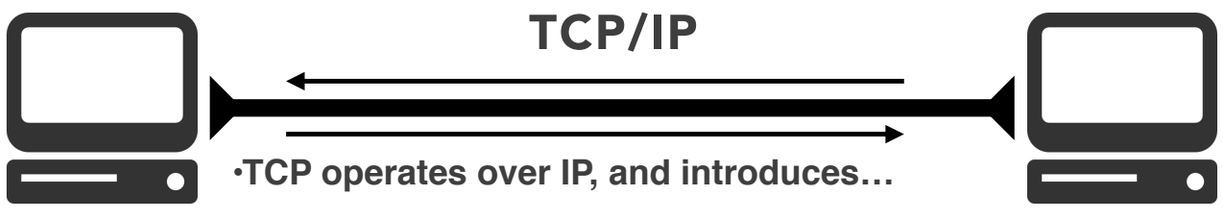
TCP, SOCKETS & SOCKET.IO

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**



- TCP operates over IP, and introduces…
- Ports: to figure out which process gets the packet
- Connections: to figure out packet ordering & loss
- Retries & flow control: to deal with packet loss
- Reliable connection that persists over time

TCP AND HTTP

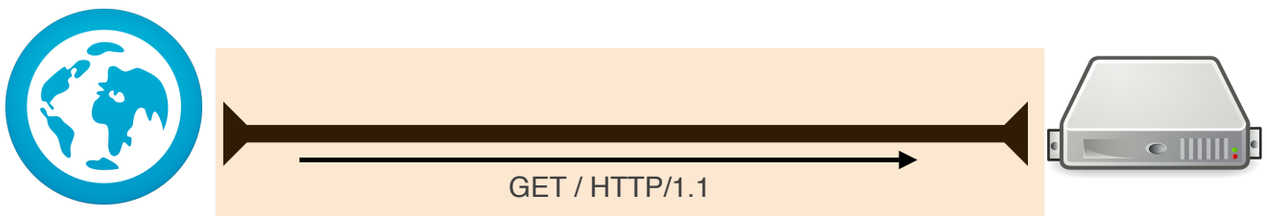
- **HTTP is an application layer protocol**
- **It (usually) operates over TCP, (usually) on port 80**
  - But “HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used” — HTTP 1.1 Spec
  - HTTPS, for instance, operates over TLS on port 443
- **Implements the idea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

TCP CONNECTION IS ESTABLISHED

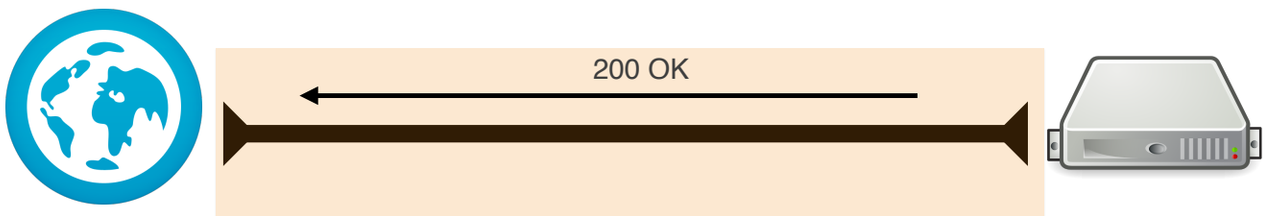
CLIENT SENDS A REQUEST

(over the connection)



SERVER SENDS A RESPONSE

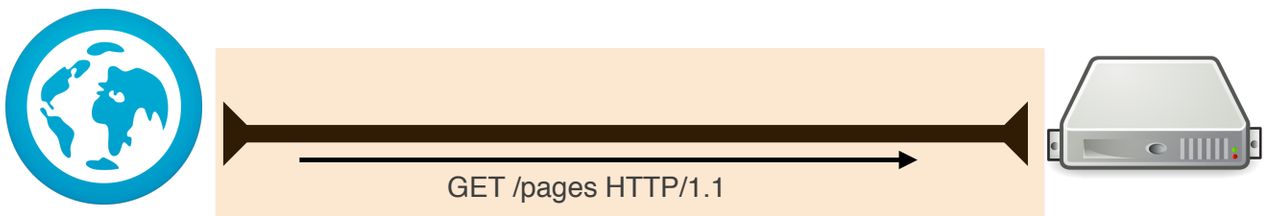
(over the connection)



TCP CONNECTION STAYS OPEN

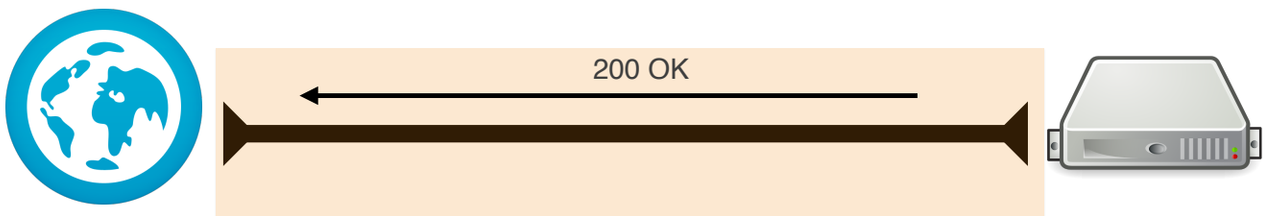
CLIENT SENDS MORE REQUESTS

(over the same connection)



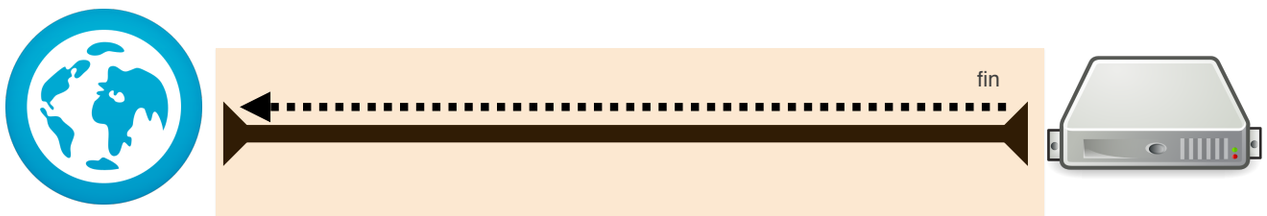
SERVER SENDS MORE RESPONSES

(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 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:

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