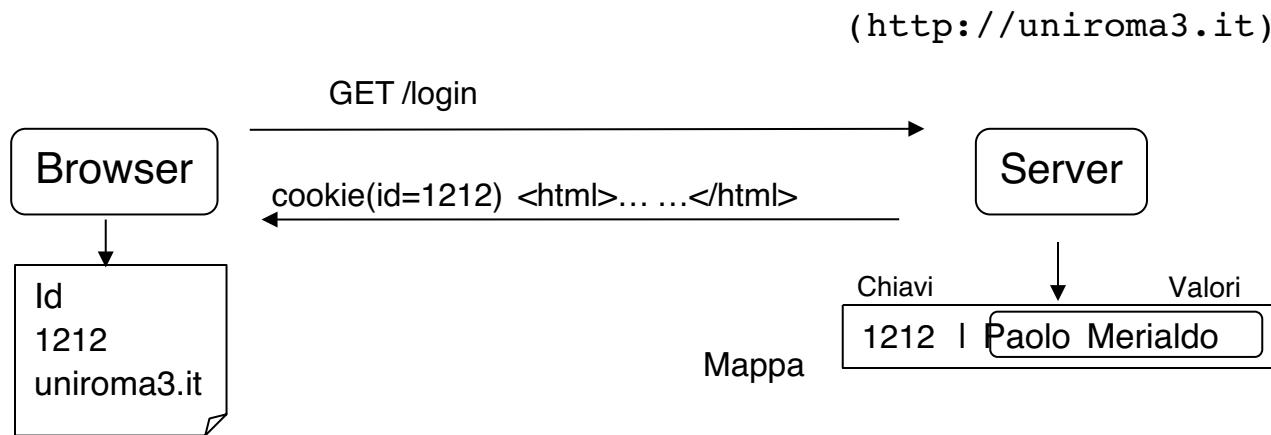
# Cookie

- Idea fondamentale
  - Utilizzare le intestazioni HTTP per "nascondere" informazioni che vengono scambiate tra client e server al fine di mantenere memoria della conversazione
- Le informazioni sono nascoste in un "cookie"
  - Nome
  - Valore
  - Validità (in secondi)
  - Dominio a cui il cookie si applica

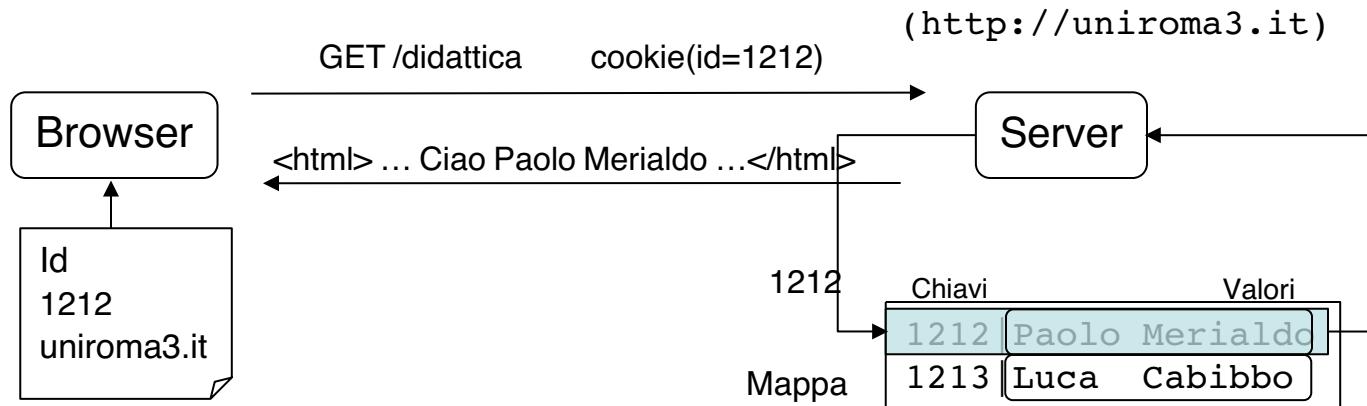
# Cookies: funzionamento

- 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

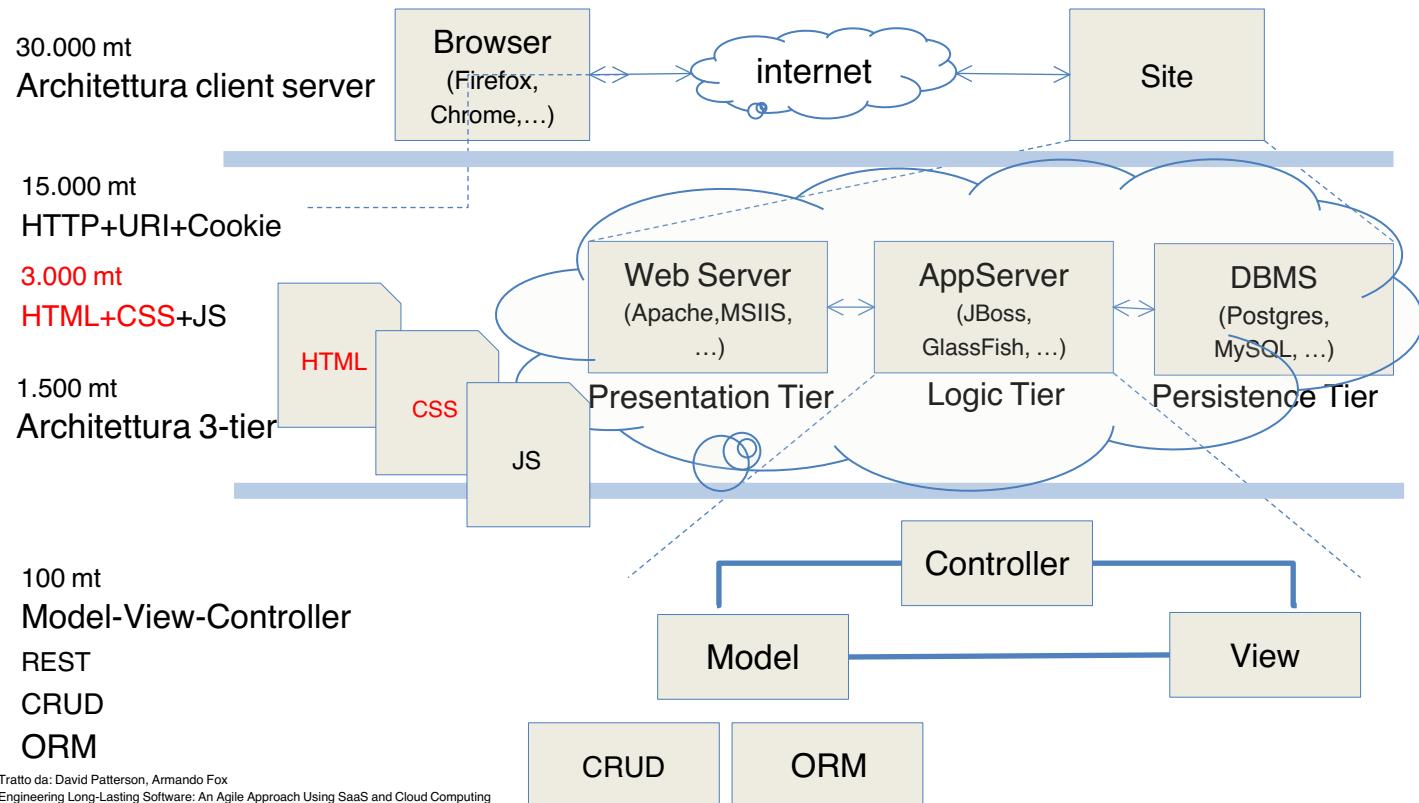
- Il trucco consiste nell'uso di un'informazione "nascosta" scambiata tra server e client
  - il client si impegna a ritornare questa informazione al server ad ogni richiesta
  - il server usa questa informazione per ricostruire ai propri fini la storia delle interazioni con il client
- Una soluzione alternativa non usa i cookie (aggiunge un token all'URL): il principio è lo stesso

# 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



# Anatomia di un sistema informativo su Web



# HTML+CSS

- L'interazione con l'utente avviene attraverso pagine Web
- Le tecnologie base per la codifica di pagine Web sono:  
HTML + CSS
  - HTML: definisce il contenuto della pagina
  - CSS: definisce la presentazione del contenuto della pagina

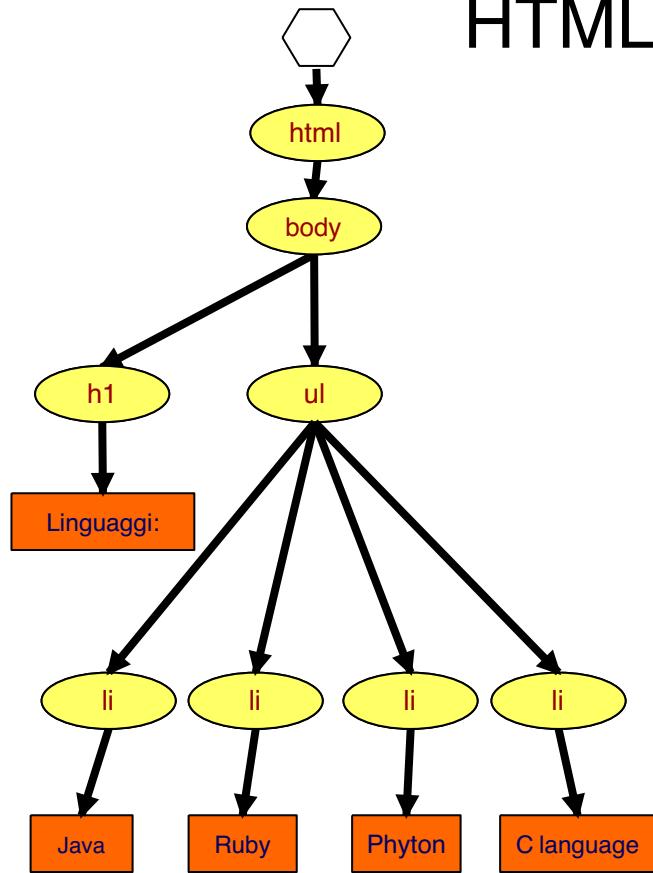
# HTML+CSS

- HTML: linguaggio di marcatura (markup) per la definizione di pagine ipertestuali
  - struttura gerarchica (albero)
- CSS: linguaggio per associare direttive di presentazione agli elementi della struttura gerarchica di un documento HTML

# HTML+CSS

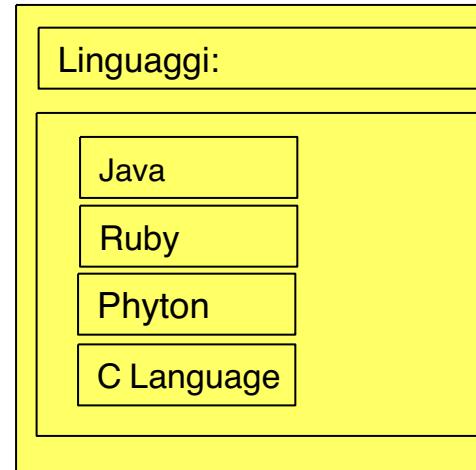
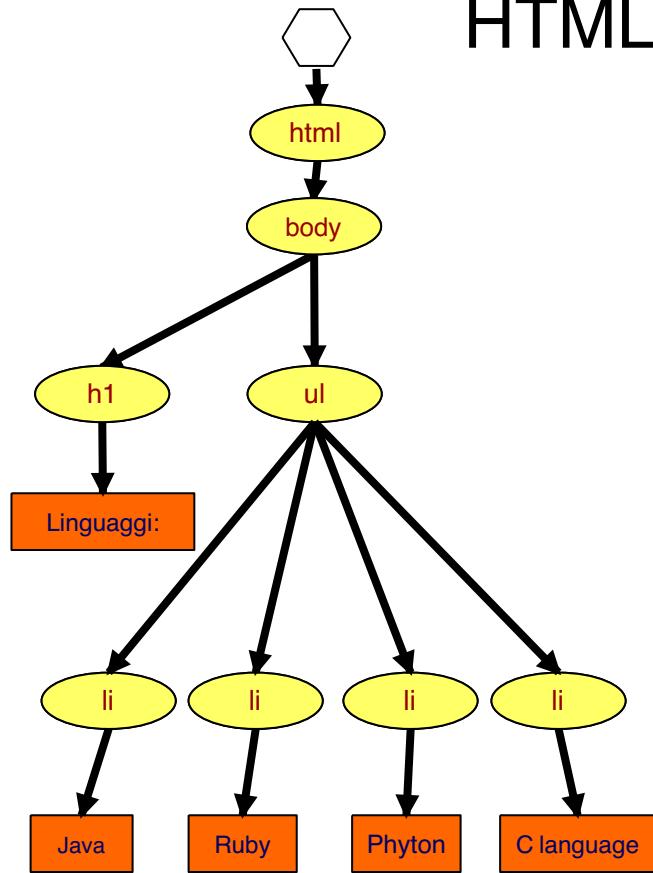
- Una struttura gerarchica ha diverse forme di rappresentazione equivalente:
  - nidificazione di nodi in un albero
  - nidificazione di elementi in un documento
  - nidificazione di riquadri

# HTML+CSS



```
<!DOCTYPE HTML>
<html>
  <body>
    <h1>Linguaggi:</h1>
    <ul>
      <li>Java</li>
      <li>Ruby</li>
      <li>Phyton</li>
      <li>C language</li>
    </ul>
  </body>
</html>
```

