SWE 432 -Web Application Development

Fall 2021

George Mason University

Dr. Kevin Moran

Week 1: Course Overview & Intro to Javascript

Welcome to SWE 432!

- **Initial Logistics:**
  - Welcome to the Lecture!
  - This Lecture is being recorded
  - ***Masks are required*** during class time
  - For the safety of everyone, there is ***no eating or drinking*** during lectures/in-class activities
    - However, there will be a 10 minute break in the middle of class.

Introductions

Instructor: Kevin Moran

Education: Ph.D. from William & Mary - 2018

Research Interests: Software Engineering , UI Analysis, Machine Learning

Office Hours: *Monday & Wednesday 1:00pm-2:00pm*

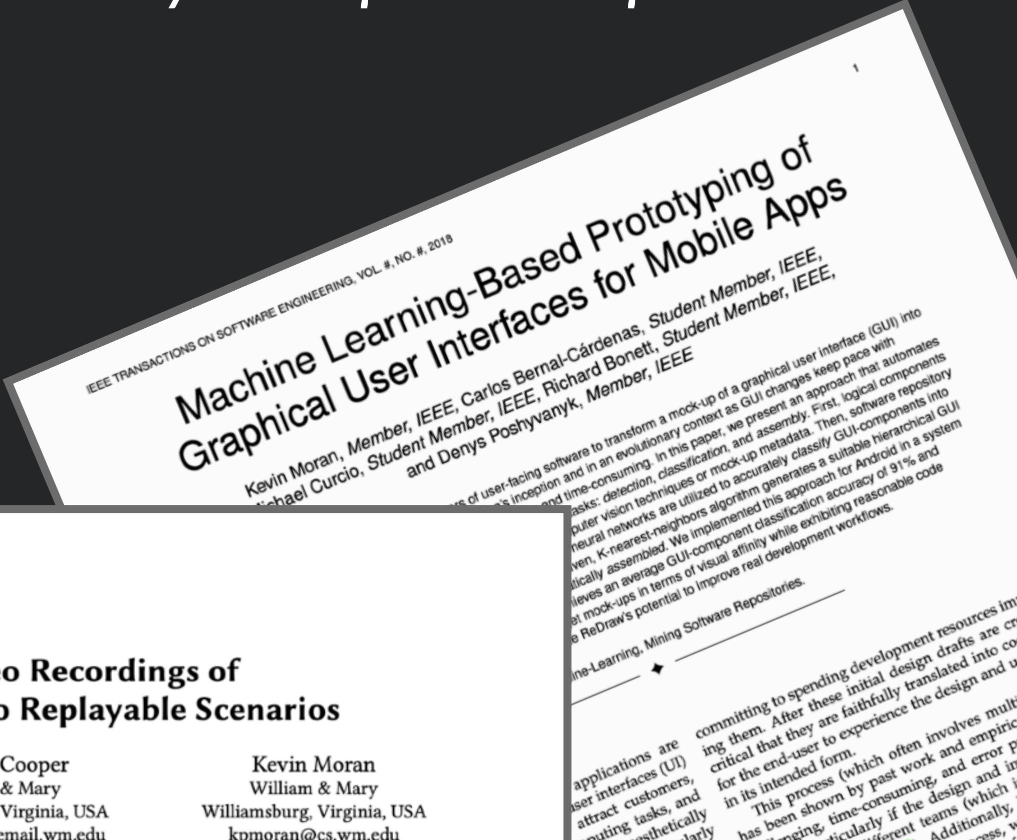
Introductions

Instructor: Kevin Moran

Education: Ph.D. from William & Mary - 2018

Research Interests: Software Engineering , UI Analysis, Machine Learning

Office Hours: *Monday & Wednesday 1:00pm-2:00pm*



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. #, NO. #, 2018 1

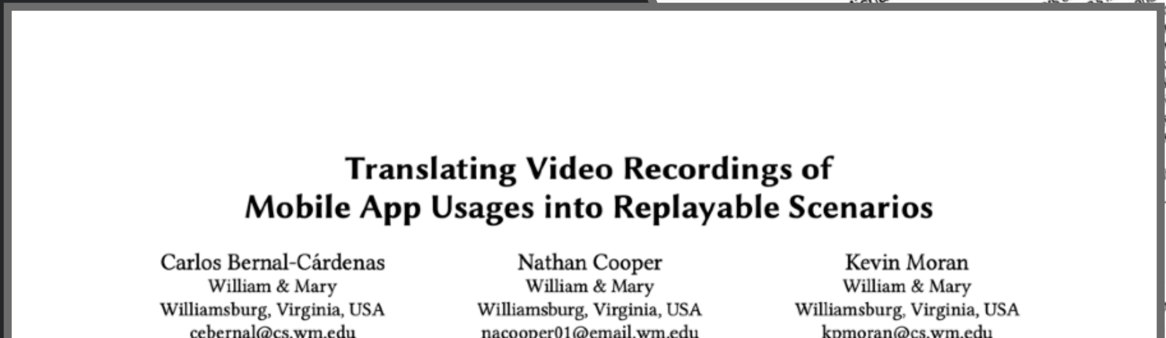
Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps

Kevin Moran, *Member, IEEE*, Carlos Bernal-Cárdenas, *Student Member, IEEE*, Michael Curcio, *Student Member, IEEE*, Richard Bonett, *Student Member, IEEE*, and Denys Poshyvanyk, *Member, IEEE*

s of user-facing software to transform a mock-up of a graphical user interface (GUI) into
's inception and in an evolutionary context as GUI changes keep pace with
nd time-consuming. In this paper, we present an approach that automates
asks: *detection*, *classification*, and *assembly*. First, logical components
puter vision techniques or mock-up metadata. Then, software repository
neural networks are utilized to accurately classify GUI-components into
ven, K-nearest-neighbors algorithm generates a suitable hierarchical GUI
ically assembled. We implemented this approach for Android in a system
ieves an average GUI-component classification accuracy of 91% and
et mock-ups in terms of visual affinity while exhibiting reasonable code
e ReDraw's potential to improve real development workflows.

ine-Learning, Mining Software Repositories.

applications are committing to spending development resources im
ser interfaces (UI) ing them. After these initial design drafts are cr
attract customers, critical that they are faithfully translated into co
uting tasks, and for the end-user to experience the design and u
esthetically in its intended form.
This process (which often involves mult
has been shown by past work and empiric
time-consuming, and error p
particularly if the design and in
different teams (which i
Additionally,



Translating Video Recordings of Mobile App Usages into Replayable Scenarios

Carlos Bernal-Cárdenas	Nathan Cooper	Kevin Moran
William & Mary	William & Mary	William & Mary
Williamsburg, Virginia, USA	Williamsburg, Virginia, USA	Williamsburg, Virginia, USA
cebernal@cs.wm.edu	nacooper01@email.wm.edu	kpmoran@cs.wm.edu

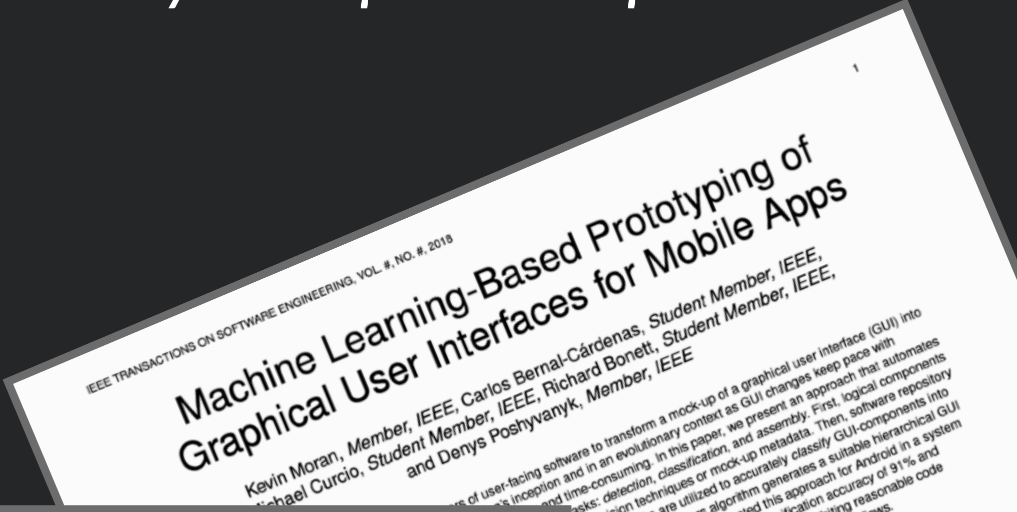
Introductions

Instructor: Kevin Moran

Education: Ph.D. from William & Mary - 2018

Research Interests: Software Engineering , UI Analysis, Machine Learning

Office Hours: *Monday & Wednesday 1:00pm-2:00pm*

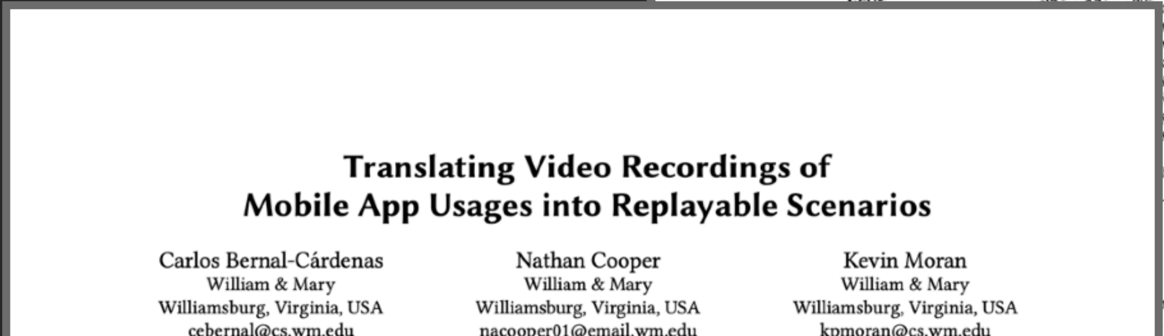


IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. #, NO. #, 2018

1

Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps

Kevin Moran, *Member, IEEE*, Carlos Bernal-Cárdenas, *Student Member, IEEE*, Michael Curcio, *Student Member, IEEE*, Richard Bonett, *Student Member, IEEE*, and Denys Poshyvanyk, *Member, IEEE*



Translating Video Recordings of Mobile App Usages into Replayable Scenarios

Carlos Bernal-Cárdenas
William & Mary
Williamsburg, Virginia, USA
cebernal@cs.wm.edu

Nathan Cooper
William & Mary
Williamsburg, Virginia, USA
nacooper01@email.wm.edu

Kevin Moran
William & Mary
Williamsburg, Virginia, USA
kpmoran@cs.wm.edu

Introductions

Teaching Assistant: David Samudio Gonzalez

Education: Current Ph.D. Student at GMU

Research Interests: Creating human-centered support tools for developers

Office Hours (Virtual): Monday, Thursday 3:00pm-4:00pm

Today's Agenda

1. Provide an overview of the ***Course Logistics*** - (15-20 mins)
2. Discuss the ***History*** and ***Present*** of the Modern Web- (20 mins)
3. Give a Brief Introduction to ***Javascript*** - (25-30 mins)
4. ***In-class Activity*** - Getting Started with Javascript - (~15 mins)

Course Logistics

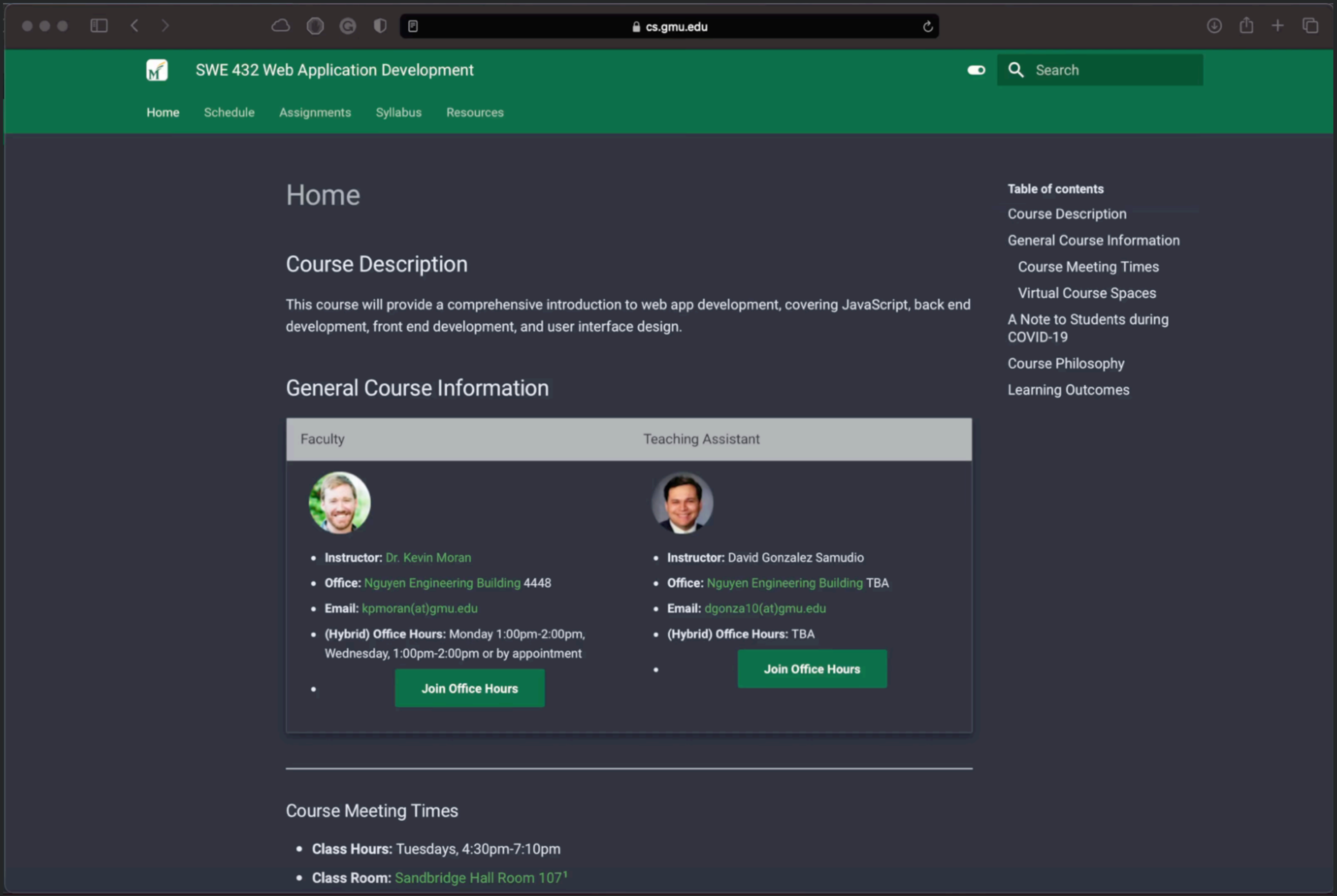
Course Lectures

- I am going to do my best to record course lectures.
- However, this is primarily for people that may have to miss class for illness, unforeseen circumstances
- You still need to attend class to take Quizzes and participate in in-class activities
- ***However, if you are feeling ill, <u>please do not come to class</u>. You can catch up on the lectures via recordings, and you can miss up to three quizzes.***

Course Resources

- Course Website: Syllabus, Schedule, Assignments, Lecture slides/recordings
- Ed Discussions: Announcements, Discussions
- Blackboard (MyMason): Grades
- Zoom: Hybrid/Virtual Office Hours

Course Website


cs.gmu.edu

SWE 432 Web Application Development

Search

Home Schedule Assignments Syllabus Resources

Home

Course Description

This course will provide a comprehensive introduction to web app development, covering JavaScript, back end development, front end development, and user interface design.

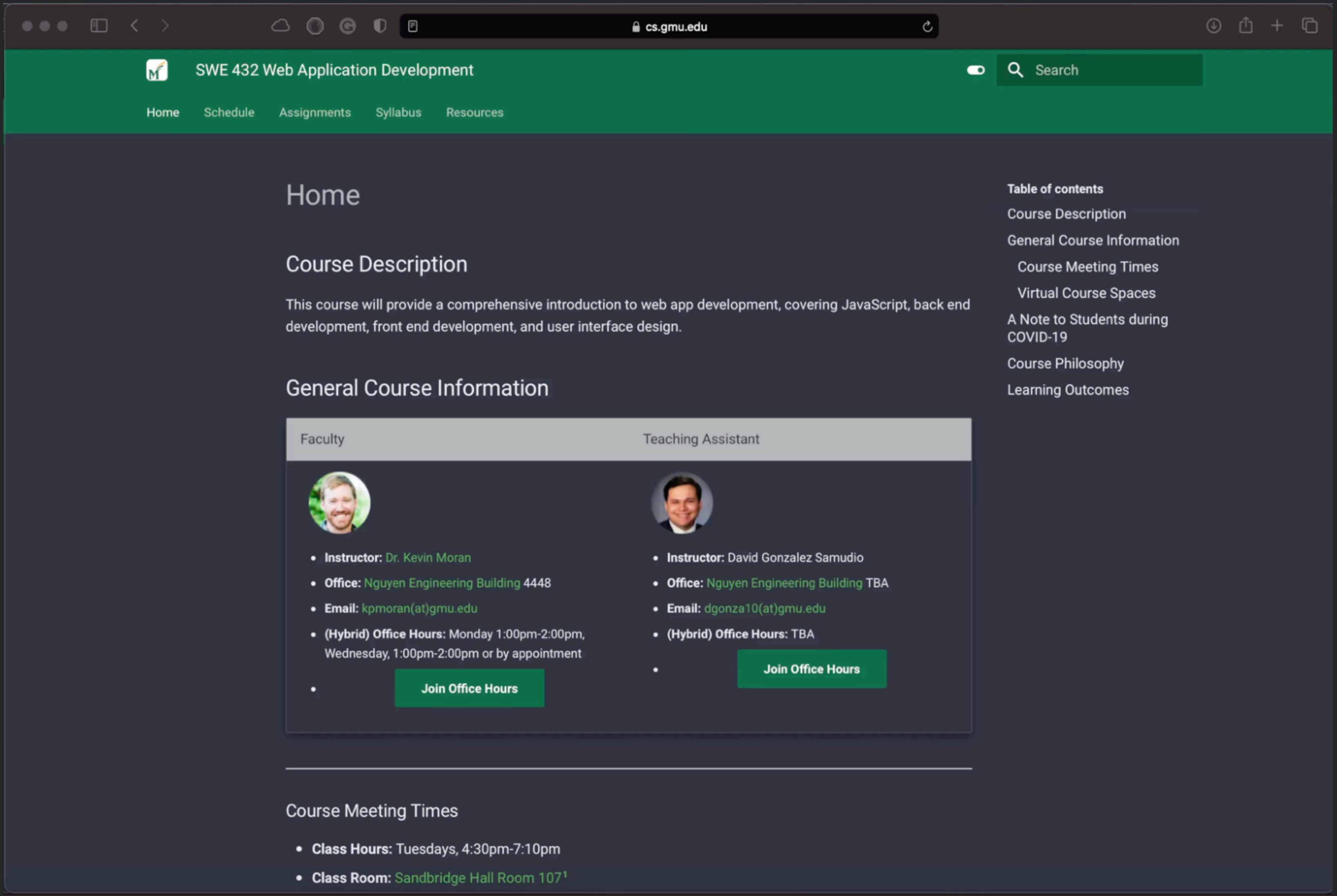
General Course Information

Faculty	Teaching Assistant
Instructor: Dr. Kevin Moran	**Instructor:** David Gonzalez Samudio
Office: Nguyen Engineering Building 4448	**Office:** Nguyen Engineering Building TBA
Email: kpmoran(at)gmu.edu	**Email:** dgonza10(at)gmu.edu
(Hybrid) Office Hours: Monday 1:00pm-2:00pm, Wednesday, 1:00pm-2:00pm or by appointment	**(Hybrid) Office Hours:** TBA
Join Office Hours	Join Office Hours

Course Meeting Times

- **Class Hours:** Tuesdays, 4:30pm-7:10pm
- **Class Room:** Sandbridge Hall Room 107[1]

CourseWebsite


cs.gmu.edu

SWE 432 Web Application Development

Search

Home Schedule Assignments Syllabus Resources

Home

Course Description

This course will provide a comprehensive introduction to web app development, covering JavaScript, back end development, front end development, and user interface design.

General Course Information

Faculty	Teaching Assistant
• **Instructor:** Dr. Kevin Moran • **Office:** Nguyen Engineering Building 4448 • **Email:** kpmoran(at)gmu.edu • **(Hybrid) Office Hours:** Monday 1:00pm-2:00pm, Wednesday, 1:00pm-2:00pm or by appointment • Join Office Hours	• **Instructor:** David Gonzalez Samudio • **Office:** Nguyen Engineering Building TBA • **Email:** dgonza10(at)gmu.edu • **(Hybrid) Office Hours:** TBA • Join Office Hours

Course Meeting Times

- **Class Hours:** Tuesdays, 4:30pm-7:10pm
- **Class Room:** Sandbridge Hall Room 107[1]

Course Materials

- There is no course textbook, however readings will be posted to the course website.
- There will be in-class activities for many lectures (***bring your laptop!***)

Grading Breakdown

- HW Assignments- (50%)
- In-Class Quizzes - (10%)
- Mid-Term Exam - (20%)
- Final Exam - (20%)

In-Class Activities

- Work together in small pairs/groups to gain experience trying out methods and concepts with examples

- No grades, but very important, as you will learn a lot from your classmates during these exercises

HW Assignments

- 5 Assignments over the course of the semester
- These will cover various web programming concepts, e.g., Javascript, Frontend/Backend development.
- ~2 weeks to complete each assignment
- Some code-related assignments will be auto-graded, we will also grade by hand for non-functional issues
- ***First HW Assignment will be posted tomorrow*** (Announcement will be made on Ed Discussions)

Late Policy - HW Assignments

- You will have ~2 weeks to complete each HW Assignment
- Can submit up to:
  - 24 hours late, lose 10%
  - 48 hours late, lose 20%
- HW submissions more than 48 hrs late will receive a 0
- ***These are still uncertain times, if you have unforeseen problems, please contact me & David <u>before</u> the deadline!***

Quizzes

- In class offered through Google Forms (again, bring your laptop to class)
- Pass/Fail (Pass if you are in class and submit a quiz, fail if you don’t)
- You can ***miss up to three quizzes*** during the semester (but no more)

Exams

- Midterm & non-comprehensive Final Exam
- Includes both in-class lectures and material from readings
- Multiple choice
- Synthesis-style, short essay questions
- Exams will be given in class, and during the assigned Final Exam period.

