

Week 3A – Web and HTML

(self study)

Dr Kiki Adhinugraha

The World Wide Web

Many slides taken from the Web Application Architectures course on Coursera by Greg Heileman

Definition (Web Application)

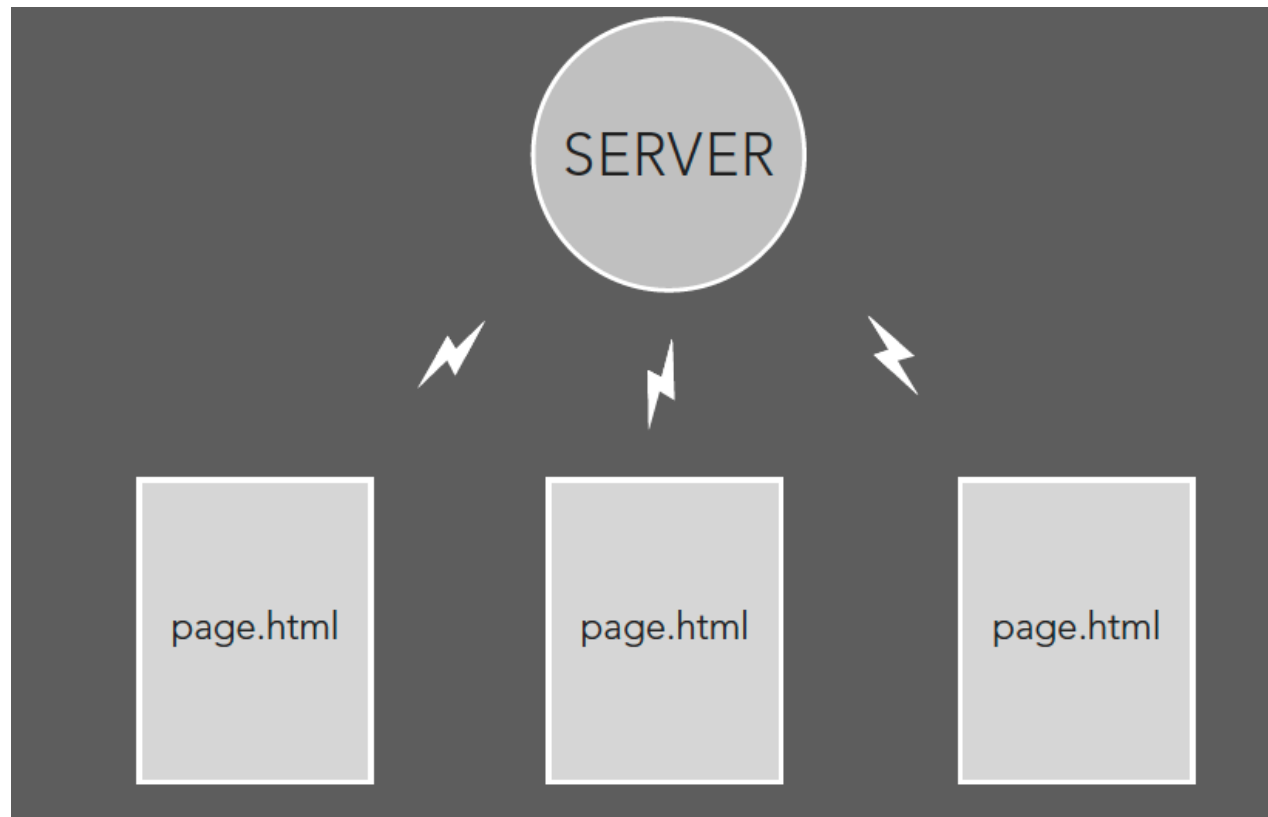
- A web application is accessed by users over a network, uses a browser as the client, and consists of a collection of client and server side scripts, HTML pages and other resources that may be spread across multiple servers. The application itself is accessed by users via a specific path within a web server, e.g. www.amazon.com
- Examples:
 - Webmail, online retail stores, online banks, online auctions, wikis, blogs, document storage, etc.

Web (World Wide Web)

- A system of interlinked documents (web pages) accessed via the Internet using HTTP.
- Web pages contain hypermedia: text, graphics, video and other multimedia, along with hyperlinks to other web pages.
- Hyperlinks give the Web its structure
- The structure of the Web is what makes it useful and gives it value

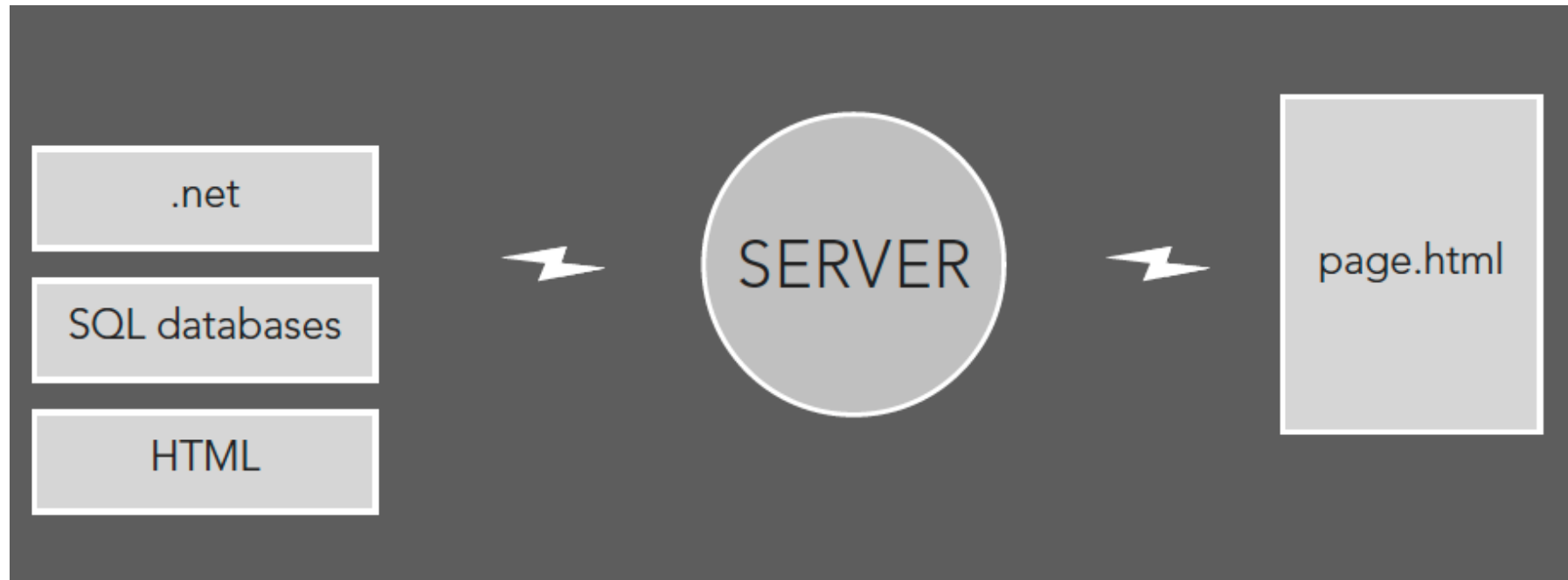
Static Web Content

- A static website is a group of self-contained individual pages (or page), sent to the browser from the server one-page-at-a-time



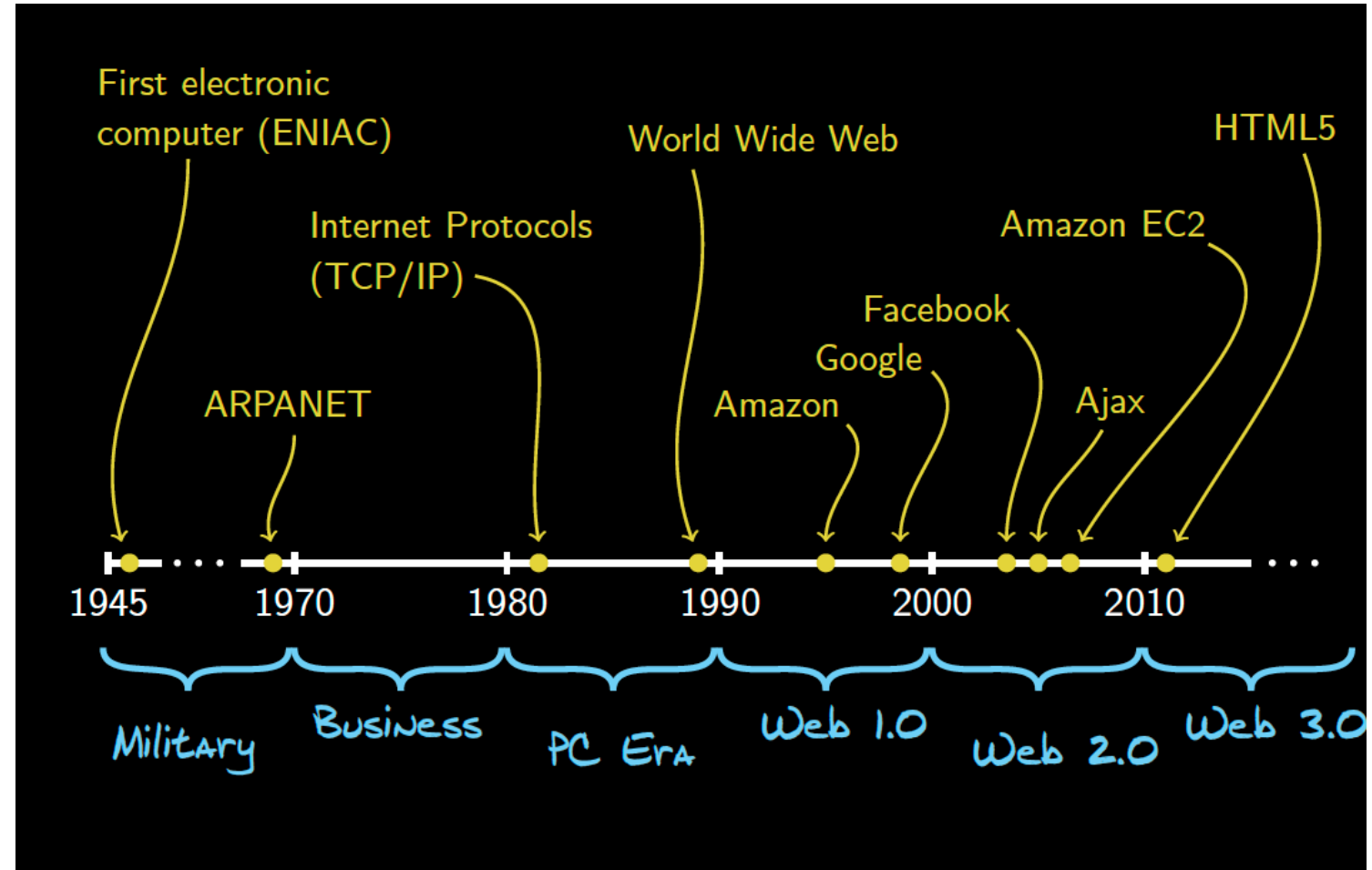
Dynamic Web Content

- Dynamic web content is **built when it is requested**, by the user directly, or programmatically while a user is on a page (e.g., Facebook updates).
- Most websites contain both static and dynamic elements.



The History of the Web

- ARPANET (Advanced Research Agency Network)
 - First introduce packet switching network
 - First to introduce TCP/IP



Web 1.0 (Static Web Sites)

- Creation of static web sites, the first web business models
- Content served from the server's filesystem instead of a RDBMS
- Pages built using Server Side Includes or CGI script instead of web application in a dynamic programming language such as PHP or Ruby
 - The Common Gateway Interface (CGI) allows you to program in any language. The CGI script is a file which gets executed when the client browser sends a request to the web server to execute your CGI file.
 - On the other hand most PHP scripts run in a web server process via mod_php which is not CGI
 - CGI is slow since it needs to be started per request. In contrast mod_php is executed in a long-running process which runs continuously.
- Few content producers, most where consumers of content.

