JavaScript

An Overview

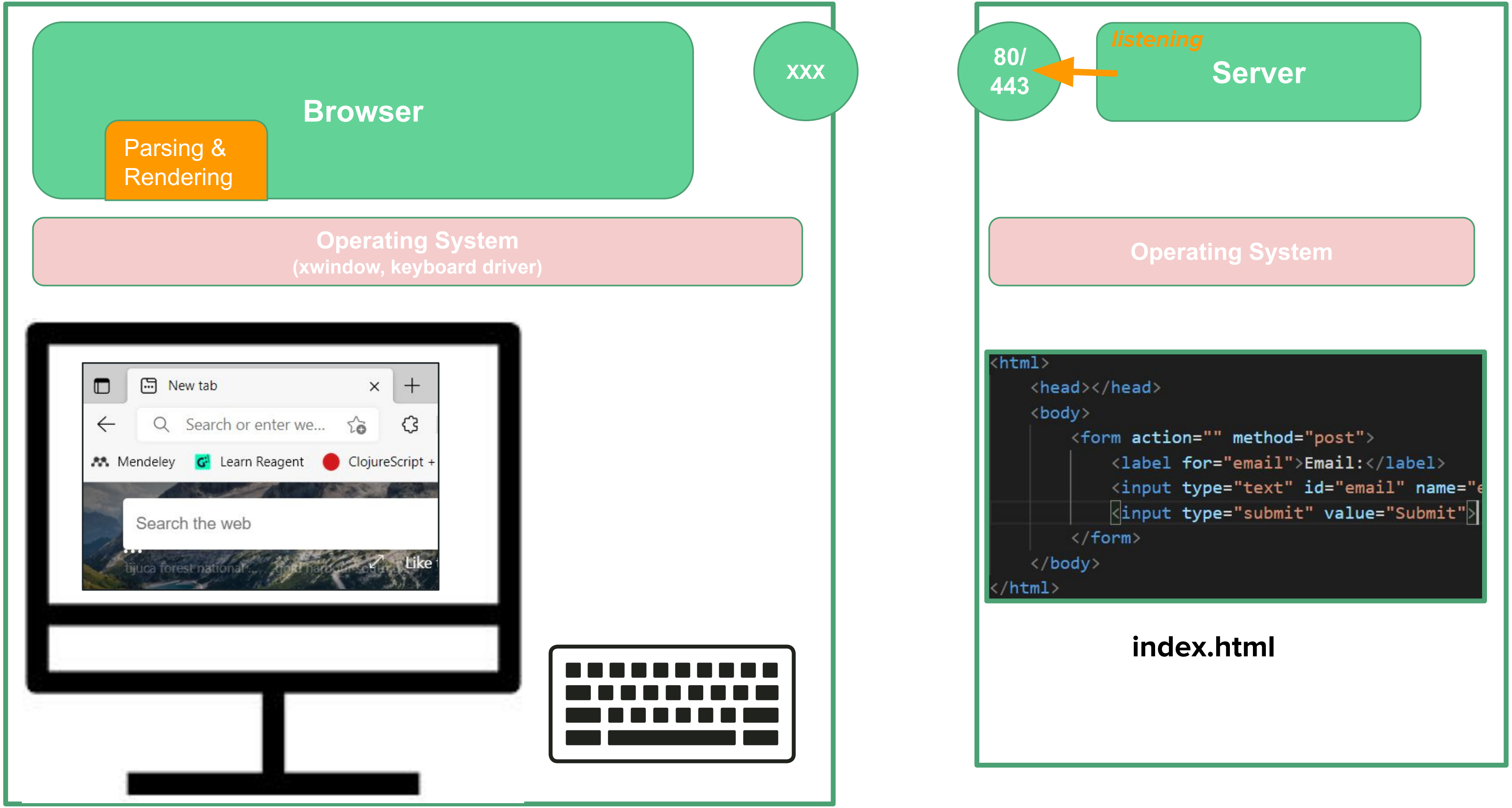
The Web Revolution

Tim Berners-Lee, a British scientist, invented the World Wide Web (WWW) in 1989, while working at CERN.

The Web was originally conceived and developed to meet the demand for **automated information-sharing between scientists** in universities and institutes around the world.

First web page: http://info.cern.ch/hypertext/WWW/TheProject.html

Source: https://home.cern/science/computing/birth-web/short-history-web

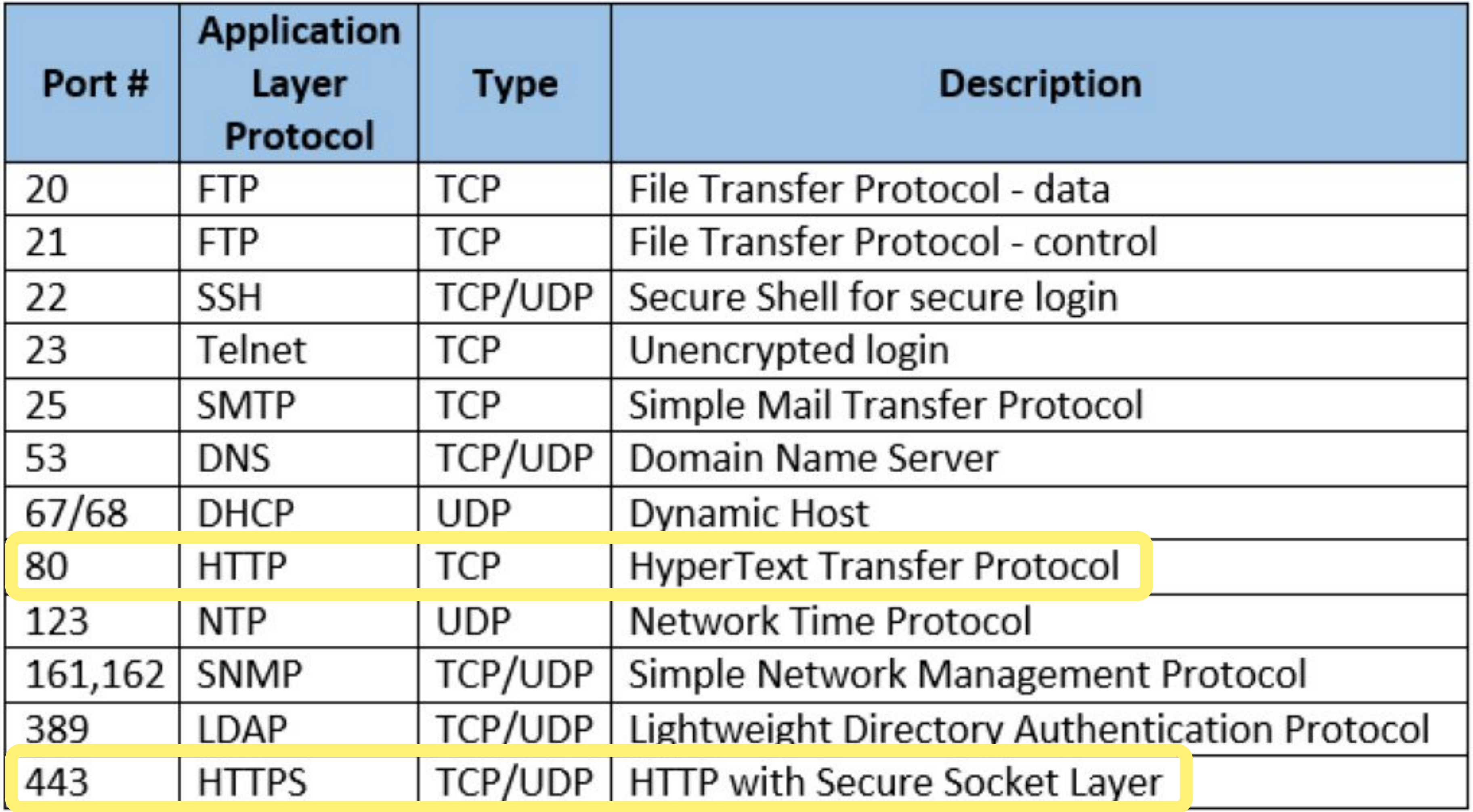


```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

index.html

Client Machine

Server Machine



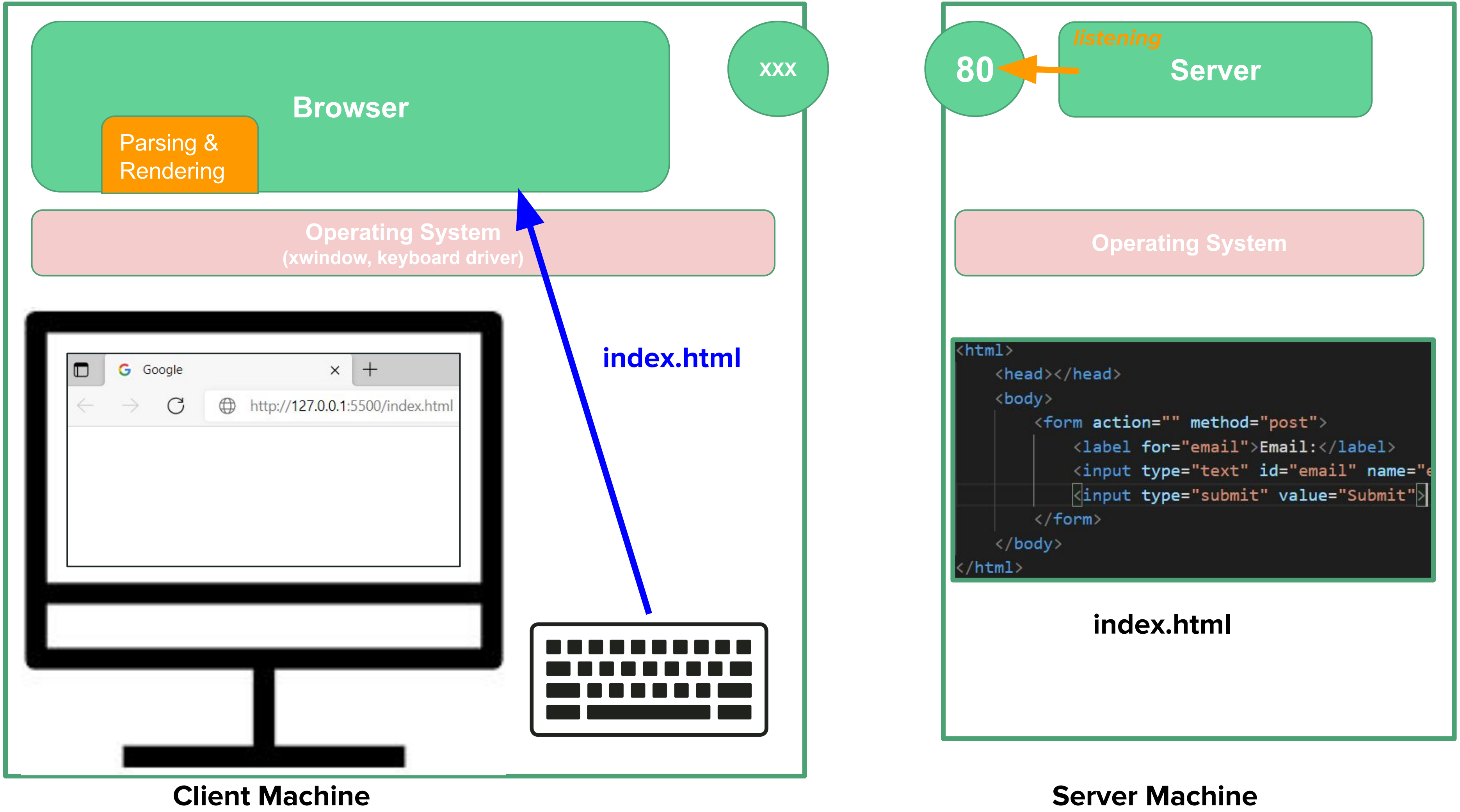
Port #	Application Layer Protocol	Type	Description
20	FTP	TCP	File Transfer Protocol - data
21	FTP	TCP	File Transfer Protocol - control
22	SSH	TCP/UDP	Secure Shell for secure login
23	Telnet	TCP	Unencrypted login
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP/UDP	Domain Name Server
67/68	DHCP	UDP	Dynamic Host
80	HTTP	TCP	HyperText Transfer Protocol
123	NTP	UDP	Network Time Protocol
161,162	SNMP	TCP/UDP	Simple Network Management Protocol
389	LDAP	TCP/UDP	Lightweight Directory Authentication Protocol
443	HTTPS	TCP/UDP	HTTP with Secure Socket Layer

What is a server?

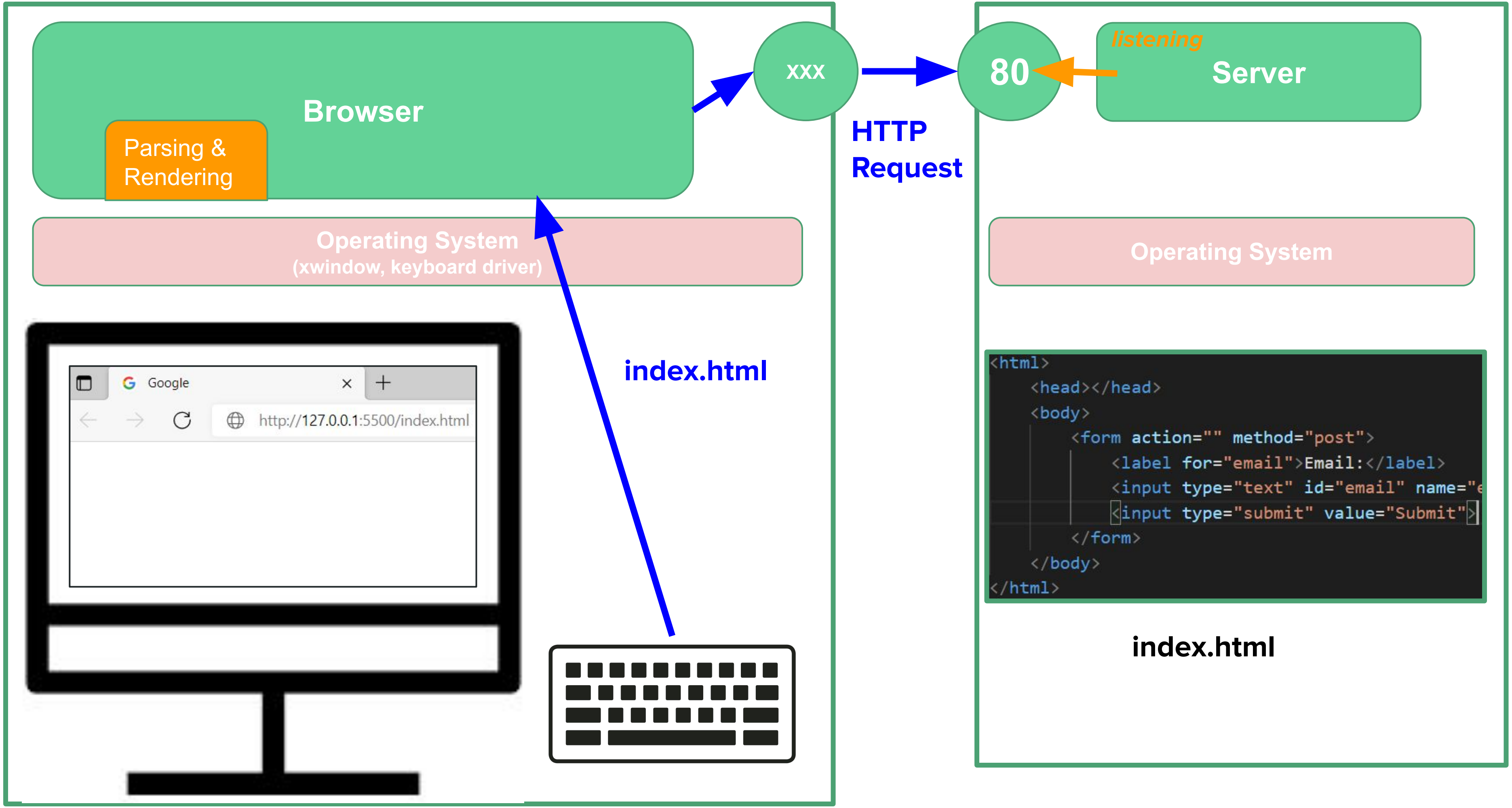
What is a HTTP server?

Can your Laptop be a (HTTP) server?

How?



```html
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```



Browser

Parsing & Rendering

xxx

HTTP Request

80

listening

Server

Operating System
(xwindow, keyboard driver)

Operating System

Google

http://127.0.0.1:5500/index.html

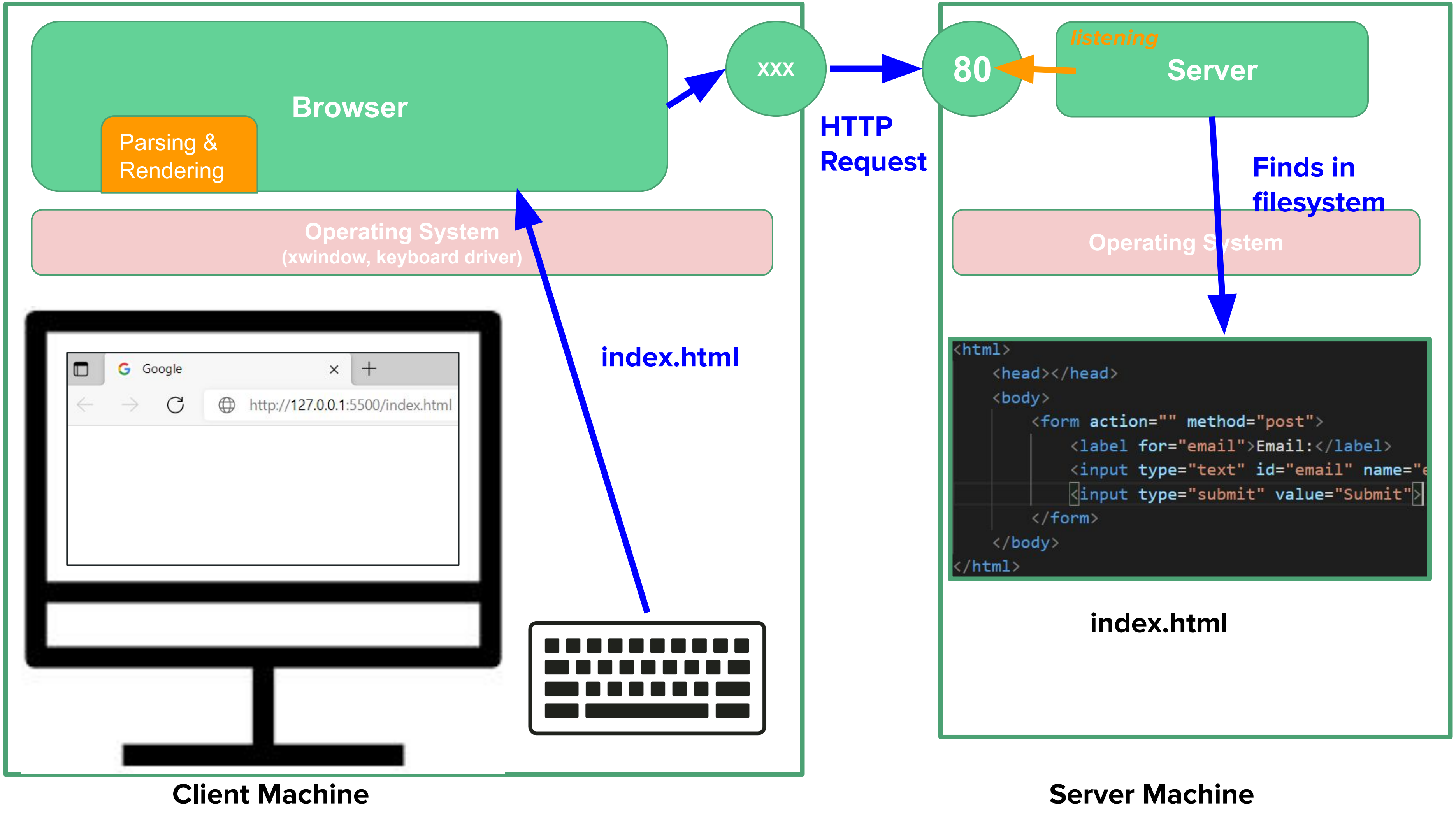
index.html

```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

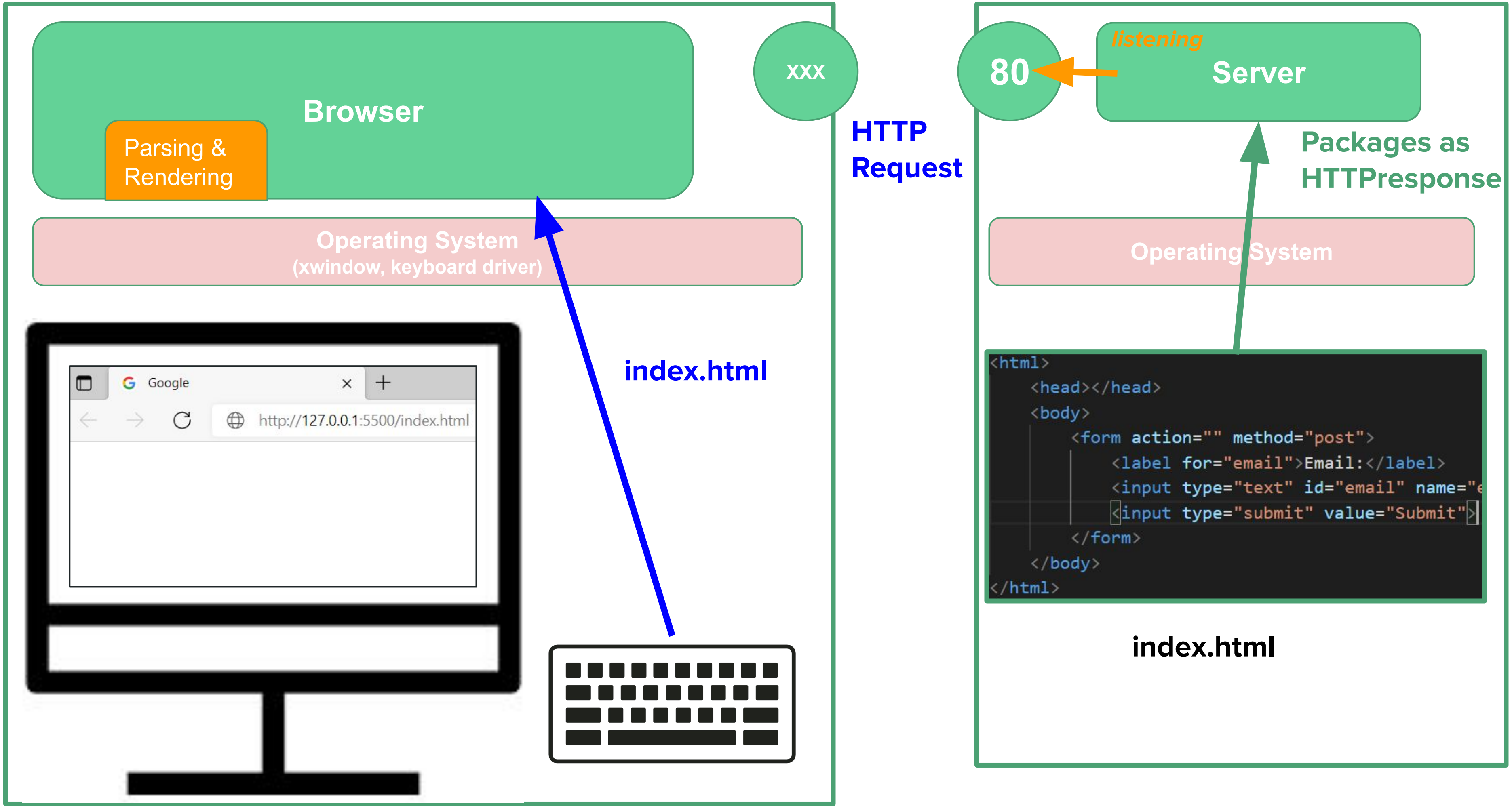
index.html

Client Machine

Server Machine



