CS1320
Creating Modern Web and Mobile Applications

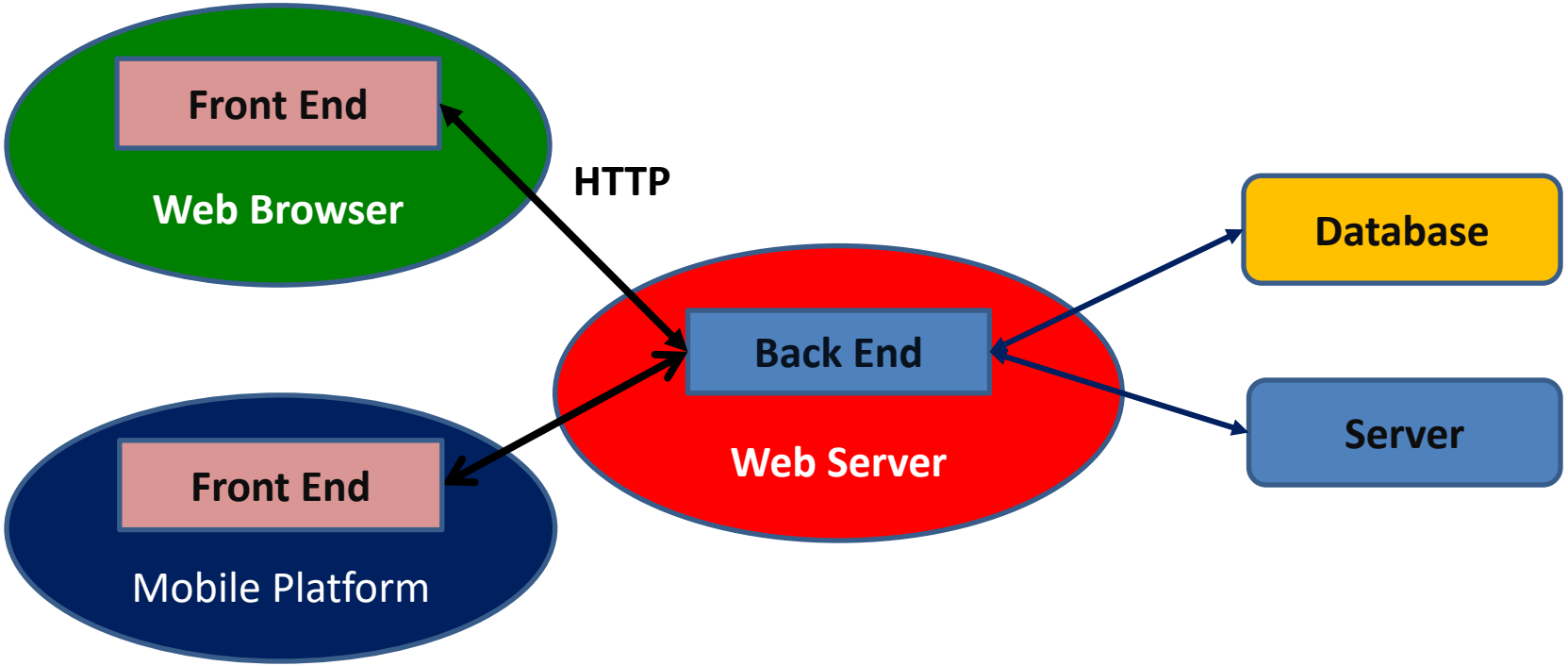
Lecture 2:

The Browser and HTML

The Browser

- What Browser do you use?
  - Why?
  - Is one browser better than another?
  - What should you use in this course?
- What does it do for you?
  - Magically makes pages appear
  - Allows interaction
  - Supports the user experience

Web Applications



Browsers as Application Front Ends

- Do they make writing quality applications easier or harder?

