

The background of the slide features a blue gradient with a subtle texture. On the left side, there is a decorative pattern of white snowflakes of various sizes and orientations, resembling falling snow. A vertical white line runs down the center of the slide.

Introduction to HTML

R.Priyadarshini,
NITK

A decorative background on the left side of the slide features a blue vertical bar on the right containing several white, six-pointed snowflake icons of varying sizes, arranged in a scattered pattern.

□ What is front end?

Converting data into graphical interface

1 HTML

HTML, otherwise known as HyperText Markup Language, is the language used to create Web pages

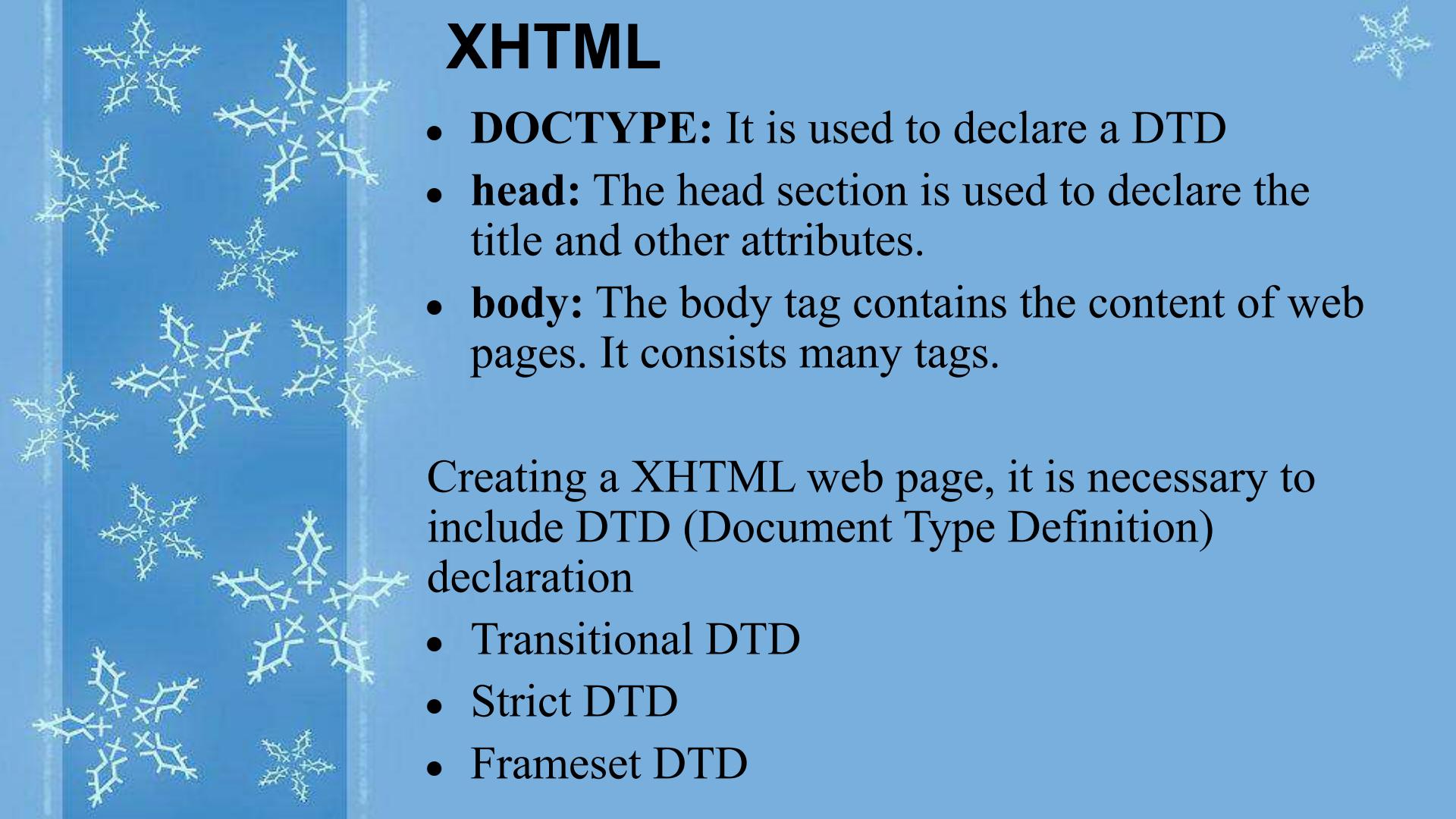
- Using HTML, you can create a Web page with text, graphics, sound, and video

2 XHTML

- XHTML stands for EXtensible HyperText Markup Language.
- It is the next step to evolution of internet.
- The XHTML was developed by World Wide Web Consortium (W3C).

3 HTML5

Fifth and last major HTML version that is a World Wide Web Consortium recommendation. HTML Living Standard. Maintained by a consortium of the major browser vendors. (Apple, Google, Mozilla, Microsoft)



XHTML

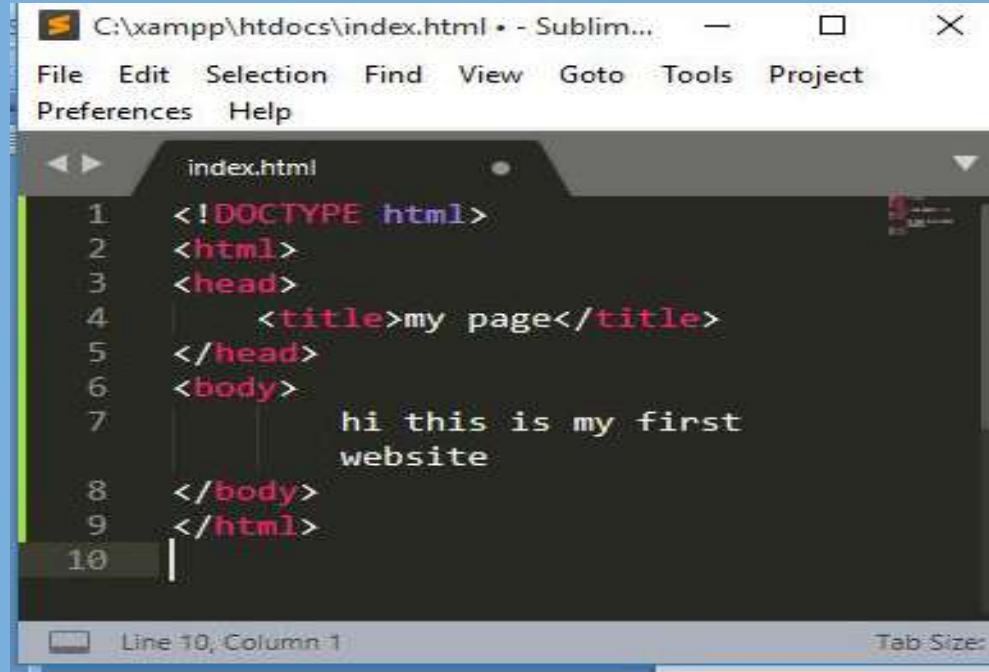
- **DOCTYPE:** It is used to declare a DTD
- **head:** The head section is used to declare the title and other attributes.
- **body:** The body tag contains the content of web pages. It consists many tags.

Creating a XHTML web page, it is necessary to include DTD (Document Type Definition) declaration

- Transitional DTD
- Strict DTD
- Frameset DTD

HTML

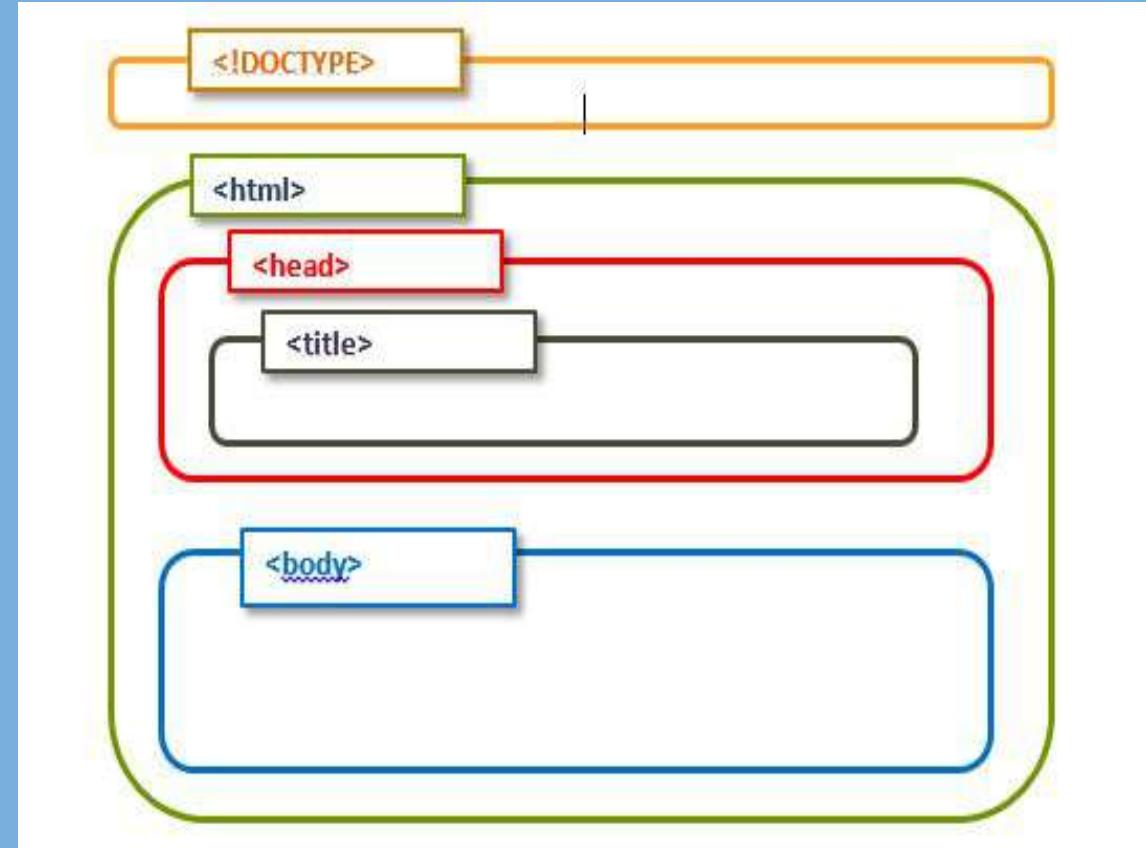
- A simple HTML document:



```
C:\xampp\htdocs\index.html - SublimeText  
File Edit Selection Find View Goto Tools Project  
Preferences Help  
index.html  
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4   <title>my page</title>  
5 </head>  
6 <body>  
7   hi this is my first  
8   website  
9 </body>  
10 </html>  
Line 10, Column 1 Tab Size: 4
```

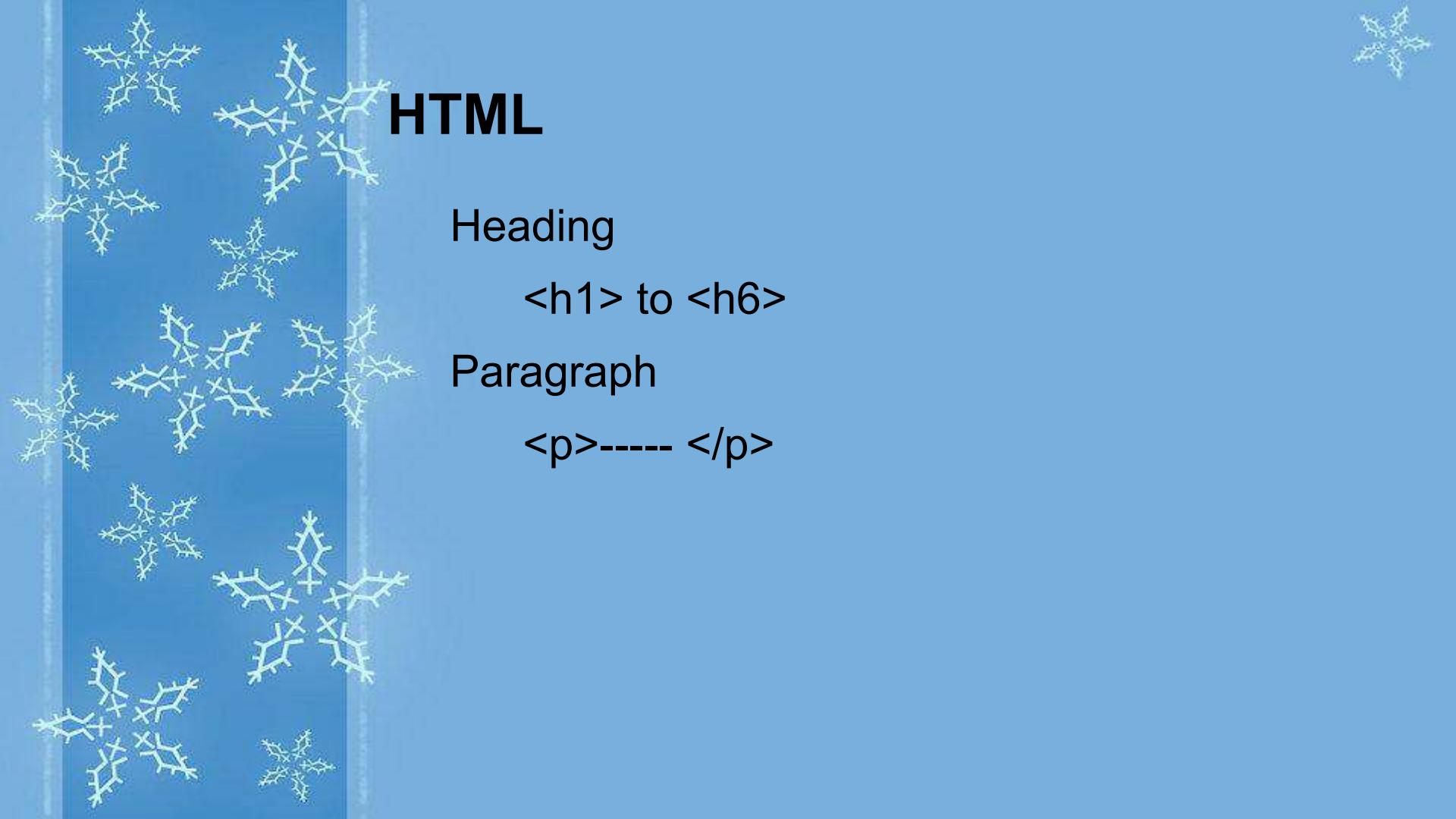
HTML

- HTML tags
 - < angular brackets>
- Page Structure



HTML EDITORS

- Note pad or text edit
- Professional HTML editors
- Notepad--□ write +save +run in web browser

The background of the slide features a blue gradient with white snowflake patterns of various sizes scattered across it. A vertical white line runs down the center of the slide.

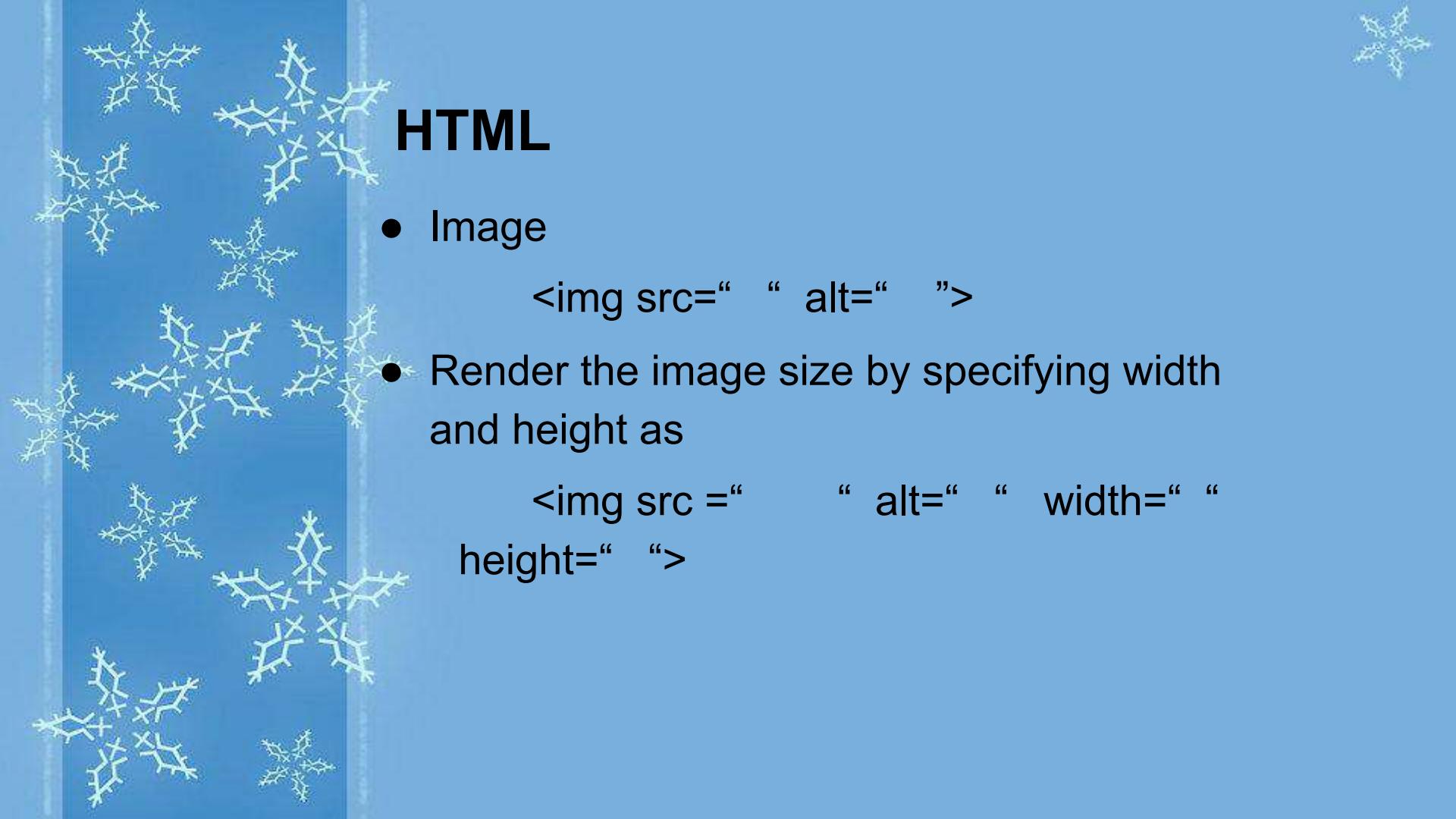
HTML

Heading

<h1> to <h6>

Paragraph

<p>----- </p>



HTML

- Image
``
- Render the image size by specifying width and height as

```
<img src = " " alt=" " width=" " height=" ">
```

The background of the slide features a blue gradient with numerous white snowflake icons of varying sizes scattered across it.

HTML

- Link : Interpage and Intrapage
 - Create a new page and provide the link
 - Specify the website address
 - Back to top (using id)
- Instead of text even images can be used as links.

The background of the slide features a blue gradient with white snowflake patterns of various sizes scattered across it.

HTML

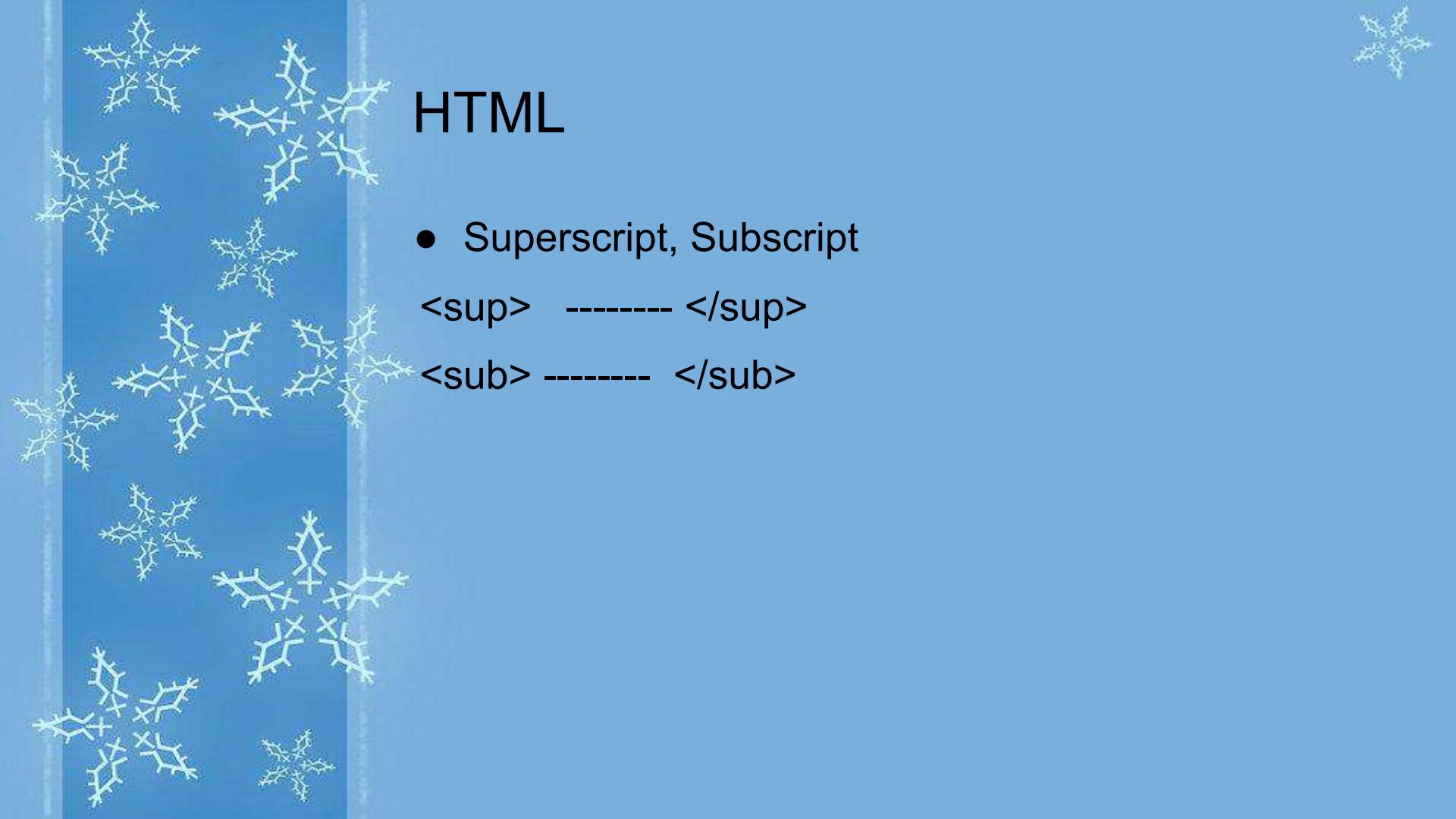
- Link:

```
<a href="https://www.nitk.ac.in/"  
target="_blank"> visit the new page </a>
```

to top :

```
<h1 id="top">this is here</h1>
```

```
<a href="#top">back to top</a>
```

The background of the slide features a blue gradient with numerous white snowflake icons of varying sizes scattered across it.

HTML

- Superscript, Subscript

```
<sup> ----- </sup>
```

```
<sub> ----- </sub>
```



HTML

- Strong , Emphasized

```
<strong> ----- </strong>
```

```
<em> ----- </em>
```

A decorative background on the left side of the slide features a blue gradient with numerous white snowflake icons of varying sizes and orientations.

HTML

- Special character

eg: copyright

```
<h1> copyright &copy; </h1>
```

The background of the slide features a blue gradient with numerous white snowflake icons of varying sizes scattered across it.

HTML

- List

- Ordered list

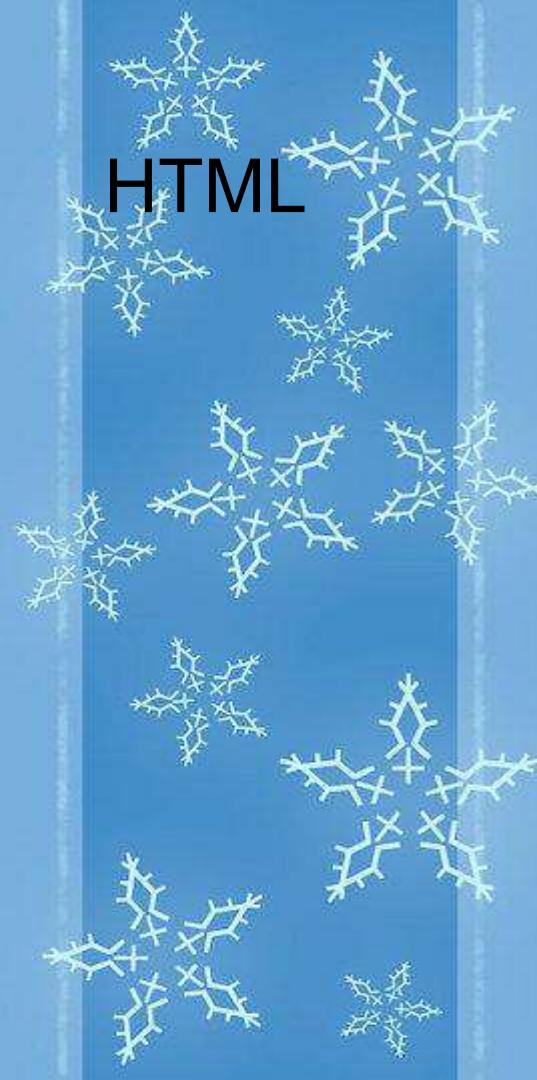
-
 - aaa
 - bbb
 -

- Unordered list

-
 - aaa
 - bbb
 -

HTML

- Table
 - Row <tr> </tr>
 - Column <td> </td>
 - Heading <th> </th>



HTML

- Form

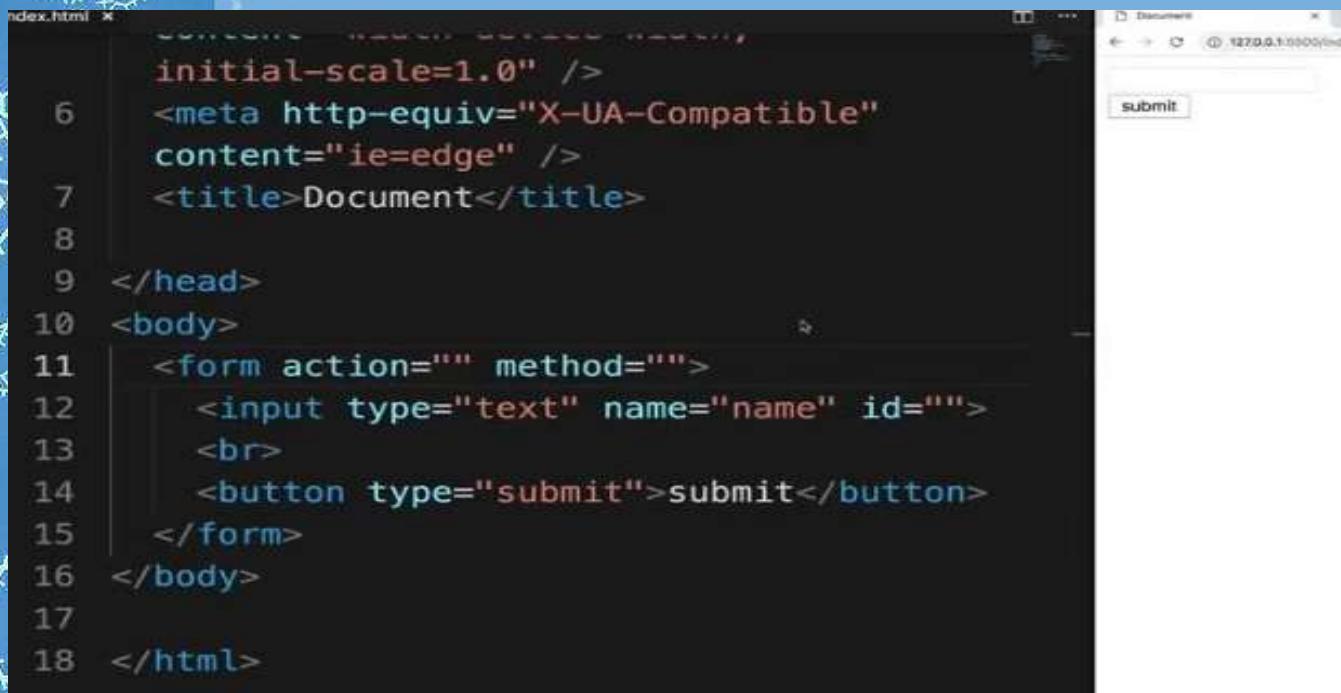
- Collect some kind of data
- <form action=" " method = " "> ----
</form>
- Action= where
- Method= how (get,Post)
- Input can be multiple type (text, email etc)
- <input type=" " name= " " id= " ">
- Type (any form) name (used for collecting data)
- id (Used for label)



HTML

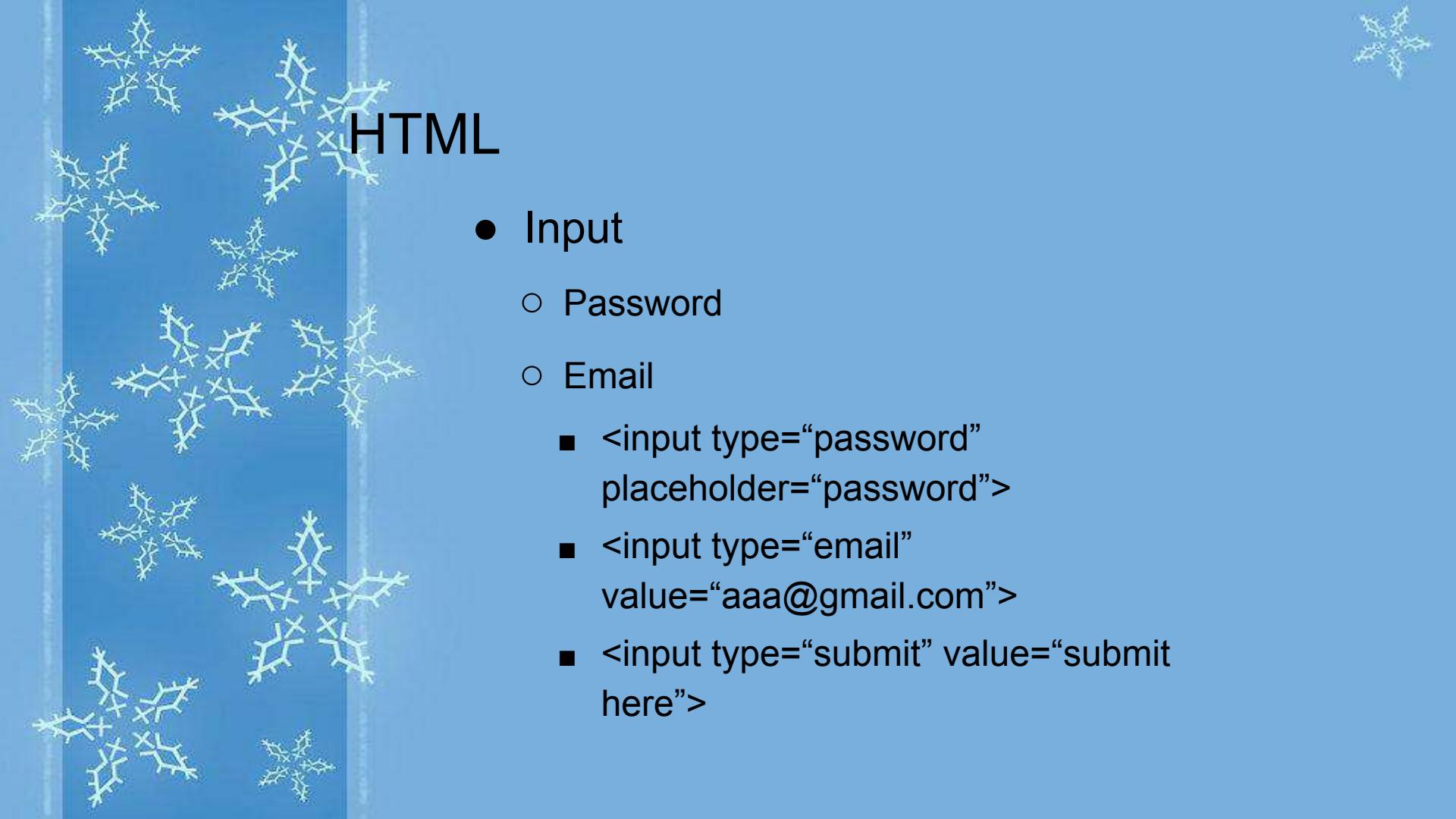
- Button

```
<button type="submit"> submit </button>
```



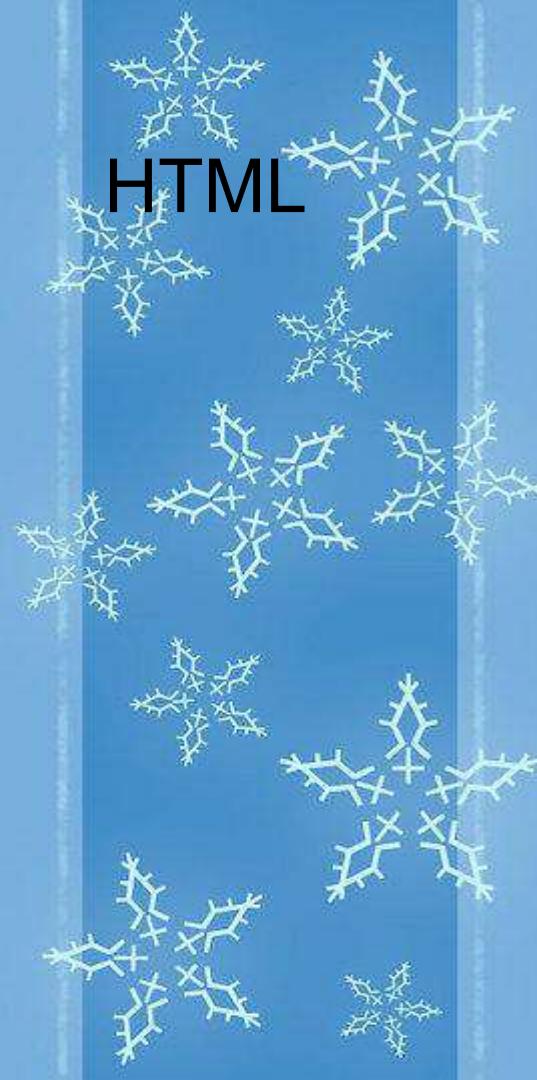
```
index.html x
1 <!DOCTYPE html>
2 <html lang="en" style="height: 100%; width: 100%; margin: 0; padding: 0;">
3   <head>
4     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5     <meta http-equiv="X-UA-Compatible" content="ie=edge" />
6     <title>Document</title>
7
8   </head>
9   <body>
10    <form action="" method="">
11      <input type="text" name="name" id="">
12      <br>
13      <button type="submit">submit</button>
14    </form>
15  </body>
16
17
18 </html>
```



The background of the slide features a blue gradient with numerous white snowflake icons of varying sizes scattered across it.

HTML

- Input
 - Password
 - Email
 - `<input type="password" placeholder="password">`
 - `<input type="email" value="aaa@gmail.com">`
 - `<input type="submit" value="submit here">`



HTML

Text area

- <textarea name="" id="" cols="20" rows="40">
</textarea>

Radio button

```
<input type="radio" name="gender" id="" value ="male"> male  
<input type="radio" name="gender" id="" value ="Female"> Female
```

A decorative background on the left side of the slide features a blue vertical bar with white snowflake patterns. The snowflakes are of various sizes and orientations, creating a winter-themed visual.

HTML

- Checkbox

- <input type="checkbox" name="gender" id="" value = "male"> male

HTML5 ADD-ONS

ELEMENTS

- <article> -- specifies it as an article
- <aside> --- keeps the text aside from header
- <mark> --- highlights the text that is present in mark tag.
- <nav> --- used for providing navigation property
- <meter> -- to measure data in a given range.
- Summary - details on dropdown

- ◉ <Time> --- just to specify it as time
- ◉ <Section> --- different sections in a file
- ◉ <wbr>----Word break
- ◉ <Date> -- date format (input type)
- ◉ Figure caption

TAGS AND OUTPUT

```
<body>
<section>
  <p>The lander and the rover will land near the lunar south pole region in a high plain between two craters, Manzinus C and Simpelius N, at a latitude of about 70° <mark> south on 7 September, 2019. The wheeled Pragyan rover will move on the lunar surface and will perform on-site chemical analysis for a period of 14 days (one lunar day). It can relay data to Earth through the Chandrayaan-2 orbiter and lander, which will fly on the same launch. The orbiter will perform its mission for one year in a circularized lunar polar orbit of 100 × 100 km. </p>
</section>

<section>
  <p>The lander and the rover will land near the lunar south pole region in a high plain between two craters, Manzinus C and Simpelius N, at a latitude of about 70° south on 7 September, 2019. The wheeled Pragyan rover will move on the lunar surface and will perform on-site chemical analysis for a period of 14 days (one lunar day). It can relay data to Earth through the Chandrayaan-2 orbiter and lander, which will fly on the same launch. The orbiter will perform its mission for one year in a circularized lunar polar orbit of 100 × 100 km. </p>
</section>
<nav>
```

```
<nav>
  <a href="https://en.wikipedia.org/wiki/Chandrayaan-2">this is all about CHANDRAYAAN2</a>
</nav>
<p> voting success </p><br>
<p>YES</p>
<meter value="15" min="0" max="20"></meter>
<p>NO</p>
<meter value="5" min="0" max="20"></meter>
<details>
  <summary>about the author.</summary>
  <p> This is written by ABC</p>
  <p>Thank you for visiting our page</p>
</details>
```

TAGS AND OUTPUT

The lander and the rover will land near the lunar south pole region in latitude of about 70° south on 7 September, 2019. The wheeled Prag will fly on the same launch. The orbiter will perform its mission for chemical analysis for a period of 14 days (one lunar day). It can relay

The lander and the rover will land near the lunar south pole region in latitude of about 70° south on 7 September, 2019. The wheeled Praghi will perform chemical analysis for a period of 14 days (one lunar day). It can relay data to the orbiter. The orbiter will fly on the same launch. The orbiter will perform its mission for at least one year.

this is all about CHANDRAYAAN2

voting success

YES



NO



► about the author.

▼ about the author.

This is written by ABC

Thank you for visiting our page

this is a word break contest so plesae trytrytrytrytry trytrytrytrytr
iiiiiiiiiiiiiiiiiiiiiiiiii



this is a picture

GRAPHICS

- <canvas> --- container for graphics, needs javascript to be embedded to draw elements.
-game application

- SVG --- Scaleable Vector Graphics
 - Circle, rectangle, rounded rectangle, polygon
- SVG- 2D ,predefined, event handling

TAGS AND OUTPUT

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title></title>
5 </head>
6 <body>
7 <svg width="200" height="300">
8     <circle cx="60" cy="60" r="40" />
9 </svg>
10 <svg width="400" height="100">
11     <rect width="400" height="100" />
12 </svg>
13 </body>
14 </html>
```



MULTIMEDIA

- Video : before HTML5 we need plug-in
 - It specifies a standard way to embedded video in a webpage.
 - <video width=" " height=" " controls>
 - <source src=" " type=" ">
 - </video>
 - Controls- play, pause, volume.

MULTIMEDIA

- Audio-adding audio files to the page.
- <audio controls>
- <source src=“ “ type=“ “>
- </audio>

MULTIMEDIA

- ◉ ADD YOUTUBE VIDEO.

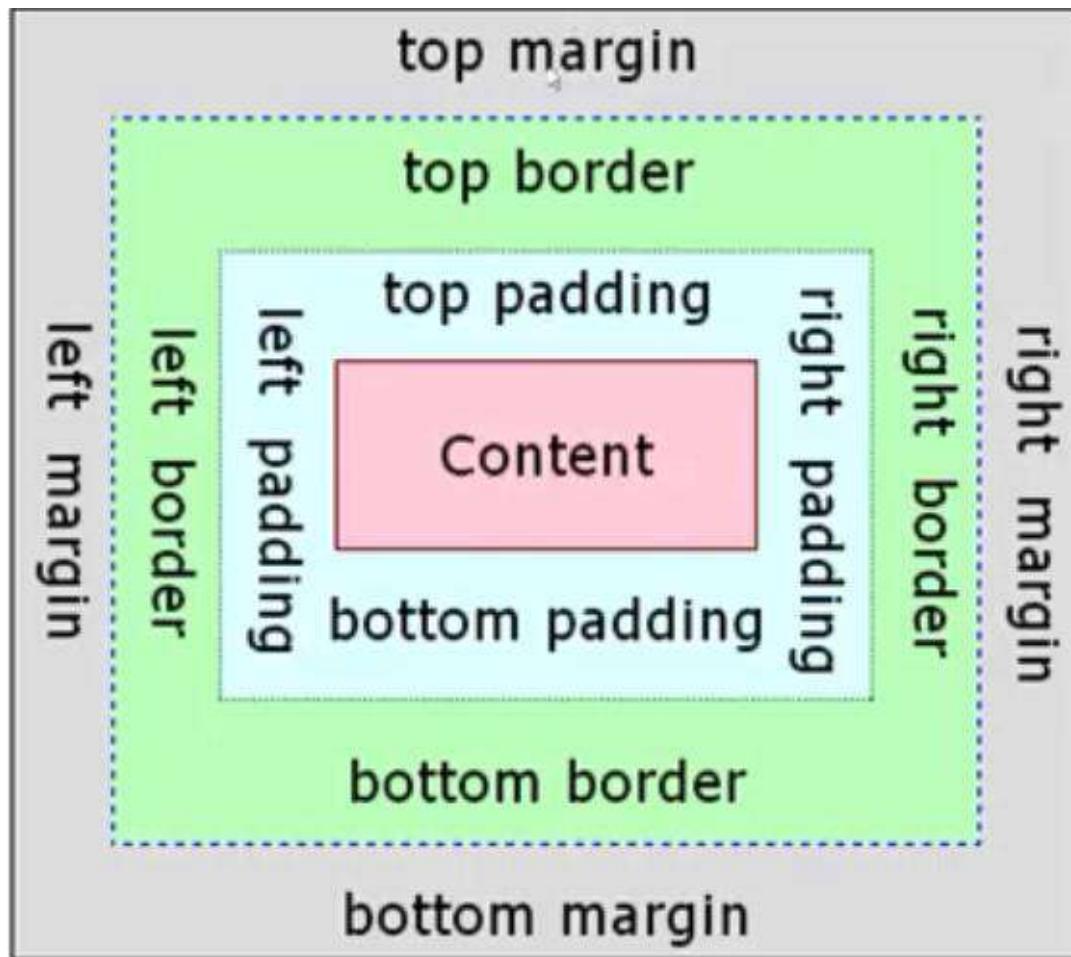
TAGS AND OUTPUT

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5  </head>
6  <body>
7  <video width="320" height="240" controls/autoplay>
8      <source src="filename.mp4" type="video/mp4">
9  </video>
10 </body>
11 </html>
```

CSS ADD-ONS

CSS

◎ BOX Model



OUTLINE

- Outline style-- must
- Outline color
- Outline width
- Outline offset- space to margin and outline

PSEUDO CLASS

- ◎ When mouse is moved over change the style of the content.
- ◎ :hover
- ◎ :focus

TAGS AND OUTPUT

```
71  
72 div.cl{position: absolute;  
73     clip:rect(0px,50px,50px,0p  
74  
75  
76 a:link {color: blue;}  
77 a:visited {color: green;}  
78 a:hover {color: hotpink;}  
79 a:activated {color: red;}  
80  
81 .li  
82 { list-style-image: url(1.jpg); }  
--
```

and Windows



```
textarea:focus {  
    background: pink;  
}
```

PSEUDO ELEMENT

- Standard change of few elements in a particular way.
- We can use elements in such cases.

TAGS AND OUTPUT

```
79   background-color: yellow;
80
81 p::first-line {
82   color: red;
83   font-variant: small-caps;
84 }
85 .li
86 { list-style-image: url(1.jpg); }
```

BEFORE ,AFTER SELECTOR

- ::after is a pseudo element which allows you to insert content onto a page from CSS.
- ::before is exactly the same only it inserts the content before any other content in the HTML instead of after.
- Eg: p ::after p ::before

```
{  
}
```

```
{  
}
```

OPACITY

- Making the image dull or making it more transparent.
- When applying for body which has images in background the text also turns opaque how to solve it?
- Opacity 0.1;

OTHER STYLES

- ◉ Rounded corner-- border-radius: 15px;
- ◉ Images in border- design

```
{  
    border: 10px solid transparent;  
    border-image: url(border.jpg) 30  
    round;}
```

OTHER STYLES

- Shadows in box
 - box shadow: 2px 2px 2px red;
- Shadows in text
 - Text shadow : 2px 2px 2px red;
- Multiple shadow- separated by ,



Text

TRANSFORMATIONS

◎ 2D transformation

- transform : rotate(30 deg);
- transform :translate(x,y);
- Scale --- x,y scale
- Skew- x,y

- 3D transformations???

TRANSITION

- Transition - State change smoothly
- transition-delay
- transition-duration - must
- transition-property
- transition-timing-function

ANIMATIONS

- Animation-changing from one style to another.
- Animation name
- Animation duration
- Animation delay
- Animation direction
- Animation iteration count

ANIMATION

```
.main_banner
{
background-color: grey;
animation-name:example;
animation-duration: 4s;
}

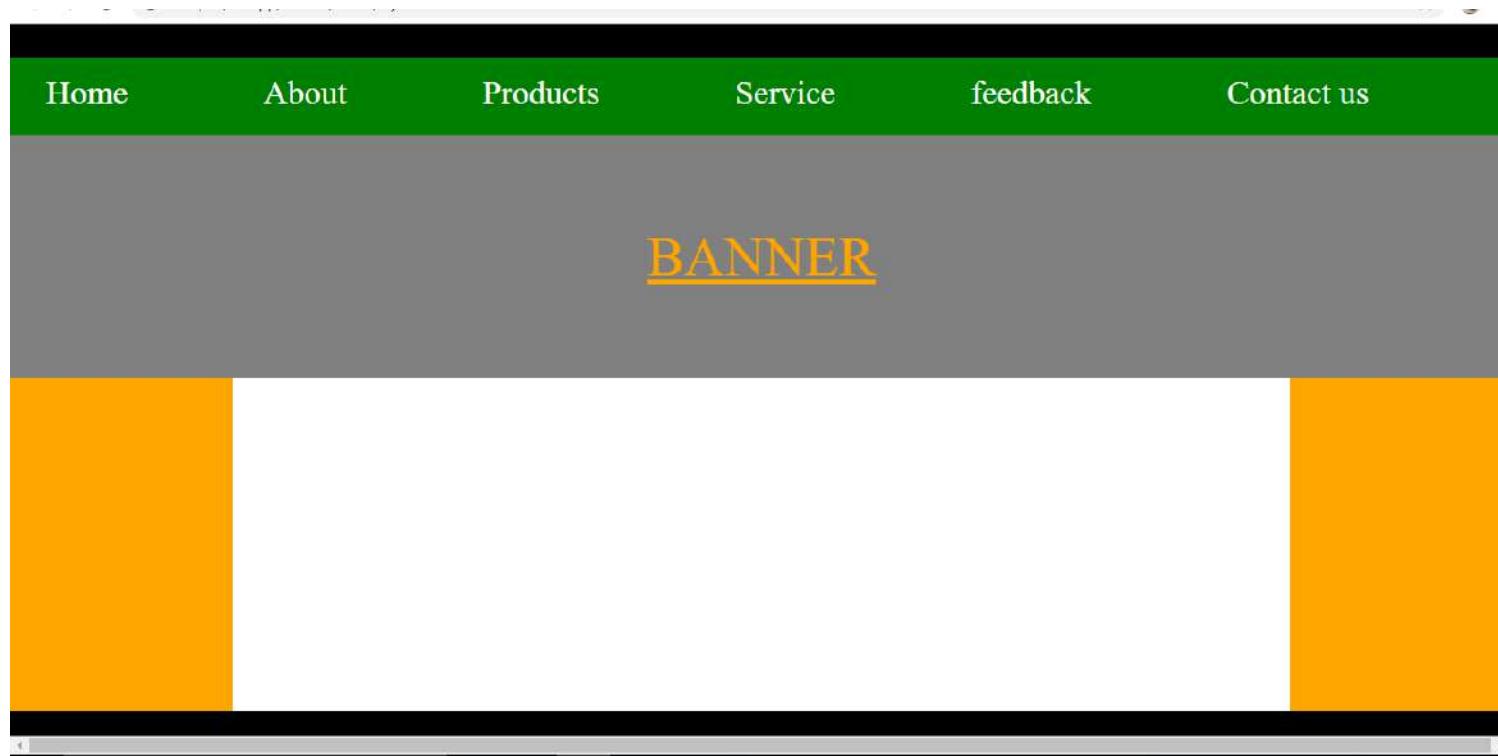
@keyframes example
{
from{background-color: grey;}
to{background-color: red;}
}
```

EXPLORE

- ◉ Pagination
- ◉ Button styling
- ◉ Resizing
- ◉ Column
- ◉ Tooltip

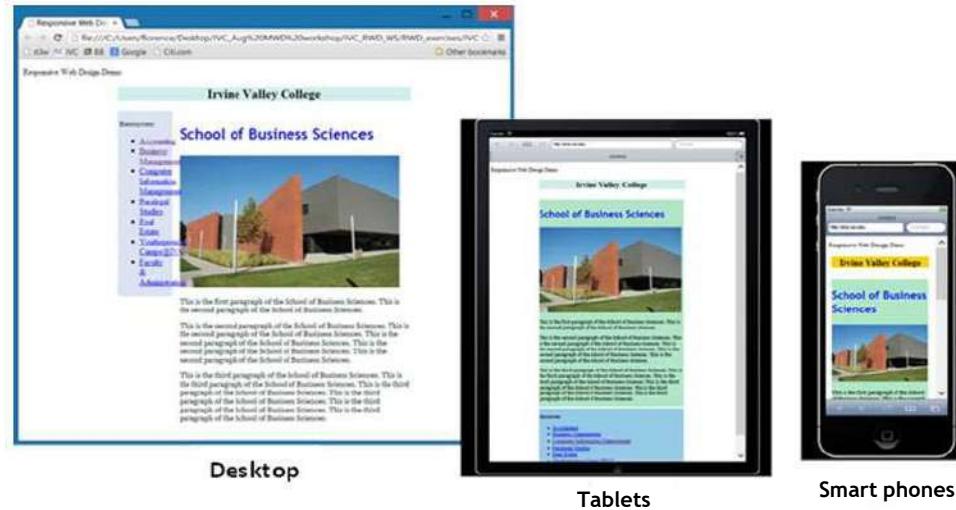
WEB LAYOUT

WEB LAYOUT



RESPONSIVE WEB DESIGN

WHAT IS RESPONSIVE WEB DESIGN (RWD)?



Desktop

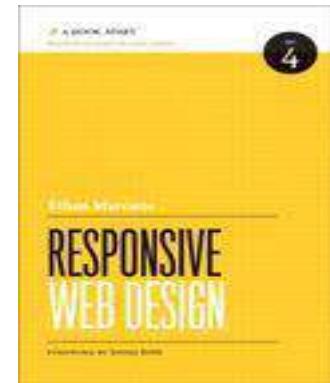
Tablets

Smart phones

- A design when the layout and content adapts to the user's devices: **screen size, platform and orientation**
- The design and development should respond to the user's behavior
- The website that have the technology to automatically respond to the user's preferences.

HISTORY OF THE RESPONSIVE WEB DESIGN

- The term Responsive Web Design was first coined by **Ethan Marcotte** in his article *A List Apart* in May 2010
<http://alistapart.com/article/responsive-web-design>
- He defined the technique of RWD by using **fluid grids, flexible images, and media queries** to deliver different visual experiences for different screen sizes.
- Ethan expanded his RWD theory and published his book titled *Responsive Web Design*.



WHY SHOULD WE BUILD A RESPONSIVE WEB?

- Each year new devices are pouring into the market, responsive web design let us build one site, and modify it to adapt the new device's screen size
- Build once for all devices
- Easy to manage content editing through a single CMS (Content Management System)
- If we have a working website:
 - Not need to rebuild new websites to adapt the new devices
 - We can convert the existing working website to a responsive Web site to adapt all kind of devices

PARADIGM SHIFT TOWARDS MOBILE

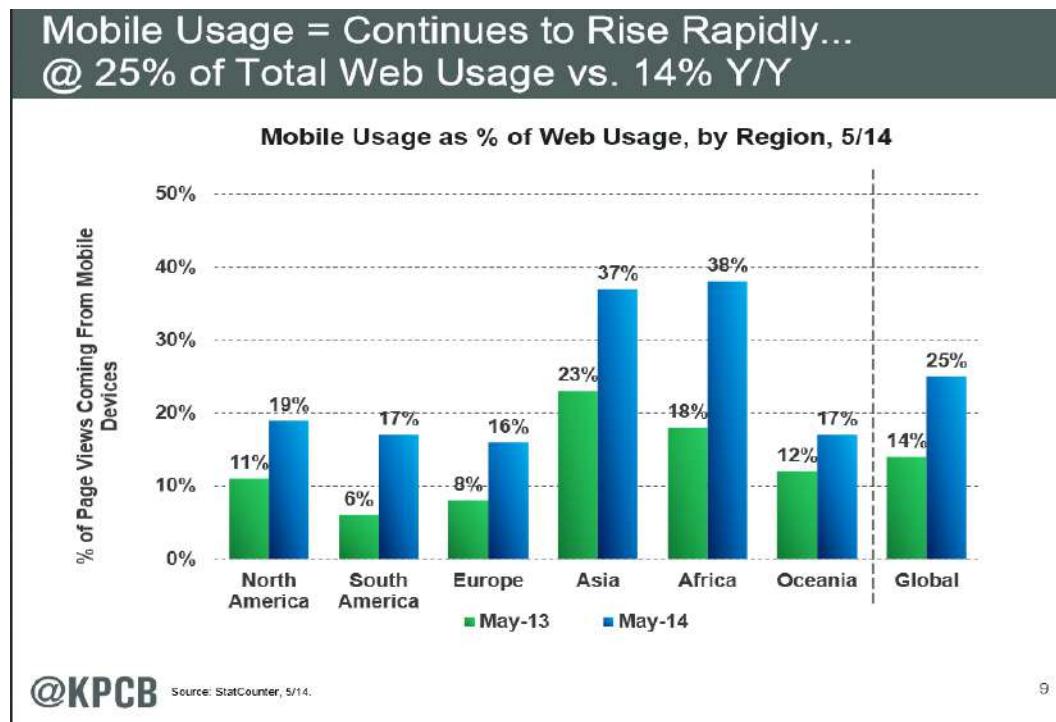
- International Data Corporation predicted that by the end of 2013, tablet sales will exceed that of portable PCs.

Mobiles Vs. Computers: 2007-2015
Global internet user projection research by Morgan Stanley



PARADIGM SHIFT TOWARDS MOBILE (CONTINUED)

- Mary Meeker in her 2012 *internet Trends Report* notes mobile makes up 15% of Web traffic, up from 10% a year ago
- Her recent report showed the following chart



ADVANTAGES OF RWD

- One single HTML document to be maintained
- One single CSS file to be maintained
- The site is easily accessible on any type of device.
- Better user experience.
Users will have a similar experience using the site when they access the site from different devices.
- Responsive Web is flexible and adaptable
- Maintaining a RWD is:
 - Easier than maintaining several website for different devices.

FUNDAMENTAL TECHNIQUES FOR RWD

- There are three parts in Responsive Web design:
 1. **Flexible, grid-based layouts**
The web sites are built using percentage for the widths
 2. **Media queries**
Use a module from the CSS3 specification
 3. **Flexible media & images**
When screen size begins to change, the media/images need to be flexible to suit the screen size

TECHNIQUES FOR RWD: FLEXIBLE, GRID-BASED, LAYOUT

- Idea behind liquid layout: it's more carefully designed in terms of proportion → use percentage
- Proportion of each page element is the target element divided by the context

Example:

- suppose your desktop layout has the main wrapper with the width of 960px and
- suppose that the target element is 300px wide
- then the proportion would be 31.25%

$$300\text{px} / 960\text{px} = 31.25\%$$

TECHNIQUES FOR RWD: MEDIA QUERIES

- Media queries is the backbone of RWD
- Media queries provide the ability to
 - Specify different styles for individual browser device circumstances
 - Specify the width of the viewport or device orientation
- Using Media queries in the CSS file to change the styling of the HTML elements is based on certain breakpoints.

TECHNIQUES FOR RWD: FLEXIBLE MEDIA & IMAGES

- Using media queries, designers are able to:
 - Extend the media declarations to include various media properties, based on device being used. Such as:
 - screen size, orientation, and color
 - write a rule that prevents images from exceeding the width of their container

THE VIEWPORT META TAG

➤ Viewport meta tag:

- Tells the browser how to behave when rendering the page - you tell the browser how big the viewport will be
- Use the viewport meta tag in the <head> section
- If we are using RWD, it's good to have the meta tag viewport as

```
<meta name="viewport"  
      content="width=device-width,  
      initial-scale=1">
```

No zooming

Adapt to the width of the device

CODING META VIEWPORT TAGS

- There are two ways to add the viewport tag for overriding the default viewport by user agent.

1. Use the @viewport CSS rule.

- This is still relatively new and mostly unsupported for now.

```
/* CSS Document */  
@viewport {width: 480px; zoom: 1;}
```

2. Use the viewport meta tag

- This is almost supported universally.

```
<meta name="viewport"  
content="width=device-width, initial-scale=1">
```

CODING META VIEWPORT TAGS (CONTINUED)

```
<meta name="viewport"  
      content="width=device-width,  
      initial-scale=1">
```

- width=device-width:
 - The page adapts to the device's width
 - Syncs with the device's width
- initial-scale=1:
 - Make the initial scale at 100%
 - When the viewport is larger than the screen width, the scale factor will shrink down to fit the width within the viewport.

CODING MEDIA QUERIES

- The following code will display the font-size at 100% if the width is at least 1024 px

```
@media screen and (min-width: 1024px) {  
  body {font-size: 100%;}  
}
```

- The following code tests the orientation and the device-width

```
@media screen and (min-device-width: 480px) and  
(orientation: landscape) {  
  body { font-size: 100%; }  
}
```

- The logical operators are pretty interchangeable:
 - The operator “and” can be replaced with “not”. The orientation “portrait” with “landscape”.

CODING MEDIA QUERIES (CONTINUED)

- The following code renders a page that the body background color will change to blue only between 500px and 700px.

```
@media screen (min-width:500px) and (Max-width:700px) {  
    body {background: blue; }  
}
```

- The following code displays an orange background color when a device hits 1024px width and changes to yellow when the display of a device drop into mobile territory.

```
@media (max-width: 1024px) {  
    body { background: orange; }  
}  
@media (max-width: 768px) {  
    body {background: yellow; }  
}
```

DEFINITIONS

- Width = width of the display area
- Device-width = width of device
- Orientation = orientation of the device
- Aspect-ratio = ratio of width to height
 - It is expressed by two numbers separated by a slash
- Device-aspect-ratio = ratio of device-width to device-height
- Resolution - density of pixels of output device (dpi)

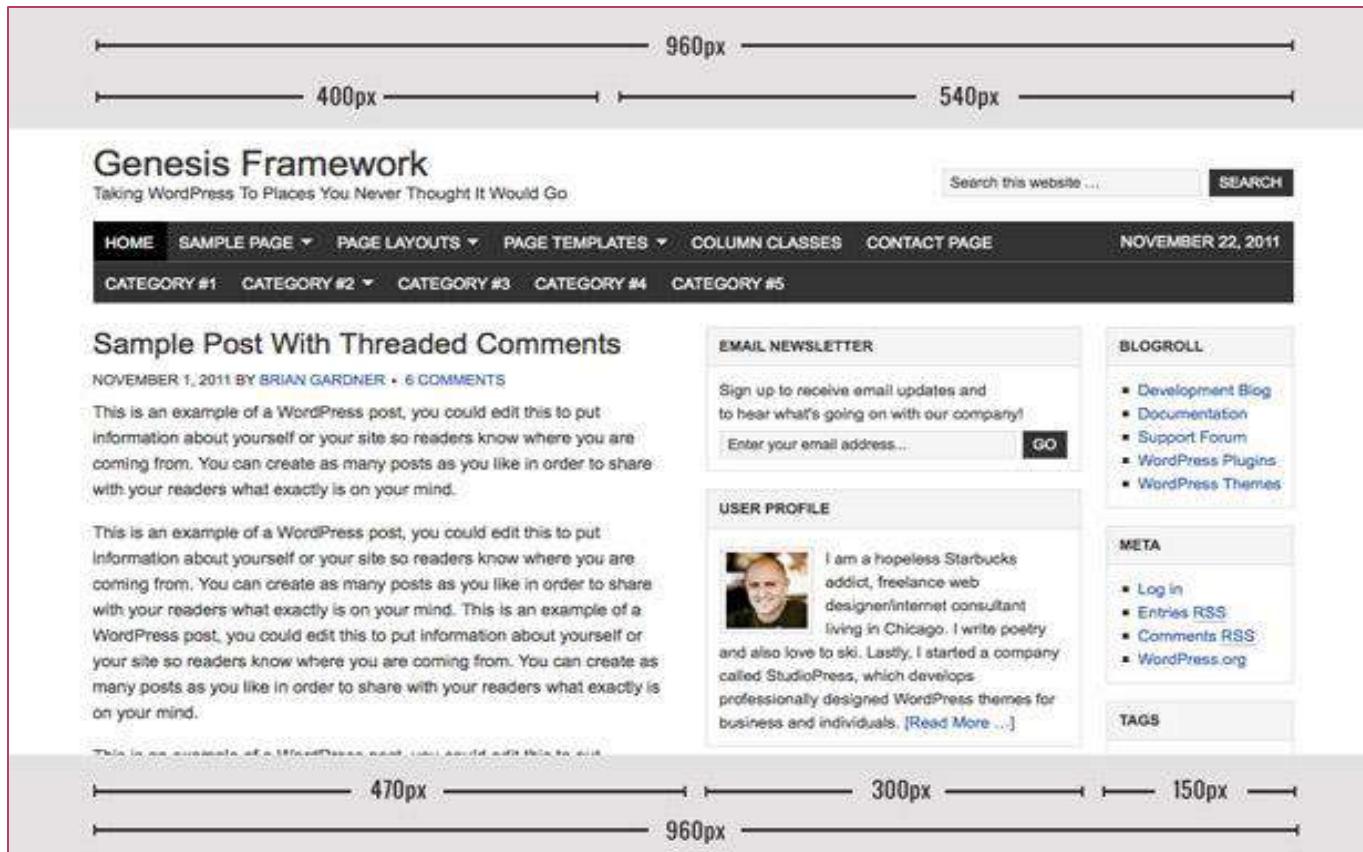
MEDIA QUERIES TOGETHER WITH VIEWPORT

- It is not a good idea to use the media queries without a meta viewport tag
- Some mobile browsers have a default layout viewport of around 850 to 1000 pixels
- The page will be much larger than the device width



CONVERTING AN EXISTING PAGE TO RWD

- Let's say the existing page has the following layout



CONVERTING AN EXISTING PAGE TO RWD (CONTINUED)

- Assume the existing page has the following basic structure of HTML code

```
<div id="wrap">
    <div id="header">
        <div id="title-area"></div>
        <div class="widget-area"></div>
    </div>
    <div id="inner">
        <div id="content-sidebar-wrap">
            <div id="content"></div>
            <div id="sidebar"></div>
        </div>
        <div id="sidebar-alt"></div>
    </div>
</div>
```

Converting an Existing page to RWD (continued)

- Assume the existing page has the following basic structure of CSS code

```
#wrap {width: 960px; }  
  
#header {width: 960px; }  
  
#title-area {width: 400px; }  
  
#header .widget-area {width: 540px; }  
  
#inner {width: 960px; }  
  
#content-sidebar-wrap {width: 790px; }  
  
#content {width: 470px; }  
  
#sidebar {width: 300px; }  
  
#sidebar-alt {width: 150px; }
```

Converting an existing page to RWD (continued)

➤ SUPPOSE THE TARGET GOAL IS 960PX WIDE

```
#wrap {width: 100%; }  
#header {width: 100%; }  
#title-area {width: 41.666667%; }  
#header .widget-area {width: 56.25%; }  
#inner {width: 100%; }  
#content-sidebar-wrap {width: 82.291667%; }  
#content {width: 48.958333%; }  
#sidebar {width: 31.25%; }  
#sidebar-alt {width: 15.625%; }
```

Formula:

(original pixels/target goal pixels)* 100%

Example for the #title-area:

$$(400\text{px}/960\text{px}) * 100\% = 41.666667\%$$

Converting an existing page to RWD (continued)

➤ The ul in the sidebar

```
/*The pixel for the margin is 25px */
.widget-area ul {
    margin: 10px 0 0 25px; }

/*the percentage conversion of the
target margin*/
.widget-area ul {
    margin: 10px 0 0 16.666667%; }
```

This pixel is 150 because that is the width of the sidebar.

$$(25/150) * 100\% = 16.666667\%;$$

➤ Flexible images

- `img { max-width: 100%; }`

CONVERTING EXERCISE: DO NOT ROUND UP!

