# Dynamic Pages

## Today

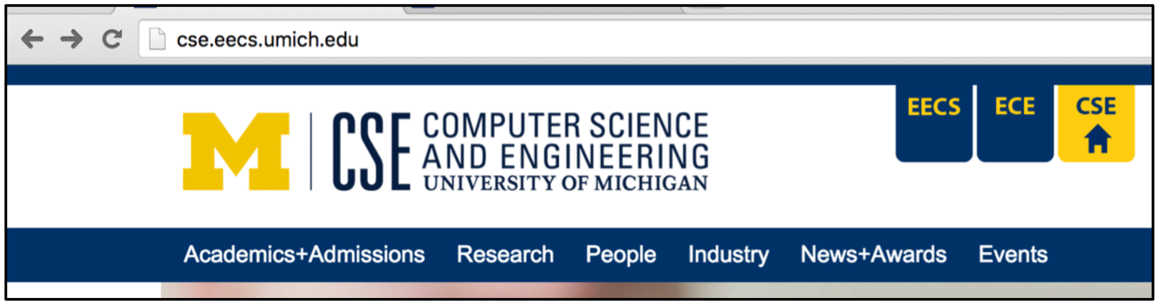
- Markup languages
- Static pages vs. dynamic pages
- Server-side dynamic pages
- Client-side dynamic pages
- Design pattern for dynamic pages

## Markup Language

- How do we improve the presentation of the plain text transmitted over HTTP ...

```
Academics+Admissions Research People Industry
News+Awards Events
```

- ... so that it looks like this?



## Markup Language

- Add tags as “mark up” to text.
- Document still “primarily” text.
- Mark up improves both structure and presentation.
- Zillions of mark up languages.

## HTML

- HyperText Markup Language
- Invented by Tim Berners-Lee in 1990
- Set of tags for rendering page

```
<html>
<body>
The content of the body element is
displayed in your browser.
</body>
</html>
```

## Hypertext

- Text with embedded links to other documents.
- Anchor tag `<a href="link">atext</a>`
- `link` can be any URL
- Prefer relative URL
  - Skip host, port, and start with relative path
  - Path relative to “current directory”
  - Crucial for portability
  - E.g `forms/submit.php?name=jack`

## Escape Strings

- Some characters have special meaning in an application context
  - / or ? in a URL
- What if you want to communicate such a character as an ordinary character?
- Need an “escape string”
- E.g. “\/” and “\?”
- And also “\\”
- &lt; &euro; &amp;
- &nbsp; (non-breaking space)

## XML

- eXtensible Markup Language
- No standard set of tags!!
- Define tags and use them
- Tags form a hierarchy of objects
  - Each open tag has a matching close
  - Must balance, like parentheses
  - Can build a tree of document objects
    - Document Object Model (DOM)

## XHTML

- HTML permits unbalanced tags.
  - <br>
  - <P> automatically closed at next <P>
- HTML cannot be derived from XML
- XHTML is a dialect of HTML that meets XML rules (proper containment)
- Still HTML, so understood by browsers
  - Basically add </br>, </P> etc.
  - (Can also use <br/>
- Can do this because HTML already had a hierarchical DOM.

## HTML5

- Merges all that has happened over years related to HTML:
  - XHTML
  - Browser-specific extensions of HTML
  - Other use cases of broad interest
- Finalized and published by W3C in 2014
- Check your HTML!
- https://validator.w3.org/nu/?doc=URL_GOES_HERE

## Content, Presentation, Layout

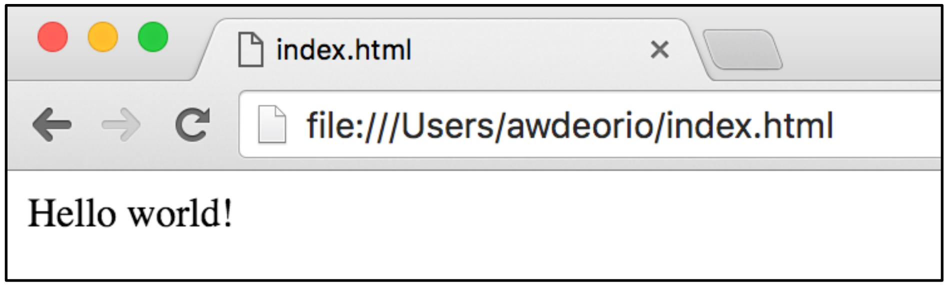
- HTML has tags for all.
  - <H1> is content
  - <B> is presentation
- Good to separate these.
- Content is data; presentation is words, tables, organization; layout is visual.
- Use HTML for content and presentation, add CSS for layout.

# CSS – try this

- HTML for content

```
<!-- index.html -->
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body><p>Hello world!</p></body>
</html>
```

- Load the page in your web browser



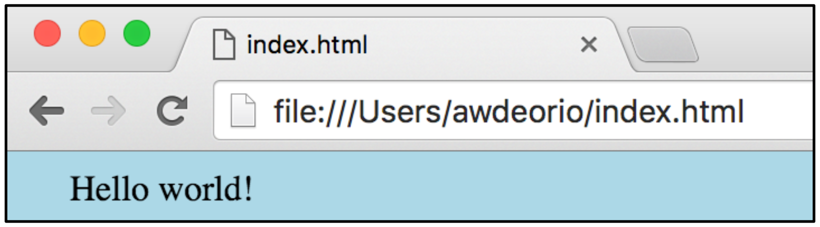
Hello world!

## CSS – try this

- CSS for layout

```
/* style.css */
body {
    background-color: lightblue;
}
p {
    margin-left: 20px;
}
```

- Now reload index.html



Hello world!

