Client Side Web API's

thruskills

https://www.thruskills.com

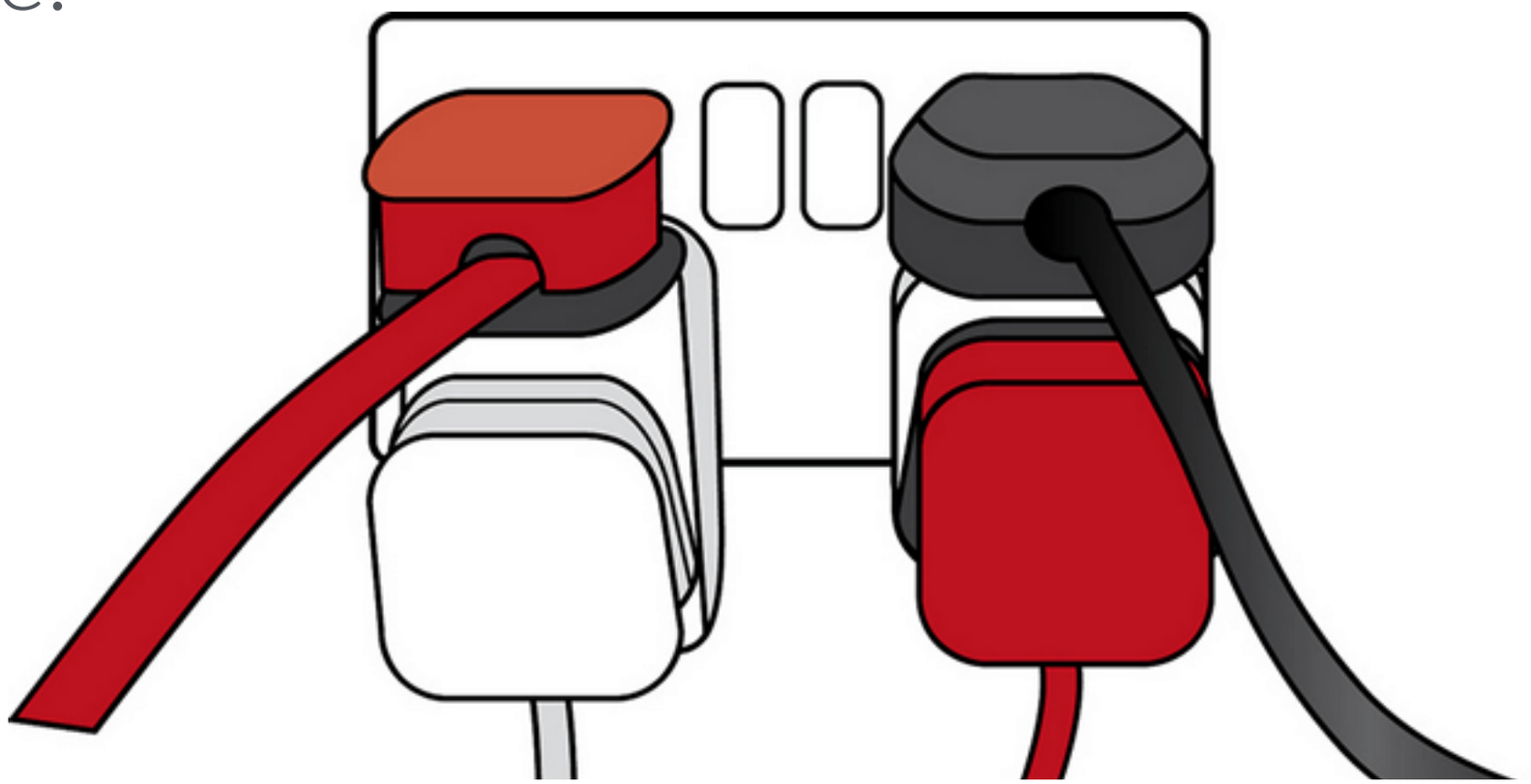
+91 831 737 5392

What you will learn

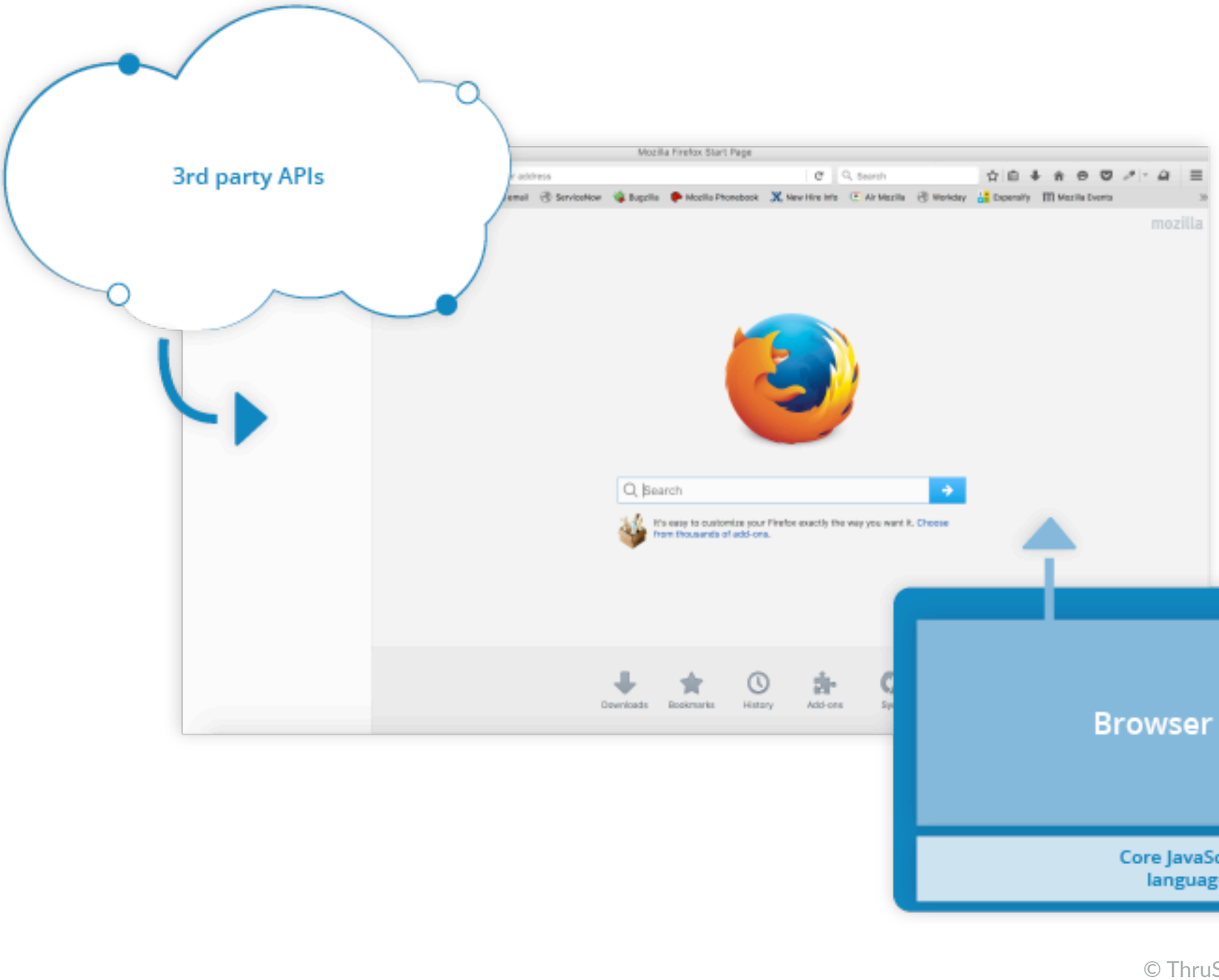
- [x] Introduction to web APIs
- [x] Manipulating documents (DOM)
- [x] Fetching data from the server
- [x] Third party APIs
- [x] Drawing graphics
- [x] Video and audio APIs
- [x] Client-side storage

What are APIs?

