UNIVERSITÀ DI TORINO

Bidirectional client-server architecture with socket.io

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

UNIVERSITÀ DI TORINO

Sockets and Websockets

- 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
  - A web socket is defined by a URL and a port
    - a URL is an address, i.e. it represents a connected computer (like a street address, e.g. Fenton Road, Sheffield)
    - a port is an address number on that machine, (like a street number, e.g. 55 Fenton Road, Sheffield)

UNIVERSITÀ DI TORINO

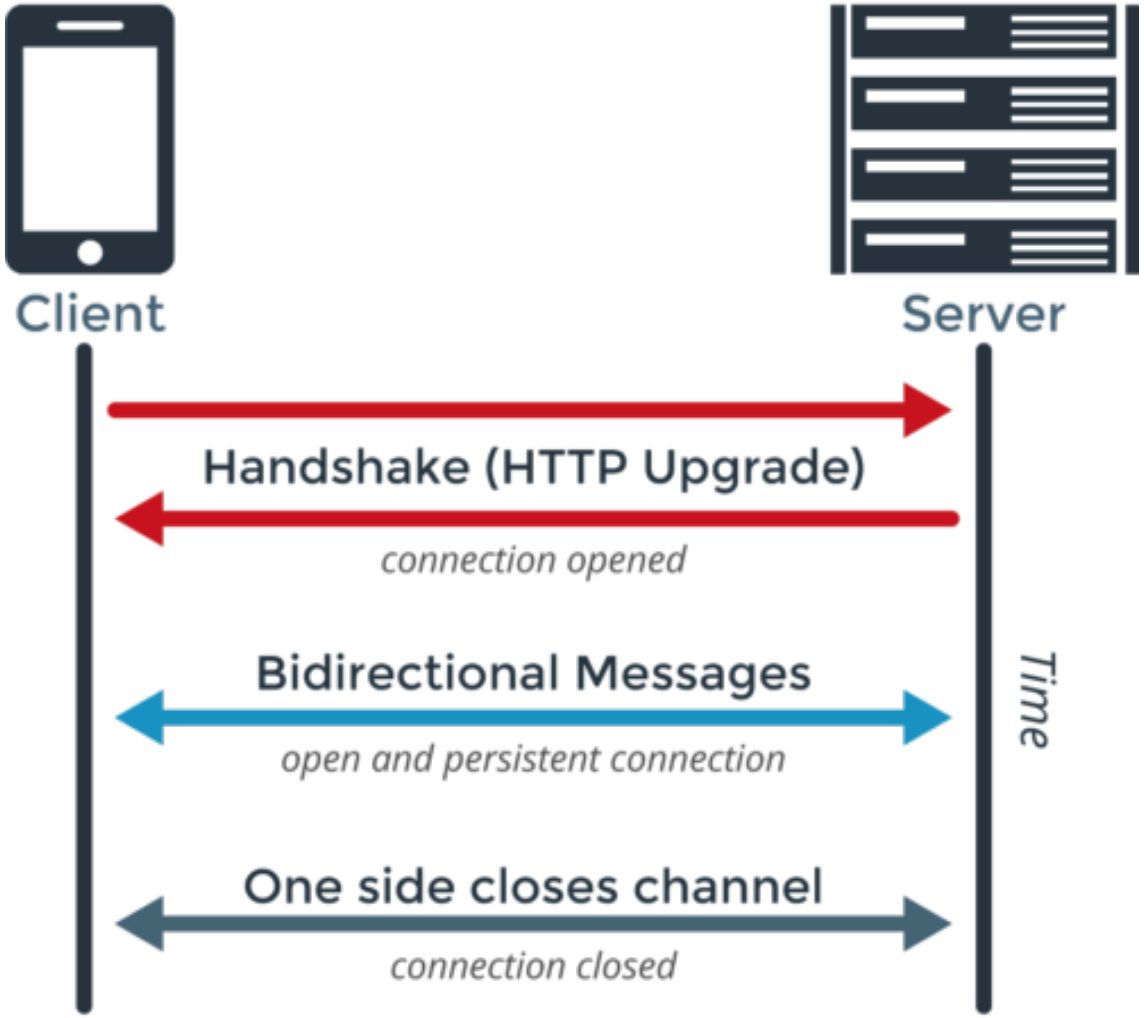
Difference between Streaming and Websockets

- HTTP streaming:
  - a variety of techniques (multipart/chunked response) that allow the server to send more than one response to a single client request
- WebSocket:
  - a transport layer built-on TCP that uses HTTP Upgrade. Unlike streaming, in WebSocket connections are bi-directional, full-duplex and long-lived. After an initial handshake request/ response, there is no transactional semantics
    - and there is very little per message overhead.
    - The client and server may send messages at any time and must handle message receipt asynchronously.

