# HELLO.
# MY NAME IS JOE.
# PLEASED TO MEET YOU.

# EVENTS AND SOCKET.IO

*Building real-time software*

```
var userTweets = new EventEmitter();

// Elsewhere in the program . . .
userTweets.on('newTweet', function (tweet) {
    console.log(tweet);
});

// Elsewhere in the program . . .
userTweets.emit('newTweet', {
    text: 'Check out this fruit I ate'
});
```

```
var userTweets = new EventEmitter();

// Elsewhere in the program . . .
userTweets.on('newTweet', function (tweet) {
    console.log(tweet);
});

// Elsewhere in the program . . .
userTweets.emit('newTweet', {
    text: 'Check out this fruit I ate'
});
```

```
var userTweets = new EventEmitter();

// Elsewhere in the program . . .
userTweets.on('newTweet', function (tweet) {
    console.log(tweet);
});

// Elsewhere in the program . . .
userTweets.emit('newTweet', {
    text: 'Check out this fruit I ate'
});
```

```
var userTweets = new EventEmitter();

// Elsewhere in the program . . .
userTweets.on('newTweet', function (tweet) {
    console.log(tweet);
});

// Elsewhere in the program . . .
userTweets.emit('newTweet', {
    text: 'Check out this fruit I ate'
});
```

```
var userTweets = new EventEmitter();

// Elsewhere in the program . . .
userTweets.on('newTweet', function (tweet) {
    console.log(tweet);
});

// Elsewhere in the program . . .
userTweets.emit('newTweet', {
    text: 'Check out this fruit I ate'
});
```

# EVENT EMITTERS

- Objects that can “emit” specific events with a payload to any amount of registered listeners

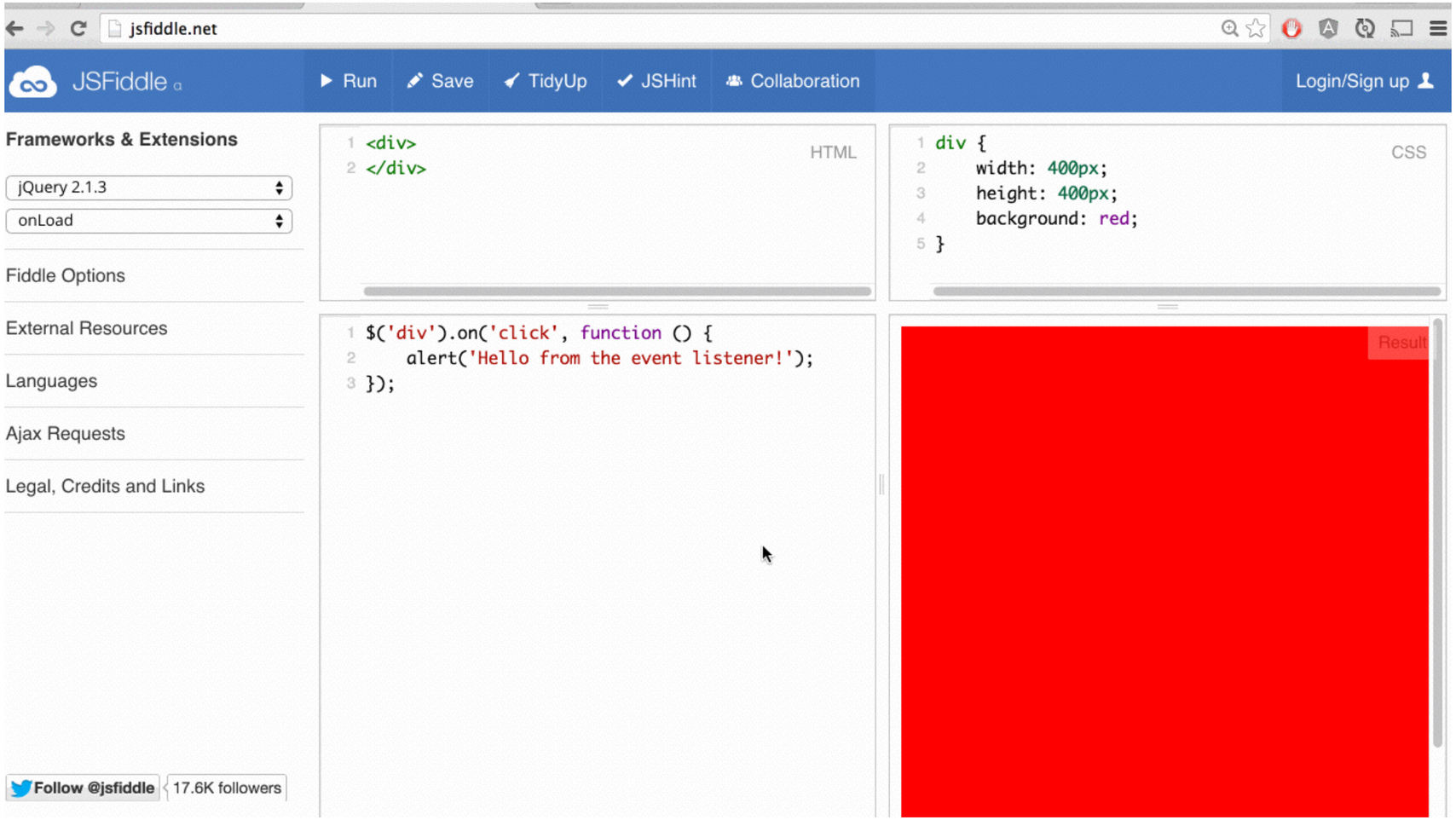
# EVENT EMITTERS

- Objects that can “emit” specific events with a payload to any amount of registered listeners

- Sometimes known as the “observer” or “pub/sub” pattern

# EVENT EMITTERS

- Objects that can “emit” specific events with a payload to any amount of registered listeners
- Sometimes known as the “observer” or “pub/sub” pattern
- Feels at-home with an *event*-driven environment


jsfiddle.net
JSFiddle
Run
Save
TidyUp
JSHint
Collaboration
Login/Sign up
Frameworks & Extensions
jQuery 2.1.3
onLoad
Fiddle Options
External Resources
Languages
Ajax Requests
Legal, Credits and Links
Follow @jsfiddle
17.6K followers
1 
2 
HTML
1 div {
2 width: 400px;
3 height: 400px;
4 background: red;
5 }
CSS
1 $('div').on('click', function () {
2 alert('Hello from the event listener!');
3 });
Result

