ITSE-2192 Fundamental of web Design and Development

Introduction to Web and WWW

Lecture One

Outline

- The Internet Basics
- The World Wide Web
- URL and HTTP

The Internet Basics

- It is network of networks
- You can consider it as one large piece of cable that connects computers in the world

Why do we Use Internet

- To share information
  - i.e Document , News
- To share resource
  - i,.e Cloud Computing [SAS, PAS,IAAS]
- To communicate with each other
  - i.e VoiP, Other Social Medias tools

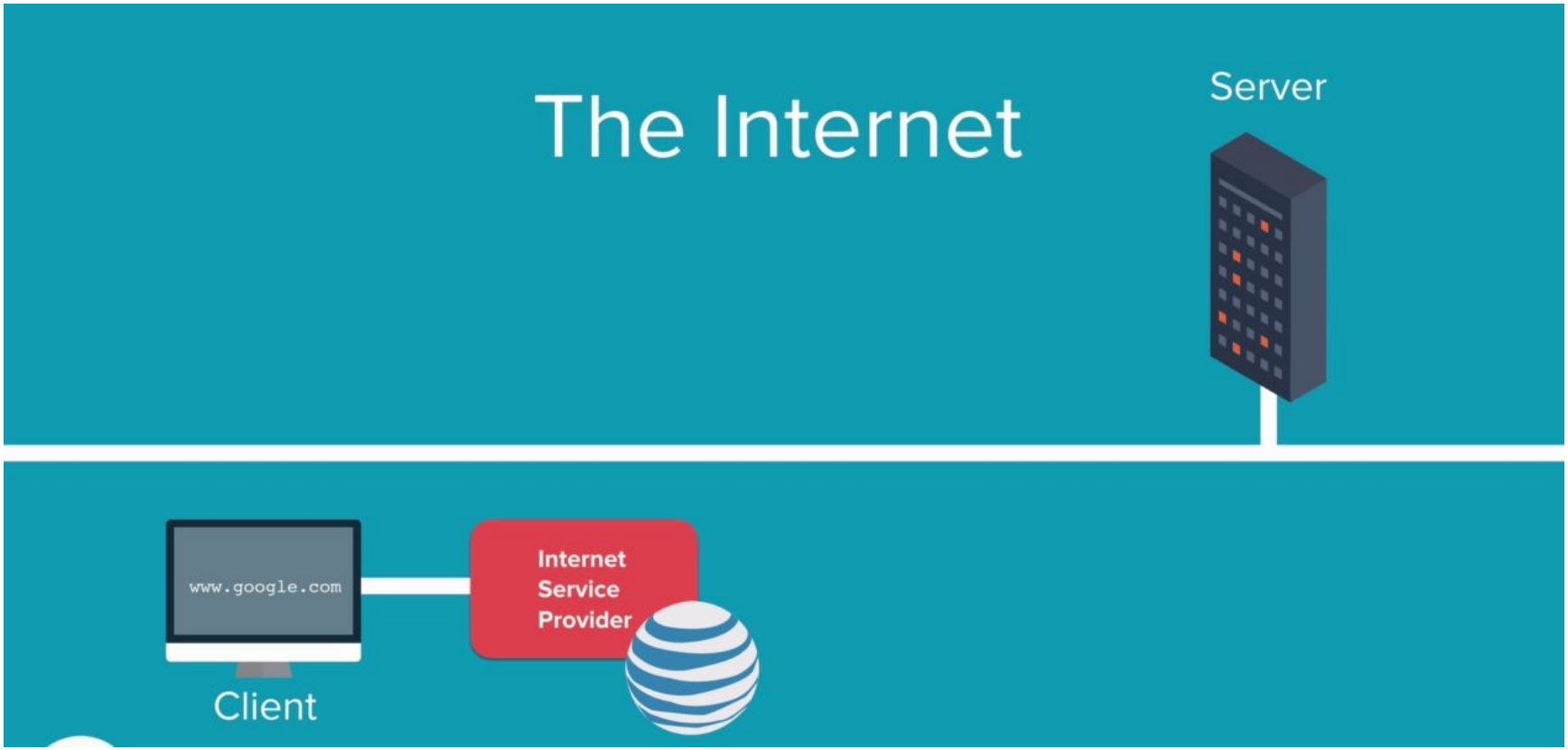
How does the internet works

- The internet consist of several components mainly software and hardware
- Software
  - Protocols Stack [TCP/IP]
- Hardware
  - Cables [Fiber Optics , CAT5], Switch's , Routers , DHCP Servers, DNS Servers, Applications Servers(E.g Web Server)
  - Client Computers

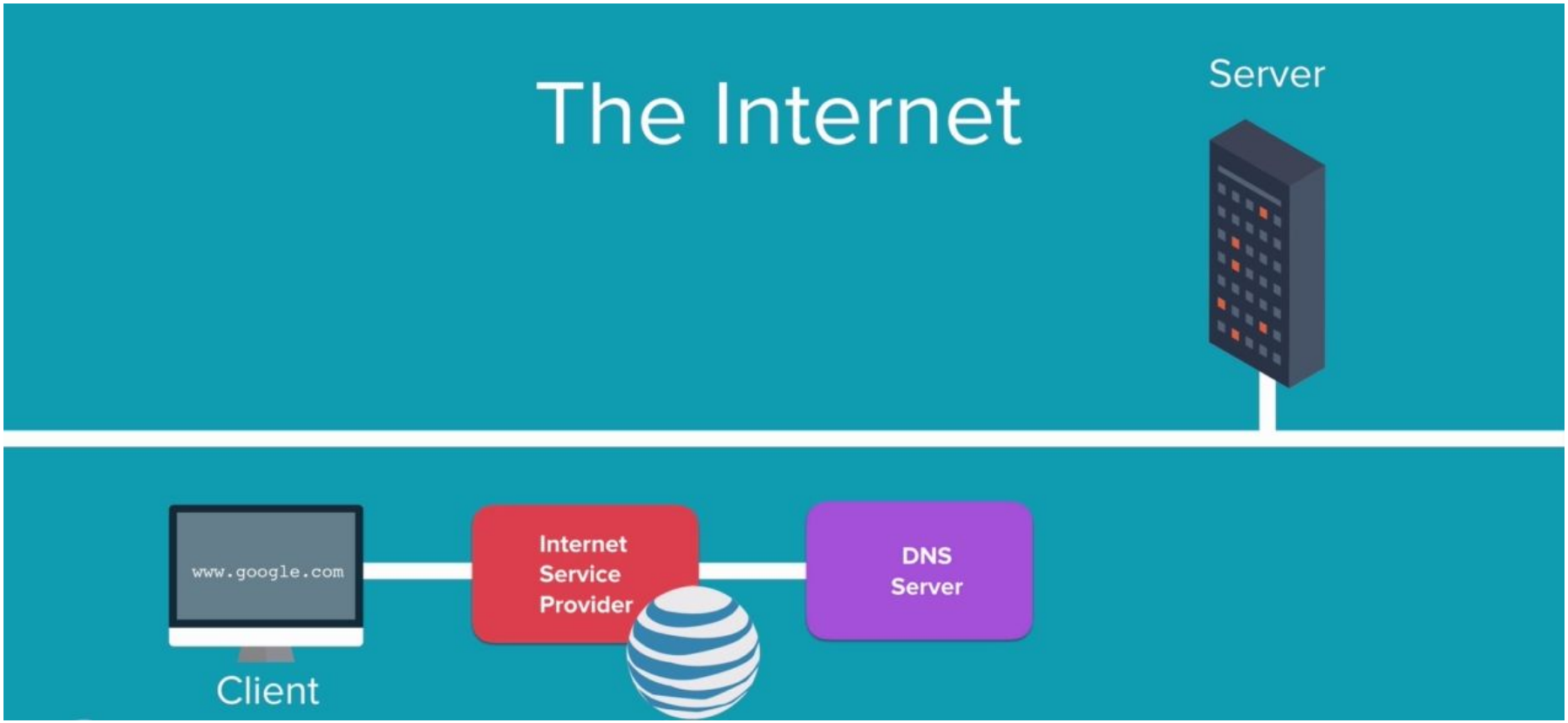
The Client Server Model

E.g The Web Server [Available 24/7]

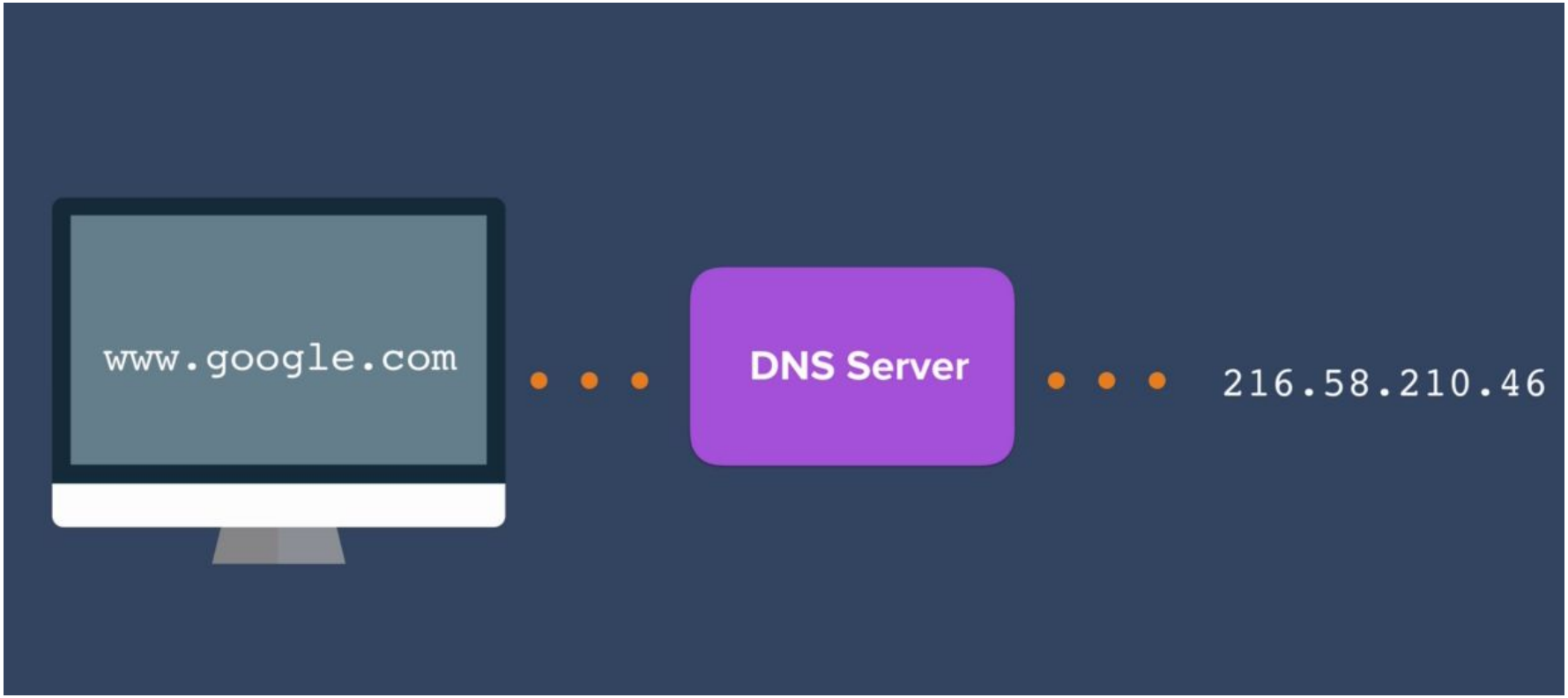
How Internet Works



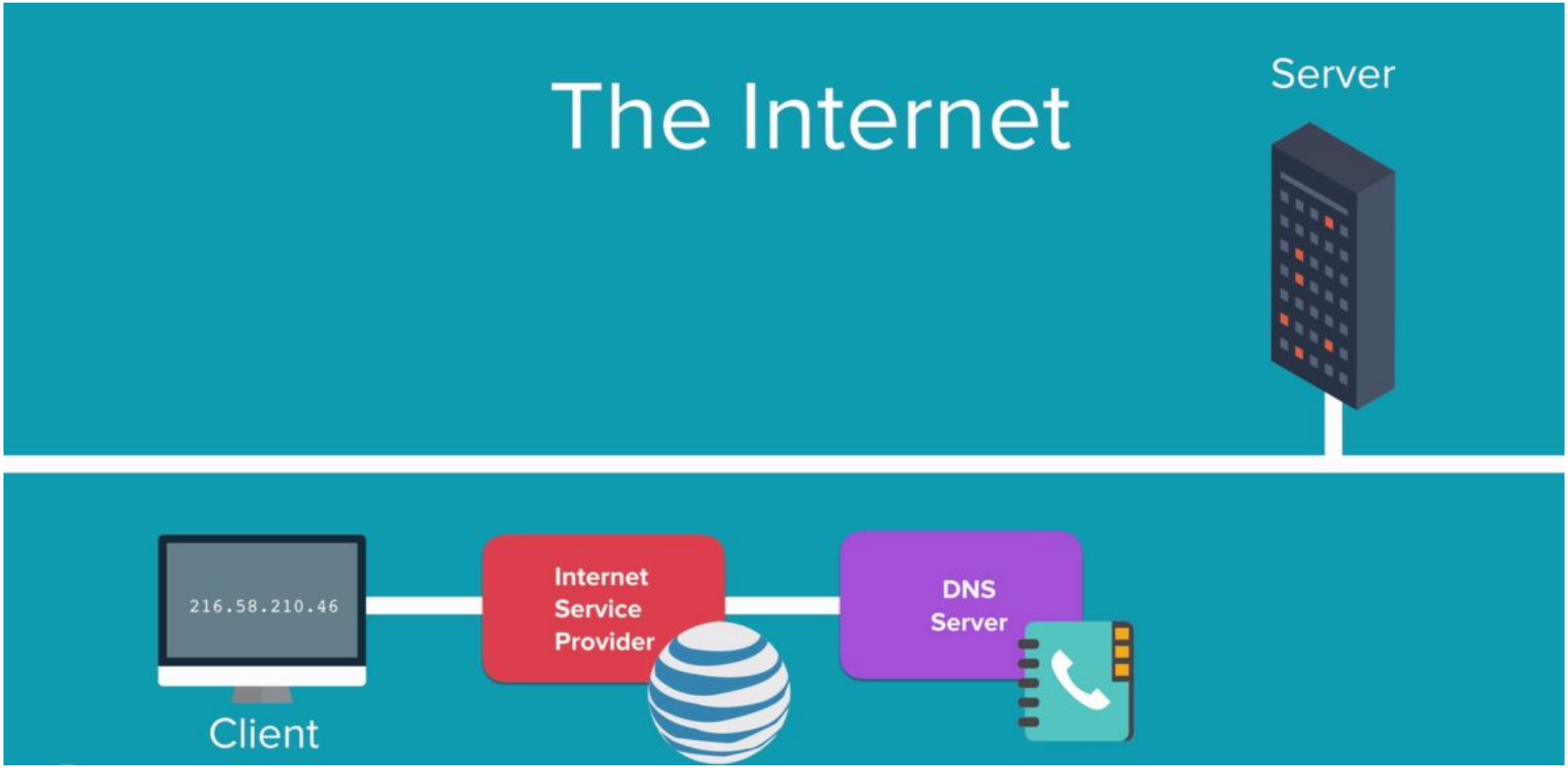
How Internet Works (Continued..)



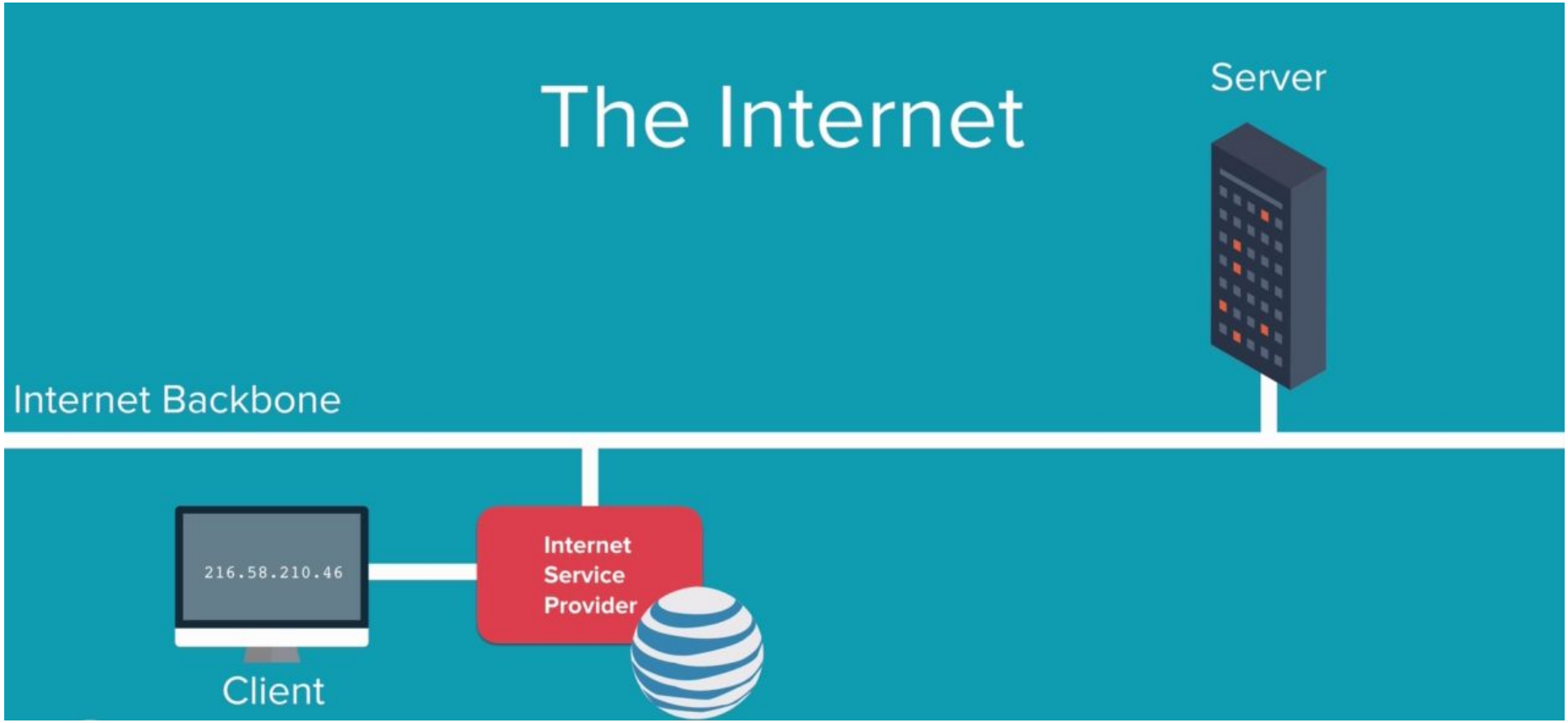
How Internet Works (Continued..)



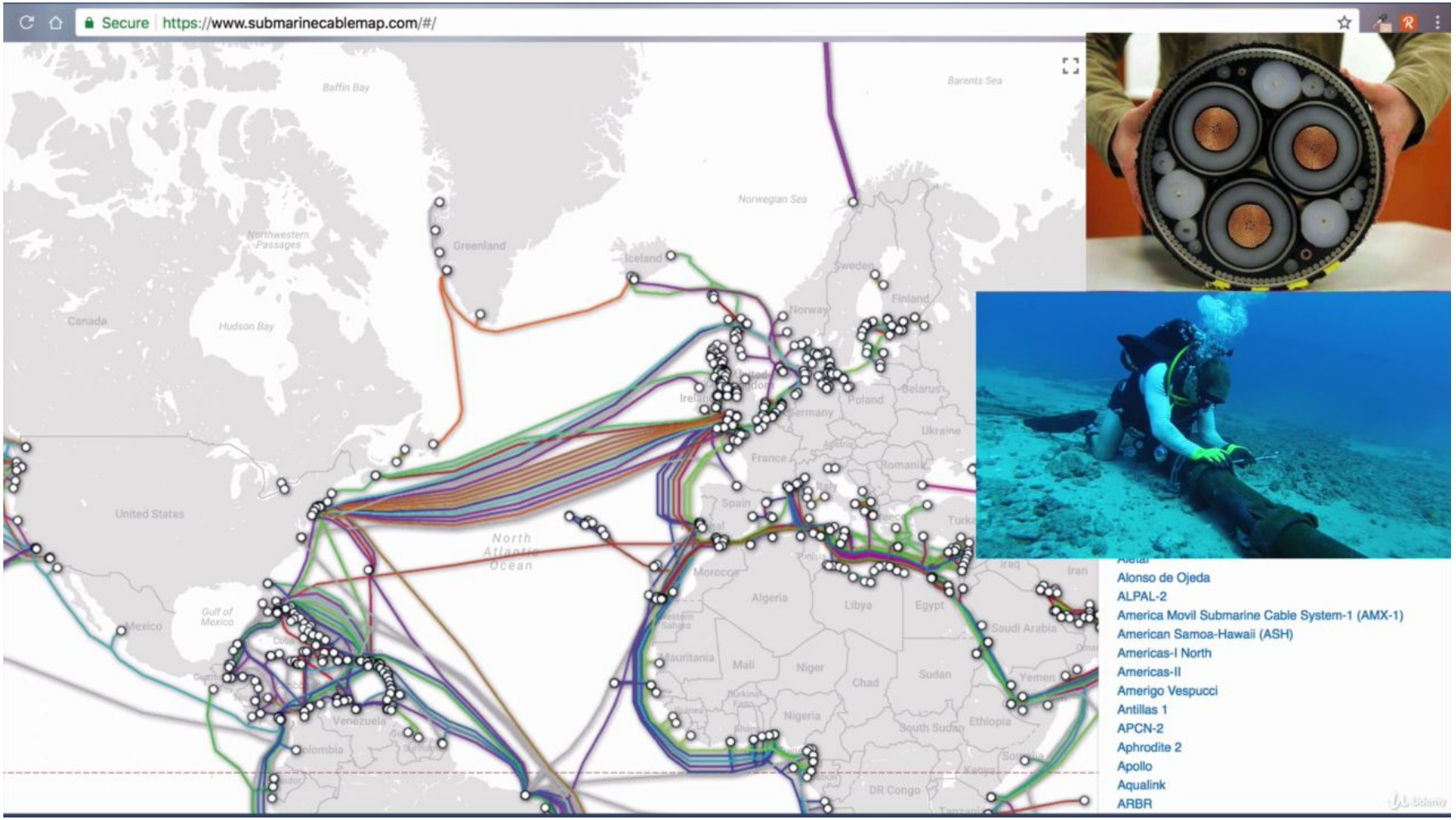
How Internet Works (Continued..)



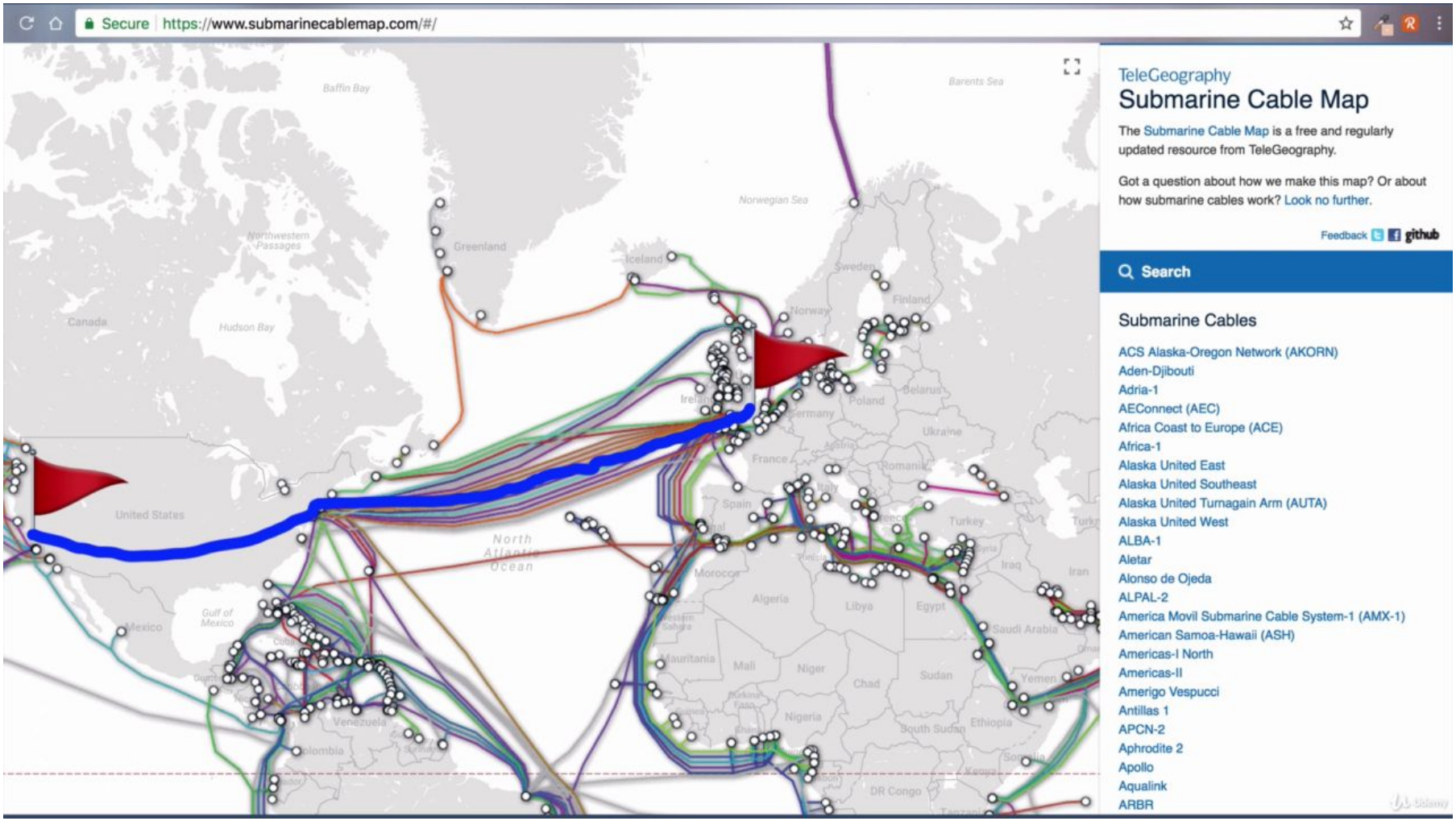
How Internet Works (Continued..)



