Introduction to JavaScript

Programming 2 @ EK

KIIL 2026

Agenda

Introduction to JavaScript

- The Javascript environment
  - Server side rendering vs. Client side rendering
  - A script vs. Java program execution
- The javascript language
  - Examples of execution
- Higher order functions & callbacks

Client vs. Server

Frontend

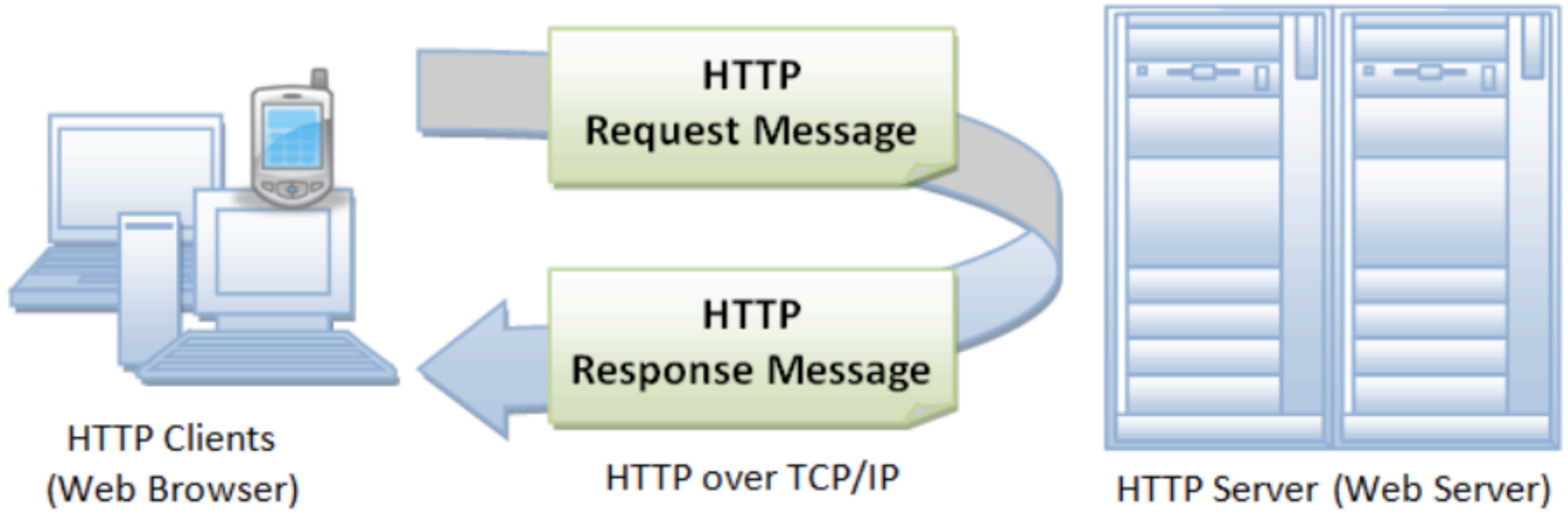
Focuses on layout, animations, content organization, navigation, graphics.

Programming languages:
JavaScript, HTML, CSS

Backend

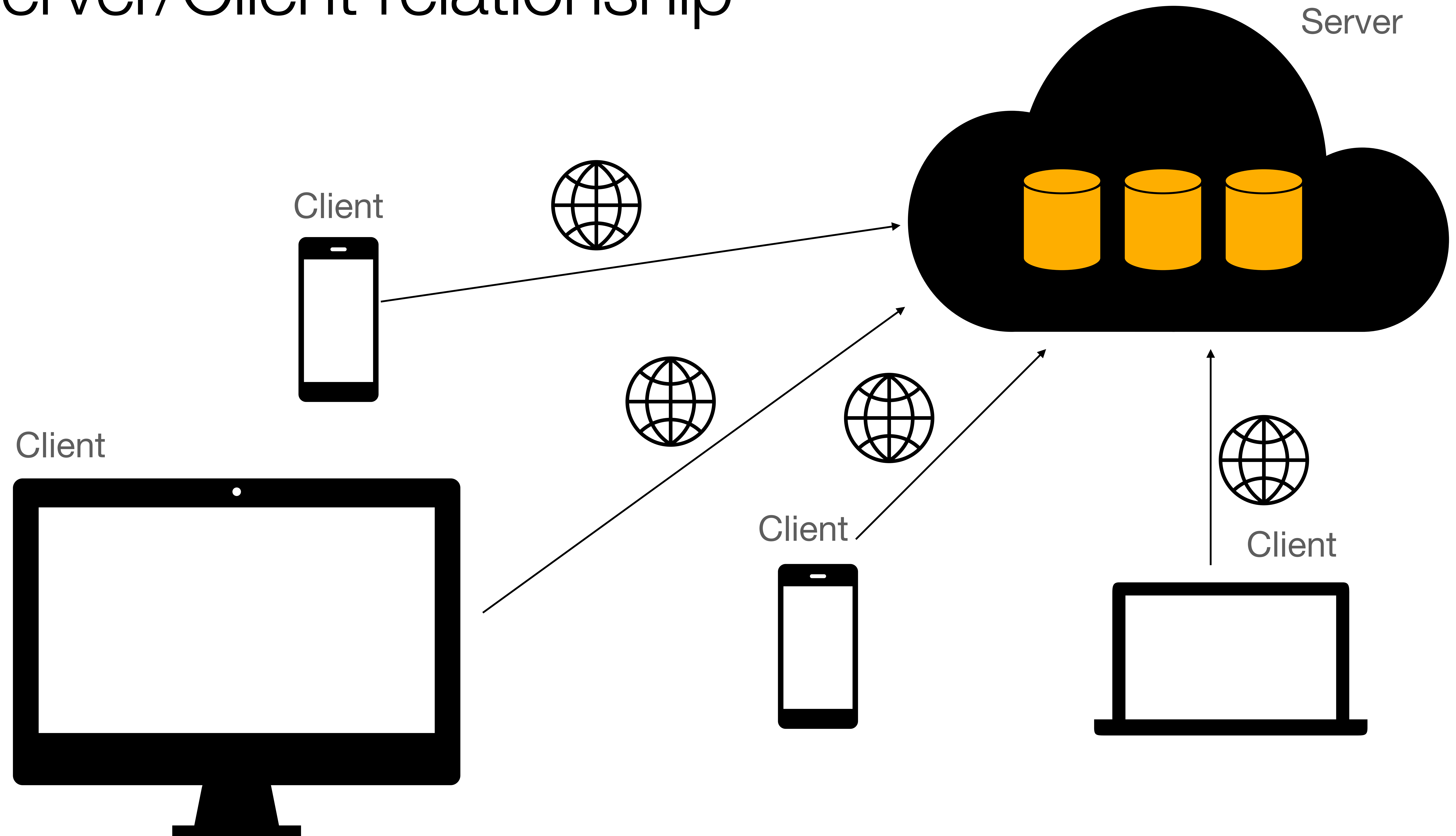
Focuses on building code, debugging, database management.