Do not round up, keep the long decimal points

- Because each browser rounds the percentage differently, if you round the percentage, you need to tweak each section

CONVERTING EXERCISE, INSERTING MEDIA QUERIES

- Add two media query break points at the end of the style section

Note: The two media queries are provided for you at the right.

```
@media screen and (max-width:830px) {  
    #content {  
        float: left;  
        width: 98%;  
        margin-top:5px;  
    }  
    nav li;  
    nav a {  
        display:block;  
    }  
}  
  
@media screen and (max-width:480px) {  
    #content {  
        float: none;  
        width:95%;  
    }  
}
```

TESTING THE RESPONSIVE DESIGN

- Test with the new media queries to see whether or not they're hitting the right breakpoints.
 - Resize the browser window to see the changes
 - This is helpful and gives immediate feed back, however:
 - The feed back is not really the actual trigger points
 - It does not show how the site will render
 - It overlooks the performance

TESTING THE RESPONSIVE DESIGN (CONTINUED)

- Use online simulator testing tools
 - There are many free online testing tools to help test more precisely and to speed up the process.
- Using online mobile emulators:
programs that simulate a specific mobile device, browser, or operating system
- Test on actual devices, best way, but it is expensive to have all the devices on hand and to purchase more new ones.

ONLINE SIMULATOR TESTING TOOLS

- Benjamin Keen Bookmarklet
 - <http://www.benjaminkeen.com/open-source-projects/smaller-projects/responsive-design-bookmarklet/>
- The following online simulator allows you to just enter the URL
 - Responsivepx by Remy Sharp: users have control of the precise width
<http://responsivepx.com/>
 - Responsive.is: it provides icon for difference devices:
<http://www.headlondon.com/>
 - Mobiltest: user can chose the devices, also provides the average load time
<http://mobitest.akamai.com/m/index.cgi>

ONLINE EMULATOR TESTING TOOLS

- TestiPhone.com
- Opera's Mini simulator
- Download and install emulators:
 - Opera's Mobile emulator
 - Apple SDK, the emulators comes with Apple's iOS
 - Android SDK, the emulators comes with Android OS.

DEBUGGING TOOLS

- Tools for debugging when the behavoir is not expected after testing
 - Opera's Remote Debugger
 - Dragongly: Debug on the desktop with the site on a mobile device
 - WebKit remote debugging
 - Weinre
 - Web Inspector

DOM-JAVASCRIPT

DOM

“It is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.”

DOCUMENT OBJECT MODEL - DOM

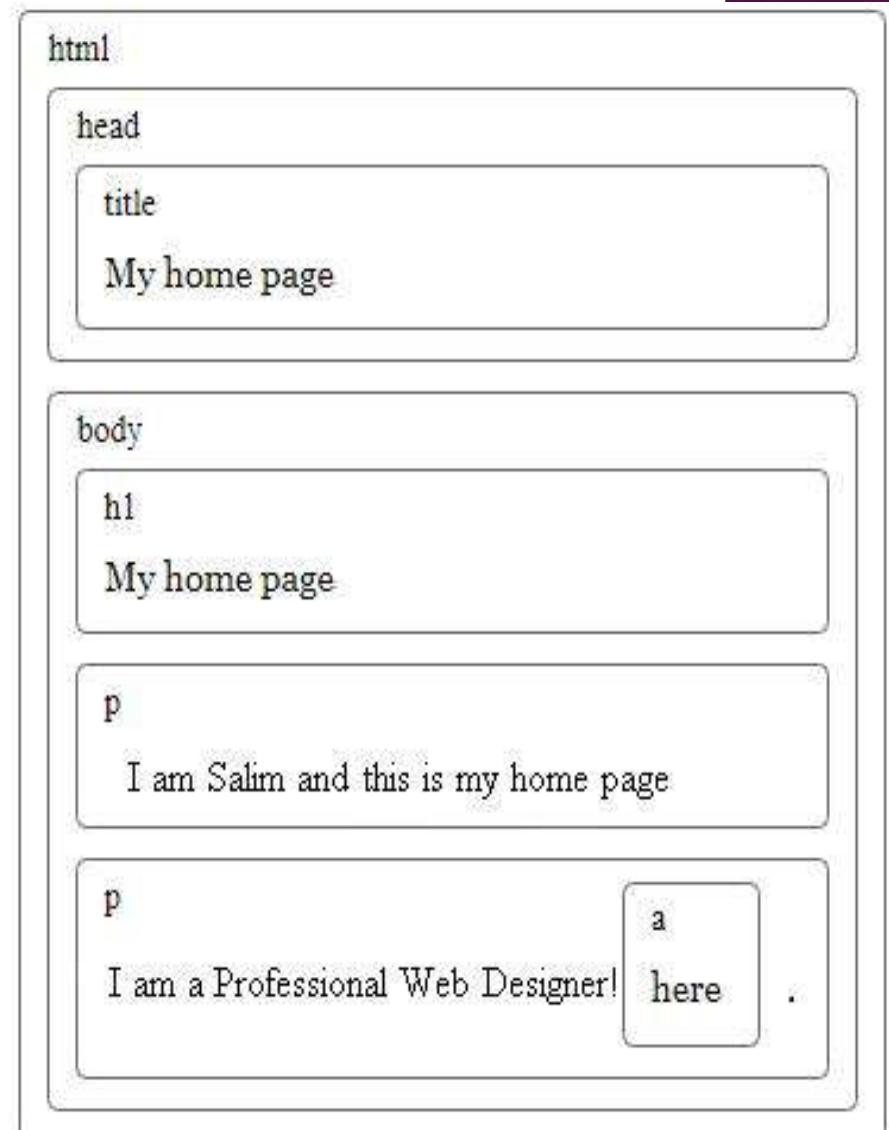
- DOM -is an essential part of making websites interactive.
- DOM is an interface that allows a programming language to manipulate the content, structure, and style of a website.
- JavaScript is the client-side scripting language that connects to the DOM in an internet browser.

DOCUMENT OBJECT MODEL - DOM

- A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects, which allow access to modification of document content.
- In short, The way a document content is accessed and modified is called the *Document Object Model*, or *DOM*

IMAGINE AN HTML DOCUMENT AS A NESTED SET OF BOXES.

```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am ABC and this is
my home page.</p>
    <p>I am a Professional Web
Designer! Click here to go to my
page
      <a
        href="http://google.com">here</a>.</p>
    </body>
  </html>
```



HOW DOM WORKS ??

- When you open a web page in your browser, the browser retrieves the page's HTML text and parses it, browser builds up a model of the document's structure and uses this model to draw the page on the screen.
- This representation of the document is one of the toys that a JavaScript program has available in its sandbox.

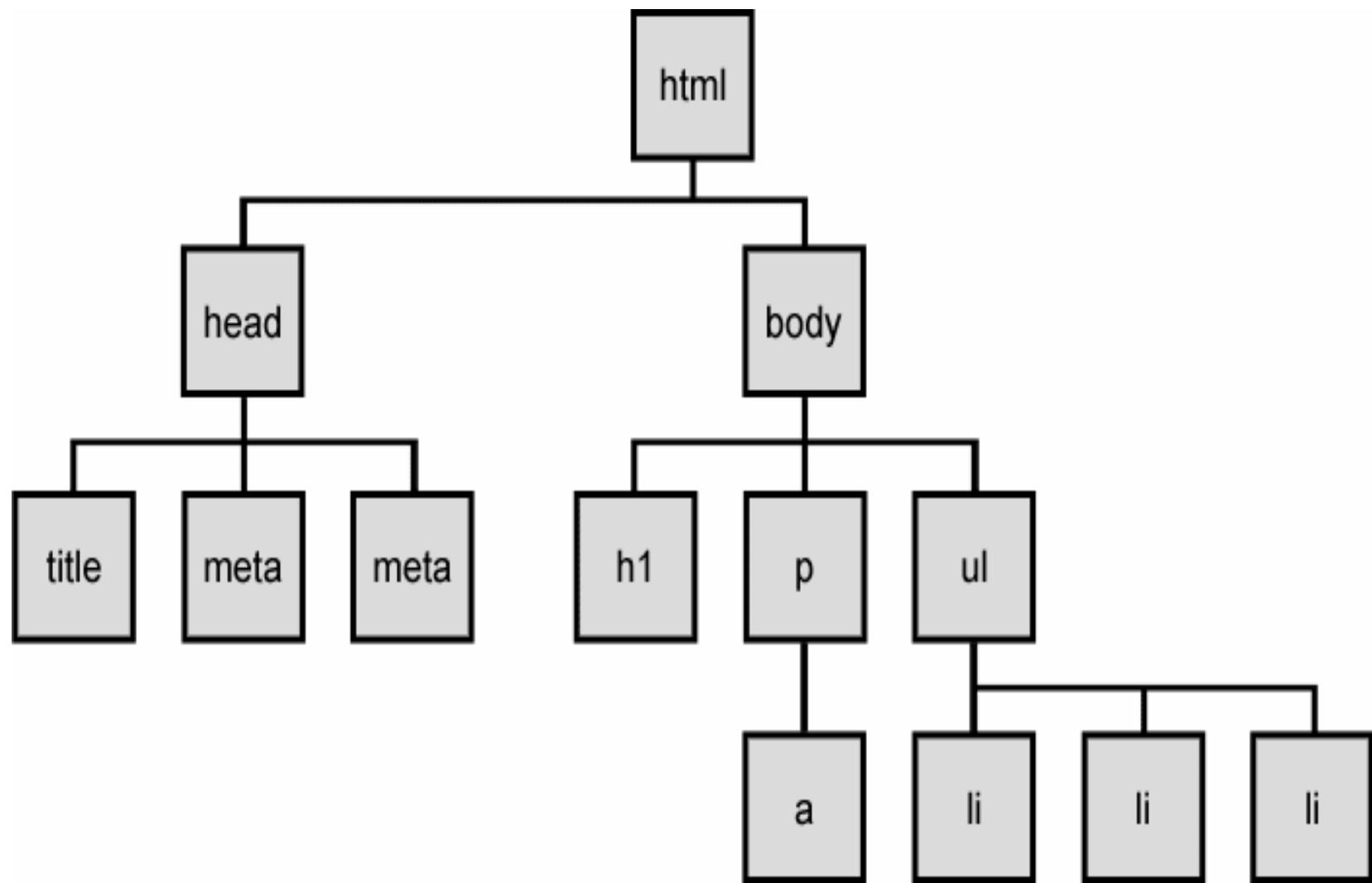
HOW DOM WORKS ??

- It is a data structure that you can read or modify. It acts as a *live* data structure: when it's modified, the page on the screen is updated to reflect the changes.

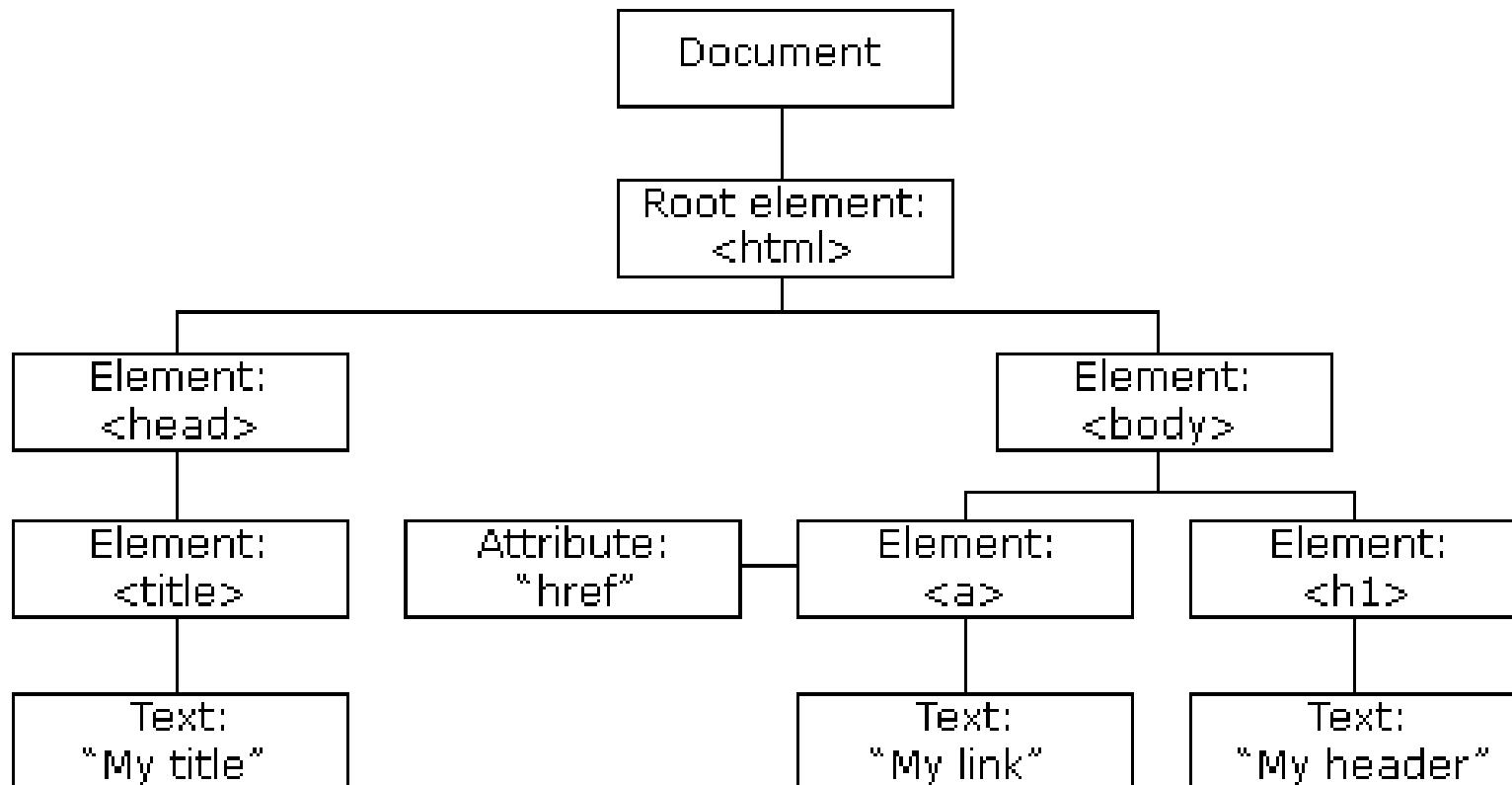
DOM = TREE



DOM TREE



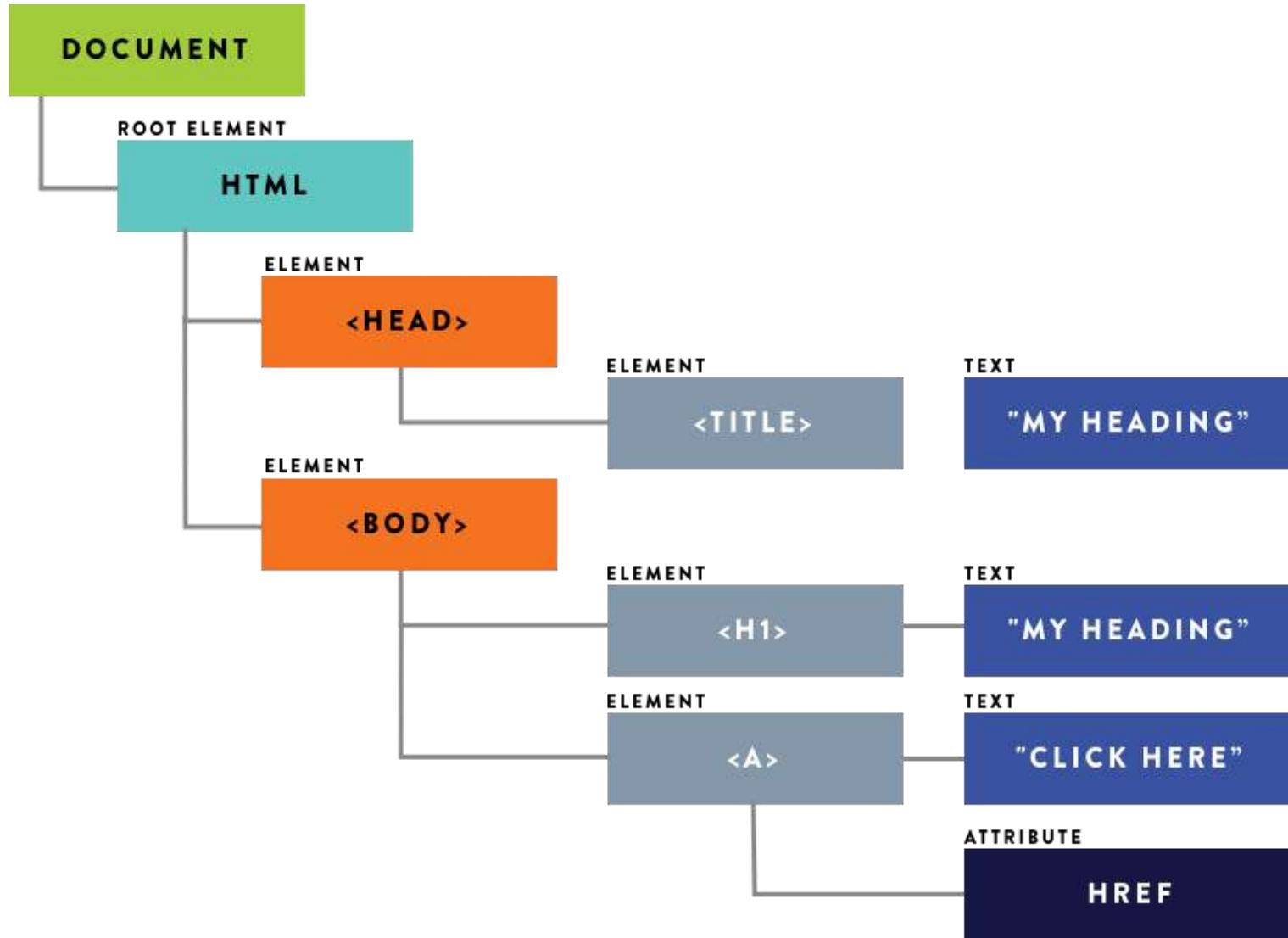
HTML DOM TREE OF OBJECTS



DOM - PARTS

- **Core DOM**
- **XML DOM**
- **HTML DOM**

ELEMENTS ACCESS IN JAVASCRIPT



ELEMENTS ACCESS IN JAVASCRIPT

We can access the desired element by using from the web document using javascript method

- **getElementById**
- **getElementByTagname**
- **getElementByClassname**
- **CSS Selector- Query selector**
`document.querySelector("tag.classname");`
- **HTML Object collection-Array**

```
Var Dom_obj=document.getElementsById("myinput")
```

HTML DOM METHODS - [METHOD / PROPERTY]

- ⦿ **getElementById** ----->Method

Most common way to access an HTML element is to use the id of the element.

- ⦿ **innerHTML** -----> Property

1. InnerHTML property is used for getting or replacing the content of HTML Elements
2. InnerHTML property can be used to get or change any HTML element, including <html> and <body>

```
document.getElementById("demo").innerHTML=  
“welcome”;
```

EVENTS

**“Event is an activity that represents a change
in the environment”**

EVENT HANDLING

- **Event Handler** is a script that gets executed in response to these events.
- This event Handler enables the web document to respond the user activities through browser window.
- This special type of programming in which events may occur and these events get responded by executing the event handlers.

Programming - “ Event-driven programming”

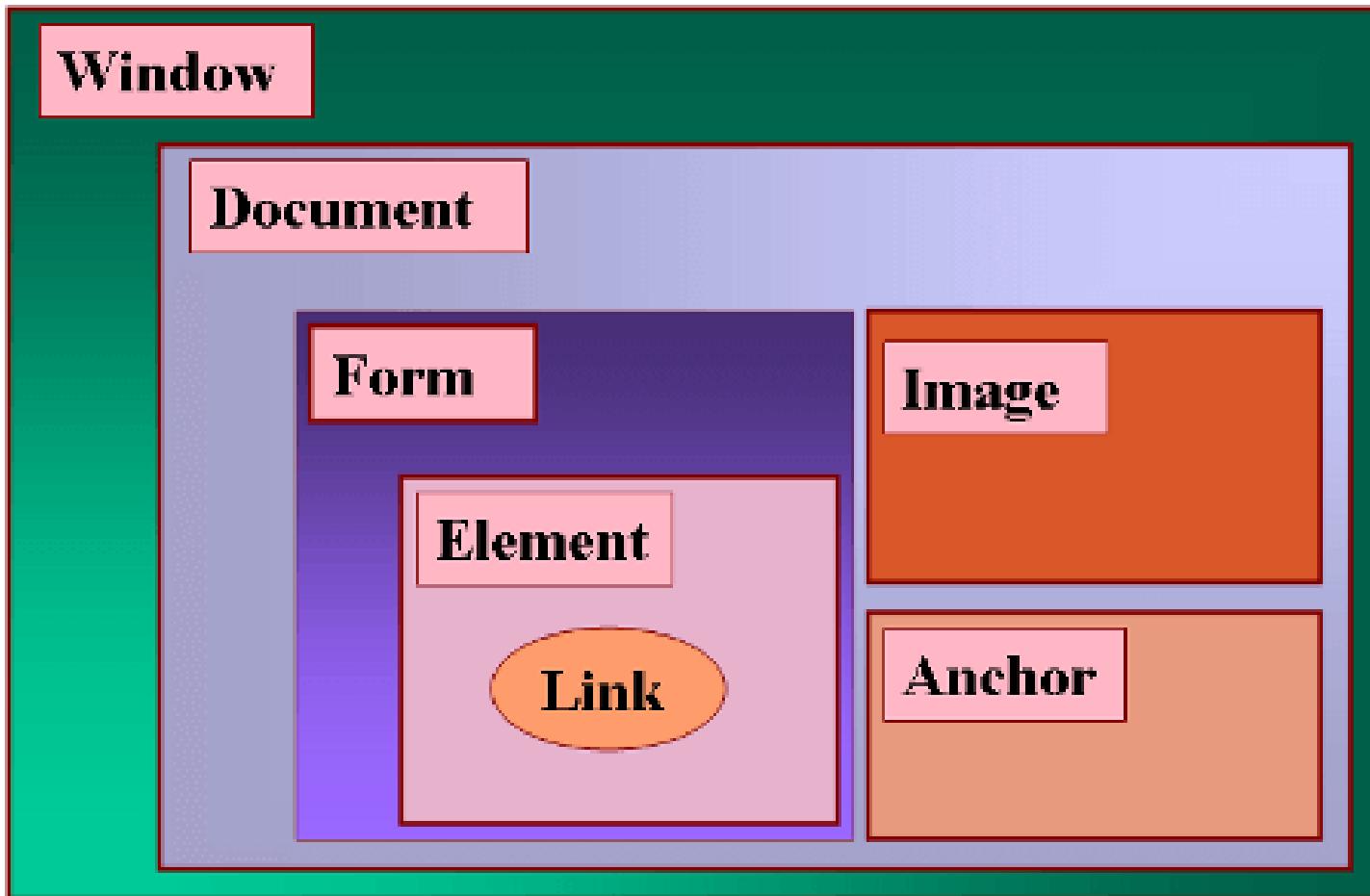
EVENTS

| EVENTS | INTRINSIC EVENT ATTRIBUTE | MEANING |
|-----------|---------------------------------|--|
| blur | onblur | Losing the focus |
| change | onchange | On occurrence of some change |
| click | onclick | When user click the mouse button |
| dblclick | ondblclick | When user double click the mouse button |
| focus | onfocus | When user requires input focus |
| keyup | onkeyup | When user releases the key from the keyboard |
| keydown | onkeydown | When user presses the key down |
| mouseover | onmouseover | When user moves the mouse away over some element |

EVENTS

| EVENTS | INTRINSIC EVENT ATTRIBUTE | MEANING |
|-----------|---------------------------------|--|
| keypress | onkeypress | When user presses the key |
| mousedown | onmousedown | When user clicks the left mouse button |
| mouseup | onmouseup | When user releases the left mouse button |
| mouseout | onmouseout | When user moves the mouse away from some element |
| load | onload | After getting the document loaded |
| reset | onreset | When the reset button is clicked |
| Submit | onsubmit | When the submit button is clicked |
| Select | onselect | On selection |
| unload | onunload | When user exits the document |

WINDOW OBJECTS



WINDOW OBJECTS

Methods

- **alert(message)**
- **prompt(message, default_text)**
- **confirm(message)**
- **close()**
- **open(url, window name)**
- **Set TimeOut()**

WINDOW OBJECTS

Properties

- **closed**
- **defaultstatus**
- **status**
- **frame**
- **location**
- **Name**
- **Self**
- **parent**

THE JAVASCRIPT LANGUAGE

WHAT IS JAVASCRIPT?

- HTML - Content Layer
- CSS -Presentation layer
- JS -Interactive Layer

HOW JS INTERACTS WITH HTML?

- Document model-Everything is represented as objects.
- JS can easily interpret those objects.

LANGUAGE ELEMENTS

- Variables
- Literals
- Operators
- Control Structures
- Functions
- Objects

JAVASCRIPT VARIABLES

- Untyped- any data type.
- Can be declared with var keyword:

```
var name;
```

- Can be created automatically by assigning a value:

```
val=1;      call="Hi John";
```

VARIABLES (CONT.)

- Using **var** to declare a variable results in a *local* variable (inside a function).
- If you don't use **var** - the variable is a global variable.

LITERALS

- The typical bunch:

- Numbers 17 123.45
- Strings "Hello Dave"
- Boolean: true false
- Arrays: [1, "Hi John", 17.234]



Arrays can hold anything!

ARRAYS

- Arrays are actually Javascript Objects.
- The only thing special in the language to support arrays is the syntax for literal.

OPERATORS

- Arithmetic, comparison, assignment, bitwise, boolean (pretty much just like C++).

+ - * / % ++ -- == != > <
&& || ! & | << >>

DIFFERENT THAN C++

- The + operator is used for addition (if both operands are numbers)
-or-
- The + operator means string concatenation (if either one of the operands is not a number)

CONTROL STRUCTURES

- Again - pretty much just like C:

if if-else ?: switch

for while do-while

- And a few not in C

for (var in object)

with (object)

JAVASCRIPT FUNCTIONS

- The keyword **function** is used to define a function (subroutine):

```
function add(x,y) {  
    return (x+y);  
}
```

- No type is specified for arguments!

JAVASCRIPT PROGRAM TO MAKE SURE

```
<SCRIPT>
function add(x,y) {
    return (x+y);
}

document.write("add(3,4) is " + add(3,4) + "<BR>");
document.write("add(\"3\", \"4\") is " + add("3","4") +
    "<BR>");

document.write("add(\"Hi\", \"Dave\") is " +
    add("Hi","Dave") + "<BR>");

document.write("add(3, \"Hi\") is " + add(3,"Hi") +
    "<BR>");

document.write("add(\"2.13blah\", 3.14) is " +
    add("2.13blah", 3.14));
</SCRIPT>
```

RECURSION IS SUPPORTED

```
function factorial(x) {  
    // use <= 0 instead of < 0  
    // to avoid problems with neg numbers  
  
    if (x<=0)  
        return(1);  
    else  
        return( x * factorial(x-1));  
}  
  
document.write("<H3>11! = " +  
factorial(11) + "</H3>");
```

OBJECTS

- Objects have attributes and methods.
- Many pre-defined objects and object types.
- Using objects follows the syntax of C++ / Java:

`objectname.attributename`

`objectname.methodname()`

THE DOCUMENT OBJECT

- Many attributes of the current document are available via the **document** object:

Title Referrer

URL Images

Forms Links

Colors

DOCUMENT METHODS

- `document.write()` like a print statement - the output goes into the HTML document.
- `document.writeln()` adds a newline after printing.

```
document.write("My title is" +  
document.title);
```

EXAMPLE

```
<HEAD>
<TITLE>JavaScript is Javalicious</TITLE>
</HEAD>
<BODY>
<H3>I am a web page and here is my
name:</H3>
<SCRIPT>
document.write(document.title) ;
</SCRIPT>
<HR>
```

THE NAVIGATOR OBJECT

- Represents the browser. Read-only!
- Attributes include:

`appName`

`appVersion` ←

`platform`

often used to determine
what kind of browser is
being used
(Netscape vs. IE)

NAVIGATOR EXAMPLE

```
if (navigator.appName ==  
    "Microsoft Internet Explorer") {  
    document.writeln("<H1>This page  
requires Netscape!</H1>");  
}
```

THE WINDOW OBJECT

- Represents the current window.
- There are possibly many objects of type **Window**, the predefined object **window** represents the current window.
- Access to, and control of, a number of properties including position and size.

WINDOW ATTRIBUTES

- **document**
- **name**
- **status** the status line
- **parent**

SOME WINDOW METHODS

`alert()`

`close()`

`prompt()`

`moveTo()` `moveBy()`

`open()`

`scroll()` `scrollTo()`

`resizeBy()` `resizeTo()`

THE MATH OBJECT

- Access to mathematical functions and constants.
- Constants: **Math.PI**
- Methods:

`Math.abs()` , `Math.sin()` , `Math.log()` ,
`Math.max()` , `Math.pow()` , `Math.random()` ,
`Math.sqrt()` , ...

MATH OBJECT IN USE

```
// returns an integer between 1 and 6
function roll() {
    var x = Math.random();

    // convert to range [0,6.0)
    x = x * 6;

    // add 1 and convert to int
    return parseInt(1+x);
}

document.writeln("Roll is " + roll() );
```

ARRAY OBJECTS

- Arrays are supported as objects.
- Attribute `length`
- Methods include:
`concat join pop push reverse sort`

SOME SIMILARITY TO C++

- Array indexes start at 0.
- Syntax for accessing an element is the same:

```
a[3]++;
```

```
blah[i] = i*72;
```

NEW STUFF (DIFFERENT THAN C++)

- Arrays can grow dynamically - just add new elements at the end.
- Arrays can have *holes*, elements that have no value.
- Array elements can be anything
 - numbers, strings, or arrays!

CREATING ARRAY OBJECTS

- With the `new` operator and a size:

```
var x = new Array(10);
```

- With the `new` operator and an initial set of element values:

```
var y = new Array(18,"hi",22);
```

- Assignment of an *array literal*

```
var x = [1,0,2];
```

ARRAYS AND LOOPS

```
var a = new Array(4);  
  
for (i=0;i<a.length;i++) {  
    a[i]=i;  
}  
  
for (j in a) {  
    document.writeln(j);  
}
```

ANOTHER EXAMPLE

```
var colors = [ "blue",
               "green",
               "yellow"];
var x = window.prompt("enter a
number");
window.bgColor = colors[x];
```

ARRAY OF ARRAYS

- Javascript does not support 2-dimensional arrays (as part of the language).
- BUT - each array element can be an array.
- Resulting syntax looks like C++!

ARRAY OF ARRAYS EXAMPLE

```
var board = [ [1,2,3],  
              [4,5,6],  
              [7,8,9] ];  
  
for (i=0;i<3;i++)  
  for (j=0;j<3;j++)  
    board[i][j]++;
```



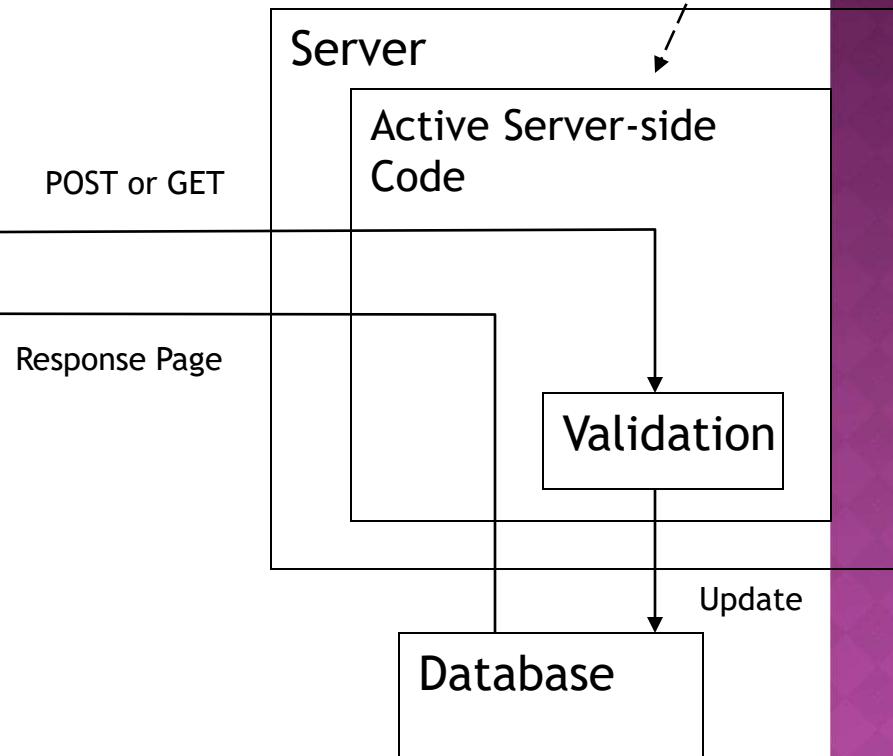
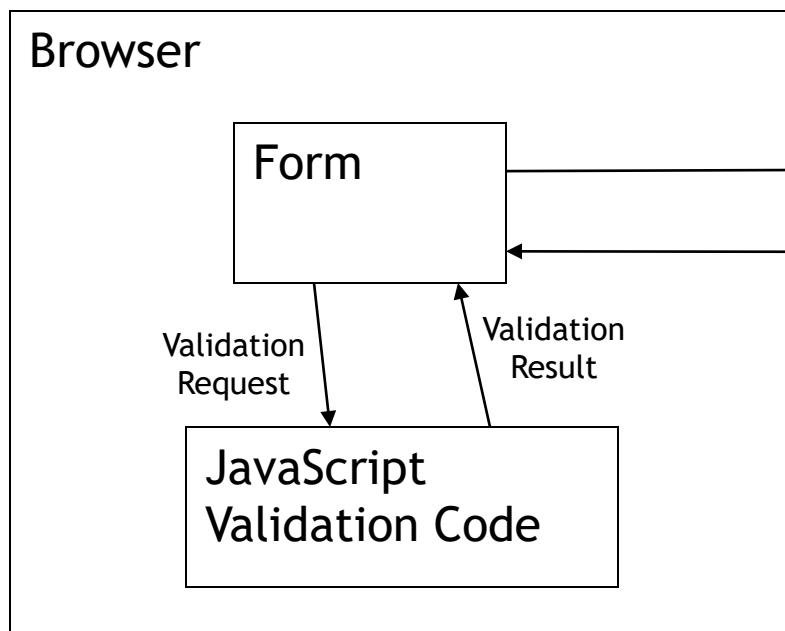
INPUT VALIDATION WITH JAVASCRIPT

- Client-side user input validation
- Use of JavaScript for form user input validation

Regular Expressions
Event Handlers

USER INPUT VALIDATION

Java, Perl, etc.



**Client-side
Validation**

**Server-side
Validation**

CLIENT-SIDE USER INPUT VALIDATION

- Avoids round-trip request to server when form has obvious errors
 - Delays (when network or server is slow)
 - Disruption of page transition
 - Unnecessary server traffic
- Notifying user of error - alternatives:
 - Pop-up “alert box” (annoying, disruptive)
 - Disable submit button plus on-form message (less obvious)

SERVER-SIDE USER INPUT VALIDATION

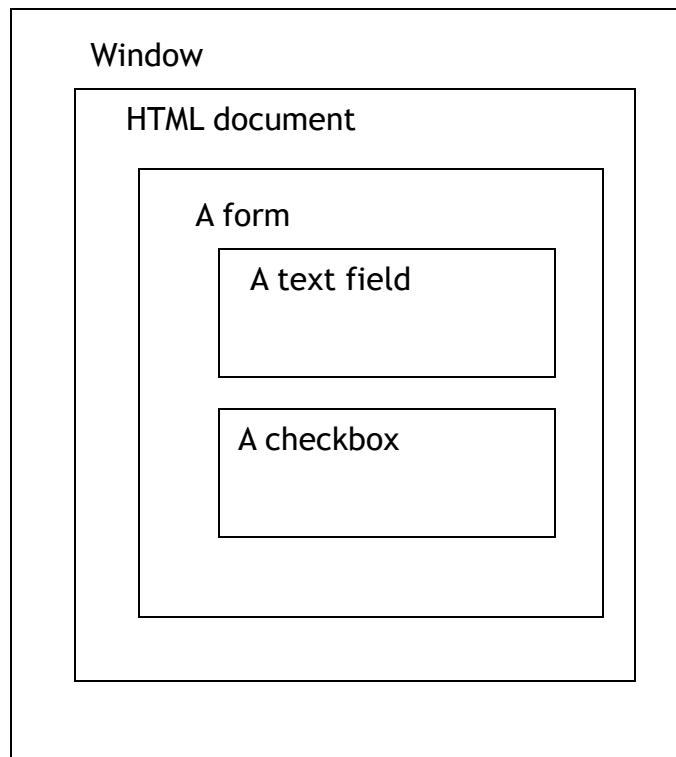
- Client-side validation does not eliminate the need to validate user input on the server
 - A malicious user could copy and modify your page to eliminate the client-side validation
 - Server-side re-validation is crucial if bad user data could be harmful or insecure
 - Client-side validation is to be considered only a convenience for the user and a means to reduce unnecessary traffic on the server
- Sometimes validation is too hard to do on the client

JAVASCRIPT

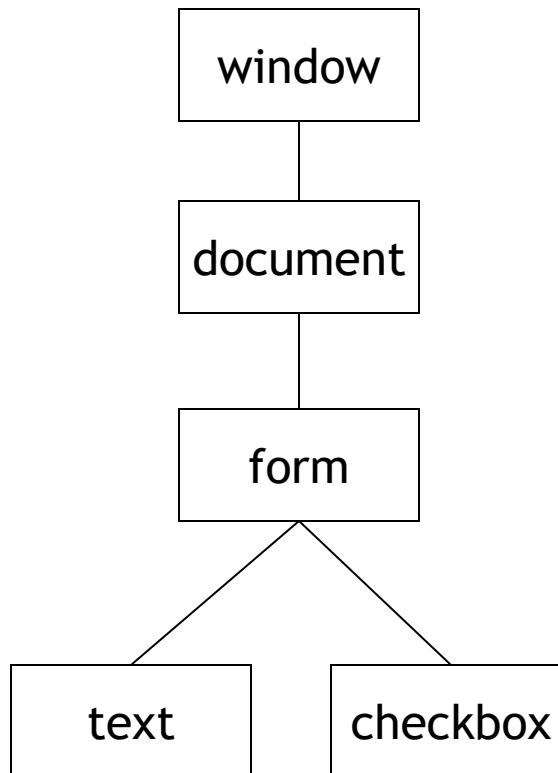
- Scripting language designed for use in *client-side* programming in web pages
- In fact, a powerful, full-featured, object-oriented language
- Originally created by Netscape, which initially called it LiveScript
- Standardized by the European Computer Manufacturers Association.

OBJECTS IN A PAGE

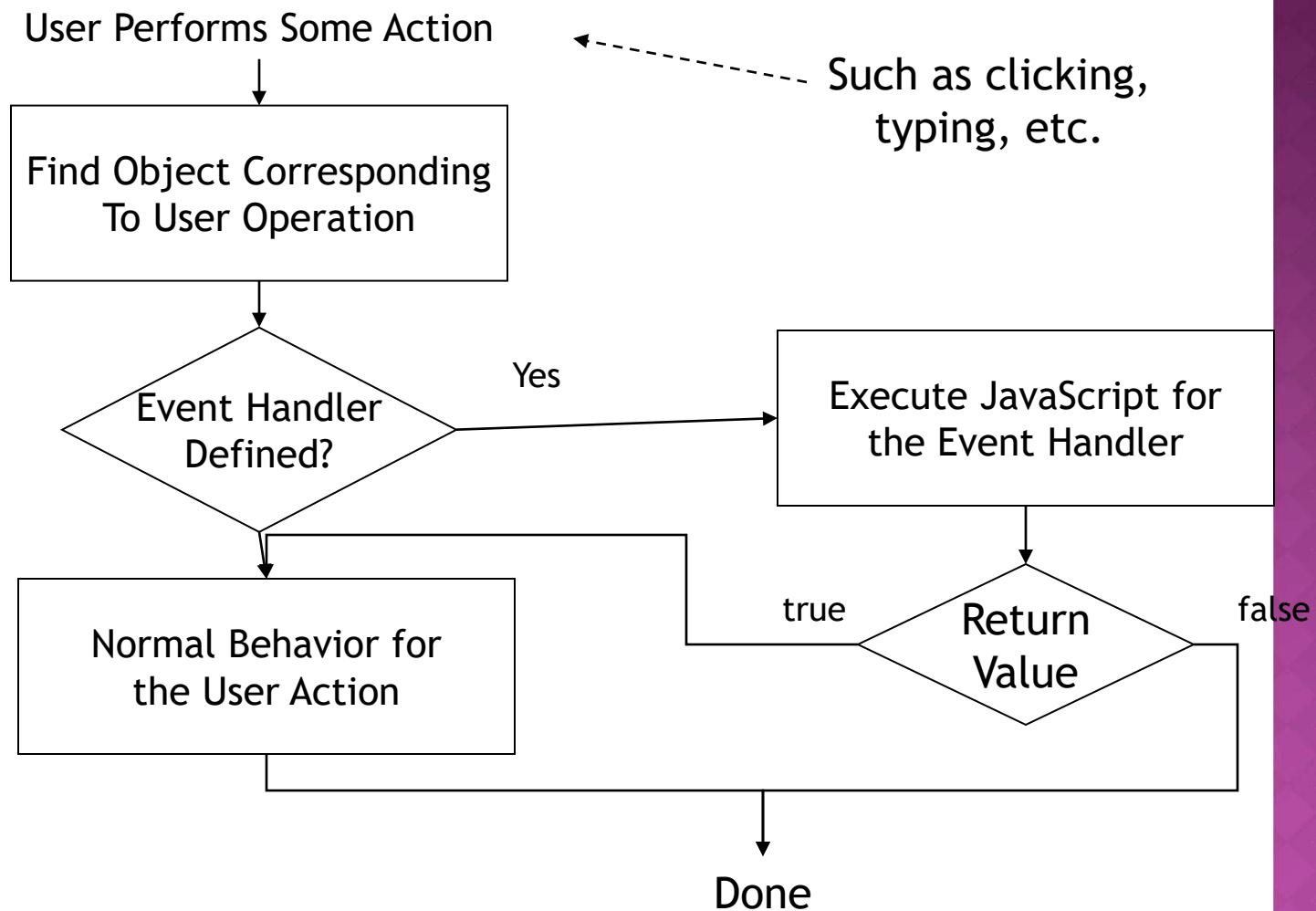
As appearing on the screen



As appearing in the JavaScript
Document Object Model



THE BROWSER INVOKING JAVASCRIPT CODE



SETTING AN EVENT HANDLER

```
<form id="myForm" method="get" action="..."  
onsubmit="return check(this)">
```

- ◉ Declares that the code `return check(this)` will be executed when the form is about to be submitted
- ◉ The code `check(this)` is a call to a function called `check`.
 - The argument `this` refers to the JavaScript object associated with the event (which is the form in this case)
- ◉ The submit will actually occur only if the `check` function returns `true`.

HOW DO WE DEFINE THE FUNCTION CHECK?

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "") {
        alert("The text field is empty.");
        return false;
    }

    if(! form.c1.checked) {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

- Put this code in the <head>
 - It will also work in the <body>, but in the <head> is preferred.
- The <script> tag “hides” the text inside it from HTML parsing
 - So <, etc. can be used without problem
 - The comment lines are used to prevent problems in older browsers that don’t recognize the <script> tag
- Alternatively, the JavaScript code can be read in from a file - discussed later.
- What does this code mean?

DEFINING A FUNCTION

```
<script language="JavaScript">  
<!--<br/>function check(form) {  
    var txtFld = form.t1;  
    if(txtFld.value == "") {  
        alert("The text field is empty.");  
        return false;  
    }  
  
    if(! form.c1.checked) {  
        alert("The checkbox is not checked.");  
        return false;  
    }  
    return true;  
}  
//-->  
</script>
```

The txtFld var is included here only to illustrate the var concept. It's unnecessary and doesn't appear on subsequent slides.

Defines a function and one parameter

- Recall: The parameter is a reference to the form object
- No type is declared for the argument
- No return value type is declared

Declares a local variable called “txtFld” and initializes it

- No type is declared for the variable
- The **var** is optional, but serves to make the variable local, thus preventing collisions with variables called “txtFld” that might be used elsewhere
- **form** contains properties that references all the elements of the form by ID or name

DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "")           ←
    {
        alert("The text field is empty.");
        return false;
    }

    if(! form.c1.checked)
    {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

form.t1.value refers to the value of the text field named **t1**.

- This tests whether the text field's value is the empty string.
- The attributes of a tag will appear as properties of the object associated with the object.
- Note that **string comparison** is done with the **==** operator (unlike Java)

DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "") {
        alert("The text field is empty.");
        return false;
    }
    if(! form.c1.checked) {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

The **alert** function is built in

- The **alert** function pops up a confirm box with the given text and an OK button.
- This is an example to illustrate the coding technique. In good design practice, a more detailed, user-friendly message might be given.

The **check** function returns **false**.

- This tells the browser to **not continue with the submit**.

DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "")
    {
        alert("The text field is empty.");
        return false;
    }
    if( ! form.c1.checked )
    {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

This tests if the checkbox is checked or not.

- The **checked** attribute is a Boolean.
- The **!** is the NOT operator.
- It is, of course, pointless to have a single checkbox that you require to be checked .
 - This is only an example.
 - Normally, there would be multiple checkboxes and you would verify that at least one of them is checked - or whatever you wish.

DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "")
    {
        alert("The text field is empty.");
        return false;
    }

    if( ! form.c1.checked )
    {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

Again there is a popup alert box and the function returns false to indicate that the submit should not occur.

The check function returns **true** if everything is OK. This causes the submit to actually occur.

HTML FOR A SUBMIT EXAMPLE

```
<html>
  <head>
    <title>Submit Example</title>
    <script language="JavaScript">

    </script>           ←
  </head>
  <body>
    <form id="myForm" method="get"
          action="javascript:alert('submitted')"
          onsubmit="return check(this);"
        >
      <input type="text" name="t1" >
      <input type="checkbox" name="c1" value="c1" >
      <input type="submit" >
    </form>

  </body>
</html>
```

JavaScript from
previous slides
goes here.

Temporary
action URL for
testing

THE JAVASCRIPT URL SCHEME

Notice the use of single quotes inside double quotes

```
action="javascript:alert('submitted')"
```

- The URL uses javascript instead of http for the scheme.
- This is understood by the browser to mean
 - Don't send a get request to the server
 - Instead, execute the given text as JavaScript code
 - The resulting value is used as the HTML for a new page
 - To stay on the same page, suppress the value with:
`void(<expression>)`
- This technique can be very useful for debugging and testing - and sometimes useful for production as well.

WHAT HAVE WE DONE SO FAR?

- Added an `onsubmit` attribute to the `<form>` tag that calls the check function, passing a reference to the form by using the `this` keyword.
- Defined the `check` function in a `<script>` tag
 - It tests the values of form fields
 - On error, it pops up an alert box and returns `false`
 - If all is OK, it returns `true`.

ALTERNATIVE FOR INCLUDING JAVASCRIPT CODE

```
<script language="JavaScript"  
src="submitExample.js" >  
</script> ← Still need the end tag
```

○ Benefits

- Can share JavaScript source among many pages
- Removes a lot of clutter from the page - improves readability
 - Becomes really important with servlet- and JSP-generated pages!
- Helps to separate page design and functionality
- Hides your JavaScript code from the *casual observer*
 - But, of course, one can always access the .js file separately.
 - There are techniques for encrypting the JavaScript file, but we won't go into them.

GETTING THE VALUES OF FORM FIELDS

| Type of <input> | Attribute | Attribute Values |
|-----------------------|----------------------|---|
| checkbox or radio | checked | true or false |
| | value | the value to be submitted if true |
| file | value | the URL of the file to be uploaded |
| hidden | value | the value to be submitted |
| text or password | value | the text to be submitted |
| button, submit, reset | N/A | N/A |
| <select> | selectedIndex | 0-based number of the <option> that is selected |
| | options[index].value | the value to be submitted |
| <textarea> | value | the text to be submitted |

SPECIAL CODE FOR ACCESSING RADIO BUTTONS

- Recall, the radio buttons that work as a group are given *the same name*
- Thus, JavaScript uses an *array* to represent them:

```
var radioGroup = form.userRating; ←  
var rb;  
  
var valueChecked = "";  
  
for(rb = 0; rb < radioGroup.length; rb++)  
    if(radioGroup[rb].checked)  
        valueChecked = radioGroup[rb].value;  
  
alert("The value checked is: " + valueChecked);  
  
if(valueChecked == "bad") #unfair validation!  
    return false;  
  
return true;  
}
```

An *array* of all the elements with the name userRating

Arrays have a built-in length property.

VALIDATION UPON FIELD CHANGE

- Check validation as soon as user changes a field
- Benefit: more convenient for user to know of error right away
- Problems:
 - Can't check for required fields
 - Won't prevent submit, so you have to re-validate with onsubmit
 - Can give a spurious error when one of two *dependent fields* are changed
 - Can be annoying if done clumsily

VALIDATION UPON FIELD CHANGE, CONT.

onchange event occurs

- field is losing focus
- but only if its value has changed

```
<input type="checkbox" name="opt1"
       onchange="validate(this)" >
<textarea cols="30" rows="10" name="comment"
       onchange="validate(this)" >
```

THE ONCHANGE EVENT

- Event occurs on a form *field*.
 - The `this` variable refers to the field, not the form
- You have the choice of
 - Defining a separate validation function for each field or
 - Using a common validation function that applies different rules for different fields:
`if(field.name == "password") ...`
- Event fires when the form element loses focus
but only if its value has changed
- Gives you the ability to tell users of errors immediately

EXAMPLE: VERIFY A 5 DIGIT ZIP CODE

```
<input type="text" name="zipcode"
      onChange="validate5Zip(this)" >

function validate5Zip(field)
{ //verify it's a 5-digit zip code
var zip = field.value;
if(zip.length != 5 || isNaN(parseInt(zip)))
{
    alert("please enter a 5 digit zip code.");
    return false;
}
return true;
}
```

Note: Use the keyboard double quote character. These open and close quotes are an artifact of PowerPoint

Converts string to integer and checks if result is “not a number”

EXAMPLE RECAST AS A SINGLE VALIDATION FUNCTION

- The validate function:

```
function validate(field)
{
    var val = field.value;
    switch(field.name)
    {
        case "zipcode":
            if(val.length != 5 || isNaN(parseInt(val)))
            {
                alert("please enter a 5 digit zip code.");
                return false;
            }
            break;
        case ... //Other fields tested here
    }
    return true;
}
```

Note the
switch
statement
works with
strings!

```
<input type="text"
       name="zipcode"
       onchange="validate(this)" >
```

ADDITIONAL TOPICS

ONSUBMIT VALIDATION COMBINED WITH ONCHANGE VALIDATION

```
function check(form)
{
  if(!validateZip(form.zip59) )
    return false;

  if(!form.c1.checked)
  {
    alert("The checkbox is ...");
    return false;
  }
  return true;
}
```

Check for required fields

Re-validate fields with onchange validation in case they remain unchanged from a default value:

- Invoke the onchange validation function used in the onsubmit handler
- Avoid duplicated code.

Sidebar

Be careful of alerts and other messages when you invoke a validation function from inside another.

Avoid duplicate warnings and other annoying behavior.

CROSS-FIELD VALIDATION

When one field's validation depends on the value of another field:

- In onsubmit validation, there's no new problem since the `form` object is passed as a parameter
- In onchange validation, you access the `form` object through the field's `form` property:

```
function validateShipAddress(field)
{
    var isSameAddress = field.form.sameAddressCheck.checked;
    if(!isSameAddress) //if user hasn't said ship address is same
        if(!field.value)
        {
            alert("Please enter a shipping address or check the box"
                  + "\"Shipping Address Is Same As Billing Address\"");
            return false;
        }
    return true;
}
```

WORKING WITH FORMS AND REGULAR EXPRESSIONS

Validating a Web Form with JavaScript

INTRODUCING REGULAR EXPRESSIONS

- A **regular expression** is a text string that defines a character pattern
- One use of regular expressions is **pattern-matching**, in which a text string is tested to see whether it matches the pattern defined by a regular expression

INTRODUCING REGULAR EXPRESSIONS

○ Creating a regular expression

- You create a regular expression in JavaScript using the command

```
re = /pattern/;
```

- This syntax for creating regular expressions is sometimes referred to as a **regular expression literal**

INTRODUCING REGULAR EXPRESSIONS

⦿ Matching a substring

- The most basic regular expression consists of a substring that you want to locate in the test string
- The regular expression to match the first occurrence of a substring is `/chars/`

INTRODUCING REGULAR EXPRESSIONS

○ Setting regular expression flags

- To make a regular expression not sensitive to case, use the regular expression literal `/pattern/i`
- To allow a global search for all matches in a test string, use the regular expression literal `/pattern/g`

INTRODUCING REGULAR EXPRESSIONS

○ Defining character positions

| Character | Description | Example |
|-----------|--|--|
| ^ | Indicates the beginning of the text string | /^GPS/ matches "GPS-ware" but not "Products from GPS-ware" |
| \$ | Indicates the end of the text string | /ware\$/ matches "GPS-ware" but not "GPS-ware Products" |
| \b | Indicates the presence of a word boundary | /\bart/ matches "art" and "artists" but not "dart" |
| \B | Indicates the absence of a word boundary | /art\B/ matches "dart" but not "artist" |

INTRODUCING REGULAR EXPRESSIONS

○ Defining character positions

| Character | Description | Example |
|-----------|---|---|
| \d | A digit (from 0 to 9) | /\dth/ matches "5th" but not "ath" |
| \D | A non-digit | /\Ds/ matches "as" but not "5s" |
| \w | A word character (an upper or lower case letter, a digit, or an underscore) | /\w\w/ matches "to" or "A1" but not "\$x" or "*" |
| \W | A non-word character | /\W/ matches "\$" or "&" but not "a", "B", or "3" |
| \s | A white space character (a blank space, tab, new line, carriage return, or form feed) | /\s\d\s/ matches " 5 " but not "5" |
| \S | A non-white space character | /\S\d\S/ matches "345" or "a5b" but not "5" |
| . | Any character | ./ matches anything |

INTRODUCING REGULAR EXPRESSIONS

○ Defining character positions

- Can specify a collection of characters known as a **character class** to limit the regular expression to only a select group of characters

INTRODUCING REGULAR EXPRESSIONS

○ Defining character positions

| Character | Description | Example |
|-------------------------|--|---|
| [<i>chars</i>] | Match any character in the list of characters, <i>chars</i> | /[dog]/ matches "god" and "dog" |
| [^ <i>chars</i>] | Do not match any character in <i>chars</i> | /[^dog]/ matches neither "god" nor "dog" |
| [<i>char1-charN</i>] | Match characters in the range <i>char1</i> through <i>charN</i> | /[a-c]/ matches the lowercase letters a through c |
| [^ <i>char1-charN</i>] | Do not match characters in the range <i>char1</i> through <i>charN</i> | /[^a-c]/ does not match the lowercase letters a through c |
| [a-z] | Match lowercase letters | /[a-z][a-z]/ matches any two consecutive lowercase letters |
| [A-Z] | Match uppercase letters | /[A-Z][A-Z]/ matches any two consecutive uppercase letters |
| [a-zA-Z] | Match letters | /[a-zA-Z][a-zA-Z]/ matches any two consecutive letters |
| [0-9] | Match digits | /[1]0-9]/ matches the numbers "10" through "19" |
| [0-9a-zA-Z] | Match digits and letters | /[0-9a-zA-Z][0-9a-zA-Z]/ matches any two consecutive letters or numbers |

INTRODUCING REGULAR EXPRESSIONS

○ Repeating characters

| Repetition Character(s) | Description | Example |
|-------------------------|--|--|
| * | Repeat 0 or more times | <code>\s*/</code> matches 0 or more consecutive white space characters |
| ? | Repeat 0 or 1 time | <code>/colou?r/</code> matches "color" or "colour" |
| + | Repeat 1 or more times | <code>\s+/</code> matches 1 or more consecutive white space characters |
| {n} | Repeat exactly <i>n</i> times | <code>\d{9}/</code> matches a nine digit number |
| {n, } | Repeat at least <i>n</i> times | <code>\d{9,}/</code> matches a number with at least nine digits |
| {n,m} | Repeat at least <i>n</i> times but no more than <i>m</i> times | <code>\d{5,9}/</code> matches a number with 5 to 9 digits |

INTRODUCING REGULAR EXPRESSIONS

○ Escape Sequences

- An **escape sequence** is a special command inside a text string that tells the JavaScript interpreter not to interpret what follows as a character
- The character which indicates an escape sequence in a regular expression is the backslash character \

INTRODUCING REGULAR EXPRESSIONS

○ Escape Sequences

| Escape Sequence | Represents | Example |
|-----------------|------------------------|--|
| \/ | The / character | /d\/d/ matches "5/9" "3/1" but not "59" or "31" |
| \\\ | The \ character | /d\\\\d/ matches "5\9" or "3\1" but not "59" or "31" |
| \. | The . character | /d.\d/d/ matches "3.20" or "5.95" but not "320" or "595" |
| * | The * character | /[a-z]{4}*/ matches "help*" or "pass*" |
| \+ | The + character | /d\+\\d/ matches "5+9" or "3+1" but not "59" or "39" |
| \? | The ? character | /[a-z]{4}\?/ matches "help?" or "info?" |
| \ | The character | /a\\lb/ matches "alb" |
| \(\) | The (and) characters | /(\d{3})\\)/ matches "(800)" or "(555)" |
| \{ \} | The { and } characters | /\{[a-z]{4}\\}/ matches "{pass}" or "{info}" |
| \^ | The ^ character | /d+\\^\\d/ matches "321^2" or "4^3" |
| \\$ | The \$ character | /\\$\\d{2}\\.\\d{2}/ matches "\$59.95" or "\$19.50" |
| \n | A new line | /\\n/ matches the occurrence of a new line in the text string |
| \r | A carriage return | /\\r/ matches the occurrence of a carriage return in the text string |
| \t | A tab | /\\t/ matches the occurrence of a tab in the text string |

INTRODUCING REGULAR EXPRESSIONS

○ Alternating Patterns and Grouping

| Characters | Description | Example |
|--------------------------|---|---|
| <i>pattern1 pattern2</i> | Matches either <i>pattern1</i> or <i>pattern2</i> | /color colour/ matches either "color" or "colour" |
| (<i>pattern</i>) | Treats <i>pattern</i> as a single group and allows a back-reference to the captured group | /(Mr)\.\s)?\w+/ matches either "Mr. Smith" or "Smith" |
| \n | Back-reference to group <i>n</i> in the regular expression | /(\s)\1/ matches consecutive occurrences of white space |
| (? <i>pattern</i>) | Treats <i>pattern</i> as a single group, but does not allow for back-referencing | |

INTRODUCING REGULAR EXPRESSIONS

- The regular expression object constructor
 - To create a regular expression object
`re = new RegExp(pattern, flags)`
 - *re* is the regular expression object, *pattern* is a text string of the regular expression pattern, and *flags* is a text string of the regular expression flags

WORKING WITH THE REGULAR EXPRESSION OBJECT

○ Regular Expression methods

| Method | Description |
|--|--|
| <code>re.compile(pattern,flags)</code> | Compiles or recompiles a regular expression <code>re</code> , where <code>pattern</code> is the text string of new regular expression pattern and <code>flags</code> are flags applied to the <code>pattern</code> |
| <code>re.exec(text)</code> | Executes a search on <code>text</code> using the regular expression <code>re</code> ; pattern results are returned in an array and reflected in the properties of the global <code>RegExp</code> object |
| <code>re.match(text)</code> | Performs a pattern match in <code>text</code> using the <code>re</code> regular expression; matched substrings are stored in an array |
| <code>text.replace(re, newsubstr)</code> | Replaces the substring defined by the regular expression <code>re</code> in the text string <code>text</code> with <code>newsubstr</code> |
| <code>text.search(re)</code> | Searches <code>text</code> for a substring matching the regular expression <code>re</code> ; returns the index of the match, or -1 if no match is found |
| <code>text.split(re)</code> | Splits <code>text</code> at each point indicated by the regular expression <code>re</code> ; the substrings are stored in an array |
| <code>re.test(text)</code> | Performs a pattern match on the text string <code>text</code> using the regular expression <code>re</code> , returning the Boolean value true if a match is found and false otherwise |

TIPS FOR VALIDATING FORMS

- Use selection lists, option buttons, and check boxes to limit the ability of users to enter erroneous data
- Indicate to users which fields are required, and if possible, indicate the format that each field value should be entered in
- Use the maxlength attribute of the input element to limit the length of text entered into a form field

TIPS FOR VALIDATING FORMS

- Format financial values using the `toFixed()` and `toPrecision()` methods. For older browsers use custom scripts to format financial data
- Apply client-side validation checks to lessen the load of the server
- Use regular expressions to verify that field values correspond to a required pattern

TIPS FOR VALIDATING FORMS

- Use the length property of the string object to test whether the user has entered a value in a required field
- Test credit card numbers to verify that they match the patterns specified by credit card companies
- Test credit card numbers to verify that they fulfill the Luhn Formula

UI/UX DESIGN PRINCIPLES

USER INTERFACE DESIGN

- Designing effective interfaces for software systems

THE USER INTERFACE

- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used

GRAPHICAL USER INTERFACES

- Most users of business systems interact with these systems through graphical interfaces although, in some cases, legacy text-based interfaces are still used

GUI CHARACTERISTICS

| Characteristic | Description |
|----------------|---|
| Windows | Multiple windows allow different information to be displayed simultaneously on the user's screen. |
| Icons | Icons represent different types of information. On some systems, icons represent files; on others, icons represent processes. |
| Menus | Commands are selected from a menu rather than typed in a command language. |
| Pointing | A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interest in a window. |
| Graphics | Graphical elements can be mixed with text on the same display. |

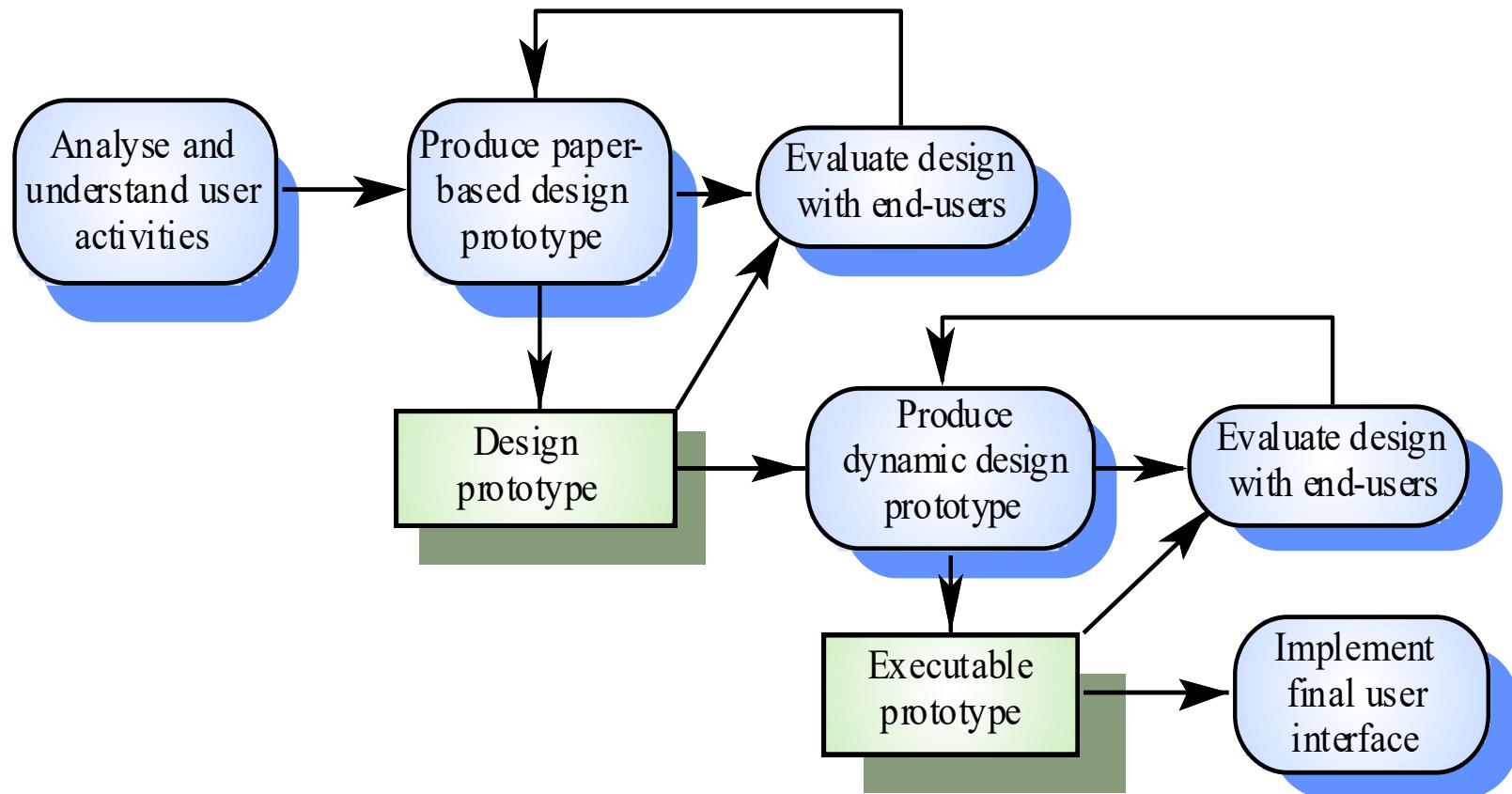
GUI ADVANTAGES

- They are easy to learn and use.
 - Users without experience can learn to use the system quickly.
- The user may switch quickly from one task to another and can interact with several different applications.
 - Information remains visible in its own window when attention is switched.
- Fast, full-screen interaction is possible with immediate access to anywhere on the screen

USER-CENTRED DESIGN

- User-centred design is an approach to UI design where the needs of the user are paramount and where the user is involved in the design process
- UI design always involves the development of prototype interfaces

USER INTERFACE DESIGN PROCESS



UI DESIGN PRINCIPLES

- ◉ UI design must take account of the needs, experience and capabilities of the system users
- ◉ Designers should be aware of people's physical and mental limitations (e.g. limited short-term memory) and should recognise that people make mistakes
- ◉ UI design principles underlie interface designs although not all principles are applicable to all designs

DESIGN PRINCIPLES

○ User familiarity

- The interface should be based on user-oriented terms and concepts rather than computer concepts. For example, an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.

