
accenture


improving
It's what we do. ™

{Adventures in Rendering Off the Main Thread}


Simon MacDonald
@macdonst@mastodon.online
Simon MacDonald
@macdonst@mastodon.online


SN
WARRIOR
88

I  JS


DEVELOPERS
JAVASCRIPT
WEBSITE

Why do we care about the main thread?

Main Thread Responsibilities

Main Thread Responsibilities

Parses HTML 

Main Thread Responsibilities

Parses HTML	✓
Builds the DOM	✓

Main Thread Responsibilities

- Parses HTML ✓
- Builds the DOM ✓
- Parses CSS ✓

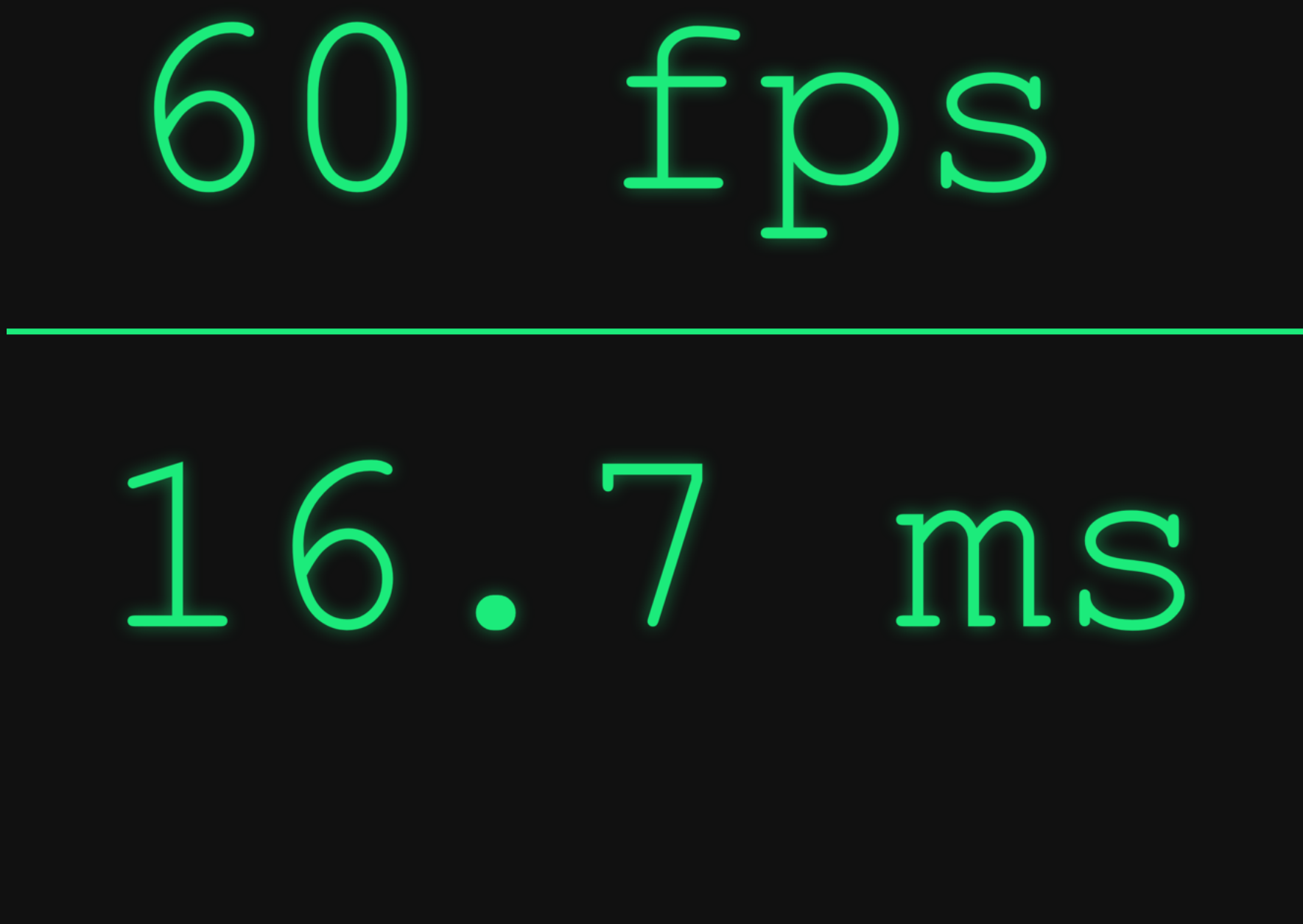
Main Thread Responsibilities

- Parses HTML ✓
- Builds the DOM ✓
- Parses CSS ✓
- Executes JavaScript ✓

Main Thread Responsibilities

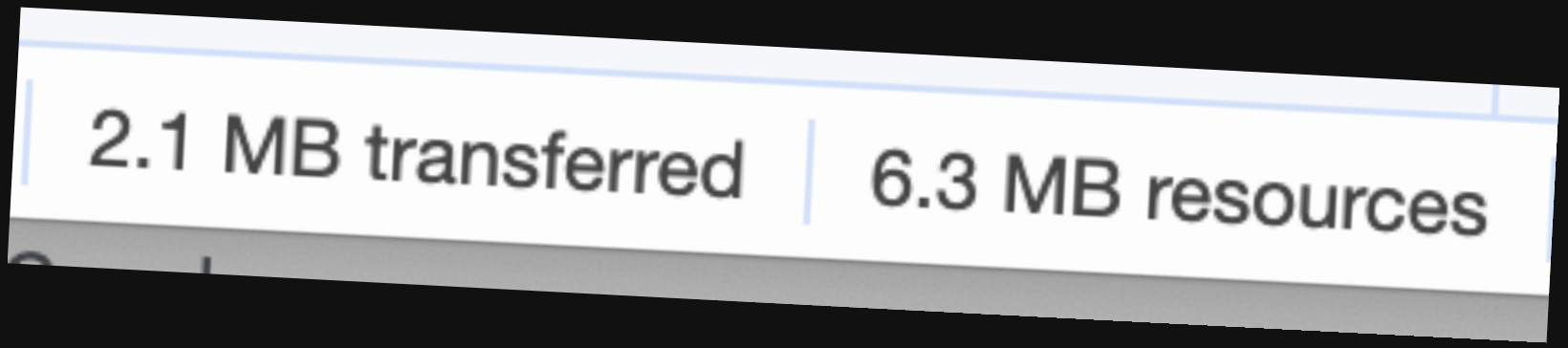
Parses HTML	✓
Builds the DOM	✓
Parses CSS	✓
Executes JavaScript	✓
Responds to User Events	✓


60 fps


60 fps
16.7 ms

State of the Art

State of the Art



index.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <title>My App</title>
5    </head>
6    <body>
7      <div id="app"></div>
8      <script type="module" src="/src/main.js"></
9      <noscript>Bad luck</noscript>
10   </body>
11 </html>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></
    <noscript>Bad luck</noscript>
  </body>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></
    <noscript>Bad luck</noscript>
  </body>
</html>
```

But what if JavaScript fails?

index.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <title>My App</title>
5    </head>
6    <body>
7      <div id="app"></div>
8      <script type="module" src="/src/main.js"></
9      <noscript>Bad luck</noscript>
10   </body>
11 </html>
```

JavaScript never fails, right?

Failure Rate

1-2%

Failure Reasons

Failure Reasons

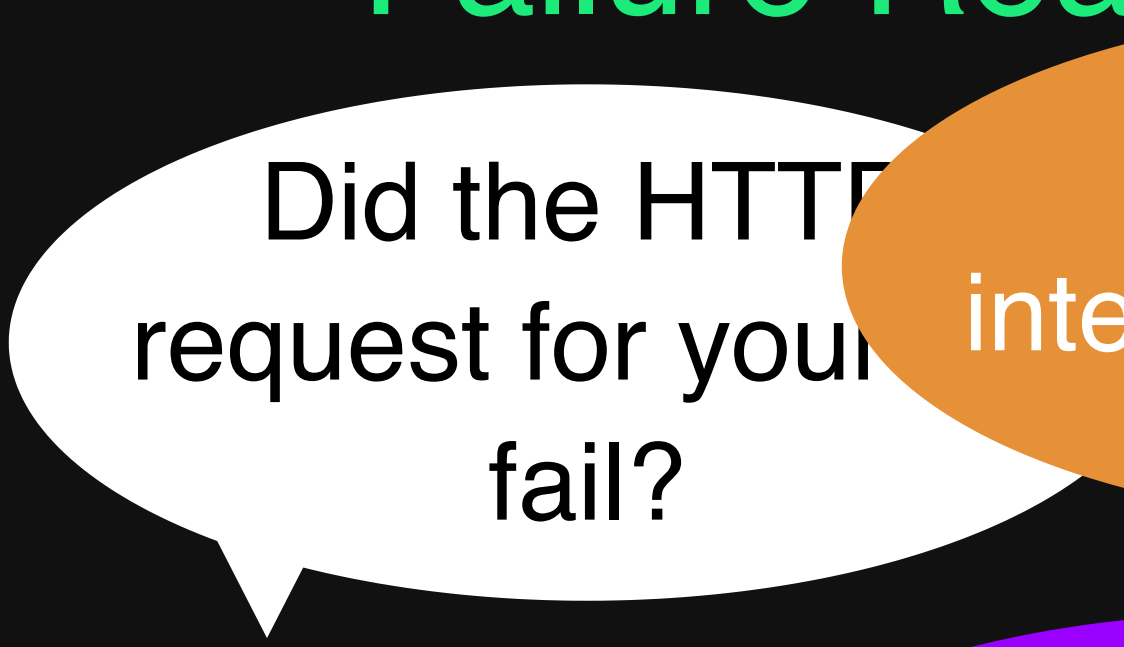
Did the HTTP request for your JS fail?