# HTML+CSS



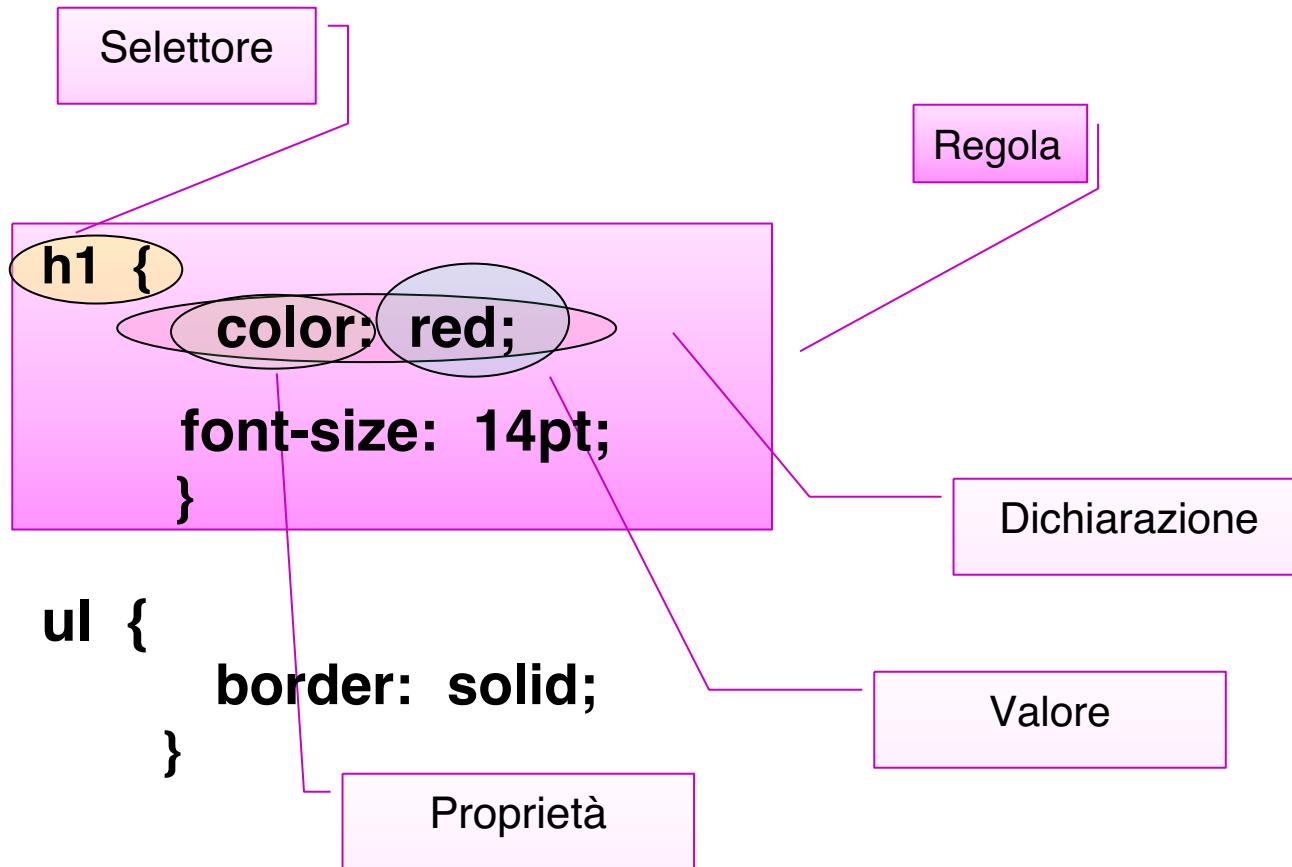
# HTML+CSS

- CSS permette di definire **regole** di presentazione per ogni riquadro di un documento HTML
- CSS offre sintassi e semantica per specificare:
  - come **selezionare** i vari riquadri
  - come **dichiarare** le proprietà di presentazione da associare ad ogni riquadro
- Per rendere efficace il meccanismo
  - Alcune proprietà (ad esempio il font) devono poter essere *ereditate* dal riquadro esterno (padre)

# HTML+CSS

- Foglio di stile CSS: lista di regole
- Regola  
**selettore { lista\_di\_dichiarazioni }**
- Dichiarazione  
**proprietà: valore;**

# HTML+CSS



# HTML+CSS

- CSS è uno standard che definisce:
  - sintassi e semantica per i selettori (con riferimento all'albero associato ad un documento)
  - la sintassi per le regole
  - l'insieme di proprietà e l'insieme dei valori che queste possono assumere

# Come funziona una richiesta Web

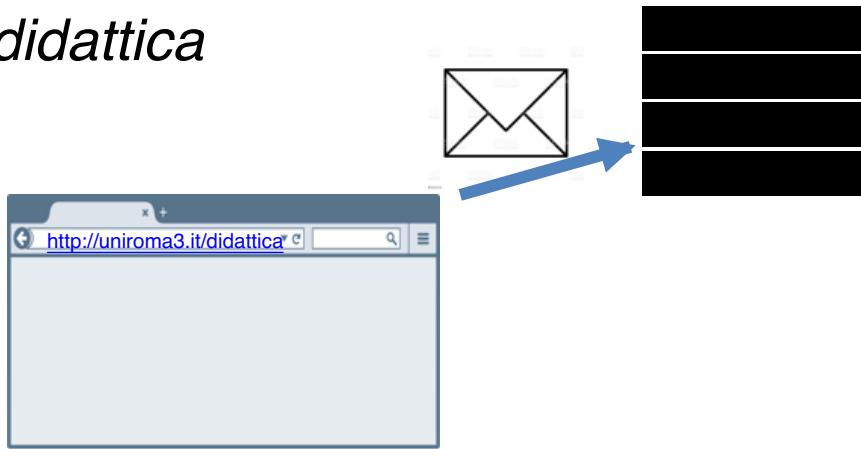
L'utente fa un richiesta http:

- digitando un URL sulla barra del browser
- seguendo un link in una pagina



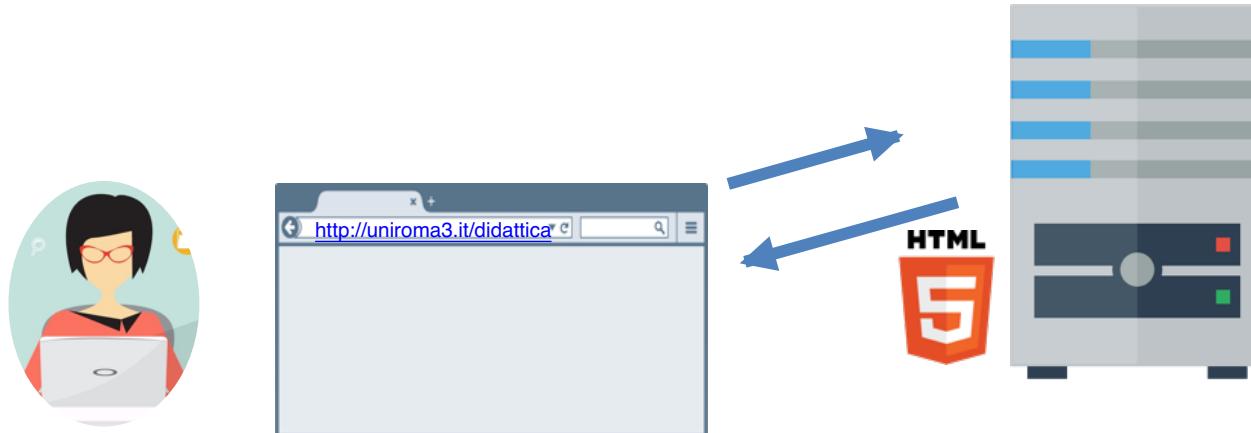
# Come funziona una richiesta Web

Il browser invia un messaggio HTTP al server "dear server  
<http://uniroma3.it>,  
*please GET me the didattica  
resource*"



# Come funziona una richiesta Web

Il server recupera il file HTML (dal file system o da un'applicazione) e lo invia al browser in un messaggio HTTP

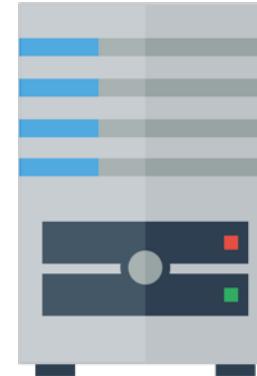


# Come funziona una richiesta Web

Il file HTML può contenere elementi

(quali ad es. <link href="style.css" rel="stylesheet"/> e )

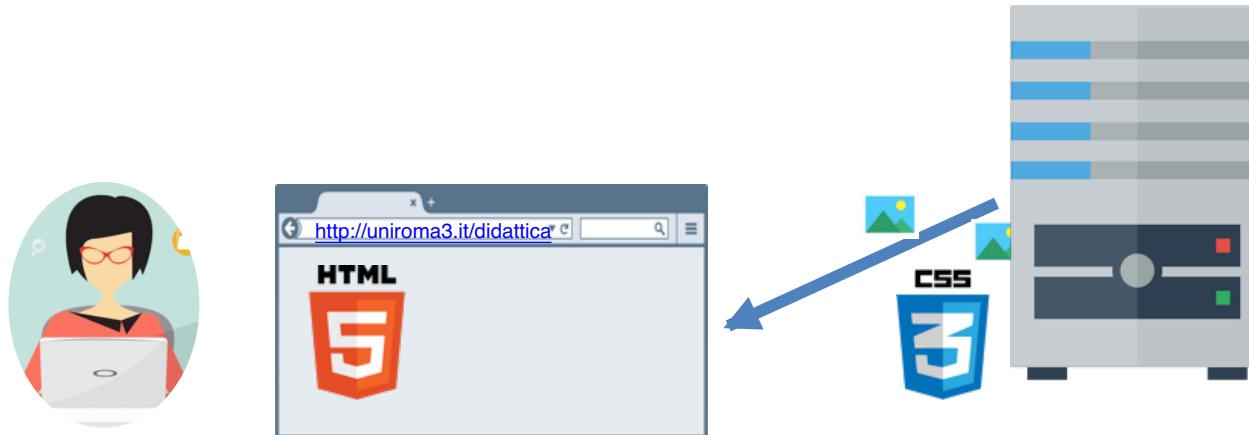
ciascuno dei quali corrisponde ad  
una nuova richiesta al server\*



\* eventualmente a server diversi

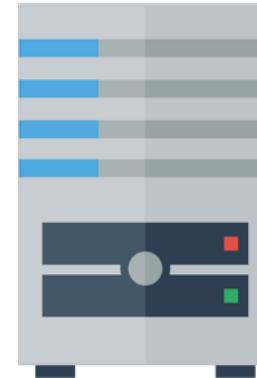
# Come funziona una richiesta Web

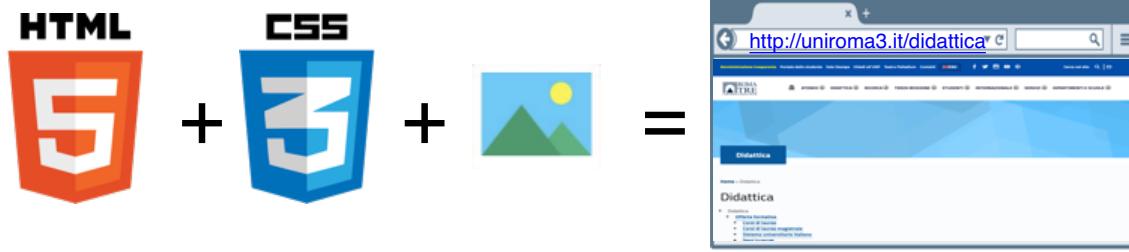
Il server risponde mandando al client le risorse richieste dal browser



