# Desenvolvimento de Aplicações com Arquitetura Baseada em Microservices

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1007] - Tópicos Avançados em SI 4
https://github.com/vinicius3w/if1007-Microservices

# Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-CompartilhaIgual 3.0 Não Adaptada


Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

AssertLab
Advanced Software and Systems Engineering Research Technologies

Centro de Informática

40 anos

# Ementa

- Microservice é um estilo e padrão de arquitetura de software em que sistemas complexos são decompostos em serviços menores que trabalham em conjunto para formar serviços maiores. Os Microservices são serviços autônomos, independentes e independentemente implantáveis. No mundo de hoje, muitas empresas usam microservices como principal padrão para a construção de aplicativos corporativos grandes e orientados a serviços.

# Ementa

- O framework Spring é um framework de programação popular com a comunidade de desenvolvedores por muitos anos. O Spring Boot removeu a necessidade de ter um conteiner leve de aplicação e forneceu um meio para implantar aplicativos leves e sem servidor. O Spring Cloud combina muitos componentes OSS Netflix e fornece um ecossistema para executar e gerenciar microservices em grande escala. Ele fornece recursos como balanceamento de carga, registro de serviço, monitoramento, gateway de serviço e assim por diante.

# Ementa

- No entanto, os microservices vêm com seus próprios desafios, como monitoramento, gerenciamento, distribuição, dimensionamento, descoberta, etc., especialmente quando se implanta em escala. A adoção de microservices sem abordar os desafios dos microservices comuns levaria a resultados catastróficos. A parte mais importante deste curso é discutir um modelo de capacidade de microservice agnóstico em termos de tecnologia que busca ajudar a resolver os desafios mais comuns do mundo de microservice.

# Objetivos

- Espera-se que os alunos vivenciem exposição prática a ferramentas, processos e princípios do desenvolvimento de aplicações baseadas na arquitetura de microservices, juntamente com as boas práticas e técnicas de implantação nos princípios de DevOps e as vantagens e desafios do uso e imersão na Computação em Nuvem através de projetos práticos, enquanto compreendem modelos e ideias de pesquisa por trás das ferramentas e processos. As aulas incluirão experiências de aprendizagem no estilo de workshops, onde os alunos irão trabalhar em um problema e receberão comentários do professor, colaboradores convidados e outros colegas de classe. Quando possível, palestras convidadas da indústria ajudarão a ilustrar exemplos de como a tecnologia é implantada na prática.

# Metodologia

- Na disciplina, será utilizada uma mistura de aulas tradicionais com atividades e workshops em sala de aula. Durante as aulas, abordaremos conceitos básicos relacionados aos tópicos a serem tratados na disciplina. Durante os workshops em sala de aula, realizaremos exemplos de exercícios com ferramentas relevantes que reforçam o material de aula. As avaliações serão baseadas nas aulas, exercícios dirigidos, workshops e o projeto final.

# Pré-Requisitos

- Para participar deste curso é desejável que os participantes tenham:
  - Conhecimento básico de inglês técnico para leitura, estudo e acompanhamento das atividades propostas, uma vez que a maior parte do material disponível está em inglês.
  - Conhecimento básico de arquitetura de computador, sistemas operacionais, redes, engenharia de software e banco de dados.
  - Conhecimento de modelos de processo de desenvolvimento de software, em especial Metodologias Ágeis, testes de software. Conhecimento em TDD e BDD serão uma vantagem. Maiores informações podem ser consultadas no site da disciplina de Engenharia de Software (IF977).
  - Experiência em sistemas de computação, armazenamento, infraestruturas de rede e computação em nuvem será uma vantagem.

# Plágio & Código de Honra

- Não trabalhe nos exercícios com aqueles que não estão no seu grupo
- Não copie e cole grandes partes de seu exercício de fontes de terceiros
- Perguntas?

# Visão Geral do Projeto

- Crie uma aplicação web digna de um portfólio, peça por peça
- Entregáveis semanais de acordo com tópicos de aula
- Grupos de projetos de duas/três pessoas
- Aplicação Web dinâmica, consumindo serviçosWeb, arquitetura e características de microservices e especificações de contêineres

# Tópicos

- Demystifying Microservices
- Building Microservices with Spring Boot
- Applying Microservices Concepts
- Microservices Evolution – A Case Study
- Scaling Microservices with Spring Cloud
- Autoscaling Microservices
- Logging and Monitoring Microservices
- Containerizing Microservices with Docker
- Managing Dockerized Microservices with Mesos and Marathon
- The Microservices Development Life Cycle

# Resources

- There is no textbook required. However, the following are some books that may be recommended:
  - Building Microservices: Designing Fine-Grained Systems
  - Spring Microservices
  - Spring Boot: Acelere o desenvolvimento de microsserviços
  - Microservices for Java Developers A Hands-on Introduction to Frameworks and Containers
  - Migrating to Cloud-Native Application Architectures
  - Continuous Integration

# Warm up

Based on Prof. Thomas LaToza (SWE 432) material

# What is the web?

- A set of standards
- TCP/IP, HTTP, URLs, HTML, CSS, ...
- A means for distributing structured and semi-structured information to the world
- Infrastructure

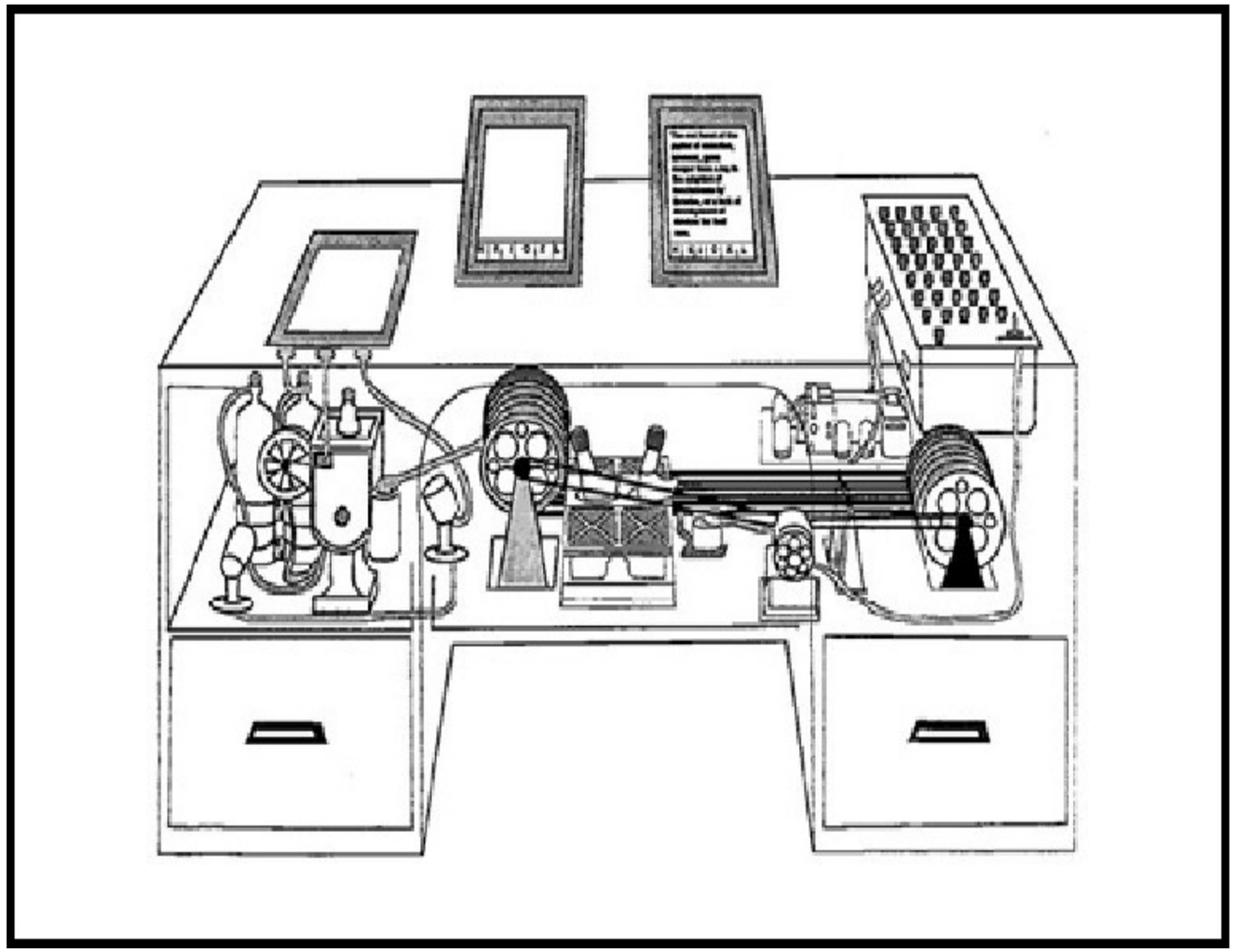
# Pre-Web

- "As We May Think", by Vannevar Bush, in The Atlantic Monthly, July 1945
- Recommended that scientists work on inventing machines for storing, organizing, retrieving and sharing the increasing vast amounts of human knowledge
- He targeted physicists and electrical engineers - there were no computer scientists in 1945

