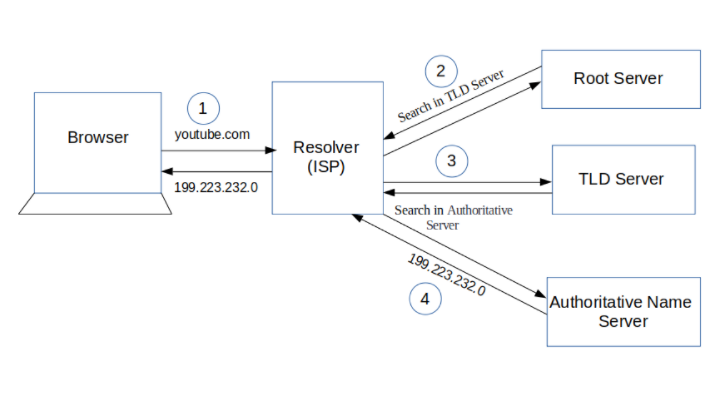
**When User enter the URL in browser, how does the browser fetch the desired output**

When user enter the URL (Uniform Resource Locator) in browser, the first part is to tell browser which protocol it should use. It can be **http, https, ftp** etc. A protocol is a set of rules that browser use for communication over the network. **https** is basically a secure version for exchanging the information.

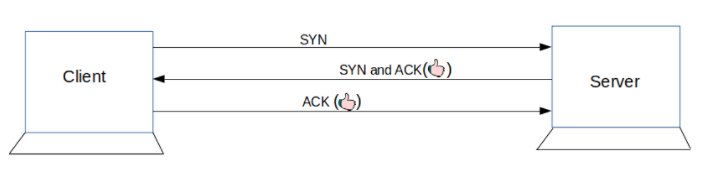
the first thing that needs to happen after selecting protocol by browser is to resolve IP address associated with domain name. DNS (Domain Name system) helps us to provide the IP address like our phonebook gives a mobile number which is associated with the person’s name. there are four layers through which this domain name query goes through:

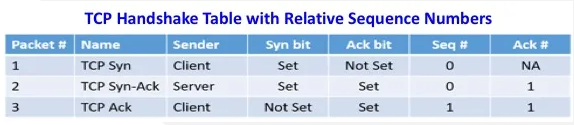
1. After hitting the URL, the **browser cache** is checked. As browser maintains its DNS records for some amount of time for the website you have visited earlier.
2. If IP address not found above step, then DNS query runs in **OS cache** followed by router cache.
3. If in the above step, DNS query does not get resolved, then it takes the help of **resolver server**. Resolver server is nothing but your **ISP (internet service provider)** where DNS query runs in **ISP cache**.
4. If in 3rd step as well, no result found, then request sends to **top or root server** of the DNS hierarchy and it tells from where this information you can get like you are searching top level domain (.com, .net, .gov, .org). it tells the resolver server to search **TLD server (**Top level domain**).** Now, resolver ask TLD server to give IP address of domain name. TLD server ask resolver (ISP) to **Authoritative Name Server,** whose responsible to knowing everything about the domain name. Finally, after this resolver (ISP) gets the IP address and send back to the browser.



Once the IP address found, it initiates connection with it. To communicate over the network, internet protocol is followed. **TCP/IP is most common protocol**. A connection is built between two using a process called **TCP 3-way handshake**. Let’s understand the process in brief:

1. A client computer sends a **SYN message** means, whether second computer is open for new connection or not.
2. Another computer or server, if open for new connection, it sends **acknowledge message** with SYN message as well.
3. After this, client computer receives its message and acknowledge by sending an **ACK message**.





Finally, the connection is built between client and server. Now, they both can communicate with each other and share information. Browser (client) sends a request to a server that I want this content. The server knows everything of what response it should send for every request. Hence, the server responds back. This response contains every information that you requested like web page, status-code, cache-control, etc.

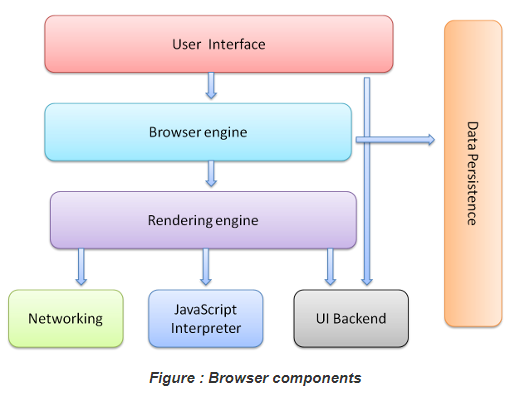
* **Functionality of the browser –**

The browser is the medium that allows you to make a request and lets a server serve you. The main function of the browser is to present web source and displaying in the browser. The resource is usually an HTML, CSS, Image, text document.

