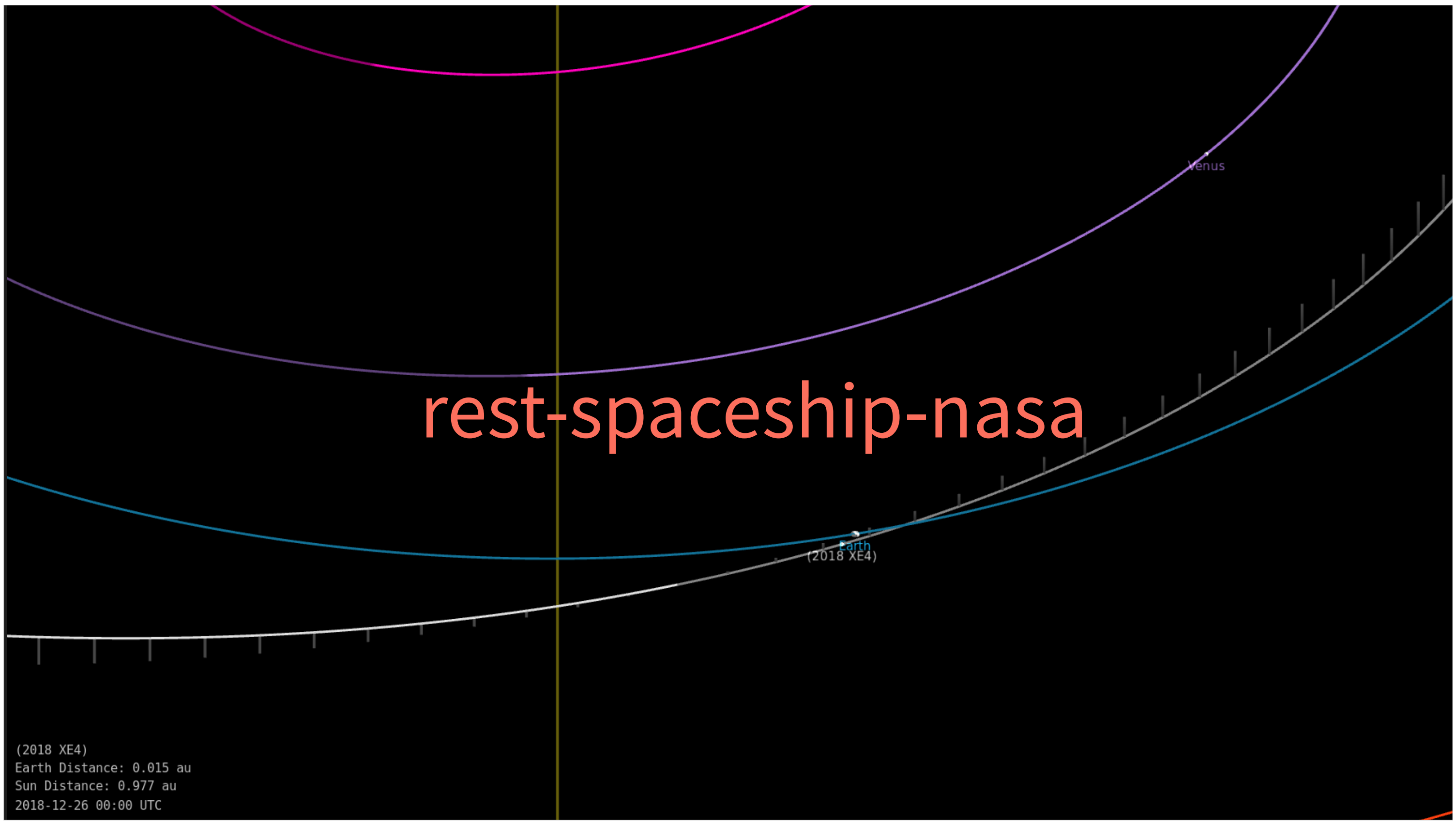
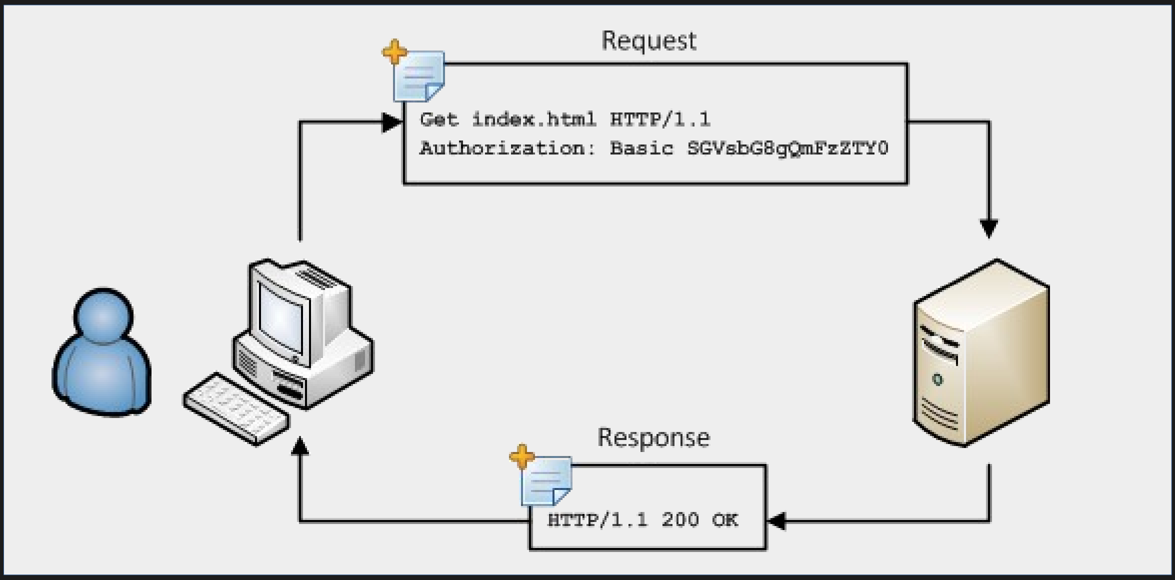
Contemporary Web Development
Lesson 7