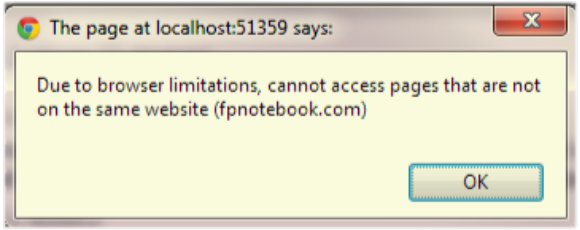
Using the Browser is Helpful

- Makes it easy to create sophisticated interfaces
  - Including images, videos, dynamics
  - Color, typography, accessibility, ..
  - Adapts to different size windows
  - Works on different platforms
- Powerful **declarative** syntax for user interfaces
- Easy prototyping of user interfaces
- Much less code to write

Using the Browser is Restrictive

- Using a browser as a front end is **limiting**
  - Limits the **user** interface
  - Limits the **user** experience
  - Limits application control of the interface
  - Highly interactive applications are more difficult (often more code, slower)
- Other limitations include
  - Limited communications capabilities
  - Limited access to the user machine
  - Pull, not push communications (separated interface)
  - Limited display capabilities
- **You need to understand the limitations**

Front End Code Restrictions



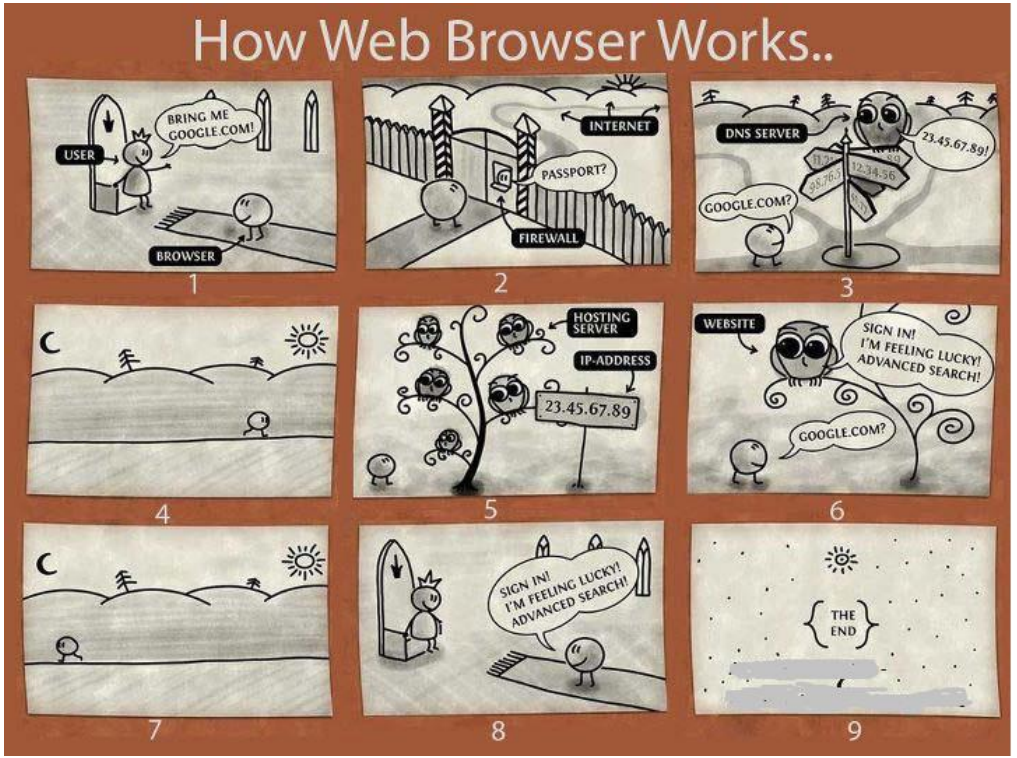
- What can the front end code talk to and see
  - Any program/file on the user's machine or network
    - This is a security/privacy problem
      - » Requires explicit user approval; Discouraged by today's browsers
    - Best not to assume this
  - Restricted local storage (cookies, html5 storage, if allowed by user)
  - The web server
    - Actually any socket on the machine serving the pages
    - Firewalls might limit access to specific ports
    - Generally only the web server and its components
- Talking to the server uses a limited set of protocols
  - HTTP or HTTPS with URLs, header fields, and possibly contents
  - Mobile applications typically use the same protocols

Uniform Resource Locators (URL)

