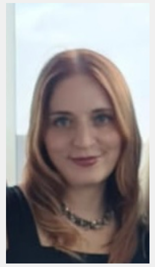
COM3504/6504 The Intelligent Web

Lecture 5: Socket.io



Dr Vitaveska Lanfranchi
v.lanfranchi@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/academic/vitaveska-lanfranchi

Mr. Andy Stratton
a.stratton@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/academic/andrew-stratton

Dr Fatima Maikore
F.Maikore@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/research-staff/fatima-maikore

Learning Objectives

- The learning objectives for this lecture are:
  - Learn about full-duplex bidirectional communication
  - How socket.io allows to implement it

BI- DIRECTIONAL SERVER COMMUNICATION

What is it?

- In the past lecture we have seen HTTP requests
- HTTP requests rely on a client request to receive a response from the server for every exchange
- There are cases where we might want full bi-directional communication
  - This enables the server to send real-time updates asynchronously
  - without requiring the client to submit a request each time
  - Allows to push data from the server to the client

Differences with Ajax/HTTP requests

- Ajax and HTTP requests allow to simulate bi-directional communication
  - Polling

    The client regularly sends AJAX request (i.e., every few seconds),

    If there's new data, the server sends it in the response.
  - Streaming

    a variety of techniques (multipart/chunked response) that allow the server to send more than one response to a single client request

    The client opens a HTTP with the server,

    The server sends HTTP headers, does not close connection

    When new data arrives, the server sends it in the response body;

Websockets

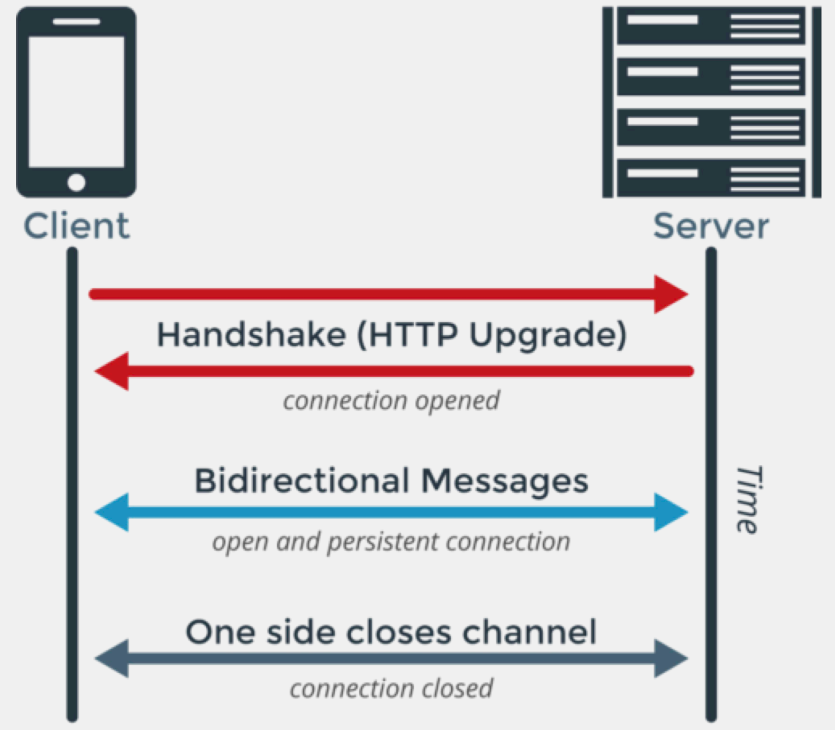
- Websockets are a Full-duplex communication protocol over TCP
- HTTP-compatible
  - designed to run on HTTP ports 80 and 443
- A WebSocket handshake uses the HTTP Upgrade header to switch from the HTTP protocol to the WebSocket Protocol.

How do they work?

- A socket is a channel of communication between processes (on the same or different computers)
  - They create a persistent connection between the client and the server and both parties can start sending data at any time
- A Websocket establishes communication between two processes on different web connected machines via the TCP protocol
- After an initial handshake request/response
  - The client and server may send messages at any time and must handle message receipt asynchronously.

