What happens when you type “google.com”?

Sotnikov Ivan (3431101/90002)

Table of Contents

The "g" key is pressed

The virgin Mechanical

The chad Opto-Mechanical

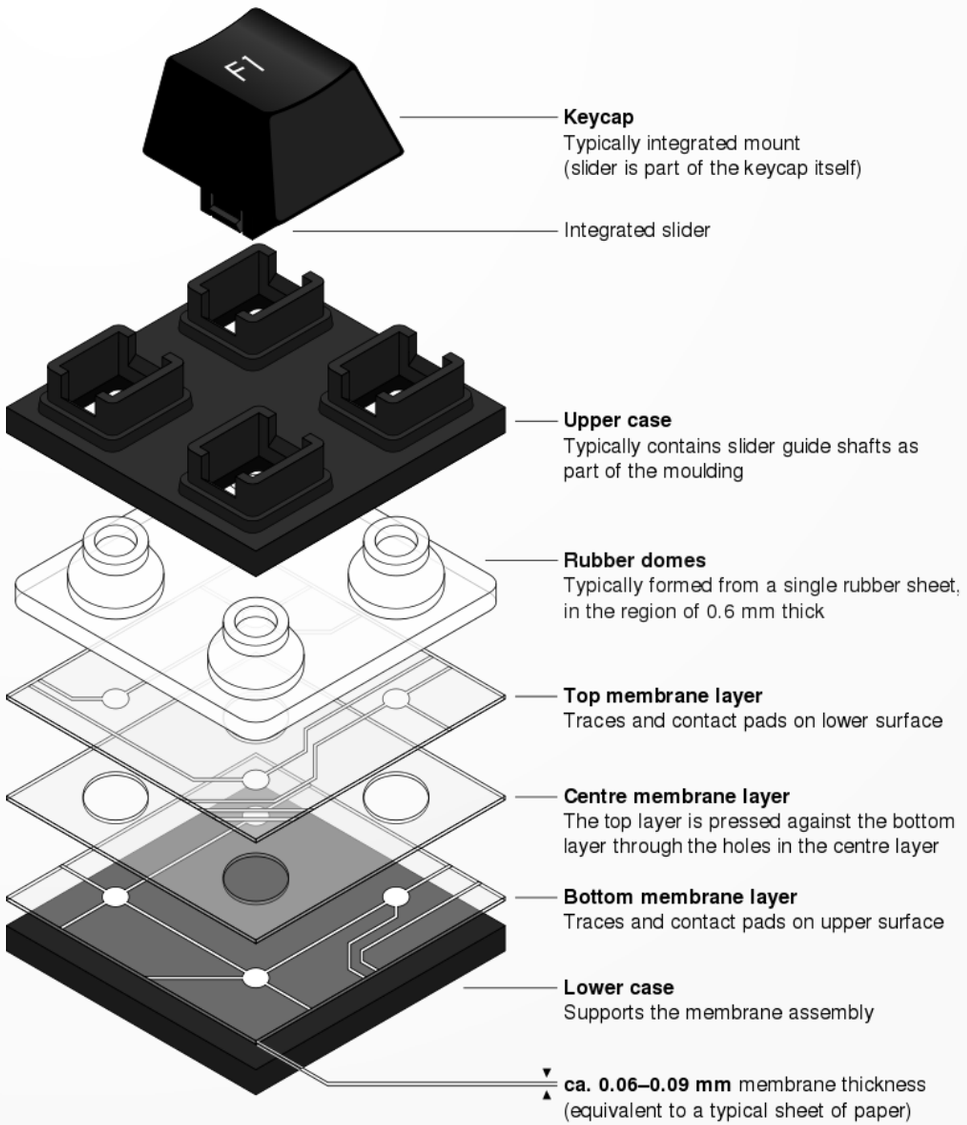
Light and Clicky

Light and Instant

The "g" key is pressed

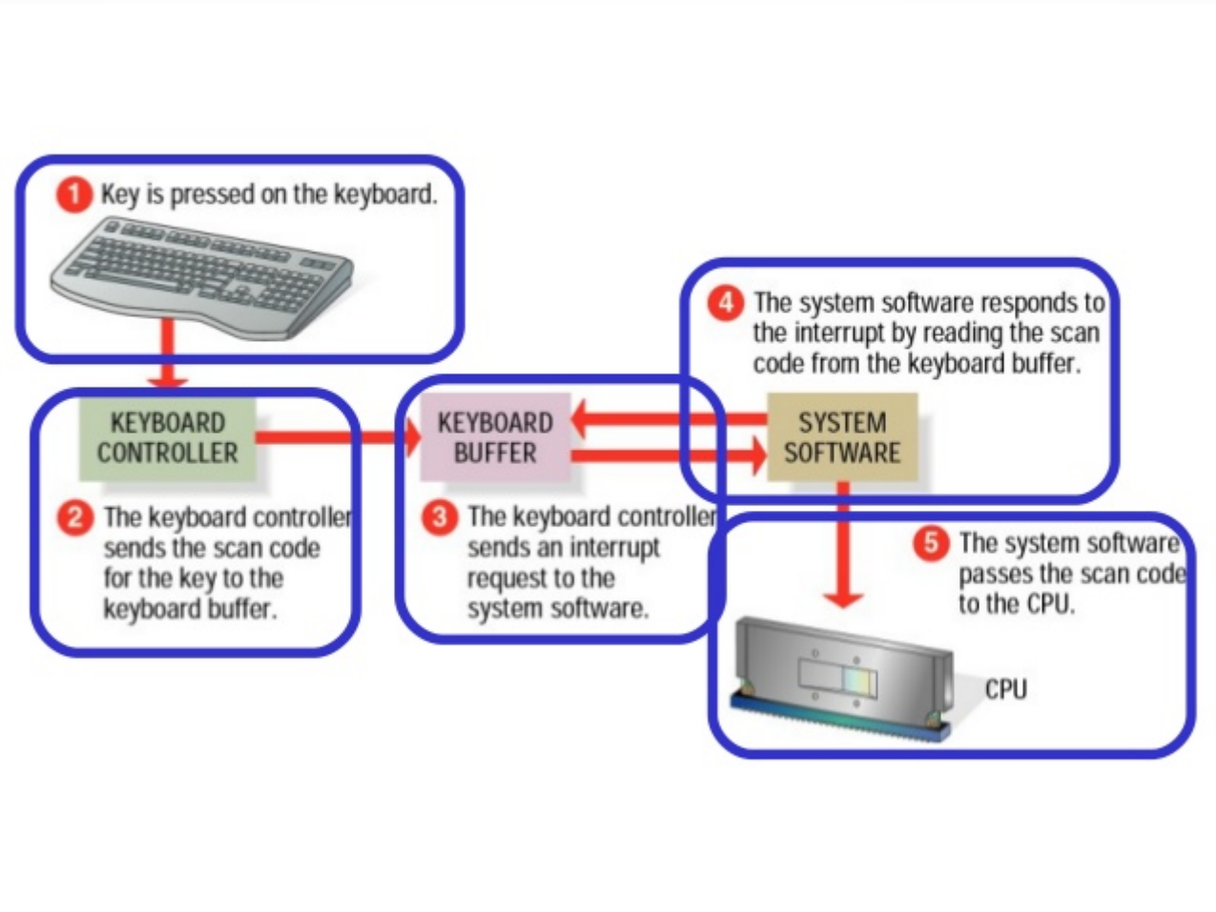
