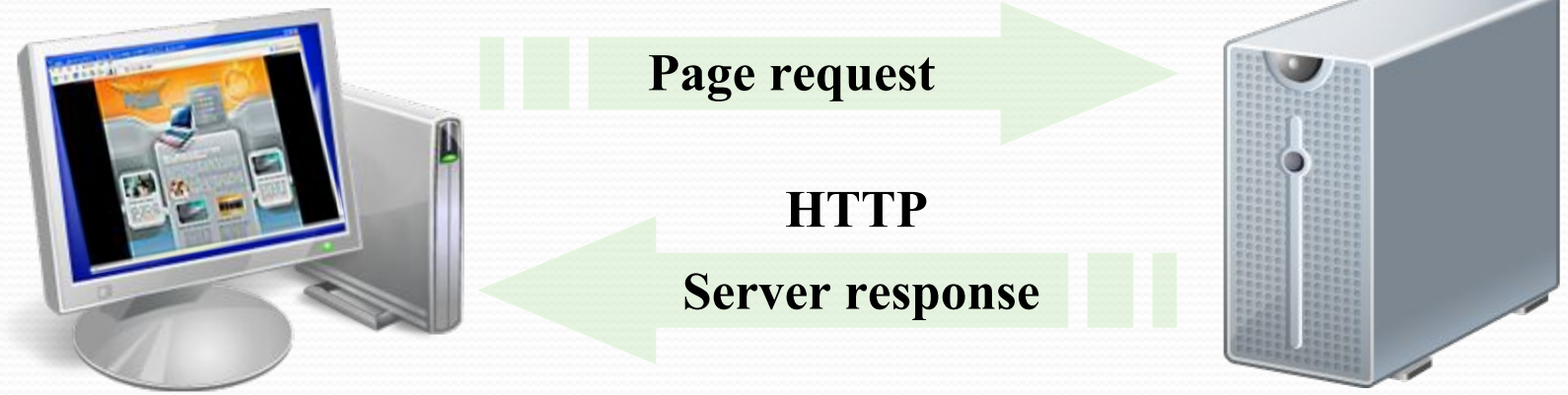
# Introduction to web programming

# How the Web Works?

- WWW use classical client / server architecture
  - HTTP is text-based request-response protocol




**Client running a Web Browser**

**Server running Web Server Software (IIS, Apache, etc.)**

# Web Browser:

- A browser is a software application used to locate, retrieve and display content on the World Wide Web, including Web pages, images, video and other files.
- As a client/server model, the browser is the client run on a computer that contacts the Web server and requests information.
- The Web server sends the information back to the Web browser which displays the results on the computer or other Internet-enabled device that supports a browser.

- Today's browsers are fully-functional software suites that can interpret and display HTML Web pages, applications, JavaScript, AJAX and other content hosted on Web servers.
- Many browsers offer plug-ins which extend the capabilities of the software so it can display multimedia information (including sound and video), or the browser can be used to perform tasks such as videoconferencing, to design web pages or add anti-phishing filters and other security features to the browser.

- **A browser is a group of structured codes which together performs a series of tasks to display a web page on the screen.**
- According to the tasks they perform, these codes are made as different components.

- **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.
- I will give examples from the open source browsers Firefox and Chrome, and Safari

- **The browser's main functionality**
- The main function of a browser 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.
- For years browsers conformed to only a part of the specifications and developed their own extensions.
- That caused serious compatibility issues for web authors. Today most of the browsers more or less conform to the specifications.

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

- 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

- Strangely enough, the browser's user interface is not specified in any formal specification, it just comes from good practices shaped over years of experience and by browsers imitating each other.
- The HTML5 specification doesn't define UI elements a browser must have, but lists some common elements.
- Among those are the address bar, status bar and tool bar.
- There are, of course, features unique to a specific browser like Firefox's downloads manager.

- **The browser's high level structure**

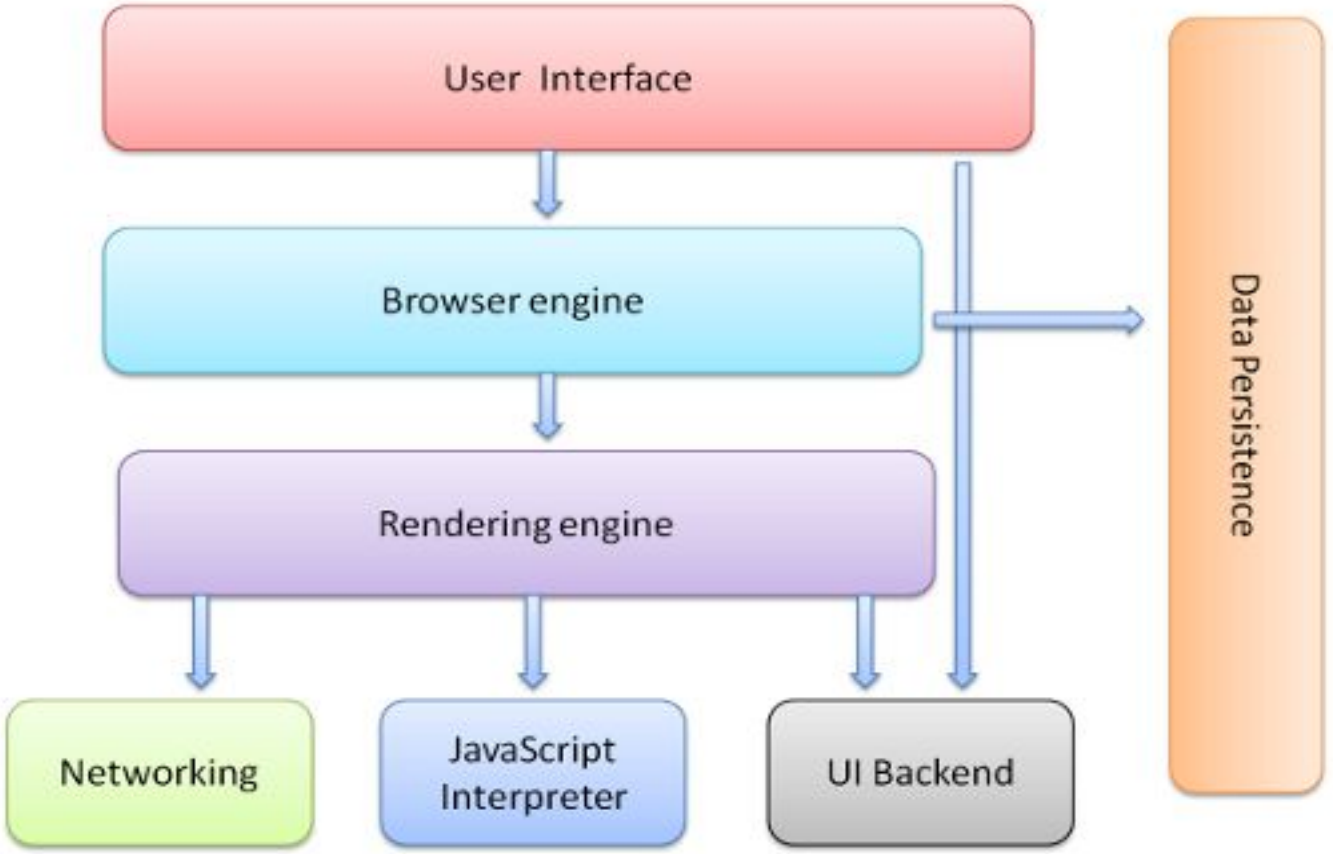



*Figure : Browser components*

- **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.

- **Networking:** for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.
- **UI backend:**
- used for drawing basic widgets like combo boxes and windows.
- This backend exposes a generic interface that is not platform specific.
- **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.

- **High-level architecture of browser**
- The below image shows the main components of a web browser:

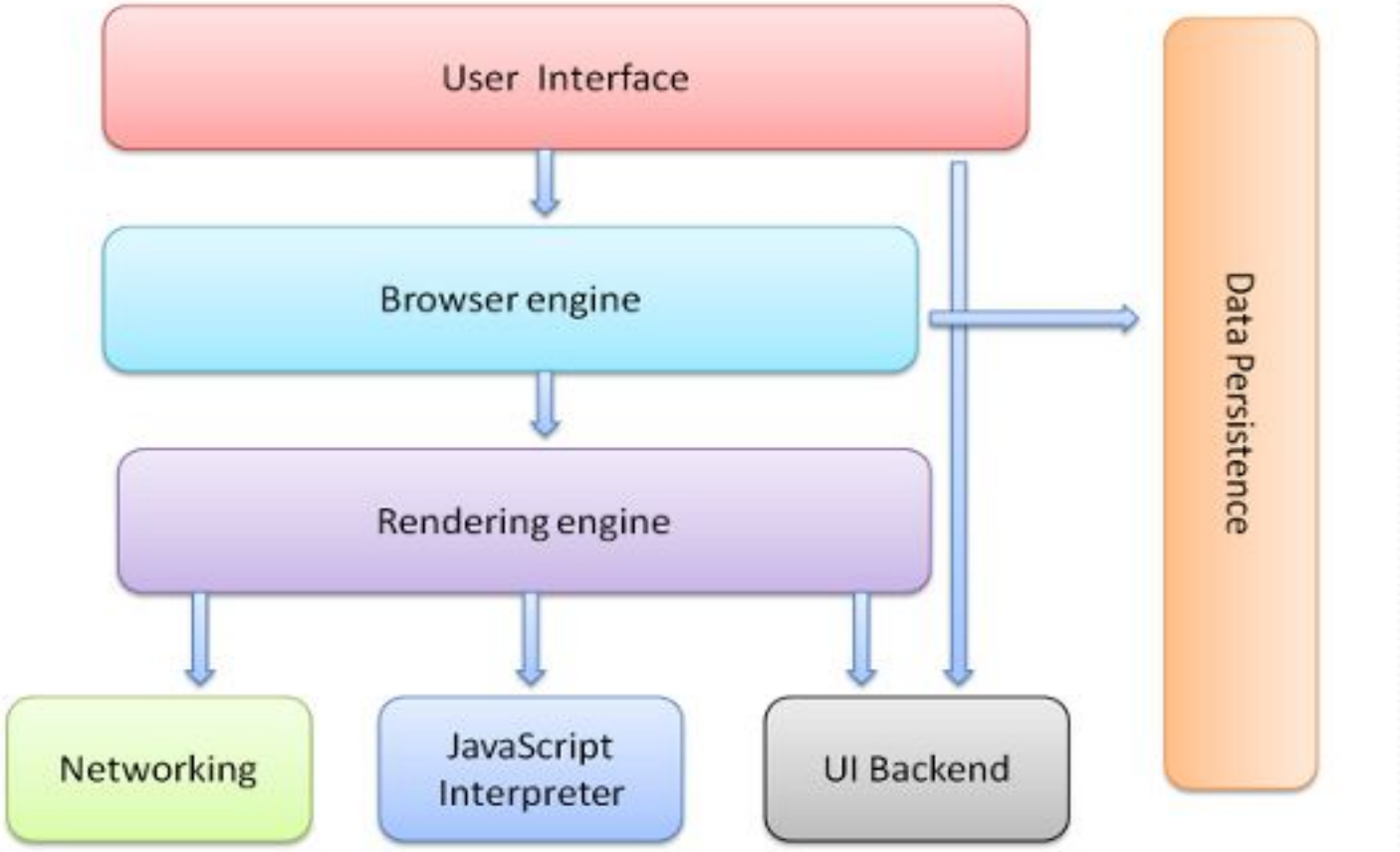


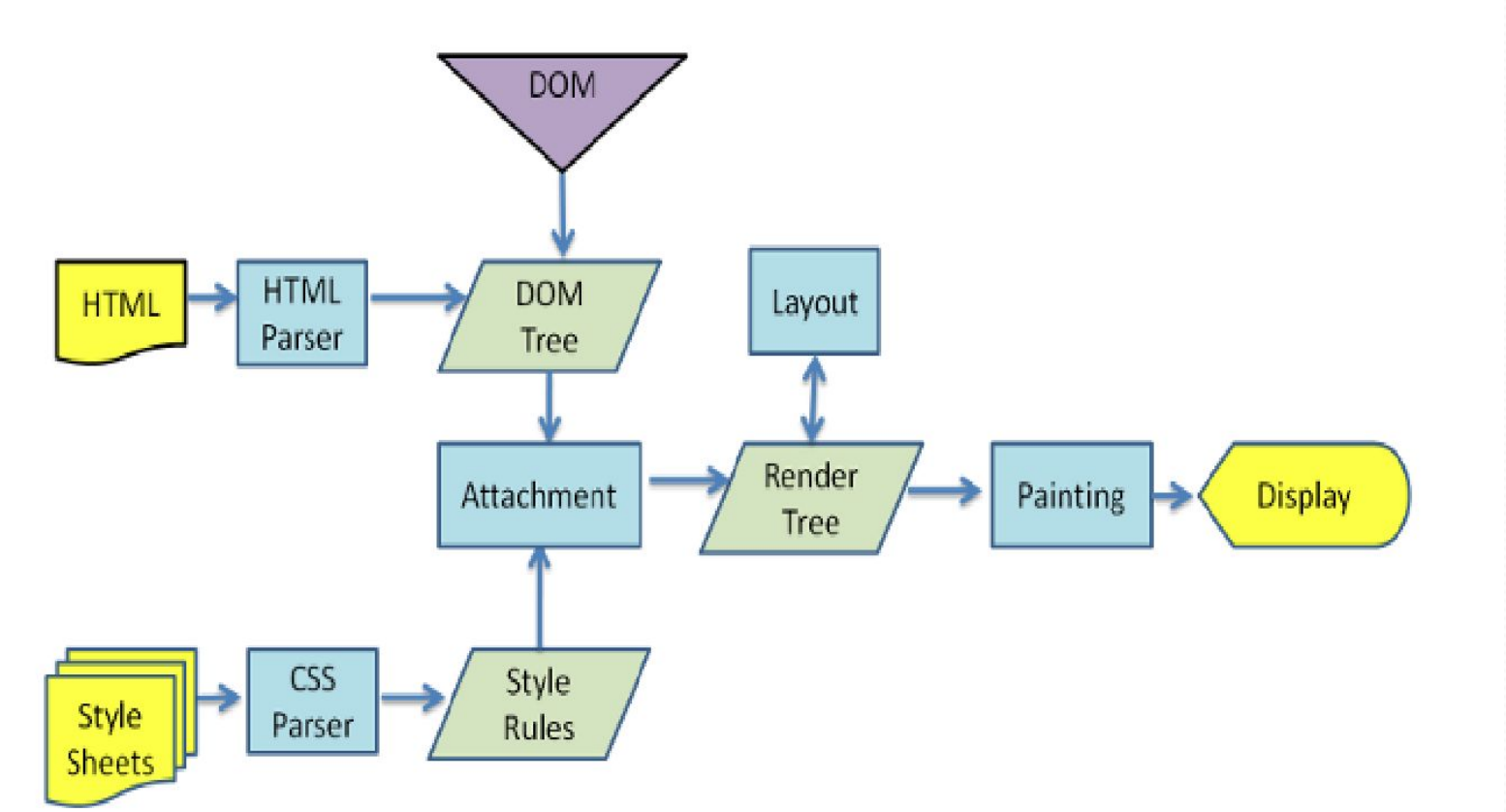

*Figure : Browser components*

- **The User Interface:**
- The user interface is the space where User interacts with the browser.
- It includes the address bar, back and next buttons, home button, refresh and stop, bookmark option, etc.
- Every other part, except the window where requested web page is displayed, comes under it.
- This component allows end-users to interact with all visual elements available on the web page.
- The visual elements include the address bar, home button, next button, and all other elements that fetch and display the web page requested by the end-user.

- **The Browser Engine:**
- It is a core component of every web browser.
- The browser engine functions as an intermediary or a bridge between the user interface and the rendering engine.
- It queries and handles the rendering engine as per the inputs received from the user interface.

- **The Rendering Engine:**
- The responsibility of the rendering engine is to display of the requested contents on the browser screen.
- The rendering engine, as the name suggests is responsible for rendering the requested web page on the browser screen.
- The rendering engine interprets the HTML, XML documents and images that are formatted using CSS and generates the layout that is displayed in the User Interface.
- However, using plugins or extensions, it can display other types data also.
- Different browsers user different rendering engines:
- * Internet Explorer: Trident
- * Firefox & other Mozilla browsers: Gecko
- * Chrome & Opera 15+: Blink
- * Chrome (iPhone) & Safari: Webkit

- Main flow examples



- DOM
- 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.
- The DOM has an almost one-to-one relation to the markup.

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

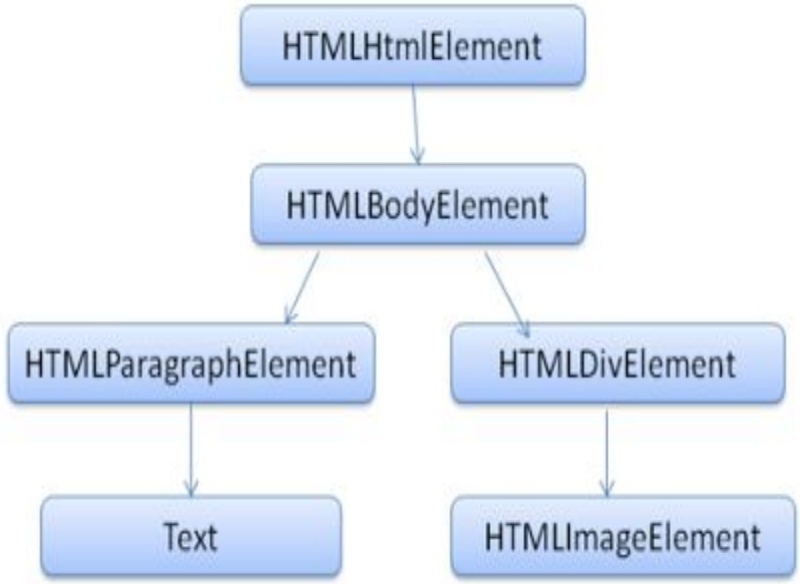
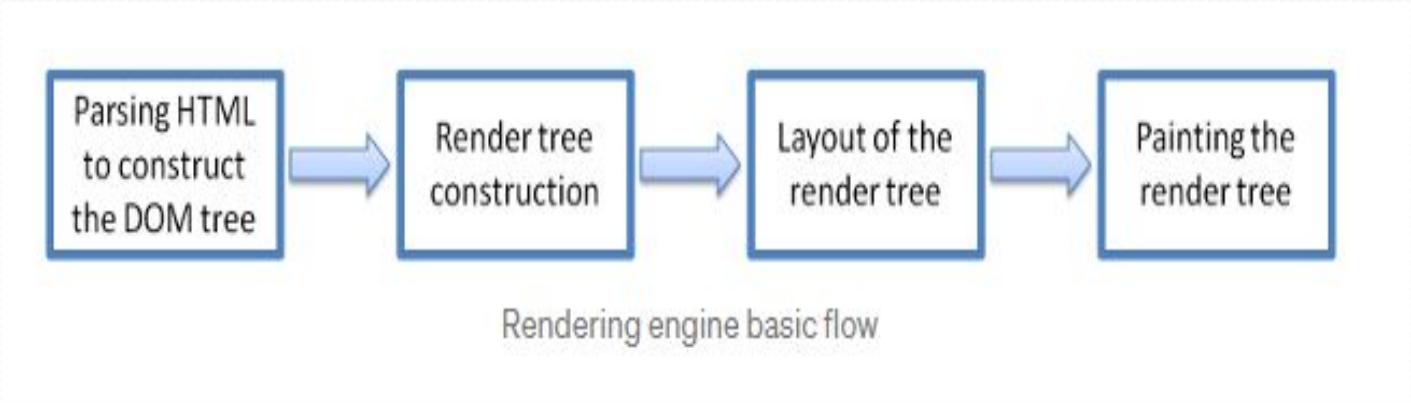



*Figure : DOM tree of the example markup*

- **Networking:**
- This component is responsible for managing network calls using standard protocols like HTTP or FTP.
- It also looks after security issues associated with internet communication.
- Component of the browser which retrieves the URLs using the common internet protocols of HTTP or FTP.
- The networking component handles all aspects of Internet communication and security.
- **JavaScript Interpreter:**
- It is the component of the browser which interprets and executes the javascript code embedded in a website.
- The interpreted results are sent to the rendering engine for display.
- If the script is external then first the resource is fetched from the network.
- Parser keeps on hold until the script is executed.
- As the name suggests, it is responsible for parsing and executing the JavaScript code embedded in a website.
- Once the interpreted results are generated, they are forwarded to the rendering engine for displaying on the user interface.

- **UI Backend:**
- UI backend is used for drawing basic widgets like combo boxes and windows.
- This component uses the user interface methods of the underlying operating system.
- It is mainly used for drawing basic widgets (windows and combo boxes).
- **Data Persistence/Storage:**
- This is a persistence layer. Browsers support storage mechanisms such as localStorage, IndexedDB, WebSQL and 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.

- **Rendering engine**
- The networking layer will start sending the contents of the requested documents to the rendering engine in chunks of 8KBs.



Rendering engine basic flow

- The rendering engine parses the chunks of HTML document and convert the elements to DOM nodes in a tree called the "c**ontent tree**" or the "**DOM tree**".
- It also parses both the external CSS files as well in style elements.

- The rendering engine parses the chunks of HTML document and convert the elements to DOM nodes in a tree called the “content tree” or the “DOM tree”.
- It also parses both the external CSS files as well in style elements.
- 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.
- Firefox calls the elements in the render tree “frames”.
- WebKit uses the term renderer or render object.
- After the construction of the render tree, it goes through a “layout process” of the render tree.

- When the renderer is created and added to the tree, it does not have a position and size.
- The process of calculating these values is called layout or reflow.
- This means giving each node the exact coordinates where it should appear on the screen.
- The position of the root renderer is 0,0 and its dimensions are the viewport–the visible part of the browser window.
- All renderers have a “layout” or “reflow” method, each renderer invokes the layout method of its children that need layout.
- **The next stage is painting.**
- 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 backend layer.

- A web browser, also known as a browser, is an application or software.
- T**he main function** of a web browser is to allow us to access websites available on the internet.
- Some of the web browsers are Google Chrome and Apple safari.
- Web browsers are mainly used to access and display websites on the internet.
- The primary function of a web browser is to render HTML (Hypertext markup language).
- HTML is a code used to design all web pages, giving each website a specific address.
- Netscape Navigator and Mosaic were the first web browsers.

## Client server Architecture components

- Business Logic(Logic Layer)
- Database(Data Layer)
- Graphical User Interface(Presentation Layer)

- Presentation Layer

  Static or dynamically generated content rendered by the browser (front-end)

- Logic Layer

  A dynamic content processing and generation level application server, e.g., Java EE, ASP.NET, PHP, ColdFusion platform (middleware)

- Data Layer

  A database, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data (back-end)

## Architecture Principles

- Client-server architecture
- Each tier (Presentation, Logic, Data) should be independent and should not expose dependencies related to the implementation
- Unconnected tiers should not communicate
- Change in platform affects only the layer running on that particular platform

## Logic Layer

- The set of rules for processing information
- Can accommodate many users
- Sometimes called middleware/ back-end
- Should not contain presentation or data access code

## Presentation Layer

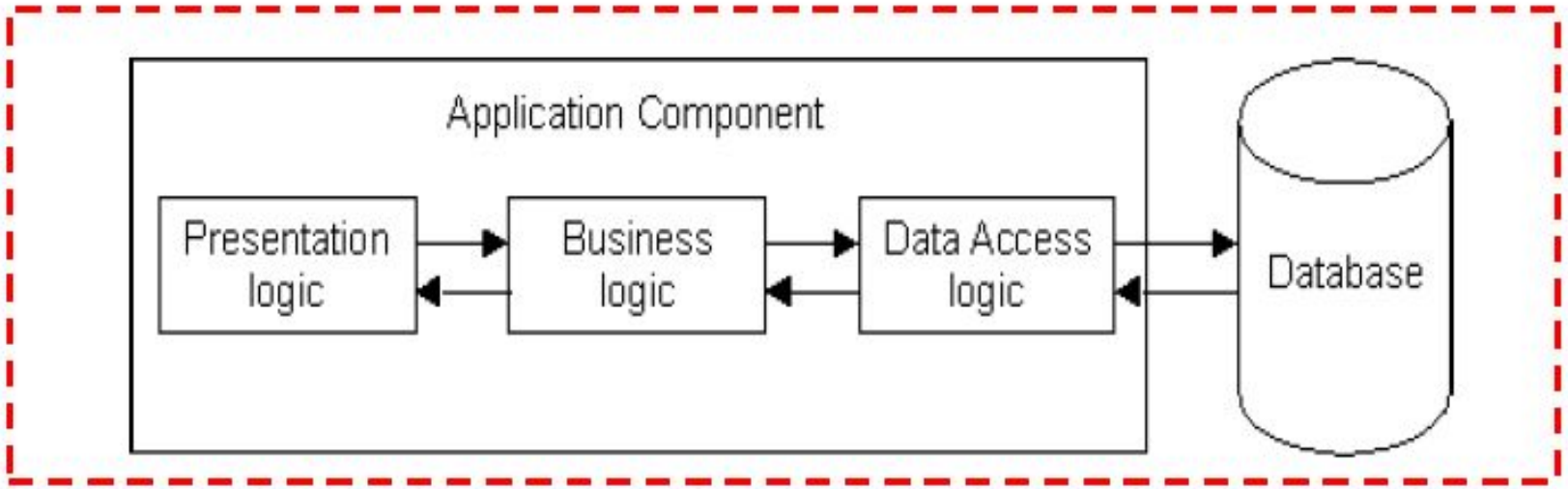
- Provides user interface
- Handles the interaction with the user
- Sometimes called the GUI or client view or front-end
- Should not contain business logic or data access code

## Data Layer

- The physical storage layer for data persistence
- Manages access to DB or file system
- Sometimes called back-end
- Should not contain presentation or business logic code

Types of Architecture:

# The 1-Tier Architecture



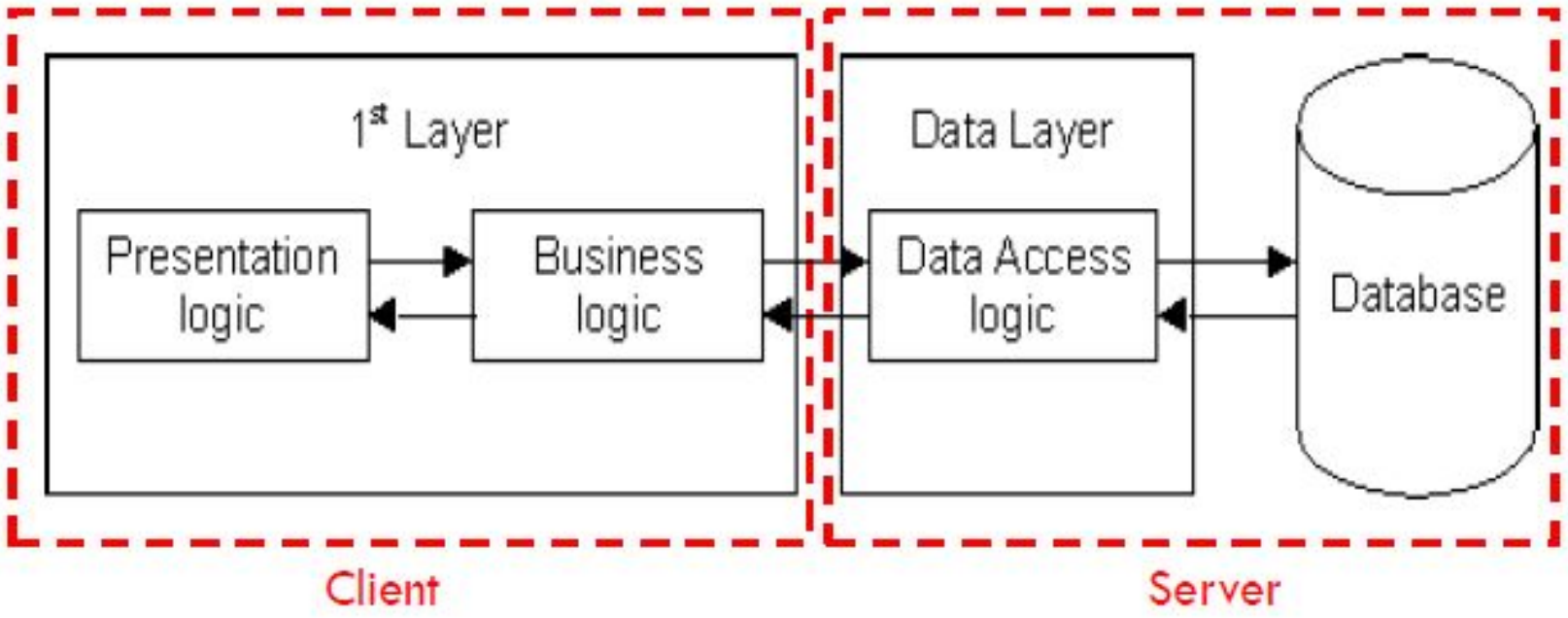

- All 3 layers are on the same machine
  - All code and processing kept on a single machine
- Presentation, Logic, Data layers are tightly connected
  - Scalability: Single processor means hard to increase volume of processing
  - Portability: Moving to a new machine may mean rewriting everything
  - Maintenance: Changing one layer requires changing other layers

# The 1-Tier Architecture

- **Advantages**
  - Easy and quick to develop
  - Useful for small offices

- **Disadvantages**
  - Difficult to upgrade
  - Not scalable
  - Don't protect valuable "Business Logic"  and database.