Operations

- Like in the standard http protocol you open and close the communication.
  - you also send and receive data
  - 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

WebSocket Communication



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

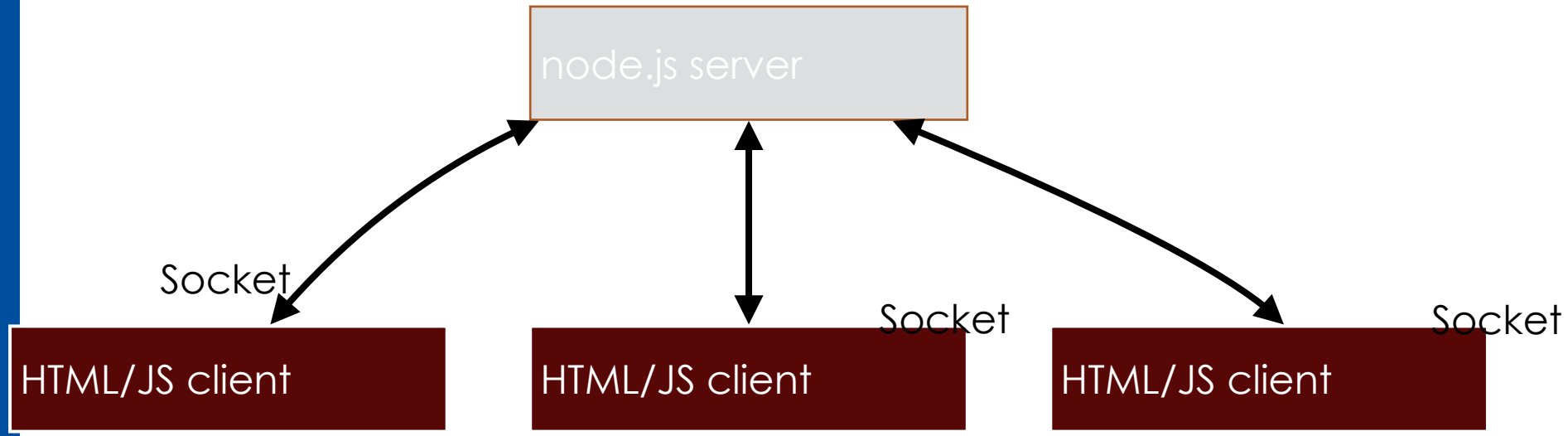
Socket.io

Socket.io

- Socket.IO enables real-time bidirectional event-based communication between browser and server
  - 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 for node.js.
  - Both components have a nearly identical API
- It primarily uses the WebSocket protocol
- It is possible to send any data,
  - Including blobs, i.e. Image, audio, video
- it is event based (on…)
- communication can be started by both client and server once connection is established and until it is closed from either sides