What ~~we all~~ I use

Garbage Membrane

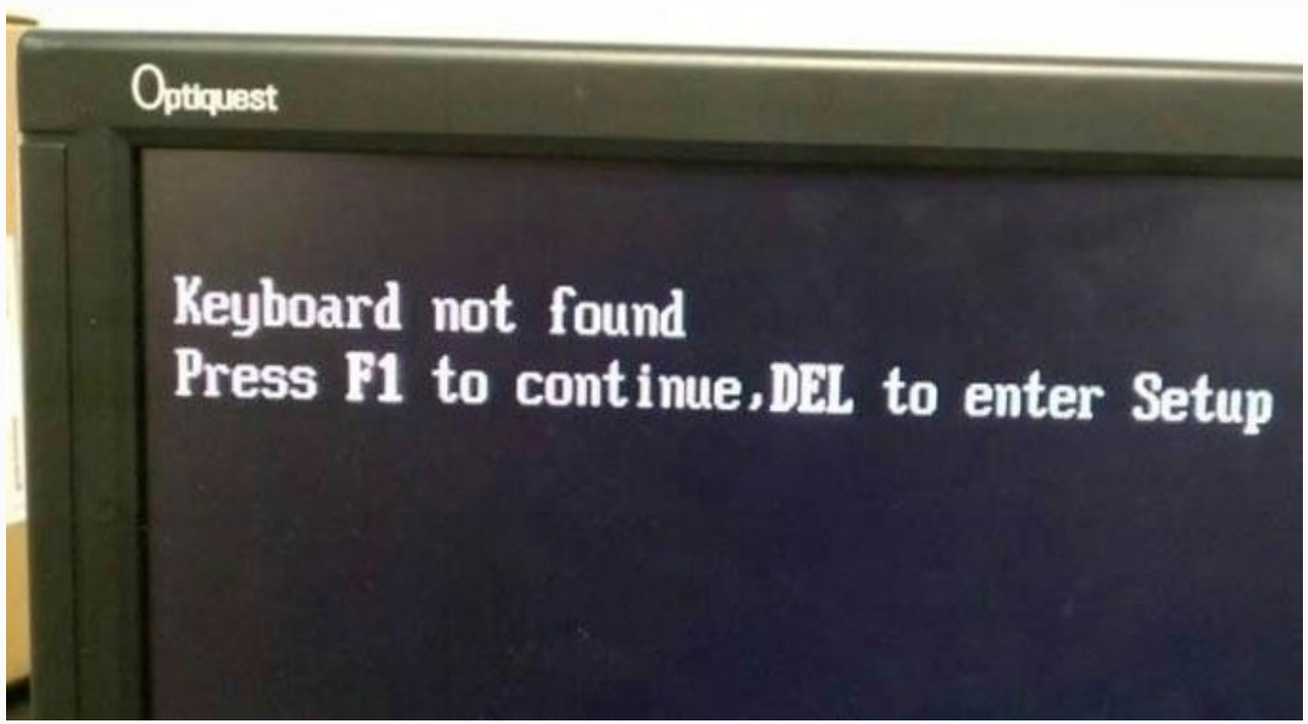


The "enter" key bottoms out

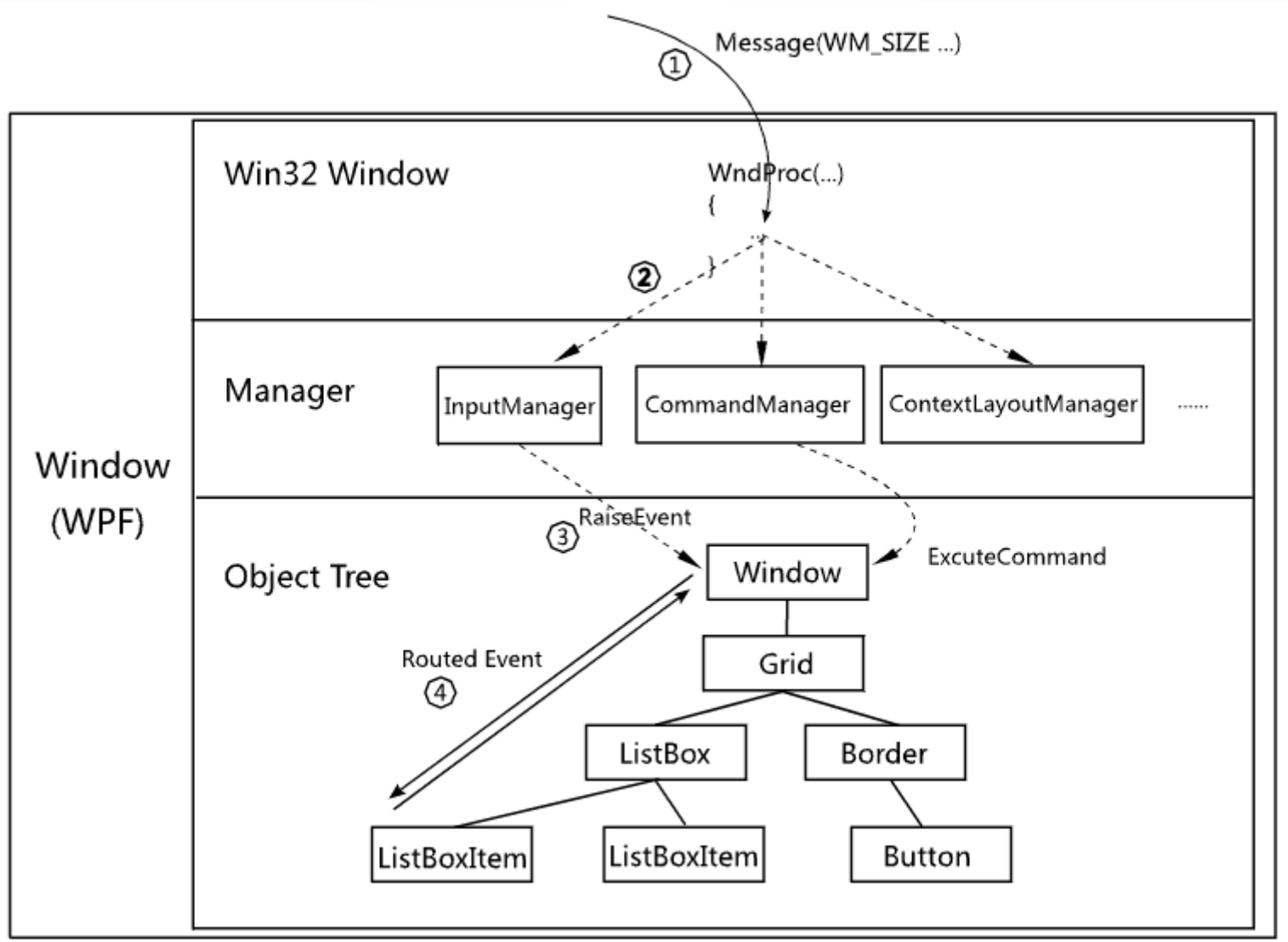
In the case of the USB keyboard:



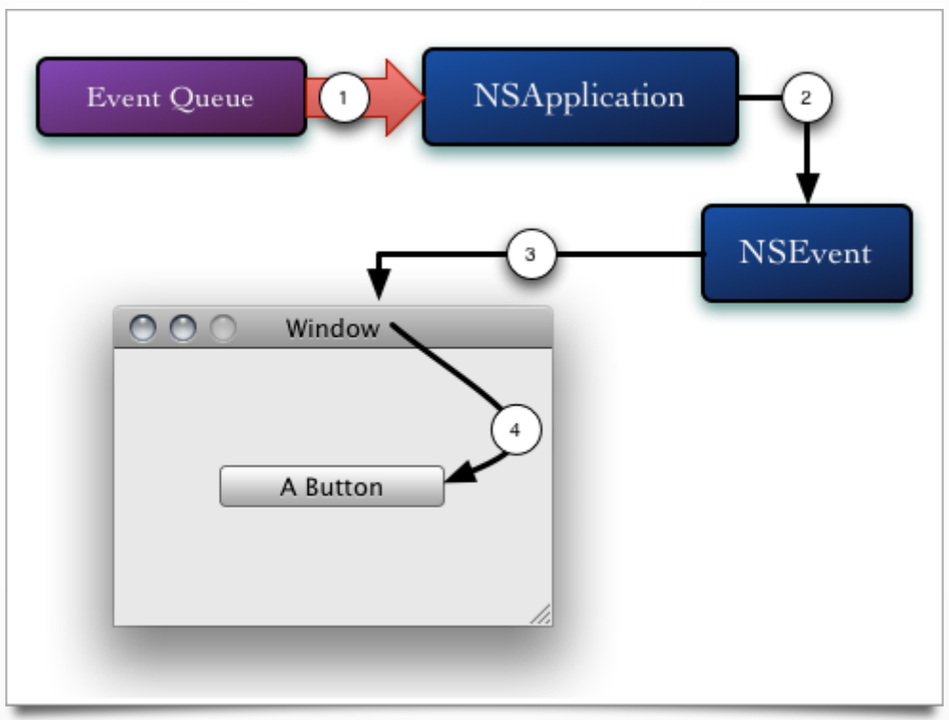
In the case of Virtual Keyboard (as in touch screen devices):



