Web programming fundamentals

Internet ???

- The Internet, sometimes called simply "the Net," is a worldwide system of computer networks --
- a network of networks
- in which users at any one computer can, if they have permission, get information from any other computer (and sometimes talk directly to users at other computers).

Internet

- The Internet is a global network of networks connecting millions of users worldwide via many computer networks using a simple standard common addressing system and basic communications protocol called TCP/IP (Transmission Control Protocol/Internet Protocol).

- Online resources have become a part of our day to day life.
- The internet is a powerful media to transmit information.
- The pages of information displayed on the internet are referred to as webpages.
- The standards and formats for presenting text and graphics on the internet are developed and approved by WWW governing authorities.
- The growing demand for attractive presentation of information using electronic means gave rise to the invention of websites.

Components of web

- Web uses the following Terms :
- Webpage : A simple text file created using HTML.
- Website : A collection of interlinked web pages containing text, images, audio and videos.
- For Example, www. google.com
- Web Browser : A web browser is a software used to view web pages or websites available on the internet For Example Internet Explorer, Google Chrome, Mozilla Firefox.
- Web Server : A Web server is an application or a computer that sends webpages over the internet using the HTTP protocol.

- The functionality of website is managed by web server.
- For Example Apache, nginx, IIS, etc..
- **URL(Uniform Resource Locator) :** It is an address of a web page on the internet. The web pages are retrieved from the original location with the help of URL.
- **HTTP : HTTP (HyperText Transfer** Protocol) is a protocol used by WWW for client server communication.
- **HTML : Hyper Text Markup** Language, enables to write code for a webpage.
- All the webpages in a website are linked with one another, with the help of hypertext links.

Working of web browser

- Web browsers are used to make it easy to access the World Wide Web.
- Browsers are able to display Web pages largely in part to an underlying Web protocol called HyperText Transfer Protocol (HTTP).
- HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
- It is what allows Web clients and Web servers to communicate with each other.
- When you enter a Web address (URL) in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page and display the information in your browser.
- All Web servers serving Web sites and pages support the HTTP protocol.

- EXAMPLE: THE URL TO REACH THE DEFINITION OF BROWSER ON WEBOPEDIA IS: HTTPS://WWW.WEBOPEDIA.COM/BROWSER.HTML
- ONCE YOU ENTER THE URL INTO YOUR ADDRESS LINE, THE BROWSER BREAKS THAT WEB ADDRESS DOWN INTO THREE DISTINCT PARTS.
- The Protocol: “http”
- The server name: “www.webopedia.com”
- The file name, which follows the server name: “browser.html”

- IN ORDER FOR YOUR BROWSER TO ACTUALLY CONNECT TO THE WEB SERVER TO RETRIEVE THE INFORMATION YOU REQUEST, IT COMMUNICATES WITH A NAME SERVER TO TRANSLATE THE SERVER NAME INTO AN IP ADDRESS.

- YOUR WEB BROWSER IS THEN ABLE TO CONNECT TO THE WEB SERVER AT THE RESOLVED IP ADDRESS ON PORT 80.
- (**HTTP** uses **port** 80 and HTTPS uses **port** 443)
- ONCE YOUR BROWSER HAS CONNECTED TO THE WEB SERVER USING HTTP, THE BROWSER THEN READS THE HYPERTEXT MARKUP LANGUAGE (HTML), THE AUTHORING LANGUAGE USED TO CREATE DOCUMENTS ON THE WORLD WIDE WEB, AND THE DATA IS THEN DISPLAYED IN YOUR WEB BROWSER.

Web browser

- A Web browser is actually a software application that runs on your Internet-connected computer.
- It allows you to view Web pages, as well as use other content and technologies such as video, graphics files, and digital certificates, to name a few.
- Some browsers will translate only text while others do support graphics and animation.
- Web browsers are not all created equal, and Web pages also will not be displayed the same in different browsers.

- There are five major browsers used on desktop today: Chrome, Internet Explorer, Firefox, Safari and Opera.
- On mobile, the main browsers are Android Browser, iPhone, Opera Mini and Opera Mobile, UC Browser, the Nokia S40/S60 browsers and Chrome–all of which, except for the Opera browsers, are based on WebKit.

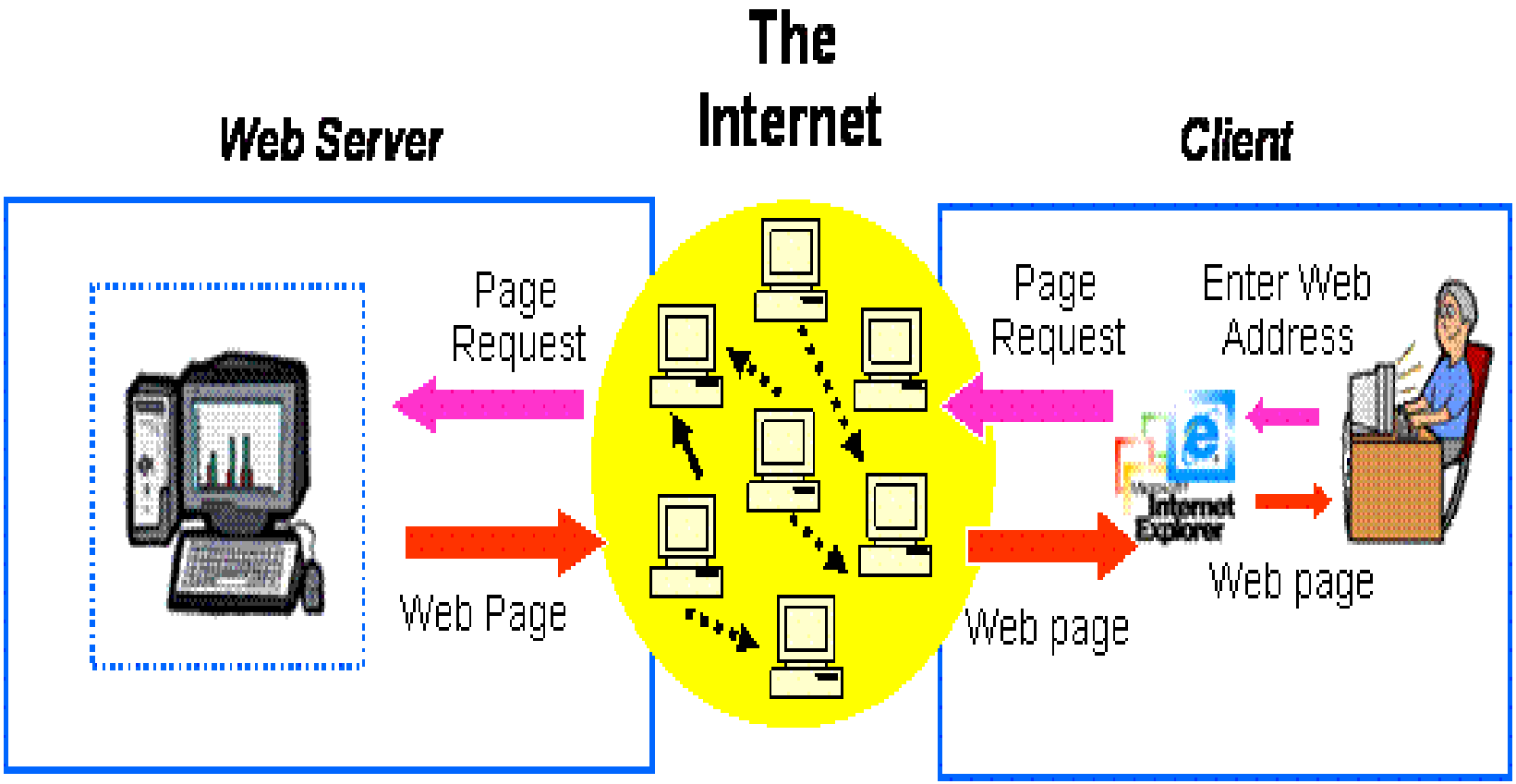
Web System architecture

- **Client Server and 2 tier Web Architectures**
- **3-Tier architectures**
- **n-Tier Architectures**

Client Server and 2 tier Web Architectures

- Typically, when you are browsing the Internet, you will be using *Web Browser* software such as Internet Explorer or Mozilla Firefox.
- The computer which is running a browser is called a ***client***, while the machine which is providing Web pages is called a ***server***.
- The web server, as the name suggests, serves your browser with Web pages (e.g. HTML, ASPX, JSP pages etc).
- This simple scenario, where the Web server is connected to one or more clients is known as a 2 tier architecture model.

A simple diagram of 2 tier Client/server architecture



3-Tier architectures

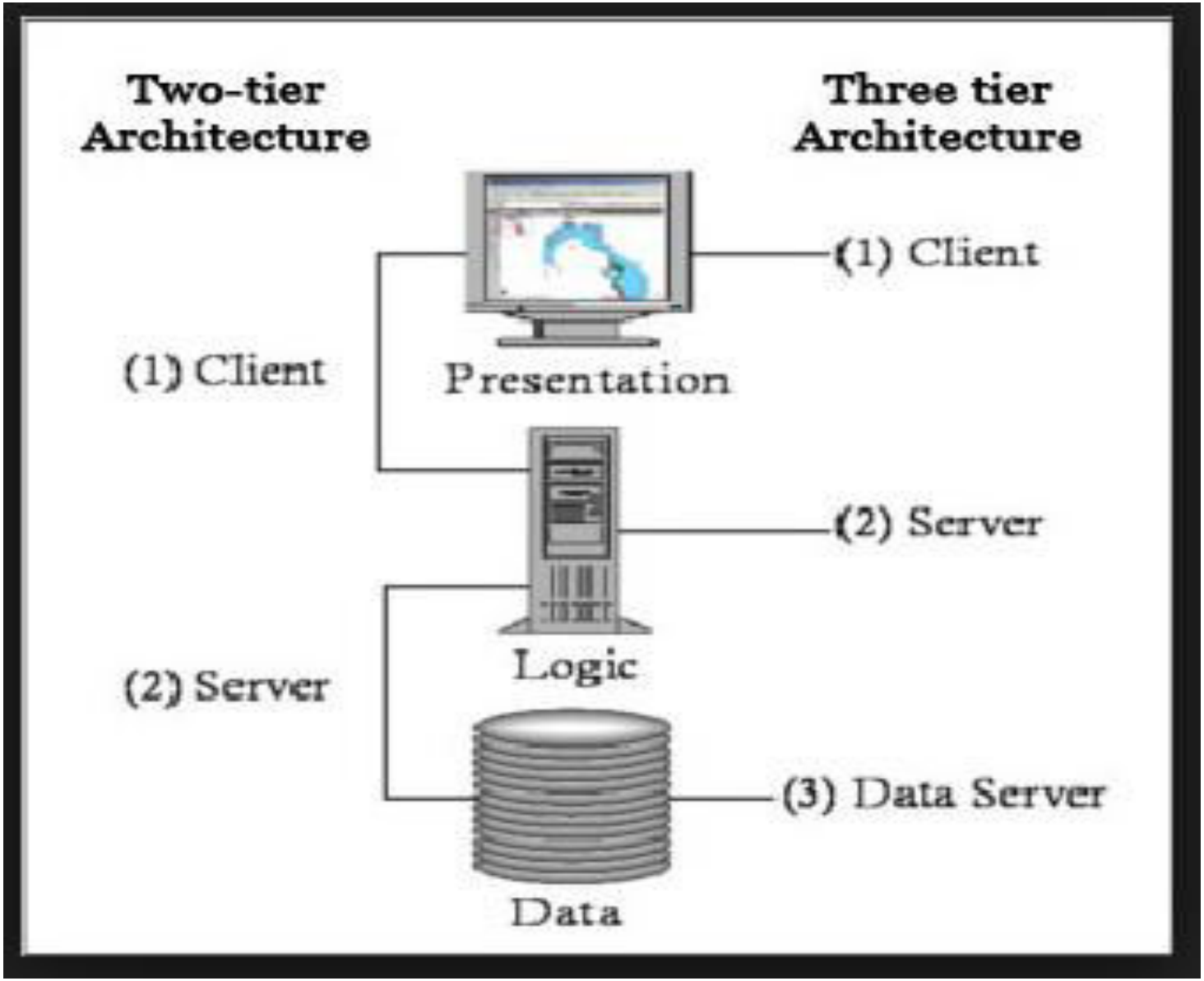
- Generally computing applications consist of three different and distinct types of functionalities.
- Presentation Services *(Client)*
- Functional Logic *(Server)*
- Data Management
- **Presentation Services**: These manifest themselves in the form of information display and user data input facilities.
- Generally the front-end for user interaction.
- For example logging in requires interaction in the form of collecting username and password information using a HTML-form.