A web socket is event-based

- The process waits for an event
  - the receiving of a communication
- the process can raise events at any time on the partner machine
  - by sending data via the socket



https://medium.com/platform-engineer/web-api-design-35df8167460

Advantages of websockets

- Most browsers support WebSocket protocol
- Most languages provides a library implementing the WebSocket API
- Speed
  - as they are based on small messages and persistent connection

SOCKET.IO

What is Socket.io?

- A library built on top of the WebSocket protocol
- Allows bidirectional, low-latency, event-based communication
- Abstraction layer
    - Hides the complexity of websockets
- There are alternatives, i.e.
    - https://github.com/websockets/ws

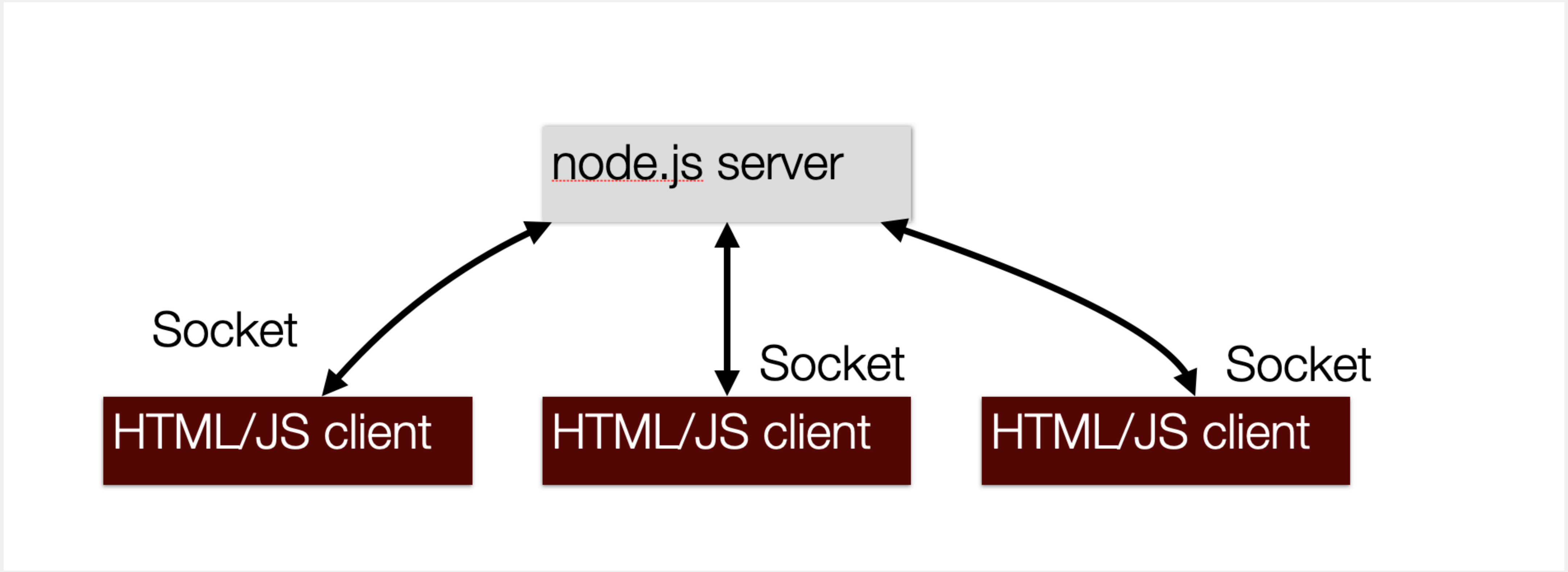
Advantages of Socket.io

- It works on every platform, browser or device, focusing equally on reliability and speed
- It has two parts:
  - a client-side library that runs in the browser,
  - a server-side library

    We will use the one for node.js.
  - Both components have a nearly identical API
- It is possible to send any data,
  - Including blobs, i.e. Image, audio, video

The University Of Sheffield.

1 server, n clients, n sockets



A WORLD TOP 100 UNIVERSITY

How does it work – on the server

- You need to require socket.io attached t your HTPP server

-
  ```
  //create socket.io
  const io = require('socket.io')(server);
  ```
- creates a new socket.io instance attached to the http server
- create a server that listens to socket.io connections
  - **io.on event handler** handles connection, disconnection, etc., events in it, using the socket object.

```
io.on("connection", (socket) => {
console.log("New Client is Connected!");
});
```

How does it work – on the server – cont.