Application Programming Interfaces (APIs) are constructs made available in programming languages to allow developers to create complex functionality more easily. They abstract more complex code away from you, providing some easier syntax to use in its place.



APIs in client-side JavaScript



JavaScript, APIs, and other JavaScript tools

JavaScript — A high-level scripting language built into browsers that allows you to implement functionality on web pages/apps. Note that JavaScript is also available in other programming environments, such as Node.

Browser APIs — constructs built into the browser that sit on top of the JavaScript language and allow you to implement functionality more easily.

JavaScript, APIs, and other JavaScript tools

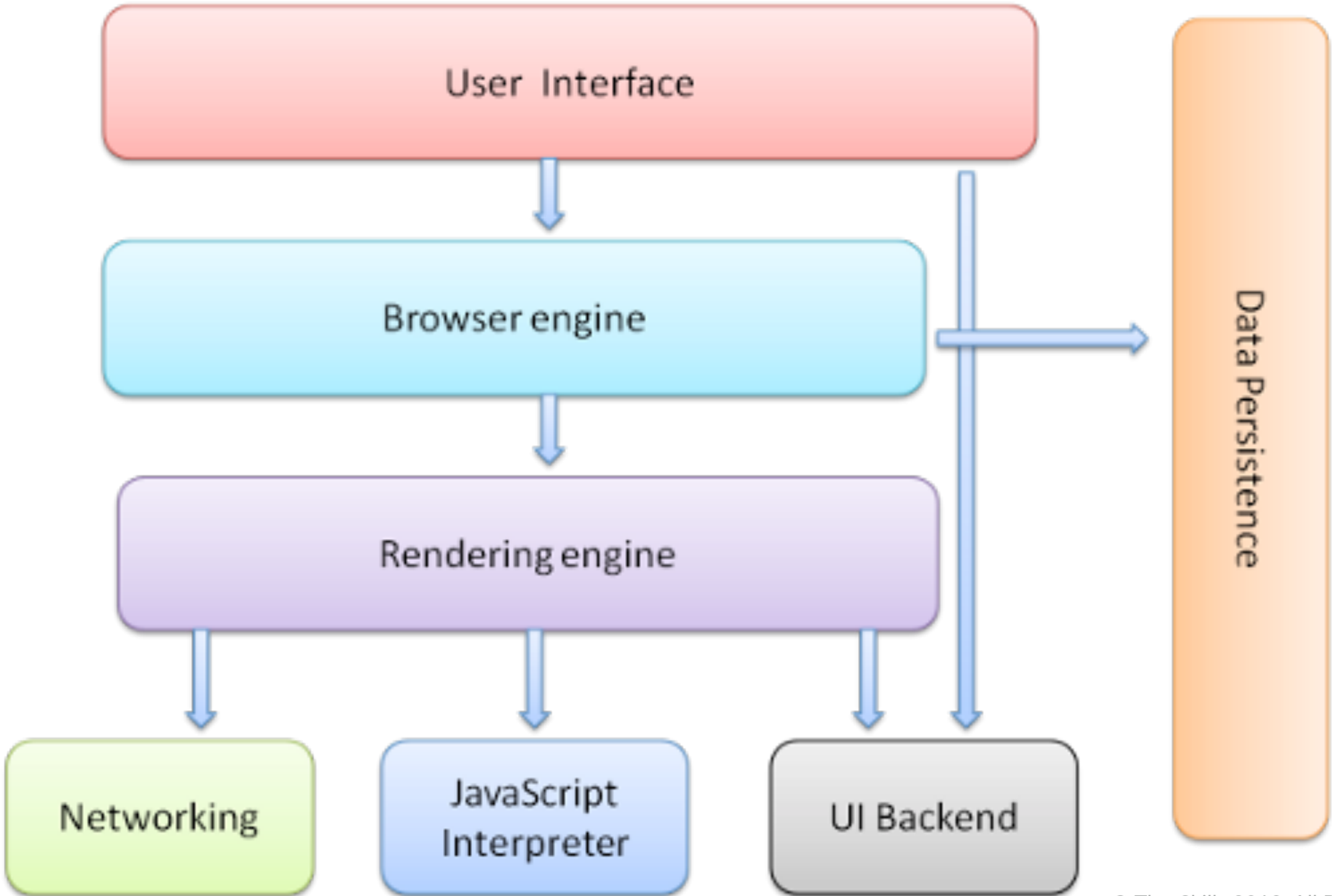
Third party APIs — constructs built into third-party platforms (e.g. Twitter, Facebook) that allow you to use some of those platform's functionality in your own web pages (for example, display your latest Tweets on your web page).