```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```



Browser

Parsing & Rendering

xxx

Operating System
(xwindow, keyboard driver)

http://127.0.0.1:5500/index.html

index.html

HTTP Request

80

listening

Server

Packages as HTTPresponse

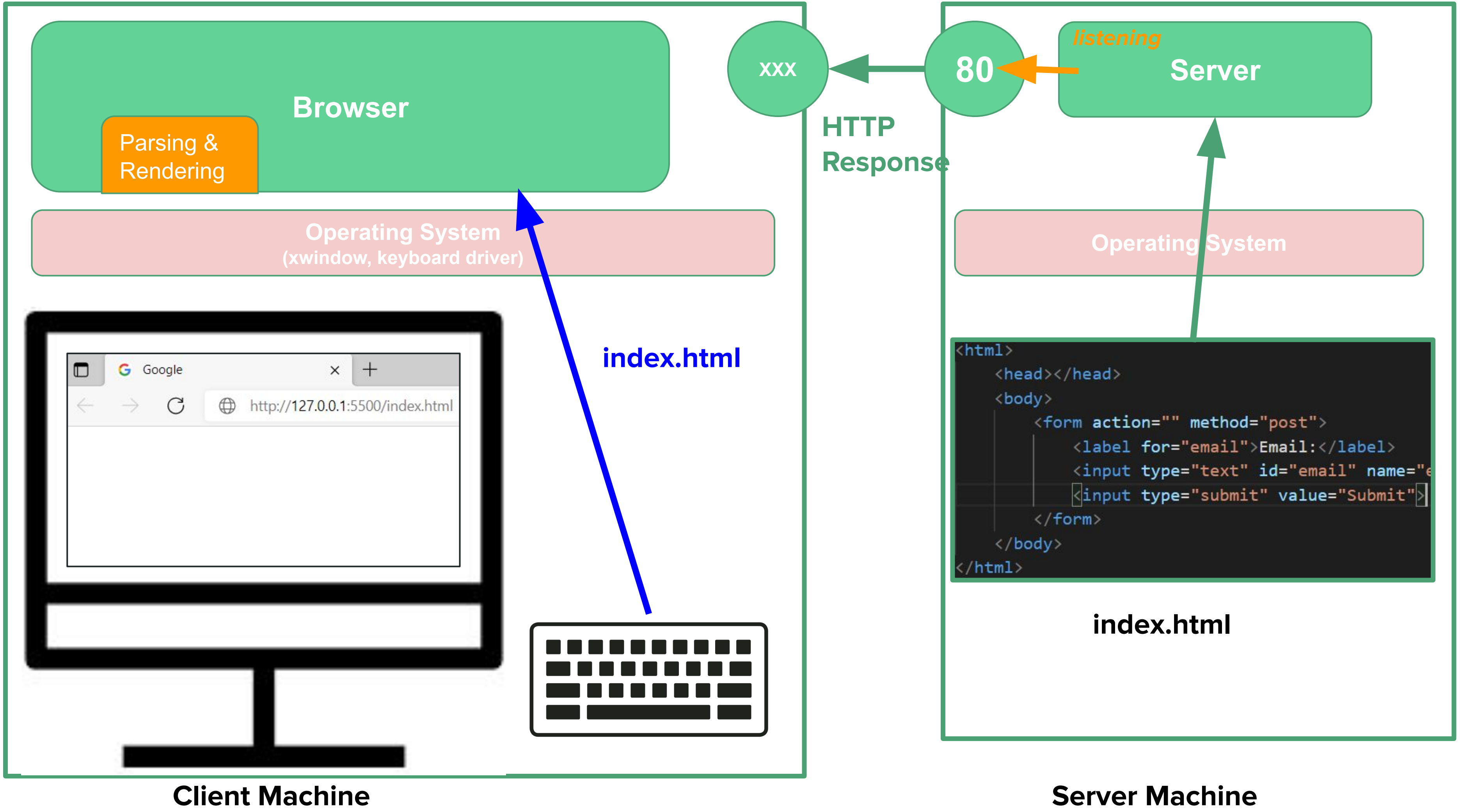
Operating System

```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

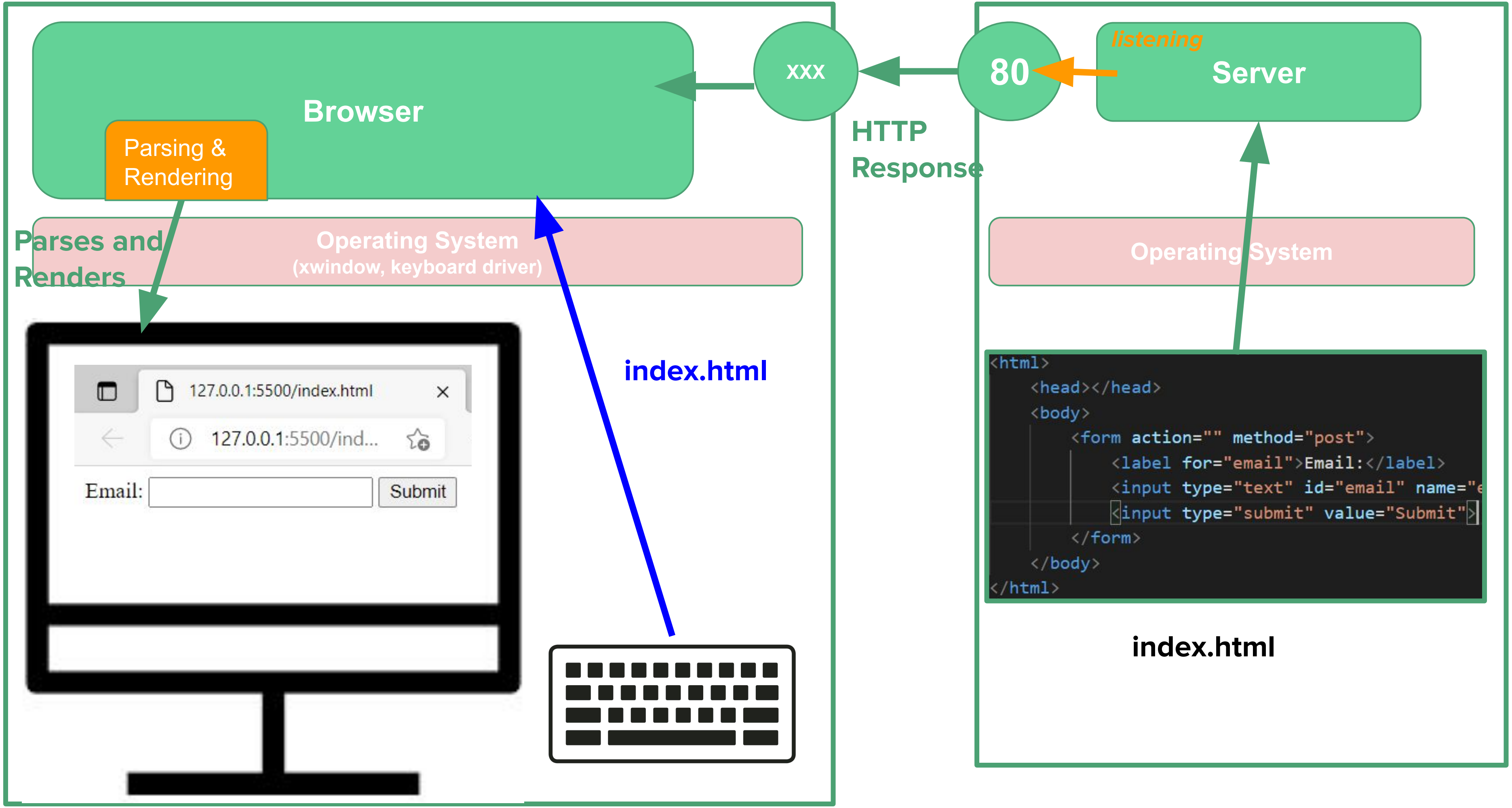
index.html

Client Machine

Server Machine



```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```



Browser

Parsing & Rendering

xxx

HTTP Response

80

listening

Server

Operating System (xwindow, keyboard driver)

Operating System

Parses and Renders

index.html

```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

index.html

Client Machine

Server Machine

Now let's ask ourselves the following question

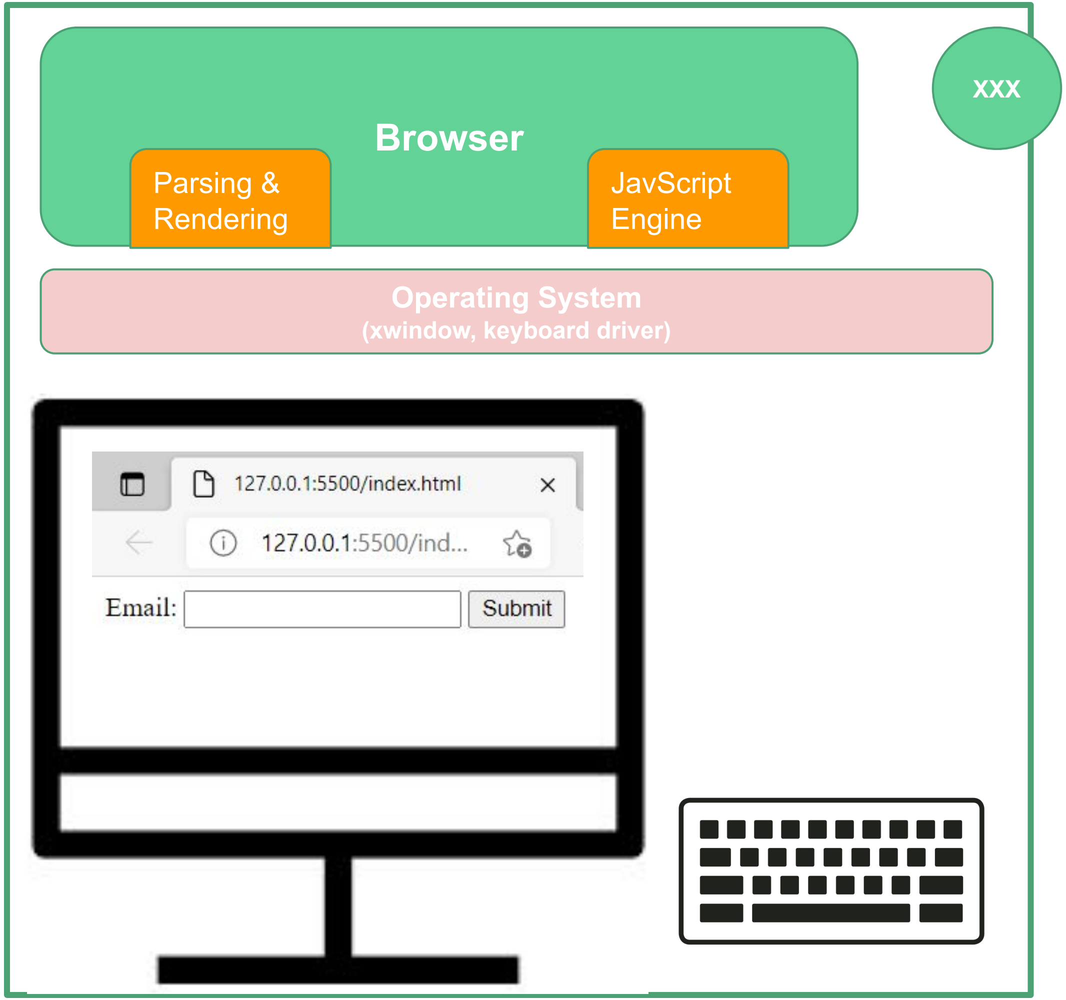
- What happens when an user types in ‘ax123’ and submits it?

  The server (hopefully!) sends back an error

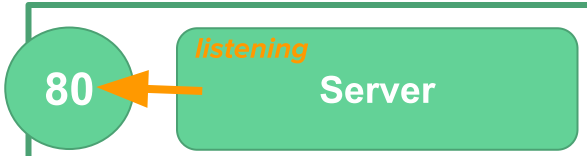
- Is that a productive use of bandwidth (and very limited) server resources?

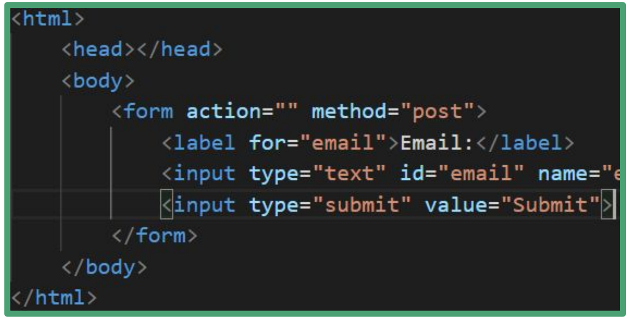
  No

- Assume we have a more complicated form
  - Requirement: someone who checks ‘yes’ for disability fills in a textbox, explaining the type of disability.
  - Is there a way to ensure that the textbox is enabled on when someone selects ‘yes’.



Client Machine





```
<html>
    <head></head>
    <body>
        <form action="" method="post">
            <label for="email">Email:</label>
            <input type="text" id="email" name="e
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

index.html

Server Machine

Lets see a quick demo

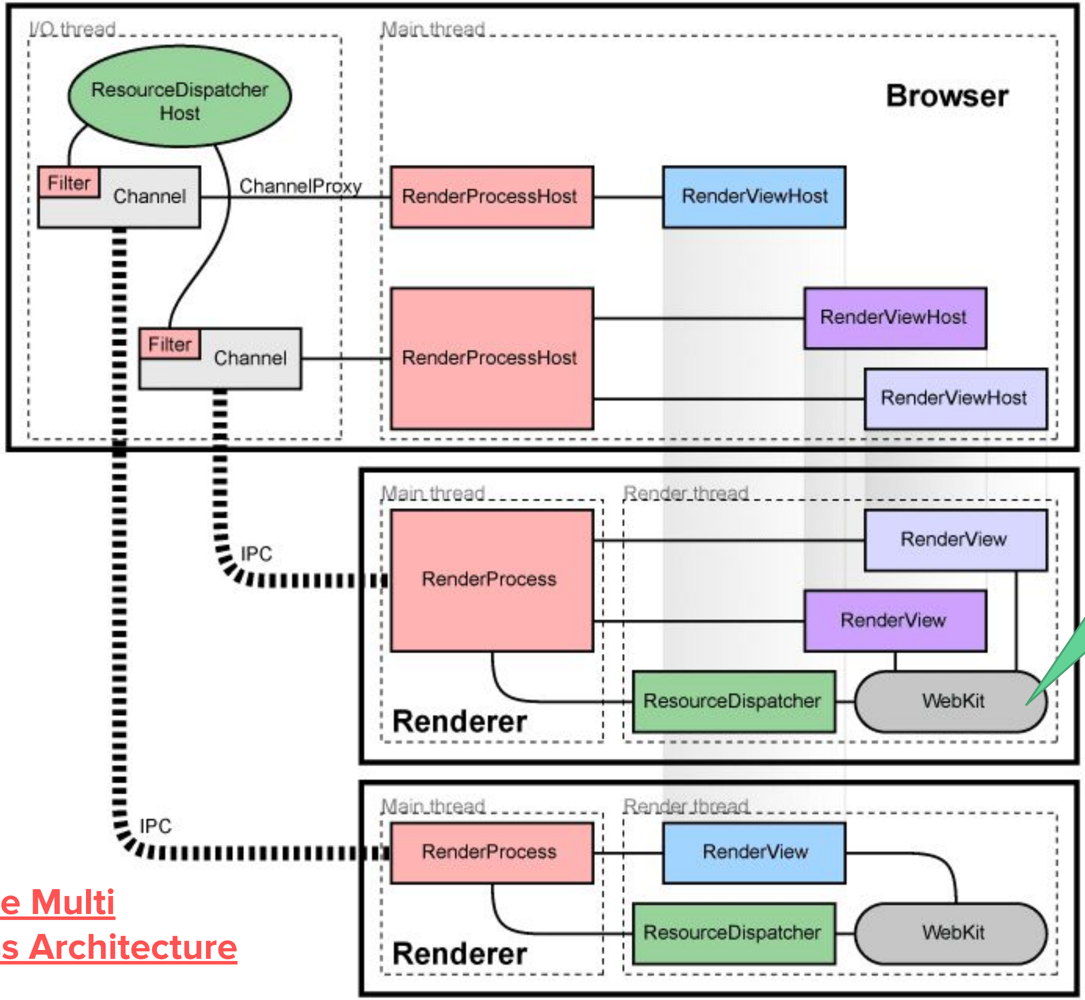
- VSCode Live server
  - HTML + JS for email validation
- Directly on the browser
  - Just Debugging
- Replit.com
  - *Same* HTML + JS for email validation

“Hosted” language

- Never intended to work alone
- Always runs with “friends”
- Hosted in an environment
  - Like in a browser
  - In a C++ shell - Node - explained later
- Relies on “friends” in the hosting environment for some of the features

Popular JavaScript Engines

- Chrome has **V8 Engine**
  - So does Node.js
- Firefox has Spidermonkey
- Safari has JavaScriptCore (also called Nitro)
- Edge has Chakra



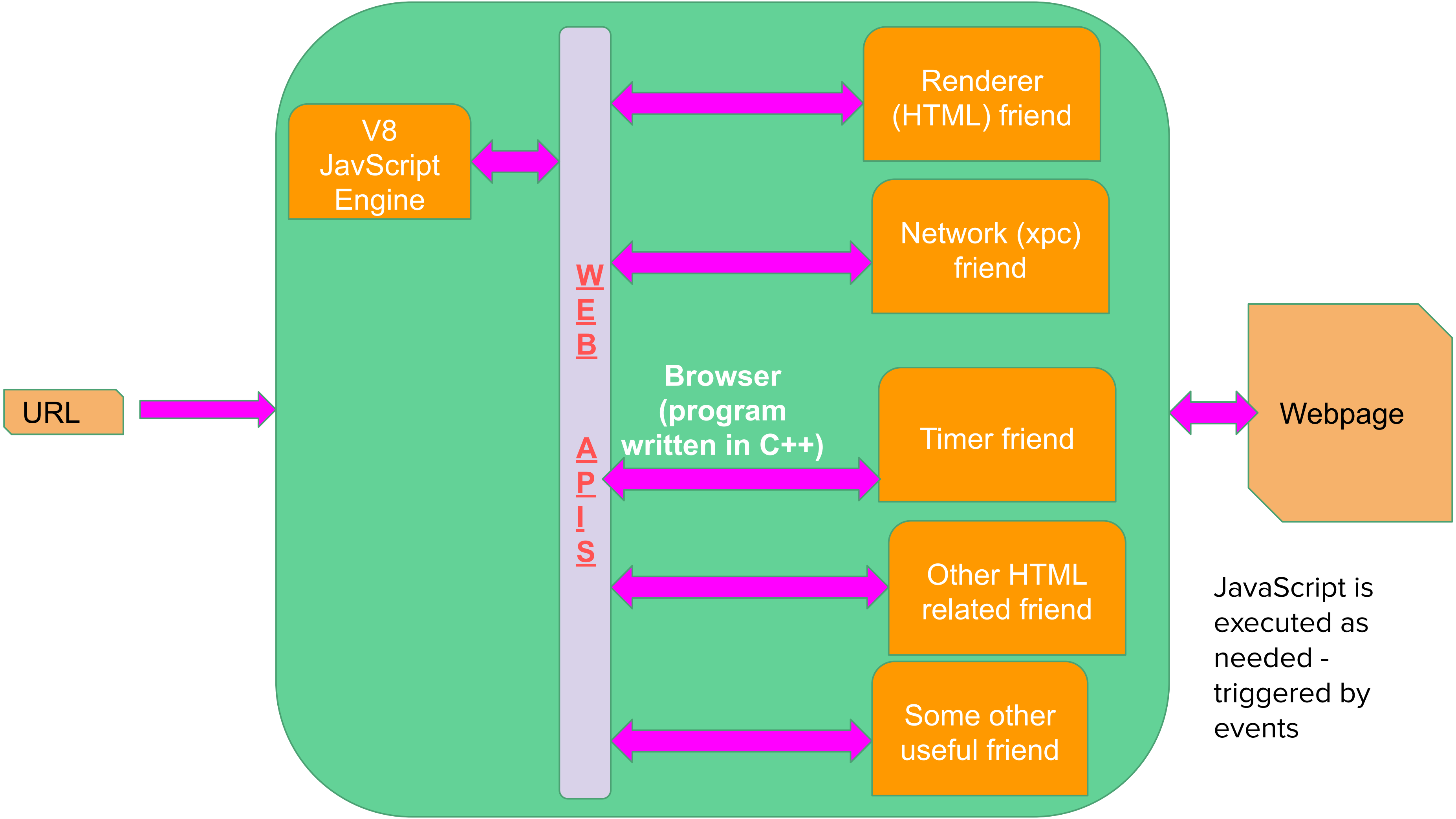
Inside WebKit there is JavaScriptCore - JavaScript Engine V8

Chrome Multi Process Architecture

Caution: *The following slides show a heavily (unapologetically) simplified model of javascript in Browser, so as to help you understand. Use this just as a tool/mental model to understand javascript-in-browser vs javascript-in-Nodejs. DO NOT interpret the modules literally! For example, there is no such thing called ‘friend’*

Javascript is single threaded!

