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'
});
```

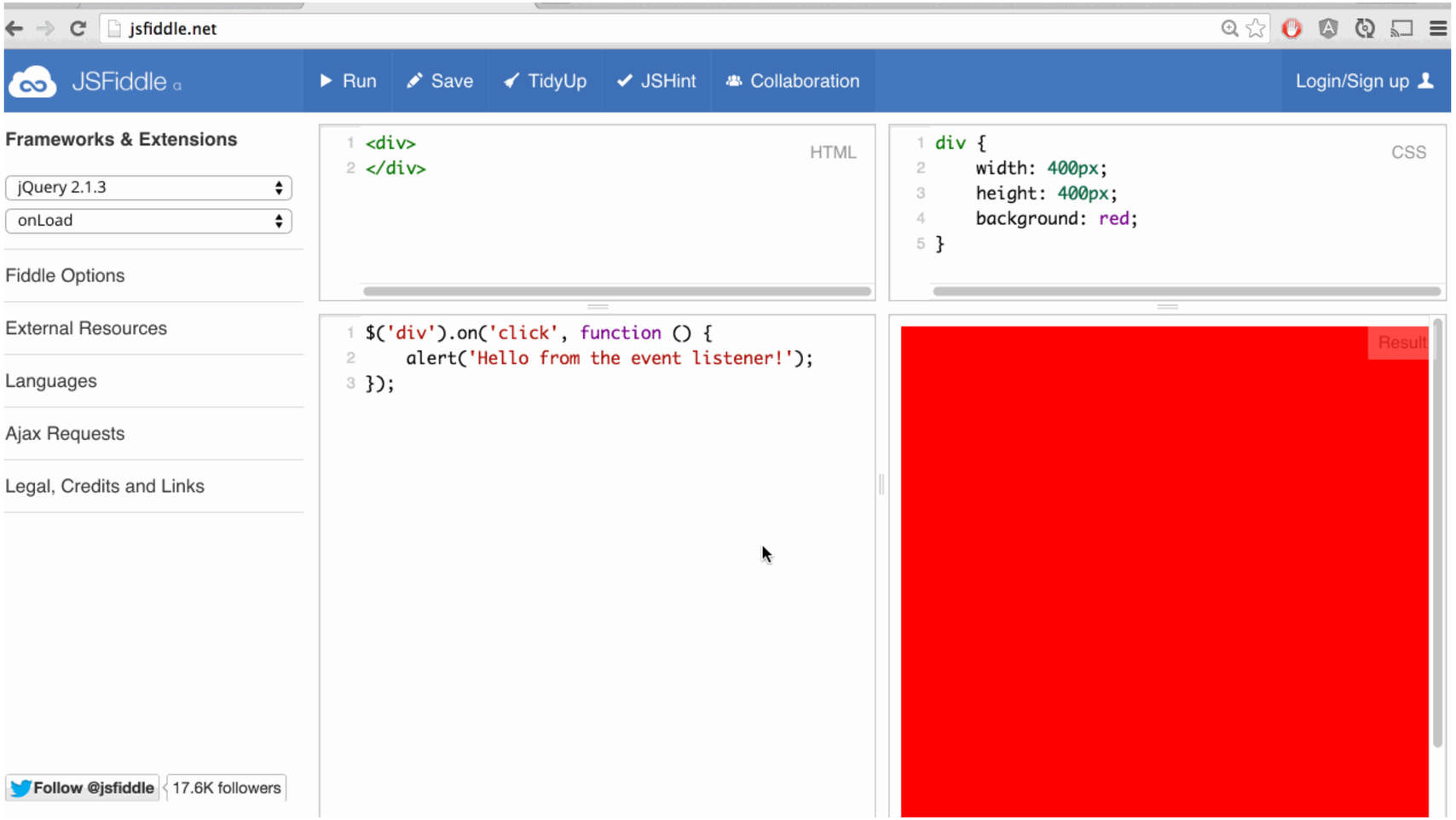
```
var userTweets = new EventEmitter();

// 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
- An instance of the “observer/observable” a.k.a “pub/sub” pattern
- Feels at-home in 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
HTML

CSS
div {
width: 400px;
height: 400px;
background: red;
}
$('div').on('click', function () {
alert('Hello from the event listener!');
});
Result

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) {

});
```

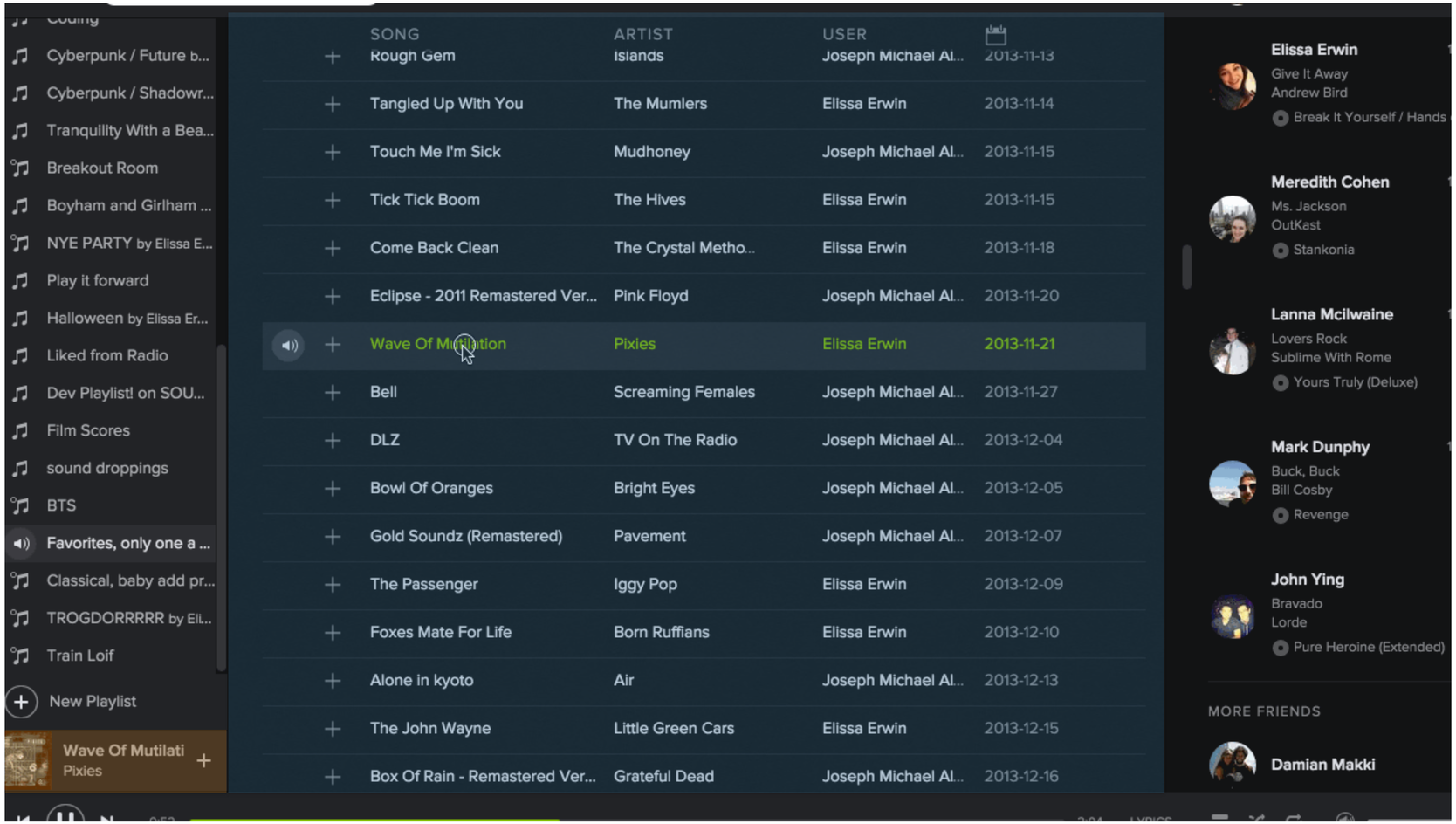
PRACTICAL USES

- Connect two decoupled parts of an application

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

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

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



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

- 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 an HTTP request
- Requests can include a body ("payload")
  - any data of any content type, e.g. JSON string, HTML, txt, jpg...
