

Course Overview

SWE 432, Fall 2017

Design and Implementation of Software for the Web

Course Topics

- How do we organize, structure and share information?
- How to make web applications
 - JavaScript, front-end and back-end development, programming models, testing, performance, privacy, security, scalability, deployment, etc.
- How to make *usable* web applications
 - User-centered design, user studies, information visualization, visual design, etc.

Logistics

- No textbook, but required readings listed on course schedule. Material covered in readings (but not in lectures) may appear on quizzes and the final.
- Group-based homework; each assignment builds on the last
- Lab-style work included in many lectures (**bring your laptop**)
- Grading:
 - 50% Homeworks
 - Late policy. 24 hours late or less: lose 10%
 - HW assignments submitted more than 24 hours late will receive a zero.
 - 20% Quizzes (drop 3 lowest)
 - 30% Final Exam

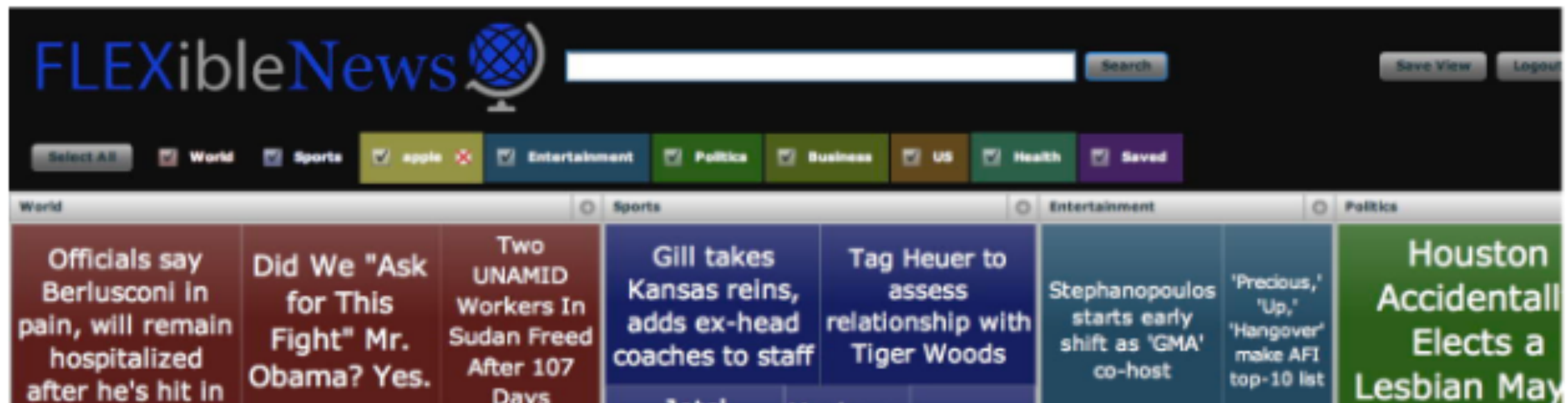
Plagiarism & Honor Code

“Just Don’t It”

- Do not work on homework with those not in your group
- Do not copy and paste large sections of your homework from third party sources
- Questions?

Project Overview

- Build a portfolio-worthy web application piece-by-piece
- Weekly deliverables follow class topics
- Will form two-person project groups
- Web app will be *dynamic*, use *web services*, and *information visualization*
- Example - News browser



Quizzes and exams

- Once a week: short quiz reviewing last week's material. We'll drop the 3 lowest.
- No midterm!

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

Perspectives in web development

Twitter, Inc. [US] | <https://twitter.com>

Home Moments Notifications Messages Search Twitter Tweet

Thomas LaToza @ThomasLaToza
Tweets 88 Following 292 Followers 230

Who to follow · Refresh · View all

- Followed by michaelhilton and others
Sahar Kokaly @saharkokaly [Follow](#)
- Followed by Mike Godfrey and others
Aiko Yamashita @aikoch... [Follow](#)
- Followed by michaelhilton and others
SebastianoPanichella @s... [Follow](#)

[Find people you know](#)

Live video · Hide

What's happening?

ProgrammableWeb @programmableweb · 1m
Why You Should Use Access Tokens to Secure an API bit.ly/2iBscvI

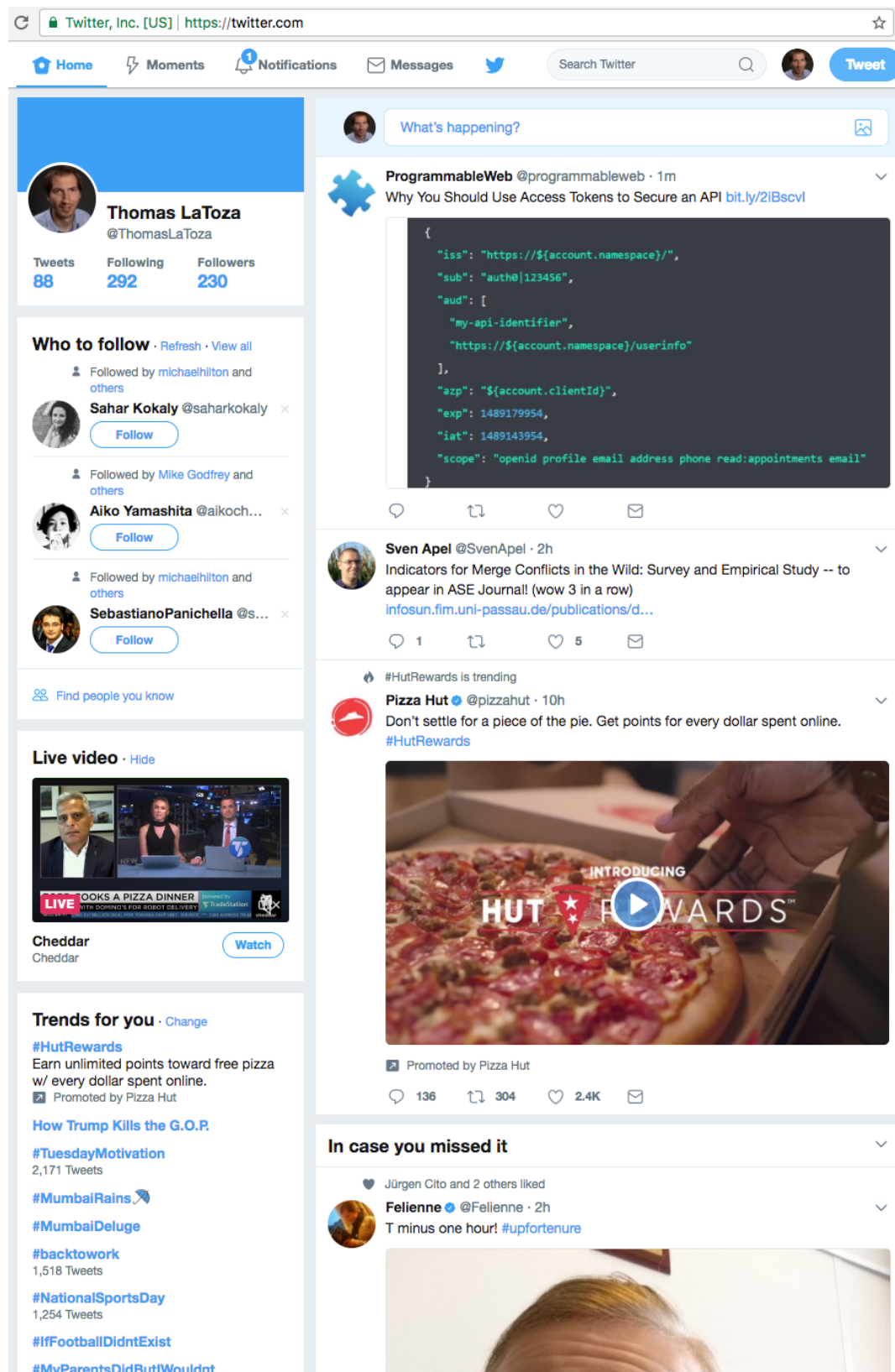
```
{
  "iss": "https://${account.namespace}/",
  "sub": "auth0|123456",
  "aud": [
    "my-api-identifier",
    "https://${account.namespace}/userinfo"
  ],
  "azp": "${account.clientId}",
  "exp": 1489179954,
  "iat": 1489143954,
  "scope": "openid profile email address phone read:appointments email"
}
```

Sven Apel @SvenApel · 2h
Indicators for Merge Conflicts in the Wild: Survey and Empirical Study -- to appear in ASE Journal! (wow 3 in a row)
infosun.fim.uni-passau.de/publications/d...

