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

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
- An instance of the “observer/observable” a.k.a “pub/sub” pattern

EVENT EMITTERS

- Objects that can “emit” specific events with a payload to any amount of registered listeners
- An instance of the “observer/observable” a.k.a “pub/sub” pattern
- Feels at-home in an *event*-driven environment

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’)
- db.on(‘connection’)
- Streams

HTTP, PART 2

Sequels are always worse than the original

WHAT WE KNOW ABOUT HTTP

WHAT WE KNOW ABOUT HTTP

- A client makes a “request” to a server

WHAT WE KNOW ABOUT HTTP

- A client makes a “request” to a server
- Server receives this “request” and generates a “response”

WHAT WE KNOW ABOUT HTTP

- A client makes a “request” to a server
- Server receives this “request” and generates a “response”
- One request, one response: them’s the rules

WHAT WE KNOW ABOUT HTTP

- A client makes a “request” to a server
- Server receives this “request” and generates a “response”
- One request, one response: them’s the rules
- Requests can include a body (payload)

WHAT WE KNOW ABOUT HTTP

- A client makes a “request” to a server
- Server receives this “request” and generates a “response”
- One request, one response: them’s the rules
- Requests can include a body (payload)
- Responses can include a body (payload)

The New York Times

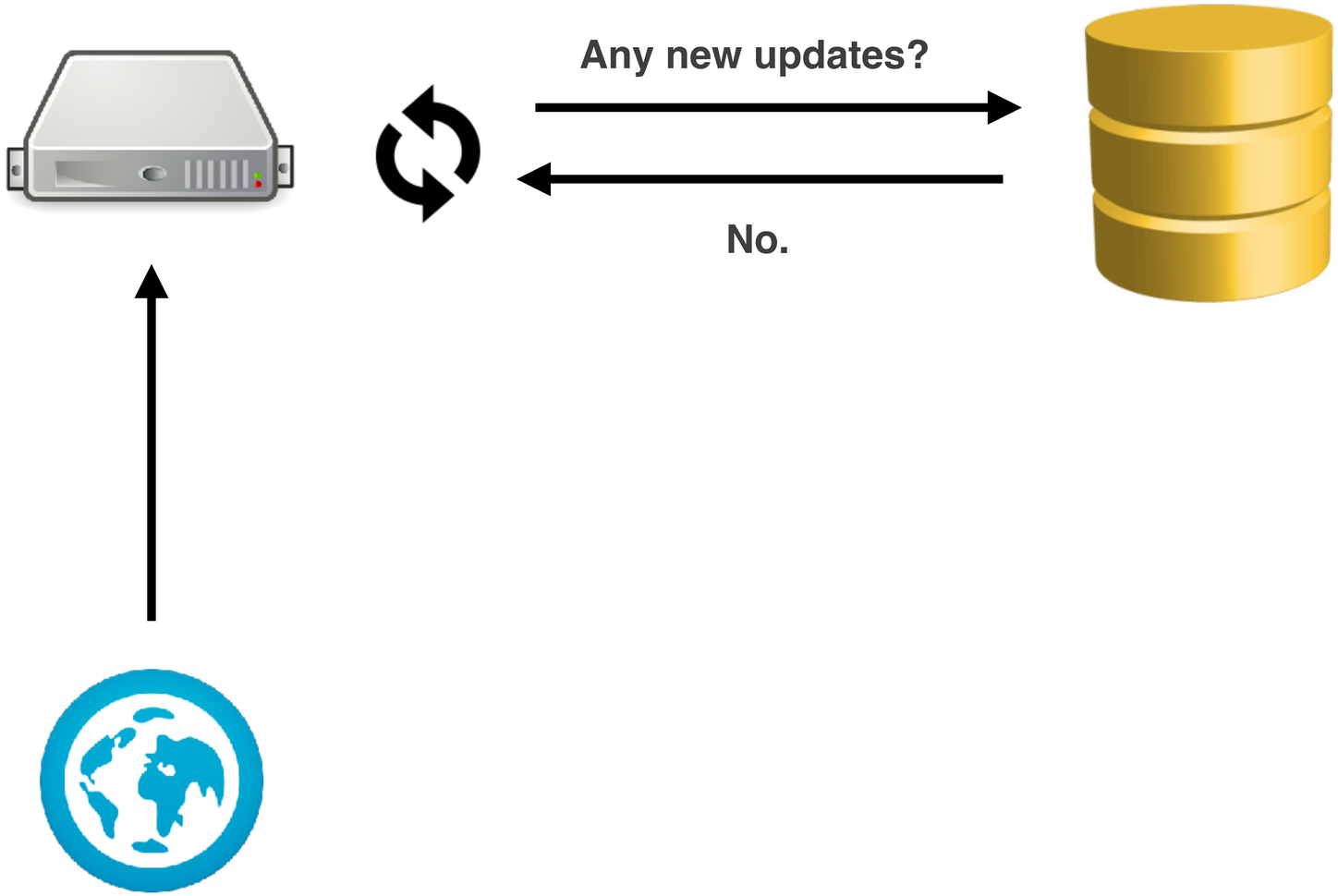
FIFA WORLD CUP
Brasil

LIVE WORLD CUP COVERAGE

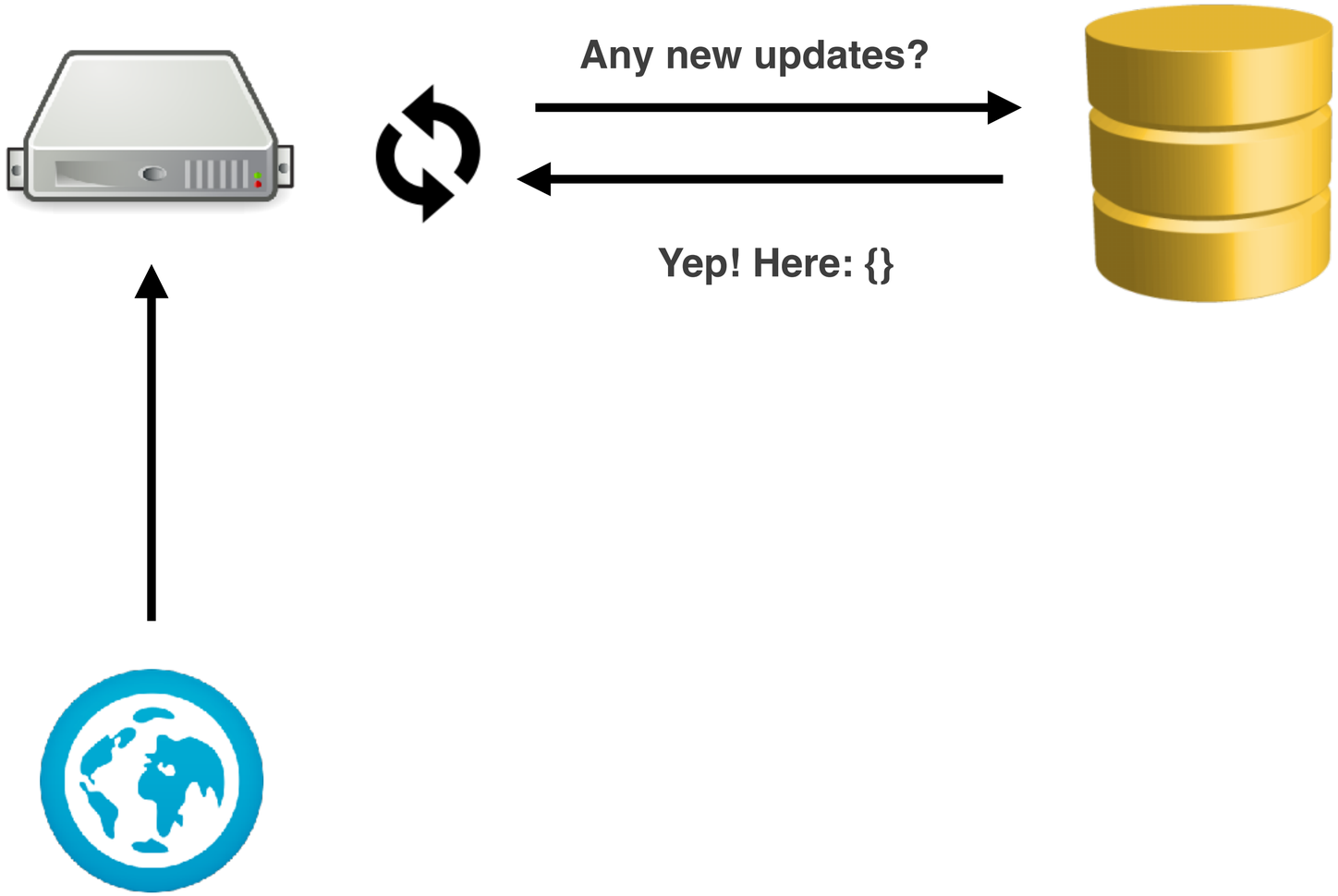
- A user visits a web page
- This web page has a live updating list of game coverage (“events”) provided by New York Times commentator (“Brazil receives yellow card”/“Germany scores goal”)
- When the event line is submitted by the commentator, it should immediately display to the user