The way the browser interprets and displays HTML files is specified in the HTML and CSS specifications. These specifications are maintained by W3C (World Wide Web Consortium) organization, which is standards organization for the web.

* **High level of component of a browser –**

1. The user interface: This includes the address bar, back/forward button, bookmarking menu etc.
2. The browser engine: Set the proper order of actions between UI and the rendering engine.
3. 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.
4. Networking: For network calls such as HTTP request, using different implementation for different platform behind a platform-independent interface.
5. UI backend: Use for drawing basic widget like combo boxes and windows. The backend exposes a generic interface that is not platform specific.
6. JavaScript interpreter: Use to parse and execute JavaScript code.
7. Data storage: This is a persistence layer, the browser may need to save all sort of data locally, such as cookies. Browser also support storage mechanism such as localStorage, IndexedDB, WebSQL and FileSystem.



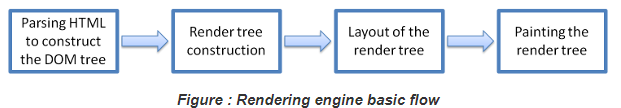
* **Rendering Engine and its uses –**

The responsibility of the rendering engine is display of the requested contents on the browser screen. By default, the rendering engine can display HTML and XML documents and images. It can display other types of data via plug-ins or extension

For example, displaying PDF documents using a PDF viewer plug-in.

Different browsers use different rendering engines: **Internet Explorer uses Trident**, **Firefox uses Gecko**, **Safari uses WebKit**. **Chrome and Opera (from version 15) use Blink, a fork of WebKit**. WebKit is an open-source rendering engine which started as an engine for the Linux platform and was modified by Apple to support Mac and Windows.

The rendering engine will start getting the contents of the requested document from the **networking layer**. This will usually be done **in 8kB chunks**. After that, this is the basic flow of the rendering engine:

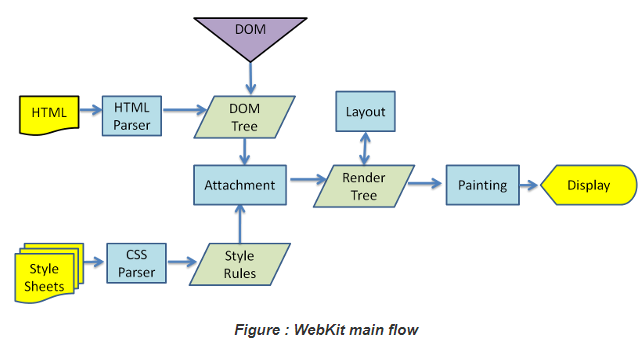


The rendering engine will start the parsing the HTML documents and convert elements into DOM (document object model) nodes in a tree called the **content tree**. The engine will parse the style data, both in external CSS files and in style elements. Now, styling information with visual instruction in the HTML used to create another tree: **render tree**. render tree contains rectangles with visual attributes like color and dimensions. The rectangles are in the right order to be displayed on the screen.

After the construction of the render tree is goes through the **layout process.** This means giving each node the exact coordinates where it should appear on the screen.

The Next stage, is painting the render tree will be traversed and each node will be painted using the UI backend layer.

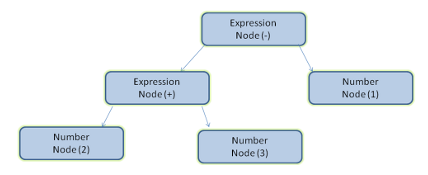
This is gradual process for better user experience, the rendering engine will try to display contents on screen as soon as possible. It will not wait until all HTML is parsed before starting to build and layout the render tree. Parts of the content will be parsed and displayed, while the process continues with the rest of the contents that keep coming from the network layer.



* **Parsers –**

Parsing is very significant process within the rendering engine. Parsing a document means translating it to structure the code can use **or** Parsing means analyzing and converting a program into an internal format that a runtime environment can actually run. The result of parsing is usually a tree of nodes that represent the structure of the document.

For example, parsing the expression 2 + 3 - 1 could return this tree:



Parsing can be separated in two sub process: **lexical analysis and syntax analysis**.