#HutRewards is trending

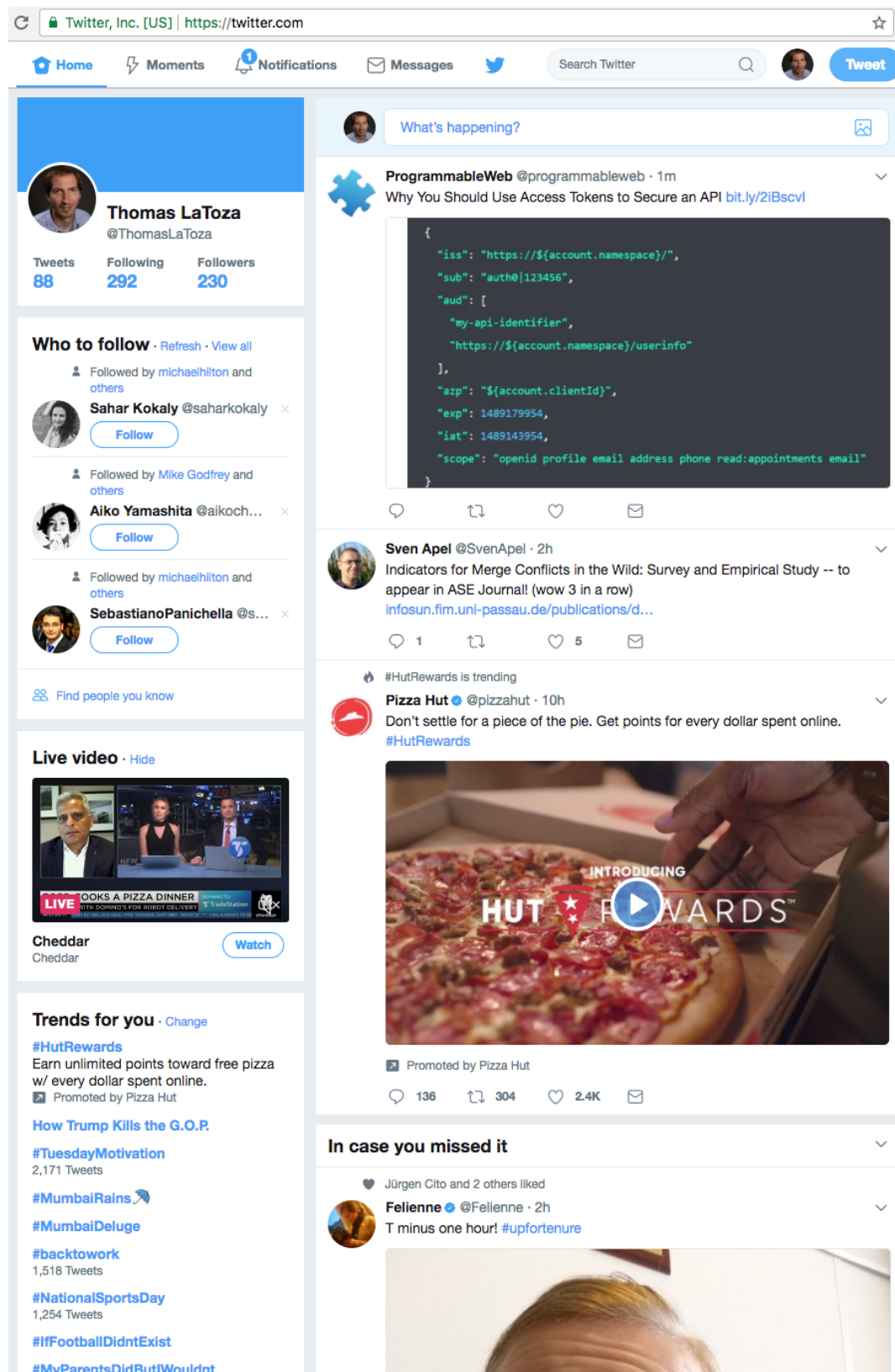
Pizza Hut @pizzahut · 10h
Don't settle for a piece of the pie. Get points for every dollar spent online.
[#HutRewards](#)

Systems Perspective



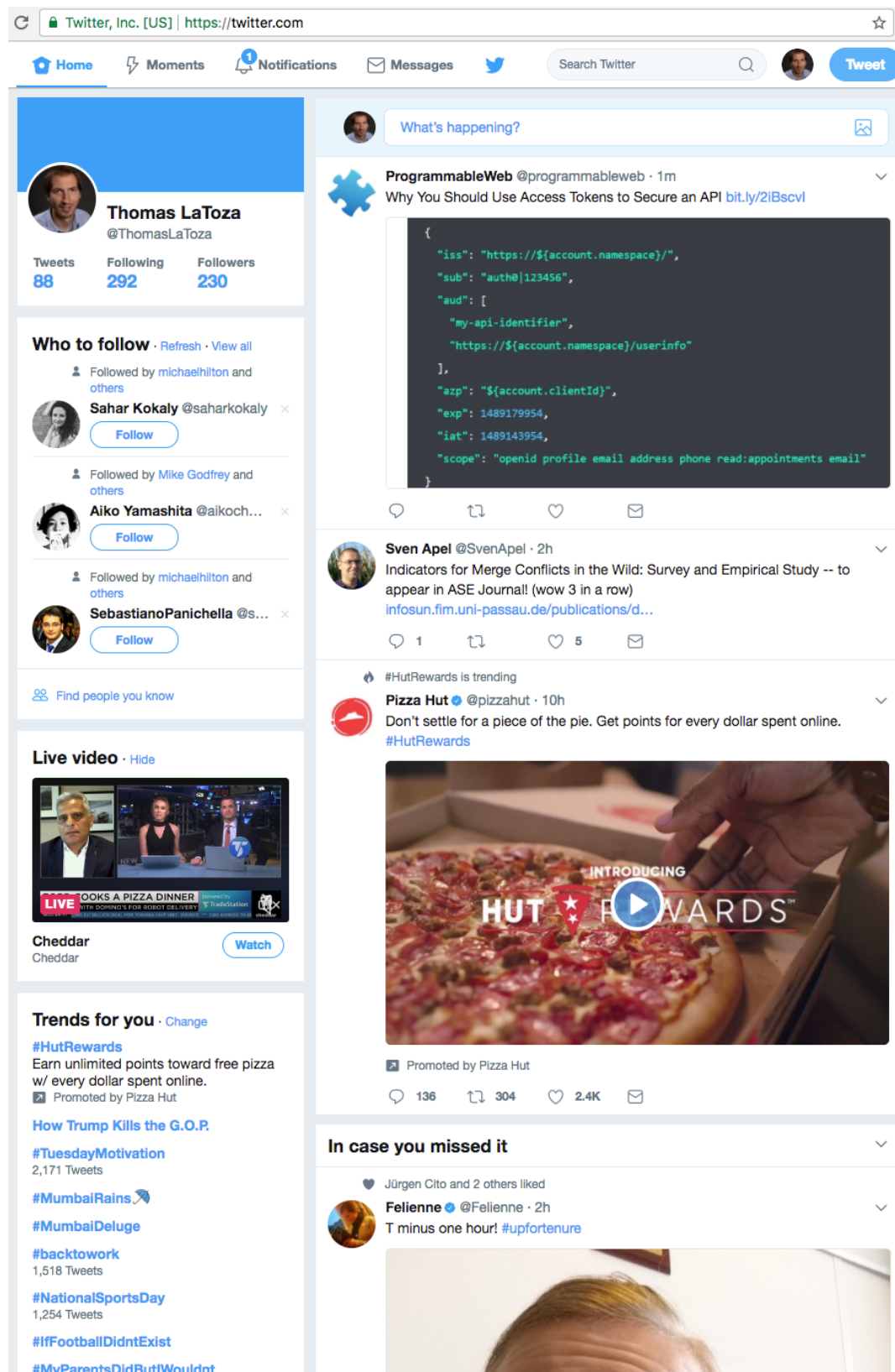
- How can we design **robust**, **efficient**, & **secure** interactions between computers?
- Individual web app may run on
 - Thousands of servers
 - Owned and managed by different orgs
 - *Millions* of clients
 - $>TBs$ of constantly changing data
- What happens when a server crashes?
- How do we prevent a malicious user from accessing user data on a server?

Software Engineering Perspective



- How can we design for **change** & **reuse**?
- Individual web app may
 - *Hundreds* of developers
 - *Millions* of lines of code
- New updates deployed many times a day
- Much functionality reused from code built by other organizations
- Offer API that allows other web apps to be built on top of it
- How can a developer successfully make a change without understanding the whole system?
- What happens when a new developer joins?

Human-Computer Interaction (HCI) Perspective



- How can we design web apps that are **usable** for their intended purpose?
- Individual web app may
 - *Millions of users*
 - Tens of different needs
- What happens when a new user interacts with the web app?
- How can we make a web app less frustrating to use?