HTTP LONG POLLING

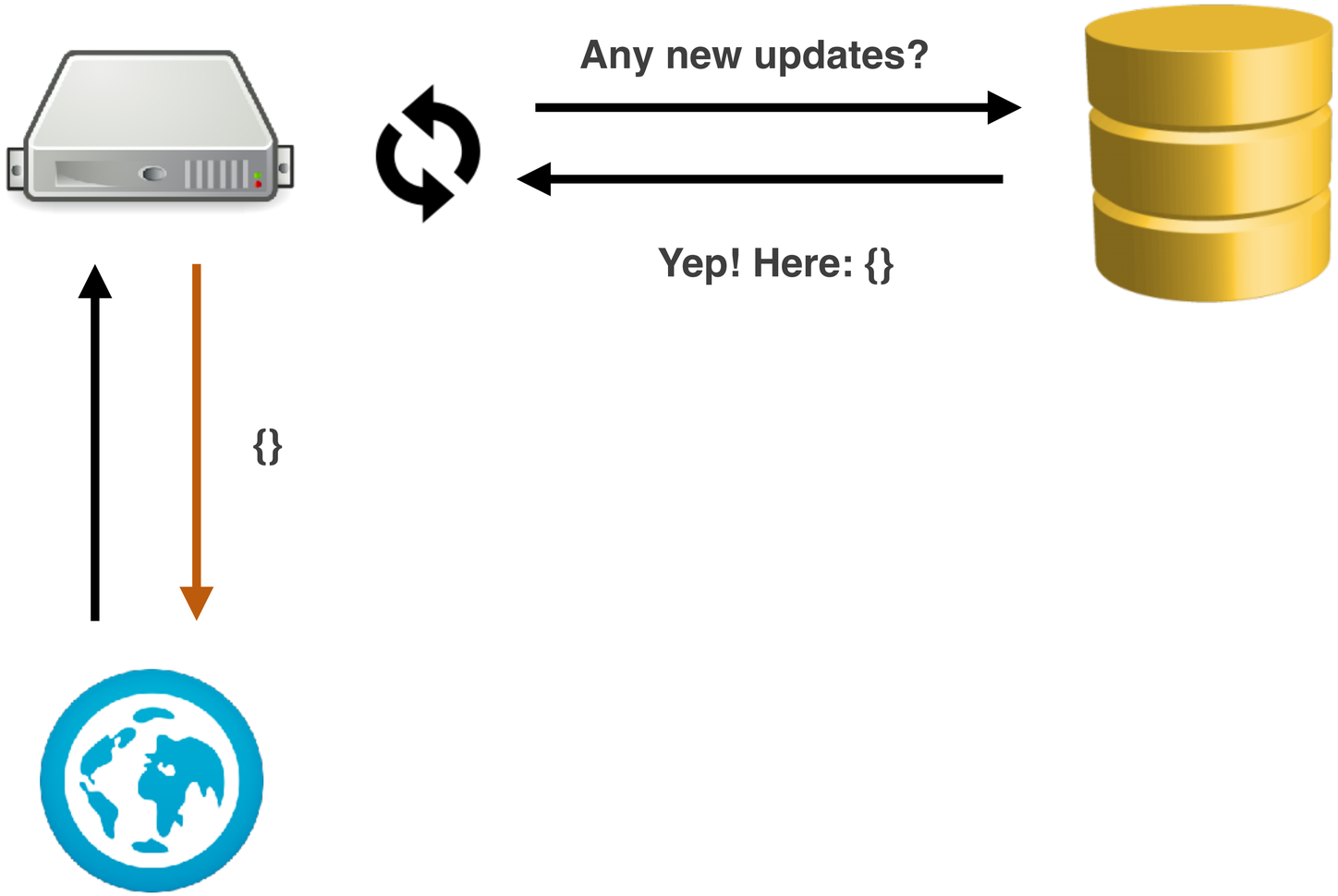
HTTP LONG POLLING



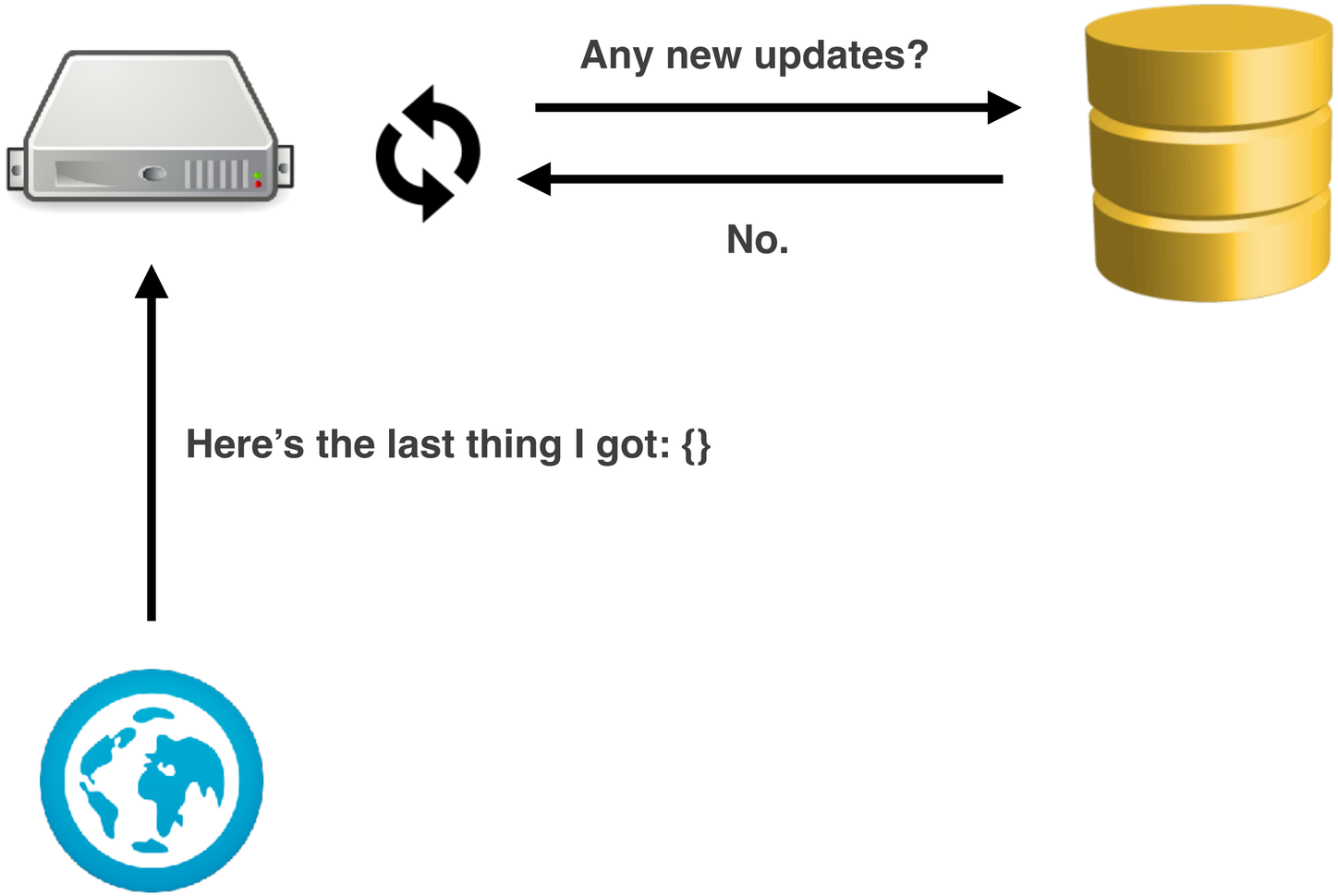
HTTP LONG POLLING



HTTP LONG POLLING



HTTP LONG POLLING



HTTP IS A REQUEST/RESPONSE PROTOCOL

HTTP IS A REQUEST/RESPONSE PROTOCOL

- Clients must send a *request* before the server can issue a *response*

HTTP IS A REQUEST/RESPONSE PROTOCOL

- Clients must send a *request* before the server can issue a *response*
- There is no way for the server to *push* data to the client without an outstanding request

HTTP IS A REQUEST/RESPONSE PROTOCOL

- Clients must send a *request* before the server can issue a *response*
- There is no way for the server to *push* data to the client without an outstanding request
- No live updates without long polling 😢

TCP

Transmission Control Protocol

TCP

TCP

- Protocol: standardized way that computers communicate with one another

TCP

- Protocol: standardized way that computers communicate with one another
- Establishes a reliable, duplex connection between two machines that persists over time

TCP

- **Protocol: standardized way that computers communicate with one another**
- **Establishes a reliable, duplex connection between two machines that persists over time**
  - **Reliable:** All your data gets there in the order you sent it

TCP

- **Protocol: standardized way that computers communicate with one another**
- **Establishes a reliable, duplex connection between two machines that persists over time**
  - **Reliable:** All your data gets there in the order you sent it
    - (or you know that it didn't)

TCP

- **Protocol: standardized way that computers communicate with one another**
- **Establishes a reliable, duplex connection between two machines that persists over time**
  - **Reliable:** All your data gets there in the order you sent it
    - (or you know that it didn't)
  - **Duplex:** Either end of the connection can send or receive bits

TCP

- **Protocol: standardized way that computers communicate with one another**
- **Establishes a reliable, duplex connection between two machines that persists over time**
  - **Reliable:** All your data gets there in the order you sent it
    - (or you know that it didn't)
  - **Duplex:** Either end of the connection can send or receive bits
  - **Persistent:** The connection lasts until one side ends it

TCP

- **Protocol: standardized way that computers communicate with one another**
- **Establishes a reliable, duplex connection between two machines that persists over time**
  - **Reliable:** All your data gets there in the order you sent it
    - (or you know that it didn't)
  - **Duplex:** Either end of the connection can send or receive bits
  - **Persistent:** The connection lasts until one side ends it
- **TCP is a *transport* layer protocol**