Programming languages:
Node.js, Python, Java



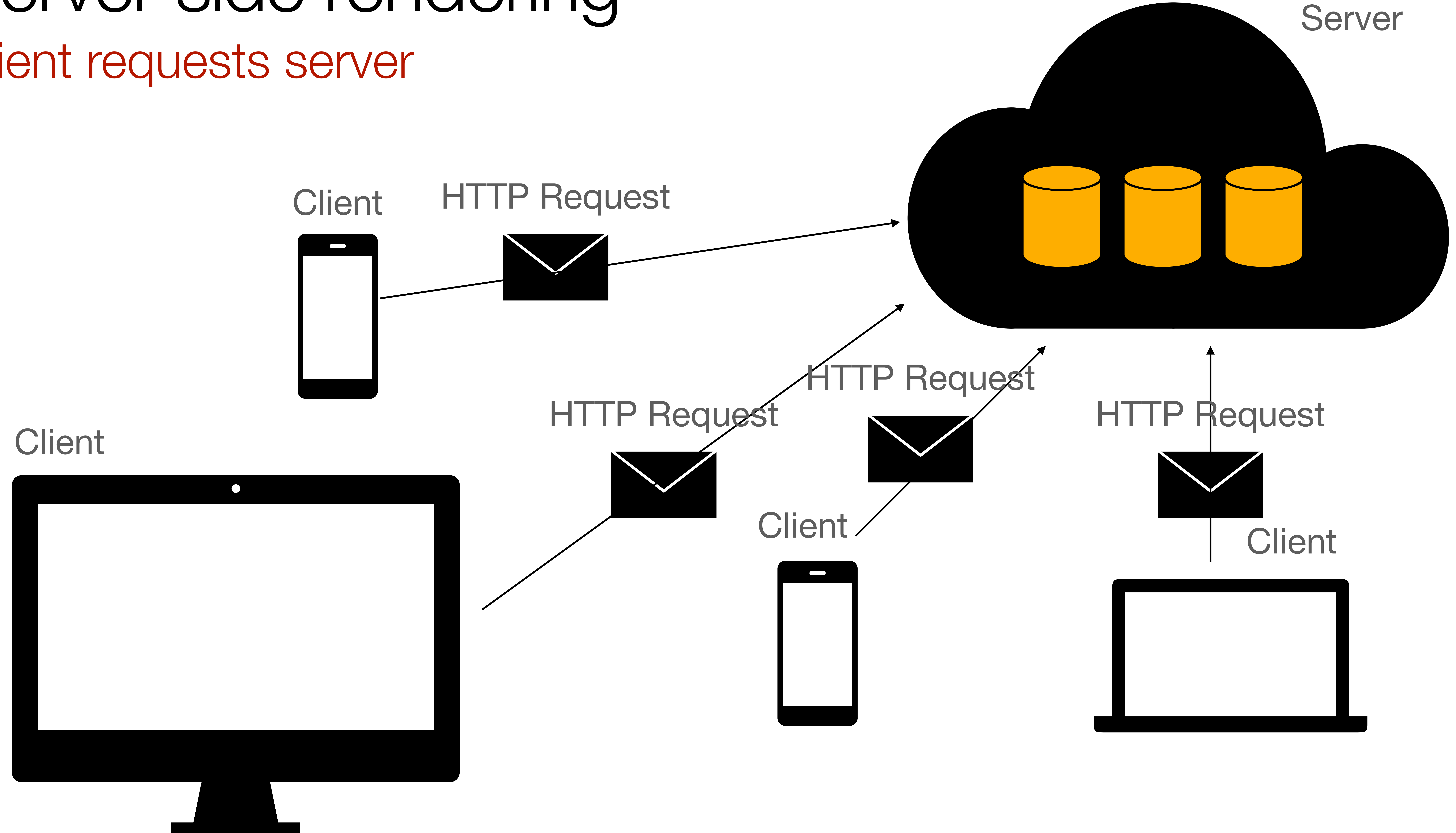
```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

Server/Client relationship



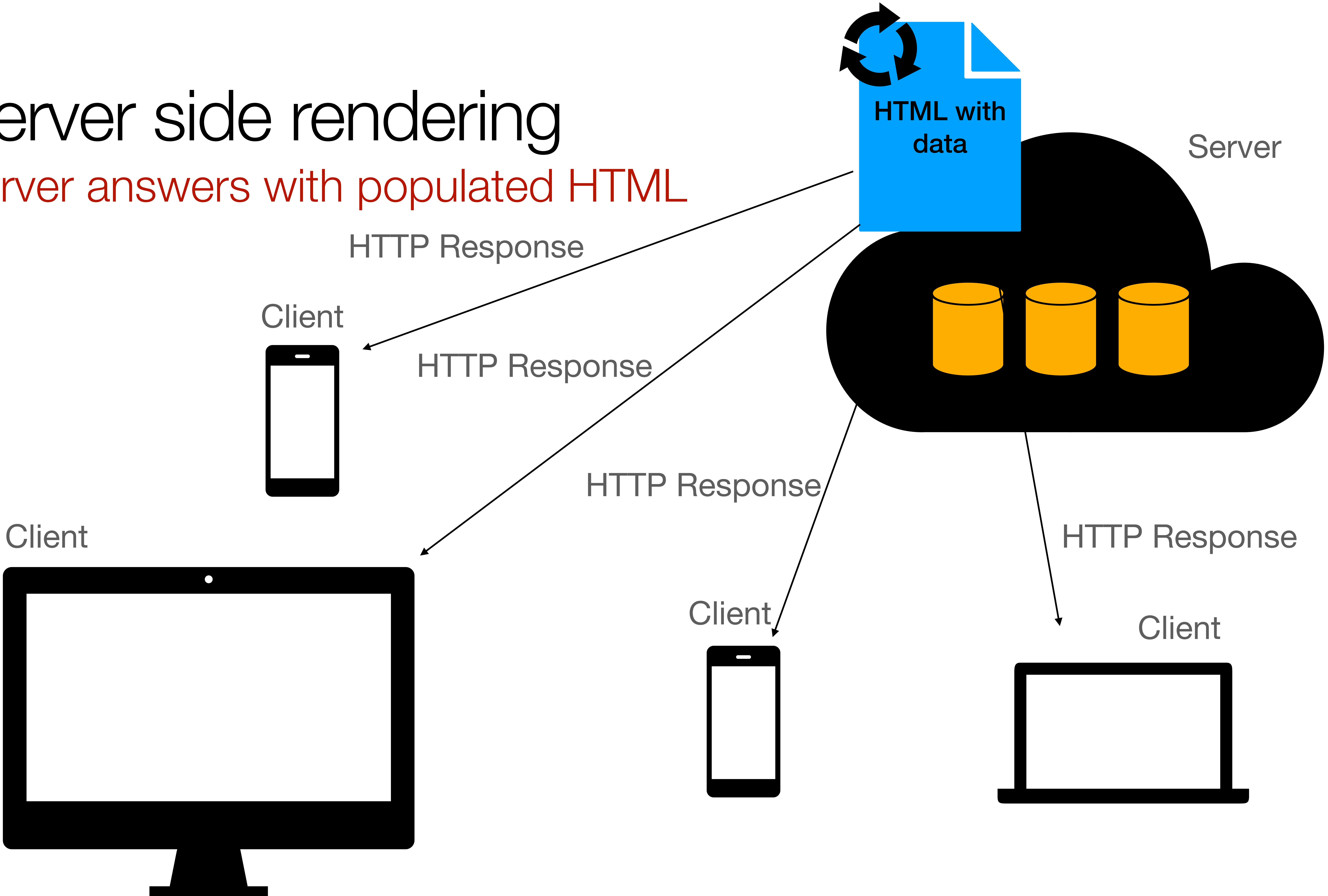
Server side rendering

Client requests server



Server side rendering

Server answers with populated HTML



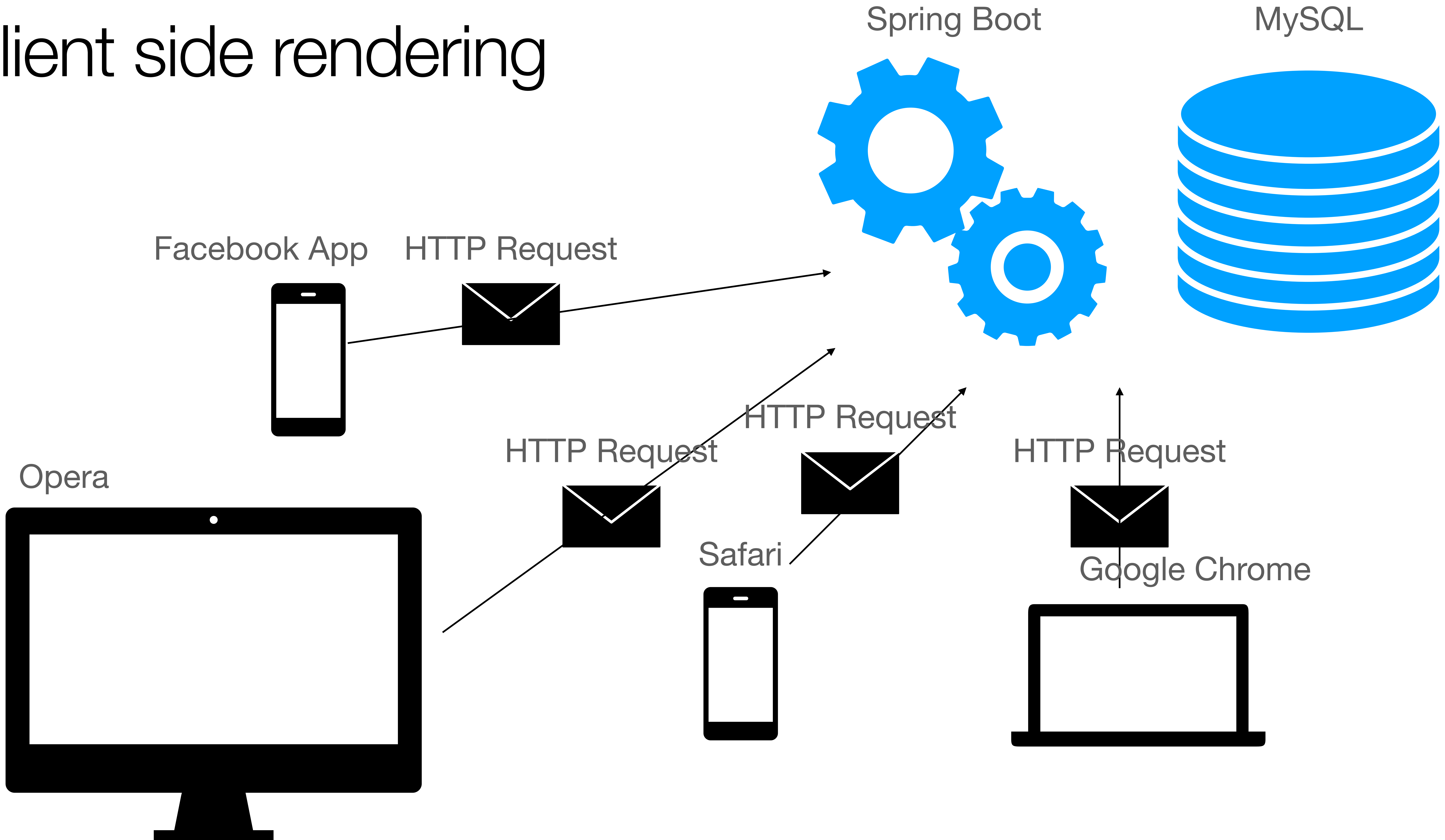
Server side rendering

Thymeleaf code