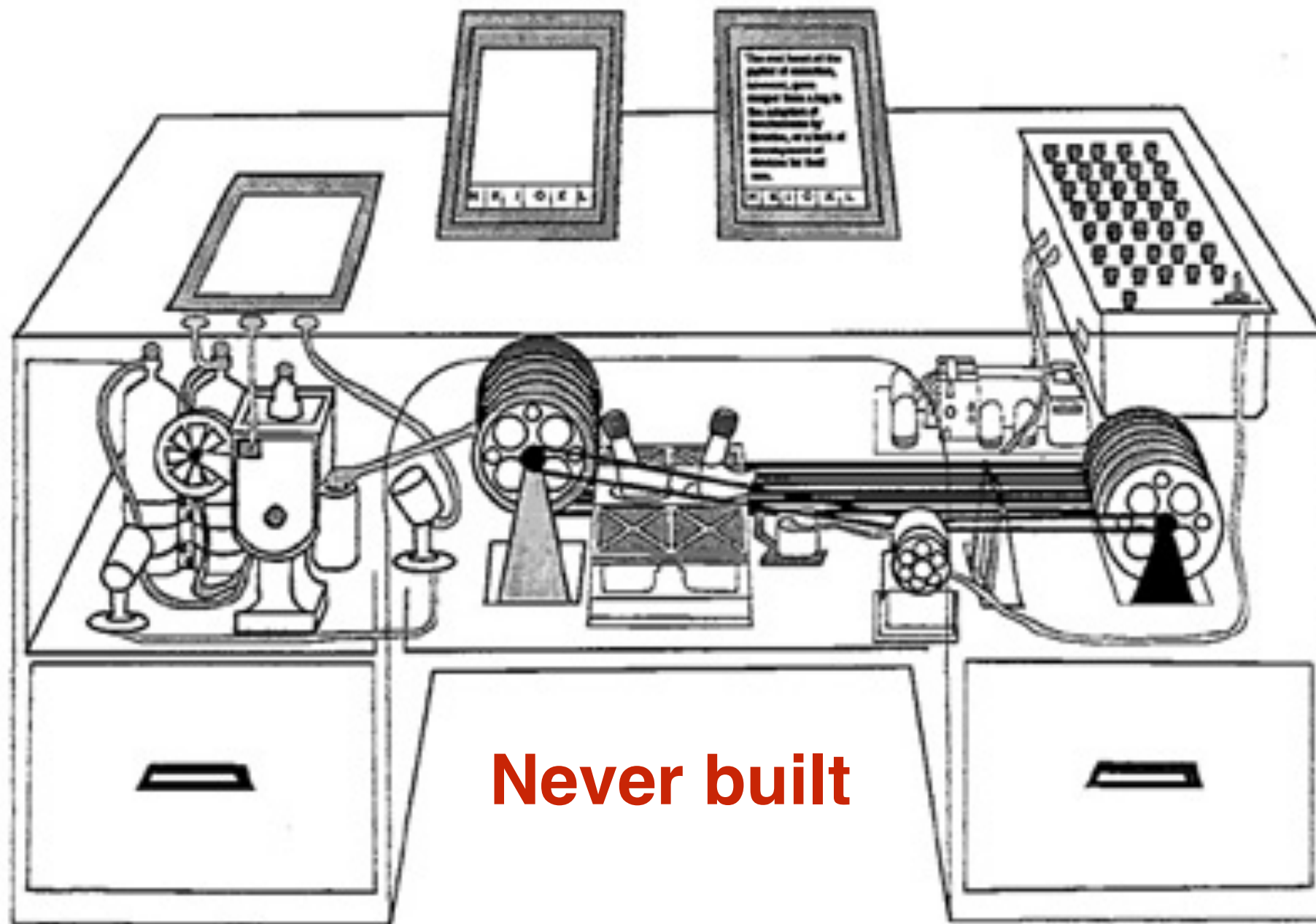
Pre-Web

- “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

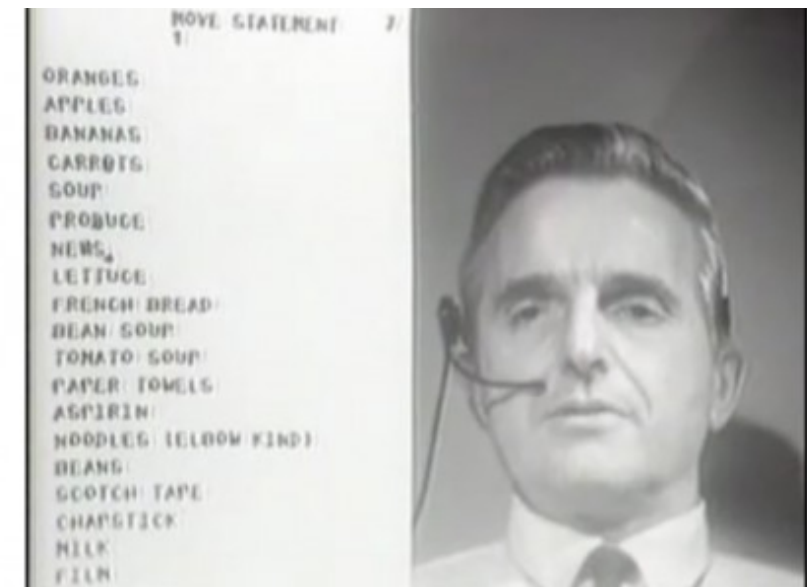
- MEMEX = MEMory EXtension
- Create and follow “associative trails” (links) and annotations between microfilm documents
- Technically based on “rapid selectors” Bush built in 1930’s to search microfilm
- Conceptually based on human associative memory rather than indexing

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

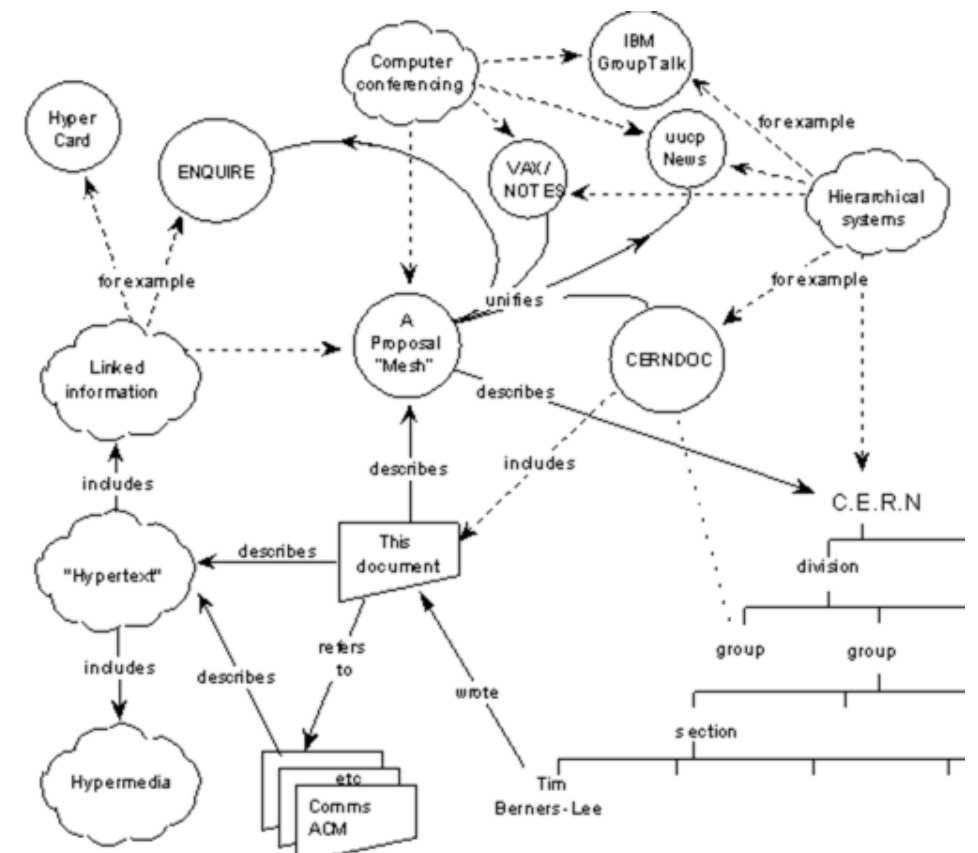


Origin of the Web

- 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!)