○ Consistency

- The system should display an appropriate level of consistency. Commands and menus should have the same format, command punctuation should be similar, etc.

○ Minimal surprise

- If a command operates in a known way, the user should be able to predict the operation of comparable commands

DESIGN PRINCIPLES

○ Recoverability

- The system should provide some resilience to user errors and allow the user to recover from errors. This might include an undo facility, confirmation of destructive actions, 'soft' deletes, etc.

○ User guidance

- Some user guidance such as help systems, on-line manuals, etc. should be supplied

○ User diversity

- Interaction facilities for different types of user should be supported. For example, some users have seeing difficulties and so larger text should be available

USER-SYSTEM INTERACTION

- Two problems must be addressed in interactive systems design
 - How should information from the user be provided to the computer system?
 - How should information from the computer system be presented to the user?
- User interaction and information presentation may be integrated through a coherent framework such as a user interface metaphor

INTERACTION STYLES

- Direct manipulation
- Menu selection
- Form fill-in
- Command language
- Natural language

| Interaction style | Main advantages | Main disadvantages | Application examples |
|--------------------------|---|--|---|
| Direct manipulation | Fast and intuitive interaction Easy to learn | May be hard to implement Only suitable where there is a visual metaphor for tasks and objects | Video games CAD systems |
| Menu selection | Avoids user error Little typing required | Slow for experienced users Can become complex if many menu options | Most general-purpose systems |
| Form fill-in | Simple data entry Easy to learn | Takes up a lot of screen space | Stock control, Personal loan processing |
| Command language | Powerful and flexible | Hard to learn Poor error management | Operating systems, Library information retrieval systems |
| Natural language | Accessible to casual users Easily extended | Requires more typing Natural language understanding systems are unreliable | Timetable systems WWW information retrieval systems |

ADVANTAGES AND DISADVANTAGES

DIRECT MANIPULATION ADVANTAGES

- Users feel in control of the computer and are less likely to be intimidated by it
- User learning time is relatively short
- Users get immediate feedback on their actions so mistakes can be quickly detected and corrected

DIRECT MANIPULATION PROBLEMS

- The derivation of an appropriate information space model can be very difficult
- Given that users have a large information space, what facilities for navigating around that space should be provided?
- Direct manipulation interfaces can be complex to program and make heavy demands on the computer system

CONTROL PANEL INTERFACE

| | | | |
|-----------|--------------|--------|--------------|
| Title | JSD. example | Grid | Busy |
| Method | JSD | | |
| Type | Network | Units | cm ► OUIT |
| Selection | Process | Reduce | Full ► PRINT |
| NODE | LINKS | FONT | LABEL EDIT |

MENU SYSTEMS

- Users make a selection from a list of possibilities presented to them by the system
- The selection may be made by pointing and clicking with a mouse, using cursor keys or by typing the name of the selection
- May make use of simple-to-use terminals such as touchscreens

ADVANTAGES OF MENU SYSTEMS

- Users need not remember command names as they are always presented with a list of valid commands
- Typing effort is minimal
- User errors are trapped by the interface
- Context-dependent help can be provided.
The user's context is indicated by the current menu selection

PROBLEMS WITH MENU SYSTEMS

- ◉ Actions which involve logical conjunction (and) or disjunction (or) are awkward to represent
- ◉ Menu systems are best suited to presenting a small number of choices. If there are many choices, some menu structuring facility must be used
- ◉ Experienced users find menus slower than command language

FORM-BASED INTERFACE

NEWBOOK

| | | | |
|------------------|----------------------|------------------|----------------------|
| Title | <input type="text"/> | ISBN | <input type="text"/> |
| Author | <input type="text"/> | Price | <input type="text"/> |
| Publisher | <input type="text"/> | Publication date | <input type="text"/> |
| Edition | <input type="text"/> | Number of copies | <input type="text"/> |
| Classification | <input type="text"/> | Loan status | <input type="text"/> |
| Date of purchase | <input type="text"/> | Order status | <input type="text"/> |

COMMAND INTERFACES

- User types commands to give instructions to the system e.g. UNIX
- May be implemented using cheap terminals.
- Easy to process using compiler techniques
- Commands of arbitrary complexity can be created by command combination
- Concise interfaces requiring minimal typing can be created

PROBLEMS WITH COMMAND INTERFACES

- Users have to learn and remember a command language. Command interfaces are therefore unsuitable for occasional users
- Users make errors in command. An error detection and recovery system is required
- System interaction is through a keyboard so typing ability is required

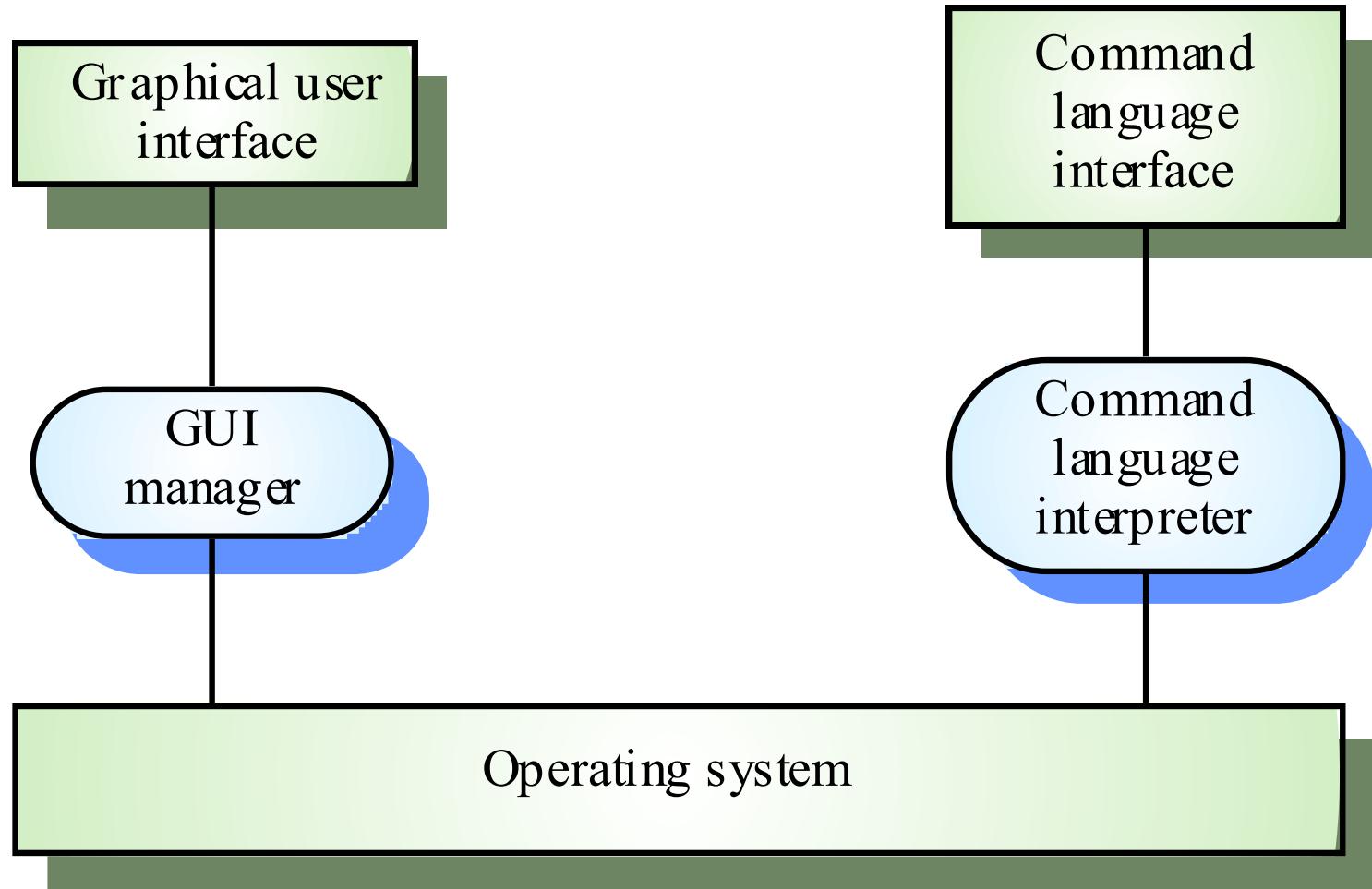
COMMAND LANGUAGES

- ◉ Often preferred by experienced users because they allow for faster interaction with the system
- ◉ Not suitable for casual or inexperienced users
- ◉ May be provided as an alternative to menu commands (keyboard shortcuts). In some cases, a command language interface and a menu-based interface are supported at the same time

NATURAL LANGUAGE INTERFACES

- The user types a command in a natural language. Generally, the vocabulary is limited and these systems are confined to specific application domains (e.g. timetable enquiries)
- NL processing technology is now good enough to make these interfaces effective for casual users but experienced users find that they require too much typing

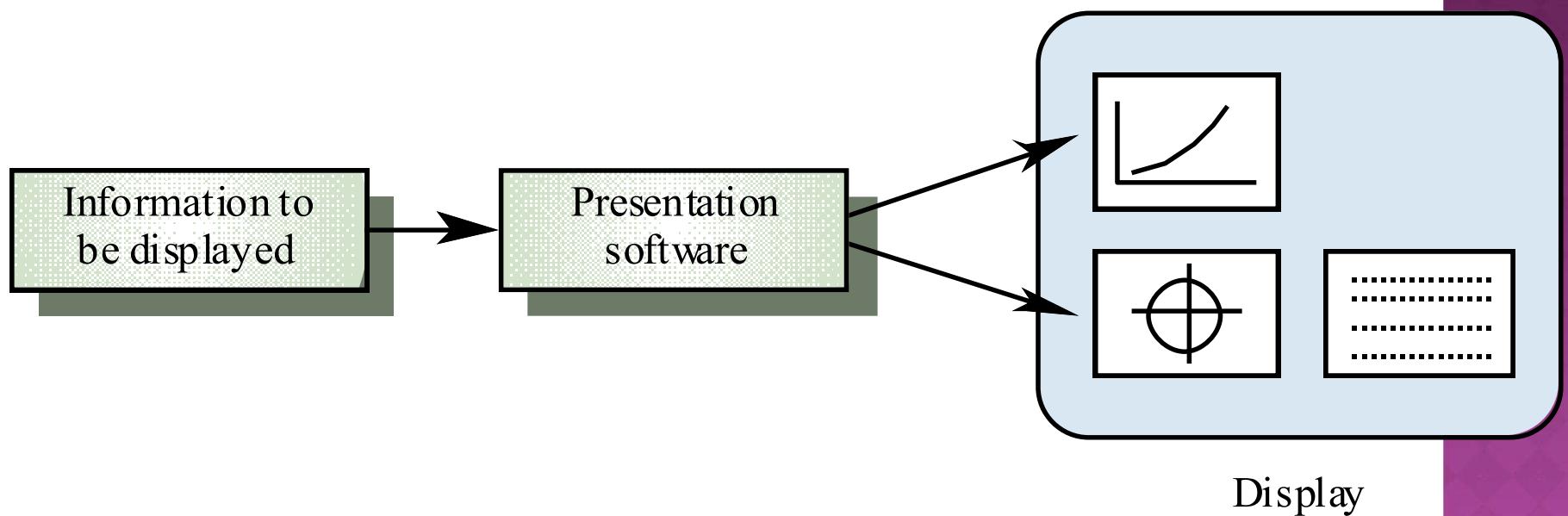
MULTIPLE USER INTERFACES



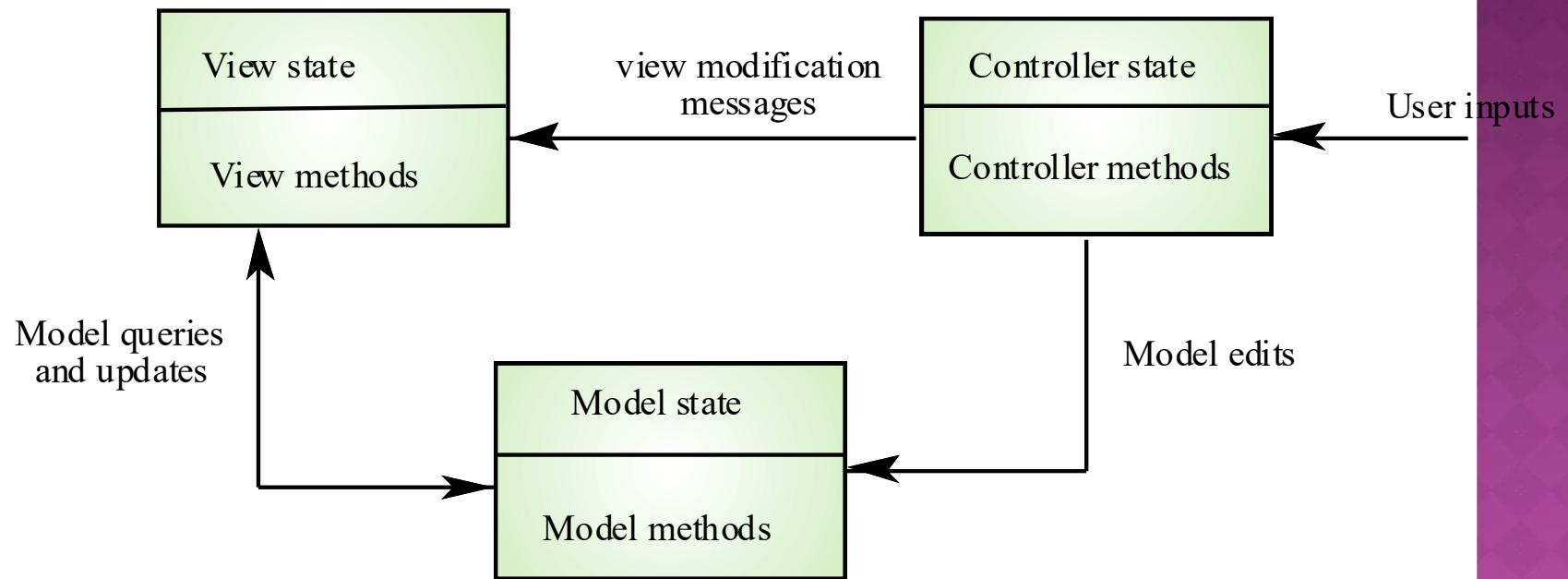
INFORMATION PRESENTATION

- Information presentation is concerned with presenting system information to system users
- The information may be presented directly (e.g. text in a word processor) or may be transformed in some way for presentation (e.g. in some graphical form)
- The Model-View-Controller approach is a way of supporting multiple presentations of data

INFORMATION PRESENTATION



MODEL-VIEW-CONTROLLER



INFORMATION PRESENTATION

○ Static information

- Initialised at the beginning of a session. It does not change during the session
- May be either numeric or textual

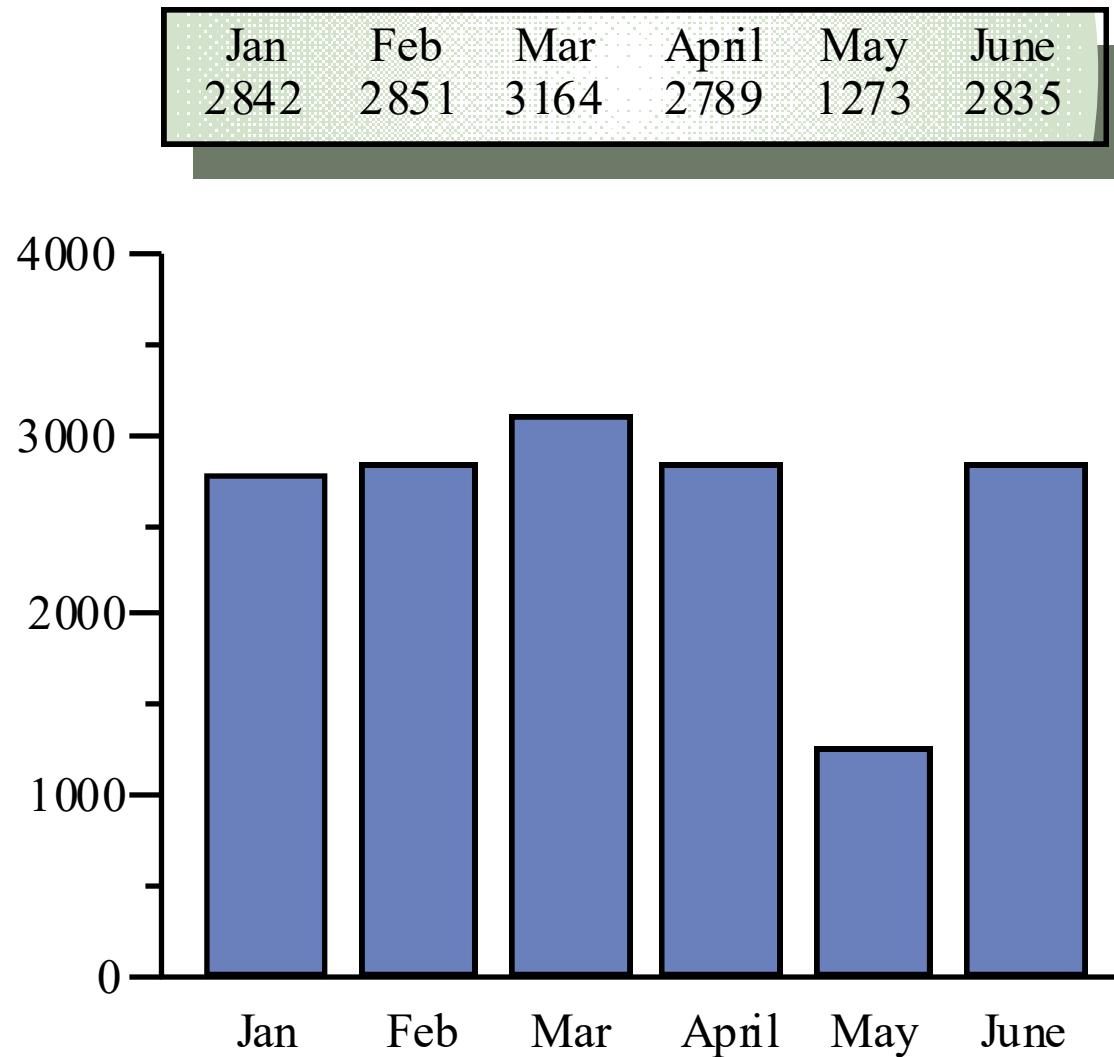
○ Dynamic information

- Changes during a session and the changes must be communicated to the system user
- May be either numeric or textual

INFORMATION DISPLAY FACTORS

- Is the user interested in precise information or data relationships?
- How quickly do information values change? Must the change be indicated immediately?
- Must the user take some action in response to a change?
- Is there a direct manipulation interface?
- Is the information textual or numeric? Are relative values important?

ALTERNATIVE INFORMATION PRESENTATIONS



ANALOGUE VS. DIGITAL PRESENTATION

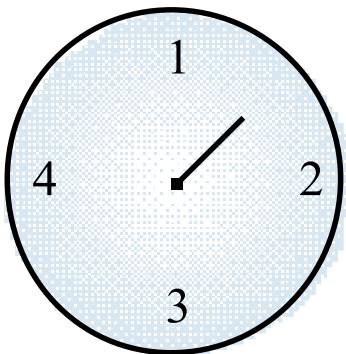
○ Digital presentation

- Compact - takes up little screen space
- Precise values can be communicated

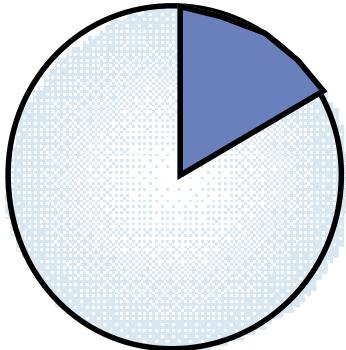
○ Analogue presentation

- Easier to get an 'at a glance' impression of a value
- Possible to show relative values
- Easier to see exceptional data values

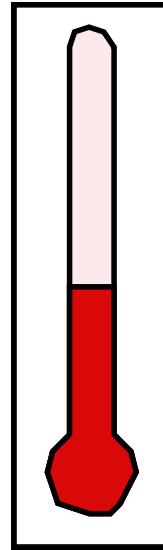
DYNAMIC INFORMATION DISPLAY



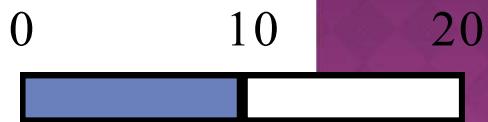
Dial with needle



Pie chart

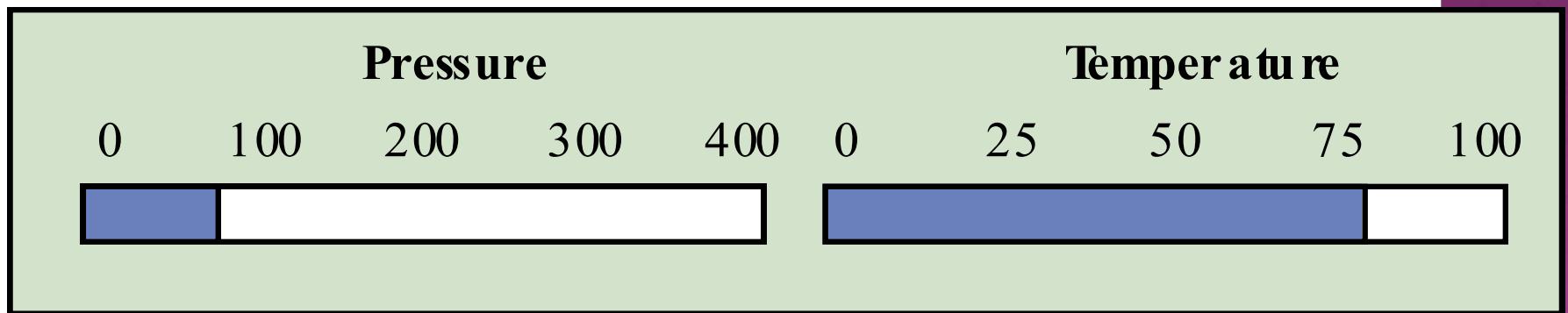


Thermometer



Horizontal bar

DISPLAYING RELATIVE VALUES



TEXTUAL HIGHLIGHTING



The filename you have chosen has been used. Please choose an other name

Ch. 16 User interface design

OK

Cancel

DATA VISUALISATION

- Concerned with techniques for displaying large amounts of information
- Visualisation can reveal relationships between entities and trends in the data
- Possible data visualisations are:
 - Weather information collected from a number of sources
 - The state of a telephone network as a linked set of nodes
 - Chemical plant visualised by showing pressures and temperatures in a linked set of tanks and pipes
 - A model of a molecule displayed in 3 dimensions
 - Web pages displayed as a hyperbolic tree

COLOUR DISPLAYS

- Colour adds an extra dimension to an interface and can help the user understand complex information structures
- Can be used to highlight exceptional events
- Common mistakes in the use of colour in interface design include:
 - The use of colour to communicate meaning
 - Over-use of colour in the display

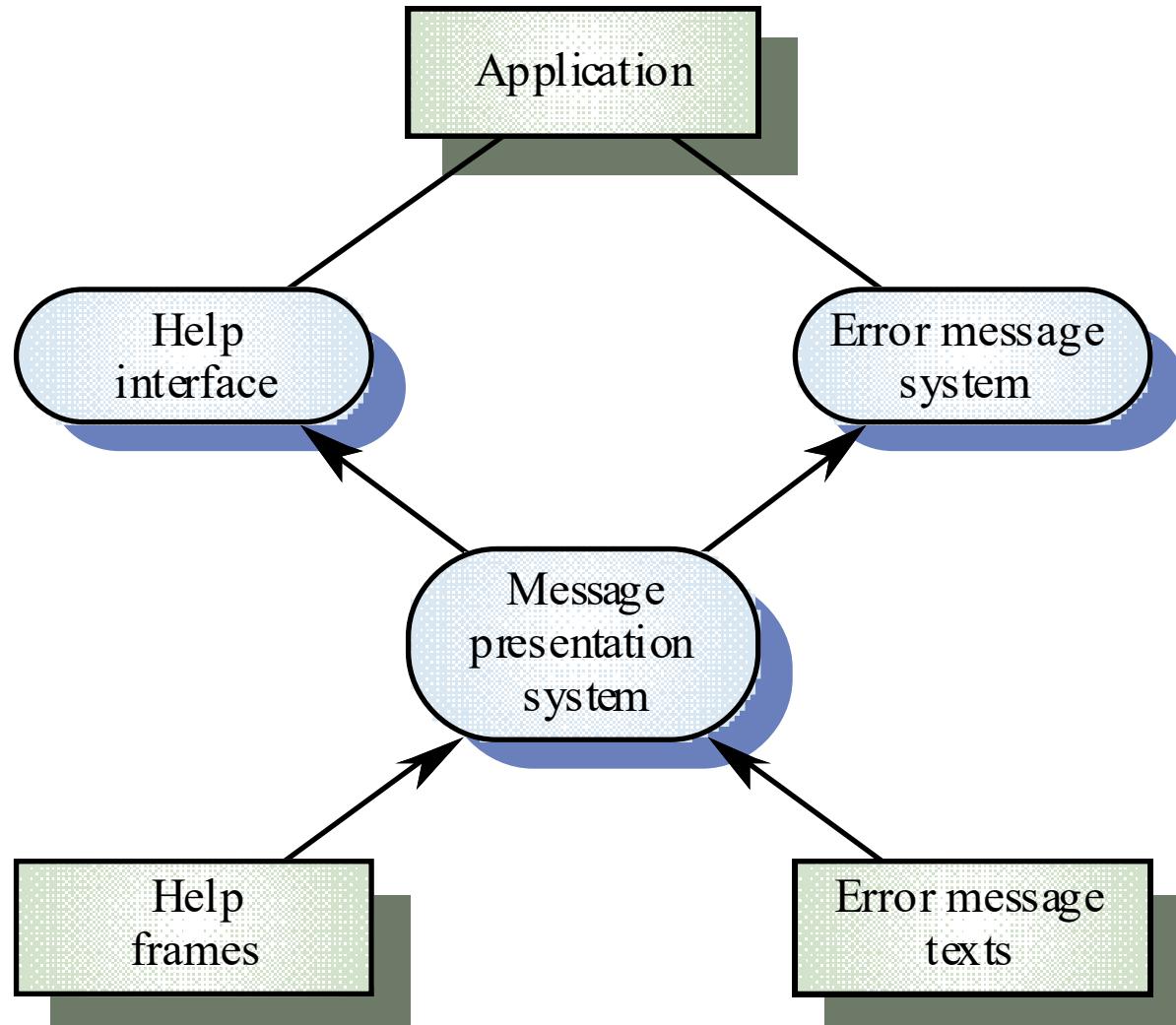
COLOUR USE GUIDELINES

- Don't use too many colours
- Use colour coding to support use tasks
- Allow users to control colour coding
- Design for monochrome then add colour
- Use colour coding consistently
- Avoid colour pairings which clash
- Use colour change to show status change
- Be aware that colour displays are usually lower resolution

USER SUPPORT

- User guidance covers all system facilities to support users including on-line help, error messages, manuals etc.
- The user guidance system should be integrated with the user interface to help users when they need information about the system or when they make some kind of error
- The help and message system should, if possible, be integrated

HELP AND MESSAGE SYSTEM



ERROR MESSAGES

- Error message design is critically important.
Poor error messages can mean that a user
rejects rather than accepts a system
- Messages should be polite, concise,
consistent
and constructive
- The background and experience of users
should be the determining factor in message
design

DESIGN FACTORS IN MESSAGE WORDING

| | |
|-------------|---|
| Context | The user guidance system should be aware of what the user is doing and should adjust the output message to the current context. |
| Experience | As users become familiar with a system they become irritated by long, 'meaningful' messages. However, beginners find it difficult to understand short terse statements of the problem. The user guidance system should provide both types of message and allow the user to control message conciseness. |
| Skill level | Messages should be tailored to the user's skills as well as their experience. Messages for the different classes of user may be expressed in different ways depending on the terminology which is familiar to the reader. |
| Style | Messages should be positive rather than negative. They should use the active rather than the passive mode of address. They should never be insulting or try to be funny. |
| Culture | Wherever possible, the designer of messages should be familiar with the culture of the country where the system is sold. There are distinct cultural differences between Europe, Asia and America. A suitable message for one culture might be unacceptable in another. |

NURSE INPUT OF A PATIENT'S NAME

Please type the patient name in the box then click ok

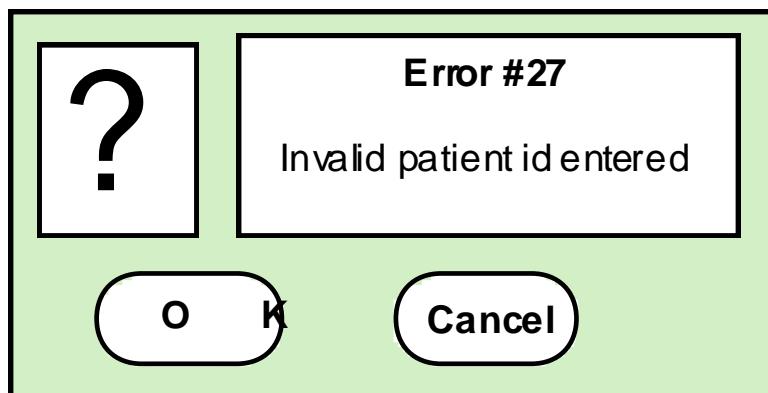
Bates , J.

OK

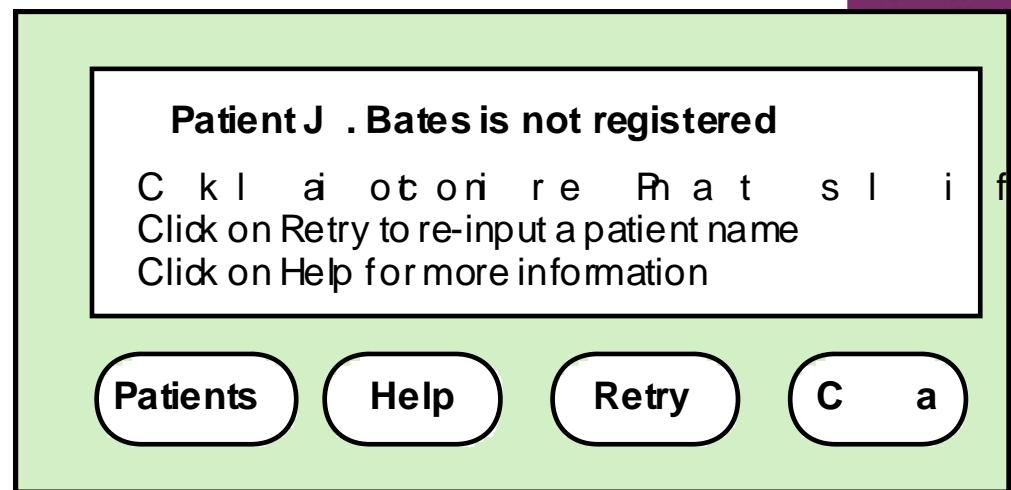
Cancel

SYSTEM AND USER-ORIENTED ERROR MESSAGES

System-oriented error message



User-oriented error message



HELP SYSTEM DESIGN

- *Help?* means ‘help I want information’
- *Help!* means “HELP. I'm in trouble”
- Both of these requirements have to be taken into account in help system design
- Different facilities in the help system may be required

HELP INFORMATION

- Should not simply be an on-line manual
- Screens or windows don't map well onto paper pages.
- The dynamic characteristics of the display can improve information presentation.
- People are not so good at reading screen as they are text.

HELP SYSTEM USE

- ◉ Multiple entry points should be provided so that the user can get into the help system from different places.
- ◉ Some indication of where the user is positioned in the help system is valuable.
- ◉ Facilities should be provided to allow the user to navigate and traverse the help system.

USER DOCUMENTATION

- As well as on-line information, paper documentation should be supplied with a system
- Documentation should be designed for a range of users from inexperienced to experienced
- As well as manuals, other easy-to-use documentation such as a quick reference card may be provided

DOCUMENT TYPES

- Functional description
 - Brief description of what the system can do
- Introductory manual
 - Presents an informal introduction to the system
- System reference manual
 - Describes all system facilities in detail
- System installation manual
 - Describes how to install the system
- System administrator's manual
 - Describes how to manage the system when it is in use

USER INTERFACE EVALUATION

- Some evaluation of a user interface design should be carried out to assess its suitability
- Full scale evaluation is very expensive and impractical for most systems
- Ideally, an interface should be evaluated against a usability specification. However, it is rare for such specifications to be produced

USABILITY ATTRIBUTES

| Attribute | Description |
|--------------------|--|
| Learnability | How long does it take a new user to become productive with the system? |
| Speed of operation | How well does the system response match the user's work practice? |
| Robustness | How tolerant is the system of user error? |
| Recoverability | How good is the system at recovering from user errors? |
| Adaptability | How closely is the system tied to a single model of work? |

SIMPLE EVALUATION TECHNIQUES

- Questionnaires for user feedback
- Video recording of system use and subsequent tape evaluation.
- Instrumentation of code to collect information about facility use and user errors.
- The provision of a grip button for on-line user feedback.

1.00

CUT Copy Format Painter Clipboard Slides

Layout New Slide Reset Delete

Font Paragraph Drawing Editing

Slides Outline

1 TIMING EVENTS

2 WHAT ARE TIMING EVENTS?

- When a particular time period is to be given as gap between same or different events to be performed we use timing events.
- The window object allows execution of code at specified time intervals.
- These time intervals are called timing events.

3 TIMING EVENTS

- The two key methods to use with JavaScript are:
- `setTimeout(function, milliseconds)` Executes a function, after waiting a specified number of milliseconds.
- `setInterval(function, milliseconds)` Same as `setTimeout()`, but repeats the execution of the function continuously.

4 APPLICATION IMAGE SLIDER

Click to add notes

83%

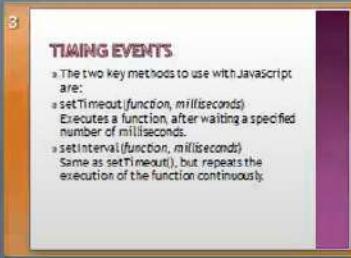
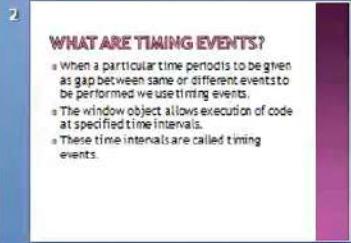
The screenshot shows a Microsoft PowerPoint slide titled "WHAT ARE TIMING EVENTS?". The slide contains a bulleted list of three points explaining what timing events are and how they work. The slide is part of a presentation with four slides in total, as indicated by the navigation pane on the left. The presentation is titled "Lec -16 4/02/2021.mp4" and is in "Compatibility Mode". The slide has a blue background with a white title area. The text is in a black sans-serif font. The navigation pane shows the following slides:

- Slide 1: TIMING EVENTS
- Slide 2: WHAT ARE TIMING EVENTS?
 - When a particular time period is to be given as gap between same or different events to be performed we use timing events.
 - The window object allows execution of code at specified time intervals.
 - These time intervals are called timing events.
- Slide 3: TIMING EVENTS
 - The two key methods to use with JavaScript are:
 - `setTimeout(function, milliseconds)` Executes a function, after waiting a specified number of milliseconds.
 - `setInterval(function, milliseconds)` Same as `setTimeout()`, but repeats the execution of the function continuously.
- Slide 4: APPLICATION IMAGE SLIDER



Slides

Outline



TIMING EVENTS

- The two key methods to use with JavaScript are:
- **setTimeout(*function, milliseconds*)**
Executes a function, after waiting a specified number of milliseconds.
- **setInterval(*function, milliseconds*)**
Same as setTimeout(), but repeats the execution of the function continuously.

Click to add notes



MALE SAICHARANREDDY



Akshay Jain



TIKAK KUMAR N



1.00

Cut Copy Format Painter Paste New Slide Reset Delete Clipboard Slides Font Paragraph Drawing Editing

Slides Outline

APPLICATION-IMAGE SLIDER

```

var slideIndex = 0;
showSlides(); // call showslide method

function showSlides()
{
    var i;

    // get the array of div's with classname image-slidernode
    var slides = document.getElementsByClassName("image-slidernode");

    // get the array of div's with classname dot
    var dots = document.getElementsByClassName("dot");

    for (i = 0; i < slides.length; i++) {
        // initially set the display to
        // none for every image.
        slides[i].style.display = "none";
    }

    // increase by 1, Global variable
    slideIndex++;

    // check for boundary
    if (slideIndex > slides.length)
    {
        slideIndex = 1;
    }
}

```

CONTD..

```

for (i = 0; i < dots.length; i++) {
    dots[i].className = dots[i].className.replace(" active", "");
}

slides[slideIndex - 1].style.display = "block";
dots[slideIndex - 1].className += " active";

// Change image every 3 seconds
setInterval(showSlides, 2000);
}

```

COOKIES
(session management)

COOKIE (SESSION MANAGEMENT)

A cookie is a method for a Web server to maintain state information about users as users navigate different pages

Click to add notes

Slide 4 of 16 | "Opulent"



+12 IA 2:26 / 13:34

PB



AP



TN

R



CONTD..

```
for (i = 0; i < dots.length; i++) {
    dots[i].className = dots[i].className.replace(" active", "");
}

slides[slideIndex - 1].style.display = "block";
dots[slideIndex - 1].className += " active";

// Change image every 2 seconds
setTimeout(showSlides, 2000);
}
```

Click to add notes

COOKIES

(session management)



Click to add notes

COOKIE (SESSION MANAGEMENT)

» A cookie is a method for a Web server to maintain state information about users as users navigate different pages

CUT Copy Format Painter Paste New Slide Delete

Layout Reset Slides

Font Paragraph

Drawing Editing

Find Replace Select

COOKIE (SESSION MANAGEMENT)

- A cookie is a method for a Web server to maintain state information about users as users navigate different pages on the site, and as users return to the site at a later time.

COOKIE (SESSION MANAGEMENT)

- Cookies are little bits of information that you can leave on a user's hard drive, where they stay even after the user leaves your site or turns off the computer, which is extremely useful when you want to remember information about visitors.

HOW COOKIES CAN BE CREATED?

- By client-side script in a HTML page
- Win32 programs that use the Microsoft Win32 Internet functions,
- By serverside script (for example, [ASP] page, Common Gateway Interface, [CGI] script, PHP, ASP .NET etc...)

HOW COOKIES CAN BE CREATED?

- Cookies allow you to give your pages a personal touch. For instance,

Click to add notes

Slide 7 of 16 "Opulent" 83%

+41

AN

D

NB

GG

SM



Akshay Jain

TN

TK



Niraj Nandish



PR

1:46 / 23:03

AYUSH DINESHBHAI MANGIRIA Priyadarshini R

1.00

Cut Copy Format Painter Paste New Slide Reset Delete Slides Font Paragraph Drawing Editing

Slides Outline

COOKIES (SESSION MANAGEMENT)

COOKIES (SESSION MANAGEMENT)

HOW COOKIES CAN BE CREATED?

HOW COOKIES CAN BE CREATED?

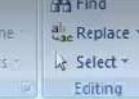
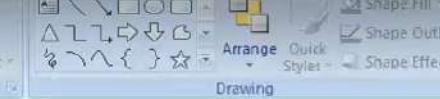
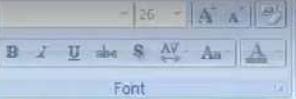
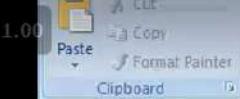
THE MOST IMPORTANT INFORMATION FOR COOKIES ARE:

ANONYMOUS COOKIES

Click to add notes

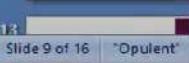
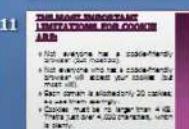
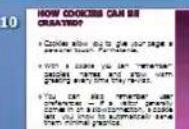
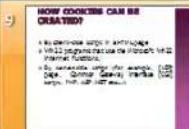
COOKIE (SESSION MANAGEMENT)

- Cookies are little bits of information that you can leave on a user's hard drive, where they stay even after the user leaves your site or turns off the computer, which is extremely useful when you want to remember information about visitors.



Slides

Outline

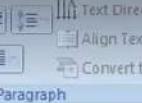
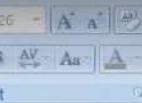
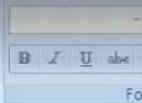
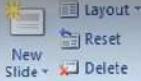


HOW COOKIES CAN BE CREATED?

- ④ By client-side script in a HTML page
- ④ Win32 programs that use the Microsoft Win32 Internet functions,
- ④ By server-side script (for example, [ASP] page, Common Gateway Interface [CGI] script, PHP, ASP .NET etc....)

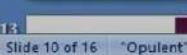
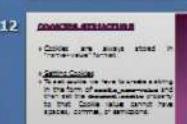
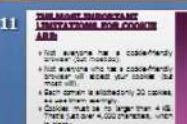
Click to add notes





Slides

Outline



HOW COOKIES CAN BE CREATED?

- ④ Cookies allow you to give your pages a personal touch. For instance,
- ④ With a cookie you can "remember" people's names and show warm greeting every time they re-visit.
- ④ You can also remember user preferences — if a visitor generally comes in on a slow connection, a cookie lets you know to automatically serve them minimal graphics.

Click to add notes

Slide 10 of 16 "Opulent"

83%



1.00

Cut Copy Format Painter Paste New Slide Reset Delete Clipboard Slides Font Paragraph Drawing Editing

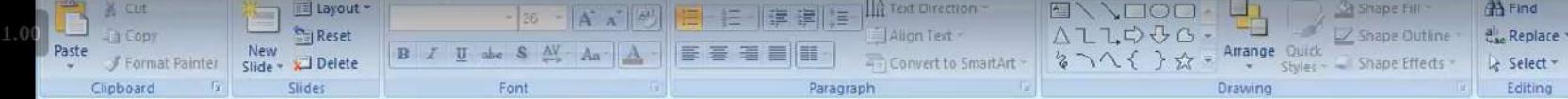
THE MOST IMPORTANT LIMITATIONS FOR COOKIE ARE:

- Not everyone has a cookie-friendly browser (but most do).
- Not everyone who has a cookie-friendly browser will accept your cookies (but most will).
- Each domain is allotted only 20 cookies, so use them sparingly.
- Cookies must be no larger than 4 KB. That's just over 4,000 characters, which is plenty.

Click to add notes

Slide 11 of 16 "Opulent"





COOKIES STRUCTURE

- ⑤ Cookies are always stored in "name=value" format.
- ⑥ Setting Cookies
- ⑦ To set cookie we have to create a string in the form of `cookie_name=value` and then set the `document.cookie` property to that. Cookie values cannot have spaces, commas, or semicolons.

Click to add notes

1.00

Cut Copy Format Painter Paste New Slide Reset Delete Slides Font Paragraph Drawing Editing

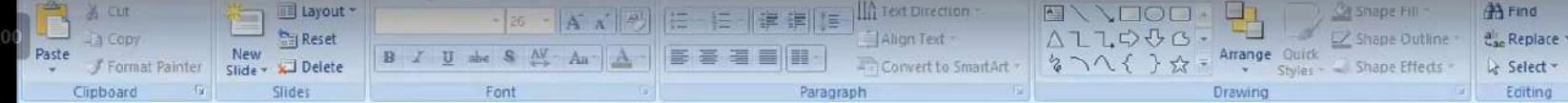
Slides Outline

EXAMPLE

```
<html><head>
<script language=javascript>
function setCookie() {
var the_name=prompt("What's your
name?","");
var the_cookie = "wm_javascript=" +
escape("username:" + the_name);
document.cookie = the_cookie;
alert("Thanks, now go to the next
page."); }
```

Click to add notes

Slide 13 of 16 "Opulent"



Slides

11 **WHAT ARE COOKIES?**

What are cookies? Cookies are small text files stored in your browser. They are used to remember information about you, such as what you have viewed or what you have done online. Cookies can also be used to track your activity.

12 **CREATE A COOKIE**

To create a cookie, you need to create a script that sets the cookie. This can be done using JavaScript. Here's an example:

```
<script>
document.cookie = "name=Akash";
</script>
```

13 **EXAMPLE**

```
<html>
<head>
<script>
function readCookie() {
    var the_cookie = document.cookie;
    var broken_cookie =
        the_cookie.split(":");
    var the_name = broken_cookie[1];
    alert("Your name is: " + the_name);
}
</script>
</head>
<body></body>
</html>
```

14 **DISPLAY A MESSAGE**

```
<html>
<head>
<script>
function readCookie() {
    var the_cookie = document.cookie;
    var broken_cookie =
        the_cookie.split(":");
    var the_name = broken_cookie[1];
    alert("Your name is: " + the_name);
}
</script>
</head>
<body></body>
</html>
```

15 **DISPLAY A MESSAGE**

```
<html>
<head>
<script>
function readCookie() {
    var the_cookie = document.cookie;
    var broken_cookie =
        the_cookie.split(":");
    var the_name = broken_cookie[1];
    alert("Your name is: " + the_name);
}
</script>
</head>
<body></body>
</html>
```

16 **EXAMPLE**

```
<html>
<head>
<script>
function readCookie() {
    var the_cookie = document.cookie;
    var broken_cookie =
        the_cookie.split(":");
    var the_name = broken_cookie[1];
    alert("Your name is: " + the_name);
}
</script>
</head>
<body></body>
</html>
```

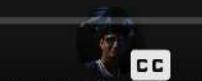
CLICK TO ADD TITLE

```
function readCookie() {
    var the_cookie = document.cookie;
    var broken_cookie =
        the_cookie.split(":");
    var the_name = broken_cookie[1];
    alert("Your name is: " + the_name);
}
setCookie()
readCookie()
</script></head>
<body></body></html>
```

Click to add notes

Slide 14 of 16 "Opulent"

83%



+62

12:39 / 23:03

Akhay Jain

Guru Akash G

Therun K

Niranjan Nandish

AYUSH DINESHBHAI MANGARIA

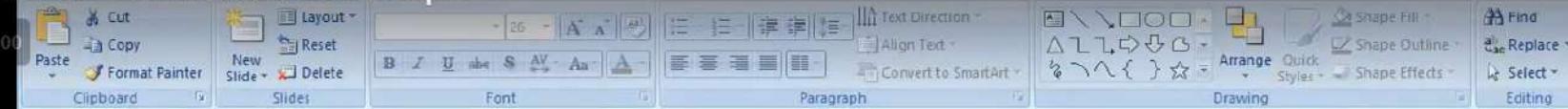
Piyadarshini K



COOKIES STRUCTURE

- The name/value pair must not contain any white space characters, commas, or semicolons. Using such characters can cause the cookie to be truncated or even discarded.
- When you assign a new cookie value to `document.cookie`, the current cookies are not replaced. The new cookie is parsed and its name/value pair is appended to the list. If we use same name, path and domain then previous cookie is replaced by new cookie. The syntax for setting cookies is:

Click to add notes



EXAMPLE

```
<head>
<title>Color Selection</title>
</head>
<body>
<select id="dd">
<option value="Select color">Select color</option>
<option value="red">red</option>
<option value="green">green</option>
<option value="blue">blue</option>
</select>
<script type="text/javascript">
window.onload=function () {
    if (document.cookie.length !=0) {
        var nam=document.cookie.split("=");
        document.bgColor=nam[1];
        document.getElementById("dd").value=nam[1];
    }
}
function setcolor()
{
    var select=document.getElementById("dd").value;
    if (select!="Select color") {
        document.bgColor=select;
        document.cookie="color"+select+";expires";
    }
}
</script>
</body>
```

Click to add notes

CUT COPY Paste Format Painter Clipboard Slides Font Paragraph Drawing Editing

Layout Reset New Slide Delete Slides

Text Direction Align Text Convert to SmartArt

Shape Fill Shape Outline Quick Styles Shape Effects

Find Replace Select

Slides Outline

12 DOCUMENT EXAMPLE

13 DOCUMENT EXAMPLE

14 DOCUMENT EXAMPLE

15 DOCUMENT EXAMPLE

16 DOCUMENT EXAMPLE

17 Click to add notes

CLICK TO ADD TITLE

• Document.cookie="user_name=aaa; expires=mon 12 feb 2021 12:00:00 UTC";

Slide 17 of 17 "Opulent" 83% 21:39 / 23:03

IT254 WTA



The World Wide Web

Basics, Concepts, Protocols

Topics of Coverage

- ▶ *Introduction to the WorldWideWeb:*
 - ▶ History of development
 - ▶ Internet and the WWW
 - ▶ Web Concepts
 - ▶ Web Architecture and Components
 - ▶ Web Protocols.

**What is a
Network?**

What is a Network?

- ❖ **topology**: The physical arrangement of entities in a network.
- ❖ **protocol**: The protocol defines a common set of rules/signals that entities on the network use to interact/communicate.
- ❖ **architecture** : e.g. Direct interaction (*peer-to-peer*) or Indirect interaction (*hierarchy, client/server architecture etc*)

Some early
networks ...

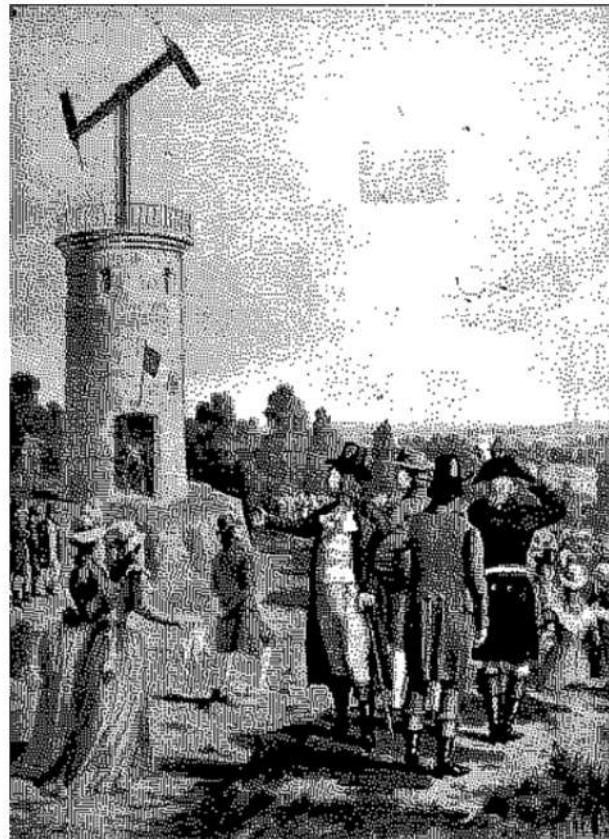
Beacon Chain Networking (400 BC)



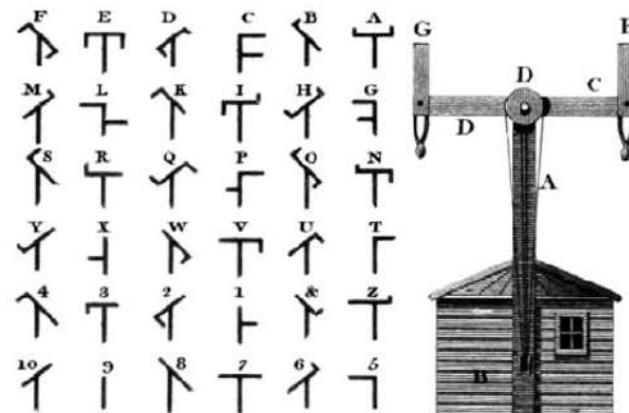
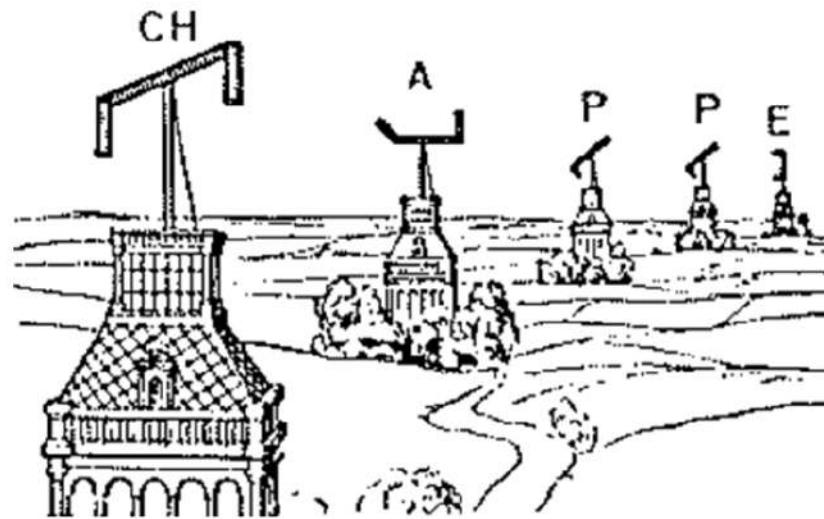
The Watch towers on the Great Wall of China

- Signals used - smoke, fire, drums, coloured flags, gunshots

Chappe's Semaphore Network (1794)

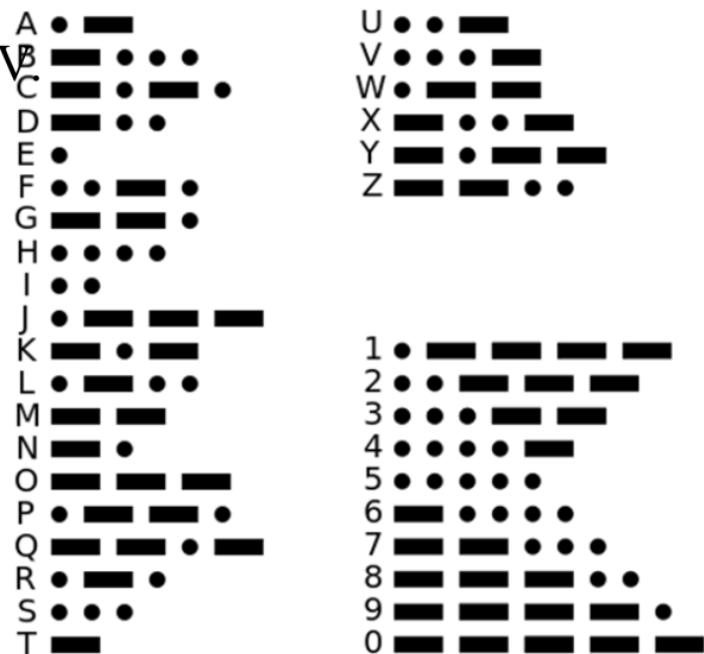


First Line (Paris to Lille),
1794



The ‘Victorian’ Internet (1840s)

- ▶ The Telegraph (1839), Transatlantic Telegraph (1858)
 - ▶ Signals sent over wires that were established over vast distances.
 - ▶ Used Morse Code consisting of dots and dashes (short /long signals)
 - ▶ Electronic signal standard of +/- 15V



Mondothèque (1934)

- ▶ a massive “search engine” envisioned by Paul Oltet.
- ▶ An imaginary device that could be at the same time, an archival, instrument, workstation, catalogue and broadcasting machine.
- ▶ Goal: to collect, organize, and share all the world’s knowledge.