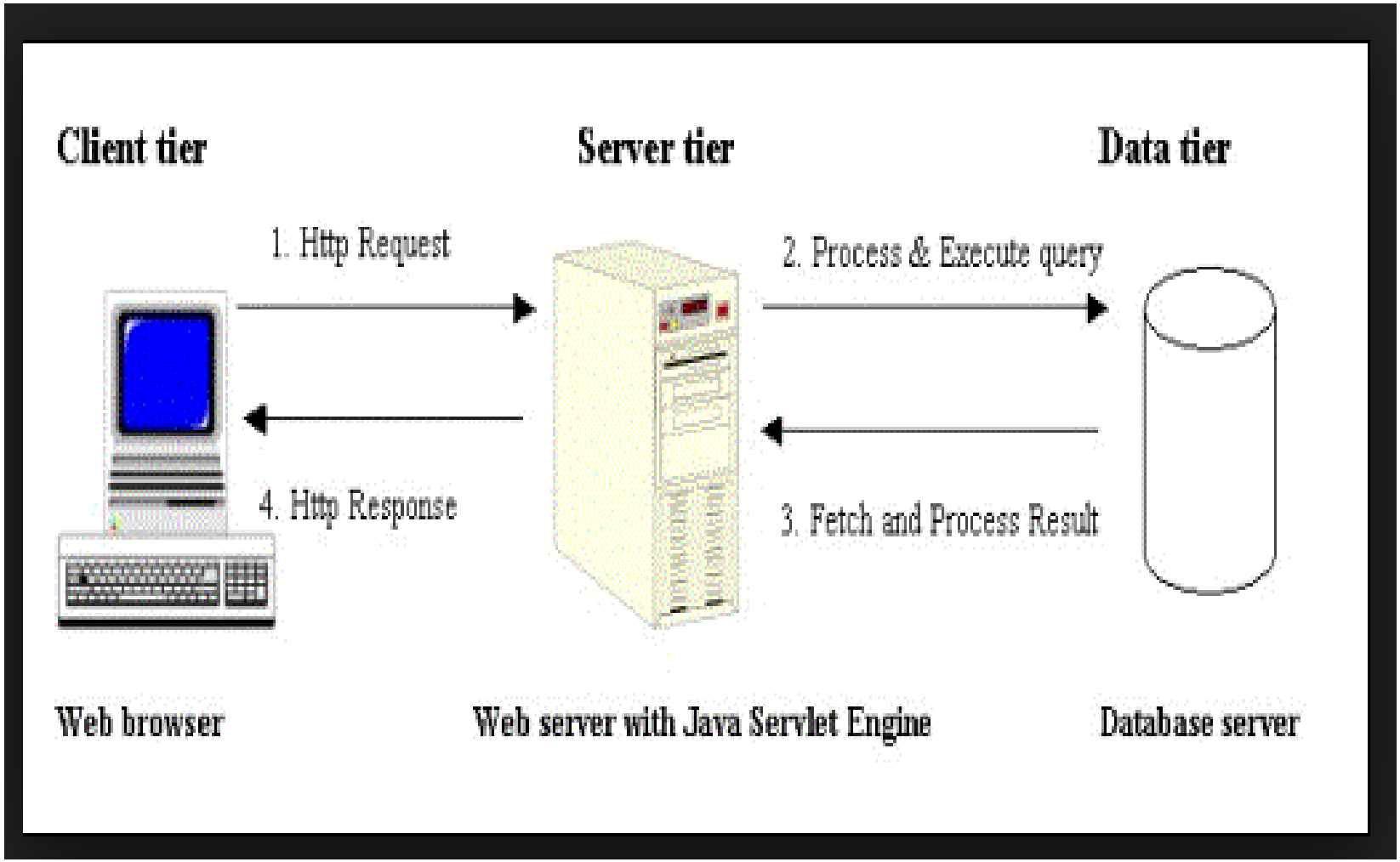
3-Tier architectures

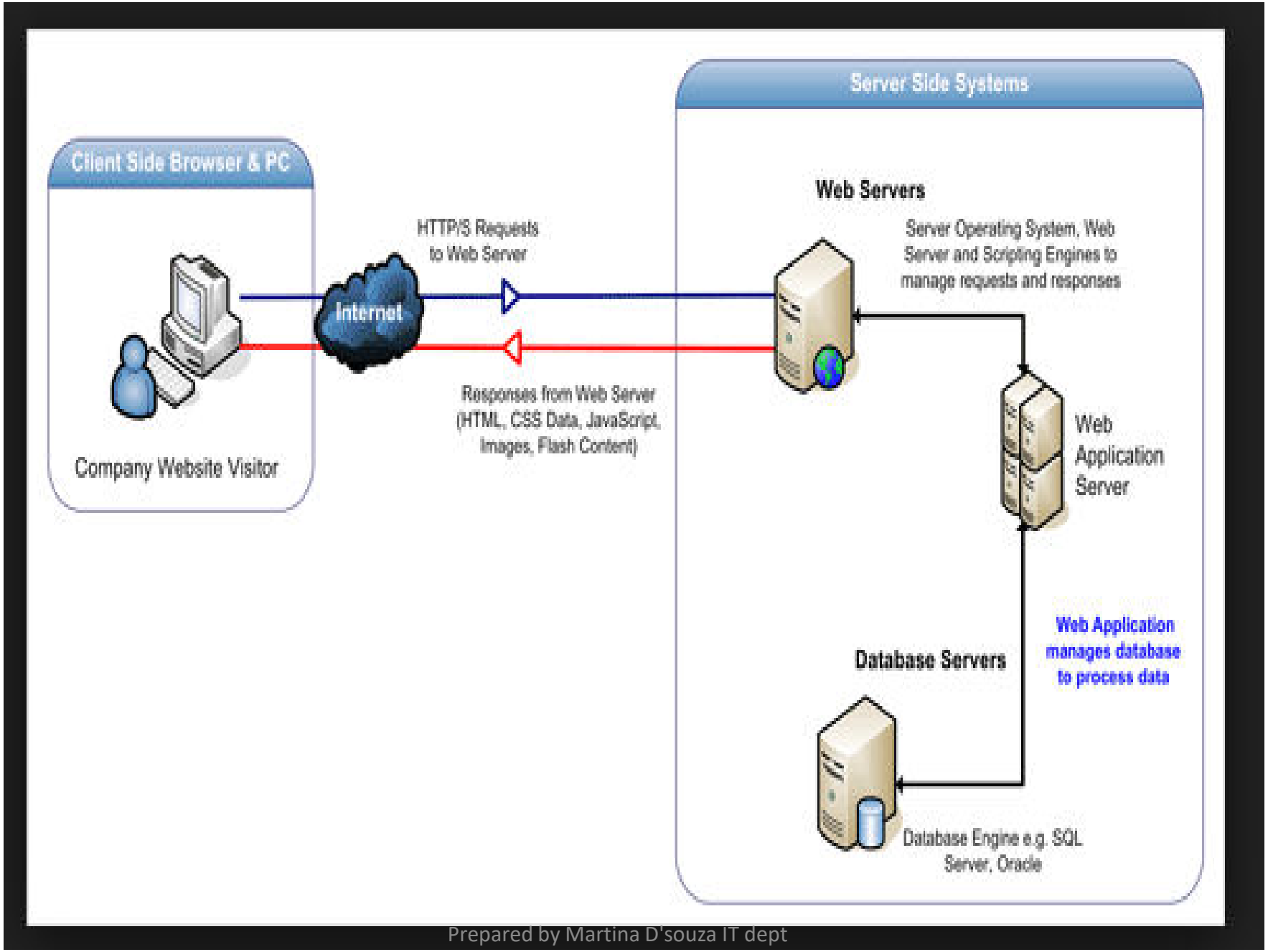
- **Functional logic**: Every application includes some data processing and this may also involve database interactivity. For example user authentication requires the logic unit to read username-password combinations from a database and compare until a good comparison (hopefully) is arrived at.

- **Data Management**: Data, its storage, insertion and retrieval, its management and alteration is central to computing applications. For example a database management system (DBMS) is required for the management of usernames and associated passwords, their owners, etc.

3 tier architecture model




Client tier
Server tier
Data tier
1. Http Request
2. Process & Execute query
4. Http Response
3. Fetch and Process Result
Web browser
Web server with Java Servlet Engine
Database server


Server Side Systems
Client Side Browser & PC
Web Servers
HTTP/S Requests to Web Server
Server Operating System, Web Server and Scripting Engines to manage requests and responses
Internet
Responses from Web Server (HTML, CSS Data, JavaScript, Images, Flash Content)
Company Website Visitor
Web Application Server
Web Application manages database to process data
Database Servers
Database Engine e.g. SQL Server, Oracle

n-Tier Architectures

- Note!! Web application architecture sometimes is referred to as multi-tier.
- In effect we could have more than 3-tiers in circumstances when the Web server requires to access one or more application servers for specialised services. This is known as n-Tier architecture
- Each layer within an N-tier architecture could be thought of as 'logical components' interacting with the layer above or below.
- Layers provide a means of grouping functionality within the application structure.

n-Tier Architectures

- Some benefits of this approach include:
- Flexibility of component location - each layer may be held on a different server, this facilitates scalable applications capable of handling heavier server loads.
- Additionally each layer is encapsulated making it possible to change one layer without affecting another.

Three-tier Architecture

- Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers:
- the presentation tier, or user interface;
- the application tier, where data is processed;
- and the data tier, where the data associated with the application is stored and managed.

Presentation tier

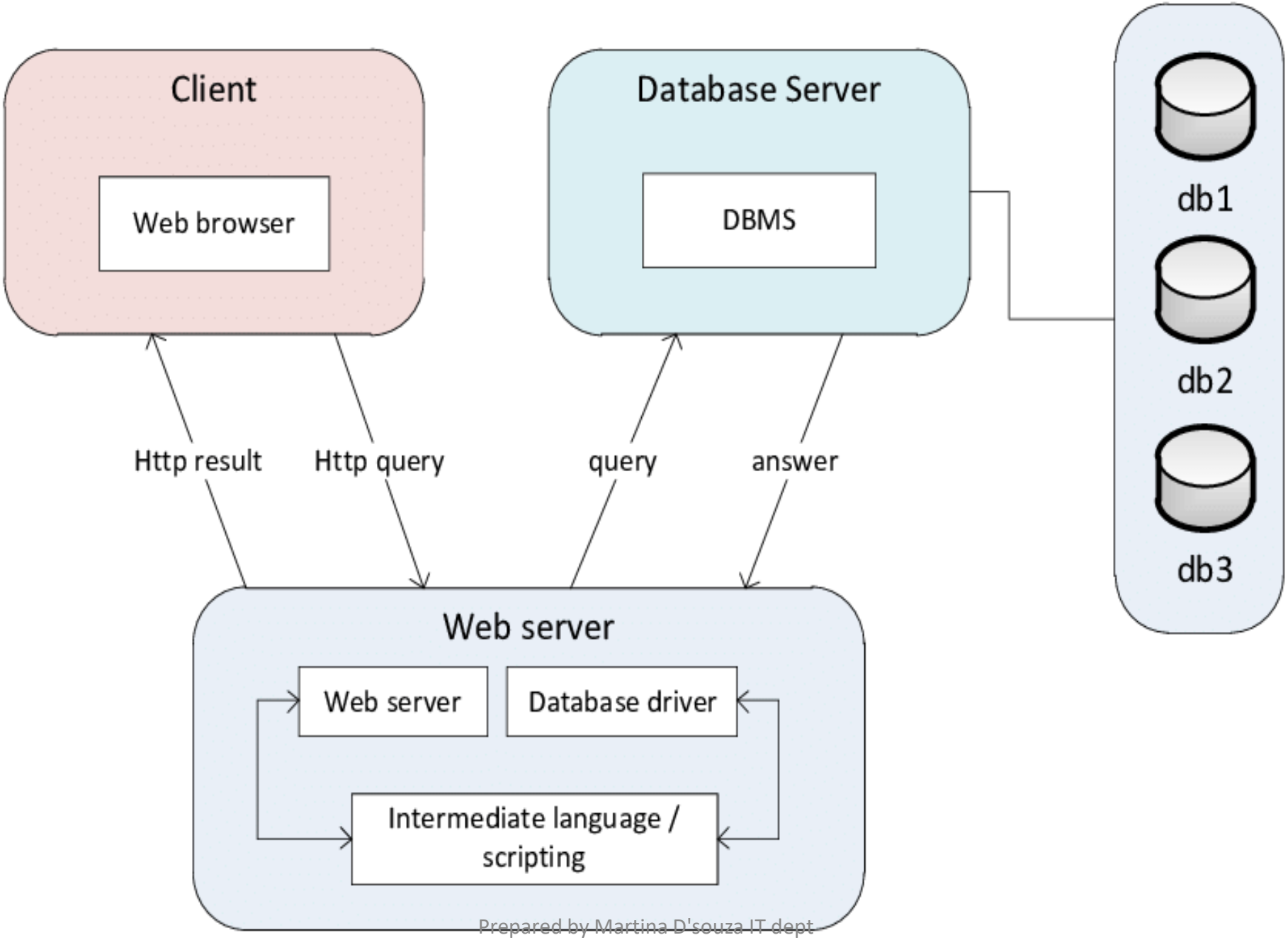
- The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application.
- Its main purpose is to display information to and collect information from the user.
- This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI).
- Web presentation tiers are usually developed using HTML, CSS and JavaScript.
- Desktop applications can be written in a variety of languages depending on the

Application tier

- The application tier, also known as the logic tier or middle tier, is the heart of the application.
- In this tier, information collected in the presentation tier is processed - sometimes against other information in the data tier - using business logic, a specific set of business rules.
- The application tier can also add, delete or modify data in the data tier.
- The application tier is typically developed using Python, Java, Perl, PHP or Ruby, and

Data tier

- The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed.
- This can be a relational database management system such as PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix or Microsoft SQL Server, or in a NoSQL Database server such as Cassandra, CouchDB or MongoDB.
- In a three-tier application, all communication goes through the


Client
Web browser
Database Server
DBMS
db1
db2
db3
Http result
Http query
query
answer
Web server
Web server
Database driver
Intermediate language / scripting

In web development

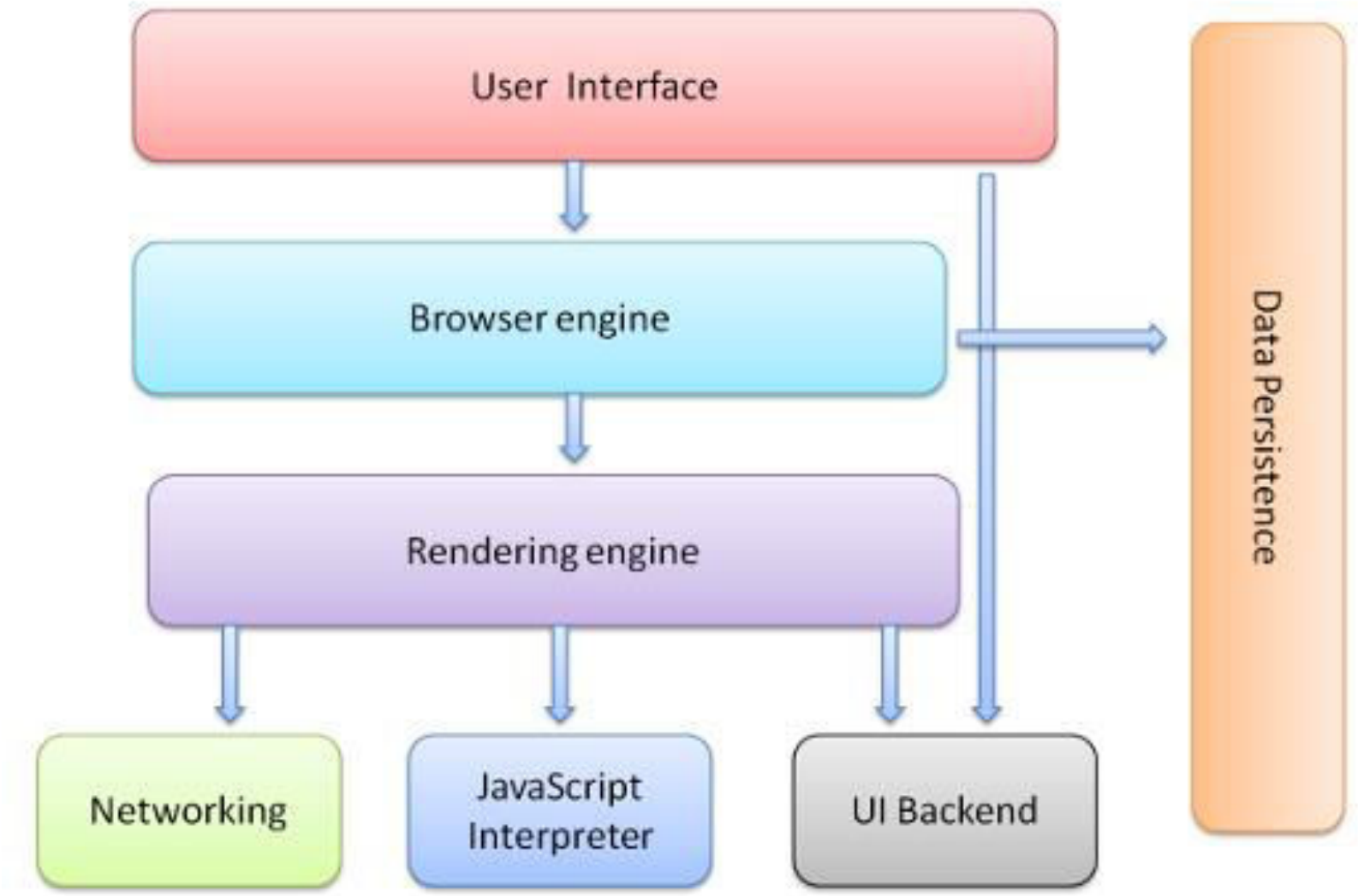
- the tiers have different names but perform similar functions:
- The **web server** is the presentation tier and provides the user interface.
- This is usually a web page or web site, such as an ecommerce site where the user adds products to the shopping cart, adds payment details or creates an account.
- The content can be static or dynamic, and is usually developed using HTML, CSS and Javascript .

- The **application server** corresponds to the middle tier, housing the business logic used to process user inputs.
- To continue the ecommerce example, this is the tier that queries the inventory database to return product availability, or adds details to a customer's profile.
- This layer often developed using Python, Ruby or PHP and runs a framework such as e Django, Rails, Symphony or ASP.NET, for example.
- The **database server** is the data or backend tier of a web application.
- It runs on database management software,

Benefits

- **Faster development**: Because each tier can be developed simultaneously by different teams, an organization can bring the application to market faster, and programmers can use the latest and best languages and tools for each tier.
- **Improved scalability**: Any tier can be scaled independently of the others as needed.
- **Improved reliability**: An outage in one tier is less likely to impact the availability or performance of the other tiers.
- **Improved security**: Because the presentation

Browser Components



The browser's main components are

- **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.

Browser Components

- **UI backend:** used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific.
- **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.