- A response can include a body ("payload")

The New York Times

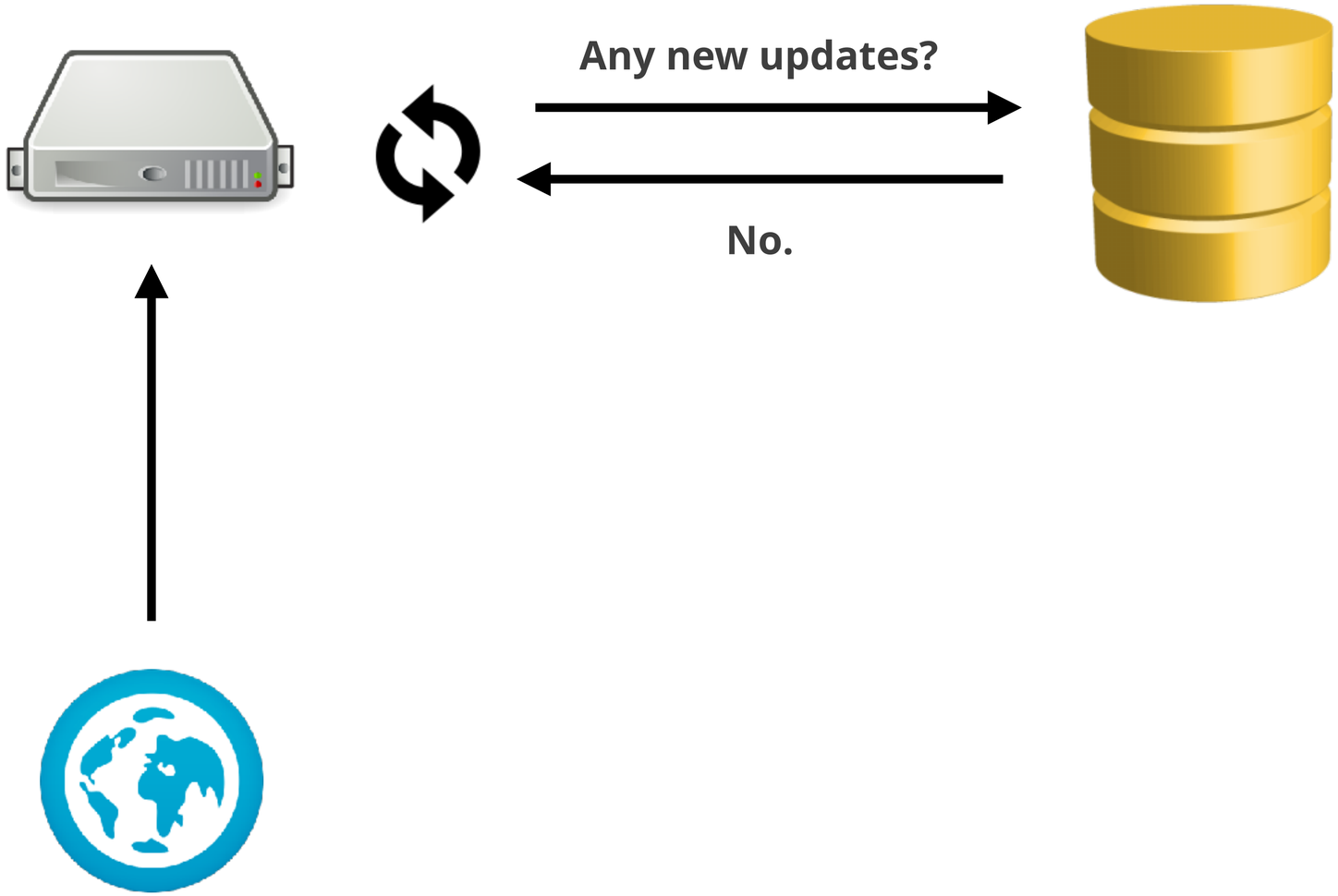
FIFA WORLD CUP
Brasil

LIVE WORLD CUP COVERAGE

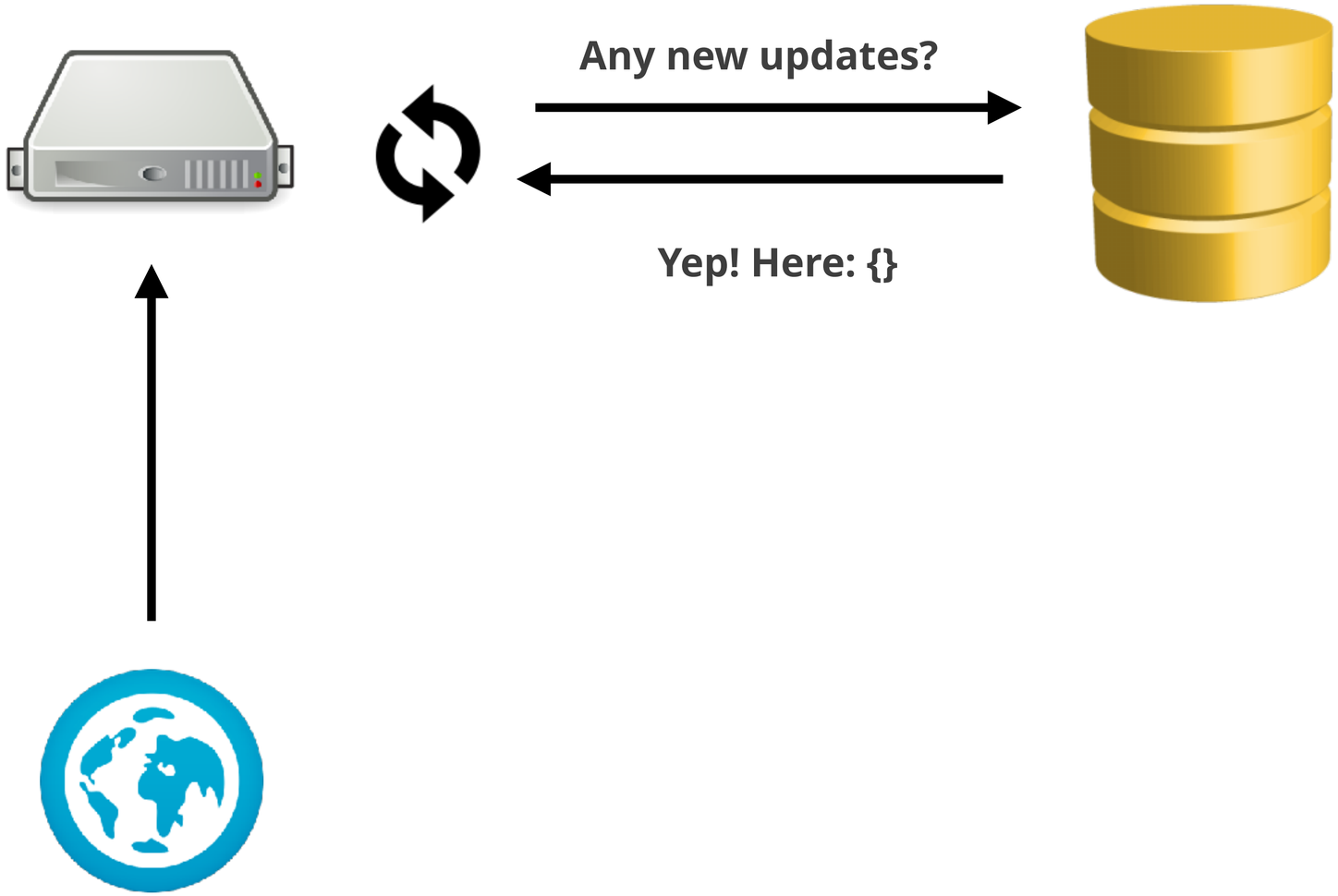
- A user visits a web page
- This web page has a live updating list of game coverage (“events”) provided by New York Times commentator (“Brazil receives yellow card”/“Germany scores goal”)
- When the event line is submitted by the commentator, it should immediately display to the user

HTTP LONG POLLING

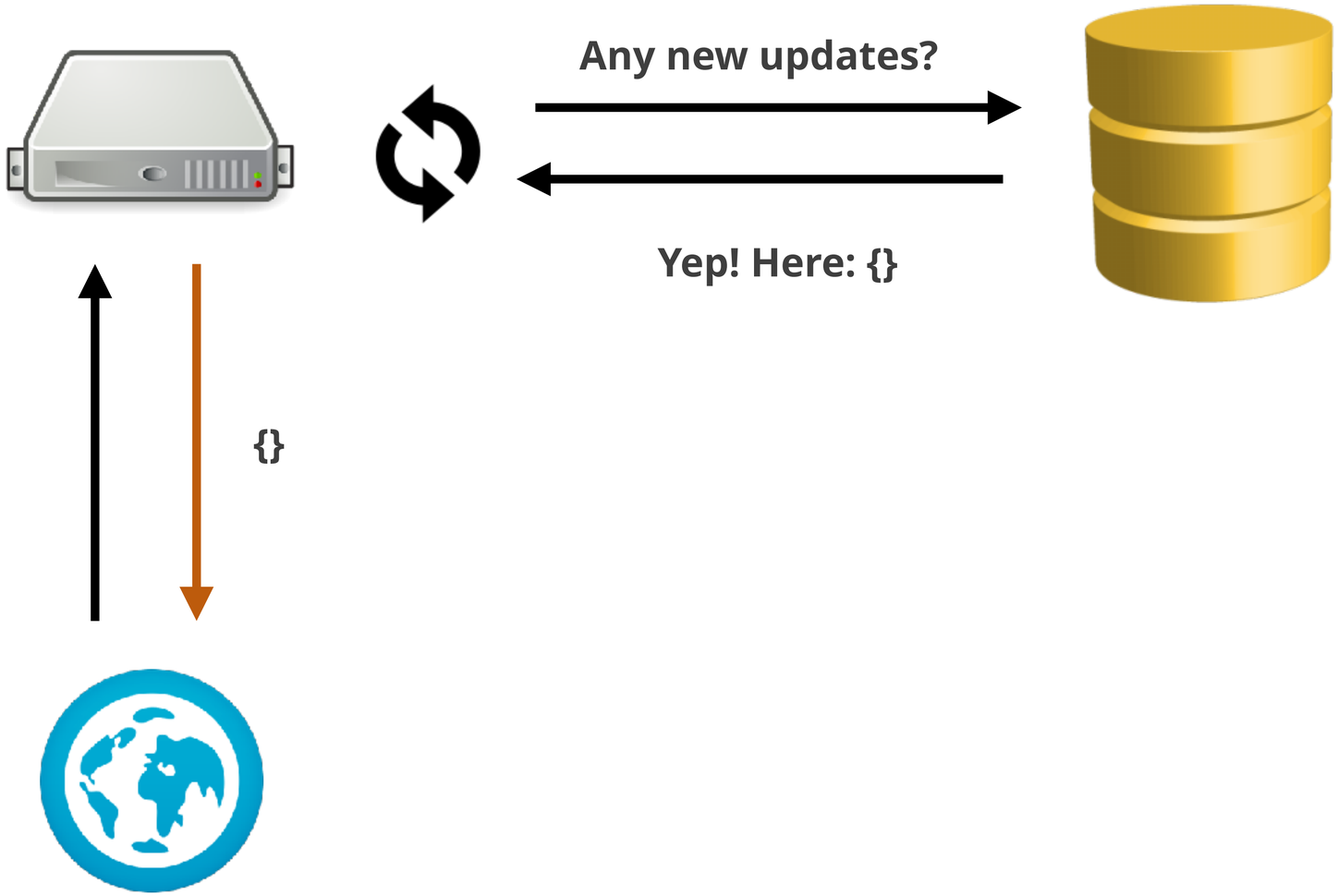
HTTP LONG POLLING



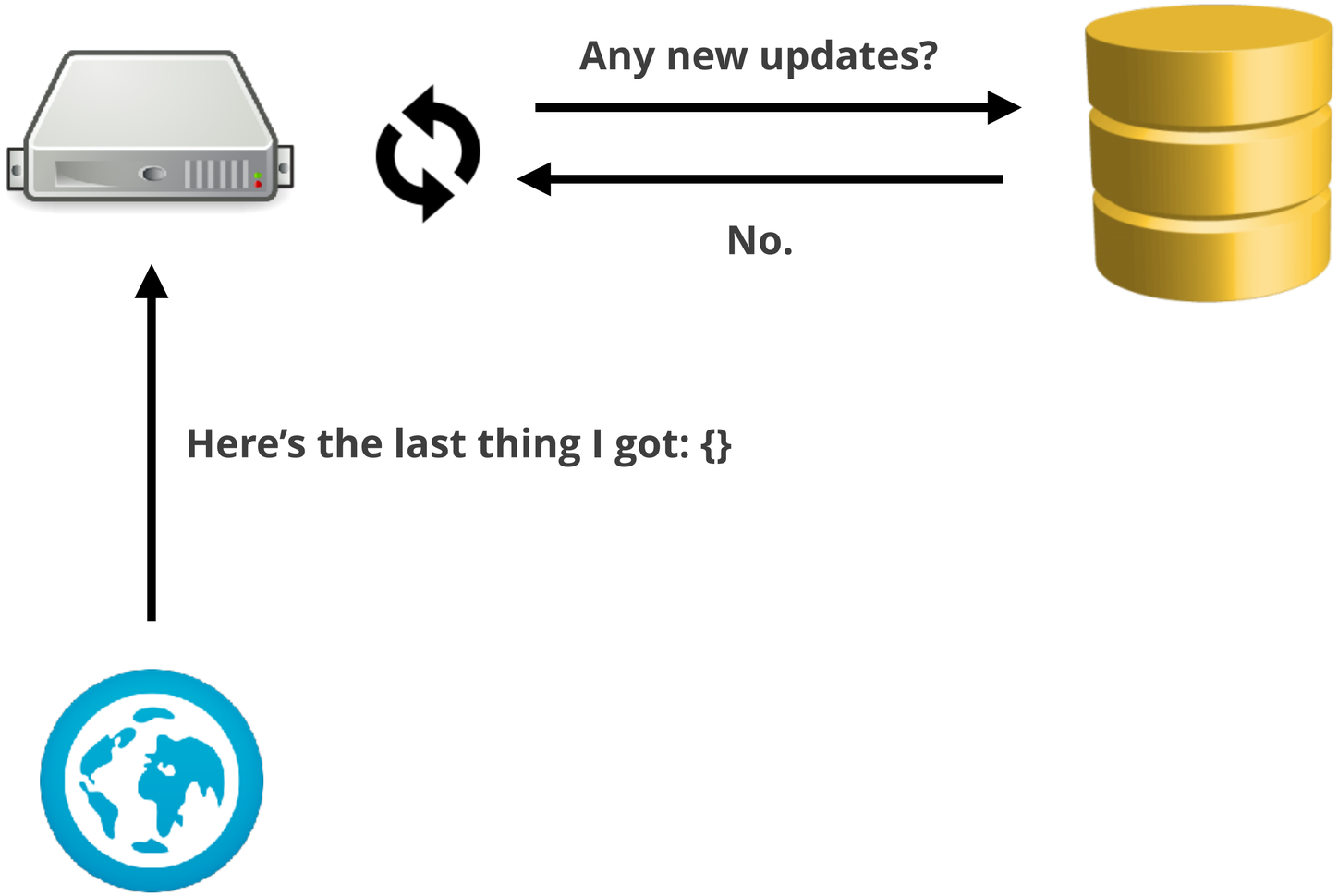
HTTP LONG POLLING



HTTP LONG POLLING



HTTP LONG POLLING



HTTP IS UNIDIRECTIONAL