© Patrick Tombelle

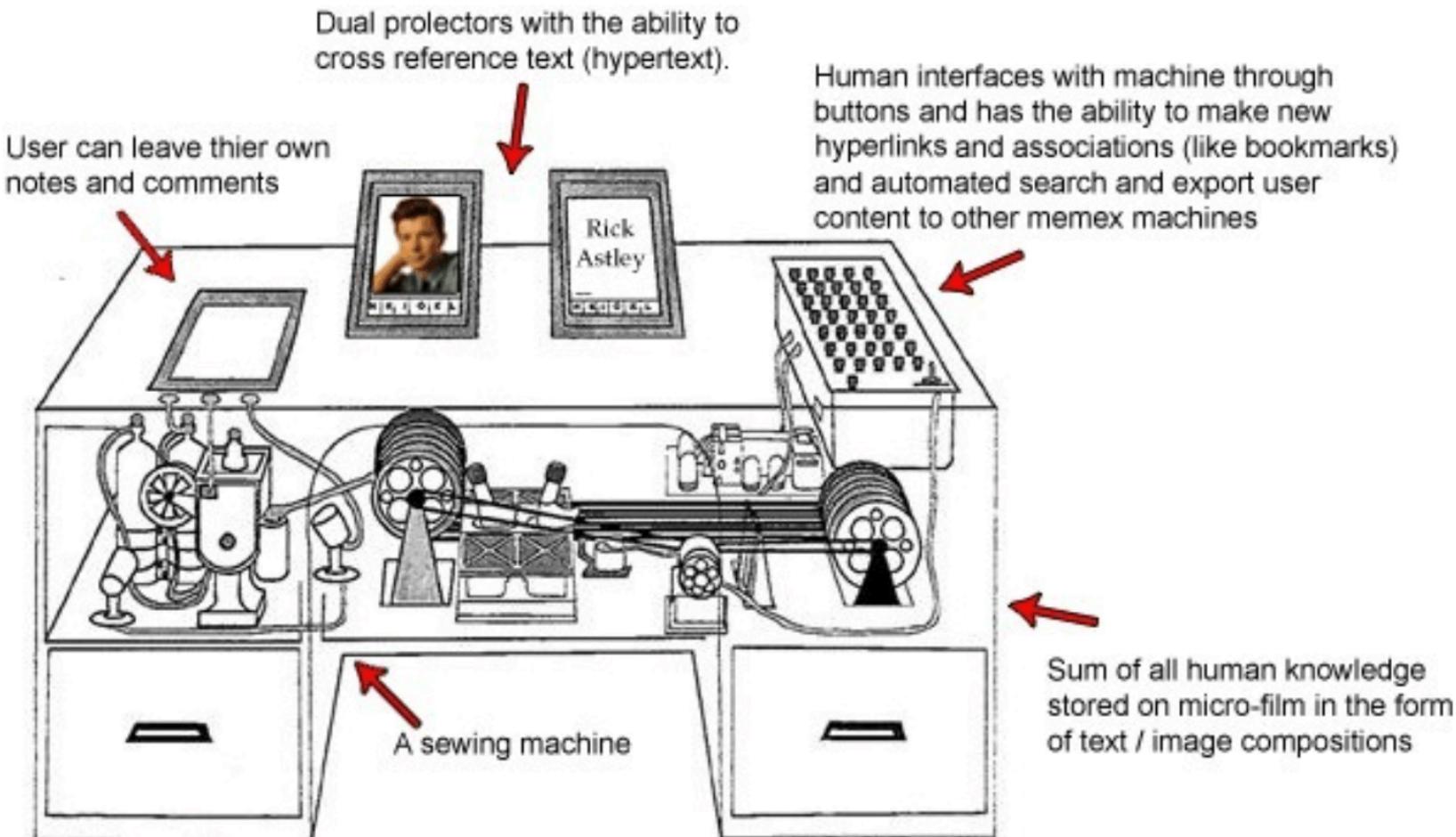
The Memex

(1945)

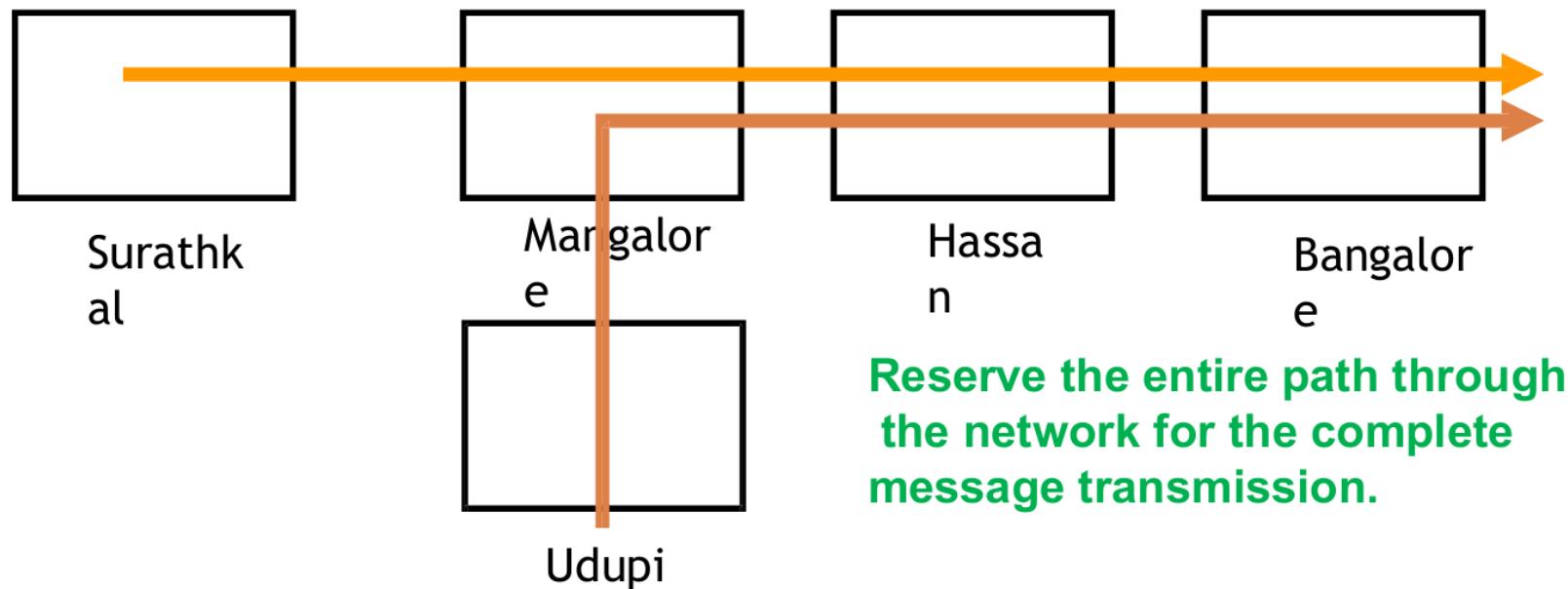
- ▶ hypothetical system that Vannevar Bush described in his 1945 *The Atlantic Monthly* article "As We May Think".
- ▶ Envisioned as a device in which individuals would compress and store all of their books, records, and communications
- ▶ influenced the development of early hypertext systems
 - ▶ eventually leading to the creation of the World Wide Web
 - ▶ personal knowledge base software

The Memex

(1945)

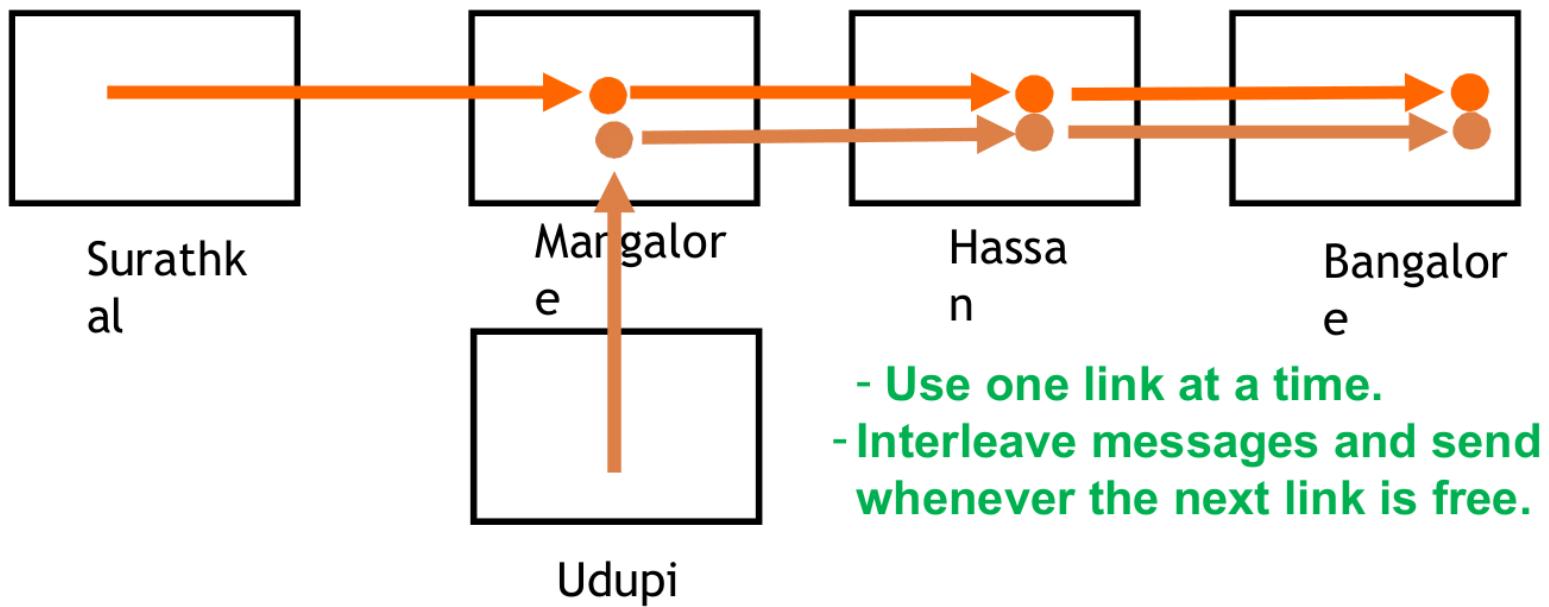


Circuit Switching (1878)



- Example: Landlines (Telephone Network)

Packet Switching (1964)



- Use one link at a time.
- Interleave messages and send whenever the next link is free.

- Example: Modern networks, The Internet

Internet and the WWW

- ▶ Difference
?

Internet is to
WWW

as **NITK** is to
IT

- ▶ **Internet** - an infrastructure of millions of cables & computers, or several smaller sub-networks that share these cables and computers.
- ▶ **The WWW (or simply the *Web*)** - the largest and most popular sub-network on the internet.

Internet and WWW Development Forums

- ▶ Standards and specifications for the design of the Internet
 - *Internet Engineering Task Force (IETF).*
- ▶ Web standards like HTML, CSS, XML, RDF were introduced and standardized - *WorldWideWeb Consortium (W3C)*
- * *IETF and W3C are open forums, allowing anyone interested in contribution to participate in the policy making/standardization process.*

Design Principles of the Internet and WWW

- ▶ **Interoperability/Universality:**
 - ▶ Different implementations of Internet Protocols actually work together.
 - ▶ Adoption of open standards to facilitate interoperability.
 - ▶ Systems can be assembled using client/server computers and software from different vendors.
- * For applications like e-commerce, buyers and sellers do not have to change/buy/upgrade software or systems to do business with each other.

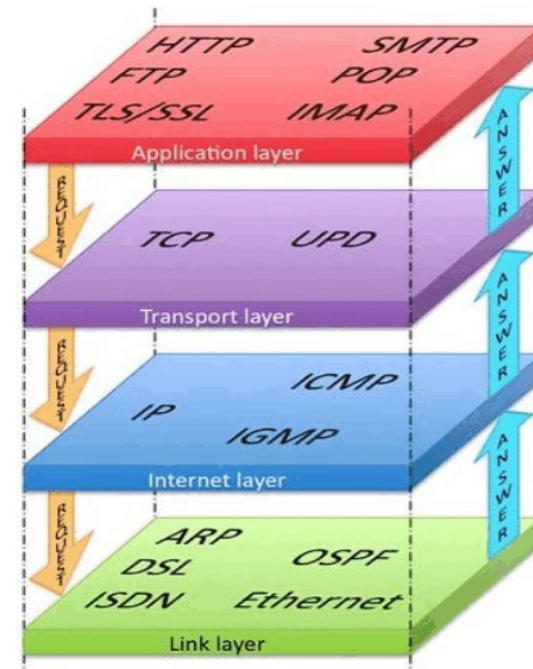
Design Principles of the Internet and WWW

Layering

- Internet protocols are designed to work in layers, higher layers building on the facilities provided by the lower layers.

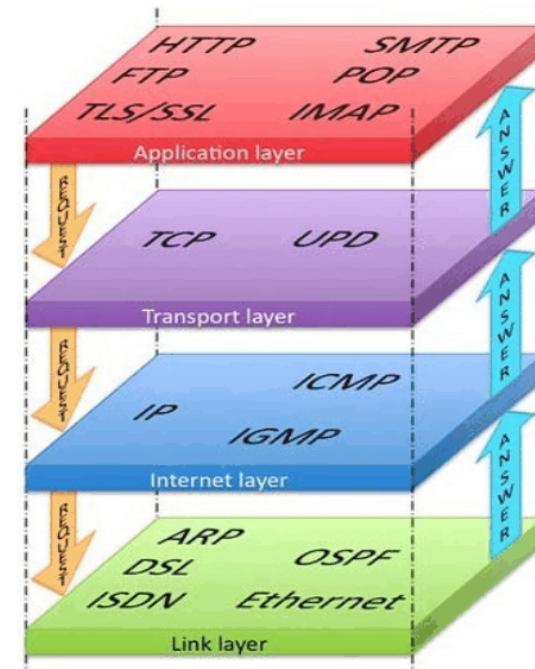
E.g.

- TCP builds on IP to create reliable byte streams
- Application layer protocols such as email build on TCP capabilities.



Design Principles of the Internet and WWW

- ▶ **Simplicity:**
 - ▶ Layering of the Internet simplifies application development.
 - ▶ IP hides the complexities of the layers below it.



Design Principles of the Internet and WWW

- ▶ **Uniform Naming and Addressing:**
 - ▶ **IP addressing:** Use of the dotted decimal form, assigning a 32/64 bit address to each computer connected to the network.
 - ▶ **DNS** - a standard way to translate human readable names for computers.
 - ▶ **URI** - a standard way to link and locate resources on the Web

Design Principles of the Internet and WWW

► End to End:

- The internet/Web is concerned only with the transmission of data, not its interpretation.
- Interpretation of data happens on the sending & receiving systems (computer/browser), not on the network.

* Analogy: Mailing a letter.

Design Principles of the Internet and WWW

► 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”!
- freedom from indiscriminate censorship and surveillance.*

* No longer holds good.

Design Principles of the Internet and WWW

► 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.
- Also known as **Net Neutrality**.

WEB SYSTEM ARCHITECTURE

WEB SYSTEM ARCHITECTURE

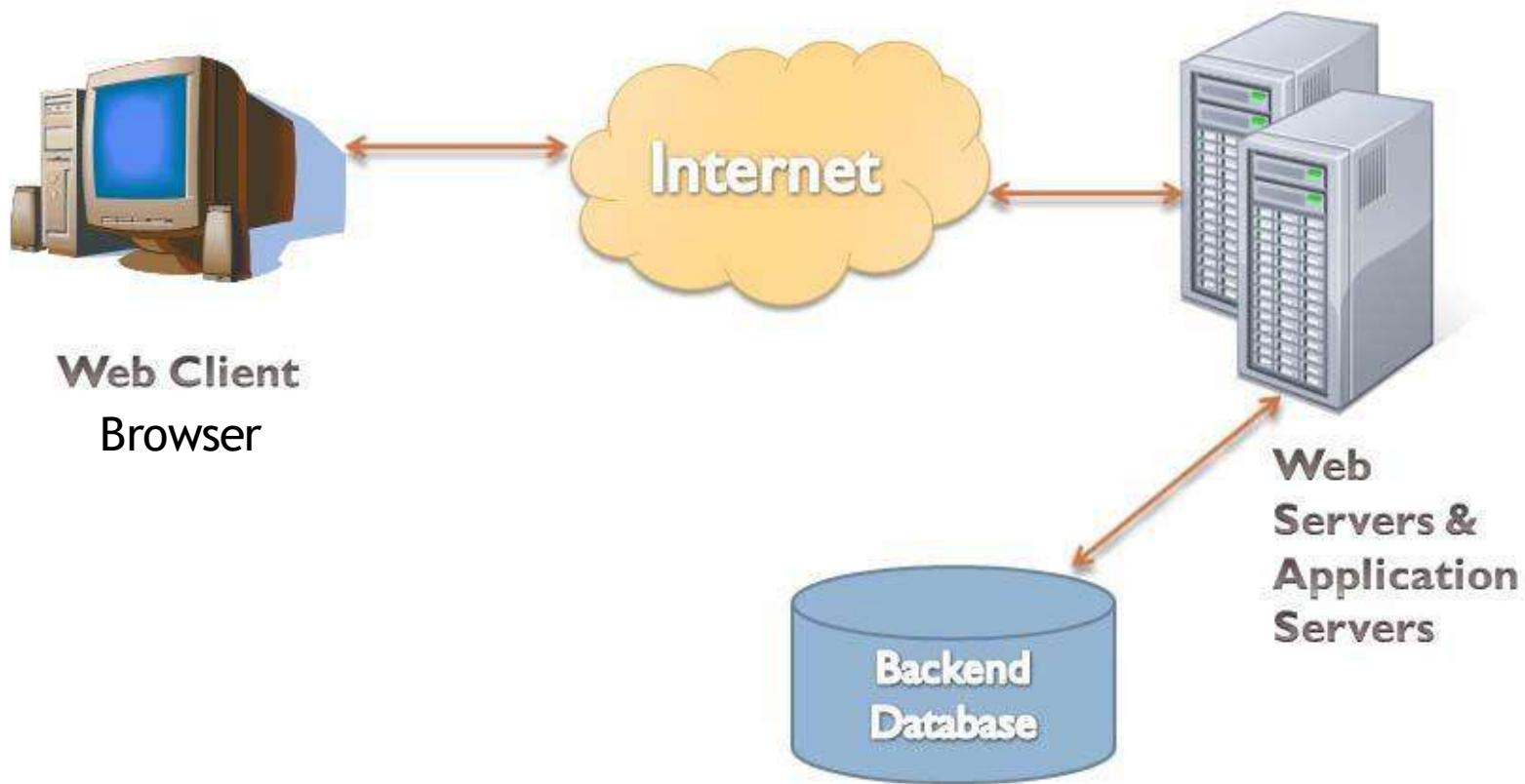


Fig: Basic Components of a Web based system

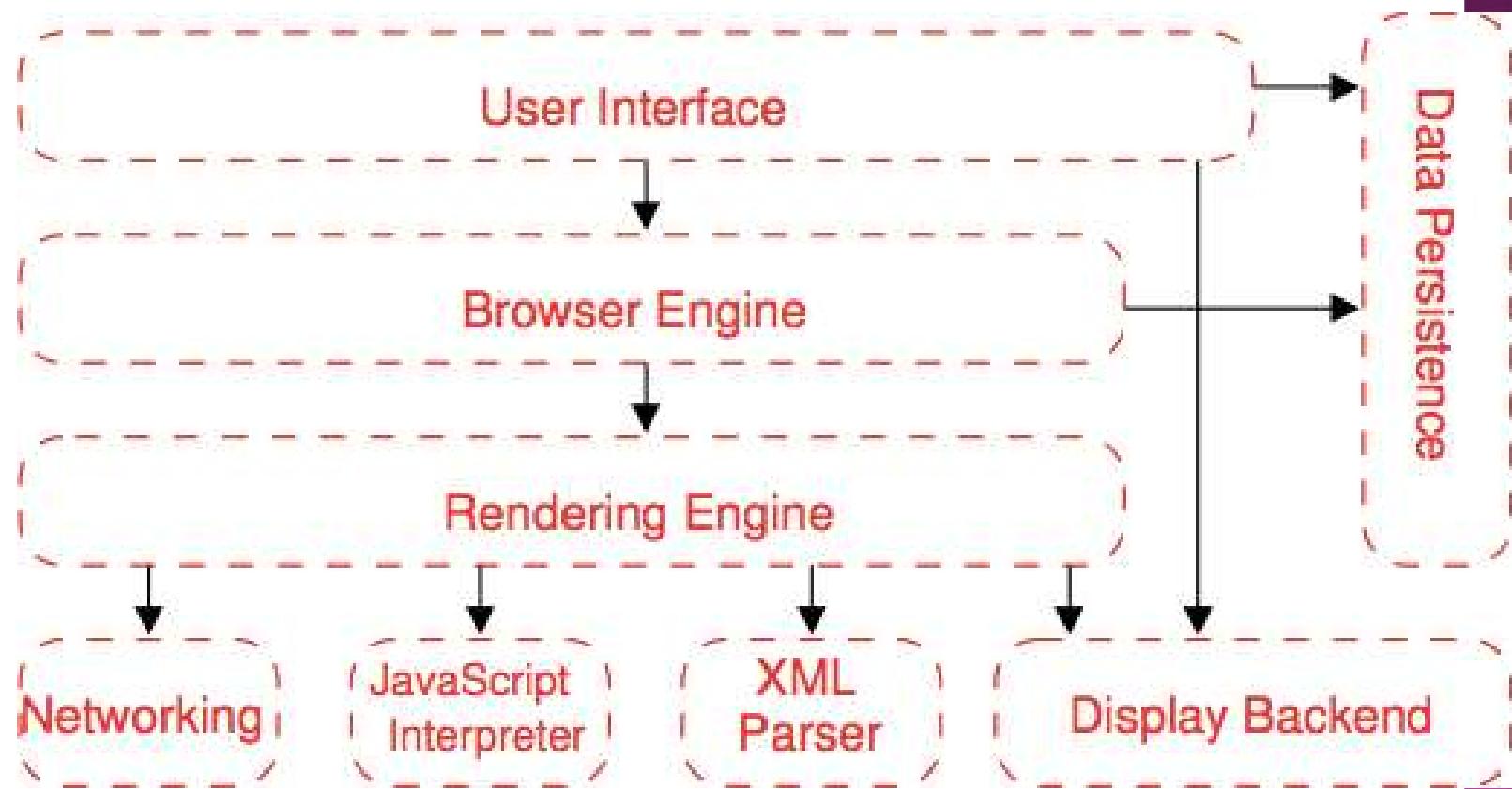
WEB SYSTEM COMPONENTS (CONTD.) WEB CLIENTS

- Types of web clients may vary depending on the application.

For.e.g.–

- Web browsers (PC, mobile, text-only browsers, voice browsers)
- Chat browsers/interfaces
- Software robots (*no direct user contact*)
- Software agents on the Web (*initiated with user action*)

WEB SYSTEM COMPONENTS (CONTD.) WEB BROWSER



WEB SYSTEM COMPONENTS (CONTD.) WEB CLIENTS

Basic tasks to be handled by a browser

- Reformat the URL entered as a valid HTTP request message.
- Use DNS to convert the host name to the appropriate IP address.
- Establish a TCP connection using the IP address of the specified web server.
- Send the HTTP request over TCP connection and wait for the server's response.
- Display document contained in the response. (e.g. direct display of plain text, rendering HTML pages etc.)

WEB SYSTEM COMPONENTS (CONTD.) WEB CLIENTS

- Some important additional functionalities provided by modern browsers
 - Automatic URL completion (...data persistence)
 - Script Management
 - Event Handling
 - Management of form GUI
 - Secure Communication
 - Session/Cookie Management
 - Handling extension mechanisms

EXTENSION MECHANISMS FOR THE WEB CLIENT

- Mechanisms that add additional capabilities to the browser, either automatically or by user intervention.

- Types -
 - a. MIMETypes or Internet Media Types
 - b. Plug-ins
 - c. Add-ons
 - d. Scripts
 - e. Applets
 - f. Controls

EXTENSION MECHANISMS FOR THE WEB CLIENT

a. MIME Types or IM Types

- **Multipurpose Internet Mail Extensions/Internet media type**
 - standard identifier used on the Web/Internet to indicate the type of data that a file contains.
 - Common uses include :
 - In web browsers - how to display or output files that are not in HTML format
 - In search engines - to classify data files on the web.
 - In email clients - to identify attachment files.

EXTENSION MECHANISMS FOR THE WEB CLIENT

a. MIME Types or IM Types (contd.)

- Each document is tagged with a *type* to identify what kind of resource it is.
- Format – *class/subclass*
- E.g. *text/html*, *image/gif*, *application/pdf*, *audio/mp3* etc.

EXTENSION MECHANISMS FOR THE WEB CLIENT

a. MIME Types or IM Types (contd.)

| file type | MIME type |
|-----------|---------------------|
| avi | video/x-msvideo |
| bmp | image/bmp |
| css | text/css |
| doc | application/msword |
| dtd | application/xml-dtd |
| dvi | application/x-dvi |
| gif | image/gif |
| html | text/html |
| ico | image/x-icon |
| midi | audio/midi |
| mov | video/quicktime |
| mp3 | audio/mpeg |
| mpeg | video/mpeg |
| pdf | application/pdf |

Complete List maintained by **Internet Assigned Numbers Authority (IANA)**

<http://www.iana.org/assignments/media-types/media-types.xhtml>

EXTENSION MECHANISMS FOR THE WEB CLIENT (CONTD.)

b. Plug-ins

- Allow adding new capabilities for handling third party software in the browser itself rather than launching a separate application.
- Applications provide plug-ins to –
 - support easy adding of new features to browsers.
 - enable third-party developers to provide abilities to handle their formats in the native browser.
 - separate source code from an application because of incompatible software licenses.

EXTENSION MECHANISMS FOR THE WEB CLIENT

B. PLUG-INS (CONTD.)

□ Features -

- Must be manually installed by user before new data type can be used.
- Browser plug-ins can modify the behavior of the browser. (e.g. adding new toolbar commands, menu items etc.)
- For e.g. Quicktime player, Adobe Reader, Macromedia Flash etc.

CLIENT

B. PLUG-INS (CONTD.)

The screenshot shows the Firefox Add-ons Manager window. On the left sidebar, there are links for "Get Add-ons", "Extensions", "Appearance", "Plugins", and "Services". The main pane displays a list of installed plugins:

- Adobe Acrobat 11.0.6.70**
Adobe PDF Plug-In For Firefox and Netscape 11.0.06 [More](#) Always Activate
- Google Talk Plugin 5.1.4.17398**
Version 5.1.4.17398 [More](#) Always Activate
- Google Talk Plugin Video Accelerator 0.1.44.29**
Google Talk Plugin Video Accelerator version:0.1.44.29 [More](#) Always Activate
- Google Talk Plugin Video Renderer 5.1.4.17398**
Version 5.1.4.17398 [More](#) Always Activate
- Google Update 1.3.22.3**
Google Update [More](#) Always Activate

A warning message at the bottom states: **Java Deployment Toolkit 7.0.250.17 is known to be vulnerable. Use with caution.** [More Information](#)

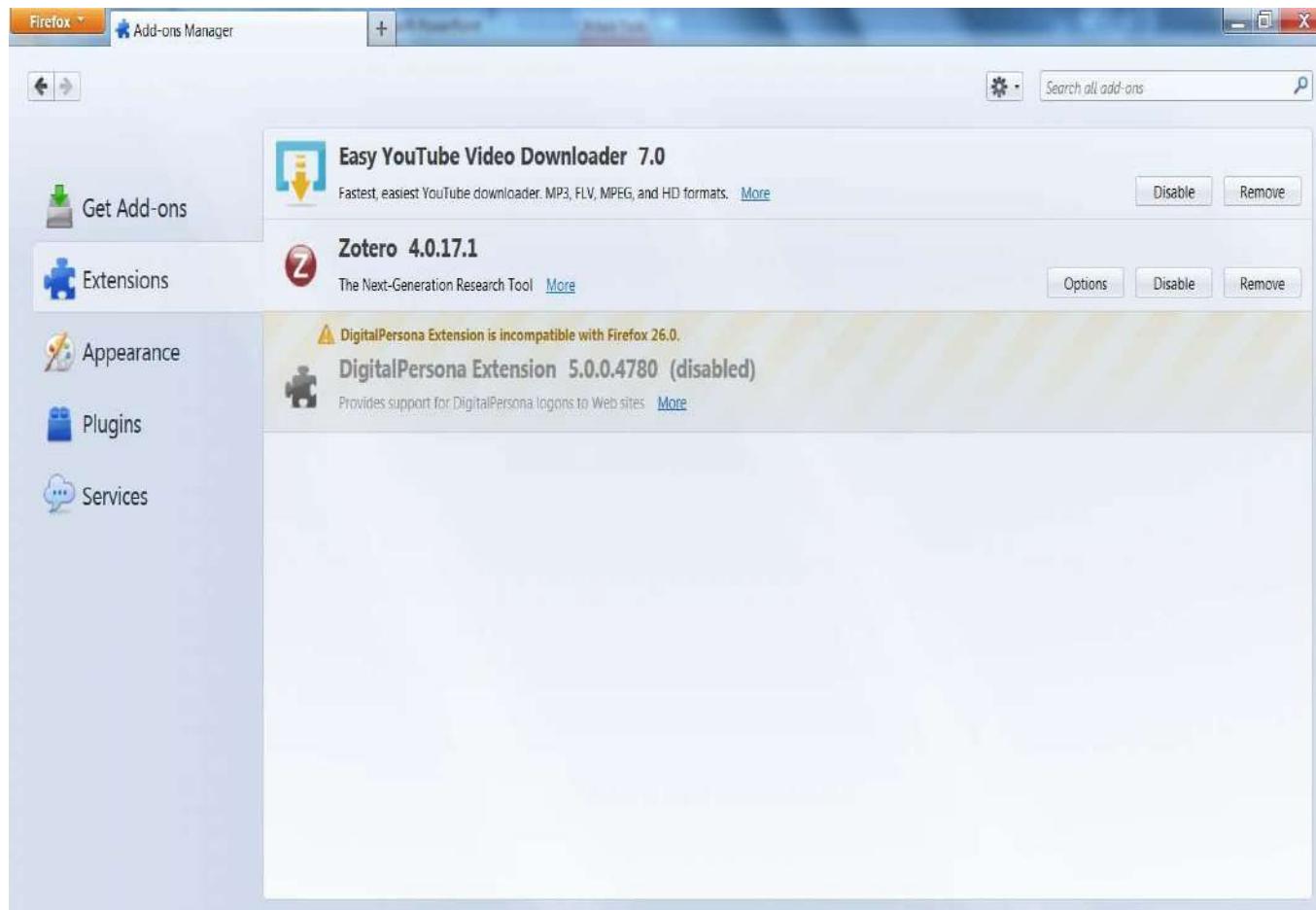
- Java Deployment Toolkit 7.0.250.17 10.25.2.17**
NPRuntime Script Plug-in Library for Java(TM) Deploy [More](#) Ask to Activate
- McAfee Security Scanner + 3.8.130.0**
McAfee MSS+ NPAPI Plugin [More](#) Always Activate
- Microsoft Office 2010 14.0.4761.1000**

EXTENSION MECHANISMS FOR THE WEB CLIENT

c. Add-ons

- used to refer to features that enhance an application.
 - Types – *extensions, themes and skins*.
- *An extension add-on* tailors the *core* features of an application by adding an optional module.
- *Theme or skin add-on* tailors the *outer* layers of an application to personalize functionality.

C. ADD-ON (CONTD.)



EXTENSION MECHANISMS FOR THE WEB CLIENT (CONTD.)

d. Scripts

- Executable scripts can be embedded in web pages.
 - run when encountered on a page or when specified events occur.
 - written in languages like JavaScript, VBScript, ActionScript etc, and are executed by an interpreter in the browser when page is displayed.
- Scripts can modify page display and increase interactivity of the page, but have limited power. (for security reasons)

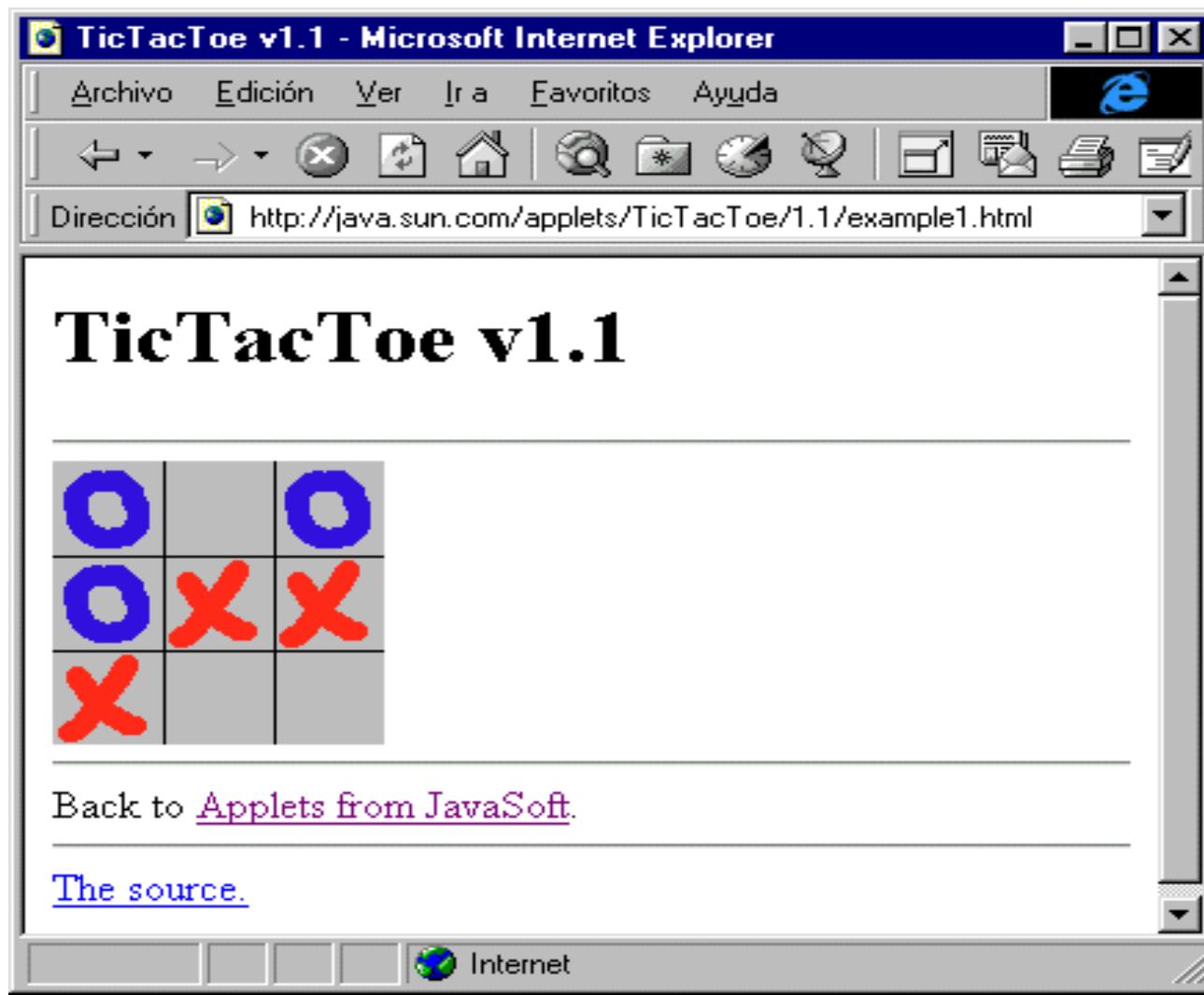
EXTENSION MECHANISMS FOR THE WEB CLIENT (CONTD.)

e. Applets

- Java applets are downloaded on demand from a server.
 - Used to create animation effects and other interactive behavior in the browser.
 - Are executed in the Java Virtual Machine supplied by the browser, thus limiting its effect on the system.
- User experience may be affected as download time can be significantly higher than that of scripts.

CLIENTS

E. APPLETS (CONTD.)



EXTENSION MECHANISMS FOR THE WEB CLIENT (CONTD.)

f. Controls

- Are software modules that are automatically downloaded and installed when a webpage containing them is encountered.
- On future references, it is automatically activated without having to be downloaded again.
 - Contain compiled code that can make changes to your machine.

EXTENSION MECHANISMS FOR THE WEB CLIENT (CONTD.)

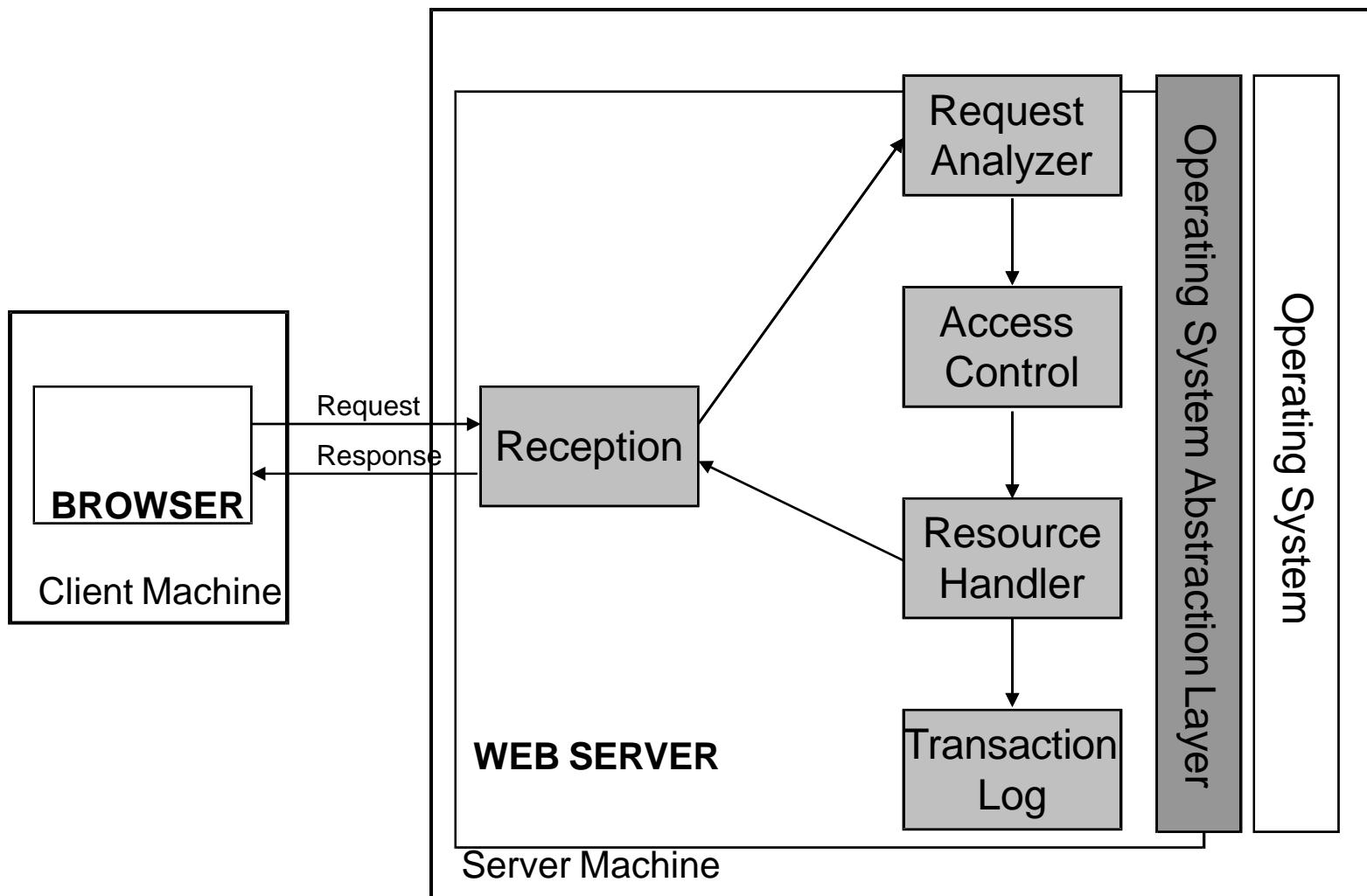
f. Controls

- Features -
 - Controls have full system control, hence user needs to allow only trusted providers.
 - Each control is digitally signed by its authoring organization. (Code Signing)
- Disadvantage:
 - If user decides not to install the control, then user experience with website may be disrupted.
 - E.g. Flickr Photo Uploader, software download controls, IE, MS-Office.

WEB SERVERS

- Basic functionality of a web server:
 - Accept HTTP requests from web clients and return an appropriate resource (if available) in the HTTP response.
 - Functionalities provided –
 - Communicating with TCP.
 - Handling multiple incoming requests and their corresponding responses.
 - Identifying resource location based on request URL.
 - Session Management.

WEB SERVER ARCHITECTURE



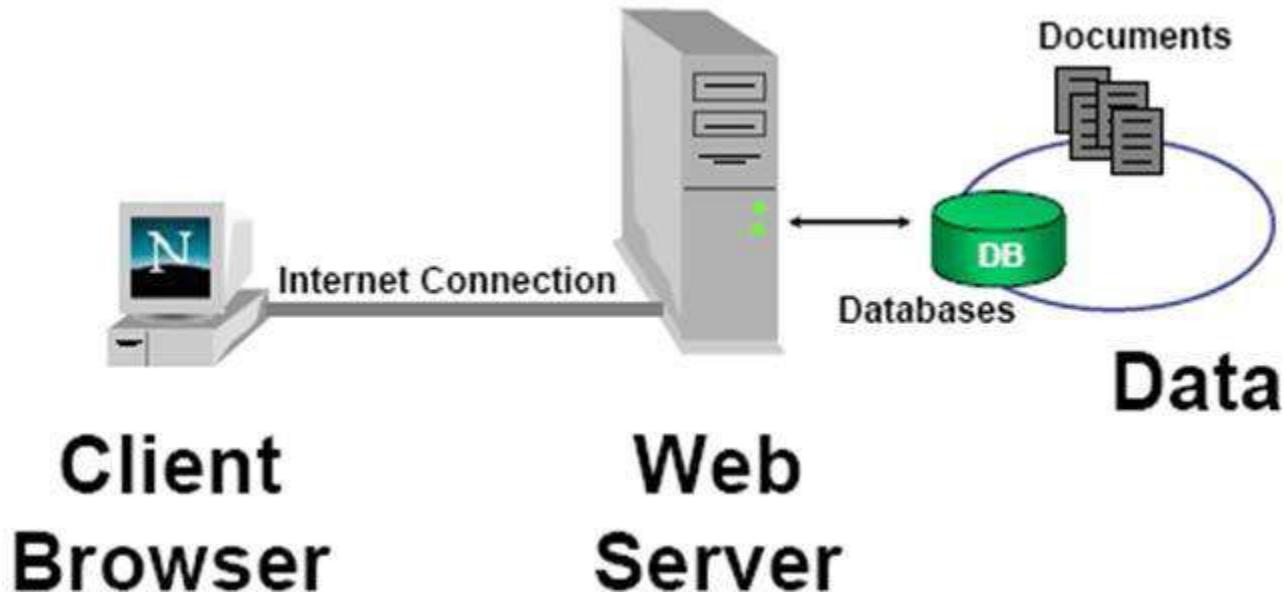
WEB SERVERS

- *HTTPd* web server was the very first web server implementation.
(developed by NCSA)
- *HTTPd* became the starting point for the free, open source Apache Server (April 1995)
- Microsoft's Internet Information Server (IIS) offers all the features of Apache.
 - IIS runs only on Windows systems, while Apache supports Windows, Unix and Mac Systems.
- Others – nginx, GWS

APPLICATION S_ER_VE_RS

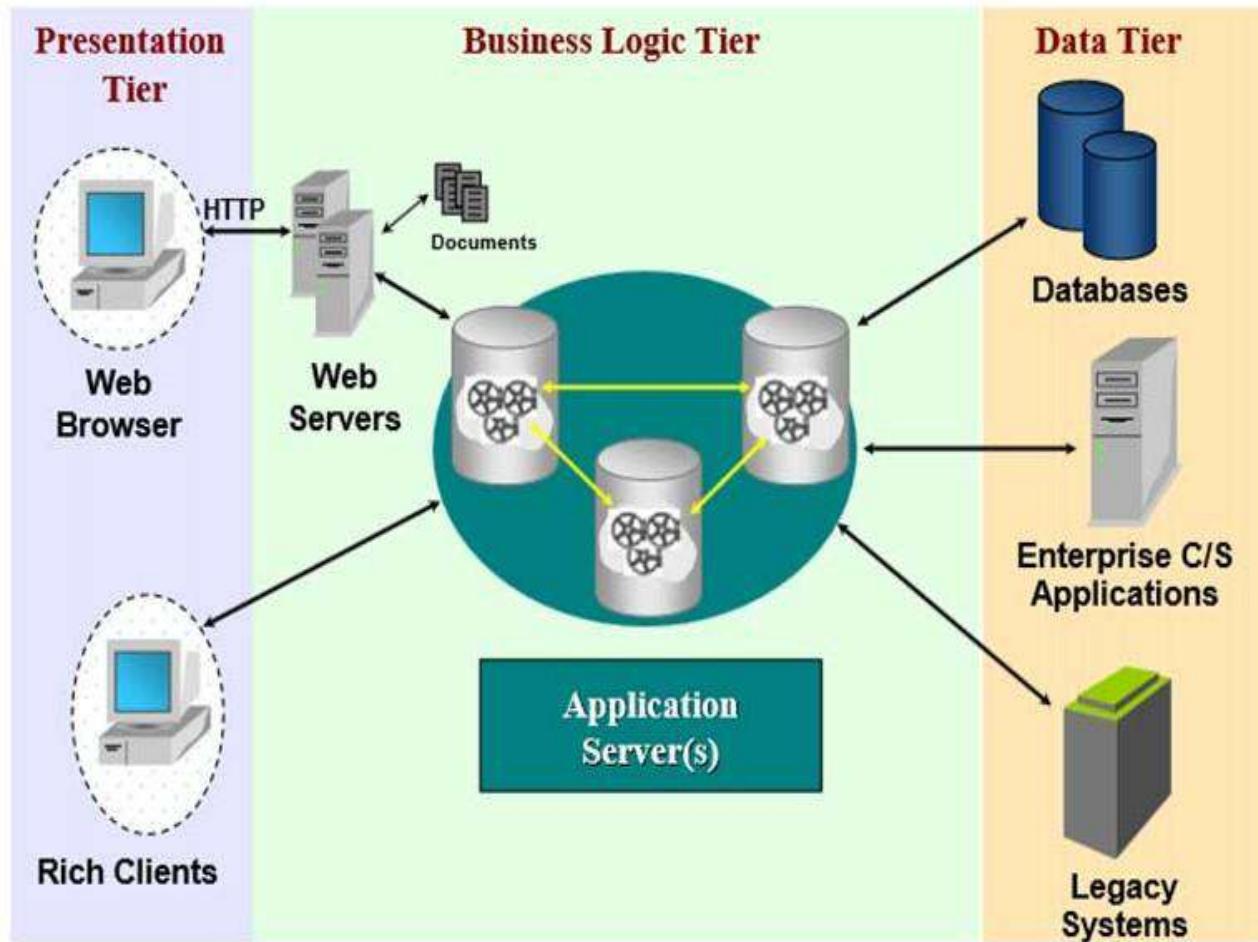
- Application Servers are middleware for Web Applications.
 - Used for connecting remote clients with applications over Internet and effectively integrating applications.
- provides middleware logic - for e.g. for transactions, security, data persistence, heterogeneous clients for complex Web Systems.
 - The goal is to provide an environment for hosting all kinds of application logic:
 - Can be used for EAI as well as Web-based integration.

BASIC WEB APPLICATIONS



ENTERPRISE WEB APPLICATIONS

APPLICATION SERVER(S) AS WEB MIDDLEWARE



APPLICATION SERVER SUPPORT FOR PRESENTATION LAYER

- provides extensive support for client-side interaction. A typical app server can support -
 - Web browsers
 - Applications and Devices
 - Chat clients
 - Mobile clients
 - E-mail programs
 - Web services clients
- Presentation logic support includes
 - Multi-device content delivery
 - Servlets, JSPs, XMLsupport, etc.
 - Personalization logic

APPLICATION SERVER FUNCTIONS

- Flexibility and Scalability
 - infrastructure for all types of e-business activities.
- Universal Business Server
 - Provides a dynamic, Web-enabled environment - scales applications, balances loads, manages transactions.
- XML Server
 - Provides the ability to dynamically exchange/modify XML documents externally, or internally as per user request.

APPLICATION SERVER FUNCTIONS (CONTD.)

- Universal Listener Framework
 - Monitors server ports to identify the presence and protocol of an incoming message.
- Application Manager
 - Agent-based management component providing real-time performance and status information.
- Security Console
 - User, group and role-based access control to every system level.

APPLICATION SERVER FUNCTIONS (CONTD.)

- Fault-Tolerance
 - Customer-Facing Fault Tolerance
 - Ensuring that software/hardware system failures or upgrades don't adversely affect users.
- Fast Fail-over
 - Speed-up application recovery
- State Management
 - Storing State information (session, user activities)

WEB SYSTEM COMPONENTS (CONTD.)



Backend System

- Supports the service system by fulfilling the user's request.
- In many cases, this is a Database Management System.

Internet

- The communication platform for web server and web client.
- Web client and web server are not connected directly, hence use a protocol (HTTP) to communicate with each other.

MORE READING...

- History of the Internet: by Gregory Gromov —
 - http://www.netvalley.com/cgi-bin/intval/net_history.pl
- Architecture of the World Wide Web (W3C)
<https://www.w3.org/TR/webarch/>

Web Protocols

HTTP 1.0 / 1.1

HTTP 0.9: THE ONE-LINE PROTOCOL

- ⌚ The original HTTP proposal by Tim Berners-Lee (1989).
- ⌚ High-level design goals
 - ⌚ file transfer functionality
 - ⌚ ability to request an index search of a hypertext archive
 - ⌚ format negotiation.
 - ⌚ an ability to refer the client to another server

HTTP 0.9: THE ONE-LINE PROTOCOL

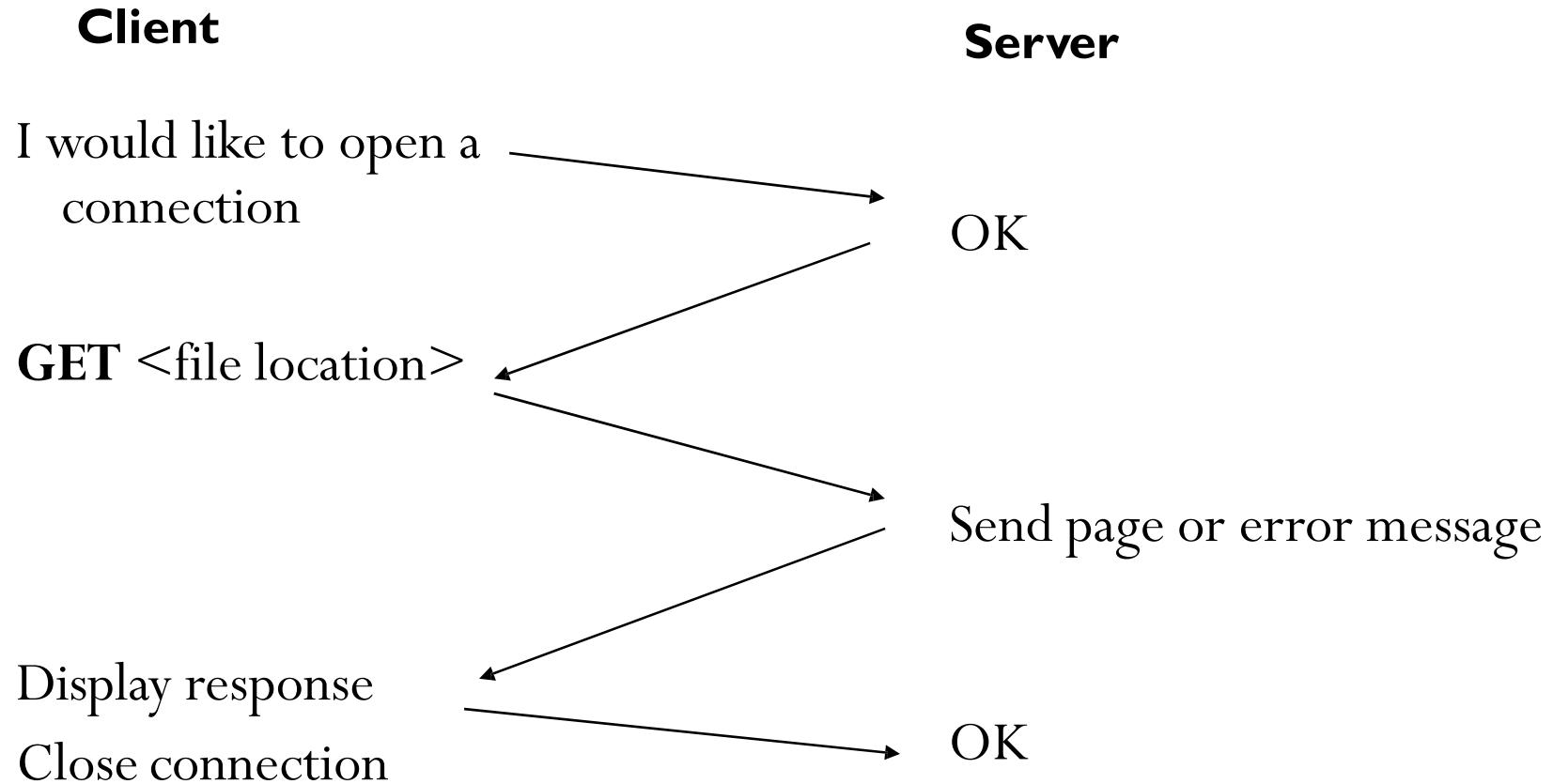
- ⌚ Implementation:
 - ⌚ Client request is a **single ASCII character string**
 - ⌚ terminated by a carriage return (CRLF).
 - ⌚ Server response is **an ASCII character stream.**
 - ⌚ Server response is in **hypertext only**
 - ⌚ Connection **is terminated after the document transfer** is complete.

HYPER TEXT TRANSFER PROTOCOL (HTTP)

- Works in a client/server computing environment.
 - Runs on top of TCP on the standardized port 80;
 - Stateless
 - Asynchronous

- Based on *message based interoperability* (request-response model).
 - Request is specified in ***text (ASCII*** format
 - response is in ***MIME / IMT format***

A HTTP CONVERSATION



HTTP is the set of rules governing the format and content of the conversation between a Web client and server

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

HOW HTTP WORKS

- Address entered - www.example.org
- Browser functions –
 - Creates a message conforming to HTTP protocol (**HTTP Request**),
 - Uses DNS to obtain the IP address for www.example.org,
 - Creates a TCP connection with the machine at this IP address,
 - Sends the HTTP request message over this TCP connection.
 - Server sends back result (**HTTP Response**).
 - Browser deciphers the response and creates a display of the information in the client area of the browser.
(Rendering)

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

HTTP Request: General format of client request.

Syntax

Request_method Resource_address HTTP/version-no

General_header(s)

Request_header(s)

Entity_header(s)

----- *blank-line* -----

Message body (Additional Data) (optional)

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP REQUEST (LINE 1)

- ⌚ Basic Request methods (introduced in HTTP 1.0)
 - ⌚ **GET** - *retrieve a webpage reached by the given Request-URL*
 - ⌚ **POST** - *posts additional data to the web server appending it to the HTTP request msg.*
 - ⌚ Data is attached after the headers.
 - ⌚ **HEAD** - *requests the header info of the webpage for a given Request-URL*

| Property | GET | POST |
|-----------------------------|--|--|
| BACK button/ Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

- Additional request methods introduced in HTTP 1.1 and beyond...
 - **PUT** – request that the enclosed entity be stored under the Request-URL
 - **DELETE** – request that the server delete resource identified by Request-URL
 - **OPTIONS** – request for information about communication options.
 - **TRACE** – invoke a remote, application-layer loopback of request message
 - **CONNECT** – used by proxies in SSL connections.
 - **PATCH** – applies partial modifications to a resource

COMPARING PUT AND POST

| PUT | POST |
|--|--|
| Replacing existing resource or Creating if resource is not exist <i>http://www.example.com/customer/{id}</i> <i>http://www.example.com/customer/123/orders/456</i> Identifier is chosen by the client | Creating new resources (and subordinate resources) <i>http://www.example.com/customer/</i> <i>http://www.example.com/customer/123/orders</i> Identifier is returned by server |
| Idempotent i.e. if you PUT a resource twice, it has no effect. Ex: Do it as many times as you want, the result will be same. $x=1;$ | POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Ex: $x++;$ |
| Works as specific | Works as abstractive |
| If you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call. | Making two identical POST requests will most-likely result in two resources containing the same information. |

SAFE AND IDEMPOTENT PROPERTIES

Safe HTTP method – method that does not modify resources.

Idempotent HTTP method – can be called many times without any change in associated outcomes.

| HTTP Method | Idempotent | Safe |
|-------------|------------|------|
| OPTIONS | yes | yes |
| GET | yes | yes |
| HEAD | yes | yes |
| PUT | yes | no |
| POST | no | no |
| DELETE | yes | no |
| PATCH | no | no |

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

HTTP REQUEST (LINE 1)

- ***Resource_address:*** is the URL that specifies location of the requested resource on web server.

- ***HTTP/version-number:*** Tells the web server what HTTP protocol the web client is using (1.0/1.1/2)

HTTP REQUEST - HTTP HEADERS

- ⌚ Used for passing additional info to the web server.

- ⌚ Three different types:
 - ⌚ General Header
 - ⌚ Request Header
 - ⌚ Entity Header

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP REQUEST - ***HTTP HEADERS***

1. General Headers

- ☛ *Date* - specifies when message is generated (*date and time*).

- ☛ *Pragma* - used for specifying freshness of resource to the server and to the intermediaries like proxy servers.

e.g. **pragma : no-cache** (*in HTTP1.0*)

Cache-Control: no-cache (*in HTTP 1.1 and later*)

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

HTTP REQUEST - HTTP HEADERS

2. Request Headers

- *Authorization* - provides authorization info to web server.
- *From or host* - provides information about the source of the HTTP message.
- *If-modified-since* - asks the web server to provide requested resource only if it has been modified since the specified time in the header.
- *Referer* - the URL from which the client arrived at the resource.
- *UserAgent* - provides information on user agent (browser) used by web client.

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP REQUEST - *HTTP HEADERS*

2. Request Header (*contd.*)

- some optional additional fields:
 - *Accept* - MIME types of resources accepted by browser
 - *Accept-Charset* - charset accepted by browser
 - *Accept-Encoding* - encoding accepted by browser
 - *Accept-Language* - language accepted by browser

HTTP 1.1
additions

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP REQUEST - *HTTP* *HEADERS*

- ***Entity headers:*** define meta-information about the entity-body or, if no body is present, about the resource identified by the request.
 - ***Allow*** - indicates request methods allowed.
 - ***Content_encoding*** - specifies compression method applied to content.
 - ***Content_length*** - indicates length of content in no. of octets.
 - ***Content-range*** - *for range requests (HTTP 1.1 only)*
 - ***Content_type*** - indicates MIME type of content.
 - ***Expires*** - indicates date and time of content expiry.
 - ***Last_modified*** - indicates when webpage was last modified.

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP REQUEST (LAST LINE)

• AdditionalData:

- web client can post additional data to the server after blank line.
- In case of POST/PUT/DELETE requests

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP REQUEST

Simple Example for a HTTP Request –

GET /index.html HTTP/1.1

host: www.nitk.ac.in

user-agent: Mozilla/5.0 (windows; ...)

accept: text/html, image/gif, image/jpeg, */*

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

- Real-world Example for a HTTP Request -

GET /depts HTTP/1.1

Host: www.nitk.ac.in

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.3) Firefox/3.5.3

Accept: text/html, application/xhtml+xml, application/xml, text/* , */*

Accept-Language: en-us, en-us, en; q=0.5

Accept-Encoding: gzip, deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7

Connection: keep-alive //only in HTTP 1.1 and HTTP/2

Keep-Alive: 300

Allow: GET, HEAD, POST

**Cookie: PREF=ID=141ca2d1746b4:U=f22e9e944a56f:FF=4:LD=en:NR=10:CR=2:TM=124956
7334:LM=1251146058:GM=1:S=qWowBrte7hrXniGp;
NID=27=n9Khexo85YHnovw93wK4qC2IZtGa1DnzVQEB6iul9tn62fsJ7gFuMVK252ceLC
D3iS54r-nHD6kWDdD1JP77akDhMI0EWzoPt3cM5g8mapG9SskdRSyEyLWcJK1LrX**

Cache-Control: max-age=0 //only in HTTP 1.1 and HTTP/2

HYPER TEXT TRANSFER PROTOCOL (CONTD.)

HTTP RESPONSE

- General format of the Web server's response to a HTTP request.

Syntax

HTTP/version-number status-code result-message (status line)

General_header(s)

Response_header(s)

Entity_header(s)

----- *blank-line* -----

Entity body (optional)

HYPER TEXT TRANSFER PROTOCOL (CONTD.) HTTP RESPONSE (LINE 1)

C Status Code:

C indicates the result of the request. (e.g. not found, unauthorized, O

| Code | Class | Standard Use |
|------|---------------|---|
| 1xx | Informational | Provides information to client before request processing is completed |
| 2xx | Success | Request has been successfully completed |
| 3xx | Redirection | Client needs to use a different resource to fulfil request |
| 4xx | Client Error | Client's request is not valid |
| 5xx | Server Error | An error occurred during server processing of a valid client request. |

| Code | Meaning |
|------|-----------------------|
| 200 | OK |
| 201 | Created |
| 204 | No Content |
| 206 | Partial Content |
| 301 | Moved Permanently |
| 302 | Moved Temporarily |
| 400 | Bad Request |
| 401 | Unauthorised |
| 403 | Forbidden |
| 404 | Not Found |
| 500 | Internal Server Error |
| 502 | Bad Gateway |
| 504 | Gateway Timeout |

- ⌚ ***Accept, accept-charset, accept-language, accept-encoding*** (*same as in HTTP request*)
- ⌚ ***Accept-Ranges*** - server indicates its acceptance of range requests/byte serving for resource. (in HTTP 1.1)
- ⌚ ***Age*** - time elapsed since response was generated by server.
- ⌚ ***Location*** - redirect the client to a location other than Request-URL for completion of the request.
- ⌚ ***Retry-After*** - indicate to client how long the service is expected to be unavailable.
- ⌚ ***Server*** - information about software used by the server to handle the request.

⌚ Entity headers:

- ⌚ *Location* - provides new URL if content has been moved to new location.
- ⌚ *Server* - provides info on server.
- ⌚ *www-authenticate* - used to provide authorization info.

⌚ Entity Body:

- ⌚ The response data is enclosed as the entity body (usually a hypertext file.)

A SAMPLE HTTP RESPONSE

HTTP/1.1 200 OK

Date: Wed, 15 Jul 2016 14:37:12 GMT

Server: Microsoft-IIS/2.0

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

Content-Encoding: gzip

Last-Modified: Fri, 10 Jul 2019 08:25:13 GMT

Server:gws

Content-Length: 3667

Allow: GET,POST,OPTIONS,HEAD

Cache-Control: max-age=60

Connection: keep-alive //only in HTTP 1.1 and HTTP/2

HTTP 1.1 ENHANCEMENTS

a) *Persistent Connections and pipelining:*

- connection is kept open such that web client can send multiple requests over the same connection.
- Helps handle *TCP Slow Start*, in a better way.
- Support for *Pipelining*

Connection:keep-alive (15s, 60 s, 300s Depending on browsers)

Connection:Upgrade

HTTP 1.1 (CONTD.)

b) *Range Request*

- ⌚ allows a web client to retrieve part of the file by using the Range header.
- ⌚ *Accept-ranges* and *Content-range* headers for facilitating byte serving.

HTTP 1.1 (CONTD.)

c) Cache Control

- Purpose of caching is to shorten retrieval time of web pages.
 - HTTP1.0
 - supports only basic caching control (*If-Modified-Since* and *Pragma*)
 - HTTP1.1
 - provides separate *Cache-control* response headers.
 - Also provides conditional headers like *If-Unmodified-Since*, *If-Match*, *If-None-Match*.

HTTP 1.1 (CONTD.)

d) *Connection Header*

- data is sent in a series of "chunks".
- Sender does not need to know the length of the content before it is sent (servers can begin transmitting dynamically-generated content)

- HTTP 1.0: *Content-length* header
- HTTP 1.1: *Transfer-encoding* header

HTTP 1.1 (CONTD.)

○ e) New Status Codes

- **C** 100 Continue: The client SHOULD continue with its request.
- **C** 101 Switching Protocols: indicates change in protocol due to Upgrade
 - message header sent by client.
- **C** 407 Proxy Authentication Required
- **C** 416 Requested Range Not Satisfiable

HTTP 1.1 (CONTD.)

- f) *Provision for additional request methods:*
 - ⌚ PUT, DELETE, OPTIONS, TRACE, CONNECT, PATCH
- g) *Better support for data compression*
 - ⌚ *accept-encoding* header
- h) *Better support for languages*
 - ⌚ *accept-language* header
- i) *Support for Proxy Authentication*

MORE READING

- ⌚ Berners-Lee, Tim, Roy Fielding, and Henrik Frystyk. "Hypertext transfer protocol--HTTP/1.0." (1996).
- ⌚ Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol—HTTP/1.1.
- ⌚ RFC 2616 – HypertextTransfer Protocol - HTTP 1.1
<https://tools.ietf.org/html/rfc2616>

Web Protocols

HTTP /2

HTTP/2

- ▶ a major revision of the HTTP.
- ▶ Approved as a Proposed Standard on February 17, 2015.
- ▶ standardization effort supported by most major browsers
 - ▶ HTTP/2 support added by the end of 2015.
- ▶ As of Sep 2019, 40.7% of top 10 million websites supported HTTP/2*

* *WorldWideWeb Technology Surveys*. W3Techs. Retrieved September 1, 2019

HTTP/2 CAPABILITIES

- ▶ **Maintain high-level compatibility with HTTP 1.1**
 - ▶ All methods, status codes, and URIs, and most header fields supported.
- ▶ **Negotiation mechanism**
 - ▶ Allows Web clients and servers to elect to use HTTP 1.1, 2.0, or potentially other non-HTTP protocols.
- ▶ **Decrease latency**
 - ▶ improve page load speed in web browsers by introducing new features.

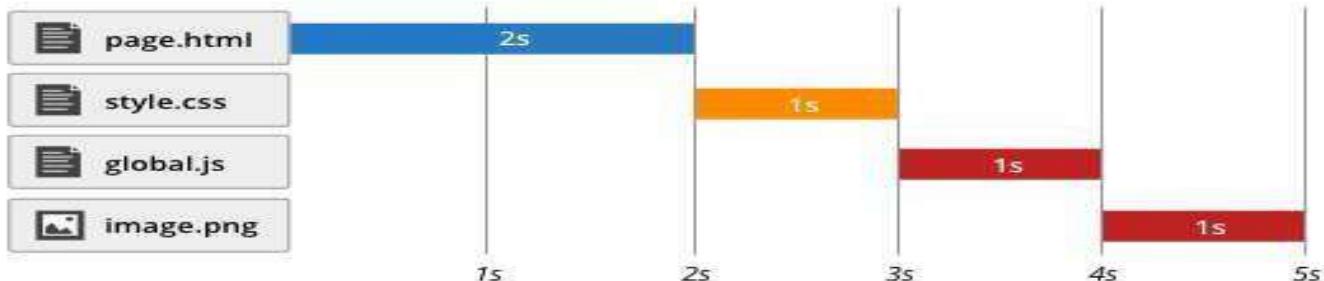
HTTP/2 - NEW FEATURES

- ▶ HTTP/2 Server Push
- ▶ Multiplexing multiple requests over a single TCP connection
- ▶ Fixing the head-of-line blocking problem in HTTP 1.x
- ▶ Support for desktop web browsers, mobile web browsers, web APIs, web servers at various scales, proxy servers, reverse proxy servers, firewalls, and content delivery networks.