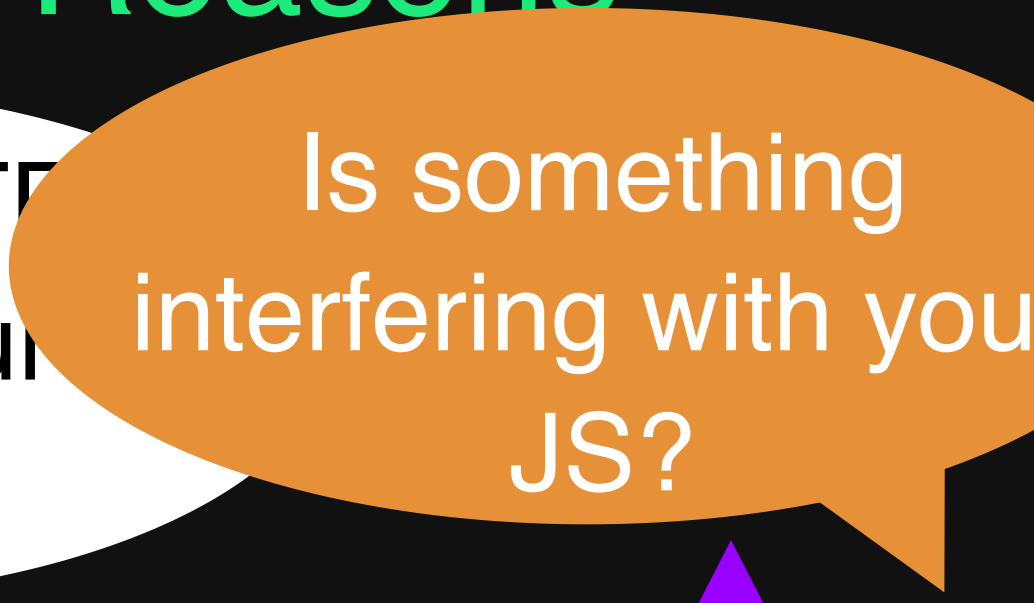
Failure Reasons

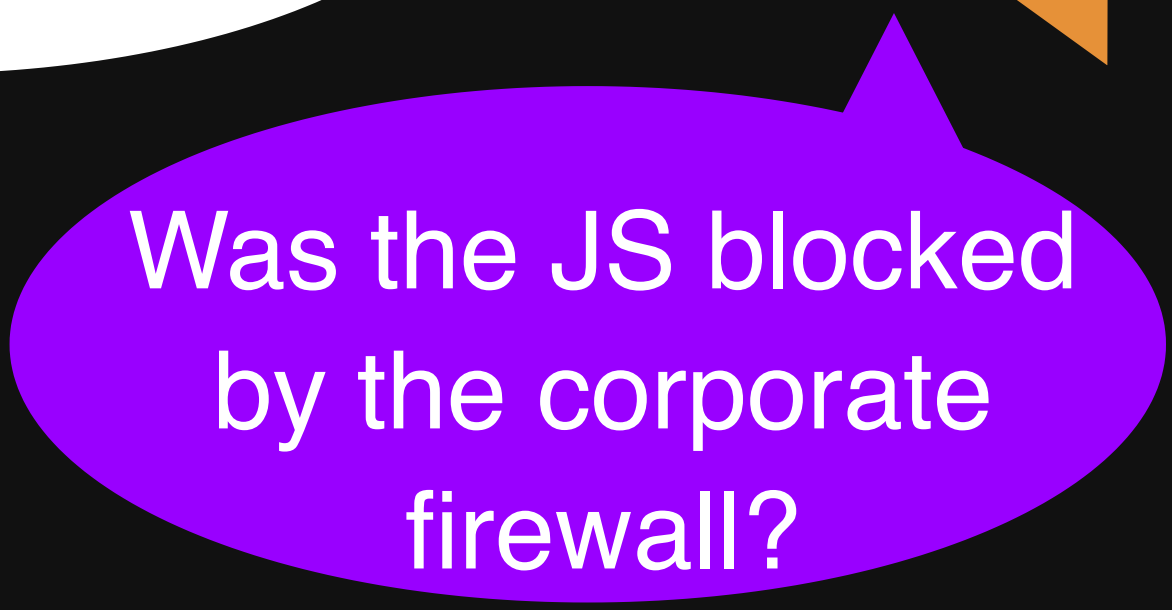
Did the HTTP request for your JS fail?

Was the JS blocked by the corporate firewall?

Failure Reasons







Failure Reasons

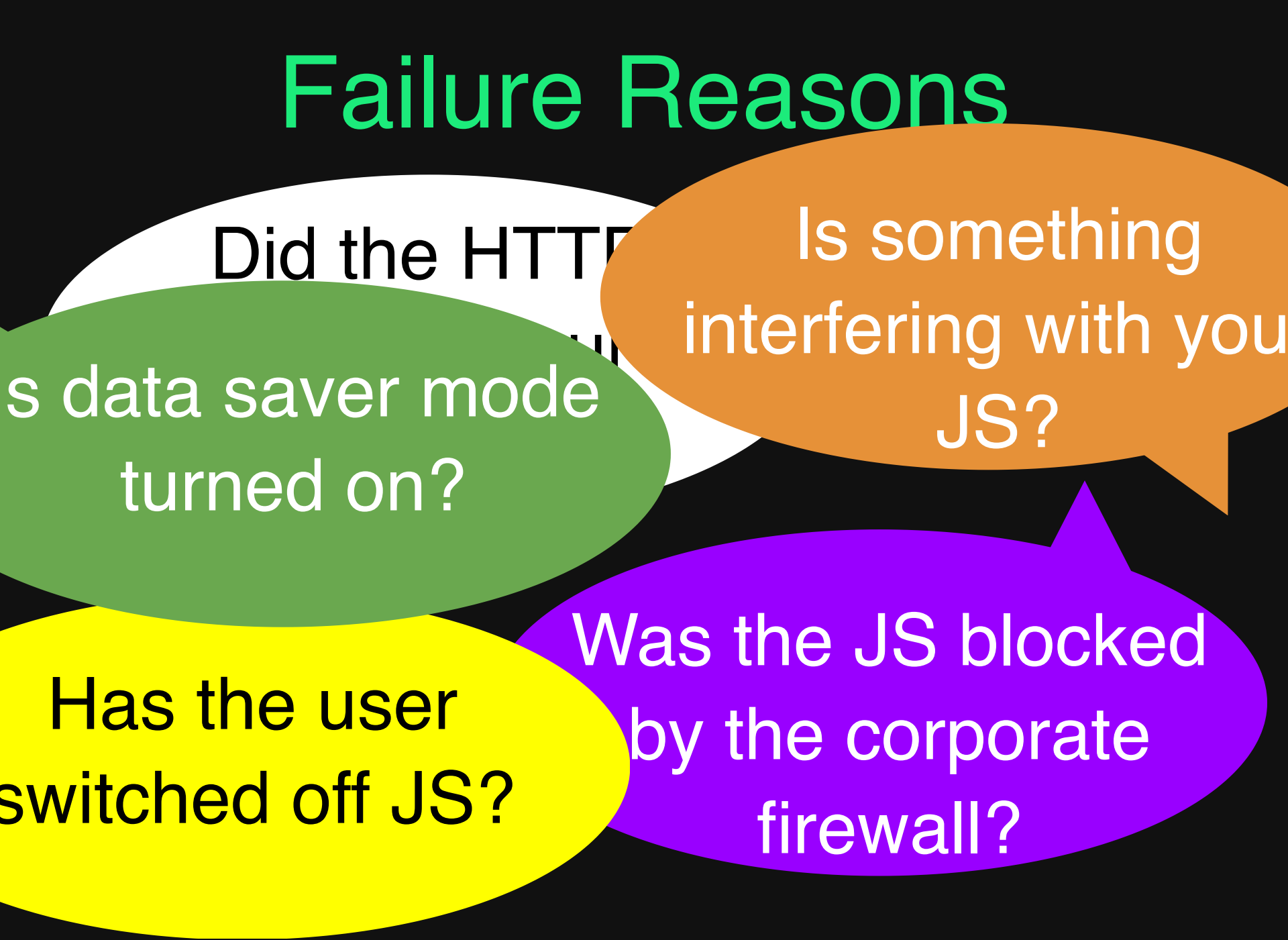
Did the HTTP request for your fail?

Is something interfering with your JS?