# The 2-Tier Architecture




- Database runs on Server
  - Separated from client
  - Easy to switch to a different database
- Presentation and logic layers still tightly connected
  - Heavy load on server
  - Potential congestion on network
  - Presentation still tied to business logic

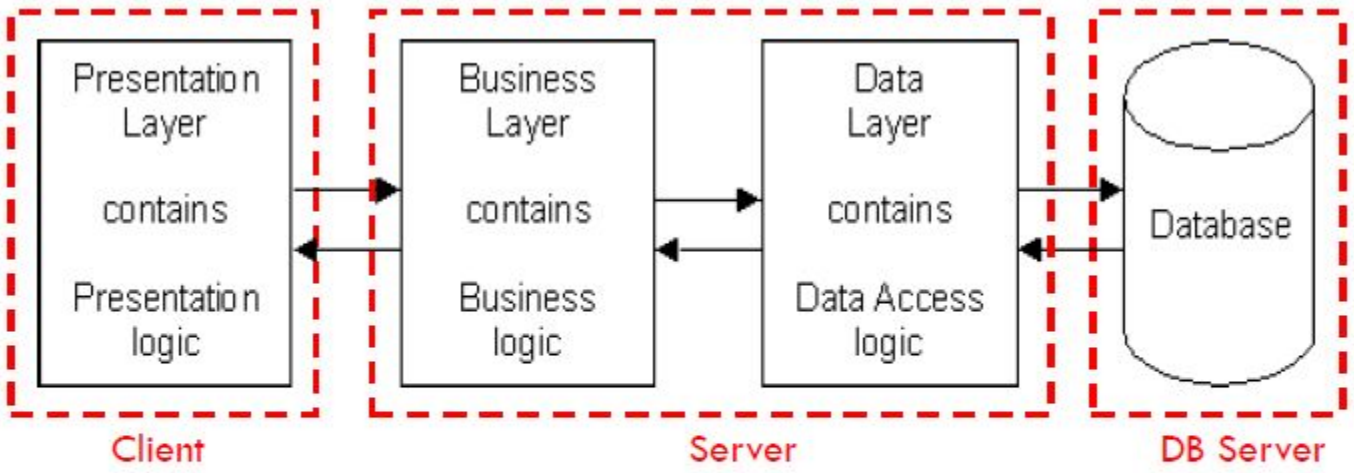
**Advantages:**

- Easy to maintain database. No memory wastage .
- Database is more secure as compared to 1 tier.
- Scalable.

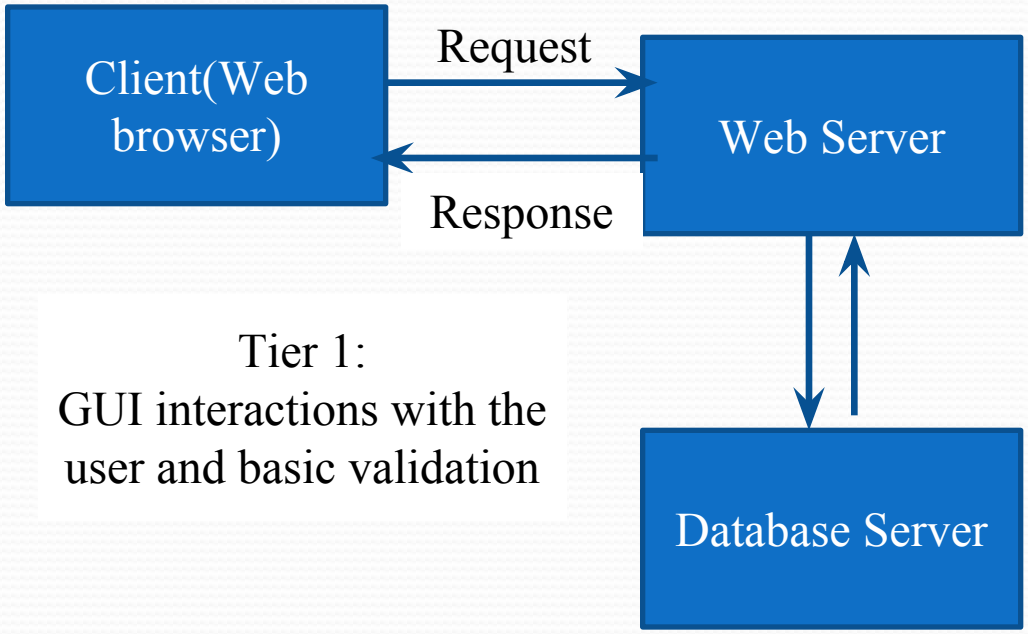
**Disadvantages:**

- Business-logic is implemented on the same PC as database.
- Increased network traffic
- Application logic can’t be reused
- Must design/implement protocol for communication between client and server.

# The 3-Tier Architecture




- Each layer can potentially run on a different machine
- Presentation, logic, data layers disconnected




Tier 1:
GUI interactions with the user and basic validation

Tier2:locating resources

Tier3: Transaction Management

# Advantages:

**Improved Security**

Since the client doesn't have direct access to the database,data layer is more secure.

Business logic is generally more secure since it is placed on a secure central server.

**Reduced Distribution:**

Change to business logic only need to be updated on application servers and need not to distributed on clients.

**Improved Availability:**

More than one server can be used so it can recover from network of server failures.

**Easier to Maintain**

**Components are reusable.**

**Faster Development:**

**Web designer does  presentation.**

**S/W engineer does Logic**

**DBA /Database Designer does data model**

- **Web Application Architecture**

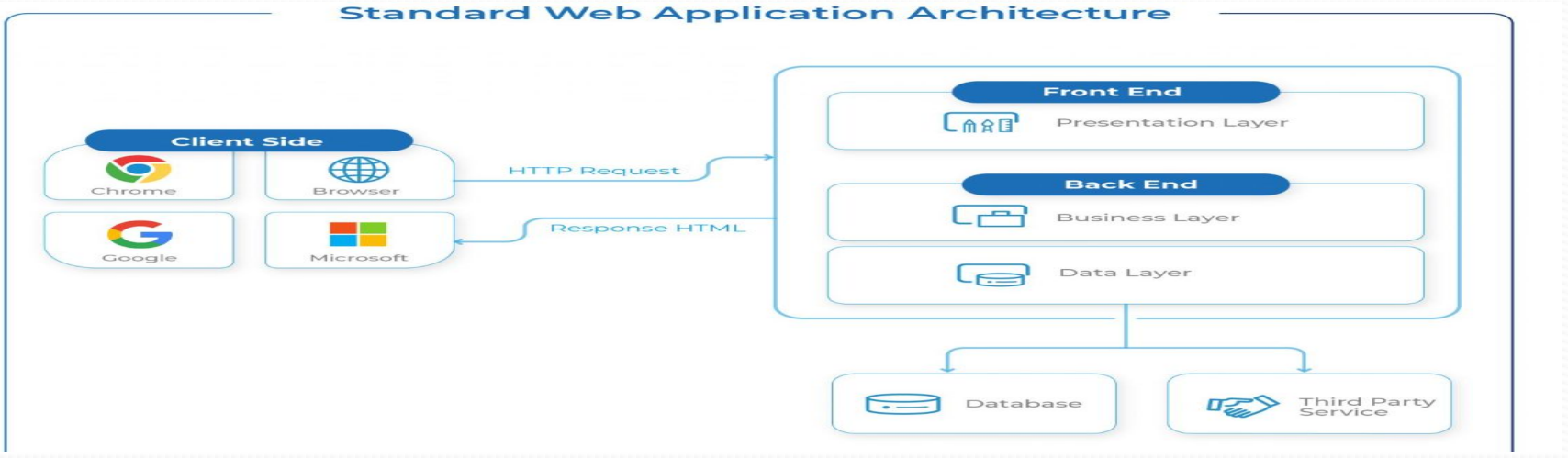
- **Web Application Architecture**
- A web app architecture presents a layout with all the software components and how they interact with each other.
- It defines how the data is delivered through HTTP and ensures that the client-side server and the backend server can understand.
- **Web Application Architecture Components**
- Typically a web-based application architecture comprises 3 core components:
- 1) **Web Browser**:
- The browser or the client-side component or the front-end component is the key component that interacts with the user, receives the input and manages the presentation logic while controlling user interactions with the application.
- User inputs are validated as well, if required.
- 2) **Web Server:**
- The web server also known as the backend component or the server-side component handles the business logic and processes the user requests by routing the requests to the right component and managing the entire application operations.
- It can run and oversee requests from a wide variety of clients.

3) **Database Server**:

- The database server provides the required data for the application.
- It handles data-related tasks.
- In a multi-tiered architecture, database servers can manage business logic with the help of stored procedures.

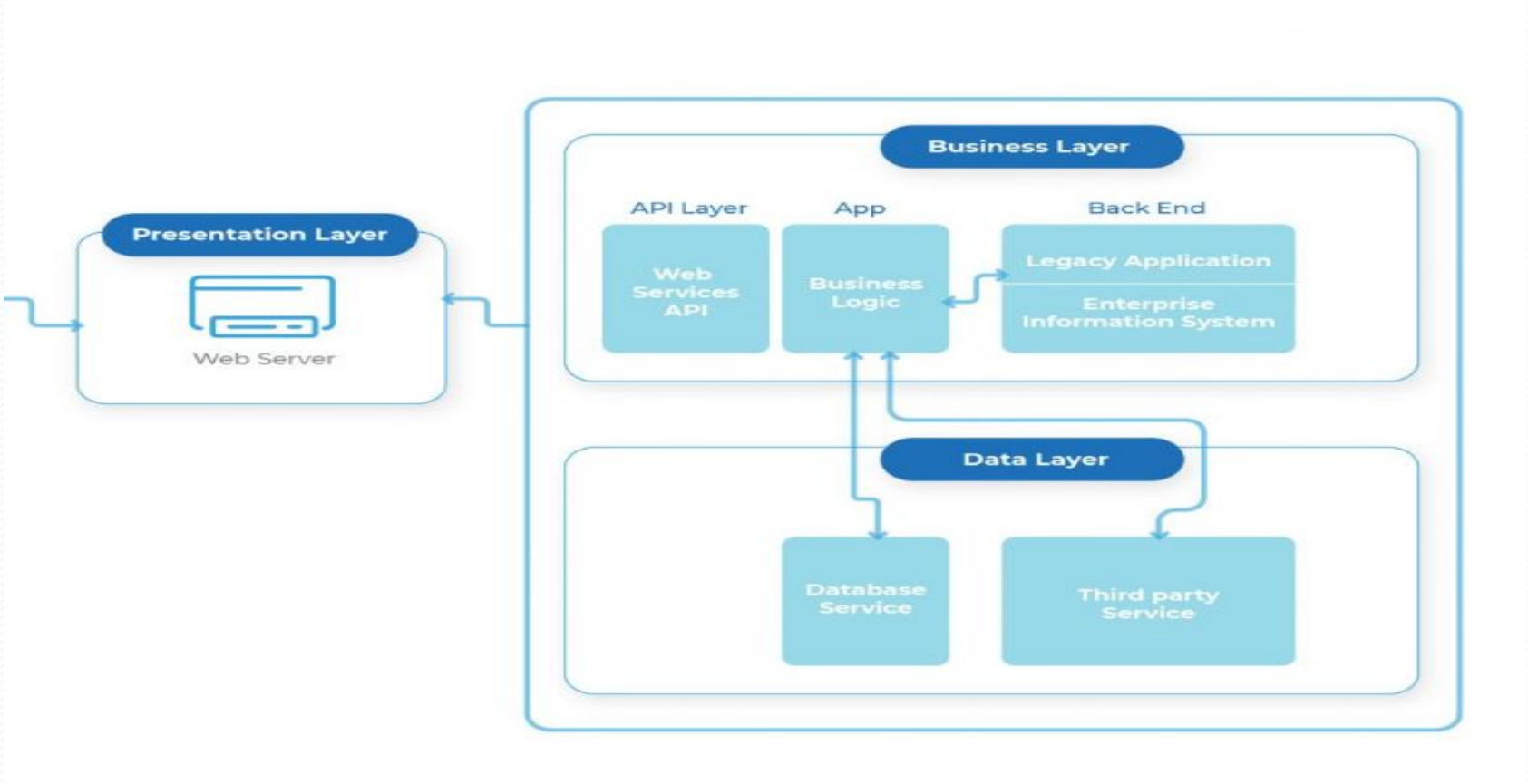
# Web application Architecture:

- In a traditional 2-Tier architecture, there are two components namely the client side system or the user interface and a backend system which is usually a database server.
- Here the business logic is incorporated into the user interface or the database server.
- The downside of 2-tier architecture is that with an increased number of users, the performance decreases.
- Moreover, the direct interaction of the database and the user device also raises some security concerns.
- Railway reservation systems, content management systems are a couple of applications that are usually built using this architecture.

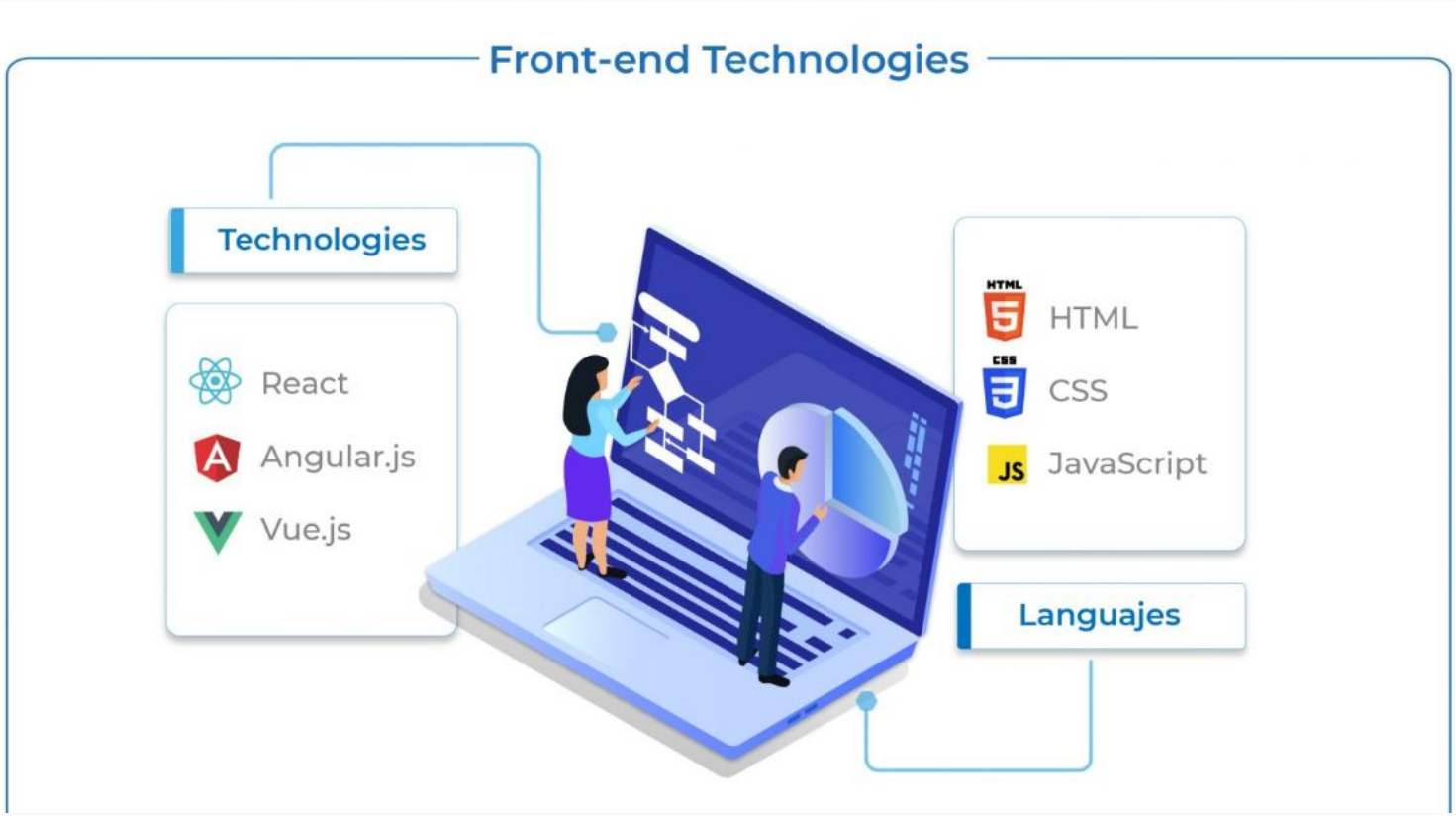


- There are three layers of a 3-Tier architecture:

1. Presentation layer / Client Layer
2. Application Layer / Business Layer
3. Data Layer



- Presentation Layer: Client-side Component (Front-end)
- The client-side component of a web application architecture enables users to interact with the server and the backend service via a browser.
- The code resides in the browser, receives requests and presents the user with the required information.
- This is where UI/UX design, dashboards, notifications, configurational settings, layout and interactive elements come into the picture.


Front-end Technologies
Technologies
React
Angular.js
Vue.js
HTML
CSS
JavaScript
Languajes

- Application Layer: Server-side Component (Back-end)
- The server-side component is the key component of the web application architecture that receives user requests, performs business logic and delivers the required data to the front-end systems.
- It contains servers, databases, web services etc.
- Web Server
- The web server uses HyperText Transfer Protocol (HTTP) along with other protocols to listen to user requests via a browser.
- It processes them by applying business logic and delivering the requested content to the end-user.

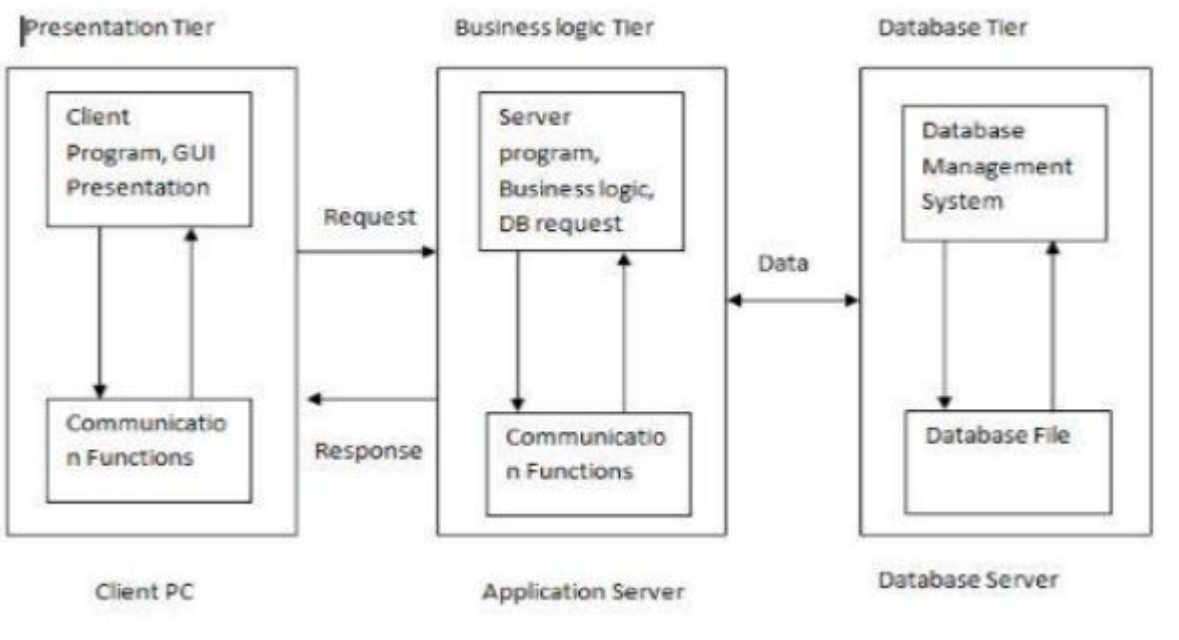
# Back-end Technologies

**Technologies**

- Node.Js
- Java
- Python
- PHP Laravel
- Go
- Ruby
- .Net

- Data Layer: Database
- A database is a key component of a web application that stores and manages information for a web app.
- Using a function, you can search, filter and sort information based on user request and present the required info to the end user.
- They allow role-based access to maintain data integrity.
- While choosing a database for your architecture of web app, the size, speed, scalability and structure are the four aspects that require your consideration.
- For structured data, SQL-based databases are a good choice. it suits financial apps wherein data integrity is a key requirement.

- In 3 tier architecture, there are 3 components:
- Client PC, An Application server and A Database Server.
- The work of server is distributed among application server and database server.
- Application server has the required communication functions.
- The data required by the business logic exists in database server.
- The required data is returned to public servers and then to client PC.

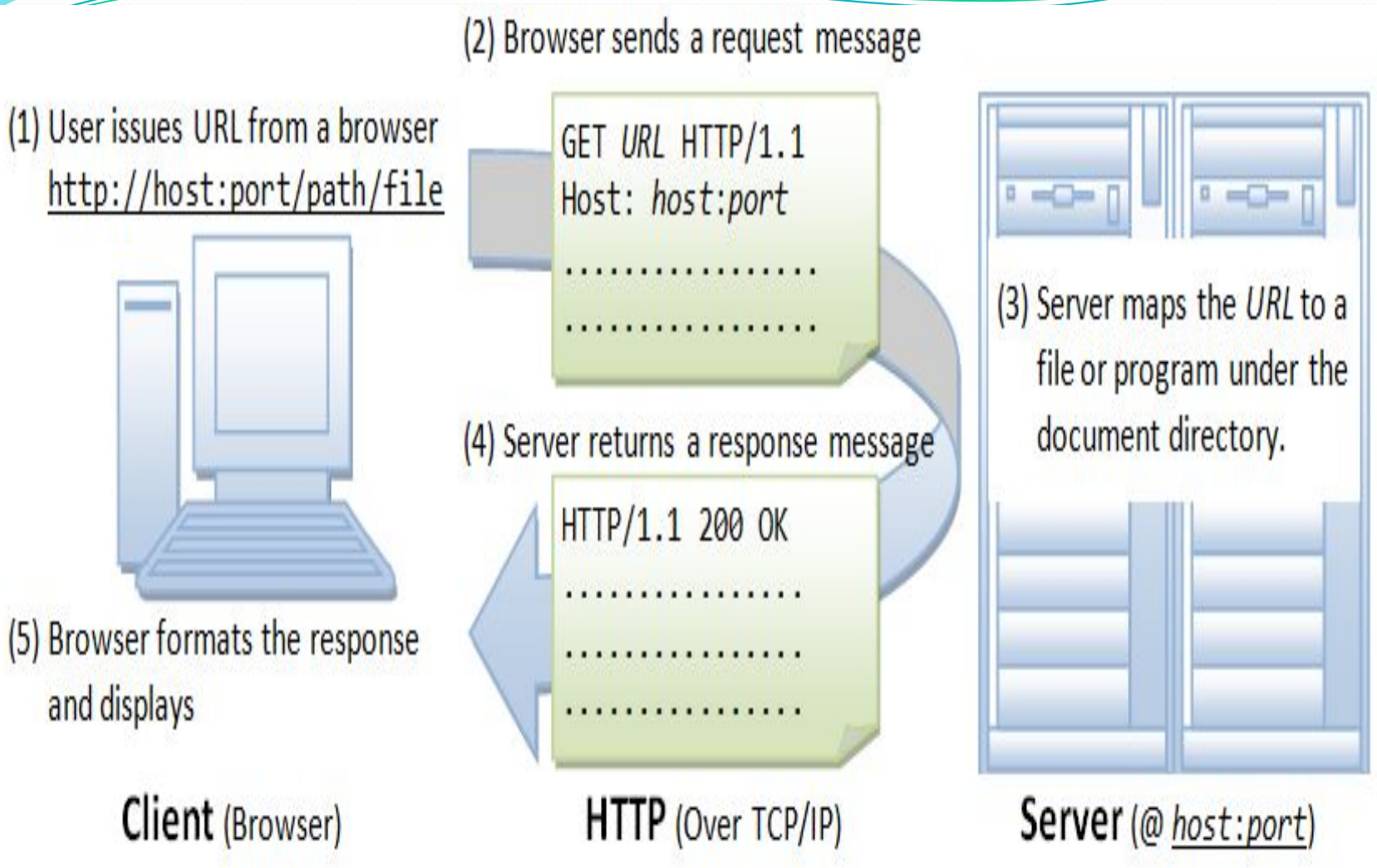

Presentation Tier
Business logic Tier
Database Tier
Client
Program, GUI
Presentation
Server
program,
Business logic,
DB request
Database
Management
System
Request
Data
Communicatio
n Functions
Response
Communicatio
n Functions
Database File
Client PC
Application Server
Database Server

- **Advantages:**
- Improved Data Integrity
- High Degree of Flexibility in deployment platform and configurations
- Improved security
- High Performance and persistent objects
- Architecture is scalable, adding users and resources in future would be easy
- Maintenance and modifications can be done effectively
- Code and data reusability can be achieved
- **Disadvantages:**
- 3 tier architecture is complex compared to 1 tier and 2 tier
- Cost of network maintenance and deployment is greater than 1 tier and 2 tier

# Web System Architecture

- A web application (or webapp), unlike standalone application, runs over the Internet.
- Examples of webapps are google, amazon, ebay, facebook and twitter. A webapp is typically a 3-tier (or multi-tier) client-server application, typically involving a database.
- Webapps run on HTTP Application Protocol over TCP/IP.
- Users access an webapp via a web browser (HTTP client).

- **Browser  Server Communication:**
- Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. http://www.test101.com/index.html, the browser turns the URL into a *request message* and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:



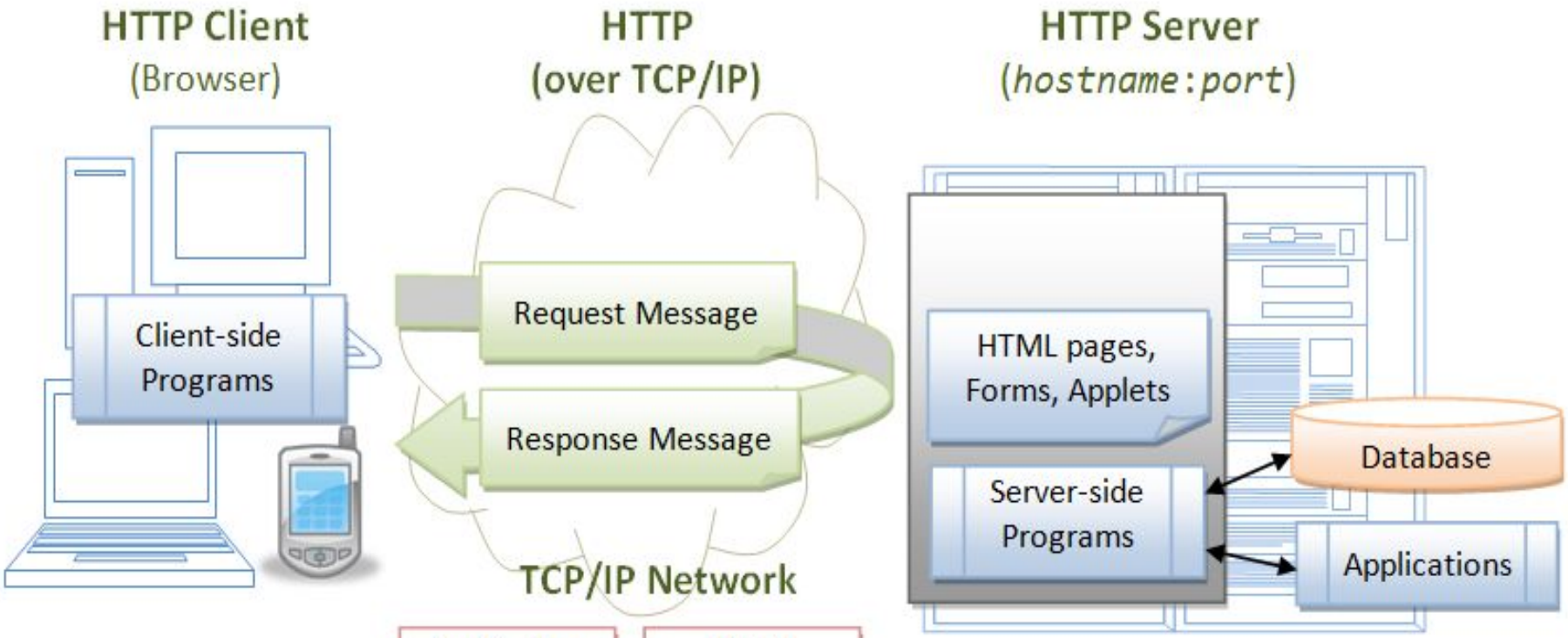

**Browser Server Communication**

- **Uniform Resource Locator (URL)**
- A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web.
- **URL has the following syntax:**
- protocol://hostname:port/path-and-file-name
- **There are 4 parts in a URL:**
- Protocol: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
- Hostname: The DNS domain name (e.g., www.test101.com) or IP address (e.g., 192.128.1.2) of the server.
- Port: The TCP port number that the server is listening for incoming requests from the clients.
- Path-and-file-name: The name and location of the requested resource, under the server document base directory.
- For example, in the URL http://www.test101.com/docs/index.html, the communication protocol is HTTP;
- the hostname is www.test101.com.
- The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP.
- The path and file name for the resource to be located is "/docs/index.html".

- For example, in the URL http://www.test101.com/docs/index.html, the communication protocol is HTTP; the hostname is www.test101.com.
- The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".
- Other examples of URL are:
- ftp://www.ftp.org/docs/test.txt mailto:user@test101.com news:soc.culture.Singapore telnet://www.test101.com/

**Web Application Components:**

- A web database application requires five components, as illustrated below:
- HTTP Server: E.g., Apache HTTP Server, Apache Tomcat HTTP Server, Microsoft IIS, and etc.
- HTTP Client (or Web Browser): E.g., MSIE, FireFox, Chrome, and etc.
- Database: E.g., Open-source MySQL, Apache Derby, mSQL, SQLite, PostgreSQL, OpenOffice's Base; Commercial Oracle, IBM DB2, SAP SyBase, MS SQL Server, MS Access.
- Client-Side Programs: could be written in HTML Form, JavaScript, VBScript, Flash, and etc.
- Server-Side Programs: could be written in Java Servlet/JSP, ASP, PHP, CGI, and etc.