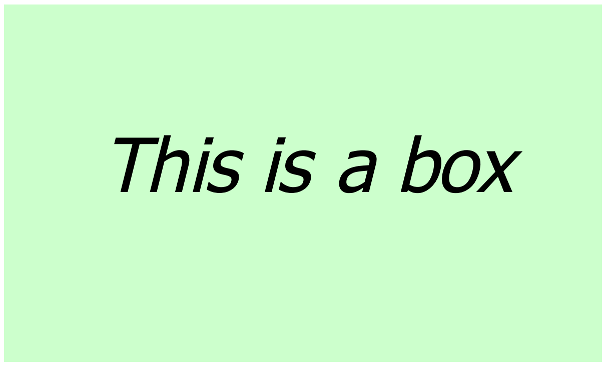
## CSS

- Lots of CSS templates available
- Quickly and easily change the look and feel of a web site
- For example, Bootstrap is popular

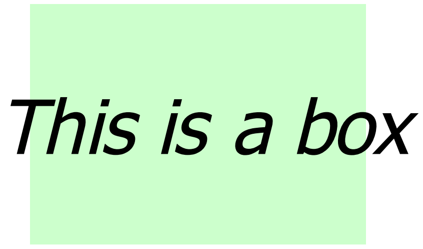
## Scaling and Positioning

- Biggest problem in layout.
- If display screen width changes, do you:
  - Keep your page fixed width any way
  - Adjust page width, and use more lines of text – increasing page length
    - How about embedded figures?
  - Adjust page width and font size so everything scales evenly
- First and third choice easy to do, but may lead to bad display results. Second choice hard to do correctly.
- Even more important with mobile sites

## Scaled Figure



Original Figure



Smaller Figure,
But same font size
(ppt default!!)

# Today

- Markup languages
- **Static pages vs. dynamic pages**
- Server-side dynamic pages
- Client-side dynamic pages
- Design pattern for dynamic pages

## Static page example

- Example: https://en.wikipedia.org/wiki/Computer

- Check out the text
- Now, view source.  You should be able to find all the text in the HTML.

## Static vs. dynamic content

- Static content is the same every time
- Dynamic content changes
- Think of the things that are impossible with simple static pages

## Static vs. dynamic content

- Static content is the same every time
- Dynamic content changes
- Think of the things that are impossible with simple static pages
    - Web search
    - Database lookups
    - Current time
    - # visitors to page
    - Everything

## Static content

- On the server side: HTTP servers are fileservers
- On the client side: browsers are HTML renderers

- Example
  - `python -m SimpleHTTPServer`
  - Copies files

## Server-side dynamic page example

- Example: https://github.com/awdeorio
- Check out the text
- Now, view source.  You can find the text.

- But, another user's github page is different, e.g., https://github.com/mikecafarella
- The server generates these on-the-fly from a database

## Client-side dynamic page example

- Example: https://www.instagram.com/

- Check out the text
- Now, view source.  You won't see the text, but you will see a bunch of JavaScript source code.

## Today

- Markup languages
- Static pages vs. dynamic pages
- **Server-side dynamic pages**
- Client-side dynamic pages
- Design pattern for dynamic pages

## Dynamic Server Content

- In the old days (1997?), almost all requests were just disk loads
- Computing the page dynamically was a **mind-blowing idea**; today it’s assumed
  - Server-Side Includes (SSI) - directives interpreted by the web server itself
  - Common Gateway Interface (CGI) - code executed as a separate process
  - Scripting Languages - PHP, ASP, JSP, Ruby, Python
  - Application Servers - J2EE, .NET, Mongrel

## Templating

- A common way to generate server-side dynamic pages
- Example: Python's `jinja2` library
- Write an HTML file with special keywords
  - e.g., `{% for album in albums %}`
- Run it through a function, along with a data structure of values to fill in
  - e.g., `render_template()`
- Output is expanded HTML

## Template example

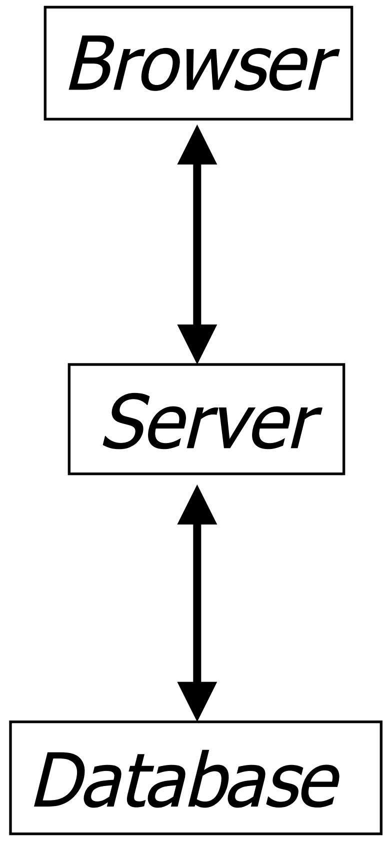
```
<-- albums.html -->
<h2>Albums</h2>
<table>{% for album in albums %}
  <tr><td>{{ album.title }}</td></tr>
{% endfor %}</table>
```

## Template example rendered

```
<-- albums.html -->
<h2>Albums</h2>
<table>
  <tr><td>I love sports</td></tr>
  <tr><td>I love football</td></tr>
  <tr><td>Around the world</td></tr>
  <tr><td>Cool Space Shots</td></tr>
</table>
```