Honor Code

- Refresh yourself of the department honor code
- HW Assignments are 100% individual
    - Discussing assignments at high level: ***OK***, sharing code: ***<u>NOT OK</u>***
    - If in doubt, ask the instructor
    - If you copy code, we ***<u>WILL</u>*** notice (see some of my recent research results on Code Traceability)
- Quizzes must be completed by you, and while in class

Policies

- ***My promises to you:***
  - Quiz results will be available by the next class; we will discuss quizzes in class
  - Homework will be graded within 1 week of submission
  - Exams will be graded within 1 week

“Hands On” Sessions

- ***New this semester!***
- Every Thursday during David’s office hours, he will offer a “hands-on” sessions
- During these sessions you can:
    - Try out concepts from class
    - Ask more detailed programming questions
- Sessions will be conducted virtually over Zoom (Zoom room will be posted soon)
- ***You are not required to attend***, but it may be helpful if you find certain subjects/concepts challenging.

A Brief Overview and History of the Modern Web

Web Sites vs Web Apps?

Web Sites vs Web Apps?

Interactive?

Web Sites vs Web Apps?

Interactive?

User-generated content?

Web Sites vs Web Apps?

Interactive?

User-generated content?

Informational vs fun?

What is the web?

What is the web?

- A set of standards
  - TCP/IP, HTTP, URLs, HTML, CSS, …

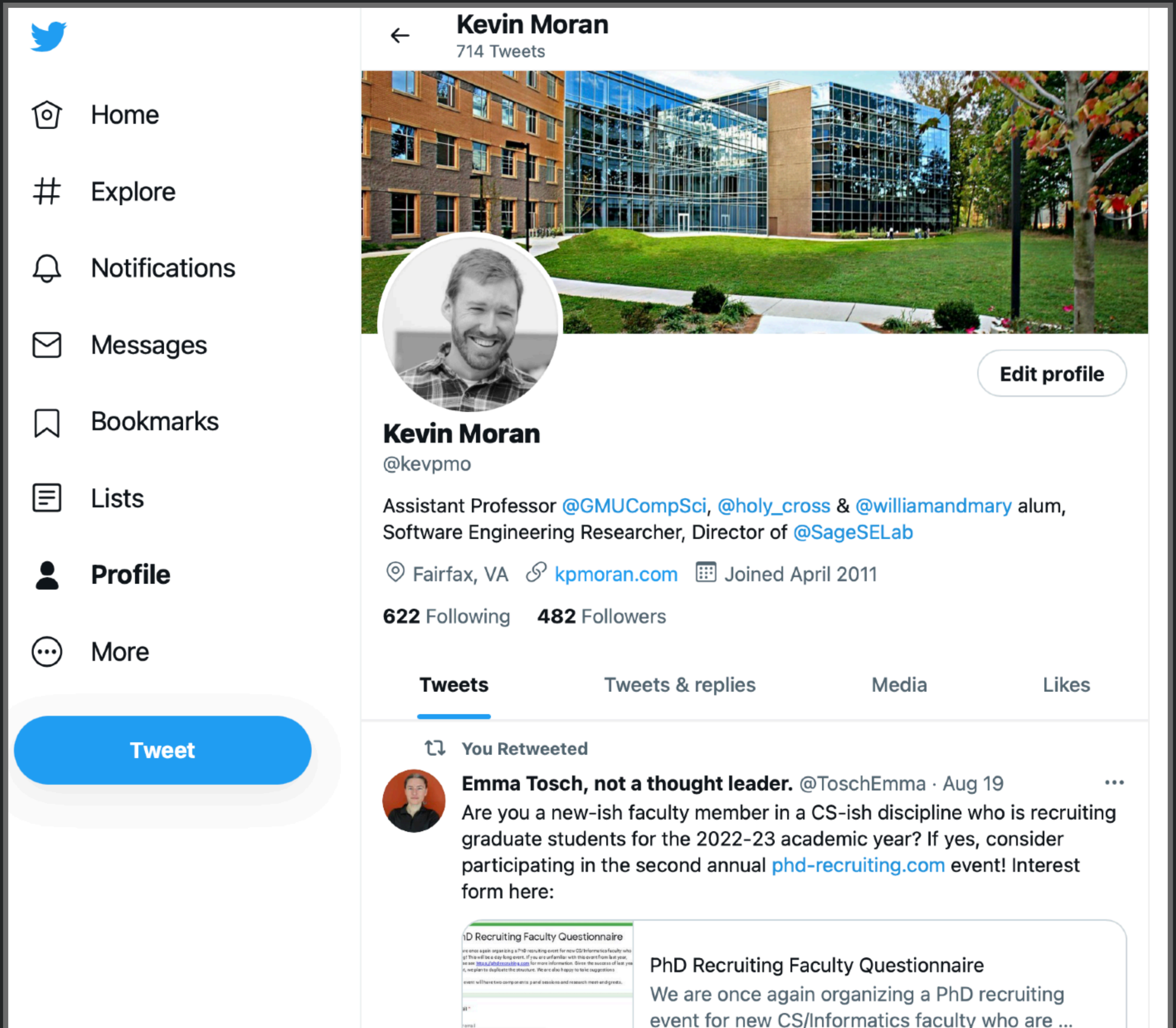
What is the web?

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

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



Kevin Moran
714 Tweets

Kevin Moran
@kevpmo

Assistant Professor @GMUCompSci, @holy_cross & @williamandmary alum, Software Engineering Researcher, Director of @SageSELab

Fairfax, VA kpmoran.com Joined April 2011

622 Following 482 Followers

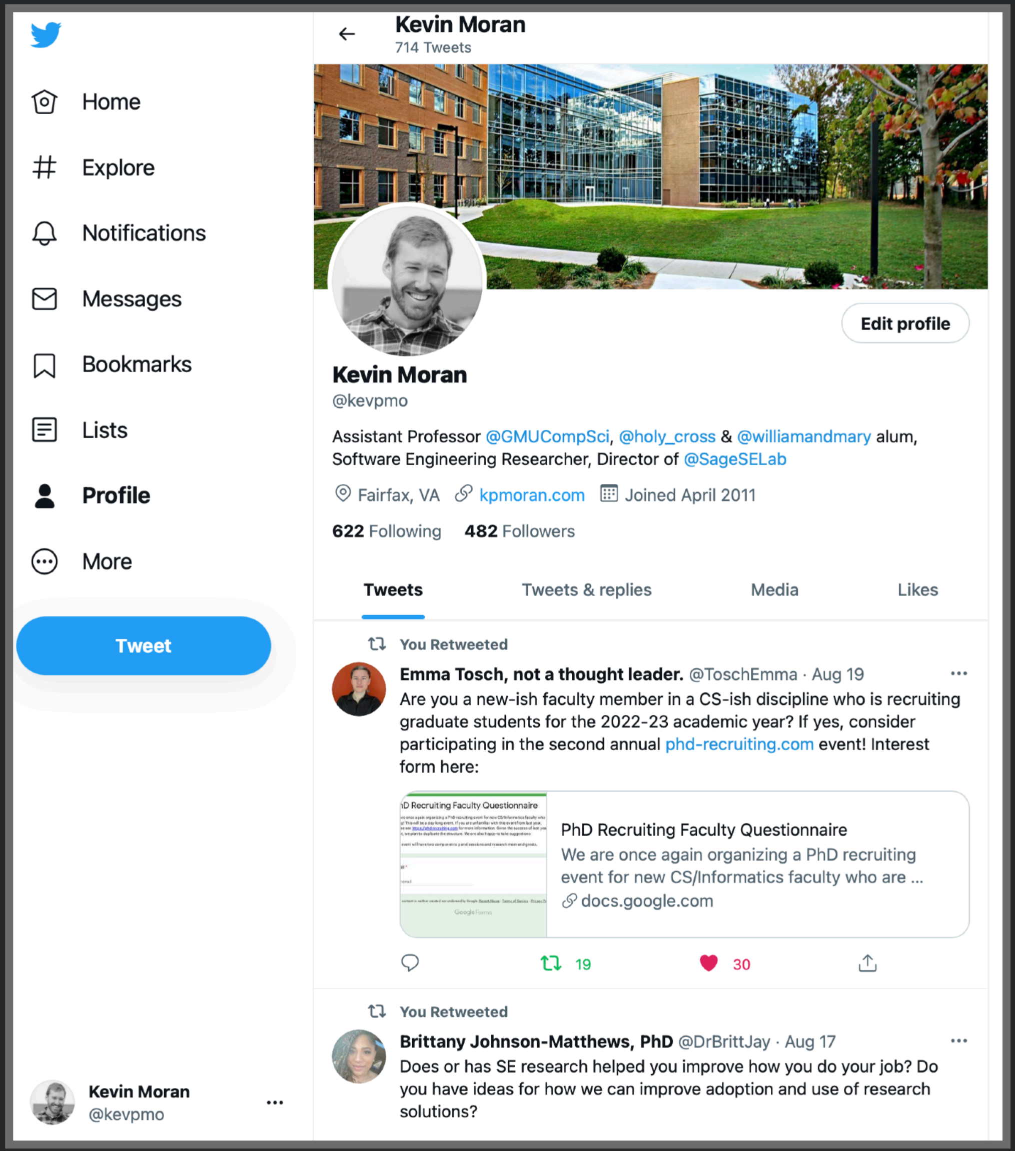
You Retweeted

Emma Tosch, not a thought leader. @ToschEmma · Aug 19

Are you a new-ish faculty member in a CS-ish discipline who is recruiting graduate students for the 2022-23 academic year? If yes, consider participating in the second annual phd-recruiting.com event! Interest form here:

PhD Recruiting Faculty Questionnaire
We are once again organizing a PhD recruiting event for new CS/Informatics faculty who are ...

Systems Perspective



Kevin Moran
714 Tweets

Kevin Moran
@kevpmo

Assistant Professor @GMUCompSci, @holy_cross & @williamandmary alum, Software Engineering Researcher, Director of @SageSELab

Fairfax, VA kpmoran.com Joined April 2011

622 Following 482 Followers

You Retweeted

Emma Tosch, not a thought leader. @ToschEmma · Aug 19

Are you a new-ish faculty member in a CS-ish discipline who is recruiting graduate students for the 2022-23 academic year? If yes, consider participating in the second annual phd-recruiting.com event! Interest form here:

PhD Recruiting Faculty Questionnaire
We are once again organizing a PhD recruiting event for new CS/Informatics faculty who are ...
docs.google.com

19 30

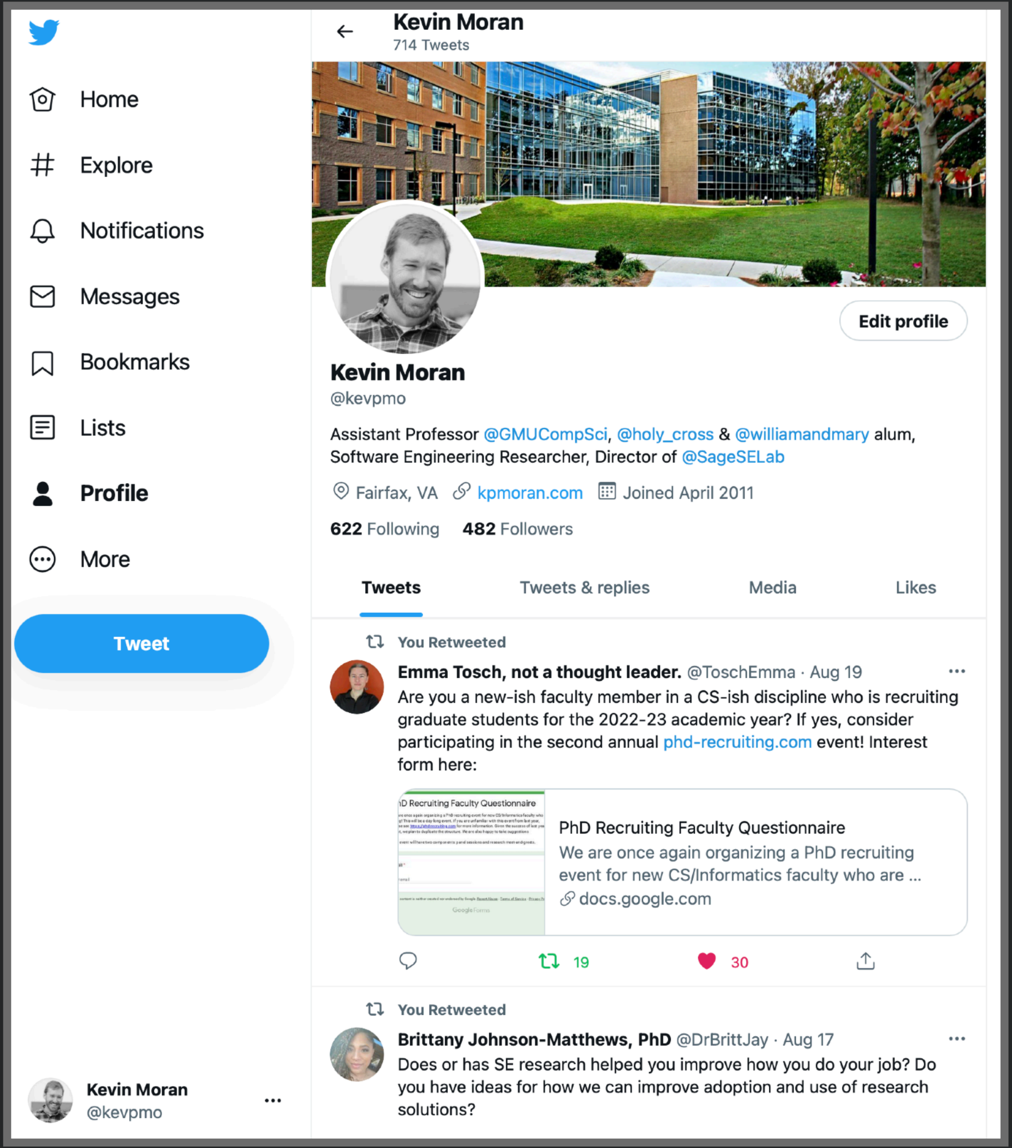
You Retweeted

Brittany Johnson-Matthews, PhD @DrBrittJay · Aug 17

Does or has SE research helped you improve how you do your job? Do you have ideas for how we can improve adoption and use of research solutions?

- 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


Kevin Moran
714 Tweets

Home
Explore
Notifications
Messages
Bookmarks
Lists
Profile
More
Tweet

Edit profile

Kevin Moran
@kevpmo

Assistant Professor @GMUCompSci, @holy_cross & @williamandmary alum, Software Engineering Researcher, Director of @SageSELab

Fairfax, VA kpmoran.com Joined April 2011

622 Following 482 Followers

Tweets Tweets & replies Media Likes

You Retweeted
Emma Tosch, not a thought leader. @ToschEmma · Aug 19
Are you a new-ish faculty member in a CS-ish discipline who is recruiting graduate students for the 2022-23 academic year? If yes, consider participating in the second annual phd-recruiting.com event! Interest form here:

PhD Recruiting Faculty Questionnaire
We are once again organizing a PhD recruiting event for new CS/Informatics faculty who are ...
docs.google.com

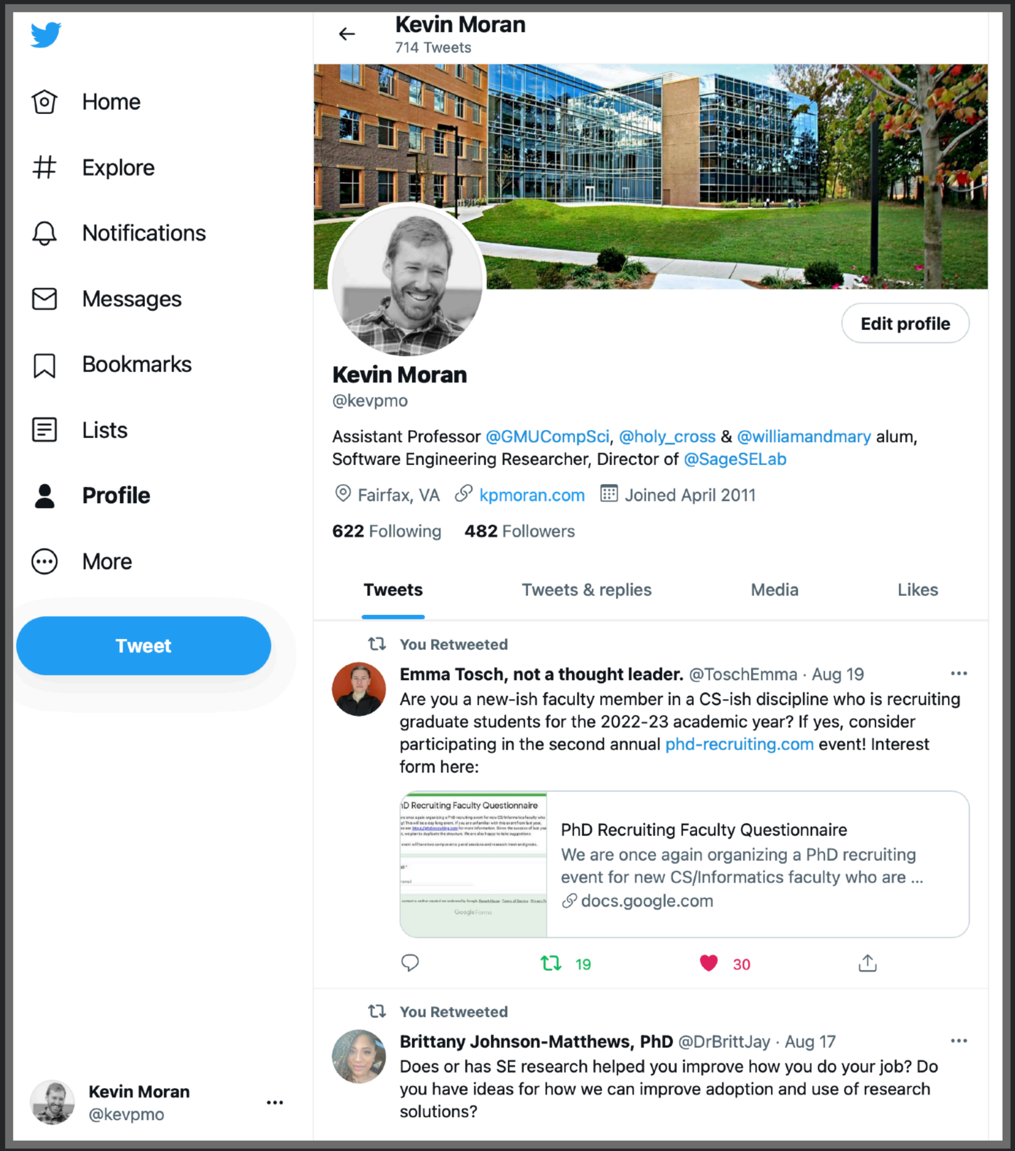
19 30

You Retweeted
Brittany Johnson-Matthews, PhD @DrBrittJay · Aug 17
Does or has SE research helped you improve how you do your job? Do you have ideas for how we can improve adoption and use of research solutions?

Kevin Moran
@kevpmo

- 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



Kevin Moran
714 Tweets

Kevin Moran
@kevpmo

Assistant Professor @GMUCompSci, @holy_cross & @williamandmary alum, Software Engineering Researcher, Director of @SageSELab

Fairfax, VA kpmoran.com Joined April 2011

622 Following 482 Followers

You Retweeted

Emma Tosch, not a thought leader. @ToschEmma · Aug 19
Are you a new-ish faculty member in a CS-ish discipline who is recruiting graduate students for the 2022-23 academic year? If yes, consider participating in the second annual phd-recruiting.com event! Interest form here:

PhD Recruiting Faculty Questionnaire
We are once again organizing a PhD recruiting event for new CS/Informatics faculty who are ...
docs.google.com

19 30

You Retweeted

Brittany Johnson-Matthews, PhD @DrBrittJay · Aug 17
Does or has SE research helped you improve how you do your job? Do you have ideas for how we can improve adoption and use of research solutions?