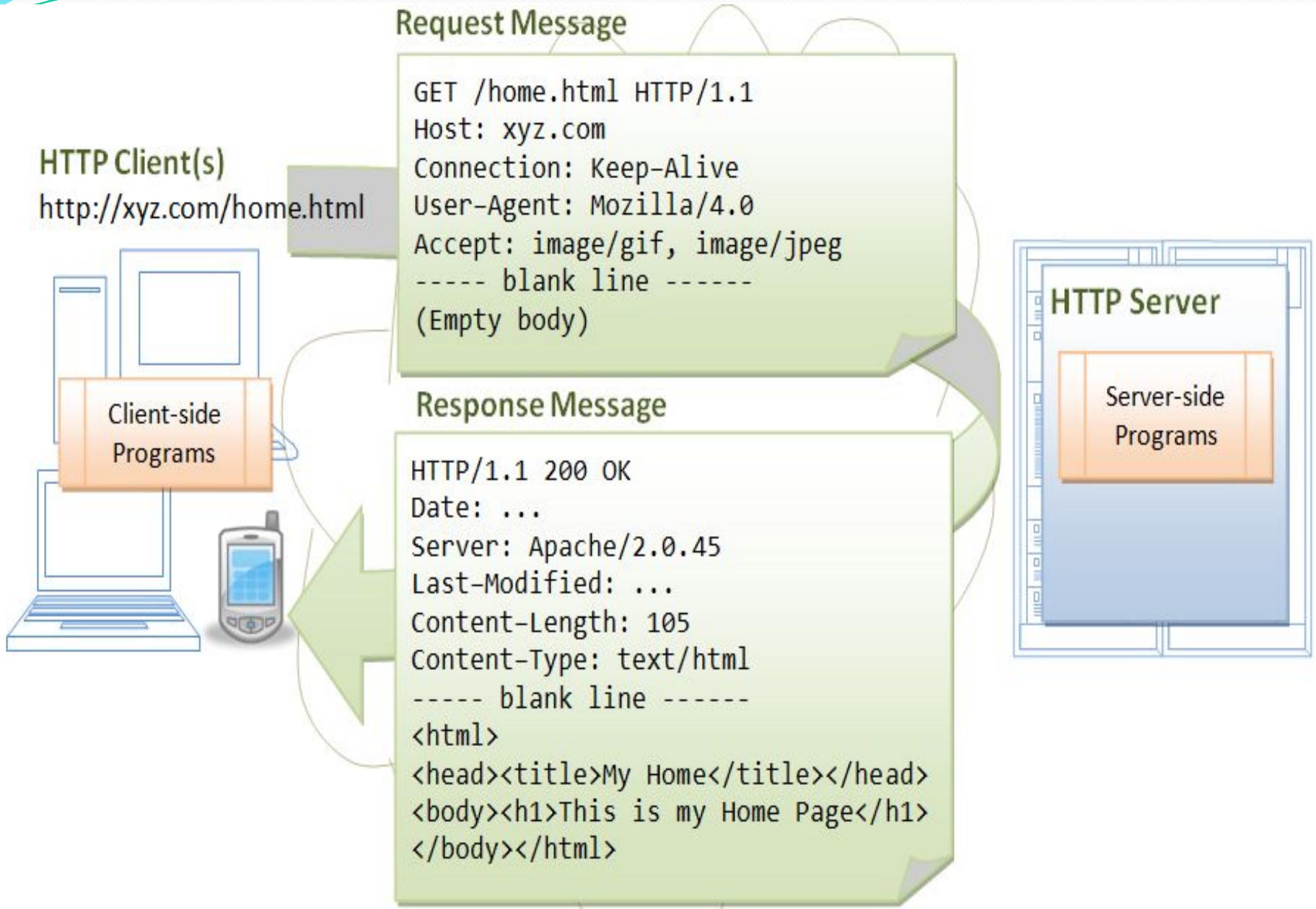

| | |
|---|---|
| Application | **HTTP** |
| Presentation | SSL |
| Session | |
| Transport | **TCP** |
| Network | **IP** |
| Data Link | IEEE 802.11x |
| Physical | |

Multiplexing (Port), Re-transmission

Addressing (IP Address), Routing

- The procedures is:
- A user, via a web browser, issue a URL to an HTTP server to start a webapp.
- A client-side program (such as an HTML form) is loaded into the browser of the client.
- The user fills up the query criteria in the form.
- The client-side program sends the query parameters to a server-side program.
- The server-side program queries the database and returns the query result to the client.
- The client-side program displays the result on the browser.

# Hypertext Transfer Protocol (HTTP)



# Hypertext Transfer Protocol(HTTP)

- Http is a simple application protocol working under a client/server computing environment.
- Client issues a request to a server and then server returns the response.
- The Request is specified in text(ASCII) format, whereas the response is specified in Multipurpose Internet Mail Extensions(MIME) format which defines different types of content types such as text, image and audio.
- The common content types for a server's responses are
- Text/html-text file in html format
- Image/JPEG- image file in JPEG format
- Image/GIF-image file in GIF format

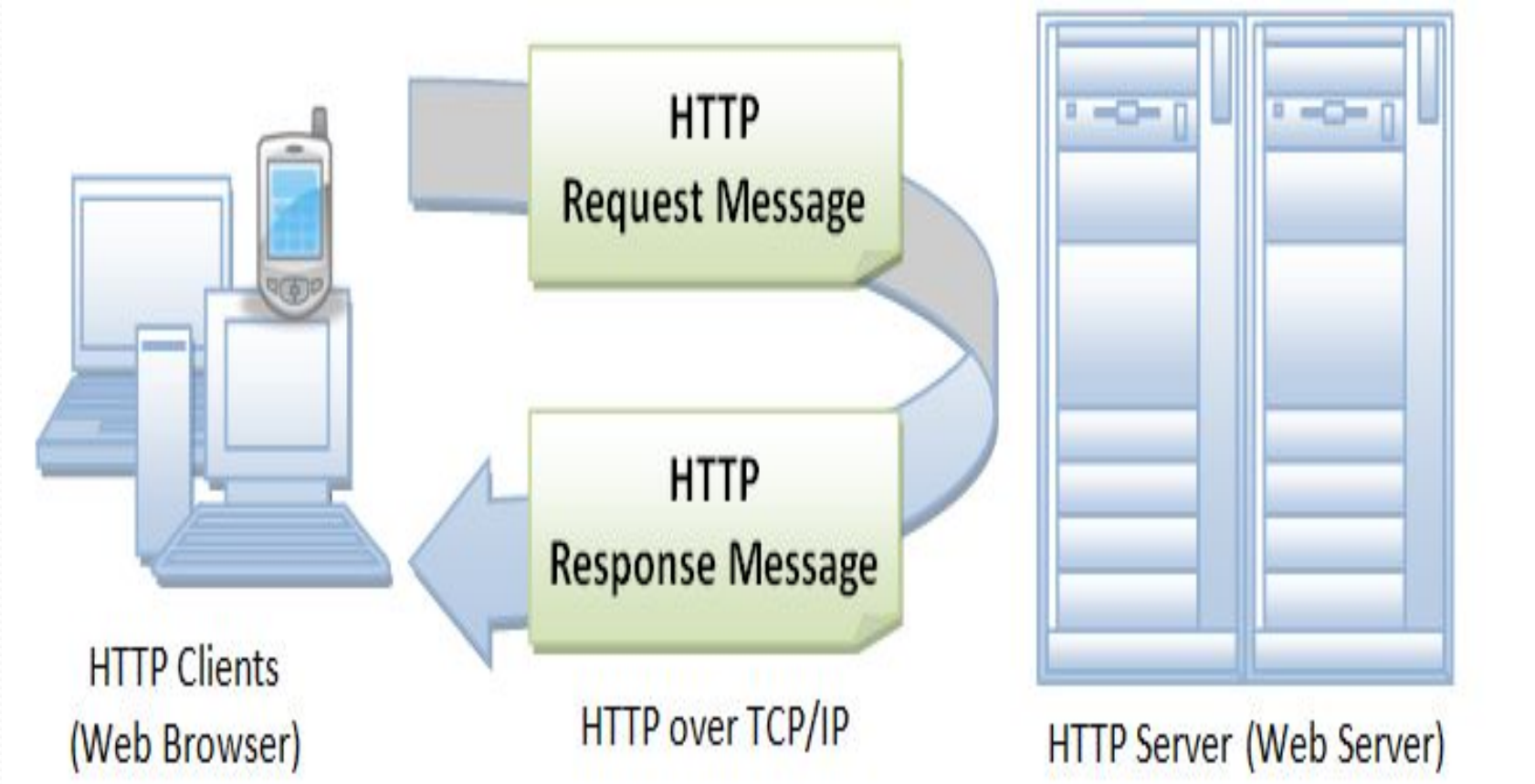
# Characteristics of HTTP

- HTTP is a *request-response client-server* protocol.
  - An HTTP client sends a request message to an HTTP server.
  - The server, in turn, returns a response message.
- In other words, HTTP is a *pull protocol*, the client *pulls* information from the server (instead of server *pushes* information down to the client).
- HTTP is a stateless protocol.
  - In other words, the current request does not know what has been done in the previous requests.

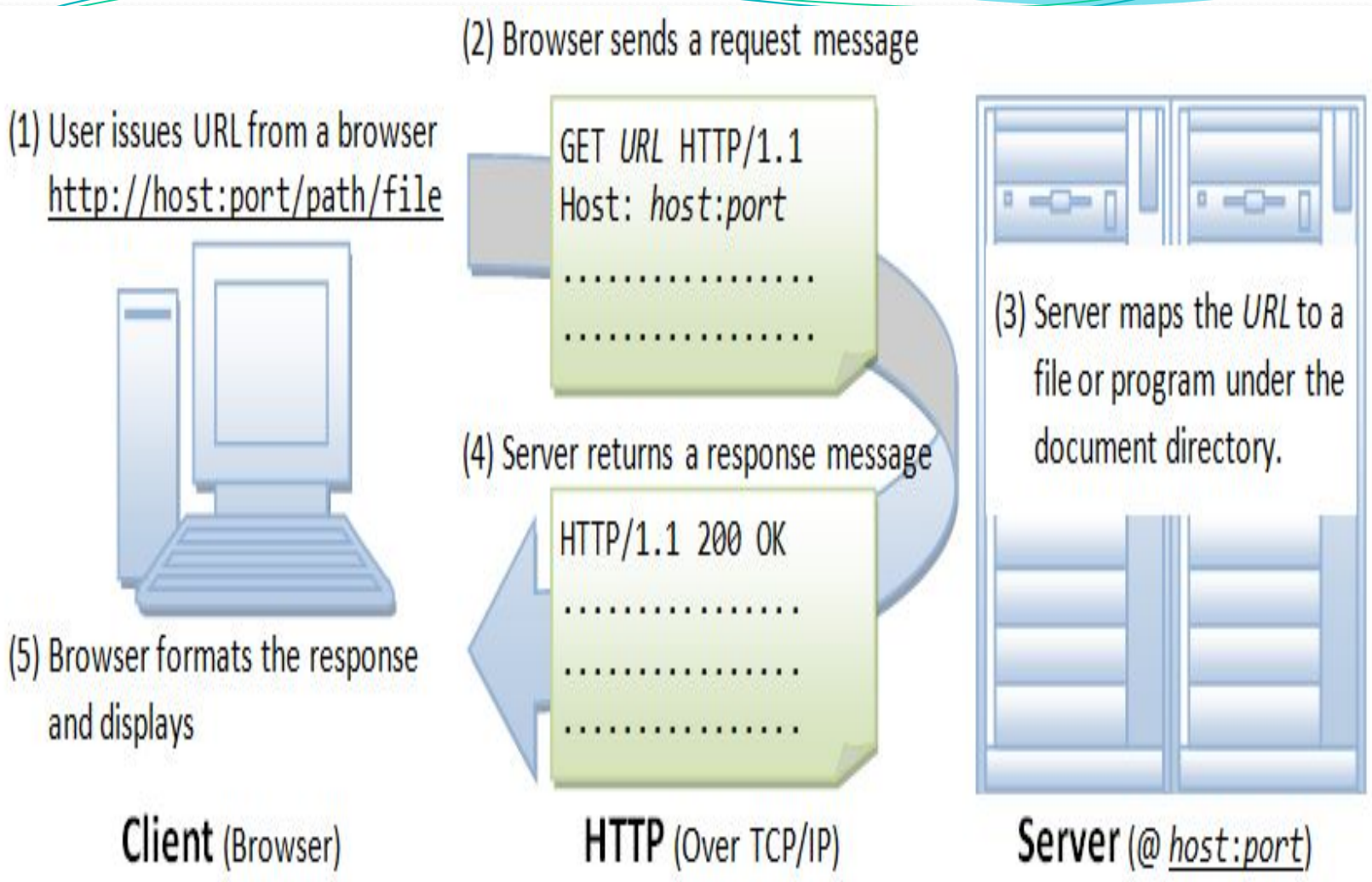
# Working of HTTP

- The web client makes a TCP connection to a web server at port 80.
- An HTTP request consisting of the specific request , required headers and additional data is forwarded to the web server.
- After processing the request, the web server returns an HTTP response consisting of the status , additional headers , and the requested resource such as a web page.

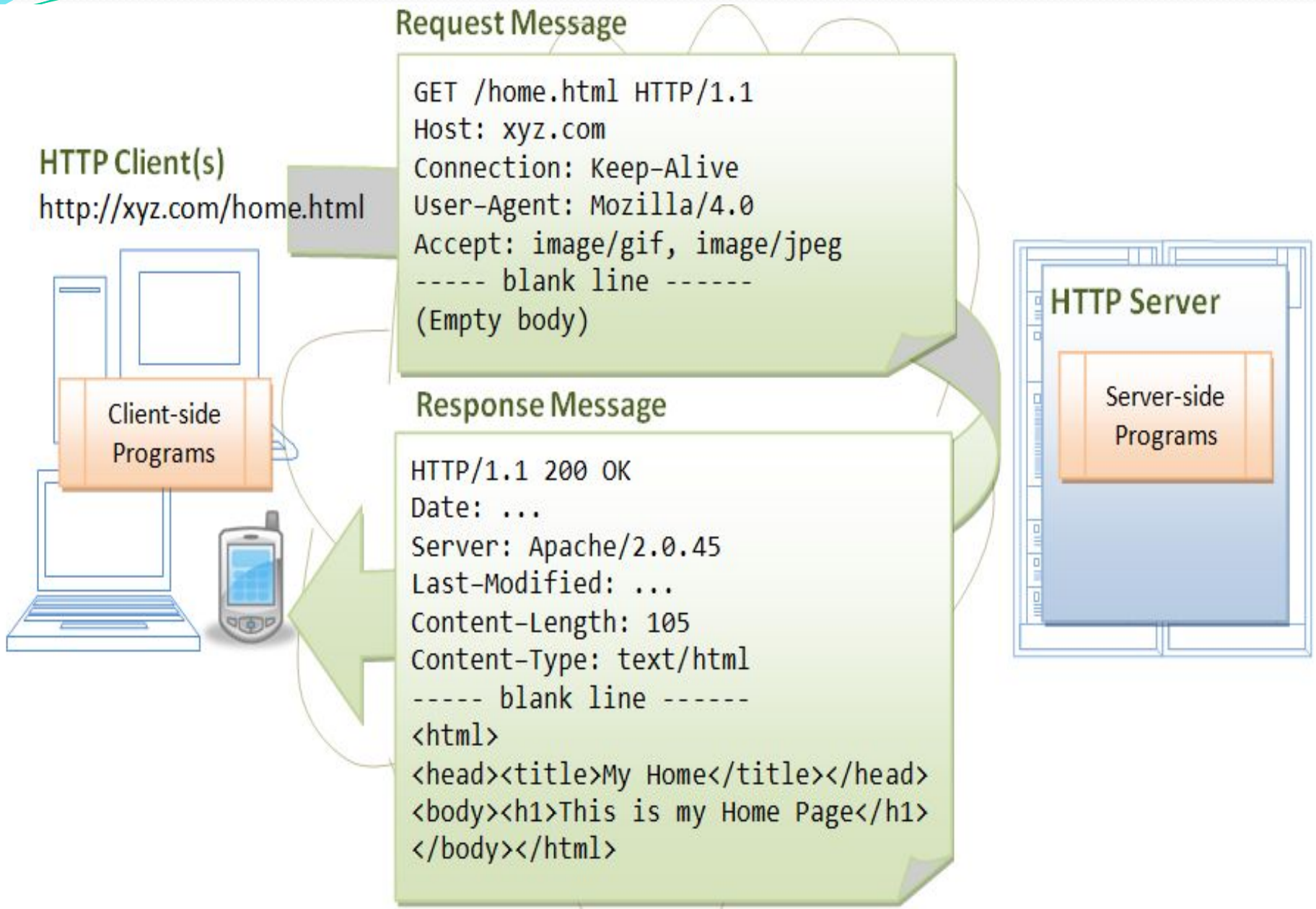
# Working of HTTP cont…



- Whenever  user issue a URL from your browser to get a web resource using HTTP, e.g. http://www.xyz.com/home.html, the browser turns the URL into a *request message* and sends it to the HTTP server.
- The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:


(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
.....
.....
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
.....
.....
.....
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

# Hypertext Transfer Protocol (HTTP)



- **HTTP Protocol**
- Whenever user enter a URL in the address bar of the browser, the browser translates the URL into a request message according to the specified protocol and sends the request message to the server.
- For example, the browser translated the URL http://www.test101.com/doc/index.html into the following request message:

  **GET /doc/index.html** HTTP/1.1
  Host: **www.test101.com**
  Accept: image/gif, image/jpeg, */*
  Accept-Language: en-us
  Accept-Encoding: gzip, deflate
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
  (blank line)

- When this request message reaches the server, the server can take either one of these actions:
- The server interprets the request received, maps the request into a *file* under the server's document directory, and returns the file requested to the client.
- The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.
- The request cannot be satisfied, the server returns an error message.

- An example of the HTTP response message is as shown:

**HTTP/1.1 200 OK**

Date: Sun, 18 Oct 2009 08:56:53

GMT Server: Apache/2.2.14 (Win32)

Last-Modified: Sat, 20 Nov 2004 07:16:26

GMT ETag: "10000000565a5-2c-3e94b66c2e680"

Accept-Ranges: bytes Content-Length: 44

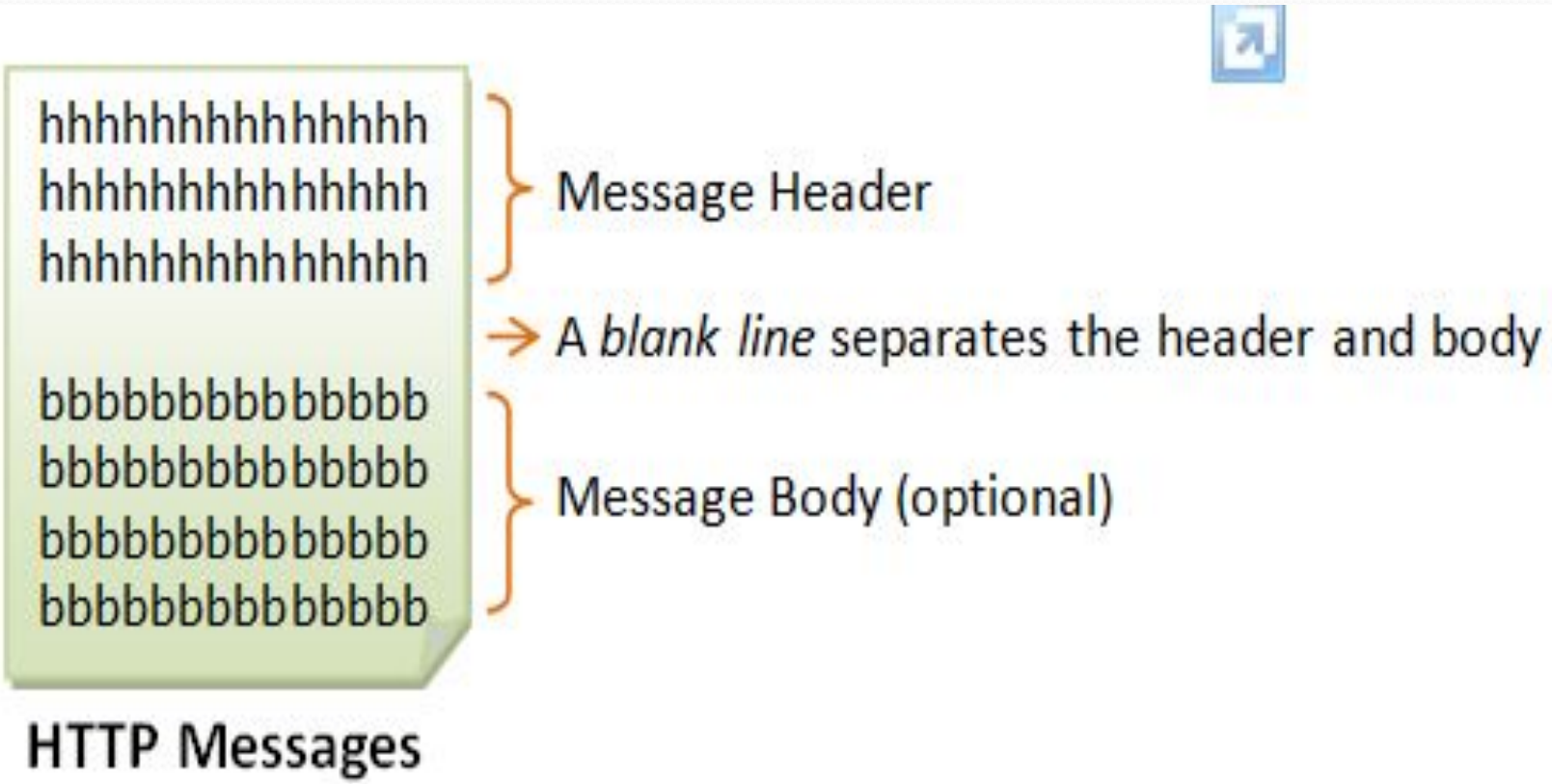
Connection: close

**Content-Type: text/html** X-Pad: avoid browser bug

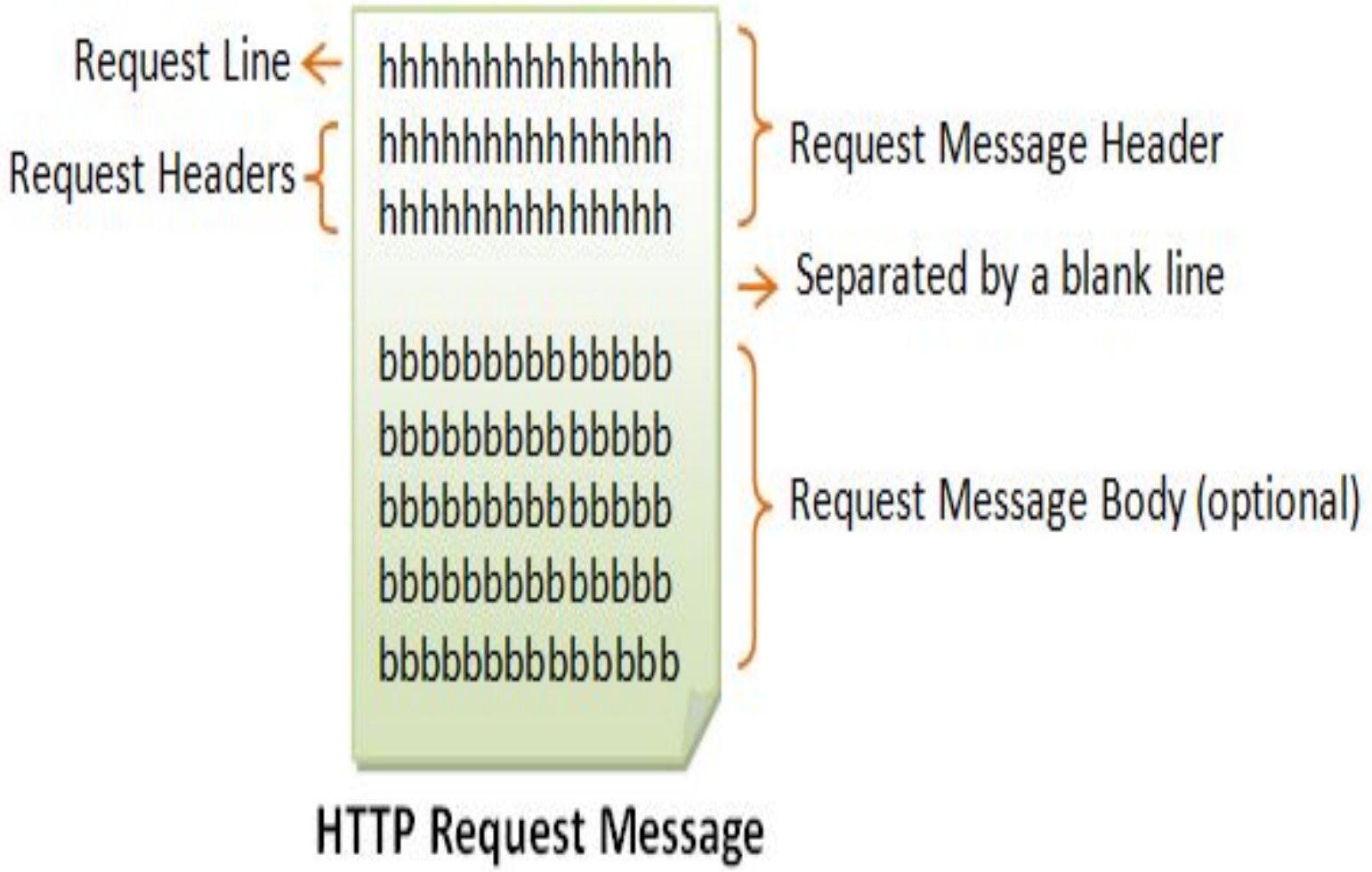
**<html><body><h1>It works!</h1></body></html>**

- The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".

- **HTTP Request and Response Messages :**
- HTTP client and server communicate by sending text messages. The client sends a *request message* to the server. The server, in turn, returns a *response message*.
- An HTTP message consists of a *message header* and an optional *message body*, separated by a *blank line*, as illustrated below:

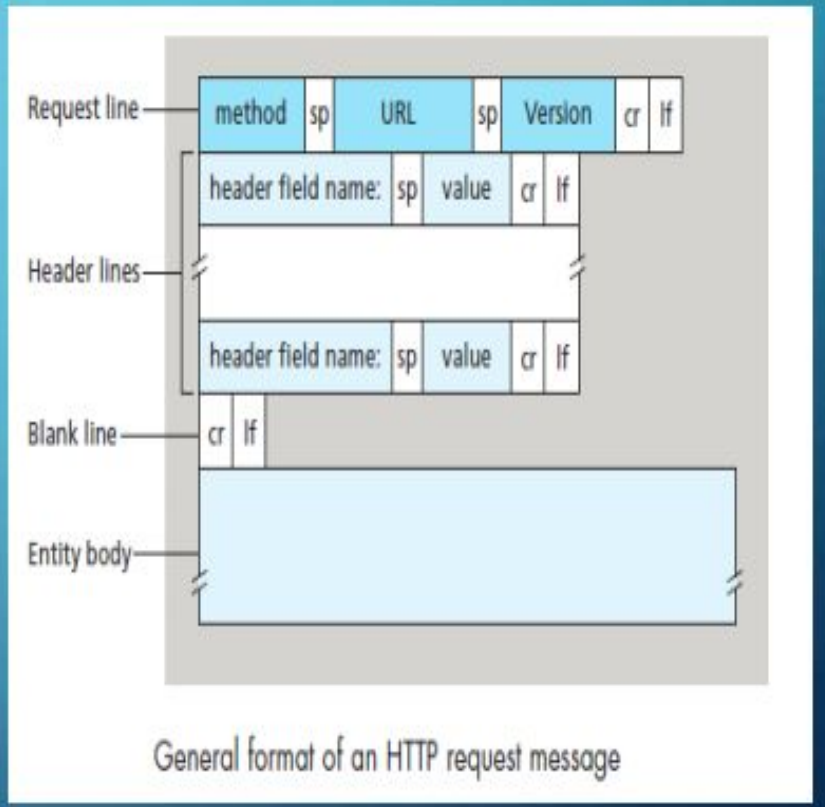

hhhhhhhhhhhhh
hhhhhhhhhhhhh
hhhhhhhhhhhhh
Message Header
A *blank line* separates the header and body
bbbbbbbbbbbbb
bbbbbbbbbbbbb
bbbbbbbbbbbbb
bbbbbbbbbbbbb
Message Body (optional)
HTTP Messages

# HTTP REQUEST MESSAGE



# HTTP REQUEST MESSAGE

- The first line of an HTTP request message is called the **request line**; the subsequent lines are called the **header lines**. The request line has three fields: the method field, the URL field, and the HTTP version field. The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE etc. The great majority of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object identified in the URL field.