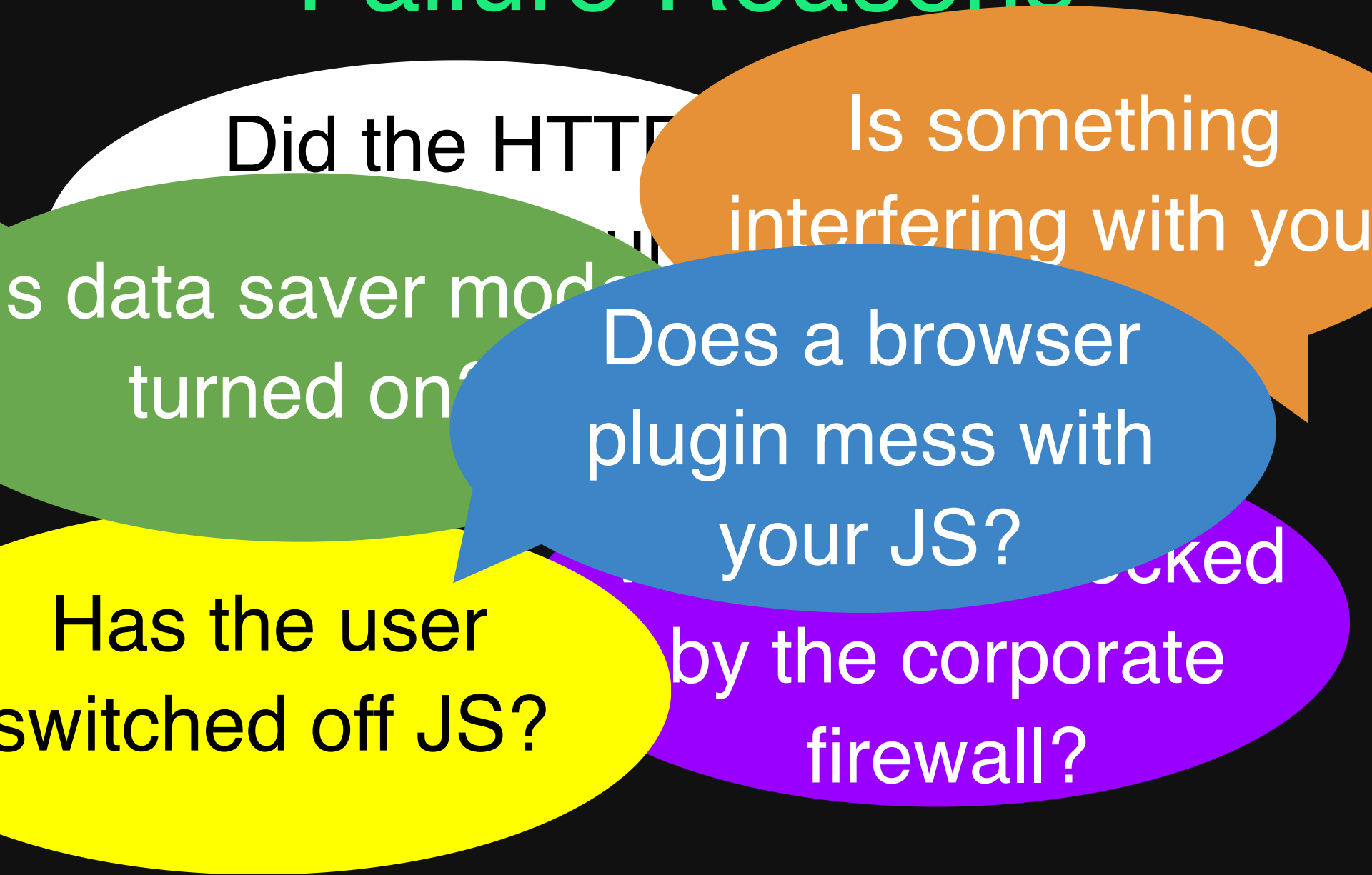
Has the user switched off JS?

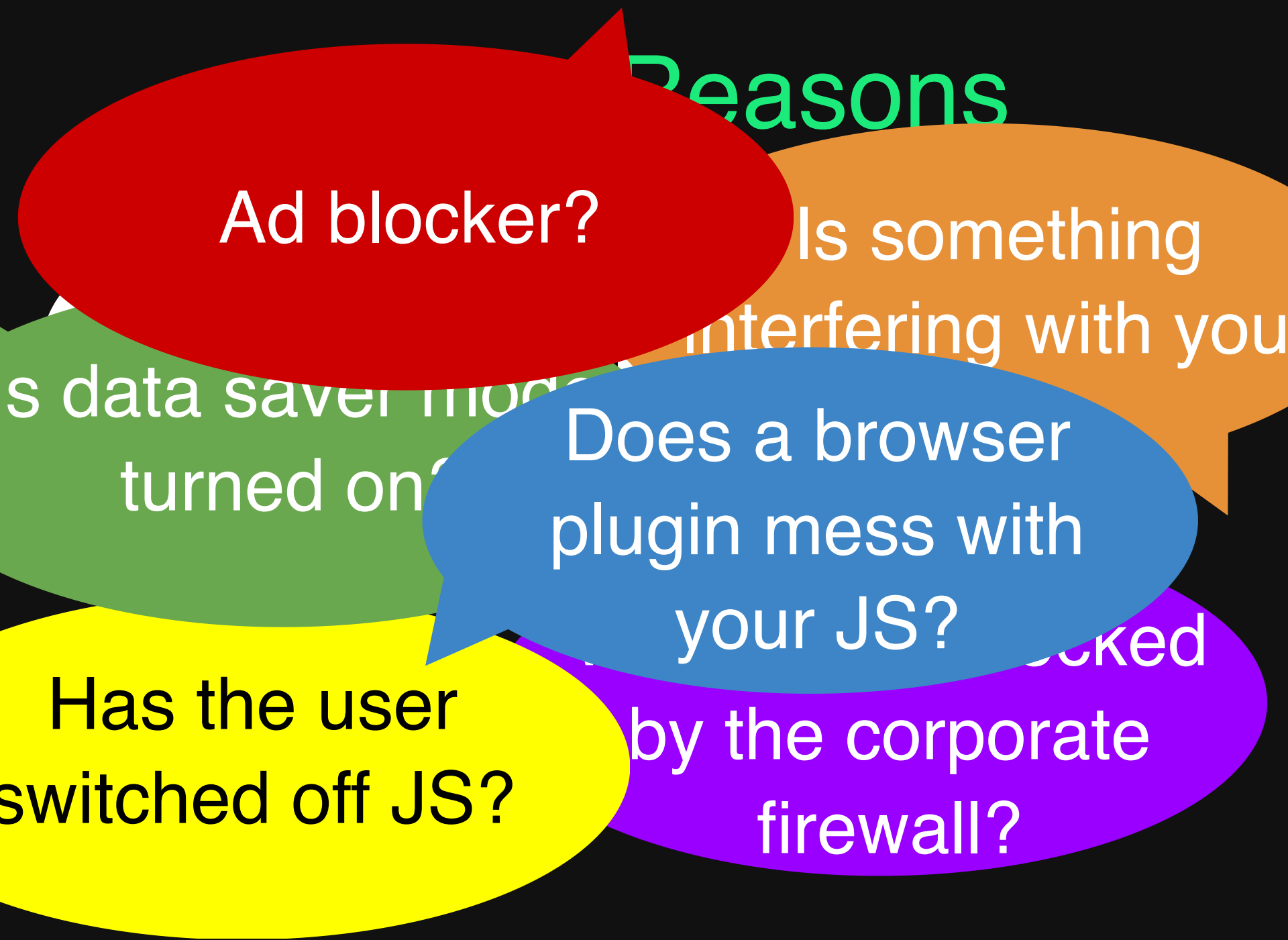
Was the JS blocked by the corporate firewall?