- But its not slow
- But how does it get work done
- It works similar to IIITS students
  - When you can’t do the assignment what do you do?
  - Ask your friend to do it and go to sleep/watch movie
  - Tell him to wake you up when he is done
- The friends are threads (workers) spawned and run by the **hosted** environment



URL

V8 JavScript Engine

WEB APIS

Browser (program written in C++)

Renderer (HTML) friend

Network (xpc) friend

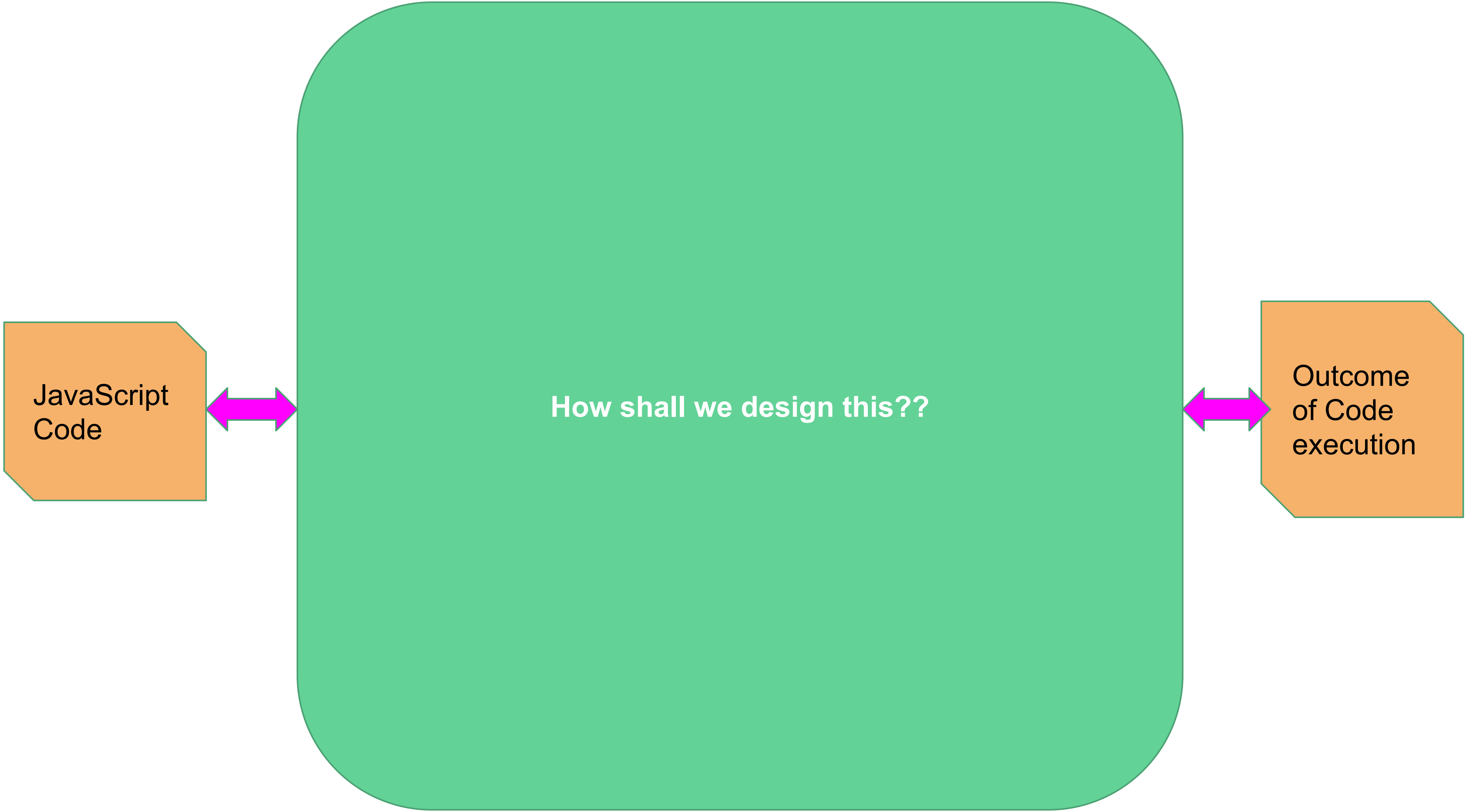
Timer friend

Other HTML related friend

Some other useful friend

Webpage

JavaScript is executed as needed - triggered by events

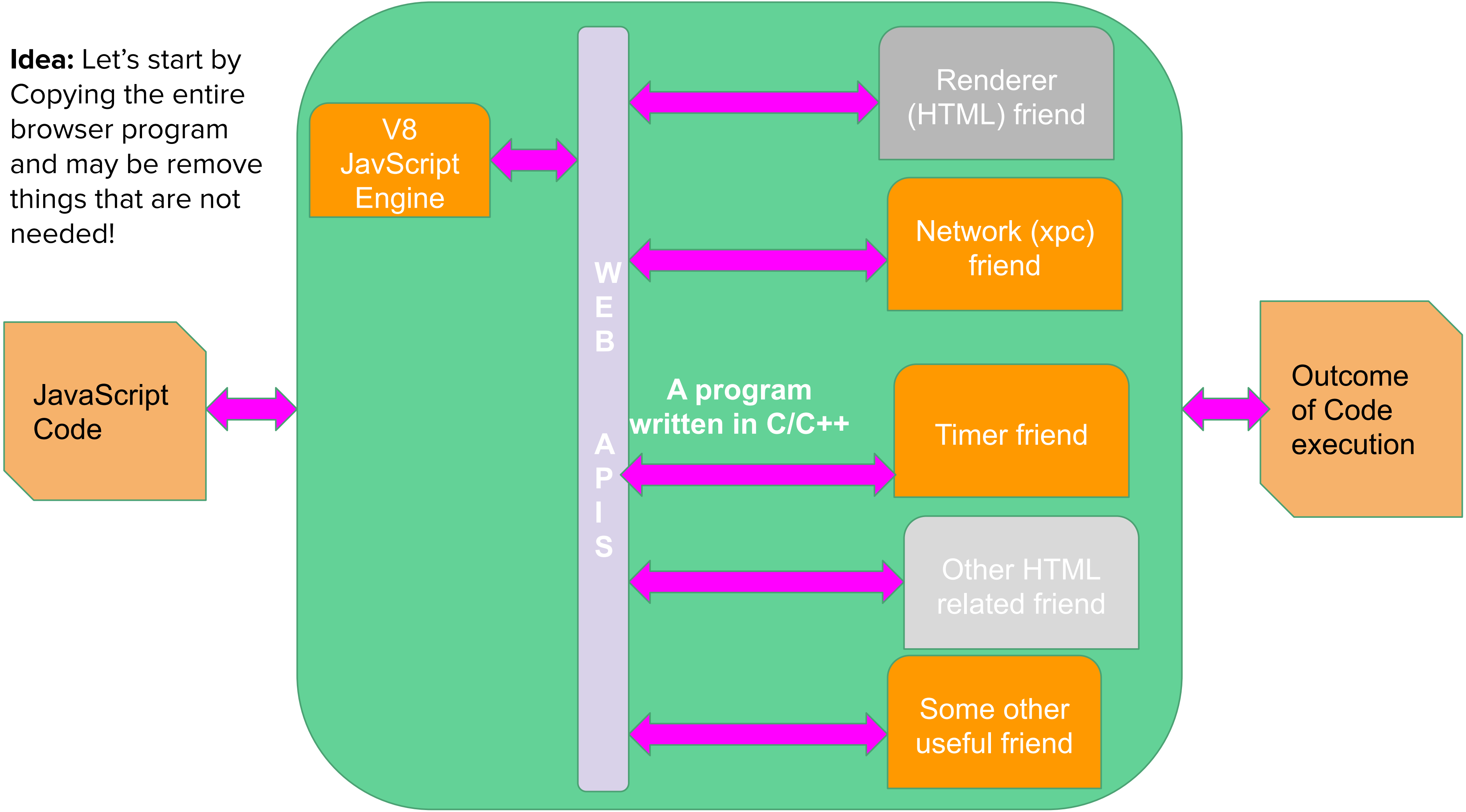


JavaScript Code

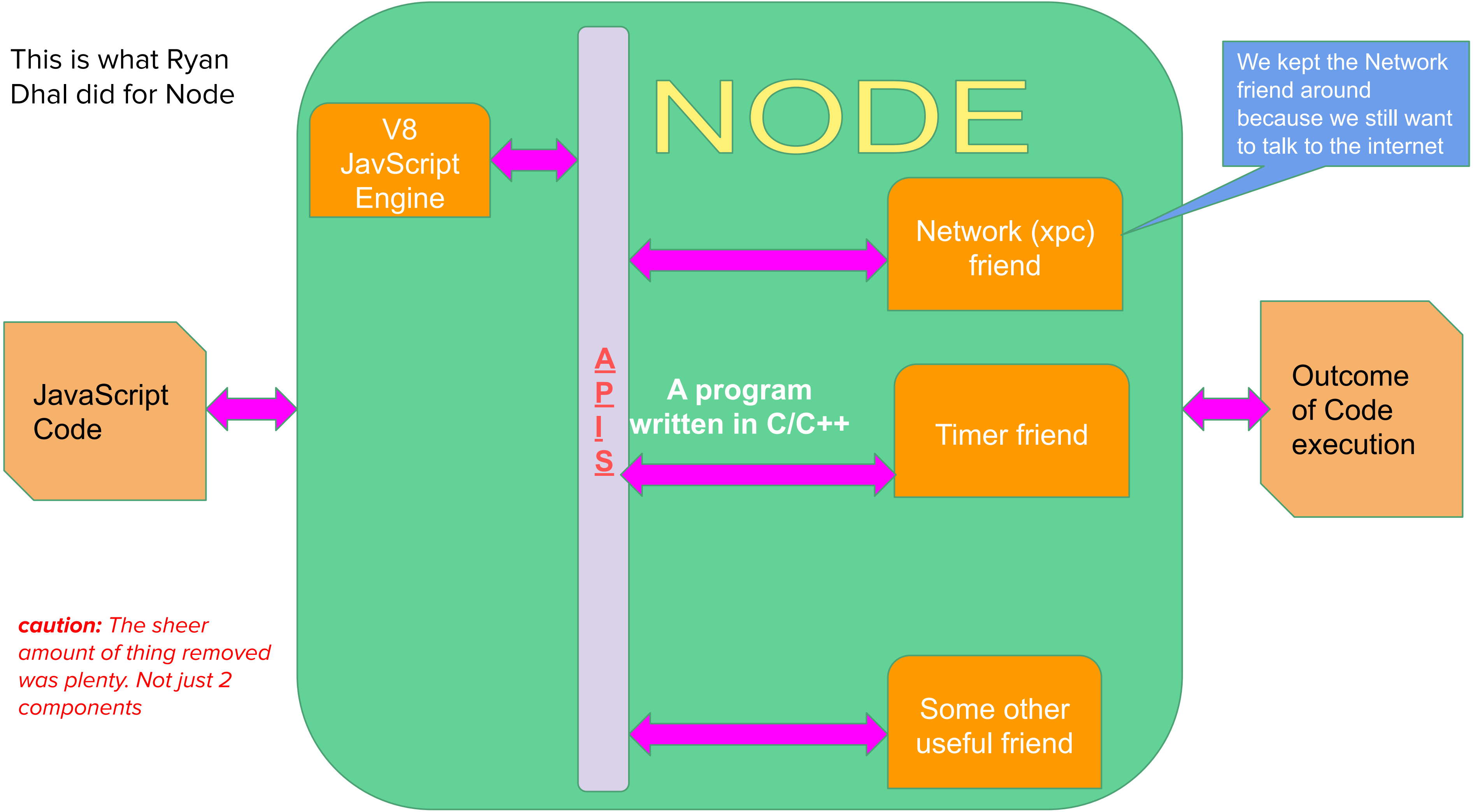
How shall we design this??

Outcome of Code execution

Idea: Let’s start by Copying the entire browser program and may be remove things that are not needed!



This is what Ryan Dhal did for Node


NODE
V8 JavScript Engine
A P I S
A program written in C/C++
Network (xpc) friend
Timer friend
Some other useful friend
JavaScript Code
Outcome of Code execution
We kept the Network friend around because we still want to talk to the internet

caution: *The sheer amount of thing removed was plenty. Not just 2 components*

Lets see a quick demo

- Same simple email validation in Node
  - VSCode
  - Replit

Simple HTTP Server:

```
var http = require("http");

http.createServer(function (req, res) {
  res.sendHeader(200,
    {"Content-Type": "text/plain"});
  res.sendBody("Hello\r\n");
  res.sendBody("World\r\n");
  res.finish();
}).listen(8000);
```

Node - Run JavaScript locally

- It can talk to the local file system through OS
- It can listen to a port for requests
- Hence you can use it to create Servers
    - Voilà now you can write javascript for server side operations
- Unlike the browser you can even increase the number of worker threads in the worker pools
    - Node (libuv) starts with 4 workers in the pool
    - You can also implement your own pools

Summary

- Created by Brendan Eich for Netscape browser (1995ish)
    - In 10 days
    - Started as ‘Scheme in a browser’
- Hosted language
    - Needs friends
    - Browser brings in all the friends
- Ryan Dhal said
    - Don’t worry JS! No need for a browser!
    - I’ll give you all the friends you need! Let’s C
    - Node (2009)

Note on Project

Recap on project for FSD1! (FFSD)

- Multi Page
- Dynamic
- Web Application
- With DB integration
    - For persistence

Stack

- HTML, CSS and/or CSS Frameworks
- Node + Express (useful middlewares)
- DBMS (SQL or NoSQL)

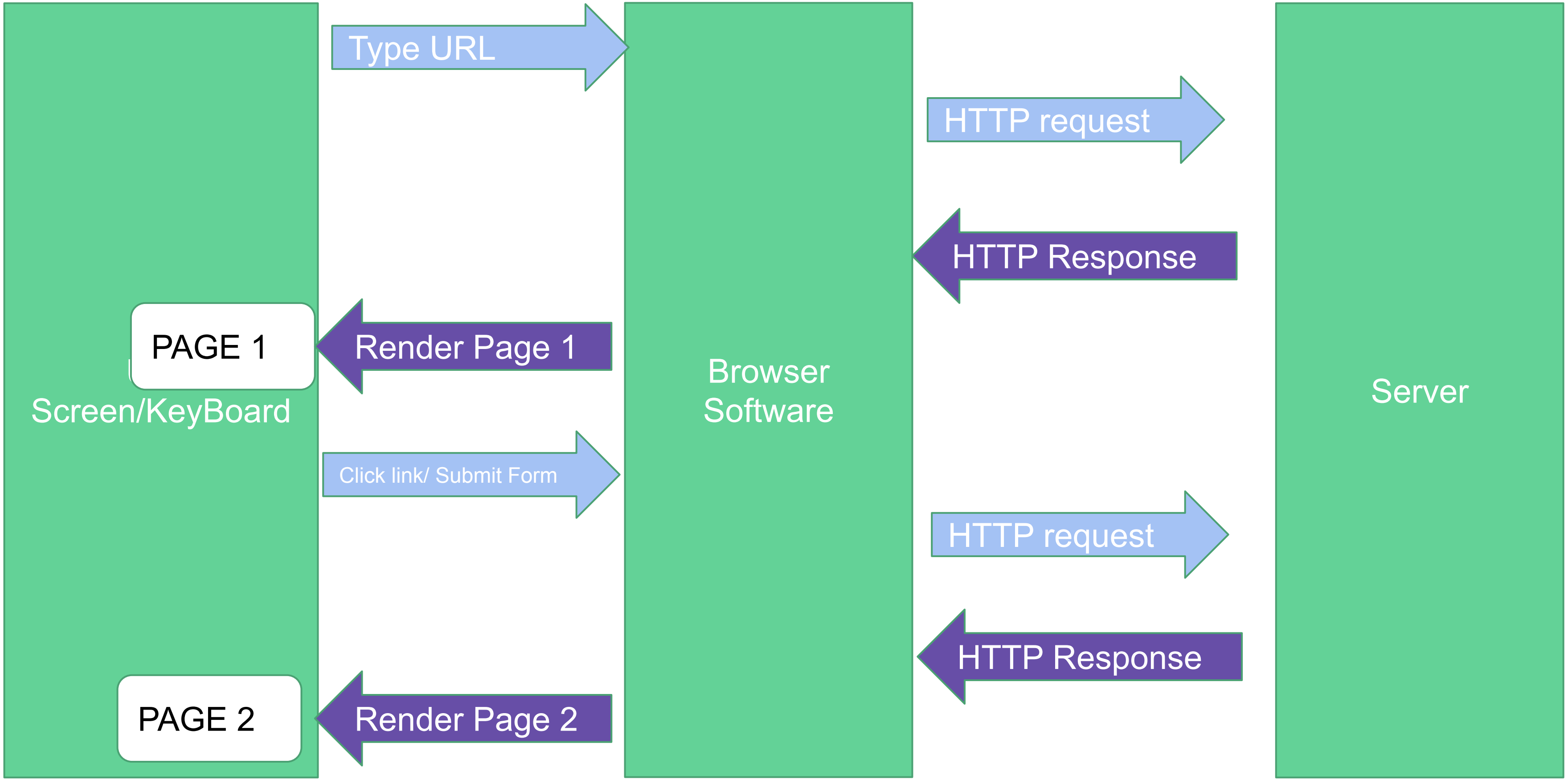
Note on project for FSD1! (FFSD)

- **Single Page**
- Dynamic
- Web Application
- With Mock Endpoint integration

Stack

- HTML, CSS and/or CSS Frameworks
- React, Redux and Node
- JSONserver (or similar for mock end point)

MULTI PAGE APPLICATION



Generic Types of Web Applications

- E-Commerce application
  - Nykaa, floweraura, etc
- E-Commerce MarketPlace
  - Ebay, Amazon, etc
- Social Network Applications (& Layers)
  - Facebook, Linkedin, etc
- Other

E-Commerce application (Type-1)

- Goods :- flowers, tires, cars
- Services:- AC repair, Car wash
- Hybrid :- Food orders (time dependent good/service)
- Content:- News, Courses, Legal documents, Training services
- Payment could be on-spot, on-delivery or recurring (subscription)

E-Commerce Marketplace application (Type-2)

- Similar to E-commerce
- But provides a platform for two types of users
  - Buyers
  - Sellers
- Sellers could be
  - Corporate
  - individual (example: book lending site)

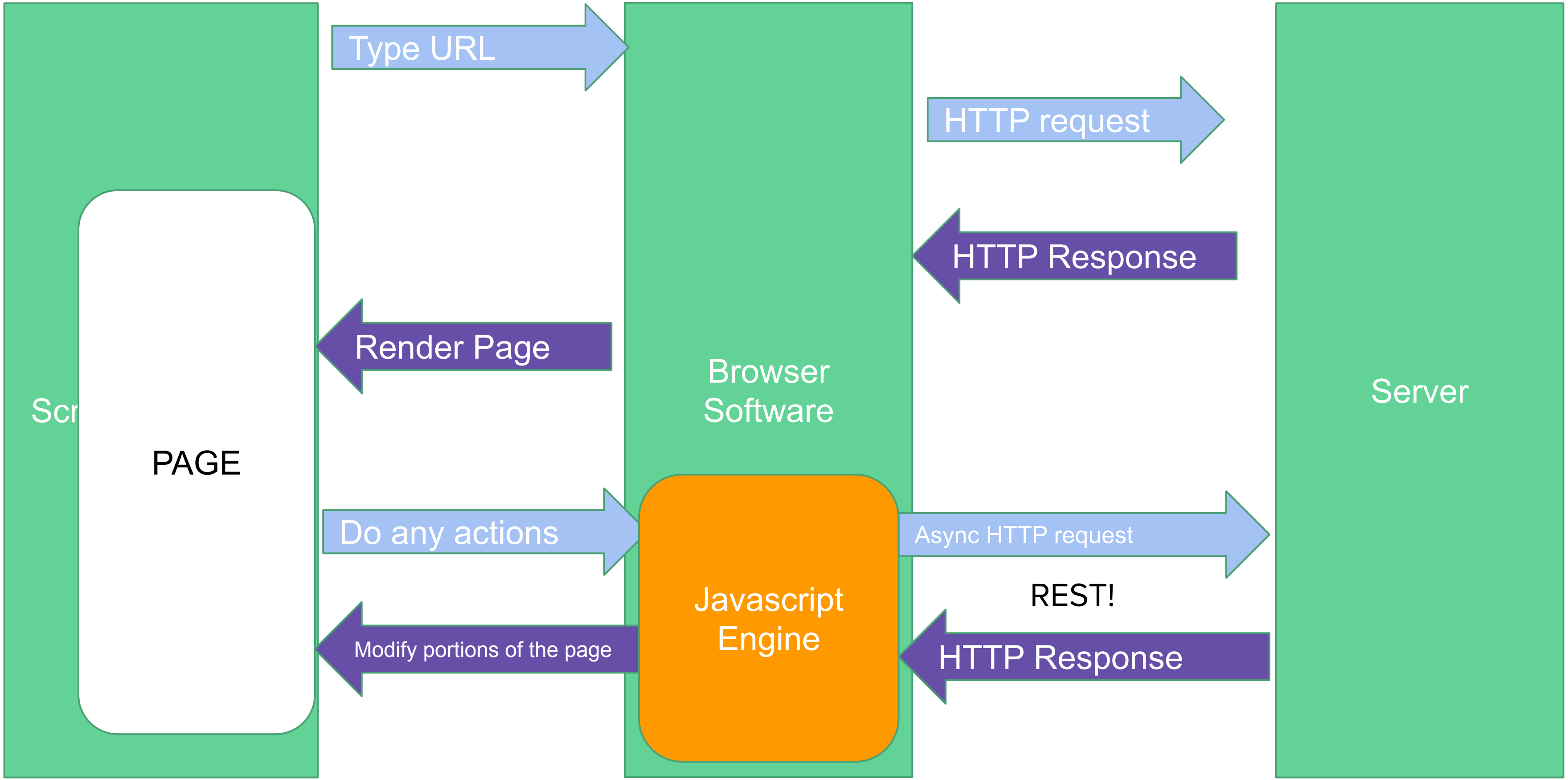
Social Networking Application (Type-3)

- Connects people
- Requires a theme
  - Professional (linkedin)
  - Personal (facebook, tinder)
- Provide auxiliary services
  - Example: linkedin-job search

Other (Type-4)

- Free-Commerce
  - Similar to E-commerce content sites, but provide it for free
  - Make money through other means
    - Ad revenue
    - Sponsors
    - Collecting and selling User behavior data
  - Example: RottenTomatoes, YouTube, etc
- Aggregators
  - News360 (news aggregator), edealinfo (deals aggregator)
- Search engines (google, duckduckgo, etc)
- Wikis
- OTT platforms
- Many Many more.....