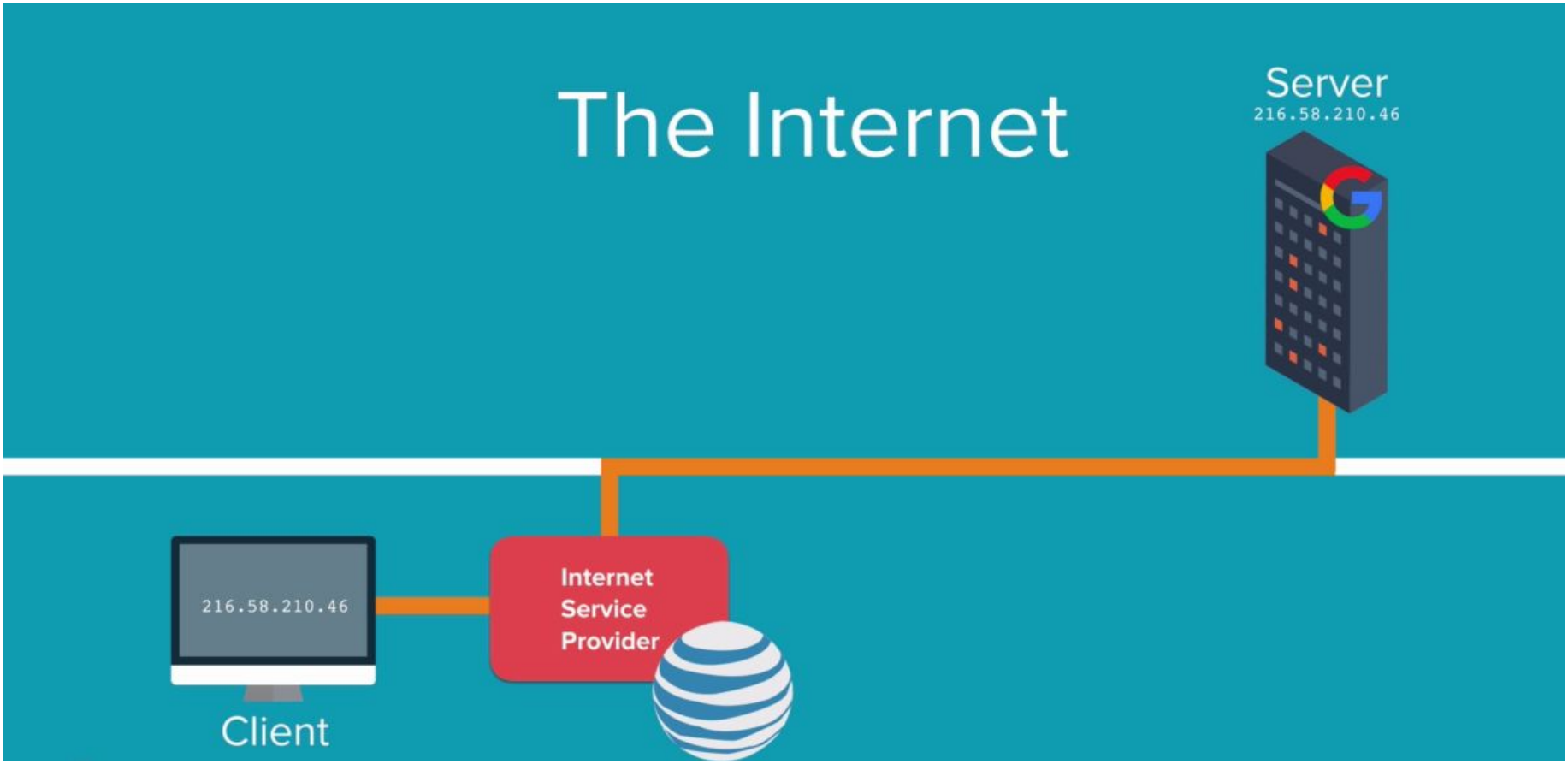
How Internet Works (Continued..)



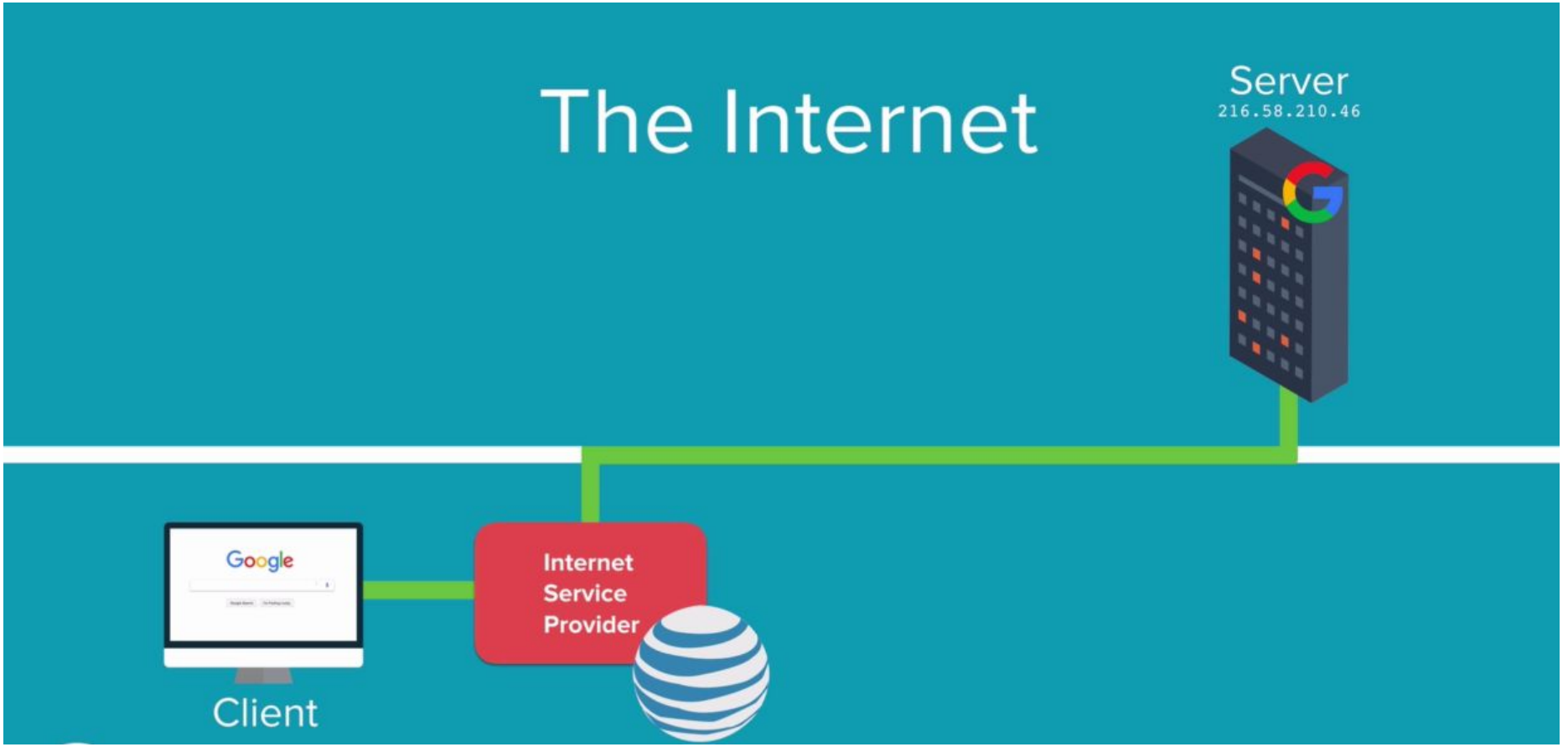
How Internet Works (Continued..)



How Internet Works (Continued..)



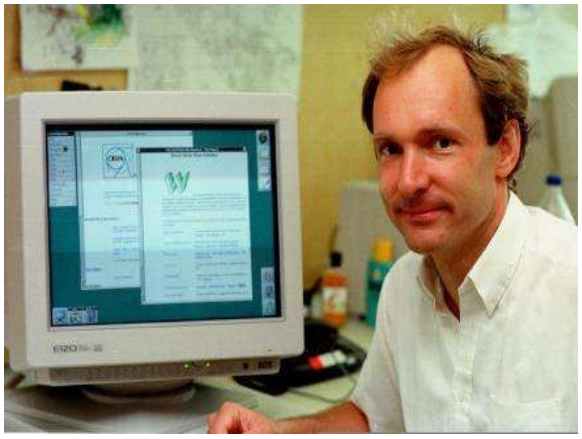
How Internet Works (Continued..)



- Amazingly all happens in a matter of seconds
- Link

The World Wide Web

- It is an information-sharing model that is built on top of the Internet
- Is a system of interlinked hypertext documents
- Accessed via the Internet with a web browser
- 1989-1990 – Tim Berners-Lee invents the World Wide Web at CERN
    - He developed HTML, URLs, and HTTP



The World Wide Web(Continued..)

- A worldwide collection of electronic documents
- Each electronic document is called a Web page
- A Website is a collection of related Web pages
  - **Types of Websites (12)** : Portal, News, Informational, Business/ Marketing, Educational, Entertainment, Advocacy, Blog, Wiki, Social Network, Content Aggregator, Personal

Web Consortium (W3C)

- An organization founded by Tim Berners-Lee In October 1994
- The universe of network-accessible information, the embodiment of human knowledge
- Primary goal is to make the web universally accessible irrespective of ability, language or culture

WWW Components

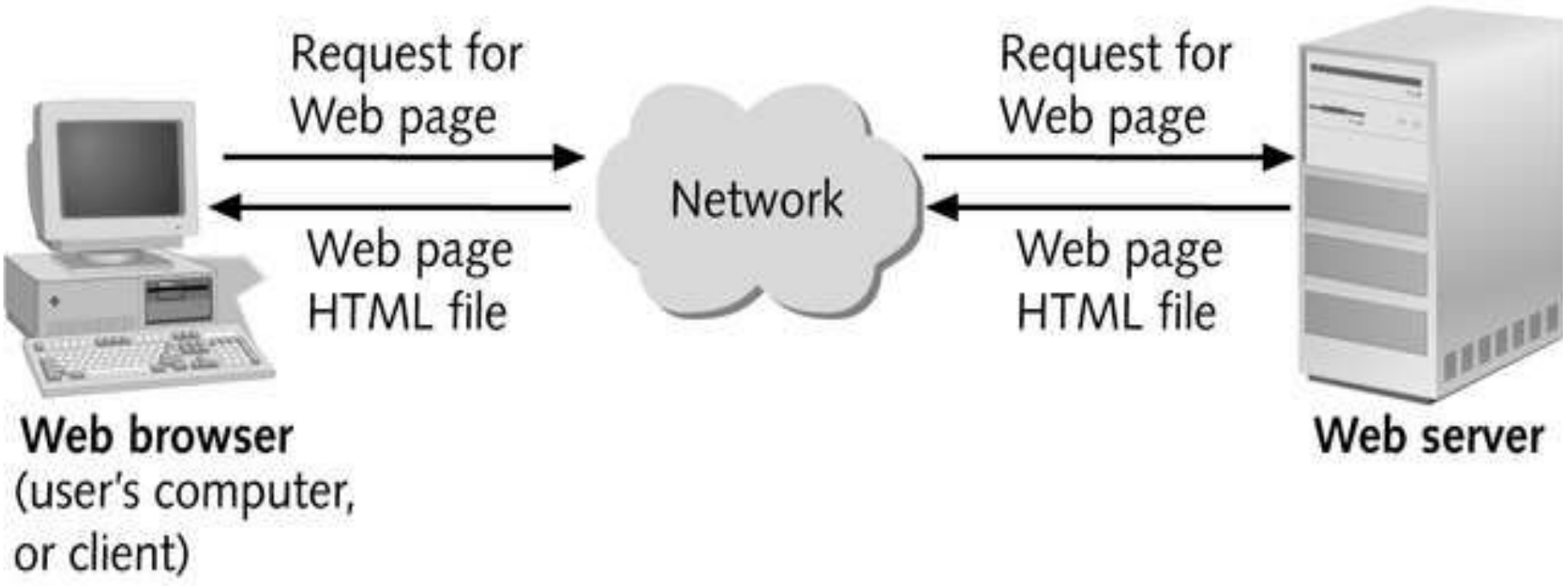
- Structural
  - Clients/Browsers , Servers ,Caches , Internet
- Semantic
  - HTTP, HTML and URL
- Working Schema
  1. Connection
  2. Request
  3. Response
  4. Close

The Web Architecture

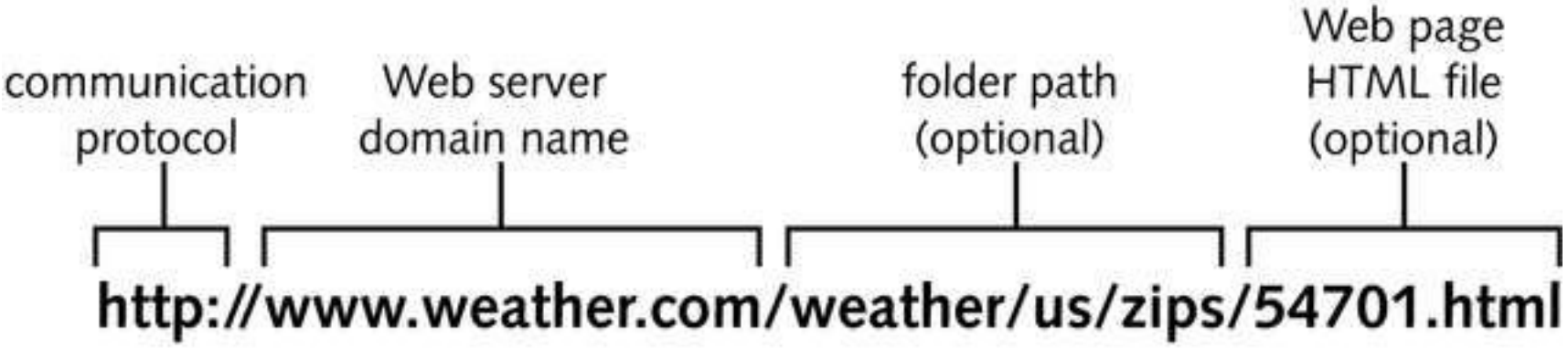
The web is a two-tiered architecture:

› A web browser displays information content, and

› A web server that transfers information to the client



Uniform Resource Locator(URL)



Hypertext Transfer Protocol(HTTP)

- HTTP is a request/response standard of a client and a server
- Typically, an HTTP client initiates a request
- Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs)

HTTP(Continued...)

- The request message consists of the following:
  - Request line
  - Headers (Accept-Language, Accept, ...)
  - An empty line
  - An optional message body
- HTTP defines eight methods ("verbs") indicating the desired action to be performed on the identified resource.
  - › HEAD › GET › POST › PUT › DELETE › TRACE › OPTIONS › CONNECT

HTTP(Continued...)

- HEAD, GET, OPTIONS and TRACE are defined as safe (no side effects)
- POST, PUT and DELETE are intended for actions which may cause side effects on the server mainly create or change data

HTTP(Continued...)

- The first line of the HTTP response is called the status line.
- The way the user agent handles the response primarily depends on the code and secondarily on the response
- Headers.
  - Success: **2xx**
  - Redirection: **3xx**
  - Client-Side Error: **4xx**
  - Server-Side Error: **5xx**

200 - *OK*

201 - *OK created*

301 - *Moved to new URL*

304 - *Not modified (Cached version)*

400 - *Bad request*

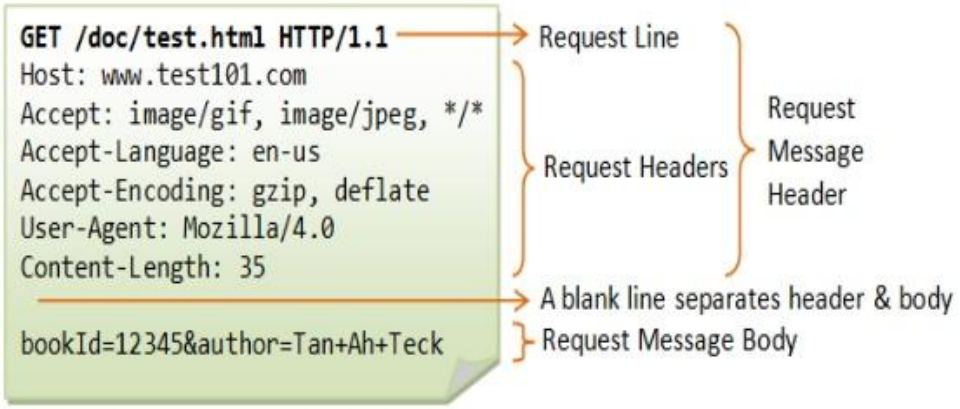
401 - *Unauthorized*

404 - *Not found*

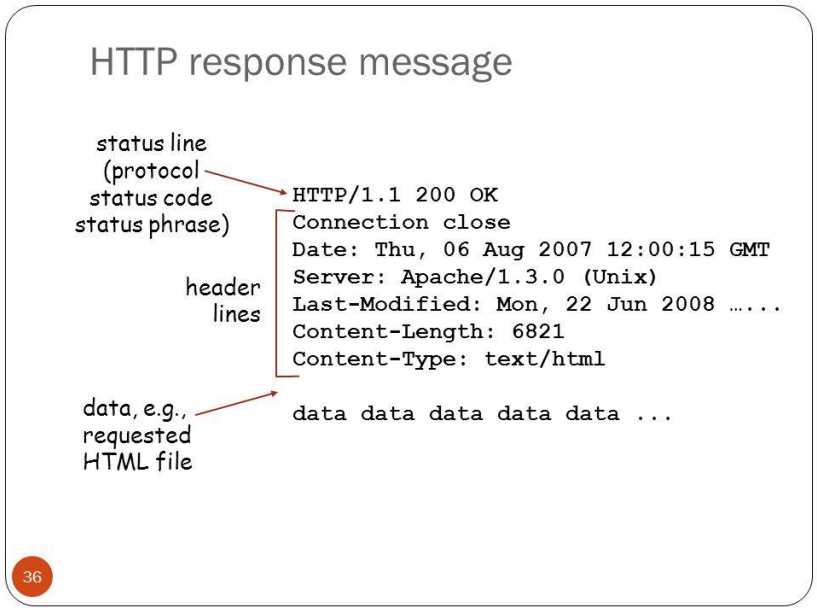
500 - *Internal server error*

Sample HTTP request and reply

Client Request



Server Response



HTTP(Continued...)

- HTTP is a stateless protocol
- Hosts do not need to retain information about users between requests
- Statelessness is a scalability property
  - For example, when a host needs to customize the content of a website for a user
- **Solution:**
  - Cookies
  - Sessions
  - Hidden variables (when the current page is a form)
  - URL encoded parameters
    - (such as /index.php?session_id=some_unique_session_code)

The Web Technologies(Clients Side)

- The Browser is responsible to interpret and display appropriate content

Assignment I

1. History of Internet [The evolution]
2. View the 5 – 10 popular websites of your choice from web archive URL and put your observation and assessment
   a. Web Archive : Link
3. List 5 website each on the 12 categories you learned
   a. Try to view their look in different years web archives
4. What are the guidelines for evaluating the value of a Web site? Try to evaluate 2-5 websites based on the guideline and put your judgment

Resource for this Course

- W3schools[https://www.w3schools.com/]
- MDN[https://developer.mozilla.org/en-US/]
- Devdoc[https://devdocs.io/]

ITSE-2192 Fundamental of web Design and Development

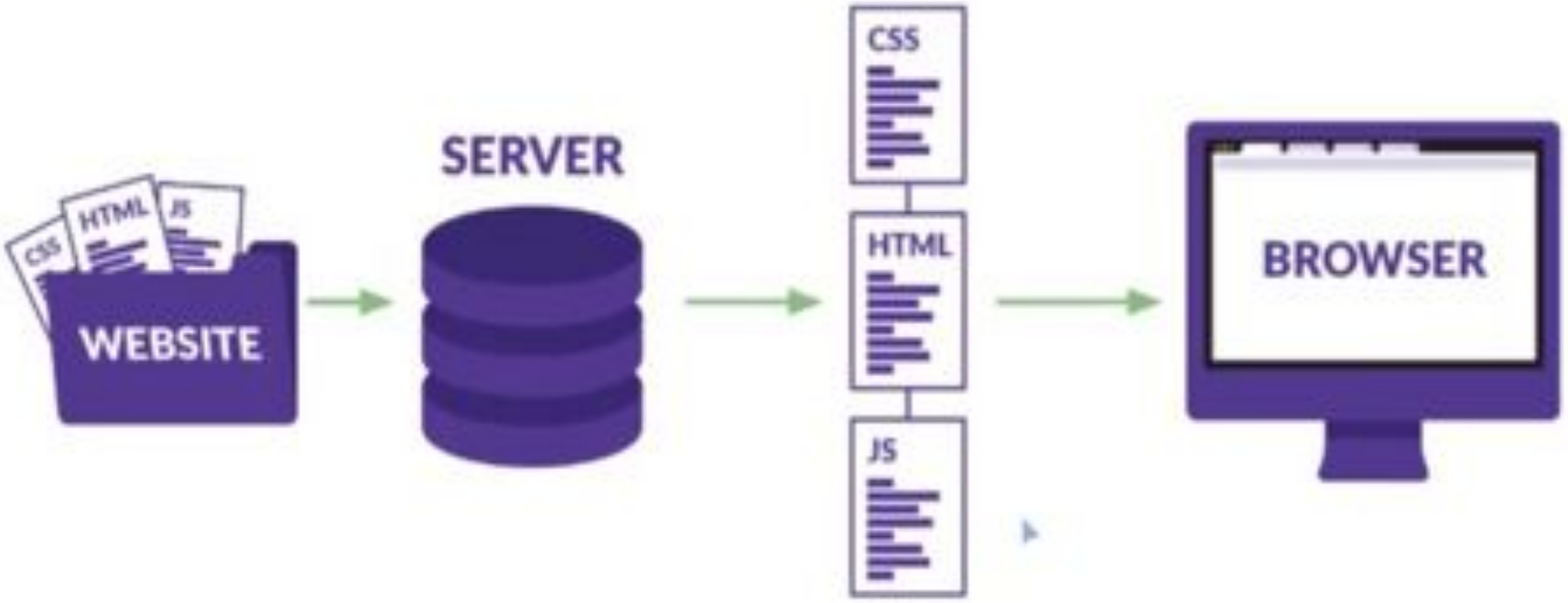
Hyper Text Markup Language(HTML)

Lecture Two

Outline

- How the Web Works (review)
- The HTML Basics
- The Idea of Markup
- HTML Tags and Attributes
- The HTML Anatomy
- Meta Data
- The Evolution of HTML

How the Web Works(review)



- Today we will see HTML

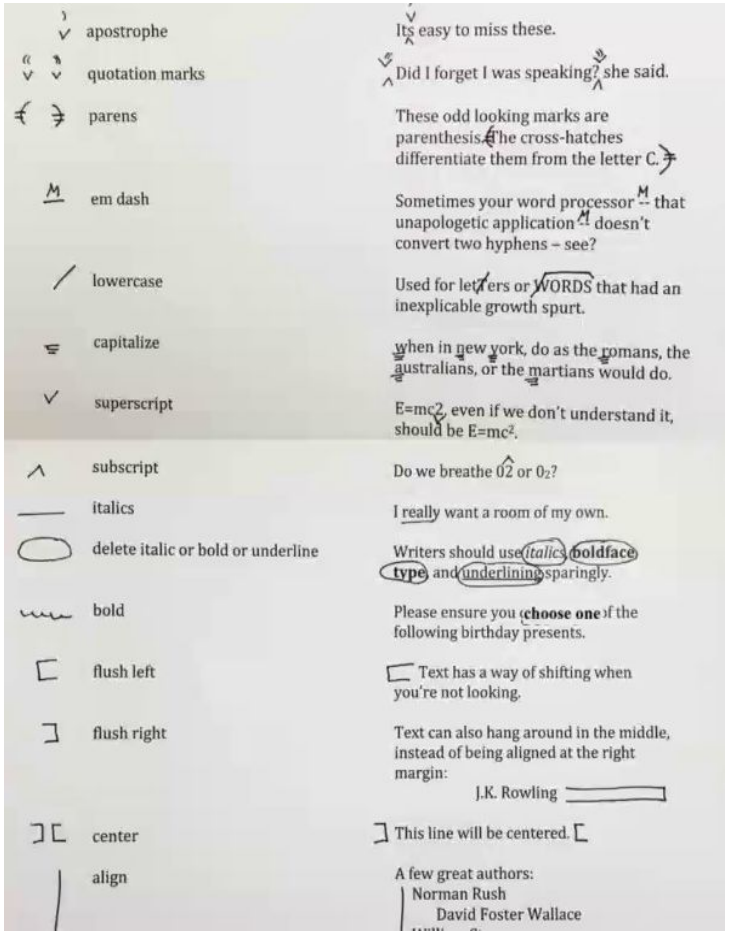
HTML

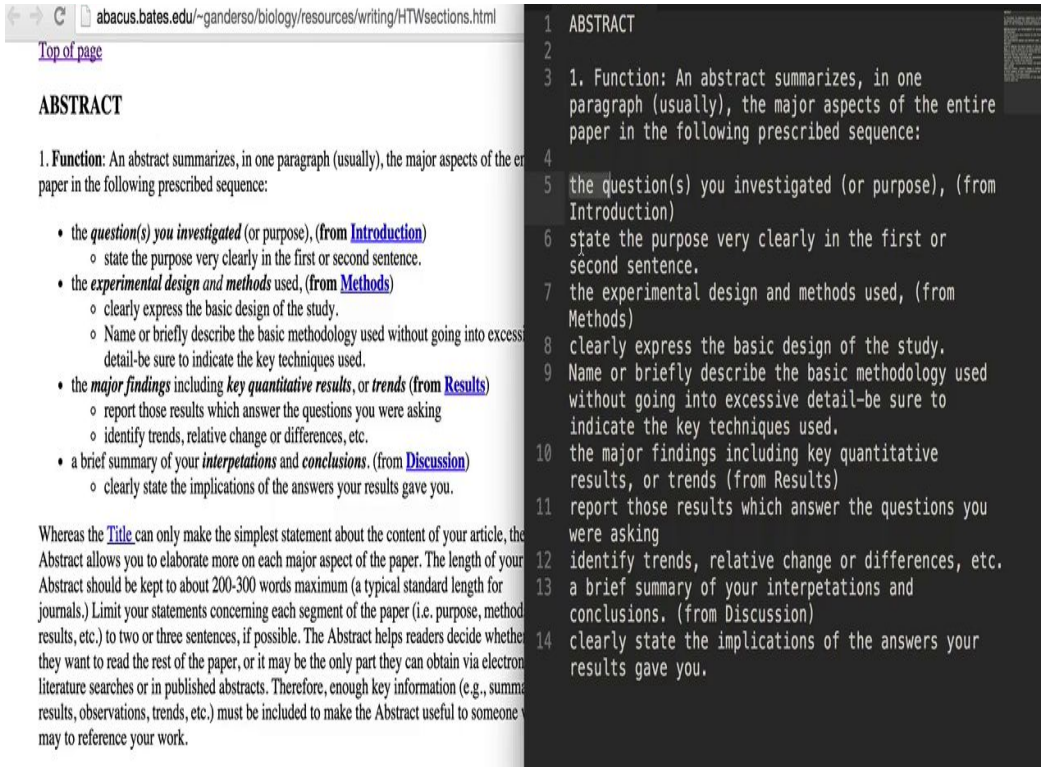
- Stands for Hypertext Markup Language
- Defines the structure and content of a web page
  - “Put image here ..”
  - “Put form here ..”
- The nouns of a web page
- The initial idea of html was to standardize the way people's share information and documents in the internet
- It allows to link electronic document
  - This link has no specific order , it just allow to go from any document to any other document
- It allows publishing and exchanging of scientific and technical documents

The idea of Markup

- The idea come from markups that where placed in manuscripts where editors markup to specify changes or structure for publishers
- How do you tell for publisher to print a scientific paper with the structure of your wish ..
  - E.g There is bullet point , there is large text here
- Markup language is not programing language

The idea of Markup(Continued..)