https://www.youtube.com/user/doddlevloggle



rest-spaceship-nasa

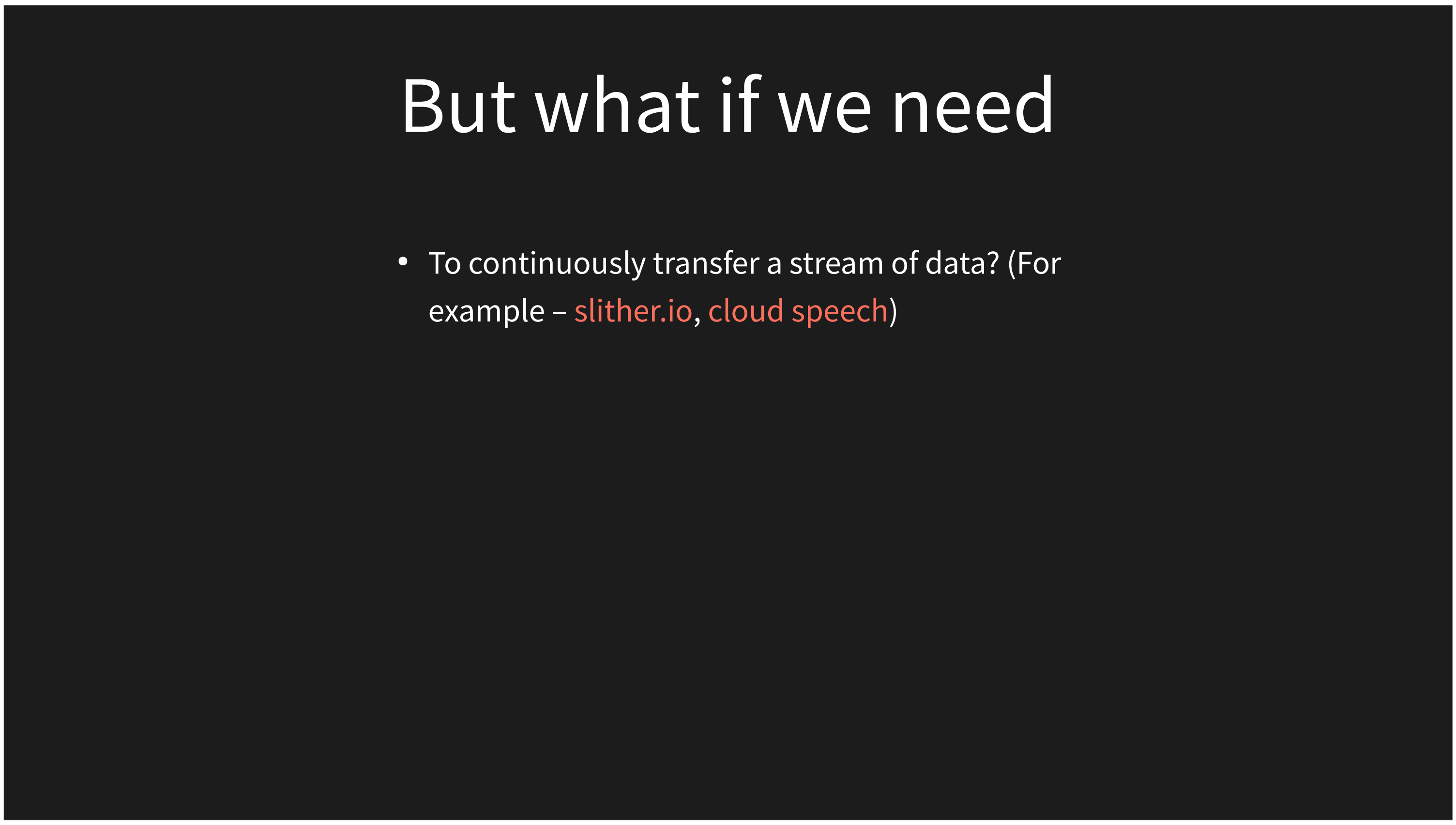
A REST Request



```
Get index.html HTTP/1.1
Authorization: Basic SGVsbG8gQmFzZTY0
```

```
HTTP/1.1 200 OK
```

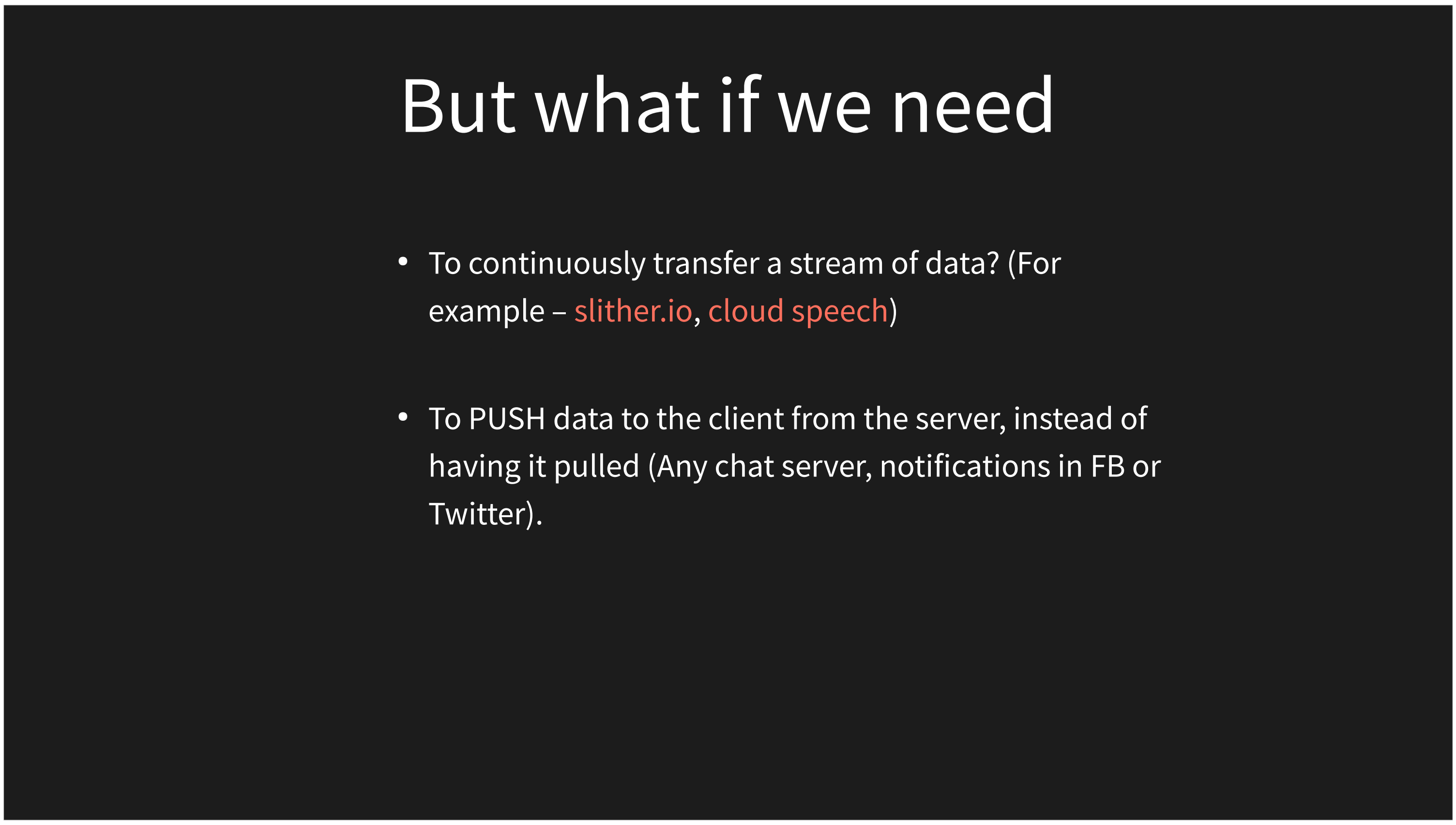
The client / user is **pulling** data from the server. A connection is made and released every time there is a need for data.



But what if we need

- To continuously transfer a stream of data? (For example – slither.io, cloud speech)