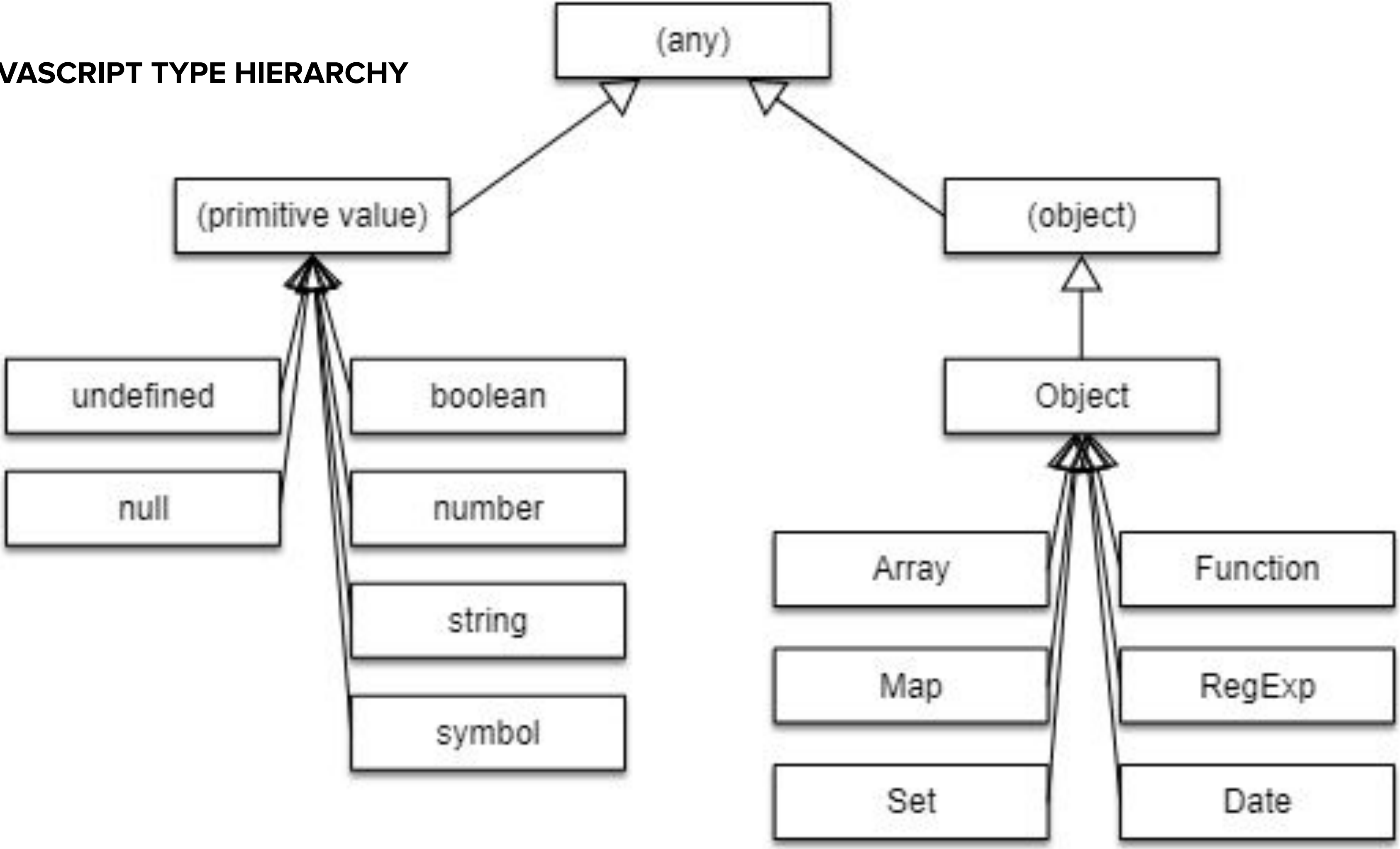
SINGLE PAGE APPLICATION



JavaScript

The Language - Basics

JAVASCRIPT TYPE HIERARCHY



Our Checklist

- ❑ Let & Const
- ❑ Tour of types
    - ❑ Primitives
        - ❑ Boolean
        - ❑ String
        - ❑ Number
    - ❑ Reference types
        - ❑ Object
        - ❑ Arrays
        - ❑ Functions

JavaScript

Event Loop, Task Queue & Call Stack

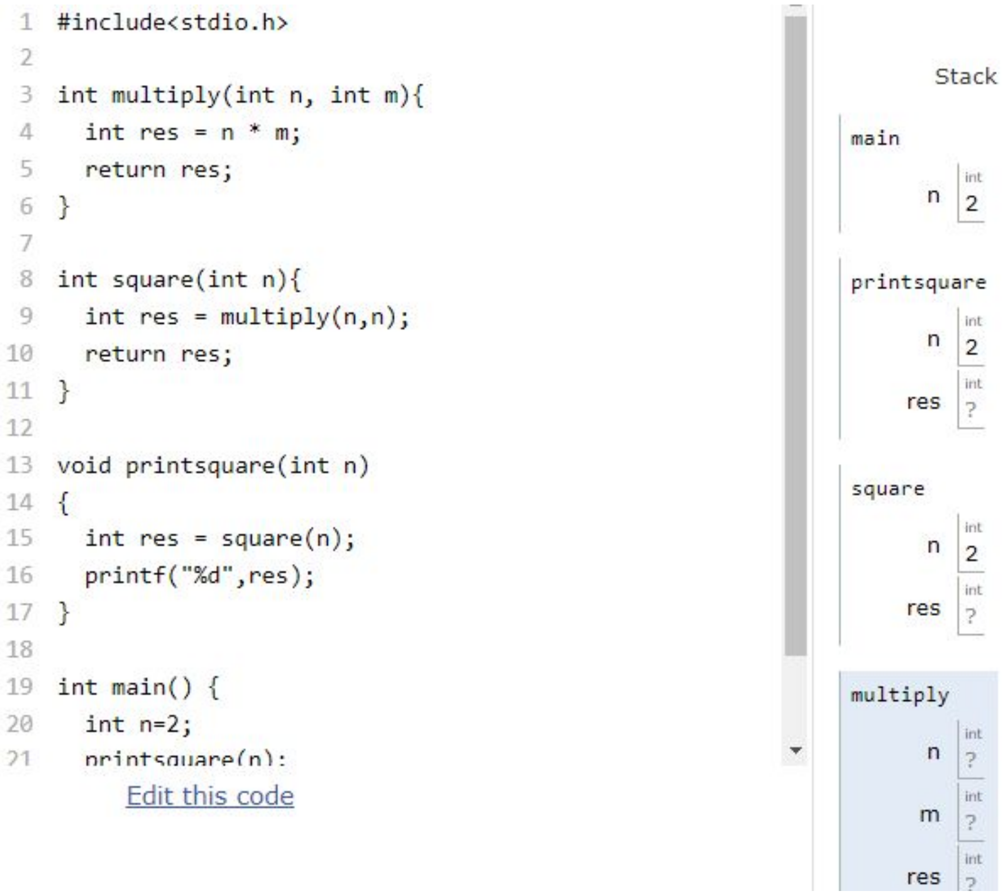
Buzz words!

- Single threaded
- Non-blocking
- Asynchronous
- Concurrent
- Event-driven
- Dynamic
- Loosely-typed

Javascript & Friends

- We already saw JS doesn’t work alone and only works with friends (provided by the Hosted environment browser/node)
- But how ?
- How does it communicate with the friends?
  - Web APIs true! But what else
- How do they communicate back?
- What’s under the hood?

Let’s start with a recap of Call Stack (for C)



```
#include<stdio.h>

int multiply(int n, int m){
  int res = n * m;
  return res;
}

int square(int n){
  int res = multiply(n,n);
  return res;
}

void printsquare(int n)
{
  int res = square(n);
  printf("%d",res);
}

int main() {
  int n=2;
  printsquare(n);
```

Edit this code

Stack

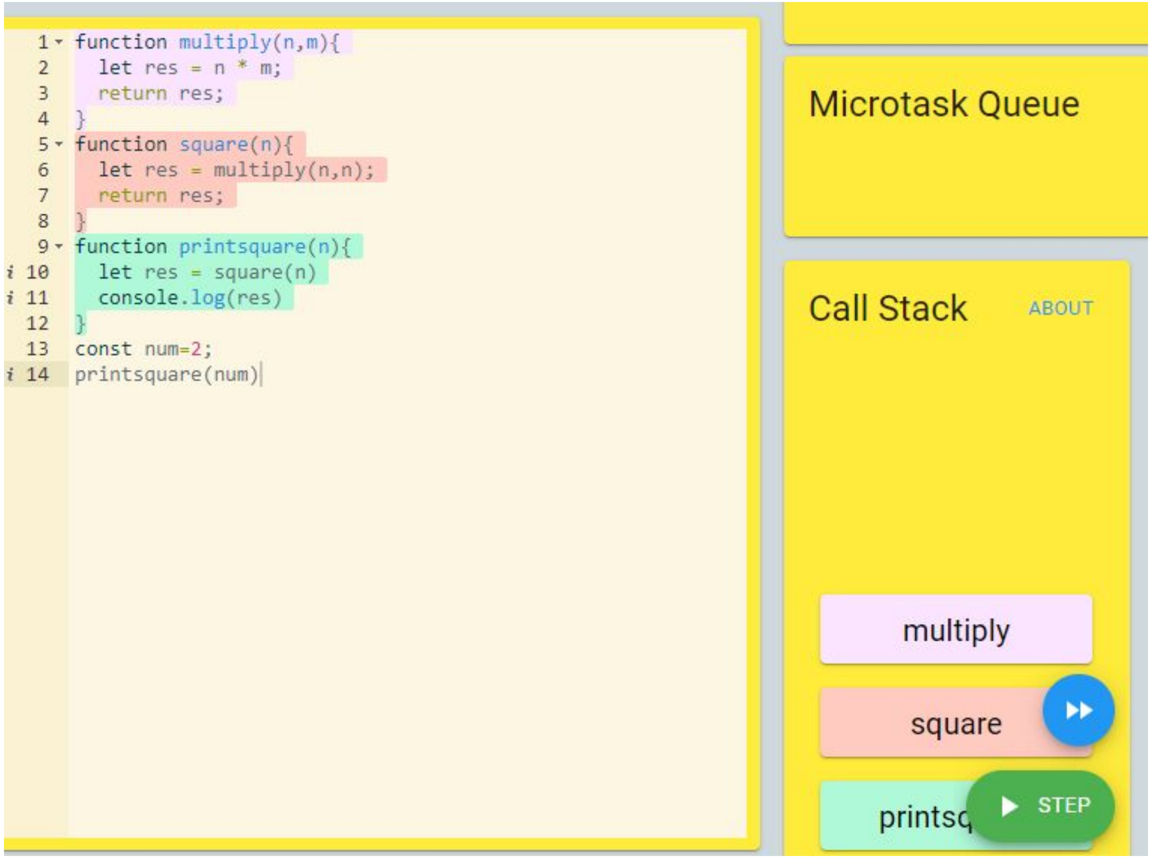
main
n int 2

printsquare
n int 2
res int ?

square
n int 2
res int ?

multiply
n int ?
m int ?
res int ?

Let’s start with a recap of Call Stack
(Not so different for Javascript)

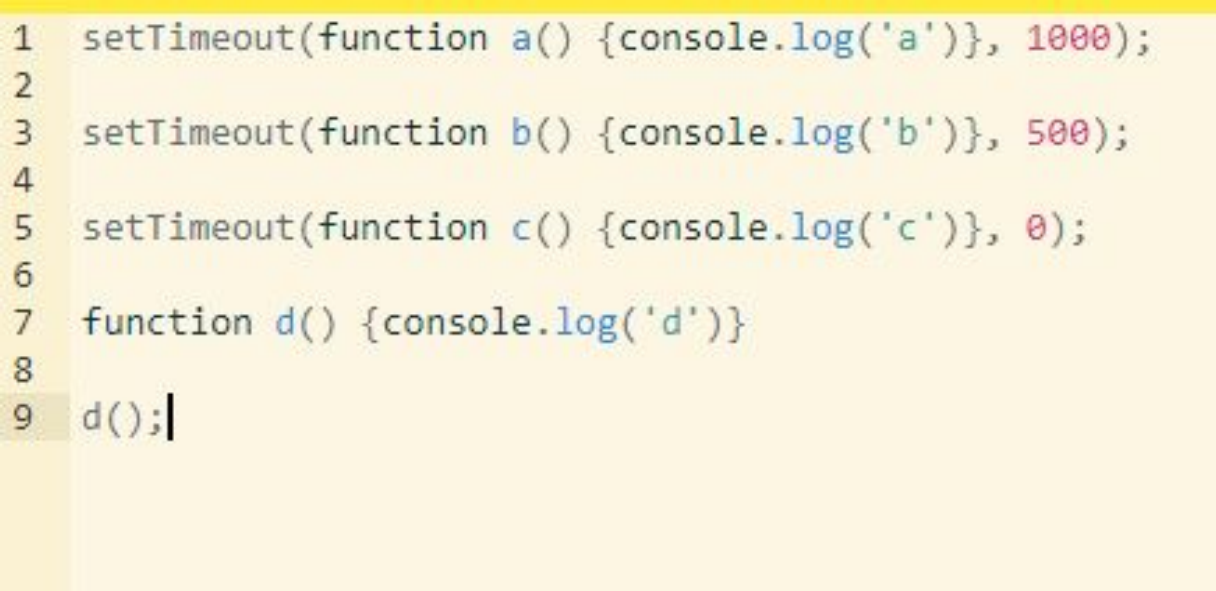


```
function multiply(n,m){
  let res = n * m;
  return res;
}
function square(n){
  let res = multiply(n,n);
  return res;
}
function printsquare(n){
  let res = square(n)
  console.log(res)
}
const num=2;
printsquare(num)
```

What does single threaded means?

- At any given time there can be ONLY ONE STACK
- For multithreaded languages, each thread gets its very own stack
  - ‘Main’ is the first thread
- Javascript uses the ‘event loop’ and the ‘task queue(s)’ to achieve the communication with the friends
- Lets see a simple example

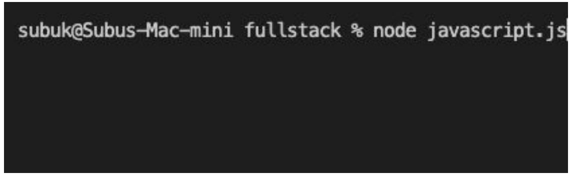
Lets see a simple example



```
setTimeout(function a() {console.log('a')}, 1000);

setTimeout(function b() {console.log('b')}, 500);

setTimeout(function c() {console.log('c')}, 0);

function d() {console.log('d')}

d();
```

Call Stack

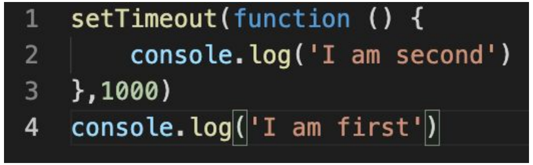
Console



```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

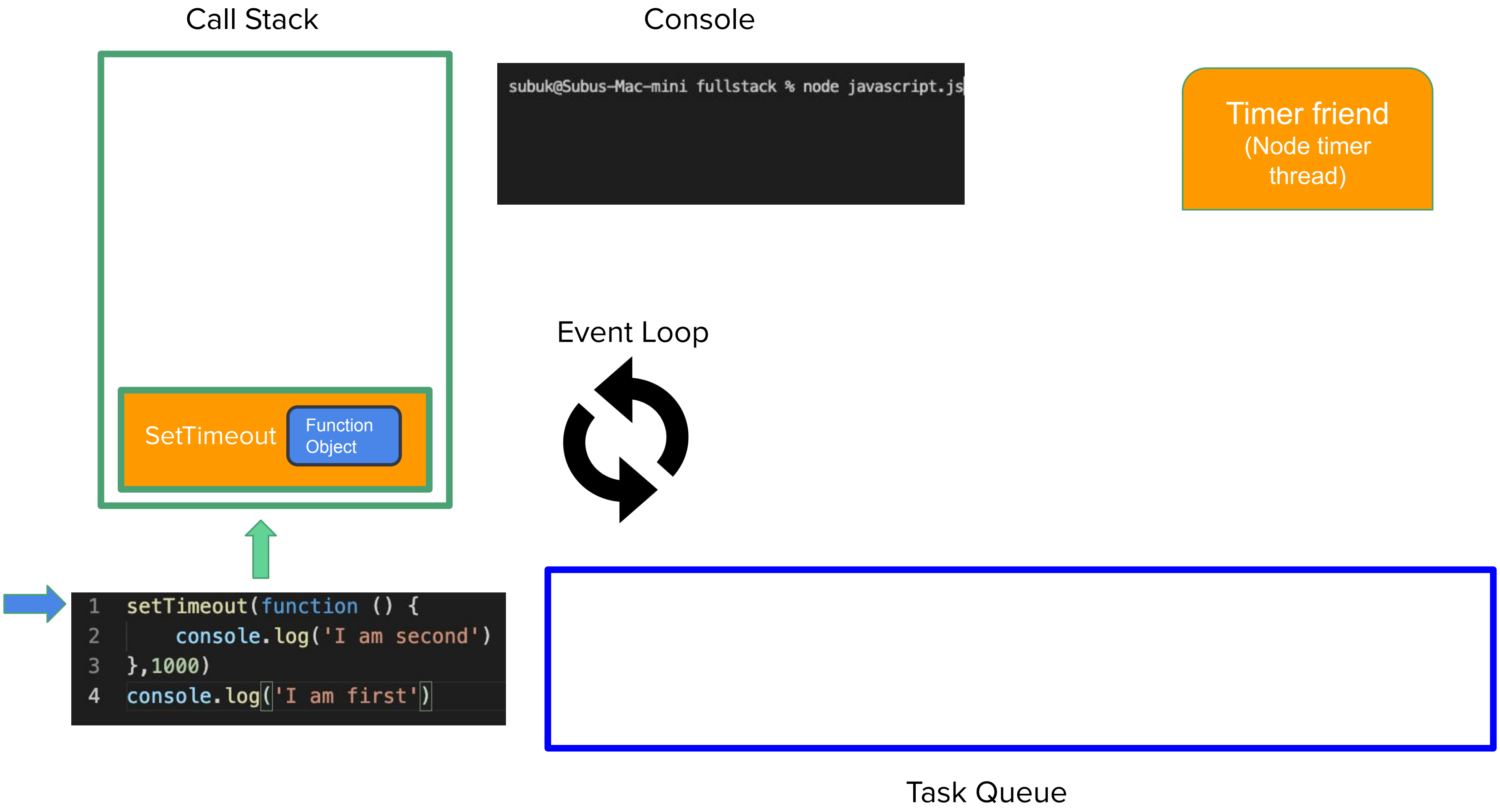
Timer friend
(Node timer
thread)

Event Loop



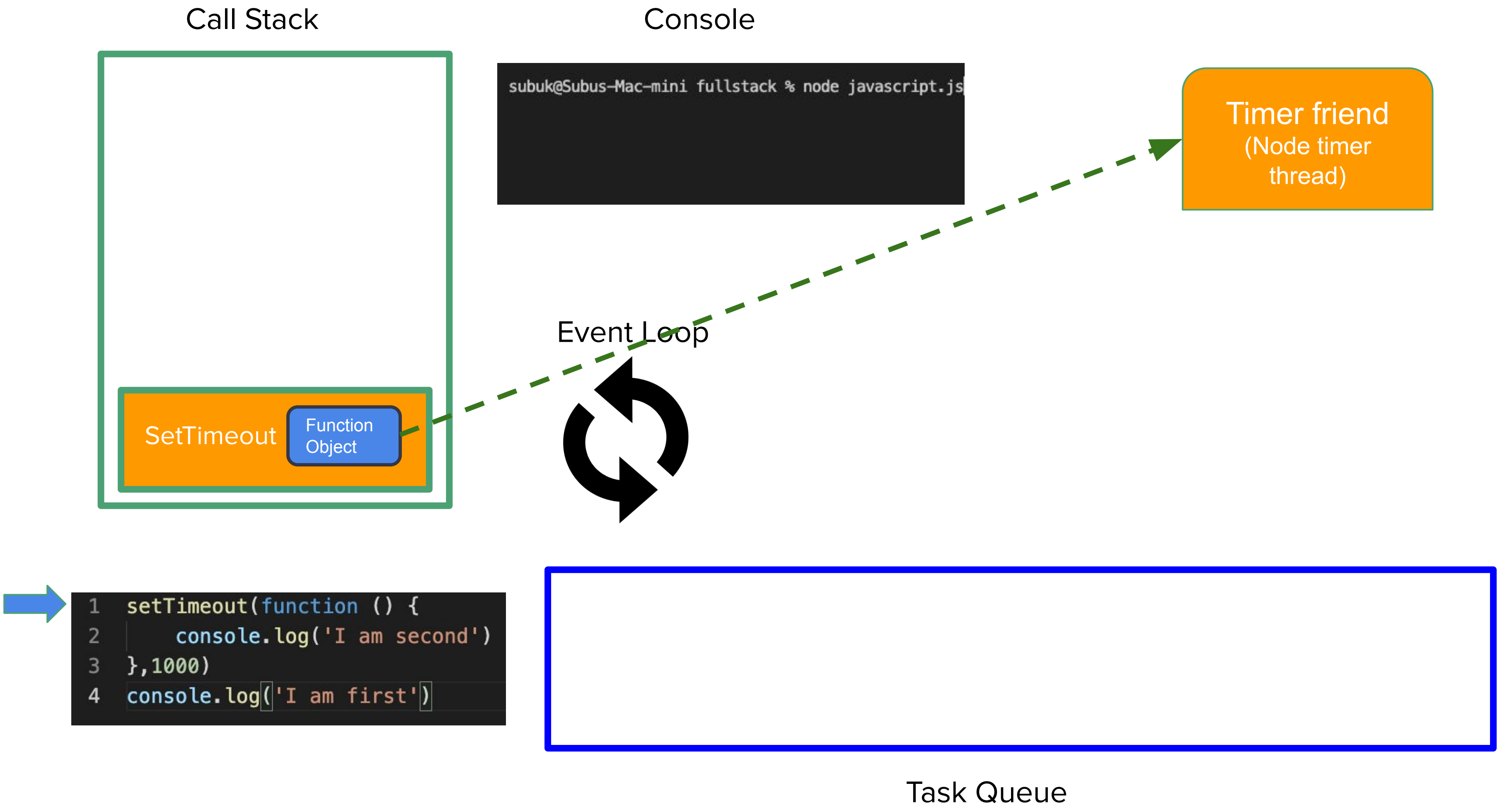
```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue



```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```



Call Stack

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

Timer friend
(Node timer
thread)

Event Loop

SetTimeout

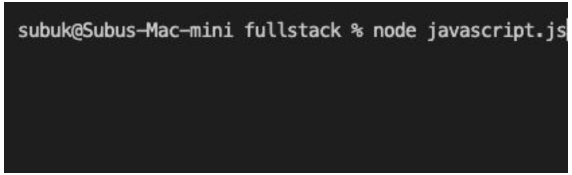
Function
Object