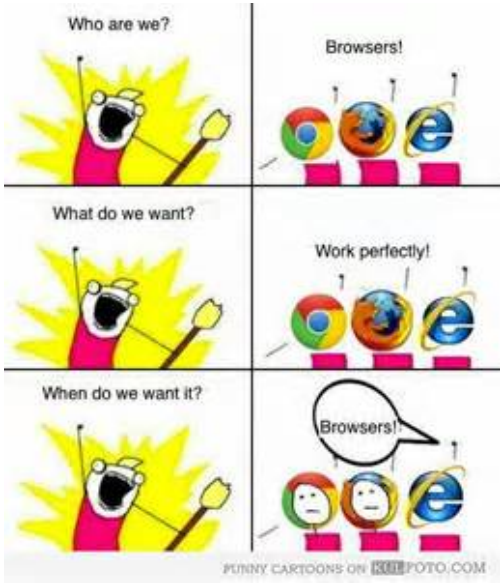
- HTTP: //www.cs.brown.edu:80/people/spr
  - #reference
  - ?name=value&name1=value1 …
- Examples
  - Wikipedia (http://en.wikipedia.org/wiki/Uniform_Resource_Locator)
  - Amazon (https://www.amazon.com/dp/B07456BG8N/ref=ods_gw_ha_rr_p_3pack?pf_rd_p=ece83bcf-c3b4-4c99-b291-7adfd1d50988&pf_rd_r=BN6NP9MBBPVFRXX1WNHX)

What the Browser Does

- You are a browser
  - Someone gives you https://www.cs.brown.edu
  - What do you do?



What the Browser Does



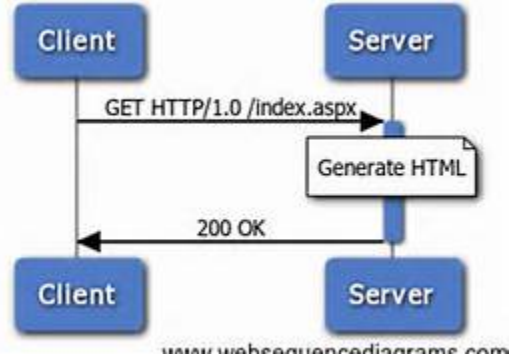
- Given a URL from the user or a program
  - Finds the proper server (based on host)
  - Opens a socket to port 80 (or other specified port)
- Sends a request on that socket (based on protocol)
  - Server then finds the corresponding data
    - Generally the file referred to; Might be dynamically computed
  - Server sends back the result
- Browser reads the response
  - Builds an internal data structure from the response (**DOM**)
  - Displays the corresponding data structure (magic)
  - **Replaces** the current page (or frame) with the new one
  - Executes JavaScript code in the response based on events

Simple HTTP Request

GET /people/spr HTTP/1.1<crlf>

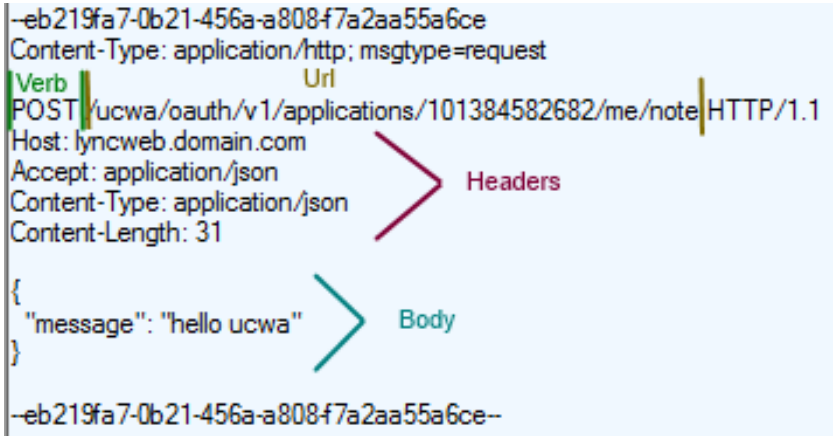
Host: www.cs.brown.edu:80<crlf>

<crlf>



HTTP Protocol: Requests

- Basic *Verbs* are GET and POST
  - GET: effectively header only
  - POST: provides content
  - GET <suburl> HTTP/1.1
- Header fields
  - name: value
  - Describe
    - Who is sending the request
    - Type of data passed and expected (e.g. text/html, text/xml)
    - Length of data; Cookies; Caching information; location; source
- Content
  - Blank line (CRLF) and then actual data