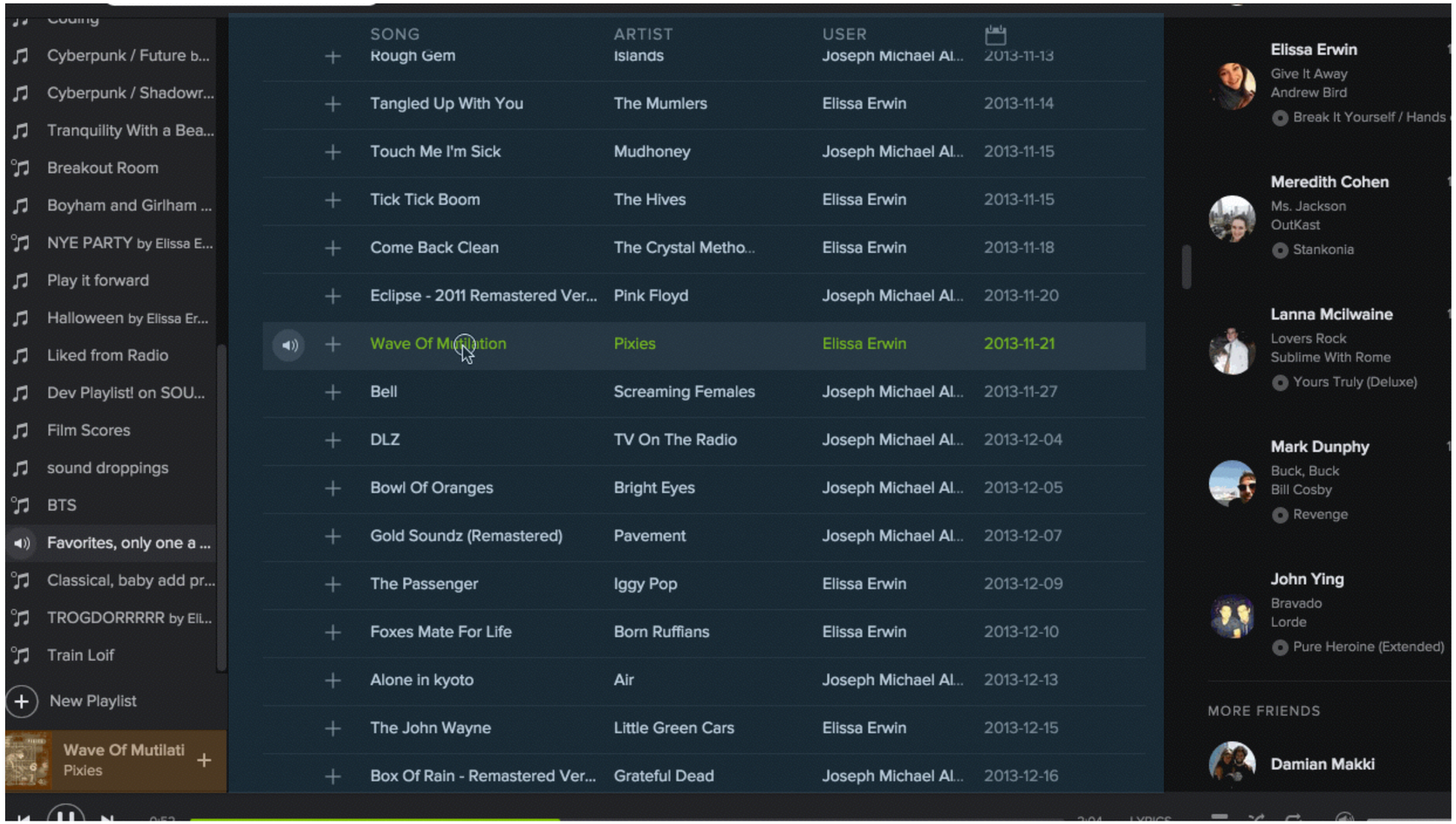
# PRACTICAL USES

- Connect two decoupled parts of an application

```
var currentTrack = new EventEmitter();
```

```
currentTrack.emit('changeTrack', newTrack);
```

```
currentTrack.on('changeTrack', function (newTrack) {
    // Display new track!
});
```



# PRACTICAL USES

- Represent multiple asynchronous events on a single entity.

```
var upload = uploadFile();

upload.on('error', function (e) {
    e.message; // World exploded!
});

upload.on('progress', function (percentage) {
    setProgressOnBar(percentage);
});

upload.on('complete', function (fileUrl, totalUploadTime) {

});
```

# ALL OVER NODE

- server.on(‘request’)
- request.on(‘data’) / request.on(‘end’)
- process.stdin.on(‘data’)
- mongoose.on(‘connection’)
- Streams

# HTTP, PART 2

*Sequels are always worse than the original*

# WHAT WE KNOW ABOUT HTTP

- A client makes a “request” to a server
- Server can receive this “request” and generate a “response”
- Only **ONE** response can (and must) be generated for a HTTP request
- A response can include a body or “payload” (HTML, JSON)