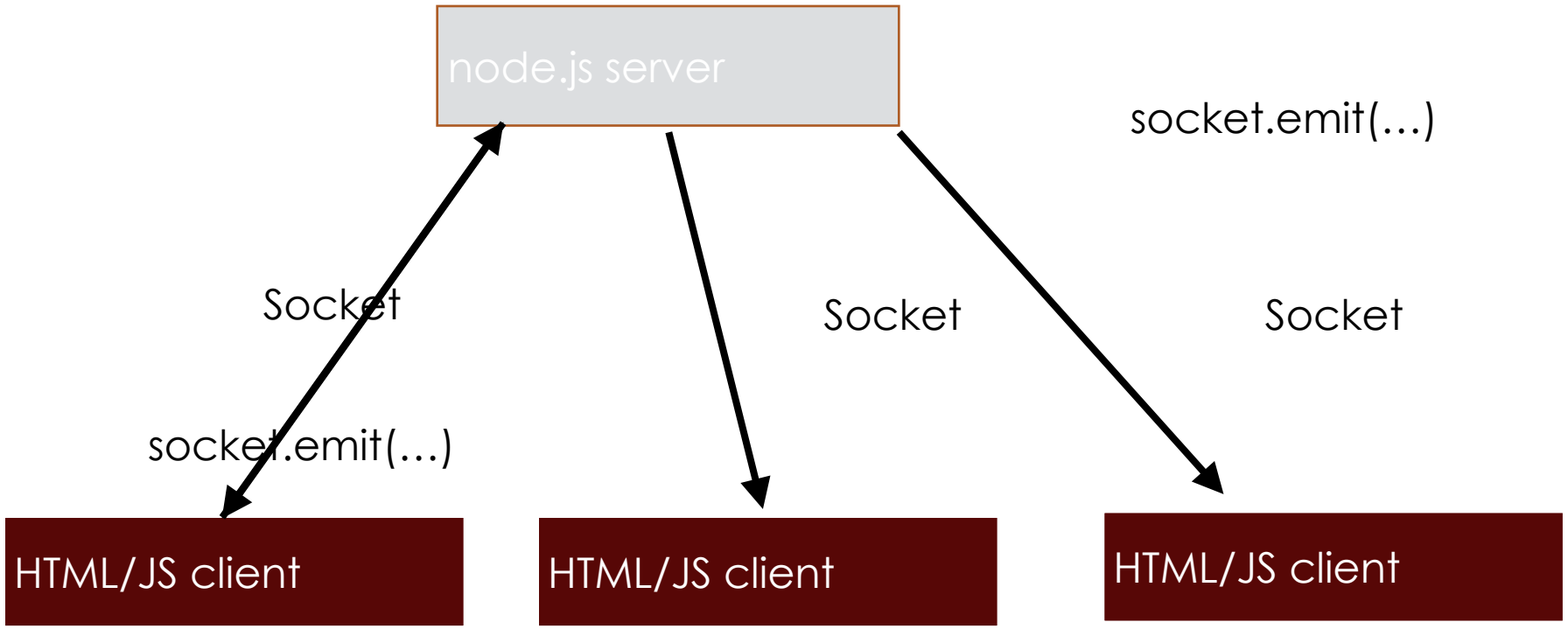
1 server, n clients, n sockets

- Socket is private channel shared by 1 client and 1 server
- However clients can communicate via the server



1 server, n clients, n sockets

- Communication happens via the command socket.emit(…) on both sides



Client Server communication in express

Express server routes/index.js

```
router.get('/', function(req, res, next) {
  res.render('index', { title: 'My Chat' });
```

HTML/JS client

```
<script src="/socket.io/socket.io.js">
</script>
…
<script>
  var socket = io();
…
```

client opens the socket and connects

client emits a message

```
socket.emit ('message', param)
```

```
…
socket.on ('message', function (param){…});
```

client receives a message

```
io.on('connection', function(socket){
  socket.on ('message',
        function (param){
                …
                socket.emit ('message' param)
        …});
});
```