Rendering Engine

- 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 main flow

- 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:

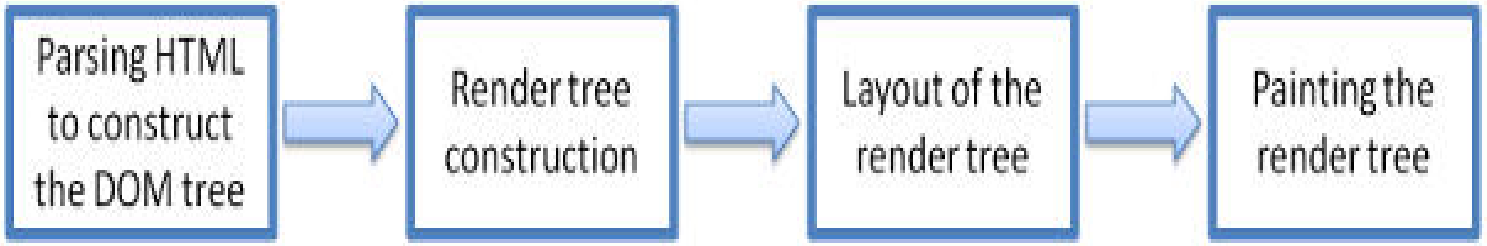
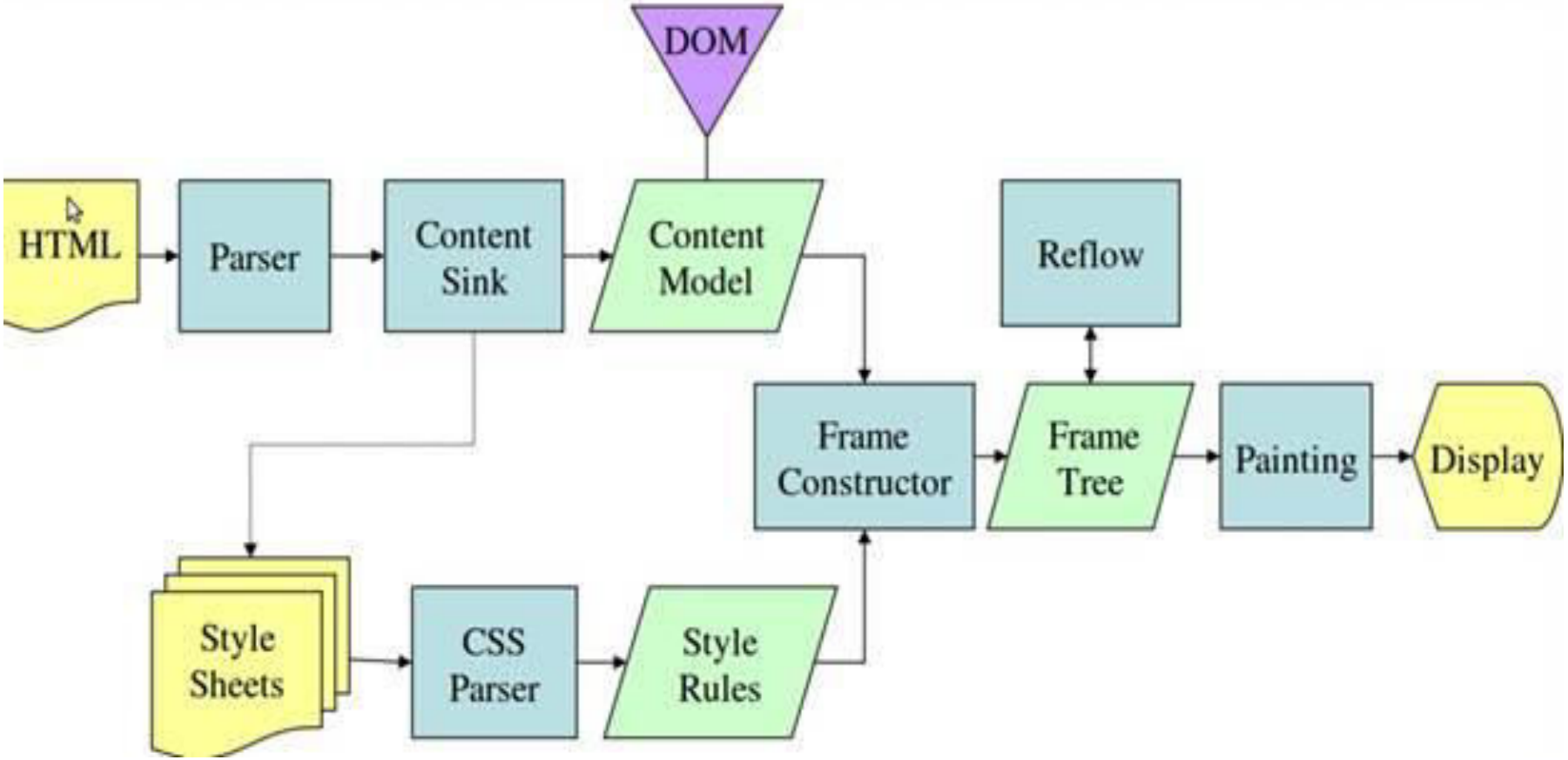


Figure : Rendering engine basic flow



Rendering process in web browsers.

Networking

- Retrieves the URLs using the common internet protocols of HTTP or FTP.
- Handles all aspects of Internet communication and security.
- May implement a cache of retrieved documents in order to reduce network traffic.

Java Script Interpreter

- Interprets and executes the java script code embedded in a website.
- The interpreted results are sent to the rendering engine for display.

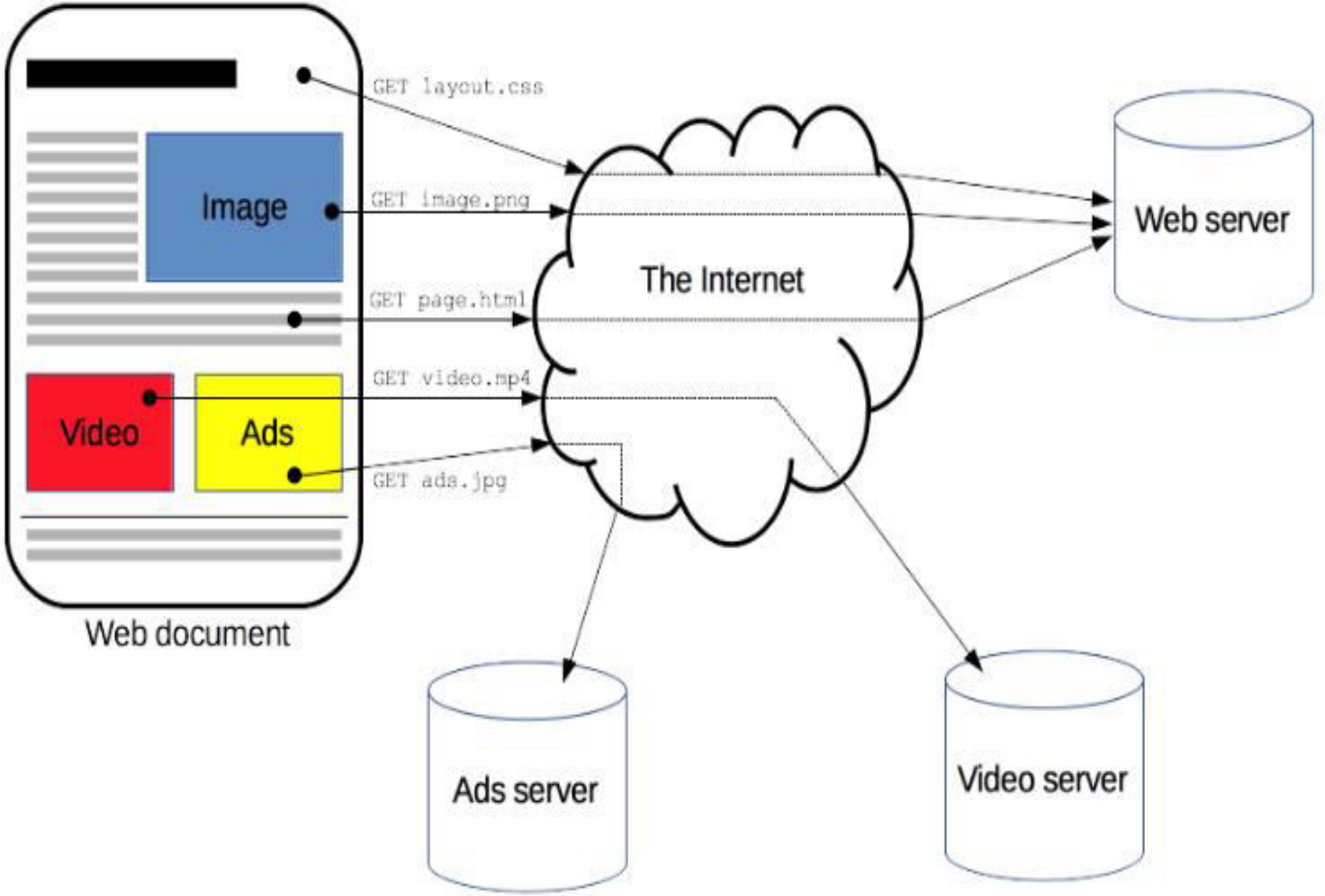
Data Persistence/Storage:

- Browsers support storage mechanisms such as
  - localStorage
  - IndexedDB
  - WebSQL
  - FileSystem

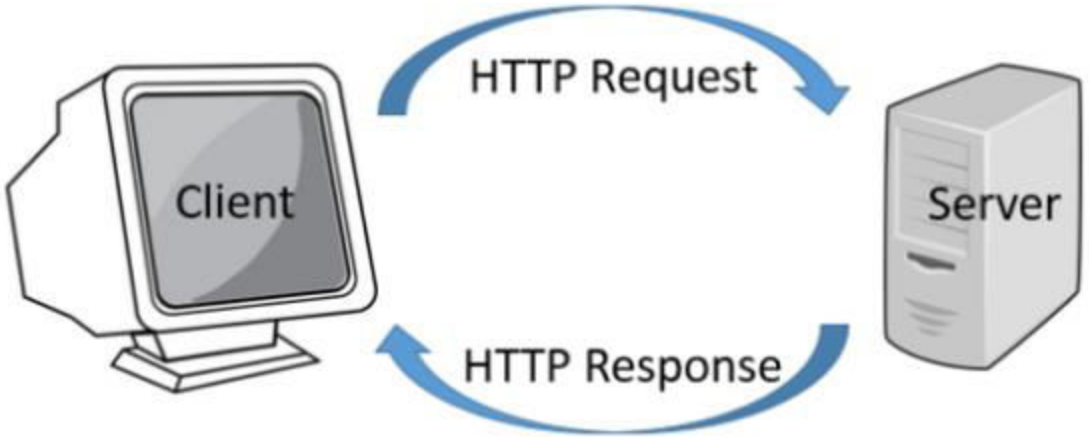
- It is a small database created on the local drive of the computer where the browser is installed.
- It manages user data such as cache, cookies, bookmarks and preferences.

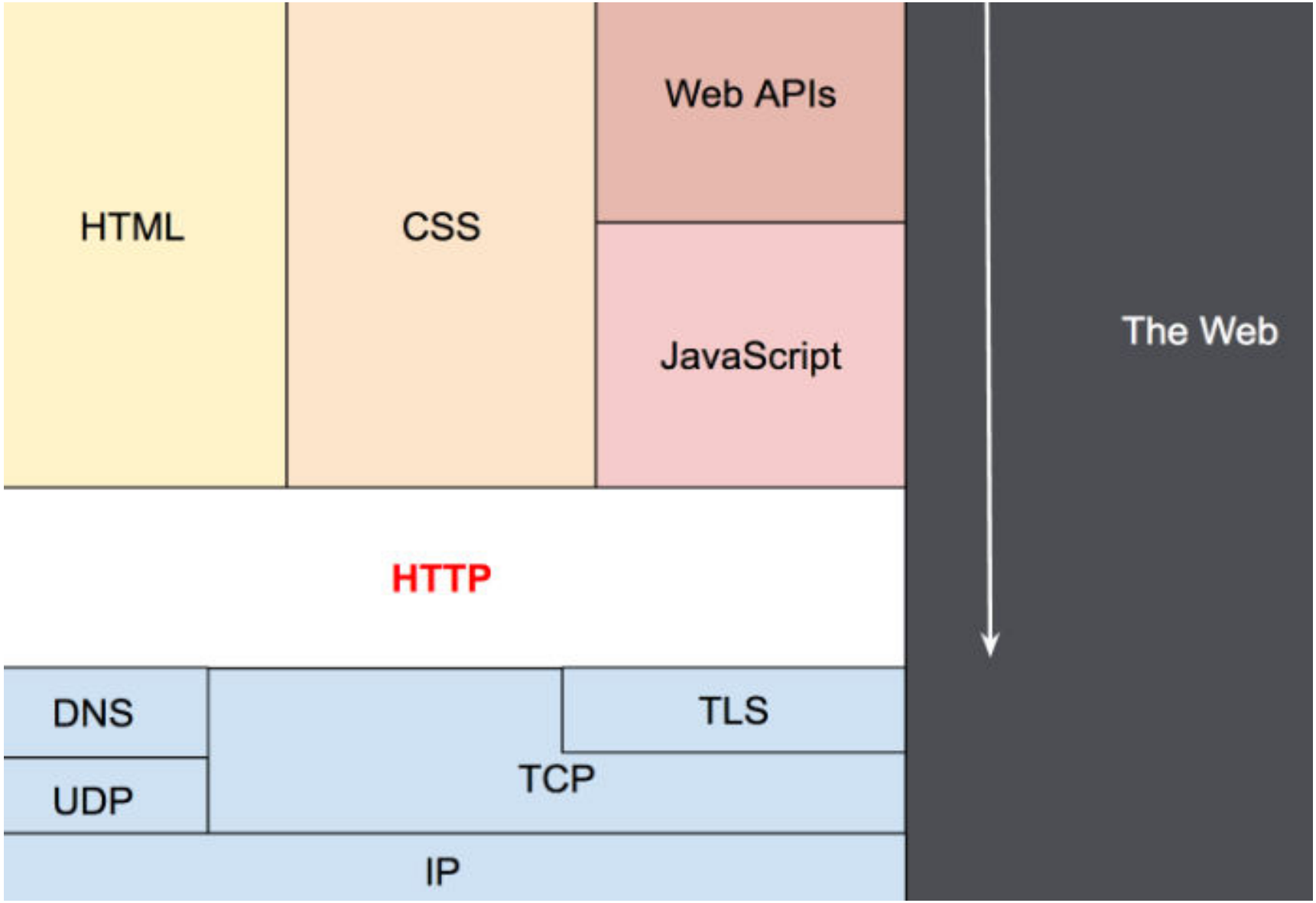
HTTP protocol

- **HTTP** is a protocol which allows the fetching of resources, such as HTML documents.
- It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser.
- A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.


GET layout.css
Image
GET image.png
The Internet
Web server
GET page.html
GET video.mp4
Video
Ads
GET ads.jpg
Web document
Ads server
Video server

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
- The messages sent by the client, usually a Web browser, are called *requests* and the messages sent by the server as an answer are called *responses*.



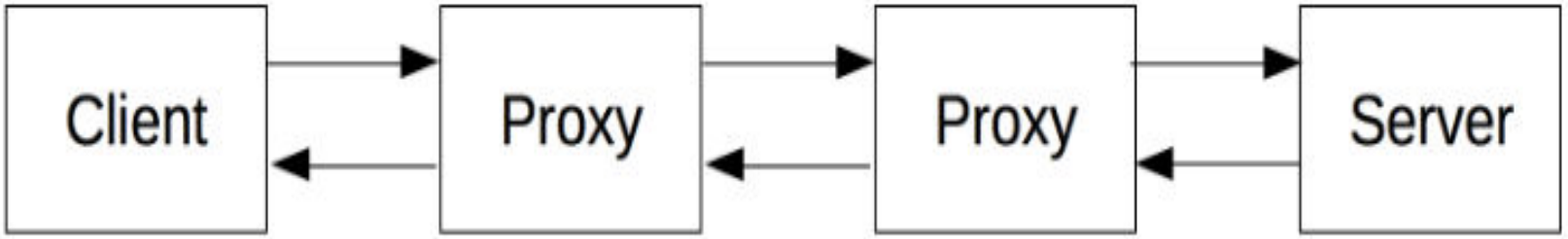

HTML
CSS
Web APIs
JavaScript
The Web
HTTP
DNS
UDP
TCP
TLS
IP

HTTP

- It is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection.
- Due to its extensibility, it is used to not only fetch hypertext documents, but also images and videos or to post content to servers, like with HTML form results.
- HTTP can also be used to fetch parts of documents to update Web pages on demand.