# Come funziona una richiesta Web

Quando il browser ha tutte le risorse,  
mostra all'utente la pagina richiesta





Una pagina Web

# Esempio

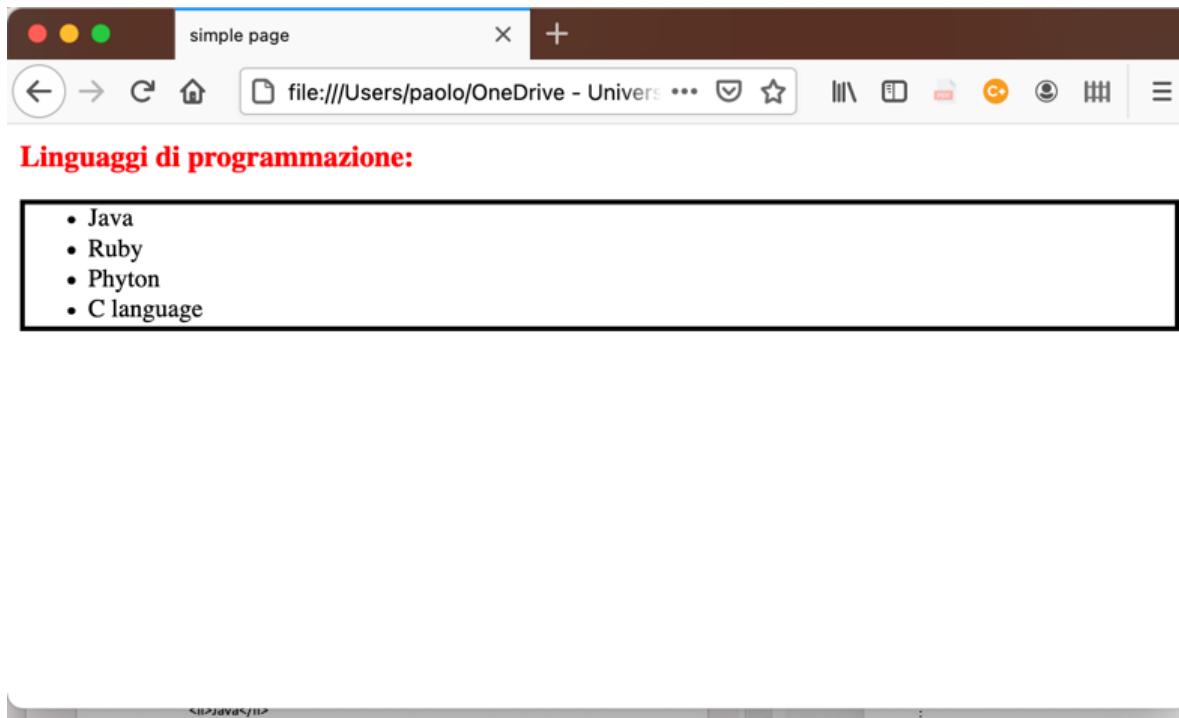
simple-page.html

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>simple page</title>
    <link rel="stylesheet"
          href="simple-style.css"/>
  </head>
  <body>
    <h1>Linguaggi di programmazione:</h1>
    <ul>
      <li>Java</li>
      <li>Ruby</li>
      <li>Phyton</li>
      <li>C language</li>
    </ul>
  </body>
</html>
```

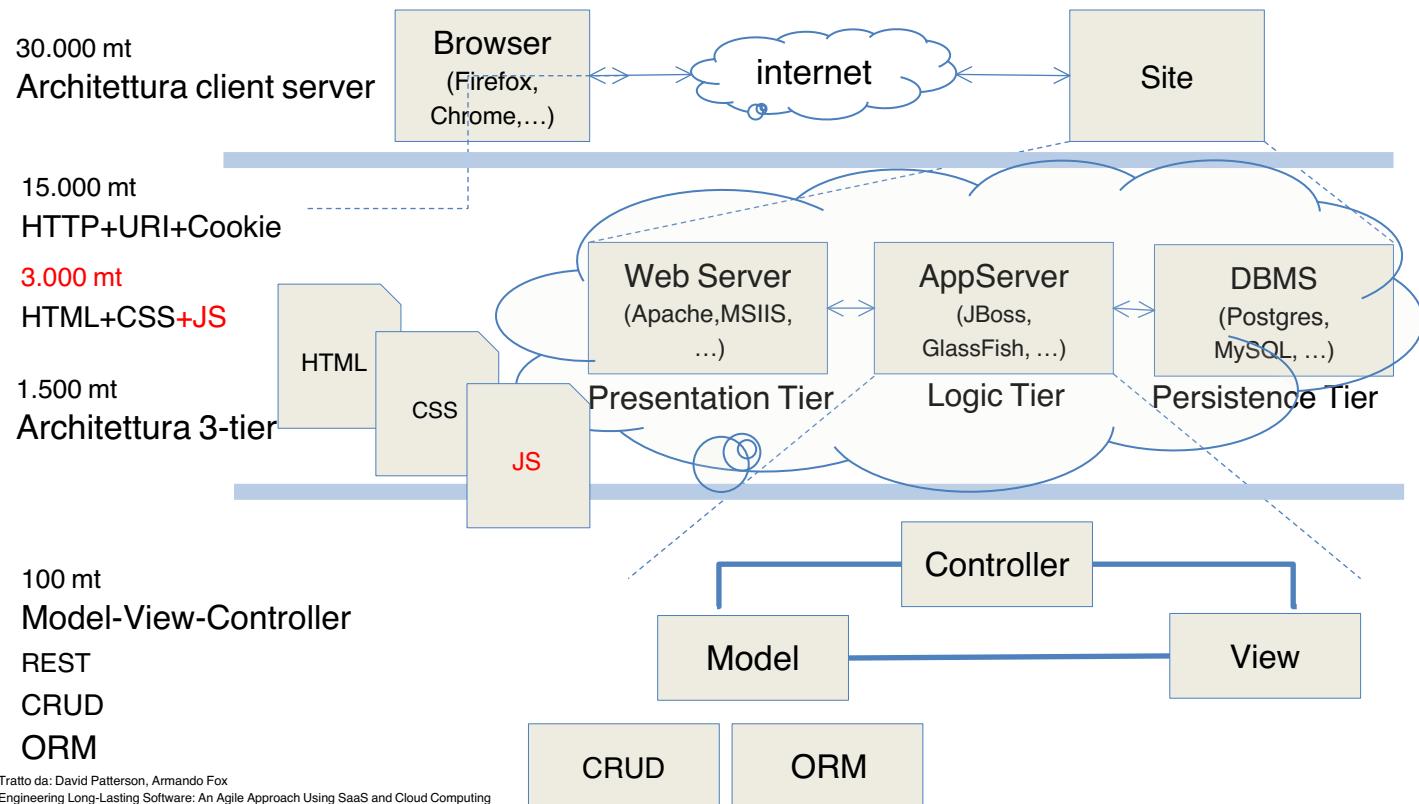
simple-style.css

```
h1 {
  color: red;
  font-size: 14pt;
}
ul {
  border: solid;
```

# Esempio



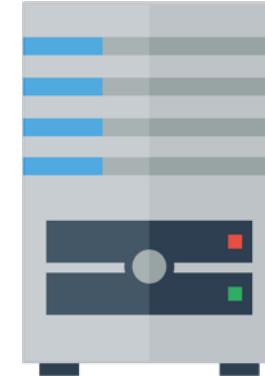
# Anatomia di un sistema informativo su Web



# JavaScript

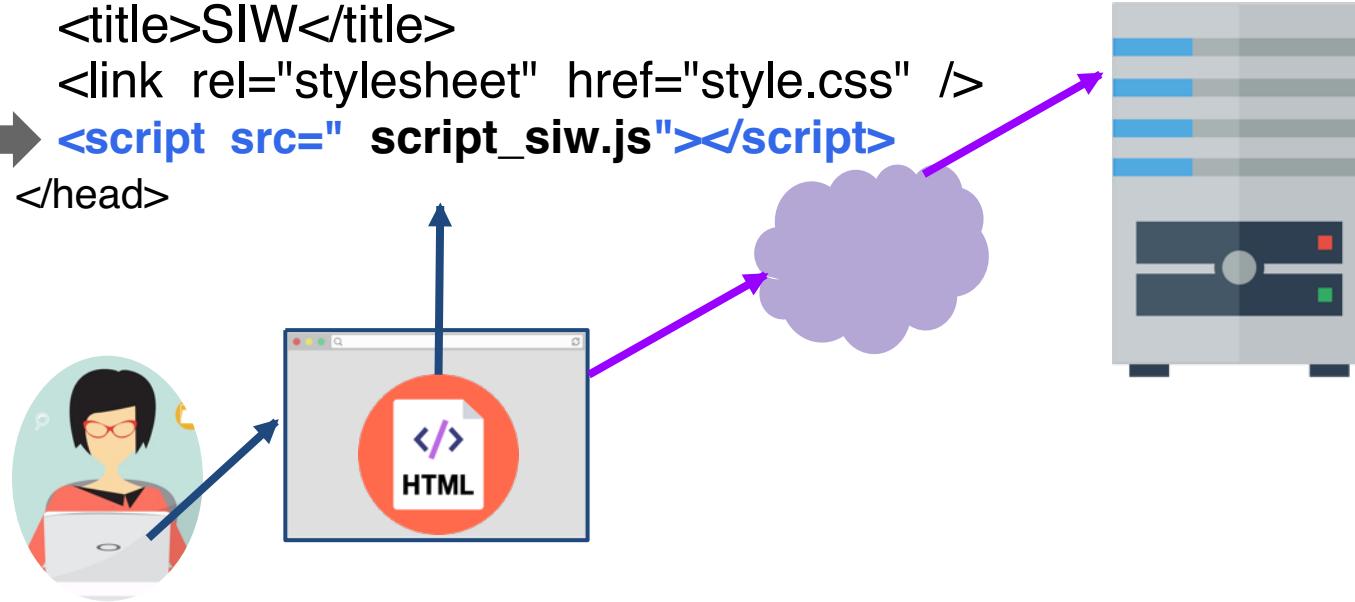
- JavaScript è un linguaggio di programmazione
- E' l'unico linguaggio di programmazione che il browser può eseguire nativamente. (Ci sono progetti per cambiare)

```
<head>
  <title>SIW</title>
  <link rel="stylesheet" href="style.css" />
  ➔ <script src="script_siw.js"></script>
</head>
```



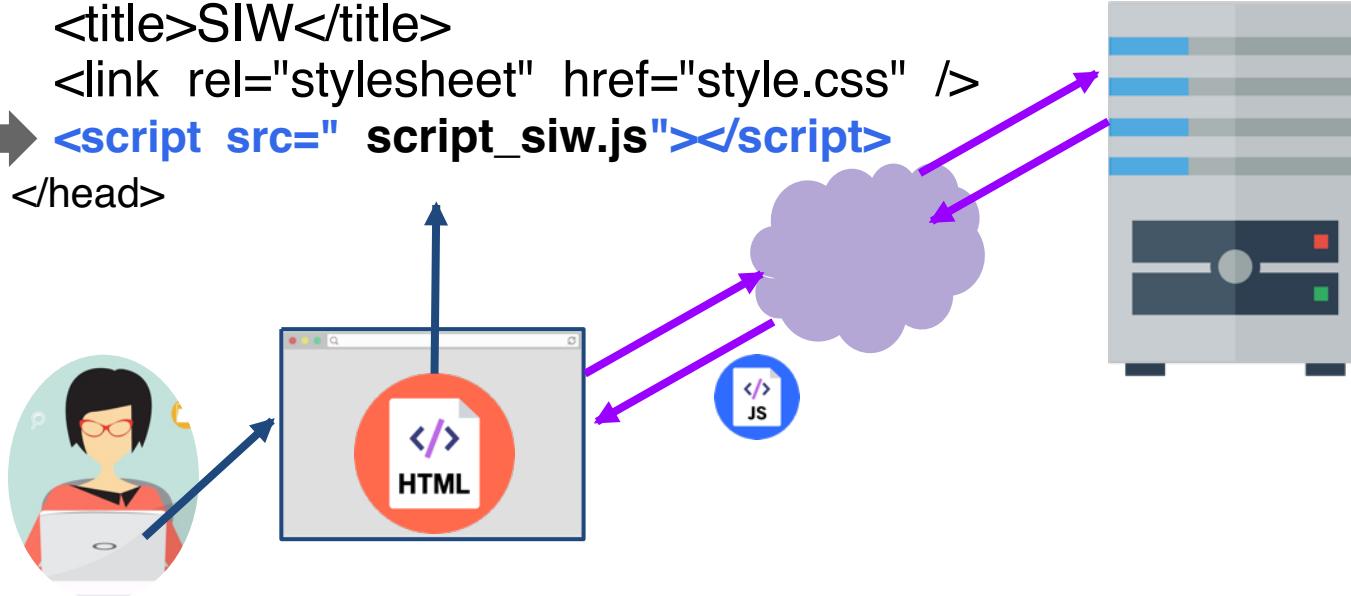
Il browser legge il file HTML e trova un tag script: sa che deve richiedere anche il file con lo script .

```
<head>
  <title>SIW</title>
  <link rel="stylesheet" href="style.css" />
  ➔ <script src=" script_siw.js "></script>
</head>
```



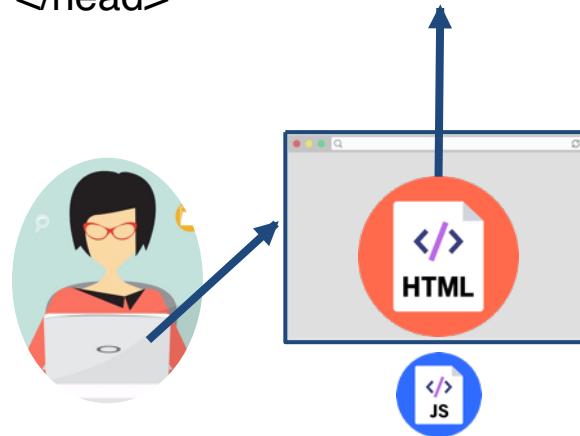
Il browser fa una richiesta al server per ottenere il file script\_siw.js file, esattamente come per qualsiasi altro tipo di file (ad es., CSS o un'immagine)

```
<head>
  <title>SIW</title>
  <link rel="stylesheet" href="style.css" />
  ➔ <script src=" script_siw.js "></script>
</head>
```



Il server risponde con il file JavaScript