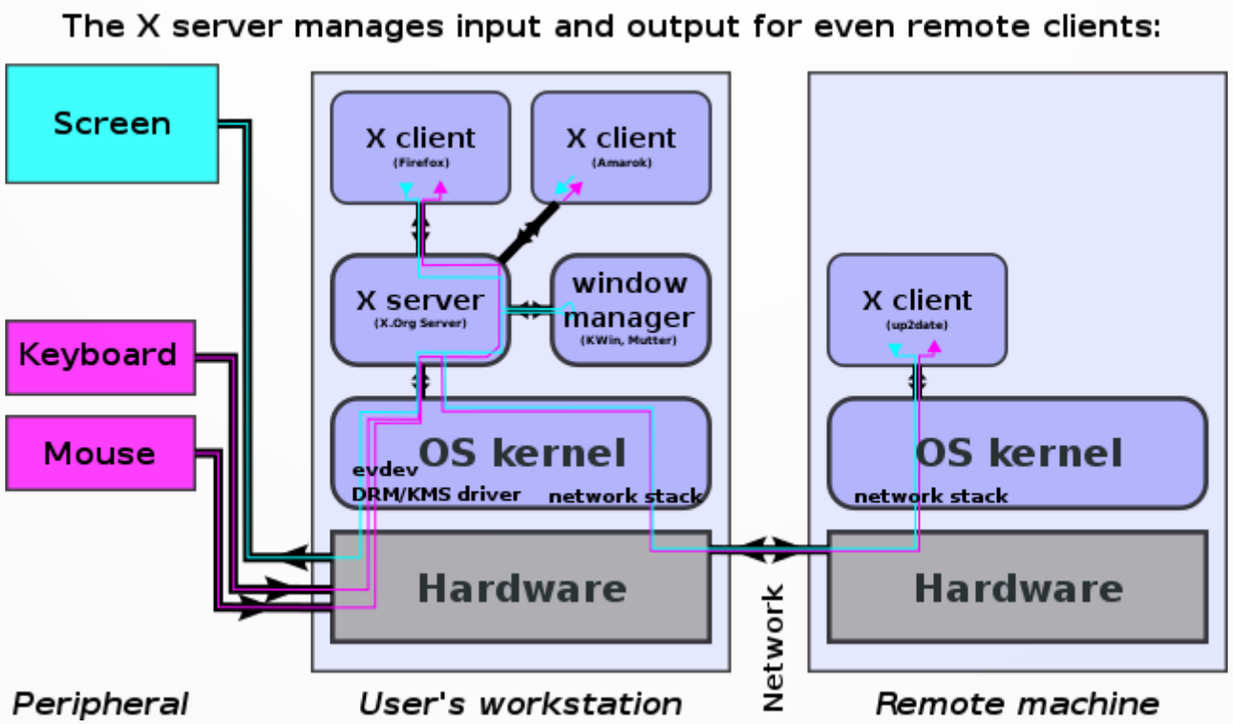
(On Windows) A ***WM_KEYDOWN*** message is sent to the app



(On OS X) A ***KeyDown*** NSEvent is sent to the app



(On GNU/Linux) the Xorg server listens for keycodes



Parse URL

- First, escape chars:
  - string.replace(/([.*+?^=!:${}()|[\]\/\\])/g, '\\$1')
- Second, parse:
  - /\b((?:[a-z][\w-]+:(?:\/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}\/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:'".,<>?«»“”‘’]))/ig

Is it a URL or a search term?

When no protocol or valid domain name is given the browser proceeds to feed the text given in the address box to the browser's default web search engine. In many cases the URL has a special piece of text appended to it to tell the search engine that it came from a particular browser's URL bar.

Convert non-ASCII Unicode characters in hostname

- The browser checks the hostname for characters that are not in *a-z*, *A-Z*, *0-9*, -, or ..
- Since the hostname is *google.com* there won't be any, but if there were the browser would apply Punycode encoding to the hostname portion of the URL.

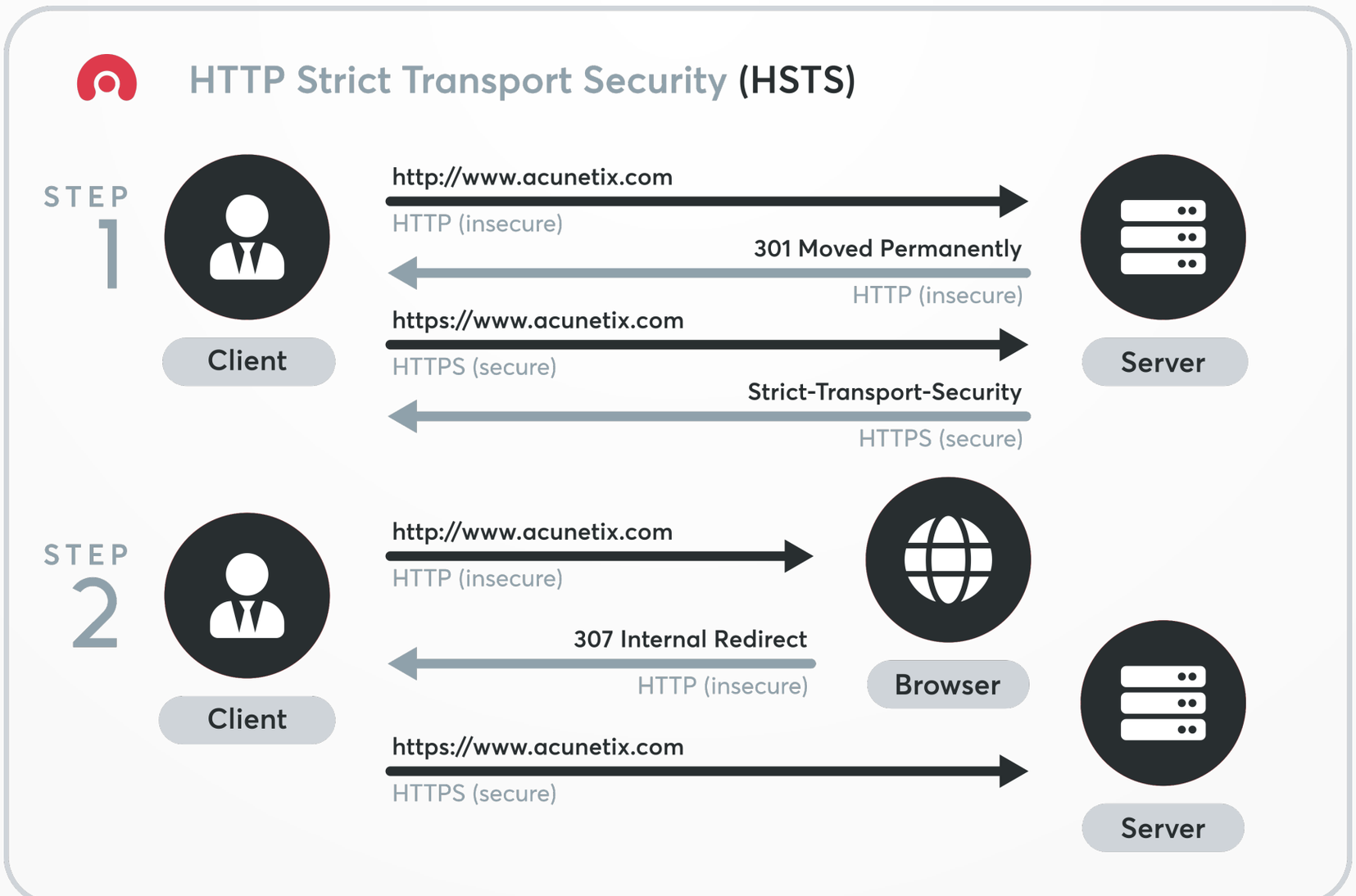
Convert non-ASCII Unicode characters in hostname

Punycode is a way to represent Unicode with the limited character subset of ASCII supported by the Domain Name System (DNS).