server receives a message

callback

server emits a message

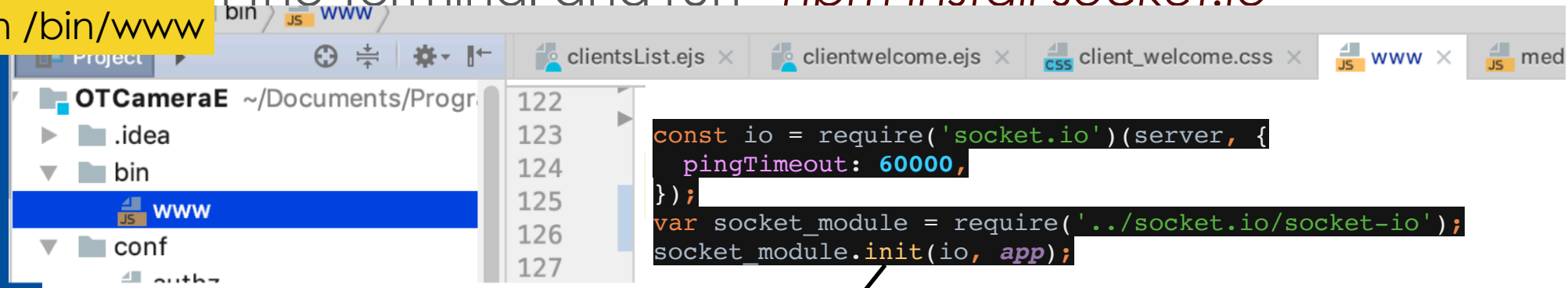
in /socket.io/socket.io.js

socket automatically closes when client navigates away from page

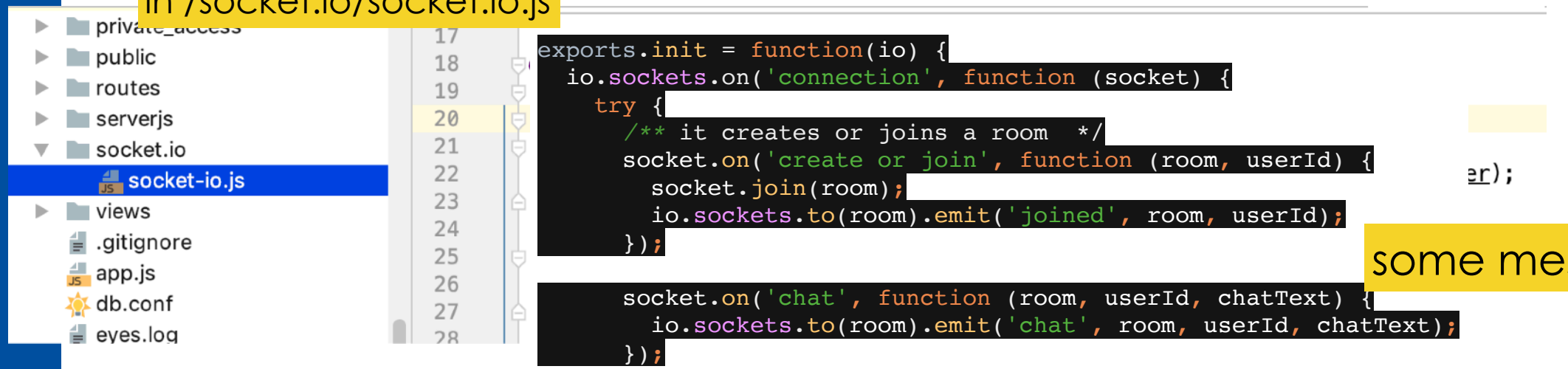
In WebStorm

- open the Terminal and run *npm install socket.io*

in /bin/www



in /socket.io/socket.io.js



some messages

UNIVERSITÀ DI TORINO

- socket.io enables sending and receiving messages

```
exports.init = function(io) {
  io.sockets.on('connection', function (socket) {
    try {
      /**
       * it creates or joins a room
       */
      socket.on('create or join', function (room, userId) {
        socket.join(room);
        io.sockets.to(room).emit('joined', room, userId);
      });

      socket.on('chat', function (room, userId, chatText) {
        io.sockets.to(room).emit('chat', room, userId, chatText);
      });
```

UNIVERSITÀ DI TORINO

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Socket.io Chat</title>
</head>
<body>
  <ul id="messages"></ul>
  <input id="input" autocomplete="off" />
  <button id="send">Send</button>
</body>
</html>
```

UNIVERSITÀ DI TORINO

In HTML

in a Javascript script associated to the client

```
/* it opens the connection */
let socket = io();
...
/* it declares the expected messages and associated actions */
socket.on('joined', function (userId){
      const messageElement = document.createElement('li');
      messageElement.textContent = userId+' has joined';
      messages.appendChild(messageElement);
    });
})
```