© CERN



Early Browsers

```
CERN Welcome

CERN

The European Laboratory for Particle Physics, located near Geneva[1] in
Switzerland[2] and France[3]. Also the birthplace of the World-Wide
Web[4].

This is the CERN laboratory main server. The support team provides a set of
Services[5] to the physics experiments and the lab. For questions and
suggestions, see WWW Support Contacts[6] at CERN

-----
About the Laboratory[7] - Hot News[8] - Activities[9] - About Physics[10] -
Other Subjects[11] - Search[12]
-----

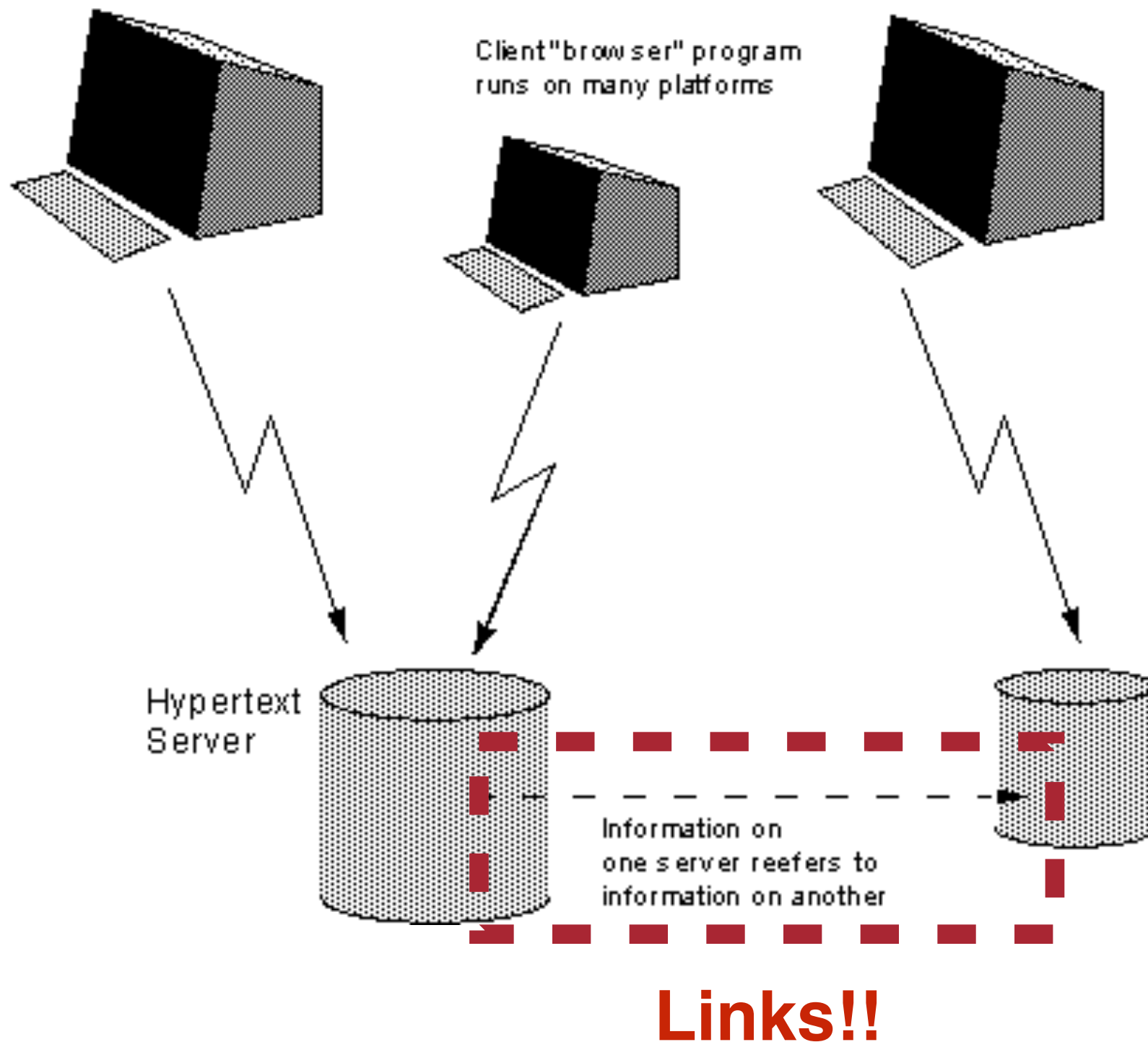
About the Laboratory

Help[13] and General information[14], divisions, groups and
activities[15] (structure), Scientific committees[16]

Directories[17] (phone & email, services & people), Scientific
Information Service[18] (library, archives or Alice), Preprint[19] Server

1-45, Back, Up, <RETURN> for more, Quit, or Help: █
```

Original WWW Architecture



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”

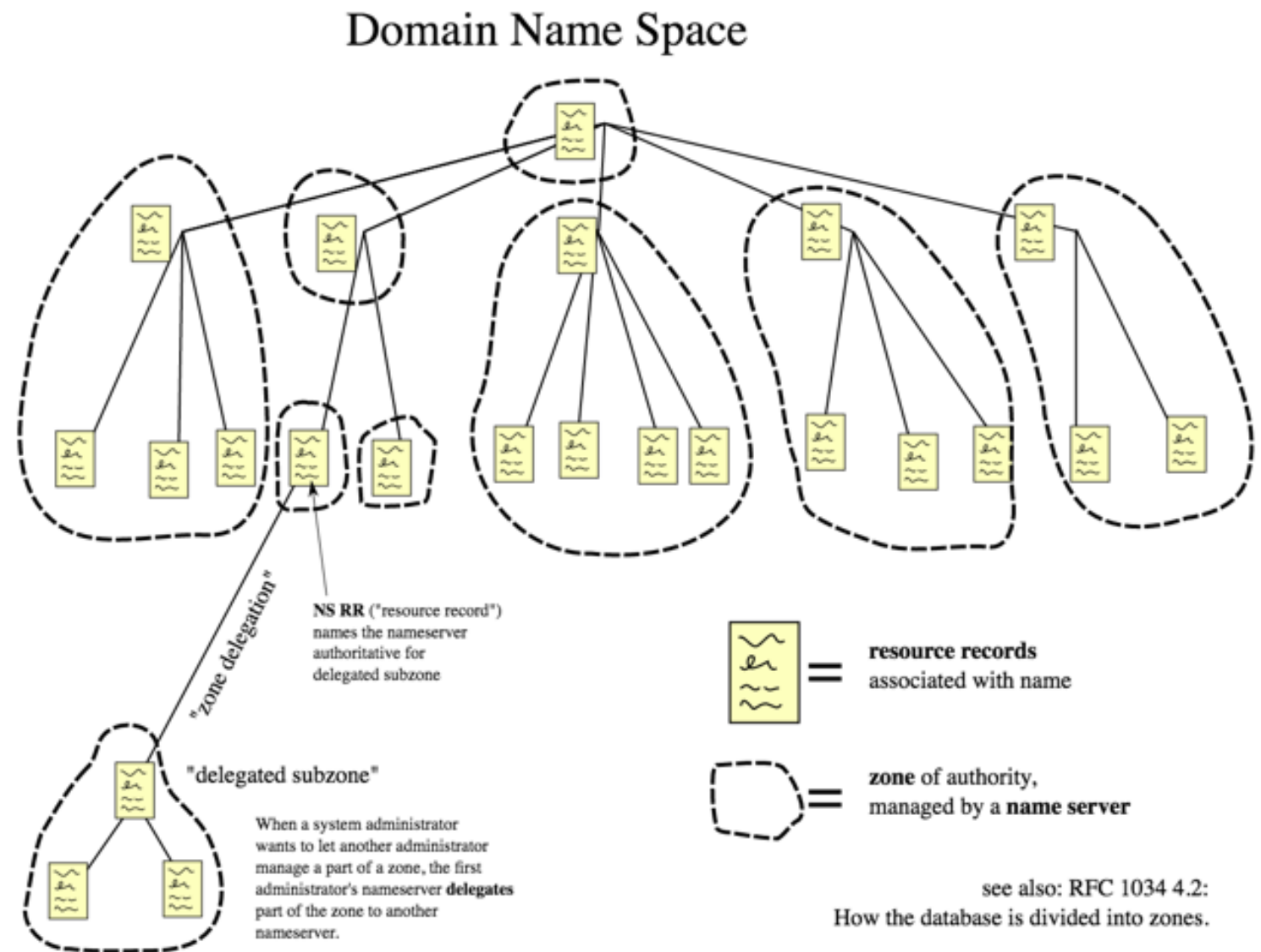
May be host name or an IP address
Optional port name (e.g., :80 for port 80)

↑
“Request syllabus/syllabi-fall16/SWE432BellJ.html”

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

DNS: Domain Name System

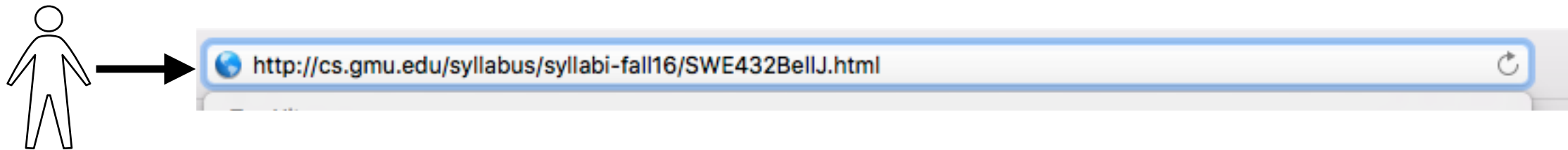
- Domain name system (DNS) (~1982)
- Mapping from names to IP addresses
- E.g. cs.gmu.edu -> 129.174.125.139



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 transferred on the web



HTTP Request

GET /syllabus/syllabi-fall16/SWE432BellJ.html **HTTP/1.1**

Host: cs.gmu.edu

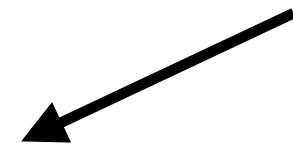
Accept: text/html



web server



Reads file from disk

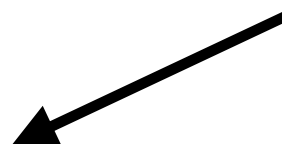


HTTP Response

HTTP/1.1 200 OK

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

<html><head>...