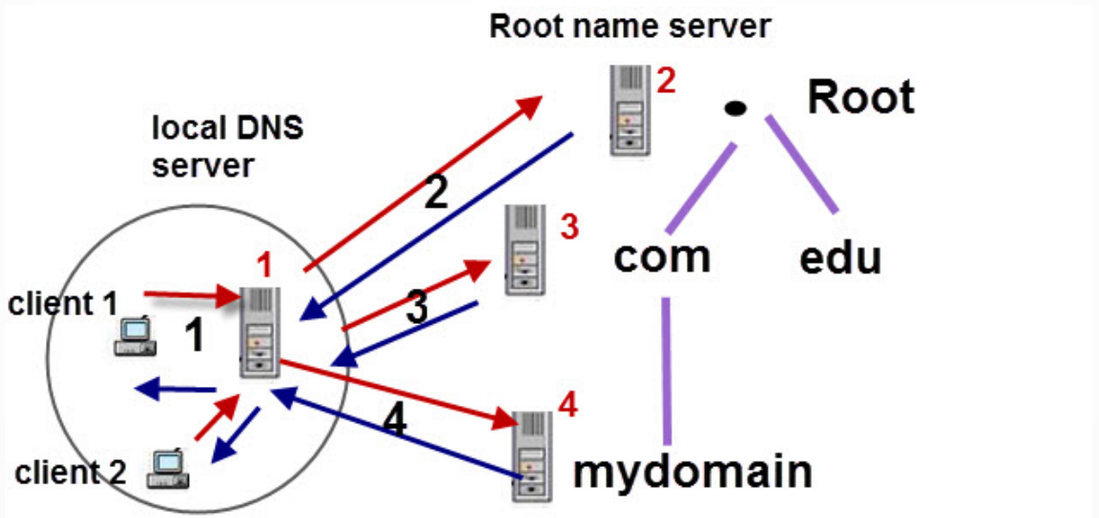
For example "čáslav" (Czech town) would be encoded as "xn—slav-4na7x".

- And "правительство.рф" (Goverment site) would be encoded as "xn--80aealotwbjpid2k.xn--p1ai".

Check HSTS list



DNS lookup



First Query

1. Client 1 queries local DNS server (1) for host www.mydomain.com
2. Local DNS server doesn't know the Answer so it queries a Root DNS server. (2)
3. Root server refers DNS server 1 to DNS server 3 responsible for .com doman name space
4. DNS server 3 refers DNS server 1 to DNS server 4 responsible for the mydomain doman name space.
5. DNS server 4 returns the answer to DNS server 1. DNS server 1 returns the answer to client 1.

Second Query

1. Client 2 queries local DNS server (1) for host www.mydomain.com
2. DNS server 1 return the answer to client 2 from it's DNS cache.

Bonus ~~meme~~: OSI model

OSI model				
Layer			Protocol data unit (PDU)	Function[6]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

Bonus ~~meme~~: OSI model

OSI model
by layer

7. Application layer [hide]
NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · DHCP · Netconf · *more....*

6. Presentation layer [hide]
MIME · XDR · ASN.1

5. Session layer [hide]
Named pipe · NetBIOS · SAP · PPTP · RTP · SOCKS · SPDY

4. Transport layer [hide]
TCP · UDP · SCTP · DCCP · SPX

3. Network layer [hide]
IP (IPv4 · IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk · X.25 PLP

2. Data link layer [hide]
ATM · ARP · IS-IS · SDLC · HDLC · CSLIP · SLIP · GFP · PLIP · IEEE 802.2 · LLC · MAC · L2TP · IEEE 802.3 · Frame Relay · ITU-T G.hn DLL · PPP · X.25 LAPB · Q.922 LAPF

1. Physical layer [hide]
EIA/TIA-232 · EIA/TIA-449 · ITU-T V-Series · I.430 · I.431 · PDH · SONET/SDH · PON · OTN · DSL · IEEE 802.3 · IEEE 802.11 · IEEE 802.15 · IEEE 802.16 · IEEE 1394 · ITU-T G.hn PHY · USB · Bluetooth · RS-232 · RS-449

V • T • E

ARP process

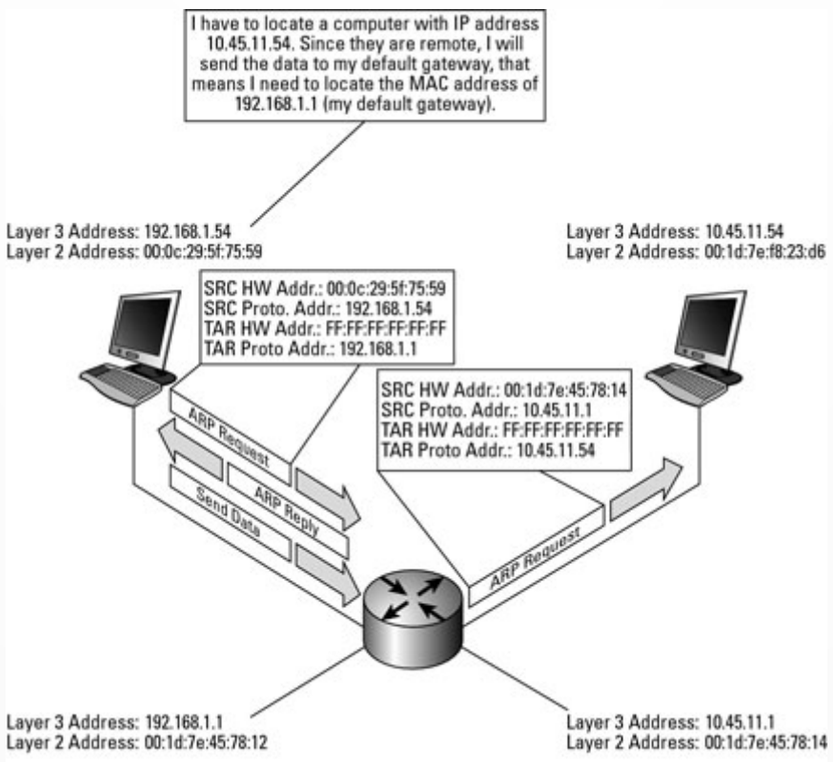
In order to send an ARP (Address Resolution Protocol) broadcast the network stack library needs the target IP address to look up. It also needs to know the MAC address of the interface it will use to send out the ARP broadcast.

The ARP cache is first checked for an ARP entry for our target IP. If it is in the cache, the library function returns the result: Target IP = MAC.

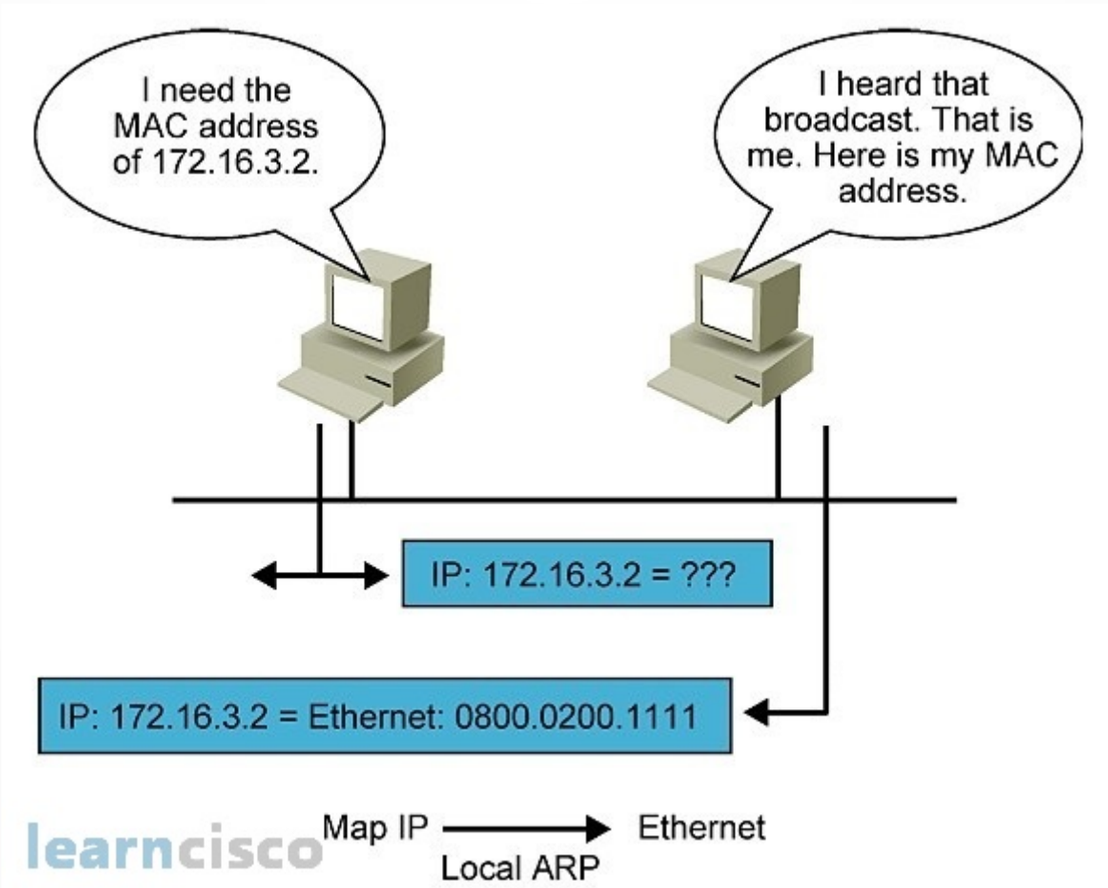
If the entry is not in the ARP cache:

- The route table is looked up, to see if the Target IP address is on any of the subnets on the local route table. If it is, the library uses the interface associated with that subnet. If it is not, the library uses the interface that has the subnet of our default gateway.
- The MAC address of the selected network interface is looked up.
- The network library sends a Layer 2 (data link layer of the OSI model) ARP request:
  - *Sender MAC: interface:mac:address:here*
  - *Sender IP: interface.ip.goes.here*
  - *Target MAC: FF:FF:FF:FF:FF:FF (Broadcast)*
  - *Target IP: target.ip.goes.here*

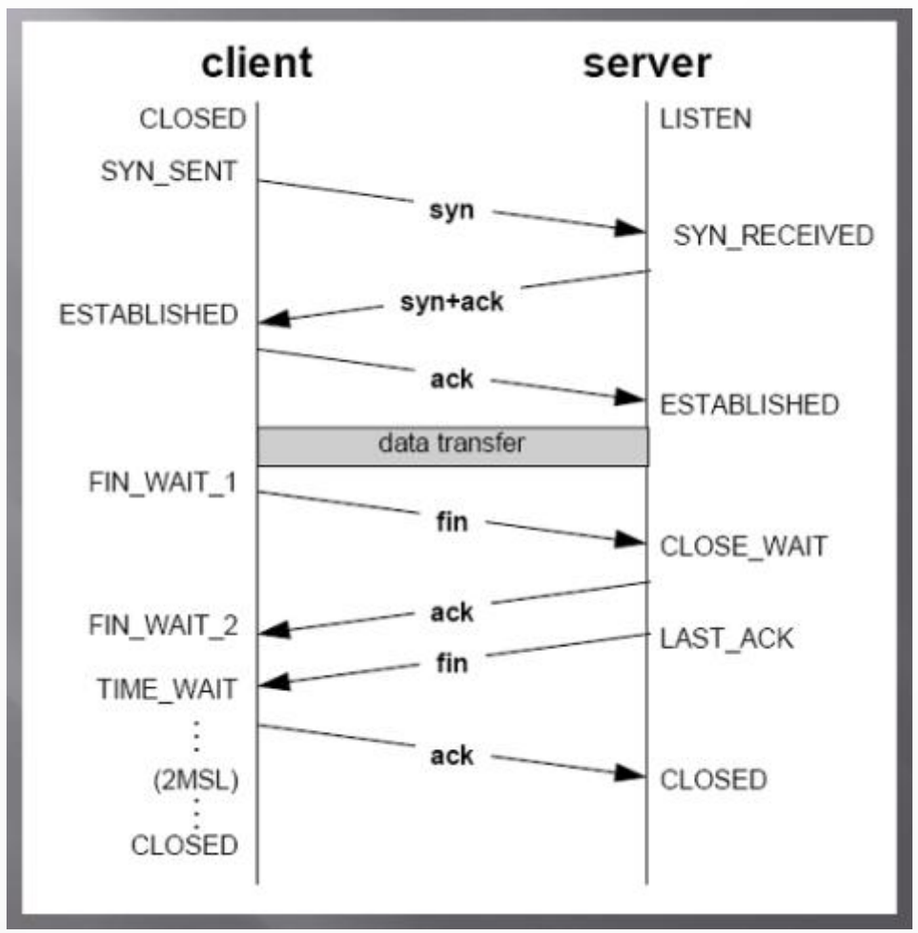
ARP process


I have to locate a computer with IP address 10.45.11.54. Since they are remote, I will send the data to my default gateway, that means I need to locate the MAC address of 192.168.1.1 (my default gateway).
Layer 3 Address: 192.168.1.54
Layer 2 Address: 00:0c:29:5f:75:59
Layer 3 Address: 10.45.11.54
Layer 2 Address: 00:1d:7e:f8:23:d6
SRC HW Addr.: 00:0c:29:5f:75:59
SRC Proto. Addr.: 192.168.1.54
TAR HW Addr.: FF:FF:FF:FF:FF:FF
TAR Proto Addr.: 192.168.1.1
SRC HW Addr.: 00:1d:7e:45:78:14
SRC Proto. Addr.: 10.45.11.1
TAR HW Addr.: FF:FF:FF:FF:FF:FF
TAR Proto Addr.: 10.45.11.54
ARP Request
ARP Reply
Send Data
ARP Request
Layer 3 Address: 192.168.1.1
Layer 2 Address: 00:1d:7e:45:78:12
Layer 3 Address: 10.45.11.1
Layer 2 Address: 00:1d:7e:45:78:14

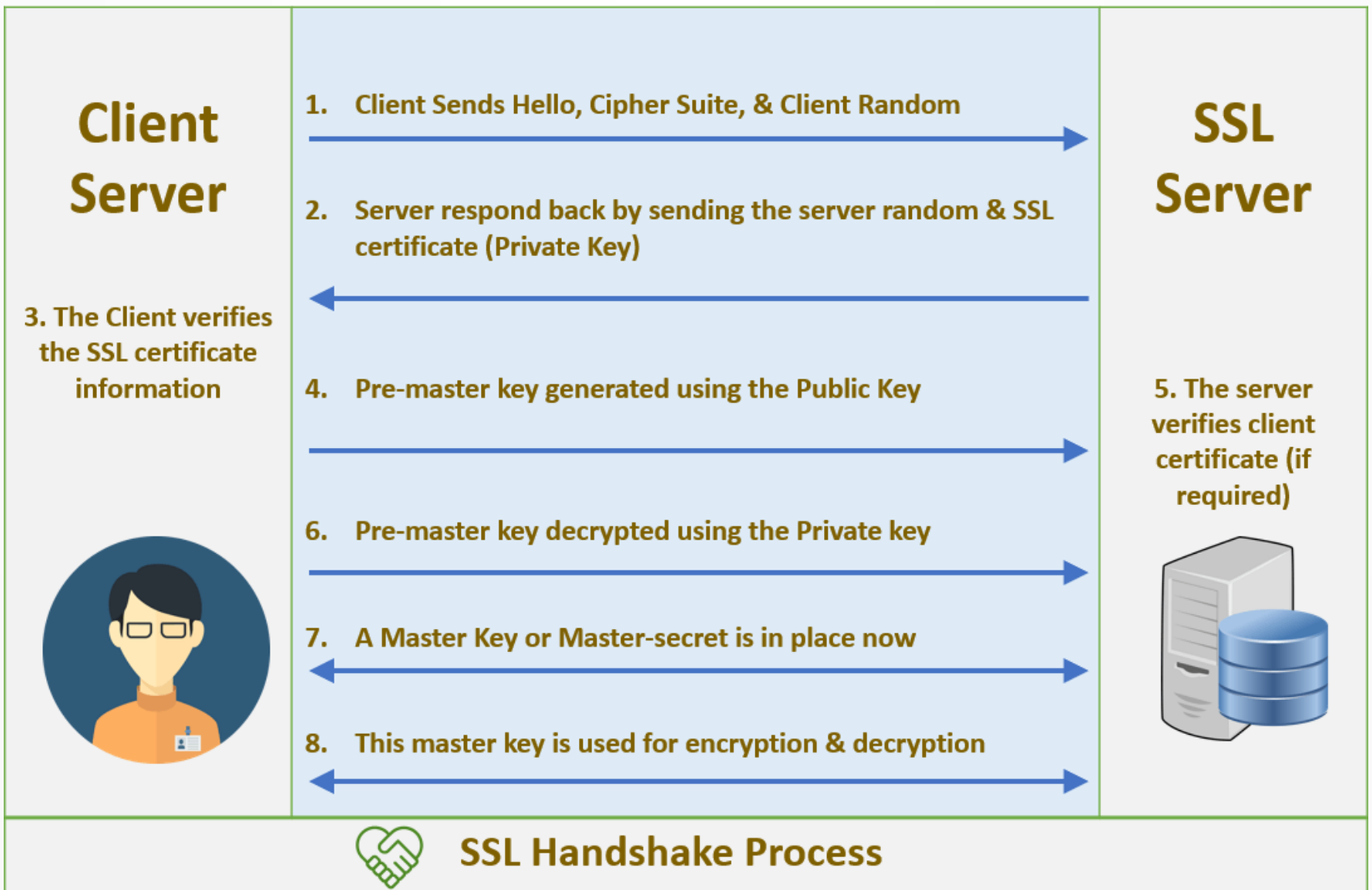
ARP process


I need the MAC address of 172.16.3.2.
I heard that broadcast. That is me. Here is my MAC address.
IP: 172.16.3.2 = ???
IP: 172.16.3.2 = Ethernet: 0800.0200.1111
learncisco
Map IP
Ethernet
Local ARP