```
<div th:object="${session.user}">
  <p>Name: <span th:text="*{firstName}">Sebastian</span>.</p>
  <p>Surname: <span th:text="*{lastName}">Pepper</span>.</p>
  <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>
```

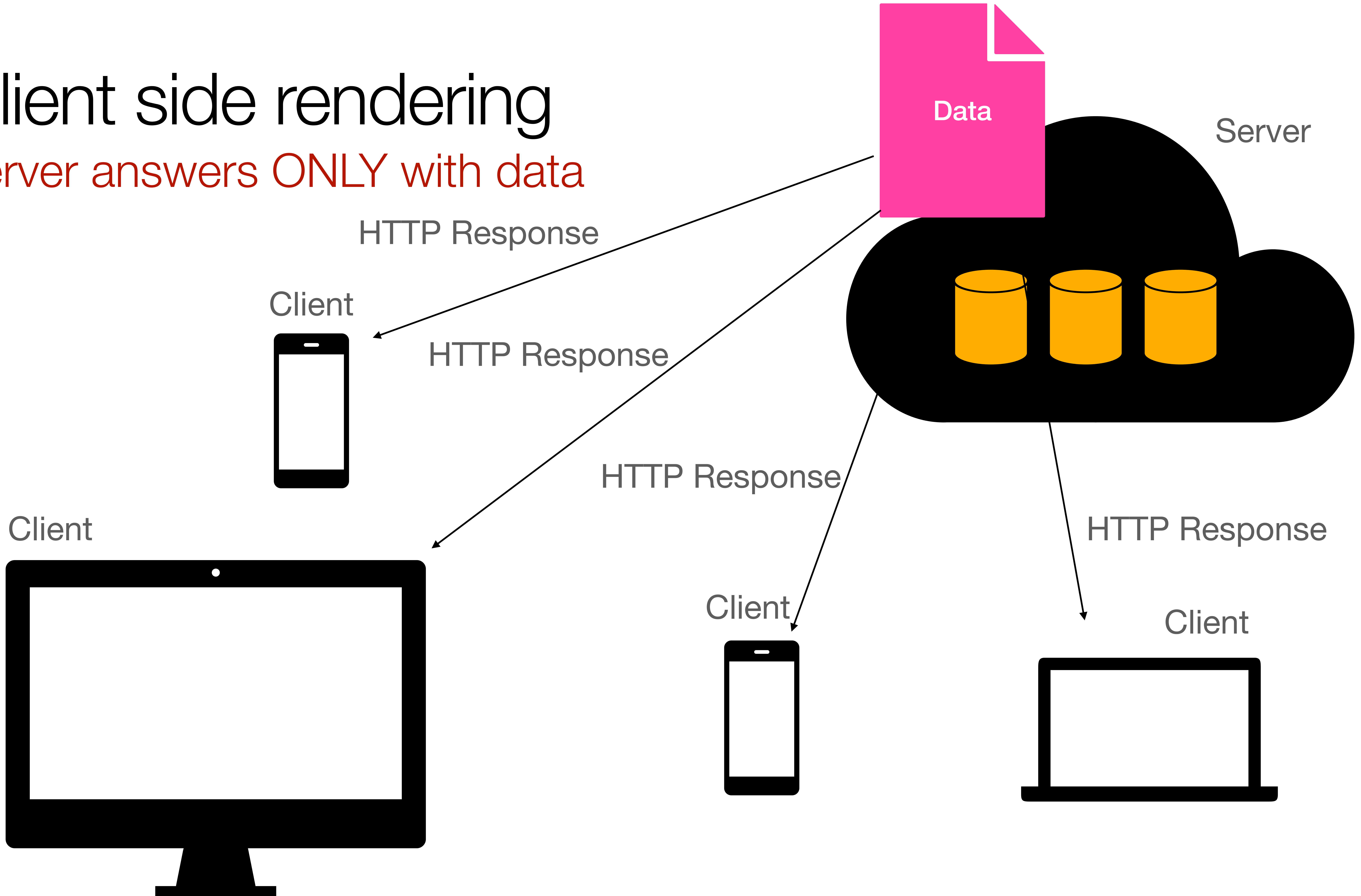
3. Semester Client side rendering

Client side rendering



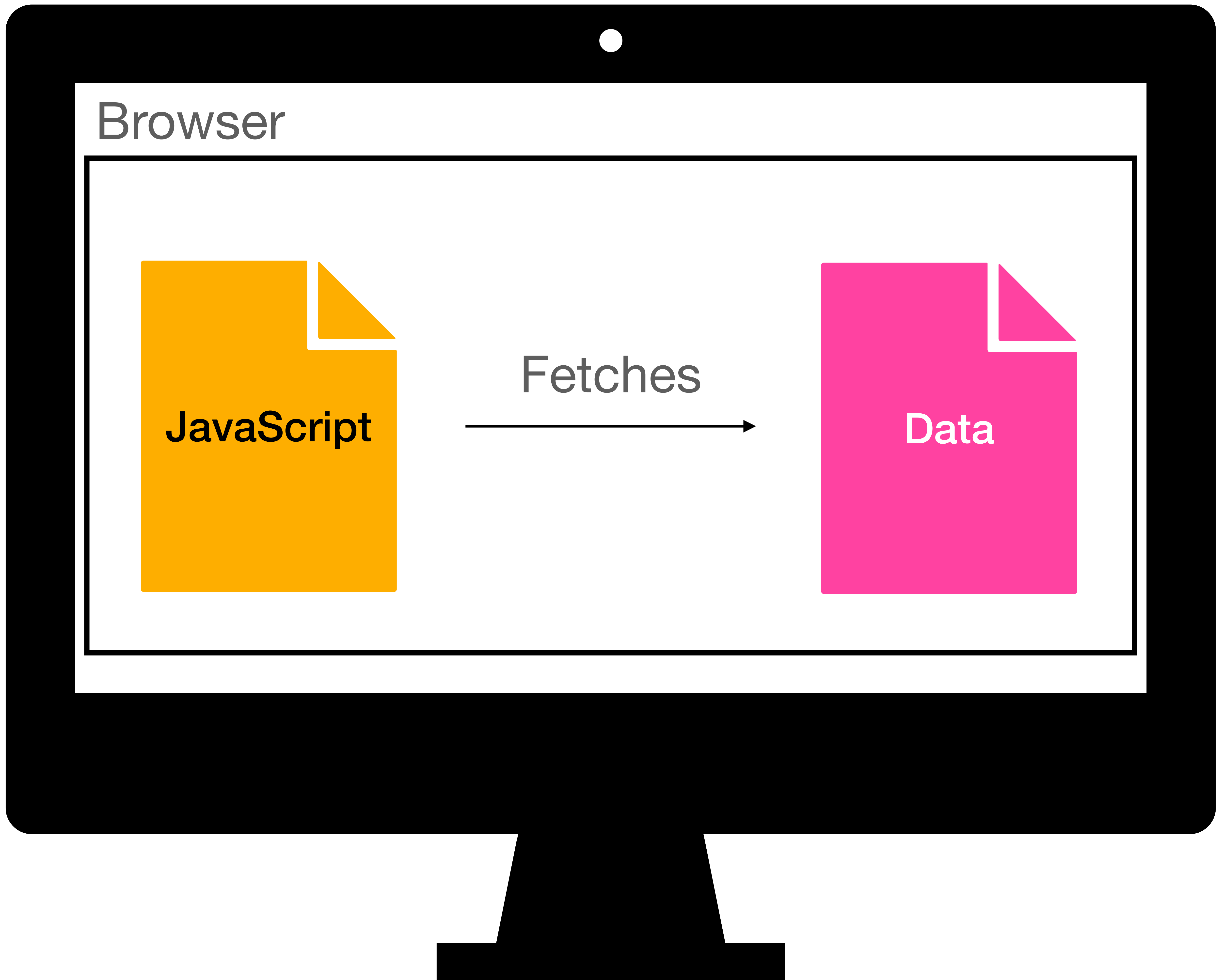