COMMUNICATION MUST BE SOLICITED BY THE CLIENT

TCP

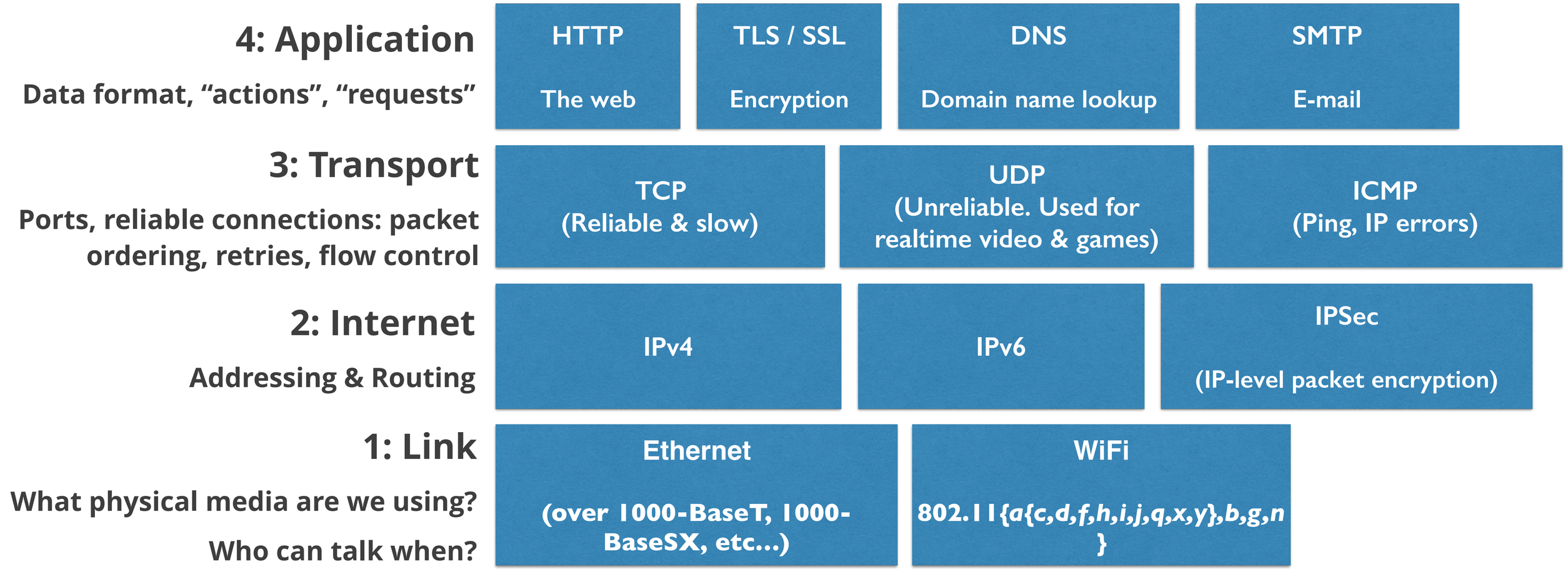
Transmission Control Protocol

TCP

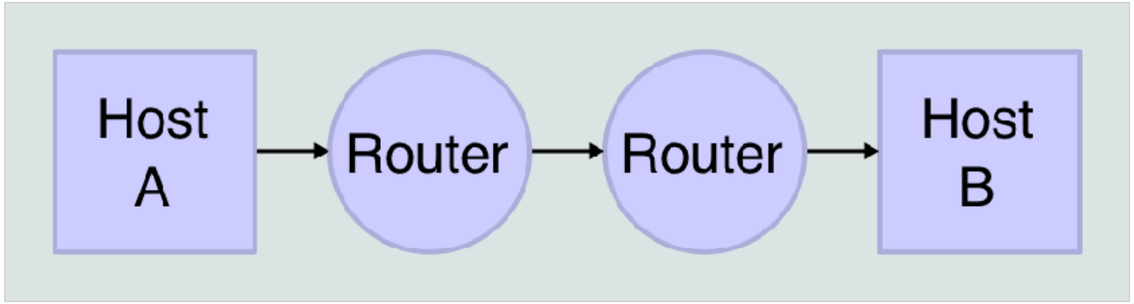
- **Protocol: standardized way that computers communicate with one another**
- **Establishes a reliable, duplex connection between two machines that persists over time**
  - **Reliable:** All your data gets there in the order you sent it
    - (or you know that it didn't)
  - **Duplex:** Either end of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

INTERNET PROTOCOL SUITE — LAYERS WITHIN LAYERS

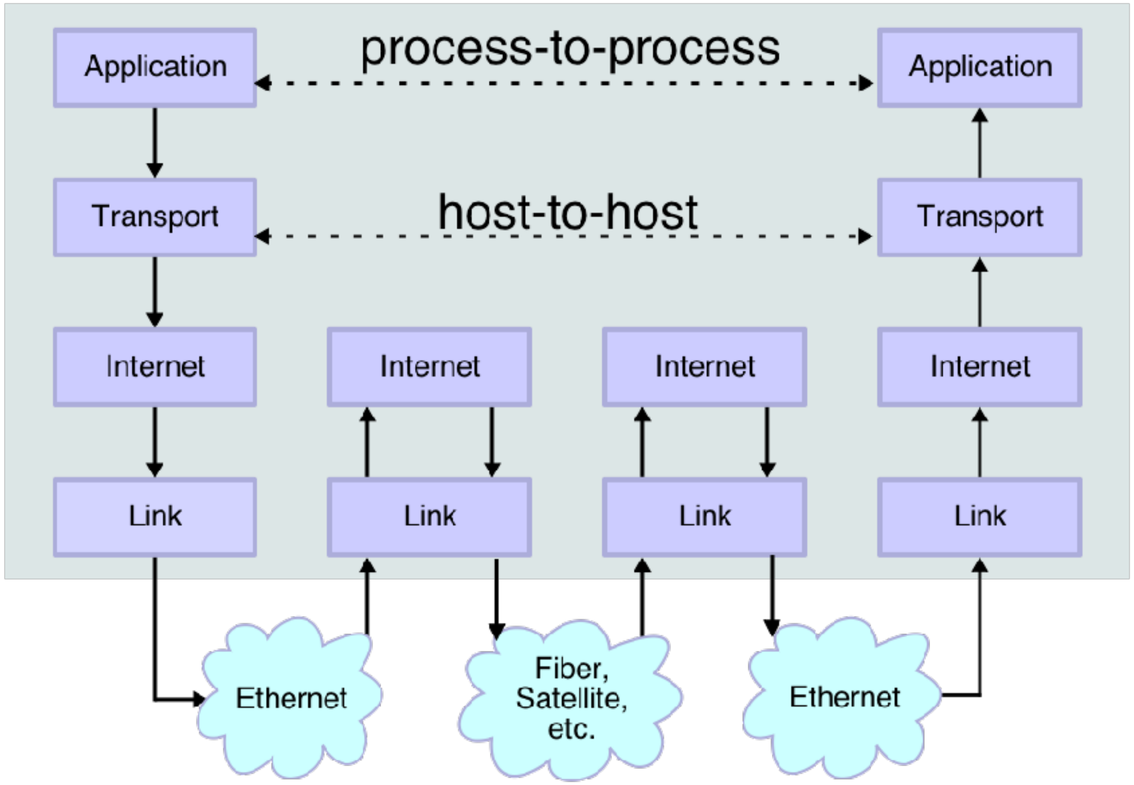
(opening into more layers)

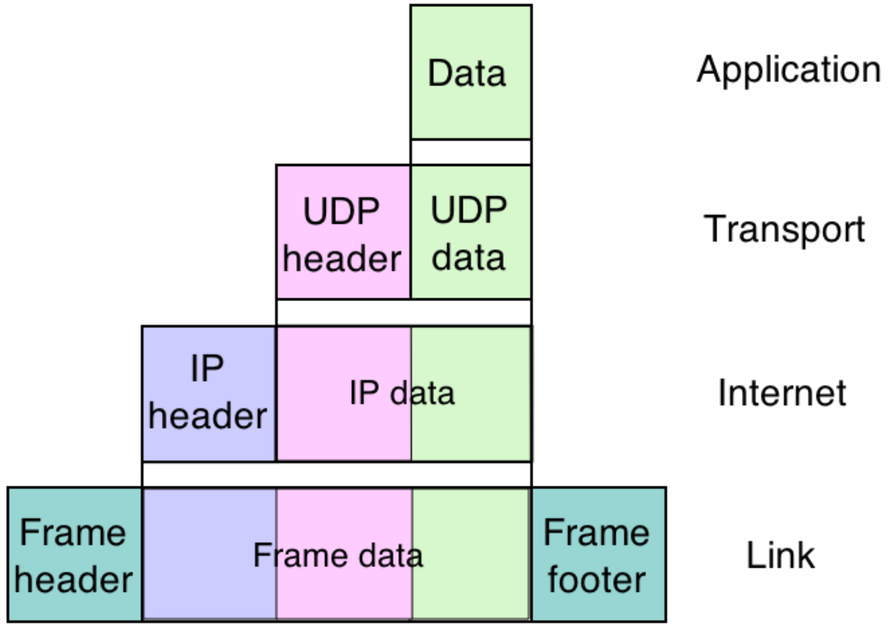