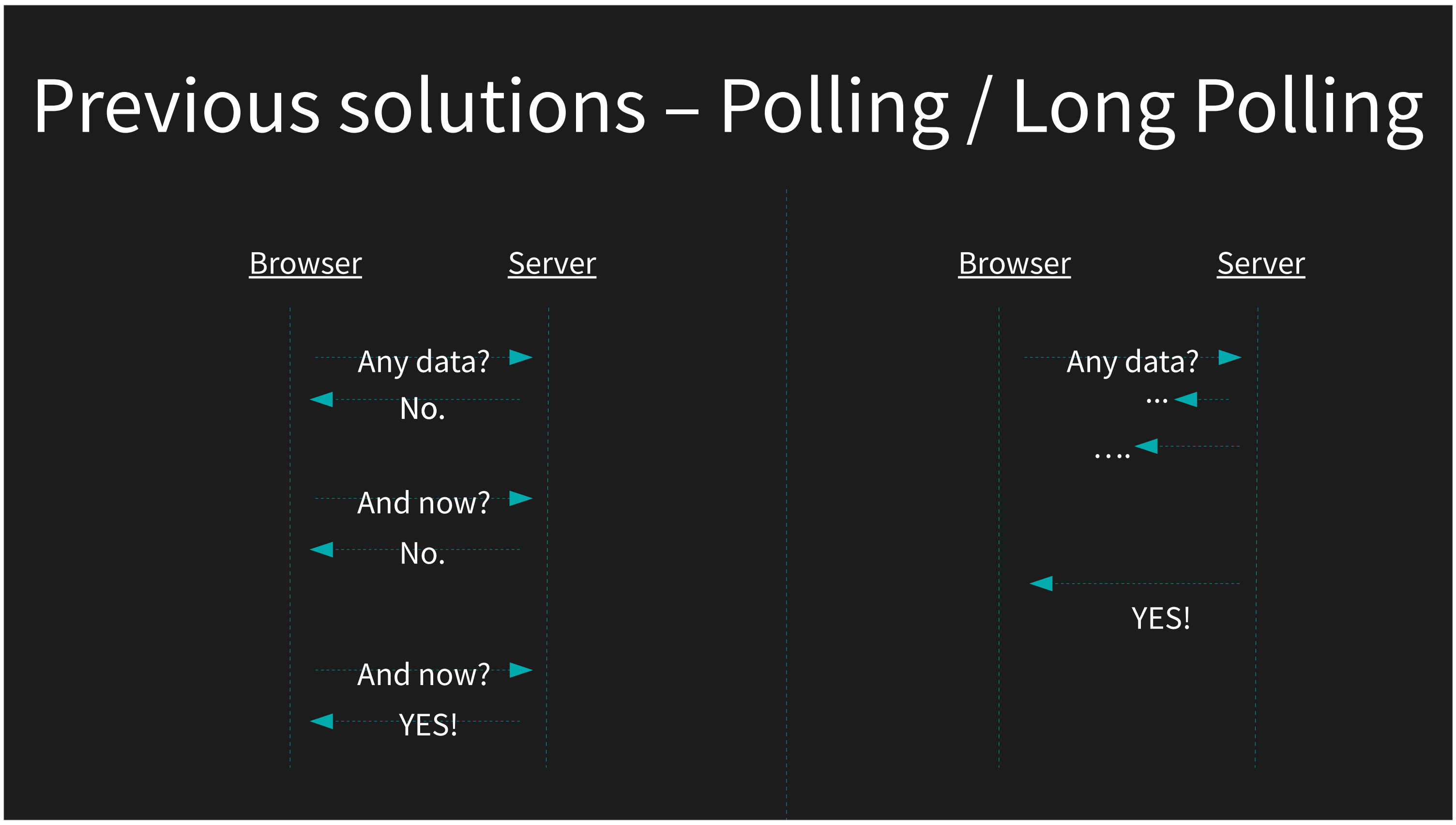
A long POST request? That’s not what the HTTP protocol was designed for.



But what if we need

- To continuously transfer a stream of data? (For example – slither.io, cloud speech)
- To PUSH data to the client from the server, instead of having it pulled (Any chat server, notifications in FB or Twitter).

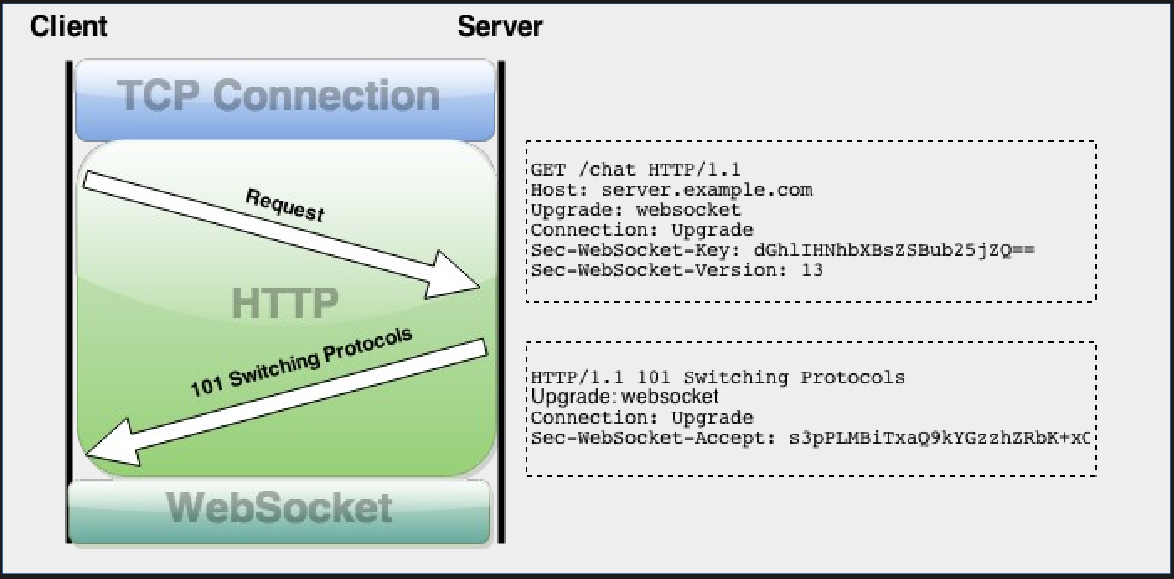
The client / user is **pulling** data from the server. A connection is made and released every time there is a need for data.



Previous solutions – Polling / Long Polling

Against this is against the standard HTTP. An HTTP web server is not meant to hold too many parallel connections.

Web Sockets

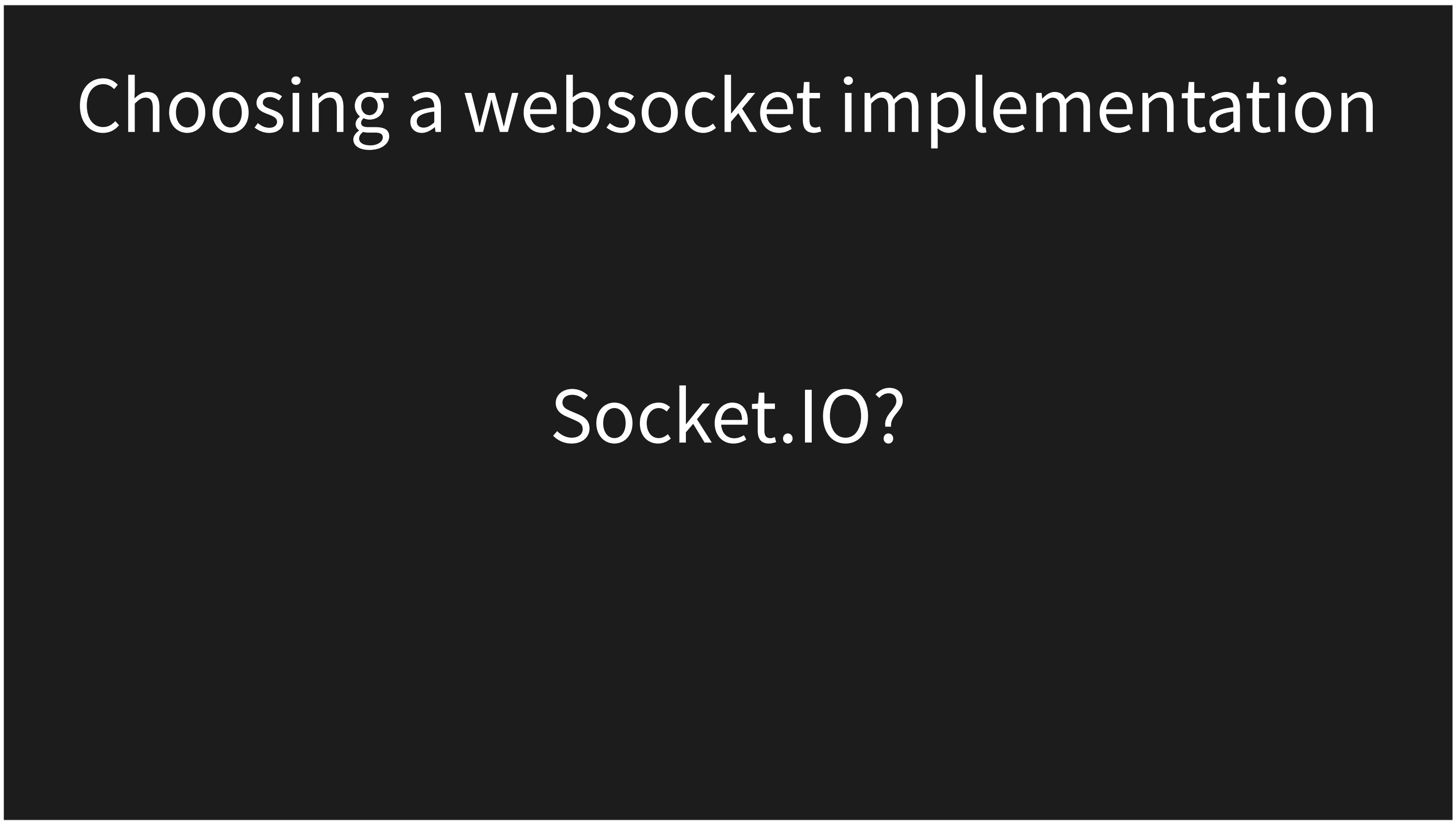


```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xO
```