```
<head>
  <title>SIW</title>
  <link rel="stylesheet" href="style.css" />
  ➔ <script src=" script_siw.js "></script>
</head>
```



A questo punto, il file JavaScript sarà eseguito (interpretato)  
"client-side", nel browser dell'utente

# Diversi modi di usare Javascript

Distinguiamo quattro approcci, che elenchiamo in ordine dal meno al più "intensivo":

1. Aggiunta di JavaScript a HTML e CSS per migliorare l'esperienza utente.
2. Creazione di applicazioni "single-page" (SPA) in cui l'esperienza dell'utente prevede che, una volta caricata la pagina iniziale, non si verifichino ulteriori caricamenti della pagina, sebbene gli elementi della pagina vengano aggiornati continuamente in risposta alla comunicazione con il server.
3. Applicazioni lato client complete (come GoogleDocs), paragonabili per complessità alle app desktop ed eventualmente in grado di funzionare anche quando non si è connessi a Internet.
4. Creazione di app lato server complete, simili a quelle che creiamo in Java, ma utilizzando un framework JavaScript come Node.js

# JavaScript

- Viene immerso nel codice HTML con un marcatore specifico

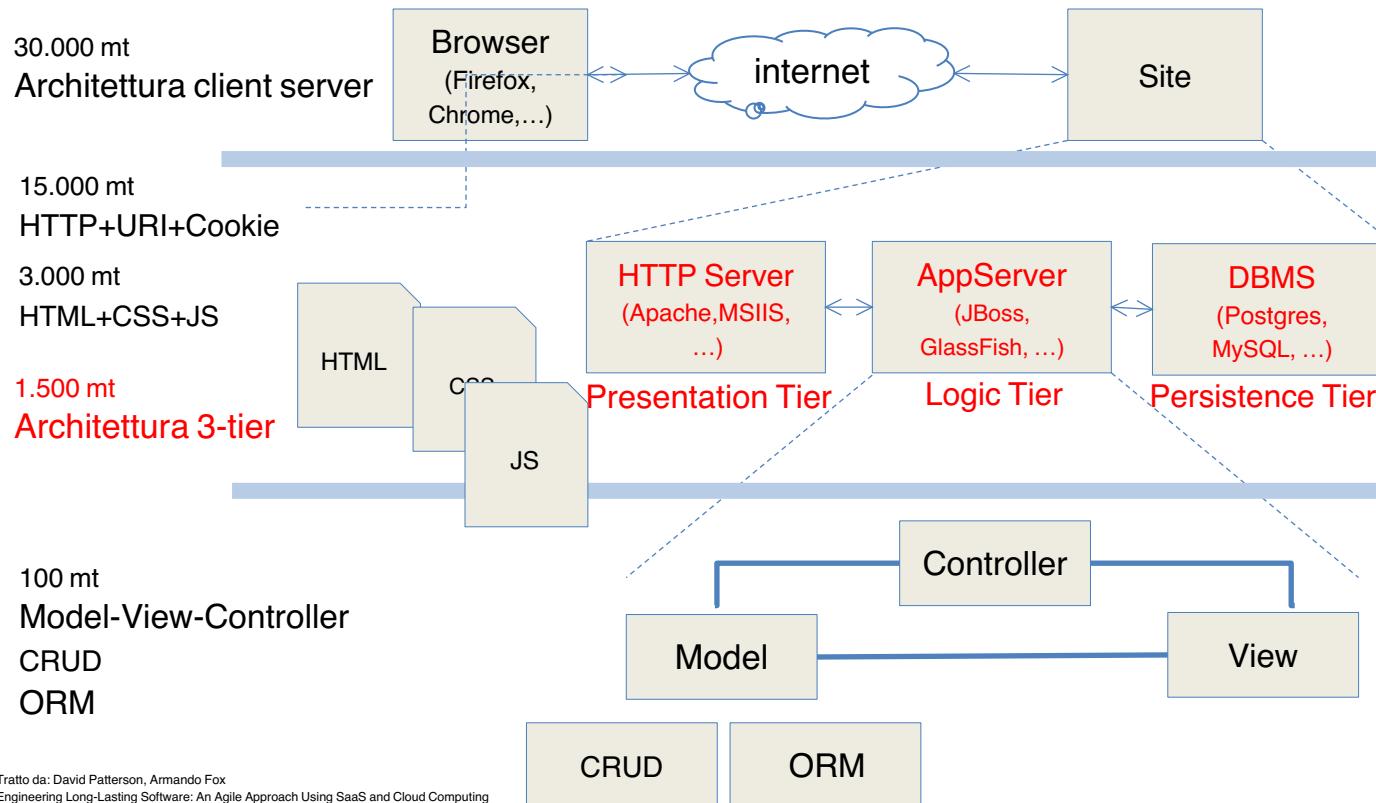
```
<!DOCTYPE html>
<html>
  <head>
    <title>SIW</title>
    <link rel="stylesheet" href="style.css" />
    <script src="filename.js"></script>
  </head>
  <body>
    ... Contenuto della pagina...
  </body>
</html>
```

# JavaScript

- Con le istruzioni JavaScript, offriamo al client (browser) la possibilità di:
  - gestire eventi sul client
  - manipolare i nodi dell'albero HTML
  - chiedere al server dati (che possono essere usati per modificare i contenuti della pagina HTML)



# Anatomia di un sistema informativo su Web



# Architettura n-Tier

- Multi tier (unità di dispiegamento)
- Idealmente ogni unità di dispiegamento corrisponde una struttura fisica (ma con la virtualizzazione questa distinzione è più articolata)

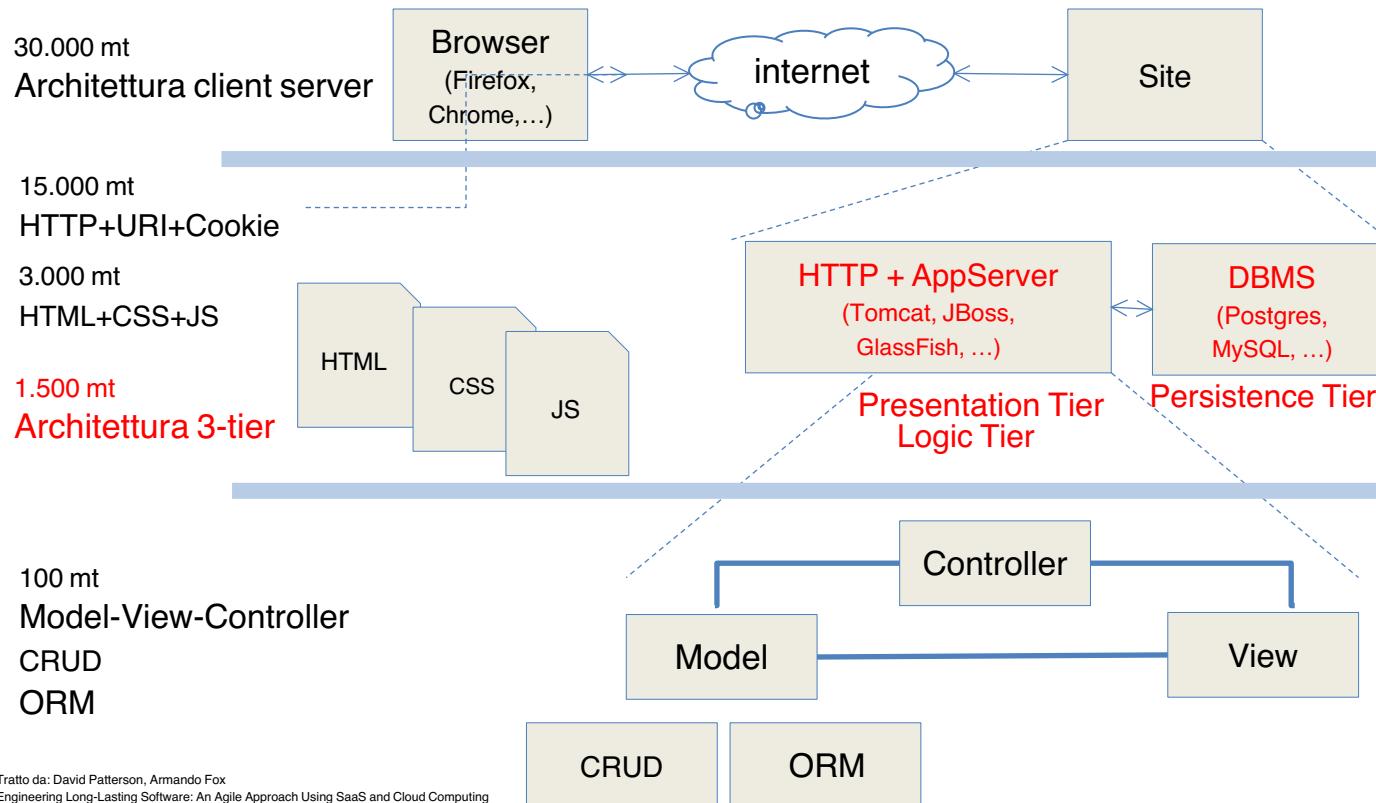
# Architettura n-Tier

- Presentation Tier
  - server HTTP: accetta le richieste e serve direttamente i contenuti statici prendendoli dal file system (css, immagini, etc.)
  - inoltra richieste per contenuti dinamici al Logic Tier

# Architettura n-Tier

- Logic Tier
  - qui gira l'applicazione che genera i contenuti dinamici
  - il ciclo di vita dell'applicazione è gestito da un **Application Server**