SWE 432 Section 002 Fall 2016 Syllabus and Schedule

"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:15pm Robinson Hall B228

Grades, Readings available as pdfs: Blackboard

Resources (Announcements, Schedule, Assignments, Discussion):

Piazza - <https://piazza.com/gmu/fall2016/swe432001/home>

Instructor: Prof. Jonathan Bell

bellj@gmu.edu

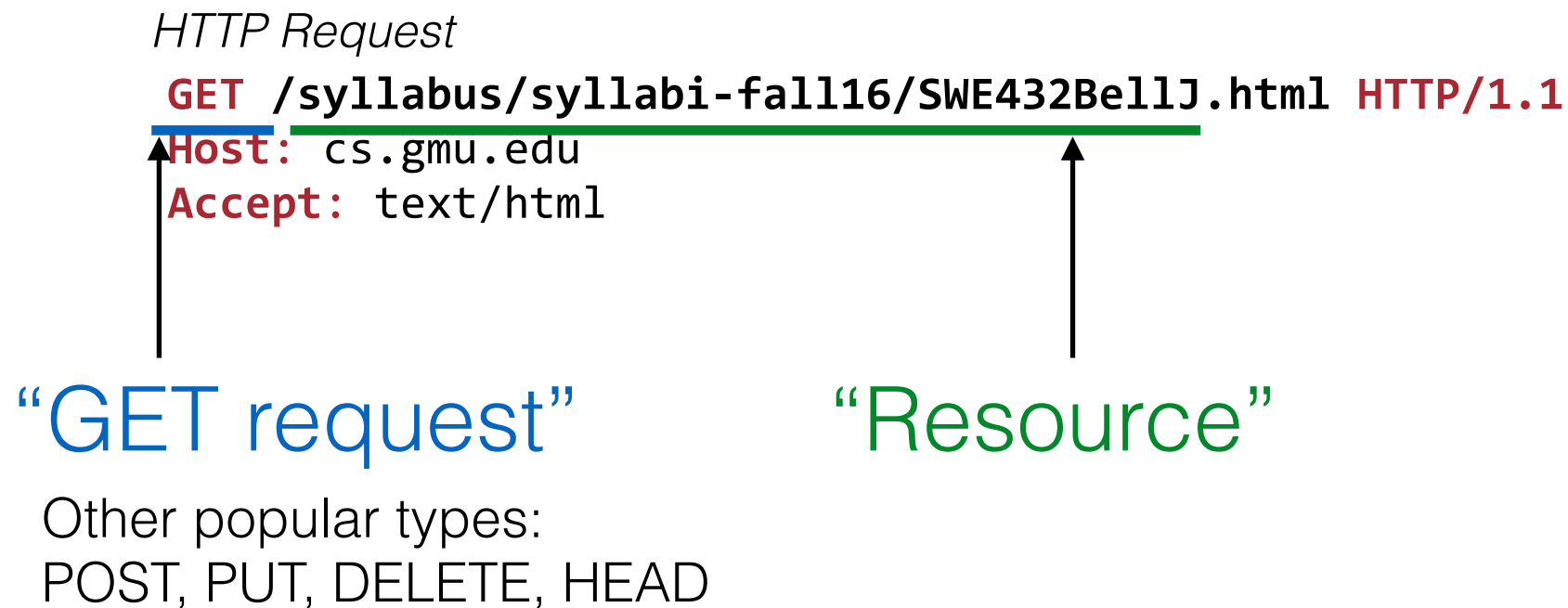
<http://jonbell.net>

Twitter: @_jon_bell_

Office: 4422 Engineering Building; (703) 993-6089

Office Hours: Anytime electronically, Tues 10:30am-12:00pm, or by appointment

HTTP Requests



- 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

“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

```
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>
```

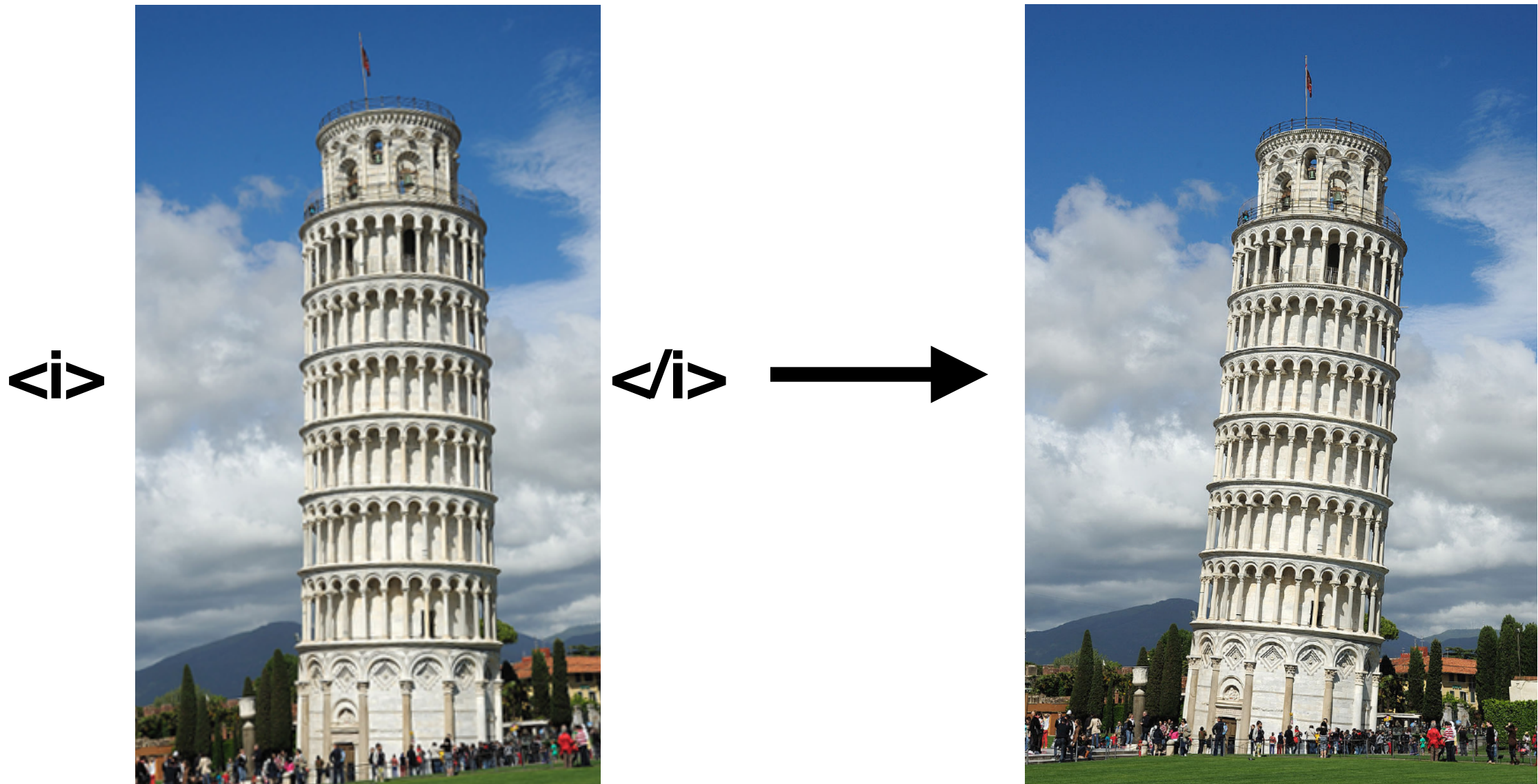
[HTML data]

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 document



HTML: HyperText Markup Language

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

`This text is bold!`



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

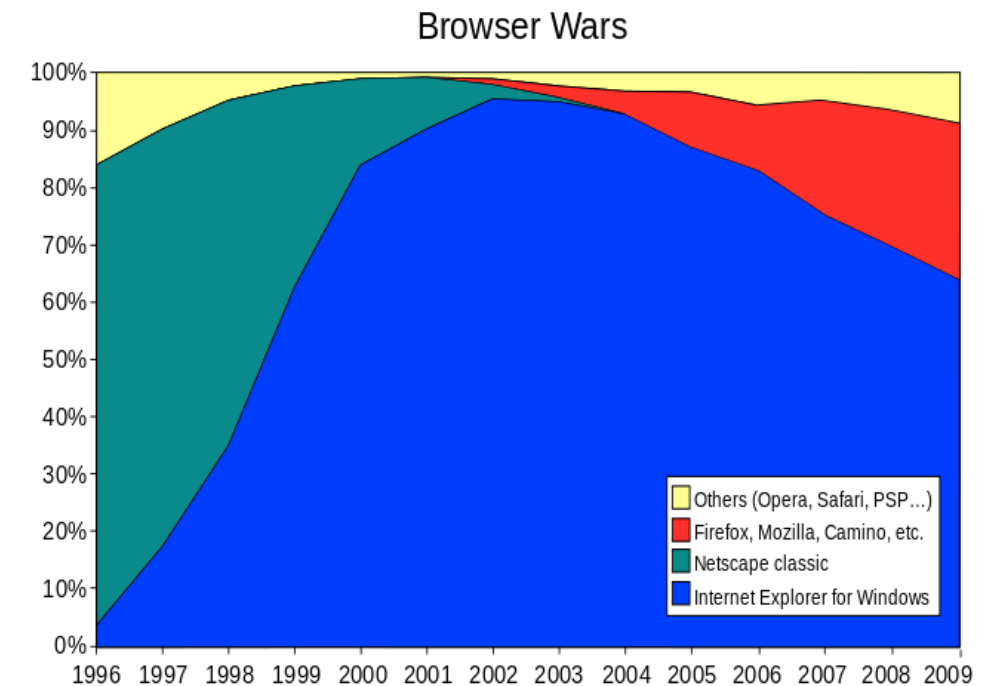
- Evolving competing architectures for organizing content and computation between browser (client) and web server
- 1990s: static web pages
- 1990s: server-side scripting (CGI, PHP, ASP, ColdFusion, JSP, ...)
- 2000s: single page apps (jQuery)
- 2010s: front-end frameworks (Angular, Aurelia, React, ...),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