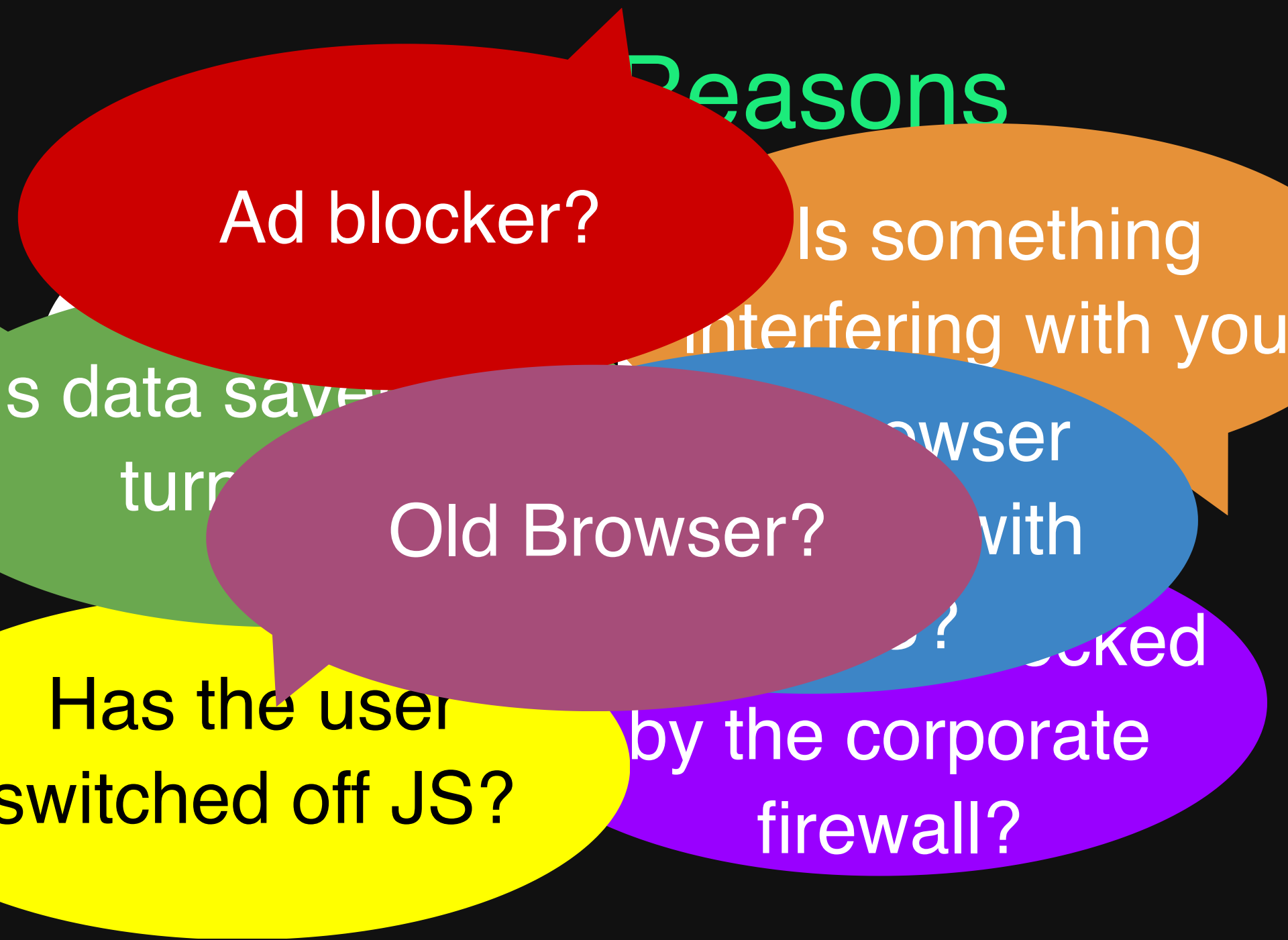
Failure Reasons



Failure Reasons

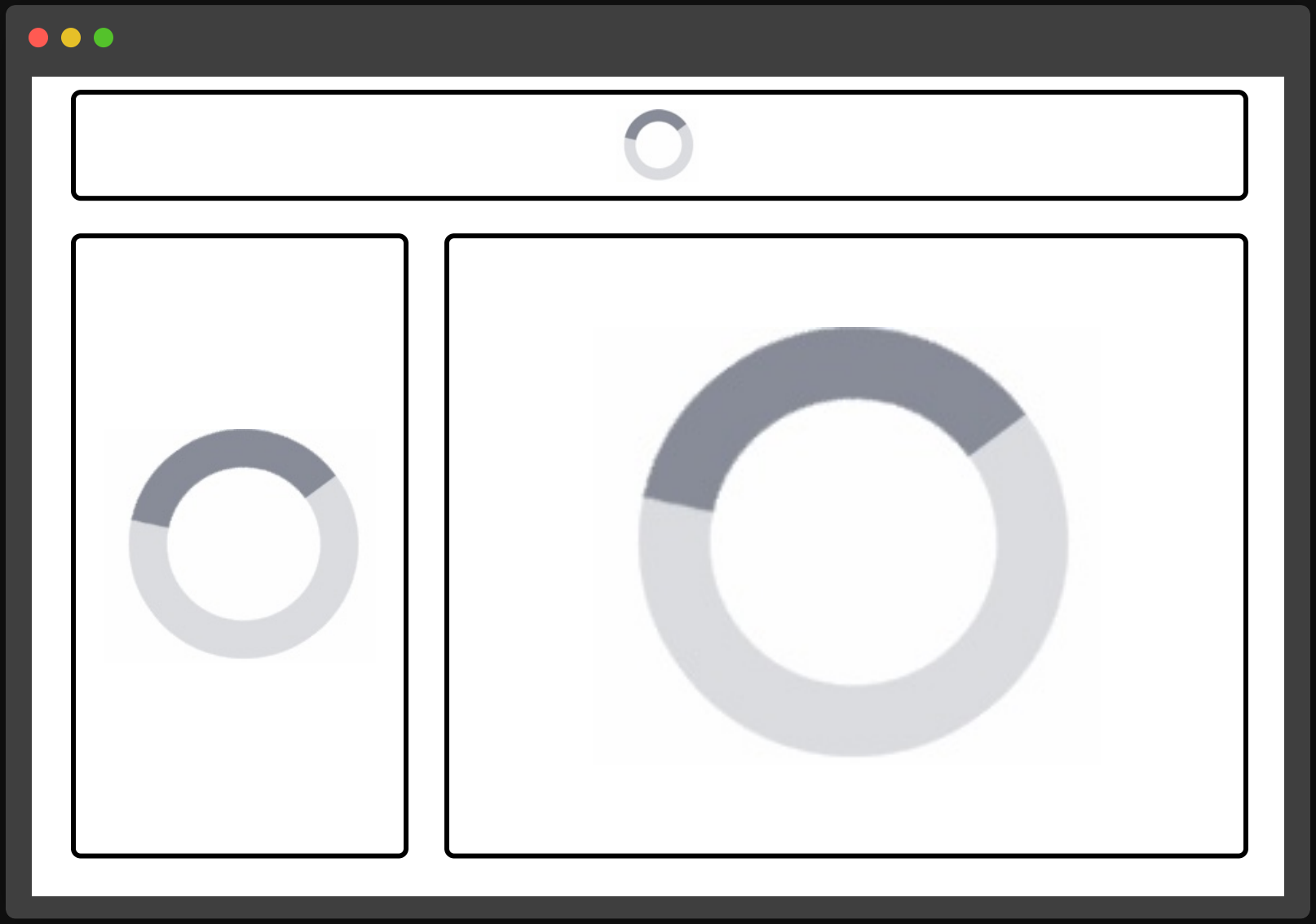



Reasons
Ad blocker?
Is something
interfering with you
s data saver mod
turned on?
Does a browser
plugin mess with
your JS?
Has the user
switched off JS?
by the corporate
firewall?


Reasons
Ad blocker?
Is something
interfering with you
s data save
turn
owser
Old Browser?
vith
?
cked
Has the user
switched off JS?
by the corporate
firewall?

Failure Rates

Page Loads	JavaScript Failures
100	1-2
1,000	10-20
10,000	100-200
100,000	1,000-2,000
1,000,000	10,000-20,000



Adventure

1.

Reduce

Reduce your JavaScript footprint by server side rendering HTML.

1

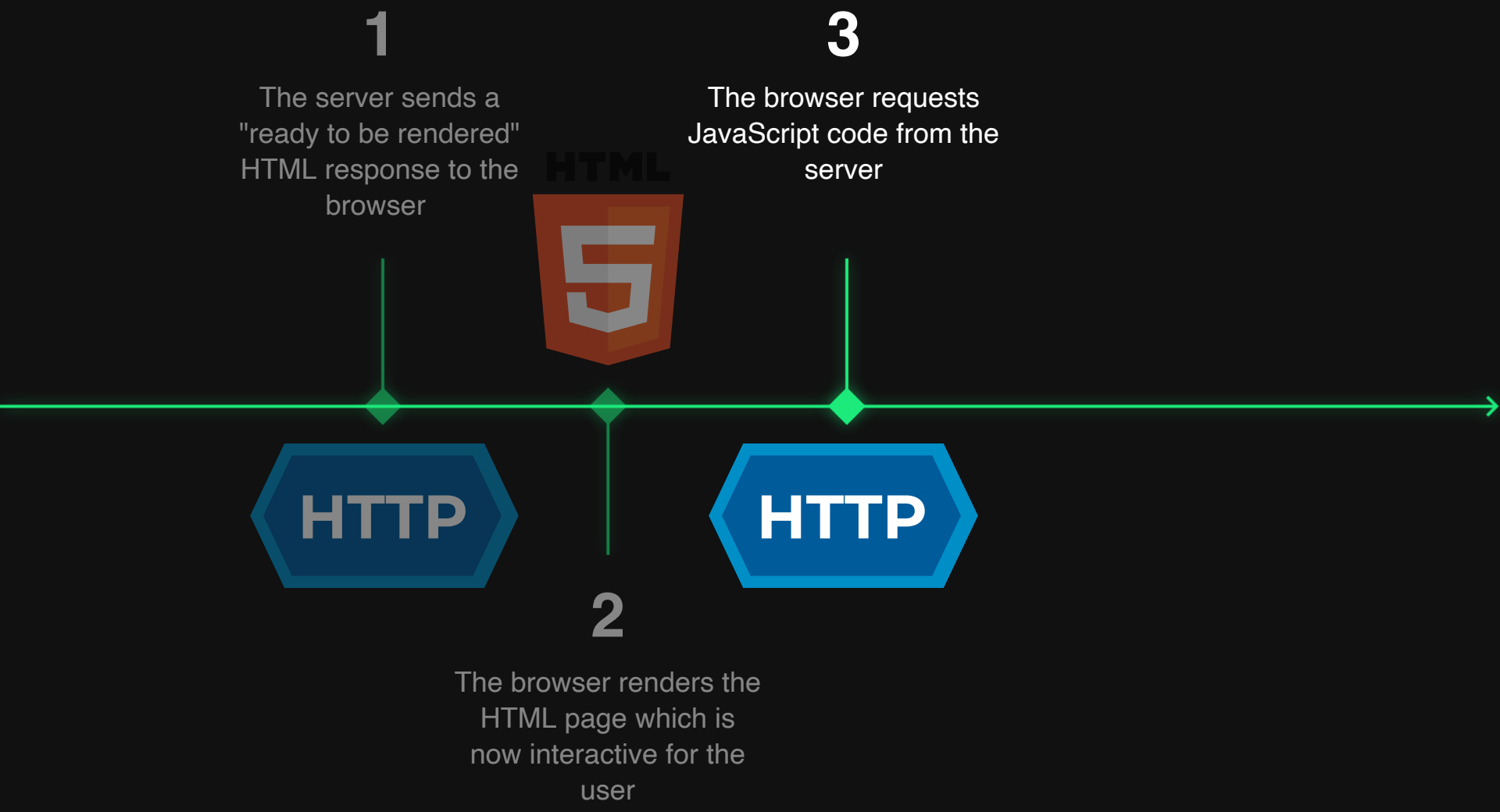
The server sends a "ready to be rendered" HTML response to the browser