Web 1.0 Applications

- As applications became richer, server-side scripts became more complicated and web 1.0 applications became very difficult to maintain
- The “Browser Wars” led to more functionality on the client side, along with compatibility issues.
- Developers began creating applications that were more interactive
 - Requires saving state.
- Technologies that improved performance emerged
 - E.g. client-side scripts, faster web servers, web caching, Content Delivery Networks (CDN)

Web 2.0 (Interactivity)

- Instead of being passive like Web 1.0, Web 2.0 allows users to interact with the web page
 - Social networking sites
 - Blogs
 - Wikis
 - Video sharing
 - Online commerce
 - Etc.
- AJAX
 - Allows web page to be updated without reloading the page
 - Request data from a server – after page has been loaded
 - Receive data from a server – after the page has loaded
 - Send data to a server – in the background

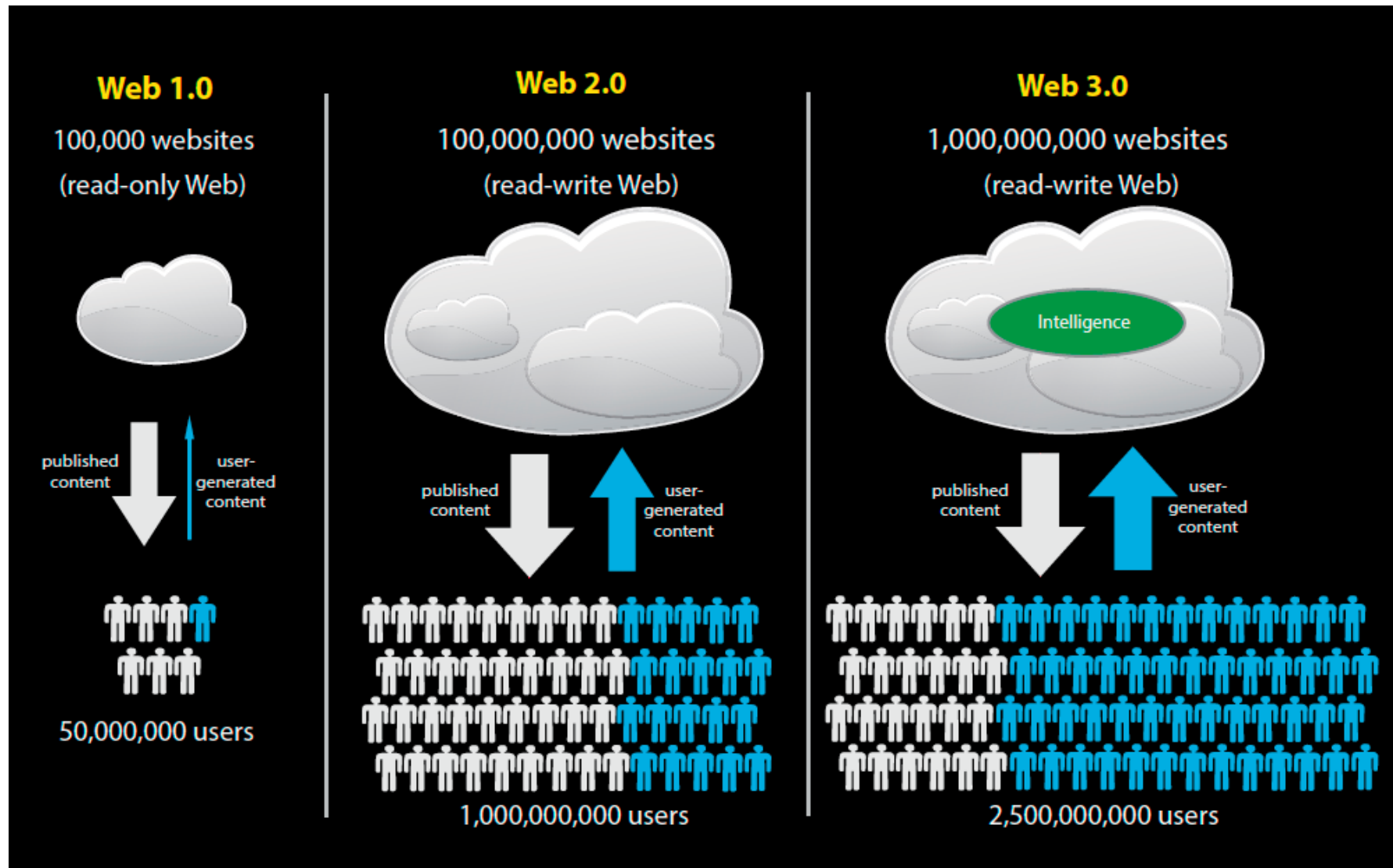
Web 3.0 (The Intelligent Web)

- Machine-facilitated understanding of information
- Natural language processing
- Machine learning and reasoning
- Recommender systems

Enablers of Web 2.0 and 3.0

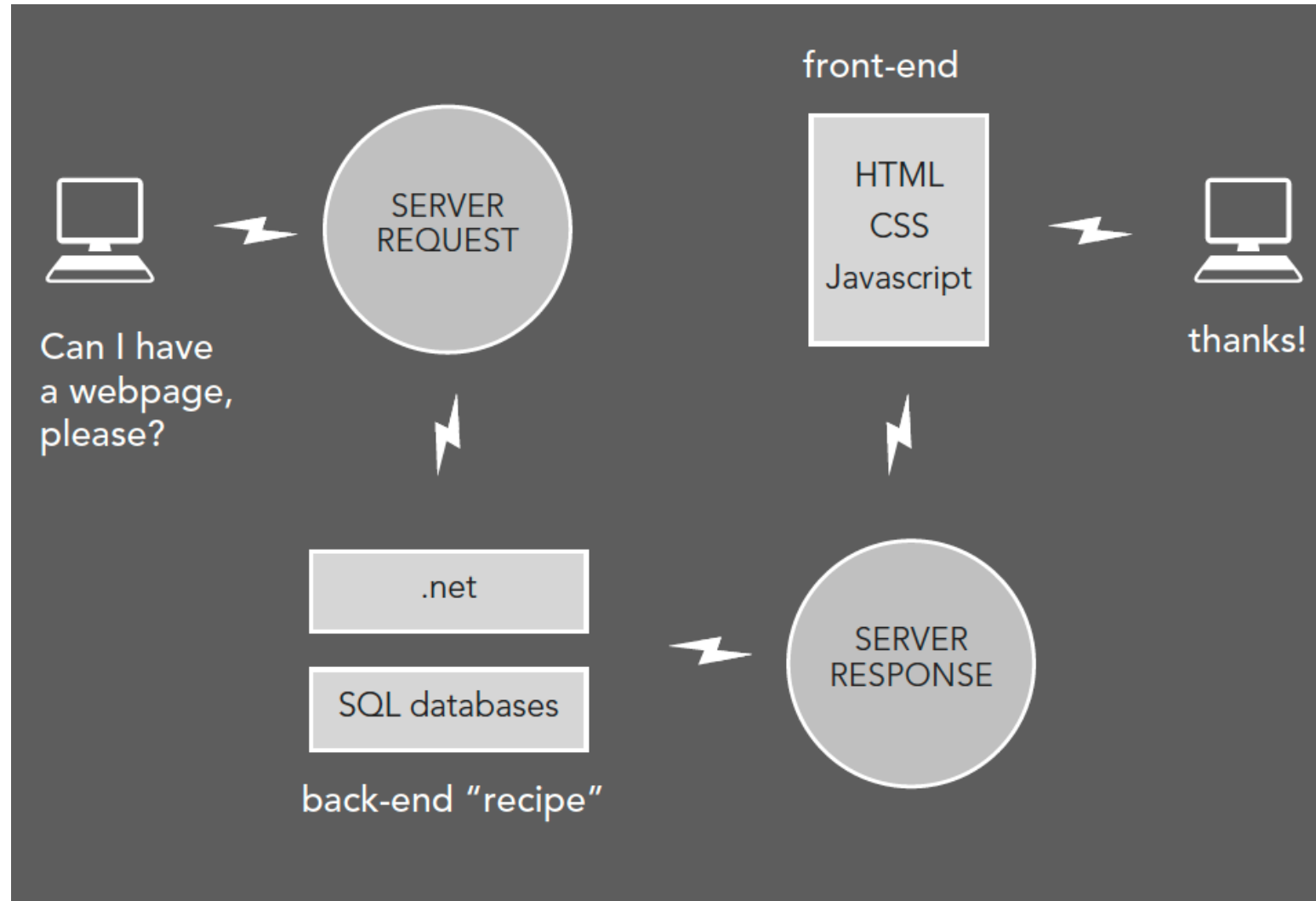
- JavaScript, XML, JSON (Ajax)
- **Web services** interoperability
- Cloud computing
 - Infrastructure, platform and software-as-a-service capabilities.
- Mobile platforms and apps leading to ubiquitous computing

Scale of Web 1.0, 2.0 and 3.0



Web Application Architectures

Basic Architecture of Web Application

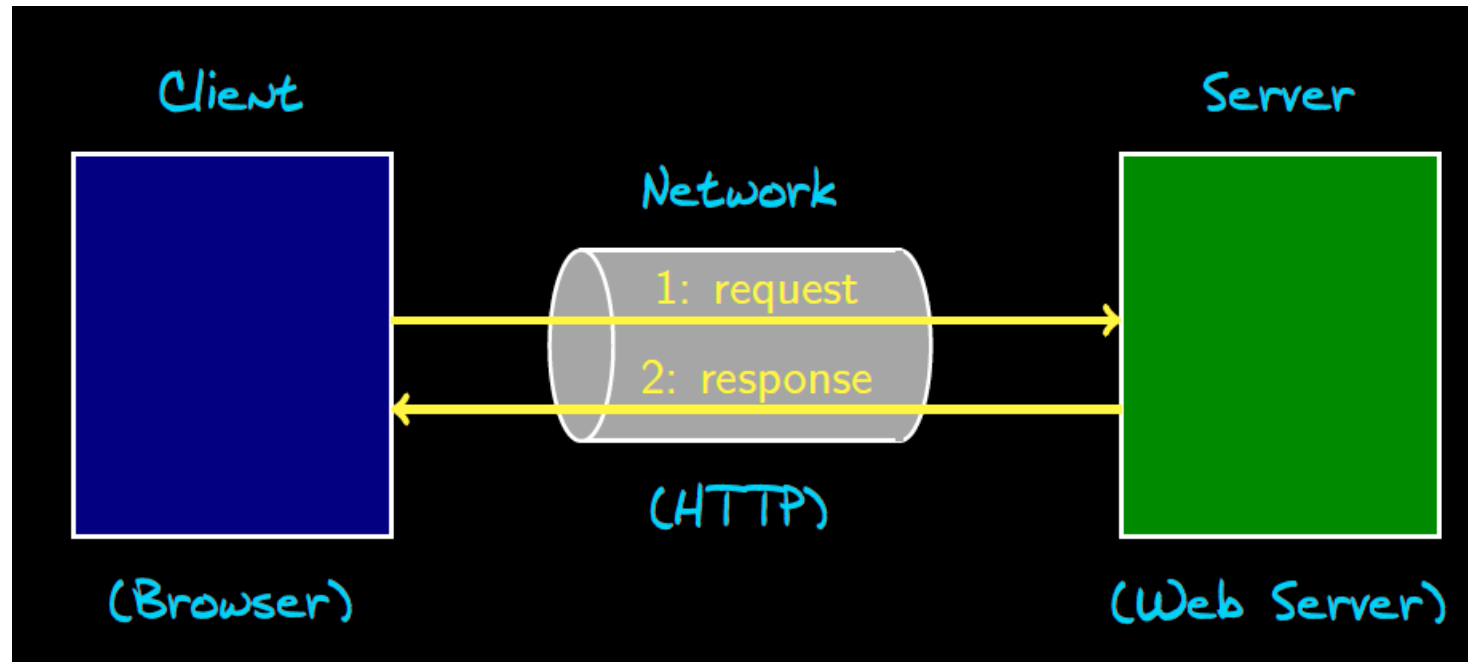


Different Web Application Architectures

- Client-Server Architecture
- 3-Tier Architecture
- 6-Tier Architecture

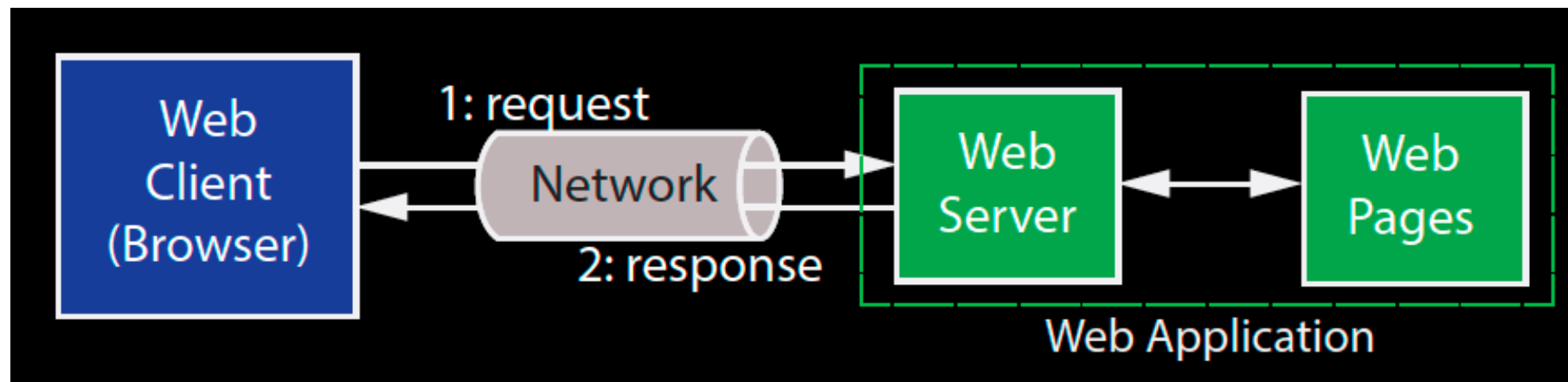
Client-Server Architecture

- The most basic model
- Server component
 - “listens” for request, and provides services and/or resources accordingly
- Client component
 - Establishes a connection to the server, and requests services and/or resources from it.