Network Topology



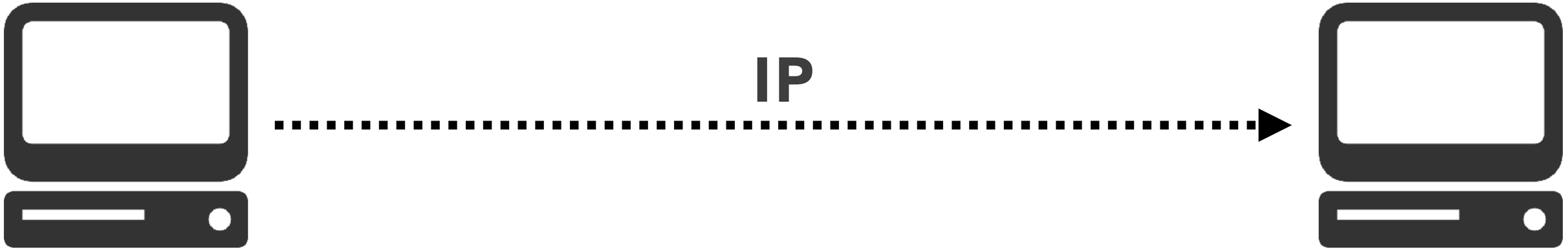
Data Flow





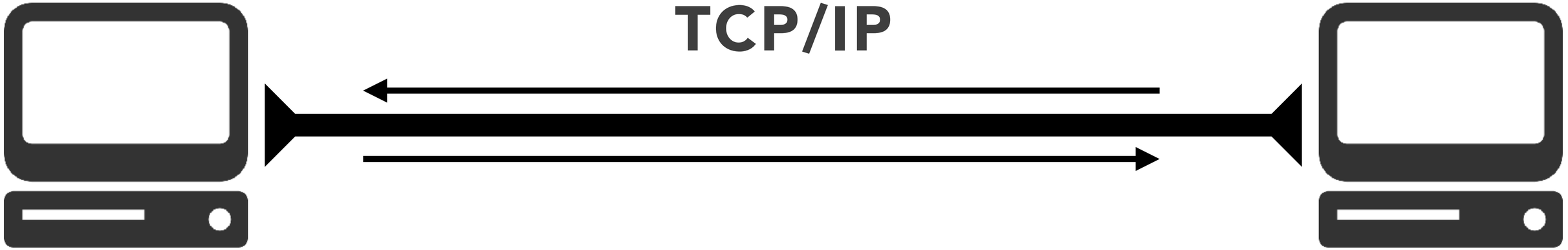
2604:2000:eecf:b700:493:e1bb:166c:f4a7

2607:f8b0:4006:80d::200e



- Machines have addresses
- Packets are sent & routed to their destination
- Do they get there? In what order? Nobody knows!
- What process was this message meant for? Unclear!
- Not a connection — a routing scheme

TCP/IP



- Connection: to figure out packet ordering & loss
- Retries & flow control: deal with packet loss
- Ports: which process gets the packet
- Reliable connection that persists over time

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

let's synchronize!

ok, let's synchronize!

ok!