Kevin Moran
@kevpmo

- 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

Pre-Web

- “As We May Think”, by Vannevar Bush, in The Atlantic Monthly, July 1945

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

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

Pre-Web - Memex

- MEMEX = MEMory EXtension

Pre-Web - Memex

- MEMEX = MEMory EXtension
- Create and follow “associative trails” (links) and annotations between microfilm documents

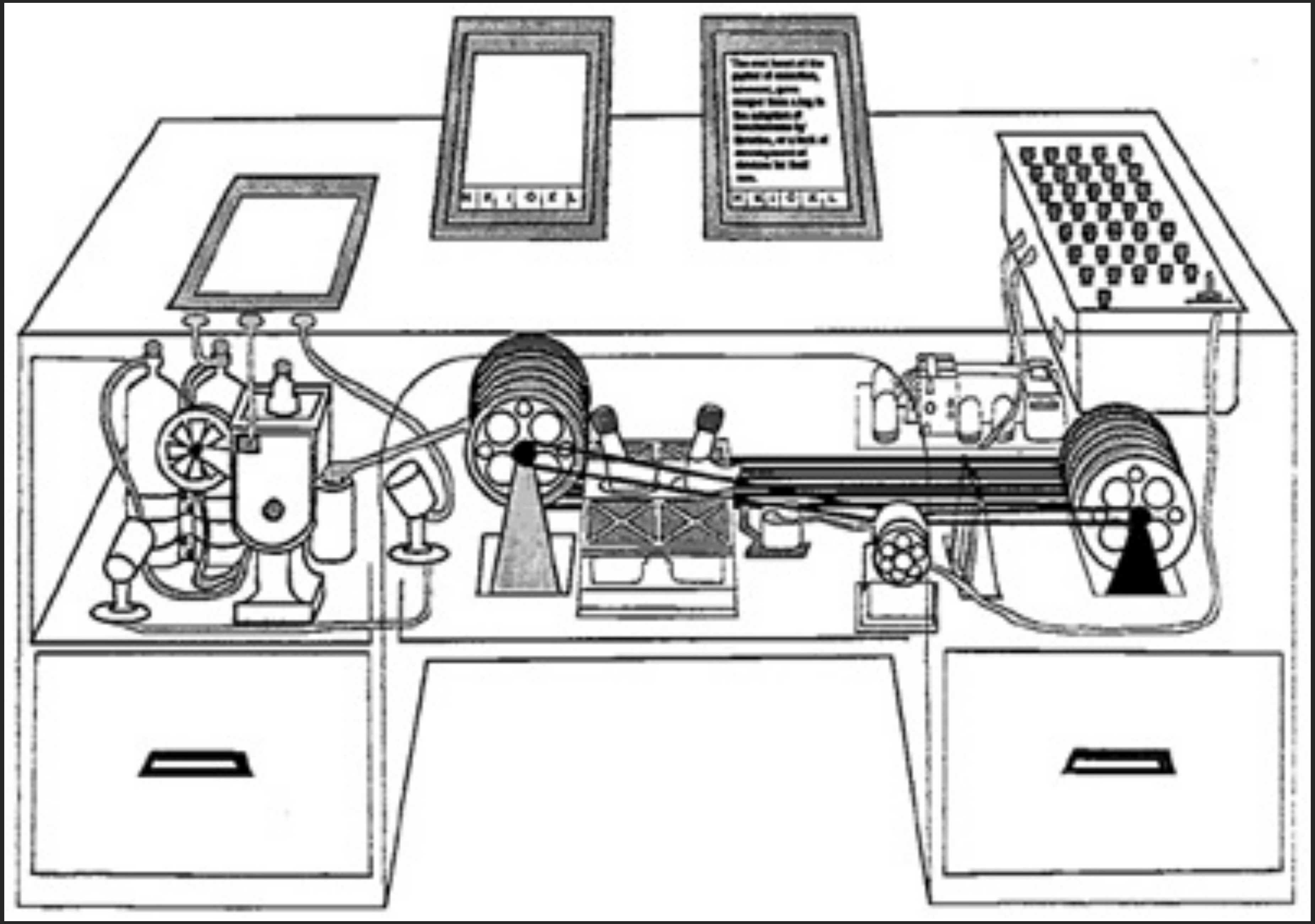
Pre-Web - Memex

- MEMEX = MEMory EXtension
- Create and follow "associative trails" (links) and annotations between microfilm documents
- Technically based on "rapid selectors" Vannevar Bush built in 1930's to search microfilm

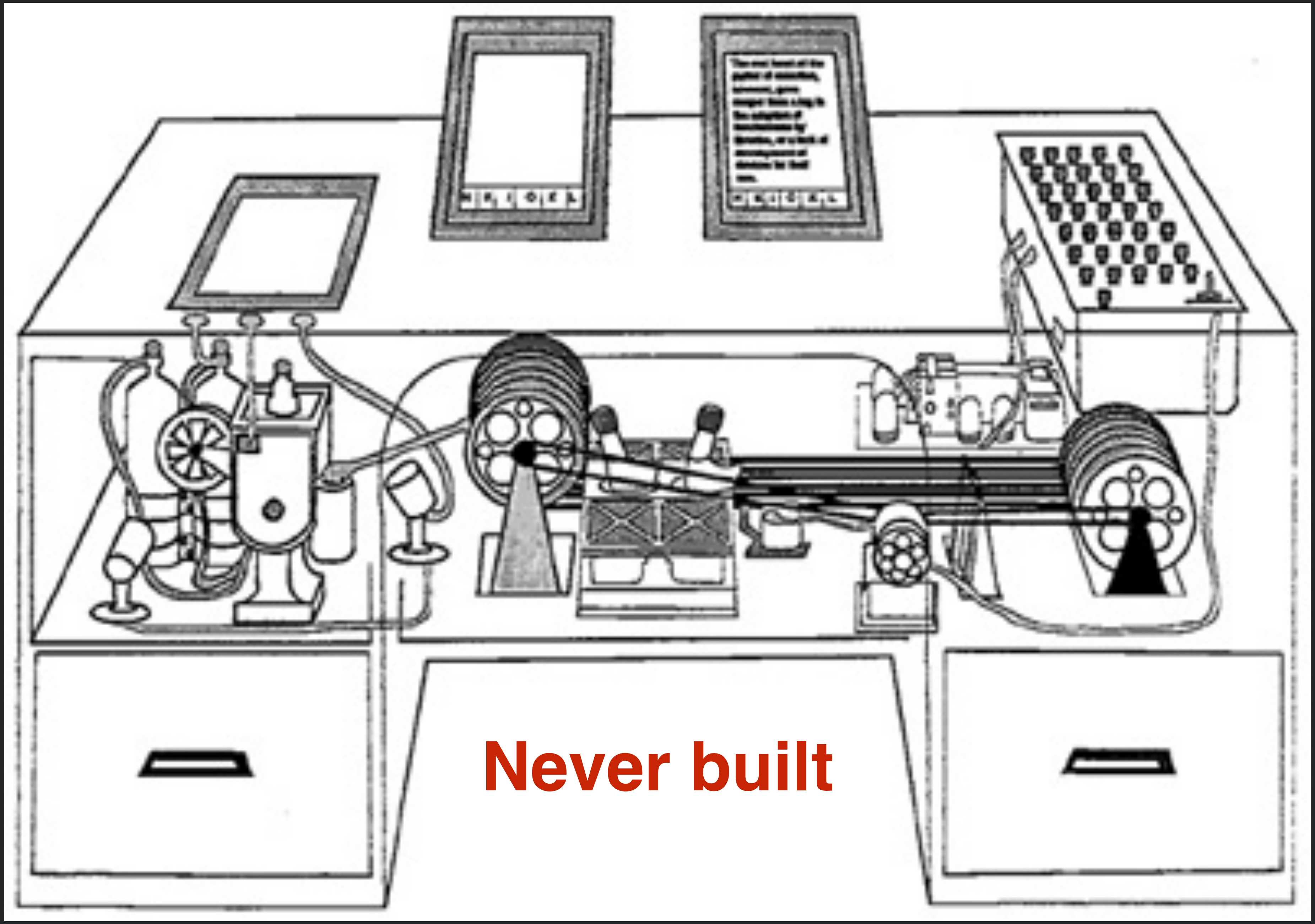
Pre-Web - Memex

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

Pre-Web - Memex

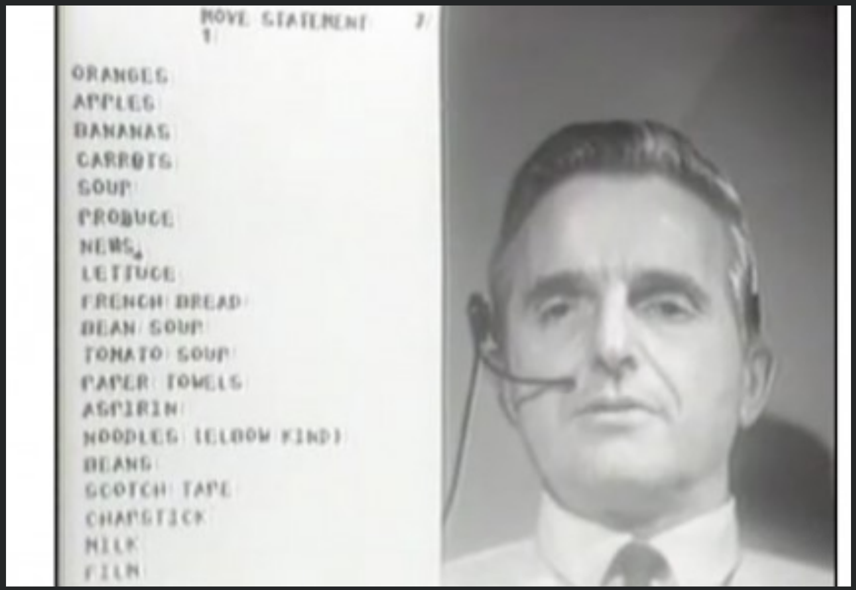


Pre-Web - Memex



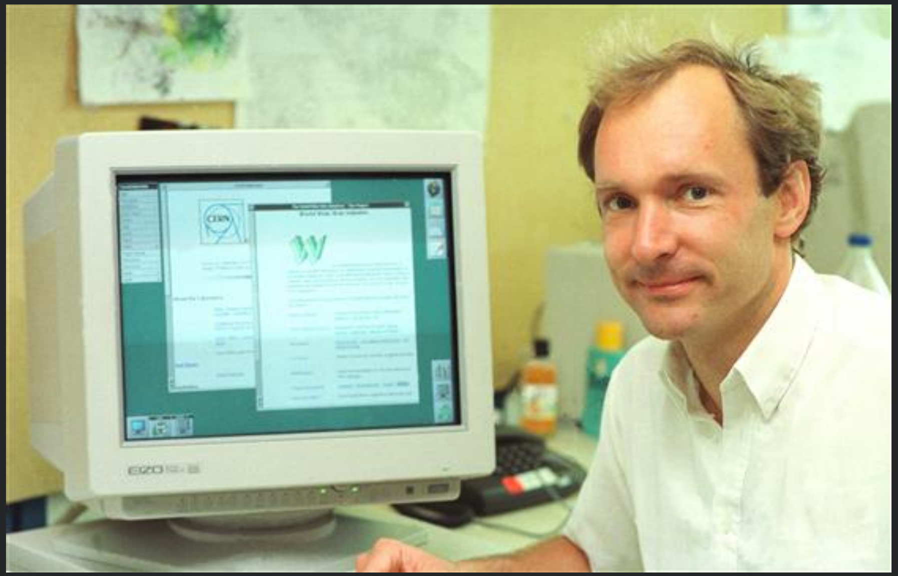
Hypertext and the WWW

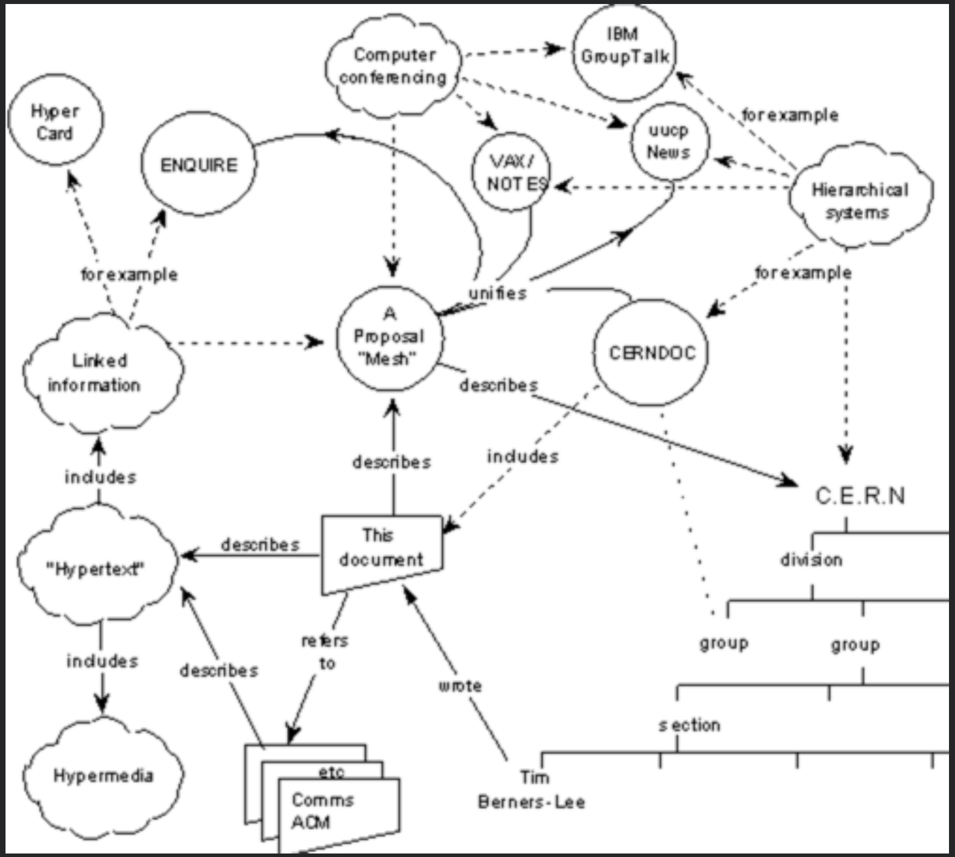
- 1965: Ted Nelson coins “hypertext” (the HT in **HT**ML) - “beyond” the linear constraints of text
- Many hypertext/hypermedia systems followed, many not sufficiently scalable to take off
- 1968: Doug Engelbart gives “the mother of all demos”, demonstrating windows, hypertext, graphics, video conferencing, 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