```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

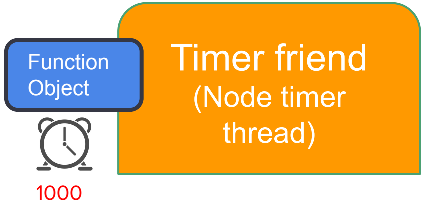
Task Queue

Call Stack

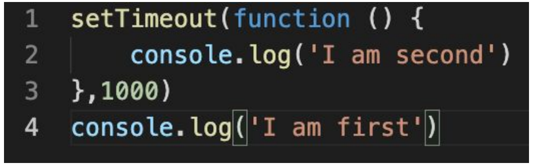
Console



```
subuk@Subus-Mac-mini fullstack % node javascript.js
```



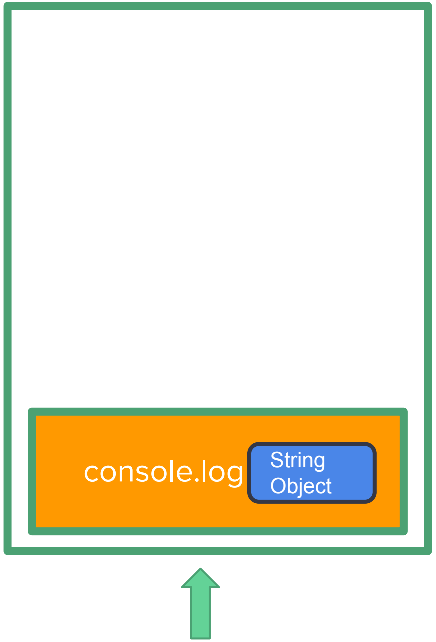
Event Loop



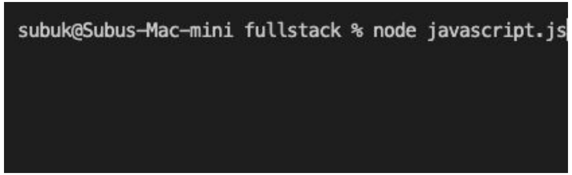
```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

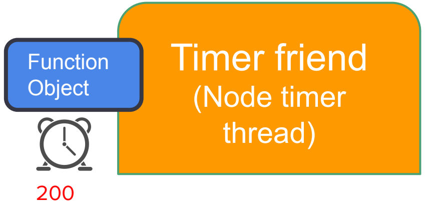
Call Stack



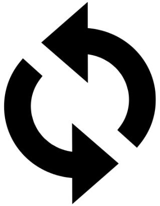
Console

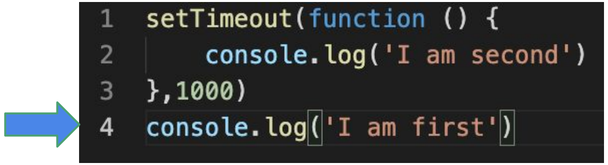


```
subuk@Subus-Mac-mini fullstack % node javascript.js
```



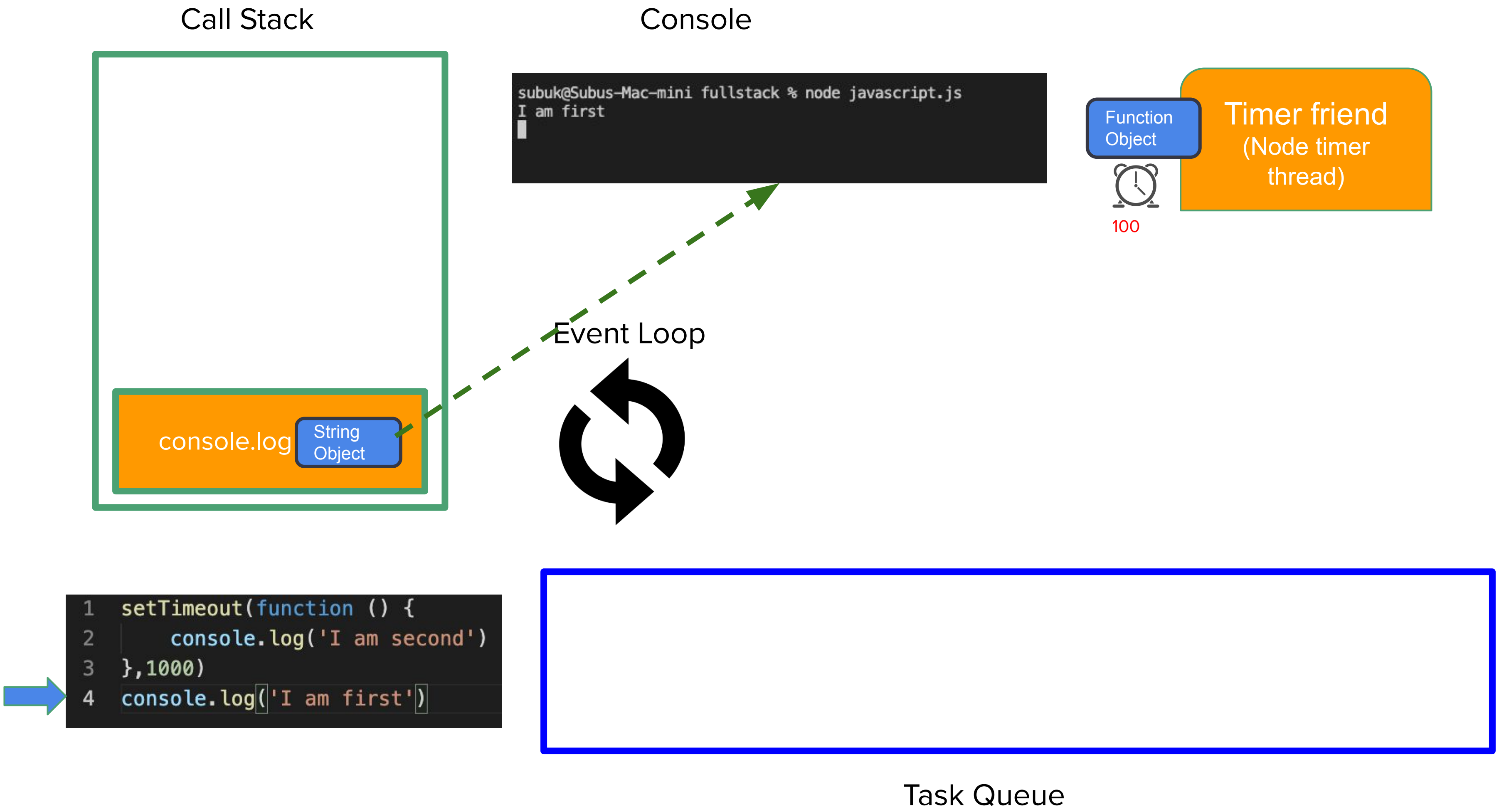
Event Loop





```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

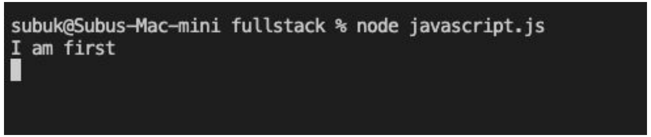


```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
█
```

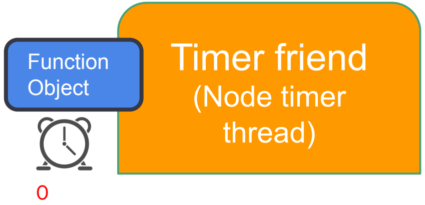
```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Call Stack

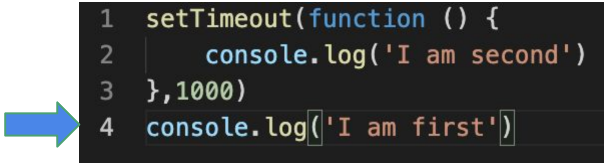
Console



```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

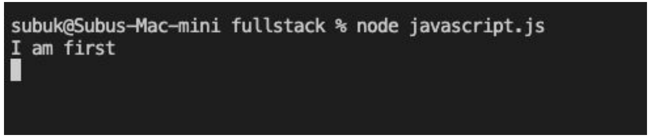


Event Loop

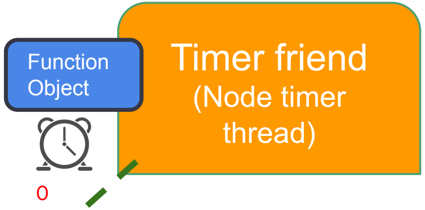


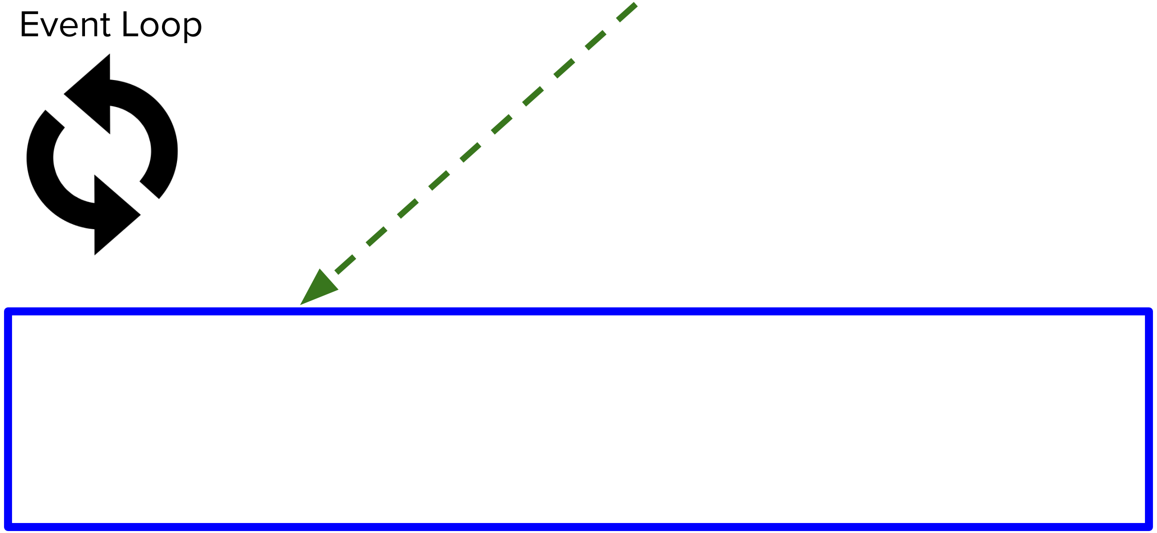
```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

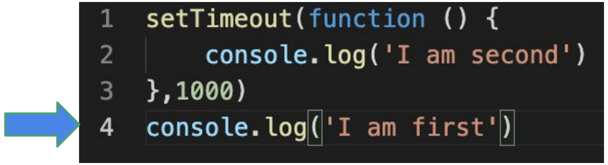
Task Queue



```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
█
```



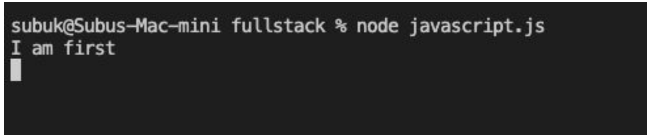




```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Call Stack

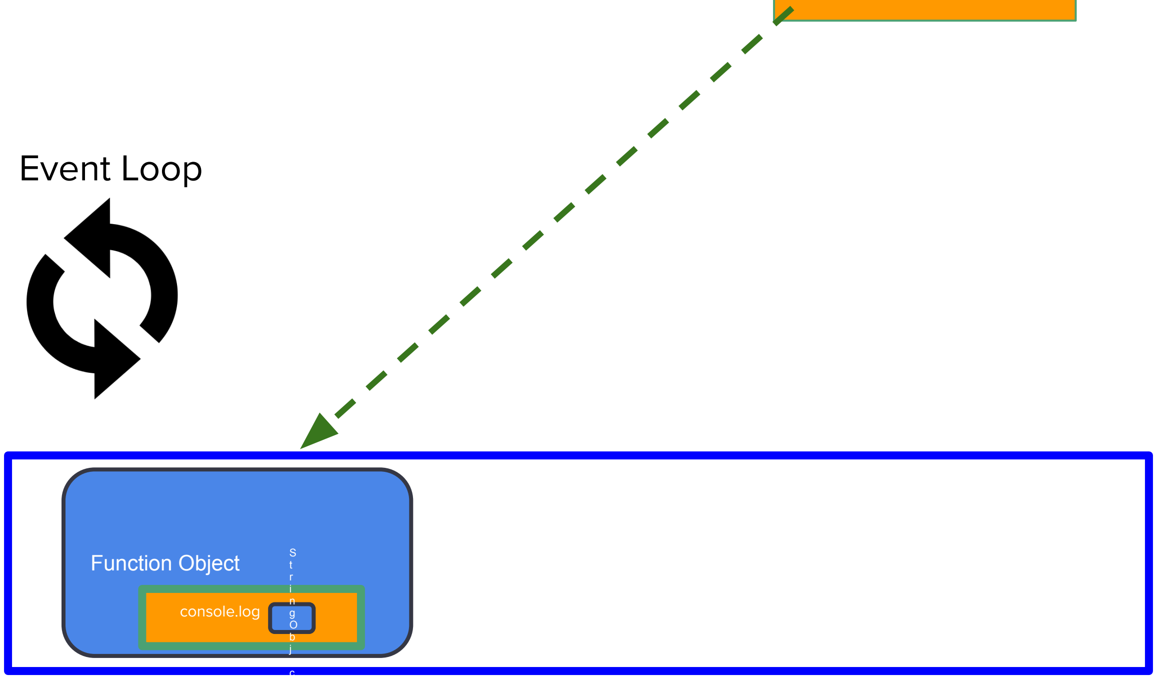
Console



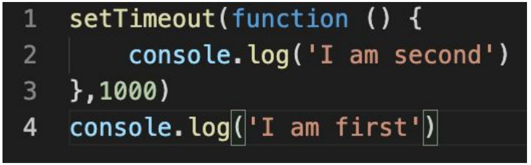
```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Timer friend
(Node timer thread)

Event Loop



Task Queue

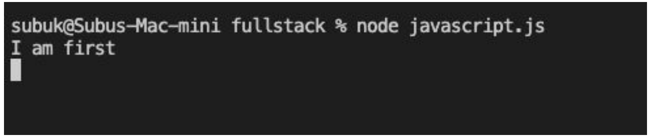


```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Call Stack

Anonymous function

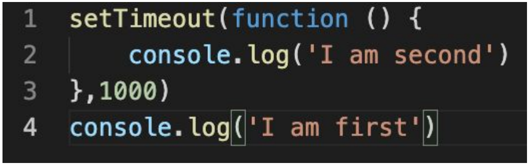
Console



```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
█
```

Timer friend
(Node timer thread)

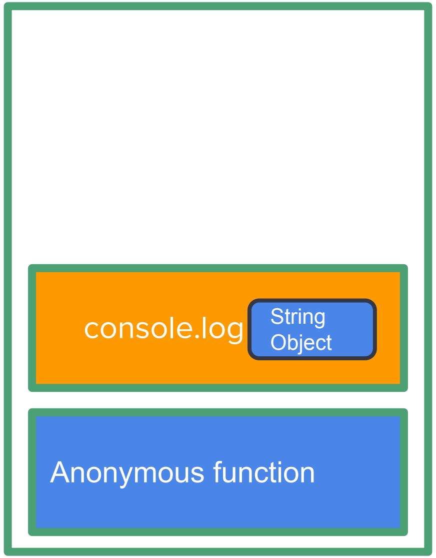
Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

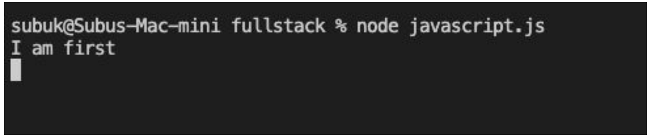


console.log

String Object

Anonymous function

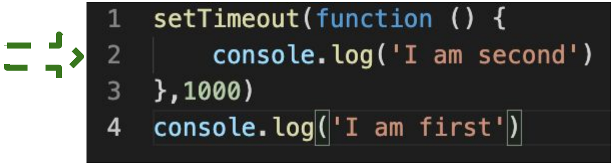
Console



```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Timer friend
(Node timer thread)

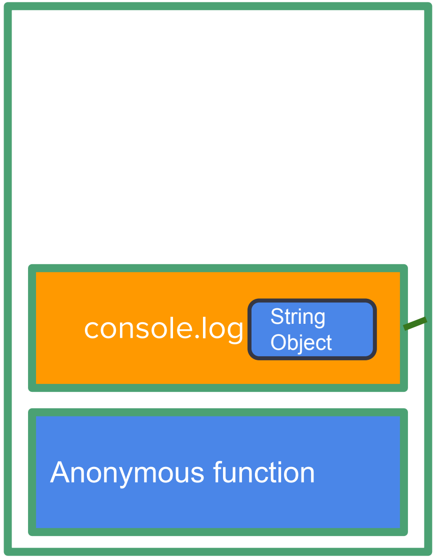
Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

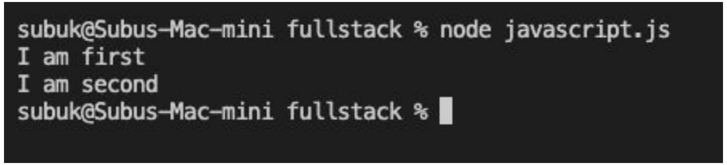
Call Stack



console.log String Object

Anonymous function

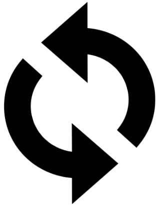
Console

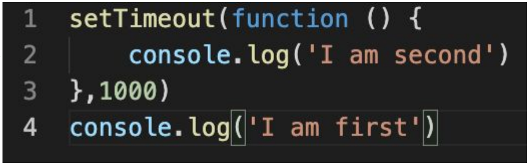


```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
I am second
subuk@Subus-Mac-mini fullstack % 
```

Timer friend
(Node timer thread)

Event Loop



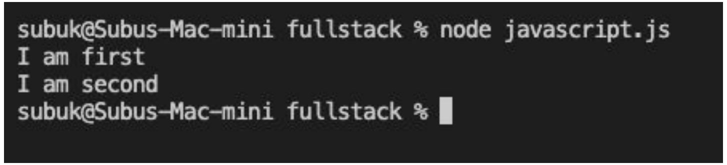


```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

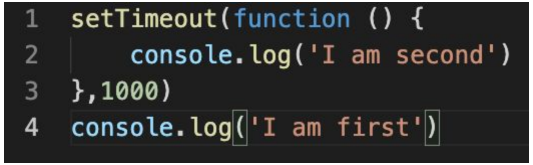
Console



```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
I am second
subuk@Subus-Mac-mini fullstack % 
```

Timer friend
(Node timer thread)

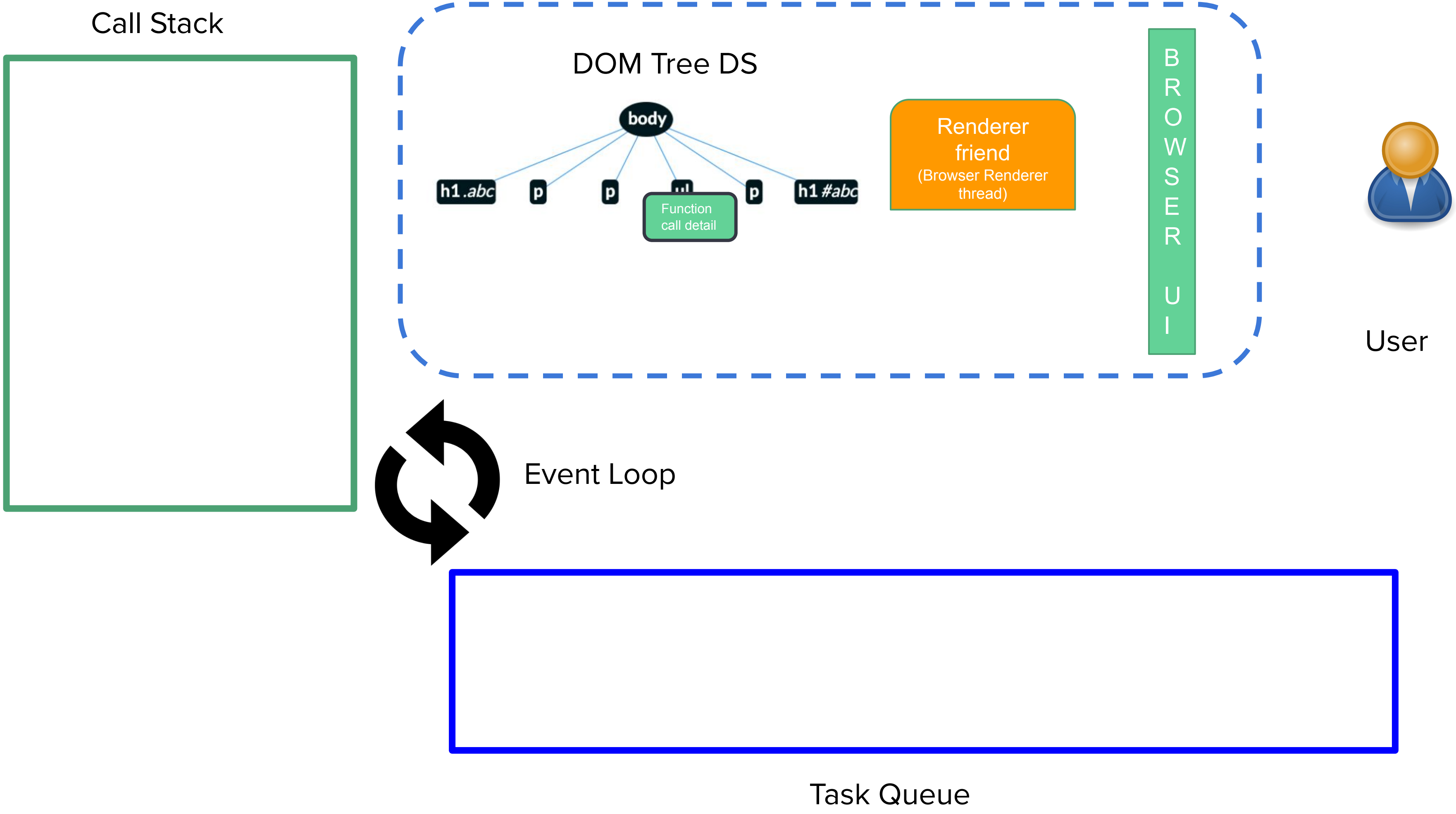
Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

What about the browser?