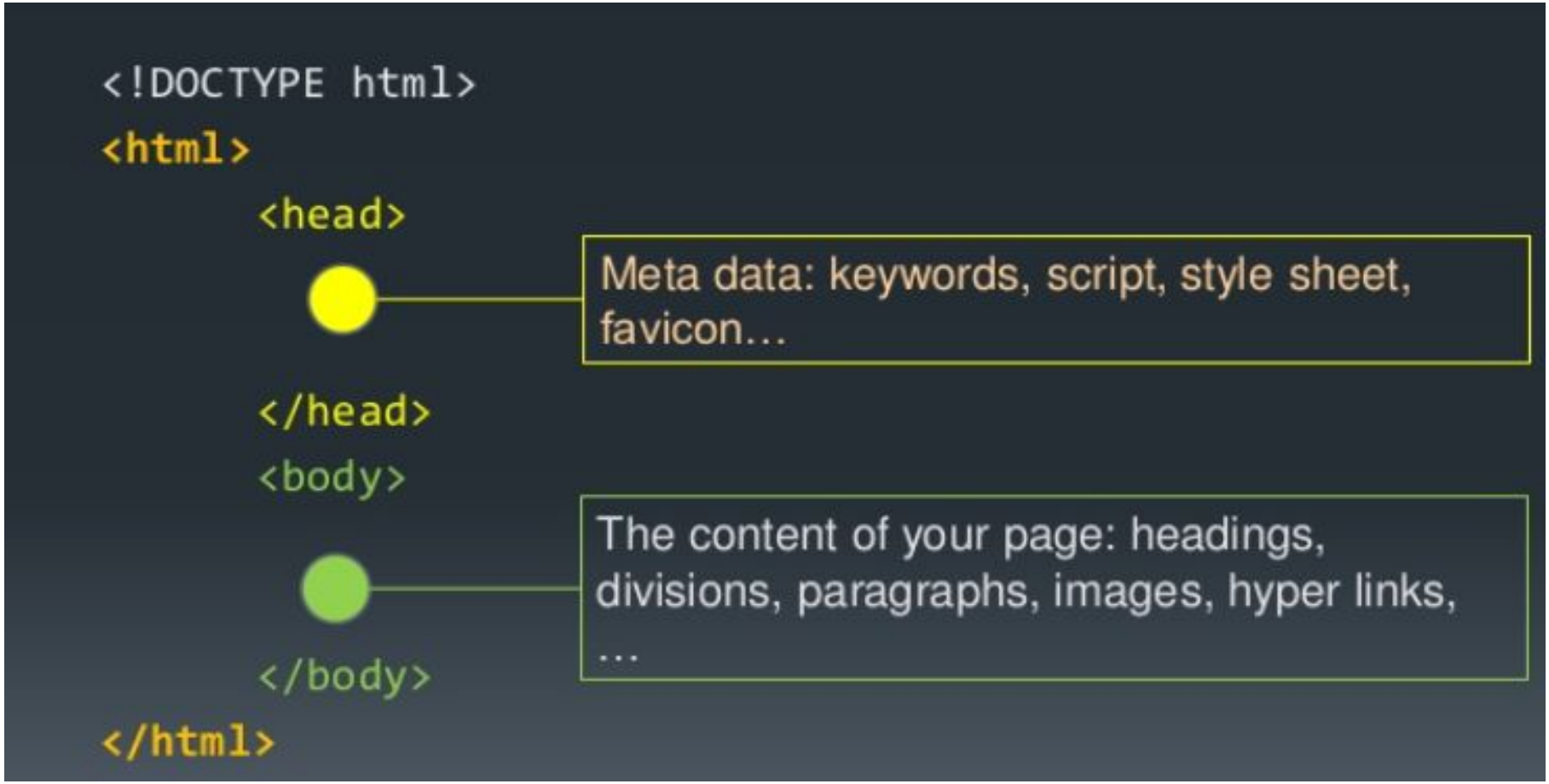




The General Rule

- HTML uses Tags to Express the structure of the document
- A tag is the representation of HTML element that labels piece of content such as heading , paragraph , table , form , etc.
- Browser use this tags to render the content of the page but not display the tags
    - Syntax <starting Tag> content </closing Tag>
- Types of tags [Based on Renderings]
    - **Block** : Tags that take the whole width when they are rendered by the browser E.g <h1- 6>, <p>
    - **Inline** : Tags that do not take the whole width E.g <a>
- Types of Tags [Based on the Enclosing Tag]
    - Empty Tag(Self Closing Tags) : Closing Tag not required
 , <hr>
    - Container Tags : Closing Tag required E.g <p> , , <table>

HTML Document Structure(Anatomy)



The Meta Data

- **<meta>** element describes the contents of the HTML document
- The head element can include any number of meta elements, each providing a single data point using the name/value pair construct
    - The value is specified in the content attribute;
    - The name is either defined in the name attribute or the http-equiv attribute, depending on the type of data being set
- The http-equiv attribute is used to simulate an http response header(i.e content-type, default-style)

```
<meta charset="utf-8" />
<meta name="author" content="Mark J Collins" />
<meta name="description" content="Sample HTML document" />
<meta http-equiv="refresh" content="45" />
```

Attributes

- Tags contain **attributes** that add additional information to the tag
- They are applied on the opening tag
- They follow name value pair
- Syntax
    - <tagname name = “value”>

```
<img src="corgi.png">

<p class="selected">woof woof</p>

<a href="www.google.com">Click me to go to Google</a>

<link rel="stylesheet" type="text/css" href="style.css">
```

- Type : Tag Specific and Global(i.e class, id)

Some Common Tags

```
<h1>I'm a header </h1>
<h2>I'm a slightly smaller header </h2>
<h6>I'm the smallest header </h6>

<p>I'm a paragraph</p>

<button>I'm a button!</button>

<ul>
        <li>List Item 1</li>
        <li>List Item 2</li>
</ul>


<ol>
        <li>List Item 1</li>
        <li>List Item 2</li>
</ol>
```

The Evolution of HTML

- HTML5 is the new standard for HTML
- The previous version of HTML was – HTML 4.01, came in 1999
- HTML5 ~= HTML + CSS3 + JavaScript API
- The future of the web and it is still under development
- Even though browser are trying their best HTML5 is not yet an official standard, and no browsers have full HTML5 support

The Evolution of HTML (Continued..)

- HTML5 is designed to deliver almost everything you want to do online without requiring additional plugins
- It does everything from animation to apps, music to movies, and can also be used to build complicated applications that run in your browser
- HTML5 is also cross-platform (it does not care whether you are using a tablet or a smartphone, a notebook, notebook or a Smart TV)

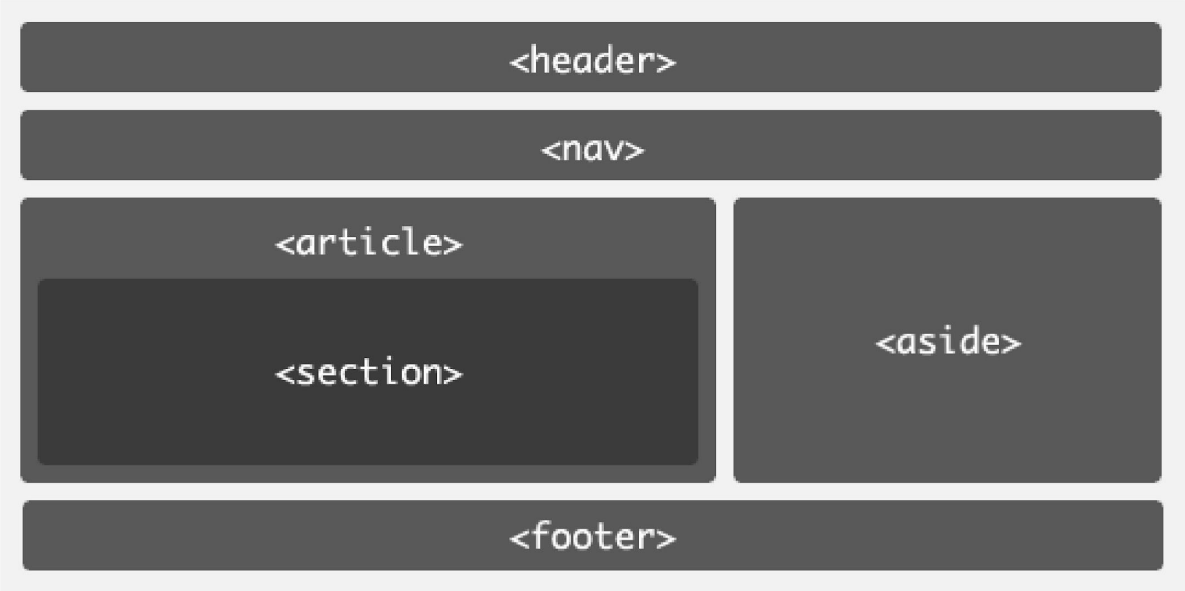
Major Changes on HTML5

1. Improved accessibility, interpretably, performance and simplified Syntax
2. No More of elements like
   - <frame>, <frameset>, <noframes>, , <big>, and <b>, <font>, <acronym>, <applet>, <basefont>, <dir>, <frame>, <strike>, <tt>
   - Obsolete attributes like bgcolor, cellspacing, cellpadding, and valign
3. The New <canvas> Element for 2D drawings
   - The canvas element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly
   - Canvas gives an easy and powerful way to draw graphics using Javascript
4. New <menu> and <figure> Elements

HTML5(Continued..)

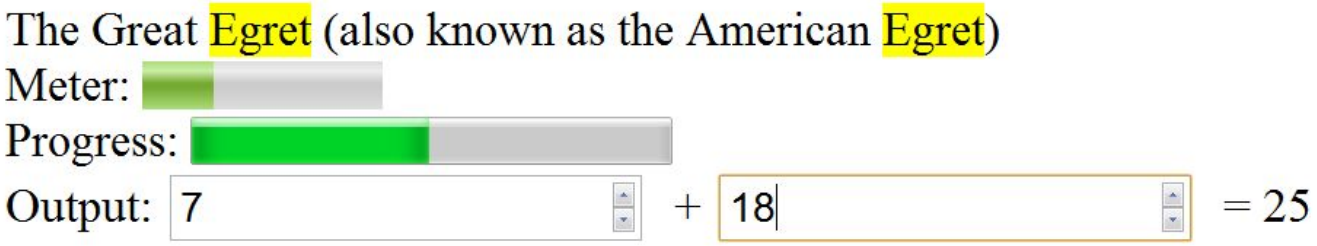
5. New semantic elements, like <article>, <header>, <footer>, <nav>, <section>

   - This was mainly achieved via in previous version of html



HTML5(Continued..)

6. Inline semantic elements
   - Mark: Defines marked text
   - Meter: Represents a scalar gauge providing a measurement within known range, or a fractional value
   - Progress: Represents the completion progress of a task
   - Output: Represents the result of a calculation
   - Time: Represents a specific moment in time



HTML5(Continued..)

7. New <audio> and <video> Elements
   - HTML5 provides a standardised way to play video directly in the browser, with no plugins like Flash, Shockwave or SilverLight
   - No <object> , <embed> elements required
8. New form Elements
   - In addition to GET and POST action , PUT and DELETE are added as an action for form
   - New Controls like email , date , datetime, url, search, range, color, tel ,number
   - New attribute like required, min, max, pattern, placeholder, autocomplete

HTML5(Continued..)

9. Support for local storage with two options to store data on client side
   - Web Storage – supported in all latest browsers.
   - Web SQL Database – supported in Opera , Chrome and Safari
10. Geolocation API provides latitude and longitude on the page where this API is used

11. Drag and Droppable API elements

Reading Assignment

- APIs that are introduced at HTML5
    - For example: Web Notifications , File API, Web Socket ..more
- Difference of the doctype , <meta> tag , <link> tag for external style sheet , <script> tag for external script in HTML5 from previous version
- How HTML5 allow offline web application (The cache manifest)
- Read More about new elements introduced in HTML5(not mentioned in this slide)
- View the HTML Validator Page and practice with it : Link

ITSE-2192 Fundamental of web Design and Development

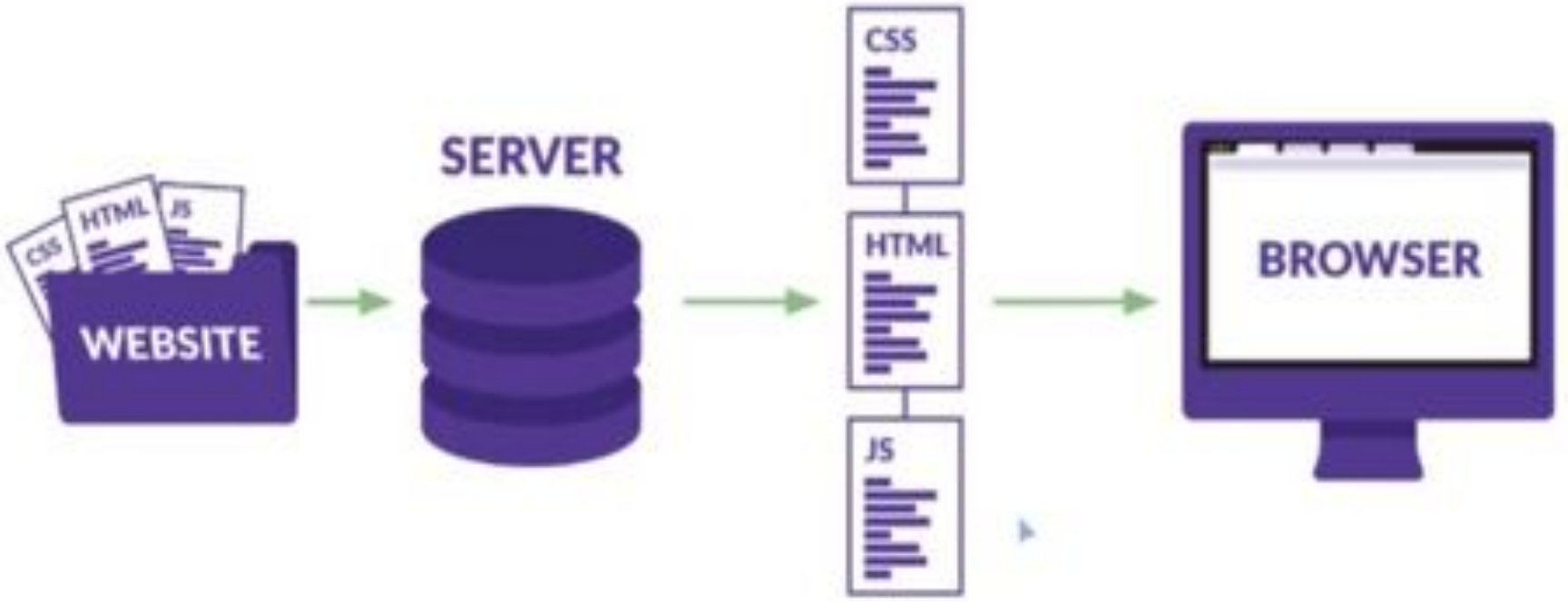
Cascading Style Sheet(CSS)

Lecture Three

Outline

- How the Web works (review)
- Why Cascading Style Sheet(CSS)
- CSS and its Mechanics
- CSS Syntax and Selectors
- Inheritance and Specificity
- The display of content
- CSS Embedding
- The Evolution of CSS

How the Web Works(review)



- Today we will see CSS

Why CSS ?

- Developers in the early 19th has only limited tools set to develop web applications and the result was <u>not fun</u>

- Only HTML was used to style the website **: <font>, , bgcolor = " red" etc.**
- **Layout :** In order to put content side by side developers used **<table >**

Problems with Tables

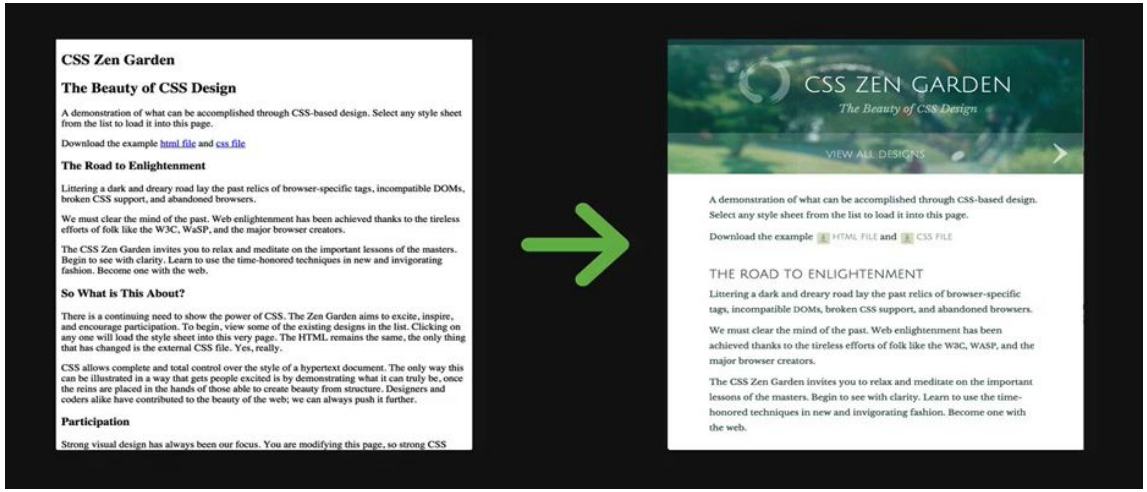
- **Verbose** : To do a very simple thing you have to write a lot of code [they as very wordy]
    - Check the Header part we have done on the lab I
- **Syntax Error :** It is very easy to make syntax errors and mess up with the rendering
    - This might create difficulty in debugging
- **Long Nesting of Tables**: In order to create more complex layout we need to embed tables within tables continuously
    - We will achieve with more columns and rows and flexible layout

What is Our Hope Then ?

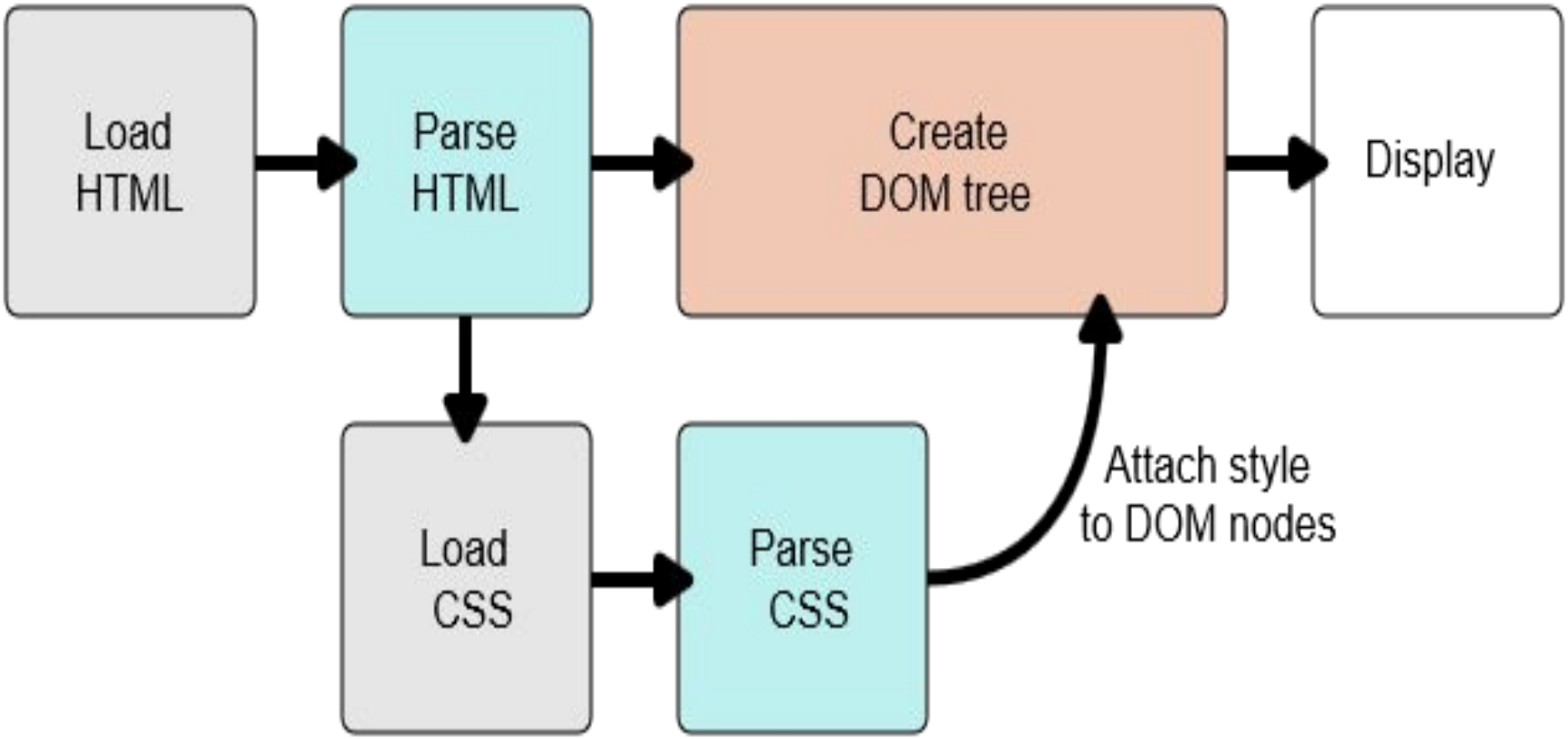
Waiting for a saviour called **CSS** to arrive from somewhere on the sky ..

Cascading Style Sheet(CSS)

- It is a style language where its sole purpose is to apply style to markup languages
- This is to mean that It can not do anything by itself
- This will help us to bring our websites from 19th to the 21 century
- It is the adjective of a webpage or the skin to **skeleton HTML**
  - “Make all text purple”
  - “Give the first image a red border “



How Does CSS Works



The General Rule

- The Syntax for writing a css :

selector{property: value , anotherProperty:value}

- **Selector :** Tag , id, class, other
- **Property :** Ways in which you can style a given HTML element
- **Value :** which chooses one out of many possible appearances for a given property

```
/*Make All h1's purple and 56px font*/
h1 {
  color: purple;
  font-size: 56px;
}

/*Give All img's a 3px red border*/

img {
  border-color: red;
  border-width: 3px;
}
```

TAG , ID and CLASS Selectors

Example : TAG	Example : ID	Example : CLASS
HTML : Say Hello CSS div { background-color : purple; } p { color : purple; }	HTML : Say Hello CSS #id1 { background-color : purple; } #id2 { color : purple; }	HTML : Say Hello CSS .cl1 { background-color : purple; } .cl2 { color : purple; }

Other Selectors

/* All */ * { padding: 0; margin: 0; }	/* Descendant Selector */ li a { text-decoration: none; }
/* Adjacent Selector */ h3 + p { border: 2px solid red; }	/* Attribute Selector */ Input[type="text"] { width: 20px ; }
/* nth of type Selector */ li: nth-of-type(even) { font-weight: bold; }	/* Pseudo-Class Selector */ a: hover { color: green; background-color: skybllue; }

- View this Article : The 30 more you need to memorize (☺)

Inheritance and Specificity

- Setting a property in the parent will affect the child since html structure is tree
- We could have multiple styles that are targeting the same element
  - **ID > Class , attributes and Pseudo-Class > Element(tags) and pseudo elements>Universal(*)**
  - Specify Is calculated by assigning numeric value for each selector
    - The Calculator
  - Note : In Case of equal specificity equality, the latest declarations found in css is applied

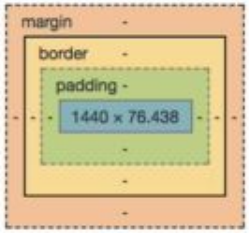
```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Css Demo</title>
<style>
body { color: skyblue;  }
ul{      color: springgreen; }
.highlight { color:yellow }
#sp{ color: red }
</style>
</head>
<body>
   <h3>This is Heading</h3>
   <p>This is Paragraph</p>
   <ul>
      <li id="sp" class="highlight">One </li>
      <li>List Two</li>
   </ul>
</body>
</html>
```

- For More : Link 1 and Link 2

The Box Model

- **MDN :** Each element Every element in HTML has a rectangular box around It which is described as a rectangular box model
    - Each box has four edges : margin edges, border edge, padding edge and

In the Browser



Values in element Size and Color

- Element Size Values [16px = 100% = 1em]
  - Static Value that do not change(E.g width = 200px)
  - Relative to the view port using Percentage: (E.g width = 50%)
  - Relative value to parent box: (E.g width = 1em)
  - Relative to root parent box : (E.g width = 1rem) //only in css3
- Element Color Values
  - Text: (E.g color= yellow)
  - Hexadecimal: (E.g color= #000000) #XX ⇒ [0 - f]
  - RGB: (E.g color=rgb(244,125,255)) # 0 – 255
  - RGBA: (E.g color=rgba(233,123,126,#0-1) //only in css3
    - The alpha(transparent) channel from 0 to 1

CSS Embedding

- **Inline :** within the tag by using the global style attribute
  - E.g <body style="background-color: red">
- **Internal:** Inside the head tag
  - E.g <style> body { background-color: blue; } </style>
- **External :** in external file with .css extension
  - E.g style.css body { background-color: green; }
- Order of Style when applied on the same element
  1. Inline
  2. Internal
  3. External

Note : ***This is where the name "Cascading" comes from . It means that styles can fall (or cascade) from one style sheet to another, enabling multiple style sheets to be used on one HTML document.***

The display of Contents (wrap up)

- Even without css, HTML elements has predefined rule on how it should be displayed
    - **Display: block/inline/inline-block/none**
        - We can not maintain width of inline element
        - none is not the same as the css property **visibility: hidden**
- The Contents is everything : the content is the first thing that determines how large should be displayed and what height and width will be
- The order comes from the HTML : The display is determined by the order of elements in the source code
    - Can change the position with position property
        - E.g position: relative/absolute/fixed ;   top/bottom/left/right = value ;
    - Floating text around element : float : right/left and clear : right/left [don't abuse it for positioning]
- Children's sit on the top of parent : Cone

Evolution of CSS

- Current version of css is version three
- It’s backwards-compatible with older versions of CSS
- What are changed ?
  - Mobile-first mentality
  - Module-based code
  - Web font support gives designers access to way more than just “web safe” fonts
  - It enables faster development, and faster load times
  - Create 2D and 3D transformations, animations, and transitions
  - New colors and image effects
  - Box-sizing has fixed some annoying alignment problems

Assignment II

- Perform the Styles on the following file :
  - Link
- Create this tic tac toe Board with Border and Margin property[to put it in the center] : Link
- Create Flags of Various Countries with the **box model** concept and **position** property [at least 12]
- Does it matter to use relative size(%,em,rem) one styling the size of an element so that the element is responsive ? why ? [write your argument]
  - Hint : consider the browsers capability of providing ***zoom feature***
- We said 16px is default browser width so what does it mean when we say 200px ? How does the browser interprets that ?
- Explain in detail(*in technical terms*) the specific changes in css3

Bonus Challenge

Use the following steps to achieve the following items :