The key is in this **Upgrade** request. After that, the connection is held and frames are being passed through the websocket protocol.

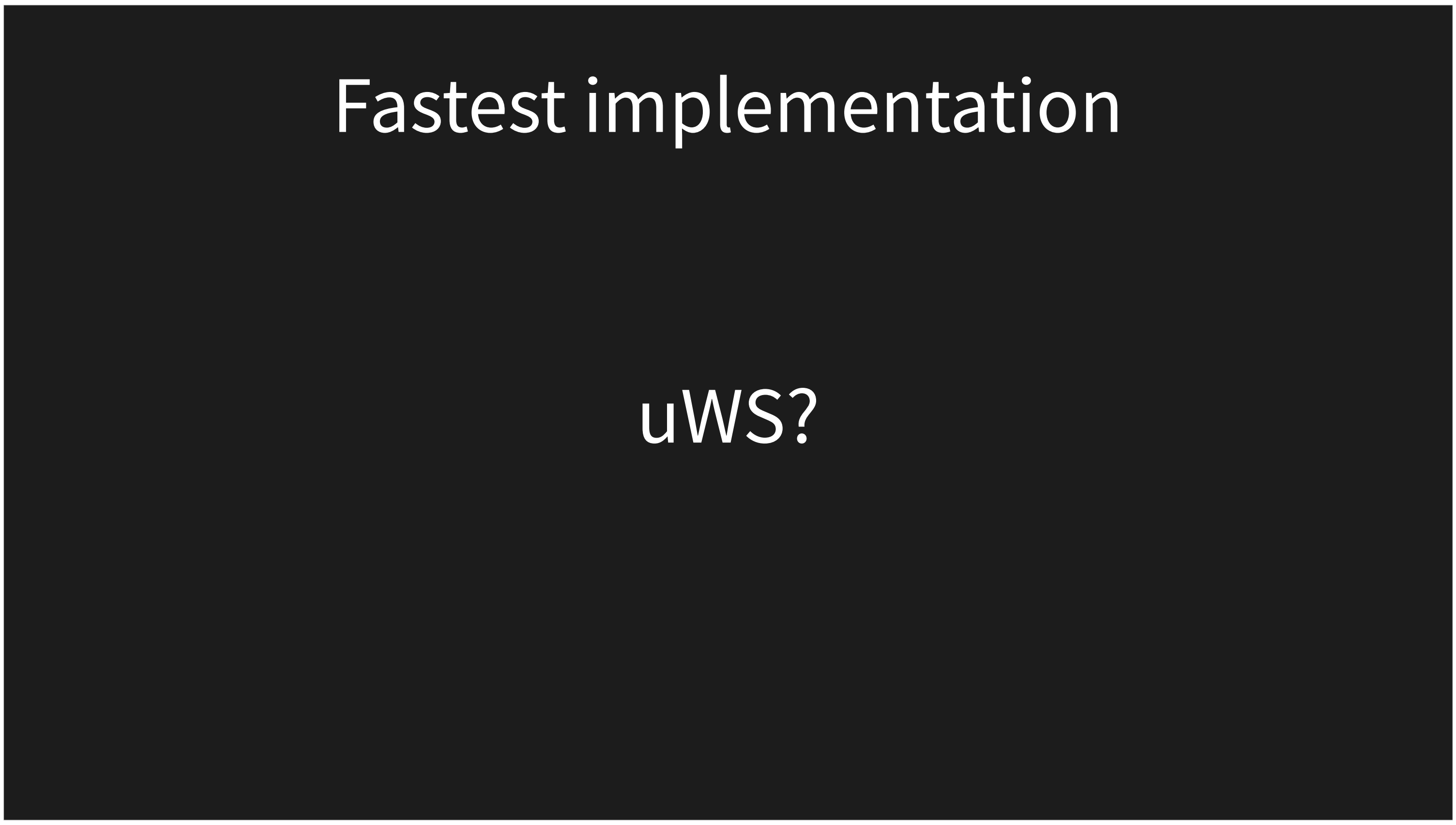
Example App - Collaborative Synthesizer



Choosing a websocket implementation

Socket.IO?

Socket.IO is the most popular implementation, but it also introduces a lot of overhead.

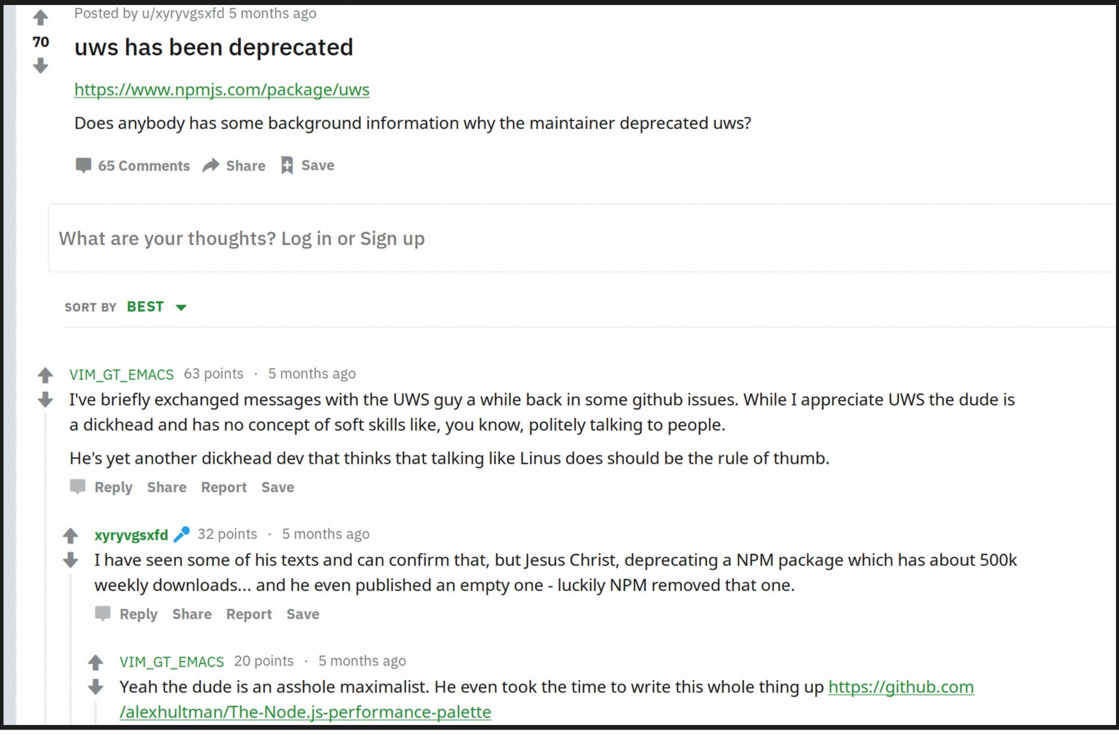


Fastest implementation

uWS?