Web 1.0 Architecture

- Web 1.0 essentially used the client server architecture.
- The web server is primarily fetching static web pages
 - Not much interactivity
- No separation of data from its presentation
- The browser is very simple
 - It only needs to render HTML

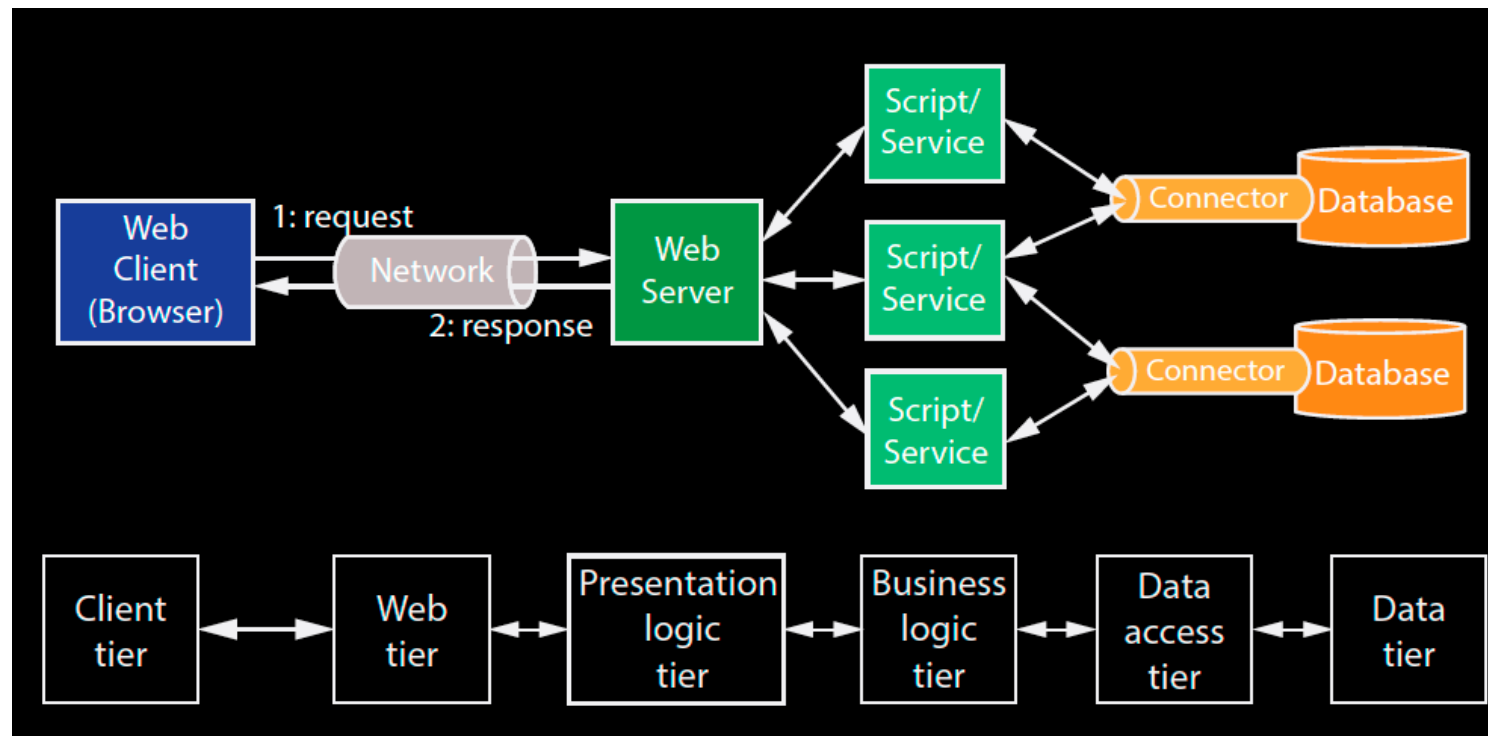


3 Tier Architecture

- The 3-tier architecture is one of the most common
 - Presentation tier
 - The user interface
 - Application (logic) tier
 - Retrieves, modifies and/or deletes data in the data tier, and sends the results to the presentation tier. Also responsible for processing the data itself.
 - Data tier
 - The source of the data associated with the application.
- The modern web application is often deployed over the Internet as a 3-tier architecture
 - Presentation tier
 - Users web browser
 - Application (logic) tier
 - The web server and logic associated with generating dynamic web content, e.g., collecting and formatting the results of a search.
 - Data tier
 - A database

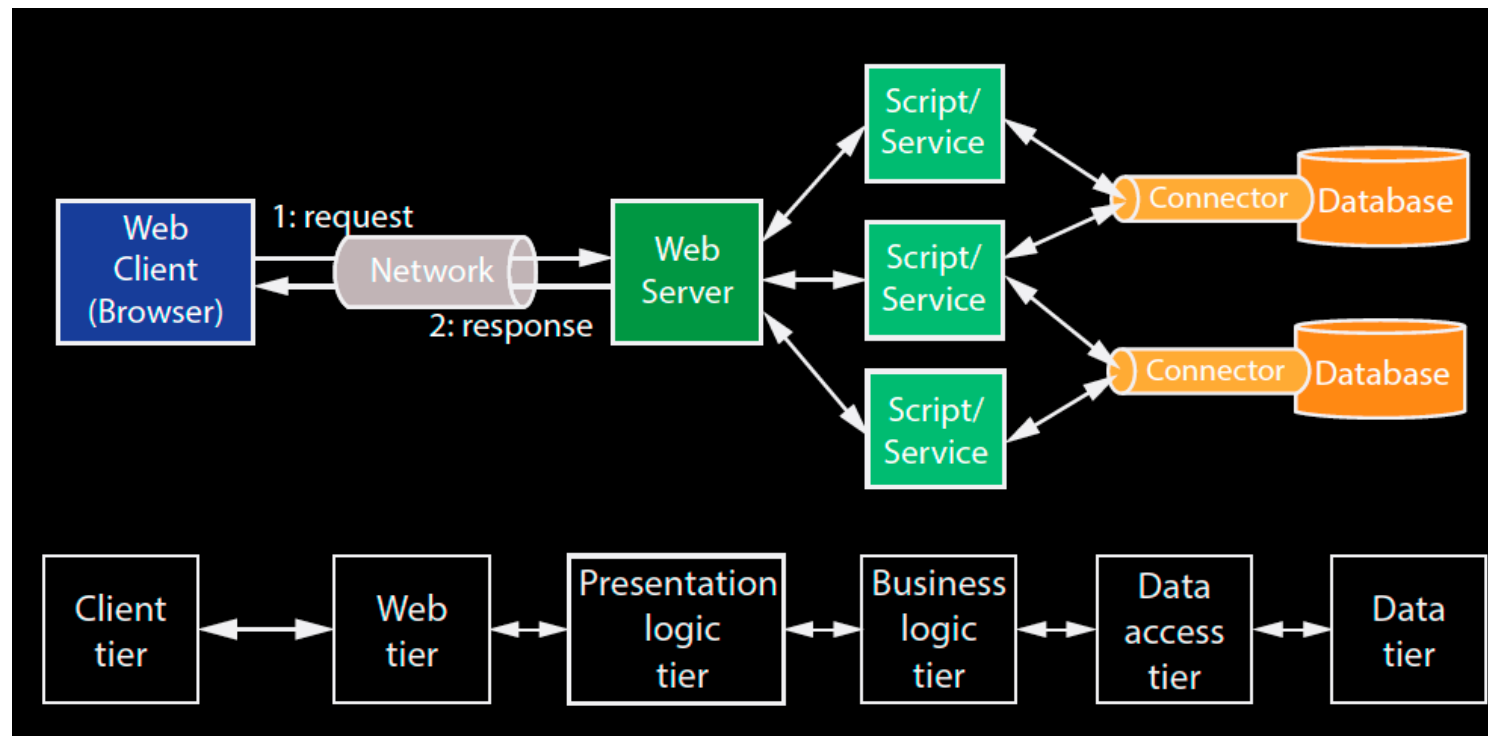
6-Tier Web Application Architecture

- The Application tier is often subdivided into two tiers
 - **Business logic tier** – Models the business objects associated with the application, e.g., accounts, inventories, etc., and captures the business rules and workflows associated with how these processes can be processed and manipulated.
 - **Data access tier** – Responsible for accessing data, and passing it to the business logic tier, e.g., account balances, transactions, etc.
 - This allows different databases to be swapped in and out more easily.

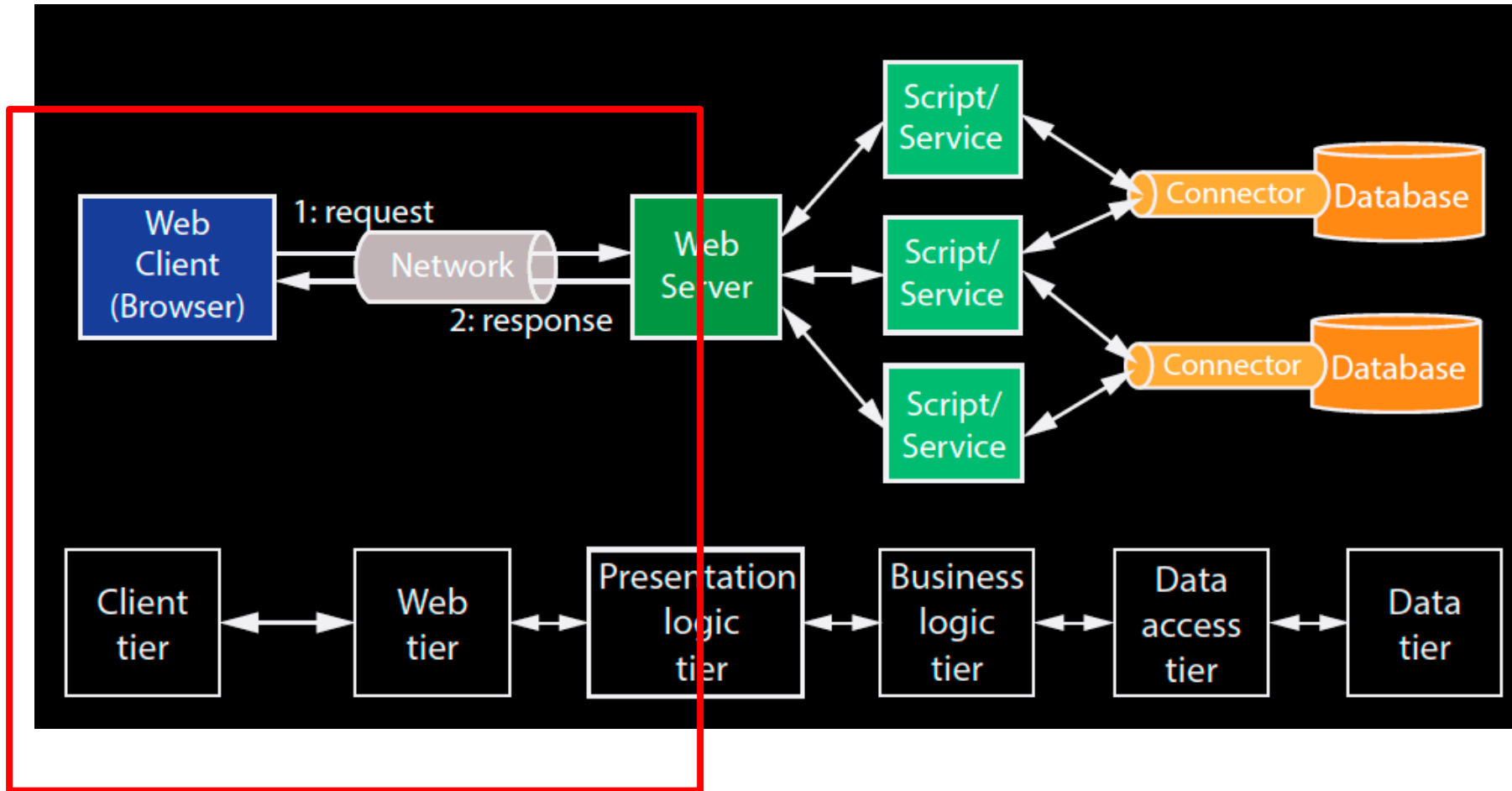


6-Tier Web Application Architecture

- The Presentation tier is often subdivided into two tiers:
 - **Client tier** - client-side user interface components.
 - **Presentation logic tier** – server-side scripts for generating web pages.
- Finally, the web server is often separated out into its own Web tier.