Static Web Pages

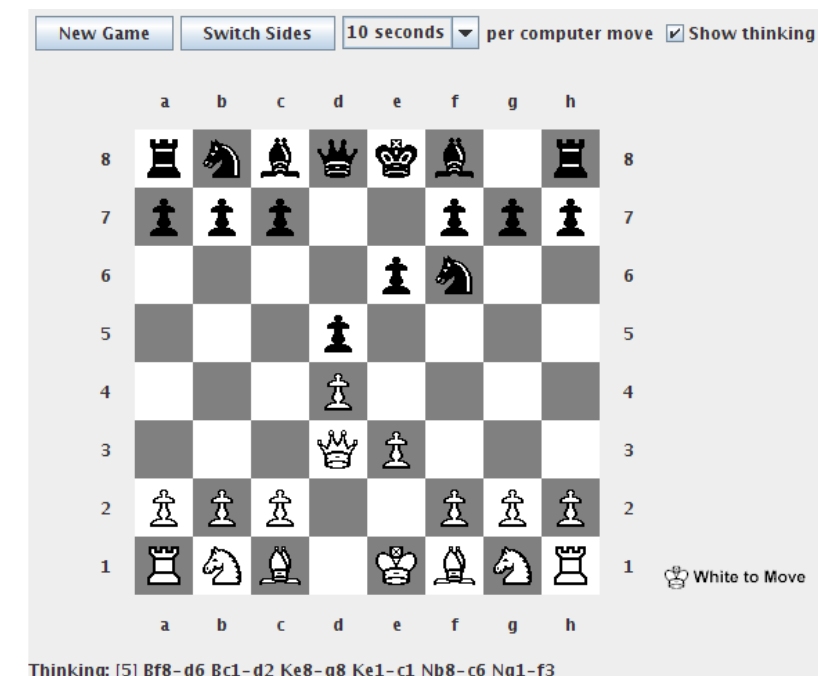
- URL corresponds to directory location on server
 - e.g. `http://domainName.com/img/image5.jpg` maps to `img/image5.jpg` file on server
- Server responds to HTTP request by returning requested files
- Advantages
 - Simple, easily cacheable, easily searchable
- Disadvantages
 - No interactivity

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

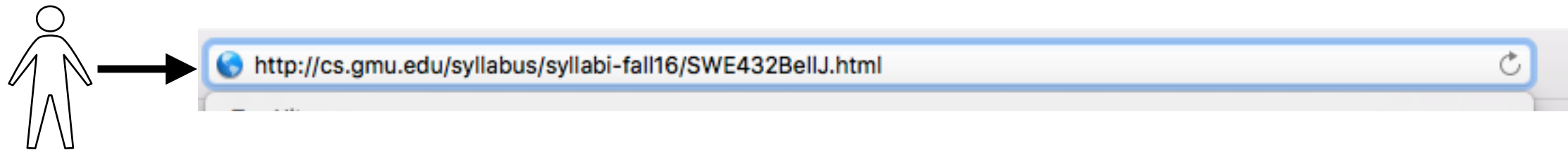


https://en.wikipedia.org/wiki/Browser_wars



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

Dynamic Web Pages



HTTP Request

GET /syllabus/syllabi-fall16/SWE432BellJ.html **HTTP/1.1**

Host: cs.gmu.edu

Accept: text/html



web server

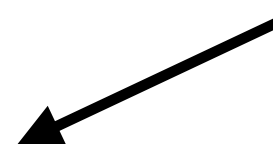


Runs a program
Reads file from disk

HTTP Response

HTTP/1.1 200 OK

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



<html><head>...

SWE 432 Section 002 Fall 2016 Syllabus and Schedule

"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:15pm Robinson Hall B228

Grades, Readings available as pdfs: Blackboard

Resources (Announcements, Schedule, Assignments, Discussion):

Piazza - <https://piazza.com/gmu/fall2016/swe432001/home>

Instructor: Prof. Jonathan Bell

bellj@gmu.edu

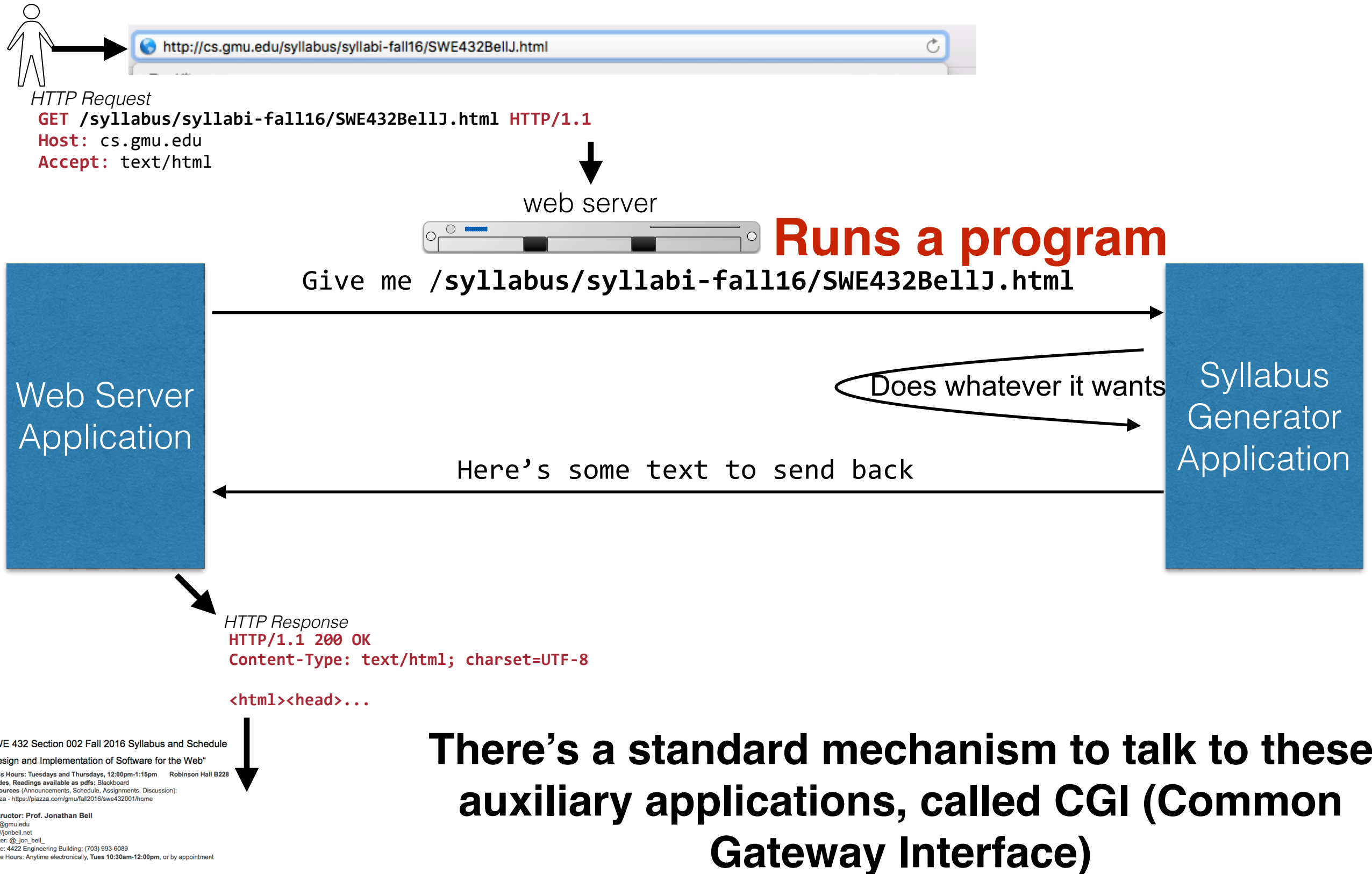
<http://jonbell.net>

Twitter: @_jon_bell_

Office: 4422 Engineering Building; (703) 993-6089

Office Hours: Anytime electronically, **Tues 10:30am-12:00pm**, or by appointment