Simple HTTP Response

HTTP/1.1 200 OK<crlf>

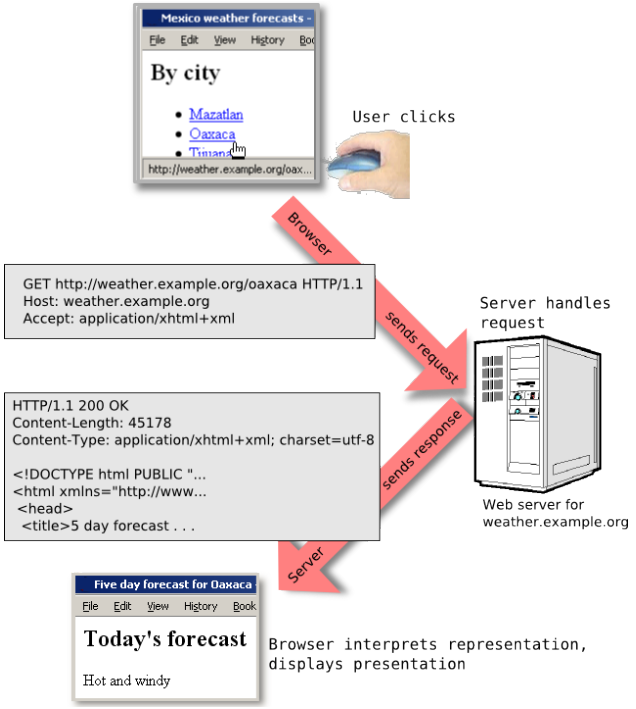
Date: Fri, 27 Jan 2012 10:25:23 EDT<crlf>

Content-Type: text/html<crlf>

Content-Length: 234<crlf>

<crlf>

<html><head> ....

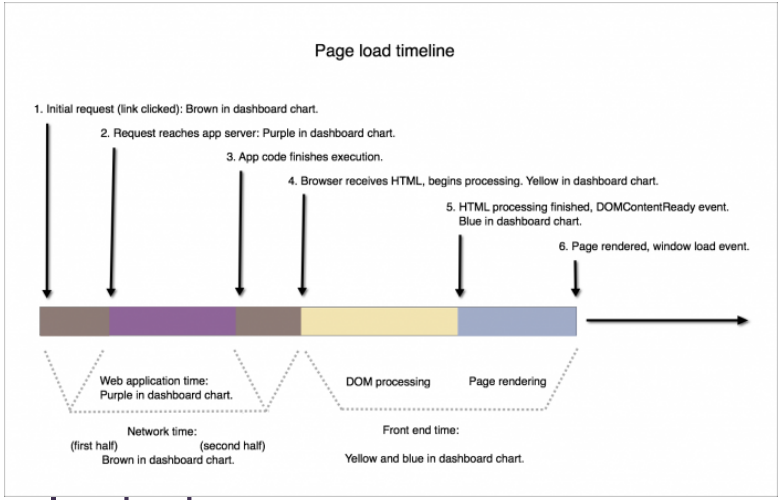


HTTP Protocol: Responses

- HTTP/1.1 <status> <description>
  - 1xx: OK / continue
  - 2xx: Success in various ways
  - 3xx: Redirection
  - 4xx: Client error
  - 5xx: Server error
- Header fields
  - Content-type, Content-length
  - Date and other optional information

Web Pages are not Simple

- Typically include more than one file
  - Can include embedded web pages (ads)
  - Can include multimedia (videos, sounds, …)
  - Can include code files (JavaScript)
  - Can include style files (CSS)
  - Can include raw data (xml, json)
- The browser queues these up and eventually downloads them
  - In parallel, but not too many at once
  - Incorporated into the DOM at the right place



HTTP is Stateless

- Each request is **independent** of other requests
  - From the same user or other users
  - From the same web page or other pages
  - Each request can be treated the same at the server
- Why?
  - Fits browser model of multiple windows, back/forward, …
  - Don't have to worry about state, errors, crashes, etc.
  - Makes the server much simpler
- What's wrong with this?
  - It makes web applications (front and back ends) much more difficult
  - We'll get back to this later in the course

HTML

- Need a way of describing what to display
  - Work with all browsers (browser-independent)
  - Window size and shape independent
  - User-creatable
- History
  - SGML: type-setting markup language (EBT)
    - Basis for HTML and XML
  - Used for manuals off-line and on-line
- HTML = { HTML4 , XHTML , **HTML5** }