HTTP/2.0 NEW FEATURES - SERVER PUSH

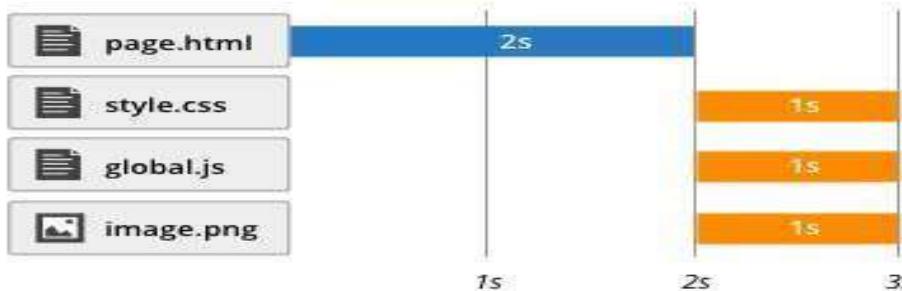
HTTP/1.1

Step 1 Request HTML



HTTP/2 Without Server Push

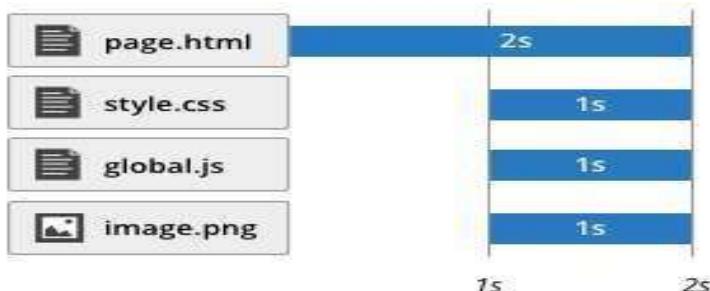
Step 1 Request HTML



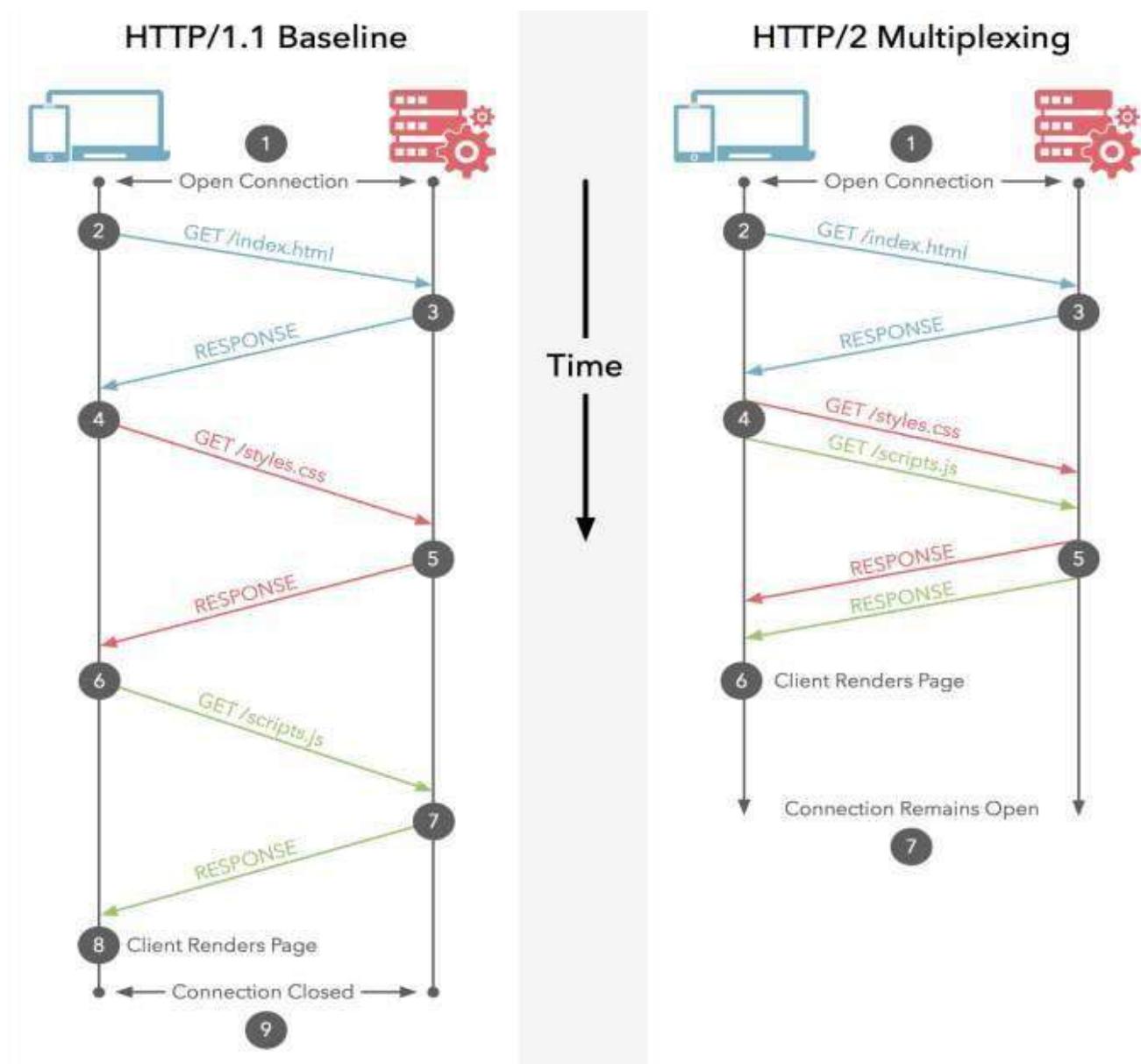
HTTP/2 With Server Push

Step 1 Request HTML

(There is no step 2)

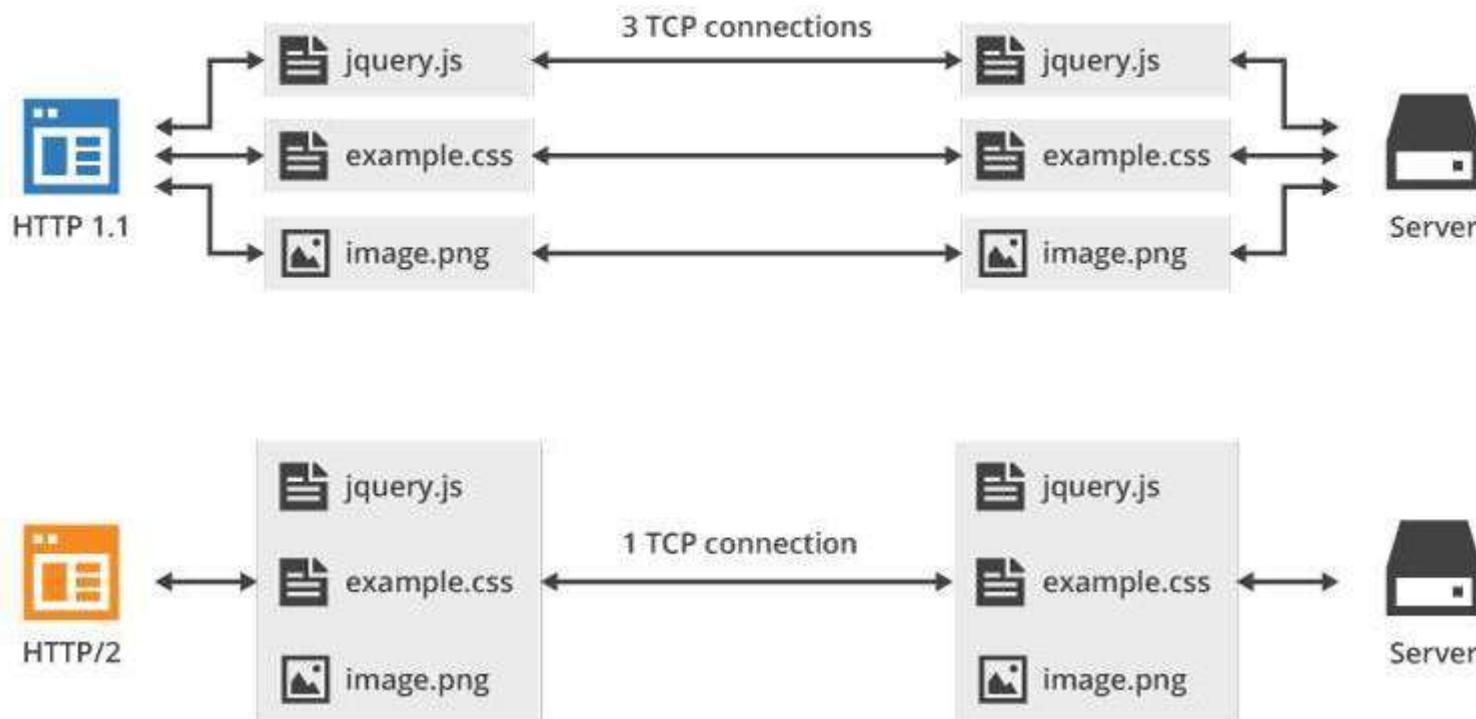


HTTP/2.0 NEW FEATURES - MULTIPLEXING



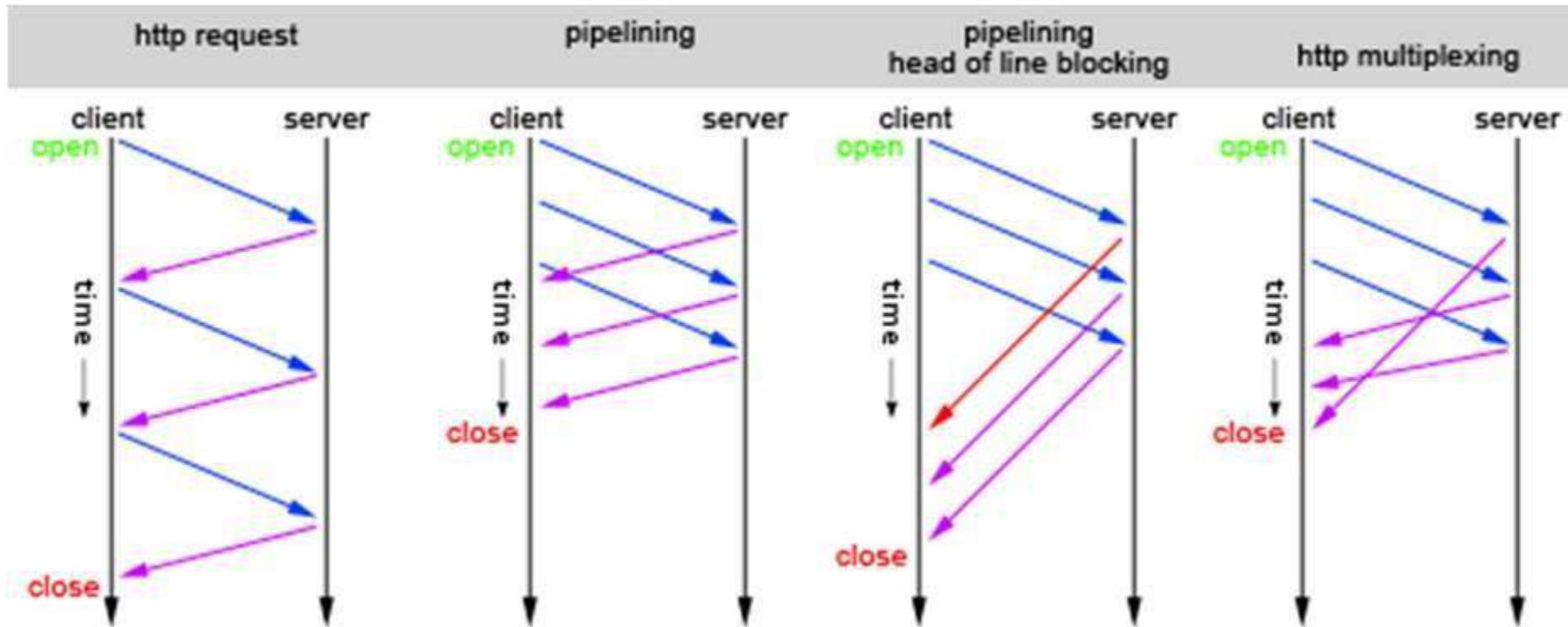
HTTP/2.0 NEW FEATURES

- ADDRESSING HEAD-OF-LINE BLOCKING



HTTP/2.0 NEW FEATURES

MULTIPLEXING & ADDRESSING HEAD-OF-LINE BLOCKING



HTTP/2.0 NEW FEATURES

- ▶ Support for
 - ▶ Desktop web browsers
 - ▶ Mobile web browsers
 - ▶ Web APIs
 - ▶ Web servers at various scales
 - ▶ Firewalls
 - ▶ Proxy servers
 - ▶ Reverse proxy servers
 - ▶ Content delivery networks.

SERVER SIDE PROGRAMMING

CLIENT-SIDE PROGRAMMING :

- It is the program that runs on the client machine (browser) and deals with the user interface/display and any other processing that can happen on client machine like reading/writing cookies.

CLIENT-SIDE PROGRAMMING :

- 1) Interact with temporary storage
- 2) Make interactive web pages
- 3) Interact with local storage
- 4) Sending request for data to server
- 5) Send request to server
- 6) work as an interface between server and user

⦿ The Programming languages for client-side programming are :

- 1) Javascript
- 2) VBScript
- 3) HTML
- 4) CSS
- 5) AJAX

SERVER-SIDE PROGRAMMING :

- It is the program that runs on server dealing with the generation of content of web page.
 - 1) Querying the database
 - 2) Operations over databases
 - 3) Access/Write a file on server.
 - 4) Interact with other servers.
 - 5) Structure web applications.
 - 6) Process user input.
- For example if user input is a text in search box, run a search algorithm on data stored on server and send the results.

- Examples :

The Programming languages for server-side programming are :

- 1) PHP
- 2) C++
- 3) Java and JSP
- 4) Python
- 5) Ruby on Rails

NODEJS

WHAT IS NODE.JS ?

- Created 2009
- Evented I/O for JavaScript
- Server Side JavaScript
- Runs on Google's V8 JavaScript Engine

WHY USE NODE.JS ?

- Node's goal is to provide an easy way to build scalable network programs.

Standard JavaScript with

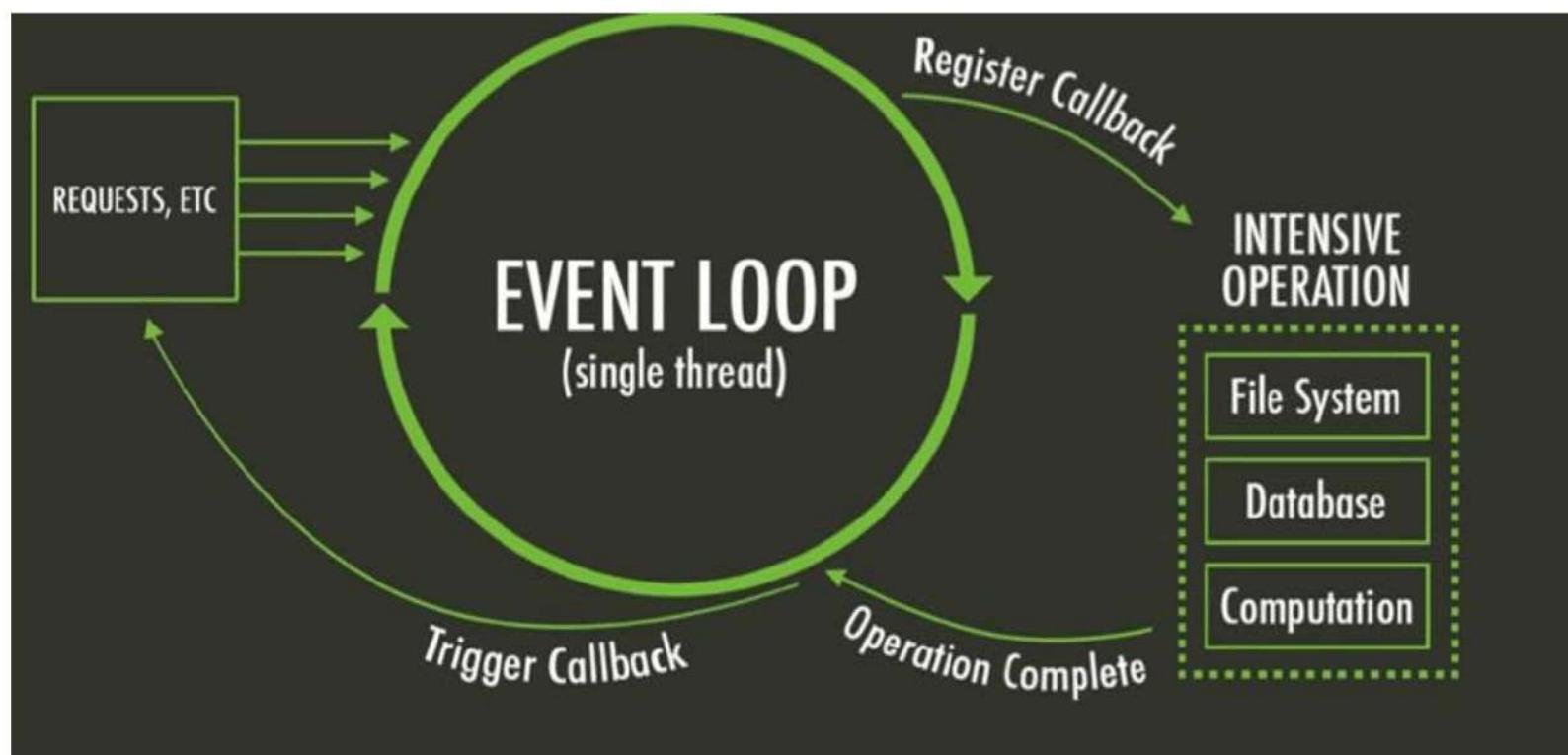
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

... but without DOM manipulation

WHAT IS UNIQUE ABOUT NODE.JS?

1. JavaScript on server-side thus making communication between client and server will happen in same language.
2. Servers are normally thread based but Node.JS is “Event” based.
3. Node.JS serves each request in a Evented loop that can handle simultaneous requests.

EVENT LOOP



WHAT CAN YOU DO WITH NODE ?

- It lets you Layered on top of the TCP library HTTP and HTTPS client/server.
- The JS is executed by the V8 javascript engine (the thing that makes Google Chrome so fast)
- Node provides a JavaScript API to access the network and file system.

WHAT WE CAN'T DO WITH NODE?

- Node is a platform for writing JavaScript applications outside web browsers.
- This is not the JavaScript we are familiar with in web browsers.
- There is no DOM built into Node, nor any other browser capability.
- Node can't run on GUI, but run on terminal

THREADS VS EVENT-DRIVEN

| Threads | Asynchronous Event-driven |
|---|---|
| Lock application / request with listener-workers threads | only one thread, which repeatedly fetches an event |
| Using incoming-request model | Using queue and then processes it |
| multithreaded server might block the request which might involve multiple events | manually saves state and then goes on to process the next event |
| Using context switching | no contention and no context switches |
| Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock | Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments |

WHY NODE.JS USE EVENT-BASED?

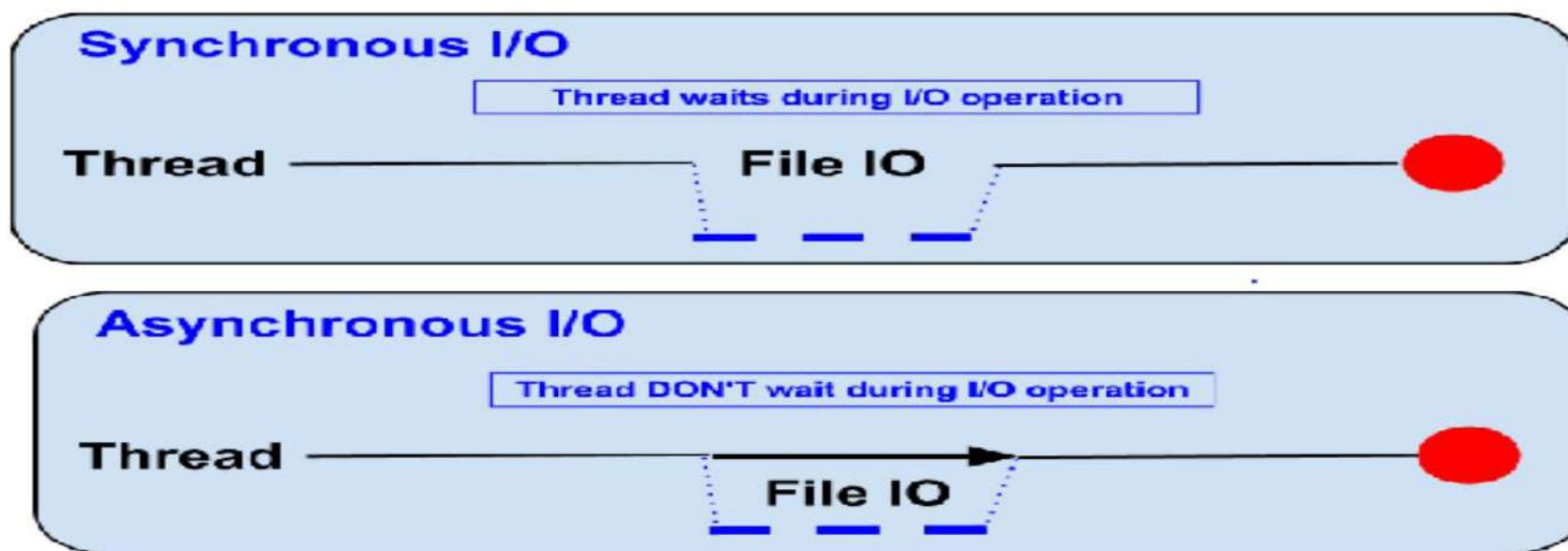
IN normal process, while processing the request the webservers will have to wait for the IO operations and thus **blocking** the next request to be processed.

Node.JS process each request as events, doesn't wait (**non-blocking**) for the IO operation to complete → it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.

BLOCKING VS NON-BLOCKING

Example :: Read data from file and show data



BLOCKING.....

- Read data from file
- Show data
- Do other tasks

```
var data = fs.readFileSync( "test.txt" );
console.log( data );
console.log( "Do other tasks" );
```

NON-BLOCKING.....

- Read data from file

When read data completed, show data

- Do other tasks

```
fs.readFile( "test.txt", function( err, data ) {  
  console.log(data);  
});
```



Callback

NODE.JS VS APACHE

1. It's faster
2. It can handle tons of concurrent requests

| Platform | Number of request per second |
|-----------------------|------------------------------|
| PHP (via Apache) | 3187,27 |
| Static (via Apache) | 2966,51 |
| Node.js | 5569,30 |

SUCCESS STORIES.....



Rails to Node

- « Servers were cut to 3 from 30 »
- « Running up to 20x faster in some scenarios »
- « Frontend and backend mobile teams could be combined [...] »



Java to Node

- « Built almost twice as fast with fewer people »
- « Double the requests per second »
- « 35% decrease in the average response time »



SUPPORTS HTTP METHOD.....

- GET
- POST
- PUT
- DELETE



When to use it ?

- Chat/Messaging
- Real-time Applications
- Intelligent Proxies
- High Concurrency Applications
- Communication Hubs
- Coordinators



NODE.JS FOR....

- Web application
- Websocket server
- Ad server
- Proxy server
- Streaming server
- Fast file upload client
- Any Real-time data apps
- Anything with high I/O



EXPRESS

- ⦿ **Minimal and flexible Node.js web application framework** that provides a robust set of features for web and mobile applications.

INSTALLING NODE.JS

- Node.js can be installed from a platform specific installer, precompiled source code, and compiled from source code
- To download the installer:
<http://www.nodejs.org>
- When installing using the platform specific installer, most users accept the defaults

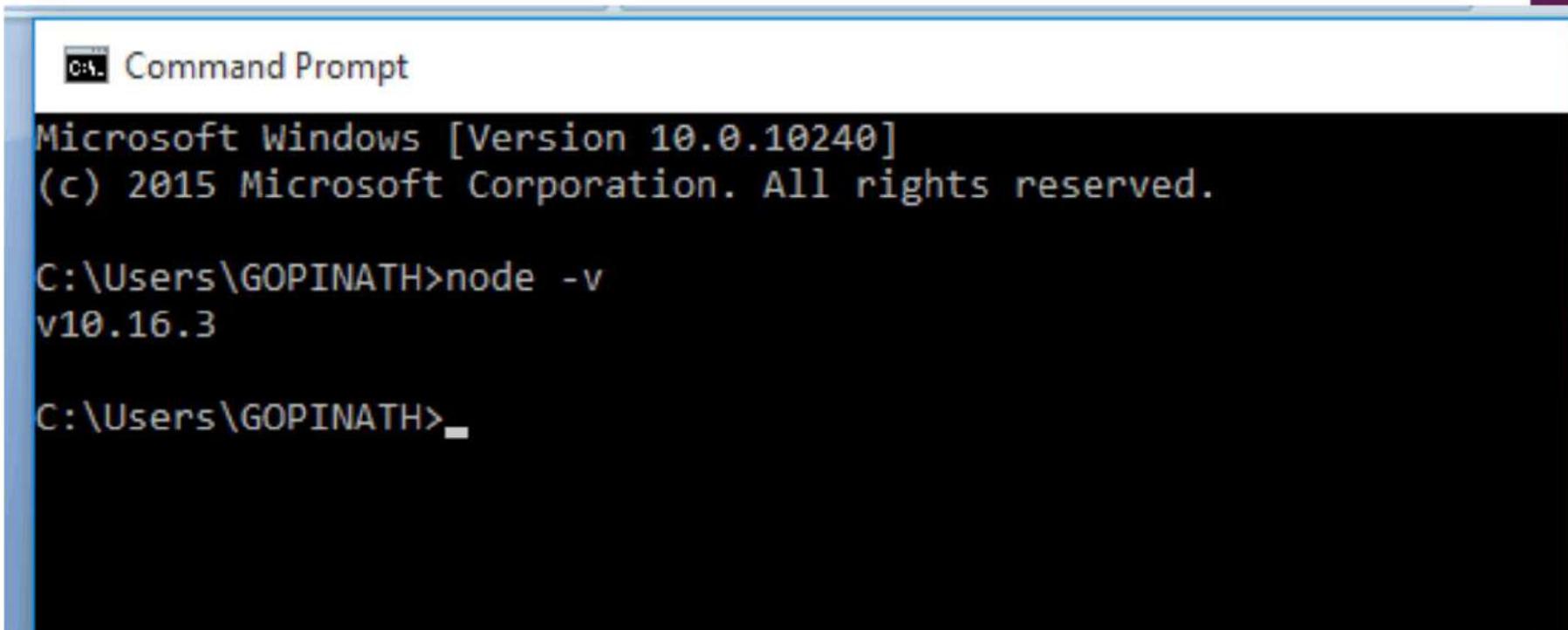
NODE.JS VERSIONS

- The installer and pre-compiled binaries can be downloaded for either the Long-Term Support version or Current Version
- Long-Term Support version is best for applications in production
- The Current Version is best for working with the newest features

INSTALL

The screenshot shows a Windows desktop environment with a browser window open to the Node.js website (nodejs.org/en/). The browser has four tabs visible in the title bar: 'You are signed in as tfit014', 'server side languages - Google', 'Gmail', and 'Nodejs'. The Node.js website features a dark header with the Node.js logo and navigation links for HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, NEWS, and FOUNDATION. The FOUNDATION link is highlighted with a green background. Below the header, a sub-header states: 'Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.' Two large green buttons are prominently displayed: '10.16.3 LTS' (labeled 'Recommended For Most Users') and '12.9.1 Current' (labeled 'Latest Features'). Below these buttons are links to 'Other Downloads | Changelog | API Docs' for each version. A note below the LTS button says: 'Or have a look at the Long Term Support (LTS) schedule.' Further down, there is a link to 'Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.' At the bottom of the page, a footer includes links to 'Report Node.js issue | Report website issue | Get Help', a copyright notice ('© Node.js Foundation. All Rights Reserved. Portions of this site originally © Joyent.'), and a trademark notice ('Node.js is a trademark of Joyent, Inc. and is used with its permission. Please review the Trademark Guidelines of the Node.js Foundation.'). The Windows taskbar at the bottom shows icons for the Start button, Search, File Explorer, and Task View, along with system status icons like battery level and signal strength. The date and time '27-Aug-19' are also visible.

VERSION



A screenshot of a Microsoft Windows Command Prompt window. The title bar reads "Command Prompt". The window displays the following text:

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\GOPINATH>node -v
v10.16.3

C:\Users\GOPINATH>
```

USING NODE ON TERMINAL

```
Command Prompt - node

Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\GOPINATH>node -v
v10.16.3

C:\Users\GOPINATH>npm -v
6.11.1

C:\Users\GOPINATH>
C:\Users\GOPINATH>node
> var name = 'priya'
undefined
> console.log(name);
priya
undefined
> function hello() { return 'hello ' + name}
undefined
> hello();
'hello priya'
>
```

CREATING MODULES AND EXPORTING

HANDLING POST REQUEST

POST

- ◉ Request method
- ◉ Request asks the server to accept/store data which is enclosed in the body of the request.
- ◉ Often used when submitting a form

POST

- app.post
- Form - method= ‘post’ action=destination.
- The rest of the work is in .js file
- Handle the post request
- When request object is used we require additional middleware for handling such post request.

POST

- Body-parser - npm body parser, which helps the npm to carry out many functionalities.
- Parses incoming request bodies in a middleware before handler.
- Available under req.body.
- Check npm-body parser.

BODY-PARSER

- Express-route specific

- When posting in a specific route.
- Eg:/contact
- Invoke body parser

BODY-PARSER

- Var bodyParser= require('body-parser');
- Var url=urlencodedparser({});
- Console.log(req.body);

BODY-PARSER

- The requested data is posted with the help of req.body
- Middleware- gives access to the body property on the request object.
- Example.

MODEL VIEW CONTROL

- ◉ Architectural Design pattern
- ◉ Most frequently used
- ◉ Separates application functionality
- ◉ Promotes organized programming

- ◉ Ruby on rails
- ◉ Express
- ◉ Flask
- ◉ Django
- ◉ angular

- ⦿ There are many ways to do it.

MODEL

- Responsible for getting and manipulating the data
- Brain of the application
- Interacts with the DB
- Communicates with the controller

VIEW

- The user interacts with the application
- UI
- Html+css+dynamic values send from the controller
- Controller Communicates with the view as well as model
- Template Engine may vary

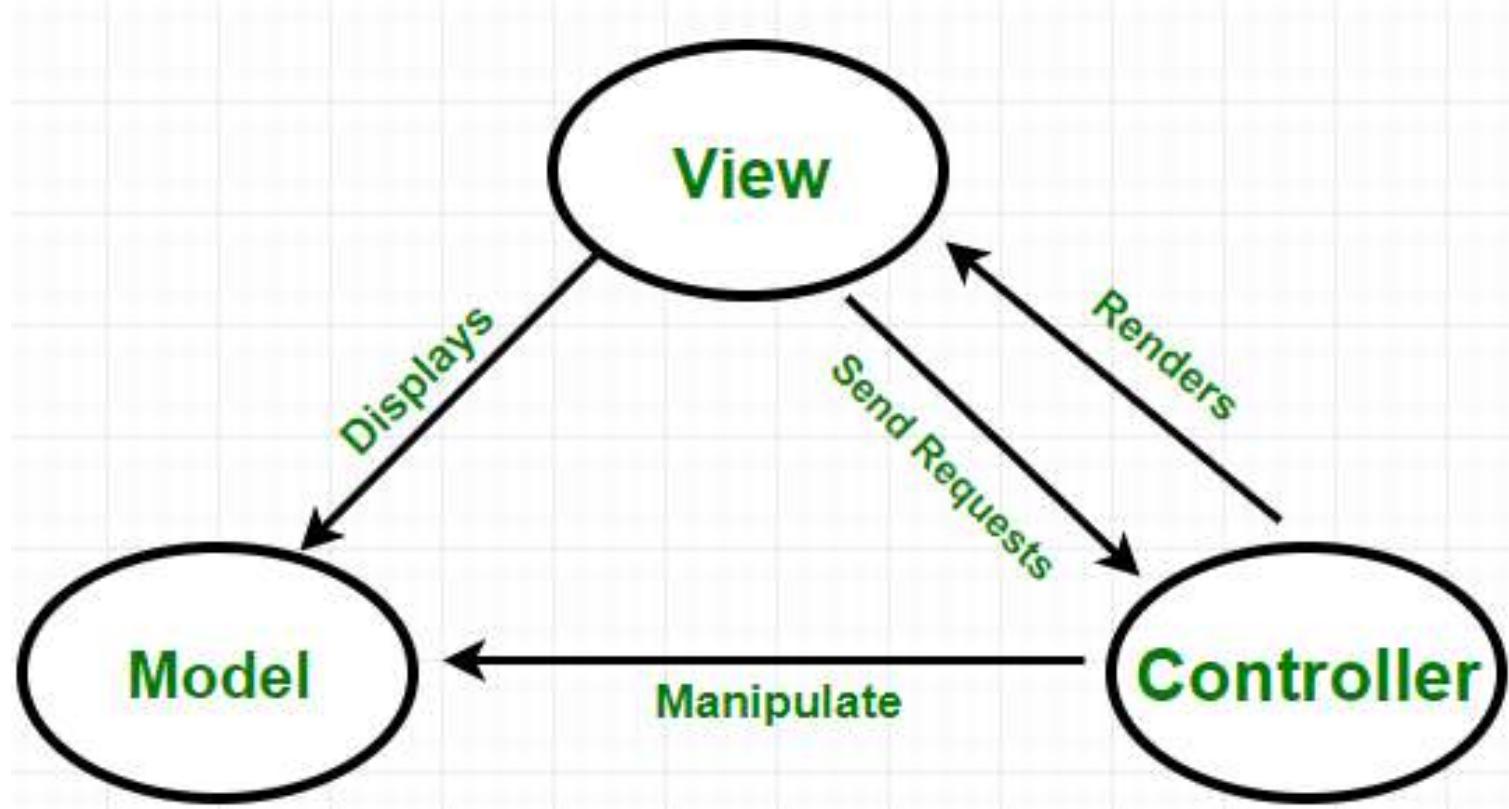
CONTROLLER

- Acts as a middle man
- Takes in user input
- Process request
- Gets data from model
- Passes data to view

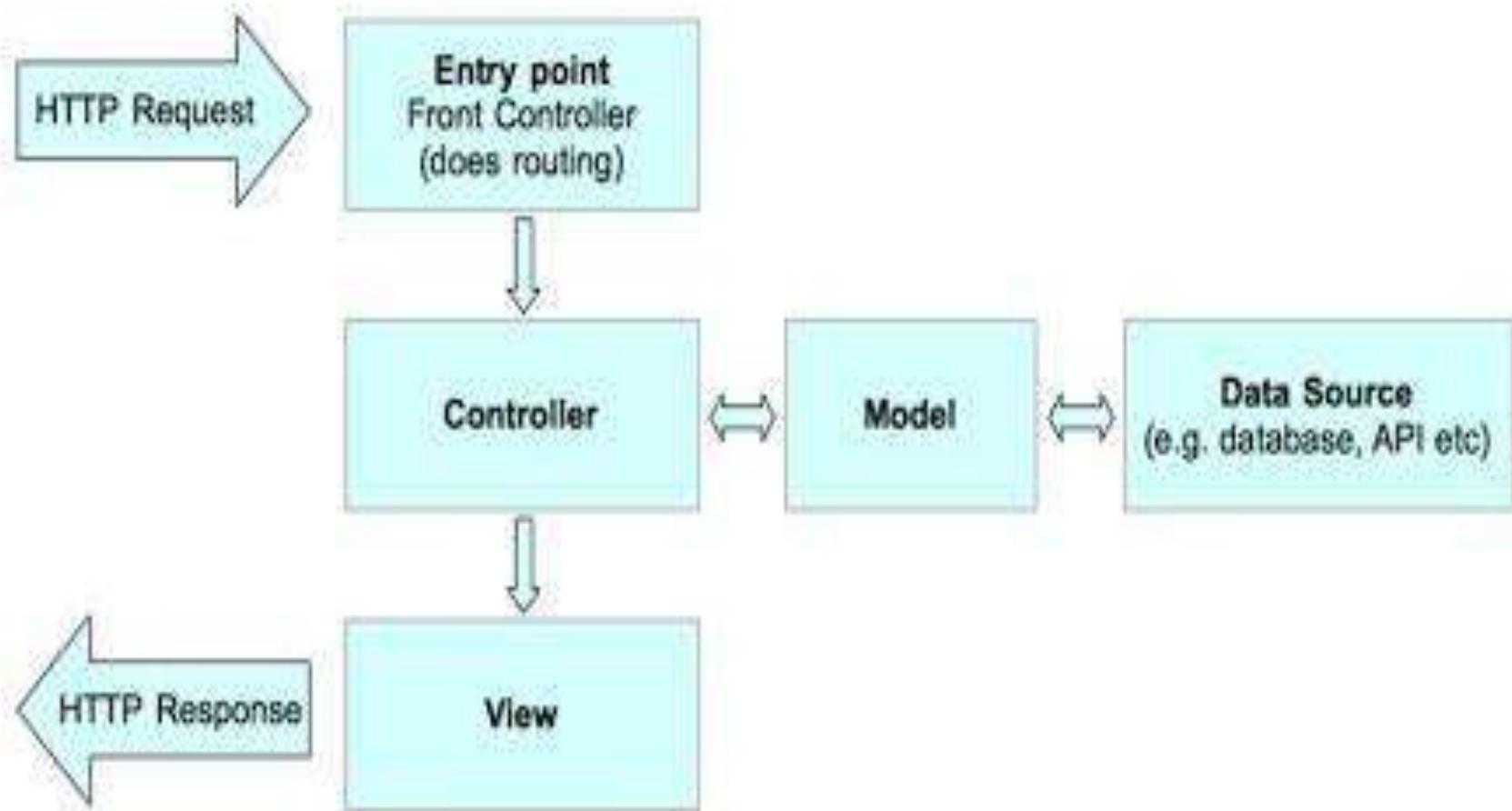
CONTROLLER

- The controller asks the model for data from DB.
- The controller takes that data and loads the view.
- Finally pass those data through view.
- The controller can also load a view without passing data , so just plain web pages can also be passed.

MVC



MVC- NODE JS



EXAMPLE

/routes

user/profile/:id=users . getProfile(id)

//when the route is set to this particular path the control and the particular function is called.

EXAMPLE

- ◎ /controllers

```
class user{ function getProfile(id)  
{  
Profile=this.UserModel.getProfile(id);  
Renderview('user/profile',profile)}
```

*//calling getprofile function in view
//data will be returned to profile and
this profile will be send to view.*

EXAMPLE

- /models

```
class usermodel
```

```
{
```

```
Function getProfile(id)
```

```
{
```

```
Data=this.db.get('select* from users where id='id');
```

```
Return data;
```

EXAMPLE

● /view

```
/users/profile  
<h1><%= profile.name %>  
<ul>  
<li>Email : <%= profile.email %>  
<li>Phone : <%= profile.phone %>  
</ul>
```

//based on view engine.

MAKING AN APPLICATION

○ To-do Application

○ Steps:

- Create a new folder - public - assets- (style.css, images, basic operation files)
- Create our package.json file
 - Install all dependencies
 - Npm init
- Install require packages
 - Express
 - EJS
 - Body-parser

SAMPLE FILE

- App.js

*//this is going to be an express application
so we require .*

```
Var express= require('express');
```

*//accesing the express functionalities in a
variable*

```
Var app=express();
```

//setting a template Engine(ejs)

```
App.set('view engine', 'ejs');
```

SAMPLE FILE

//static file loader

```
App.use('/assets',express.static('./public'));
```

//listening to port

```
App.listen(3000);
```

*//next step will be create folders
according to mvc pattern.*

THE MVC ARCHITECTURE AND DESIGN PRINCIPLES

1. *Divide and conquer*: Three components can be somewhat independently designed.
2. *Increase cohesion*: Components have **stronger** layer cohesion than if the view and controller were together in a single UI layer.
3. *Reduce coupling*: **Minimal** communication channels among the three components.
4. *Increase reuse*: The **view** and **controller** normally make **extensive** use of **reusable components** for various kinds of UI controls.
5. *Design for flexibility*: It is usually quite easy to change the UI by changing the **view**, the **controller**, or both.
6. *Design for testability*: Can *test* application separately from the UI.

STATE MANAGEMENT & SESSION TRACKING

STATE MANAGEMENT

- ◉ What to do when server data gets updated?
- ◉ How to refresh our user Interface?
- ◉ What happens if the user refreshes the browser URL?
- ◉ Should we reload all the data?

STATE MANAGEMENT

- ◉ Common ways to handle state related questions.
- ◉ Defines data flow

STATE MANAGEMENT

- Server side scripting:

client -server communication -http

http-stateless

unable to carry data

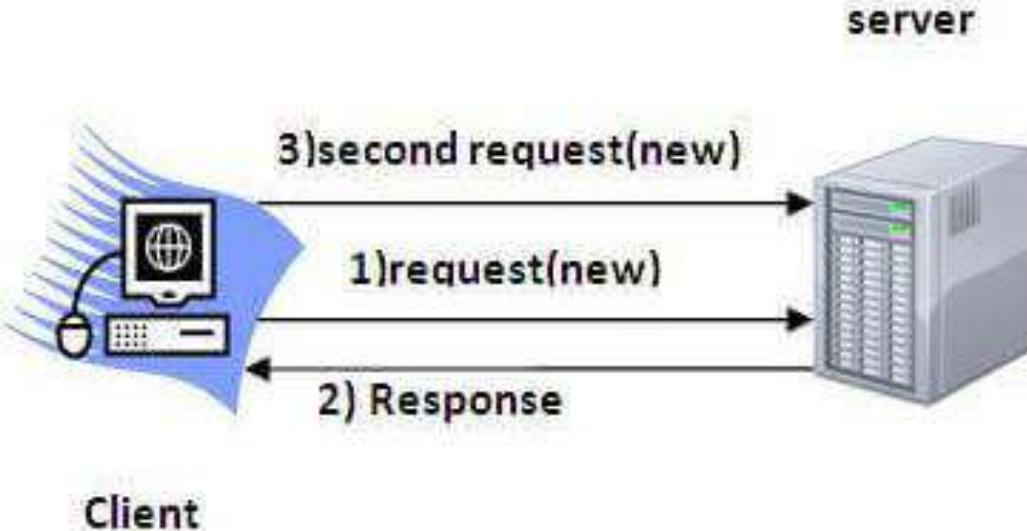
STATE MANAGEMENT

- Eg: when user enters data in webform and when it is submitted the http will not carry any data ,so to maintain those data we have a concept of state management.
- Two types: 1. client side :cookie
 2.Server side: sessions.

SESSION TRACKING

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- HTTP is stateless that means each request is considered as the new request.

SESSION TRACKING



WHY SESSION?

- To recognize the user .
- When there is a need to maintain the conversational state, session tracking is needed.
- For example, in a shopping cart application a client keeps on adding items into his cart using multiple requests.
- When every request is made, the server should identify in which client's cart the item is to be added. So in this scenario, there is a certain need for session tracking.
- Solution is, when a client makes a request it should introduce itself by providing unique identifier every time.
- There are five different methods to achieve this.

SESSION TRACKING METHODS:

- User authorization
- Hidden fields
- URL rewriting
- Cookies
- Session tracking API

USER AUTHORIZATION

- Users can be authorized to use the web application in different ways. Basic concept is that the user will provide username and password to login to the application. Based on that the user can be identified and the session can be maintained.

HIDDEN FIELDS

- <INPUT TYPE="hidden" NAME="technology" VALUE="servlet">
- Hidden fields like the above can be inserted in the webpages and information can be sent to the server for session tracking.
- These fields are not visible directly to the user, but can be viewed using view source option from the browsers.
- This type doesn't need any special configuration from the browser or server and by default available to use for session tracking.
- This cannot be used for session tracking when the conversation included static resources like html pages.

URL REWRITING

- When a request is made, additional parameter is appended with the url.
- In general added additional parameter will be sessionid or sometimes the userid.
- It will suffice to track the session.
- This type of session tracking doesn't need any special support from the browser.
- Disadvantage is, implementing this type of session tracking is tedious. We need to keep track of the parameter as a chain link until the conversation completes and also should make sure that, the parameter doesn't clash with other application parameters.

URL REWRITING

- Original

URL: http://server:port/servlet/ServletName

Rewritten

URL: http://server:port/servlet/ServletName
?sessionid=7456

COOKIES

- Cookies are the mostly used technology for session tracking.
- Cookie is a key value pair of information, sent by the server to the browser.
- This should be saved by the browser in its space in the client computer.
- Whenever the browser sends a request to that server it sends the cookie along with it.
- Then the server can identify the client using the cookie.

COOKIES

- Session tracking is easy to implement and maintain using the cookies.
- Disadvantage is that, the users can opt to disable cookies using their browser preferences. In such case, the browser will not save the cookie at client computer and session tracking fails.

SESSION TRACKING API

- Session tracking API is built on top of the first four methods.
- This is in order to help the developer to minimize the overhead of session tracking.

USING EXPRESS FRAMEWORK

The image shows a code editor window with three tabs: 'session.js', 'sessionview.js', and 'express2js'. The 'express2js' tab is active and contains the following code:

```
1 var express= require('express');
2 var bodyParser = require('body-parser');
3 var cookieparser= require('cookie-parser');
4
5 var app= express();
6 app.use(cookieparser());
7 var urlencodedParser = bodyParser.urlencoded({ extended: false })
8 app.get('/', function (req,res) {
9     res.cookie('mycookie','hi this is my cookie');
10    res.end('hi welcome');
11 });
12
13 app.get('/remove', function(req,res)
14 {
15     res.clearCookie('mycookie');
16 });
17
18 app.listen(3000,'127.0.0.1');
19 console.log('you are listening to 3000');
20
```

MAXAGE-COOKIE

The image shows a code editor interface with two tabs: 'session.js' and 'sessionview.js'. The 'session.js' tab is active, displaying the following code:

```
1 var express = require('express');
2 var cookieParser = require('cookie-parser');
3 var session = require('express-session');
4
5 var app = express();
6
7 app.use(cookieParser());
8 app.use(session({secret: "secret!", saveUninitialized:'true', resave:'true', cookie:{maxAge: 30000}}));
9
10 app.get('/', function(req, res){
11     if(req.session.page_views){
12         req.session.page_views++;
13         res.send("You visited this page " + req.session.page_views + " times");
14     } else {
15         req.session.page_views = 1;
16         res.send("Welcome to this page for the first time!");
17     }
18 });
19 app.listen(3000);
```

The code is written in JavaScript and defines an Express application. It uses the cookie-parser middleware to parse cookies and the express-session middleware to handle sessions. The session secret is set to 'secret!', and session data is saved even if it's not initialized. The session cookie has a maximum age of 30,000 milliseconds (30 seconds). The application handles a single route ('/') that checks if a user has visited the page before. If they have, it increments the page view count and sends a response indicating the total number of visits. If they haven't, it initializes the page view count to 1 and sends a welcome message.

INTRODUCTION TO MONGO DB(NO SQL DATA BASE)

Overview

- What is NO SQL Data base?
- Types of NO SQL Data base.
- What is Mongo DB?
- Why Mongo DB?
- Mongo DB Architecture.
- Document (JSON) Structure.
- Differences between XML and JSON.
- Different Methods.
- When to use Mongo DB?

WHAT IS NO SQL DATA BASE

- ◉ It's Not No SQL it's NOT ONLY SQL.
- ◉ It's not even a replacement to RDBMS.

As compared to the good olden days we are saving more and more data.

Connection between the data is growing in which we require an architecture that takes advantage of these two key issues.

TYPES OF NO SQL DATA BASE

○ Key Value pair

Dynamo DB

Azure Table Storage (ATS)

(#key,#value)
(Name, Tom)
(Age,25)
(Role, Student)
(University, CU)

○ Document Based

Mongo Db

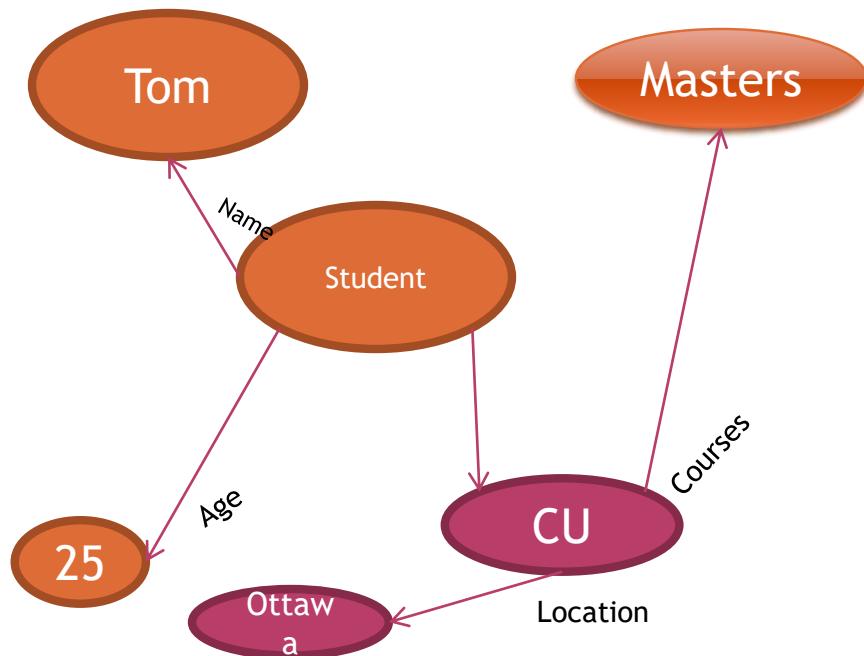
AmazonSimple DB
Couch DB

```
[  
 {  
   "Name": "Tom",  
   "Age": 30,  
   "Role": "Student",  
   "University": "CU",  
 }  
 ]
```

TYPES OF NO SQL DATA BASE

Graph database

- Neo4j
- Infogrid



Column Oriented database

| Row Id | Columns | |
|--------|---------|---------|
| 1 | Name | Tom |
| | Age | 25 |
| | Role | Student |

Bigtable(Google)

H Base

WHAT IS MONGO DB

MongoDB is a cross-platform, document oriented database that provides

- High performance.
- High availability.
- Easy scalability.

MongoDB works on concept of collection and document.

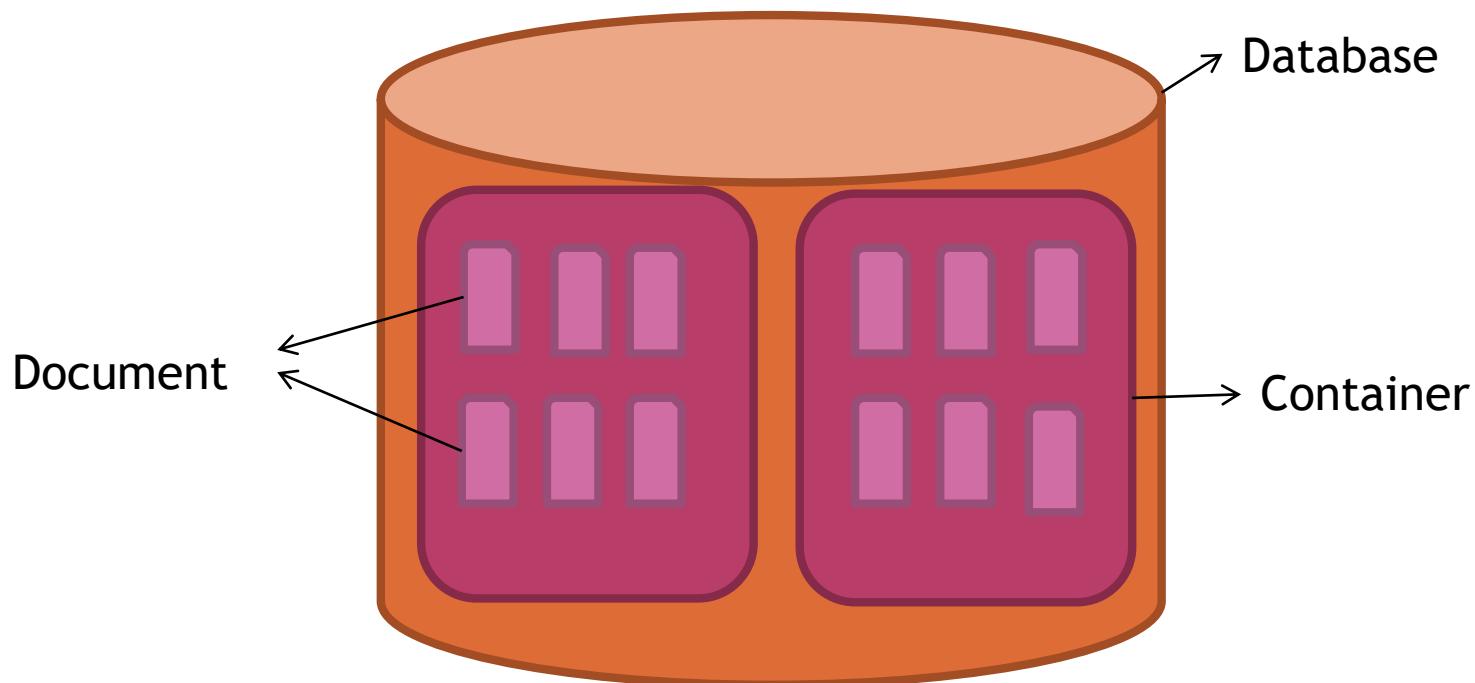
WHY MONGO DB?

- All the modern applications deals with huge data.
- Development with ease is possible with mongo DB.
- Flexibility in deployment.
- Rich Queries.
- Older database systems may not be compatible with the design.

And it's a document oriented storage:- Data is stored in the form of JSON Style.

MONGO DB ARCHITECTURE

Architecture : -



DOCUMENT(JSON) STRUCTURE

- The document has simple structure and very easy to understand the content
- JSON is smaller, faster and lightweight compared to XML.
- For data delivery between servers and browsers, JSON is a better choice
- Easy in parsing, processing, validating in all languages
- JSON can be mapped more easily into object oriented system.

```
[  
 {  
  "Name": "Tom",  
  "Age": 30,  
  "Role": "Student",  
  "University": "CU",  
 }  
 {  
  "Name": "Sam",  
  "Age": 32,  
  "Role": "Student",  
  "University": "OU",  
 }  
 ]
```

DIFFERENCE BETWEEN XML AND JSON

| XML | JSON |
|---|---|
| It is a markup language. | It is a way of representing objects. |
| This is more verbose than JSON. | This format uses less words. |
| It is used to describe the structured data. | It is used to describe unstructured data which include arrays. |
| JavaScript functions like <code>eval()</code> , <code>parse()</code> doesn't work here. | When <code>eval</code> method is applied to JSON it returns the described object. |
| <p>Example:</p> <pre><car> <company>Volkswagen</comp any> <name>Vento</name> <price>800000</price> </car></pre> | { "company": "Volkswagen", "name": "Vento", "price": 800000} |

WHY JSON?

- JSON is faster and easier than XML when you are using it in AJAX web applications:
- Steps involved in exchanging data from web server to browser involves:

Using XML

1. Fetch an XML document from web server.
2. Use the XML DOM to loop through the document.
3. Extract values and store in variables.
4. It also involves type conversions.

Using JSON

1. Fetch a JSON string.
2. Parse the JSON string using eval() or parse() JavaScript functions.

THE INSERT() METHOD

- To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.
- The basic syntax of **insert()** command is as follows –

“db.COLLECTION_NAME.insert(document)”

- Example: -

```
db.StudentRecord.insert ({  
    "Name": "Tom",  
    "Age": 30,  
    "Role": "Student",  
    "University": "CU",  
},  
{  
    "Name": "Sam",  
    "Age": 22,  
    "Role": "Student",  
    "University": "OU",  
}  
)
```

THE FIND() METHOD

- To query data from MongoDB collection, you need to use MongoDB's **find()** method.
- The basic syntax of **find()** method is as follows –
“db.COLLECTION_NAME.find()”
- **find()** method will display all the documents in a non-structured way.
- To display the results in a formatted way, you can use **pretty()** method.
“db.mycol.find().pretty()”

- Example: -

```
db.StudentRecord.find()  
    .pretty()
```

THE REMOVE() METHOD

- MongoDB's **remove()** method is used to remove a document from the collection. **remove()** method accepts two parameters. One is deletion criteria and second is **justOne** flag.
- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.
- Syntax
- **db.COLLECTION_NAME.remove(DELETION_CRITERIA)**

Remove based on
DELETION_CRITERIA

```
db.StudentRecord.remove(  
  {"Name": "Tom"})
```

Remove Only One:-
Removes first record

```
db.StudentRecord.remove(  
  DELETION_CRITERIA,1)
```

Remove all Records

```
db.StudentRecord.remove()
```

WHEN TO USE MONGO DB?

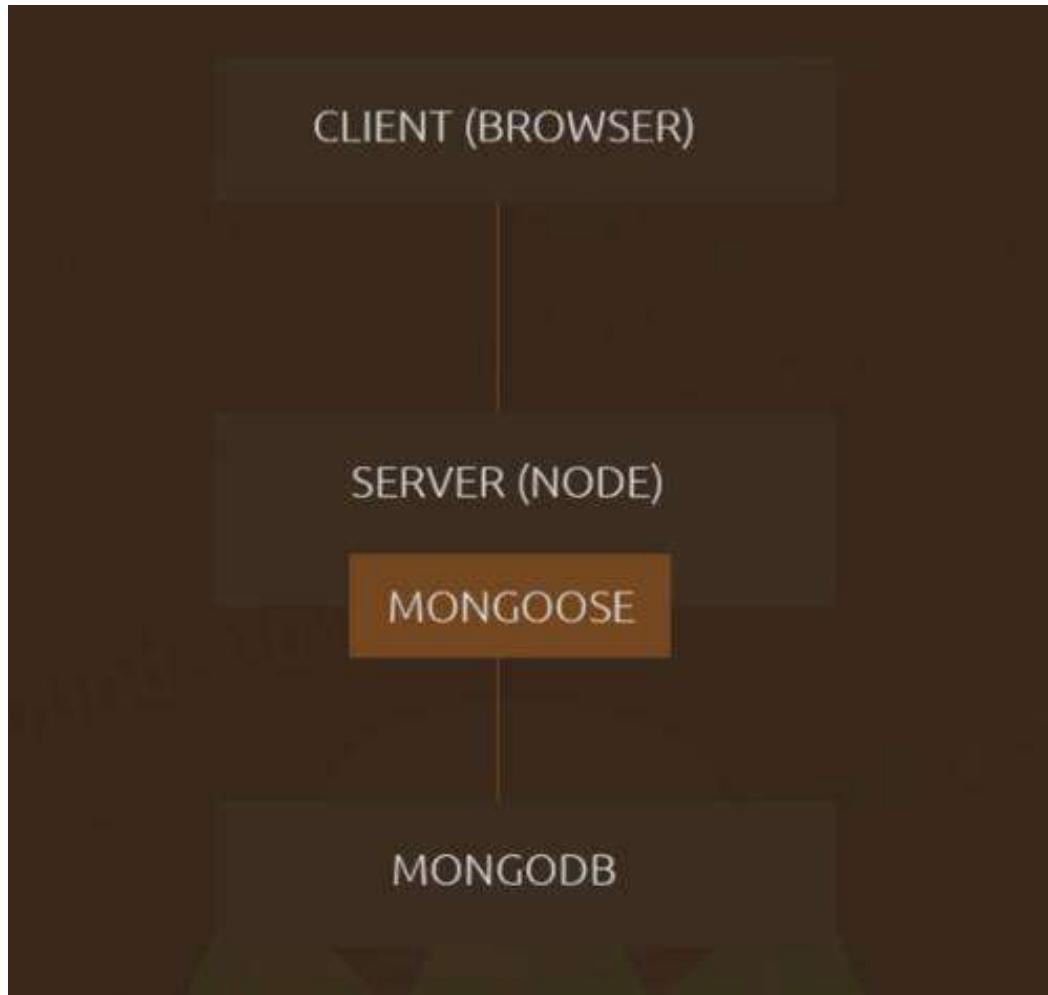
When your requirements has these properties :

- You absolutely must store unstructured data. Say things coming from 3rd-party API you don't control, logs whose format may change any minute, user-entered metadata, but you want indexes on a subset of it.
- You need to handle more reads/writes than single server can deal with and master-slave architecture won't work for you.
- You change your schema very often on a large dataset.

MONGO IN NODEJS

- NO SQL DB
- Instead of storing the data in table we store it in documents of collection of objects.
- Easier to communicate
- Use it anytime we want to store data in our application
- Mongo is the M in MEAN

MONGO IN NODEJS



MONGOOSE

- Package that can be installed in node js , which makes communication with mongodb easier.
- STEPS:
 - Install MongoDB and Mongoose
 - Use basic CRUD operations
 - Mocha - Testing enviornment

○ Download MongoDB:

- MongoDB.com
- Check Os
- It doesn't know where to store data
(dir) - create /data/db folder in c drive.

○ Prerequisite

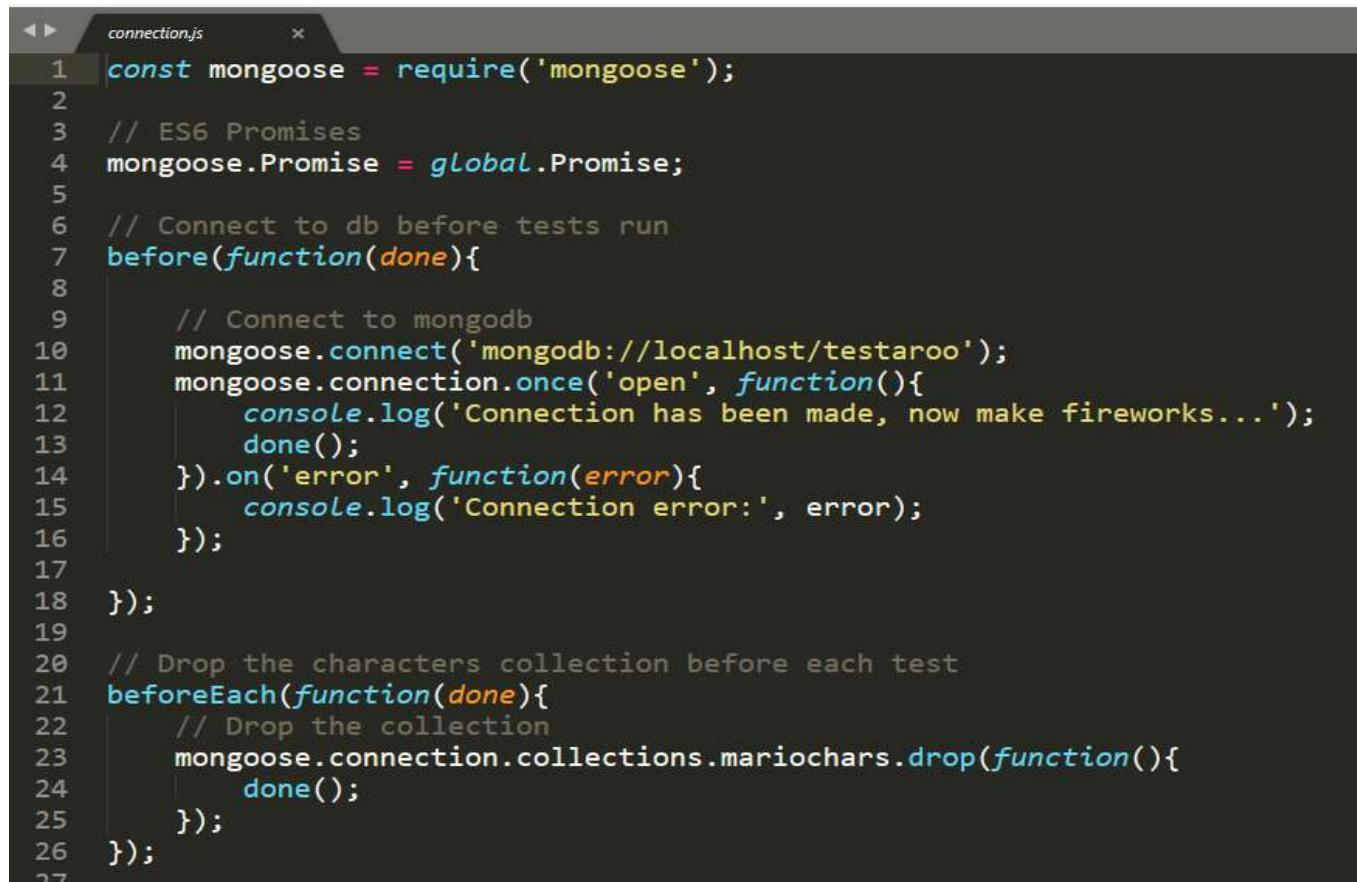
- Javascript
- Install nodejs

CONNECTING TO MONGODB

- Just because we have installed mongoose it doesn't automatically know to connect to MONGODB.
- It doesn't even know that a DB exist.
- We have to explicitly tell mongoose to connect.
- How?

CREATING A CONNECTION

- Test/connection.js
- Connection file contains all connections.



A screenshot of a code editor window titled "connection.js". The code is written in JavaScript and uses the Mongoose library to manage MongoDB connections. It includes setup for ES6 promises, a connection before tests, and a drop operation before each test.