TCP AND HTTP

TCP AND HTTP

- HTTP is an application layer protocol

TCP AND HTTP

- HTTP is an application layer protocol
- It (usually) operates over TCP, (usually) on port 80

TCP AND HTTP

- **HTTP is an application layer protocol**
- **It (usually) operates over TCP, (usually) on port 80**
  - But “HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used” — HTTP 1.1 Spec

TCP AND HTTP

- **HTTP is an application layer protocol**
- **It (usually) operates over TCP, (usually) on port 80**
  - But “HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used” — HTTP 1.1 Spec
  - HTTPS, for instance, operates over TLS on port 443

TCP AND HTTP

- **HTTP is an application layer protocol**
- **It (usually) operates over TCP, (usually) on port 80**
  - But “HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used” — HTTP 1.1 Spec
  - HTTPS, for instance, operates over TLS on port 443
- **Implements the idea of a “session”, which establishes a TCP socket for the client to make requests and the server to issue responses**

CLIENT OPENS A TCP CONNECTION TO SERVER

CLIENT OPENS A TCP CONNECTION TO SERVER

let’s synchronize!

CLIENT OPENS A TCP CONNECTION TO SERVER

let's synchronize!

CLIENT OPENS A TCP CONNECTION TO SERVER

let's synchronize!

ok, let's synchronize!

CLIENT OPENS A TCP CONNECTION TO SERVER

let's synchronize!

ok, let's synchronize!

CLIENT OPENS A TCP CONNECTION TO SERVER

let's synchronize!

ok, let's synchronize!

ok!

CLIENT OPENS A TCP CONNECTION TO SERVER

let's synchronize!

ok, let's synchronize!

ok!

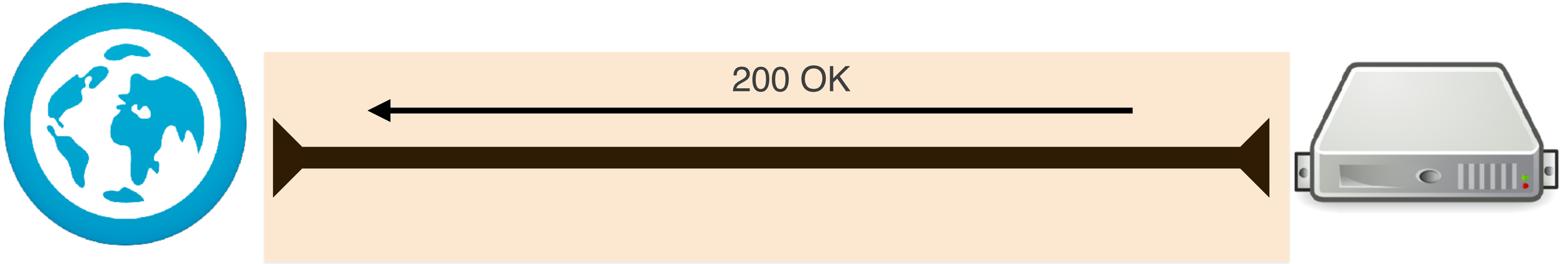
TCP CONNECTION IS ESTABLISHED

CLIENT SENDS A REQUEST

(over the connection)

SERVER SENDS A RESPONSE

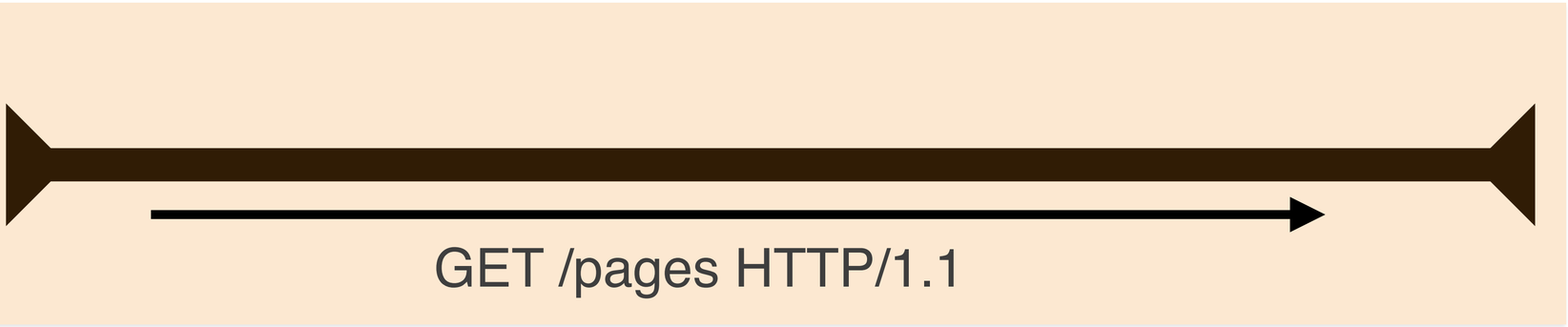
(over the connection)