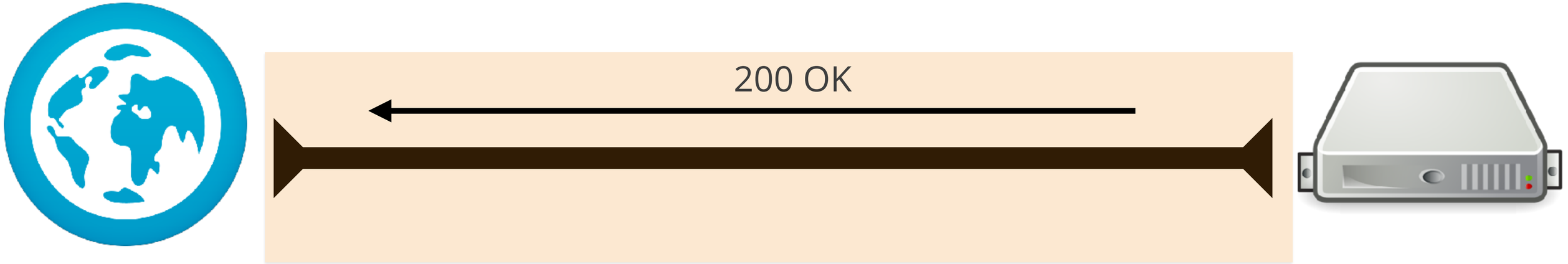
TCP CONNECTION IS ESTABLISHED

CLIENT SENDS A REQUEST

(over the connection)

SERVER SENDS A RESPONSE

(over the connection)



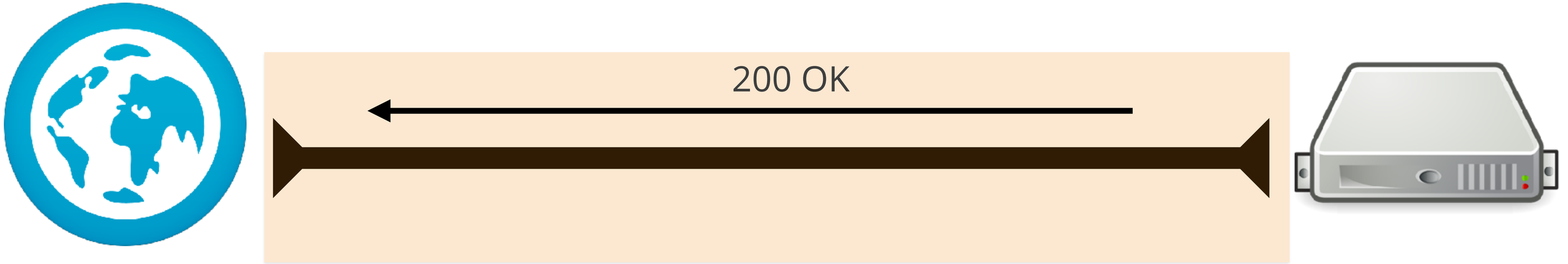
TCP CONNECTION STAYS OPEN

CLIENT SENDS MORE REQUESTS

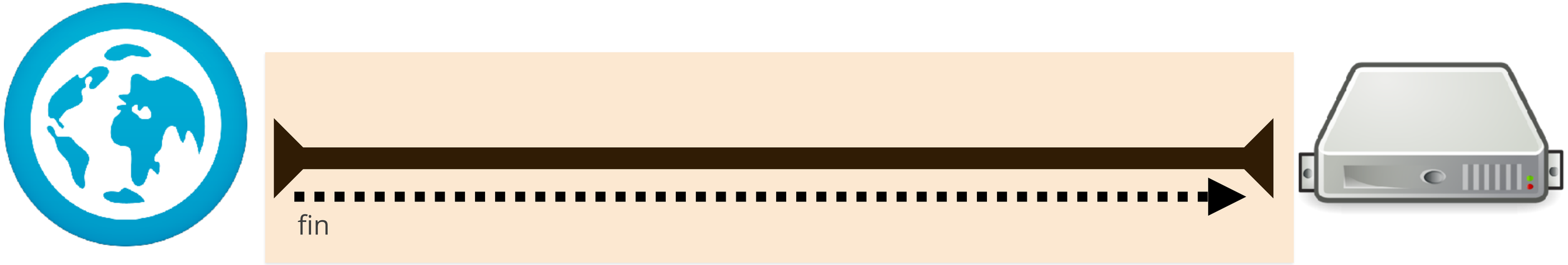
(over the same connection)

SERVER SENDS MORE RESPONSES

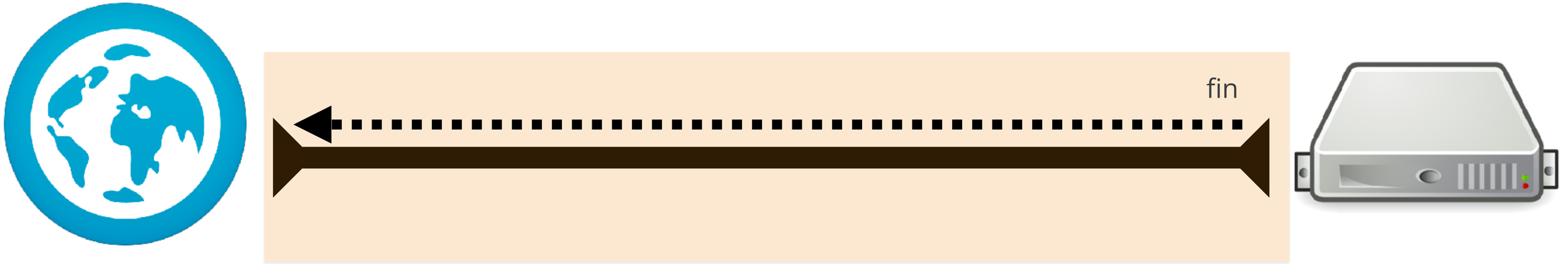
(over the same connection)



EVENTUALLY, YOU CLOSE THE TAB



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



AND ONE OF YOU ENDS THE CONNECTION

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