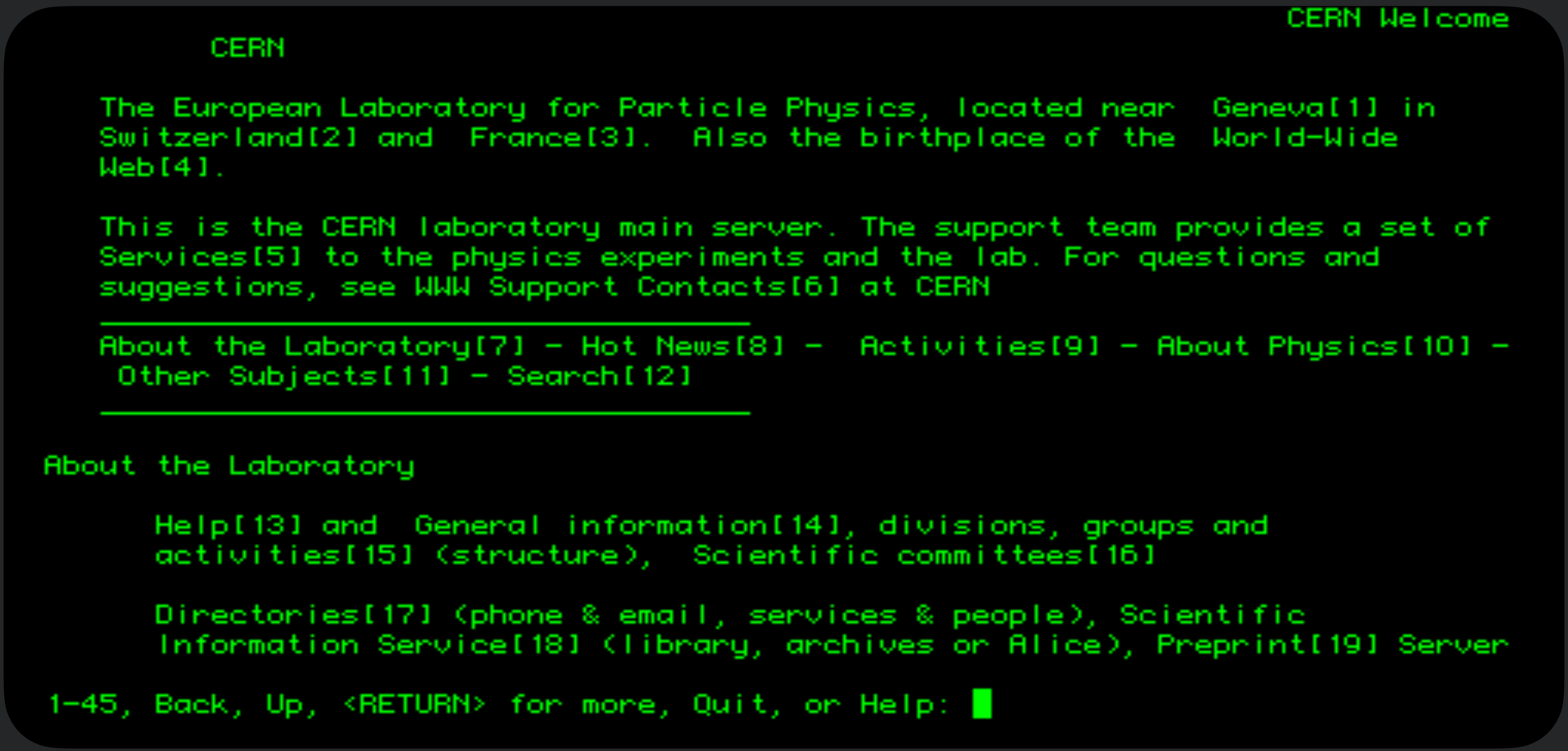
Origin of the Web

- 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

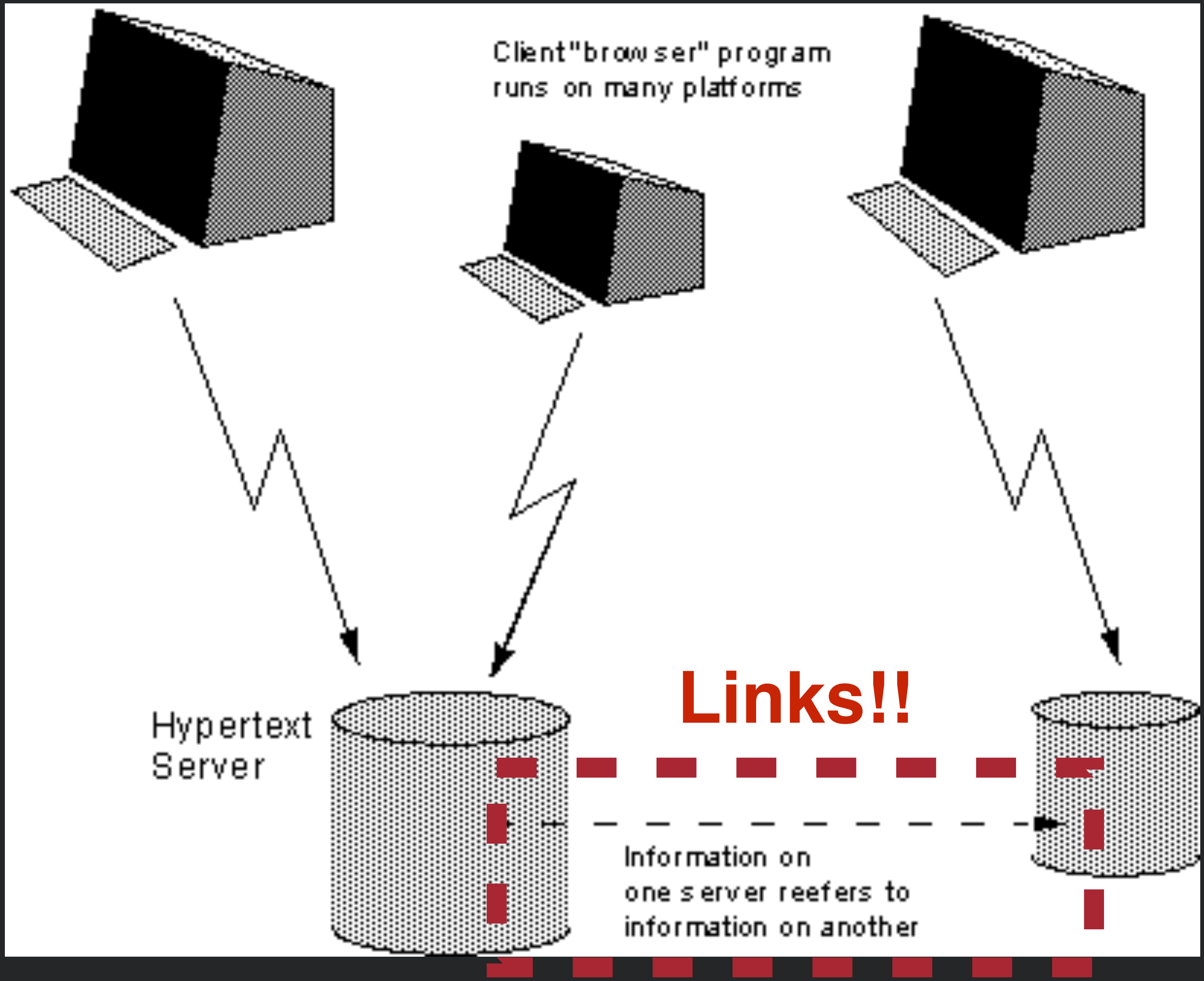


Early Browsers



```
                                                                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


Client "browser" program runs on many platforms
Hypertext Server
Links!!
Information on one server reefers to information on another

URI: Universal Resource Identifier

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

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

URI: Universal Resource Identifier

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

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

$\uparrow$

"Use HTTP scheme"

Other popular schemes:

ftp, mailto, file

URI: Universal Resource Identifier

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

http://cs.gmu.edu/~kpmoran/swe-432-f21.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)

URI: Universal Resource Identifier

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

http://cs.gmu.edu/~kpmoran/swe-432-f21.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 ~kpmoran/swe-432-f21.html"

URI: Universal Resource Identifier

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

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

"Use HTTP scheme"

Other popular schemes:
ftp, mailto, file

"Connect to cs.gmu.edu"

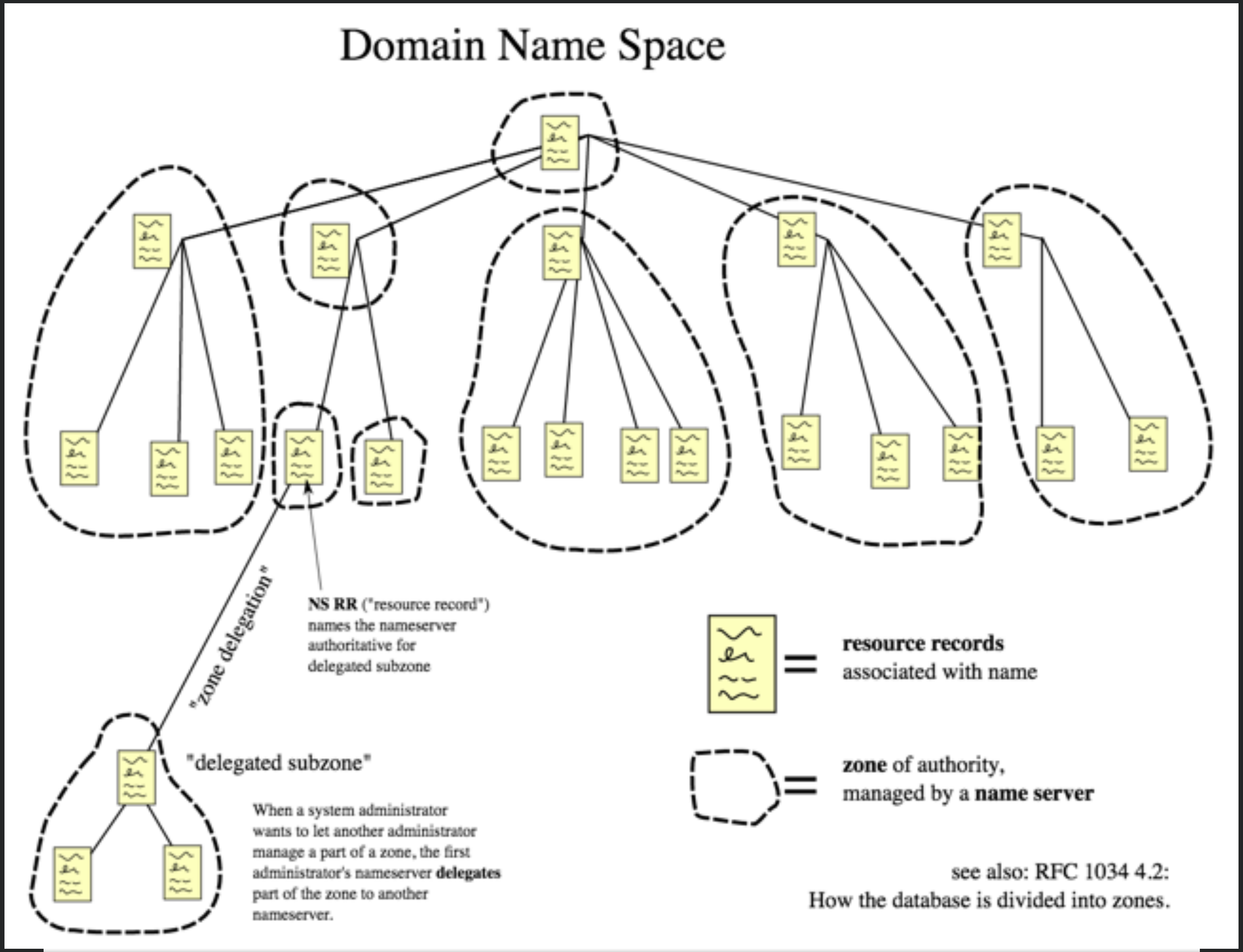
May be host name or an IP address
Optional port name (e.g., :80 for port 80)

"Request ~kpmoran/swe-432-f21.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

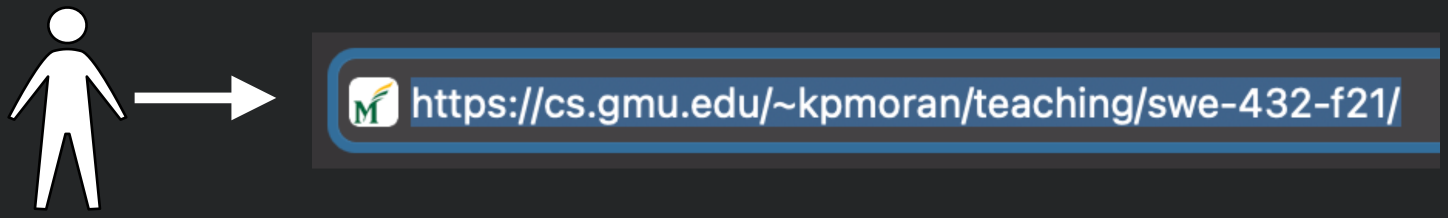


Domain Name Space
"zone delegation"
NS RR ("resource record") names the nameserver authoritative for delegated subzone
"delegated subzone"
When a system administrator wants to let another administrator manage a part of a zone, the first administrator's nameserver **delegates** part of the zone to another nameserver.
= **resource records** associated with name
= **zone** of authority, managed by a **name server**
see also: RFC 1034 4.2: How the database is divided into zones.

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

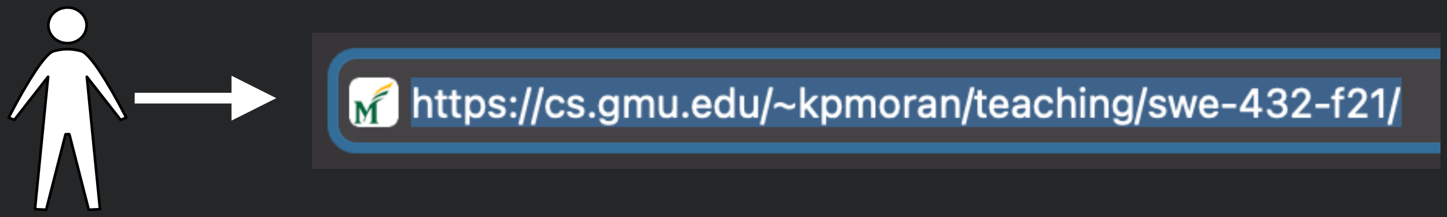
HTTP: HyperText Transfer Protocol

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



HTTP: HyperText Transfer Protocol

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

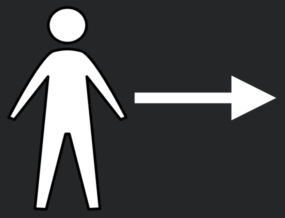

https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

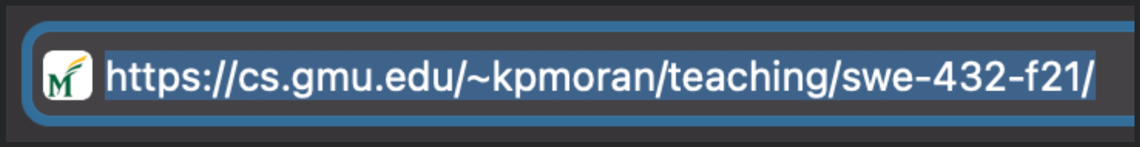
HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

HTTP: HyperText Transfer Protocol

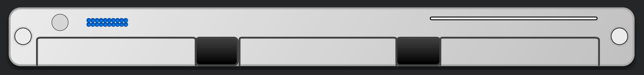
High-level protocol built on TCP/IP that defines how data is transferred on the web





https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

web server

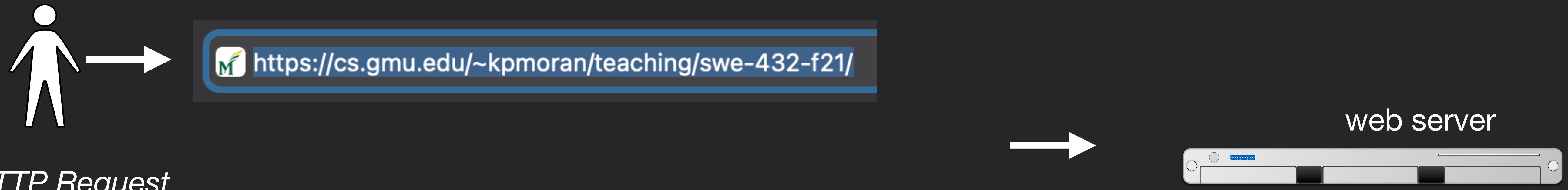


HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

HTTP: HyperText Transfer Protocol

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



HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

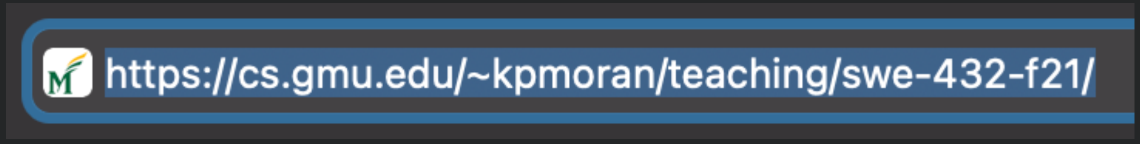
Reads file from disk

HTTP Response

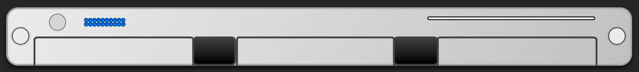
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

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

HTTP: HyperText Transfer Protocol

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



https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/



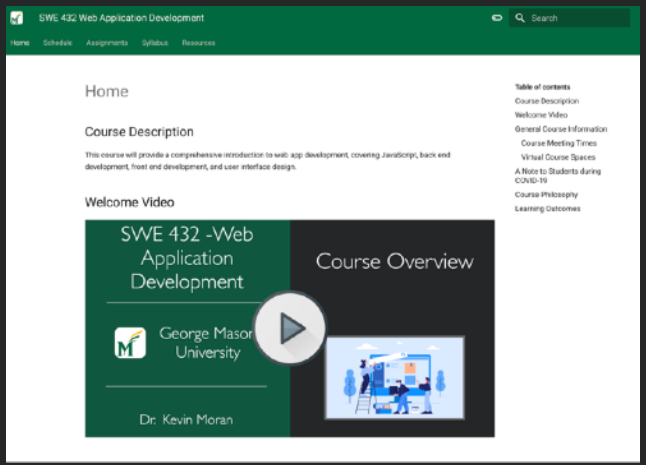
HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Reads file from disk

HTTP Response

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

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



SWE 432 Web Application Development

Search

Home Schedule Assignments Syllabus Resources

Home

Course Description

This course will provide a comprehensive introduction to web app development, covering JavaScript, back end development, front end development, and user interface design.

Welcome Video

SWE 432 -Web Application Development

George Mason University

Dr. Kevin Moran

Course Overview

Table of contents

Course Description

Welcome Video

General Course Information

Course Meeting Times

Virtual Course Spaces

A Note to Students during COVID-19

Course Philosophy

Learning Outcomes

HTTP Requests

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Other popular types:
POST, PUT, DELETE, HEAD

- 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 Requests

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

“GET request”

Other popular types:
POST, PUT, DELETE, HEAD

- 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 Requests

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

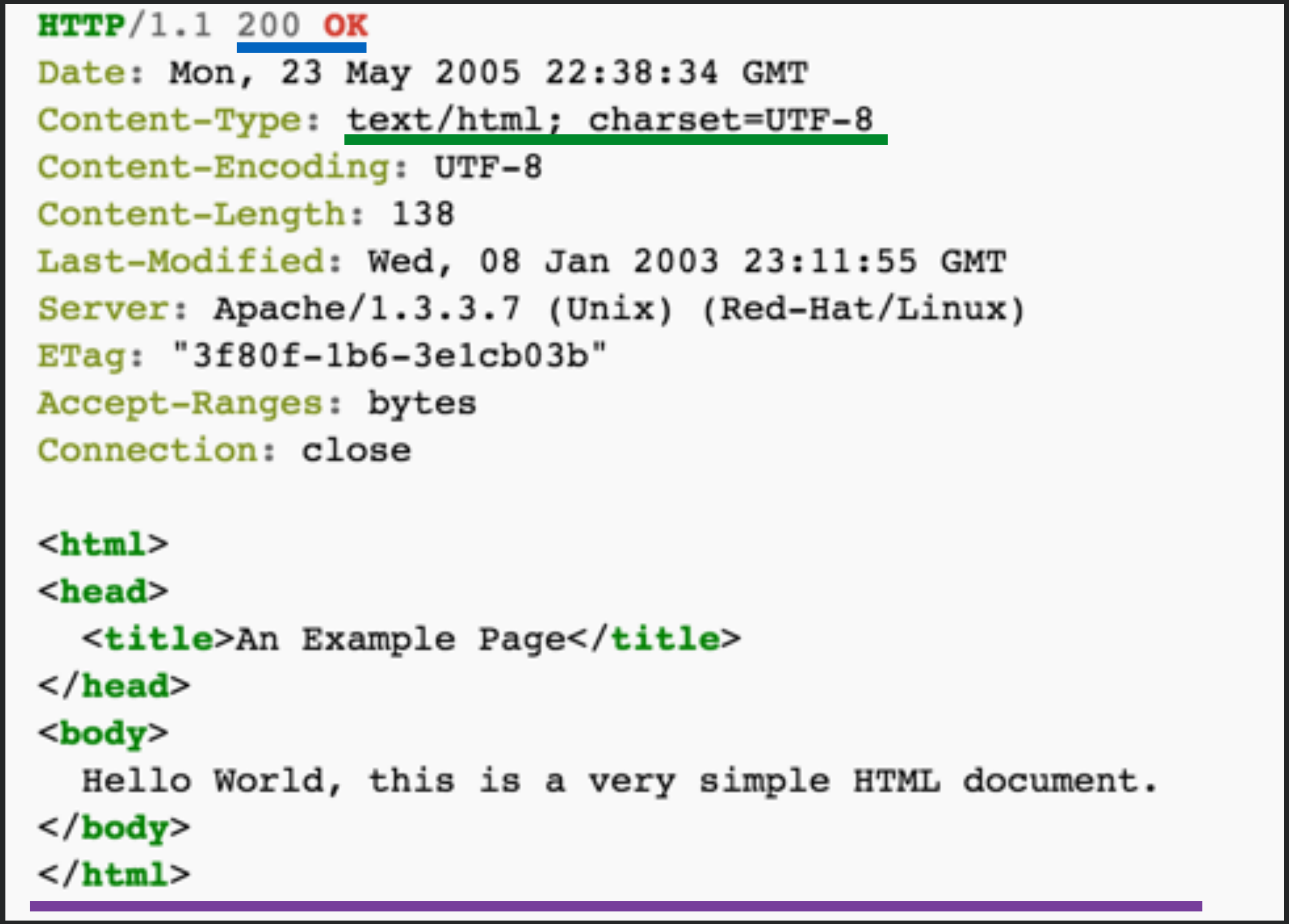
“GET request”

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

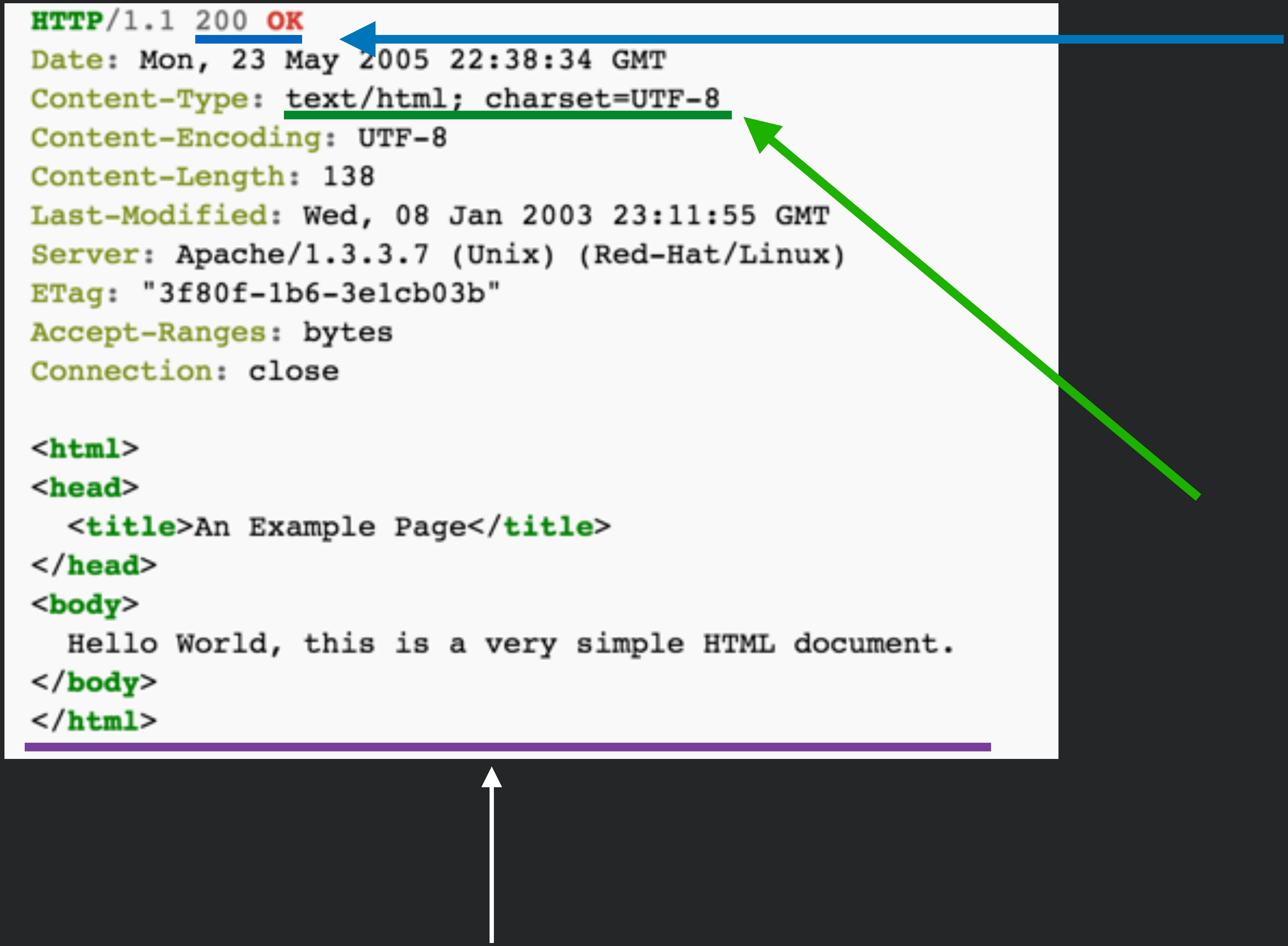


```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

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

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

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

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 document

- NOT a programming language

HTML: HyperText Markup Language

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

- NOT a programming language
- Tags are added to markup the text, encompassed with <>'s

HTML: HyperText Markup Language

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

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

HTML: HyperText Markup Language

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

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

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

HTML: HyperText Markup Language

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

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

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

$\downarrow$

`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

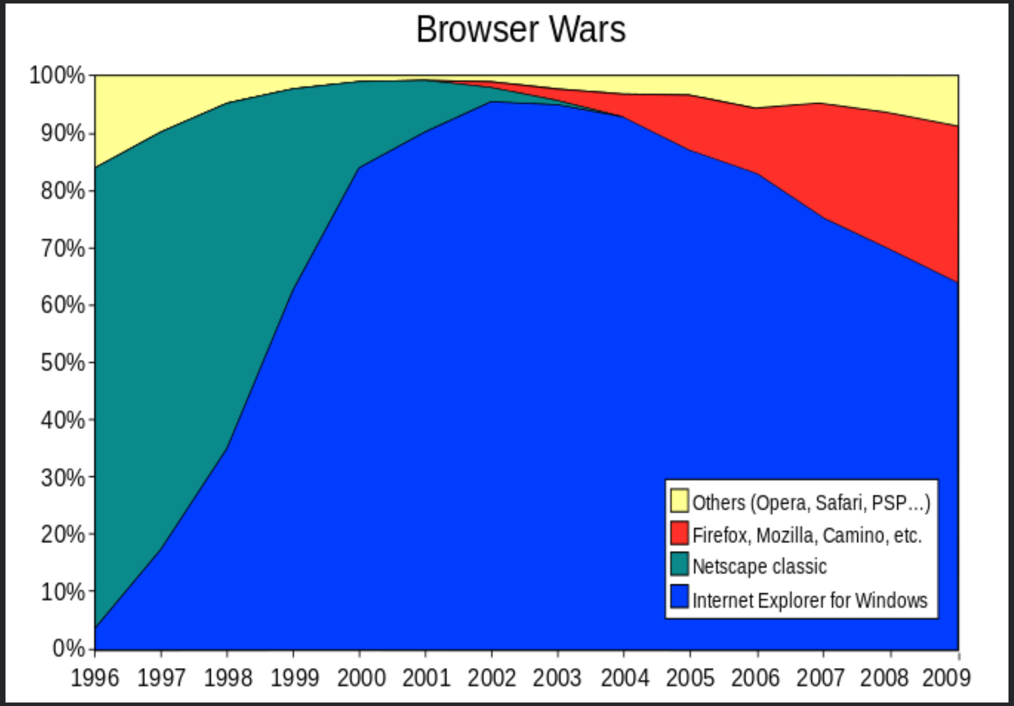
- Evolving competing architectures for organizing content and computation between browser (client) and web server
- 1990s: static web pages
- 1990s: server-side scripting (CGI, PHP, ASP, ColdFusion, JSP, …)
- 2000s: single page apps (JQuery)
- 2010s: front-end frameworks (Angular, React, Vue…), microservices

Static Web Pages

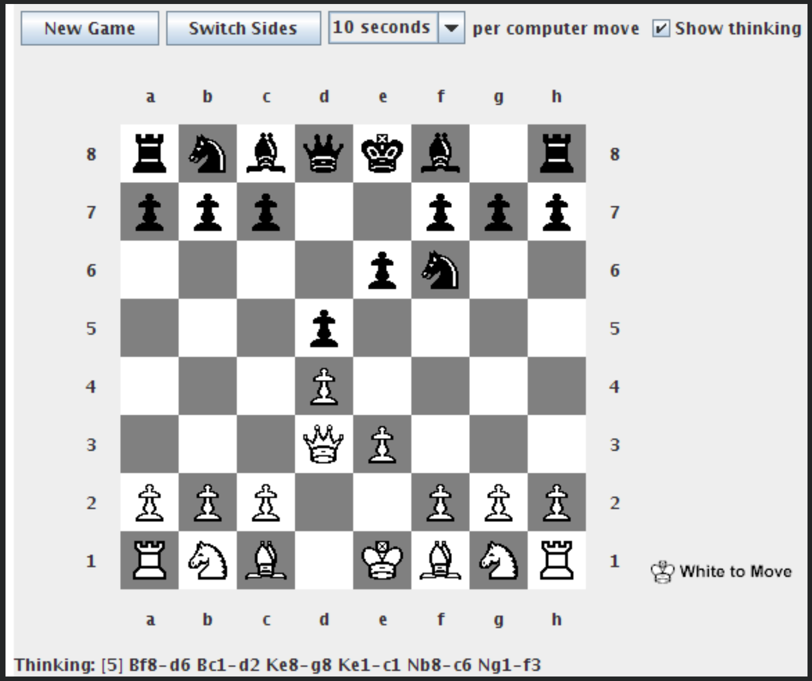
- URL corresponds to directory location on server
  - e.g. http://domainName.com/img/image5.jpg maps to img/image5.jpg file on server
- Server responds to HTTP request by returning requested files
- Advantages
  - Simple, easily cacheable, easily searchable
- Disadvantages
  - No interactivity

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

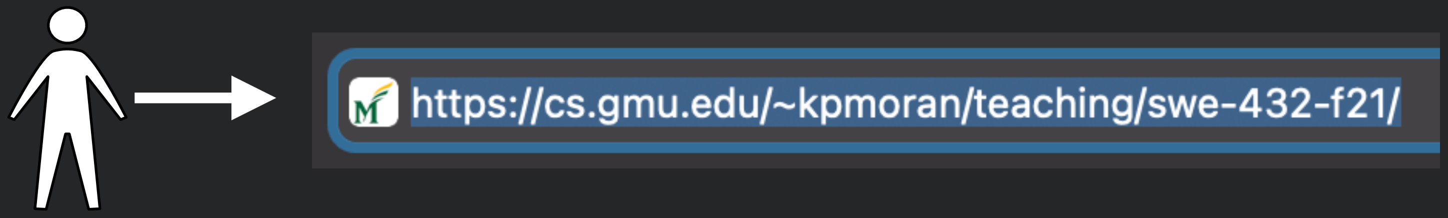


Thinking: [5] Bf8-d6 Bc1-d2 Ke8-g8 Ke1-c1 Nb8-c6 Ng1-f3

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

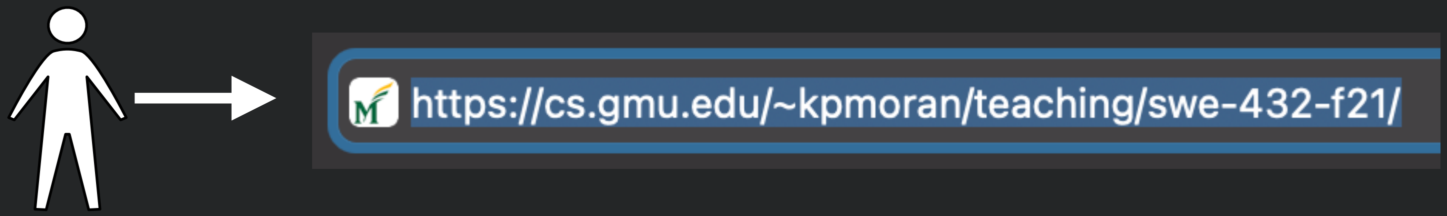
Dynamic Web Pages

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



Dynamic Web Pages

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

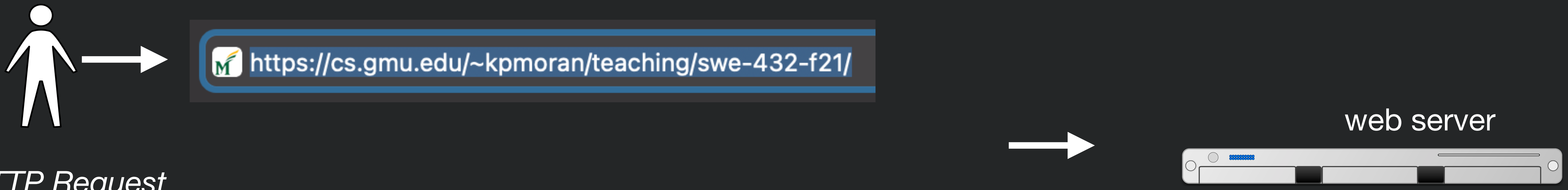


HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Dynamic Web Pages

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

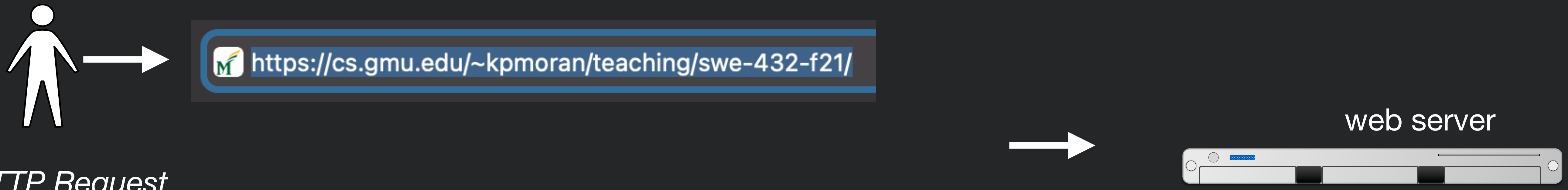


HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Dynamic Web Pages

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



HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

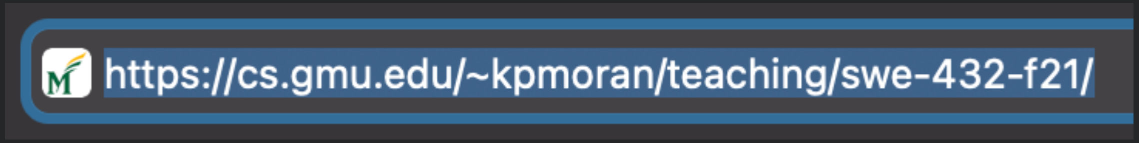
RUNS a PROGRAM!!!