TCP CONNECTION STAYS OPEN

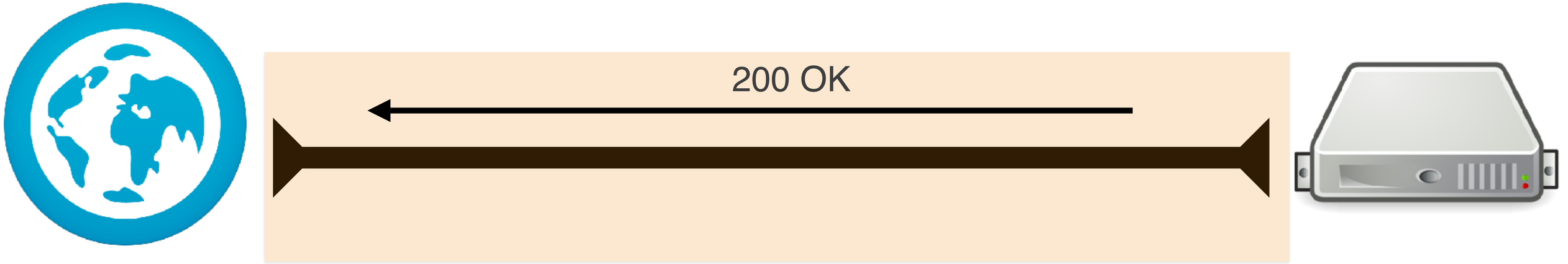
CLIENT SENDS MORE REQUESTS

(over the same connection)



SERVER SENDS MORE RESPONSES

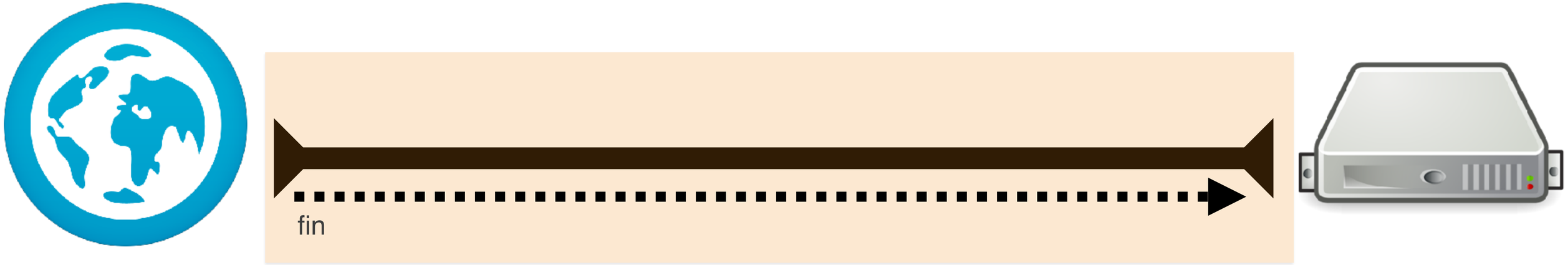
(over the same connection)



EVENTUALLY, YOU CLOSE THE TAB

EVENTUALLY, YOU CLOSE THE TAB

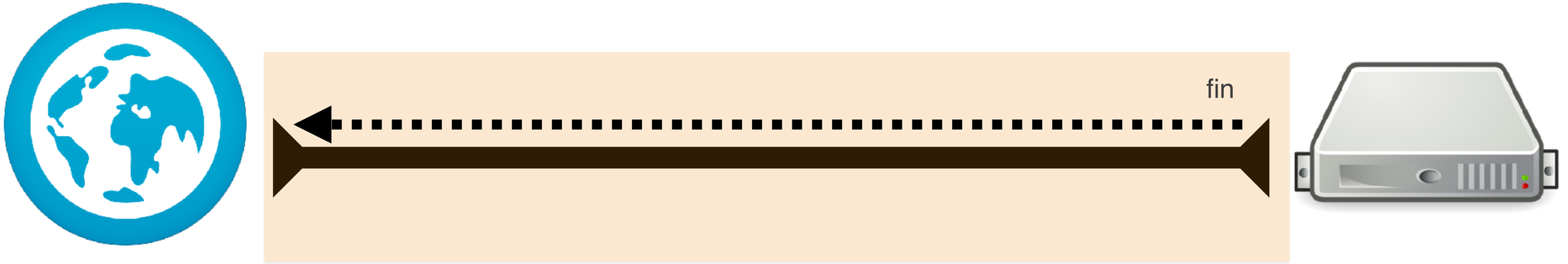
EVENTUALLY, YOU CLOSE THE TAB



OR YOU DON'T SAY ANYTHING FOR A WHILE AND THE SERVER TIMES OUT

OR YOU DON'T SAY ANYTHING FOR A WHILE AND THE SERVER TIMES OUT

OR YOU DON'T SAY ANYTHING FOR A WHILE AND THE SERVER TIMES OUT



AND ONE OF YOU ENDS THE CONNECTION

HTTP 1.1 REQUEST / RESPONSE CYCLE

HTTP 1.1 REQUEST / RESPONSE CYCLE

- Client sends a request

HTTP 1.1 REQUEST / RESPONSE CYCLE

- Client sends a request
- Server sends a response

HTTP 1.1 REQUEST / RESPONSE CYCLE

- Client sends a request
- Server sends a response
- Server can’t “push” more data to the client unless the client makes another request