## Dynamic pages

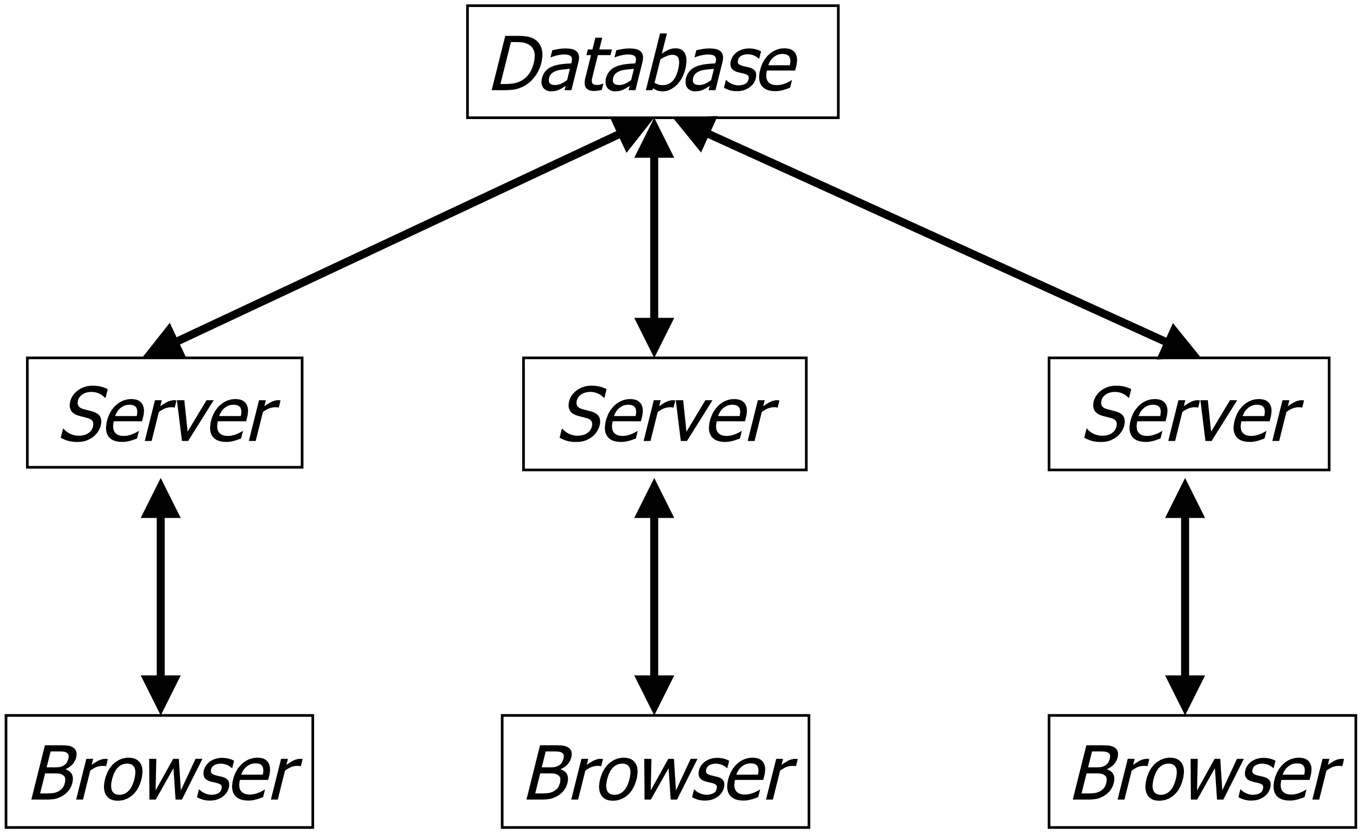
- We need a database to store the values from which we render pages
- 2 (or 3) tier model



## N-tier model

- We’ve separated persistent storage from user interaction
- Other separations?
  - Manageability (what if db goes down?)
  - Security & Mining (logging all user steps)
  - Efficiency (a single dataset in memory?)
- Web apps often break down pieces of code functionality into machines

## In Real Life ...



## Dynamic Server Recap

- Many techniques for mixing content and code
  - PHP, Python, Rails, etc, etc
- Dynamic pages rendered to user at query-time

## Today

- Markup languages
- Static pages vs. dynamic pages
- Server-side dynamic pages
- Client-side dynamic pages
- Design pattern for dynamic pages

## Dynamic Client, too

- All rendered pages used to be static
- Plain HTML not as bad as statically-generated pages, but certainly limiting
- What would not work if all we had was server-side dynamic pages?

## Dynamic Client, too

- All rendered pages used to be static
- Plain HTML not as bad as statically-generated pages, but certainly limiting
  - No in-browser chat
  - No browser-based field validation
  - No grabbable maps
  - No deferred data loading in Gmail
  - No ridiculous ads
    - Grab the monkey
    - Roll over for sound
    - etc

## Dynamic Client Content

- Similarly, all rendered pages used to be completely static
- That all changed with:
  - Adobe Flash
  - JavaScript
  - VBScript
  - Java
  - The <blink> tag
- Actually, all of these except blink

## Dynamic Client Content

- Lots of different options for client-side content, but most people nowadays converging on AJAX (Asynchronous JavaScript and XML)
    - JavaScript for running in the browser
    - XML for data exchange with server, via HTTP (without page load!)
    - After initial page load, similar to client/server program, not traditional Web page loads

## Worst Name in the World

- LiveScript was invented by Netscape, and is great!
- Java was hot, also starting to be built into browsers, so Netscape renamed LiveScript to JavaScript
  - They have nothing to do with each other
  - Except they are both pieces of mid-90s browser tech endorsed by Netscape
  - Also, they have extremely similar names
  - Until 2003 or so, extremely confusing
  - Java in browsers is hardly used anymore