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

WEBSOCKET

- Application-layer protocol
- Message-based
- Either the client or server can choose to send a message at any time
  - No “requests” or “responses” unless *you* design the protocol for them
- Allows for awesome real-time software
- So how do you open a WebSocket?

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

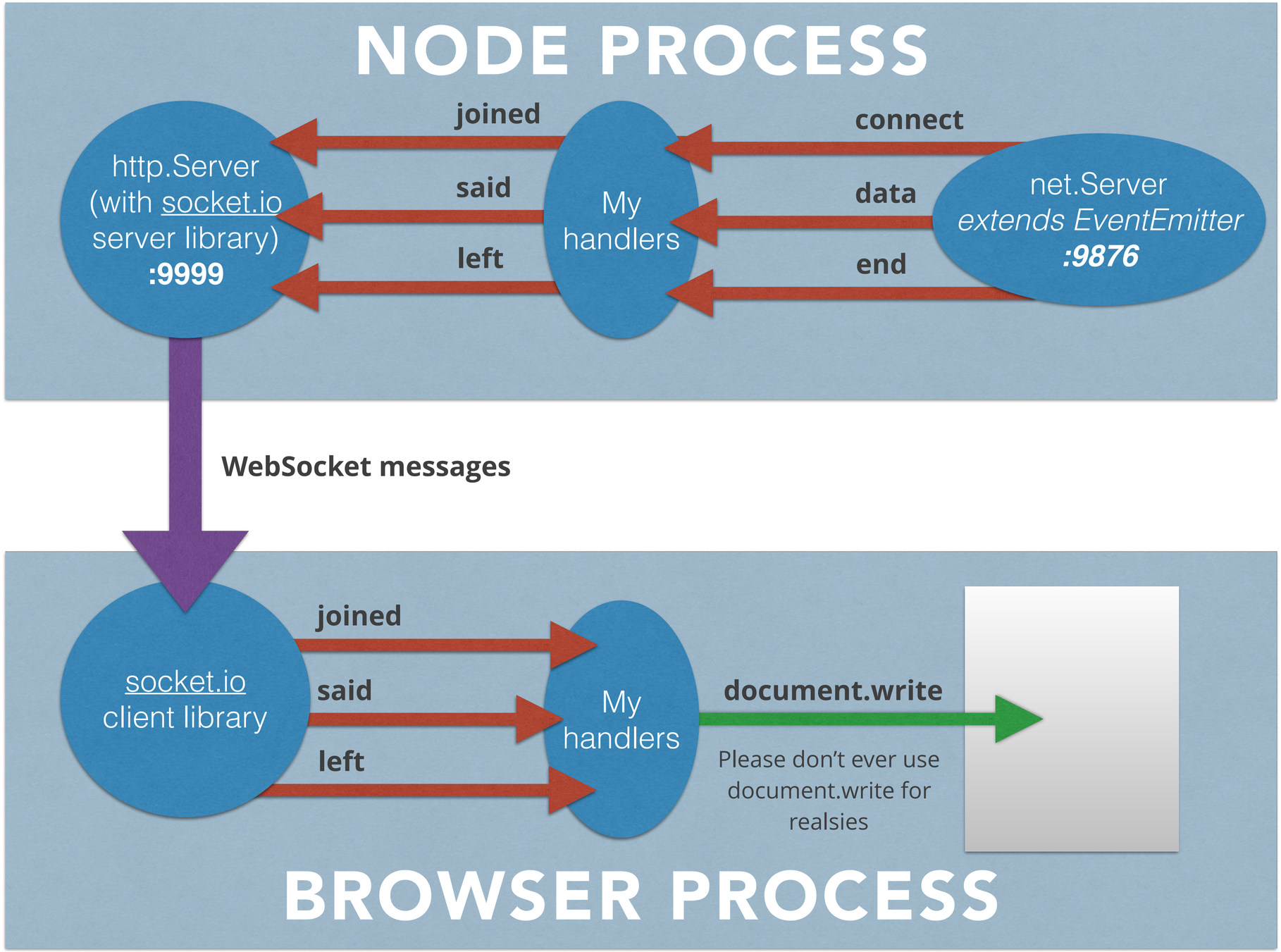
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

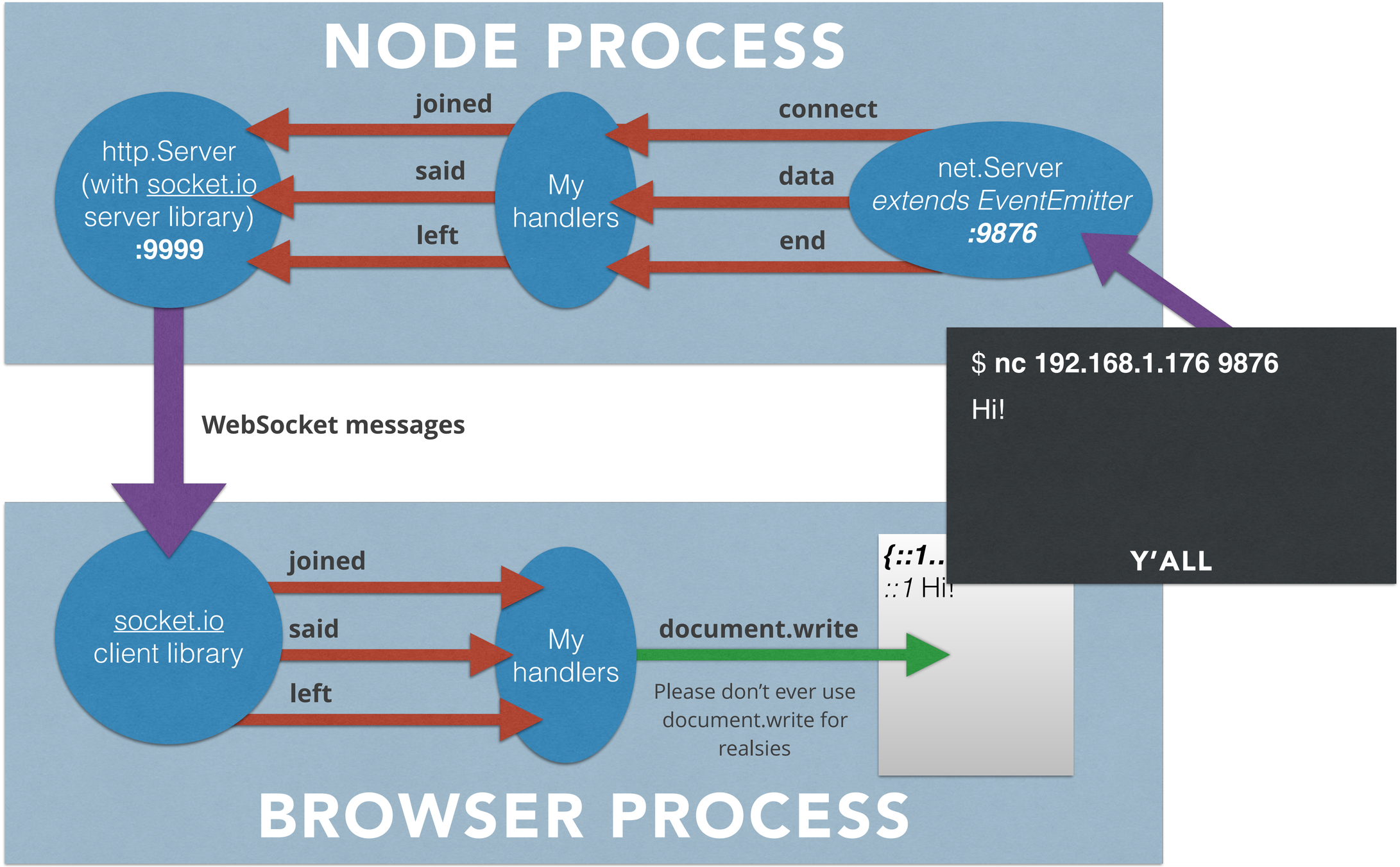
Sec-WebSocket-Protocol: chat

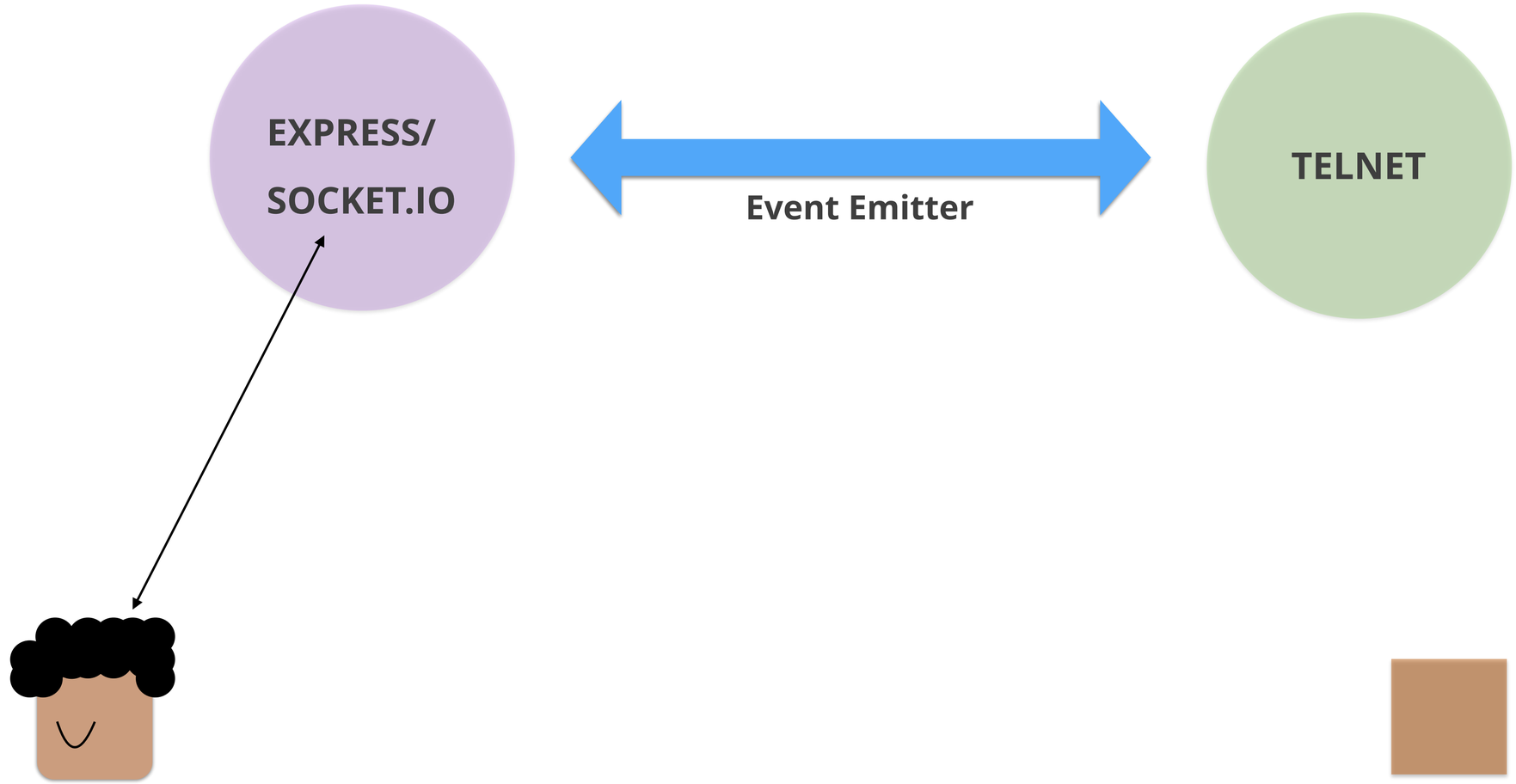
And now WebSocket has taken over the connection.