```
const mongoose = require('mongoose');
// ES6 Promises
mongoose.Promise = global.Promise;
// Connect to db before tests run
before(function(done){
    // Connect to mongodb
    mongoose.connect('mongodb://localhost/testaroo');
    mongoose.connection.once('open', function(){
        console.log('Connection has been made, now make fireworks...');
        done();
    }).on('error', function(error){
        console.log('Connection error:', error);
    });
});
// Drop the characters collection before each test
beforeEach(function(done){
    // Drop the collection
    mongoose.connection.collections.mariochars.drop(function(){
        done();
    });
});
```

COLLECTION AND MODEL

```
1 |const mongoose = require('mongoose');
2 |const Schema = mongoose.Schema;
3 |
4 // Create a Schema and a Model
5
6 const MarioCharSchema = new Schema({
7   name: String,
8   weight: Number
9 });
10
11 const MarioChar = mongoose.model('mariochar', MarioCharSchema);
12
13 module.exports = MarioChar;
14
```

MOCHA

- Testing library
 - Application to check the working.
 - Checks in all the stages.
-
- Npm install mocha -save
 - Create test files.

CREATING TEST

```
Const assert=require('assert');
Describe('some demo test', function () {
It ('adds two number together' , function ()
{
assert(2+3 ===5); } );});
```

SAVE DATA

- We have just used mongoose to create it, but it is not in our DB yet.
- So what we have to do is save the char variable.
- When we create this new instance mongoose gives us a lot of methods to work/interact with DB.

SAVE DATA

```
1  saving_test.js
2
3
4  const assert = require('assert');
5  const MarioChar = require('../models/mariochar');
6
7  // Describe our tests
8  describe('Saving records', function(){
9
10    // Create tests
11    it('Saves a record to the database', function(done){
12
13      const char = new MarioChar({
14        name: 'Mario'
15      });
16
17      char.save().then(function(){
18        assert(!char.isNew);
19        done();
20      });
21    });
22  });

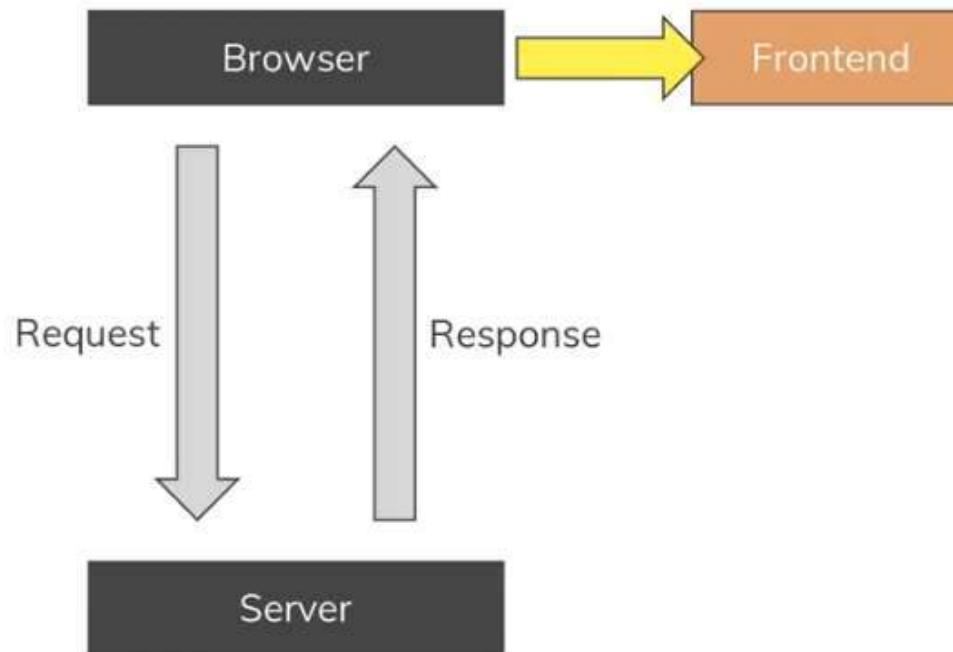
```

- ◉ .save is an asynchronous method.
- ◉ .then is default promise library.

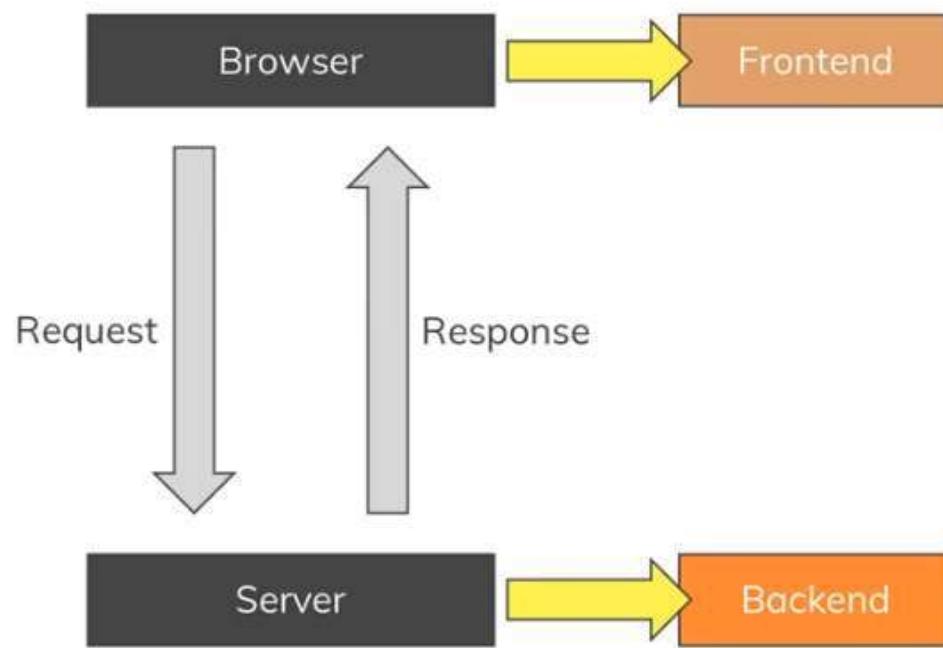
FULL STACK - WEB DEVELOPMENT

FRONTEND DEVELOPMENT

What?

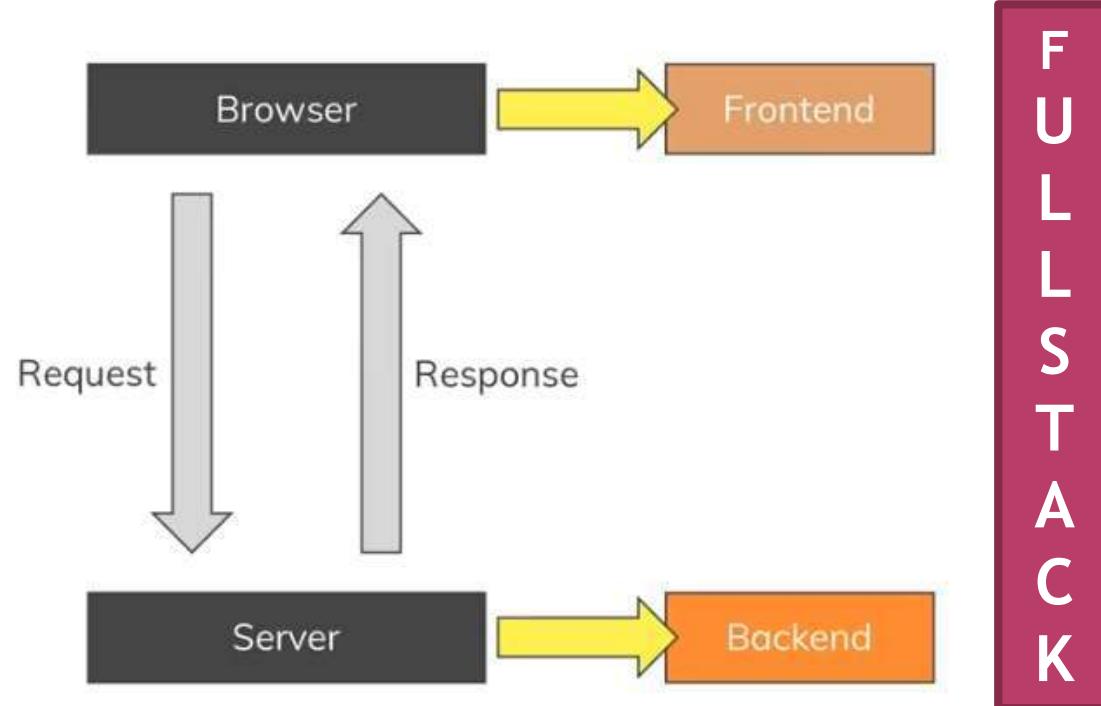


BACKEND DEVELOPMENT



FULL STACK DEVELOPMENT

What?



FRONTEND

Technologies/ Languages

- HTML
- CSS
- JavaScript
- CSS Pre-processors (Sass, Stylus...)
- JavaScript Libraries (e.g. lodash) and Frameworks (Angular, React, Vue)
- Build Tools (npm, Webpack, ...)

You'll work on ...

- JS-driven User Interfaces
- Re-usable UI Components with JS logic and CSS Styling
- Forms & Input Validation
- Backend Communication Channels
- UX Strategies (PWAs, Live Updates)

Less Relevant Technologies/ Languages

- Server-side Languages (e.g. Node, PHP)
- Databases/ Query Languages (e.g. SQL)
- Server Configuration

You'll NOT work on ...

- Server-side Business Logic (e.g. User Authentication, Order Handling)
- Automatic E-Mail Notifications
- Database Access

BACKEND

Technologies/ Languages

- Server-side Languages like Node, PHP
- Frameworks like Express, Laravel
- Databases & Query Languages
- Partly: Server Configuration
- Basic HTML, CSS, JavaScript

You'll work on ...

- Server-side Business Logic (e.g. User Authentication, Order Handling)
- Automatic Notifications
- Data Validation
- Data Storage/ Database Access
- Scheduled Processes

Less Relevant Technologies/ Languages

- Advanced JavaScript & CSS
- JavaScript Libraries & Frameworks
- Build Tools (npm, Webpack)

You'll NOT work on ...

- Client-side Validation
- Complex User Interfaces
- Advanced UX Strategies (PWAs, ...)

FULL STACK

Technologies/ Languages

- HTML, CSS, JavaScript
- Server-side Languages like Node
- Server-side Frameworks like Express
- Advanced JavaScript & CSS
- Basic JS Libraries/ Frameworks
- Databases & Query Language

You'll work on ...

- Both Server-side Logic and Client-side User Interfaces
- Client-side and Server-side Data Validation
- Data Storage/ Database Access
- Everything else

Less Relevant Technologies/ Languages

- Advanced Libraries or Frameworks (both on Backend and Frontend)
- Build Tools (use Templates/ CLIs instead)

You'll NOT work on ...

- Very Complex User Interfaces
- Very Complex Server-side Logic

POPULAR STACKS

- ◉ **LAMP stack:** JavaScript - Linux - Apache - MySQL - PHP
- ◉ **LEMP stack:** JavaScript - Linux - Nginx - MySQL - PHP
- ◉ **MEAN stack:** JavaScript - MongoDB - Express - AngularJS - Node.js
- ◉ **Django stack:** JavaScript - Python - Django - MySQL
- ◉ **Ruby on Rails:** JavaScript - Ruby - SQLite - PHP

ADVANTAGES

- You can make a prototype very rapidly
- You can provide help to all the team members
- You can reduce the cost of the project
- You can reduce the time used for team communication
- You can switch between front and back end development based on requirements
- You can better understand all aspects of new and upcoming technologies

DISADVANTAGES

- ◉ The solution chosen can be wrong for the project
- ◉ The solution chosen can be dependent on developer skills
- ◉ The solution can generate a key person risk
- ◉ Being a full stack developer is increasingly complex

FULL STACK JAVASCRIPT

- JavaScript has been around for over 20 years.
- It is the dominant programming language in web development.
- In the beginning JavaScript was a language for the web client (browser).
- Then came the ability to use JavaScript on the web server (with Node.js).

FULL STACK JAVASCRIPT

- Today the hottest buzzword is "Full Stack JavaScript".
- The idea of "Full Stack JavaScript" is that all software in a web application, both client side and server side, should be written using JavaScript only.

FULL STACK JS

- A full stack JavaScript developer is a person who can develop both **client** and **server** software.
- In addition to mastering HTML and CSS, he/she also knows how to:
- Program a **browser** (like using JavaScript, jQuery, Angular, or Vue)
- Program a **server** (like using Node.js)
- Program a **database** (like using MongoDB)

BACK END LANGUAGES

• ~~PHP~~

• ~~ASP~~

• ~~C++~~

• ~~C#~~

• ~~Java~~

• ~~Python~~

• Node.js

• Ruby

• REST

• GO

• SQL

• MongoDB

FULL STACK JAVASCRIPT BENEFITS

- ◉ Code reuse. Shared libraries, templates, and models.
- ◉ Best practice accumulated by 20 years of JavaScript.
- ◉ JavaScript is an evolving standard with a bright future.
- ◉ Easy to learn.
- ◉ No compilation. Faster development.
- ◉ Great distribution: npm.

EXAMPLE

- MEAN STACK
- M- Mongo DB
- E- Express js
- A- Angular js
- N- Node js.



RISE OF THE RESPONSIVE SINGLE PAGE APP



Image:

<http://johnpolacek.github.io/scrolldeck.js/decks/responsive/>

RESPONSIVE

- Unified across experiences
- Can be embedded as mobile app
- Better deployment and & maintenance
- Mobile users need to get access to everything



Image: <http://coenraets.org/blog/wp-content/uploads/2011/10/directory11.png>

SINGLE--PAGE APPLICATIONS (SPA)

- Web app that fits on a **single web page**
 - Fluid UX, like desktop app
 - Examples like Gmail, Google maps
- Html page contains **mini--views** (HTML Fragments) that can be loaded in the background
- **No reloading** of the page,
- Requires handling of **browser history, navigation and bookmarks**

JAVASCRIPT

- SPAs are implemented using **JavaScript** and **HTML**

CHALLENGES IN SPA

- **DOM Manipulation**
 - How to manipulate the view efficiently?
- **History**
 - What happens when pressing back button?
- **Routing**
 - Readable URLs?
- **Data Binding**
 - How bind data from model to view?
- **View Loading**
 - How to load the view?
- Lot of coding! You could **use a framework instead**
 - ...

SINGLE-PAGE APPLICATION

- Single page apps typically have
 - “application like” interaction
 - dynamic data loading from the server-side API
 - fluid transitions between page states
 - more JavaScript than actual HTML
- They typically do not have
 - support for crawlers (not for sites relying on search traffic)
 - support for legacy browsers (IE7 or older, dumbphone browsers)

SPAS ARE GOOD FOR ...

- “App-like user experience”
- Binding to your own (or 3rd party) RESTful API
- Replacement for Flash or Java in your web pages
- Hybrid (native) HTML5 applications
- Mobile version of your web site

*The SPA sweet spot is likely not on web sites,
but on content-rich cross-platform mobile apps*

PJAX

- Pjax is a technique that allows you to progressively enhance normal links on a page so that clicks result in the linked content being loaded via Ajax and the URL being updated using HTML5 pushState, avoiding a full page load.
- In browsers that don't support pushState or that have JavaScript disabled, link clicks will result in a normal full page load. The Pjax Utility makes it easy to add this functionality to existing pages.

<http://yuilibrary.com/yui/docs/pjax/>

SPAs AND OTHER WEB APP ARCHITECTURES

| | Server-side | Server-side + AJAX | PJAX | SPA |
|--------------------------|---|---|---|--|
| What | Server round-trip on every app state change | Render initial page on server, state changes on the client | Render initial page on server, state changes on server, inject into DOM on client-side | Serve static page skeleton from server; render every change on client-side |
| How | UI code on server; links & form posting | UI code on both ends; AJAX calls, ugly server API | UI code on server, client to inject HTTP, server API if you like | UI code on client, server API |
| Ease of development | | | | |
| UX & responsiveness | | | | |
| Robots & old browsers | | | | |
| Who's using it? | Amazon, Wikipedia; banks, media sites etc. | Facebook?; widgets, search | Twitter, Basecamp, GitHub | Google+, Gmail, FT; mobile sites, startups |

ANGULAR_JS

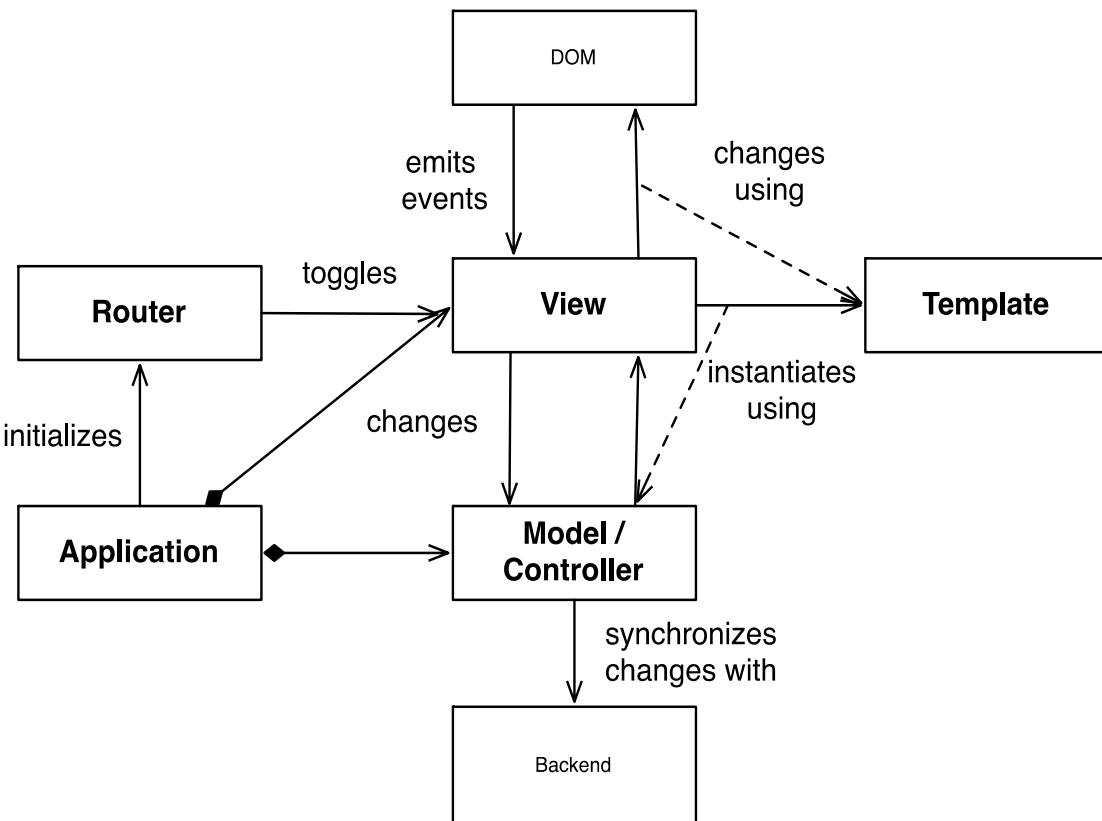
ANGULAR JS

- **Single Page App Framework** for JavaScript
- Implements client--side **MVC** pattern
 - Separation of presentation from business logic and presentation state
- **No direct DOM manipulation**, less code
- Support for all major browsers
- Supported by Google
- Large and fast growing community

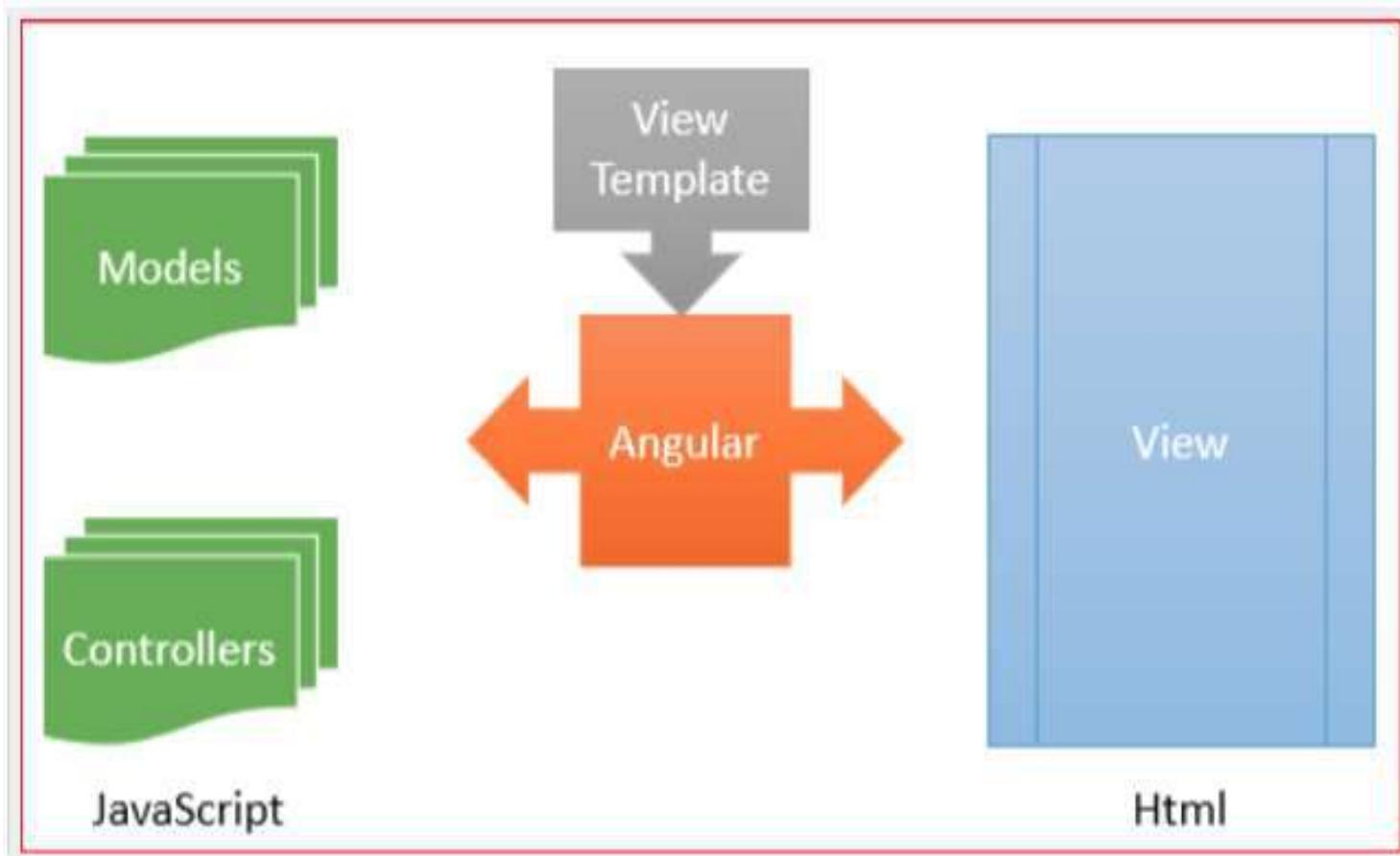
ANGULARJS - MAIN CONCEPTS

- Templates
- Directives
- Expressions
- Data binding
- Scope
- Controller s
- Modules
- Filters
- Services
- Routing

ANATOMY OF A BACKBONE SPA

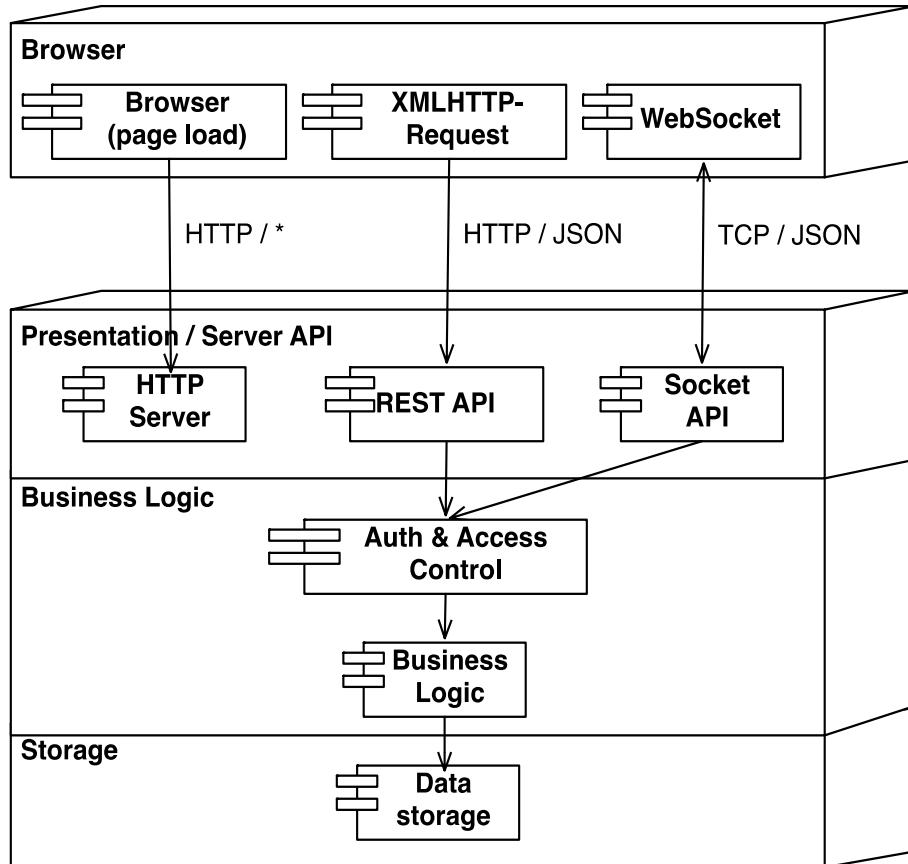


- Application as a ‘singleton’ reference holder
- Router handles the navigation and toggles between views
- Models synchronize with Server API
- Bulk of the code in views
- All HTML in templates



From Gary Arora

SPA CLIENT-SERVER COMMUNICATION

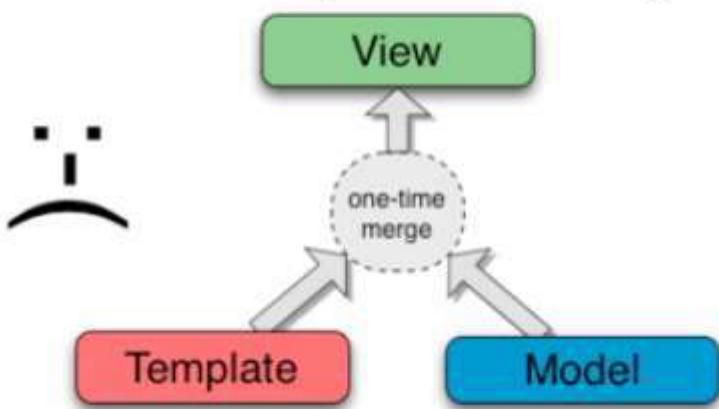


- HTML and all the assets are loaded in first request
- Additional data is fetched over XMLHttpRequest
- If you want to go real-time, WebSockets ([socket.io](#)) can help you
- When it gets slow, cluster the backend behind a caching reverse proxy like [Varnish](#)

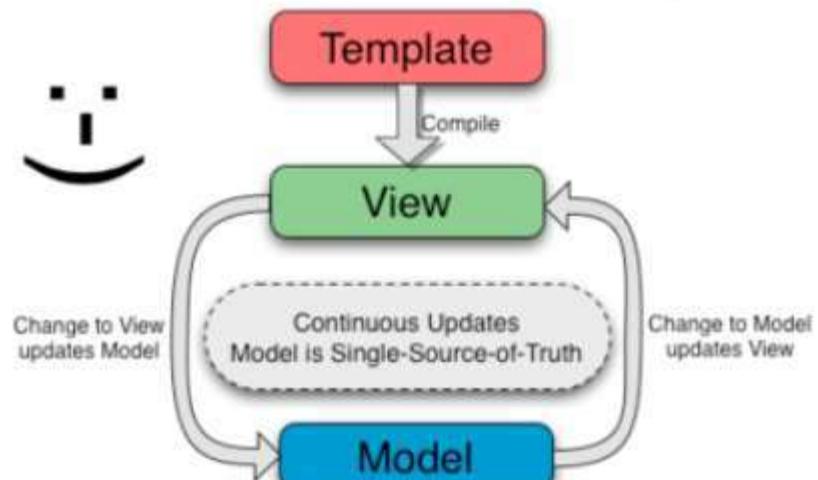
HOW IT WORKS?



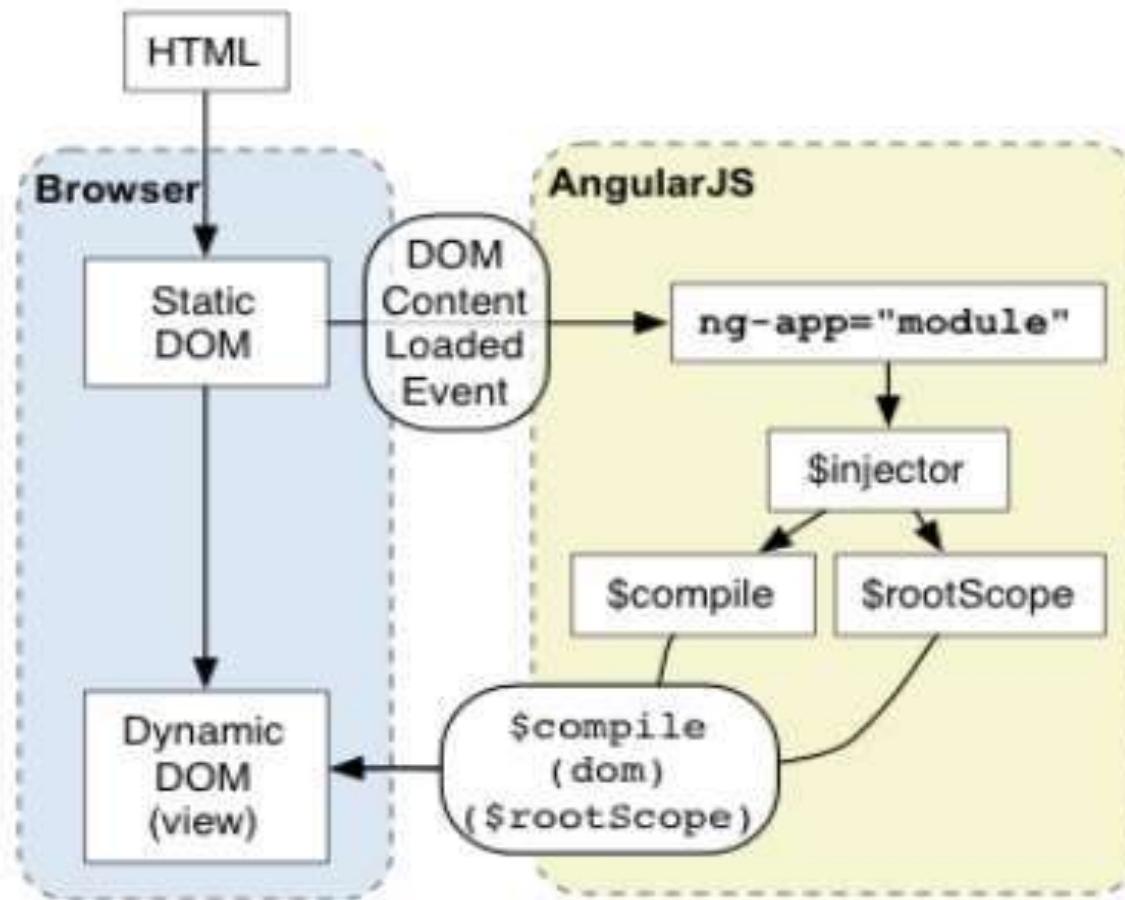
One-Way Data Binding



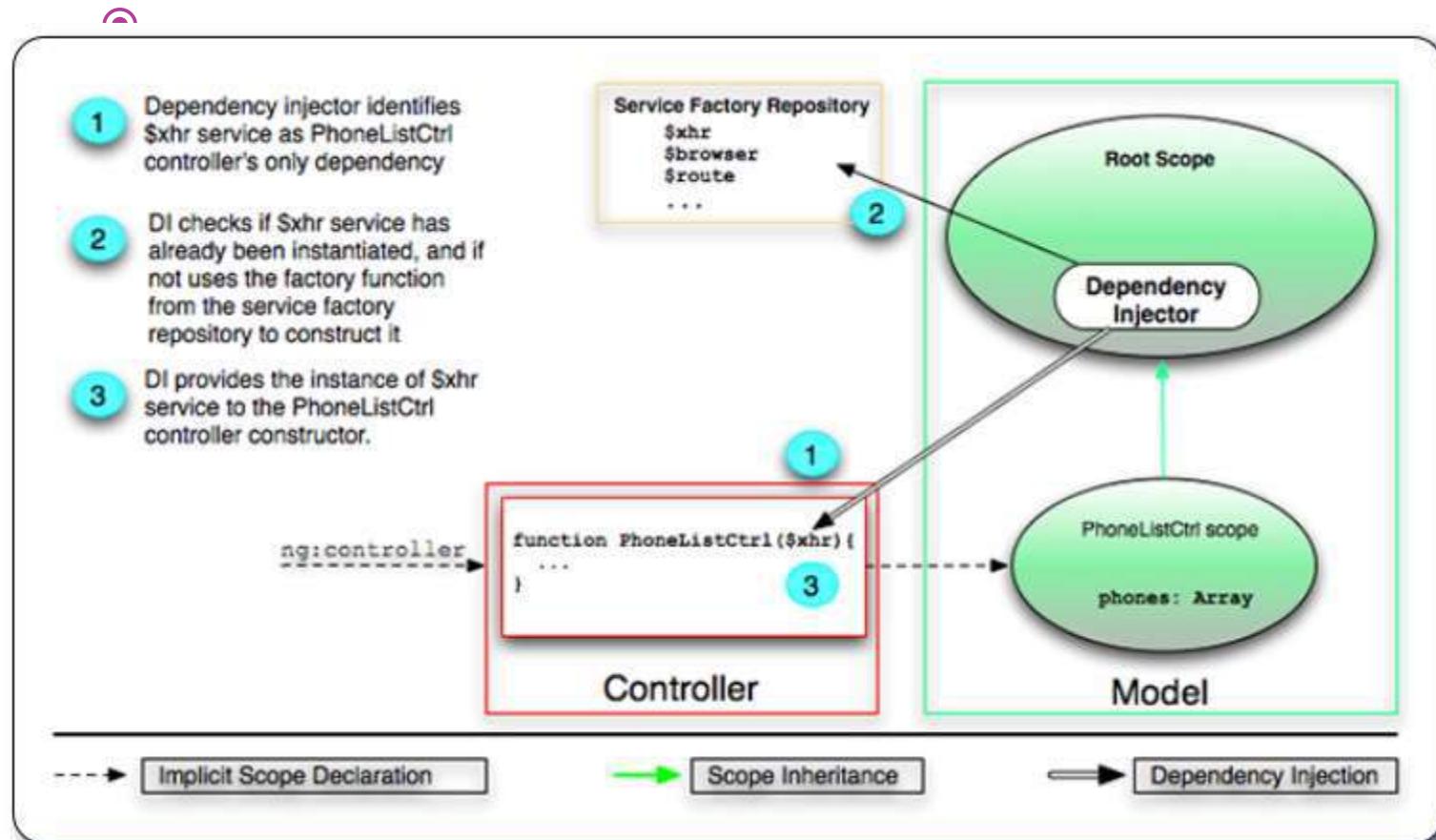
Two-Way Data Binding



HOW IT WORKS?



HOW IT WORKS?



GETTING STARTED WITH ANGULAR_JS

BASIC CONCEPTS

- **1) Templates**
 - HTML with additional markup, directives, expressions, filters ...
- **2) Directives**
 - Extend HTML using ng-app, ng-bind, ng-model
- **3) Filters**
 - Filter the output: filter, orderBy, uppercase
- **4) Data Binding**
 - Bind model to view using expressions {{ }}

pippo

FIRST EXAMPLE - TEMPLATE

Template

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  </head>
  <body>
    <div ng-app>
      <!-- store the value of input field into a variable name --&gt;
      &lt;p&gt;Name: &lt;input type="text" ng-model="name"&gt;&lt;/p&gt;
      <!-- display the variable name inside (innerHTML) of p --&gt;
      &lt;p ng-bind="name"&gt;&lt;/p&gt;
    &lt;/div&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre>
```

2) DIRECTIVES

- **Directives** apply special behavior to attributes or elements in HTML
 - Attach behaviour, transform the DOM
- Some directives
 - **ng-app**
 - Initializes the app
 - **ng-model**
 - Stores/updates the value of the input field into a variable
 - **ng-bind**
 - Replace the text content of the specified HTML with the value of given expression

ABOUT NAMING

- AngularJS HTML Compiler supports multiple formats
 - ng-bind
 - Recommended Format
 - data-ng-bind
 - Recommended Format to support HTML validation
 - ng_bind, ng:bind, x-ng-bind
 - Legacy, don't use

LOT OF BUILT IN DIRECTIVES

- ngApp
- ngClick
- ngController
- ngModel
- ngRepeat
- ngSubmit
- ngDb1Click
- ngMouseEnte
r
- ngMouseMove
- ngMouseLeav
e
- ngKeyDown
- ngForm

2) EXPRESSIONS

- Angular **expressions** are JavaScript–like code snippets that are usually placed in bindings
 - `{{ expression }}`.
- Valid Expressions
 - `{{ 1 + 2 }}`
 - `{{ a + b }}`
 - `{{ items[index] }}`
- Control flow (loops, if) are not supported!
- You can use **filters** to format or filter data

EXAMPLE

Number 1:
Number 2:

13

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
    </script>
  </head>
  <body>
    <div ng-app>
      <p>Number 1: <input type="number" ng-model="number1"></p>
      <p>Number 2: <input type="number" ng-model="number2"></p>
      <!-- expression -->
      <p>{{ number1 + number2 }}</p>
    </div>
  </body>
</html>
```

The diagram illustrates the components of the provided AngularJS code. Three callout bubbles point to specific parts of the code:

- A large blue callout bubble points to the entire `ng-app` directive in the `<div>` tag.
- A smaller blue callout bubble points to the `ng-model` attribute on the first `<input>` tag.
- A large blue callout bubble points to the `ng-model` attribute on the second `<input>` tag.
- A large blue callout bubble points to the `expression` part of the code, specifically the `number1 + number2` part of the `ng-bind` expression.

NG-INIT AND NG-REPEAT DIRECTIVES

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  </head>
  <body>
    <div data-ng-init="names = ['Jack', 'John', 'Tina']">
      <h1>Cool loop!</h1>
      <ul>
        <li data-ng-repeat="name in names">{{ name }}</li>
      </ul>
    </div>
  </body>

</html>
```

- Jack
- John
- Tina

Cool loop!

3) FILTER

- With **filter**, you can **format or filter** the output
- **Formatting**
 - currency, number, date, lowercase, uppercase
- **Filtering**
 - filter, limitTo
- **Other**
 - orderBy, json

USING FILTERS - EXAMPLE

Cool loop!

```
<!DOCTYPE html>
<html data-ng-app="">
<head>
<title>Title</title>
<meta charset="UTF-8" />
<style media="screen"></style>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
</head>
<body>
<div data-ng-init="customers = [{name:'tina'}, {name:'jack'}]">
<h1>Cool loop!</h1>
<ul>
<li data-ng-repeat="customer in customers | orderBy:'name'">
{{ customer.name | uppercase }}</li>
</ul>
</div>
</body>
</html>
```

- JACK
- TINA



USING FILTERS - EXAMPLE

Customers

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  >
  </head>
  <body>
    <div data-ng-init=
"customers = [{name:'jack'}, {name:'tina'}, {name:'john'}, {name:'donald'}]">
      <h1>Customers</h1>
      <ul>
        <li data-ng-repeat="customer in customers | orderBy:'name' | filter:'john'">{{customer.name | uppercase }}</li>
      </ul>
    </div>
  </body>

</html>
```

- JOHN

USING FILTERS - USER INPUT FILTERS THE DATA

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script
      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
    </script>
  </head>
  <body>
    <div data-ng-init=
"customers = [{name:'jack'}, {name:'tina'}, {name:'john'},
{name:'donald'}]"
    >
      <h1>Customers</h1>

      <input type="text" data-ng-model="userInput" />
      <ul>
        <li data-ng-repeat="customer in customers | orderBy:'name' |
filter:userInput">{{ customer.name | uppercase }}</li>
      </ul>
    </div>
  </body>

</html>
```

Customers

- JACK
- JOHN

API REFERENCE

<https://docs.angularjs.org/api/ng/filter/filter>

The screenshot shows a web browser displaying the AngularJS API Reference for the `filter` component. The URL in the address bar is `https://docs.angularjs.org/api/ng/filter/filter`. The page header includes the Angular logo and navigation links for Home, Learn, Develop, and Discuss. A search bar is present at the top right. The main content area shows the `filter` component details, including its description as a filter in the `ng` module, its usage in HTML template binding and JavaScript, and its arguments. A sidebar on the left lists other filter-related components like `currency`, `date`, `json`, `limitTo`, `lowercase`, `number`, `orderBy`, `uppercase`, `auto`, `service`, and `ngAnimate`.

filter
- filter in module ng

Selects a subset of items from `array` and returns it as a new array.

Usage

In HTML Template Binding

```
{% filter_expression | filter : expression : comparator %}
```

In JavaScript

```
$filter('filter')(array, expression, comparator)
```

Arguments

| Param | Type | Details |
|------------|---------|--|
| array | Array | The source array. |
| expression | string | The predicate to be used for selecting items from <code>array</code> . |
| | boolean | Can be one of: |

VIEWS, CONTROLLERS, SCOPE

MODEL - VIEW - CONTROLLERS

- **Controllers** provide the **logic** behind your app.
 - So use controller when you need logic behind your UI
- AngularJS apps are controller by controllers
- Use **ng-controller** to define the controller
- Controller is a **JavaScript Object, created by standard JS object constructor**

MODEL - VIEW - CONTROLLERS

a controller is a JavaScript function

- It contains data
- It specifies the behavior
- It should contain only the business logic needed for a single view.

VIEW, CONTROLLER AND SCOPE

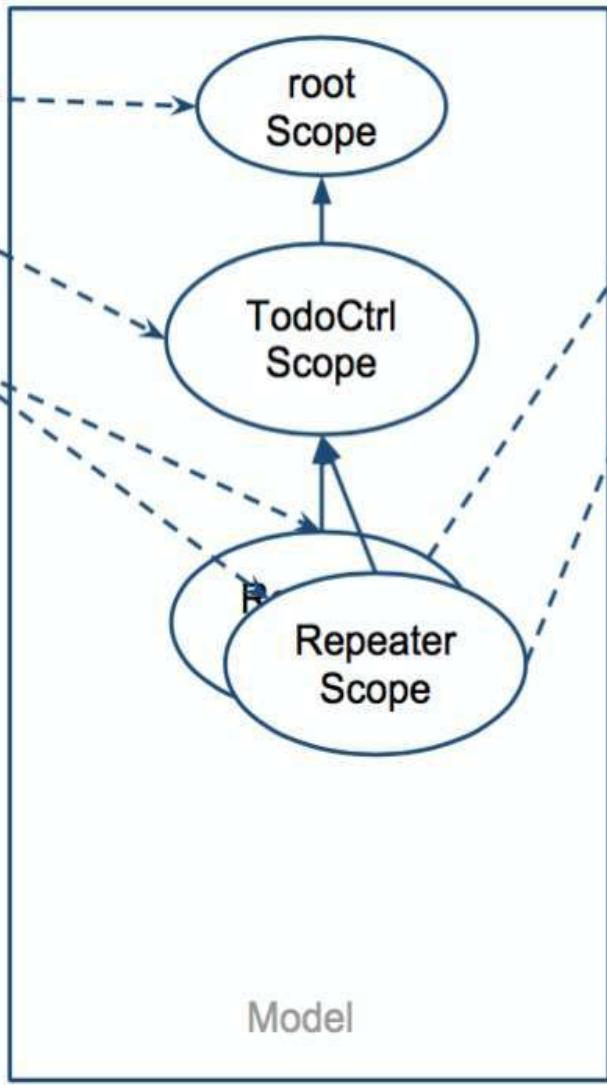


\$scope is an object that can *be used to communicate* between View and Controller

SCOPE

```
<html ng-app>
  -----
<table ng-controller="TodoCtrl">
  <tr ng-repeat="todo in todos">
    <td>{{todo.id}}</td>
  </tr>
</table>
</html>
```

Template



| | | | |
|------|-------|-----------|-------|
| 1001 | false | Groceries | 01/08 |
| 1002 | false | Barber | 01/08 |

View

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="https://ajax.googleapis.com/
    ajax/libs/angularjs/1.4.8/angular.min.js">
  </script>

  </head>
  <body>
    <div data-ng-app="myApp" data-ng-controller="NumberCtrl">
      <p>Number: <input type="number" ng-model="number"></p>
      <p>Number = {{ number }}</p>
      <button ng-click="showNumber()">Show Number</button>
    </div>
    <script>
      var app = angular.module('myApp', []);
      app.controller('NumberCtrl', function($scope) {
        $scope.number = 1;
        $scope.showNumber = function() {
          window.alert( "your number= " + $scope.number );
        };
      });
    </script>
  </body>
</html>
```

Number: 6

Number = 6

Show Number

www.w3schools.com dice:

your number= 6

Impedisci alla pagina di creare altre finestre di dialogo.

OK

MODULES

- **Module** is a reusable container for different features of your app
 - **Controllers**, services, filters, directives...
- If you have a lot of controllers, you are **polluting JS namespace**
- Modules can be loaded in any order
- We can build our **own filters and directives!**

WHEN TO USE CONTROLLERS

- Use controllers
 - set up the initial state of \$scope object
 - add behavior to the \$scope object
- Do not
 - Manipulate DOM (use **databinding, directives**)
 - Format input (use **form controls**)
 - Filter output (use **filters**)
 - Share code or state (use **services**)

APP EXPLAINED

- App runs inside **ng-app** (div)
- AngularJS will invoke the constructor with a \$scope – object
- \$scope is an object that links controller to the view

MODULES, ROUTES, SERVICES

EXAMPLE: OWN FILTER

```
// declare a module

var myAppModule =
  angular.module('myApp', []);

// configure the module.
// in this example we will create a greeting filter
myAppModule.filter('greet', function() {
  return function(name) {
    return      ' + name +
      'Hello   !';
  }
});
```

HTML USING THE FILTER

```
<div ng-app="myApp">  
  <div>  
    {{ 'World' | greet }}  
  </div>  
</div>
```

TEMPLATE FOR CONTROLLERS

```
// Create new module 'myApp' using angular.module  
// method.  
// The module is not dependent on any other module  
var myModule = angular.module('myModule',  
    []);  
  
myModule.controller('MyCtrl', function ($scope) {  
    // Your controller code here!  
});
```

CREATING A CONTROLLER IN MODULE

```
var myModule = angular.module('myModule',  
    []);  
  
myModule.controller('MyCtrl', function ($scope) {  
  
    var model = { "firstname": "Jack",  
                 "lastname": "Smith" };  
  
    $scope.model = model;  
    $scope.click = function () {  
        alert($scope.model.firstname)  
    };  
});
```

```

<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style
      media="screen"></style>
    <script src="../angular.min.js"></script>
    <script>  src="mymodule.js"></script>
  </head>
  <body>
    <div ng-app="myModule"
      <div ng-
        controller="MyCtrl">
          <p>Firstname: <input type="text" ng-
            model="model.firstname"></p>
          <p>Lastname: <input type="text" ng-model="model.lastname"></p>
          <p>{model.firstname + " " + model.lastname}</p>
          <button ng-click="click()">Show Number</button>
        </div>
      </div>
    </body>
  </html>

```

This is now the model object from MyCtrl. Model object is shared with view and controller

ROUTING

ROUTING

- Since **we are building** a SPA app,
everything happens in **one page**
 - How should **back--button** work?
 - How should **linking** between "pages" work?
 - How about **URLs**?
- **Routing** comes to rescue!

```
<html data-ng-app="myApp">
<head>
  <title>Demonstration of Routing -
  index</title>
  <meta charset="UTF-8" />
  <script src="../angular.min.js" type="text/javascript"></script>
  <script src="angular-route.min.js" type="text/javascript"></script>
  <script src="myapp.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-
  view=""></div>
</body>
</html>
```

The content of
this will
change
dynamically

We will have
to load
additional
modules

```
// This module is dependent on ngRoute. Load
ngRoute
// before this.

var myApp = angular.module('myApp', ['ngRoute']);

// Configure routing.

myApp.config(function($routeProvider) {
    // Usually we have different controllers for different views.
    // In this demonstration, the controller does nothing.
    $routeProvider.when('/', {
        templateUrl: 'view1.html',
        controller: 'MySimpleCtrl' });

    $routeProvider.when('/view2', {
        templateUrl: 'view2.html',
        controller: 'MySimpleCtrl'
    });

    $routeProvider.otherwise({ redirectTo: '/' });
});

// Let's add a new controller to MyApp
myApp.controller('MySimpleCtrl', function ($scope)
{
});
```

VIEWS

- **view1.html:**

```
<h1>View 1</h1>  
<p><a href="#" />To View 2</a></p>
```

- **view2.html:**

```
<h1>View 2</h1>  
<p><a href="#" />To View 1</a></p>
```

WORKING IN LOCAL ENVIRONMENT

- If you get "cross origin requests are only supported for HTTP" ..
- Either
 - 1) Disable web security in your browser
 - 2) Use some web server and access files
http://..
- To **disable** web security in chrome
 - taskkill /F /IM chrome.exe
 - "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-web-security --allow-file-access-from-files

SERVICES

- Controller should be very thin;
- Meaning, most of the business logic and persistent data in your application should be taken care of or stored in a services.
- For memory purposes, controllers are instantiated only when they are needed and discarded when they are not.
- Because of this, every time you switch a route or reload a page, Angular cleans up the current controller.
- Services however provide a means for keeping data around for the lifetime of an application while they also can be used across different controllers in a consistent manner.

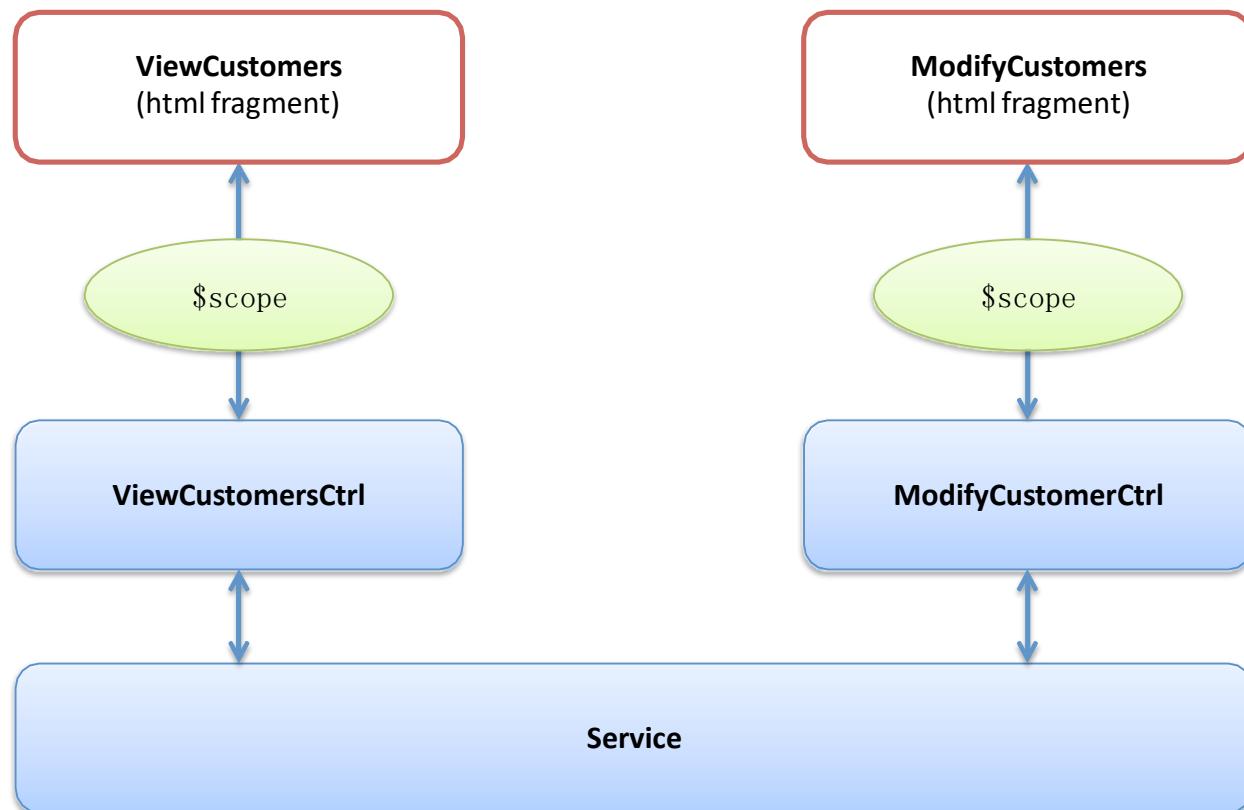
SERVICES

- View--independent **business logic** should **not be** in a controller
 - Logic should be in a **service component**
- **Controllers are view specific, services are app--spesific**
 - We can move from view to view and service is still alive
- Controller's responsibility is to bind model to view.
Model can be fetched from service!
 - Controller is not responsible for manipulating (create, destroy, update) the data. **Use Services instead!**
- AngularJS **has many built-in services**, see
 - <http://docs.angularjs.org/api/ng/service>
 - Example: \$http

SERVICES

- Angular provides us with three ways to create and register our own service.
 - Factory
 - Service
 - Provider

SERVICES



FACTORY

- When you're using a **Factory** you create an object, add properties to it, then return that same object.
- When you pass this service into your controller, those properties on the object will now be available in that controller through your factory.

ANGULARJS CUSTOM SERVICES USING FACTORY

```
// Let's add a new controller to MyApp. This controller uses
Service!  myApp.controller('ViewCtrl', function ($scope,
CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Let's add a new controller to MyApp. This controller uses
Service!  myApp.controller('ModifyCtrl', function ($scope,
CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Creating a factory object that contains services for the
// controllers.
myApp.factory('CustomerService', function()
{
    var factory = {};
    factory.contacts = [{name: "Jack", salary: 3000}, {name:
"Tina", salary: 5000}, {name: "John", salary: 4000}];
    return factory;
});
```

SERVICE

- When you're using **Service**, it's instantiated with the 'new' keyword.
- Because of that, you'll add properties to 'this' and the service will return 'this'.
- When you pass the service into your controller, those properties on 'this' will now be available on that controller through your service.

ALSO SERVICE

```
// Service is instantiated with new - keyword.  
// Service function can use "this" and the return  
// value is this.
```

```
myApp.service('CustomerService', function()  
{    this.contacts      =  
        [ {name: "Jack", salary: 3000},  
          {name: "Tina", salary: 5000},  
          {name: "John", salary: 4000} ];  
});
```

PROVIDERS

- **Providers** are the only service you can pass into your `.config()` function. Use a provider when you want to provide module-wide configuration for your service object before making it available.

ANIMATIONS AND UNIT TESTING

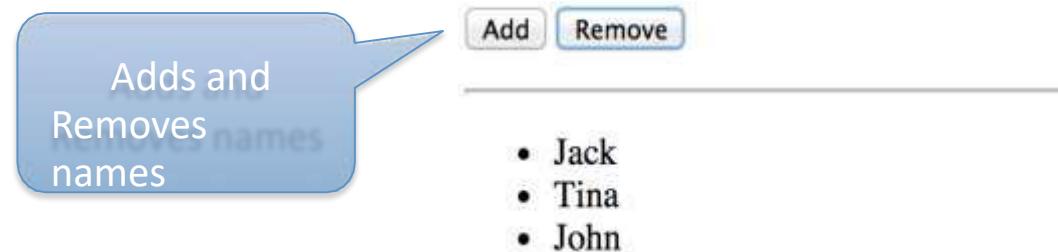
ANGULARJS ANIMATIONS

- Include ngAnimate module as dependency
- Hook animations for common directives such as ngRepeat, ngSwitch, ngView
- Based on CSS classes
 - If HTML element has class, you can animate it
- AngularJS adds special classes to your html-elements

EXAMPLE FORM

```
<bodY ng-controller="AnimateCtrl">  
  <button ng-click="add()">Add</button>  
  <button ng-  
    click="remove()">Remove</button></p>  
  <ul>  
    <li ng-repeat="customer in  
      customers">{ customer.name }</li  
  </ul>  
</body>
```

Animation Test



ANIMATION CLASSES

- When adding a new name to the model, ng-repeat knows the item that is either added or deleted
- CSS classes are added at runtime to the repeated element ()
- When adding new element:
 - <li class="... ng-enter ng-enter-active">New Name
- When removing element
 - <li class="... ng-leave ng-leave-active">New Name

DIRECTIVES AND CSS

| Event | Starting CSS | Ending CSS | Directives |
|-------|--------------|------------------|---|
| enter | .ng-enter | .ng-enter-active | ngRepeat, ngInclude, ngIf, ngView |
| leave | .ng-leave | .ng-leave-active | ngRepeat, ngInclude, ngIf, ngView |
| move | .ng-move | .ng--move.active | ngRepeat |

EXAMPLE CSS

```
/* starting animation */  
.ng-enter {  
  -webkit-transition:  
    1s; transition: 1s;  
  margin-left: 100%;  
}  
  
/* ending animation */  
.ng-enter-active {  
  margin-left: 0;  
}  
  
/* starting animation */  
.ng-leave {  
  -webkit-transition:  
    1s; transition: 1s;  
  margin-left: 0;  
}  
  
/* ending animation */  
.ng-leave-active {  
  margin-left: 100%;  
}
```

TEST DRIVEN DESIGN

- Write tests firsts, then your code
- AngularJS emphasizes modularity, so it can be easy to test your code
- Code can be tested using several unit testing frameworks, like QUnit, Jasmine, Mocha ...

QUNIT

- Download qunit.js and qunit.css
- Write a simple HTML page to run the tests
- Write the tests

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>QUnit Example</title>
    <link rel="stylesheet" href="qunit-
1.10.0.css">
    <script src="qunit-1.10.0.js"></script>
</head>
<body>
    <div id="qunit"></div>
    <script type="text/javascript">

        function calculate(a, b) {
            return a + b;
        }

        test( "calculate test", function()
            ok( calculate(5,5) === "Ok!
10,
"
            );
            ok( calculate(5,0) === "Ok!  );
            5,   ok( calculate(-5,5)
"
            === 0,
"
        );
    </script>
    >
</body>
</html>
```

THREE ASSERTIONS

- Basic
 - `ok(boolean [, message]) ;`
- If *actual == expected*
 - `equal(actual, expected [, message]) ;`
- if *actual === expected*
 - `deepEqual(actual, expected [, message]) ;`
- Other
 - <http://qunitjs.com/cookbook/#automation>
g- unit-testing

**WRAPPING
UP**

WRAPPING UP

- AngularJS is a modular JavaScript SPA framework
- Lot of great features, but learning curve can be hard
- Great for CRUD (create, read, update, delete) apps, but not suitable for every type of apps
- Works very well with some JS libraries (JQuery)

**AJAX +
REST**

AJAX

- **Asynchronous JavaScript + XML**
 - XML not needed, **very often** JSON
- Send data and retrieve asynchronously from server in background
- **Group of technologies**
 - HTML, CSS, DOM, XML/JSON, XMLHttpRequest object and JavaScript

\$HTTP - EXAMPLE (AJAX) AND ANGULARJS

```
<script type="text/javascript">
  var myapp = angular.module("myapp", []);

  myapp.controller("MyController", function($scope, $http) {
    $scope.myData = {};
    $scope.myData.doClick = function(item, event) {
      var responsePromise = $http.get("text.txt");

      responsePromise.success(function(data, status, headers,
        config) {
        $scope.myData.fromServer = data;
      });
      responsePromise.error(function(data, status, headers,
        config) { alert("AJAX failed!");
      });
    }
  });
</script>
```

RESTFUL

- Web Service APIs that adhere to REST architectural constraints are called RESTful
- Constraints
 - Base URI, such as `http://www.example/resources`
 - Internet media type for data, such as JSON or XML
 - Standard HTTP methods: GET, POST, PUT, DELETE
 - Links to reference reference state and related resources

RESTFUL API HTTP METHODS (WIKIPEDIA)

RESTful API HTTP methods

| Resource | GET | PUT | POST | DELETE |
|---|---|--|---|--|
| Collection URI, such as <code>http://example.com/resources</code> | List the URIs and perhaps other details of the collection's members. | Replace the entire collection with another collection. | Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation. ^[17] | Delete the entire collection. |
| Element URI, such as <code>http://example.com/resources/item17</code> | Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type. | Replace the addressed member of the collection, or if it doesn't exist, create it. | Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it. ^[17] | Delete the addressed member of the collection. |

AJAX + RESTFUL

- The web app can fetch using RESTful data from server
- Using AJAX this is done asynchronously in the background
- AJAX makes HTTP GET request using url ..
 - – <http://example.com/resources/item17>
- .. and receives data of item17 in JSON ...
- .. which can be displayed in view (web page)

EXAMPLE: WEATHER API

- Weather information available from wunderground.com
 - You have to **make account** and receive a **key**
- To get Helsinki weather in JSON
 - <http://api.wunderground.com/api/your-key/> conditions/q/Helsinki.json

```
{  
  "response": {  
    "version":  
      "0.1",  
    "termsofService":  
      "http://www.wunderground.com/weather/api/d/terms.html",  
    "features": {  
      "conditions"  
        : 1  
    }  
  },  
  "current_observation"  
  : {  
    "image": {  
      "url":  
        "http://icons.wxug.com/graphics/wu2/logo_130x80.png",  
      "title": "Weather Underground",  
      "link": "http://www.wunderground.com"  
    },  
    "display_location": {  
      "full": "Helsinki,  
Finland",  
      "city":  
        "Helsinki",  
      "state": "",  
      "state_name":  
        "Finland",  
      "country": "FI",  
      "country_iso3166":  
        "FI",  
      "zip":  
        "00000",  
      "magic": "1",  
      "wmo": "02974",  
      "latitude": "60.31999969",  
      "longitude": "24.96999931",  
      "elevation": "56.00000000"  
    }  
}
```

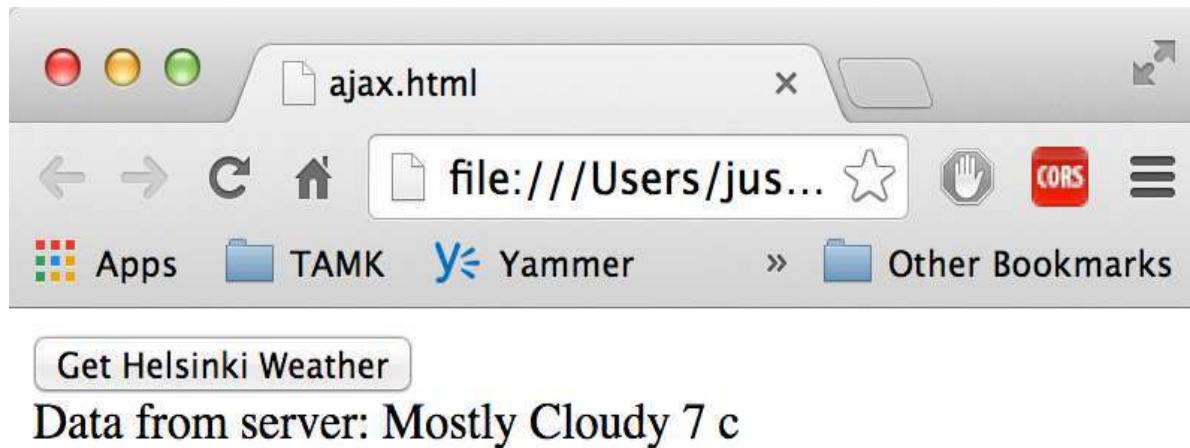
```
<!DOCTYPE html>
<html>
<head>
  <script src="../angular.min.js" type="text/javascript"></script>
  <title></title>
</head>

<body data-ng-app="myapp">
  <div data-ng-controller="MyController">
    <button data-ng-click="myData.doClick(item, $event)">Get Helsinki
      Weather</button><br /> Data from server: {{myData.fromServer}}
  </div>

<script type="text/javascript">
  var myapp = angular.module("myapp",
    []);
  myapp.controller("MyController", function($scope,
    $http) {
    $scope.myData = {};
    $scope.myData.doClick = function(item, event) {
      var responsePromise =
        $http.get("http://api.wunderground.com/api/key/conditions/
          q/Helsinki.json");
      responsePromise.success(function(data, status, headers, config) {
        $scope.myData.fromServer = "" + data.current_observation.weather +
          " " + data.current_observation.temp_c + " c";
      });
      responsePromise.error(function(data, status, headers, config) {
        alert("AJAX failed!");
      });
    }
  );
</script>
</body>
</html>
```

This is
JSON
object!