- *L'application server di solito include anche un http server e quindi è in grado gestire anche richieste di contenuti statici*

# Anatomia di un sistema informativo su Web (nostra architettura)



# Application Server

- Infrastruttura architetturale: sistema che offre servizi per la gestione e l'esecuzione di applicazioni Web complesse
  - Obiettivo: efficienza (nell'esecuzione delle applicazioni)
- Infrastruttura per la programmazione:  
librerie, paradigmi di programmazione
  - Obiettivo: efficacia (nello sviluppo del sw)

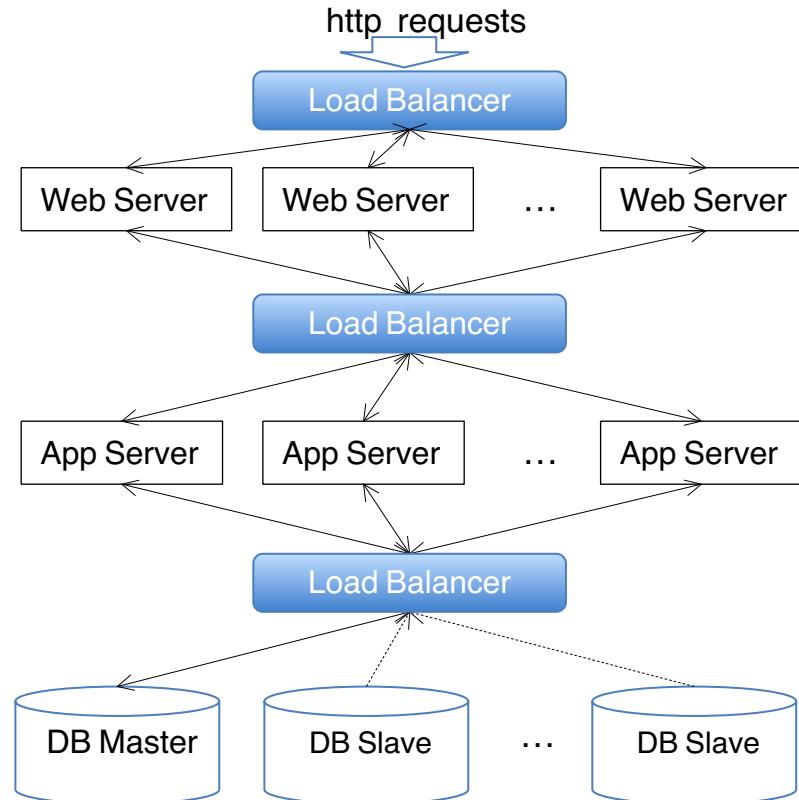
# Architettura n-Tier

- L'architettura n-Tier permette una elevata scalabilità
- Presentation tier e application tier possono essere facilmente replicati orizzontalmente
- Per il persistence tier serve qualche considerazione aggiuntiva

# Architettura n-Tier

- Persistence Tier
  - ha la responsabilità di gestire la persistenza dei dati,
    - di solito è un DBMS relazionale (per noi sarà così)
  - è l'unico Tier in cui non si può applicare una soluzione ridondante *shared-nothing*
  - per ovviare, considerando che spesso nei SIW le operazioni di lettura sono molto più numerose e frequenti di quelle di scrittura, si applicano architetture *master-slave*
    - qualunque *slave* può soddisfare richieste di lettura
    - solo il *master* può soddisfare richieste di scrittura (e si preoccupa di propagare gli aggiornamenti agli slave)

# Architettura n-Tier



# Cloud Computing

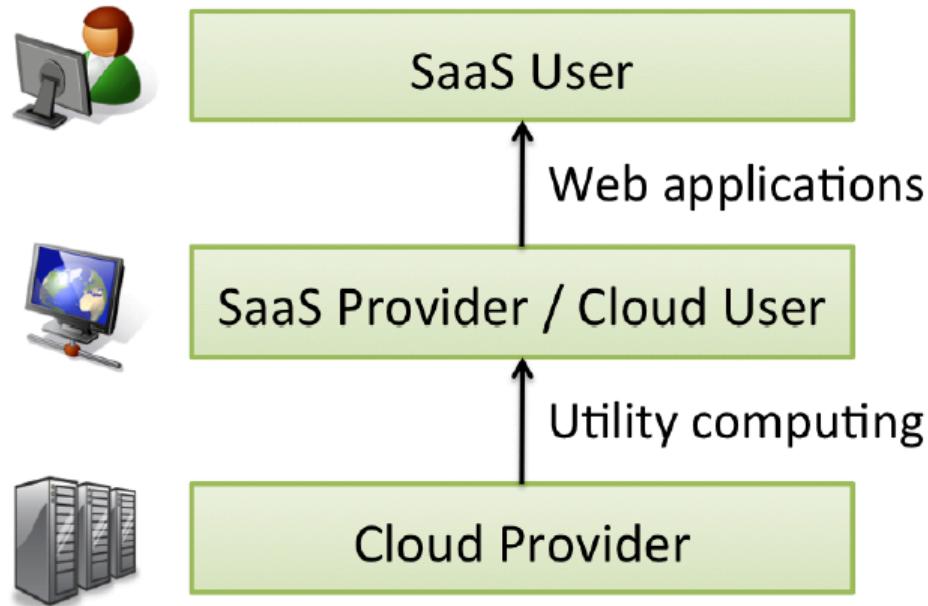
- Uno stack completo dell'architettura n–Tier si può dispiegare in una piattaforma di Cloud Computing
  - Amazon AWS
  - Google AppEngine
  - Microsoft Azure
  - Heroku
  - etc.

# Cloud Computing

- Risorse di calcolo: una commodity (come acqua, energia elettrica, gas)
  - questa visione risale al 1966
- Il termine Cloud Computing si riferisce a:
  - Applicazioni SaaS (Software As A Service)
  - HW e SW nei datacenter che ospitano i servizi SaaS
- HW e SW di gestione dei Datacenter: Cloud



# Cloud Computing



# Software as a Service (SaaS)

- Vantaggi SaaS
  - Service providers: installazione e manutenzione, controllo centralizzato
  - End users: accesso al servizio “anytime, anywhere”, condivisione dati e collaborazione elaborazione, dati memorizzati nell'infrastruttura
- Cloud Computing permette a chiunque di offrire servizi SaaS senza dover possedere un datacenter (no "Gatekeepers")