Opening of a socket



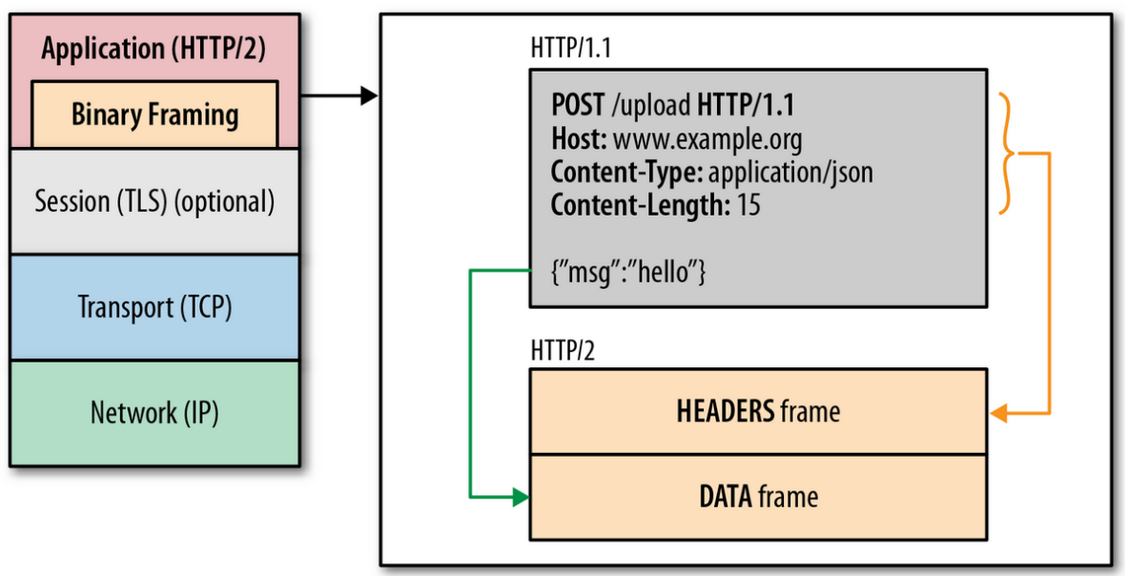
TLS handshake



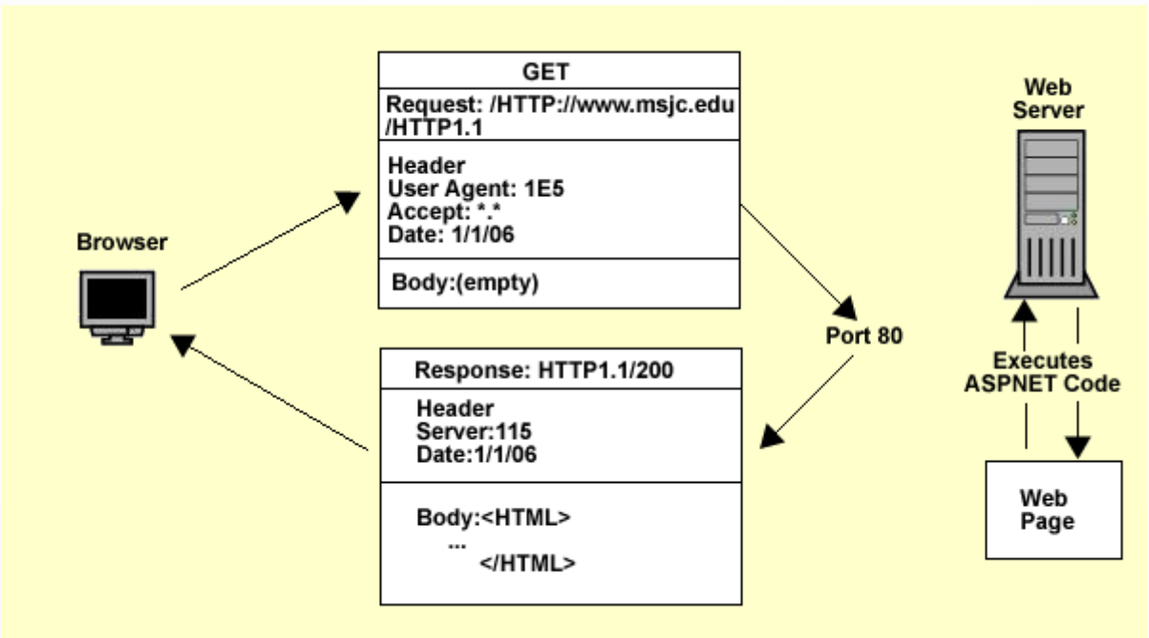
HTTP protocol

HTTP/2 in one slide…

1. **One TCP connection**
2. **Request → Stream**
   - Streams are multiplexed
   - Streams are prioritized
3. **Binary framing layer**
   - Prioritization
   - Flow control
   - Server push
4. **Header compression (HPACK)**