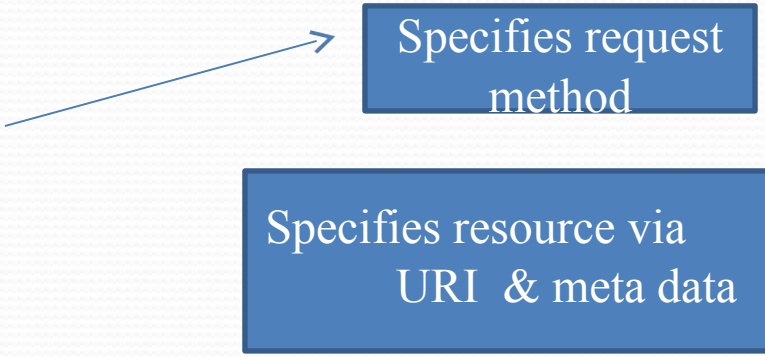



General format of an HTTP request message

- **Request Line**
- The first line of the header is called the *request line*, followed by optional *request headers*.
- The request line has the following syntax:
- *request-method-name request-URI HTTP-version*
- *request-method-name*: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
- *request-URI*: specifies the resource requested.
- *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.
- Examples of request line are:
- GET /test.html HTTP/1.1
- HEAD /query.html HTTP/1.0
- POST /index.html HTTP/1.1

- **Request Headers**
- The request headers are in the form of name:value pairs. Multiple values, separated by commas, can be specified.
- *request-header-name*: *request-header-value1*, *request-header-value2*, ...Examples of request headers are:
- Host: www.xyz.com
- Connection: Keep-Alive
- Accept: image/gif, image/jpeg, */*
- Accept-Language: us-en, fr, cn

# HTTP Request Format




**Request Line**

GET /index.html
HTTP/1.0

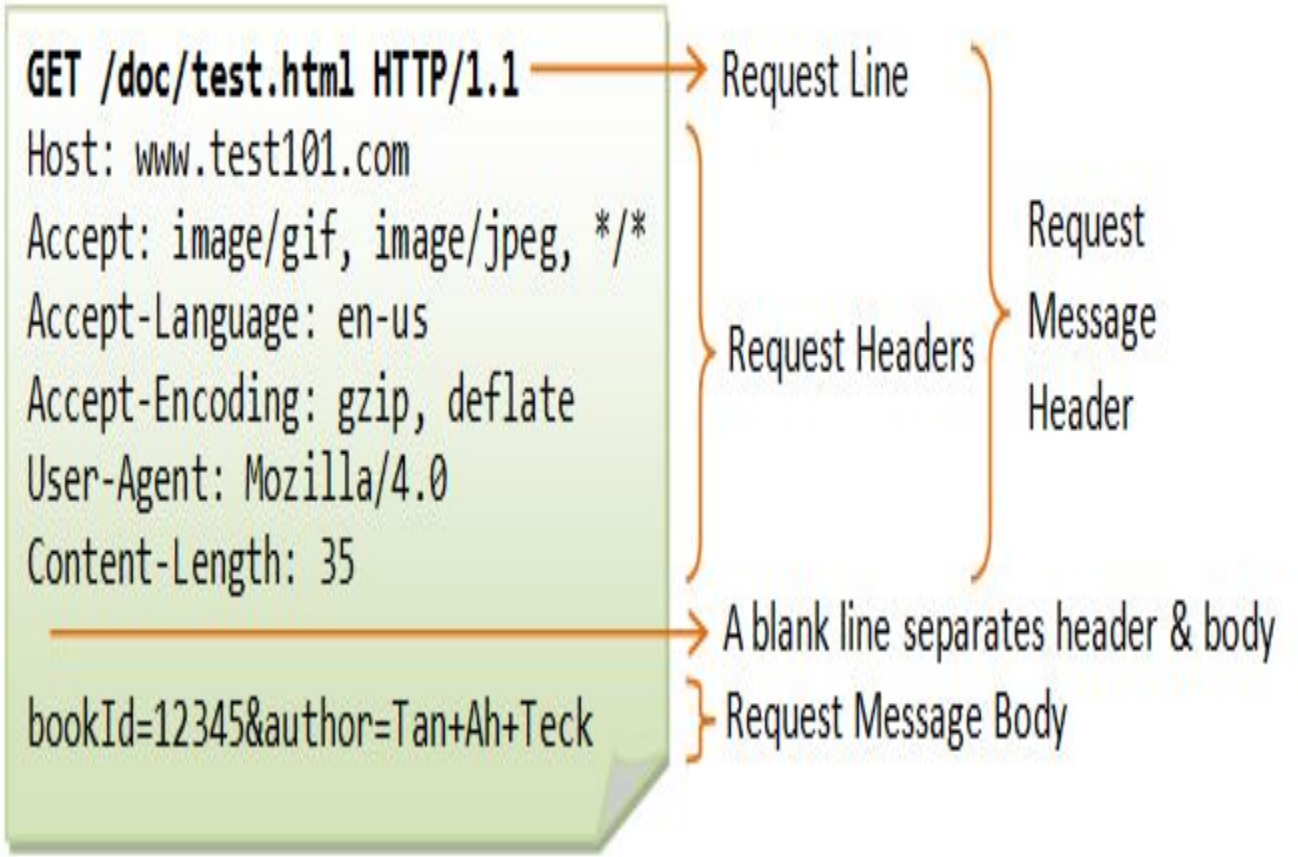
**Header Lines**

Host: www.content-networking.com
Date: BBBBBBBBBBBB
User-Agent: Mozilla/5.0 (en) (WINNT; U)
Accept-Language: en-us

**Carriage Return/Line Feed**

**Message Body**

Content-length: (Message Payload)


GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
bookId=12345&author=Tan+Ah+Teck
Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body

# HTTP 1.0 request methods

| Method | Description |
| --- | --- |
| GET | It gets or retrieves a web page |
| Head | It requests the header information of the web page . Response is the same as that for GET with the body or contents of the web page removed. |
| POST | It post the additional data to the web server in the HTTP request message. The additional data is attached after header. |

## HTTP 1.1 additional 5 methods:

- **PUT** Requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.
- **DELETE** Deletes the specified resource.
- **TRACE** Echoes back the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.
- **OPTIONS** Returns the HTTP methods that the server supports for specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.
- **CONNECT** Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

# Request Methods

- GET
  - whatever information is identified by the Reuest-URI
  - Can Get static content and data produced by a program

- POST
  - Submit information to Web Server
  - Eg: posting to blog, submission of user form…
  - Information is included in message body
  - The actual function depends on request URI

Example
POST/phonebook.cgi.HTTP/1
.0 Date:
User-Agent:
Accept Language:
en-us Content Length:
14
98490 55266

Looks up phone book for the number
Could have been also achieved by
Get
But in that case number would have been in the
Resource URL
Which would have been stored in the log

# Request Methods…contd (ii)

- HEAD
  - Servers response does not include message body
  - Useful for getting resource metadata without transferring the resource
  - Also useful for debugging , checking for validity, accessibility and modification
- PUT
  - Requests a server store the enclosed data under the supplied Request URL.
  - Creates the resource if it does not create
  - Not useful for web publishing (FTP is preferred for security purposes)
- DELETE
  - Removes the Web object
  - Needs to be carefully used for security reasons

# Request Methods…contd (iii)

- TRACE method
  - Invokes a remote appliction layer feedback of the request message
  - Useful for testing what is being received at the server
  - Also possible to forward to intermediaries for debugging purposes
- OPTIONS
  - Requests information about communication options available to server

## Difference between GET and POST Method:

1) **Transmission of Request Parameters**:- In GET the request parameters are transmitted as a query String appended to the request URL . Whereas, in POST the request parameters are transmitted within the body of the request.

2) **URL size**:- If you are using the GET method, you are limited to a maximum of 256 characters in the URL, minus the number of characters in the actual path.

The POST method is not limited by the size of the URL for submitting name/value pairs. These pairs are transferred in the header not in the URL.

3) **Purpose**:- GET method is used only for retrieving data.

Whereas, POST method is used for retrieving as well as saving or updating data, ordering a product, or sending e-mail messages.

- 4) **Caching Ability**:- A GET request is often cacheable. Whereas, a POST request is hardly be cacheable.

5) **Type Of Resource**:- GET method is typically used to access static resources, such as HTML documents and images. Whereas, POST method is typically used to transmit the information that is request-dependent, or when a large amount of complex information must be sent to the server.

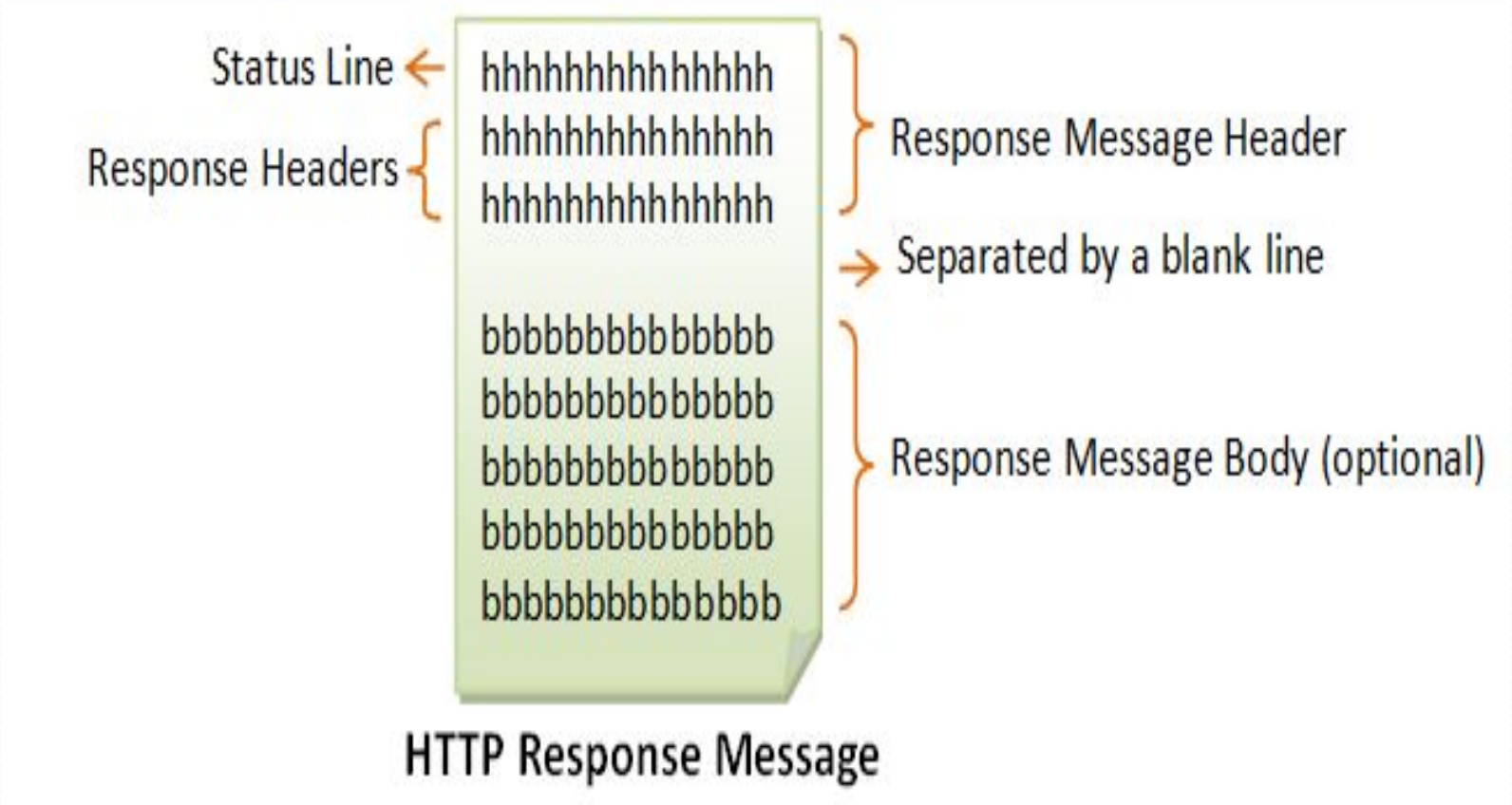
6)In GET method the query string is part of the URL so the resulting web page , whose content depends upon that information , can be bookmarked.
In POST method data is not sent as a part of the URL , so the resulting page can’t be bookmarked.

7. In GET method the Query string is visible .That's not good when sensitive data , such as password. Visited URLS are usually saved in the browser's history list.
8. In GET method visible data in the query string is good for debugging purpose. You can see what is being passed to the server side program.
9. Encryption of data is not used in GET method. Encryption of data is used in POST method.
10. GET method is not reliable. POST method is reliable.
11. Execution of GET method is faster and execution of POST method is slower.

| GET | POST |
|---|---|
| 1) In case of Get request, only **limited amount of data** can be sent because data is sent in header. | In case of post request, **large amount of data** can be sent because data is sent in body. |
| 2) Get request is **not secured** because data is exposed in URL bar. | Post request is **secured** because data is not exposed in URL bar. |
| 3) Get request **can be bookmarked.** | Post request **cannot be bookmarked.** |
| 4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered | Post request is **non-idempotent.** |
| 5) Get request is **more efficient** and used more than Post. | Post request is **less efficient** and used less than get. |

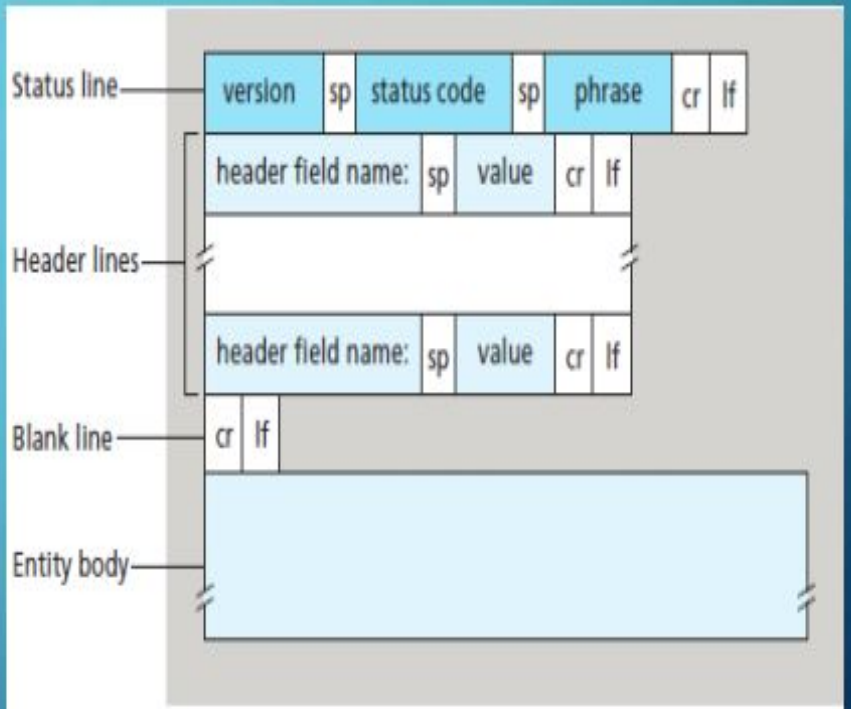
## HTTP Response Message

The format of the HTTP response message is as follows



# HTTP RESPONSE MESSAGES

- It has three sections: an initial **status line**, **header lines**, and then the **entity body**. The entity body contains the requested object itself. The status line has three fields: the protocol version field, a status code, and a corresponding status message.




General format of an HTTP response message

- **Status Line**
- The first line is called the status line, followed by optional response header(s).
- The status line has the following syntax:
- HTTP-version status-code reason-phrase
- HTTP-version: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- status-code: a 3-digit number generated by the server to reflect the outcome of the request.
- reason-phrase: gives a short explanation to the status code.
- Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".
- Examples of status line are:
- HTTP/1.1 200 OK
- HTTP/1.0 404 Not Found
- HTTP/1.1 403 Forbidden

- **Response Headers**
- The response headers are in the form name:value pairs:
- *response-header-name*: *response-header-value1*, *response-header-value2*, ...Examples of response headers are:
- Content-Type: text/html
- Content-Length: 35
- Connection: Keep-Alive
- Keep-Alive: timeout=15, max=100
- The response message body contains the resource data requested.

- Response-Header = Location ;
  Server ;
  | WWW-Authenticate

# HTTP Response Format

| | |
|---|---|
| **Status line** | HTTP/1.0 200 OK |
| **Header Lines** | Date: BBBBBBBBBBBB<br>Server: Apache/1.3.12<br>(Unix) Last-Modified:<br>(date) Content Type:<br>text/html |
| **Carriage Return/Line Feed** | |
| **Message Body** | Content-length:<br>(Message<br>Payload) |

Status line with result code and

Specifies server & resource meta data

- The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:
- 1xx: Informational - Not used, but reserved for future use
- 2xx: Success - The action was successfully received, understood, and accepted.
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

# SOME COMMON STATUS CODES AND ASSOCIATED PHRASES

- Some common status codes and associated phrases include:
  - 200 OK: Request succeeded and the information is returned in the response.
  - 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
  - 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.
  - 404 Not Found: The requested document does not exist on this server.
  - 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

| Code | Type | Example Reasons |
|---|---|---|
| 1xx | Informational | Request received, continuing process |
| 2xx | Success | Action successfully received, understood, and accepted |
| 3xx | Redirection | Further action must be taken to complete the request |
| 4xx | Client error | Request contains bad syntax or cannot be fulfilled |
| 5xx | Server error | Server failed to fulfill an apparently valid request |

**Five types of HTTP result codes.**

- Status-Code = "200" ; OK | "201" ; Created | "202" ; Accepted | "204" ; No Content | "301" ; Moved Permanently | "302" ; Moved Temporarily | "304" ; Not Modified | "400" ; Bad Request | "401" ; Unauthorized | "403" ; Forbidden | "404" ; Not Found | "500" ; Internal Server Error | "501" ; Not Implemented | "502" ; Bad Gateway | "503" ; Service Unavailable

# Status codes HTTP 1.0

- 200 OK Standard response for successful HTTP requests.
- The actual response will depend on the request method used.
- In a GET request, the response will contain an entity corresponding to the requested resource.
- In a POST request the response will contain an entity describing or containing the result of the action.
- 201 Created The request has been fulfilled and resulted in a new resource being created.
- 202 Accepted The request has been accepted for processing, but the processing has not been completed.
- The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.

- 204 No Content The server successfully processed the request, but is not returning any content
- 301 Moved Permanently This and all future requests should be directed to the given URI.
- 302 Found This is an example of industry practice contradicting the standard.
- The HTTP/1.0 specification required the client to perform a temporary redirect (the original describing phrase was "Moved Temporarily"), but popular browsers implemented 302 with the functionality of a 303 See Other
- 304 Not Modified Indicates that the resource has not been modified since the version specified by the request headers If-Modified-Since or If-Match.
- This means that there is no need to retransmit the resource, since the client still has a previously-downloaded copy.

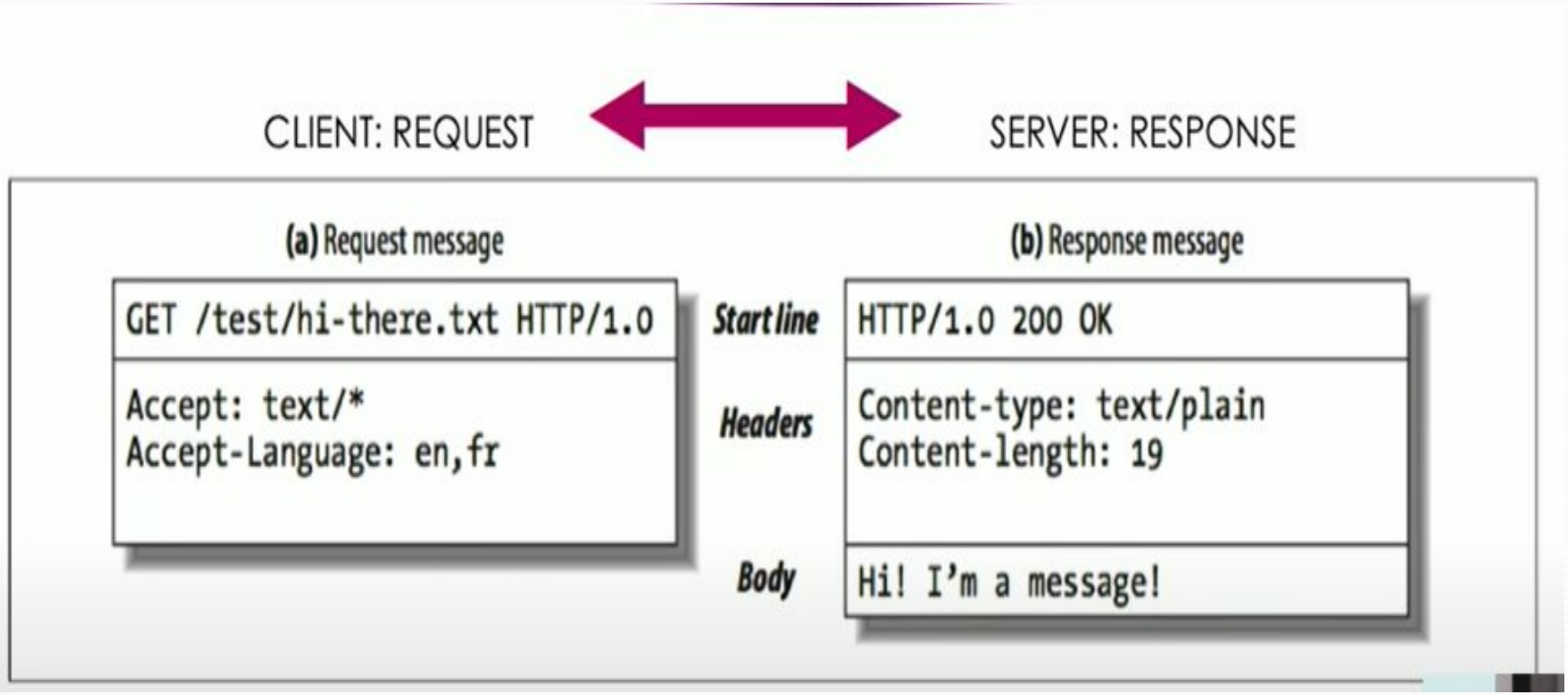
- 400 Bad Request The request cannot be fulfilled due to bad syntax.
- 401 Unauthorized Similar to *403 Forbidden*, but specifically for use when authentication is required and has failed or has not yet been provided.
- The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.

# Result Code and Phrase

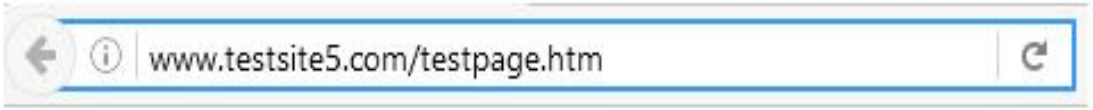
- 1xx: Informational – Not Done Yet
- 2xx: Success – You win
- 3xx:Redirection-You lose but try again
- 4xx:Client Error – You lose, your fault
- 5xx:Server Error – You lose, my bad

200 OK
204 No Content
300 Mutiple Choices
301 Moved Permanently
302 Moved Temporarily
304 Not Modified
400 Bad Request
401 Unauthorized
404 Not Found
500 Internal Server Error

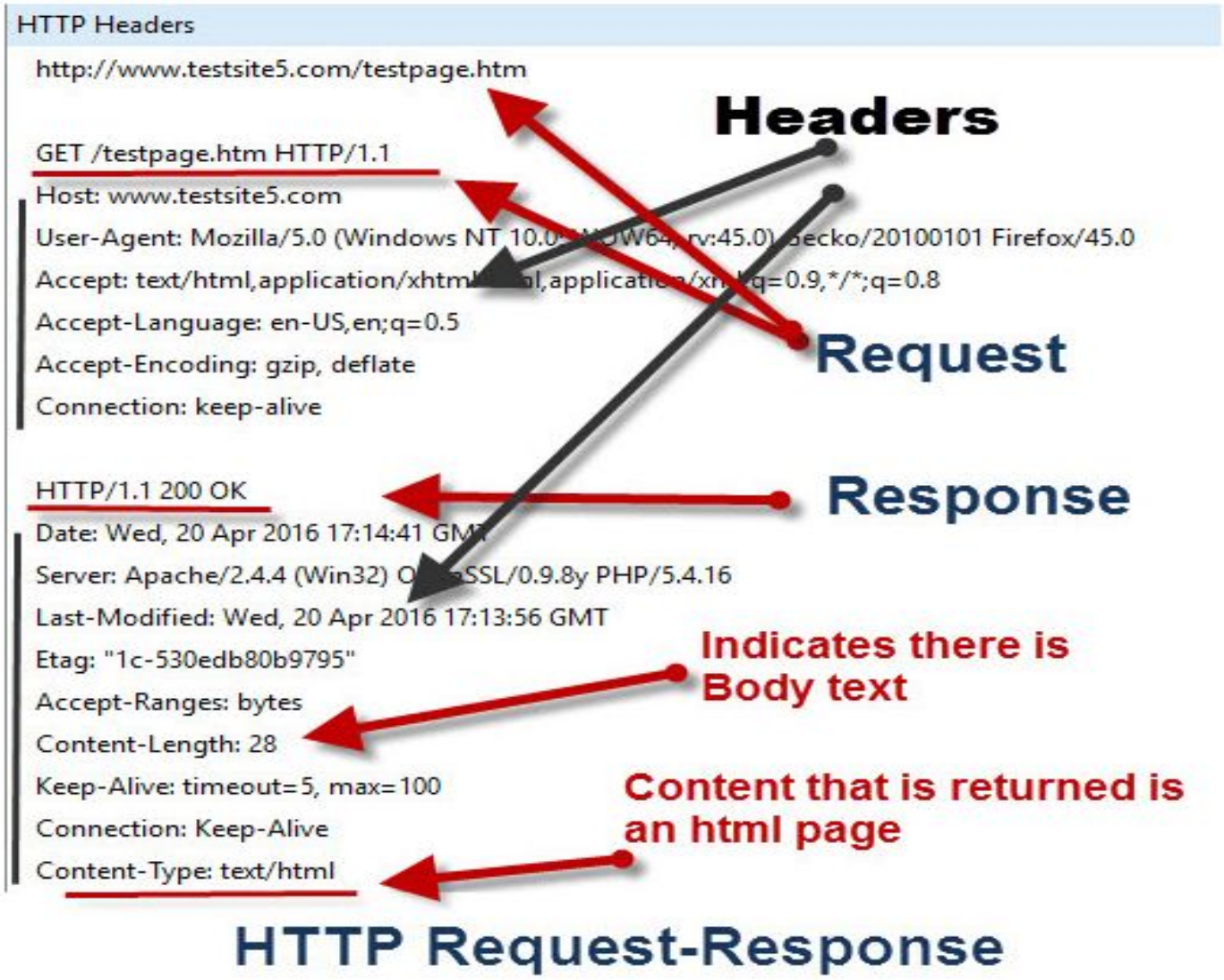
# HTTP Request and Response:



- **Request Response Example**
- We are going to examine he requests and response when we access a simple web page (testpage.htm)
- Here is what I enter in the browser address bar:
-




and this is the response that the browser displays:

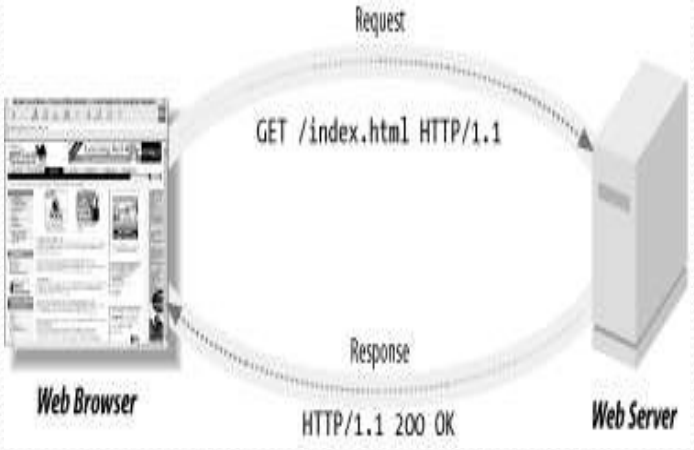

HTTP Headers
http://www.testsite5.com/testpage.htm
Headers
GET /testpage.htm HTTP/1.1
Host: www.testsite5.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Request
HTTP/1.1 200 OK
Response
Date: Wed, 20 Apr 2016 17:14:41 GMT
Server: Apache/2.4.4 (Win32) OpenSSL/0.9.8y PHP/5.4.16
Last-Modified: Wed, 20 Apr 2016 17:13:56 GMT
Etag: "1c-530edb80b9795"
Indicates there is Body text
Accept-Ranges: bytes
Content-Length: 28
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content that is returned is an html page
Content-Type: text/html
HTTP Request-Response

# HTTP Summary:

- **What is HTTP (Hypertext Transfer Protocol)?**
- The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems that allows users to communicate data on the World Wide Web.
- What is the purpose of HTTP?
- HTTP was invented alongside HTML to create the first interactive, text-based web browser: the original World Wide Web. Today, the protocol remains one of the primary means of using the Internet.

# HTTP

- HTTP stands for **Hypertext Transfer Protocol.**
- HTTP provides a set of rules and standards that govern how information is transmitted on the World Wide Web.
- Computers on the World Wide Web use the HyperText Transfer Protocol to talk with each other
- **http://www.google.co.in**




- The first part of an address (URL) of a site on the Internet, signifying a document written in Hypertext Markup Language (HTML)

- HTTP is a client-server protocol by which two machines communicate using a reliable, connection-oriented transport service such as the TCP.
  - A browser is an *HTTP client* because it sends requests to an *HTTP server* (Web server), which then sends responses back to the client.
  - An HTTP server is a program that sits listening on a machine's port for HTTP requests.
  - The standard (and default) port for HTTP servers to listen on is 80, though they can use any port.
- HTTP can be "implemented on top of any other protocol on the Internet, or on other networks.“
- HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used.” e.g.TCP.

- HTTP is stateless. The lifetime of a connection corresponds to a single request-response sequence

- An HTTP client opens a tcp/ip connection to the server via a socket, transmits a request for a document, then waits for a reply from the server. Once the request-response sequence is completed, the socket is closed.