```
 socket.on('chat message', (userId, chatText) => {
      const messageElement = document.createElement('li');
      messageElement.textContent = msg;
      messages.appendChild(messageElement);
    });
```

UNIVERSITÀ DI TORINO

Broadcasting

- Broadcasting means sending a message to everyone else
  - except for the socket that starts it

Broadcasting messages

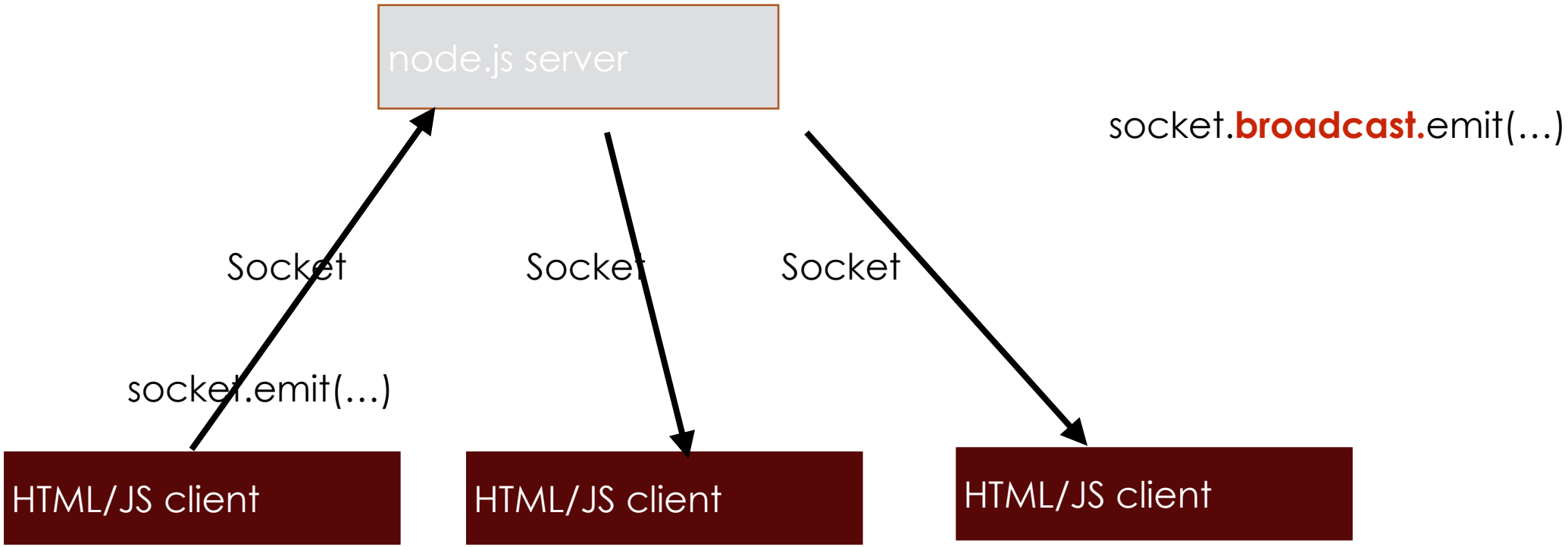
To broadcast, simply add a `broadcast` flag to `emit` and `send` method calls. Broadcasting means sending a message to everyone else except for the socket that starts it.

Server

```
var io = require('socket.io')(80);

io.on('connection', function (socket) {
  socket.broadcast.emit('user connected');
});
```

socket.broadcast.emit

- Communication is not returned to the originating client



UNIVERSITÀ DI TORINO

Rooms

Within each namespace, you can also define arbitrary channels that sockets can `join` and `leave`.

Joining and leaving

You can call `join` to subscribe the socket to a given channel:

```
io.on('connection', function(socket){
  socket.join('some room');
});
```

And then simply use `to` or `in` (they are the same) when broadcasting or emitting:

```
io.to('some room').emit('some event'):
```

To leave a channel you call `leave` in the same fashion as `join`.

This is on the server side
The client can be in just one room at a time

Default room

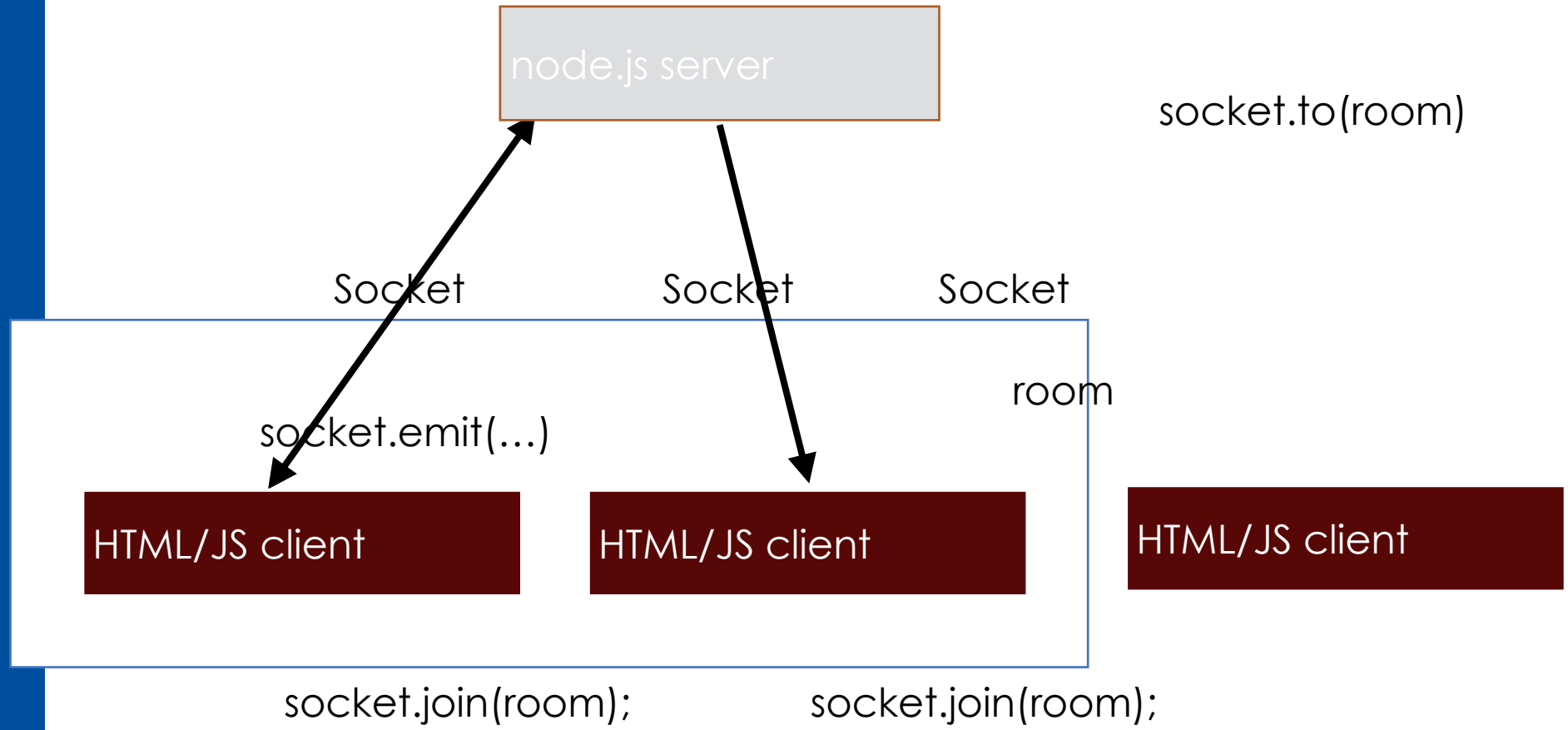
Each `Socket` in Socket.IO is identified by a random, unguessable, unique identifier `Socket#id`. For your convenience, each socket automatically joins a room identified by this id.

This makes it easy to broadcast messages to other sockets:

```
io.on('connection', function(socket){
  socket.on('say to someone', function(id, msg){
    socket.broadcast.to(id).emit('my message', msg);
  });
});
```

1 server, n clients, n sockets

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



Connecting to room

- Client side:

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

- Server side (in /socket.io/socket-io.js)

```
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

We have two namespaces here: /chat and /news

```
exports.init = function(io) {

  // the chat namespace
  const chat= io
      .of('/chat')
      .on('connection', function (socket) {
    try {
          socket.on('create or join', function (room, userId) {
...

// the news namespace
const news= io
      .of('/news')
      .on('connection', function (socket) {
    try {
          socket.on('create or join', function (room, userId) {
```

in /socket.io/socket-io.js

Namespaces

- On the client side we have the equivalent of multiple sockets

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

...