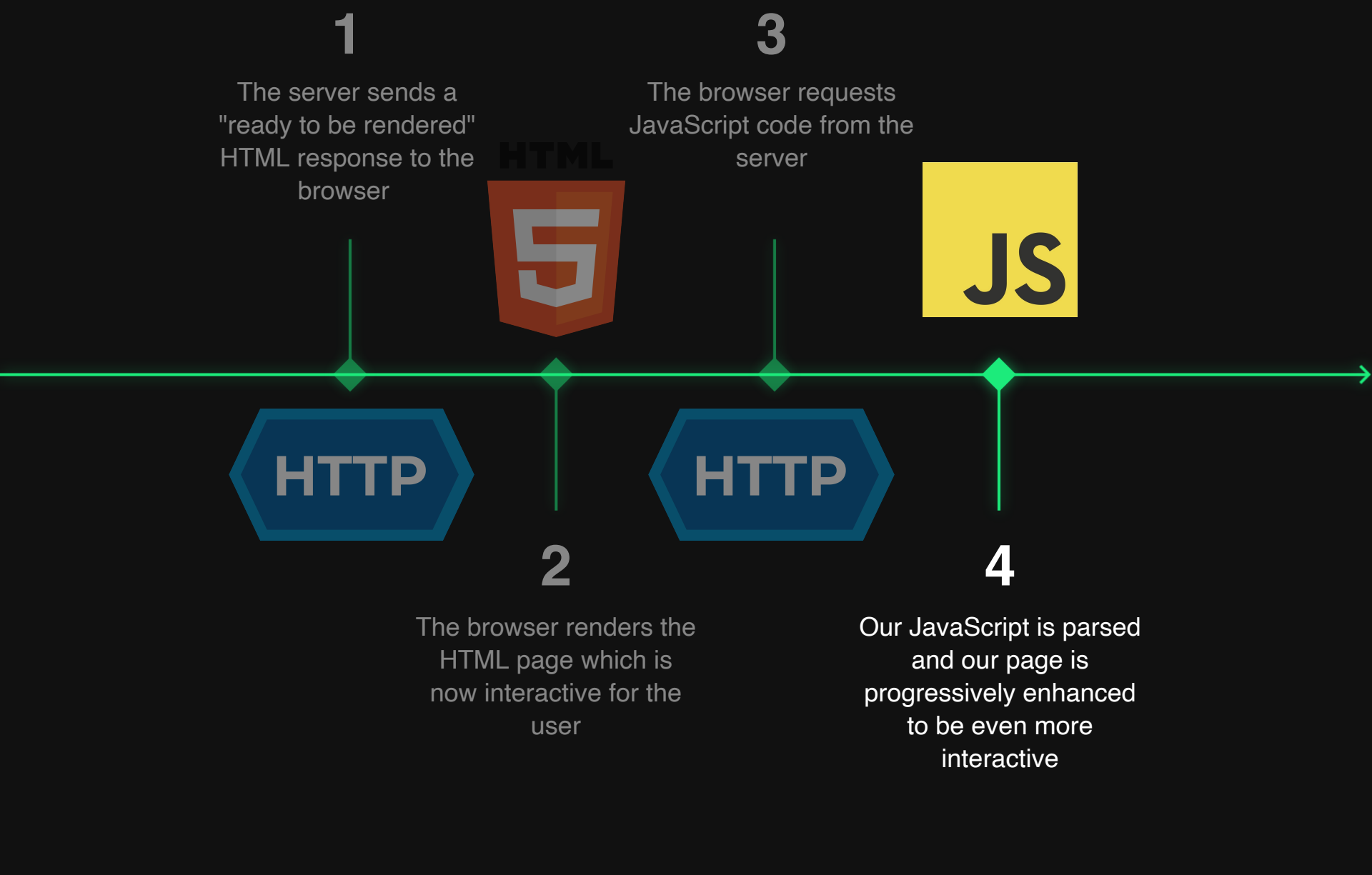
1

The server sends a "ready to be rendered" HTML response to the browser

2

The browser renders the HTML page which is now interactive for the user


1
The server sends a "ready to be rendered" HTML response to the browser
HTML
3
The browser requests JavaScript code from the server
HTTP
HTTP
2
The browser renders the HTML page which is now interactive for the user


1
The server sends a "ready to be rendered" HTML response to the browser
HTML
3
The browser requests JavaScript code from the server
JS
HTTP
HTTP
2
The browser renders the HTML page which is now interactive for the user
4
Our JavaScript is parsed and our page is progressively enhanced to be even more interactive

SSR Benefits

Photo by Rayson Tan on Unsplash

SSR Benefits

- Time to First Interaction is fast

Photo by Rayson Tan on Unsplash

SSR Benefits

- Time to First Interaction is fast
- Better SEO

Photo by Rayson Tan on Unsplash

SSR Benefits

- Time to First Interaction is fast
- Better SEO
- Works on older devices and slow network connections

Photo by Rayson Tan on Unsplash

Isn't SSR slow 🐢?

Isn't SSR slow  ?

No

The first lesson in rendering off the main thread is to leverage your server to do the initial render

Adventure

1.

Reduce

Reduce your JavaScript footprint by server side rendering HTML

2.

Reuse

Compose our application with re-usable components

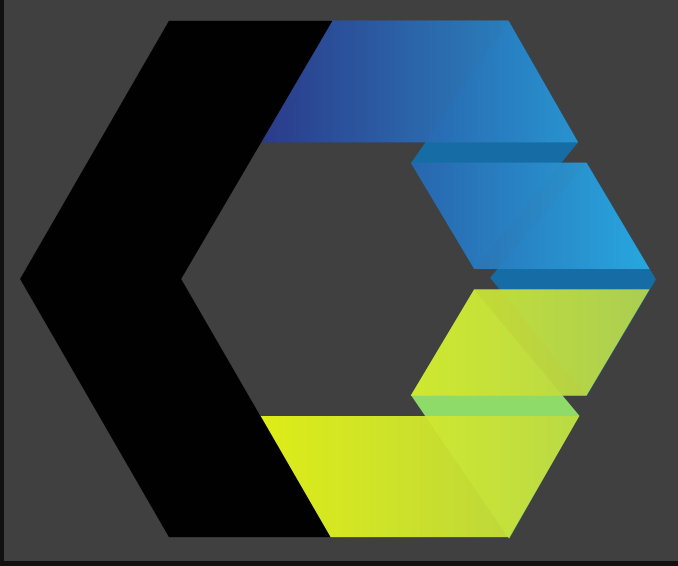
Can I still use a component framework when using SSR?

Yes

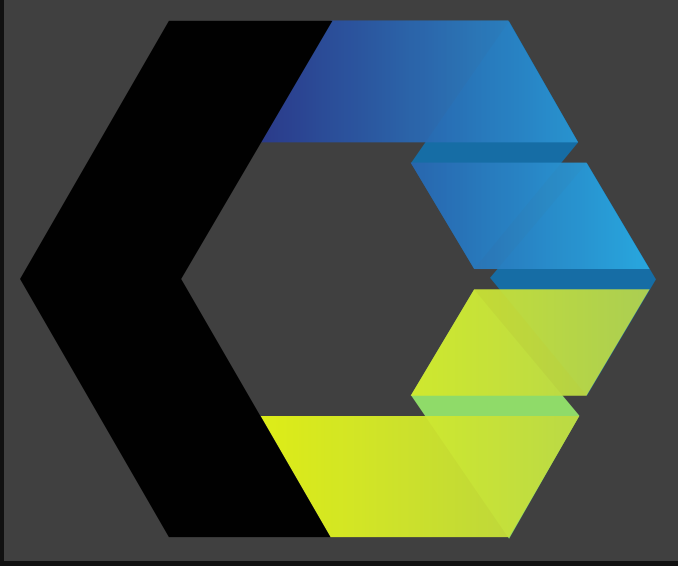
Yes

But...

Why use a framework when Web Components exist?

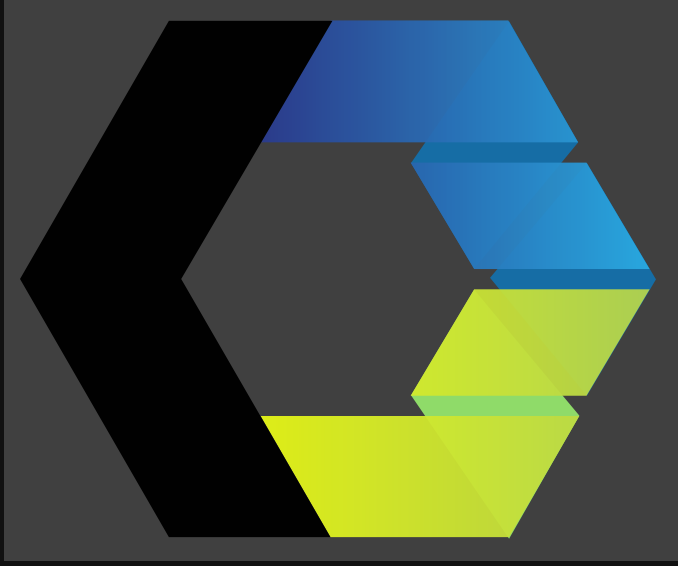


Why use a framework when Web Components exist?



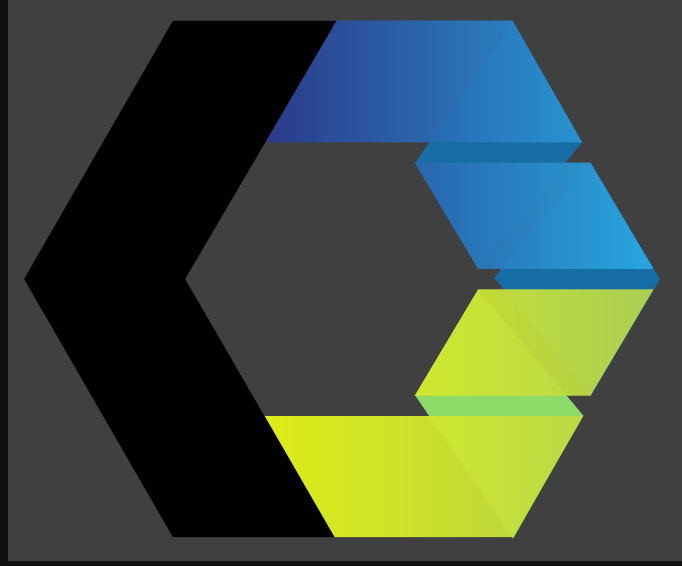
- Custom Elements ➡️ define components

Why use a framework when Web Components exist?



- Custom Elements ➡️ define components
- Shadow DOM ➡️ encapsulate styles

Why use a framework when Web Components exist?



- Custom Elements ➡️ define components
- Shadow DOM ➡️ encapsulate styles
- Slots and Templates ➡️ re-use

Hello World Web Component

```
const template = document.createElement('template');
template.innerHTML = `
  <h1>
    Hello <slot name="name"></slot>
  </h1>`;


class HelloWorld extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.appendChild(template.content.cloneNode(true));
  }
}

customElements.define('hello-world', HelloWorld);
```

Hello World Web Component

```
const template = document.createElement('template');
template.innerHTML = `
  <h1>
    Hello <slot name="name"></slot>
  </h1>`;


class HelloWorld extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.appendChild(template.content.cloneNode(true));
  }
}

customElements.define('hello-world', HelloWorld);
```

Hello World Web Component

```
const template = document.createElement('template');
template.innerHTML = `
  <h1>
    Hello <slot name="name"></slot>
  </h1>`;


class HelloWorld extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.appendChild(template.content.cloneNode(true));
  }
}

customElements.define('hello-world', HelloWorld);
```

Hello World Web Component

```
const template = document.createElement('template');
template.innerHTML = `
  <h1>
    Hello <slot name="name"></slot>
  </h1>`;


class HelloWorld extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.appendChild(template.content.cloneNode(true));
  }
}

customElements.define('hello-world', HelloWorld);
```

Hello World Web Component

```
1 <hello-world>
2   <span slot="name">
3     Prairie Dev Con
4   </span>
5 </hello-world>
6 <hello-world>
7   <span slot="name">Calgary</spa
8 </hello-world>
```

Hello Prairie
Dev Con
Hello
Calgary

Aren't you contradicting yourself? Web components need JavaScript, right?