Components of HTTP-based systems

- HTTP is a client-server protocol:
- requests are sent by one entity, the user-agent (or a proxy on behalf of it).
- Most of the time the user-agent is a Web browser, but it can be anything, for example a robot that crawls the Web to populate and maintain a search engine index.
- Each individual request is sent to a server, which handles it and provides an answer, called the *response*.
- Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches,



In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more.

Due to the layered design of the Web, these are hidden in the network and transport layers.

HTTP is on top, at the application layer.

Client: the user-agent

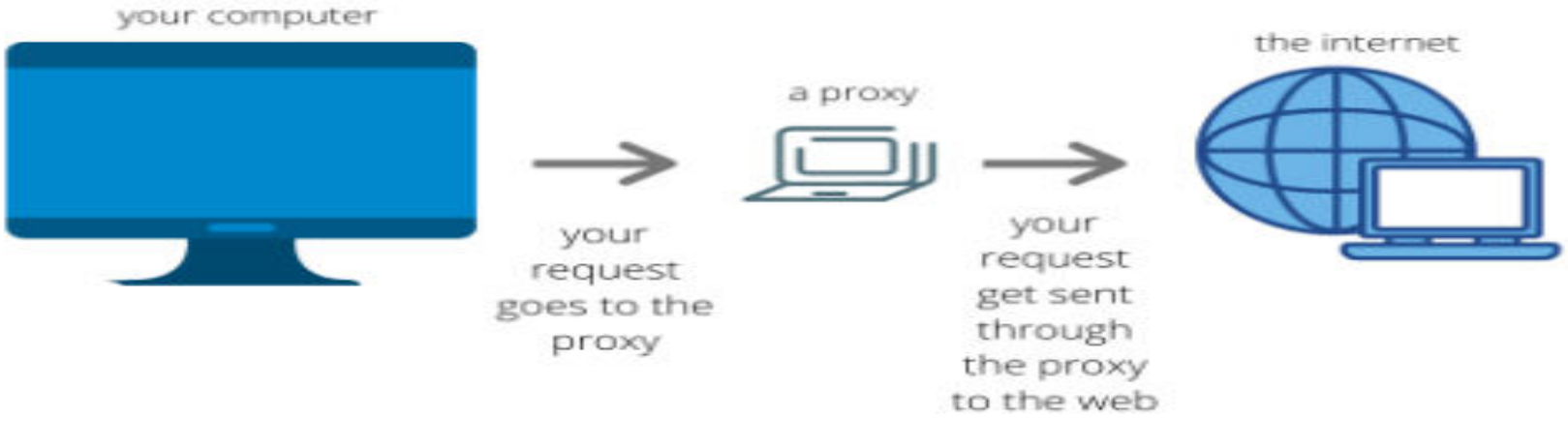
- The *user-agent* is any tool that acts on the behalf of the user.
- This role is primarily performed by the Web browser;
- other possibilities are programs used by engineers and Web developers to debug their applications.
- The browser is **always** the entity initiating the request.

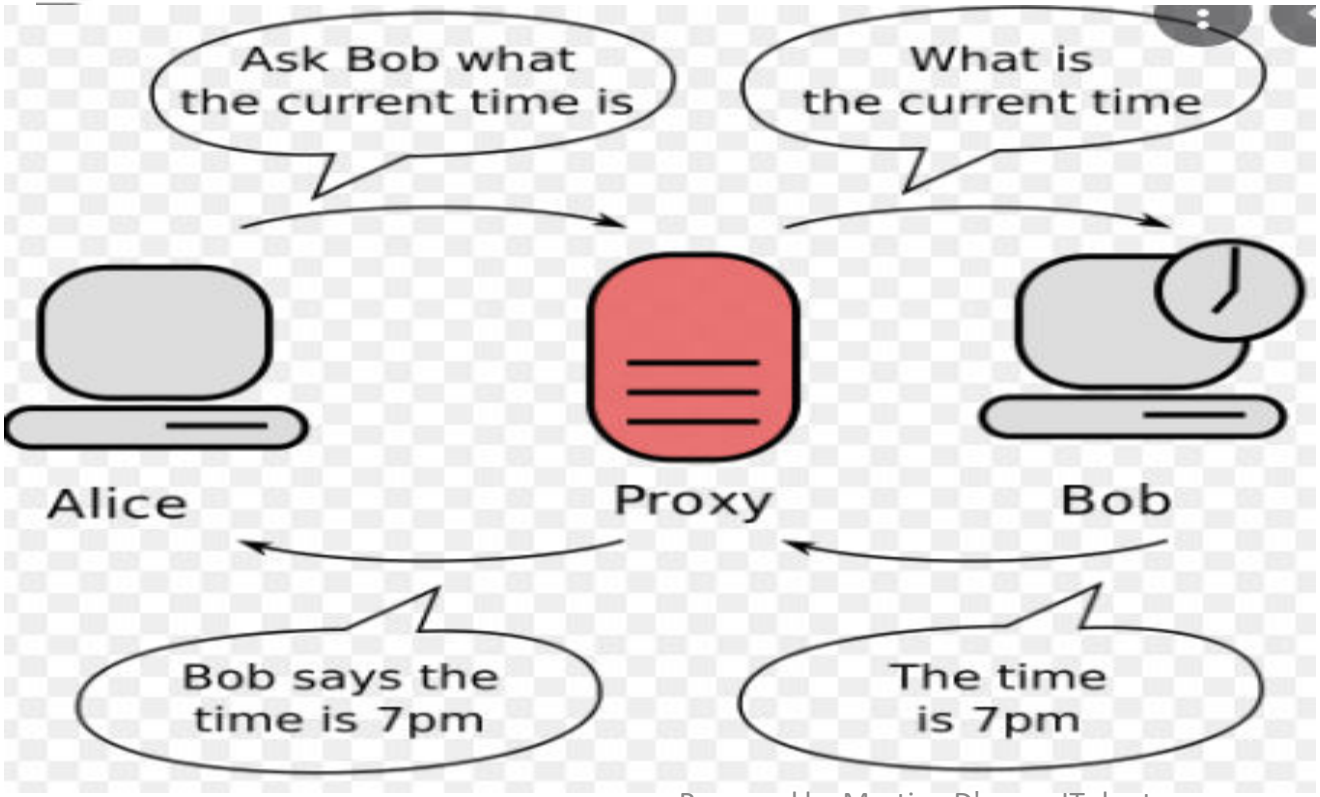
The Web server

- *serves* the document as requested by the client.
- A server appears as only a single machine virtually: this is because it may actually be a collection of servers, sharing the load (load balancing) or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.
- A server is not necessarily a single machine, but several server software instances can be hosted on the same machine.
- With HTTP/1.1 and the Host header, they may even share the same IP address.

Proxies

- Between the Web browser and the server, numerous computers and machines relay the HTTP messages.
- Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially making a significant impact on performance.
- Those operating at the application layers are generally called **proxies**.
- These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server.


your computer
your request goes to the proxy
a proxy
your request get sent through the proxy to the web
the internet


Ask Bob what the current time is
What is the current time
Alice
Proxy
Bob
Bob says the time is 7pm
The time is 7pm

Role of Proxy servers

- Proxy servers act as a firewall and web filter, provide shared network connections, and cache data to speed up common requests.
- A good proxy server keeps users and the internal network protected from the bad stuff that lives out in the wild internet.
- Lastly, proxy servers can provide a high level of privacy.

Proxies

- Proxies may perform numerous functions:
- caching (the cache can be public or private, like the browser cache)
- filtering (like an antivirus scan or parental controls)
- load balancing (to allow multiple servers to serve the different requests)
- authentication (to control access to different resources)
- logging (allowing the storage of historical information)

Features of HTTP

- HTTP is simple
- HTTP messages can be read and understood by humans, providing easier testing for developers, and reduced complexity for newcomers.

- HTTP is extensible
- Introduced in HTTP/1.0, HTTP headers make this protocol easy to extend and experiment with. New functionality can even be introduced by a simple agreement between a client and a server about a new header's semantics.

- HTTP is stateless, but not sessionless
- HTTP is stateless: there is no link between two requests being successively carried out on the same connection.
- This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets.
- But while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions.
- Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

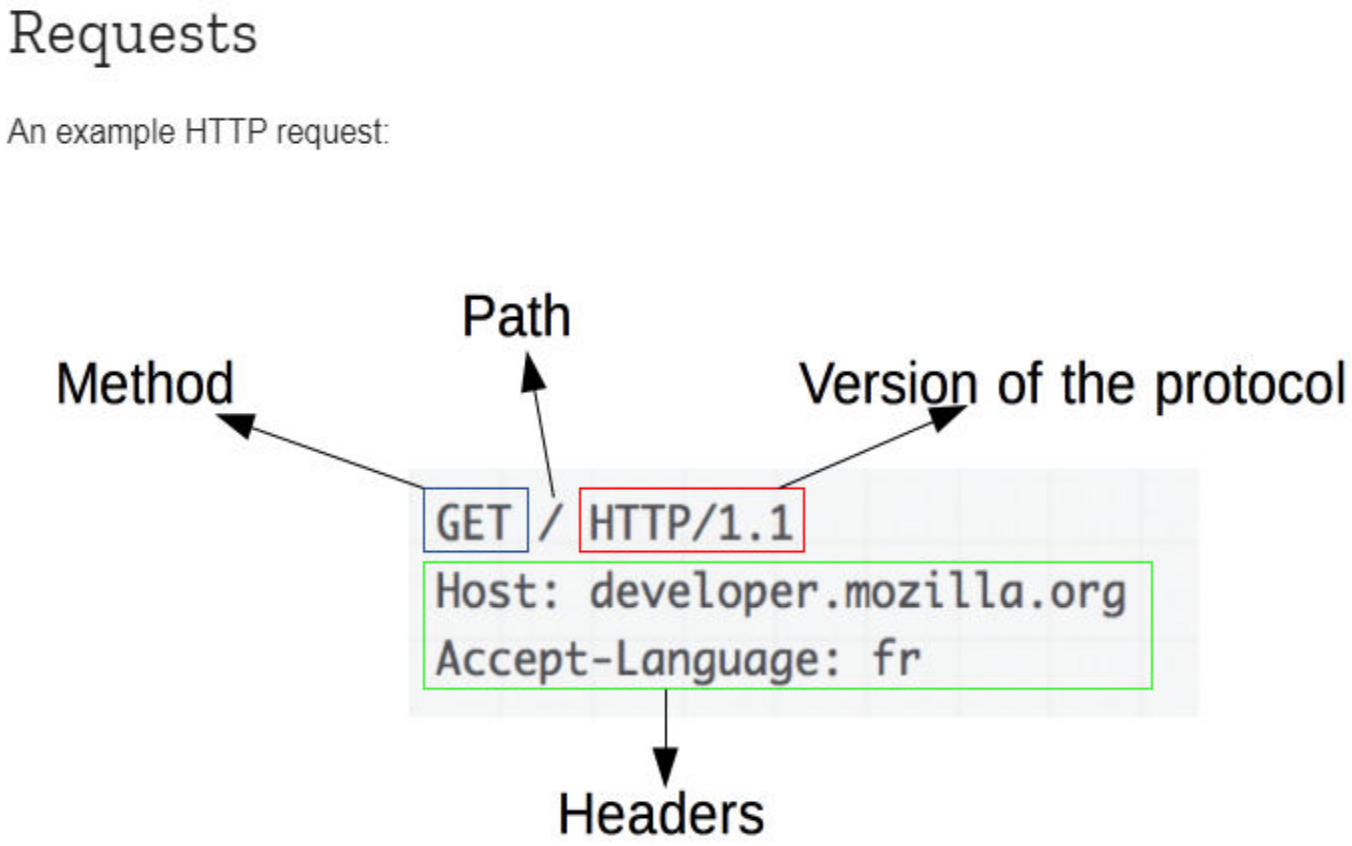
- HTTP and connections
- Among the two most common transport protocols on the Internet, TCP is reliable and UDP isn't.
- HTTP therefore relies on the TCP standard, which is connection-based.
- Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips.
- The default behavior of HTTP/1.0 is to open a separate TCP connection for each HTTP request/response pair.

HTTP flow

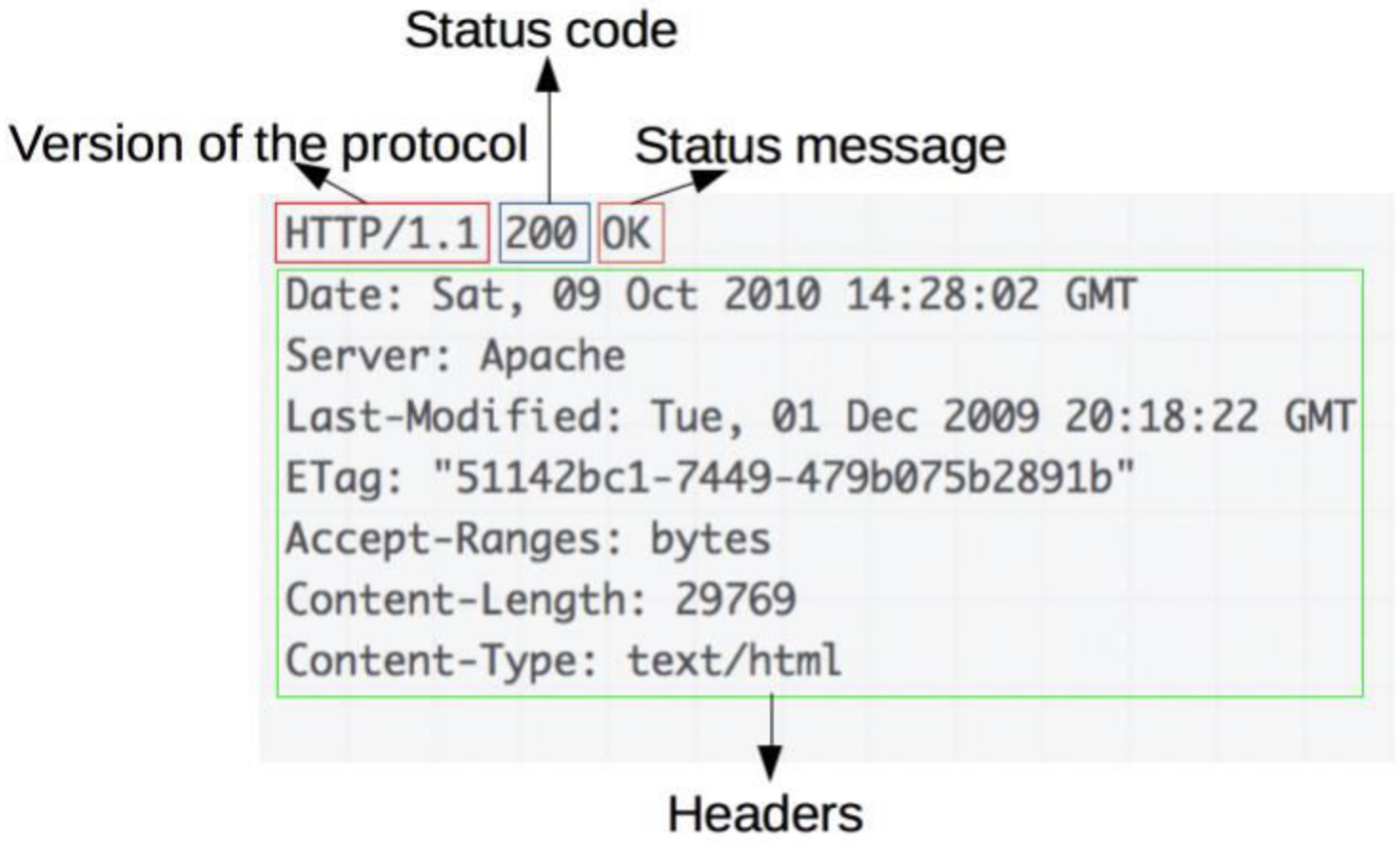
- When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:
- Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer.
- The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
- Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same.
- Read the response sent by the server,
- Close or reuse the connection for further requests.
- If HTTP pipelining is activated, several requests can be sent without waiting for the first response to be fully received.