HTTP Response

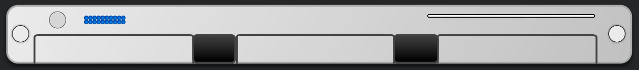
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

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

Dynamic Web Pages

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


https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

web server



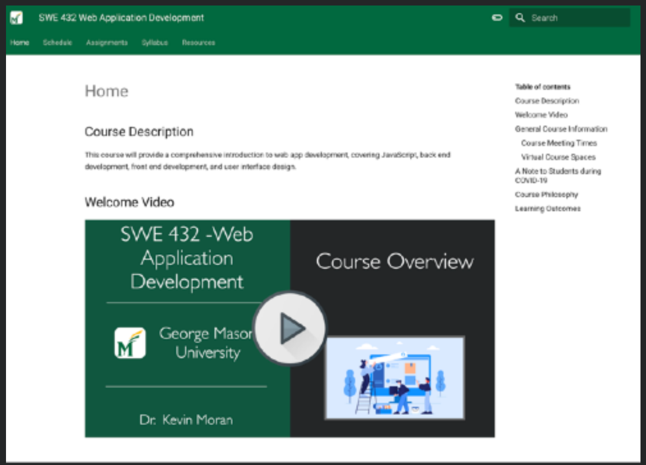
HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

RUNS a PROGRAM!!!

HTTP Response

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

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


SWE 432 Web Application Development

Search

Home Schedule Assignments Syllabus Resources

Home

Course Description

This course will provide a comprehensive introduction to web app development, covering JavaScript, back end development, front end development, and user interface design.

Welcome Video

SWE 432 -Web Application Development

George Mason University

Dr. Kevin Moran

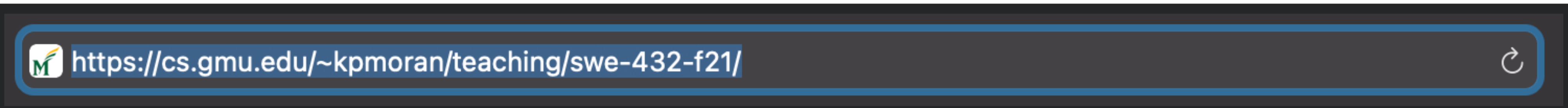
Course Overview

Table of contents

- Course Description
- Welcome Video
- General Course Information
  - Course Meeting Times
  - Virtual Course Spaces
- A Note to Students during COVID-19
- Course Philosophy
- Learning Outcomes

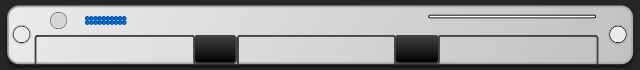
Dynamic Web Pages

HTTP Request


https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

```
GET /swe-432-f21/index.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

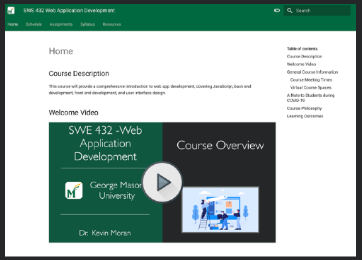
web server



Runs a program

Web Server Application

Syllabus Generator Application



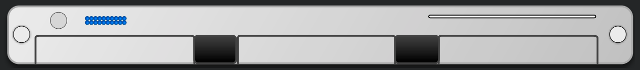
Dynamic Web Pages

https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

HTTP Request

```
GET /swe-432-f21/index.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

web server

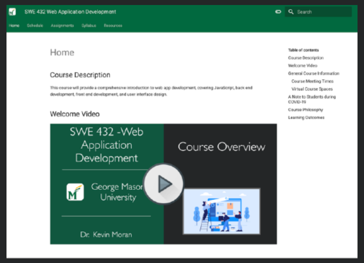


Runs a program

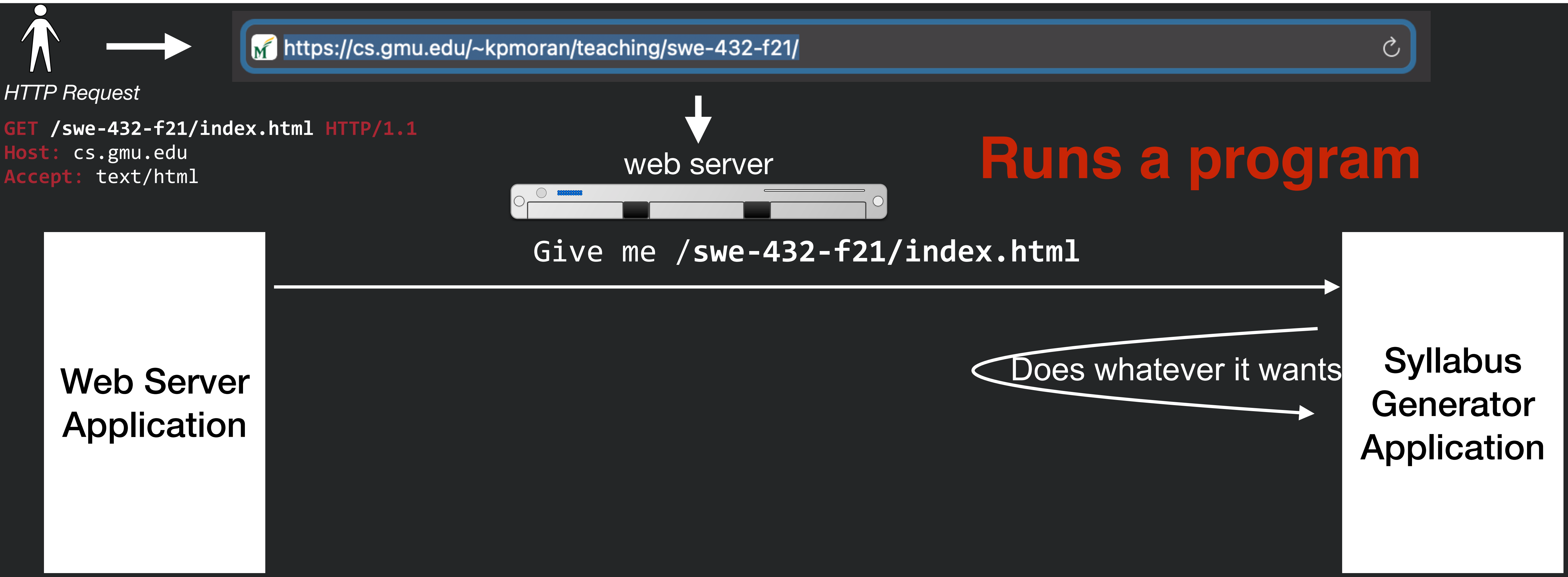
Web Server Application

Give me /swe-432-f21/index.html

Syllabus Generator Application

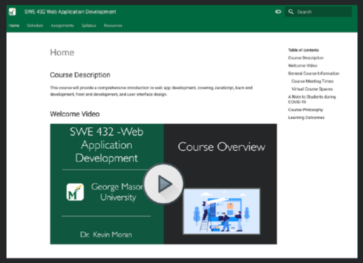


Dynamic Web Pages

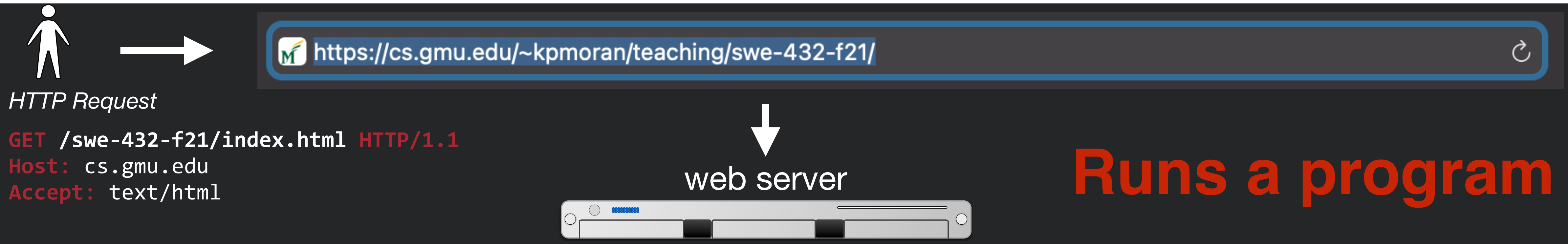

https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/
HTTP Request
web server
Give me /swe-432-f21/index.html
Web Server Application
Does whatever it wants
Syllabus Generator Application

```
GET /swe-432-f21/index.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Runs a program



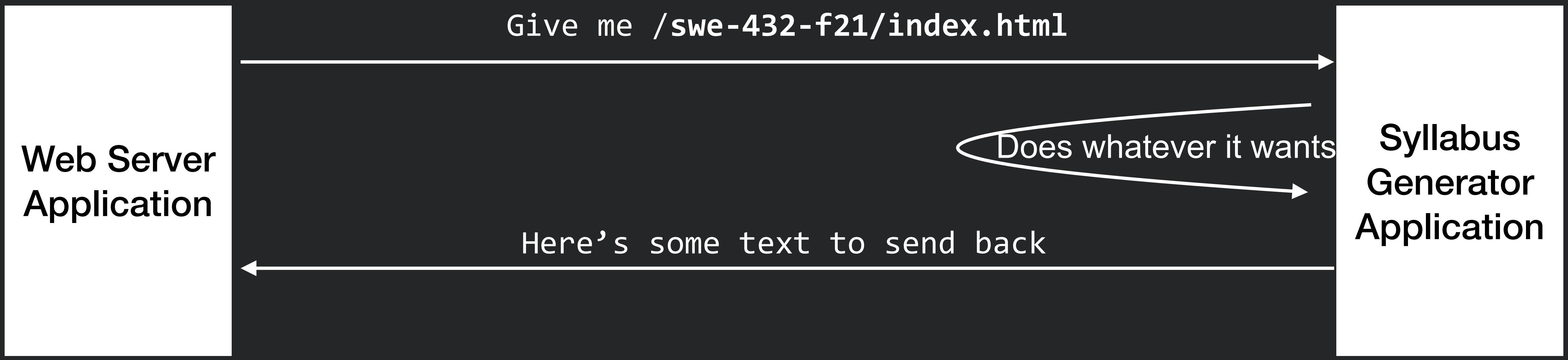
Dynamic Web Pages

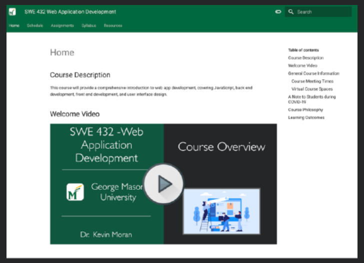


HTTP Request

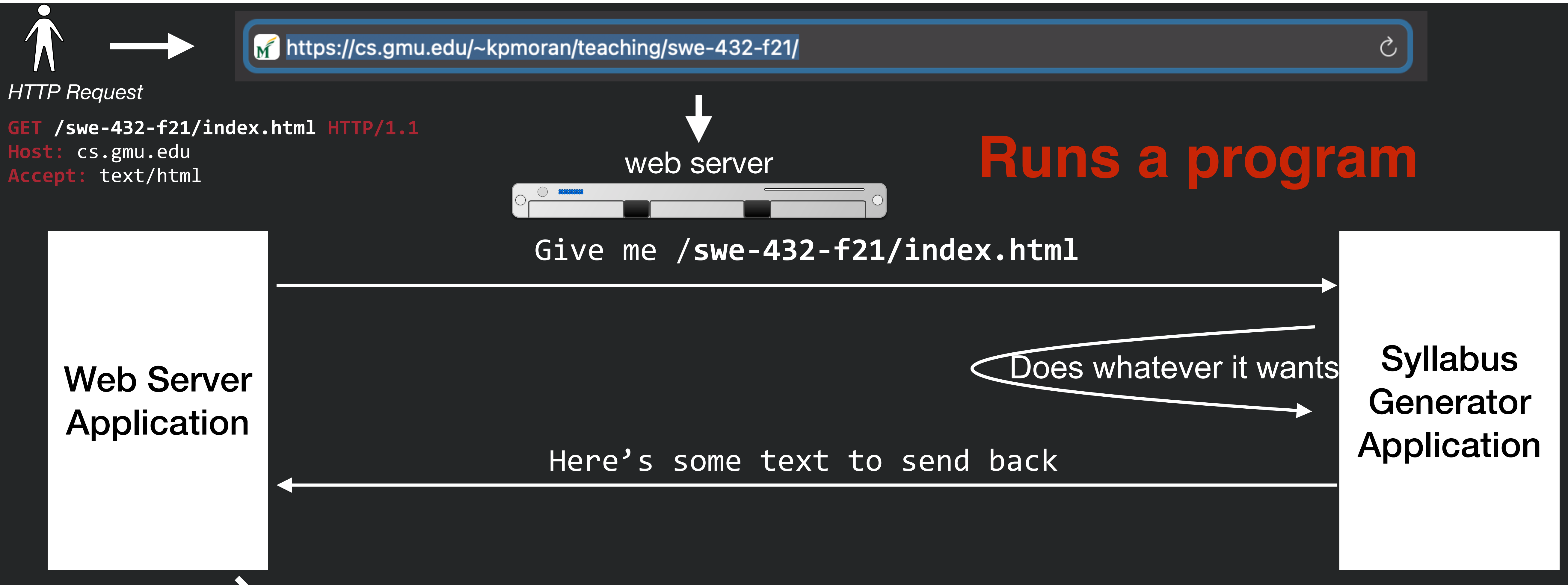
```
GET /swe-432-f21/index.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Runs a program


Web Server Application
Give me /swe-432-f21/index.html
Does whatever it wants
Here's some text to send back
Syllabus Generator Application



Dynamic Web Pages

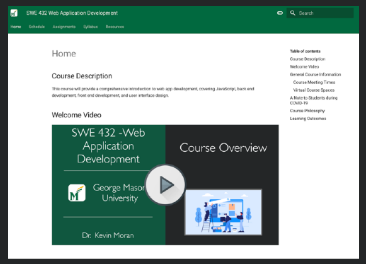

https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/
HTTP Request
web server
Runs a program
Web Server Application
Give me /swe-432-f21/index.html
Does whatever it wants
Syllabus Generator Application
Here's some text to send back

HTTP Request

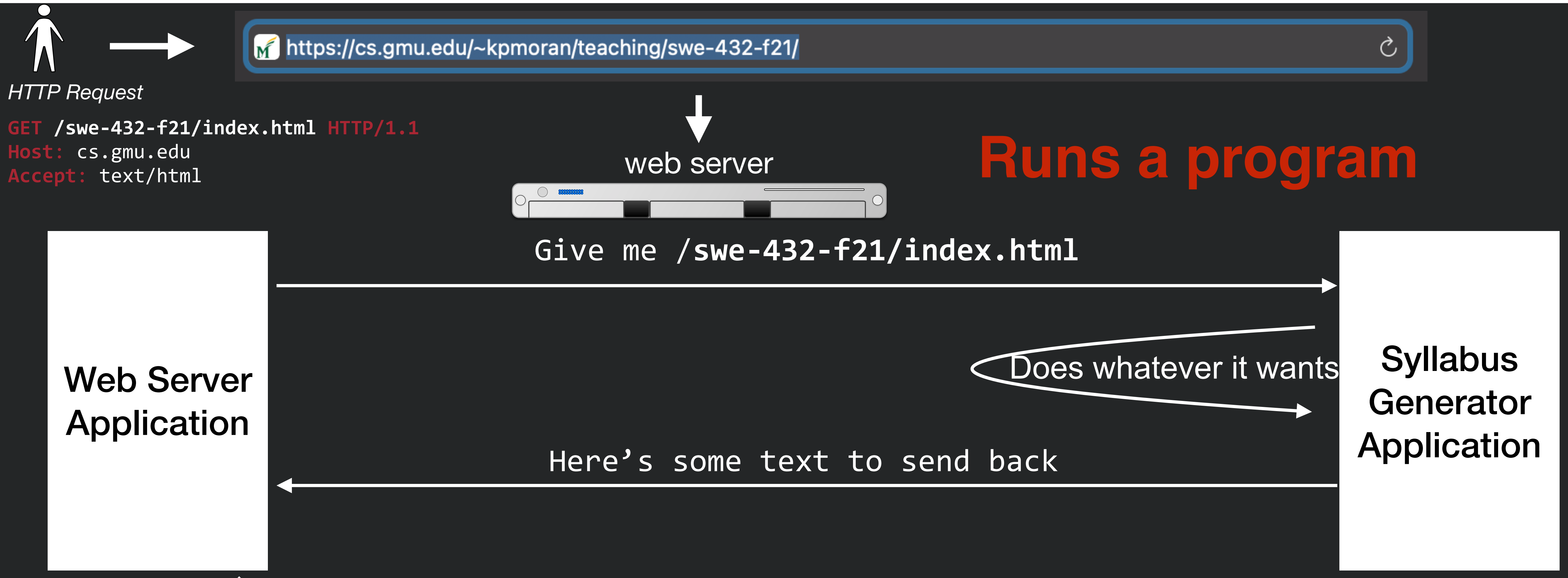
```
GET /swe-432-f21/index.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

HTTP Response

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

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



Dynamic Web Pages


https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/
web server
Web Server Application
Give me /swe-432-f21/index.html
Does whatever it wants
Syllabus Generator Application
Here's some text to send back

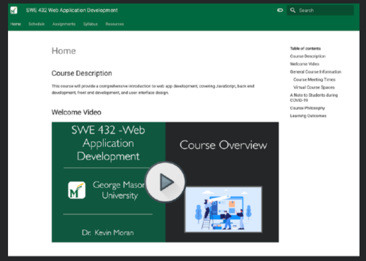
HTTP Request

```
GET /swe-432-f21/index.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Runs a program

HTTP Response

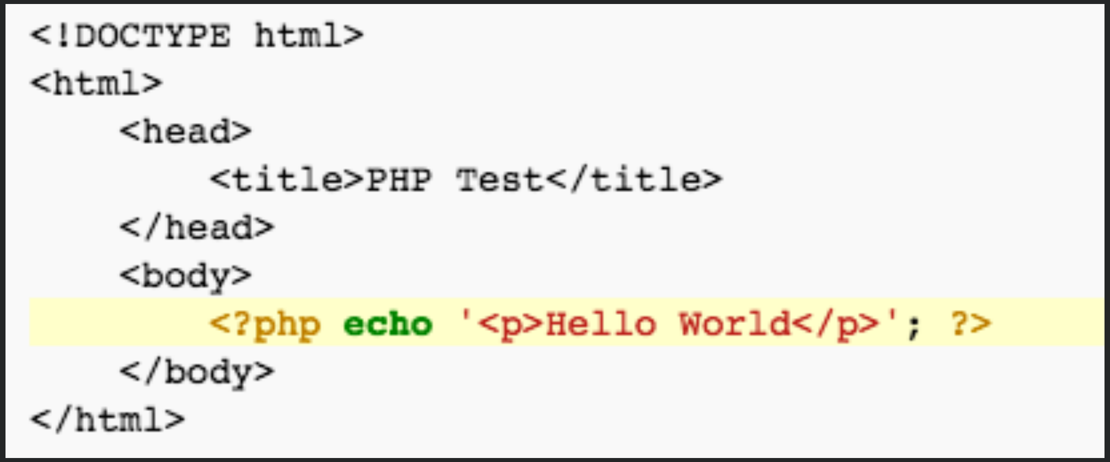
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