- There is no "memory" between client connections.

- The pure HTTP server implementation treats every request as if it was brand-new.

- Http pages are stored on your computer and internet caches.

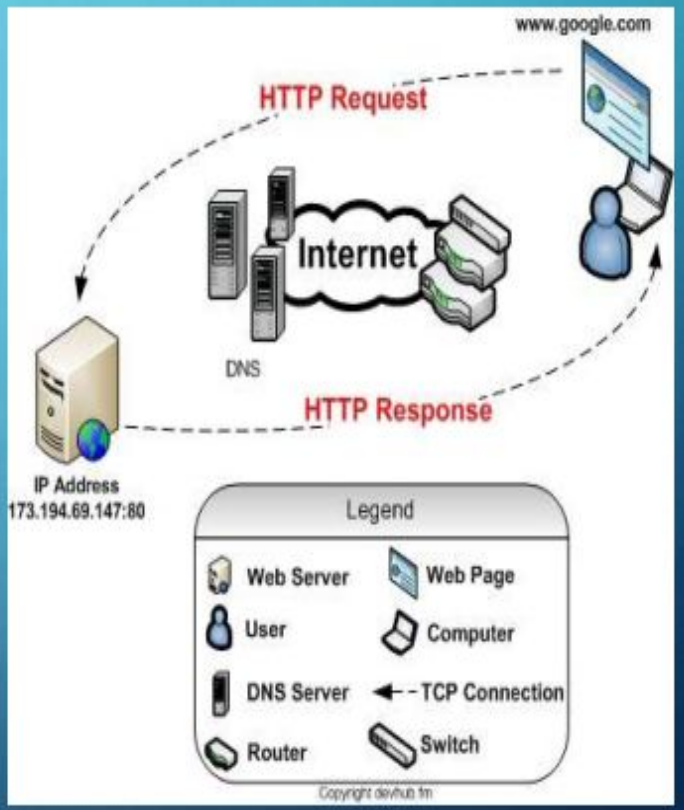
- The pages load faster, but they are stored on systems that you potentially don't have control over.eg: ISP's caching proxy

- How does HTTP work?
- As a request-response protocol, HTTP gives users a way to interact with web resources such as HTML files by transmitting hypertext messages between clients and servers. HTTP clients generally use Transmission Control Protocol (TCP) connections to communicate with servers.

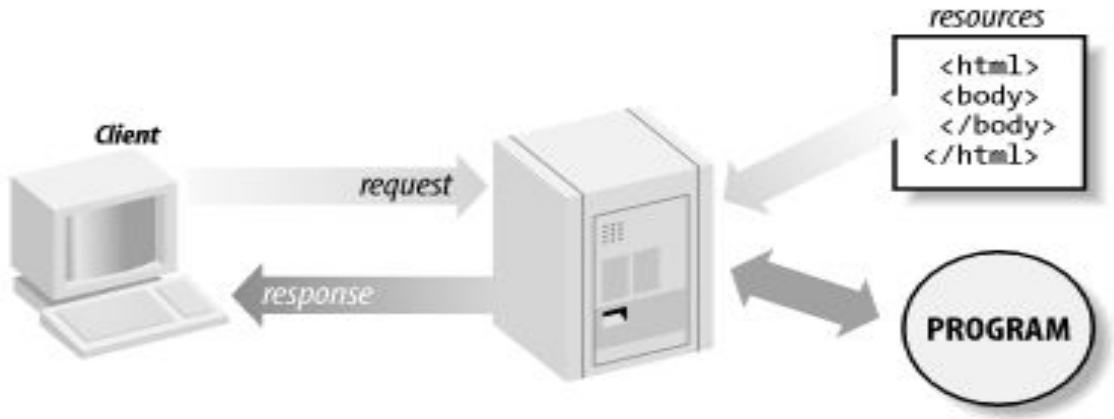
- HTTP utilizes specific request methods in order to perform various tasks. All HTTP servers use the GET and HEAD methods, but not all support the rest of these request methods:

# HOW HTTP WORKS?

- HTTP is implemented in two programs: a client program and a server program, executing on different end systems, talk to each other by exchanging HTTP messages.
- The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces.



- **The HTTP Request/Response Model:**
- HTTP and all extended protocols based on HTTP are based on a very simple communications model.
- how it works: a client, typically a web browser, sends a request for a resource to a server, and the server sends back a response corresponding to the resource (or a response with an error message if it can't process the request for some reason).
- A resource can be a number of things, such as a simple HTML file returned verbatim to the browser or a program that generates the response dynamically.
- This request/response model is illustrated as follows.



# Advantages of HTTP

- Platform independent- Allows Straight cross platform porting.
- No Runtime support required to run properly.
- Usable over Firewalls! Global applications possible.
- Not Connection Oriented- No network overhead to create and maintain session state and information.

# HTTP Limitations

Security Concerns

- Privacy
  - Anyone can see content
- Integrity
  - Someone might alter content. HTTP is insecure since no encryption methods are used.
  - Hence is subject to man in the middle and eavesdropping of sensitive information.
- Authentication
  - Not clear who you are talking with. Authentication is sent in the clear — Anyone who intercepts the request can determine the username and password being used.

- Stateless - Need State management techniques to maintain the information across multiple request-response cycles.

# Hypertext transfer protocol(HTTP)

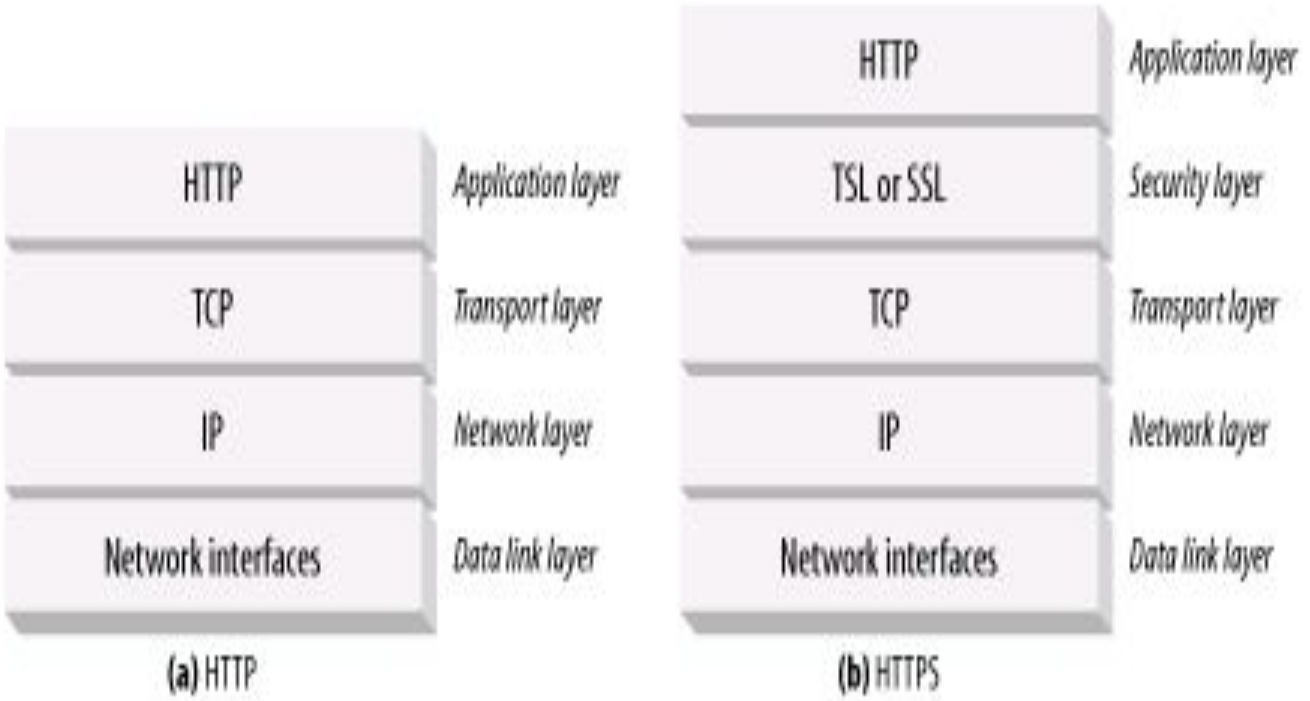
- The HTTP provides a standard for web browsers & servers to communicate.
- HTTP is the foundation of data communication for the WWW.
- HTTP is an application layer network protocol built on top of TCP. HTTP clients & servers communicate via HTTP request & response message.
- Hypertext is structured text that uses logical links(hyper links) between nodes containing text.
- HTTP is the protocol to exchange or transfer hypertext.
- HTTP is called a “stateless protocol” because each command is executed independently, without any knowledge of the commands that

- E.g.- when you enter a URL in your browser, this actually sends an HTTP command to the web server directing it to fetch & transmit the requested web page.

- There are 2 major versions of HTTP:-
  - HTTP/1.0
  - HTTP/1.1

# HTTP Properties:-

- Client/server model
- Simplicity
- Flexibility and content typing
- Connection less
- Stateless
- Meta information


HTTP
Application layer
TCP
Transport layer
IP
Network layer
Network interfaces
Data link layer
(a) HTTP
HTTP
Application layer
TSL or SSL
Security layer
TCP
Transport layer
IP
Network layer
Network interfaces
Data link layer
(b) HTTPS


http
HTTP
TCP
IP
https
HTTP
SSL or TLS
TCP
IP

- **Hypertext Transfer Protocol Secure (HTTPS)** is an extension of the Hypertext Transfer Protocol (HTTP).
- It is used for secure communication over a computer network, and is widely used on the Internet.
- In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS) or, formerly, Secure Sockets Layer (SSL).
- **The protocol is therefore also referred to as HTTP over TLS, or HTTP over SSL.**
- **The principal motivations for HTTPS are authentication of the accessed website, and protection of the privacy and integrity of the exchanged data while in transit.**

- **It protects against man-in-the-middle attacks, and the bidirectional encryption of communications between a client and server protects the communications against eavesdropping and tampering.**
- The authentication aspect of HTTPS requires a **trusted third party to sign server-side digital certificates.**

- **HTTP is Hypertext Transfer Protocol.**
- HTTP offers set of rules and standards which govern how any information can be transmitted on the World Wide Web.
- HTTP provides standard rules for web browsers & servers to communicate.
- HTTP is an application layer network protocol which is built on top of TCP.
- HTTP uses Hypertext structured text which establishes the logical link between nodes containing text.
- It is also known as “stateless protocol” as each command is executed separately, without using reference of previous run command.

- **Advantages of HTTP**
  - HTTP can be implemented with other protocol on the Internet, or on other networks
  - HTTP pages are stored on computer and internet caches, so it is quickly accessible
  - Platform independent which allows cross-platform porting
  - Does not need any Runtime support
  - Usable over Firewalls! Global applications are possible
  - Not Connection Oriented; so no network overhead to create and maintain session state and information

- **Limitations of HTTP**
  - There is no privacy as anyone can see content
  - Data integrity is a big issue as someone can alter the content. That's why HTTP protocol is an insecure method as no encryption methods are used.
  - Not clear who you are talking about. Anyone who intercepts the request can get the username and password.

- **Key Difference between HTTP and HTTPS**
  - HTTP lacks a security mechanism to encrypt the data, whereas HTTPS provides SSL or TLS Digital Certificate to secure the communication between server and client.
  - HTTP operates at the Application Layer, whereas HTTPS operates at Transport Layer.
  - HTTP by default operates on port 80, whereas HTTPS by default operates on port 443.
  - HTTP transfers data in plain text, while HTTPS transfers data in cipher text (encrypt text).
  - HTTP is fast as compared to HTTPS because HTTPS consumes computation power to encrypt the communication

- **What is HTTPS?**
- HTTPS stands for Hyper Text Transfer Protocol Secure.
- It is highly advanced and secure version of HTTP.
- It uses the port no. 443 for Data Communication.
- It allows the secure transactions by encrypting the entire communication with SSL.
- It is a combination of SSL/TLS protocol and HTTP.
- It provides encrypted and secure identification of a network server.
- HTTP also allows you to create a secure encrypted connection between the server and the browser. It offers the bi-directional security of Data.
- This helps you to protect potentially sensitive information from being stolen.
- In HTTPS protocol SSL transactions are negotiated with the help of key-based encryption algorithm.
- This key is generally either 40 or 128 bits in strength.

- **Advantages of HTTPS**
  - In most cases, sites running over HTTPS will have a redirect in place. Therefore, even if you type in HTTP:// it will redirect to an https over a secured connection
  - It allows users to perform secure e-commerce transaction, such as online banking.
  - SSL technology protects any users and builds trust
  - An independent authority verifies the identity of the certificate owner. So each SSL Certificate contains unique, authenticated information about the certificate owner

- **Difference Between HTTP and HTTPS**
- The below table demonstrates what is difference between HTTP and HTTPS:

| Parameter | HTTP | HTTPS |
|---|---|---|
| Protocol | It is hypertext transfer protocol. | It is hypertext transfer protocol with secure. |
| Security | It is less secure as the data can be vulnerable to hackers. | It is designed to prevent hackers from accessing critical information. It is secure against such attacks. |
| Port | It uses port 80 by default | It was use port 443 by default. |
| Starts with | HTTP URLs begin with http:// | HTTPs URLs begin with https:// |
| Used for | It's a good fit for websites designed for information consumption like blogs. | If the website needs to collect the private information such as credit card number, then it is a more secure protocol. |
| Scrambling | HTTP does not scramble the data to be transmitted. That's why there is a higher chance that transmitted information is available to hackers. | HTTPS scrambles the data before transmission. At the receiver end, it descrambles to recover the original data. Therefore, the transmitted information is secure which can't be hacked. |

| | | |
|---|---|---|
| Domain Name Validation | HTTP website do not need SSL. | HTTPS requires SSL certificate. |
| Data encryption | HTTP website doesn't use encryption. | HTTPS websites use data encryption. |
| Search Ranking | HTTP does not improve search rankings. | HTTPS helps to improve search ranking. |
| Speed | Fast | Slower than HTTP |
| Vulnerability | Vulnerable to hackers | It Is highly secure as the data is encrypted before it is seen across a network. |

- **Limitations of HTTPS**
  - HTTPS protocol can’t stop stealing confidential information from the pages cached on the browser
  - SSL data can be encrypted only during transmission on the network. So it can’t clear the text in the browser memory
  - HTTPS can increase computational overhead as well as network overhead of the organization

- What is HTTPS?
- The S in HTTPS stands for "secure." HTTPS uses TLS (or SSL) to encrypt HTTP requests and responses, so in the example above, instead of the text, an attacker would see a bunch of seemingly random characters.

- Instead of:
- GET /hello.txt HTTP/1.1
- User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.1 zlib/1.2.11
- Host: www.example.com
- Accept-Language: en
- The attacker sees something like:

- t8Fw6T8UV81pQfyhDkhebbz7+oiwldr1j2gHBB3L3RFTRsQCp aSnSBZ78Vme+DpDVJPvZdZUZHpzbbcqmSW1

- In HTTPS, how does TLS/SSL encrypt HTTP requests and responses?
- TLS uses a technology called public key cryptography:
- there are two keys, a public key and a private key, and the public key is shared with client devices via the server's SSL certificate.
- When a client opens a connection with a server, the two devices use the public and private key to agree on new keys, called session keys, to encrypt further communications between them.

- All HTTP requests and responses are then encrypted with these session keys, so that anyone who intercepts communications can only see a random string of characters, not the plaintext.

# HTTPS

- HTTP stands for Hypertext Transfer Protocol, and is the underlying protocol used throughout the World Wide Web.
- HTTP defines what type of data may be transmitted, how that data is formatted, and how servers should respond to specific commands.
- When a client initiates communication with a server, an HTTP command is sent out requesting access to the desired page.
- HTTP is extremely effective, as its almost universal adoption attests.
- But HTTP isn't very secure. This is because HTTP lacks data encryption and authentication, essentially transmitting data out in the open where anyone can access it.
- **HTTPS (Hypertext Transfer Protocol Secure) solves this problem, creating a secure, encrypted connection between the client and the server.**
- **This secures websites against eavesdropping, tampering and data theft.**

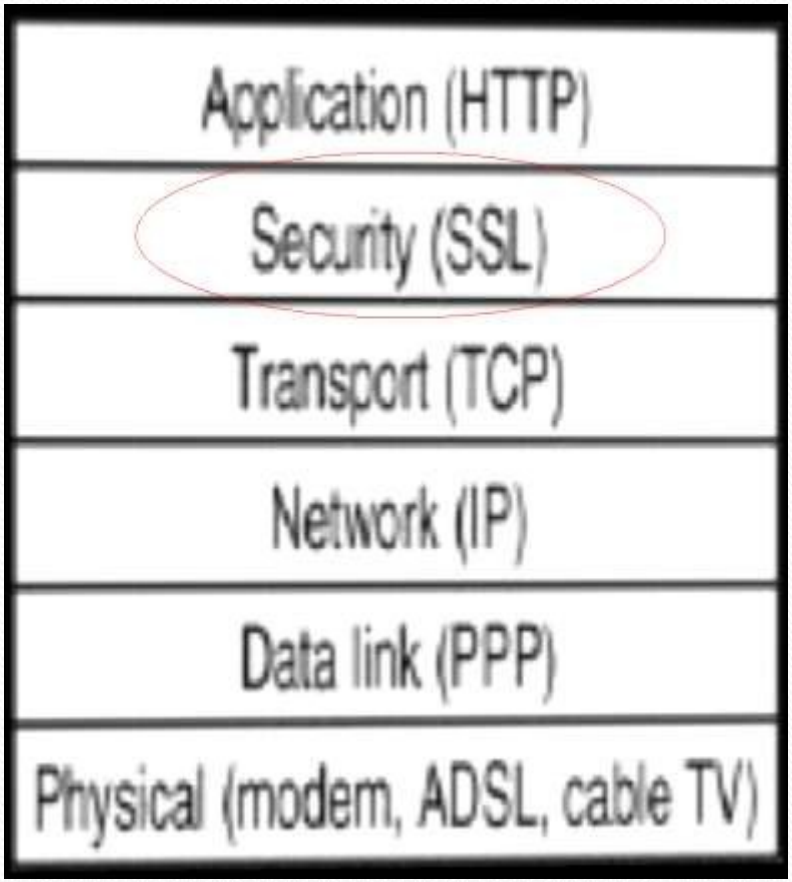
- Originally, HTTPS was designed specifically for e-commerce and business websites that regularly handle sensitive information (such as passwords and credit card details), but new recommendations suggest that every website — even ones that are strictly informational — should use HTTPS.
- The Uniform Resource Identifier (URI) scheme HTTPS has identical usage syntax to the HTTP scheme. However, HTTPS signals the browser to use an added encryption layer of SSL/TLS to protect the traffic.
- SSL/TLS is especially suited for HTTP, since it can provide some protection even if only one side of the communication is authenticated.
- This is the case with HTTP transactions over the Internet, where typically only the server is authenticated (by the client examining the server's certificate).
- HTTPS creates a secure channel over an insecure network.
- This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks, provided that adequate cipher suites are used and that the server certificate is verified and trusted.

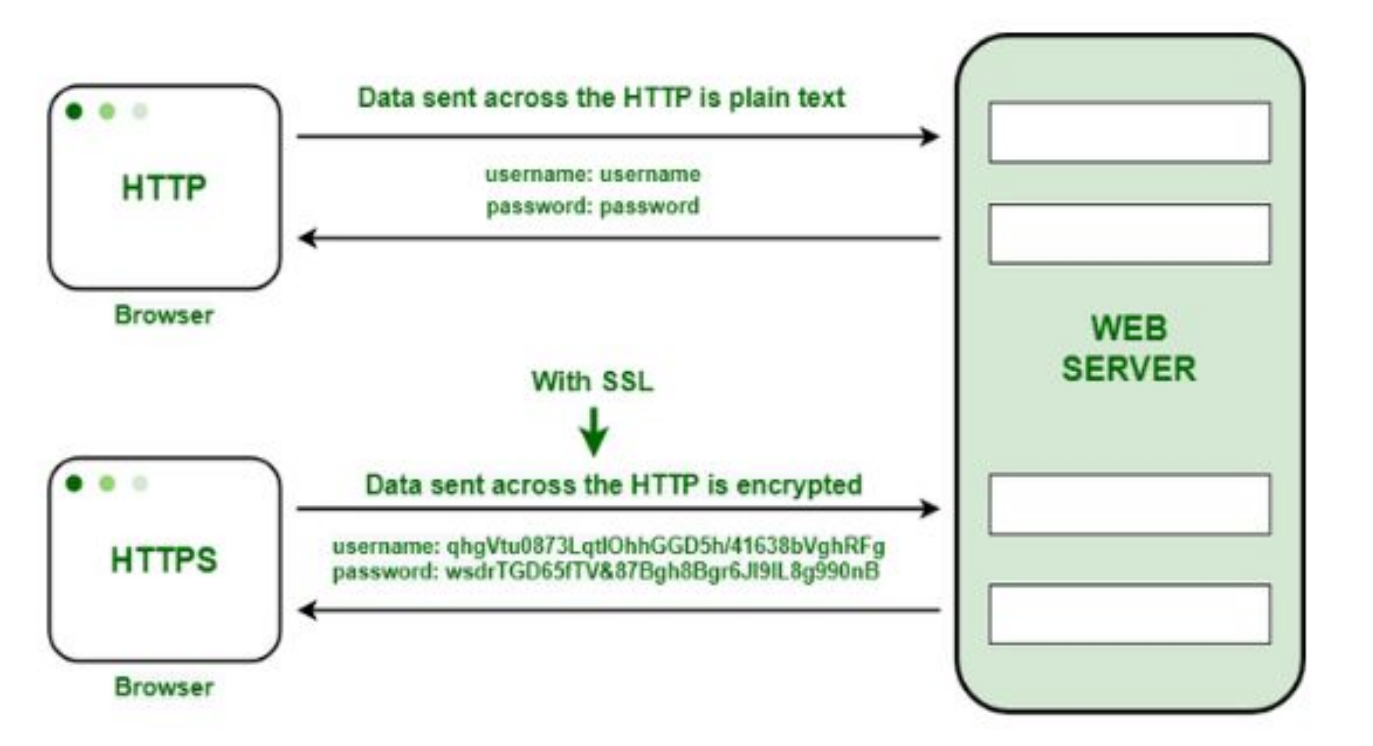
- Because HTTPS piggybacks HTTP entirely on top of TLS, the entirety of the underlying HTTP protocol can be encrypted.
- This includes the request URL (which particular web page was requested), query parameters, headers, and cookies (which often contain identifying information about the user).
- However, because website addresses and port numbers are necessarily part of the underlying TCP/IP protocols, HTTPS cannot protect their disclosure.
- In practice this means that even on a correctly configured web server, eavesdroppers can infer the IP address and port number of the web server, and sometimes even the domain name (e.g. www.example.org, but not the rest of the URL) that a user is communicating with, along with the amount of data transferred and the duration of the communication, though not the content of the communication.

| HTTP | HTTPS |
| --- | --- |
| Hypertext Transfer Protocol | Hypertext Transfer Protocol Secure |
| ⊗ Lacks data encryption and authentication, essentially transmitting data out in the open where anyone can access it. | ✓ Creates a secure, encrypted connection between the client and the server. This secures websites against eavesdropping, tampering, and data theft. |

# HTTPS

- HTTPS stands for Hypertext Transfer Protocol over Secure Socket Layer, or HTTP over SSL.
- SSL acts like a sub layer under regular HTTP application layering.
- HTTPS encrypts an HTTP message prior to transmission and decrypts a message upon arrival.




Data sent across the HTTP is plain text
HTTP
username: username
password: password
Browser
WEB
SERVER
With SSL
Data sent across the HTTP is encrypted
HTTPS
username: qhgVtu0873LqtlOhhGGD5h/41638bVghRFg
password: wsdrTGD65fTV&87Bgh8Bgr6JI9lL8g990nB
Browser

- HTTPS by default uses port 443 as opposed to the standard HTTP port of 80.

- URL's beginning with HTTPS indicate that the connection between client and browser is encrypted using SSL

  e.g.: https://login.yahoo.com/config/login_verify2?&.src=ym

- SSL transactions are negotiated by means of a key based encryption algorithm between the client and the server, this key is usually either 40 or 128 bits in strength (the higher the number of bits the more secure the transaction).

## Need SSL if…

- you have an online store or accept online orders and credit cards
- you offer a login or sign in on your site
- you process sensitive data such as address, birth date, license, or ID numbers
- you need to comply with privacy and security requirements

- Certification Authority (CA) is an entity that issues digital certificates for use by other parties. It is an example of a trusted third party.
- e.g. VeriSign, Thwate, Geotrust etc

- Ability to connect to server via HTTP secure consists of:
  - Generating key
  - Generating certificate signing request
  - Generating self signed certificate
  - Certificate Authority signed certificate
  - Configuring web server.

# How SSL Overcomes HTTP Security Concerns

- Secure Sockets Layer technology protects your Web site and makes it easy for your Web site visitors to trust you in three essential ways:
  - Privacy
    - An SSL Certificate enables **encryption** of sensitive information during online transactions.
  - Integrity.
    - A Certificate Authority **verifies** the identity of the certificate owner when it is issued.
  - Authentication.
    - Each SSL Certificate contains unique, **authenticated** information about the certificate owner.

- **Following are the benefits of using HTTPS**
- Secures your data in-transit.
- Protects your website from Phishing, MITM and other data breaches.
- Builds trust on your website visitors.
- Removes “NOT Secure” warnings.
- Help you improve website ranking.

# Limitations of HTTPS

- An HTTPS server can only provide one "virtual host" behind a single socket, as opposed to multiple ones behind an http socket.
  - This is because all security negotiation takes place before the HTTP protocol starts & hence before the server knows which URL the client is asking for.
- HTTPS cannot prevent stealing confidential information from the pages cached on the browser.
  - Since in SSL data is encrypted only during transmission on the network, it is in clear text in the browser memory
- HTTPS is slightly slower than HTTP.
  - HTTPS adds computational overhead as well as network overhead.

# What is Https?

- HTTPS stands for Hypertext Transfer Protocol over Secure Socket Layer, Or HTTP over SSL is a web protocol developed by Netscape.
- HTTPS is a combination of HTTP and SSL/TLS protocols.
- HTTPS uses one-time encryption key to encrypt data send to and receive from the server.
- The 'S' at the end of HTTPS stands for 'Secure'. It means all communications between your browser and the website are encrypted.

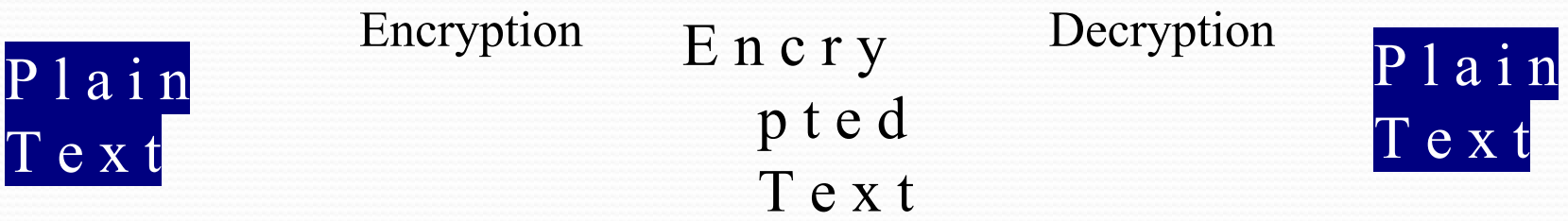
# How Does HTTPS Work?

- HTTPS pages typically use one of two secure protocols to encrypt communications.
  - SSL (Secure Sockets Layer)
  - TLS (Transport Layer Security)
- An asymmetric system uses two 'keys' to encrypt (communications, a 'public' key and a 'private' key.
- Anything encrypted with the public key can only be decrypted by the private key.

## Conti...

- The security of HTTPS is that of the underlying SSL,Which typically uses long-term public key and private key to generate a short term session key which is then used to encrypt data between client and server.
- X509 certificates are used to authenticate the server.

# How SSL Work-Encryption?



# What is a HTTPS certificate?

- When you request a HTTPS connection to a webpage , the website will initially send its SSL certificate to your browser.

- This certificate contains the public key needed to begin the secure session.

- Based on this initial exchange, your browser and the website then initiate the 'SSL handshake'.

- The SSL handshake involves the generation of shared secrets to establish a uniquely secure connection between yourself and the website.

# HTTPS Use

- HTTPS protocol used in the following scenarios

1) Banking Websites
2) Payment Gateway
3) Shopping Websites
4) Email Apps

- **Difference from HTTP:**
- HTTPS URLs begin with "https://" and use port 443 by default, whereas, HTTP URLs begin with "http://" and use port 80 by default.

- HTTP is not encrypted and thus is vulnerable to man-in-the-middle and eavesdropping attacks, which can let attackers gain access to website accounts and sensitive information, and modify webpages to inject malware or advertisements.
- HTTPS is designed to withstand such attacks and is considered secure against them (with the exception of HTTPS implementations that use deprecated versions of SSL).