Call Stack

DOM Tree DS

body

h1 .abc

p

p

ul

p

h1 #abc

Function call detail

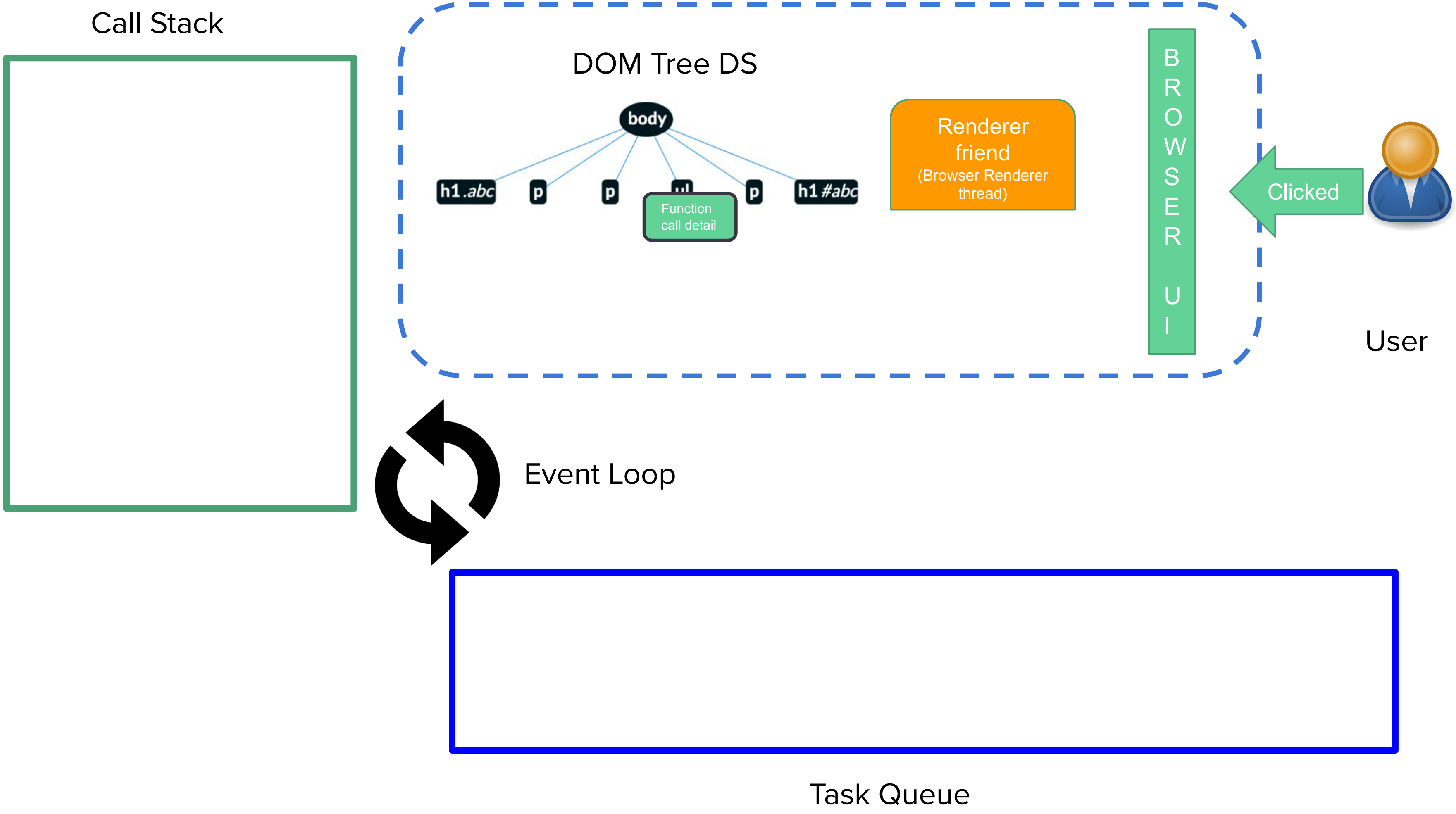
Renderer friend (Browser Renderer thread)

BROWSER UI

User

Event Loop

Task Queue



Call Stack

DOM Tree DS

body

h1.abc

p

p

ul

p

h1#abc

Function call detail

Renderer friend (Browser Renderer thread)

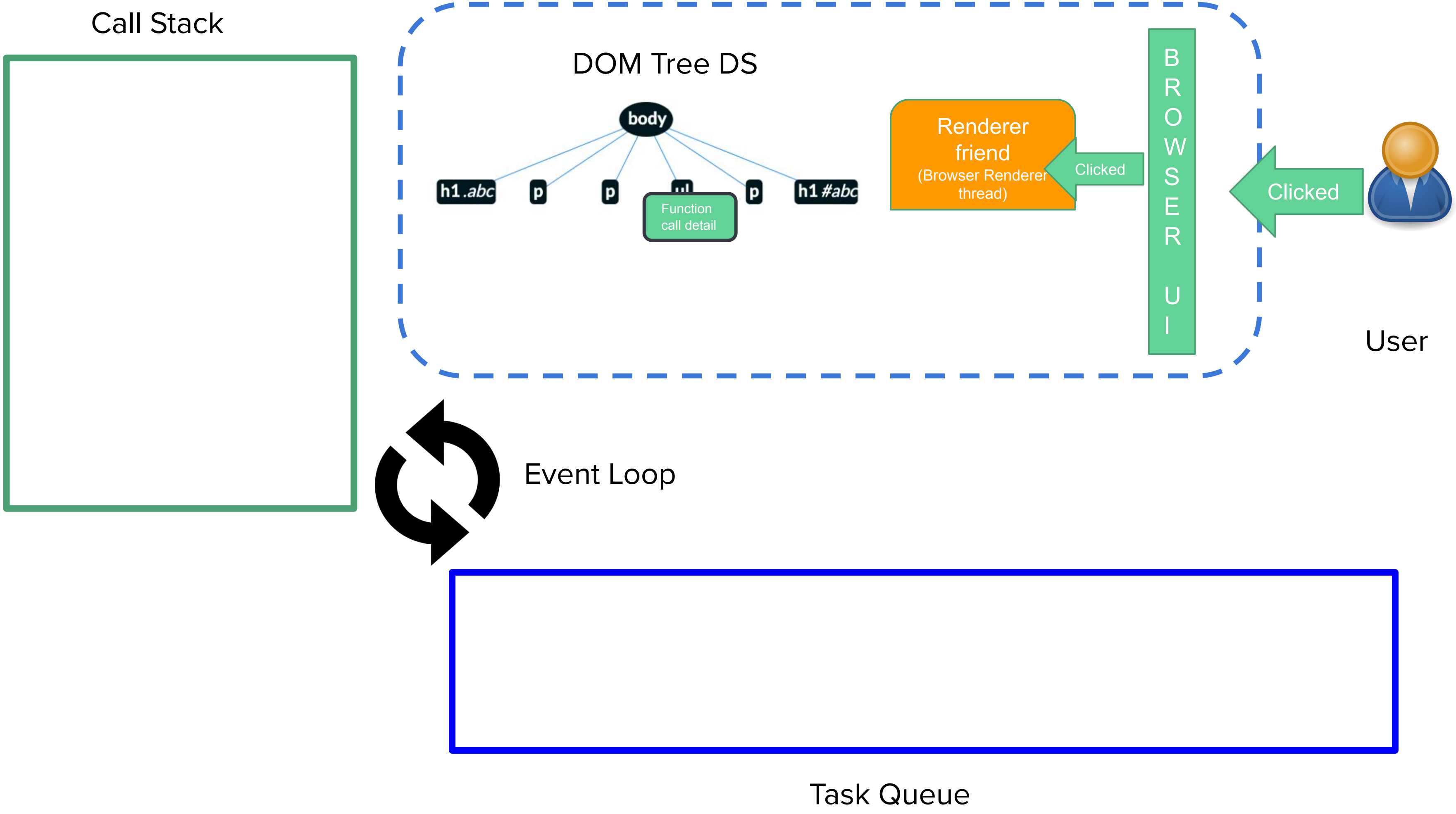
BROWSER UI

Clicked

User

Event Loop

Task Queue



Call Stack

DOM Tree DS

body

h1 .abc

p

p

ul

p

h1 #abc

Function call detail

Renderer friend (Browser Renderer thread)

Clicked

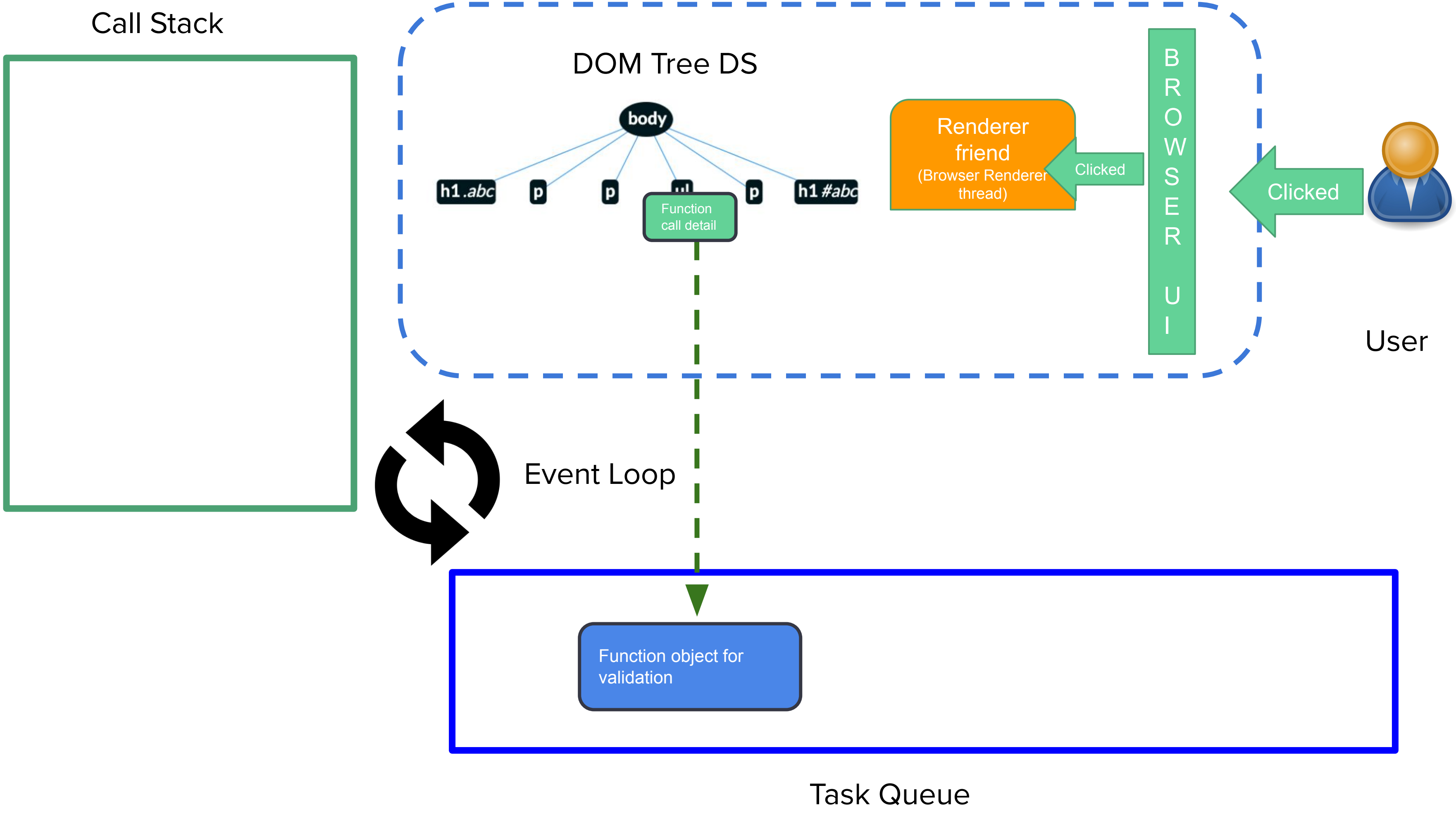
BROWSER UI

Clicked

User

Event Loop

Task Queue



Call Stack

DOM Tree DS

body

h1.abc

p

p

ul

p

h1#abc

Function call detail

Renderer friend (Browser Renderer thread)

Clicked

BROWSER UI

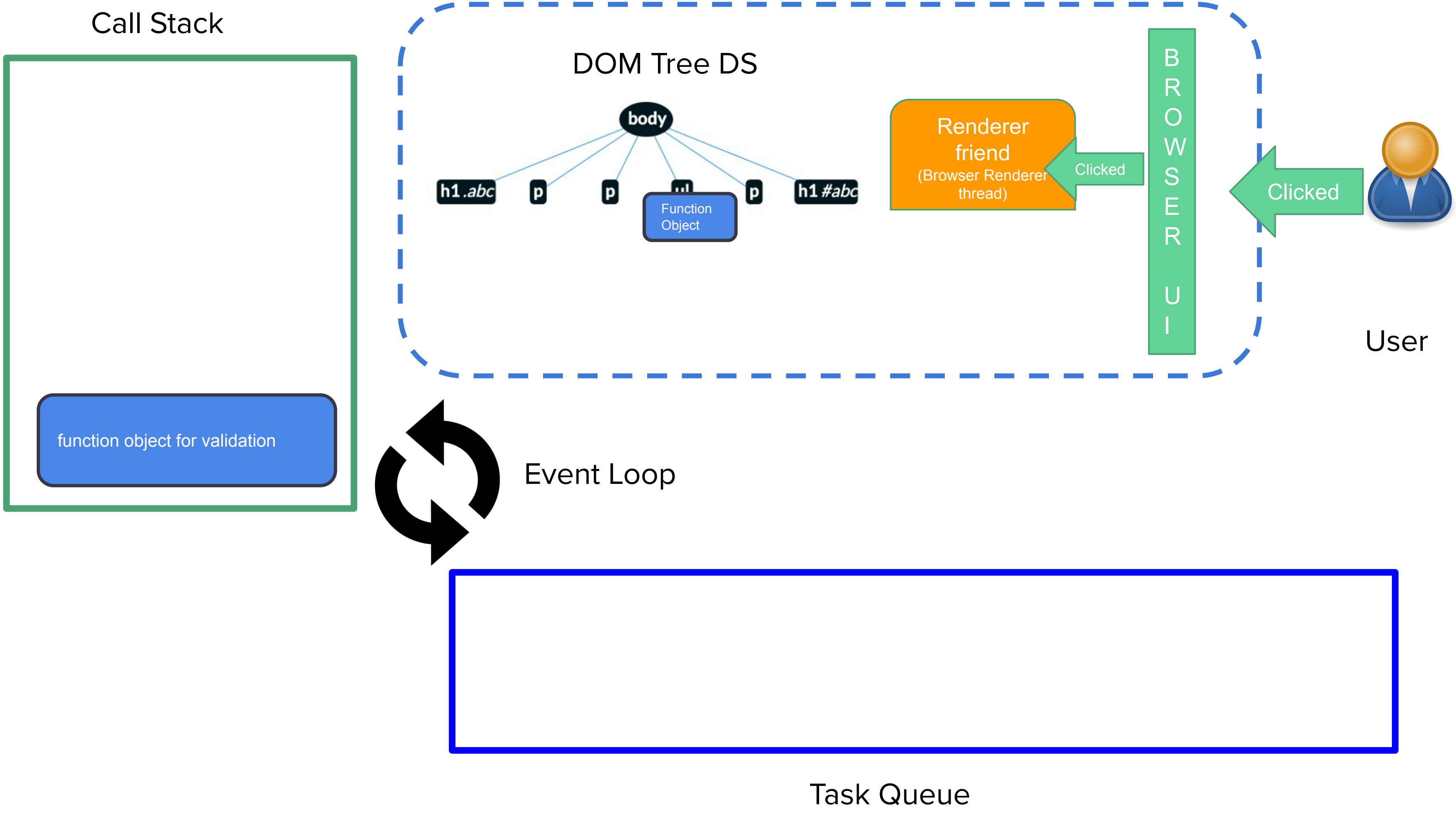
Clicked

User

Event Loop

Function object for validation

Task Queue



Call Stack

function object for validation

DOM Tree DS

body

h1.abc

p

p

ul

p

h1#abc

Function Object

Renderer friend (Browser Renderer thread)

Clicked

BROWSER UI

Clicked

User

Event Loop

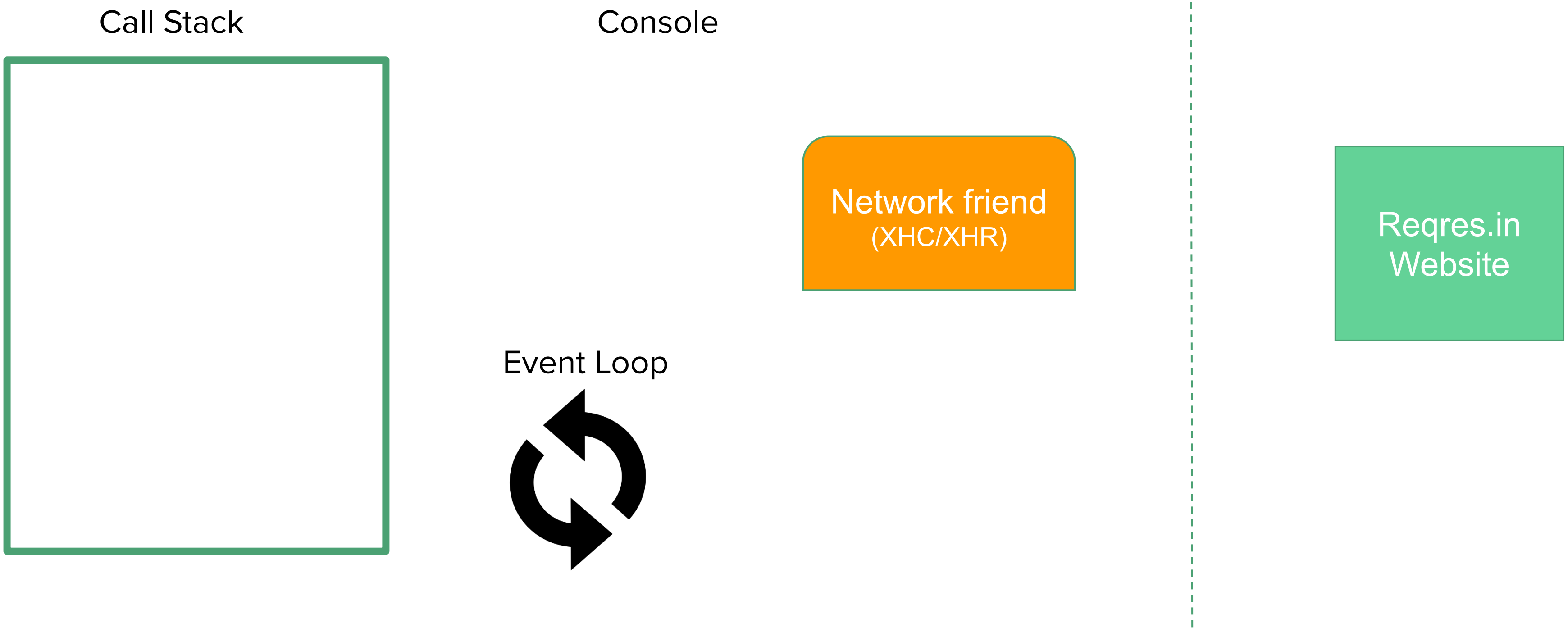
Task Queue

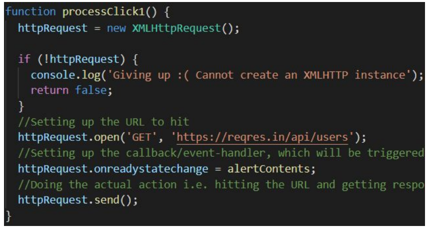
Who does all This?

- libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js
- libev is a high-performance event loop/event model with lots of features used by the Chromium project (needs some verification)

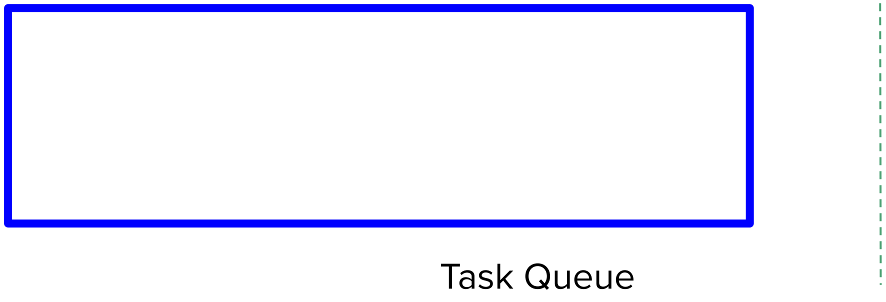
Summary

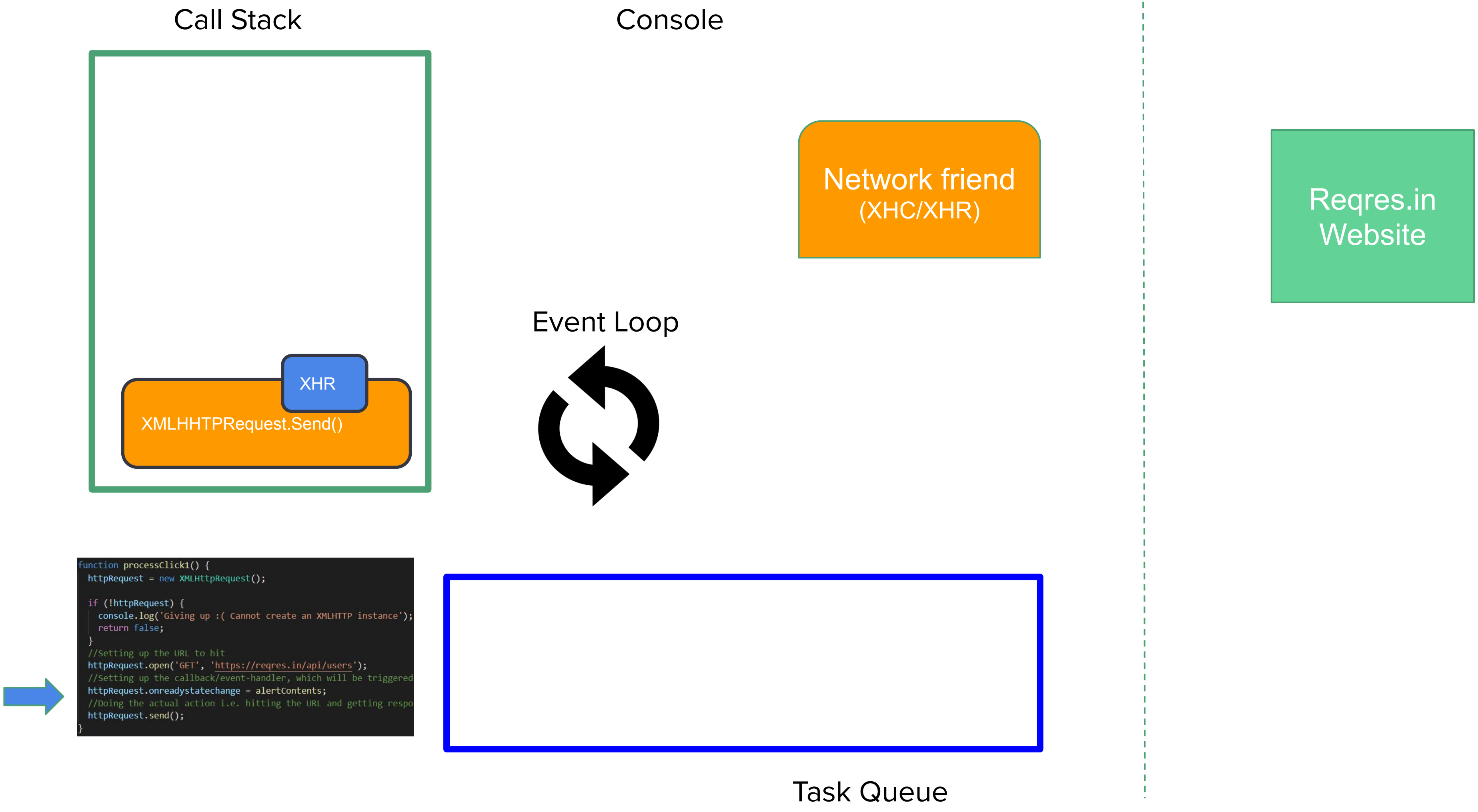
- The infrastructure around JS engines + Hosted environment allows for an effortless asynchronous solution.
- The components involve
  - Call Stack
  - Event Loop
  - Task Queue
  - *Microtask Queue (we will discuss this when we see promises)*



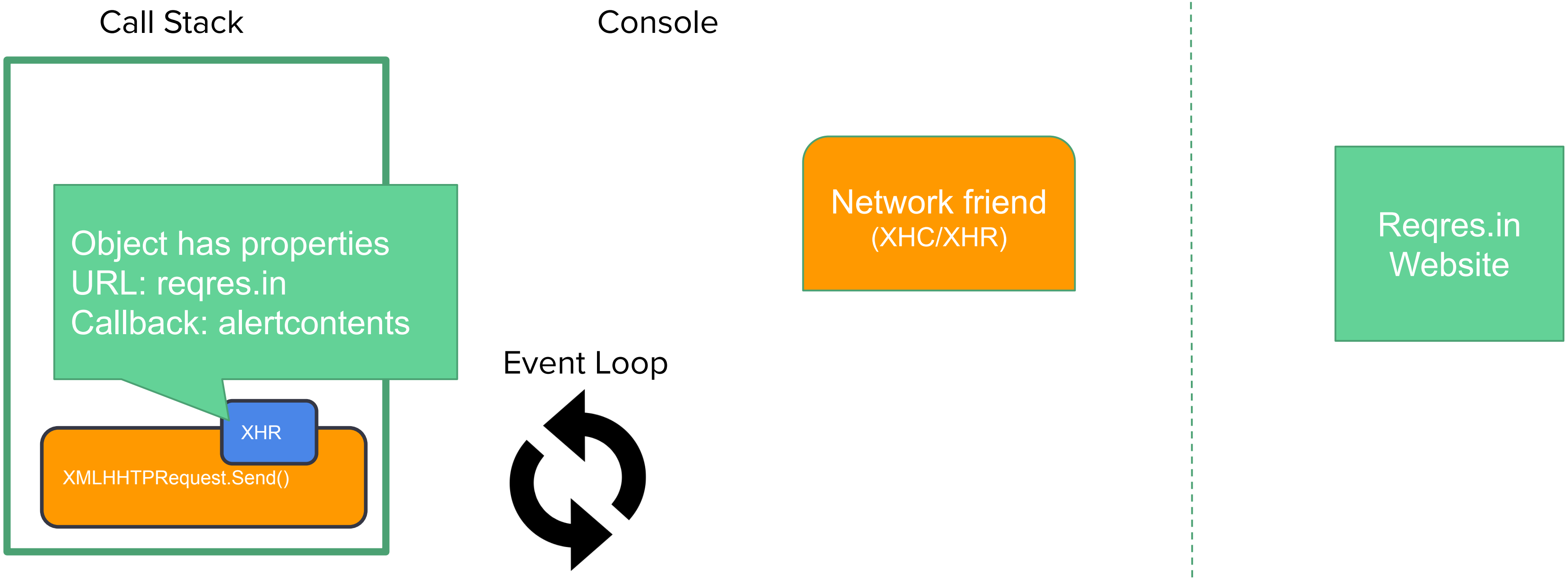


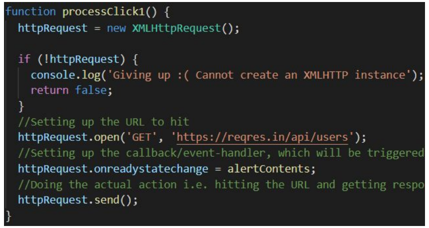
```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```





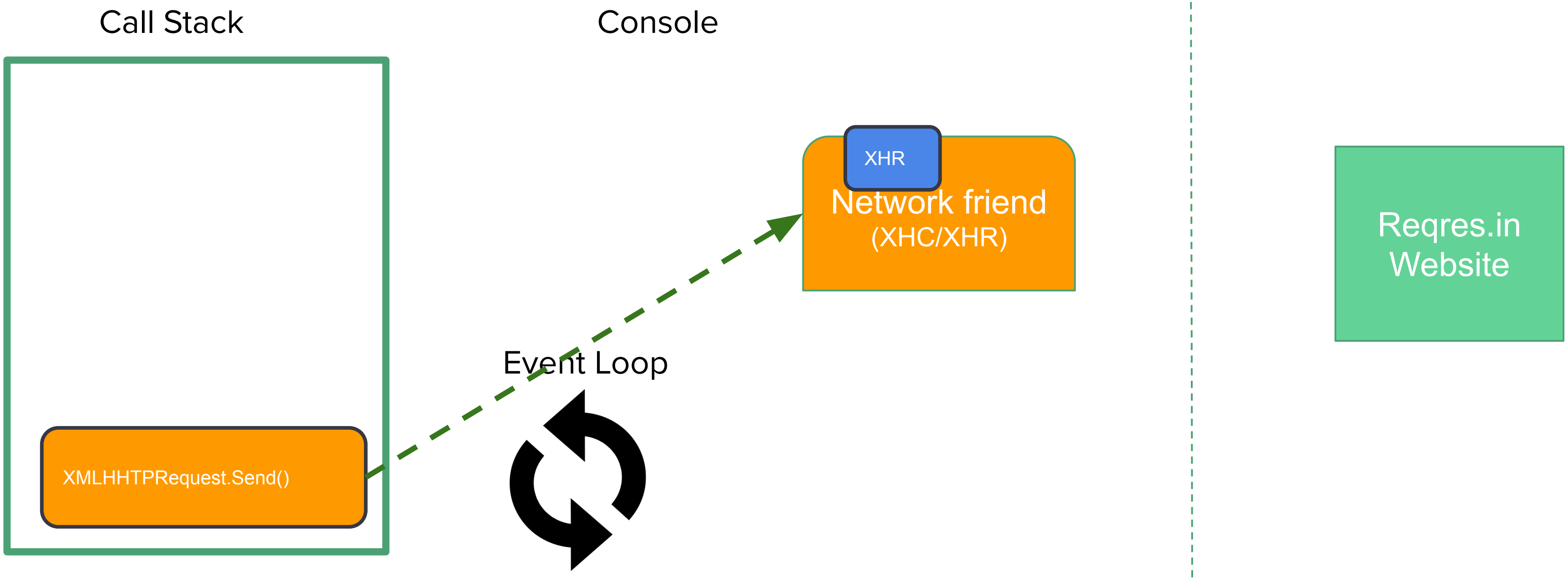
```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```

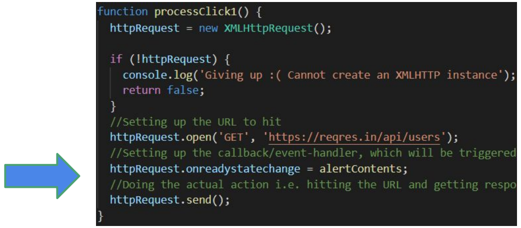




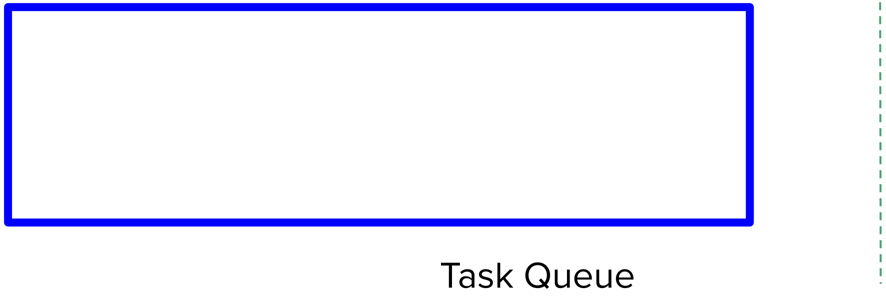
```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```

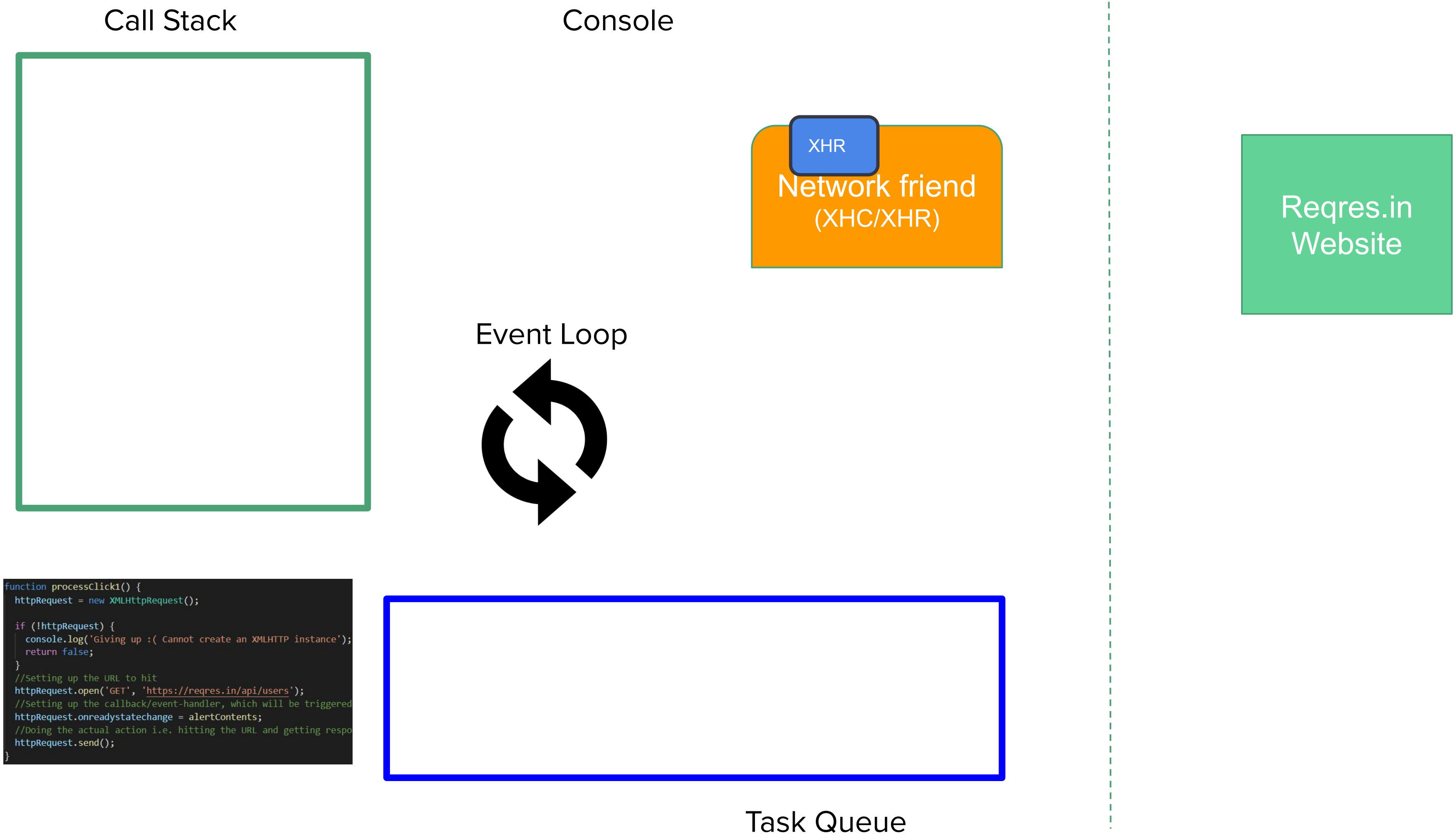
Task Queue

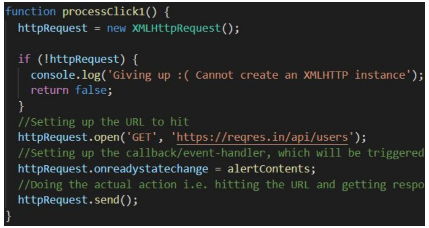




```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```



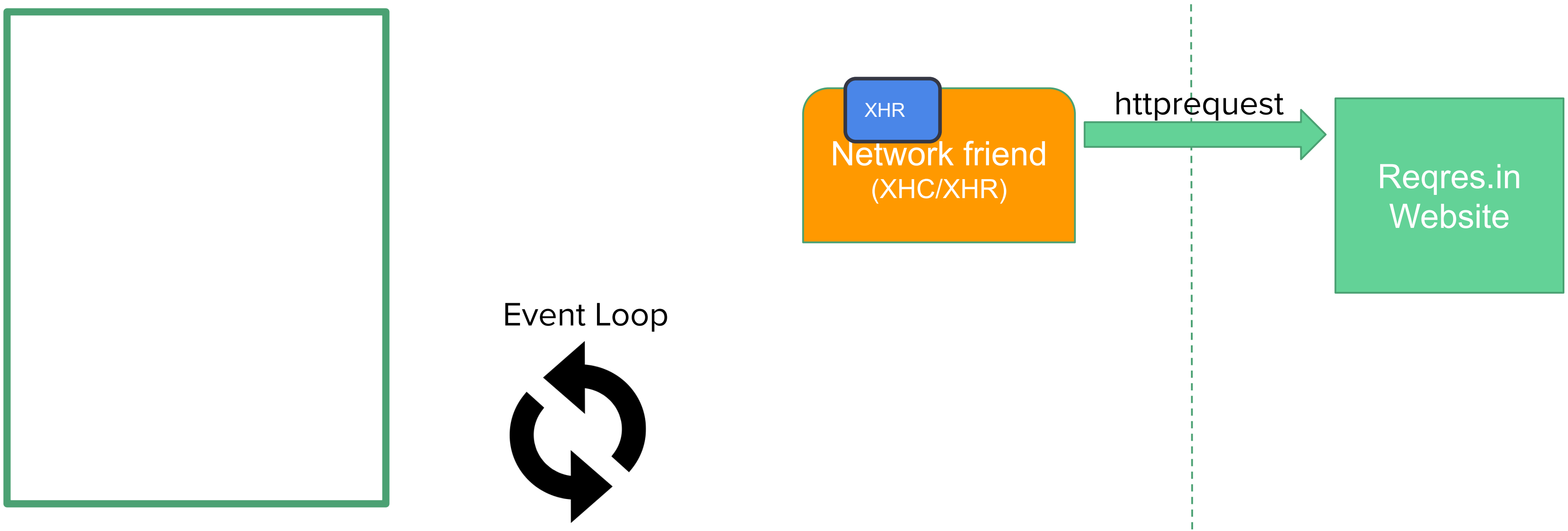


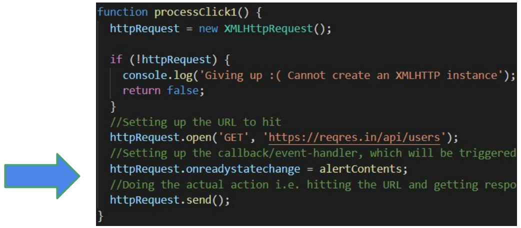


```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```

Call Stack

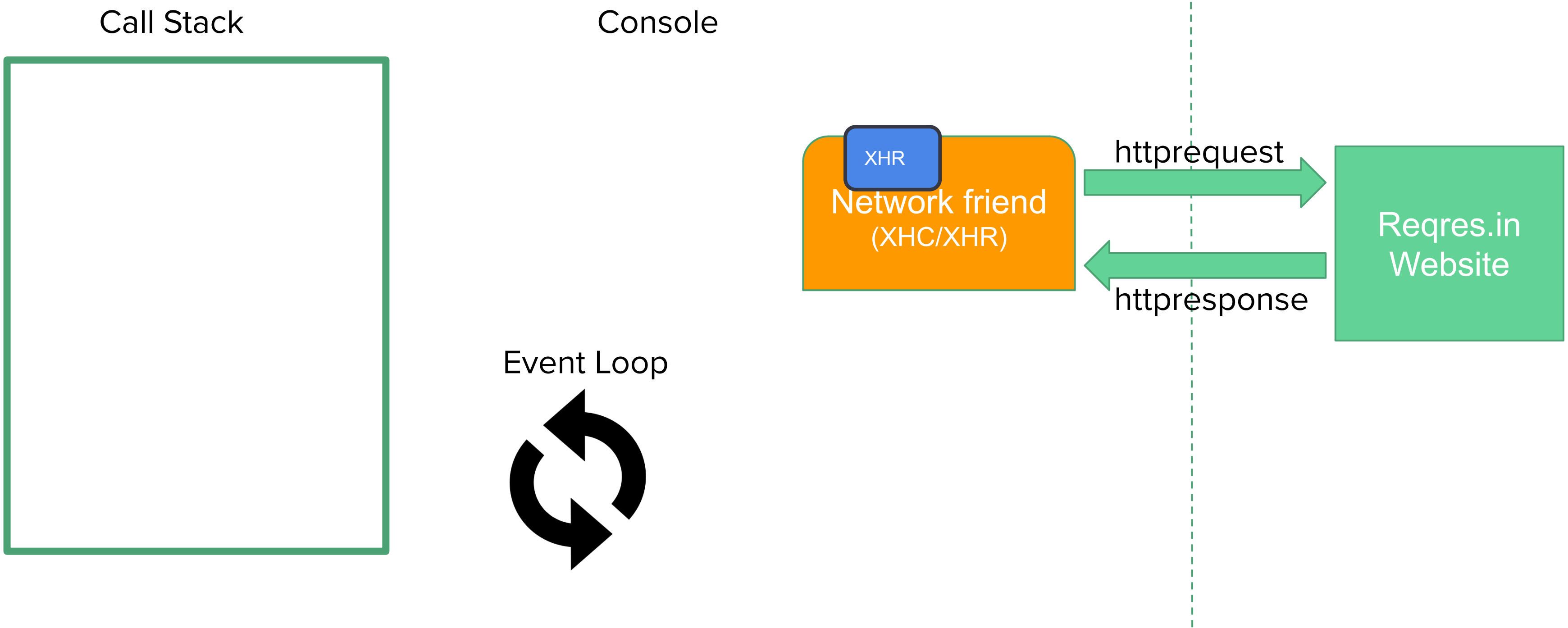
Console

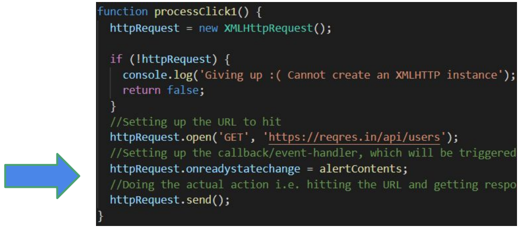




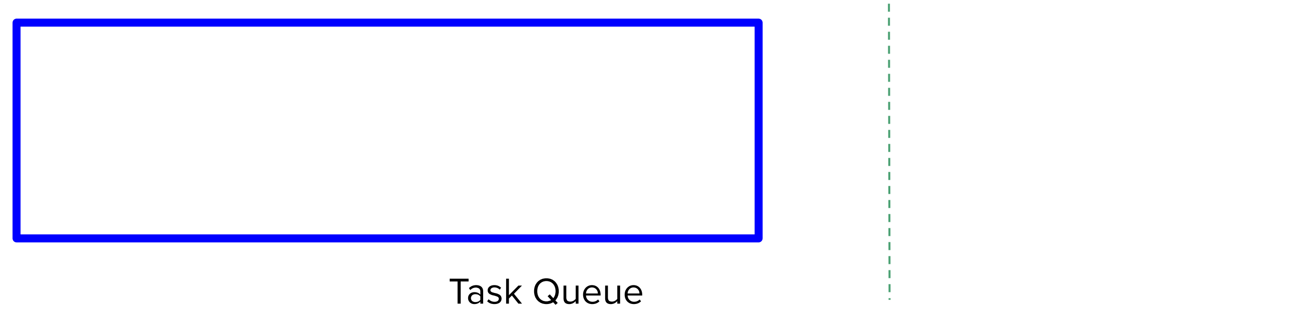
```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```