Socket.IO is supposedly based on uWS (replaced engine.io), but with a lot of overhead. When I searched for more information I found some npm drama.

NPM Drama


70

Posted by u/xyryvgsxfd 5 months ago

uws has been deprecated

https://www.npmjs.com/package/uws

Does anybody has some background information why the maintainer deprecated uws?

VIM_GT_EMACS 63 points · 5 months ago

I've briefly exchanged messages with the UWS guy a while back in some github issues. While I appreciate UWS the dude is a dickhead and has no concept of soft skills like, you know, politely talking to people.

He's yet another dickhead dev that thinks that talking like Linus does should be the rule of thumb.

xyryvgsxfd 32 points · 5 months ago

I have seen some of his texts and can confirm that, but Jesus Christ, deprecating a NPM package which has about 500k weekly downloads... and he even published an empty one - luckily NPM removed that one.

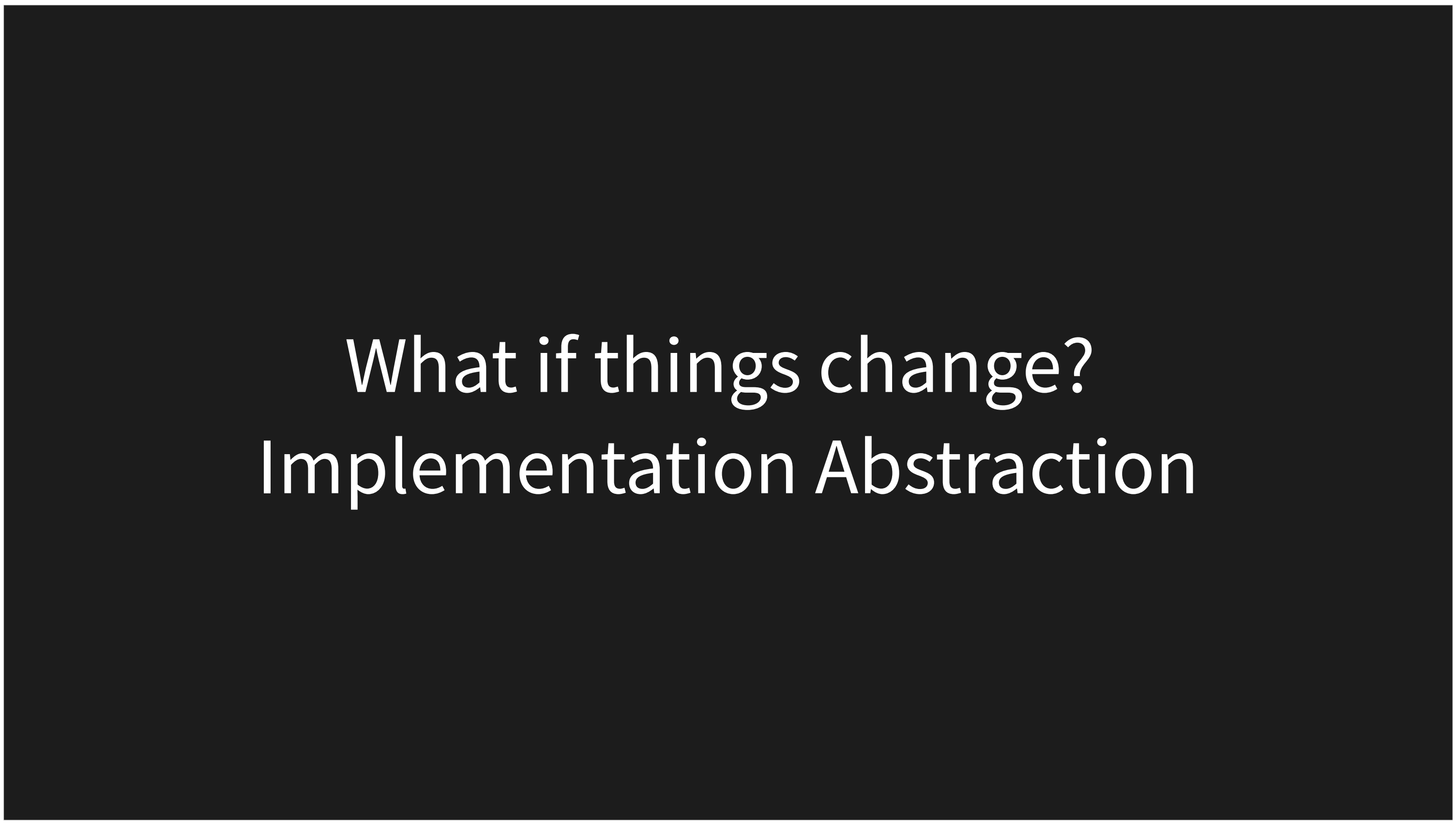
VIM_GT_EMACS 20 points · 5 months ago

Yeah the dude is an asshole maximalist. He even took the time to write this whole thing up https://github.com/alexhultman/The-Node.js-performance-palette

It’s still being continued though.
And is being sponsored by a bitcoin company.