The Presentation Tier

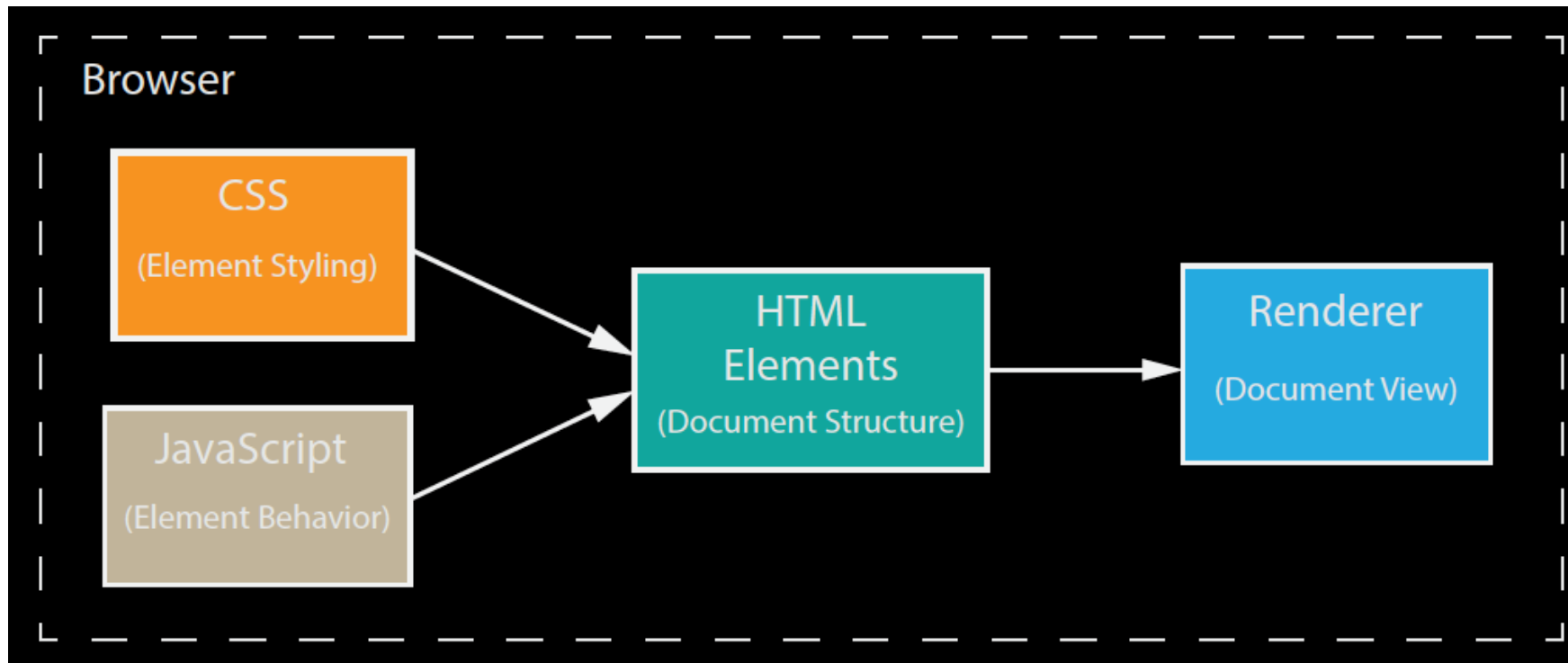


The Presentation Tier

- The presentation tier (presentation logic and client tiers) produces what users see and interact with via the browser in the web application
- A high-quality website has
 - A visually appealing layout
 - An intuitive navigation structure
- It's beyond the scope of this course to consider web site design and navigation structure in detail
 - Designing layouts from scratch requires some graphic design skills

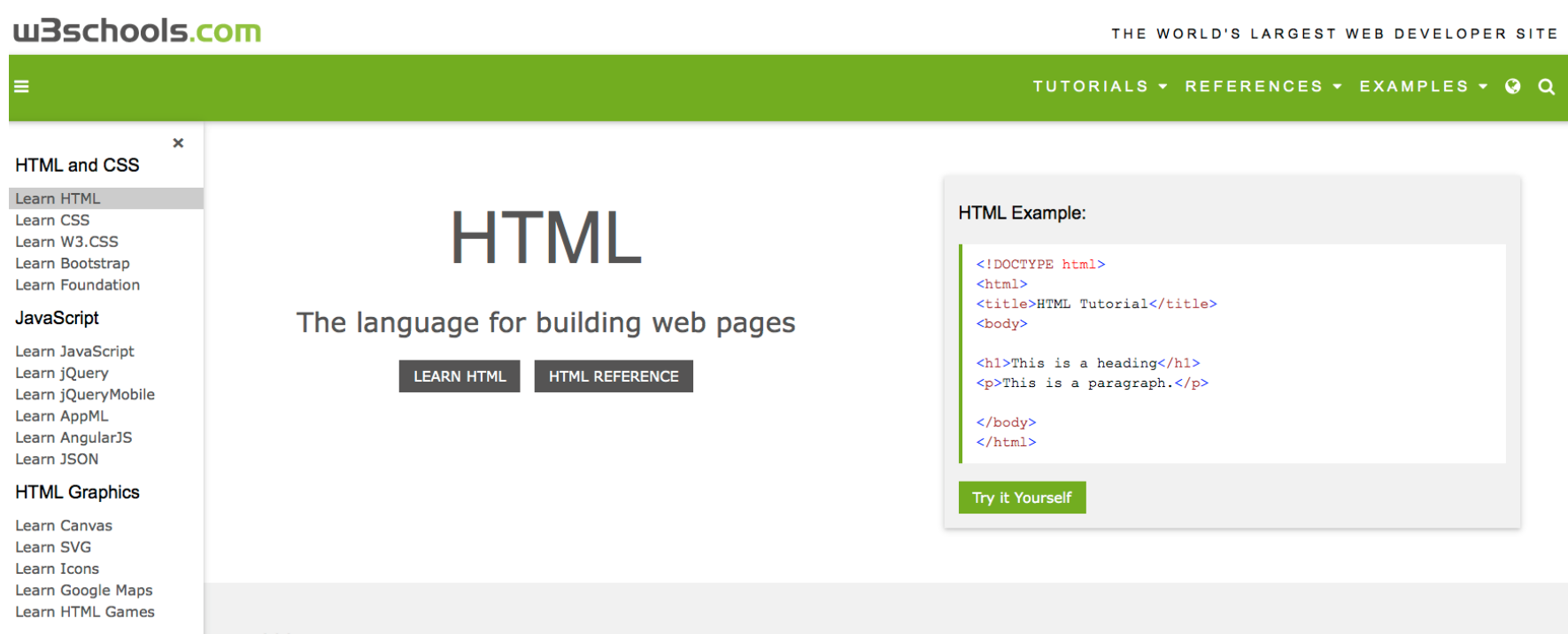
The Presentation Tier

- HTML – the language that is used to specify web pages
- CSS – the language used for styling web pages
- JavaScript – the language used for adding behavior to pages



Best Way to Learn HTML, CSS and Javascript

- By far the best way to learn HTML, CSS and Javascript is to just go to this web site
 - <http://www.w3schools.com>
- Then play around with all the examples. They have examples for everything!
- A lot of the slides for this subject are developed from these examples.



HTML is example of Declarative Programming

- HTML stands for Hyper Text Markup Language
- Markup languages are examples of declarative programming languages
 - Program flow control is *not* specified
 - Programs specify *what* not *how*
- For HTML this means that you specify *what* should appear on a webpage but *not* how it should look (i.e. how it's styled) or behave (e.g. when it's visible, animations).
- If created properly, HTML documents should follow the separation of content and presentation principle.
 - E.g. HTML

```
<div>  
  
    <h1> Introduction </h1>  
  
</div>
```

The HTML Element

- The `<html>` tag marks the opening of the outermost element associated with every HTML document.
- The `<html>` element is the root of the document — thus, it's the container for all of the HTML elements in a document.

```
<!DOCTYPE ...>
<html>
  <head>
    <!-- HTML head elements -->
  </head>
  <body>
    <!--HTML body elements -->
  </body>
</html>
```

HTML Document Structure

Every HTML document has the following structure:

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <!-- the document head -->
```

```
  </head>
```

```
  <body>
```

```
    <!-- the document body -->
```

```
  </body>
```

```
</html>
```

HTML Document Structure

- The **DOCTYPE** declaration must be the first thing that appears in an HTML document.
- This declaration is *not* an HTML tag, it lets the web browser know what version of HTML the page is written in.
- For HTML5, use the following declaration:
`<!DOCTYPE html>`
- Older versions of the HTML standard will have more information:

Ex. HTML 4.01 Strict –

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
    "http://www.w3.org/TR/html4/strict.dtd">
```

HTML – Elements

- Each individual component in an HTML document is represented by an HTML **element**.
- An HTML document is structured as a tree of elements.
- An HTML elements consists of
 - A **start tag** that must contain the element's name, and may also contain additional **attributes**.
 - A **end tag** that matches the name provided in the start tag.
 - The **content** that may appear between the start and end tags
 - E.g.

```
<div id = "main-panel">  
    <h1> Introduction </h1>  
</div>
```

The Head Element

- The `<head>` element is the container for head elements.
 - A required head element is the document title, specified using the `<title>` element.
- Other common head elements include:
 - `<link>` – used to link with external style sheets. This element is always empty, it can only contain attributes.
 - Eg. `<link rel="stylesheet" href="theme.css">`
 - The `rel` attribute specifies the relationship between the document and the linked resource.
 - The `href` attribute specifies the location (URL) of the external resource. The URL may be:
 - `absolute` – pointing to another web site.
 - `relative` – pointing to a file within the web site.

Common Head Elements

- `<script>` – used to define a client-side script, or to include one from an external source, typically this is JavaScript.

E.g.

```
<script>
    function message() {
        document.write("Hello World!")
    }
</script>
```

Eg. If `src` is present, the `<script>` element must be empty.

```
<script src="myscripts.js"></script>
```

Again, the URL can be either absolute or relative.

Common Head Elements

- **<meta>** – used to provide metadata within an HTML document.
 - **Metadata** is data about data — it is machine parable, but will not be displayed in the webpage.
 - **Metadata** can be used by browsers (e.g., when to refresh the webpage), search engines (keywords), or other web services.

- E.g.

```
<head>  
  <meta charset="UTF-8">  
  <meta name="description" content="My MOOC">  
  <meta name="keywords" content="MOOC,Web App">  
  <meta name="author" content="G. Heileman">  
</head>
```

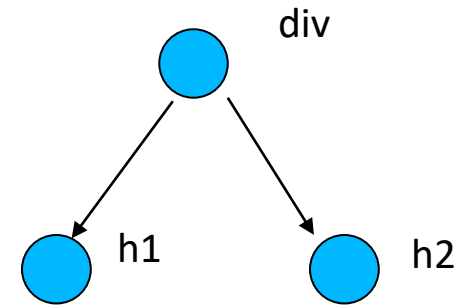
The Body Element

- The <body> element contains the HTML elements that will actually be rendered in the browser.
 - I.e., the body holds the actual “content” of the HTML document.
- Every HTML body element is classified as being either a **block-level** or a **text-level (inline)** element.
- Block-level elements define the major structure of a Web page, e.g., headings, paragraphs, etc., and they always produce a new line in the document.
- Inline elements define the minor structure of the web page, e.g., bold or emphasized text, and they do not produce a new line in the document.

HTML – Elements

- Elements may appear nested inside other elements
 - For example

```
<div id="main-panel">  
<h1> Introduction </h1>  
<h2> History of HTML </h2>  
</div>
```



- The h1 and h2 elements are nested inside the div element , so h1 and h2 are the children of div, and div is the parent of h1 and h2.
- These parent-child relationships lead to the tree structure of HTML documents.
- In the next set of slides we show many example tags
- Note: A listing of all available HTML tags can be found at: www.w3schools.com/tags/

Heading and Paragraph Elements

Web page:

This is a Heading

This is a paragraph.