VIEW AFTER PRESSING THE BUTTON



\$RESOURCE

- Built on top of \$http service, \$resource is a factory that lets you interact with RESTful backends easily
- \$resource does not come bundled with main Angular script, separately download
 - angular-resource.min.js
- Your main app should declare dependency on the ngResource module in order to use \$resource

GETTING STARTED WITH \$RESOURCE

- \$resource expects classic RESTful backend
 - http://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services
- You can create the backend by whatever technology. Even JavaScript, for example Node.js
- We are not concentrating now how to build the backend.

USING \$RESOURCE ON GET

```
// Load ngResource before this
var restApp = angular.module('restApp', ['ngResource']);

restApp.controller("RestCtrl", function($scope, $resource) {
    $scope.doClick = function() {
        var title = $scope.movietitle;
        var searchString =
http://api.rottentomatoes.com/api/public/v1.0/movies.json?apikey=key&q= + title + '&page_limit=5';

        var result = $resource(searchString);

        var root = result.get(function() // {method:'GET'
        {
            $scope.movies = root.movies;
        } });
    });
});
```

Tuntematon

- Tuntematon sotilas (The Unknown Soldier) - 1955
- Tuntematon emanta (The Unknown Woman) - 2011
- The Unknown Soldier (Tuntematon sotilas) - 1985

\$RESOURCE METHODS

- \$resource contains convenient methods for
 - get ('GET')
 - save ('POST')
 - query ('GET', isArray:true)
 - remove ('DELETE')
- Calling these will invoke \$http (ajax call) with the specified http method (GET, POST, DELETE), destination and parameters

PASSING PARAMETERS

```
// Load ngResource before this
var restApp =
angular.module('restApp', ['ngResource']);

restApp.controller("RestCtrl", function($scope, $resource)
{
    $scope.doClick = function() {
        var searchString =
http://apirottentomatoes.com/api/public/v1.0/movies.json?apikey=key&q=:title&page\_limit=5;
        var result = $resource(searchString);
        var root = result.get({title: $scope.movieTitle}, function()
        {
            $scope.movies = root.movies;
        });
    };
});
```

:title →
parametrize
d URL

Giving the
parameter
from
\$scope

USING SERVICES

```
// Load ngResource before this
var restApp =
angular.module('restApp', ['ngResource']);

restApp.controller("RestCtrl", function($scope, MovieService) {
    $scope.doClick = function() {
        var root = MovieService.resource.get({title:
            $scope.movieTitle},
        function() {
            $scope.movies = root.movies;
        });
    }
};

restApp.factory('MovieService',
    function($resource) {    factory = {};
    factory.resource =
    $resource('http://api.rottentomatoes...&q=:title&page_limit=5');
    return factory;
}) ;
```

Controller
responsible
for binding

Service
responsible
for the
resource

SIMPLE VERSION

```
// Load ngResource before this
var restApp =
angular.module('restApp', ['ngResource']);

restApp.controller("RestCtrl",
function($scope,
    $scope.onClick = function() {
        var root =
            MovieService.get({title:
                function() {
            }});
        $scope.movies = root.movies;
    })
);

restApp.factory('MovieService', function($resource) {
    return
        $resource('http://api.rottentomatoes...&q=:title&page_limit=5');
}) ;
```

Just call get from
MovieService

```
MovieService)
{
    $scope.movieTitle
},
```

Returns the
resource

WEBSOCKETS

NEW-AGE WEB

- Evolution of web apps
 - Dynamic and real-time application
 - Webmail, Chat, word processing, etc.
- HTTP is not designed for web apps
 - Large overhead
 - Hanging-GET is necessary for real-time server push

WHAT IS A WEBSOCKET?

- **Communication protocol** used to send / receive data on the Internet
- Like HTTP, but little different... WebSockets are **way** more efficient
- Persistent **2-way** connection between browser <--> server
- Easy to build **real-time** applications:
 - Chat
 - Notifications
 - Online games
 - Financial trading
 - Data visualization dashboards
 - Live maps
 - Collaboration apps

HTTP

Half-duplex (like walkie-talkie)

Traffic flows in **1 direction** at a time



WebSocket

Full-duplex (like phone)

Bi-directional traffic flow



WEBSOCKETS

- New protocol over TCP
- Opening handshake
- HTTP request and response
- Newly defined WebSocket frame
- New API for JavaScript

WEBSOCKETS

- Intended to replace hanging-GET based bidirectional channel
 - Two XMLHttpRequest → One WebSocket
- Full duplex
- Smaller overhead → Fewer TCP connection
- Simpler API

HTTP

Half-duplex (like walkie-talkie)

Traffic flows in **1 direction** at a time

Connection typically **closes** after 1 request / response pair

1. Request from client to server

2. Response from server to client

Headers (**1000s of bytes**)

150ms to establish new TCP connection for each HTTP message

Polling overhead (constantly sending messages to check if new data is ready)

WebSocket

Full-duplex (like phone)

Bi-directional traffic flow

Connection **stays open**

Both client and server are simultaneously “**emitting**” and “**listening**” (.on events)

Uses “frames” (**2 bytes**)

50ms for message transmission

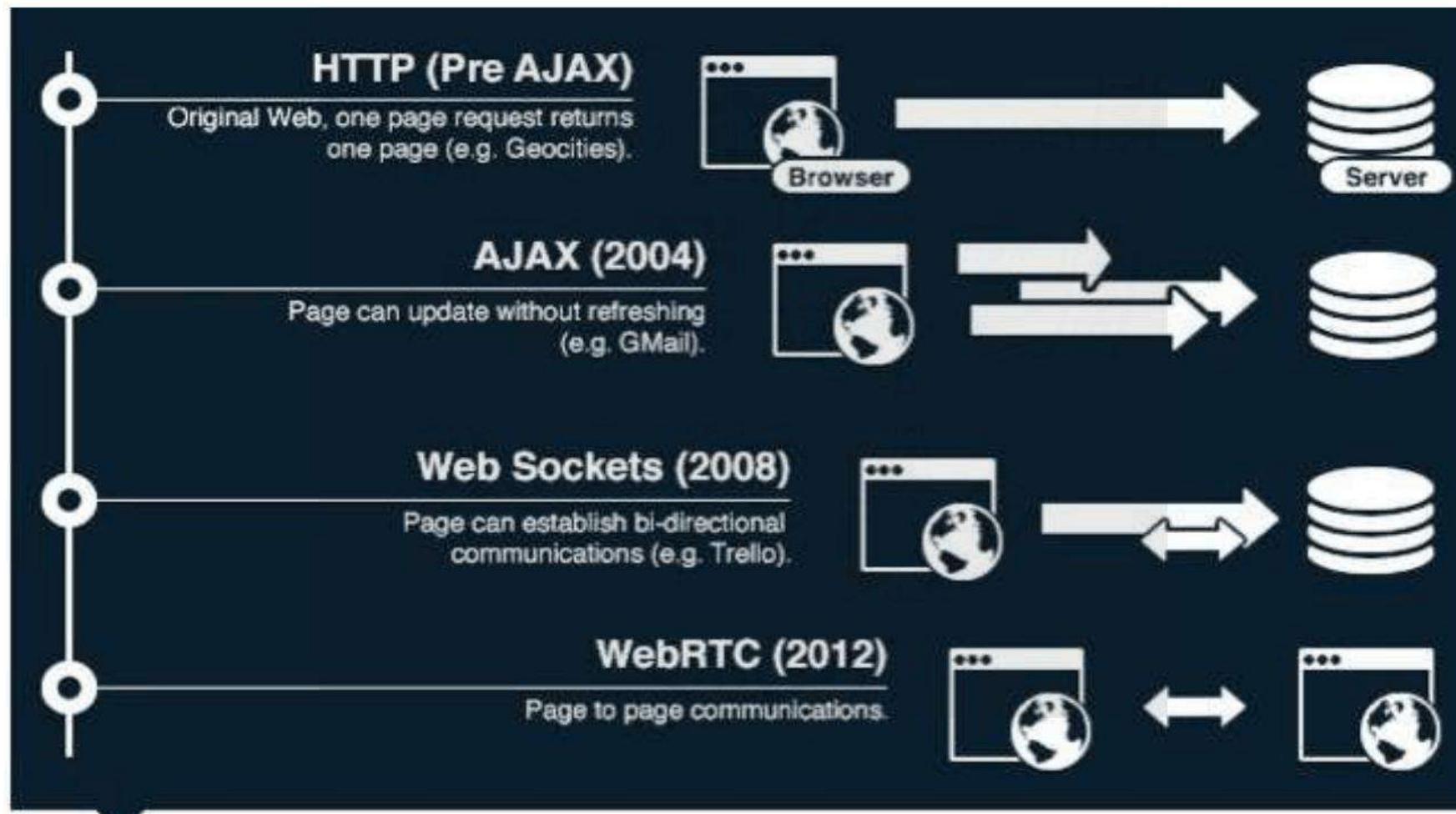
No polling overhead (only sends messages when there is data to send)

LIFE BEFORE WEB SOCKETS...

- Simulate real-time communication with HTTP
- **AJAX:** browser sends requests at regular intervals to check for updates
but headers cause latency and polling is very resource intensive
- **Comet:** server push technique
but complex, non-standardized implementation
- **Streaming:** more efficient than AJAX and Comet
but only 1 direction



HOW DID WE GET HERE?



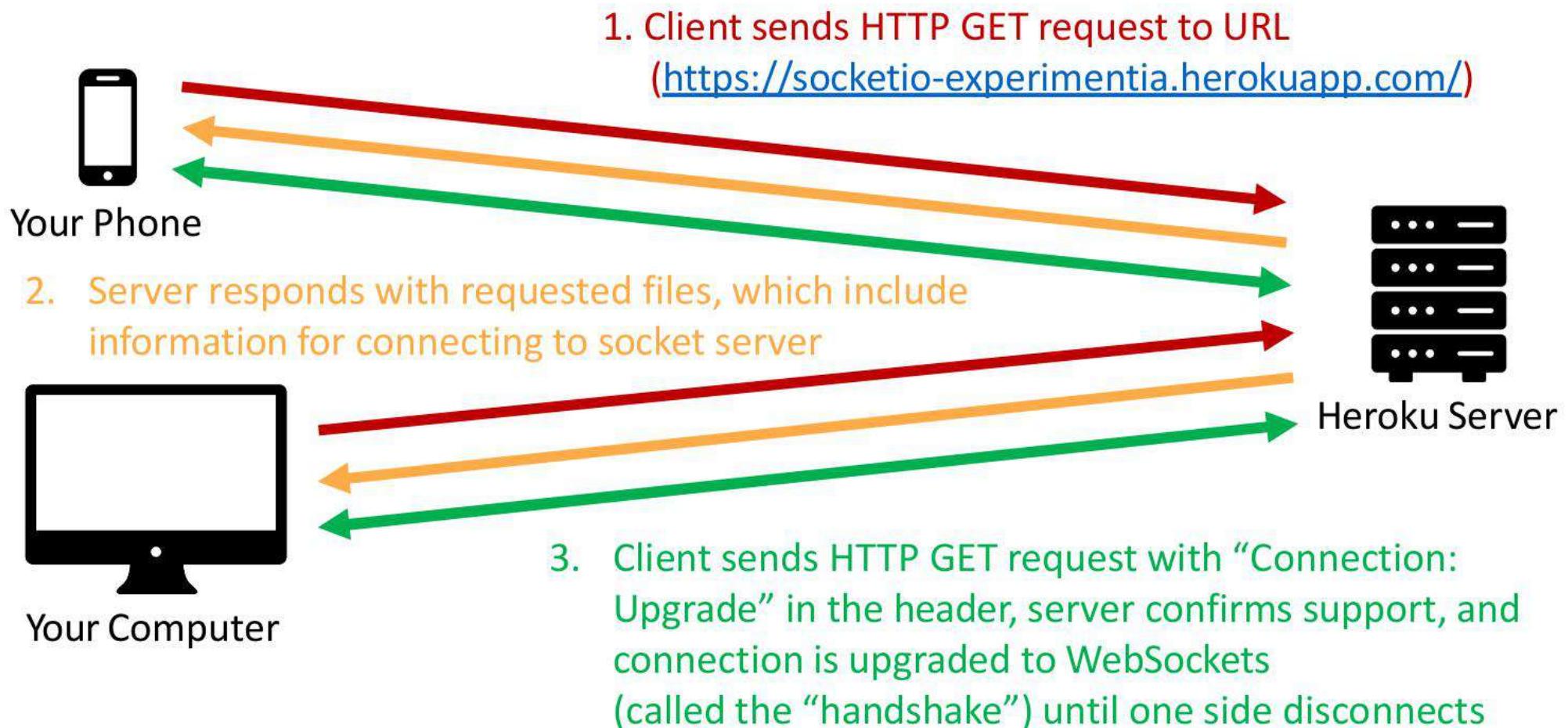
HOW DO WEBSOCKETS WORK?

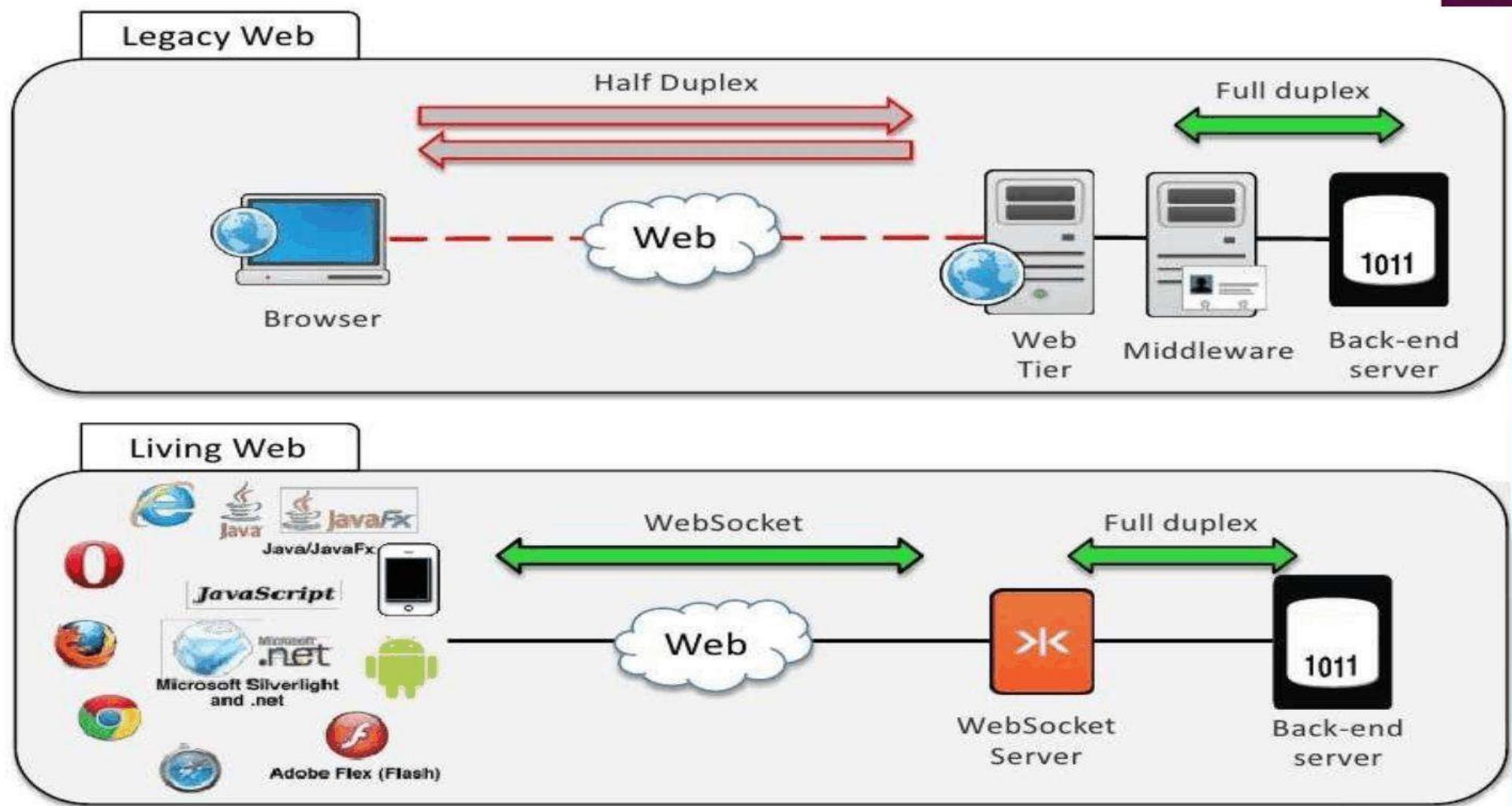


HOW DO WEBSOCKETS WORK?



HOW DO WEBSOCKETS WORK?





WEBSOCKETS - REQUIREMENTS

- Coexist with HTTP on the same port
 - Use 80/443 which are rarely blocked
- Work with HTTP infrastructure
 - Proxy and firewall
- Allow cross origin connection

HTML5 WEBSOCKET SPECIFICATION

- TCP based, bi-directional, full-duplex messaging
- Establish connection (Single TCP connection)
 - Send messages in both direction (Bi-directional)
 - Send message independent of each other (Full Duplex)
- End connection

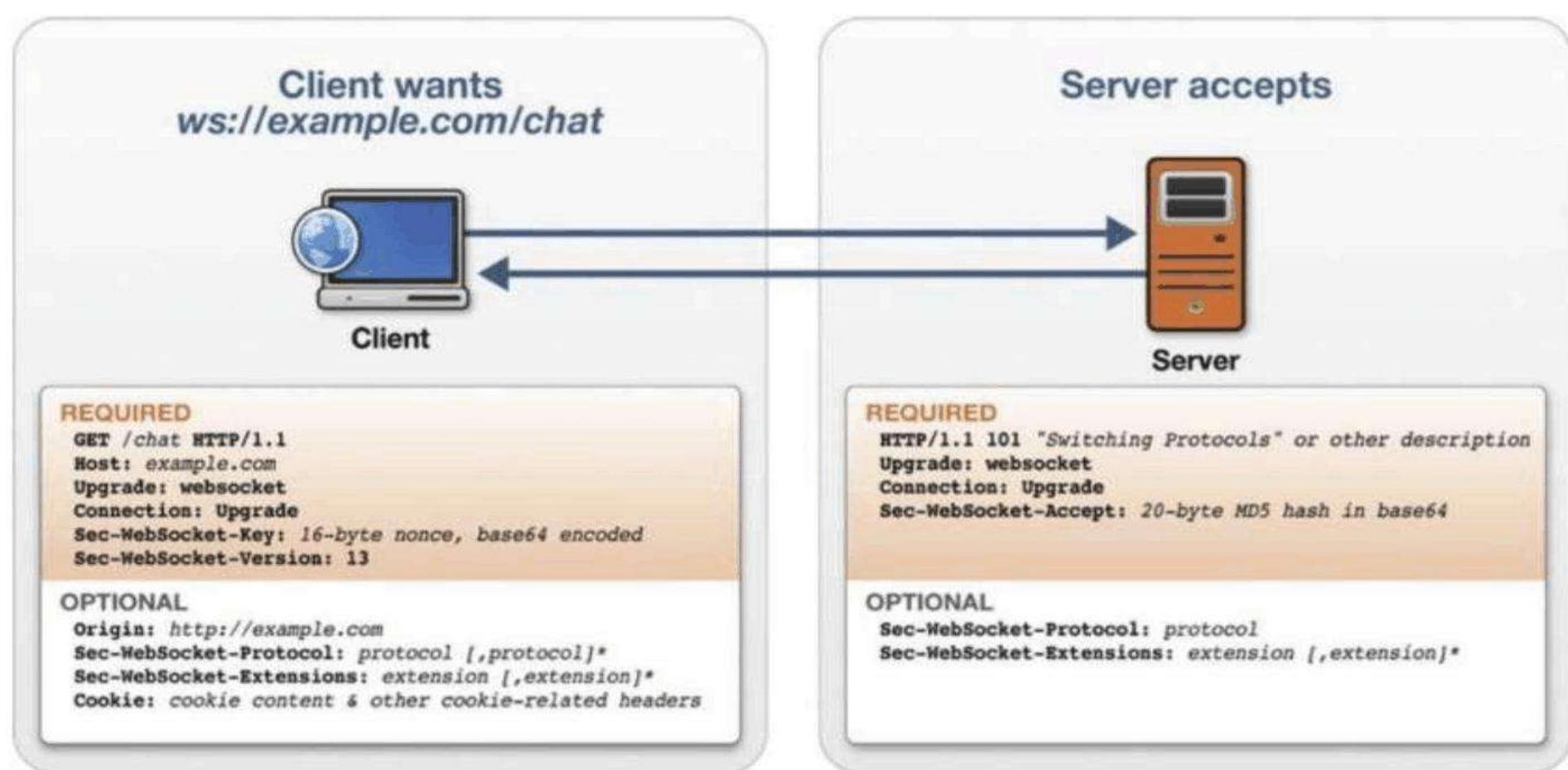
HTML5 WEBSOCKET SPECIFICATION

- Useful for building true real-time functionality into Web applications.

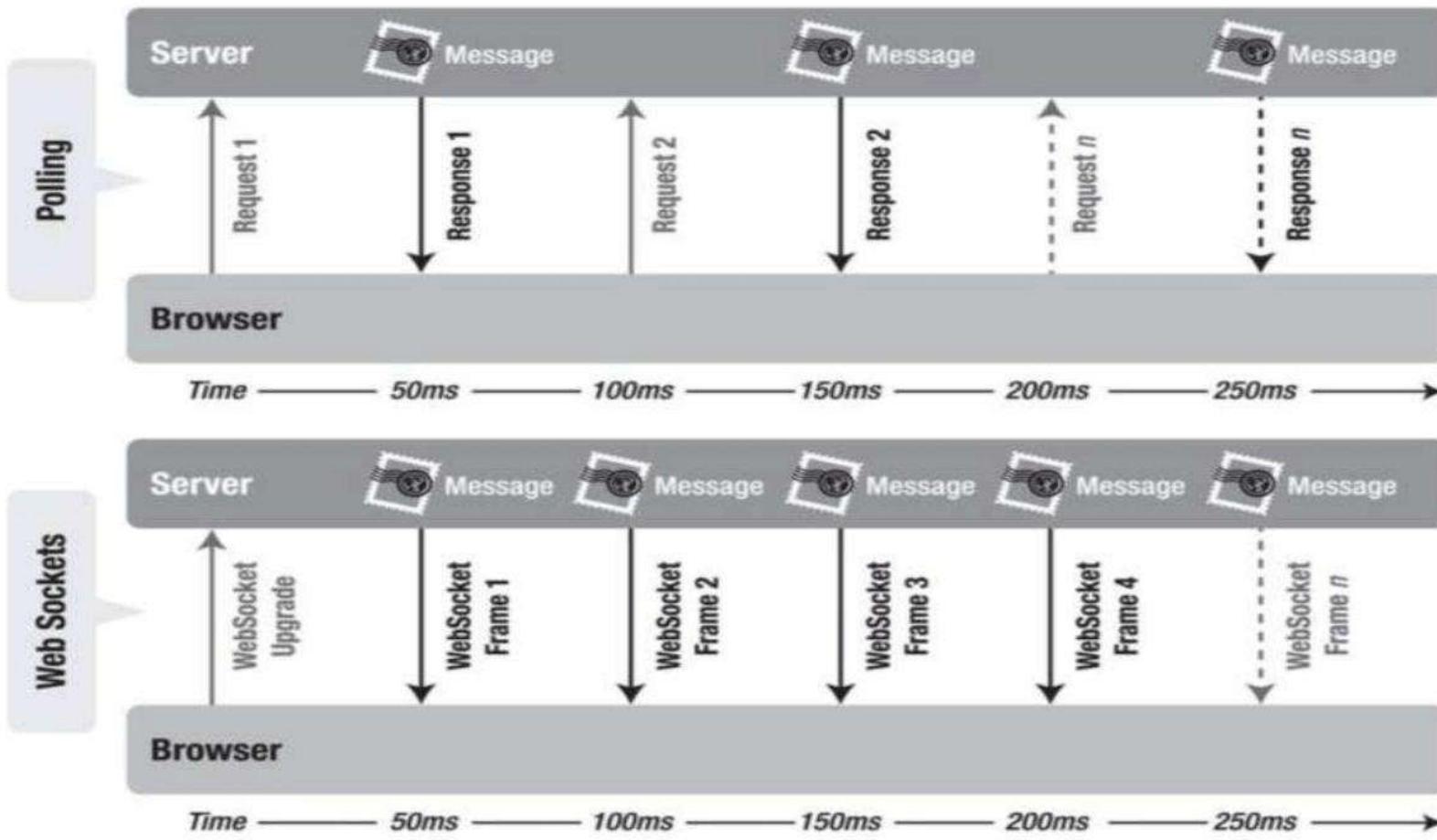
- E.g.

- Online Chat
 - collaborative document editing
 - massively multiplayer online (MMO) games
 - stock trading applications etc.

THE WEBSOCKET HANDSHAKE



LATENCY COMPARISON BETWEEN THE POLLING AND WEBSOCKET APPLICATIONS



WEBSOCKET SMALL HEADER SIZE → EFFICIENCY

Assume HTTP header is 871 bytes (some are 2000bytes)

- Use case A: 1,000 clients polling every second: Network traffic is $(871 \cdot 1,000) = 871,000$ bytes = 6,968,000 bits per second (6.6 Mbps)

- Use case B: 10,000 clients polling every second: Network traffic is $(871 \cdot 10,000) = 8,710,000$ bytes = 69,680,000 bits per second (66 Mbps)

- Use case C: 100,000 clients polling every 1 second: Network traffic is $(871 \cdot 100,000) = 87,100,000$ bytes = 696,800,000 bits per second (665 Mbps)

- Use case A: 1,000 clients receive 1 message per second: Network traffic is $(2 \cdot 1,000) = 2,000$ bytes = 16,000 bits per second (0.015 Mbps)

- Use case B: 10,000 clients receive 1 message per second: Network traffic is $(2 \cdot 10,000) = 20,000$ bytes = 160,000 bits per second (0.153 Mbps)

- Use case C: 100,000 clients receive 1 message per second: Network traffic is $(2 \cdot 100,000) = 200,000$ bytes = 1,600,000 bits per second (1.526 Mbps)

Regular HTTP

HTTP with websocket

WEBSOCKET PROTOCOL - STATUS CODES

- Uses 4-digit codes in contrast to HTTP's 3-digit codes.
- Predefined codes
 - 1000 Normal closure
 - 1001 Peer is going away
 - 1002 Protocol error
 - 1003 Received unacceptable data
 - 1004 Too large message

IS THERE ANYTHING BAD ABOUT WEBSOCKETS?

- Not all browsers support WebSockets
- Different browsers treat WebSockets differently

SOCKET.IO

- **2 JavaScript libraries:**
 - socket.io-client (front-end)
 - socket.io (back-end using NodeJS)
- **Cross-browser compatibility** by automatically using the best protocol for the user's browser
 - WebSockets
 - Comet
 - Flash



socket.io

SOCKET.IO SERVER CONFIGURATION

```
① // add the HTTP server
① var http = require('http');
① // add Express server framework for NodeJS
var express = require('express');
① // add Socket.io server framework
var socketIO = require('socket.io');
① // create instance of Express
① var app = express();
① // create Express HTTP server
① var server = http.createServer(app);
① // tell Express HTTP server which port to run on
server.listen(8080);
// tell Socket.io to add event listeners to Express HTTP server
var io = socketIO().listen(server);
```

The diagram illustrates the sequential steps in the code:

1. Create HTTP server: Points to the line `var server = http.createServer(app);`
2. Add WebSocket support to HTTP server: Points to the line `var io = socketIO().listen(server);`

SOCKET.IO SERVER CODE

```
// listens for new socket connections from clients
// triggers a callback function when 'connection' event occurs
io.sockets.on('connection', function(socket) {
    // do stuff (ex. keep track of # of socket connections)
    connections.push(socket);
    // emit / broadcast custom event and data (payload) to client
    io.sockets.emit('updateStudents', payload);
}

// listen for custom events "emitted" by client
socket.on('join', function(payload) {
    var newStudent = {
        socketID: this.id,
        name: payload.name
    }
    this.emit('joined', newStudent);
})
```

1. Listen 2. Emit 1. Listen 2. Emit

SOCKET.IO CLIENT CODE

```
// add Socket.io client framework
var io = require('socket.io-client');

// add Socket.io client framework
this.socket = io('http://localhost:3000');

// listen for socket connection from server
this.socket.on('connect', function() {
    var newStudent = {name: nameFromForm, type: "student"};
    // emit custom event with data (newStudent) back to server
    this.emit('join', newStudent);
})

// listen for custom events with data (payload) "emitted" by server
this.socket.on('joined', function(payload) {
    // do stuff with payload...
})
```

1. Listen ←

2. Emit ←

MORE ABOUT WEBSOCKETS

- RFC 6455 - The WebSocket Protocol - IETFTools
 - <https://tools.ietf.org/html/rfc6455>
- Introducing WebSockets: Bringing Sockets to the Web
 - <http://www.html5rocks.com/en/tutorials/websockets/basics/>
- WebSocket.org
 - <https://www.websocket.org/>
 - <https://www.websocket.org/demos.html>

KEY REFERENCES

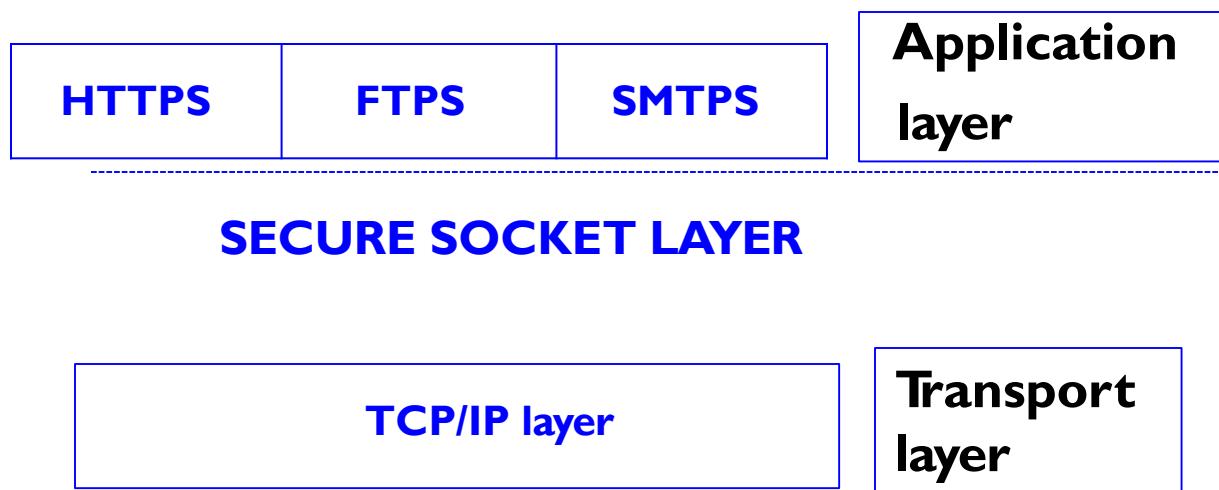
- <https://www.lynda.com/Web-Development-tutorials/Building-Polling-App-Socket-IO-React-js/387145-2.html>
- <https://socket.io/get-started/chat/>
- <http://www.jonahnisenson.com/what-are-websockets-and-why-do-i-need-socket-io/>
- <https://nodesource.com/blog/understanding-socketio/>
- http://enterprisewebbook.com/ch8_websockets.html
- <http://blog.teamtreehouse.com/an-introduction-to-websockets>
- <https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/>

Web Protocols

HTTPS

HTTPS

- ▶ Acronym for HTTP over SSL, HTTP over TLS and HTTP Secure.
- ▶ Utilizes the Secure Sockets Layer meta-protocol over TCP/IP.



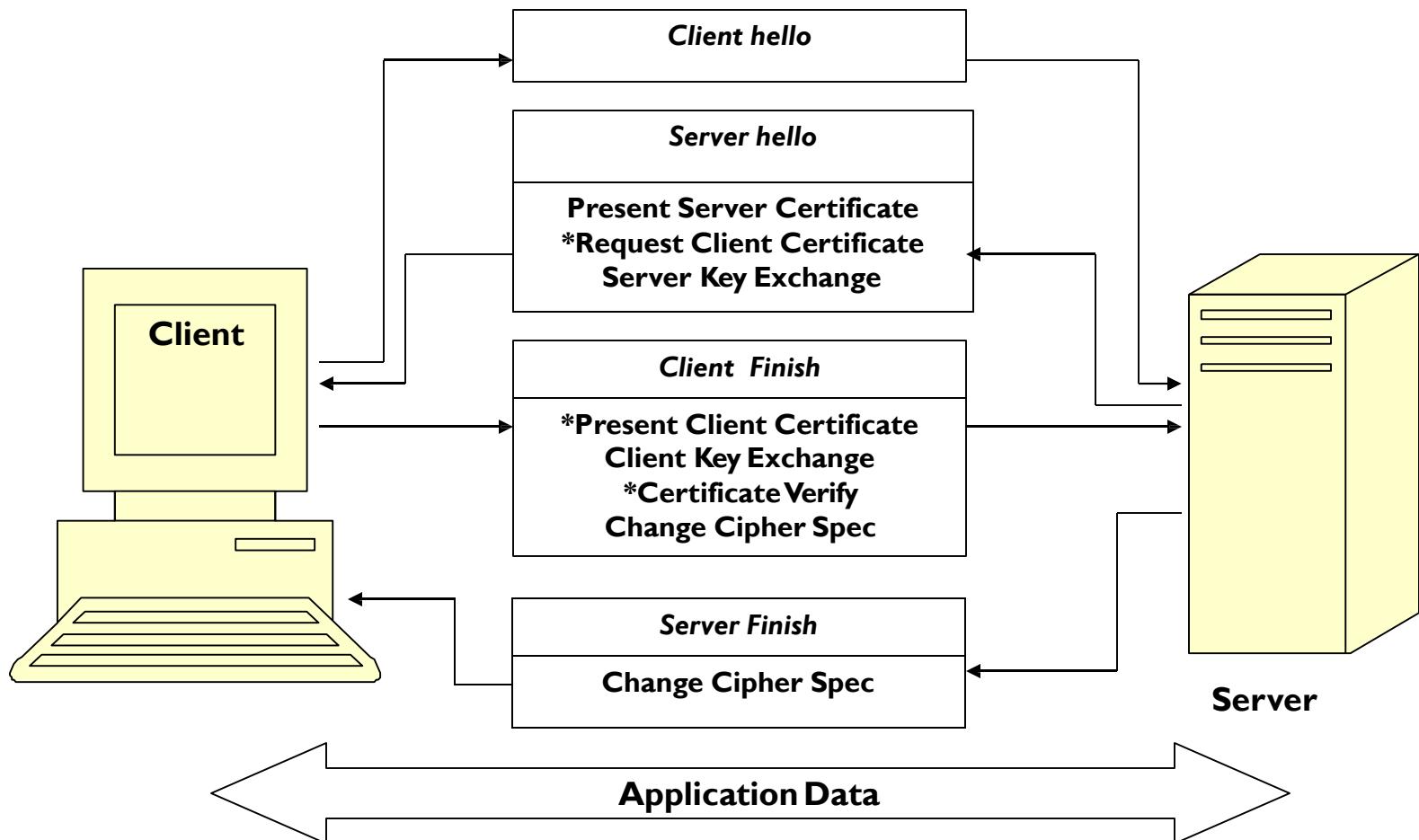
- ▶ SSL/TLS connection uses a dedicated TCP/IP socket (e.g. port 443 for https)

HTTPS - OBJECTIVES

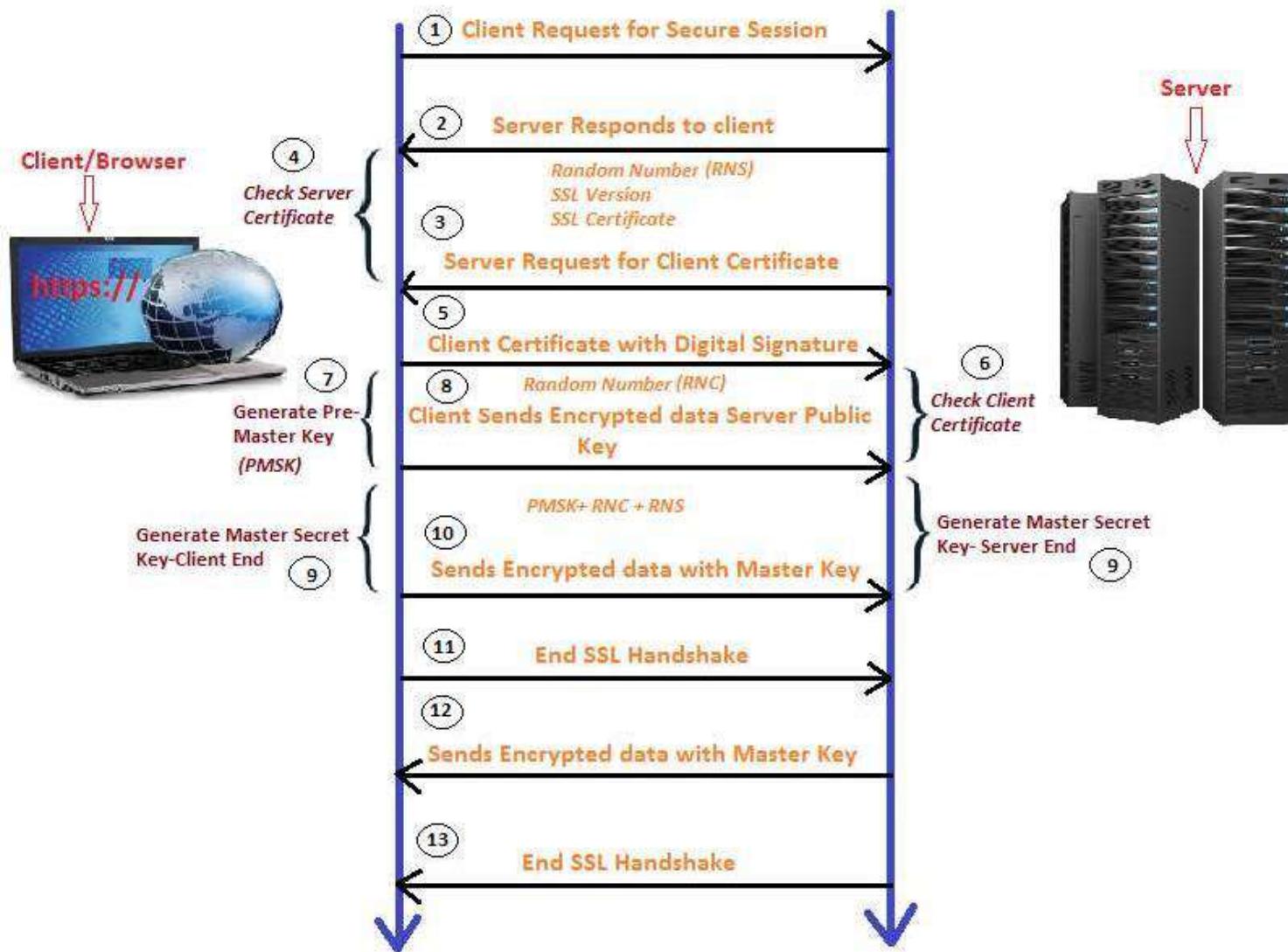
- ▶ **Authentication** of the web site and associated web server
 - ▶ preventing Man-in-the-middle attacks
- ▶ **Integrity:** Bidirectional encryption of communications between a client and server,
 - ▶ protect against tampering with and/or forging the contents of communication.
- ▶ **Confidentiality:** ensuring that communication remains confidential.
- ▶ **Verification and Non-repudiation:** reasonable guarantee that one is communicating with precisely the web site that one intended to communicate with (as opposed to an impostor)

SSL

HANDSHAKE



WORKING OF HTTPS



WORKING OF HTTPS

- ▶ A browser requests a secure page (usually https://).
 - ▶ The web server sends its public key with its certificate.
 - ▶ Browser checks that the certificate was issued by a trusted party (usually a trusted root CA), that the certificate is still valid and that the certificate is related to the site contacted.
 - ▶ Browser then uses the public key, to encrypt a random symmetric encryption key and sends it to the server with the encrypted URL required as well as other encrypted http data.
 - ▶ Web server decrypts the symmetric encryption key using its private key and uses the symmetric key to decrypt the URL and http data.
 - ▶ Web server sends back the requested html document and http data encrypted with the symmetric key.
 - ▶ Browser decrypts the http data and html document using the symmetric key and displays the information.
-

CERTIFICATE AUTHORITY (CA)

- ▶ a trusted third party organization that is used by both interacting parties.
 - ▶ issues digital certificates.
- ▶ Digital certificate –
 - ▶ certifies the ownership of a public key by the subject named on the certificate.
 - ▶ allows others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the public key that is certified.
- ▶ Some CA companies –Verisign, Thawte, GlobalSign, GeoTrust etc.

CERTIFICATE AUTHORITY (CA)

- ▶ Certification Authority Functions:
 - ▶ Accept applications for certificates.
 - ▶ Thoroughly verify the identity of the person/ organization etc applying for the certificate.
 - ▶ Issue certificates.
 - ▶ Revoke/Expire certificates.
 - ▶ Provide status information about the certificates that it has issued.

DIGITAL CERTIFICATES

- ▶ A digital file that certifies the identity of an individual or institution, or even a router seeking access to digital information stored on a computer/device.
- ▶ issued by a Certification Authority, and serves the same purpose as a driver's license or a passport.

DIGITAL CERTIFICATES (CONTD.)

- ▶ Contents -
 - ▶ name of the certificate holder,
 - ▶ a serial number,
 - ▶ expiration dates,
 - ▶ a copy of the certificate holder's public key (used for encrypting messages and digital signature)
 - ▶ the digital signature of the certificate-issuing authority (CA) so that a recipient can verify that the certificate is real.

DIGITAL CERTIFICATES (CONTD.)

- ▶ Four main types of digital certificates:

1. Server Certificates
2. Personal Certificates
3. Organization Certificates
4. Developer Certificates

DIGITAL CERTIFICATES (CONTD.) BASIC CERTS



NITK SURATHKAL

You must have an account to proceed further

By logging in below, it is deemed that the user is signing [this declaration \(PDF\)](#).
See [IT-Policy \(PDF\)](#) ::: [FAQ](#) ::: [HelpMe](#)

- All elements on the page fetched using HTTPS (with some exceptions)
- For all elements:
 - HTTPS cert is issued by a CA trusted by *browser*
 - HTTPS cert is *valid* (e.g. not expired)
 - *CommonName* in cert matches domain in URL.

THE LOCK UI:

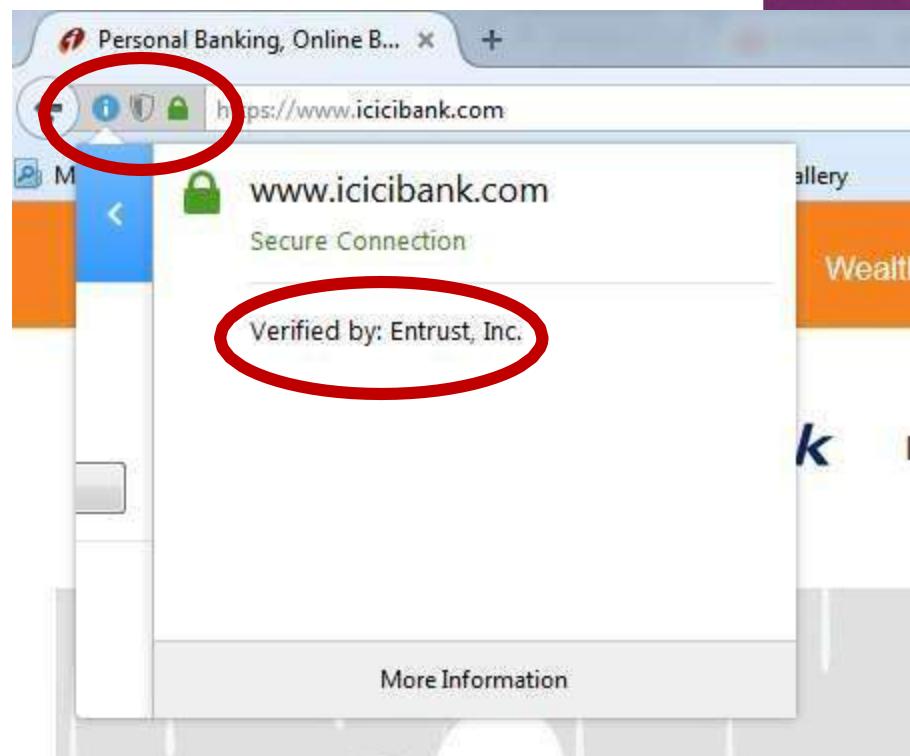
help users authenticate site

- ▶ Firefox 3:

(no SSL)

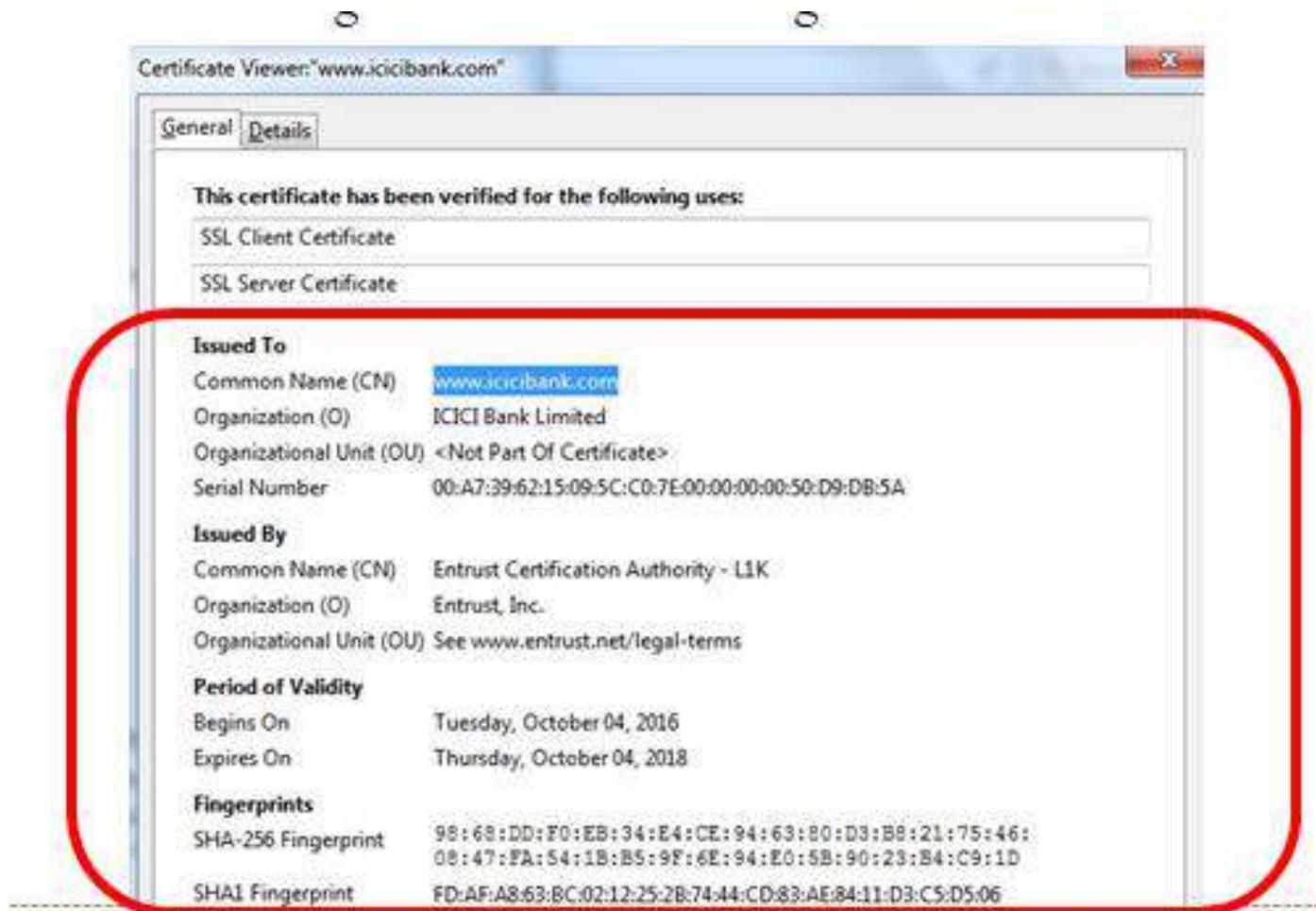


(SSL)



The lock UI: help users authenticate site

- ▶ Firefox 3: clicking on bottom lock icon gives additional info...



DIGITAL CERTIFICATES (CONTD.)

EXTENDED VALIDATION (EV) CERTS

- ▶ Harder to obtain than regular certs
 - ▶ requires human lawyer at CA to approve cert request
- ▶ Designed for banks and large e-commerce sites
 - ▶ Helps block “semantic attacks”: www.icicibank.com



Security Overview



This page is secure (valid HTTPS).

Valid Certificate

The connection to this site is using a valid, trusted server certificate.

[View certificate](#)

Secure TLS connection

The connection to this site is using a strong protocol version and cipher suite.

Secure Resources

All resources on this page are served securely.

Certificate

General Details Certification Path



Certificate Information

This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer
- Proves your identity to a remote computer

* Refer to the certification authority's statement for details.

Issued to: *.paytm.com

Issued by: GeoTrust SSL CA - G3

Valid from 13- 10- 2015 **to** 27- 08- 2017

[Issuer Statement](#)

Learn more about [certificates](#)

OK

HTTP VS. HTTPS...

- ▶ HTTP URLs begin with "http://" and use port 80 by default.
 - ▶ HTTPS URLs begin with "https://" and use port 443 by default
- ▶ HTTP is insecure and is subject to man-in-the-middle and eavesdropping attacks, which can let attackers gain access to website accounts and sensitive information.
 - ▶ HTTPS is designed to withstand such attacks and is considered secure against such attacks (with SSL2.0 and above).
- ▶ HTTPS is marginally slower than HTTP. When large amounts of data are processing over a port, performance can degrade**.

Web Services

List of Topics

- What are Web Services ?
- How it works ?
- Why Web Services?
- Architecture
- Components
- Security
- How Web Service is implemented?
- Web services samples
- REST

What are web services?

- Web Services can convert your application into a Web-application which can publish its function or message to the rest of the world.
- The basic Web Services platform is XML + HTTP
- A application which run on web (Internet or Intranet) and provides generic services
- The services provided are through the web and in a standardized format which makes it generic and independent on the platform or the protocol on which the service was requested.

What are Web Services cont...

- Web services are open standard (XML, SOAP, HTTP etc.) based Web applications that interact with other web applications for the purpose of exchanging data
- There are two types of Webservices
 - **SOAP** (JAX-WS , Java API for XML Web Services)
 - **REST** (JAX-RS, Java API for RESTful Web Services)

To summarize, a complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

Why Web services ?

- **Exposing the existing function on to network:**

A Web service is a unit of managed code that can be remotely invoked using HTTP, that is, it can be activated using HTTP requests. So, Web Services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other application can use the functionality of your program.

- **Connecting Different Applications i.e. Interoperability:**

Web Services allows different applications to talk to each other and share data and services among themselves. Other applications can also use the services of the web services. For example VB or .NET application can talk to java web services and vice versa. So, Web services is used to make the application platform and technology independent.

Why Web services continued...

- **Standardized Protocol:**

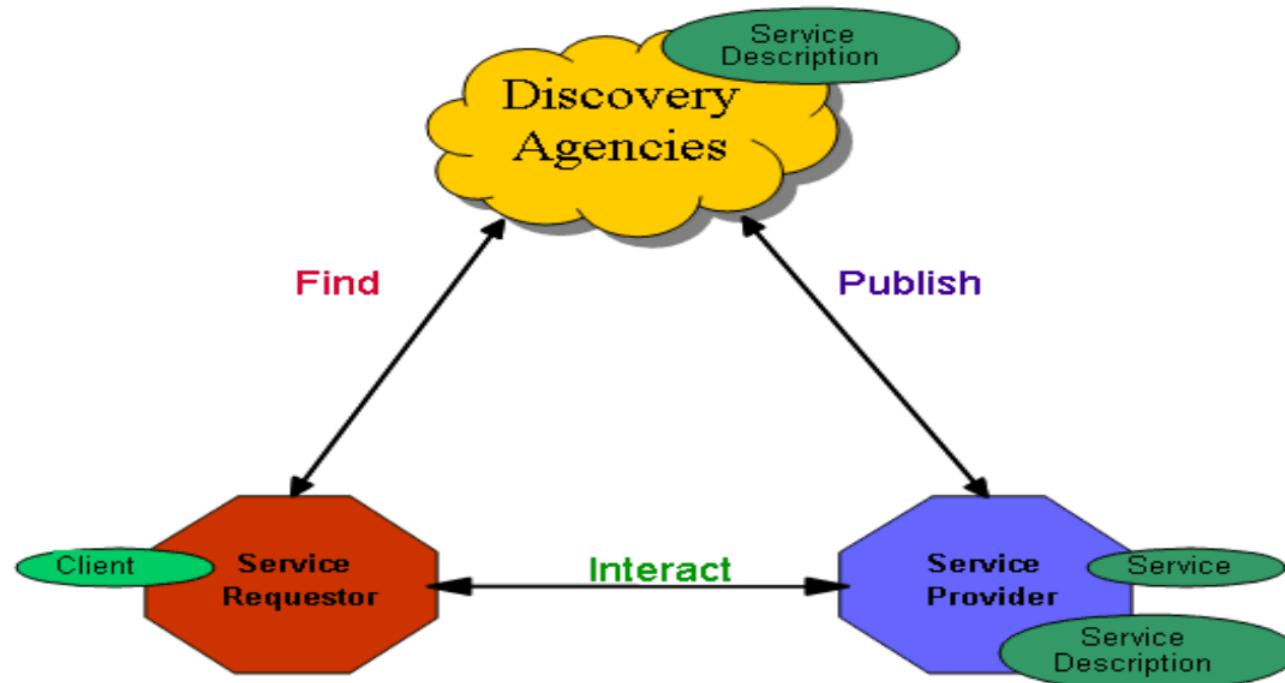
Web Services uses standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description and Service Discovery layers) uses the well defined protocol in the Web Services protocol stack. This standardization of protocol stack gives the business many advantages like wide range of choices, reduction in the cost due to competition and increase in the quality.

- **Low Cost of communication:**

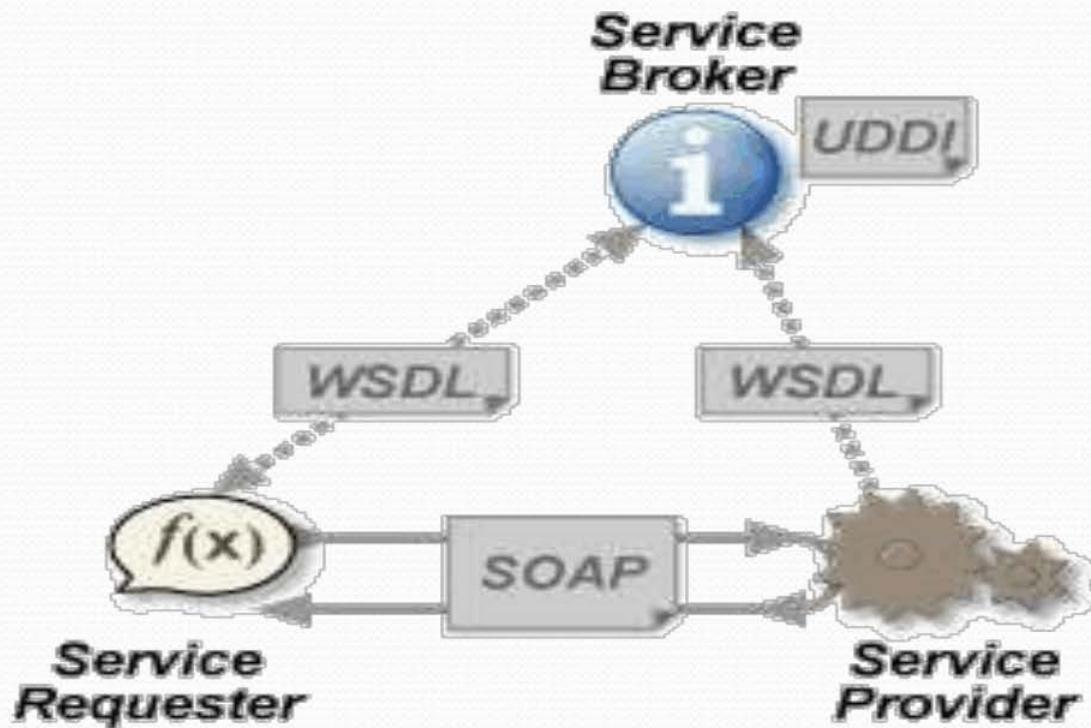
Web Services uses SOAP over HTTP protocol for the communication, so you can use your existing low cost internet for implementing Web Services. Beside SOAP over HTTP, Web Services can also be implemented on other reliable transport mechanisms like FTP etc.

Architecture

Service Oriented Architecture



Architecture cont...

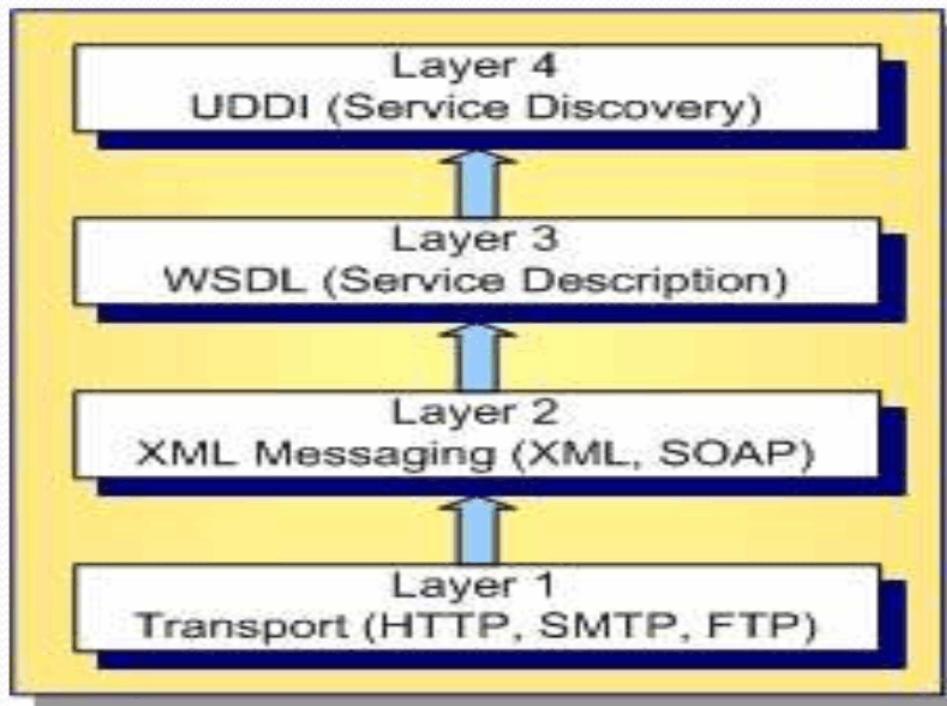


Web service roles

There are three major roles within the web service architecture

- **Service provider:** This is the provider of the web service. The service provider implements the service and makes it available on the Internet.
- **Service requestor:** This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.
- **Service registry:** This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones.

Web Service protocol stack



Protocol stack cont...

- **Service transport:** This layer is responsible for transporting messages between applications. Currently, this layer includes hypertext transfer protocol (HTTP), Simple Mail Transfer Protocol (SMTP), file transfer protocol (FTP), and newer protocols, such as Blocks Extensible Exchange Protocol (BEEP).
- **XML messaging:** This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.
- **Service description:** This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).
- **Service discovery:** This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

Components of Web Service

- SOAP (Simple Object Access Protocol)
 - Protocol based on XML , used for message transfer
- WSDL (Web Service Description Language)
 - XML file used to describe the Web service and how to access them
- UDDI (Universal Description and Discovery Integration)
 - Used to register and search for web service
 - Directory of web service
- JAX-RPC
 - For intercommunication
- HTTP
 - For message transfer

What is SOAP?

SOAP is an XML-based protocol to let applications exchange information over HTTP.

Or more simple: SOAP is a protocol for accessing a Web Service.

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C standard

What is WSDL?

WSDL is an XML-based language for locating and describing Web services.

- WSDL stands for Web Services Description Language
- WSDL is based on XML
- WSDL is used to describe Web services
- WSDL is used to locate Web services
- WSDL is a W3C standard

What is UDDI?

UDDI is a directory service where companies can register and search for Web services.

- UDDI stands for Universal Description, Discovery and Integration
- UDDI is a directory for storing information about web services
- UDDI is a directory of web service interfaces described by WSDL
- UDDI communicates via SOAP

XML-RPC

This is the simplest XML based protocol for exchanging information between computers.

- XML-RPC is a simple protocol that uses XML messages to perform RPCs.
- Requests are encoded in XML and sent via HTTP POST.
- XML responses are embedded in the body of the HTTP response.
- XML-RPC is platform-independent.
- XML-RPC allows diverse applications to communicate.
- XML-RPC is the easiest way to get started with web services.

Security

- **Confidentiality**

If a client sends an XML request to a server, then question is that can we ensure that the communication remains confidential?

Answer lies here

- XML-RPC and SOAP run primarily on top of HTTP.
- HTTP has support for Secure Sockets Layer (SSL).
- Communication can be encrypted via the SSL.
- SSL is a proven technology and widely deployed.

- **Authentication**

If a client connects to a web service, how do we identify the user? And is the user authorized to use the service?