## JavaScript Execution

- Usually, but not always, inside browser
  - Sometimes at HTTP server, instead of PHP!
  - Node.js is very trendy
- JS code runs when triggered by <script>
- Has read/write access to the page’s DOM, or Document Object Model
  - HTML parsed into DOM tree
  - JS programs read or write to the DOM
  - DOM changes reflected on screen

## Very simple JS DOM Example

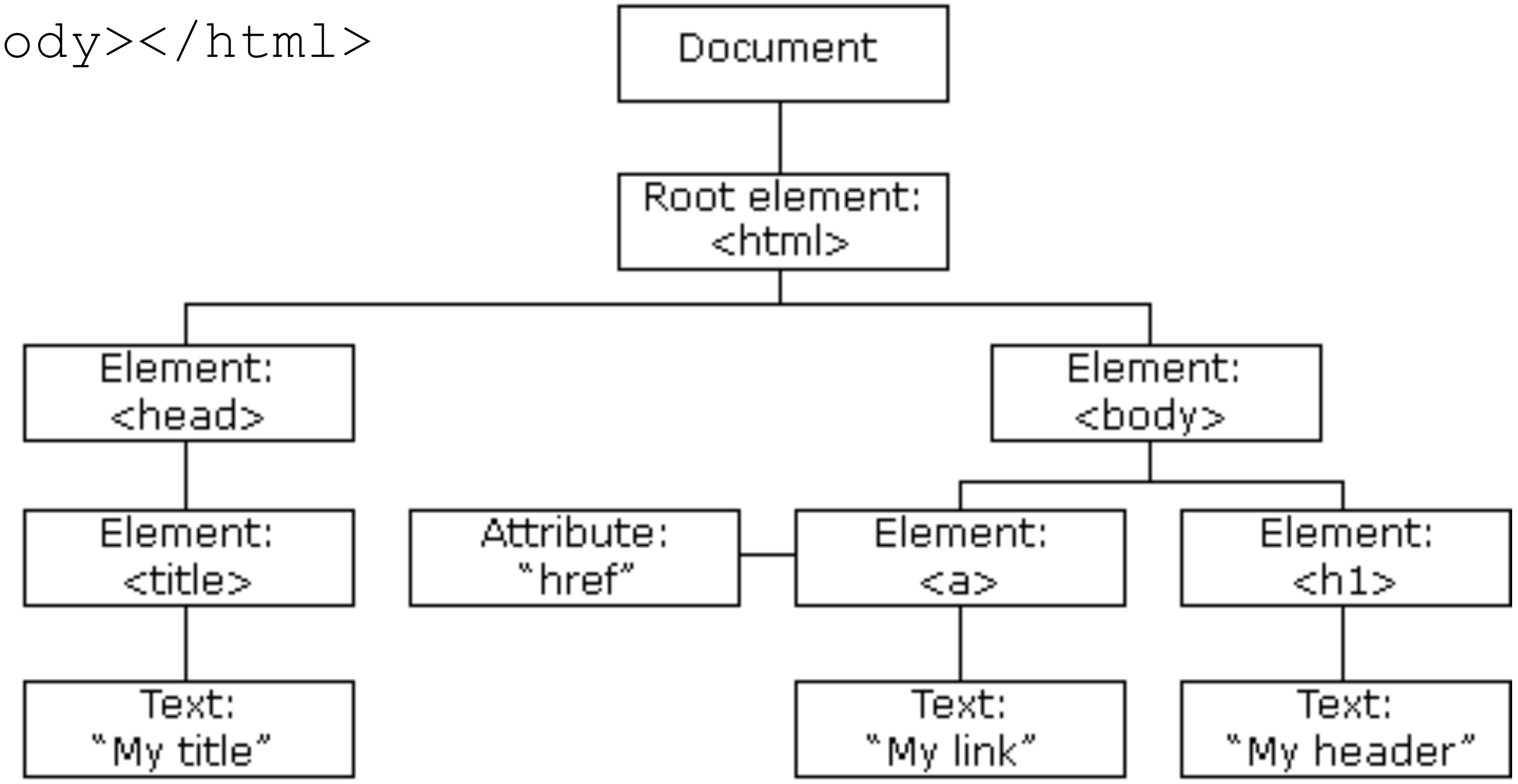
```
<html>
<body>

<script type="text/javascript">
document.write("Hello World!");
</script>

</body>
</html>
```

# JS DOM Example 2

```
<html>
<title>My title</title><body>
<a id="L1" href="">My link</a>
<h1>My header</h1>
</body></html>
```



## JS DOM Example 2

```
<script type="text/javascript">
document.getElementById('L1').href =
"http://www.yahoo.com/";
</script>
```