1. Visit the css demo website : csszengarden
2. Try to view all designs [On the right top corner]
3. Choose any of the designs that you want to do it in yourself from scratch
4. Download the HTML and CSS [don't view the css code until you finish]
5. Compare your result with the css given by the Author , evaluate how much % of the techniques did you share with the author's way of doing the css
    a. **Note :** techniques do not have to be the same . E.g you might use float left to achieve side by side , the author might use display inline to achieve the same thing , ..etc
6. [**Contributing**] Once you finish[**1-5**] then try to think of new design that the public can use
    a. When you finish the new design perform a get pull request for the author in github
    b. Note : This will be good for your css skill and for your CV

ITSE-2192 Fundamental of web Design and Development

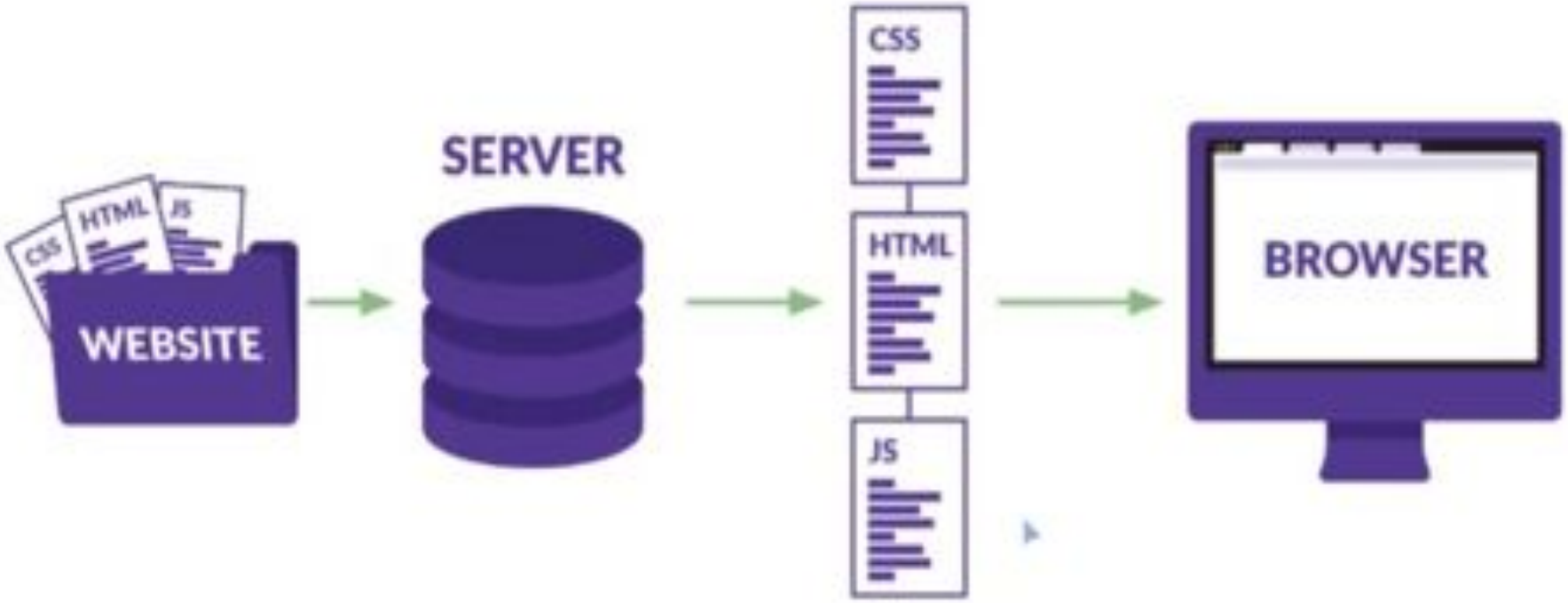
Javascript Basics

Lecture Four

Outline

- How the web Works(review)
- The JavaScript EcoSystem
- The ECMAScript Foundation
- The Basics of JavaScript Language
- How the Language works behind the scene
- How to incorporate JavaScript in HTML

How the Web Works(review)



- Today we will start JavaScript

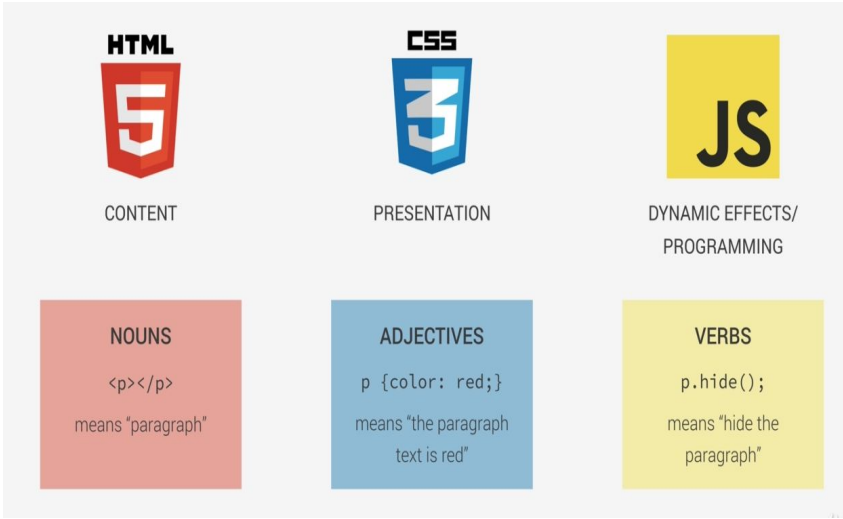
JavaScript Ecosystem

ECMAScript

- It was created to standardize javascript and used by browsers to interpret javascript
- **ECMAScript** provides the rule , detail and guidelines that a scripting language must contain to be considered part of ECMAScript
- Versions
  - **ECMAScript 5.1(2011):** fully supported by browsers but few features are outdated
  - **ECMAScript 2015(ES6):** New standard, partial support, new functions and while others become deprecated
    - Some functions might need transpiler tools like **Babel** in order to work with all the browsers
    - ECMAScript 6 compatibility table : kangax
  - **ECMAScript 2016(ES7):** Added some new functions
  - **ECMAScript 2017(ES8):** Added some new functions like : Async/Await
  - **ES.Next:** Development version, most of the functions are proposals, but not released yet
  -

Javascript

- Created by **Brendan Eich** and initially called LiveScript , called JScript by Microsoft (reverse engineered)
- Scripting language that follows ECMAScript specification
- Javascript [Interpreted Language => Just In TIme Compiled]
  - **Lightweight** : Require less memory and relatively simple syntax
  - **Cross Platform :** Web , Mobile , Desktop
  - **Object Oriented** : prototype based
- Adds **logic** and **interactivity** to a page
  - “Do Some Math”
  - “Change the color when the user clicks ”
  - “Load new data from twitter”
- It is the action or **verb** of webpage



- (Why we learn ?) **1.** *Most popular language in the world and* ***2.*** *It powers the entire modern internet*

What Can We do with Javascript

- **Game :** Unity [Javascript is supported]
- **Business Application:** Typescript for large and scalable applications
- **Native Applications for Mobiles :** CORDOVA
- **Mac and Windows Application :** Electron
- **Web Server :** Node

Variables and Data Type

- **Variable :** is a way in programing language to store a value , so that we can reuse the value repeatedly

  Syntax : var VariableName ;

- **Variable Naming Rule :** can start only with character , _ , $, we can't use reserved keywords , we can not use operators and can not start with digit
- **let and const** : Introduced in ES6
  - **let** : for creating **block** level scope (**{}**)
  - **const** : a constant value that does not change
    - Note : variable with const must be initialized at the declaration

var vs let : Link , Link

E.g

```
var yourName = prompt("What is Your Name ?? ");

alert("You Name is : " + yourName) ;
```

Variables and Data Type (Continued..)

- Types of Data in Javascript
  - Primitive
    - Number : Floating point numbers, for decimals and integers
    - String : Sequence of characters , used for text
    - Boolean : Logical data type that can be true or false
    - Undefined : Data Type of a variable that does not have a value yet
    - Null : Means Non existent
  - Object : This means an object type
- JavaScript is a ***dynamically typed language*** : Data types are automatically assigned to variables
- **We can use typeof function** to check data type
  - **E.g typeof "Hi" //string**

Quiz

Which one is valid declaration and why (💁💁) ?

var myName = “Ababa M.”

var 2maximum = 12

var if = “Some Text”

var var2 = “Var 2”

var myvar/Item = 12

Variables and Data Type (Continued..)

- **Variable Mutation :** Javascript variables are mutable

```
var myVar = "Ababa";

myVar =  18 ;

myVar = true ;

console.log(myVar)
```

why is this happening ??

- **Type Coercion :** Javascript does the conversion of variable data type at concatenation of different data : Read More

```
var name, age, isMarried;
name = "Kabada", age = 12 , isMarried = true ;
console.log(name + ' is ' + age + "years old" +"is he married" + isMarried )
```

Variables and Data Type (Continued..)

Type Coercion : When and How is Useful ??

- There are only three types of conversion : **to string** , **to boolean** and **to number**

When : Values of all Types has associated Boolean value and Useful on **if block**

- Boolean('') // false
- Boolean(0) // false
- Boolean(-0) // false
- Boolean(NaN) // false
- Boolean(null) // false
- Boolean(undefined) // false
- Boolean(false) // false

E.g

```
if(123) {
        console.log("True");
      }
else{
    console.log("False");
    }
```

Javascript Operators

Category	Operator	Name/Description	Example	Result
Arithmetic	+	Addition	3+2	5
	-	Subtraction	3-2	1
	*	Multiplication	3*2	6
	/	Division	10/5	2
	%	Modulus	10%5	0
	++	Increment and then return value	X=3; ++X	4
		Return value and then increment	X=3; X++	3
	--	Decrement and then return value	X=3; --X	2
		Return value and then decrement	X=3; X--	3
Logical	&&	Logical “and” evaluates to true when both operands are true	3>2 && 5>3	False
	||	Logical “or” evaluates to true when either operand is true	3>1 || 2>5	True
	!	Logical “not” evaluates to true if the operand is false	3!=2	True
Comparison	==	Equal	5==9	False
	!=	Not equal	6!=4	True
	<	Less than	3<2	False
	<=	Less than or equal	5<=2	False
	>	Greater than	4>3	True
	>=	Greater than or equal	4>=4	True
String	+	Concatenation(join two strings together)	“A”+”BC”	ABC

Note : Equality : == (perform type coercion) , === (equality of type and value)

Javascript Operators (Continued..)

Operator Precedence

Precedence Table : MDN Link

```
var result = 30 - 9 >= 18

console.log(result);

what is the output ? why ?
```

Associativity : left to right or right to left

```
var x, y ;
x = y = (3+5) * 4 - 6;

console.log(x, y);

what is the output ? why ?
```

Control Structure

Conditional Statement

if else

```
var fName = " kabada";  var age = 16;

if(age < 13)
 {
   console.log(fName + "is a Boy . ");
}
else if(age >13 && age < 20)
 {
   console.log(fName + "is a Tennager. ");
}
else
 {
   console.log(fName + "is a Man ");
}
```

Can you do this with ternary operator ??

Control Structure (Continued..)

Conditional Statement

switch

```
const  paymentMethod = “cash”;

switch(paymetnMethod)
{
case ‘cash’: console.log(“Your payment is in cash”) ;
break;

case ‘check’: console.log(“Your payment is in check”) ;
break;

case ‘card’: console.log(“Your payment is in card”) ;
break;

default : console.log(“Your payment is in check”) ;

}
```

Control Structure (Continued..)

Looping

For Loop	**Statements**
for(let i=0; i< 10; i++) { console.log(i); }	**break ;** **continue;** what does they do ??
While Loop let i = 0 ; while (i<10) { console.log(i); i++ }	other loops : do while → **forEach** [will see later]

Function

- Funcion is set of related instructions written in a a block code that is supposed to execute as per wish and help us to achieve the DRY principle

Syntax

Practice fun On

👇👇

Karel Board

```
function functionName(parameter1, parameter2 )
{    // opening
     // Function Body
} // closing
```

Function Steps

1. Creating and Definition of the function
2. Invoking the function : functionname(arg1, arg2);

Function (Quiz)

Can you define two functions

1. One to calculate age of a person
2. Other that decided weather the person has retired job or not that invoke age calculator

Function (Continued..)

Some notes on functions :

1. function can return a value by using the return statement at the end
2. We can pass default value of parameters if we think the functions can be invoked with missed parameters : **parameter = "Default Value "**
3. Functions Can be written as expression [**function expression**]

```
E.g
const sum = function(no1, no2 = 12) { return no1 + no2}
console.log(sum(12,14))
```

4. Functions can be invoked immediately [IIFEs]

```
E.g
(function(tec)
{
console.log("Learning" + tec);
})  ("Javascript IIFs");
```

Arrays and Objects

- Arrays : Collations of values

Syntax :

- **Creating Array**
  - var names = [‘Ababe’, “Kebede ”];
  - var years = new Array(1990,1998,2000);
- **Accessing Members**
  - names[index] //it is zero based
- **Array Length**
  - names.length
- **Assigning Value (mutating array data)**
  - names [index] = value
- **Useful Methods**
  - push, unshift, pop , shift, indexof

Arrays and Objects (Continued..)

Iterating over Array

forEach :Easy built-in way of iterating over an array

Syntax arrVar.forEach(function(item, index, array){ }, object)

E.g

```
const colors = [ "Red" , "Blue" , "Orange", "Green"]

colors.forEach(function(color)

{
  // color is a placeholder , you can call it whatever variable name you want
   console.log(color);

}
);
```

Arrays and Objects (Continued..)

Objects : Key value pair where an order of items does not matter and can be accessed via name [unlike index with arrays]

Syntax : name = {key : value}, key : value } ; or name = new Object();

E.g

```
const john = {
    firstName : 'John' ,
    lastName : "Smith" ,
   birthYear : 1990 ,
  family : ['Mark', 'Bob', 'Anane'] ,
  calcAge : function(now) {    return now - this.birthYear  }
}
```

Arrays and Objects (Continued..)

Syntax

- **Accessing Elements : objectname.property or objectName['property']**

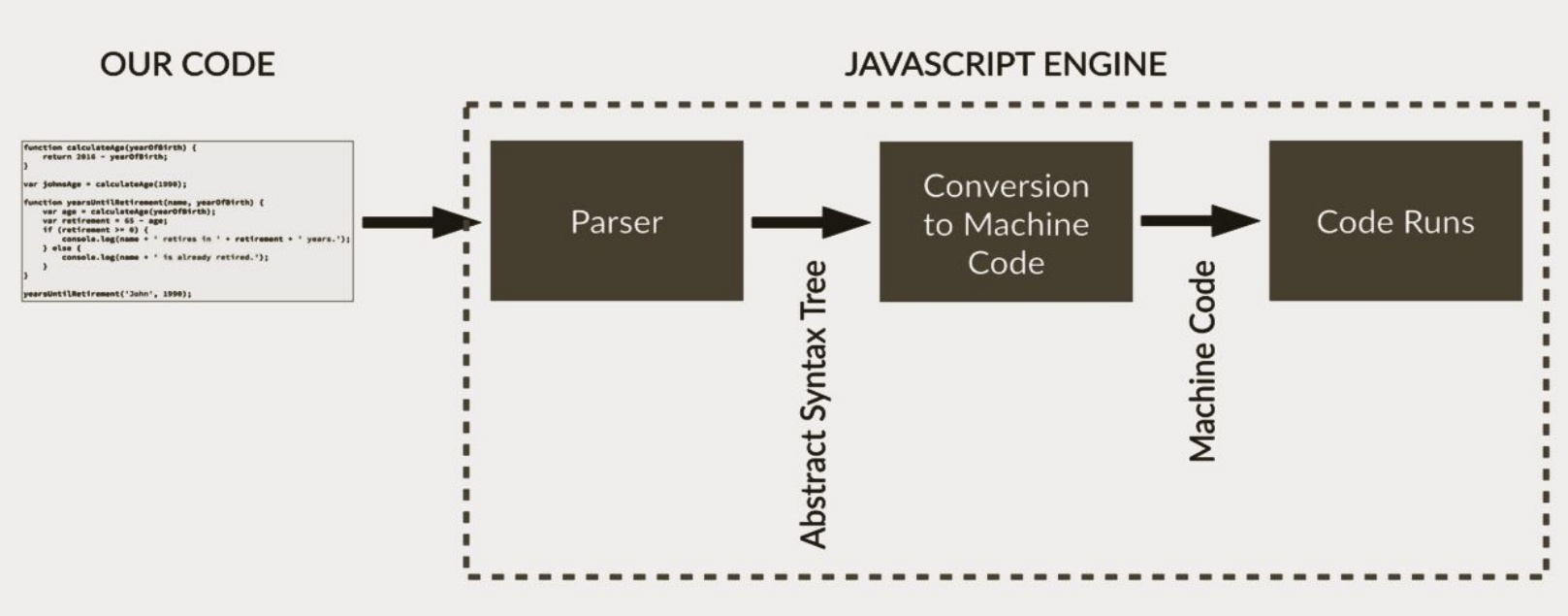
E.g

```
console.log(john.firstName);

console.log(john.calcAge(2020));

console.log(john['lastName']);
```

- **Mutate :**
  - **E.g John.birthYear = 1980**

How the Language work Behind the Scenes

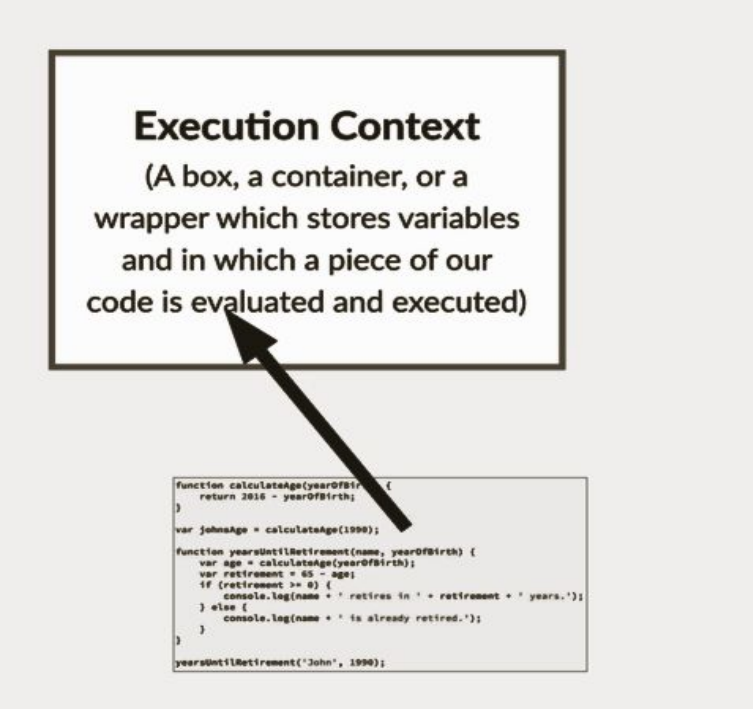
- **Execution :** Javascript is hosted in an environment which is most typically a web browser [Others : Node, ..]
  - Various things happens when a code is executed
  - The host has a javascript engine that executes the code[**Google's V8 used by chrome,** Spider Monkey, ,Chakra, Javascript Core ,Rhino, , .etc]



- Different engines might has variations but the overall idea is the same

How the Language work Behind (Continued..)

Execution Context : A wrapper that helps to manage the code that is running

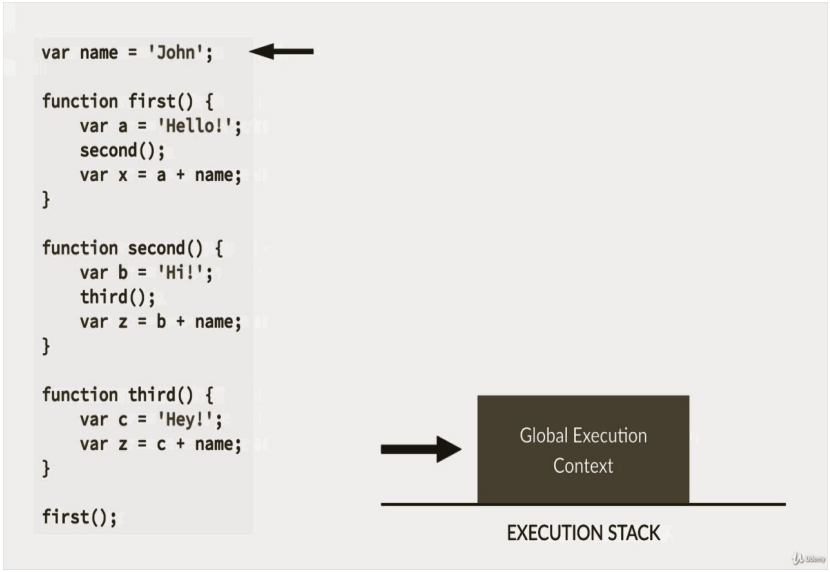


- Code that is not inside any function is in global context
- Associated with global object , that is **window** in the context of browser

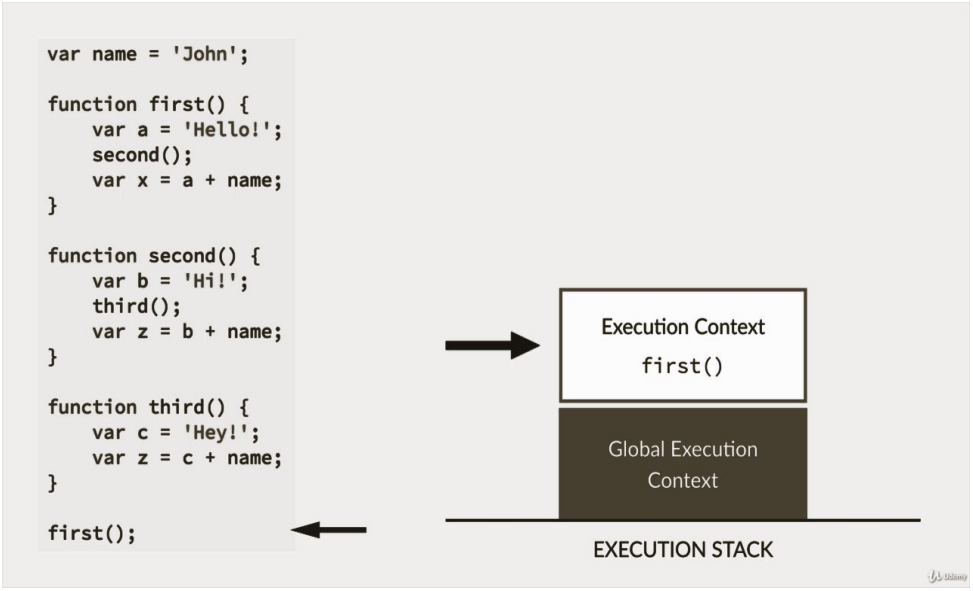
- **lastName == window.lasteName** // true

How the Language work Behind (Continued..)

Execution Stack : Storing the execution contexts in a stack



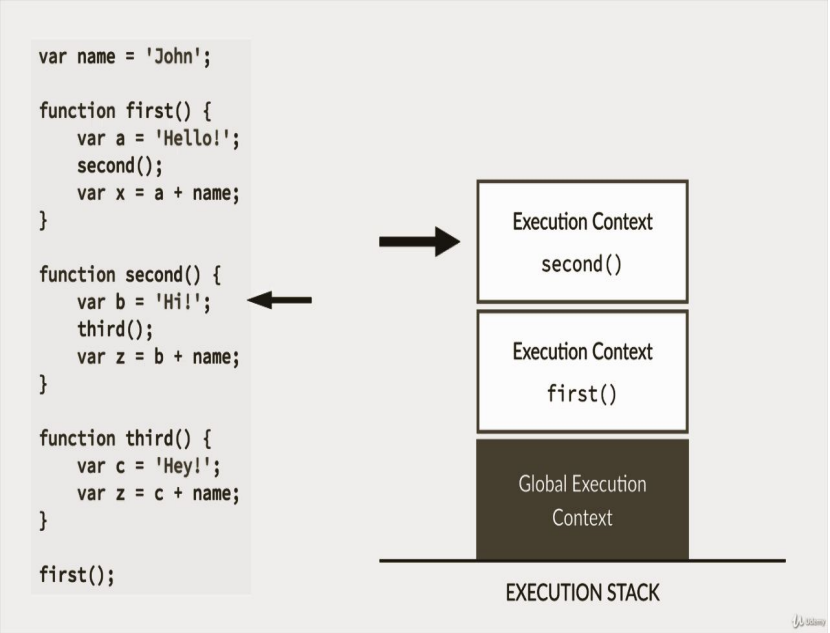
1



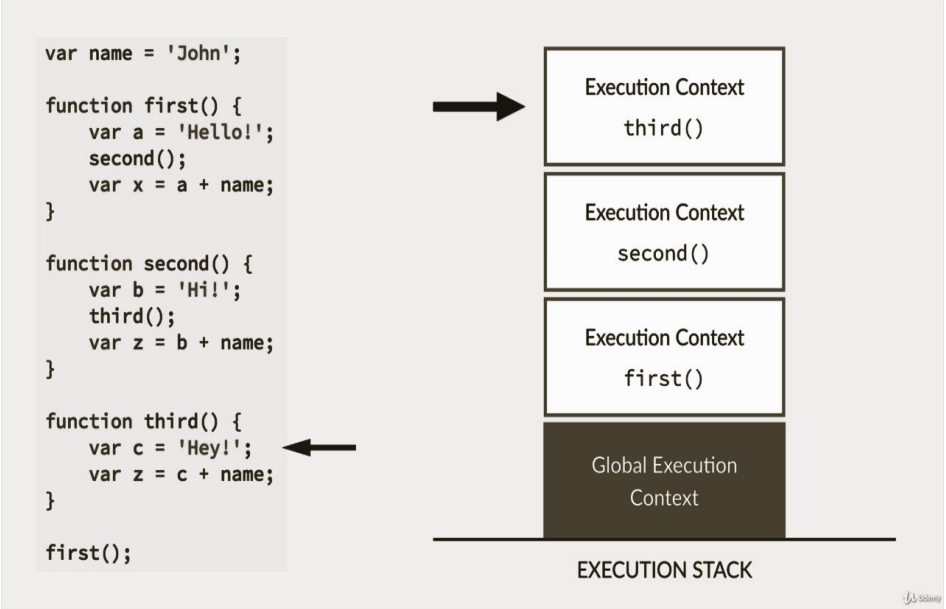
2

How the Language work Behind (Continued..)