JavaScript libraries — Usually one or more JavaScript files containing custom functions that you can attach to your web page to speed up or enable writing common functionality. Examples include **jQuery**, Mootools and **React**.

JavaScript, APIs, and other JavaScript tools

JavaScript frameworks — The next step up from libraries, JavaScript frameworks (e.g. **Angular** and Ember) tend to be packages of HTML, CSS, JavaScript, and other technologies that you install and then use to write an entire web application from scratch. The key difference between a library and a framework is “Inversion of Control”. When calling a method from a library, the developer is in control. With a framework, the control is inverted: the framework calls the developer's code.

The browser's high level structure



The browser's high level structure...

The user interface: this includes the address bar, back/ forward button, bookmarking menu, etc. Every part of the browser display except the window where you see the requested page.

The browser engine: marshals actions between the UI and the rendering engine.

The rendering engine : responsible for displaying requested content. For example if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen.

The browser's high level structure...

Networking: for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.

UI backend: used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods..

JavaScript interpreter. Used to parse and execute JavaScript code.

Data storage. This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browsers also support storage mechanisms such as localStorage, IndexedDB, WebSQL and FileSystem.

The important parts of a web browser

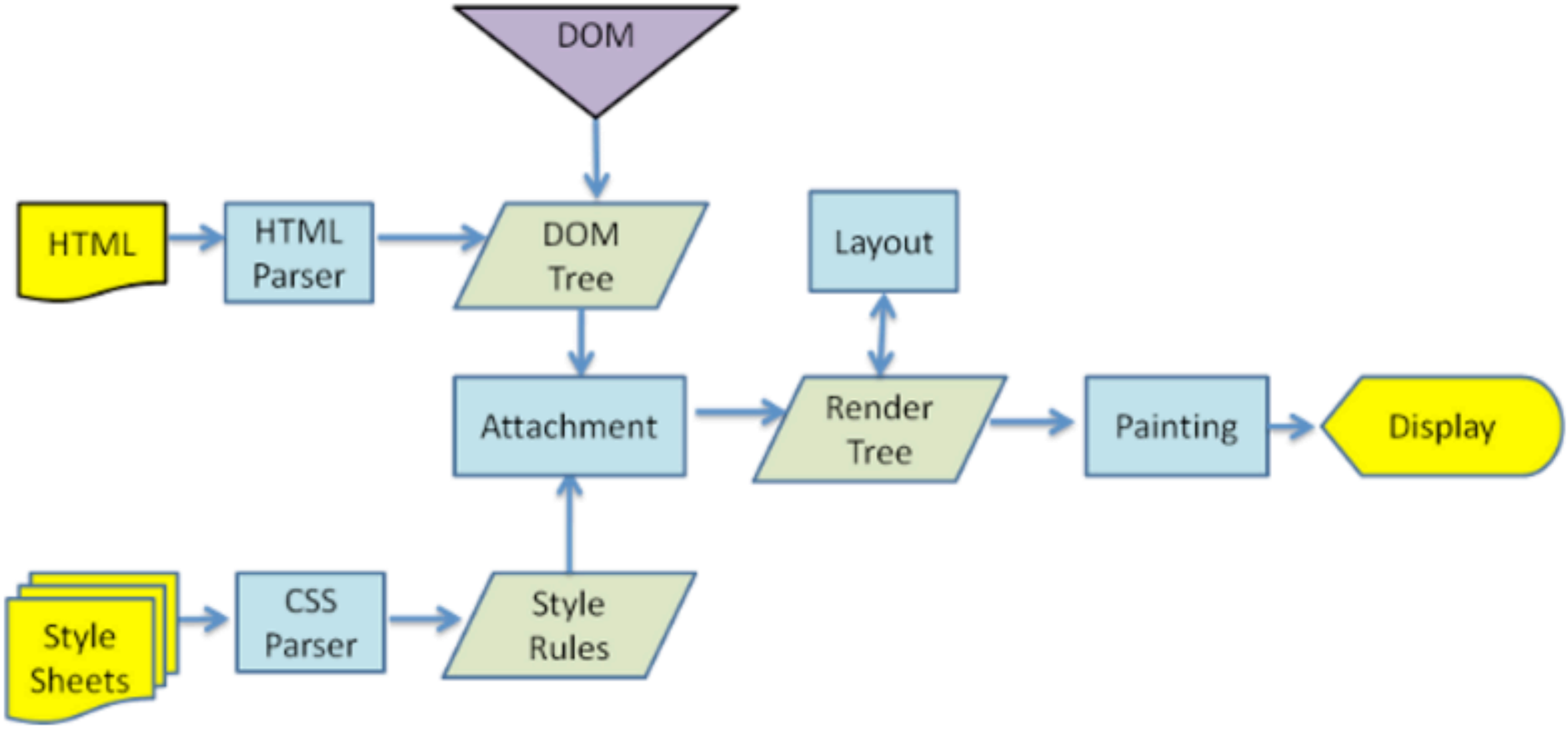
The important parts of a web browser

The **window** is the browser tab that a web page is loaded into; this is represented in JavaScript by the `Window` object.

The **navigator** represents the state and identity of the browser (i.e. the user-agent) as it exists on the web. In JavaScript, this is represented by the `Navigator` object.