Task Queue



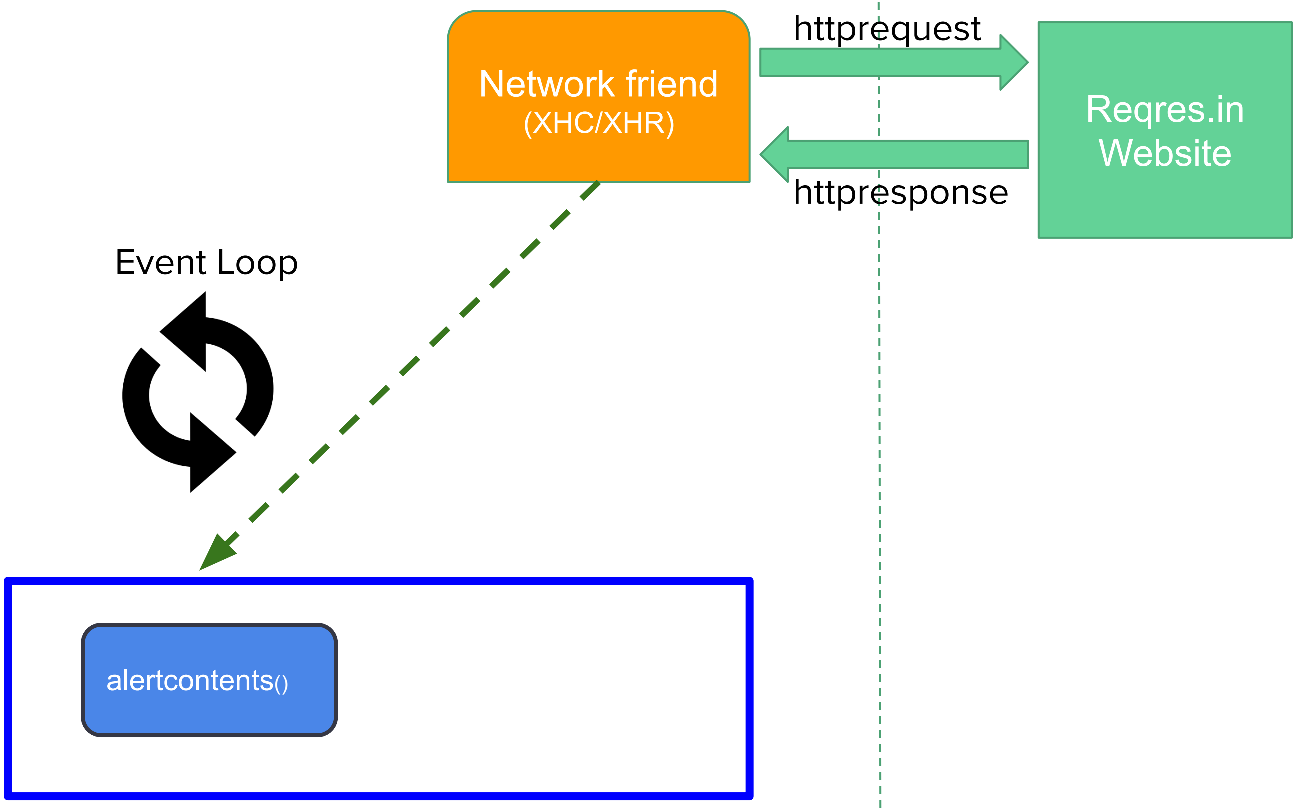


```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```

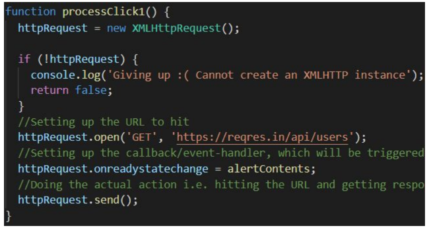


Call Stack

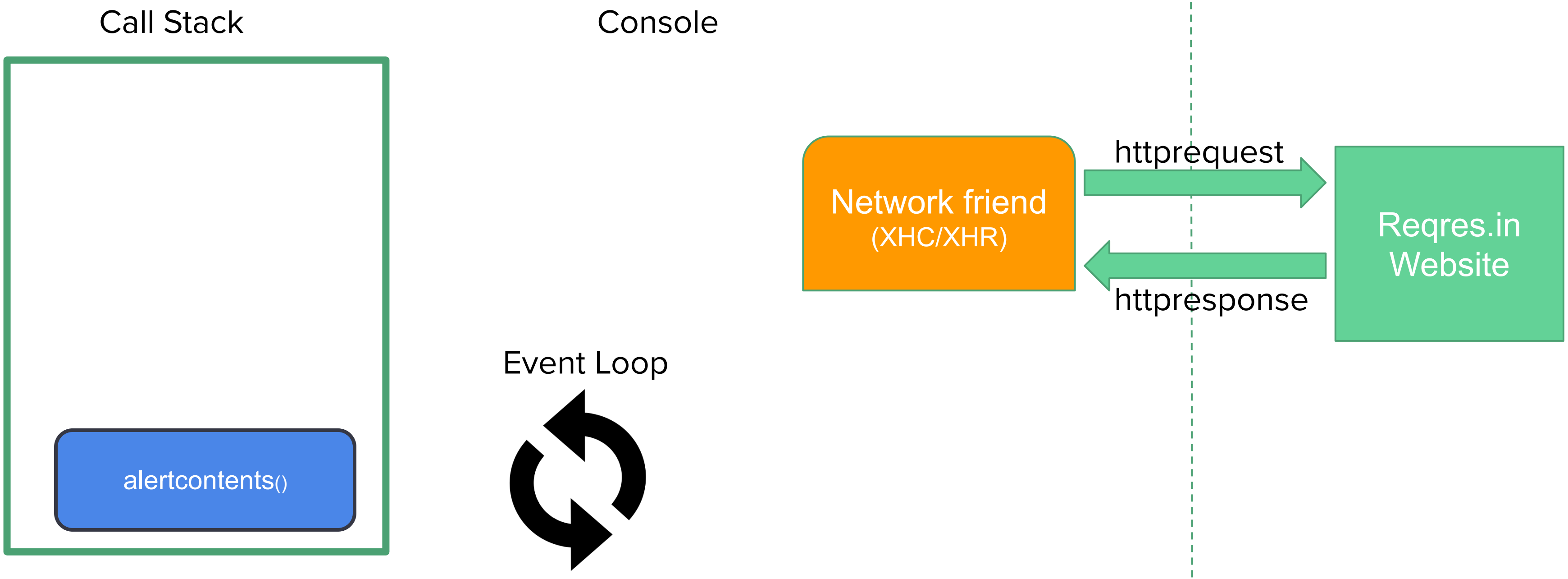
Console

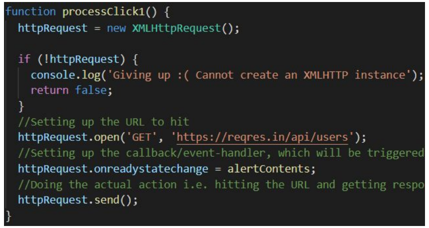


Task Queue



```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```



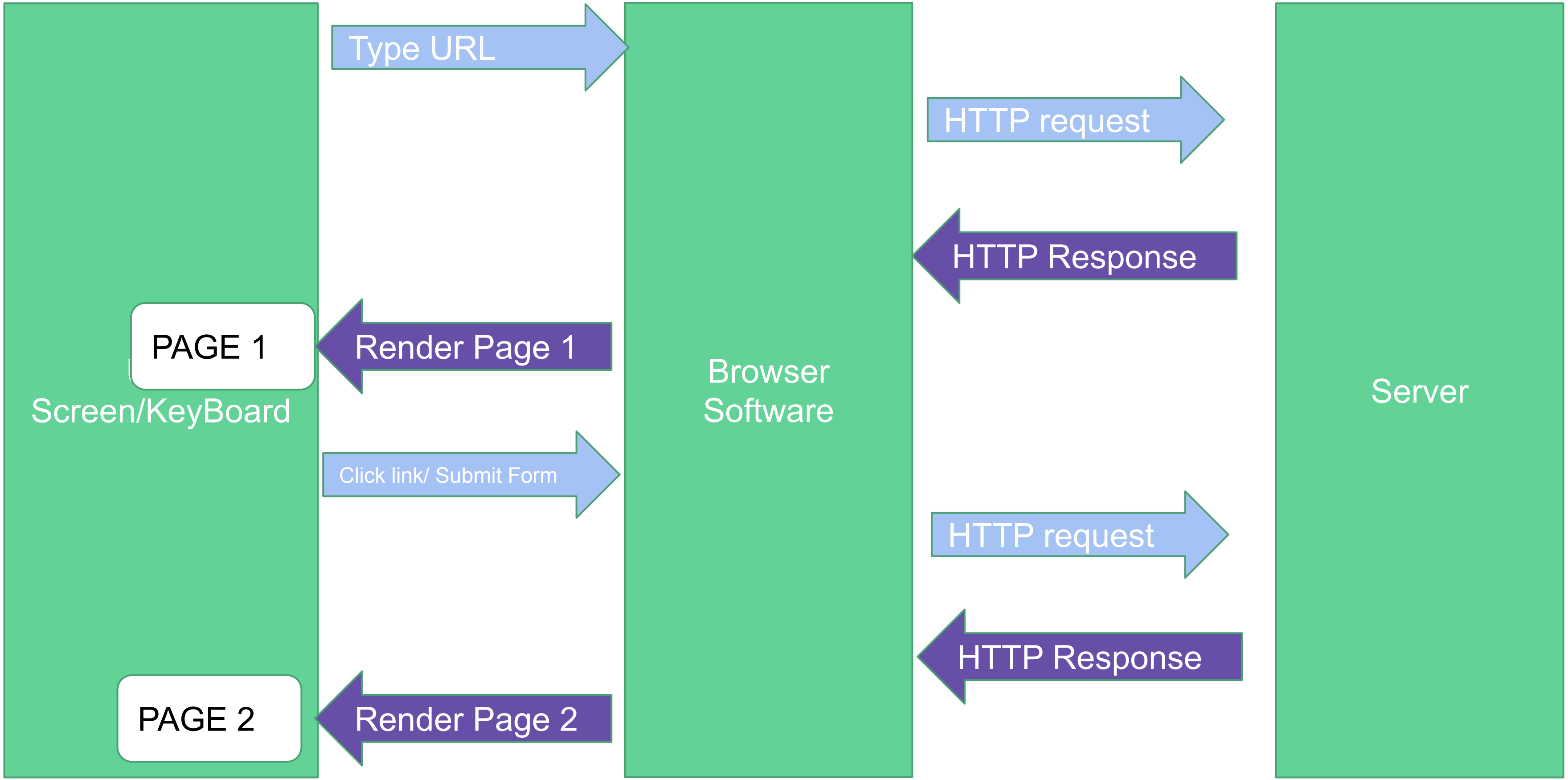


```
function processClick1() {
  httpRequest = new XMLHttpRequest();

  if (!httpRequest) {
    console.log('Giving up :( Cannot create an XMLHTTP instance');
    return false;
  }
  //Setting up the URL to hit
  httpRequest.open('GET', 'https://reqres.in/api/users');
  //Setting up the callback/event-handler, which will be triggered
  httpRequest.onreadystatechange = alertContents;
  //Doing the actual action i.e. hitting the URL and getting respo
  httpRequest.send();
}
```

Task Queue

Multi-Page vs Single-Page

MULTI PAGE APPLICATION



Screen/KeyBoard

PAGE 1

PAGE 2

Type URL

Render Page 1

Click link/ Submit Form

Render Page 2

Browser Software

HTTP request

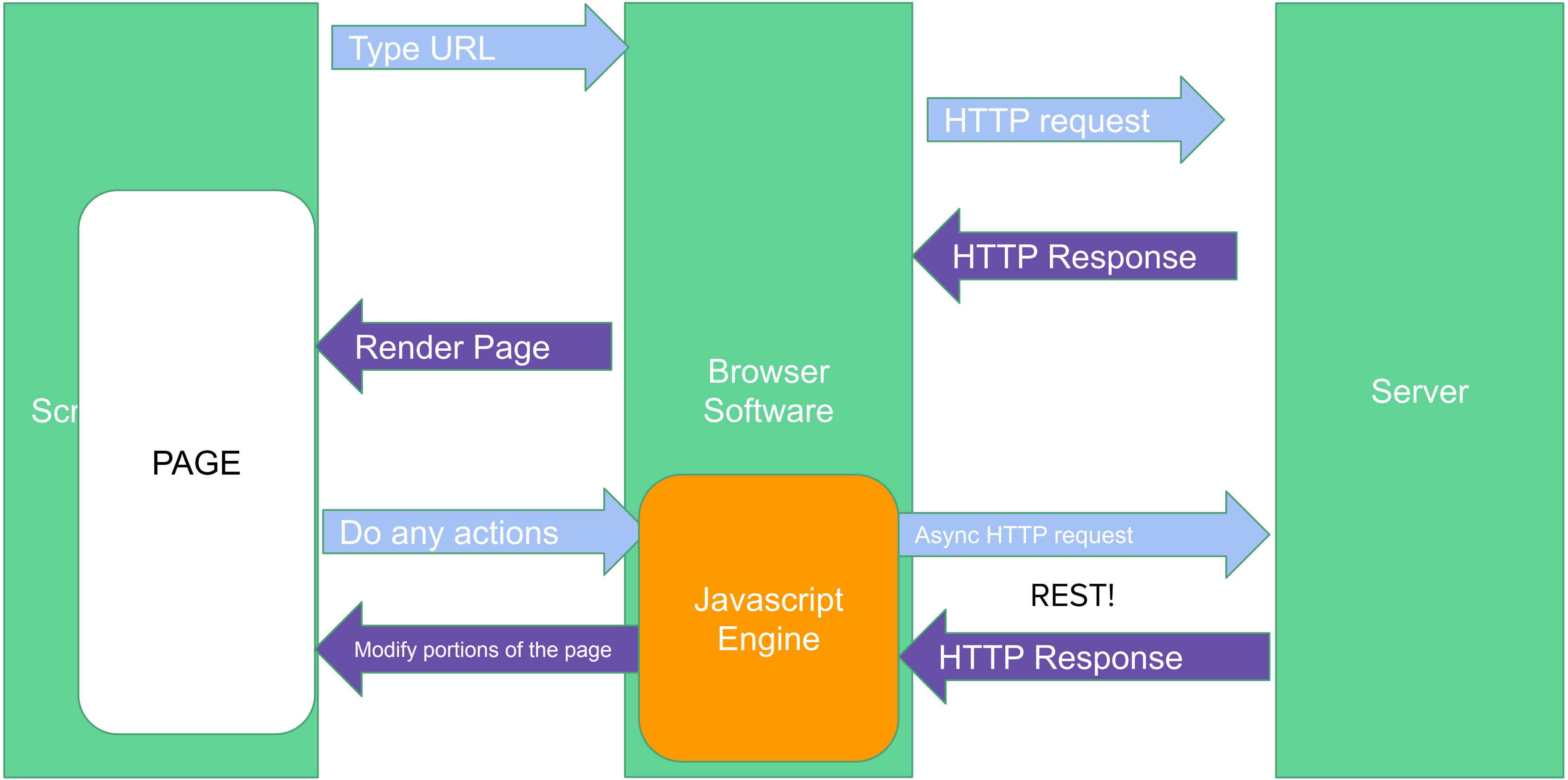
HTTP Response

HTTP request

HTTP Response

Server

SINGLE PAGE APPLICATION

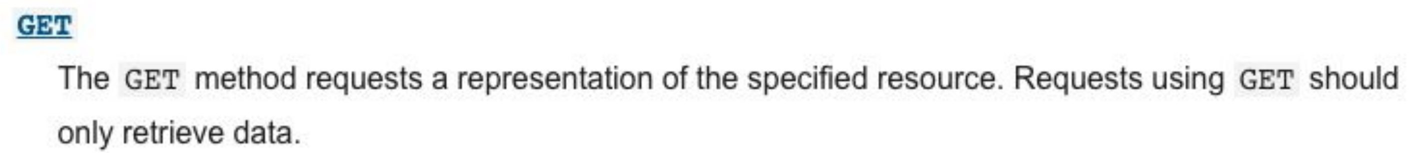


REST - Representational State Transfer

- REST is an architectural style for services
- Identification & manipulation of resources using **HTTP Verbs**
- For now, you can think of “**resources**” as
  - Domain Objects (models) : Customers, Orders, etc
- Deeper exploration of webservices in FSD3!
  - Including History, legacy types & future

HTTP Verbs

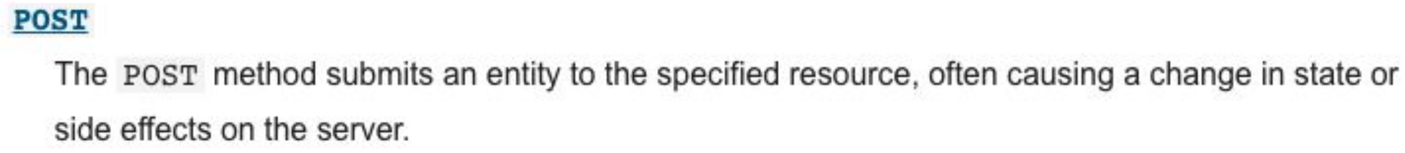
C



GET

The `GET` method requests a representation of the specified resource. Requests using `GET` should only retrieve data.

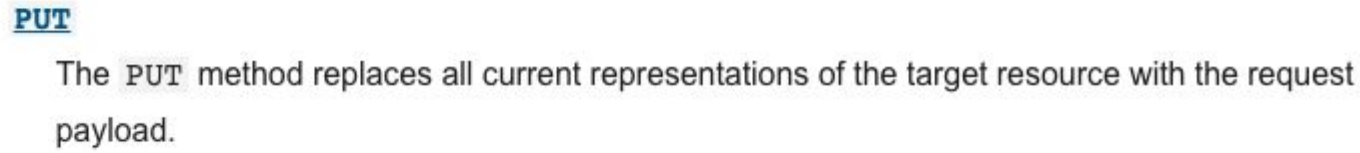
R



POST

The `POST` method submits an entity to the specified resource, often causing a change in state or side effects on the server.

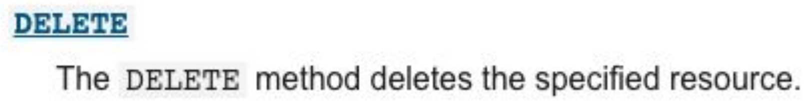
U



PUT

The `PUT` method replaces all current representations of the target resource with the request payload.

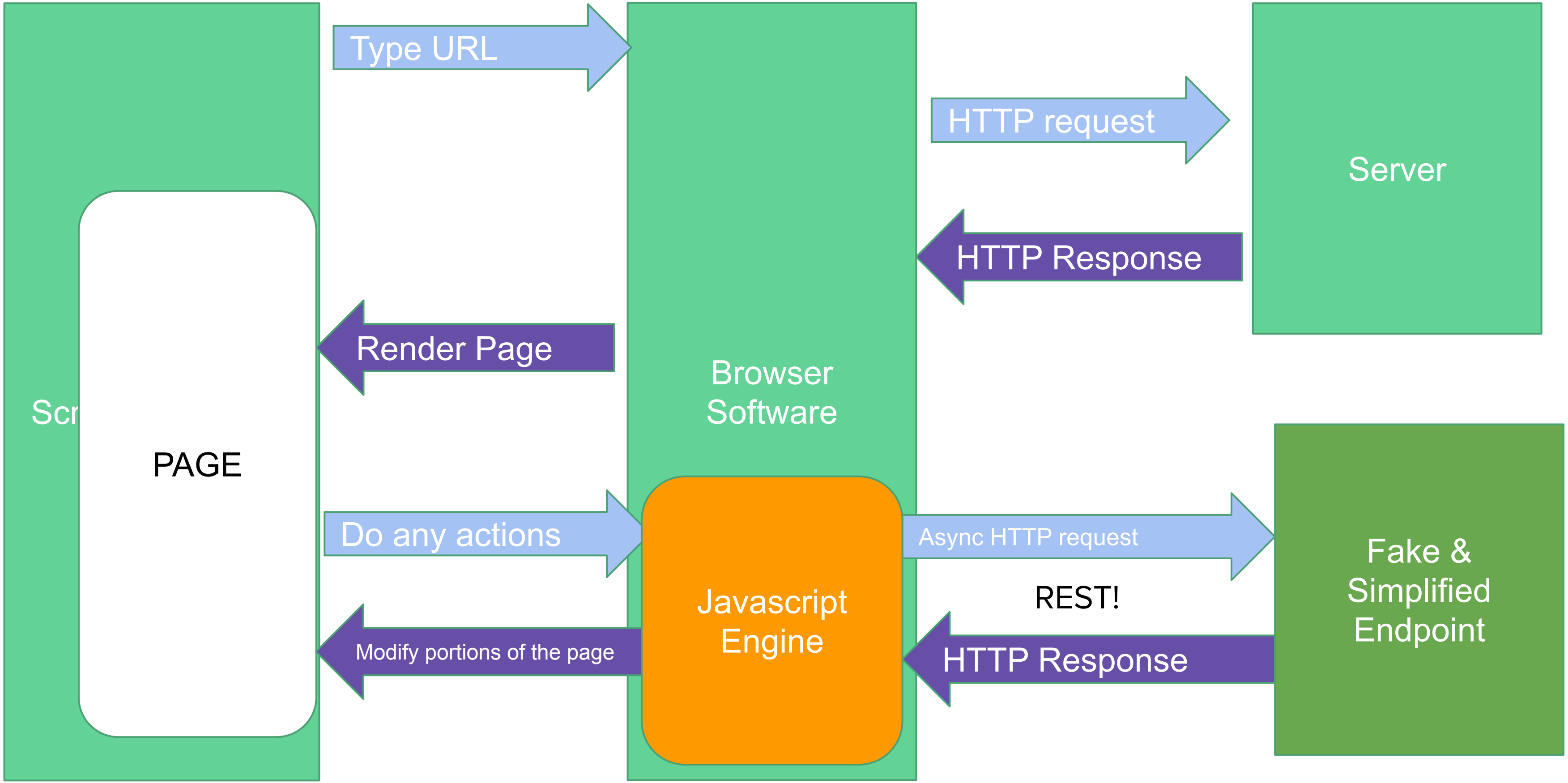
D



DELETE

The `DELETE` method deletes the specified resource.

SINGLE PAGE APPLICATION - FRONT END DEVELOPMENT PHASE



So that you can focus primarily on front end development!

LET US UNDERSTAND HTTP

- https://developer.mozilla.org/en-US/docs/Web/HTTP

GET SOME REST

- https://restfulapi.net/