Execution Stack : Storing the execution contexts in a stack



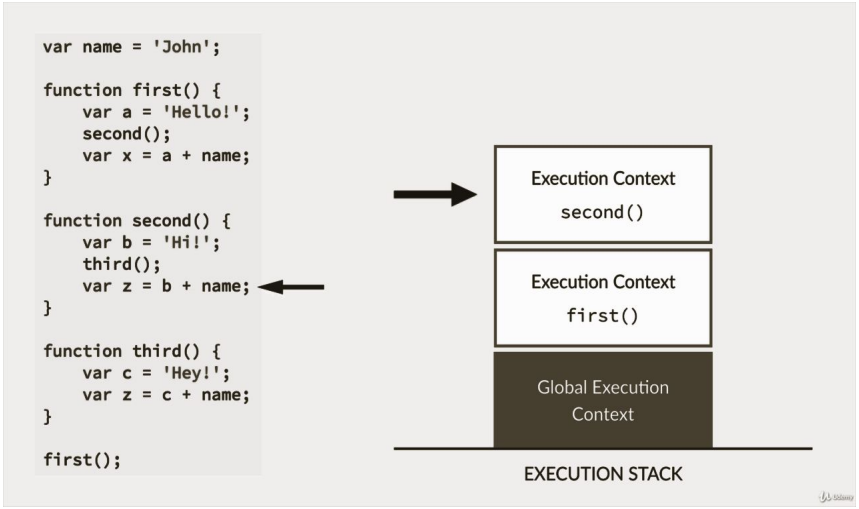
3



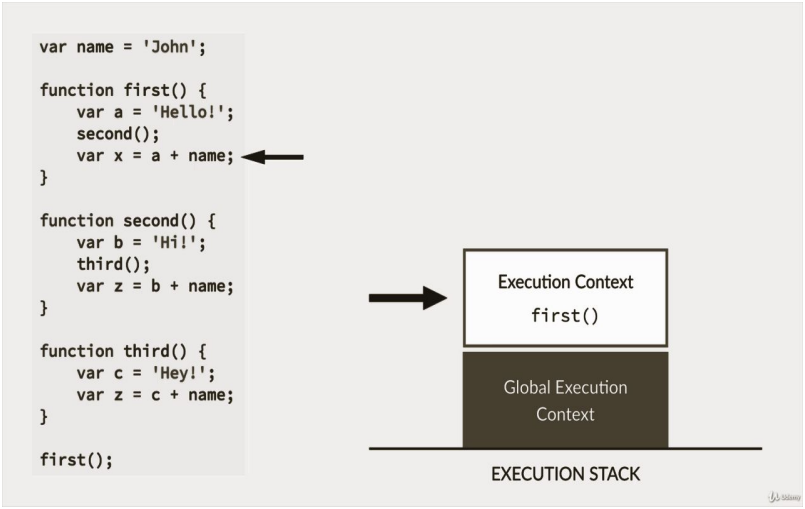
4

How the Language work Behind (Continued..)

Execution Stack : Storing the execution contexts in a stack

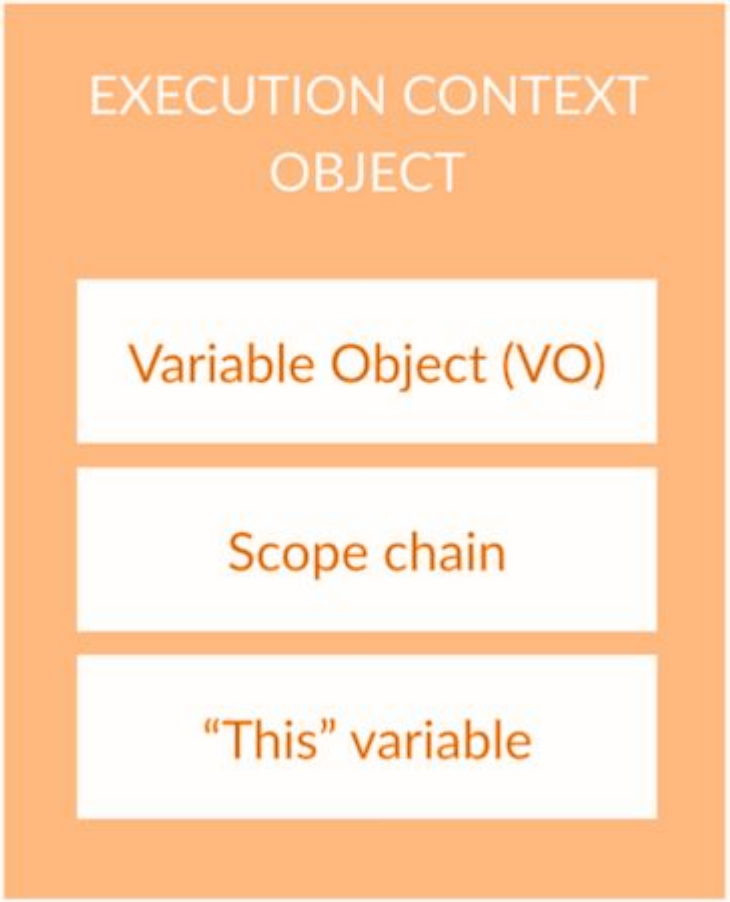


5



6

How the Language work Behind (Continued..)



The Execution happens in two phases

1. ***Creation Phase***
   a. Creation of Variable Object(VO)
   b. Creation of Scope Chain
   c. Determine the value of ‘this’ variable
2. ***Execution Phase***

   The code of the function that generated the current execution context is run line by line

How the Language work Behind (Continued..)

- **The Variable Object**
  - The argument object is created , containing all the arguments that were passed into the function
  - Code is Scanned for **function declarations** : for each function, a property is created in the Variable Object , **point to the function**
  - Code is scanned for **variable decelerations**: for each variable, a property is created in the Variable Object , and set to **undefined**
    - This phenomena is Called **Hoisting** : means functions and variables are available before the code starts executing

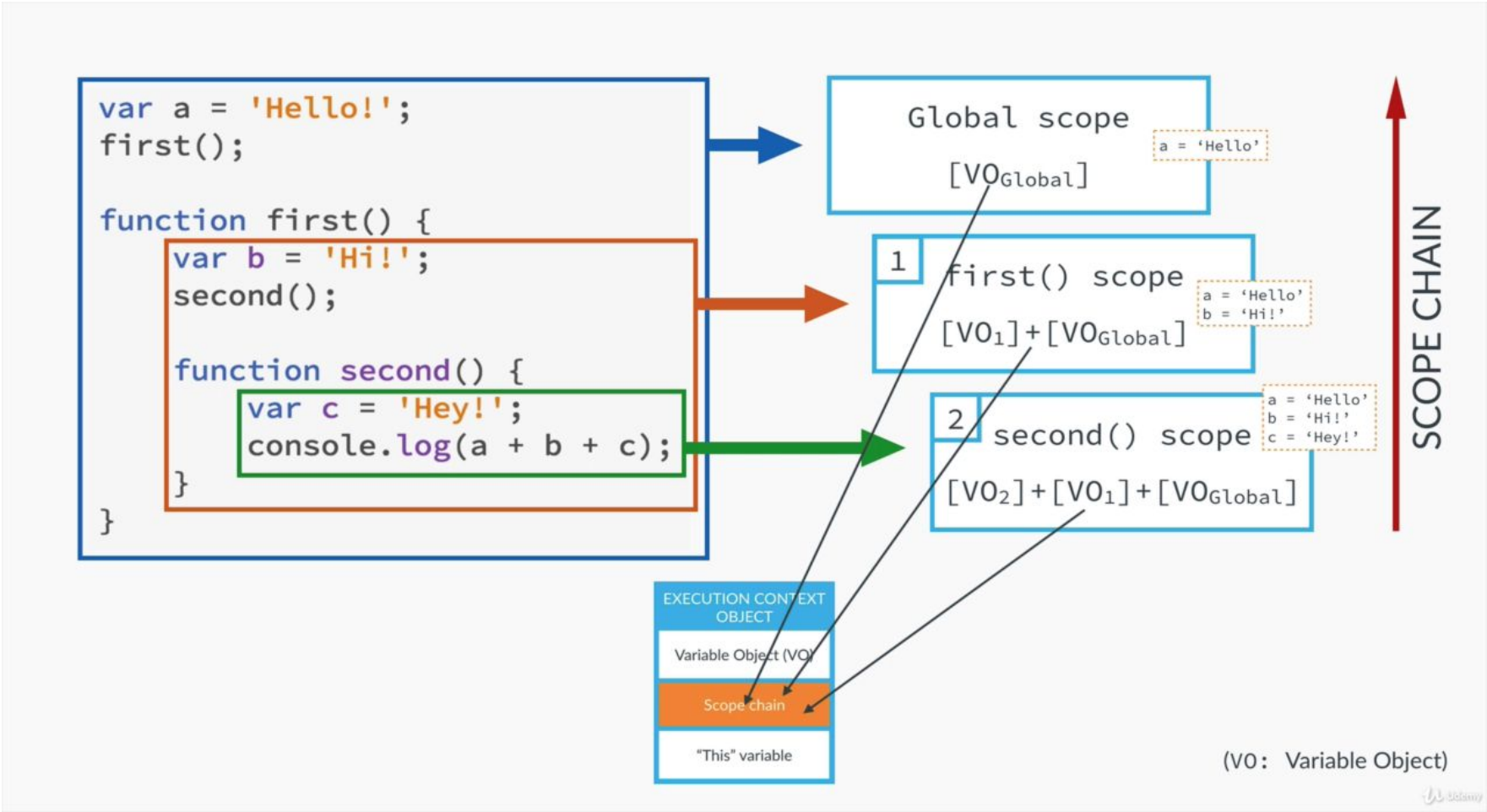
```
//hoisting for function
 calcAge(1998);
 function calcAge(year)  {   console.log( now - year); }

//hoisting for variable
 console.log(age);
 var age = 12;
 console.log(age);
```

How the Language work Behind (Continued..)

- **The Scop Chain**
  - Scoping answers the question “Where can we access a certain variable **?** ”
  - Each new functions created a scope: the space/environment, in which the variables that it defines are accessible
  - **Lexical Scoping :** a function that is lexically within another function gets access to the scope of the outer function

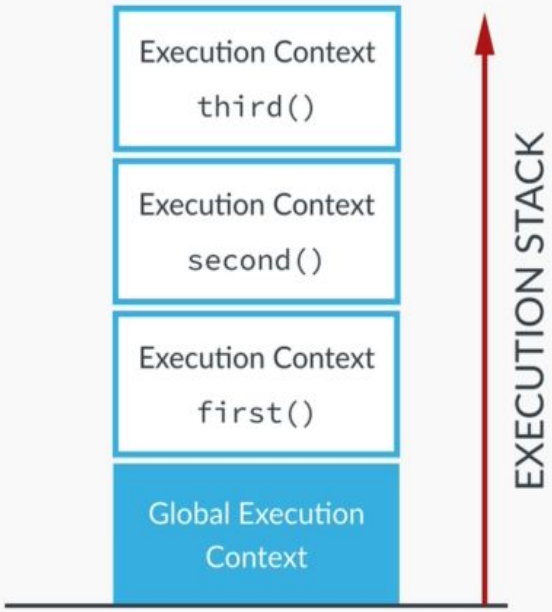
How the Language work Behind (Continued..)

- The Scop Chain

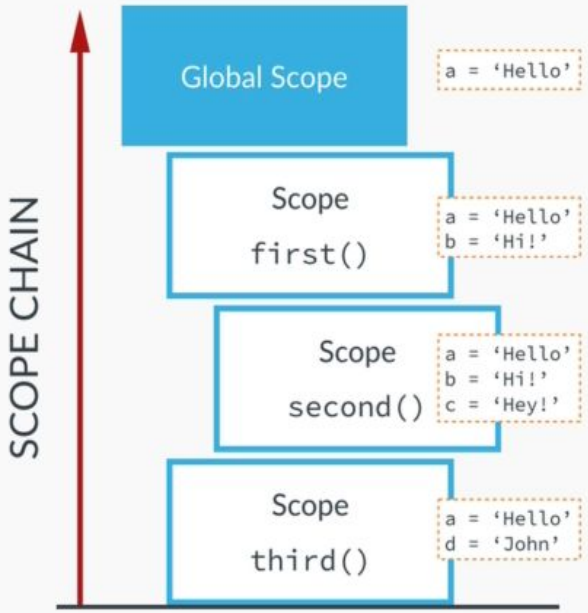


How the Language work Behind (Continued..)

- **The Execution Context** vs **The Scope Chain**




```
var a = 'Hello!';
first();

function first() {
    var b = 'Hi!';
    second();

    function second() {
        var c = 'Hey!';
        third()
    }
}

function third() {
    var d = 'John';
    console.log(a + b + c + d);
}
```



Order in which functions are **called**

≠

Order in which functions are **written lexically**

How the Language work Behind (Continued..)

- The **'this'** keyword
  - **Regular function call:** the this keyword points at the global object(the window object, in the browser)
  - **Method call:** the this variable points to the object that is calling the method
- **Note:** The this keyword is not assigned a value until a function where it is defined is actually called

```
console.log(this)              //window
function testThis()
{
   console.log(this);          //window
}
var john = {

   calcAge : function() {      console.log(this);   }
}
john.calcAge();               //john
```

Incorporating Javascript in HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Incorporate JavaScript</title>
</head>
<body>

<h1>JavaScript Basics </h1>

<!-- Javascript Inline Script  -->
<script >

  console.log("Helo World !!!!")

</script>

<!-- Javascript External  Script-->
    <script src="script.js"></script>

</body>
</html>
```

Reading Assignment

1. Is Javascript Interpreted Language in it entirety ? : Check this Link and Make Up your justification
2. The history of “typeof null” : Link
3. Explain in detail why hoisting is different with let and const ?
4. Semicolons in JavaScript: To Use or Not to Use? : Link
5. Expression vs Statement in Javascript ?

ITSE-2192 Fundamental of web Design and Development

DOM and BOM

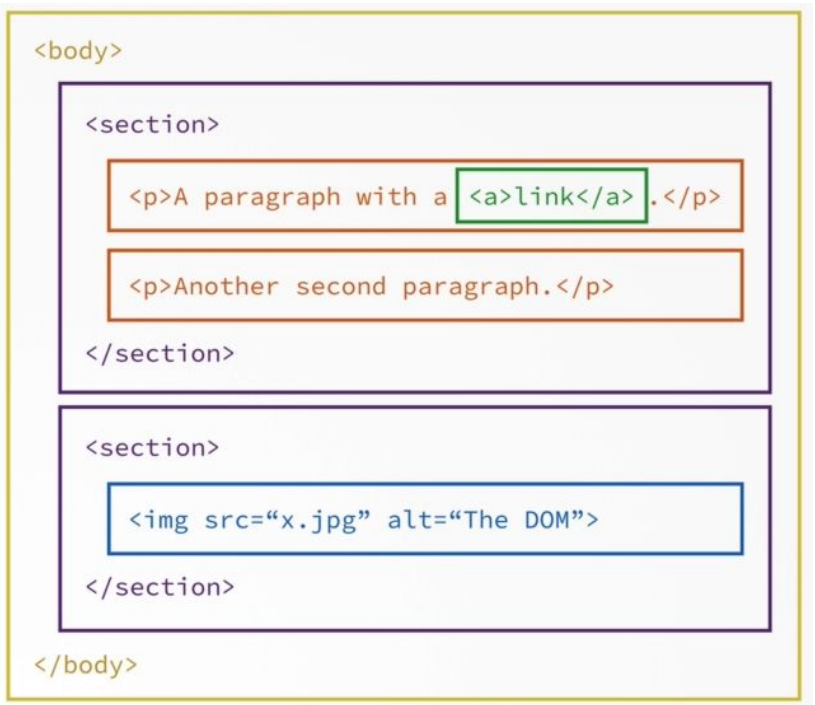
Lecture Five

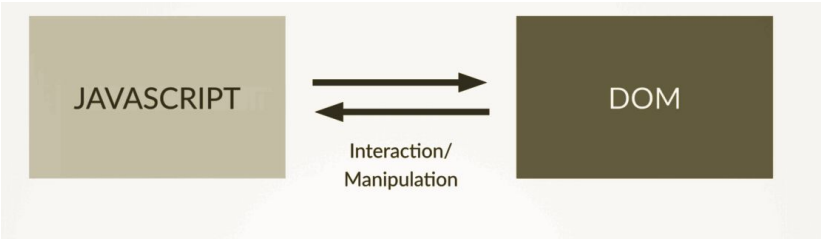
Outline

- Introduction to DOM
- Traversing the DOM
- Selecting an Element in DOM
- DOM Traversing
- The BOM

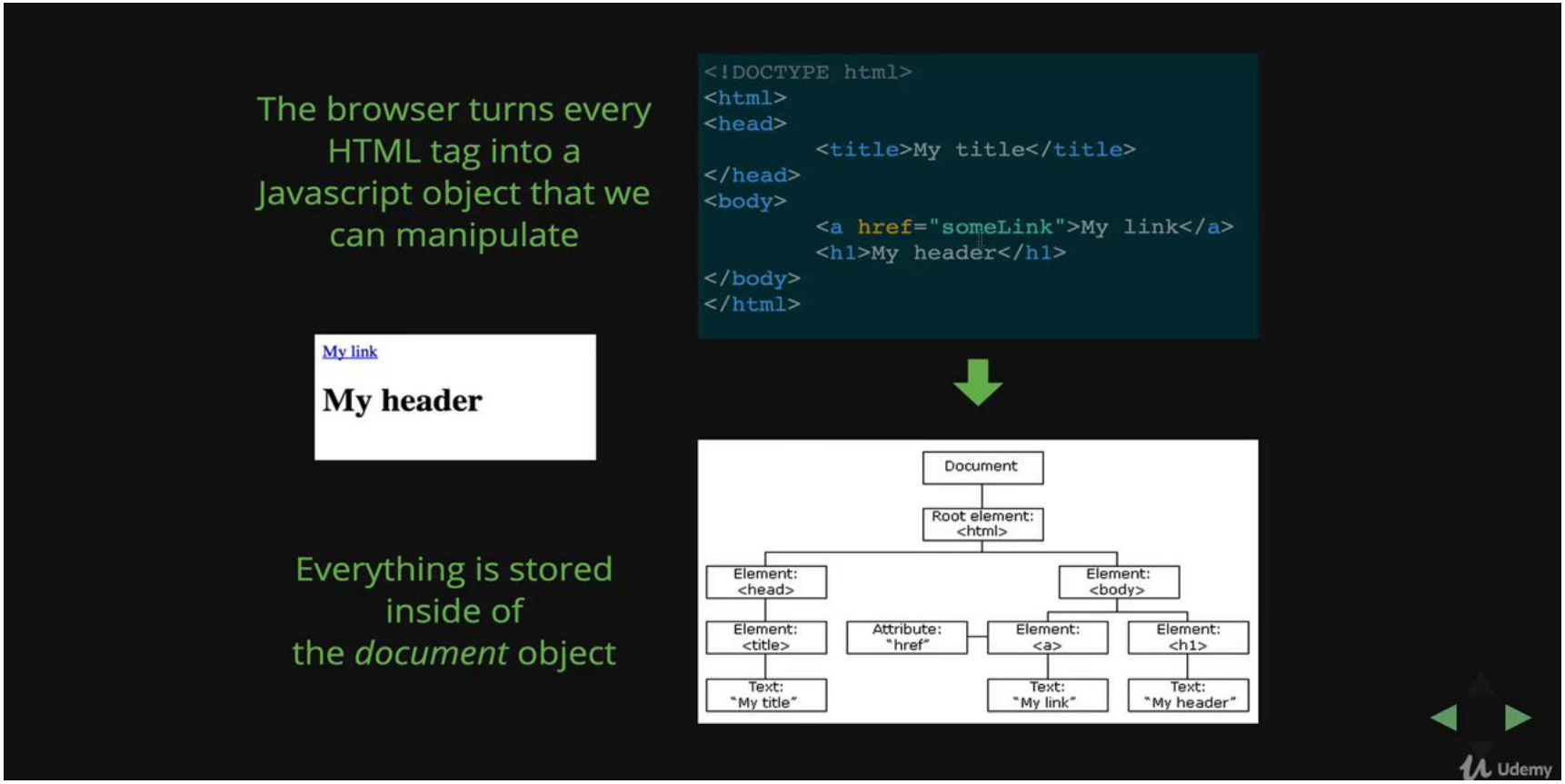
The Document Object Model

- DOM : Document Object Model
- Structured representation of an HTML Document
- javascript can be used to read/write/manipulate the DOM
- For Each HTML box, there is an object in the DOM that we can access and interact with
- Object Oriented Representation





The Document Object Model (Continued..)



Can view : dir(document)

- document.URL/ head/body/links .....etc

What can We Do With DOM

- There are various things we can achieve with DOM
  - Form Validation
  - Dropdown Menus
  - Interactivity and Animation : LInk
  - Game
  - Manipulate CSS

The Document Object Model (Continued..)

The process takes two phases :

1. **Select an element**
   a. **E.g var h1 = document.querySelector("h1") ;**
2. **Manipulate an element :** Style , content , add/remove/update node , event
   a. **E.g h1.style.color = "blue" ;**

E.g 2

```
var body = document.querySelector("body");
var isRed = false ;

setInterval(function()
{
if(isRed)   body.style.background = "white";

else body.style.background = "red";

}
isRed = ! isRed;
,1000);
```

Other Selectores in DOM

There are various methods to select an element in the DOM Tree :

- All returns are an object [check with **dir**)
  - **Single Element**
    - document.getElementById()
    - document.querySelector() : returns the first element that matches a given CSS-style selector
  - **Multiple Elements**
    - document.getElementsByClass() : return an HTML Collection
    - document.getElementsByTagName() : return an HTML Collection
    - document.getElementsByName() : return a Node List
    - document.querySelectorAll() : return a Node List

Node Traversing

- With the HTML DOM, you can navigate the node tree using node relationships (Node Traversing)
- Node Traversing Selectores
  - **Individual :** document.forms[#] , document.all[#], ..etc
  - **Node Position:** document.first/lastElementChild , document.first/lastChild , document.childNodes, document.children
  - **Parent :** listItem.parentNode , listItem.parentElement
  - **Sibling :** listItem.next/previousSibling, listItem.next/previousElementSibling

Manipulation of DOM

- Once an element is selected , the next step is manipulating the element
- There are various manipulations :
  - Changing an element style
    - style object : color , border , fontSize, margin , .etc
  - Adding/Removing class
    - classList object : add , remove , replace,toggle
  - Changing the content of the tag
    - textContent, innerText, and innerHTML property
  - Changing the attributes (src, href , etc)
    - getAttributes, setAttributes

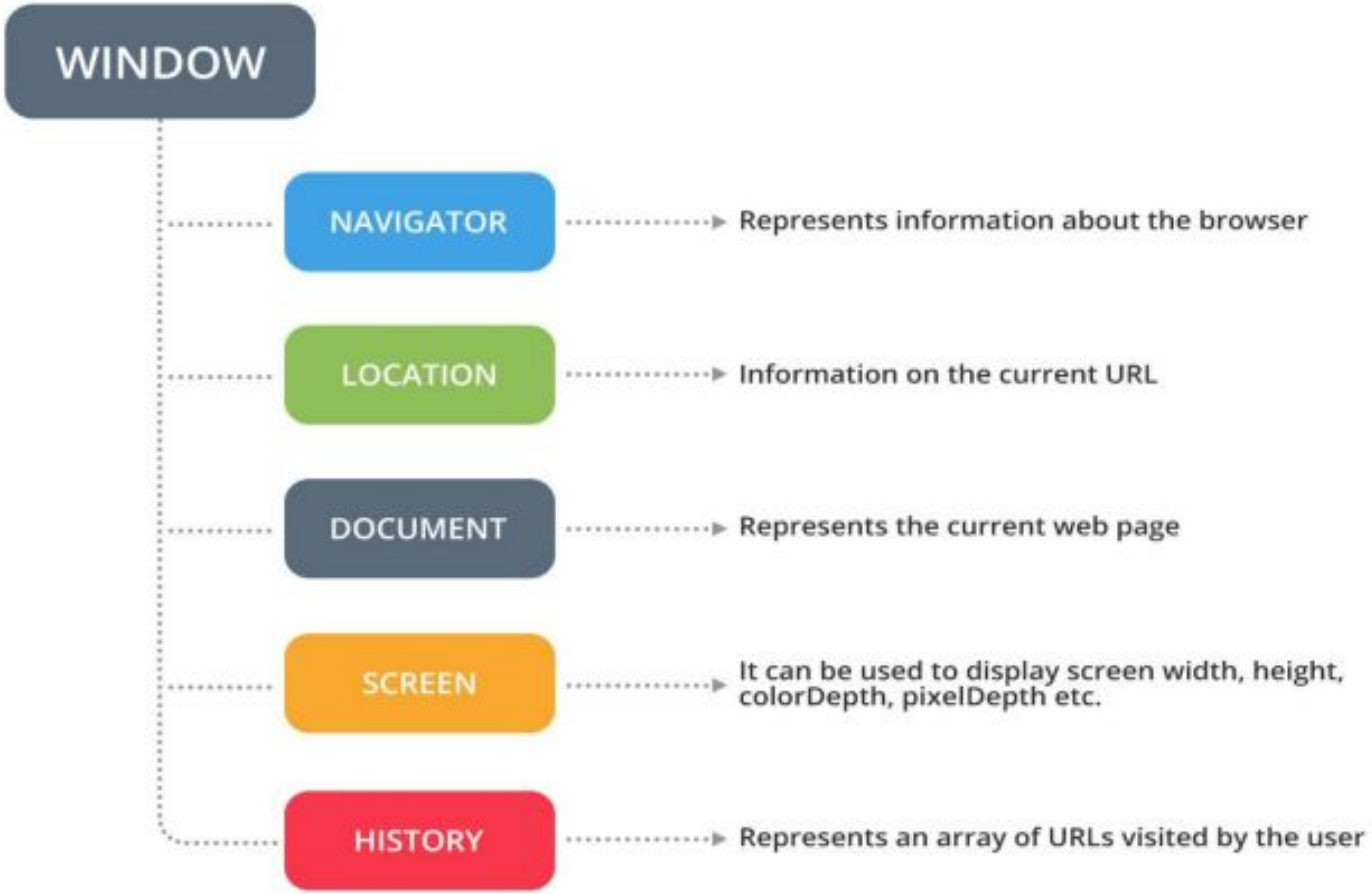
Browser Object Model

- **The Browser Object Model (BOM)** is used to interact with the browser.
- These BOM provides access to the various characteristics of a browser (Browser window itself, Screen characteristics, Browser history etc...)
- The default object of browser is **window** means we can call all the functions of window by specifying window or directly
- If a document contain frames (<iframe> tags), the browser creates one window
- Object for the HTML document, and one additional window object for each frame.

Example :

```
window.alert("Hello World");
```

BOM – Object Hierarchy



BOM – Object Hierarchy

- **BOM – Navigator Object :** appCodeName, appName, appVersion , userAgent , language, cookieEnabled, platform
- **BOM – History Object :** back(), forward(), go(#), length
- **BOM – Location Object :** window.location.href , window.location.hostname ,window.location.pathname, window.location.protocol, window.location.hash , window.location.port , window.location.assign() , window.location.reload() ,window.location.replace()
- **BOM – Window Size :** window.innerHeight , window.innerWidth , availHeight(), availWidth() , height(), width()
- **BOM – Window Methods** : window.alert(), window.confirm() ,window.prompt() ,window.close() ,window.moveTo() , window.resizeTo()
- **BOM Timing Events :**
  - setTimeout(function,duration) , setInterval(function,duration) , clearTimeout(timeout)

Reading Assignment

1. Is it a good idea to manipulate the style of an element with javascript ? why ?
2. Open and change the interface of Google with the console for temporary basis
3. Differences b/n DOM and BOM ?
4. Read in Depth about HTML Collection and Node List

ITSE-2192 Fundamental of web Design and Development

Advanced Javascript

Lecture Six

Outline

- Working with Data types
- Array in Depth
- Working with Function
- Object Oriented Javascript
- Exception in Javascript
- ES6 Modules

Data types in ES6

Symbol : Symbols are new primitive type introduced in ES6

- Symbols are completely unique identifiers.
- **Syntax: Symbol('debugging string')**
  - E.g const id = Symbol("id")
- Every time you invoke `Symbol()` we get a new and unique symbol
  - Symbol() === Symbol() //false
- Symbols are often used to identify object properties