The New York Times

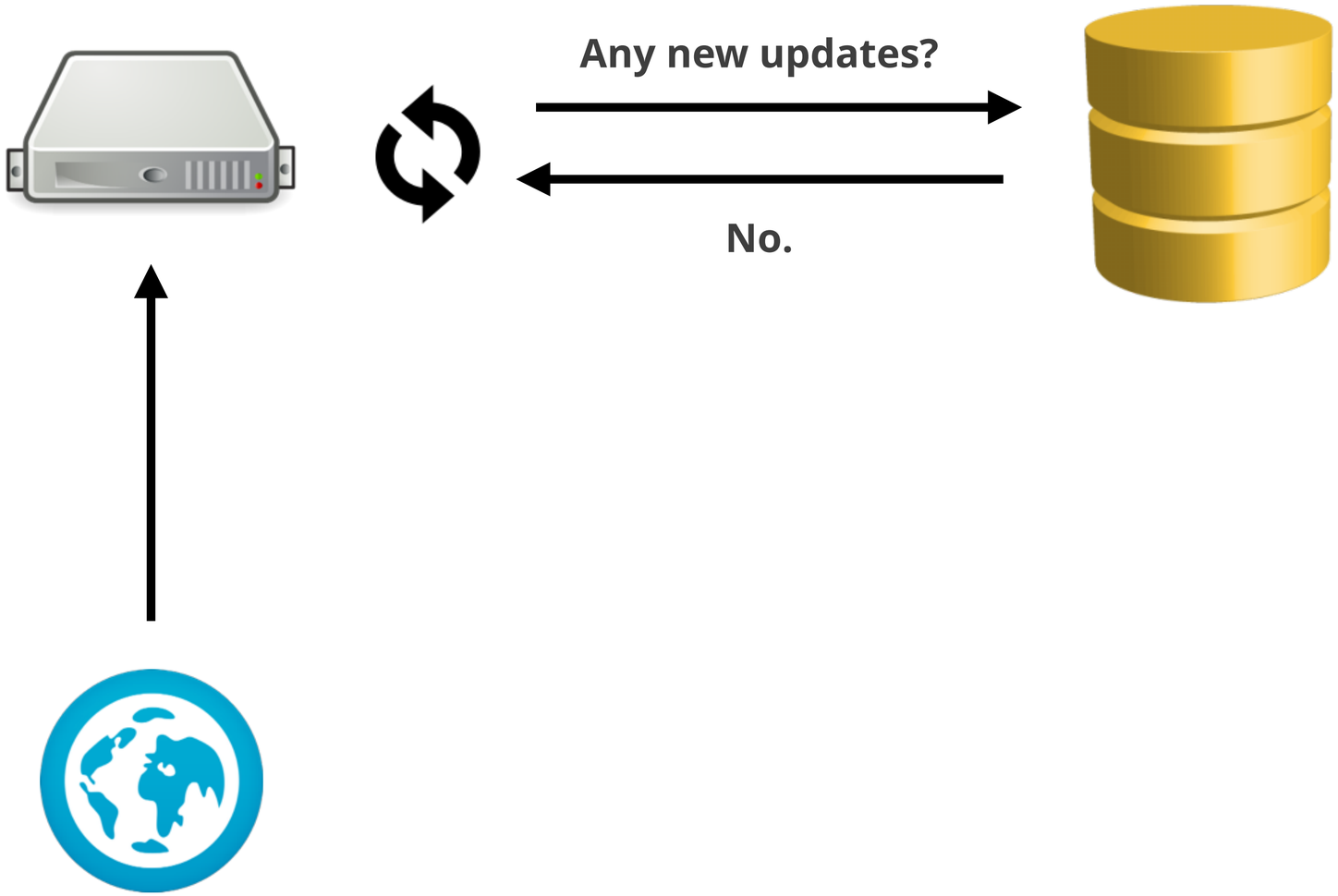

FIFA WORLD CUP
Brasil

# LIVE WORLD CUP COVERAGE

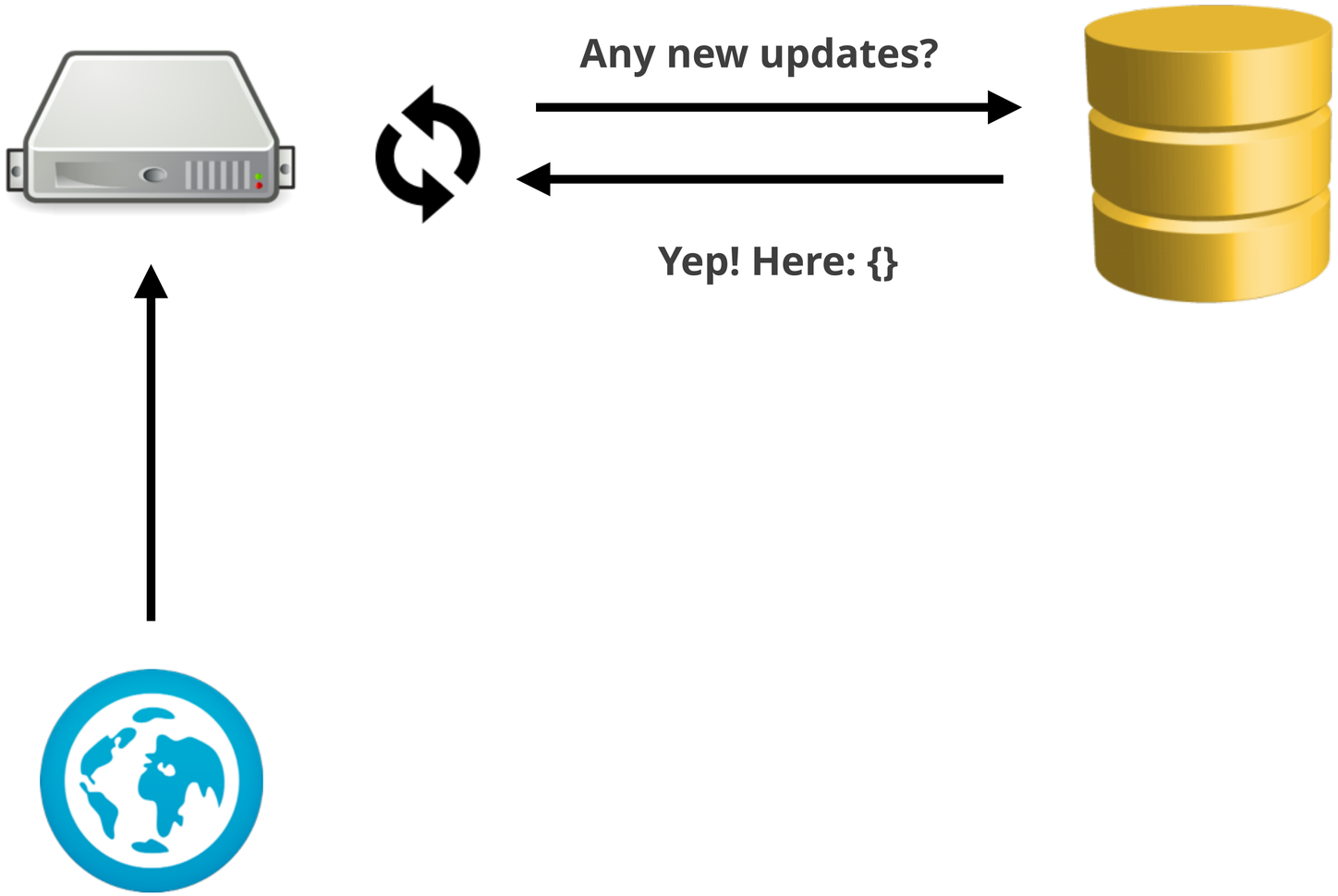
- A user visits a web page
- This web page has a live updating list of game coverage (“events”) provided by New York Times commentator (“Brazil receives yellow card”/“Germany scores goal”)
- When the event line is submitted by the commentator, it should immediately display to the user