Corresponding code:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

- The above example shows you the following
 - <h1> heading element
 - <p> paragraph element

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_default

Paragraph Element

Web page:

This is a paragraph.

This is a paragraph.

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<p>This is a paragraph.
<p>This is a paragraph.

</body>
</html>
```

- The above example shows you the following
 - <p> paragraph element

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_no_endtag

Heading Elements

Web page:

This is heading 1

This is heading 2

This is heading 3

This is heading 4

This is heading 5

This is heading 6

- The above example shows you the following
 - <h1>, <h2>, ... heading elements

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>

</body>
</html>
```

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_headings

Comments

Web page:

This is a paragraph.

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<!-- This is a comment -->
<p>This is a paragraph.</p>
<!-- Comments are not displayed in the browser -->

</body>
</html>
```

- The above shows you how to write comments. Comments are enclosed in the following element tags <!-- and -->

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_comment

Attributes

- Attributes give elements additional meaning and context.
- Attributes appear as a list of name-value pairs inside the start tag, where the value must be enclosed in quotes, and the name and value are separated by an equal sign.
 - E.g.

```
<div id="main-panel" class="modern">  
  <h1> Introduction </h1>  
</div>
```
- All HTML elements support id attribute (i.e., it's a global attribute). An id must be **unique** within an HTML document.
- Another global attribute is **class**, which can be applied to a **collection** of elements.

HTML Classes

Code:

```
<!DOCTYPE html>
<html>
<head>
<style>
div.cities {
    background-color:black;
    color:white;
    margin:20px;
    padding:20px;
}
</style>
</head>
<body>

<div class="cities">
<h2>London</h2>
<p>London is a city the UK</p>
</div>

<div class="cities">
<h2>Paris</h2>
<p>Paris is the capital of France.</p>
</div>
</body>
</html>
```

Corresponding Web page:

London

London is a city the UK

Paris

Paris is the capital of France.

- In this example we use the class attribute "cities" to group the two blocks of text together.
- Then we can set the style for the blocks with the cities attribute at the top using **div.cities**

Links

Web page:

[Visit our HTML tutorial](http://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_w3schools)

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<a href="http://www.w3schools.com/html/">Visit our HTML tutorial</a>

</body>
</html>
```

- The above show you how to put links into HTML documents. It is done using the following format
 - ` text `
 - Where URL is the string representing the URL to the web page you want to link to.
 - text should be replaced with actual text you want to displayed for the link.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_w3schools

Images

Web page:

Spectacular Mountain



Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<h2>Spectacular Mountain</h2>


</body>
</html>
```

- Images can be embedded into HTML using the element

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_mountain

Tables

Web page:

Jill	Smith	50
Eve	Jackson	94
John	Doe	80

- The whole table code needs to be enclosed inside `<table ...>` and `</table>`
- Each row of the table is contained inside `<tr>` and `</tr>`
- Each element of each row is contained inside `<td>` and `</td>`

- http://www.w3schools.com/html/tryit.asp?filename=tryhtml_table

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<table style="width:100%">
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>80</td>
  </tr>
</table>

</body>
</html>
```

Unordered Lists

Web page:

Unordered List with Default Bullets

- Coffee
- Tea
- Milk

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<h2>Unordered List with Default Bullets</h2>

<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>

</body>
</html>
```

- The whole unordered list is enclosed within the following
 - and
- Each element of the list is enclosed within in the following
 - and
- http://www.w3schools.com/html/tryit.asp?filename=tryhtml_lists_unordered

Ordered Lists

Web page:

Ordered List

1. Coffee
2. Tea
3. Milk

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<h2>Ordered List</h2>

<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>

</body>
</html>
```

- The whole ordered list is enclosed in the following
 - and
- Each element of the list is enclosed within in the following
 - and
- http://www.w3schools.com/html/tryit.asp?filename=tryhtml_lists_ordered

HTML Blocks (div)

Web page:



Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<div style="background-color:black; color:white; padding:20px;">

<h2>London and Melbourne</h2>

<p>London is a city in the UK</p>
<p>Melbourne is a city in Australia</p>

</div>

</body>
</html>
```

- Using `<div .. > ... </div>` you can specify a block of code. Then you can apply a style to the entire block.
- http://www.w3schools.com/html/tryit.asp?filename=tryhtml_div_capitals

HTML Inline (span)

Web page:

My **Important** Heading

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

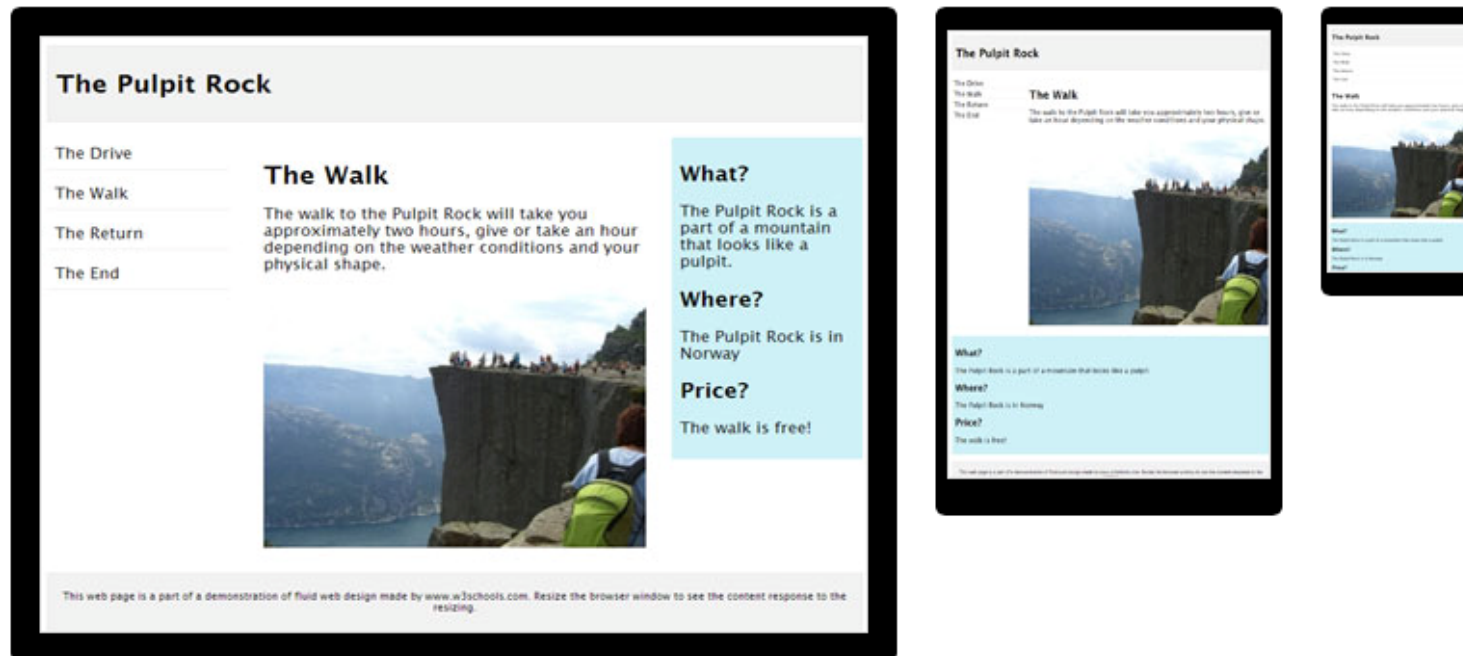
<h1>My <span style="color:red">Important</span> Heading</h1>

</body>
</html>
```

- The HTML span tag is an inline element that is often used as a container for some text
 - An inline element does not start on a new line and only takes up as much width as necessary
- http://www.w3schools.com/html/html_blocks.asp

Responsive Web Design

- The aim of responsive web design is to make web pages look good on all devices (desktop, tablet, phones)
- Responsive Web Design is about using CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.



Responsive Web Design

- Play around with the first example shown on this page to see how it works.
 - http://www.w3schools.com/html/html_responsive.asp
- In this subject we will not be talking any more about responsive web design. To learn responsive web design properly you will need to take some other course or find resources on the internet.

HTML Formatting

Web page:

This text is normal.

This text is strong.

This text is italic.

This text is emphasized.

HTML Small Formatting

HTML **Marked** Formatting

My favorite color is ~~blue~~ red.

My favorite color is red.

This is _{subscripted} text.

This is ^{superscripted} text.

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<p>This text is normal.</p>
<p><strong>This text is strong.</strong></p>
<p><i>This text is italic</i>.</p>
<p><em>This text is emphasized</em>.</p>
<h2>HTML <small>Small</small> Formatting</h2>
<h2>HTML <mark>Marked</mark> Formatting</h2>
<p>My favorite color is <del>blue</del> red.</p>
<p>My favorite <ins>color</ins> is red.</p>
<p>This is <sub>subscripted</sub> text.</p>
<p>This is <sup>superscripted</sup> text.</p>

</body>
</html>
```

HTML Styling

Web page:

This is a paragraph.

This is a paragraph.

This is a paragraph.

This is a paragraph.

Centered text

- The **style** attribute can be used to change the style of the different HTML elements. The types of style changes include
 - Background color
 - Font
 - Text color
 - Size
 - Alignment (left, center, right)

Corresponding code:

```
<!DOCTYPE html>
<html>
<body style="background-color:lightgrey;">

<p>This is a paragraph.</p>
<p style="color:red;">This is a paragraph.</p>
<p style="font-family:courier;">This is a paragraph.</p>
<p style="font-size:160%;">This is a paragraph.</p>
<h2 style="text-align:center;">Centered text</h2>