WELCOME TO

ENHANCE

https://enhance.dev

WHAT IS ENHANCE?

Enhance is a web standards-based HTML framework. It's designed to provide a dependable foundation for building lightweight, flexible, and future-proof web applications.

Author

Enhance allows developers to write components as pure functions that return HTML. Then render them on the server to deliver complete HTML immediately available to the end user.

Standards

Enhance takes care of the tedious parts, allowing you to use today's Web Platform standards more efficiently. As new standards and platform features become generally available, Enhance will make way for them.

Progressive

Enhance allows for easy progressive enhancement so that working HTML can be further developed to add additional functionality with JavaScript.

Enhance Hello-World

https://codepen.io/macdonst/embed/zYeLYJP?default-tab=html%2Cresult&editable=true

Benefits of enhance

Benefits of enhance

All components are server side renderable with no changes.

Benefits of enhance

All components are server side renderable with no changes.

Leverage your CSS skills and style encapsulation in the light DOM.

Benefits of enhance

All components are server side renderable with no changes.

Leverage your CSS skills and style encapsulation in the light DOM.

Use slots without the shadow DOM

Benefits of enhance

All components are server side renderable with no changes.

Leverage your CSS skills and style encapsulation in the light DOM.

Use slots without the shadow DOM

Avoid excessive JavaScript code by leveraging the platform and web standards

The second lesson in rendering off the main thread is to reuse what the browser already provides

Adventure

1.

Reduce

Reduce your JavaScript footprint by server side rendering HTML

2.

Reuse

Use the platform to provide a reusable component architecture with Web Components

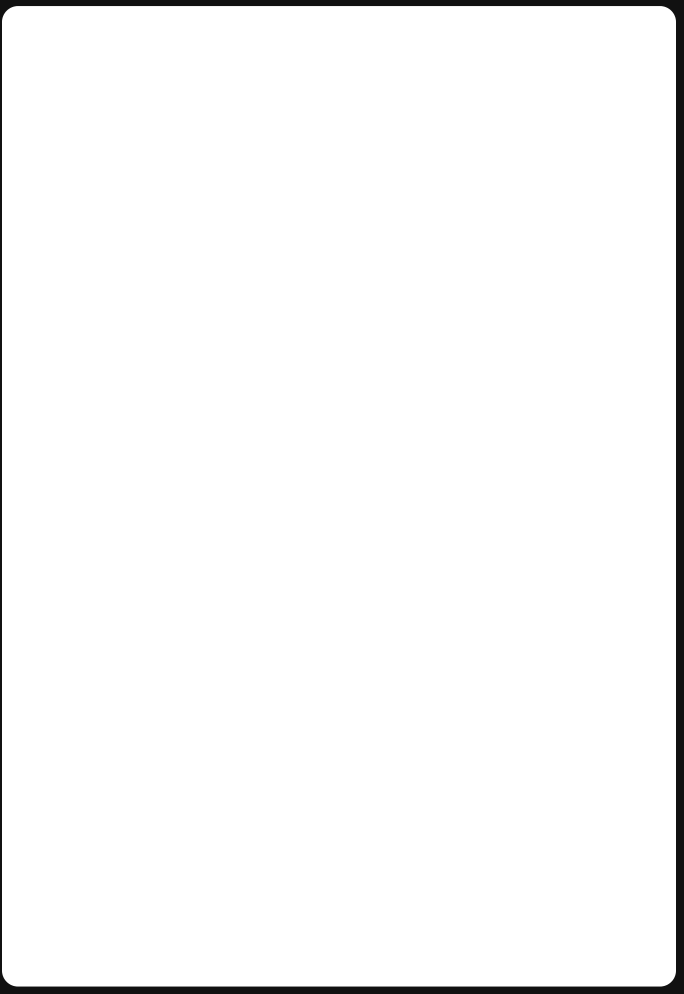
3.

~~Recycle~~ Offload

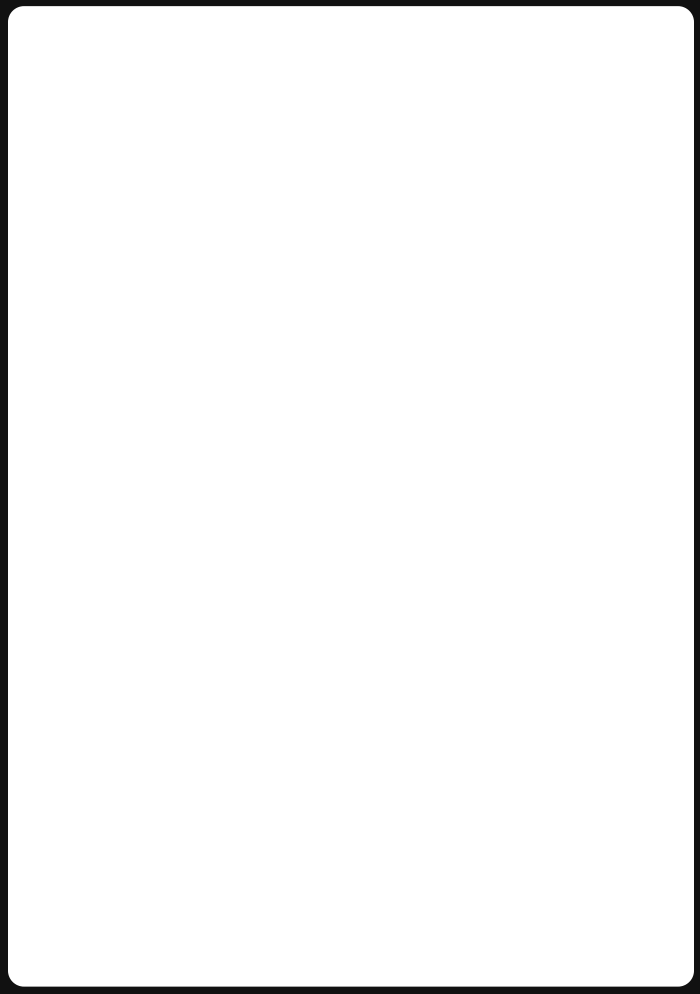
Expensive or time consuming tasks to web workers

Web Workers

“ Web Workers makes it possible to run a script operation in a background thread separate from the main execution thread of a web application.