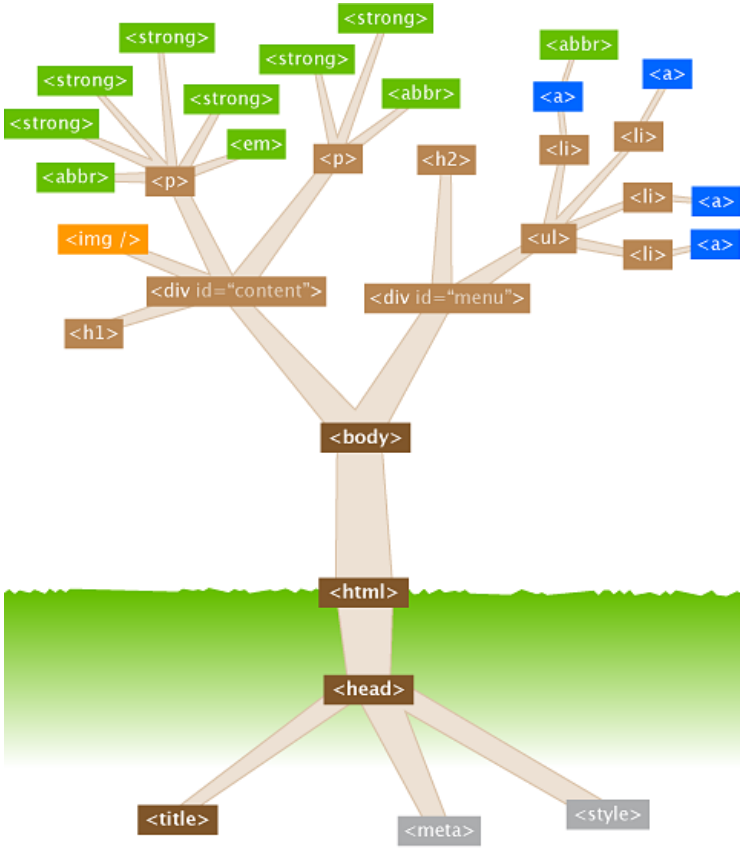
Felix always added HTML tags to his articles ... though he really questioned their effectiveness.

HTML Structure

- HTML is a tree structure
  - Internal nodes represent structure
  - Leaf nodes represent content
- Specified textually as a tree

```
<node>
    <subnode field='value'>
        Text in a  leaf node
        <leafnode />
    </subnode>
</node>
```

- Maintained internally as a tree (DOM)
- Nodes have names, attributes
- Text is a special type of leaf node



Simple HTML Example

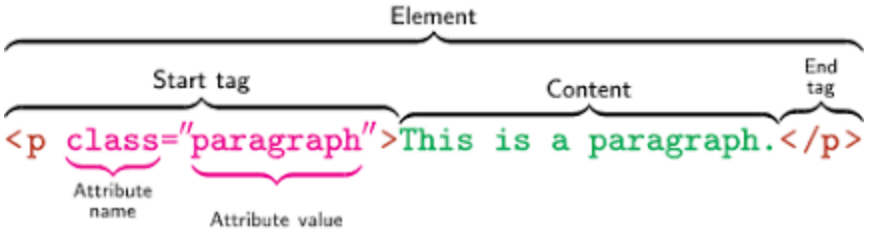
```
<!DOCTYPE html>
<HTML>
    <HEAD>
        <META charset="utf-8" />
        <TITLE>Page title. Shown in tabs.</TITLE>
    </HEAD>
    <BODY>
        <DIV>
            <H1>Simple Page!</H1>
            <P>
                This is a totally <EM>bare-bones</EM> page.
            </P>
        </DIV>
    </BODY>
</HTML>
```

HTML Top-Level Components

- Header: basic information about the page
  - Title, character set, redirect information, …
  - Styles (CSS): information on how to display
    - Can be in separate files
    - Prelab (tutorial) + Lab Wednesday
    - Assignment 1
  - Scripts (JavaScript)
    - Dynamic interactivity
    - Can be in separate files
    - Next Friday + Monday + Wednesday + Lab following week
    - Assignment 2
- Body: the data to display
  - Description of what should be presented on the page
  - Semi-structured text
  - Prelab (tutorial) + Lab Wednesday

HTML Elements

- <SPAN ID='id001' CLASS='emph type1'> text goes here … </SPAN>
  - Element name
  - Attribute – value pairs
  - ID: should be unique, used for identifying the element
  - CLASS: standard names + user names
    - Can have multiple entries
    - Can occur on multiple elements
    - Use for identifying elements, formatting
- <IMG src="image.png" alt="description" />
  - Simple tags need no end tag
- <P> … <P> …
  - HTML does automatic error correction, inserting end tags at times
  - Best to do the end tags yourself
- simple text is a special type of leaf element