# ❑ difference between HTTP and

| S.NO | HTTP | HTTPs |
|---|---|---|
| 1. | It is hypertext transfer protocol. | It is hypertext transfer protocol with secure. |
| 2. | It is not secure & unreliable. | It is secure & reliable. |
| 3. | HTTP URLs begin with http://. | HTTPs URLs begin with https://. |
| 4. | It uses port 80 by default . | It was use port 443 by default. |
| 5. | It is subject to man-in-the-middle & eavesdropping attacks. | It is designed to withstand such attacks & is considered secure against such attacks. |

# ❑ difference between HTTP and

| S.NO | HTTP | HTTPs |
|---|---|---|
| 6. | HTTP message is not encrypted & is not safe to send. | HTTPs message is encrypted, including the headers & the request/response load. |
| 7. | It is used for many purposes such as a website article that is open & available to everyone, this lack of security is of on important. | If a website is one that needs to collect private information such as credit card number then it is a more secure protocol. |
| 8. | Operates at application layer. | Operates at transport layer. |
| 9. | No certificates required. | Certificates required. |

# Difference Between HTTPS vs HTTP

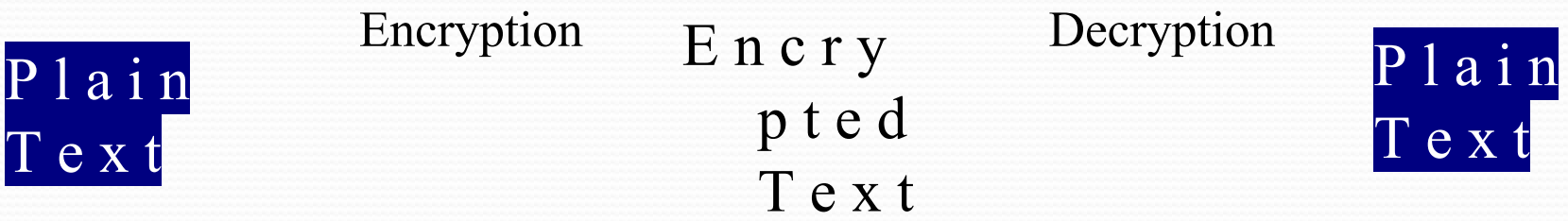
| HTTPS | HTTP |
| --- | --- |
| Encryption Layer enabled | No encryption layer |
| Data protection | No protection from attackers |
| Ranking Boost with Google | No ranking boost |
| Protection against Phishing | Easy to replicate and thus vulnerable to phishing |
| Leveraged to gain customer trust | Website safety cannot be leveraged |
| Payments Card Industry Compliance | Non-compliance with Payments Card Industry regulation |

| | |
|---|---|
| Needs Testing after conversion to HTTPS | No testing time and cost |
| Certification and Validation have costs associated with them | No certification / validation cost |
| Post validation redirection needed as a one-time activity | No redirections or relinking required |
| Chrome user friendly | Chrome users get a notification about "unsecured site" |

# SSL(Secure Sockets Layer)

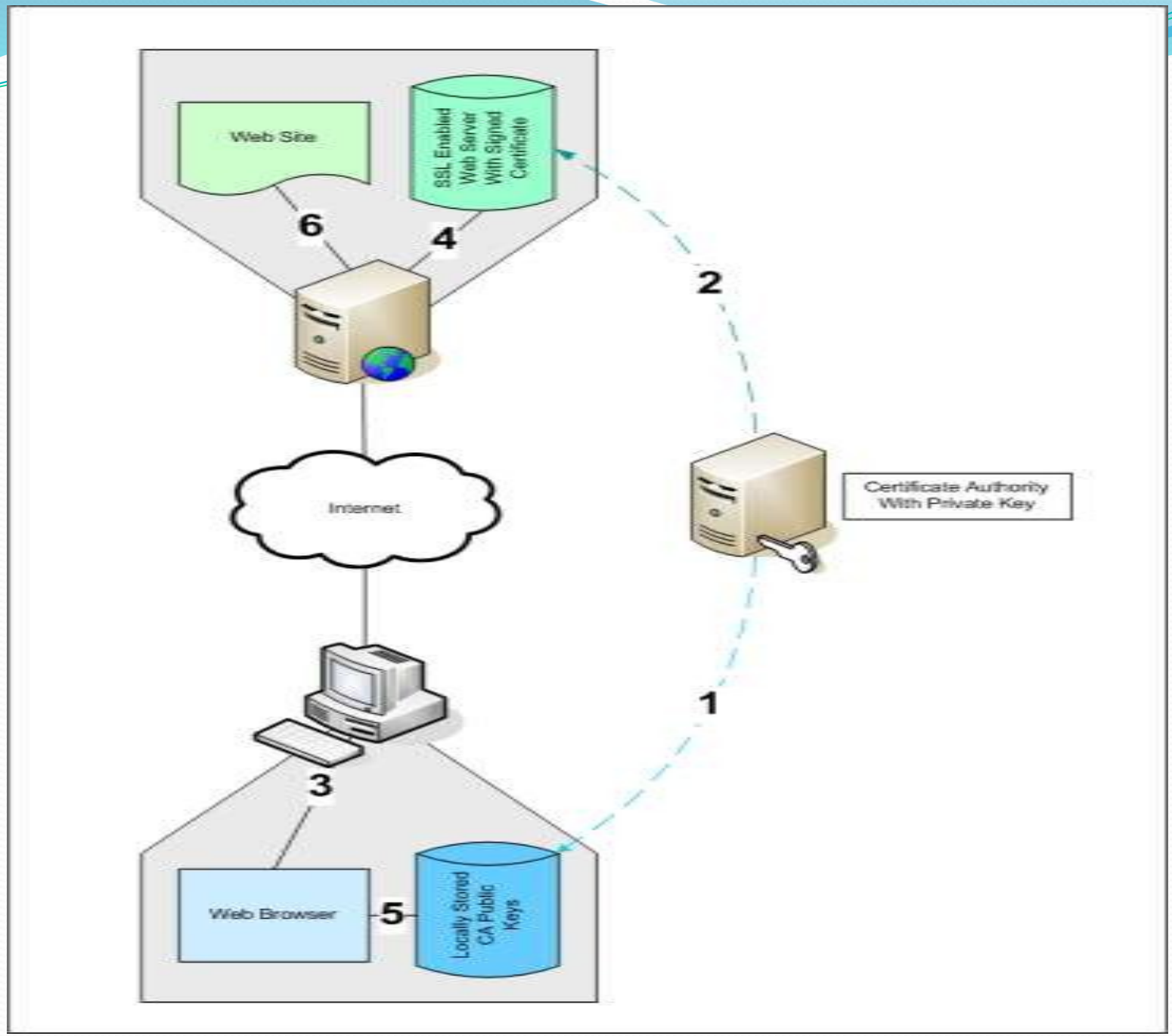
- SSL stands for Secure Sockets Layer, and was first developed by Netscape back in 1994.
- SSL encryption/decryption is a method by which internet connections are kept secure, whether they be client to client, server to server, or (much more regularly) client to server.
- This prevents unauthorized third parties from seeing or altering any data being exchanged across the internet.

# How SSL Work-Encryption?



# SSL Diagram

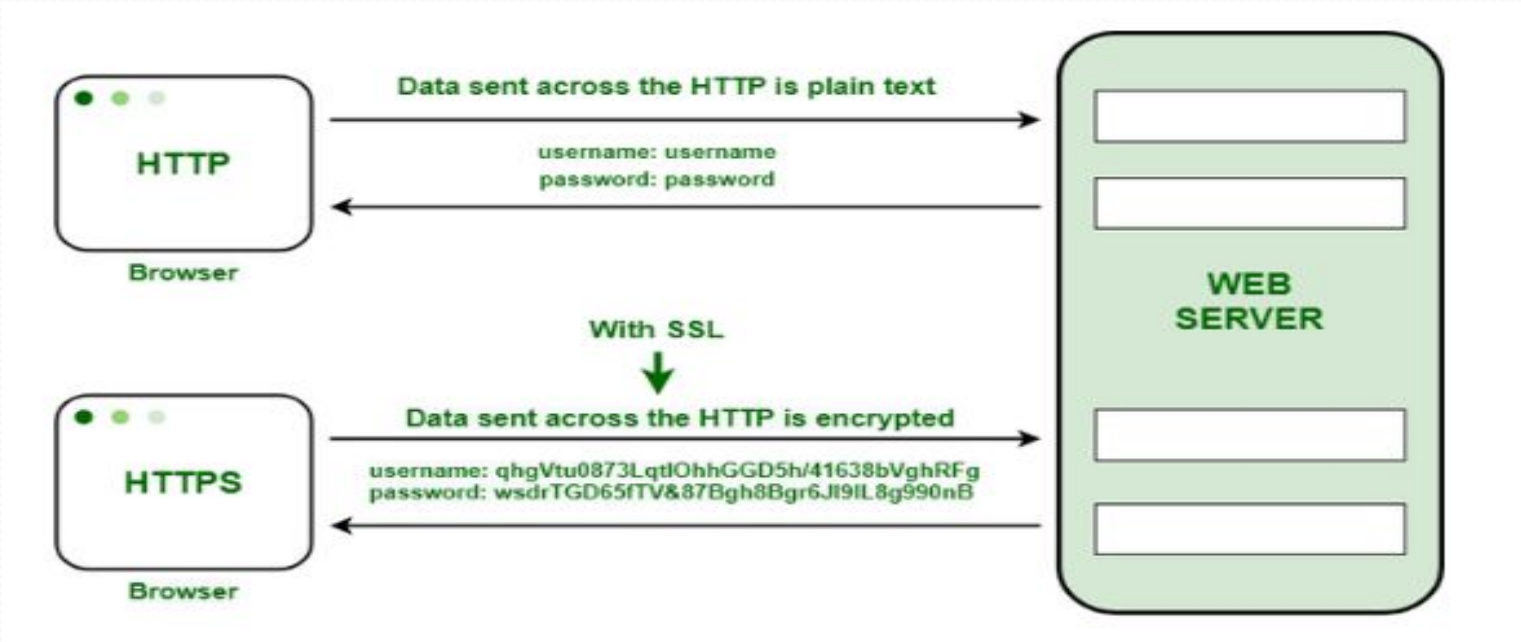
- When any modern browser is installed, it is sent with several *CA issuer certificates*.
- These issuer certificates contain a public key for the issuer, among other information.
- When a web designer decides to use SSL he needs to purchase a *certificate that is signed* using the CA's private key.
- The web browser starts a connection to an HTTPS site. Along with this request the client sends all supported encryption schemes.
- As a response to the browser's connection request, the Server sends a copy of the certificate from *step 2*. Along with this transmission is the server's answer to the encryption negotiation.


Web Site
SSL Enabled Web Server With Signed Certificate
6
4
2
Internet
Certificate Authority With Private Key
1
3
Web Browser
5
Locally Stored CA Public Keys

- Once a certificate is downloaded, the signature of the certificate (*that was signed using the CA's private key*) is checked using the CA's public key (installed in the browser in *step 1.*
- The connection succeeds, the client can now download and upload to the web site with the security of encryption.

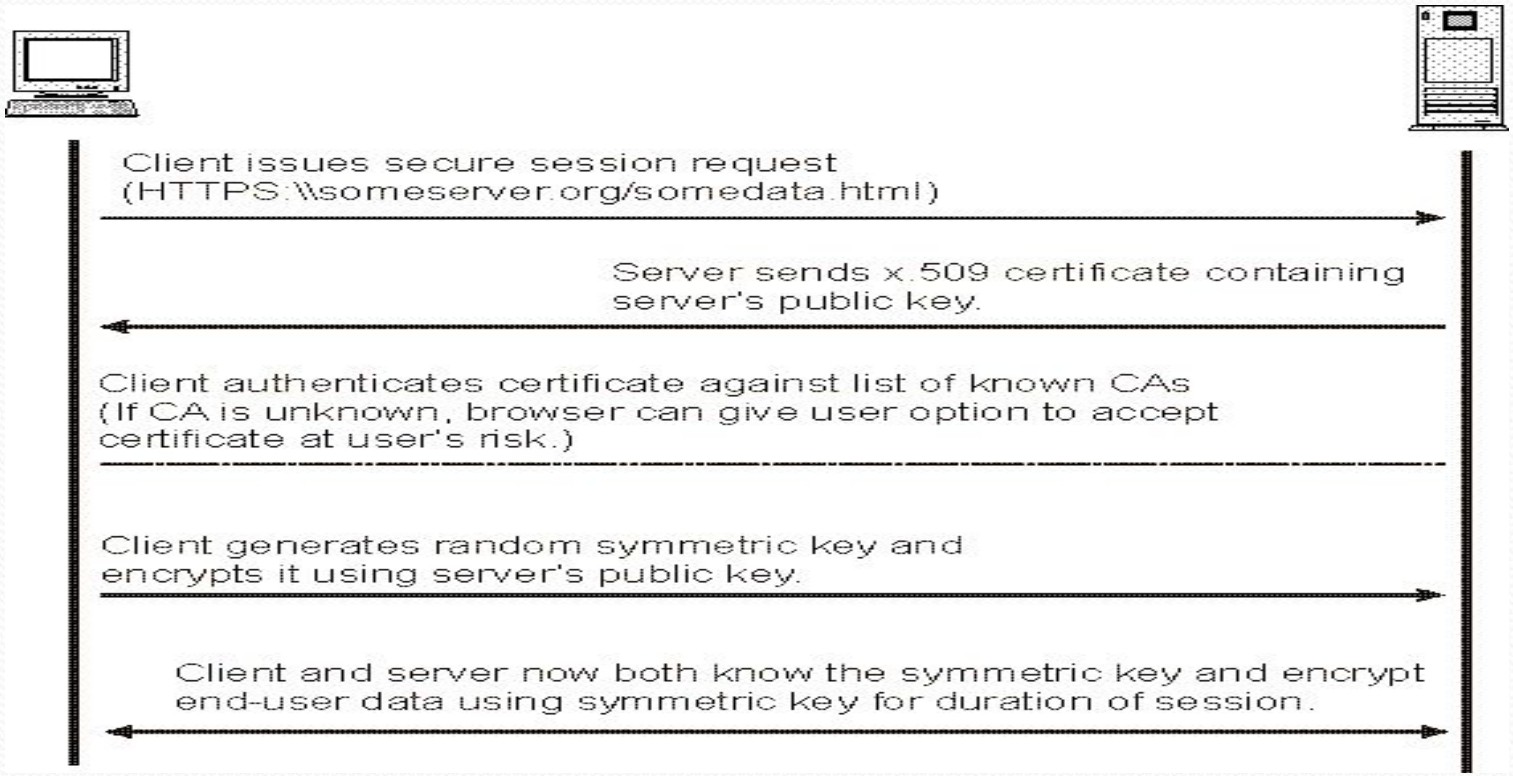
# SSL

- SSL Secure Socket Layer.
- Protocol for establishing Secure link between client and Server.
- SSL Provides confidentiality, Authentication, and Data integrity in internet communication.



# SSL Handshake

- A HTTP-based SSL connection is always initiated by the client using a URL starting with https:// instead of with http://.
- At the beginning of an SSL session, an SSL handshake is performed
- This handshake produces the cryptographic parameters of the session.
- Simplified Overview:



1. **Browser** connects to a web server (website) secured with SSL (https). Browser requests that the server identify itself.

2. **Server** sends a copy of its SSL Certificate, including the server's public key.

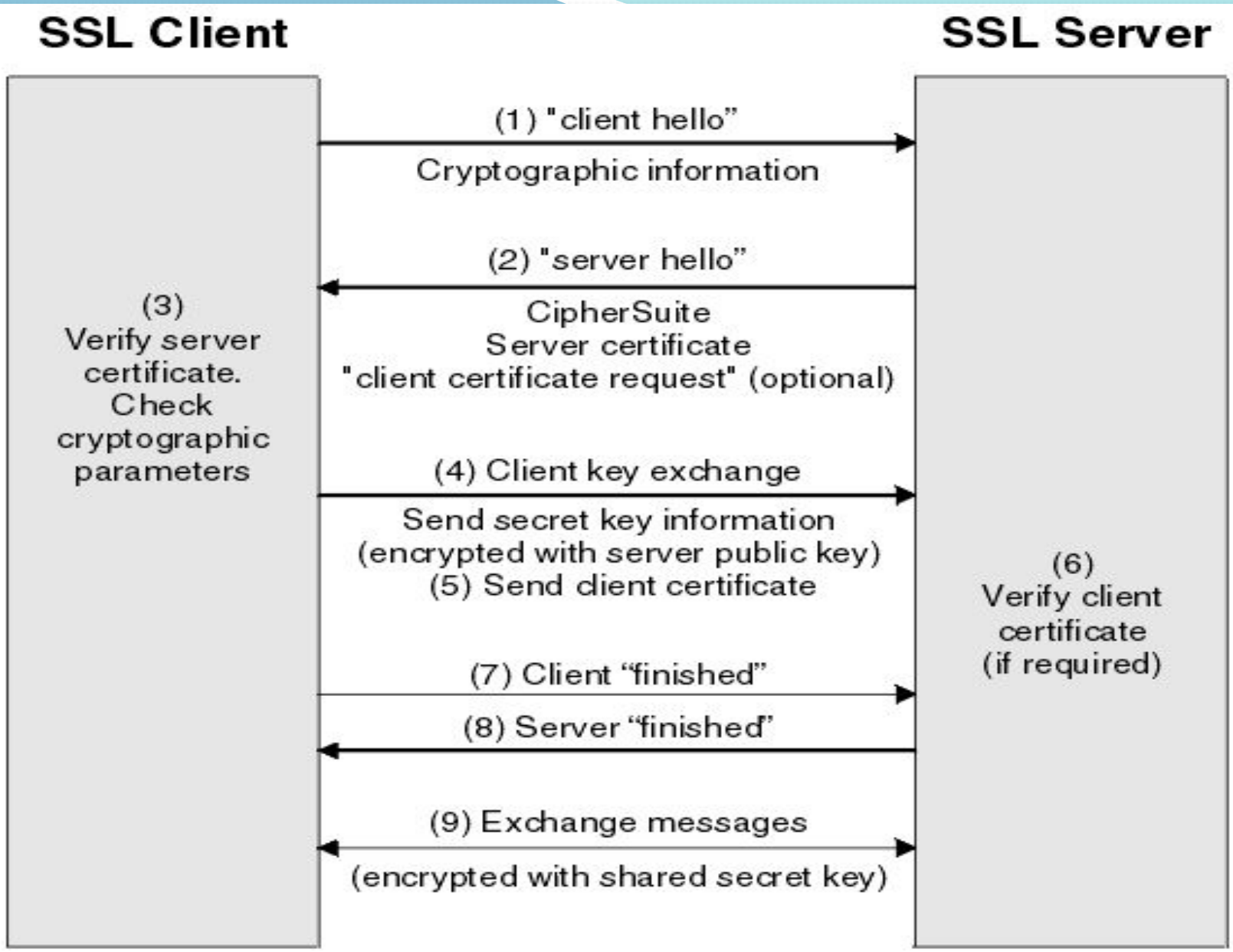
3. **Browser** checks the certificate root against a list of trusted CAs and that the certificate is unexpired, unrevoked, and that its common name is valid for the website that it is connecting to. If the browser trusts the certificate, it creates, encrypts, and sends back a symmetric session key using the server's public key.

4. **Server** decrypts the symmetric session key using its private key and sends back an acknowledgement encrypted with the session key to start the encrypted session.

5. **Server** and Browser now encrypt all transmitted data with the session key.

# SSL Handshake

- The SSL or TLS handshake enables the SSL or TLS client and server to establish the secret keys with which they communicate.
- **Steps that enable the SSL or TLS client and server to communicate with each other:**
  - Agree on the version of the protocol to use.
  - Select cryptographic algorithms.
  - Authenticate each other by exchanging and validating digital certificates.
  - Generates a shared secret key.
  - SSL or TLS then uses the shared key for the symmetric encryption of messages, which is faster than asymmetric encryption.




***SSL or TLS handshake***

- The SSL or TLS client sends a client hello message that lists cryptographic information such as the SSL or TLS version and, in the client's order of preference, the CipherSuites supported by the client.
- The message also contains a random byte string that is used in subsequent computations.
- The protocol allows for the client hello to include the data compression methods supported by the client.
- The SSL or TLS server responds with a server hello message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string.
- The server also sends its digital certificate.
- If the server requires a digital certificate for client authentication, the server sends a client certificate request that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CAs).

- The SSL or TLS client verifies the server's digital certificate.
- The SSL or TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data.
- The random byte string itself is encrypted with the server's public key.
- If the SSL or TLS server sent a client certificate request, the client sends a random byte string encrypted with the client's private key, together with the client's digital certificate, or a no digital certificate alert.
- This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.
- The SSL or TLS server verifies the client's certificate.

- The SSL or TLS client sends the server a finished message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
- The SSL or TLS server sends the client a finished message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
- For the duration of the SSL or TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

# Transport Layer Security

- TLS stands for Transport Layer Security, and is essentially SSL, but more secure.
- More accurately, **the Internet Engineering Task Force** has deprecated SSL in favor of TLS.
- Much like SSL, TLS is a cryptographic protocol which provides privacy, authentication and data integrity over computer networks, and is used in web browsing, instant messaging, email and more.

- 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).
- TLS was proposed by the Internet Engineering Task Force (IETF), an international standards organization, and the first version of the protocol was published in 1999. The most recent version is TLS 1.3, which was published in 2018.

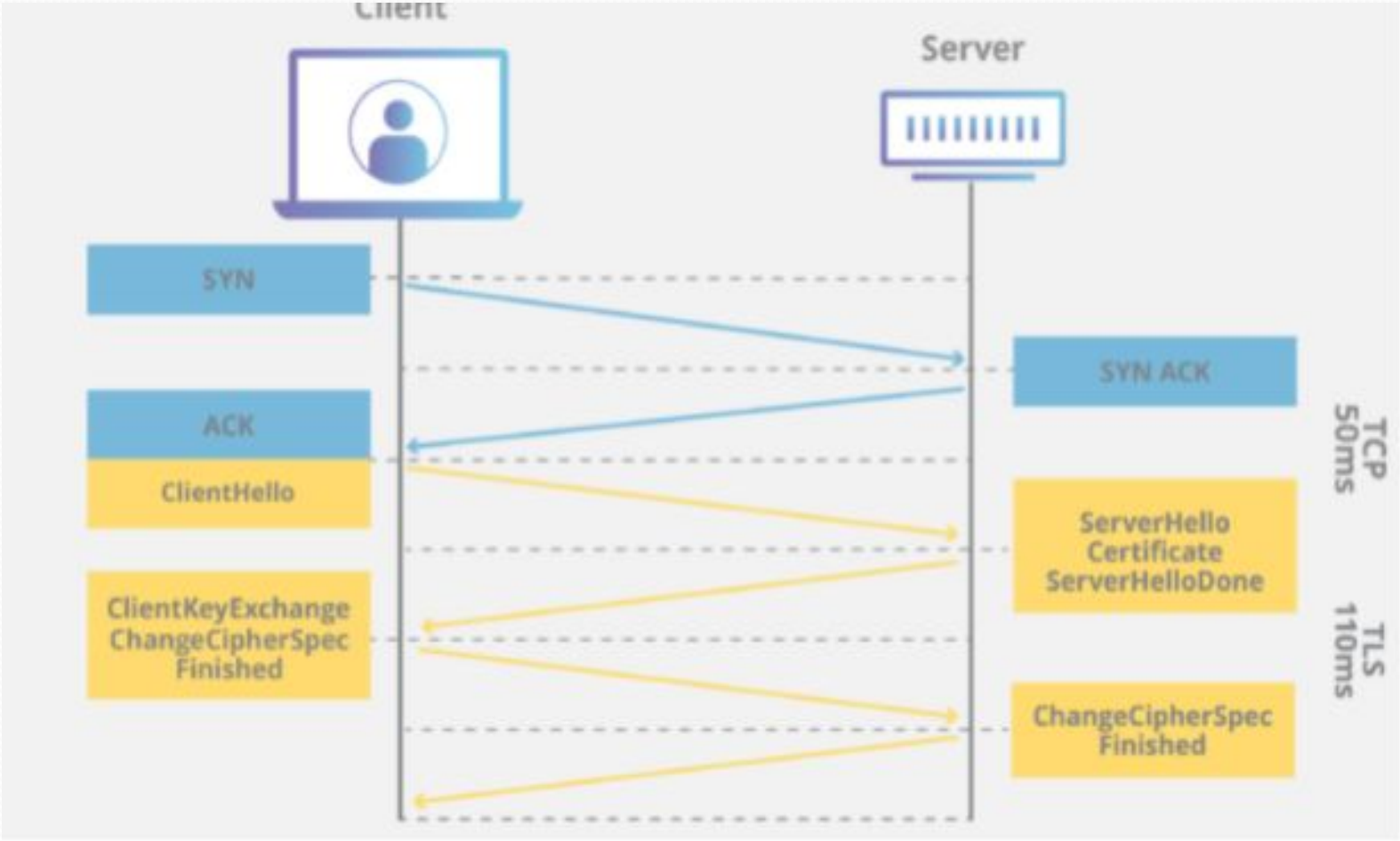
- **What does TLS do?**
- There are three main components to what the TLS protocol accomplishes: **Encryption, Authentication, and Integrity.**
- Encryption: hides the data being transferred from third parties.
- Authentication: ensures that the parties exchanging information are who they claim to be.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

- **How does TLS work?**
- For a website or application to use TLS, it must have a TLS certificate installed on its origin server (the certificate is also known as an "SSL certificate" because of the naming confusion described above).
- A TLS certificate is issued by a certificate authority to the person or business that owns a domain.
- The certificate contains important information about who owns the domain, along with the server's public key, both of which are important for validating the server's identity.
- 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 (see below) they will use
- Authenticate the identity of the server using the server's TLS certificate
- Generate session keys for encrypting messages between them after the handshake is complete.

- 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.
- 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

- SSL refers to Secure Sockets Layer whereas TLS refers to Transport Layer Security.
- How similar both are? SSL and TLS are cryptographic protocols that authenticate data transfer between servers, systems, applications and users.
- For example, a cryptographic protocol encrypts the data that is exchanged between a web server and a user.
- SSL was a first of its kind of cryptographic protocol. TLS on the other hand, was a recent upgraded version of SSL.

- **Why do you need an SSL/TLS certificate?**
- Cyber security has become a serious threat that is spreading across all sections of the internet.
- From schools to enterprises and individuals, it puts user data of all types and sizes at risk.
- The risk is especially higher when there is exchange of information through client and server systems.

- There is a need for secure system that encrypt data flow from either side.
- An SSL/TLS certificate helps with that.
- It acts as an endpoint encryption system that encrypt data preventing unauthorized access by hackers.

- In the present day, SSL has also gained importance as a serious ranking signal due to Google's announcement.
- Websites with SSL certificates gain better search ranking traction, have better user experience and do not pose any security concerns — even during eCommerce transactions.

- **Differences between SSL and TLS**
- **Cipher suites**
- SSL protocol offers support for Fortezza cipher suite. TLS does not offer support. TLS follows a better standardization process that makes defining of new cipher suites easier like RC4, Triple DES, AES, IDEA, etc.

- **Alert messages**
- SSL has the “No certificate” alert message. TLS protocol removes the alert message and replaces it with several other alert messages.

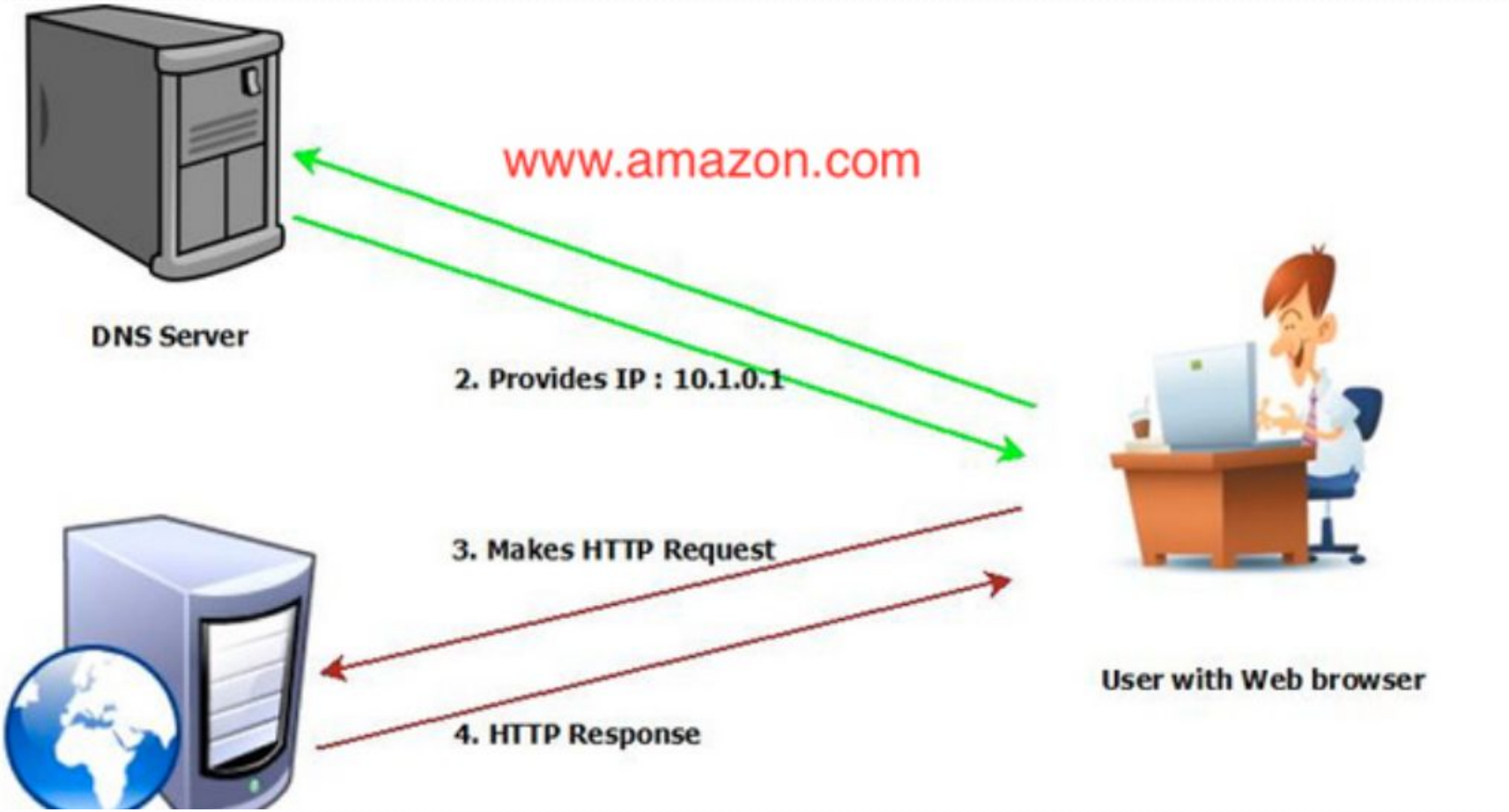
- **Record Protocol**
- SSL uses Message Authentication Code (MAC) after encrypting each message while TLS on the other hand uses HMAC — a hash-based message authentication code after each message encryption.