Following options can be considered but there is no clear consensus on a strong authentication scheme.

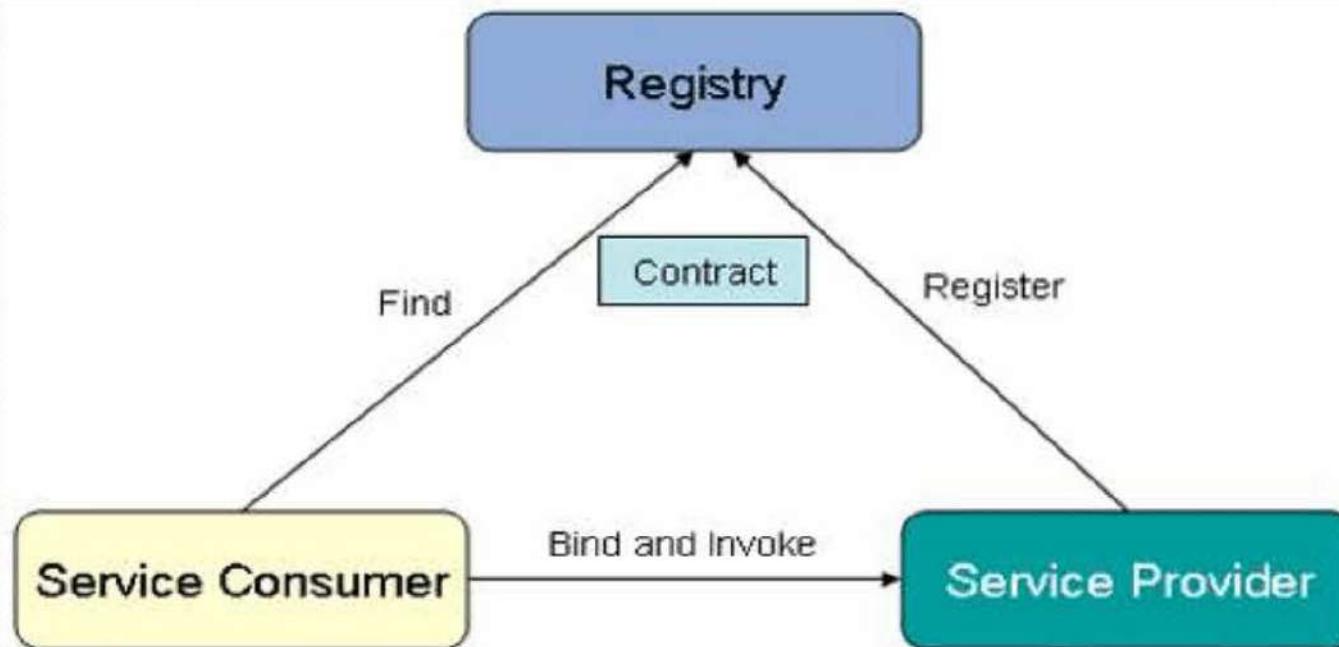
- HTTP includes built-in support for Basic and Digest authentication, and services can therefore be protected in much the same manner as HTML documents are currently protected.

Security cont...

- SOAP Security Extensions: Digital Signature (SOAP-DSIG). DSIG leverages public key cryptography to digitally sign SOAP messages. This enables the client or server to validate the identity of the other party.
- The Organization for the Advancement of Structured Information Standards (OASIS) is working on the Security Assertion Markup Language (SAML, It is an XML-based open standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider).
- **Network Security**
There is currently no easy answer to this problem, and it has been the subject of much debate. For now, if you are truly intent on filtering out SOAP or XML-RPC messages, one possibility is to filter out all HTTP POST requests that set their content type to text/xml.

How Web Service is implemented ?

- Build and Publish
- Find
- Bind and Invoke



Web Service Implementation Cont...

- **Pre Conditions**
 - No one will be able to use your service
- **Step 1 :Build and Publish**
 - *Build*
 - Create your application
 - Create your contract file (WSDL)
 - *Publish*
 - Register your application as a web service onto any registry
 - This process happens on UDDI using a separate SOAP request
 - This process is useful only if your web service should be accessible using Internet
- **Post conditions**
 - Your web service will be available to the Public (If published to any registry) or will be accessible to intranet users

Web Service Implementation Cont...

- **Pre Conditions**

- The user needs to access a service but is not aware of the service details

- **Step 2 :Find**

- Find
 - Search in the registry for a service which provides your needs
 - Obtain the necessary details about the service

- **Post conditions**

- The user will have all the details about the service and gets ready to contact the service

Web Service Implementation Cont...

- **Pre Conditions**

- The user will have the contract necessary to identify and call the service

- **Step 3 :Bind**

- Bind
 - Use the contract file to build the request message.
 - Invoke
 - Send a request to the service and request for the necessary operation available from the service.
 - The request should be sent in the protocol which is required by the service

- **Post conditions**

- The user will receive the response from the service in a format specified in the contract file.

Web Services samples

- Eclipse tutorial link :

<http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/index.html>

- Bottom to Top approach
 - JAVA to WSDL
- Top to bottom approach
 - WSDL to JAVA

WEB 1.0 TO WEB 3.0 - EVOLUTION OF THE WEB AND ITS VARIOUS CHALLENGES

OUTLINE

- Difference Between Web & Internet.
- Web 1.0(Read-only Static web).
- Sad Facts of Web 1.0.
- Web 2.0(Read-write interactive web).
- Principles of Web 2.0.
- Web 1.0 vs Web 2.0.
- Sad Facts of Web 2.0.
- Web 3.0 (Read-write intelligent web).
- Technologies of Web 3.0
- Comparison Among Existing Web
-



Before describing web we need to understand one thing very clearly

INTRODUCTION

World Wide Web ≠ Internet Service

We must remember that both are not the same
Web is different than Internet



What is the Web?

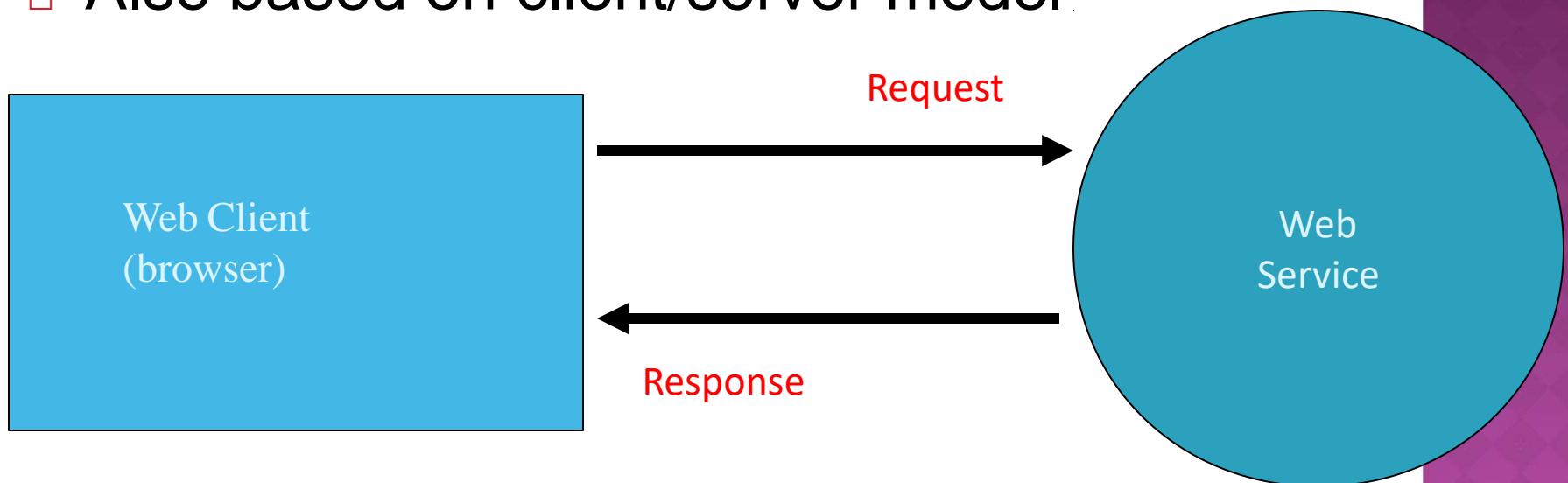
WORLD WIDE WEB

- The world wide web is larger collection of interconnected Documents or Content
- Facilitates communication between peopleand also computers



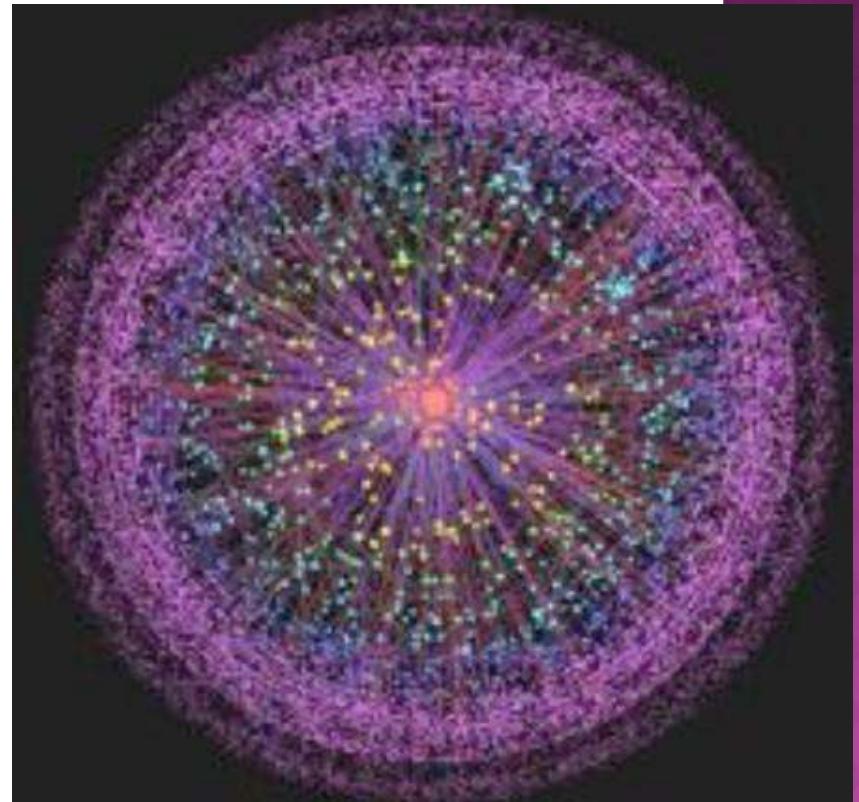
CONTD.

- Web based on Hypertext
- Also based on client/server model



INTERNET

The Internet is the collection of interconnected computer Networks.



WEB 1.0

Web 1.0 [Push]

Web 1.0 is an old internet that only allows people to read from the internet.



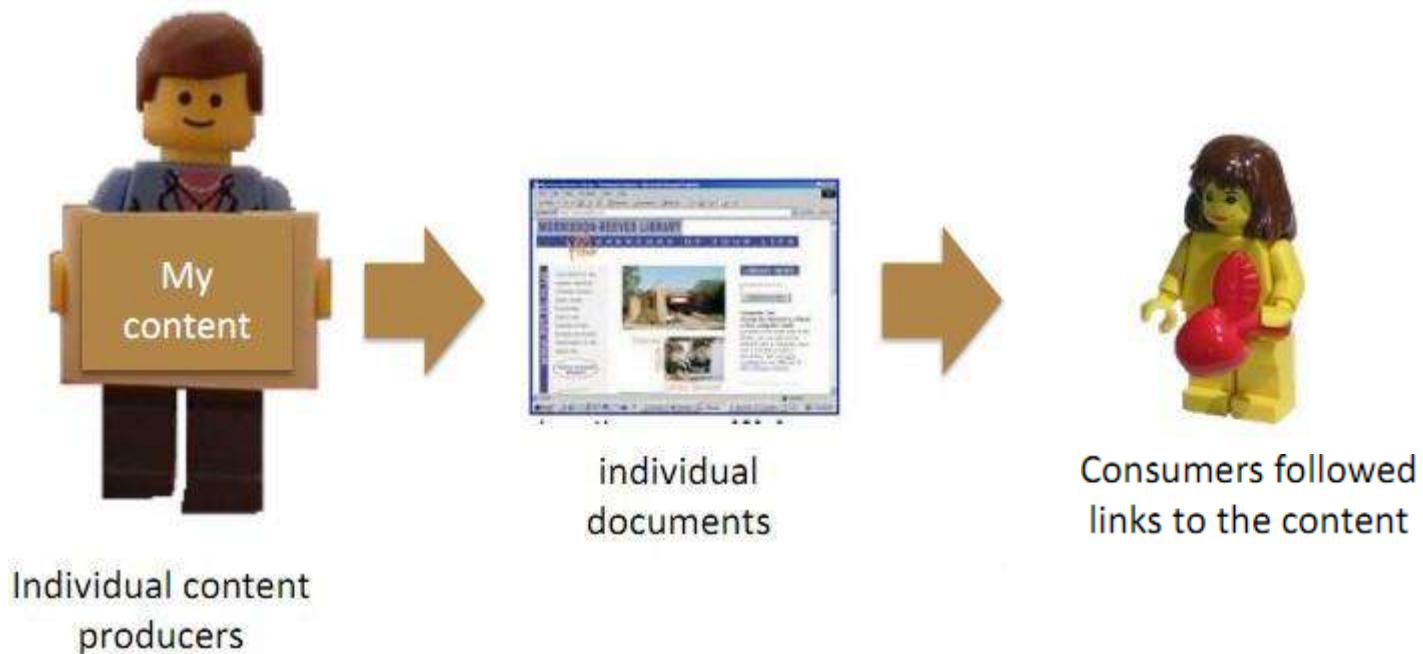
Web1.0 is a one-way platform

WEB 1.0(READ-ONLY STATIC WEB)

- First stage of the World Wide linking web pages and hyperlink
- Most read-only Web. It focused on companies home pages
- Dividing the world wide web into usable directories
- It means Web is use as “Information Portal”.
- Everyone has their personal own little corner in the cyberspace
- It started with the simple idea “Put content together”
- Media companies put content in the web and pushes it to user. using web 1.0 Companies Like BBC,CNN able to get online.

CONTD.

Things works in web 1.0



CONTD.

.



SAD FACTS OF WEB

1.0

SAD FACTS

- Read only Web
- Limited user interaction
- Keyword based (dumb) search ----- Web Directories
- The Lack of standards ----- Browsers war

NEXT STEP

When we got a grip on the technical part, web became clearer and then we discover

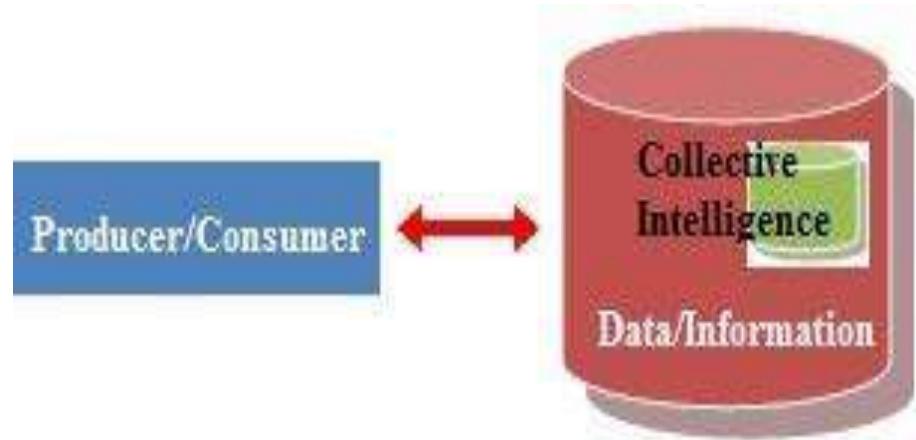
- Power of Networks
- Power of Links
- Power of Collaboration
- Power of content and reach
- Power of Friends

15

Web 2.0

And then the Next step is





- Web 2.0 [Share]

Web2.0 is a two-way
Platform

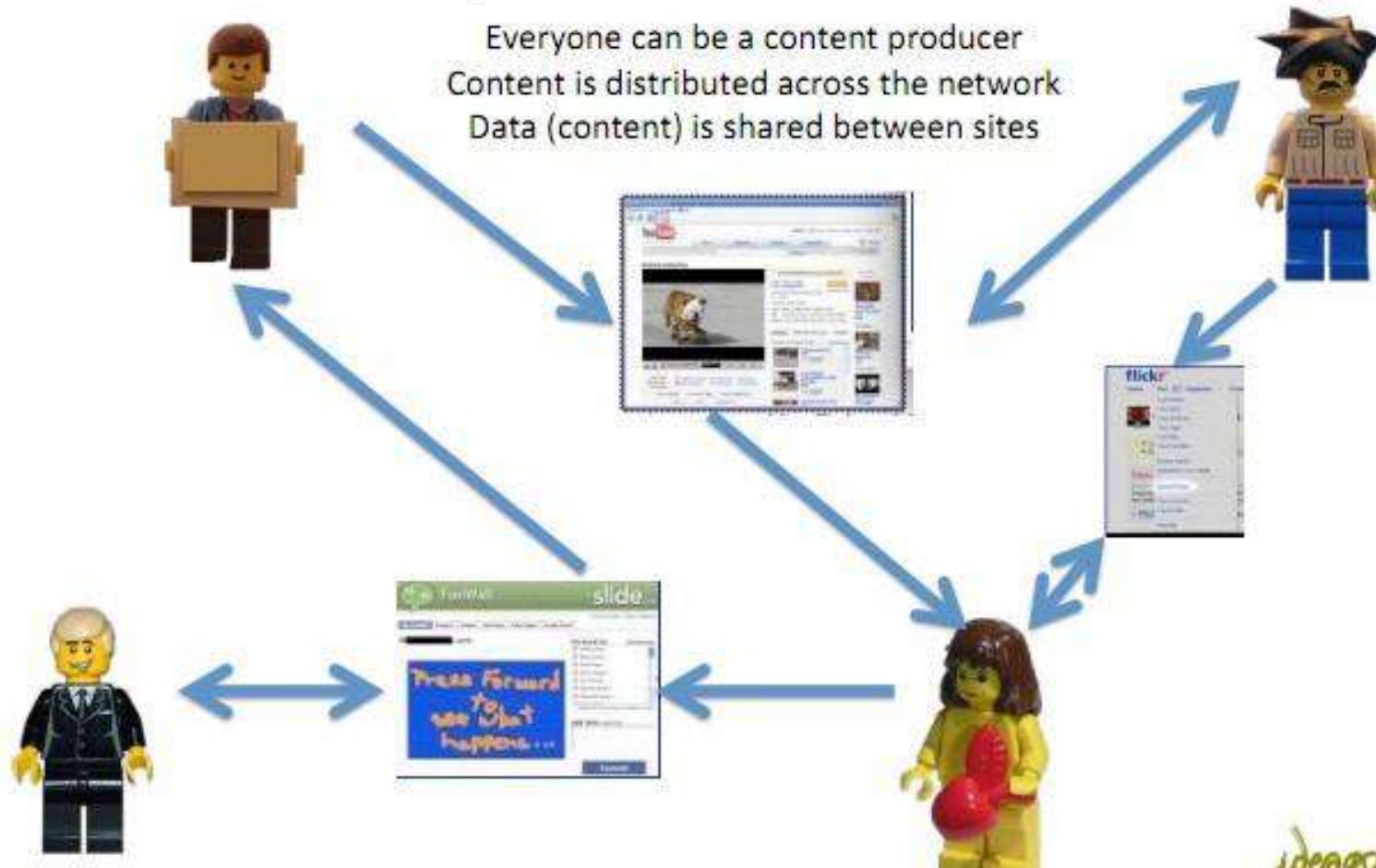
A term used to describe a new generation of
Web services and applications with an
increasing emphasis on human collaboration.

WEB 2.0(READ-WRITE INTERACTIVE WEB)

- It is a platform that gives users the possibility (liberty) to control their data.
- This is about user-generated content and the read-write web.
- People are consuming as well as contributing information through blogs or sites like Flicker, YouTube, Digg, etc.



In Web 2.0 you have a distribution relationship





PRINCIPLES OF WEB 2.0

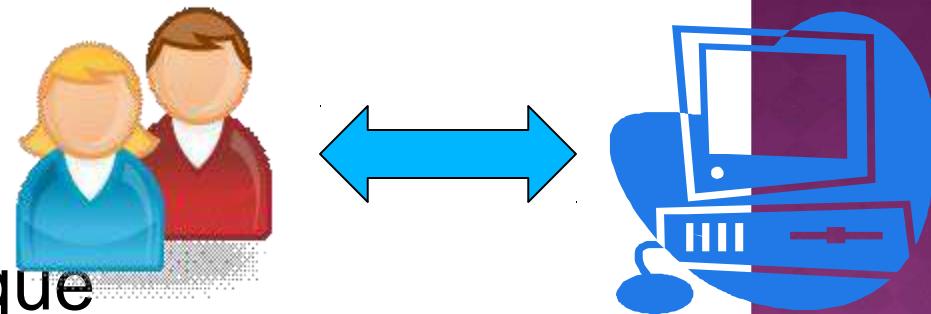
No Products but Services

- “There are no products, only solutions”
- A problem solving approach
- Must Provide Simple Solutions

CONTD...

Customization

- Every individual is unique
- Some people want to be different
- Allow him to choose instead of forcing him to use what you have made
- Make him feel home
 - e.g. My yahoo, Google Homepage, MySpace , Firefox extensions



Concepts

Web 2.0 can be described in 3 parts which are as follows:

- Rich Internet Application (RIA) - It defines the experience brought from desktop to browser .whether it is from a graphical point of view or usability point of view. Some people relate RIA with *AJAX* and *Flash*.
- Service-oriented Architecture (SOA) - It is a key piece in Web 2.0 which defines how Web 2.0 applications expose its functionality so that other applications can integrate the functionality and produce a set of much richer applications (Examples are: Feeds, RSS, Mash-ups)

CONTD...

Social Web – It defines how Web 2.0 tend to interact much more with the end user and making the end user an integral part.



CONTD...

Social Web

- A third important part of Web 2.0 is the Social Web. The term is currently used to describe how people socialize or interact with each other throughout the Web .
- The social web consists of a number of online tools and platforms where people share their perspectives, opinions, thoughts and experiences
- Web 2.0 Applications tend to interact much more with the end user. As such, the end user is not only a user of the application but also a participant

CONTD.

User can participate by :-

- Podcasting
- Blogging
- Tagging
- Contributing to RSS
- Social bookmarking
- Social networking

..



CONTD...

Technologies

The client-side/web browser technologies used in Web 2.0 development are :

Ajax(Asynchronous JavaScript +XML)

Ajax programming uses JavaScript to upload and download new data from the web server without full page reload.

Adobe Flex

Flex makes it easier for programmers to populate large data grids, charts, and other heavy user interactions. Applications programmed in Flex, are compiled and displayed as Flash within the browser

WEB 1.0 VS WEB 2.0

Web 1.0

- The mostly read only Web
- 45million global user(1996).
- Focused on companies
- Home pages
- Owning content
- HTML,portals
- Web forms.
- Netscape
- Page views

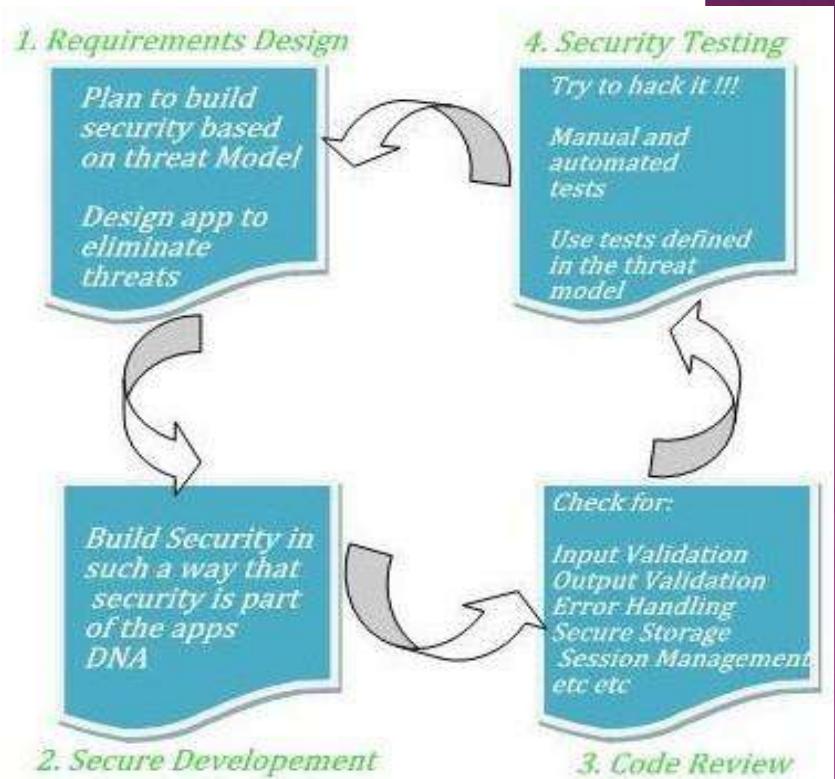
Web 2.0

- The widely read -write web
- 1 billion + global user(2006)
- Focused on communities
- Blogs
- Sharing content
- XML,RSS
- Web Application
- Google
- Cost per click

WEB 2.0

Sad Facts

- Same old Keyword based search.
- Web application are still rigid
- Each Website have its own data and it is not sharing it.
- Computers can not understand any thing
- Web 2.0 is Social change. The technical part has not change much.



INTRODUCING NEW KIND OF WEB

Main Reasons

- How will our information be organized.
- Will we still do the “surfing” or will the machine surf for us

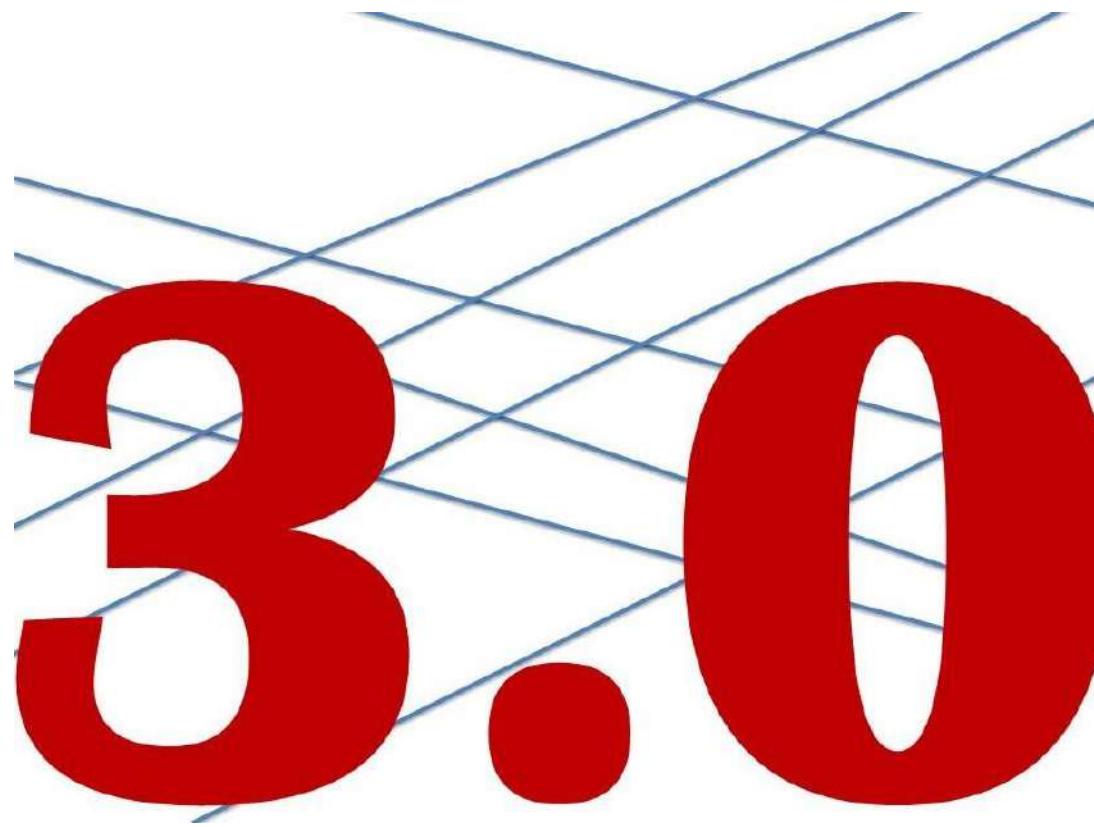
NEW CONCEPT IS WEB OF DATA

Beyond the present Web Lets move towards the web
of Data

WEB OF DATA

New kind of Web capable of reading and understanding content and context.

When the web can understand content it can better satisfy the request of people and machines.



3.0

WEB 3.0 READ-WRITE INTELLIGENT

Semantic Web

- It is a Web of data.
- changing the web into a language that can be read and categorized by the system rather than humans.



Artificial Intelligence

- Extracting meaning from the way we people interact with the Mobility
- everything, everywhere



EXAMPLE

- Suppose, I am a stamp collector...
- Over the years I've collected a lot of stamps.
- About every stamp, I made a document
- That's a lot of documents



Postzegel "Post Office"

| | |
|--------------------|-------------------------------|
| Naam: | Post Office |
| Land van herkomst: | Engeland |
| Waarde: | 1 cent |
| Ontwerp: | Mr Adam Green |
| Ontwerp: | Postoffice boxen in wit kader |
| Gedrukt van: | Postzegelhuis 't Zegeltje |
| Handelswaarde: | 200 euro |
| Datum van aankoop: | 11 oktober 1987 |

Omschrijving:
Als kind heb ik ooit eens postzegels gespaard. Gewoon omdat ik er veel uitgekeken had. Maar op een gegeven moment is dat opeens ophouden en ik had geen idee waar die postzegels gehouden zijn. Ongeveer 10 jaar later wende ik in Nederland en kreeg post overal vandaan. Mijn blik viel op de postzegels en ik wist aan wie dat. Ik kocht nouw postzegels, ik spaarde alleen wat er door de brievenbus kwam en verder wat mensen, die ervan wisten, mij gaven. Deze verzameling kwam aan een einde toen ik naar Canada emigreerde - ik had geen plaats om ze mee te nemen, en dus bleven zij achter.

CONTD. .

- How can I find a specific stamp?
- Google?
- This is the web we have today: a huge collection of documents
- The words of all those documents are indexed.
We can search for keywords.



CONTD. .

- Now, suppose I Google for all red stamps
- Not very intelligent...



Red stamps

Stamps from Cambodia
(Khmer Rouge)

Stamps from the Red Sea

Stamps from the 140th
anniversary of the Red Cross
Stamps with red dragons

CONTD. .

- Not very intelligent, but how can a computer know what I mean?
- When we structurally describe that a stamp is a stamp and red is a color.
- Describing data in a structured way can best be done in a database.
- Different databases can be connected.

CONTD. .

In 1980 you could buy this stamp for 1 cent

This is a stamp

Now it's worth 3 euros

This stamp is from the United Kingdom



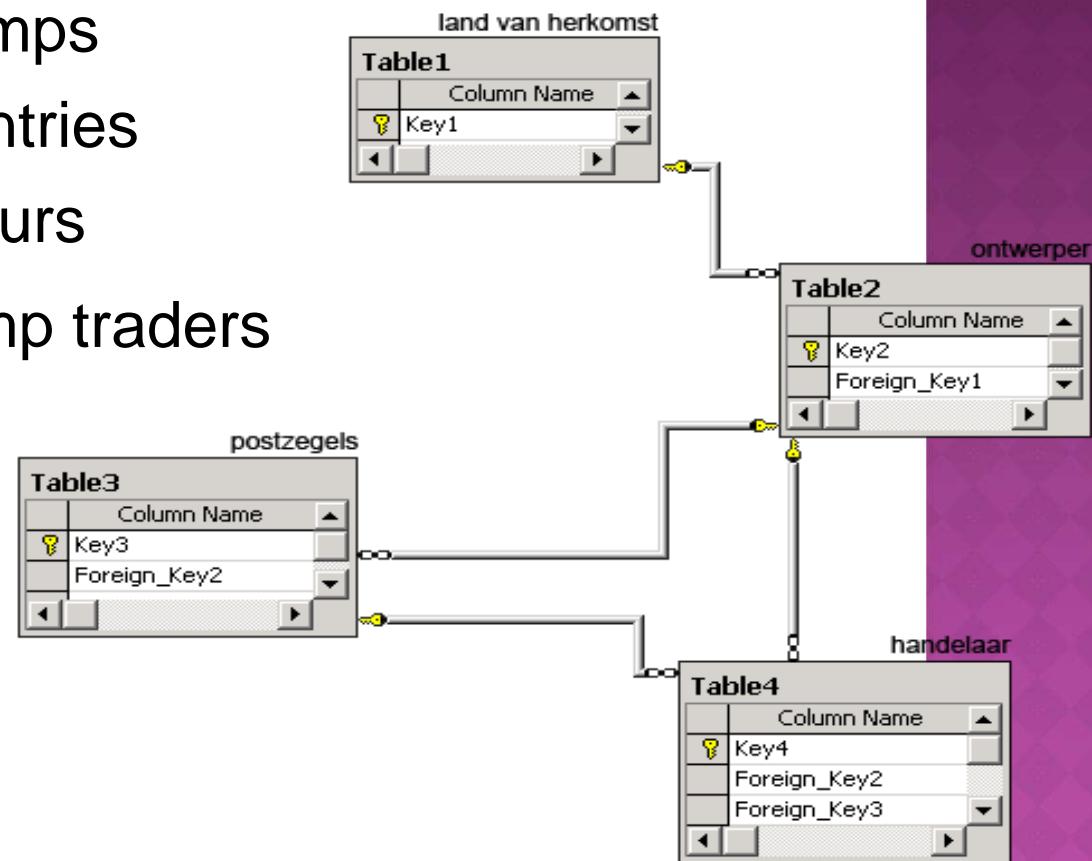
This stamp is used between 1978 - 1981

The picture on the stamp is a PO Box

This stamp is designed by John Bryan Dunmore

CONTD. .

- A database with stamps
- A database with countries
- A database with colours
- A database with stamp traders



Paper ID: 97

EXAMPLE - WEB 3.0 AS DATABASES INTEGRATION

- One view of Web 3.0 is the web being a big collection of databases which can be connected on demand.
- Agreements are made on the structure of data and the way data is described. Where the data is located is irrelevant.
- Linking data is the power of web 3.0.
-

SOME TECHNOLOGIES OF WEB 3.0

- RDF
- XML
- URI
- SPARQL
- XDI
- XRI
- SWRL
- XFN
- OWL
- API
- OAUTH

WEB 3.0

In computing, a Uniform Resource Identifier (URI) is string of characters used to identify a name or a resource on the Internet.

e.g

PHP is programming Language

POWL is an application written in PHP

It use triple {subject,property,object} model

hasWebSite("#php", "<http://www.php.net/>")

isA("#php", "#language")

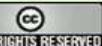
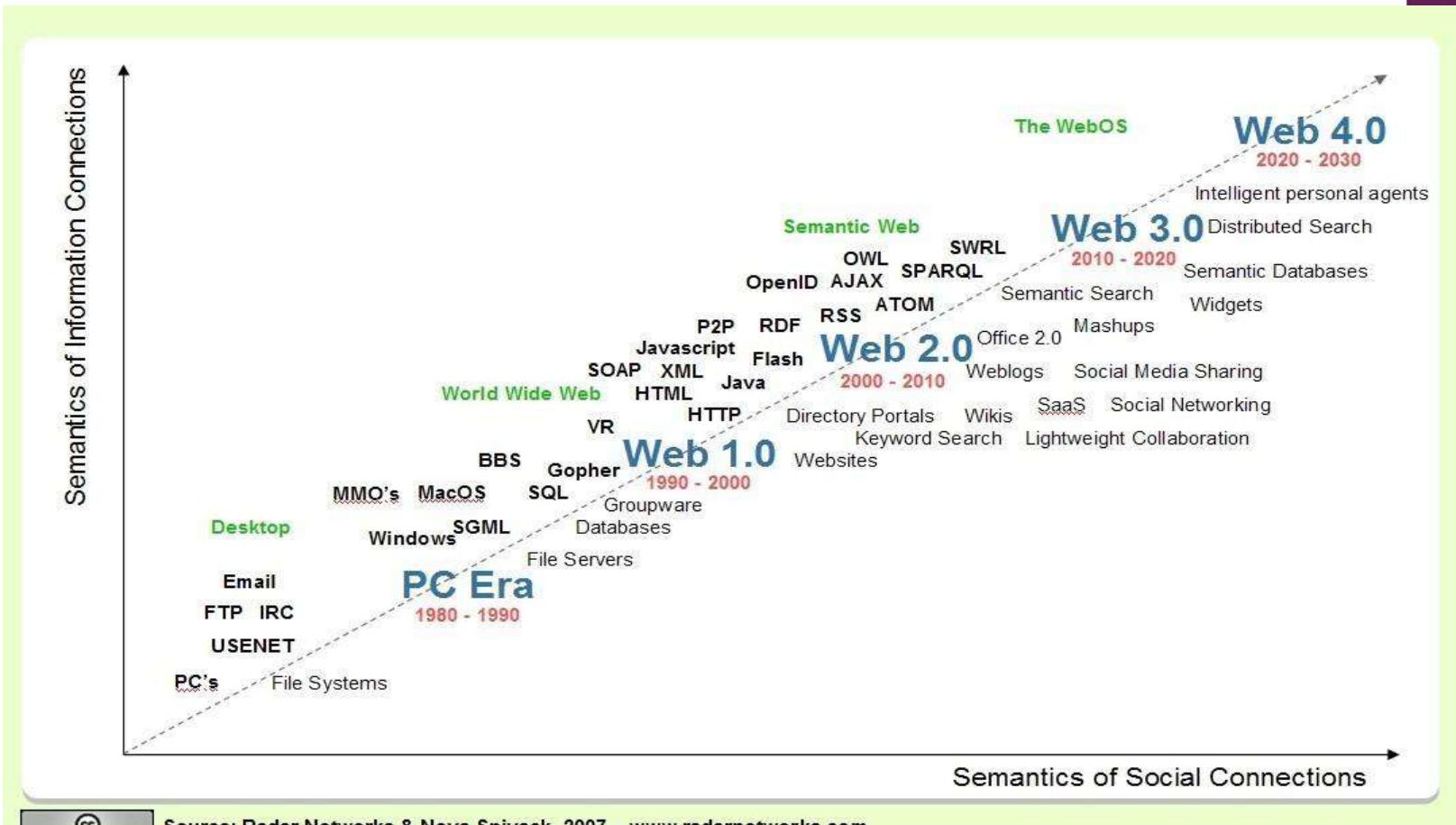
isWrittenIn(<http://powl.sf.net/>, "#php")

It is all about triple of URIs

WEB 2.0 VS 3.0

- Web 2.0 is all about the power of networks
- Basically, web 2.0 is a social change. The technical part of the web hasn't changed very much.
- But, web 3.0 will be driven by technological changes
- Web 3.0 - the semantic web - is about the meaning of information.

WEB HISTORY AND FUTURE



Source: Radar Networks & Nova Spivack, 2007 – www.radarnetworks.com

EXAMPLE WEB

3.0

- Freebase
 - <http://www.firebaseio.com>
- Amazon (“If you liked this, you will like this!”)
 - <http://www.amazon.com>
- Netvibes (pull your Web 2.0 apps together!)
 - <http://www.netvibes.com>

1.50

CUT Copy Format Painter
Paste New Slide Delete
Clipboard Slides

Font Paragraph Drawing Editing

Text Direction Align Text Convert to SmartArt

Shape Fill Shape Outline Quick Styles Shape Effects

Find Replace Select

Slides Outline

1 WEB Evolution

2 Web- 3.0

3 Semantic Web

4 AI+Articles, Intelligence

5 3D Graphics

6 Click to add notes

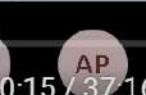
Slide 2 of 22 "Shift"

115%

Web- 3.0

To understand the nuances and subtleties of Web 3.0, let's look at the four properties of Web 3.0:

- Semantic Web
- Artificial Intelligence
- 3D Graphics
- Ubiquitous



Semantic web

Semantic web along with artificial intelligence are the two cornerstones of web 3.0.

The semantic web will help teach the computer what the data means and that will evolve artificial intelligence that can utilize that information.

The core idea is to create a spider web of knowledge across the internet which will help it to understand the meaning of words to generate, share, and connect content through search and analysis.

1.50

CUT Copy Format Painter Clipboard Slides Font Paragraph Drawing Editing

Reset New Slide Delete

Text Direction Align Text Convert to SmartArt

Find Replace Select

Slides Outline

Challenges of Web 3.0 Implementation

Let's look at some of the biggest challenges of Web 3.0 implementation:

- Vastness
- Vagueness
- Uncertainty
- Inconsistency
- Deceit

Click to add notes

Slide 7 of 22 "Shift"

Advantages and Disadvantages of Web 3.0

The Advantages of the Web 3.0:

- Increased information linking: Semantic web will help in the connectivity of online data.
- Efficient searching
- Better marketing.
- More efficient web browsing.
- Effective communication.
- Change human interaction.

Click to add notes

Click to add title

The Disadvantages of Web 3.0:

- Less advanced devices will not be able to handle Web 3.0.
- Web 1.0 websites will seem that much more obsolete
- It can be very complicated for newcomers to understand.

Click to add notes

CUT
Copy
Format Painter
Clipboard

RESET
New Slide
Delete

Slides

Font
Text Direction
Align Text
Convert to SmartArt

Shape Fill
Shape Outline
Arrange
Quick Styles
Shape Effects

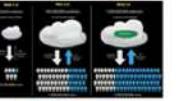
Find
Replace
Select
Editing

Slides

Outline

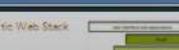
8
Overview and Technologies of Web 2.0
1. Increased interactivity among users
2. User-generated content
3. User rating
4. User reviews
5. User participation
6. User interaction

9
The Evolution of Web 3.0
1. Increased number of websites and users
2. Web 2.0 content is over the main new website
3. Increase in popularity for interaction in websites

10
Web 1.0 → Web 2.0 → Web 3.0: The Evolution


11
The Evolution of the Web


12
The big difference between Web 2.0 and Web 3.0
1. Open to the Web and built upon open source software built for an open and
accessible community of developers and consumers in the form of the web.
2. Transition to the web's social culture participation in internet, public or
private forums, user-created third-party
3. Transformation in how services work, with more emphasis on personalizing
other audiences from a previous, static

13
Semantic Web Stack


Slide 10 of 22 "Shift"

Web 1.0 → Web 2.0 → Web 3.0: The Evolution

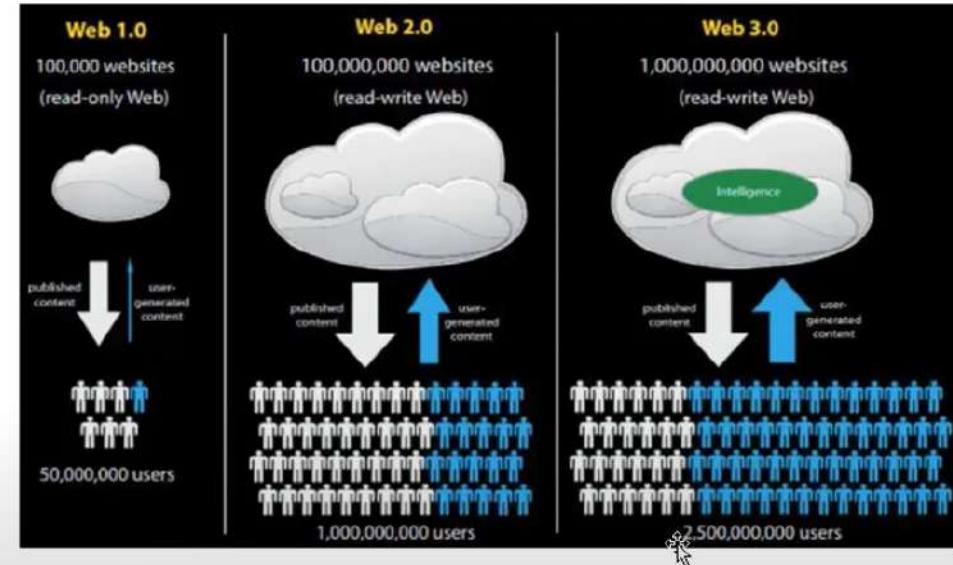


Image Credit: Research Hubs

Click to add notes

115%

+4

NK

KM

AP

KL

D

PS

SM

AS

IA

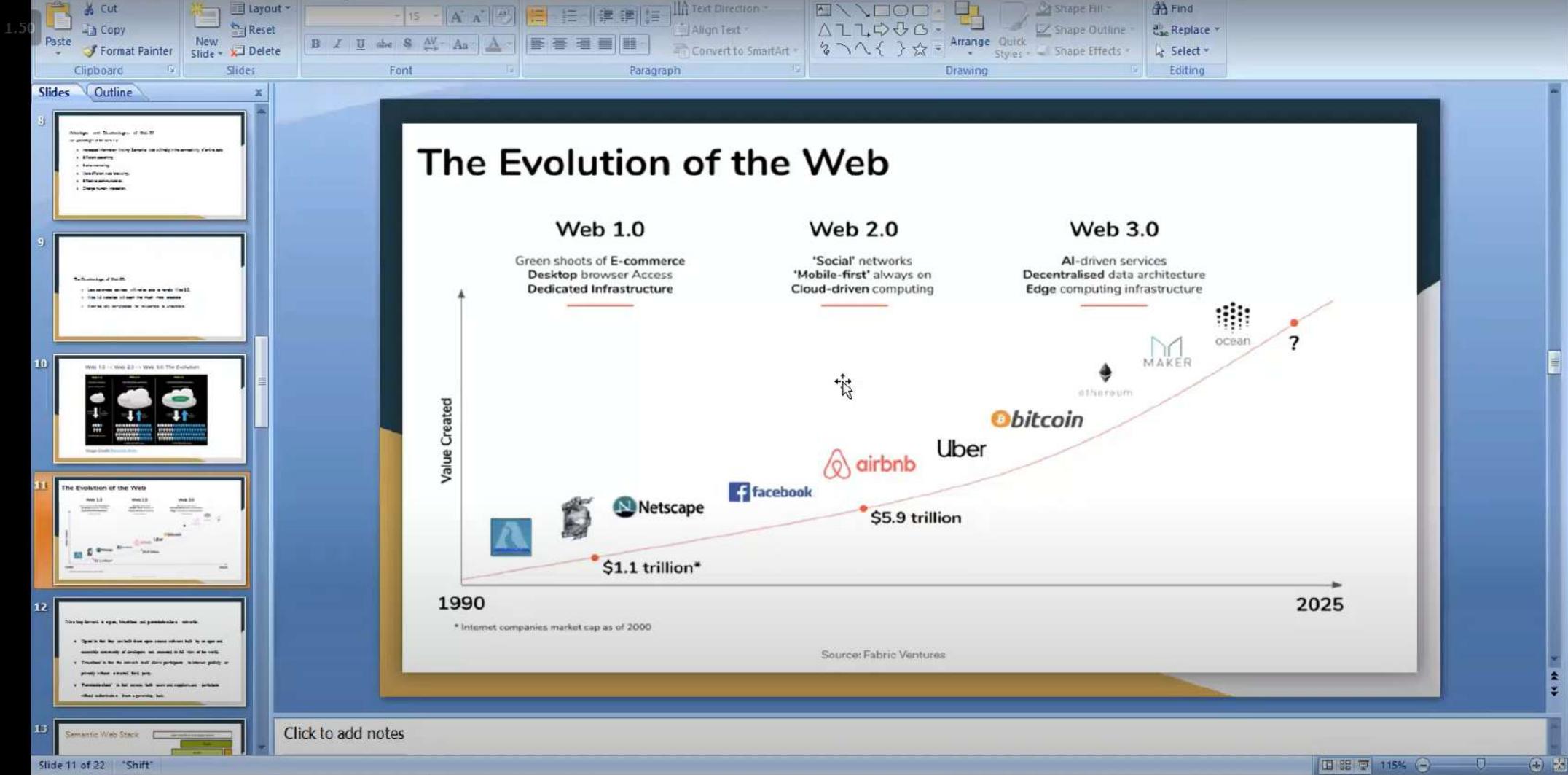


CC

HD

PR

L



Click to add notes



D

PS

SM

AS

IA



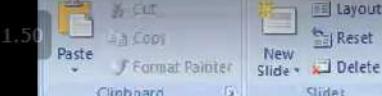
It is a leap forward to **open**, **trustless** and **permissionless** networks.

- '**Open**' in that they are built from open source software built by an open and accessible community of developers and executed in full view of the world.
- '**Trustless**' in that the network itself allows participants to interact publicly or privately without a trusted third party.
- '**Permissionless**' in that anyone, both users and suppliers, can participate without authorisation from a governing body.

Click to add notes



16:59 / 37:16

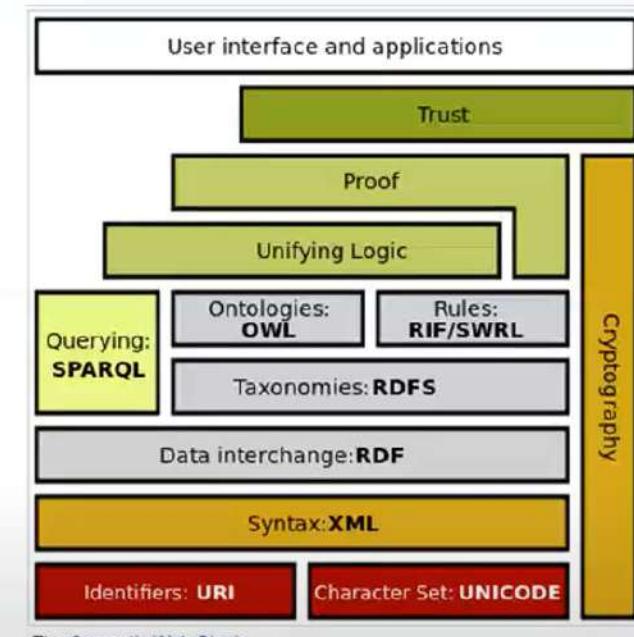


Slides

Outline

Semantic Web Stack

- XML provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within.
- XML Schema is a language for providing and restricting the structure and content of elements contained within XML documents.
- RDF is a simple language for expressing data models, which refer to objects ("web resources") and their relationships.



Click to add notes



1.50

Cut Copy Format Painter Clipboard Slides Font Paragraph Drawing Editing

13 Semantic Web Stack

14 RDF is a simple language for semantic web sources.

15 Web semantics works.

16

17

18

Click to add notes

Slide 14 of 22 "Shift"

115%

- RDF Schema extends RDF and is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.
- SPARQL is a protocol and query language for semantic web data sources.
- RIF is the W3C Rule Interchange Format. It's an XML language for expressing Web rules that computers can execute. RIF provides multiple versions, called dialects.

Web semantic works:

Figure 1: RDF considers data set

| ID | Author | Title | Publisher | Year |
|--------------------|--------|------------------|-----------|------|
| ISBN 0-00-651409-X | id_xyz | The Glass Palace | id_qpr | 2000 |

| ID | Name | Home page |
|--------|--------------|---|
| id_xyz | Amitav Ghosh | http://www.amitavghosh.com |

| ID | Publisher Name | City |
|--------|----------------|--------|
| id_qpr | Harper Collins | London |

Click to add notes

1.50

Cut Copy Format Painter Paste New Slide Reset Delete Slides Font Paragraph Drawing Editing

Slides Outline

13 Semantic Web Stack

14 RDF triples

15 Web semantic works

16 RDF diagram

17 RDF diagram

18 RDF diagram

Slide 16 of 22 "Shift"

Click to add notes

115%

The diagram illustrates a semantic web graph with the following nodes and edges:

- Nodes:
 - `http://.../isbn/000651409X`
 - `The Glass Palace`
 - `2000`
 - `London`
 - `HarpersCollins`
 - `Amitav Ghosh`
 - `http://www.amitavghosh.com`
- Edges:
 - `http://.../isbn/000651409X` → `The Glass Palace` (label: `a:title`)
 - `http://.../isbn/000651409X` → `2000` (label: `a:year`)
 - `http://.../isbn/000651409X` → `London` (label: `a:city`)
 - `http://.../isbn/000651409X` → `HarpersCollins` (label: `a:p_name`)
 - `London` → `HarpersCollins` (label: `a:publisher`)
 - `Amitav Ghosh` → `http://www.amitavghosh.com` (label: `a:homepage`)
 - `Amitav Ghosh` → `Amitav Ghosh` (label: `a:name`)

Source: Ivan Herman, Question (and answer) on the semantic web, 2006, P. 25

The diagram illustrates a Semantic Web graph with nodes and edges representing various entities and their relationships. A red circle highlights the URI <http://...isbn/000851409X>, which appears twice in the graph. Red arrows point from the text "Same URI = Same Resources" to these two occurrences of the URI.

```

graph TD
    GP["The Glass Palace"] -- scelle --> ISBN1
    ISBN1["http://...isbn/000851409X"]
    ISBN1 -- ayear --> 2000
    ISBN1 -- apublic --> L
    L["London"]
    L -- acity --> HC
    HC["HarpersCollins"]
    HC -- aig_name --> AG
    AG["Amitav Ghosh"]
    AG -- ahomepage --> AG_Homepage["http://www.amitavghosh.com"]
    AG_Homepage -- aauthor --> ISBN1
    AG_Homepage -- aauthor --> ISBN2
    ISBN2["http://...isbn/2020386687"]
    ISBN2 -- foriginal --> F
    F["Amitav Ghosh"]
    F -- Enom --> AG
    ISBN2 -- ftraducteur --> CB
    CB["Christiane Besse"]
    CB -- Enom --> F
    L -- apublic --> ISBN2
    ISBN2 -- apublic --> LPD
    LPD["Le palais des miroirs"]
    LPD -- auteur --> CB
  
```

Same URI = Same Resources

Source: Ivan Herman, Question (and answer) on the semantic web, 2006, P. 27

Click to add notes

Slides **Outline**

13 Semantic Web Stack

14 RDF triples example

15 Web semantic works

16 RDF diagram

17 Ivan Herman's diagram

18 Semantic Web Stack

The diagram illustrates a semantic web graph with the following entities and their relationships:

- Entities:** "The Glass Palace", "2000", "London", "Harcourt", "Anirban Ghosh", "http://www.anirbanghosh.com", "Le polar des meurirs", "http://isbn/1420336601", and "Christiane Bozzo".
- Relationships:**
 - "The Glass Palace" is an "article" and has a "year" of "2000".
 - "London" is a "city" and is associated with "Harcourt".
 - "Harcourt" is a "publisher" of "The Glass Palace".
 - "Anirban Ghosh" is the "author" of "The Glass Palace".
 - "Anirban Ghosh" is the "author" of "Le polar des meurirs".
 - "Le polar des meurirs" is the "original" work of "http://isbn/1420336601".
 - "Le polar des meurirs" is the "filter" of "http://isbn/1420336601".
 - "Le polar des meurirs" is the "translator" of "http://isbn/1420336601".
 - "Christiane Bozzo" is the "from" person of "http://isbn/1420336601".

Source: Ivan Herman, Question (and answer) on the semantic web, 2006, P.28

Click to add notes

Figure 5: the merge continues to other sets of data

Source: Ivan Herman, Question (and answer) on the semantic web, 2006, P. 38

Click to add notes

URI ,URL,URN

A Venn diagram with three overlapping circles. The largest circle is blue and labeled "URI (identifier)" with the ISBN "0-486-27557-4". The middle-left circle is red and labeled "URN (name)" with the URN "urn:isbn:0-486-27557-4". The middle-right circle is green and labeled "URL (locator)" with the URL "https://google.com". The intersection of all three circles contains the text "uri uri uri miessler 2020". Below the diagram is the name "DANIEL MIESSLER 2020".

All URLs are URIs,
but not all URIs are
URLs.

Click to add notes

CSS Selectors:

1. .intro -All elements with class="intro"
2. #Lastname-The element with id="Lastname"
3. .intro, #Lastname-All elements with class="intro", and the element with id="Lastname"
4. h1-All <h1> elements.
5. h1, p-All <h1> elements and all <p> elements.
6. div p-All <p> elements that are inside a <div> element.
7. div > p-All <p> elements where the parent is a <div> element.
8. ul + p-The <p> element that are next to a elements.
9. ul ~ table-All <table> elements that are siblings of a element.
- 10.*-Universal
- 11.p.myquote-All <p> elements with class="myquote".
- 12.[id]- All elements with an id attribute.
- 13.[id=my-Address]- All elements with an id attribute value equal to "my-Address"
- 14.[id\$=ess]- All elements with an id attribute value ending with "ess"
- 15.[id|=my]- All elements with an id attribute value equal to "my" or starting with "my" followed by a hyphen (-)
- 16.[id^=L]- All elements with an id attribute value starting with the letter "L"
- 17.[title~=beautiful]- All elements with a title attribute value containing the word "beautiful"
- 18.[id*=s]- All elements with an id attribute value containing the string "s"
- 19.:checked-All checked form elements
- 20.:disabled-All disabled form elements
- 21.:enabled-All enabled form elements.
- 22.:empty-All empty elements
- 23.:focus-The element that currently has focus.
- 24.p:first-child-All <p> elements that are the first child of their parent.

25.p::first-letter-The first letter of all <p> elements.

26.p::first-line-The first line of all <p> elements.

27.p:first-of-type-All <p> elements that are the first <p> element of their parent.

28.h1:hover-All <h1> elements, but only when you hover them.

Try hover (mouse over) the H1 element in the result.

29.input:in-range-All <input> elements with a max and/or min value, where the value is within the specific range.

30.input:out-of-range-All <input> elements with a max and/or min value, where the value is outside the specific range.

31.input:invalid-All <input> elements where the value is invalid according to their limitations.

32.input:valid-All <input> elements where the value is valid according to their limitations.

33.p:lang(it)- All <p> elements with a lang attribute value starting with "it"

34.p:last-child-All <p> elements that are the last child of their parent.

35.p:last-of-type-All <p> elements that are the last <p> element of their parent.

36.tr:nth-child(even)- tr:nth-child(even)

37.tr:nth-child(odd)- All odd <tr> elements.

38.li:nth-child(1)- All elements that are the first child of their parent, counting from the element.

39.li:nth-last-child(1)- All elements that are the first child of their parent, counting from the element.

40.li:nth-of-type(2)- All elements that are the second element of their parent.

41.li:nth-last-of-type(2) -All elements that are the second element of their parent, counting from the element.

42.b:only-child- All elements that are the only child of their parent.

43.h3:only-of-type -All <h3> elements that are the only child of its type, of their parent.

44.:root - The document's root element.

Introduction to HTML:

1.Internal :

```
<!DOCTYPE html>

<html>
  <head>
    <style>
      body {
        background-color: linen;
      }
      h1 {
        color: maroon;
        margin-left: 40px;
      }
    </style>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

2.Inline:

```
<!DOCTYPE html>

<html>
  <body>
```

```
<h1 style="color:blue;text-align:center;">This is a heading</h1> <p style="color:red;">This is a paragraph.</p></body></html>
```

3. External CSS:

```
<!DOCTYPE html>

<html>

<head>

<title>Layout for my WEBSITE</title>

<link rel="stylesheet" type="text/css" href="style.css">

</head>

<body>

<div id="wrapper">

<div id="header"></div>

<div id="content">

<div id="left-panel">

<div id="navbar"></div>

<div id="news"></div>

</div>

<div id="right-panel"></div>

</div>

<div id="footer"></div>

</div>

</body>

</html>
```

CSS:

```
*{padding: 0px; margin: 0px;}
```

```
#wrapper {
```

```
/* border:solid; */
```

```
padding: 15px;
```

```
}
```

```
@media(min-width: 1200px)
```

```
{
```

```
#header{
```

```
height: 120px;
```

```
background-color: grey;
```

```
border-radius: 15px;
```

```
}
```

```
#content{
```

```
/*border:solid red; */
```

```
margin-top: 15px;
```

```
}
```

```
#content::after{
```

```
content: " ";
```

```
display: block;
```

```
clear: both;
```

```
}
```

```
#left-panel
{
    /* border:solid blue;*/
    width: 20%;
    float: left;

}

#navbar{
    background-color: pink;
    height: 400px;
    border-radius: 15px;
}

#news
{
    background-color: green;
    height: 200px;
    margin-top: 15px;
    border-radius: 15px;
}

#right-panel{
    background-color: yellow;
    height: 620px;
    width: 78%;
    border-radius: 15px;
    float: right;
```

```
}

#footer{
    height: 100px;
    border-radius: 15px;
    background-color: blue;
    margin-top: 15px;
}

}

@media(min-width: 992px) and (max-width: 1199px)

{
    #header{
        height: 120px;
        background-color: grey;
        border-radius: 15px;
    }

}

#content{
    /* border:solid red;*/
    margin-top: 15px;
}

#content::after{
    content: " ";
    display: block;
    clear: both;
```

```
}

#left-panel{
    /*border:solid blue;*/
    width: 20%;
    float: left;

}

#navbar{
    background-color: pink;
    height: 400px;
    border-radius: 15px;
}

#news{
    background-color: green;
    height: 200px;
    margin-top: 15px;
    border-radius: 15px;
}

#right-panel{
    background-color: yellow;
    height: 620px;
    width: 78%;
```

```
border-radius: 15px;  
float: right;  
  
}  
  
#footer{  
height: 100px;  
border-radius: 15px;  
background-color: blue;  
margin-top: 15px;  
}  
  
}
```

```
@media(min-width: 768px) and (max-width: 991px)
```

```
{  
#header{  
height: 120px;  
background-color: grey;  
border-radius: 15px;  
}  
  
}
```

```
#content{  
/* border:solid red; */  
margin-top: 15px;  
display: table;  
width: 100%;
```

```
}

#left-panel

{

/*border:solid blue;*/

width: 100%;

display: table-footer-group;

}

#left-panel::after{

content: " ";

display: block;

clear: both;

}

#navbar{

background-color: pink;

height: 90px;

border-radius: 15px;

float: right;

width: 78%;

}

#news

{
```

```
background-color: green;  
height: 90px;  
  
border-radius: 15px;  
float: left;  
width: 20%;  
  
}  
  
#right-panel{  
background-color: yellow;  
height: 620px;  
width: 100%;  
border-radius: 15px;  
margin-bottom: 15px;  
  
}  
  
#footer{  
height: 100px;  
border-radius: 15px;  
background-color: blue;  
margin-top: 15px;  
}  
}
```

```
@media(max-width: 767px){  
    #header{  
        height: 120px;  
        background-color: grey;  
        border-radius: 15px;  
    }  
  
    #content{  
        /*border:solid red;*/  
        margin-top: 15px;  
        display: table;  
        width: 100%;  
    }  
  
    #left-panel{  
        /*border:solid blue;*/  
        width: 100%;  
        display: table-footer-group;  
    }  
}
```

```
#navbar{  
background-color: pink;  
height: 90px;  
border-radius: 15px;  
margin-bottom: 15px;  
width: 100%;
```

```
}
```

```
#news
```

```
{  
background-color: green;  
height: 90px;  
  
border-radius: 15px;  
  
width: 100%;
```

```
}
```

```
#right-panel{  
background-color: yellow;  
height: 620px;  
width: 100%;  
border-radius: 15px;  
margin-bottom: 15px;
```

```
}

#footer{
    height: 100px;
    border-radius: 15px;
    background-color: blue;
    margin-top: 15px;
}

}
```

Example for Animation:

```
<!DOCTYPE html>

<html>
    <head>
        <style>
            div {
                width: 300px;
                height: 300px;
                background-color: red;
                animation-name: example;
                animation-duration: 60s;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

```
@keyframes example {  
    0% {background-color: red;}  
    25% {background-color: yellow;}  
    50% {background-color: blue;}  
    100% {background-color: green;}  
}  
</style>  
</head>  
<body>  
<div></div>  
</body>  
</html>
```