- Once the user is connected to the Server can send a message to that client using the emit event
  - a welcome message by emitting a welcome event

```
io.sockets.emit("welcome", {description: "Hello and Welcome to the Server"});
```

- Then the server can send other messages to all clients by using the broadcast event

How does it work – on the client

- On the client side you need to write code that responds to events
- For each event we create listeners with callbacks
- You can use the socket.io javascript library
  - (I used it from CDN, you can download and add)

```
<script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
```

How does it work – on the client – cont.

- First of all you need to create a socket object

```
var socket = io();
```

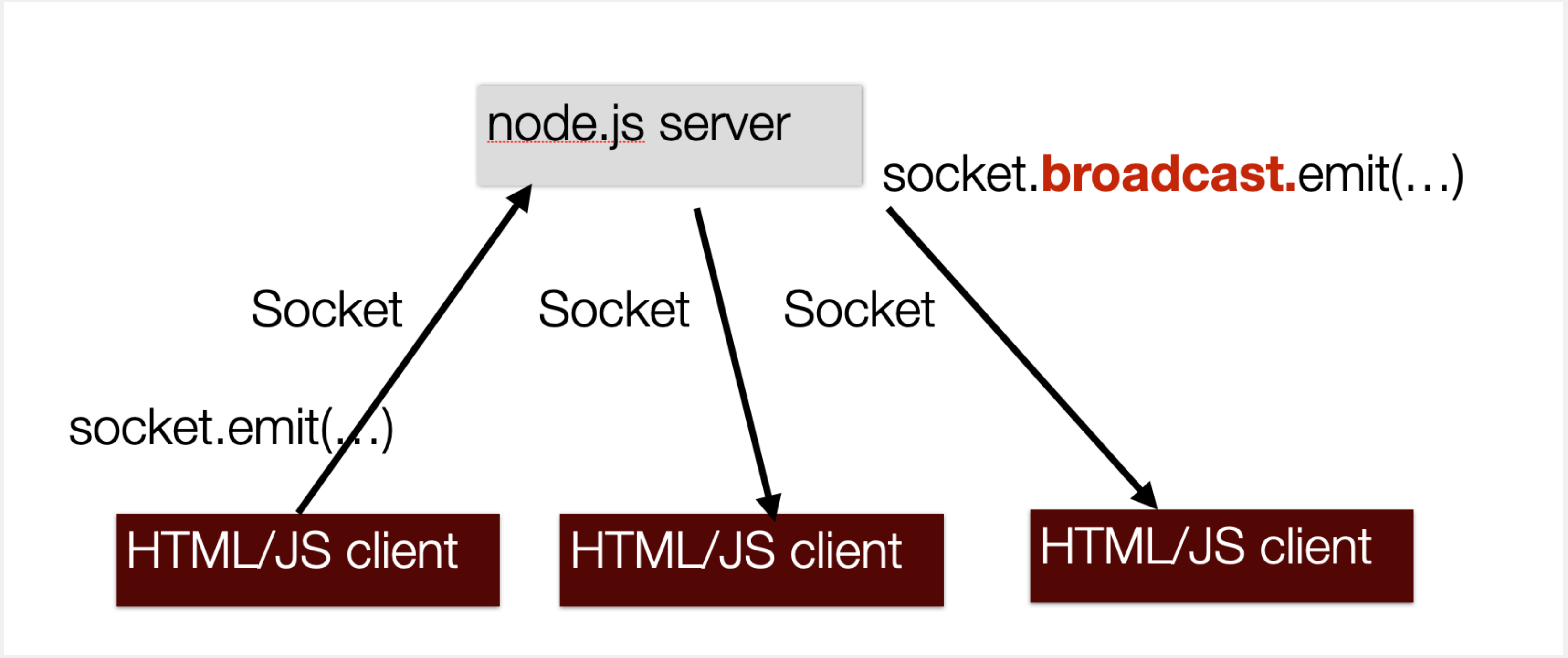
- We create an event listener that write into an element of my index.ejs page

```
socket.on('welcome',function(data){
 document.getElementById('Welcome').innerHTML = data.description;
});
```

```
<h1 id="Welcome"></h1>
```

Broadcasting

- Broadcasting means sending a message to everyone else
  - except for the socket that starts it
- Communication is not returned to the originating client