**Lexical analysis** is the process of breaking the input into tokens. Tokens are the language vocabulary: the collection of valid building blocks. In human language it will consist of all the words that appear in the dictionary for that language.

**Syntax analysis** is the applying of the language syntax rules.

Parsers usually divide the work between two components: the lexer (sometimes called tokenizer) that is responsible for breaking the input into valid tokens, and the parser that is responsible for constructing the parse tree by analyzing the document structure according to the language syntax rules.

There are two types of parsers: **top-down parsers and bottom-up parsers**. Top-down parsers examine the high-level structure of the syntax and try to find a rule match. Bottom-up parsers start with the input and gradually transform it into the syntax rules, starting from the low-level rules until high level rules are met.

HTML cannot be parsed using the top-down or bottom-up parsers because HTML can not be easily defined by a context that parsers need but they will be used in parsing CSS and JavaScript. HTML parsing involves tokenization and tree construction. HTML tokens include start and end tags, as well as attribute names and values. If the document is well-formed, parsing it is straightforward and faster. The parser parses tokenized input into the document, building up the document tree.

When the HTML parser finds non-blocking resources, such as an image, the browser will request those resources and continue parsing. Parsing can continue when a CSS file is encountered, **but <script> tags—particularly those without an async or defer attribute—blocks rendering, and pauses parsing of HTML.**

When the browser encounters CSS styles, it parses the text into the CSS Object Model a data structure it then uses for styling layouts and painting. The browser then creates a render tree from both these structures to be able to paint the content to the screen. JavaScript is also downloaded, parsed, and then executed. JavaScript parsing is done during compile time or whenever the parser is invoked, such as during a call to a method.

* **Script Processor –**

The model of the web is synchronous. Creator expect scripts to be parsed and executed immediately when the parser reaches a <script> tag. **The parsing of the document halts until the script has been executed. If the script is external then the resource must first be fetched from the network**. **this is also done synchronously, and parsing halts until the resource is fetched**. This was the model for many years and is also specified in HTML4 and 5 specifications. Creator can add the "defer" attribute to a script, in which case it will not halt document parsing and will execute after the document is parsed.

HTML5 adds an option to mark the script as asynchronous so it will be parsed and executed by a different thread.

* **Tree Construction -**

While the DOM tree is being constructed, the browser constructs another tree, the render tree. This tree is of visual elements in the order in which they will be displayed. It is the visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order. A renderer knows how to lay out and paint itself and its children.

Each renderer represents a rectangular area usually corresponding to a node's CSS box. It includes geometric information like width, height and position. The box type is affected by the "display" value of the style attribute that is relevant to the node.

* **Order of scripts processing –**

when the parser reaches a <script> tag. The parsing of the document halts until the script has been executed. But for resolving this we can add defer attributes or mark the script as asynchronous. After this, while executing scripts, another thread parses the rest of the document and finds out what other resources need to be loaded from the network and loads them. In this way, resources can be loaded on parallel connections and overall speed is improved it’s called **speculative parsing**. Note: the speculative parser only parses references to external resources like external scripts, style sheets and images: it doesn't modify the DOM tree–that is left to the main parser.

* **Layout and painting –**

When the renderer is created and added to the tree, it does not have a position and size. Calculating these values is called layout or reflow. Layout is a recursive process. It begins at the root renderer, which corresponds to the <html> element of the HTML document. The position of the root renderer is 0,0 and its dimensions are the viewport–the visible part of the browser window.

Layout continues recursively through some or all of the frame hierarchy, computing geometric information for each renderer that requires it. HTML uses a flow-based layout model, meaning that most of the time it is possible to compute the geometry in a single pass.

In order not to do a full layout for every small change, browsers use a **dirty bit system**. A renderer that is changed or added marks itself and its children as **dirty: needing layout**.

There are two flags: **dirty** and **children are dirty** which means that although the renderer itself may be OK, it has at least one child that needs a layout.

The layout usually has the following pattern:

* Parent renderer determines its own width.
* Parent goes over children and

1. Place the child renderer (sets its x and y).
2. Calls child layout if needed–they are dirty or we are in a global layout, or for some other reason–which calculates the child's height.

* Parent uses children's accumulative heights and the heights of margins and padding to set its own height–this will be used by the parent renderer's parent.
* Sets its dirty bit to false.

**In the painting stage**, the render tree is traversed and the renderer's **paint ()** method is called to display content on the screen. Painting uses the UI infrastructure component.