# Cloud Computing

- Dal punto di vista dell'hardware
  1. **Illusione di infinite risorse di calcolo:** non c'è più la necessità di pianificare l'approvvigionamento di risorse;
  2. **Flessibilità:** si può partire con poche risorse hw e farle crescere (o decrescere) a seconda delle necessità;
  3. **Pagamento a consumo:** si paga solo ciò che si consuma.

# Cloud Computing

## Illusione di infinite risorse di calcolo



Lettura consigliata: "The Datacenter as a Computer" (Google)

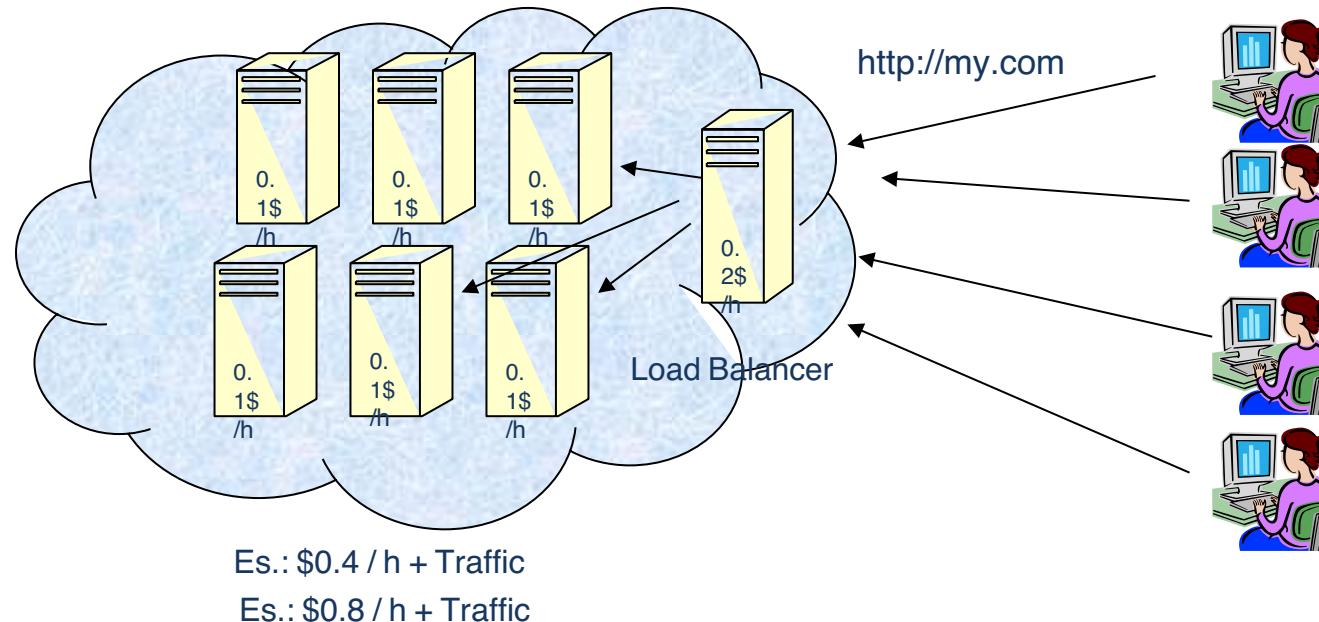
<http://www.morganclaypool.com/doi/abs/10.2200/S00193ED1V01Y200905CAC006>

Video interessante: "Google container data center tour"

<http://www.youtube.com/watch?v=zRwPSFpLX8I>

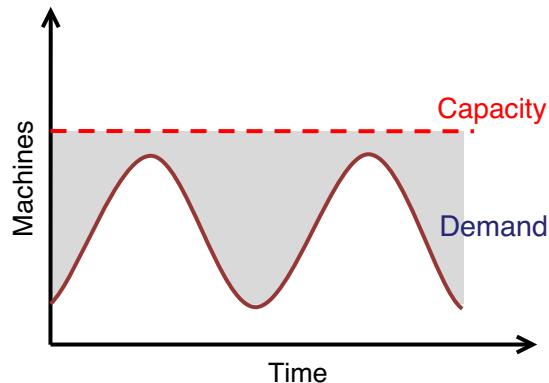
# Cloud Computing

Flessibilità + Pagamento a Consumo

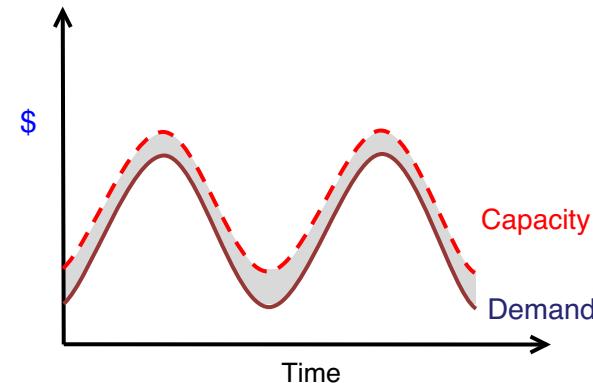


# Cloud Computing

Flessibilità: un approvvigionamento statico tarato sui picchi spreca risorse, ma è necessario per soddisfare gli SLA\*



“Statically provisioned”  
data center



“Virtual” data center  
in the cloud

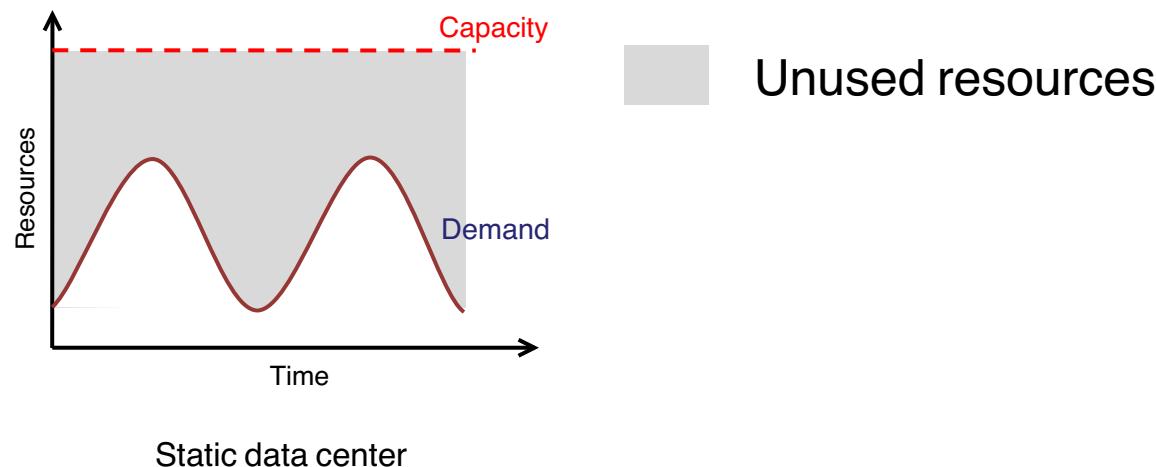


Unused resources

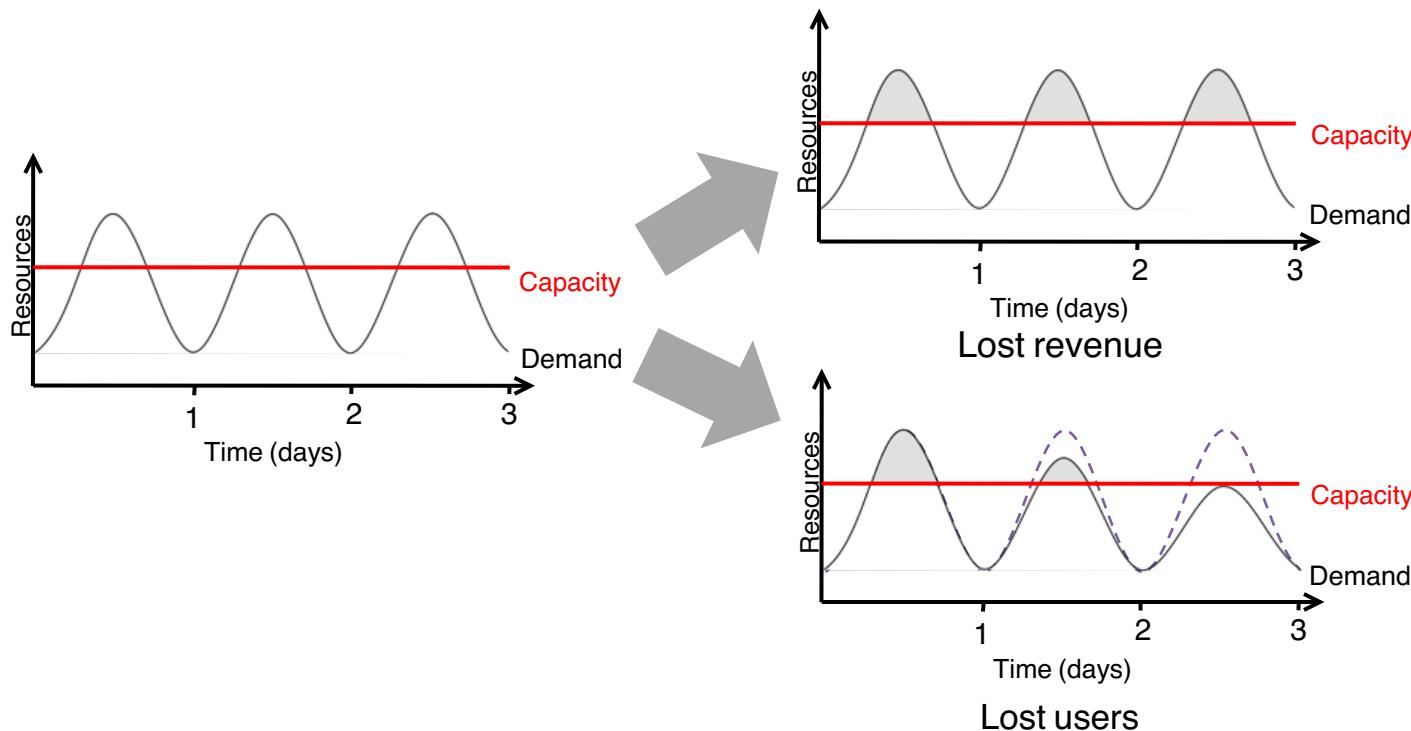
\* SLA = Service Level Agreement (Accordi sul livello di servizio)

# Rischi di sovradimensionamento

Sottoutilizzo (e quindi spreco) di risorse se le predizioni sul picco sono troppo ottimistiche



# Rischi di sottodimensionamento



# Cloud Computing

- Esempio (datato, ma efficace)
  - Elastic Compute Cloud (EC2) from Amazon Web Services (AWS) sells 1.0-GHz x86 “slices” for 10 cents per hour, and a new “slice”, or instance, can be added in 2 to 5 minutes
  - Amazon’s Scalable Storage Service (S3) charges \$0.12 to \$0.15 per gigabyte-month, with additional bandwidth charges of \$0.10 to \$0.15 per gigabyte to move data in to and out of AWS over the Internet
- Vedi: <https://aws.amazon.com/it/ec2/pricing/on-demand/>

# Cloud Computing

- Not (just) Capital Expense vs. Operation Expense!
- “Cost associativity”: 1,000 CPUs for 1 hour same price as 1 CPUs for 1,000 hours (@\$0.08/hour)
- ***Major enabler for SaaS startups***
  - Animoto (<http://animoto.com>) traffic doubled every 12 hours for 3 days when released as Facebook plug-in
  - Scaled from 50 to >3500 servers
  - ***...then scaled back down***
- Gets IT gatekeepers out of the way
  - not unlike the PC revolution

<http://animoto.com/>