# HTTP LONG POLLING

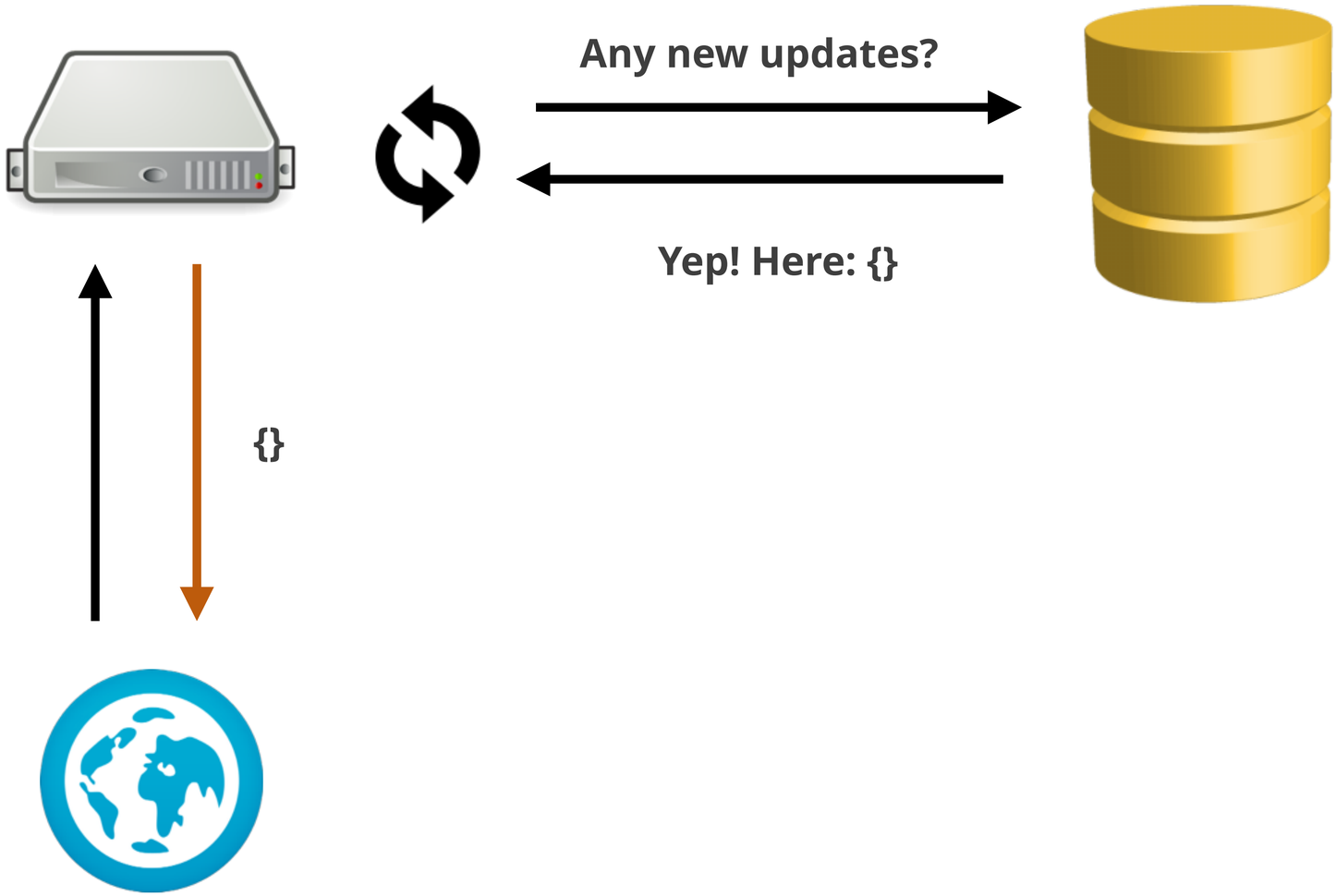
# HTTP LONG POLLING



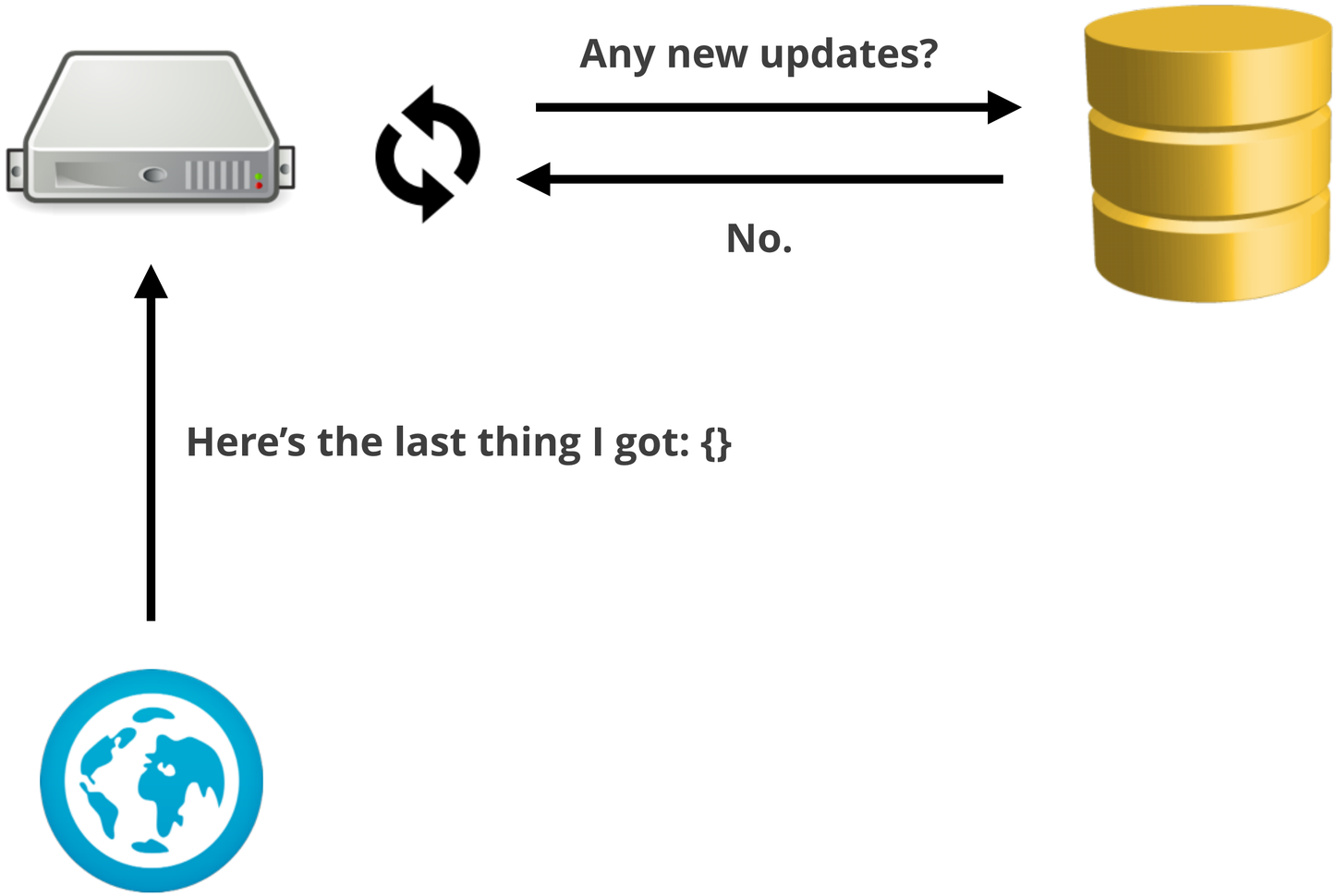
# HTTP LONG POLLING



# HTTP LONG POLLING



# HTTP LONG POLLING



# HTTP IS UNIDIRECTIONAL

COMMUNICATION MUST BE SOLICITED BY THE CLIENT

# TCP

*Transmission Control Protocol*

# TCP

- Protocol: standardized way that computers “communicate” with one another

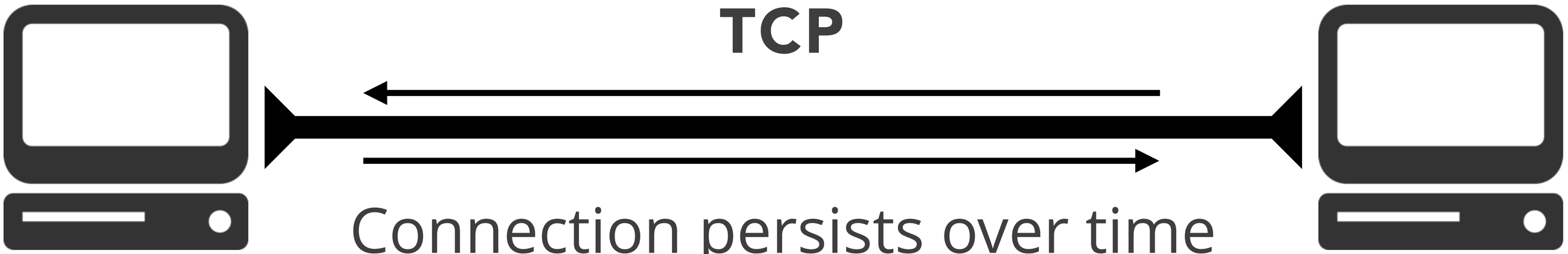
# TCP

- Protocol: standardized way that computers “communicate” with one another
- Establishes a duplex (two-way) persisted connection (or socket)

# TCP

- Protocol: standardized way that computers “communicate” with one another
- Establishes a duplex (two-way) persisted connection (or socket)
- Exists at the “transport” layer of the Internet model.


Connection persists over time




Connection persists over time

# TCP AND HTTP

- HTTP is a protocol that lives at the “application” layer of the Internet model.

# TCP AND HTTP

- HTTP is a protocol that lives at the “application” layer of the Internet model.
- It is an abstraction on top of TCP.

# TCP AND HTTP

- HTTP is a protocol that lives at the “application” layer of the Internet model.
- It is an abstraction on top of TCP.
- Implements the idea of a “session”, which establishes a TCP socket for the client/server communication and data transport.