```
let user = {   [id]: 123  };
```

- String Concatenation with template literal
  - **Syntax** : **`${data}`**
    - E.g let name = "Jone" ; console.log(`Hello ${name}`);

For Loop(Newer flavors)

There have been to types of for loops introduced in ES6

- **for of :** a loop iterating over iterable objects, including: built-in String, Array, array-like objects
  - **E.g** const iterable = [10, 20, 30];

```
for (const value of iterable) {
    console.log(value);
}
```

- **for in :** iterates over all enumerable properties of an object that are keyed by strings (ignoring ones keyed by Symbols)
  - **E.g** const **object** = { a: 1, b: 2, c: 3 };

```
for (const property in object) {
  console.log(`${property}: ${object[property]}`);
}
```

Arrays and Objects

- **Destructuring** : the act of decomposing/unpacking an iterable object to a list of variables
  - **Array :** [var1,var2] = arrayObject
  - **Object:** {var1,var2} = object;
    - The name of the object properties must be the same as the variables names
    - To use different variable name : Syntax originalName : newName
    - We can have default value incase the object does not have the property

E.g

```
//destructuring array

const arr = ['John',25];

const [name, age]  = arr;

console.log(name);
console.log(age);
```

E.g

```
//destructuring object

const obj = {name: "John", age:22}

const {name, age: ag =2}  =obj ;

console.log(name);
console.log(age);
```

Arrays and Objects

- **Spread Syntax** : used to take an array and spread to list of variables
  - **Syntax** : **...array**
  - It can be applied also on **Node List**

E.g

```
//spread syntax on function

function summation(a,b,c,d,e)
{
   return a + b + c + d + e;
}

const arr = [1,2,3,4,5];

let sum = summation(...arr);

console.log(sum);
```

E.g

```
//spread syntax on array merge

const arr1 = [1,2,3,4];
const arr2 = [5,,6,7,8];

const merg = [...arr1, 'additional',
...arr2];

console.log(merg);
```

Working with Functions

- There is new way of writing simple IIFE in ES6 for a block scope data
  - **Syntax : { let a = 1 ; const b = 2 }**
    - The data is not accessible from outside this scope
- **Arrow Functions :** new way of declaring functions that do not get their own this keyword
  - **Syntax : (parm1,parm2..) => { };**
  - **Note :**
    - We can leave the parenthesis if there is only one parameters
    - If there is no parameters we can use underscore(_) instead of parenthesis
    - We can leave the curly brace and return statement if the the body has one line
    - when returning object we have to use enclosing parentheses
      - **E.g** const message = () => ({message : 'Hey There '});

Working with Functions

- What does it mean by Arrow Functions do not get their own **this**

```
//ES5 Solution
let boxDiv = document.querySelector('#cl') ;
let box = {
    color : 'Red',
    clickMe: function ()
   {
       var self = this;
    boxDiv.addEventListener('click', function()
    {
       alert('This box is ' + self.color );

    }
 ); }
 }

 box.clickMe();
```

```
//ES6 Solution
let boxDiv = document.querySelector('#cl') ;
let box = {
    color : 'Red',
    clickMe: function ()
   {

   boxDiv.addEventListener('click', ()=>
    {
       alert('This box is ' + this.color );
    }
 ); }

 }

 box.clickMe();
```

Working with Functions (Continued..)

- **Rest Parameters :** Allows to store multiple arguments in a single array
  - **Syntax : functionName(p1,p2 , ...varName)**
  - This parameter must only come at last if there are other parameters

E.g

```
function sendCars(p1,p2, ...nList)
{
   let total = 0 ;
   nList.forEach( nu => {  total +=nu ;  });
   console.log(total);
 }
sendCars(1,2,3,4,5,6);    //Result ???
```

Working with Functions (Continued..)

- There are various ways to call a function apart from the normal function invocation : **call , apply and bind**
- **call and apply:** mainly used to change the value of **this**
  - change the context of the function based on the object it is called

call : takes list of arguments	**apply : takes array argument**
let oldCar = { carId : 123, **getId** : function (pref) { console.log(pref + this.carId) ; } }; let newCar = {carId : 456}; **//changing the context of the function** oldCar.**getId**.**call**(newCar, “My Id : ”);	let oldCar = { carId : 123, **getId** : function (pref) { console.log(pref + this.carId); } }; let newCar = {carId : 456}; **//changing the context of the function** oldCar.**getId**.**apply**(newCar, ['Id : ']);

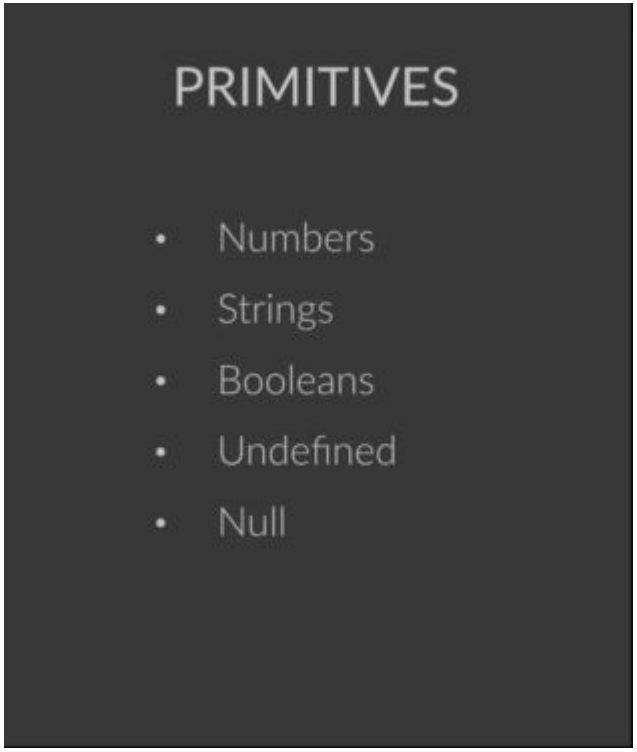
Working with Functions (Continued..)

bind : create a copy of the function and assigning new object context

```
let oldCar = {

   carId : 123,
   getId : function ()
   {
      console.log(this.carId);
   }
};

let newCar = {carId : 456};

 //copy the fun and changing the context of the function
let newFun = oldCar.getId.bind(newCar);

newFun()
```

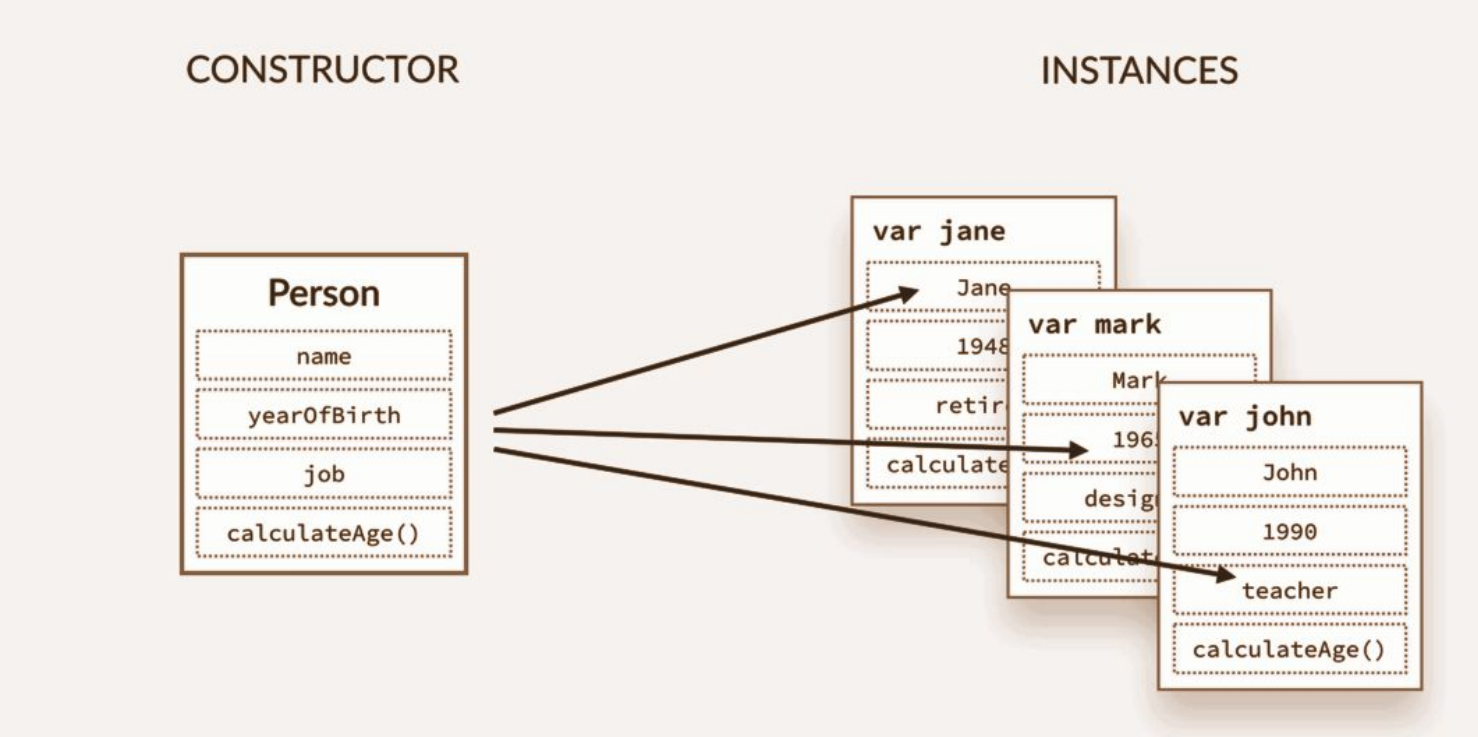
Object Oriented Javascript

- Everything in is an object (well, almost everything)



Object Creation in Javascript

- The most common way to create objects is using **object literal** , but with this method we can only create a single object
- To create class/blueprint for object we can use three techniques : **constructor function** , **object.create method** and **ES6 class**



Object Creation in Javascript (Continued..)

- **Constructor function**: use Function expression or regular function definition [**Method 1**]

Syntax

```
function NameFun(par1, par2, ..)
{

 this.par1 = par1 ;
 this.par2 = part2;
 ....

 this.funExp : function() {};
}

var ob = new NameFun();
```

E.g

```
function Person (name, job, yearofB)
{

this.name = name ;
this.job = job;
this.yearofB = yearofB;

this.calcAge = function(){ return new Date().getFullYear() - this.yearofB};


}

let jon = new Person("Jone", "Teaching", 1990);
console.log(jon);
console.log(jon.calcAge());
```

Object Creation in Javascript (Continued..)

- **Object.create**: we can use Standard object as a template[**Method 2**]

Syntax

var obj = Object.create(templet);

E.g

```
const personObject = {
    calcAge: function(){ return new Date().getFullYear() -
this.yearofB; } };

let jon = Object.create(personObject);
jon.name = "Jhone";
jon.yearofB = 1990;
jon.job = "Teaching";
// The other way
let jane = Object.create(personObject,
 {
  name : {value : "Jane"} ,
  yearofB: {value : 1992},
  job: {value : "Designer"}
 }
);

console.log(jon);
console.log(jane);
```

Object Creation in Javascript (Continued..)

- To use **ES6** way of using the **class** keyword[**Method 3**]

Syntax

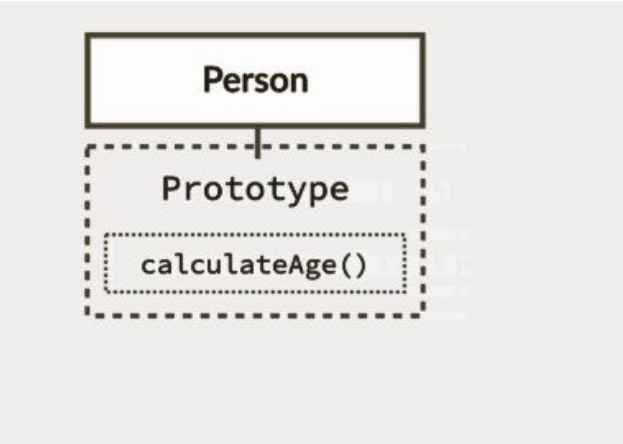
```
class ClassNAme
{
  constracture(pro1,pro2, ..)
  {
     this.pro1 = pro1 ;
     ...
  }

functionName()
{
…..

}
}
```

E.g

```
class Cat {

constracture(name, color)
{
   this.name = name;
   this.name = color;
}
speak() {
   console.log("Meewoo");
}

}

const cat  = new Cat('fluee', 'white')
console.log(cat);
cat.speak();
```

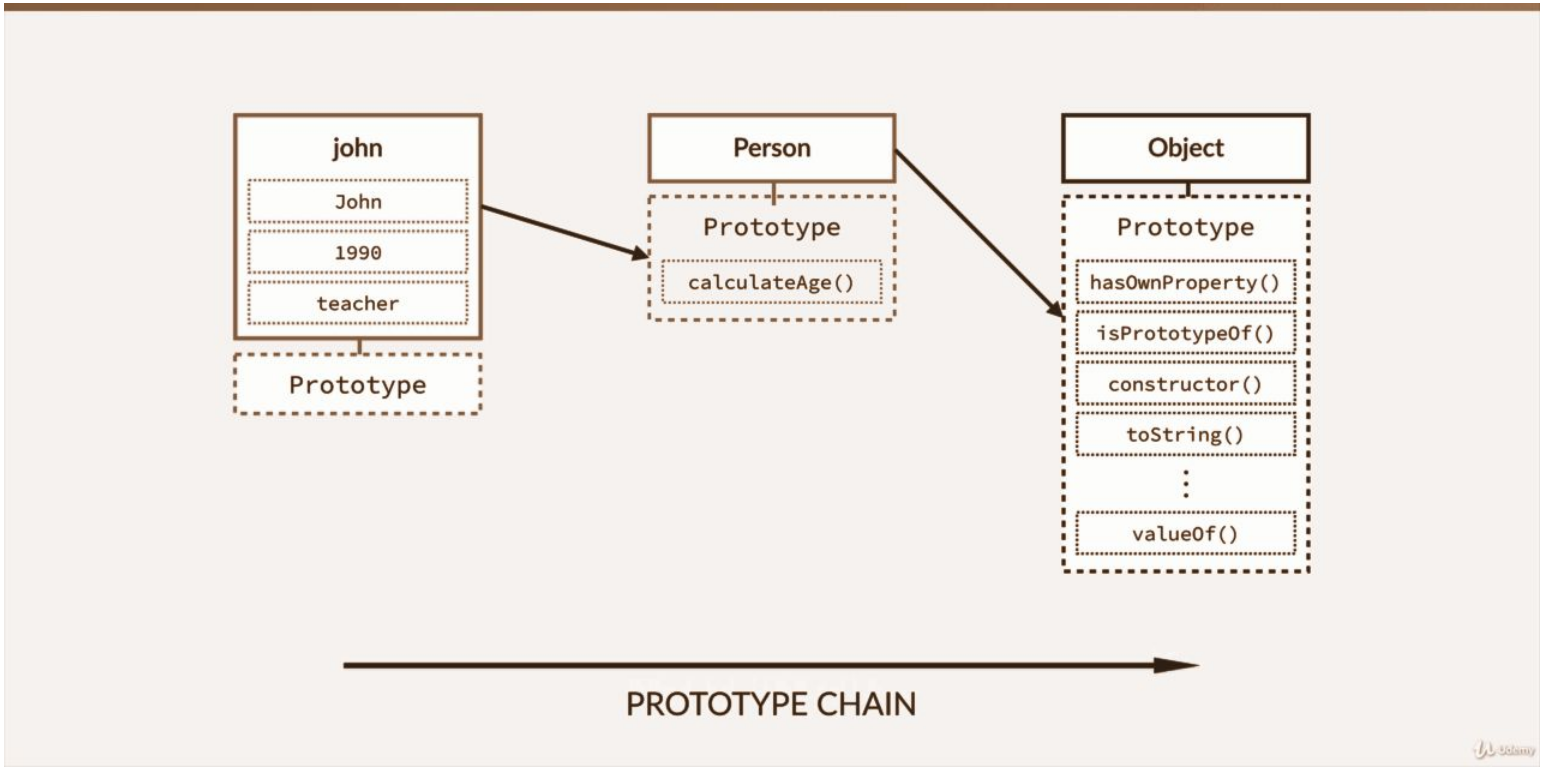
Inheritance in JavaScript

- Javascript is prototype based language
- Every Javascript object has a property **prototype** , which makes inheritance possible (**__proto__**)
  - **Functions Prototype :** is the objects instance that will become the prototype for all objects created using this function as a constructor
    - It is where we put methods and properties that we want to other objects to inherit
  - **Objects Prototype: i**s the objects instance from which the object is inherited



Inheritance in JavaScript (Continued..)

- When a certain method/property is called , the search starts at the object itself , and if it can not found , the search moves on to the object's prototype .
  - This continues until the method is found: prototype chain



Inheritance in JavaScript (Continued..)

- Constructor Function Prototype : Method 1 for creating inheritance

Syntax

ObjectName.prototype.PropName = Pro/Method

E.g

```
function Person(firstName , lastName)
{
  this.firstName = firstName;
  this.lastName = lastName;
}

Person.prototype.getFullName = function(){
      console.log(`${this.firstName}  ${this.lastName}`)
   }

const p1 = new Person('Merry', 'Jhonsen') ;

pl.getFullName();
```

Inheritance in JavaScript (Continued..)

- Constructor Function Prototype : Method 1 for creating multiple inheritance

```
function Customer(firstName,lastName,membership)
{

   Person.call(this,firstName,lastName); //constructor

   this.membership = membership;

}

Customer.prototype = Object.create(Person.prototype);
Customer.prototype.constructor = Customer;//change constructor


const c1 = new Customer('Jhone', 'Kally', 'PRO');

c1.getFullName();
```

How to override ??

Inheritance in JavaScript (Continued..)

- ES6 class extends : Method 2 for creating inheritance

Syntax

childClass extends ParentClass

You can use to view info about the object

👇👇

console.info(objectName)

E.g

```
class Person
{
        constructor(fName, lName)
        {
            this.fName = fName;
            this.lName = lName;
        }
        greeting()
        {
           return `Hello ${this.fName} ${this.lName}`;
        }
}
class Customer extends Person
{
        constructor(fName, lName)
        {
            super(fName, lName);
        }
}
```

Exception in Javascript

- Exception : are objects that stops the normal flow of a program
- How to Handle error anticepted code : try/catch ..finally

Syntax : **try { // error expected code } catch(ErrorType) { handler code }**

Example

```
try
{
   let car = newCar;
}
catch(error)
{
  console.log('Error : ' + error)
}
finally
{
   console.log('This always execute ...');
 }
console.log('continuing the execution ….')
```

ES6 Modules

- To make objects, functions, classes or variables available to the outside world it's as simple as exporting them and then importing them where needed in other files.
  - Syntax **export NameOfElement [variable , function , class]**
  - **Import Element from 'fileName.js'**

```
export const myNumbers = [1, 2, 3, 4];

export function myLogger() {
  console.log(myNumbers, animals);
}

export class Alligator {
   constructor() {
     // ...
   }
}

or  [As a Last LIne]
export { myNumbers, myLogger, Alligator };

Alias
export { myLogger as Logger }
```

```
/
import { myLogger, Alligator } from 'app.js';

//with alias
import myLogger as Logger from 'app.js';

//Importing all exported members
import * as Utils from 'app.js';


Use : access to members with the dot notation

 Utils.myLogger();
```

Reading Assignment (Bonus)

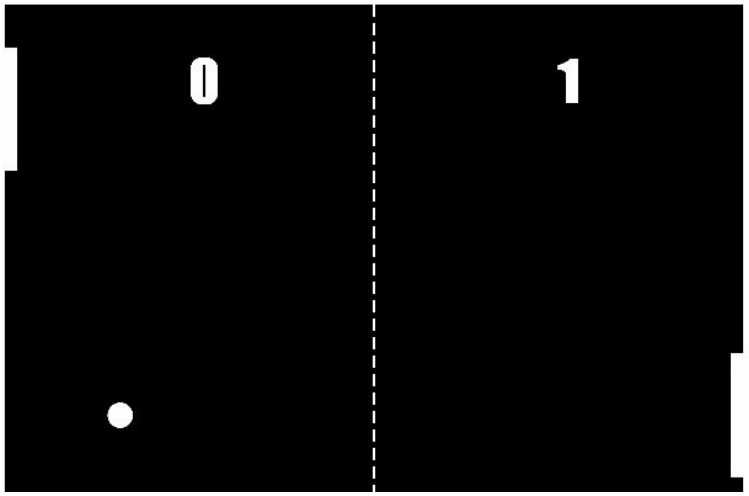
1. What are Tagged Templates?
2. Explain and demonstrate first class function
    a. passing functions arguments
    b. functions return functions
3. Explain and demo closures
4. Demonstrate the Reflect API
    a. In construction and method call , in prototypes and Reflect and properties
5. What is Export Default ? Explain with examples
6. What are iterators and generators ? Explain with example
7. What are Sets and Maps in JS ? Explain with example

⇒ Demo with simple project [that incorporates all the concepts]

Bonus Challenge (Game Development)

1. Follow this tutorial to develop 2D breakout Game using Canvas : LInk
2. Add the following features [Update the Game]
    a. Make the game to have two players [User , Computer]
        i. Add Computer as Second User [Simple AI]
    b. Improve the UI
        i. Use CSS Library
        ii. Add Help Menu
        iii. Make the Two Vertical paddles [One in the Left for user , one on the right Computer]
    c. Persisite User Score using local storage [Load Score on Game start , Update Score]
    d. Write the code with an OOP Approach [ES6 Class]

Sample Demo Example 👉👉👉



ITSE-2192 Fundamental of web Design and Development

Asynchronous Programing in Javascript

Lecture Seven

Outline

- JavaScript Definition
- Asynchronous Programing
- Event Loop
- Callback Basics
- From callback to Promise
- Async and await
- Ajax and API Overview
- XmlHttpRequest(XHR) object
- Fetch API

Javascript

What is Javascript ?

“ A Single-Threaded , Non-Blocking , asynchronous , and concurrent language “

One Thread == One Call Stack == One thing at a time

Why Asynchronous ?

The choice for Async is :

- Allow a block of code to run in the background
- Move on Immediately : Non-Blocking

What is Blocking ??

- **Blocking Process :** Relatively Long Process like Network Request , Complex Operation

Note : Asynchronous code is handled by what is called the **event loop**

Handling Async Code

There are few ways to work with Async code in Js :

- Callbacks
- Promises
- Async/Await

Most Async code you work with will be part of an API or Library

- XMLHttpRequest and Fetch (ES6)
- jQuery Ajax ,Axios , other Http Libraries
- Node Js fs Module

ASYNC VS SYNC (Example)

```
let second = ()=> {
console.log(“How are u doing ? ”)
}

let first = () => {
console.log(“Hey there ! ”);

second()

console.log(“The end ”);

}
```

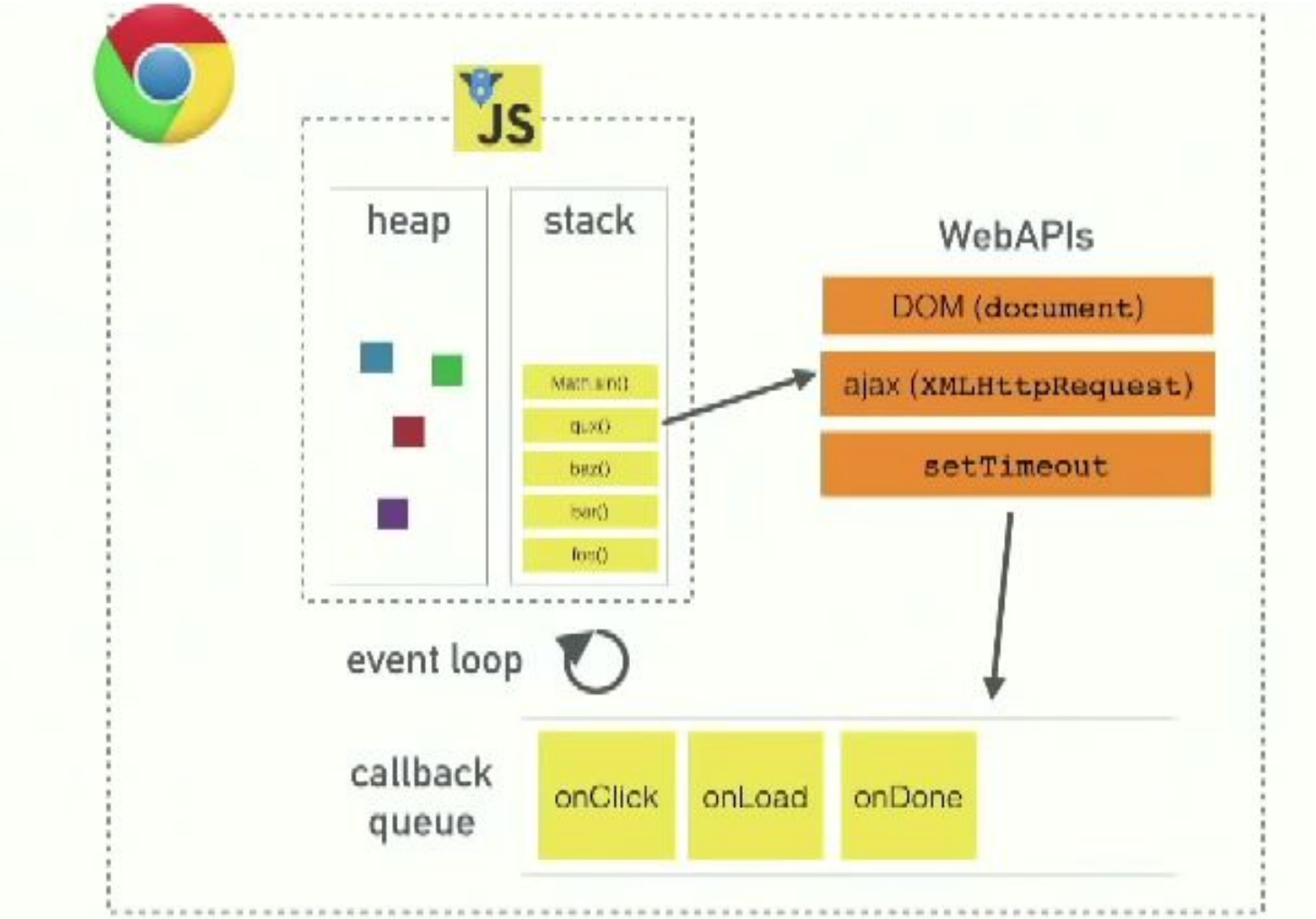
Sync

```
let second = ()=> {

    setTimeout(()=>{
      console.log(“Async hey There!”);
} , 2000);