Main Thread



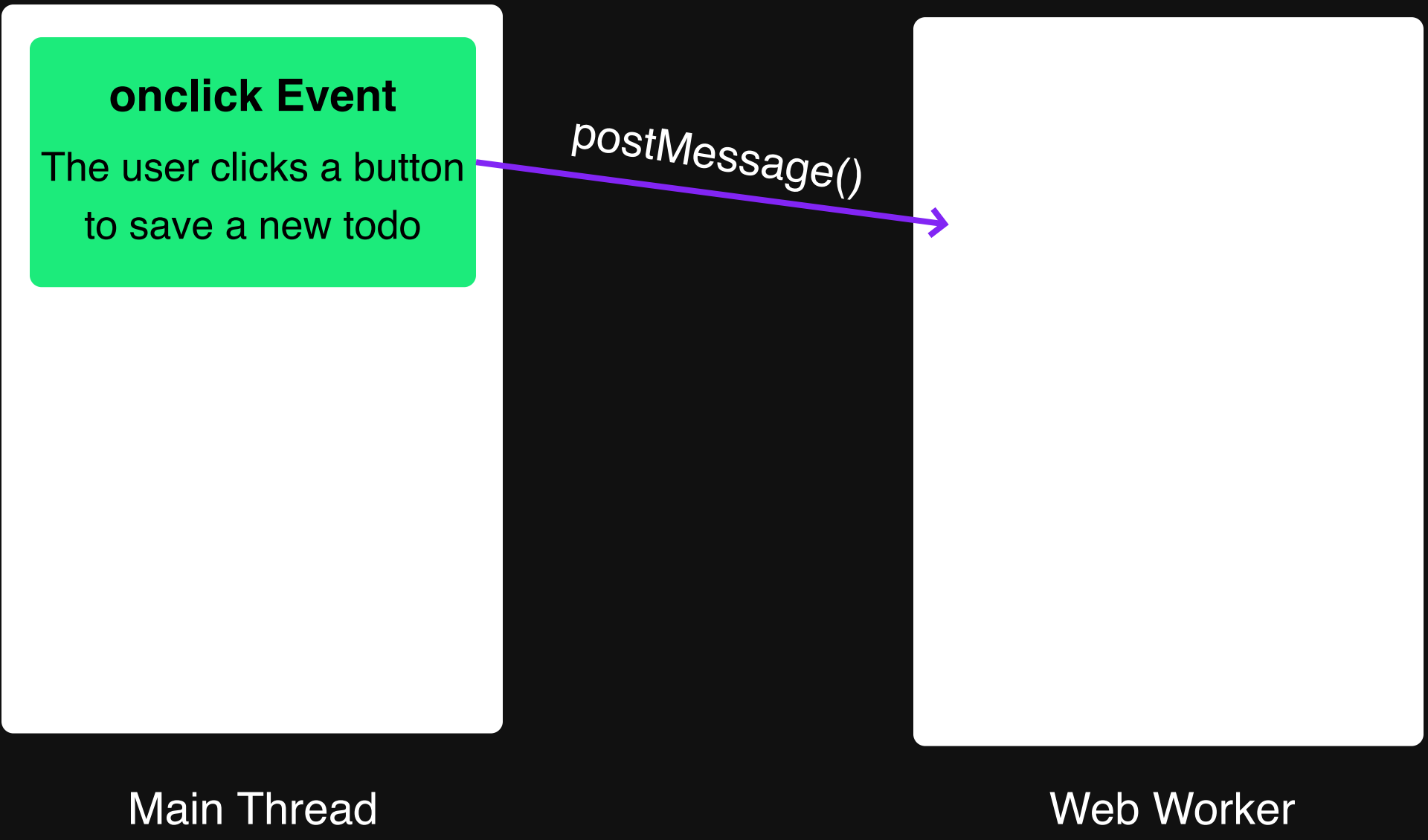
Web Worker

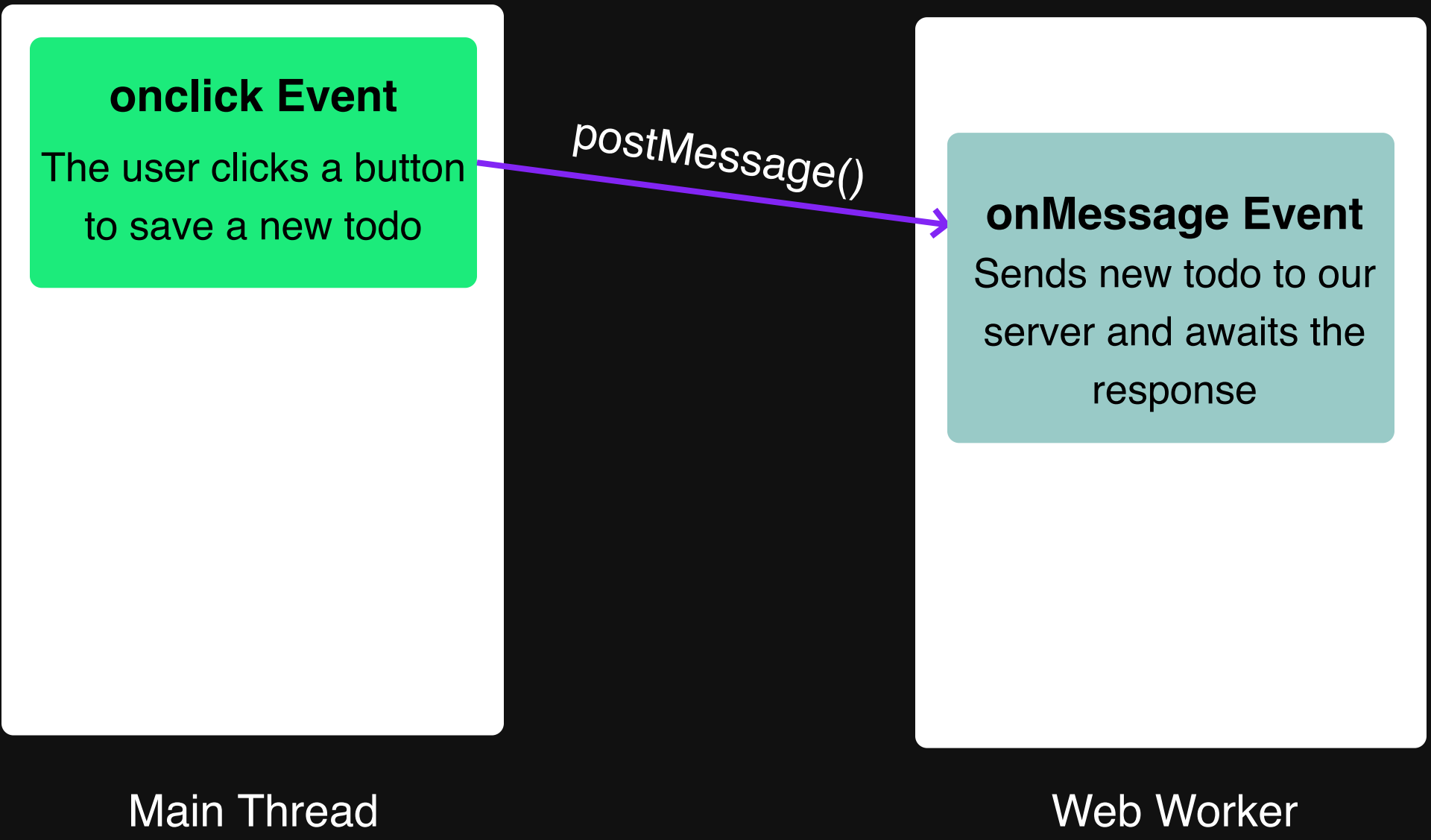
onclick Event

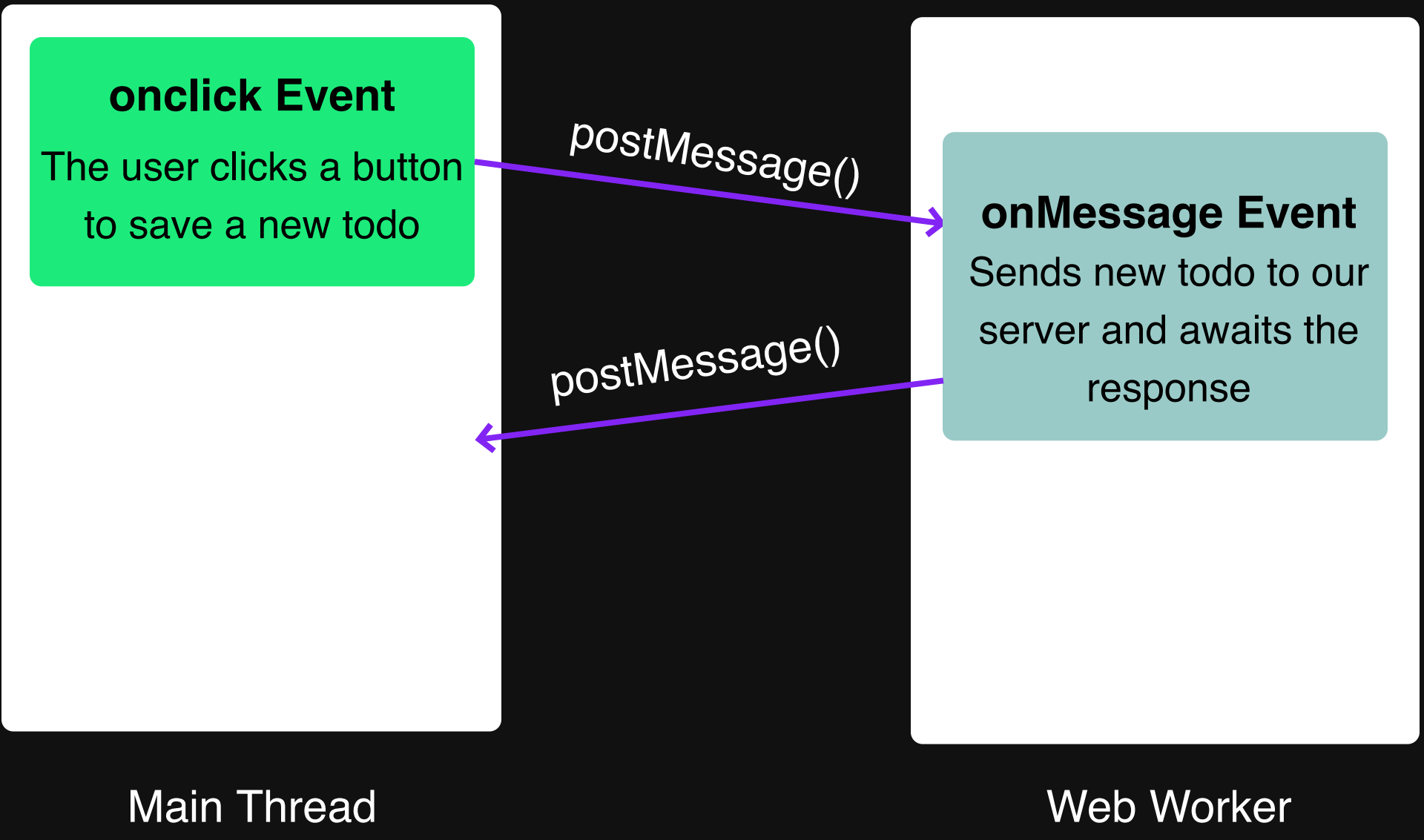
The user clicks a button to save a new todo