# HTTP: CLIENT SOLICITS CONNECTION

# TCP CONNECTION IS ESTABLISHED FOR DATA TRANSPORT

# HTTP USES THIS TCP CONNECTION TO SEND REQUEST FROM CLIENT

# HTTP USES THIS TCP CONNECTION TO SEND RESPONSE FROM SERVER

# HTTP TERMINATES TCP CONNECTION

# HTTP'S USAGE OF TCP

- To transport data between client/server

# HTTP'S USAGE OF TCP

- To transport data between client/server
- Source IP / Source Port <—> Dest. IP / Dest. Port

# HTTP'S USAGE OF TCP

- To transport data between client/server
- Source IP / Source Port <—> Dest. IP / Dest. Port
- Header "Connection: Keep-Alive" allows for the TCP socket to not be terminated and reused for subsequent requests

# HTTP'S USAGE OF TCP

- To transport data between client/server
- Source IP / Source Port <—> Dest. IP / Dest. Port
- Header "Connection: Keep-Alive" allows for the TCP socket to not be terminated and reused for subsequent requests
- Client still solicits any communication

# WEBSOCKETS AND SOCKET.IO

# WEBSOCKETS

- Implementation of pure TCP sockets between a browser application and a server application

# WEBSOCKETS

- Implementation of pure TCP sockets between a browser application and a server application

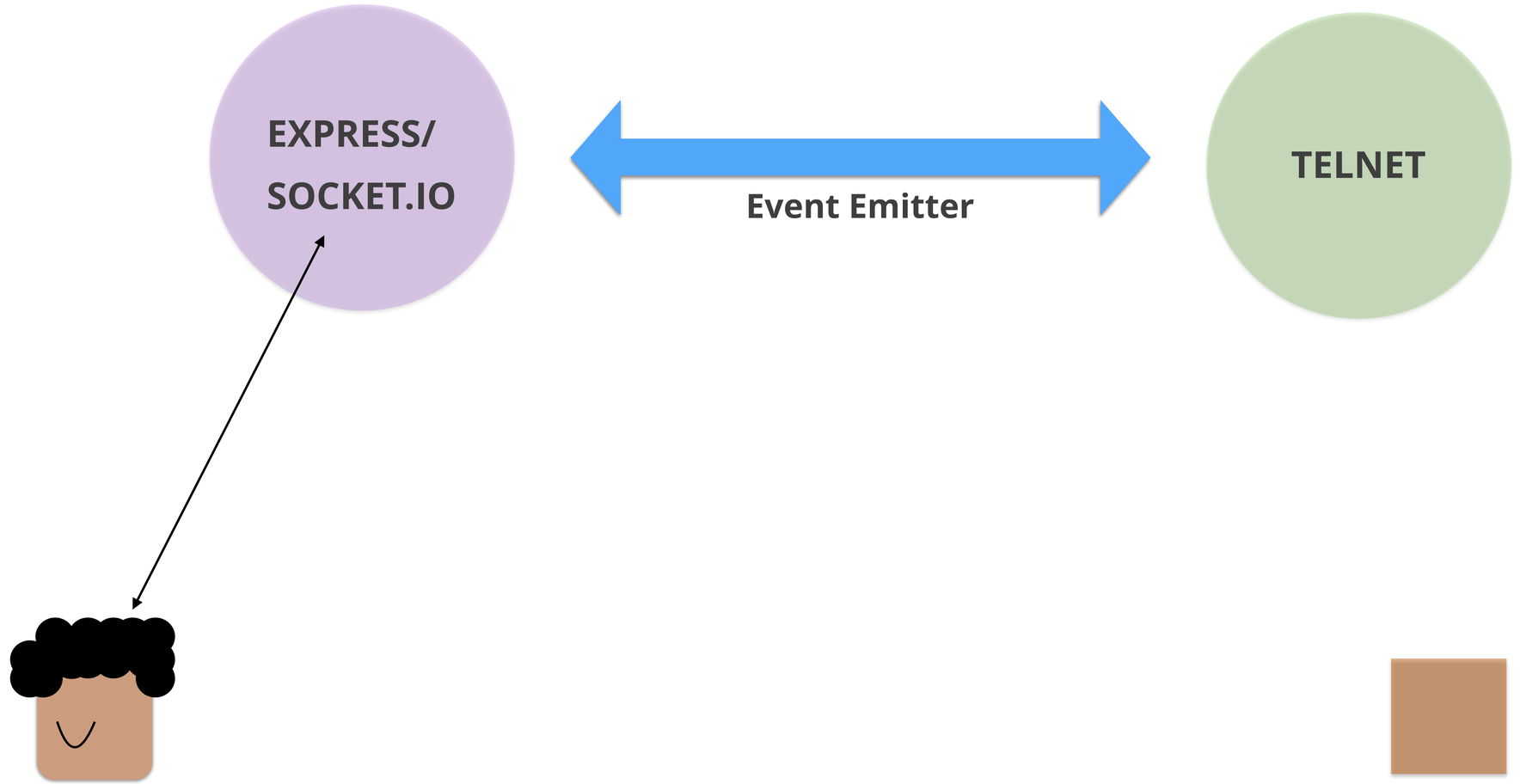
- Allows for bi-directional communication, included a server arbitrarily pushing data to a browser without solicitation