# Pre-Web - MEMEX

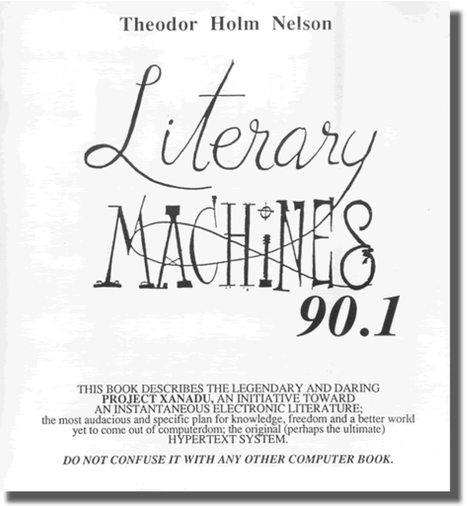
- MEMEX = MEMory Extension
- Create and follow “associative trails” (links) and annotations between microfilm documents
- Technically based on “rapid selectors” Bush built in1930’s to search microfilm
- Conceptually based on human associative memory rather than indexing

# Pre-Web - Memex



# Hypertext and the WWW

- 1965: Ted Nelson coins “hypertext” (the HT in HTML) - “beyond” the linear constraints of text
- Many hypertext/hypermedia systems followed, many not sufficiently scalable to take off
- 1968: Doug Engelbart gives “the mother of all demos”, demonstrating windows, hypertext, graphics, video conferencing, the mouse, collaborative real-time editor
- 1969: ARPANET comes online
- 1980: Tim Berners-Lee writes ENQUIRE, a notebook program which allows links to be made between arbitrary nodes with titles
-



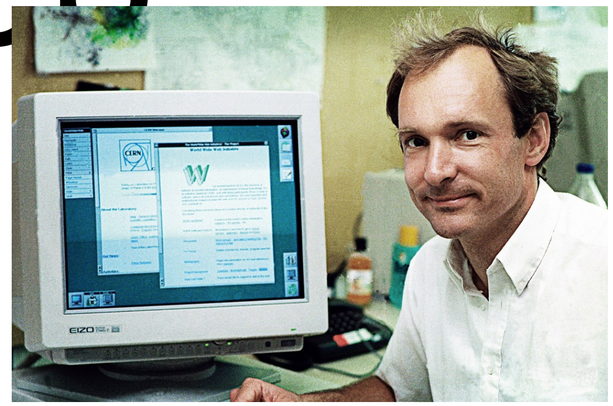

AssertLab
Advanced Software and Systems Engineering Research Technologies

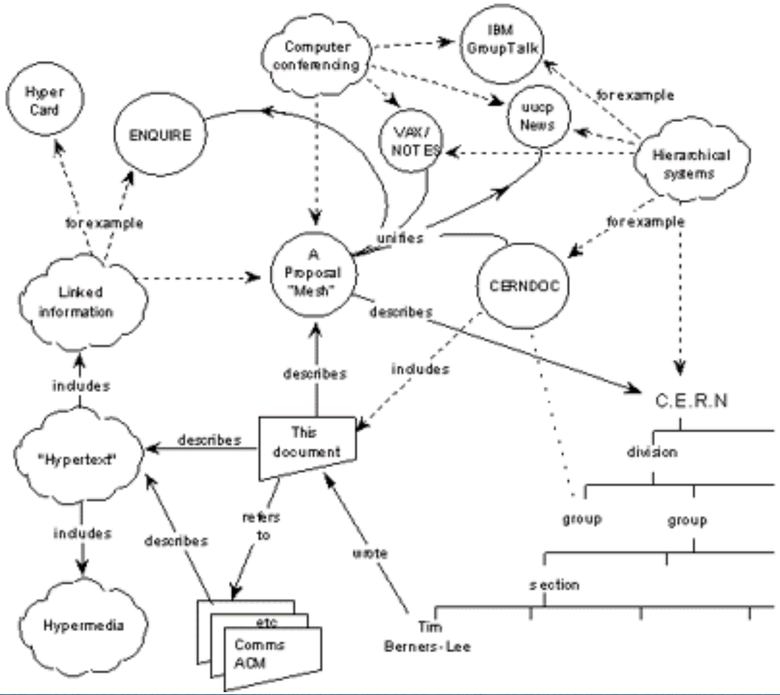
Centro de Informática

40 anos

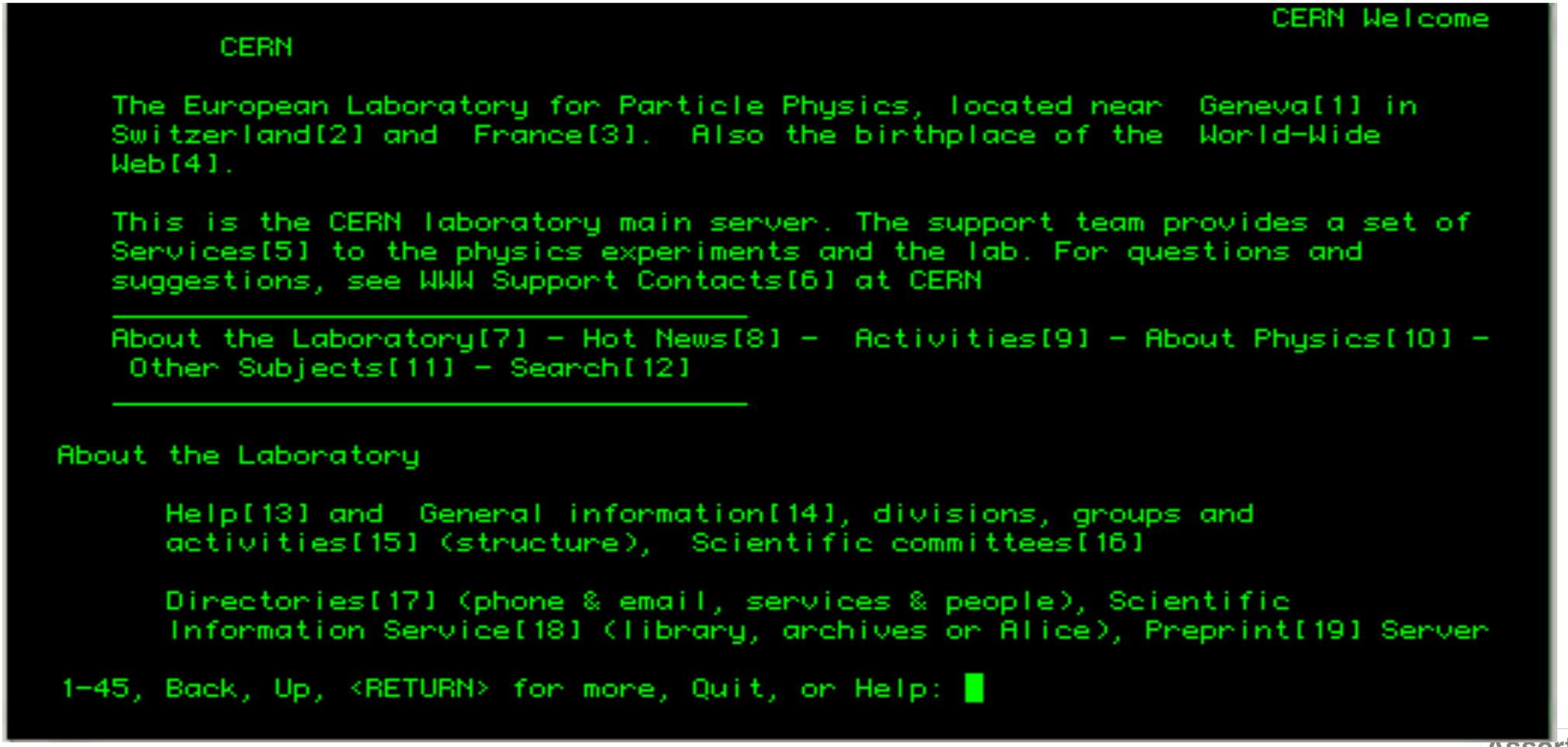
# Origin of the Web

- 1989: Tim Berners-Lee, “Information Management: A Proposal”
  - Became what we know as the WWW
  - A “global” hypertext system full of links (which could be single directional, and could be broken!)