<http://blog.animoto.com/2008/04/21/amazon-ceo-jeff-bezos-on-animoto/>

A satellite image of the Earth's surface, focusing on the Northern Hemisphere. It shows the continents of Europe, Africa, and the Middle East. The land is depicted in various shades of green, brown, and tan, representing vegetation and terrain. The oceans are a deep blue. The text "Client Server" is overlaid in the upper left quadrant.

Client Server

A satellite map of Europe showing landmasses in green and brown, and bodies of water in blue. The word "HTTP" is overlaid in large, white, sans-serif capital letters, centered over the continent.

HTTP

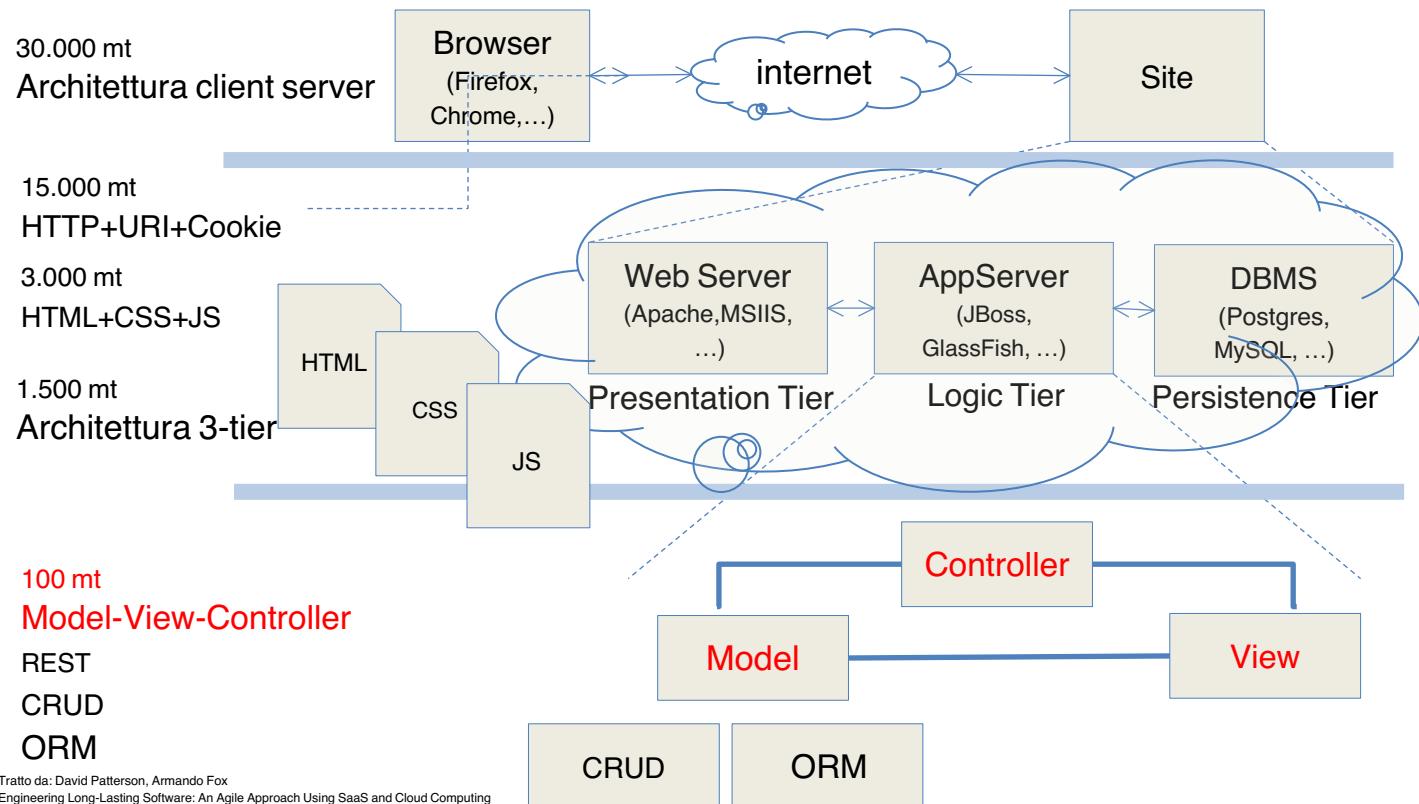
The background of the image is a high-resolution aerial photograph of a city, likely Paris, showing a dense grid of buildings, a winding river (the Seine), and various parks and green spaces. The text is overlaid on this image.

HTML – CSS - Js

An aerial photograph of the Colosseum and the surrounding Roman archaeological sites in Rome, Italy. The Colosseum is the central focus, with its distinctive elliptical shape and tiered seating. To its left are the remains of the Subura district and the Horti Lamiani. To the right is the Circo Massimo. The image shows a mix of ancient structures, modern infrastructure like roads and cars, and green spaces.

n-tier

# Anatomia di un sistema informativo su Web



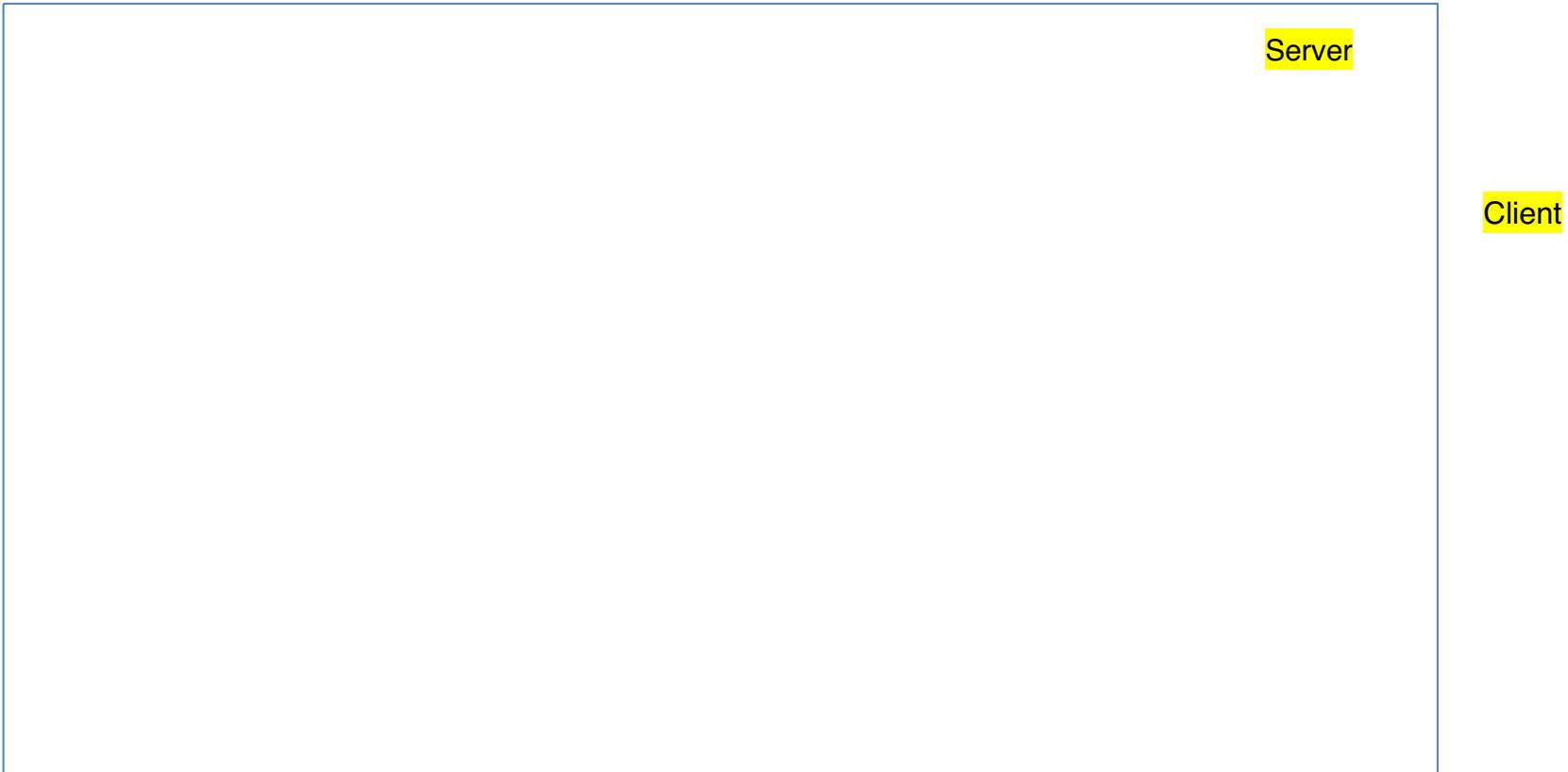
An aerial photograph of a road intersection. The roads are grey asphalt with white zebra crossings at the intersections. There are several trees with green foliage in the upper left corner. The text "MVC- CRUD-ORM" is overlaid in red.

MVC- CRUD-ORM

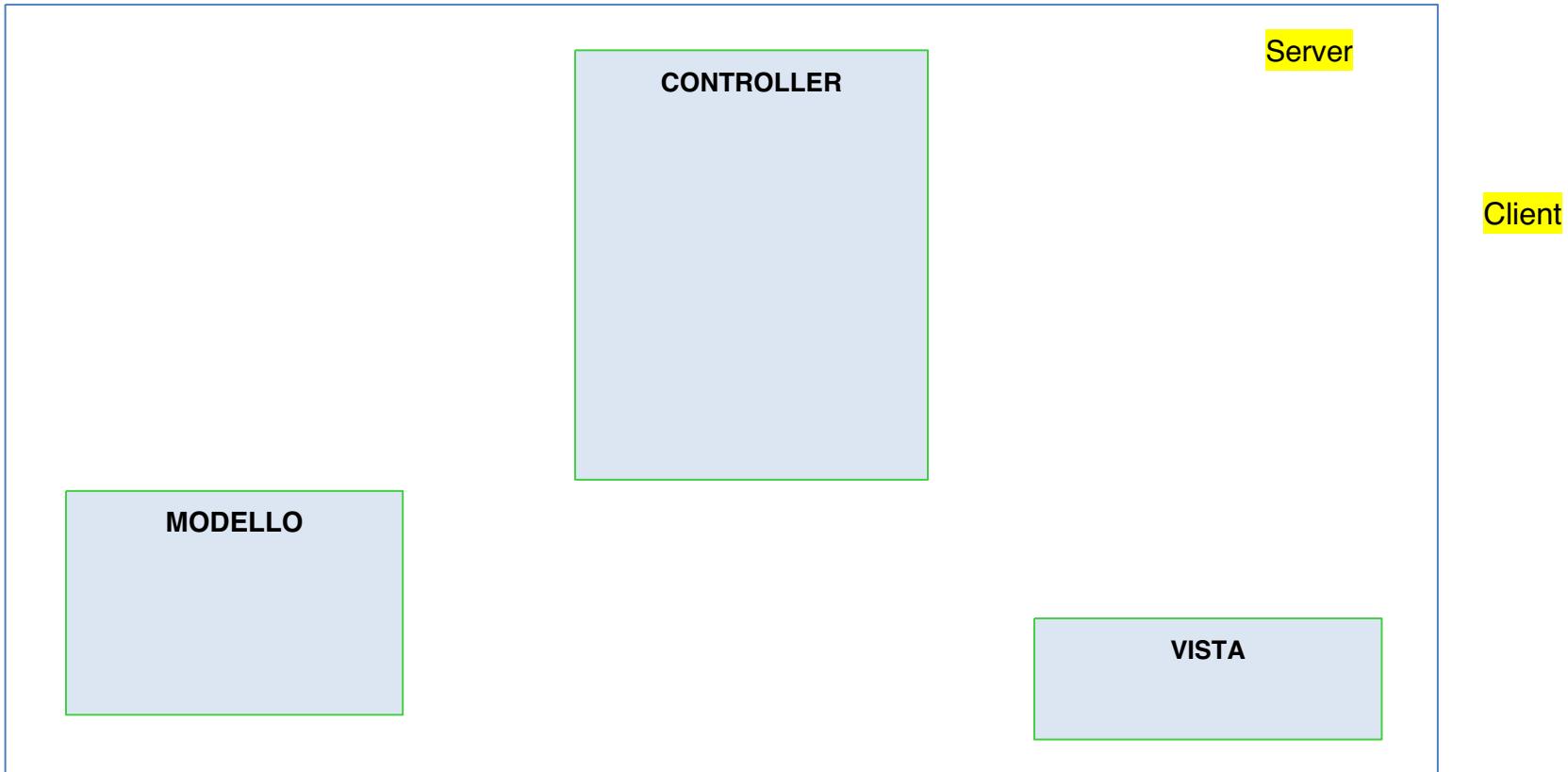
# Come viene gestita una richiesta Web

- L'application server riceve una richiesta HTTP  
GET <http://uniroma3.it/?s=merialdo>
- L'url viene mappato su un metodo di una classe **Controller**
  - Le classi Controller sono specializzate nel gestire la richiesta: verificano la validità dei dati passati dal client tramite parametri associati alla richiesta http (es., i voti e la data di un esame)
- Il Controller invoca metodi di una classe del **Modello**
  - Le classi del Modello encapsulano i dati e le operazioni offerte dal sistema (es., prenotazione esame, elenco iscritti)
- Sulla base della risposta del Modello, il Controller decide quale sarà la risposta da inviare al client e delega la responsabilità di creare la risposta ad una opportuna **Vista**