HTTP protocol

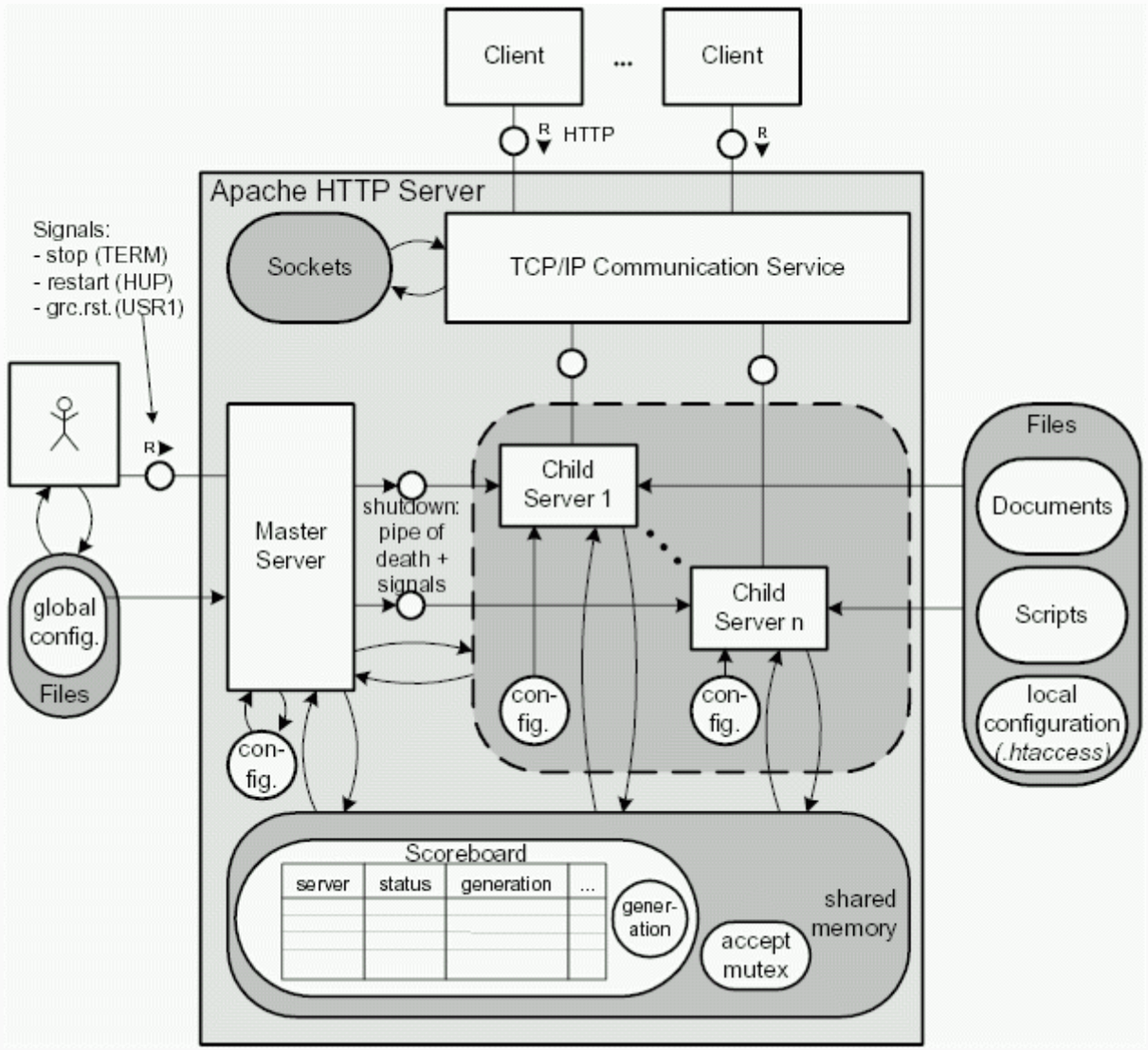


HTTP Server Request Handle

The HTTPD (HTTP Daemon) server is the one handling the requests/responses on the server side. The most common HTTPD servers are Apache or nginx for Linux and IIS for Windows.

- The HTTPD (HTTP Daemon) receives the request.
- The server breaks down the request to the following parameters:
  - HTTP Request Method (either *GET*, *HEAD*, *POST*, *PUT*, *PATCH*, *DELETE*, *CONNECT*, *OPTIONS*, or *TRACE*). In the case of a URL entered directly into the address bar, this will be *GET*.
  - Domain, in this case - google.com.
  - Requested path/page, in this case - / (as no specific path/page was requested, / is the default path).
- The server verifies that there is a Virtual Host configured on the server that corresponds with google.com.
- The server verifies that google.com can accept GET requests.
- The server verifies that the client is allowed to use this method (by IP, authentication, etc.).
- If the server has a rewrite module installed (like mod_rewrite for Apache or URL Rewrite for IIS), it tries to match the request against one of the configured rules. If a matching rule is found, the server uses that rule to rewrite the request.
- The server goes to pull the content that corresponds with the request, in our case it will fall back to the index file, as "/" is the main file (some cases can override this, but this is the most common method).
- The server parses the file according to the handler. If Google is running on PHP, the server uses PHP to interpret the index file, and streams the output to the client.

HTTP Server Request Handle



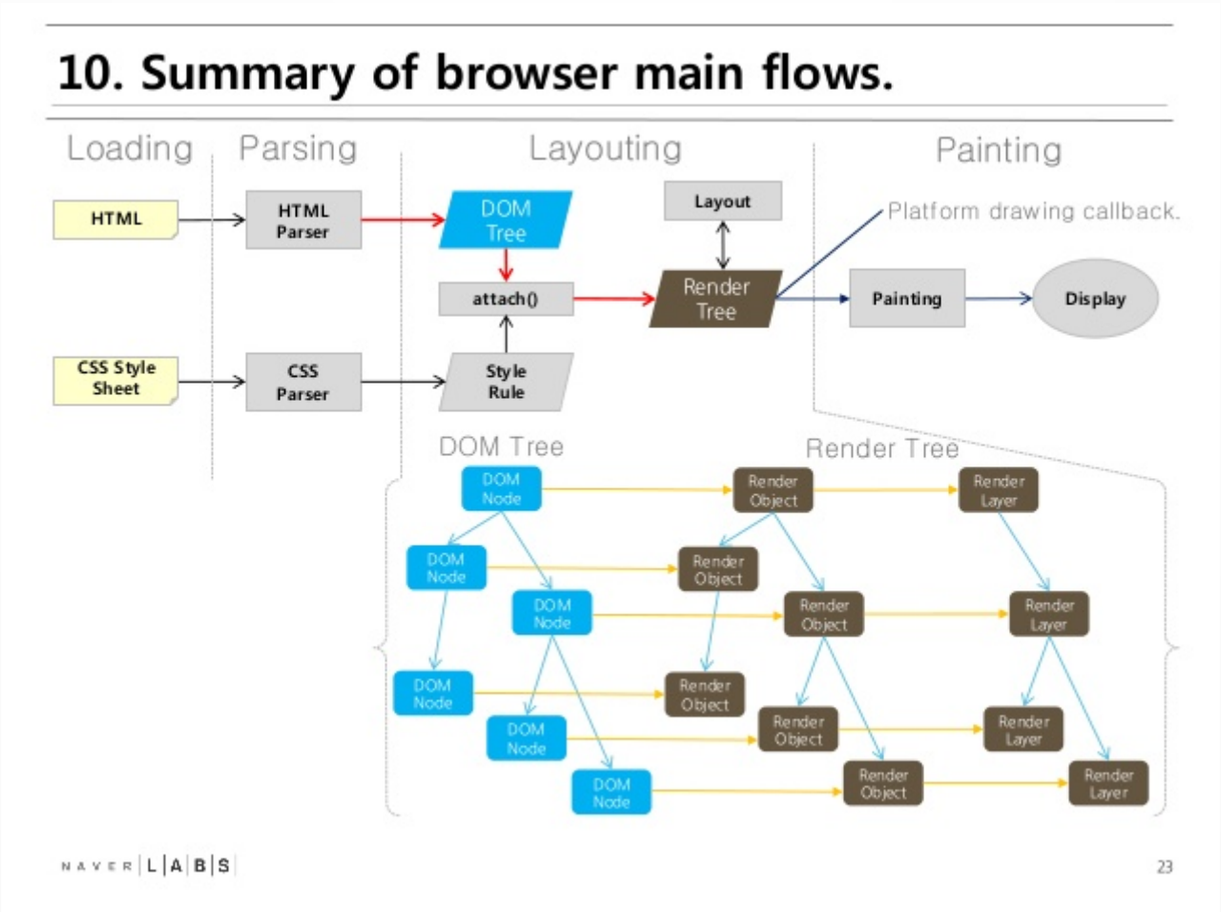
Behind the scenes of the Browser

Once the server supplies the resources (HTML, CSS, JS, images, etc.) to the browser it undergoes the below process:

- Parsing - HTML, CSS, JS
- Rendering - Construct DOM Tree → Render Tree → Layout of Render Tree → Painting the render tree

Behind the scenes of the Browser

Cool presentation



Browser

The browser's functionality is to present the web resource you choose, by requesting it from the server and displaying it in the browser window. The resource is usually an HTML document, but may also be a PDF, image, or some other type of content. The location of the resource is specified by the user using a URI (Uniform Resource Identifier).

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

Browser user interfaces have a lot in common with each other. Among the common user interface elements are:

- An address bar for inserting a URI
- Back and forward buttons
- Bookmarking options
- Refresh and stop buttons for refreshing or stopping the loading of current documents
- Home button that takes you to your home page

Browser High Level Structure

The components of the browsers are:

- **User interface**: The user interface 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.
- **Browser engine**: The browser engine marshals actions between the UI and the rendering engine.
- **Rendering engine**: The rendering engine is 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.
- **Networking**: The networking handles network calls such as HTTP requests, using different implementations for different platforms behind a platform-independent interface.
- **UI backend**: The UI backend is 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 engine**: The JavaScript engine is used to parse and execute JavaScript code.
- **Data storage**: The data storage 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.

HTML parsing

The rendering engine starts getting the contents of the requested document from the networking layer. This will usually be done in 8kB chunks.