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

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

Disconnection

- e.g. when client moves away from page

```
io.on('connection', function(socket){
  console.log('a user connected');
…
  socket.on('disconnect', function(){
    console.log('user disconnected');
  });
…
});
```

socket.io callbacks

Sending and getting data (acknowledgements)

Sometimes, you might want to get a callback when the client confirmed the message reception.

To do this, simply pass a function as the last parameter of `.send` or `.emit`. What's more, when you use `.emit`, the acknowledgement is done by you, which means you can also pass data along:

Server (app.js)

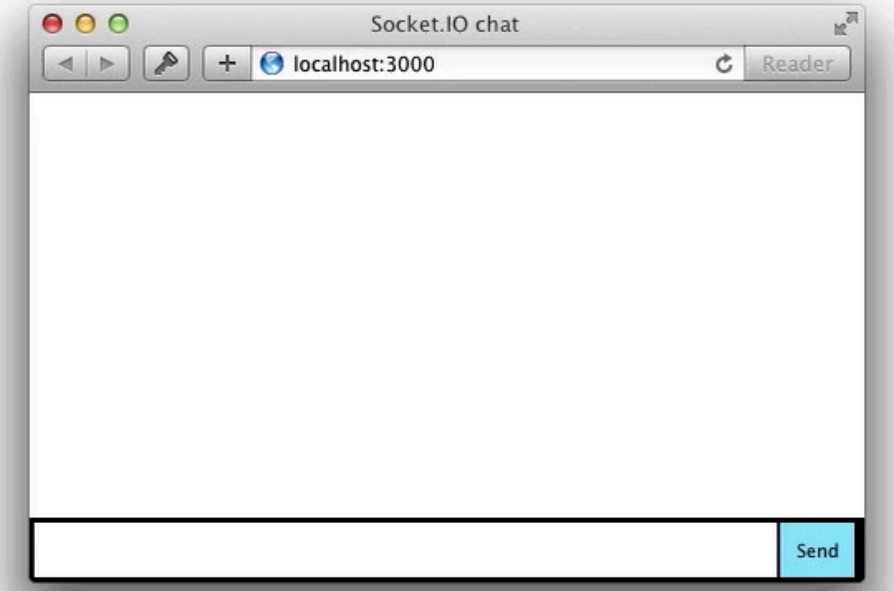
```
var io = require('socket.io')(80);

io.on('connection', function (socket) {
  socket.on('ferret', function (name, fn) {
    fn('woot');
  });
});
```

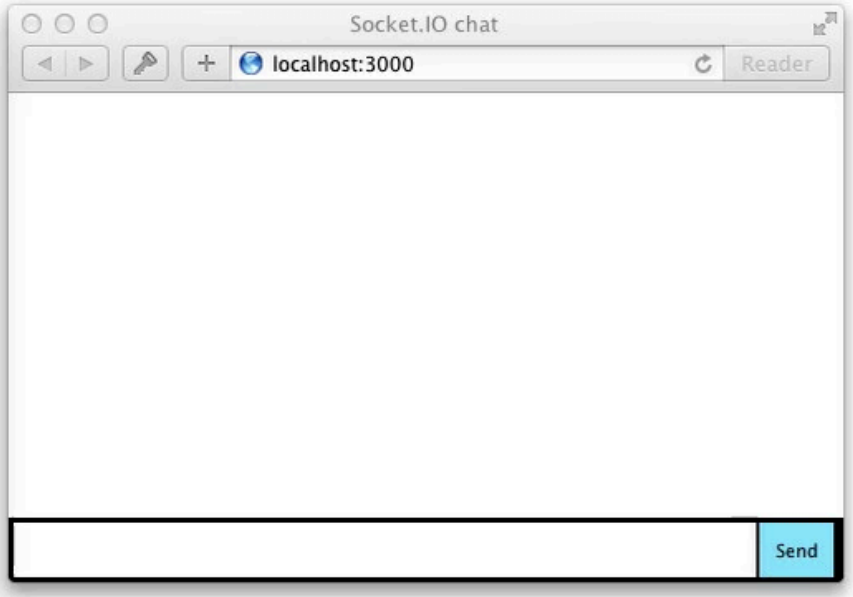
Client (index.html)

```
<script>
  var socket = io(); // TIP: io() with no ar
gs does auto-discovery
  socket.on('connect', function () { // TIP:
 you can avoid listening on `connect` and li
sten on events directly too!
    socket.emit('ferret', 'tobi', function (
data) {
      console.log(data); // data will be 'wo
ot'
    });
  });
</script>
```