# Domain Name System

- **The Domain Name System (DNS)** is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com.
- 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).

- **The Domain Name System (DNS)** is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network.
- **The Domain Name System delegates the responsibility of assigning domain names and mapping those names to Internet resources by designating authoritative name servers for each domain.**
- **Network administrators may delegate authority over sub-domains of their allocated name space to other name servers.**

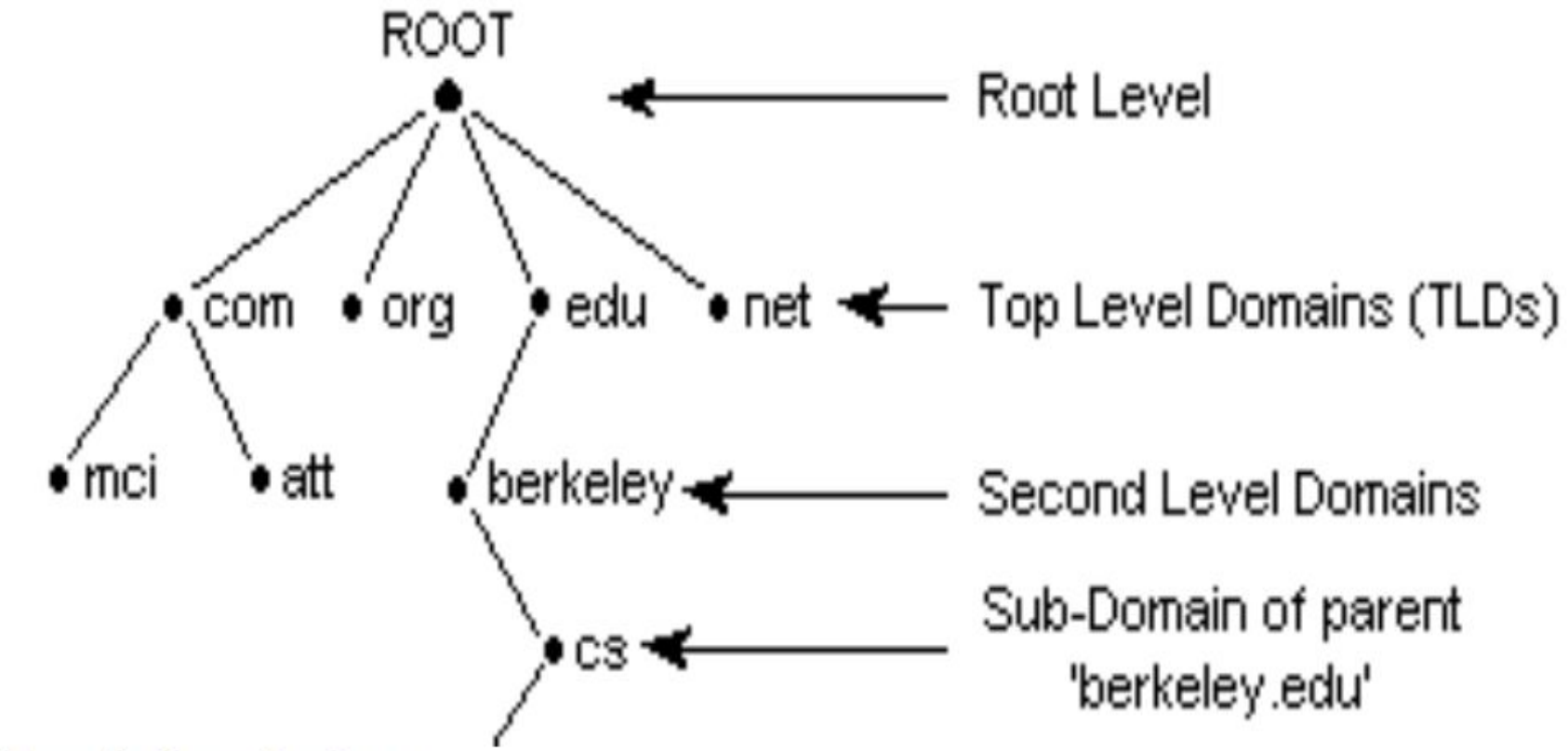
- The DNS provides mapping between human-readable names (like www.amazon.com) and their associated IP addresses (like 205.251.242.103).
- DNS can be best compared to a phone book where you look up the phone numbers listed by easier-to-remember names.
- DNS comes under the application layer protocol.
- A user types www.amazon.com in his browser, which then queries the DNS server for amazon.com's IP addresses.
- The servers return Amazon's address so the browser can request data from Amazon's web host, which returns the elements necessary to build their home page in the local browser.


www.amazon.com
DNS Server
2. Provides IP : 10.1.0.1
3. Makes HTTP Request
4. HTTP Response
User with Web browser

- **How DNS Works:**
- Domain Name System
- **Domain Names** A domain name is a human-readable name—like amazon.com—that we type in a web browser URL field.
- The Internet Corporation for Assigned Names and Numbers (ICANN) manages these domain names.
- Top Level Domain (TLD)TLD refers to the last part of a domain name.
- For example, the .com in amazon.com is the Top Level Domain. The most common TLDs include .com, .net, org, and .info. Country code TLDs represent specific geographic locations. For example: .in represents India. Here are some more examples:
- com – Commercial businesses.
- gov – U.S. government agencies.
- edu – Educational institutions such as universities.
- org – Organizations (mostly non-profit).
- mil – Military.net – Network organizations.
- eu – European Union.

- **Second Level Domain:**
- This is the part of a domain name which comes right before the TLD—amazon.com—for example.
- **Sub Domain:**
- A subdomain can be created to identify unique content areas of a web site.
- For example, the aws of aws.amazon.com.

## DNS Hierarchy




Domain Name Registrar

# Structure of domain Name

- Last name. subdomain. second-level domain. top-level domain

  EXAMPLE: vijay.Bombay.vni.com

# Top Level Domains

Top level domains are classified into 3 categories:

- Organizational or generic domains
- Geographical or country domains
- Reverse domains

# Organizational/Generic domains

- It consists of three character code which indicates the primary function of the organization or their generic behavior
- Most commonly used top level domains are:
- **.com** for commercial organization for → eg
- **.net** networking organizations for → www.yahoo.com eg
- **.gov** government organizations for → eg www.state.nj.gov eg
- **.edu** educational organizations → www.uducause.edu
- **.org** for non-commercial organizations for → eg www.eklavya.org eg
- **.mil** military organizations → www.dod.mil
- **.int** for international organizations → eg www.itu.int

# Geographical/Country Domains

- It consists of two characters which represents different countries/regions all around the world
- These codes have been standardized by International Standard Organizational (ISO)

EXAMPLE:

- .in → India
- .jp → Japan
- .us → United States
- .fr → france
- .it → Italy
- .cn → China
- .au → Australia

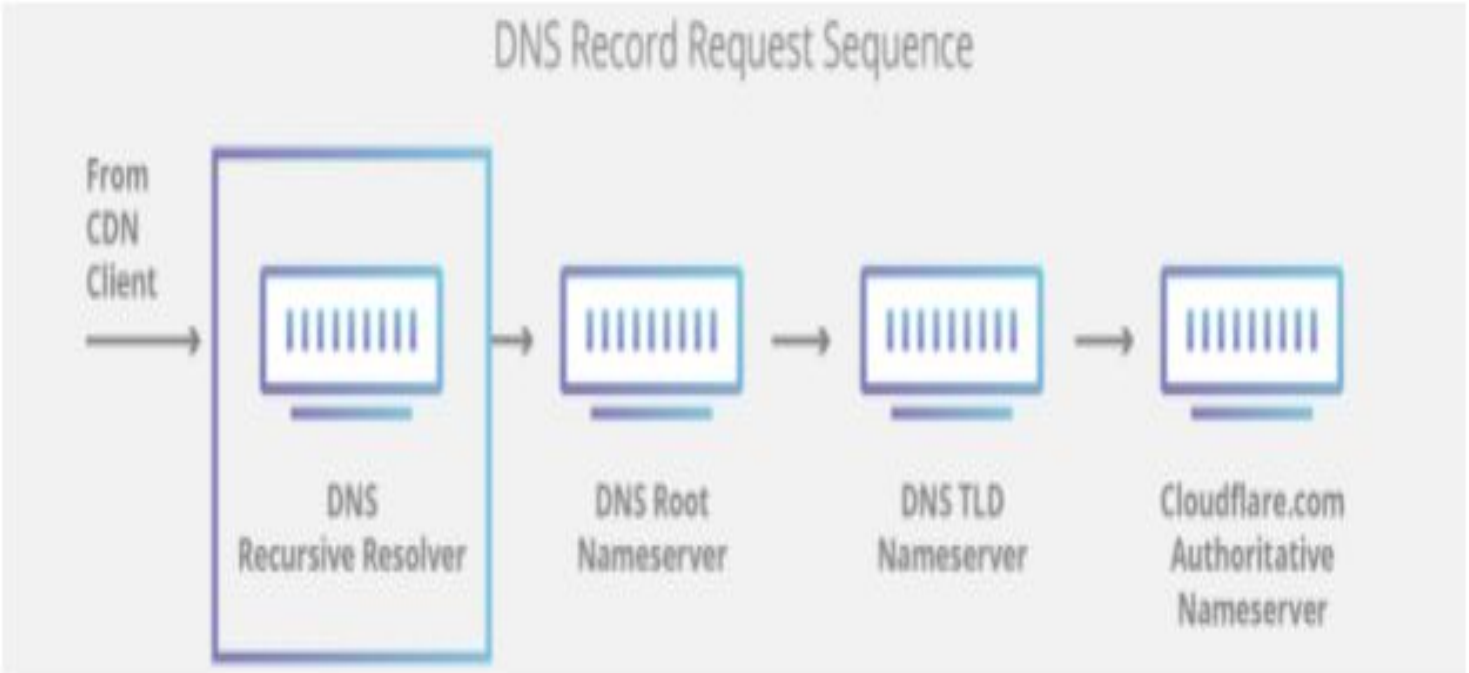
- **How does DNS work?**
- The process of DNS resolution involves converting a hostname (such as www.example.com) into a computer-friendly IP address (such as 192.168.1.1).
- An IP address is given to each device on the Internet, and that address is necessary to find the appropriate Internet device - 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.
- In order to understand the process behind the DNS resolution, it's important to learn about the different hardware components a DNS query must pass between.
- For the web browser, the DNS lookup occurs " behind the scenes" and requires no interaction from the user's computer apart from the initial request.

- **There are 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.

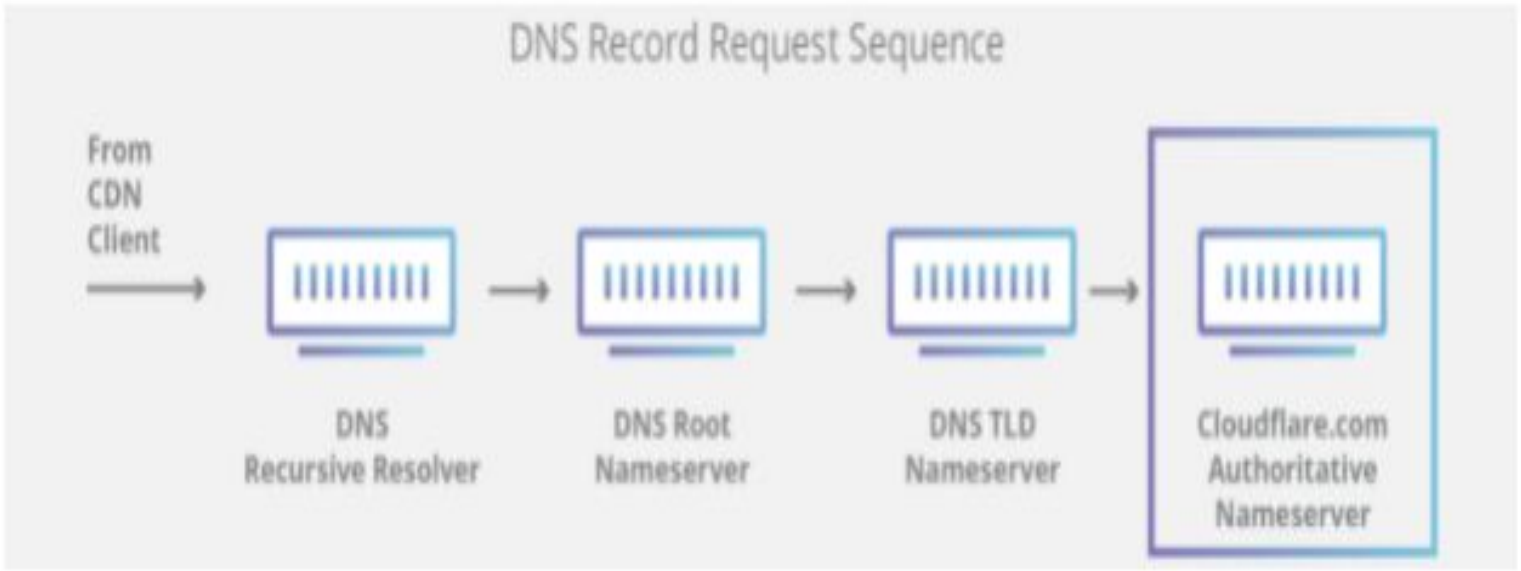
- **Root nameserver** - The root server is the first step in translating (resolving) human readable host names into IP addresses.
- It can be thought of like an index in a library that points to different racks of books - typically it serves as a reference to other more specific locations.
- **TLD nameserver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

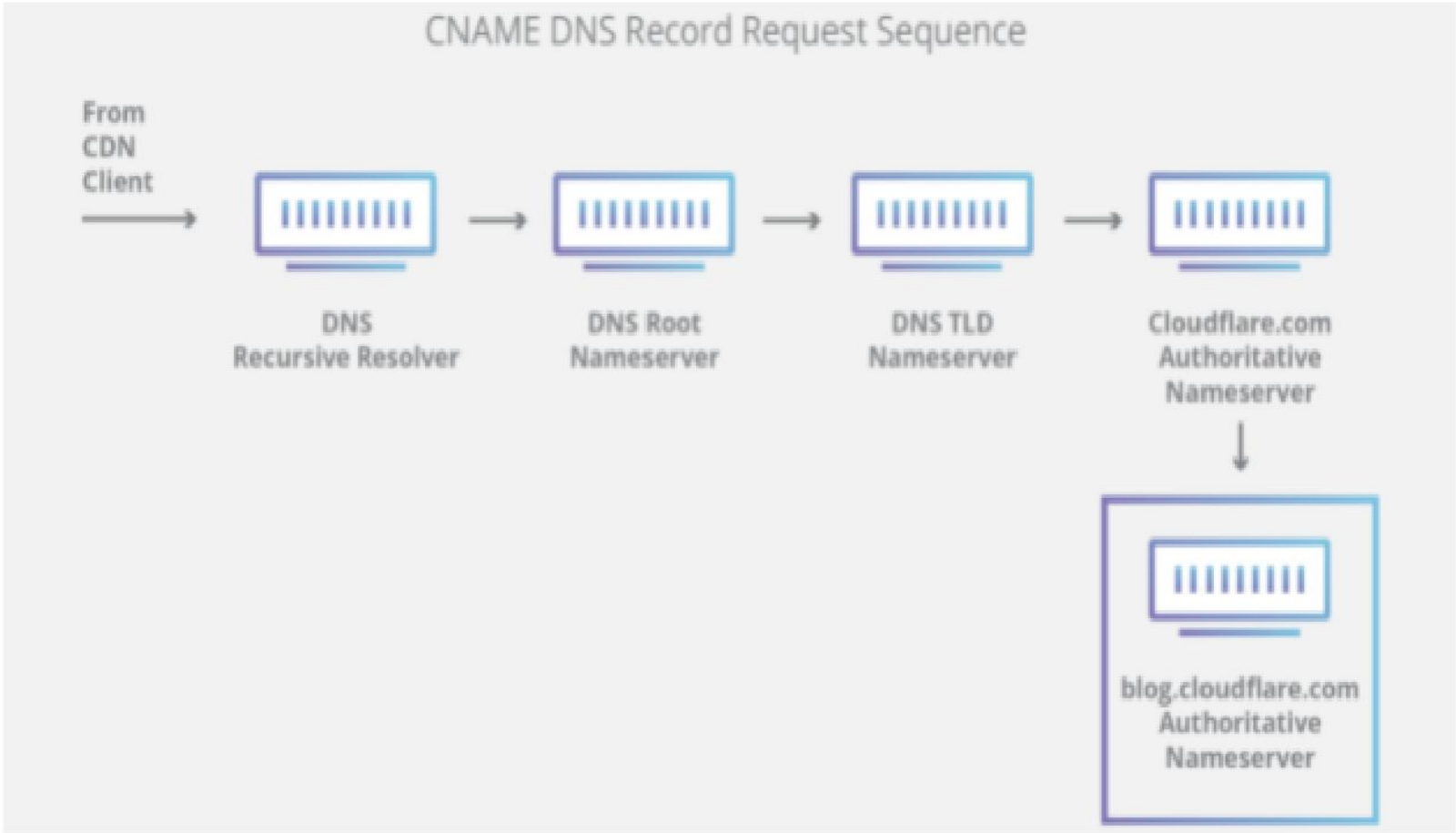
## Recursive DNS resolver

- The recursive resolver is the computer that responds to a recursive request from a client and takes the time to track down the DNS record.
- It does this by making a series of requests until it reaches the authoritative DNS nameserver for the requested record (or times out or returns an error if no record is found).
- Recursive DNS resolvers do not always need to make multiple requests in order to track down the records needed to respond to a client; caching is a data persistence process that helps short-circuit the necessary requests by serving the requested resource record earlier in the DNS lookup.


DNS Record Request Sequence
From
CDN
Client
DNS
Recursive Resolver
DNS Root
Nameserver
DNS TLD
Nameserver
Cloudflare.com
Authoritative
Nameserver

- Authoritative DNS server
- An authoritative DNS server is a server that actually holds, and is responsible for, DNS resource records.
- This is the server at the bottom of the DNS lookup chain that will respond with the queried resource record, ultimately allowing the web browser making the request to reach the IP address needed to access a website or other web resources.
- An authoritative nameserver can satisfy queries from its own data without needing to query another source, as it is the final source of truth for certain DNS records.


DNS Record Request Sequence
From
CDN
Client
DNS
Recursive Resolver
DNS Root
Nameserver
DNS TLD
Nameserver
Cloudflare.com
Authoritative
Nameserver


CNAME DNS Record Request Sequence
From
CDN
Client
DNS
Recursive Resolver
DNS Root
Nameserver
DNS TLD
Nameserver
Cloudflare.com
Authoritative
Nameserver
blog.cloudflare.com
Authoritative
Nameserver

- **What are the steps in a DNS lookup?**
- DNS is concerned with a domain name being translated into the appropriate IP address.
- It follow the path of a DNS lookup as it travels from a web browser, through the DNS lookup process, and back again.

- **Note:**
- Often DNS lookup information will be cached either locally inside the querying computer or remotely in the DNS infrastructure.
- There are typically 8 steps in a DNS lookup.
- When DNS information is cached, steps are skipped from the DNS lookup process which makes it quicker.
- The example below outlines all 8 steps when nothing is cached.

## The 8 steps in a DNS lookup:

1. A user types ‘example.com’ into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. 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.
4. When searching for example.com, our request is pointed toward the .com TLD.
5. The resolver then makes a request to the .com TLD.

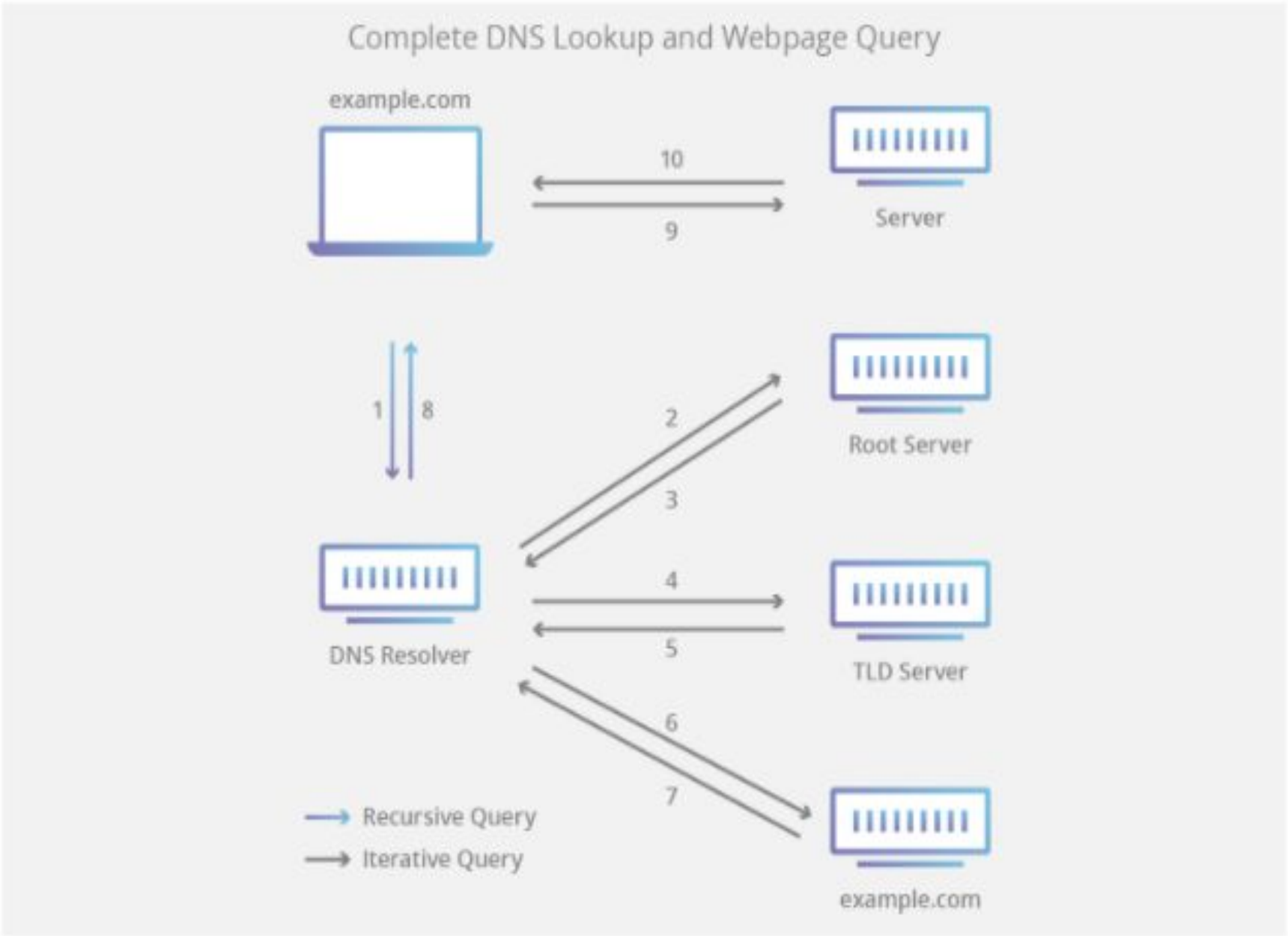
6. The TLD server then responds with the IP address of the domain's nameserver, example.com.

7. 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.

8.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

# Domain Name

- Domain name is a way to identify and locate computers connected to internet
- No two organizations can have same domain name
- A domain name always consists of two or more components separated by periods called dots (.)
  EXAMPLE: www.yahoo.co.in, www.facebook.com etc.
- Once a domain has been established subdomains can be created within the domain
  EXAMPLE: The domain for the large company could be “Vni.com” and within this domain subdomains can be created for each of the company’s regional office.
  Eg: Bombay.vni.com

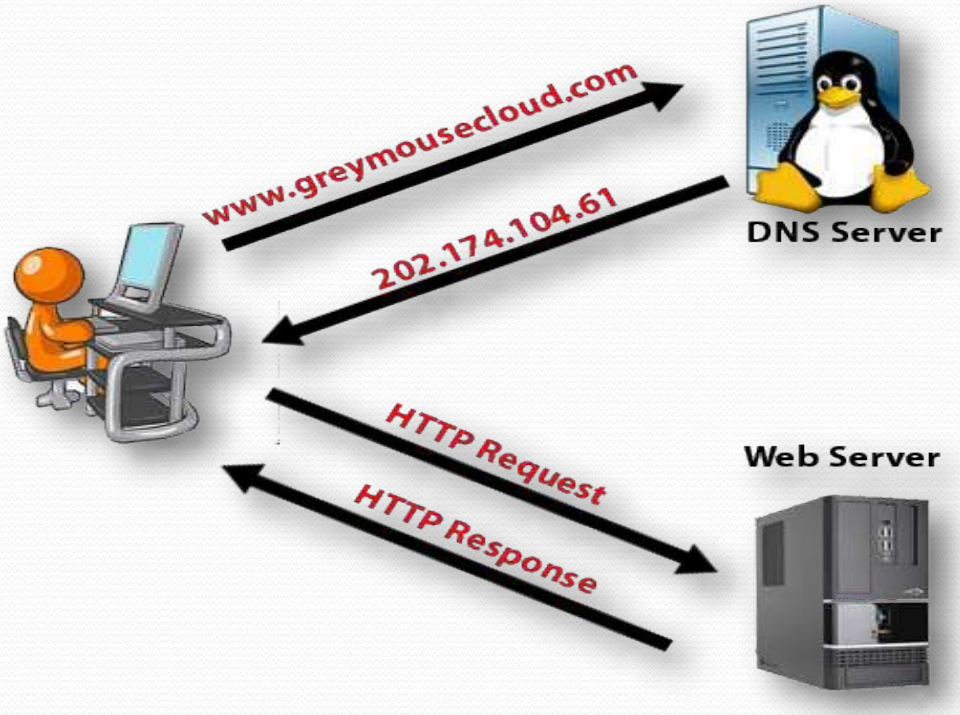
- Each domain name has a corresponding IP address

- When the user types the domain name in the address bar, the corresponding IP address is supplied.
- Such a translation is possible with the help of system called DNS (DOMAIN NAME SYSTEM)

- DEFINITION:

"DOMAIN NAME SYSTEM is a collection of the databases that contain information about domain names and their corresponding IP address."

# Working Of DNS



- When an application program needs to communicate with other computer, it needs to translate the name and the other computer into its IP address. The applications program that requests the service then becomes the client of DNS.

- It then sends the request to DNS server. The server looks up the name and then returns correct IP address.

- A large number of DNS servers may be involved to get the right IP address. After receiving the correct IP address, the communication between two computers starts.

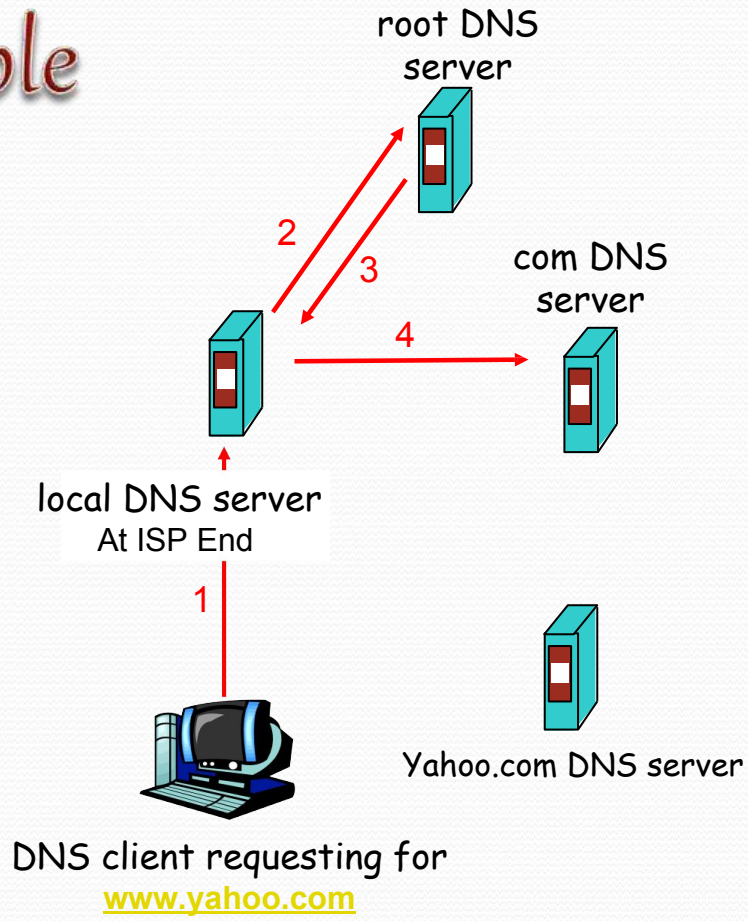
# DNS example

1. When you type name www.yahoo.com into your browser it asks local DNS server (at ISP's end) for its IP address.

2. When local DNS server does not find the IP address of given name, it forwards request to root DNS server and again enquire about IP address of it.