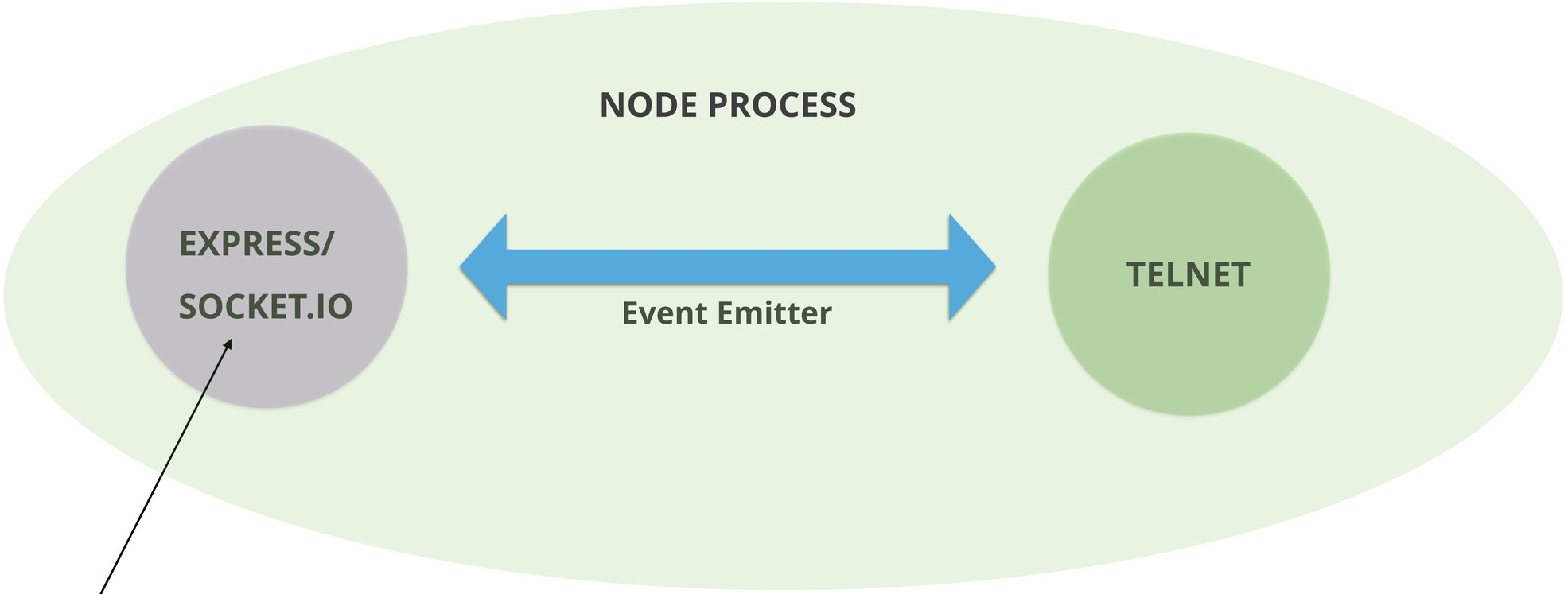
SOCKET.IO

- 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 EventEmitters
  - EventEmitters are a good fit for a message-based protocol


NODE PROCESS
http.Server
(with socket.io
server library)
:9999
joined
said
left
My
handlers
connect
data
end
net.Server
extends EventEmitter
:9876
WebSocket messages
socket.io
client library
joined
said
left
My
handlers
document.write
Please don't ever use
document.write for
realsies
BROWSER PROCESS

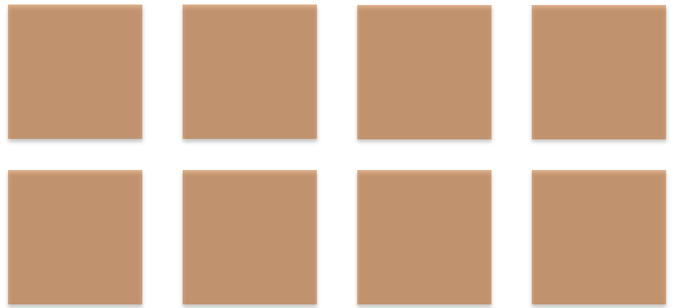

NODE PROCESS
http.Server
(with socket.io
server library)
:9999
joined
said
left
My
handlers
connect
data
end
net.Server
extends EventEmitter
:9876
$ nc 192.168.1.176 9876
Hi!
Y'ALL
WebSocket messages
socket.io
client library
joined
said
left
My
handlers
document.write
Please don't ever use
document.write for
