<u>HTTP1.1 vs HTTP2</u>

HTTP1.1, released in 1999, has been the dominant protocol for web communication for over two decades. The salient features are text-based communication and head-of-line blocking where requests are handled in a sequential order.

Later in 2015, a faster and most reliable upgrade of the HTTP1.1 was released i.e. HTTP2. It's performance oriented where it prioritizes content during the loading process. HTTP2 gives developer the flexibility to assign the weighted value to different data streams, depending on which client prioritizes the rendering.

The key differences between these two protocols:

1. Multiplexing: HTTP1.1 is like the single lane, where each request waits in line for its turn. This can lead to delays if one request takes longer. HTTP2, on the other hand, is like a multi-lane, allowing multiple requests to travel simultaneously on a single TCP connection such the resources won't stop one another. This significantly reduces waiting times and improves overall performance.

2. Binary Framing: HTTP1.1 uses plain text for communication, which is large and inefficient. HTTP2 utilizes binary framing, a compact and efficient way to encode data. This smaller size translates to faster transmission and reduced network congestion.

3. Server Push: Traditionally, servers only send data when requested. HTTP2 introduces server push, where the server can anticipate and proactively send resources a client might need. This can significantly improve page load times, especially for complex websites.

4. Header Compression: Headers, which contain information about requests and responses, can be quite large. HTTP1.1 compresses them, but HTTP2 takes it a step further with HPACK, a more advanced compression technique that eliminates redundancy and further shrinks headers. This saves precious bytes and speeds up communication.

5. Stream Prioritization: Not all requests are created equal. HTTP2 allows you to prioritize certain requests over others, ensuring critical resources are delivered first. This is crucial for real-time applications and content that needs immediate attention.

Though HTTP1.1 is still functional, widely supported it's a solution for low budget and low traffic websites. But in the current trend to improve user experiences and SEO rankings developers are looking for optimal performance and future proofing of the websites.

HTTP2 is the preferred option as it loads the pages faster, reduced mobile data usage and improved security. And on the other hand, web hoisting providers made it easy for developers for the seamless transition from HTTP1.1 older websites.

Objects and its Internal Representation

In Javascript Objects are a complex combination of primitive data type and also reference data type in form key-value pairs. Vaguely Objects can be described as unordered collection of related data where keys can be variables, functions often called as properties and methods.

For example, an employee data can be stored using properties like name, id, DOB and methods like salaryCaluclation, taxReduction.

Each object is a unique entity, with its own memory address and internal structure. This structure is like a blueprint, defining how the object's data is organized and accessed. The blueprint can be of class that has constructors, getters, setters, functions and sometimes other objects as well.

When the memory is allocated, there are three things one must know in order to write a clean code:

1. Heap: The actual data associated with the object's properties is stored. It can have primitive data types or even references to other objects.
2. Hidden class: Objects of the same type might share a hidden class called the prototype chain, storing common properties and methods, further improving efficiency.

The closures, garbage collection, and the intricacies of object equality also plays a vital role in retrieving the output from the object class.

Unassigned properties of an object are undefined and not a null value. The Object property name can be any valid JavaScript string(except spaces, hyphen and starts with numbers). Objects are sometimes called associative arrays, since each property is associated with a string value in that [‘propertyName’] can be used to access it. One can use **for..in** to iterate and access the properties of the object.

In JavaScript Objects are created using:

- **Object literal:** Define using properties and it’s values in curly braces

```
let employee = { name: 'Anand',

id:'EL1234',

dob:'12/10/1997' };
```

- **Constructor:** It’s a a function and with help of new keyword

```
function Employee(name, id) {
this.name = name;
this.id = id;
}
let emp1 = new Vehicle(’Sana’, 'GV1234’);
console.log(emp1.name); //Output: Sana
```

- ***new* keyword:** It creates a new JavaScript object with the number of properties of

```
var employee = new Object();

employee.firstName = “Sana”;

employee.lastName = “Zine”;

employee.age = 34;
```

choice

- ***.create()* method:** It’s created using the Object.create() method which allows developer to choose the prototype object type without defining it’s constructor**.**

```
var Employee = {
  role: 'Developer',
  displayRole: function() {
    console.log(this.role);
  }
};
// Create new employee type called emp1

var emp1 = Object.create(Employee);
emp1.displayRole();     // Output: Developer

//updating the property value of the created new object

emp1.role = 'Frontend';
fish.displayRole();     // Output: Frontend
```

We can't directly access the internal memory representation of objects but when Property access and Method invocation are done the JavaScript first checks if it exists within the object itself or not. If not found, it walks up the prototype chain until it finds it or reaches the end. Thus objects allows for code reusability and inheritance.