</body>
</html>
```

Problem with using HTML Styling

- The Tags only provided limited formatting.
 - H1 is always that big.
 - Links always have an underscore.
- You can manually change the format for all the H1 headlines, but what if you have 100 pages in your web site?
- Can we customize the HTML tags?

Cascading Style Sheets (CSS)

Separate Content from Styling

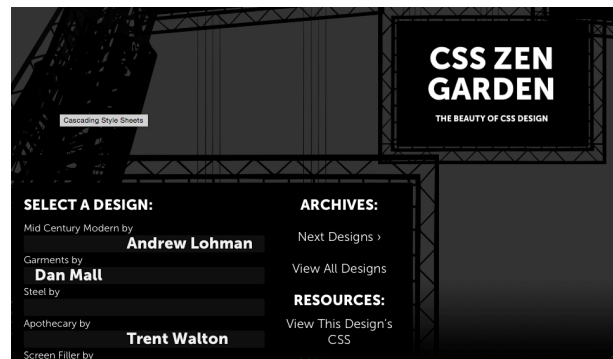
- Using cascading style sheets (CSS) allows you to separate content from styling
 - The HTML document just contains the contents
 - The CSS sheet only contain the styling
- This allows you to apply the style to many different web pages easily
- This also allows you to apply different styles to the same web page.

CSS Zen Garden



A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.

Download the example [HTML FILE](#) and [CSS FILE](#)

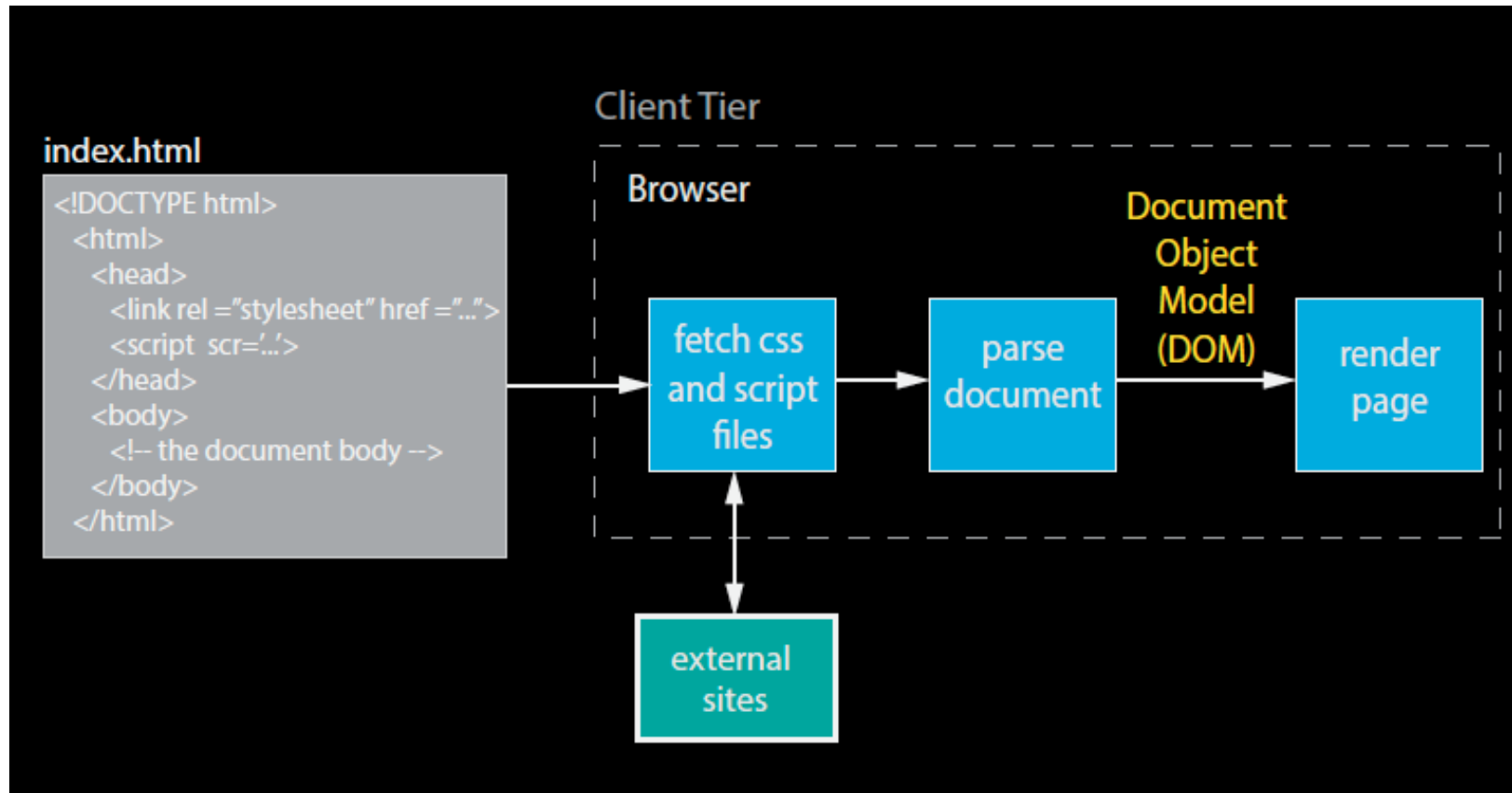


- One of the best demonstrations of the power of CSS is the following web site
 - <http://www.csszengarden.com>
- Here you can try out many different CSS style sheets that are all used to render the **same** HTML content.

Cascading Style Sheets (CSS)

- What is style?
 - Style is a list of formatting instructions.
- A Cascading Style Sheet is a file with a list of formatting instructions.
- CSS style sheets are the modern way to control the appearance and layout of your web pages.

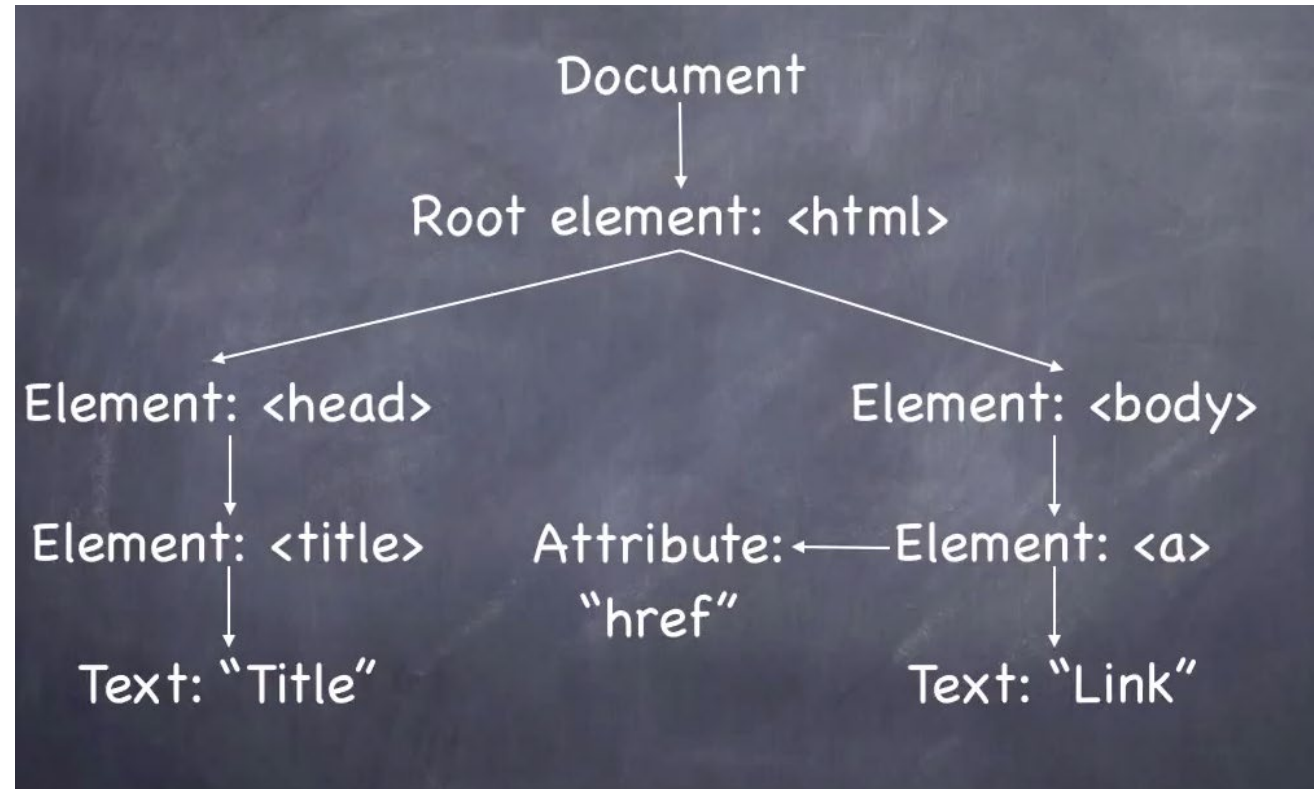
Displaying a Document



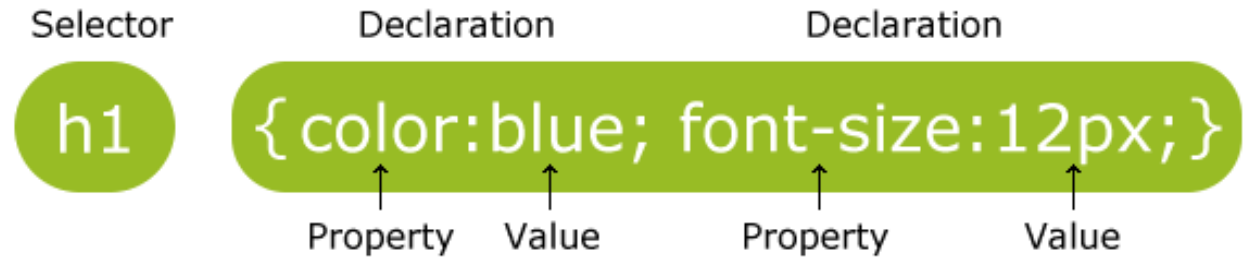
- Your browser first loads the HTML document.
 - Inside the HTML document is a reference to a style sheet
 - The HTML document then loads the style sheet
 - The HTML document is then parsed with the styles applied
 - Next a DOM is created and is finally used to render the page.

The Document Object Model (DOM) Tree

- The document object model (DOM) is the elements of your HTML document organized as a tree.
- Using the DOM we can easily specify which specific element of an HTML document we want to modify or apply a particular style to.



CSS Rules



- CSS involves creating rules that specify how particular HTML elements should appear.
- A CSS rule has the following syntax:
 - selector {declaration}
- where,
 - selector – specifies the elements the rule will be applied to.
 - declaration – a semicolon separated list of property:value pairs.

Eg. `h1, h2 {
 font-weight:bold;
 font-family:arial;
 color:black
}`

- The formatting declarations shown in this CSS rule will be applied to all level 1 and level 2 headings in the HTML document.
- Note: Invalid CSS rules are simply ignored.

Three Common Selectors

- Element Selector – applies to elements of a given type
 - `h1 {color:purple} /* h1 elements purple */`
 - `h2, b {color:red} /* h2, b elements red */`
- Class Selector – applies to elements of a given class
 - `p.main {font-style:normal} /* p elems, class main */`
 - `.main {color:red} /* all elements, class main */`
- ID selector – applies to elements with a given ID
 - `#chapter1 {text-align:center}`
- A list of additional selectors can be found at:
 - http://www.w3schools.com/cssref/css_selectors.asp

Example Class Selector

Web page:

Blue heading

Blue paragraph.

Not Blue paragraph.

Corresponding code:

```
<!DOCTYPE html>
<html>
<head>
<style>
.bluec {
    color: blue;
}
</style>
</head>
<body>

<h1 class="bluec">Blue heading</h1>
<p class="bluec">Blue paragraph.</p>
<p >Not Blue paragraph.</p>

</body>
</html>
```

- The class selector lets you select a group of elements to apply the same style to.
- In the above example the <h1> and first <p> elements both have their class attribute set to bluec. This is then used by CSS to specify the blue color to them.

Example Element Selector

Web page:

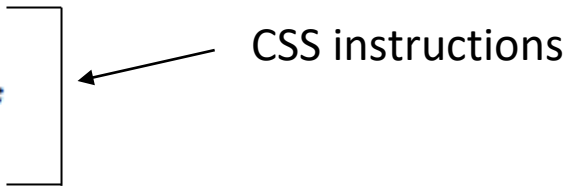
Hello World!

This paragraph is styled with CSS.

Corresponding code:

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
    color: red;
}
</style>
</head>

<body>
<p>Hello World!</p>
<p>This paragraph is styled with CSS.</p>
</body>
</html>
```



CSS instructions

- In this example the element selector is used to apply a format (**red font**) to all elements with tag **p**.

Example ID Selector

Web page:

Hello World!

This paragraph is not affected by the style.

Corresponding code:

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
    color: blue;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```

- In this example the ID selector is used to apply a format (blue font) to only one particular element (the element with attribute id set to para1)

Referencing an external CSS file

CSS style specified within document:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      color: blue;
    }
  </style>
</head>
<body>
<p> This is a test <p>
</body>
</html>
```

external CSS file:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href = "mystyle.css">
</head>
<body>
<p> This is a test <p>
</body>
</html>
```

- The **<link>** tag is used to specify an external CSS style sheet.

Only Specific Elements will Be Effected

Web page:

This heading is affected

This paragraph is not affected.

Corresponding code:

```
<!DOCTYPE html>
<html>
<head>
<style>
p.center {
    text-align: center;
    color: red;
}
</style>
</head>
<body>

<h1 class="center">This heading is affected</h1>
<p class="center">This paragraph is not affected.</p>

</body>
</html>
```

- In the above example we used the p.center in the CSS part. This means that only elements of type <p> and belong to the class center will be centered.
 - Even though the heading is also of class center, it will not be centered.

Multiple classes per element

Web page:

This heading will not be affected

This is red and centered.

This is red, centered and large.

Corresponding code:

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
    text-align: center;
    color: red;
}

.large {
    font-size: 300%;
}
</style>
</head>
<body>

<h1>This heading will not be affected</h1>
<p class="center">This is red and centered.</p>
<p class="center large">This is red, centered and large.</p>

</body>
</html>
```

- In the above example the last paragraph belongs to 2 classes (center and large), therefore it will have styling of both classes applied.
- In the CSS part we specified how each of the two classes should be styled.

Grouping Selectors

```
h1 {  
    text-align: center;  
    color: red;  
}  
  
h2 {  
    text-align: center;  
    color: red;  
}  
  
p {  
    text-align: center;  
    color: red;  
}
```

In the above example we are applying the same styling to each type selector. A shorthand way of doing this is to apply one styling to a group of selectors as follows.