# WEBSOCKETS

- Implementation of pure TCP sockets between a browser application and a server application
- Allows for bi-directional communication, included a server arbitrarily pushing data to a browser without solicitation
- Allows for awesome real-time software

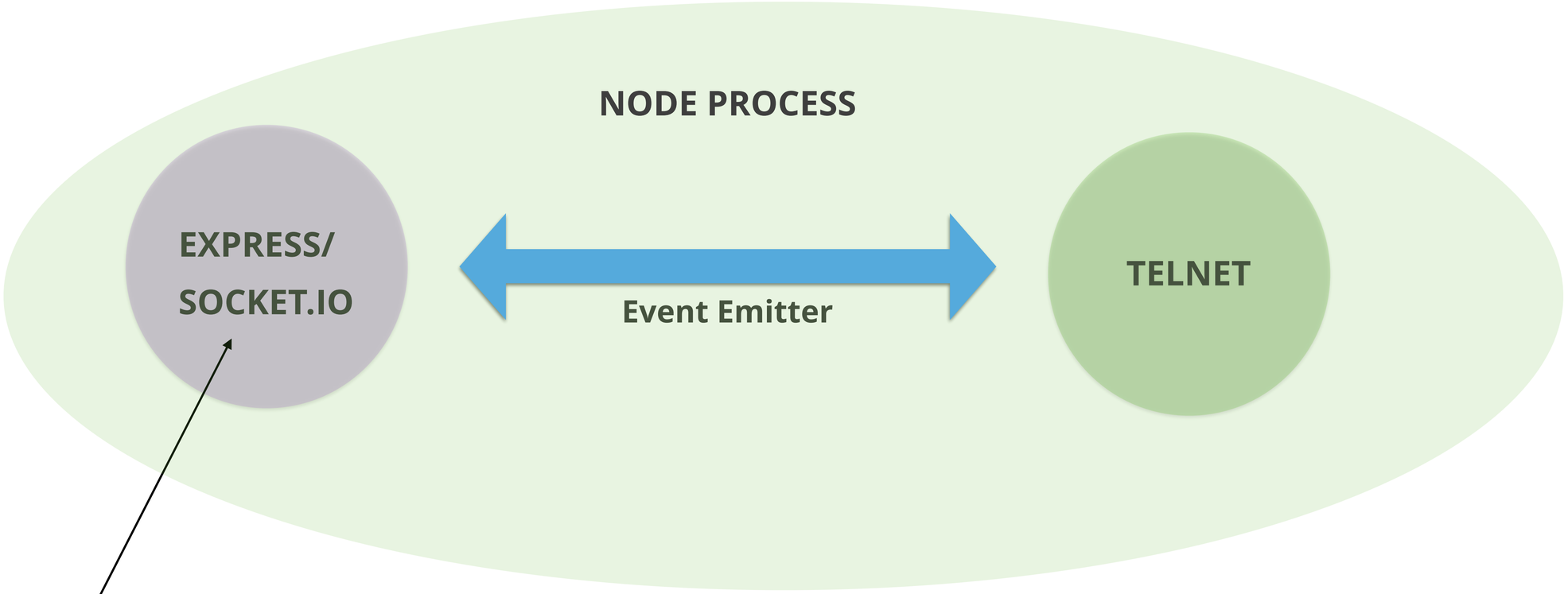
# SOCKET.IO

- A duet of libraries (one for server-side [node.js] and one for client-side [the browser])
- Abstracts the complex implementation of websockets for easy use
- Is used primarily via Event Emitters