Client side rendering

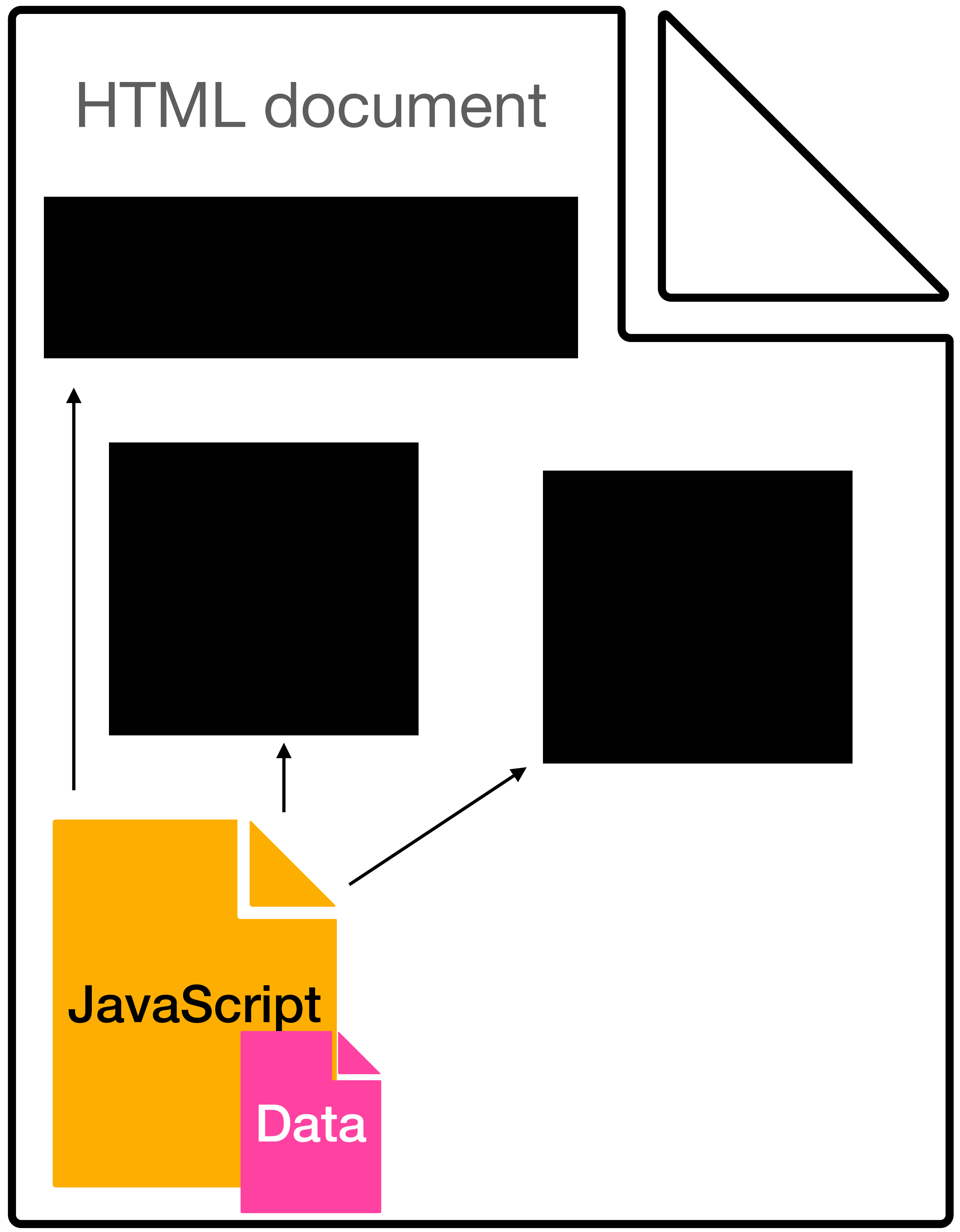
Server answers ONLY with data



Client side rendering

Server answers ONLY with data

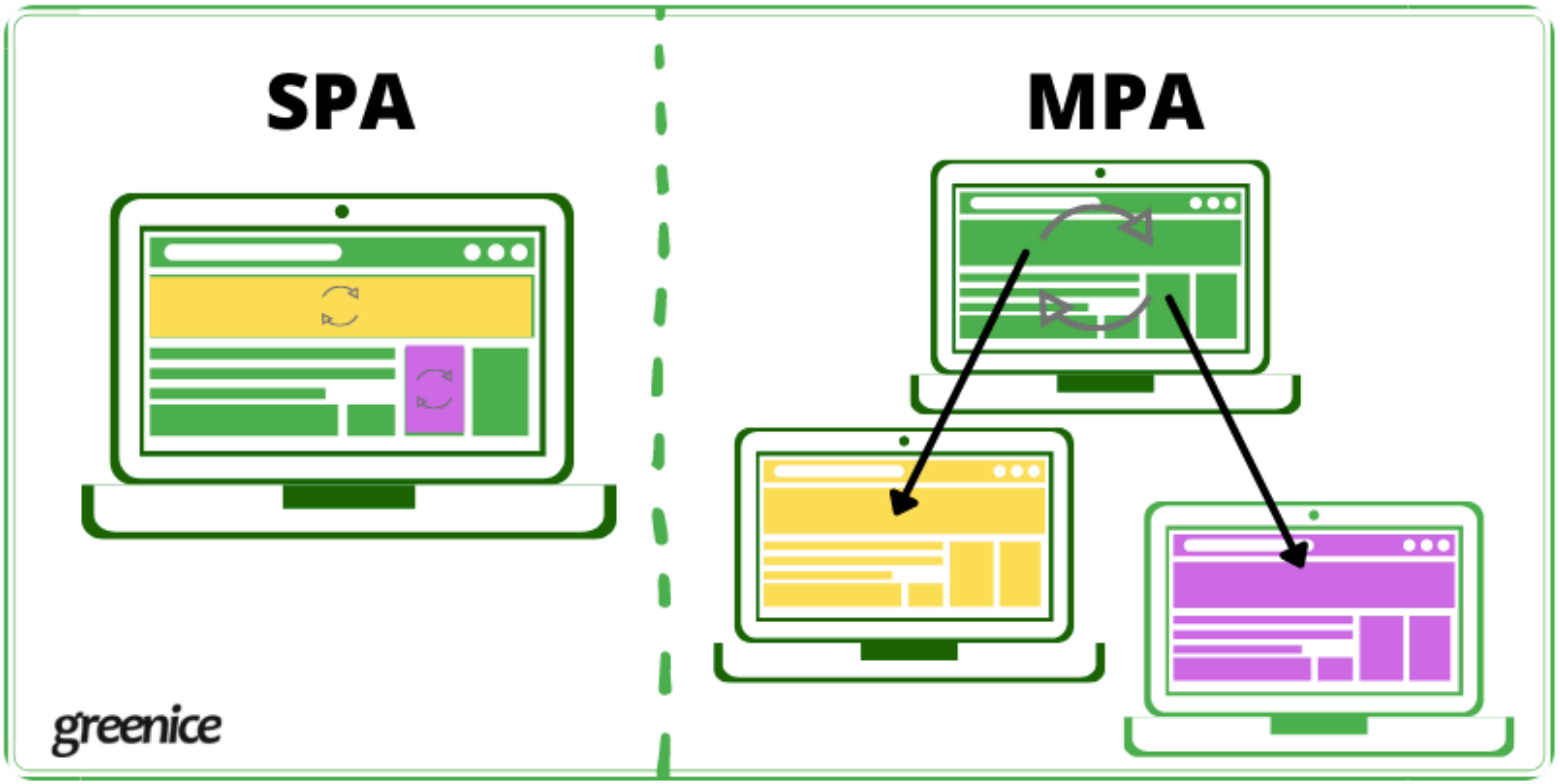




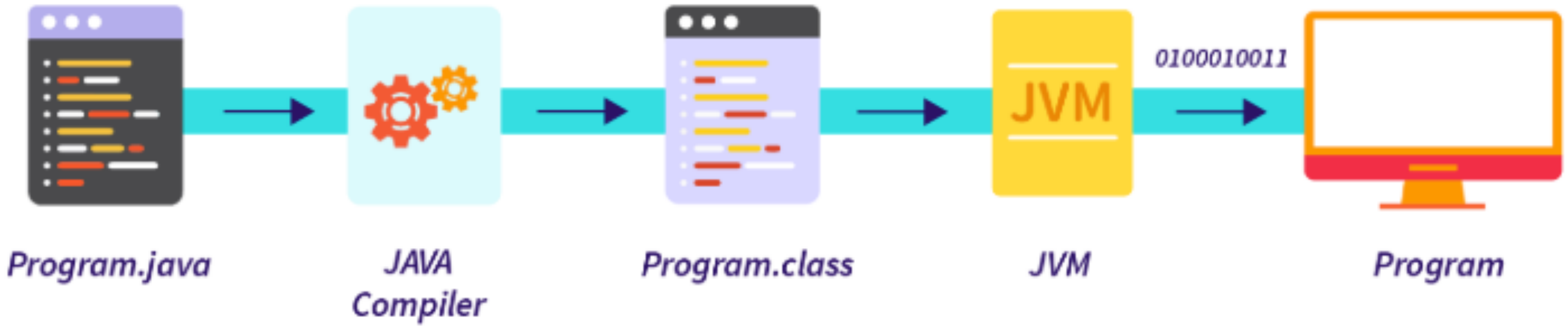
Single Page Applications (SPA):

Only 1 HTML file - Javascript changes everything

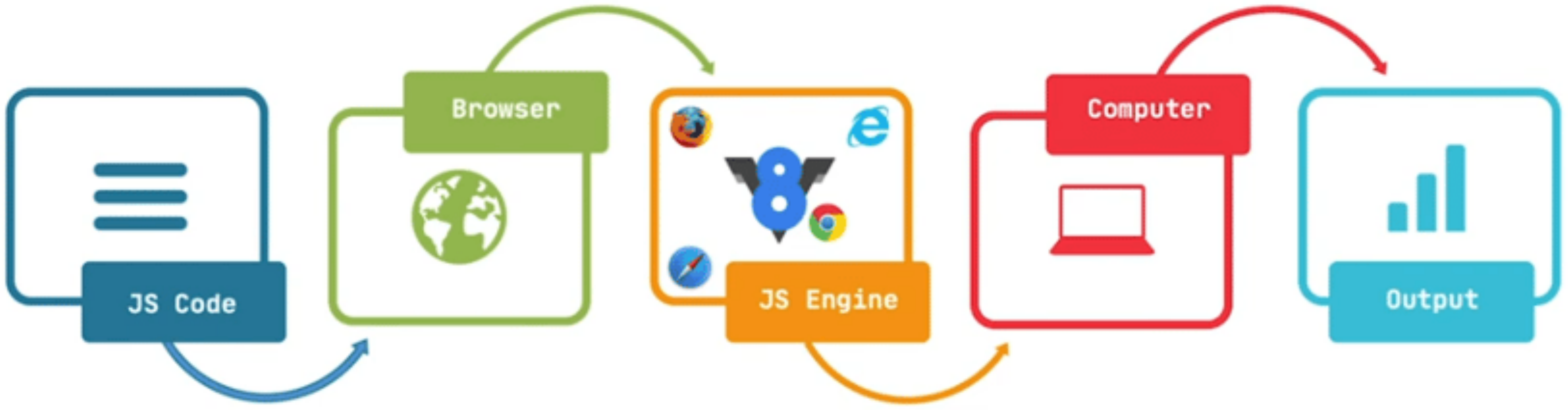
React.js / Vue.js / Angular.js / Svelte.js