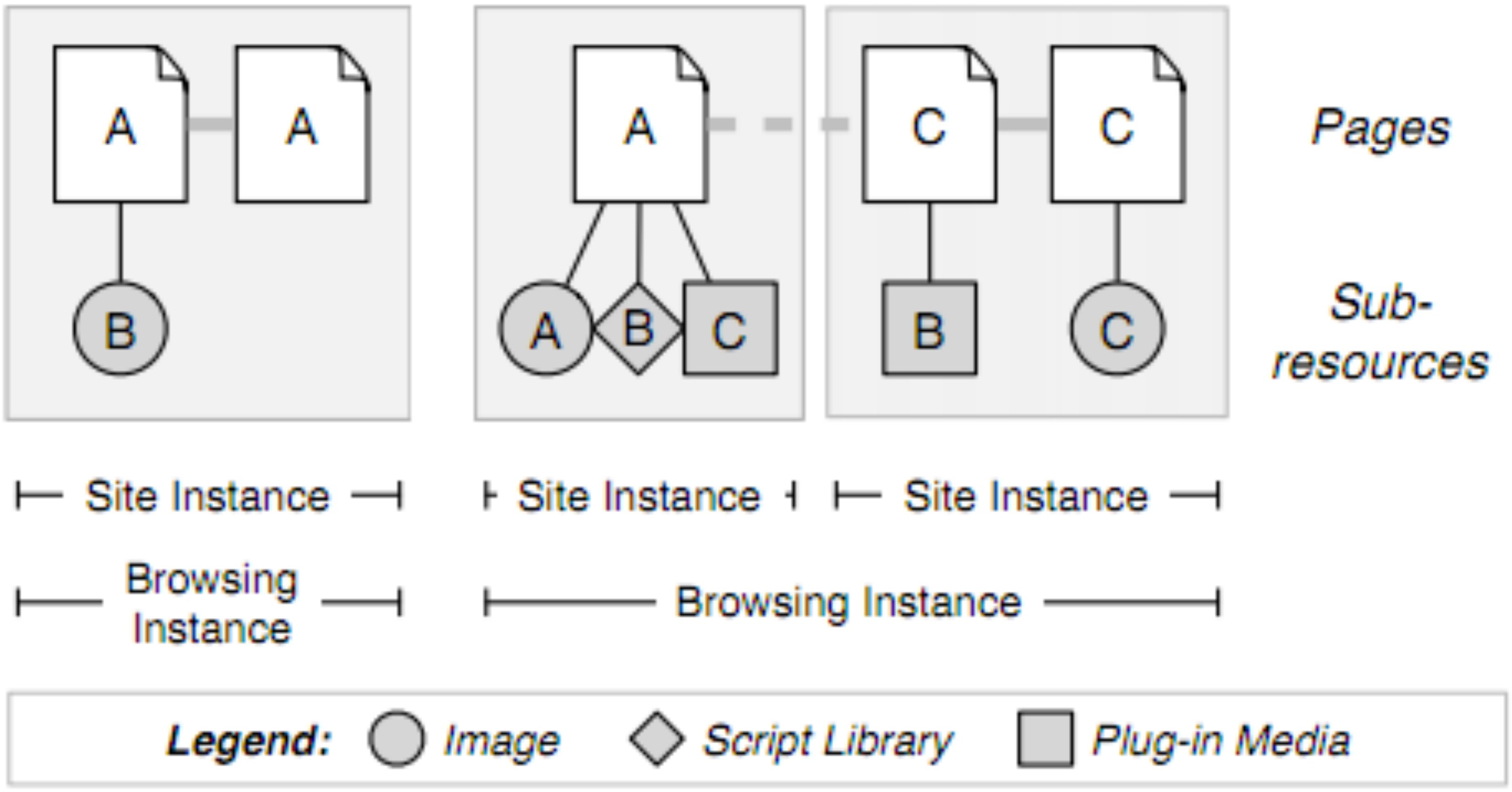
## Browser Security

- JS is potential security nightmare! So in-browser programs are sandboxed
  - No disk or syscalls or infinite loops
  - Same-origin policy → scripts from the same origin can access each others' data + methods
  - → scripts from different origins cannot see each other ("origin" defined by domain, app protocol, and port)