CSS: Style versus Content

- Separate the style from the content
- Basic Syntax

```
Selectors {
        Property : value;
        Property : value;
}
```

- Including inline
  - <STYLE> …. </STYLE>
  - <DIV STYLE='property: value' …>
- Including via links
  - `<LINK rel='stylesheet' href='path.css' type='text/css' />`
- Tricky Parts: selectors, property names, property values, syntax errors
- Code Bubbles Before and After
  - Http://www.cs.brown.edu/people/spr/codebubbles
  - Http://www.cs.brown.edu/people/spr/codebubbles/indexold.html

CSS Selectors

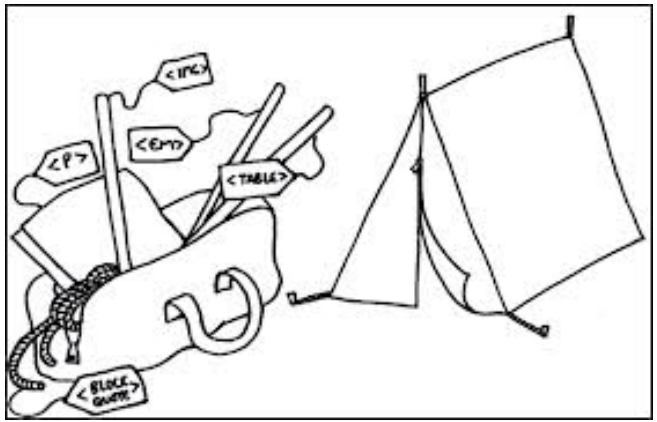
- h3 { … }
  - Apply to all h3 tags
- .emph { … }
  - Apply to all elements with class emph
- #idtag { }
  - Applies to the element with id = idtag
- Combinations, nestings, etc. are possible
- These are used for identifying elements or sets of elements
  - For styling
  - For dynamic web pages
  - For web scraping

CSS Selectors

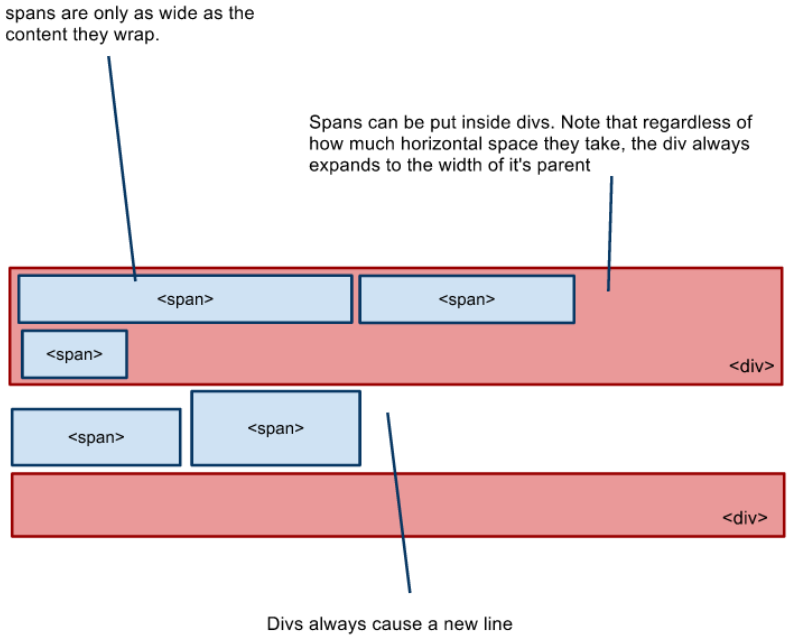
Selector	Role
p{ }	Tag selector, **all p tags**
#para{ }	Id para (***unique***)
.para1{ }	Class para1 (***multiple***)
p.para{ }	P tag with class para
P .para{}	P with child having class para
div p{}	p tag having parent div.
*{}	All tags{ **Universal Selector**}
h1, h3, h5{}	Only h1, h3 and h5 (***grouping***)
.para a{ }	A with parent para class
body{}	Parent of all tags