Dynamic Web Pages



Server Side Scripting

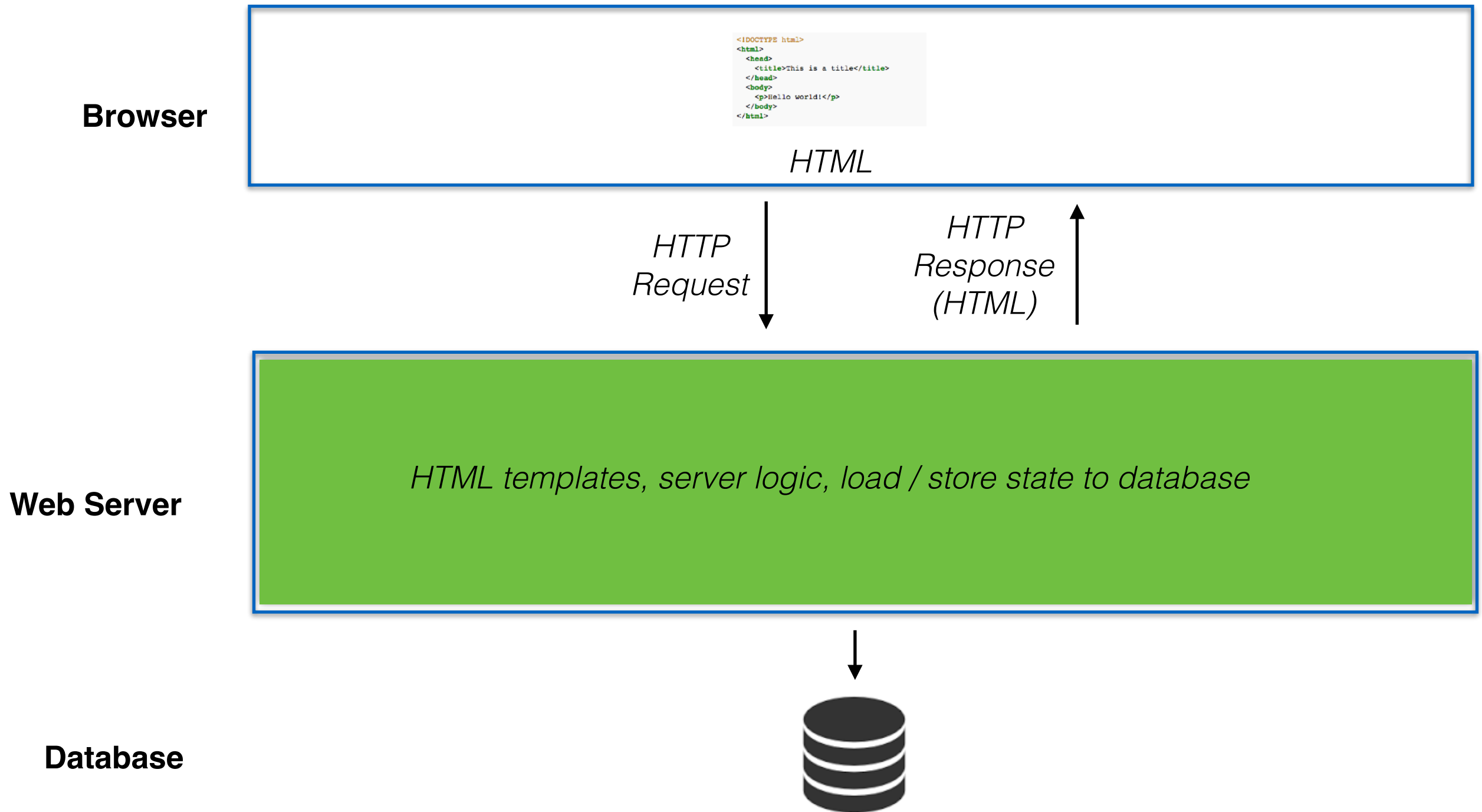
- Generate HTML on the server through scripts