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



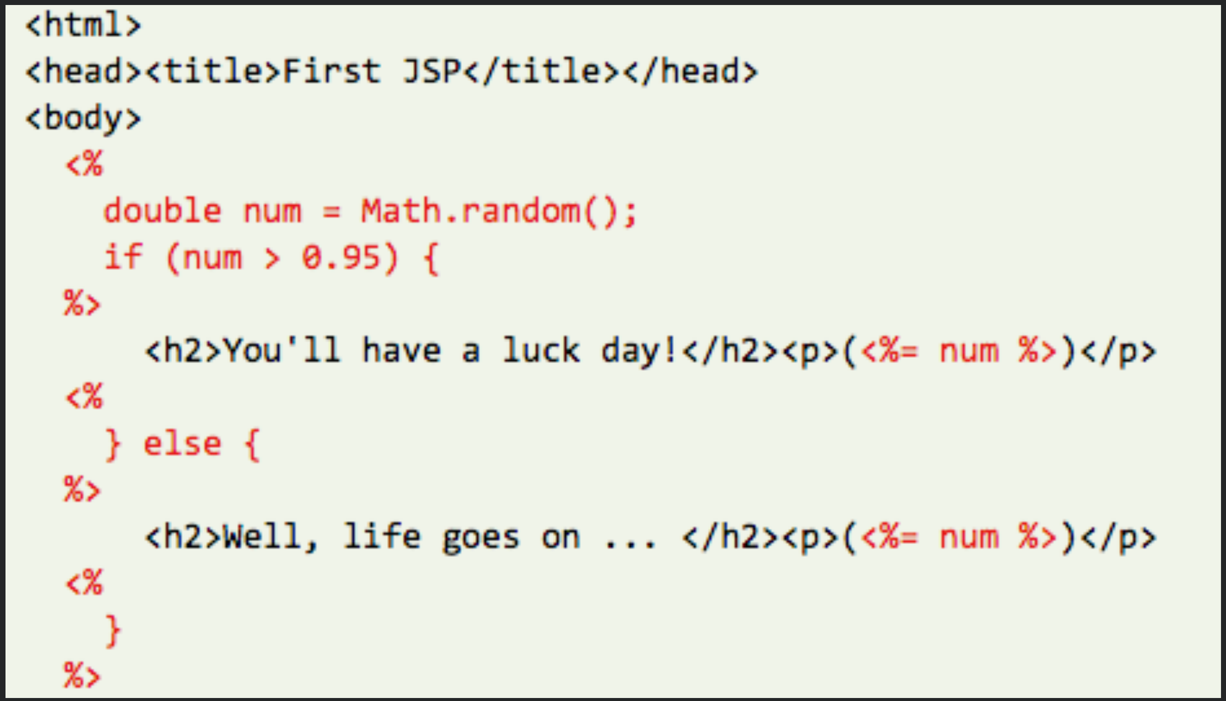
There's a standard mechanism to talk to these auxiliary applications, called CGI (Common Gateway Interface)

Server Side Scripting

- Generate HTML on the server through scripts



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

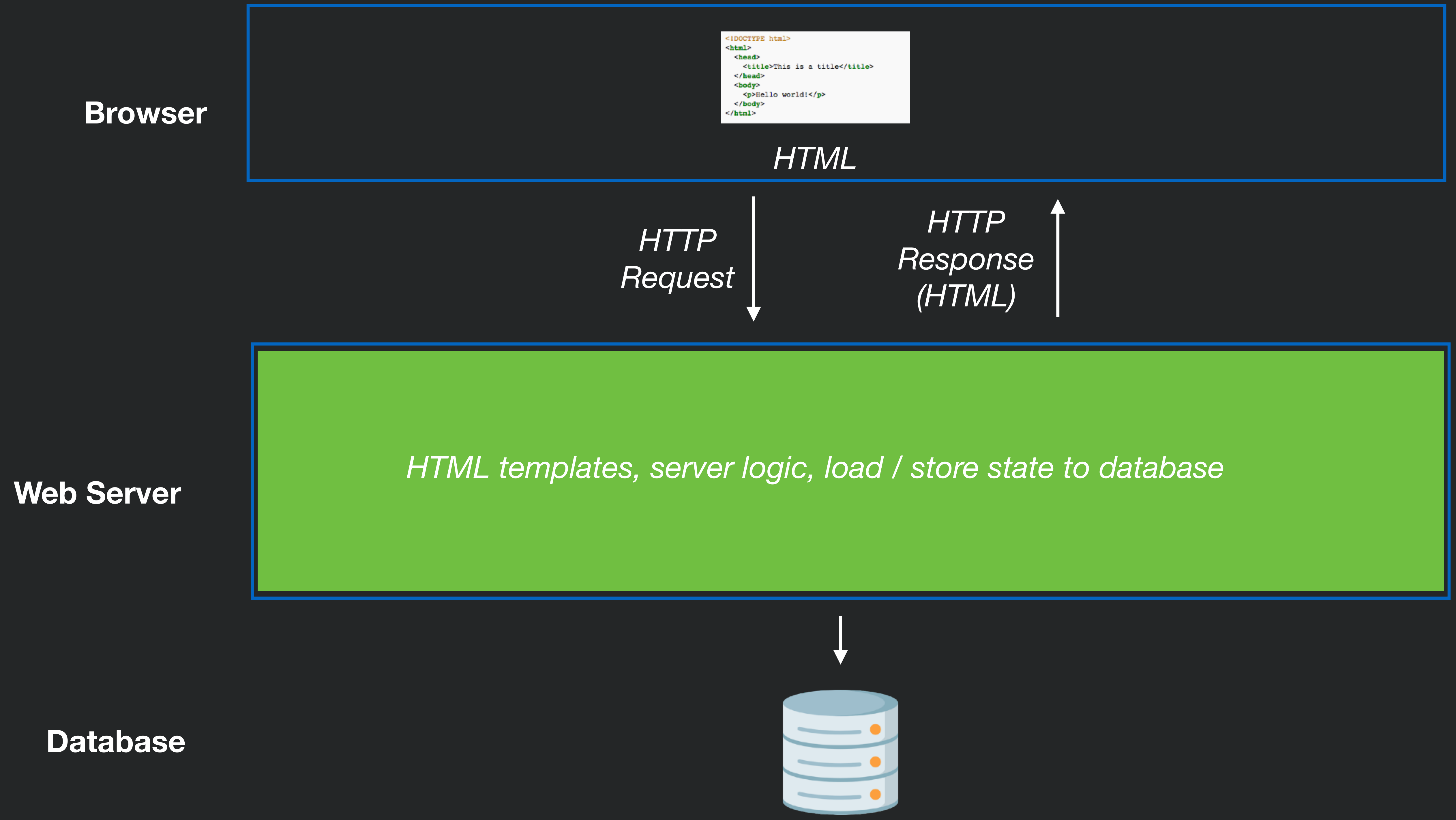


```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

- Early approaches emphasized embedding server code *inside* html pages

- Examples: CGI

Server Side Scripting Site



```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Limitations

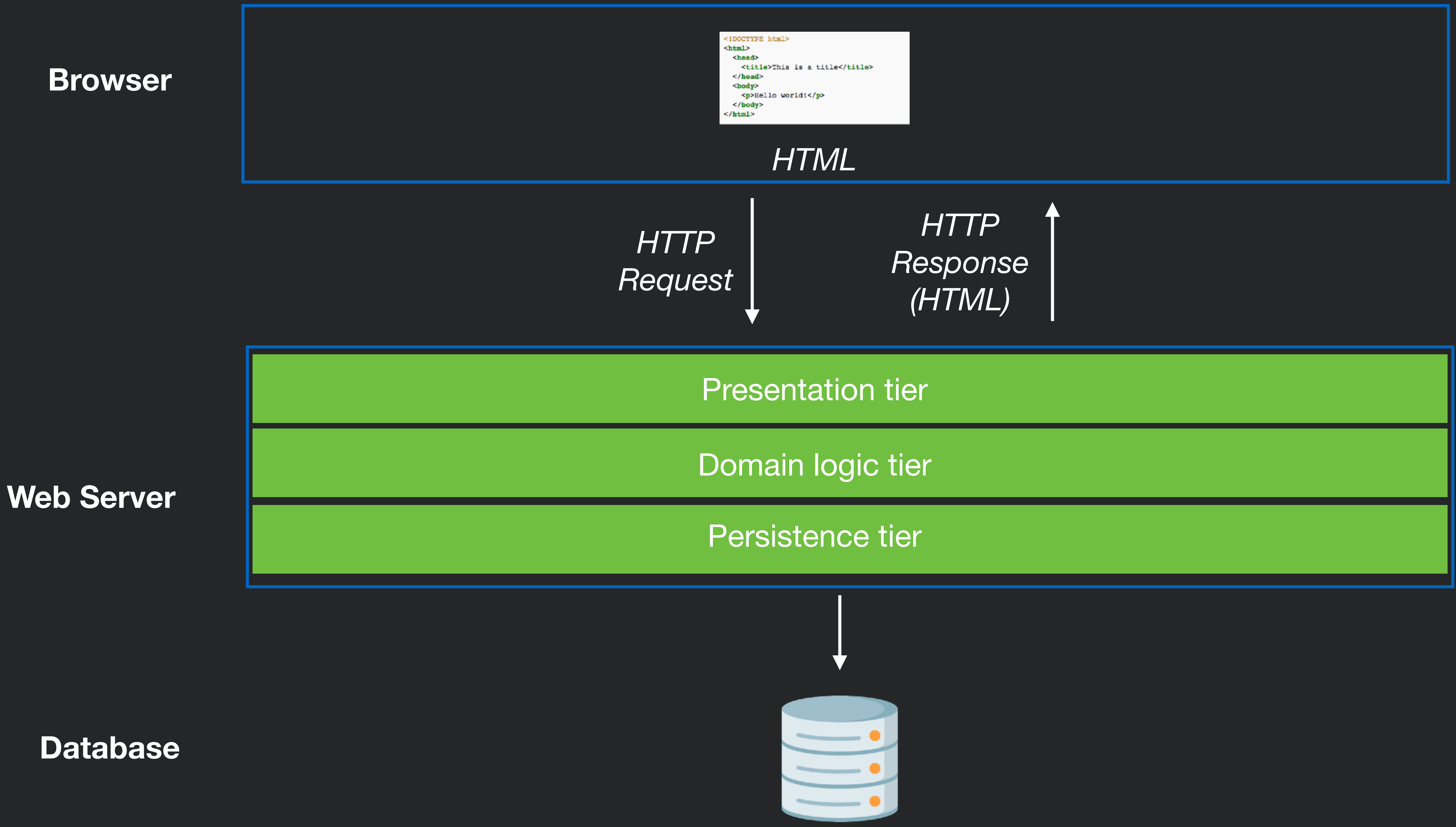
- Poor **modularity**
  - Code representing logic, database interactions, generating HTML presentation all tangled
  - Example of a Big Ball of Mud [1]
  - Hard to understand, difficult to maintain
- Still a step up over static pages!

[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



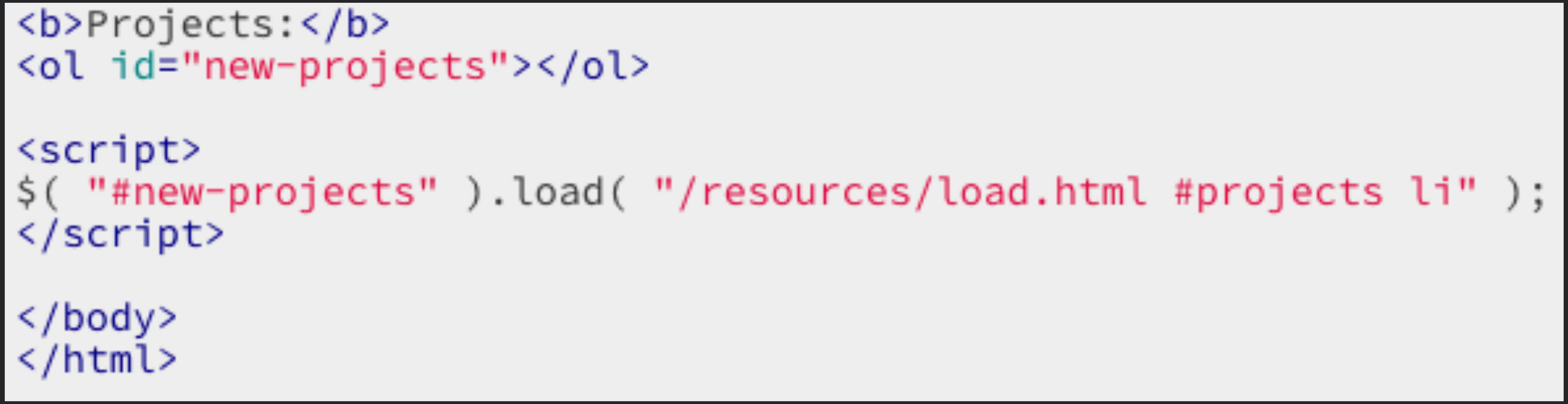
```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

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)

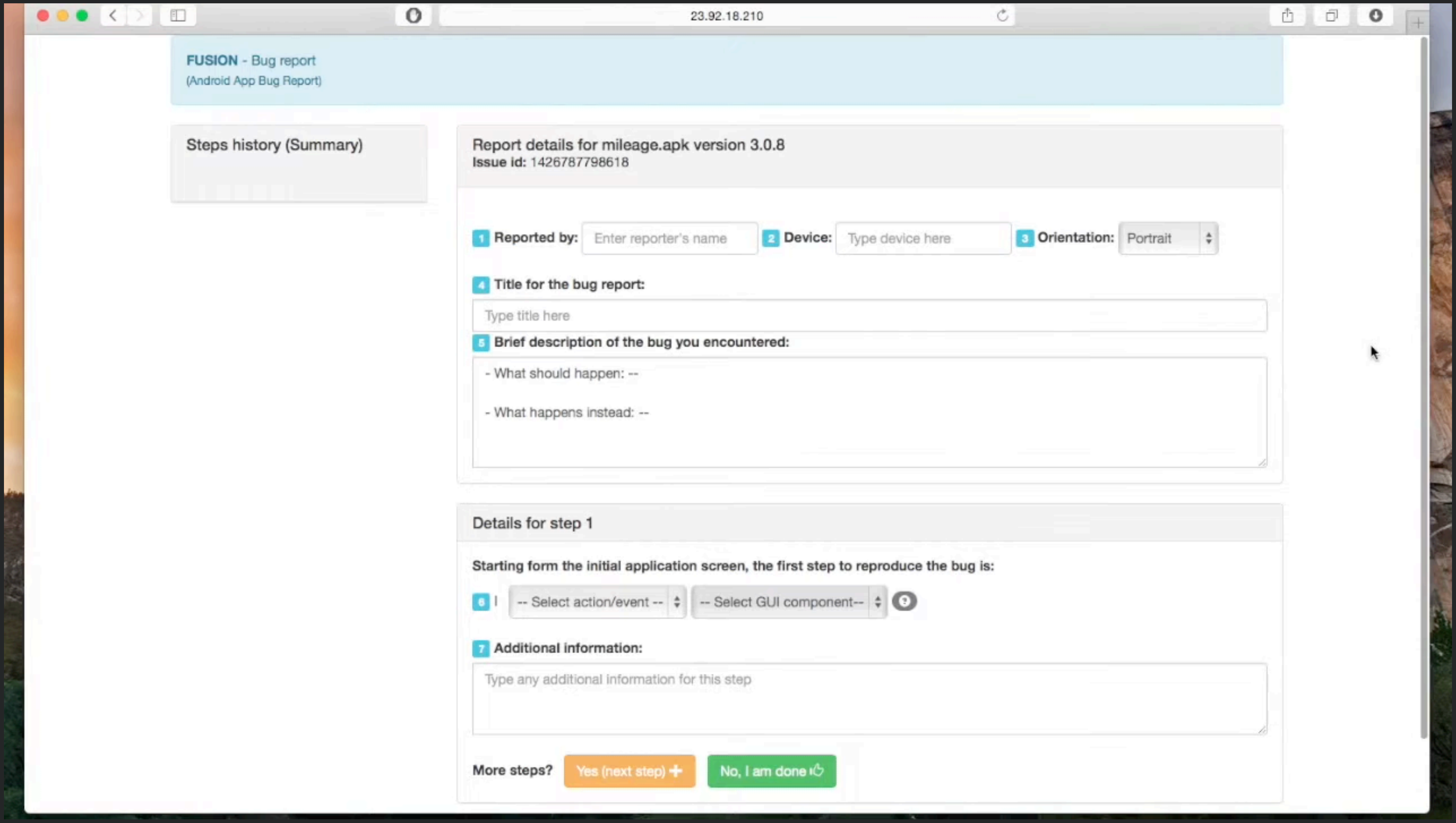
- Client-side logic sends messages to server, receives response
- Logic is associated with a single HTML pages, written in Javascript
- HTML elements dynamically added and removed through DOM manipulation



```
<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

Single Page App Example



23.92.18.210

FUSION - Bug report
(Android App Bug Report)

Steps history (Summary)

Report details for mileage.apk version 3.0.8
Issue id: 1426787798618

1 **Reported by:** Enter reporter's name 2 **Device:** Type device here 3 **Orientation:** Portrait

4 **Title for the bug report:**
Type title here

5 **Brief description of the bug you encountered:**
- What should happen: --

- What happens instead: --

Details for step 1

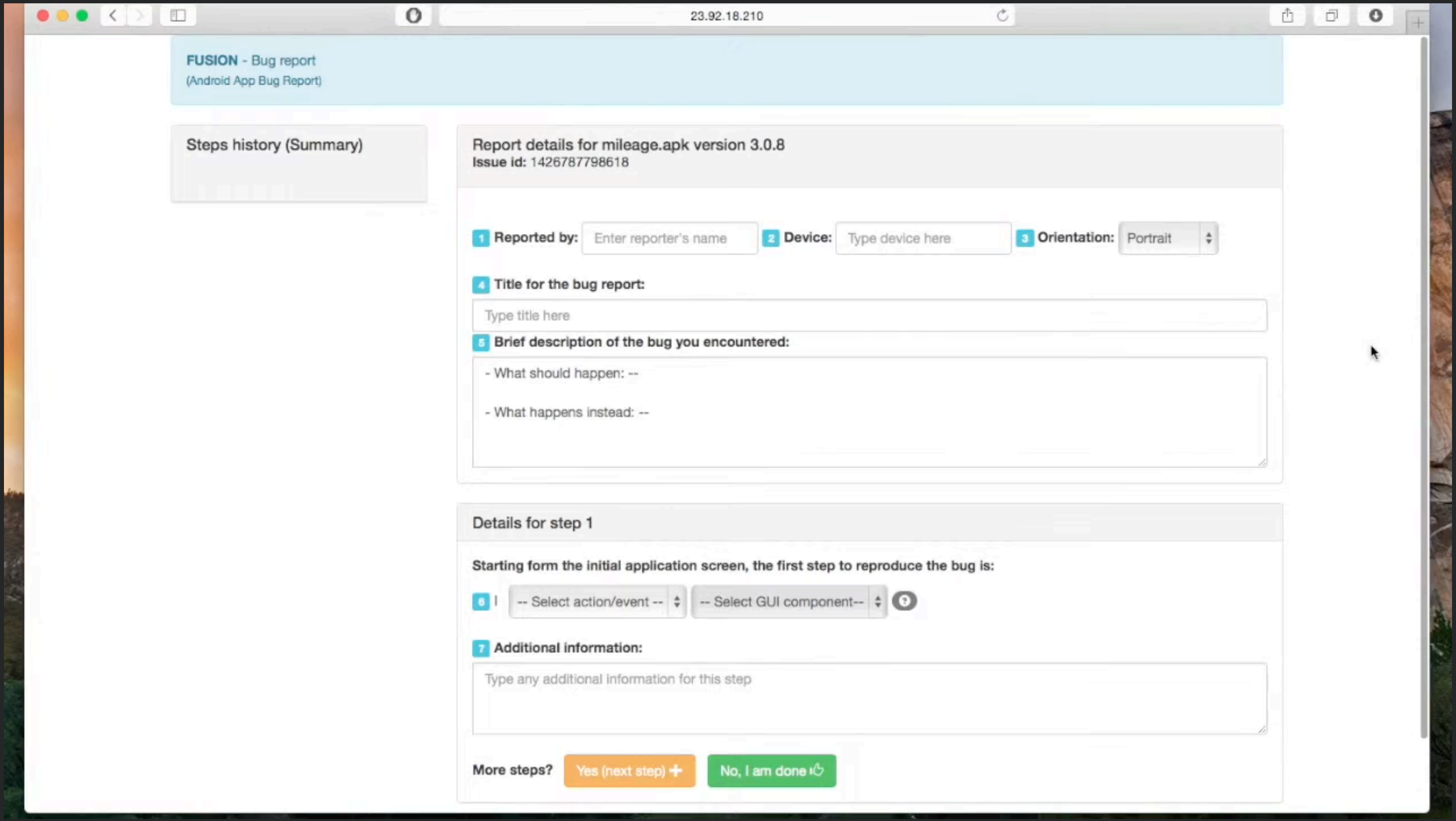
Starting form the initial application screen, the first step to reproduce the bug is:

6 I -- Select action/event -- -- Select GUI component--

7 **Additional information:**
Type any additional information for this step

More steps? Yes (next step) No, I am done

Single Page App Example


23.92.18.210

FUSION - Bug report
(Android App Bug Report)

Steps history (Summary)

Report details for mileage.apk version 3.0.8
Issue id: 1426787798618

1 **Reported by:** Enter reporter's name 2 **Device:** Type device here 3 **Orientation:** Portrait

4 **Title for the bug report:**
Type title here

5 **Brief description of the bug you encountered:**
- What should happen: --
- What happens instead: --

Details for step 1

Starting form the initial application screen, the first step to reproduce the bug is:

6 I -- Select action/event -- -- Select GUI component--

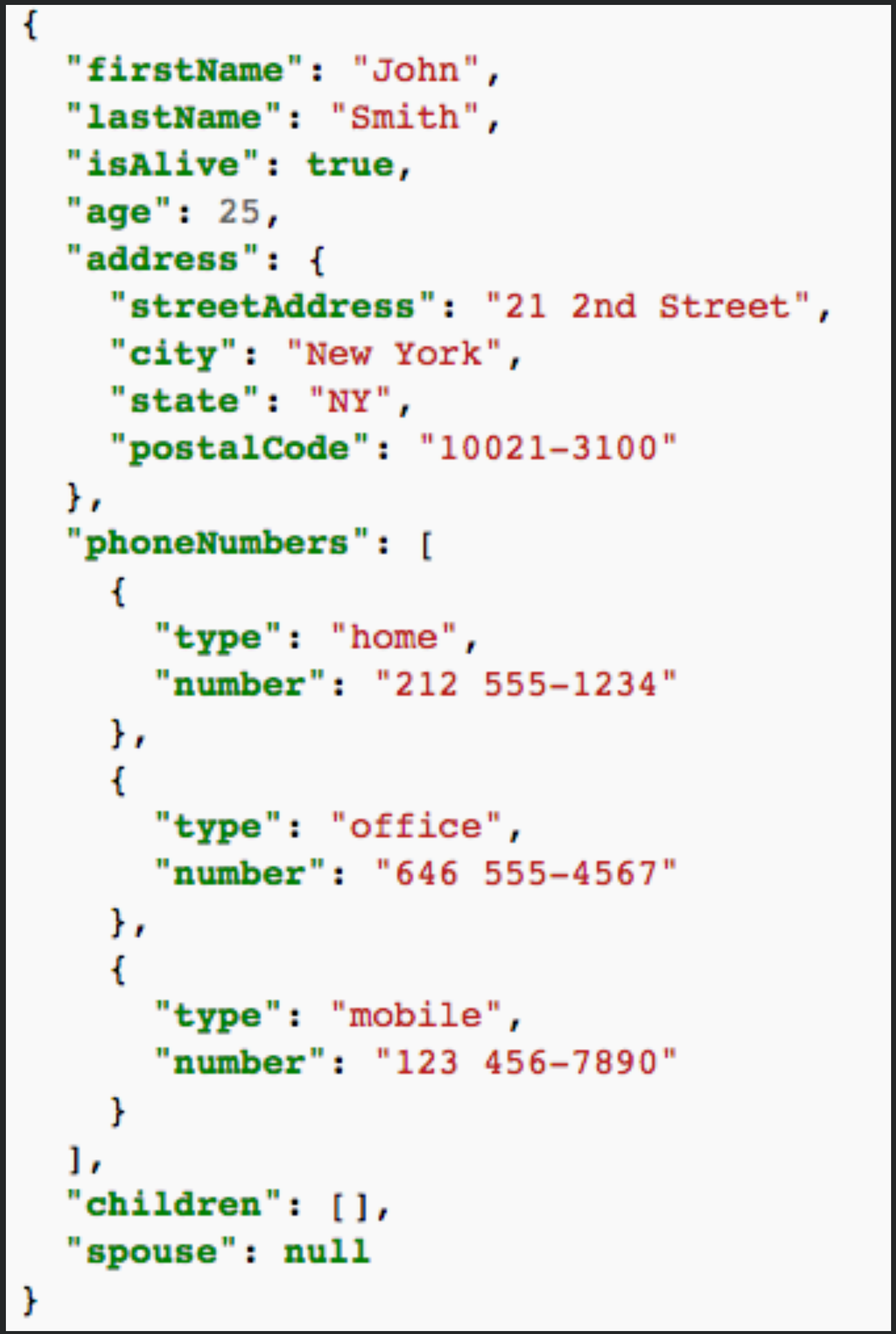
7 **Additional information:**
Type any additional information for this step

More steps? Yes (next step) No, I am done

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

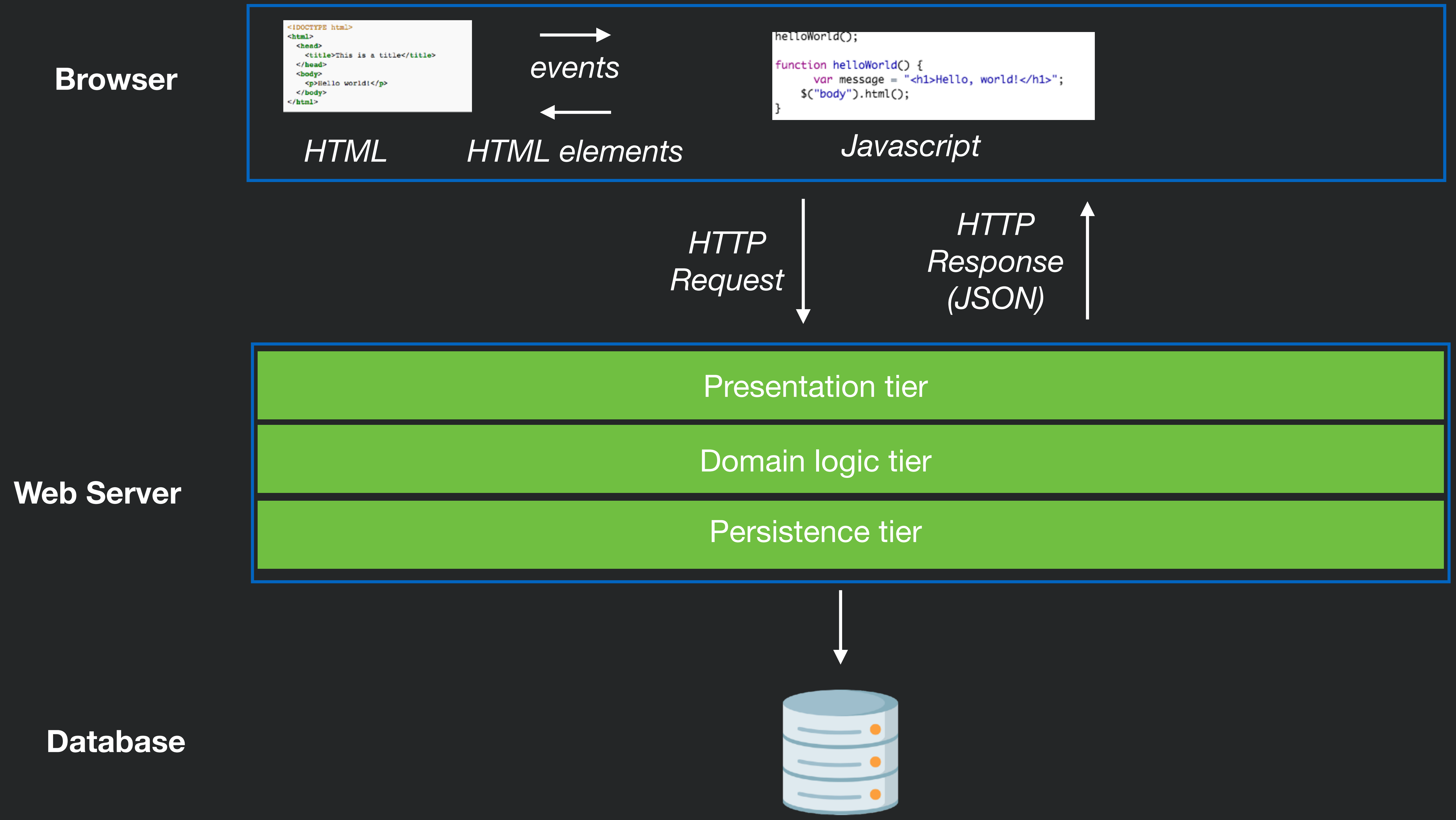
JSON



```
{
  "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



```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

```
helloWorld();

function helloWorld() {
        var message = "<h1>Hello, world!</h1>";
     $("body").html();
}
```

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

Front End Frameworks

- Client is organized into separate *components,* capturing model of web application data

Front End Frameworks

- Client is organized into separate *components,* capturing model of web application data
- Components are reusable, have encapsulation boundary (e.g., class)

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*

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

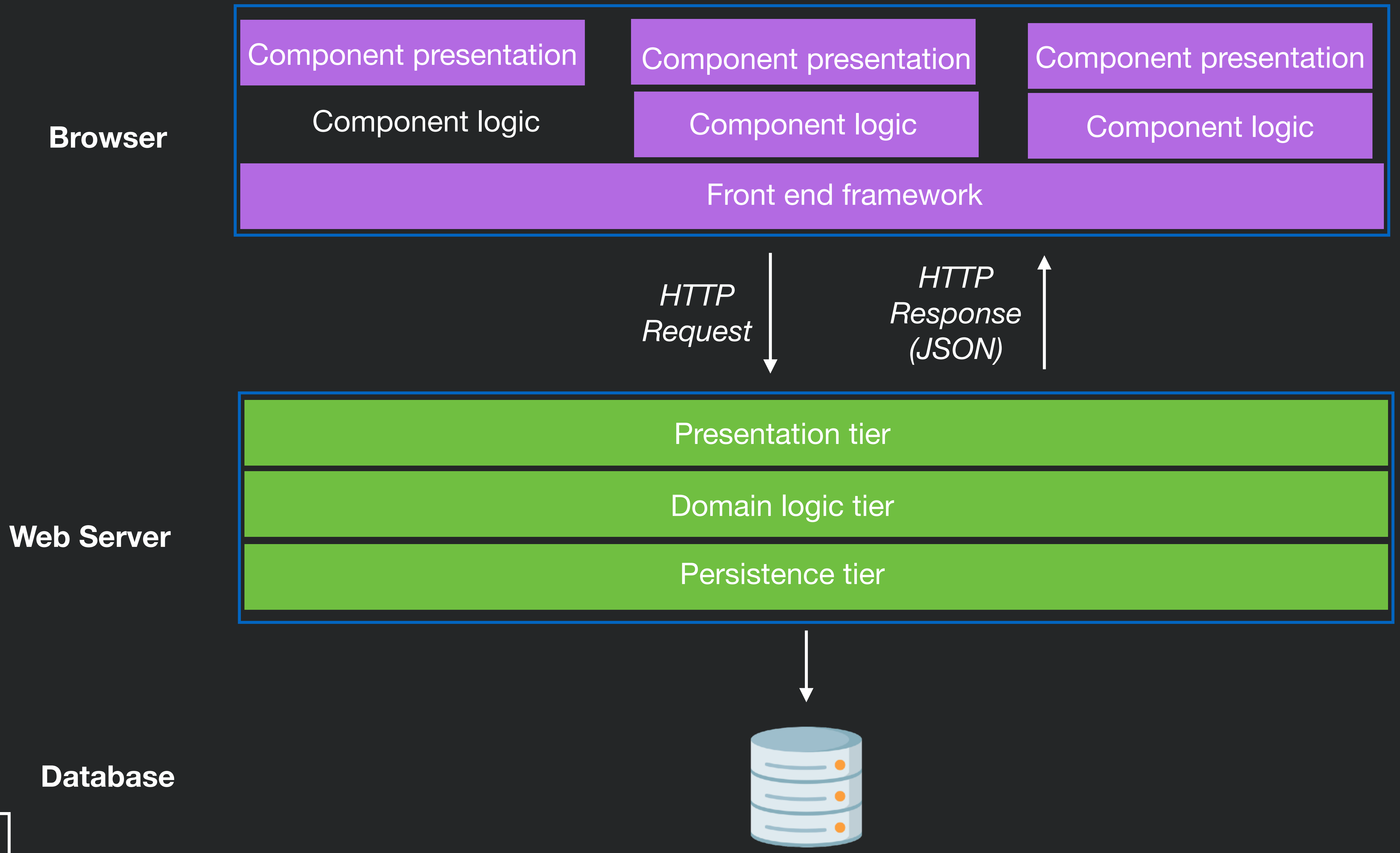
Front End Frameworks

- 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

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



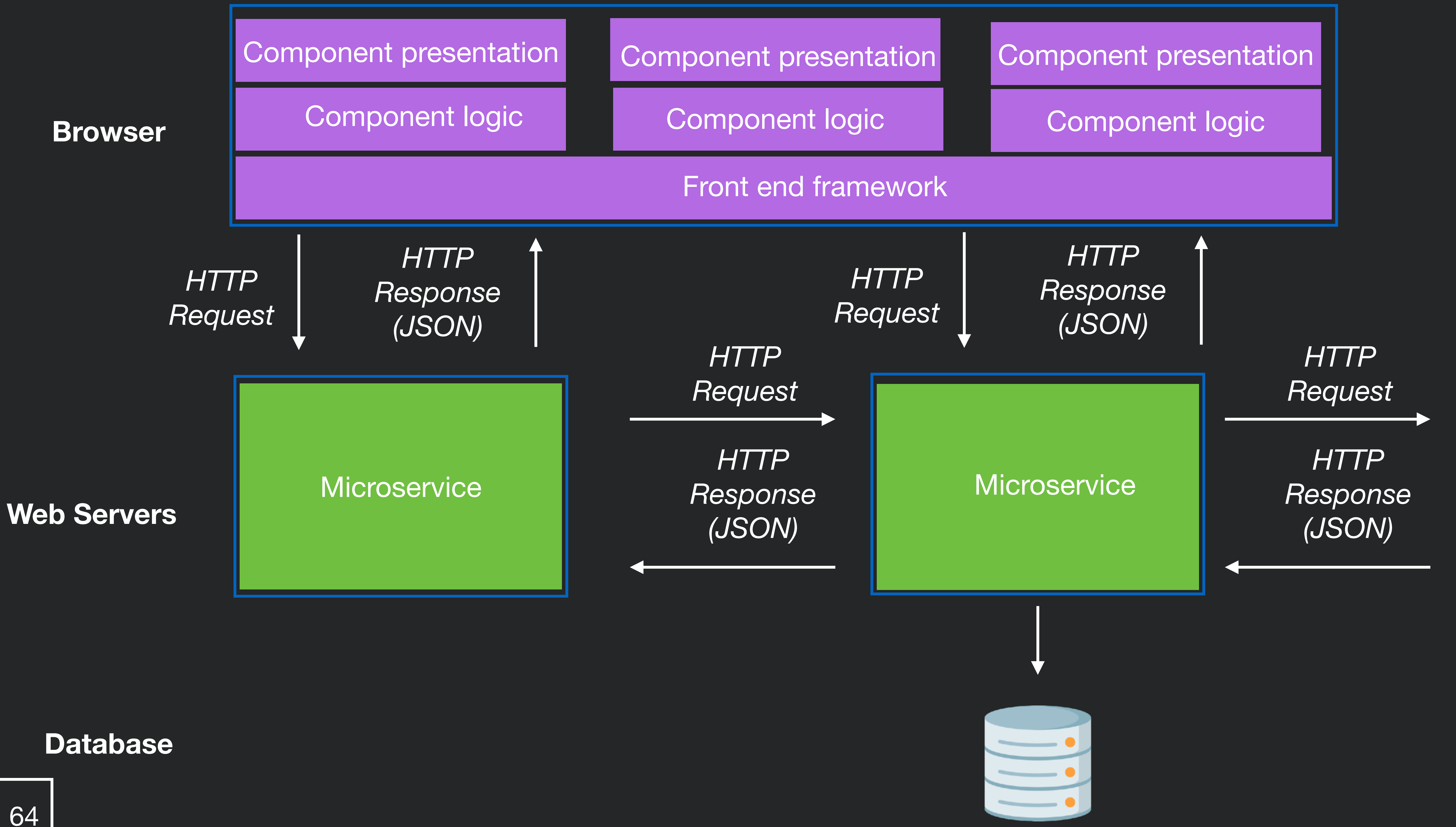
Limitations

- 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

Constant change in web architectural styles

- Key drivers

Constant change in web architectural styles

- Key drivers
  - Maintainability (new ways to achieve better modularity)

Constant change in web architectural styles

- Key drivers
  - Maintainability (new ways to achieve better modularity)
  - Reuse (organizing code into modules)

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)

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)

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)

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

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

The web today

- Many technologies for each architectural style
    - Most support more than one

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

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

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.

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.

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.

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.

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.

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.

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

What is this Course?

- **What is this course?**
  - Three main parts:
    - *Learn* Foundational web development knowledge
    - *Experience* popular web programming Frameworks/Tools
    - *Explore* fundamentals of good Web App Design

10 Minute Break

SWE 432 - Web Application Development

George Mason University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
David Gonzalez Samudio

Class will start in:

10:01

SWE 432 - Web Application Development

George Mason University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
David Gonzalez Samudio

Class will start in:

10:01

Introduction to Javascript

Course Timeline

You are here. →

- JavaScript and Backend development (first half of semester)
    - JavaScript, back-end development, programming models, testing, performance, privacy, security, scalability, deployment, etc.
- Frontend development and user experience design (second half of semester)
    - Templates and data binding, React, user-centered design, user studies, information visualization, visual design, etc.

This Lecture

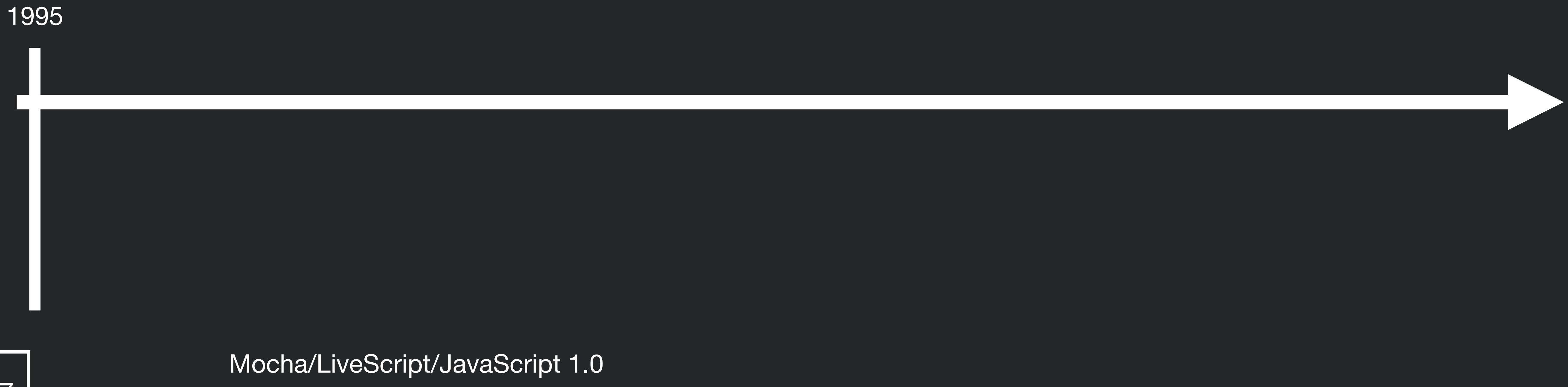
- Brief history of JavaScript/ECMAScript
- Overview of core syntax and language semantics
- Overview of key libraries
- In class activity working with JavaScript
- Next:
    - Testing and tooling

JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that’s all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language

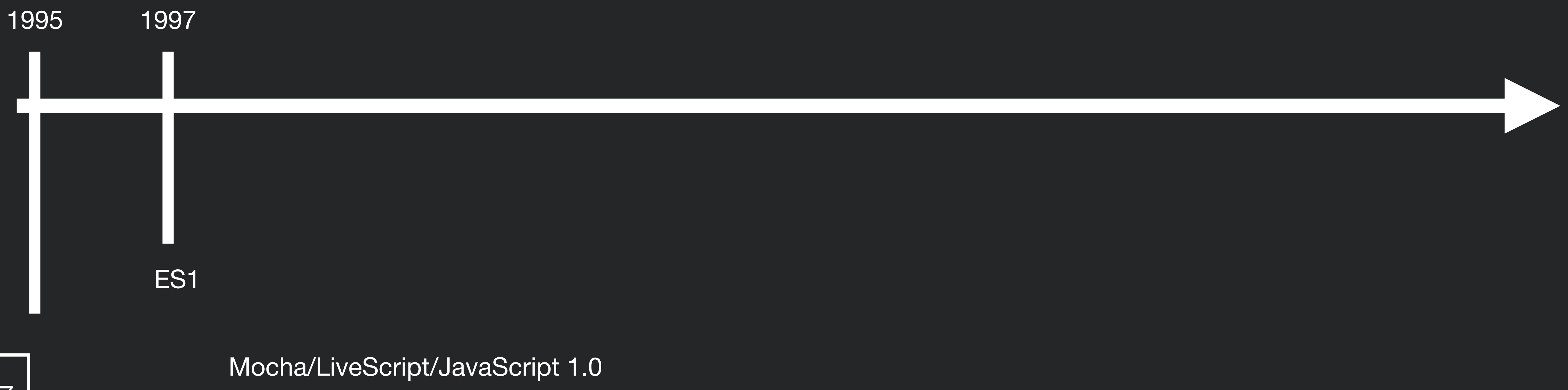
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
    - No relation to Java (maybe a little syntax, that's all)
    - Naming was marketing ploy
- ECMAScript -> International standard for the language



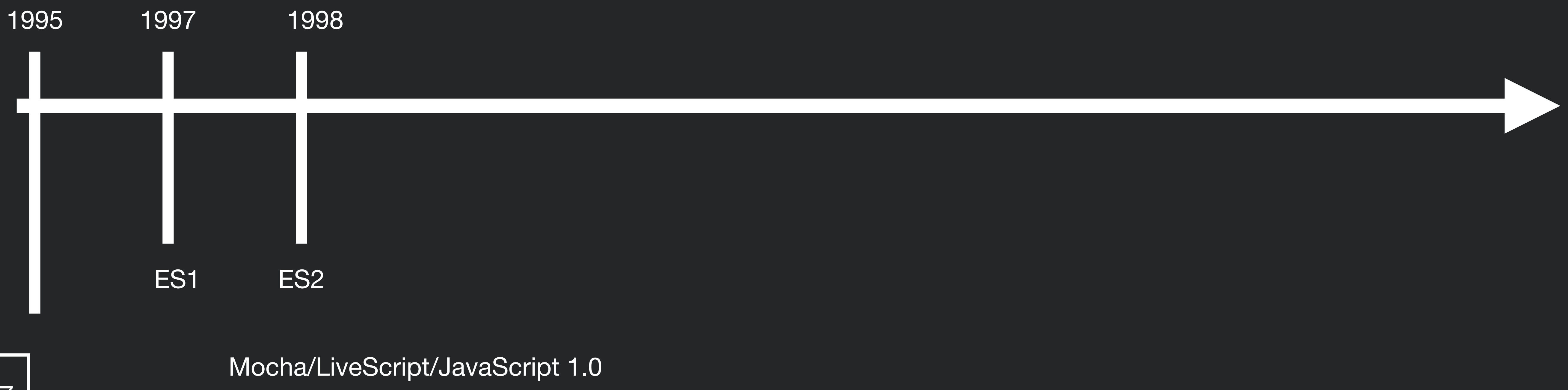
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that's all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language



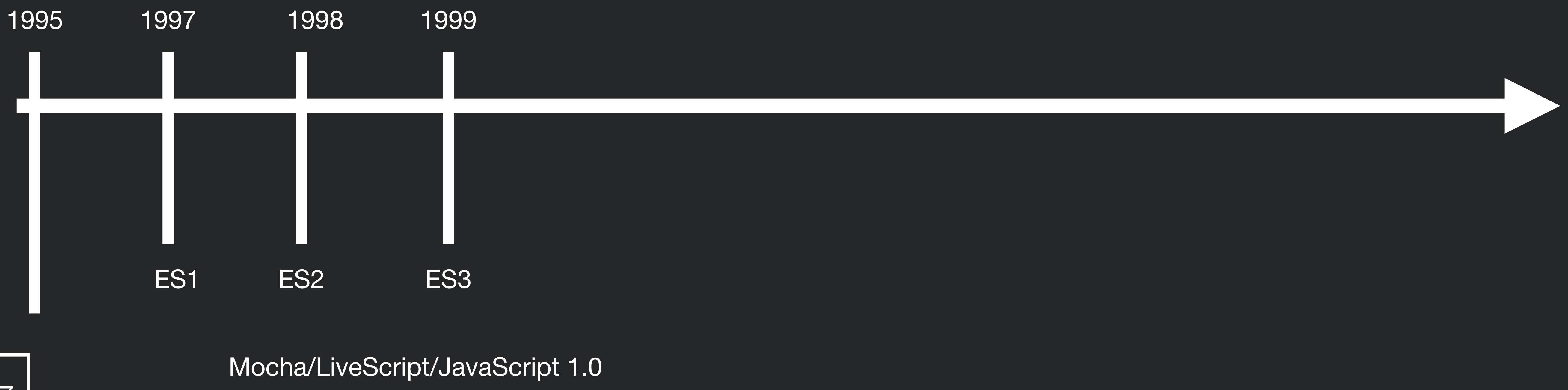
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that's all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language