}

let first = () => {

  console.log(“Hey there ! ”);
  second()
  console.log(“The end ”);
}
```

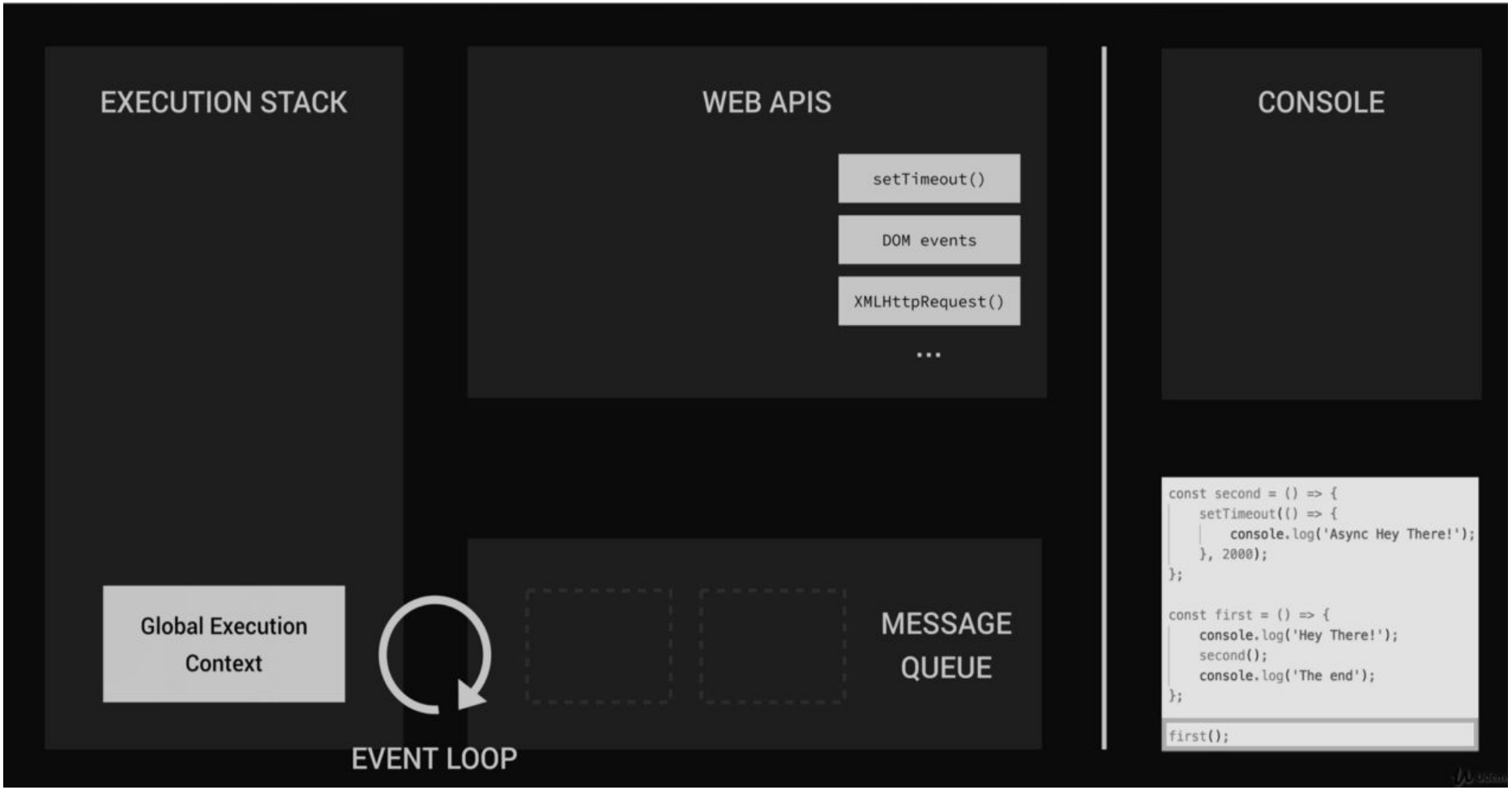
ASync

Javascript Host Environment

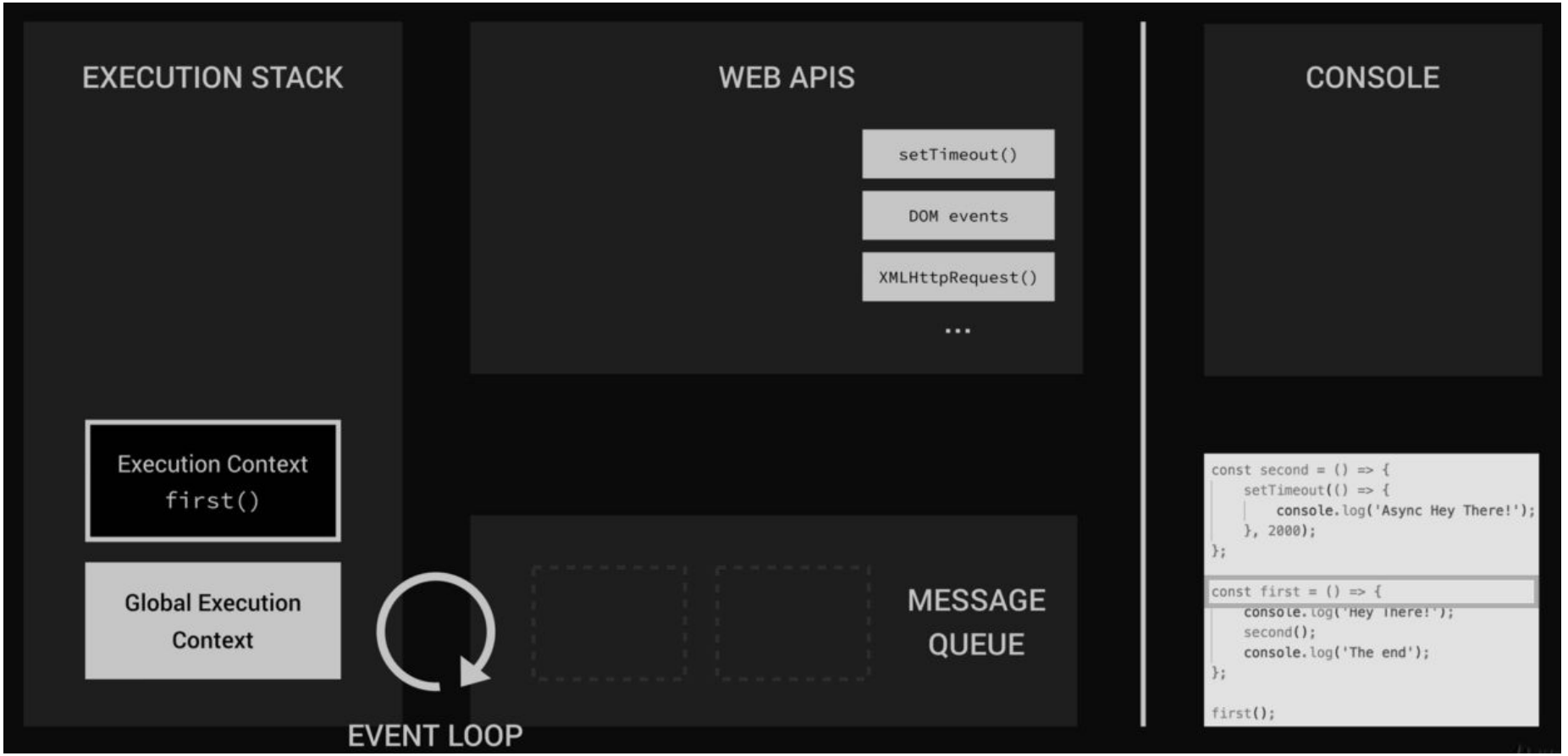
Visualizer : Link , Link 2



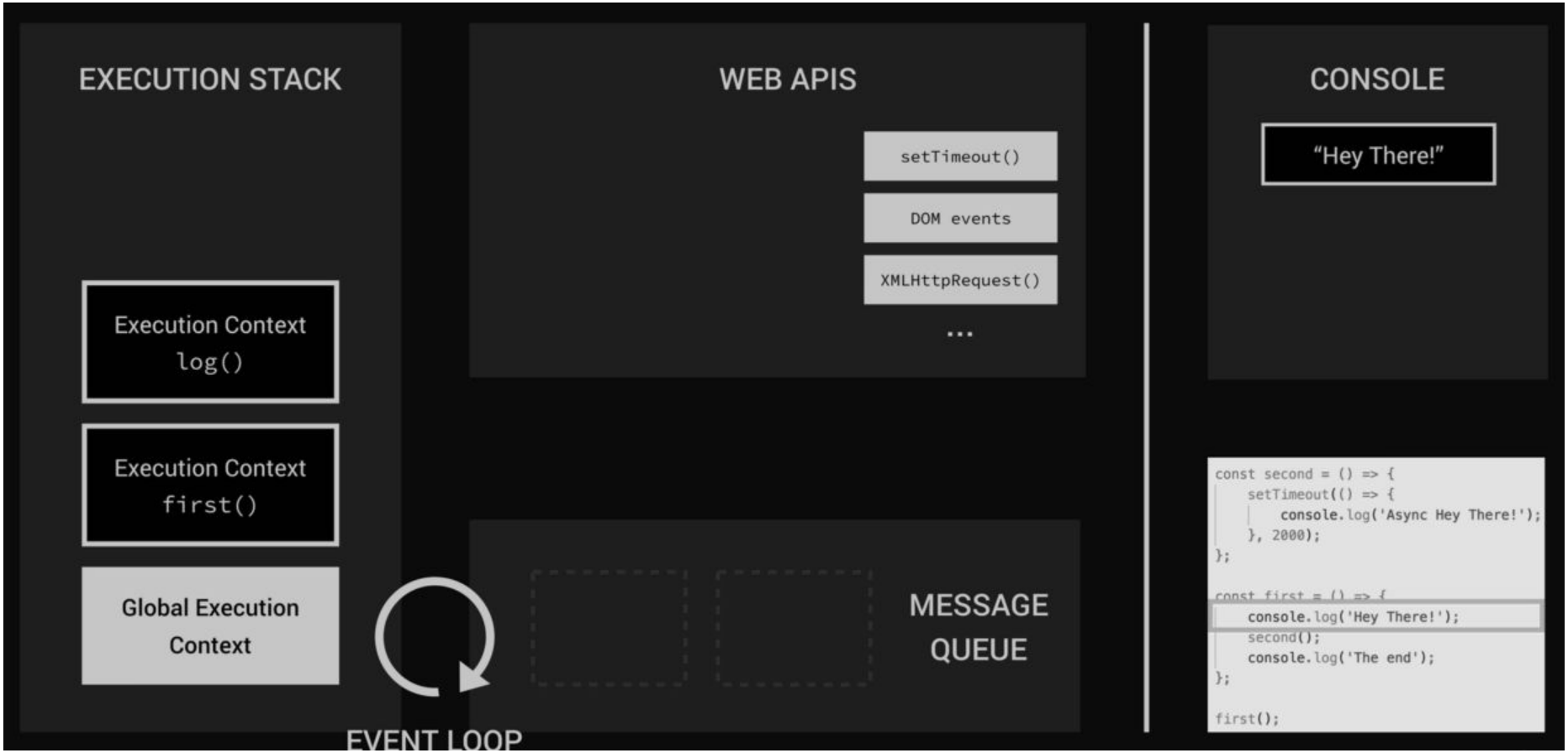
The Event Loop



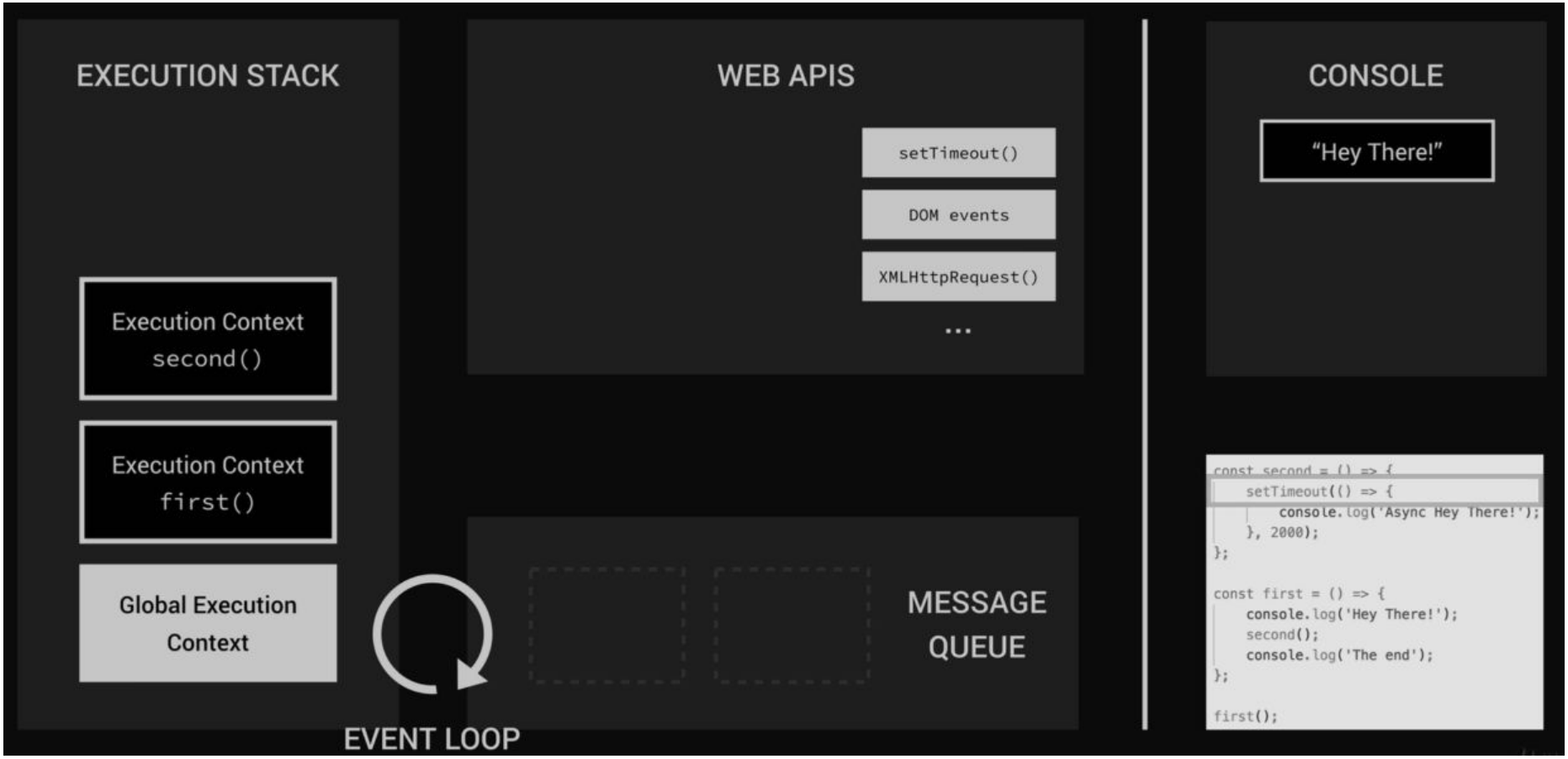
The Event Loop(ctnd...)



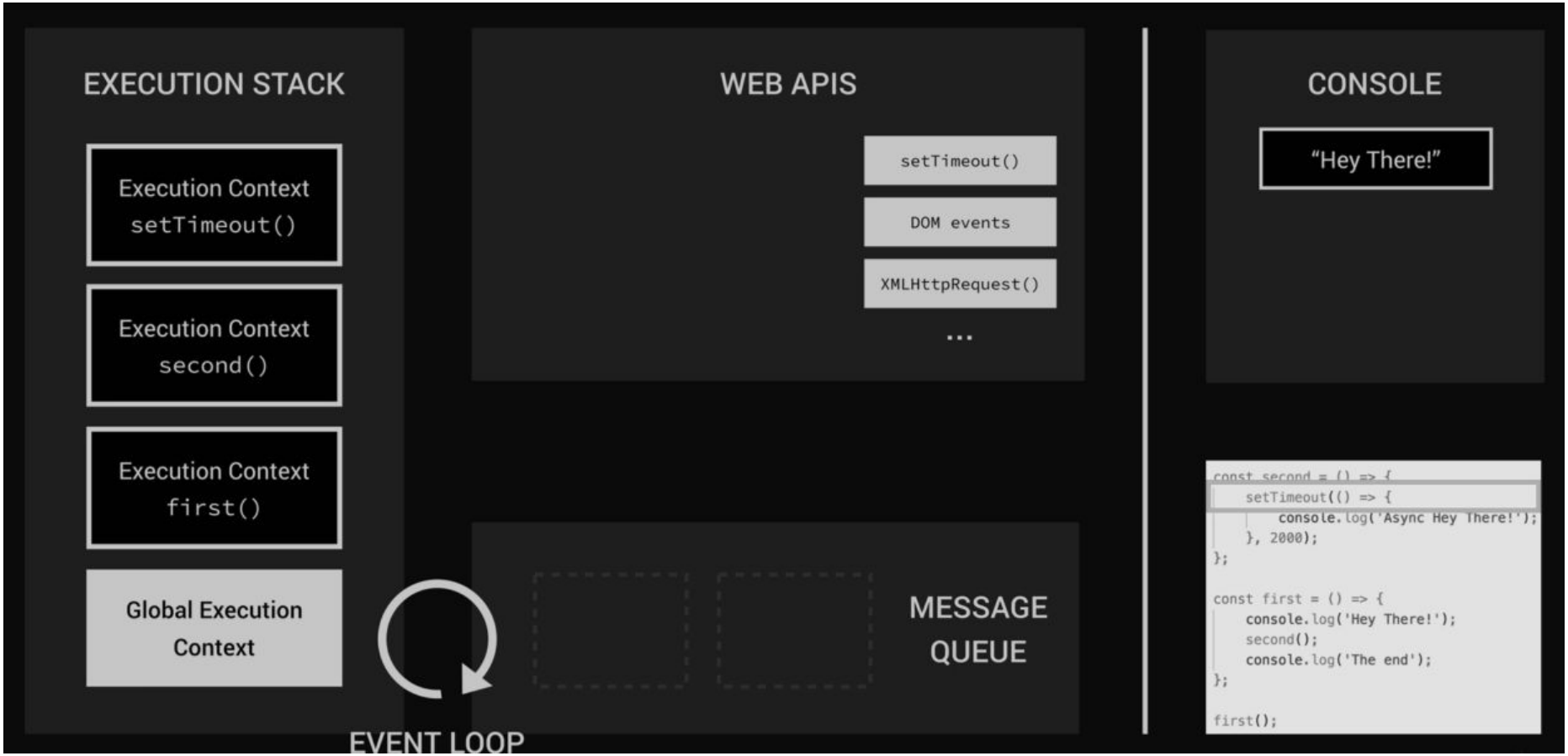
The Event Loop(ctnd...)



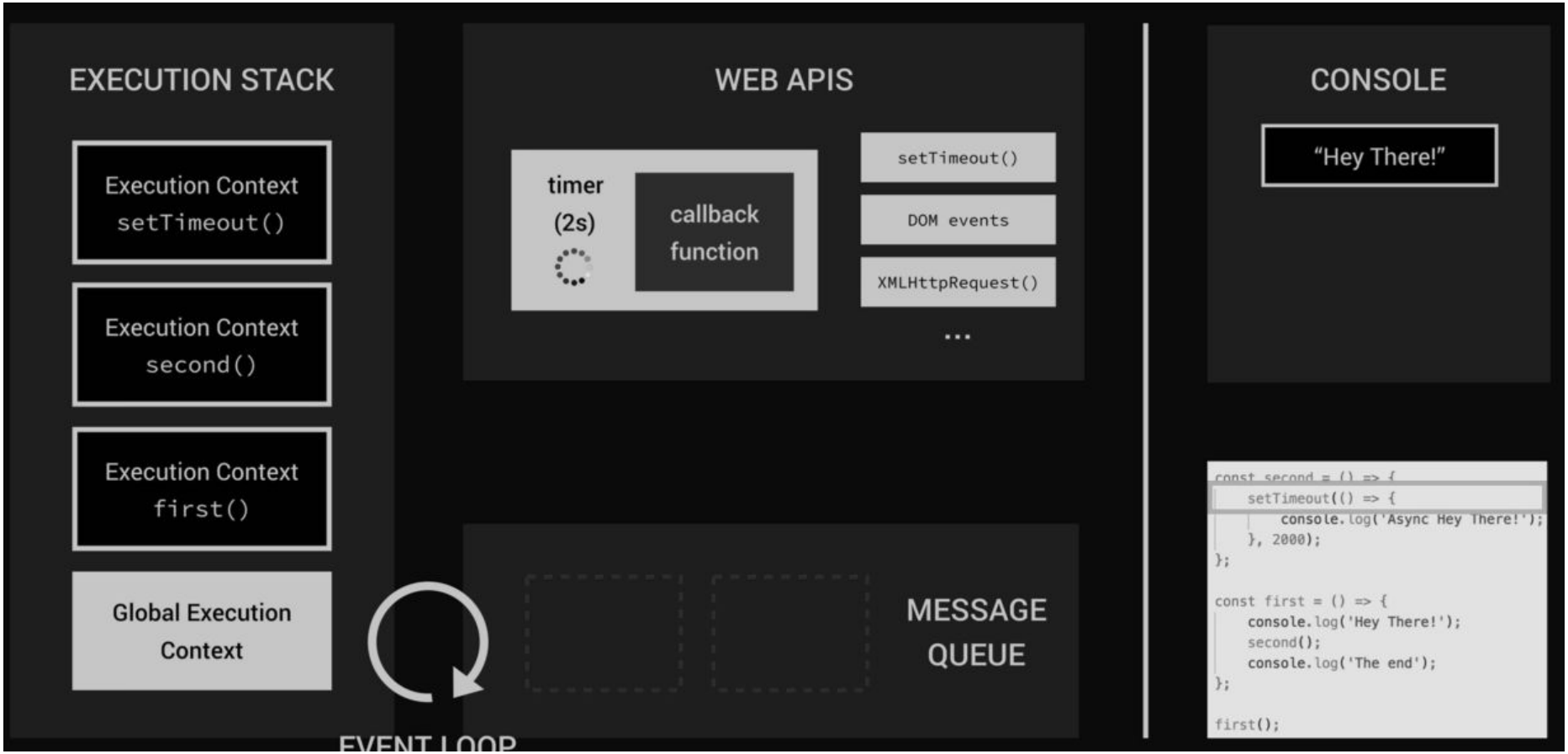
The Event Loop(ctnd...)



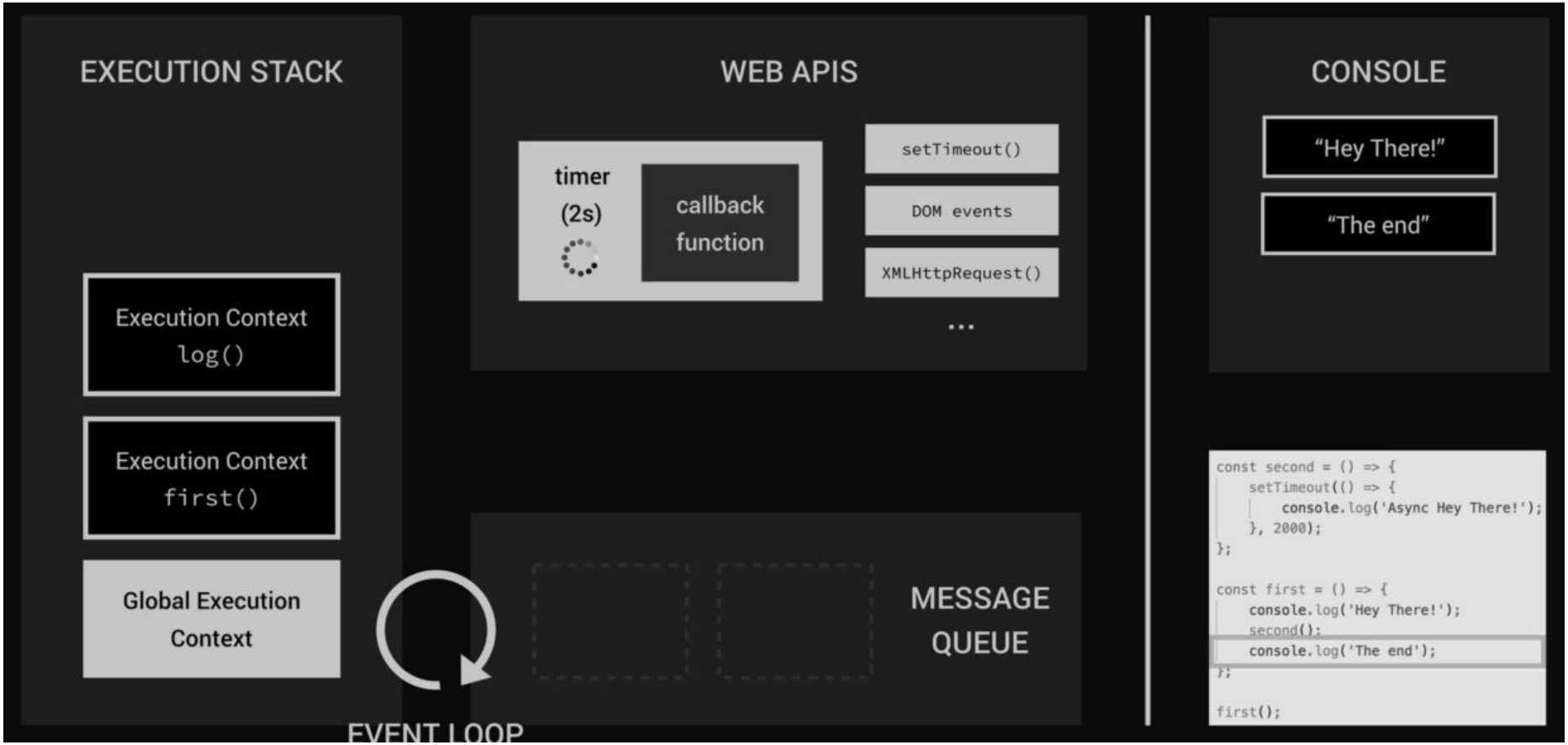
The Event Loop(ctnd...)



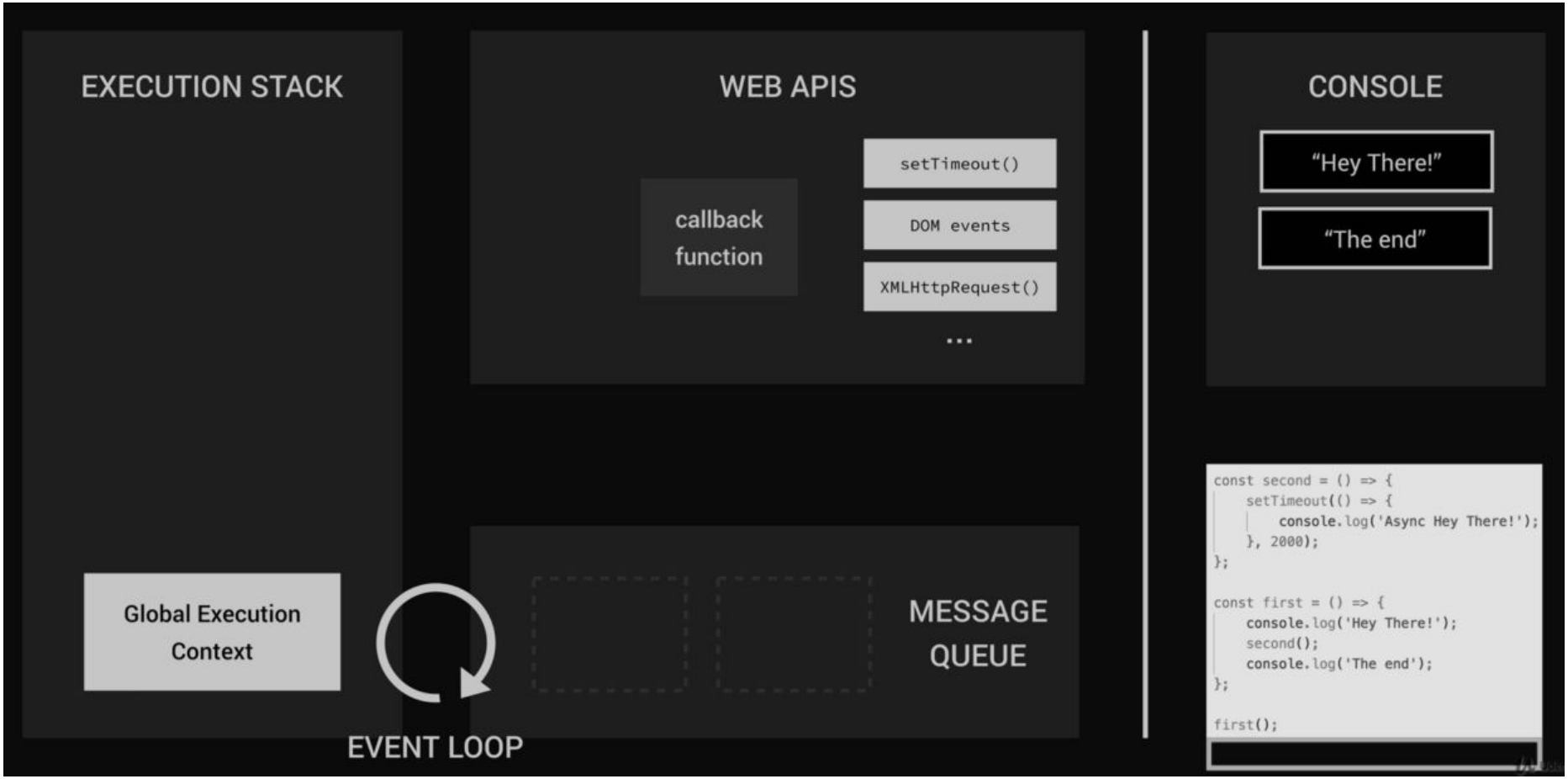
The Event Loop(ctnd...)



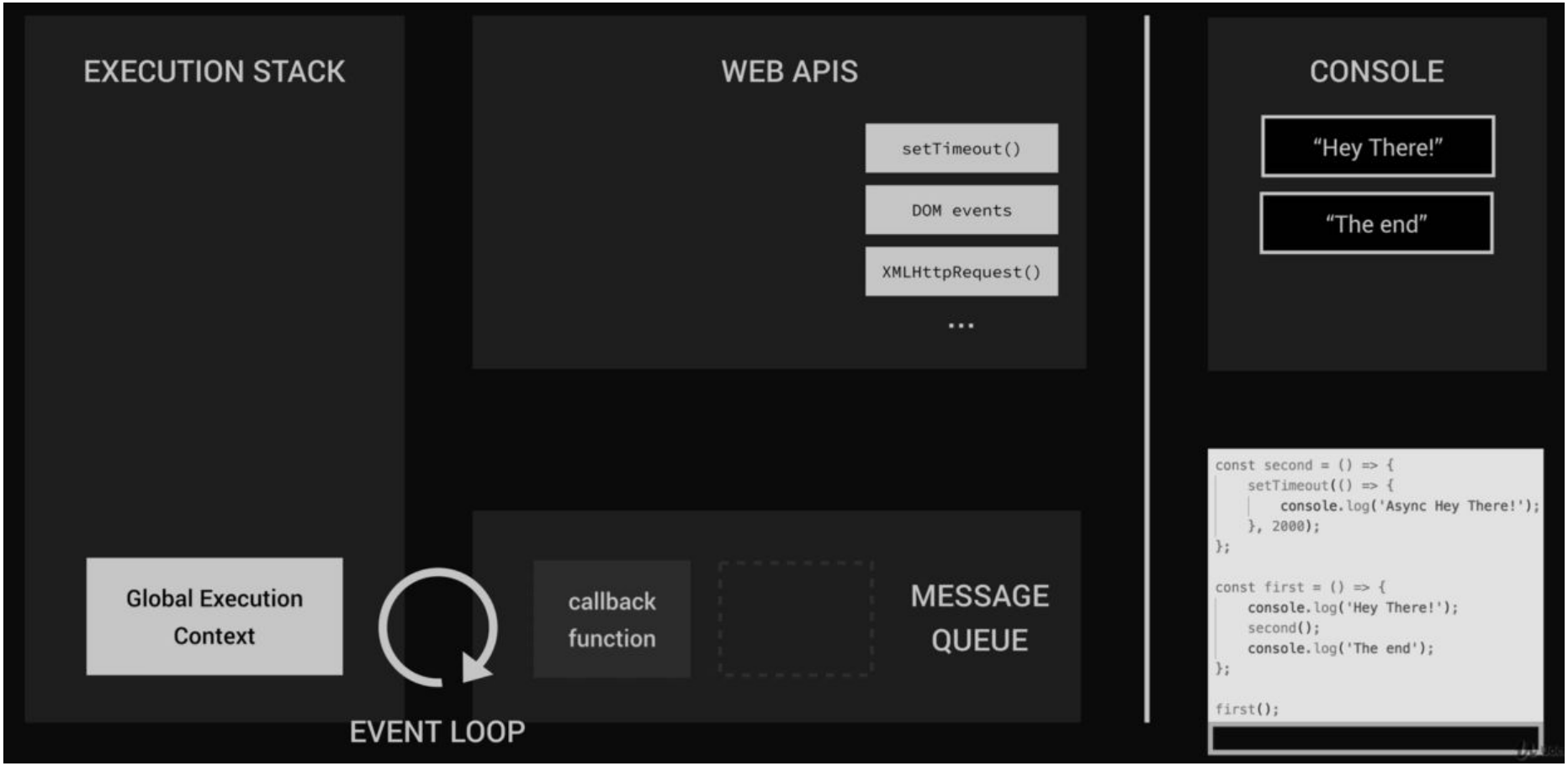
The Event Loop(ctnd...)



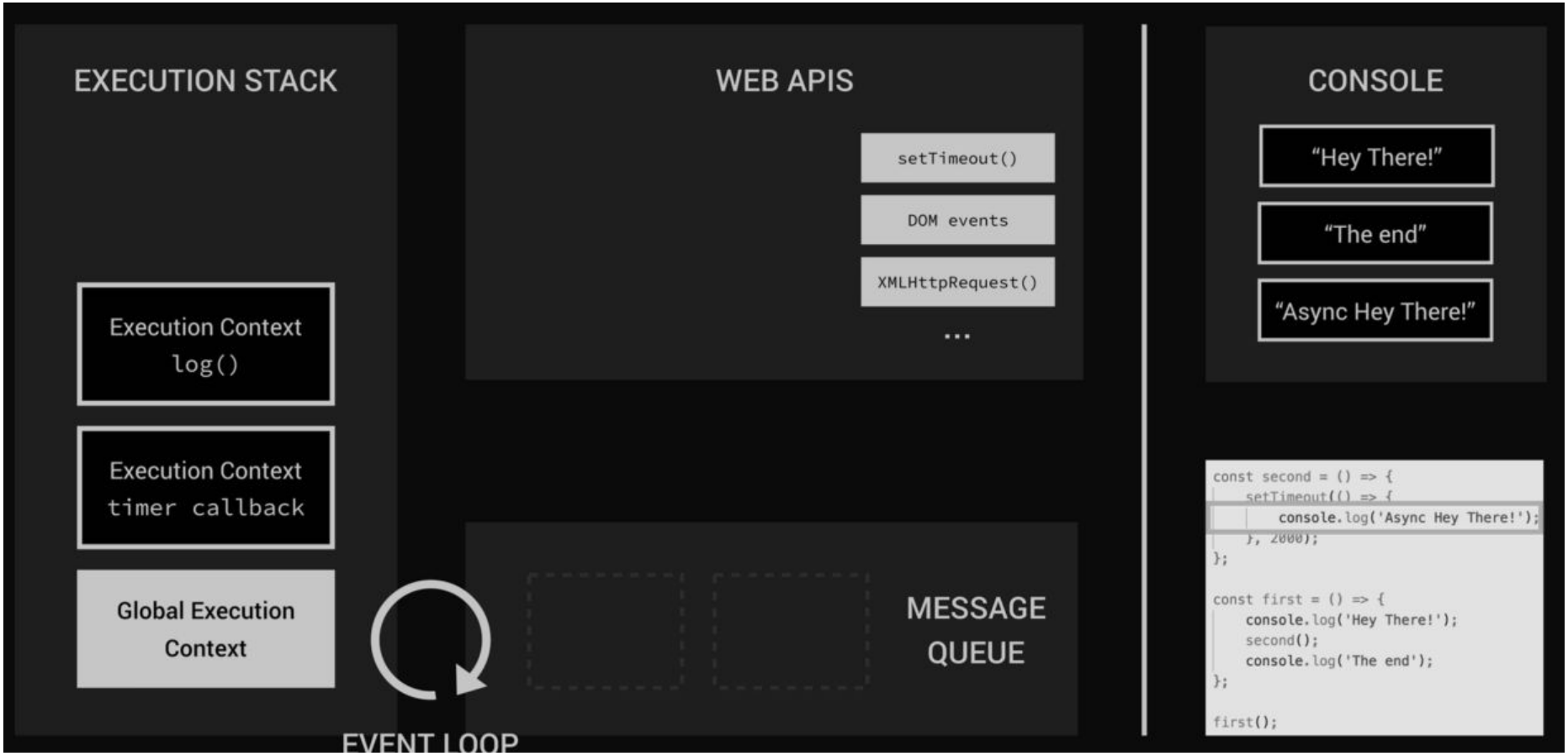
The Event Loop(ctnd...)



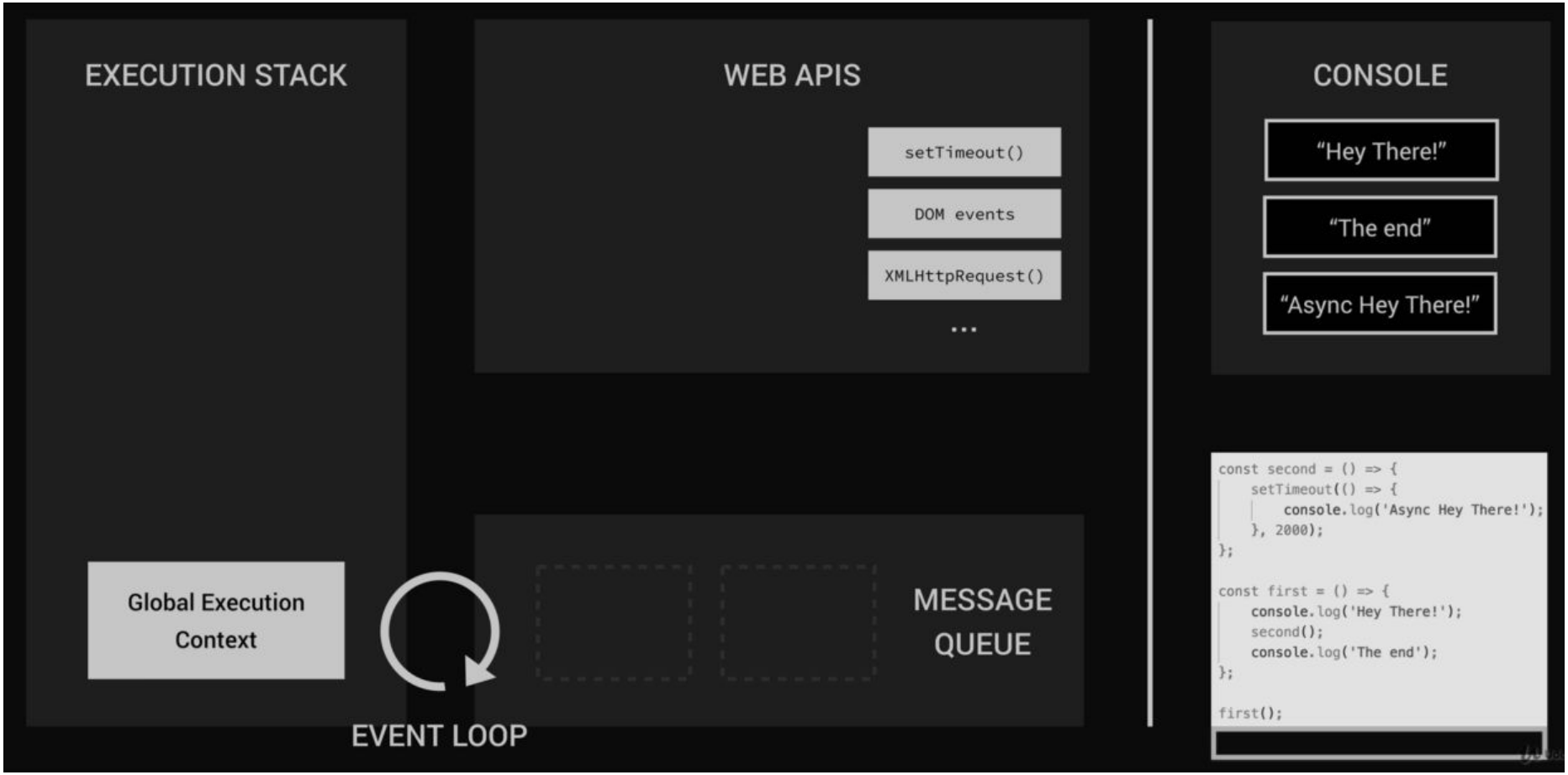
The Event Loop(ctnd...)



The Event Loop(ctnd...)



The Event Loop(ctnd...)



Callback

- A callback is a function passed as an argument to another function
- callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load)