Drama-less implementation

WS

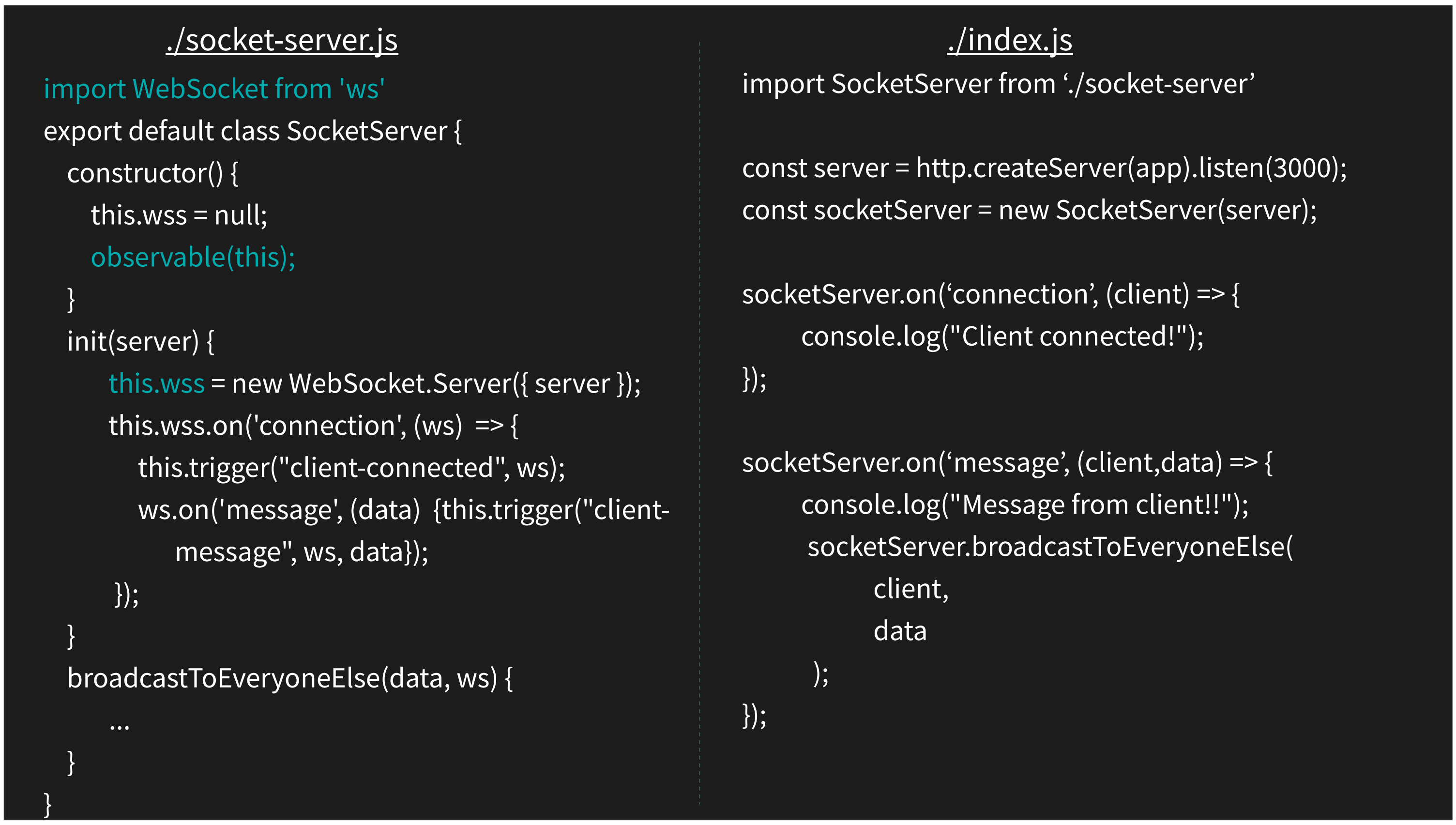


What if things change? Implementation Abstraction

Whatever library we chose, we want to abstract it so that it can’t be changed easily. We essentially build proxy objects,

The OOP Approach : Encapsulation

In the OOP approach we create an object/class that handles the socket opertions, and that object also has a state.



./socket-server.js

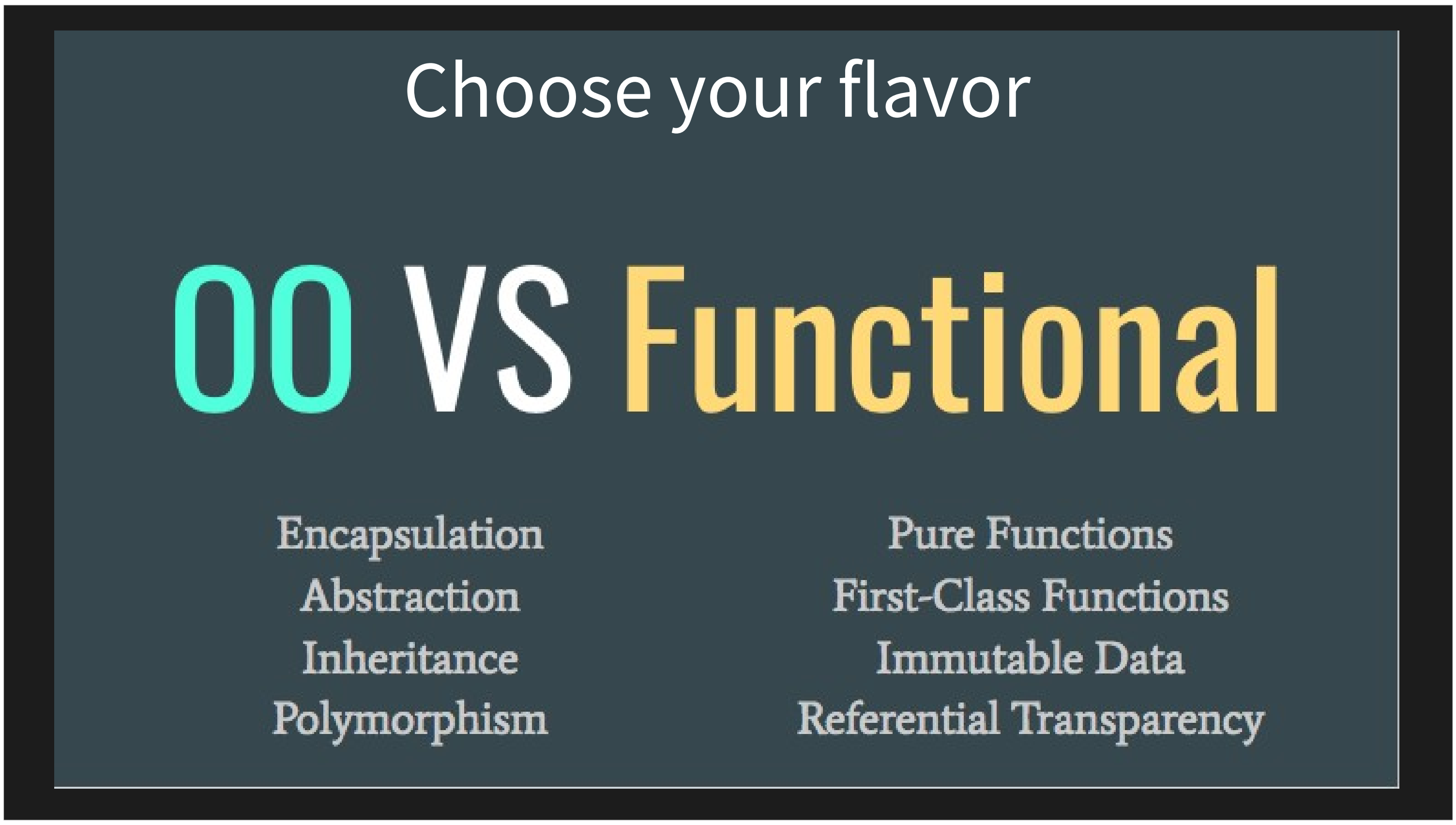
```
import WebSocket from 'ws'
export default class SocketServer {
  constructor() {
    this.wss = null;
    observable(this);
  }
  init(server) {
     this.wss = new WebSocket.Server({ server });
     this.wss.on('connection', (ws)  => {
        this.trigger("client-connected", ws);
        ws.on('message', (data)  {this.trigger("client-
            message", ws, data});
     });
  }
  broadcastToEveryoneElse(data, ws) {
     ...
  }
}
```

./index.js

```
import SocketServer from '‘./socket-server’

const server = http.createServer(app).listen(3000);
const socketServer = new SocketServer(server);

socketServer.on(‘connection’, (client) => {
      console.log("Client connected!");
});

socketServer.on(‘message’, (client,data) => {
      console.log("Message from client!!");
       socketServer.broadcastToEveryoneElse(
             client,
             data
       );
});
```

Let’s try more a functional approach.

People who argue against OOP claim that you shouldn’t mix between data and function. Data goes into functions and is manipulated by them. It’s **arguably** easier to maintain/debug ,results in less code and is faster.


Choose your flavor

OO VS Functional

Encapsulation	Pure Functions
Abstraction	First-Class Functions
Inheritance	Immutable Data
Polymorphism	Referential Transparency

In the OOP approach we create an object/class that handles the socket opertions, and that object also has a state.