SPA
MPA
greenice

Java execution



JavaScript execution



JavaScript can run in the browser and interact with HTML

Exercise: Execute basic JavaScript

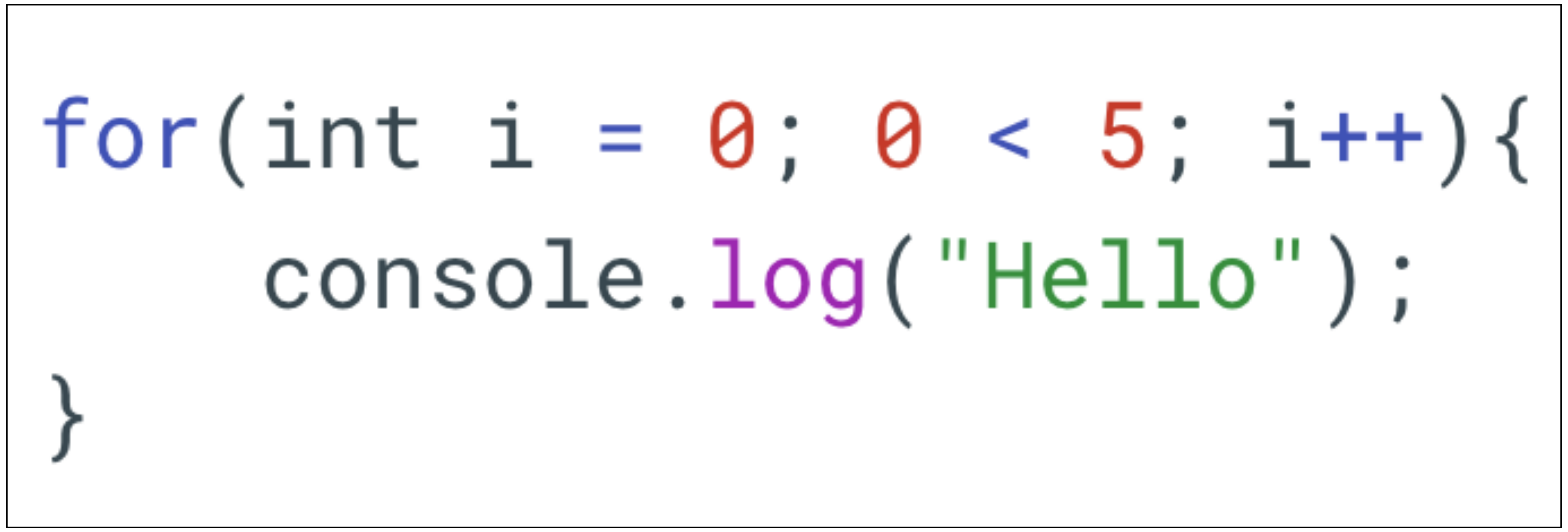
Java vs. JavaScript

Similarities

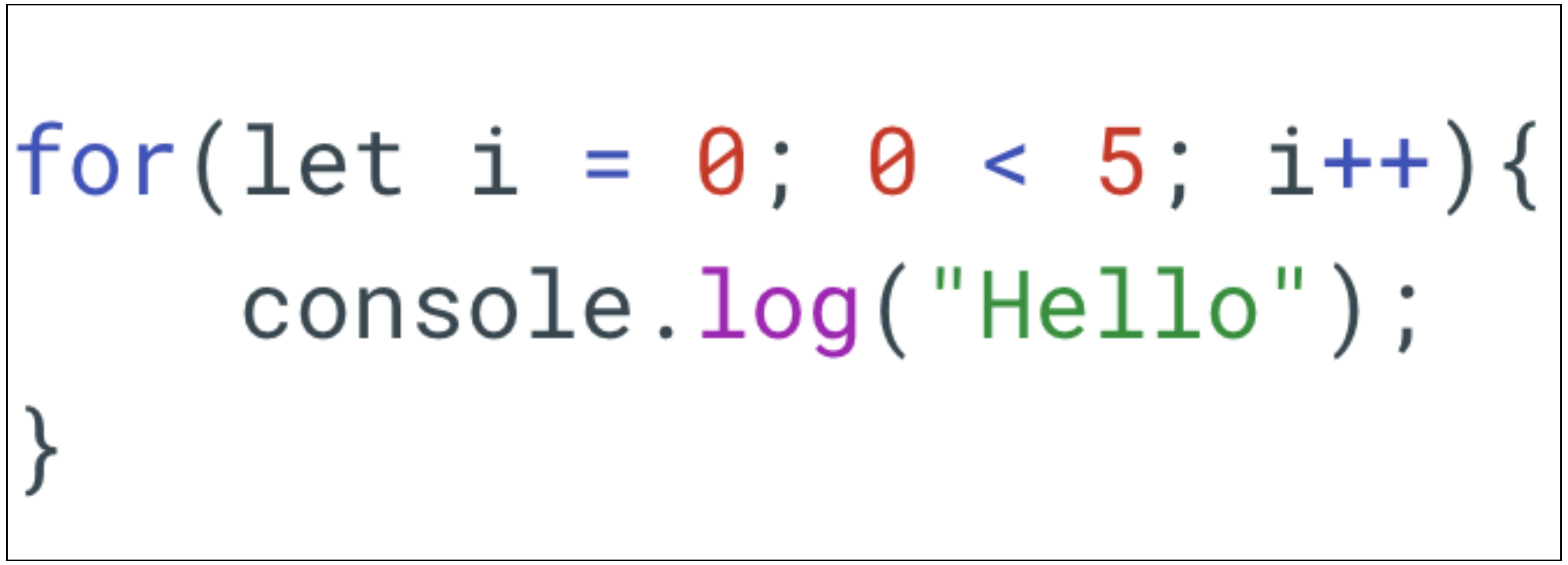
```
if(name.length > 5){
    return name;
}
```

Java vs. JavaScript

Similarities