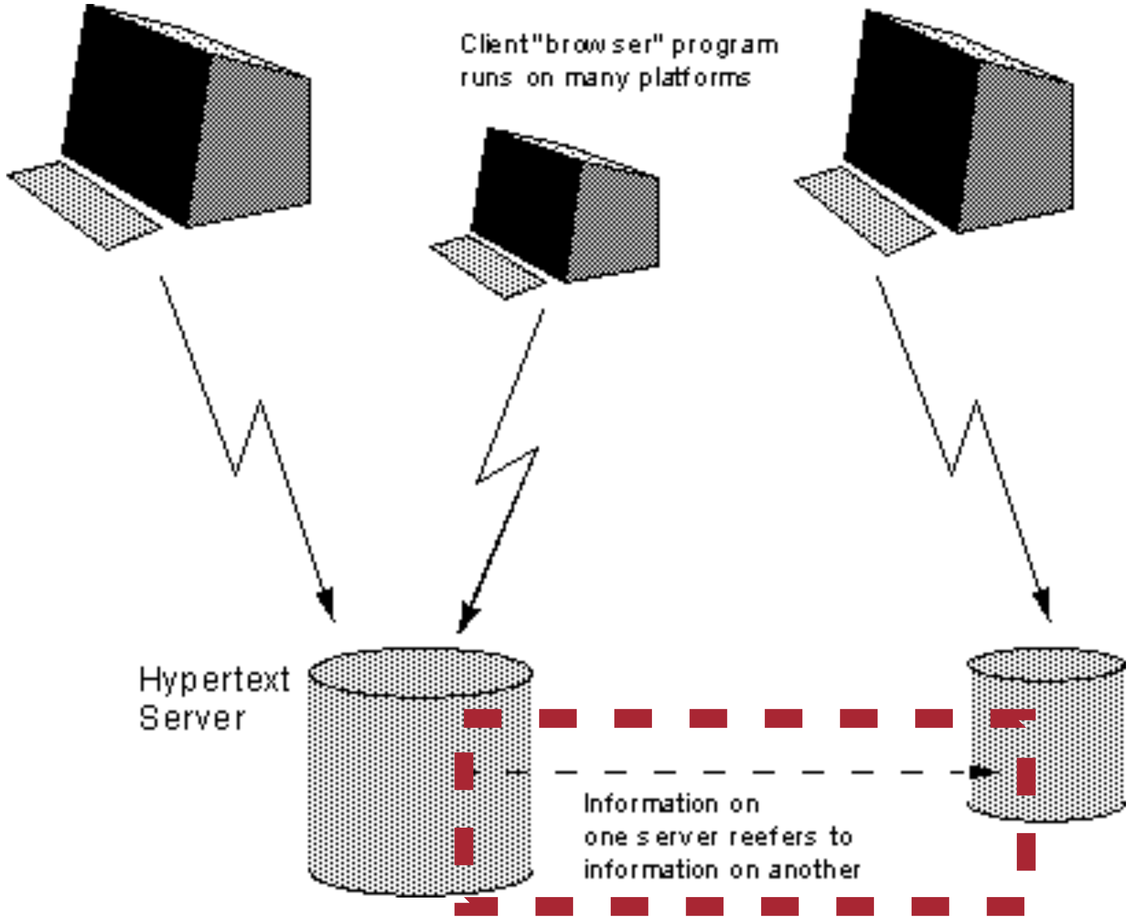




# Early Browsers



# Original WWW Architecture




**Links!!**

# URI: Universal Resource Identifier

URI: <scheme>://<authority><path>?<query>

http://cs.gmu.edu/syllabus/syllabi-fall16/SWE432BellJ.html

"Use HTTP scheme"

Other popular schemes:
ftp, mailto, file

"Connect to cs.gmu.edu"

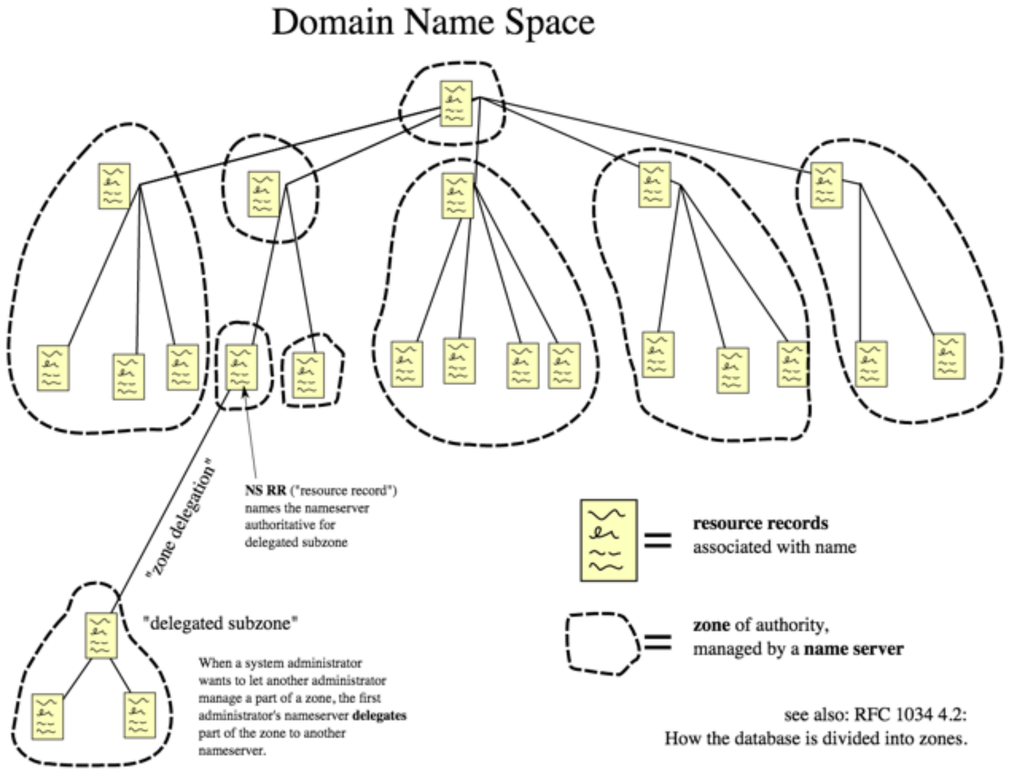
May be host name or an IP address
Optional port name (e.g., :80 for port 80)

"Request syllabus/syllabi-fall16/SWE432BellJ.html"

More details: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

# DNS: Domain Name System

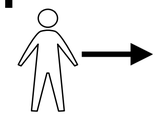
- Domain name system (DNS) (~1982)
  - Mapping from names to IP addresses
- e.g. cs.gmu.edu -> 129.174.125.139

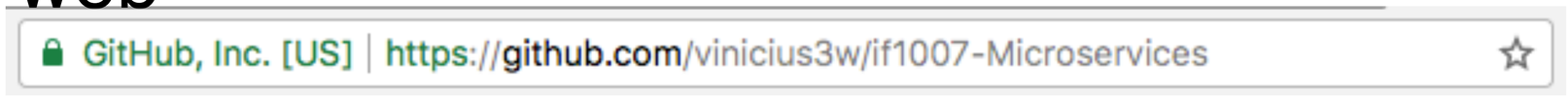



The hierarchical Domain Name System for class *Internet*, organized into zones, each served by a name server

# HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is trans
on the web






HTTP Request

**GET** /vinicius3w/if1007-Microservices/blob/master/README.md/ HTTP/1.1

Host: github.com

Accept: text/html

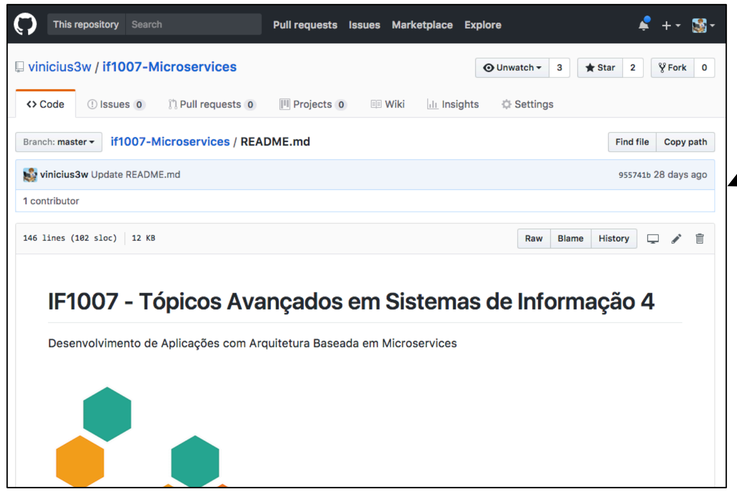
web server

Reads file from disk

*HTTP Response*

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```



# HTTP Requests

HTTP Request

GET /vinicius3w/if1007-Microservices/blob/master/README.md HTTP/1.1

Host: github.com

Accept: text/html



"GET request"

Other popular types:
POST, PUT, DELETE, HEAD

"Resource"

- Request may contain additional header lines specifying, e.g. client info, parameters for forms, cookies, etc.
- Ends with a carriage return, line feed (blank line)
- May also contain a message body, delineated by a blank line

# HTTP Responses

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

"OK response"

Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

"HTML returned content"

Common MIME types:
application/json
application/pdf
image/png

[HTML data]

# Properties of HTTP

- Request-response
  - Interactions always initiated by client request to server
  - Server responds with results
- Stateless
  - Each request-response pair independent from every other
  - Any state information (login credentials, shopping carts, etc.) needs to be encoded somehow

# HTML: HyperText Markup Language

HTML is a markup language - it is a language for describing parts of a

# HTML: HyperText Markup Language

- NOT a programming language
- Tags are added to markup the text, encompassed with <>'s
- Simple markup tags: <b>,<i>, <u> (bold, italic, unde

```
<b>This text is bold!</b>
```

↓

**This text is bold!**

# Web vs. Internet

**Web** — **HTML** **CSS** **Browser**

| | | |
|---|---|---|
| **Internet** | Application layer | DNS, FTP, **HTTP**, IMAP, POP, SSH, Telnet, TLS/SSL, … |
| | Transport layer | TCP, UDP, … |
| | Internet layer | IP, ICMP, IPSec, … |
| | Link layer | PPP, MAC (Ethernet, DSL, ISDN, …), … |

# The Modern Web

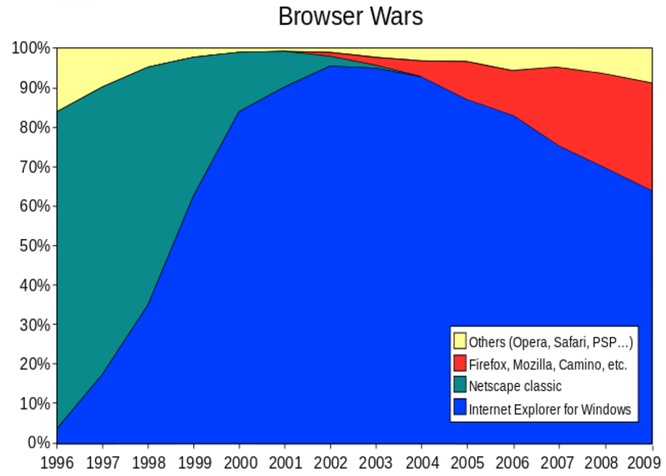
- Evolving computing architectures for organizing content and computation between browser (client) and web server
- 1990s: static web pages
- 1990s: server-side scripting (CGI, PHP, ASP, ColdFusion, JSP, ...)
- 2000s: single page apps (JQuery)
- 2010s: front-end frameworks (Angular, Aurelia, React, ...), microservices

# Static Web Pages

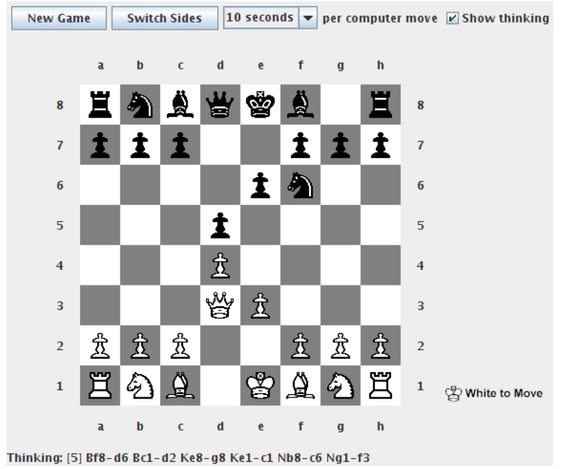
- URL corresponds to directory location on server
  - e.g. http://domainName.com/img/image5.jpg maps to img/image5.jpg file on server
- Server responds to HTTP request by returning requested files
- Advantages
  - Simple, easily cacheable, easily searchable
- Disadvantages
  - No interactivity

# Web 1.0 Problems

- At this point, most  sites were “read only”
- Lack of standards for advanced content - “browser war”
- No rich client content... the best you could hope for was a Java applet



