**Difference between HTTP1.1 vs HTTP2**

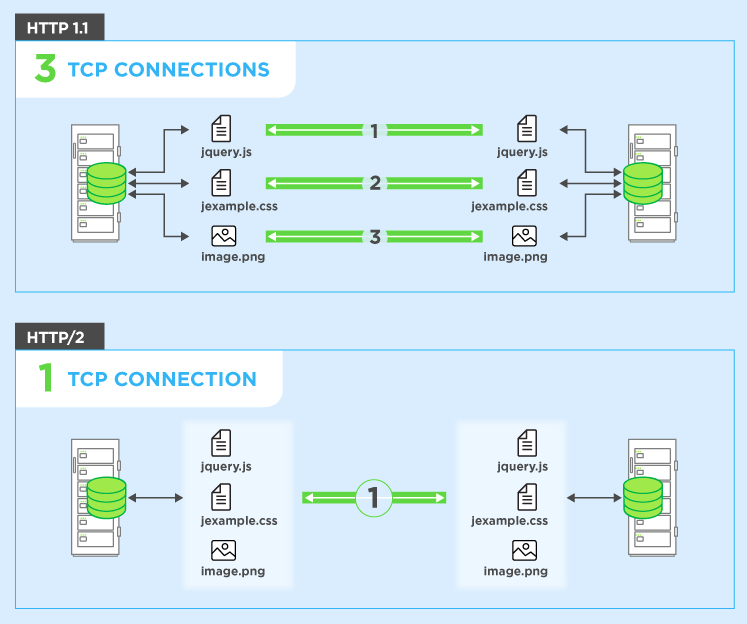
The story about a guest in a restaurant and waiter, that would be an example

For HTTP/1.1 and HTTP/2 protocol with a slight difference. Imagine that waiters are TCP connections and you want to order your meal and a bottle of water. For HTTP/1.1 that would mean that you ask one waiter for your meal and another one for water, hence you would allocate two TCP connections. For HTTP/2 that would mean that you ask only one waiter for both, but he brings them separately. You only allocate one TCP connection and that will already result with lower server load, plus the server would have one extra free connection (waiter) for the next client (guest).

# Request multiplexing

HTTP/2 can send **multiple requests** for data in parallel over a **single** TCP connection. This is **the most** **advanced** **feature** of the HTTP/2 protocol because it **allows you to download web files asynchronously from one server**. Most modern browsers limit TCP connections to one serve.

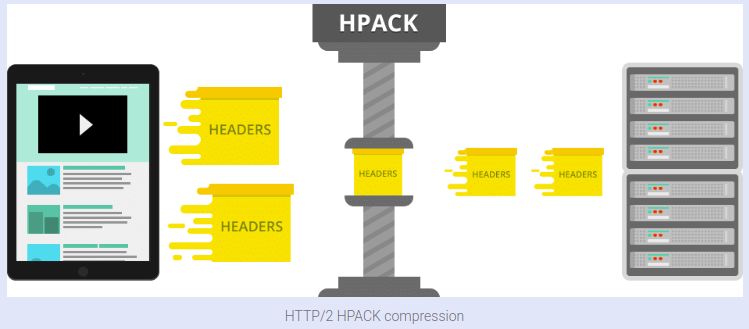
This reduces additional round trip time (RTT), **making your website load faster** without any optimization, and makes domain sharding unnecessary



# Header compression

HTTP/2 compress a large number of redundant header frames. It uses the HPACK specification as a simple and secure approach to header compression. Both client and server maintain a list of headers used in previous client-server requests.

HPACK compresses the individual value of each header before it is transferred to the server, which then looks up the encoded information in a list of previously transferred header values to reconstruct the full header information.



# Binary protocol

The latest HTTP version has evolved significantly in terms of capabilities and attributes such as transforming from a text protocol to a binary protocol. HTTP1.1 used to process text commands to complete request-response cycles. HTTP/2 will use binary commands (in 1s and 0s) to execute the same tasks. This attribute eases complications with framing and simplifies implementation of commands that were confusingly intermixed due to commands containing text and optional spaces.

Browsers using HTTP/2 implementation will convert the same text commands into binary before transmitting it over the network.

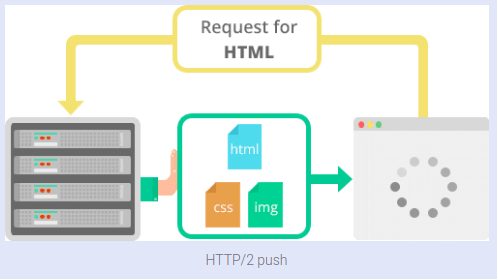


# Benefits:

* Low overhead in parsing data — a critical value proposition in HTTP/2 vs HTTP1.
* Less prone to errors.
* Lighter network footprint.
* Effective network resource utilization.
* Eliminating security concerns associated with the textual nature of HTTP1.1such as response splitting attacks.
* Enables other capabilities of the HTTP/2 including compression, multiplexing, prioritization, flow control and effective handling of TLS.
* Compact representation of commands for easier processing and implementation.
* Efficient and robust in terms of processing of data between client and server.
* Reduced network latency and improved throughput.

# HTTP/2 Server Push

This capability allows the server to send additional cacheable information to the client that isn’t requested but is anticipated in future requests. For example, if the client requests for the resource X and it is understood that the resource Y is referenced with the requested file, the server can choose to push Y along with X instead of waiting for an appropriate client request.



# Benefits:

* The client saves pushed resources in the cache.
* The client can reuse these cached resources across different pages.
* The server can multiplex pushed resources along with originally requested information within the same TCP connection.
* The server can prioritize pushed resources — a key performance differentiator in HTTP/2 vs HTTP1.
* The client can decline pushed resources to maintain an effective repository of cached resources or disable Server Push entirely.
* The client can also limit the number of pushed streams multiplexed concurrently.

The real difference between HTTP/1.1 and HTTP/2 comes with server push example.

Imagine that the guest (Client) asks (sends request) waiter (Server) for a meal, then the waiter gets the meal from the restaurant chef (your application logic), but the waiter also thinks you would need a bottle of water so he brings that too with your meal. The end result of this would be only one TCP connection and only one request that will significantly lower the server load.

**Objects And Its Internal Representation In JavaScript**

Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.  
An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like updateAddress, updateNam, etc.

**Objects and properties**

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

var myCar = new Object();  
myCar.make = 'Ford';  
myCar.model = 'Mustang';  
myCar.year = 1969;

Unassigned properties of an object are [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) (and not [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null)).

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see [property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors)). Objects are sometimes called *associative arrays*, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the myCar object as follows:

myCar['make'] = 'Ford';  
myCar['model'] = 'Mustang';  
myCar['year'] = 1969;

**Creating Objects In JavaScript :**

**Create JavaScript Object with Object Literal**

One of easiest way to create a javascript object is object literal, simply define the property and values inside curly braces as shown below

let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};

**Create JavaScript Object with Constructor**

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below

function Vehicle(name, maker) {  
 this.name = name;  
 this.maker = maker;  
}  
let car1 = new Vehicle(’Fiesta’, 'Ford’);  
let car2 = new Vehicle(’Santa Fe’, 'Hyundai’)  
console.log(car1.name); //Output: Fiesta  
console.log(car2.name); //Output: Santa Fe

**Using the JavaScript Keyword new**

The following example also creates a new JavaScript object with four properties:

Example

var person = new Object();  
person.firstName = “John”;  
person.lastName = “Doe”;  
person.age = 50;  
person.eyeColor = “blue”;