KEYNOTE JOE AND HIS BROWSER

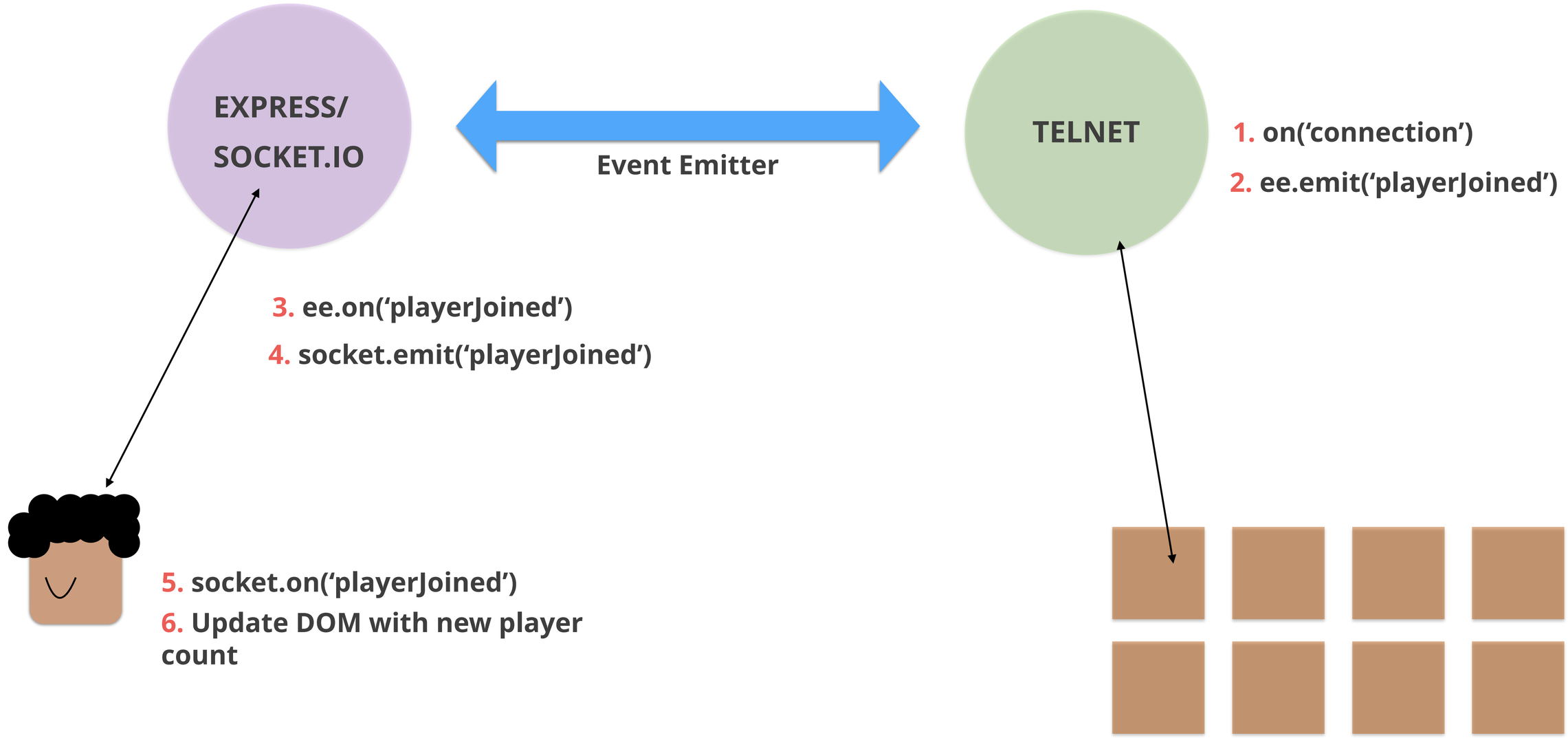
Y'ALL




KEYNOTE JOE AND HIS BROWSER

Y'ALL

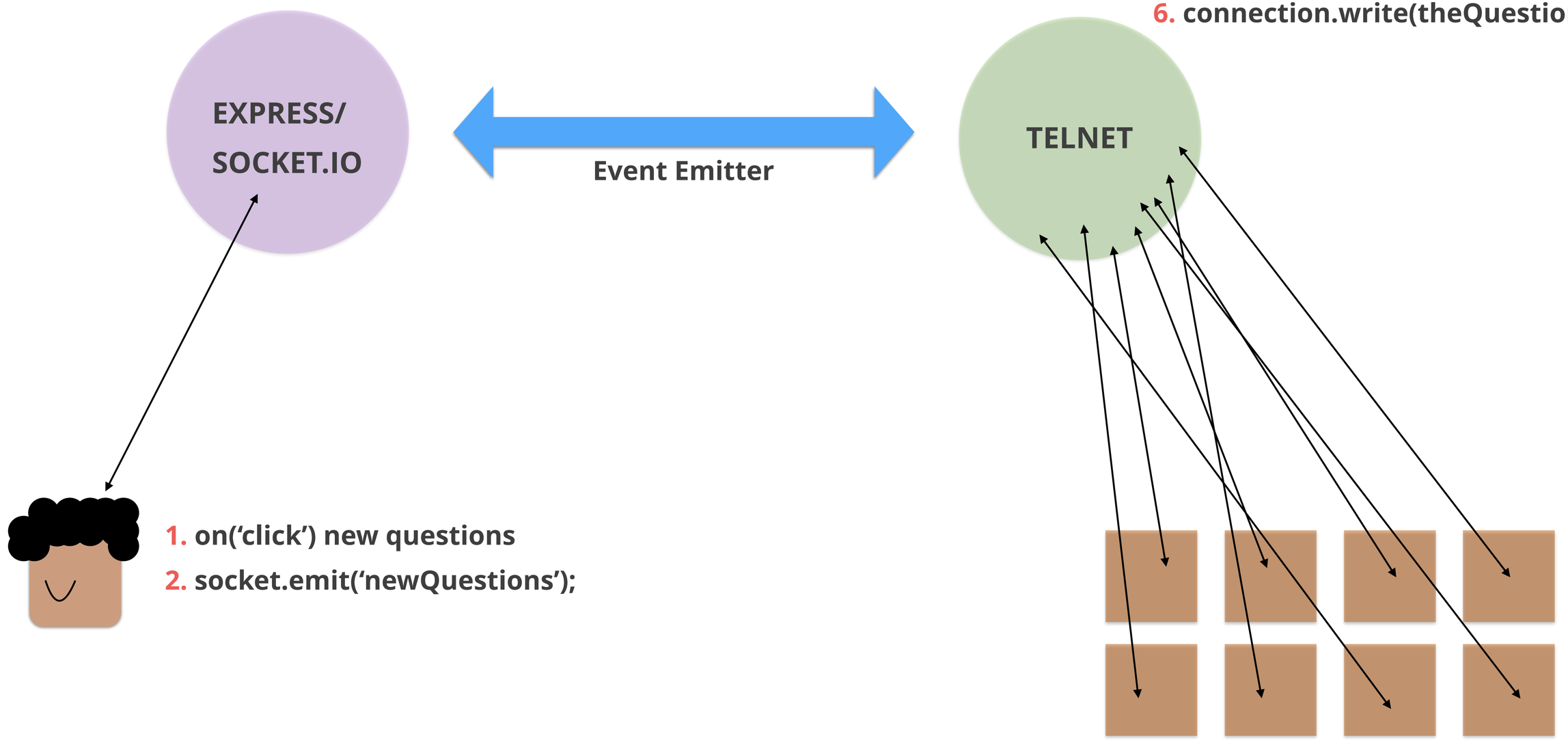
# PLAYER JOINS



# NEW QUESTIONS

3. socket.on('newQuestions');
4. ee.emit('sendNewQuestions');

5. ee.on('sendNewQuestions');
6. connection.write(theQuestion)




1. on('click') new questions
2. socket.emit('newQuestions');

# ANSWERS

1. connection.on('data');
2. ee.emit('answer')