DO NOT return a private message via socket.emit - it would be a public message!!!!

https://socket.io/get-started/chat/





Instant messaging and chat

Whatsapp or Skype - like

UNIVERSITÀ DI TORINO

Goal

- Creating an instant messaging and chat system
- Design:
  - Node.js/Express serves a file index.html
  - Index.html opens a socket and joins a room
    - the client tells the server it is joining a room
    - the server opens the room if not existing and joins the client to it
    - the server tells everybody in the room the client has joined
  - every time the user writes and sends a message
    - the client sends the text to the server
    - the client writes on its own message panel
    - the server broadcasts it to everybody else
    - the other clients in the room write the message onto their message panels

sending a message

node.js server

```
io.on('connection', function(socket){
  socket.on ('joining',
          function (userId, roomId){
            socket.join(room);
            socket.broadcast.to(room).emit
                    ('updatechat',
                    socket.username +
                    ' has joined this room', '');});
        socket.on('sendchat', function (data) {
            io.sockets.in(socket.room).emit
                    ('updatechat', socket.username,
                    data);
         });});
```

io.sockets.in (or io.sockets.to)
broadcasts to all sockets in the room including the calling one

HTML/JS client 1

```
socket.on ('updatechat',
          function (message){
            …write on message panel
          });
            socket.emit('sendchat', message)
```

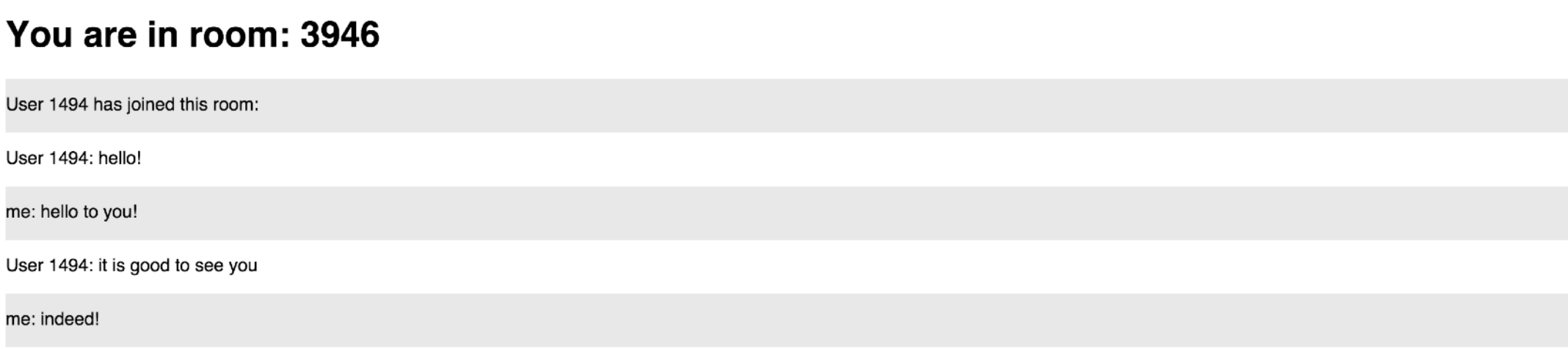
HTML/JS client 2

```
<script src="/socket.io/socket.io.js">
</script>
…
<script>
  var socket = io();
  socket.on ('updatechat',
          function (message){
            …write on message panel
          });
```

UNIVERSITÀ DI TORINO

The rest is just a form!

- We will see how to build a complete system in the lab



UNIVERSITÀ DI TORINO

What you should know

- how to create bidirectional client/server architectures using socket.io
- how to create the connection
- how to create a room and have different clients in different rooms
- the difference between sending messages and broadcasting
- how to build a simple chat system

UNIVERSITÀ DI TORINO

Questions?

© Prof. Fabio Cravegna, Università di Torino