```
h1, h2, p {  
    text-align: center;  
    color: red;  
}
```

CSS Comments

```
p {  
    color: red;  
    /* This is a single-line comment */  
    text-align: center;  
}  
  
/* This is  
a multi-line  
comment */
```

More CSS Features

- To learn more CSS Features please play around in the w3school web site below.
 - <http://www.w3schools.com/css/default.asp>
- There is also some more examples in the appendix for lecture 3

HTML 5

See Appendix.

This will not be on the exam

HTML Forms

HTML Forms

- So far we've learned how to use HTML to create static websites.
- In this lecture we'll see how to build interactivity into websites.
- The first step is to figure out how to collect information from users - we do this by using the HTML form element.
- User information is collected in an HTML form via a web browser, and then submitted to a web server for processing.

HTML Forms

- A HTML form is a section of a document that may contain normal markup, as well as special elements called controls (checkboxes, radio buttons, drop-down lists, file selection, etc.), as well as labels for the controls.
- Users “complete” a form by modifying its controls (entering text or making selections).
- When a completed form is submitted, its data is first processed by a **user agent**, running as part of the **browser**, before it is actually submitted to a processing agent (e.g., a **web server** or a mail server) on the **server side**.

HTML Forms

- The basic structure of a form element is as follows:

```
<form action="http://www.example.com/log" method="get">  
  <!-- form controls and other HTML markup -->  
  <input type="submit" value="Log In">  
</form>
```

- The action attribute is used to specify the URL of the **processing agent** on the **server side** that will receive the data collected in the form.

HTML Form Method

```
<form action="http://www.example.com/log" method="get">  
  <!-- form controls and other HTML markup -->  
  <input type="submit" value="Log In">  
</form>
```

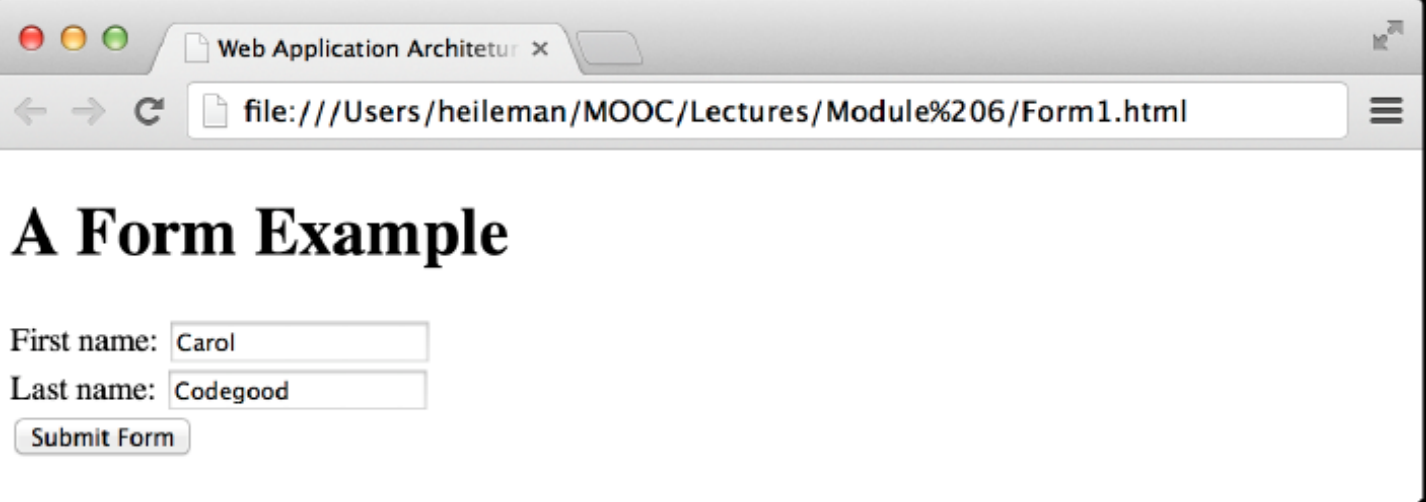
- The method attribute is used to specify the HTTP request method that the user agent will use to send the data. HTML user agents must support:
 - **GET** – in which case the form data must be sent as part of the URL, i.e., the data is URL-encoded, and appended to the request URL.
 - **POST** – in which case the form data must be included in the HTTP message request body.

For Submission – GET Request

- URL encoding works as follows:
 - The form data is separated from the URI by a “?”
 - Each name/value pair is separated by “&”
 - Each name is separated from its value by a “=” (“unsafe” characters, e.g., “/” and “&”, are escaped).
- E.g.
`www.example.com/?firstname=Carol&lastname=Codegood`

GET Example

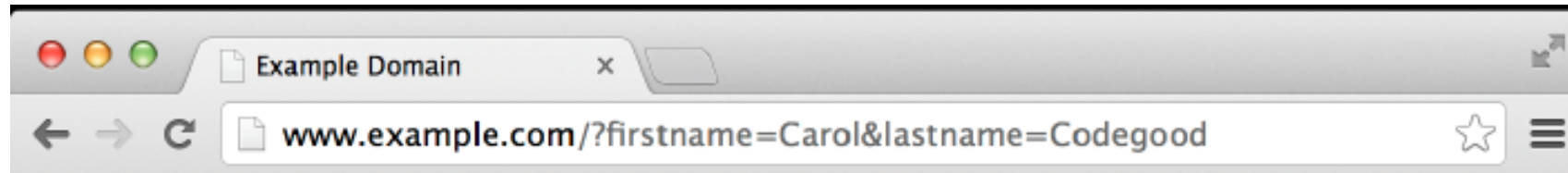
```
<form action="http://www.example.com" method="get">  
<label>  
    First name:<input type="text" name="firstname"><br>  
</label>  
<label>  
    Last name:<input type="text" name="lastname"><br>  
</label>  
<input type="submit" value="Submit Form">  
</form>
```



The screenshot shows a web browser window with a single tab titled "Web Application Architektur x". The address bar displays the file path: `file:///Users/heileman/MOOC/Lectures/Module%206/Form1.html`. The page content features a heading "A Form Example" in a large, bold, black serif font. Below the heading is a form with two text input fields. The first field is labeled "First name:" and contains the text "Carol". The second field is labeled "Last name:" and contains the text "Codegood". Below these fields is a button labeled "Submit Form".

Get Example

- Here is what happens after the form as been submitted.
- Notice the URL which encodes the information that is submitted to the processing agent.



Example Domain

This domain is established to be used for illustrative examples in documents. You may use this domain in examples without prior coordination or asking for permission.

[More information...](#)

The GET Request

- The GET method should be used when a form is idempotent (i.e., it does not cause side effects). E.g., if the form data is used to search a database, there are no side effects.
- You should not send sensitive data in a form using the GET method — the sensitive data will end up in the URL and anyone monitoring the network traffic can see it.
- The GET method should not be used if there is a large amount of form data, or if the form data contains non-ASCII characters or binary data.
- The GET method cannot be used if the form contains a file upload control — files cannot be passed in the URL.

GET is the default method

```
<form action="http://www.example.com">
```

Is the same as:

```
<form action="http://www.example.com" method="get">
```

The Importance of the name attribute

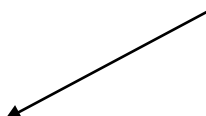
```
<!DOCTYPE html>
<html>
<body>

<form action="action_page.php">
  First name:<br>
  <input type="text" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

<p>If you click the "Submit" button, the form-data will be
sent to a page called "action_page.php".</p>

<p>Notice that the value of the "First name" field will not
be submitted, because the input element does not have a name
attribute.</p>

</body>
</html>
```



No name attribute

- In the above example the first input element does not have the **name** attribute specified. But the second input element does.
- The result is data from the first input element will not be sent to the server.

The Importance of the name attribute

First name:

Last name:

If you click the "Submit" button, the form-data will be sent to a page called "action_page.php".

Notice that the value of the "First name" field will not be submitted, because the input element does not have a name attribute.

- In the above example the first input element does not have the **name** attribute specified. But the second input element does.
- As can be seen below the result is data from the first input element will not be sent to the server.

Your input was received as:

The server has processed your input and returned this answer.

Use the back button in the browser to return to the example.



The HTML tutorial will not teach you how servers are processing input. Server input is explained in our PHP and ASP tutorials.

The POST Request

- If the server-side processing associated with the form **causes side effects**, e.g., modification of a database, subscription to a service, etc., then the **POST method** should be used.
- Furthermore, if the form data is sensitive, the HTTPS protocol should be established, so that the form data will be encrypted in the message body before it is sent to the server.

Post Request Example

```
<form action="http://www.example.com" method="post">
<label>
    First name:<input type="text" name="firstname"><br>
</label>
<label>
    Last name:<input type="text" name="lastname"><br>
</label>
<input type="submit" value="Submit Form">
</form>
```

- Same as previous example except method **post** is used instead of get.

Post Request Example

- The form looks the same in the browser as for the get request example.



The screenshot shows a web browser window with a single tab titled "Web Application Architectur...". The address bar displays the file path: "file:///Users/heileman/MOOC/Lectures/Module%206/Form2.html". The main content area features a heading "A Form Example" in a large, bold, black serif font. Below the heading is a form with two text input fields. The first field is labeled "First name:" and contains the text "Helen". The second field is labeled "Last name:" and contains the text "Hackmeister". Below these fields is a button labeled "Submit Form".

Web Application Architectur x

file:///Users/heileman/MOOC/Lectures/Module%206/Form2.html

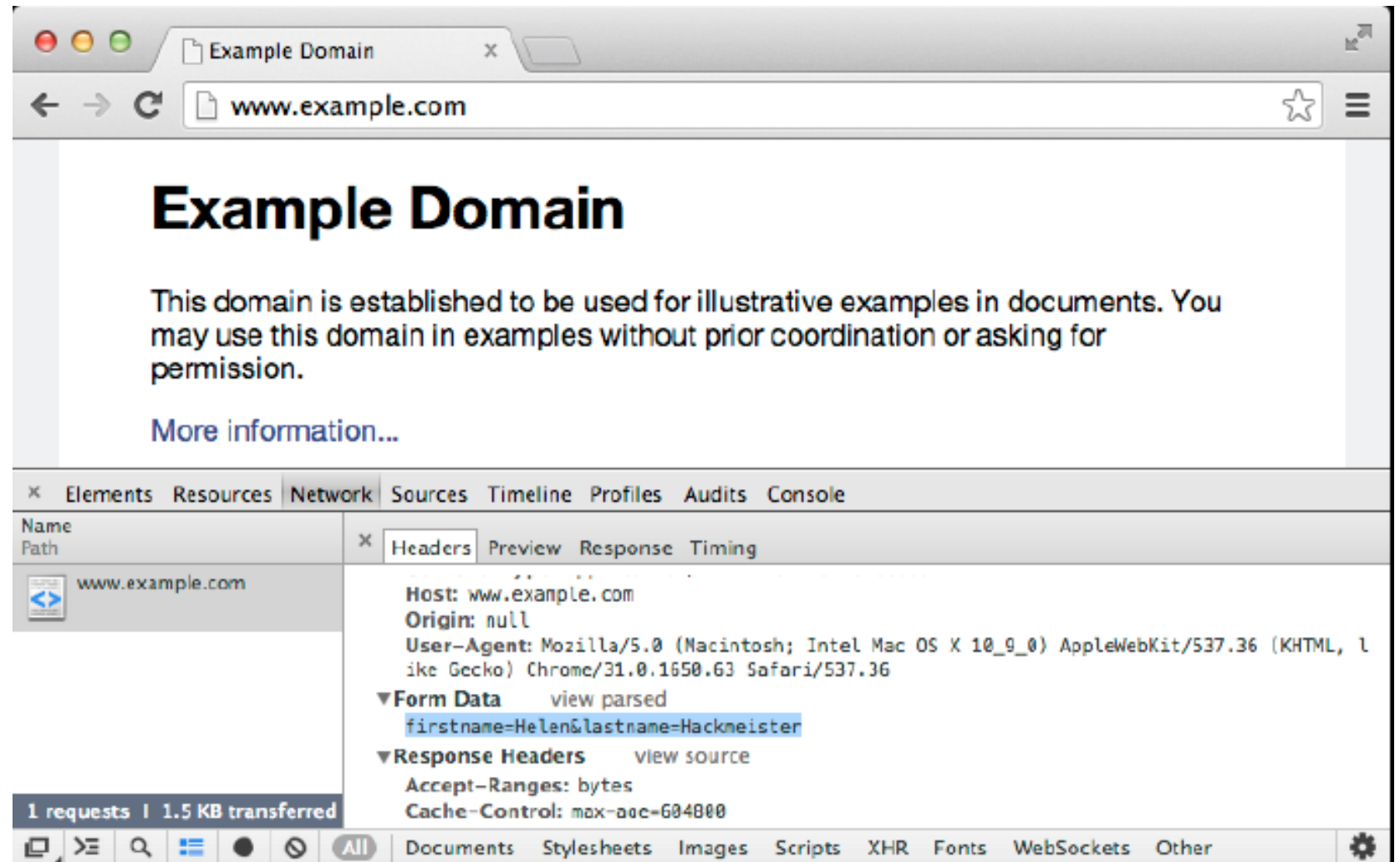
A Form Example

First name:

Last name:

Post Request Example

- The data is now in the message body instead of the URL.



Form Submission Process

- The user agent running in the browser identifies the successful controls, and builds a form data set — a sequence of control-name/current-value pairs for the successful controls.
- The form data set is encoded by the user agent according to the **content type** specified in the **enctype** attribute of the form element.
 - **application/x-www-form-urlencoded** — this is the default, form data is encoded as name-value pairs.
 - **multipart/form-data** — form data is encoded as a message, with a separate part for each control.
 - **text/plain** — form data is encoded as plain text.
- The user agent submits the encoded data set to the processing agent running on the server side using the HTTP protocol method specified by the **action** attribute in the form.

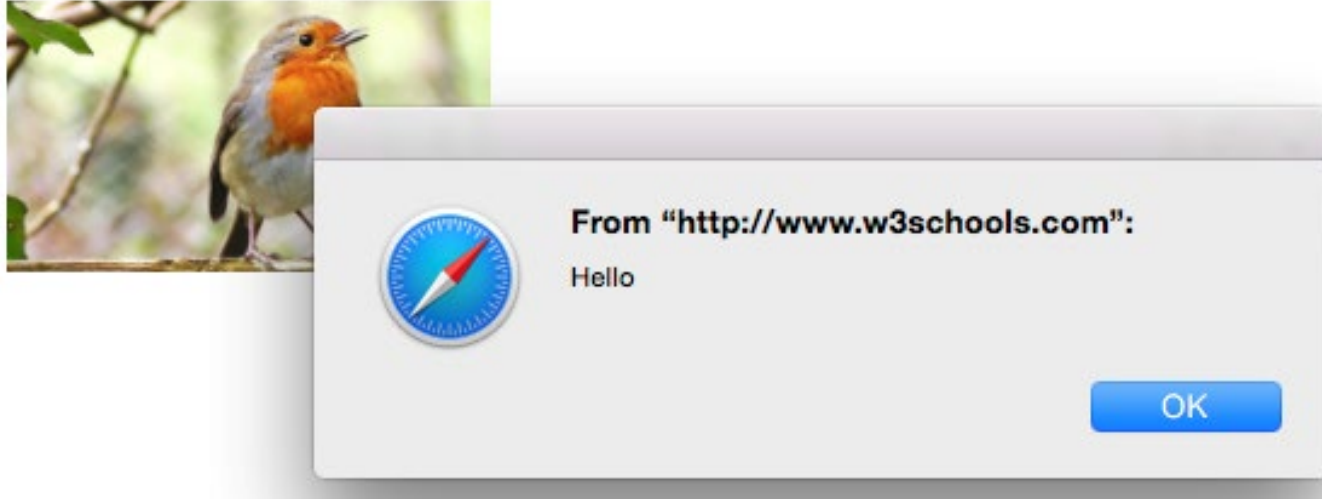
Form Controls

- Users interact with forms through named **form controls**.
- Form controls are specified using an input or select element that must appear in the content section of the form element, i.e., between the **<form>** and **</form>** tags.
- The name of a control is specified using the name attribute.
- A control has an **initial value** and a **current value**, both of which are
- character strings. The current value is first set to the initial value, but may change according to user supplied input.
- Available form controls include: text, date, buttons, checkboxes, radio buttons, select boxes (drop-down lists), file select boxes, hidden controls, etc. (see www.w3schools.com for a full list).

Form Controls (Buttons)

- **Button controls** are specified using either the button element or the input element. The type attribute, which should always be specified (as different browsers have different defaults for the type), has three possible values:
 - submit – Causes the form to be submitted.
 - reset – Causes the form to be reset, i.e. all controls are assigned their initial values.
 - button – Creates a push button, that typically has a client-side script associated with it through the event attribute. When the button is pressed and released, the associated script is executed.
- With the input element, the type attribute may be specified as an image. This creates a graphical submit button. The **src** attribute specifies the URL of the image file that will decorate the button.
 - E.g. `<input type="image" onclick="alert('Hello World!')" src="https://goo.gl/bkAvMR" width="200px">`

The role attribute



In this example the user clicks the picture of the bird and the popup appears.

One way to do this as explained in the previous slide is the following