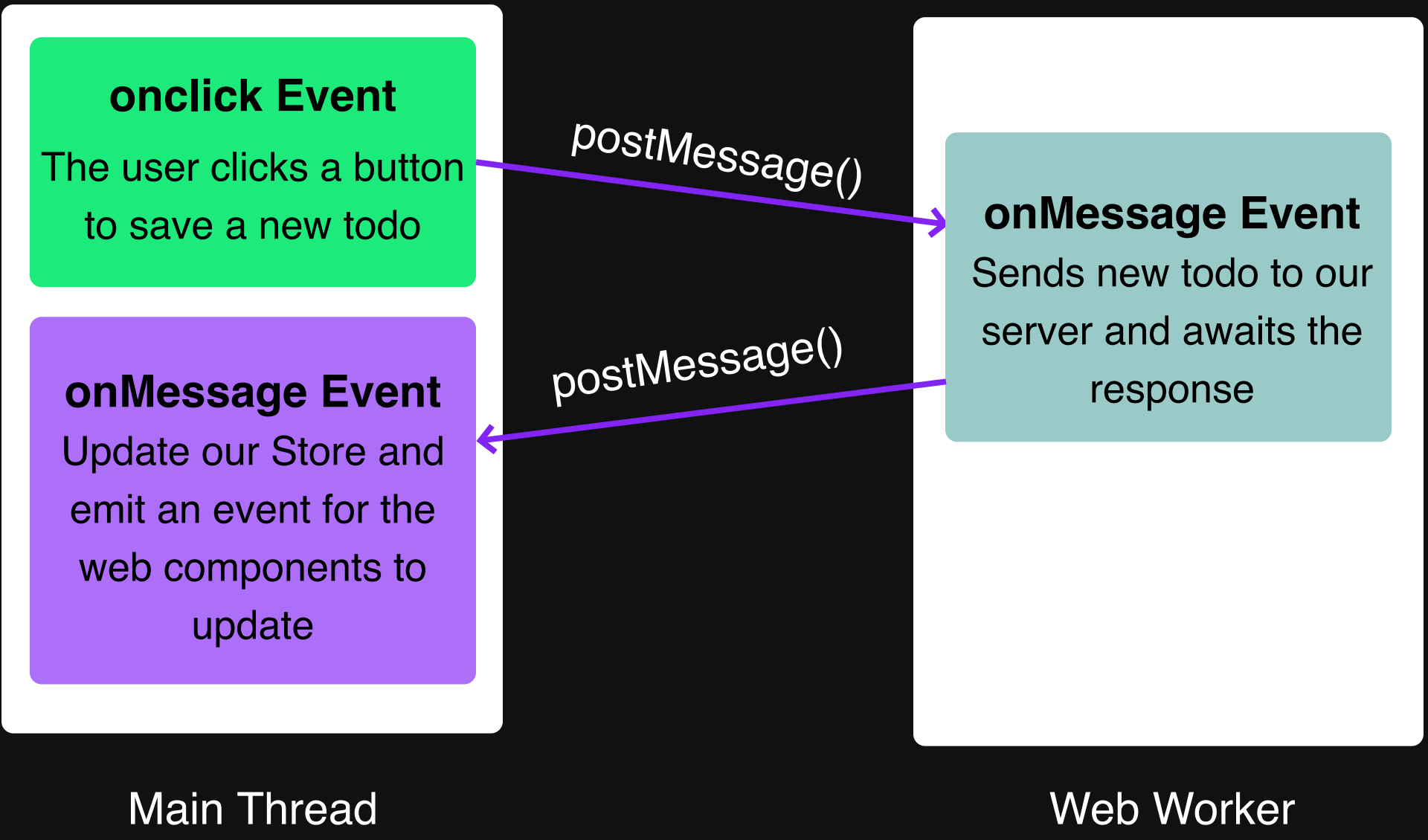
Main Thread

Web Worker


onclick Event
The user clicks a button to save a new todo
postMessage()
Main Thread
Web Worker


onclick Event
The user clicks a button to save a new todo
postMessage()
onMessage Event
Sends new todo to our server and awaits the response
Main Thread
Web Worker


onclick Event
The user clicks a button to save a new todo
postMessage()
onMessage Event
Sends new todo to our server and awaits the response
postMessage()
Main Thread
Web Worker


onclick Event
The user clicks a button to save a new todo
postMessage()
onMessage Event
Sends new todo to our server and awaits the response
onMessage Event
Update our Store and emit an event for the web components to update
postMessage()
Main Thread
Web Worker

Todo API

```
// Todo API
let worker
export default function saveTodo(todo) {
  if (!worker) {
    worker = new Worker('worker.mjs')
    worker.onmessage = mutate
  }

  worker.postMessage({
    data: todo
  })
}

const store = Store()
function mutate(result) {
  const copy = Array.from(store.todos)
```

Todo API

```
// Todo API
let worker
export default function saveTodo(todo) {
  if (!worker) {
    worker = new Worker('worker.mjs')
    worker.onmessage = mutate
  }

  worker.postMessage({
    data: todo
  })
}

const store = Store()
function mutate(result) {
  const copy = Array.from(store.todos)
```

Todo API

```
let worker
export default function saveTodo(todo) {
  if (!worker) {
    worker = new Worker('worker.mjs')
    worker.onmessage = mutate
  }

  worker.postMessage({
    data: todo
  })
}

const store = Store()
function mutate(result) {
  const copy = Array.from(store.todos)
  copy.push(result)
  store.todos = copy
```

Web Worker

```
// worker.js
self.onmessage = async function message({ data }) {
  try {
    const result = await (await fetch(
      `/todos/${data.key}`, {
          body: payload,
          credentials: 'same-origin',
          headers: {
            'Content-Type': 'application/json'
          },
          method: 'POST'
        }
    )).json()

    self.postMessage(result)
```

Web Worker

```
// worker.js
self.onmessage = async function message({ data }) {
  try {
    const result = await (await fetch(
      `/todos/${data.key}`, {
          body: payload,
          credentials: 'same-origin',
          headers: {
            'Content-Type': 'application/json'
          },
          method: 'POST'
        }
    )).json()

    self.postMessage(result)
  }
  catch (err) {
```

Web Worker

```
    const result = await (await fetch(
      `/todos/${data.key}`, {
          body: payload,
          credentials: 'same-origin',
          headers: {
            'Content-Type': 'application/json'
          },
          method: 'POST'
        }
    )).json()

    self.postMessage(result)
  }
  catch (err) {
    console.error(err)
  }
}
```

Todo API

```
// Todo API
let worker
export default function saveTodo(todo) {
  if (!worker) {
    worker = new Worker('worker.mjs')
    worker.onmessage = mutate
  }

  worker.postMessage({
    data: todo
  })
}

const store = Store()
function mutate(result) {
  const copy = Array.from(store.todos)
```

Store API

```
// store.js
function subscribe (fn, props) {
  return listeners.push(fn)
}
function unsubscribe (fn) {
  return listeners.splice(listeners.indexOf(fn), 1)
}
_state.subscribe = subscribe
_state.unsubscribe = unsubscribe
function notify () {
  listeners.forEach(fn => {
    fn(payload)
  })
}
const handler = {
  set: function (obj, prop, value) {
```

Store API

```
// store.js
function subscribe (fn, props) {
  return listeners.push(fn)
}
function unsubscribe (fn) {
  return listeners.splice(listeners.indexOf(fn), 1)
}
_state.subscribe = subscribe
_state.unsubscribe = unsubscribe
function notify () {
  listeners.forEach(fn => {
    fn(payload)
  })
}
const handler = {
  set: function (obj, prop, value) {
```

Store API

```
  }
  function unsubscribe (fn) {
    return listeners.splice(listeners.indexOf(fn), 1)
  }
  _state.subscribe = subscribe
  _state.unsubscribe = unsubscribe
  function notify () {
    listeners.forEach(fn => {
      fn(payload)
    })
  }
  const handler = {
    set: function (obj, prop, value) {
      if (obj[prop] !== value) {
        obj[prop] = value
        set(notify)
      }
```

Store API

```
_state.subscribe = subscribe
_state.unsubscribe = unsubscribe
function notify () {
  listeners.forEach(fn => {
    fn(payload)
  })
}
const handler = {
  set: function (obj, prop, value) {
    if (obj[prop] !== value) {
      obj[prop] = value
      set(notify)
    }
    return true
  }
}
const store = new Proxy(_state, handler)
```

Store API

```
_state.subscribe = subscribe
_state.unsubscribe = unsubscribe
function notify () {
  listeners.forEach(fn => {
    fn(payload)
  })
}
const handler = {
  set: function (obj, prop, value) {
    if (obj[prop] !== value) {
      obj[prop] = value
      set(notify)
    }
    return true
  }
}
const store = new Proxy(_state, handler)
```

Todo List Web Component

```
// todo-list.js
import CustomElement from "@enhance/custom-element"
class TodosList extends CustomElement {
  constructor () {
    super()
    this.api = API()
    this.update = this.update.bind(this)
  }

  connectedCallback () {
    this.api.subscribe(this.update, [ 'todos' ])
  }

  disconnectedCallback () {
    this.api.unsubscribe(this.update)
  }
```

Todo List Web Component

```
// todo-list.js
import CustomElement from "@enhance/custom-element"
class TodosList extends CustomElement {
  constructor () {
    super()
    this.api = API()
    this.update = this.update.bind(this)
  }

  connectedCallback () {
    this.api.subscribe(this.update, [ 'todos' ])
  }

  disconnectedCallback () {
    this.api.unsubscribe(this.update)
  }
```

Todo List Web Component

```
class TodosList extends CustomElement {
  constructor () {
    super()
    this.api = API()
    this.update = this.update.bind(this)
  }

  connectedCallback () {
    this.api.subscribe(this.update, [ 'todos' ])
  }

  disconnectedCallback () {
    this.api.unsubscribe(this.update)
  }

  update ({ todos }) {
    const activeTodos = todos.filter(t => !t.completed)
```

Todo List Web Component

```
    this.update = this.update.bind(this)
  }

  connectedCallback () {
    this.api.subscribe(this.update, [ 'todos' ])
  }

  disconnectedCallback () {
    this.api.unsubscribe(this.update)
  }

  update ({ todos }) {
    const activeTodos = todos.filter(t => !t.completed)
    const completedTodos = todos.filter(t => t.completed)
    this.activeTodosList.todos = activeTodos
    this.completedTodosList.todos = completedTodos
  }
```

Todo List Web Component

```
  }

  connectedCallback () {
    this.api.subscribe(this.update, [ 'todos' ])
  }

  disconnectedCallback () {
    this.api.unsubscribe(this.update)
  }

  update ({ todos }) {
    const activeTodos = todos.filter(t => !t.completed)
    const completedTodos = todos.filter(t => t.completed)
    this.activeTodosList.todos = activeTodos
    this.completedTodosList.todos = completedTodos
  }
}
```

The third lesson in rendering off the main thread is to use web workers for expensive operations after you've mastered the first two lessons!

In conclusion…

In conclusion…

Reduce: the amount of JavaScript required to build your HTML

In conclusion…

Reduce: the amount of JavaScript required to build your HTML

Reuse: what the platform already provides

In conclusion…

Reduce: the amount of JavaScript required to build your HTML

Reuse: what the platform already provides

Offload: expensive or time consuming tasks to web workers

Use JavaScript

Not a lot

Mostly for progressive enhancement

h/t Michael Pollan

ENHANCE

The HTML framework

Read the Docs:	enhance.dev/docs
Join our Discord:	enhance.dev/discord
Follow us on Mastodon:	fosstodon.org/@enhance_dev


Thanks!