HTTP 1.1 REQUEST / RESPONSE CYCLE

- **Client sends a request**
- **Server sends a response**
- **Server can't "push" more data to the client unless the client makes another request**
  - …Even though there's this tasty TCP connection just sitting around

WEBSOCKETS AND SOCKET.IO

WEBSOCKETS START WITH HTTP

Client says:

Server replies:

WEBSOCKETS START WITH HTTP

Client says:

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://example.com

Server replies:

WEBSOCKETS START WITH HTTP

Client says:

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://example.com

Server replies:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat

WEBSOCKETS START WITH HTTP

Client says:

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://example.com

Server replies:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat

And now WebSocket has taken over the connection.

SOCKET.IO

- **You don’t have to implement that**
- **Socket.IO is 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**
- **Extensively uses** EventEmitter**s**
  - EventEmitters are a good fit for a message-based protocol

USE CASES

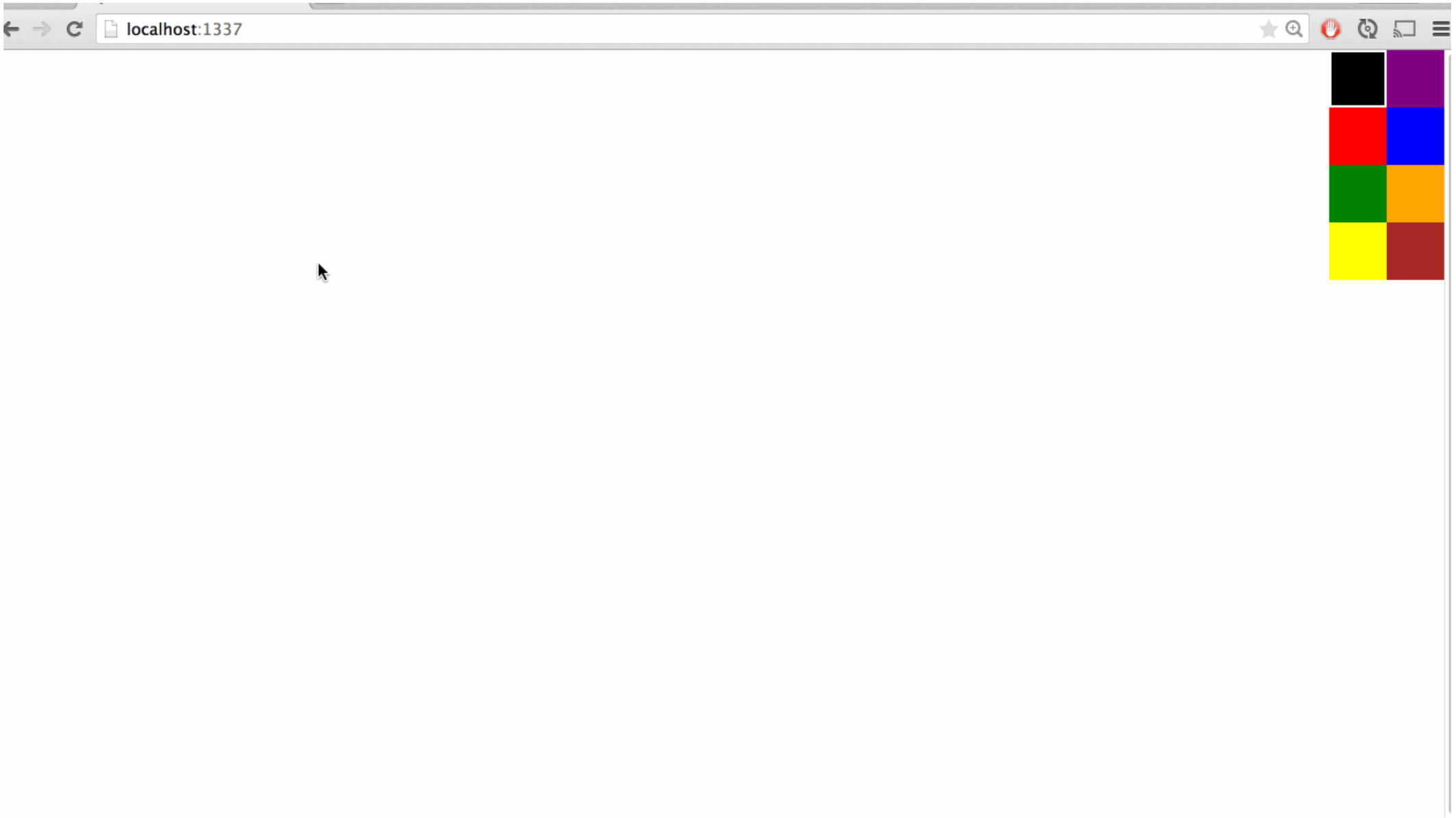
- Networked enabled games
- Chat applications
- Collaborative applications
- Any “real-time” software