```
<input type="image" onclick="alert('Hello World!)" src="https://goo.gl/bkAvMR"
width="200px">
```

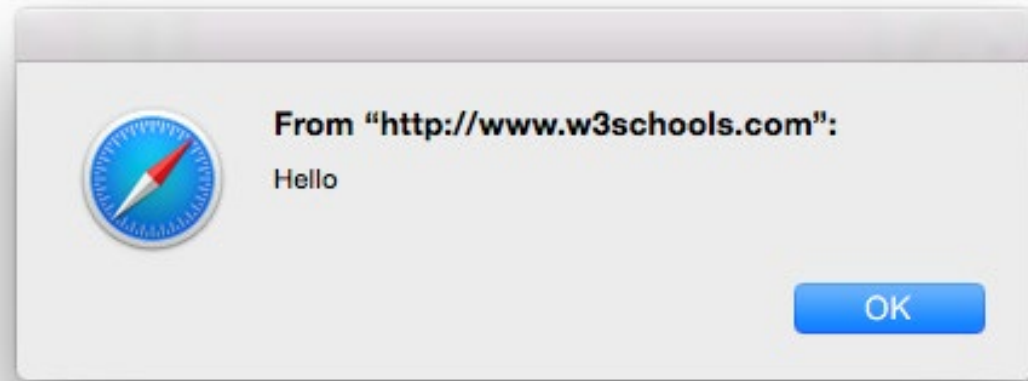
Another way to do it is to use the role attribute like the following:

```
<a role="button" onclick="alert('Hello')"> </a>
```

- The benefit of this second approach is that it is more flexible. Instead of making an image clickable we can also make anything else clickable. See next slide for an example of making heading clickable.

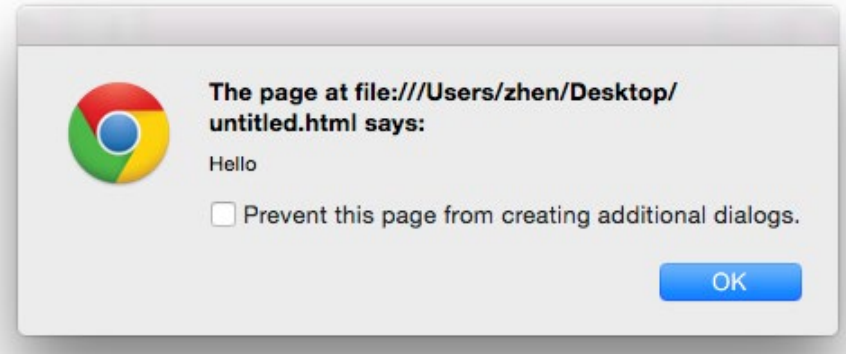
The role attribute

Please Click This Heading



- In this example the user clicks the heading and the popup appears.
- This can be achieved by the following line.
 - `<h1 role="button" onclick="alert('Hello')" style="color:red;"> Please Click This Heading</h1>`

The role attribute (change cursor to hand)



```
<!DOCTYPE html>

<html>

<head>

<style>

    [role=button] {
        cursor:pointer
    }

</style>

</head>

<body>

    <a role="button" onclick="alert('Hello')"> </a>

</body>

</html>
```

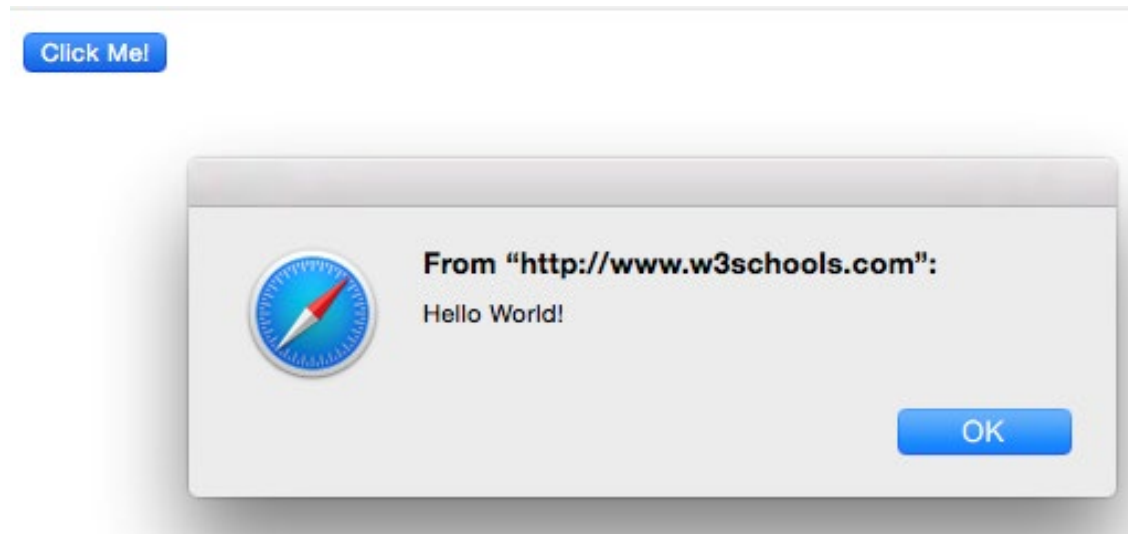
- This code uses CSS to style the attribute `role=button` so that the hand cursor is used when ever someone hovers over an element with `role button` set.
- Note the `[]` is used in CSS to select all elements that have a certain attribute set to a particular value.

Form Controls (Buttons)

- Button controls have numerous attributes that support **event-driven programming**.
- This programming style supports interactivity in browsers, i.e., you click a button (an event), the event is processed (typically by running a script) and something happens in the browser window.
- Some of the events attributes that can be specified for a button include: **onblur, onfocus, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onkeypress, onkeydown, onkeyup, onreset**.
- For each of these attributes, the value supplied is the script that should run when the corresponding events occurs.

Button and Onclick Event Example

Web page:



- In this example when the user clicks the button the **onclick event** is triggered.
- A window is popped up with the words **Hello World!** using the **alert** method.

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<button type="button" onclick="alert('Hello World!')">Click
Me!</button>

</body>
</html>
```

Form Controls (Radio button and Checkboxes)

- **Checkboxes** and **radio** buttons are specified using the input element.
 - These are essentially “on/off” switches that can be toggled by the user.
 - Several of these controls can share the same control name.
 - A switch is “on” when the control element’s checked attribute is set.
 - When a form is submitted, only the “on” checkbox and radio button controls are treated as successful.
 - If several radio button controls share the same name, they are treated as mutually exclusive. i.e., when one is switched “on” all of the others with the same name are switched “off.”
 - Multiple checkboxes with the same name may simultaneously be switched “on” in a form.

Radio Button Example

Web page:

☒ Male
☐ Female
☐ Other

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<form>
  <input type="radio" name="gender" value="male" checked>
Male<br>
  <input type="radio" name="gender" value="female">
Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>

</body>
</html>
```

Form Controls (Text Input and File Select)

- There are two types of **text controls**, both of them use the input element, and are specified using the type attribute:
 - **text** – creates a single line text input control.
 - **textarea** – creates a multi-line text input control.
- The **file select control** also uses the input element, and the type attribute has the value file. This control allows a user to select a file, whose contents will be submitted with the form.
- The **password input control** uses the input element, and the type attribute has the value password. With this control, user input is shown in the browser as dots or asterisks.

Text Area Example

Web page:

The cat was playing in the garden.

Submit

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<form action="action_page.php">
  <textarea name="message" rows="10" cols="30">
    The cat was playing in the garden.
  </textarea>
  <br><br>
  <input type="submit">
</form>

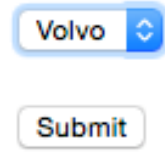
</body>
</html>
```

Form Controls (Select Boxes)

- Drop-down **select boxes** are specified using the select element.
 - Each choice offered by the menu is represented by an option element, and each select element must contain at least one option element.
 - The **optgroup** element allows several of the menu choices to be grouped together. These must appear directly within the select element, i.e., groups in general may not be nested.

Select Box Example

Web page:



A screenshot of a web form. It features a dropdown menu with the text 'Volvo' and a small blue arrow icon to its right. Below the dropdown menu is a rectangular button with the text 'Submit'.

Corresponding code:

```
<!DOCTYPE html>
<html>
<body>

<form action="action_page.php">
  <select name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
  <br><br>
  <input type="submit">
</form>

</body>
</html>
```

Conclusion

- We have learnt the basics of web architectures
- We also learn the basics of HTML
- CSS also styling commands to be applied to multiple pages
- HTML forms necessary for submitting information from the client to the server.