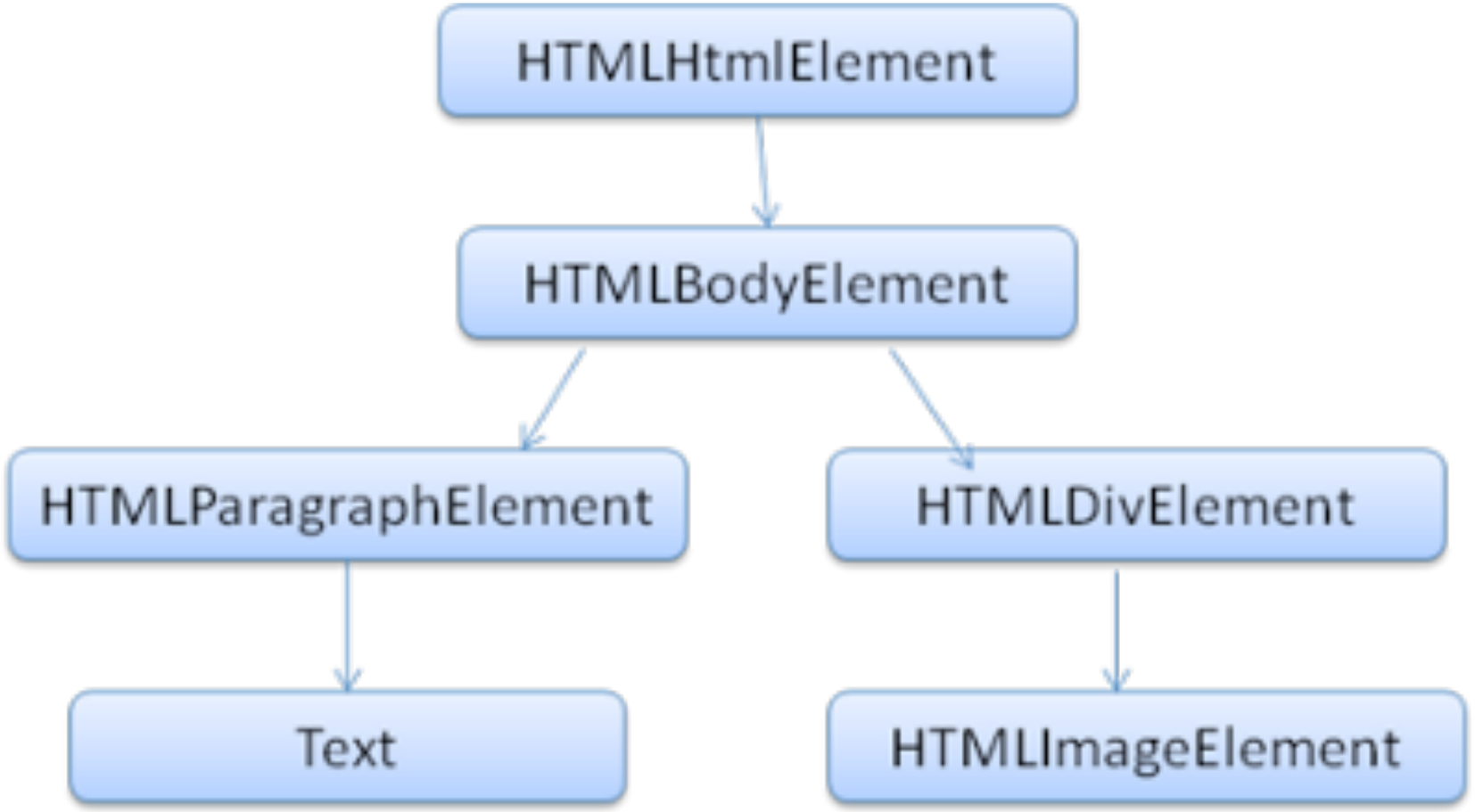
The **document** (represented by the DOM in browsers) is the actual page loaded into the window, and is represented in JavaScript by the `Document` object.

The document object model (DOM)