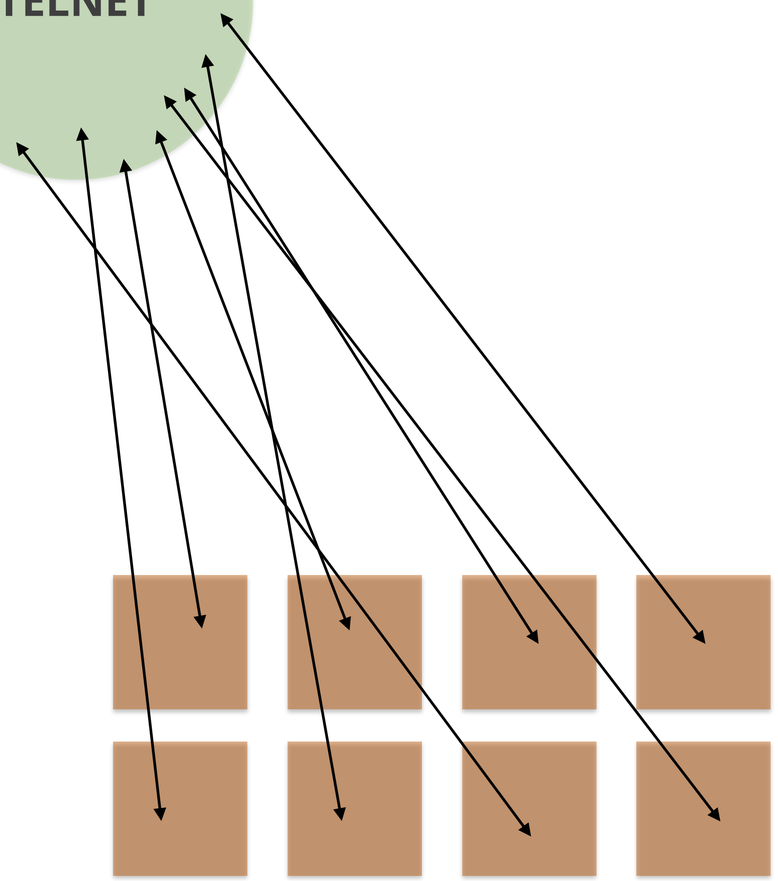
3. ee.on('answer')
4. socket.emit('answer')

EXPRESS/
SOCKET.IO

Event Emitter

TELNET

5. socket.on('answer')
6. Add to DOM



# USE CASES

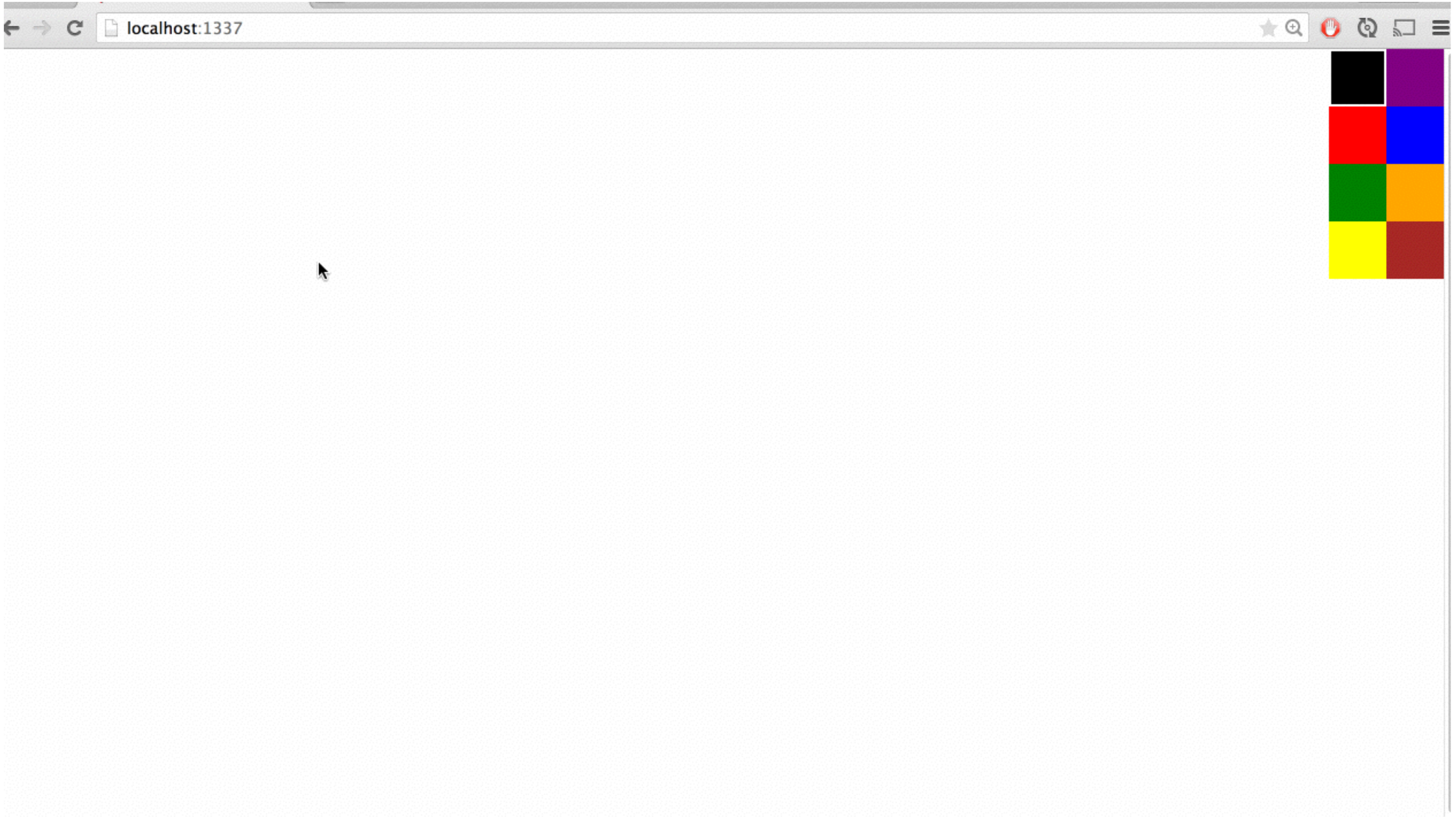
- Networked enabled games
- Chat applications
- Collaborative applications
- Any “real-time” software

# DRAWBACKS

- Servers have to maintain persisted state of connections
- Significantly more overhead
- Not as efficient or on-demand as HTTP — could have a socket sit dormant for a long time

# OTHER NOTES

- Documentation leaves a lot to be desired
- Automatically uses fallbacks for different capabilities and environments (long polling, Flash)
- Has “rooms” and “namespaces” for socket organization


localhost:1337