Going Functional on the server side

./socket-server.js

```
import WebSocket from 'ws'
export function init(server, messageHandler) {
  const wss = new WebSocket.Server({ server });
  wss.on('connection', (ws)  => {
    ws.on('message', (data) => {
      messageHandler(data, {client: ws,  server: wss})})
  });
}
export function broadcastToEveryoneElse(data,
context) {
      ...
   }
  });
}
```

./index.js

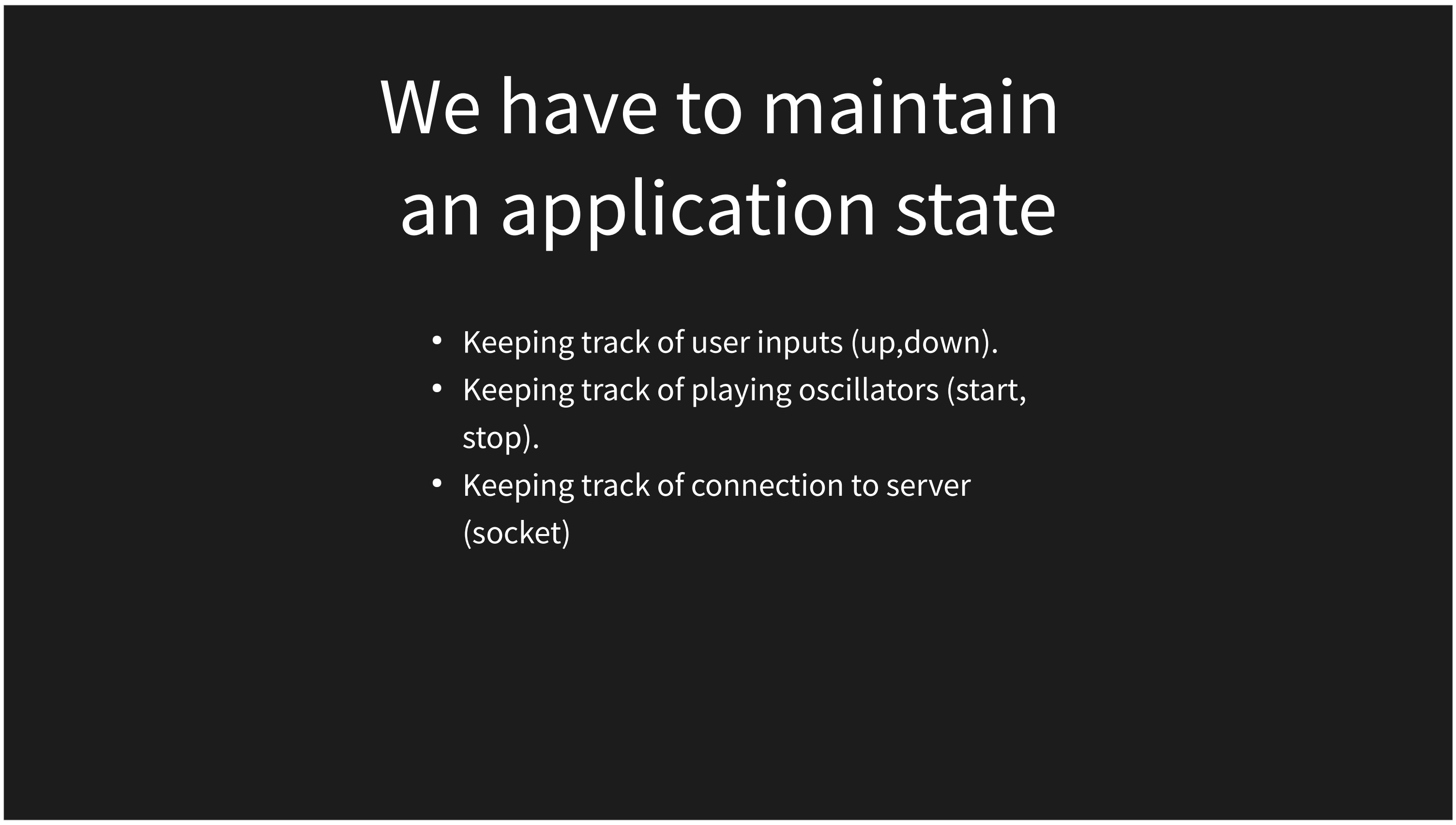
```
const server = http.createServer(app).listen(3000);
SocketServer.init(server,onMessage);

function onMessage(data,context) {
        SocketServer.broadcastToEveryoneElse(
                data,context
        );
}
```

There is actually a state, but it’s all handled by the library. All I am doing is providing functions that "mediate" between the library and the application.

The code turned out very small and efficient.

The abstraction is also hidden by the "context" variable that I’m passing around between the modules.

On the client it gets harder



We have to maintain an application state

- Keeping track of user inputs (up,down).
- Keeping track of playing oscillators (start, stop).
- Keeping track of connection to server (socket)

But I still want to maintain my modules stateless and functional. For example I want to be able to replace them on the fly with HMR.

How about an isolated, observable, mutable state/store.

./state.js

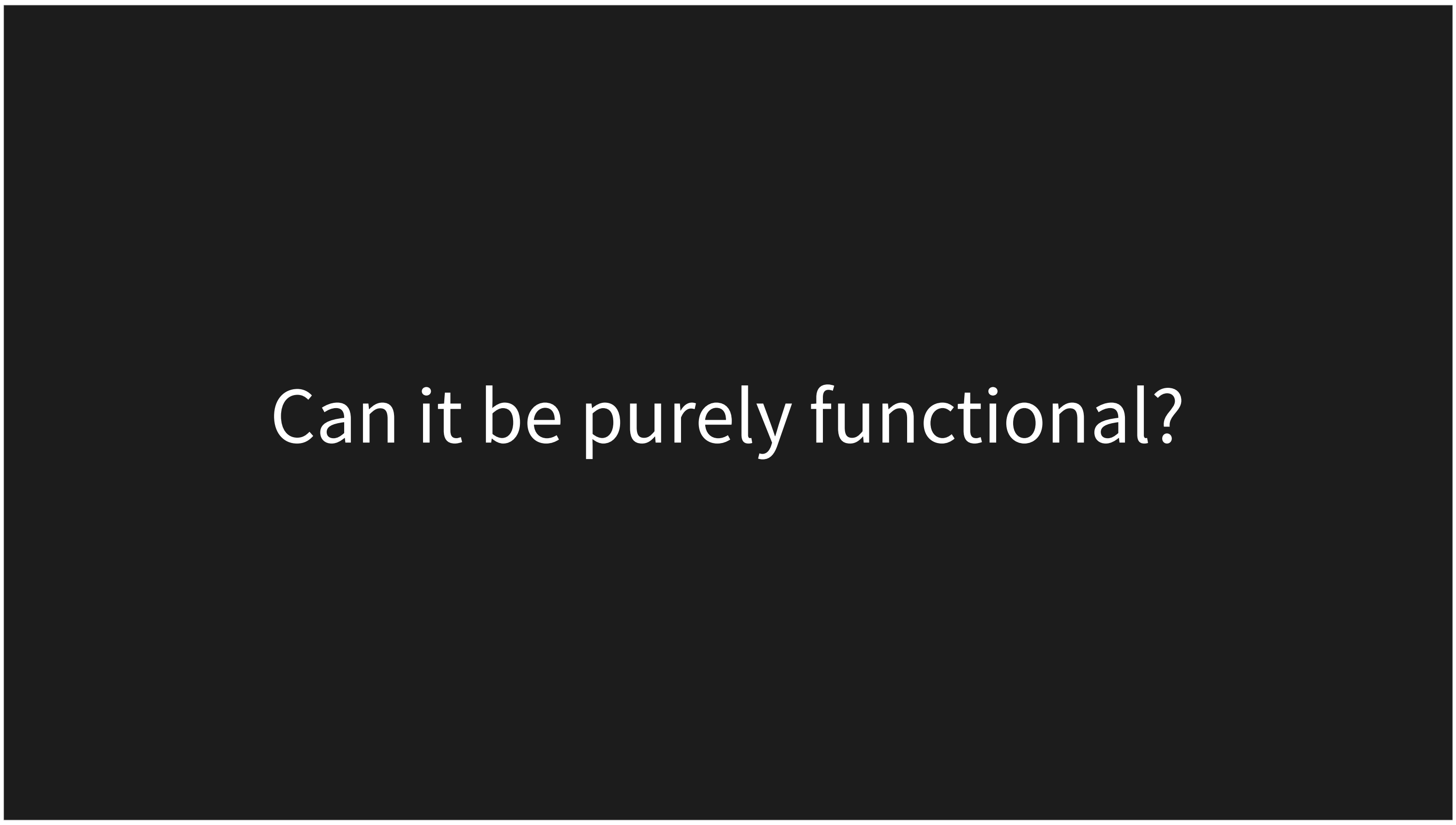
```
import observable from './observable-mixin'

export const Events = {}
observable(Events);

export const Synth = {};
export const Socket = {};
export const Input = {};
```

- A separation of concerns by using namespaces.
- Data is not shared between components, but only flows through events.