HTTP messages

- There are two types of HTTP messages, requests and responses, each with its own format.



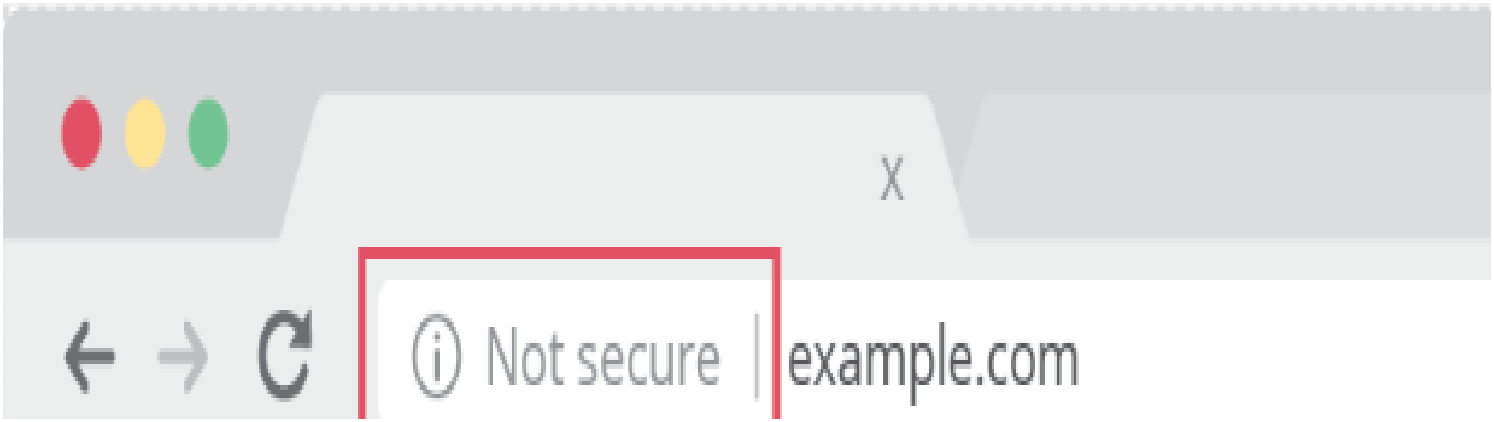
Response


Status code
Version of the protocol
Status message
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
Headers

What is HTTPS?

- Hypertext transfer protocol secure (HTTPS) is the secure version of HTTP, which is the primary protocol used to send data between a web browser and a website.
- HTTPS is encrypted in order to increase security of data transfer.
- This is particularly important when users transmit sensitive data, such as by logging into a bank account, email service, or health insurance provider.
- Any website, especially those that require login credentials, should use HTTPS.
- In modern web browsers such as Chrome, websites that do not use HTTPS are marked differently than those that are.

- Web browsers take HTTPS seriously; Google Chrome and other browsers flag all non-HTTPS websites as not secure.



How does HTTPS work?

- HTTPS uses an encryption protocol to encrypt communications.
- The protocol is called Transport Layer Security (TLS), although formerly it was known as Secure Sockets Layer (SSL).
- This protocol secures communications by using an asymmetric public key infrastructure.
- This type of security system uses two different keys to encrypt communications between two parties:
- The private key
- The public Key

- The private key - this key is controlled by the owner of a website and it's kept, private.
- This key lives on a web server and is used to decrypt information encrypted by the public key.
- The public key - this key is available to everyone who wants to interact with the server in a way that's secure.
- Information that's encrypted by the public key can only be decrypted by the private key.

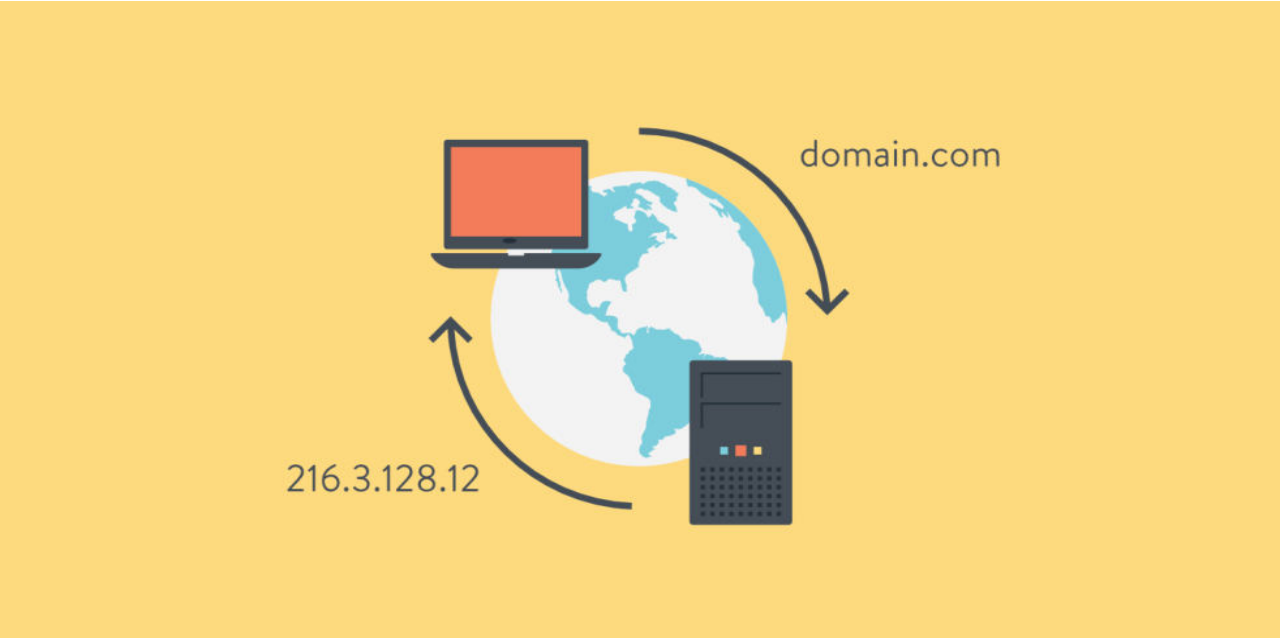
How is HTTPS different from HTTP?

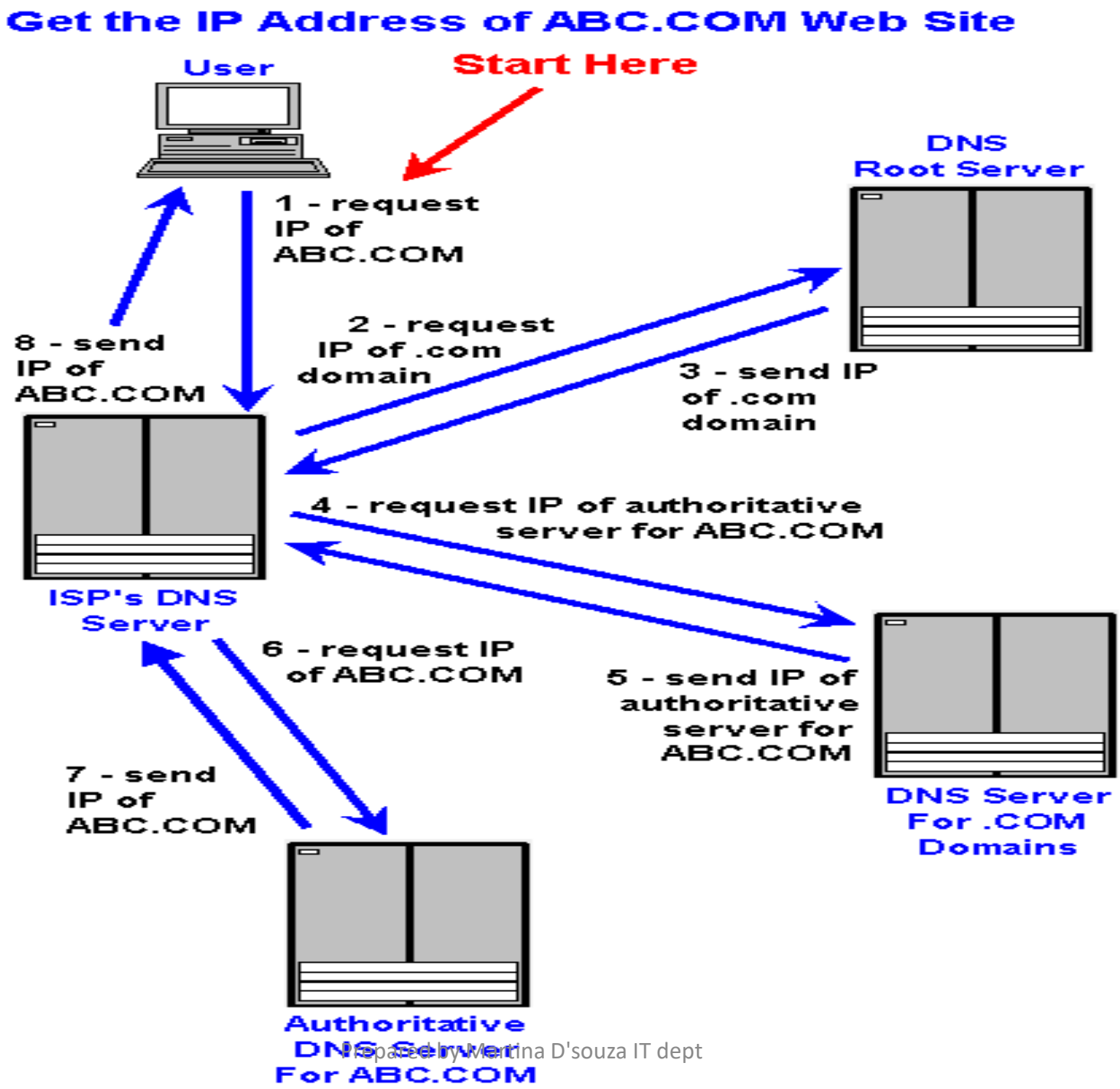
- HTTPS is not a separate protocol from HTTP.
- It is simply using TLS/SSL encryption over the HTTP protocol.
- HTTPS occurs based upon the transmission of TLS/SSL certificates, which verify that a particular provider is who they say they are.
- When a user connects to a webpage, the webpage will send over its SSL certificate which contains the public key necessary to start the secure session.
- The two computers, the client and the server, then go through a process called an SSL/TLS handshake, which is a series of back-and-forth communications used to establish a secure connection.

What is DNS?

- The Domain Name System (DNS) is the phonebook of the Internet.
- Web browsers interact through Internet Protocol (IP) addresses.
- DNS translates domain names to IP addresses so browsers can load Internet resources.
- Each device connected to the Internet has a unique IP address which other machines use to find the device.
- DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).

DNS





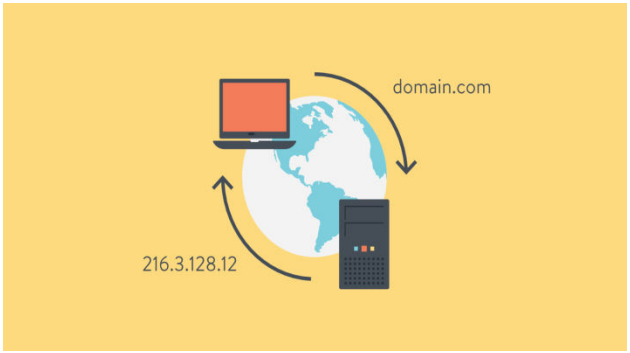
How does DNS work?

- The process of DNS resolution involves converting a hostname (such as www.example.com) into a computer-friendly IP address (such as 192.168.1.1).
- An IP address is given to each device on the Internet, and that address is necessary to find the appropriate Internet device - like a street address is used to find a particular home.
- When a user wants to load a webpage, a translation must occur between what a user types into their web browser (example.com) and the machine-friendly address necessary to locate the example.com webpage.

4 DNS servers involved in loading a webpage

- DNS recursor -
- The recursor can be thought of as a librarian who is asked to go find a particular book somewhere in a library.
- The DNS recursor is a server designed to receive queries from client machines through applications such as web browsers.
- Typically the recursor is then responsible for making additional requests in order to satisfy the client's DNS query.

- **Root nameserver -**
- The root server is the first step in translating (resolving) human readable host names into IP addresses.
- It can be thought of like an index in a library that points to different racks of books - typically it serves as a reference to other more specific locations.



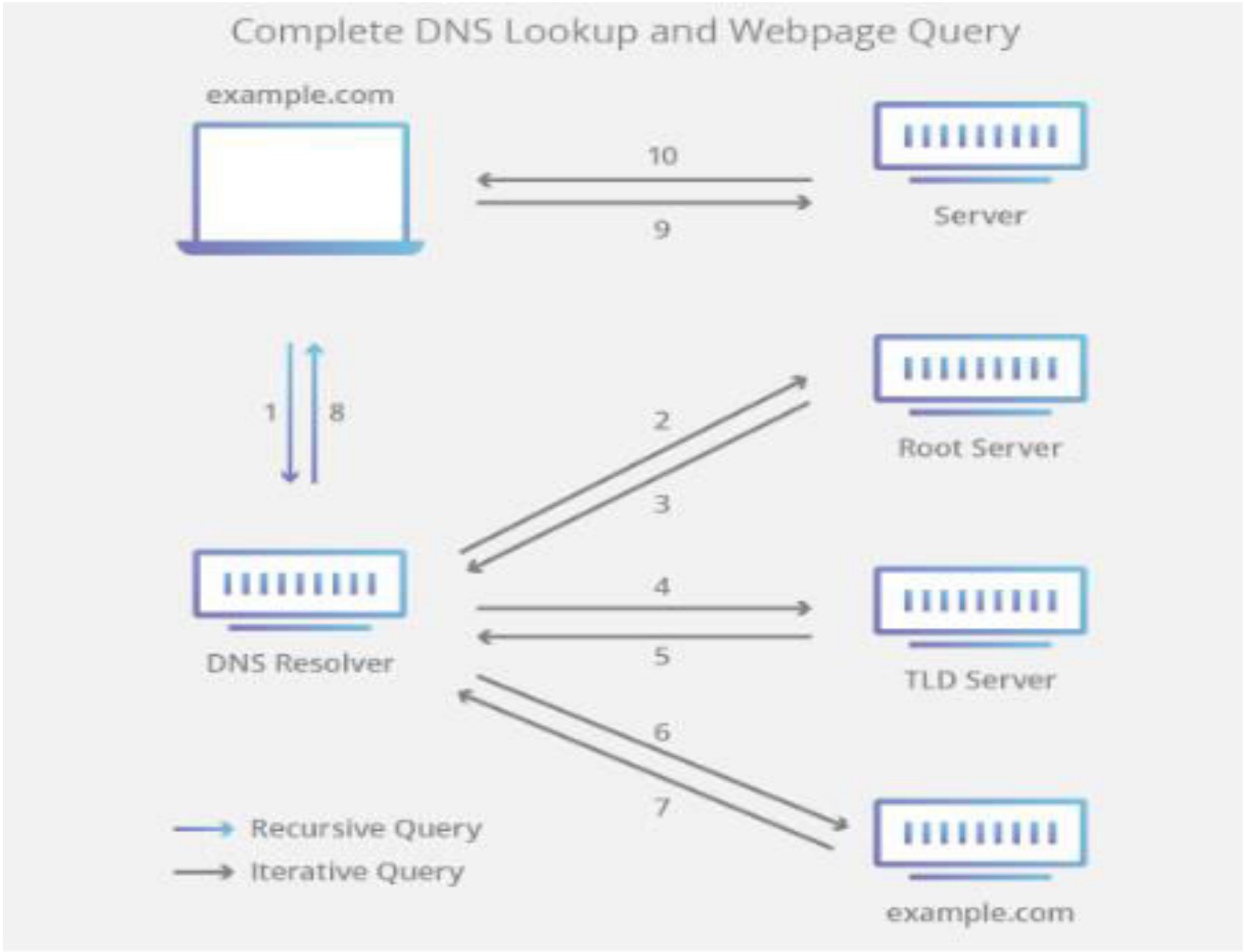
- TLD nameserver -
- The top level domain server (TLD) can be thought of as a specific rack of books in a library.
- This nameserver is the next step in the search for a specific IP address, and it hosts the last portion of a hostname (In example.com, the TLD server is “com”).