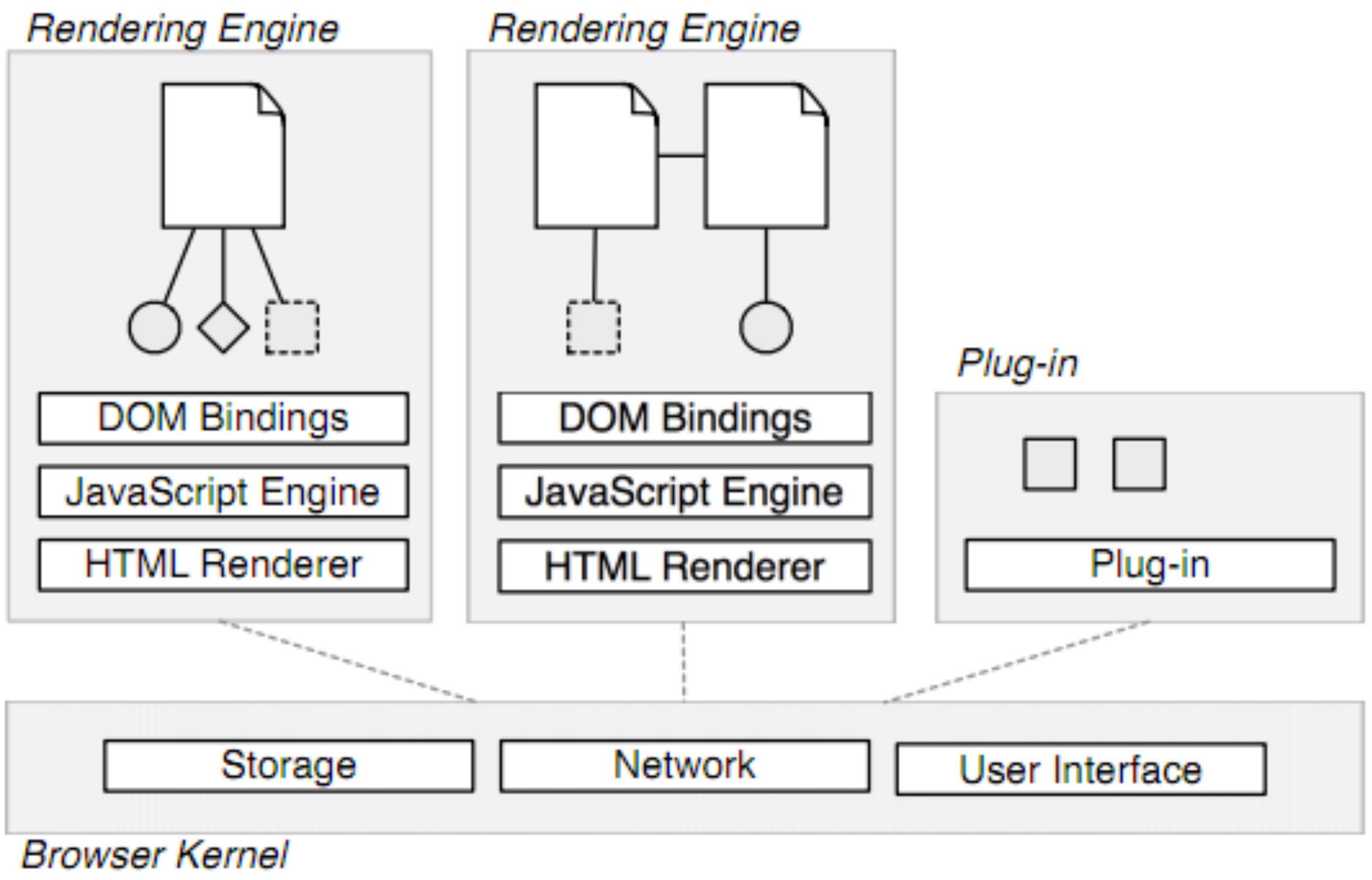
## Attacks

- Cross-site scripting
  - User visits aaa.com
  - Site aaa.com computes nefarious URL, convinces user to click on it
  - URL is something like:
    - http://bbb.com/hello?param1=*!*!@,<h1>Your-account-is-empty-sucker</h1>
  - Site bbb.com takes URL params to compute page
  - Evil content is displayed to user, with inserted-HTML. Could have stolen info (how?)
  - Many have been found against Google
- Sometimes called “tag injection”
  - What is “SQL injection”?

## Browser Internals



## Browser Internals



- Chrome introduced process separation to browser architecture
- One app dies, the others survive

## Today

- Markup languages
- Static pages vs. dynamic pages
- Server-side dynamic pages
- Client-side dynamic pages
- Design pattern for dynamic pages

## Dynamic pages in IRL (in real life) ...

- What if two users are trying to edit my account info?
- What if one browser displays data, then another browser changes the data?
- Where do changes get stored, where do edits take place?

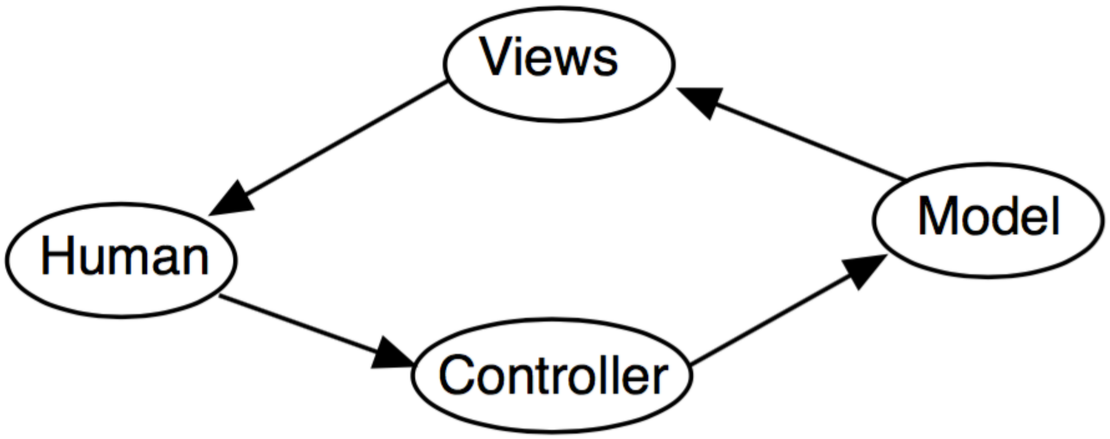
- What programming design pattern can help?

## Model View Controller (MVC)

- Way to sort out who does what
  - Different pieces of code (*abstractions*) for different jobs
- A good approach to the problem of how to separate the functionality "engine" underlying an application from the User Interface code
- Important because there is a tendency for the functionality code to get intertwined with the UI code, making it difficult to change either one - either for revisions/modifications, or to port to a different UI library or platform