```
for(int i = 0; 0 < 5; i++){
    console.log("Hello");
}
```



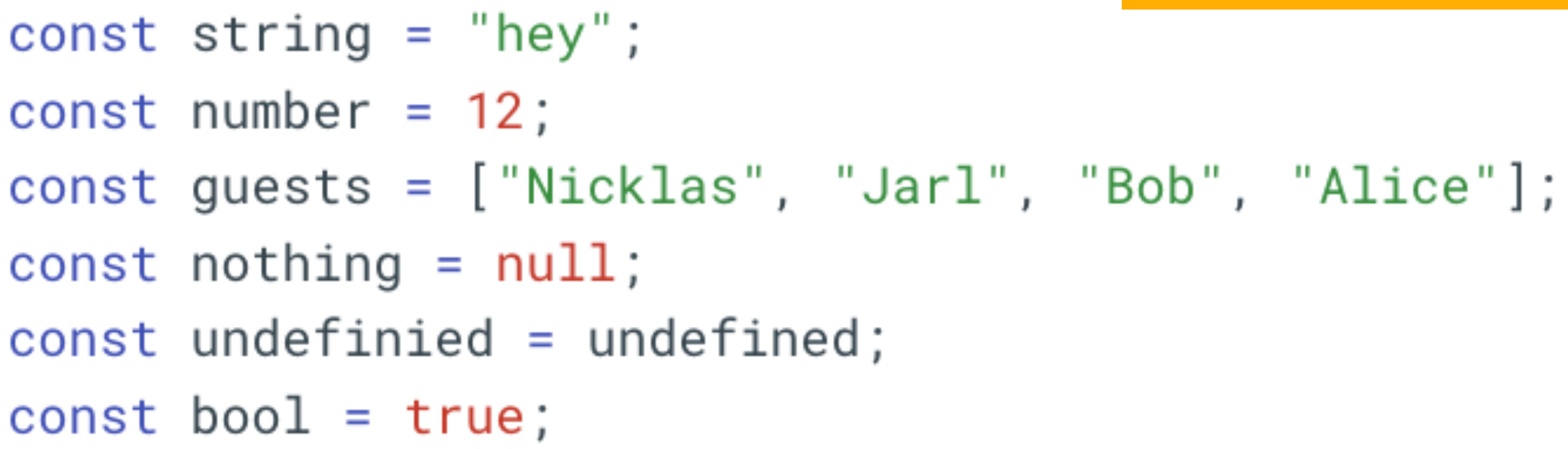
```
for(let i = 0; 0 < 5; i++){
    console.log("Hello");
}
```

JavaScript uses dynamic typing

Java uses static typing

No Data types

Type is inferred by the interpreter

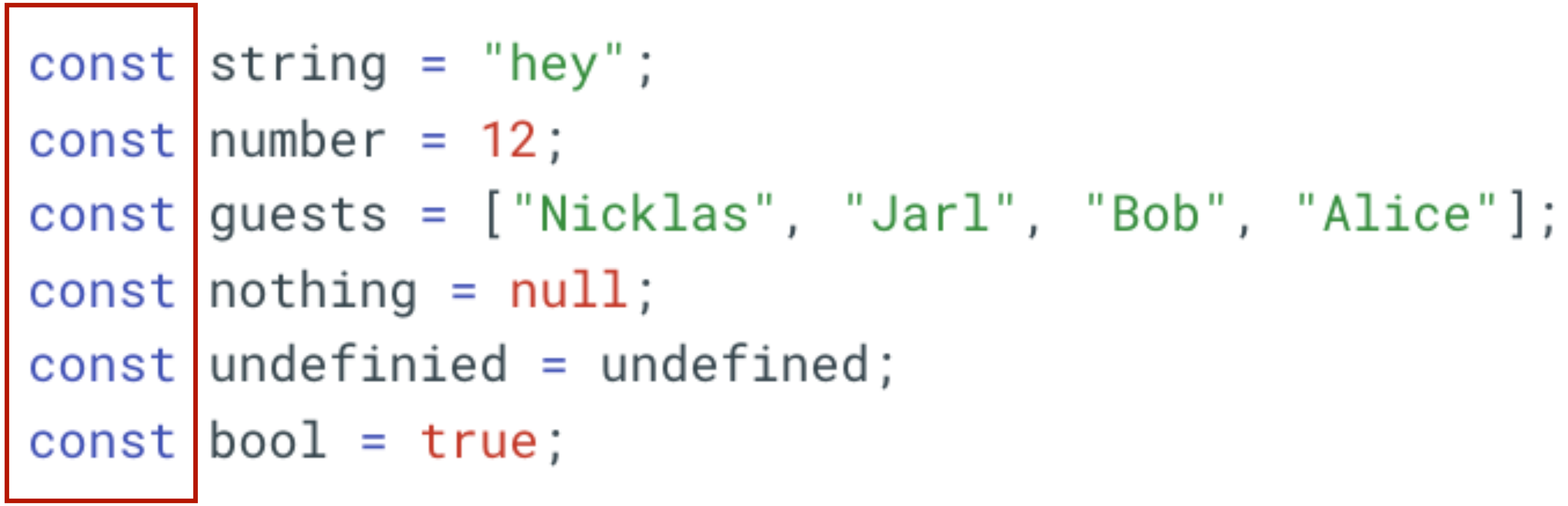