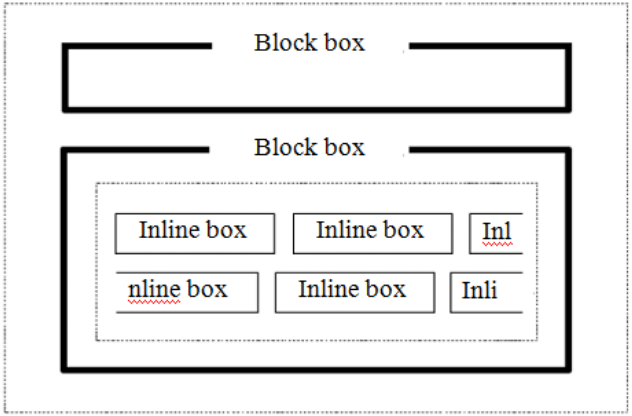
Basic HTML Body Components

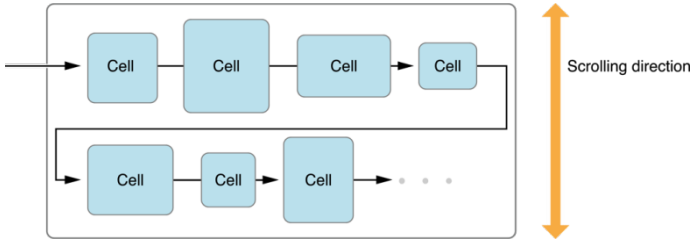
- Simple text
- Descriptions of how to display text
  - <EM>text</EM>, <SPAN CLASS='emph'>text</SPAN>
  - Managed by CSS
- Page layout and organization
  - Headers, paragraphs, blocks
  - Lists and Tables
- Interactive regions
  - Forms: text fields, buttons, …
  - Canvas, SVG regions
- Divided between inline and block elements
  - Characterized by role in page layout



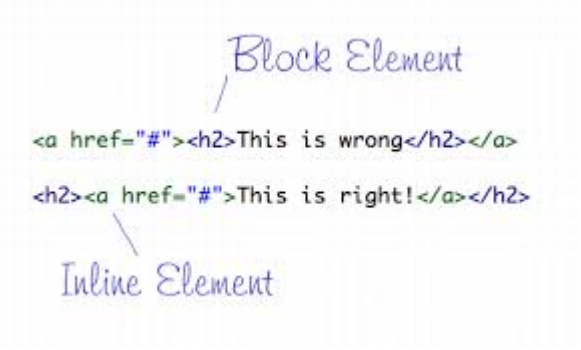
HTML Flow and Layout





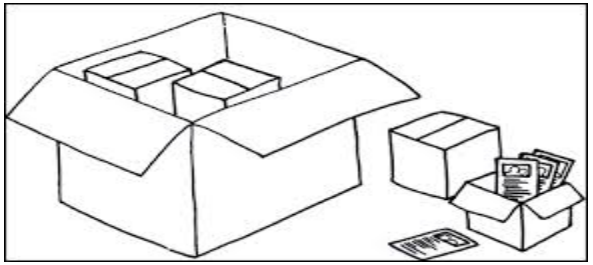


HTML Inline Elements

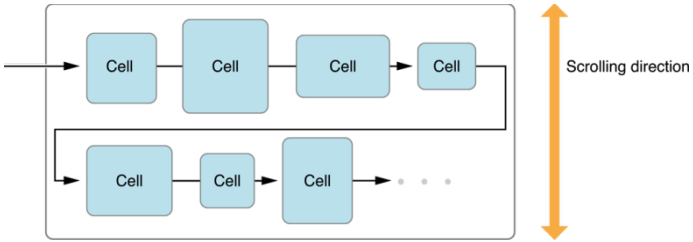


Tag:	Usage:
• A	• <A href="http://google.com">Google</A>
• STRONG	• <STRONG>Usually Bold</STRONG>
• EM	• <EM>Usually italicized</EM>
• SPAN	• Text which is <SPAN>logically divisible</SPAN>
• IMG	• <IMG src="image.png" alt="description" />
• text	• Arbitrary text to display