realsies
{::1..
::1 Hi!
BROWSER PROCESS


EXPRESS/
SOCKET.IO
Event Emitter
TELNET
KEYNOTE JOE AND HIS BROWSER
Y'ALL

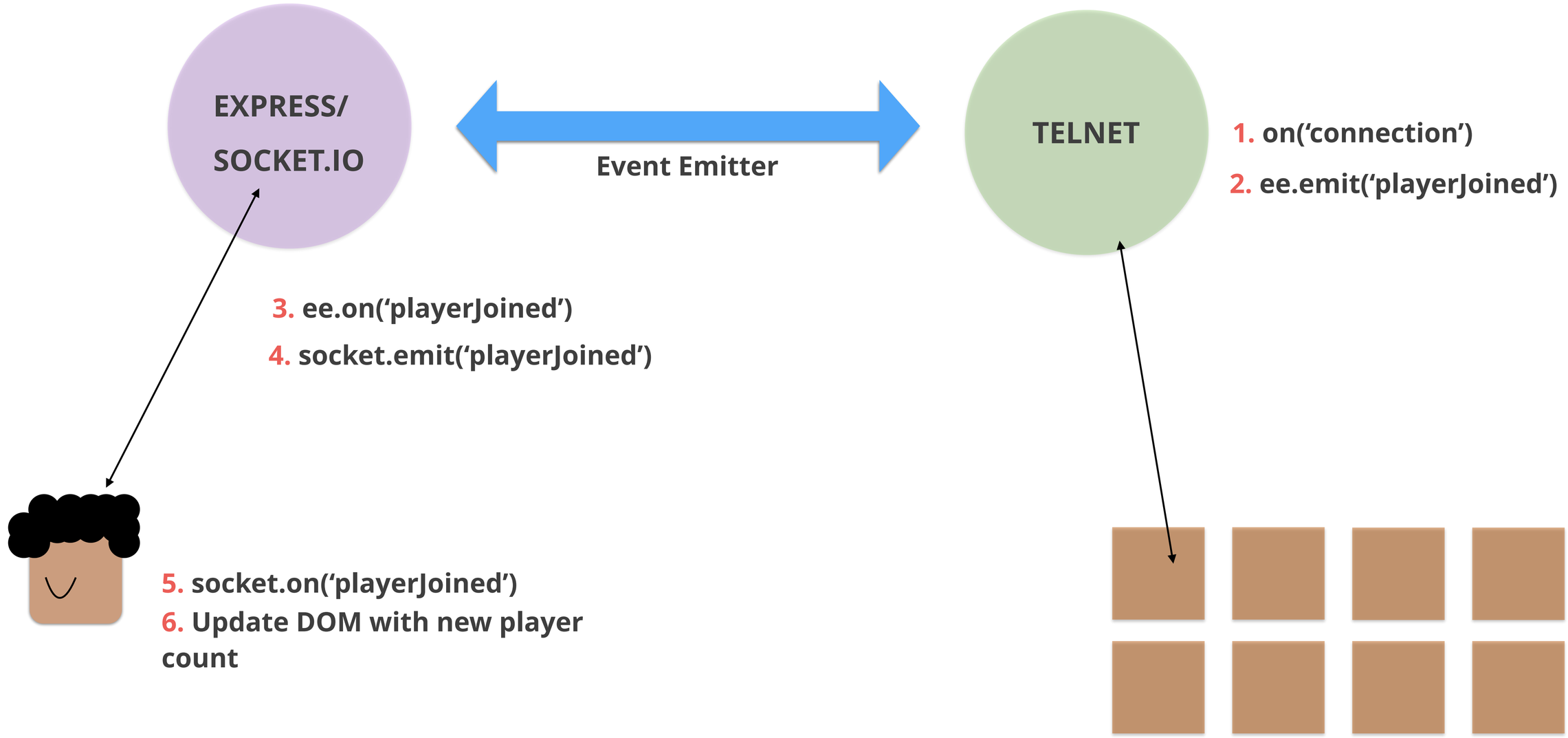


KEYNOTE JOE AND HIS BROWSER

Y'ALL



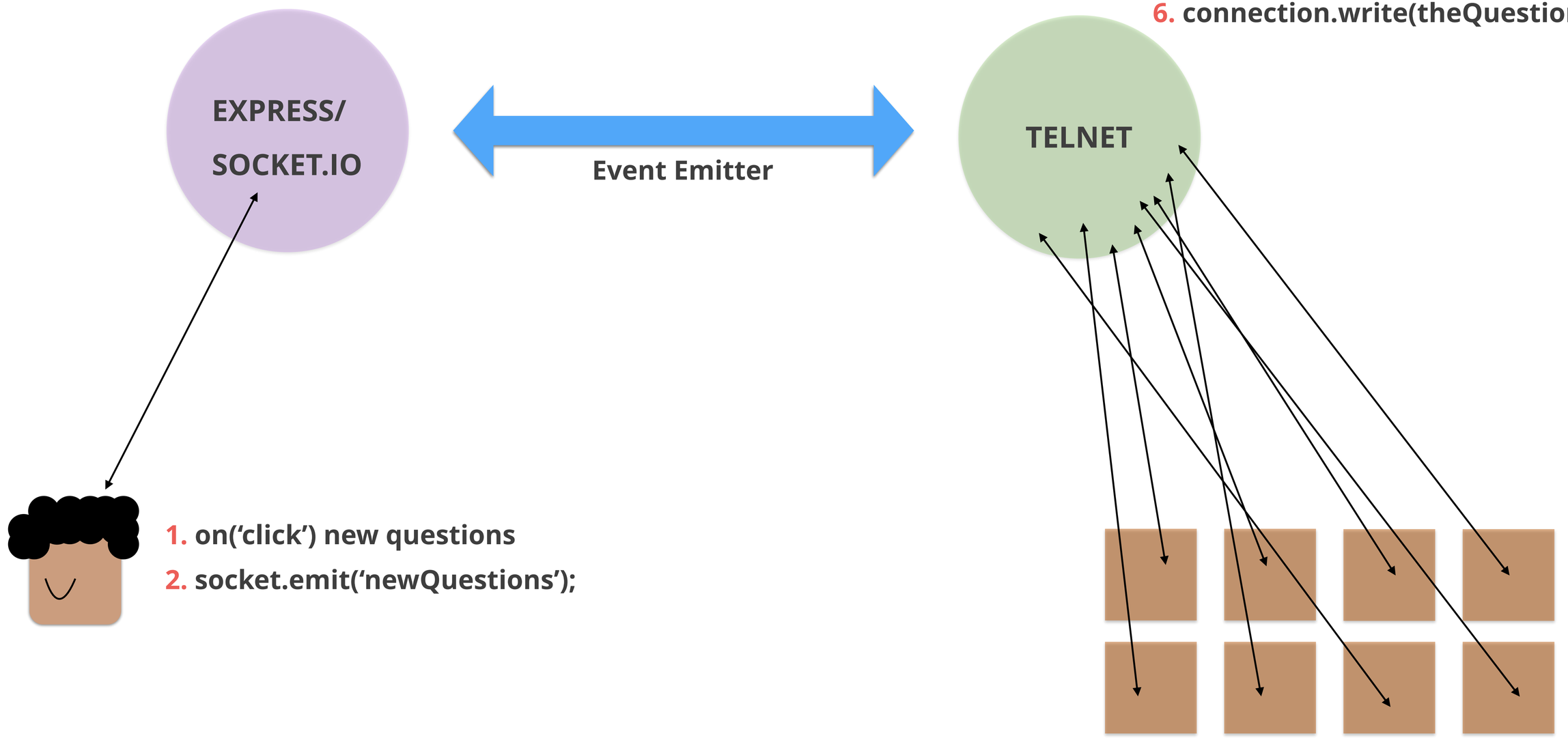
PLAYER JOINS



NEW QUESTIONS

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

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



ANSWERS

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

3. ee.on(‘answer’)
4. socket.emit(‘answer’)

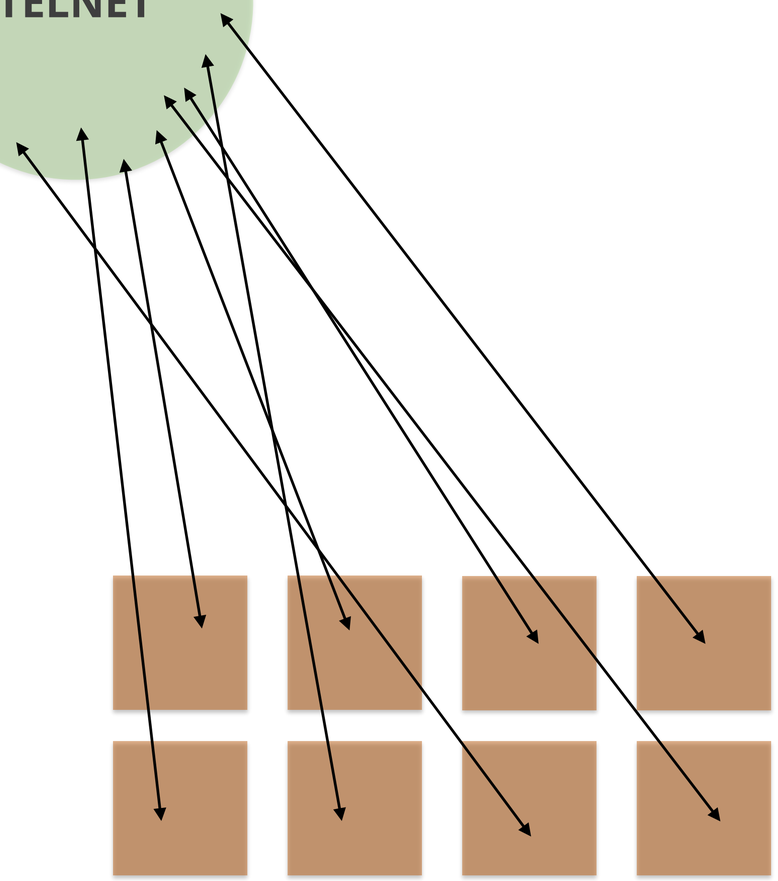
EXPRESS/

SOCKET.IO

Event Emitter

TELNET

5. socket.on(‘answer’)
6. Add to DOM



USE CASES

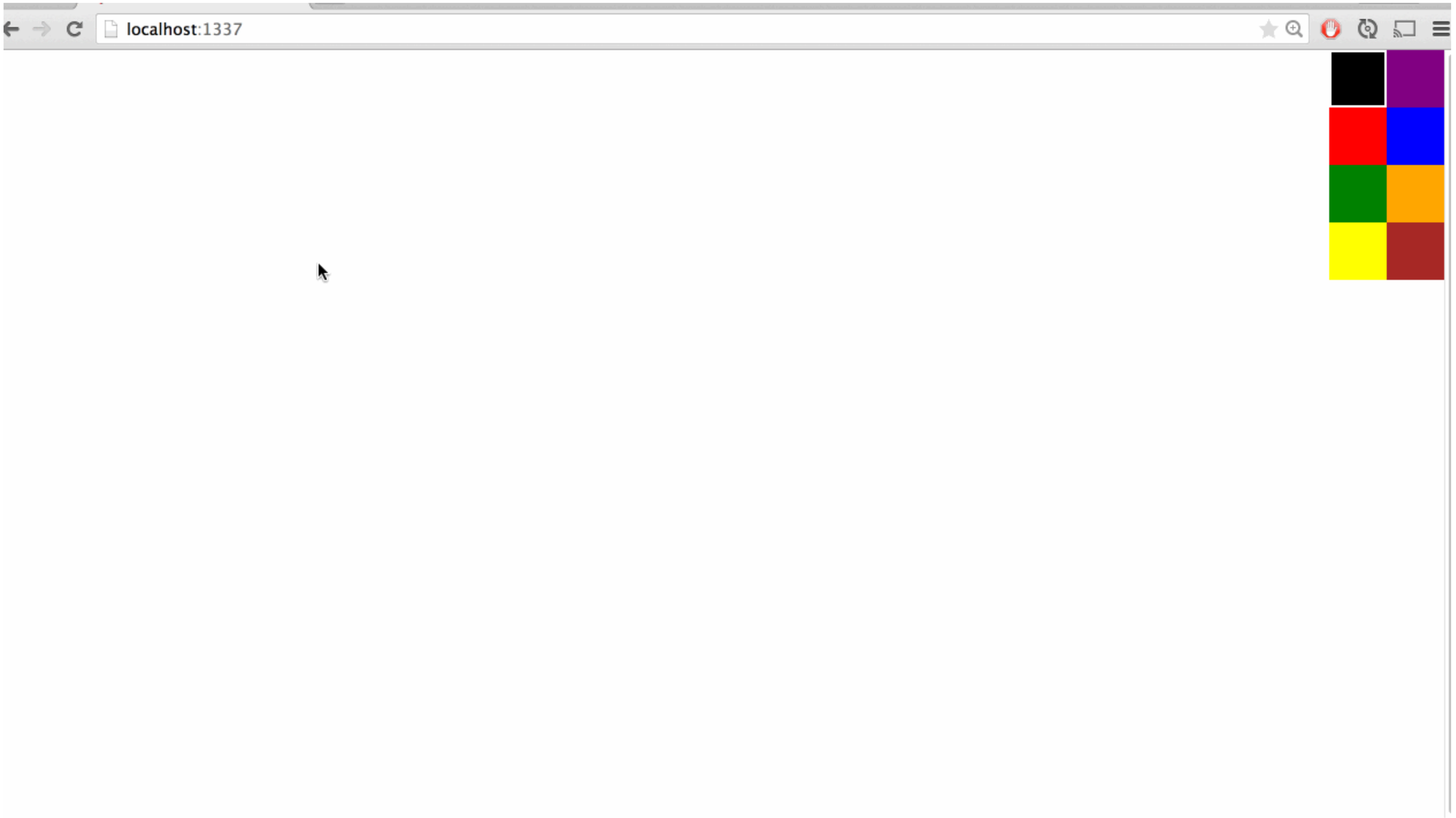
- Networked enabled games
- Chat applications
- Collaborative applications
- Any “real-time” software

DRAWBACKS

- 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