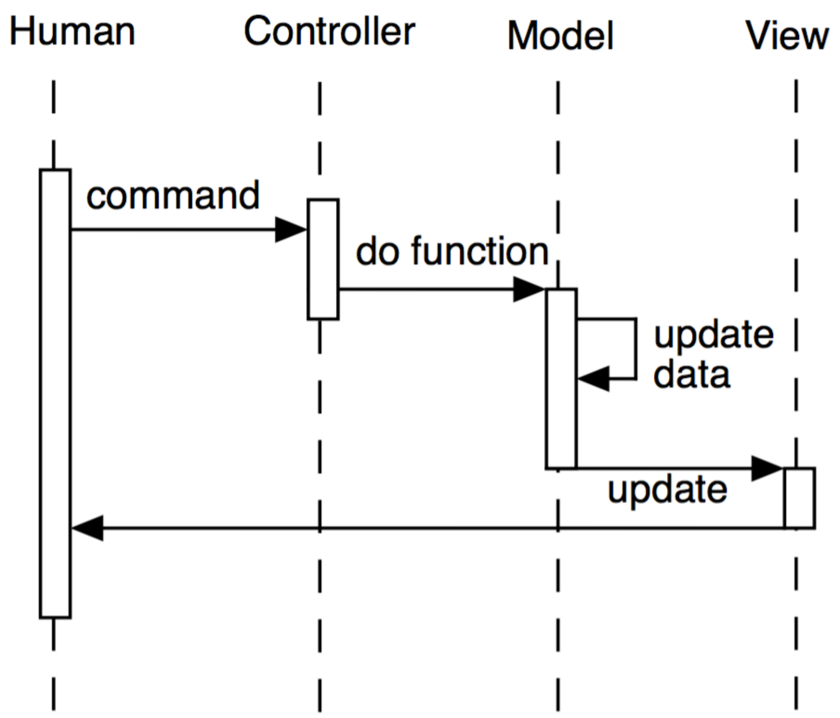
## Model View Controller (MVC)

- Human user operates the Controller to tell the Model what to do. Model tells the Views what has changed. Human looks at the Views to decide what to do next.



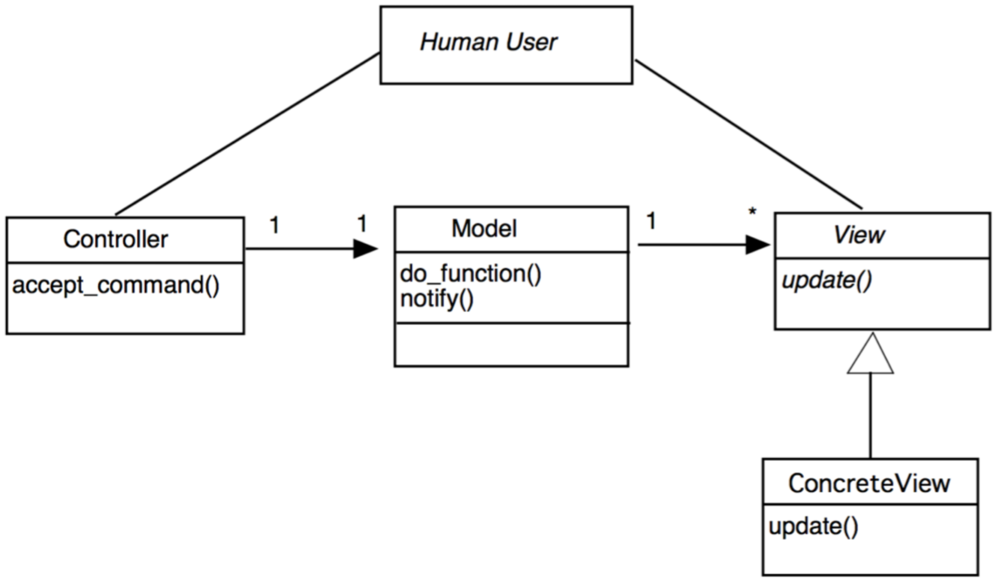
# Model View Controller (MVC)

- Human user operates the Controller to tell the Model what to do. Model tells the Views what has changed. Human looks at the Views to decide what to do next.



# MVC in GUIs

- MVC is often used when writing GUI code
- You might even have classes called Model, View and Controller



## MVC on the Web

- In a server-side web application (like project 1)
- Model: database
- View: HTML/CSS, updated web page
- Controller: HTML forms, GET query parameters and POST requests

## MVC Example

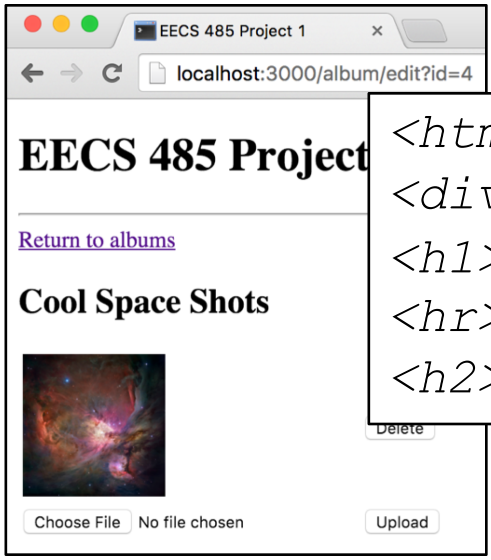
- An example from project 1: uploading an image

1. User views page
2. User clicks "upload" and selects a file
3. New picture appears in album

# MVC Example

1. User views page

View is comprised by HTML, which is generated from jinja2 template code.  Taken together, these are the View.



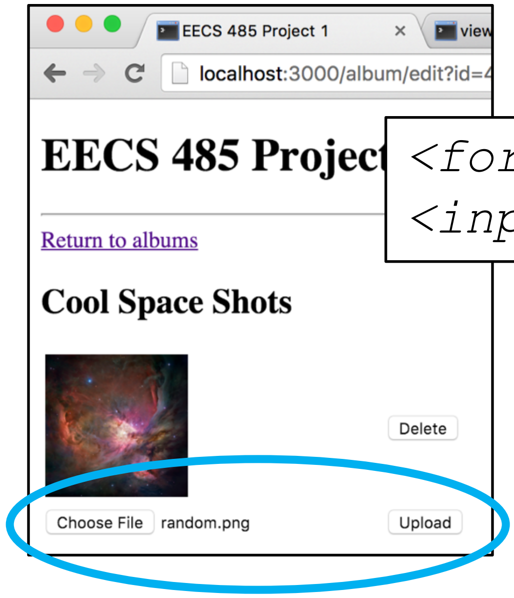
```
<html><body>
<div class=page>
<h1>EECS 485 Project 1</h1>
<hr>
<h2>Cool Sp
```

```
@main.route('/')
def main_route():
  # ...
  return render_template('index.html',
      **options)
```

# MVC Example

2. User clicks "upload" and selects a file.

Controller: HTML form, POST request, Flask handler. Taken together, these form the controller.