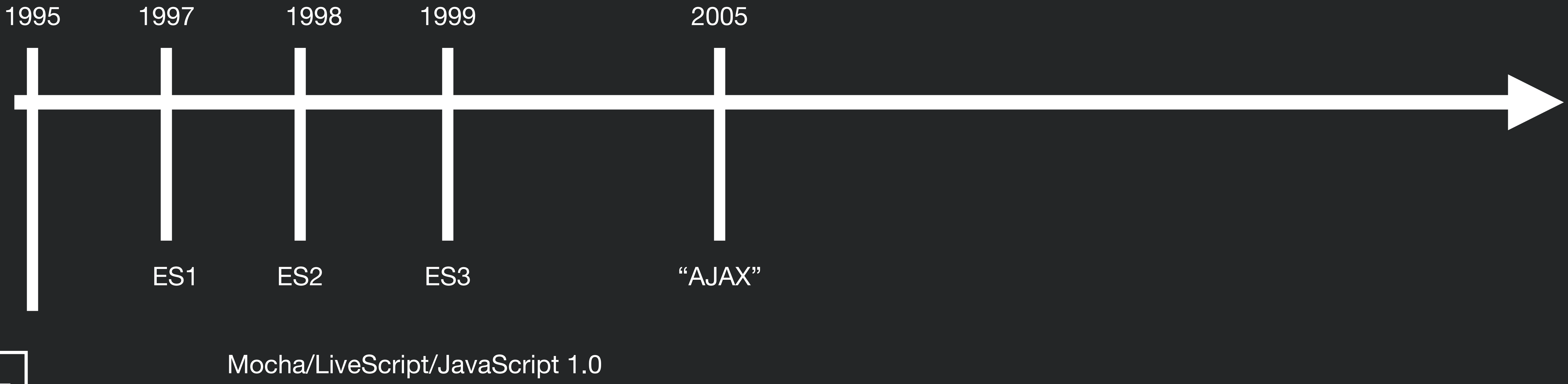
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that’s all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language



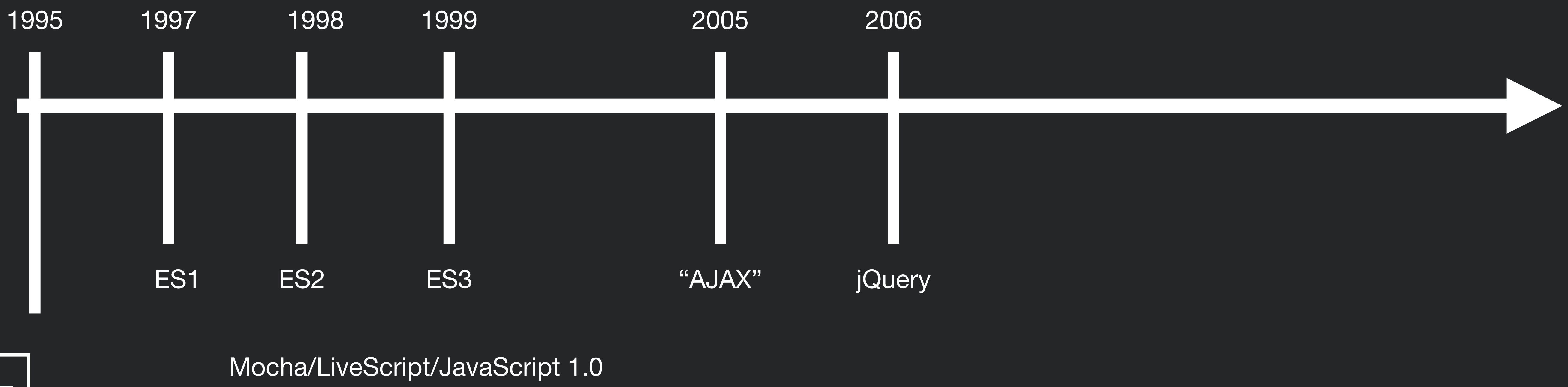
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that's all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language



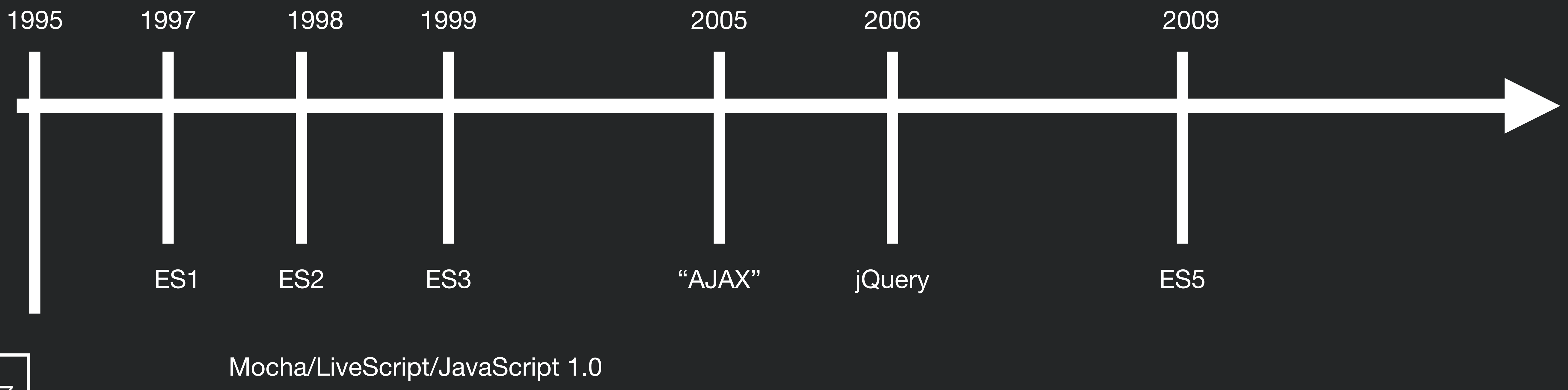
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that's all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language



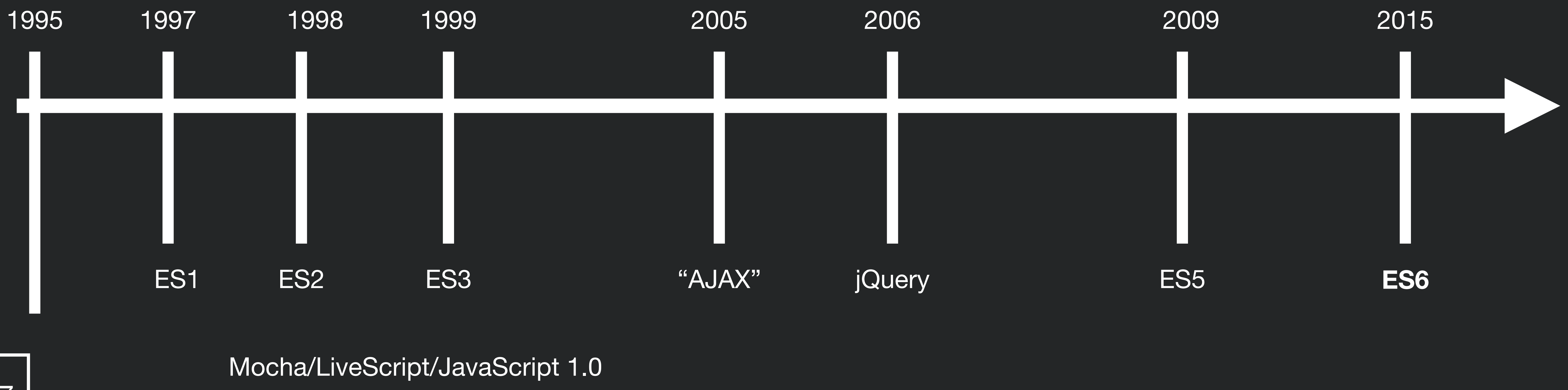
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
  - No relation to Java (maybe a little syntax, that's all)
  - Naming was marketing ploy
- ECMAScript -> International standard for the language



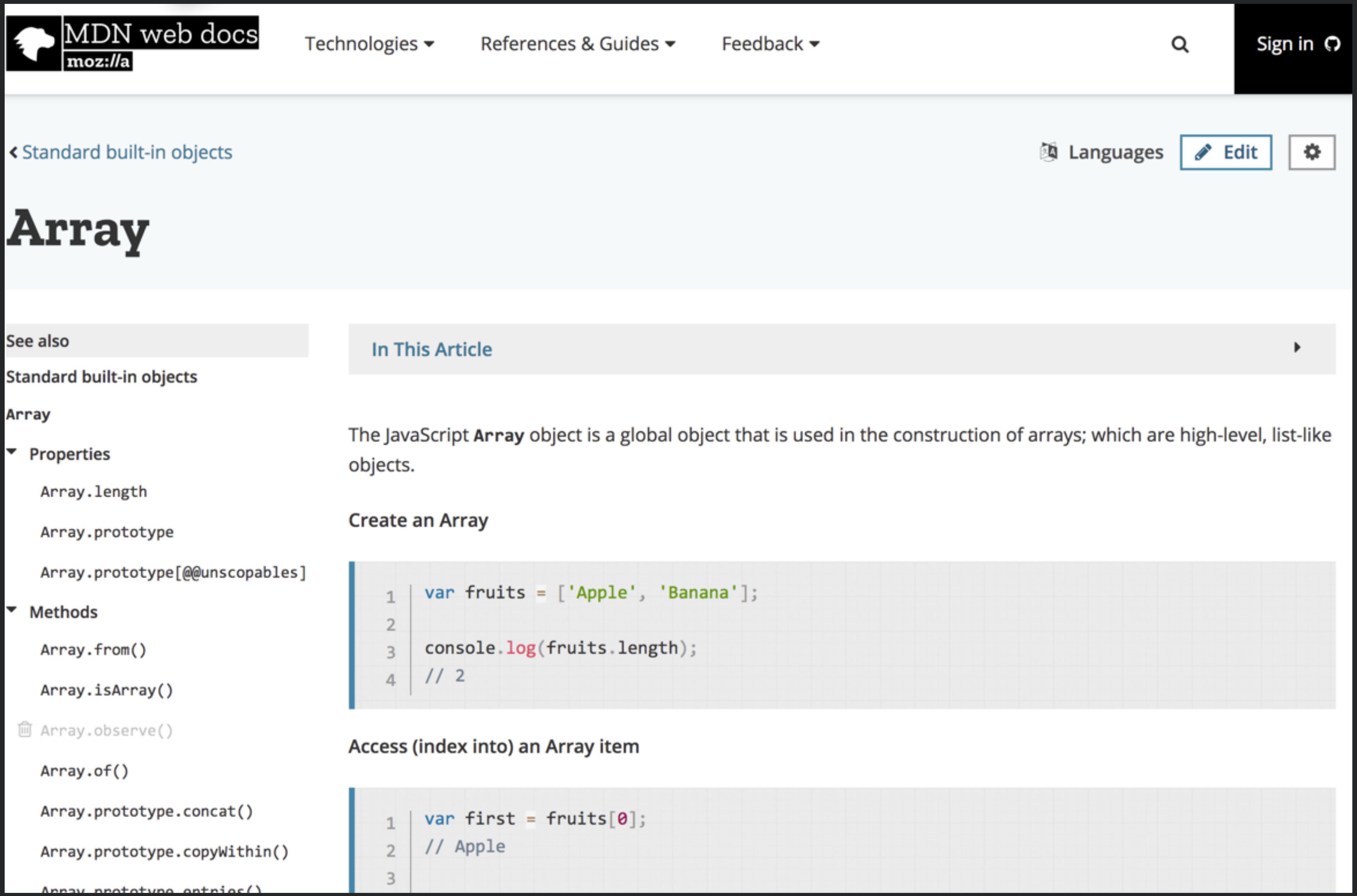
JavaScript: Some History

- JavaScript: 1995 at Netscape (supposedly in only 10 days)
    - No relation to Java (maybe a little syntax, that's all)
    - Naming was marketing ploy
- ECMAScript -> International standard for the language



Reference materials

- Not any "official" documentation
- Most definitive source for JavaScript, DOM, HTML, CSS: Mozilla Development Network (MDN)
- StackOverflow posts, blogs often have good examples



Array

The JavaScript **Array** object is a global object that is used in the construction of arrays; which are high-level, list-like objects.

Create an Array

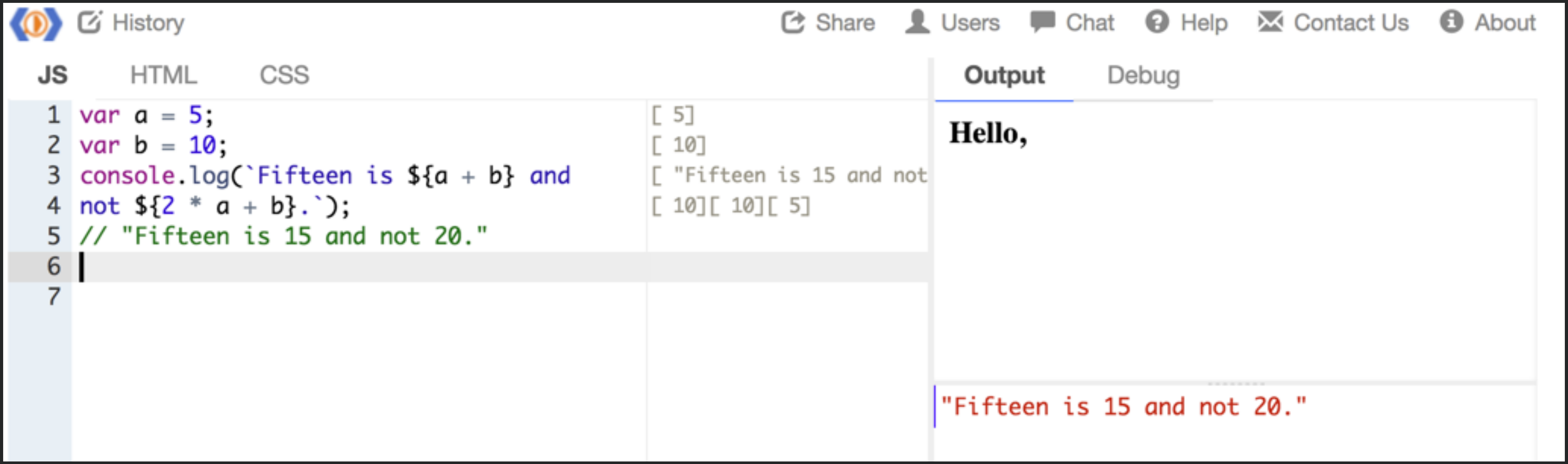
```
var fruits = ['Apple', 'Banana'];

console.log(fruits.length);
// 2
```

Access (index into) an Array item

```
var first = fruits[0];
// Apple

```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Pastebins



```
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
// "Fifteen is 15 and not 20."
```

```
[ 5]
[ 10]
[ "Fifteen is 15 and not
[ 10][ 10][ 5]
```

```
Hello,
```

```
"Fifteen is 15 and not 20."
```

- Code snippet hosted on the web with an in-browser editor
- Used to share code and experiment with small code snippets
- Examples: JSFiddle, JSBin, Replit, Codesandbox

Variables

- Variables are *loosely* typed
  - String:
    ```
    var strVar = 'Hello';
    ```
  - Number:
    ```
    var num = 10;
    ```
  - Boolean:
    ```
    var bool = true;
    ```
  - Undefined:
    ```
    var undefined;
    ```
  - Null:
    ```
    var nulled = null;
    ```
  - Objects (includes arrays):
    ```
    var intArray = [1,2,3];
    ```
  - Symbols (named magic strings):
    ```
    var sym = Symbol(‘Description of the symbol’);
    ```
  - Functions (We’ll get back to this)
- Names start with letters, $ or _
- Case sensitive

Const

- Can define a variable that cannot be assigned again using const

```
const numConst = 10; //numConst can’t be
changed
```

- For objects, properties may change, but object identity may not.

More Variables

- Loose typing means that JS figures out the type based on the value

```
 let x; //Type: Undefined
x = 2; //Type: Number
x = 'Hi'; //Type: String
```

- Variables defined with let (but not var) have block scope
  - If defined in a function, can only be seen in that function
  - If defined outside of a function, then global. Can also make arbitrary blocks:

```
{
    let a = 3;
}
//a is undefined
```

Loops and Control Structures

- `if` - pretty standard

```
if (myVar >= 35) {
    //...
} else if(myVar >= 25){
    //...
} else {
    //...
}
```

- Also get `while`, `for`, and `break` as you might expect

```
while(myVar > 30){
    //...
}

for(var i = 0; i < myVar; i++){
     //...
     if(someOtherVar == 0)
      break;
}
```

Operators

Operators

Operators

Operator	Meaning	Examples

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

!=	Inequality	`age != 21`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

!=	Inequality	`age != 21`
>	Greater than	`age > 19`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`
<=	Less or equal	`age <= 20`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples

==	Equality	`age == 20`
		`age == '20'`

!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`
<=	Less or equal	`age <= 20`
===	Strict equal	`age === 20`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	age == 20 age == '20'
!=	Inequality	age != 21
>	Greater than	age > 19
>=	Greater or Equal	age >= 20
<	Less than	age < 21
<=	Less or equal	age <= 20
===	Strict equal	age === 20
!==	Strict Inequality	age !== '20'

Annoying

Functions

- At a high level, syntax should be familiar:

```
function add(num1, num2) {
    return num1 + num2;
}
```

- Calling syntax should be familiar too:

```
var num = add(4,6);
```

- Can also assign functions to variables!

```
var magic = function(num1, num2){
    return num1+num2;
}
var myNum = magic(4,6);
```

- Why might you want to do this?

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

   var r = add(); // 55
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

   var r = add(); // 55
   var r = add(40); //85
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

   var r = add(); // 55
   var r = add(40); //85
   var r = add(2,4); //6
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}


           add(40,10,20); //70
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

```
var add = (a,b) => {
    return a+b;
}
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

Parameters

```
var add = (a,b) => {
        return a+b;
}
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

Parameters

```
    var add = (a,b) => {
        return a+b;
    }


var add = (a,b) => a+b;
```

If your arrow function only has one expression, JavaScript will automatically add the word “return”

Objects

Objects

- What are objects like in other languages? How are they written and organized?

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*
- Remember - JS is not really typed… if it doesn't care between a number and a string, why care between two kinds of objects?

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*
- Remember - JS is not really typed… if it doesn't care between a number and a string, why care between two kinds of objects?

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

```
console.log(profHacker.fullName);
```

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

```
console.log(profHacker.fullName);//function...
```

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

~~console.log(profHacker.fullName~~);//function...

JSON: JavaScript Object Notation

Open standard format for transmitting *data* objects.

- No functions, only key / value pairs
- Values may be other objects or arrays

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: “ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409",
    fullName: {
        firstName: “Alyssa”,
        lastName: “P Hacker”}
};
```

JSON Object

Interacting w/ JSON

- Important functions
- JSON.parse(jsonString)
  - Takes a *String* in JSON format, creates an *Object*
- JSON.stringify(obj)
  - Takes a Javascript *object*, creates a JSON *String*
- Useful for persistence, interacting with files, debugging, etc.
  - e.g., console.log(JSON.stringify(obj));

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
```

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
```

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
```

Arrays are actually objects… and come with a bunch of “free” functions

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
var classMembers = students.concat(faculty);
```

Arrays are actually objects… and come with a bunch of “free” functions

Some Array Functions

- Length

```
var numberOfStudents = students.length;
```

- Join

```
var classMembers = students.concat(faculty);
```

- Sort

```
var sortedStudents = students.sort();
```

- Reverse

```
var backwardsStudents = sortedStudents.reverse();
```

- Map

```
var capitalizedStudents = students.map(x =>
                                   x.toUpperCase());
  //["ALICE","BOB","CAROL"]
```

For Each

For Each

- JavaScript offers two constructs for looping over arrays and objects

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
     console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

```
for(var prop in profHacker){
     console.log(prop + ": " + profHacker[prop]);
}
```

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

```
for(var prop in profHacker){
    console.log(prop + ": " + profHacker[prop]);
}
```

Output:

```
firstName: Alyssa
lastName: P Hacker
teaches: SWE 432
office: ENGR 6409
```

Arrays vs Objects

- Arrays are Objects
- Can access elements of both using syntax

```
var val = array[idx];
```

- Indexes of arrays must be integers
- Don’t find out what happens when you make an array and add an element with a non-integer key :)

String Functions

- Includes many of the same String processing functions as Java
- Some examples
  - var stringVal = ‘George Mason University’;
  - stringVal.endsWith(‘University’)   // returns true
  - stringVal.match(….)   // matches a regular expression
  - stringVal.split(‘ ‘) // returns three separate words

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Template Literals

- Enable embedding expressions **inside** strings

```
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
// "Fifteen is 15 and not 20."
```

- Denoted by a back tick grave accent `, **not** a single quote

Set Collection

```javascript
var mySet = new Set();

mySet.add(1); // Set { 1 }
mySet.add(5); // Set { 1, 5 }
mySet.add(5); // Set { 1, 5 }
mySet.add('some text'); // Set { 1, 5, 'some text' }
var o = {a: 1, b: 2};
mySet.add(o);

mySet.add({a: 1, b: 2}); // o is referencing a different object so this is okay

mySet.has(1); // true
mySet.has(3); // false, 3 has not been added to the set
mySet.has(5);              // true
mySet.has(Math.sqrt(25));  // true
mySet.has('Some Text'.toLowerCase()); // true
mySet.has(o); // true

mySet.size; // 5

mySet.delete(5); // removes 5 from the set
mySet.has(5);    // false, 5 has been removed

mySet.size; // 4, we just removed one value
console.log(mySet);// Set {1, "some text", Object {a: 1, b: 2}, Object {a: 1, b: 2}}
```

Map Collection

```javascript
var myMap = new Map();

var keyString = 'a string',
    keyObj = {},
    keyFunc = function() {};

// setting the values
myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, 'value associated with keyObj');
myMap.set(keyFunc, 'value associated with keyFunc');

myMap.size; // 3

// getting the values
myMap.get(keyString);    // "value associated with 'a string'"
myMap.get(keyObj);       // "value associated with keyObj"
myMap.get(keyFunc);      // "value associated with keyFunc"

myMap.get('a string');   // "value associated with 'a string'"
                         // because keyString === 'a string'
myMap.get({});           // undefined, because keyObj !== {}
myMap.get(function() {}) // undefined, because keyFunc !== function () {}
```

In Class Exercise

https://jsfiddle.net/4sgz8dn3/

Acknowledgements

Slides adapted from Dr. Thomas LaToza's SWE 432 course