  - Le Viste sono classi specializzate nel produrre una risposta. Di solito la risposta è una pagina HTML contenente i dati del modello (es. pagina con l'elenco degli iscritti ad un esame)

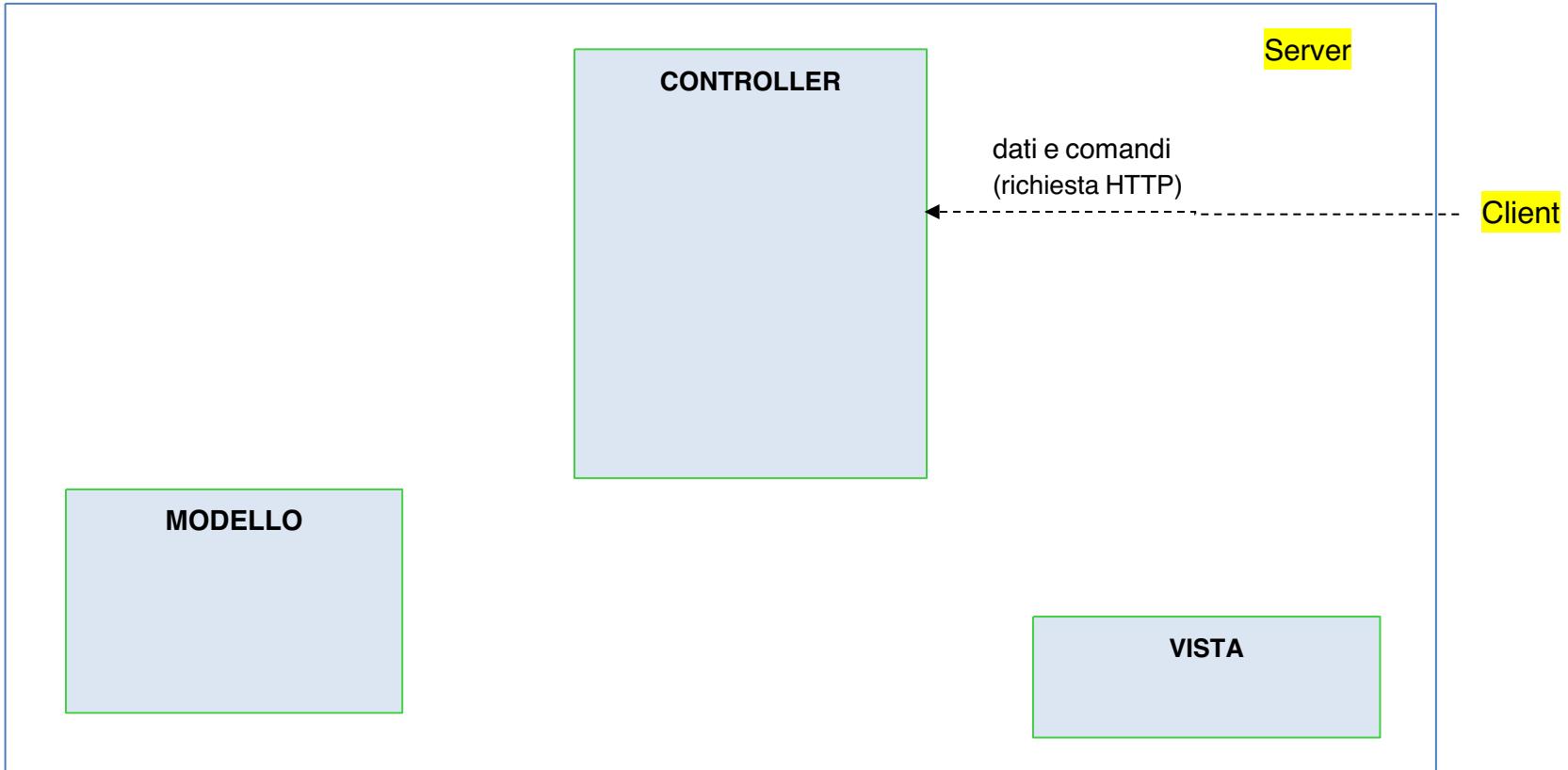
# Model-View-Controller



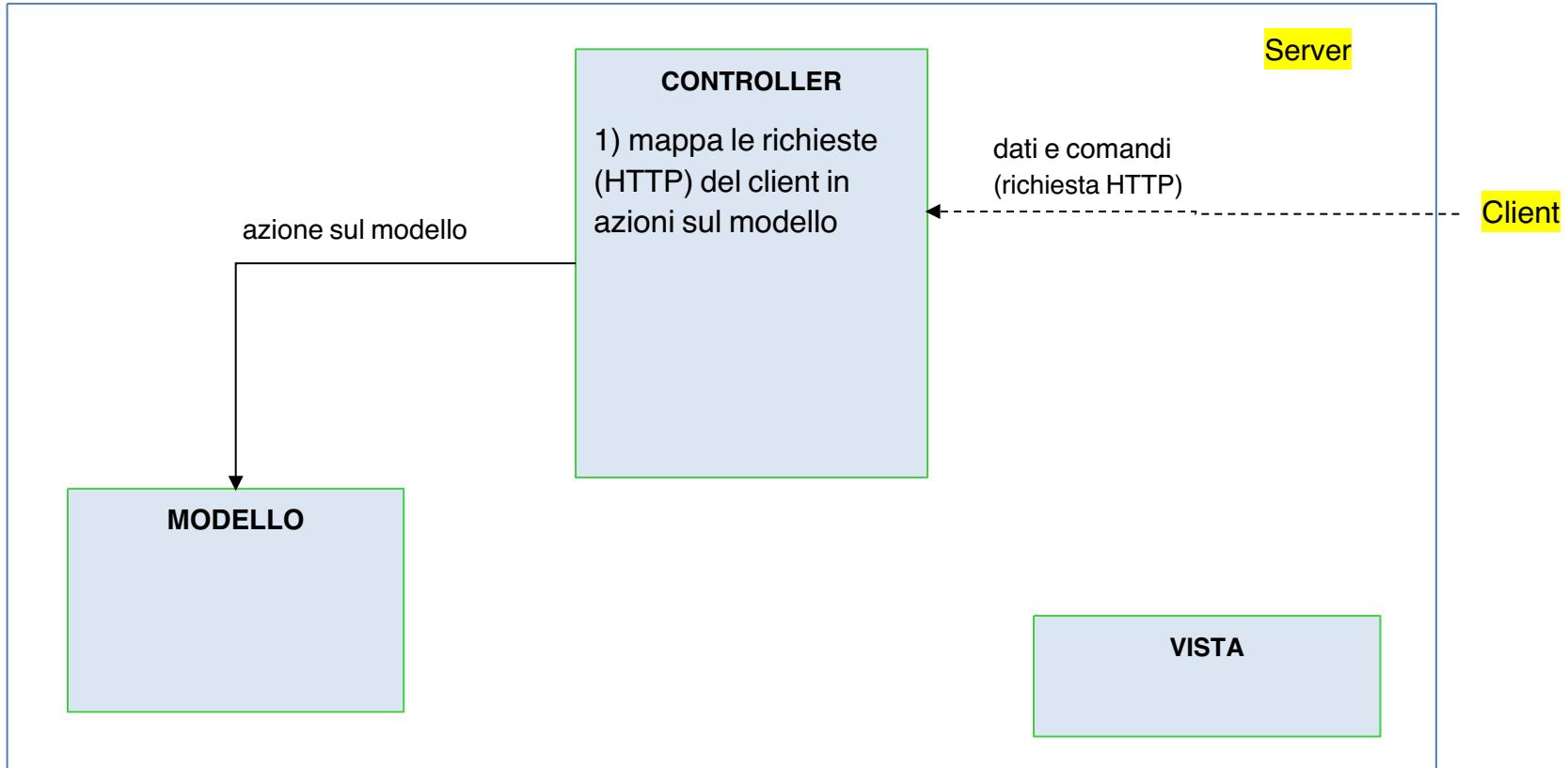
# Model-View-Controller



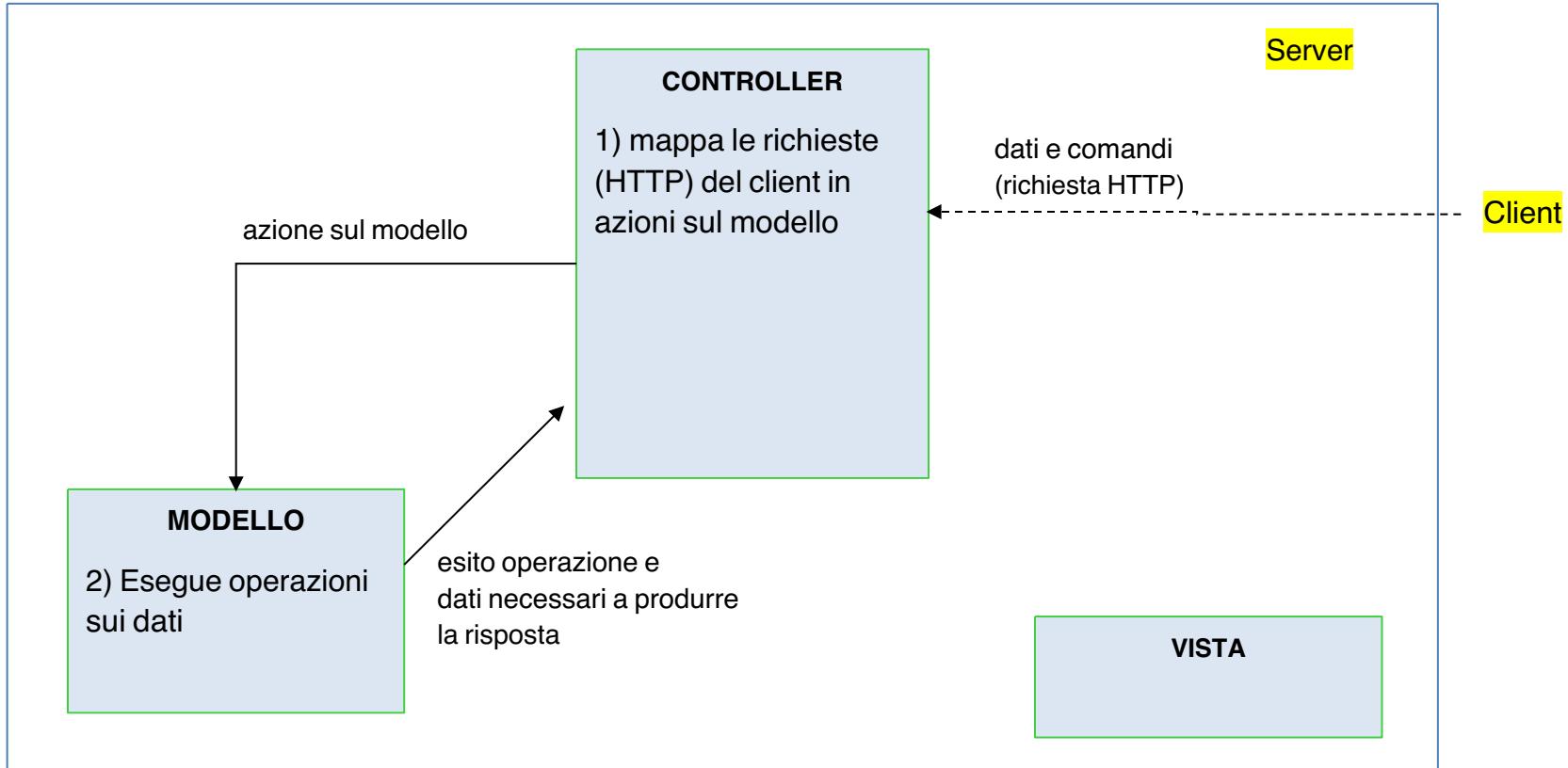
# Model-View-Controller



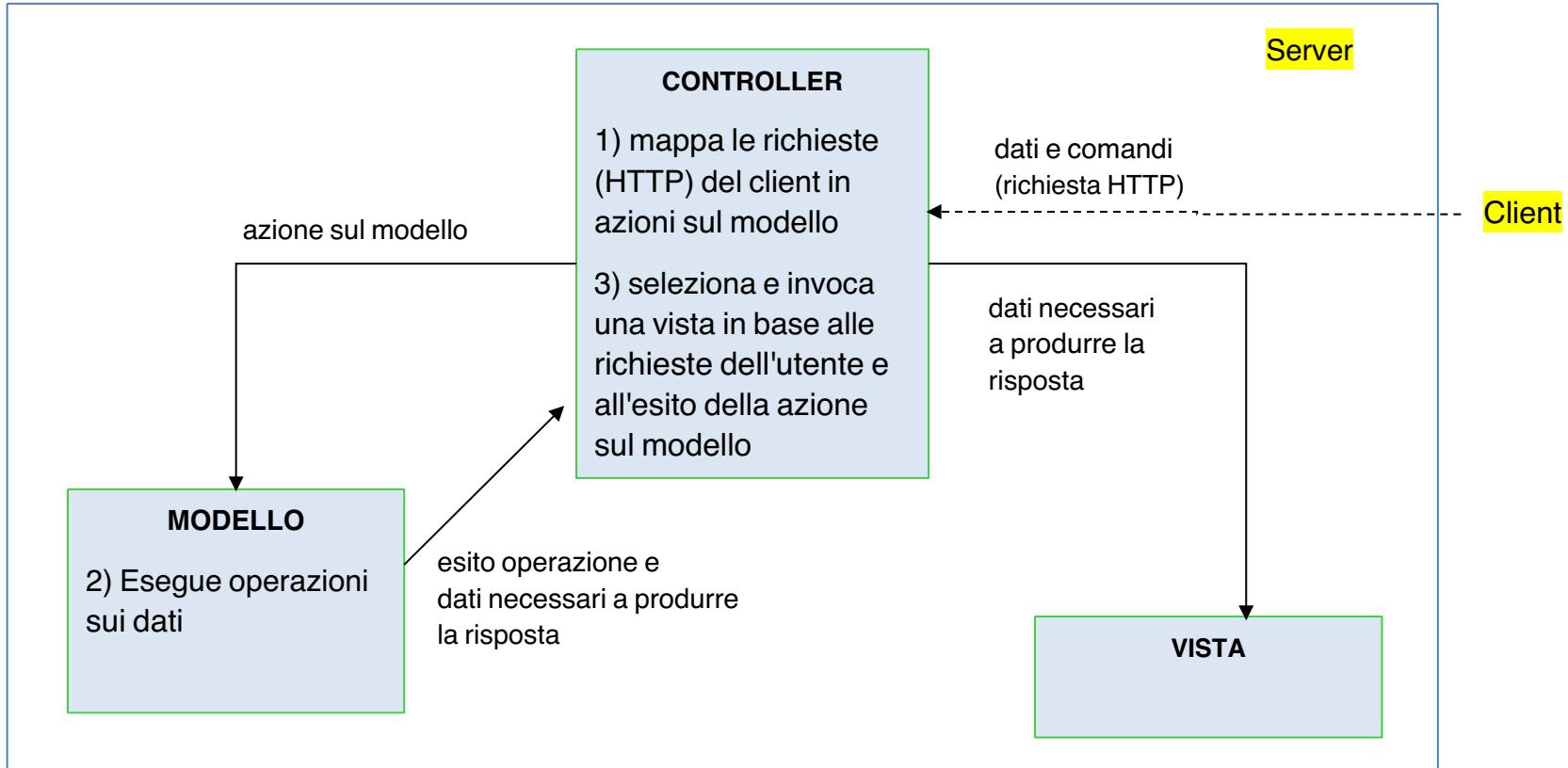
# Model-View-Controller



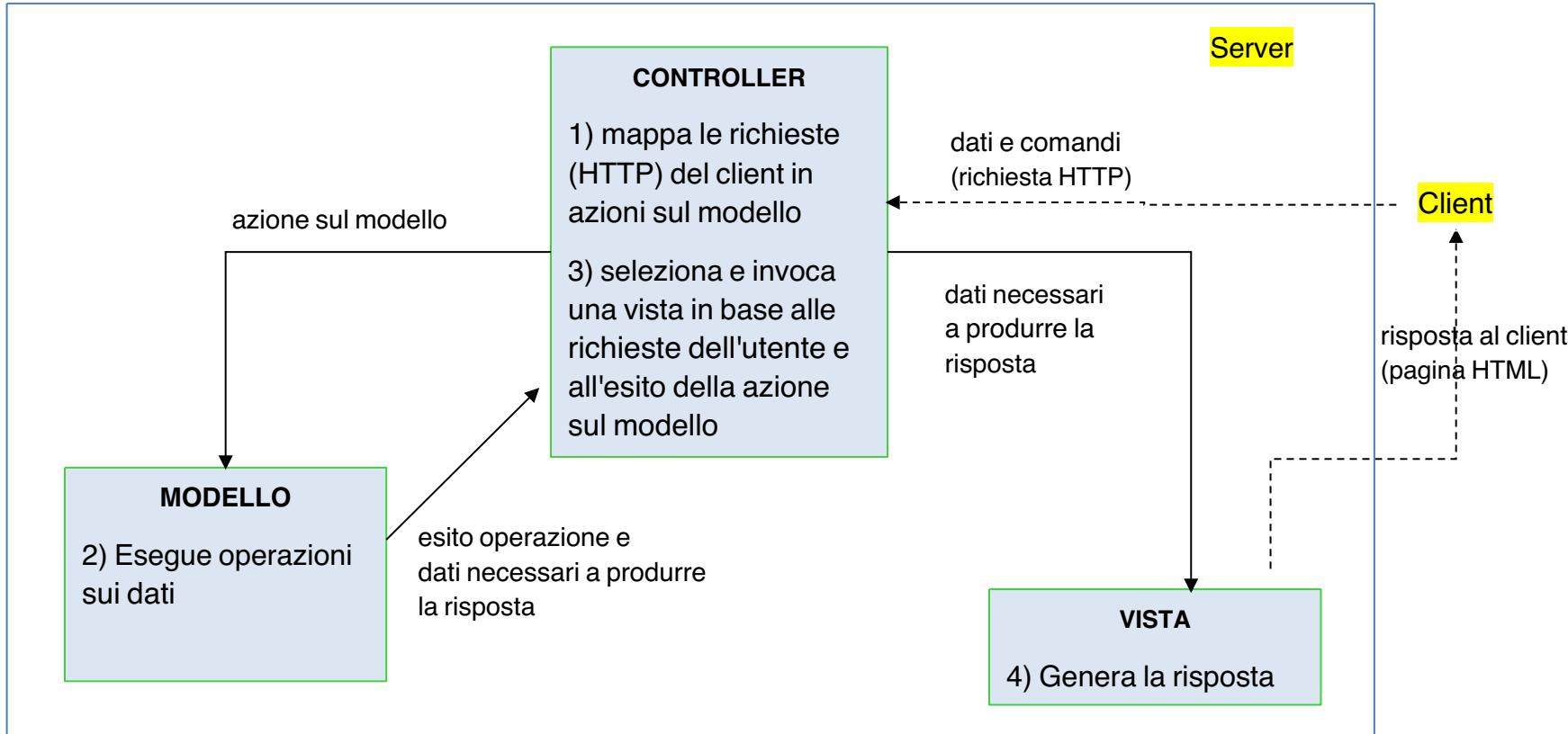
# Model-View-Controller



# Model-View-Controller



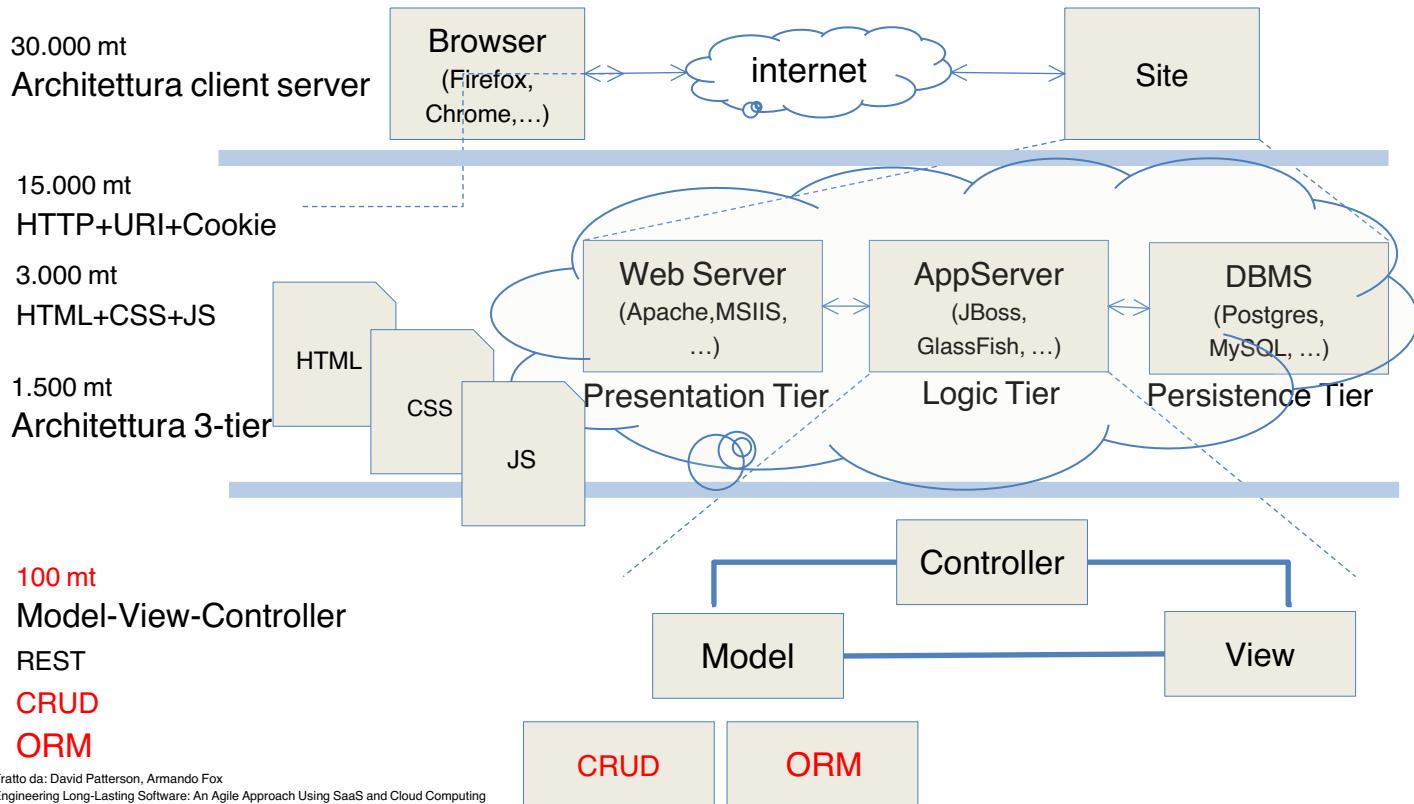
# Model-View-Controller



# Model-View-Controller

- In Java (Spring):
  - Model
    - @Entity: classi Java che modellano le entità del dominio
    - @Service: classe stateless, offre metodi che corrispondono alle operazioni offerte dal sistema
  - View
    - pagine dinamiche (JSP, Thymeleaf, o altre tecnologie di templating): componenti specializzati nella generazione della risposta
  - Controller
    - @Controller: classe Java specializzata a gestire la richiesta e a smistare la gestione della risposta

# Anatomia di un sistema informativo su Web



# CRUD

- I dati devono essere resi persistenti, cioè salvati in un database
- Le principali operazioni della persistenza sono:
  - **C**reate: crea una nuova entità nel database
  - **R**etrieve: recupera una (o più) entità dal database
  - **U**pdate: aggiorna una (o più) entità del database
  - **D**elete: cancella una (o più) entità dal database

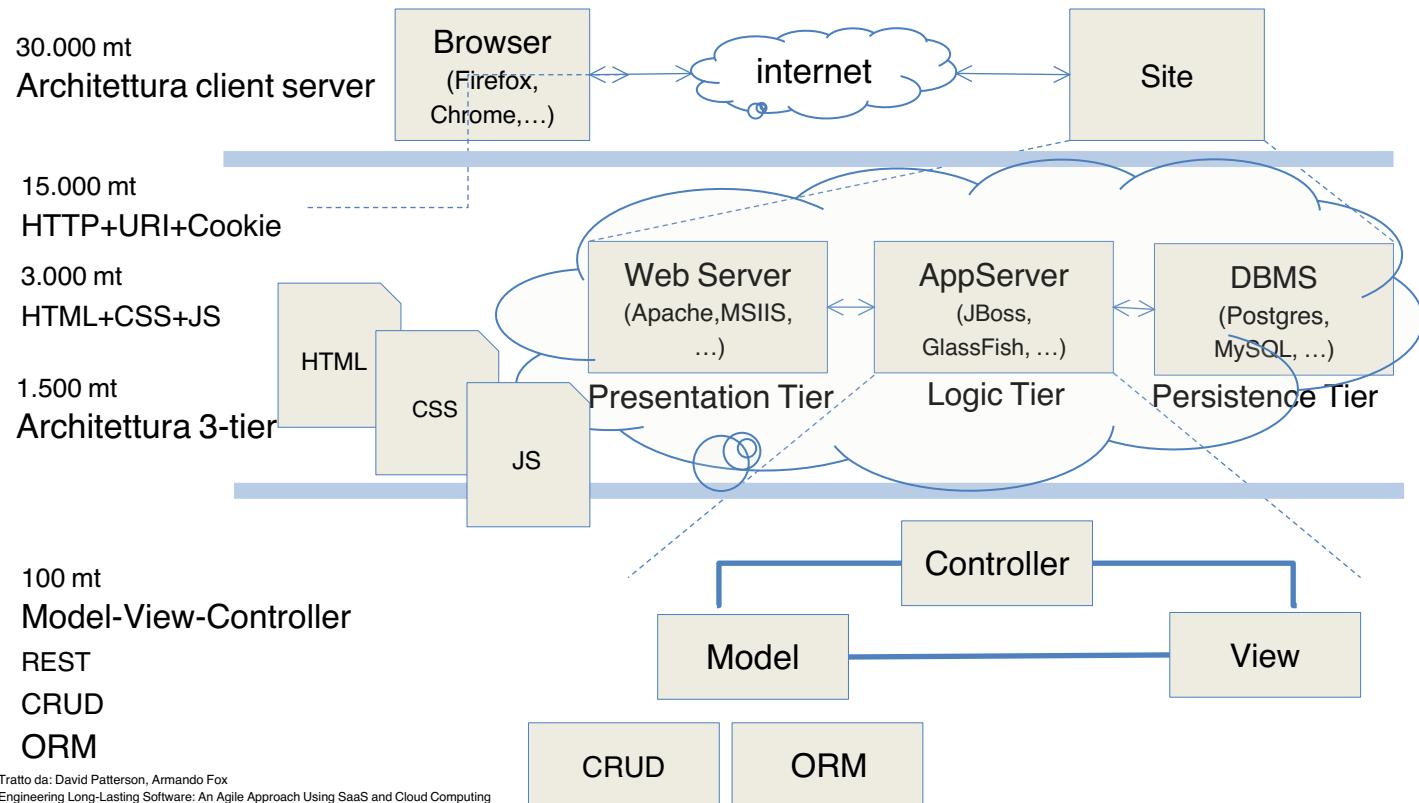
# CRUD

- Problema principale: conciliare in maniera efficace ed efficiente il mondo OO con il mondo relazionale
  - classi – tabelle
  - oggetti – tuple
  - riferimenti – foreign key
  - navigazione – join
- Altri problemi: consistenza e concorrenza
  - gestione delle transazioni

# ORM

- Scrivere il codice per la gestione della persistenza è estremamente noioso, ripetitivo, e fonte di errori
- Gli ORM (Object Relational Mapping) sono librerie che facilitano questa operazione
  - modello complesso: la semplicità del codice si paga con una maggiore complessità concettuale
  - Esempi: JPA, Hibernate

# Anatomia di un sistema informativo su Web



**Vice sceriffo Wendell:** Ragionamento lineare, sceriffo  
**Sceriffo Ed Tom Bell:** Con l'età, si diventa linearì