EECS 485 Project

Return to albums

Cool Space Shots

```
<form action="" method="post"  ... >
<input type="submit" name="upload" ... >
```

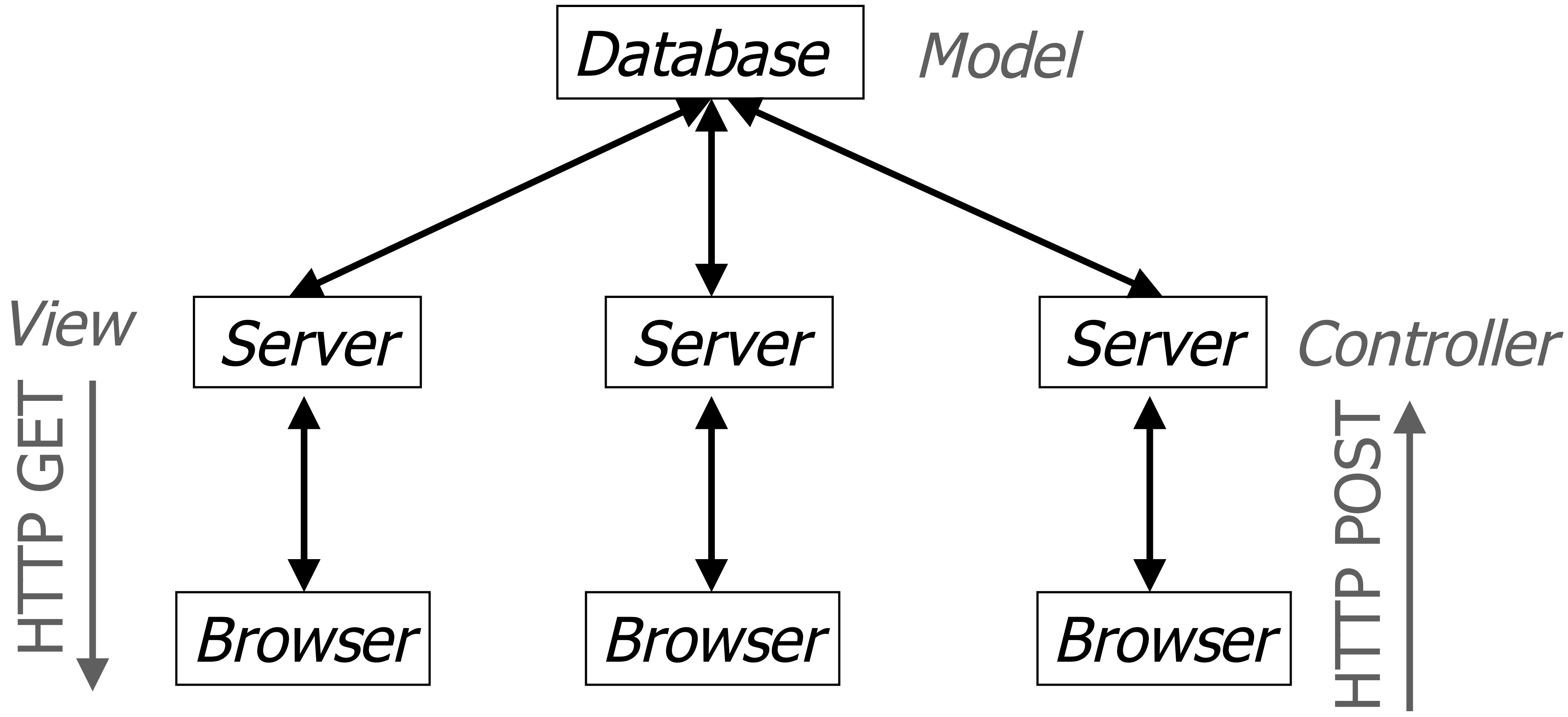
```
@album.route('/album/edit', ...
def album_edit_route(): ('/')
  # ...
  if request.method == 'POST' and
    request.form['op'] == 'add':
  # ...
```

# MVC Example

- Model: database and file system

```
@album.route('/album/edit', ...
def album_edit_route():  ('/')
  # ...
  if request.method == 'POST' and
    request.form['op'] == 'add':
  # ...
  file.save(filename)
  # ...
  cur.execute(
    "INSERT INTO photo (picid, format) " +
    "VALUES ('{}', '{}') ".format(picid, format)
)
```

# MVC in deployment



## More MVC

- What would change (M?, V?, C?) if ...
- Data stored distributed geographically?
- Want to take advantage of VR goggles?
- Browser over new kind of phone?

## More MVC

- What would change (M?, V?, C?) if ...

- Data stored distributed geographically?
  - Model

- Want to take advantage of VR goggles?
  - Controller

- Browser over new kind of phone?
  - View

## Object-Relational Mapping

- Addresses the “impedance mismatch” between relational and object worlds
  - Map object classes to database tables
    - Table=>class
    - Column=>attribute
    - Row=>object instance
- Used by Rails (around for a long time)
  - Ruby is the language
  - Rails is the framework
- In Python, you can use the `sqlalchemy` library