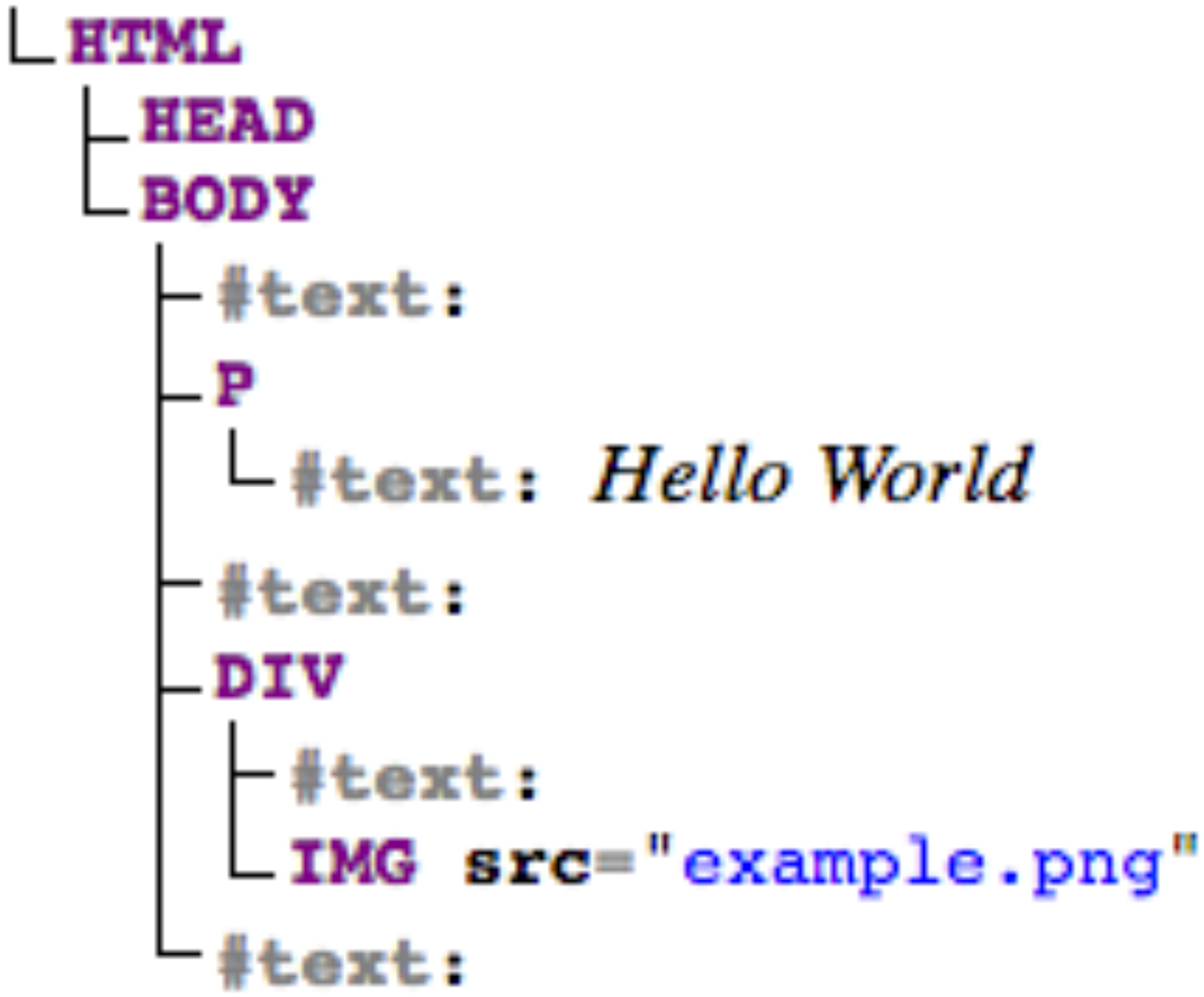


The document object model (DOM)

```
<html>
  <body>
    <p>
      Hello World
    </p>
    <div> <img src="example.png"/></div>
  </body>
</html>
```



The document object model (DOM)



Note: This DOM tree diagram was created using Ian Hickson's Live DOM viewer.

The document object model (DOM)

Element node: An element, as it exists in the DOM.

Root node: The top node in the tree, which in the case of HTML is always the HTML node (other markup vocabularies like SVG and custom XML will have different root elements).

Child node: A node directly inside another node. For example, IMG is a child of SECTION in the above example.

Descendant node: A node anywhere inside another node. For example, IMG is a child of SECTION in the above example, and it is also a descendant. IMG is not a child of BODY, as it is two levels below it in the tree, but it is a descendant of BODY.

Parent node: A node which has another node inside it. For example, BODY is the parent node of SECTION in the above example.

Sibling nodes: Nodes that sit on the same level in the DOM tree. For example, IMG and P are siblings in the above example.

Text node: A node containing a text string.

The document object model (DOM)

document.getElementById(id) : Find an element by element id

document.getElementsByTagName(name) : Find elements by tag name

document.getElementsByClassName(name) : Find elements by class name

element.innerHTML = new html content; Change the inner HTML of an element

element.attribute = new value; Change the attribute value of an HTML element

element.setAttribute(attribute, value); Change the attribute value of an HTML element

element.style.property = new style; Change the style of an HTML element

document.createElement(element); Create an HTML element

document.removeChild(element) ; Remove an HTML element

document.appendChild(element); Add an HTML element

document.replaceChild(element) ; Replace an HTML element

document.write(text) ; Write into the HTML output stream

Any Questions

Thank You

https://www.thruskills.com

+91 831 737 5392

thruskills