3. The root DNS server replies " I do not know the IP address of www.yahoo.com but know the IP address of the com DNS server".

4. The local DNS then asks the com DNS server for IP address



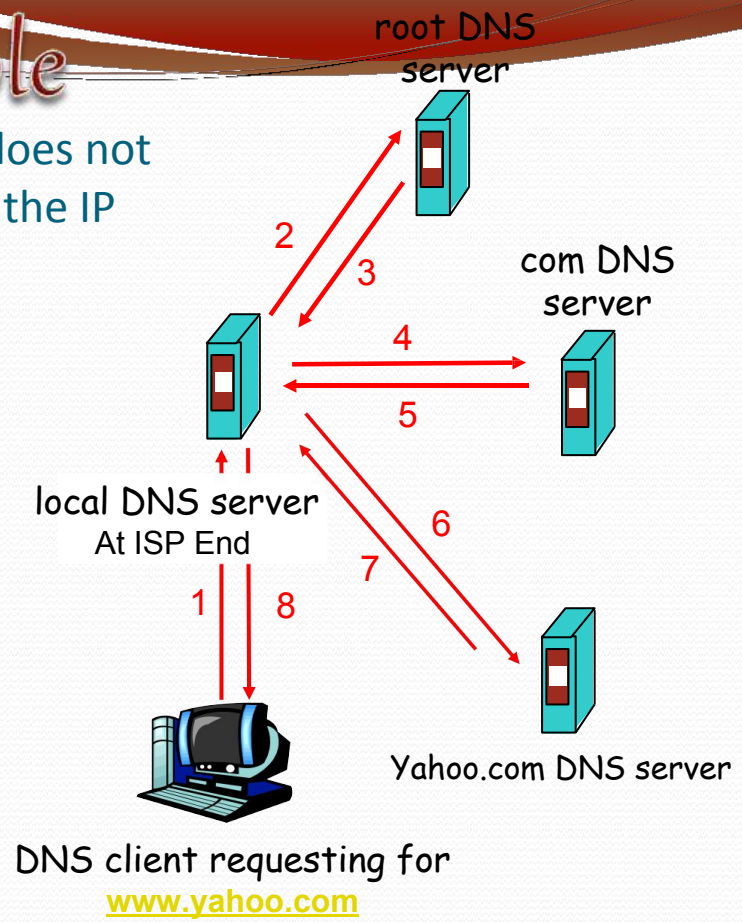
# DNS example

5. The com DNS server replies with same answer it does not know the IP address of www.yahoo.com but know the IP address of yahoo.com DNS server which is then return to local DNS server.

6. **The local DNS server then ask the yahoo.com DNS server for IP address**

7. **It then replies with IP address corresponding to www.yahoo.com which it has**

8. **The local DNS server then sends this IP address back to the client computer that send the request**



# Features Of dns

# Global Distribution

- Data is maintained locally, but retrievable globally
- No single computer has all DNS data
- DNS lookups can be performed by any device
- Remote DNS data is locally catchable to improve performance

# Scalability

- No limit to the size of the database
- One server has over 20,000,000 names
- No limit to the number of queries
- 24,000 queries per second handled easily
- Queries distributed among masters, slaves, and caches

# Dynamicity

- Database can be updated dynamically
  - Add/delete/modify of any record
- Modification of the master database triggers replication
  - Only master can be dynamically updated
    - ☞Creates a single point of failure

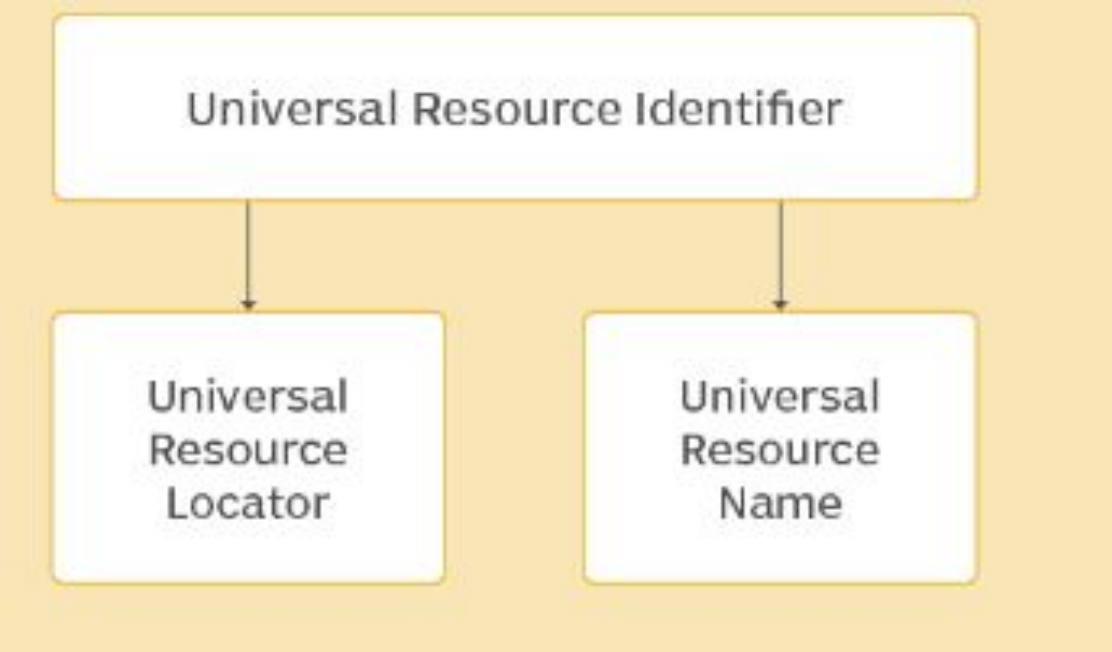
# Reliability

- Data is replicated
  - Data from master is copied to multiple slaves
- Clients can query
  - Master server
  - Any of the copies at slave servers
- Clients will typically query local caches
- DNS protocols can use either UDP or TCP
  - If UDP, DNS protocol handles retransmission, sequencing, etc.

## What is URI?

- A URI or Uniform Resource Identifier is a string identifier that refers to a resource on the internet.
- It is a string of characters that is used to identify any resource on the internet using location, name, or both.
- **A URI contains scheme, authority, path, query, and a fragment.**
- **Some most common URI schemes are HTTP, HTTPs, ftp, telnet, etc.**

- A URI has two subsets: URL (Uniform Resource Locator) and URN (Uniform Resource Number). If it contains only a name, it means it is not a URL. Instead of directly URI, we mostly see the URL and URN in the real world.


Universal Resource Identifier
Universal Resource Locator
Universal Resource Name

- **Syntax of URI**
- **The Syntax of URI is given below:**
- scheme:[//authority]path[?query][#fragment]
- **Scheme:**
- The first component of URI is scheme that contain a sequence of characters that can be any combination of letter, digit, plus sign, or hyphen (_), which is followed by a colon (:).
- The popular schemes are http, file, ftp etc.
- **Authority:** The authority component is optional and preceded by two slashes (//). It contains three sub-components:
  - **userinfo:** It may contain a username and an optional password separated by a colon. The sub-component is followed by the @ symbol.
  - **host:** It contains either a registered name or an IP address. The IP address must be enclosed within [] brackets.

- **Port: Optional**
- **Path:**
- It consists of a sequence of path segments separated by a slash(/).
- These slashes imply a hierarchical structure.
- The path begins with a single slash, whether or not an authority is present.
- However, the path cannot start with a double slash.
- This part of the syntax may closely resemble a particular file path but does not always
- **Query:optional**
- It is an optional component, which is preceded by a question mark(?).
- It contains a query string of non-hierarchical data.
- It is often a sequence of attribute-value pairs separated by a delimiter, such as an ampersand (&) or semicolon.
- A question mark separates the query from the part that comes before it.
- foo://techtarget.com:8042/over/there?name=parrot#beak

- **Fragment:**
- It is also an optional component, preceded by a hash(#) symbol.
- It consists of a fragment identifier that provides direction to a secondary resource.
- If the primary resource is an HTML document or article, the fragment may be an ID attribute of a specific element of that resource.
- In this case, a web browser will scroll this particular element into view.

- Eg
- telnet://192.0.2.16:80/
- ftp://ds.internic.net/internet-drafts/draft-ietf-uri-irl-fun-req-02.txt
- https://blog.hubspot.com/website/application-programming-interface-api
- urn:oasis:names:specification:docbook:dtd:xml:4.1.2

- **Uniform Resource Locator (URL)**
- A URL is used to identify and locate webpages.
- A URI identifies a resource but does not imply or guarantee access to it. A URL, however, not only identifies the resource, but also specifies how it can be accessed or where it is located.
- This is why a URL contains unique components, such as the protocol, domain and/or subdomain, in addition to other URI components.
- **A URL is a subset of URIs.**
- This means all URLs are URIs.
- However, not all URIs are URLs.
- A URL is a location-dependent URI that may or may not be persistent

- A URL begins by stating the protocol that should be used to access and locate the logical or physical resource on a particular network.
- Therefore:
- If the resource is a webpage, the URL starts with the protocol HTTP or HTTPS.
- If the resource is a file, the URL begins with the protocol FTP.
- For an email address, the URL starts with the protocol "mailto.“
- https://www.techtarget.com/whatis/definition/URI-Uniform-Resource-Identifier

- **Uniform Resource Name (URN)**
- Like a URL, a URN identifies a resource.
- However, unlike a URL, a URN is location-independent and persistent, meaning that it always identifies the same resource over time.
- A URN continues to persist even when the resource no longer exists or becomes unavailable.
- A URN does not state which protocol should be used to locate and access the resource.
- Instead, it labels the resource with a persistent, location-independent and unique identifier.

- Some examples of URI
- mailto:hey.Doe@example.com
- news:comp.infosystems.www.servers.unix
- urn:oasis:names:specification:docbook:dtd:xml:4.1.2

- **What is the URL?**
- A URL or Uniform Resource Locator is used to find the location of the resource on the web.
- It is a reference for a resource and a way to access that resource.
- A URL always shows a unique resource, and it can be an HTML page, a CSS document, an image, etc.
- A URL uses a protocol for accessing the resource, which can be HTTP, HTTPS, FTP, etc.
- It is mainly referred to as the address of the website, which a user can find in their address bars.
- An example of an URL is given below:

- A URN has three components:
- The label "urn"
- A colon
- A character string as the unique identifier
- urn:oasis:names:specification:docbook:dtd:xml:4.1.2
- urn:example:animal:ferret:nose

- **Syntax of URL**
- Each HTTP URL follow the syntax of its generic URI.
- Hence the syntax of the URL is also similar to the syntax of URI.
- It is given below:
- **scheme:[//authority]path[?query][#fragment]**

- The above URL is made up of the following components:

- **Scheme:**
- The URL's first component is a scheme, which represents a protocol that a browser must need to use to request the resource.
- The commonly used protocols for websites are HTTP or HTTPS.
- **Authority:**
- The authority includes two sub-components, domain name and Port, separated by a colon.
- The domain name can be anything, the registered name of the resource like javatpoint.com, and port is the technical gate to access the resource on a webserver.
- The port number 80 is used for HTTP and 443 is used for HTTPS.

- **Path:** The path indicates the complete path to the resource on the webserver. It can be like /software/htp/index.html.
- **Query String:** It is the string that contains the name and value pair. If it is used in a URL, it follows the path component and gives the information. Such as "?key1=value1&key2=value2".
- **Fragment:** It is also an optional component, preceded by a hash(#) symbol.
- It consists of a fragment identifier that provides direction to a secondary resource.

- **Key differences between URI and URL**
- URI contains both URL and URN to identify the name and location or both of a resource; in contrast, URL is a subset of URI and only identifies the location of the resource.
- The example of URI is urn:isbn:0-476-27557-4, whereas the example of URL, is https://google.com.
- The URI can be used to find resources in HTML, XML, and other files also, whereas, URL can only be used to locate a web page.
- Each URL can be a URI, whereas all URIs cannot always be URLs.

| URI | URL |
| --- | --- |
| URI is an acronym for Uniform Resource Identifier. | URL is an acronym for Uniform Resource Locator. |
| URI contains two subsets, URN, which tell the name, and URL, which tells the location. | URL is the subset of URI, which tells the only location of the resource. |
| All URIs cannot be URLs, as they can tell either name or location. | All URLs are URIs, as every URL can only contain the location. |
| A URI aims to identify a resource and differentiate it from other resources by using the name of the resource or location of the resource. | A URL aims to find the location or address of a resource on the web. |
| An example of a URI can be ISBN 0-486-35557-4. | An example of an URL is https://www.javatpoint.com. |
| It is commonly used in XML and tag library files such as JSTL and XSTL to identify the resources and binaries. | It is mainly used to search the webpages on the internet. |
| The URI scheme can be protocol, designation, specification, or anything. | The scheme of URL is usually a protocol such as HTTP, HTTPS, FTP, etc. |

# Difference between URL and URI:

| URL | URI |
| --- | --- |
| URL is used to describe the identity of an item. | URI provides a technique for defining the identity of an item. |
| URL links a web page, a component of a web page or a program on a web page with the help of accessing methods like protocols. | URI is used to distinguish one resource from other regardless of the method used. |
| URL provides the details about what type of protocol is to be used. | URI doesn't contains the protocol specification. |
| URL is a type of URI. | URI is the superset of URL. |

- **JSON**

- **JSON** or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange.
- The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627.
- The official Internet media type for JSON is application/json.
- The JSON filename extension is .json.

## What is JSON?

- JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange.
- It is a language-independent data format.
- It supports almost every kind of language, framework, and library.
- JSON is an open standard for exchanging data on the web.
- It supports data structures like objects and arrays.
- So, it is easy to write and read data from JSON.
- In JSON, data is represented in key-value pairs, and curly braces hold objects, where a colon is followed after each name.
- The comma is used to separate key-value pairs.
- Square brackets are used to hold arrays, where each value is comma-separated.

- **What is JSON**
  - JSON stands for JavaScript Object Notation.
  - JSON is an open standard data-interchange format.
  - JSON is lightweight and self-describing.
  - JSON originated from JavaScript.
  - JSON is easy to read and write.
  - JSON is language independent.
  - JSON supports data structures such as arrays and objects.
  - **JSON is a text format for storing and transporting data**
  - **JSON is "self-describing" and easy to understand**

- **Features of JSON**
  - Simplicity
    - JSON is easy to read and write.
  - Self-Describing
  - It is a lightweight text-based interchange format.
  - Extensibility
  - Interoperability
  - JSON is language independent.

- **Uses of JSON**
- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

**Example:**

```
{
  "book": [

    {
      "id":"01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },

    {
      "id":"07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }
  ]
}
```

- JSON syntax is basically considered as a subset of JavaScript syntax; it includes the following −
- Data is represented in name/value pairs.
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).
- Square brackets hold arrays and values are separated by ,(comma).
- JSON supports the following two data structures −
- Collection of name/value pairs − This Data Structure is supported by different programming languages.
- Ordered list of values − It includes array, list, vector or sequence etc.

```
{
  "book": [

    {
      "id": "01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },

    {
      "id": "07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }

  ]
}
```

| Sr.No. | Type & Description |
| --- | --- |
| 1 | **Number**<br>double- precision floating-point format in JavaScript |
| 2 | **String**<br>double-quoted Unicode with backslash escaping |
| 3 | **Boolean**<br>true or false |
| 4 | **Array**<br>an ordered sequence of values |
| 5 | **Value**<br>it can be a string, a number, true or false, null etc |

| | |
|---|---|
| 6 | **Object**<br>an unordered collection of key:value pairs |
| 7 | **Whitespace**<br>can be used between any pair of tokens |
| 8 | **null**<br>empty |

- **Why do we use JSON?**
  - Since JSON is an easy-to-use, lightweight language data interchange format in comparison to other available options, it can be used for API integration.

- **Following are the advantages of JSON:**
- **Less Verbose:**
  - In contrast to XML, JSON follows a compact style to improve its users' readability.
  - While working with a complex system, JSON tends to make substantial enhancements.
- **Faster:**
  - The JSON parsing process is faster than that of the XML because the DOM manipulation library in XML requires extra memory for handling large XML files.
  - However, JSON requires less data that ultimately results in reducing the cost and increasing the parsing speed.

- **Readable:**
  - The JSON structure is easily readable and straightforward.
  - Regardless of the programming language that you are using, you can easily map the domain objects.
- **Structured Data:**
  - In JSON, a map data structure is used, whereas XML follows a tree structure.
  - The key-value pairs limit the task but facilitate the predictive and easily understandable model.

# JSON DATA TYPES

| Sr.No. | Type & Description |
|---|---|
| 1 | **Number**<br>double- precision floating-point format in JavaScript |
| 2 | **String**<br>double-quoted Unicode with backslash escaping |
| 3 | **Boolean**<br>true or false |
| 4 | **Array**<br>an ordered sequence of values |
| 5 | **Value**<br>it can be a string, a number, true or false, null etc |

| | |
|---|---|
| 6 | **Object**<br>an unordered collection of key:value pairs |
| 7 | **Whitespace**<br>can be used between any pair of tokens |
| 8 | **null**<br>empty |

# JSON Data Types

| Data Type | Description | Example |
|---|---|---|
| String | A string is always written in double-quotes. It may consist of numbers, alphanumeric and special characters. | "student", "name", "1234", "Ver_1" |
| Number | Number represents the numeric characters. | 121, 899 |
| Boolean | It can be either True or False. | true |
| Null | It is an empty value. | |

- **JSON Objects**
- In JSON, objects refer to dictionaries, which are enclosed in curly brackets, i.e., { }.
- These objects are written in key/value pairs, where the key has to be a string and values have to be a valid JSON data type such as string, number, object, Boolean or null.
- Here the key and values are separated by a colon, and a comma separates each key/value pair.

- For example:
- {"name" : "Jack", "employeeid" : 001, "present" : false}

- **JSON Arrays**
  - In JSON, arrays can be understood as a list of objects, which are mainly enclosed in square brackets [ ].
  - An array value can be a string, number, object, array, boolean or null.

**For example:**

```
[{
"PizzaName" : "Country Feast",
"Base" : "Cheese burst",
"Toppings" : ["Jalepenos", "Black Olives", "Extra cheese", "Sausages", "Cherry tomatoes"],
"Spicy" : "yes",
"Veg" : "yes"
},

{
"PizzaName" : "Veggie Paradise",
"Base" : "Thin crust",
"Toppings" : ["Jalepenos", "Black Olives", "Grilled Mushrooms", "Onions", "Cherry tomatoes"],
"Spicy" : "yes",
"Veg" : "yes"
}
]
```

- **JSON Vs XML**
- JSON stands for JavaScript Object Notation, whereas XML stands for Extensive Markup Language.
- Nowadays, JSON and XML are widely used as data interchange formats, and both have been acquired by applications as a technique to store structured data.

| | |
|---|---|
| JSON is easy to learn. | XML is quite more complex to learn than JSON. |
| It is simple to read and write. | It is more complex to read and write than JSON. |
| It is data-oriented. | It is document-oriented. |
| JSON is less secure in comparison to XML. | XML is highly secured. |
| It doesn't provide display capabilities. | It provides the display capability because it is a markup language. |
| It supports the array. | It doesn't support the array |

Example :

```
[
{
"name" : "Peter",
"employed id" : "E231",
"present" : true,
"numberofdayspresent" : 29
},
{
"name" : "Jhon",
"employed id" : "E331",
"present" : true,
"numberofdayspresent" : 27
}
]
```

Example :

```
<name>
<name>Peter</name>
</name>
```

# Differences between the json and XML.

| JSON | XML |
|---|---|
| JSON stands for javascript object notation. | XML stands for an extensible markup language. |
| The extension of json file is .json. | The extension of xml file is .xml. |
| The internet media type is application/json. | The internet media type is application/xml or text/xml. |
| The type of format in JSON is data interchange. | The type of format in XML is a markup language. |
| It is extended from javascript. | It is extended from SGML. |
| It is open source means that we do not have to pay anything to use JSON. | It is also open source. |
| The object created in JSON has some type. | XML data does not have any type. |
| The data types supported by JSON are strings, numbers, Booleans, null, array. | XML data is in a string format. |
| It does not have any capacity to display the data. | XML is a markup language, so it has the capacity to display the content. |
| JSON has no tags. | XML data is represented in tags, i.e., start tag and end tag. |

| | |
|---|---|
| JSON is quicker to read and write. | XML file takes time to read and write because the learning curve is higher. |
| JSON can use arrays to represent the data. | XML does not contain the concept of arrays. |
| It can be parsed by a standard javascript function. It has to be parsed before use. | XML data which is used to interchange the data, must be parsed with respective to their programming language to use that. |
| It can be easily parsed and little bit code is required to parse the data. | It is difficult to parse. |
| File size is smaller as compared to XML. | File size is larger. |
| JSON is data-oriented. | XML is document-oriented. |
| It is less secure than XML. | It is more secure than JSON. |

## Similarities between the json and XML:

- **Self-describing:** Both json and xml are self-describing as both xml data and json data are human-readable text.
- **Hierarchical:** Both json and xml support hierarchical structure. Here hierarchical means that the values within values.
- **Data interchange format:** JSON and XML can be used as data interchange formats by many different programming languages.
- **Parse:** Both the formats can be easily parsed.
- **Retrieve:** Both formats can be retrieved by using HTTP requests. The methods used for retrieving the data are GET, PUT, POST.

- **JSON Example 1**

```
{"employees":[
    {"name":"Shyam", "email":"shyamjaiswal@gmail.com"},
    {"name":"Bob", "email":"bob32@gmail.com"},
    {"name":"Jai", "email":"jai87@gmail.com"}
]}
```

- **The XML representation of above JSON example is given below.**

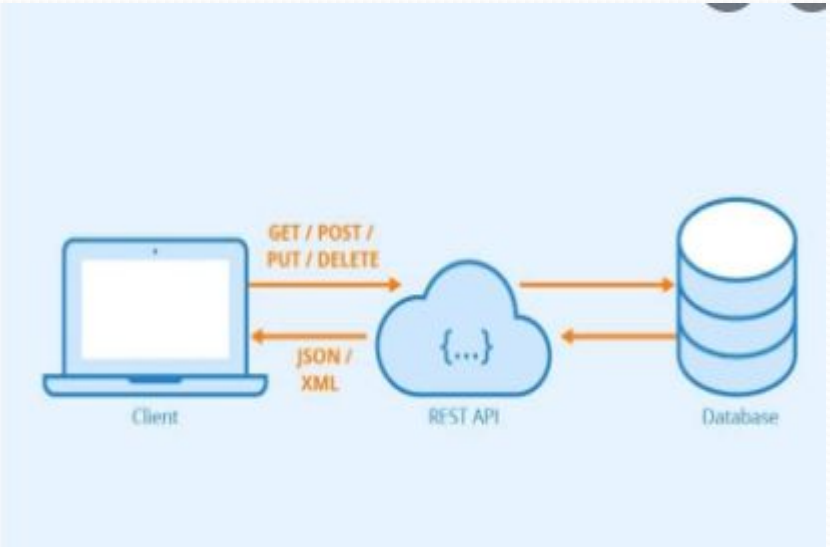
```
<employees>
   <employee>
      <name> Shyam</name>
      <email>shyamjaiswal@gmail.com</email>
   </employee>
   <employee>
      <name>Bob</name>
      <email>bob32@gmail.com</email>
   </employee>
   <employee>
      <name>Jai</name>
      <email>jai87@gmail.com</email>
   </employee>
</employees>
```

- **JSON Comments:**
- JSON doesn't support comments. It is not a standard.
- But you can do some tricks such as adding extra attribute for comment in JSON object, for example:

```
{
   "employee": {
      "name":       "Bob",
      "salary":      56000,
      "comments":    "He is a nice man"
   }
}
```

- **REST — Representational State Transfer**
- REST stands for **Re**presentational **S**tate **T**ransfer (It is sometimes spelled “REST”).
- It relies on a stateless, client-server, cacheable communications protocol — and in virtually all cases, the HTTP protocol is used.
-

- **REST is an architecture** style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

- **What is REST?**
- **REpresentational State Transfer**
- REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other.
- REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server.

- **Separation of Client and Server**
- In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other.
- This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.

- **Statelessness**
- Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa.
- In this way, both the server and the client can understand any message received, even without seeing previous messages.
- This constraint of statelessness is enforced through the use of resources, rather than commands.
- Resources describe any object, document, or thing that you may need to store or send to other services.

- Because REST systems interact through standard operations on resources, they do not rely on the implementation of interfaces.
- These constraints help RESTful applications achieve reliability, quick performance, and scalability, as components that can be managed, updated, and reused without affecting the system as a whole, even during operation of the system.

- RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data.
- Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

# HTTP methods / URIs for collection/item

| | http://api.co/v2/cars/ | http://api.co/v2/cars/1234 |
|---|---|---|
| GET | List all the cars | Retrieve an individual car |
| POST | Create a new car | Error |
| PUT | Replace the entire collection with a whole new list of cars | Replace or create an individual car |
| DELETE | Delete all the cars | Delete an individual car |

- Communication between Client and Server
- In the REST architecture, clients send requests to retrieve or modify resources, and servers send responses to these requests.
- Let’s take a look at the standard ways to make requests and send responses.

- **Making Requests**
- REST requires that a client make a request to the server in order to retrieve or modify data on the server.
- A request generally consists of:
  - **an HTTP verb, which defines what kind of operation to perform**
  - **a header, which allows the client to pass along information about the request**
  - **a path to a resource**
  - **an optional message body containing data**
- **HTTP Verbs**
- There are 4 basic HTTP verbs we use in requests to interact with resources in a REST system:

- GET — retrieve a specific resource (by id) or a collection of resources
- POST — create a new resource
- PUT — update a specific resource (by id)
- DELETE — remove a specific resource by id

- **Headers and Accept parameters:**
- In the header of the request, the client sends the type of content that it is able to receive from the server.
- This is called the Accept field, and it ensures that the server does not send data that cannot be understood or processed by the client.
- The options for types of content are MIME Types (or Multipurpose Internet Mail Extensions.
- MIME Types, used to specify the content types in the Accept field, consist of a type and a subtype. They are separated by a slash (/).
- For example, a text file containing HTML would be specified with the type text/html.
- If this text file contained CSS instead, it would be specified as text/css. A generic text file would be denoted as text/plain.
- This default value, text/plain, is not a catch-all, however.
- If a client is expecting text/css and receives text/plain, it will not be able to recognize the content.

- **Paths**
- Requests must contain a path to a resource that the operation should be performed on.
- Paths should contain the information necessary to locate a resource

- **Sending Responses**
- Content Types
- In cases where the server is sending a data payload to the client, the server must include a content-type in the header of the response.
- This content-type header field alerts the client to the type of data it is sending in the response body.
- These content types are MIME Types, just as they are in the accept field of the request header.
- The content-type that the server sends back in the response should be one of the options that the client specified in the accept field of the request.

- For example, when a client is accessing a resource with id 23 in an articles resource with this GET Request:

- GET /articles/23 HTTP/1.1
- Accept: text/html, application/xhtml

# Response Codes

- Responses from the server contain status codes to alert the client to information about the success of the operation.
- As a developer, you do not need to know every status code (there are many of them), but you should know the most common ones and how they are used:

| Status code | Meaning |
| --- | --- |
| 200 (OK) | This is the standard response for successful HTTP re |
| 201 (CREATED) | This is the standard response for an HTTP request th |
| 204 (NO CONTENT) | This is the standard response for successful HTTP re |
| 400 (BAD REQUEST) | The request cannot be processed because of bad reque |
| 403 (FORBIDDEN) | The client does not have permission to access this r |
| 404 (NOT FOUND) | The resource could not be found at this time. It is |
| 500 (INTERNAL SERVER ERROR) | The generic answer for an unexpected failure if ther |

If we wanted to view all customers, the request would look like this:

```
GET http://fashionboutique.com/customers
Accept: application/json
```

A possible response header would look like:

```
Status Code: 200 (OK)
Content-type: application/json
```

Create a new customer by posting the data:

```
POST http://fashionboutique.com/customers
Body:
{
  “customer”: {
    “name” = “Scylla Buss”,
    “email” = “scylla.buss@codecademy.org”
  }
}
```

The server then generates an `id` for that object and returns it back to the client, with a header like:

```
201 (CREATED)
Content-type: application/json
```

- **What is a REST API?**
- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems**
- 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 Design Principles:**
- At the most basic level, an API is a mechanism that enables an application or service to access a resource within another application or service.
- The application or service that accessing  resourse is called the client, and the application or service containing the resource is called the server.
- Some APIs, such as SOAP or XML-RPC, impose a strict framework on developers.
- But REST APIs can be developed using virtually any programming language and support a variety of data formats.
- The only requirement is that they align to the following six REST design principles - also known as architectural constraints.

## Principles of REST

1. **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

**4.Uniform interface** :–

- By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved.
- In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components.
- REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

**5. Layered system**:-

- The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.

**6. Code on demand (optional)**:-

- REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

- **Resource:**
- The key abstraction of information in REST is a **resource**.
- Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on.
- REST uses a **resource identifier** to identify the particular resource involved in an interaction between components.
- The state of the resource at any particular timestamp is known as **resource representation**.
- A representation consists of data, metadata describing the data and **hypermedia** links which can help the clients in transition to the next desired state.

- The data format of a representation is known as a **media type**.
- The media type identifies a specification that defines how a representation is to be processed. **A truly RESTful API looks like *hypertext***.
- Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

1. **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.

**2. 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.

**3. 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.

**4. Cacheability**:

- When possible, resources should be cacheable on the client or server side.
- Server responses also need to contain information about whether caching is allowed for the delivered resource.
- The goal is to improve performance on the client side, while increasing scalability on the server side.

**5.Layered system architecture**.

- In REST APIs, the calls and responses go through different layers. As a rule of thumb, don't assume that the client and server applications connect directly to each other.
- 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.

**6. 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.