```
function print(callback) {

    callback();

      }
```

- **Challenge :** Callback Hell
  - Solution : Promise , Async/Wait

Callback in Async

```
const message = function() {
console.log("shown after 3
sec");
}

setTimeout(message, 3000);
```

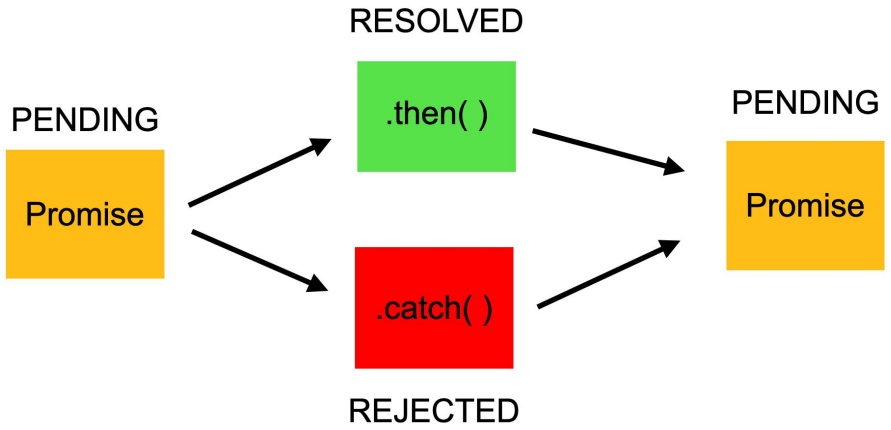
```
firstFunction(args, function() {
  secondFunction(args, function() {
    thirdFunction(args, function() {
      // And so on…
    });
  });

  });
```

Any Problem Here ???

Promise

- A Promise is a JavaScript object that links producing code and consuming code
- Promises are one of the ways we can deal with asynchronous operations in JavaScript
- Still use callback functions with Promises, but in a different way (chaining)

"I Promise a Result!"



Note : There can be only a single result or an error

Promise(ctnd...)

```
const myPromise = new Promise((resolve, reject) => {
    let condition;

    if(condition is met) {
        resolve('Promise is resolved');
    } else {
        reject('Promise is rejected');
    }

  });
```

producing code

```
myPromise.then((message) => {
    console.log(message);
}).catch((message) => {
    console.log(message);

  });
```

consuming code

Promise Chaining

```
firstRequest()
  .then(function(response) {
    return secondRequest(response);
}).then(function(nextResponse) {
    return thirdRequest(nextResponse);
}).then(function(finalResponse) {
    console.log('Final response: ' + finalResponse);

  }).catch(failureCallback);
```

Async/Await

- Async and Await make promises easier to write
  - async makes a function return a Promise
  - await makes a function wait for a Promise
- They make async code look more like old-school synchronous code
- Mainly used in API Request like **fetch**

```
E.g  async function hello() { return "Hello" };

     hello();    //Returns  Promise
```

To actually consume the value returned when the promise fulfills, since it is returning a promise, we could use a .then() block:

```
hello().then((value) => console.log(value))
```

Async/Await(ctnd...)

- **The await keyword:** You can use await when calling any function that returns a Promise, including web API functions.
- The await keyword is only valid inside async functions
- The **purpose** of async/await is to simplify the syntax necessary to consume promise-based APIs

E.g

```
async function hello() {
    return greeting = await Promise.resolve("Hello");
};

hello().then(alert);
```

Ajax and API

- **Ajax** Stands for **Asynchronous Javascript and XML**
  - Set of Web Technologies
  - Send and Receive data asynchronously
  - Does not interface with the current page (No Page Reload)
  - JSON has replaced XML for the most part
- **API** stands for **Application Programming Interface**
  - It is a software intermediary that allows two applications to talk to each other
  - Types : **Database APIs , Operating systems APIs, Web APIs**
  - Developers can use **web APIs** to extend the functionality of their apps or sites. E.g Google Maps API
  - **API specifications/protocols : RPC , SOAP , REST , GraphQL**

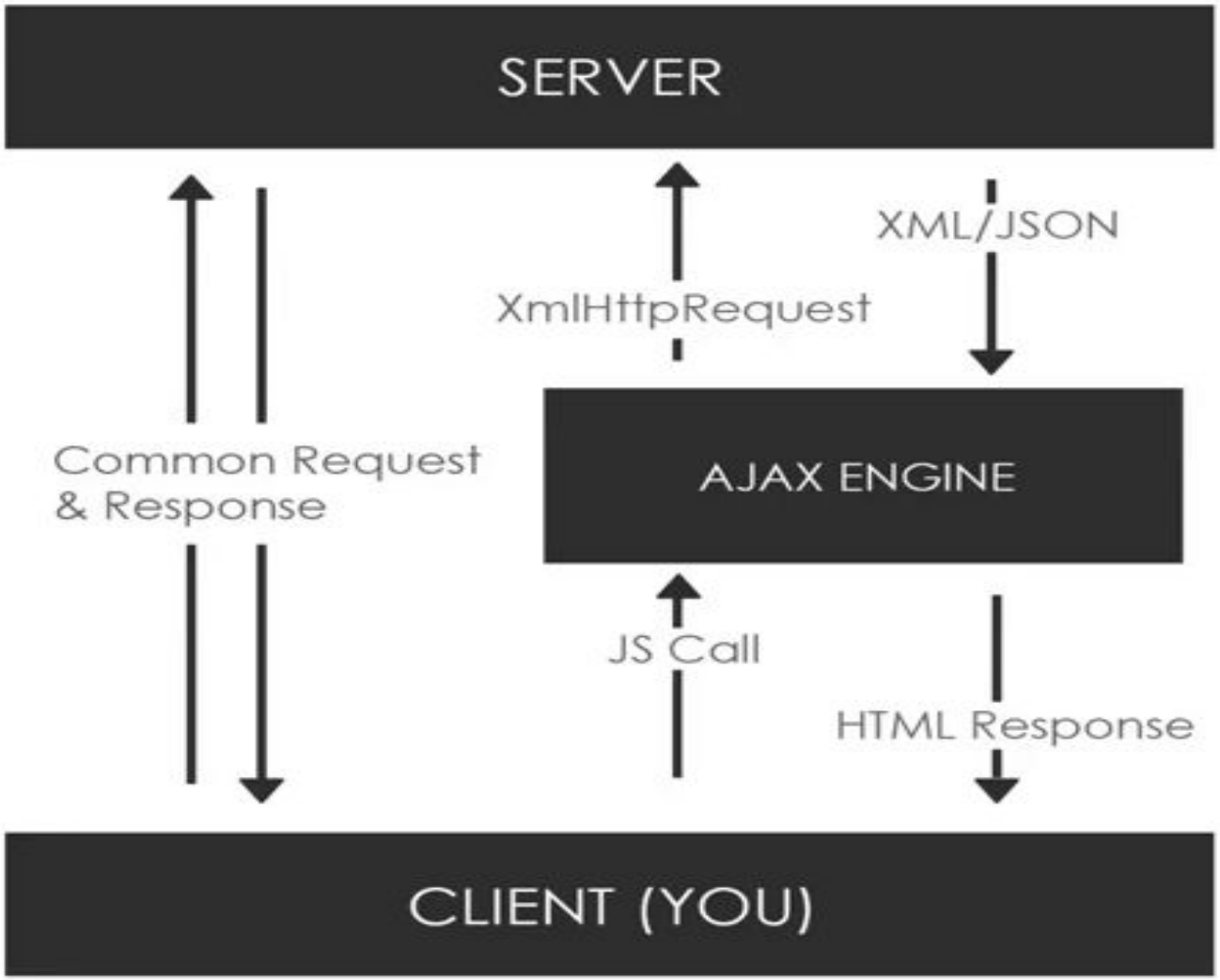
JSON Code vs XML Code

```
{
  "student": [

      {
         "id":"01",
         "name": "Tom",
         "lastname": "Price"
      },

      {
         "id":"02",
         "name": "Nick",
         "lastname": "Thameson"
      }
  ]
}
```

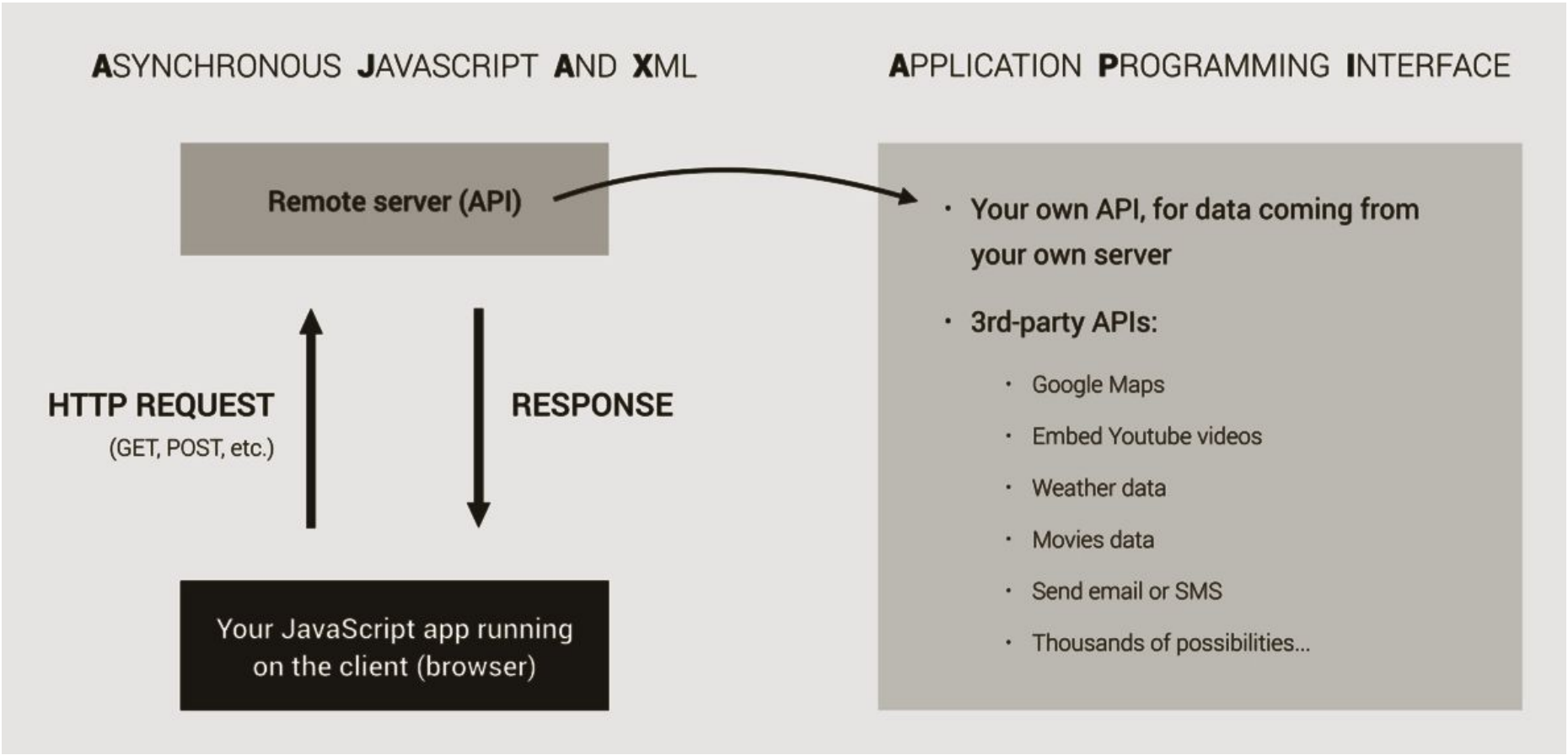
```
<?xml version="1.0" encoding="UTF-8"
?>
<root>
     <student>
          <id>01</id>
          <name>Tom</name>
          <lastname>Price</lastname>
     </student>
     <student>
          <id>02</id>
          <name>Nick</name>
       <lastname>Thameson</lastname>
     </student>
</root>
```

JSON Validator : Link

Ajax and API(ctnd...)



Ajax and API(ctnd...)



XmlHttpRequest(XHR) object

- XHR : is API in the form of an object
- Provided by the browsers JS Environment
- Have methods to transfer data between client and server
- Can be used other protocols other than http
- Can work with data other than XML (JSON , plain text)

Other Libraries

Fetch API (ES6), Axios , Superagent , Jquery , Node HTTP

API Request

```
// 1. Create a new XMLHttpRequest object
let xhr = new XMLHttpRequest();

// 2. Initialize it, usually right after new XMLHttpRequest
xhr.open(method, URL, [async, user, password])

// 3. Send the request over the network
xhr.send([body]);

// 3. Listen to xhr events for response. [load , error , progress]
xhr.onload = function() {
  alert(`Loaded: ${xhr.status} ${xhr.response}`);
};
```

Fetch API

- Fetch : New way in ES6 to make network requests similar to XHR
  - The main difference is that the Fetch API uses Promises
- **Syntax : let promise = fetch(url, [options])**
  - **options :** method, headers etc
- **`Response`** provides multiple promise-based methods to access the body in various formats:
  - **`response.text()`**
  - **`response.json()`**
  - **`response.formData()`**
  - **`response.blob()`**
  - **`response.arrayBuffer()`**

Fetch API Example

Promise

```
fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits')
  .then(response => response.json())
  .then(commits => alert(commits[0].author.login));
```

Async/Wait

```
let url = 'https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits';
let response = await fetch(url);

let commits = await response.json(); // read response body and parse as JSON

alert(commits[0].author.login);
```

Reading Assignment

- Can you demonstrate a long process via a loop where the browser will stack and does not allow us to click a button ?
- What is Cross origin Policy ?
- Compare and Contrast various API specifications

The END !!!