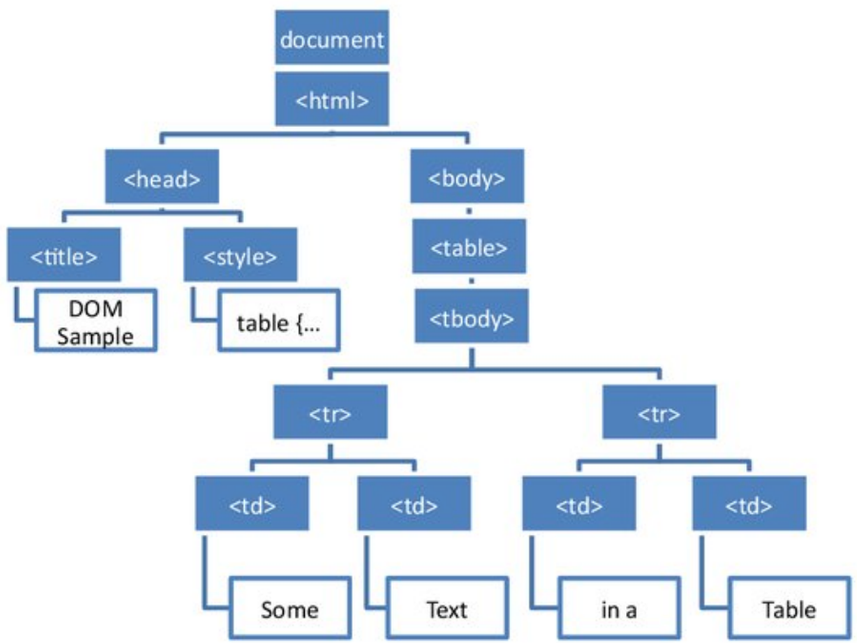
The primary job of HTML parser is to parse the HTML markup into a parse tree.

The output tree (the "parse tree") is a tree of DOM element and attribute nodes. DOM is short for Document Object Model. It is the object presentation of the HTML document and the interface of HTML elements to the outside world like JavaScript. The root of the tree is the "Document" object. Prior of any manipulation via scripting, the DOM has an almost one-to-one relation to the markup.

HTML parsing

DOM Tree


```
<!DOCTYPE html>
<html>
    <head>
        <title>DOM Sample</title>
        <style type="text/css">
            table {
                border: 1px solid black;
            }
        </style>
    </head>
    <body>
        <table>
            <tbody>
                <tr>
                    <td>Some</td>
                    <td>Text</td>
                </tr>
                <tr>
                    <td>in a</td>
                    <td>Table</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```



The parsing algorithm

HTML cannot be parsed using the regular top-down or bottom-up parsers.

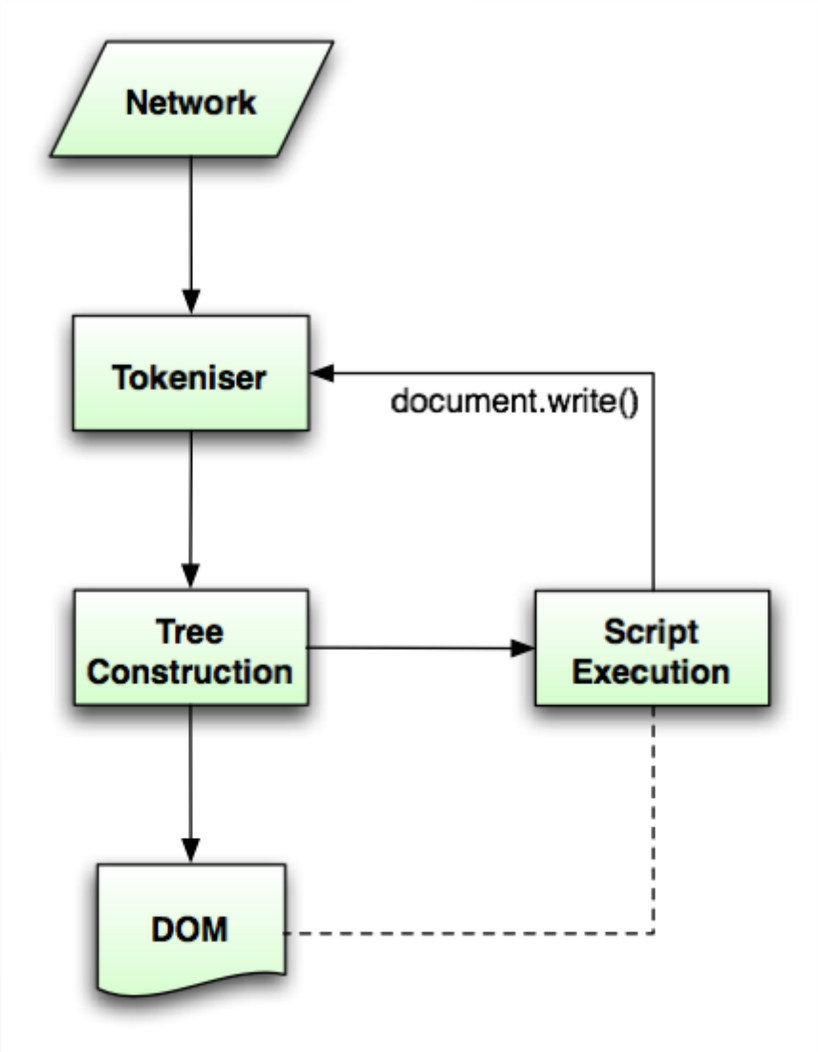
The reasons are:

- The forgiving nature of the language.
- The fact that browsers have traditional error tolerance to support well known cases of invalid HTML.
- The parsing process is reentrant. For other languages, the source doesn't change during parsing, but in HTML, dynamic code (such as script elements containing document.write() calls) can add extra tokens, so the parsing process actually modifies the input.

Unable to use the regular parsing techniques, the browser utilizes a custom parser for parsing HTML. The parsing algorithm is described in detail by the HTML5 specification.

The algorithm consists of two stages: tokenization and tree construction.

The parsing algorithm



Actions when the parsing is finished

The browser begins fetching external resources linked to the page (CSS, images, JavaScript files, etc.).

At this stage the browser marks the document as interactive and starts parsing scripts that are in "deferred" mode: those that should be executed after the document is parsed. The document state is set to "complete" and a "load" event is fired.

Note there is never an "Invalid Syntax" error on an HTML page. Browsers fix any invalid content and go on.

CSS interpretation

- Parse CSS files, <*style*> tag contents, and *style* attribute values using "CSS lexical and syntax grammar"
- Each CSS file is parsed into a *StyleSheet object*, where each object contains CSS rules with selectors and objects corresponding CSS grammar.
- A CSS parser can be top-down or bottom-up when a specific parser generator is used.

Page Rendering

- Create a 'Frame Tree' or 'Render Tree' by traversing the DOM nodes, and calculating the CSS style values for each node.
- Calculate the preferred width of each node in the 'Frame Tree' bottom up by summing the preferred width of the child nodes and the node's horizontal margins, borders, and padding.
- Calculate the actual width of each node top-down by allocating each node's available width to its children.
- Calculate the height of each node bottom-up by applying text wrapping and summing the child node heights and the node's margins, borders, and padding.
- Calculate the coordinates of each node using the information calculated above.
- More complicated steps are taken when elements are *floated*, positioned *absolutely* or r*elatively*, or other complex features are used. See http://dev.w3.org/csswg/css2/ and http://www.w3.org/Style/CSS/current-work for more details.
- Create layers to describe which parts of the page can be animated as a group without being re-rasterized. Each frame/render object is assigned to a layer.
- Textures are allocated for each layer of the page.
- The frame/render objects for each layer are traversed and drawing commands are executed for their respective layer. This may be rasterized by the CPU or drawn on the GPU directly using D2D/SkiaGL.
- All of the above steps may reuse calculated values from the last time the webpage was rendered, so that incremental changes require less work.
- The page layers are sent to the compositing process where they are combined with layers for other visible content like the browser chrome, iframes and addon panels.
- Final layer positions are computed and the composite commands are issued via Direct3D/OpenGL. The GPU command buffer(s) are flushed to the GPU for asynchronous rendering and the frame is sent to the window server.

GPU Rendering

- During the rendering process the graphical computing layers can use general purpose *CPU* or the graphical processor *GPU* as well.
- When using *GPU* for graphical rendering computations the graphical software layers split the task into multiple pieces, so it can take advantage of *GPU* massive parallelism for float point calculations required for the rendering process.

Post-rendering and user-induced execution

After rendering has completed, the browser executes JavaScript code as a result of some timing mechanism (such as a Google Doodle animation) or user interaction (typing a query into the search box and receiving suggestions). Plugins such as Flash or Java may execute as well, although not at this time on the Google homepage. Scripts can cause additional network requests to be performed, as well as modify the page or its layout, causing another round of page rendering and painting.

The end.