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

```
<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 code *inside* html pages
- Examples: CGI

Server Side Scripting Site



Limitations

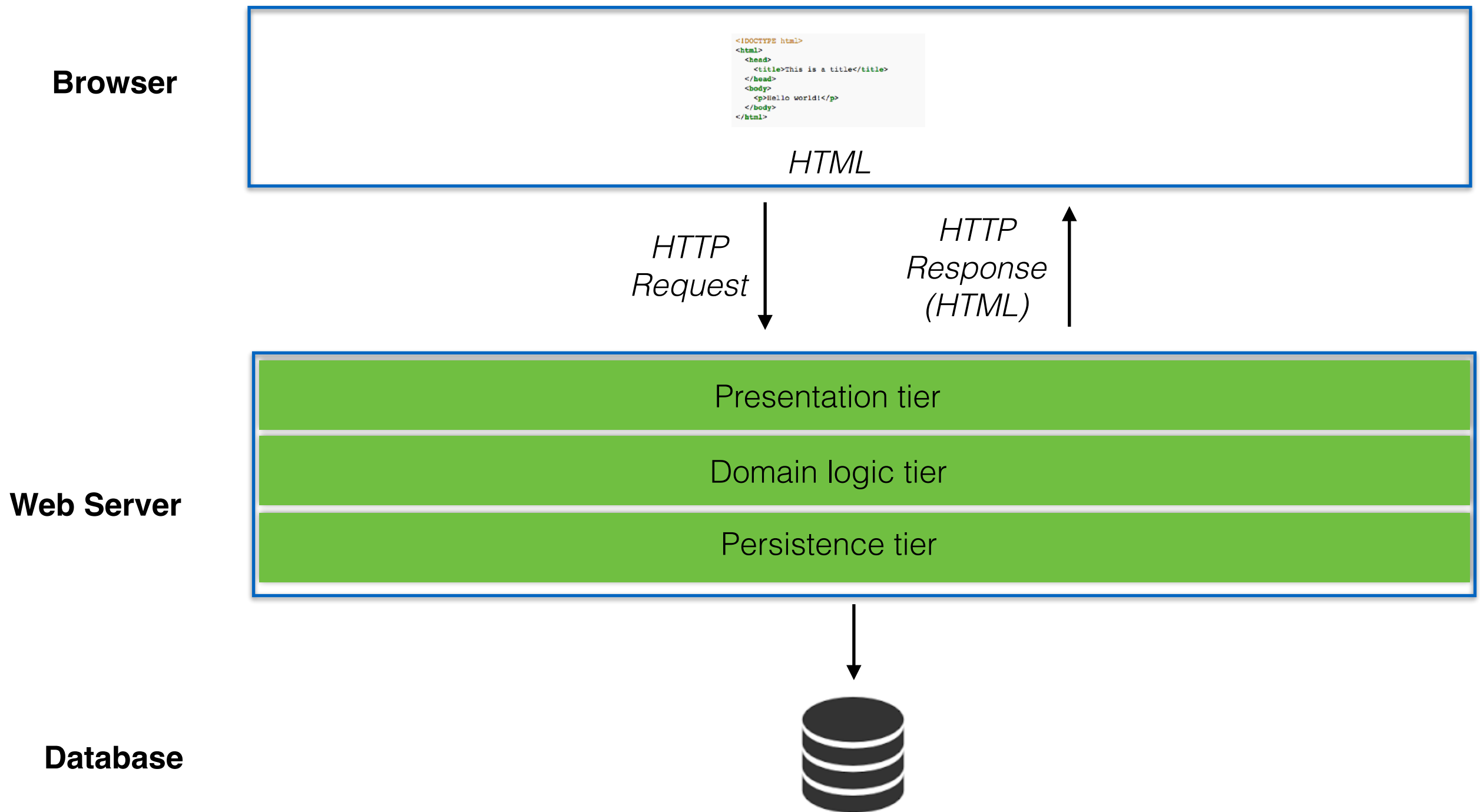
- Poor **modularity**
 - Code representing logic, database interactions, generating HTML presentation all tangled
 - Example of a Big Ball of Mud [1]
 - Hard to understand, difficult to maintain
- Still a step up over static pages!

[1] <http://www.laputan.org/mud/>

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 without 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

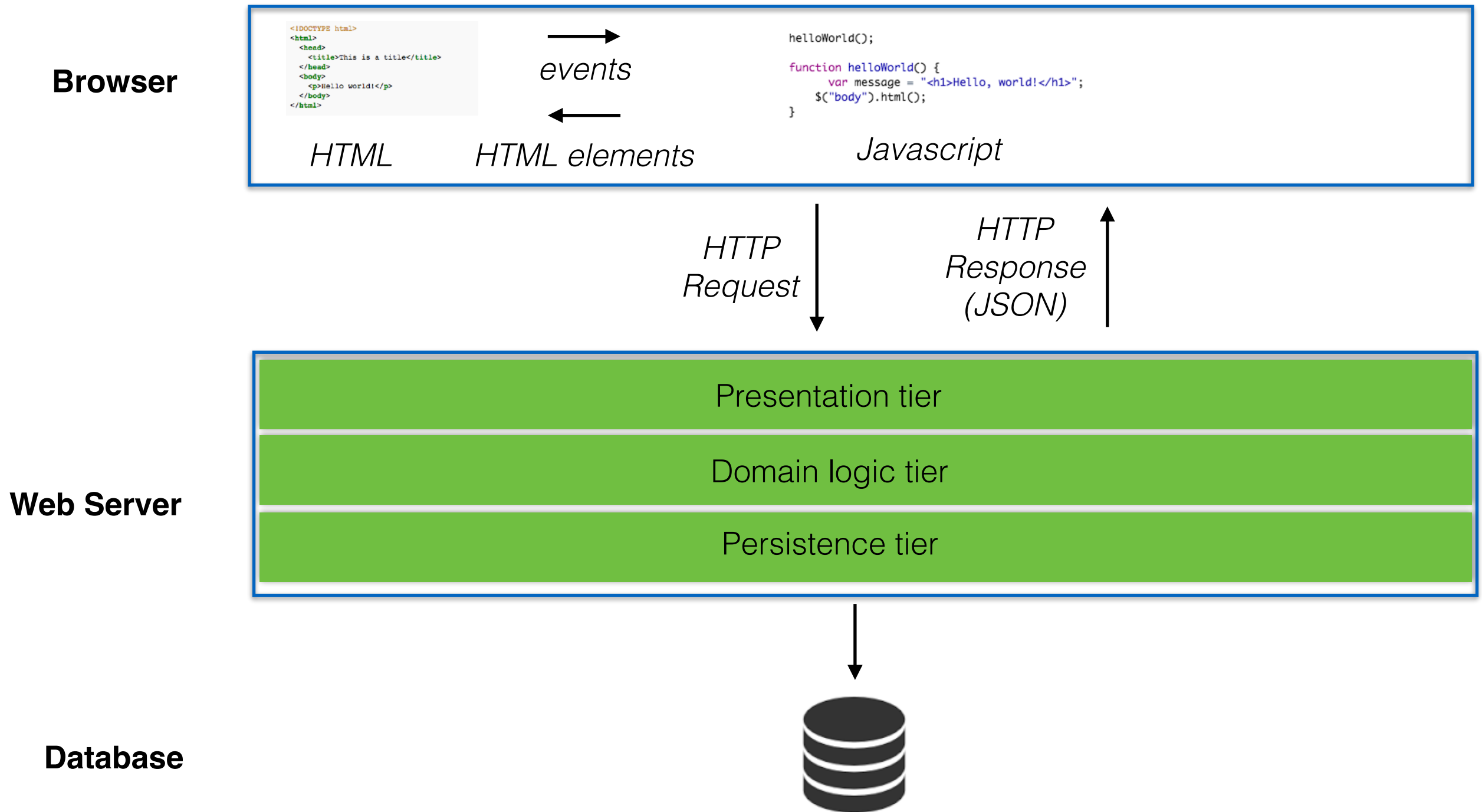
- AJAX: Asynchronous Javascript and XML
 - Set of technologies for sending asynchronous request from web page to server, receiving response
- DOM Manipulation
 - Methods for updating the HTML elements in a page *after* the page may already have loaded
- JSON: JavaScript Object Notation
 - Standard syntax 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



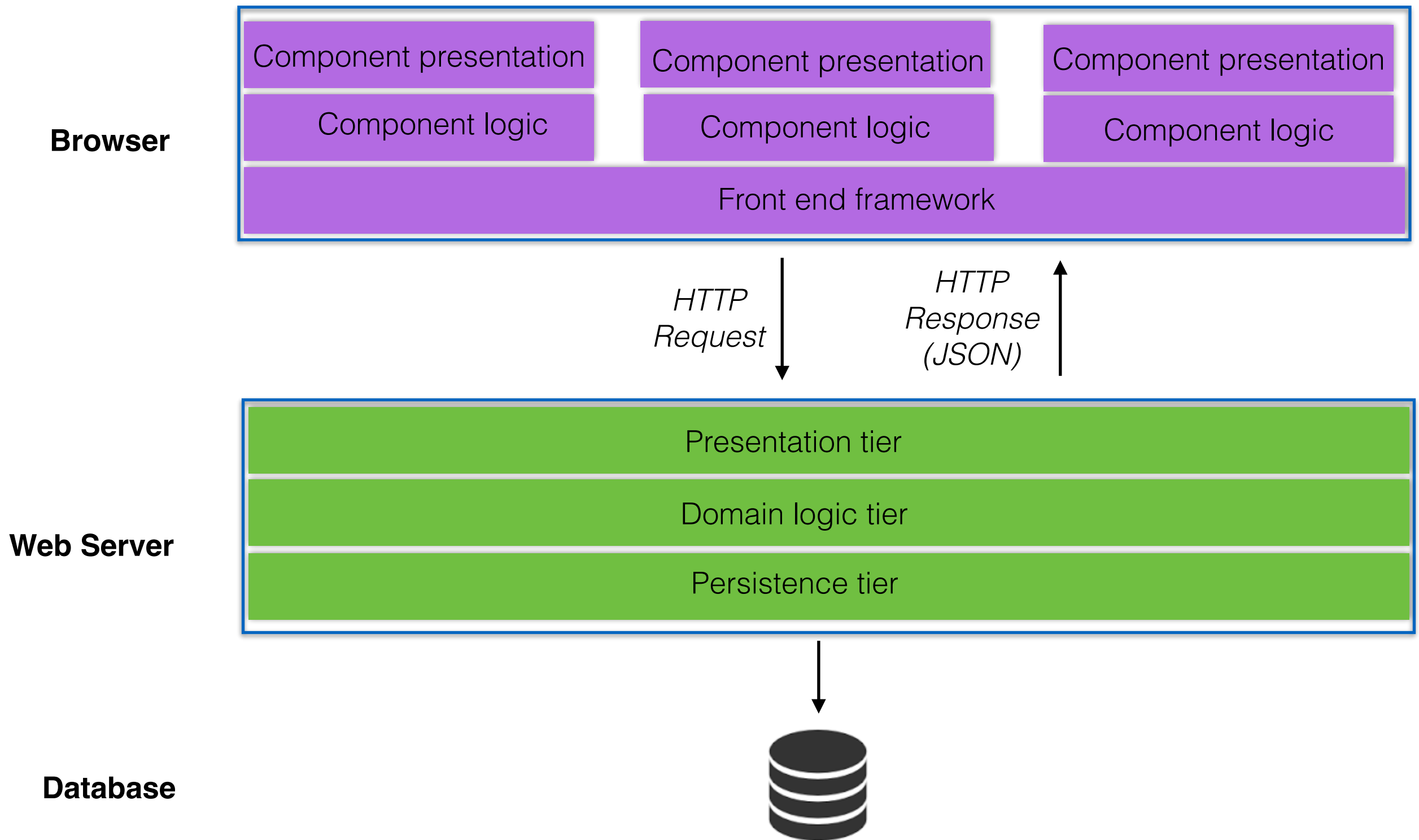
Limitations

- 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

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 coupling
- Examples: Meteor, Ember, Angular, Aurelia, React

Front End Framework Site



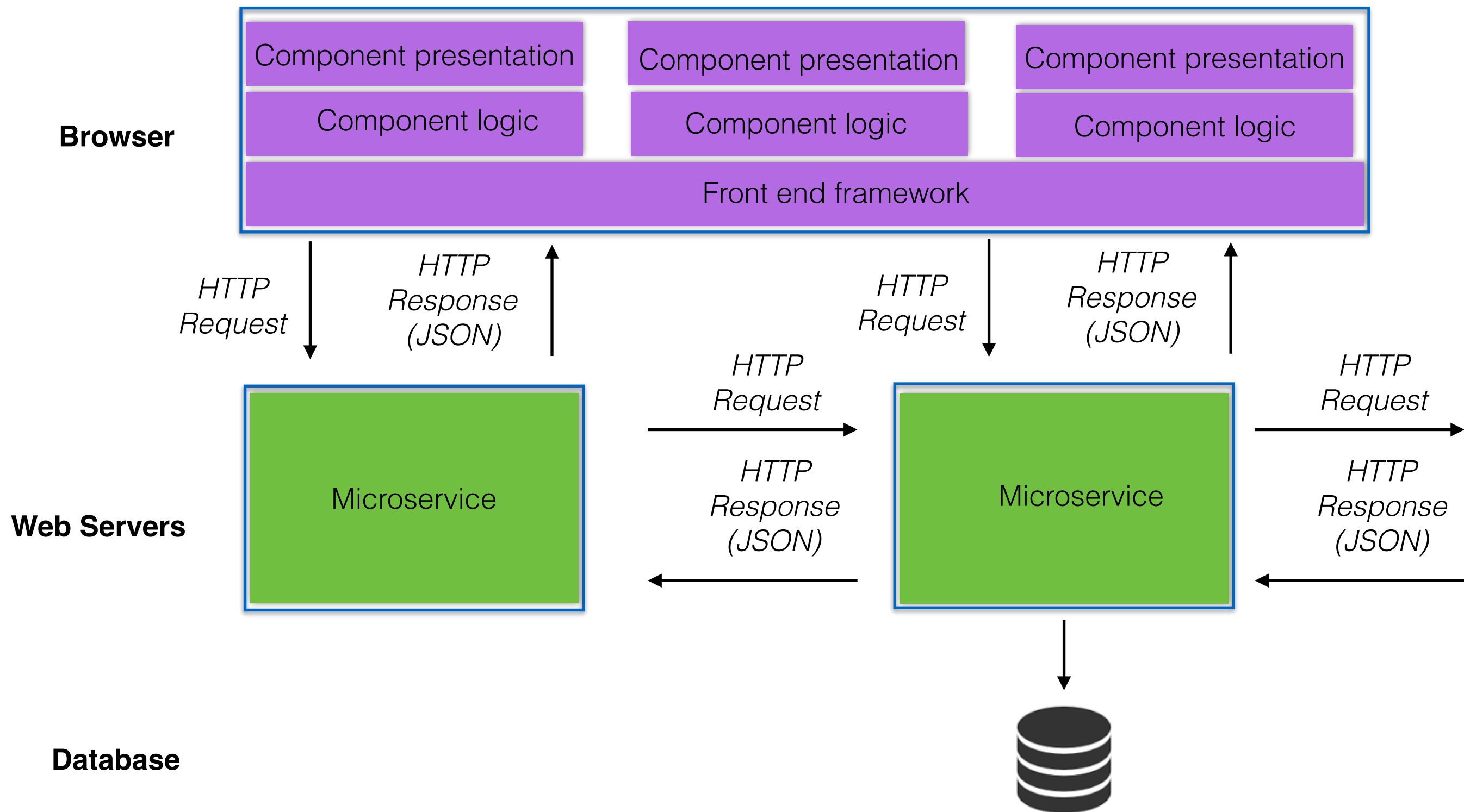
Limitations

- 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.

From <http://webfoundation.org/about/vision/history-of-the-web/>

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