```
const string = "hey";
const number = 12;
const guests = ["Nicklas", "Jarl", "Bob", "Alice"];
const nothing = null;
const undefinied = undefined;
const bool = true;
```

A constant cannot be changed

JavaScript uses dynamic typing

Java uses static typing



```
const string = "hey";
const number = 12;
const guests = ["Nicklas", "Jarl", "Bob", "Alice"];
const nothing = null;
const undefinied = undefined;
const bool = true;
```

var vs. let

Variables: Scope

Bad	Is function scoped or globally scoped

```
var country = "Denmark"
```

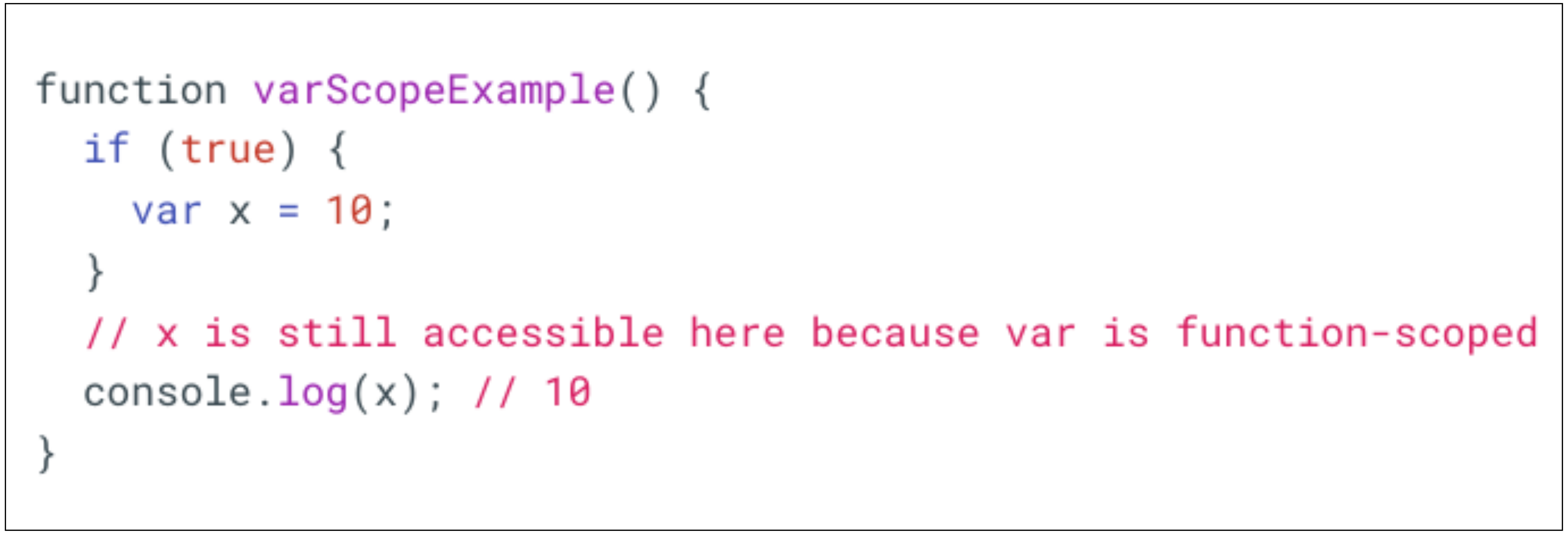
Good	Is block scoped