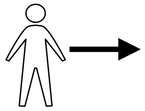

https://en.wikipedia.org/wiki/Browser_wars




https://en.wikipedia.org/wiki/Java_applet

# Dynamic Web Pages

High-level protocol built on TCP/IP that defines how data

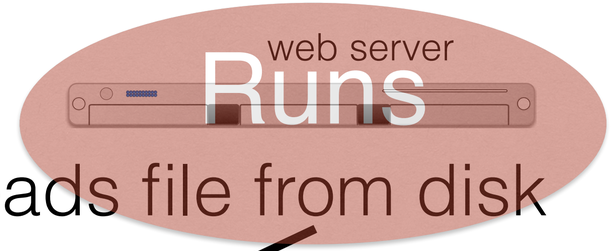



HTTP Request

**GET** /vinicius3w/if1007-Microservices/blob/master/README.md/ HTTP/1.1

Host: github.com

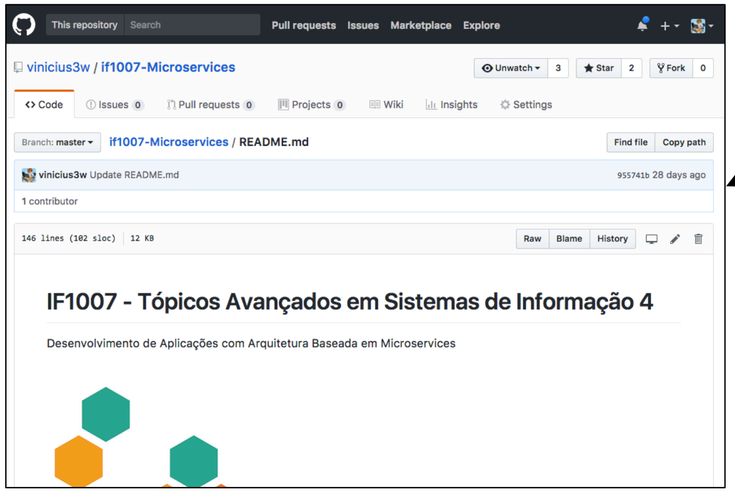
Accept: text/html



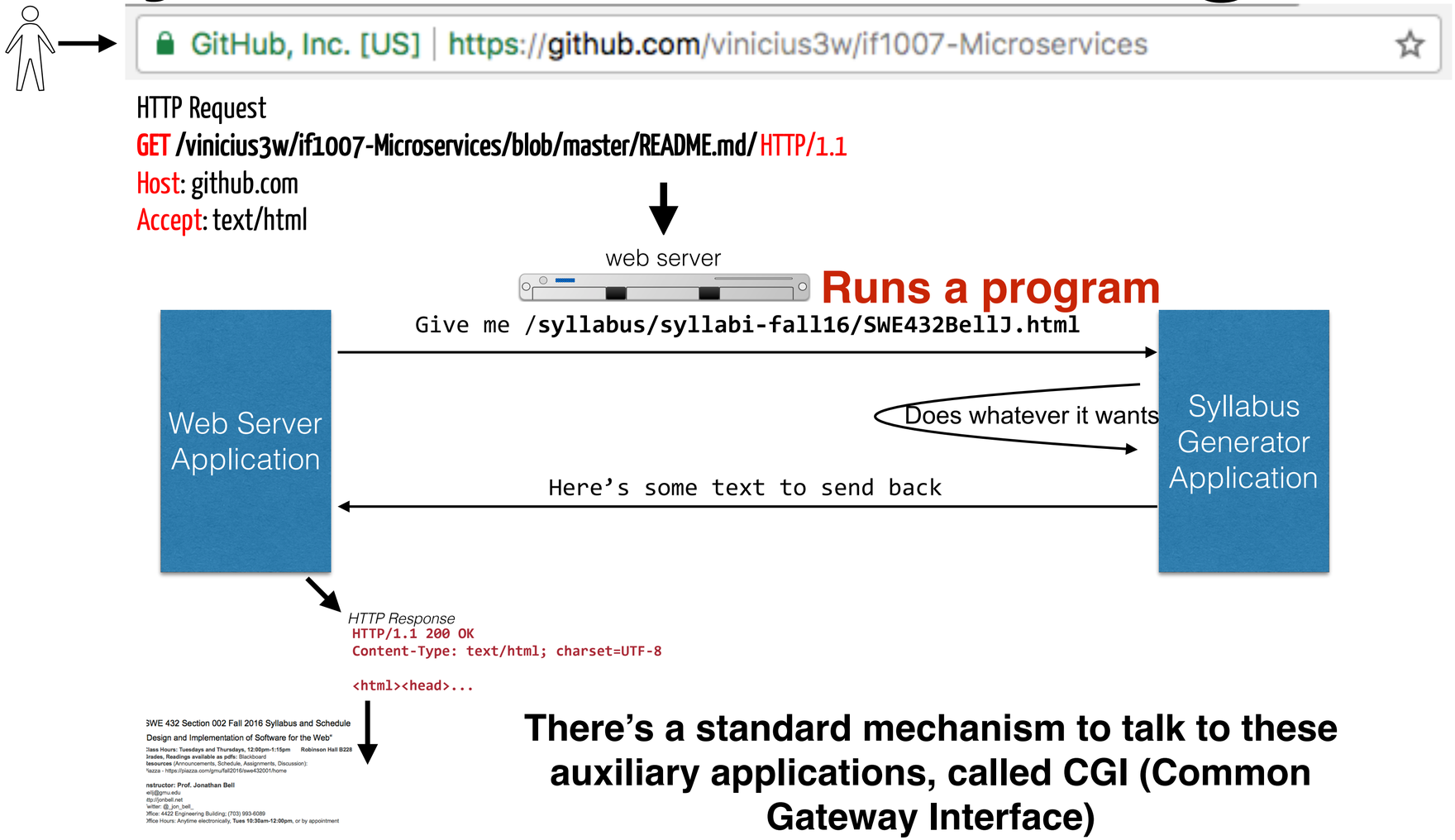

*HTTP Response*

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```



# Dynamic Web Pages




**There's a standard mechanism to talk to these auxiliary applications, called CGI (Common Gateway Interface)**

# Server Side Scripting

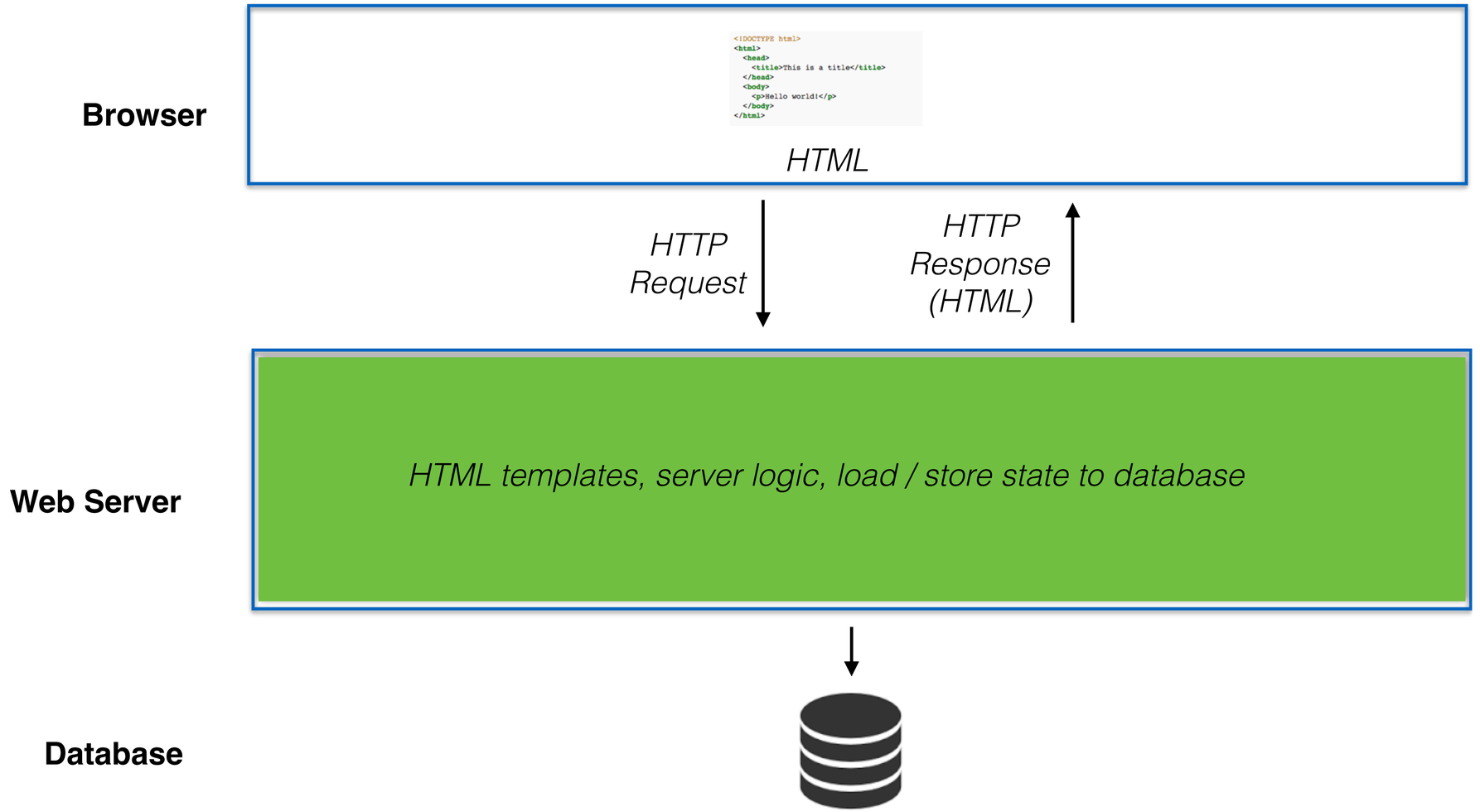
- Generate HTML on the server through scripts

```php
<!DOCTYPE html>
<html>
    <head>
        <title>PHP Test</title>
    </head>
    <body>
        <?php echo '<p>Hello World</p>'; ?>
    </body>
</html>
```

```jsp
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

- Early approaches emphasized embedding server
- Examples: CGI

# Server Side Scripting Site



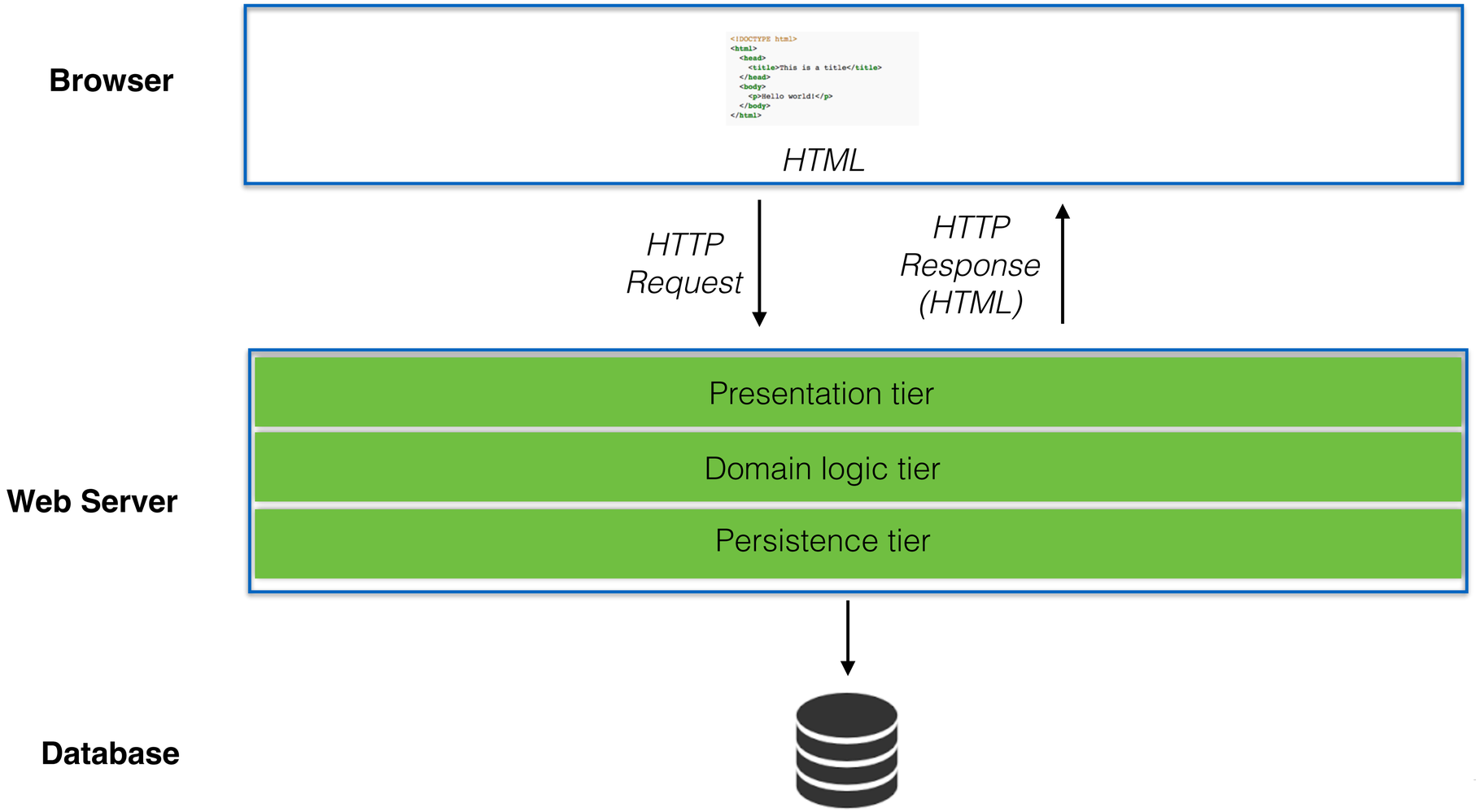
# Limitations

- Poor modularity
  - Code representing logic, database interactions, generating HTML presentation all tangled
  - Example of a Big Ball of Mud [http://www.laputan.org/mud]
  - Hard to understand, difficult to maintain
- Still a step up over static pages!

# Server Side Frameworks

- Framework that structures server into tiers, organizes logic into classes
- Create separate tiers for presentation, logic, persistence layer
- Can understand and reason about domain logic withou looking at presentation (and vice versa)
- Examples: ASP.NET, JSP

# Server Side Framework Site



# Limitations

- Need to load a whole new web page to get new data
  - Users must wait while new web page loads, decreasing responsiveness & interactivity
  - If server is slow or temporarily non-responsive, whole user interface hangs!
  - Page has a discernible refresh, where old content is replaced and new content appears rather than seamless transition

# Single Page Application (SPA)

- Client-side logic sends messages to server, receives response Logic is associated with a single HTML pages, written in Javascript
- HTML elements dynamically added and removed through DOM manipulation