./socket-client.js

- The state object is the "memory" of the program.
- It also maintains the event chained, who is subscribed to whom.
- Communication between compomnents is done only through events. They modify their own state and send and event.

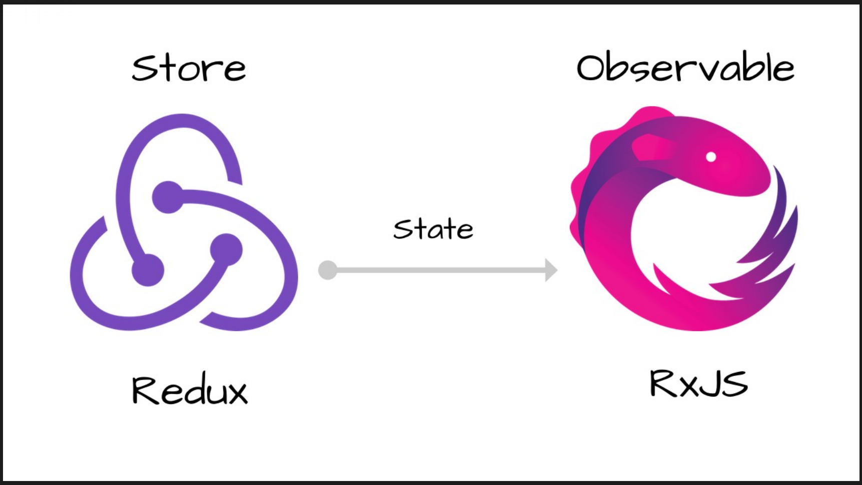
```
import ReconnectingWebSocket from 'reconnecting-websocket'
import {Socket,  Events} from './state'

export function init(url) {
   const socket = new ReconnectingWebSocket(url);
   socket.addEventListener('open', () => {
      Socket.handle = socket;
   });
   socket.addEventListener('close', () => {
      Socket.handle = null;
   });
   socket.addEventListener('message', (msg) => {Events.trigger('socket-message', msg)});
}
export function send(data) {
   if (Socket.Handle) {
      Socket.handle.send(data);
   }
}
```

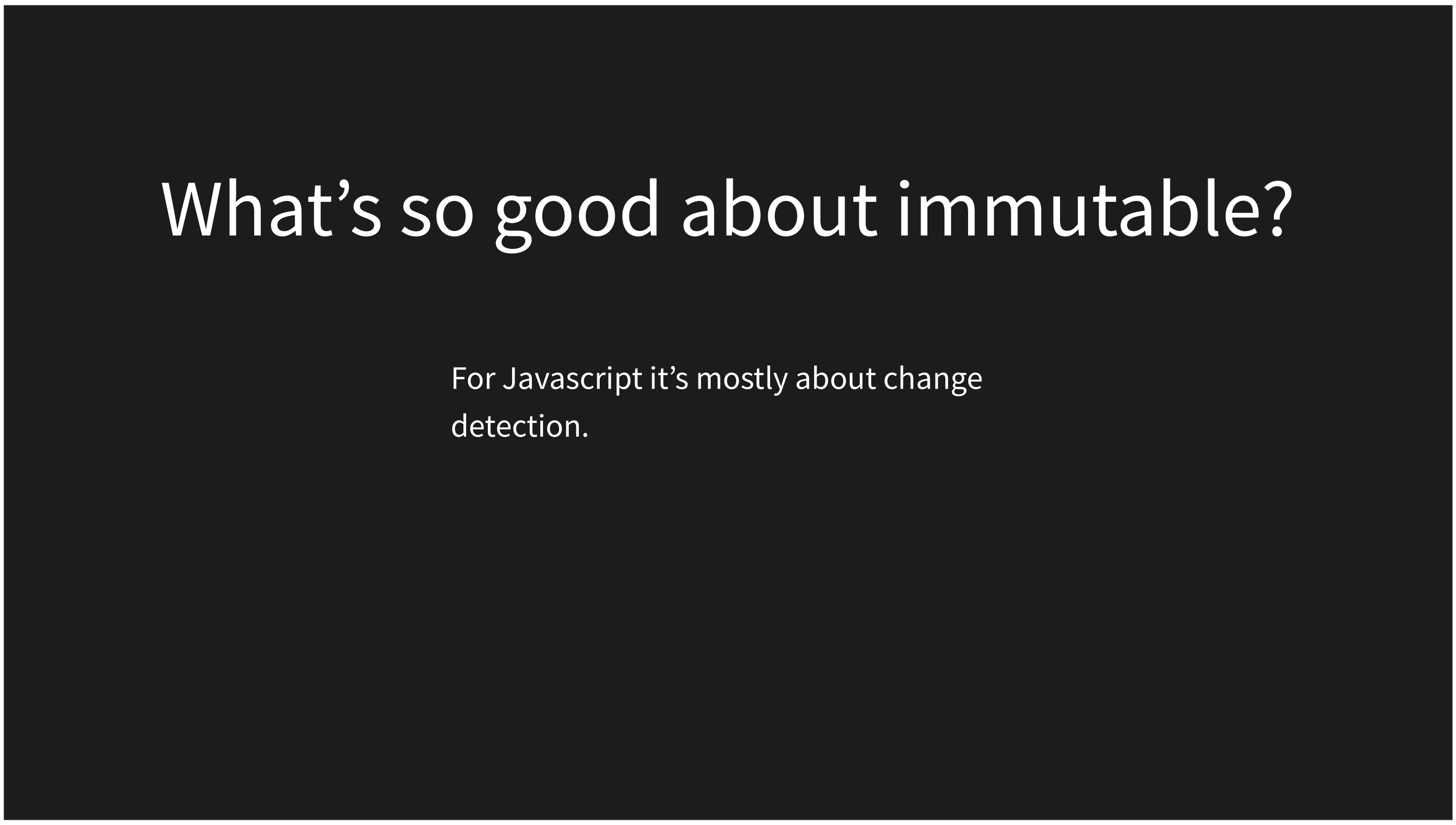


Can it be purely functional?

If instead of modifying the state object, we keep making new ones and return them. For Observable we can use the method of chaining callbacks.

Yeah





What’s so good about immutable?

For Javascript it’s mostly about change detection.

One of the biggest advantage is for concurrency, but javascript is single threaded.

Exercise

When your server components, are stateless, you can easily share the load between them, do a failover, move them around etc, with no need to do any replication.

Modify the collab-synth so that each user will get their own color when playing on the keys.

- Add a new client component named Piano, with its own namespace in the State object. It will export the functions:
  - noteOn(clientId, element)
  - noteOff(clientId, element)
- These functions will color the piano keyboard in a color that is random and unique for each client.
- In the server, you need to generate a client ID for each connection, and send it with the socket events. You can use a module leverl ID Counter.
- For now, no need to support more than 10 client connections.

NGrok