HTML Block Elements

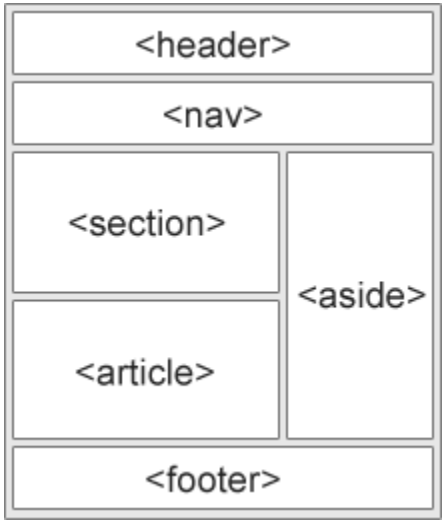


- H1, H2, H3, H4, H5, H6 (header)
- P (paragraph)
- UL, OL, LI (unordered list, ordered list, list item)
- DIV (logical division)
  - HTML5 div specialization (such as 'header', 'footer', 'section', and 'article')
- IMG (image)
- TABLE, TR, TH, TD (tables)



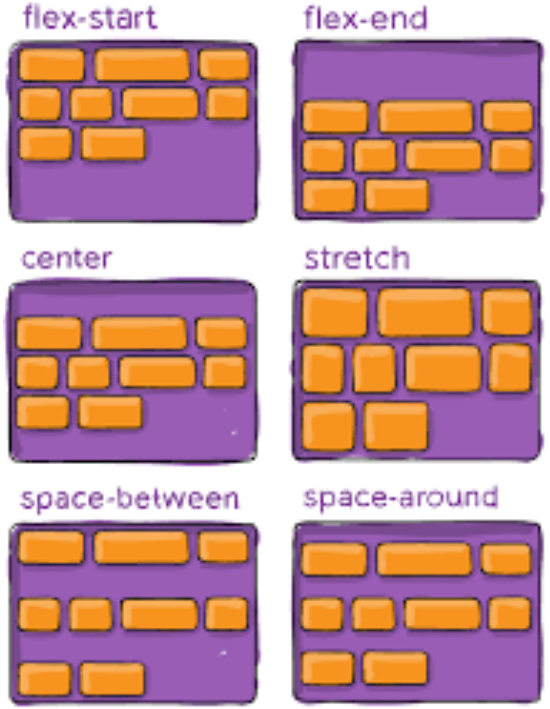
Organizing Page Layout

- Basic flow and layout is somewhat primitive
  - Want more control; want a better user experience
- Tables
- CSS (absolute and relative positions)
- FlexBox (modern CSS) – constraint based positioning
  - display : flex
- Handling different browser sizes (responsiveness)
  - Mobile browsers, large screen browsers, …
  - Can be done with CSS
  - BootStrap simplifies this



Flex Boxes : Controlling Flow and Layout

- Can control the flow in the parent
  - Direction, justification, alignment, wrap
  - Done with CSS properties and attributes
- Can control individual children
  - Order
  - What expands and shrinks and what doesn't
  - Default size
- All done in terms of CSS
  - Can have different styles for different size screens
- Covered in pre-lab and you will use in lab and homeworks



BootStrap: Responsive Pages

- BootStrap provides an organized approach to responsive layout
  - Instead of you creating appropriate CSS for each relevant layout size
- Changing size of display changes desired layout
  - Wide – place things horizontally
  - Narrow – place things vertically
- BootStrap makes these decisions dynamically based on size information
- Basic concept – a container with components
  - Components have relative sizes (multiples of 1/12)
  - Components can be nested
  - Bootstrap lays out elements in each component responsively
- Covered in pre-lab and you will use in lab and homeworks

Next Time

- Monday: Universal Accessibility
- Student project proposals are due
- Homework:
  - Assignment 0 due by next Friday (collaboration form)
  - Prelab 1 due Wednesday
- Class Prep for Monday

  Take a web site of your choosing. Using the accessibility options on your computer, try to access it via a screen reader, a high-contrast display, or with 4X or larger magnification or other accessibility feature. Come prepared to discuss your experience.

Final Projects

- Why use external sponsors?
- Student projects
  - Requirements and specifications
  - Importance of having a well-defined project
  - You will need a suite of test users (other than yourselves)

HTML Examples

- Course home page
- Displaying text
- Using CSS and styles

HTTP is Stateless

Does this mean:

A. It doesn't matter what country the HTTP request came from?
B. HTTP requests can be linked to other requests using session ids?
C. HTTP requests can come in any order?
D. Each HTTP request is independent of each other?
E. HTTP requests can't contain any information about the user or the browser?