```
<b>Projects:</b>
<ol id="new-projects"></ol>

<script>
$( "#new-projects" ).load( "/resources/load.html #projects li" );
</script>

</body>
</html>
```

- Processing that does not require server may occur entirely client side, dramatically increasing responsiveness & reducing needed server resources
- Classic example: Gmail

# SPA Enabling Technologies

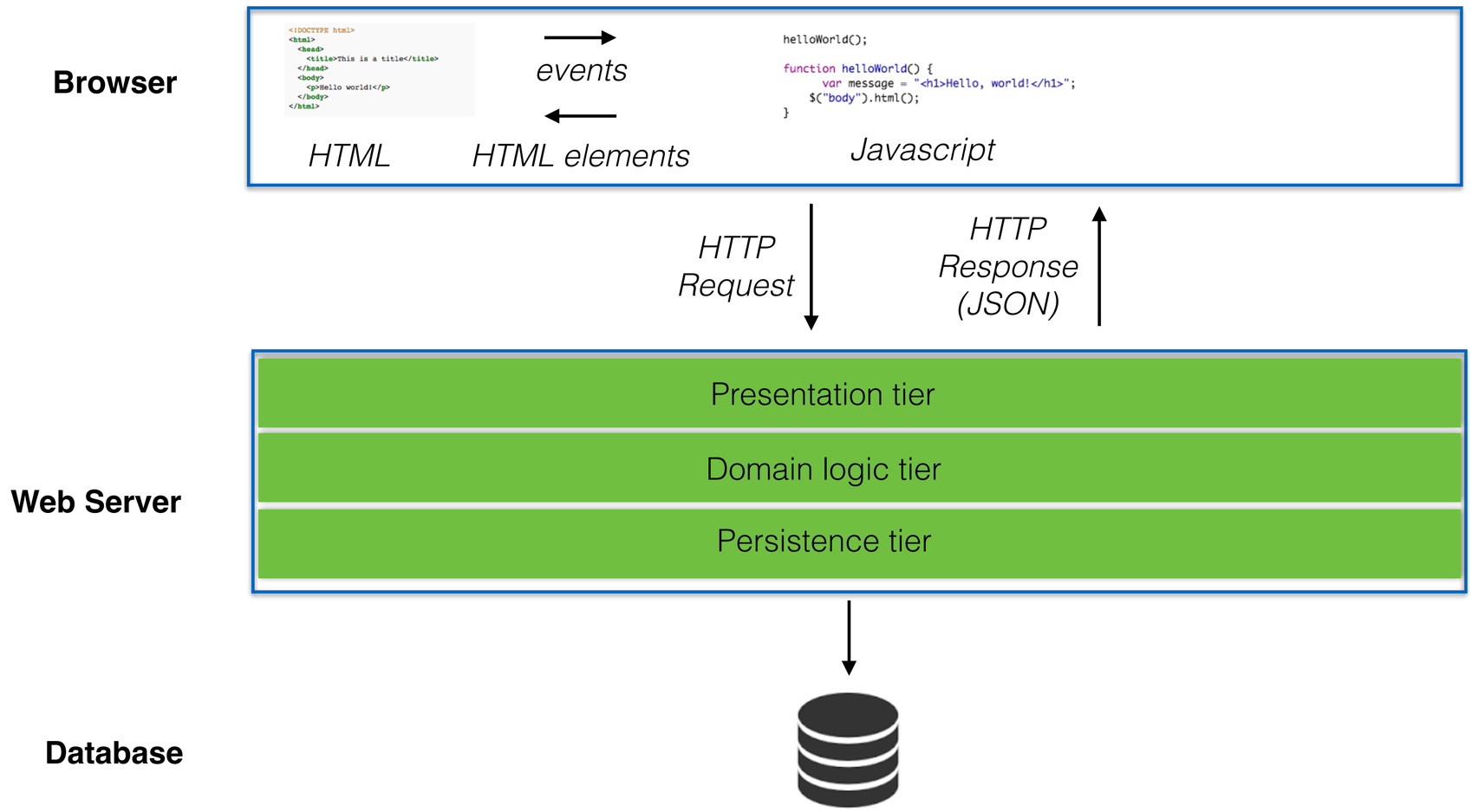
- AJAX: Asynchronous Javascript and XML
  - Set of technologies for sending asynchronous request from web page to server, receiving response
- DOM Manipulation
  - Methods for updating the HTML elements in a page after the page may already have loaded
- JSON: JavaScript Object Notation
  - Standard syntax for describing and transmitting Javascript data objects
- JQuery
  - Wrapper library built on HTML standards designed for AJAX and DOM manipulation

**JSON**

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

https://en.wikipedia.org/wiki/JSON

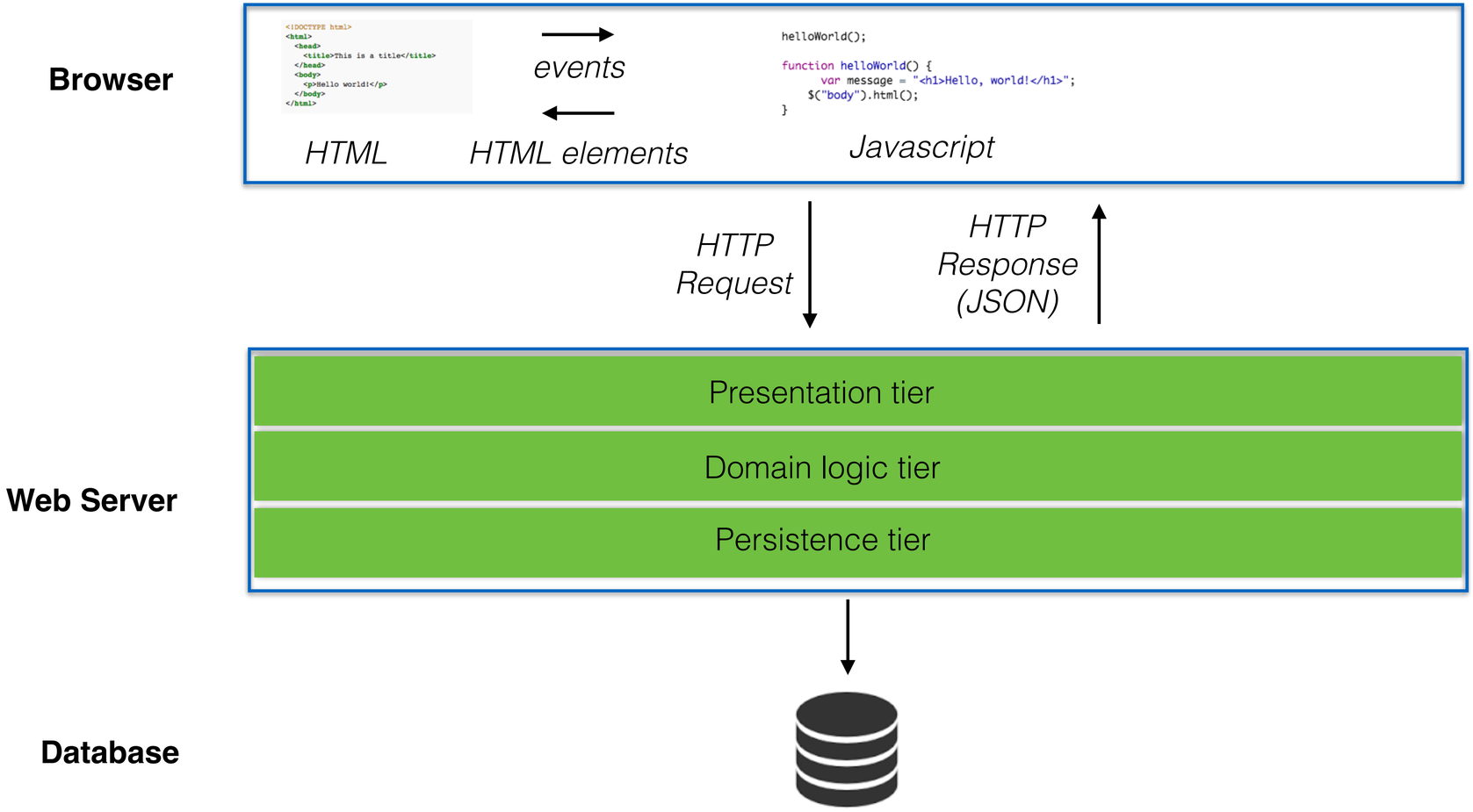
# Single Page Application Site



# Limitations

- Poor modularity client-side
  - As logic in client grows increasingly large and complex, becomes Big Ball of Mud
  - Hard to understand & maintain
  - DOM manipulation is brittle & tightly coupled, where small changes in HTML may cause unintended changes (e.g., two HTML elements with the same id)
  - Poor reuse: logic tightly coupled to individual HTML elements, leading to code duplication of similar functionality in many places

# Single Page Application Site



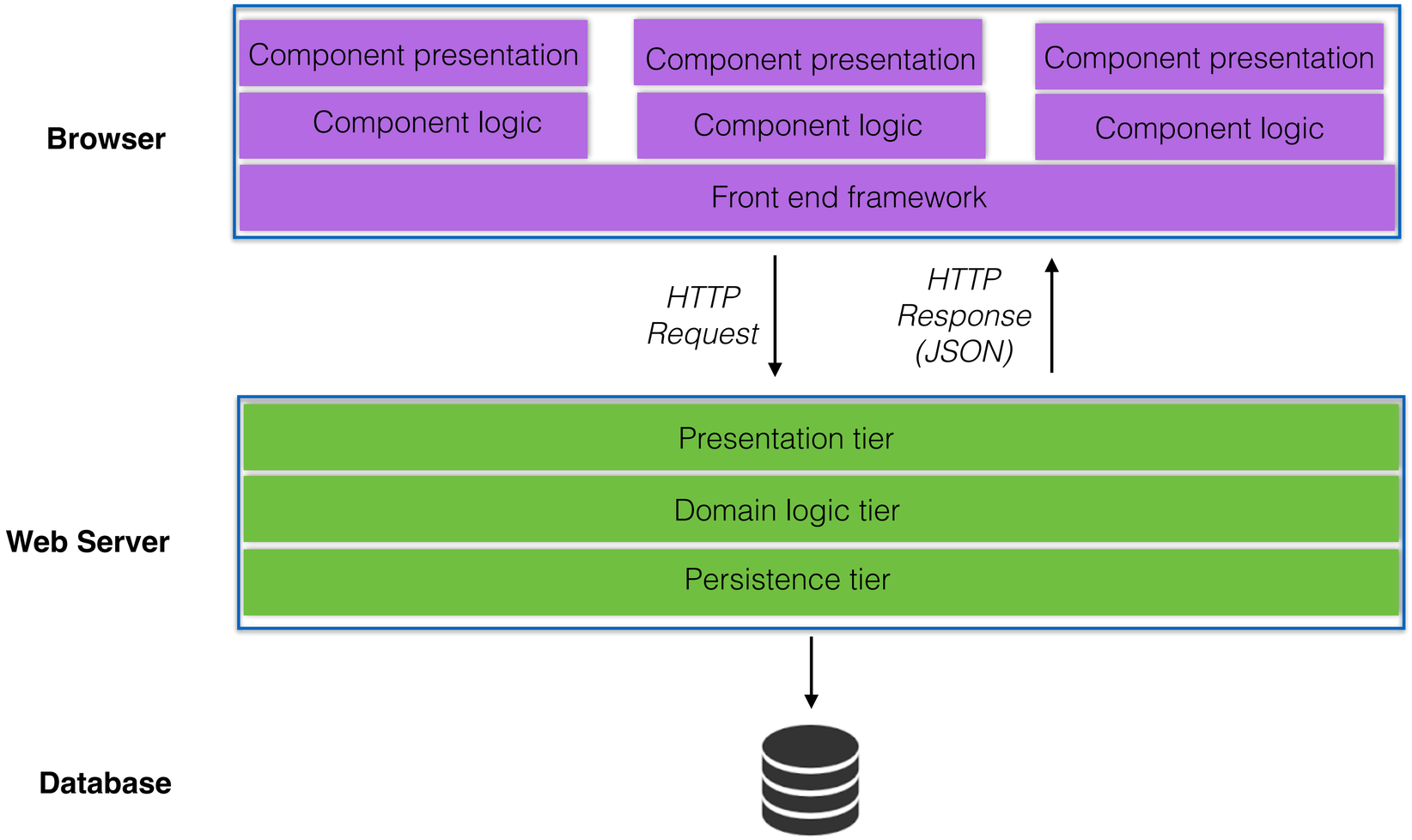
# Limitations

- Poor modularity client-side
  - As logic in client grows increasingly large and complex, becomes Big Ball of Mud
  - Hard to understand & maintain
  - DOM manipulation is brittle & tightly coupled, where small changes in HTML may cause unintended changes (e.g., two HTML elements with the same id)
  - Poor reuse: logic tightly coupled to individual HTML elements, leading to code duplication of similar functionality in many places

# Front End Frameworks

- Client is organized into separate components, capturing model of web application data
- Components are reusable, have encapsulation boundary (e.g., class)
- Components separate logic from presentation
- Components dynamically generate corresponding code based on component state
  - In contrast to HTML element manipulation, framework generates HTML, not user code, decreasing clouping
- Examples: Meteor, Ember, Angular, Aurelia, React

# Front End Framework Site



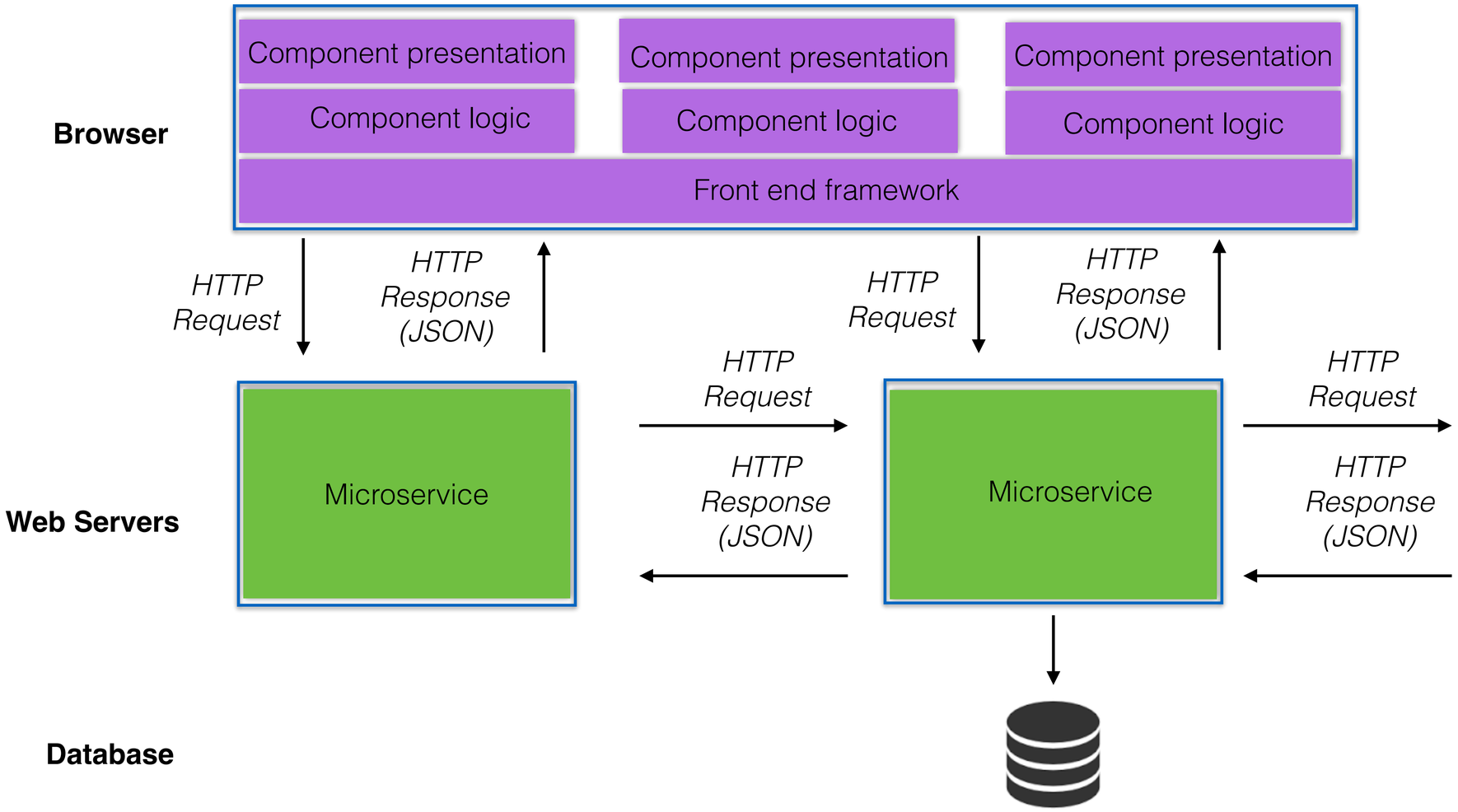
# Limitations

- Duplication of logic in client & server
  - As clients grow increasingly complex, must have logic in both client & server
  - May even need to be written twice in different languages! (e.g., Javascript, Java)
  - Server logic closely coupled to corresponding client logic. Changes to server logic require corresponding client logic change.
  - Difficult to reuse server logic

# Microservices

- Small, focused web server that communicates through data requests & responses
  - Focused only on logic, not presentation
- Organized around capabilities that can be reused in multiple context across multiple applications
- Rather than horizontally scale identical web servers, vertically scale server infrastructure into many, small focused servers

# Microservice Site



# Architectural Styles

- Architectural style specifies
  - how to partition a system
  - how components identify and communicate with each other
  - how information is communicated
  - how elements of a system can evolve independently

# Constant change in web architectural styles

- Key drivers
  - Maintainability (new ways to achieve better modularity)
  - Reuse (organizing code into modules)
  - Scalability (partitioning monolithic servers into services)
  - Responsiveness (movement of logic to client)
  - Versioning (support continuous roll-out of new features)
- Web standards have enabled many possible solutions
- Explored through many, many frameworks, libraries, and programming languages

# The web today

- Many technologies for each architectural style
  - Most support more than one
- Applications often evolve from one architectural style to another