Create rooms

- A room is a subchannel that sockets can join or leave
- Messages can be sent only to clients connected to that room
- To join a room

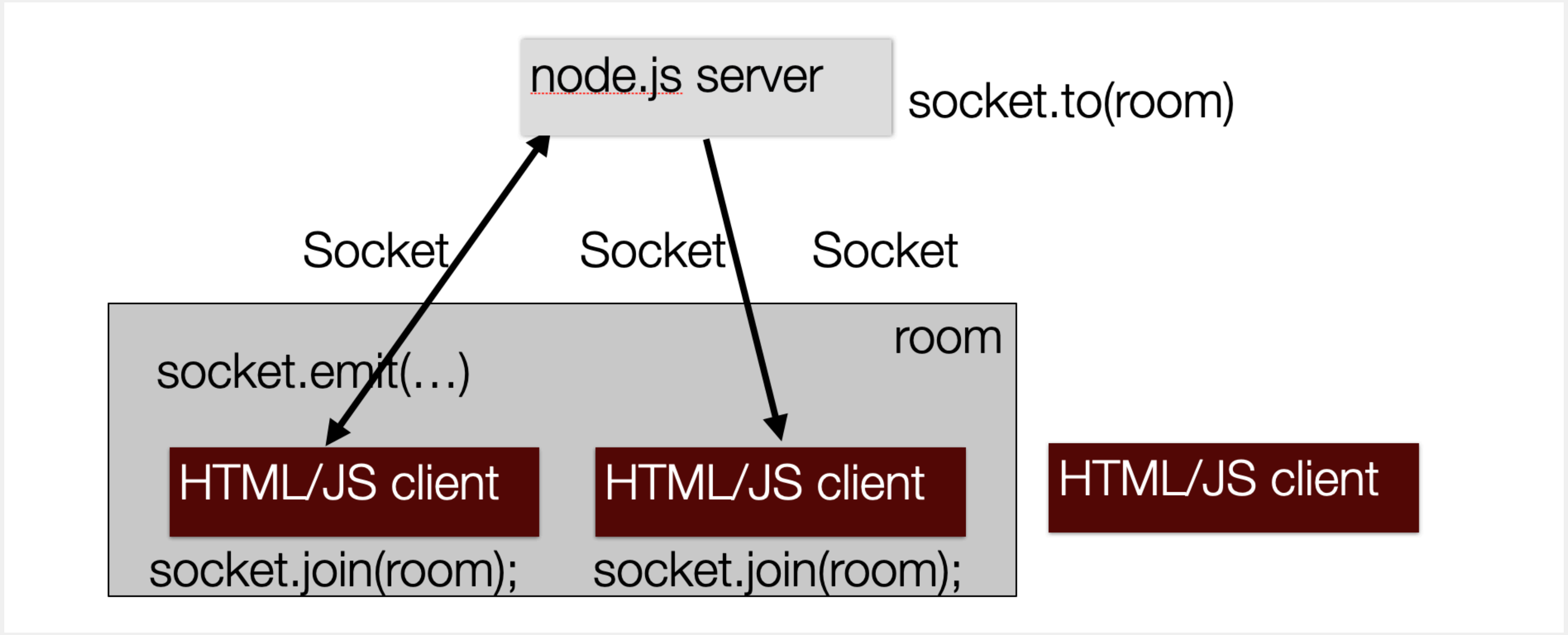
```
io.on("connection", socket => {
socket.join("some room");
});
```

- To message everyone in a room

```
socket.to("roomID").emit
```

In a room

- Once you are in a room, socket.emit(…) just reaches those in the same room



Joining a room

- Client side:

```
function connectToRoom(roomNo, name) {
    socket.emit('create or join', roomNo, name);
}
```

- Server side:

```
socket.on('create or join', function (room,
userId) {
  socket.join(room);
 });
```

Now the client is in the room

Namespaces

- Namespaces enable dedicated channels (e.g. like in Slack)
  - All users can access all channels
- On the client side it is the equivalent of multiple sockets

```
let chat= io.connect('/chat');
let news= io.connect('/news');


...


chat.on('joined', function (){
...
})


news.on(‘joined', function (){
...
})
```

A cheatsheet for Socket.io

- https://socket.io/docs/v3/emit-cheatsheet/

Questions

?