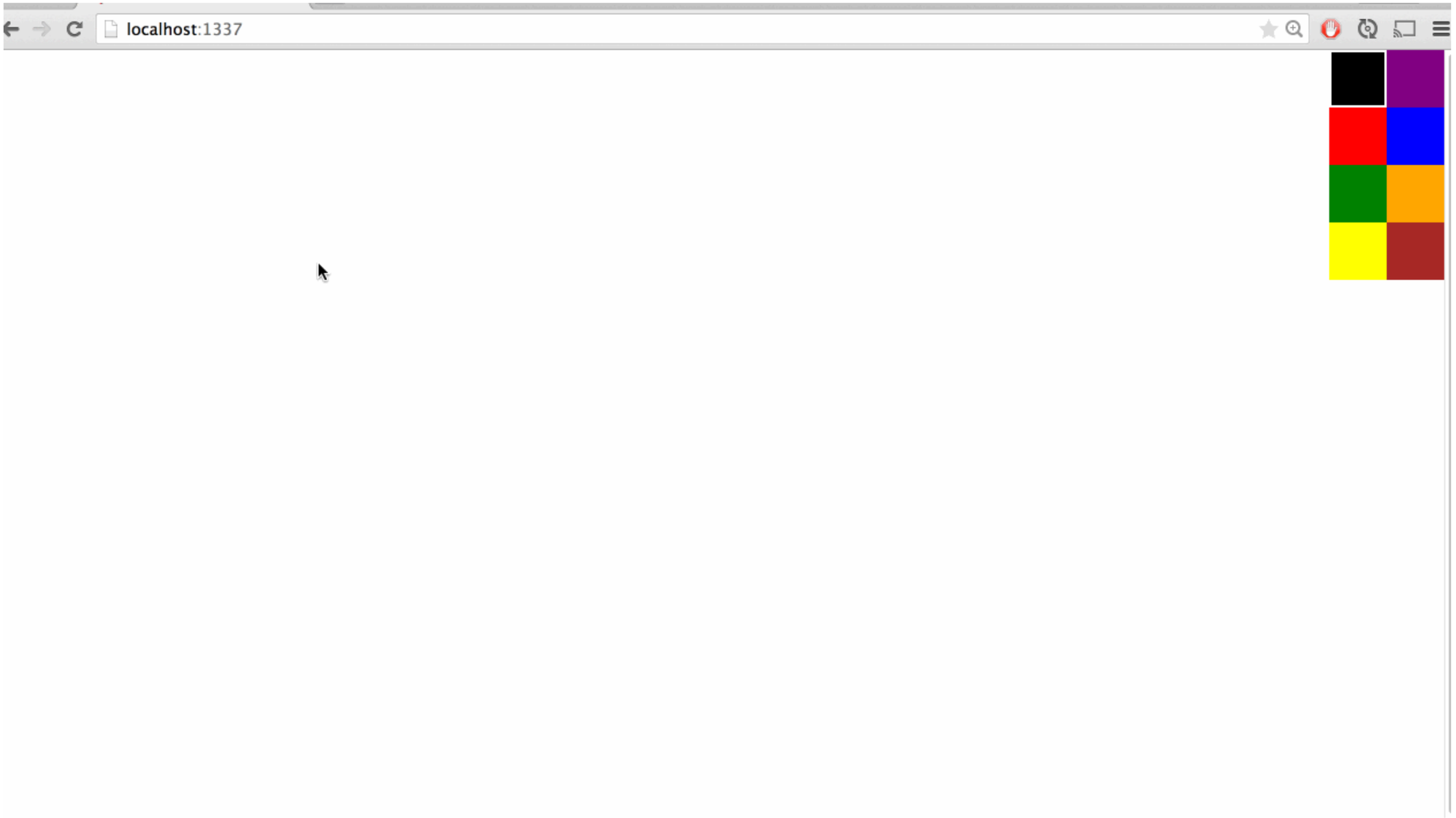
DRAWBACKS

- **The server now *must* hold on to the connection**
- **Connections are expensive (they require memory within the operating system)**
- **If a socket sits dormant for a long time, it's wasting server resources.**
  - You could fix this in your app, though! You have the power!

OTHER SOCKET.IO 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
- Can “broadcast” to all sockets within a “room”


localhost:1337


localhost:1337