  - Leads to applications combining multiple architectural styles
  - e.g., Single page app that uses server side scripting for a separate set of pages
- Newer architectural styles not always better
  - More complex, may be overkill for simple sites

# Philosophy of the Internet

- Decentralisation: No permission is needed from a central authority to post anything on the Web, there is no central controlling node, and so no single point of failure ... and no “kill switch”! This also implies freedom from indiscriminate censorship and surveillance.
- Non-discrimination: If I pay to connect to the internet with a certain quality of service, and you pay to connect with that or a greater quality of service, then we can both communicate at the same level. This principle of equity is also known as Net Neutrality.
- Bottom-up design: Instead of code being written and controlled by a small group of experts, it was developed in full view of everyone, encouraging maximum participation and experimentation.
- Universality: For anyone to be able to publish anything on the Web, all the computers involved have to speak the same languages to each other, no matter what different hardware people are using; where they live; or what cultural and political beliefs they have. In this way, the Web breaks down silos while still allowing diversity to flourish.
- Consensus: For universal standards to work, everyone had to agree to use them. Tim and others achieved this consensus by giving everyone a say in creating the standards, through a transparent, participatory process at W3C.

# Internet Governance

- IETF = Internet Engineering Task Force
- Open, all-volunteer organization
- Organized into working groups on specific topics
- Request for Comments
  - One of a series, begun in 1969, of numbered informational documents and standards followed by commercial software and freeware in the Internet and Unix communities
  - All Internet standards are recorded in RFCs

# Internet Governance

- World Wide Web Consortium (W3C)
- Defines data formats and usage conventions as well as Internet protocols relevant to Web
- Members pay fees depending on country, revenues and non-profit/for-profit status
- Otherwise organized similar to IETF, but writes “Recommendations” instead of “Requests for Comments”
- http://www.w3.org/

# Homework 1.1

- Basic course setup
  - Telegram channel: https://t.me/regcin
  - Properly setting up your Telegram profile by providing a picture will help the teaching staff learn your name. Upload a current headshot picture of you (not anyone else, not a cartoon picture of you, etc.) to your profile. Besides, make sure you have your first and last name as part of your profile.

# Homework 1.2

- Learning Git
  - Solve the first four levels in: http://pcottle.github.io/learnGitBranching/
    - Introduction Sequence (5%)
    - Ramping Up (5%)
    - Moving Work Around (5%)
    - A Mixed Bag (5%)
    - For extra credit, complete "Advanced Topics". (5%)
- For submission, you only need to demonstrate completing the levels, which can be done taking a screenshot. However, you should keep track of your solutions to help you remember how to solve these types of issues in the future, or recover if your progress gets lost.

# Homework 1.3

- Hooks
  - Create a local git repository (using git init) in a new directory. Create a "post-commit" file in .git/hooks/. Inside the file, create a command that will open a web page immediately after a commit is performed to that repo.
- Some hints:
  - http://stackoverflow.com/questions/8967902/why-do-you-need-to-put-bin-bash-at-the-beginning-of-a-script-file
  - chmod
  - start for windows, open for mac/linux
- In your solution, provide the content of  "post-commit". Finally, take a screencast, or a gif recording of the process. See details below.

# Submit

- Submit in our telegram channel, https://t.me/regcin, a MD file containing the following:
  - Complete slack profile by deadline (25).
  - Screenshot of completed git tutorial (25).
  - Hooks (25)
  - Screencast (25)
- For your screenshot embed in the markdown file of your LOGIN-HW1.md. Include a link to your screencast video/gif. Include your concept answers in your markdown file.