- Authoritative nameserver - This final nameserver can be thought of as a dictionary on a rack of books, in which a specific name can be translated into its definition.
- The authoritative nameserver is the last stop in the nameserver query.
- If the authoritative name server has access to the requested record, it will return the IP address for the requested hostname back to the DNS Recursor (the librarian) that made the initial request.

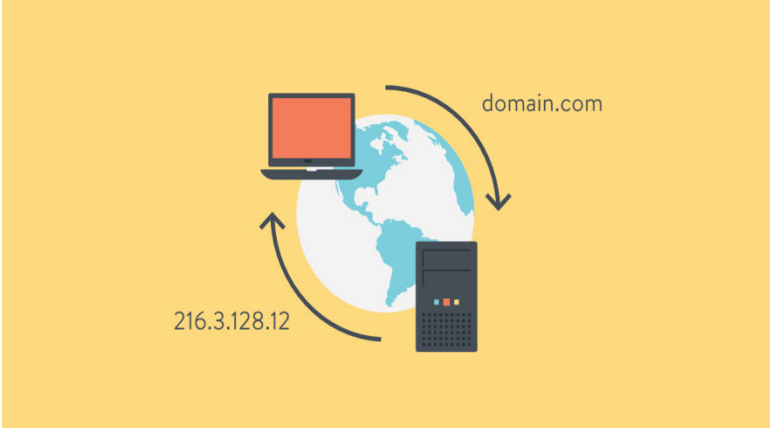
The 8 steps in a DNS lookup:

- A user types ‘example.com’ into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
- The resolver then queries a DNS root nameserver (.).
- The root server then responds to the resolver with the address of a Top Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
- The resolver then makes a request to the .com TLD.
- The TLD server then responds with the IP address of the domain’s nameserver, example.com.

- Lastly, the recursive resolver sends a query to the domain's nameserver.
- The IP address for example.com is then returned to the resolver from the nameserver.
- The DNS resolver then responds to the web browser with the IP address of the domain requested initially.
- Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:
- The browser makes a HTTP request to the IP address.
- The server at that IP returns the webpage to be rendered in the browser (step 10).


Complete DNS Lookup and Webpage Query
example.com
10
9
Server
1
8
2
3
Root Server
DNS Resolver
4
5
TLD Server
6
7
Recursive Query
Iterative Query
example.com

3 types of DNS queries:



- **Recursive query** - In a recursive query, a DNS client requires that a DNS server (typically a DNS recursive resolver) will respond to the client with either the requested resource record or an error message if the resolver can't find the record.

- **Iterative query** - in this situation the DNS client will allow a DNS server to return the best answer it can.
- If the queried DNS server does not have a match for the query name, it will return a referral to a DNS server authoritative for a lower level of the domain namespace.
- The DNS client will then make a query to the referral address.
- This process continues with additional DNS servers down the query chain until either an error or timeout occurs.

- **Non-recursive query** - typically this will occur when a DNS resolver client queries a DNS server for a record that it has access to either because it's authoritative for the record or the record exists inside of its cache. Typically, a DNS server will cache DNS records to prevent additional bandwidth consumption and load on upstream servers.

Transport Layer Security (TLS)

- Transport Layer Security, or TLS, is a widely adopted security protocol designed to facilitate privacy and data security for communications over the Internet.
- A primary use case of TLS is encrypting the communication between web applications and servers, such as web browsers loading a website.
- TLS can also be used to encrypt other communications such as email, messaging, and voice over IP (VoIP).

Difference between TLS and SSL

- TLS evolved from a previous encryption protocol called Secure Sockets Layer (SSL), which was developed by Netscape.
- TLS version 1.0 actually began development as SSL version 3.1, but the name of the protocol was changed before publication in order to indicate that it was no longer associated with Netscape.
- Because of this history, the terms TLS and SSL are sometimes used interchangeably.

difference between TLS and HTTPS

- HTTPS is an implementation of TLS encryption on top of the HTTP protocol, which is used by all websites as well as some other web services.
- Any website that uses HTTPS is therefore employing TLS encryption.

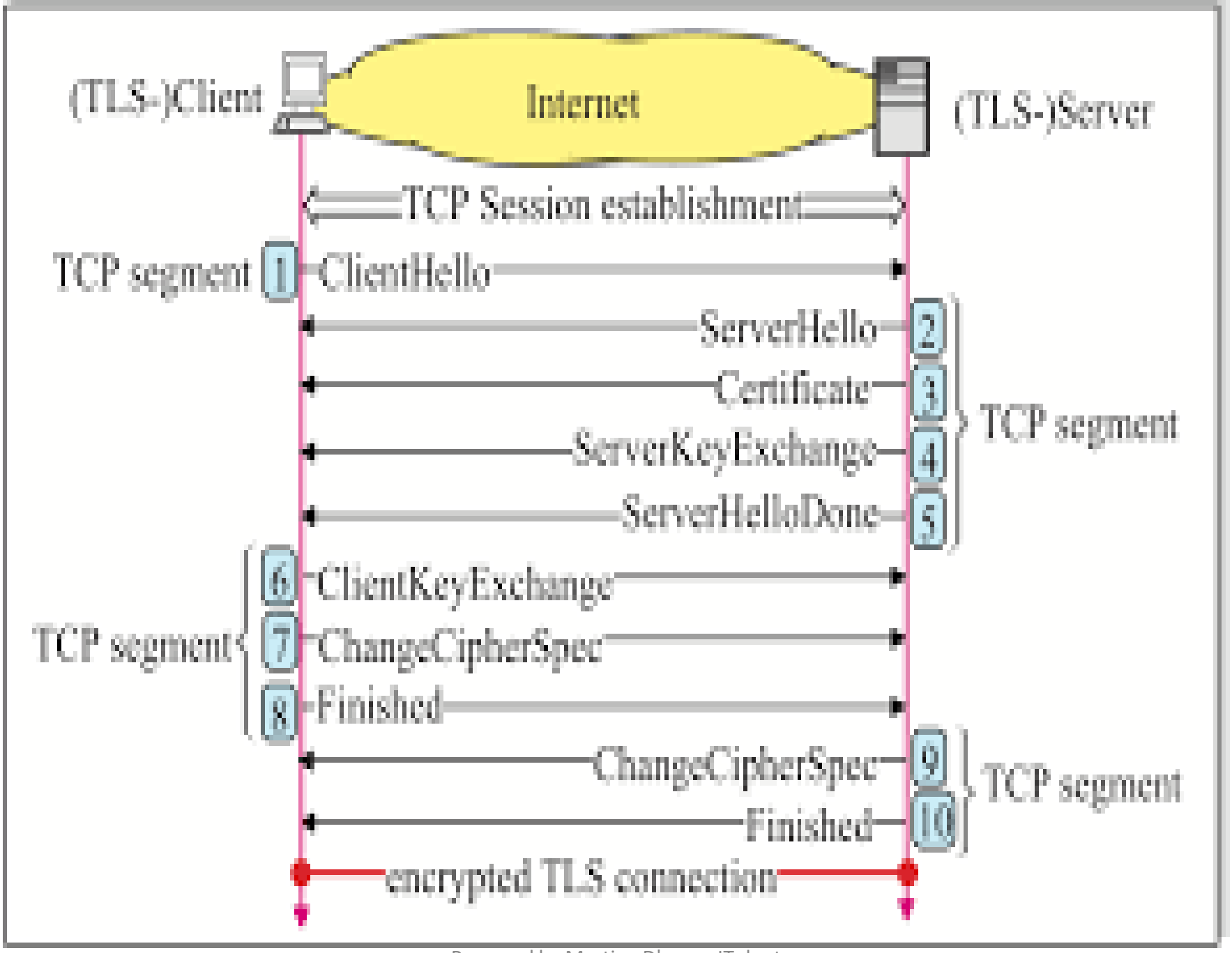
What does TLS do?

- Main components the TLS protocol accomplishes:
- Encryption, Authentication, and Integrity.
- **Encryption:** hides the data being transferred from third parties.
- **Authentication:** ensures that the parties exchanging information are who they claim to be.
- **Integrity:** verifies that the data has not been forged or tampered with.

How does TLS work?

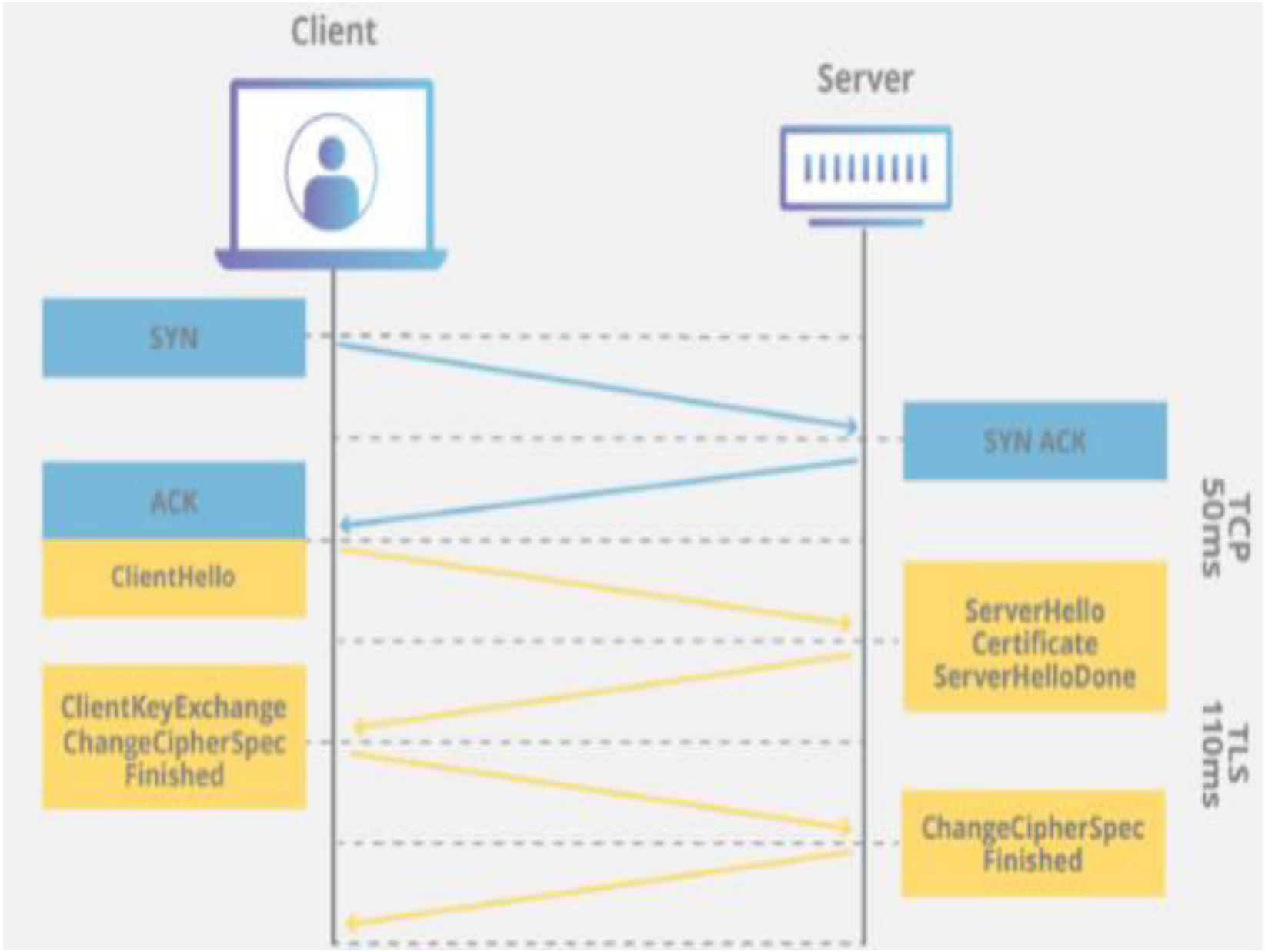
- A TLS connection is initiated using a sequence known as the TLS handshake.
- When a user navigates to a website that uses TLS, the TLS handshake begins between the user's device (also known as the *client* device) and the web server.
- During the TLS handshake, the user's device and the web server:
- Specify which version of TLS (TLS 1.0, 1.2, 1.3, etc.) they will use
- Decide on which cipher suites they will use
- Authenticate the identity of the server using the server's TLS certificate

- Generate session keys for encrypting messages


(TLS-)Client
Internet
(TLS-)Server
TCP Session establishment
TCP segment
1
ClientHello
ServerHello
2
Certificate
3
ServerKeyExchange
4
ServerHelloDone
5
TCP segment
TCP segment
6
ClientKeyExchange
7
ChangeCipherSpec
8
Finished
ChangeCipherSpec
9
Finished
10
TCP segment
encrypted TLS connection

- The TLS handshake establishes a cipher suite for each communication session.
- The cipher suite is a set of algorithms that specifies details such as which shared encryption keys, or session keys, will be used for that particular session.
- TLS is able to set the matching session keys over an unencrypted channel known as public key cryptography.
- The handshake also handles authentication, which usually consists of the server proving its identity to the client.

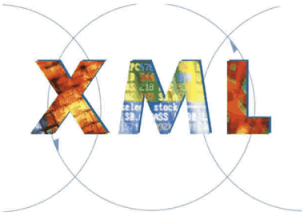
- This is done using public keys.
- Public keys are encryption keys that use one-way encryption, meaning that anyone with the public key can unscramble the data encrypted with the server's private key to ensure its authenticity, but only the original sender can encrypt data with the private key.
- The server's public key is part of its TLS certificate.
- Once data is encrypted and authenticated, it is then signed with a message authentication code (MAC).
- The recipient can then verify the MAC to ensure the integrity of the data.


Client
Server
SYN
SYN ACK
ACK
ClientHello
ServerHello
Certificate
ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
TCP
50ms
TLS
110ms

(XML)

*e**X**tensible **M**arkup **L**anguage*

What is XML?

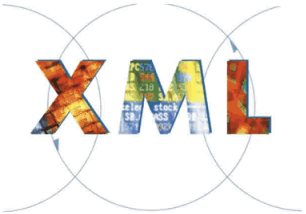


- *e**X**tensible **M**arkup **L**anguage*
- Markup language for documents containing structured information
- Defined by four specifications:
  - XML, the Extensible Markup Language
  - XLL, the Extensible Linking Language
  - XSL, the Extensible Style Language
  - XUA, the XML User Agent

XML….

- Based on Standard Generalized Markup Language (SGML)
- Version 1.0 introduced by World Wide Web Consortium (W3C) in 1998
- Bridge for data exchange on the Web

Comparisons



XML

- Extensible set of tags
- Content orientated
- Standard Data infrastructure
- Allows multiple output forms

HTML

- Fixed set of tags
- Presentation oriented
- No data validation capabilities
- Single presentation

XML Elements

- An XML element is made up of a start tag, an end tag, and data in between.
- Example:

  <director> Matthew Dunn </director>
- Example of another element with the same value:

  <actor> Matthew Dunn </actor>
- XML tags are case-sensitive:

  <CITY> <City> <city>
- XML can abbreviate empty elements, for example:

  <married> </married> can be abbreviated to

  <married/>

XML Elements (cont'd)

- An attribute is a name-value pair separated by an equal sign (=).
- Example:

  <City ZIP="94608"> Emeryville </City>

- Attributes are used to attach additional, secondary information to an element.