```
let country = "Denmark"
```

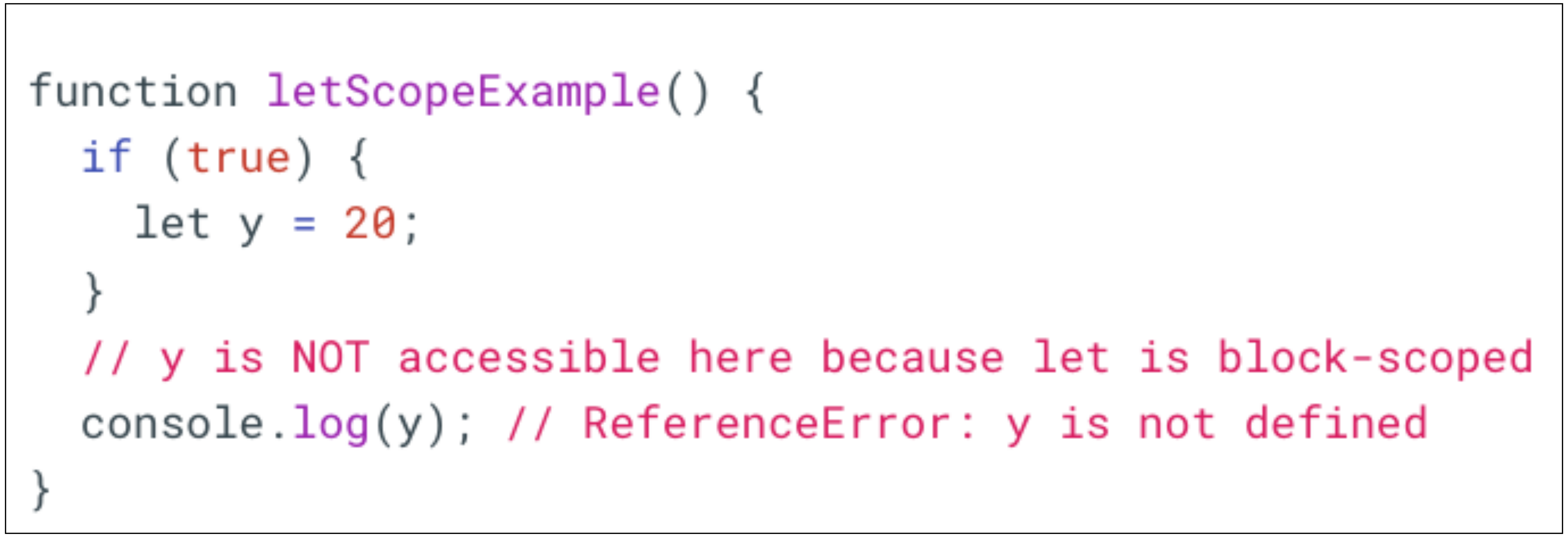
Further explanation: https://javascript.info/var

var vs. let

Variables: Scope



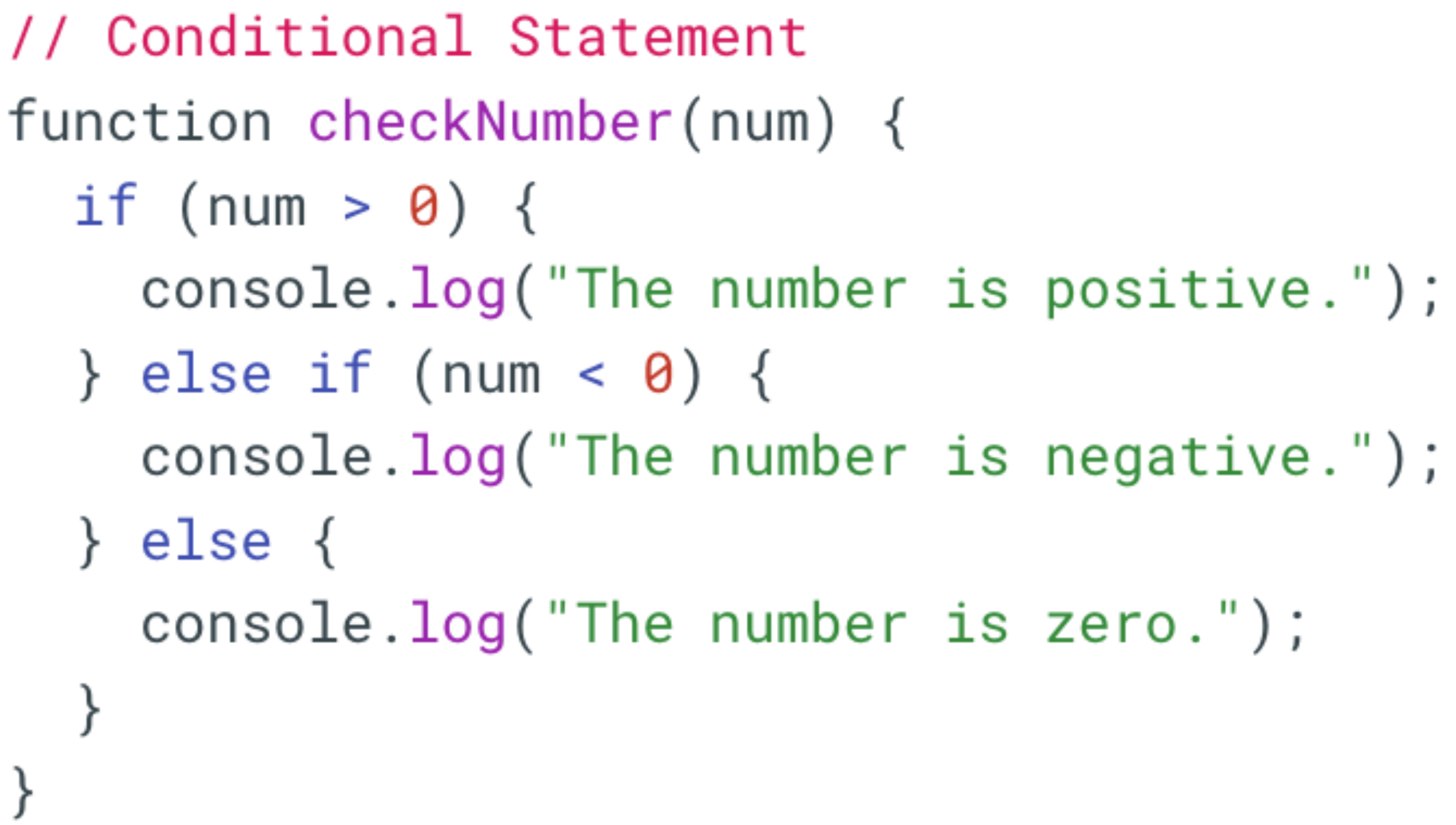
```
function varScopeExample() {
  if (true) {
    var x = 10;
  }
  // x is still accessible here because var is function-scoped
  console.log(x); // 10
}
```



```
function letScopeExample() {
  if (true) {
    let y = 20;
  }
  // y is NOT accessible here because let is block-scoped
  console.log(y); // ReferenceError: y is not defined
}
```

If statement

Control flow



```
// Conditional Statement
function checkNumber(num) {
  if (num > 0) {
    console.log("The number is positive.");
  } else if (num < 0) {
    console.log("The number is negative.");
  } else {
    console.log("The number is zero.");
  }
}
```

Loops

Control flow

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}

while (start >= 0) {
    console.log(start);
    start--;
  }
```

Functions

Control flow

```
function isValidPassword(password){
    if(length > 10){
        return true;
    }
    else{
        return false
    }
}

const result = isValidPassword("password")
```

Make groups

Exercises: Javascript Control flow

Java vs. JavaScript

Java is class-based

```java
public class Main {
    public static void main(String[] args) {
        // Create two Pokemon objects
        Pokemon pikachu = new Pokemon("Pikachu", "Electric", 5);
        Pokemon charmander = new Pokemon("Charmander", "Fire", 8);

        // Show their details
        pikachu.displayInfo();
        charmander.displayInfo();

        // Both Pokemon attack!
        pikachu.attack();
        charmander.attack();
    }
}
```

```java
public class Pokemon {
    private String name;
    private String type;
    private int level;

    // Constructor
    public Pokemon(String name, String type, int level) {
        this.name = name;
        this.type = type;
        this.level = level;
    }

    // Example method to simulate attacking
    public void attack() {
        System.out.println(name + " attacks with a " + type + "-type move!");
    }

    // Example method to display Pokemon information
    public void displayInfo() {
        System.out.println("Name: " + name
            + ", Type: " + type
            + ", Level: " + level);
    }

    // Getter and setter methods (optional)
    public String getName() {
        return name;
    }

    public String getType() {
        return type;
    }

    public int getLevel() {
        return level;
    }

    public void setLevel(int level) {
        this.level = level;
    }
}
```

Objects

No need for classes

Javascript Object

```
const pokemon = {
    name:"Pikachu",
    type: "Electric",
    generation: 1,
    hasEvolution: true,
    makeSound: function(){
        console.log("Pika pikachu");
    }
}
```

```
//Prints Pikachu
console.log(name)

//Prints Pika Pikachi
pokemon.makeSound()
```

Object exercises