XML Documents Must Have a Root Element

- XML documents must contain one **root** element that is the **parent** of all other elements:
- <root>
  <child>
  <subchild>.....</subchild>
  </child>
  </root>

Syntax

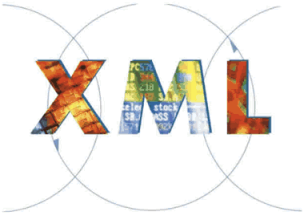
```
<?xml version="1.0" encoding="UTF-8"?>
  <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>We are Meeting today!</body>
  </note>
```

XML Documents

- A basic XML document is an XML element that can, but might not, include nested XML elements.
- Example:

```
<?xml version="1.0"  encoding=“UTF-8” ?>
 <books>
        <book isbn=“123”>
            <title> Second Chance </title>
            <author> Matthew Dunn </author>
        </book>
    </books>
```

XML Documents (cont'd)



- Guidelines:
  - All elements must have an end tag.
  - All elements must be cleanly nested (overlapping elements are not allowed).
  - All attribute values must be enclosed in quotation marks.
  - Each document must have a unique first element, the root node.

Breakfast Menu

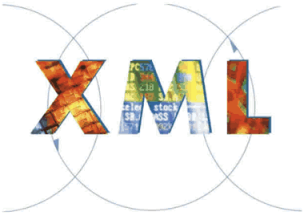
```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream</description>
    <calories>900</calories>
  </food>
</breakfast_menu>
```

Note

- An XML document with correct syntax is called "Well Formed".

- An XML document validated against a DTD is both "Well Formed" and "Valid".

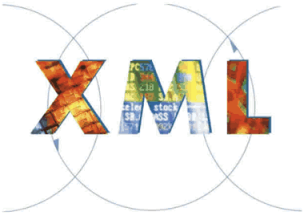
Document Type Definitions (DTD)



- A DTD defines the structure and the legal elements and attributes of an XML document.
- An XML document may have an optional DTD.
- DTD serves as grammar for the underlying XML document, and it is part of XML language.
- DTD has the form:

  <!DOCTYPE name [markupdeclaration]>

DTD (cont'd)



- Consider an XML document:

```
<db><person><name>Alan</name>
                    <age>42</age>
                    <email>agb@usa.net </email>
        </person>
        <person>……</person>
        ……
</db>
```

DTD (cont'd)

- DTD for it might be:

```
<!DOCTYPE db [
    <!ELEMENT db (person*)>
    <!ELEMENT person (name, age, email)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT age (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
]>
```

#PCDATA means parseable character data.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE note SYSTEM "Note.dtd">
  <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
  weekend!</body>
  </note>
```

Note.dtd

```
<!DOCTYPE note
  [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  ]>
```

JSON - Introduction

Java**S**cript **O**bject **N**otation

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is a syntax for storing and exchanging data.
- JSON is "self-describing" and easy to understand
- JSON is language independent, because JSON uses JavaScript syntax, but the JSON format is text only.
- Text can be read and used as a data format by any programming language.

Exchanging Data

- When exchanging data between a browser and a server, the data can only be text.

- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

- Can also convert any JSON received from the server into JavaScript objects.

Why use JSON?

- JSON format is text only, it can easily be sent to and from a server, and can be used as a data format by any programming language.
- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

- if received data from a server, in JSON format, use it like any other JavaScript object.

JSON syntax

- The JSON syntax is a subset of the JavaScript syntax.

JSON Syntax Rules

- JSON syntax is derived from JavaScript object notation syntax:

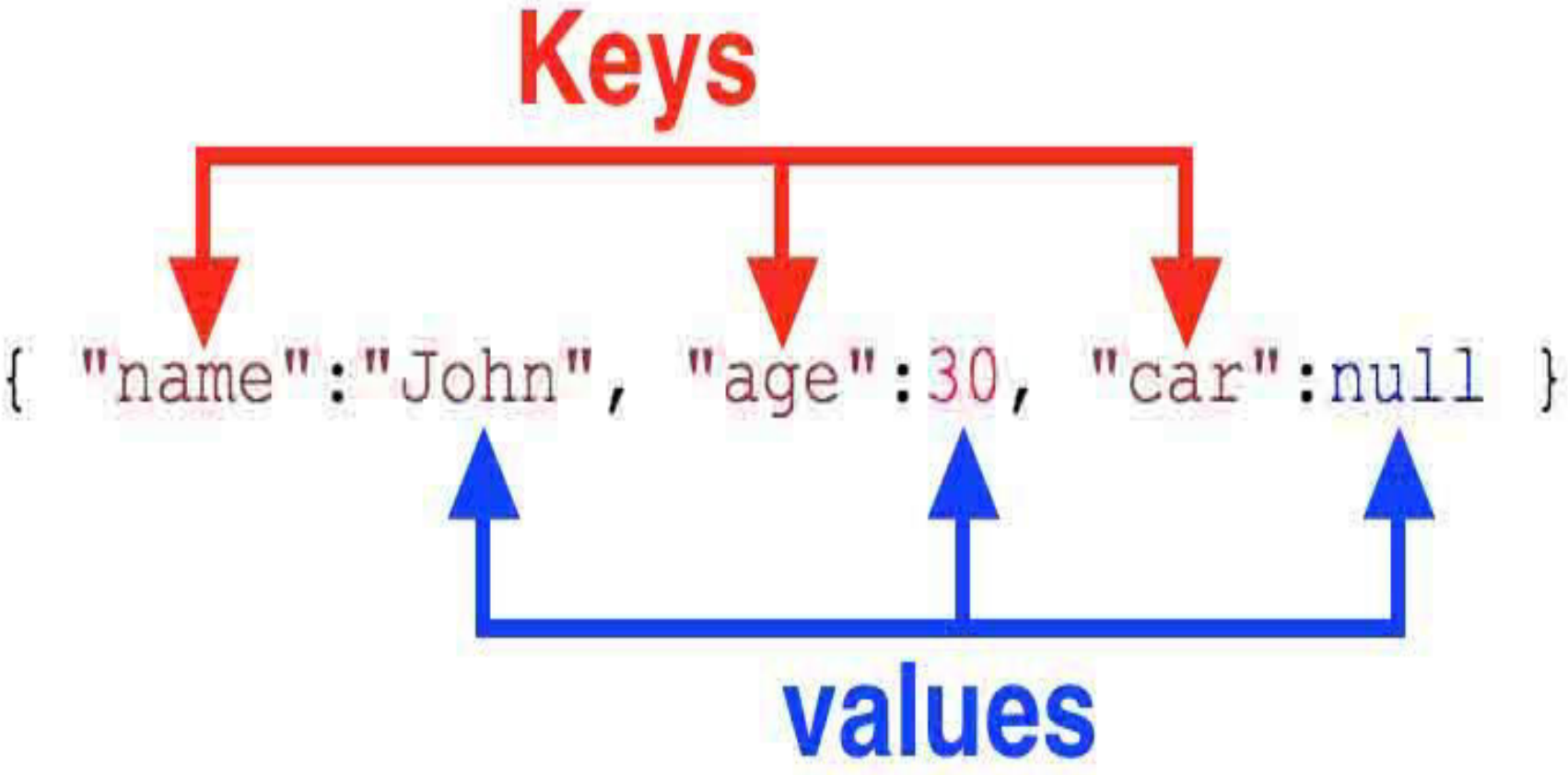
✓ Data is in name/value pairs

✓ Data is separated by commas

✓ Curly braces hold objects

✓ Square brackets hold arrays

JSON Data Format



JSON Data

- JSON Data - A Name and a Value
- JSON data is written as name/value pairs.
- JSON names require double quotes. JavaScript names don't.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"name":"John"

JSON Data Types

- Valid Data Types

✓ a string

✓ a number ---*integer or a floating point.*

✓ an object (JSON object)

✓ an array

✓ a boolean

✓ *null*

Invalid Datatypes

JSON values **cannot** be one of the following data types:

➢ a function

➢ a date

➢ *undefined*

Valid Datatypes

- Strings in JSON

{ "name":"John" }

- JSON Numbers
- Numbers in JSON must be an integer or a floating point.

{ "age":30 }

- JSON Objects

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

- JSON Arrays

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

- JSON Booleans

```
{ "sale":true }
```

- JSON null

```
{ "middlename":null }
```

JSON - Evaluates to JavaScript Objects

- The JSON format is almost identical to JavaScript objects.
- In JSON, *keys* must be strings, written with double quotes:

{ "name":"John" }

- In JavaScript, keys can be strings, numbers, or identifier names:

{ name:"John" }

Curly braces hold objects

JSON Uses JavaScript Syntax

- With JavaScript can create an object and assign data to it, like this:

Example

- var person = { name: "John", age: 31, city: "New York" };
- Access a JavaScript object like this:

Example

- person.name; *// returns John*

  OR

- person["name"];

JSON Files

- The file type for JSON files is ".json“
- The MIME type for JSON text is "application/json"

JSON vs XML

- Both JSON and XML can be used to receive data from a web server.
- The following JSON and XML examples both define an employees object, with an array of 3 employees:

In XML:

```
<employees>
    <employee>
      <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
      <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
      <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

In JSON

- {"employees":[
  { "firstName":"John", "lastName":"Doe" },

  { "firstName":"Anna", "lastName":"Smith" },

  { "firstName":"Peter", "lastName":"Jones" }

]}

Square brackets hold arrays

JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

The biggest difference is:

- XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

JSON.parse()

- A common use of JSON is to exchange data to/from a web server.
- When receiving data from a web server, the data is always a string.
- Parse the data with JSON.parse(), and the data becomes a JavaScript object.

Example - Parsing JSON

- Say, received this text from a web server:

'{ "name":"John", "age":30, "city":"New York"}'

- *Use the JavaScript function JSON.parse() to convert text into a JavaScript object:*

```
var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

- Use the JavaScript object in page:

Example

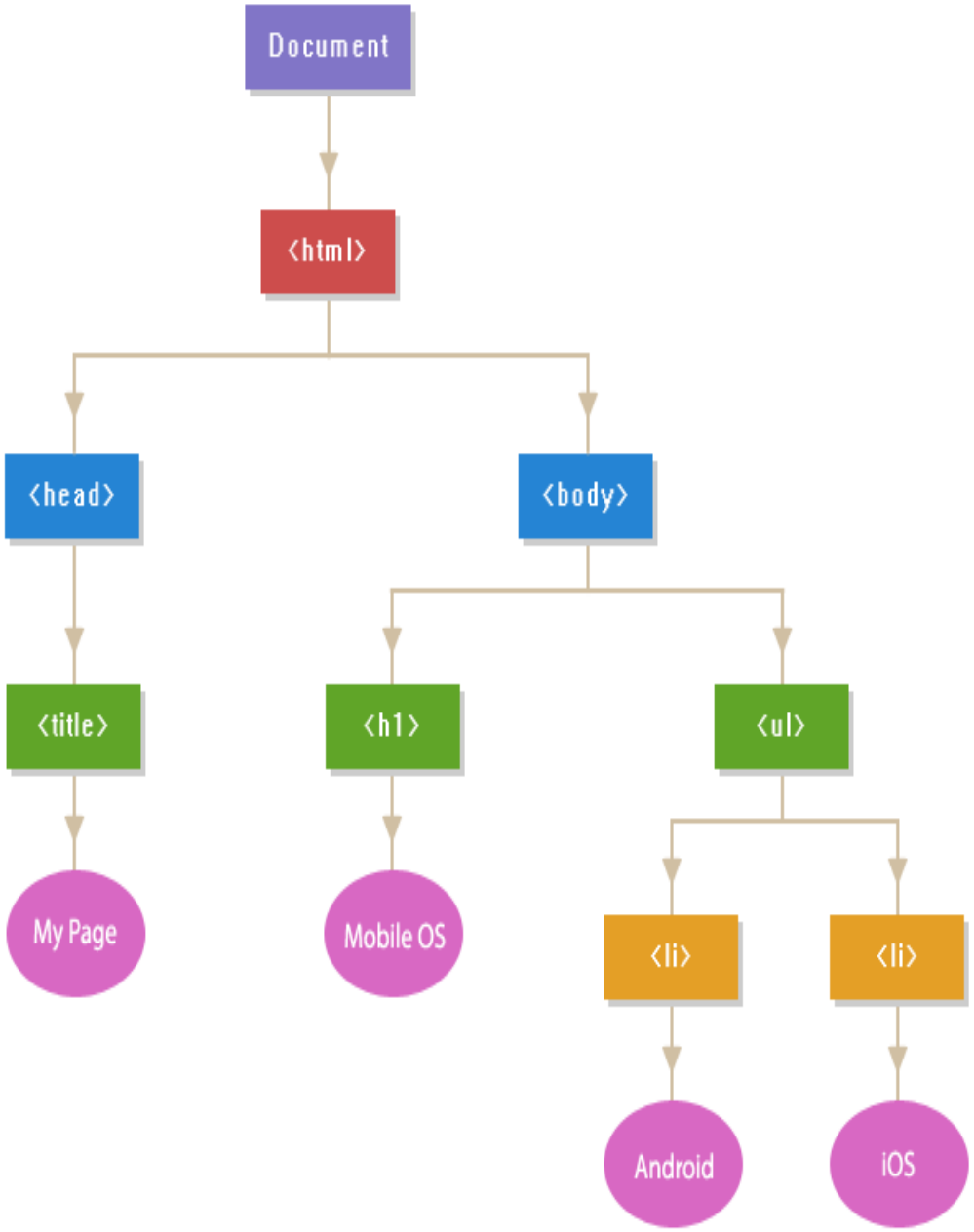
```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
obj.name + ", " + obj.age;
</script>
```

DOM

- The **Document Object Model** (**DOM**) is the data representation of the objects that comprise the structure and content of a document on the web.
- The Document Object Model (DOM) is a programming interface for HTML and XML documents.
- It represents the page so that programs can change the document structure, style, and content.
- The DOM represents the document as nodes and objects.

- In the DOM, all parts of the document, such as elements, attributes, text, etc. are organized in a hierarchical tree-like structure; similar to a family tree in real life that consists of parents and children.
- In DOM terminology these individual parts of the document are known as *nodes*.
- The Document Object Model that represents HTML document is referred to as HTML DOM. Similarly, the DOM that represents the XML document is referred to as XML DOM.

```
<!DOCTYPE html>
<html>
<head>
<title>My
   Page</title>
 </head>
<body> <h1>Mobile
   OS</h1>
 <ul>
<li>Android</li>
 <li>iOS</li>
</ul>
 </body>
```



- HTML attributes such as id, class, title, style, etc. are also considered as nodes in DOM hierarchy but they don't participate in parent/child relationships like the other nodes do.
- They are accessed as properties of the element node that contains them.
- Each element in an HTML document such as image, hyperlink, form, button, heading, paragraph, etc. is represented using a JavaScript object in the DOM hierarchy, and each object contains properties and methods to describe and manipulate these objects.

DOM

- A Web page is a document.
- This document can be either displayed in the browser window or as the HTML source.
- But it is the same document in both cases.
- The Document Object Model (DOM) represents that same document so it can be manipulated.
- The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.

- All of the properties, methods, and events available for manipulating and creating web pages are organized into objects.
- This documentation provides an object-by-object reference to the DOM.
- The modern DOM is built using multiple APIs that work together.
- The core DOM defines the objects that fundamentally describe a document and the objects within it.
- This is expanded upon as needed by other APIs that add new features and capabilities to the DOM. For example, the HTML DOM API adds support for representing HTML documents to the core DOM.

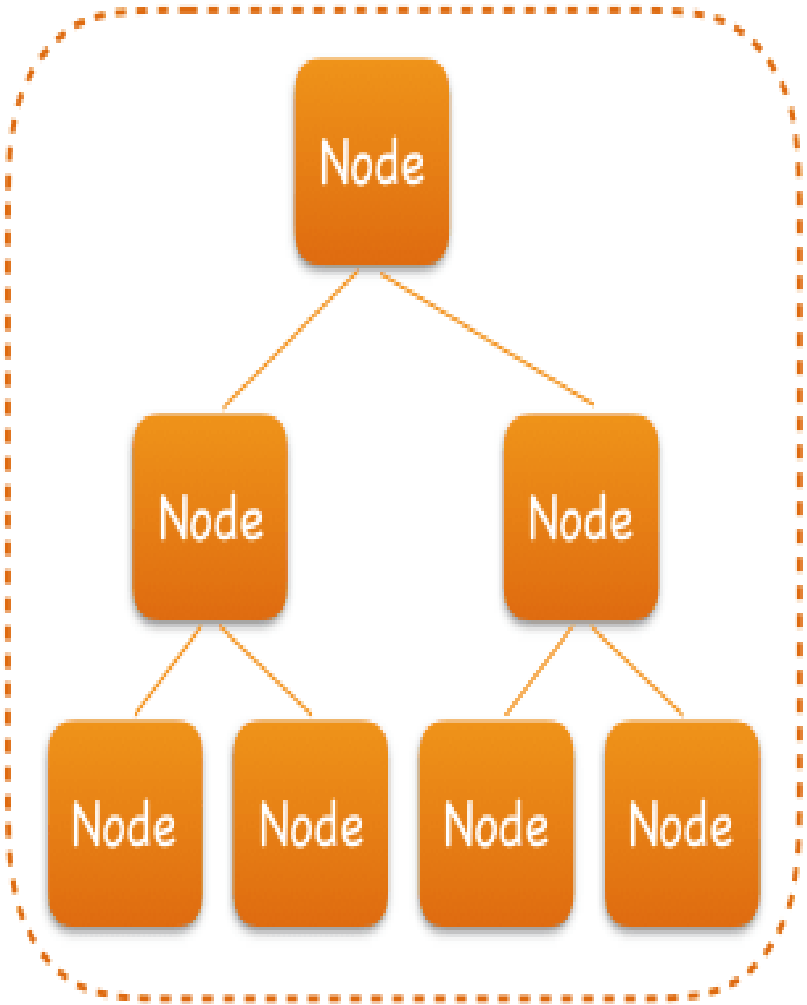
XML DOM

XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student id="001">
        <name>Tom</name>
        <gender>male</gender>
    </student>
    <student id="002">
        <name>Jerry</name>
        <gender>male</gender>
    </student>
</students>
```

Document Object Model (DOM)

Document object



Features of XML DOM

- Any node can have only one parent node. There cannot be a two-parent node for a single node.
- A parent node can have more than one child nodes though.
- Child nodes can have their child nodes assigned in any numbers.
- Child nodes can have other types of nodes assigned to them called “attribute” nodes.
- These nodes are used to contain some extra characteristics of the node. This node is different than the child node.
- The nodes of the same level in a tree are called “sister” nodes.
- DOM identifies the structure of information stored in a hierarchy.

URL

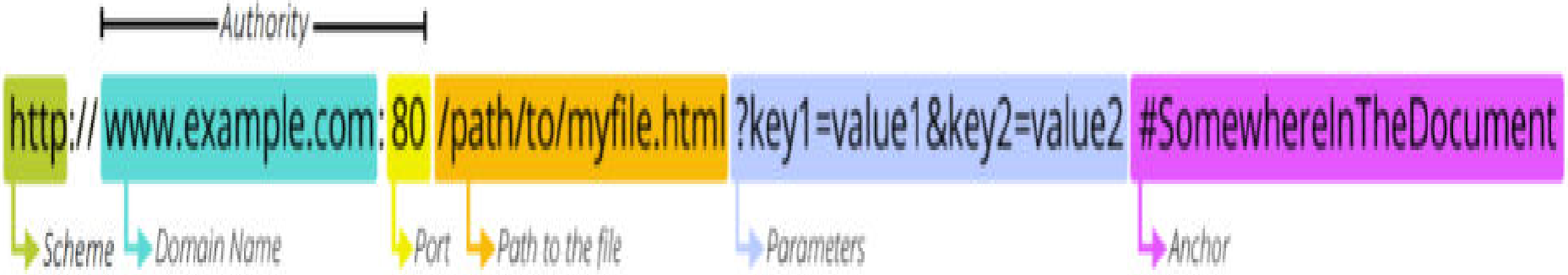
- A **Uniform Resource Locator** (**URL**), termed a **web address**, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
- **Uniform Resource Locator (URL)** refers to a web address which uniquely identifies a document over the internet.
- A URL is a specific type of Uniform Resource Identifier (URI), although many people use the two terms interchangeably.
- URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database access (JDBC), and many other applications.

URL

- **URL** stands for *Uniform Resource Locator*.
- A URL is nothing more than the address of a given unique resource on the Web.
- Each valid URL points to a unique resource.
- Such resources can be an HTML page, a CSS document, an image, etc.

- https://developer.mozilla.org
  https://developer.mozilla.org/en-US/docs/Learn/
  https://developer.mozilla.org/en-US/search?q=URL

- A URL is composed of different parts, some mandatory and others optional.
- The most important parts are highlighted on the URL below:



think of a URL like a regular postal mail address: the scheme represents the postal service you want to use, the domain name is the city or town, and the port is like the zip code; the path represents the building where your mail should be delivered; the parameters represent extra information such as the number of the apartment in the building; and, finally, the anchor represents the actual person to whom you've addressed your mail.

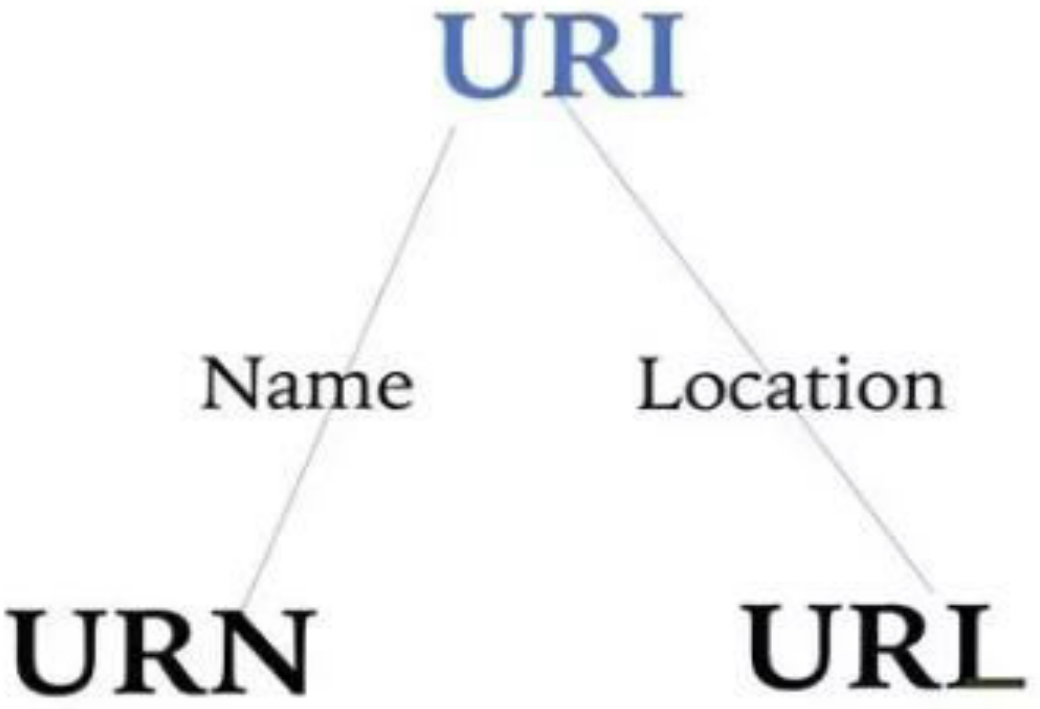
URL

- This document can be a web page, image, audio, video or anything else present on the Internet
- Most web browsers display the URL of a web page above the page in an address bar.
- A typical URL could have the form http://www.example.com/index.html, which indicates a
- protocol (http),
- hostname (www.example.com), and
- file name (index.html).

Example

- http://www.house.gov/house/House_Calendar.html

- Protocol: **http**
- Host computer name: **www**
- Second Level domain name: **house**
- Top level domain name: **gov**
- Directory name : **house**
- File name: **House_Calendar.html**



URI which specifies the Location is URL

URI which specifies the Name is URN

URI which specifies both Location and name is URI

Difference between URI, URL and URN

URI: http://www.assignmenthelp.net/assignment_help/What-is-a-URL

URL: http://www.assignmenthelp.net/assignment_help

URN: www.assignmenthelp.net/assignment_help/What-is-a-URL

Commonly used TLDs

- Top Level Domains (TLDs)

.com	Used for commercial entities. It is the most popular
.edu	Four year, accredited colleges and universities.
.net	Originally used for networking organizations such as Internet Service Providers
.org	Designed for miscellaneous organisation, including non profit groups.

URL Types

- There are two forms of URL as listed below:
- Absolute URL
- Relative URL
- **ABSOLUTE URL**
- Absolute URL is a complete address of a resource on the web. This completed address comprises of protocol used, server name, path name and file name.

- For example http://www.internetpgrm.com/ internet_technology /index.html. where:
- **http** is the protocol.
- **internetpgrm.com** is the server name.
- **index.htm** is the file name.
- The protocol part tells the web browser how to handle the file.
- Similarly we have some other protocols also that can be used to create URL are:
- FTP https
- Gopher mailto
- news

RELATIVE URL

- Relative URL is a partial address of a webpage. Unlike absolute URL, the protocol and server part are omitted from relative URL.

- Relative URLs are used for internal links i.e. to create links to file that are part of same website as the WebPages on which you are placing the link.

RELATIVE URL

- For example, to link an image on internetpgrm.com/internet_technology/internet_referemce_models, we can use the relative URL which can take the form

- **/internet_technologies/internet-osi_model.jpg.**

Difference between Absolute and Relative URL

Absolute URL	Relative URL
Used to link web pages on different websites	Used to link web pages within the same website.
Difficult to manage.	Easy to Manage
Changes when the server name or directory name changes	Remains same even of we change the server name or directory name.
Take time to access	Comparatively faster to access.

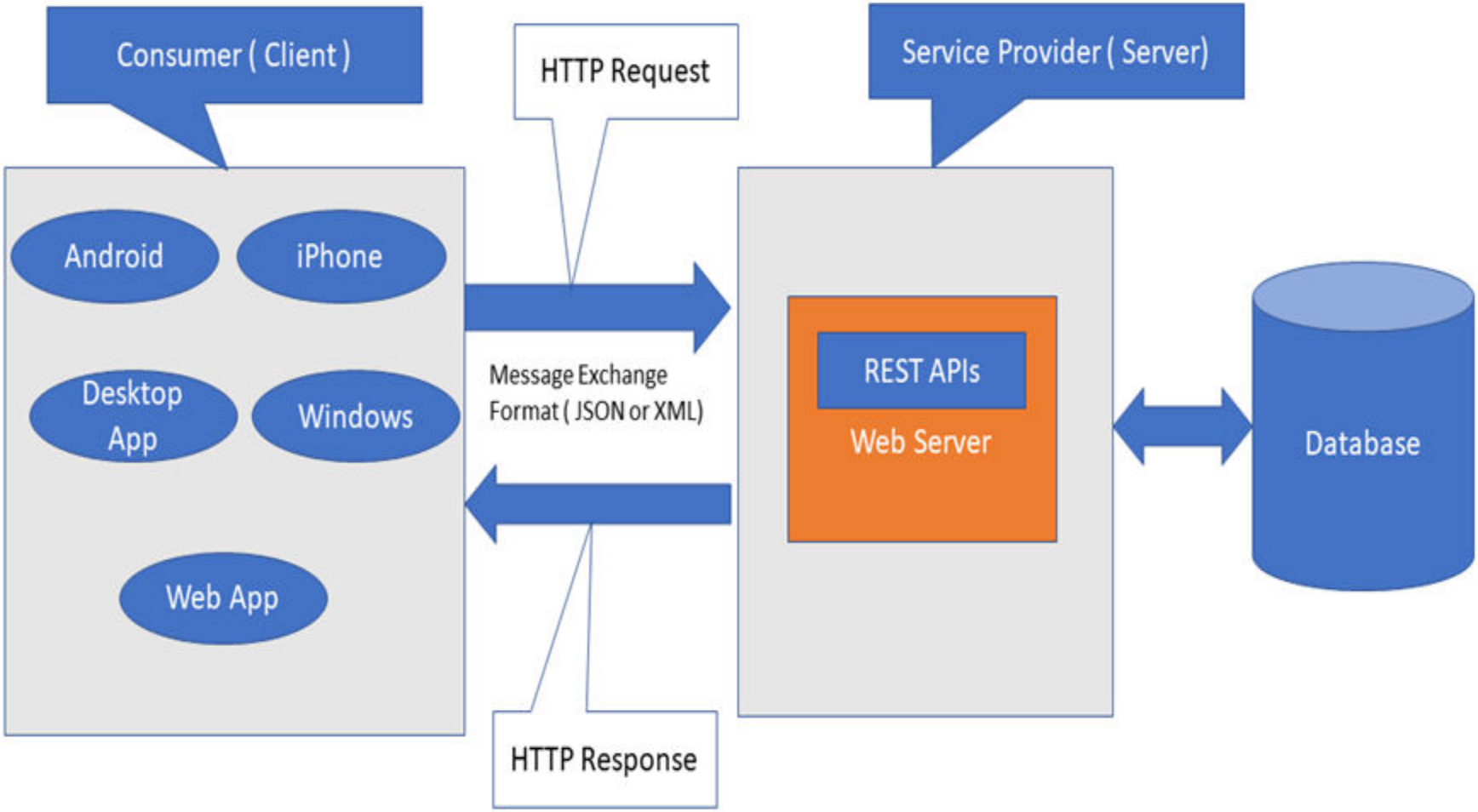
Internet v/s WWW

- The Internet is a global network of networks
- while the Web, also referred formally as World Wide Web (www) is collection of information which is accessed via the Internet.
- Another way to look at this difference is; the Internet is infrastructure while the Web is service on top of that infrastructure.
- The internet is millions of computers connect by thousands of networks while
- the world wide web is billions of pages that can be accessed through the internet.
- A collection of related web pages.
- Main page of a web site; usually the first page

What is a REST API?

- An API, or *application programming interface*, is a set of rules that define how applications or devices can connect to and communicate with each other.
- A REST API is an API that conforms to the design principles of the REST, or ***representational state transfer*** architectural style.
- For this reason, REST APIs are sometimes referred to RESTful APIs.
- REST APIs provide a flexible, lightweight way to integrate applications, and have emerged

REST – Architecture



- REST uses less bandwidth, simple and flexible making it more suitable for internet usage.
- It’s used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.

REST design principles – also known as architectural constraints:

- **Uniform interface.** All API requests for the same resource should look the same, no matter where the request comes from.
- The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI).
- Resources shouldn't be too large but should contain every piece of information that the client might need.

REST API

- **Client-server decoupling.**
- In REST API design, client and server applications must be completely independent of each other.
- The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways.
- Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.
- **Statelessness.** REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it.
- In other words, REST APIs do not require any server-side sessions. Server applications aren't allowed to store any data related to a client request.

- **Cacheability**.
- The goal is to improve performance on the client side, while increasing scalability on the server side.
- **Layered system architecture**. In REST APIs, the calls and responses go through different layers.
- There may be a number of different intermediaries in the communication loop.
- REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.
- Code on demand (optional). REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets).
- In these cases, the code should only run on-demand.

How REST APIs work

- REST APIs communicate via HTTP requests to perform standard database functions like creating, reading, updating, and deleting records (also known as CRUD) within a resource.
- For example, a REST API would use a GET request to retrieve a record, a POST request to create one, a PUT request to update a record, and a DELETE request to delete one.
- All HTTP methods can be used in API calls.
- A well-designed REST API is similar to a website running in a web browser with built-in HTTP functionality.

- The state of a resource at any particular instant, or timestamp, is known as the resource representation.
- This information can be delivered to a client in virtually any format including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text.
- JSON is popular because it's readable by both humans and machines—and it is programming language-agnostic.
- Request headers and parameters are also important in REST API calls because they include important identifier information such as metadata, authorizations, uniform resource identifiers (URIs), caching, cookies and more.

- Request headers and response headers, along with conventional HTTP status codes, are used within well-designed REST APIs.